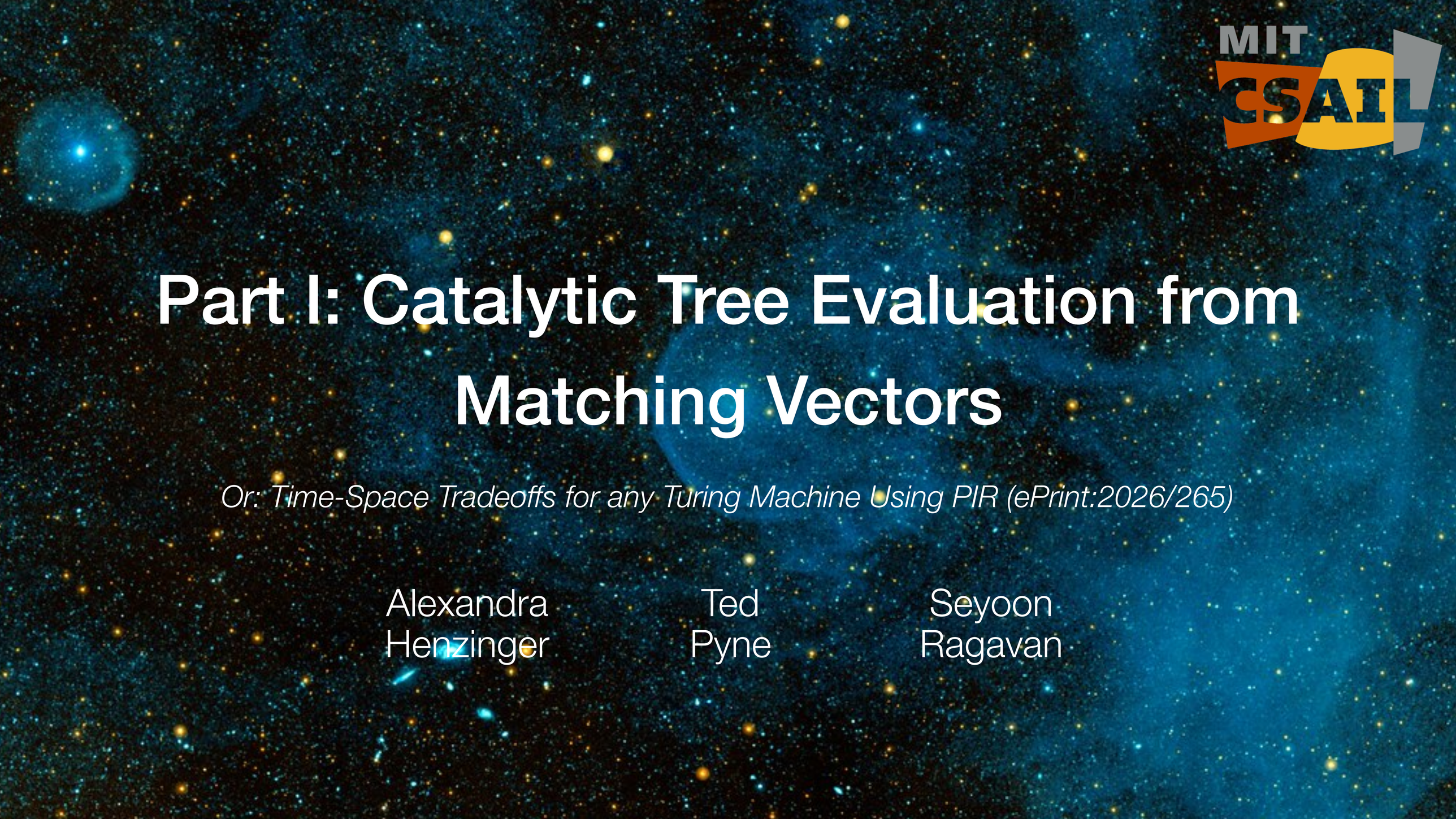
PIR for Time-Space Tradeoffs & Time-Space Tradeoffs for PIR

Alexandra Henzinger

Ted Pyne

Seyoon Ragavan

Thanks Alexandra for some of these slides (and the title)!



Part I: Catalytic Tree Evaluation from Matching Vectors

Or: Time-Space Tradeoffs for any Turing Machine Using PIR (ePrint:2026/265)

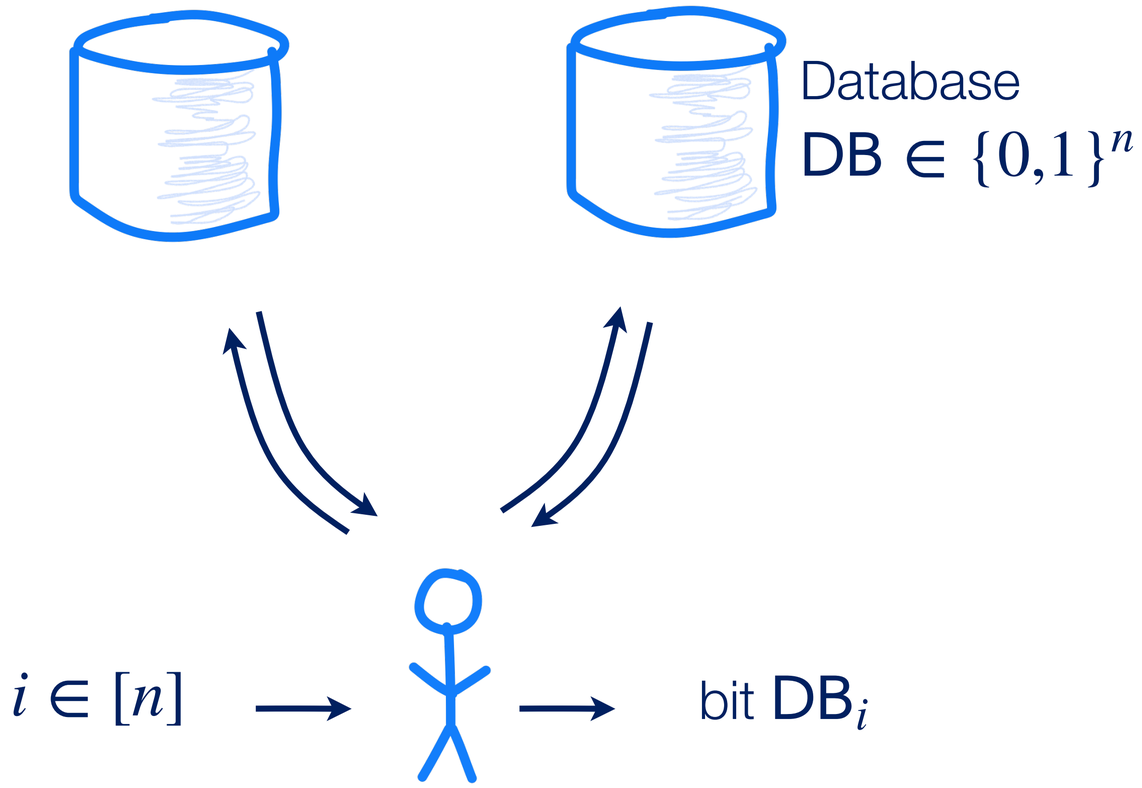
Alexandra Henzinger

Ted Pyne

Seyoon Ragavan

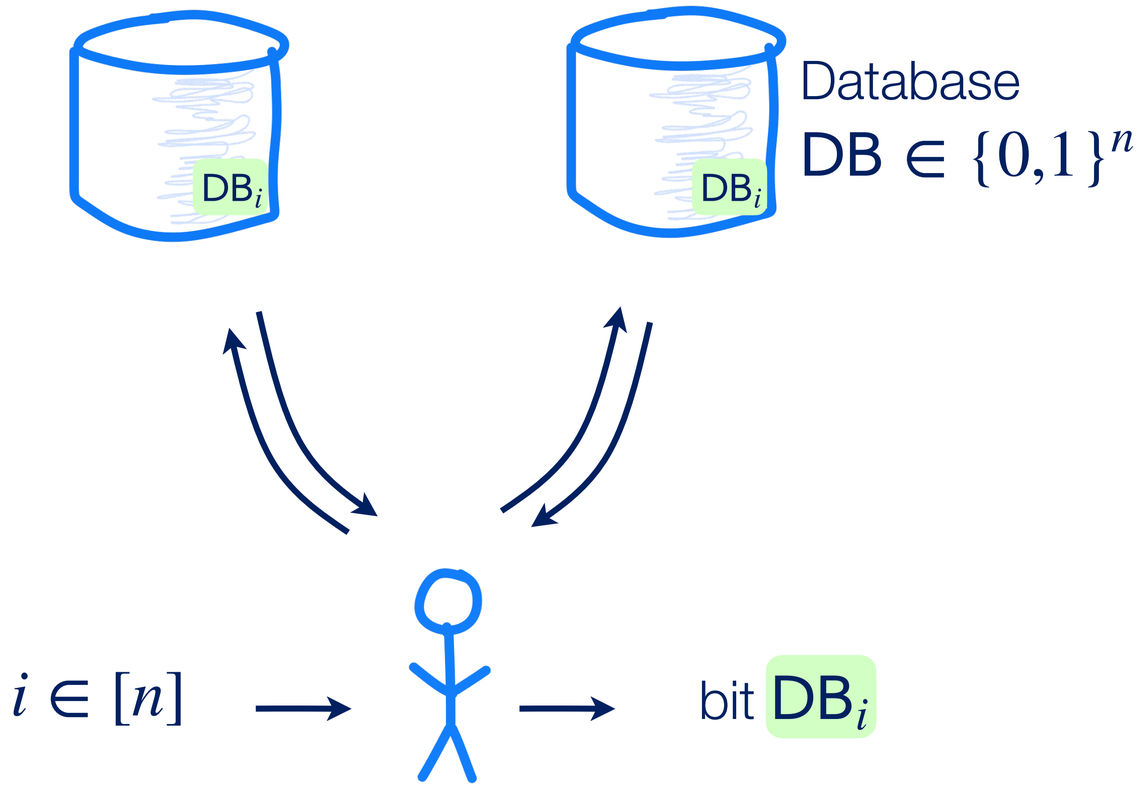
Private Information Retrieval [CGKS95,KO97]

Goal: privately read an entry from a remote database



Private Information Retrieval [CGKS95,KO97]

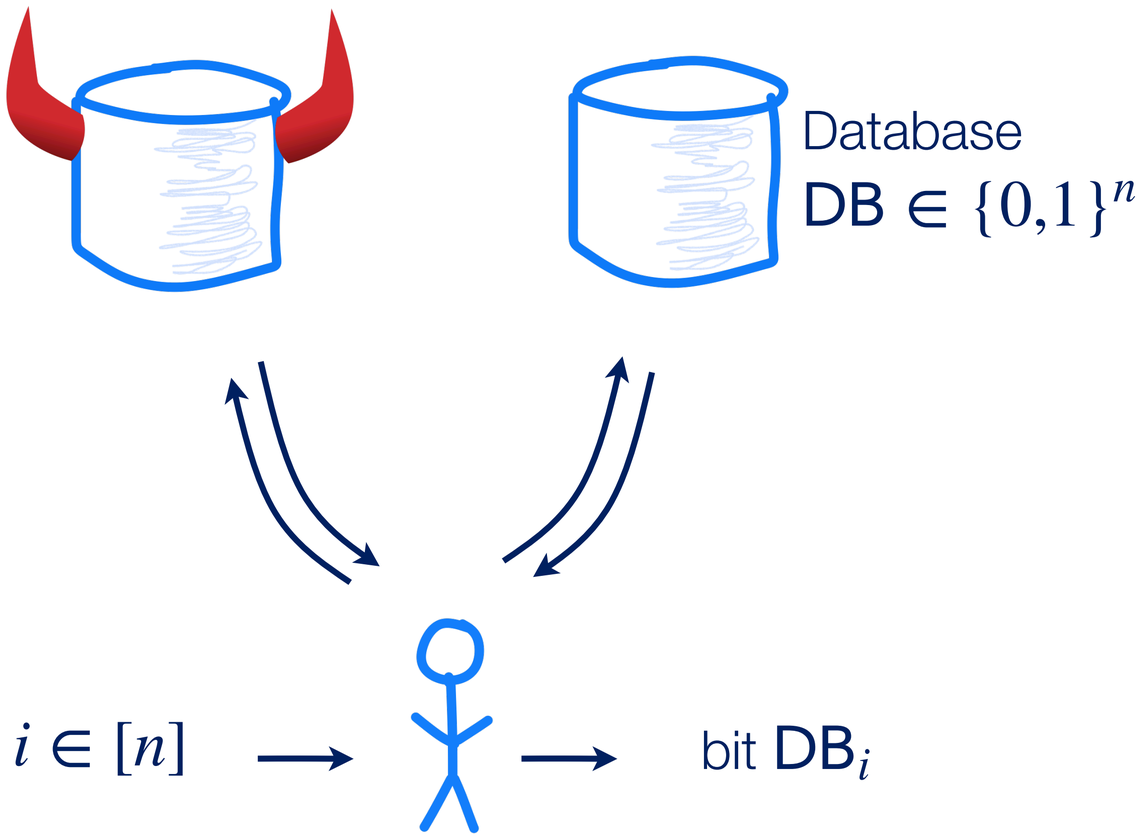
Goal: privately read an entry from a remote database



Correctness:
for all DB $\in \{0,1\}^n$ and $i \in [n]$,
a user interacting with two honest servers learns DB$_i$.

Private Information Retrieval [CGKS95,KO97]

Goal: privately read an entry from a remote database



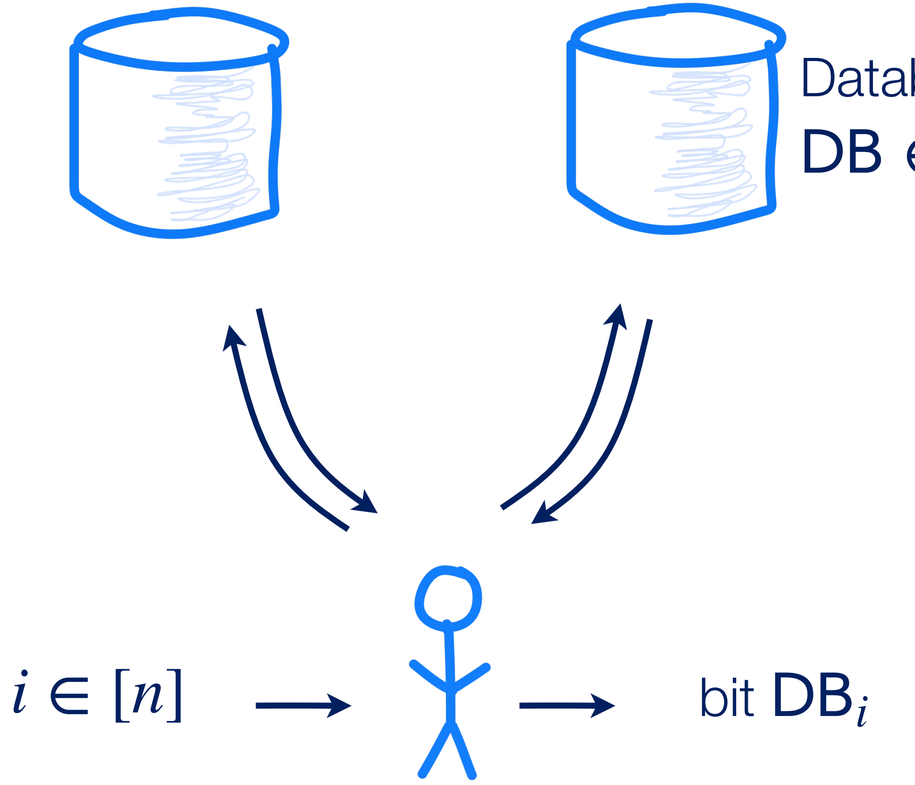
Correctness:
for all $\mathsf{DB} \in \{0,1\}^n$ and $i \in [n]$,
a user interacting with two honest servers learns DB_i.

Privacy:
an attacker compromising one server learns nothing about i,
even if malicious.

Formally: information-theoretic/computational indistinguishability of the server's view, on reads to any two indices i or i'.

Private Information Retrieval [CGKS95,KO97]

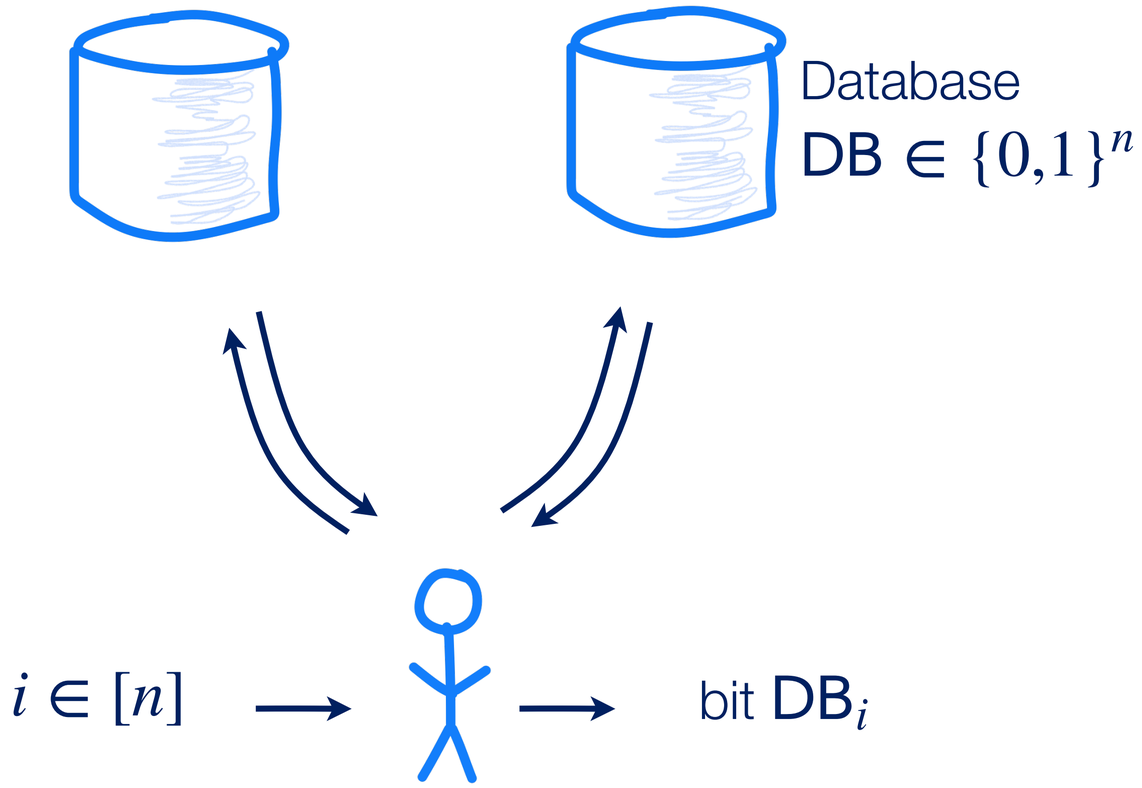
Goal: privately read an entry from a remote database



- Private web search [Coeus'21, Tiptoe'23, Wally'24]
- Private advertising [J'01,ObliviAd'12,AdVeil'20…]
- Private media delivery [Popcorn'16,PIRSONA'21,CrowdSurf'25,…]
- Metadata-hiding messaging [PIRTor'11,Pung'16,Talek'20,Addra'21,…]
- Private auditing and blocklists [Checklist'20,SimplePIR'23,HPW'25,…]
- Private file storage [Dory'20,Peer2PIR'25,…]
- **On your iPhone:** Private caller-ID

Private Information Retrieval [CGKS95,KO97]

Goal: privately read an entry from a remote database

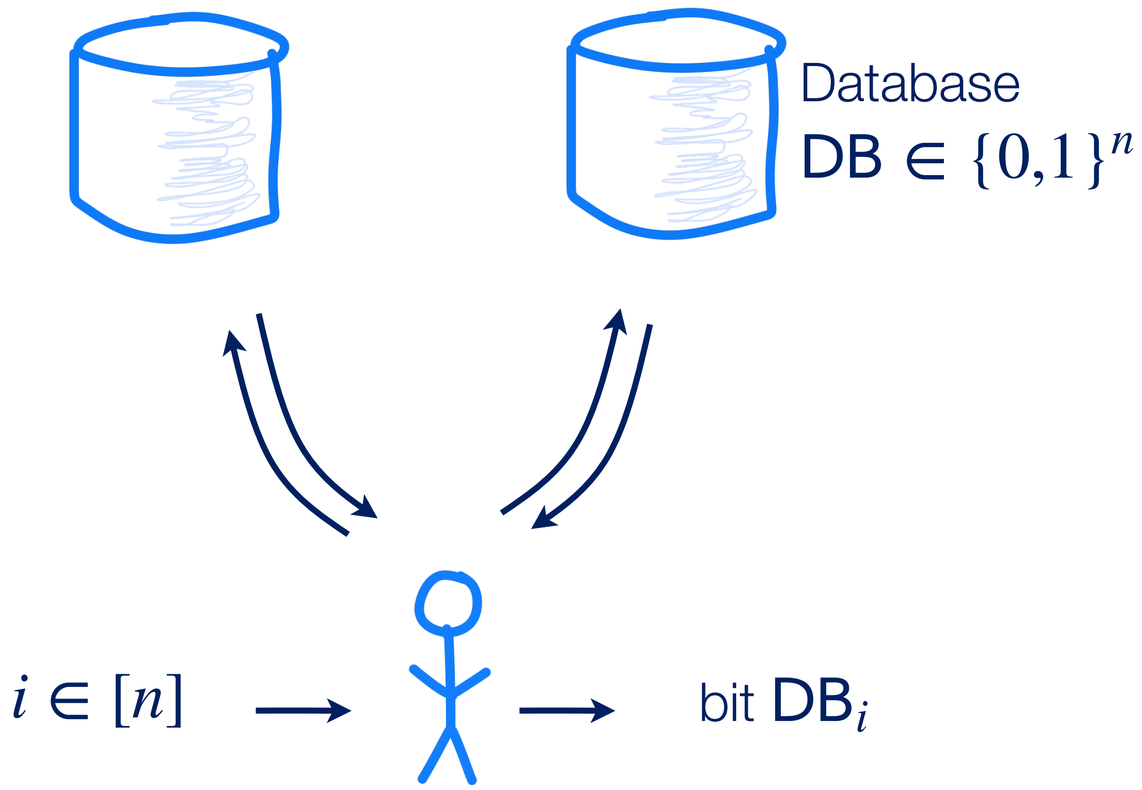


We have PIR constructions with very little communication:

- No privacy: $\log n + 1$

Private Information Retrieval [CGKS95,KO97]

Goal: privately read an entry from a remote database

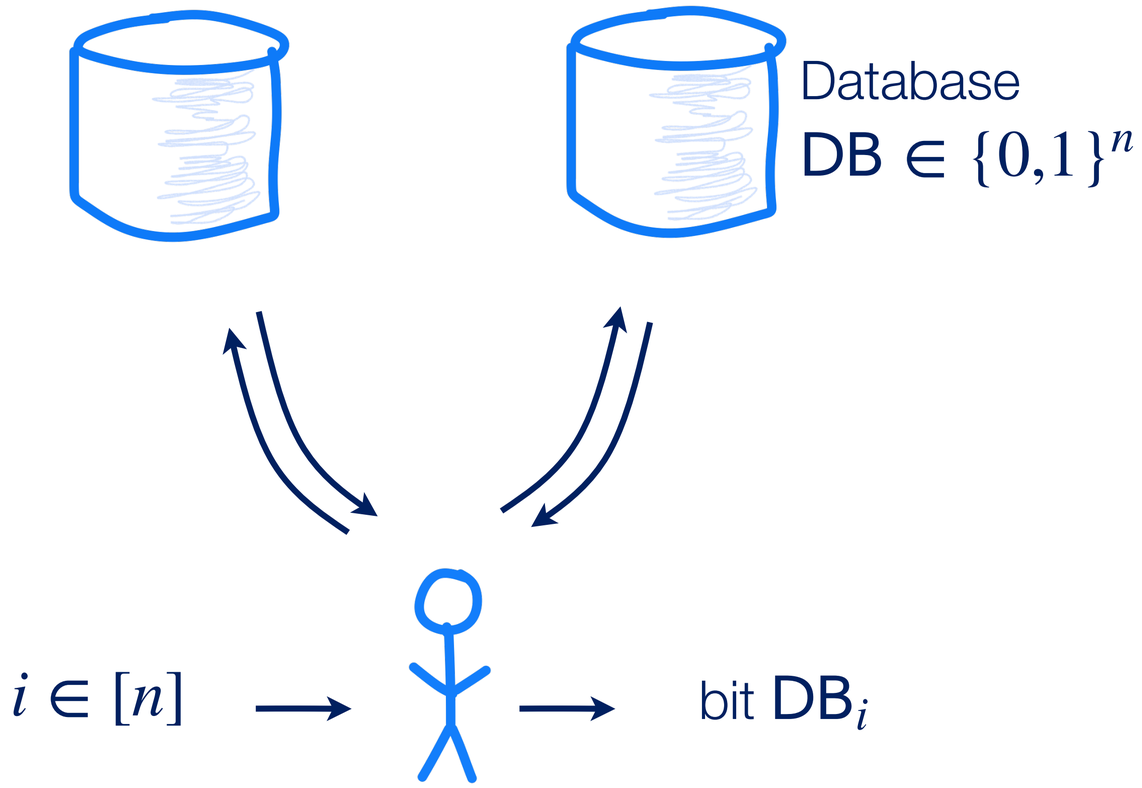


We have PIR constructions with very little communication:

- No privacy: $\log n + 1$
- Info-theoretic privacy: $n^{o(1)}$

Private Information Retrieval [CGKS95,KO97]

Goal: privately read an entry from a remote database

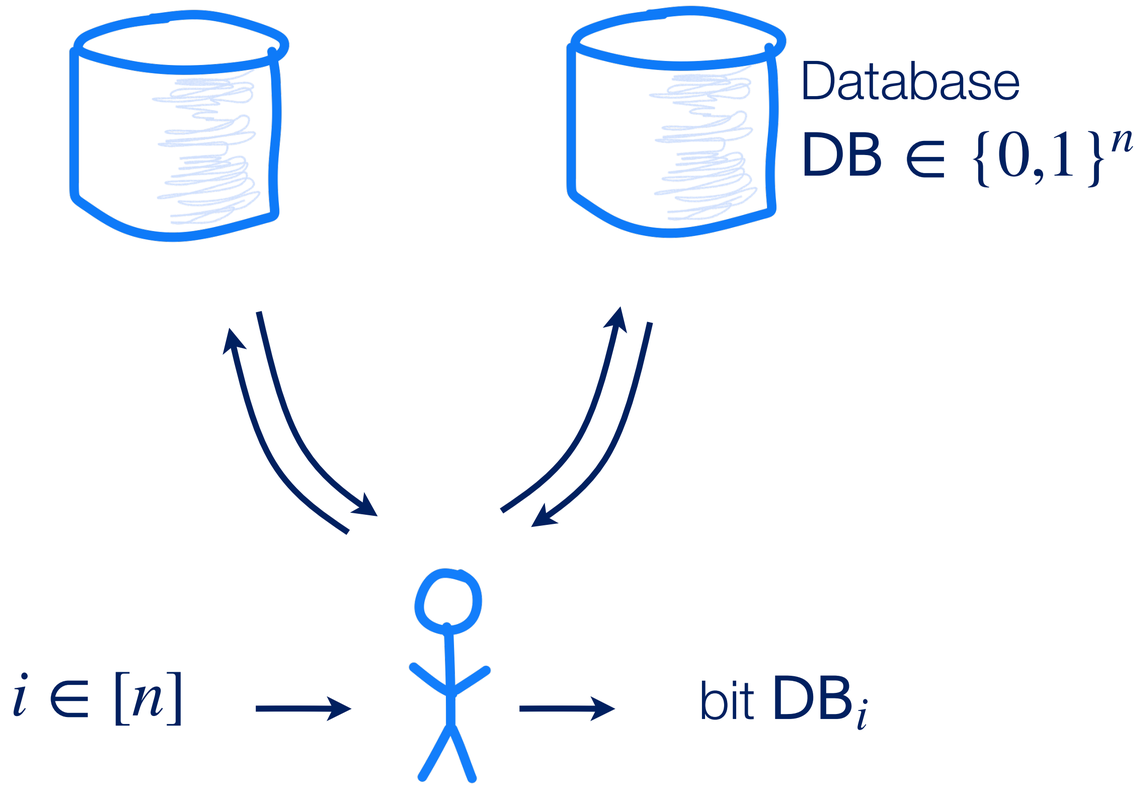


We have PIR constructions with very little communication:

- No privacy: $\log n + 1$
- Info-theoretic privacy: $n^{o(1)}$
- Comp. privacy: $O(\lambda \cdot \log n)$
  [KO97,CMS99,DG16,BGI16]

Private Information Retrieval [CGKS95,KO97]

Goal: privately read an entry from a remote database



We have PIR constructions with very little communication:

- No privacy: $\log n + 1$
- **Info-theoretic privacy:** $n^{o(1)}$
- Comp. privacy: $O(\lambda \cdot \log n)$
  [KO97,CMS99,DG16,BGI16]

This talk: leveraging techniques from low-communication, information-theoretic PIR for complexity theory

Roadmap

- Tree evaluation: background and results

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR ↔ Tree Evaluation

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR $\leftrightarrow$ Tree Evaluation
- Formal abstraction: matching vectors and decoding polynomials

Roadmap

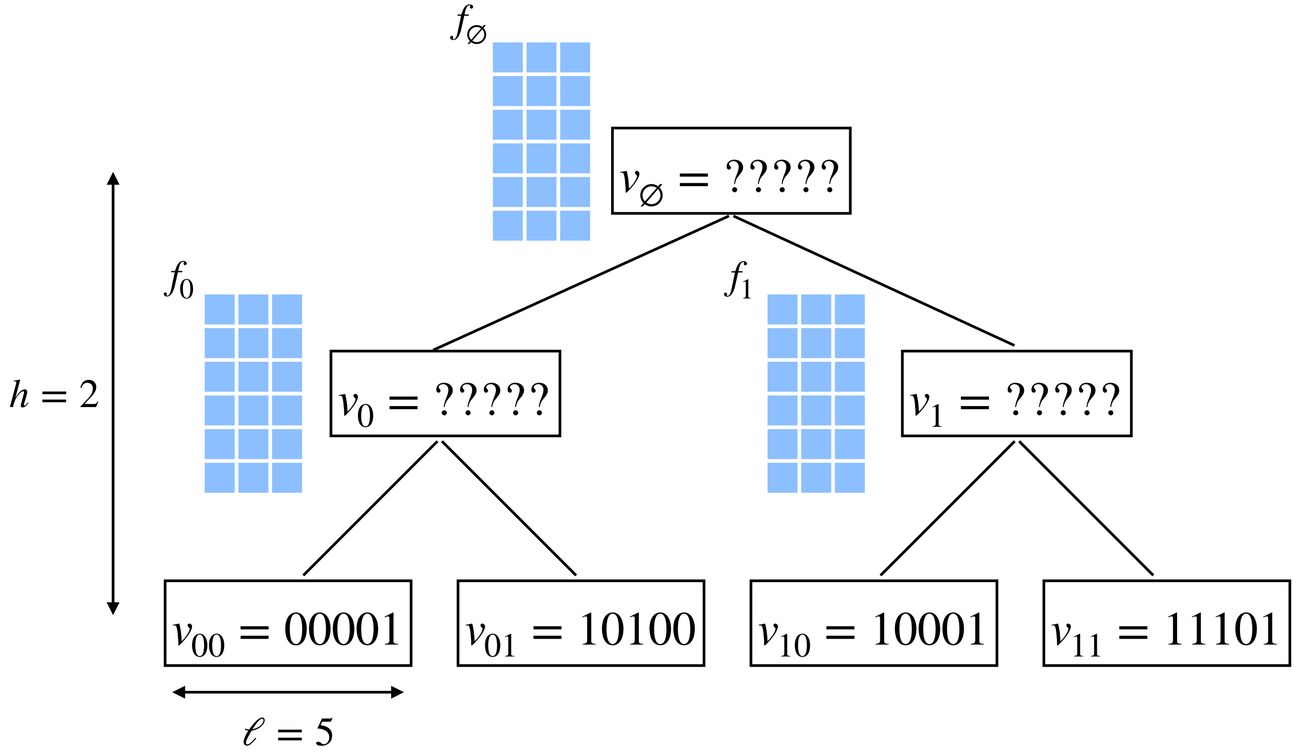
- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR $\leftrightarrow$ Tree Evaluation
- Formal abstraction: matching vectors and decoding polynomials
- Conclusion

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR ↔ Tree Evaluation
- Formal abstraction: matching vectors and decoding polynomials
- Conclusion

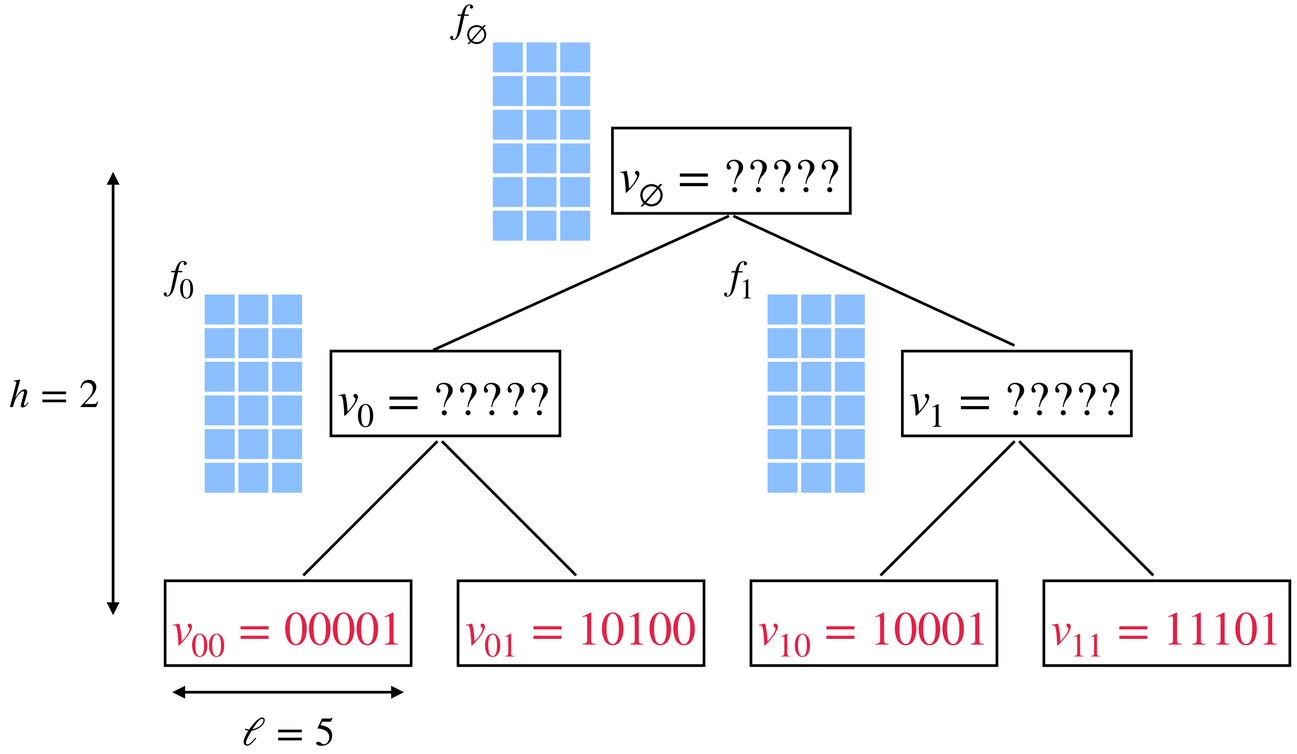
The Tree Evaluation Problem

- Complete binary tree of height h



The Tree Evaluation Problem

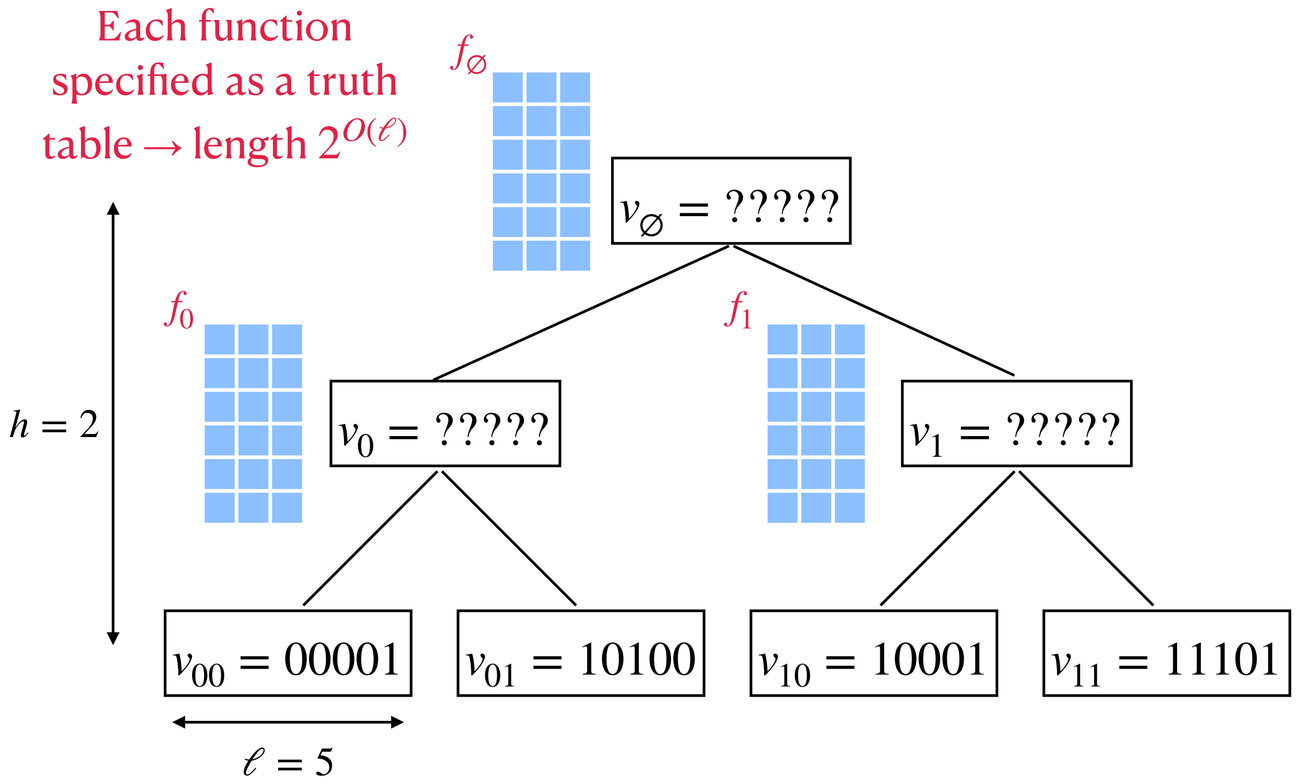
- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf



The Tree Evaluation Problem

- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

$$f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$$



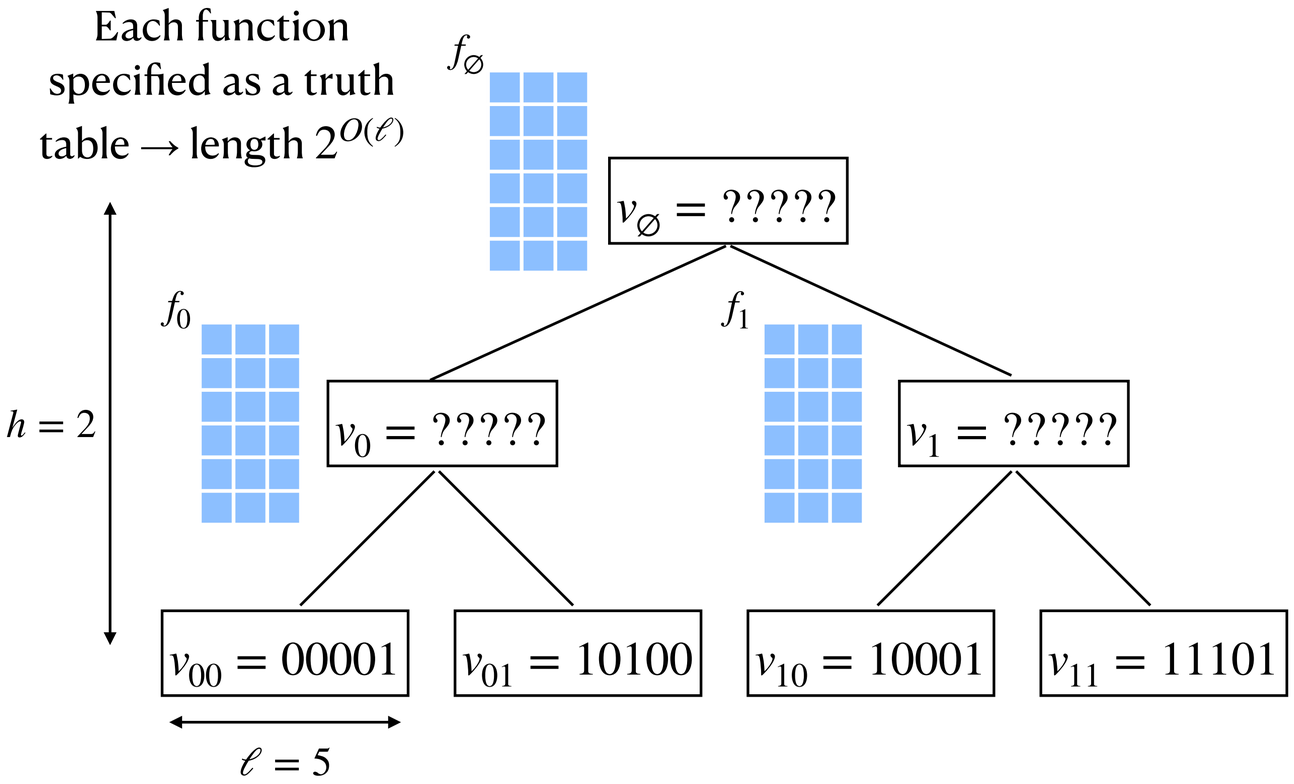
The Tree Evaluation Problem

- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

$$f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$$

- Goal: propagate the leaf values up the tree

$$v_u = f_u(v_{u0}, v_{u1})$$



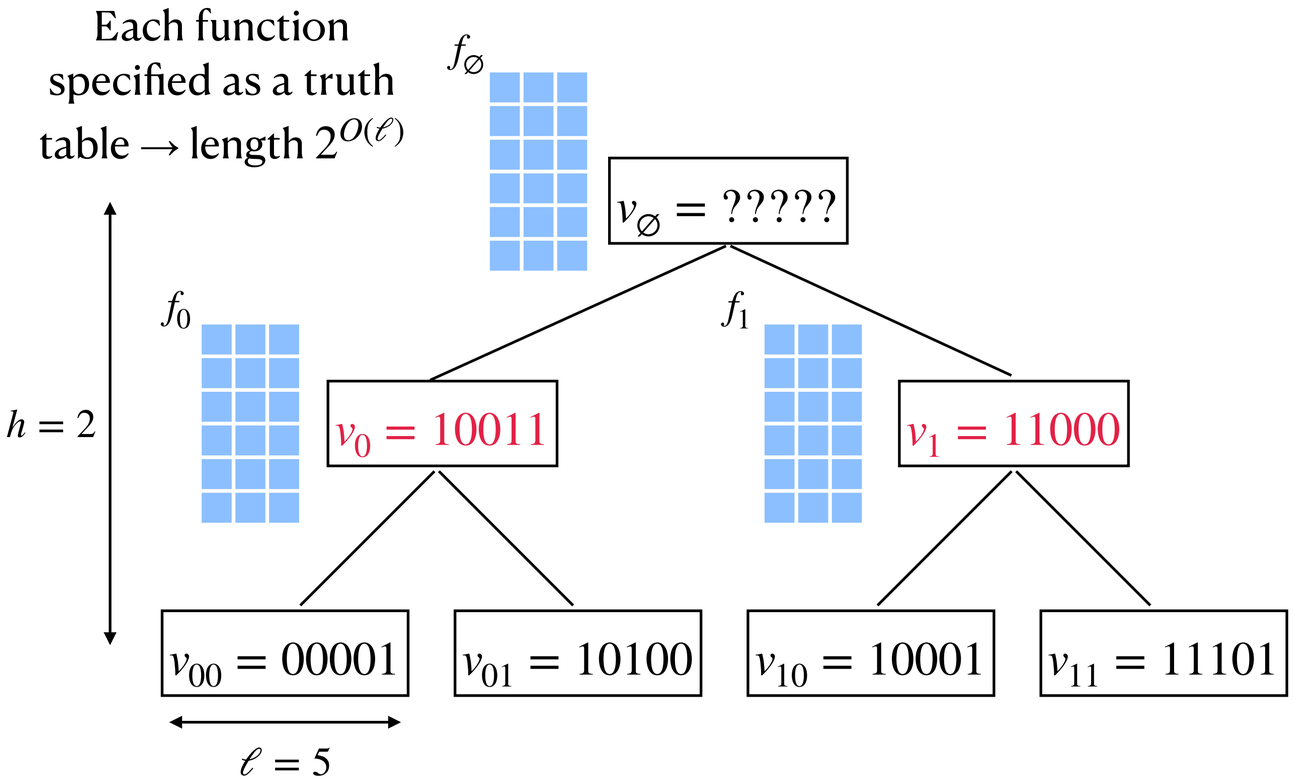
The Tree Evaluation Problem

- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

$$f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$$

- Goal: propagate the leaf values up the tree

$$v_u = f_u(v_{u0}, v_{u1})$$



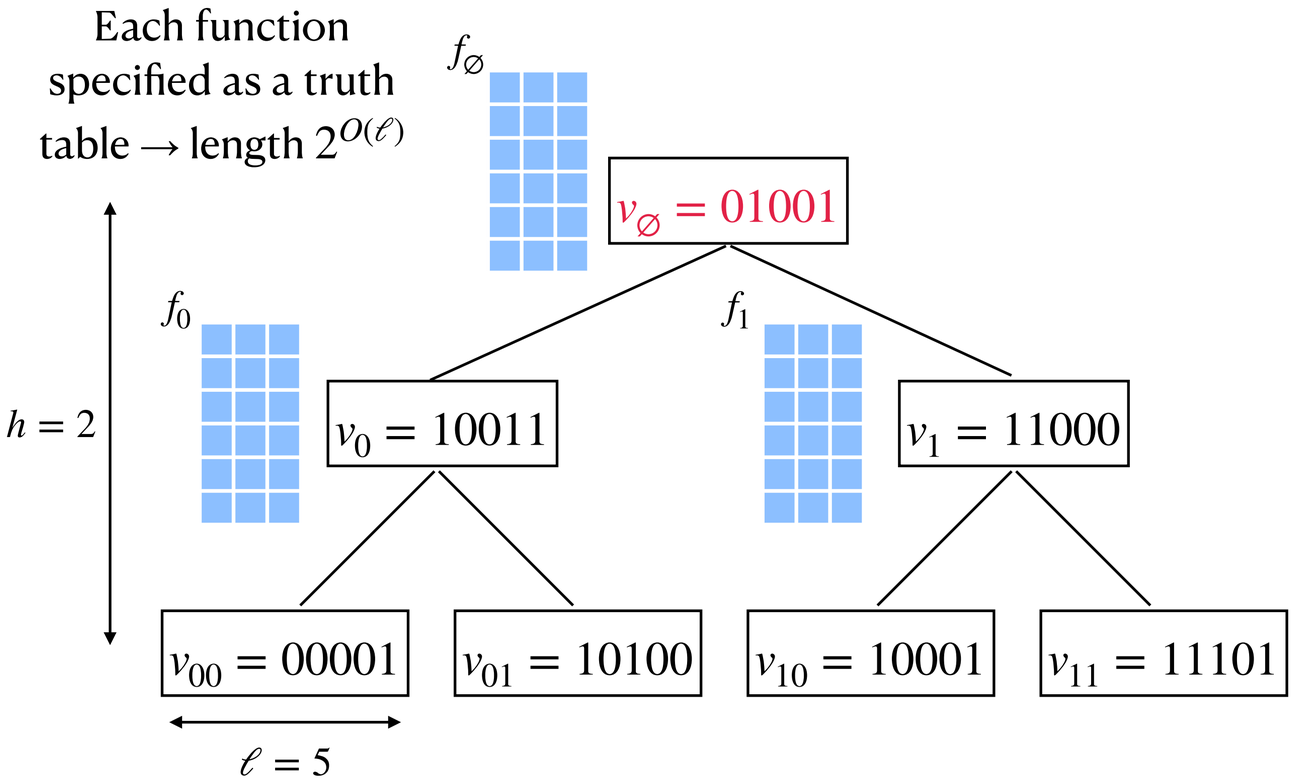
The Tree Evaluation Problem

- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

$$f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$$

- Goal: propagate the leaf values up the tree

$$v_u = f_u(v_{u0}, v_{u1})$$



The Tree Evaluation Problem

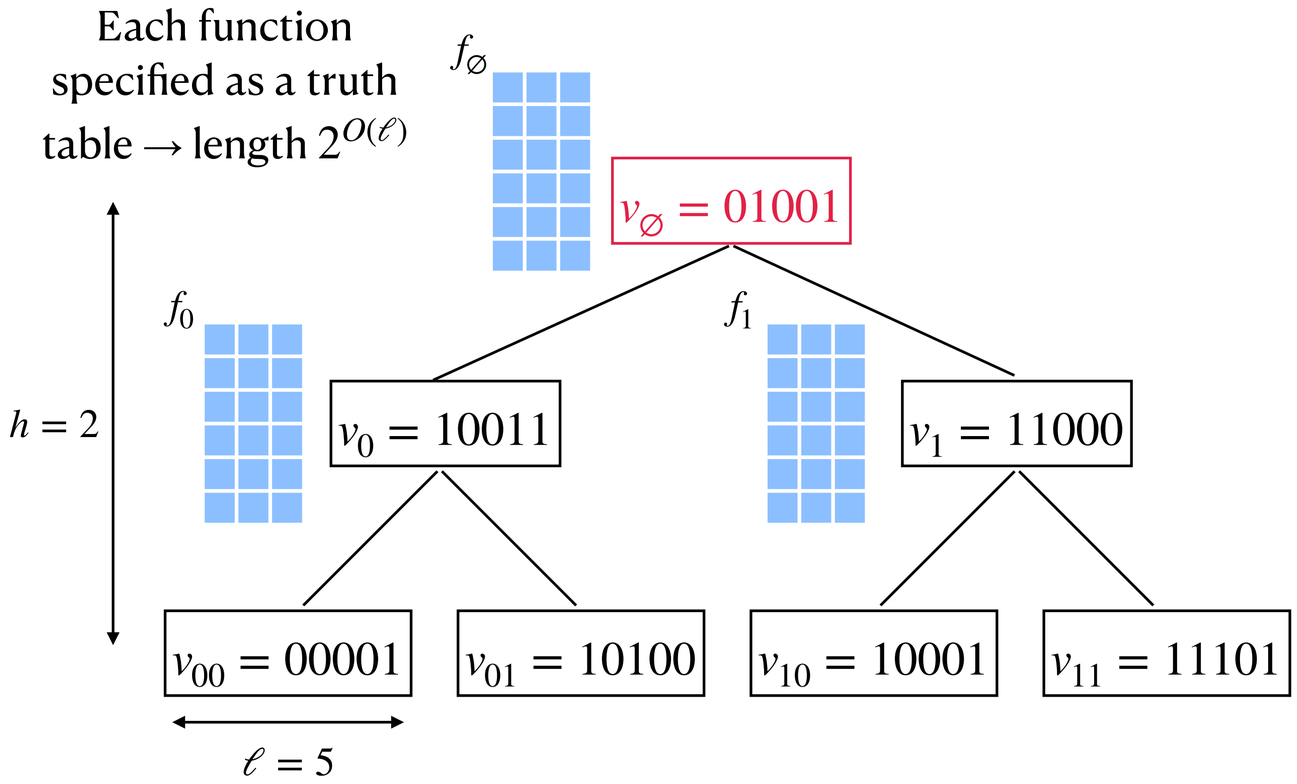
- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

$$f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$$

- Goal: propagate the leaf values up the tree

$$v_u = f_u(v_{u0}, v_{u1})$$

- Output: the root value



The Tree Evaluation Problem

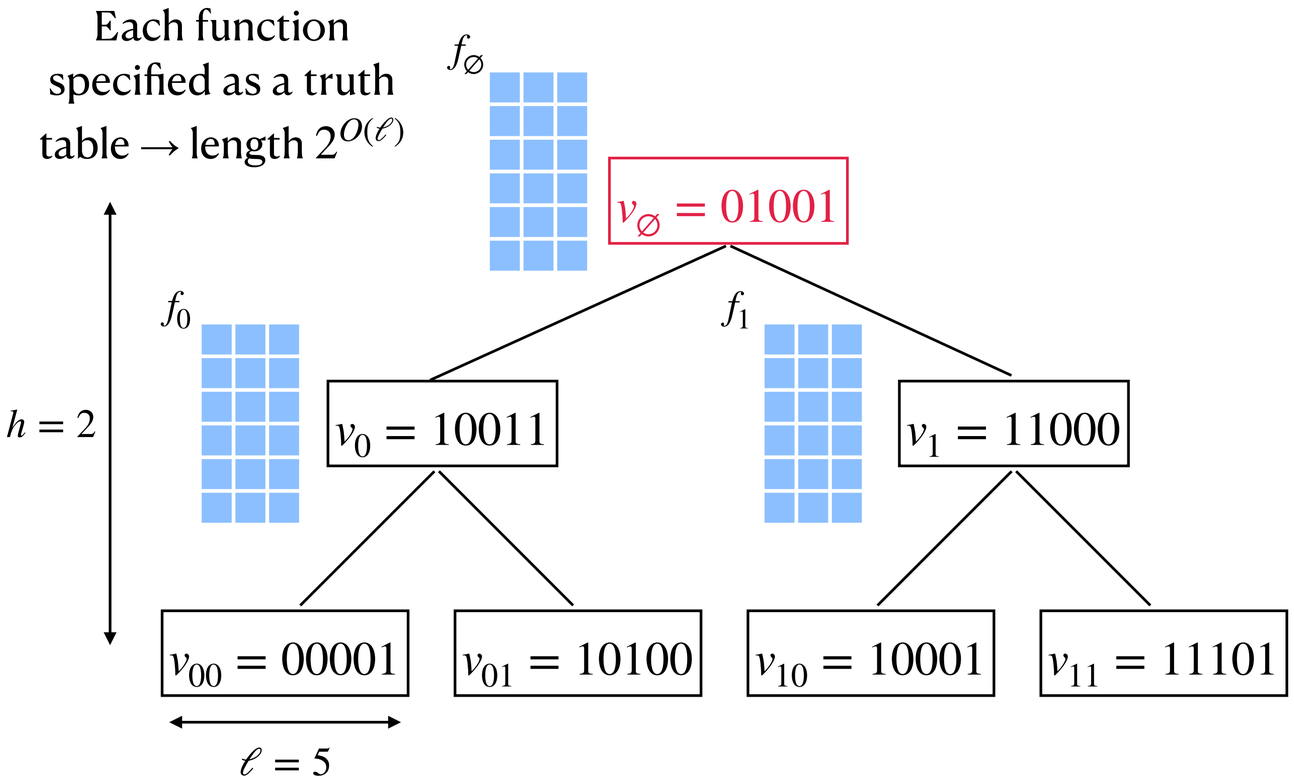
- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

$$f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$$

- Goal: propagate the leaf values up the tree

$$v_u = f_u(v_{u0}, v_{u1})$$

- Output: the root value



$$n = \text{total input length} = \text{poly}(2^{h+\ell})$$

The Tree Evaluation Problem

- Complete binary tree of height h
- Inputs:
  - A ℓ-bit string at each leaf
  - Function at each internal node

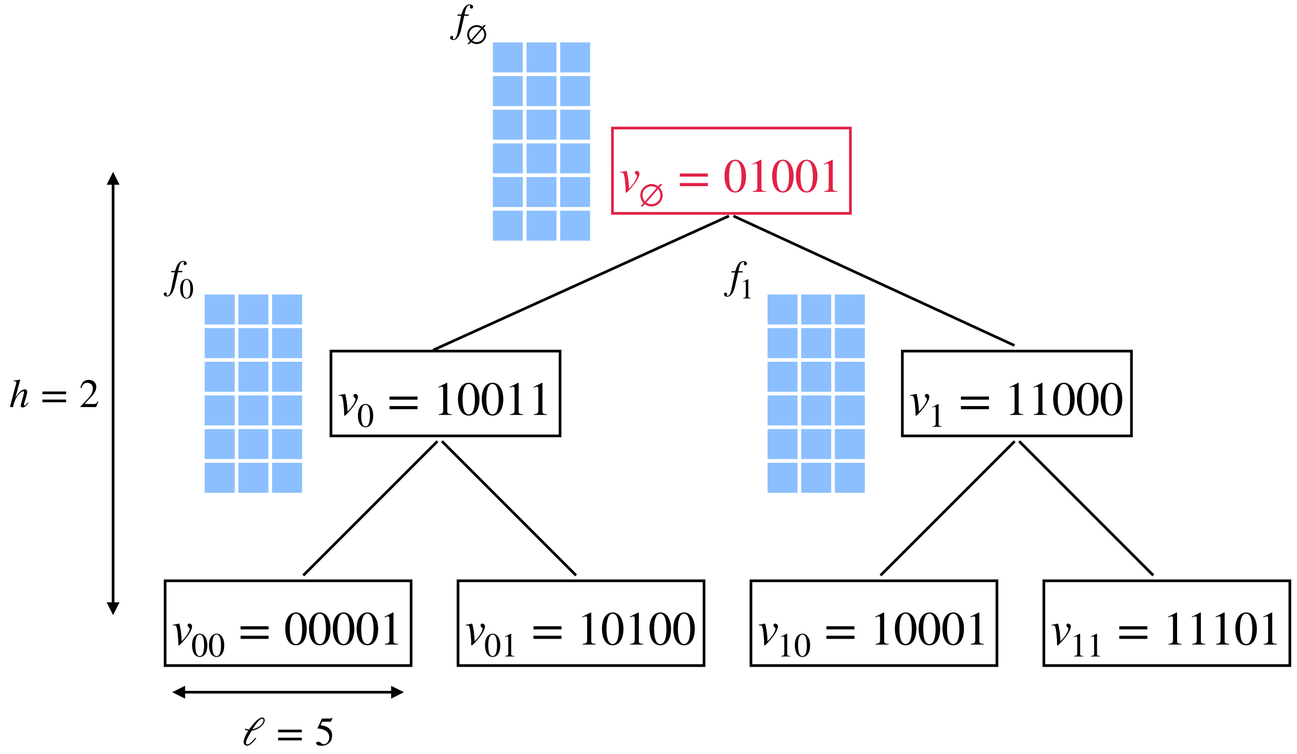
$$f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$$

- Goal: propagate the leaf values up the tree

$$v_u = f_u(v_{u0}, v_{u1})$$

- Output: the root value

This talk: low-space algorithms for Tree Evaluation

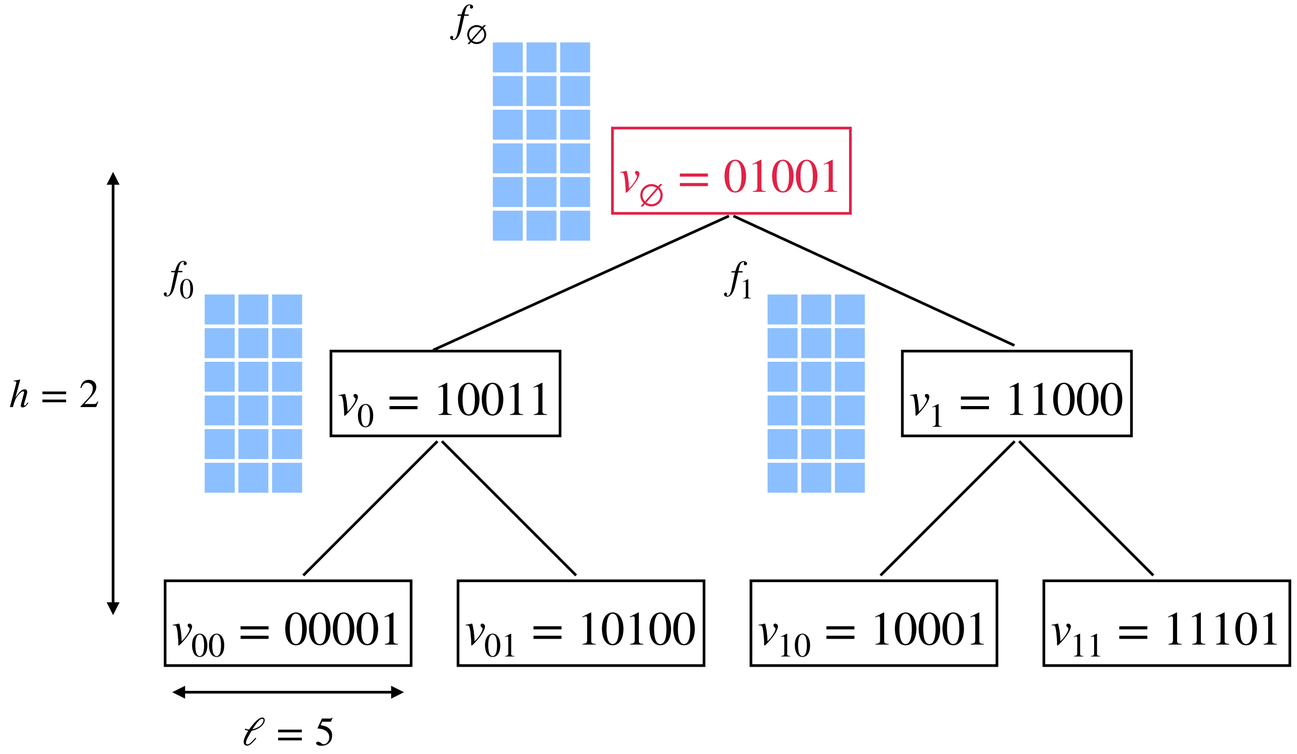


n = total input length = $\text{poly}(2^{h+\ell})$

The Tree Evaluation Problem

This talk: low-space algorithms for Tree Evaluation

Baseline: depth-first recursion ($O(\log^2 n)$ space, $\mathrm{poly}(n)$ time)



n = total input length = $\mathrm{poly}(2^{h+\ell})$


TreeEval 🤝 Time-Space Tradeoffs

TreeEval 🤝 Time-Space Tradeoffs

- *[Williams, STOC 2025]* Any time T computation reduces to a TreeEval instance with $h, \ell = O(\sqrt{T})$ (so $n = 2^{O(\sqrt{T})}$)

TreeEval 🤝 Time-Space Tradeoffs

- *[Williams, STOC 2025]* Any time T computation reduces to a TreeEval instance with $h, \ell = O(\sqrt{T})$ (so $n = 2^{O(\sqrt{T})}$)
- World 1: there is a more clever TreeEval algorithm that uses $O(\log n)$ space

TreeEval 🤝 Time-Space Tradeoffs

- *[Williams, STOC 2025]* Any time T computation reduces to a TreeEval instance with $h, \ell = O(\sqrt{T})$ (so $n = 2^{O(\sqrt{T})}$)
- World 1: there is a more clever TreeEval algorithm that uses $O(\log n)$ space
  - Then <u>any</u> time T computation could be done in space $O(\sqrt{T})$

TreeEval 🤝 Time-Space Tradeoffs

- *[Williams, STOC 2025]* Any time T computation reduces to a TreeEval instance with $h, \ell = O(\sqrt{T})$ (so $n = 2^{O(\sqrt{T})}$)
- World 1: there is a more clever TreeEval algorithm that uses $O(\log n)$ space
  - Then <u>any</u> time T computation could be done in space $O(\sqrt{T})$
- World 2: no such algorithm exists

TreeEval 🤝 Time-Space Tradeoffs

- *[Williams, STOC 2025]* Any time T computation reduces to a TreeEval instance with $h, \ell = O(\sqrt{T})$ (so $n = 2^{O(\sqrt{T})}$)
- World 1: there is a more clever TreeEval algorithm that uses $O(\log n)$ space
  - Then <u>any</u> time T computation could be done in space $O(\sqrt{T})$
- World 2: no such algorithm exists
  - Then TreeEval $\in$ P but not L $\Rightarrow$ L $\subsetneq$ P

TreeEval 🤝 Time-Space Tradeoffs

- *[Williams, STOC 2025]* Any time T computation reduces to a TreeEval instance with $h, \ell = O(\sqrt{T})$ (so $n = 2^{O(\sqrt{T})}$)
- World 1: there is a more clever TreeEval algorithm that uses $O(\log n)$ space
  - Then <u>any</u> time T computation could be done in space $O(\sqrt{T})$
- World 2: no such algorithm exists
  - Then TreeEval $\in$ P but not L $\Rightarrow$ L $\subsetneq$ P
  - P: problems solvable in time poly(n) on n-bit inputs

TreeEval 🤝 Time-Space Tradeoffs

- *[Williams, STOC 2025]* Any time T computation reduces to a TreeEval instance with $h, \ell = O(\sqrt{T})$ (so $n = 2^{O(\sqrt{T})}$)
- World 1: there is a more clever TreeEval algorithm that uses $O(\log n)$ space
  - Then <u>any</u> time T computation could be done in space $O(\sqrt{T})$
- World 2: no such algorithm exists
  - Then TreeEval $\in$ P but not L $\Rightarrow$ L $\subsetneq$ P
  - P: problems solvable in time poly(n) on n-bit inputs
  - L: problems solvable in space $O(\log n)$

Tree Evaluation Through the Years

- *[S. Cook et al., TOCT 2012]* Defined TreeEval, conjectured that $O(\log^2 n)$ space is optimal
- Many works, 2012-2019: evidence for this conjecture in restricted models of computation

Tree Evaluation Through the Years

- *[S. Cook et al., TOCT 2012]* Defined TreeEval, conjectured that $O(\log^2 n)$ space is optimal
- Many works, 2012-2019: evidence for this conjecture in restricted models of computation
- *[J. Cook-Mertz, STOC 2020]* **Conjecture is false!**
  - An algorithm in space $O(\log^2 n/\log\log n)$

Tree Evaluation Through the Years

- *[S. Cook et al., TOCT 2012]* Defined TreeEval, conjectured that $O(\log^2 n)$ space is optimal
- Many works, 2012-2019: evidence for this conjecture in restricted models of computation
- *[J. Cook-Mertz, STOC 2020]* **Conjecture is false!**
  - An algorithm in space $O(\log^2 n/\log\log n)$
- *[J. Cook-Mertz, STOC 2024]* **Conjecture is REALLY false.**
  - An algorithm in space $O(\log n \log\log n)$

Tree Evaluation Through the Years

- *[S. Cook et al., TOCT 2012]* Defined TreeEval, conjectured that $O(\log^2 n)$ space is optimal
- Many works, 2012-2019: evidence for this conjecture in restricted models of computation
- *[J. Cook-Mertz, STOC 2020]* **Conjecture is false!**
  - An algorithm in space $O(\log^2 n / \log\log n)$
- *[J. Cook-Mertz, STOC 2024]* **Conjecture is REALLY false.**
  - An algorithm in space $O(\log n \log\log n)$
- *[Williams, STOC 2025]* Any time T computation can be reduced to solving a TreeEval instance of size $2^{O(\sqrt{T})} \rightarrow$ doable with space $\tilde{O}(\sqrt{T})$!

Tree Evaluation Through the Years

- *[S. Cook et al., TOCT 2012]* Defined TreeEval, conjectured that $O(\log^2 n)$ space is optimal
- Many works, 2012-2019: evidence for this conjecture in restricted models of computation
- ***[J. Cook-Mertz, STOC 2020]* Conjecture is false!**
  - An algorithm in space $O(\log^2 n/\log\log n)$
- ***[J. Cook-Mertz, STOC 2024]* Conjecture is REALLY false.**
  - An algorithm in space $O(\log n \log\log n)$
- *[Williams, STOC 2025]* Any time T computation can be reduced to solving a TreeEval instance of size $2^{O(\sqrt{T})} \rightarrow$ doable with space $\tilde{O}(\sqrt{T})$!

Q: What is the key tool enabling these algorithms by Cook and Mertz?

A: Catalytic computation!

Catalytic Computing = "Bad" Programming

Introduced by *[Buhrman et al., STOC 2014]*

- Setting: our algorithm has computed some new information, and needs to store it somewhere

`10010010`

Working memory: `01111010|00010111|00101000|10100011`

Catalytic Computing = "Bad" Programming

Introduced by *[Buhrman et al., STOC 2014]*

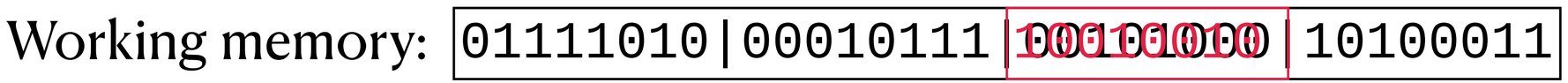
- Setting: our algorithm has computed some new information, and needs to store it somewhere
- Most algorithms ("good programming"): allocate new space for this information

Working memory: `01111010|00010111|00101000|10100011` `10010010`

Catalytic Computing = "Bad" Programming

Introduced by *[Buhrman et al., STOC 2014]*

- Setting: our algorithm has computed some new information, and needs to store it somewhere
- Most algorithms ("good programming"): allocate new space for this information
- Catalytic computing ("bad programming"): write this information on top of occupied space!



