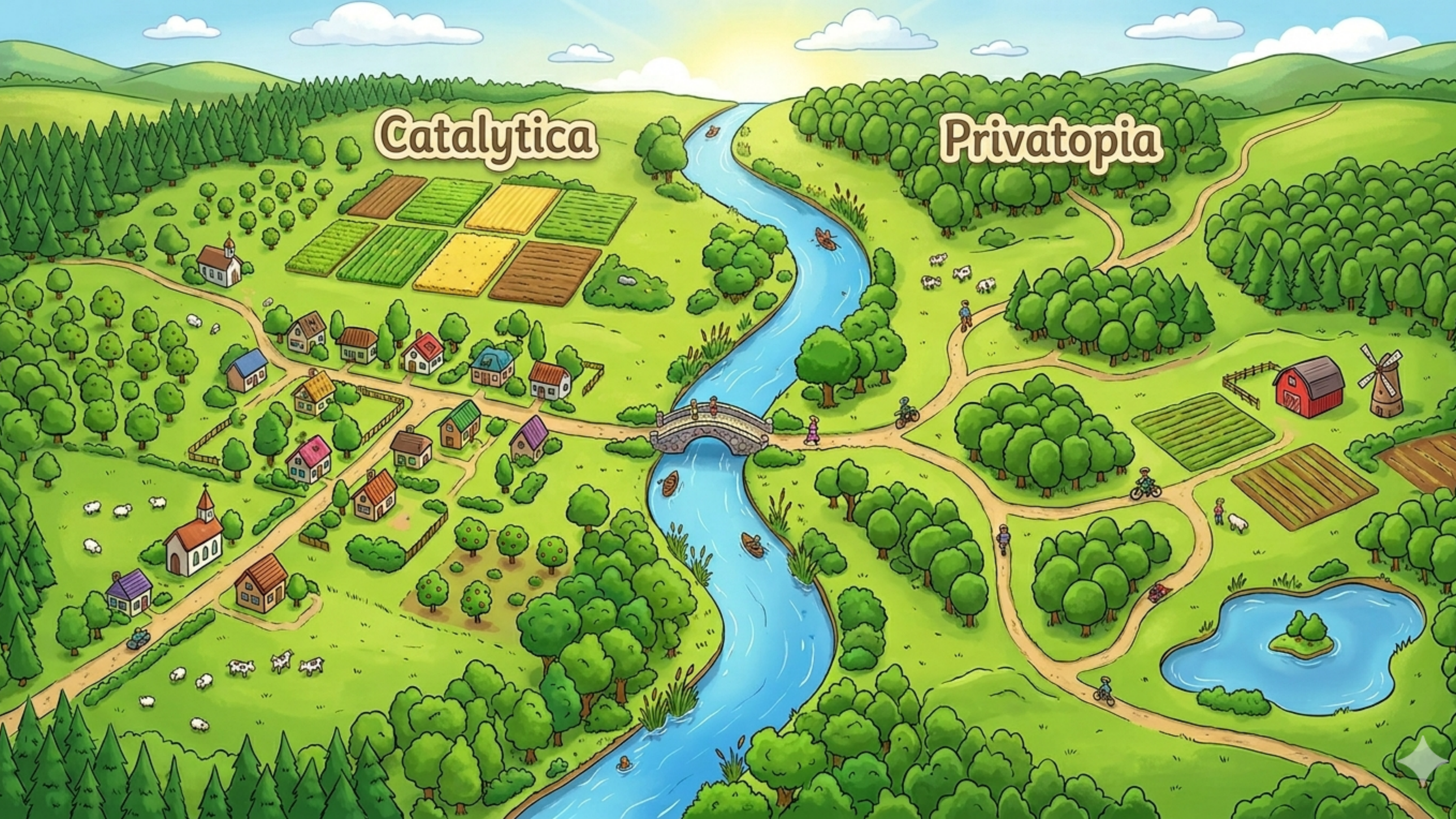
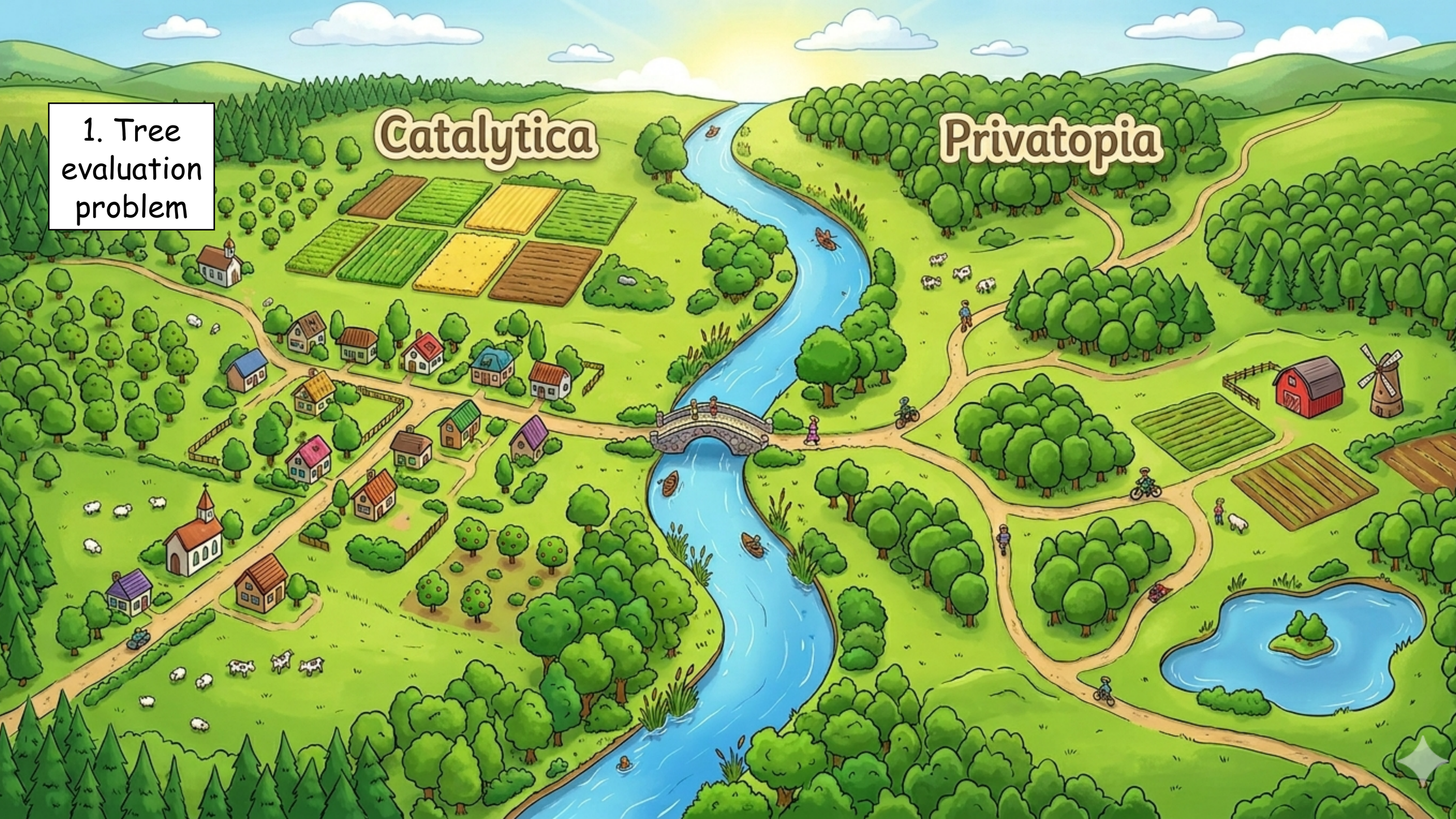
Catalytic Tree Evaluation from Matching Vectors

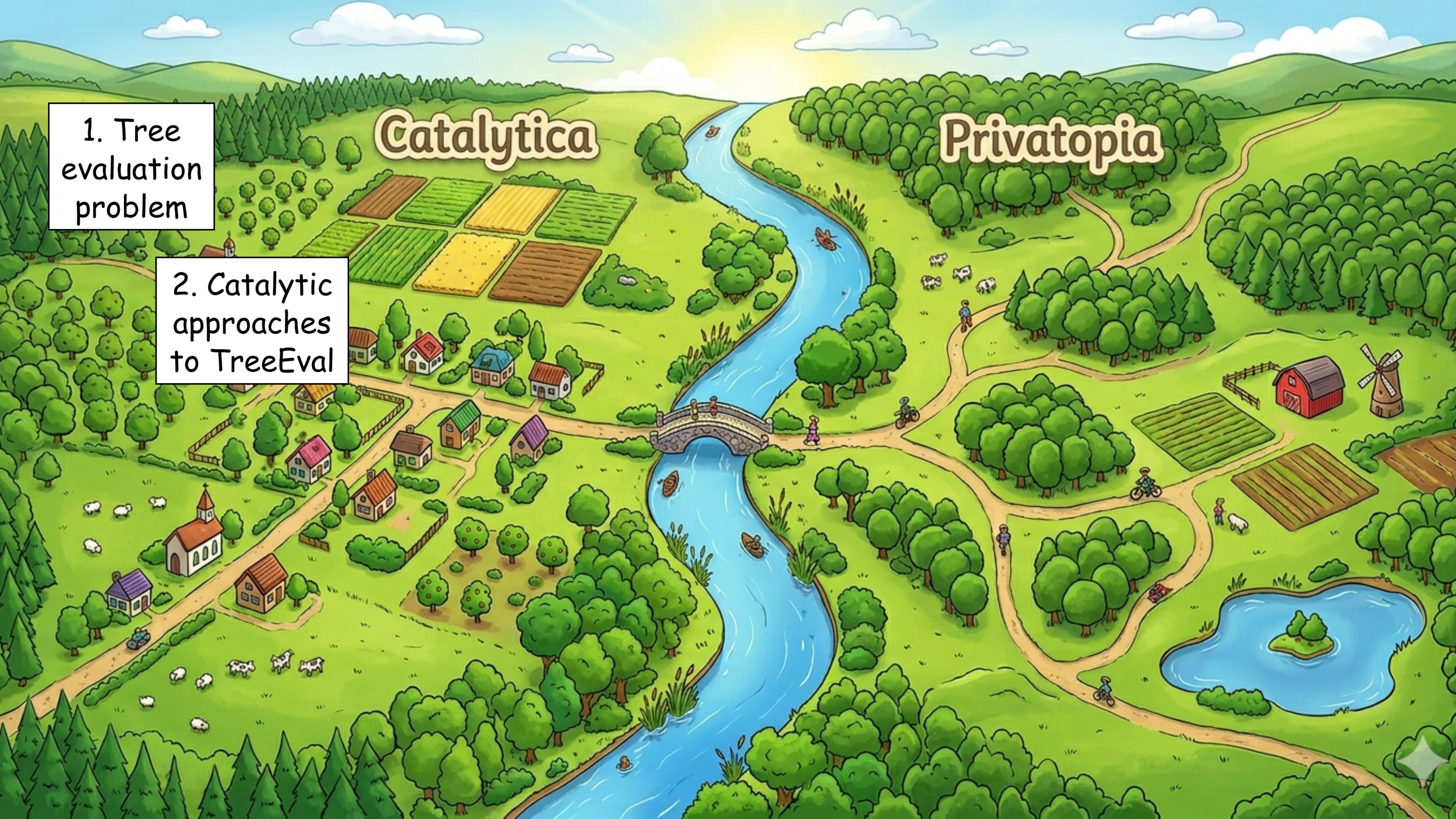
Seyoon Ragavan (MIT)

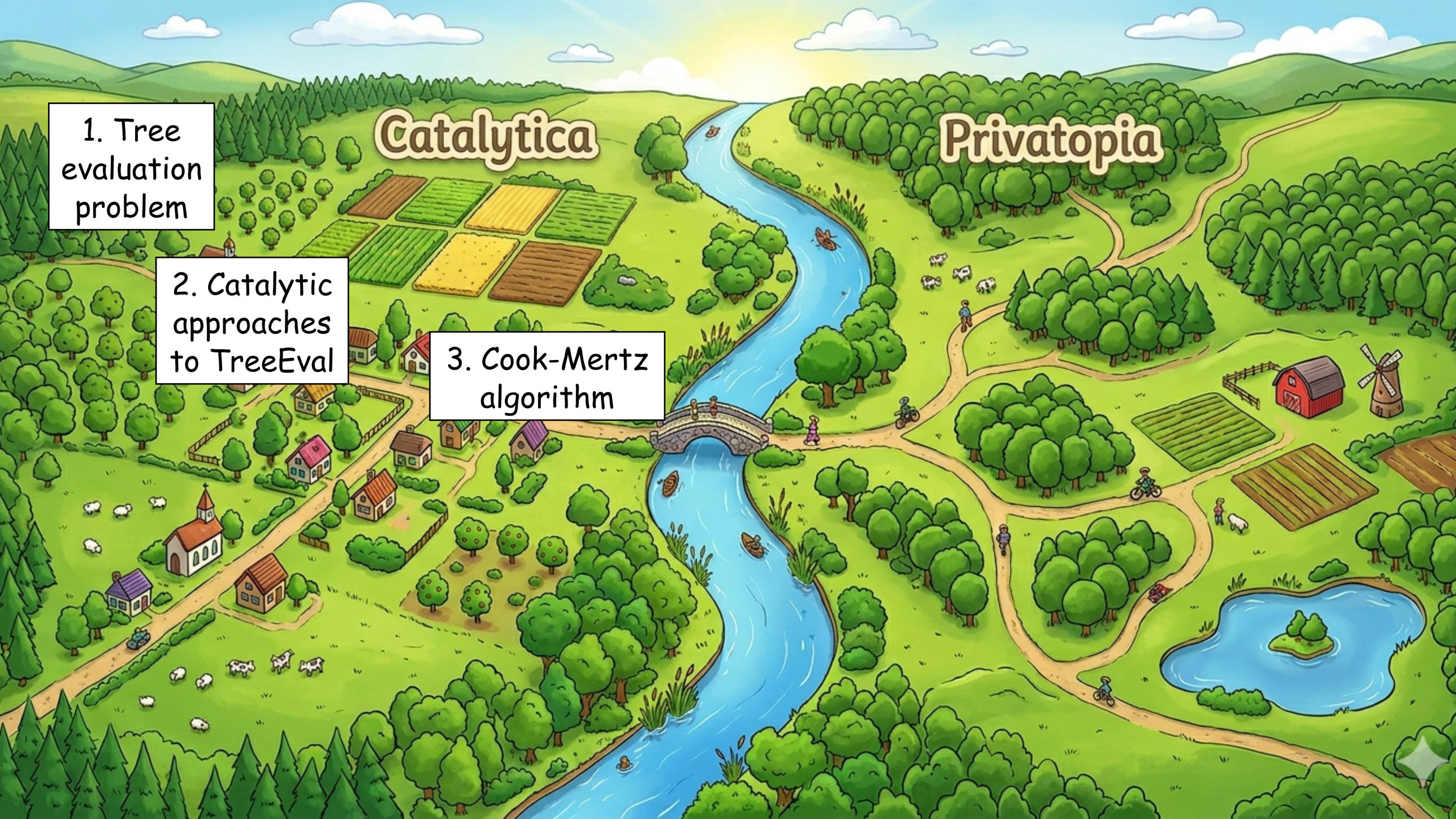
Joint work with Alexandra Henzinger and Ted Pyne (MIT)

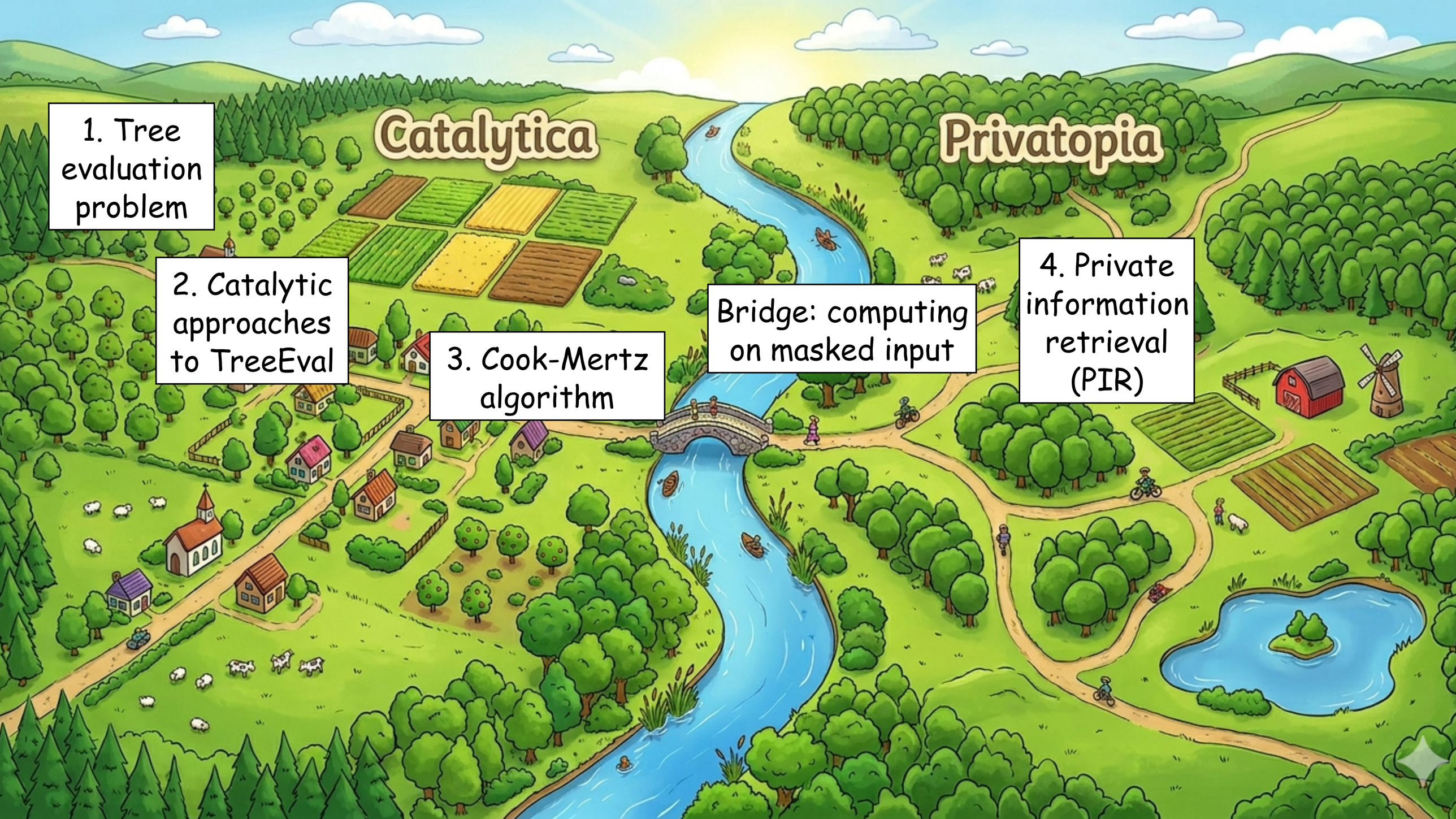
arXiv:2602.14320

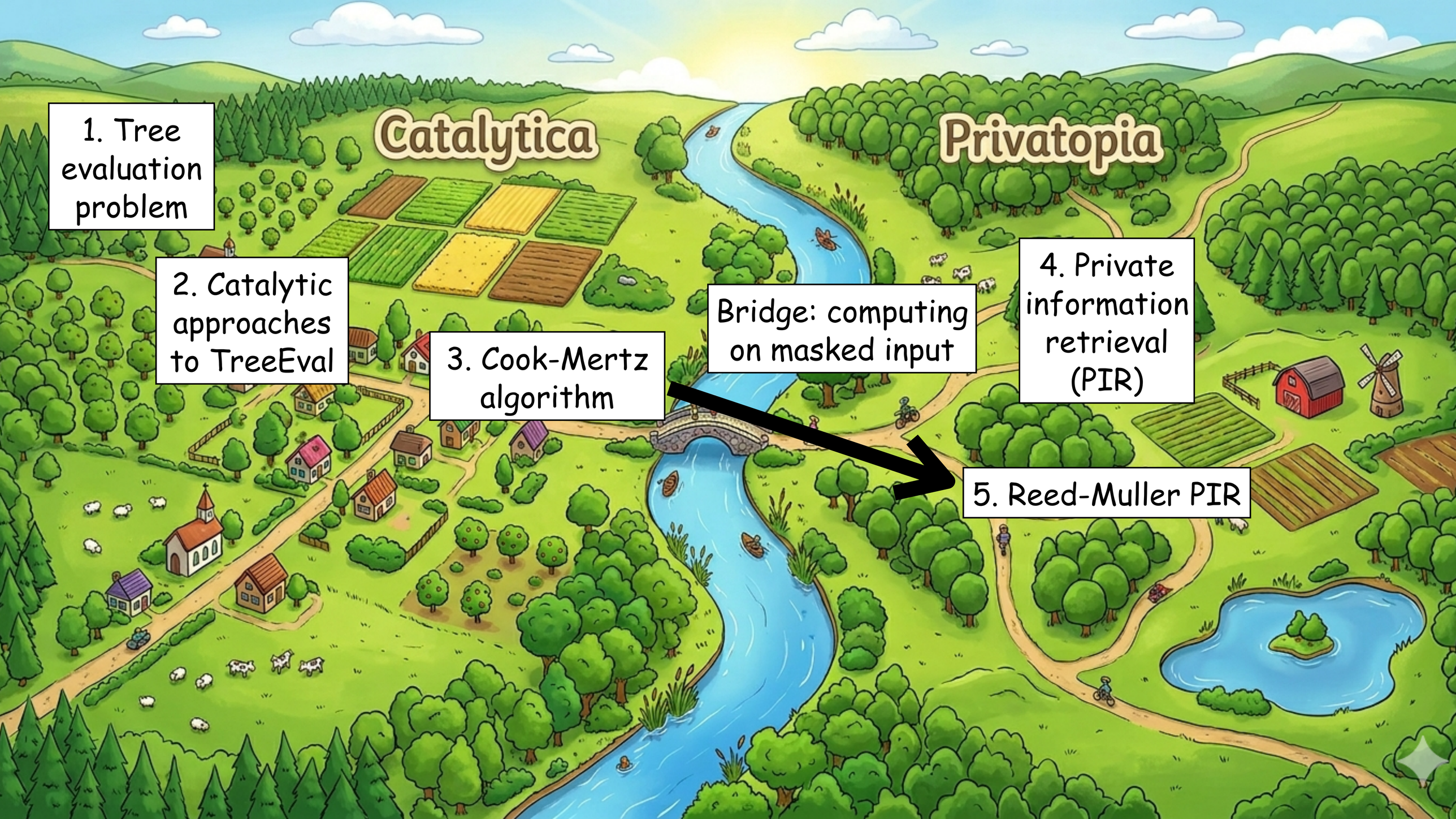

Catalytica
Privatopia

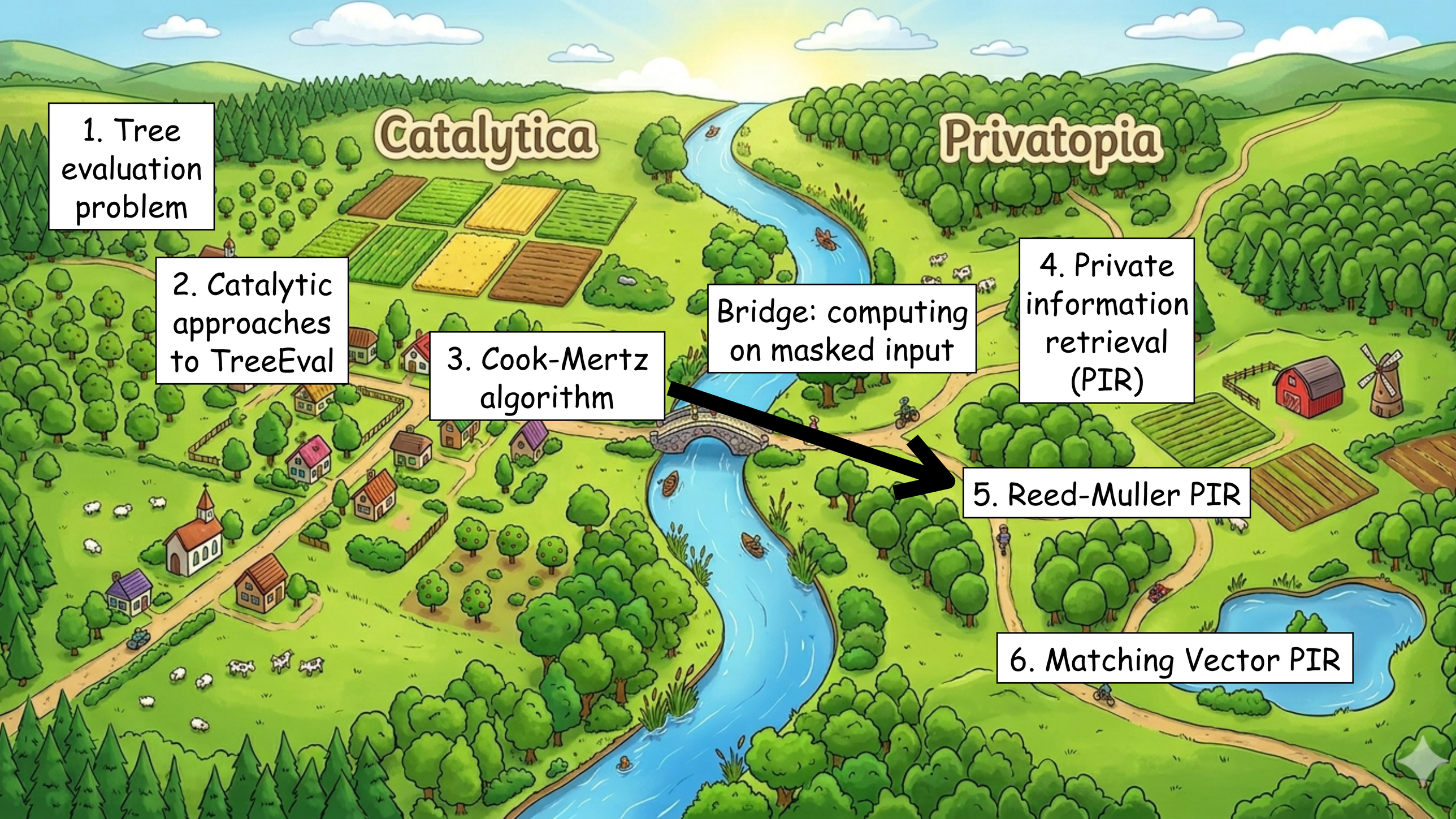

1. Tree evaluation problem
Catalytica
Privatopia

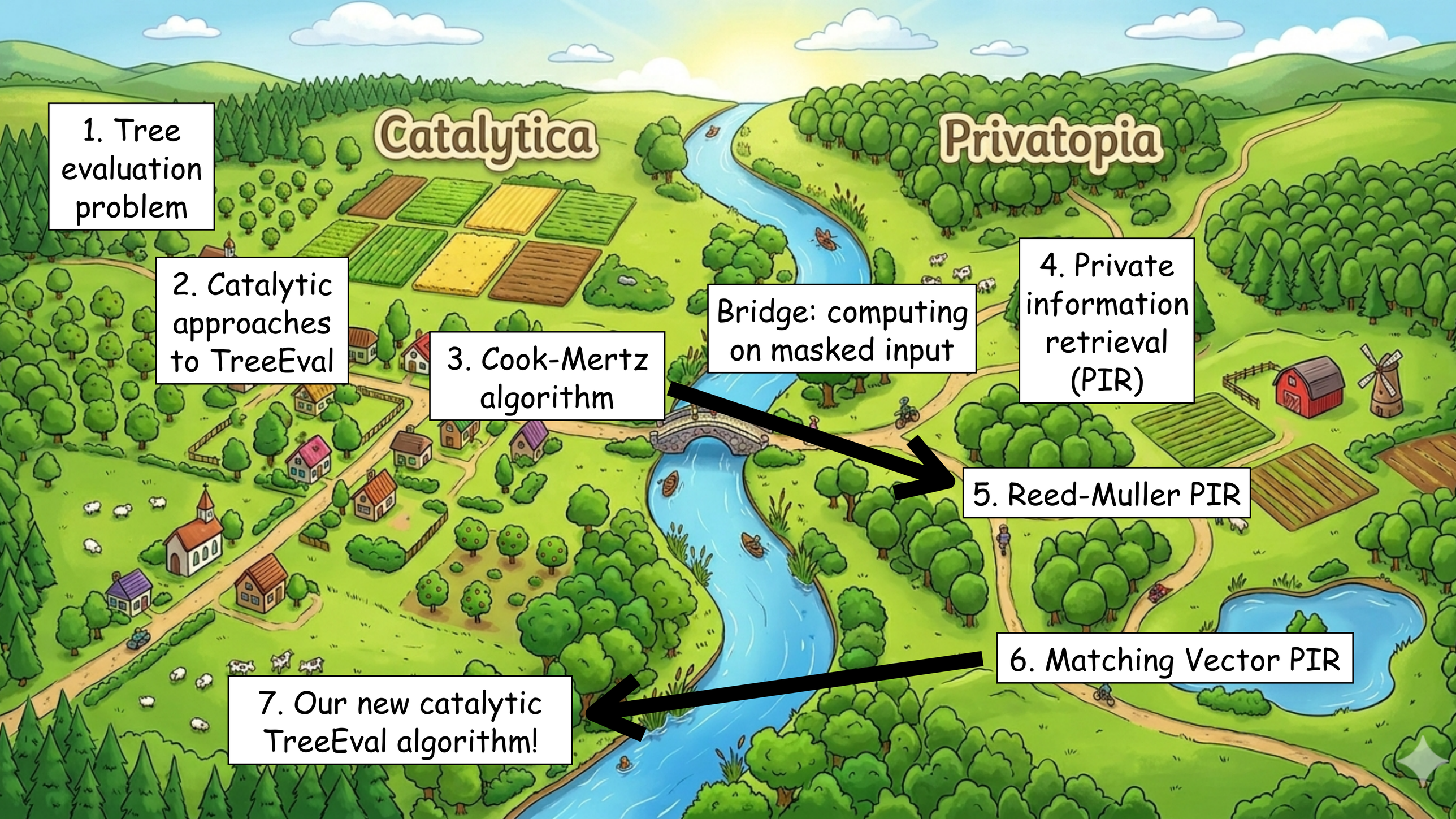

1. Tree evaluation problem
2. Catalytic approaches to TreeEval
Catalytica
Privatopia

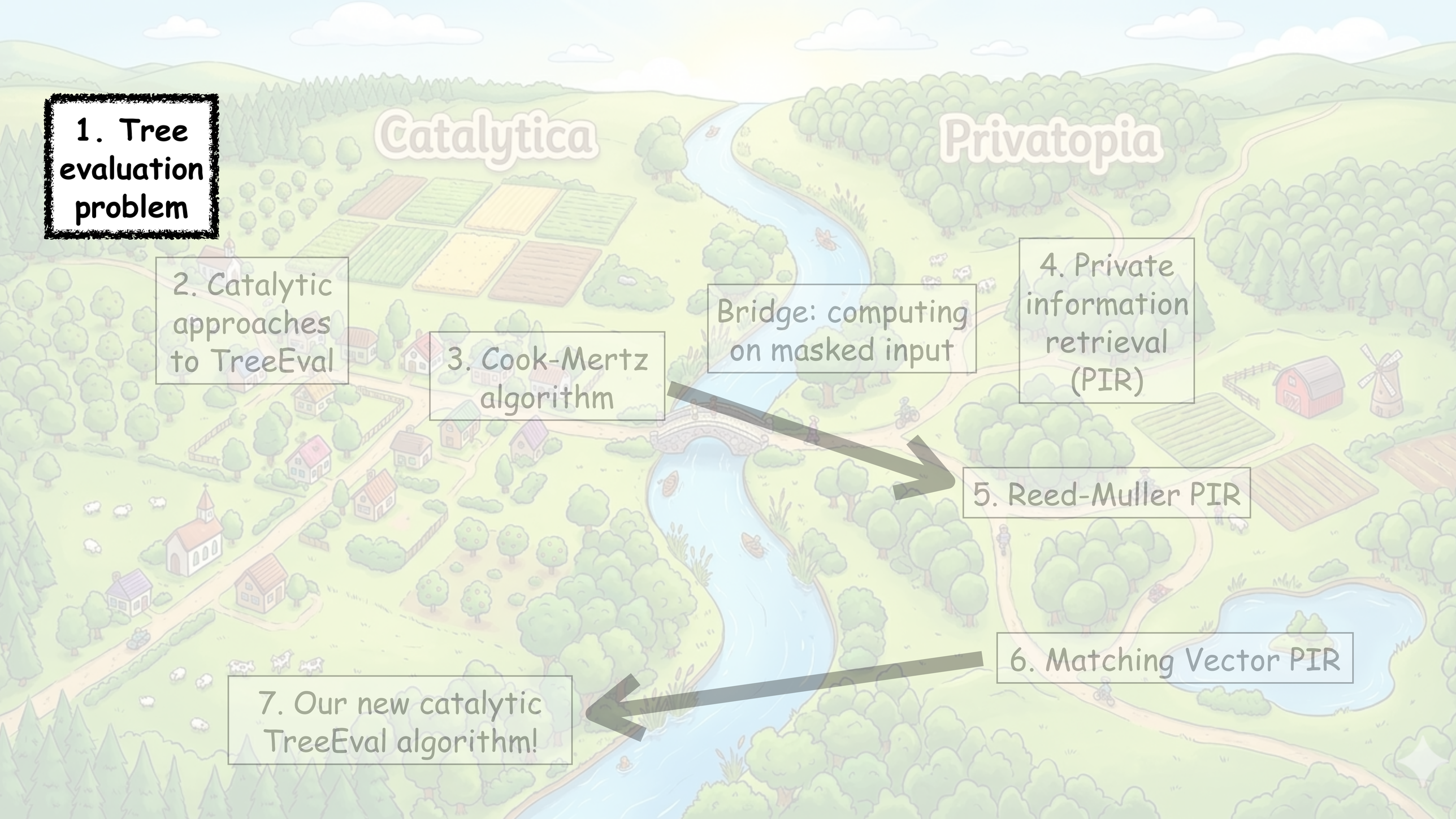

1. Tree evaluation problem
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Catalytica
Privatopia


1. Tree evaluation problem
Catalytica
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
Privatopia
4. Private information retrieval (PIR)


1. Tree evaluation problem
Catalytica
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
Privatopia
4. Private information retrieval (PIR)
5. Reed-Muller PIR

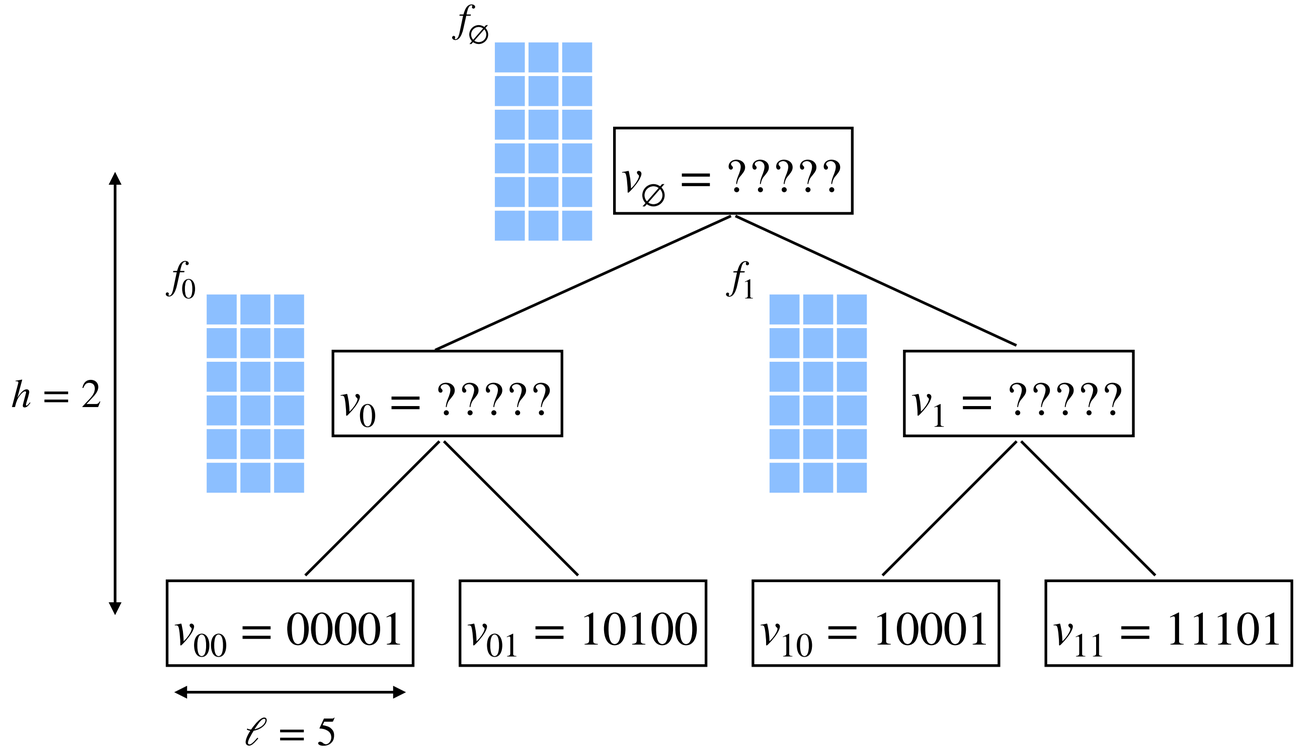

1. Tree evaluation problem
Catalytica
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
Privatopia
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR


1. Tree evaluation problem
Catalytica
Privatopia
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!


1. Tree evaluation problem
Catalytica
Privatopia
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

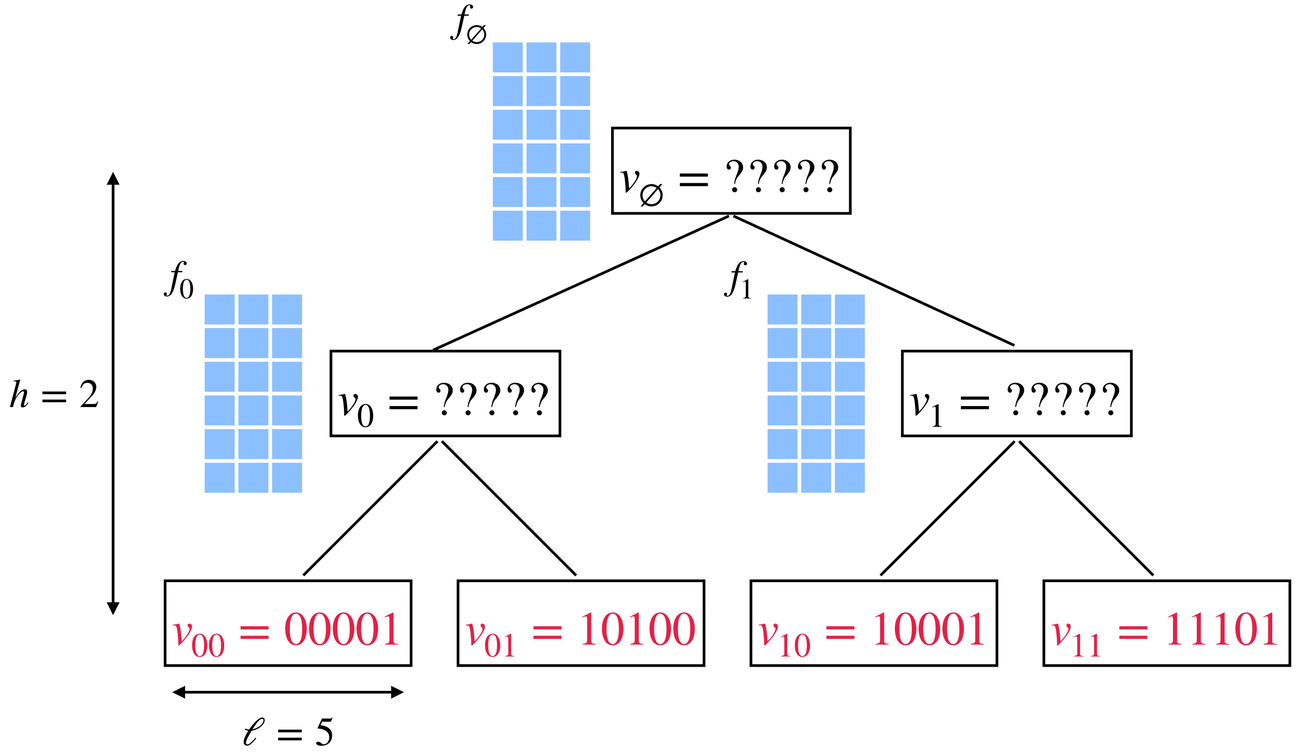
The Tree Evaluation Problem

- Complete binary tree of height h



The Tree Evaluation Problem

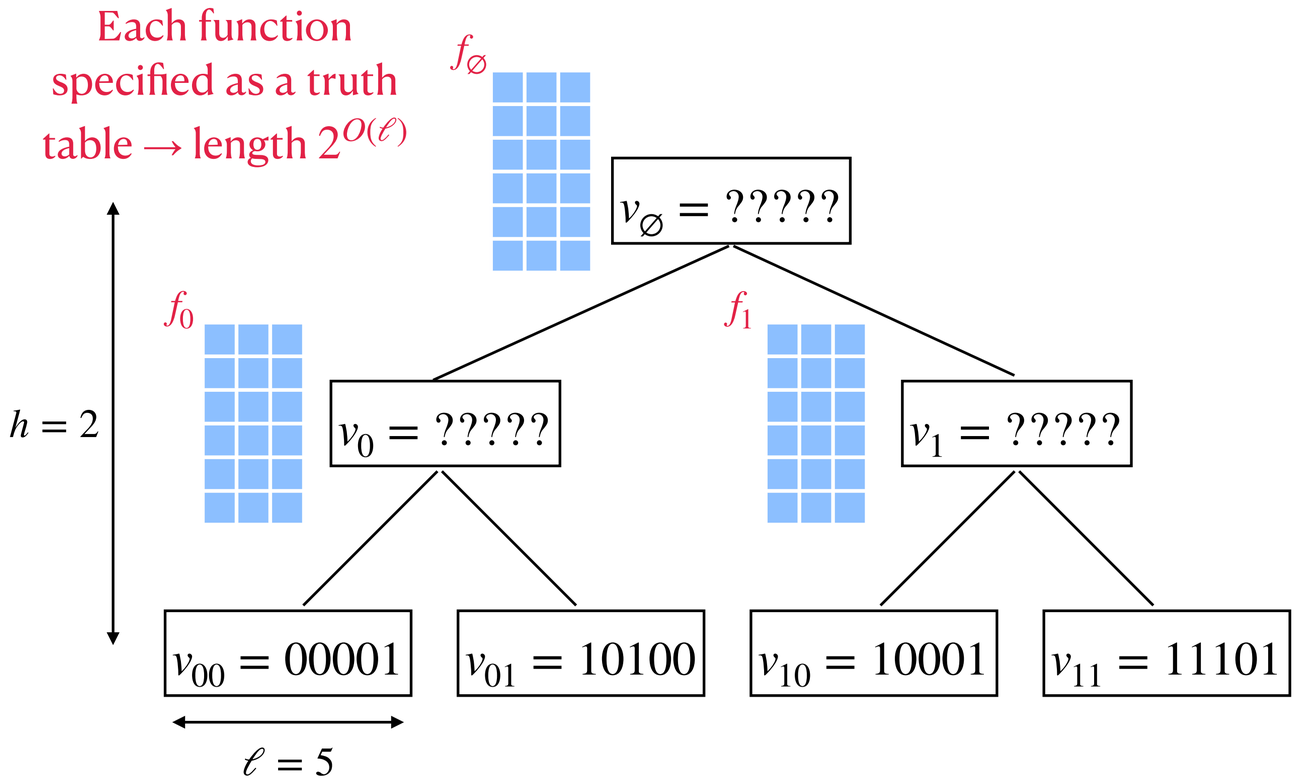
- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf



