Oblivious Computation
Part I - Lower Bounds and Tree Based ORAMs

Gilad Asharov
Bar-Ilan University

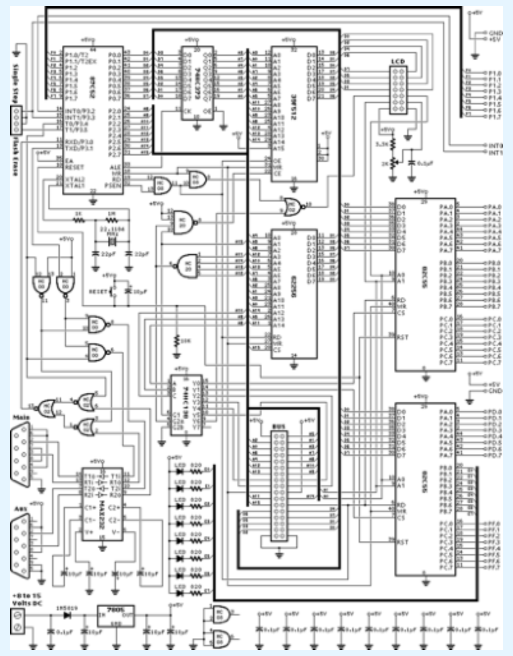
Some slides were created by: **Elaine Shi**, **Ilan Komargodski**

The 12th Bar-Ilan Winter School on Cryptography
Advances in Secure Computation

Center for Research in Applied Cryptography and Cyber Security

Models of Computation

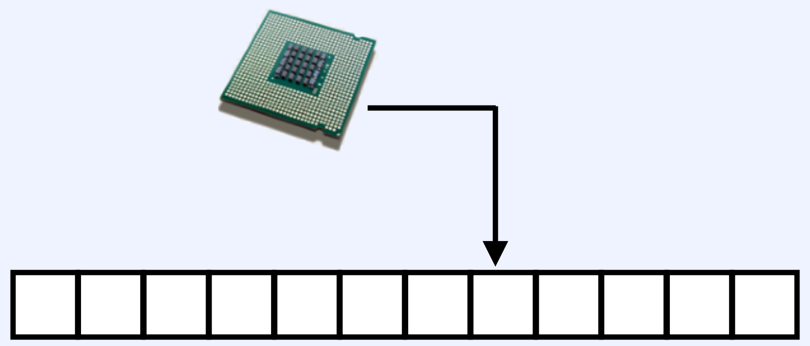
Circuits



Emulate easily →

← $T^3 \log T$
[CR73,PF79]
$(T \gg N)$

RAM Model



Random Access Machine

CPU Operation
Memory access

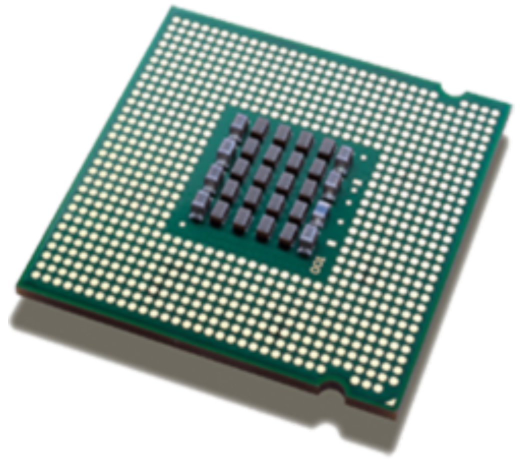
Metrics:

Size (how many wires, gates)
Depth (parallelism)

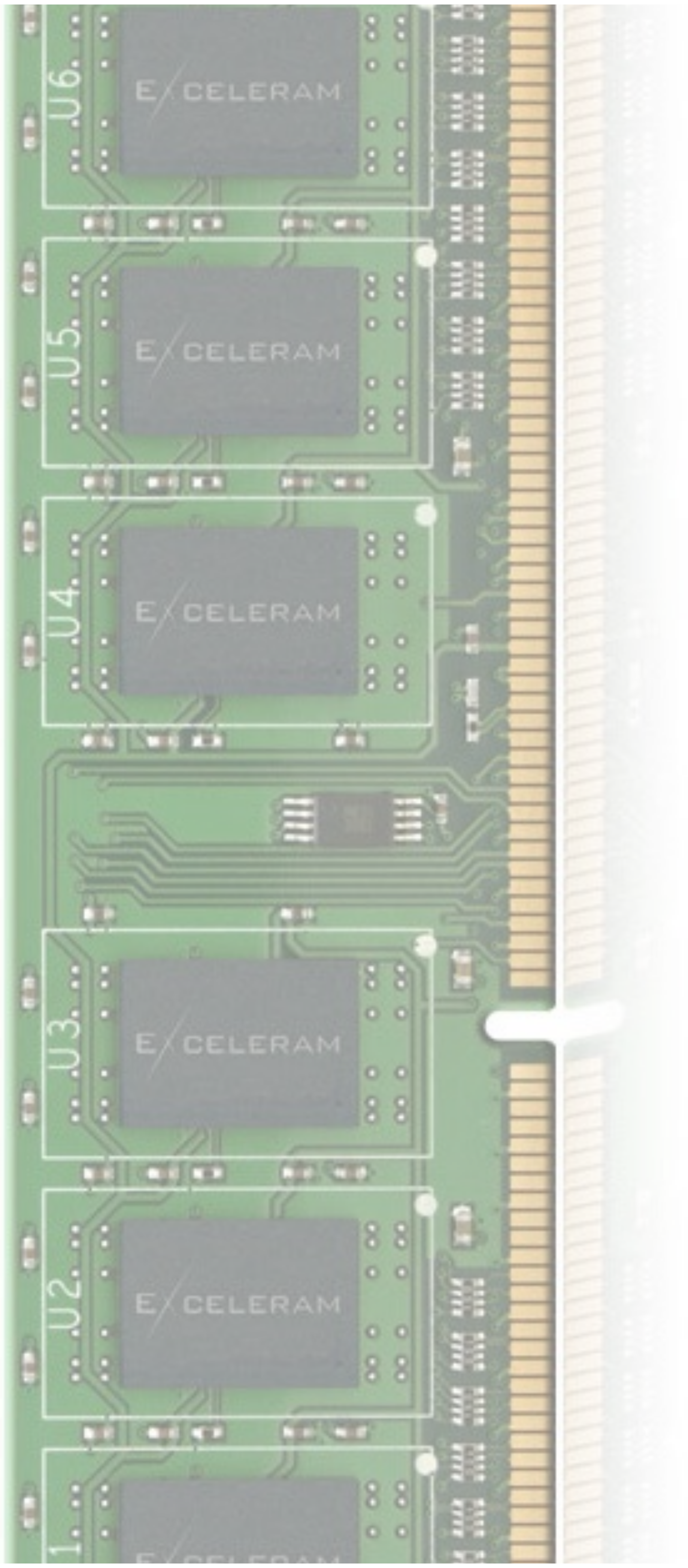
Time	T
Size of the memory	N

Center for Research in Applied Cryptography and Cyber Security

Access Patterns Reveal Information!



secure processor



Access Patterns Reveal Information!

```
func search(val, s, t)
  mid = (s+t)/2
  if val<mem[mid]
    search(val,0,mid)
  else search(val, mid+1, t)
```

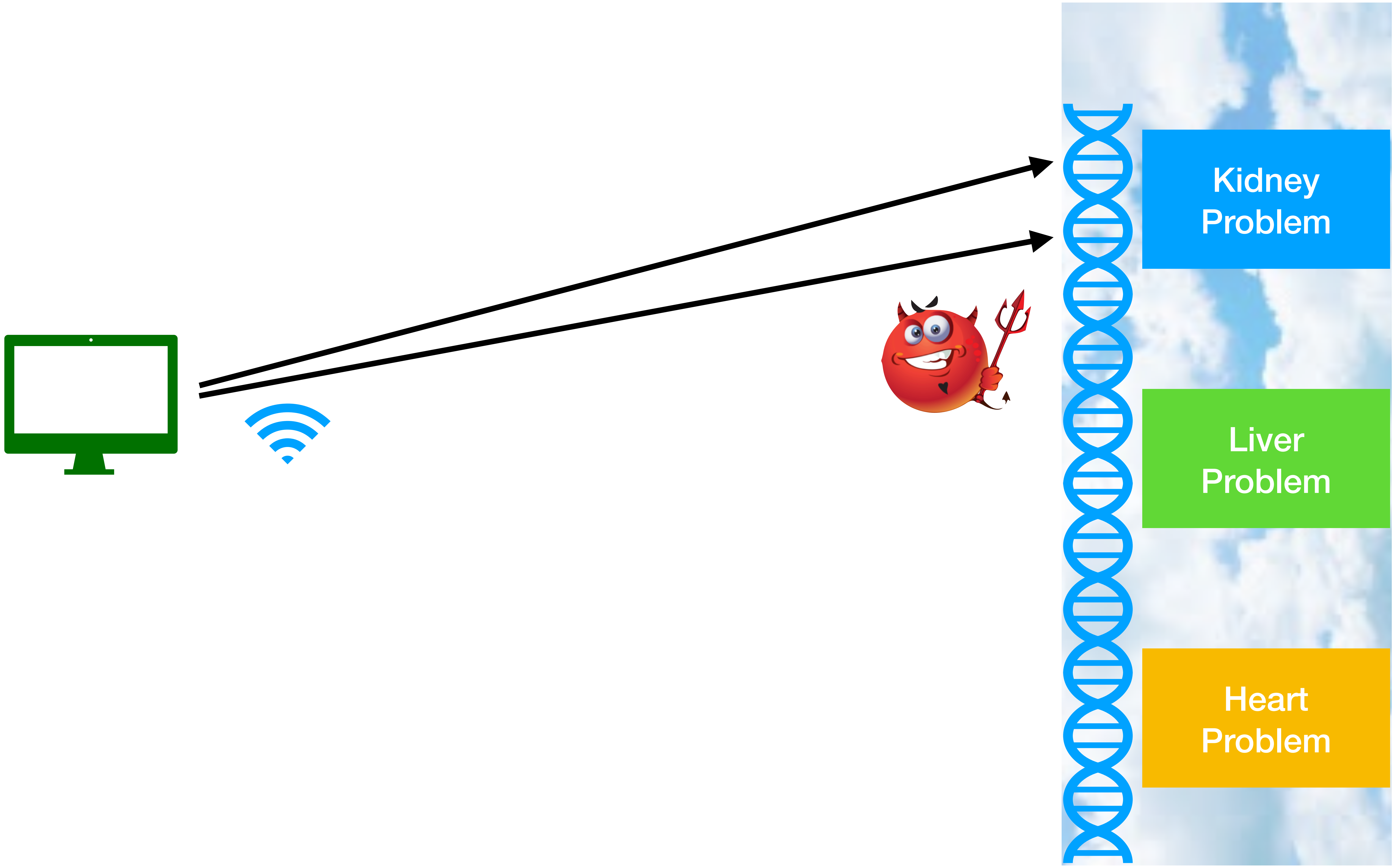
Access Pattern of **binary search** leaks the **rank** of the number being searched

Access Patterns Reveal Information!

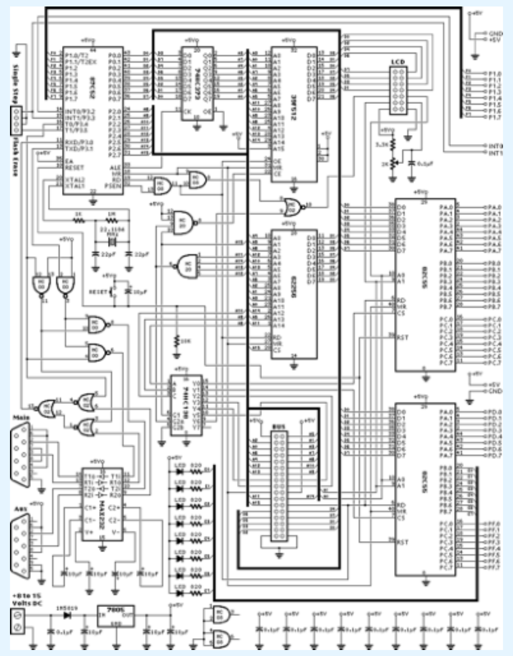
```
if (secret variable)
   Read mem[x]
else
   Write mem[y]
```

Access pattern reveals the value of the **secret variable**

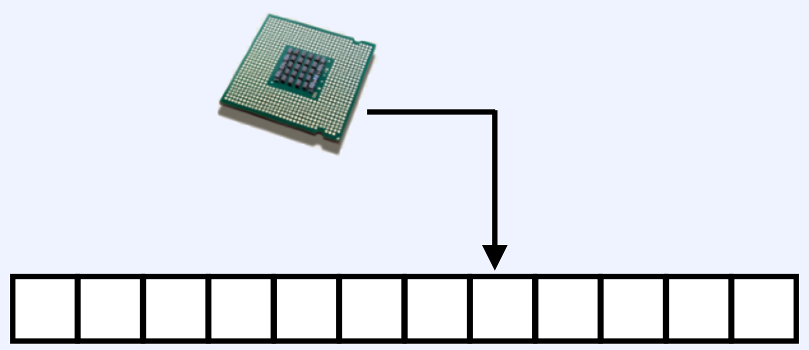
Access Patterns Reveal Information!



