NanoGram: Garbled RAM with Õ(log N) Overhead

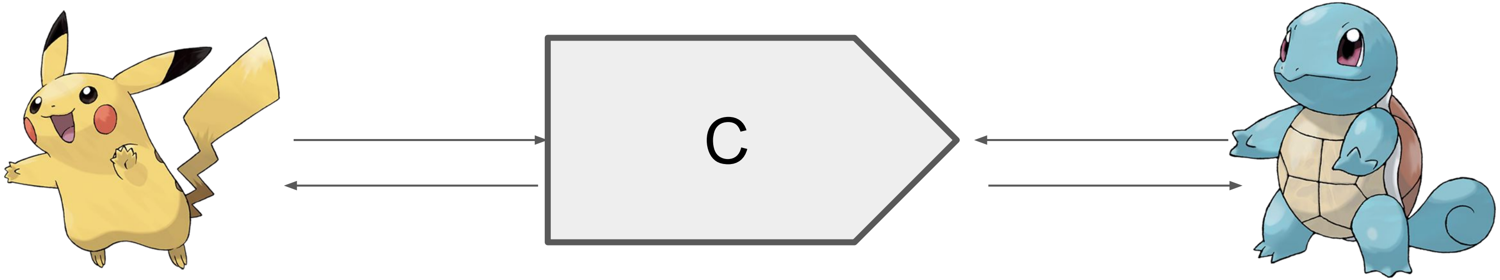
Andrew Park
CMU

Wei-Kai Lin
Northeastern

Elaine Shi
CMU

Garbled Circuits [Yao82,Yao86]

Garbled Circuits [Yao82,Yao86]

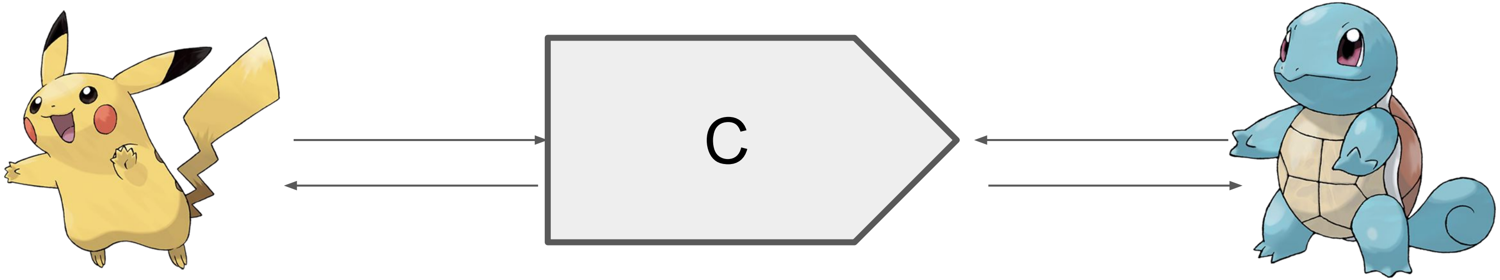


C

Garbler

Evaluator

Garbled Circuits [Yao82,Yao86]

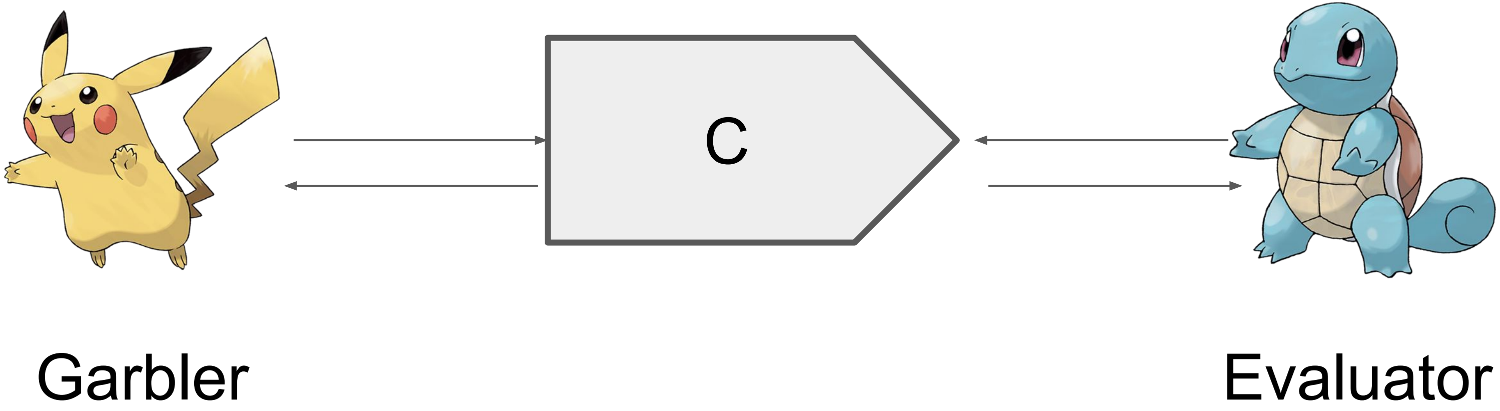


Garbler

Evaluator

😃 Non Interactive

Garbled Circuits [Yao82,Yao86]



C

Garbler

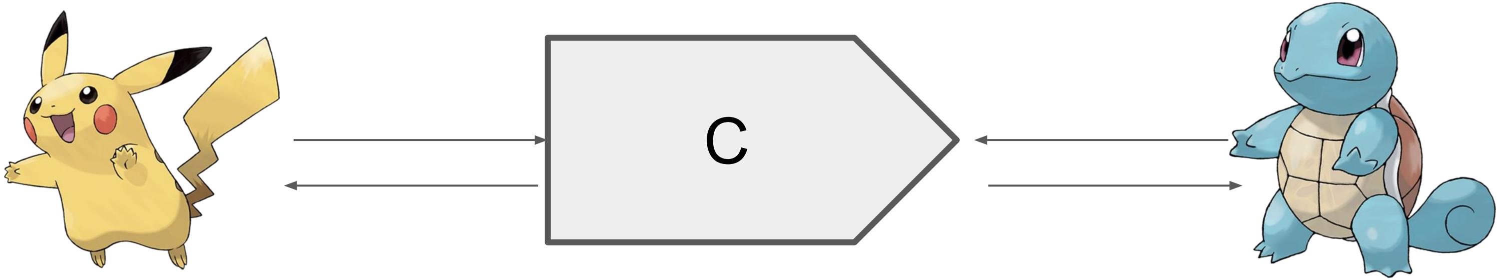
Evaluator

😃 Non Interactive

😞 Large RAM-to-circuit conversion

Garbled RAM [LO13]

Garbled RAM [LO13]

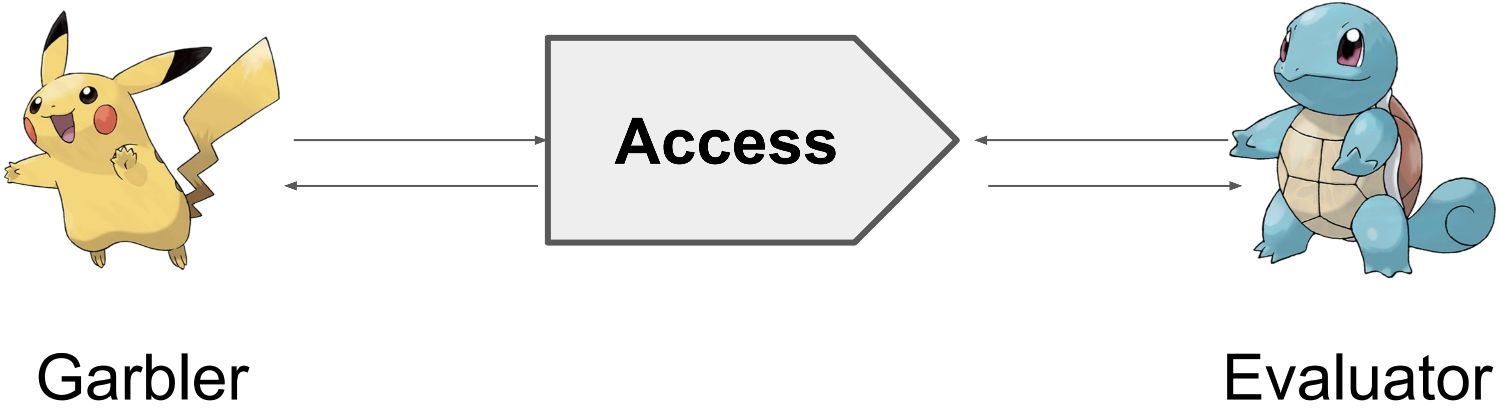


C

Garbler

Evaluator

Garbled RAM [LO13]

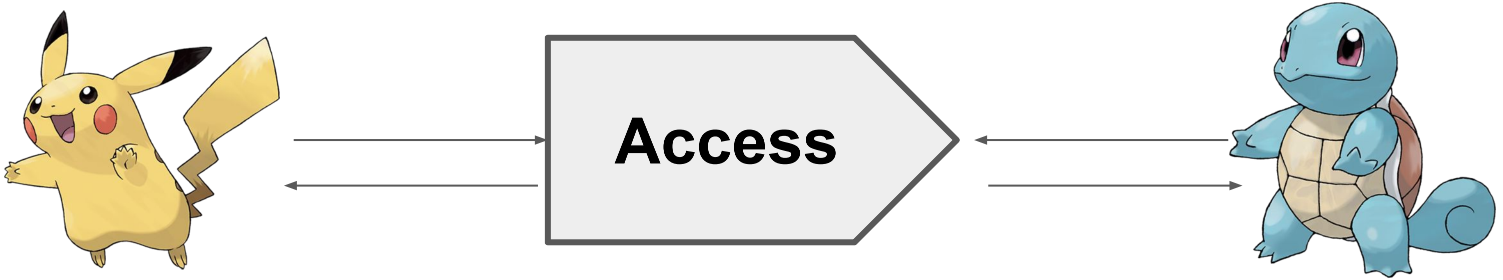


Access

Garbler

Evaluator

Garbled RAM [LO13]



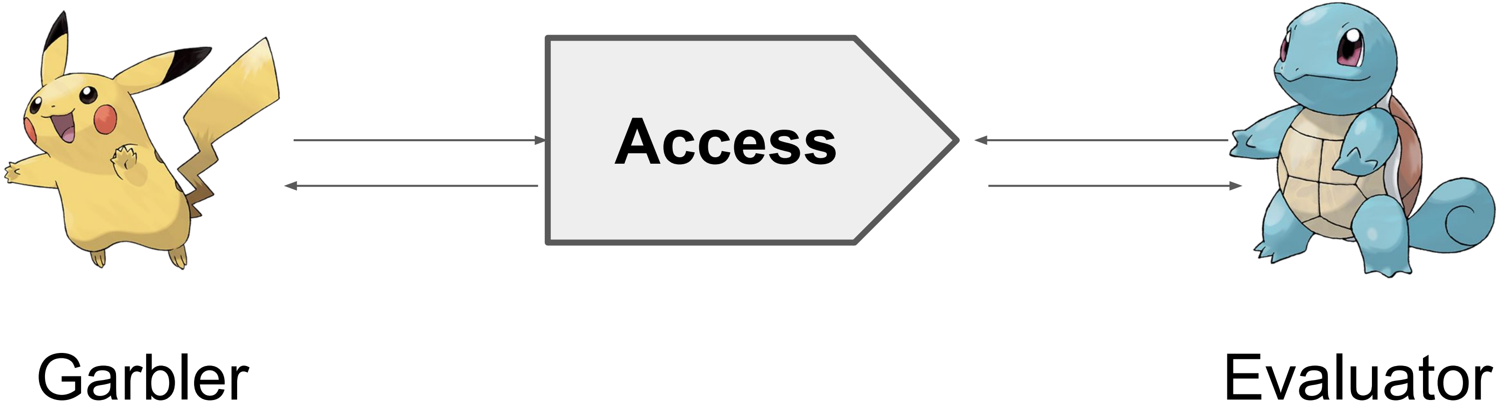
Garbler

Evaluator

Input: $\{\{i\}\}$

Output: $\{\{x_i\}\}$

Garbled RAM [LO13]



Input: {{i}}

Output: {{x_i}}

Goal: Access takes time **sublinear** in size of RAM

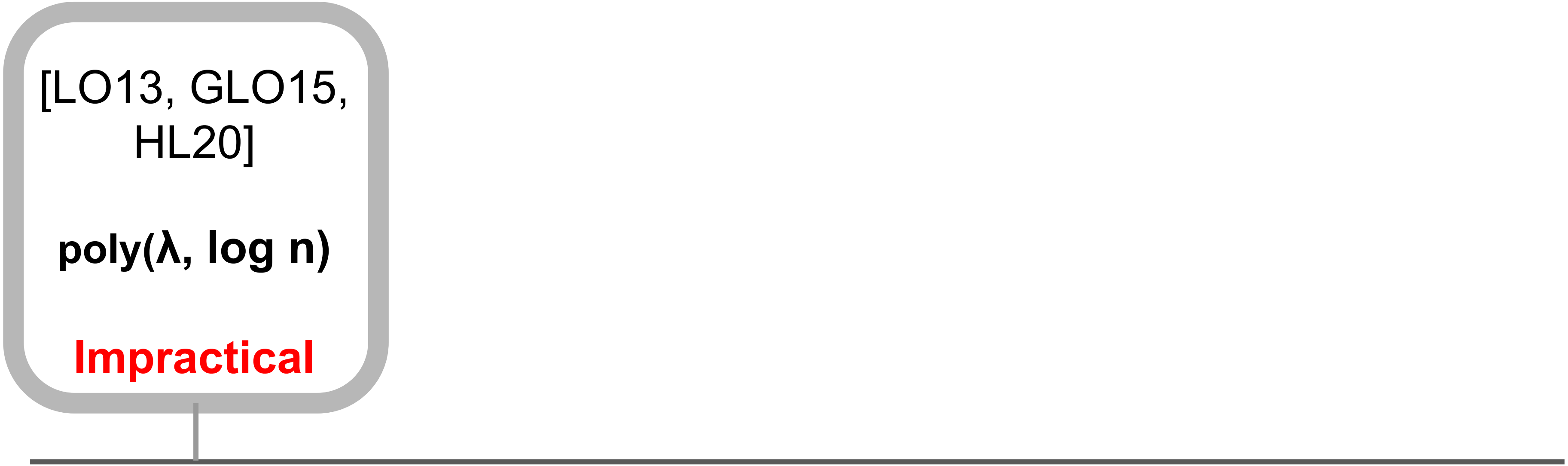
Garbled RAM Landscape

Garbled RAM Landscape

[LO13, GLO15, HL20]

poly(λ, log n)