Catalytic Computing: Formal Definition

Introduced by *[Buhrman et al., STOC 2014]*

- A CatSpaceTime($C(n), S(n), T(n)$) algorithm has an input $x \in \{0,1\}^n$ and a *catalytic tape* $\tau_{\text{init}} \in \{0,1\}^{O(C(n))}$

Catalytic Computing: Formal Definition

Introduced by *[Buhrman et al., STOC 2014]*

- A CatSpaceTime($C(n), S(n), T(n)$) algorithm has an input $x \in \{0,1\}^n$ and a *catalytic tape* $\tau_{\text{init}} \in \{0,1\}^{O(C(n))}$
  - Time limit: $O(T(n))$

Catalytic Computing: Formal Definition

Introduced by *[Buhrman et al., STOC 2014]*

- A CatSpaceTime($C(n), S(n), T(n)$) algorithm has an input $x \in \{0,1\}^n$ and a *catalytic tape* $\tau_{\text{init}} \in \{0,1\}^{O(C(n))}$
  - Time limit: $O(T(n))$
  - Space limit: $O(S(n))$ clean bits + the catalytic tape containing τ

Catalytic Computing: Formal Definition

Introduced by *[Buhrman et al., STOC 2014]*

- A CatSpaceTime($C(n), S(n), T(n)$) algorithm has an input $x \in \{0,1\}^n$ and a *catalytic tape* $\tau_{\text{init}} \in \{0,1\}^{O(C(n))}$
  - Time limit: $O(T(n))$
  - Space limit: $O(S(n))$ clean bits + the catalytic tape containing τ
  - At the end: the catalytic tape must be restored to the state τ_{init}

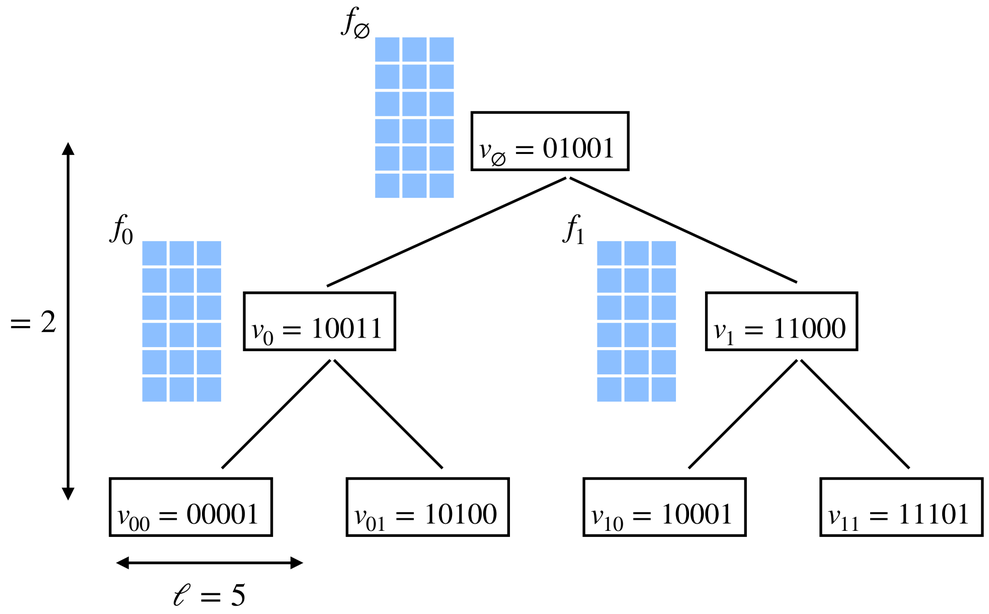
Catalytic Computing: Formal Definition

Introduced by *[Buhrman et al., STOC 2014]*

- A CatSpaceTime($C(n), S(n), T(n)$) algorithm has an input $x \in \{0,1\}^n$ and a *catalytic tape* $\tau_{\text{init}} \in \{0,1\}^{O(C(n))}$
  - Time limit: $O(T(n))$
  - Space limit: $O(S(n))$ clean bits + the catalytic tape containing τ
  - At the end: the catalytic tape must be restored to the state τ_{init}
- CatSpaceTime($C(n), S(n), T(n)$) $\subseteq$ CatSpaceTime($0, C(n) + S(n), T(n)$)
  - “Free space is at least as good as catalytic space”

Catalytic Tree Evaluation Through the Years

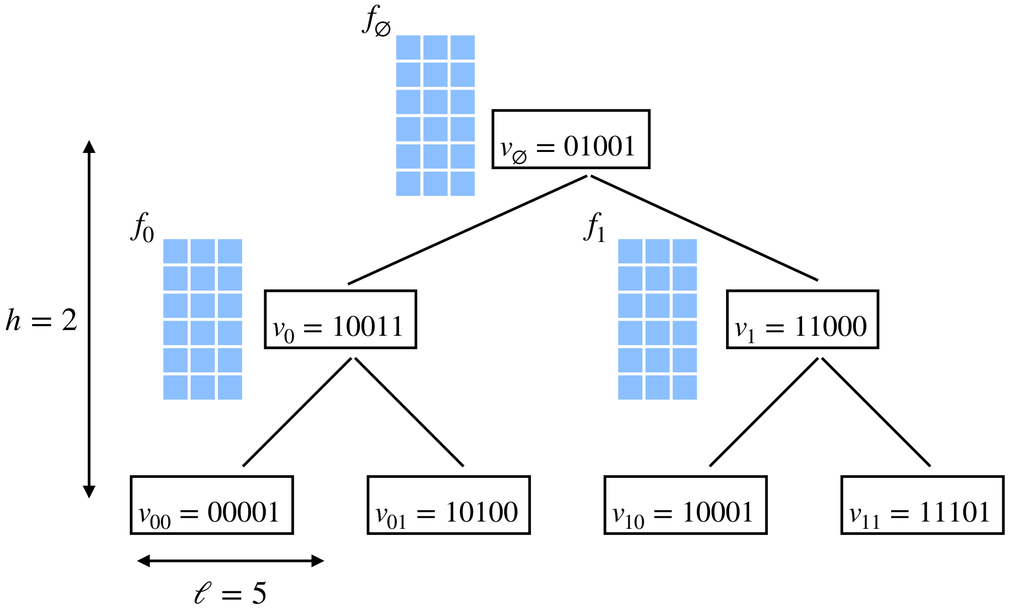
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	poly(n)



n = total input length = poly$(2^{h+\ell})$

Catalytic Tree Evaluation Through the Years

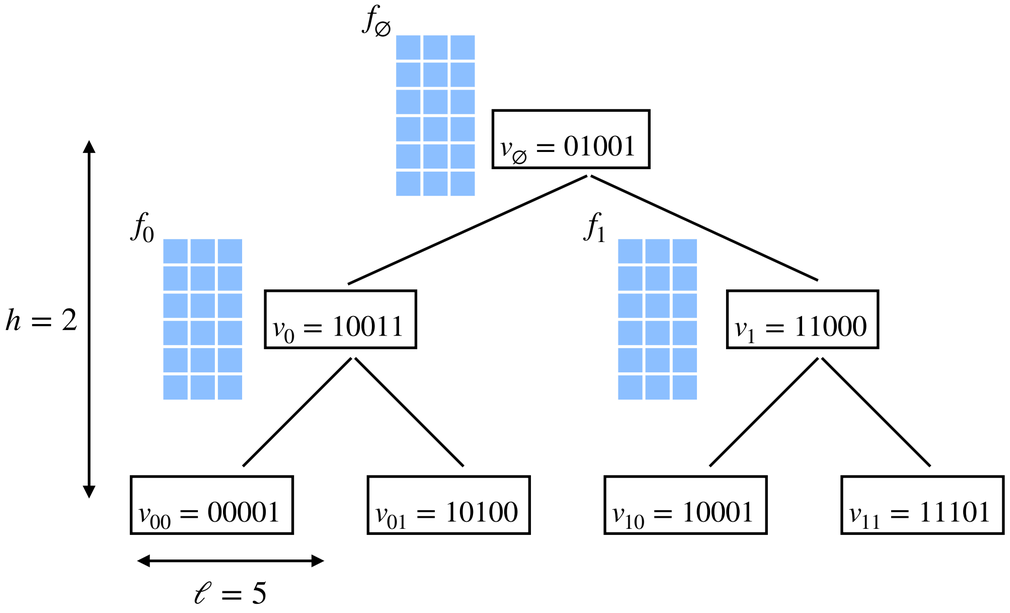
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	$\text{poly}(n)$
Buhrman et al. (STOC 2014)	$\text{poly}(n)$	$\log n$	$\text{poly}(n)$



n = total input length = $\text{poly}(2^{h+\ell})$

Catalytic Tree Evaluation Through the Years

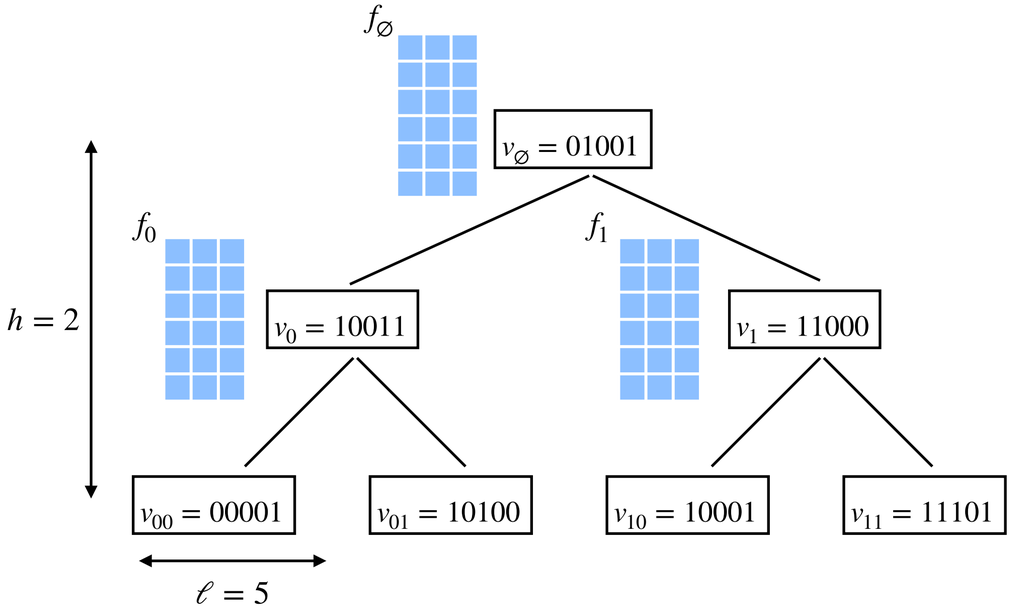
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	$\text{poly}(n)$
Buhrman et al. (STOC 2014)	$\text{poly}(n)$	$\log n$	$\text{poly}(n)$
J. Cook-Mertz (STOC 2024)		$O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$
	$\ell = O(\log n)$	$h \log \ell = O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$



n = total input length = $\text{poly}(2^{h+\ell})$

Catalytic Tree Evaluation Through the Years

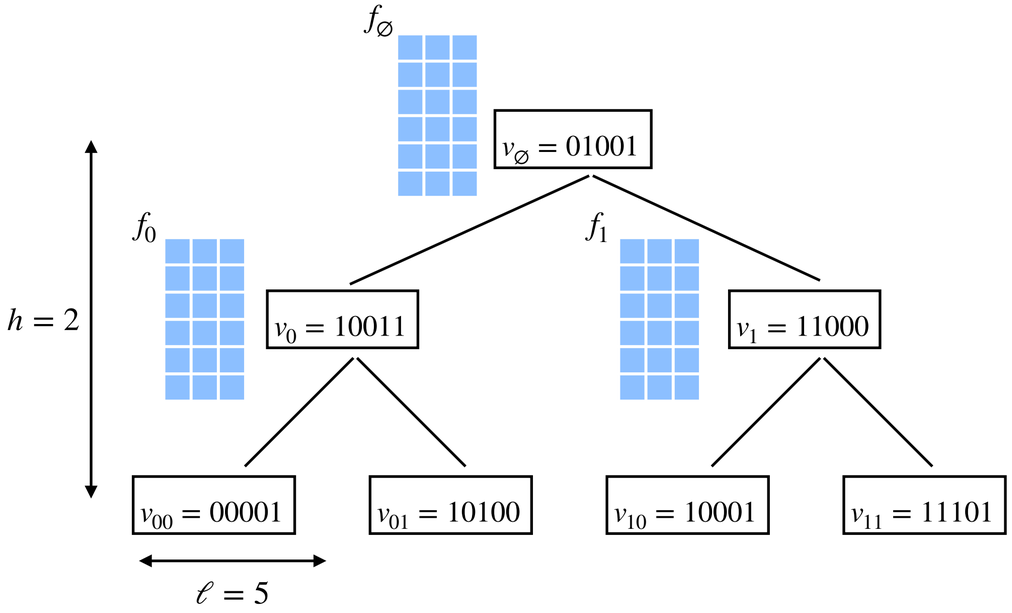
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	$\text{poly}(n)$
Buhrman et al. (STOC 2014)	$\text{poly}(n)$	$\log n$	$\text{poly}(n)$
J. Cook-Mertz (STOC 2024)		$O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$
	$\ell = O(\log n)$	$h \log \ell = O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$
Our work	$2^{\ell^\epsilon} = 2^{\log^\epsilon n}$	$\ell + h = O(\log n)$	$\text{poly}(n)$



n = total input length = $\text{poly}(2^{h+\ell})$

Catalytic Tree Evaluation Through the Years

Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	$\text{poly}(n)$
Buhrman et al. (STOC 2014)	$\text{poly}(n)$	$\log n$	$\text{poly}(n)$
J. Cook-Mertz (STOC 2024)		$O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$
	$\ell = O(\log n)$	$h \log \ell = O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$
Our work	$2^{\ell^\epsilon} = 2^{\log^\epsilon n}$	$\ell + h = O(\log n)$	$\text{poly}(n)$



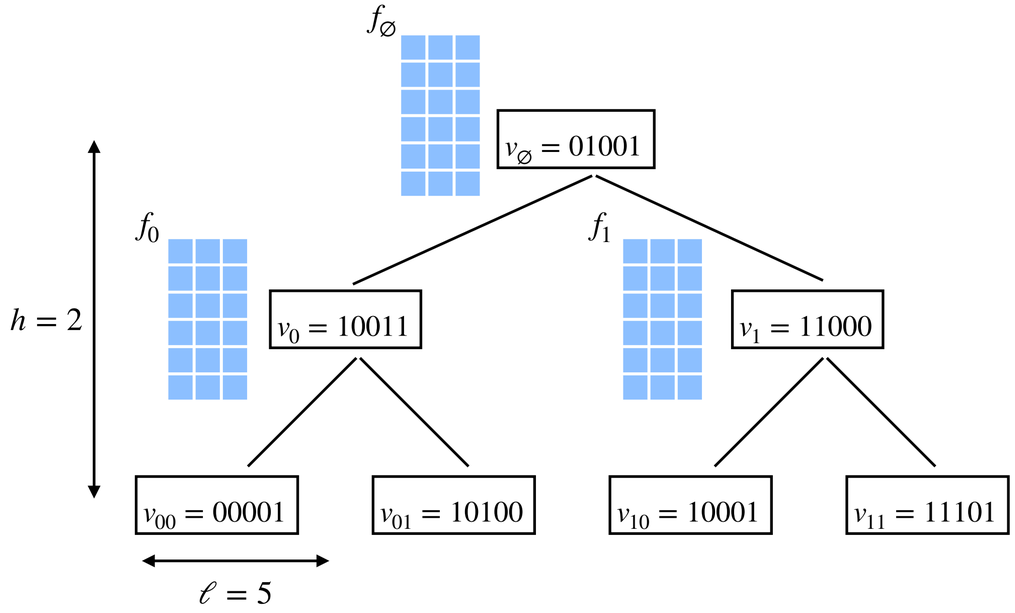
n = total input length = $\text{poly}(2^{h+\ell})$

Strictly better than Buhrman et al.

Incomparable with Cook-Mertz

Simulating Time T Turing Machines

TreeEval Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(T)$	$2^{O(\sqrt{T})}$
Buhrman et al. (STOC 2014)	$2^{O(\sqrt{T})}$	$O(\sqrt{T})$	$2^{O(\sqrt{T})}$
J. Cook-Mertz (STOC 2024)		$O(\sqrt{T}\log T)$	$2^{O(\sqrt{T}\log T)}$
	$\ell = O(\sqrt{T})$	$h\log\ell$ $= O(\sqrt{T}\log T)$	ℓ^h $= 2^{O(\sqrt{T}\log T)}$
Our work	$2^{\ell^\epsilon} = 2^{T^{\epsilon/2}}$	$\ell + h = O(\sqrt{T})$	$2^{O(\sqrt{T})}$



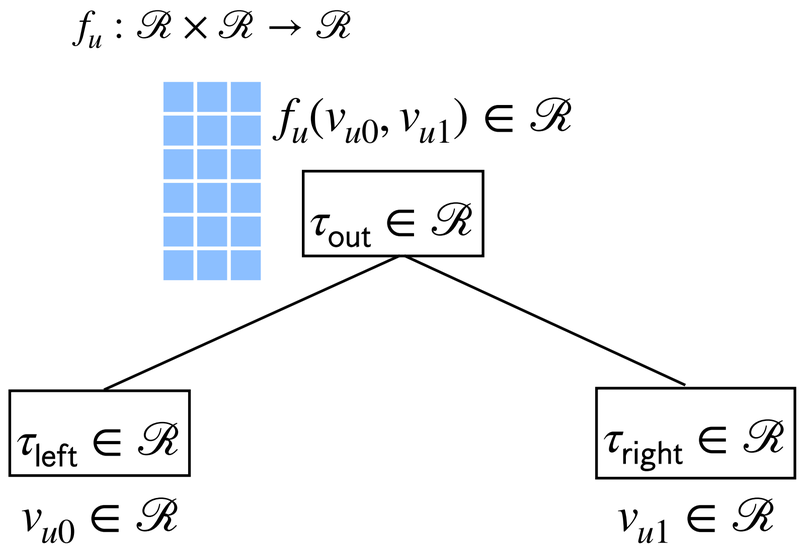
n = total input length $= 2^{O(\sqrt{T})}$

Strictly better than Buhrman et al.

Incomparable with Cook-Mertz

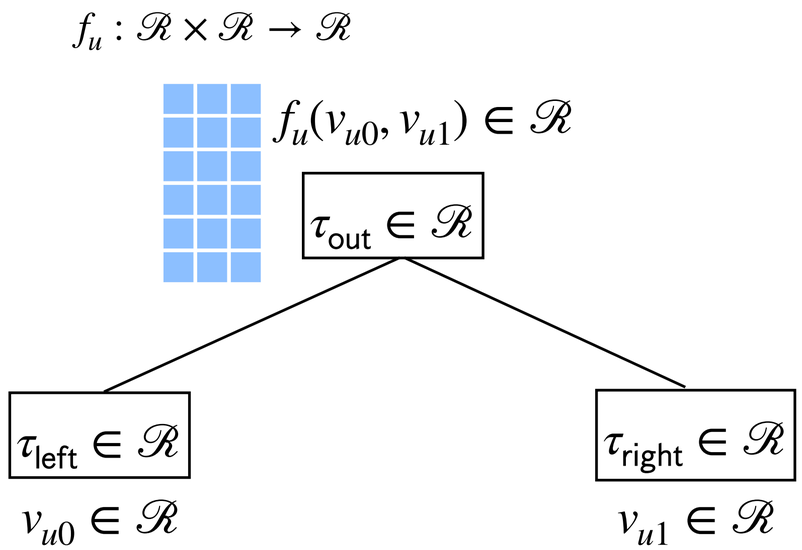
Approach: The One-Level Gadget

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$



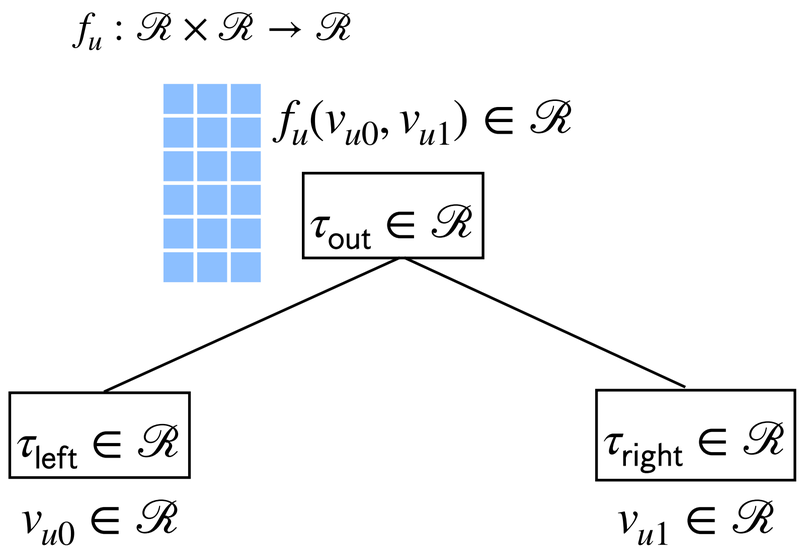
Approach: The One-Level Gadget

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Arbitrary catalytic registers $\tau_{\text{left}}, \tau_{\text{right}}, \tau_{\text{out}} \in \mathscr{R}$



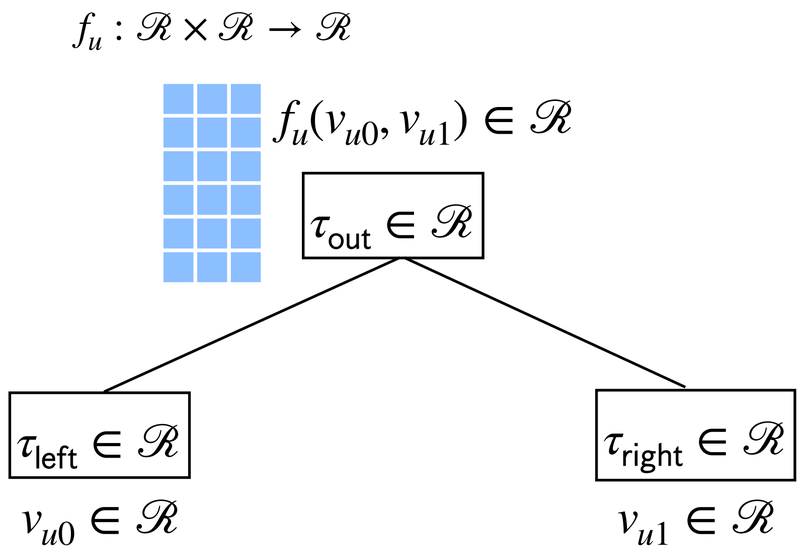
Approach: The One-Level Gadget

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Arbitrary catalytic registers $\tau_{\mathsf{left}}, \tau_{\mathsf{right}}, \tau_{\mathsf{out}} \in \mathscr{R}$
  - Oracles $\mathcal{O}_{u0}, \mathcal{O}_{u1}$ implicitly computing v_{u0}, v_{u1}



Approach: The One-Level Gadget

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Arbitrary catalytic registers $\tau_{\text{left}}, \tau_{\text{right}}, \tau_{\text{out}} \in \mathscr{R}$
  - Oracles $\mathcal{O}_{u0}, \mathcal{O}_{u1}$ implicitly computing v_{u0}, v_{u1}

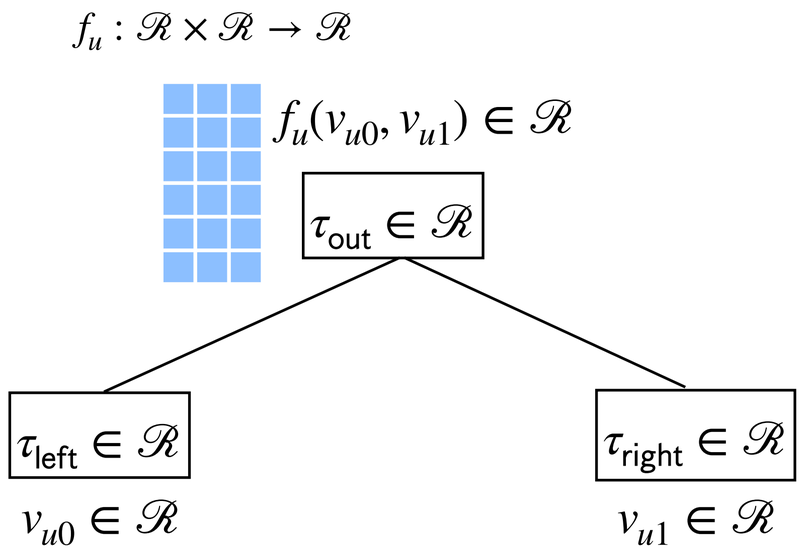


Cheatsheet

$\mathcal{O}_u(\tau \in \mathscr{R})$: updates $\tau \leftarrow \tau \pm v_u$

Approach: The One-Level Gadget

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Arbitrary catalytic registers $\tau_{\text{left}}, \tau_{\text{right}}, \tau_{\text{out}} \in \mathscr{R}$
  - Oracles $\mathcal{O}_{u0}, \mathcal{O}_{u1}$ implicitly computing v_{u0}, v_{u1}
- Goal: implement the oracle $\mathcal{O}_u$ on τ_{out} i.e., want the registers to end up in the state



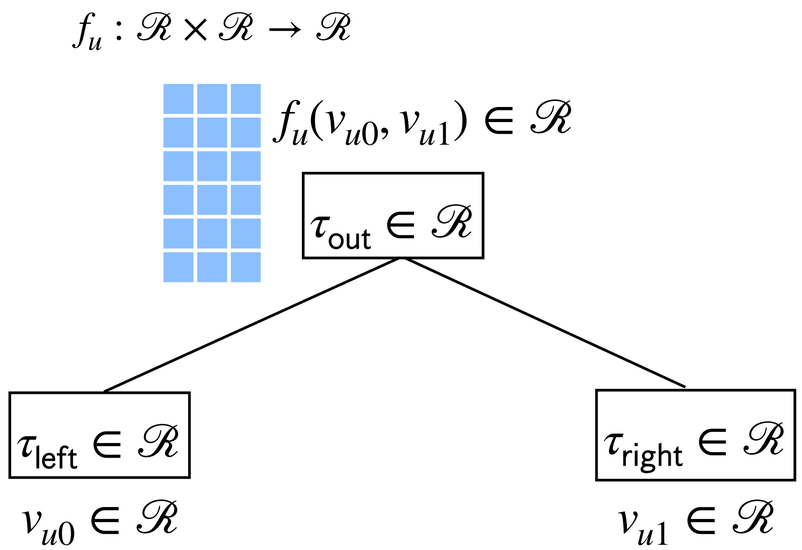
Cheatsheet

$\mathcal{O}_u(\tau \in \mathscr{R})$: updates $\tau \leftarrow \tau \pm v_u$

Approach: The One-Level Gadget

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Arbitrary catalytic registers $\tau_{\text{left}}, \tau_{\text{right}}, \tau_{\text{out}} \in \mathscr{R}$
  - Oracles $\mathcal{O}_{u0}, \mathcal{O}_{u1}$ implicitly computing v_{u0}, v_{u1}
- Goal: implement the oracle $\mathcal{O}_u$ on τ_{out} i.e., want the registers to end up in the state

$$(\tau_{\text{left}}, \tau_{\text{right}}, \tau_{\text{out}} \pm f_u(v_{u0}, v_{u1}))$$



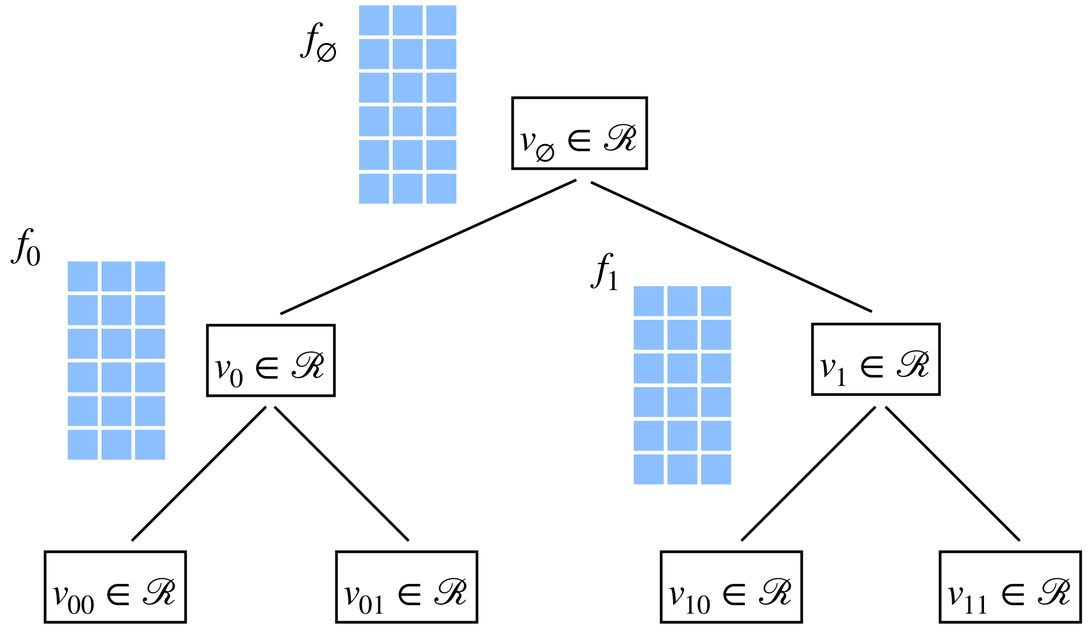
Cheatsheet

$\mathcal{O}_u(\tau \in \mathscr{R})$: updates $\tau \leftarrow \tau \pm v_u$

From One Level to Many

Theorem (informal): assume there is a $\mathsf{poly}(2^{\ell})$ time algorithm for the one-level gadget that makes q oracle queries and uses free space s. Then:

$$\mathsf{TreeEval}(h, \ell) \in \mathsf{CatSpaceTime}(\log|\mathscr{R}|, hs+\log\log|\mathscr{R}|+\ell, \mathsf{poly}(q^h, 2^{\ell}))$$



Metrics

- $\log|\mathscr{R}|$: final catalytic space
- $h \cdot$ [free space]: final free space
- [oracle queries]^h: final runtime

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR ↔ Tree Evaluation
- Formal abstraction: matching vectors and decoding polynomials
- Conclusion

Abstraction: PIR from Private Polynomial Evaluation

A common step in many PIR protocols

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$v \in \mathbb{F}^{\ell}$

- Low-degree extension f_{DB}

Abstraction: PIR from Private Polynomial Evaluation

A common step in many PIR protocols

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$v \in \mathbb{F}^{\ell}$

- Low-degree extension f_{DB}
  - Query index $i \in [n]$ encoded as a point $p \in \mathbb{F}^{\ell}$

Abstraction: PIR from Private Polynomial Evaluation

A common step in many PIR protocols

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$v \in \mathbb{F}^{\ell}$

- Low-degree extension f_{DB}
  - Query index $i \in [n]$ encoded as a point $p \in \mathbb{F}^{\ell}$
  - Require $f_{\mathsf{DB}}(p) = \mathsf{DB}_i$

Abstraction: PIR from Private Polynomial Evaluation

A common step in many PIR protocols

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 … Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$v \in \mathbb{F}^{\ell}$

- Low-degree extension f_{DB}
  - Query index $i \in [n]$ encoded as a point $p \in \mathbb{F}^{\ell}$
  - Require $f_{\mathsf{DB}}(p) = \mathsf{DB}_i$
  - Parameter counting $\to$ need $2^{\ell} \geq n$

Abstraction: PIR from Private Polynomial Evaluation

A common step in many PIR protocols

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$v \in \mathbb{F}^{\ell}$

- Low-degree extension f_{DB}
  - Query index $i \in [n]$ encoded as a point $p \in \mathbb{F}^{\ell}$
  - Require $f_{\mathsf{DB}}(p) = \mathsf{DB}_i$
  - Parameter counting $\to$ need $2^{\ell} \geq n$
- Correctness: client can deduce $f_{\mathsf{DB}}(v)$ from $\{q_j, a_j\}_{j \in [S]}$

Abstraction: PIR from Private Polynomial Evaluation

A common step in many PIR protocols

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$v \in \mathbb{F}^{\ell}$

- Low-degree extension f_{DB}
  - Query index $i \in [n]$ encoded as a point $p \in \mathbb{F}^{\ell}$
  - Require $f_{\mathsf{DB}}(p) = \mathsf{DB}_i$
  - Parameter counting $\to$ need $2^{\ell} \geq n$
- Correctness: client can deduce $f_{\mathsf{DB}}(v)$ from $\{q_j, a_j\}_{j \in [S]}$
- Privacy: each server learns nothing about about v

Reed-Muller PIR

The heart of Cook-Mertz's one-level gadget

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$v \in \mathbb{F}^{\ell}$

- Queries: Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$

Reed-Muller PIR

The heart of Cook-Mertz's one-level gadget

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

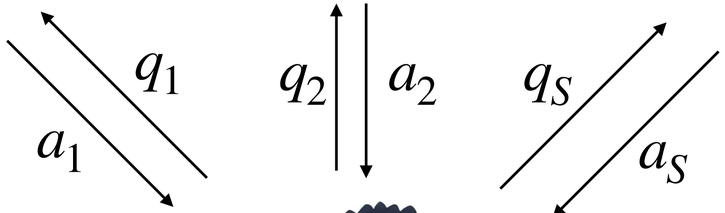
$v \in \mathbb{F}^{\ell}$

- Queries: Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
- Server j replies with $a_j = f_{\mathsf{DB}}(q_j) = f_{\mathsf{DB}}(v + j\tau)$

Reed-Muller PIR

The heart of Cook-Mertz's one-level gadget

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear



Client

$v \in \mathbb{F}^{\ell}$

- Queries: Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
- Server j replies with $a_j = f_{\mathsf{DB}}(q_j) = f_{\mathsf{DB}}(v + j\tau)$
- Alice's reconstruction:

Reed-Muller PIR

The heart of Cook-Mertz's one-level gadget

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

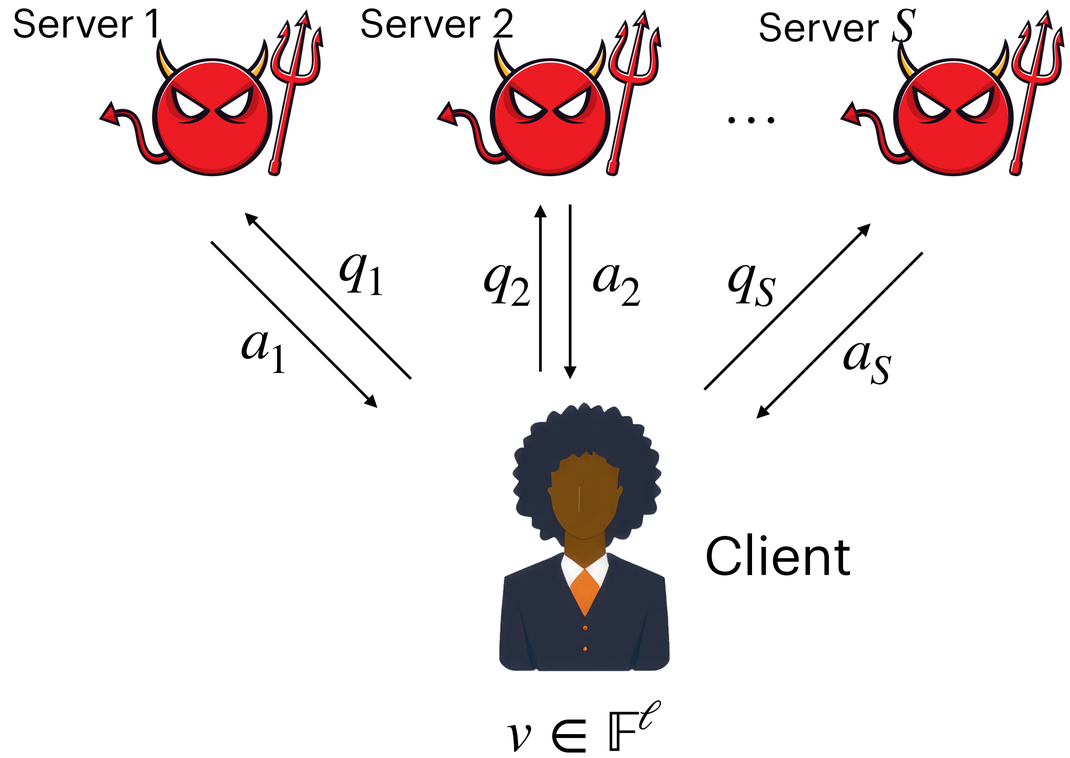
$v \in \mathbb{F}^{\ell}$

- Queries: Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
- Server j replies with $a_j = f_{\mathsf{DB}}(q_j) = f_{\mathsf{DB}}(v + j\tau)$
- Alice's reconstruction:
  - Define the line $L = \{(v + \lambda\tau) : \lambda \in \mathbb{F}\}$

Reed-Muller PIR

The heart of Cook-Mertz's one-level gadget

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear



- Queries: Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
- Server j replies with $a_j = f_{\mathsf{DB}}(q_j) = f_{\mathsf{DB}}(v + j\tau)$
- Alice's reconstruction:
  - Define the line $L = \{(v + \lambda\tau) : \lambda \in \mathbb{F}\}$
  - Idea: recover $g = f_{\mathsf{DB}}\big|_L$ via Lagrange interpolation

Reed-Muller PIR

The heart of Cook-Mertz's one-level gadget

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 … Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

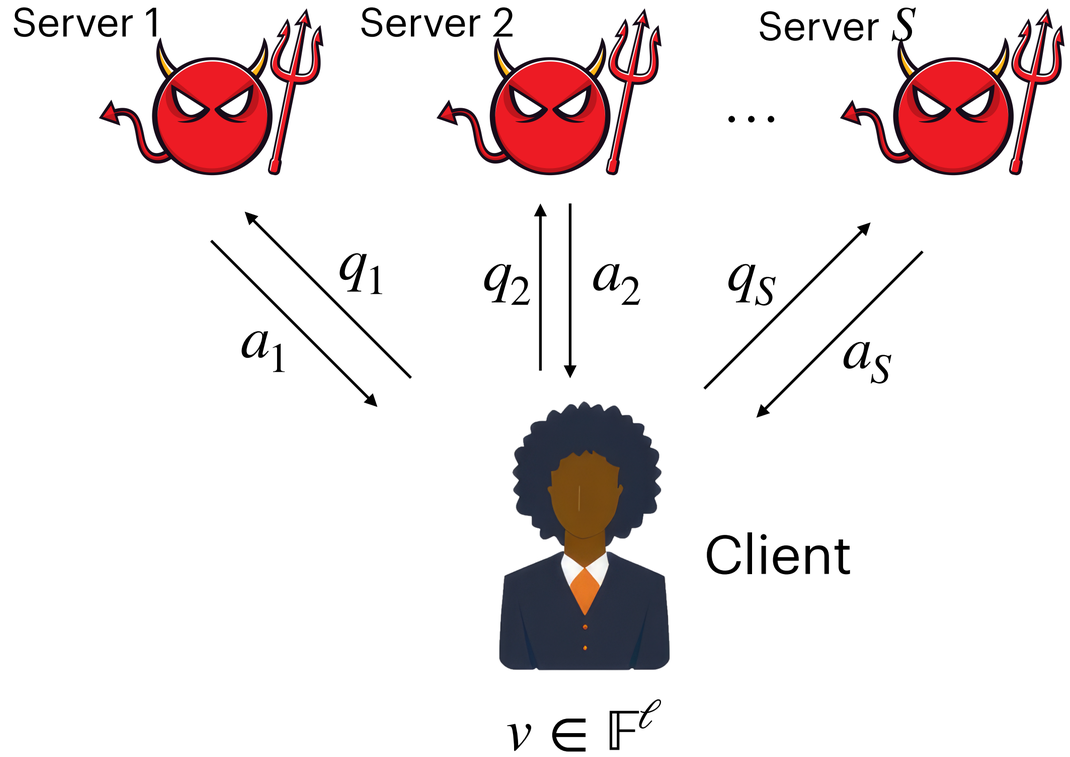
$v \in \mathbb{F}^{\ell}$

- Queries: Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
- Server j replies with $a_j = f_{\mathsf{DB}}(q_j) = f_{\mathsf{DB}}(v + j\tau)$
- Alice's reconstruction:
  - Define the line $L = \{(v + \lambda\tau) : \lambda \in \mathbb{F}\}$
  - Idea: recover $g = f_{\mathsf{DB}}\big|_L$ via Lagrange interpolation
  - $\deg g \leq \ell$ and Alice gets S evaluations → possible as long as $|\mathbb{F}| > S > \ell$

Reed-Muller PIR

The heart of Cook-Mertz's one-level gadget

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear



- Queries: Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
- Server j replies with $a_j = f_{\mathsf{DB}}(q_j) = f_{\mathsf{DB}}(v + j\tau)$
- Alice's reconstruction:
  - Define the line $L = \{(v + \lambda\tau) : \lambda \in \mathbb{F}\}$
  - Idea: recover $g = f_{\mathsf{DB}}\big|_L$ via Lagrange interpolation
  - $\deg g \leq \ell$ and Alice gets S evaluations → possible as long as $|\mathbb{F}| > S > \ell$
  - Answer: $g(0) = f_{\mathsf{DB}}(v)$

Reed-Muller PIR

Efficiency metrics

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$

- Parameters: only need
  $|\mathbb{F}| > S > \ell$ so can set
  $|\mathbb{F}|, S = O(\ell)$

Reed-Muller PIR

Efficiency metrics

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$

- Parameters: only need $|\mathbb{F}| > S > \ell$ so can set $|\mathbb{F}|, S = O(\ell)$
- **# of servers:** $S = O(\ell)$

Reed-Muller PIR

Efficiency metrics

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 Server S

$\cdots$

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$

- Parameters: only need $|\mathbb{F}| > S > \ell$ so can set $|\mathbb{F}|, S = O(\ell)$
- **# of servers:** $S = O(\ell)$
- **Communication complexity:** vector of ℓ field elements per server, so:

Reed-Muller PIR

Efficiency metrics

$f_{\mathsf{DB}} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 Server S

$\cdots$

q_1 a_1 q_2 a_2 q_s a_s

Client

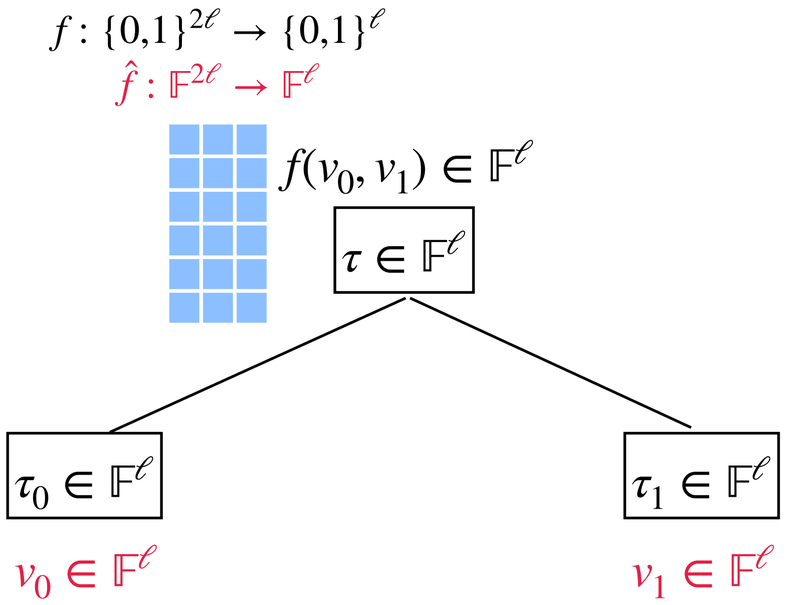
$v \in \mathbb{F}^{\ell}$

- Parameters: only need $|\mathbb{F}| > S > \ell$ so can set $|\mathbb{F}|, S = O(\ell)$
- **# of servers:** $S = O(\ell)$
- **Communication complexity:** vector of ℓ field elements per server, so:

$$\mathsf{CC} = O(\ell \log |\mathbb{F}|) = O(\ell \log \ell)$$

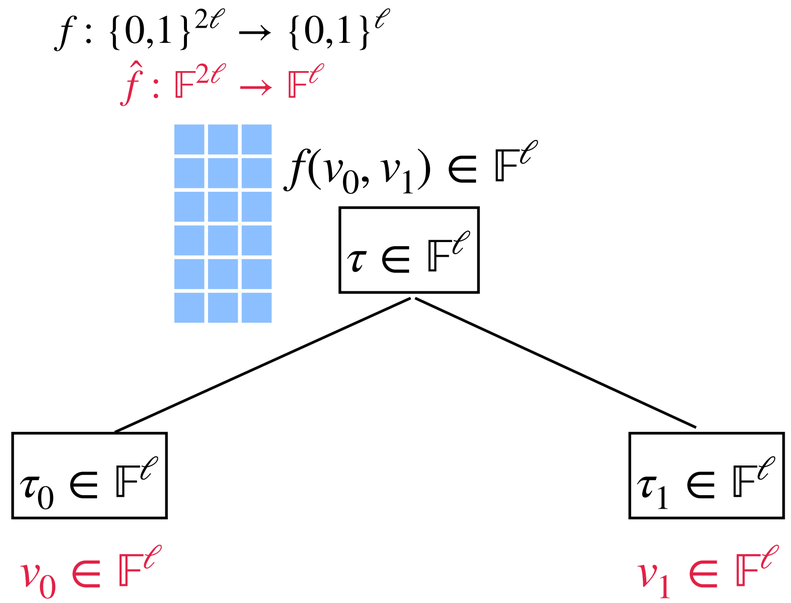
From Reed-Muller to Cook-Mertz

- Embed $\{0,1\}$ into a field $\mathbb{F}$ of size $\geq 2\ell + 2$



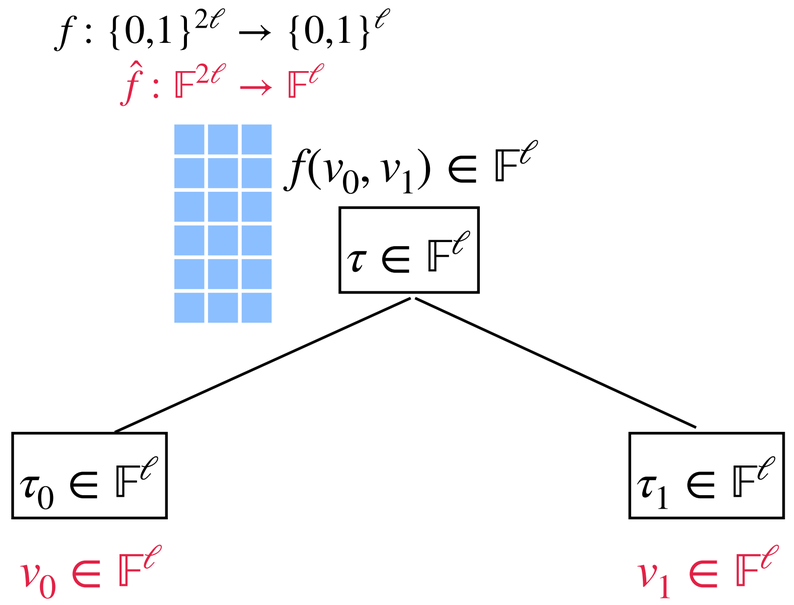
From Reed-Muller to Cook-Mertz

- Embed $\{0,1\}$ into a field $\mathbb{F}$ of size $\geq 2\ell + 2$
- Ring $\mathscr{R} = \mathbb{F}^{\ell}$



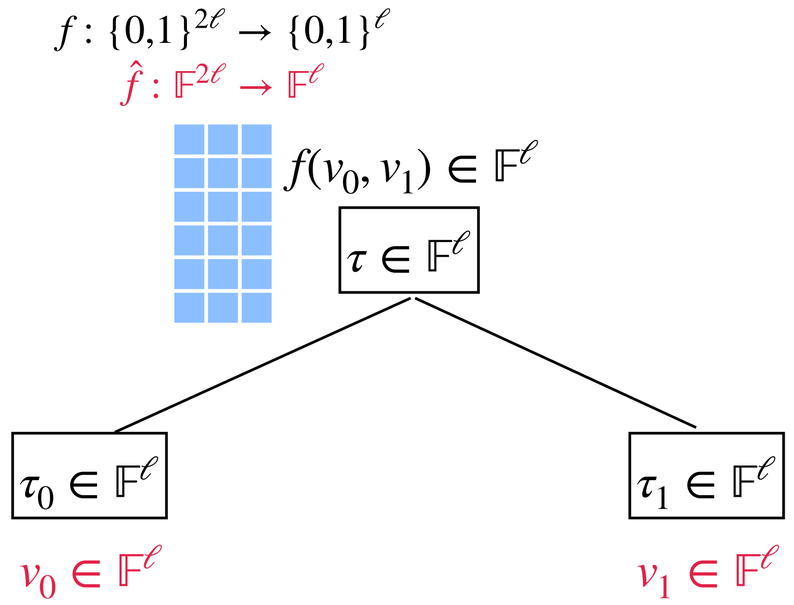
From Reed-Muller to Cook-Mertz

- Embed $\{0,1\}$ into a field $\mathbb{F}$ of size $\geq 2\ell + 2$
- Ring $\mathscr{R} = \mathbb{F}^{\ell}$
- Multilinear extension $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$



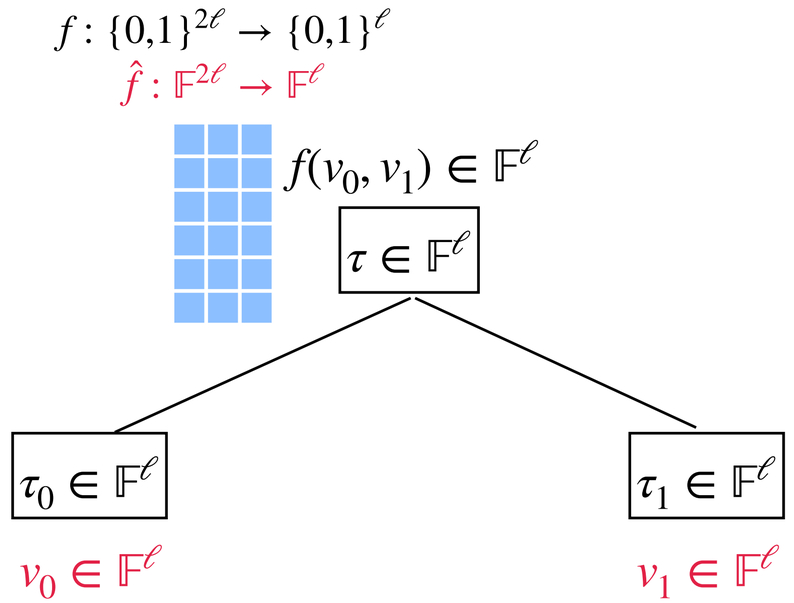
From Reed-Muller to Cook-Mertz

- Embed $\{0,1\}$ into a field $\mathbb{F}$ of size $\geq 2\ell + 2$
- Ring $\mathscr{R} = \mathbb{F}^{\ell}$
- Multilinear extension $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - For all $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$



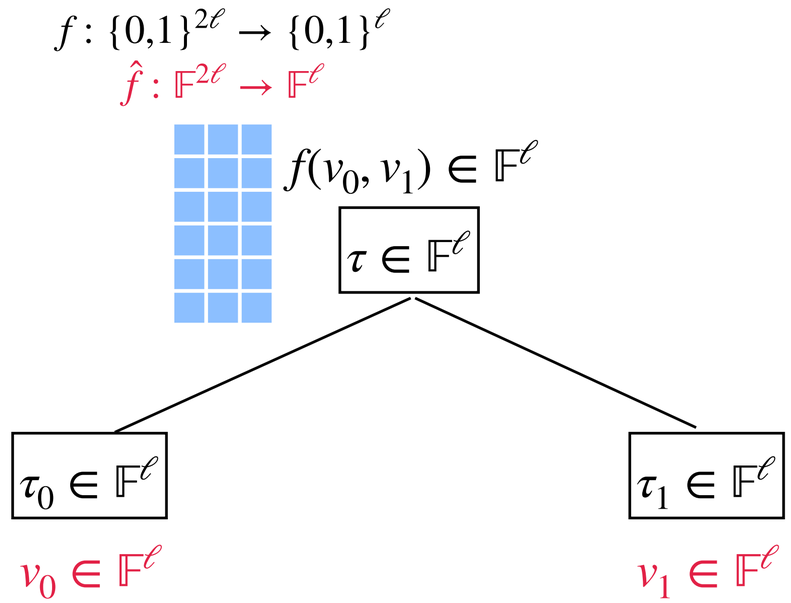
From Reed-Muller to Cook-Mertz

- Embed $\{0,1\}$ into a field $\mathbb{F}$ of size $\geq 2\ell + 2$
- Ring $\mathscr{R} = \mathbb{F}^{\ell}$
- Multilinear extension $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - For all $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$
- Refresher on the one-level gadget:



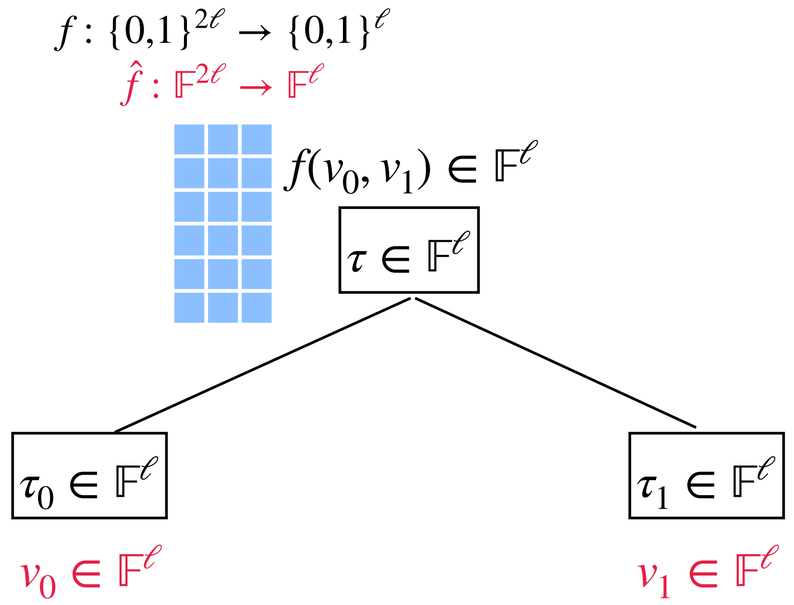
From Reed-Muller to Cook-Mertz

- Embed $\{0,1\}$ into a field $\mathbb{F}$ of size $\geq 2\ell + 2$
- Ring $\mathscr{R} = \mathbb{F}^{\ell}$
- Multilinear extension $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - For all $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$
- Refresher on the one-level gadget:
  - Input: oracle that adds or subtracts v_b to the register containing τ_b



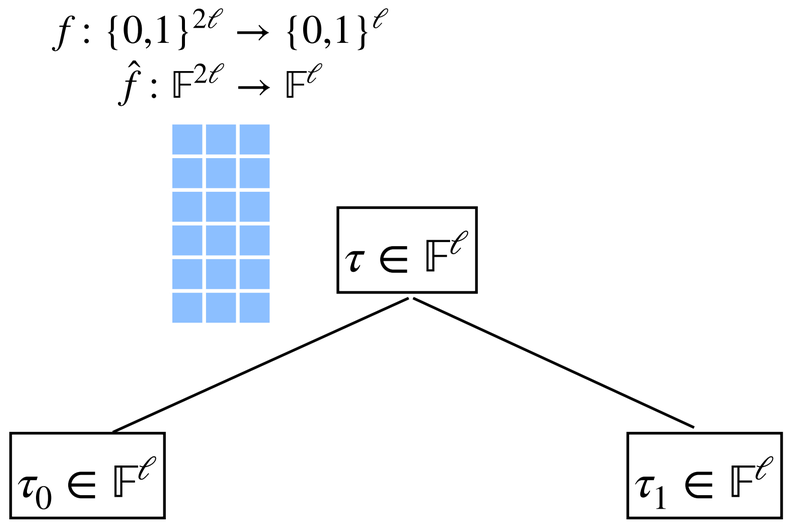
From Reed-Muller to Cook-Mertz

- Embed $\{0,1\}$ into a field $\mathbb{F}$ of size $\geq 2\ell + 2$
- Ring $\mathscr{R} = \mathbb{F}^{\ell}$
- Multilinear extension $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - For all $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$
- Refresher on the one-level gadget:
  - Input: oracle that adds or subtracts v_b to the register containing τ_b
  - Output: add or subtract $f(v_0, v_1)$ to τ



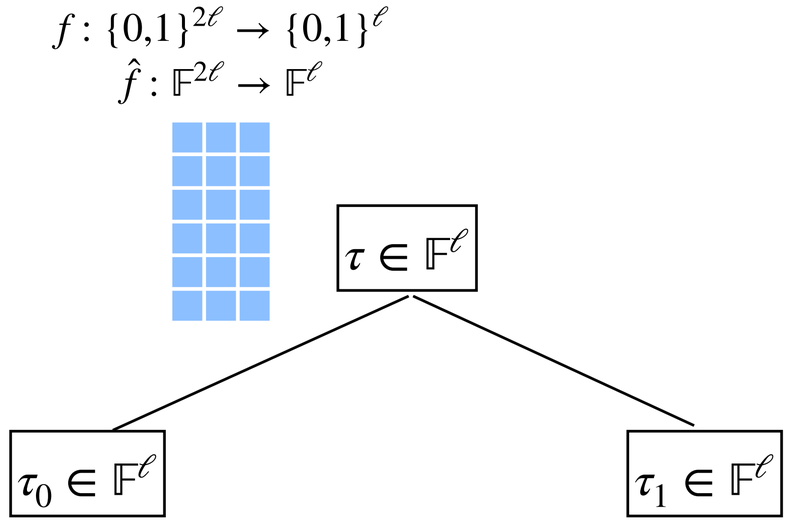
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$



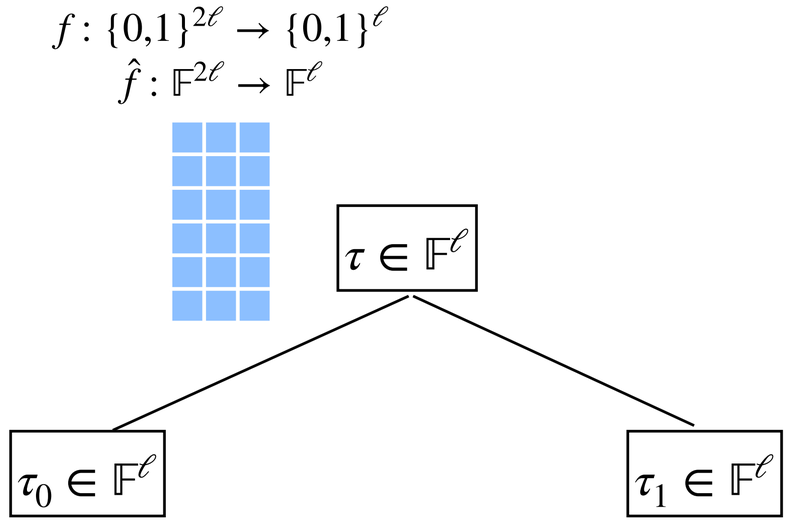
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$



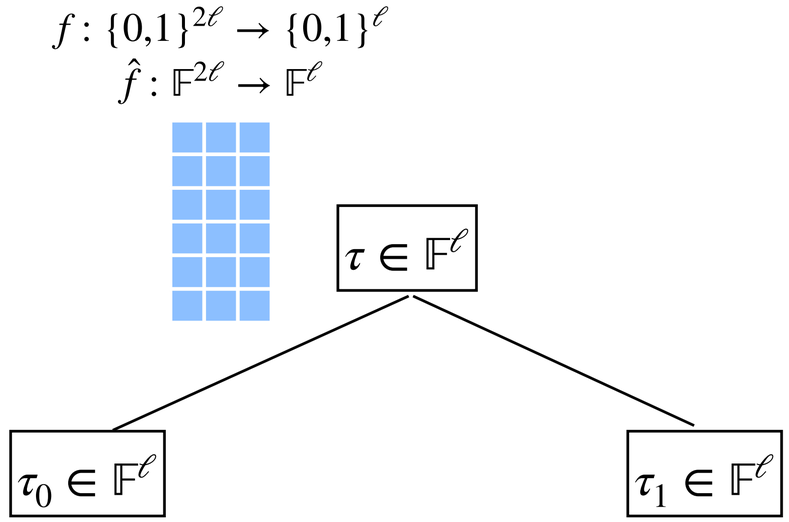
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$



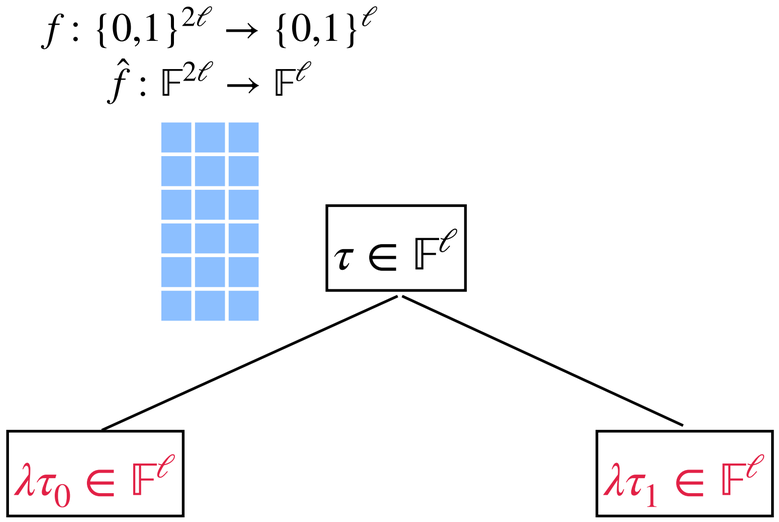
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:



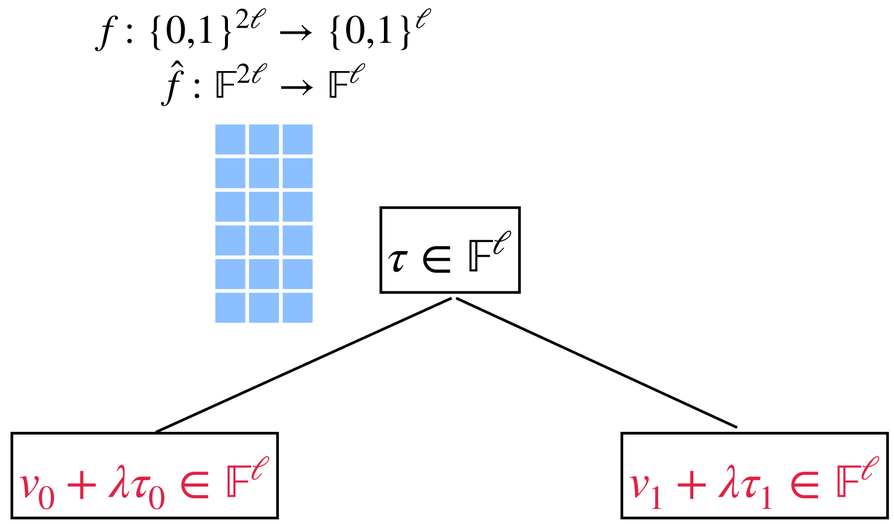
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$



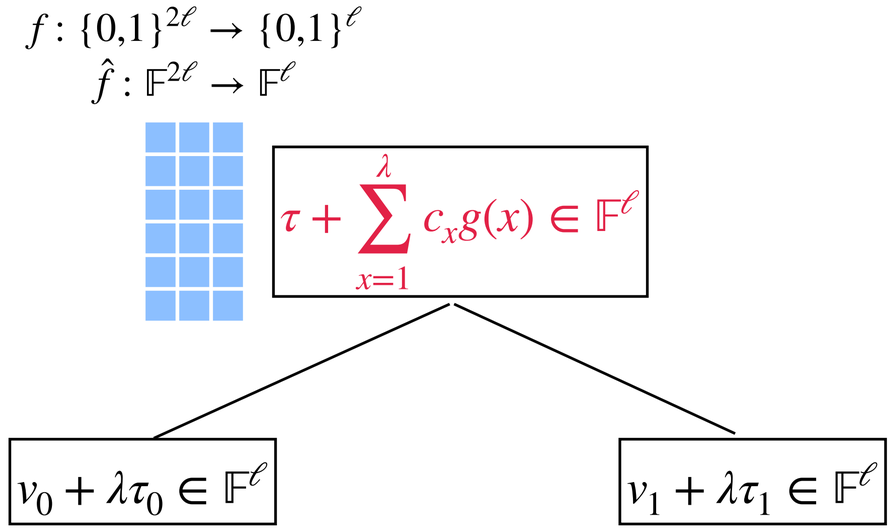
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls → add v_0, v_1 into the input registers



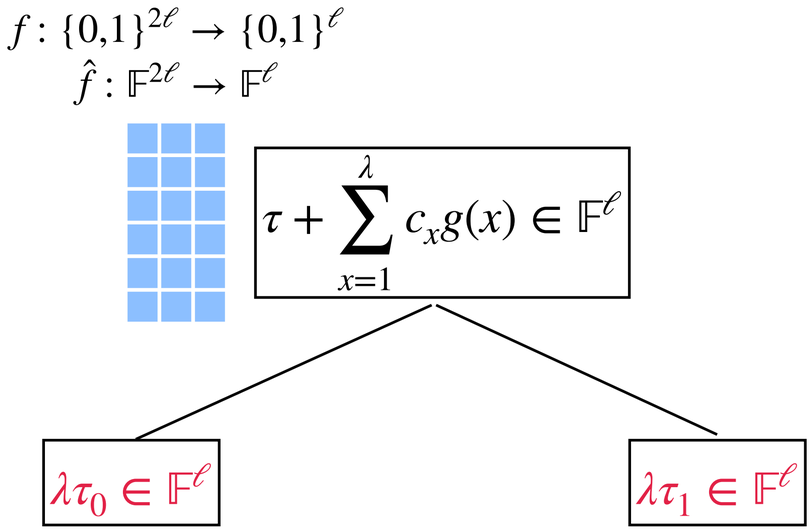
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls → add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ



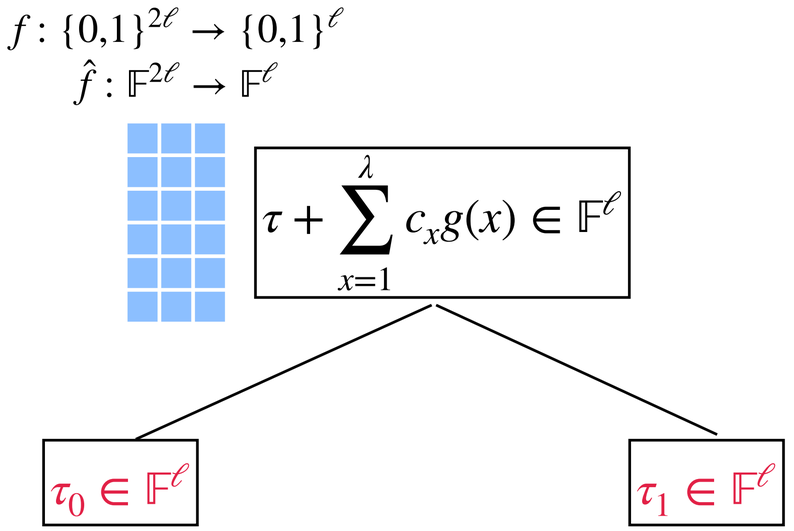
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls → add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls → subtract v_0, v_1 from the input registers



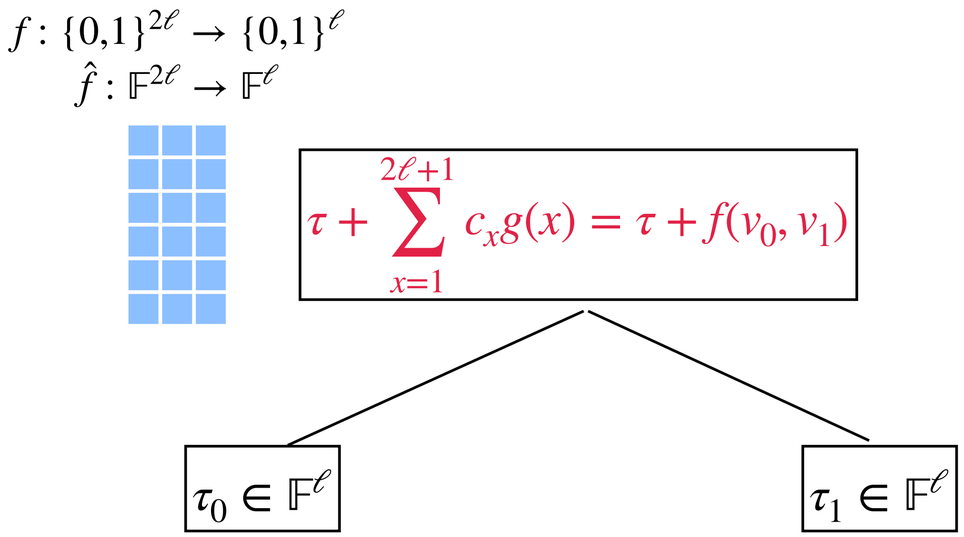
Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls → add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls → subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ



Details

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls → add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls → subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ



Efficiency Analysis

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls $\rightarrow$ add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls $\rightarrow$ subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ

Metrics

- $\mathscr{R} = \mathbb{F}^{\ell}$

Efficiency Analysis

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls $\rightarrow$ add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls $\rightarrow$ subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ

Metrics

- $\mathscr{R} = \mathbb{F}^{\ell}$
  - Final catalytic space
    $\log|\mathscr{R}| = \ell \log \ell$
    $= O(\log n \log\log n)$

Efficiency Analysis

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls $\rightarrow$ add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls $\rightarrow$ subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ

Metrics

- $\mathscr{R} = \mathbb{F}^{\ell}$
  - Final catalytic space
    $\log|\mathscr{R}| = \ell \log \ell$
    $= O(\log n \log\log n)$
- Free space: $\log \ell$

Efficiency Analysis

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls $\rightarrow$ add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls $\rightarrow$ subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ

Metrics

- $\mathscr{R} = \mathbb{F}^{\ell}$
  - Final catalytic space
    $\log|\mathscr{R}| = \ell \log \ell$
    $= O(\log n \log\log n)$
- Free space: $\log \ell$
  - Final free space:
    $h \log \ell = O(\log n \log\log n)$

Efficiency Analysis

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls $\rightarrow$ add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls $\rightarrow$ subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ

Metrics

- $\mathscr{R} = \mathbb{F}^{\ell}$
  - Final catalytic space
    $\log|\mathscr{R}| = \ell \log \ell$
    $= O(\log n \log\log n)$
- Free space: $\log \ell$
  - Final free space:
    $h \log \ell = O(\log n \log\log n)$
- Oracle queries: $O(\ell)$