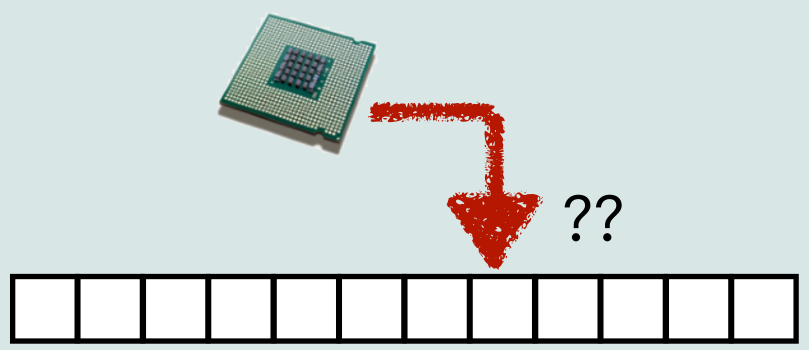
Circuits



RAM Model



Oblivious RAM Model



A program in the RAM model
Access Pattern is “oblivious”:
Can be **simulated** from (T,N)

Example: Sorting

- **Merge sort:** $O(n \log n)$
  - non oblivious
- **Bubble sort:** $O(n^2)$
  - oblivious

Merge((1,2,3),(4,5,6)**)**

1,2,3 4,5,6
1,2,3 4,5,6
1,2,3 4,5,6
1,2,3 4,5,6
1,2,3 4,5,6
1,2,3 4,5,6

Merge((1,3,5),(2,4,6)**)**

1,3,5 2,4,6
1,3,5 2,4,6
1,3,5 2,4,6
1,3,5 2,4,6
1,3,5 2,4,6
1,3,5 2,4,6

BubbleSort(1,2,3,4**)**

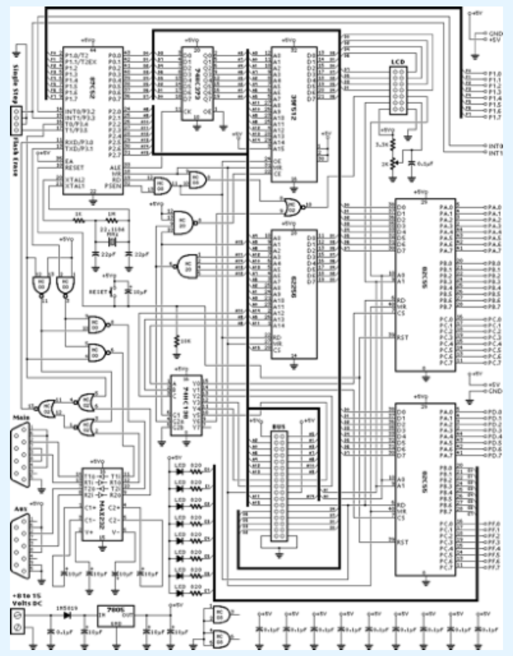
1,2,3,4
1,2,3,4
1,2,3,4
1,2,3,4
1,2,3,4
1,2,3,4
1,2,3,4

BubbleSort(4,3,2,1**)**

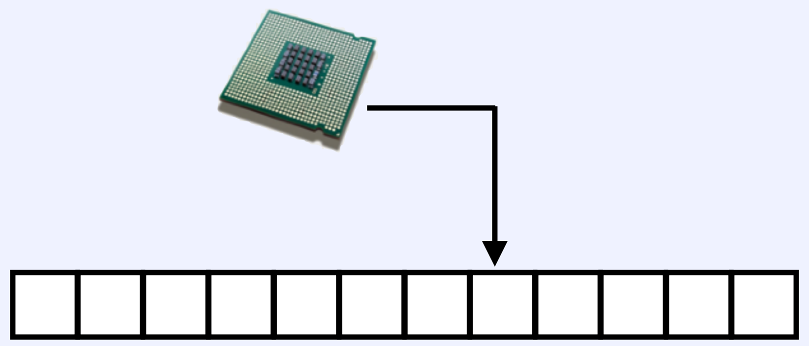
4,3,2,1
3,4,2,1
3,2,4,1
3,2,1,4
2,3,1,4
2,1,3,4
1,2,3,4

Models of Computation

Circuits

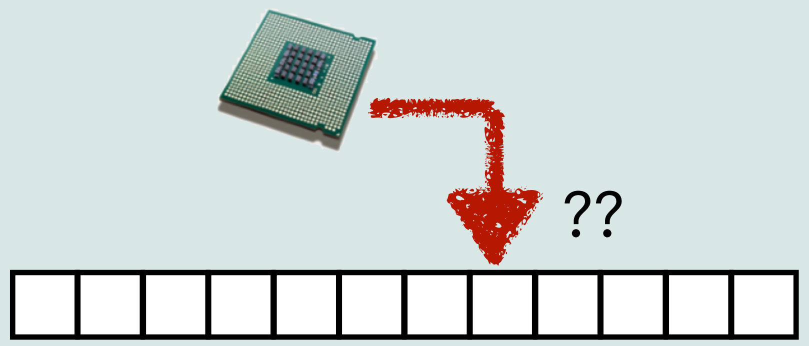


RAM Model



Oblivious RAM Compiler

Oblivious RAM Model



A program in the RAM model
Access Pattern is "oblivious":
Can be simulated from (T,N)

Usually, $N' = O(N)$
T'/T : **overhead** of the compilation

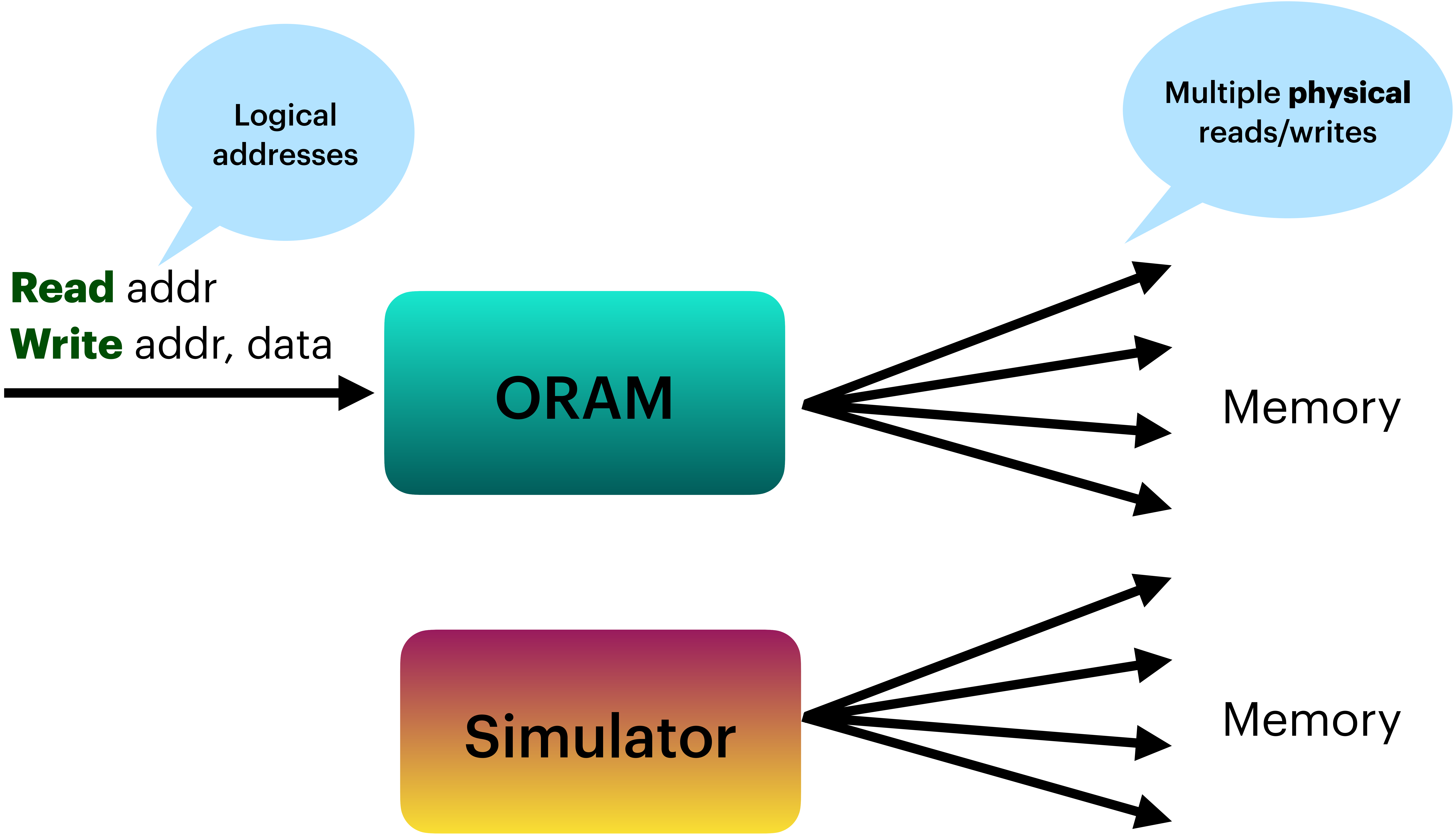
Oblivious RAM Compiler	RAM Program with (T,N) —> Oblivious RAM Program with (T', N')
Trivial Compiler	RAM Program with (T,N) —> Oblivious RAM Program with (TN, N)

Oblivious RAM (ORAM)

An algorithmic technique that **provably** encrypts access patterns

Goldreich and Ostrovsky (87′,90′,96′)

📍 **Permuting** and **shuffling** elements around the memory



Security: *Physical accesses independent of input logical sequence*

ORAM schemes

Cloud computing:

Shroud: [RPMRS, Fast'12]
Metal: [CP, NDSS'20]
Ring ORAM: [RFKSS+, SEC'15]
ObliviStore: [SS, S&P'13]
S3ORAM: [HOY, CCS'17], [HYG'19]
TaoStore: [SZALT, S&P'16]
O. R. ORAM: [CCR, CCS'19]
Obliviate: [AKSL, NDSS'18]
Others: [WNLCS+, CCS'14], [BNPWH, CCS'15]

Theoretical crypto:

[GHL+, Eurocrypt'14], [GHR+, FOCS'14], [GLO, FOCS'15], [GLOS, STOC'15], [BCP, TCC'16], [CLT, TCC'16], [DDFRSW, TCC'16], [LO, CRYPTO'17], [CCS, Asiacrypt'17], [CNS, TCC'18], [CKNPS, Asiacrypt'18], [CL, TCC'19]

Programming lang:

[LHS, CSF'13],
[DSLH, POPL'20]

Database:

Obladi: [CBCHAA, OSDI'18]
ObliDB: [EZ, VLDB'20]

Architecture, secure processor:

OpenPiton: [BMFN+, CACM'19]
Phantom: [MLSTS+, CCS'13]
Ghostrider: [LHMHTS, ASPLOS'15, Best Paper]
Ascend: [RFK+, TDSC'19] , [FRY+, HCPA'14],
Raccoon: [LRT, SEC'15]
Klotski: [ZSYZSJ, ASPLOS'20]
ZeroTrace: [SGF, NDSS'18]
Obscuro: [AJX+, NDSS'19]
Others: [HO+, PETS'19], [HB+, CODASPY'20] [RRM, C&S'20]

Multi-party computation:

ObliVM: [WHCSS, CCS'14], [LWNHS, S&P'15] [NWIWTS, S&P'15],
ObliVC: [ZE'15]
SPDZ: [KY, Eurocrypt'18]
Others: [GKK+, CCS'12], [GHJR, ACNS'15], [Keller'17], [GKW, Asiacrypt'18]

Blockchain, ML, misc:

Blockchain: [CZJKJS, CCS'17]
Proof of retrievability: [CKW, Eurocrypt'13]
Privacy-preserving ML: [NWIWTS, S&P'15], [WLNHS, S&P'15]

[GO'87,90,96]

Hierarchical ORAM

$O(\sqrt{N})$

$O(\log^3 N)$

[GM'11,KLO'12]

Hierarchical ORAM

$\approx O(\log^2 N)$

[SCSL'11, SDS+13, WCS'15]

Tree Based ORAM

$O(\log^3 N) \Longrightarrow O(\log^2 N)$

Simple, small constants

Statistical

[PPRY'18, AKL+'20]

Hierarchical ORAM

$O(\log N)$

Matching the lower bound!