Garbled RAM Landscape

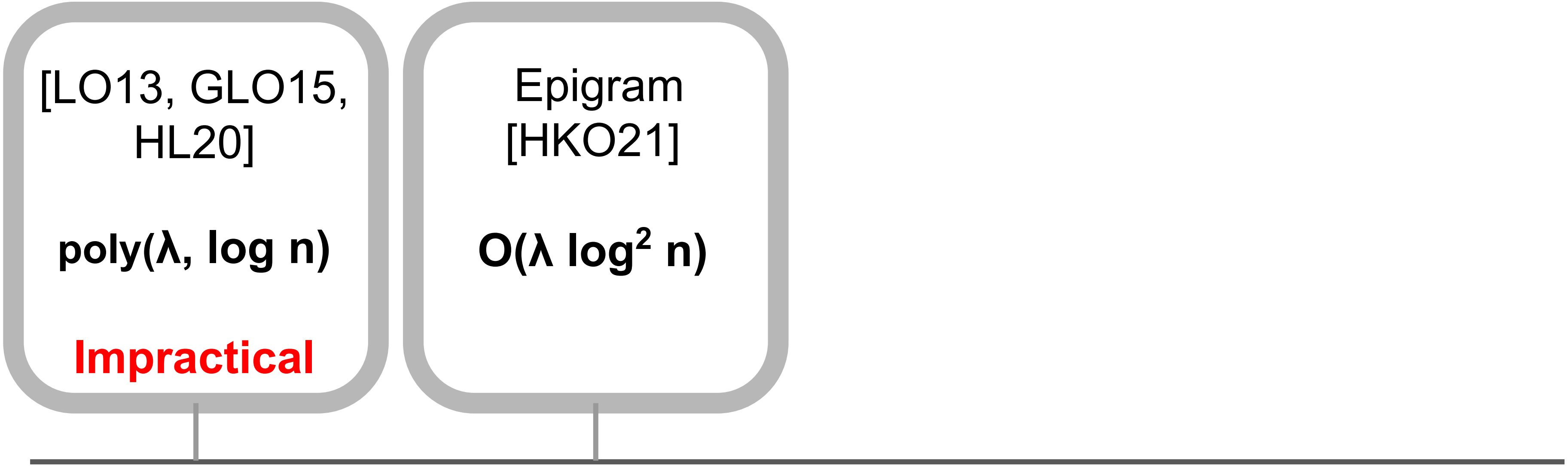


[LO13, GLO15, HL20]

poly(λ, log n)

Impractical

Garbled RAM Landscape



[LO13, GLO15, HL20]

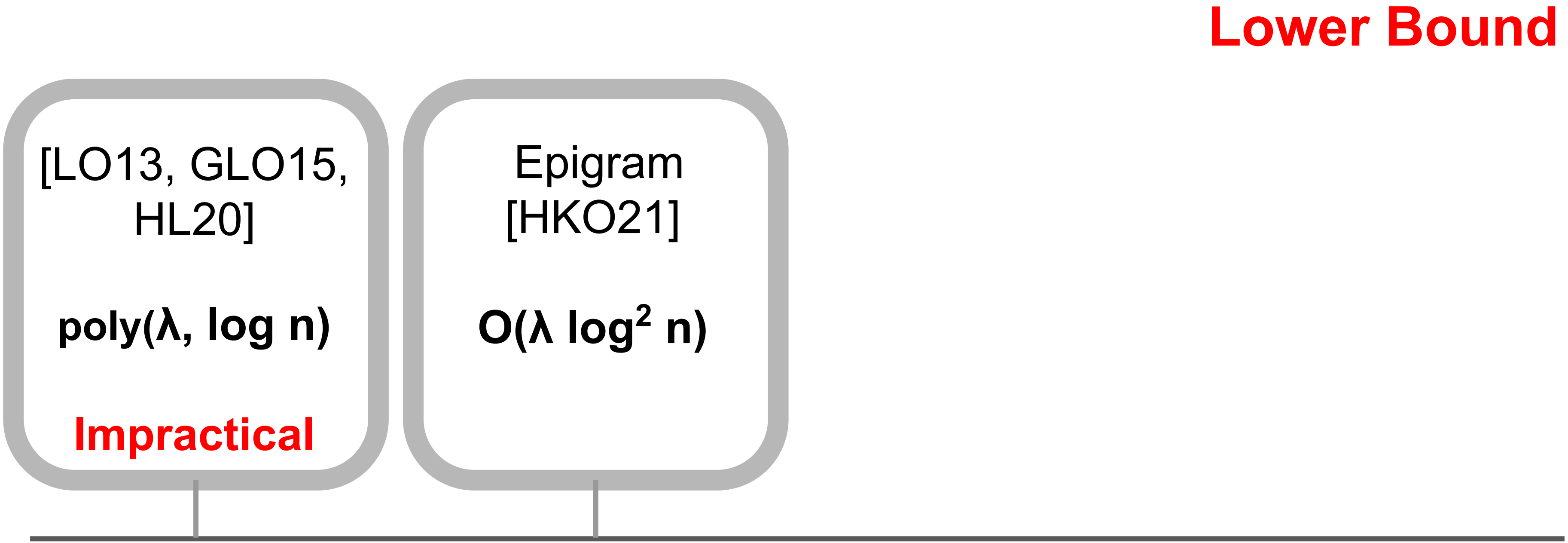
poly(λ, log n)

Impractical

Epigram [HKO21]

$O(\lambda \log^2 n)$

Garbled RAM Landscape



Lower Bound

[LO13, GLO15, HL20]

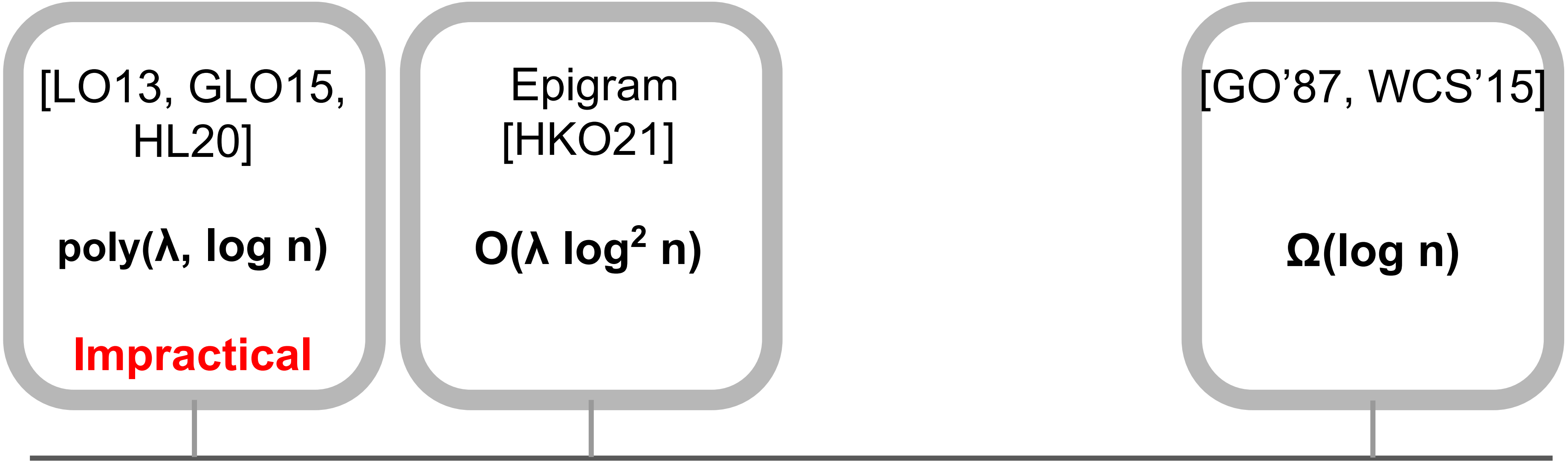
poly(λ, log n)

Impractical

Epigram [HKO21]

$O(\lambda \log^2 n)$

Garbled RAM Landscape



[LO13, GLO15, HL20]

poly(λ, log n)

Impractical

Epigram [HKO21]

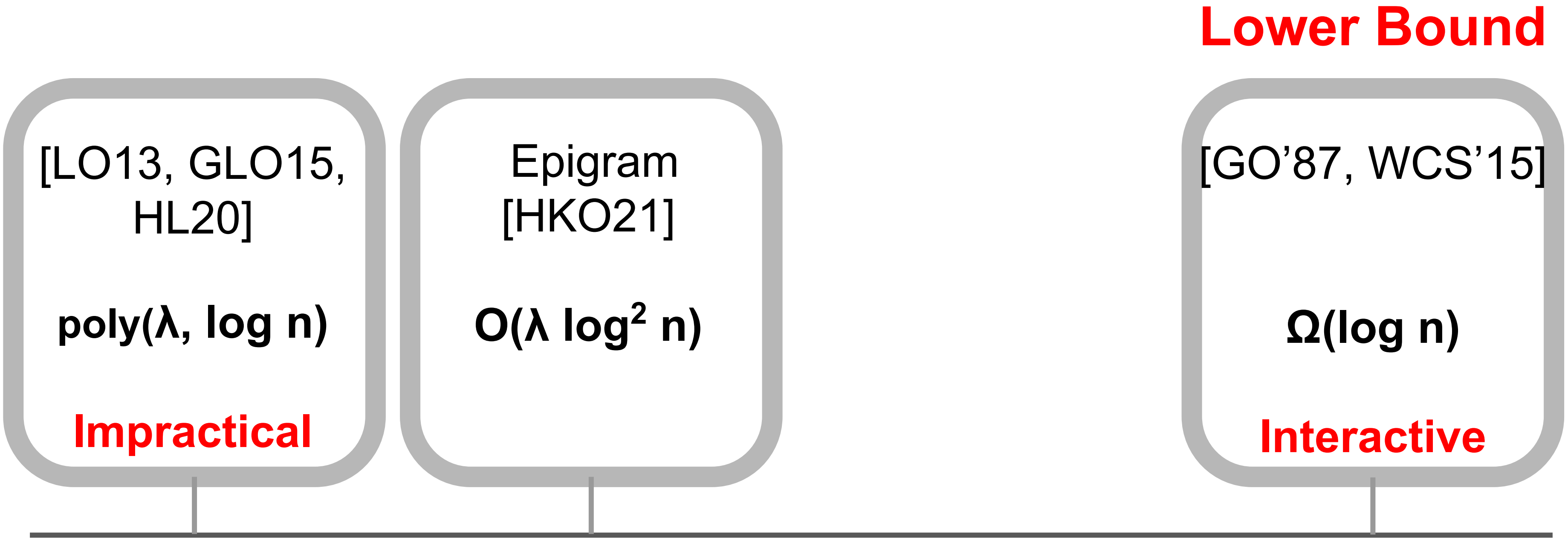
$O(\lambda \log^2 n)$

Lower Bound

[GO’87, WCS’15]

$\Omega(\log n)$

Garbled RAM Landscape



Lower Bound

[LO13, GLO15, HL20]

$\text{poly}(\lambda, \log n)$

Impractical

Epigram [HKO21]

$O(\lambda \log^2 n)$

[GO'87, WCS'15]

$\Omega(\log n)$

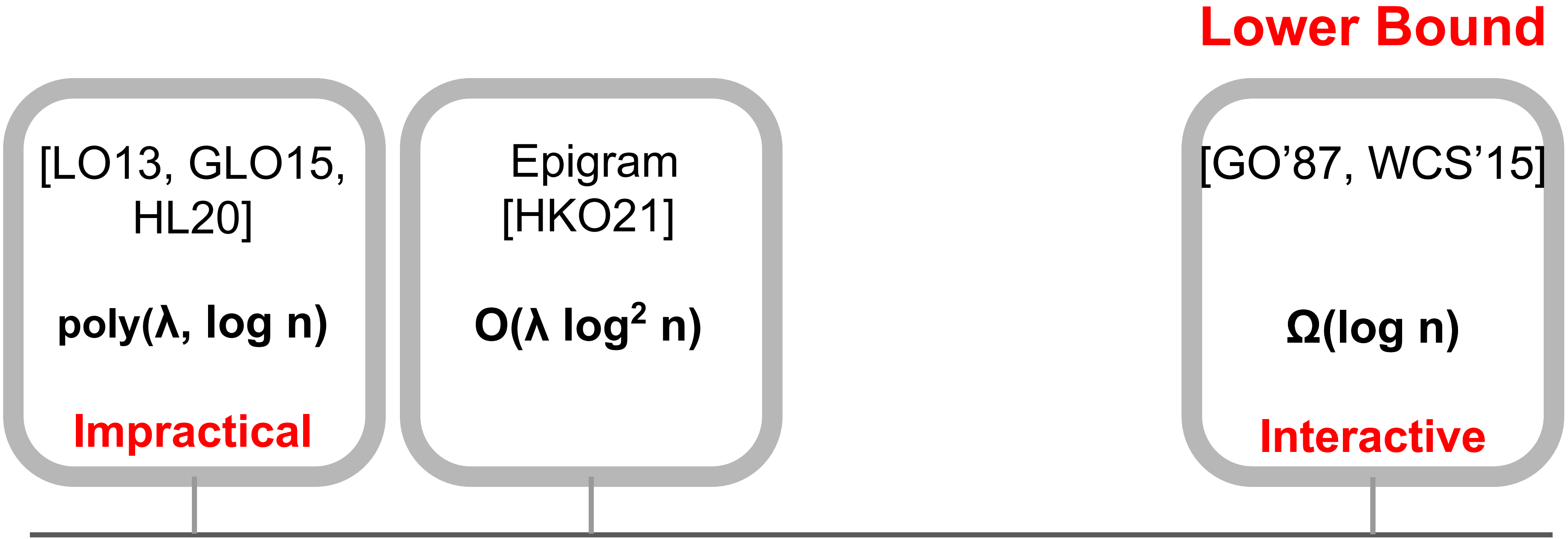
Interactive

This Work

This Work

Can we have a (non-interactive) garbled RAM scheme whose asymptotical performance is competitive to the interactive state-of-the-art?