Efficiency Analysis

- Setup
  - Multilinear $\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
  - Line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$, $g = \hat{f}\Big|_L$
  - Lagrange interpolation: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$
- For each $\lambda = 1,2,\ldots,2\ell + 1$:
  - Update $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$
  - Two oracle calls $\rightarrow$ add v_0, v_1 into the input registers
  - Add $c_\lambda \hat{f}(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) = c_\lambda g(\lambda)$ into τ
  - Two oracle calls $\rightarrow$ subtract v_0, v_1 from the input registers
  - Reset (τ_0, τ_1) by dividing by λ

Metrics

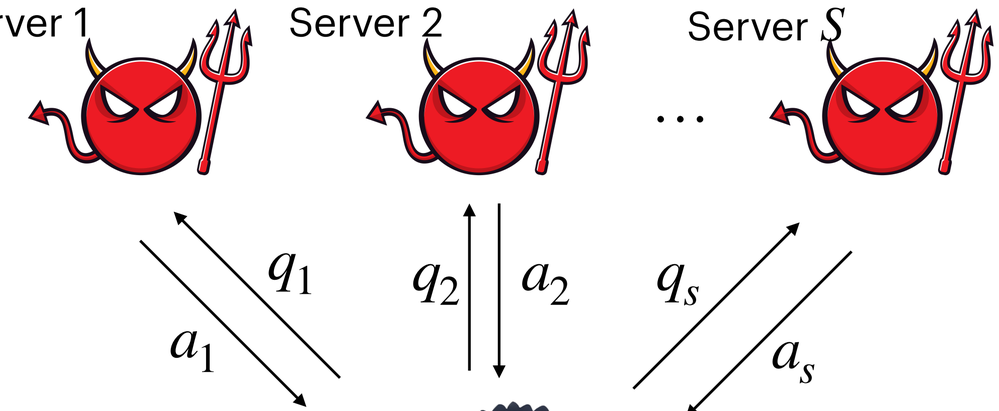
- $\mathscr{R} = \mathbb{F}^{\ell}$
  - Final catalytic space
    $\log|\mathscr{R}| = \ell \log \ell$
    $= O(\log n \log\log n)$
- Free space: $\log \ell$
  - Final free space:
    $h \log \ell = O(\log n \log\log n)$
- Oracle queries: $O(\ell)$
  - Final runtime: $\ell^h = n^{O(\log\log n)}$

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR ↔ Tree Evaluation
- Formal abstraction: matching vectors and decoding polynomials
- Conclusion

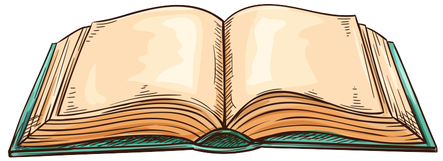
Informal Dictionary: PIR ↔ TreeEval

$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear



Client

$v \in \mathbb{F}^{\ell}$



PIR	TreeEval One-Level Gadget

Informal Dictionary: PIR ↔ TreeEval

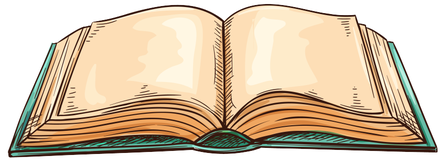
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$



PIR	TreeEval One-Level Gadget
DB	Truth table of f

Informal Dictionary: PIR ↔ TreeEval

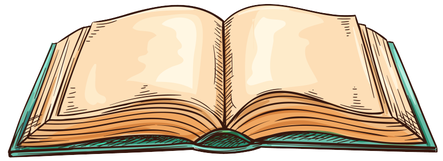
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$



PIR	TreeEval One-Level Gadget
DB	Truth table of f
Client's answer	f(v)

Informal Dictionary: PIR ↔ TreeEval

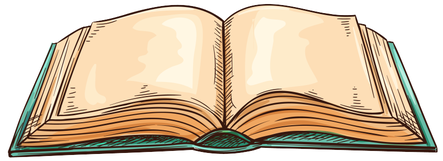
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$



PIR	TreeEval One-Level Gadget
DB	Truth table of f
Client's answer	f(v)
# of servers	# of oracle calls

Informal Dictionary: PIR ↔ TreeEval

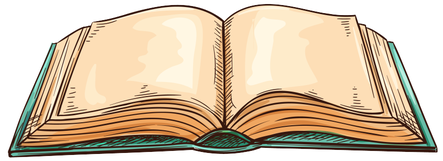
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$



PIR	TreeEval One-Level Gadget
DB	Truth table of f
Client's answer	f(v)
# of servers	# of oracle calls
Question	Written to catalytic tape by child oracle

$q_j = v + j\tau$ for Reed-Muller

Informal Dictionary: PIR ↔ TreeEval

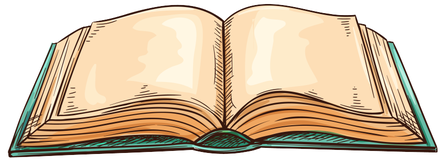
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

$v \in \mathbb{F}^{\ell}$



PIR	TreeEval One-Level Gadget	
DB	Truth table of f	
Client's answer	f(v)	
# of servers	# of oracle calls	
Question	Written to catalytic tape by child oracle	$q_j = v + j\tau$ for Reed-Muller
Server's answer	Written to catalytic tape by one-level gadget	$a_j = \hat{f}(v + j\tau)$ for Reed-Muller

Informal Dictionary: PIR ↔ TreeEval

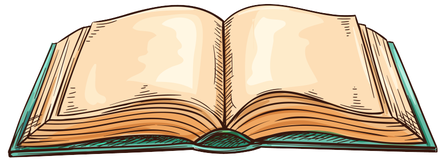
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

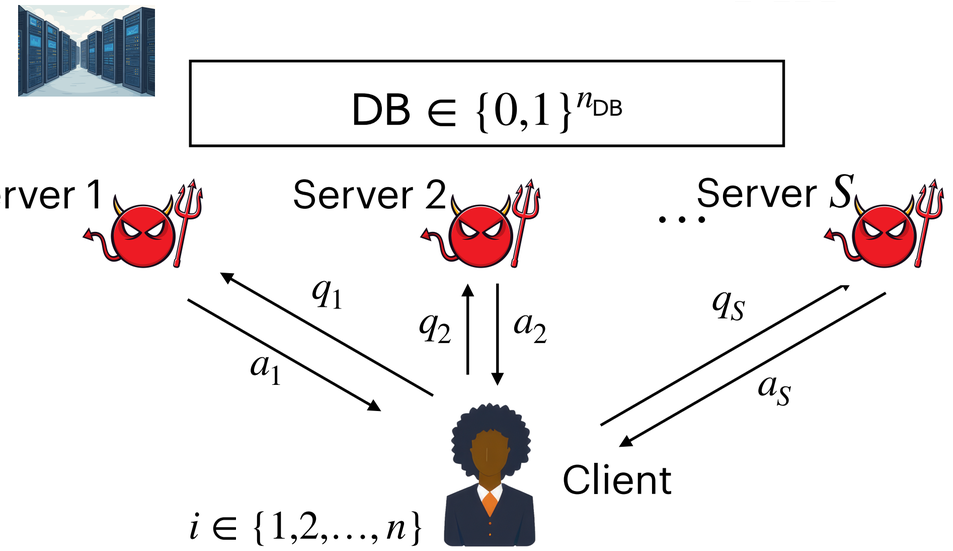
Client

$v \in \mathbb{F}^{\ell}$



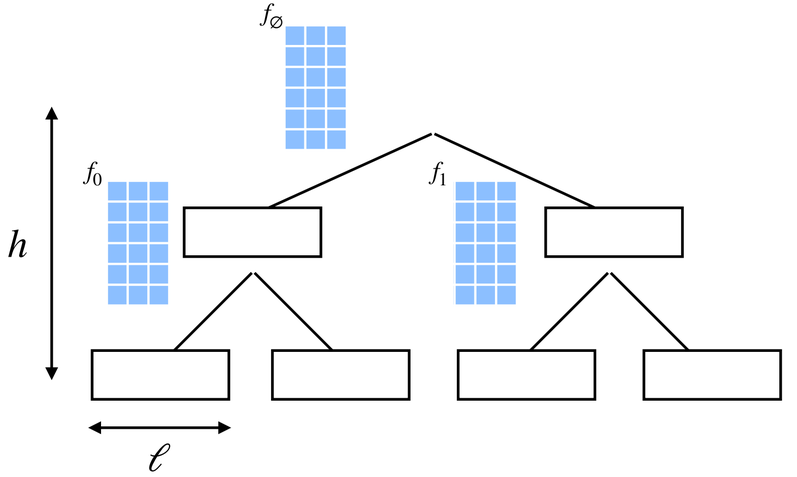
PIR	TreeEval One-Level Gadget	
DB	Truth table of f	
Client's answer	f(v)	
# of servers	# of oracle calls	
Question	Written to catalytic tape by child oracle	$q_j = v + j\tau$ for Reed-Muller
Server's answer	Written to catalytic tape by one-level gadget	$a_j = \hat{f}(v + j\tau)$ for Reed-Muller
CC	Catalytic tape size	

PIR



- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

TreeEval



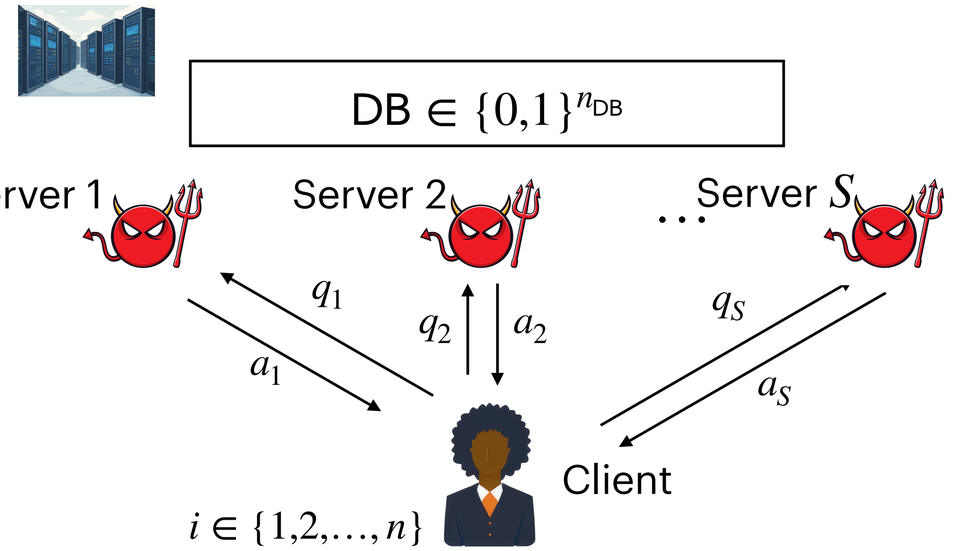
n = total input length = $\mathrm{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:**

  $O(h \log S + \ell + \log \mathrm{CC})$

- **Runtime:** $\mathrm{poly}(S^h, 2^\ell)$

PIR

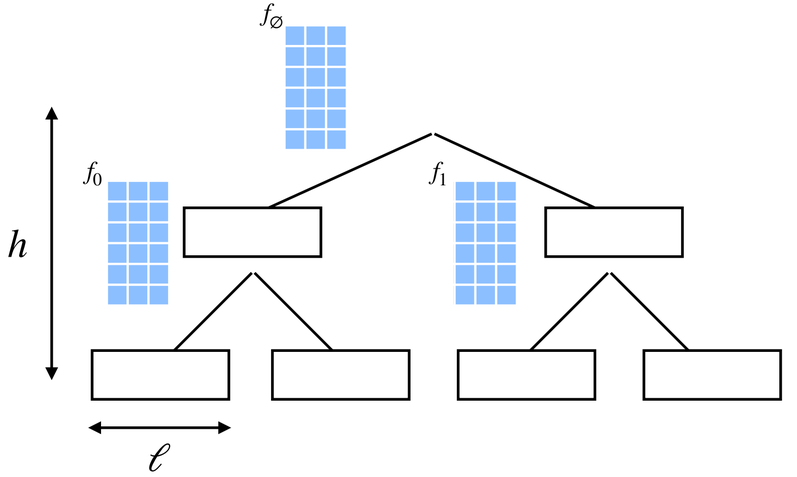


- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$

TreeEval

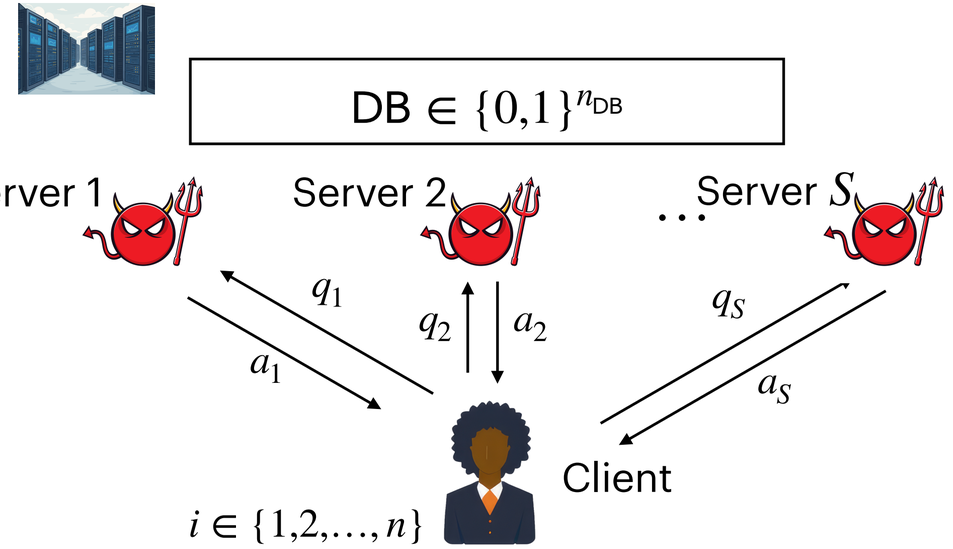


n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:**

  $O(h \log S + \ell + \log \mathsf{CC})$
- **Runtime:** $\mathsf{poly}(S^h, 2^\ell)$

PIR

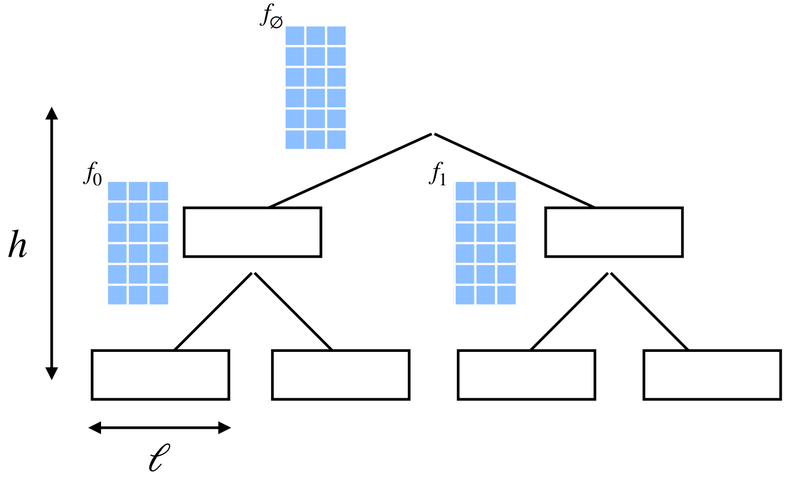


- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$ —*Super-constant*→
  - Super-logarithmic free space, super-polynomial runtime 😭
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$

TreeEval

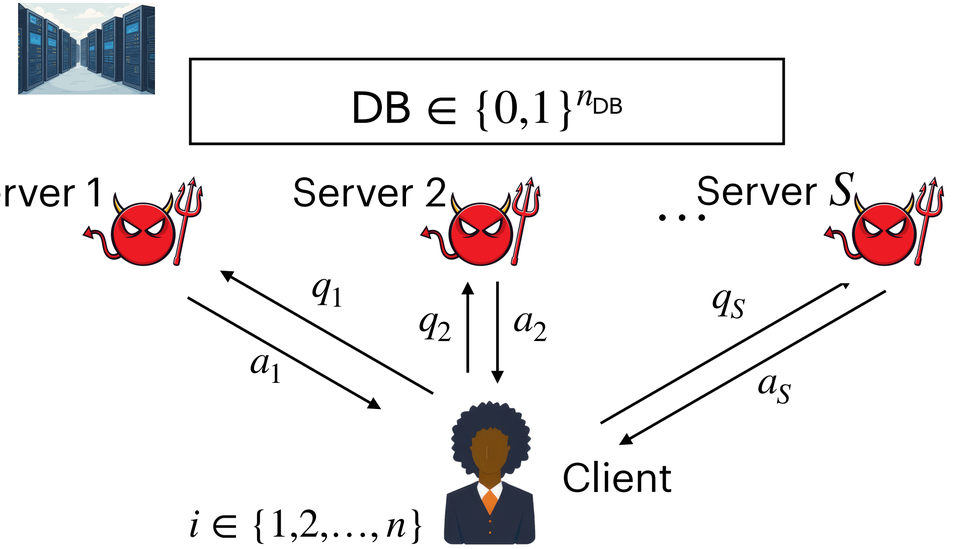


n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:**

  $O(h \log S + \ell + \log \mathsf{CC})$
- **Runtime:** $\mathsf{poly}(S^h, 2^\ell)$

PIR



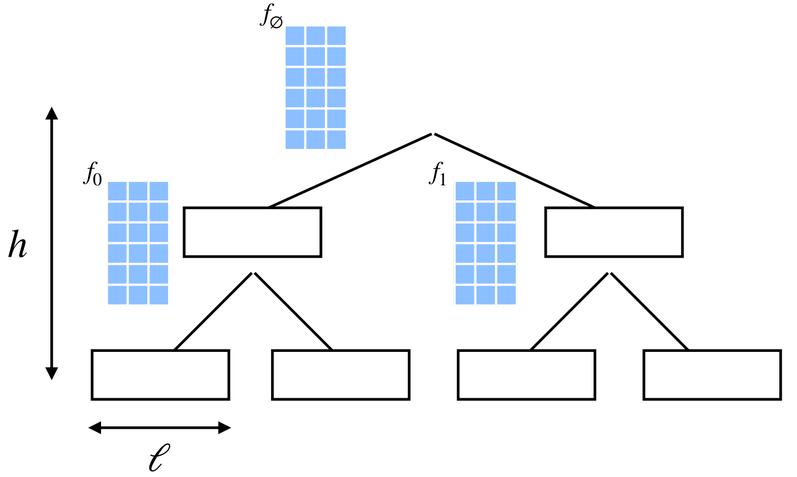
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$ —*Super-constant*→
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$ —*Almost log!*→

- Super-logarithmic free space, super-polynomial runtime 😭
- $O(\log n \log\log n)$ catalytic space 😄

TreeEval

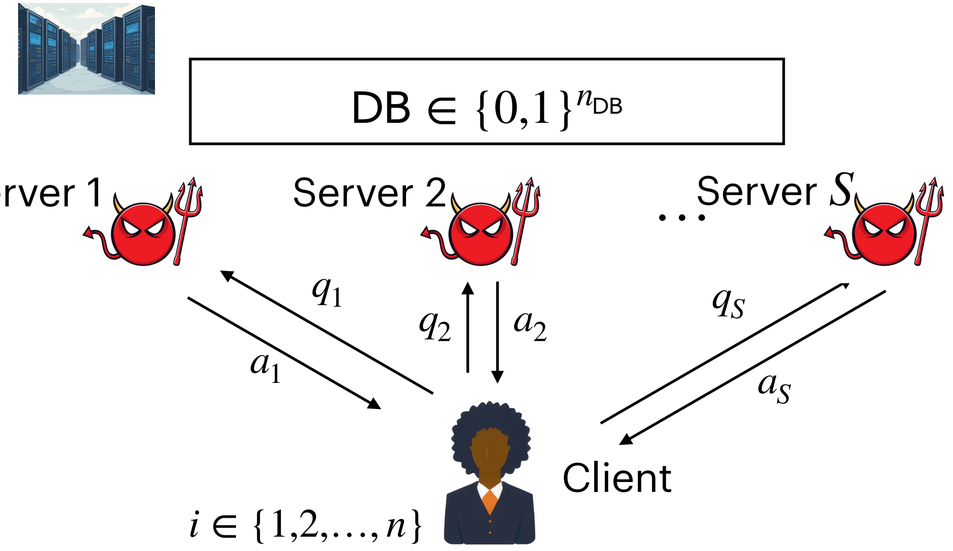


n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:**

  $O(h \log S + \ell + \log \mathsf{CC})$
- **Runtime:** $\mathsf{poly}(S^h, 2^\ell)$

PIR



- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

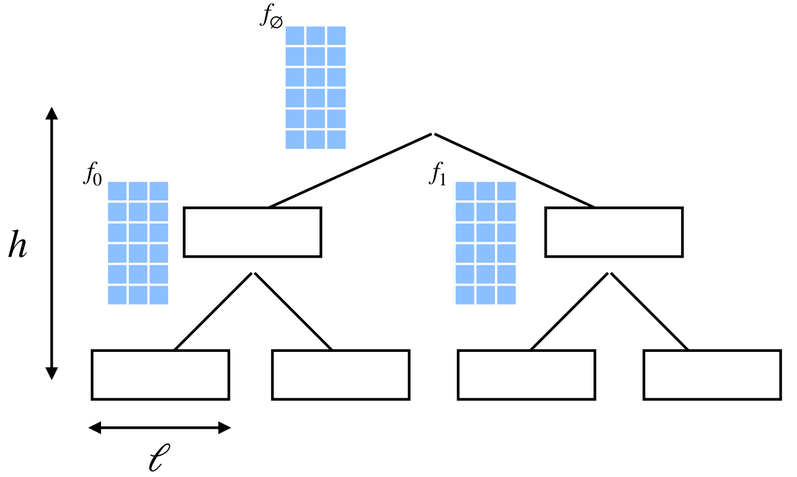
Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$ —*Super-constant*→
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$ —*Almost log!*→

- Super-logarithmic free space, super-polynomial runtime 😭
- $O(\log n \log\log n)$ catalytic space 😄

Matching Vector PIR → Our Algorithm

TreeEval



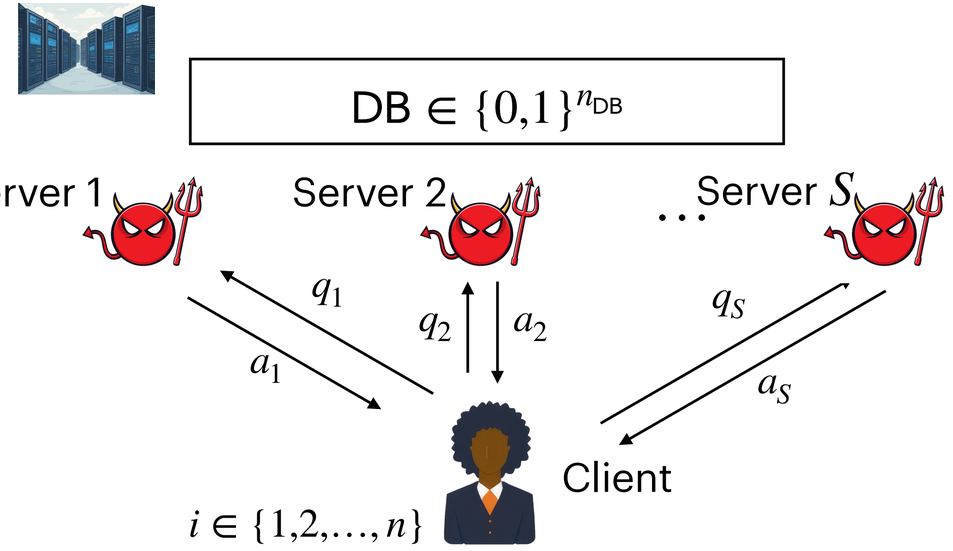
n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:**

  $O(h \log S + \ell + \log \mathsf{CC})$

- **Runtime:** $\mathsf{poly}(S^h, 2^\ell)$

PIR



- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

Reed-Muller PIR → Cook-Mertz

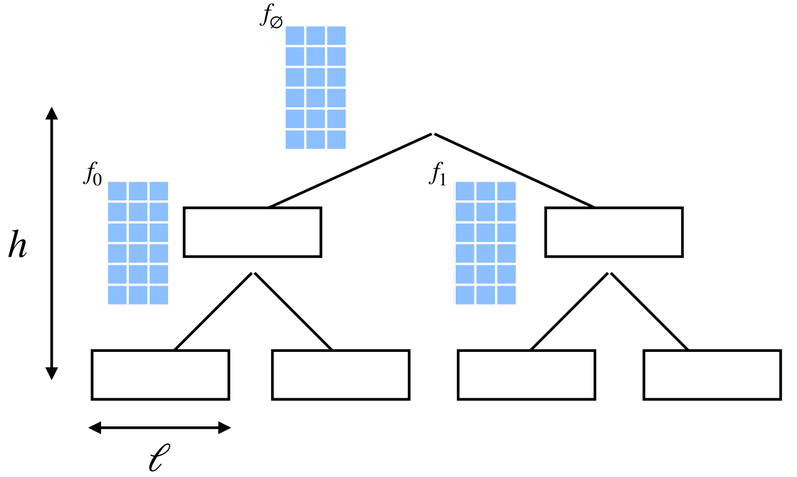
- $S = O(\log n_{\mathsf{DB}})$ —*Super-constant*→
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$ —*Almost log!*→

- Super-logarithmic free space, super-polynomial runtime 😭
- $O(\log n \log\log n)$ catalytic space 😄

Matching Vector PIR → Our Algorithm

- $S = O(1)$
- $\mathsf{CC} = 2^{\log^{\epsilon} n_{\mathsf{DB}}}$

TreeEval



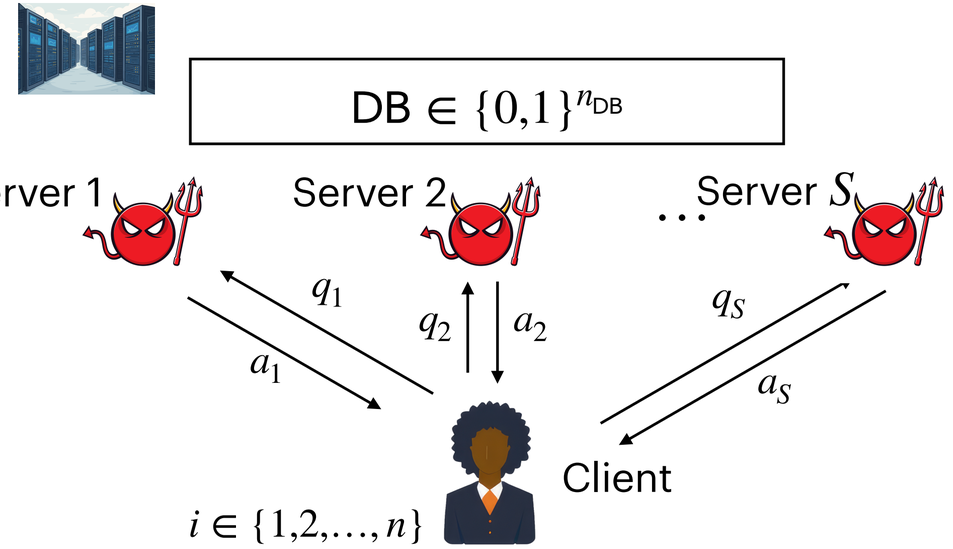
n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:**

  $O(h \log S + \ell + \log \mathsf{CC})$

- **Runtime:** $\mathsf{poly}(S^h, 2^{\ell})$

PIR



- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$ — *Super-constant* →
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$ — *Almost log!* →

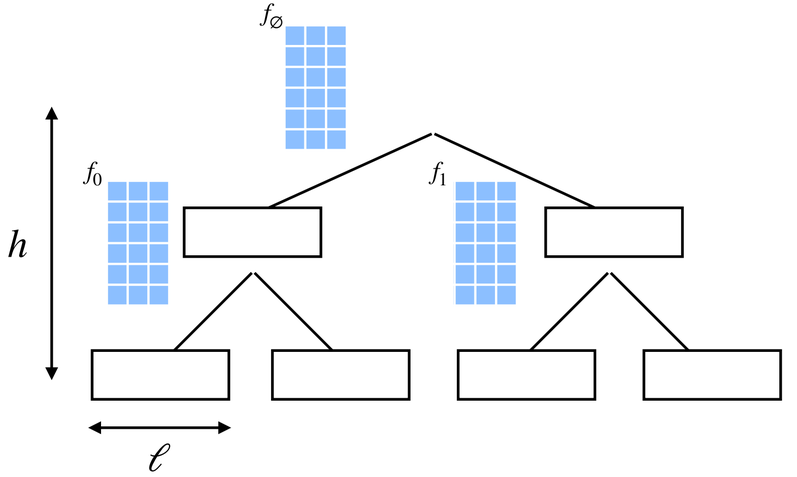
- Super-logarithmic free space, super-polynomial runtime 😭
- $O(\log n \log\log n)$ catalytic space 😄

Matching Vector PIR → Our Algorithm

- $S = O(1)$ — *Constant!* →
- $\mathsf{CC} = 2^{\log^{\epsilon} n_{\mathsf{DB}}}$

- Logarithmic free space, polynomial runtime 😄

TreeEval



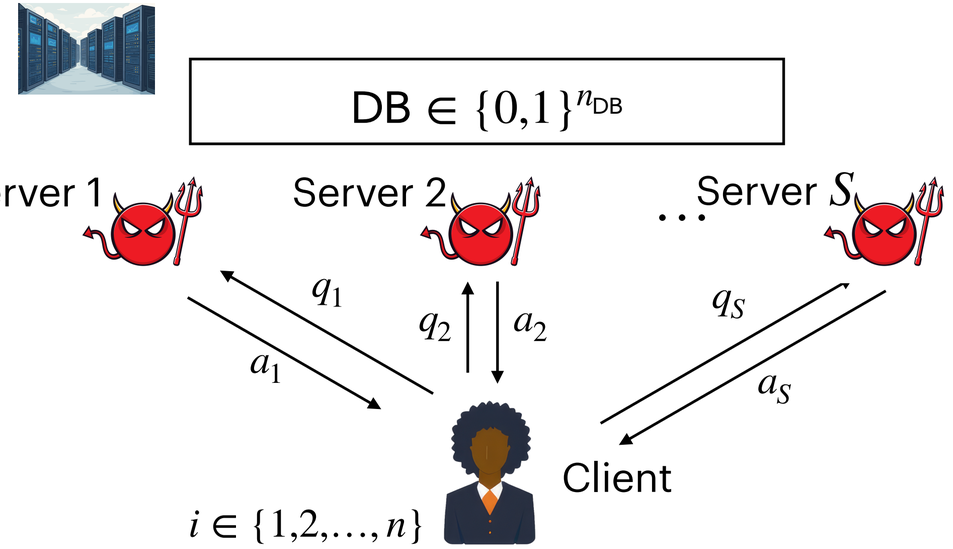
n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:**

$$O(h \log S + \ell + \log \mathsf{CC})$$

- **Runtime:** $\mathsf{poly}(S^h, 2^{\ell})$

PIR



- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

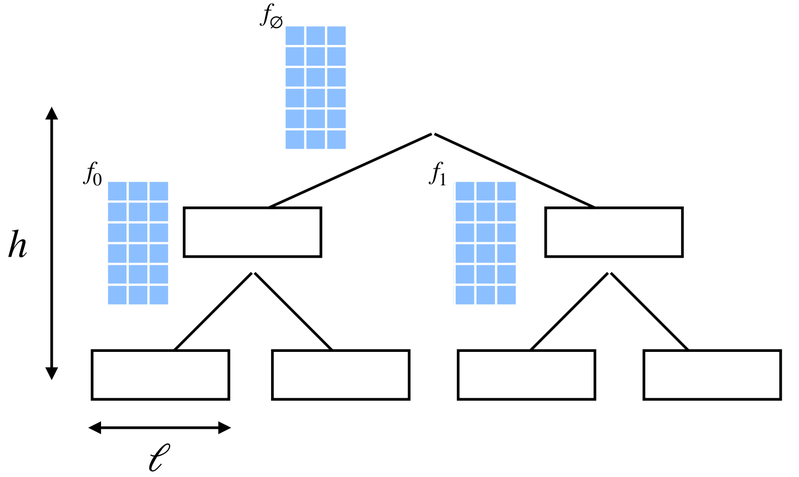
Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$ — *Super-constant* → Super-logarithmic free space, super-polynomial runtime 😭
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$ — *Almost log!* → $O(\log n \log\log n)$ catalytic space 😄

Matching Vector PIR → Our Algorithm

- $S = O(1)$ — *Constant!* → Logarithmic free space, polynomial runtime 😄
- $\mathsf{CC} = 2^{\log^{\epsilon} n_{\mathsf{DB}}}$ — *Sub-polynomial* → $2^{\log^{\epsilon} n}$ catalytic space 😭

TreeEval



n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **Catalytic space:** CC
- **Free space:** $O(h \log S + \ell + \log \mathsf{CC})$
- **Runtime:** $\mathsf{poly}(S^h, 2^{\ell})$

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR ↔ Tree Evaluation
- Formal abstraction: matching vectors and decoding polynomials
- Conclusion

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{m} \Leftrightarrow i = j$

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{m} \Leftrightarrow i = j$
  - m is constant
  - n is the given DB size, and d will ultimately govern CC (so want to minimise it)

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{m} \Leftrightarrow i = j$
  - m is constant
  - n is the given DB size, and d will ultimately govern CC (so want to minimise it)
- Rank argument: if m is prime, then $d \geq n^{1/m}$

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{m} \Leftrightarrow i = j$
  - m is constant
  - n is the given DB size, and d will ultimately govern CC (so want to minimise it)
- Rank argument: if m is prime, then $d \geq n^{1/m}$
- *[Barrington-Beigel-Rudich '92, Grolmusz '00]* If $m = pq$ for distinct primes p, q, then there exists a construction attaining $d \leq 2^{\tilde{O}(\sqrt{\log n})}$ 🤯

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{m} \Leftrightarrow i = j$
  - m is constant
  - n is the given DB size, and d will ultimately govern CC (so want to minimise it)
- Rank argument: if m is prime, then $d \geq n^{1/m}$
- *[Barrington-Beigel-Rudich '92, Grolmusz '00]* If $m = pq$ for distinct primes p, q, then there exists a construction attaining $d \leq 2^{\tilde{O}(\sqrt{\log n})}$ 🤯
  - Even more: $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p}$, and similarly mod q

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{m} \Leftrightarrow i = j$
  - m is constant
  - n is the given DB size, and d will ultimately govern CC (so want to minimise it)
- Rank argument: if m is prime, then $d \geq n^{1/m}$
- *[Barrington-Beigel-Rudich '92, Grolmusz '00]* If $m = pq$ for distinct primes p, q, then there exists a construction attaining $d \leq 2^{\tilde{O}(\sqrt{\log n})}$ 🤯
  - Even more: $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p}$, and similarly mod q
  - Define the canonical set $S_m^* = \{x \in \mathbb{Z}_{pq} : x \equiv 0,1 \pmod{p \text{ and } q}\}$

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{m} \Leftrightarrow i = j$
  - m is constant
  - n is the given DB size, and d will ultimately govern CC (so want to minimise it)
- Rank argument: if m is prime, then $d \geq n^{1/m}$
- *[Barrington-Beigel-Rudich '92, Grolmusz '00]* If $m = pq$ for distinct primes p, q, then there exists a construction attaining $d \leq 2^{\tilde{O}(\sqrt{\log n})}$ 🤯
  - Even more: $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p}$, and similarly mod q
  - Define the canonical set $S_m^* = \{x \in \mathbb{Z}_{pq} : x \equiv 0,1 \pmod{p \text{ and } q}\}$
  - This construction is referred to as an S_m^*-matching vector family

S-Decoding Polynomials

- Work over a finite field $\mathbb{F}$ with mth root of unity γ

S-Decoding Polynomials

- Work over a finite field $\mathbb{F}$ with mth root of unity γ
- Let $S \subseteq \mathbb{Z}_m$ be any set containing 0 (not necessarily S_m^*)

S-Decoding Polynomials

- Work over a finite field $\mathbb{F}$ with mth root of unity γ
- Let $S \subseteq \mathbb{Z}_m$ be any set containing 0 (not necessarily S_m^*)
- Criteria for $P \in \mathbb{F}[Z]$ to be an S-decoding polynomial:

S-Decoding Polynomials

- Work over a finite field $\mathbb{F}$ with mth root of unity γ
- Let $S \subseteq \mathbb{Z}_m$ be any set containing 0 (not necessarily S_m^*)
- Criteria for $P \in \mathbb{F}[Z]$ to be an S-decoding polynomial:
  - For all $j \in S \backslash \{0\}$, we have $P(\gamma^j) = 0$

S-Decoding Polynomials

- Work over a finite field $\mathbb{F}$ with mth root of unity γ
- Let $S \subseteq \mathbb{Z}_m$ be any set containing 0 (not necessarily S_m^*)
- Criteria for $P \in \mathbb{F}[Z]$ to be an S-decoding polynomial:
  - For all $j \in S \backslash \{0\}$, we have $P(\gamma^j) = 0$
  - For $j = 0$, $P(\gamma^j) = P(1) = 1$

S-Decoding Polynomials

- Work over a finite field $\mathbb{F}$ with mth root of unity γ
- Let $S \subseteq \mathbb{Z}_m$ be any set containing 0 (not necessarily S_m^*)
- Criteria for $P \in \mathbb{F}[Z]$ to be an S-decoding polynomial:
  - For all $j \in S \backslash \{0\}$, we have $P(\gamma^j) = 0$
  - For $j = 0$, $P(\gamma^j) = P(1) = 1$
- Main metric for P: sparsity (not degree!)

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: S-decoding polynomial of sparsity r

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: S-decoding polynomial of sparsity r
- Then there exists r-server PIR with communication $\approx O(d)$

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: S-decoding polynomial of sparsity r
- Then there exists r-server PIR with communication $\approx O(d)$

Building TreeEval

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: S-decoding polynomial of sparsity r
- Then there exists r-server PIR with communication $\approx O(d)$

Building TreeEval

- Assume $s_1 + s_2 = 0 \Rightarrow s_1 = s_2 = 0$ for $s_1, s_2 \in S$

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: S-decoding polynomial of sparsity r
- Then there exists r-server PIR with communication $\approx O(d)$

Building TreeEval

- Assume $s_1 + s_2 = 0 \Rightarrow s_1 = s_2 = 0$ for $s_1, s_2 \in S$
- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: S-decoding polynomial of sparsity r
- Then there exists r-server PIR with communication $\approx O(d)$

Building TreeEval

- Assume $s_1 + s_2 = 0 \Rightarrow s_1 = s_2 = 0$ for $s_1, s_2 \in S$
- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: $(S + S)$-decoding polynomial of sparsity r

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Building PIR

- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: S-decoding polynomial of sparsity r
- Then there exists r-server PIR with communication $\approx O(d)$

Building TreeEval

- Assume $s_1 + s_2 = 0 \Rightarrow s_1 = s_2 = 0$ for $s_1, s_2 \in S$
- Ingredient 1: S-matching vector family of size n over $\mathbb{Z}_m^d$
- Ingredient 2: $(S + S)$-decoding polynomial of sparsity r
- Then $\mathsf{TreeEval}_{h,\ell} \in \mathsf{CatSpaceTime}(d, h \log r + \ell, \mathsf{poly}(r^h \cdot 2^\ell))$

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Fact that I won't prove today: Reed-Muller PIR and Cook-Mertz can also be instantiated as special cases of this abstraction!

MV + Decoding Polynomials → PIR and TreeEval

(minor technical caveats)

Fact that I won't prove today: Reed-Muller PIR and Cook-Mertz can also be instantiated as special cases of this abstraction!

This is a clean unifying framework for Cook-Mertz's and our results, and opens clear pathways for future improvements.

Roadmap

- Tree evaluation: background and results
- Reed-Muller PIR and the Cook-Mertz Algorithm
- Informal dictionary for PIR ↔ Tree Evaluation
- Formal abstraction: matching vectors and decoding polynomials
- Conclusion

Matching-Vector PIR: A History

Goal: PIR on a database of size n with $O(1)$ servers

Construction	# of servers	CC	Uses matching vectors?	TLDR
Before ‘08	S	$n^{\tilde{O}(1/S)}$	No	Reed-Muller++
Yekhanin ‘08	3	$n^{O(1/\log\log n)}$	Yes	Enter matching vectors!
Efremenko ‘09	3	$2^{\tilde{O}(\sqrt{\log n})}$		Better decoding polynomial
	2^r	$2^{\tilde{O}((\log n)^{1/r})}$		Used Grolmusz’s MV construction
Dvir-Gopi ‘16	2^{r-1}	$2^{\tilde{O}((\log n)^{1/r})}$		Using derivatives
Ghasemi-Kopparty-Sudan ‘25	3	$2^{\tilde{O}((\log n)^{1/3})}$		Best of both worlds: Efremenko’s decoding polynomial and derivatives
*Gupte-**R**-ChatGPT ‘26*	r	$2^{\tilde{O}((\log n)^{1/r})}$		GKS’25 + optimal decoding polynomials

Open Questions on TreeEval

- Is TreeEval $\in$ L?

Open Questions on TreeEval

- Is TreeEval $\in$ L?
  - More modestly: TreeEval in simultaneous poly(n) time and $O(\log^{1.99} n)$ space?

Open Questions on TreeEval

- Is TreeEval $\in$ L?
  - More modestly: TreeEval in simultaneous poly(n) time and $O(\log^{1.99} n)$ space?
- Find better matching vector constructions or decoding polynomial constructions!

Open Questions on TreeEval

- Is TreeEval $\in$ L?
  - More modestly: TreeEval in simultaneous poly(n) time and $O(\log^{1.99} n)$ space?
- Find better matching vector constructions or decoding polynomial constructions!
  - Would push down our catalytic space, our main bottleneck

Open Questions on TreeEval

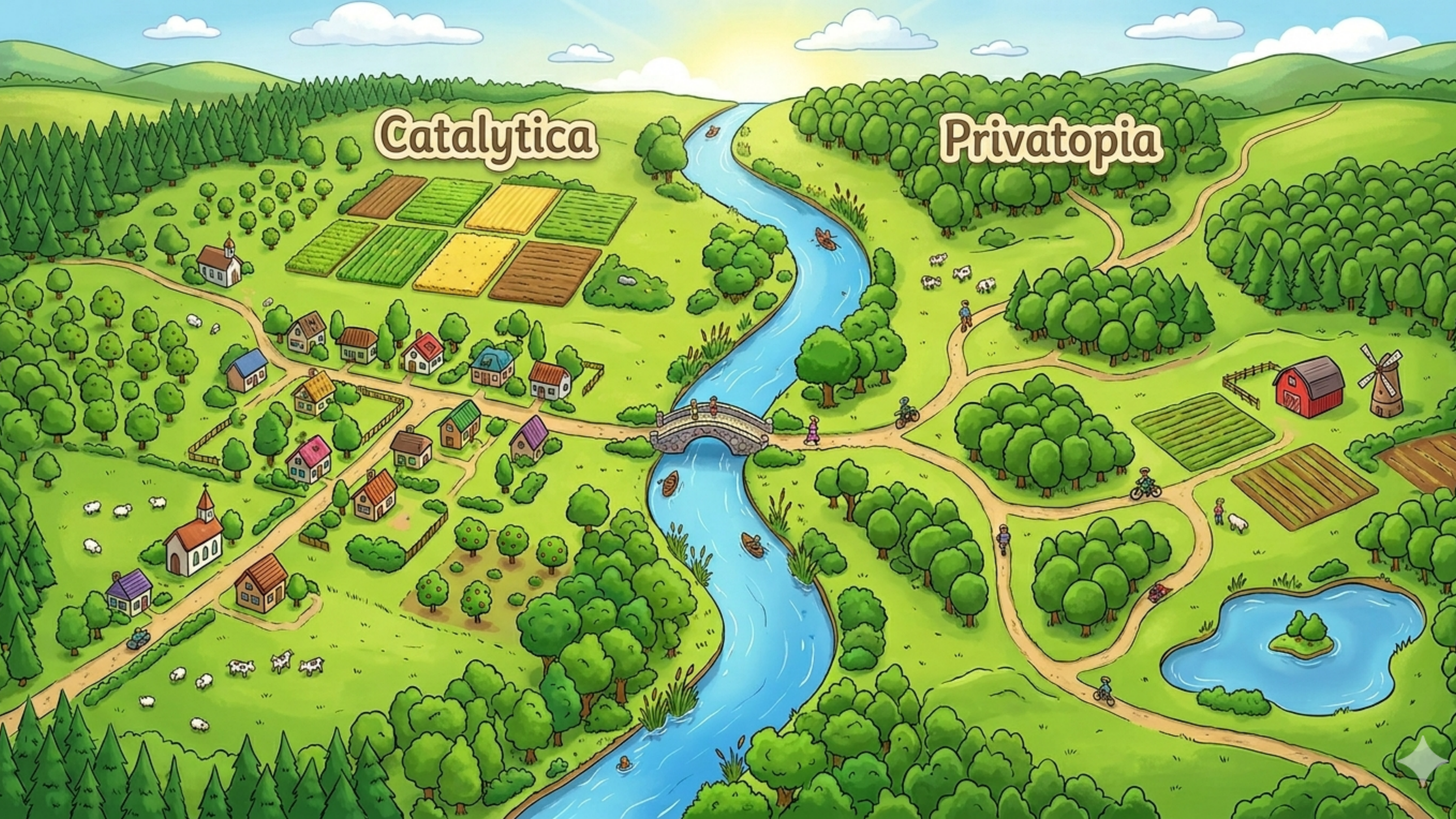
- Is TreeEval $\in$ L?
  - More modestly: TreeEval in simultaneous poly(n) time and $O(\log^{1.99} n)$ space?
- Find better matching vector constructions or decoding polynomial constructions!
  - Would push down our catalytic space, our main bottleneck
  - Best lower bound is $d \geq O(\log n \log \log n)$. If achievable:

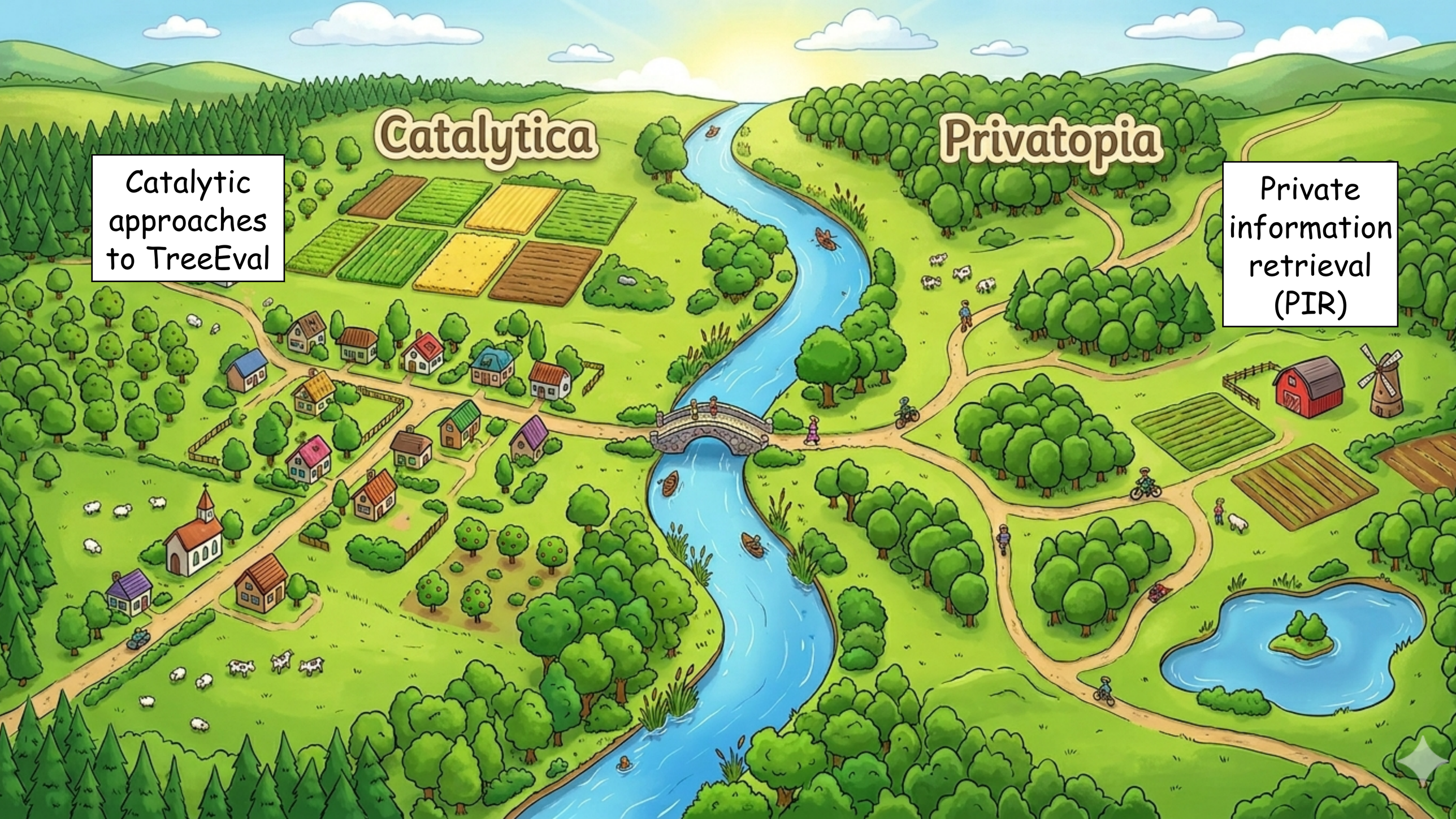
Open Questions on TreeEval

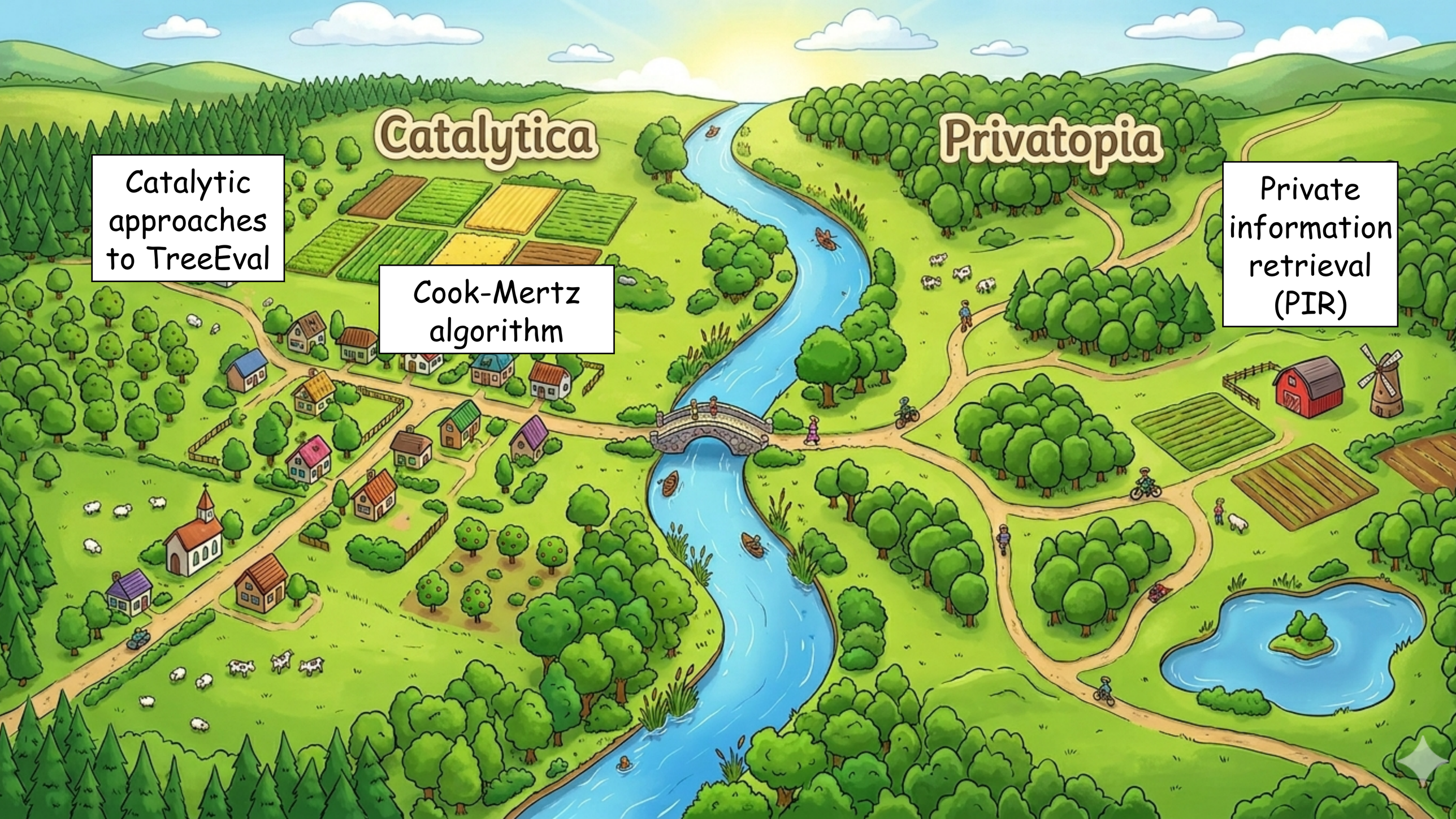
- Is TreeEval $\in$ L?
  - More modestly: TreeEval in simultaneous poly(n) time and $O(\log^{1.99} n)$ space?
- Find better matching vector constructions or decoding polynomial constructions!
  - Would push down our catalytic space, our main bottleneck
  - Best lower bound is $d \geq O(\log n \log\log n)$. If achievable:
    - $O(\log n \log\log n)$ catalytic space (matching Cook-Mertz), $O(\log n)$ free space, and poly(n) runtime

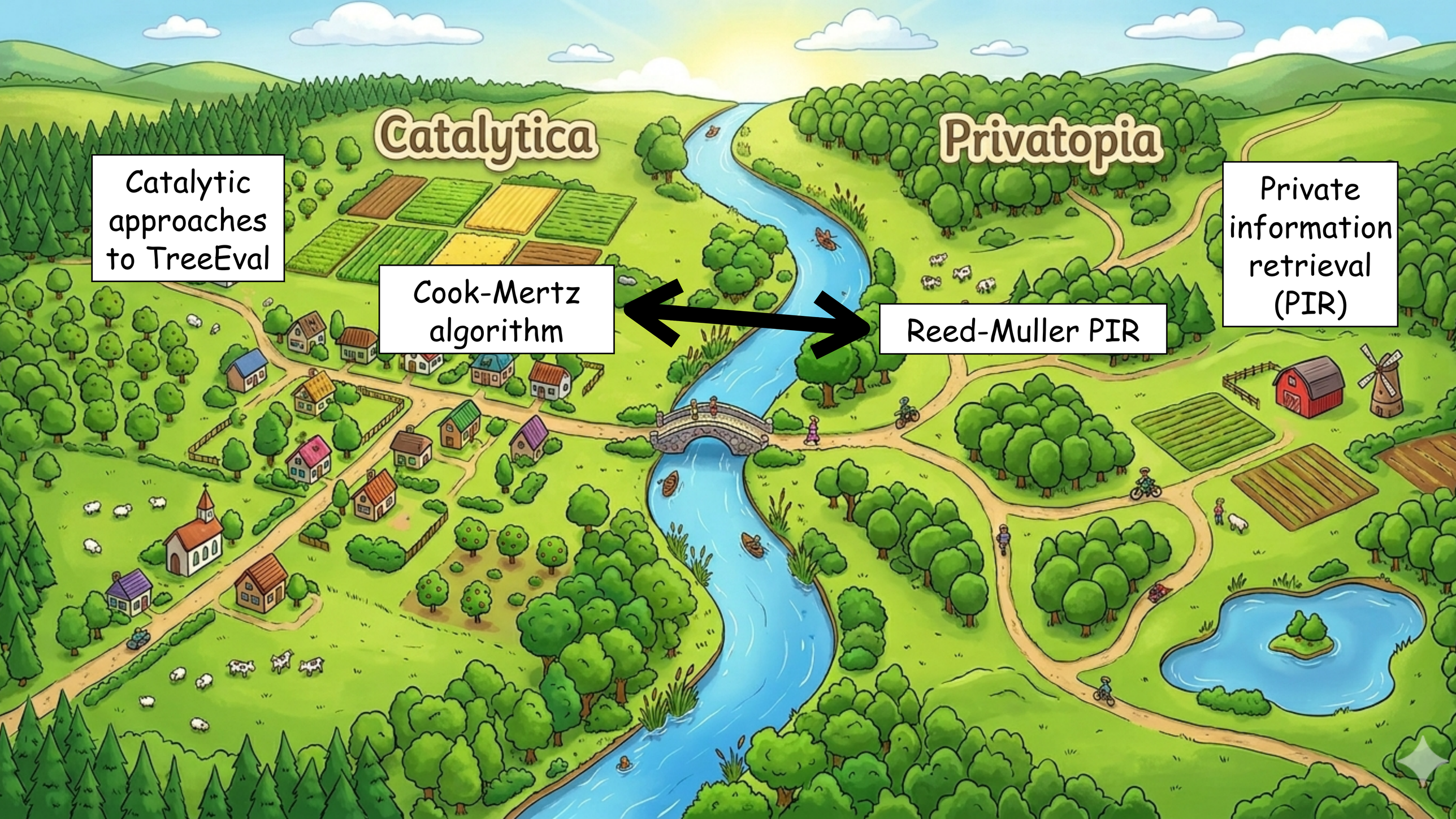
Open Questions on TreeEval

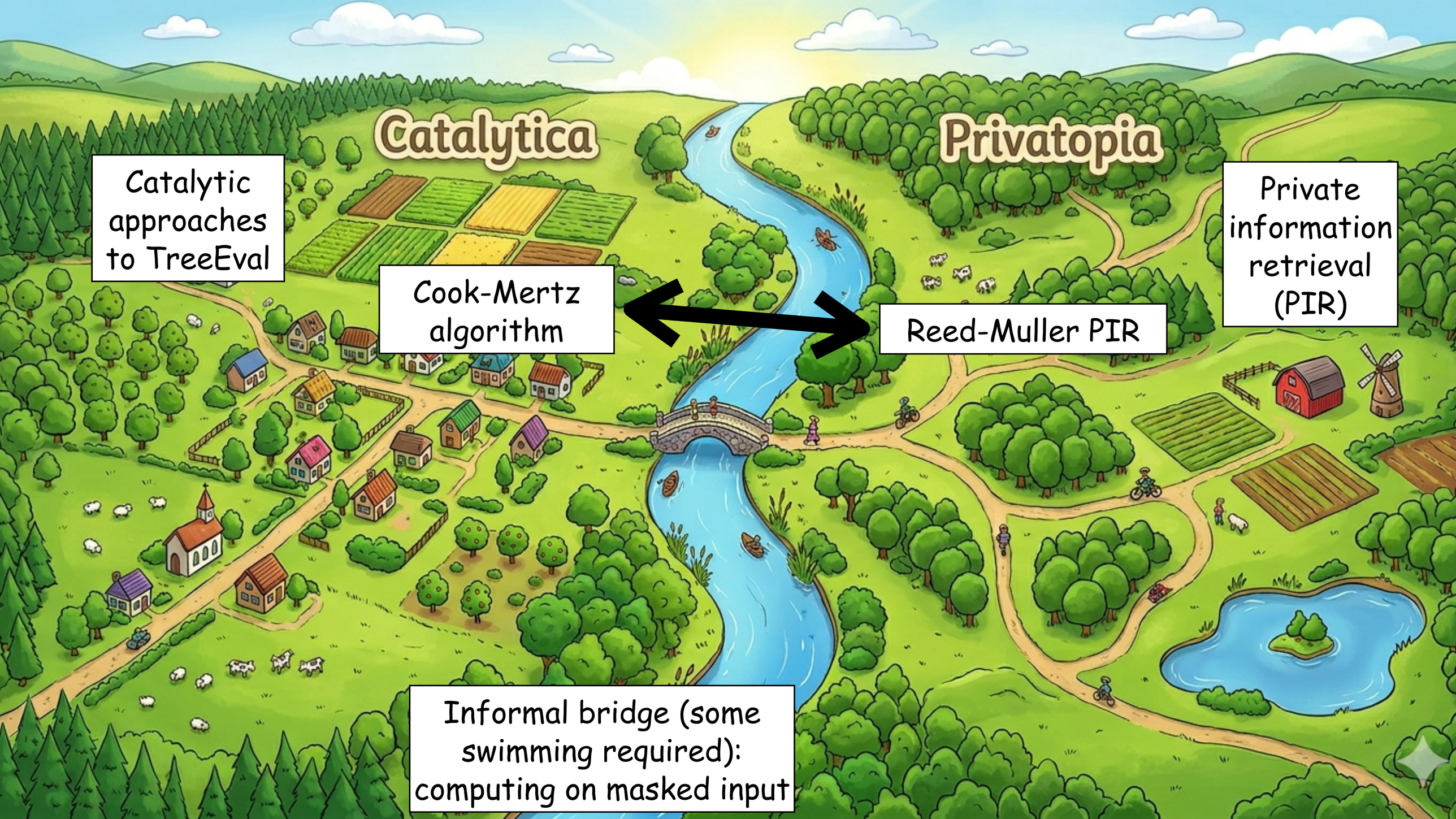
- Is TreeEval $\in$ L?
  - More modestly: TreeEval in simultaneous poly(n) time and $O(\log^{1.99} n)$ space?
- Find better matching vector constructions or decoding polynomial constructions!
  - Would push down our catalytic space, our main bottleneck
  - Best lower bound is $d \geq O(\log n \log\log n)$. If achievable:
    - $O(\log n \log\log n)$ catalytic space (matching Cook-Mertz), $O(\log n)$ free space, and poly(n) runtime
- Other applications of information-theoretic cryptography to catalytic computing?