(Big constant)

Computational

[GO'87,90,96] [LN'18]

Lower Bound

$\Omega(\log N)$

Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$

[GoldreichOstrovsky'96, LarsenNeilsen'18]

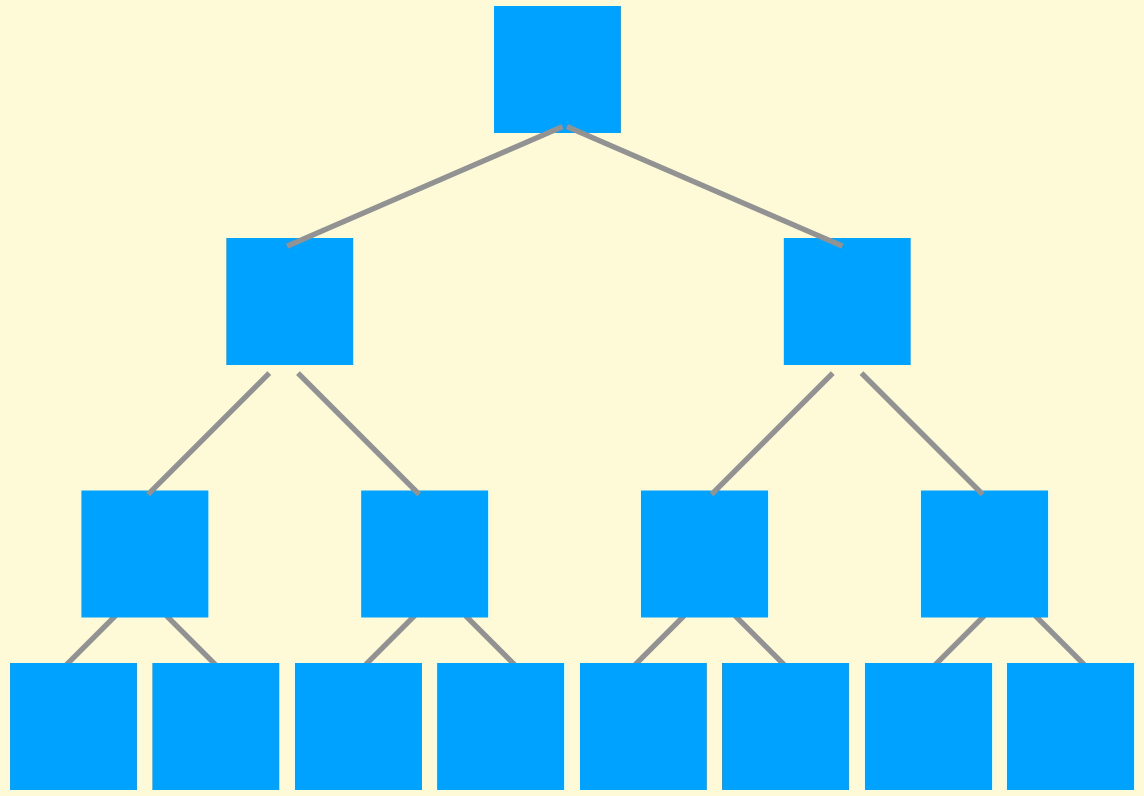
Hierarchical

[O90,GO96]

$O(\log N)$

Computational security

[OptORAMa,AKLNPS'20]



Tree based ORAM

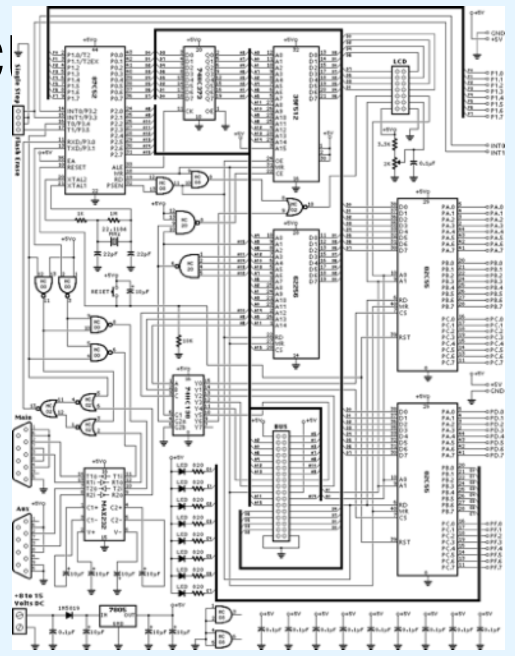
[Shi,Chan,Stefanov11]

$O(\log^2 N)$

Statistical security

[PathORAM,CircuitORAM]

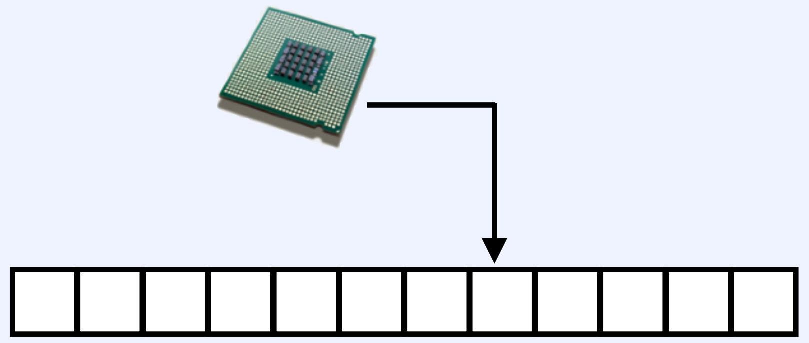
Circ



No dynamic memory accesses A[i]

Oblivious
Parallelism

RAM Model

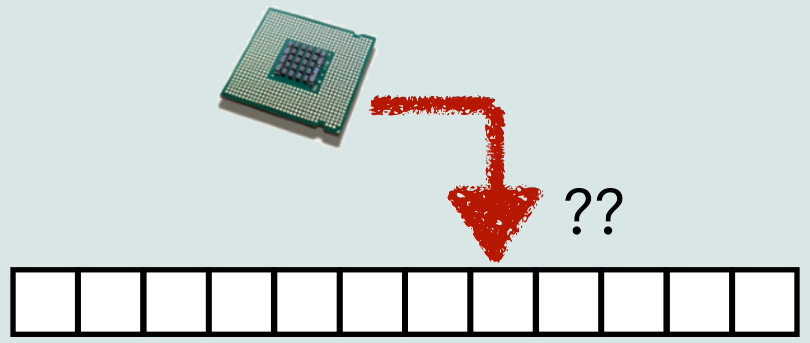


Dynamic memory accesses A[i]

Not Oblivious
No Parallelism

Oblivious RAM Compiler

Oblivious RAM Model



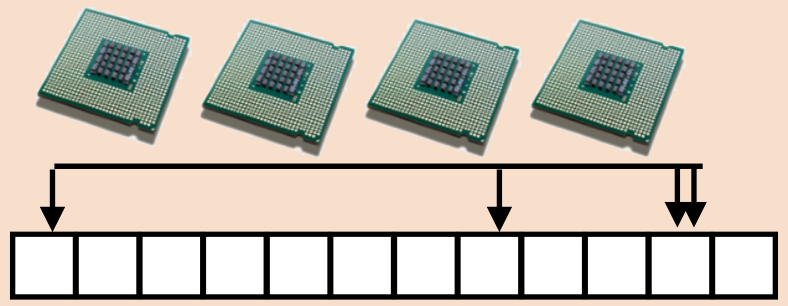
Dynamic memory accesses A[i]

Oblivious
No Parallelism

<u>Oblivious PRAM compiler:</u>
Introduced by Boyle, Chung and Pass in 2016

<u>Recent work [**A**KLPS, SODA'22]:</u>
Any PRAM program with T parallel time and N space
$\Longrightarrow T \log N$ parallel time and N space

Parallel RAM Model

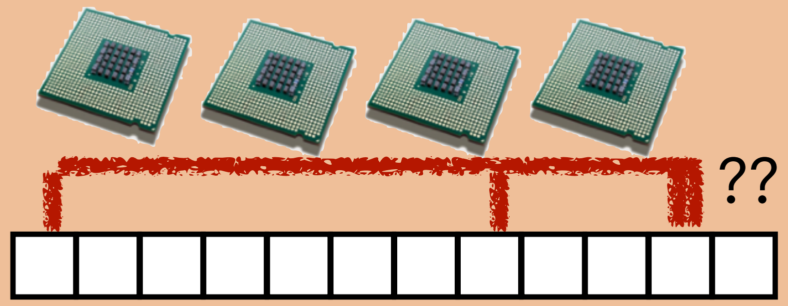


Dynamic memory accesses A[i]

Not Oblivious
Parallelism

Oblivious **P**RAM Compiler

Oblivious Parallel RAM Model



Dynamic memory accesses A[i]

Oblivious
Parallelism

Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$

[GoldreichOstrovsky'96, LarsenNeilsen'18]

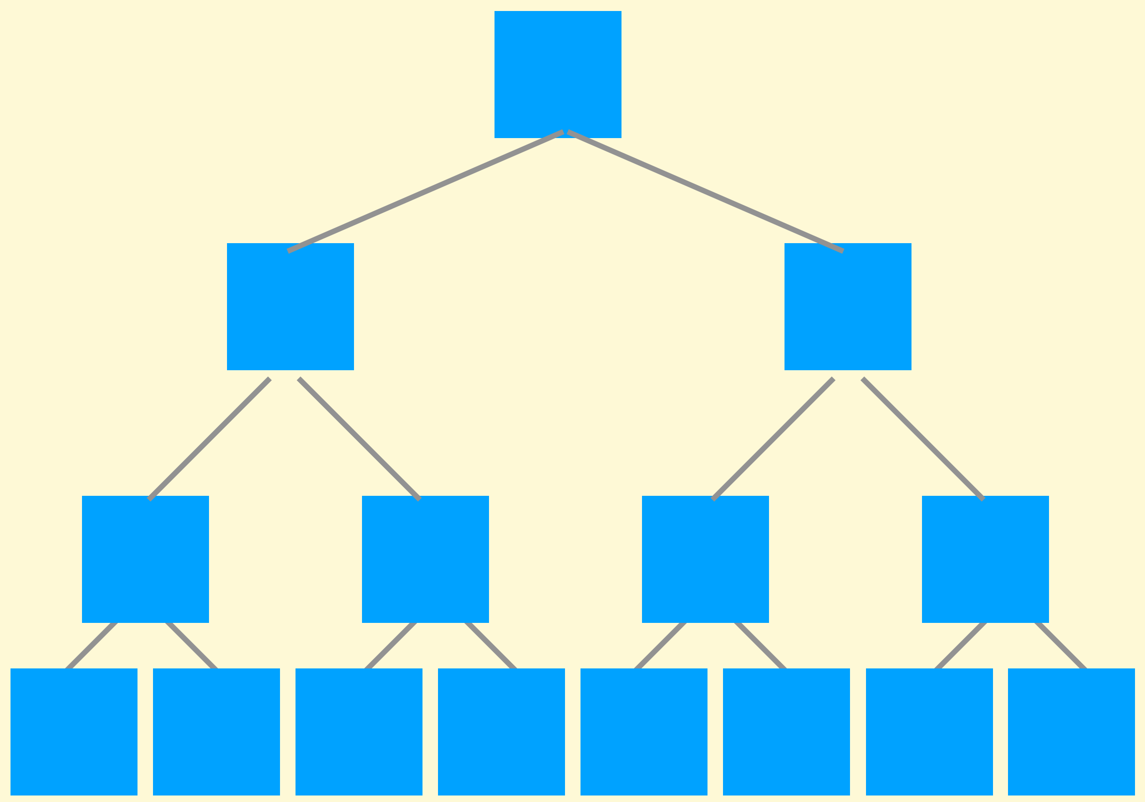
Hierarchical

[O90,GO96]

$O(\log N)$

Computational security

[OptORAMa,AKLNPS'20]