This Work



Lower Bound

[LO13, GLO15, HL20]

$\mathrm{poly}(\lambda, \log n)$

Impractical

Epigram [HKO21]

$O(\lambda \log^2 n)$

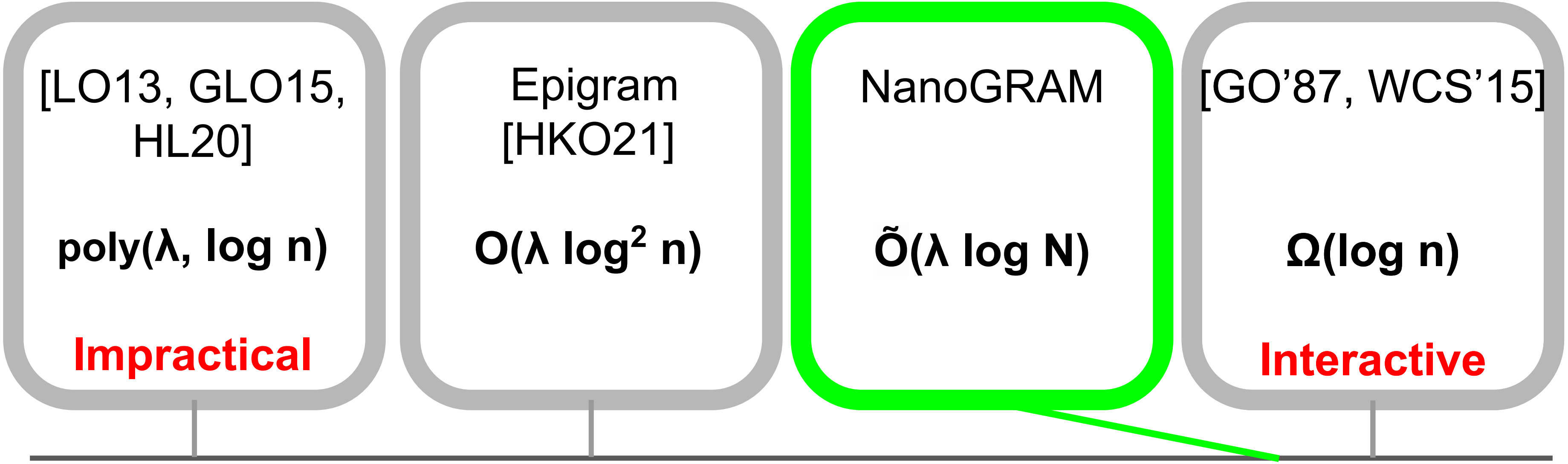
[GO'87, WCS'15]

$\Omega(\log n)$

Interactive

This Work

Lower Bound



[LO13, GLO15, HL20]

poly(λ, log n)

Impractical

Epigram [HKO21]

$O(\lambda \log^2 n)$

NanoGRAM

$\tilde{O}(\lambda \log N)$

[GO’87, WCS’15]

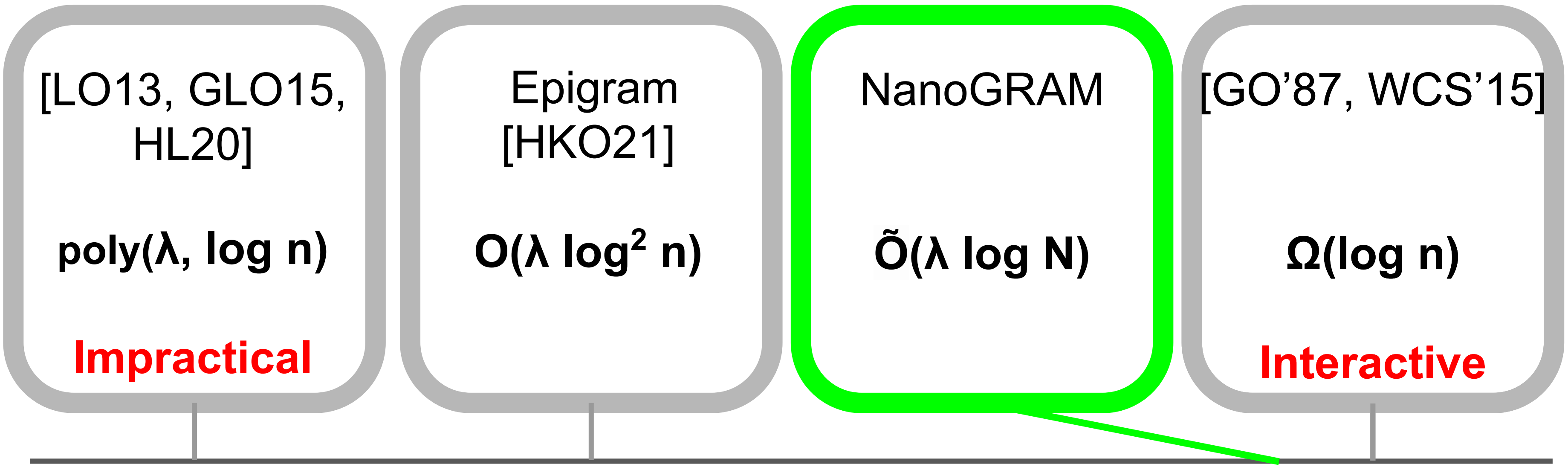
$\Omega(\log n)$

Interactive

This Work

Near optimal dependence on N

Lower Bound



[LO13, GLO15, HL20]

poly(λ, log n)

Impractical

Epigram [HKO21]

$O(\lambda \log^2 n)$

NanoGRAM

$\tilde{O}(\lambda \log N)$

[GO’87, WCS’15]

$\Omega(\log n)$

Interactive

Outline

The Language Translation Problem

Strawman: Garbled ORAM Tree

Our Techniques

Outline

The Language Translation Problem

Strawman: Garbled ORAM Tree

Our Techniques

The Language Translation Problem

Each garbled circuit speaks a time-dependent language

The Language Translation Problem

Each garbled circuit speaks a time-dependent language

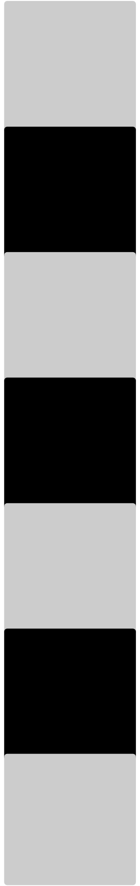
{{input}} ←Encode(sk, data, L)

The Language Translation Problem

T = 0

T = 1

T = 2



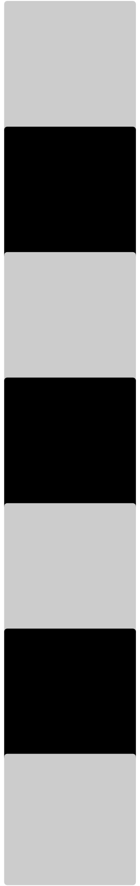
The Language Translation Problem

T = 0

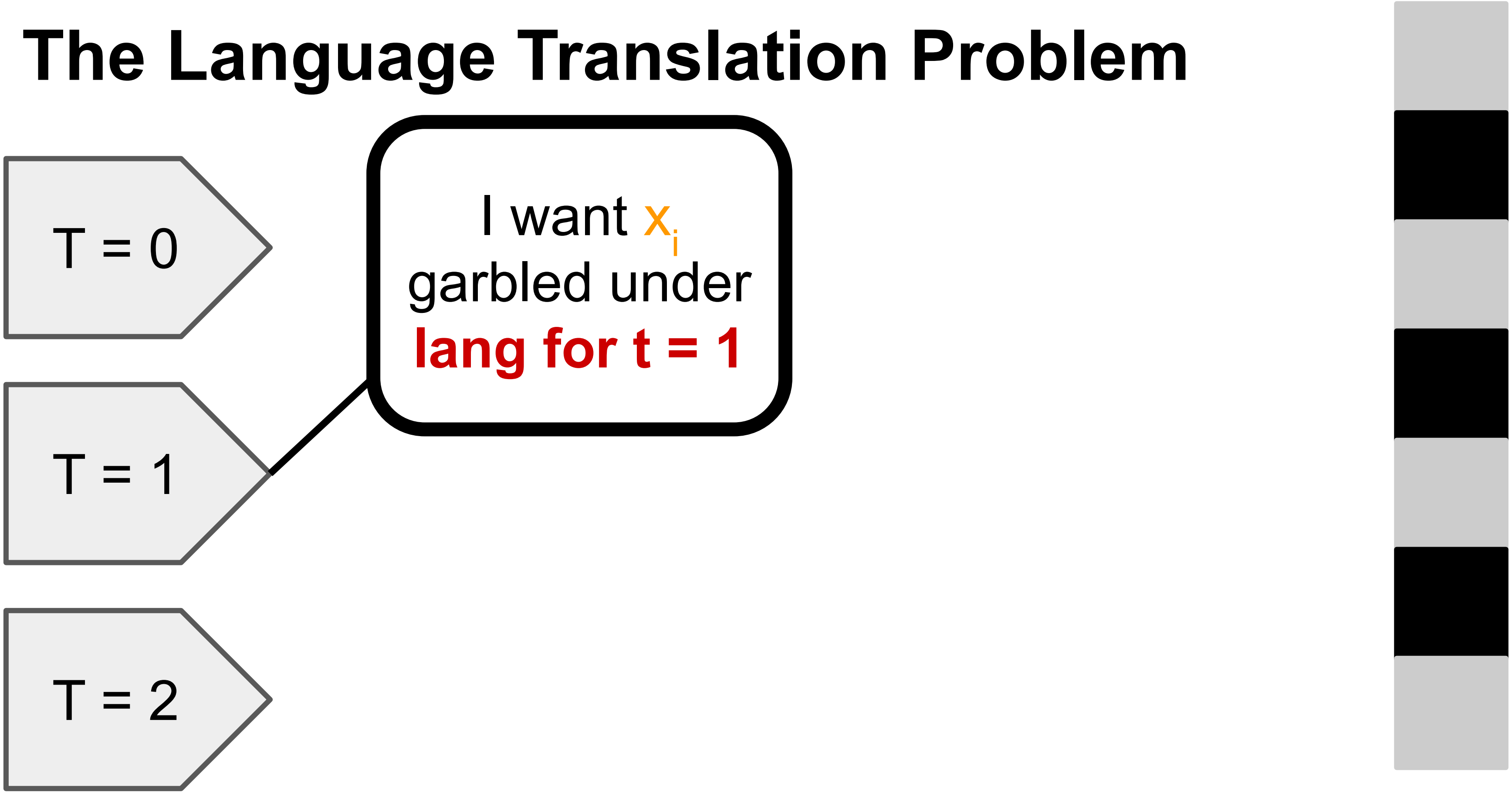
Each gate expects input garbled under time-dependent language