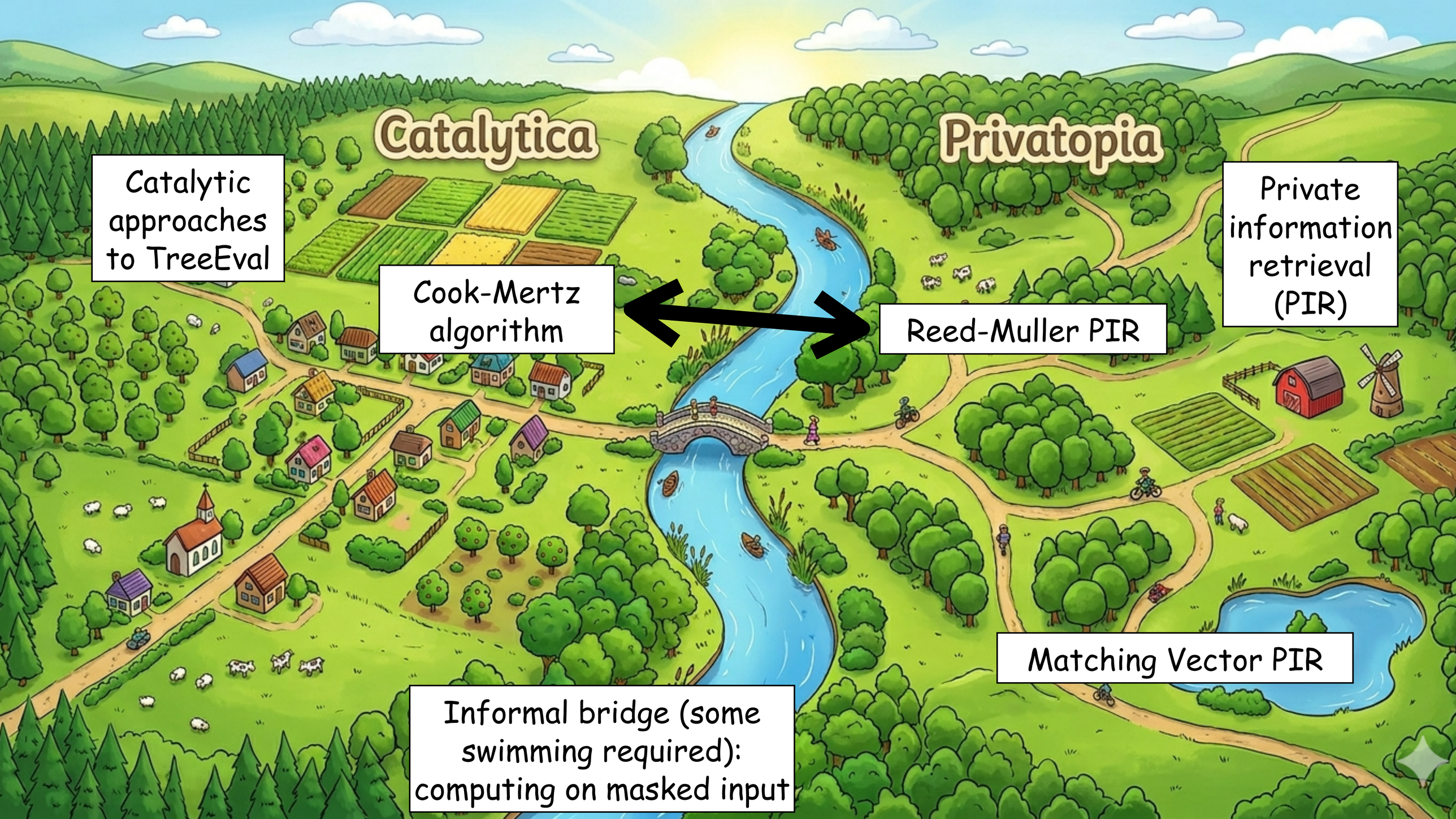

Catalytica
Privatopia

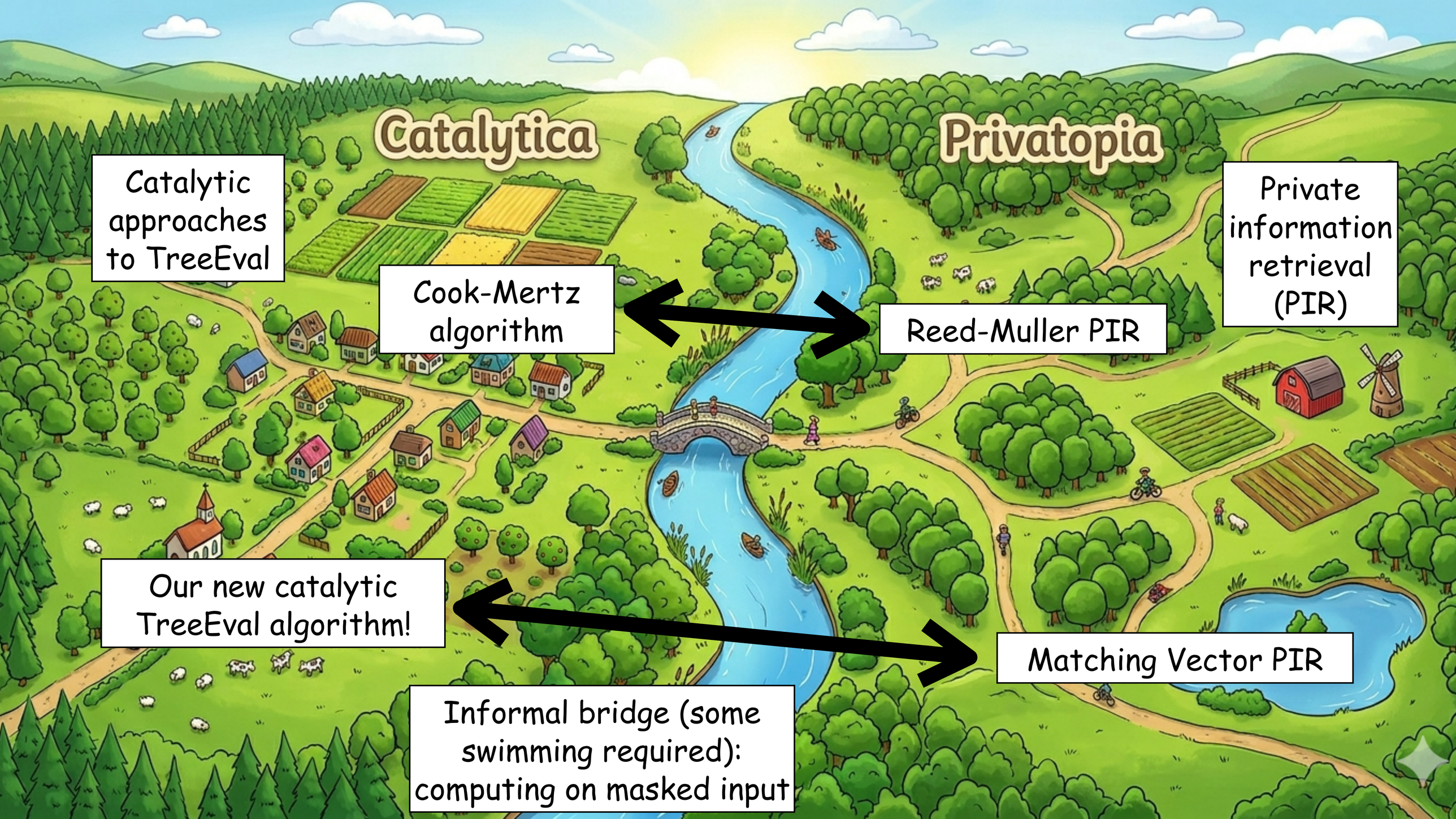

Catalytica
Privatopia
Catalytic approaches to TreeEval
Private information retrieval (PIR)

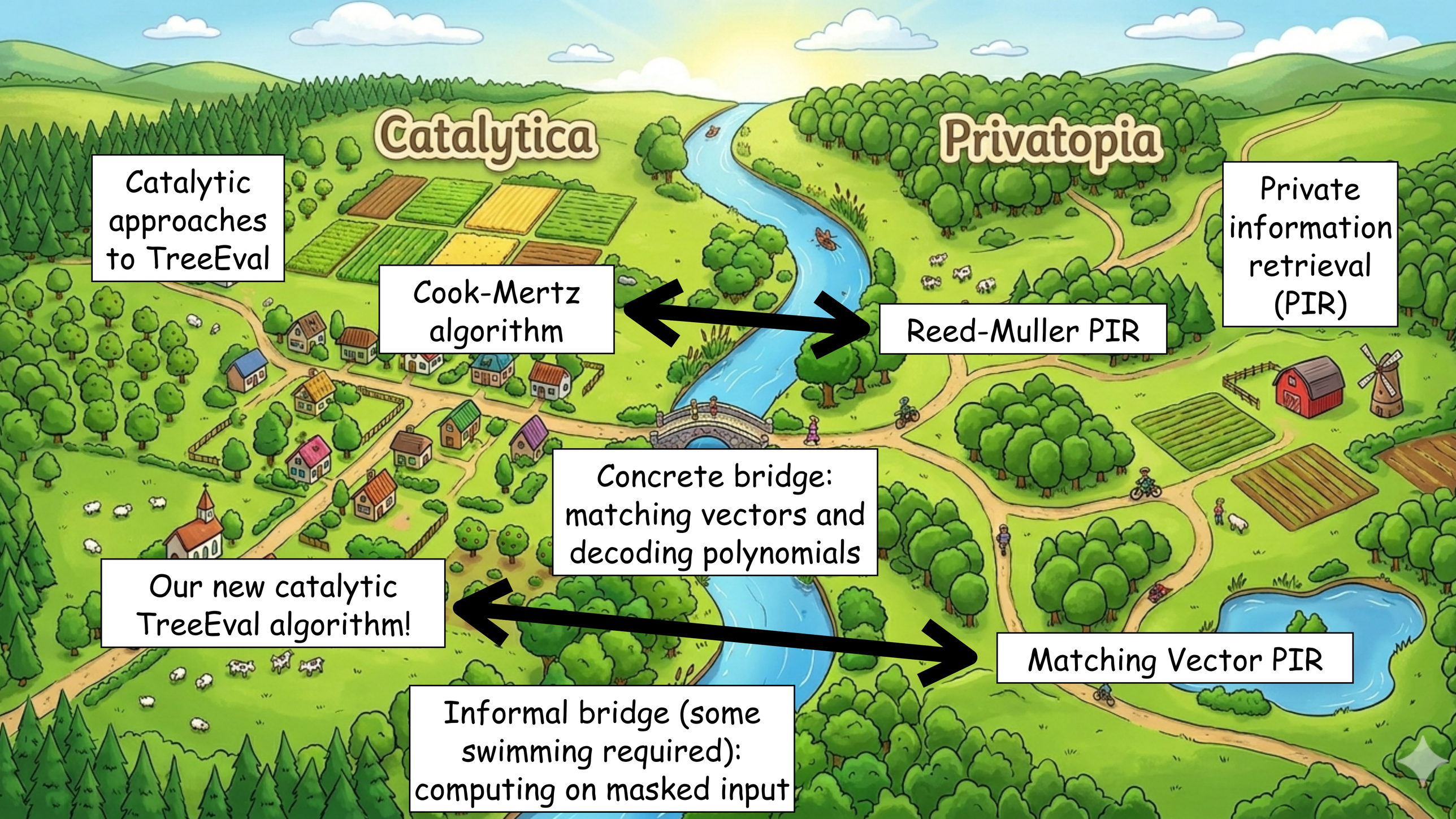

Catalytica
Privatopia
Catalytic approaches to TreeEval
Cook-Mertz algorithm
Private information retrieval (PIR)

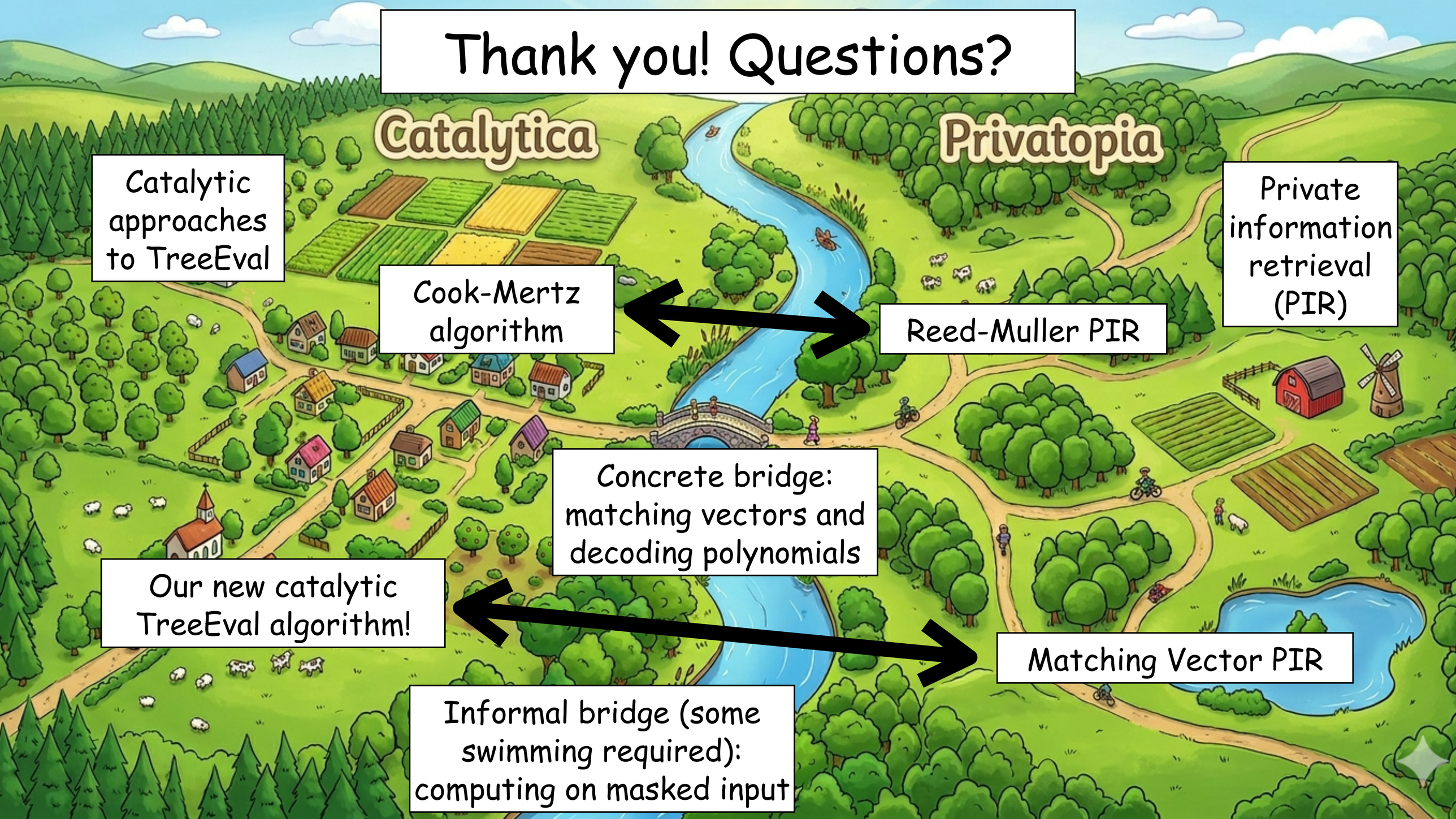

Catalytica
Privatopia
Catalytic approaches to TreeEval
Cook-Mertz algorithm
Reed-Muller PIR
Private information retrieval (PIR)


Catalytica
Privatopia
Catalytic approaches to TreeEval
Cook-Mertz algorithm
Reed-Muller PIR
Private information retrieval (PIR)
Informal bridge (some swimming required): computing on masked input


Catalytica
Privatopia
Catalytic approaches to TreeEval
Cook-Mertz algorithm
Reed-Muller PIR
Private information retrieval (PIR)
Matching Vector PIR
Informal bridge (some swimming required): computing on masked input


Catalytica
Privatopia
Catalytic approaches to TreeEval
Private information retrieval (PIR)
Cook-Mertz algorithm
Reed-Muller PIR
Our new catalytic TreeEval algorithm!
Matching Vector PIR
Informal bridge (some swimming required): computing on masked input


Catalytica
Privatopia
Catalytic approaches to TreeEval
Cook-Mertz algorithm
Reed-Muller PIR
Private information retrieval (PIR)
Concrete bridge: matching vectors and decoding polynomials
Our new catalytic TreeEval algorithm!
Matching Vector PIR
Informal bridge (some swimming required): computing on masked input

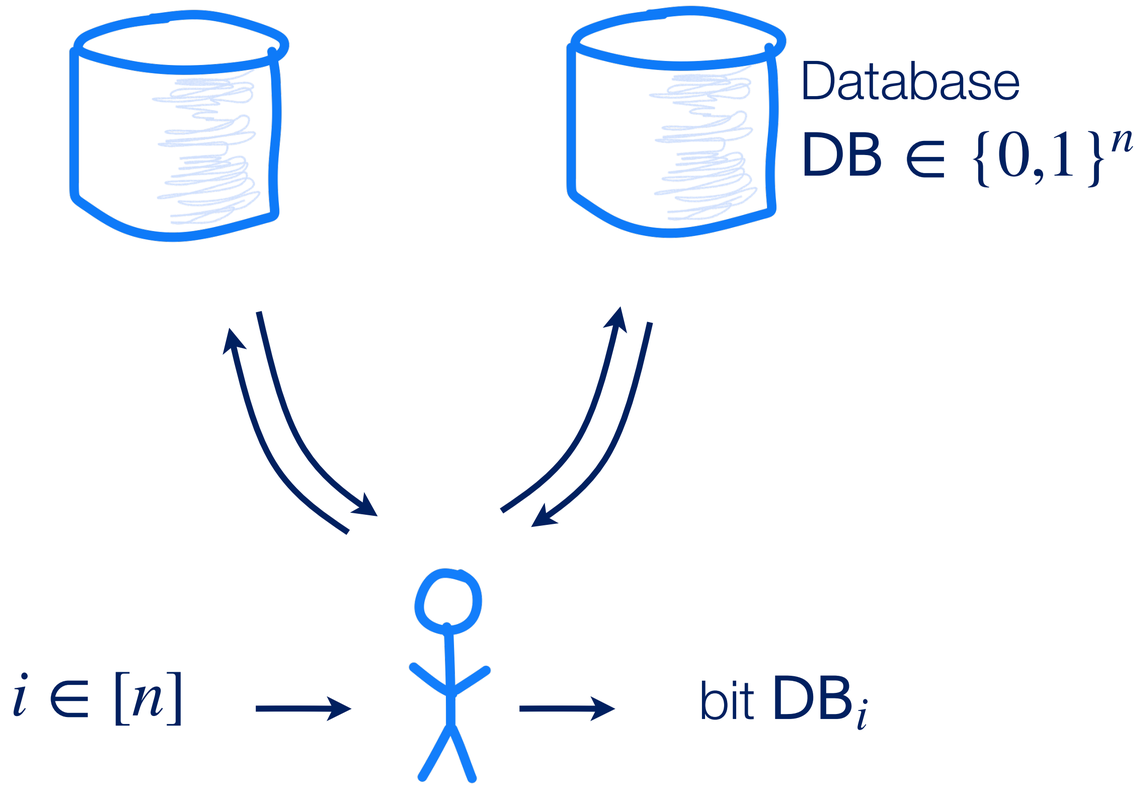

Thank you! Questions?
Catalytica
Privatopia
Catalytic approaches to TreeEval
Cook-Mertz algorithm
Reed-Muller PIR
Private information retrieval (PIR)
Concrete bridge: matching vectors and decoding polynomials
Our new catalytic TreeEval algorithm!
Matching Vector PIR
Informal bridge (some swimming required): computing on masked input

Part II: Two-Server Private Information Retrieval in Sublinear Time and Quasilinear Space

Or: Speeding Up PIR by Trading Time and Space (Eurocrypt 2026)

Alexandra Henzinger Seyoon Ragavan

Limitations of Low-Communication PIR

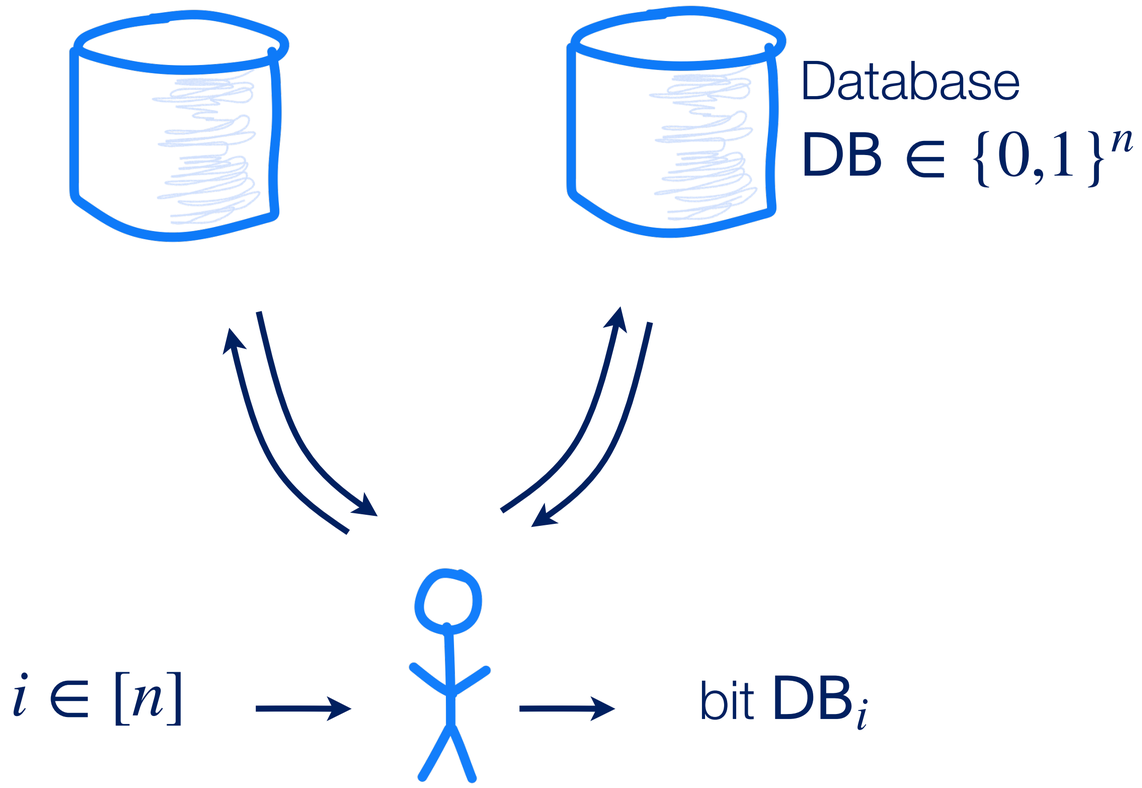


We have PIR constructions with very little communication:

- No privacy: $\log n + 1$
- Info-theoretic privacy: $n^{o(1)}$
- Comp. privacy: $O(\lambda \cdot \log n)$

[KO97,CMS99,DG16,BGI16]

Limitations of Low-Communication PIR



We have PIR constructions with very little communication:

- No privacy: $\log n + 1$
- Info-theoretic privacy: $n^{o(1)}$
- Comp. privacy: $O(\lambda \cdot \log n)$
  [KO97,CMS99,DG16,BGI16]

…but these require lots of server work:

- No privacy: $O(1)$ time
- With privacy: $\Omega(n)$ time
  [BIM00, PY22]

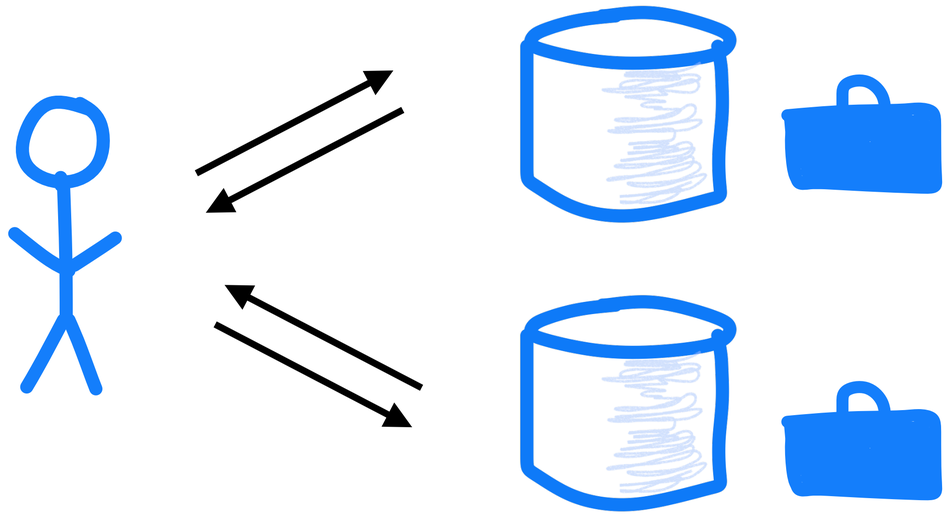
Solution: Trade Time for Space

[BIM'00,BIPW'17,CHR'17, HOWW'18,LMW'23,GLMDS'25]

Solution: Trade Time for Space

[BIM'00,BIPW'17,CHR'17, HOWW'18,LMW'23,GLMDS'25]

Referred to as "PIR with preprocessing" or "doubly-efficient PIR" (unkeyed)

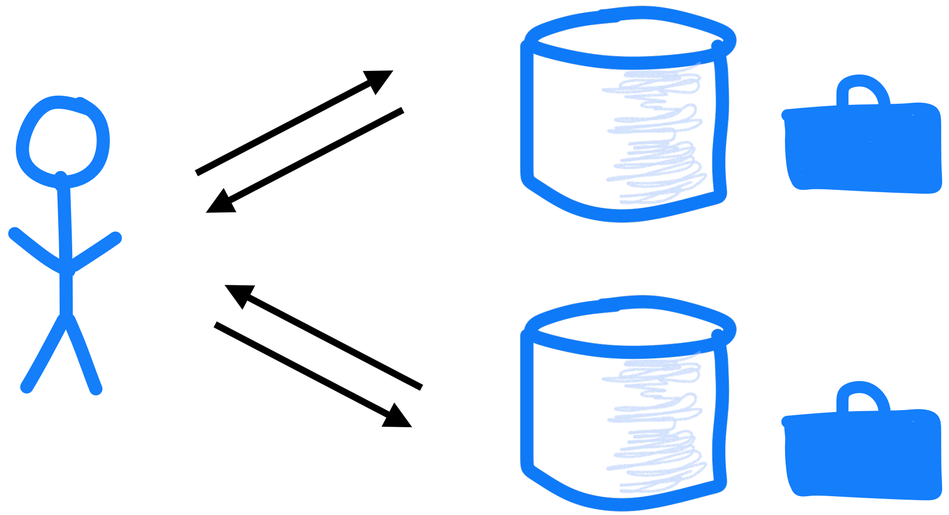


- Servers preprocess the DB into some data structure of size $\mathbf{poly}(n)$
- Hope: they can answer queries in time $\ll n$ using this data structure

Solution: Trade Time for Space

[BIM'00,BIPW'17,CHR'17, HOWW'18,LMW'23,GLMDS'25]

Referred to as "PIR with preprocessing" or "doubly-efficient PIR" (unkeyed)



- Servers preprocess the DB into some data structure of size $\mathbf{poly}(n)$
- Hope: they can answer queries in time $\ll n$ using this data structure

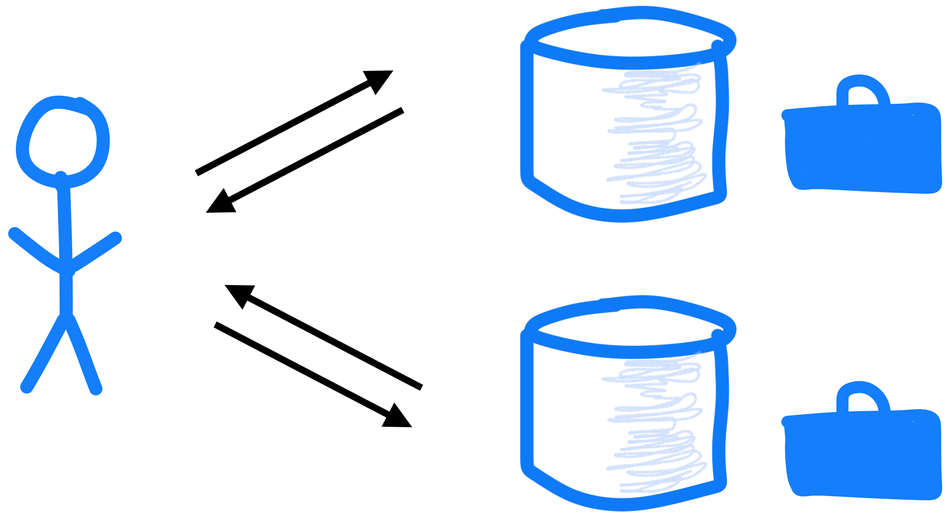
Efficiency Metrics

- Storage: size of the data structure

Solution: Trade Time for Space

[BIM'00,BIPW'17,CHR'17, HOWW'18,LMW'23,GLMDS'25]

Referred to as "PIR with preprocessing" or "doubly-efficient PIR" (unkeyed)



- Servers preprocess the DB into some data structure of size $\mathbf{poly}(n)$
- Hope: they can answer queries in time $\ll n$ using this data structure

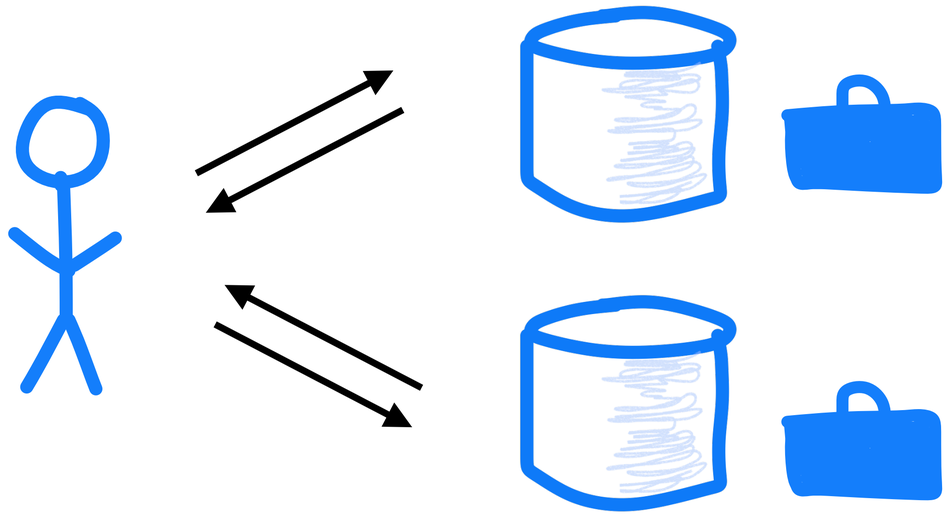
Efficiency Metrics

- Storage: size of the data structure
  - At most $\mathbf{poly}(n)$

Solution: Trade Time for Space

[BIM'00,BIPW'17,CHR'17, HOWW'18,LMW'23,GLMDS'25]

Referred to as "PIR with preprocessing" or "doubly-efficient PIR" (unkeyed)



- Servers preprocess the DB into some data structure of size $\mathbf{poly}(n)$
- Hope: they can answer queries in time $\ll n$ using this data structure

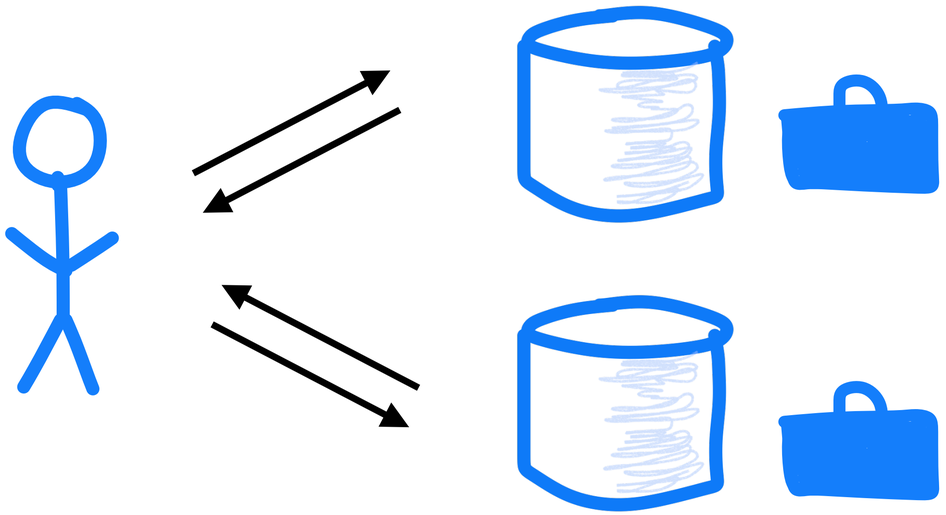
Efficiency Metrics

- Storage: size of the data structure
  - At most $\mathbf{poly}(n)$
  - Necessarily $\geq n$
  - Goal: $n^{1+o(1)}$

Solution: Trade Time for Space

[BIM'00,BIPW'17,CHR'17, HOWW'18,LMW'23,GLMDS'25]

Referred to as "PIR with preprocessing" or "doubly-efficient PIR" (unkeyed)



- Servers preprocess the DB into some data structure of size $\mathbf{poly}(n)$
- Hope: they can answer queries in time $\ll n$ using this data structure

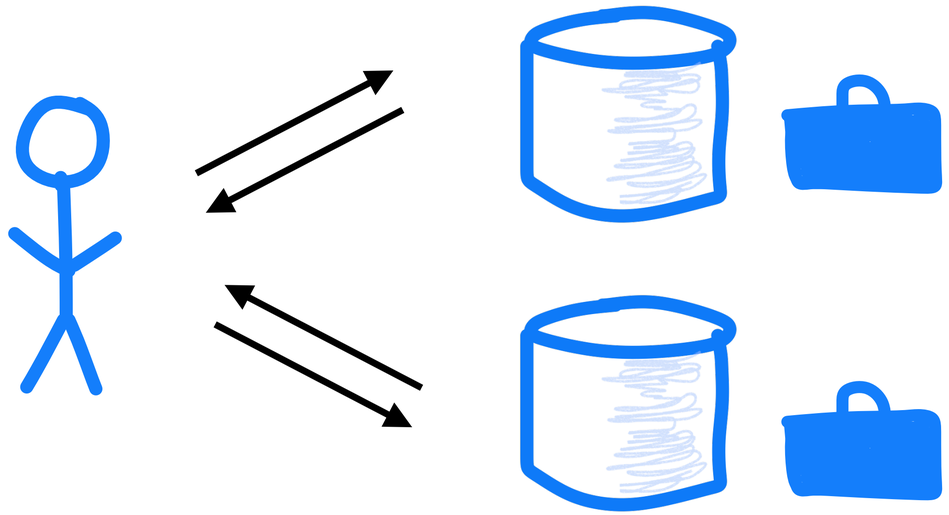
Efficiency Metrics

- Storage: size of the data structure
  - At most $\mathbf{poly}(n)$
  - Necessarily $\geq n$
  - Goal: $n^{1+o(1)}$
- Server time per query
  - At most $n^{1-\Omega(1)}$
  - Goal: $n^{o(1)}$

Solution: Trade Time for Space

[BIM'00,BIPW'17,CHR'17, HOWW'18,LMW'23,GLMDS'25]

Referred to as "PIR with preprocessing" or "doubly-efficient PIR" (unkeyed)



- Servers preprocess the DB into some data structure of size $\mathbf{poly}(n)$
- Hope: they can answer queries in time $\ll n$ using this data structure

Efficiency Metrics

- Storage: size of the data structure
  - At most $\mathbf{poly}(n)$
  - Necessarily $\geq n$
  - Goal: $n^{1+o(1)}$
- Server time per query
  - At most $n^{1-\Omega(1)}$
  - Goal: $n^{o(1)}$

Disclaimer: most constructions (including ours) sacrifice communication efficiency for sublinear server time

25 years of work on PIR with preprocessing

Construction	Setting	Time per query	Storage
[BIM00]	2 servers Info-theoretic	$n^{0.82}$	$n^{1.54}$

25 years of work on PIR with preprocessing

Construction	**Setting**	**Time per query**	**Storage**	**Concrete storage** **(2GB DB, 0.7MB comm.)**
[BIM00]	2 servers Info-theoretic	$n^{0.82}$	$n^{1.54}$	760000 TB (~$10M in hard drives)

25 years of work on PIR with preprocessing

Construction	**Setting**	**Time per query**	**Storage**	**Concrete storage** **(2GB DB, 0.7MB comm.)**
[BIM00]	2 servers Info-theoretic	$n^{0.82}$	$n^{1.54}$	760000 TB (~$10M in hard drives)
[LMW23]	1 server, secure assuming RingLWE	$n^{o(1)}$	$n^{1+o(1)}$	> 2000000 TB (~$20M in hard drives)

25 years of work on PIR with preprocessing

Construction	**Setting**	**Time per query**	**Storage**	**Concrete storage** **(2GB DB, 0.7MB comm.)**
[BIM00]	2 servers Info-theoretic	$n^{0.82}$	$n^{1.54}$	760000 TB (~$10M in hard drives)
[LMW23]	1 server, secure assuming RingLWE	$n^{o(1)}$	$n^{1+o(1)}$	> 2000000 TB (~$20M in hard drives)
Our work	2 servers Info-theoretic	$n^{0.82}$	$n^{1+o(1)}$	1 TB (one $50 hard drive)

Theorem. For any database of $n > 10^6$ bits, there exists information-theoretic, two-server PIR with preprocessing with:

- $1.5 \cdot \sqrt{\log_2 n} \cdot n$ bits of storage,
- $12 \cdot n^{0.82}$ server RAM lookups per query,
- $12 \cdot n^{0.82}$ bits of communication.

Our schemes support a broader time-space tradeoff, that strictly improves on prior work.

Theorem. For any database of $n > 10^6$ bits, there exists information-theoretic, two-server PIR with preprocessing with:

- $1.5 \cdot \sqrt{\log_2 n} \cdot n$ bits of storage,
- $12 \cdot n^{0.82}$ server RAM lookups per query,
- $12 \cdot n^{0.82}$ bits of communication.

First info-theoretic PIR with

1. constant number of servers,
2. quasilinear storage, and
3. polynomially-sublinear time.

Our schemes support a broader time-space tradeoff, that strictly improves on prior work.

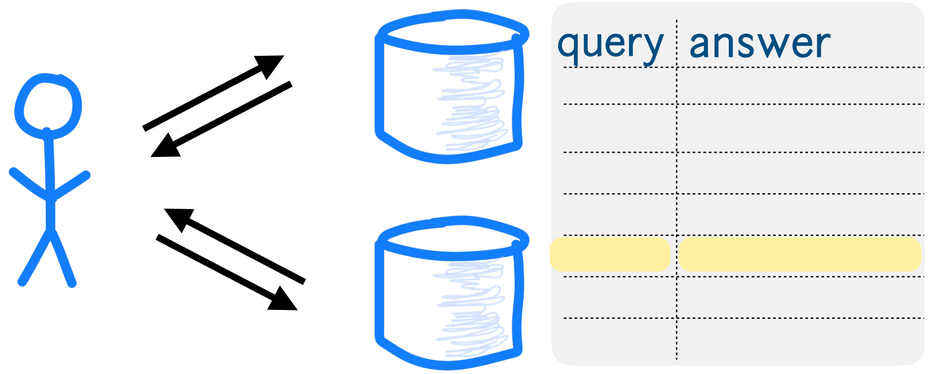
Previous Work [BIM00, GLM+25]: PIR with Preprocessing from Polynomials and Derivatives

Paradigm: Brute Force Preprocessing

- Starting point: any PIR protocol with query length ℓ_q and answer length ℓ_a

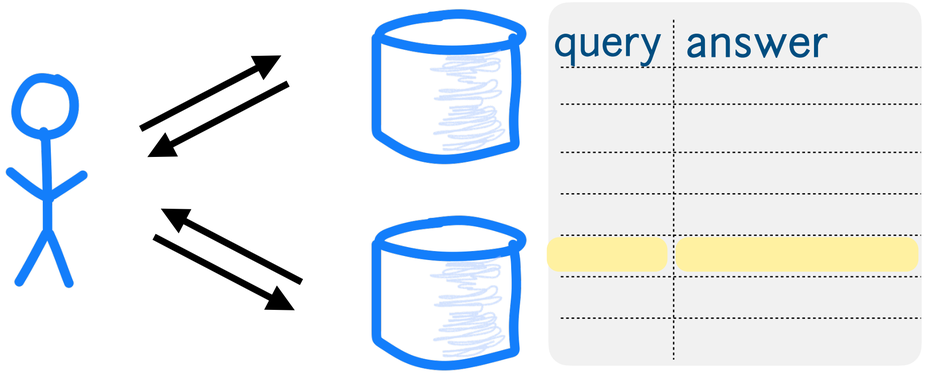
Paradigm: Brute Force Preprocessing

- Starting point: any PIR protocol with query length ℓ_q and answer length ℓ_a
- Preprocess by precomputing answers to every possible query



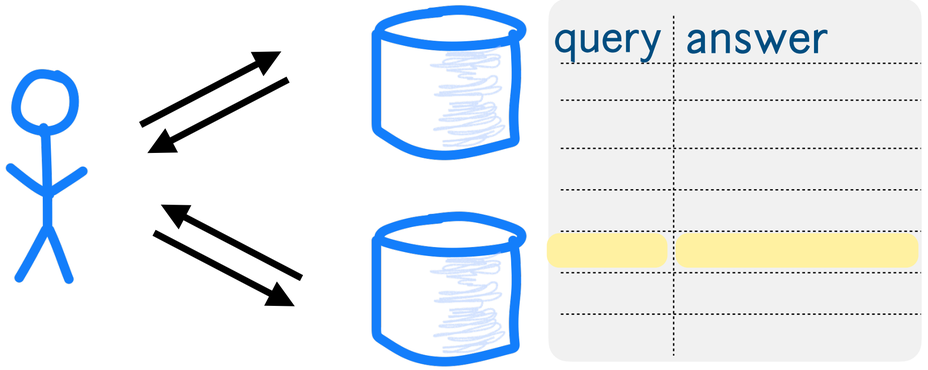
Paradigm: Brute Force Preprocessing

- Starting point: any PIR protocol with query length ℓ_q and answer length ℓ_a
- Preprocess by precomputing answers to every possible query
- Resulting PIR with preprocessing:
  - Storage: $\ell_a \cdot 2^{\ell_q}$



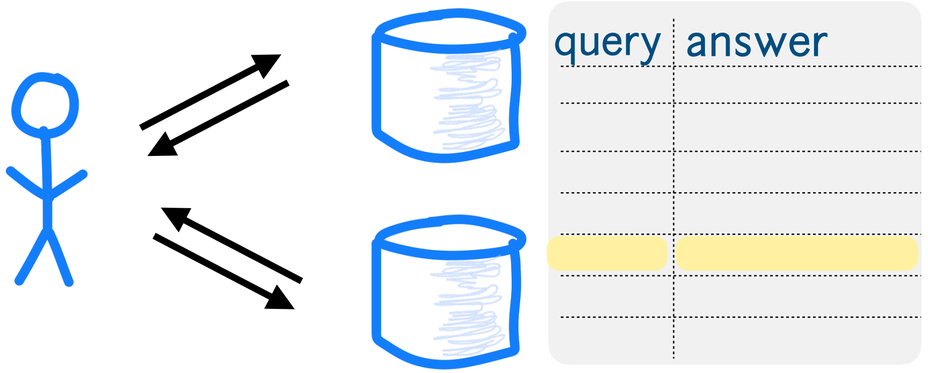
Paradigm: Brute Force Preprocessing

- Starting point: any PIR protocol with query length ℓ_q and answer length ℓ_a
- Preprocess by precomputing answers to every possible query
- Resulting PIR with preprocessing:
  - Storage: $\ell_a \cdot 2^{\ell_q}$
  - Server time per query: $\ell_q + \ell_a$



Paradigm: Brute Force Preprocessing

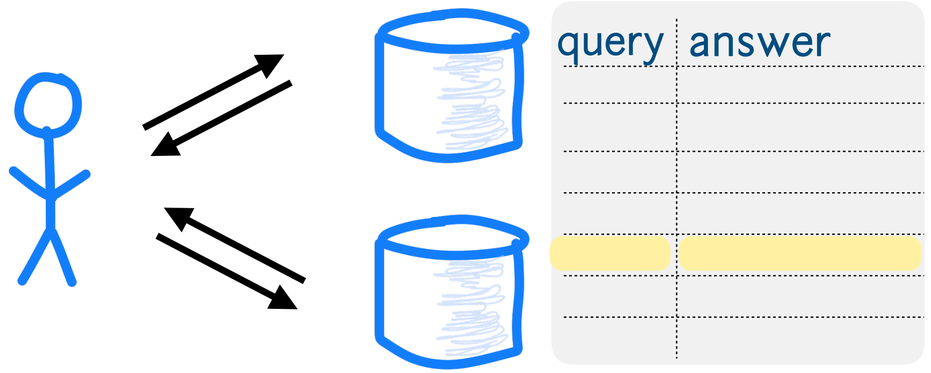
- Starting point: any PIR protocol with query length ℓ_q and answer length ℓ_a
- Preprocess by precomputing answers to every possible query
- Resulting PIR with preprocessing:
  - Storage: $\ell_a \cdot 2^{\ell_q}$
  - Server time per query: $\ell_q + \ell_a$



Moral: we should look for PIR protocols with query length $\ell_q \leq O(\log n)$

Paradigm: Brute Force Preprocessing

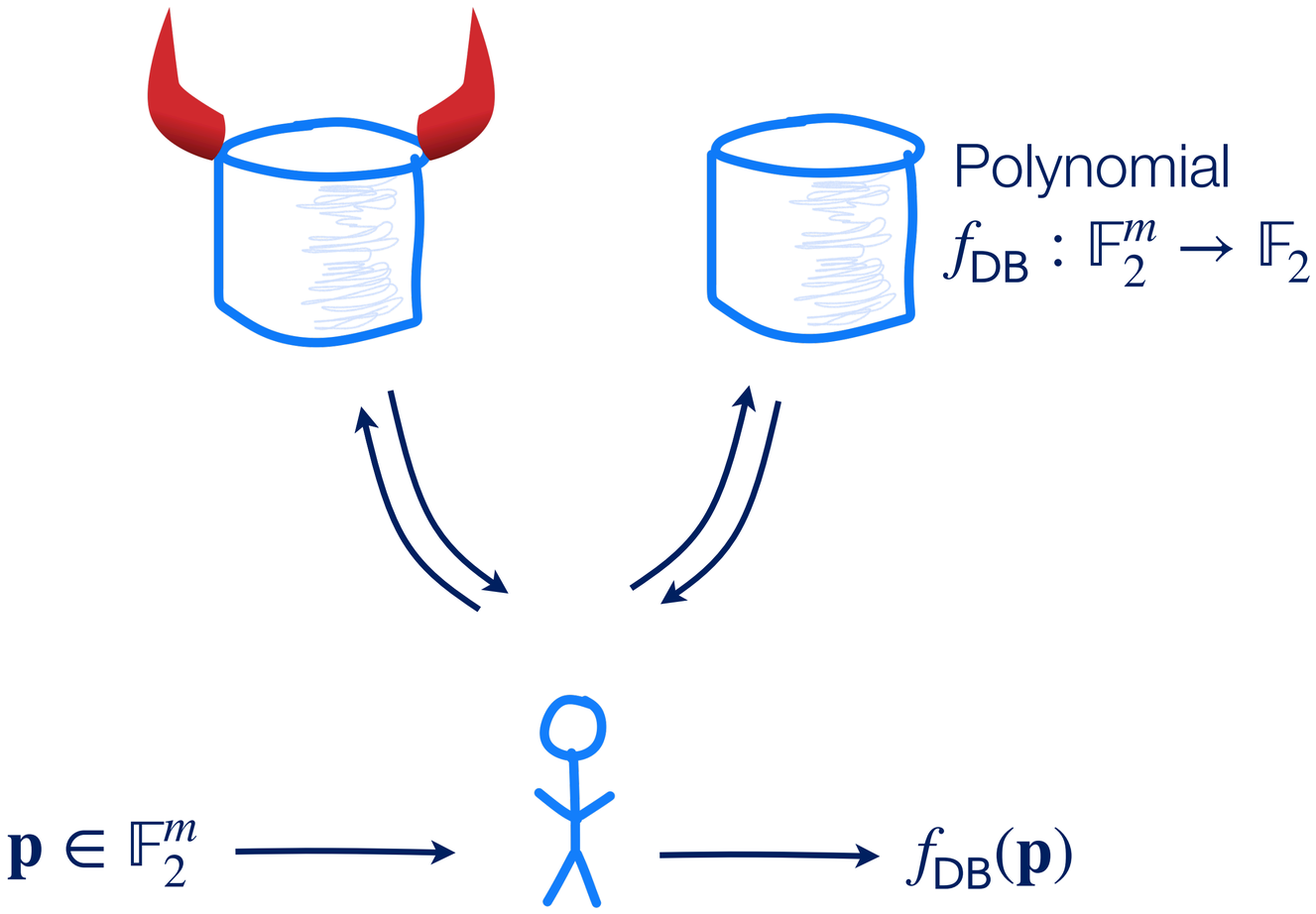
- Starting point: any PIR protocol with query length ℓ_q and answer length ℓ_a
- Preprocess by precomputing answers to every possible query
- Resulting PIR with preprocessing:
  - Storage: $\ell_a \cdot 2^{\ell_q}$
  - Server time per query: $\ell_q + \ell_a$



Moral: we should look for PIR protocols with query length $\ell_q \leq O(\log n)$

Spoiler: our construction will deviate from this brute force paradigm!
(but we will still need query length $\ell_q \leq O(\log n)$)

Same abstraction: private polynomial evaluation



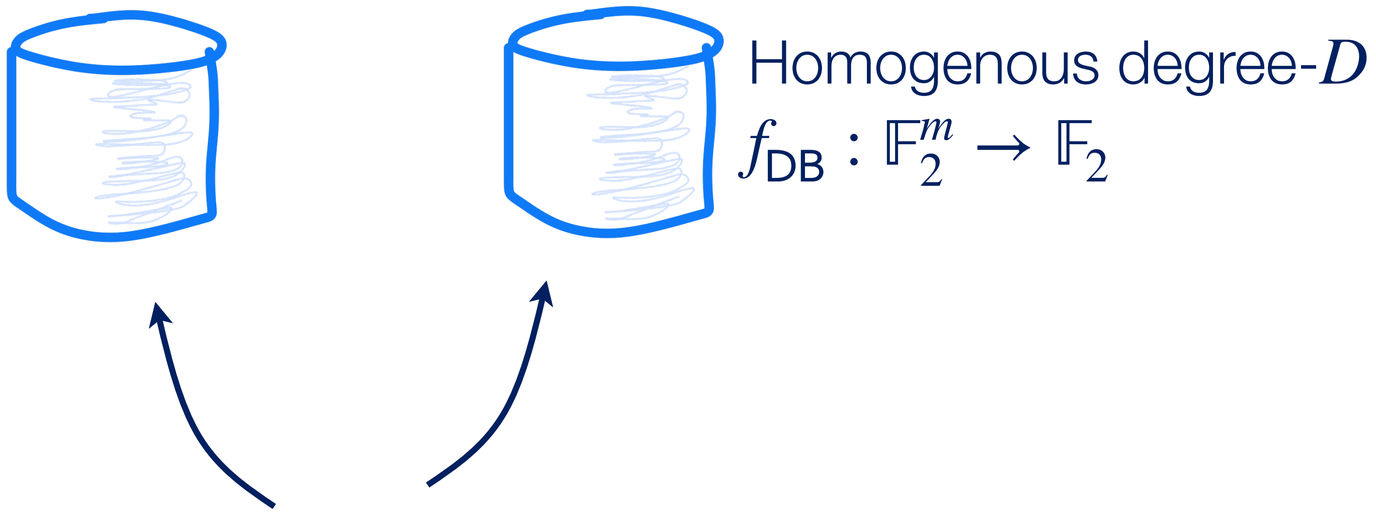
We need:

1. f_{DB} is homogenous and degree-D
2. $\mathbf{p}$ will encode the query index $i \in [n]$
3. $f_{\mathsf{DB}}(\mathbf{p}) = \mathsf{DB}_i$
4. Parameter counting $\rightarrow \binom{m}{D} \geq n$

Correctness: for any f_{DB} and point $\mathbf{p}$, a user interacting with two honest servers learns $f_{\mathsf{DB}}(\mathbf{p})$.

Privacy: each server learns nothing about $\mathbf{p}$, even if the server is malicious.

PIR from derivatives [WY05]



Homogenous degree-D
$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line
$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

Fact: *leading coefficient of* $f_{\mathsf{DB}}(\mathbf{L}(t))$ *is* $f_{\mathsf{DB}}(\mathbf{p})$

PIR from derivatives [WY05]

Homogenous degree-D
$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Query: $\mathbf{r} = L(0)$

Query: $\mathbf{p} + \mathbf{r} = L(1)$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line
$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

Fact: *leading coefficient of* $f_{\mathsf{DB}}(\mathbf{L}(t))$ *is* $f_{\mathsf{DB}}(\mathbf{p})$

PIR from derivatives [WY05]

Homogenous degree-D

$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Query: $\mathbf{r} = L(0)$

Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r} = L(1)$

Ans: $f_{\mathsf{DB}}(\mathbf{p} + \mathbf{r}), \nabla f_{\mathsf{DB}}(\mathbf{p} + \mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

Fact: *leading coefficient of* $f_{\mathsf{DB}}(\mathbf{L}(t))$ *is* $f_{\mathsf{DB}}(\mathbf{p})$

PIR from derivatives [WY05]

Homogenous degree-D
$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Query: $\mathbf{r} = L(0)$

Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r} = L(1)$

Ans: $f_{\mathsf{DB}}(\mathbf{p} + \mathbf{r}), \nabla f_{\mathsf{DB}}(\mathbf{p} + \mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line
$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

Recover $f_{\mathsf{DB}} \circ \mathbf{L}$ from evaluations and first-order derivatives at 0 and 1 via Hermite interpolation

Fact: *leading coefficient of* $f_{\mathsf{DB}}(\mathbf{L}(t))$ *is* $f_{\mathsf{DB}}(\mathbf{p})$

PIR from derivatives [WY05]

Homogenous degree-D
$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Query: $\mathbf{r} = L(0)$

Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r} = L(1)$

Ans: $f_{\mathsf{DB}}(\mathbf{p} + \mathbf{r}), \nabla f_{\mathsf{DB}}(\mathbf{p} + \mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line
$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

Recover $f_{\mathsf{DB}} \circ \mathbf{L}$ from evaluations and first-order derivatives at 0 and 1 via Hermite interpolation

Fact: *leading coefficient of* $f_{\mathsf{DB}}(\mathbf{L}(t))$ *is* $f_{\mathsf{DB}}(\mathbf{p})$

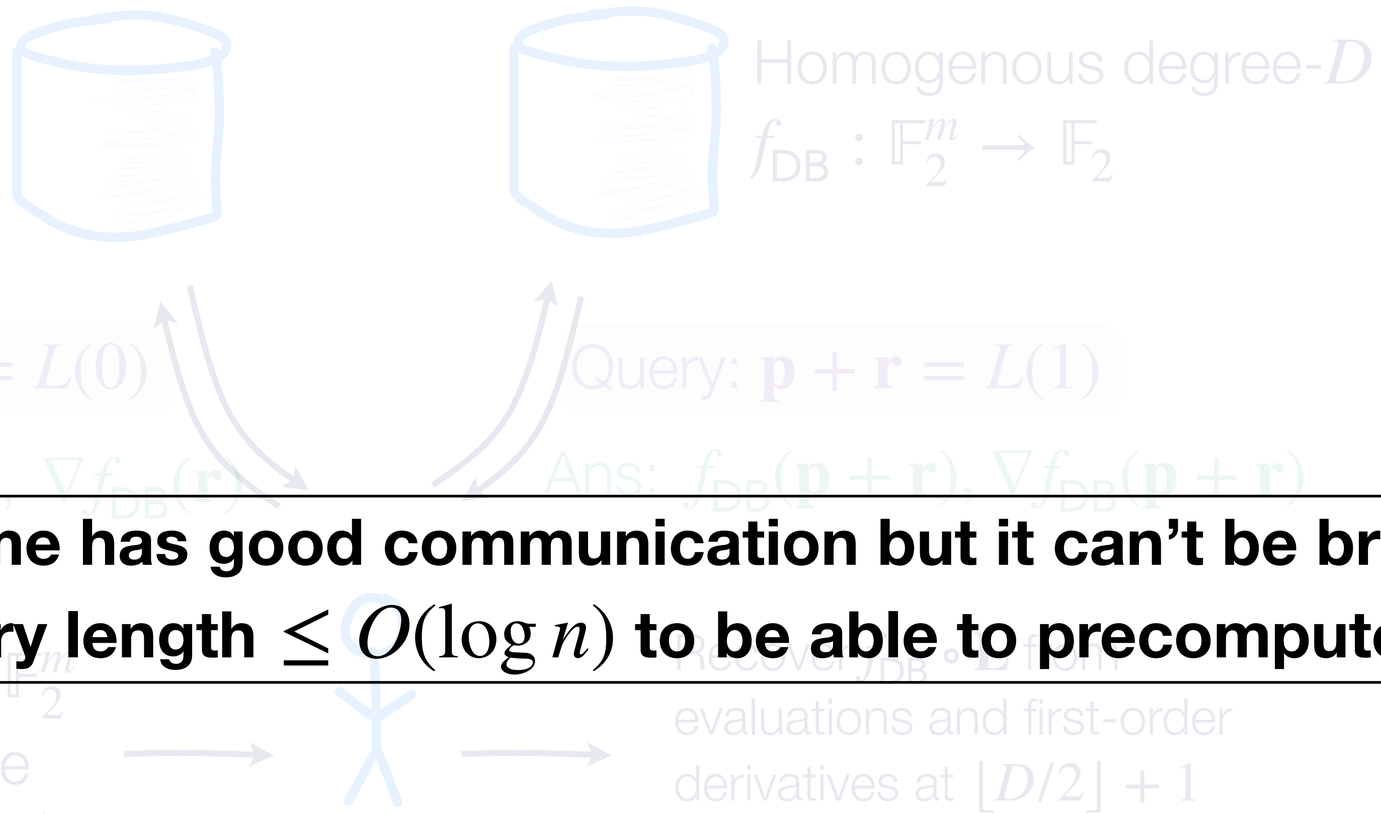
With 2 servers, gives "balanced" PIR with

- ➡ $D = 3$
- ➡ $\binom{m}{D} \geq n \implies m = n^{1/3}$
- ➡ query $n^{1/3}$ and answer $n^{1/3}$ (partial derivatives in each direction)

PIR from derivatives [WY05]

With 2 servers, gives "balanced" PIR with

- ➡ $D = 3$
- ➡ $\binom{m}{D} \geq n \implies m = n^{1/3}$
- ➡ query $n^{1/3}$ and answer $n^{1/3}$ (partial derivatives in each direction)



This scheme has good communication but it can't be brute-force preprocessed!

Need query length $\leq O(\log n)$ to be able to precompute answers to all queries

Fix: even more derivatives! [BIM00,GLM+25]

Homogenous degree-D

$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Query: $\mathbf{r} = L(0)$

Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r} = L(1)$

Ans: $f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line $\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

Recover $f_{\mathsf{DB}} \circ \mathbf{L}$ from evaluations and $\leq \lfloor D/2 \rfloor$-order derivatives at 2 points via Hermite interpolation

Fix: even more derivatives! [BIM00,GLM+25]

Homogenous degree-D
$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Query: $\mathbf{r} = L(0)$

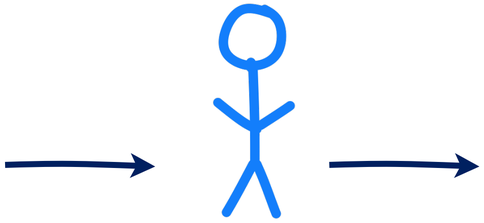
Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r} = L(1)$

Ans: $f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line
$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$



Recover $f_{\mathsf{DB}} \circ \mathbf{L}$ from evaluations and $\leq \lfloor D/2 \rfloor$-order derivatives at 2 points via Hermite interpolation

With 2 servers, gives "imbalanced" PIR with

- ➡ $m = (1 + o(1)) \cdot \log n$
- ➡ $D = m/2$
- ➡ query $m \approx \log n$ and answer $\binom{m}{\lfloor D/2 \rfloor} \approx n^{0.82}$ (partial derivatives to order $\leq \lfloor D/2 \rfloor$)

Brute Force → PIR with Preprocessing

Precomputing answers for every query

Homogenous deg-D

$f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$

Query: $\mathbf{r}$

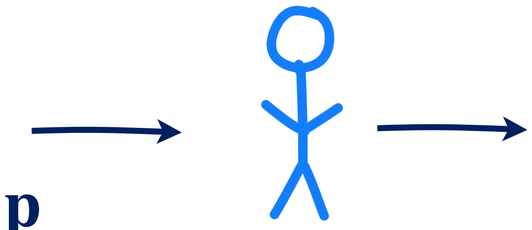
Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r}$

Ans: $f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

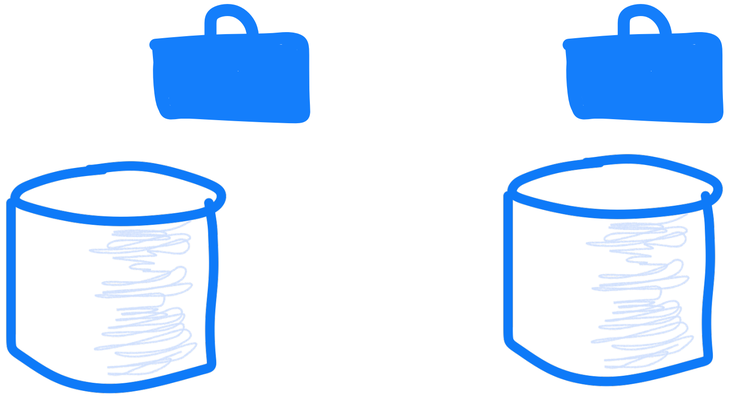
Sample line $\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$



Recover $f_{\mathsf{DB}} \circ \mathbf{L}$ from evaluations and $\leq \lfloor D/2 \rfloor$-order derivatives at 2 points via Hermite interpolation

Brute Force → PIR with Preprocessing

Precomputing answers for every query



$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

Query: $\mathbf{r}$

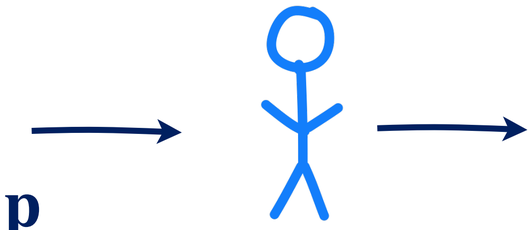
Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r}$

Ans: $f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

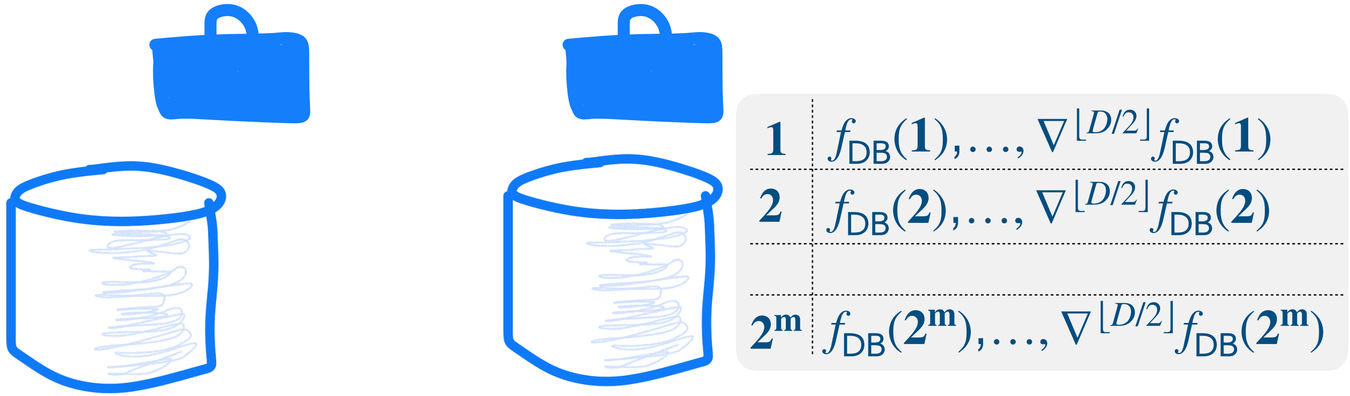
Sample line $\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$



Recover $f_{\mathsf{DB}} \circ \mathbf{L}$ from evaluations and $\leq \lfloor D/2 \rfloor$-order derivatives at 2 points via Hermite interpolation

Brute Force → PIR with Preprocessing

Precomputing answers for every query



$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

Query: $\mathbf{r}$

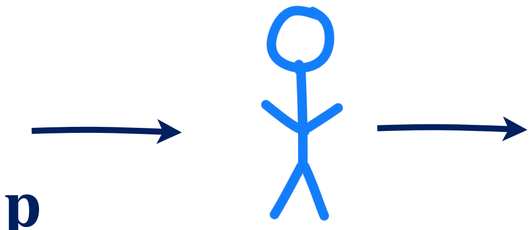
Ans: $f_{\mathsf{DB}}(\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{r})$

Query: $\mathbf{p} + \mathbf{r}$

Ans: $f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \nabla f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r}),\ \ldots,\ \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{p}+\mathbf{r})$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line $\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$



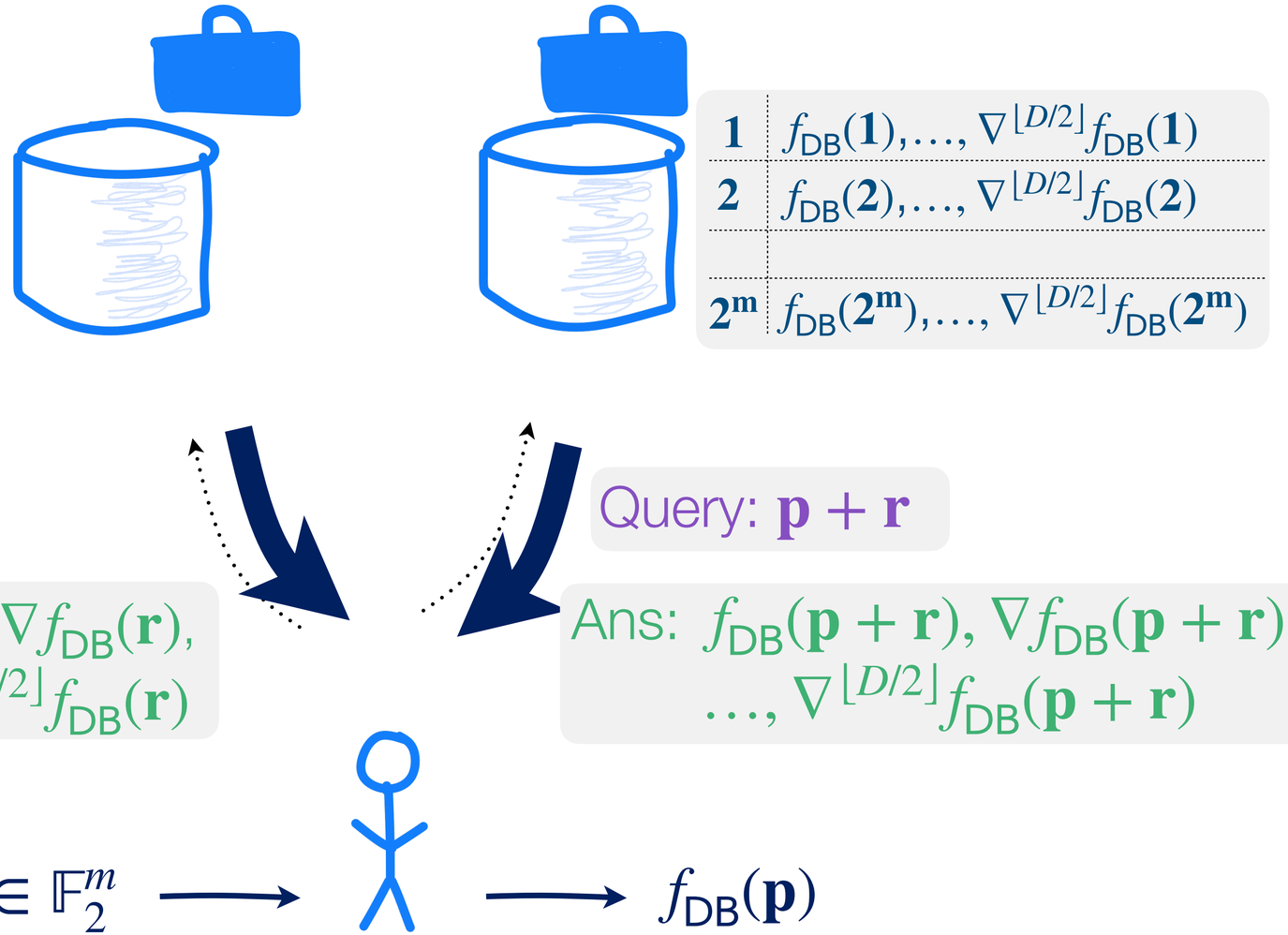
Recover $f_{\mathsf{DB}} \circ \mathbf{L}$ from evaluations and $\leq \lfloor D/2 \rfloor$-order derivatives at 2 points via Hermite interpolation

With 2 servers, gives preprocessing PIR with

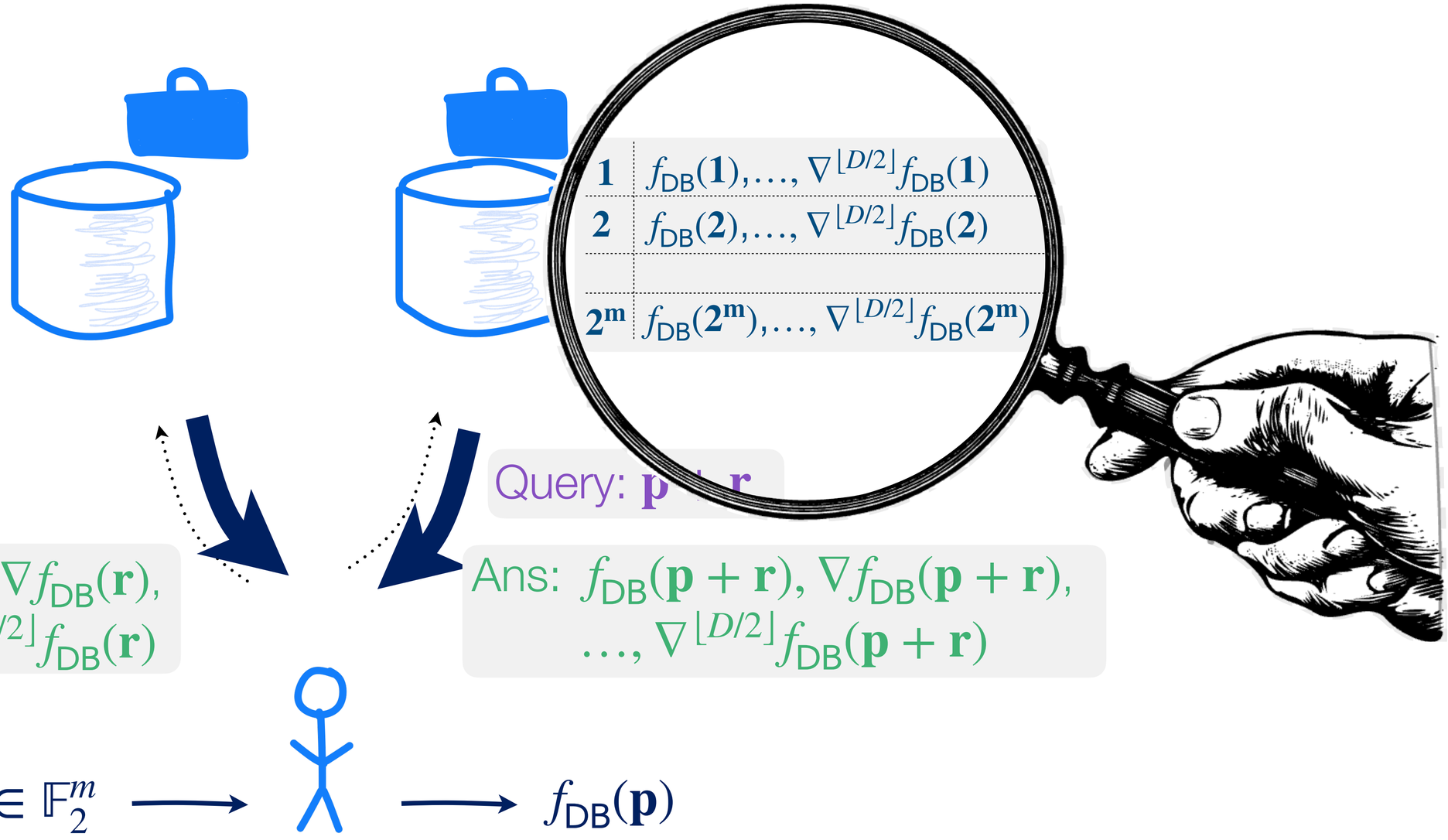
- ➡ Query $O(\log n)$ and answer $n^{0.82}$
- ➡ $2^m \cdot n^{0.82} = n^{1.82+o(1)}$ server storage
- ➡ $n^{0.82}$ server time

Our Work: Preprocessing Better than Brute Force

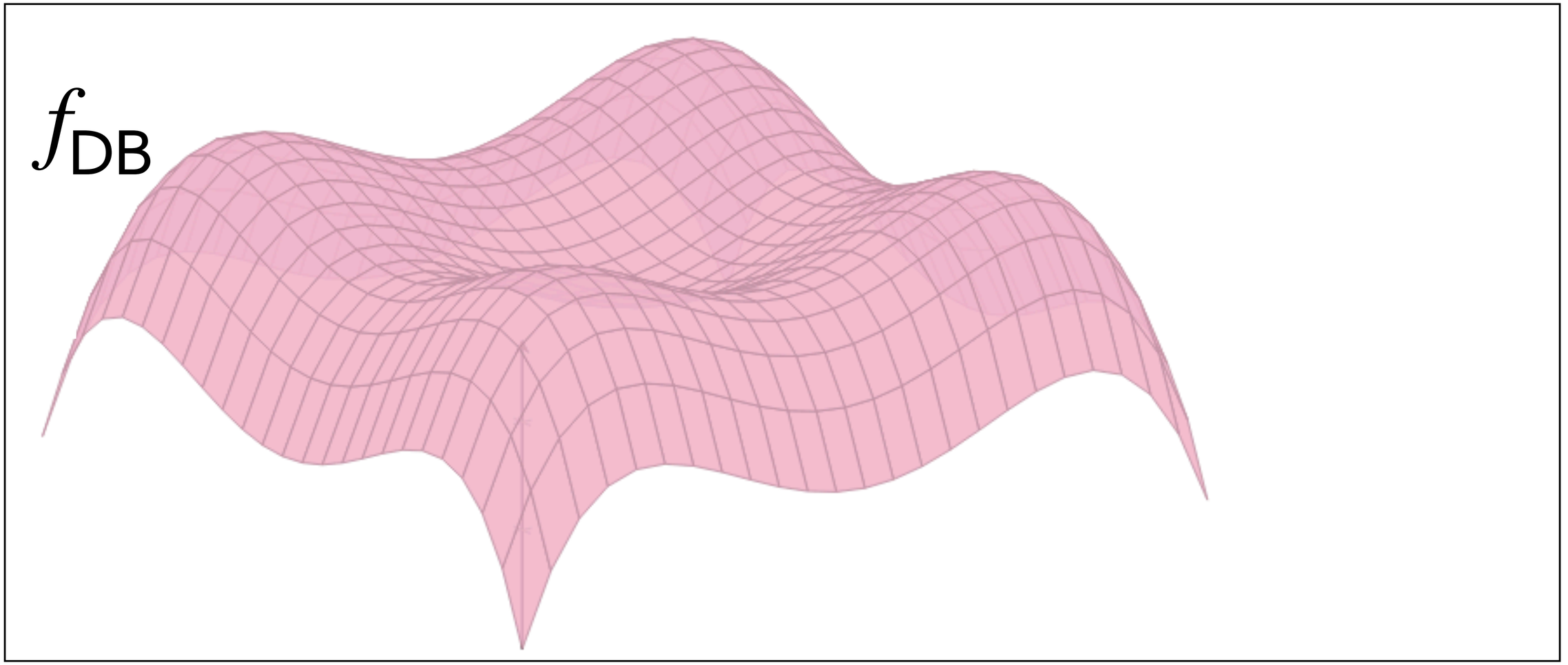
Finding Redundancy in the Brute-Force Data Structure



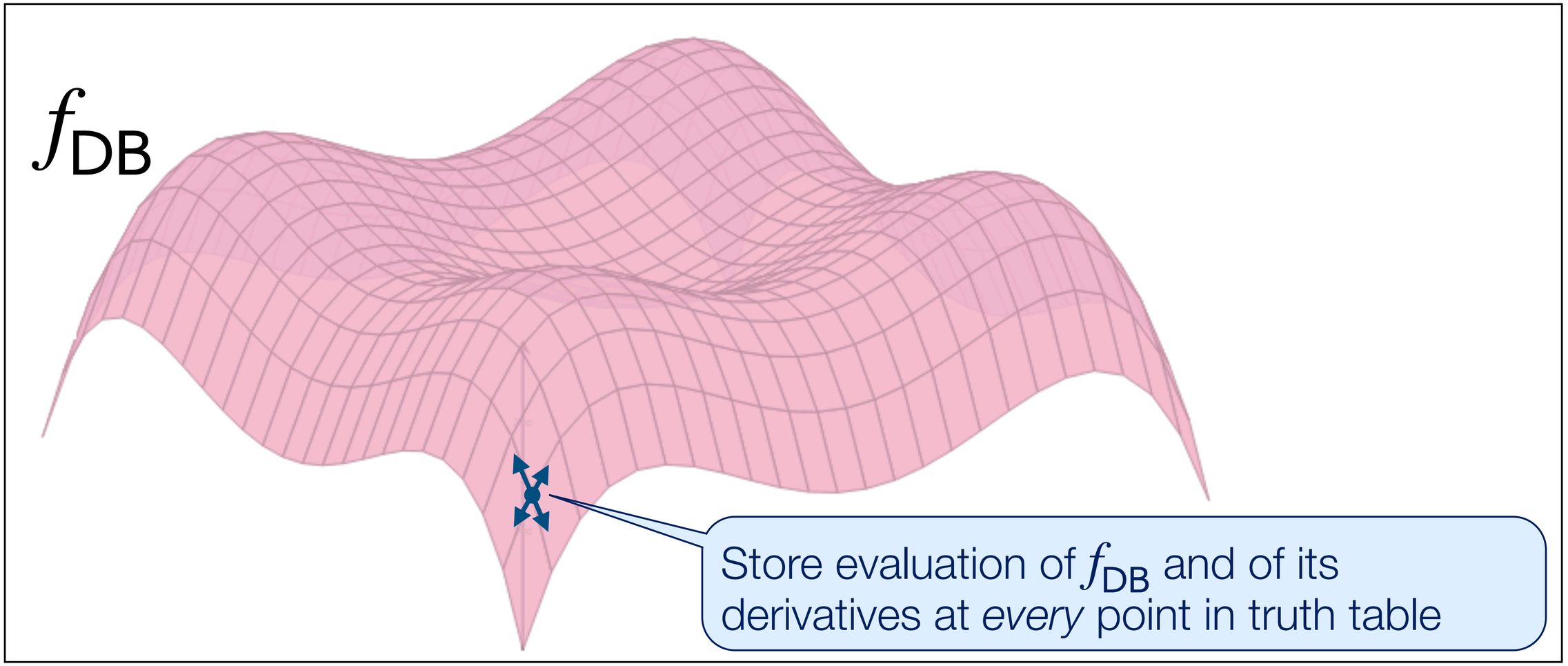
Finding Redundancy in the Brute-Force Data Structure