Tree based ORAM

[Shi,Chan,Stefanov11]

$O(\log^2 N)$

Statistical security

[PathORAM,CircuitORAM]

Lower Bounds

Any ORAM compiler results in $\Omega(\log N)$ overhead

Lower Bounds

Goldreich and Ostrovsky ['96]:

$\Omega(\log N)$

- **Balls and Bins** model
- Statistical Security
- **Offline** ORAM

Counting argument

Boyle and Naor ['16]:

An $\Omega(\log N)$ lower bound for **offline** ORAM **not** in **the balls and bins model** implies an $\Omega(N \log N)$ lower bound for **sorting circuits**

Larsen and Nielsen ['18]:

$\Omega(\log N)$

- **Not** in **Balls and Bins model**
- Computational Security
- ***Online*** ORAM

Information transfer technique

Offline ORAM: the entire logical sequence is known in advance; including all addresses **and data**

The Lower Bound [LN'18]

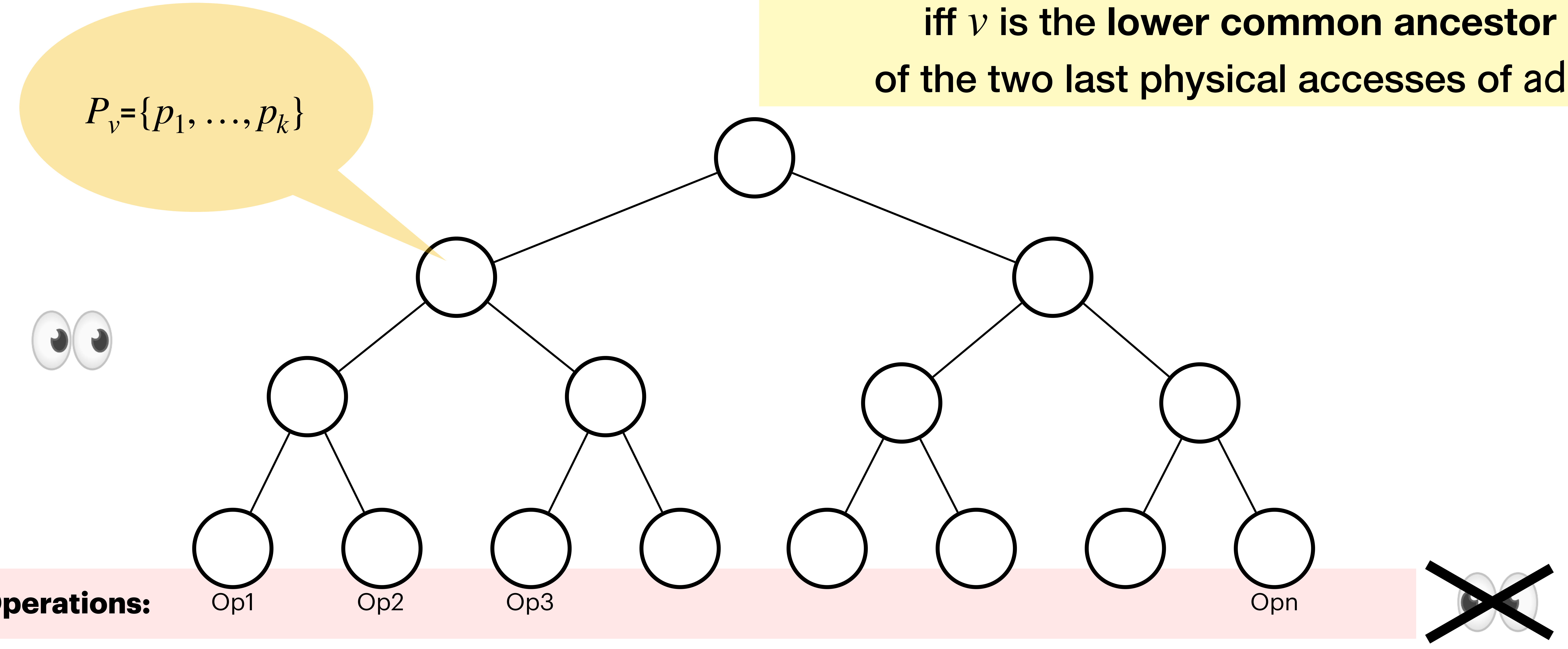
- Based on **information transfer** technique of Pătrașcu & Demaine '06
- Cell probe model [Yao'81] - computation is free, only charge for probes

The Lower Bound [LN'18]

Assign $p_i^j = (\text{Read/Write}, \text{addr})$ to an internal node v

iff v is the **lower common ancestor**

of the two last physical accesses of addr

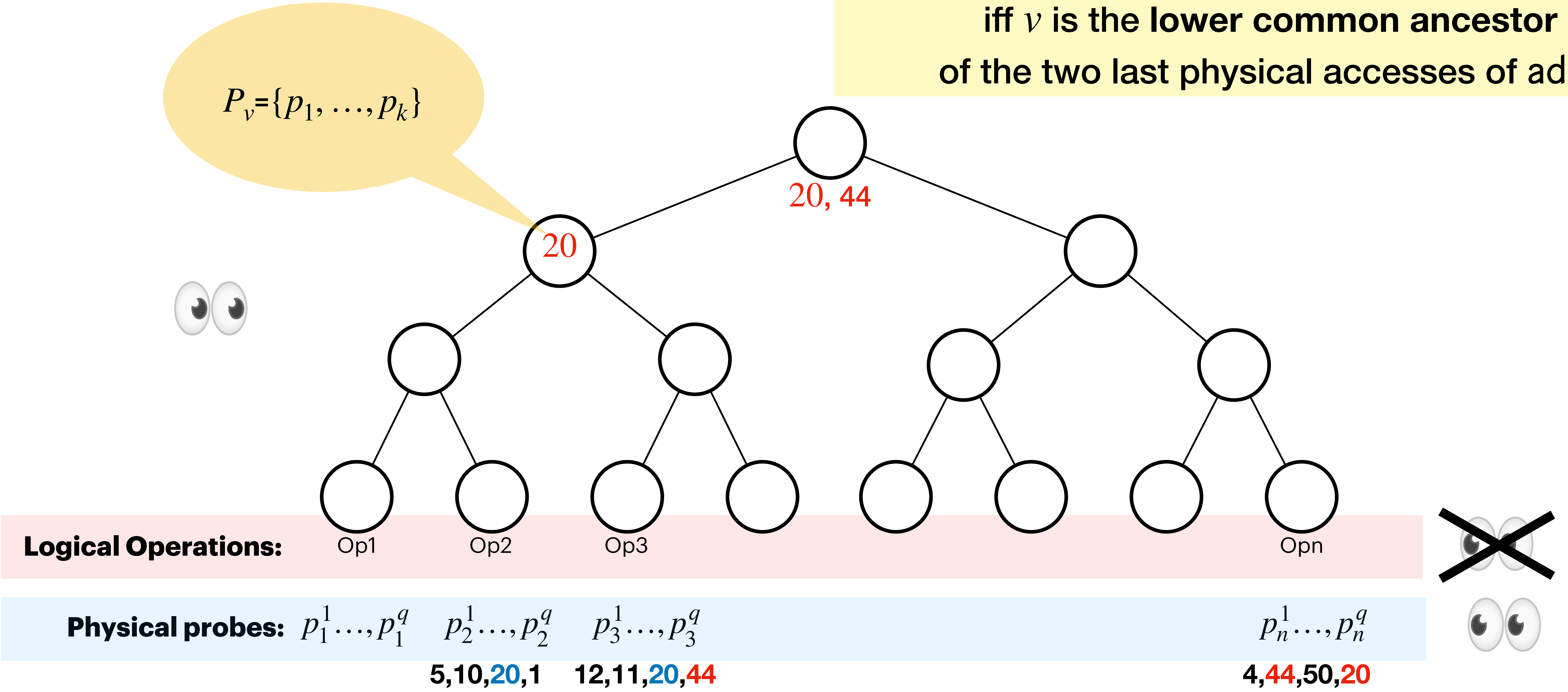


Logical Operations: Op1 Op2 Op3 Opn

Physical probes: $p_1^1 \ldots, p_1^q$ $p_2^1 \ldots, p_2^q$ $p_3^1 \ldots, p_3^q$ $p_n^1 \ldots, p_n^q$

Example

Assign $p_i^j = (\text{Read/Write, addr})$ to an internal node v iff v is the **lower common ancestor** of the two last physical accesses of addr



Example

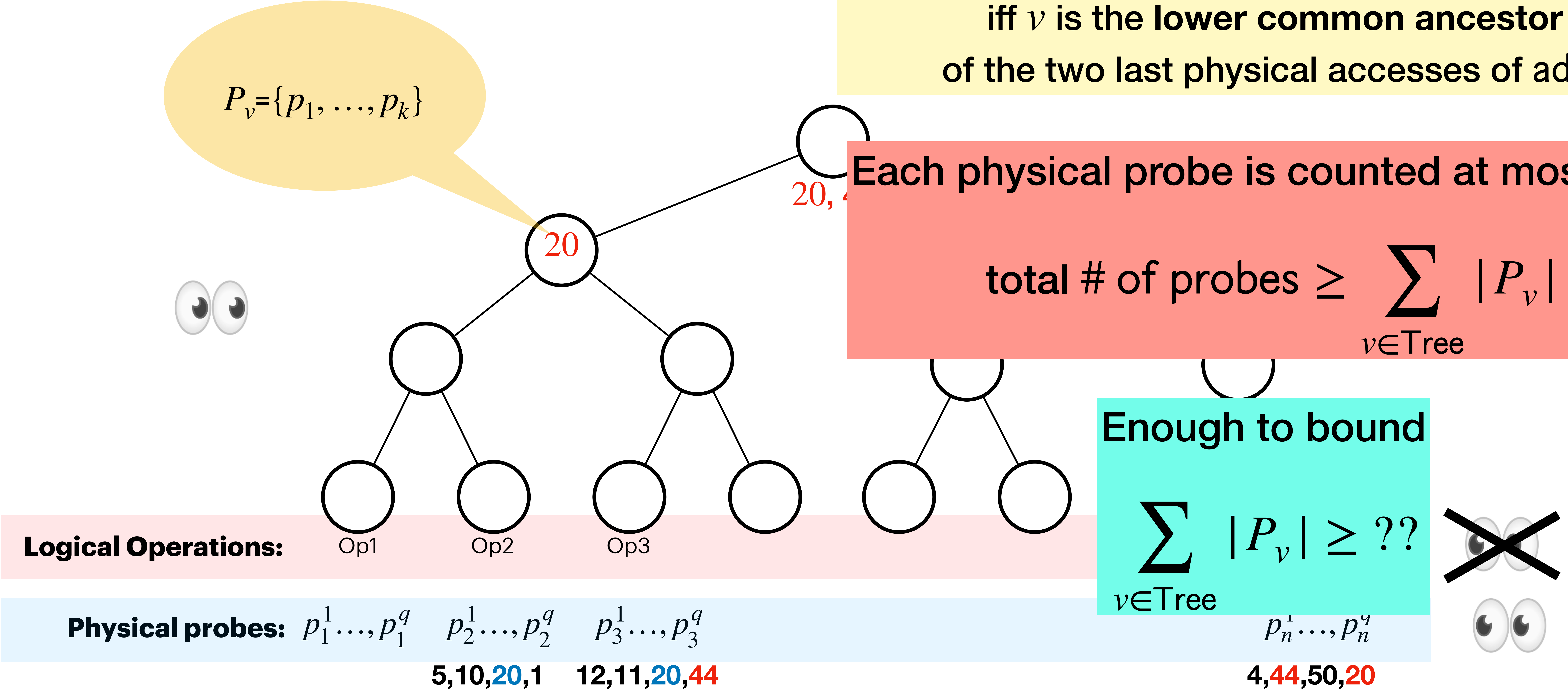
Assign $p_i^j = (\text{Read/Write}, \text{addr})$ to an internal node v iff v is the **lower common ancestor** of the two last physical accesses of addr