The Tree Evaluation Problem

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

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



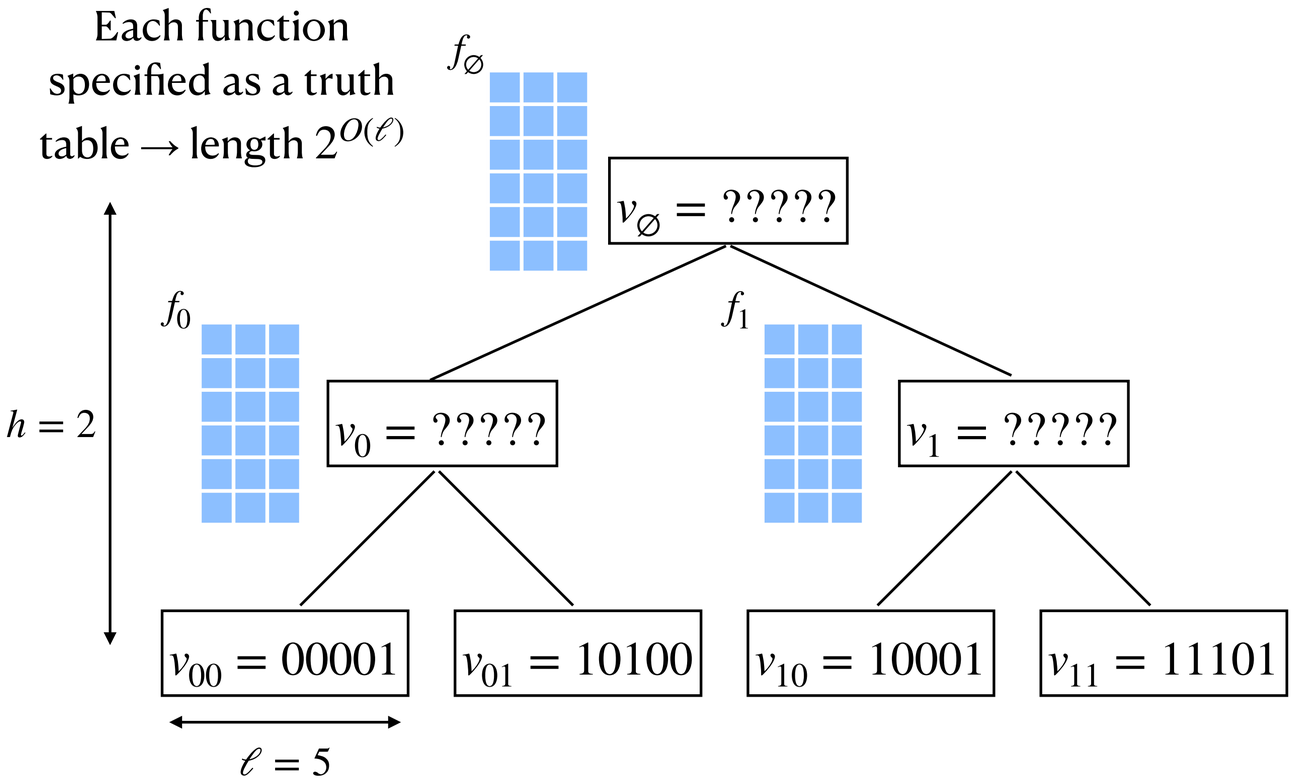
The Tree Evaluation Problem

- 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

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



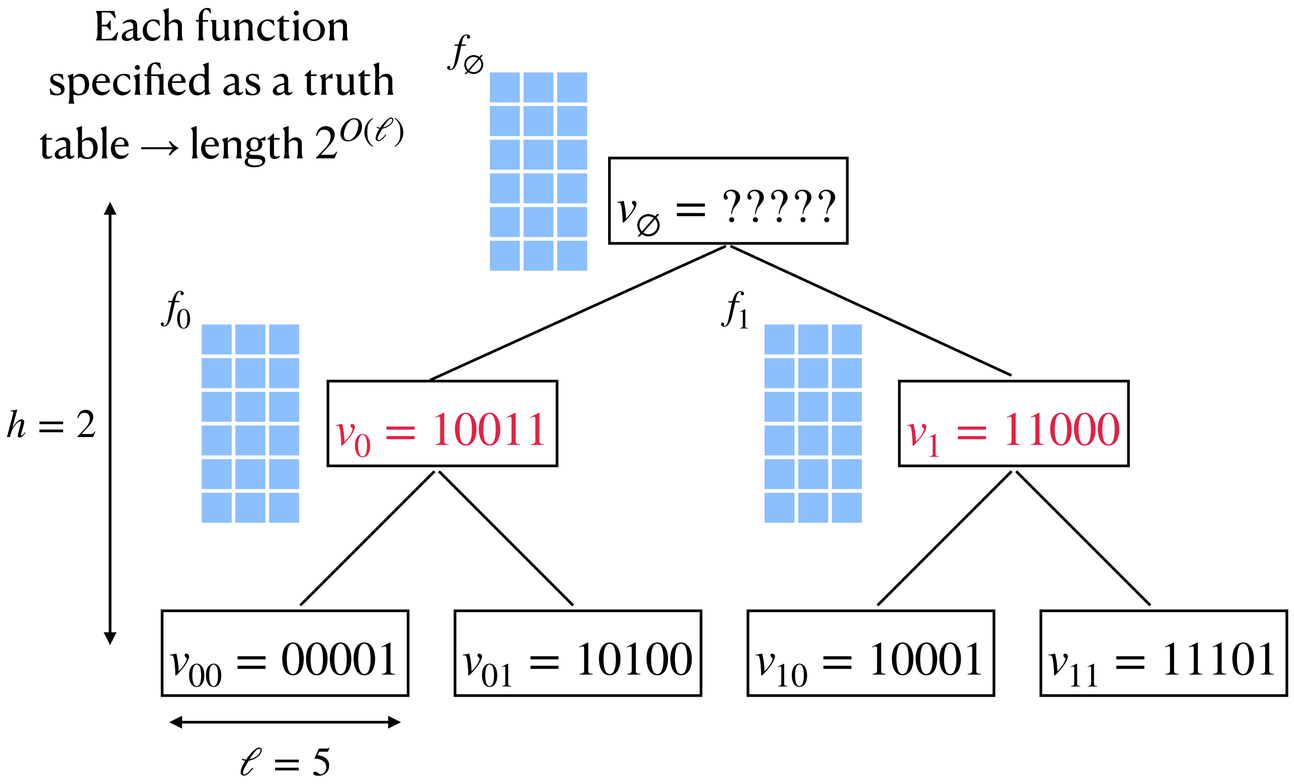
The Tree Evaluation Problem

- 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

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



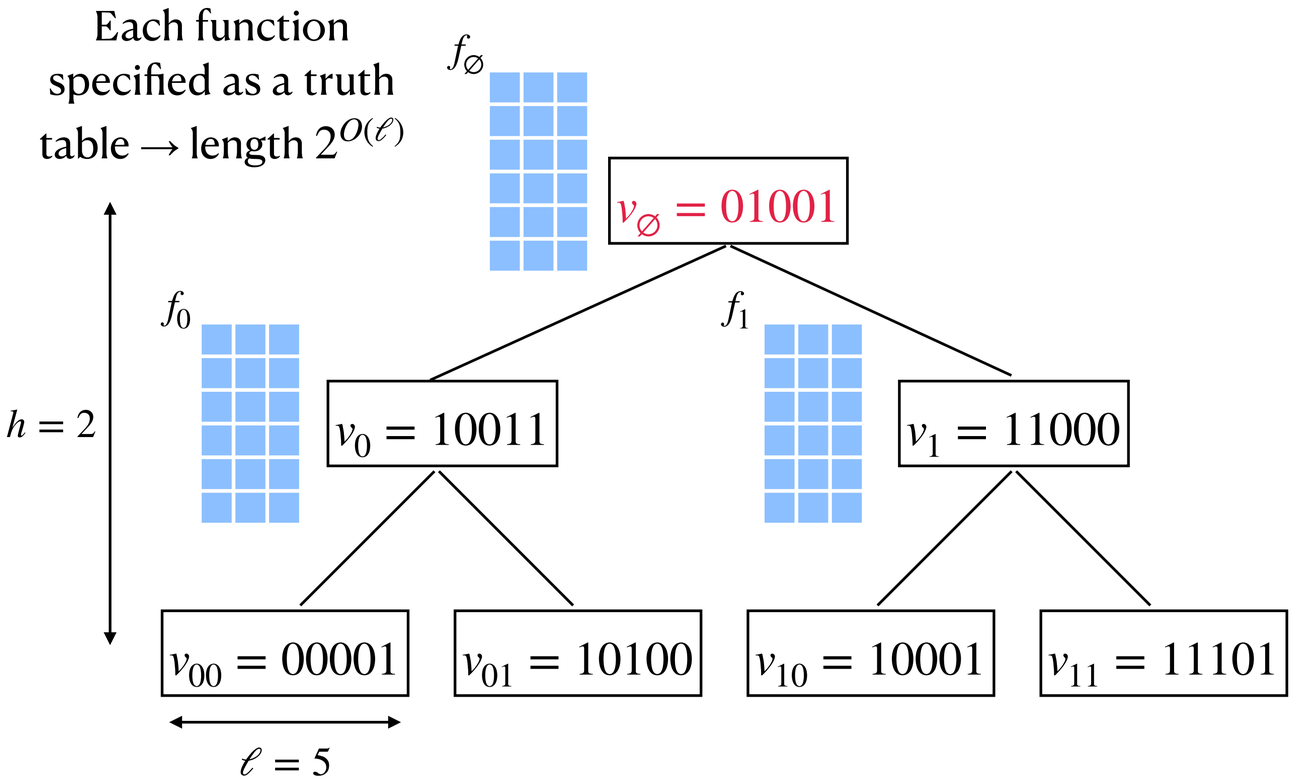
The Tree Evaluation Problem

- 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

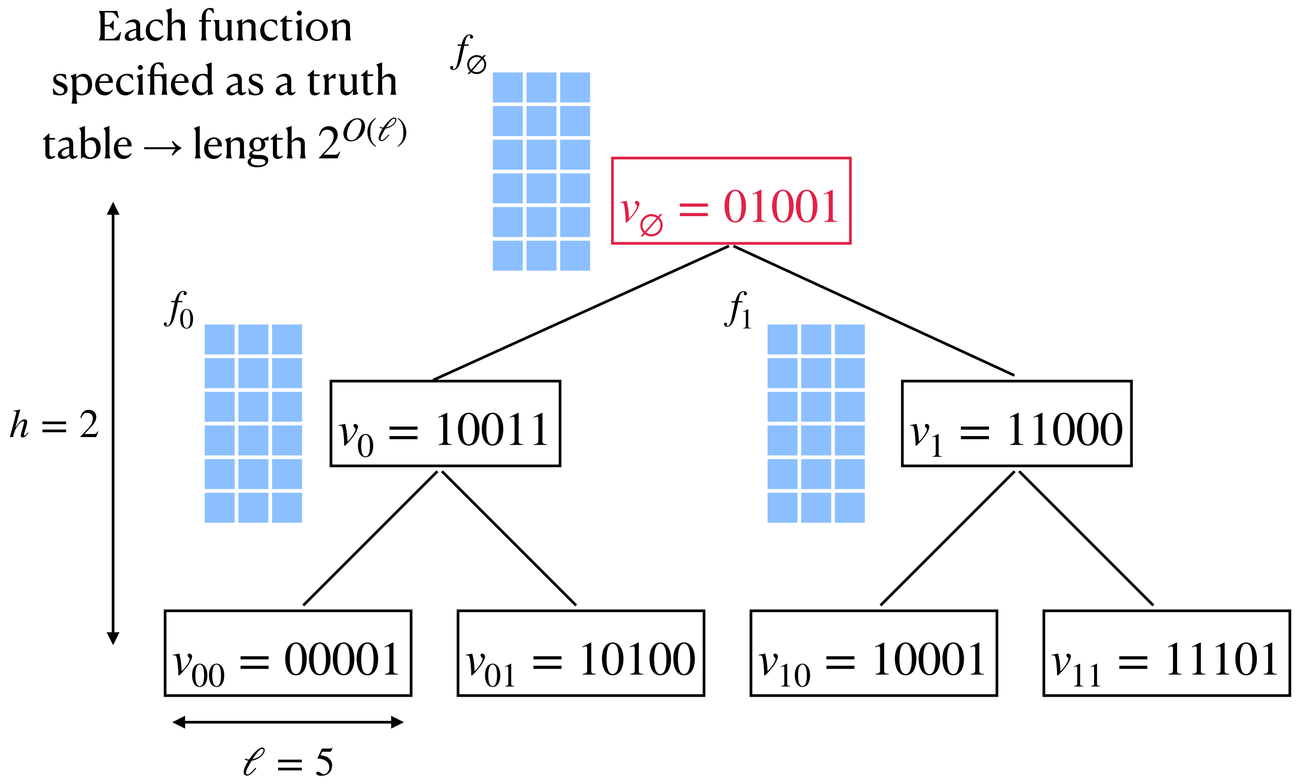
- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

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

- Goal: propagate the leaf values up the tree

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

- Output: the root value



The Tree Evaluation Problem

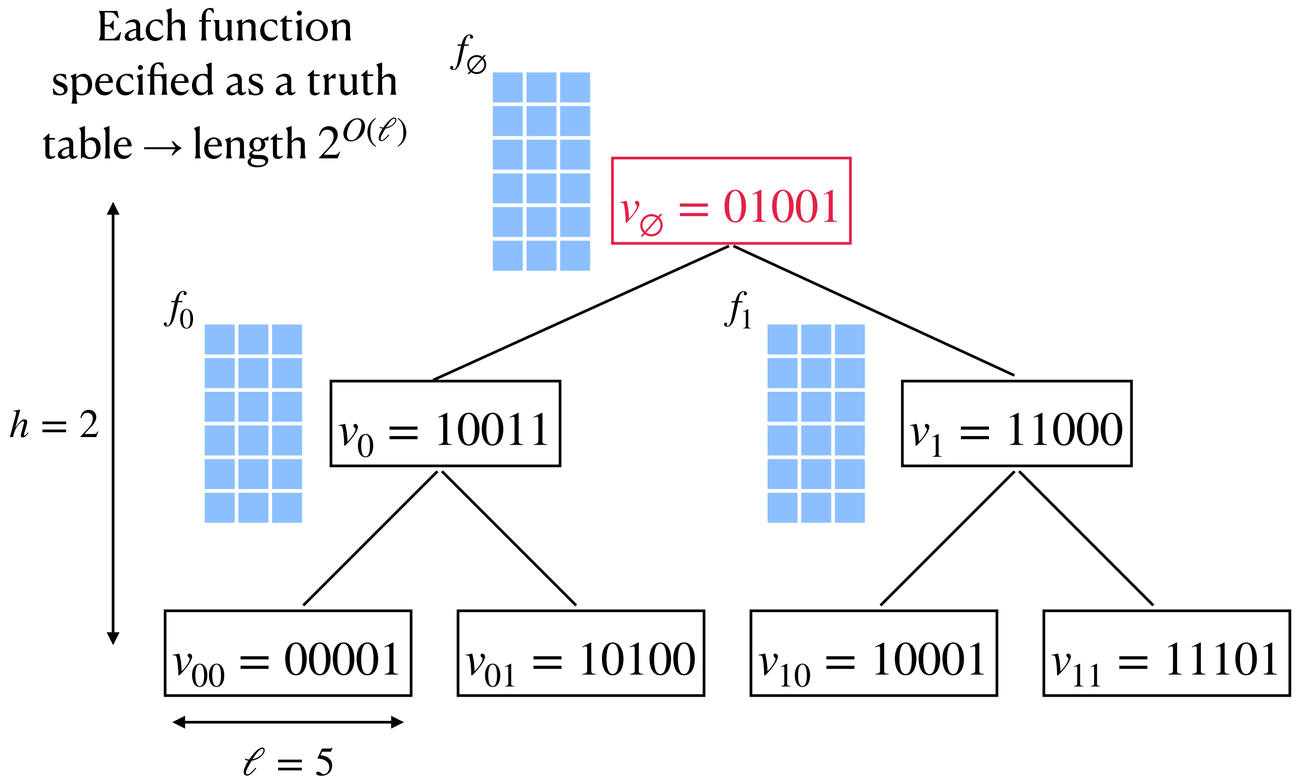
- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node

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

- Goal: propagate the leaf values up the tree

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

- Output: the root value



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

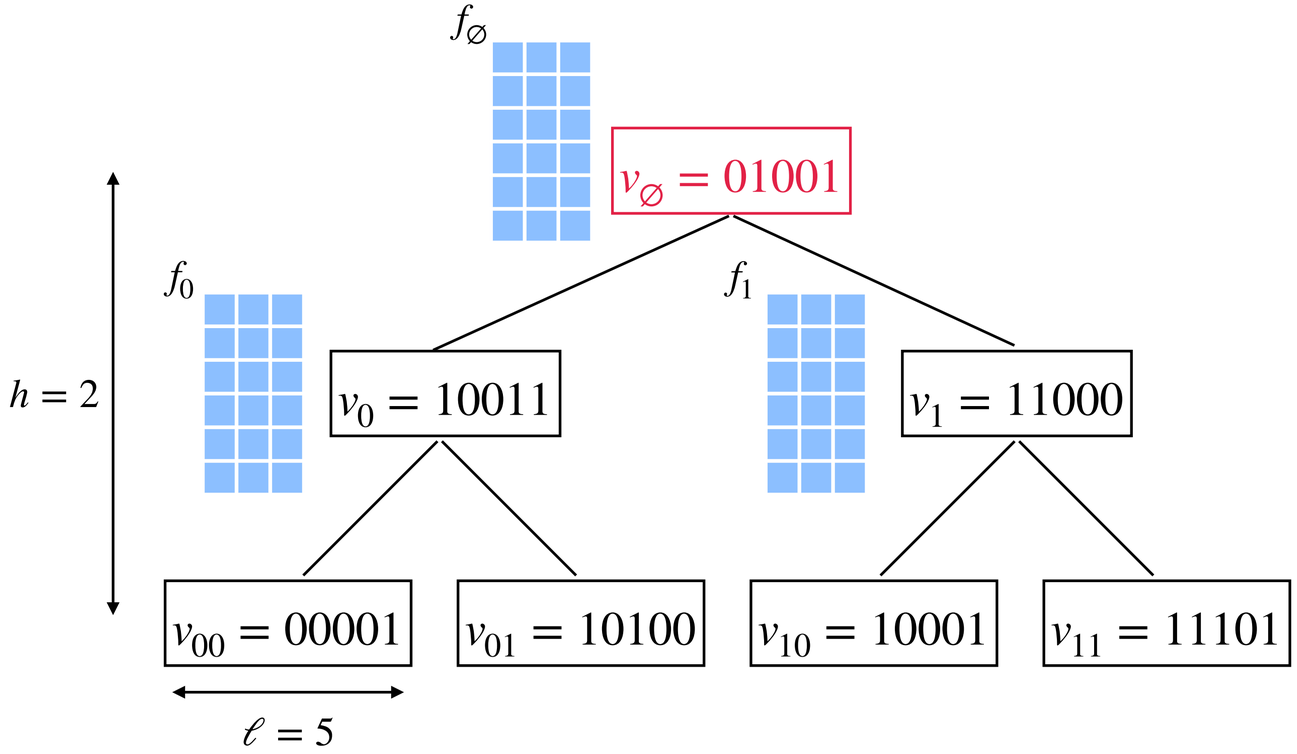
The Tree Evaluation Problem

- Complete binary tree of height h
- Inputs:
  - An ℓ-bit string at each leaf
  - Function at each internal node
- Goal: propagate the leaf values up the tree

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

- Output: the root value

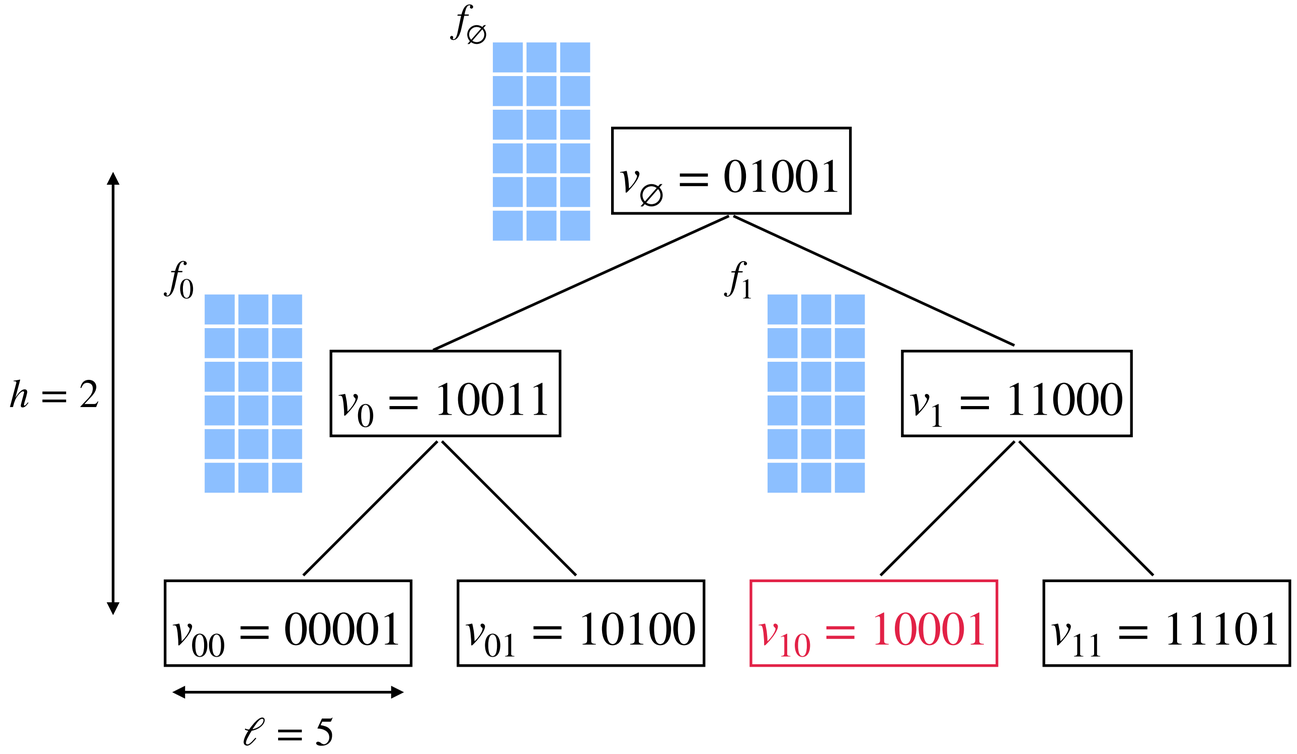
Today's talk: low-space algorithms for Tree Evaluation



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

Depth-First Recursion: $\log^2 n$ space, $\text{poly}(n)$ time

- Recursive function $\text{Ans}(u \in \{0,1\}^{\leq h})$:
  - If $\text{len}(u) = h$, output leaf value

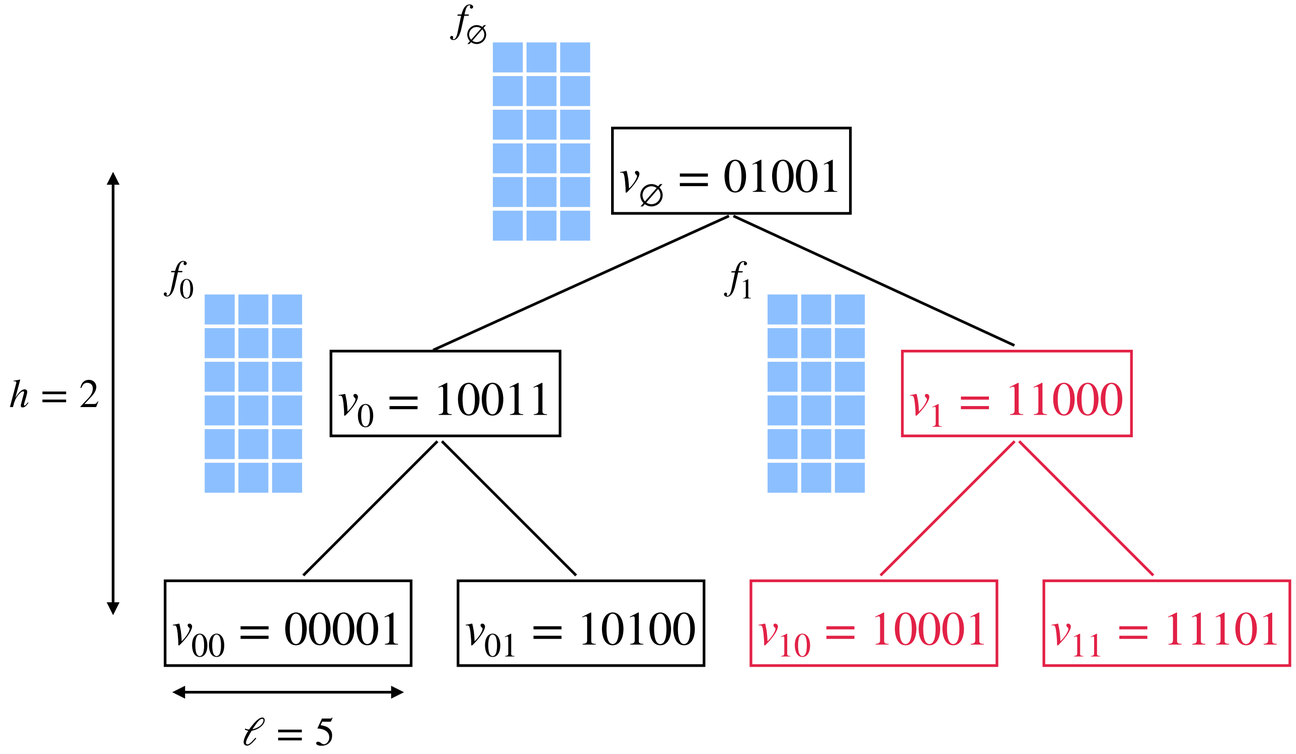


n = total input length = $\text{poly}(2^{h+\ell})$

Depth-First Recursion: $\log^2 n$ space, $\mathrm{poly}(n)$ time

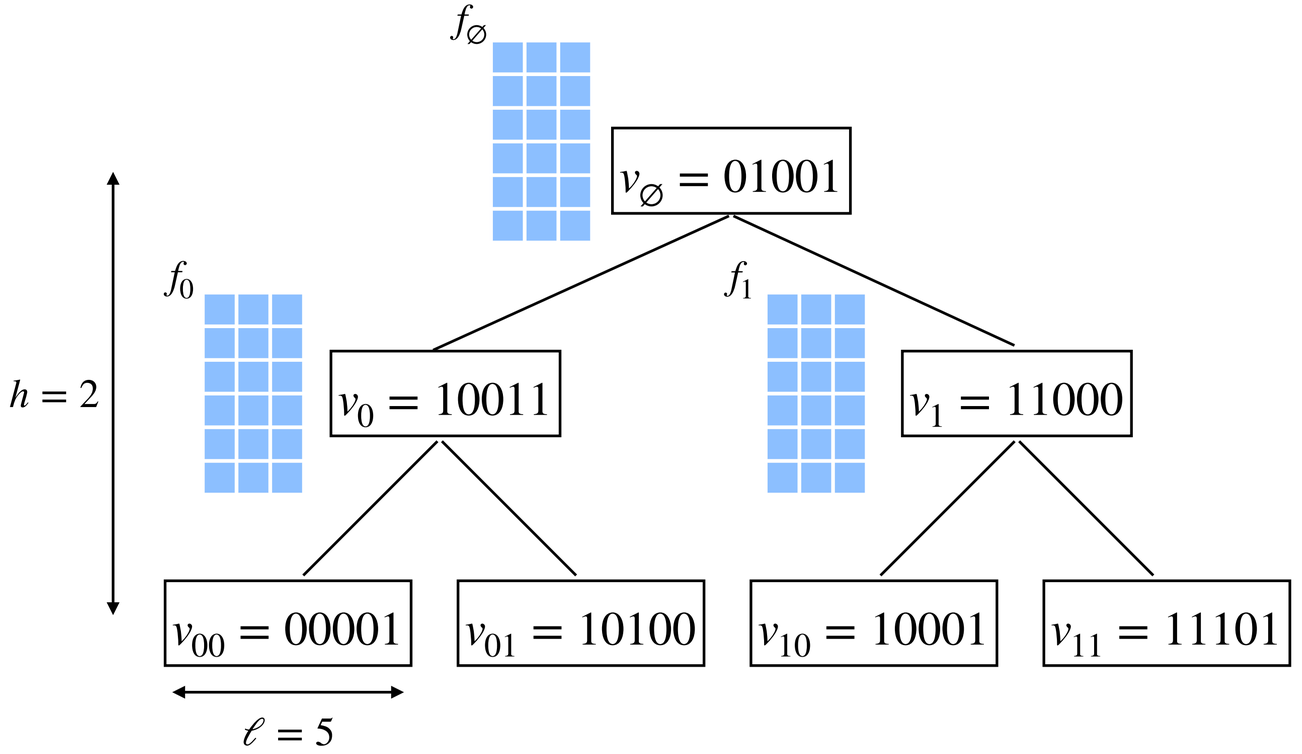
- Recursive function $\mathrm{Ans}(u \in \{0,1\}^{\leq h})$:
  - If $\mathrm{len}(u) = h$, output leaf value
  - Else, output

$$f_u(\mathrm{Ans}(u0), \mathrm{Ans}(u1))$$



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

Depth-First Recursion: $\log^2 n$ **space,** $\mathrm{poly}(n)$ **time**



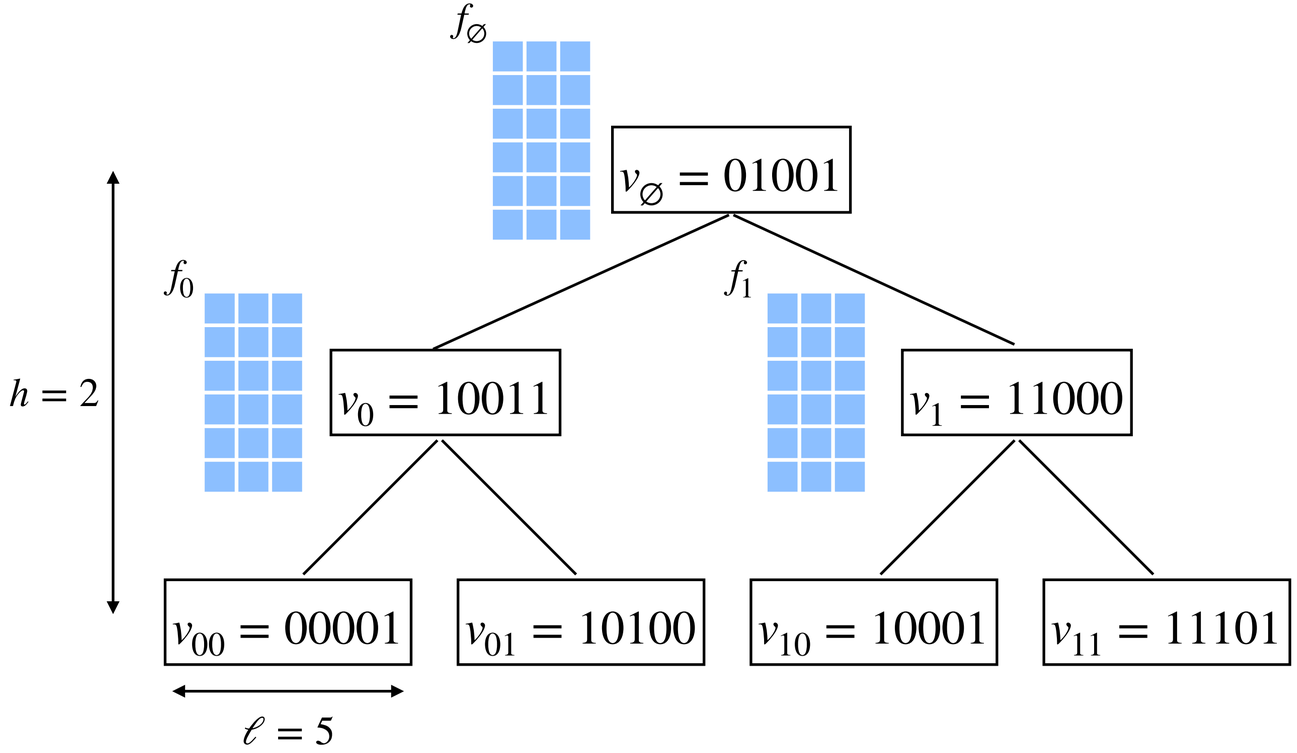
n = total input length = $\mathrm{poly}(2^{h+\ell})$

Depth-First Recursion: $\log^2 n$ space, $\mathrm{poly}(n)$ time

- Recursive function $\mathsf{Ans}(u \in \{0,1\}^{\leq h})$:
  - If $\mathsf{len}(u) = h$, output leaf value
  - Else, output

$$f_u(\mathsf{Ans}(u0), \mathsf{Ans}(u1))$$

- Algorithm: output $\mathsf{Ans}(\varnothing)$



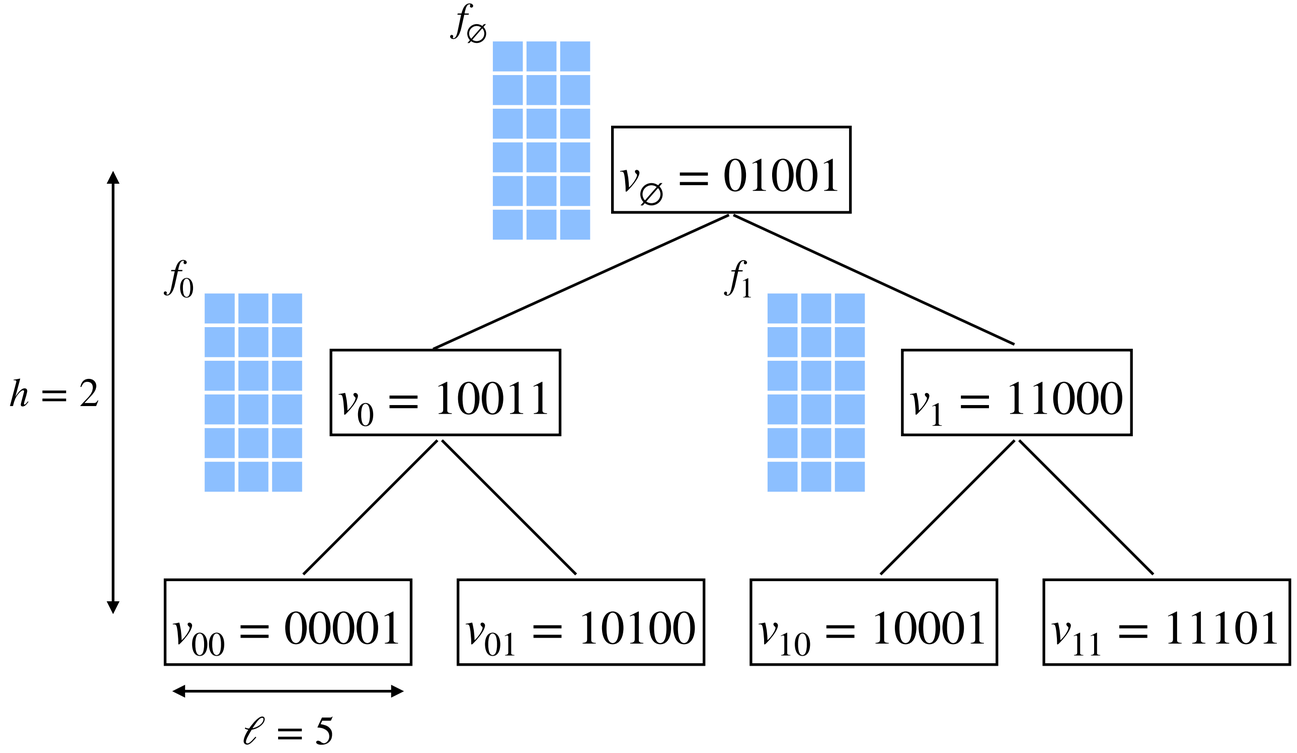
n = total input length = $\mathrm{poly}(2^{h+\ell})$

Depth-First Recursion: $\log^2 n$ space, $\mathrm{poly}(n)$ time

- Recursive function $\mathsf{Ans}(u \in \{0,1\}^{\leq h})$:
  - If $\mathsf{len}(u) = h$, output leaf value
  - Else, output

$$f_u(\mathsf{Ans}(u0), \mathsf{Ans}(u1))$$

- Algorithm: output $\mathsf{Ans}(\varnothing)$
- Runtime: $\mathrm{poly}(2^h, 2^\ell) = \mathrm{poly}(n)$



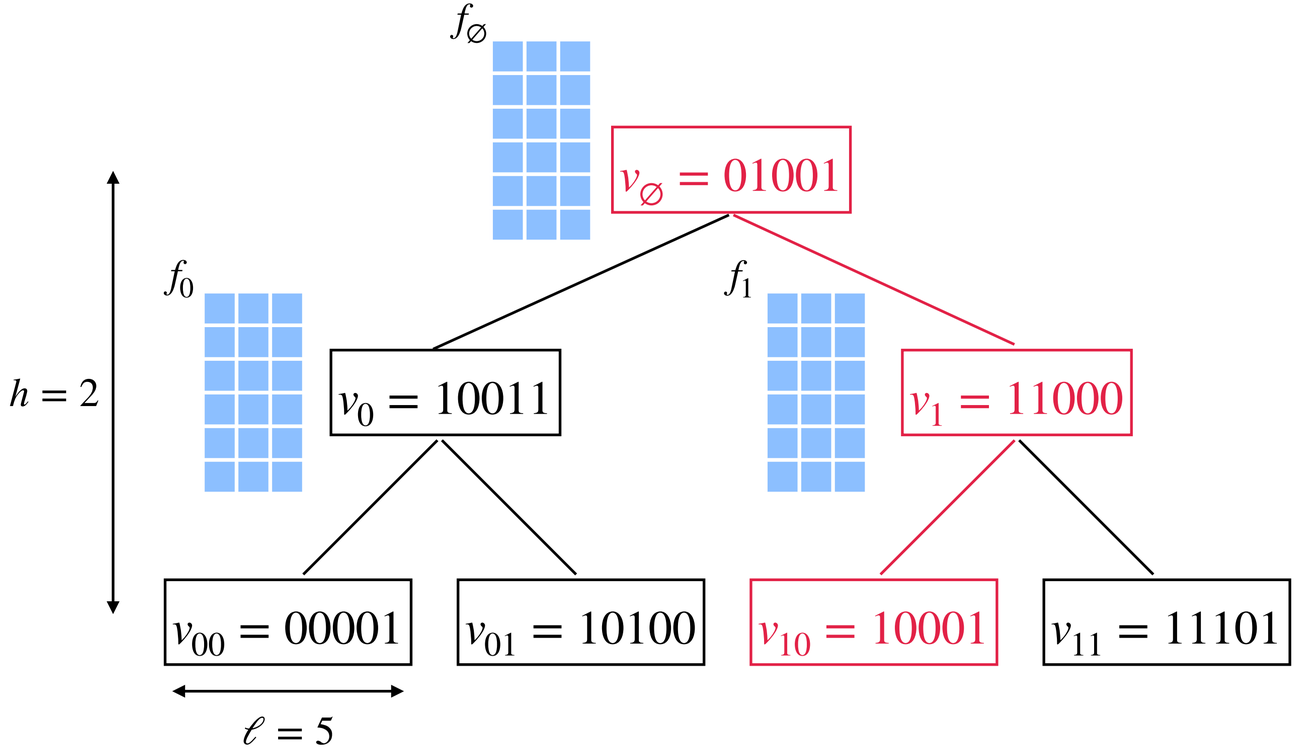
n = total input length = $\mathrm{poly}(2^{h+\ell})$

Depth-First Recursion: $\log^2 n$ space, $\mathrm{poly}(n)$ time

- Recursive function $\mathrm{Ans}(u \in \{0,1\}^{\leq h})$:
  - If $\mathrm{len}(u) = h$, output leaf value
  - Else, output

$$f_u(\mathrm{Ans}(u0), \mathrm{Ans}(u1))$$

- Algorithm: output $\mathrm{Ans}(\varnothing)$
- Runtime: $\mathrm{poly}(2^h, 2^\ell) = \mathrm{poly}(n)$
- Space: store a path of values from root to leaf $\rightarrow O(h\ell) = O(\log^2 n)$



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

Can We Do $O(\log n)$ Space?

And applications to time-space tradeoffs

Can We Do $O(\log n)$ Space?

And applications to time-space tradeoffs

- World 1: yes, there is a more clever algorithm that uses $O(\log n)$ space

Can We Do $O(\log n)$ Space?

And applications to time-space tradeoffs

- World 1: yes, there is a more clever algorithm that uses $O(\log n)$ space
  - *[Williams, STOC 2025]* This would imply that **<u>any</u>** time T computation could be done in space $O(\sqrt{T})$

Can We Do $O(\log n)$ Space?

And applications to time-space tradeoffs

- World 1: yes, there is a more clever algorithm that uses $O(\log n)$ space
  - *[Williams, STOC 2025]* This would imply that **<u>any</u>** time T computation could be done in space $O(\sqrt{T})$
- World 2: no such algorithm exists

Can We Do $O(\log n)$ Space?

And applications to time-space tradeoffs

- World 1: yes, there is a more clever algorithm that uses $O(\log n)$ space
  - *[Williams, STOC 2025]* This would imply that **<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

Open Question:

L $\subsetneq$ P?

Can We Do $O(\log n)$ Space?

And applications to time-space tradeoffs

- World 1: yes, there is a more clever algorithm that uses $O(\log n)$ space
  - *[Williams, STOC 2025]* This would imply that **<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

Open Question:

L $\subsetneq$ P?

Can We Do $O(\log n)$ Space?

And applications to time-space tradeoffs

- World 1: yes, there is a more clever algorithm that uses $O(\log n)$ space
  - *[Williams, STOC 2025]* This would imply that **<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 $O(\log n)$

Open Question:

L $\subsetneq$ P?