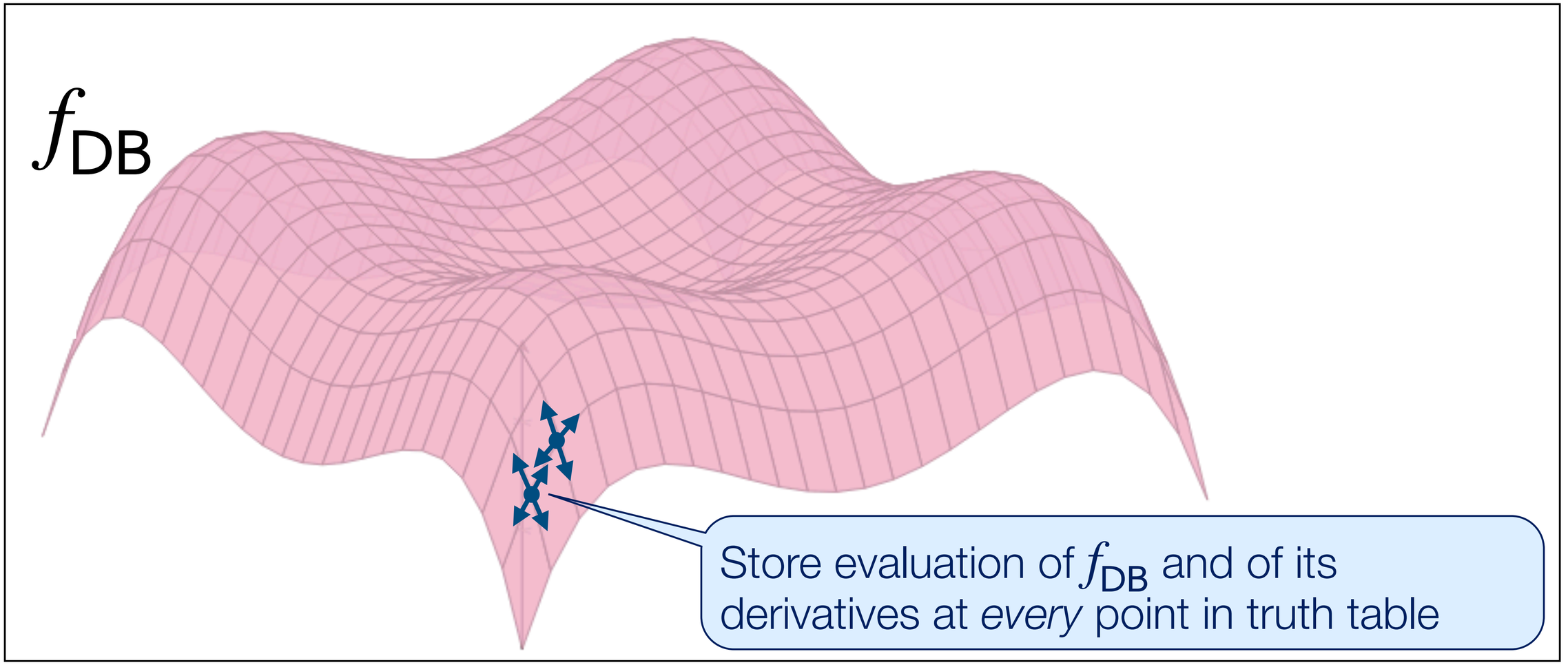
Finding Redundancy in the Brute-Force Data Structure



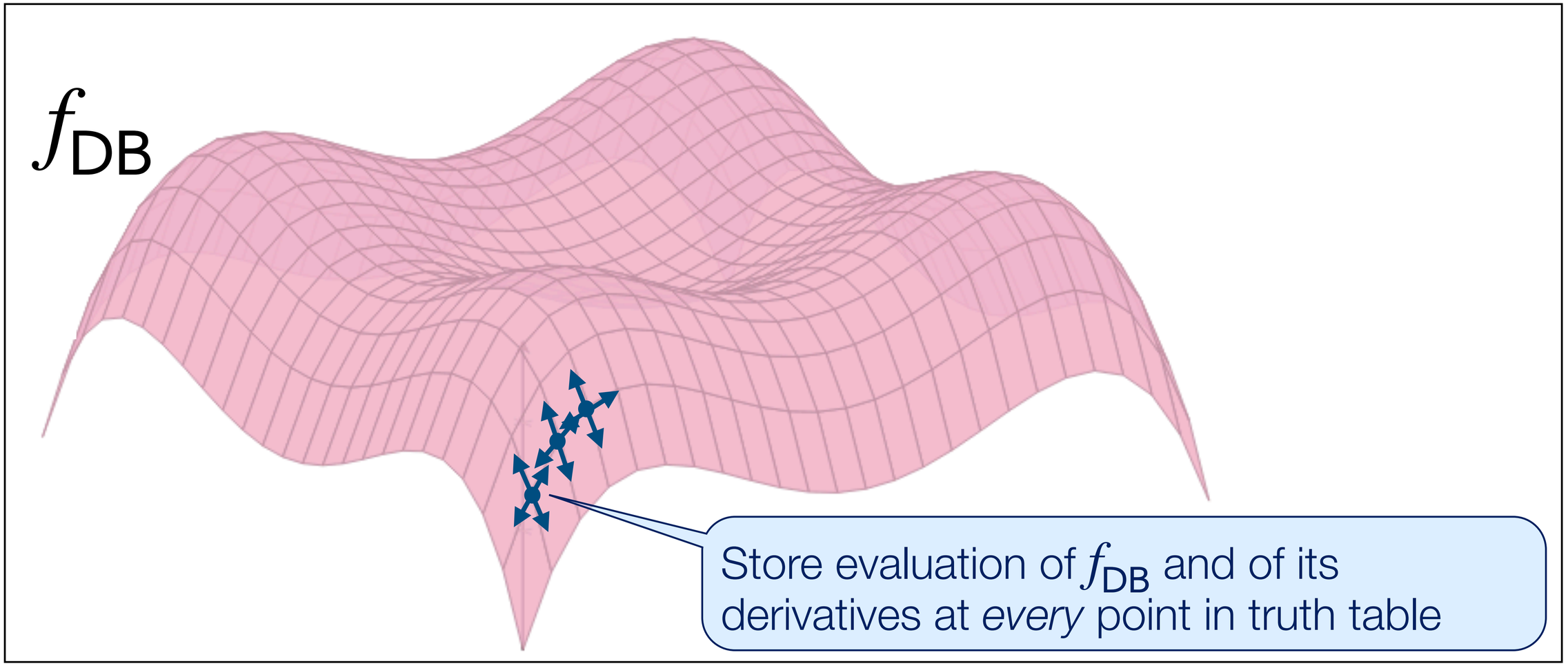
Finding Redundancy in the Brute-Force Data Structure



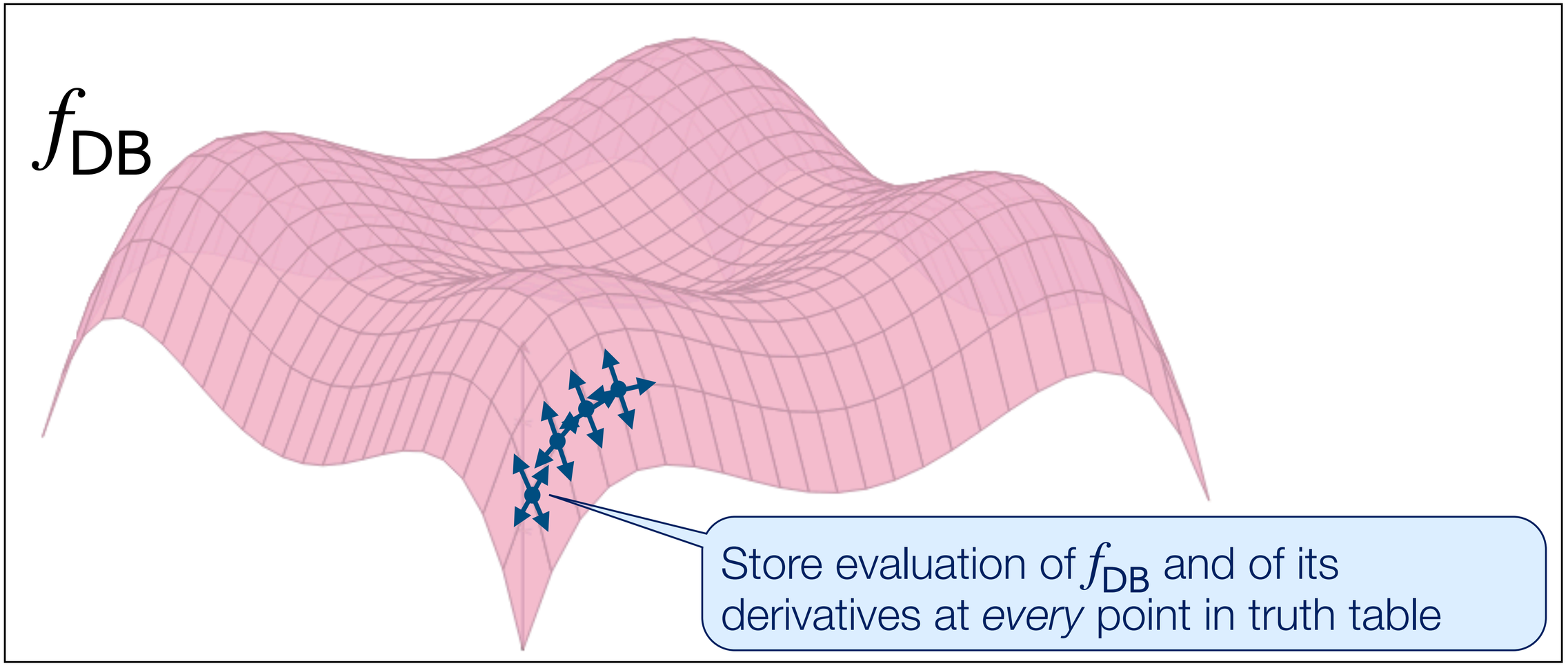
Finding Redundancy in the Brute-Force Data Structure



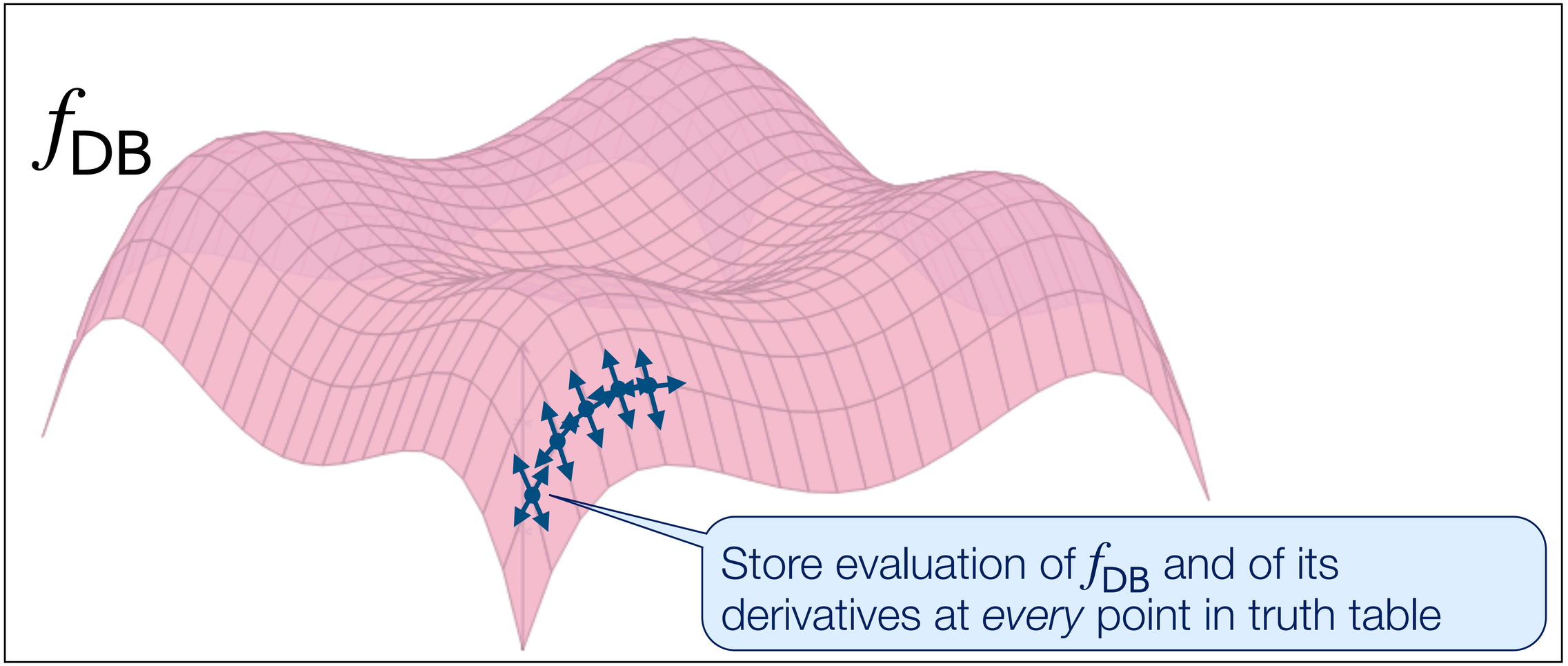
Finding Redundancy in the Brute-Force Data Structure



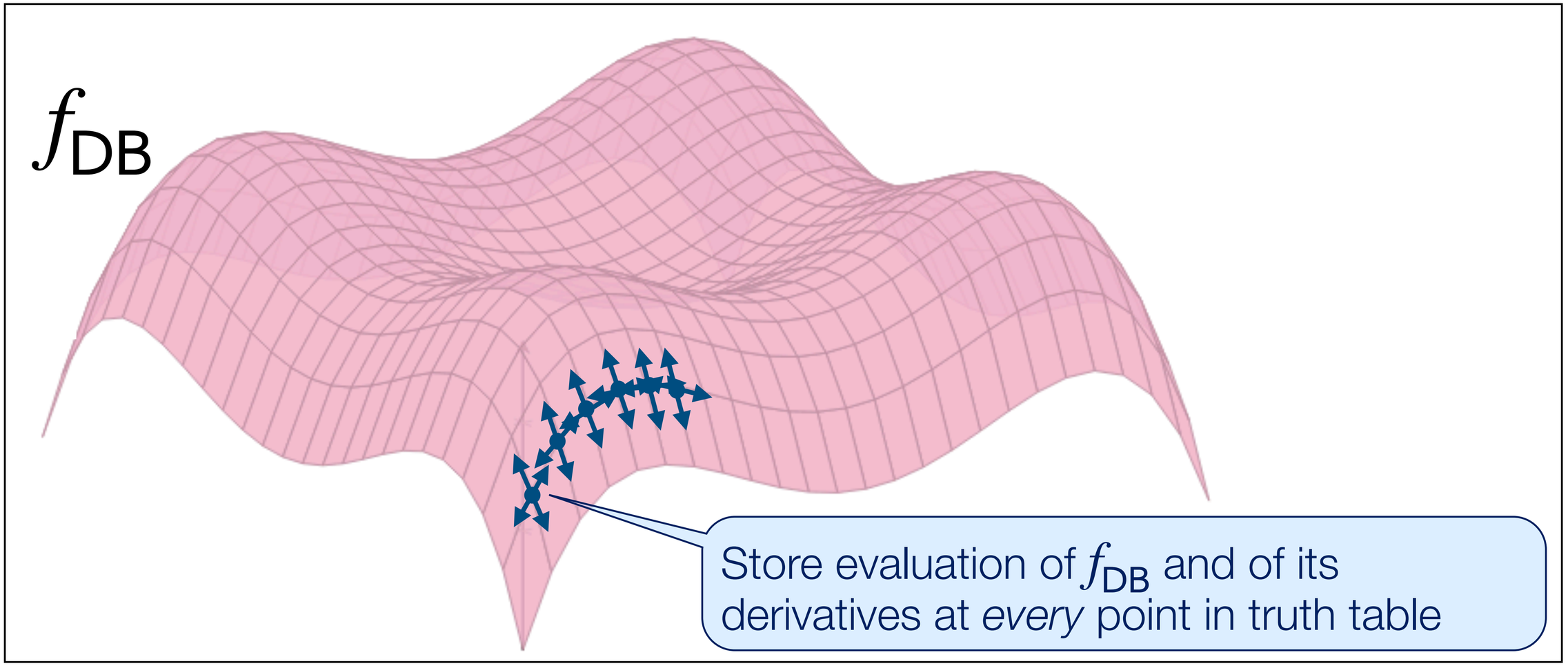
Finding Redundancy in the Brute-Force Data Structure



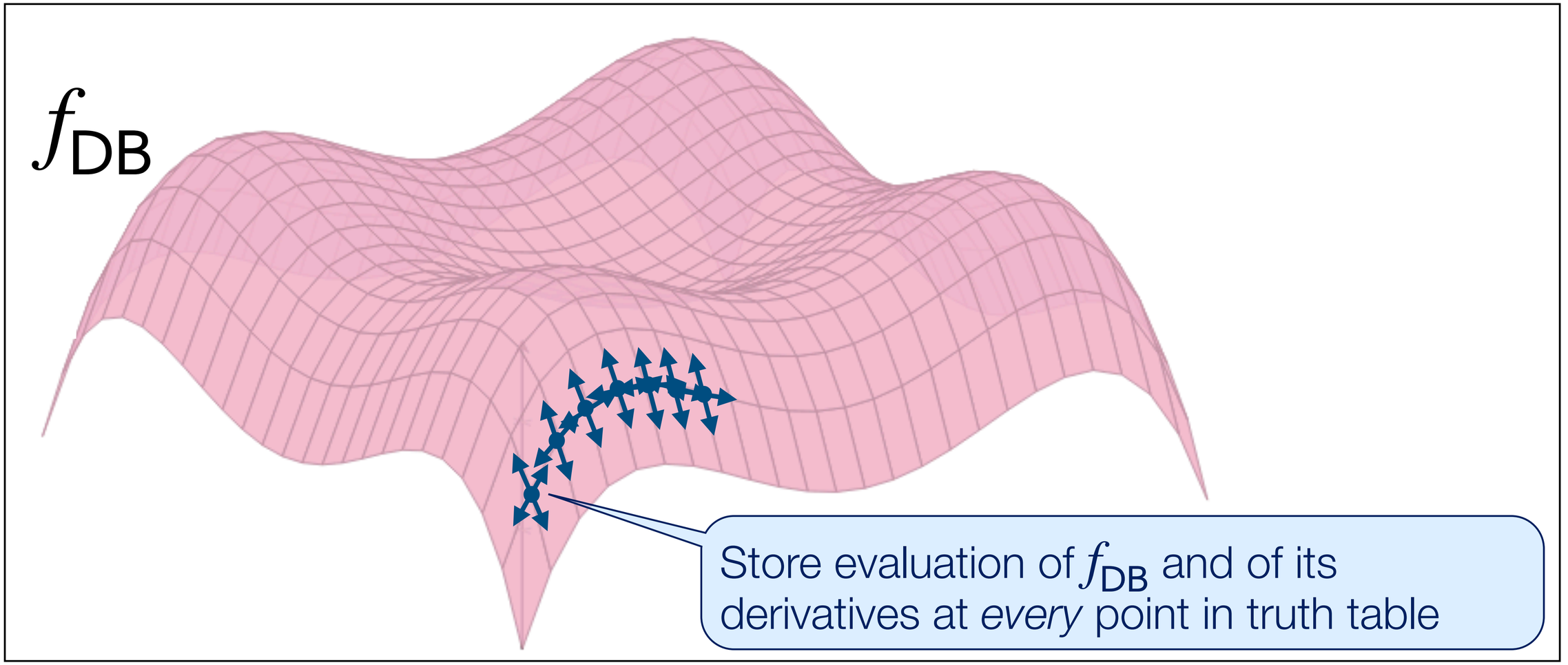
Finding Redundancy in the Brute-Force Data Structure



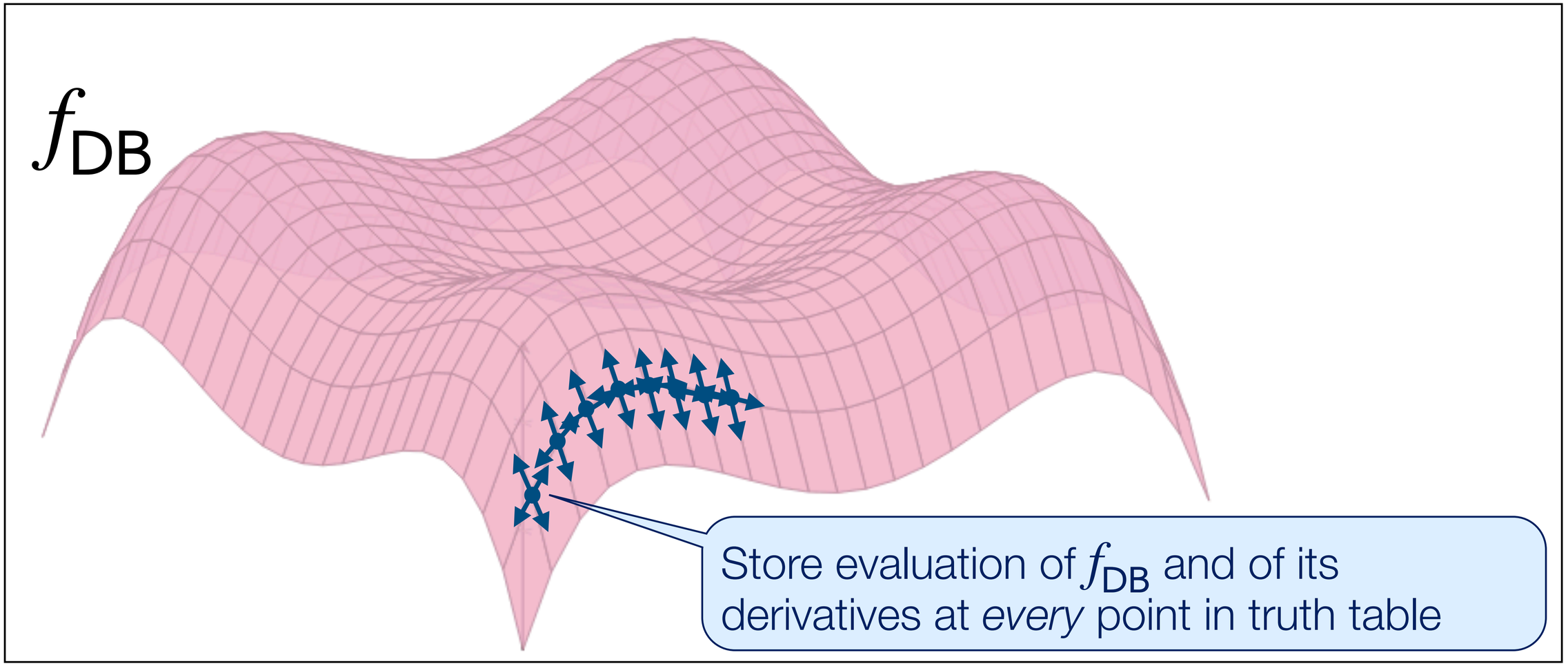
Finding Redundancy in the Brute-Force Data Structure



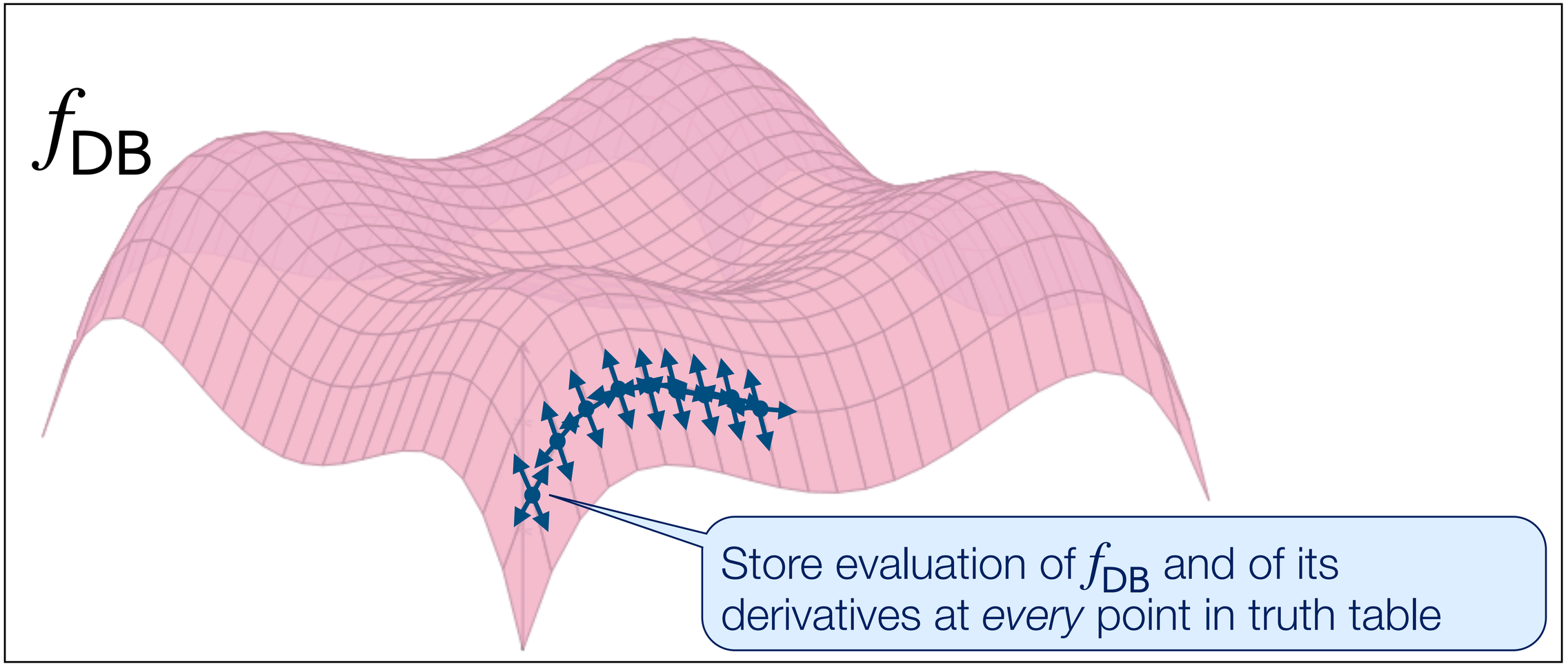
Finding Redundancy in the Brute-Force Data Structure



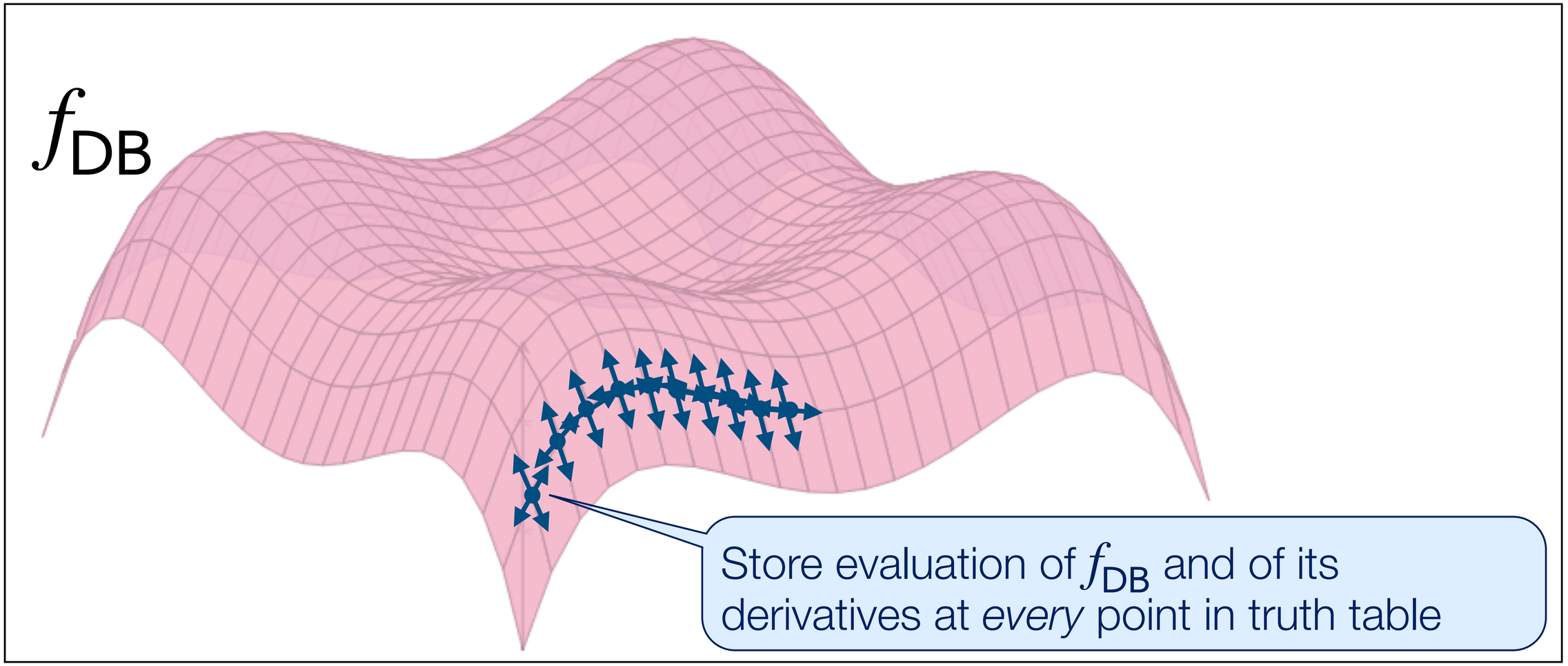
Finding Redundancy in the Brute-Force Data Structure



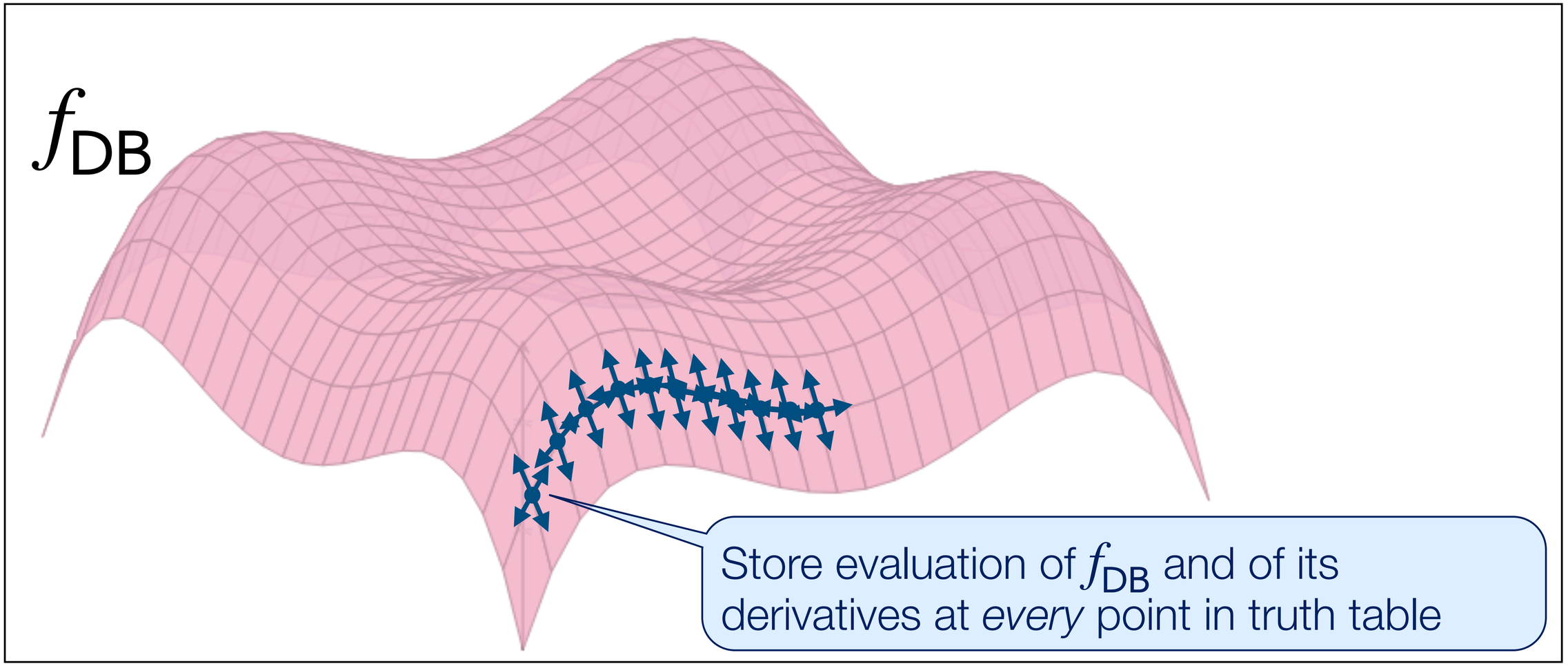
Finding Redundancy in the Brute-Force Data Structure



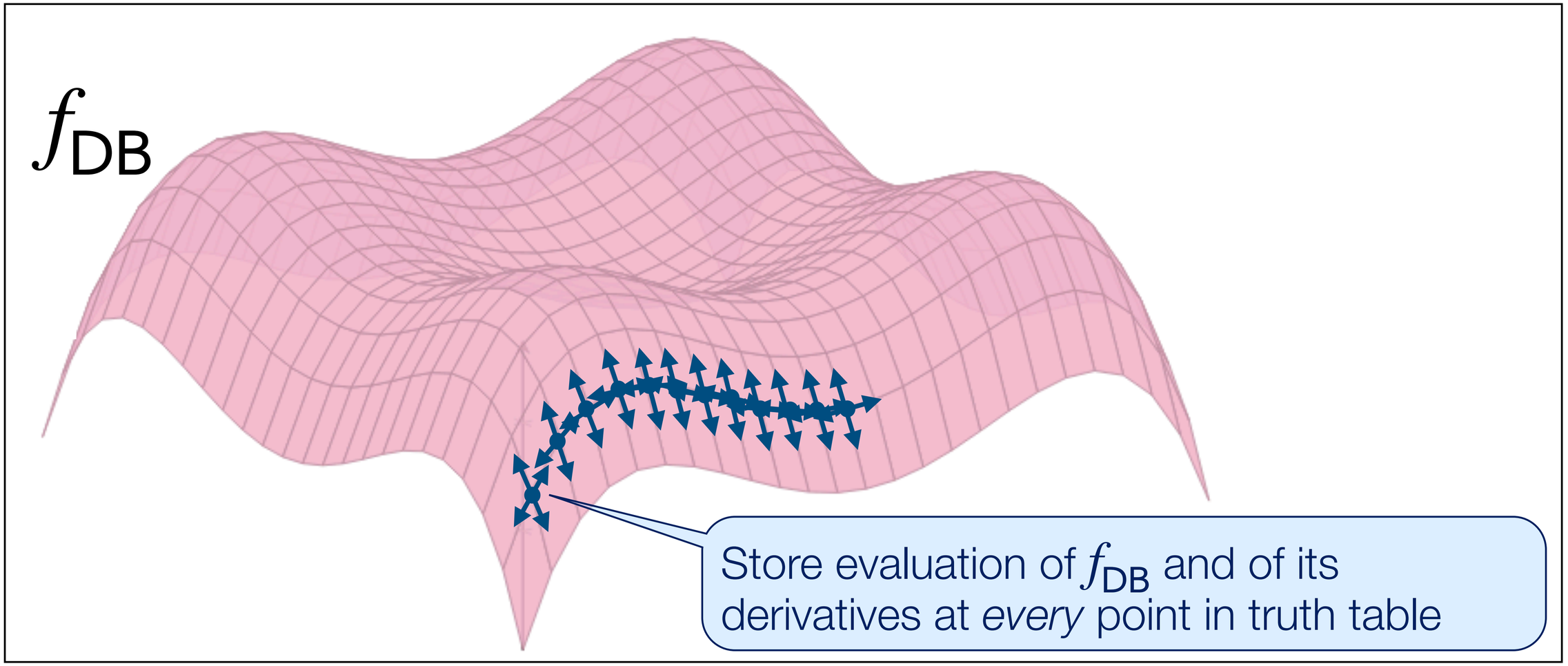
Finding Redundancy in the Brute-Force Data Structure



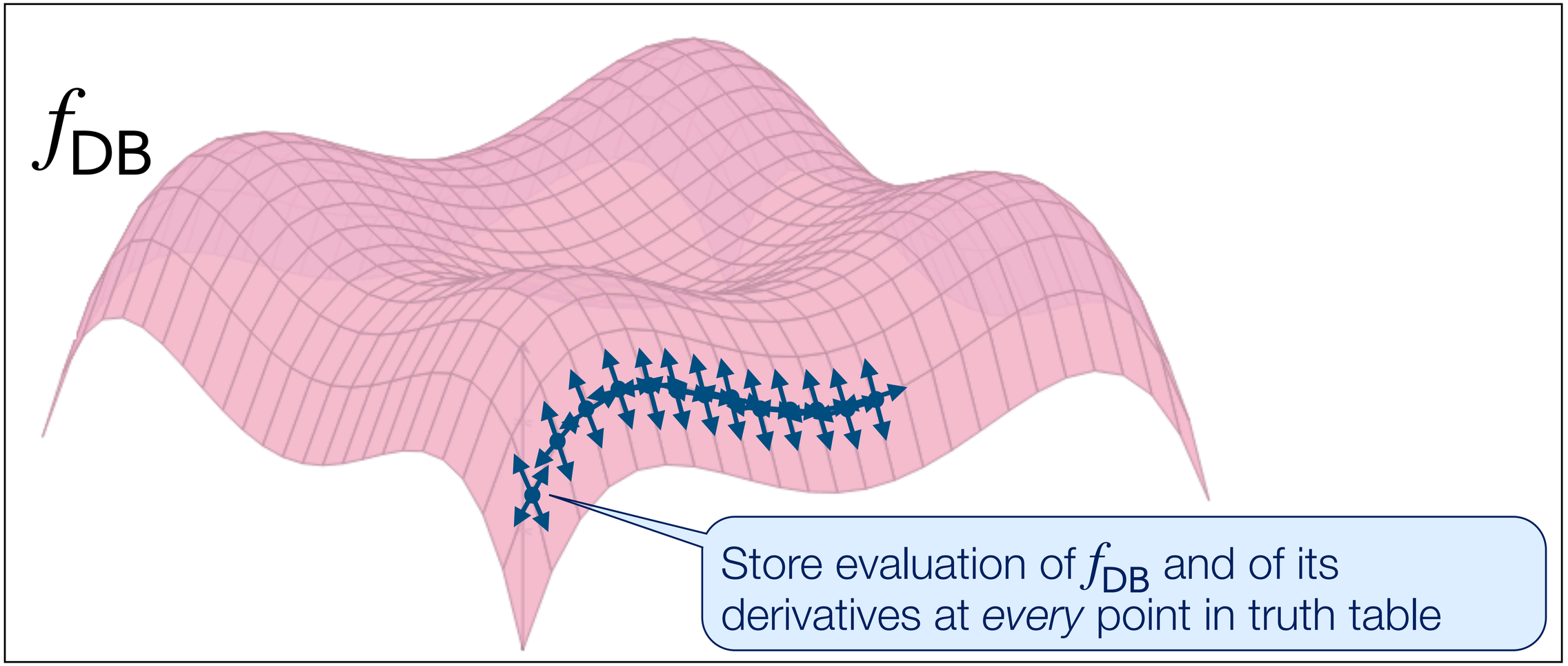
Finding Redundancy in the Brute-Force Data Structure



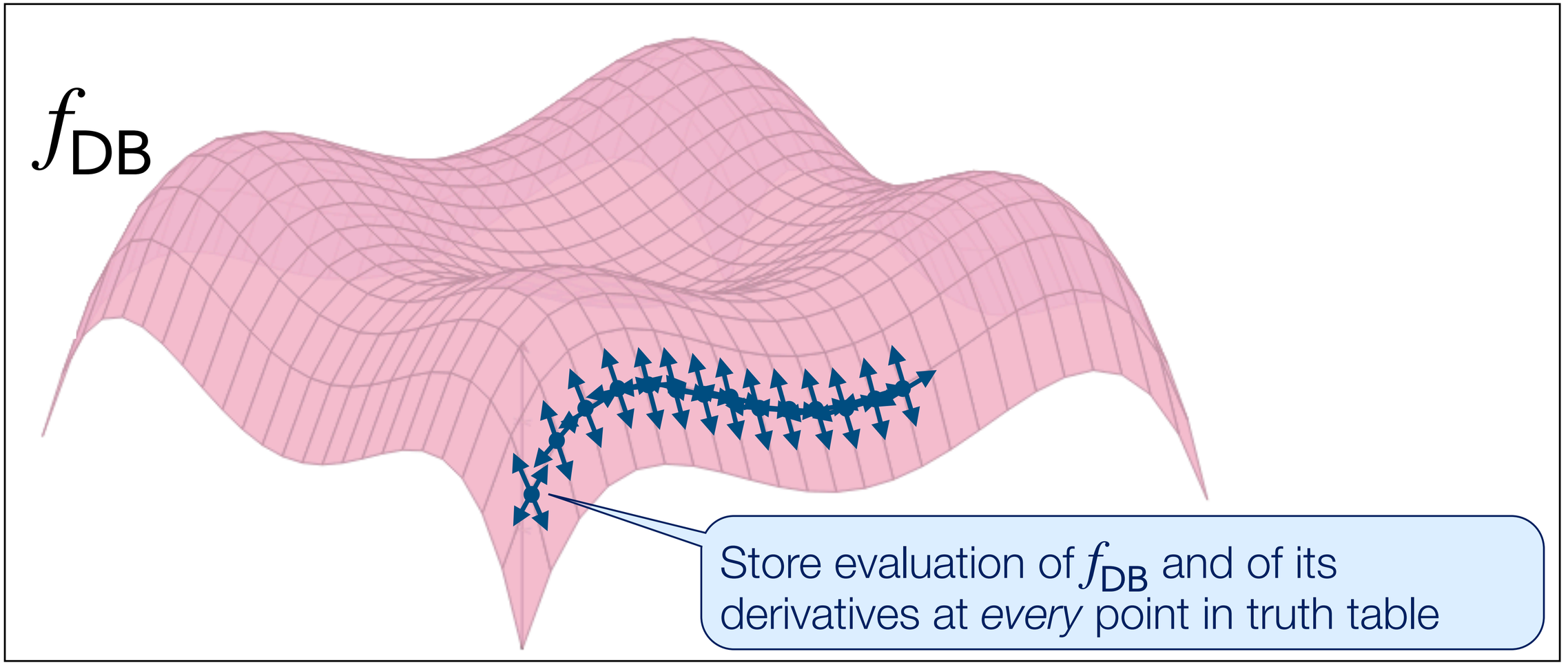
Finding Redundancy in the Brute-Force Data Structure



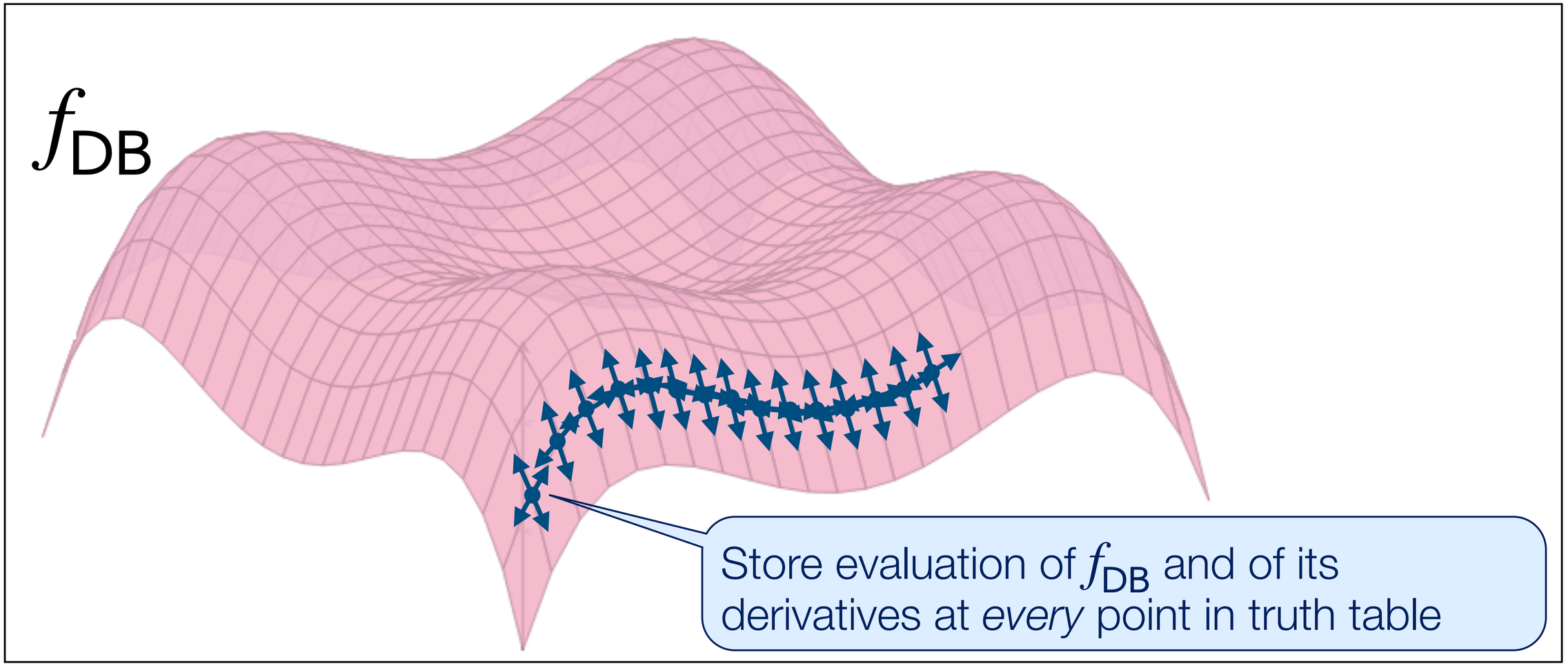
Finding Redundancy in the Brute-Force Data Structure



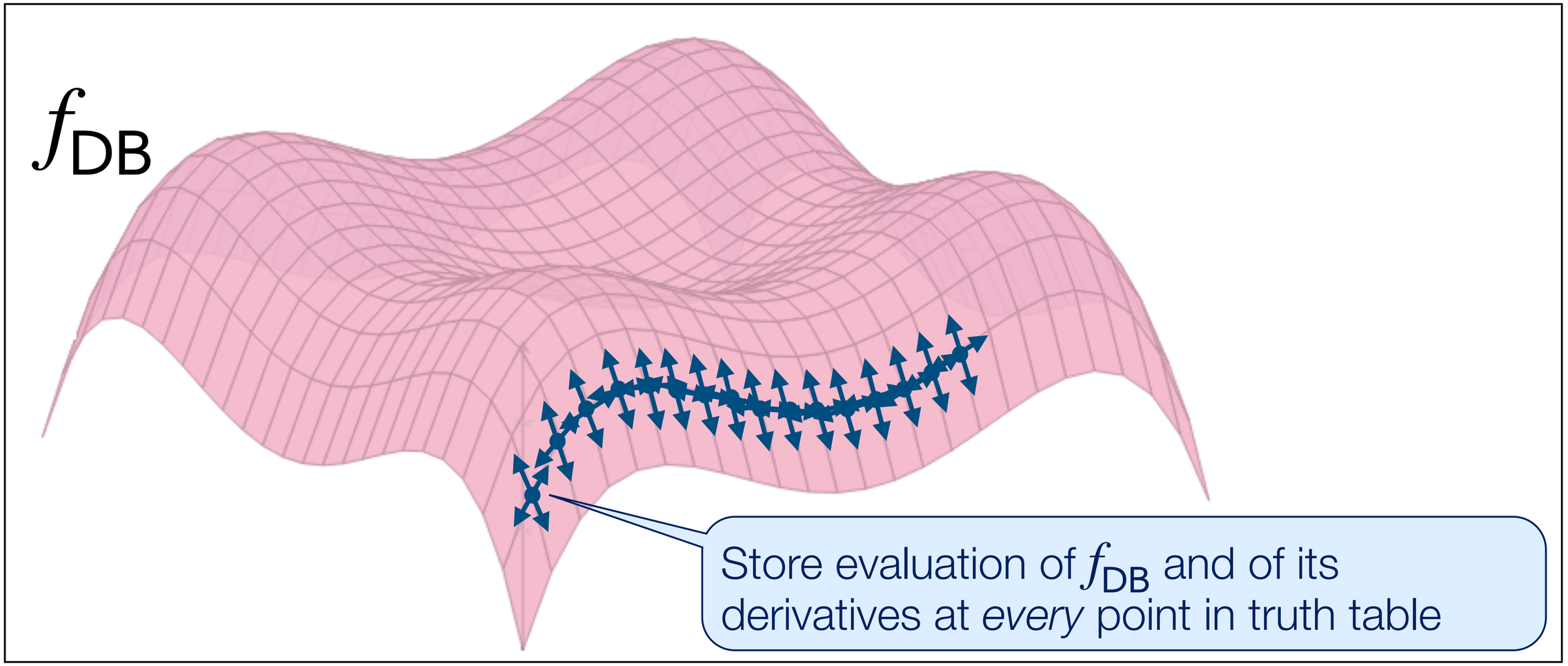
Finding Redundancy in the Brute-Force Data Structure



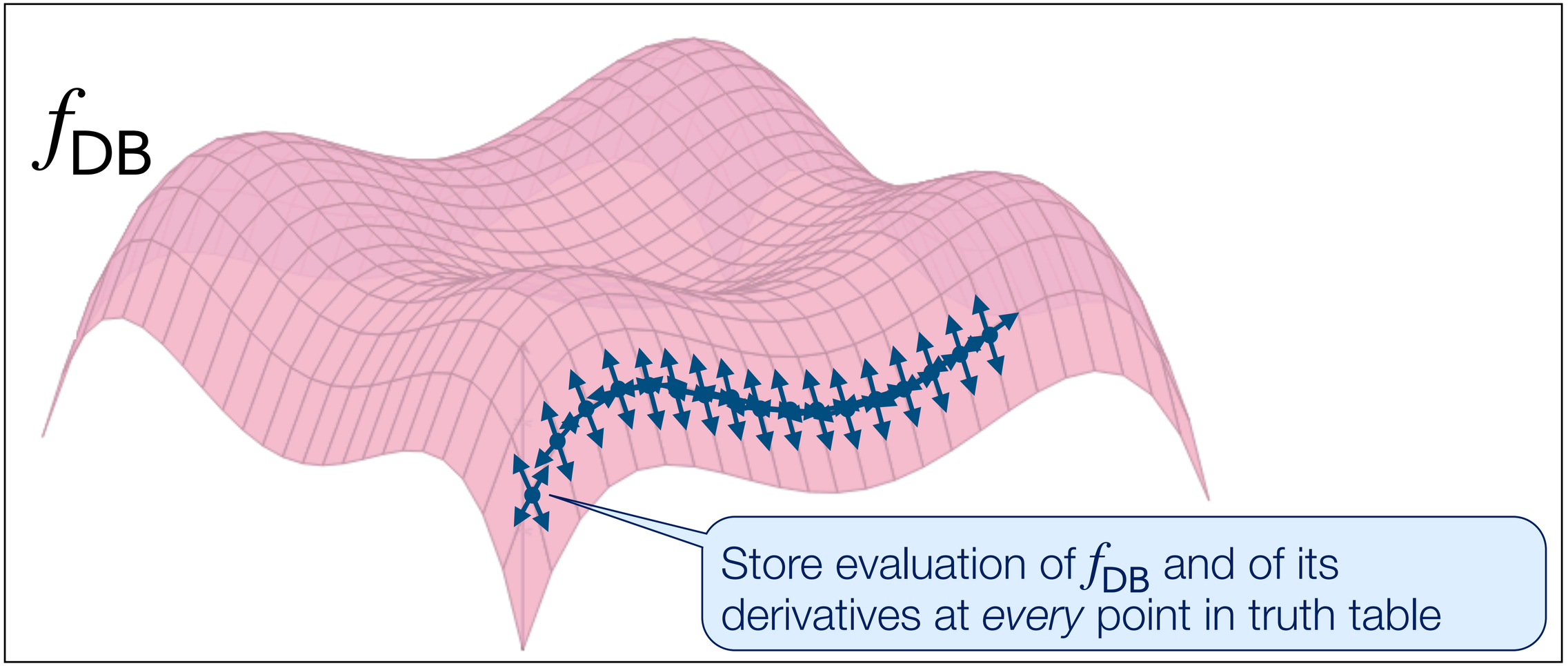
Finding Redundancy in the Brute-Force Data Structure



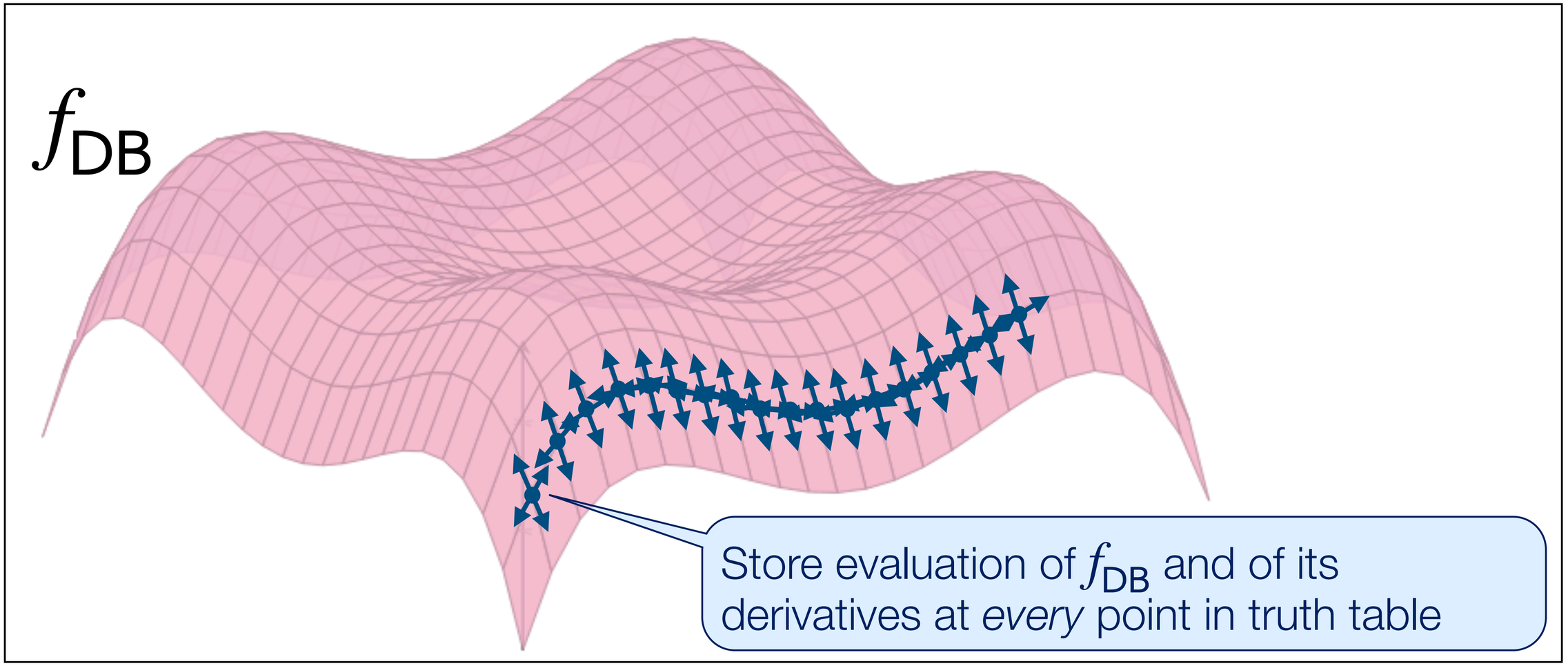
Finding Redundancy in the Brute-Force Data Structure



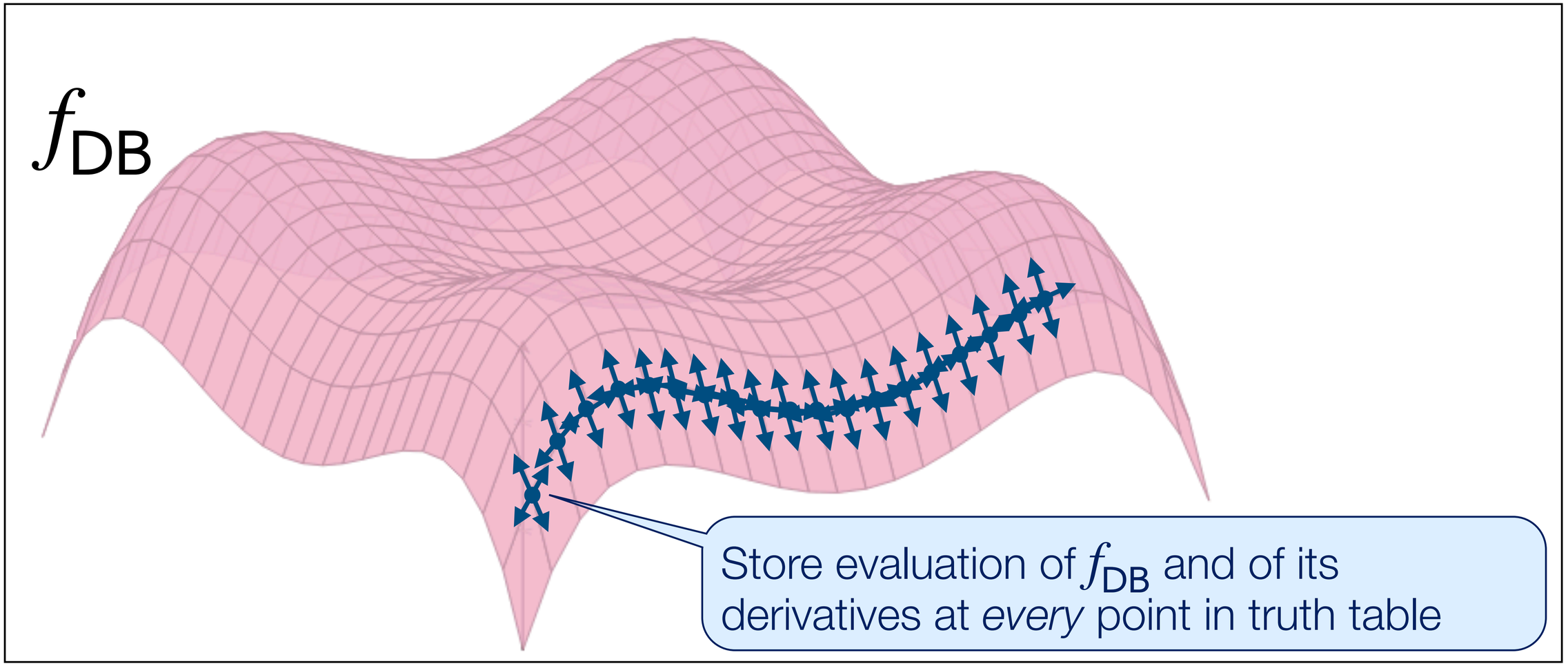
Finding Redundancy in the Brute-Force Data Structure



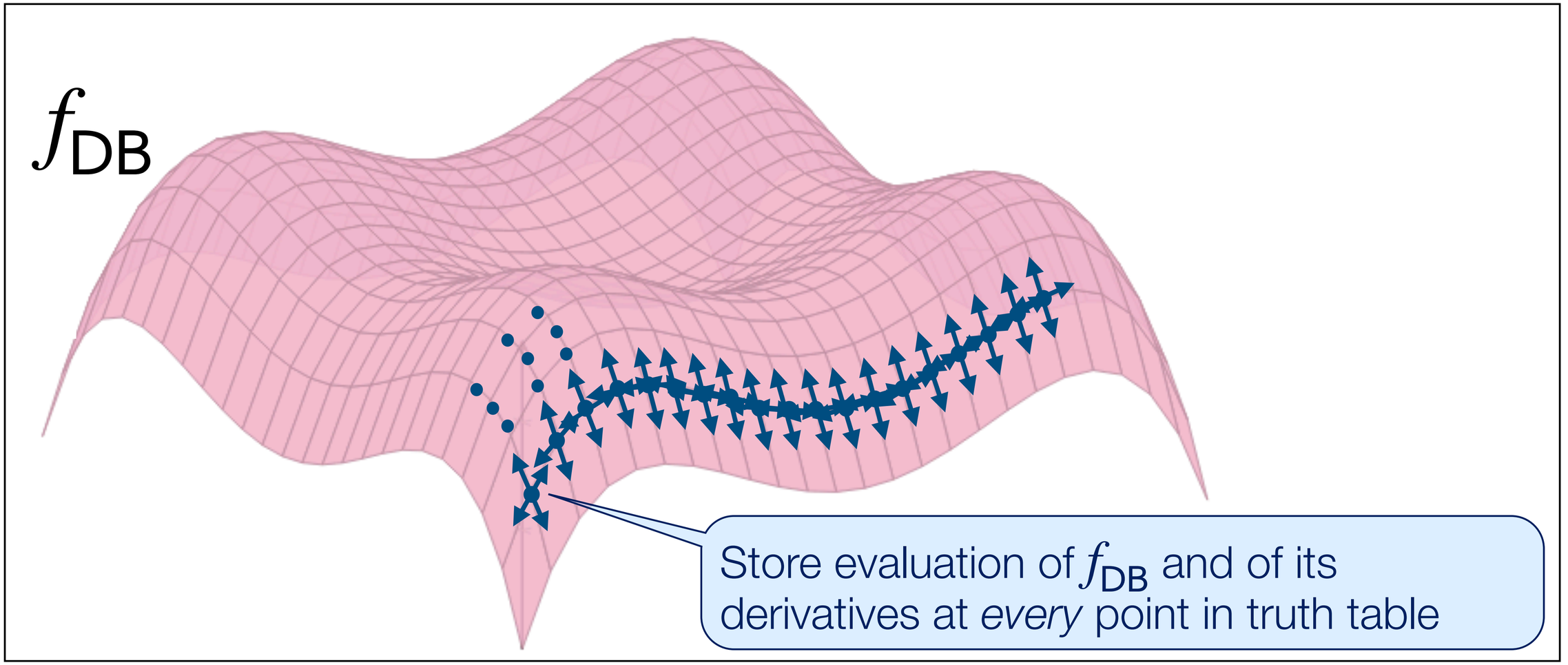
Finding Redundancy in the Brute-Force Data Structure



Finding Redundancy in the Brute-Force Data Structure



Finding Redundancy in the Brute-Force Data Structure



Fact 0. For a linear function $f : \mathbb{F} \to \mathbb{F}$, we have

$$f'(x) = f(x+1) - f(x)\,.$$

Fact 0. For a linear function $f : \mathbb{F} \to \mathbb{F}$, we have

$$f'(x) = f(x+1) - f(x)\,.$$

Fact 1. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\nabla f_{\mathsf{DB}}(\mathbf{x}) = \begin{bmatrix} \vdots \\ f_{\mathsf{DB}}(\mathbf{x} + \mathbf{u}_i) - f_{\mathsf{DB}}(\mathbf{x}) \\ \vdots \end{bmatrix} \in \mathbb{F}_2^m$$

Fact 0. For a linear function $f : \mathbb{F} \to \mathbb{F}$, we have

$$f'(x) = f(x+1) - f(x)\,.$$

Fact 1. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\nabla f_{\mathsf{DB}}(\mathbf{x}) = \begin{bmatrix} \vdots \\ f_{\mathsf{DB}}(\mathbf{x} + \mathbf{u}_i) - f_{\mathsf{DB}}(\mathbf{x}) \\ \vdots \end{bmatrix} \in \mathbb{F}_2^m$$

In other words: anyone can deduce $\nabla f_{\mathsf{DB}}(\mathbf{x})$ from the evaluations of f_{DB} in a Hamming ball of radius 1 around point $\mathbf{x}$.

Fact 0. For a linear function $f : \mathbb{F} \to \mathbb{F}$, we have

$$f'(x) = f(x+1) - f(x)\,.$$

Fact 1. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\nabla f_{\mathsf{DB}}(\mathbf{x}) = \begin{bmatrix} \vdots \\ f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i) - f_{\mathsf{DB}}(\mathbf{x}) \\ \vdots \end{bmatrix} \in \mathbb{F}_2^m$$

Fact 2. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\nabla^2 f_{\mathsf{DB}}(\mathbf{x}) = \begin{bmatrix} f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i+\mathbf{u}_j) - f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i) - f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_j) + f_{\mathsf{DB}}(\mathbf{x}) & \cdots \\ \vdots & \ddots \end{bmatrix}$$

Fact 0. For a linear function $f : \mathbb{F} \to \mathbb{F}$, we have

$$f'(x) = f(x+1) - f(x)\,.$$

Fact 1. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\begin{bmatrix} & \vdots & \end{bmatrix}$$

In other words: anyone can deduce $\nabla^2 f_{\mathsf{DB}}(\mathbf{x})$ from the evaluations of f_{DB} in a Hamming ball of radius 2 around point $\mathbf{x}$.

Fact 2. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\nabla^2 f_{\mathsf{DB}}(\mathbf{x}) = \begin{bmatrix} f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i+\mathbf{u}_j) - f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i) - f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_j) + f_{\mathsf{DB}}(\mathbf{x}) & \cdots \\ \vdots & \ddots \end{bmatrix}$$

Fact 0. For a linear function $f : \mathbb{F} \to \mathbb{F}$, we have

$$f'(x) = f(x+1) - f(x)\,.$$

Fact 1. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\nabla f_{\mathsf{DB}}(\mathbf{x}) = \begin{bmatrix} \vdots \\ f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i) - f_{\mathsf{DB}}(\mathbf{x}) \\ \vdots \end{bmatrix} \in \mathbb{F}_2^m$$

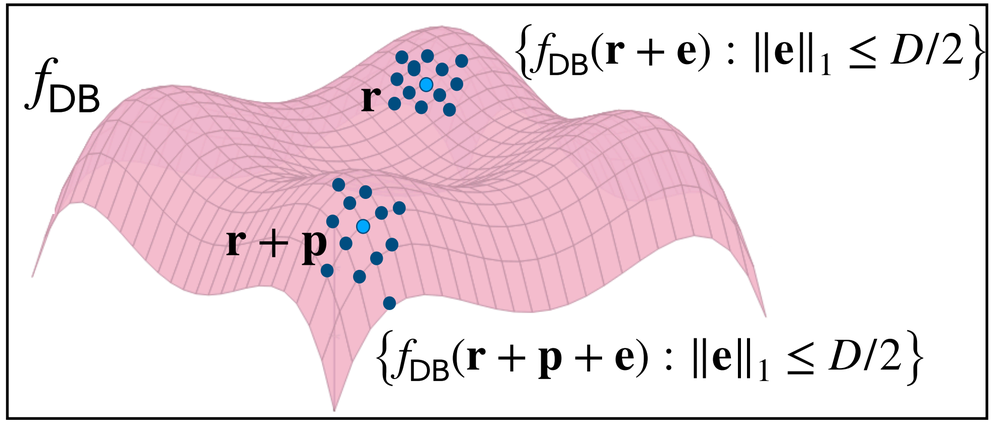
Fact 2. Since f_{DB} is multilinear, for any evaluation point $\mathbf{x} \in \mathbb{F}_2^m$,

$$\nabla^2 f_{\mathsf{DB}}(\mathbf{x}) = \begin{bmatrix} f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i+\mathbf{u}_j) - f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_i) - f_{\mathsf{DB}}(\mathbf{x}+\mathbf{u}_j) + f_{\mathsf{DB}}(\mathbf{x}) & \cdots \\ \vdots & \ddots \end{bmatrix}$$

Idea: Save storage by using evaluations in Hamming balls instead of derivatives.

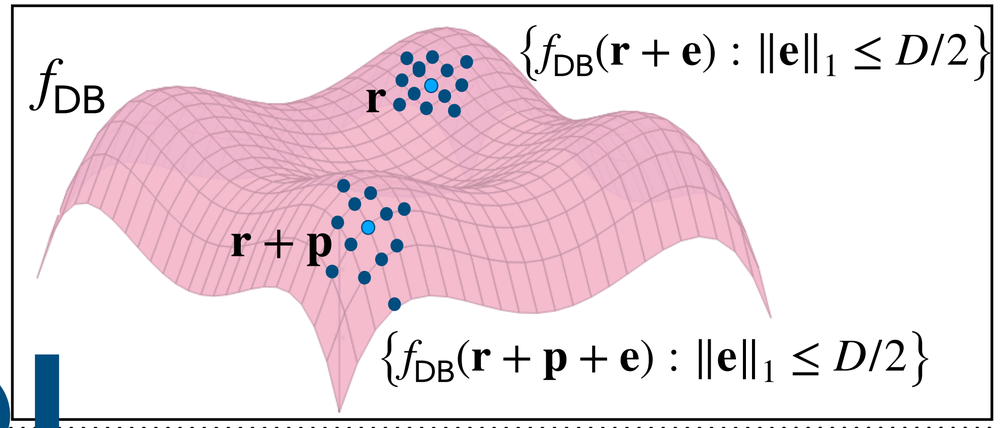
Idea: Save storage by using evaluations in Hamming balls instead of derivatives.