Each physical probe is counted at most once

$$\text{total \# of probes} \geq \sum_{v \in \text{Tree}} |P_v|$$

Enough to bound

$$\sum_{v \in \text{Tree}} |P_v| \geq ??$$

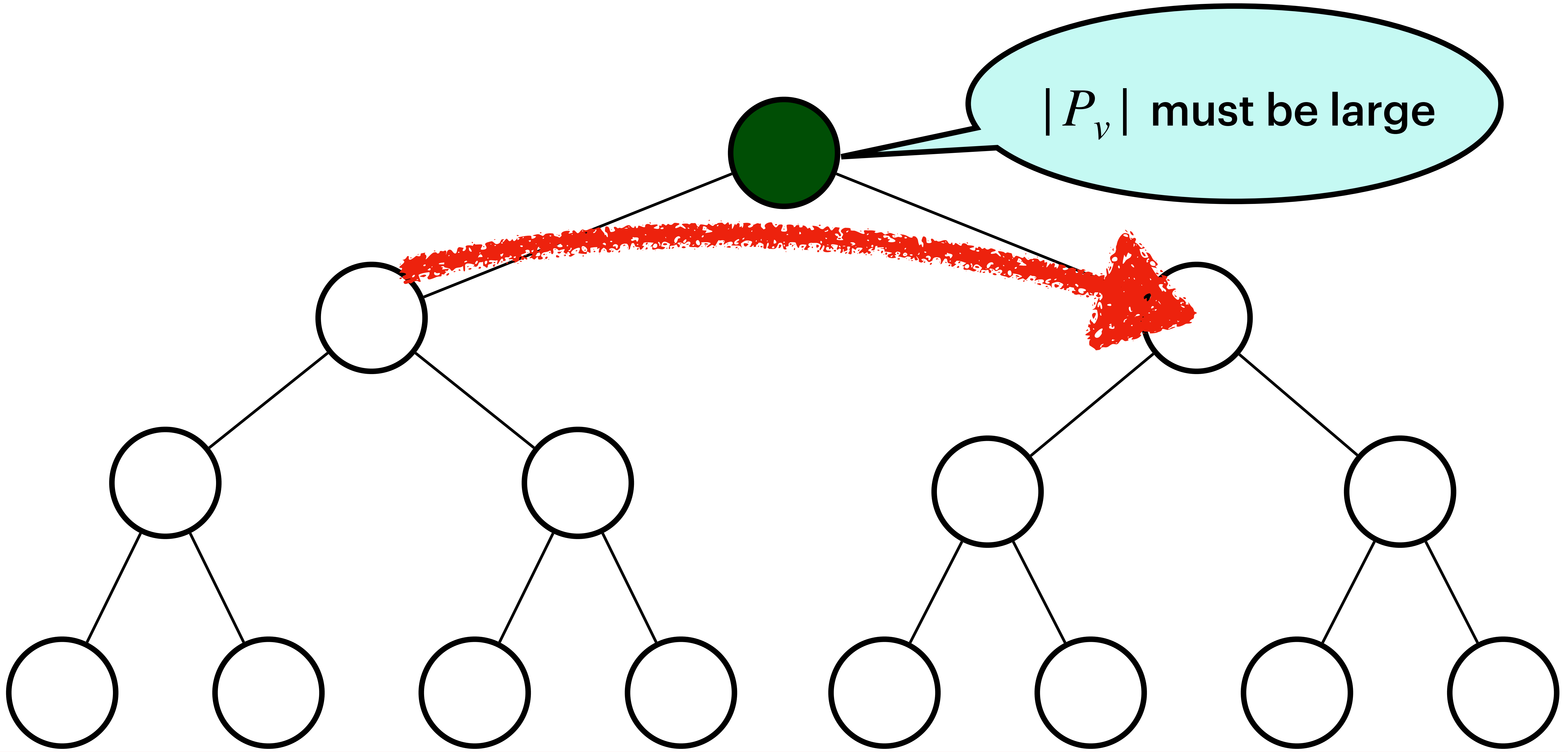


Based on the **physical access pattern** - the adversary can compute the tree

Security: For all logical sequences, for all v, $|P_v|$ should be similar

For every v, we can show a **logical sequence** forcing $|P_v|$ to be large

Assumes **online** ORAM



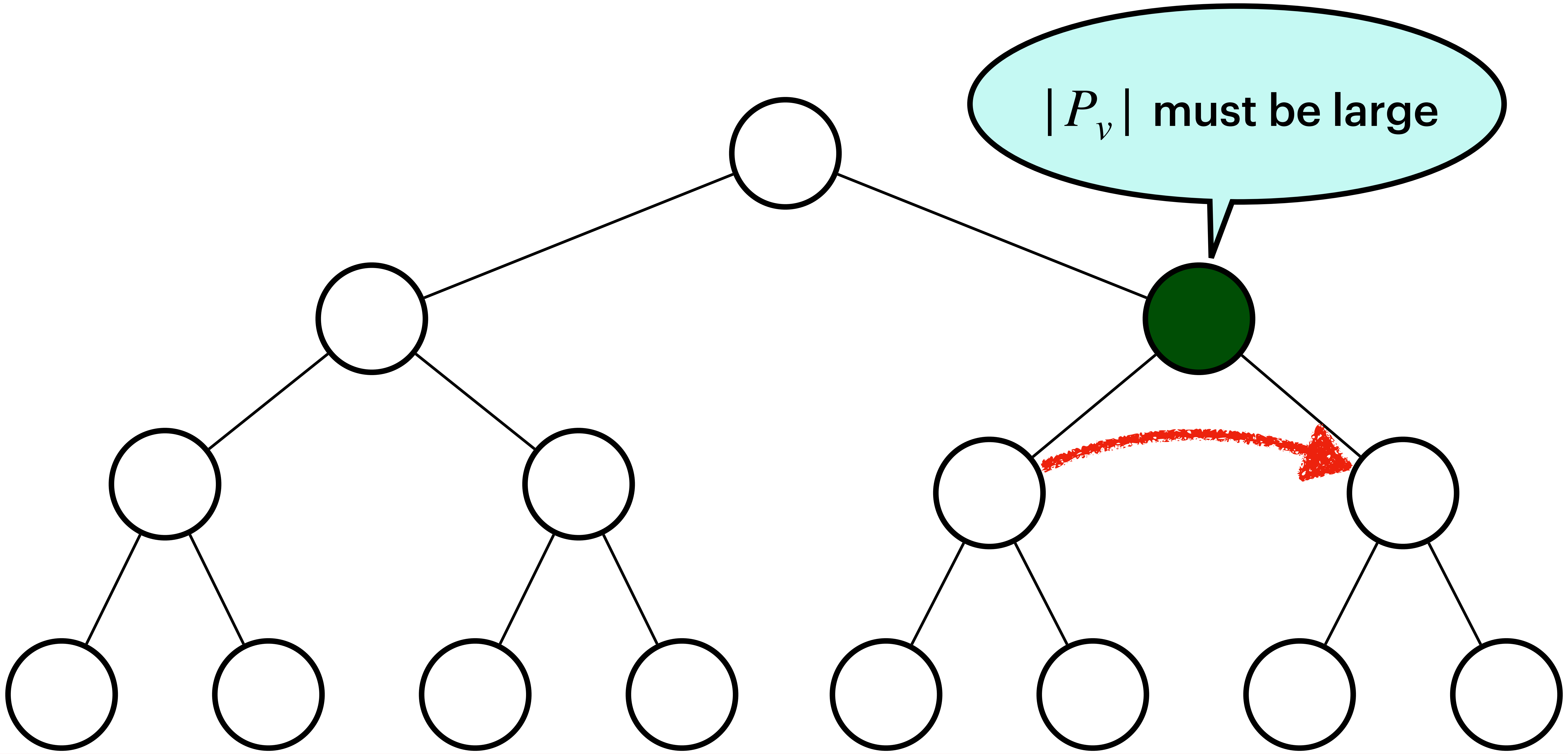
Logical Operations: **Write**(1,r1) **Write**(2,r2) **Write**(3,r3) **Write**(4,r4) **Read**(1) **Read**(2) **Read**(3) **Read**(4)

Based on the **physical access pattern** - the adversary can compute the tree

Security: For all logical sequences, for all v, $|P_v|$ should be similar

For every v, we can show a **logical sequence** forcing $|P_v|$ to be large

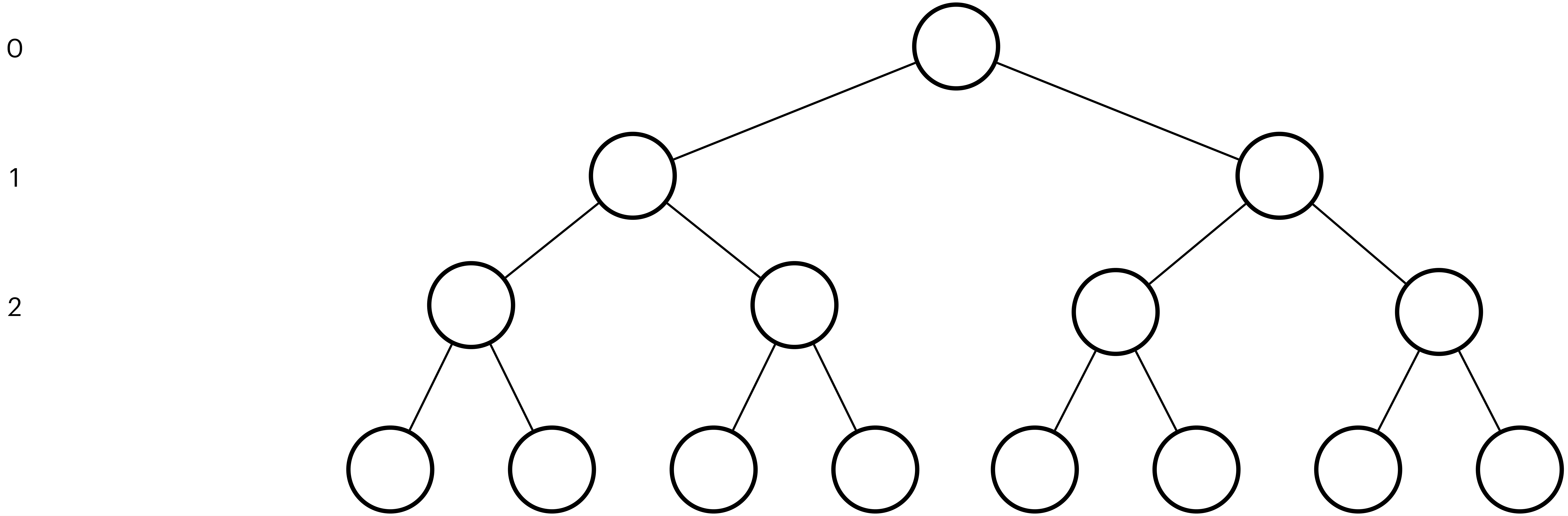
Assumes **online** ORAM



Logical Operations: **Write**(1,r1) **Write**(2,r2) **Read**(1) **Read**(2)

Claim: For every node in depth d, $E[|P_v|] \geq \frac{N}{2^d}$

Proof by encoding / compression argument



Logical Operations: **Write**(1,r1) **Write**(2,r2) **Read**(1) **Read**(2)

Claim: For every node in depth d, $E[|P_v|] \geq \frac{N}{2^d}$

E[total #of probes] $\geq \sum_{v \in Tree} E[|P_v|] = \sum_{v \in Tree} \frac{N}{2^d} = \sum_{d=0}^{\log N - 1} 2^d \cdot \frac{N}{2^d} = N \log N$

We considered **logical sequences** of length N

$\Omega(\log N)$ overhead per operation (in expectation)

References

Goldreich and Ostrovsky: **Software Protection and Simulation on Oblivious RAMs,** JACM 1996

Boyle and Naor: **Is There an Oblivious RAM Lower Bound?** ITCS 2016

Larsen and Nielsen: **Yes! There is an Oblivious RAM Lower Bound**, CRYPTO 2018

Weiss and Wichs: **Is there an Oblivious RAM Lower Bound for Online Reads?** TCC 2018

Pavel Hubacek, Michal Koucky, Karel Kral, Veronika Slivova: **Strong Lower Bounds for Online ORAM**, TCC 2019

Jacob, Larsen, Nielsen: **Lower bounds for oblivious data structures**, SODA 2019

Persiano and Yeo: **Lower bounds for differentially private RAMs,** EUROCRYPT 2019

Larsen, Simkin, Yeo: **Lower bounds for multi-server oblivious RAMs,** TCC 2020

Komargodski and Lin: **A logarithmic lower bound for oblivious RAM (for all parameters),** CRYPTO 2021

And more...