T = 1

T = 2



The Language Translation Problem



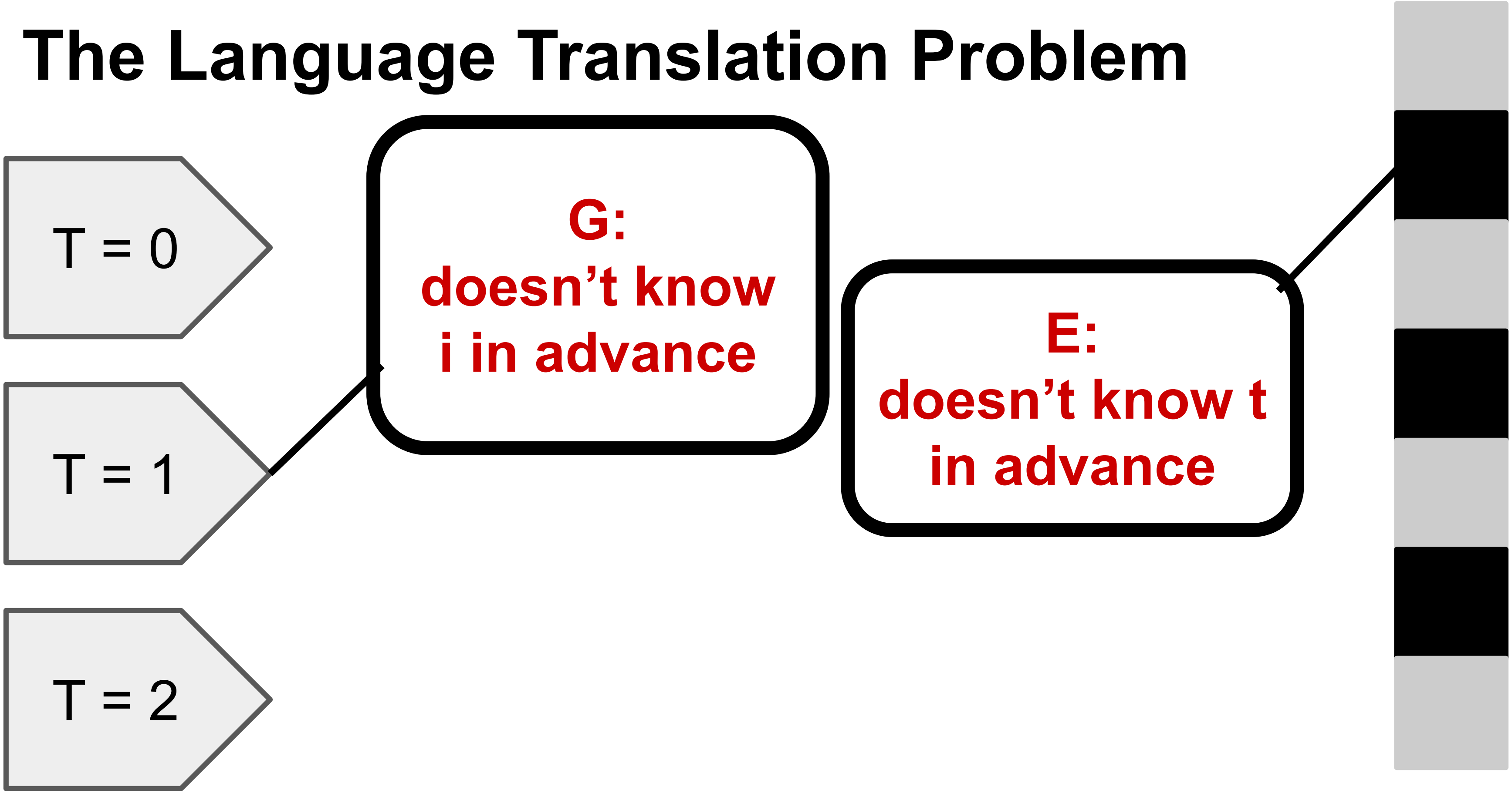
T = 0

T = 1

T = 2

I want x_i garbled under **lang for t = 1**

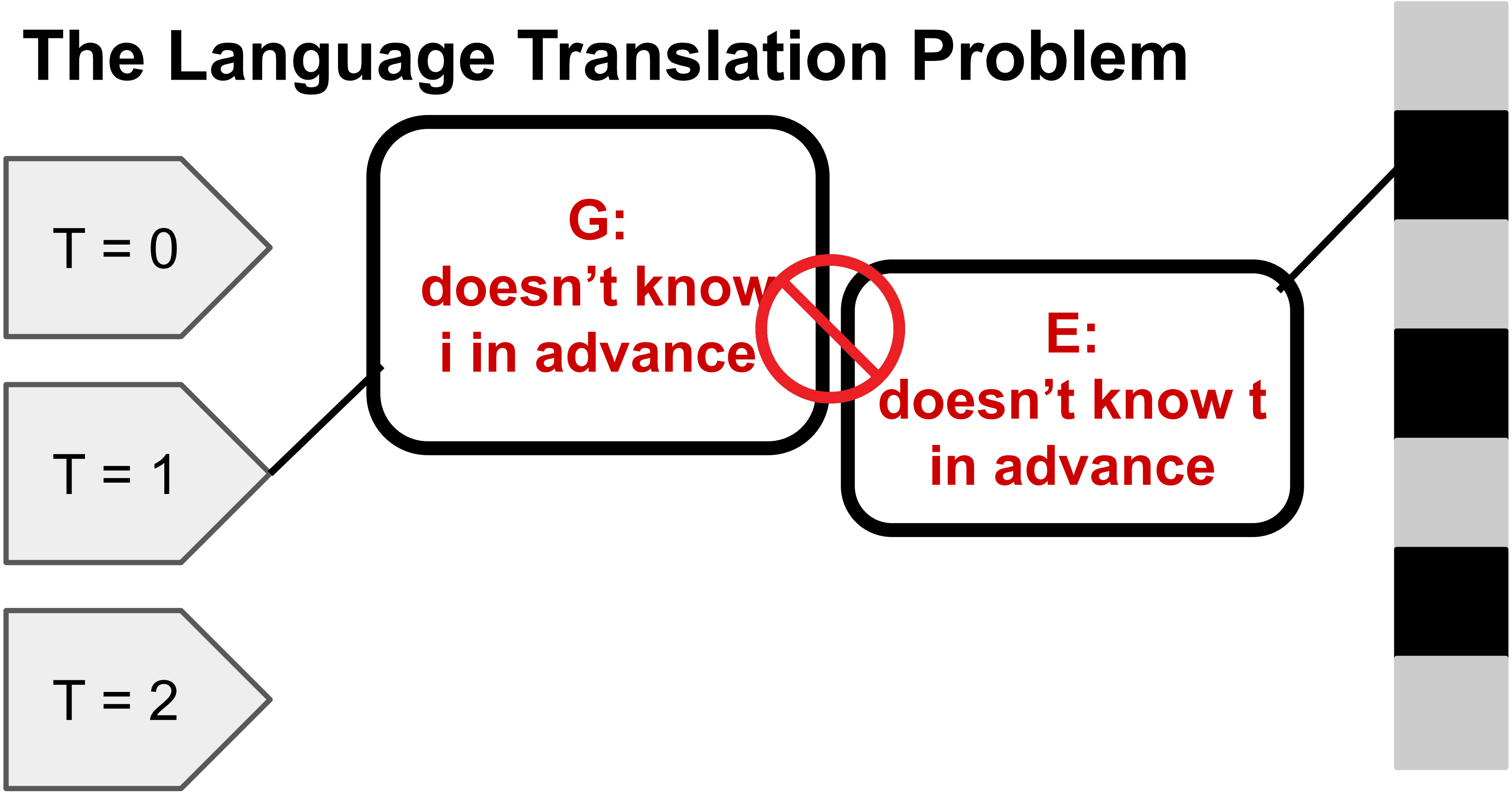
The Language Translation Problem



G:
doesn’t know i in advance

E:
doesn’t know t in advance

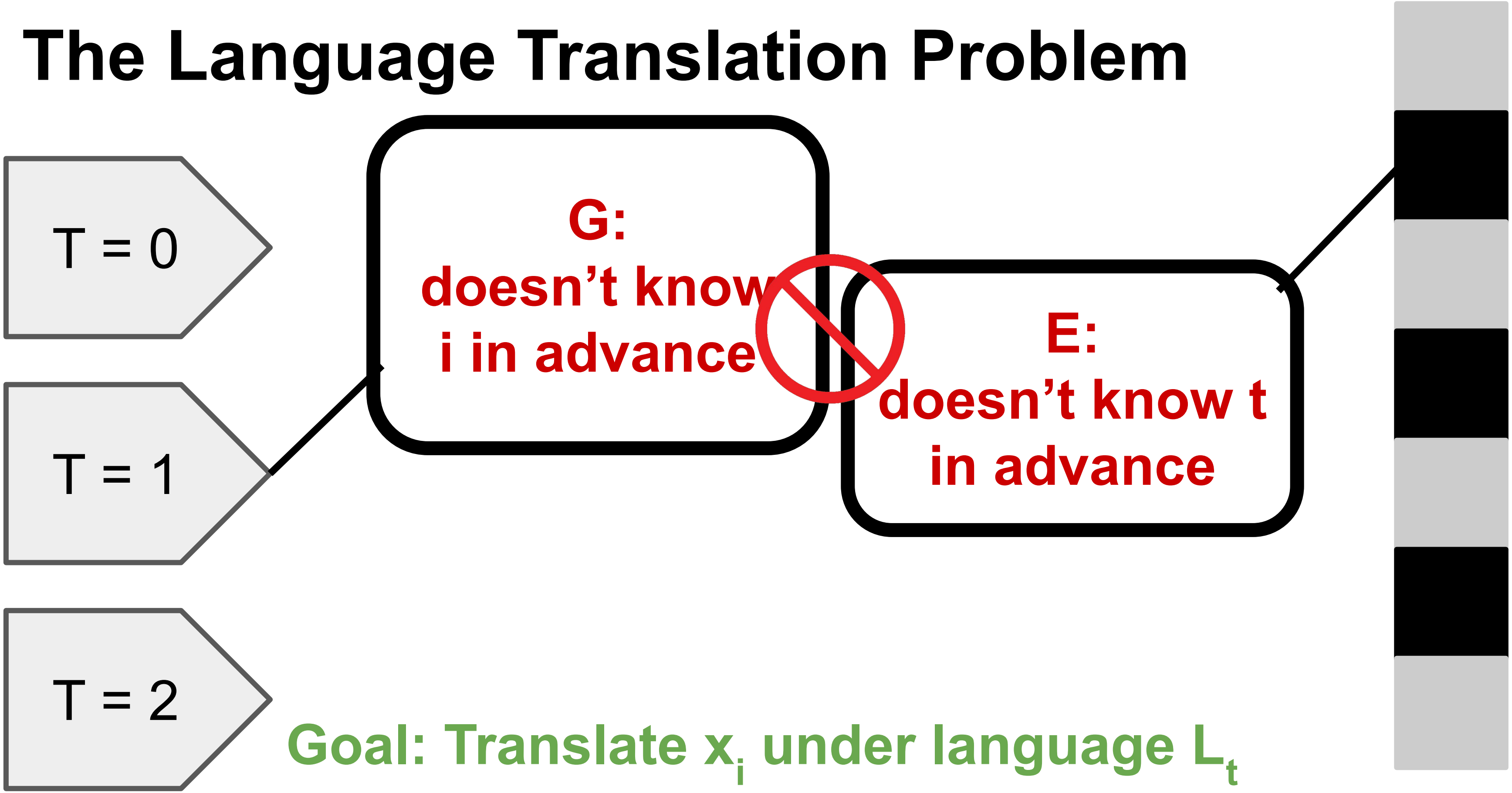
The Language Translation Problem



G:
doesn’t know i in advance

E:
doesn’t know t in advance

The Language Translation Problem



T = 0

T = 1

T = 2

G:
doesn’t know
i in advance

E:
doesn’t know t
in advance

Goal: Translate x_i under language L_t

Outline

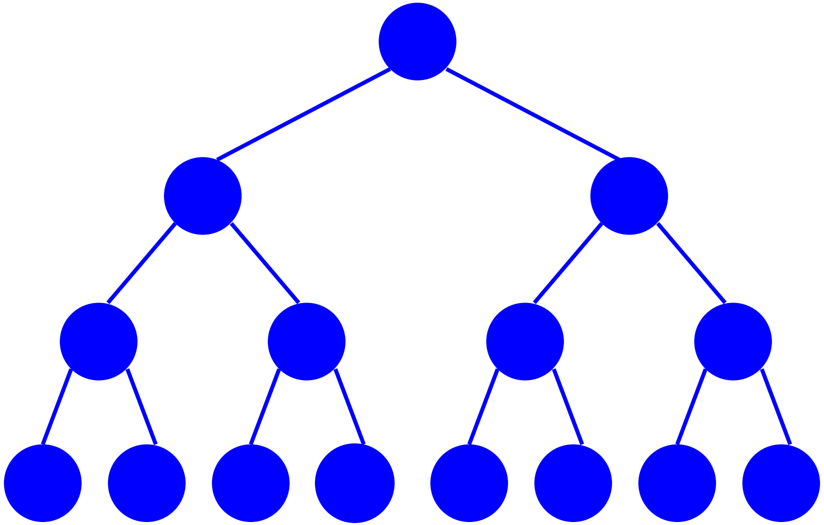
The Language Translation Problem

Strawman: Garbled ORAM Tree

Our Techniques

Strawman: Garbled ORAM Tree

Strawman: Garbled ORAM Tree

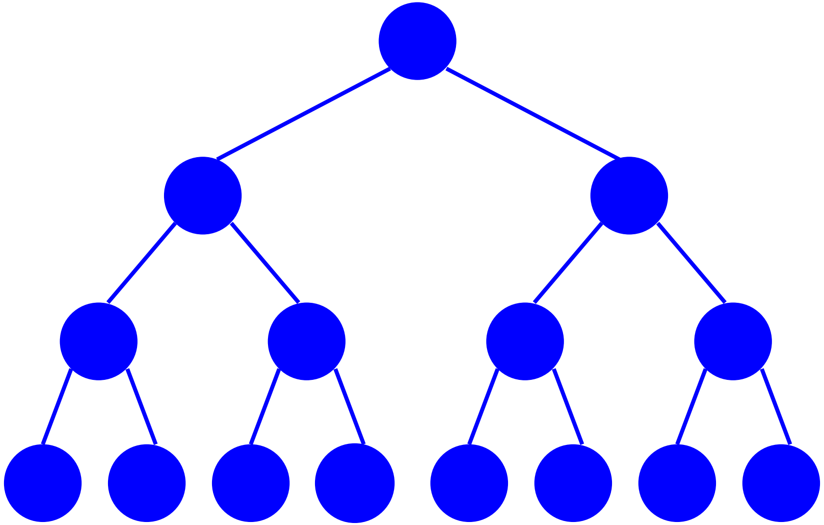


● = Garbled Data Structure

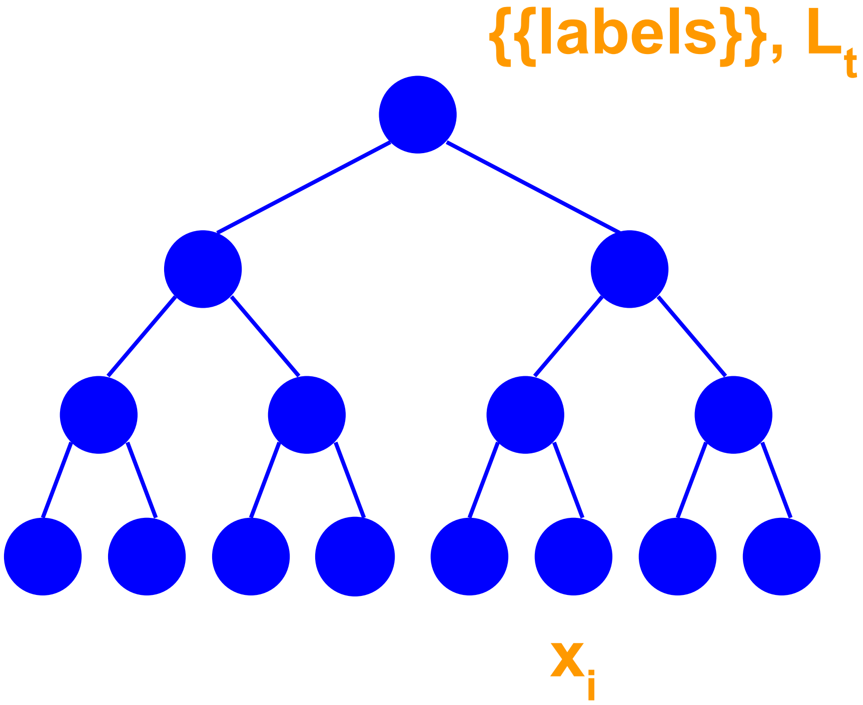
Each ● has its own local clock

Strawman: Garbled ORAM Tree

{{labels}}, L_t

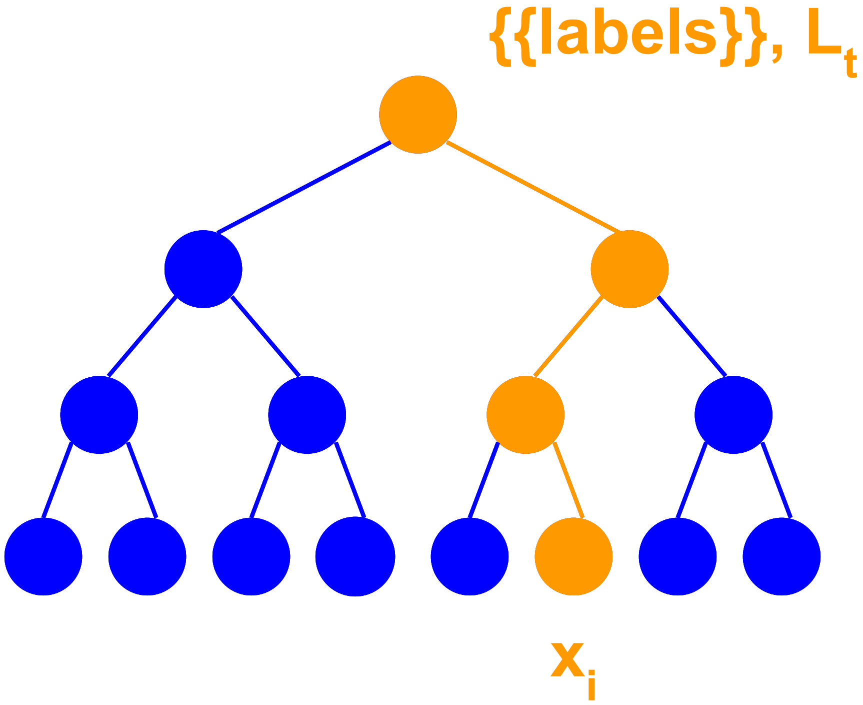


Strawman: Garbled ORAM Tree



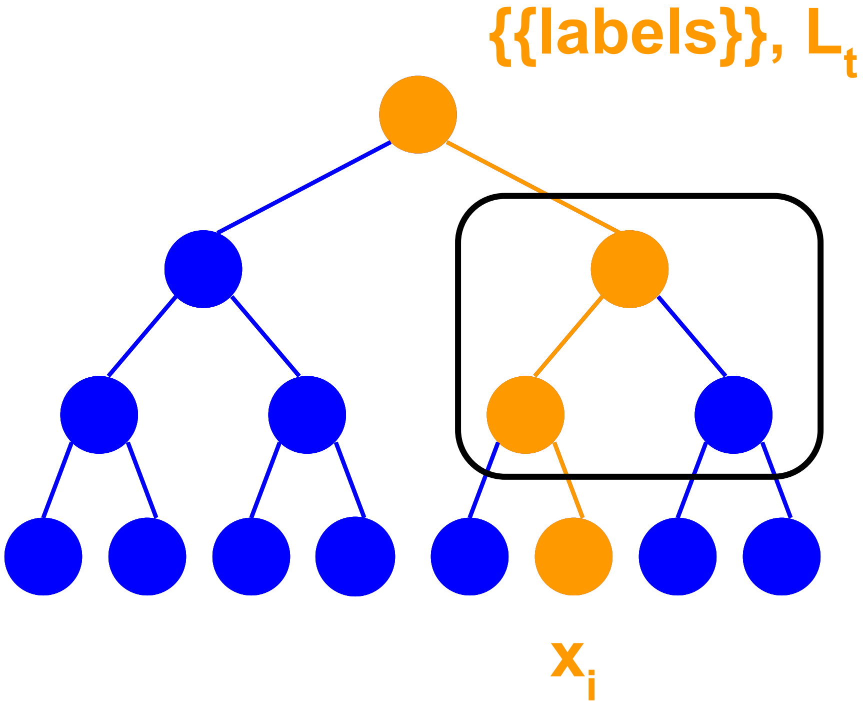
O(log N) labels, one to read from each bucket