Tree Evaluation Through the Years

Tree Evaluation Through the Years

- *[S. Cook et al., TOCT 2012]* Defined TreeEval, conjectured that $O(\log^2 n)$ space is optimal

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!**

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.**

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

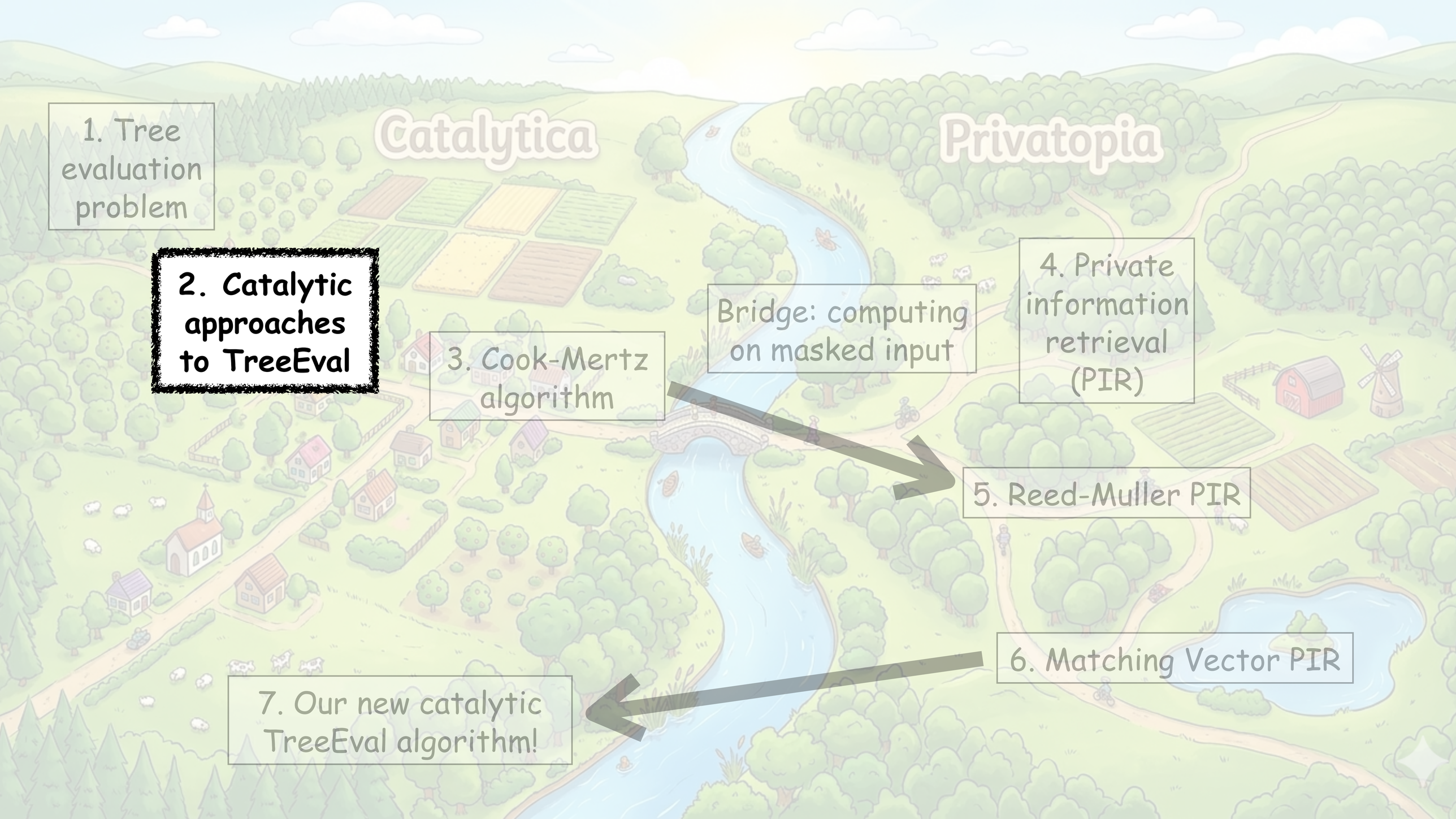
- *[S. Cook et al., TOCT 2012]* Defined TreeEval, conjectured that $O(\log^2 n)$ space is optimal
- Many works, 2012-2019: evidence for this conjecture in restricted models of computation
- *[J. Cook-Mertz, STOC 2020]* **Conjecture is false!**
  - An algorithm in space $O(\log^2 n / \log\log n)$
- *[J.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


1. Tree evaluation problem
Catalytica
Privatopia
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

Catalytic Computing = "Bad" Programming

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

`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

`10010010`

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

Catalytic Computing = "Bad" Programming

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

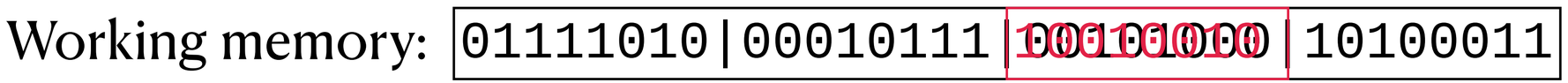
- Setting: our algorithm has computed some new information, and needs to store it somewhere
- Most algorithms ("good programming"): allocate new space for this information

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]*

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}

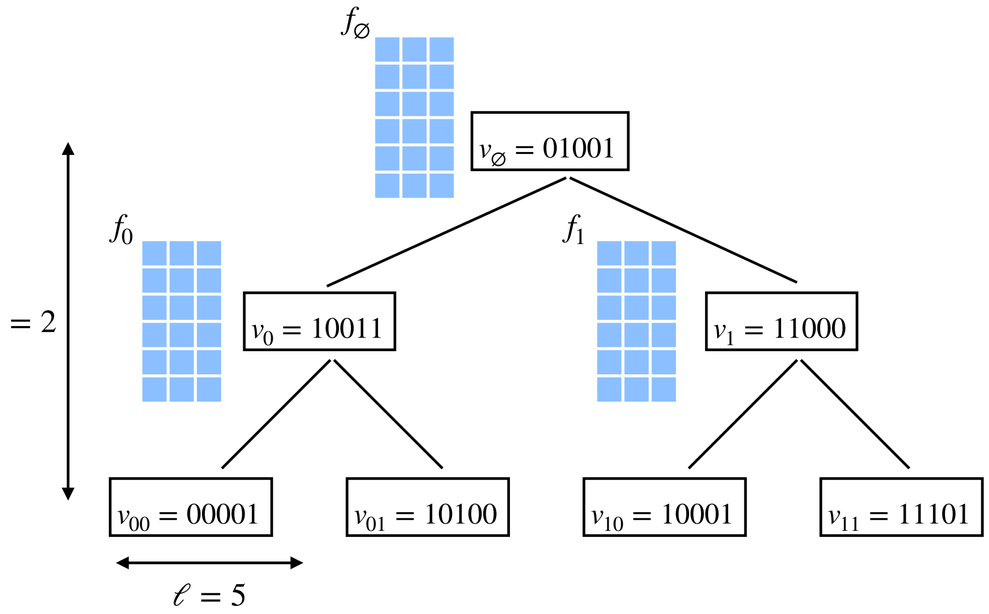
Catalytic Computing: Formal Definition

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

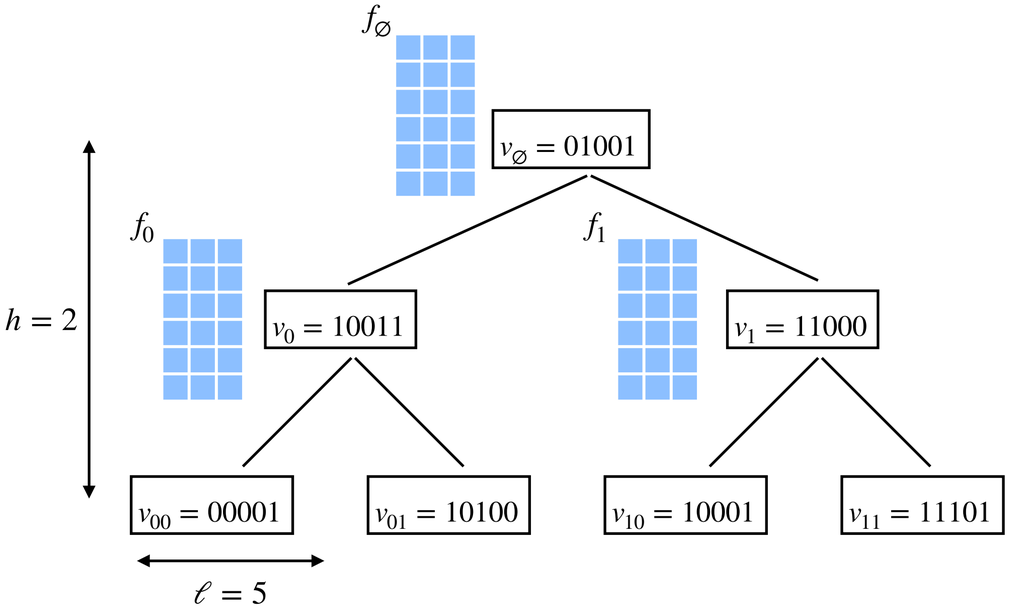
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	poly(n)



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

Catalytic Tree Evaluation Through the Years

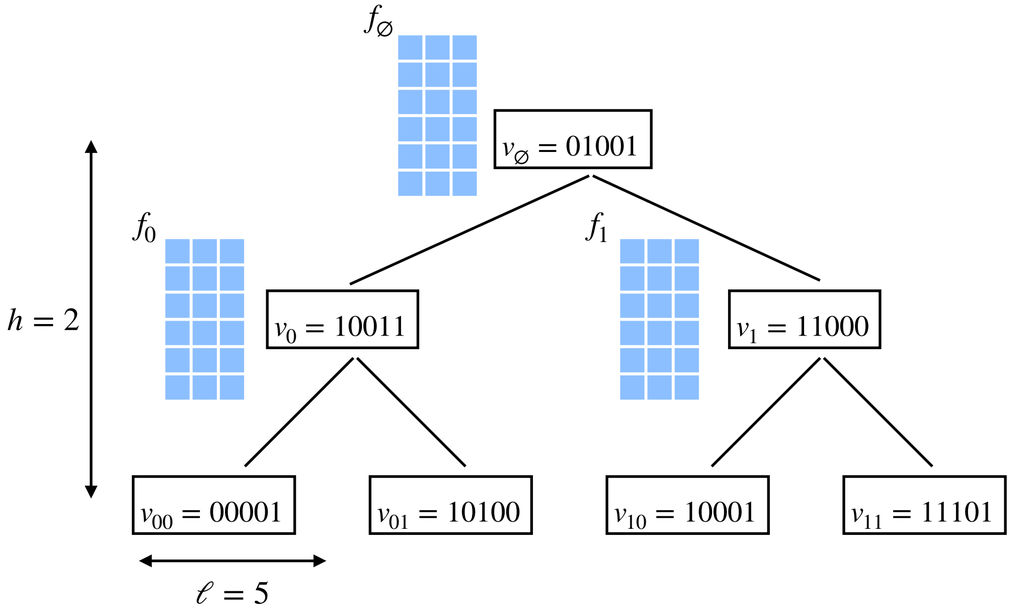
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	$\text{poly}(n)$
Buhrman et al. (STOC 2014)	$\text{poly}(n)$	$\log n$	$\text{poly}(n)$



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

Catalytic Tree Evaluation Through the Years

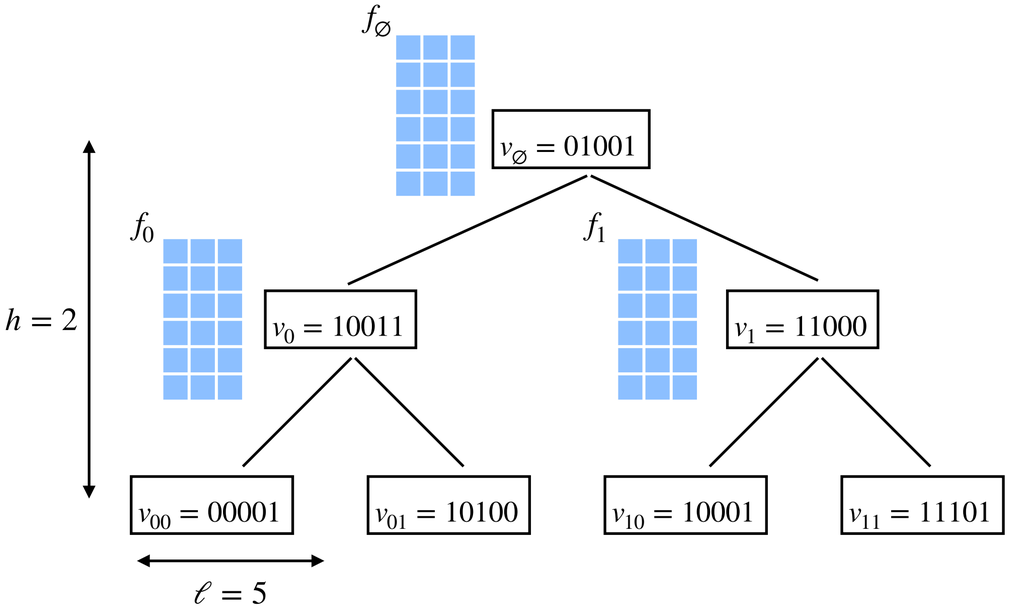
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	$\text{poly}(n)$
Buhrman et al. (STOC 2014)	$\text{poly}(n)$	$\log n$	$\text{poly}(n)$
J. Cook-Mertz (STOC 2024)		$O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$
	$\ell = O(\log n)$	$h \log \ell = O(\log n \log\log n)$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



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

Catalytic Tree Evaluation Through the Years

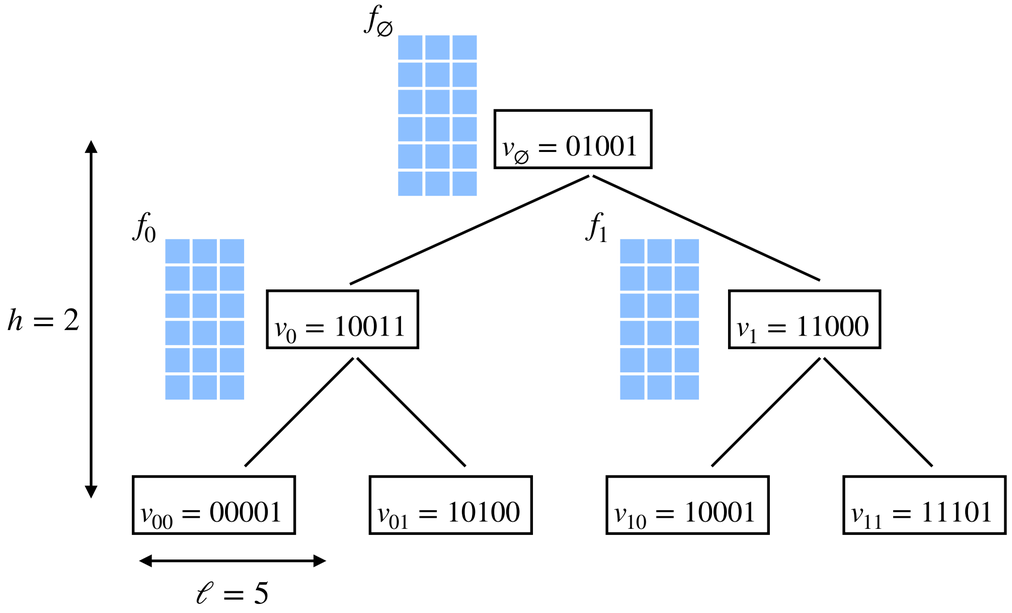
Algorithm	Catalytic space	Free space	Time
Depth-first		$h\ell = O(\log^2 n)$	$\text{poly}(n)$
Buhrman et al. (STOC 2014)	$\text{poly}(n)$	$\log n$	$\text{poly}(n)$
J. Cook-Mertz (STOC 2024)		$O(\log n \log\log n)$	$\ell^h = \text{quasipoly}(n)$
	$\ell = O(\log n)$	$h \log \ell = O(\log n \log\log n)$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

Cook-Mertz works by evaluating functions on masked inputs.

Cook-Mertz works by evaluating functions on masked inputs.

So does private information retrieval (PIR).

Cook-Mertz works by evaluating functions on masked inputs.

So does private information retrieval (PIR).

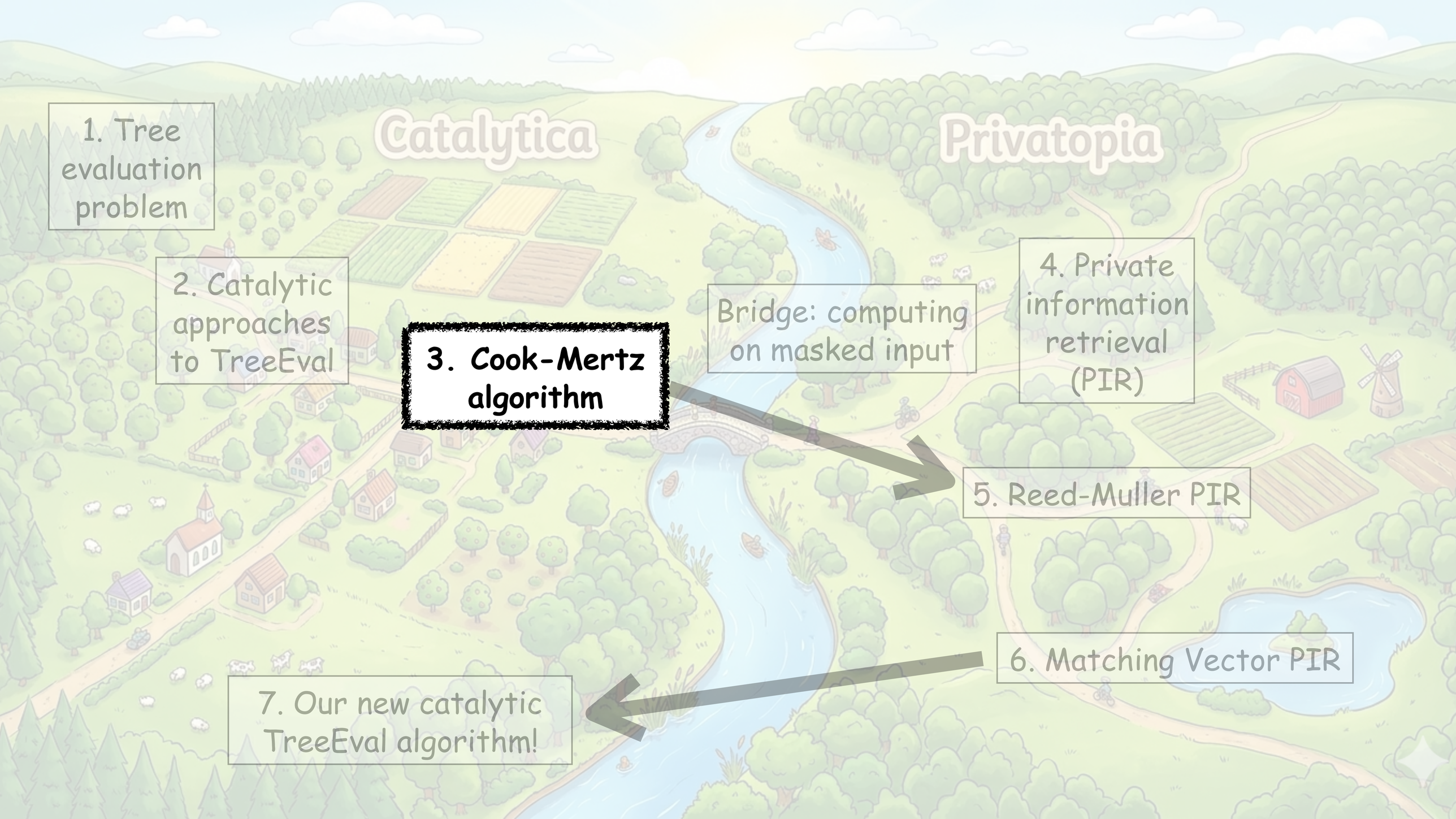
In fact, Cook-Mertz is secretly running a Reed-Muller PIR!

Cook-Mertz works by evaluating functions on masked inputs.

So does private information retrieval (PIR).

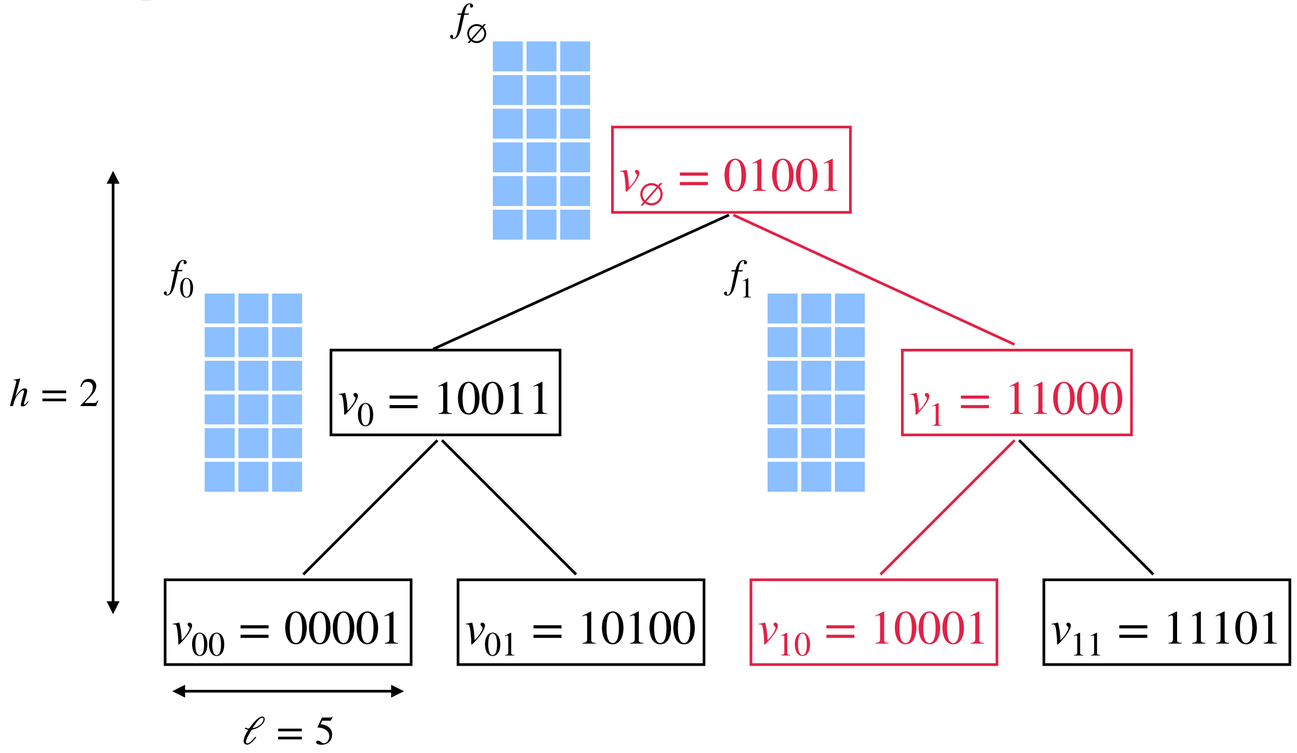
In fact, Cook-Mertz is secretly running a Reed-Muller PIR!

We leverage this connection and use matching-vector PIR to get new algorithms for TreeEval.


Catalytica
Privatopia
1. Tree evaluation problem
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

Bird's Eye View

Recursion + catalytic ideas

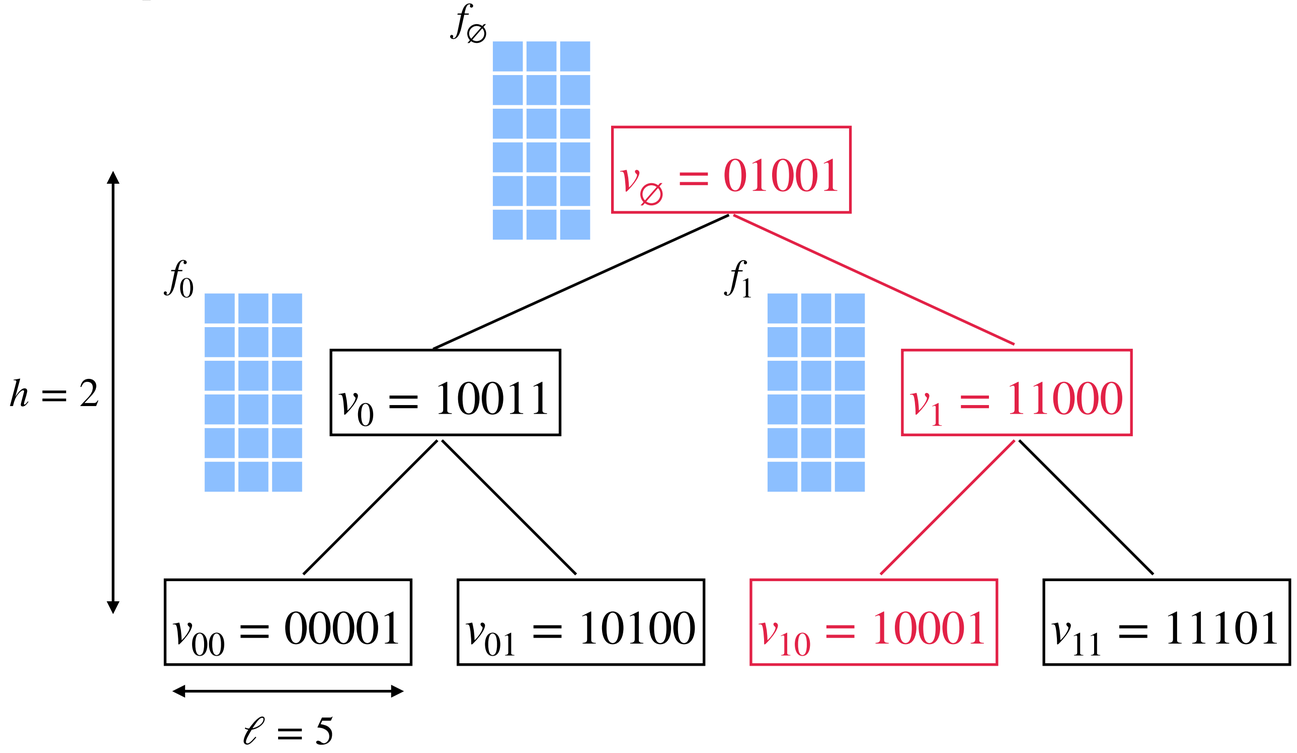


$$n = \text{total input length} = \text{poly}(2^{h+\ell})$$

Bird's Eye View

Recursion + catalytic ideas

- Memory bottleneck of naive recursive algorithm: storing all values (ℓ bits each) along a path (length h) in the tree

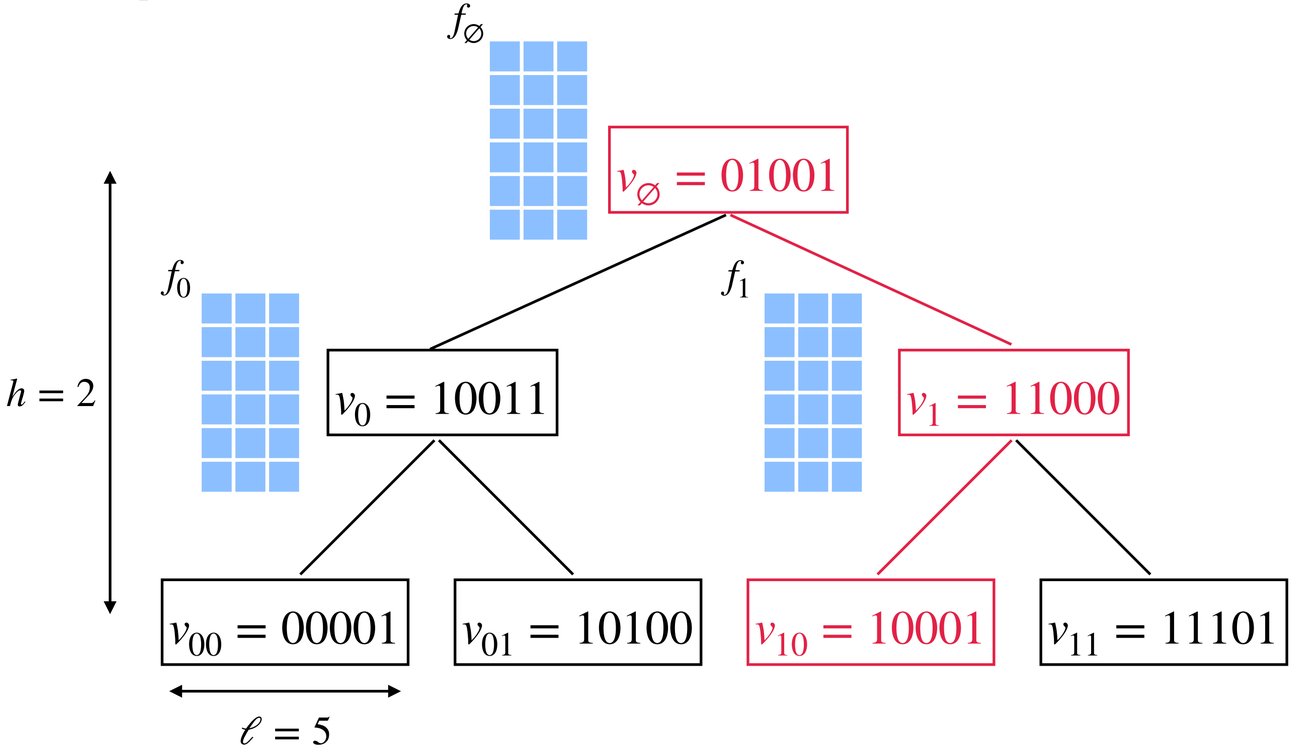


n = total input length = $\text{poly}(2^{h+\ell})$

Bird's Eye View

Recursion + catalytic ideas

- Memory bottleneck of naive recursive algorithm: storing all values (ℓ bits each) along a path (length h) in the tree
- Cook-Mertz: use catalytic tricks to "store these values on top of each other"

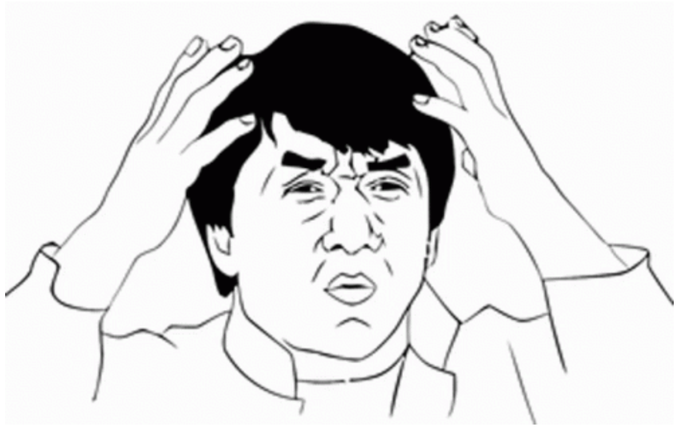


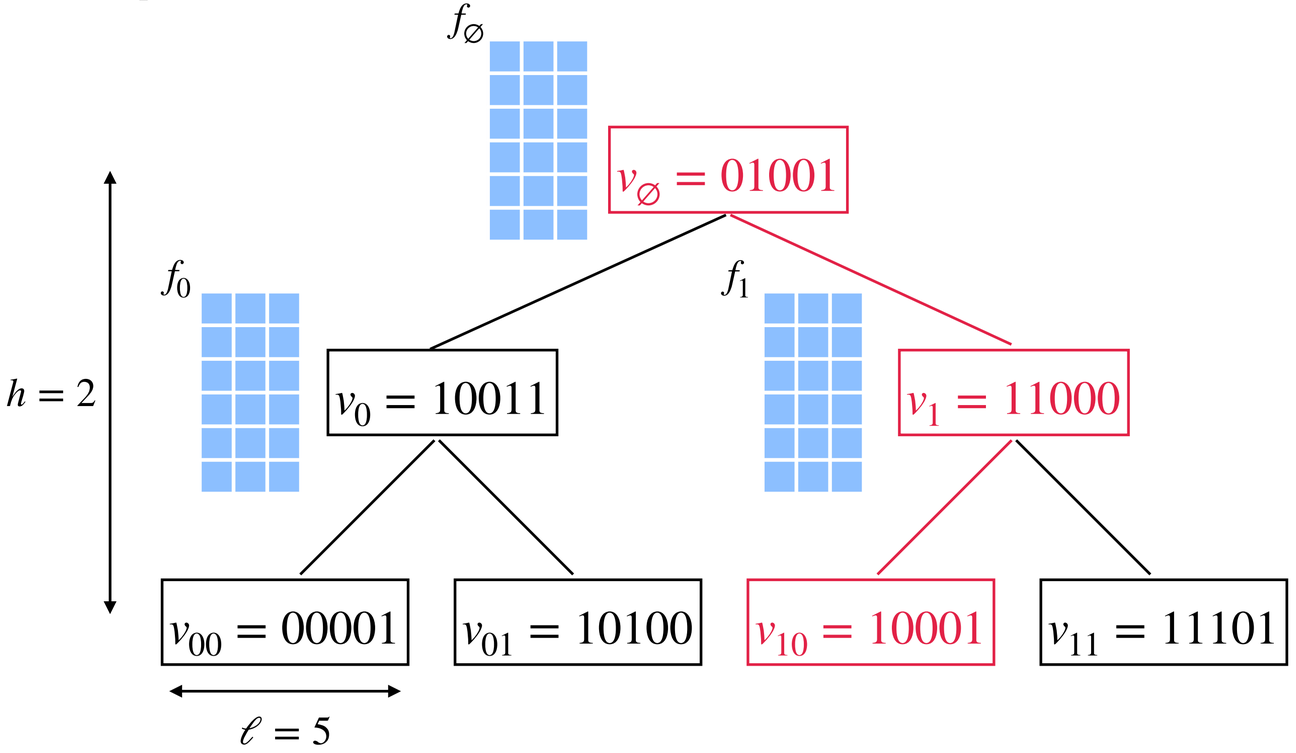
n = total input length = $\text{poly}(2^{h+\ell})$

Bird's Eye View

Recursion + catalytic ideas

- Memory bottleneck of naive recursive algorithm: storing all values (ℓ bits each) along a path (length h) in the tree
- Cook-Mertz: use catalytic tricks to "store these values on top of each other"





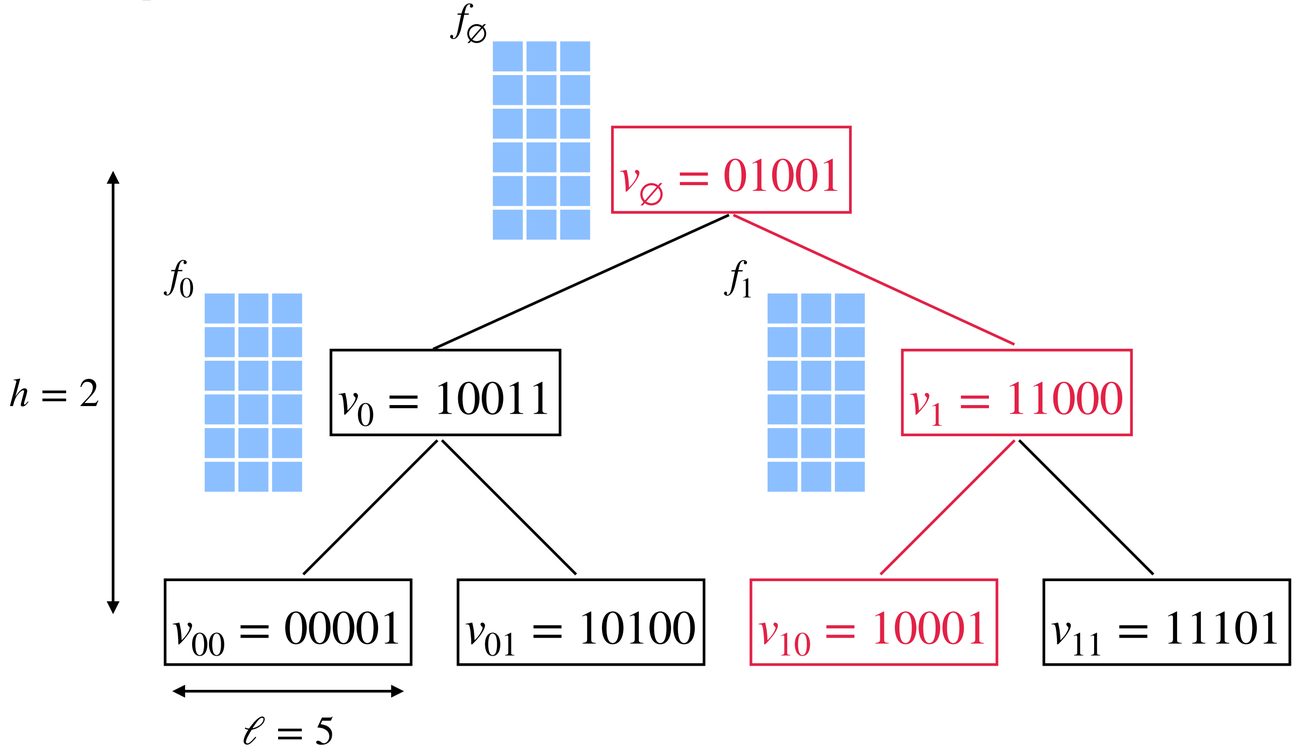
n = total input length = poly$(2^{h+\ell})$

Bird's Eye View

Recursion + catalytic ideas

- Memory bottleneck of naive recursive algorithm: storing all values (ℓ bits each) along a path (length h) in the tree
- Cook-Mertz: use catalytic tricks to "store these values on top of each other"
- **Our notion of "computing value v_u": implement an oracle $\mathcal{O}_u$ that on input $c \in \{0,1\}$ updates**

$$\tau \leftarrow \tau + (-1)^c v_u$$

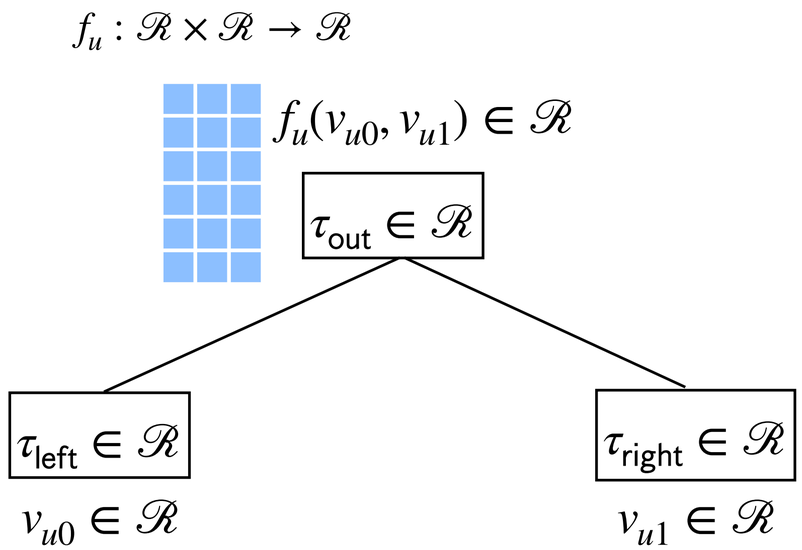


n = total input length = $\text{poly}(2^{h+\ell})$

Reduction to a Simpler Problem: The One-Level Gadget

The One-Level Gadget

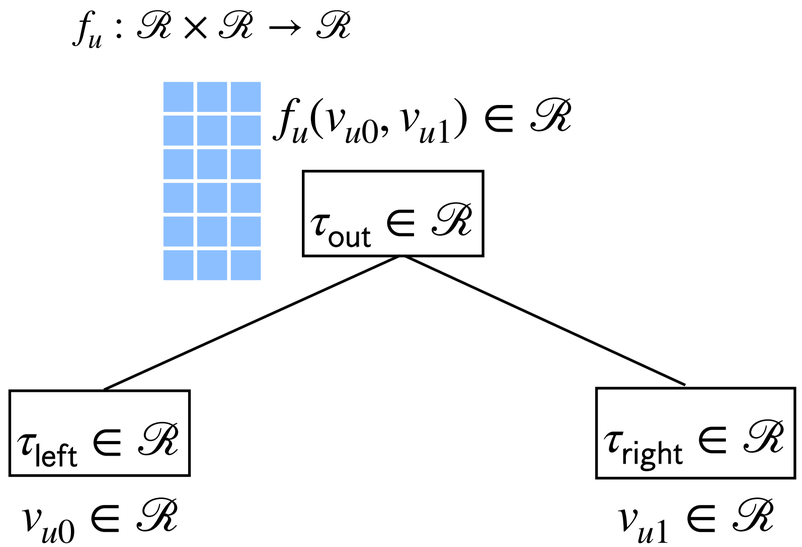
Or: boosting an oracle one level



The One-Level Gadget

Or: boosting an oracle one level

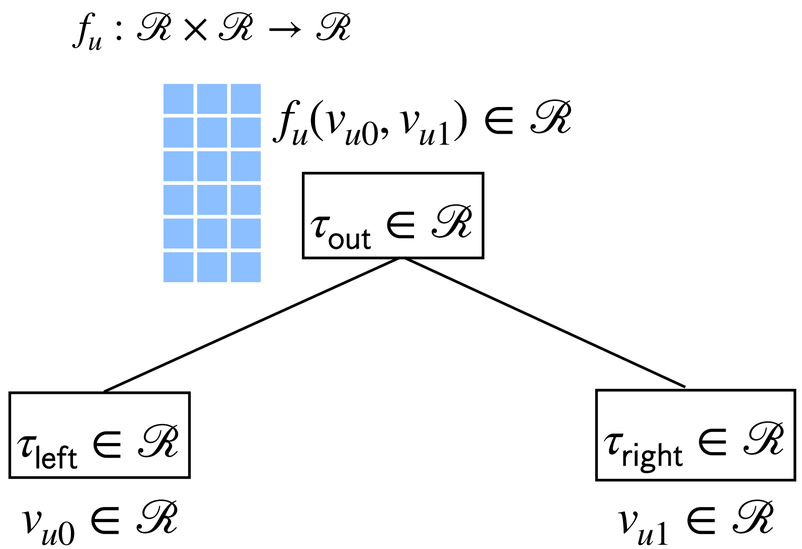
- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$



The One-Level Gadget

Or: boosting an oracle one level

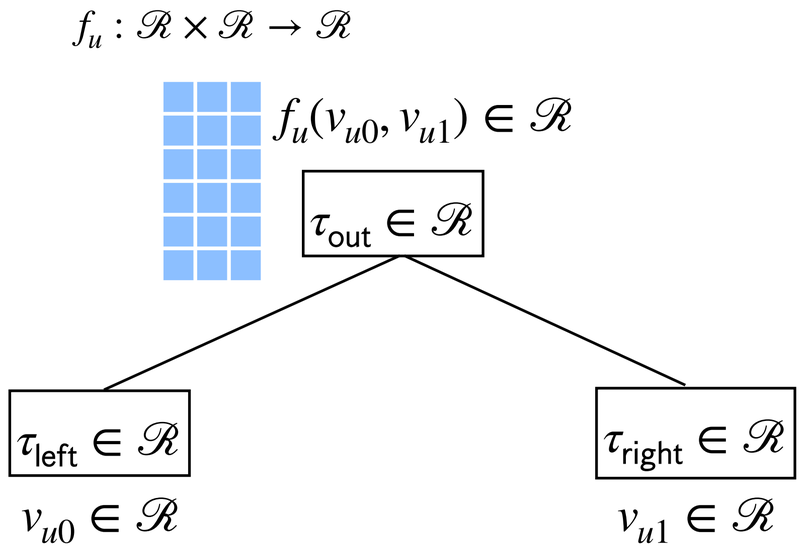
- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Control bit $c \in \{0,1\}$



The One-Level Gadget

Or: boosting an oracle one level

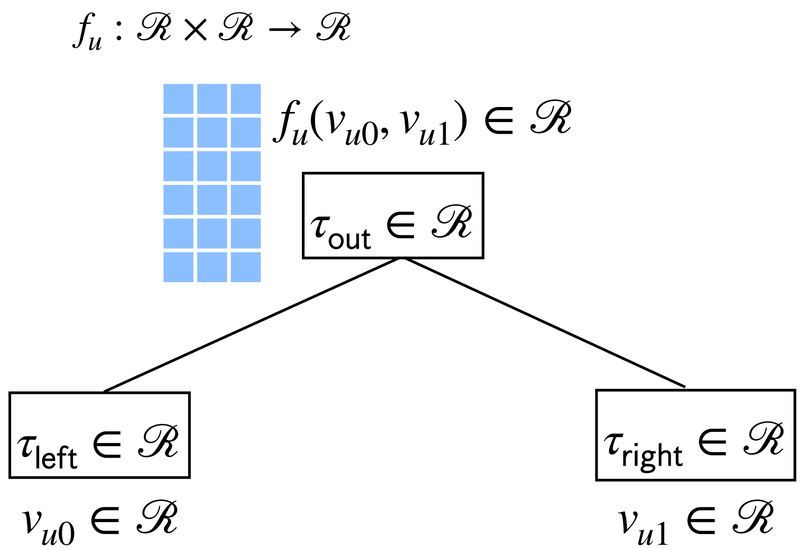
- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Control bit $c \in \{0,1\}$
  - Arbitrary catalytic registers $\tau_{\text{left}}, \tau_{\text{right}}, \tau_{\text{out}} \in \mathscr{R}$



The One-Level Gadget

Or: boosting an oracle one level

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Control bit $c \in \{0,1\}$
  - 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