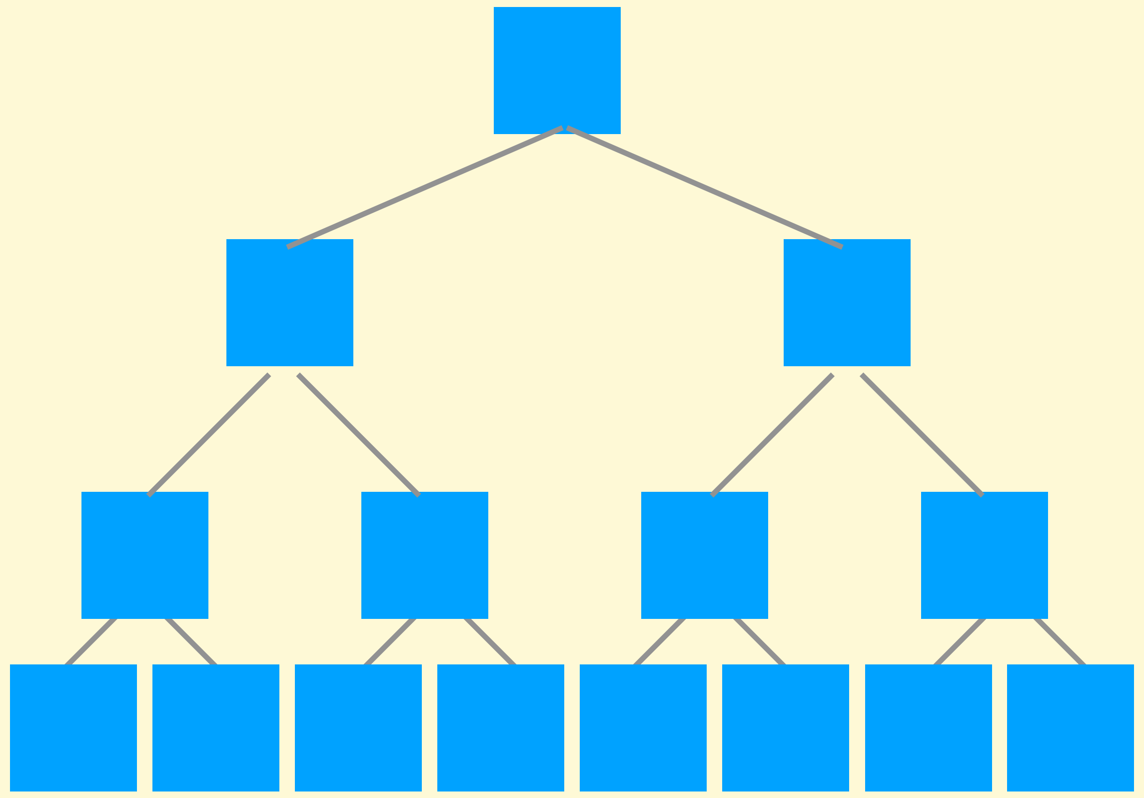
Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$

[GoldreichOstrovsky'96, LarsenNeilsen'18]

Hierarchical

[O90,GO96]



Tree based ORAM

[Shi,Chan,Stefanov11]

$O(\log N)$

Computational security

[OptORAMa'20]

$O(\log^2 N)$

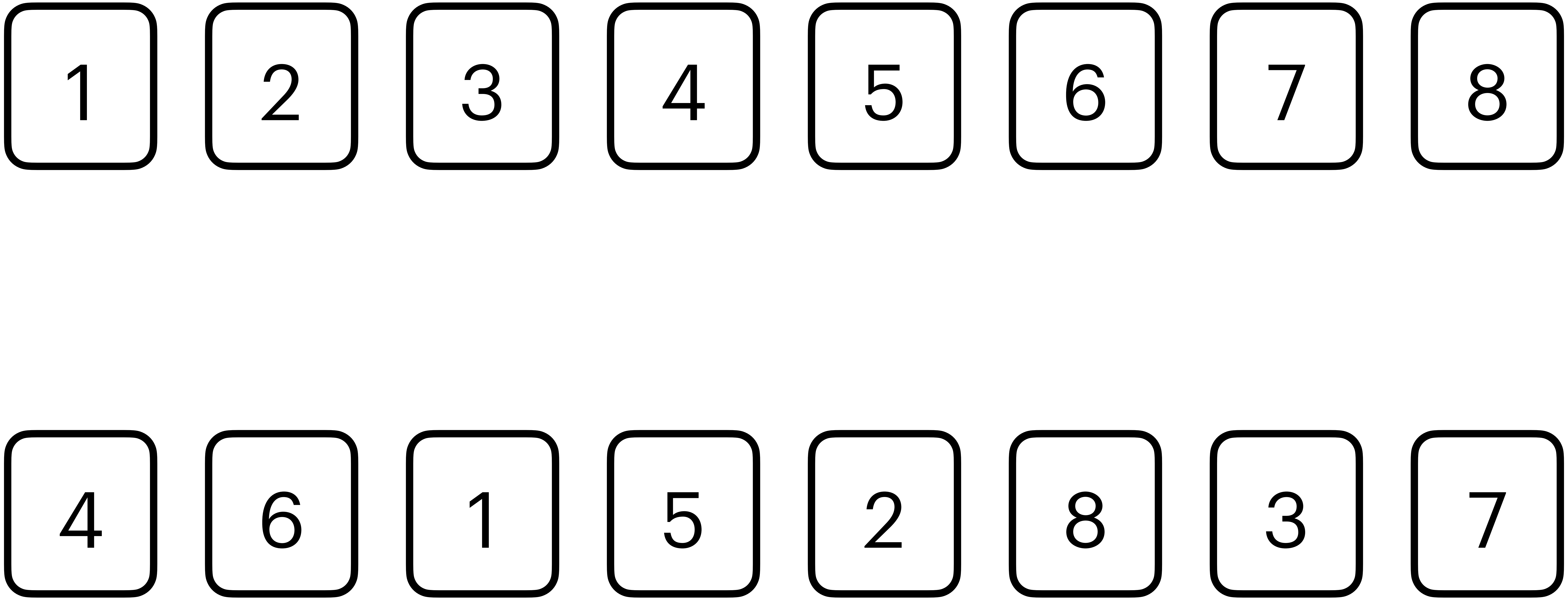
Statistical security

[PathORAM,CircuitORAM]

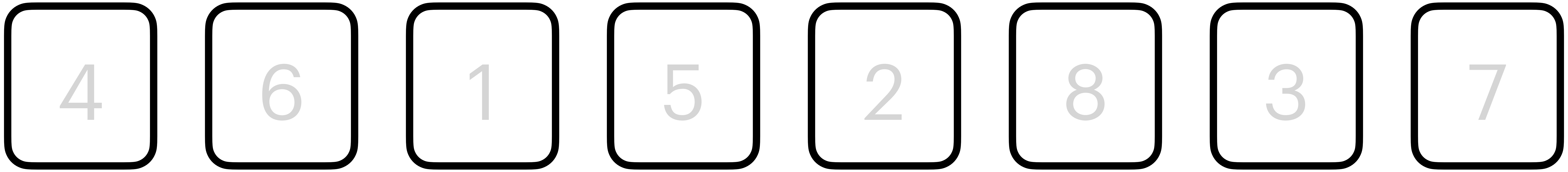
Tree Based ORAM

Simple constructions, statistical security, $O(\log^2 N)$ overhead

Strawman: Randomly Permute Blocks in Memory

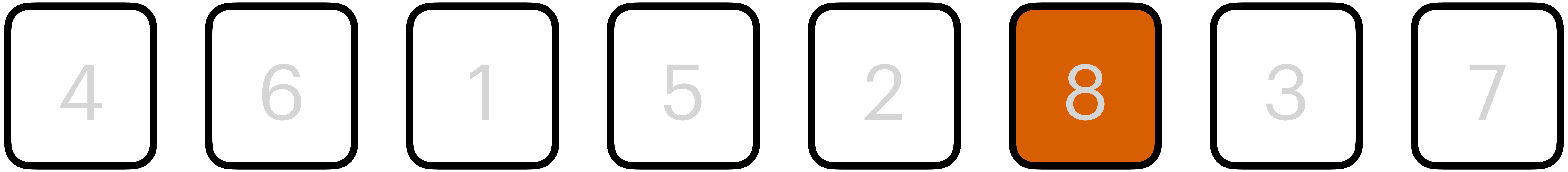


Strawman: Randomly Permute Blocks in Memory



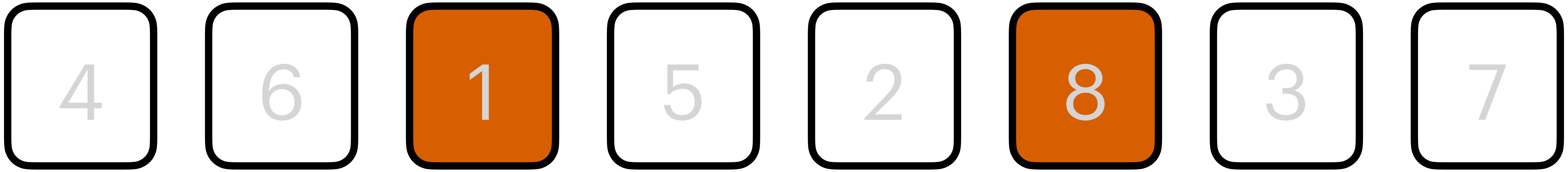
Strawman: Randomly Permute Blocks in Memory

The adversary has no clue what the client is accessing



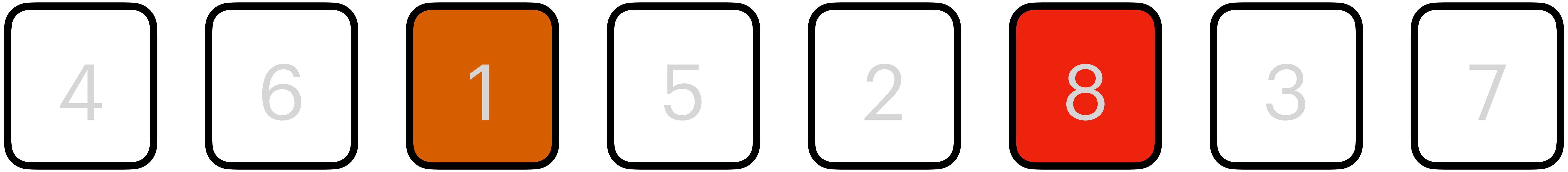
Strawman: Randomly Permute Blocks in Memory

The adversary has no clue what the client is accessing



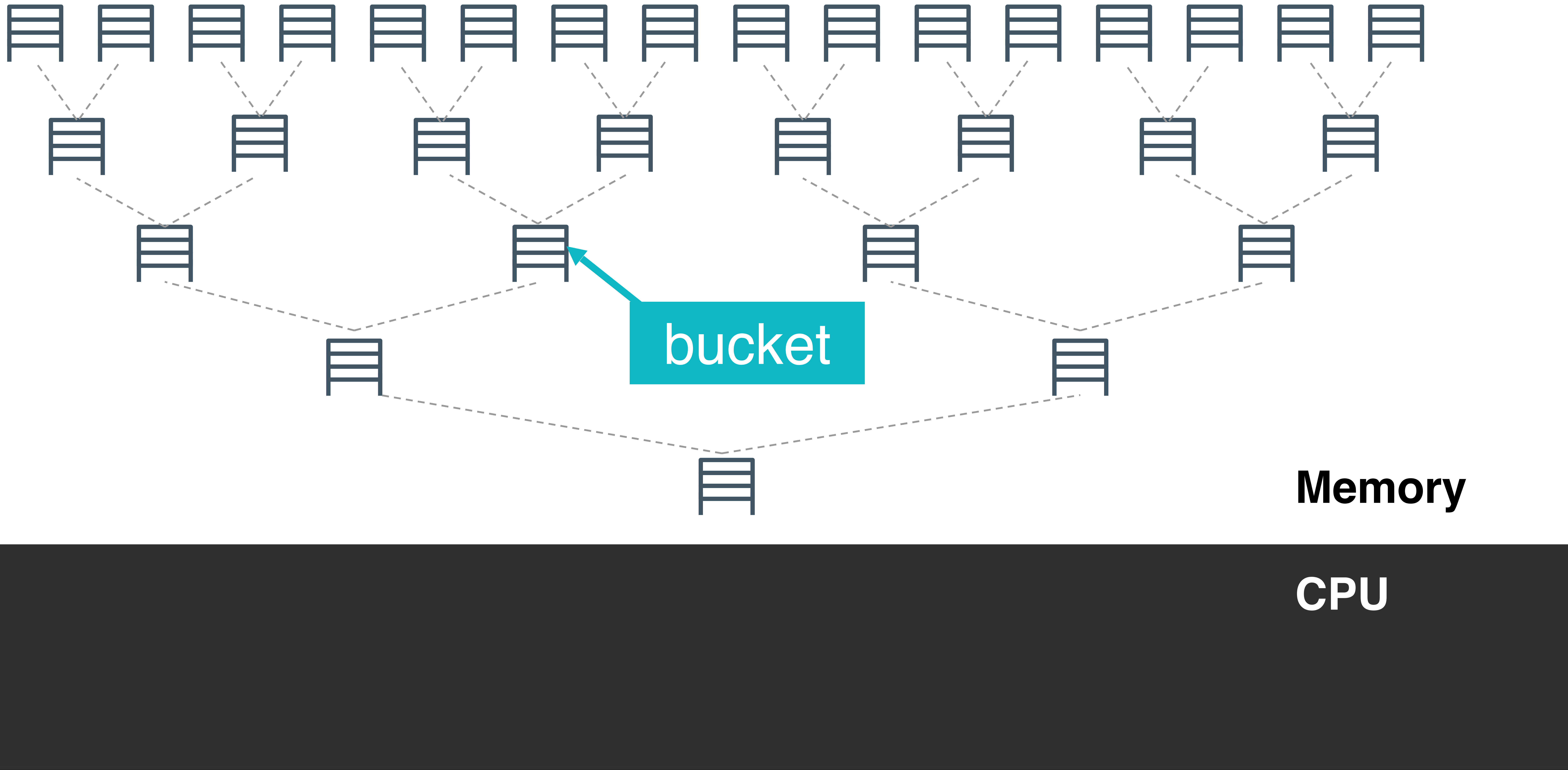
Strawman: Randomly Permute Blocks in Memory

Repeated query!!!