Strawman: Garbled ORAM Tree



GOAL: Route L_t to x_i on path

Strawman: Garbled ORAM Tree

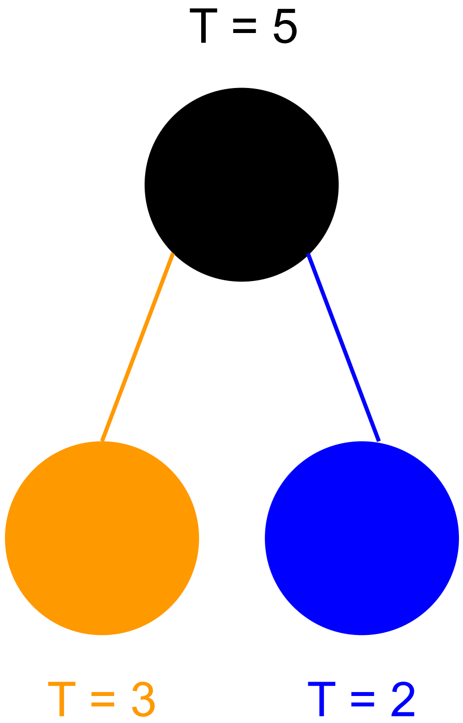


GOAL: Route L_t to x_i on path

Garbled Switch:
Gadget for Language Translation

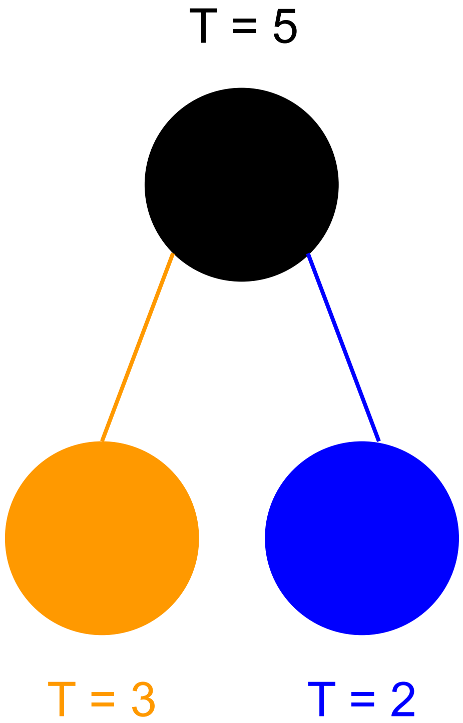
Garbled Switch [HKO21]

Garbled Switch [HKO21]



Every node has its local clock

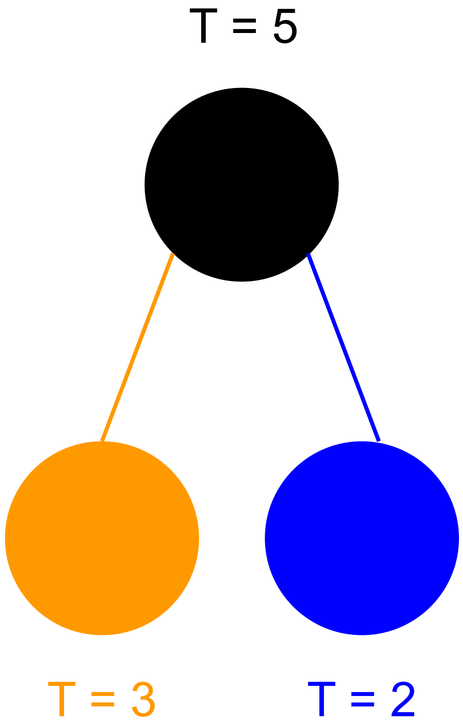
Garbled Switch [HKO21]



Every node has its local clock

When invoked, local time increments

Garbled Switch [HKO21]

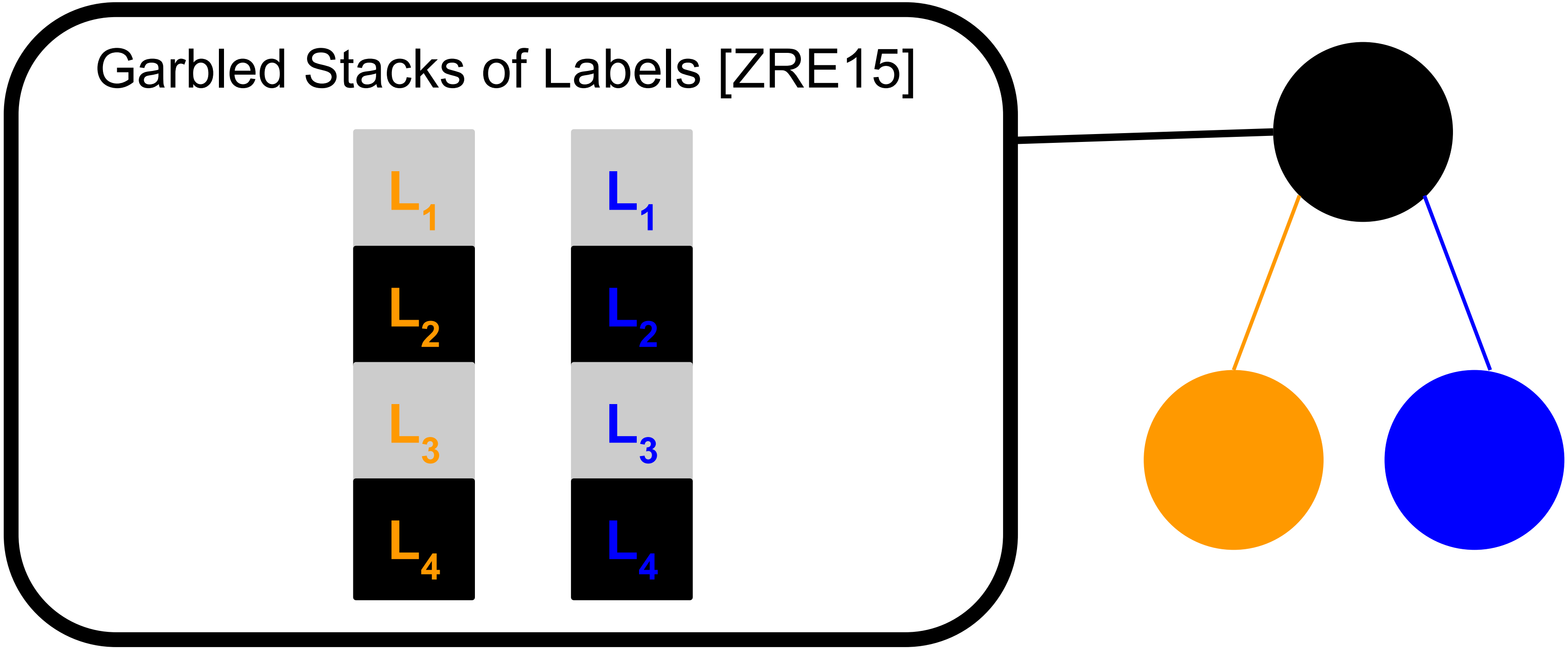


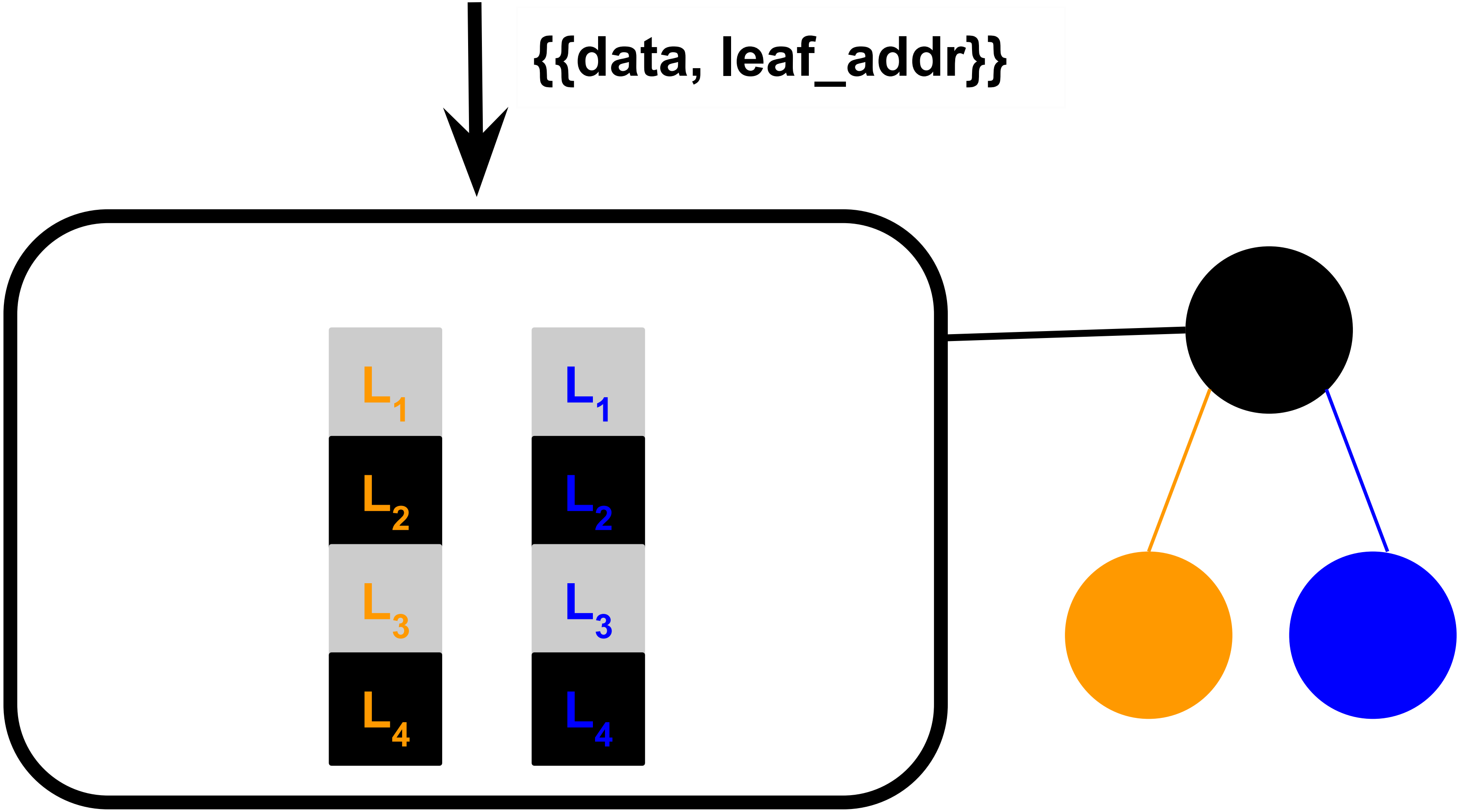
Every node has its local clock

When invoked, local time increments

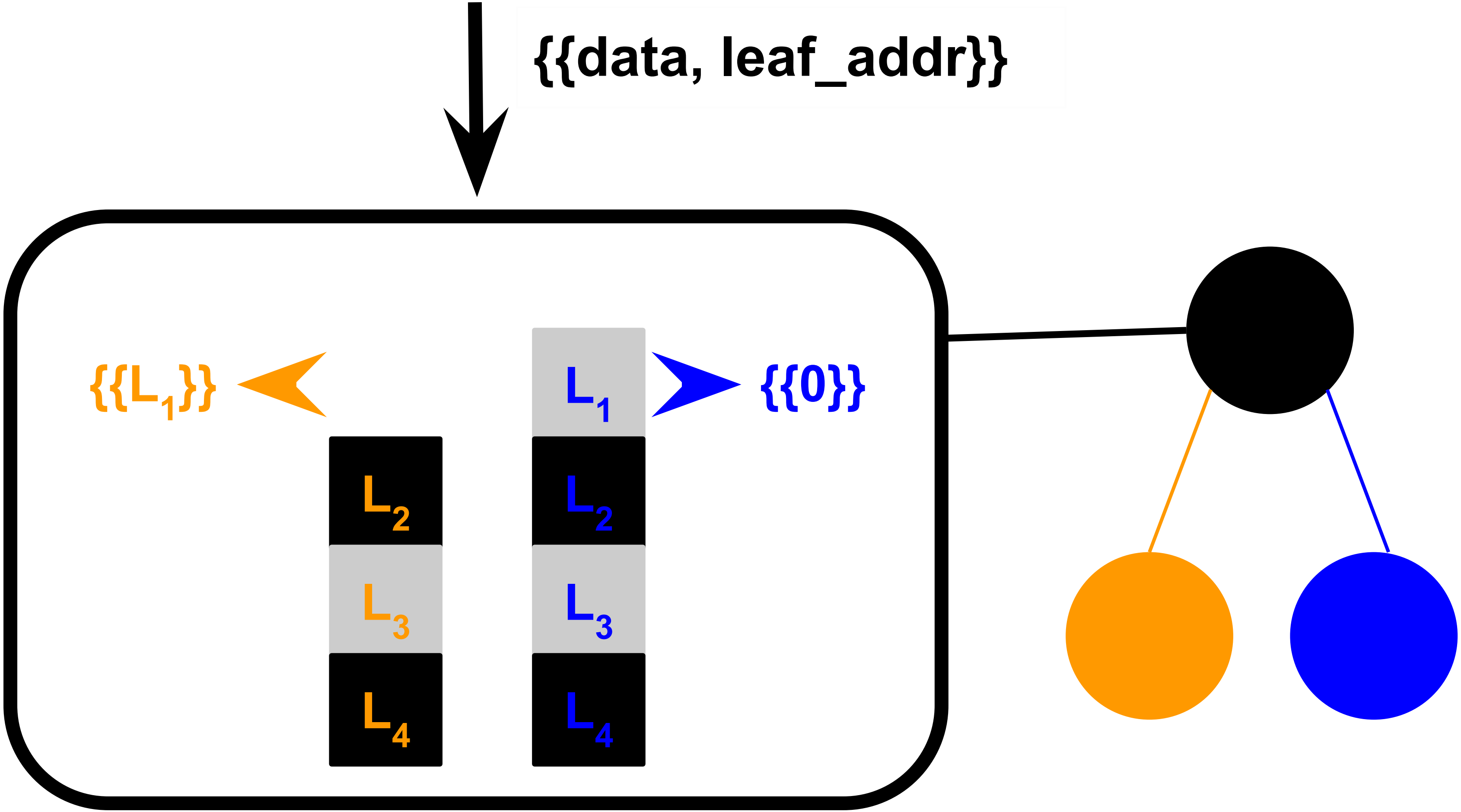
Garbled message must speak the local time dependent language of node

Garbled Stacks of Labels [ZRE15]