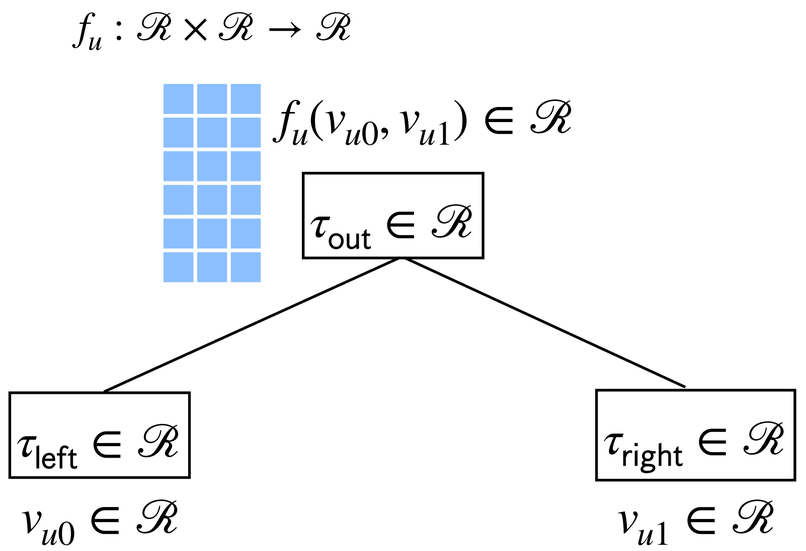
$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathscr{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

The One-Level Gadget

Or: boosting an oracle one level

- Embed $\{0,1\}^{\ell}$ in some ring $\mathscr{R}$
- Inputs:
  - Control bit $c \in \{0,1\}$
  - 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

$$(\tau_{\text{left}}, \tau_{\text{right}}, \tau_{\text{out}} + (-1)^c f_u(v_{u0}, v_{u1}))$$



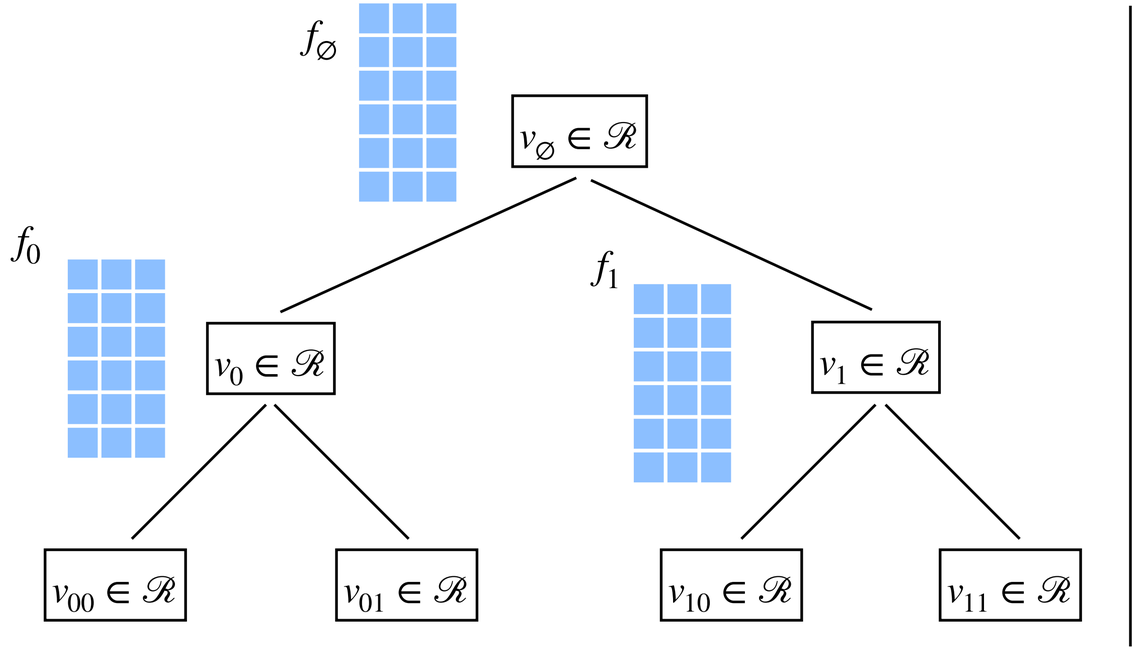
Cheatsheet

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

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:

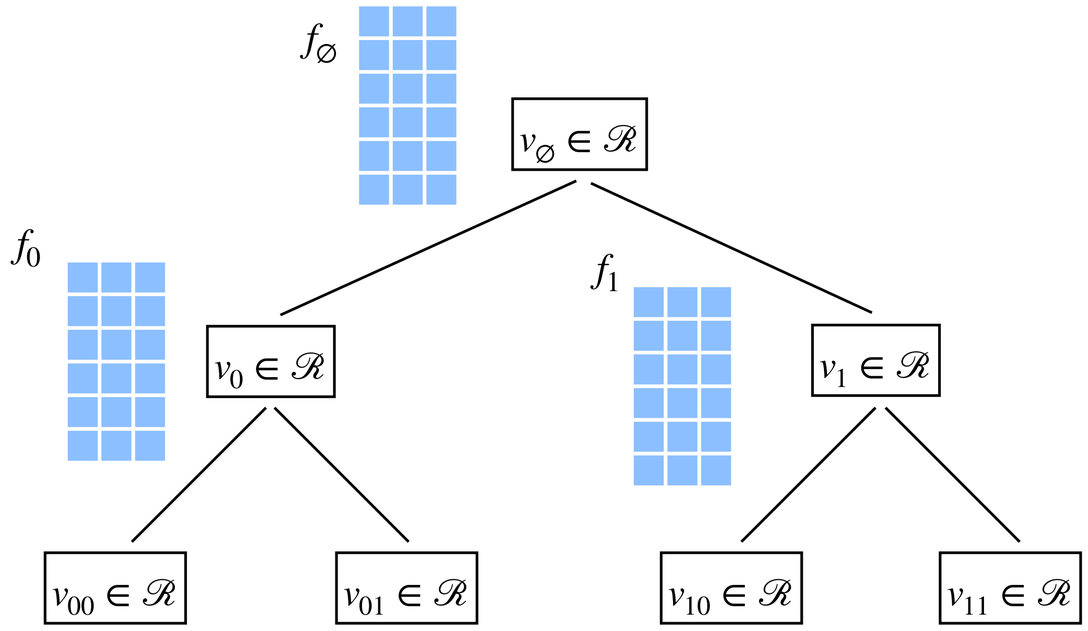
$$\mathsf{TreeEval}(h, \ell) \in \mathsf{CatSpaceTime}(\log|\mathscr{R}|, hs+\log\log|\mathscr{R}|+\ell, \mathsf{poly}(q^h, 2^{\ell}))$$



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

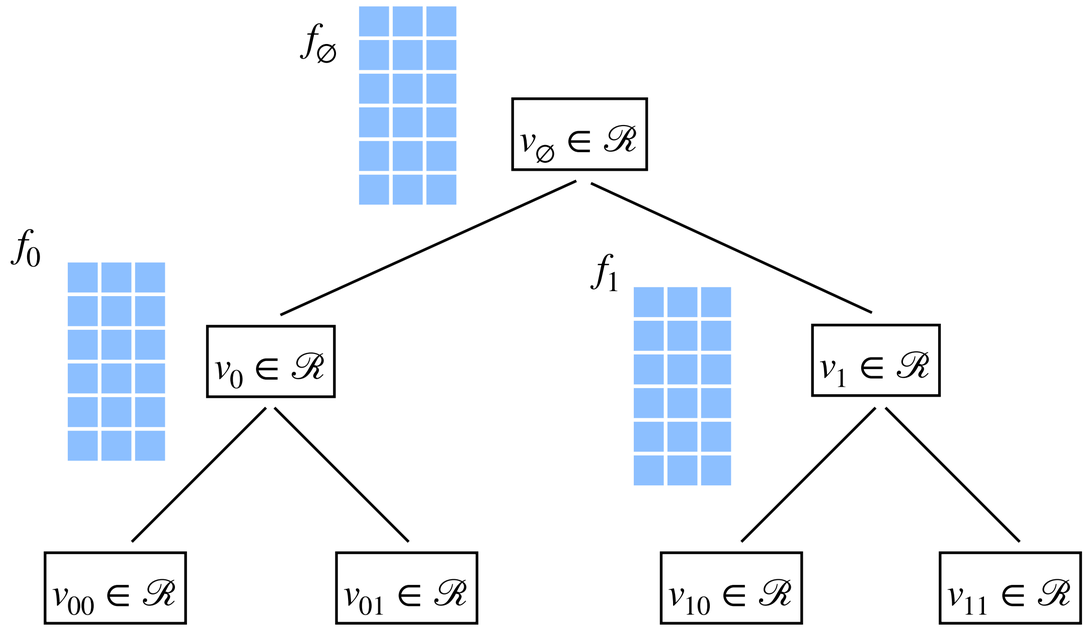
|00110101|10010100|00001110|

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

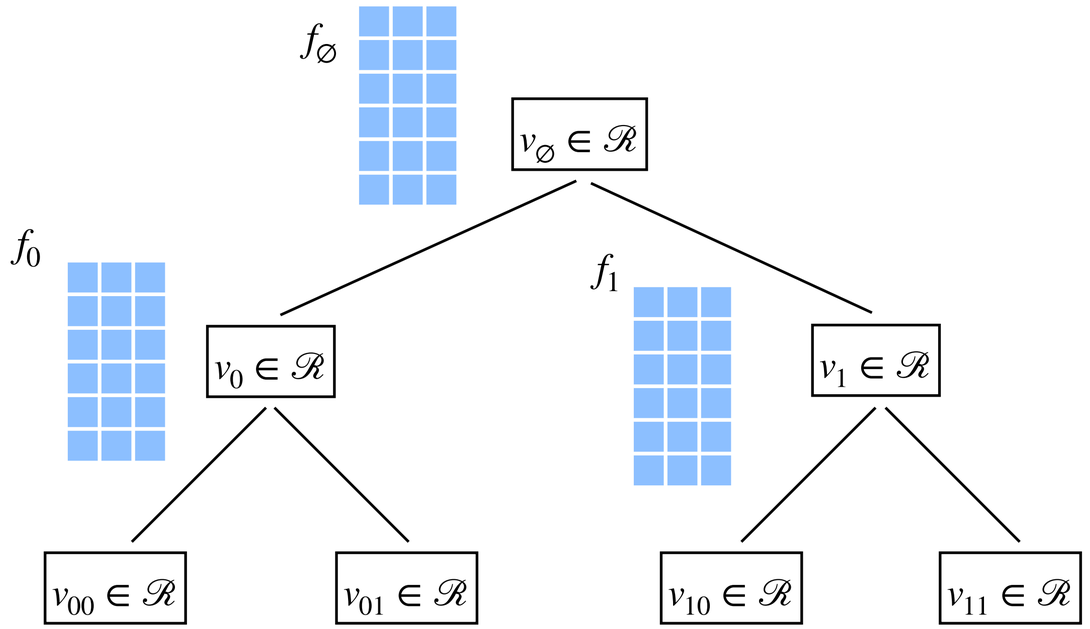
|00110101|10010100|00001110|

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

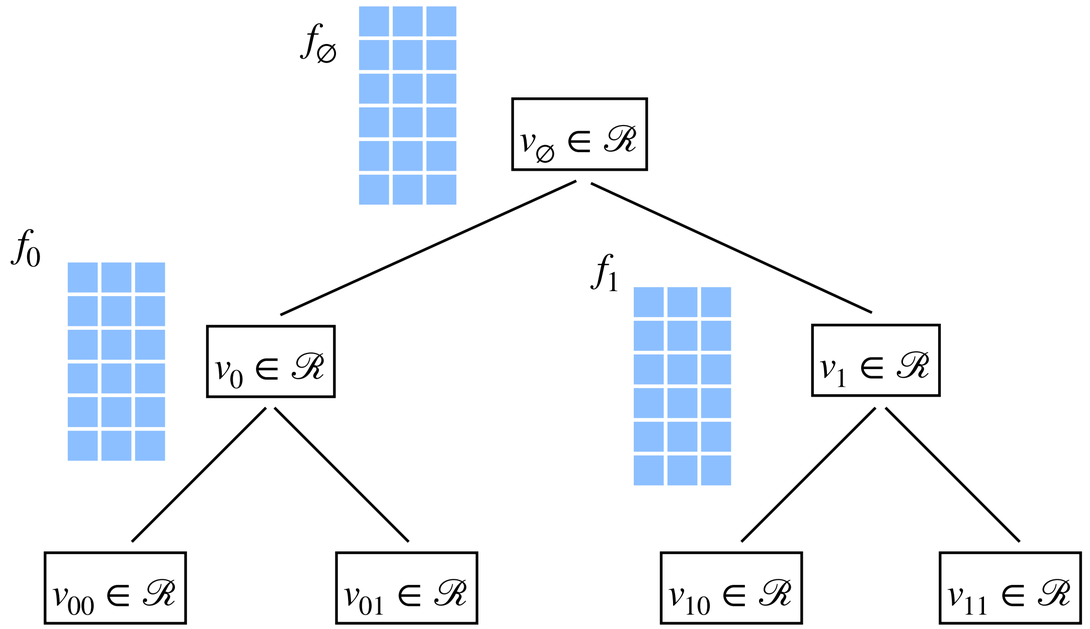
|00110101|10010100|00001110|

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

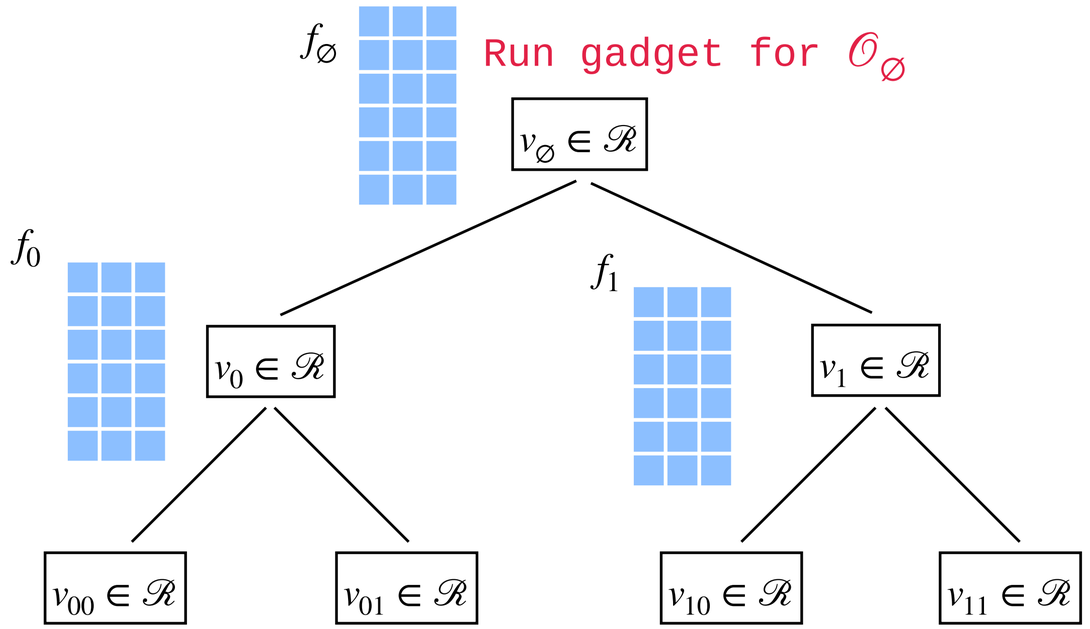
|00110101|10010100|00001110|

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

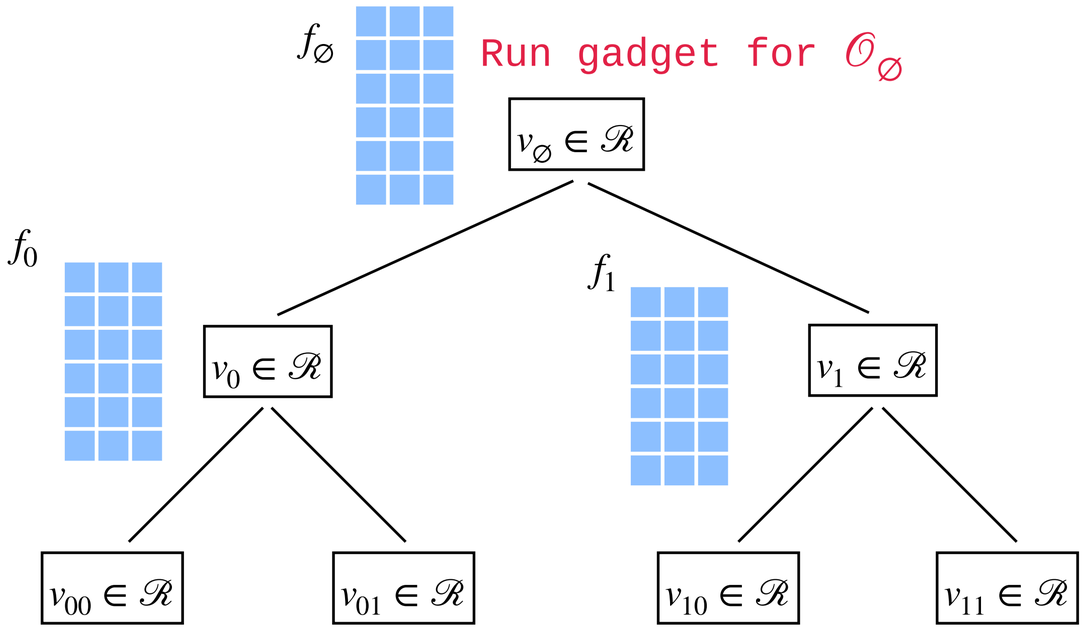
|00110101|10010100|00001110|

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|

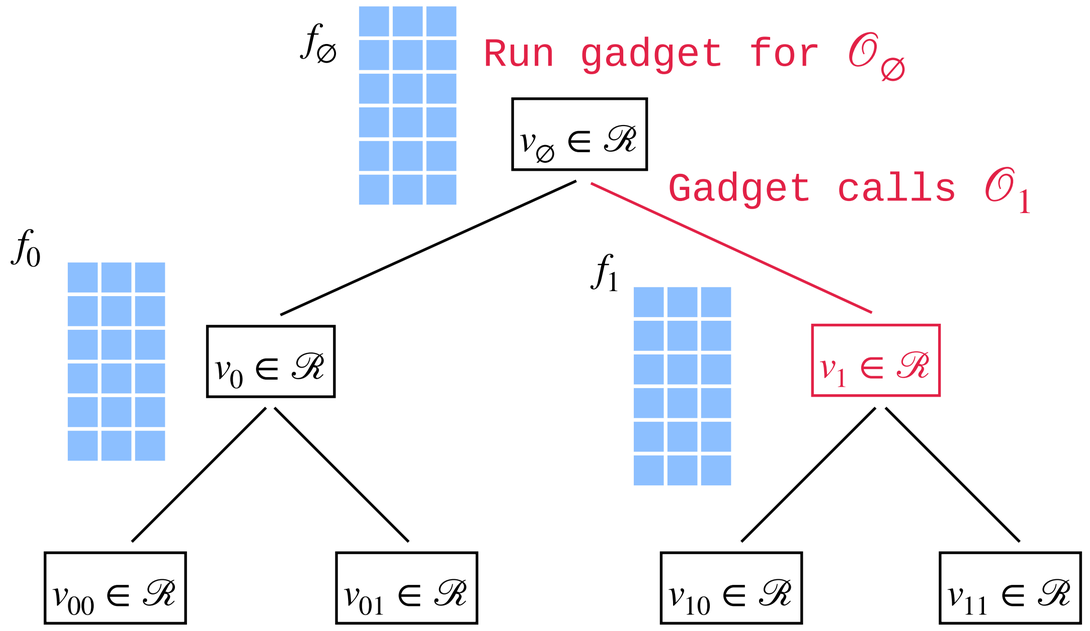
Root
$\varnothing$

From One Level to Many

Cheatsheet

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

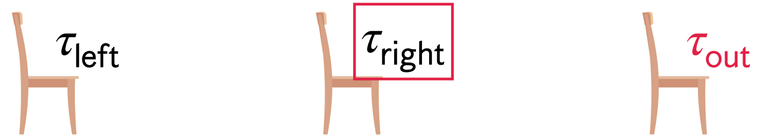
- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

|00110101|10010100|00001110|

Root
$\varnothing$

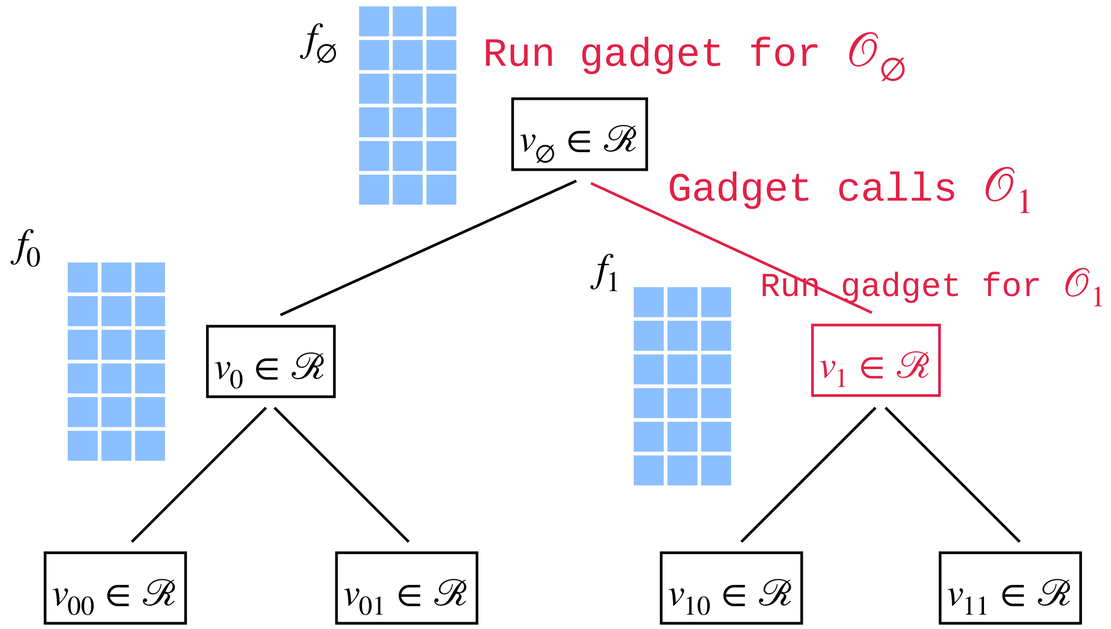


From One Level to Many

Cheatsheet

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

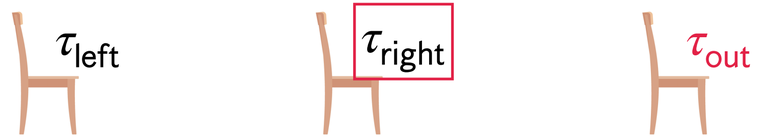
- **Proof idea:** our goal is to implement $\mathcal{O}_\varnothing$
- Use the one-level gadget to achieve this recursively
- Need only 3 registers of size $\log|\mathcal{R}|$, rather than h registers as in naive recursion



Catalytic tape

|00110101|10010100|00001110|

Root
$\varnothing$

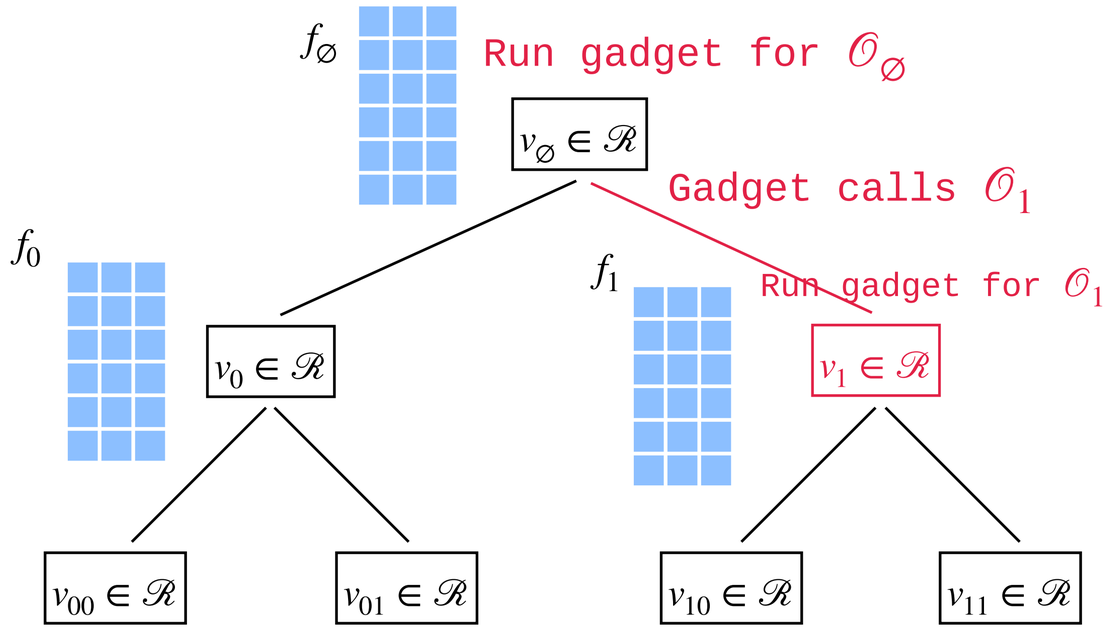


From One Level to Many

Cheatsheet

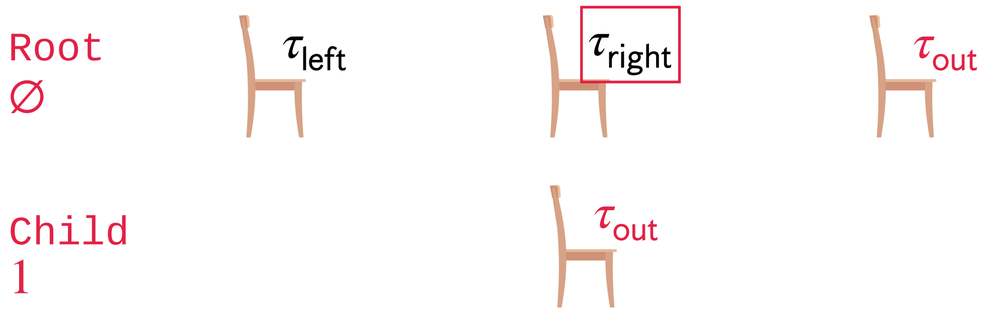
$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **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|

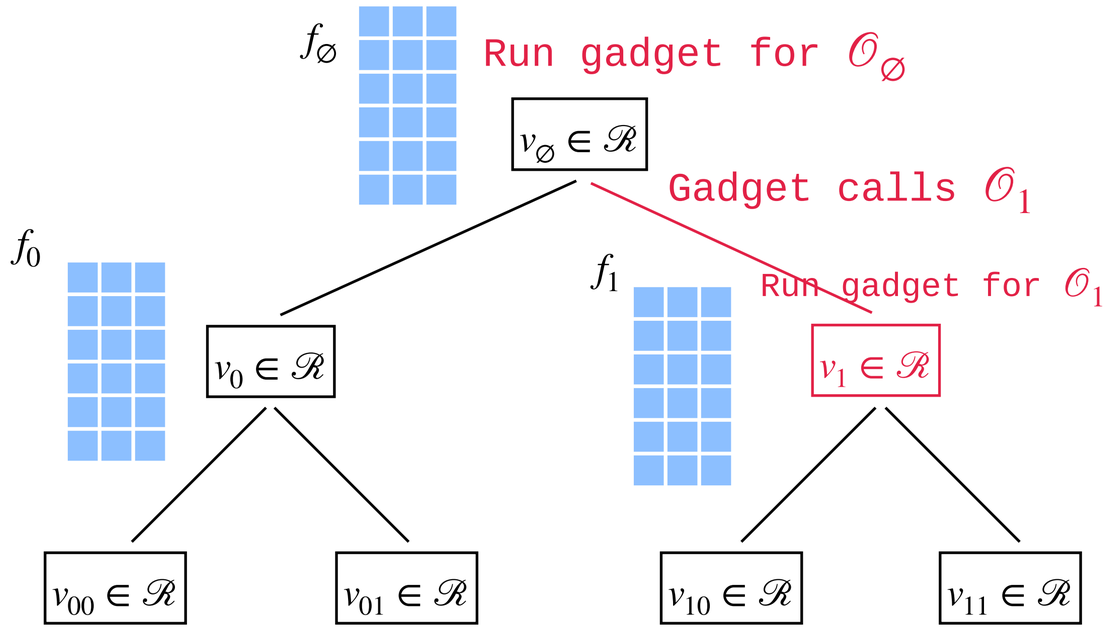


From One Level to Many

Cheatsheet

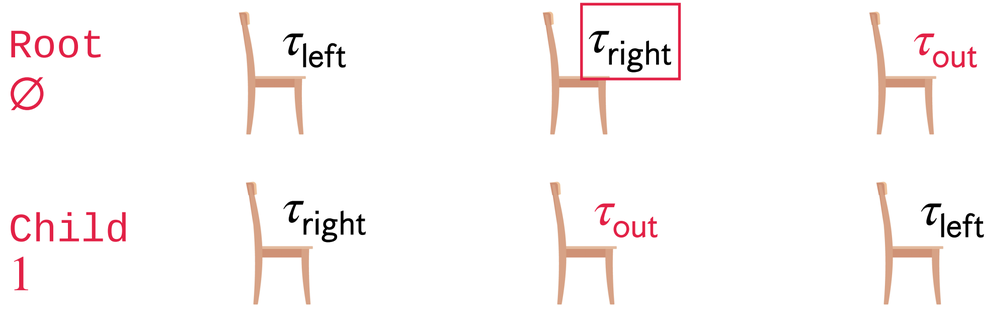
$\mathcal{O}_u(c \in \{0,1\}, \tau \in \mathcal{R})$
updates $\tau \leftarrow \tau + (-1)^c v_u$

- **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|

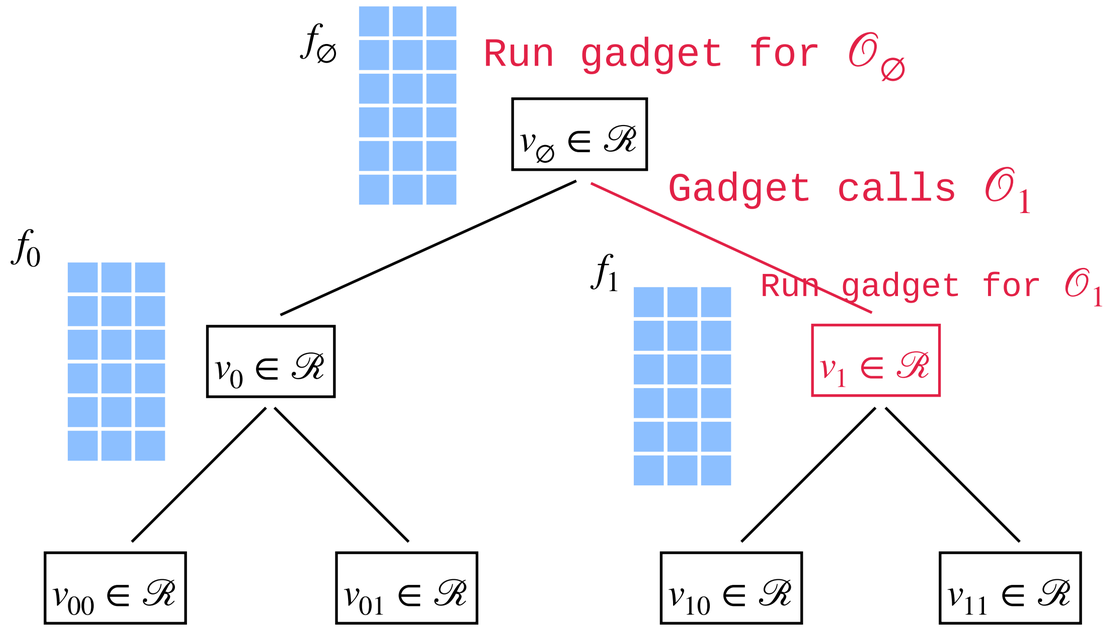


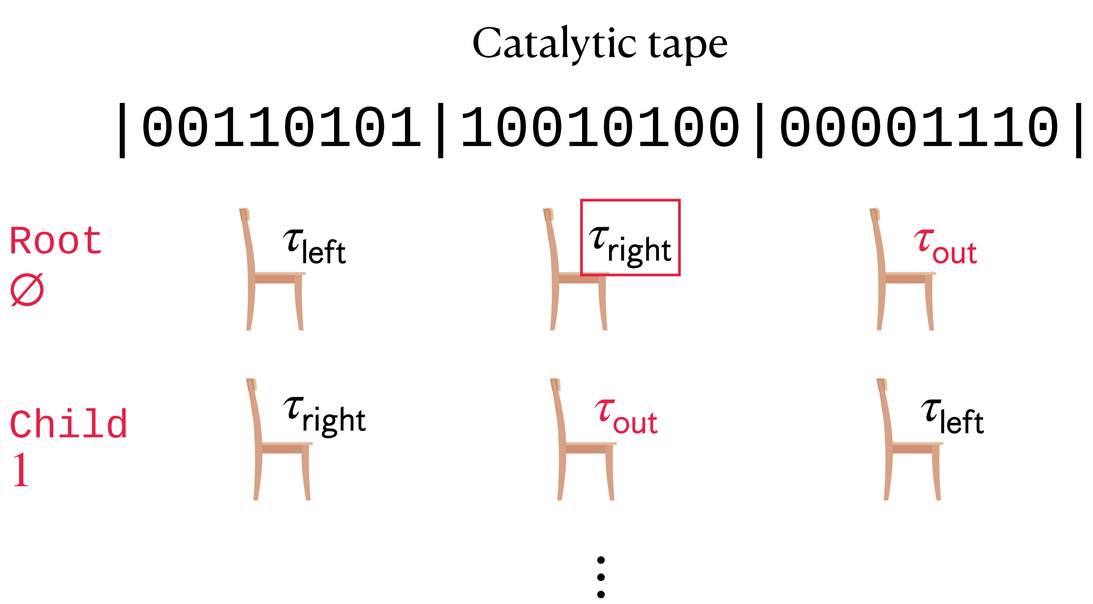
From One Level to Many

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



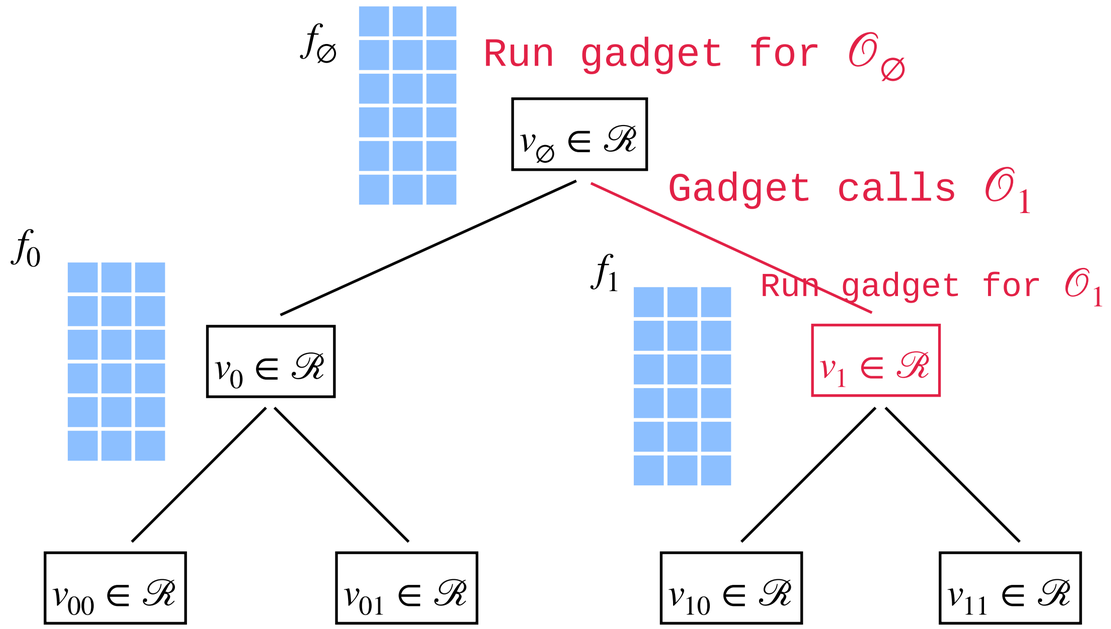


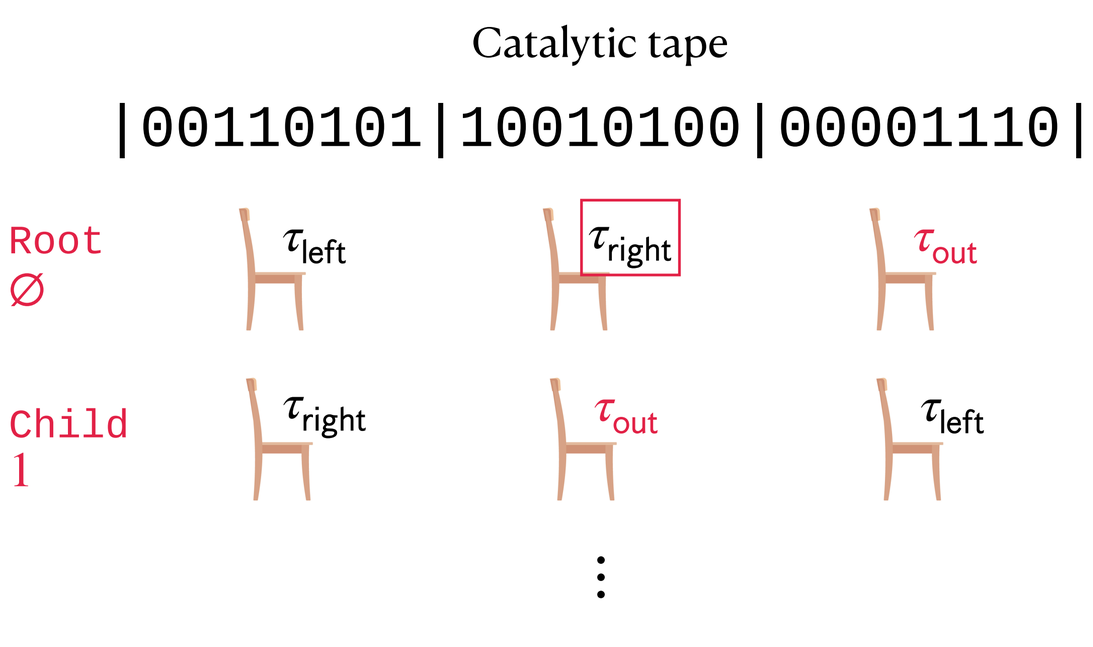
From One Level to Many

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

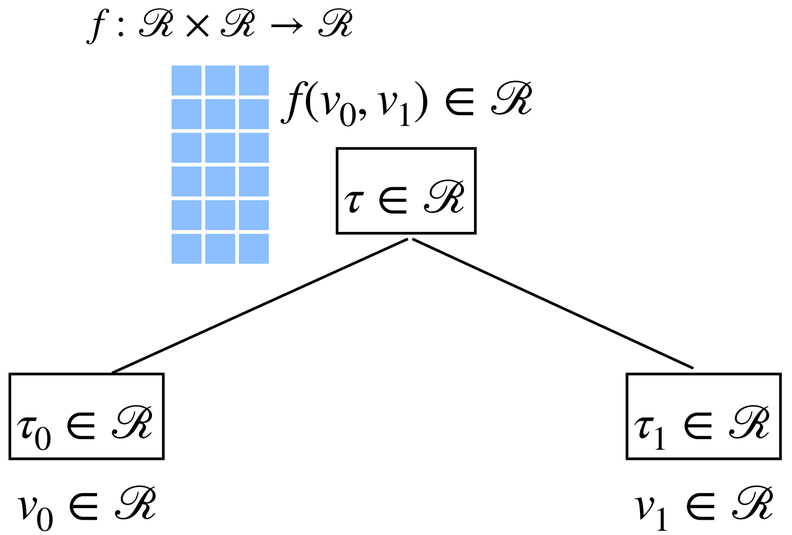
Centrepiece of Cook-Mertz and our work: all I will focus on for the rest of the talk

- 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

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

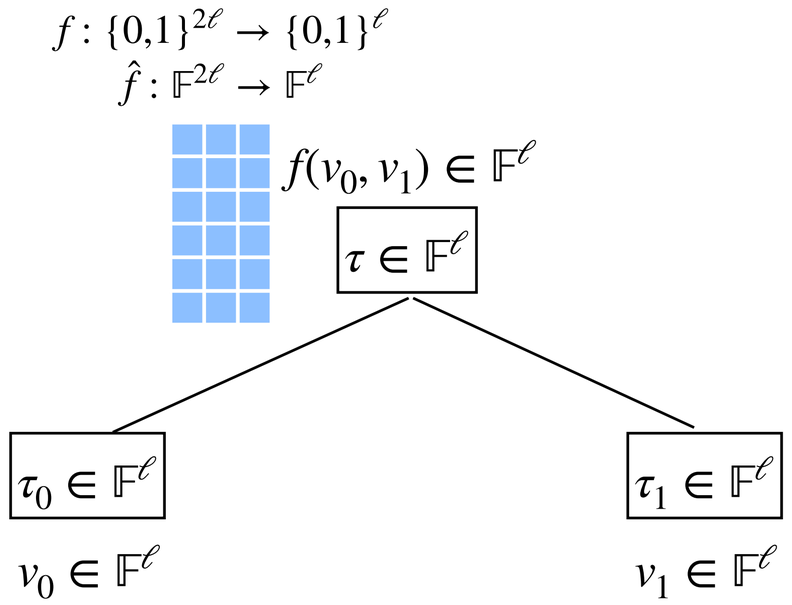


Metrics

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

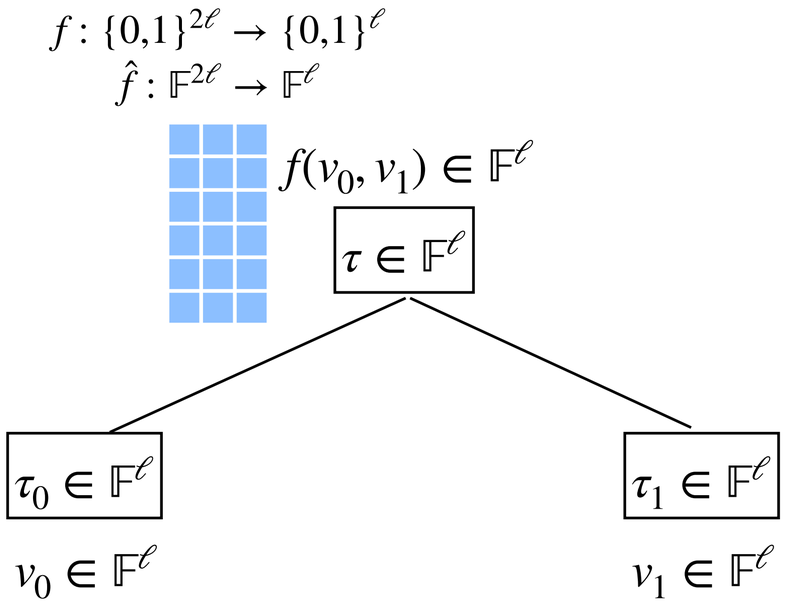
Cook-Mertz's One-Level Gadget

Setup


$f : \{0,1\}^{2\ell} \to \{0,1\}^{\ell}$
$\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
$f(v_0, v_1) \in \mathbb{F}^{\ell}$
$\tau \in \mathbb{F}^{\ell}$
$\tau_0 \in \mathbb{F}^{\ell}$
$\tau_1 \in \mathbb{F}^{\ell}$
$v_0 \in \mathbb{F}^{\ell}$
$v_1 \in \mathbb{F}^{\ell}$