On query points $\mathbf{r}$ and $\mathbf{r}+\mathbf{p}$, the servers send back:



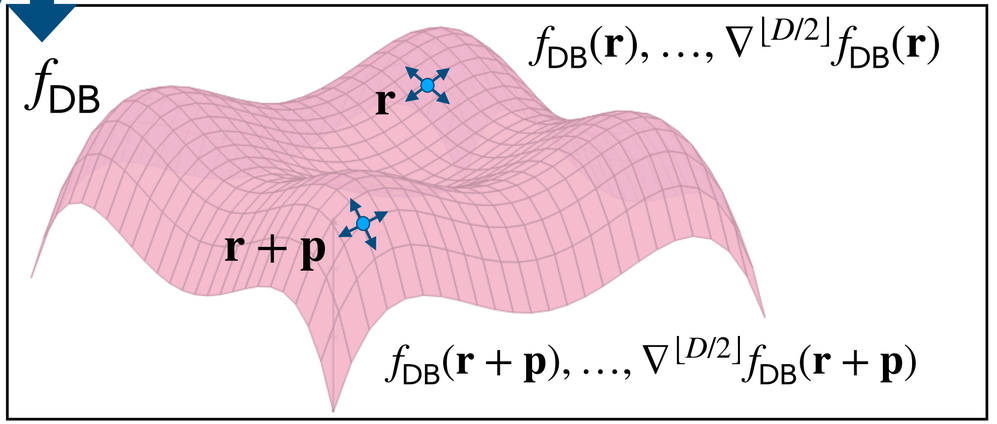
Idea: Save storage by using evaluations in Hamming balls instead of derivatives.

On query points $\mathbf{r}$ and $\mathbf{r}+\mathbf{p}$, the servers send back:



From these replies, the user computes:

1. Partial derivatives of f_{DB} at $\mathbf{r}, \mathbf{r}+\mathbf{p}$ (via finite differences)



Idea: Save storage by using evaluations in Hamming balls instead of derivatives.

On query points $\mathbf{r}$ and $\mathbf{r}+\mathbf{p}$, the servers send back:

f_{DB} $\mathbf{r}$ $\mathbf{r}+\mathbf{p}$ $\{f_{\mathrm{DB}}(\mathbf{r}+\mathbf{e}) : \|\mathbf{e}\|_1 \le D/2\}$ $\{f_{\mathrm{DB}}(\mathbf{r}+\mathbf{p}+\mathbf{e}) : \|\mathbf{e}\|_1 \le D/2\}$

1.

From these replies, the user computes:

1. Partial derivatives of f_{DB} at $\mathbf{r}, \mathbf{r}+\mathbf{p}$ (via finite differences)

f_{DB} $\mathbf{r}$ $\mathbf{r}+\mathbf{p}$ $f_{\mathrm{DB}}(\mathbf{r}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathrm{DB}}(\mathbf{r})$ $f_{\mathrm{DB}}(\mathbf{r}+\mathbf{p}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathrm{DB}}(\mathbf{r}+\mathbf{p})$

2.

2. $f_{\mathrm{DB}}(\mathbf{p})$ (via Hermite interpolation as before)

f_{DB} $\mathbf{p}$ $f_{\mathrm{DB}}(\mathbf{p})$

Our Resulting PIR with Preprocessing

$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m})$

Query: $\mathbf{r}$

Ans:
$\{f_{\mathsf{DB}}(\mathbf{r}+\mathbf{e}) : \|\mathbf{e}\| \leq D/2\}$

Query: $\mathbf{p}+\mathbf{r}$

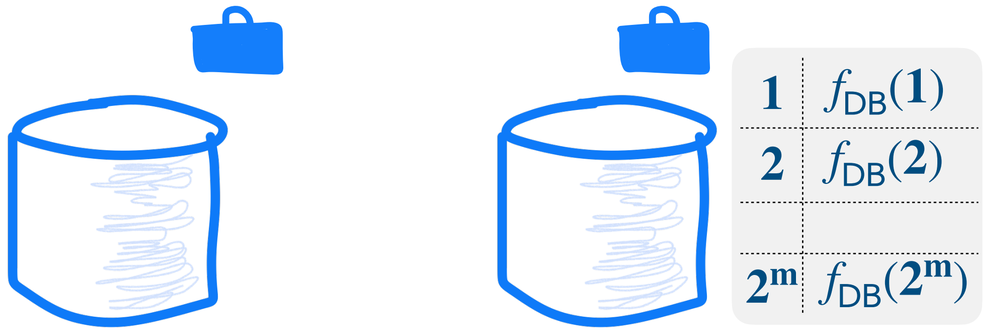
Ans:
$\{f_{\mathsf{DB}}(\mathbf{r}+\mathbf{p}+\mathbf{e}) : \|\mathbf{e}\| \leq D/2\}$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line
$L(t) = \mathbf{r} + t \cdot \mathbf{p}$

Recover $f_{\mathsf{DB}}(\mathbf{p})$ via finite differences and Hermite interpolation

Our Resulting PIR with Preprocessing



$\mathbf{1}$	$f_{\mathrm{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathrm{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathrm{DB}}(\mathbf{2^m})$

Query: $\mathbf{r}$

Ans: $\{f_{\mathrm{DB}}(\mathbf{r}+\mathbf{e}) : \|\mathbf{e}\| \le D/2\}$

Query: $\mathbf{p}+\mathbf{r}$

Ans: $\{f_{\mathrm{DB}}(\mathbf{r}+\mathbf{p}+\mathbf{e}) : \|\mathbf{e}\| \le D/2\}$

Point $\mathbf{p} \in \mathbb{F}_2^m$

Sample line $L(t) = \mathbf{r} + t \cdot \mathbf{p}$

Recover $f_{\mathrm{DB}}(\mathbf{p})$ via finite differences and Hermite interpolation

With 2 servers, gives preprocessing PIR with

- ➡ Same comm. as [BIM00]: query $O(\log n)$ answer $n^{0.82}$
- ➡ Same time as [BIM00]: $O(n^{0.82})$ work
- ➡ Quasilinear space: $2^m = n^{1+o(1)}$ bits

Theorem. On any database of $n > 10^6$ bits, there exists information-theoretic, two-server PIR with preprocessing with:

- $1.5 \cdot \sqrt{\log n} \cdot n$ bits of server storage,
- $12 \cdot n^{0.82}$ server RAM lookups per query, and
- $12 \cdot n^{0.82}$ bits of communication per query.

Concrete Evaluation: 2GB Database

Construction	Storage	Memory accessed per query
Our work	1 TB	0.7 MB
[BIM00]	760000 TB	0.7 MB
XOR PIR [CGKS95]	2 GB (DB only)	2 GB (entire DB)

Concrete Evaluation: 2GB Database

Construction	Storage	Memory accessed per query	Communication
Our work	1 TB	0.7 MB	0.7 MB
[BIM00]	760000 TB	0.7 MB	0.7 MB
XOR PIR [CGKS95]	2 GB (DB only)	2 GB (entire DB)	0.05 MB

Concrete Evaluation: 2GB Database

Construction	Storage	Memory accessed per query	Communication	Throughput (queries/s)
Our work	1 TB	0.7 MB	0.7 MB	3804
[BIM00]	760000 TB	0.7 MB	0.7 MB	
XOR PIR [CGKS95]	2 GB (DB only)	2 GB (entire DB)	0.05 MB	374

Summary

- First information-theoretic PIR with $n^{1+o(1)}$ storage, $n^{1-\Omega(1)}$ server time, and $O(1)$ servers

Summary

- First information-theoretic PIR with $n^{1+o(1)}$ storage, $n^{1-\Omega(1)}$ server time, and $O(1)$ servers
  - Two servers, server time $n^{0.82}$

Summary

- First information-theoretic PIR with $n^{1+o(1)}$ storage, $n^{1-\Omega(1)}$ server time, and $O(1)$ servers
  - Two servers, server time $n^{0.82}$
- Not covered in this talk:
  - Shrinking the communication with linearly or fully homomorphic encryption

Summary

- First information-theoretic PIR with $n^{1+o(1)}$ storage, $n^{1-\Omega(1)}$ server time, and $O(1)$ servers
  - Two servers, server time $n^{0.82}$
- Not covered in this talk:
  - Shrinking the communication with linearly or fully homomorphic encryption
  - Additional improvements for > 2 servers

Summary

- First information-theoretic PIR with $n^{1+o(1)}$ storage, $n^{1-\Omega(1)}$ server time, and $O(1)$ servers
  - Two servers, server time $n^{0.82}$
- Not covered in this talk:
  - Shrinking the communication with linearly or fully homomorphic encryption
  - Additional improvements for > 2 servers
- Seems like it could be the first practically feasible PIR with preprocessing!

Summary

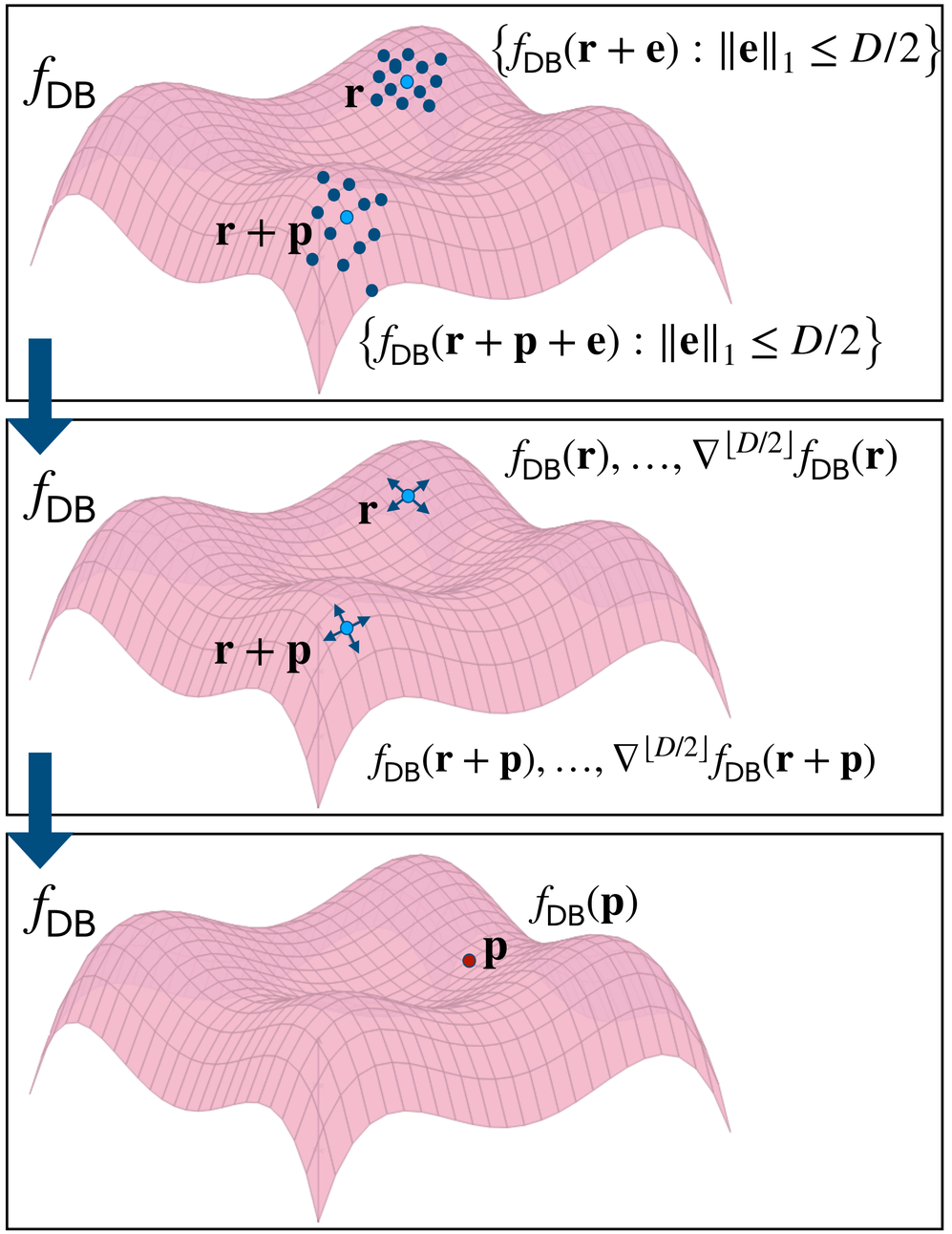
- First information-theoretic PIR with $n^{1+o(1)}$ storage, $n^{1-\Omega(1)}$ server time, and $O(1)$ servers
  - Two servers, server time $n^{0.82}$
- Not covered in this talk:
  - Shrinking the communication with linearly or fully homomorphic encryption
  - Additional improvements for > 2 servers
- Seems like it could be the first practically feasible PIR with preprocessing!
- Open question: our result but with $n^{o(1)}$ communication (maybe with $\mathbf{poly}(n)$ storage)?
  - Also practically relevant; would reduce network-related expenses and latency

Questions?

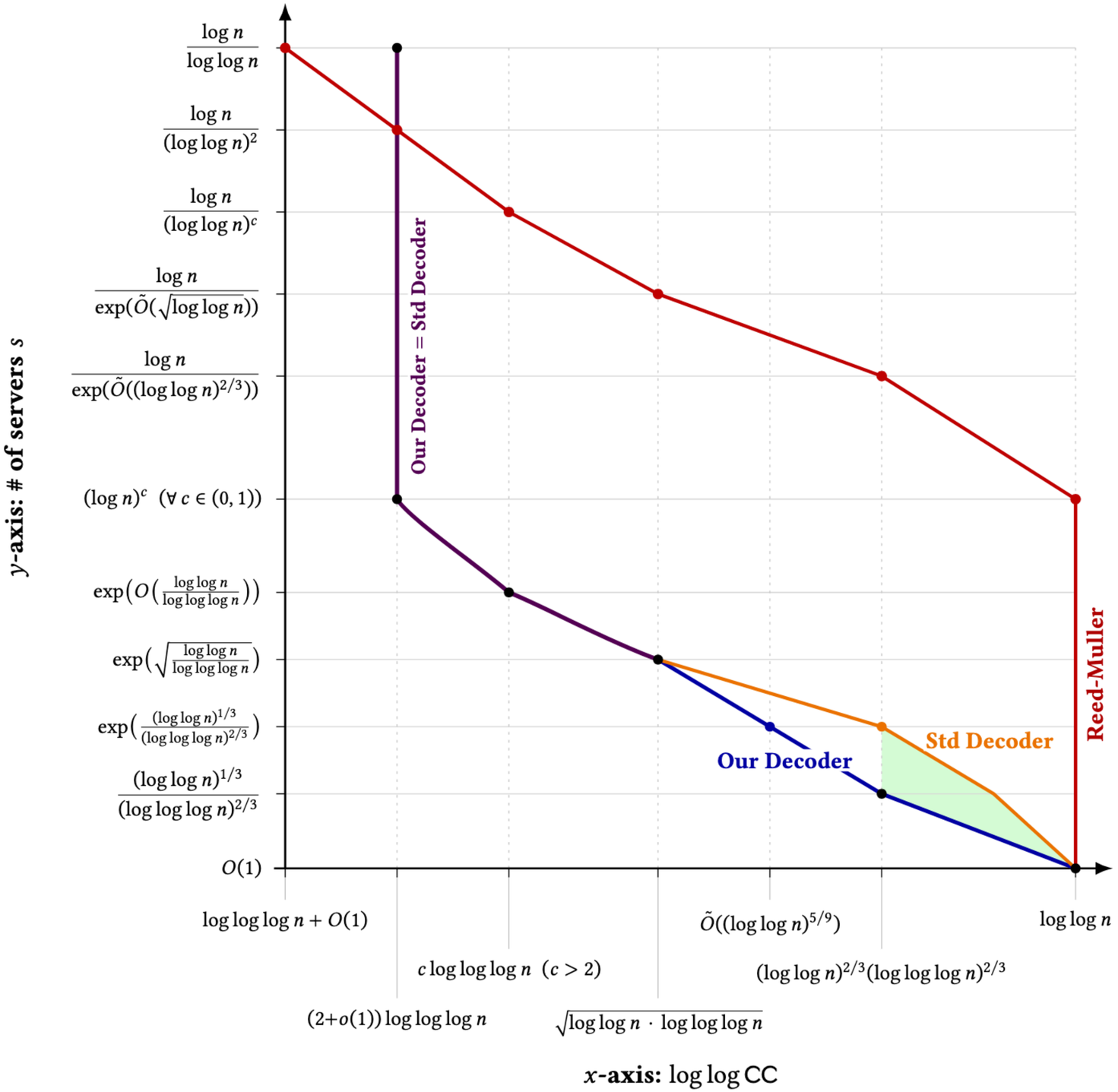
paper

code

ePrint: 2025/2008



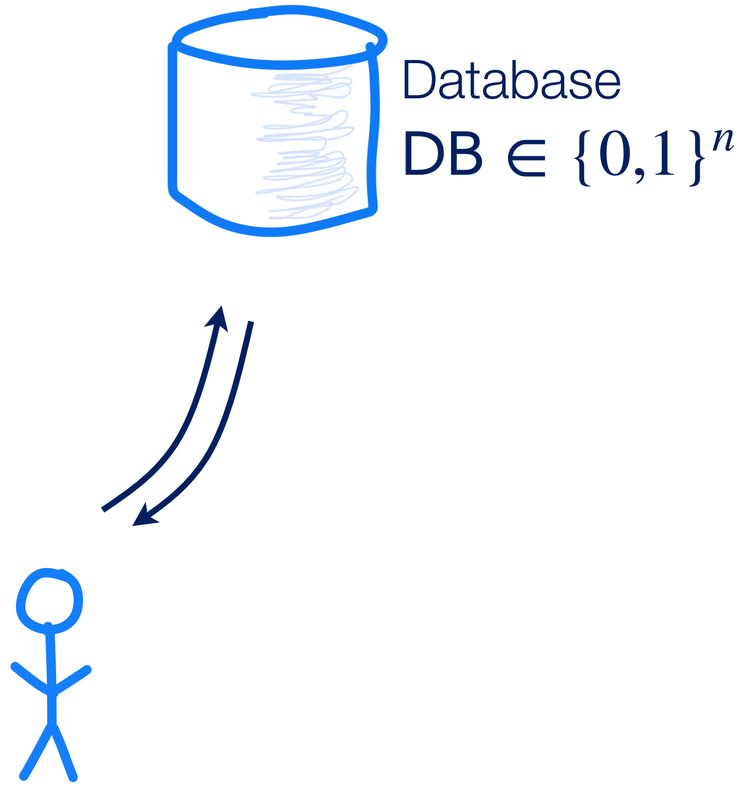
Bonus Slide on MV PIR with Optimal Decoding Polynomials


y-axis: # of servers s
$\frac{\log n}{\log\log n}$
$\frac{\log n}{(\log\log n)^2}$
$\frac{\log n}{(\log\log n)^c}$
$\frac{\log n}{\exp(\tilde{O}(\sqrt{\log\log n}))}$
$\frac{\log n}{\exp(\tilde{O}((\log\log n)^{2/3}))}$
$(\log n)^c \quad (\forall\, c \in (0,1))$
$\exp\left(O\left(\frac{\log\log n}{\log\log\log n}\right)\right)$
$\exp\left(\sqrt{\frac{\log\log n}{\log\log\log n}}\right)$
$\exp\left(\frac{(\log\log n)^{1/3}}{(\log\log\log n)^{2/3}}\right)$
$\frac{(\log\log n)^{1/3}}{(\log\log\log n)^{2/3}}$
$O(1)$
Our Decoder = Std Decoder
Reed-Muller
Std Decoder
Our Decoder
$\log\log\log n + O(1)$
$c \log\log\log n \quad (c > 2)$
$(2+o(1))\log\log\log n$
$\sqrt{\log\log n \cdot \log\log\log n}$
$\tilde{O}((\log\log n)^{5/9})$
$(\log\log n)^{2/3}(\log\log\log n)^{2/3}$
$\log\log n$
x-axis: log log CC

Bonus Slides on PIR with Preprocessing

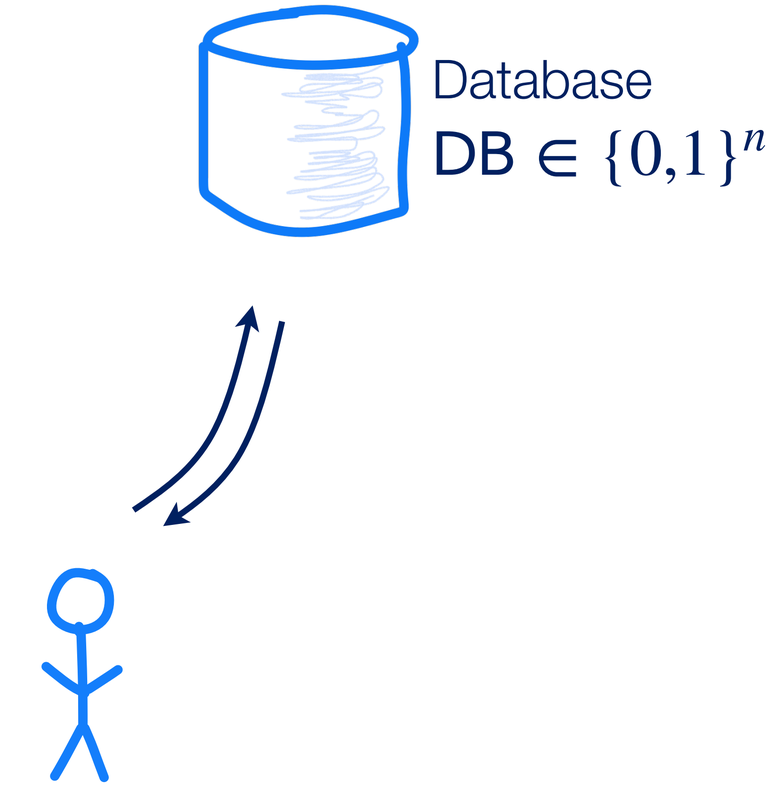
Bonus Slides on Reducing Communication Using Homomorphic Encryption

Homomorphic Encryption for Single-Server PIR



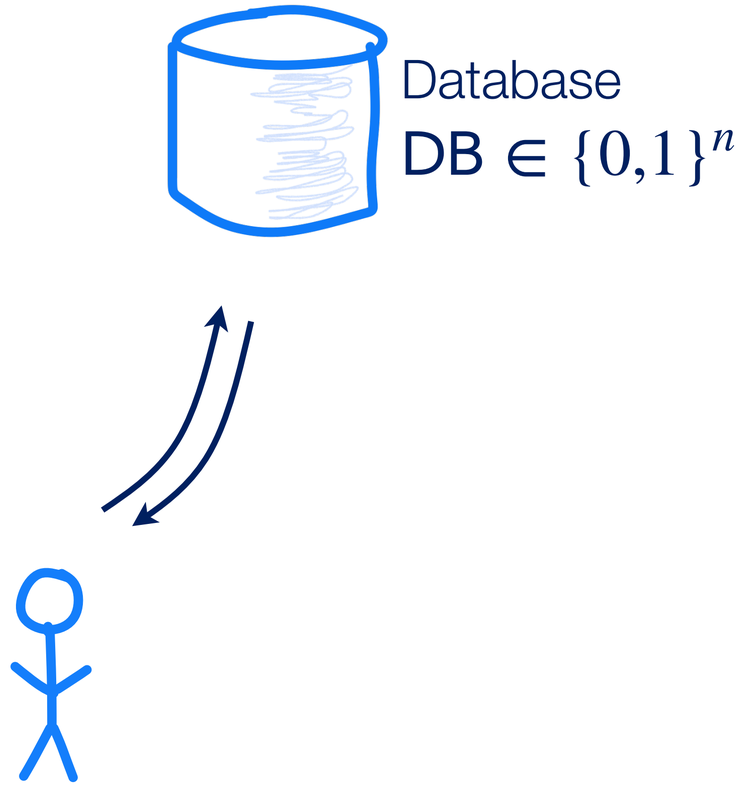
Homomorphic Encryption for Single-Server PIR

- PIR: computing $\mathsf{DB}[i]$ without revealing i



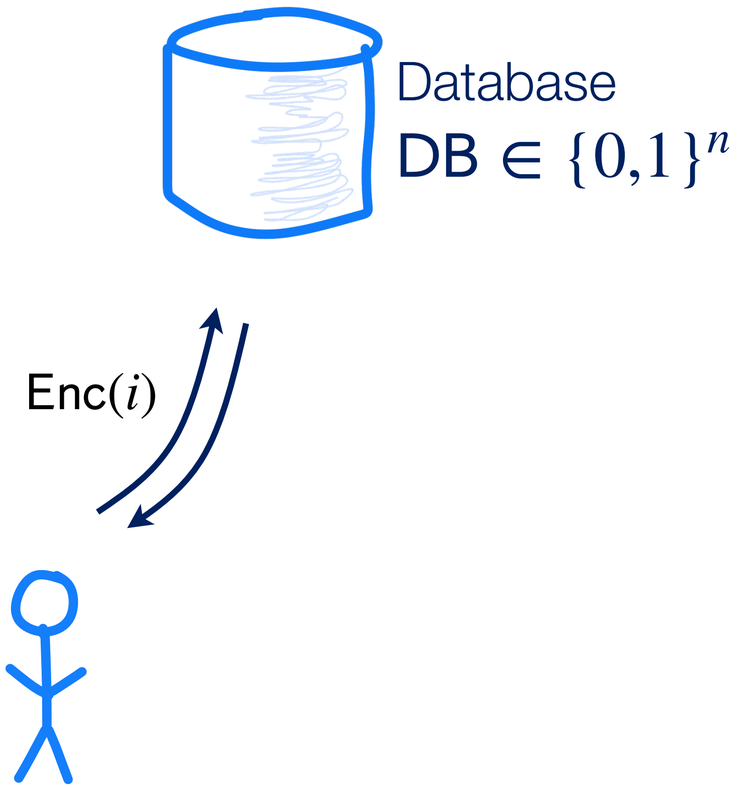
Homomorphic Encryption for Single-Server PIR

- PIR: computing $\mathsf{DB}[i]$ without revealing i
- Naive FHE solution:



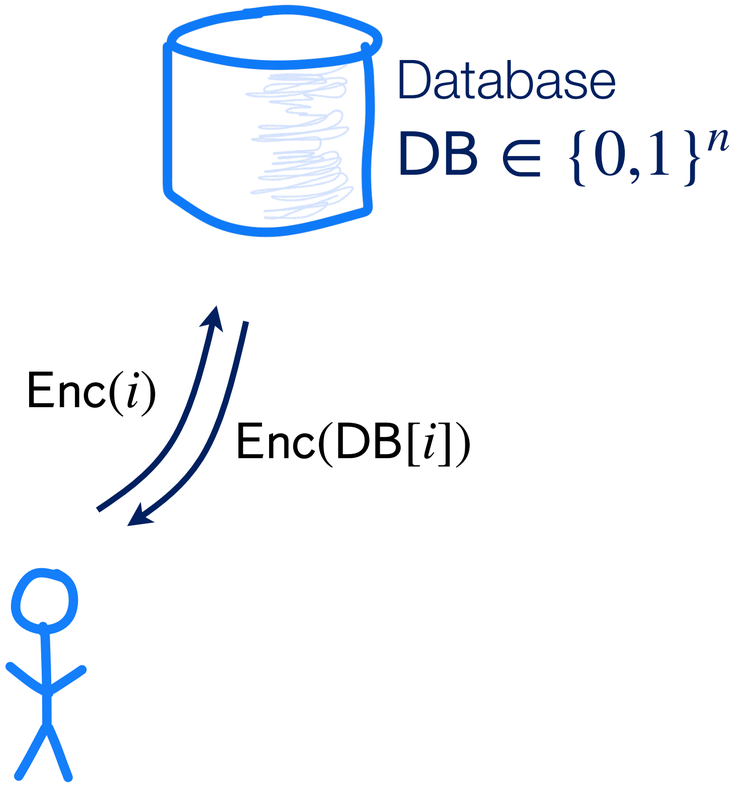
Homomorphic Encryption for Single-Server PIR

- PIR: computing $\mathsf{DB}[i]$ without revealing i
- Naive FHE solution:
  - Query: $\mathsf{ct} \leftarrow \mathsf{FHE}.\mathsf{Enc}(i)$



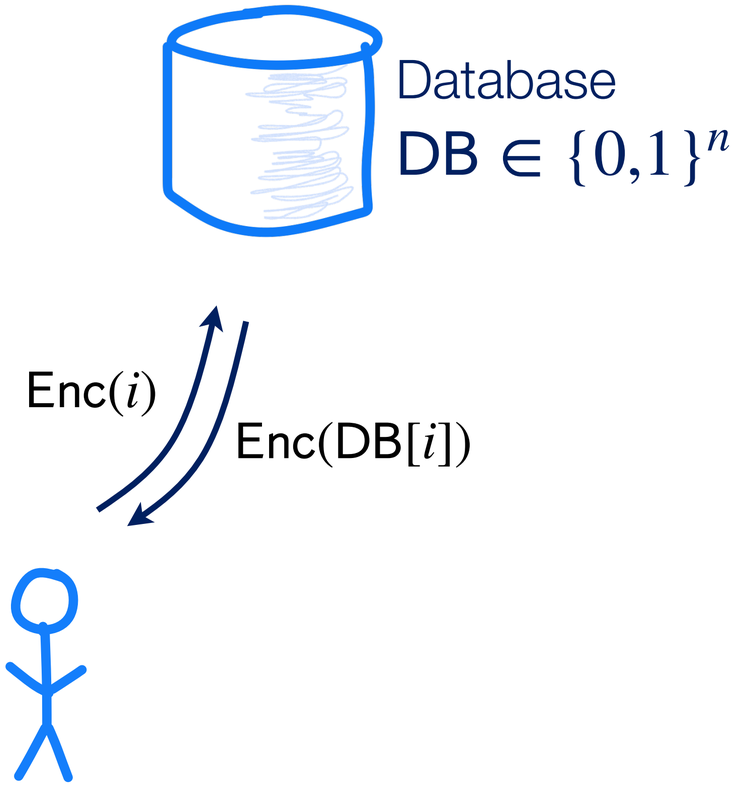
Homomorphic Encryption for Single-Server PIR

- PIR: computing $\mathsf{DB}[i]$ without revealing i
- Naive FHE solution:
  - Query: $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(i)$
  - Server answer: $\mathsf{FHE.Eval}(\mathsf{ct}, \mathsf{DB}[\,\cdot\,])$



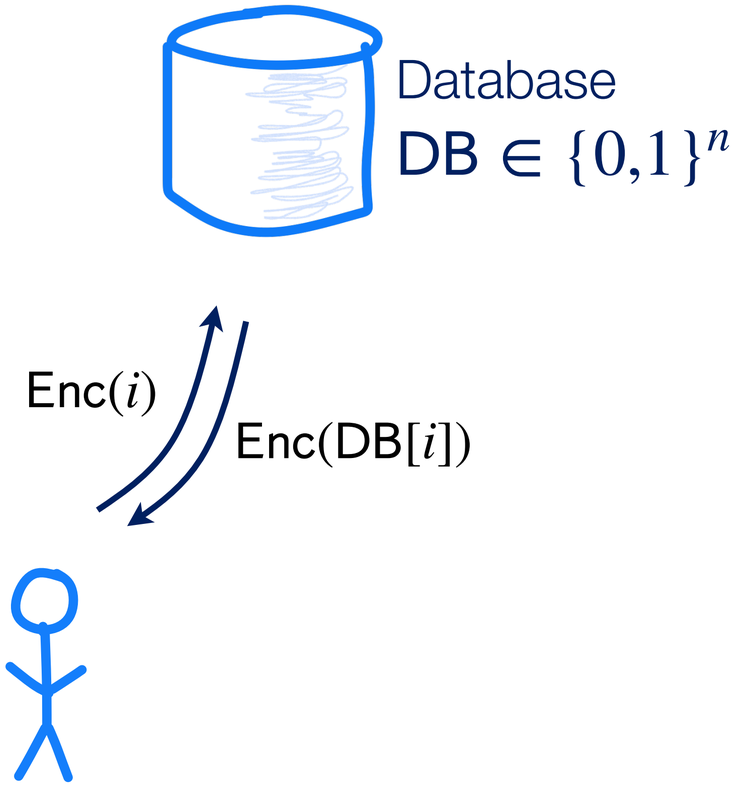
Homomorphic Encryption for Single-Server PIR

- PIR: computing $\mathsf{DB}[i]$ without revealing i
- Naive FHE solution:
  - Query: $\mathsf{ct} \leftarrow \mathsf{FHE}.\mathsf{Enc}(i)$
  - Server answer: $\mathsf{FHE}.\mathsf{Eval}(\mathsf{ct}, \mathsf{DB}[\,\cdot\,])$
  - User decrypts to learn $\mathsf{DB}[i]$



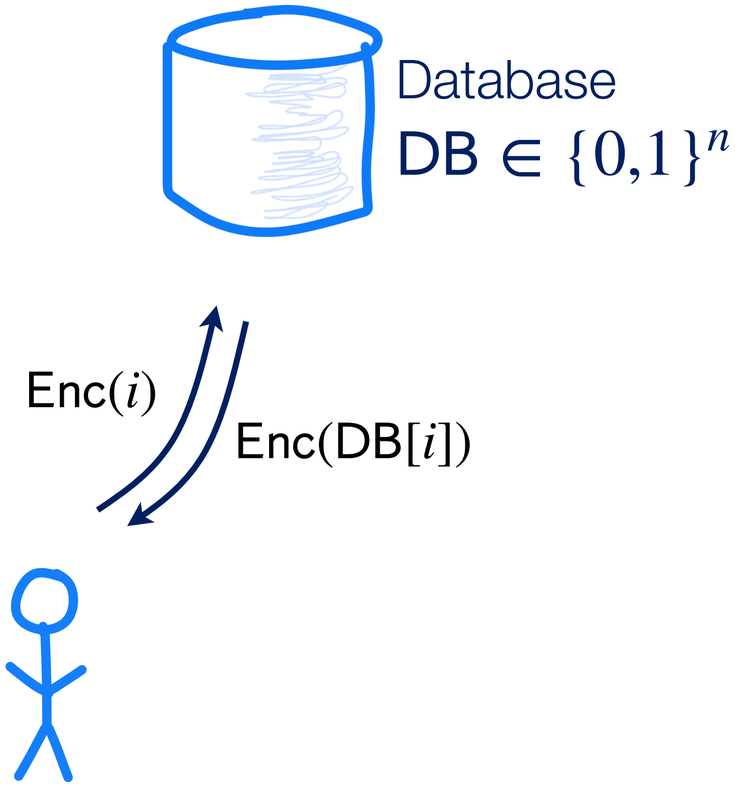
Homomorphic Encryption for Single-Server PIR

- PIR: computing $\mathsf{DB}[i]$ without revealing i
- Naive FHE solution:
  - Query: $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(i)$
  - Server answer: $\mathsf{FHE.Eval}(\mathsf{ct}, \mathsf{DB}[\,\cdot\,])$
  - User decrypts to learn $\mathsf{DB}[i]$
- But $\mathsf{DB}[\,\cdot\,]$ is a size $O(n)$ circuit!



Homomorphic Encryption for Single-Server PIR

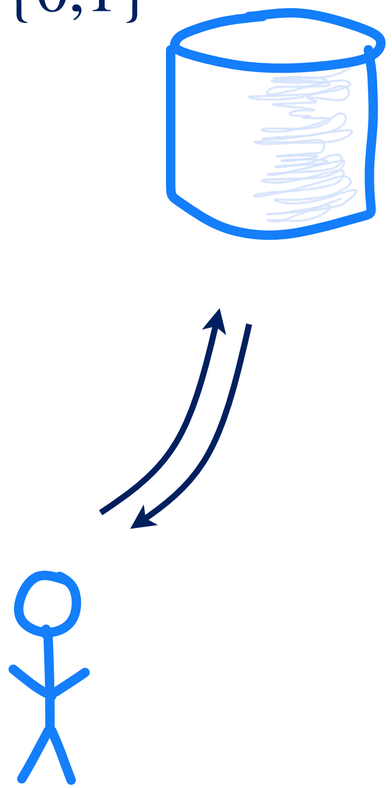
- PIR: computing $\mathsf{DB}[i]$ without revealing i
- Naive FHE solution:
  - Query: $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(i)$
  - Server answer: $\mathsf{FHE.Eval}(\mathsf{ct}, \mathsf{DB}[\,\cdot\,])$
  - User decrypts to learn $\mathsf{DB}[i]$
- But $\mathsf{DB}[\,\cdot\,]$ is a size $O(n)$ circuit!
  - Server time per query: $O(n)$



Homomorphic Encryption for 2-Server PIR

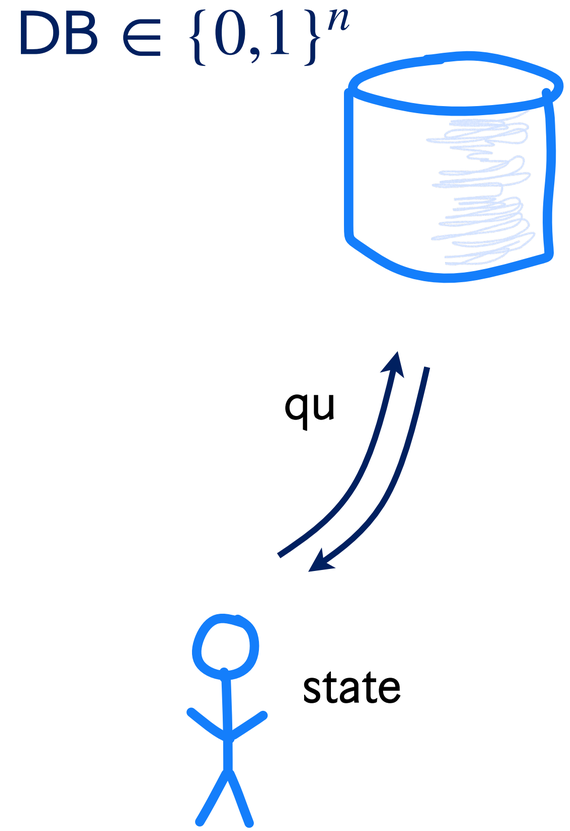
Database

DB $\in \{0,1\}^n$



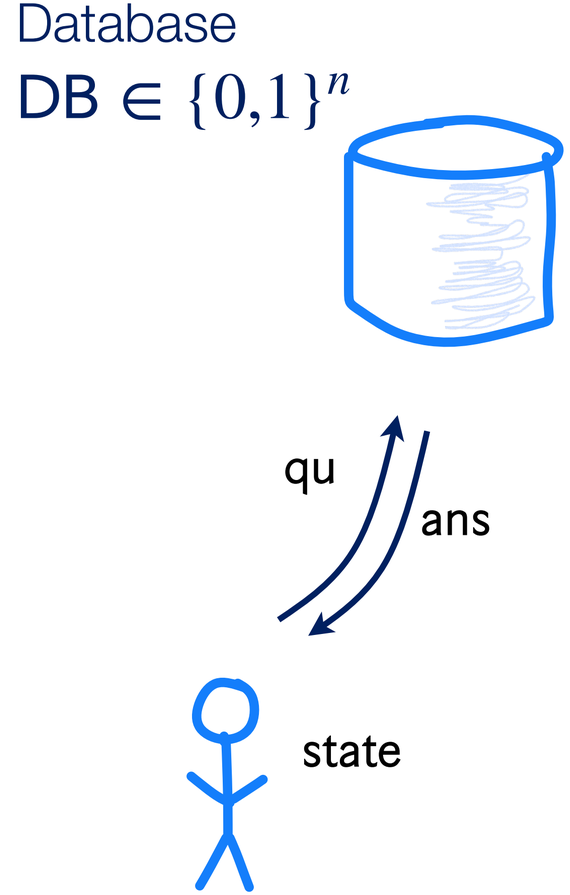
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$



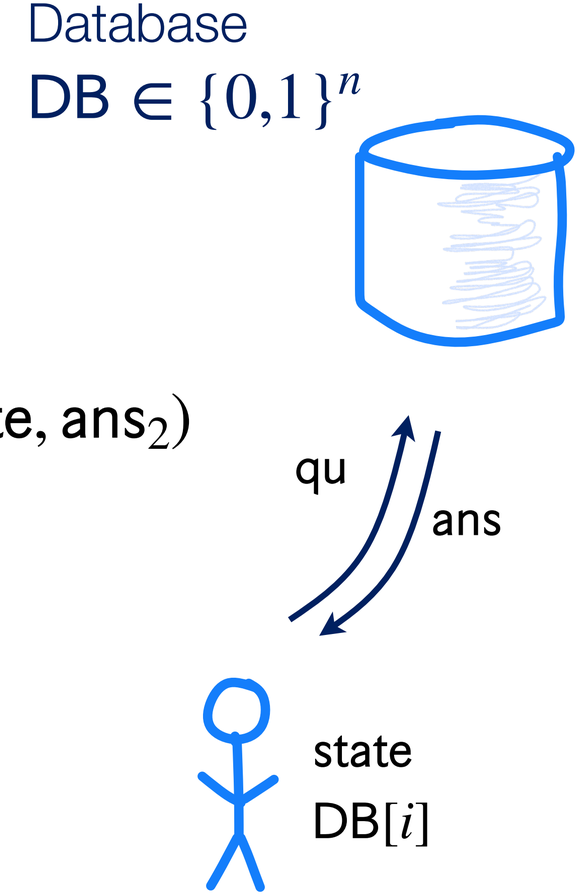
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$



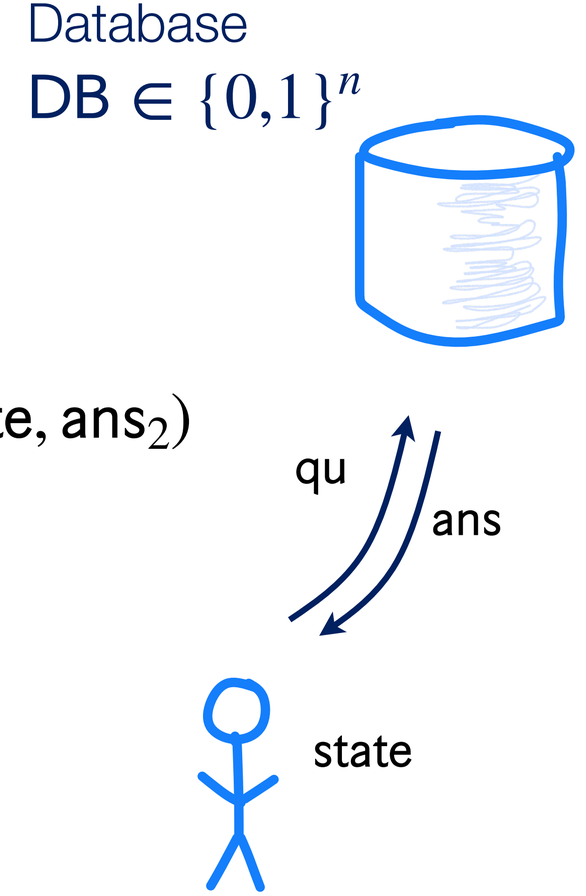
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$



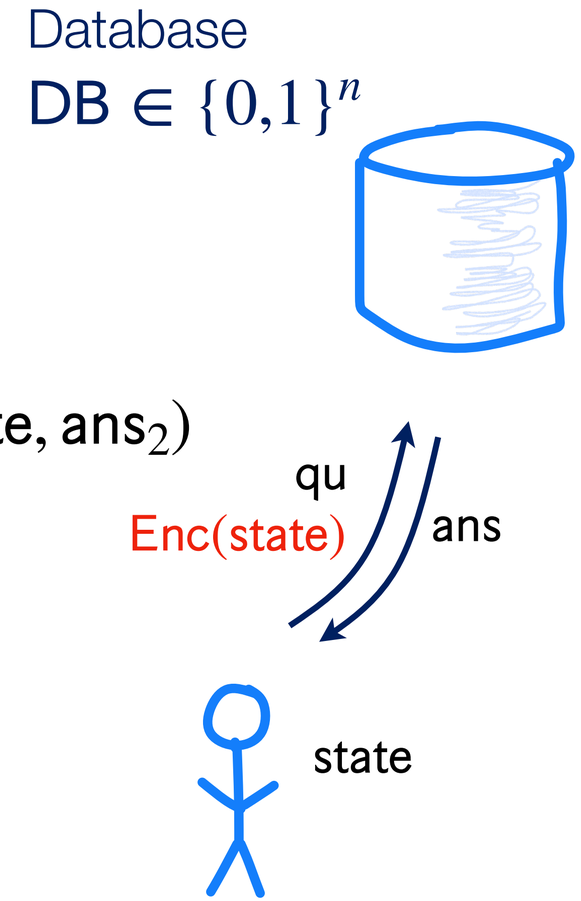
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$
- Observation (coming up): **Reconstruct** is a small circuit!



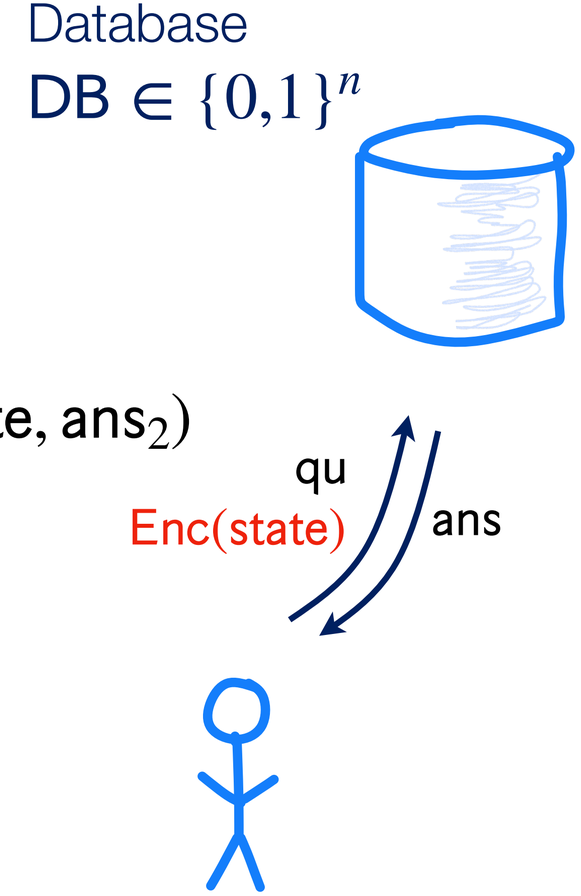
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$
- Observation (coming up): **Reconstruct** is a small circuit!
  - Include $\mathsf{ct} \leftarrow \mathsf{FHE}.\mathsf{Enc}(\mathsf{state})$ in query



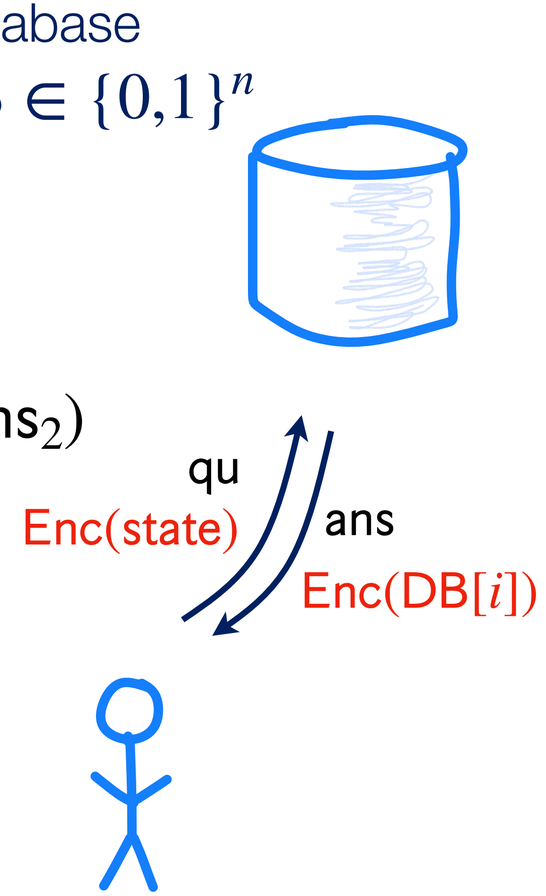
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$
- Observation (coming up): **Reconstruct** is a small circuit!
  - Include $\mathsf{ct} \leftarrow \mathsf{FHE}.\mathsf{Enc}(\mathsf{state})$ in query



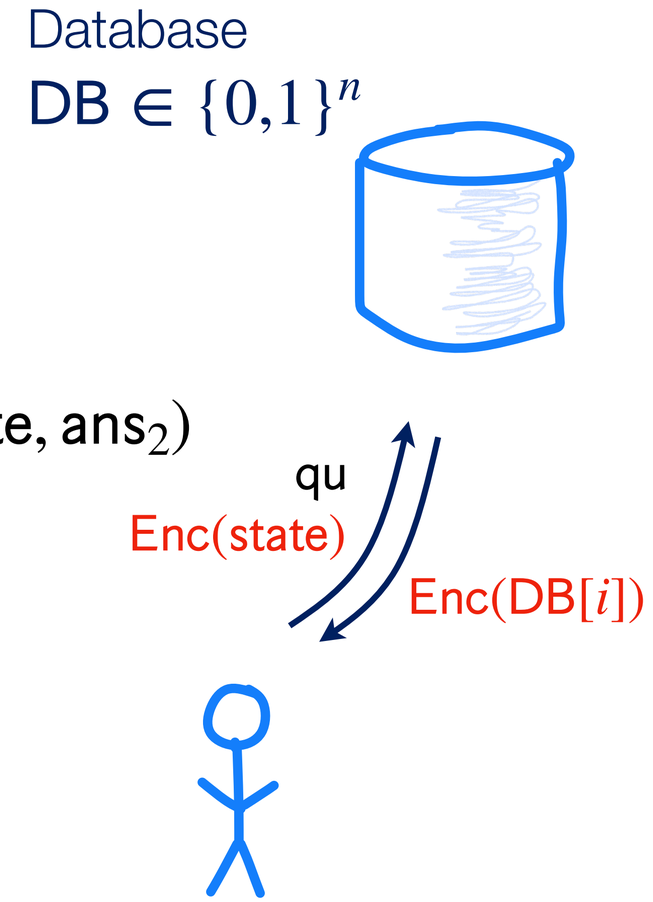
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$
- Observation (coming up): **Reconstruct** is a small circuit!
  - Include $\mathsf{ct} \leftarrow \mathsf{FHE}.\mathsf{Enc}(\mathsf{state})$ in query
  - Server i computes ans_i then $\mathsf{FHE}.\mathsf{Eval}(\mathsf{ct}, \mathsf{Reconstruct}(\mathsf{ans}_i, \cdot))$



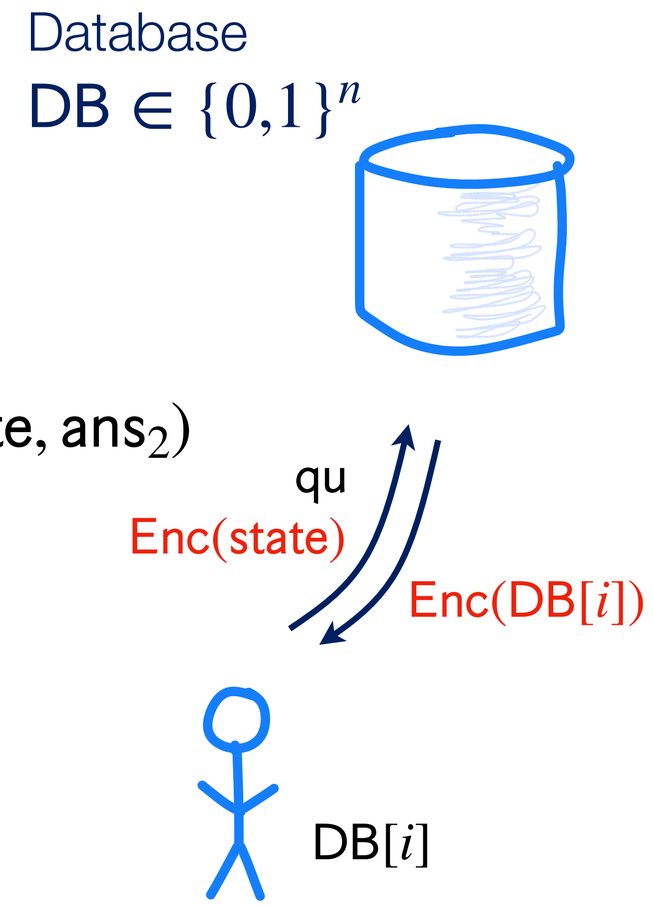
Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$
- Observation (coming up): **Reconstruct** is a small circuit!
  - Include $\mathsf{ct} \leftarrow \mathsf{FHE}.\mathsf{Enc}(\mathsf{state})$ in query
  - Server i computes ans_i then $\mathsf{FHE}.\mathsf{Eval}(\mathsf{ct}, \mathsf{Reconstruct}(\mathsf{ans}_i, \cdot))$



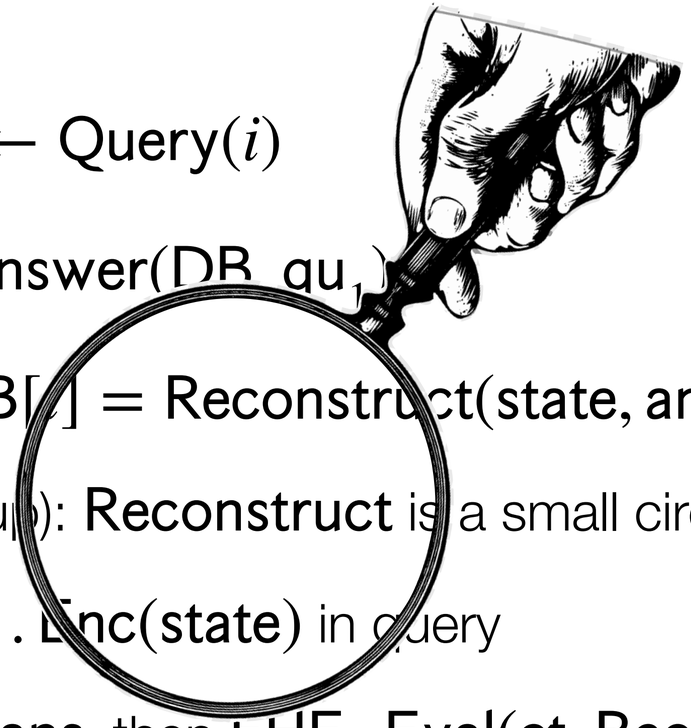
Homomorphic Encryption for 2-Server PIR

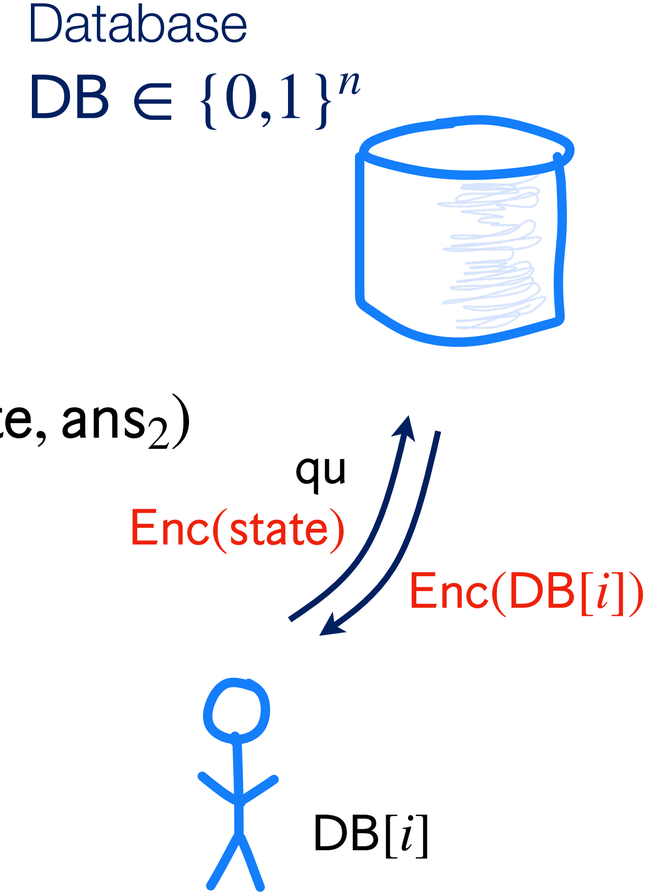
- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$
- Observation (coming up): **Reconstruct** is a small circuit!
  - Include $\mathsf{ct} \leftarrow \mathsf{FHE}.\mathsf{Enc}(\mathsf{state})$ in query
  - Server i computes ans_i then $\mathsf{FHE}.\mathsf{Eval}(\mathsf{ct}, \mathsf{Reconstruct}(\mathsf{ans}_i, \cdot))$
  - Client decrypts!



Homomorphic Encryption for 2-Server PIR

- Three phases:
  - Query: $\mathsf{state}, \mathsf{qu}_1 \leftarrow \mathsf{Query}(i)$
  - Answer: $\mathsf{ans}_1 = \mathsf{Answer}(\mathsf{DB}, \mathsf{qu}_1)$
  - Reconstruction: $\mathsf{DB}[i] = \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_1) + \mathsf{Reconstruct}(\mathsf{state}, \mathsf{ans}_2)$
- Observation (coming up): **Reconstruct** is a small circuit!
  - Include $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{state})$ in query
  - Server i computes ans_i then $\mathsf{FHE.Eval}(\mathsf{ct}, \mathsf{Reconstruct}(\mathsf{ans}_i, \cdot))$
  - Client decrypts!





Our PIR Reconstruction

Cheatsheet

$m \approx \log n$

$D \approx m/2$

Our PIR Reconstruction

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree D

Our PIR Reconstruction

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree D
- Queries: $\mathbf{r}, \mathbf{r} + \mathbf{p}$
  - state $= \mathbf{p}$

Our PIR Reconstruction

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree D
- Queries: $\mathbf{r}, \mathbf{r} + \mathbf{p}$
  - state $= \mathbf{p}$
- Answers: $\left\{ f(\mathbf{x} + \mathbf{e}) : \mathbf{x} \in \{\mathbf{r}, \mathbf{r} + \mathbf{p}\}, \|\mathbf{e}\| \leq D/2 \right\}$

Our PIR Reconstruction

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree D
- Queries: $\mathbf{r}, \mathbf{r}+\mathbf{p}$
  - state $= \mathbf{p}$
- Answers: $\left\{ f(\mathbf{x}+\mathbf{e}) : \mathbf{x} \in \{\mathbf{r}, \mathbf{r}+\mathbf{p}\}, \|\mathbf{e}\| \le D/2 \right\}$
- Reconstruction:

$$\mathsf{DB}_i = f_{\mathsf{DB}}(\mathbf{p}) = \sum_{\substack{\|\mathbf{e}\| \le \lfloor D/2 \rfloor \\ \mathbf{e} \le \mathbf{p}}} \left(\underbrace{f(\mathbf{r}+\mathbf{e})}_{\text{from server 1's answer}} + \underbrace{f(\mathbf{p}+\mathbf{r}+\mathbf{e})}_{\text{from server 2's answer}} \right)$$

Our PIR Reconstruction

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree D
- Queries: $\mathbf{r}, \mathbf{r}+\mathbf{p}$
  - state $= \mathbf{p}$
- Answers: $\left\{f(\mathbf{x}+\mathbf{e}) : \mathbf{x} \in \{\mathbf{r}, \mathbf{r}+\mathbf{p}\}, \|\mathbf{e}\| \le D/2\right\}$
- Reconstruction:

$$\mathsf{DB}_i = f_{\mathsf{DB}}(\mathbf{p}) = \sum_{\substack{\|\mathbf{e}\| \le \lfloor D/2 \rfloor \\ \mathbf{e} \le \mathbf{p}}} \left(\underbrace{f(\mathbf{r}+\mathbf{e})}_{\text{from server 1's answer}} + \underbrace{f(\mathbf{p}+\mathbf{r}+\mathbf{e})}_{\text{from server 2's answer}} \right)$$

- For each $\mathbf{e}$, $\mathbf{1}[\mathbf{e} \le \mathbf{p}] = \mathbf{p}^{\mathbf{e}} = \prod_{i \in [m]} p_i^{e_i}$, which is degree $\le D/2$ in $\mathbf{p}$

Abstract Setup

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- Two polynomials $g_1, g_2 : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree $D/2$ for servers 1 and 2 respectively

$$g_1(\mathbf{x}) = \sum_{\| \mathbf{e} \| \leq \lfloor D/2 \rfloor} \left(\underbrace{f(\mathbf{r}+\mathbf{e})}_{\text{from server 1's answer}} \right) \mathbf{x}^{\mathbf{e}}$$

$$g_2(\mathbf{x}) = \sum_{\| \mathbf{e} \| \leq \lfloor D/2 \rfloor} \left(\underbrace{f(\mathbf{p}+\mathbf{r}+\mathbf{e})}_{\text{from server 2's answer}} \right) \mathbf{x}^{\mathbf{e}}$$

Abstract Setup

Cheatsheet

$m \approx \log n$

$D \approx m/2$

Abstract Setup

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- Two polynomials $g_1, g_2 : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree $D/2$ for servers 1 and 2 respectively
- User's goal: evaluate $g_1(\mathbf{p}) + g_2(\mathbf{p})$ for $\|\mathbf{p}\| = D$, without revealing $\mathbf{p} \in \mathbb{F}_2^m$

Abstract Setup

Cheatsheet

$m \approx \log n$

$D \approx m/2$

- Two polynomials $g_1, g_2 : \mathbb{F}_2^m \to \mathbb{F}_2$ of degree $D/2$ for servers 1 and 2 respectively
- User's goal: evaluate $g_1(\mathbf{p}) + g_2(\mathbf{p})$ for $\|\mathbf{p}\| = D$, without revealing $\mathbf{p} \in \mathbb{F}_2^m$
- All we need to do is get server 1 to help the user evaluate $g_1(\mathbf{p})$ and likewise for server 2

Succinct PIR from FHE

Cheatsheet

$m \approx \log n$

$D \approx m/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

Succinct PIR from FHE

Cheatsheet

$m \approx \log n$

$D \approx m/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Communication: upload $m \cdot \mathsf{poly}(\lambda) = \log n \cdot \mathsf{poly}(\lambda)$, download $\mathsf{poly}(\lambda)$

Succinct PIR from FHE

Cheatsheet

$m \approx \log n$

$D \approx m/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Communication: upload $m \cdot \mathsf{poly}(\lambda) = \log n \cdot \mathsf{poly}(\lambda)$, download $\mathsf{poly}(\lambda)$
- Server computation: same as before + evaluating a degree $D/2$ polynomial in m variables

Succinct PIR from FHE

Cheatsheet

$m \approx \log n$

$D \approx m/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Communication: upload $m \cdot \mathsf{poly}(\lambda) = \log n \cdot \mathsf{poly}(\lambda)$, download $\mathsf{poly}(\lambda)$
- Server computation: same as before + evaluating a degree $D/2$ polynomial in m variables
  - Runtime $\approx \binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{H(1/4)} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$

Succinct PIR from FHE

Cheatsheet

$m \approx \log n$

$D \approx m/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Communication: upload $m \cdot \mathsf{poly}(\lambda) = \log n \cdot \mathsf{poly}(\lambda)$, download $\mathsf{poly}(\lambda)$
- Server computation: same as before + evaluating a degree $D/2$ polynomial in m variables
  - Runtime $\approx \binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{H(1/4)} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$
  - $H(\alpha) \in [0,1]$: binary entropy of $\alpha \in [0,1]$

Succinct PIR from FHE

Cheatsheet

$m \approx \log n$

$D \approx m/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Communication: upload $m \cdot \mathsf{poly}(\lambda) = \log n \cdot \mathsf{poly}(\lambda)$, download $\mathsf{poly}(\lambda)$
- Server computation: same as before + evaluating a degree $D/2$ polynomial in m variables
  - Runtime $\approx \binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{H(1/4)} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$
  - $H(\alpha) \in [0,1]$: binary entropy of $\alpha \in [0,1]$

Theorem: with compact **fully homomorphic encryption***, we get 2-server PIR with server storage $n^{1+o(1)}$, time per query $O(n^{0.82})$ and communication $O(\log n)$.

What About Linear Homomorphism?

What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$

What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE}.\mathsf{Enc}(\mathbf{p}^{\otimes D/2})$ in the query

What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE}.\mathsf{Enc}(\mathbf{p}^{\otimes D/2})$ in the query
  - Communication cost: $\binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$, same as before 😭

What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE.Enc}(\mathbf{p}^{\otimes D/2})$ in the query
  - Communication cost: $\binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$, same as before 😭
  - Very imbalanced: uploading $n^{0.82}$ ciphertexts and downloading just one → can rebalance!

What About Linear Homomorphism?

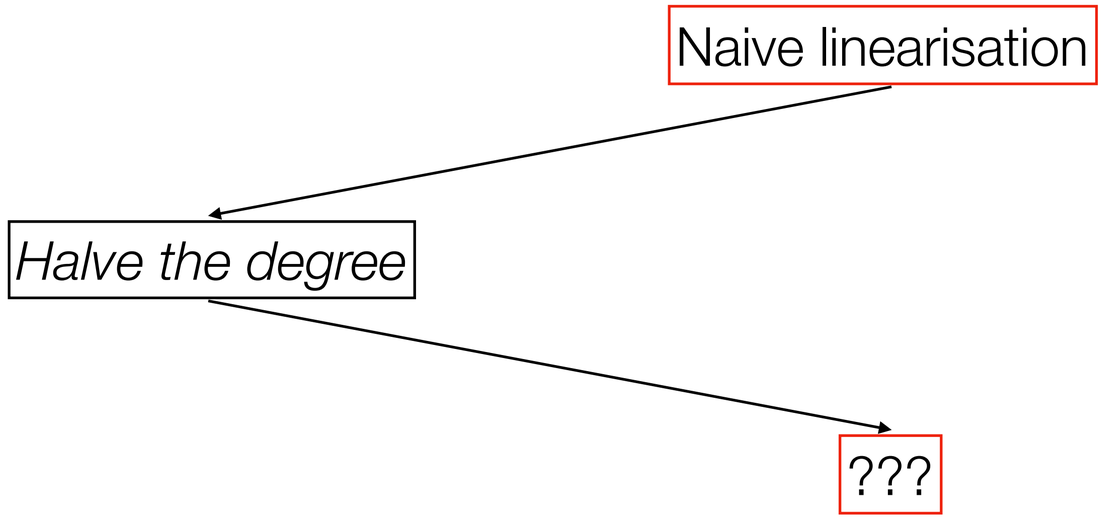
- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE.Enc}(\mathbf{p}^{\otimes D/2})$ in the query
  - Communication cost: $\binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$, same as before 😭
  - Very imbalanced: uploading $n^{0.82}$ ciphertexts and downloading just one → can rebalance!

Naive linearisation

???

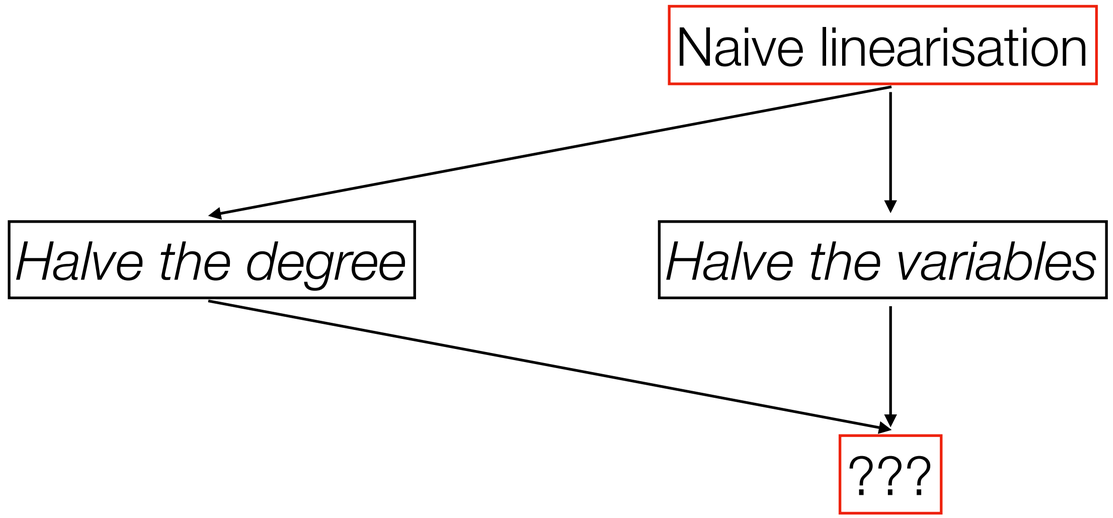
What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE.Enc}(\mathbf{p}^{\otimes D/2})$ in the query
  - Communication cost: $\binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$, same as before 😭
  - Very imbalanced: uploading $n^{0.82}$ ciphertexts and downloading just one → can rebalance!



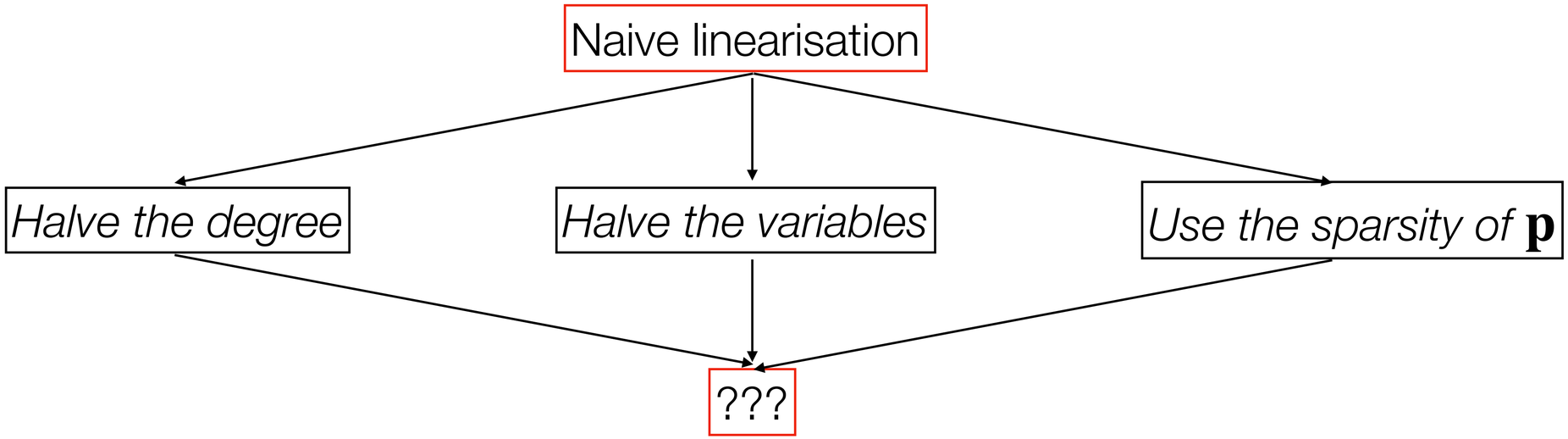
What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE.Enc}(\mathbf{p}^{\otimes D/2})$ in the query
  - Communication cost: $\binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$, same as before 😭
  - Very imbalanced: uploading $n^{0.82}$ ciphertexts and downloading just one → can rebalance!