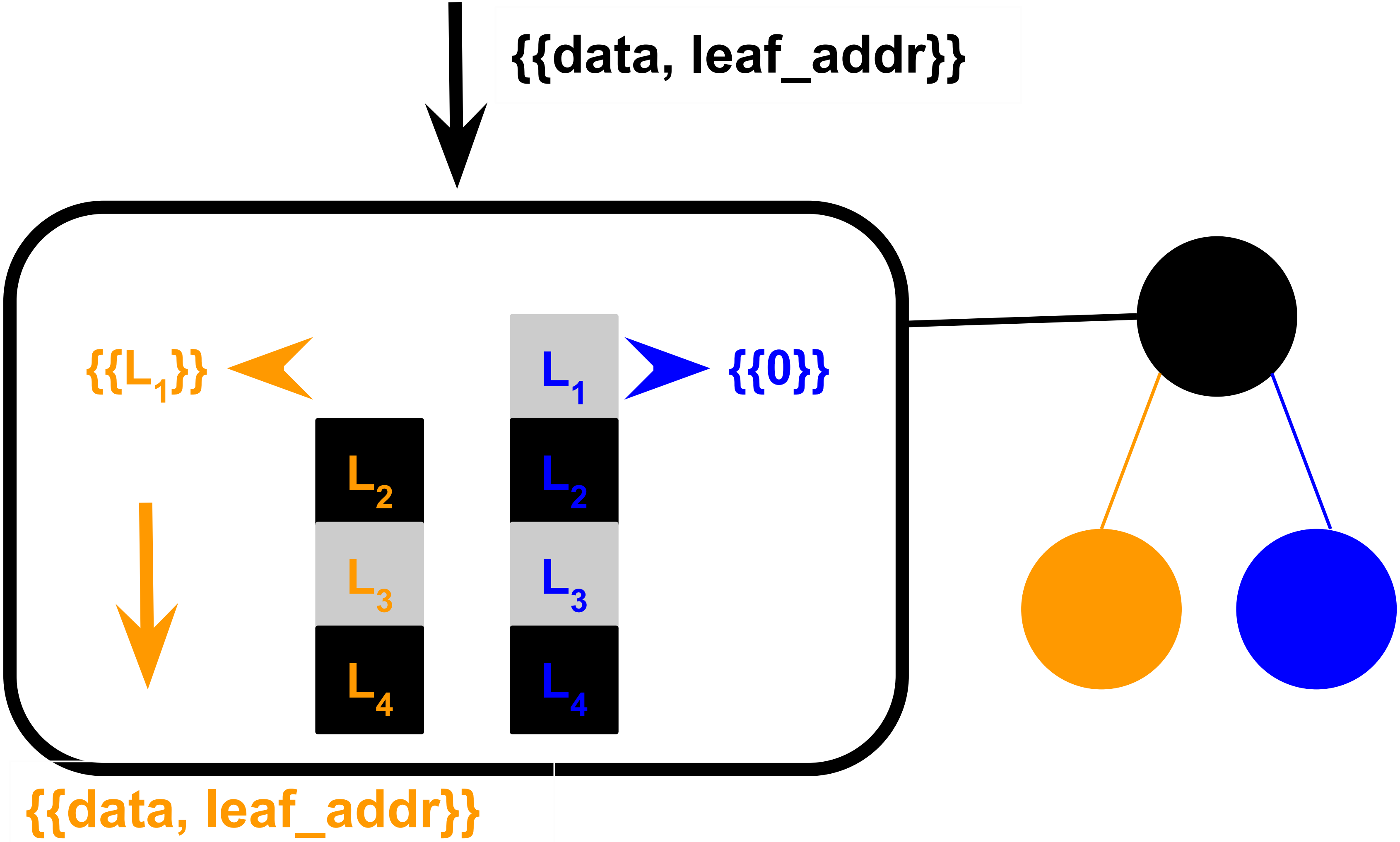




{{data, leaf_addr}}



{{data, leaf_addr}}

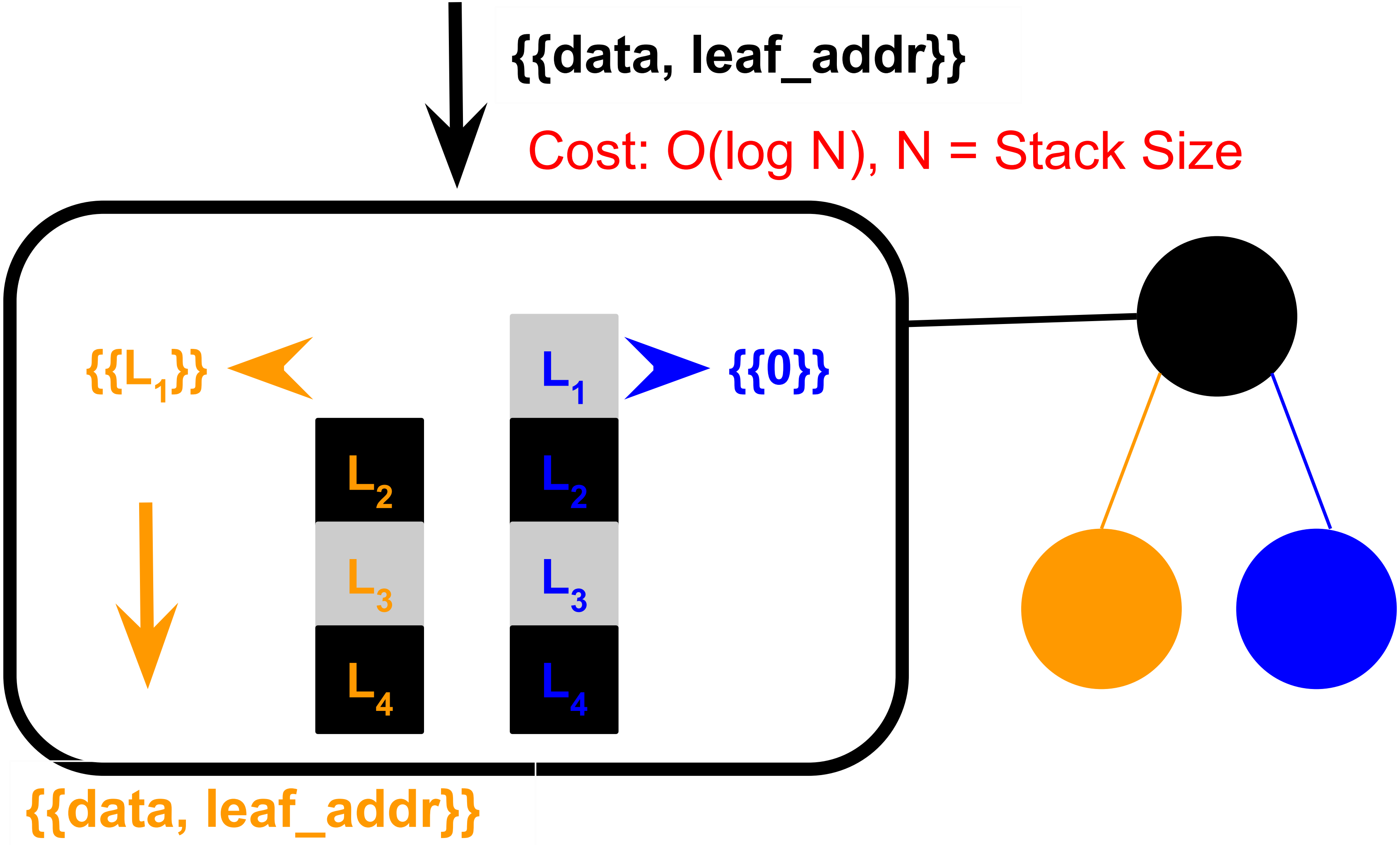


{{data, leaf_addr}}

{{L_1}}

{{0}}

{{data, leaf_addr}}



{{data, leaf_addr}}

Cost: O(log N), N = Stack Size

{{L_1}}

{{0}}

L_1

L_2 L_2

L_3 L_3

L_4 L_4

{{data, leaf_addr}}

Outline

The Language Translation Problem

Construction of [HKO21]

Our Techniques

Two Reasons of Inefficiency

Two Reasons of Inefficiency

⚠ Large Switches (i.e. Root Node) = Large Number of Access

Two Reasons of Inefficiency

⚠ Large Switches (i.e. Root Node) = Large Number of Access

⚠ Passing large payload length (O(log n) labels,λ bits long)

Our Contributions

Large Switches (i.e. Root Node) = Large Number of Access

Passing large payload length ($O(\log n)$ labels,λ bits long)

Our Contributions

Large Switches (i.e. Root Node) = Large Number of Access

Polylog sized buckets with dynamic finalization (Bucket ORAM)

Passing large payload length (O(log n) labels,λ bits long)

Our Contributions

Large Switches (i.e. Root Node) = Large Number of Access

Polylog sized buckets with dynamic finalization (Bucket ORAM)

Passing large payload length (O(log n) labels,λ bits long)

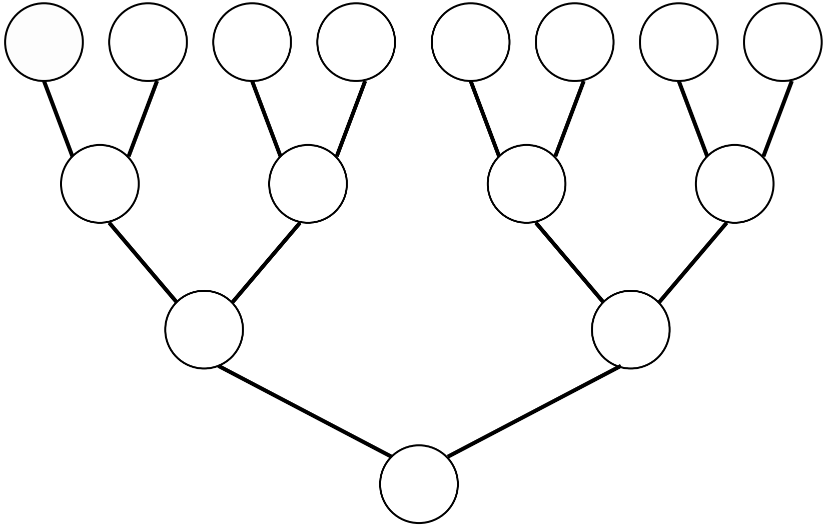
Passing Single Label Using XOR Trick (see paper)

Trick: Break down switches into smaller size

Each node at level i has T /(B · 2^i) copies of GSwitch + GBkt

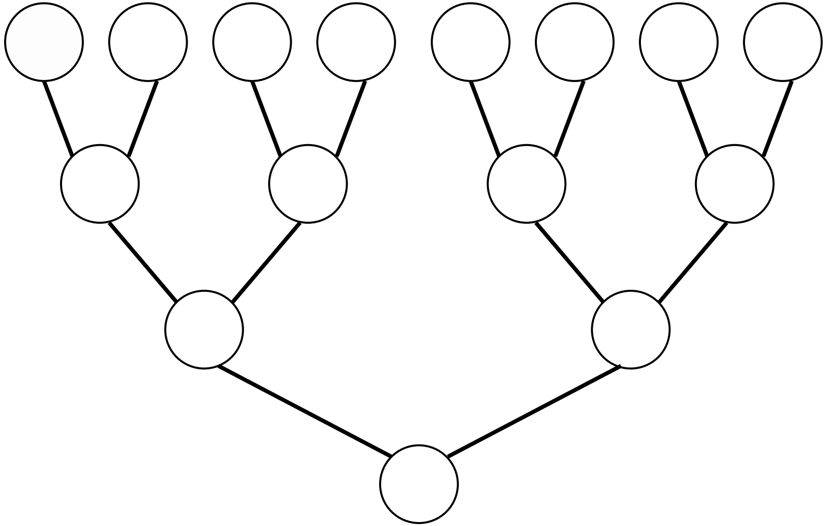
Trick: Break down switches into smaller size

Each node at level i has T /(B · 2^i) copies of GSwitch + GBkt



Trick: Break down switches into smaller size

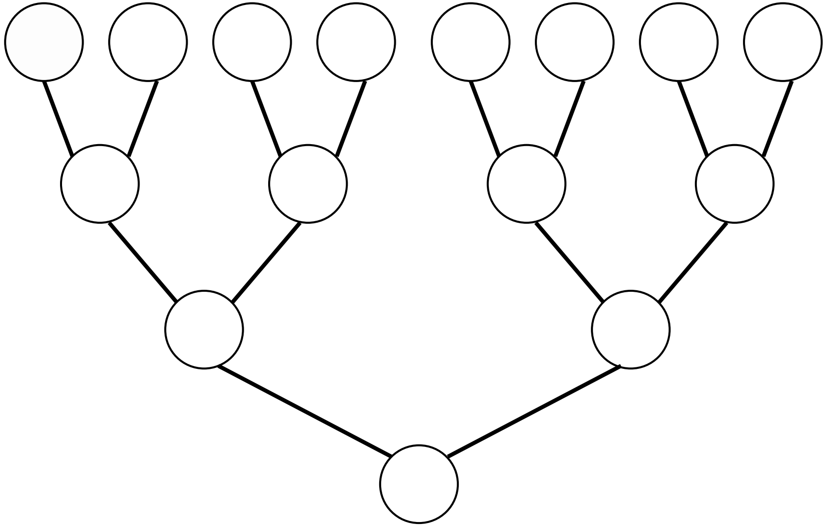
Each node at level i has T /(B · 2^i) copies of GSwitch + GBkt



Bucket are of size O(log N)

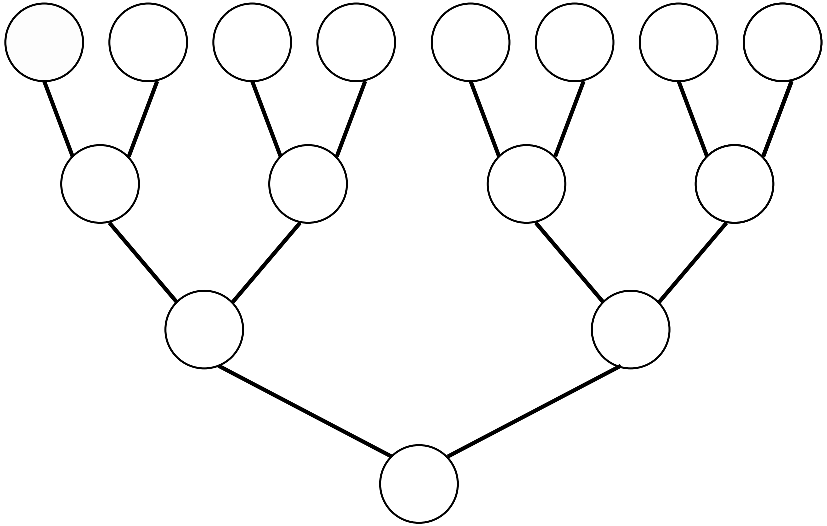
Trick: Break down switches into smaller size