Blocks must move around in memory!

Each bucket stores real and dummy blocks

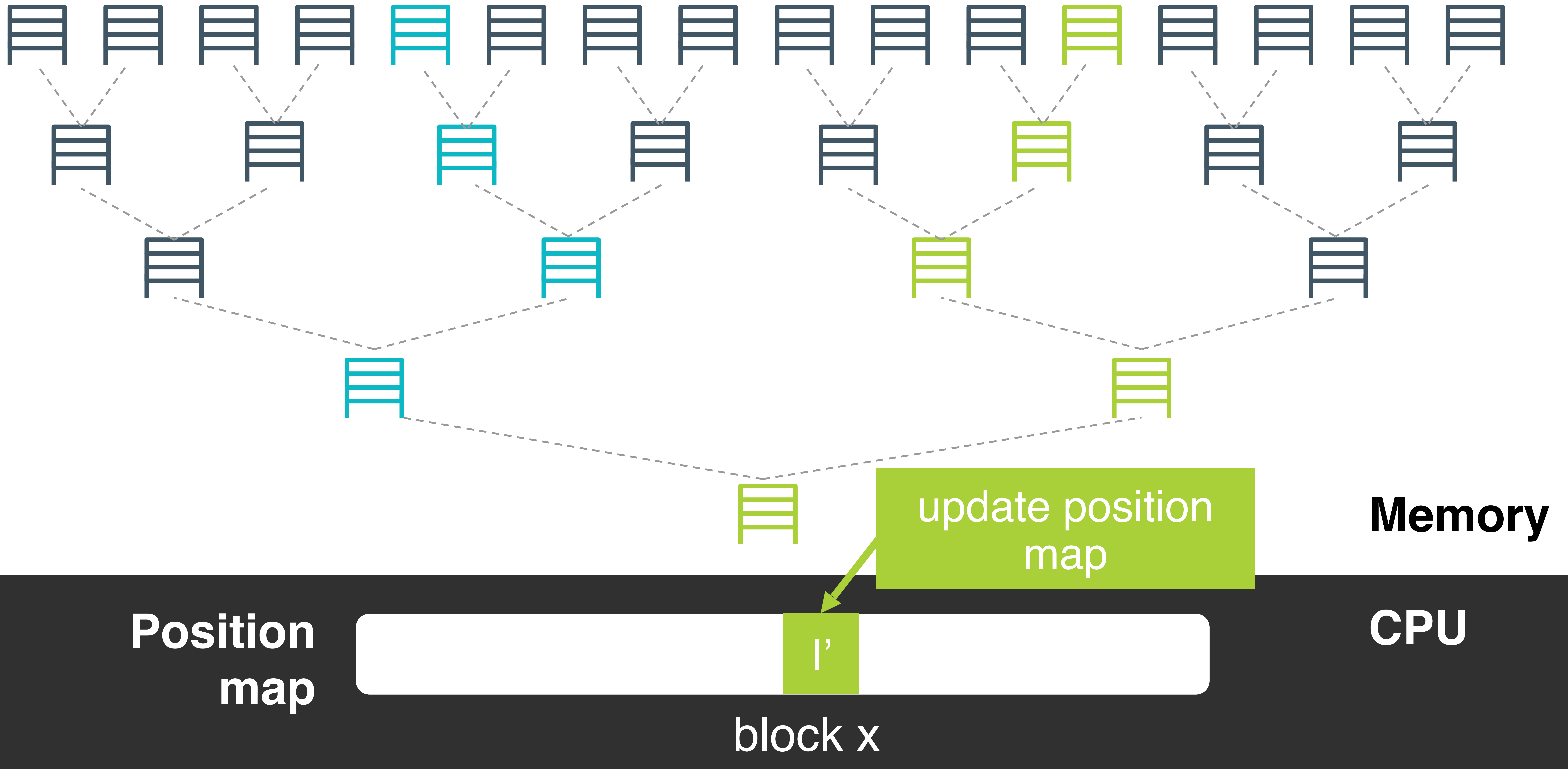


Path invariant: every block mapped to a random path

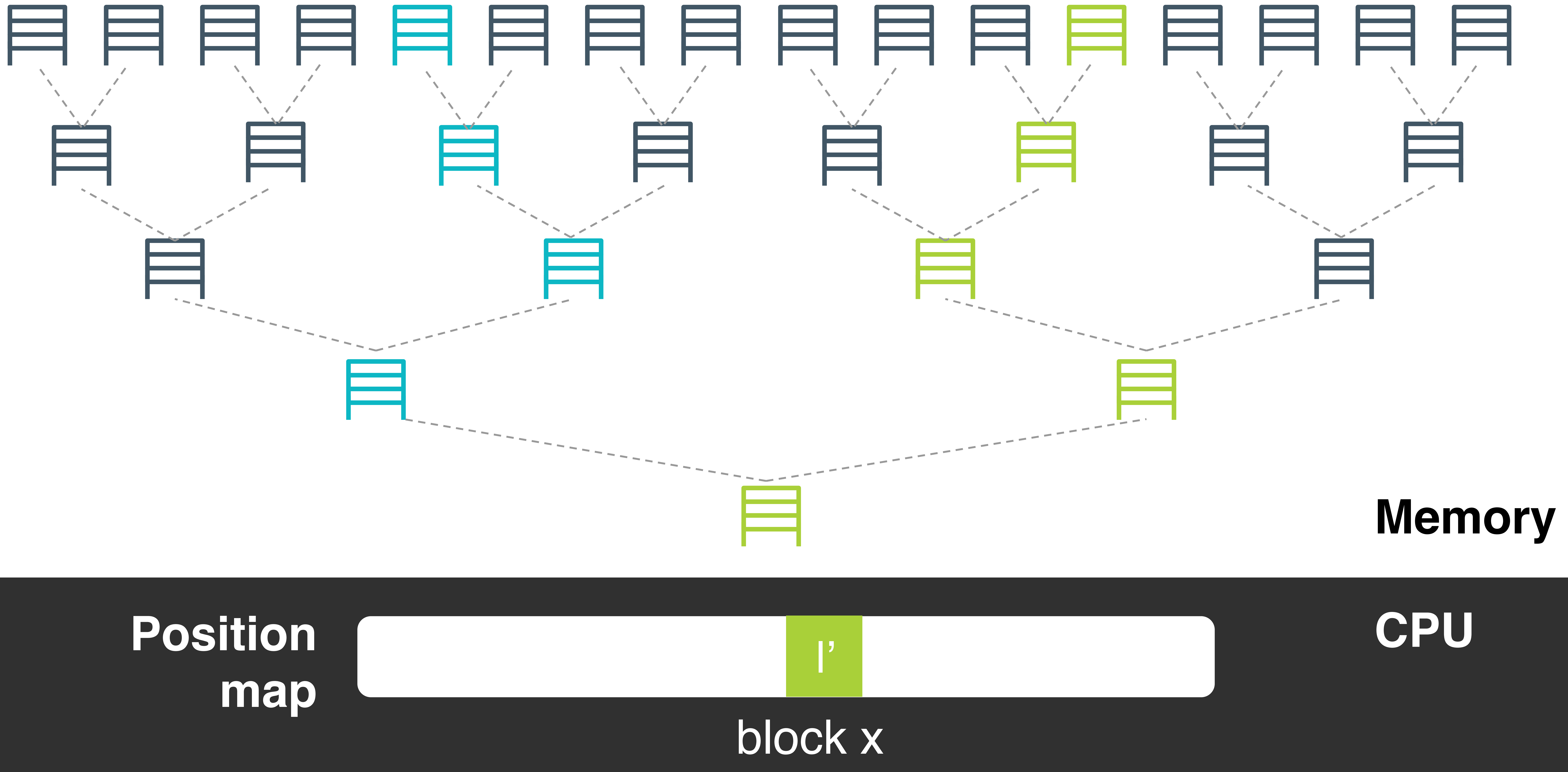
Reading a block is simple!

After being read, block x must relocate!

Pick a new random path and move x there



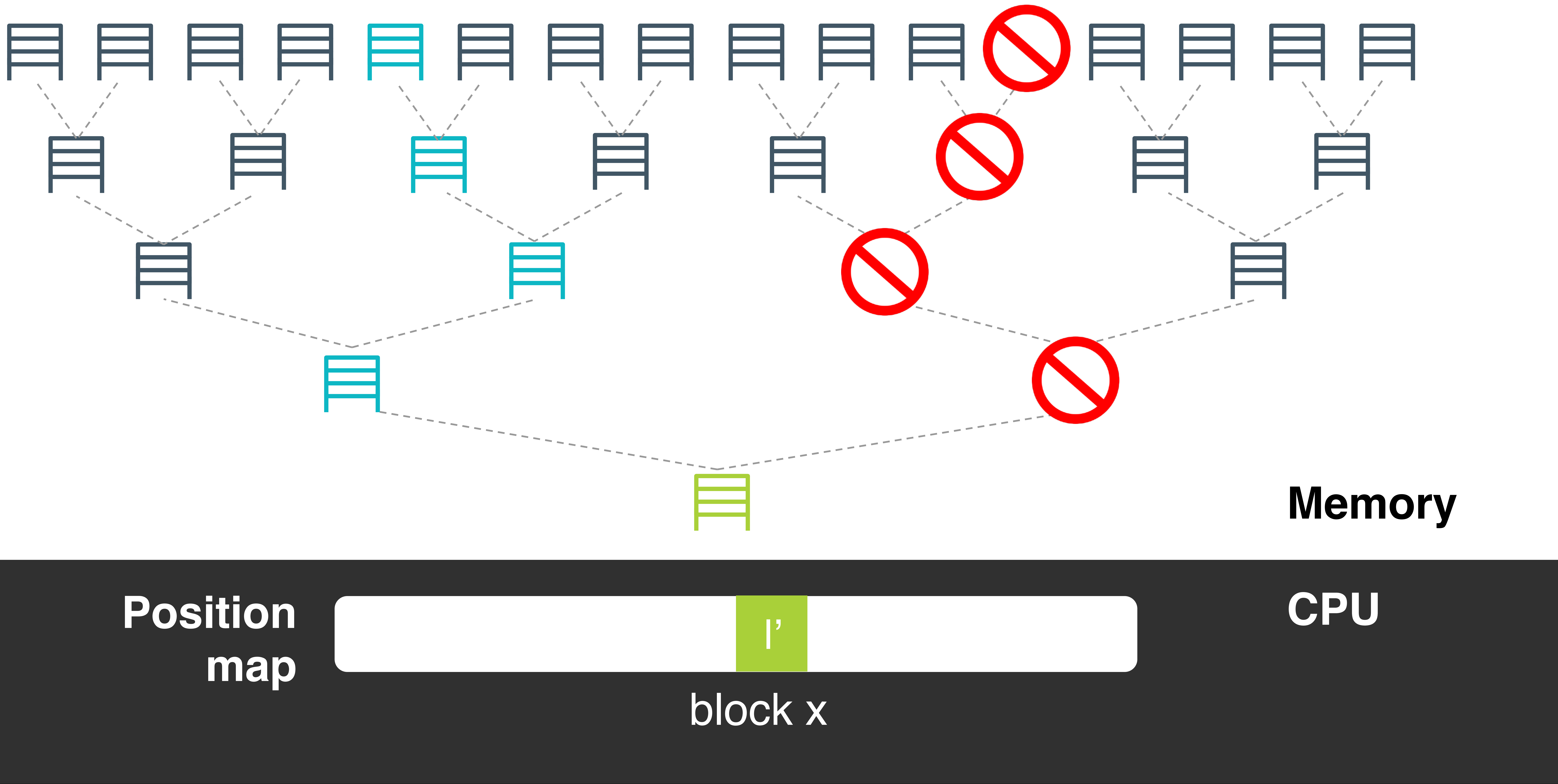
Where on the new path can we write **block x** ?