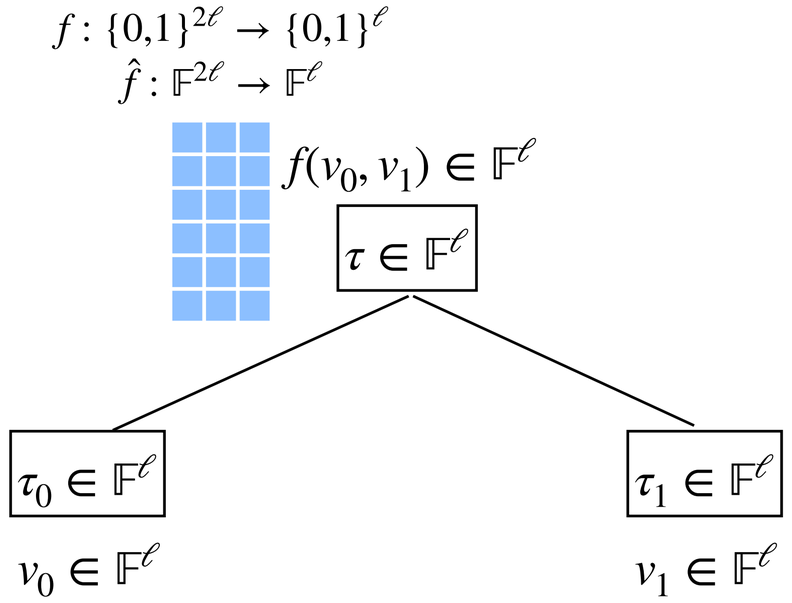
Setup

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



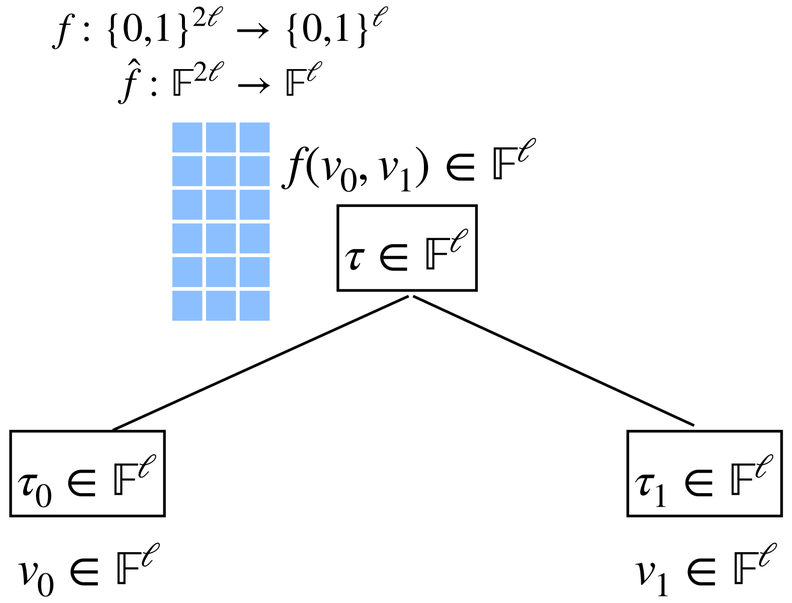
Setup

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



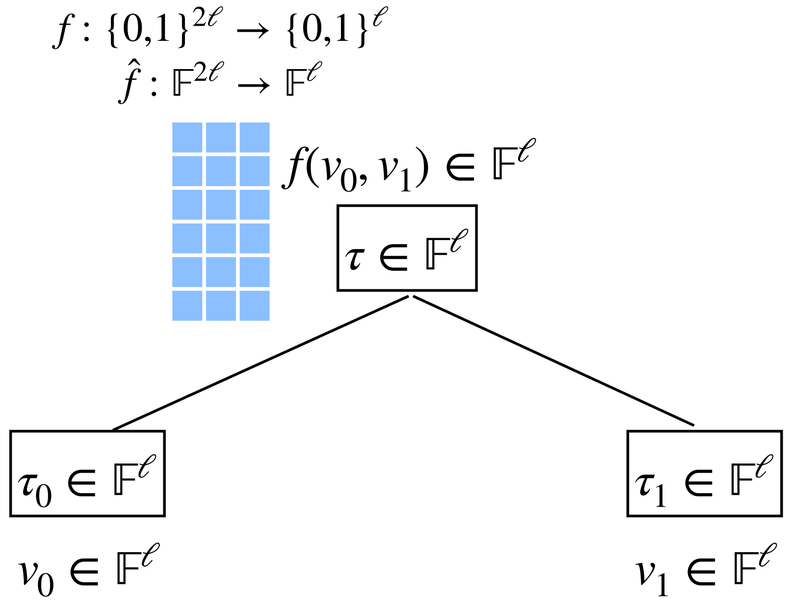
Setup

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



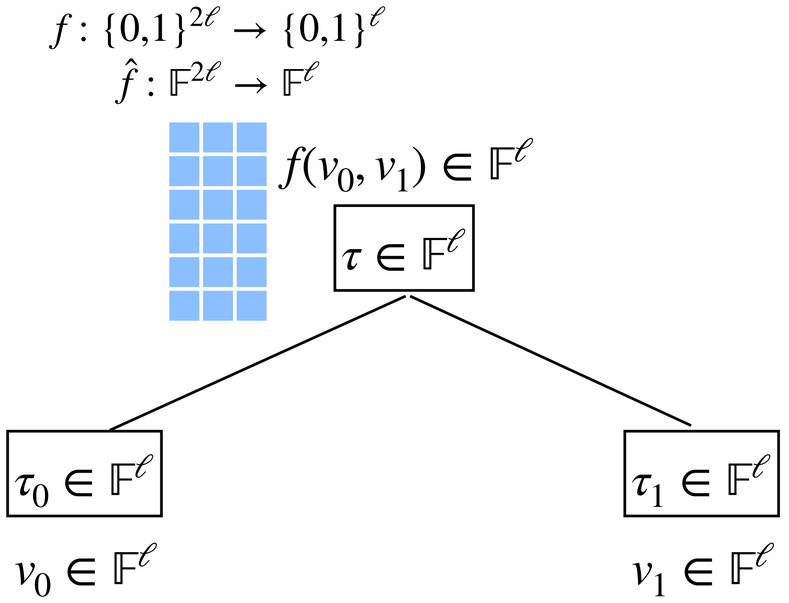
Setup

- 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 $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$



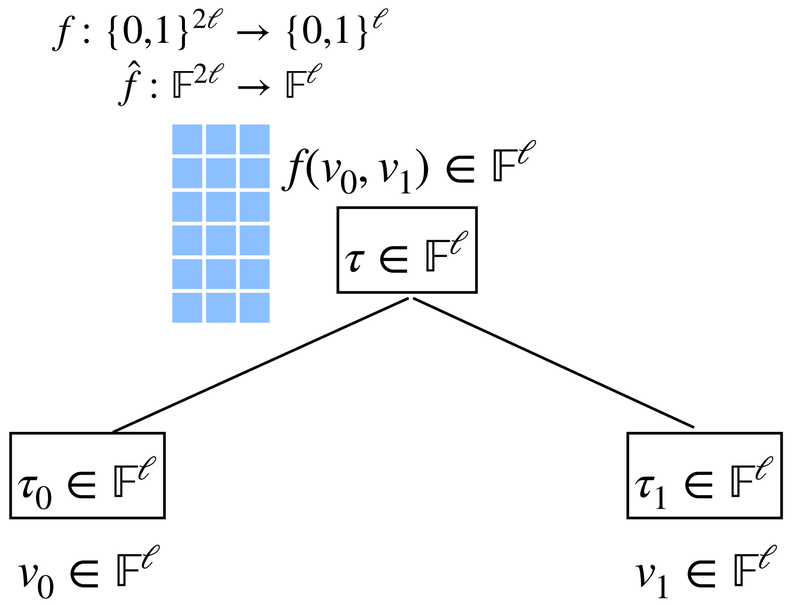
Setup

- 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 $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$
- Define the line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$



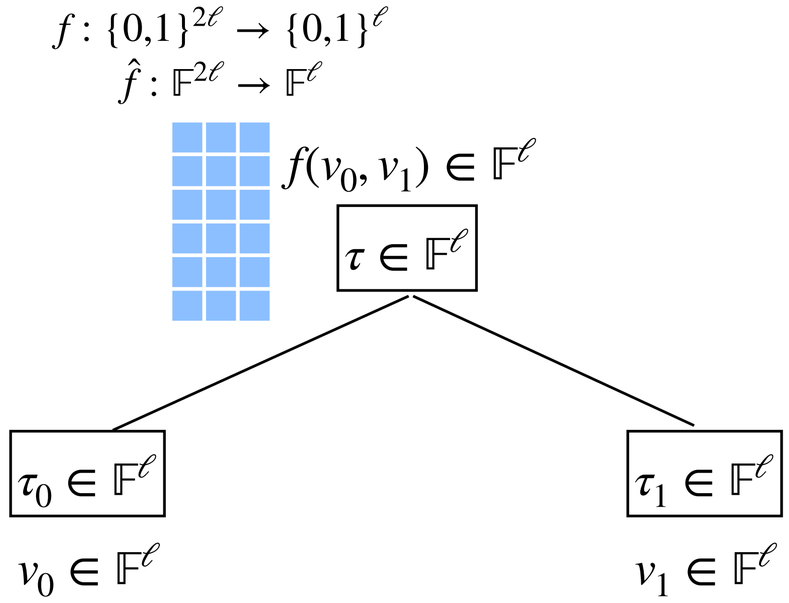
Setup

- 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 $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$
- Define the line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$
- Idea: recover $g = \hat{f}\Big|_L$ via Lagrange interpolation

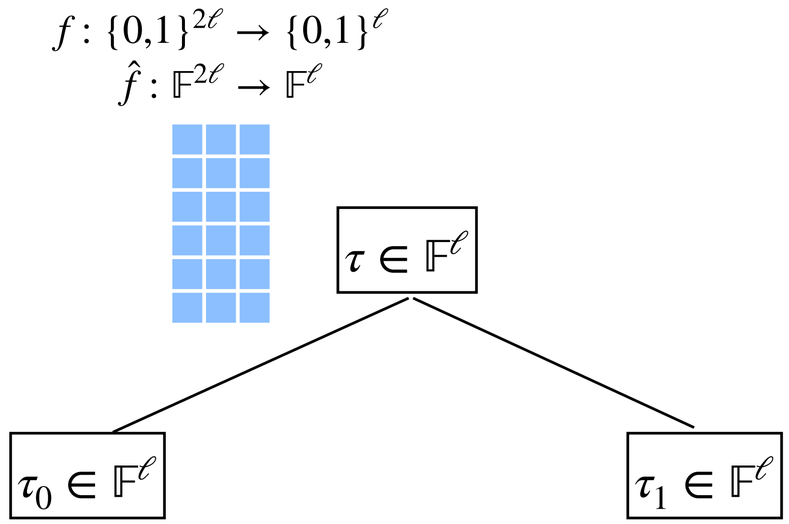


Setup

- 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 $x, y \in \{0,1\}^{\ell}$: $\hat{f}(x, y) = f(x, y)$
- Define the line $L = \{(v_0 + \lambda\tau_0, v_1 + \lambda\tau_1) : \lambda \in \mathbb{F}\}$
- Idea: recover $g = \hat{f}\Big|_L$ via Lagrange interpolation
  - Then can just add $g(0)$ into τ

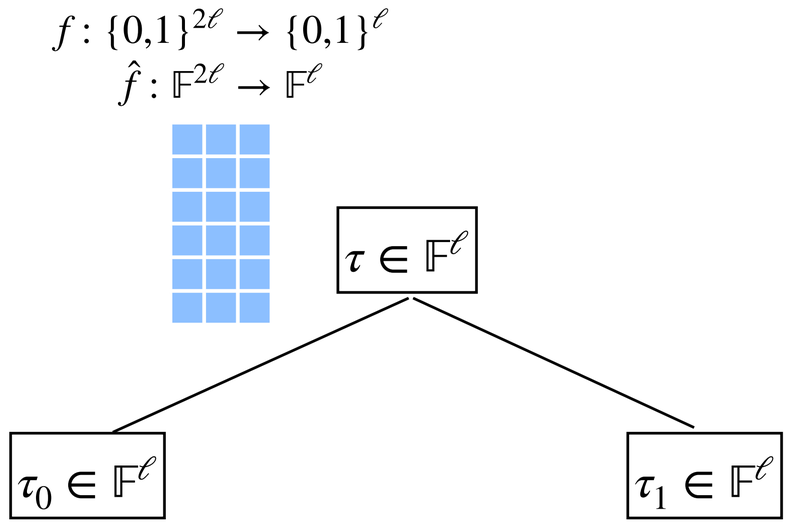


Details


$f : \{0,1\}^{2\ell} \to \{0,1\}^{\ell}$
$\hat{f} : \mathbb{F}^{2\ell} \to \mathbb{F}^{\ell}$
$\tau \in \mathbb{F}^{\ell}$
$\tau_0 \in \mathbb{F}^{\ell}$
$\tau_1 \in \mathbb{F}^{\ell}$

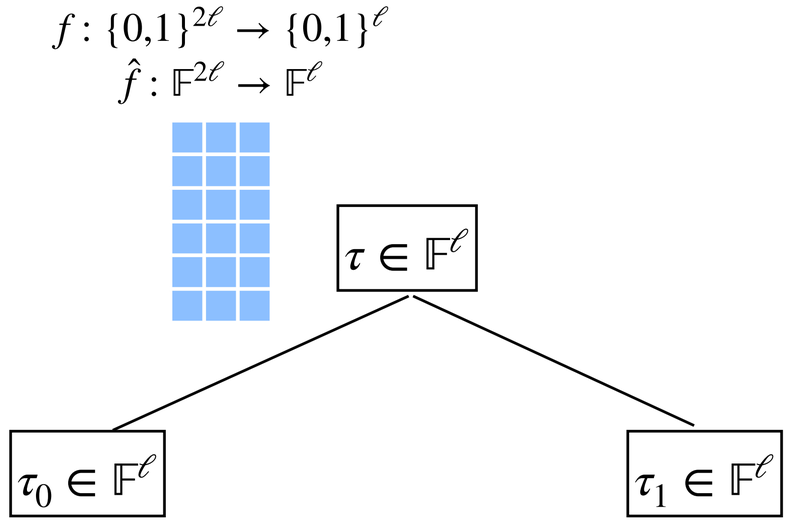
Details

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



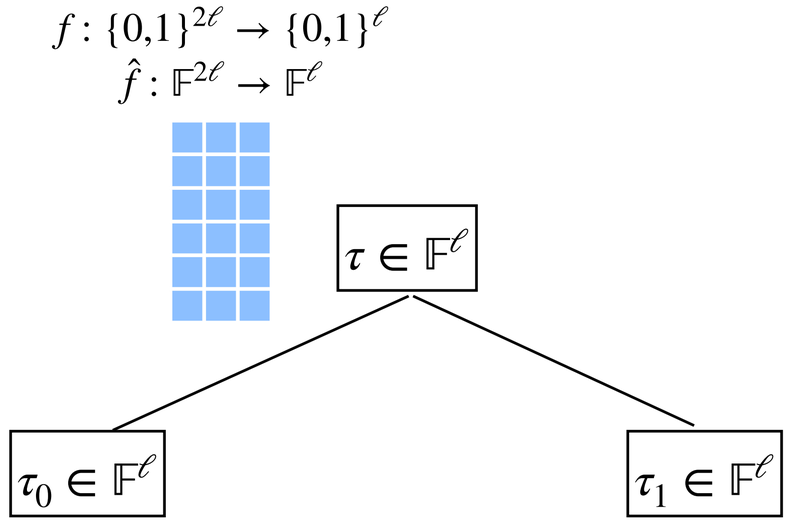
Details

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



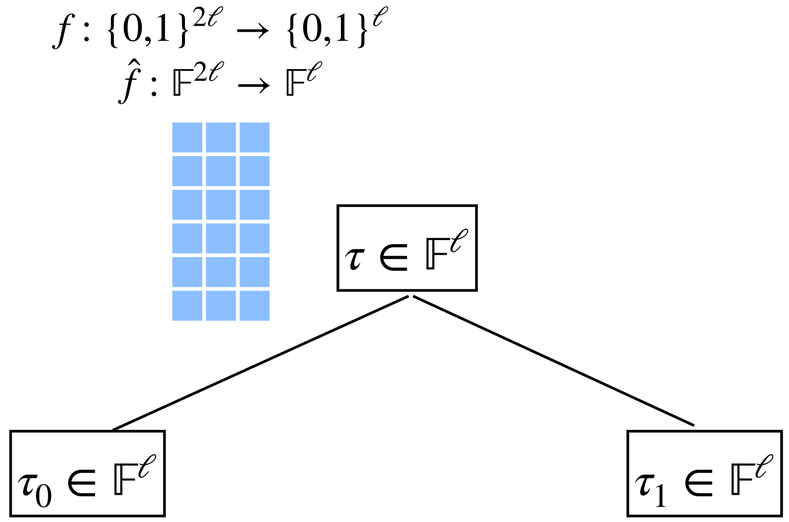
Details

- 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: $g(0) = \sum_{\lambda=1}^{2\ell+1} c_\lambda g(\lambda)$



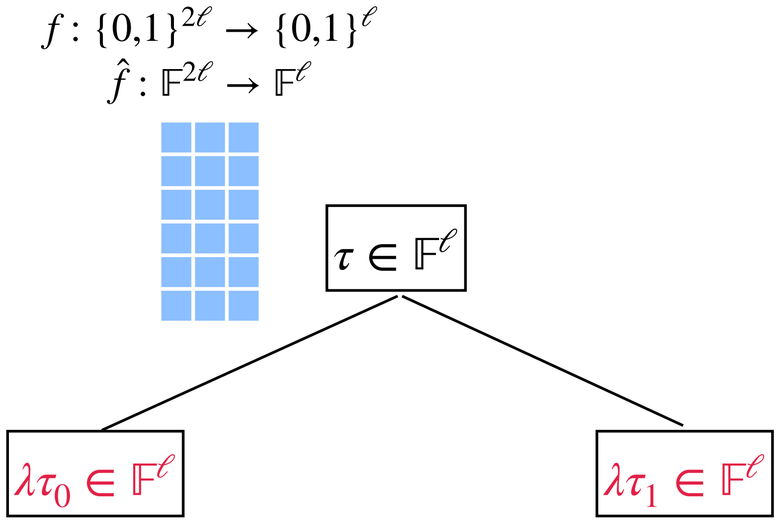
Details

- 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 $\lambda = 1,2,\ldots,2\ell + 1$:



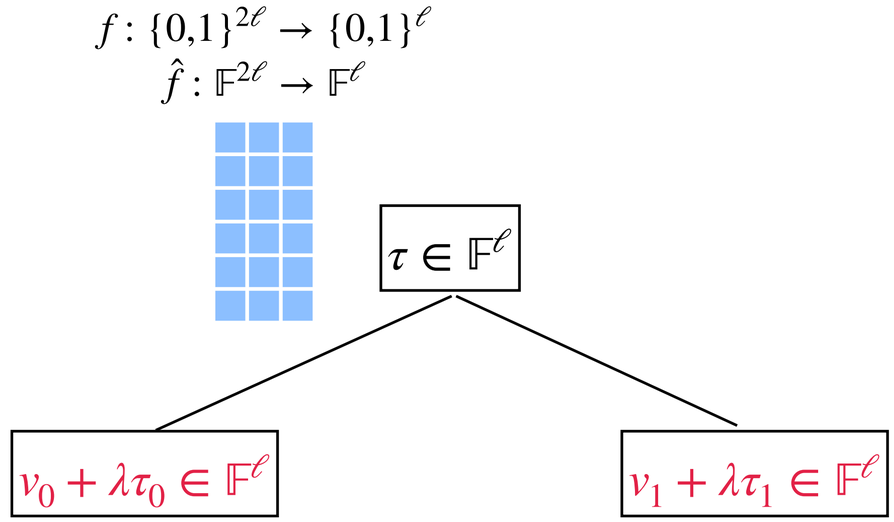
Details

- 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 $(\tau_0, \tau_1) \leftarrow (\lambda\tau_0, \lambda\tau_1)$



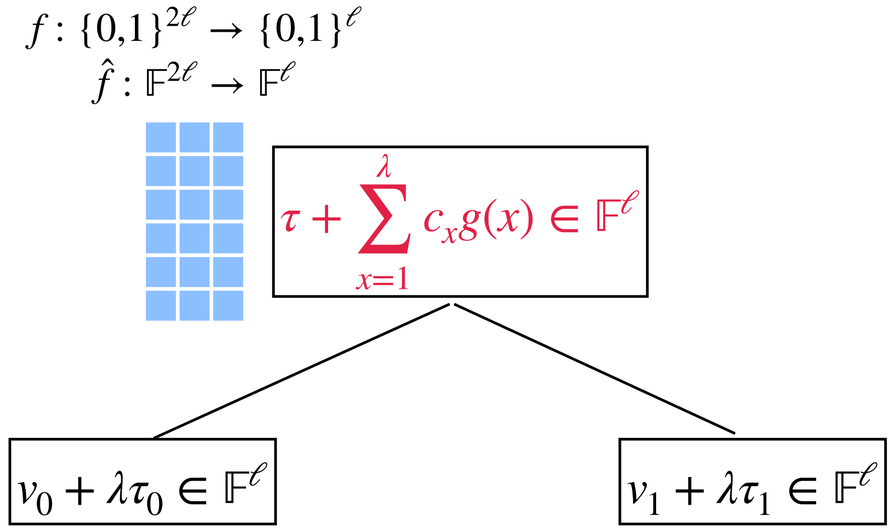
Details

- 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



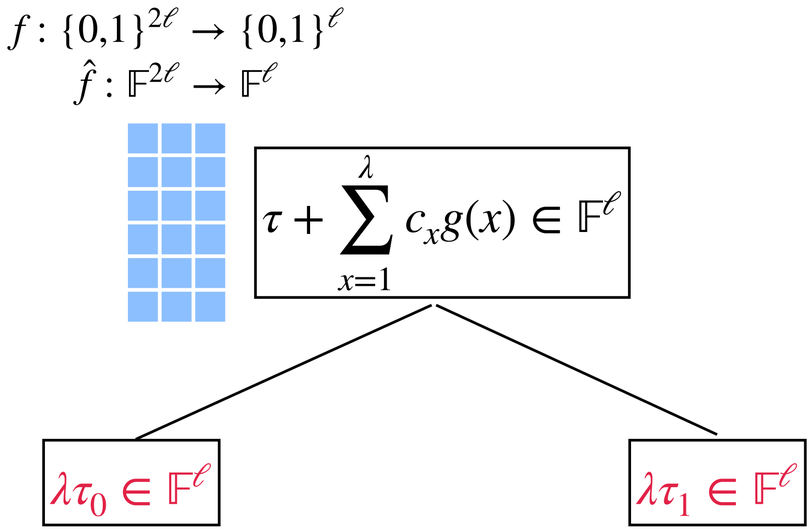
Details

- 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



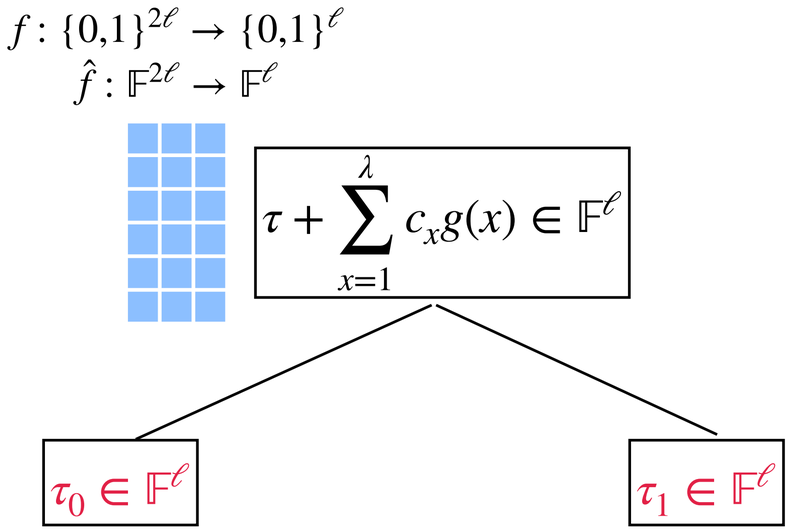
Details

- 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



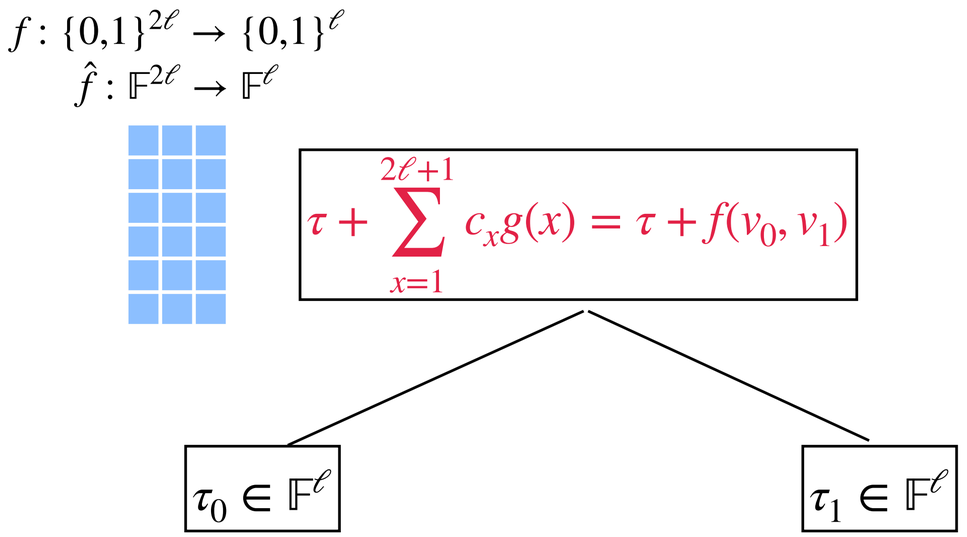
Details

- 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

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

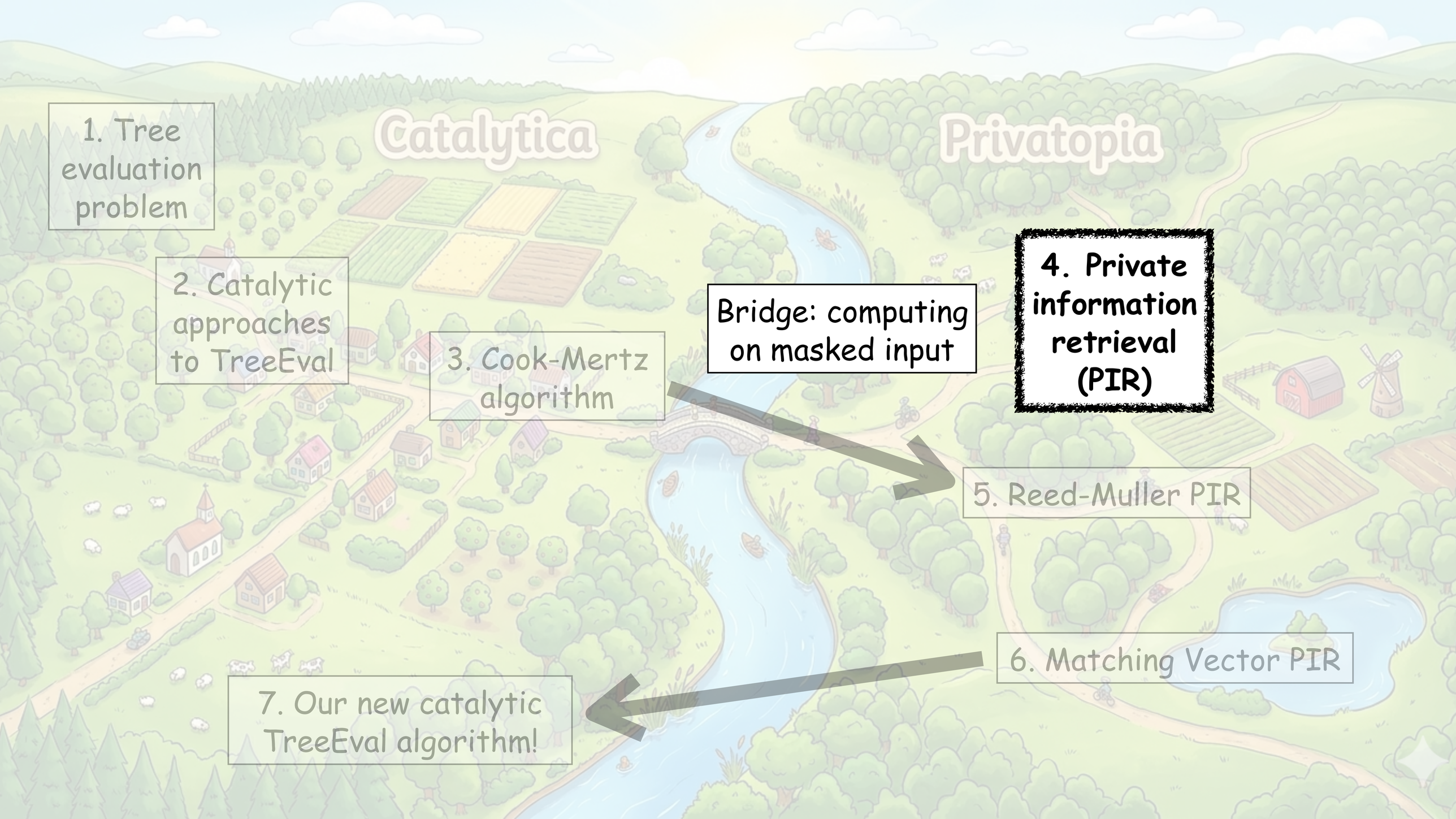
- $\mathscr{R} = \mathbb{F}^{\ell}$
  - Final catalytic space
    $\log|\mathscr{R}| = \ell \log \ell$
    $= O(\log n \log\log n)$
- Free space: $\log \ell$
  - Final free space:
    $h \log \ell = O(\log n \log\log n)$
- Oracle queries: $O(\ell)$

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: $\ell^h = n^{O(\log\log n)}$


Catalytica
Privatopia
1. Tree evaluation problem
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

Cook-Mertz: used Lagrange interpolation to evaluate a polynomial $\hat{f}$ on input (v_0, v_1), but given only maskings

$$\underbrace{(v_0, v_1)}_{\text{input}} + \underbrace{\lambda(\tau_0, \tau_1)}_{\text{mask}} \text{ (for } \lambda \neq 0\text{)}.$$

Cook-Mertz: used Lagrange interpolation to evaluate a polynomial $\hat{f}$ on input (v_0, v_1), but given only maskings

$$\underbrace{(v_0, v_1)}_{\text{input}} + \underbrace{\lambda(\tau_0, \tau_1)}_{\text{mask}} \text{ (for } \lambda \neq 0\text{)}.$$

Q: Why masking?

A: To reuse space without destroying information.

Cook-Mertz: used Lagrange interpolation to evaluate a polynomial $\hat{f}$ on input (v_0, v_1), but given only maskings

$$\underbrace{(v_0, v_1)}_{\text{input}} + \underbrace{\lambda(\tau_0, \tau_1)}_{\text{mask}} \text{ (for } \lambda \neq 0\text{).}$$

Q: Why masking?

A: To reuse space without destroying information.

(This is why we need $\lambda \neq 0$.)

Cook-Mertz: used Lagrange interpolation to evaluate a polynomial $\hat{f}$ on input (v_0, v_1), but given only maskings

$$\underbrace{(v_0, v_1)}_{\text{input}} + \underbrace{\lambda(\tau_0, \tau_1)}_{\text{mask}} \text{ (for } \lambda \neq 0\text{).}$$

Q: Why masking?
A: To reuse space without destroying information.
(This is why we need $\lambda \neq 0$.)

Another reason why one would use masking is **privacy...**

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

Server 1 

Server 2 

Server S ... 

Client

$i \in \{1,2,\ldots,n\}$

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

Server 1 Server 2 ... Server S

q_1 q_2 q_S

Client

$i \in \{1,2,\ldots,n\}$

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

Server 1 Server 2 ... Server S



q_1 a_1 q_2 a_2 q_S a_S

Client

$i \in \{1,2,\ldots,n\}$

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

Server 1 Server 2 Server S …

q_1 a_1 q_2 a_2 q_S a_S

Client

$i \in \{1,2,\ldots,n\}$

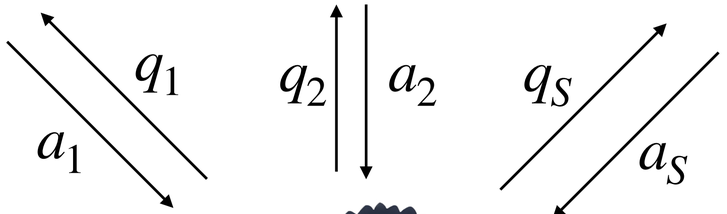
- Correctness: client can deduce DB_i from $\{q_j, a_j\}_{j\in[S]}$

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

Server 1 Server 2 ... Server S



Client

$i \in \{1,2,\ldots,n\}$

- Correctness: client can deduce DB_i from $\{q_j, a_j\}_{j\in[S]}$
- Privacy: each server learns nothing about about i provided that they can't talk to each other

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

Server 1 Server 2 Server S …

q_1 a_1 q_2 a_2 q_S a_S

Client

$i \in \{1,2,\ldots,n\}$

- Correctness: client can deduce DB_i from $\{q_j, a_j\}_{j\in[S]}$
- Privacy: each server learns nothing about about i provided that they can't talk to each other
  - Formally: the marginal distribution of each q_j is independent of i

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$i \in \{1,2,\ldots,n\}$

- Correctness: client can deduce DB_i from $\{q_j, a_j\}_{j\in[S]}$
- Privacy: each server learns nothing about about i provided that they can't talk to each other
  - Formally: the marginal distribution of each q_j is independent of i
- Efficiency metrics:
  - # **of servers**

Private Information Retrieval (PIR)

Closely related to locally-decodable codes (LDCs)

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

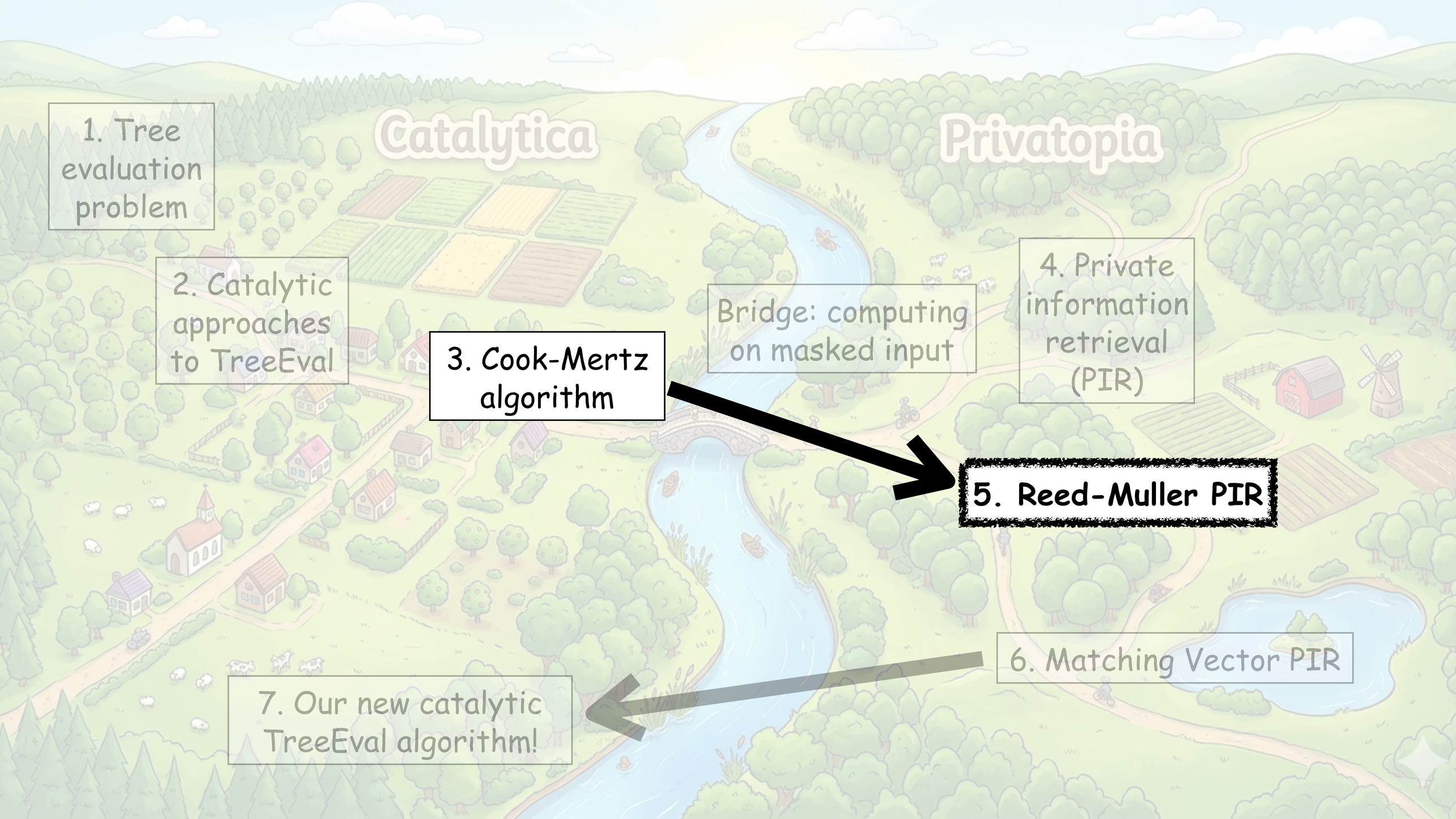
Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

$i \in \{1,2,\ldots,n\}$

- Correctness: client can deduce DB_i from $\{q_j, a_j\}_{j\in[S]}$
- Privacy: each server learns nothing about about i provided that they can't talk to each other
  - Formally: the marginal distribution of each q_j is independent of i
- Efficiency metrics:
  - # **of servers**
  - **Communication complexity** CC: total number of bits in q_j and a_j for one j


Catalytica
Privatopia
1. Tree evaluation problem
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

The Reed-Muller PIR

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

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

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

The Reed-Muller PIR

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

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

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

- Correctness: client can deduce $\hat{f}(v)$ from $\{q_j, a_j\}_{j \in [S]}$

The Reed-Muller PIR

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

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

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

- Correctness: client can deduce $\hat{f}(v)$ from $\{q_j, a_j\}_{j \in [S]}$
- Privacy: each server learns nothing about about v

The Reed-Muller PIR

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

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

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

- Correctness: client can deduce $\hat{f}(v)$ from $\{q_j, a_j\}_{j \in [S]}$
- Privacy: each server learns nothing about about v
  - Achieve this with masking!