Each node at level i has T /(B · 2^i) copies of GSwitch + GBkt



Trick: Break down switches into smaller size

Each node at level i has T /(B · 2^i) copies of GSwitch + GBkt



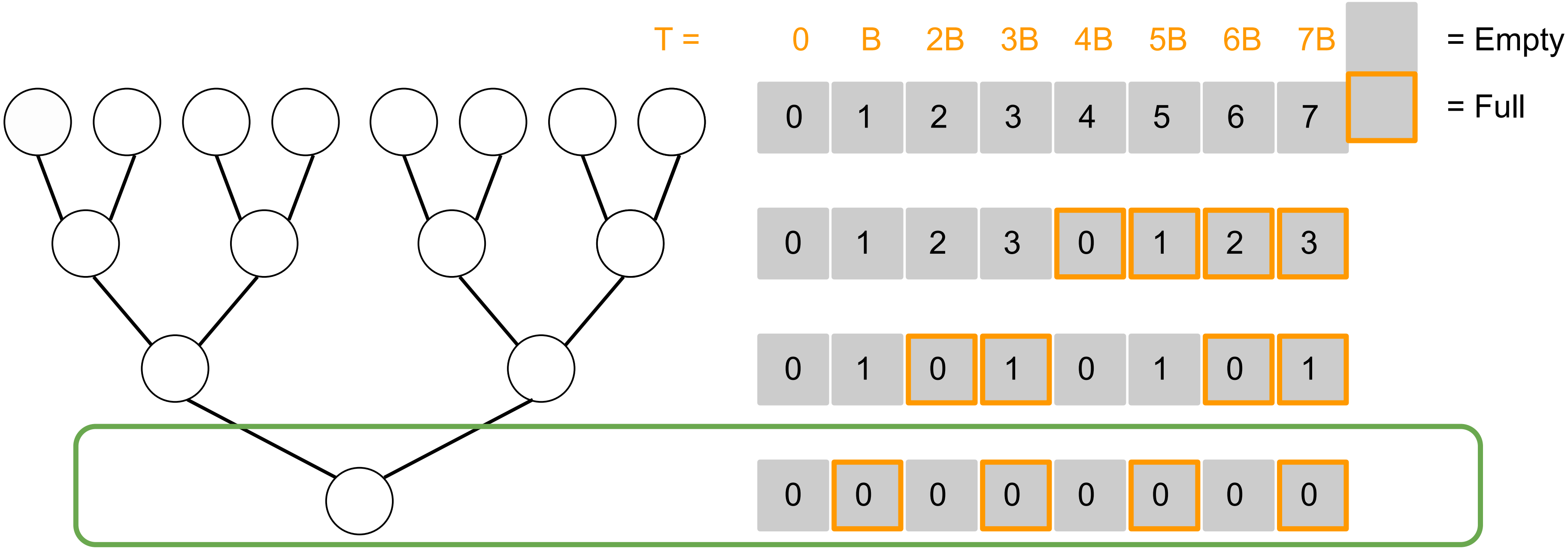
T =	0	B	2B	3B	4B	5B	6B	7B
	0	1	2	3	4	5	6	7
	0	1	2	3	▣ 0	▣ 1	▣ 2	▣ 3
	0	1	▣ 0	▣ 1	0	1	▣ 0	▣ 1
	0	▣ 0	0	▣ 0	0	▣ 0	0	▣ 0

□ = Empty

▣ = Full

Trick: Break down switches into smaller size

Each node at level i has T /(B · 2^i) copies of GSwitch + GBkt



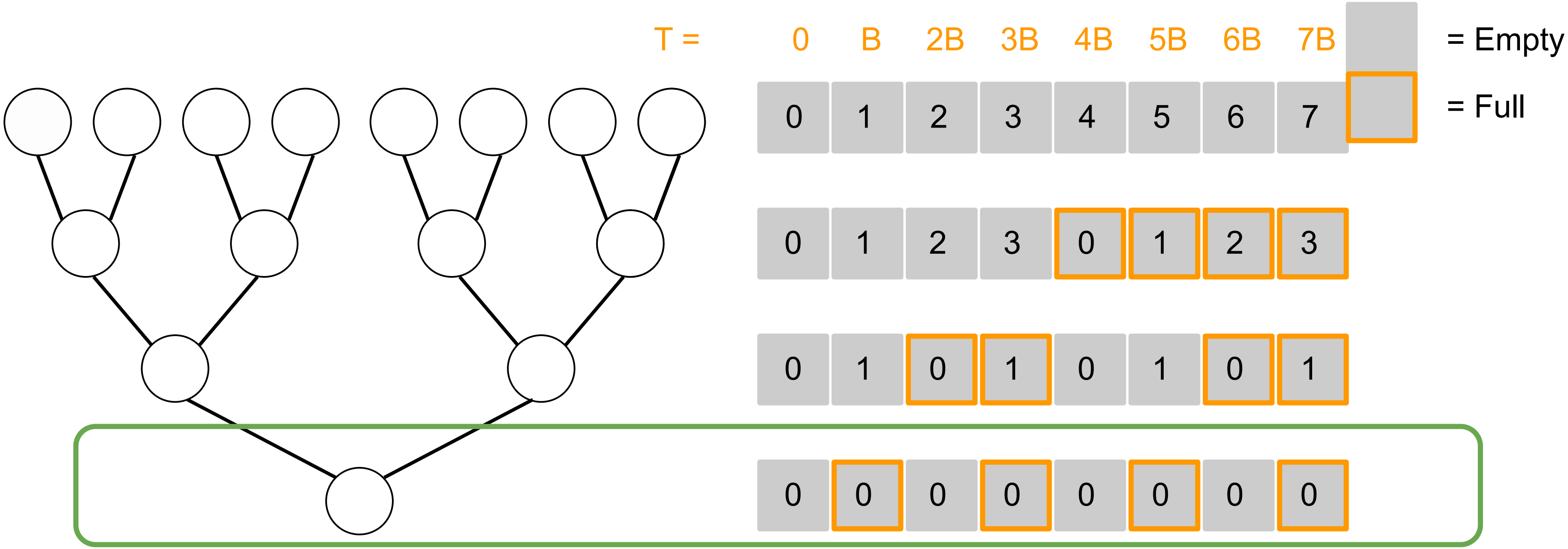
T =	0	B	2B	3B	4B	5B	6B	7B
	0	1	2	3	4	5	6	7
	0	1	2	3	0	1	2	3
	0	1	0	1	0	1	0	1
	0	0	0	0	0	0	0	0

■ = Empty

□ = Full

Trick: Break down switches into smaller size

Each node at level i has T /(B · 2^i) copies of GSwitch + GBkt

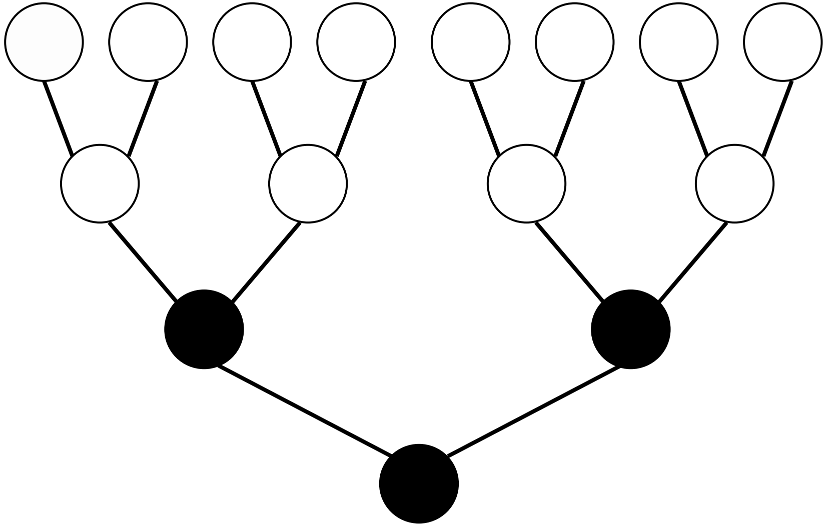


T =	0	B	2B	3B	4B	5B	6B	7B
	0	1	2	3	4	5	6	7
	0	1	2	3	0	1	2	3
	0	1	0	1	0	1	0	1
	0	0	0	0	0	0	0	0

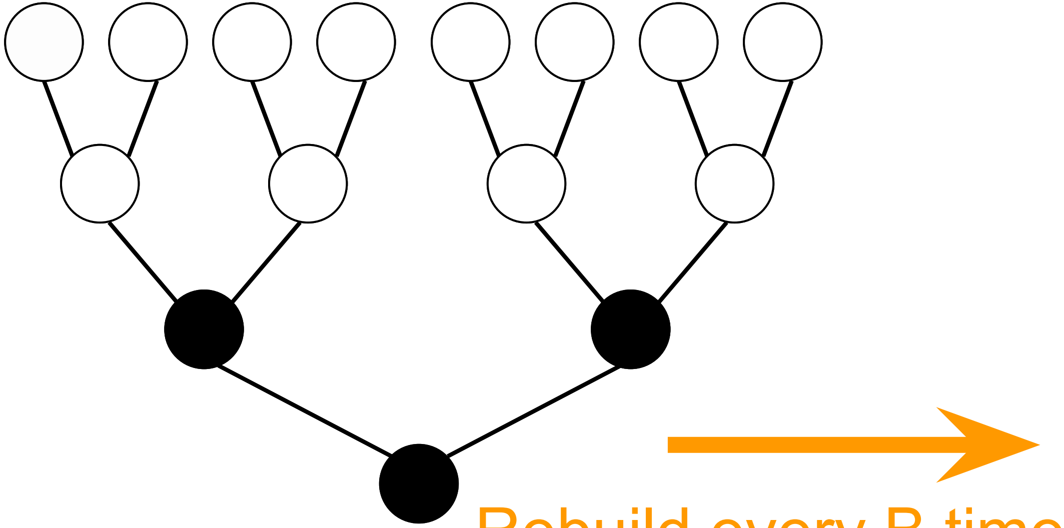
Break up O(N)-sized switch into O(log N)-sized

Rebuilding the Garbled Buckets

Rebuilding the Garbled Buckets

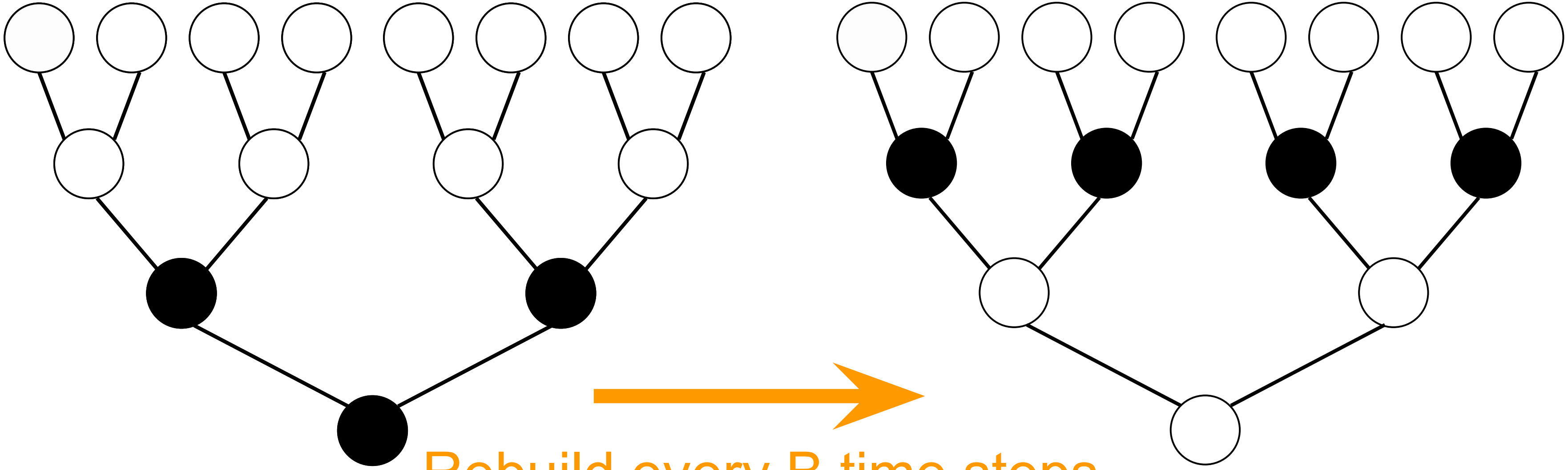


Rebuilding the Garbled Buckets



Rebuild every B time steps
B = Stash Size

Rebuilding the Garbled Buckets



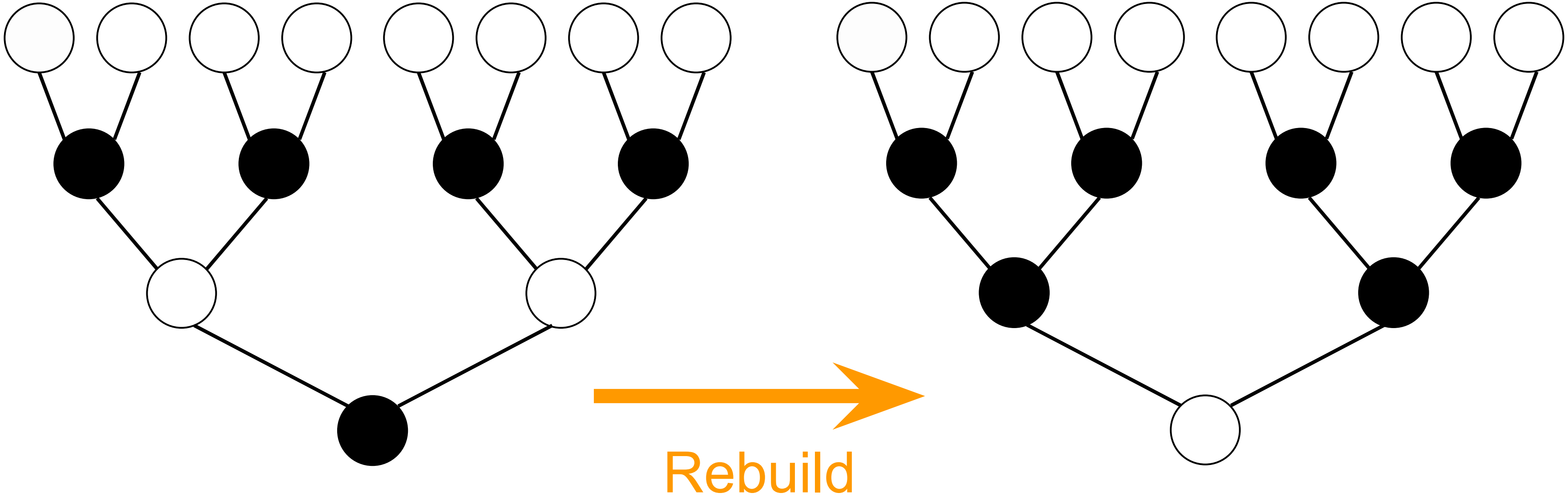
Rebuild every B time steps
B = Stash Size