What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $\|\mathbf{p}\| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE.Enc}(\mathbf{p}^{\otimes D/2})$ in the query
  - Communication cost: $\binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$, same as before 😭
  - Very imbalanced: uploading $n^{0.82}$ ciphertexts and downloading just one → can rebalance!



What About Linear Homomorphism?

- Goal: compute $g(\mathbf{p})$ for g of degree $D/2$ and $||\mathbf{p}|| = D$, without revealing $\mathbf{p}$
- Naive attempt: linearise the computation by including $\mathsf{LHE.Enc}(\mathbf{p}^{\otimes D/2})$ in the query
- Communication cost: $\binom{m}{D/2} \cdot \mathsf{poly}(\lambda) \approx n^{0.82} \cdot \mathsf{poly}(\lambda)$, same as before 😭
- Verifier: uploading $n^{0.82}$ ciphertexts and downloading just one? 🤔 can rebalance!

Natural target: reduce communication from $n^{0.82}$ to $\sqrt{n^{0.82}} = n^{0.41}$

Naive linearisation

Halve the degree

Halve the variables

Use the sparsity of $\mathbf{p}$

???

LHE → Computing Deg 2 Polynomials

LHE → Computing Deg 2 Polynomials

- **Claim** [KO97]: assume LHE. Then if the user has $\mathbf{x}, \mathbf{y} \in \mathbb{F}^{\ell}$ and the server has $\mathbf{A} \in \mathbb{F}^{\ell \times \ell}$, the user can learn $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{y}$ without revealing $\mathbf{x}, \mathbf{y}$, with $\ell \cdot \mathsf{poly}(\lambda)$ communication.

LHE → Computing Deg 2 Polynomials

- **Claim** [KO97]: assume LHE. Then if the user has $\mathbf{x}, \mathbf{y} \in \mathbb{F}^{\ell}$ and the server has $\mathbf{A} \in \mathbb{F}^{\ell \times \ell}$, the user can learn $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{y}$ without revealing $\mathbf{x}, \mathbf{y}$, with $\ell \cdot \mathsf{poly}(\lambda)$ communication.
- **Proof:**
  - User sends $\mathsf{LHE}.\mathsf{Enc}(\mathbf{y})$

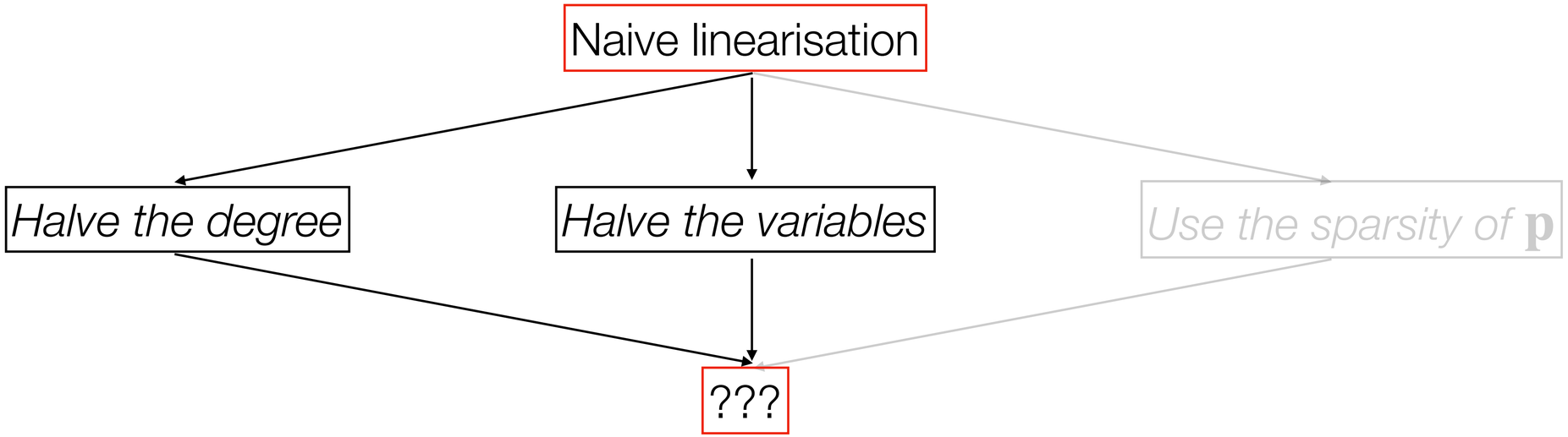
LHE → Computing Deg 2 Polynomials

- **Claim** [KO97]: assume LHE. Then if the user has $\mathbf{x}, \mathbf{y} \in \mathbb{F}^{\ell}$ and the server has $\mathbf{A} \in \mathbb{F}^{\ell \times \ell}$, the user can learn $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{y}$ without revealing $\mathbf{x}, \mathbf{y}$, with $\ell \cdot \mathsf{poly}(\lambda)$ communication.
- **Proof:**
  - User sends $\mathsf{LHE.Enc}(\mathbf{y})$
  - Server replies with $\mathsf{LHE.Enc}(\mathbf{A}\mathbf{y})$

LHE → Computing Deg 2 Polynomials

- **Claim** [KO97]: assume LHE. Then if the user has $\mathbf{x}, \mathbf{y} \in \mathbb{F}^{\ell}$ and the server has $\mathbf{A} \in \mathbb{F}^{\ell \times \ell}$, the user can learn $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{y}$ without revealing $\mathbf{x}, \mathbf{y}$, with $\ell \cdot \mathsf{poly}(\lambda)$ communication.
- **Proof:**
  - User sends $\mathsf{LHE.Enc}(\mathbf{y})$
  - Server replies with $\mathsf{LHE.Enc}(\mathbf{A}\mathbf{y})$
  - User decrypts and locally takes the inner product with $\mathbf{x}$

Rebalancing



Halving the Degree/Variables

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

Halving the Degree/Variables

- Want to write $g(\mathbf{p})$ as a degree 2 polynomial in as few variables as possible

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

Halving the Degree/Variables

- Want to write $g(\mathbf{p})$ as a degree 2 polynomial in as few variables as possible
- Idea 1: use $\mathbf{p}^{\otimes D/4}$

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

Halving the Degree/Variables

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Want to write $g(\mathbf{p})$ as a degree 2 polynomial in as few variables as possible
- Idea 1: use $\mathbf{p}^{\otimes D/4}$
  - Number of variables (and communication): $\binom{m}{D/4} \approx n^{H(1/8)} \approx n^{0.54}$

Halving the Degree/Variables

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Want to write $g(\mathbf{p})$ as a degree 2 polynomial in as few variables as possible
- Idea 1: use $\mathbf{p}^{\otimes D/4}$
  - Number of variables (and communication): $\binom{m}{D/4} \approx n^{H(1/8)} \approx n^{0.54}$
- Idea 2: use $\mathbf{p}_{1:m/2}^{\otimes D/2}$ and $\mathbf{p}_{m/2+1:m}^{\otimes D/2}$

Halving the Degree/Variables

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Want to write $g(\mathbf{p})$ as a degree 2 polynomial in as few variables as possible
- Idea 1: use $\mathbf{p}^{\otimes D/4}$
  - Number of variables (and communication): $\binom{m}{D/4} \approx n^{H(1/8)} \approx n^{0.54}$
- Idea 2: use $\mathbf{p}_{1:m/2}^{\otimes D/2}$ and $\mathbf{p}_{m/2+1:m}^{\otimes D/2}$
  - Number of variables: $\binom{m/2}{D/2} \approx 2^{mH(1/2)/2} \approx n^{0.5}$

Halving the Degree/Variables

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

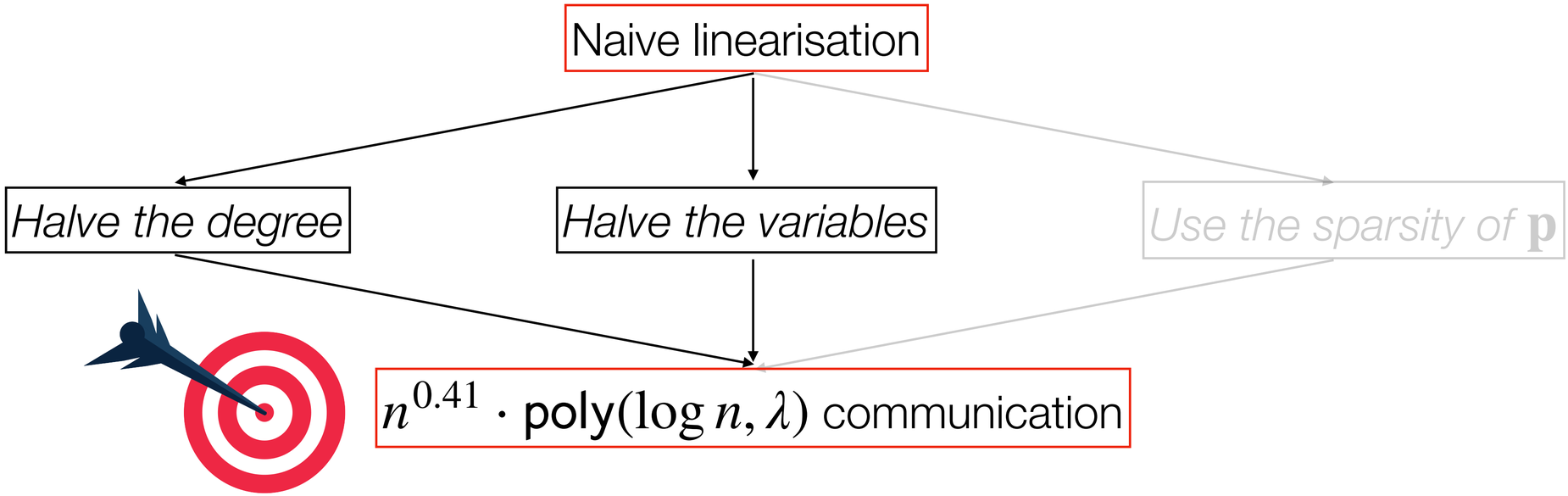
$D \approx m/2$

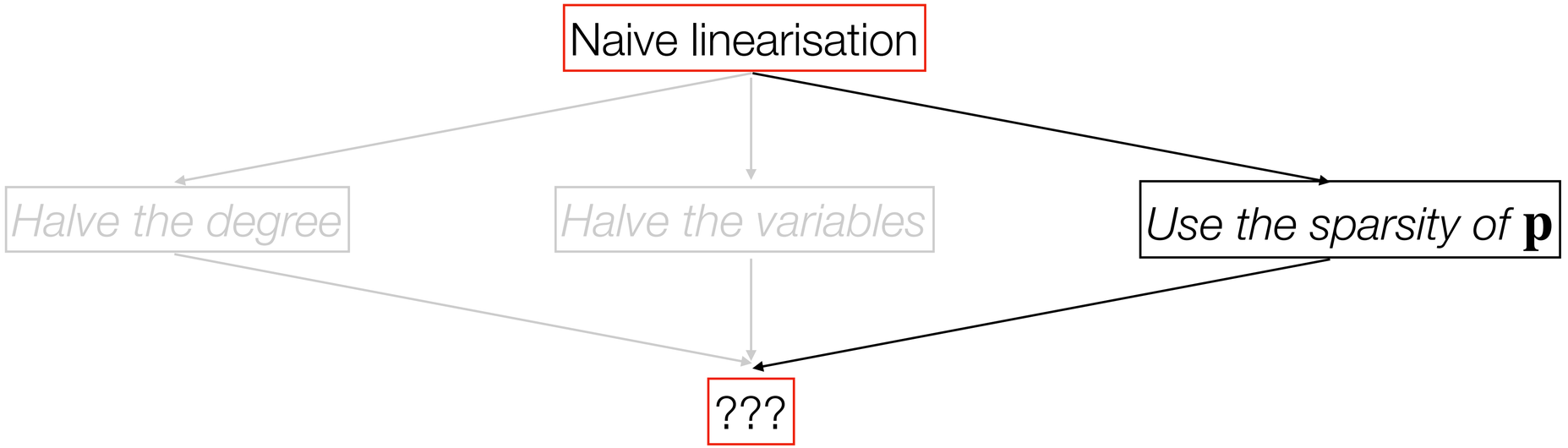
Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\binom{m}{\alpha m} \approx 2^{mH(\alpha)} \approx n^{H(\alpha)}$

- Want to write $g(\mathbf{p})$ as a degree 2 polynomial in as few variables as possible
- Idea 1: use $\mathbf{p}^{\otimes D/4}$
  - Number of variables (and communication): $\binom{m}{D/4} \approx n^{H(1/8)} \approx n^{0.54}$
- Idea 2: use $\mathbf{p}_{1:m/2}^{\otimes D/2}$ and $\mathbf{p}_{m/2+1:m}^{\otimes D/2}$
  - Number of variables: $\binom{m/2}{D/2} \approx 2^{mH(1/2)/2} \approx n^{0.5}$
- Careful combination of these ideas: $\approx \binom{m/2}{D/4} \approx 2^{mH(1/4)/2} \approx n^{0.41}$


Naive linearisation
Halve the degree
Halve the variables
Use the sparsity of $\mathbf{p}$
$n^{0.41} \cdot \mathsf{poly}(\log n, \lambda)$ communication


Naive linearisation
Halve the degree
Halve the variables
Use the sparsity of p
???

Leveraging the Sparsity of $\mathbf{p}$ [BIM04]

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\|\mathbf{p}\| = D$

Leveraging the Sparsity of $\mathbf{p}$ [BIM04]

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\|\mathbf{p}\| = D$

- Original naive idea: compute $\langle \mathbf{p}^{\otimes D/2}, \mathsf{coefs}(g) \rangle$ under the hood

Leveraging the Sparsity of $\mathbf{p}$ [BIM04]

Cheatsheet

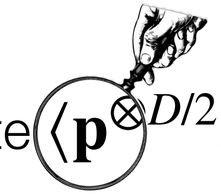
$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\|\mathbf{p}\| = D$

- Original naive idea: compute  under the hood

Leveraging the Sparsity of $\mathbf{p}$ [BIM04]

Cheatsheet

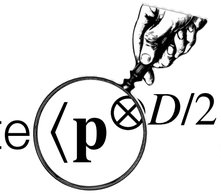
$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\|\mathbf{p}\| = D$

- Original naive idea: compute  $\langle \mathbf{p}^{\otimes D/2}, \mathsf{coefs}(g) \rangle$ under the hood
- Observation: $\mathbf{p}$ only has D nonzero entries → $\mathbf{p}^{\otimes D/2}$ has $\leq 2^D \ll \binom{m}{D/2}$ nonzero entries!

Leveraging the Sparsity of $\mathbf{p}$ [BIM04]

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\|\mathbf{p}\| = D$

- Original naive idea: compute $\langle \mathbf{p}^{\otimes D/2}, \mathbf{coefs}(g) \rangle$ under the hood

- Observation: $\mathbf{p}$ only has D nonzero entries → $\mathbf{p}^{\otimes D/2}$ has $\leq 2^D \ll \binom{m}{D/2}$ nonzero entries!

- Idea: view $\mathbf{coefs}(g)$ as a “mini-database” and run a single-server “mini-PIR” protocol (KO97, IKOS04) to retrieve $\mathbf{coefs}(g)$ in just these 2^D positions

Leveraging the Sparsity of $\mathbf{p}$ [BIM04]

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

$D \approx m/2$

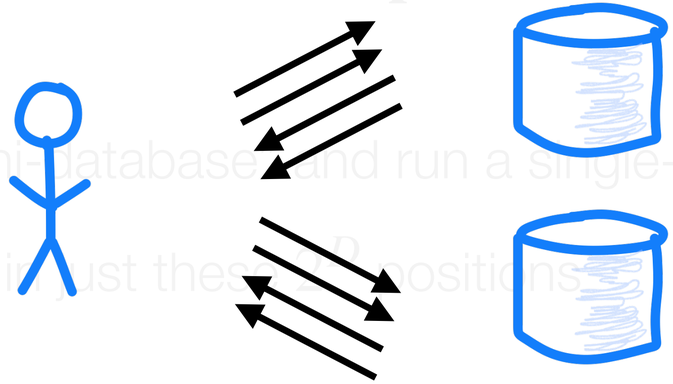
Evaluating $g(\mathbf{p})$

$\deg g \leq D/2$

$\|\mathbf{p}\| = D$

- Original naive idea: compute $\langle \mathbf{p}^{\otimes D/2}, \mathbf{coefs}(g) \rangle$ under the hood
- Observation: $\mathbf{p}$ only has D nonzero entries $\Rightarrow \mathbf{p}^{\otimes D/2}$ has $\leq 2^{D/2} \ll \binom{m}{D/2}$ nonzero entries!
- Idea: view $\mathbf{coefs}(g)$ as a "mini-database" and run a single-server "mini-PIR" protocol (KO97, IKOS04) to retrieve $\mathbf{coefs}(g)$ in just these D positions

Batch PIR with many, non-adaptive queries



[IKOS'04,HHG'13,GKL'10,AS'16,H'16,ACLS'18,CHLR'18]

Leveraging the Sparsity of $\mathbf{p}$ [BIM04]

Cheatsheet

$\mathbf{p} \in \mathbb{F}_2^m$

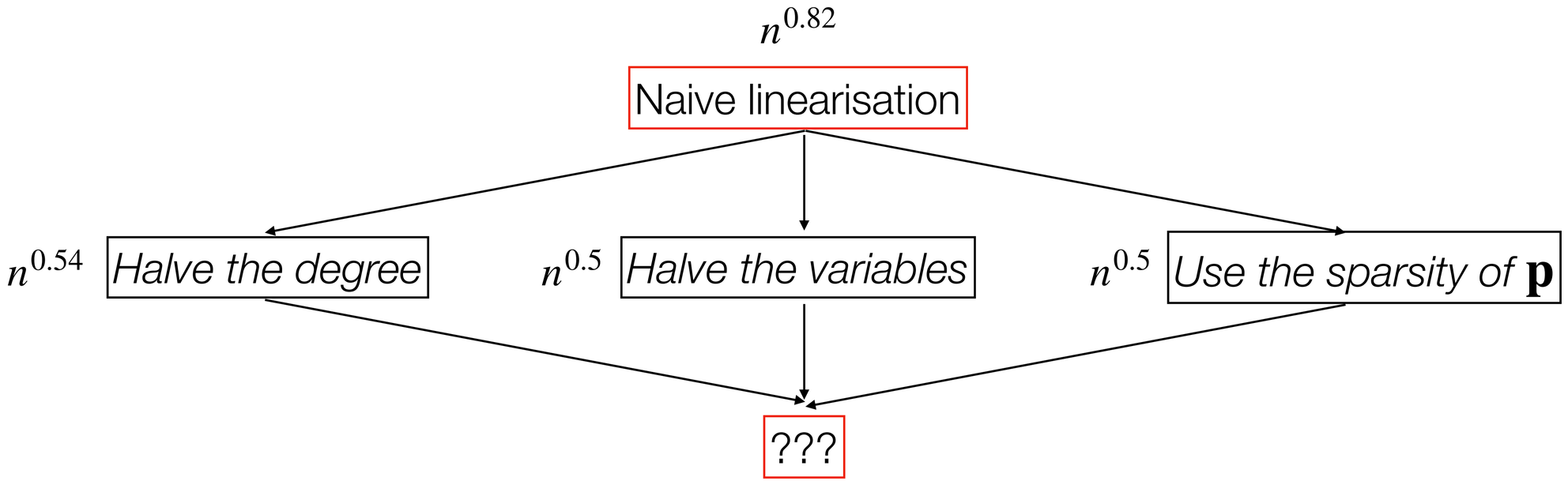
$D \approx m/2$

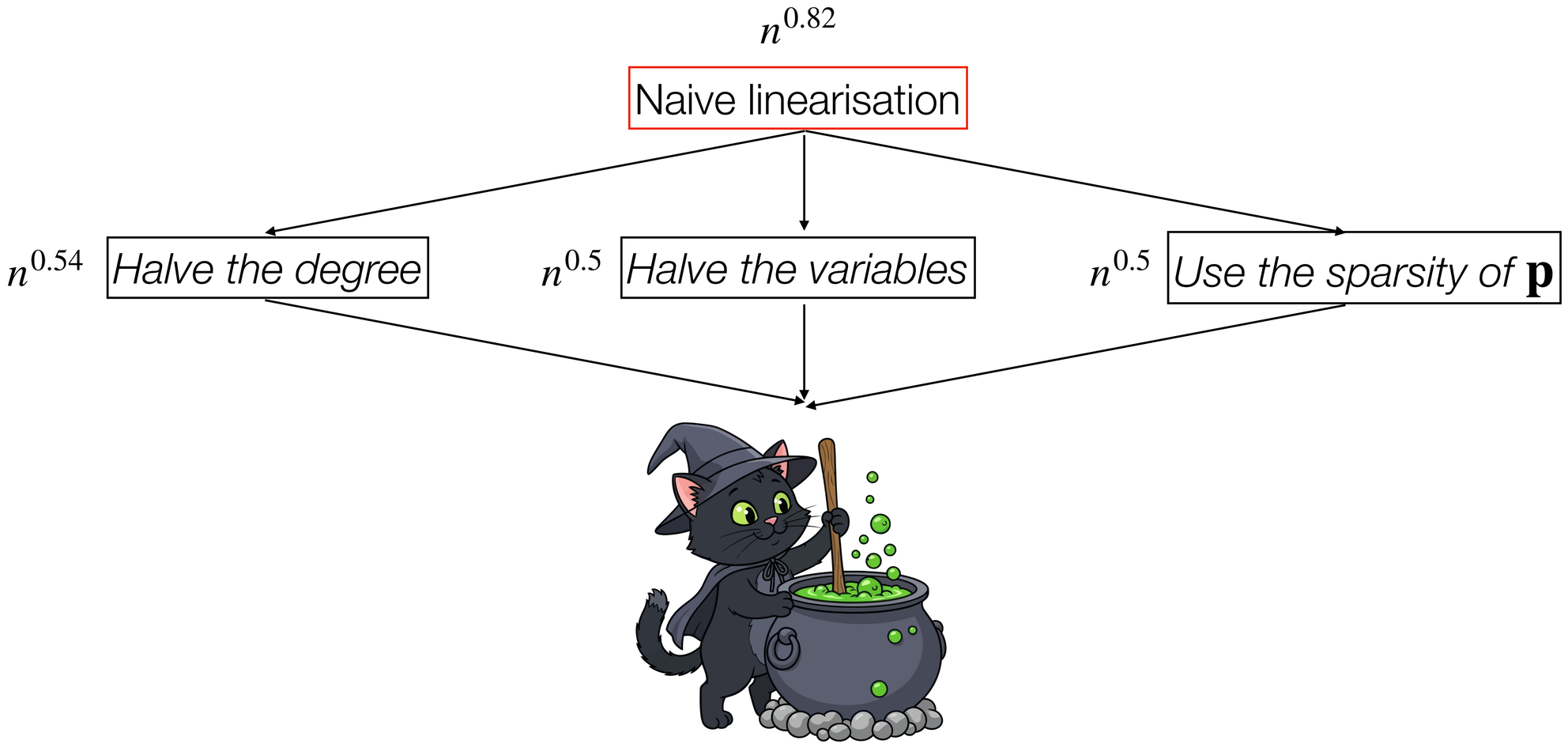
Evaluating $g(\mathbf{p})$

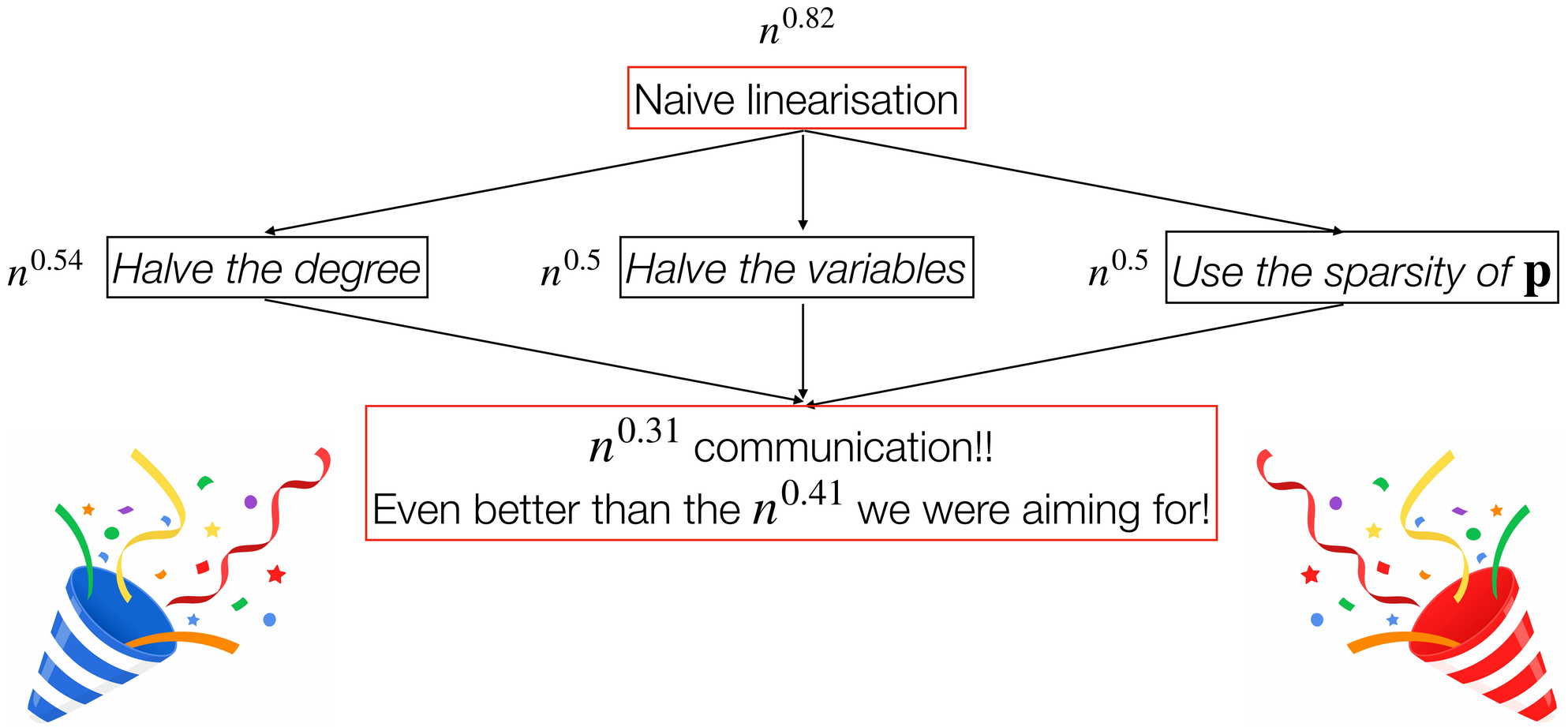
$\deg g \leq D/2$

$\|\mathbf{p}\| = D$

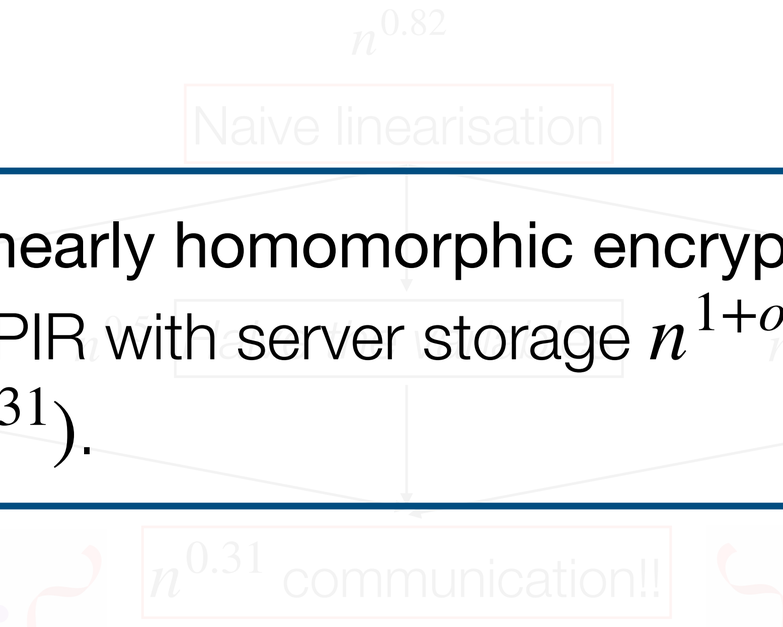
- Original naive idea: compute  $\langle \mathbf{p}^{\otimes D/2}, \mathbf{coefs}(g) \rangle$ under the hood
- Observation: $\mathbf{p}$ only has D nonzero entries → $\mathbf{p}^{\otimes D/2}$ has $\leq 2^D \ll \binom{m}{D/2}$ nonzero entries!
- Idea: view $\mathbf{coefs}(g)$ as a “mini-database” and run a single-server “mini-PIR” protocol (KO97, IKOS04) to retrieve $\mathbf{coefs}(g)$ in just these 2^D positions
- Communication: $\approx 2^D \approx 2^{m/2} \approx n^{0.5}$


$n^{0.82}$
Naive linearisation
$n^{0.54}$
Halve the degree
$n^{0.5}$
Halve the variables
$n^{0.5}$
Use the sparsity of $\mathbf{p}$
???


$n^{0.82}$
Naive linearisation
$n^{0.54}$
Halve the degree
$n^{0.5}$
Halve the variables
$n^{0.5}$
Use the sparsity of $\mathbf{p}$


$n^{0.82}$
Naive linearisation
$n^{0.54}$
Halve the degree
$n^{0.5}$
Halve the variables
$n^{0.5}$
Use the sparsity of $\mathbf{p}$
$n^{0.31}$ communication!!
Even better than the $n^{0.41}$ we were aiming for!

Theorem: with compact **linearly homomorphic encryption** [known from DDH, DCR, QR, LWE], we get 2-server PIR with server storage $n^{1+o(1)}$, time per query $O(n^{0.82})$ and communication $O(n^{0.31})$.



Bonus Slides on ≥ 3 Servers

$s = 2$ Servers: A Refresher

Query: $\mathbf{L}(0) = \mathbf{r}$

Query: $\mathbf{L}(1) = \mathbf{p} + \mathbf{r}$

Cheatsheet

Field: $\mathbb{F}_2$

f_{DB} multilinear

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

$\binom{m}{D} \geq n$

$s = 2$ Servers: A Refresher

Query: $\mathbf{L}(0) = \mathbf{r}$

Ans: $\nabla^{\leq D/2} f_{\mathsf{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(1) = \mathbf{p} + \mathbf{r}$

Ans: $\nabla^{\leq D/2} f_{\mathsf{DB}}(\mathbf{L}(1))$

Cheatsheet

Field: $\mathbb{F}_2$

f_{DB} multilinear

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

$\binom{m}{D} \geq n$

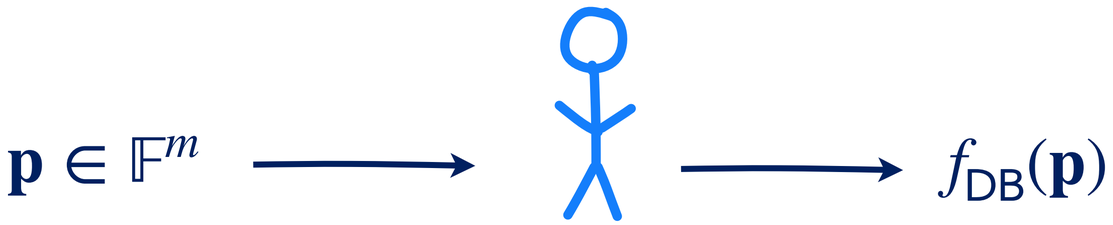
$s = 2$ Servers: A Refresher

Query: $\mathbf{L}(0) = \mathbf{r}$

Ans: $\nabla^{\leq D/2} f_{\mathsf{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(1) = \mathbf{p} + \mathbf{r}$

Ans: $\nabla^{\leq D/2} f_{\mathsf{DB}}(\mathbf{L}(1))$



Cheatsheet

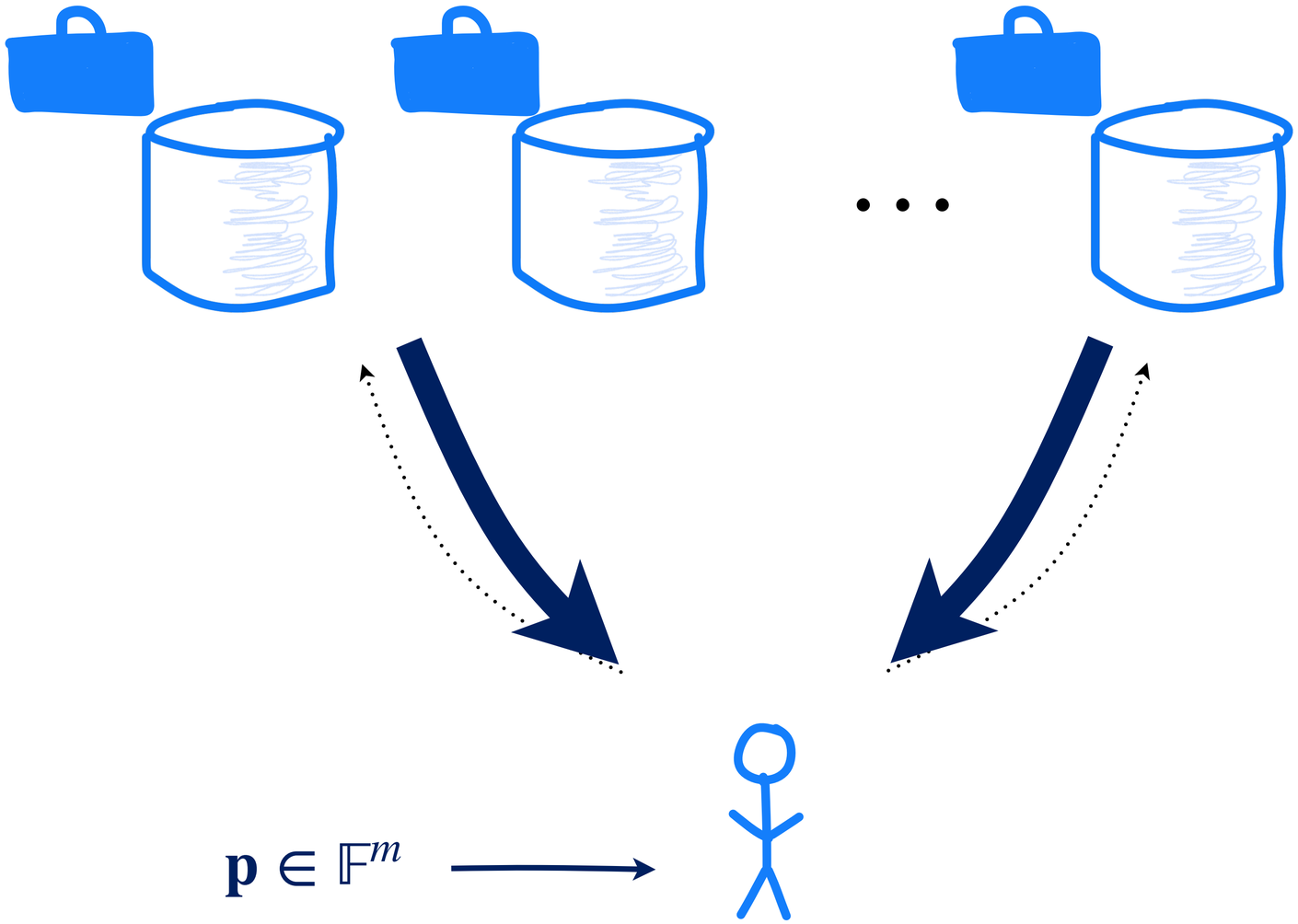
Field: $\mathbb{F}_2$

f_{DB} multilinear

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

$\binom{m}{D} \geq n$

$s > 2$ Servers: What Changes? [GLMDS25]



Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

$\binom{m}{D} \geq n$

$s > 2$ Servers: What Changes? [GLMDS25]

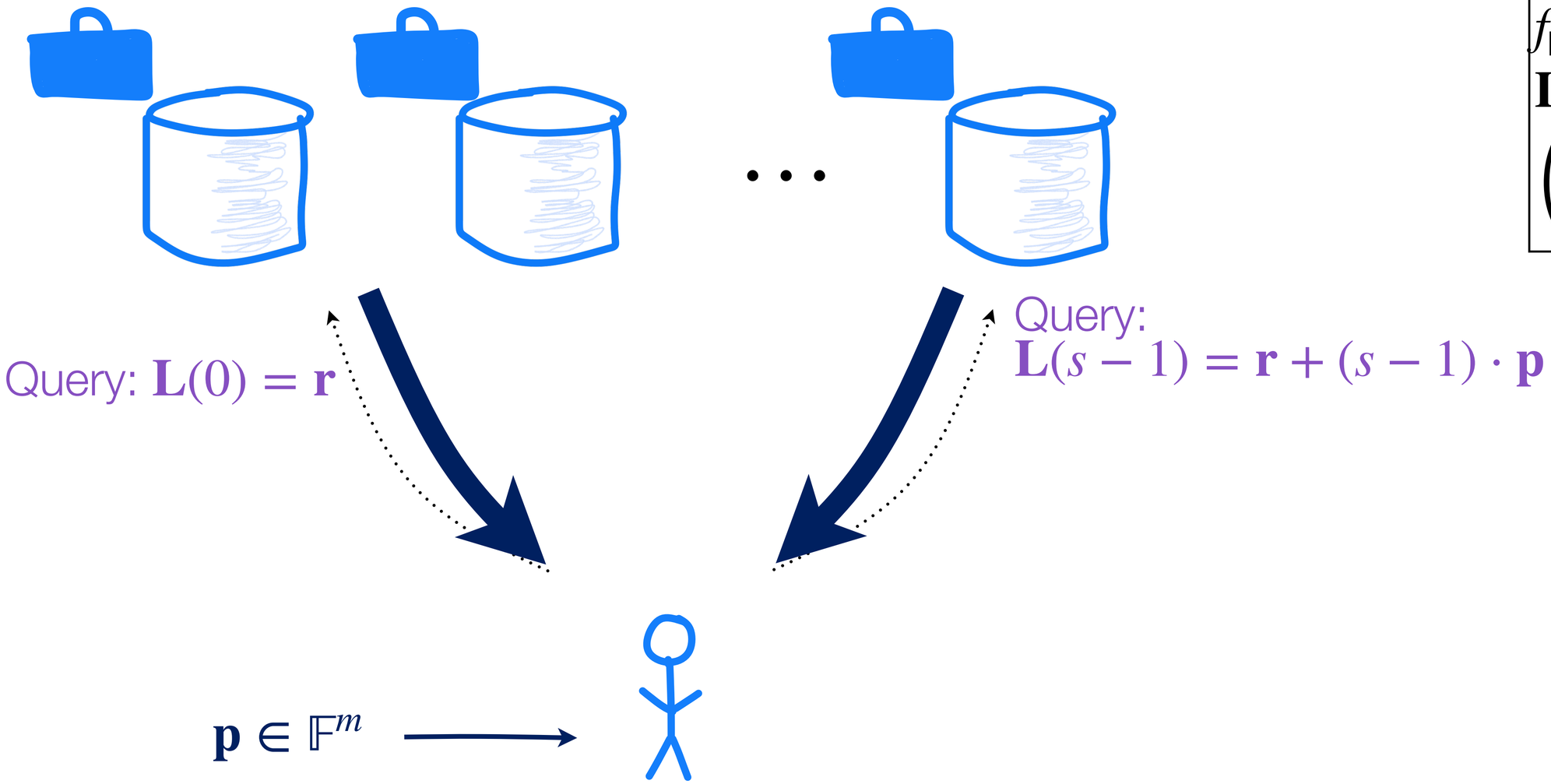
Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

$\binom{m}{D} \geq n$



$s > 2$ Servers: What Changes? [GLMDS25]

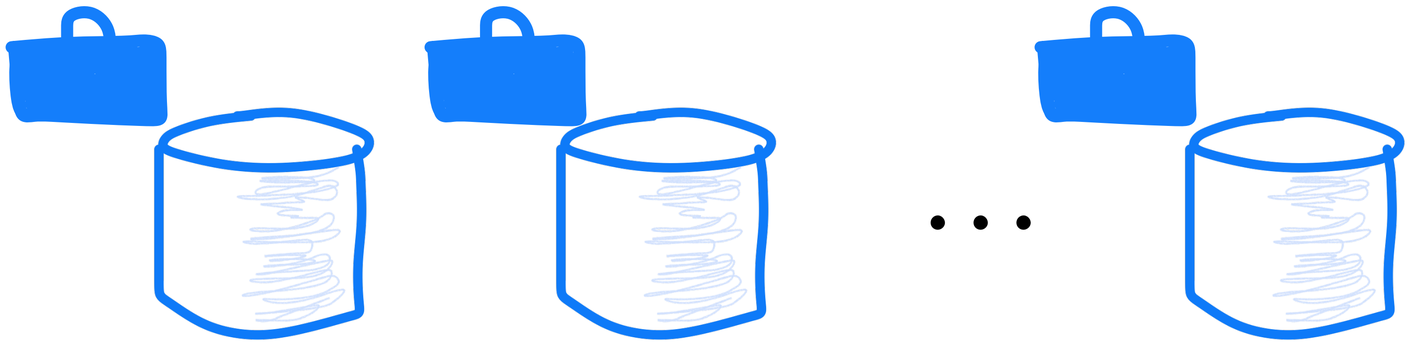
Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

$\binom{m}{D} \geq n$



Query: $\mathbf{L}(0) = \mathbf{r}$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(s-1) = \mathbf{r} + (s-1) \cdot \mathbf{p}$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(s-1))$

$\mathbf{p} \in \mathbb{F}^m$

$s > 2$ Servers: What Changes? [GLMDS25]

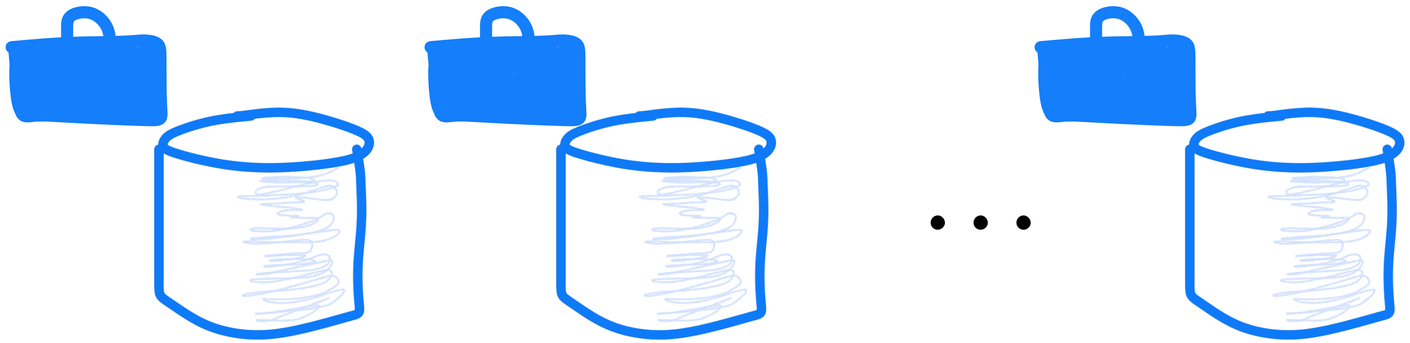
Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\mathbf{L}(t) = \mathbf{r} + t \cdot \mathbf{p}$

$\binom{m}{D} \geq n$

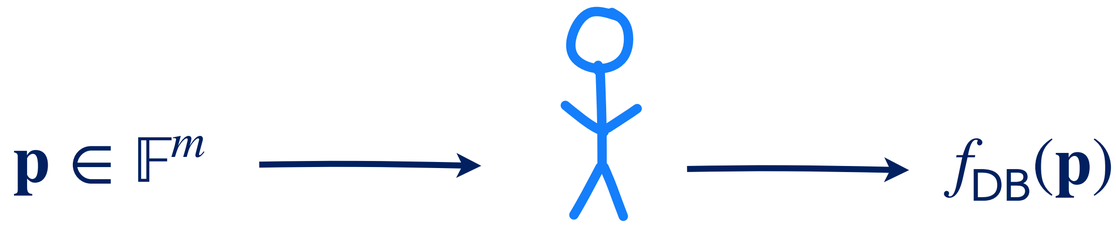


Query: $\mathbf{L}(0) = \mathbf{r}$

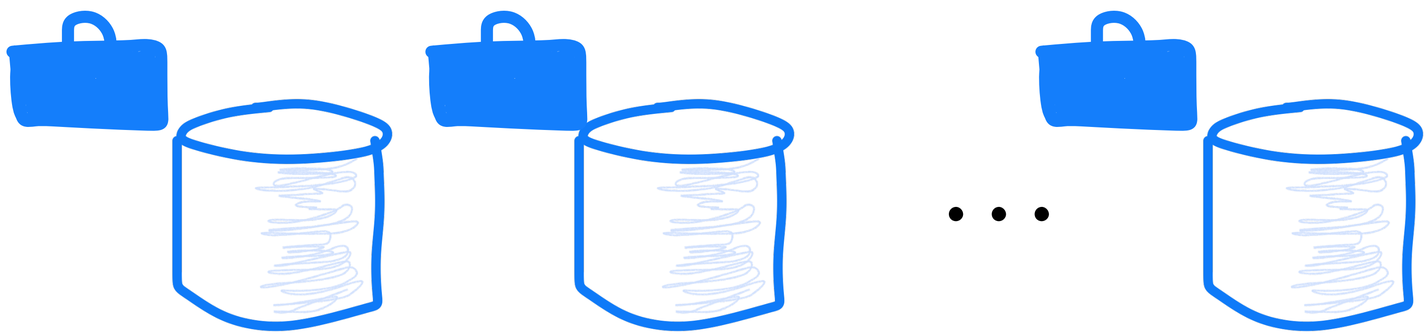
Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(s-1) = \mathbf{r} + (s-1) \cdot \mathbf{p}$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(s-1))$



Collusion Resistance via Shamir Secret Sharing [BIK05]



Query: $\mathbf{L}(0)$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(s-1)$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(s-1))$

$\mathbf{p} \in \mathbb{F}^m$ → → $f_{\mathsf{DB}}(\mathbf{p})$

Cheatsheet

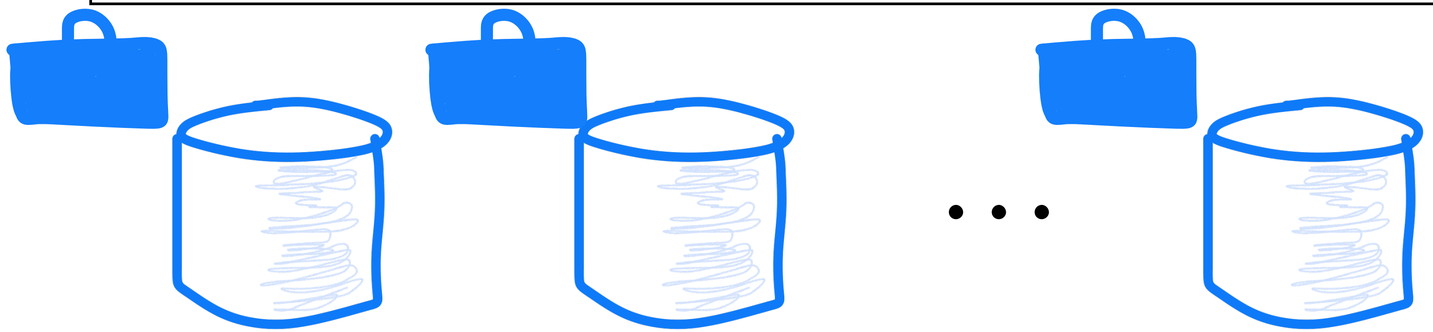
Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\binom{m}{D} \geq n$

Collusion Resistance via Shamir Secret Sharing [BIK05]

Security against 1 server: use a line of slope $\mathbf{p}$

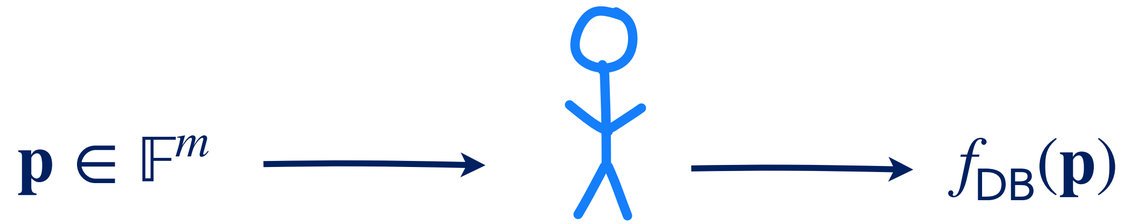


Query: $\mathbf{L}(0)$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(s-1)$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(s-1))$



Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\binom{m}{D} \geq n$

Collusion Resistance via Shamir Secret Sharing [BIK05]

Security against 1 server: use a line of slope $\mathbf{p}$

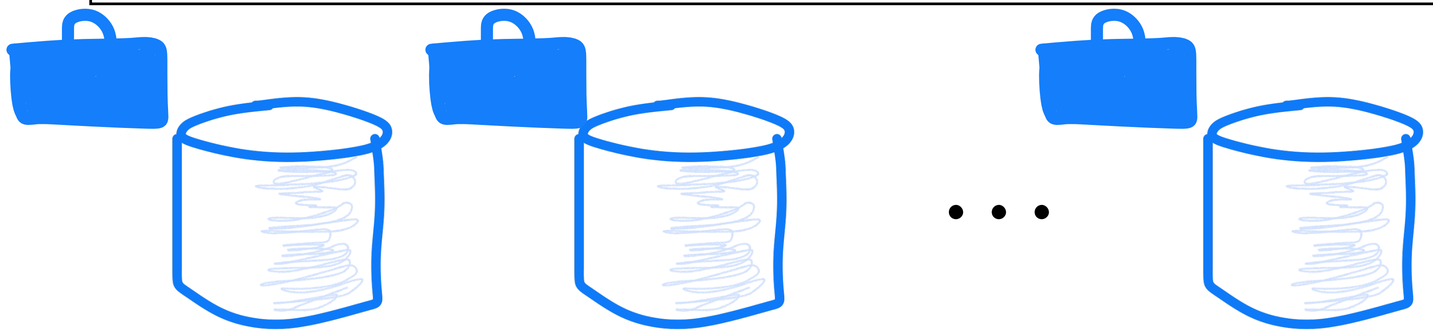
Security against c colluding servers: use a degree $\mathbf{c}$ curve of "slope" $\mathbf{p}$

Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\binom{m}{D} \geq n$

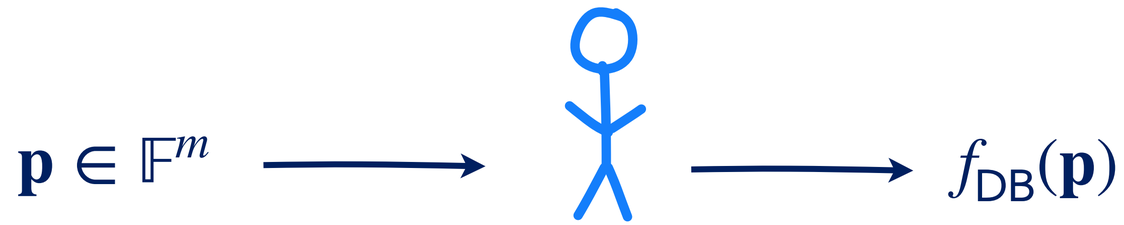


Query: $\mathbf{L}(0)$

Query: $\mathbf{L}(s-1)$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(0))$

Ans: $\nabla^{\leq D/s} f_{\mathsf{DB}}(\mathbf{L}(s-1))$



Collusion Resistance via Shamir Secret Sharing [BIK05]

Security against 1 server: use a line of slope $\mathbf{p}$

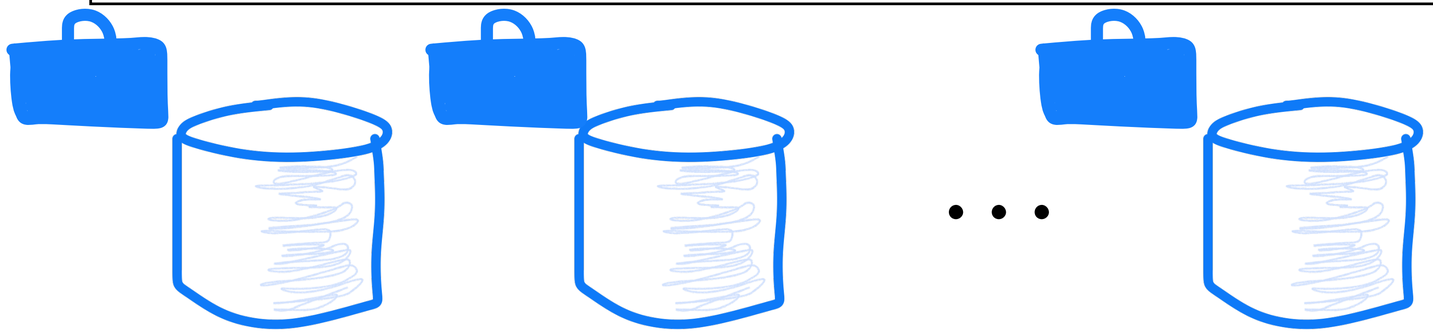
Security against c colluding servers: use a degree $\mathbf{c}$ curve of "slope" $\mathbf{p}$

Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\binom{m}{D} \geq n$



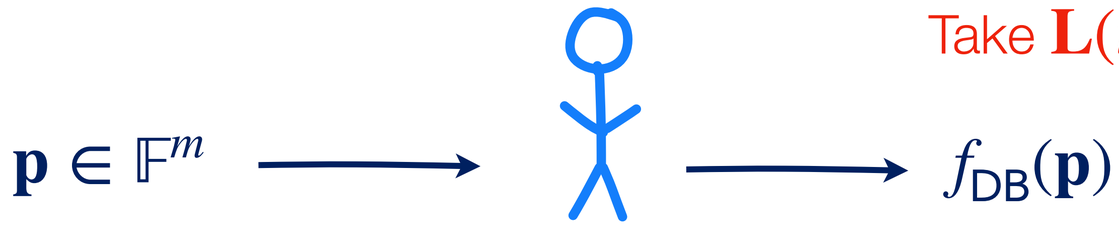
Query: $\mathbf{L}(0)$

Ans: $\nabla^{\leq D/s} f_{\text{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(s-1)$

Ans: $\nabla^{\leq D/s} f_{\text{DB}}(\mathbf{L}(s-1))$

Take $\mathbf{L}(t) = \mathbf{r}_0 + \mathbf{r}_1 \cdot t + \ldots + \mathbf{r}_{c-1} \cdot t^{c-1} + \mathbf{p} \cdot t^c$



$\mathbf{p} \in \mathbb{F}^m$

$f_{\text{DB}}(\mathbf{p})$

Collusion Resistance via Shamir Secret Sharing [BIK05]

Security against 1 server: use a line of slope $\mathbf{p}$

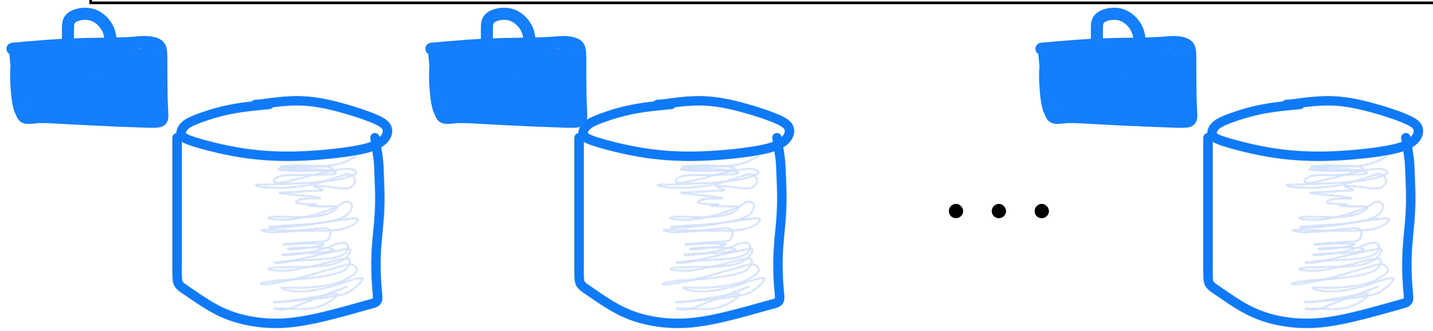
Security against c colluding servers: use a degree $\mathbf{c}$ curve of "slope" $\mathbf{p}$

Cheatsheet

Field: $\mathbb{F}_q$ (for $q \geq s$)

f_{DB} multilinear

$\binom{m}{D} \geq n$



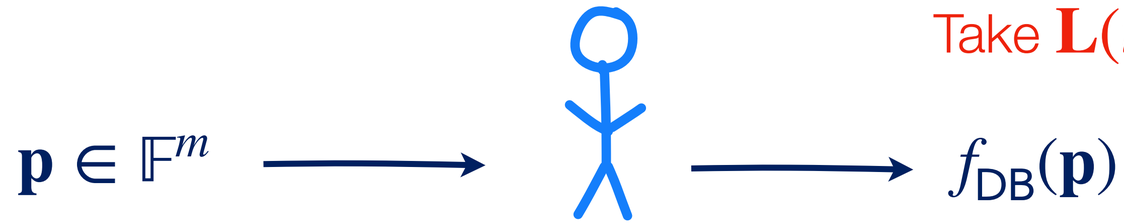
Query: $\mathbf{L}(0)$

Ans: $\nabla^{\leq cD/s} f_{\text{DB}}(\mathbf{L}(0))$

Query: $\mathbf{L}(s-1)$

Ans: $\nabla^{\leq cD/s} f_{\text{DB}}(\mathbf{L}(s-1))$

Take $\mathbf{L}(t) = \mathbf{r}_0 + \mathbf{r}_1 \cdot t + \ldots + \mathbf{r}_{c-1} \cdot t^{c-1} + \mathbf{p} \cdot t^c$



$\mathbf{p} \in \mathbb{F}^m$ → $f_{\text{DB}}(\mathbf{p})$

$s > 2$ Servers: Our Improvements

$s > 2$ Servers: Our Improvements

- New idea 1: the same finite differences technique as in the 2-server case!

$s > 2$ Servers: Our Improvements

- New idea 1: the same finite differences technique as in the 2-server case!
- New idea 2: vary the **individual** degree of f_{DB}

Role of Individual Degree

Role of Individual Degree

- Up to now: $f_{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ multilinear with total degree D

Role of Individual Degree

- Up to now: $f_{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ multilinear with total degree D
  - $\binom{m}{D}$ degrees of freedom → $\binom{m}{D} \geq n$

Role of Individual Degree

- Up to now: $f_{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ multilinear with total degree D
  - $\binom{m}{D}$ degrees of freedom → $\binom{m}{D} \geq n$
  - Number of derivatives: $\binom{m}{D/s}$

Role of Individual Degree

- Up to now: $f_{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ multilinear with total degree D
  - $\binom{m}{D}$ degrees of freedom → $\binom{m}{D} \geq n$
  - Number of derivatives: $\binom{m}{D/s}$
- Generalisation: what if we let f_{DB} have higher individual degree $d > 1$?

Role of Individual Degree

- Up to now: $f_{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ multilinear with total degree D
  - $\binom{m}{D}$ degrees of freedom → $\binom{m}{D} \geq n$
  - Number of derivatives: $\binom{m}{D/s}$
- Generalisation: what if we let f_{DB} have higher individual degree $d > 1$?
  - 😁: more degrees of freedom → larger database!