The Reed-Muller PIR

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

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

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

- Correctness: client can deduce $\hat{f}(v)$ from $\{q_j, a_j\}_{j \in [S]}$
- Privacy: each server learns nothing about about v
  - Achieve this with masking!
  - Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$

The Reed-Muller PIR

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

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

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

- Correctness: client can deduce $\hat{f}(v)$ from $\{q_j, a_j\}_{j \in [S]}$
- Privacy: each server learns nothing about about v
  - Achieve this with masking!
  - Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
  - Server j replies with $a_j = \hat{f}(q_j) = \hat{f}(v + j\tau)$

The Reed-Muller PIR

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

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

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_S a_S

Client

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

- Correctness: client can deduce $\hat{f}(v)$ from $\{q_j, a_j\}_{j \in [S]}$
- Privacy: each server learns nothing about about v
  - Achieve this with masking!
  - Alice samples $\tau \leftarrow \mathbb{F}^{\ell}$ at random and sets $q_j = v + j\tau$
  - Server j replies with $a_j = \hat{f}(q_j) = \hat{f}(v + j\tau)$
  - Alice recovers $\hat{f}(v)$ with Lagrange interpolation

Reed-Muller PIR Efficiency Metrics

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

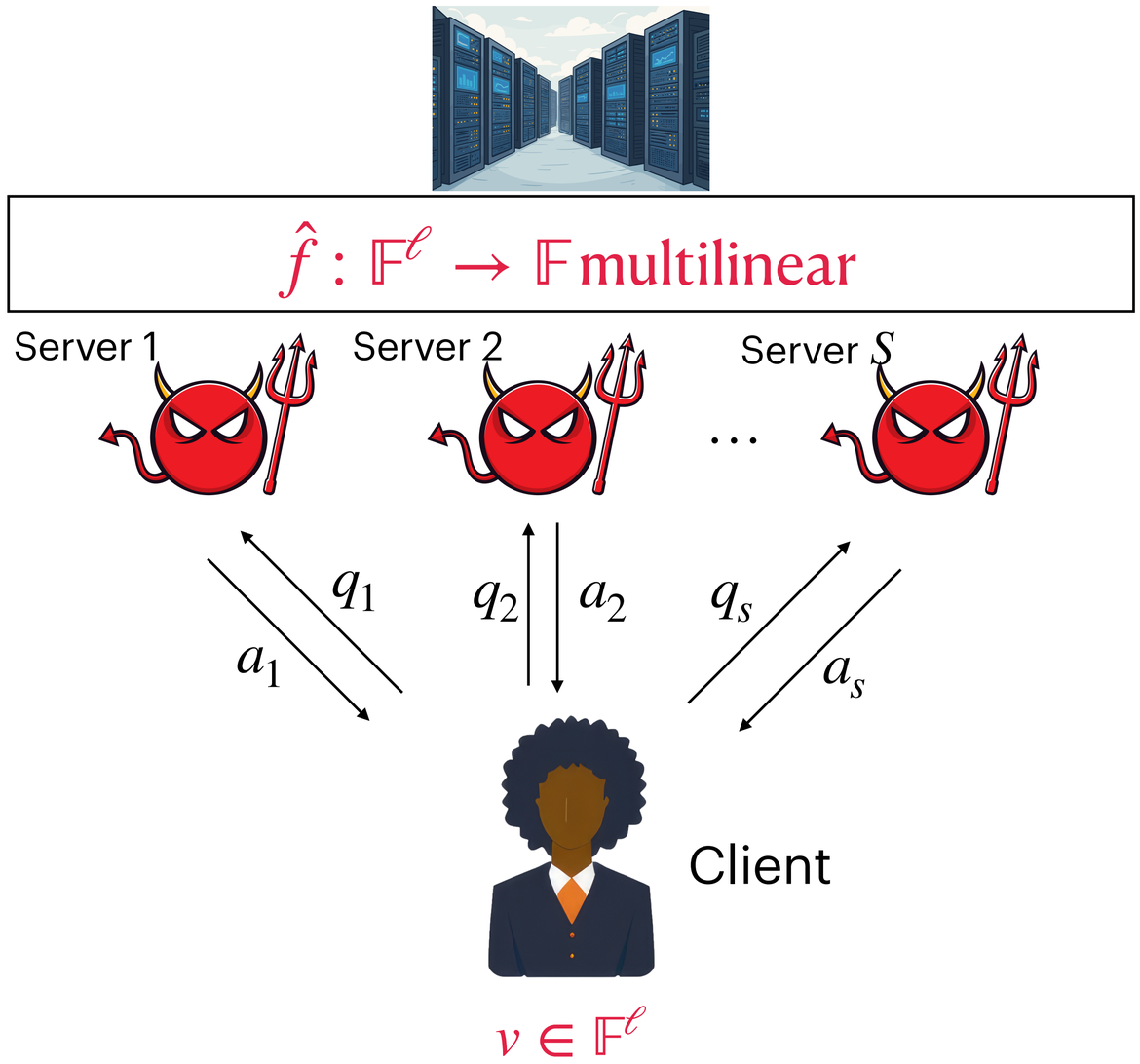
Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

Client

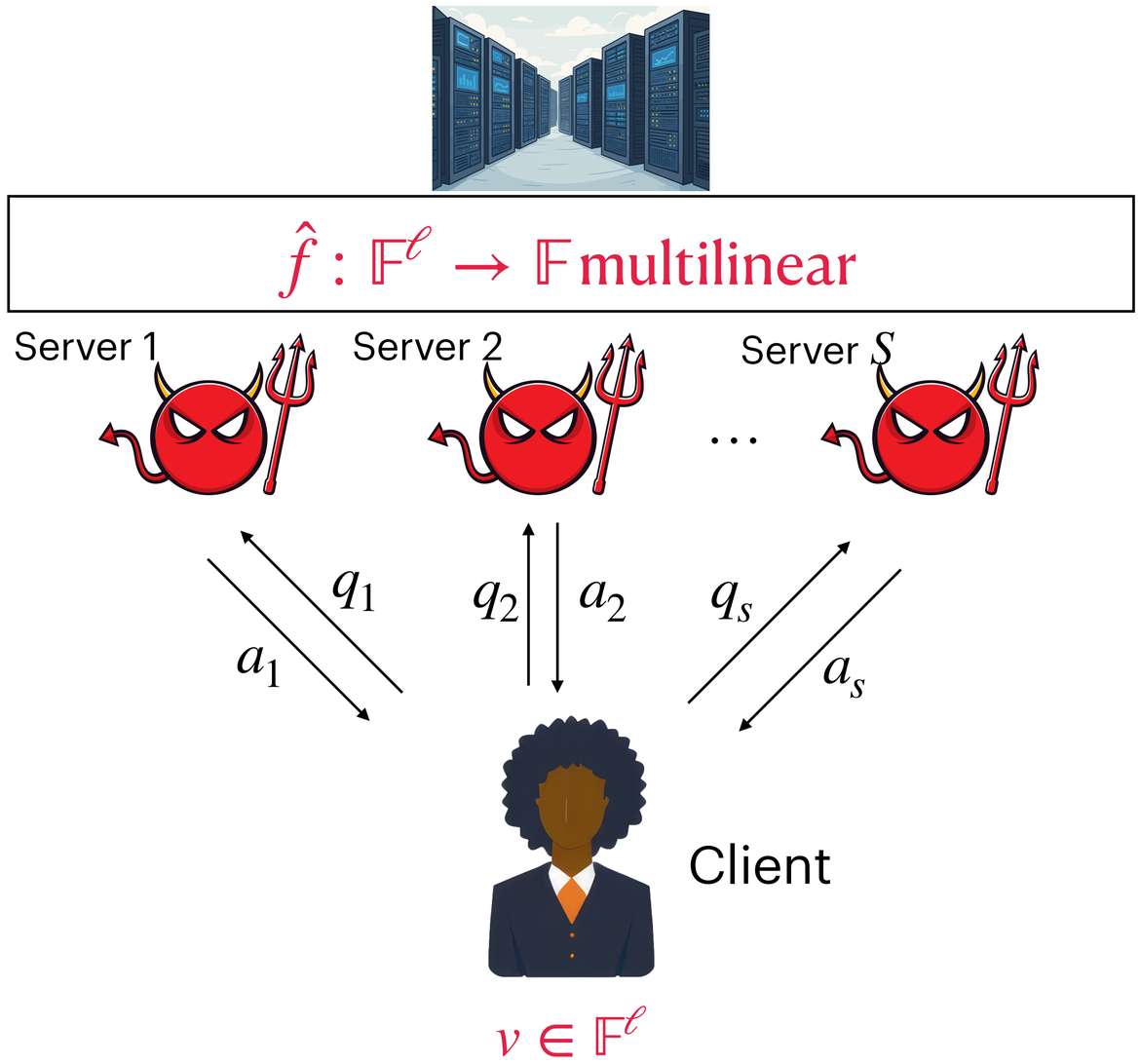
$v \in \mathbb{F}^{\ell}$

Reed-Muller PIR Efficiency Metrics



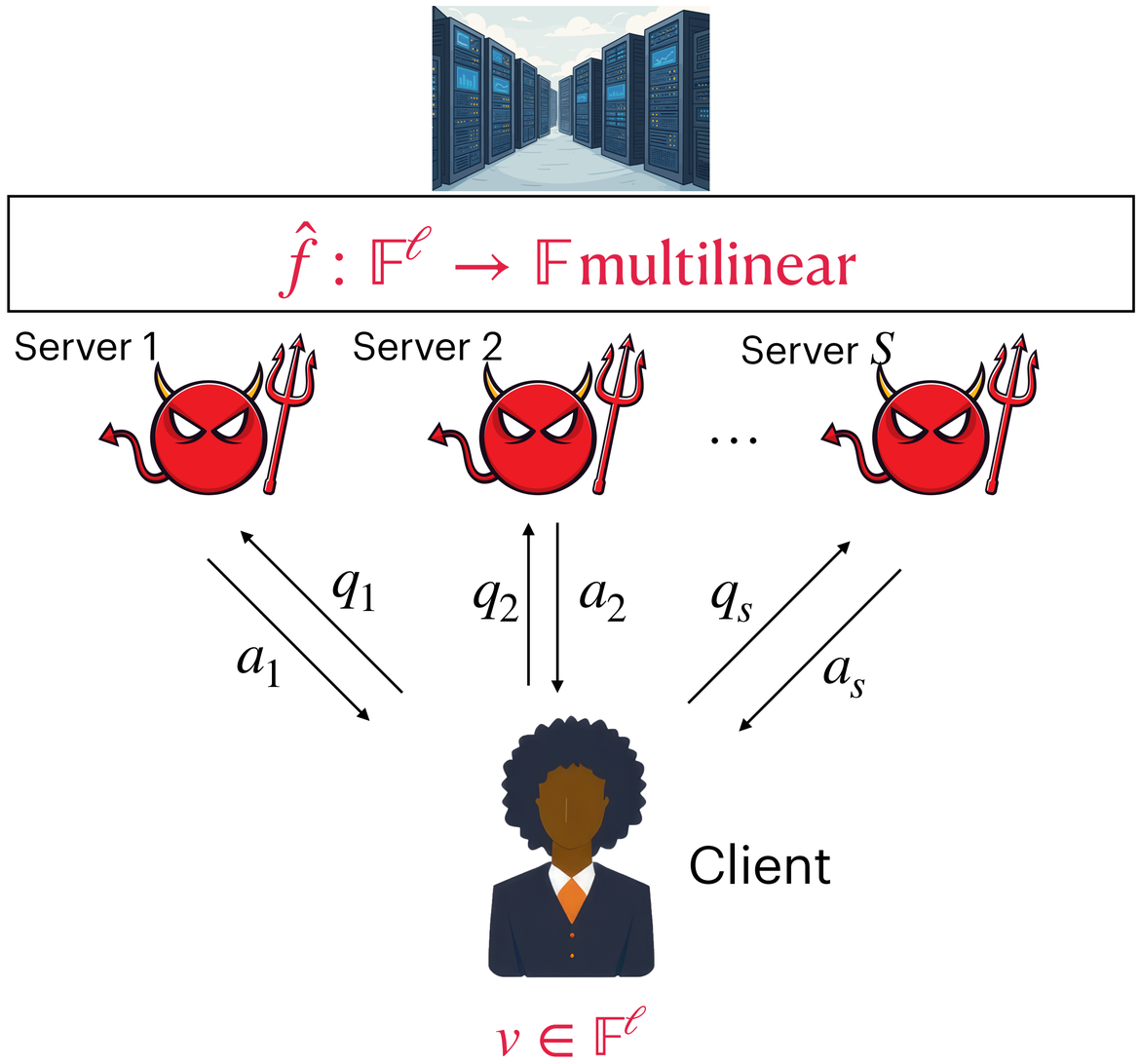
- **# of servers:** $S = O(\ell)$

Reed-Muller PIR Efficiency Metrics



- **# of servers:** $S = O(\ell)$
- **Communication complexity:**

Reed-Muller PIR Efficiency Metrics



- **# of servers:** $S = O(\ell)$
- **Communication complexity:**

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

Informal Dictionary: PIR ↔ TreeEval

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

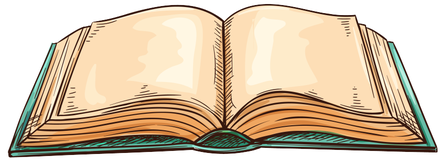
Server 1 Server 2 ... Server S



q_1 a_1 q_2 a_2 q_s a_s

Client

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



Informal Dictionary: PIR ↔ TreeEval

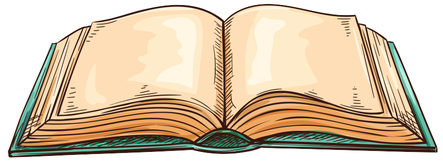
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

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

Informal Dictionary: PIR ↔ TreeEval

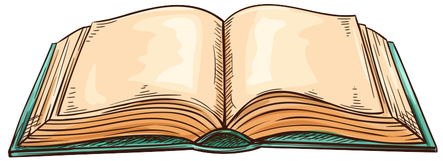
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

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

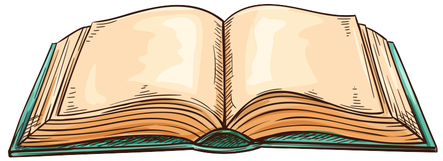
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

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

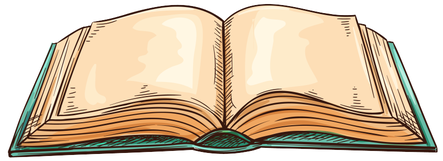
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

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

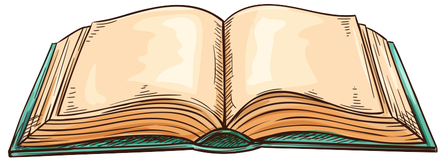
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

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

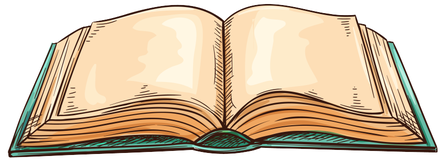
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

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

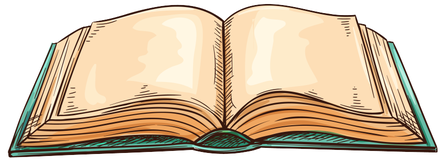
$\hat{f} : \mathbb{F}^{\ell} \to \mathbb{F}$ multilinear

Server 1 Server 2 ... Server S

q_1 a_1 q_2 a_2 q_s a_s

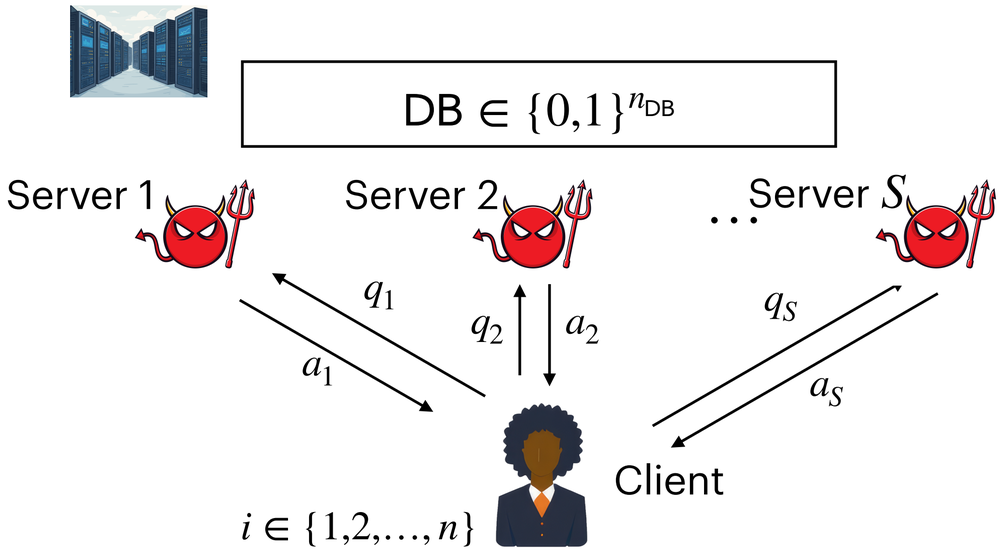
Client

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



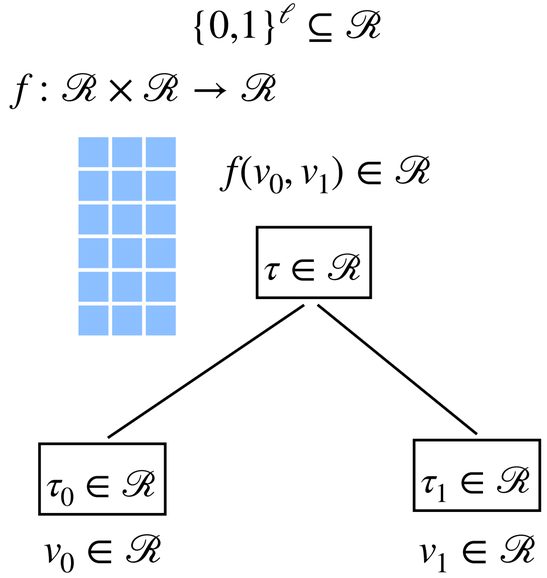
PIR	TreeEval One-Level Gadget	
DB	Truth table of f	
Client's answer	f(v)	
# of servers	# of oracle calls	
Question	Written to catalytic tape by child oracle	$q_j = v + j\tau$ for Reed-Muller
Server's answer	Written to catalytic tape by one-level gadget	$a_j = \hat{f}(v + j\tau)$ for Reed-Muller
CC	Catalytic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

PIR



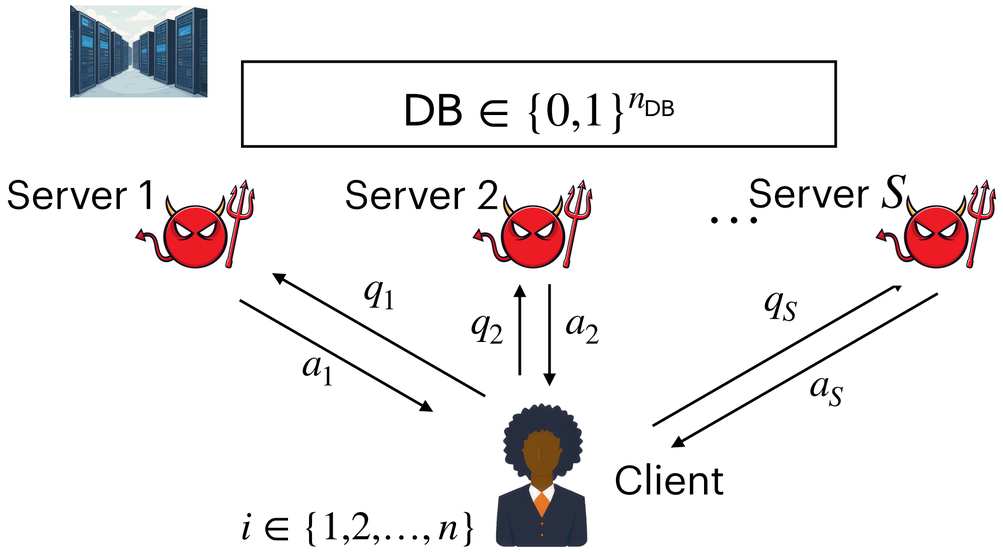
- **Mask:** sampled randomly by client to ensure privacy

TreeEval **One-Level Gadget**



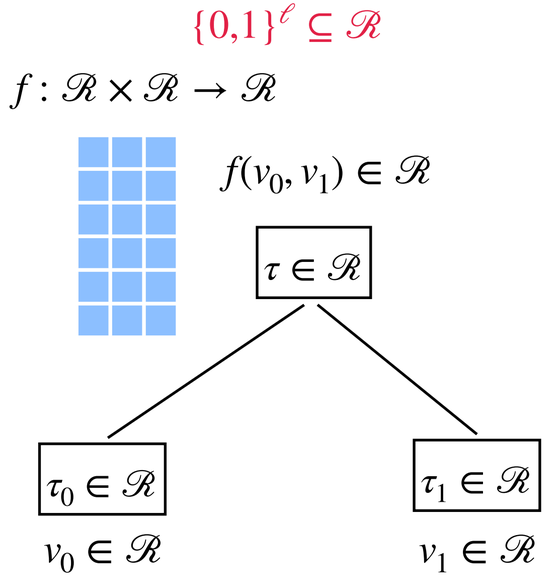
- **Mask:** the existing contents of the catalytic tape

PIR



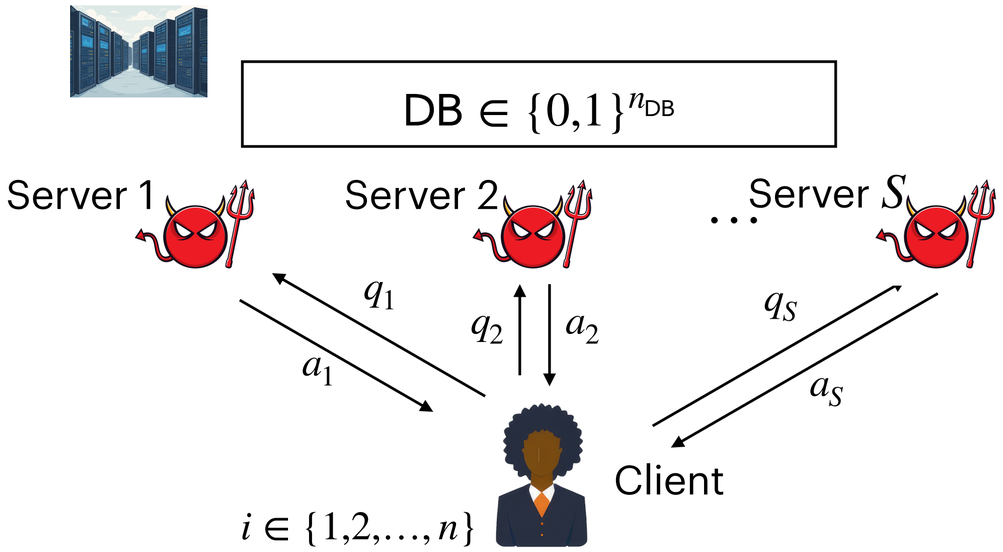
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$

TreeEval **One-Level Gadget**



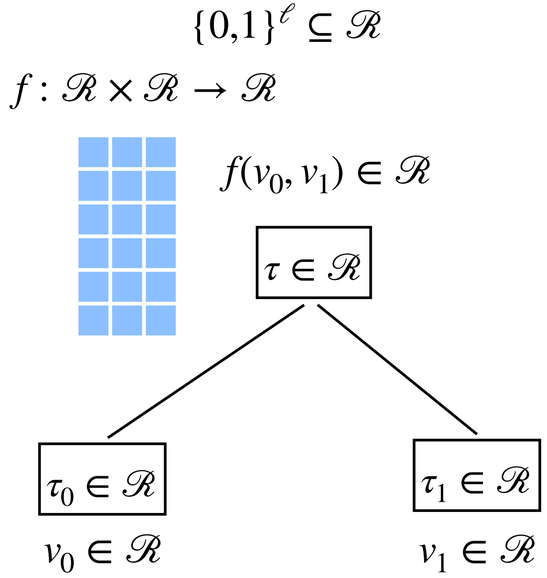
- **Mask:** the existing contents of the catalytic tape

PIR



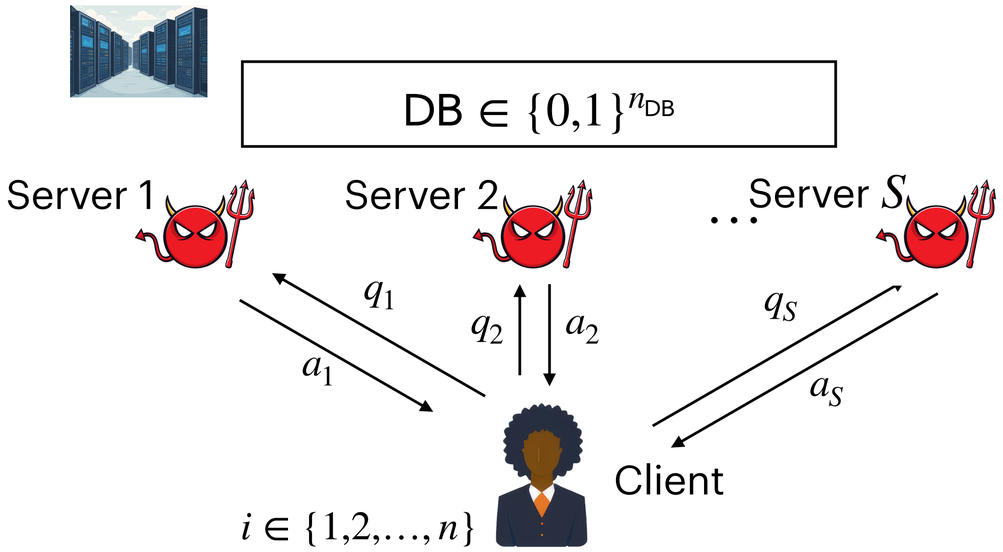
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$
- Communication cost: CC
- # of servers: S

TreeEval **One-Level Gadget**



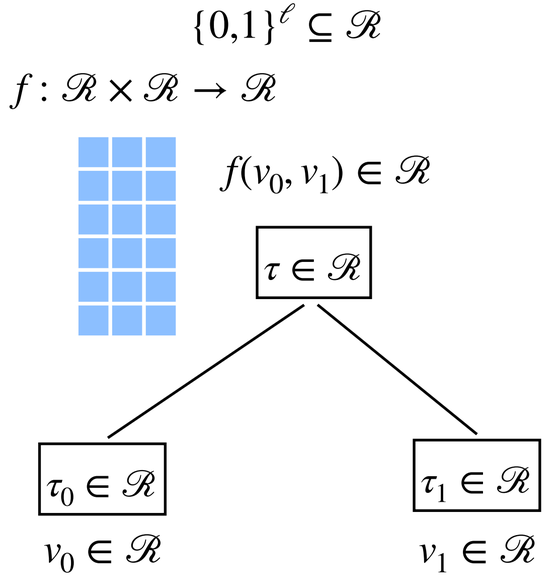
- **Mask:** the existing contents of the catalytic tape

PIR



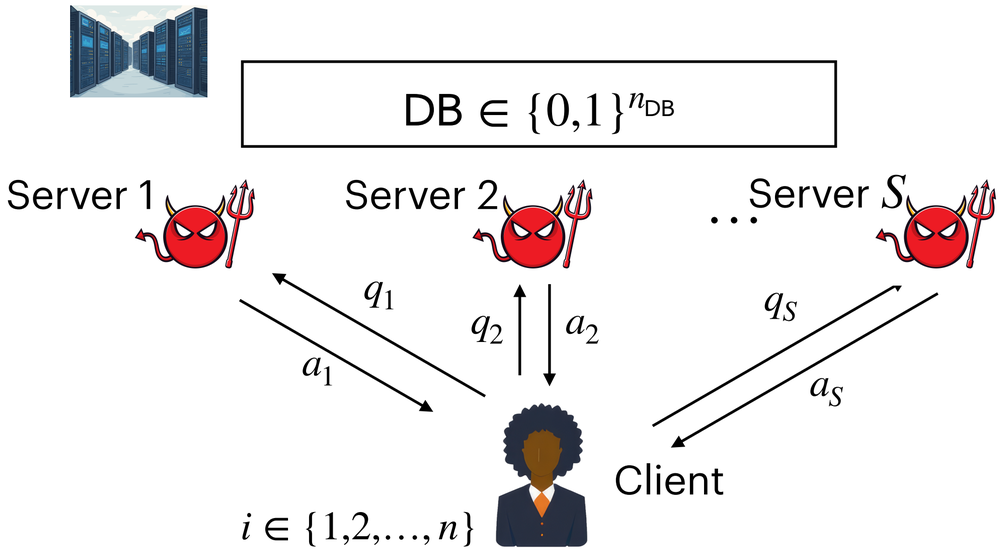
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$
- Communication cost: CC
- # of servers: S

TreeEval **One-Level Gadget**



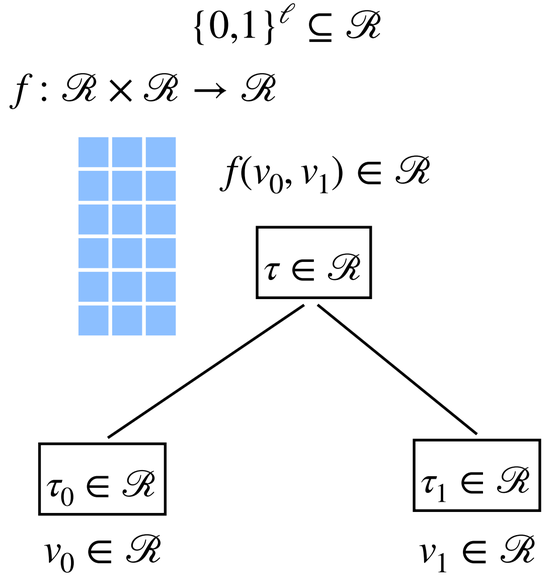
- **Mask:** the existing contents of the catalytic tape
- $\log|\mathscr{R}| = O(\mathsf{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

PIR



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

TreeEval **One-Level Gadget**

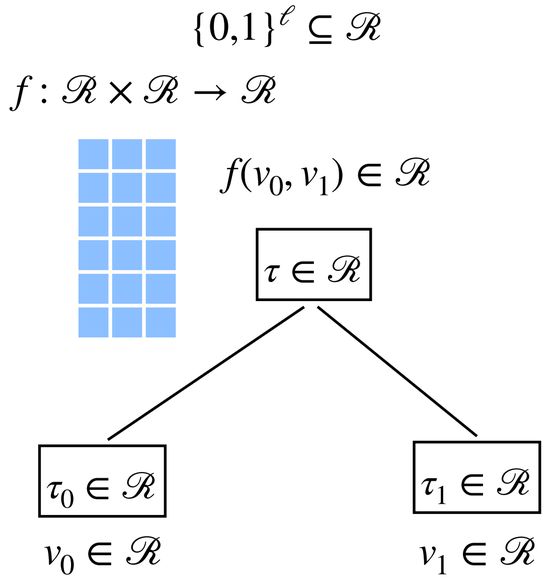


- **Mask:** the existing contents of the catalytic tape
- $\log|\mathscr{R}| = O(\mathsf{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

PIR

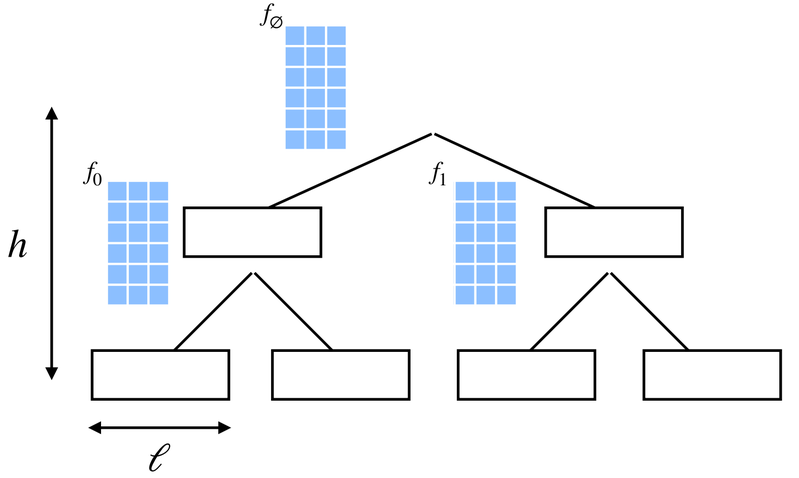
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$
- Communication cost: CC
- # of servers: S

TreeEval **One-Level Gadget**



- **Mask:** the existing contents of the catalytic tape
- $\log|\mathscr{R}| = O(\mathsf{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

TreeEval

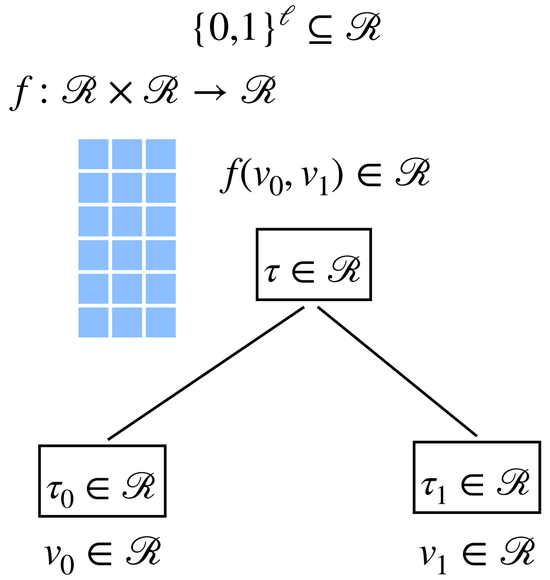


n = total input length = $\mathsf{poly}(2^{h+\ell})$

PIR

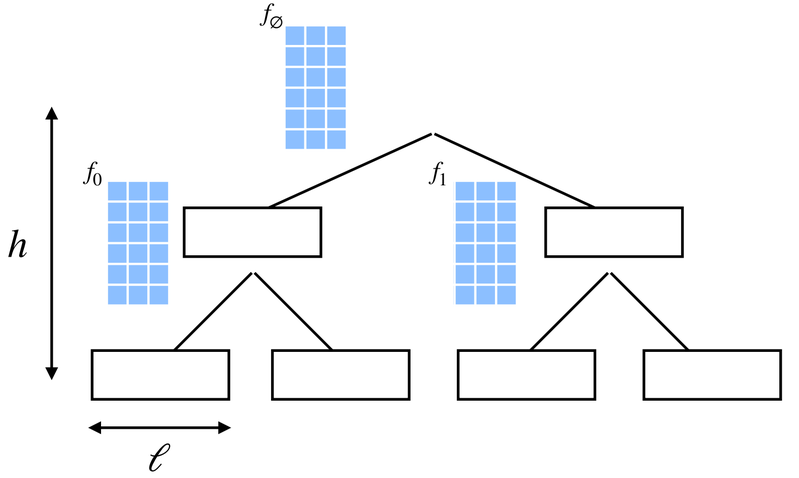
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$
- Communication cost: CC
- # of servers: S

TreeEval **One-Level Gadget**



- **Mask:** the existing contents of the catalytic tape
- $\log|\mathscr{R}| = O(\mathsf{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

TreeEval



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

- **Catalytic space:** CC

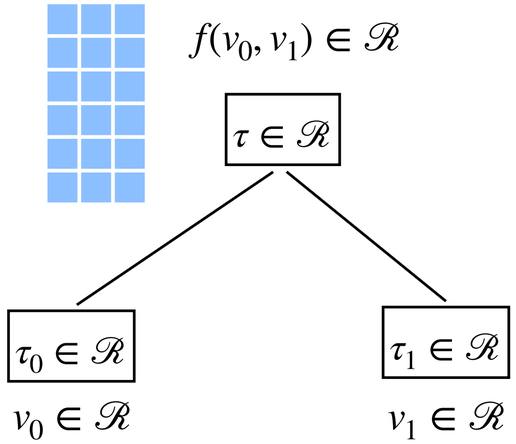
PIR

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

TreeEval **One-Level Gadget**

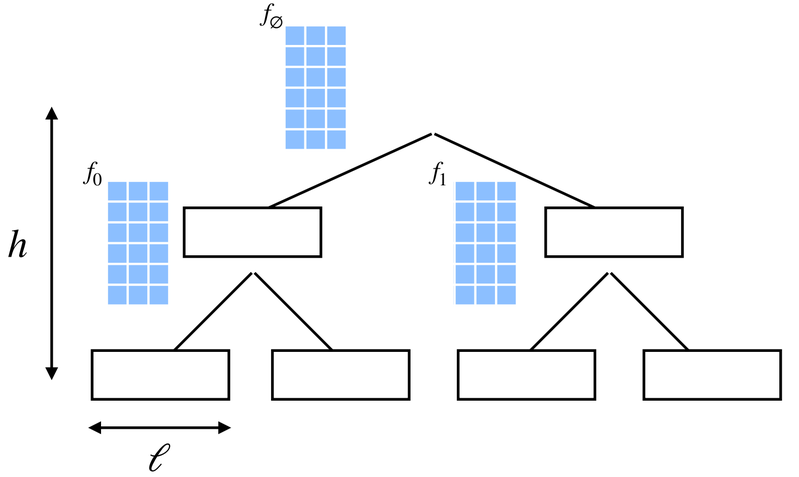
$\{0,1\}^{\ell} \subseteq \mathscr{R}$

$f : \mathscr{R} \times \mathscr{R} \to \mathscr{R}$



- **Mask:** the existing contents of the catalytic tape
- $\log|\mathscr{R}| = O(\mathsf{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

TreeEval



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

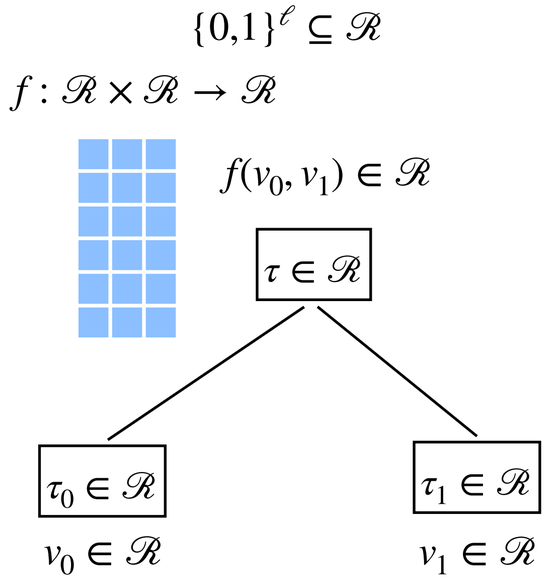
- **Catalytic space:** CC
- **Free space:**

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

PIR

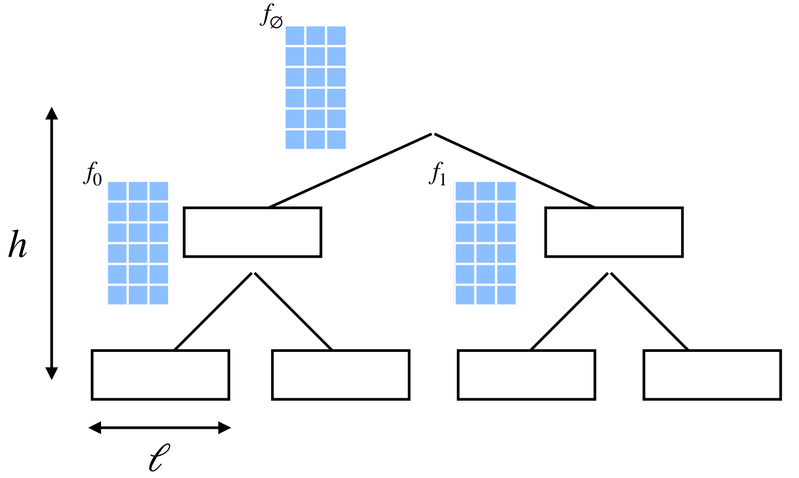
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$
- Communication cost: CC
- # of servers: S

TreeEval **One-Level Gadget**



- **Mask:** the existing contents of the catalytic tape
- $\log|\mathscr{R}| = O(\mathsf{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

TreeEval

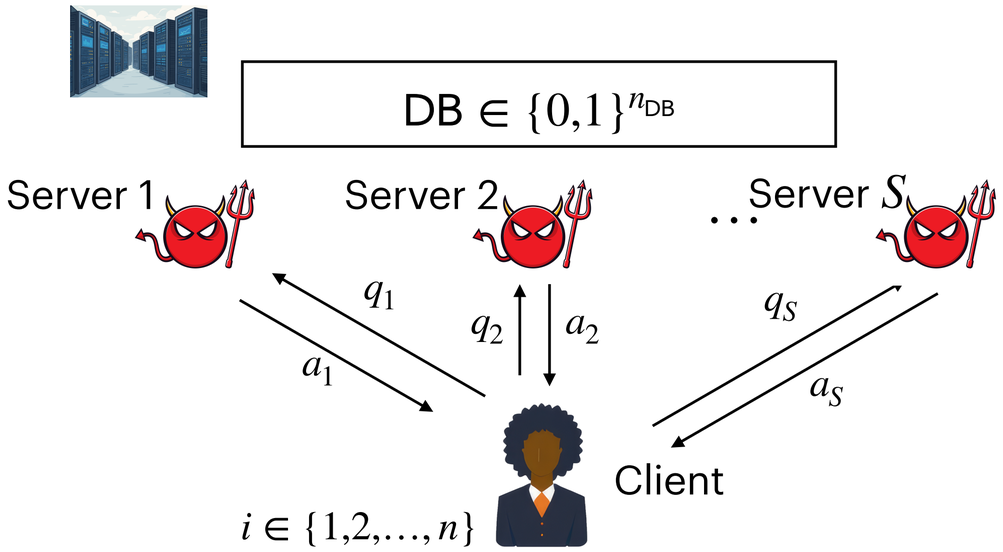


n = total input length = $\mathsf{poly}(2^{h+\ell})$

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

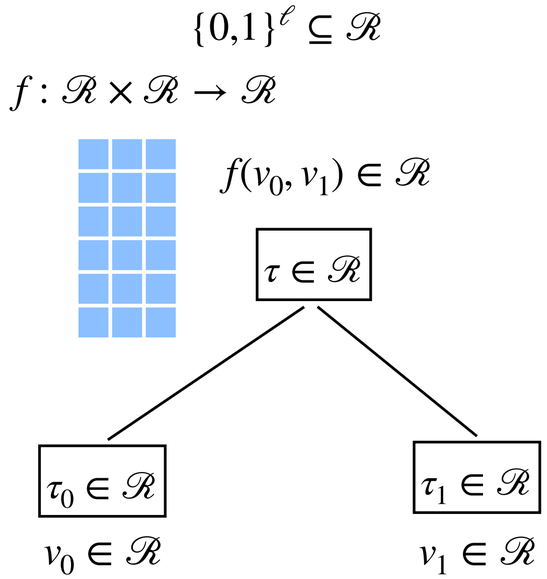
  $O(h \log S + \ell + \log \mathsf{CC})$
- **Runtime:** $\mathsf{poly}(S^h, 2^\ell)$

PIR



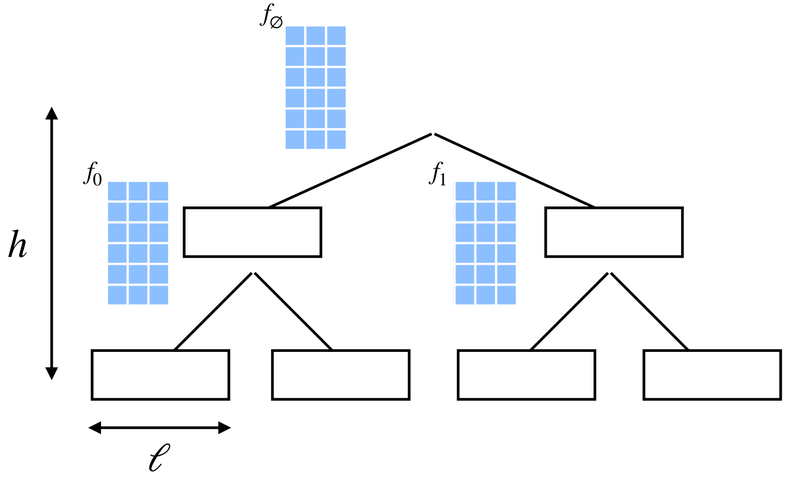
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$
- Communication cost: CC
- # of servers: S

TreeEval **One-Level Gadget**



- **Mask:** the existing contents of the catalytic tape
- $\log|\mathscr{R}| = O(\mathsf{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

TreeEval

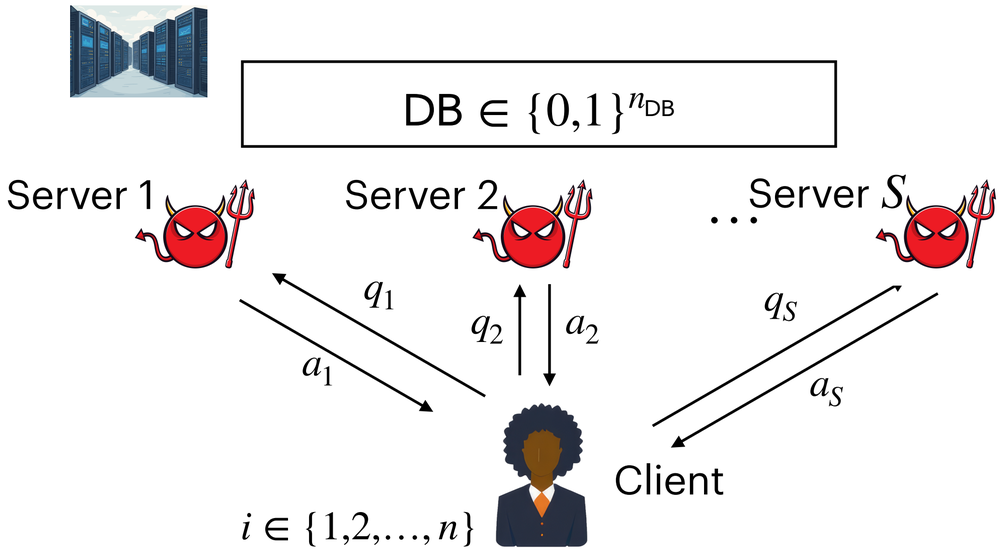


n = total input length = $\mathsf{poly}(2^{h+\ell})$

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

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

PIR

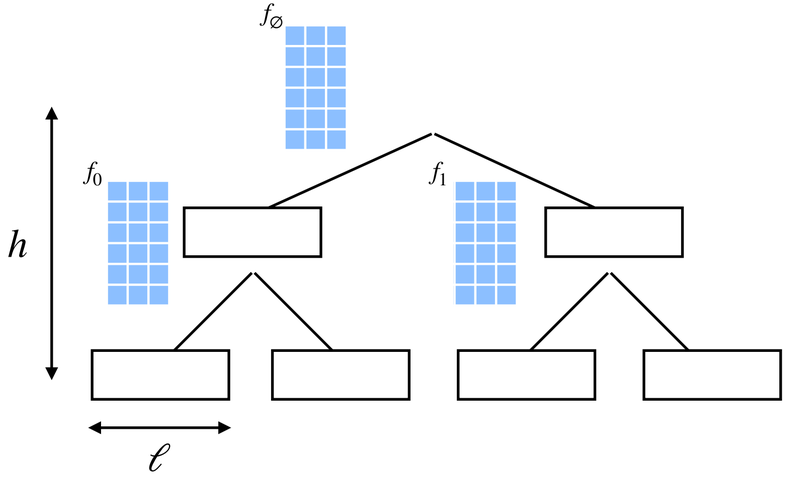


- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)}$
- Communication cost: CC
- # of servers: S

TreeEval One-Level Gadget

- **Mask:** the existing contents of the catalytic tape
- $\log |\mathcal{R}| = O(\mathrm{CC})$
- # of oracle calls: $O(S)$
- Free space: $O(\log S)$