Issue: Dynamic Rebuild

Accesses are unknown at garbling time

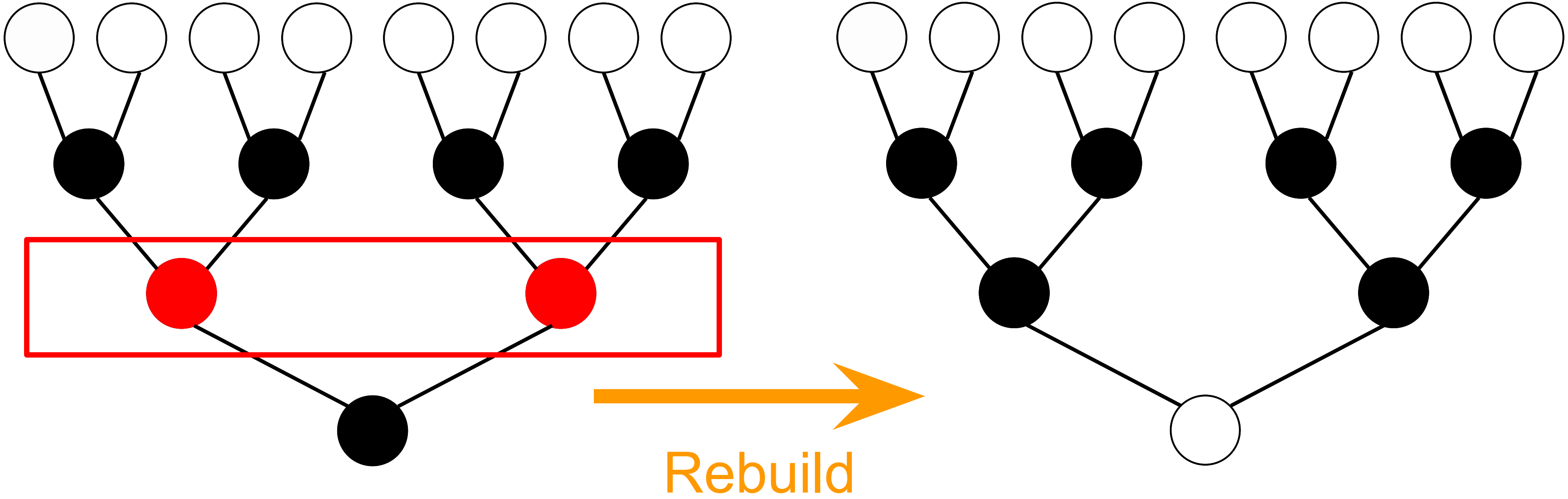
Issue: Dynamic Rebuild

Accesses are unknown at garbling time



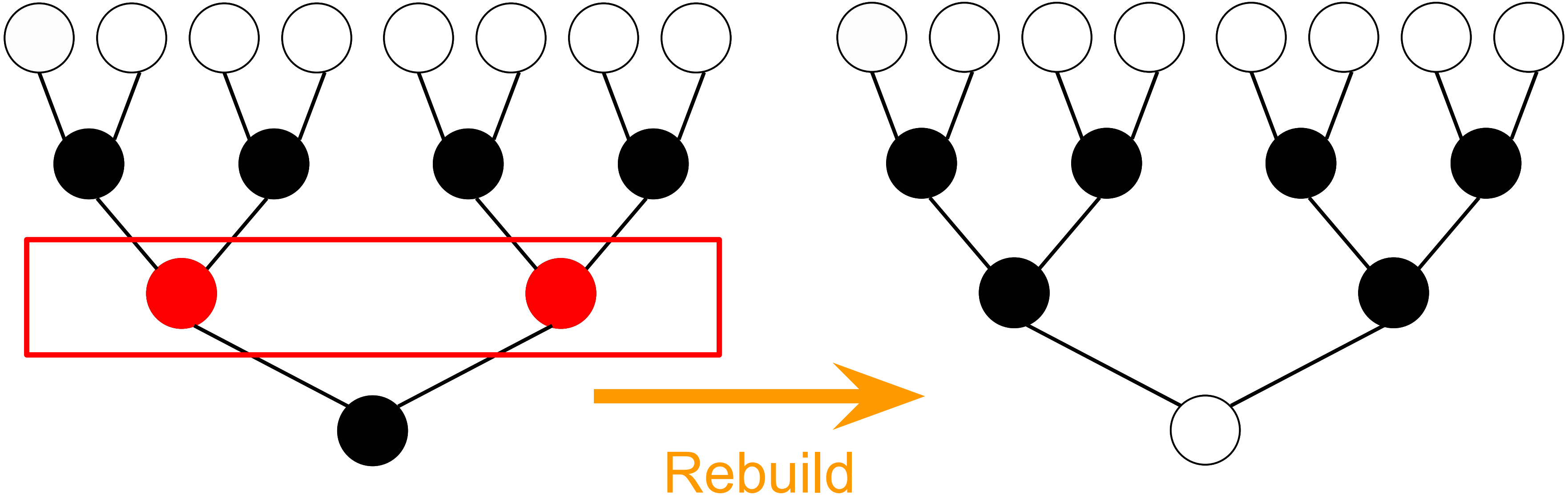
Issue: Dynamic Rebuild

Accesses are unknown at garbling time



Issue: Dynamic Rebuild

Accesses are unknown at garbling time



Rebuild

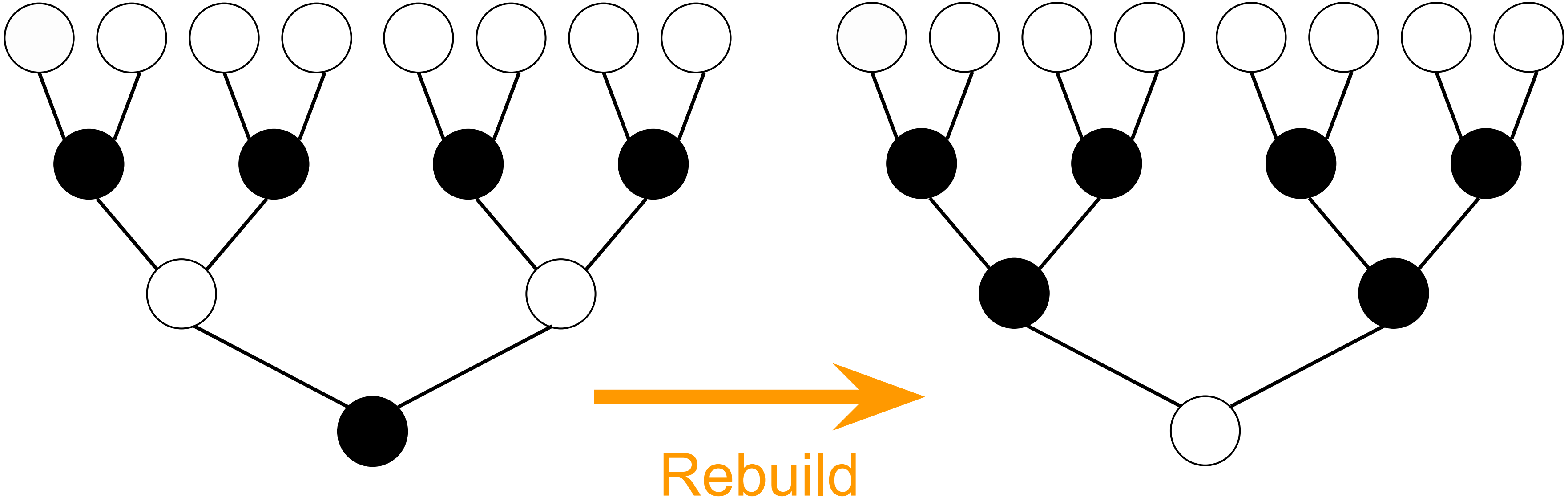
Local clocks of children have advanced to unknown dynamic value

Solution: Dynamic Finalization

Equip Garbled data structures with Finalize routine

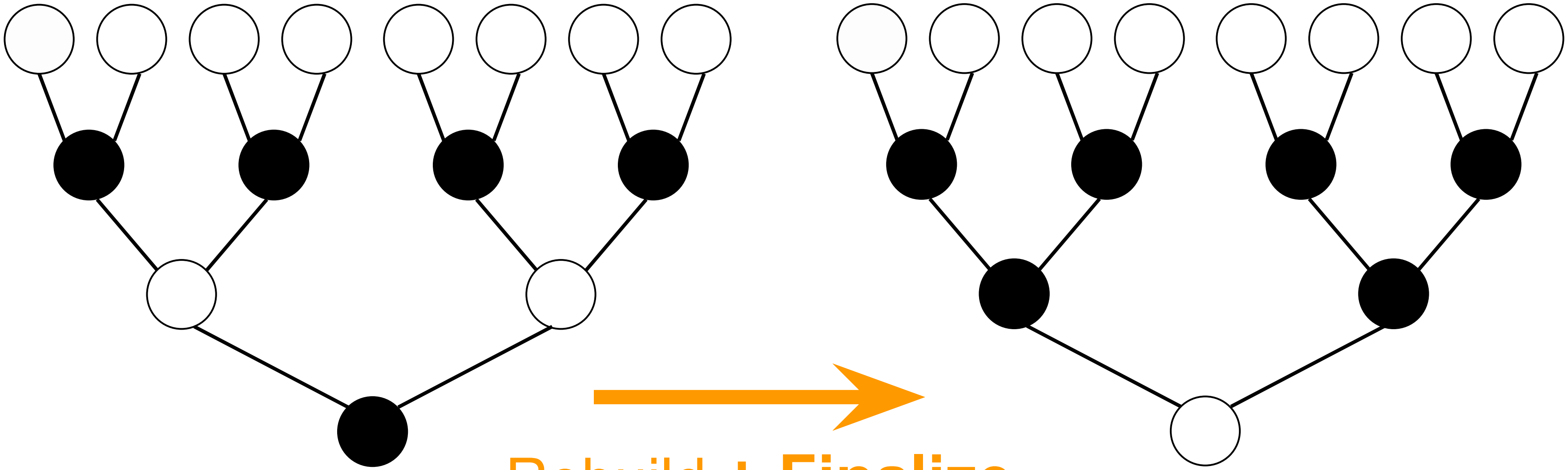
Solution: Dynamic Finalization

Equip Garbled data structures with Finalize routine



Solution: Dynamic Finalization

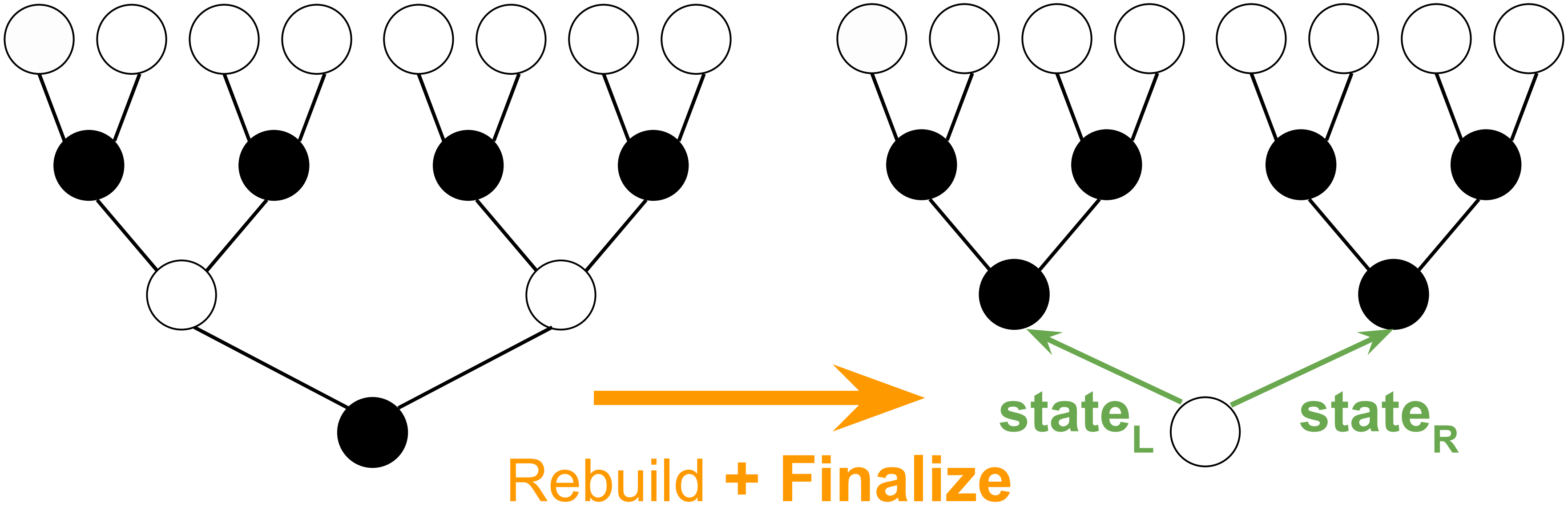
Equip Garbled data structures with Finalize routine



Rebuild **+ Finalize**

Solution: Dynamic Finalization

Equip Garbled data structures with Finalize routine



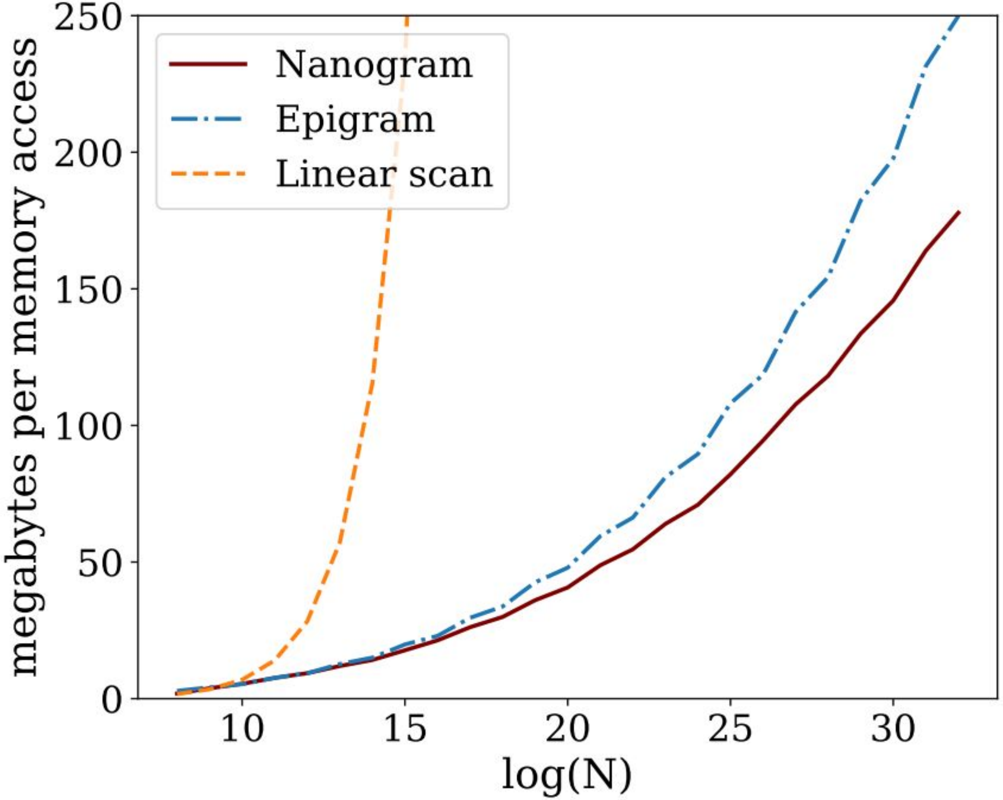
Additional Optimizations (See Paper)

Avoiding λ factor blowup when garbling

Modular framework for garbled algorithm composition

Practical Optimizations

Concrete Performance



THANK YOU!

https://eprint.iacr.org/2022/191.pdf