Role of Individual Degree

- Up to now: $f_{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ multilinear with total degree D
  - $\binom{m}{D}$ degrees of freedom → $\binom{m}{D} \geq n$
  - Number of derivatives: $\binom{m}{D/s}$
- Generalisation: what if we let f_{DB} have higher individual degree $d > 1$?
  - 😁: more degrees of freedom → larger database!
  - 😭: also more derivatives to send → more time and communication per query

Role of Individual Degree

- Up to now: $f_{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ multilinear with total degree D
  - $\binom{m}{D}$ degrees of freedom → $\binom{m}{D} \geq n$
  - Number of derivatives: $\binom{m}{D/s}$
- Generalisation: what if we let f_{DB} have higher individual degree $d > 1$?
  - 😁: more degrees of freedom → larger database!
  - 😭: also more derivatives to send → more time and communication per query
- **TLDR: the sweet spot for d increases as the number of servers increases**

Special Case: Storage $n^{1+o(1)}$, Individual Degree $s - 1$

Special Case: Storage $n^{1+o(1)}$, Individual Degree $s-1$

Theorem: for constant prime s, we get information-theoretic s-server PIR with:

Special Case: Storage $n^{1+o(1)}$, Individual Degree $s-1$

Theorem: for constant prime s, we get information-theoretic s-server PIR with:

- server storage $n^{1+o(1)}$ and

Special Case: Storage $n^{1+o(1)}$, Individual Degree $s-1$

Theorem: for constant prime s, we get information-theoretic s-server PIR with:

- server storage $n^{1+o(1)}$ and
- communication and time per query $n^{\alpha+o(1)}$, where

Special Case: Storage $n^{1+o(1)}$, Individual Degree $s-1$

Theorem: for constant prime s, we get information-theoretic s-server PIR with:

- server storage $n^{1+o(1)}$ and
- communication and time per query $n^{\alpha+o(1)}$, where

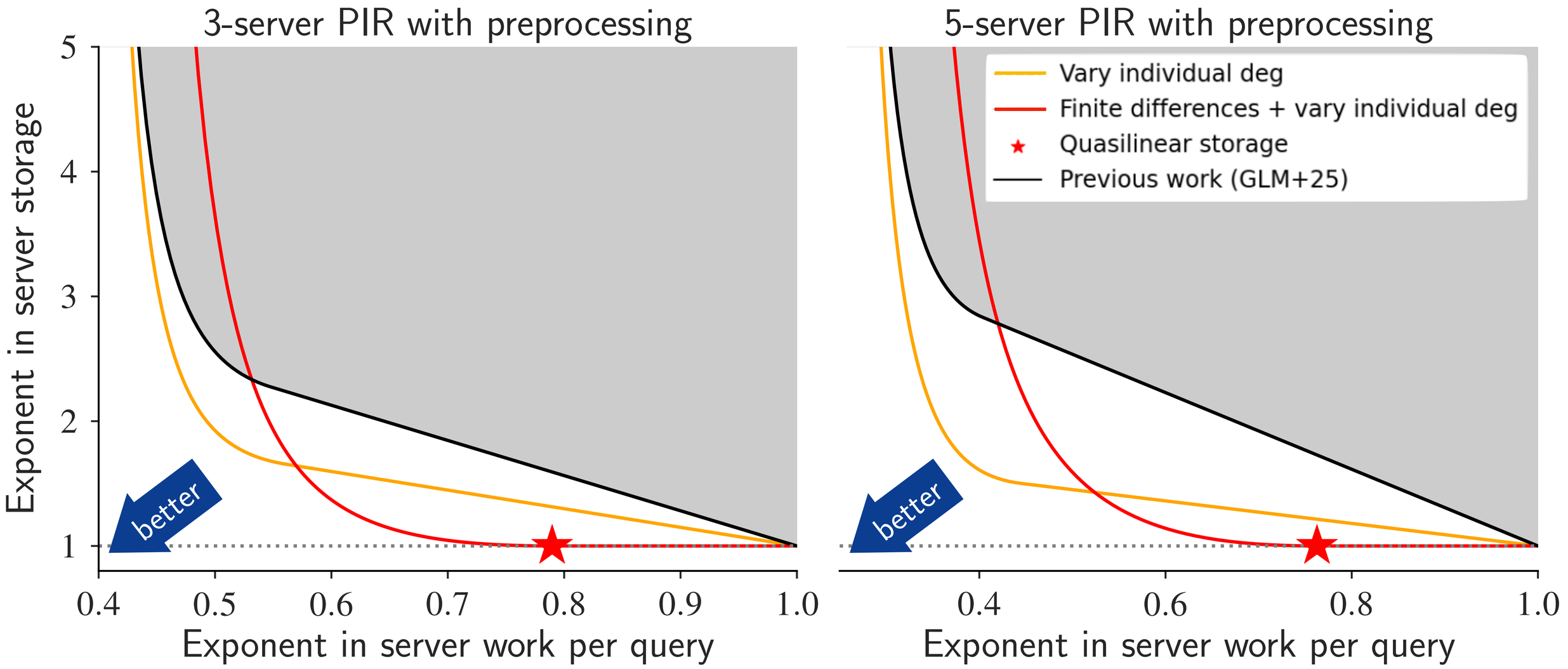
$$\alpha = 1 + \frac{1}{\log s} - \left(\frac{s+1}{2s}\right)\frac{\log(s+1)}{\log s} \approx \frac{1}{2} + \frac{1}{\log s} \text{ as } s \text{ grows}$$

Special Case: Storage $n^{1+o(1)}$, Individual Degree $s-1$

Theorem: for constant prime s, we get information-theoretic s-server PIR with:

- server storage $n^{1+o(1)}$ and
- communication and time per query $n^{\alpha+o(1)}$, where

$$\alpha = 1 + \frac{1}{\log s} - \left(\frac{s+1}{2s}\right)\frac{\log(s+1)}{\log s} \approx \frac{1}{2} + \frac{1}{\log s} \text{ as } s \text{ grows}$$


3-server PIR with preprocessing
5-server PIR with preprocessing
Exponent in server storage
Exponent in server work per query
Vary individual deg
Finite differences + vary individual deg
Quasilinear storage
Previous work (GLM+25)
better
better
1
2
3
4
5
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Bonus Slides on Locally Decodable Codes

Smooth Locally Decodable Codes [KT00]

Smooth Locally Decodable Codes [KT00]

- Goal: encode a message in such a way that we can recover any bit of the message with a small number of accesses to the codeword

Smooth Locally Decodable Codes [KT00]

- Goal: encode a message in such a way that we can recover any bit of the message with a small number of accesses to the codeword
- Three functions:

Smooth Locally Decodable Codes [KT00]

- Goal: encode a message in such a way that we can recover any bit of the message with a small number of accesses to the codeword
- Three functions:
  - Encode(msg $\in \{0,1\}^n$) $\rightarrow c \in \Sigma^\ell$

Cheatsheet

n: message length

ℓ: codeword length

Σ: alphabet

q: locality

Smooth Locally Decodable Codes [KT00]

- Goal: encode a message in such a way that we can recover any bit of the message with a small number of accesses to the codeword
- Three functions:
  - Encode(msg $\in \{0,1\}^n$) $\to c \in \Sigma^\ell$
  - Randomised Query($i \in [n]$) $\to$ qu $\in [\ell]^q$

Cheatsheet

n: message length

ℓ: codeword length

Σ: alphabet

q: locality

Smooth Locally Decodable Codes [KT00]

- Goal: encode a message in such a way that we can recover any bit of the message with a small number of accesses to the codeword
- Three functions:
  - Encode(msg $\in \{0,1\}^n$) $\to c \in \Sigma^\ell$
  - Randomised Query($i \in [n]$) $\to$ qu $\in [\ell]^q$
  - Decode($i \in [n]$, qu $\in [\ell]^q$, c[qu]) $\to$ msg[i]

Cheatsheet

n: message length

ℓ: codeword length

Σ: alphabet

q: locality

Smooth Locally Decodable Codes [KT00]

- Goal: encode a message in such a way that we can recover any bit of the message with a small number of accesses to the codeword
- Three functions:
  - $\mathsf{Encode}(\mathsf{msg} \in \{0,1\}^n) \to c \in \Sigma^\ell$
  - Randomised $\mathsf{Query}(i \in [n]) \to \mathsf{qu} \in [\ell]^q$
  - $\mathsf{Decode}(i \in [n], \mathsf{qu} \in [\ell]^q, c[\mathsf{qu}]) \to \mathsf{msg}[i]$
- **Smoothness:** marginal distribution of qu_j is uniform over $[\ell]$ for all j

Cheatsheet

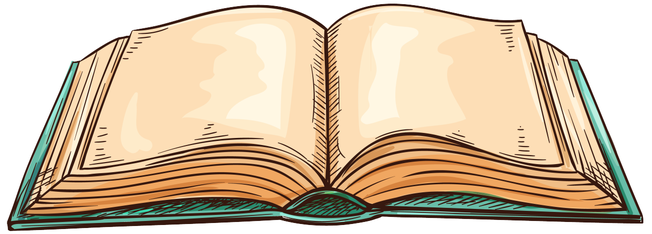
n: message length

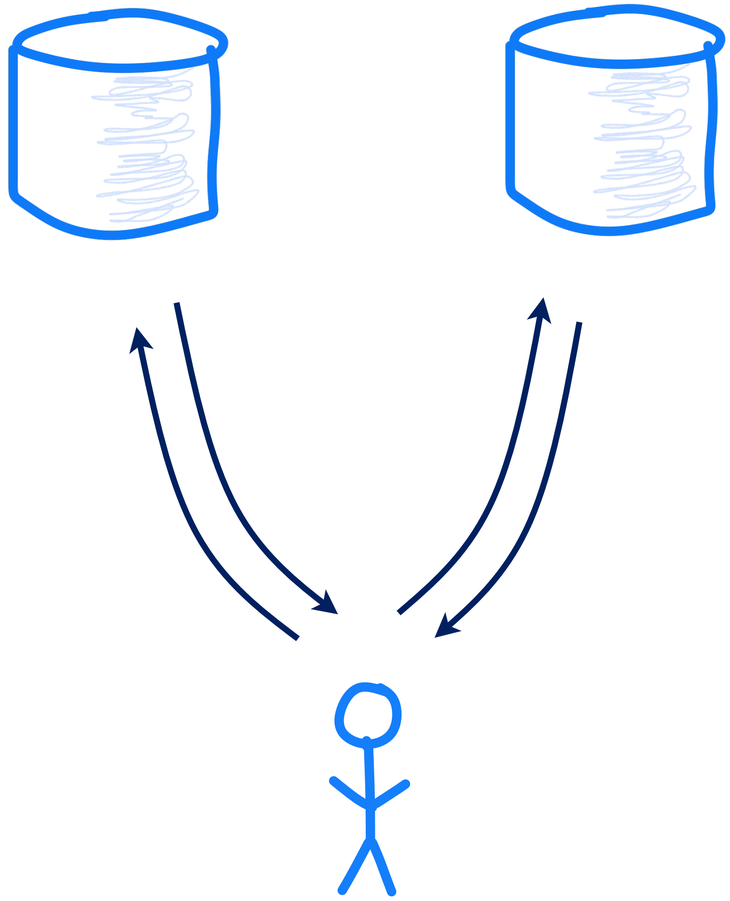
ℓ: codeword length

Σ: alphabet

q: locality

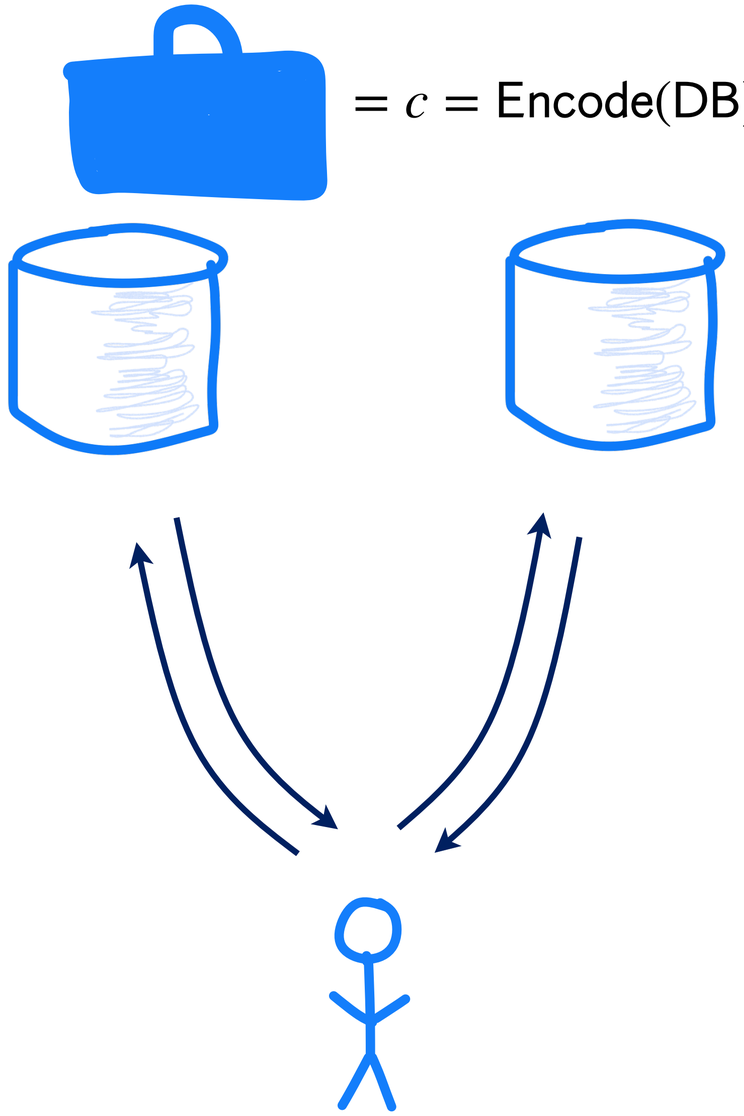
2-query LDCs → 2-server PIR

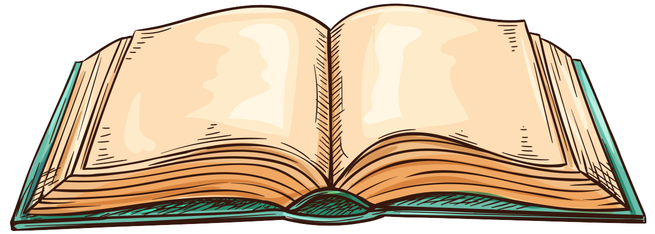




LDCs	PIR
Message	DB
Encoding	Preprocessing
Decoding	Reconstruction

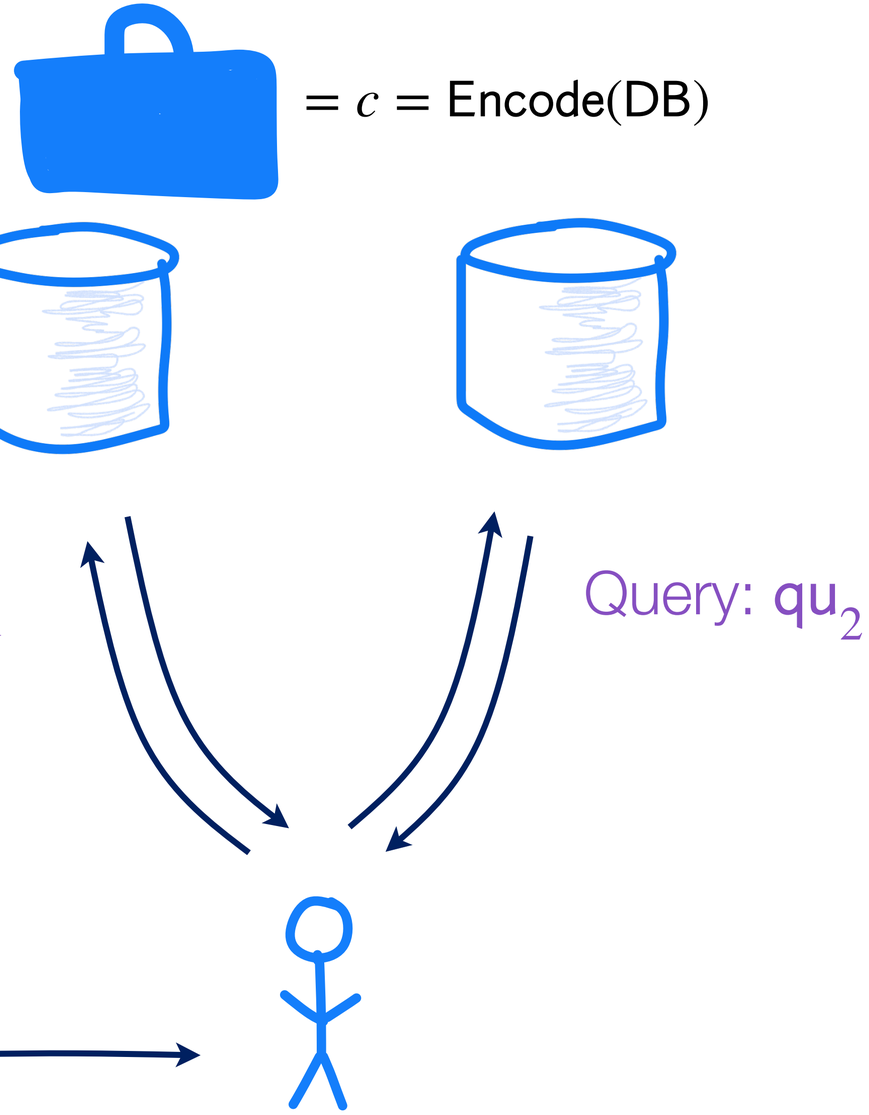
2-query LDCs → 2-server PIR

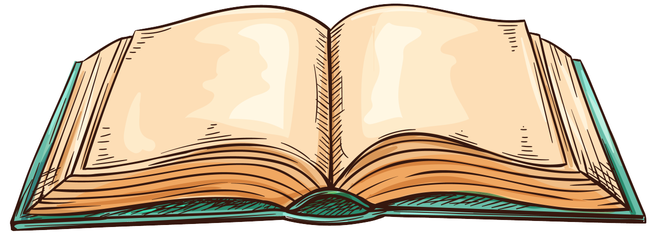




LDCs	PIR
Message	DB
Encoding	Preprocessing
Decoding	Reconstruction

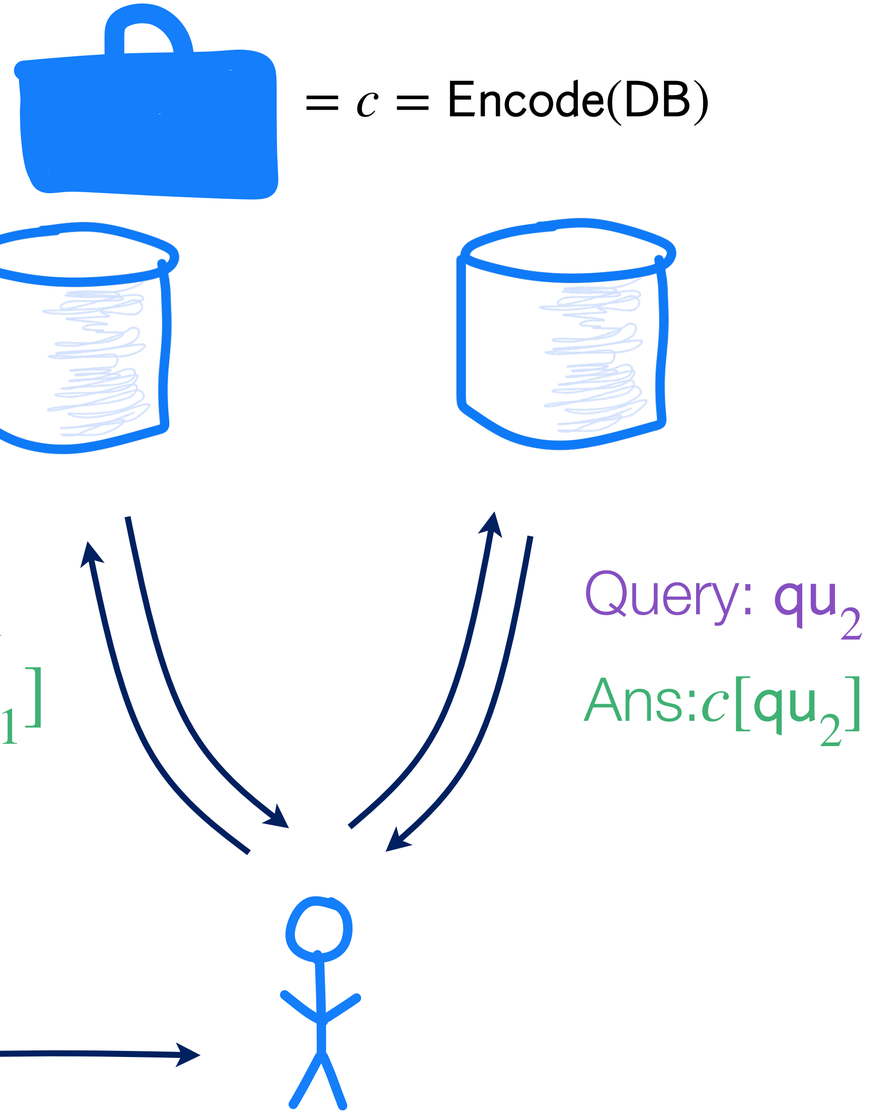
2-query LDCs → 2-server PIR

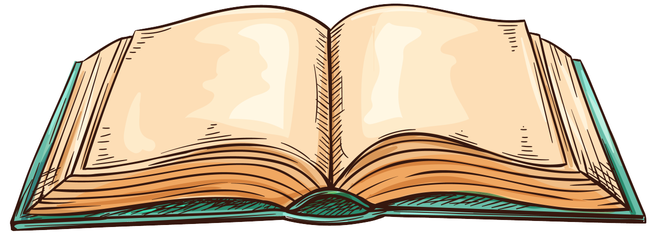




LDCs	PIR
Message	DB
Encoding	Preprocessing
Decoding	Reconstruction

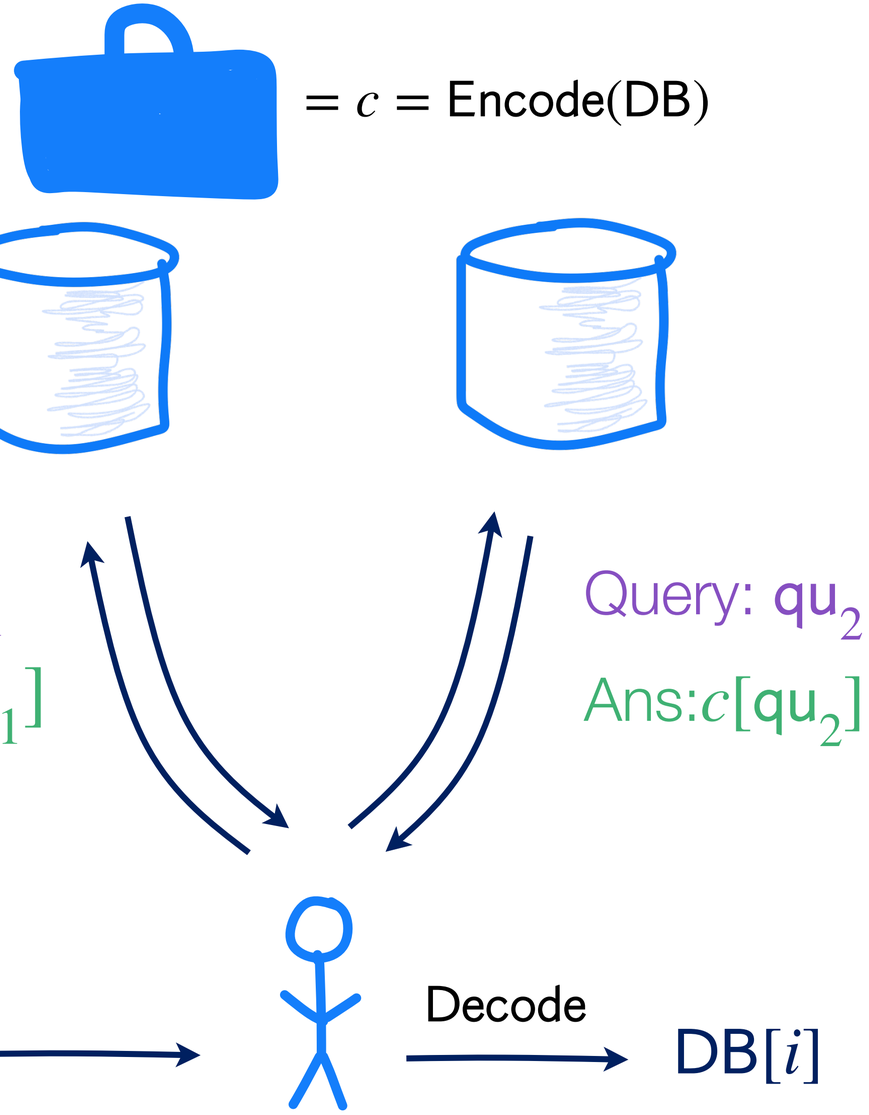
2-query LDCs → 2-server PIR

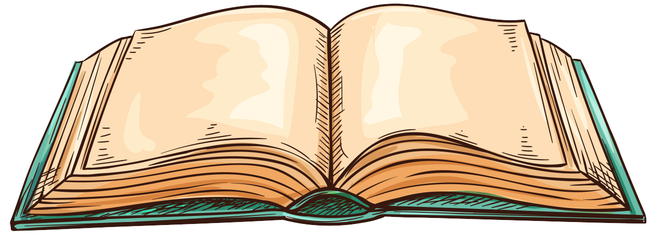




LDCs	PIR
Message	DB
Encoding	Preprocessing
Decoding	Reconstruction

2-query LDCs → 2-server PIR





LDCs	PIR
Message	DB
Encoding	Preprocessing
Decoding	Reconstruction

Casting BIM00 PIR as an LDC

$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

Casting BIM00 PIR as an LDC

- Encode(DB $\in \{0,1\}^n$):
  - Interpolate $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of total degree D
  - $\ell = 2^m, \Sigma = \{0,1\}^{\binom{m}{D/2}}$

$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

Casting BIM00 PIR as an LDC

- Encode(DB $\in \{0,1\}^n$):
  - Interpolate $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of total degree D
  - $\ell = 2^m, \Sigma = \{0,1\}^{\binom{m}{D/2}}$
- Query($i \in [n]$): calculate $\mathbf{p} = \mathsf{E}(i) \in \mathbb{F}_2^m$ and query $\mathbf{r}, \mathbf{r} + \mathbf{p}$

$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

Casting BIM00 PIR as an LDC

- Encode(DB $\in \{0,1\}^n$):
  - Interpolate $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of total degree D
  - $\ell = 2^m, \Sigma = \{0,1\}^{\binom{m}{D/2}}$
- Query($i \in [n]$): calculate $\mathbf{p} = \mathsf{E}(i) \in \mathbb{F}_2^m$ and query $\mathbf{r}, \mathbf{r} + \mathbf{p}$
- **Decode**: Hermite interpolation

$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

Casting BIM00 PIR as an LDC

- Encode(DB $\in \{0,1\}^n$):
  - Interpolate $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of total degree D
  - $\ell = 2^m, \Sigma = \{0,1\}^{\binom{m}{D/2}}$
- Query($i \in [n]$): calculate $\mathbf{p} = \mathsf{E}(i) \in \mathbb{F}_2^m$ and query $\mathbf{r}, \mathbf{r} + \mathbf{p}$
- **Decode**: Hermite interpolation

$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

Q: Does our finite differences technique imply a new 2-query LDC?

Casting BIM00 PIR as an LDC

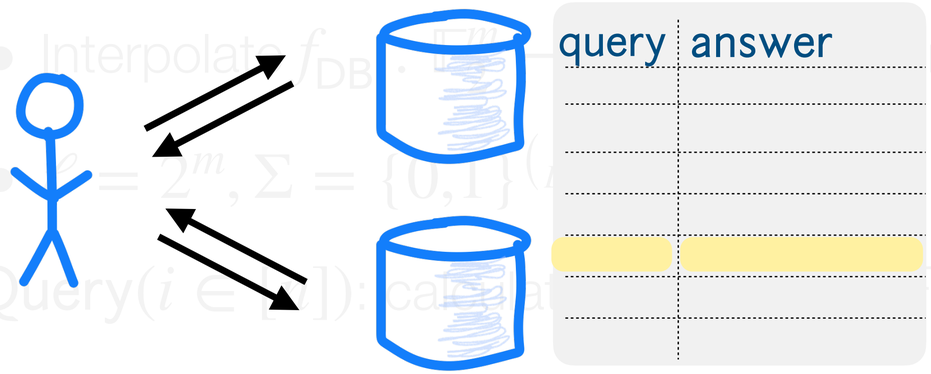
- Encode(DB $\in \{0,1\}^n$):
  - Interpolate $f_{\mathsf{DB}} : \mathbb{F}_2^m \to \mathbb{F}_2$ of total degree D
  - $\ell = 2^m, \Sigma = \{0,1\}^{\binom{m}{D/2}}$
- Query($i \in [n]$): calculate $\mathbf{p} = \mathsf{E}(i) \in \mathbb{F}_2^m$ and query $\mathbf{r}, \mathbf{r} + \mathbf{p}$
- **Decode**: Hermite interpolation

$\mathbf{1}$	$f_{\mathsf{DB}}(\mathbf{1}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{1})$
$\mathbf{2}$	$f_{\mathsf{DB}}(\mathbf{2}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2})$
$\mathbf{2^m}$	$f_{\mathsf{DB}}(\mathbf{2^m}), \ldots, \nabla^{\lfloor D/2 \rfloor} f_{\mathsf{DB}}(\mathbf{2^m})$

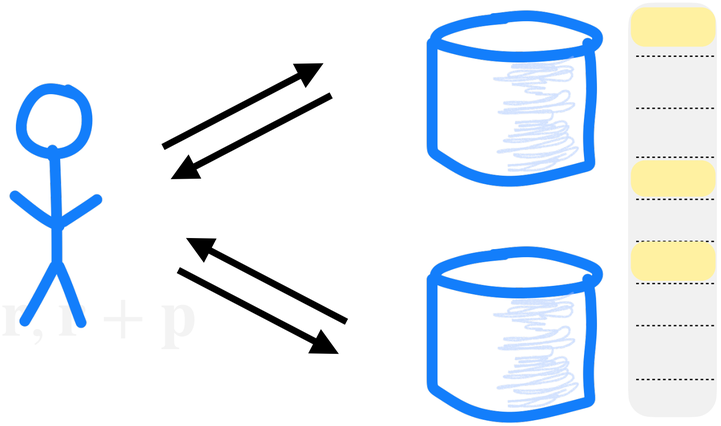
Q: Does our finite differences technique imply a new 2-query LDC?
A: Actually no! LDCs are rigid: they require you to separately write out the answer for every query

Casting BIM00 PIR as an LDC

LDCs and BIM00 PIR



Our PIR



Q: Does our finite differences technique imply a new 2-query LDC?
A: Actually no! LDCs are rigid: they require you to separately write out the answer for every query

Defining Batch-Smooth LDCs

Cheatsheet

n: message length

ℓ: codeword length

Σ: alphabet

q: locality

b: number of batches

Defining Batch-Smooth LDCs

- Three functions:
  - Encode(msg $\in \{0,1\}^n$) $\to c \in \Sigma^\ell$
  - Randomised Query($i \in [n]$) $\to$ qu $\in [\ell]^q$
  - Decode($i \in [n]$, qu $\in [\ell]^q$, c[qu]) $\to$ msg[i]
- **Smoothness:** marginal distribution of qu_j is uniform over $[\ell]$ for all j

Cheatsheet

n: message length

ℓ: codeword length

Σ: alphabet

q: locality

b: number of batches

Defining Batch-Smooth LDCs

- Three functions:
  - Encode(msg $\in \{0,1\}^n$) $\to c \in \Sigma^\ell$
  - Randomised Query($i \in [n]$) $\to$ qu $\in [\ell]^q$
  - Decode($i \in [n]$, qu $\in [\ell]^q$, c[qu]) $\to$ msg[i]
- **Smoothness:** marginal distribution of qu_j is uniform over $[\ell]$ for all j
- **Batch-smoothness:** reorganise the queries into b batches:

1	$\mathsf{qu}_1, \ldots, \mathsf{qu}_{q/b}$
2	$\mathsf{qu}_{q/b+1}, \ldots, \mathsf{qu}_{2q/b}$
b	$\mathsf{qu}_{(b-1)q/b+1}, \ldots, \mathsf{qu}_q$

Cheatsheet

n: message length

ℓ: codeword length

Σ: alphabet

q: locality

b: number of batches

Defining Batch-Smooth LDCs

- Three functions:
  - Encode(msg $\in \{0,1\}^n$) $\to c \in \Sigma^\ell$
  - Randomised Query($i \in [n]$) $\to$ qu $\in [\ell]^q$
  - Decode($i \in [n]$, qu $\in [\ell]^q$, c[qu]) $\to$ msg[i]
- **Smoothness:** marginal distribution of qu_j is uniform over $[\ell]$ for all j
- **Batch-smoothness:** reorganise the queries into b batches:

1	$\mathsf{qu}_1, \ldots, \mathsf{qu}_{q/b}$
2	$\mathsf{qu}_{q/b+1}, \ldots, \mathsf{qu}_{2q/b}$
b	$\mathsf{qu}_{(b-1)q/b+1}, \ldots, \mathsf{qu}_q$

Then each row's distribution should be independent of the index i being queried

Cheatsheet

n: message length

ℓ: codeword length

Σ: alphabet

q: locality

b: number of batches

BIM00 PIR as a Batch-Smooth LDC

Cheatsheet

n: message length

$$\binom{m}{D} \geq n$$

BIM00 PIR as a Batch-Smooth LDC

Cheatsheet

n: message length

$\binom{m}{D} \geq n$

- Number of batches: $b = 2$

BIM00 PIR as a Batch-Smooth LDC

Cheatsheet

n: message length

$\binom{m}{D} \geq n$

- Number of batches: $b = 2$
- Alphabet: $\Sigma = \{0,1\}$

BIM00 PIR as a Batch-Smooth LDC

Cheatsheet

n: message length

$\binom{m}{D} \geq n$

- Number of batches: $b = 2$
- Alphabet: $\Sigma = \{0,1\}$
- Codeword length: $\ell = 2^m \cdot \binom{m}{D/2}$

BIM00 PIR as a Batch-Smooth LDC

Cheatsheet

n: message length

$\binom{m}{D} \geq n$

- Number of batches: $b = 2$
- Alphabet: $\Sigma = \{0,1\}$
- Codeword length: $\ell = 2^m \cdot \binom{m}{D/2}$
- Number of queries: $q = 2\binom{m}{D/2}$

Our PIR as a Batch-Smooth LDC

Cheatsheet

n: message length

$\binom{m}{D} \geq n$

- Number of batches: $b = 2$
- Alphabet: $\Sigma = \{0,1\}$
- Codeword length: $\ell = 2^m \cdot \binom{m}{D/2}$ Finite differences 2^m
- Number of queries: $q = 2\binom{m}{D/2}$

Our PIR as a Batch-Smooth LDC

Cheatsheet

n: message length

$\binom{m}{D} \geq n$

- Number of batches: $b = 2$
- Alphabet: $\Sigma = \{0,1\}$
- Codeword length: $\ell = 2^m \cdot \binom{m}{D/2}$ Finite differences 2^m
- Number of queries: $q = 2\binom{m}{D/2}$

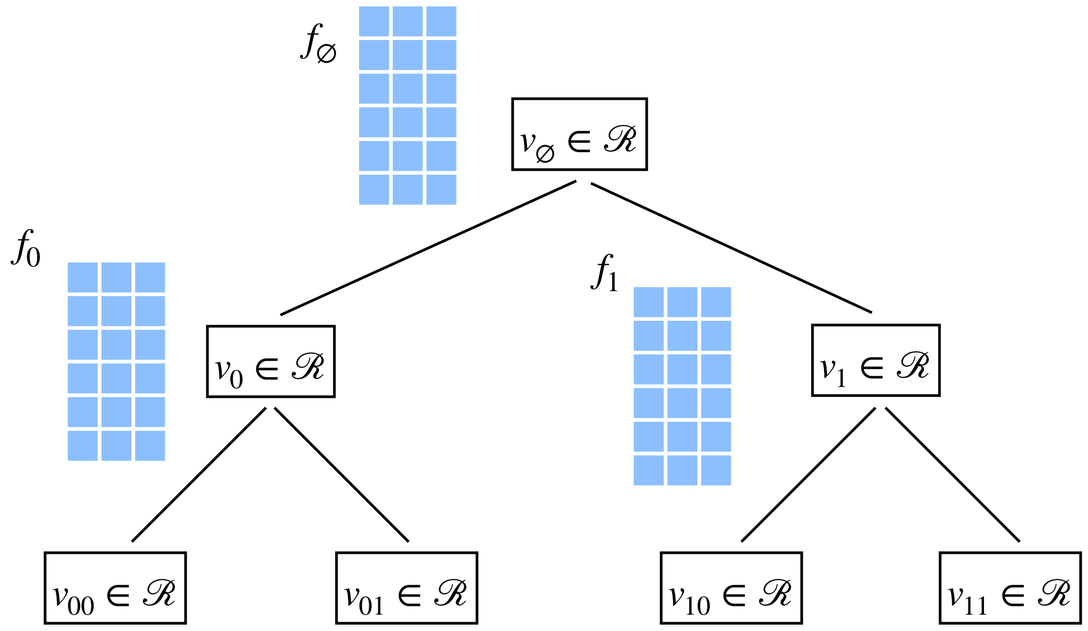
Consequence: the first batch-smooth LDC with constant alphabet size, constant number of batches b, codeword length $n^{1+o(1)}$, and polynomially sublinear number of queries $q = n^{1-\Omega(1)}$

Bonus Slides on TreeEval

From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$



Catalytic tape

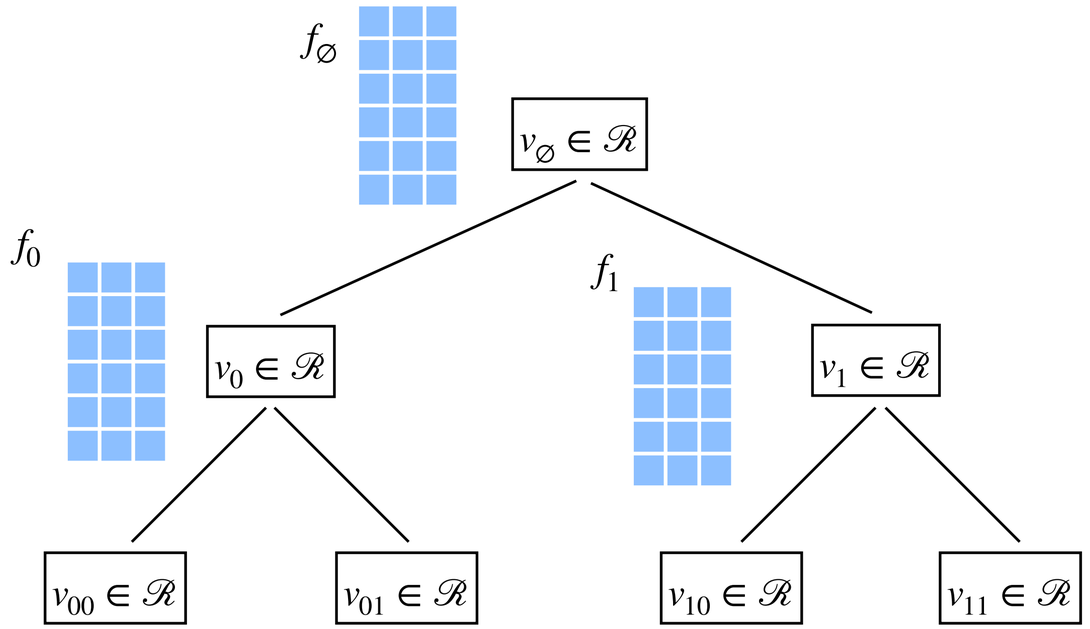
|00110101|10010100|00001110|

From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$



Catalytic tape

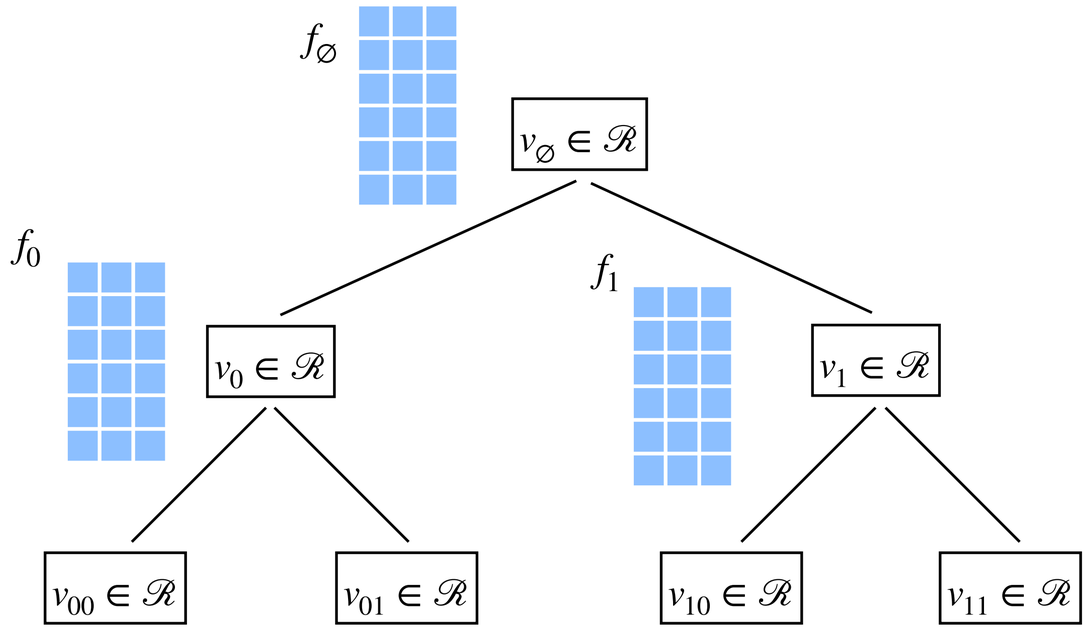
|00110101|10010100|00001110|

From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively



Catalytic tape

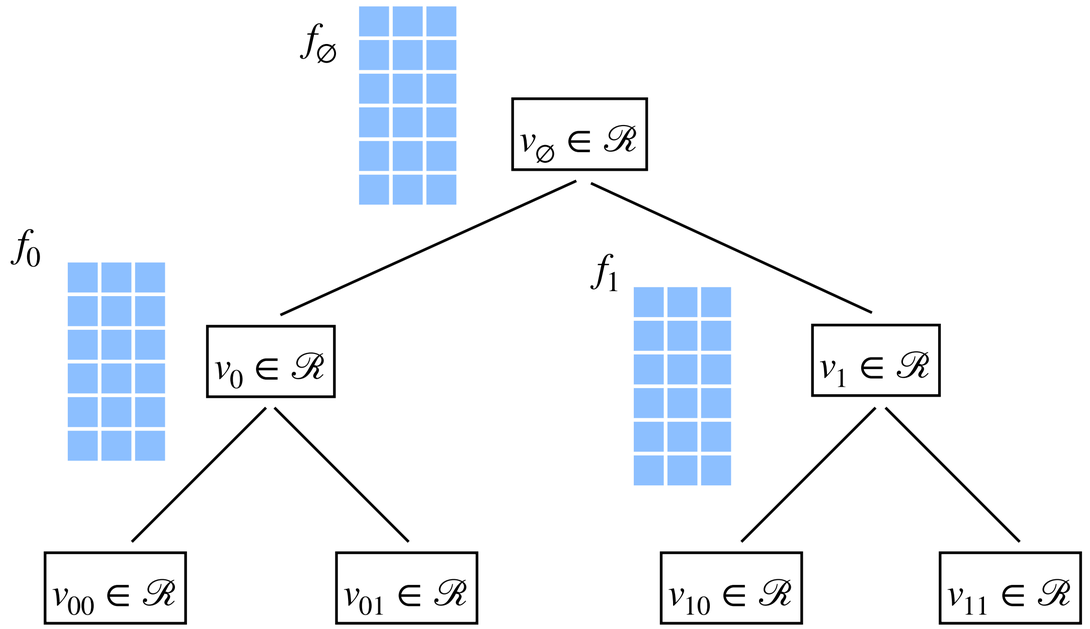
|00110101|10010100|00001110|

From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

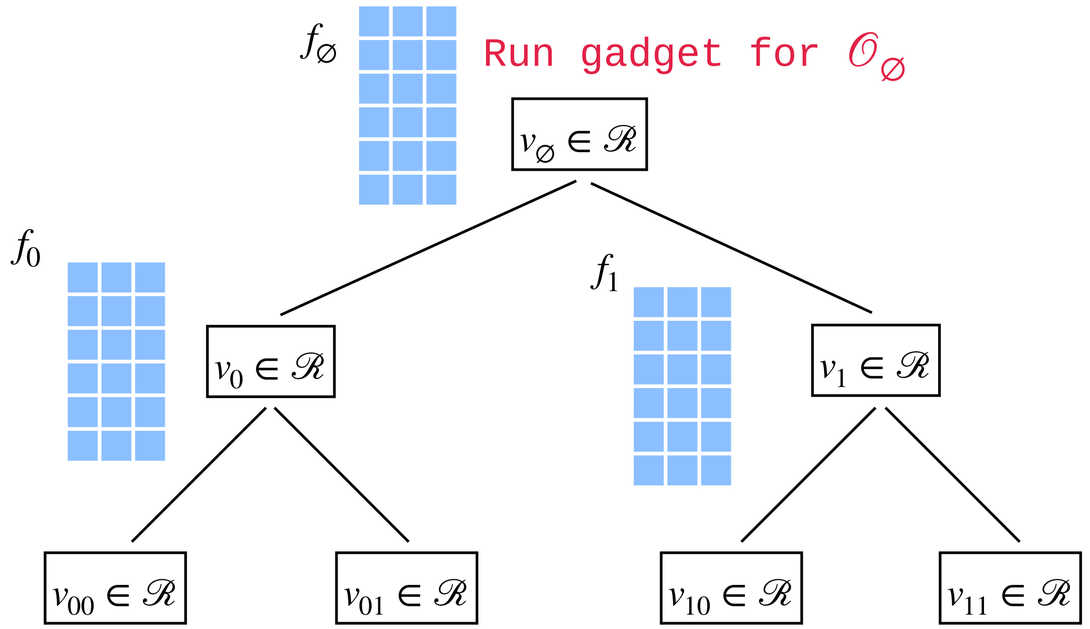
|00110101|10010100|00001110|

From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

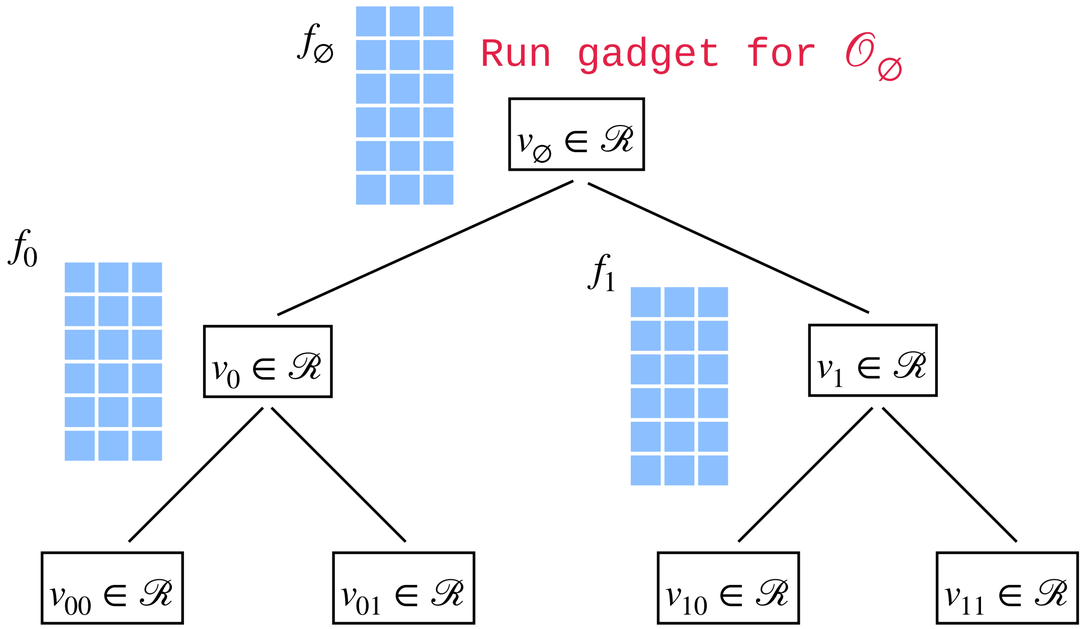
|00110101|10010100|00001110|

From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

|00110101|10010100|00001110|

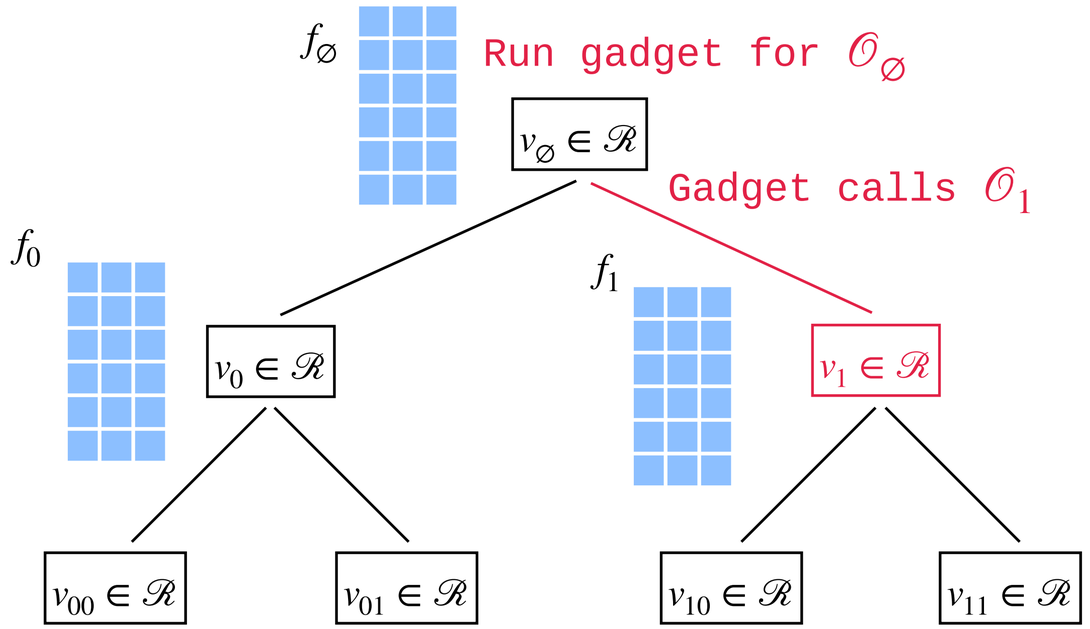
Root
$\varnothing$

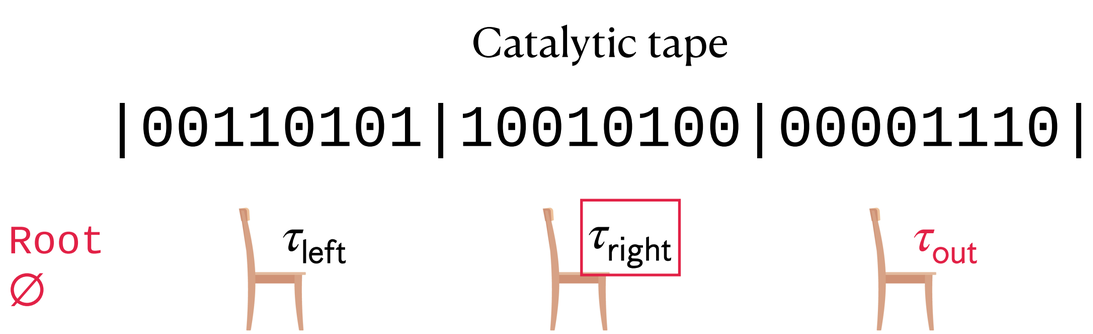
From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



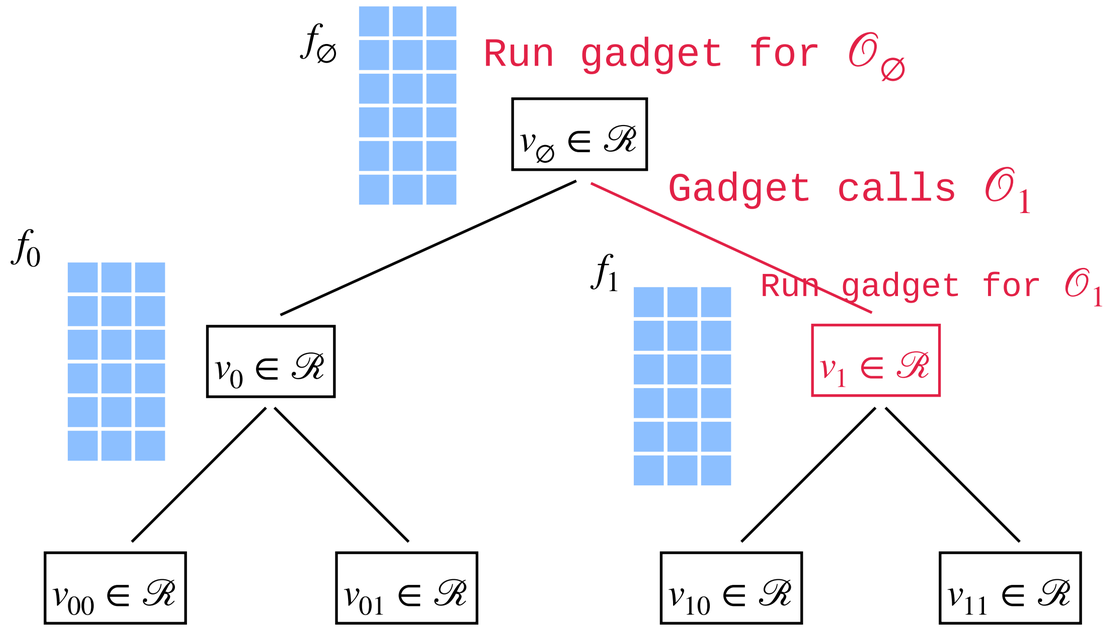


From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

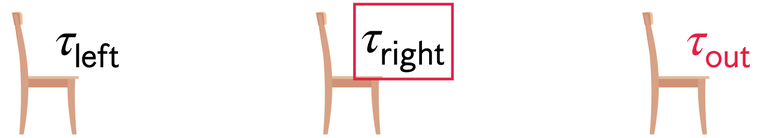
- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

|00110101|10010100|00001110|

Root
$\varnothing$

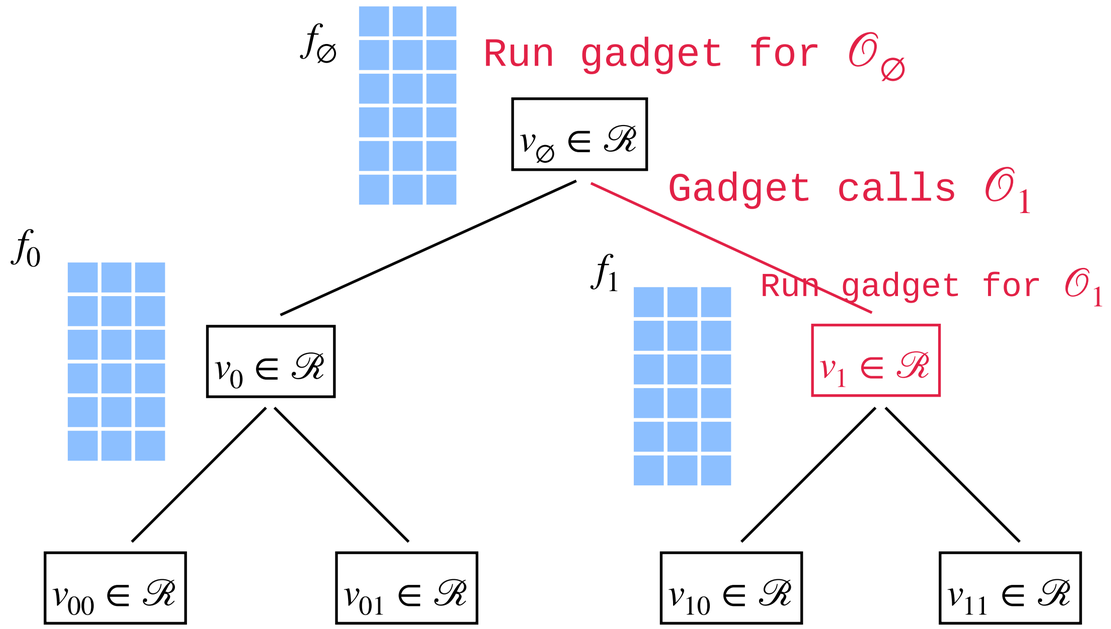


From One Level to Many

Cheatsheet

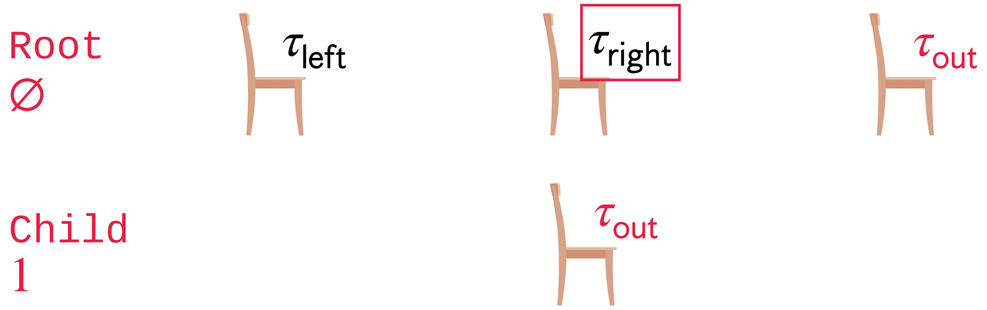
$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

|00110101|10010100|00001110|

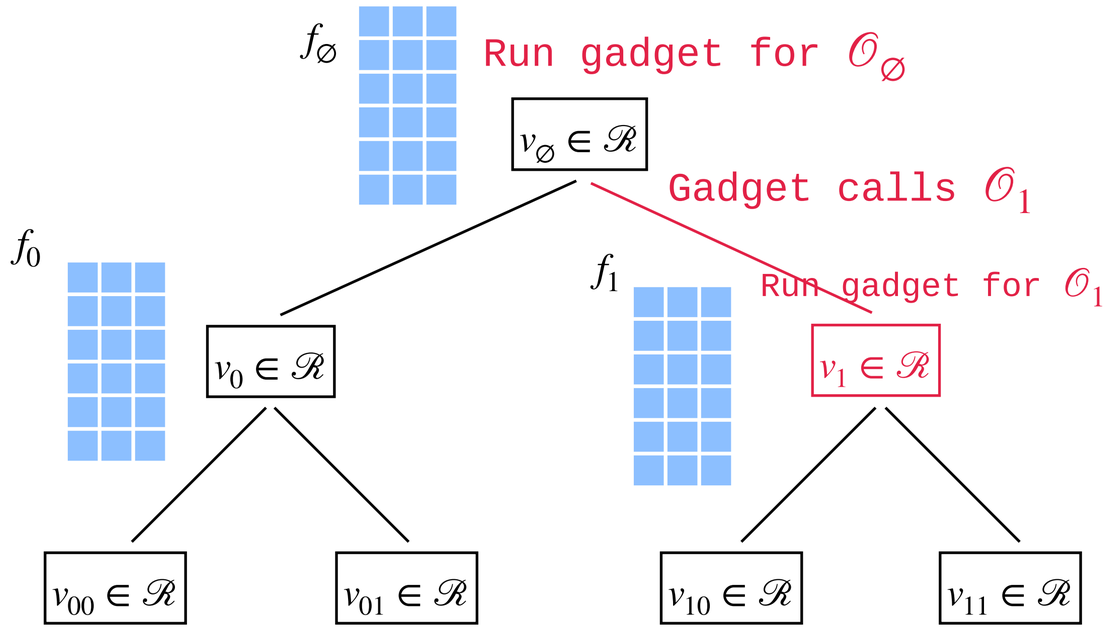


From One Level to Many

Cheatsheet

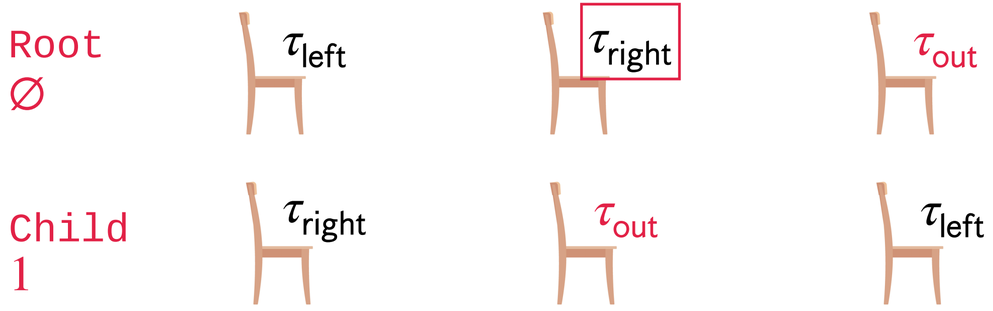
$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

|00110101|10010100|00001110|

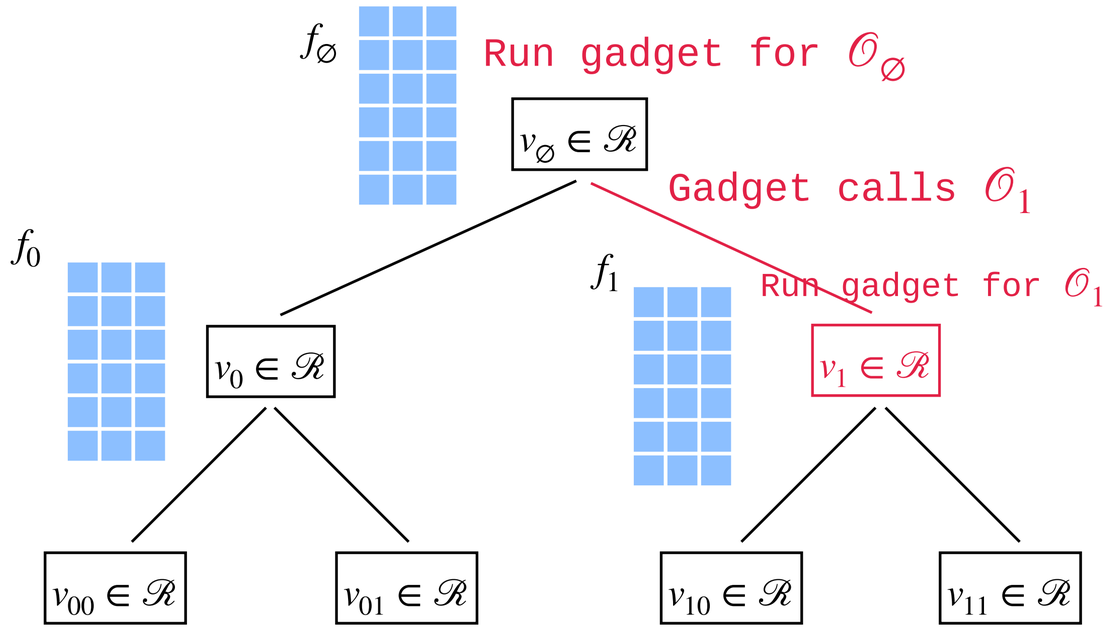


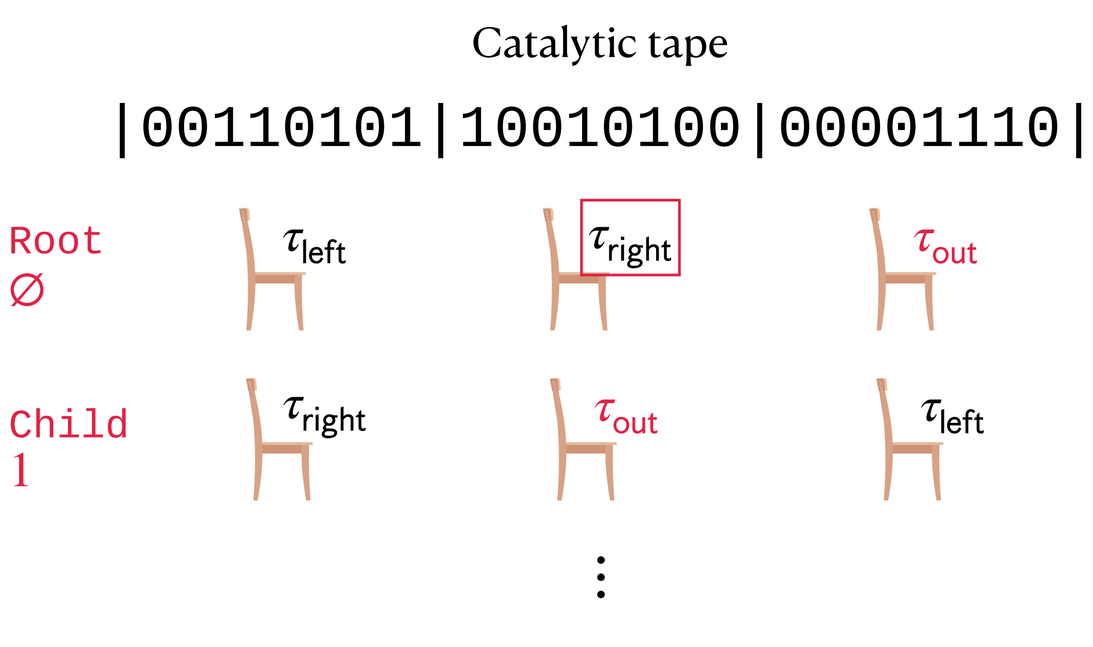
From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



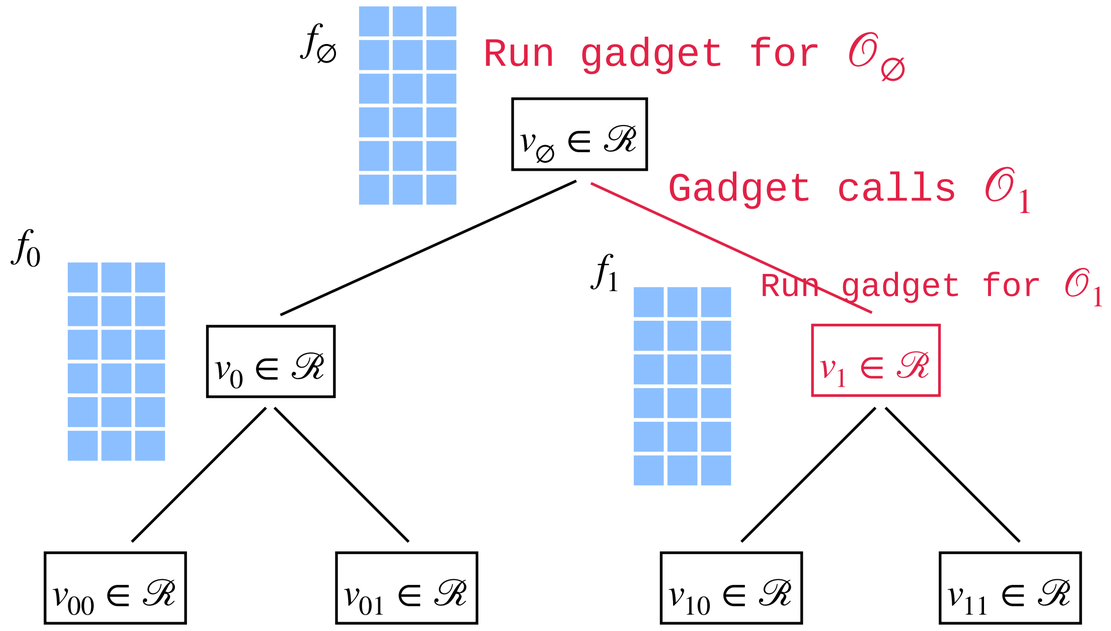


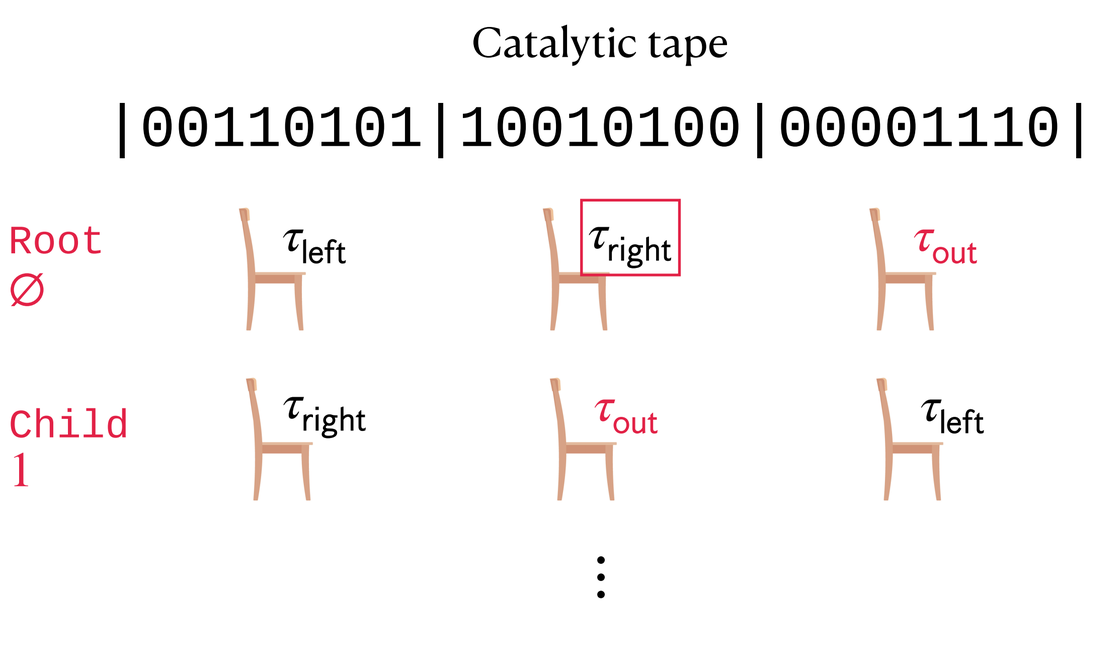
From One Level to Many

Cheatsheet

$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- Catalytic space: $O(\log |\mathcal{R}|)$
- Free space (for low-level information): s per node in a call stack of height $h \to hs$
- Time: q recursive calls at each node in a tree of height $h \to q^h$





The One-Level Gadget: Summary

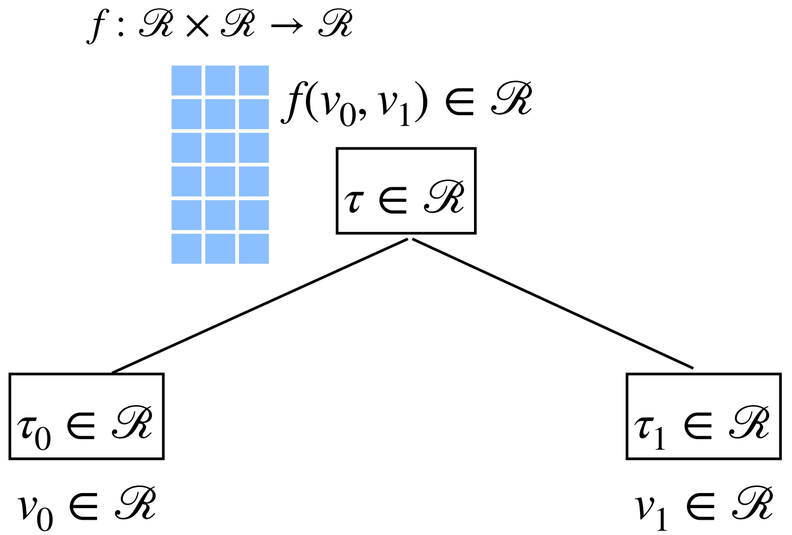
Centrepiece of Cook-Mertz and our work: all I will focus on for the rest of the talk

- Embed $\{0,1\}^\ell$ in some ring $\mathscr{R}$
- Inputs:
  - Control bit $c' \in \{0,1\}$
  - Arbitrary catalytic registers $\tau_0, \tau_1, \tau \in \mathscr{R}$
  - Oracle $\mathcal{O}_{v_0,v_1}(b, c \in \{0,1\})$: updates

$$\tau_b \leftarrow \tau_b + (-1)^c v_b$$

- Output: want the registers to end up in the state

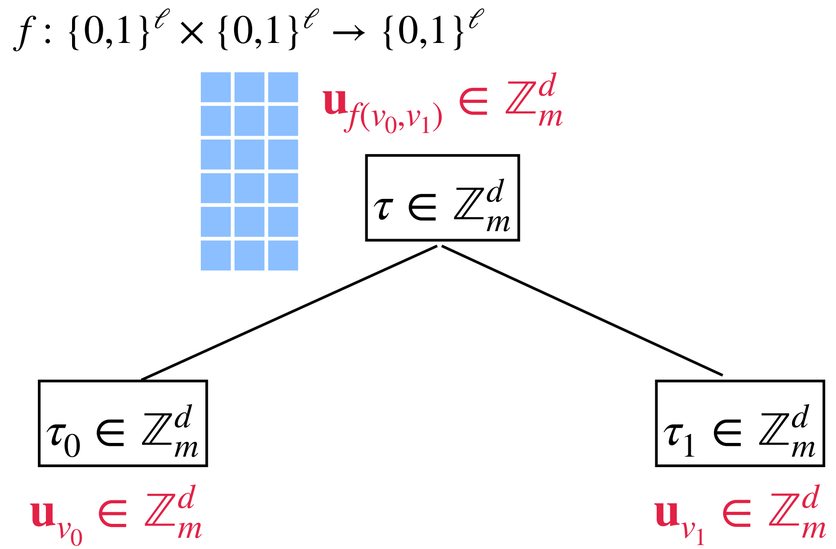
$$(\tau_0, \tau_1, \tau) = (\tau_{0,\text{init}}, \tau_{1,\text{init}}, \tau_{\text{init}} + (-1)^{c'} f(v_0, v_1))$$



Metrics

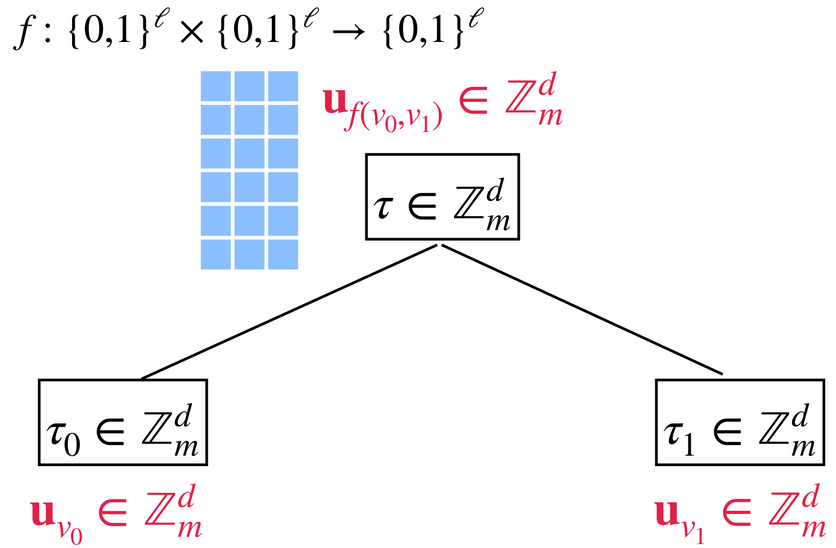
- $\log |\mathscr{R}|$: final catalytic space
- $h \cdot$ [free space]: final free space
- [oracle queries]^h: final runtime

The Fine Print



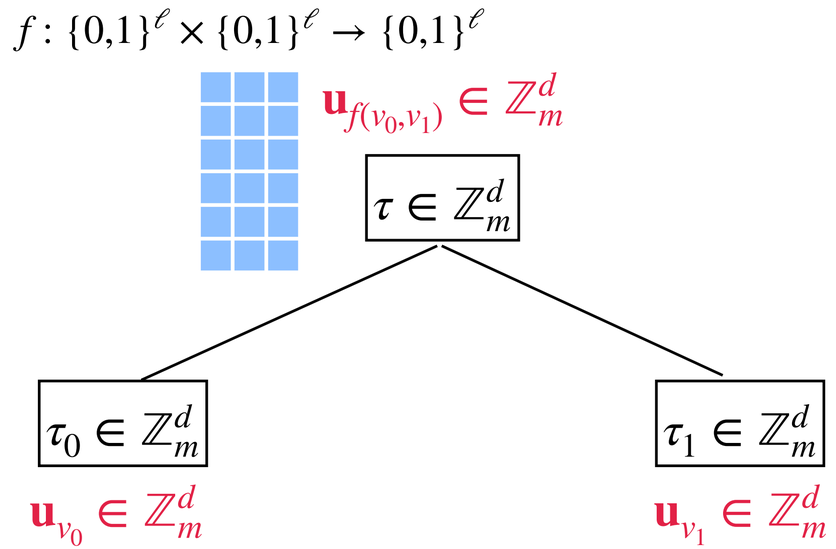
The Fine Print

- Challenge 1 (easy): the PIR scheme needs to be "algebraic" or "additive" to be usable for tree evaluation
  - True for all PIR protocols described in this talk



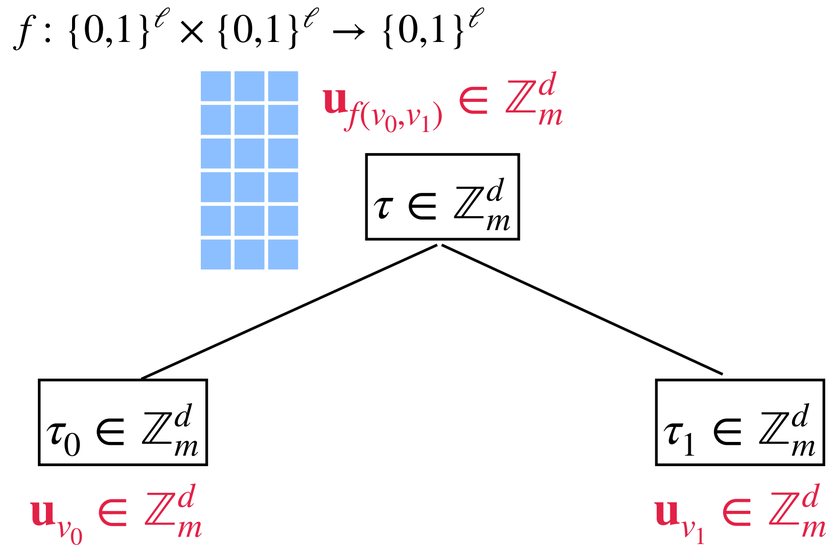
The Fine Print

- Challenge 1 (easy): the PIR scheme needs to be "algebraic" or "additive" to be usable for tree evaluation
  - True for all PIR protocols described in this talk
- Challenge 2 (harder but doable): in the one-level gadget, we can only catalytically write down $\mathbf{u}_{v_0}$ or $\mathbf{u}_{v_1}$



The Fine Print

- Challenge 1 (easy): the PIR scheme needs to be "algebraic" or "additive" to be usable for tree evaluation
  - True for all PIR protocols described in this talk
- Challenge 2 (harder but doable): in the one-level gadget, we can only catalytically write down $\mathbf{u}_{v_0}$ or $\mathbf{u}_{v_1}$
  - For MV PIR: want a MV family of size $2^{2\ell}$ and write down $\mathbf{u}_{v_0||v_1}$ which we can't



The Fine Print

- Challenge 1 (easy): the PIR scheme needs to be "algebraic" or "additive" to be usable for tree evaluation
  - True for all PIR protocols described in this talk
- Challenge 2 (harder but doable): in the one-level gadget, we can only catalytically write down $\mathbf{u}_{v_0}$ or $\mathbf{u}_{v_1}$
  - For MV PIR: want a MV family of size $2^{2\ell}$ and write down $\mathbf{u}_{v_0||v_1}$ which we can't
  - High-level idea: notice that $\{\mathbf{u}_a \otimes \mathbf{u}_b\}_{a,b\in\{0,1\}^\ell}$ is also a MV family that is indexed by $a||b$