TreeEval

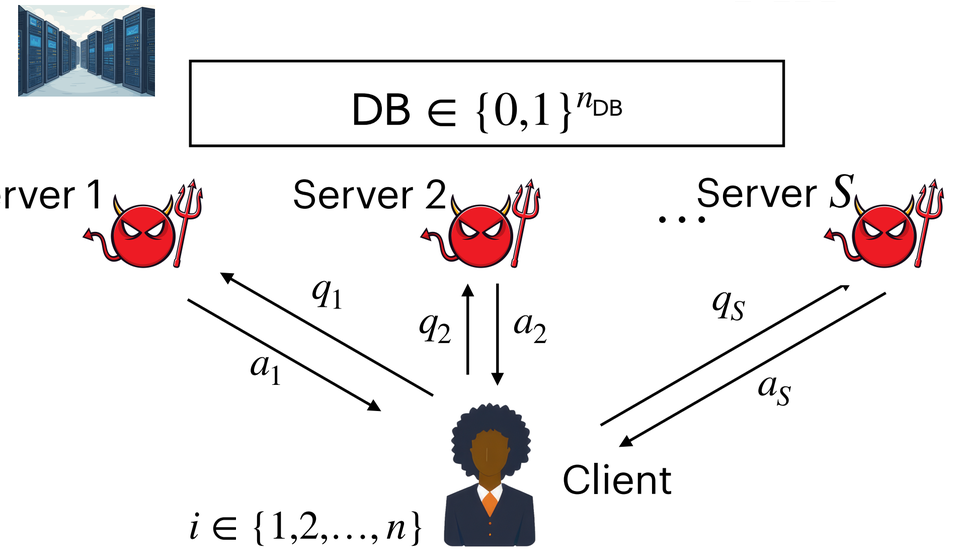


n = total input length = $\mathrm{poly}(2^{h+\ell})$

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

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

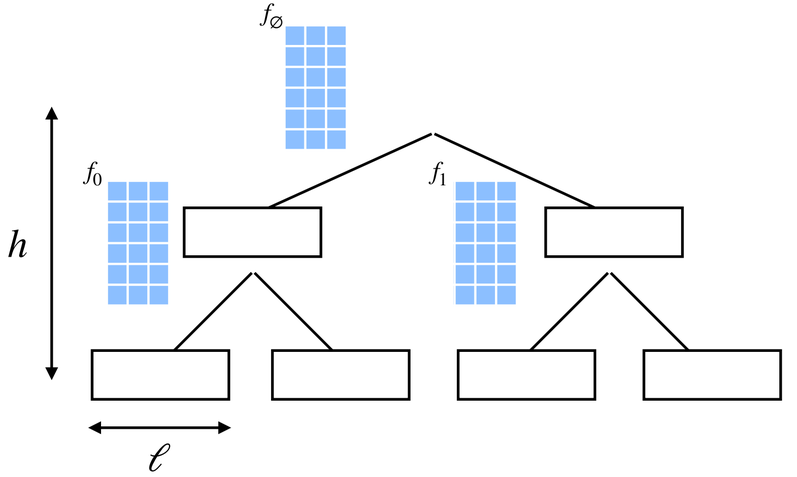
PIR



- **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

TreeEval

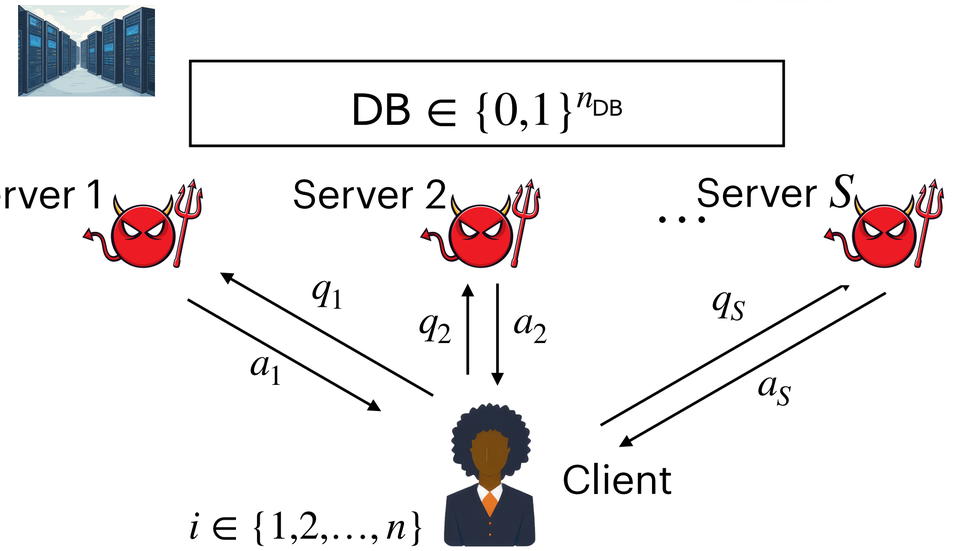


n = total input length = $\mathsf{poly}(2^{h+\ell})$

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

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

PIR

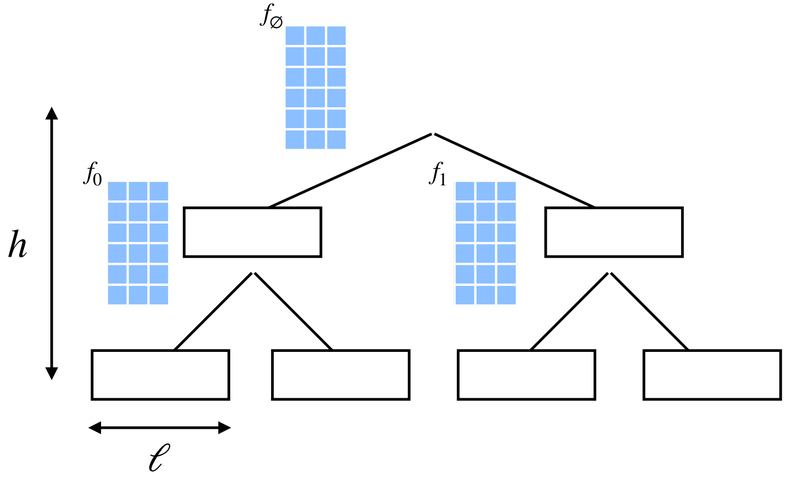


- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

Reed-Muller PIR → Cook-Mertz

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

TreeEval

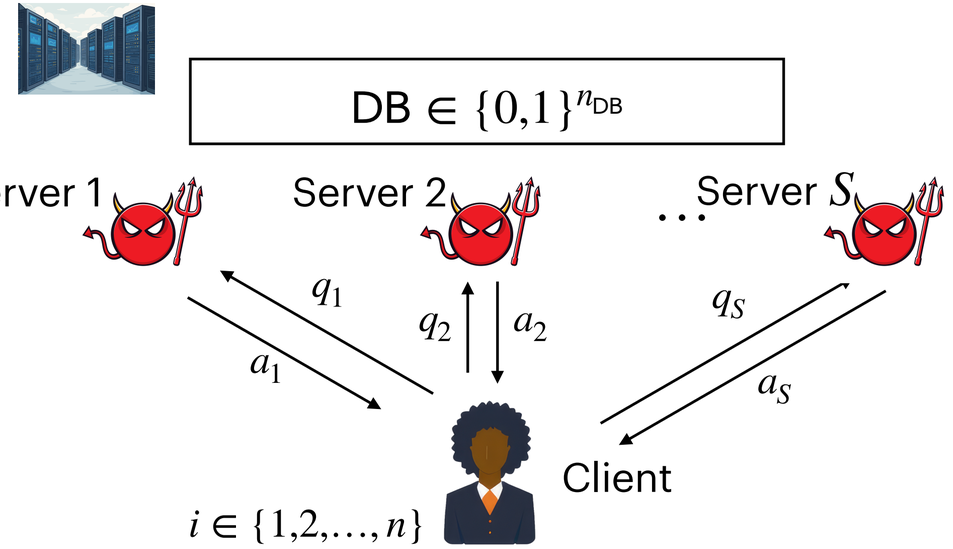


n = total input length = $\mathsf{poly}(2^{h+\ell})$

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

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

PIR

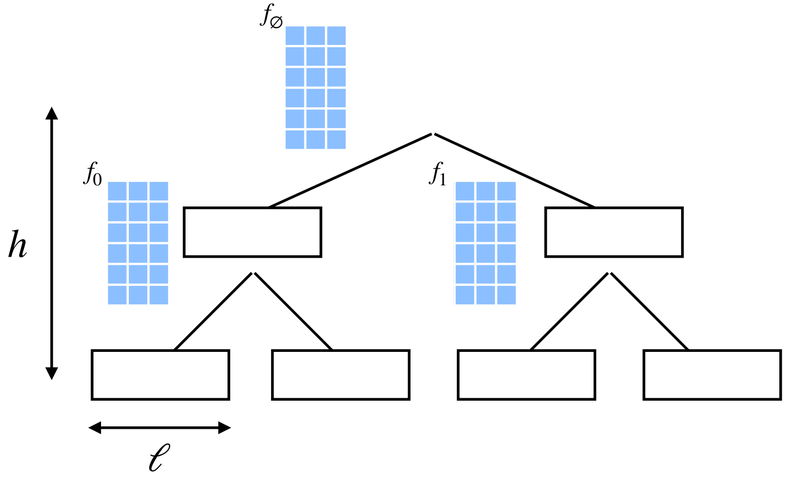


- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

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

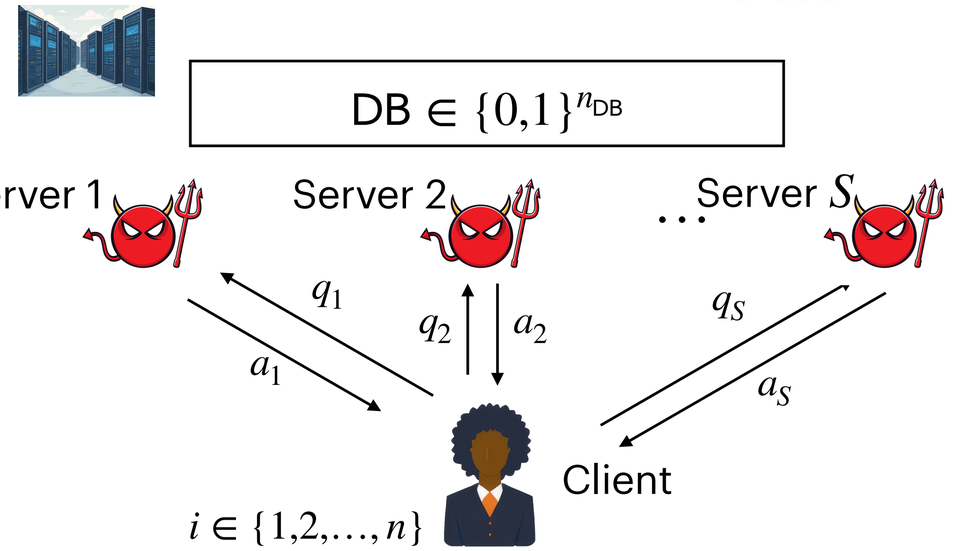


n = total input length = $\mathsf{poly}(2^{h+\ell})$

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

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

PIR



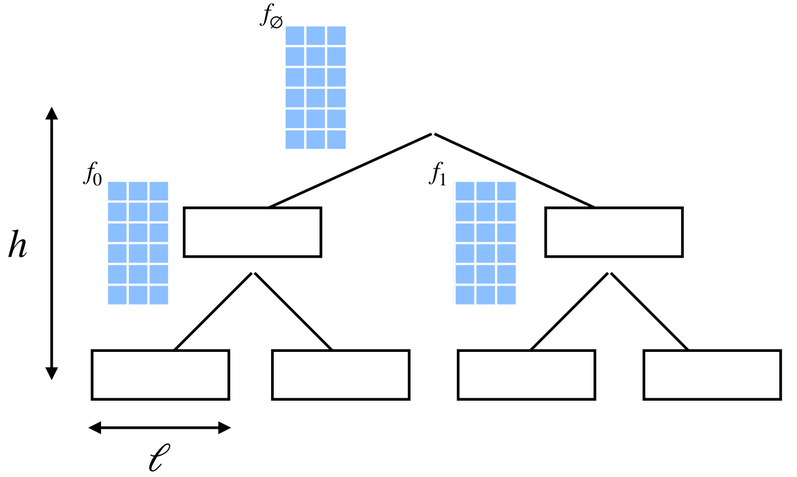
- **Mask:** sampled randomly by client to ensure privacy
- DB size: $n_{\mathsf{DB}} = 2^{O(\ell)} \sim n$
- Communication cost: CC
- # of servers: S

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



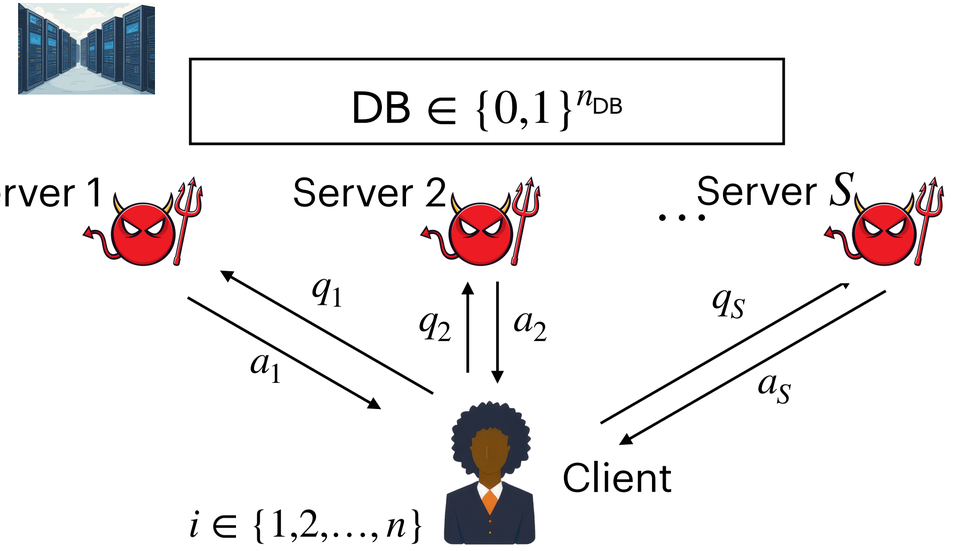
n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **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

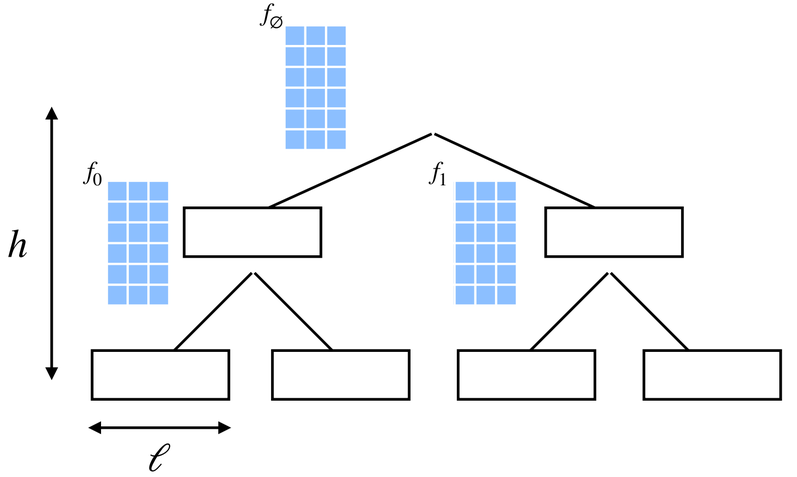
Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$ —*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



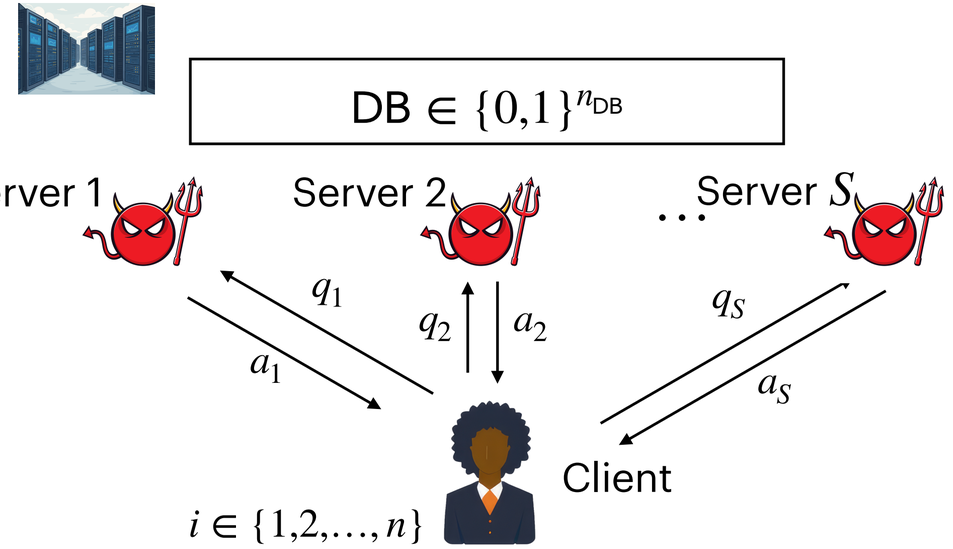
n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **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

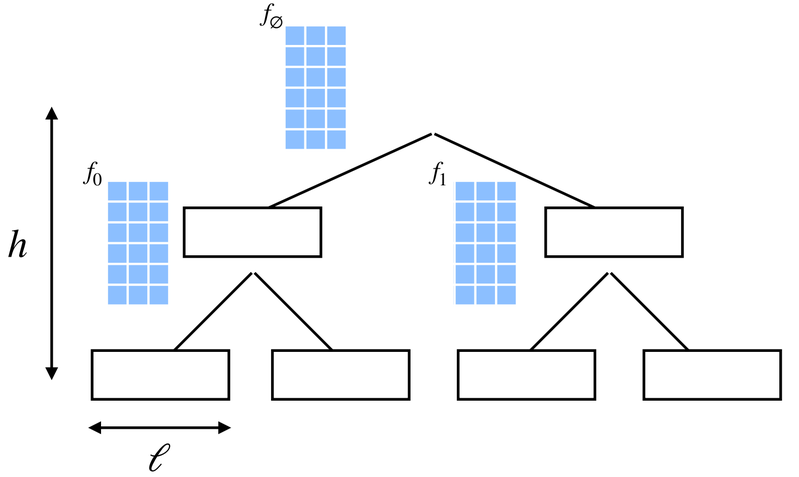
- $S = O(\log n_{\mathsf{DB}})$ —*Super-constant*→
- $\mathsf{CC} = O(\log n_{\mathsf{DB}} \log\log n_{\mathsf{DB}})$ —*Almost log!*→

- 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



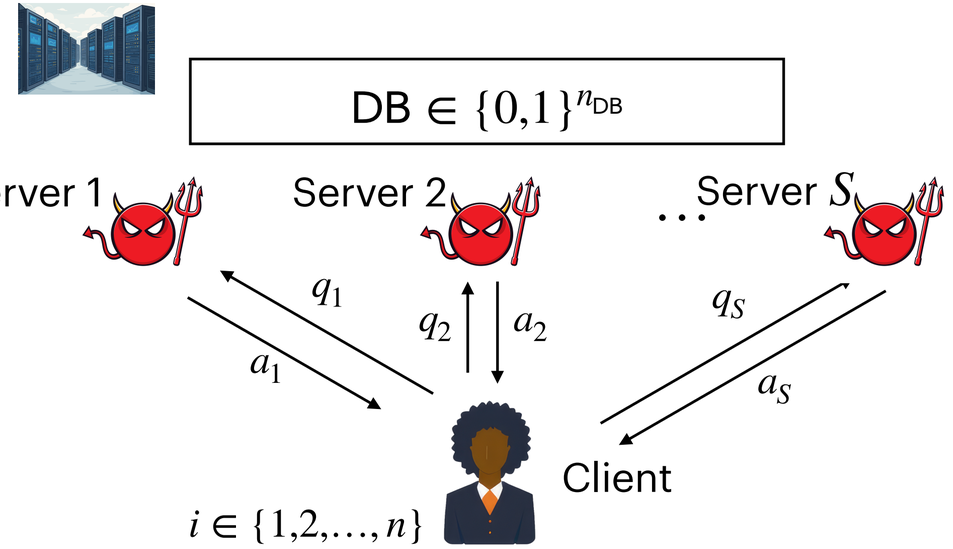
n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **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!*→

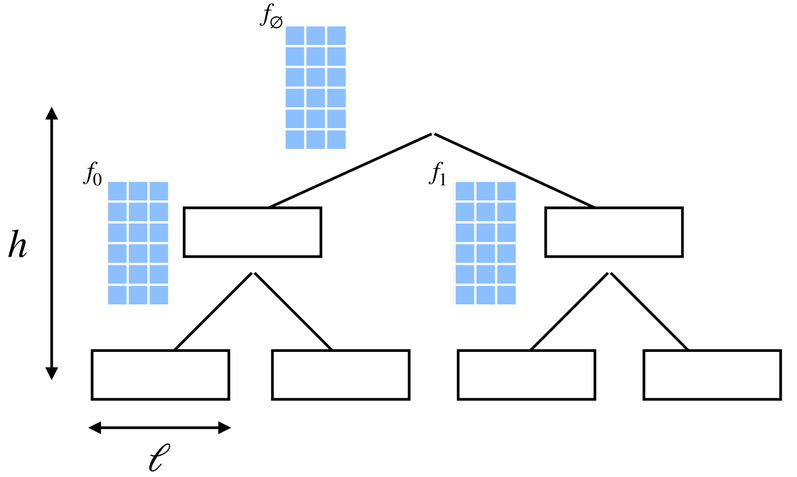
- Super-logarithmic free space, super-polynomial runtime 😭
- $O(\log n \log\log n)$ catalytic space 😄

Matching Vector PIR → Our Algorithm

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

- Logarithmic free space, polynomial runtime 😄

TreeEval



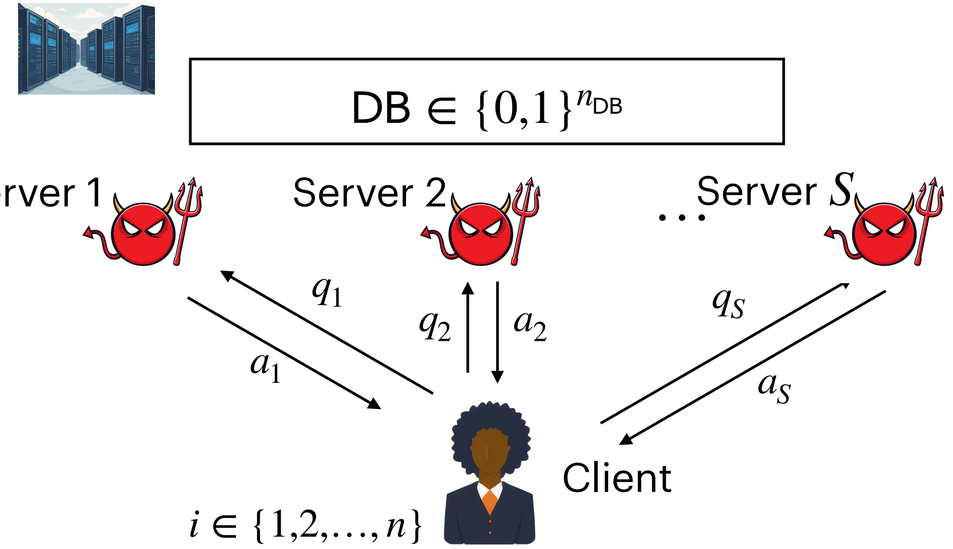
n = total input length = $\mathsf{poly}(2^{h+\ell})$

- **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

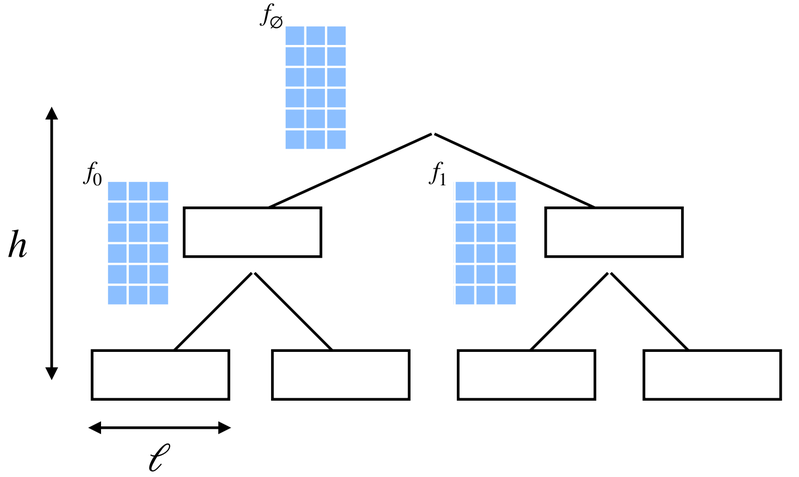
Reed-Muller PIR → Cook-Mertz

- $S = O(\log n_{\mathsf{DB}})$ — *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:** $\mathsf{poly}(S^h, 2^{\ell})$

Disclaimers

Disclaimers

- We do NOT get a generic compiler from PIR to TreeEval

Disclaimers

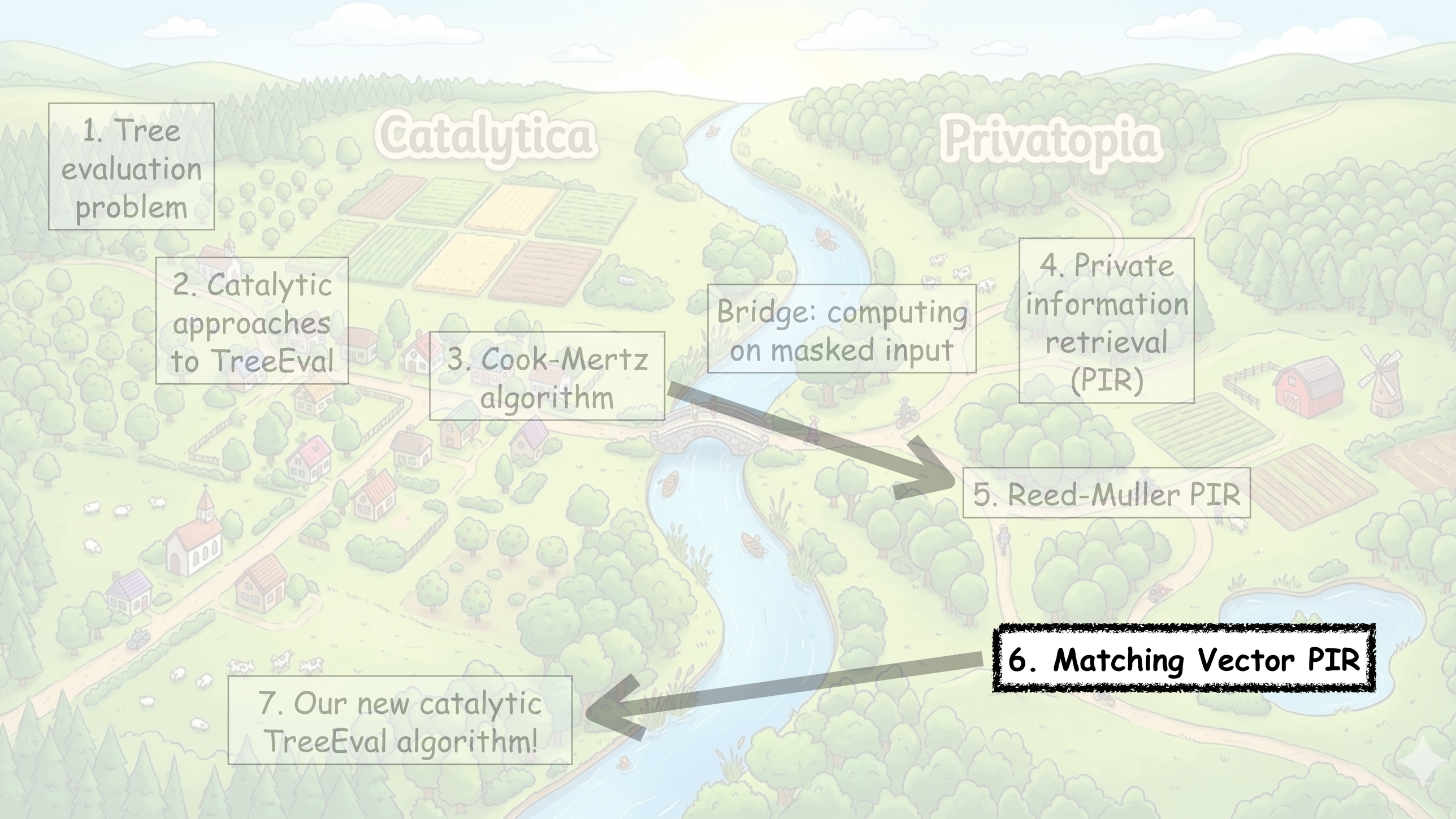
- We do NOT get a generic compiler from PIR to TreeEval
- Need the PIR protocol to have some special properties:
  - **Additive:** the client's and servers' computations are all fairly linear

Disclaimers

- We do NOT get a generic compiler from PIR to TreeEval
- Need the PIR protocol to have some special properties:
  - **Additive:** the client's and servers' computations are all fairly linear
  - **Concatenation-friendly:** client should be able to simulate queries for $a||b \in \{0,1\}^{2\ell}$ given individual queries for $a, b \in \{0,1\}^{\ell}$

Disclaimers

- We do NOT get a generic compiler from PIR to TreeEval
- Need the PIR protocol to have some special properties:
  - **Additive:** the client's and servers' computations are all fairly linear
  - **Concatenation-friendly:** client should be able to simulate queries for $a||b \in \{0,1\}^{2\ell}$ given individual queries for $a, b \in \{0,1\}^{\ell}$
- Connection is still fruitful for finding new approaches to TreeEval!


1. Tree evaluation problem
Catalytica
Privatopia
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

Matching-Vector PIR: A History

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

Construction	# of servers	CC	Uses matching vectors?
Before ‘08	S	$n^{\tilde{O}(1/S)}$	No

Matching-Vector PIR: A History

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

Construction	# of servers	CC	Uses matching vectors?
Before '08	S	$n^{\tilde{O}(1/S)}$	No
Yekhanin '08	3	$n^{O(1/\log\log n)}$	Yes

Matching-Vector PIR: A History

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

Construction	# of servers	CC	Uses matching vectors?
Before '08	S	$n^{\tilde{O}(1/S)}$	No
Yekhanin '08	3	$n^{O(1/\log\log n)}$	Yes
Efremenko '09	3	$2^{\tilde{O}(\sqrt{\log n})}$	

Matching-Vector PIR: A History

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

Construction	# of servers	CC	Uses matching vectors?
Before '08	S	$n^{\tilde{O}(1/S)}$	No
Yekhanin '08	3	$n^{O(1/\log\log n)}$	Yes
Efremenko '09	3	$2^{\tilde{O}(\sqrt{\log n})}$	
	2^r	$2^{\tilde{O}((\log n)^{1/r})}$	

Matching-Vector PIR: A History

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

Construction	# of servers	CC	Uses matching vectors?
Before '08	S	$n^{\tilde{O}(1/S)}$	No
Yekhanin '08	3	$n^{O(1/\log\log n)}$	Yes
Efremenko '09	3	$2^{\tilde{O}(\sqrt{\log n})}$	
	2^r	$2^{\tilde{O}((\log n)^{1/r})}$	
Dvir-Gopi '16	2^{r-1}	$2^{\tilde{O}((\log n)^{1/r})}$	

Matching-Vector PIR: A History

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

Construction	# of servers	CC	Uses matching vectors?
Before '08	S	$n^{\tilde{O}(1/S)}$	No
Yekhanin '08	3	$n^{O(1/\log\log n)}$	Yes
Efremenko '09	3	$2^{\tilde{O}(\sqrt{\log n})}$	
	2^r	$2^{\tilde{O}((\log n)^{1/r})}$	
Dvir-Gopi '16	2^{r-1}	$2^{\tilde{O}((\log n)^{1/r})}$	
Ghasemi-Kopparty-Sudan '25	3	$2^{\tilde{O}((\log n)^{1/3})}$	

Matching-Vector PIR: A History

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

Construction	# of servers	CC	Uses matching vectors?
Before '08	S	$n^{\tilde{O}(1/S)}$	No
Yekhanin '08	3	$n^{O(1/\log\log n)}$	Yes
Efremenko '09	3	$2^{\tilde{O}(\sqrt{\log n})}$	
	2^r	$2^{\tilde{O}((\log n)^{1/r})}$	
Dvir-Gopi '16	2^{r-1}	$2^{\tilde{O}((\log n)^{1/r})}$	
Ghasemi-Kopparty-Sudan '25	3	$2^{\tilde{O}((\log n)^{1/3})}$	

Today: 4 servers, CC $= 2^{\tilde{O}(\sqrt{\log n})}$

Matching Vectors

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{m} \Leftrightarrow i = j$

Matching Vectors

- Collection of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{Z}_m^d$ such that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{m} \Leftrightarrow i = j$
  - 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 $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{m} \Leftrightarrow i = j$
  - 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 $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{m} \Leftrightarrow i = j$
  - 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 $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{m} \Leftrightarrow i = j$
  - 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

Efremenko's 4-Server Construction

Setup

Efremenko's 4-Server Construction

Setup

- Let R be any commutative ring with elements g, h of respective multiplicative orders p, q such that $(g-1)(h-1) \neq 0$

Efremenko's 4-Server Construction

Setup

- Let R be any commutative ring with elements g, h of respective multiplicative orders p, q such that $(g-1)(h-1) \neq 0$
  - Example 1: $\mathbb{Z}_N$ where $N \equiv 1 \pmod{pq}$ is a prime

Efremenko's 4-Server Construction

Setup

- Let R be any commutative ring with elements g, h of respective multiplicative orders p, q such that $(g-1)(h-1) \neq 0$
  - Example 1: $\mathbb{Z}_N$ where $N \equiv 1 \pmod{pq}$ is a prime
  - Example 2: $\mathbb{F}[X]/(X^{pq}-1)$ for some field $\mathbb{F}$

Efremenko's 4-Server Construction

Setup

- Let R be any commutative ring with elements g, h of respective multiplicative orders p, q such that $(g-1)(h-1) \neq 0$
  - Example 1: $\mathbb{Z}_N$ where $N \equiv 1 \pmod{pq}$ is a prime
  - Example 2: $\mathbb{F}[X]/(X^{pq}-1)$ for some field $\mathbb{F}$
- Let $\mathsf{CRT} : \mathbb{Z}_p \times \mathbb{Z}_q \to \mathbb{Z}_{m=pq}$ be the Chinese remainder homomorphism

Efremenko's 4-Server Construction

Setup

- Let R be any commutative ring with elements g, h of respective multiplicative orders p, q such that $(g-1)(h-1) \neq 0$
  - Example 1: $\mathbb{Z}_N$ where $N \equiv 1 \pmod{pq}$ is a prime
  - Example 2: $\mathbb{F}[X]/(X^{pq}-1)$ for some field $\mathbb{F}$
- Let $\mathsf{CRT} : \mathbb{Z}_p \times \mathbb{Z}_q \to \mathbb{Z}_{m=pq}$ be the Chinese remainder homomorphism
- We will label the servers as "00", "01", "10", and "11"

Efremenko's 4-Server Construction

Queries and Answers

Efremenko's 4-Server Construction

Queries and Answers

- Client (has an index $i \in [n]$): samples a **random mask r** $\leftarrow \mathbb{Z}_m^d$. Queries:

Efremenko's 4-Server Construction

Queries and Answers

- Client (has an index $i \in [n]$): samples a **random mask $\mathbf{r} \leftarrow \mathbb{Z}_m^d$**. Queries:
  - Server "00": $\mathbf{r}$
  - Server "11": $\mathbf{r} + \mathbf{u}_i$

Efremenko's 4-Server Construction

Queries and Answers

- Client (has an index $i \in [n]$): samples a **random mask $\mathbf{r} \leftarrow \mathbb{Z}_m^d$**. Queries:
  - Server "00": $\mathbf{r}$
  - Server "11": $\mathbf{r} + \mathbf{u}_i$
  - Server "01": $\mathsf{CRT}(\mathbf{r} \bmod p, \mathbf{r} + \mathbf{u}_i \bmod q)$

Efremenko's 4-Server Construction

Queries and Answers

- Client (has an index $i \in [n]$): samples a **random mask $\mathbf{r} \leftarrow \mathbb{Z}_m^d$**. Queries:
  - Server "00": $\mathbf{r}$
  - Server "11": $\mathbf{r} + \mathbf{u}_i$
  - Server "01": $\mathsf{CRT}(\mathbf{r} \bmod p, \mathbf{r} + \mathbf{u}_i \bmod q)$
  - Server "10": $\mathsf{CRT}(\mathbf{r} + \mathbf{u}_i \bmod p, \mathbf{r} \bmod q)$

Efremenko's 4-Server Construction

Queries and Answers

- Client (has an index $i \in [n]$): samples a **random mask $\mathbf{r} \leftarrow \mathbb{Z}_m^d$**. Queries:
  - Server "00": $\mathbf{r}$
  - Server "11": $\mathbf{r} + \mathbf{u}_i$
  - Server "01": $\mathsf{CRT}(\mathbf{r} \bmod p, \mathbf{r} + \mathbf{u}_i \bmod q)$
  - Server "10": $\mathsf{CRT}(\mathbf{r} + \mathbf{u}_i \bmod p, \mathbf{r} \bmod q)$
- Each server, given its query $\mathbf{v} \in \mathbb{Z}_m^d$: replies with

$$\sum_{j=1}^{n} \mathsf{DB}_j \cdot g^{\langle \mathbf{v}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{v}, \mathbf{u}_j \rangle} \in R$$

Efremenko's 4-Server Construction

How the Client Reconstructs DB_i

Cheatsheet

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p \text{ and } q}$$

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{pq} \Leftrightarrow i = j$$

Efremenko's 4-Server Construction

How the Client Reconstructs DB_i

- Client computation: [00's answer] - [01's answer] - [10's answer] + [11's answer]

Cheatsheet

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p \text{ and } q}$$

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{pq} \Leftrightarrow i = j$$

Efremenko's 4-Server Construction

How the Client Reconstructs DB_i

- Client computation: [00's answer] - [01's answer] - [10's answer] + [11's answer]
- Gives:

$$\sum_{j=1}^{n} \mathsf{DB}_j \cdot \left[g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} + g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \right]$$

Cheatsheet

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p \text{ and } q}$$

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{pq} \Leftrightarrow i = j$$

Efremenko's 4-Server Construction

How the Client Reconstructs DB_i

- Client computation: [00's answer] - [01's answer] - [10's answer] + [11's answer]
- Gives:

$$\sum_{j=1}^{n} \mathsf{DB}_j \cdot \left[g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} + g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \right]$$

$$= \sum_{j=1}^{n} \mathsf{DB}_j \cdot g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot \left[\left(1 - g^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \cdot \left(1 - h^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \right]$$

Cheatsheet

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p \text{ and } q}$$

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{pq} \Leftrightarrow i = j$$

Efremenko's 4-Server Construction

How the Client Reconstructs DB_i

- Client computation: [00's answer] - [01's answer] - [10's answer] + [11's answer]
- Gives:

$$\sum_{j=1}^{n} \mathsf{DB}_j \cdot \left[g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} + g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \right]$$

$$= \sum_{j=1}^{n} \mathsf{DB}_j \cdot g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot \left[\left(1 - g^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \cdot \left(1 - h^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \right]$$

  - If $j \neq i$, then either $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{p} \Rightarrow g^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle} = 1$, or similarly $h^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle} = 1$

Cheatsheet

$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p \text{ and } q}$

$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{pq} \Leftrightarrow i = j$

Efremenko's 4-Server Construction

How the Client Reconstructs DB_i

- Client computation: [00's answer] - [01's answer] - [10's answer] + [11's answer]
- Gives:

$$\sum_{j=1}^{n} \mathsf{DB}_j \cdot \left[g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} + g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \right]$$

$$= \sum_{j=1}^{n} \mathsf{DB}_j \cdot g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot \left[\left(1 - g^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \cdot \left(1 - h^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \right]$$

- If $j \neq i$, then either $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 0 \pmod{p} \Rightarrow g^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle} = 1$, or similarly $h^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle} = 1$
- If $j = i$, then $g^{\langle \mathbf{u}_i, \mathbf{u}_i \rangle} = g$ and $h^{\langle \mathbf{u}_i, \mathbf{u}_i \rangle} = h$

Cheatsheet

$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p \text{ and } q}$

$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{pq} \Leftrightarrow i = j$

Efremenko's 4-Server Construction

How the Client Reconstructs DB_i

- Client computation: [00's answer] - [01's answer] - [10's answer] + [11's answer]
- Gives:

$$\sum_{j=1}^{n} \mathsf{DB}_j \cdot \left[g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} - g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} + g^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}+\mathbf{u}_i, \mathbf{u}_j \rangle} \right]$$

$$= \sum_{j=1}^{n} \mathsf{DB}_j \cdot g^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_j \rangle} \cdot \left[\left(1 - g^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \cdot \left(1 - h^{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}\right) \right]$$

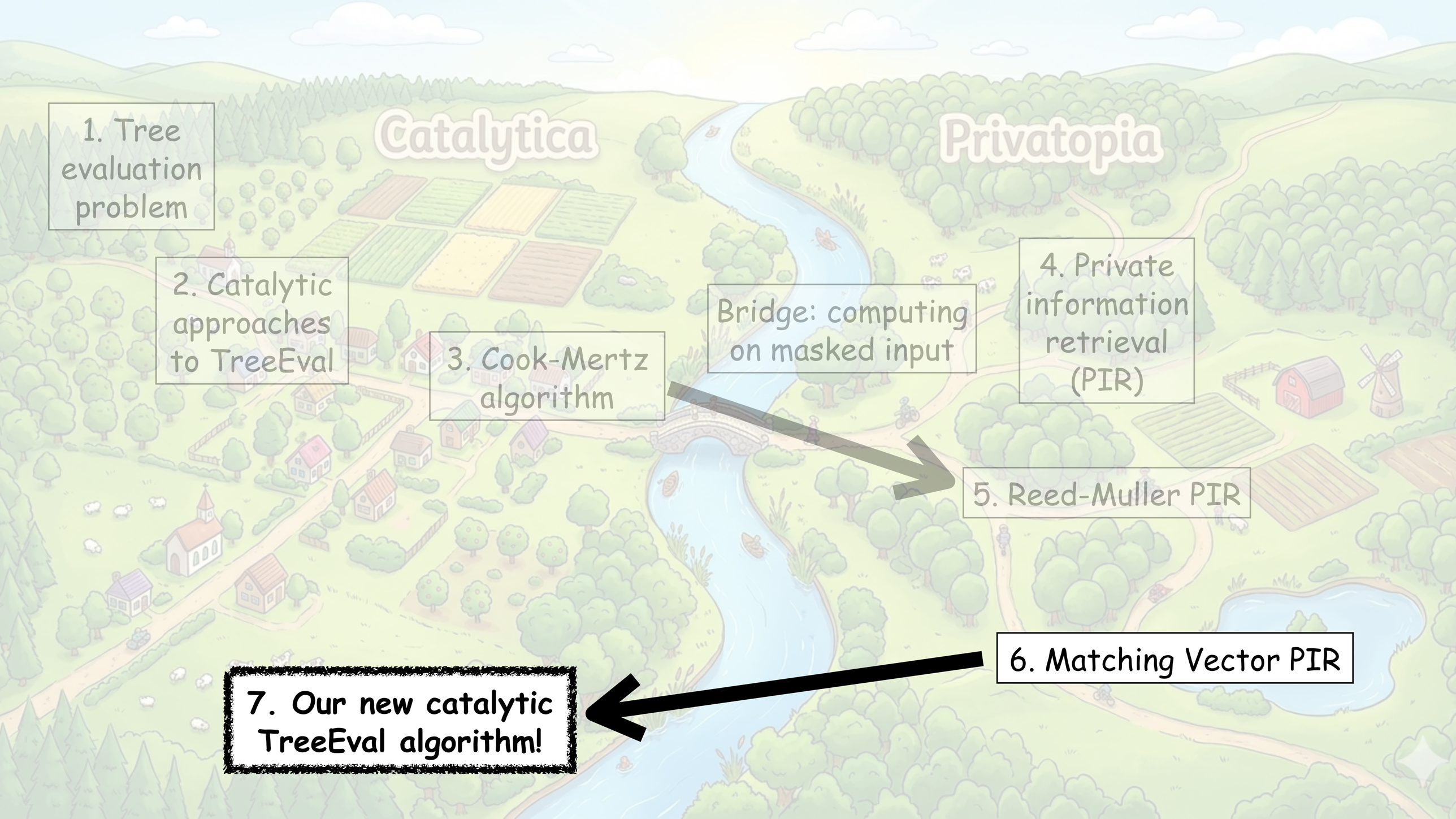
- Final result:

$$\mathsf{DB}_i \cdot \underbrace{g^{\langle \mathbf{r}, \mathbf{u}_i \rangle} \cdot h^{\langle \mathbf{r}, \mathbf{u}_i \rangle} \cdot (1-g)(1-h)}_{\text{nonzero}} \rightarrow \mathsf{DB}_i \in \{0,1\}$$ 🎉

Cheatsheet

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \in \{0,1\} \pmod{p \text{ and } q}$$

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv 1 \pmod{pq} \Leftrightarrow i = j$$


Catalytica
Privatopia
1. Tree evaluation problem
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

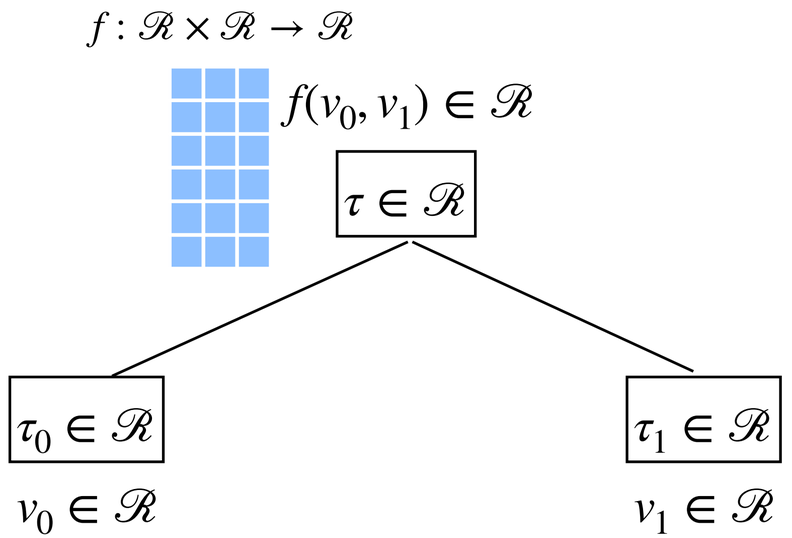
Refresher: The One-Level Gadget

- 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 $\mathcal{O}(b, c \in \{0,1\})$: update

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

- 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

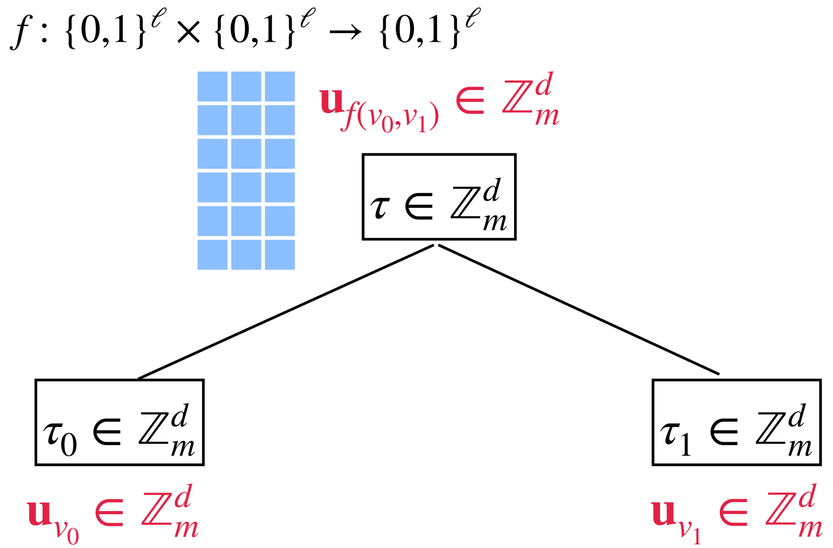
Our One-Level Gadget Using Matching Vectors

- Let $\{\mathbf{u}_v \in \mathbb{Z}_m^d\}_{v \in \{0,1\}^\ell}$ be a MV family
- Ring $\mathscr{R} = \mathbb{Z}_m^d$
- Inputs:
  - Control bit $c' \in \{0,1\}$
  - Arbitrary catalytic registers $\tau_0, \tau_1, \tau \in \mathscr{R}$
  - Oracle $\mathcal{O}(b, c \in \{0,1\})$: update

$$\tau_b \leftarrow \tau_b + (-1)^c \cdot \mathbf{u}_{v_b}$$

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

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



Efficiency Analysis

Efficiency Analysis

- CC bottleneck: a vector in $\mathbb{Z}_m^d \to O(d) = 2^{\tilde{O}(\sqrt{\log n})}$

Efficiency Analysis

- CC bottleneck: a vector in $\mathbb{Z}_m^d \to O(d) = 2^{\tilde{O}(\sqrt{\log n})}$
- Our resulting TreeEval algorithm:
  - Catalytic space: $O(\mathsf{CC}) = 2^{\tilde{O}(\sqrt{\log n})}$

Efficiency Analysis

- CC bottleneck: a vector in $\mathbb{Z}_m^d \to O(d) = 2^{\tilde{O}(\sqrt{\log n})}$
- Our resulting TreeEval algorithm:
  - Catalytic space: $O(\mathsf{CC}) = 2^{\tilde{O}(\sqrt{\log n})}$
  - Free space: $O(h \log S + \ell + \log \mathsf{CC}) = O(\log n)$

Efficiency Analysis

- CC bottleneck: a vector in $\mathbb{Z}_m^d \to O(d) = 2^{\tilde{O}(\sqrt{\log n})}$
- Our resulting TreeEval algorithm:
  - Catalytic space: $O(\mathsf{CC}) = 2^{\tilde{O}(\sqrt{\log n})}$
  - Free space: $O(h \log S + \ell + \log \mathsf{CC}) = O(\log n)$
  - Runtime: $\mathsf{poly}(S^h, 2^\ell) = \mathsf{poly}(2^{h+\ell}) = \mathsf{poly}(n)$

Efficiency Analysis

- CC bottleneck: a vector in $\mathbb{Z}_m^d \to O(d) = 2^{\tilde{O}(\sqrt{\log n})}$
- Our resulting TreeEval algorithm:
  - Catalytic space: $O(\mathsf{CC}) = 2^{\tilde{O}(\sqrt{\log n})}$
  - Free space: $O(h \log S + \ell + \log \mathsf{CC}) = O(\log n)$
  - Runtime: $\mathsf{poly}(S^h, 2^\ell) = \mathsf{poly}(2^{h+\ell}) = \mathsf{poly}(n)$
- Generalisation to 2^t servers: use t primes instead of two

Efficiency Analysis

- CC bottleneck: a vector in $\mathbb{Z}_m^d \to O(d) = 2^{\tilde{O}(\sqrt{\log n})}$
- Our resulting TreeEval algorithm:
  - Catalytic space: $O(\mathsf{CC}) = 2^{\tilde{O}(\sqrt{\log n})}$
  - Free space: $O(h \log S + \ell + \log \mathsf{CC}) = O(\log n)$
  - Runtime: $\mathsf{poly}(S^h, 2^\ell) = \mathsf{poly}(2^{h+\ell}) = \mathsf{poly}(n)$
- Generalisation to 2^t servers: use t primes instead of two
  - Gives $2^{\log^\epsilon n}$ catalytic space, $O_\epsilon(\log n)$ free space, and $\mathsf{poly}_\epsilon(n)$ runtime

Conclusion

Cook-Mertz used Lagrange interpolation to evaluate a polynomial, given access only to **maskings of the input by the catalytic tape's contents.**

Cook-Mertz used Lagrange interpolation to evaluate a polynomial, given access only to **maskings of the input by the catalytic tape's contents.**

Private information retrieval (PIR) also computes on masked inputs, but to ensure privacy.

Cook-Mertz used Lagrange interpolation to evaluate a polynomial, given access only to **maskings of the input by the catalytic tape's contents.**

Private information retrieval (PIR) also computes on masked inputs, but to ensure privacy.

In fact, Cook-Mertz is secretly running a Reed-Muller PIR!

Cook-Mertz used Lagrange interpolation to evaluate a polynomial, given access only to **maskings of the input by the catalytic tape's contents.**

Private information retrieval (PIR) also computes on masked inputs, but to ensure privacy.

In fact, Cook-Mertz is secretly running a Reed-Muller PIR!

We leverage this connection and use matching-vector PIR to get new algorithms for TreeEval.

Open Questions on TreeEval

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?
- Candidate approach: find better matching vector constructions!

Open Questions on TreeEval

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

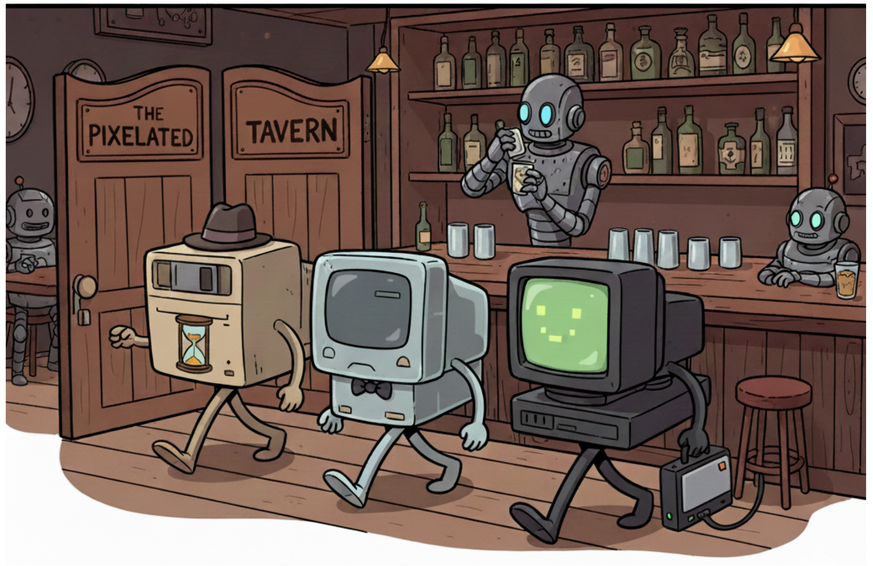
- Is TreeEval $\in$ L?
  - More modestly: TreeEval in simultaneous poly(n) time and $O(\log^{1.99} n)$ space?
- Candidate approach: find better matching vector constructions!
  - Would push down our catalytic space, our main bottleneck
  - Best lower bound is $d \geq O(\log n \log\log n)$. If achievable: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

Three Turing Machines Walk Into a Bar

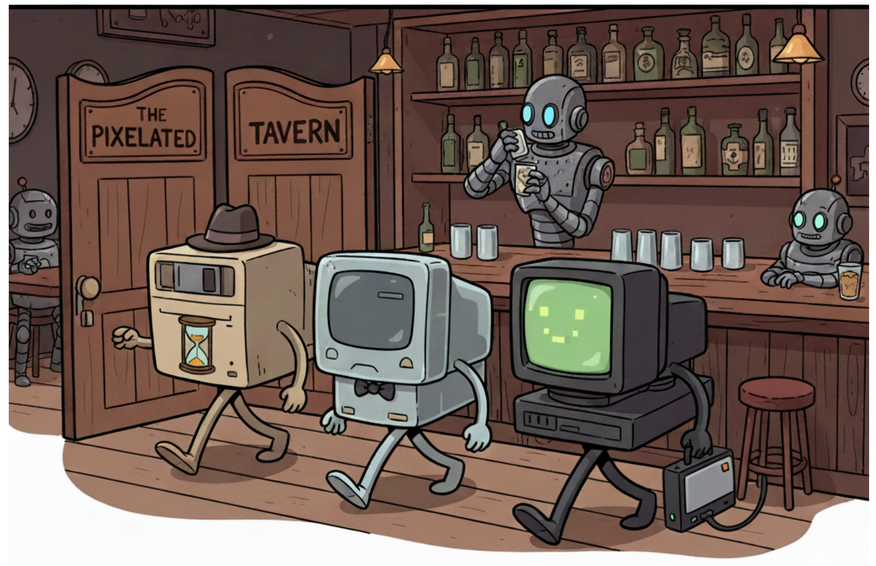
[Williams, STOC 2025]



Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

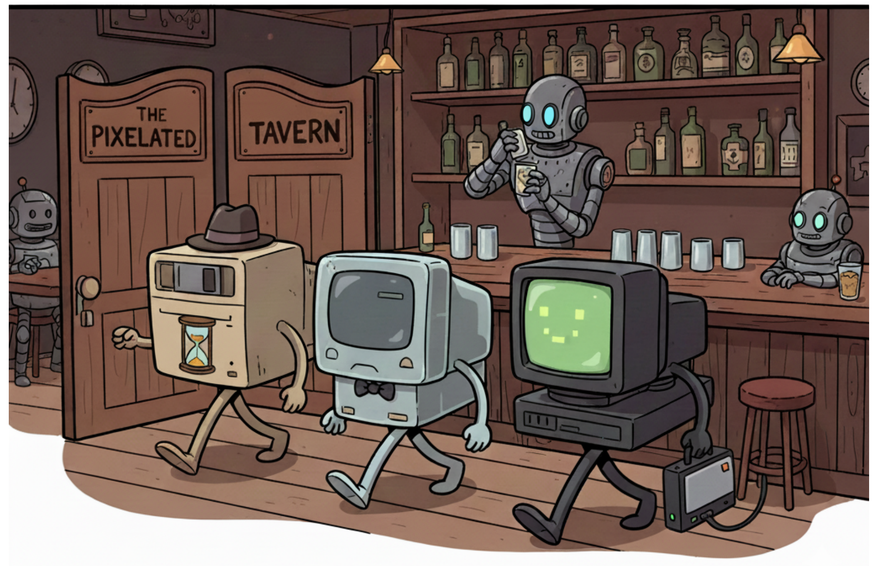
- Computer A: has to run in time $\leq T$, but has unlimited memory



Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

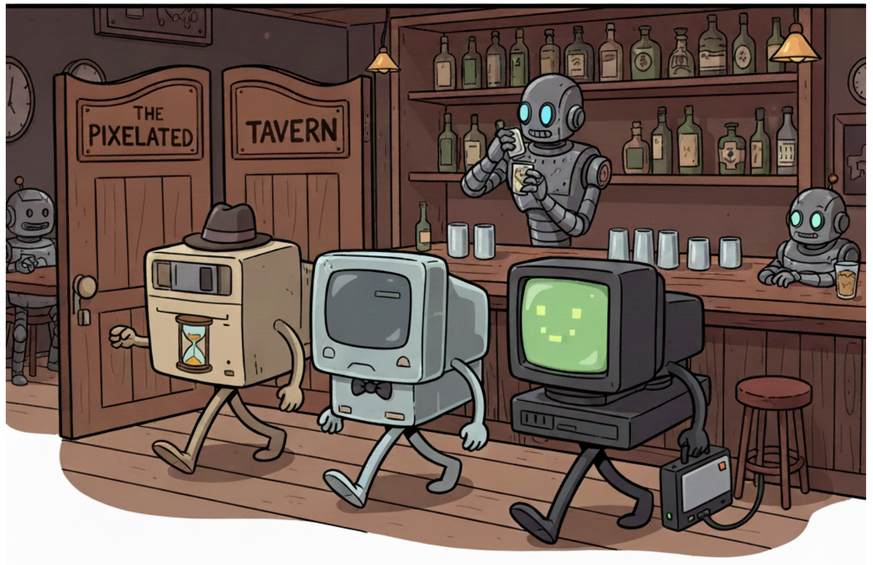
- Computer A: has to run in time $\leq T$, but has unlimited memory
- Computer B: has memory $\sqrt{T \log T}$, but can run in unlimited time



Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

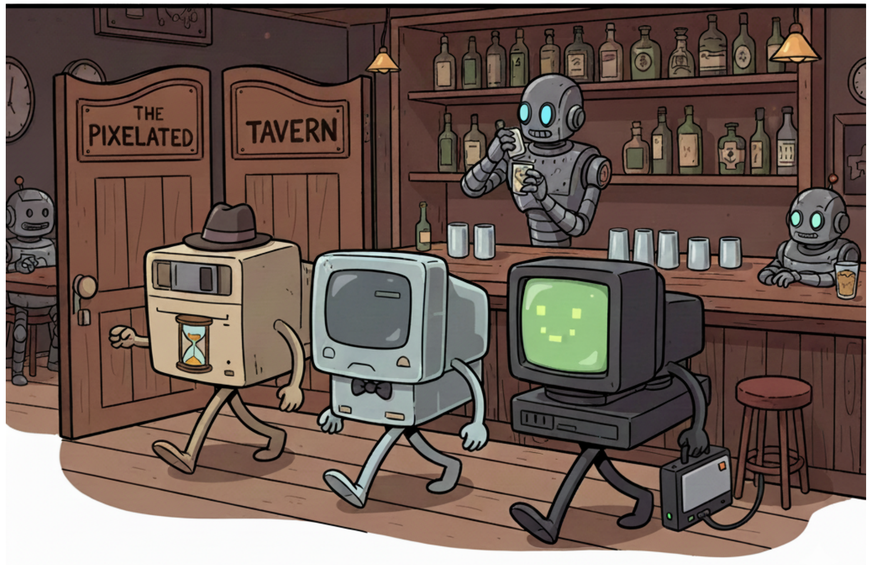
- Computer A: has to run in time $\leq T$, but has unlimited memory
- Computer B: has memory $\sqrt{T \log T}$, but can run in unlimited time
- Computer C: has memory $\sqrt{T}$, unlimited time, and a full catalytic hard drive of size $2^{T^{0.001}}$



Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

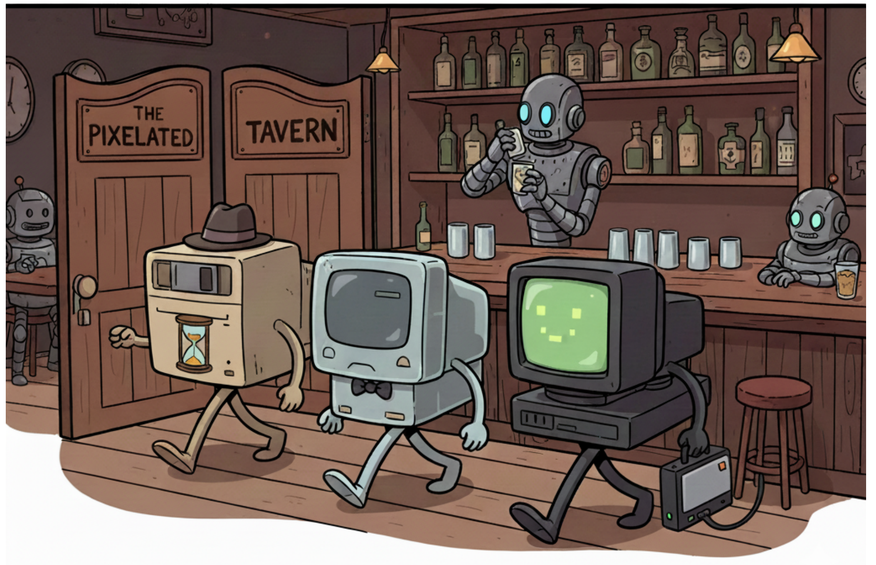
- Computer A: has to run in time $\leq T$, but has unlimited memory
- Computer B: has memory $\sqrt{T \log T}$, but can run in unlimited time
- Computer C: has memory $\sqrt{T}$, unlimited time, and a full catalytic hard drive of size $2^{T^{0.001}}$
  - Can temporarily modify the hard drive's contents
  - Must restore to the original state at the end!



Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

- Computer A: has to run in time $\leq T$, but has unlimited memory
- Computer B: has memory $\sqrt{T \log T}$, but can run in unlimited time
- Computer C: has memory $\sqrt{T}$, unlimited time, and a full catalytic hard drive of size $2^{T^{0.001}}$
  - Can temporarily modify the hard drive's contents
  - Must restore to the original state at the end!

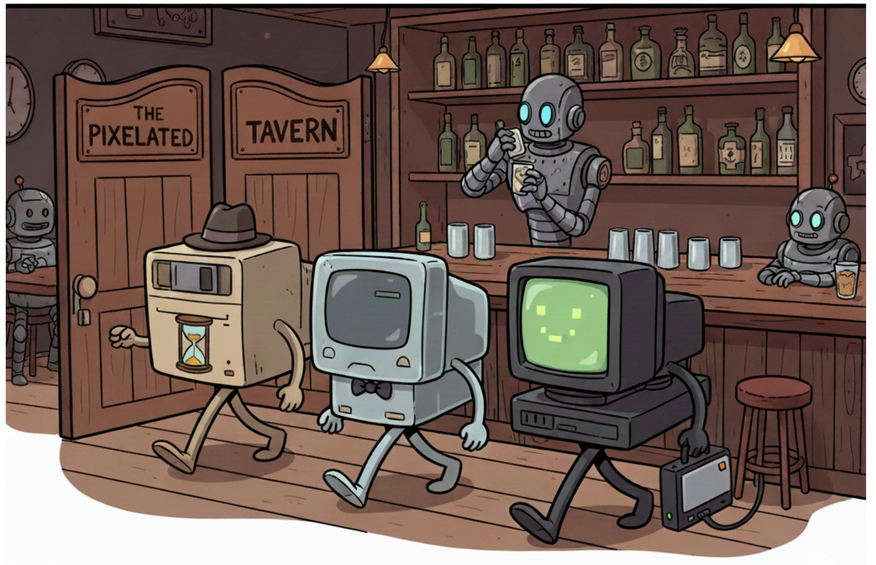


Q: How do the powers of these computers compare?

Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

- Computer A: has to run in time $\leq T$, but has unlimited memory
- Computer B: has memory $\sqrt{T \log T}$, but can run in unlimited time
- Computer C: has memory $\sqrt{T}$, unlimited time, and a full catalytic hard drive of size $2^{T^{0.001}}$
  - Can temporarily modify the hard drive's contents
  - Must restore to the original state at the end!

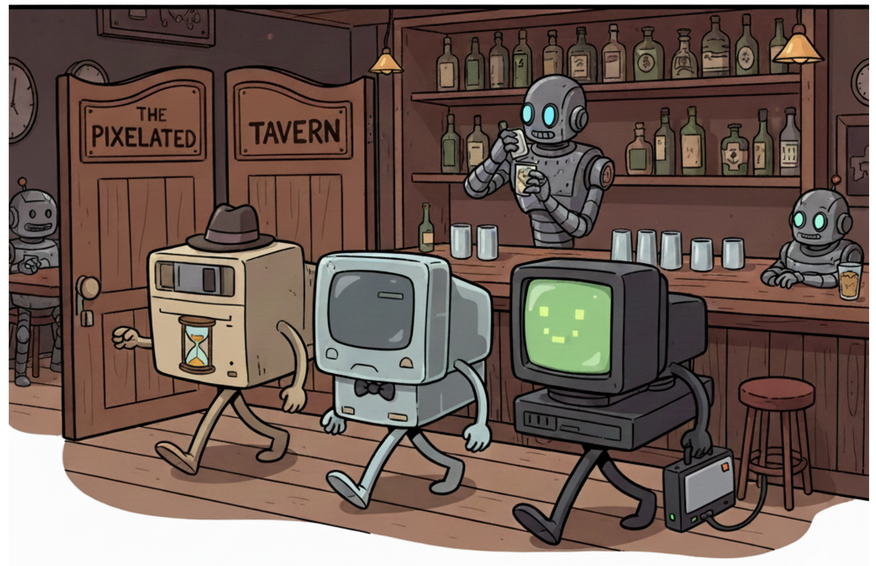


Q: How do the powers of these computers compare?

Is one of them "strictly weakest" i.e. the other two computers could do absolutely anything that it could?

Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

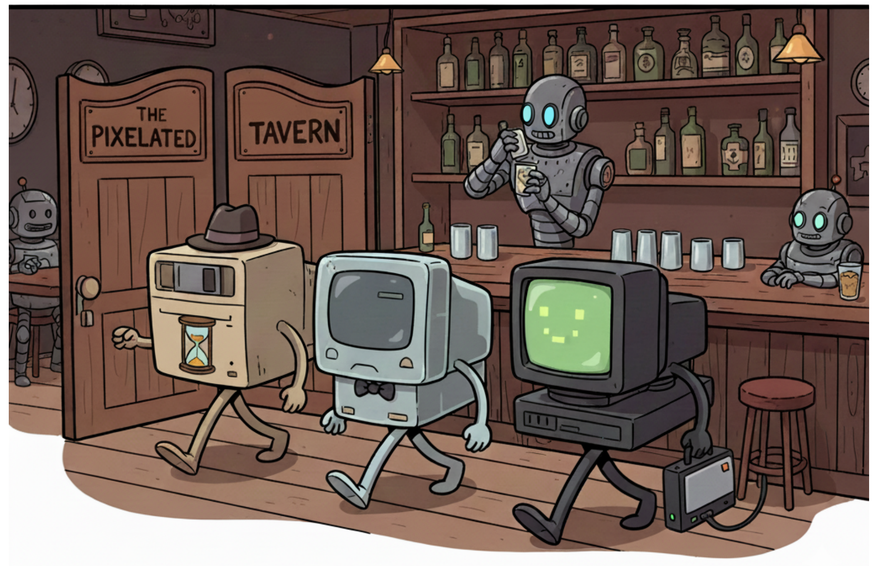


- Computer A: time $\leq T$, unlimited memory
- Computer B: memory $\sqrt{T \log T}$, unlimited time
- Computer C: memory $\sqrt{T}$, unlimited time, full hard drive of size $2^{T^{0.001}}$

Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

- Folklore: if Computer B had memory T, then it would be more powerful than Computer A

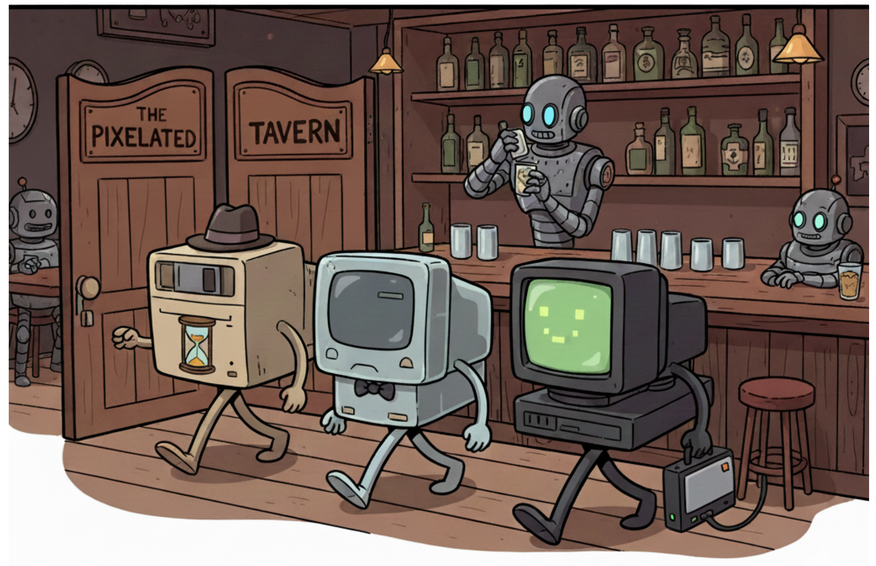


- Computer A: time $\leq T$, unlimited memory
- Computer B: memory $\sqrt{T \log T}$, unlimited time
- Computer C: memory $\sqrt{T}$, unlimited time, full hard drive of size $2^{T^{0.001}}$

Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

- Folklore: if Computer B had memory T, then it would be more powerful than Computer A
- *[Hopcroft-Paul-Valiant '75]* If Computer B instead had memory $T/\log T$, then it would be more powerful than Computer A

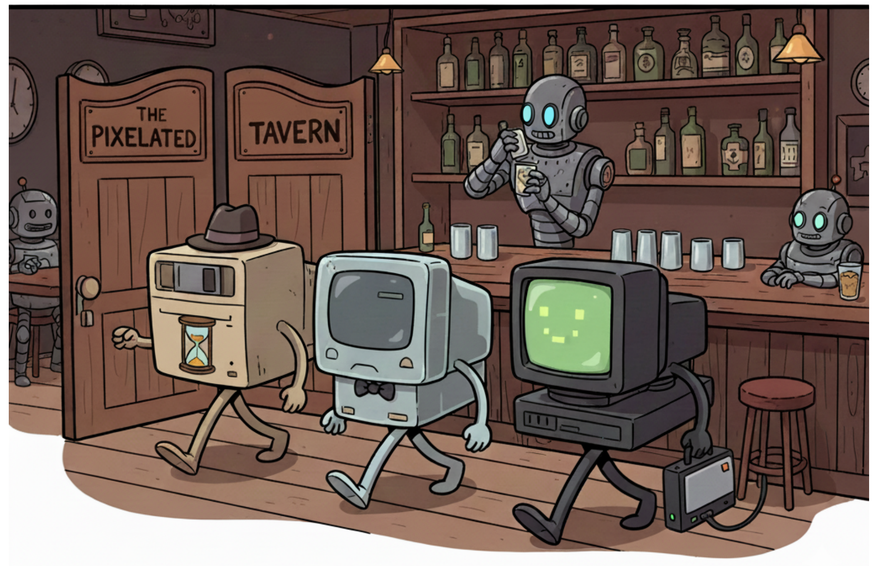


- Computer A: time $\leq T$, unlimited memory
- Computer B: memory $\sqrt{T \log T}$, unlimited time
- Computer C: memory $\sqrt{T}$, unlimited time, full hard drive of size $2^{T^{0.001}}$

Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

- Folklore: if Computer B had memory T, then it would be more powerful than Computer A
- *[Hopcroft-Paul-Valiant '75]* If Computer B instead had memory $T/\log T$, then it would be more powerful than Computer A
- *[J. Cook-Mertz '24, Williams '25]* Computer B is more powerful than Computer A!

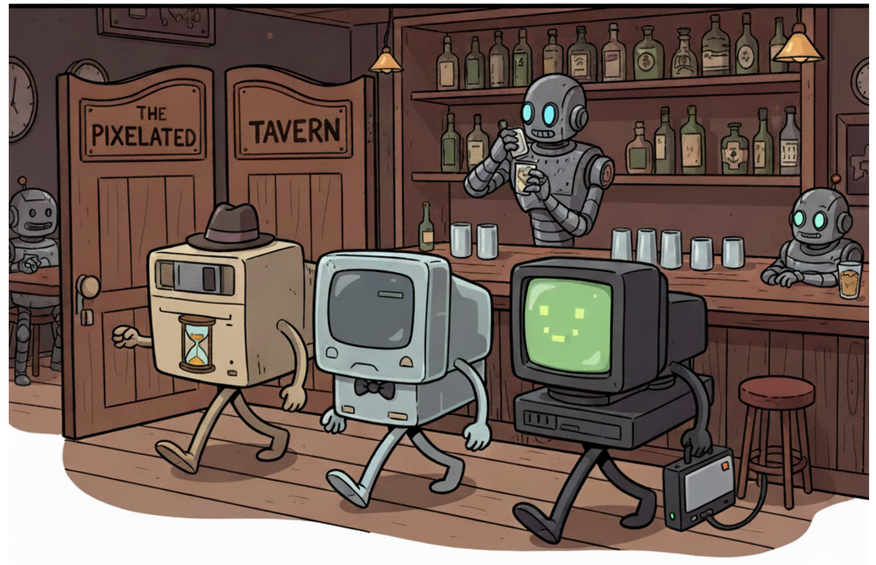


- Computer A: time $\leq T$, unlimited memory
- Computer B: memory $\sqrt{T \log T}$, unlimited time
- Computer C: memory $\sqrt{T}$, unlimited time, full hard drive of size $2^{T^{0.001}}$

Three Turing Machines Walk Into a Bar

[Williams, STOC 2025]

- Folklore: if Computer B had memory T, then it would be more powerful than Computer A
- *[Hopcroft-Paul-Valiant '75]* If Computer B instead had memory $T/\log T$, then it would be more powerful than Computer A
- *[J. Cook-Mertz '24, Williams '25]* Computer B is more powerful than Computer A!
- **Our work** + *[Williams '25]:* Computer C is also more powerful than Computer A!

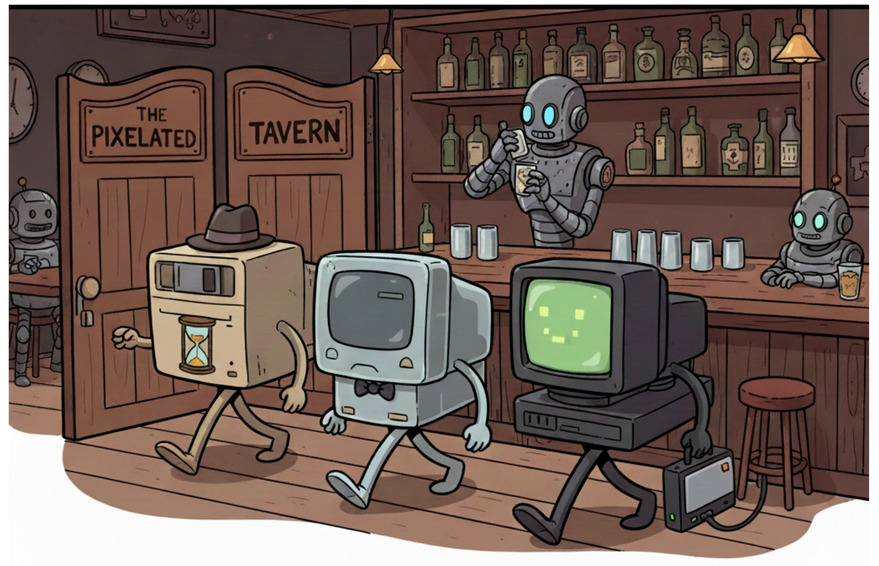


- Computer A: time $\leq T$, unlimited memory
- Computer B: memory $\sqrt{T \log T}$, unlimited time
- Computer C: memory $\sqrt{T}$, unlimited time, full hard drive of size $2^{T^{0.001}}$

Three Turing Machines Walk Into a Bar

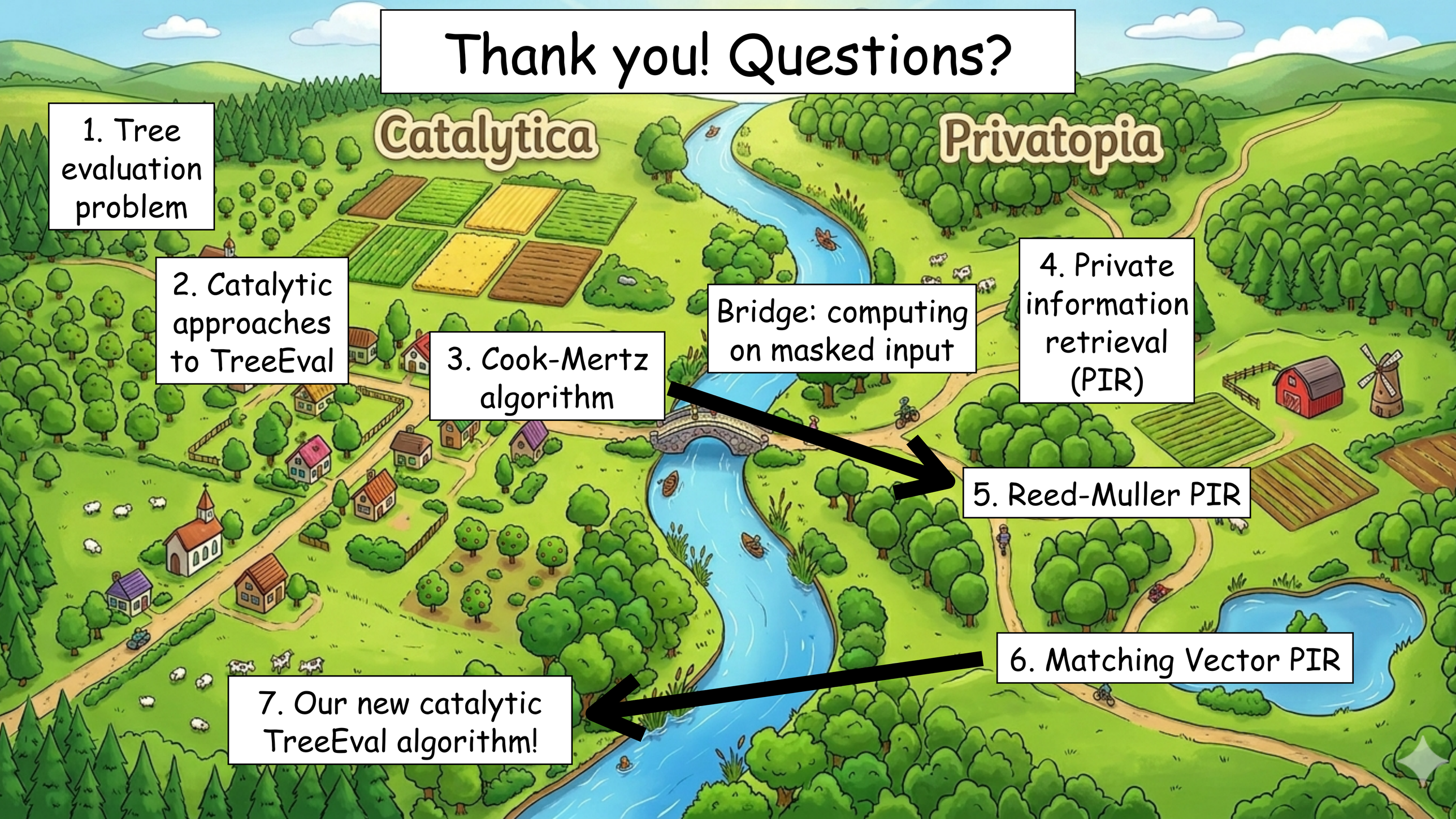
[Williams, STOC 2025]

- Folklore: if Computer B had memory T, then it would be more powerful than Computer A
- *[Hopcroft-Paul-Valiant '75]* If Computer B instead had memory $T/\log T$, then it would be more powerful than Computer A
- *[J. Cook-Mertz '24, Williams '25]* Computer B is more powerful than Computer A!
- **Our work** + *[Williams '25]:* Computer C is also more powerful than Computer A!



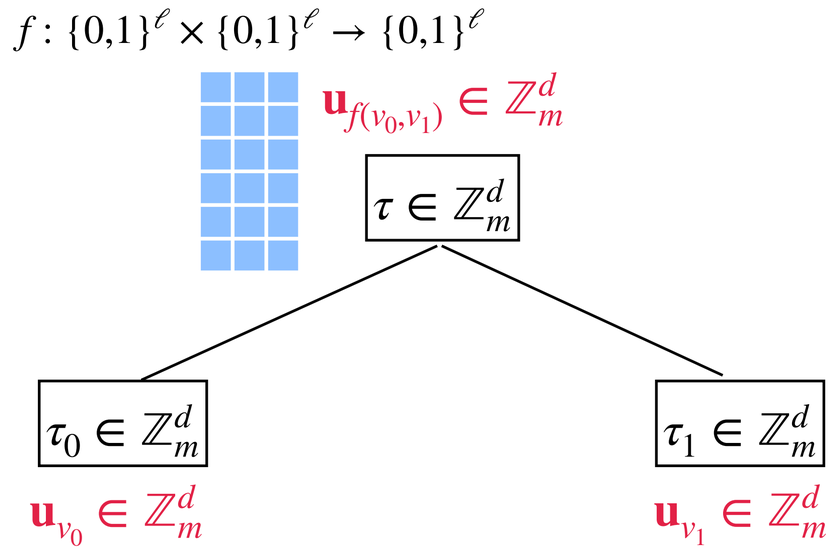
Answer: Computer A is strictly weakest

- Computer A: time $\leq T$, unlimited memory
- Computer B: memory $\sqrt{T \log T}$, unlimited time
- Computer C: memory $\sqrt{T}$, unlimited time, full hard drive of size $2^{T^{0.001}}$


Thank you! Questions?
Catalytica
Privatopia
1. Tree evaluation problem
2. Catalytic approaches to TreeEval
3. Cook-Mertz algorithm
Bridge: computing on masked input
4. Private information retrieval (PIR)
5. Reed-Muller PIR
6. Matching Vector PIR
7. Our new catalytic TreeEval algorithm!

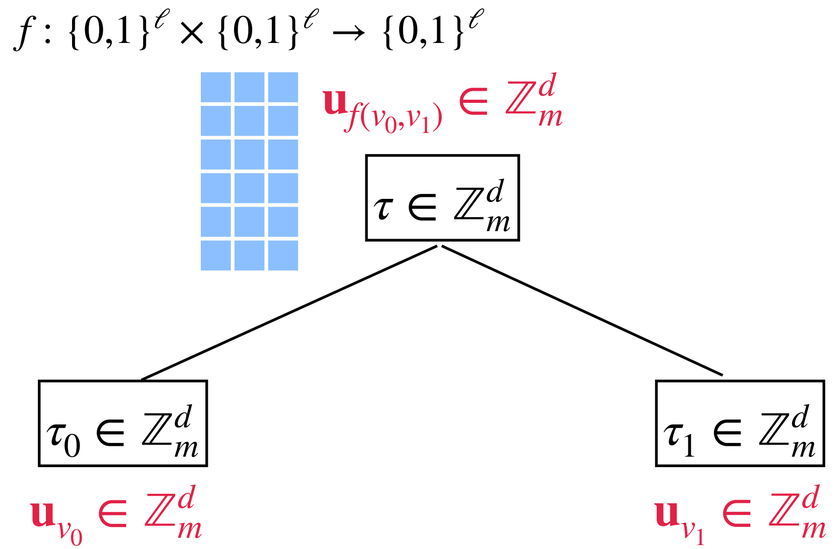
Bonus Slides

The Fine Print



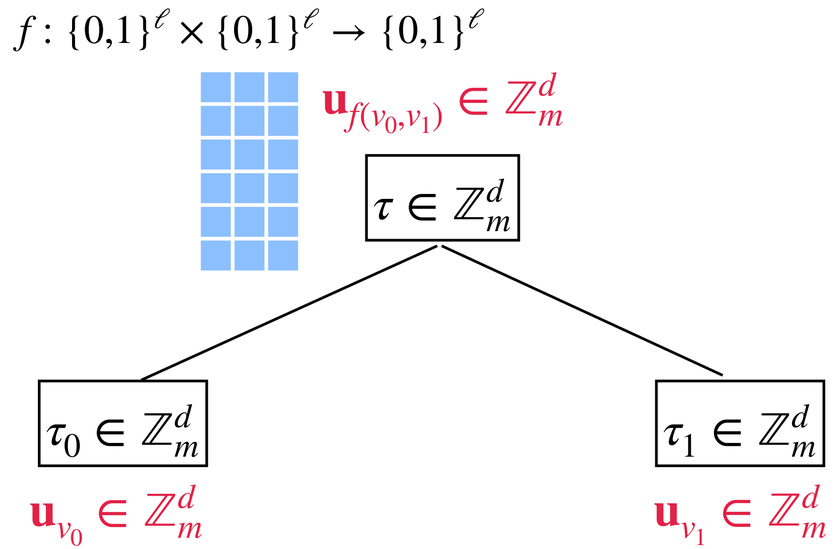
The Fine Print

- 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



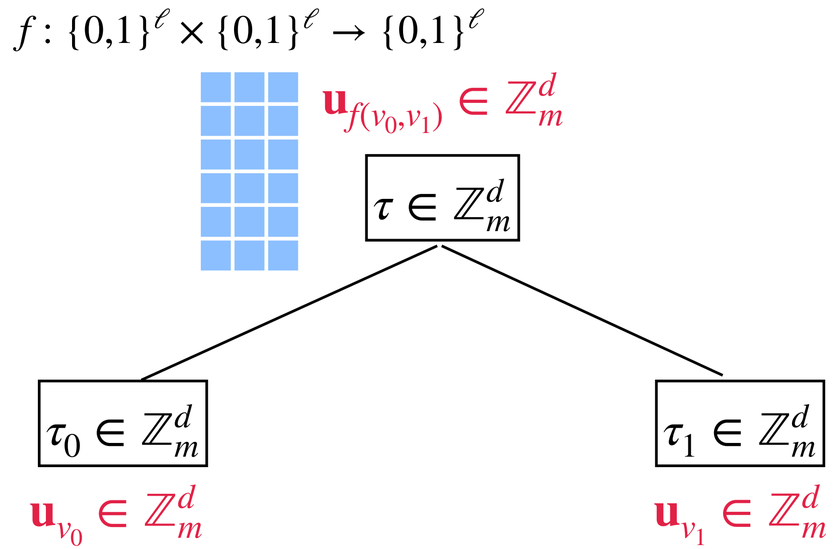
The Fine Print

- 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 $a||b$