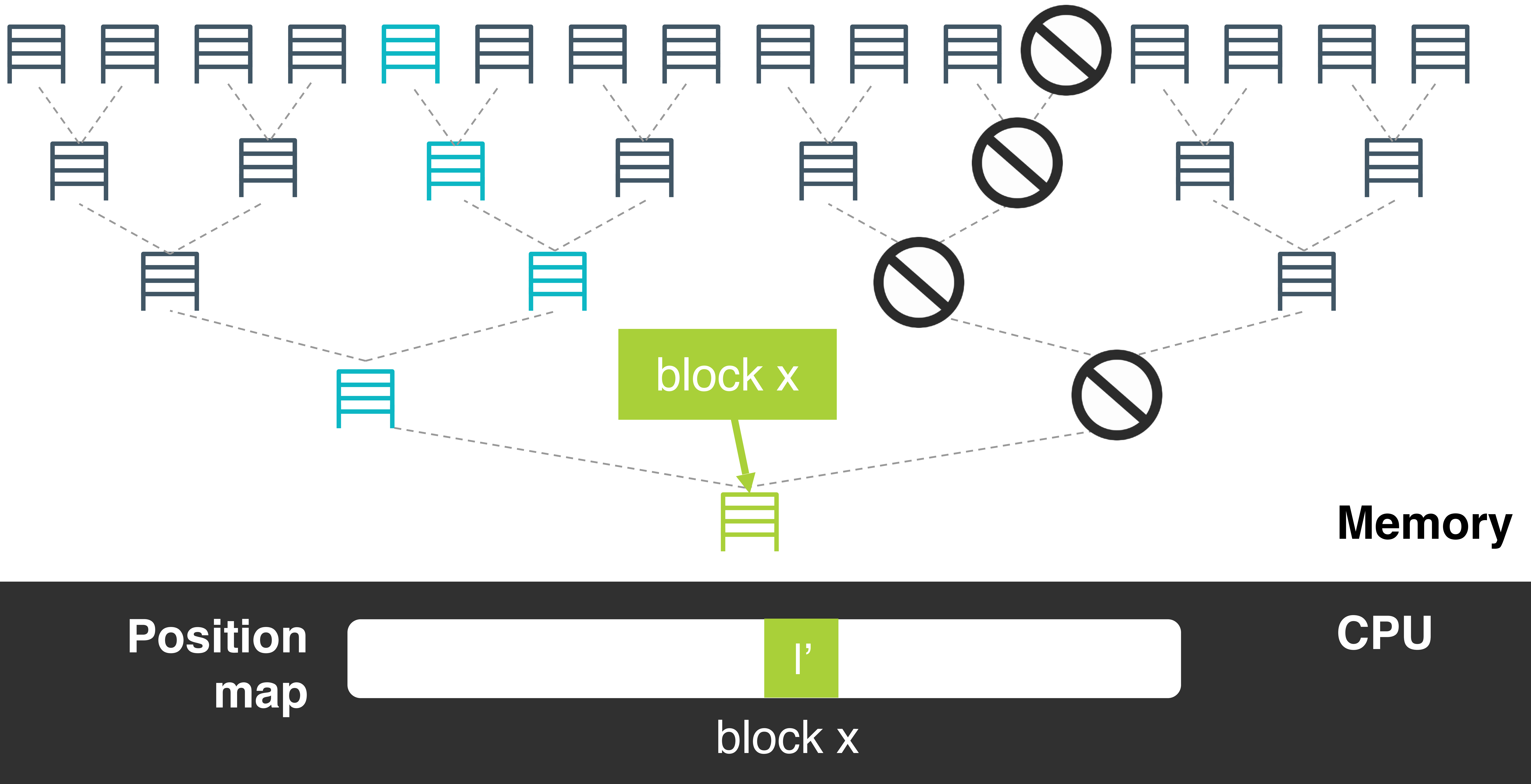

Can we write it to the leaf?
Memory
Position map
l'
block x
CPU

Can we write it to the leaf?

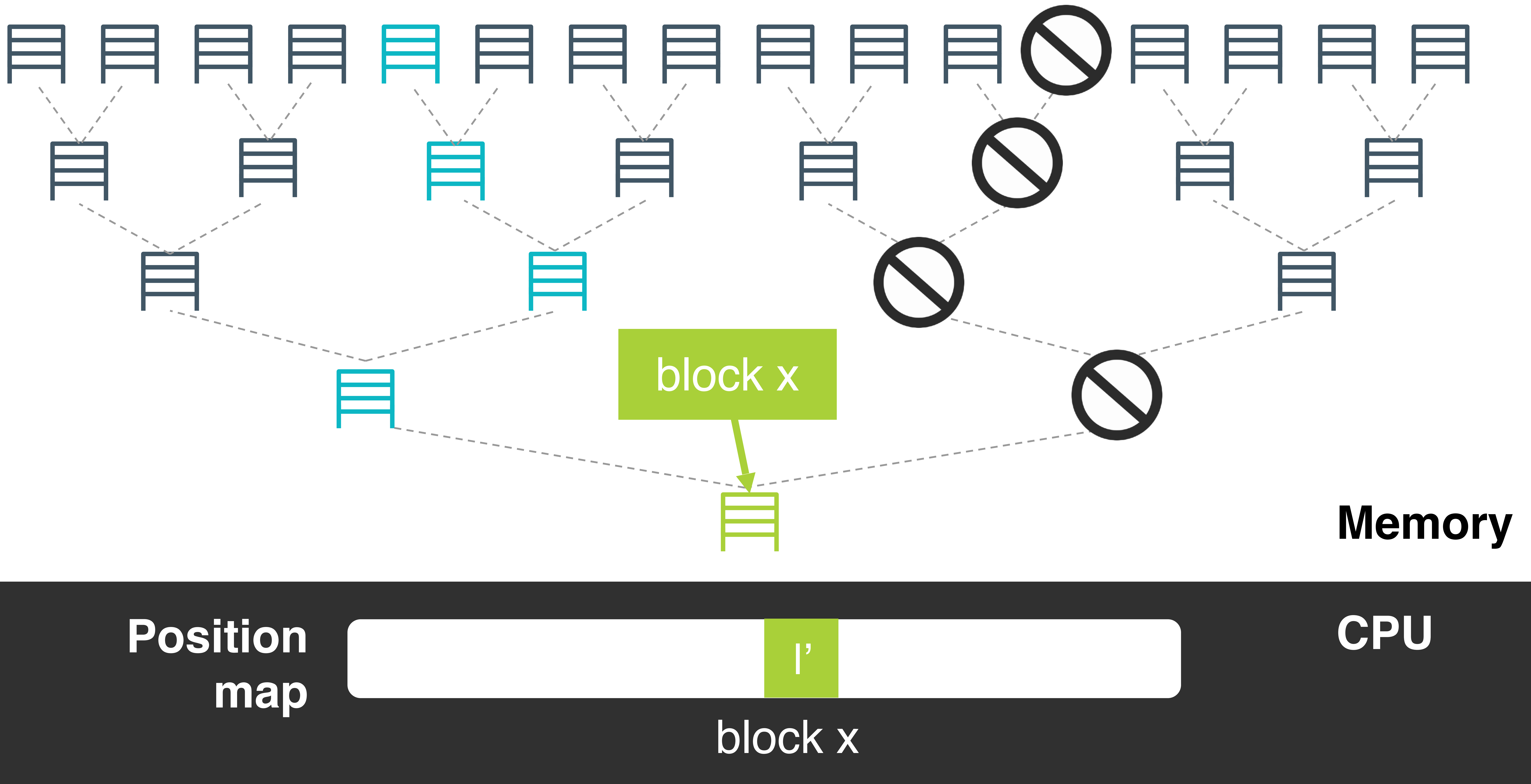
Writing to any non-root bucket leaks information



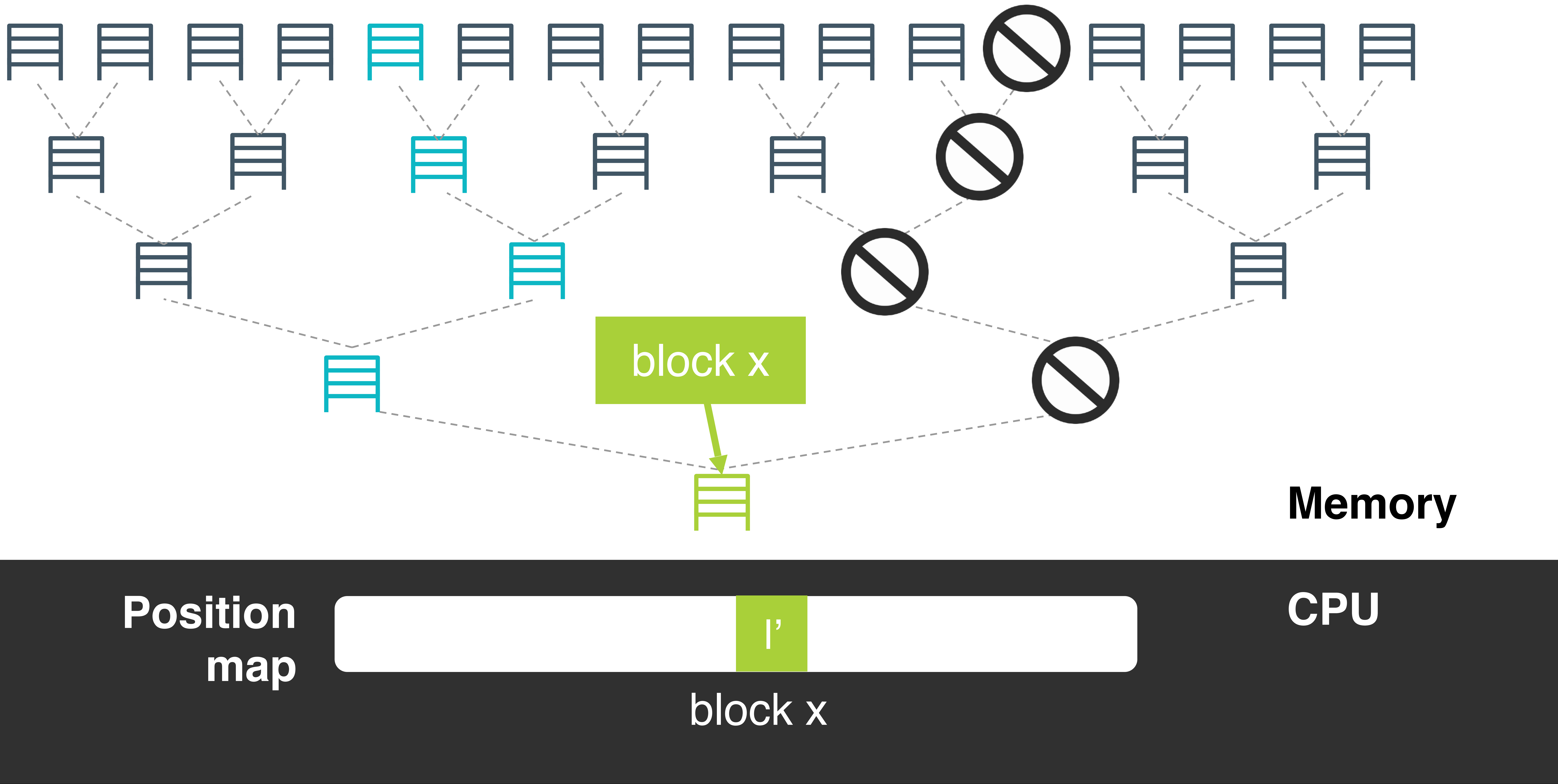
Write it to the root!



Security: every request, visit a **random** path that has **not** been revealed



Problem?

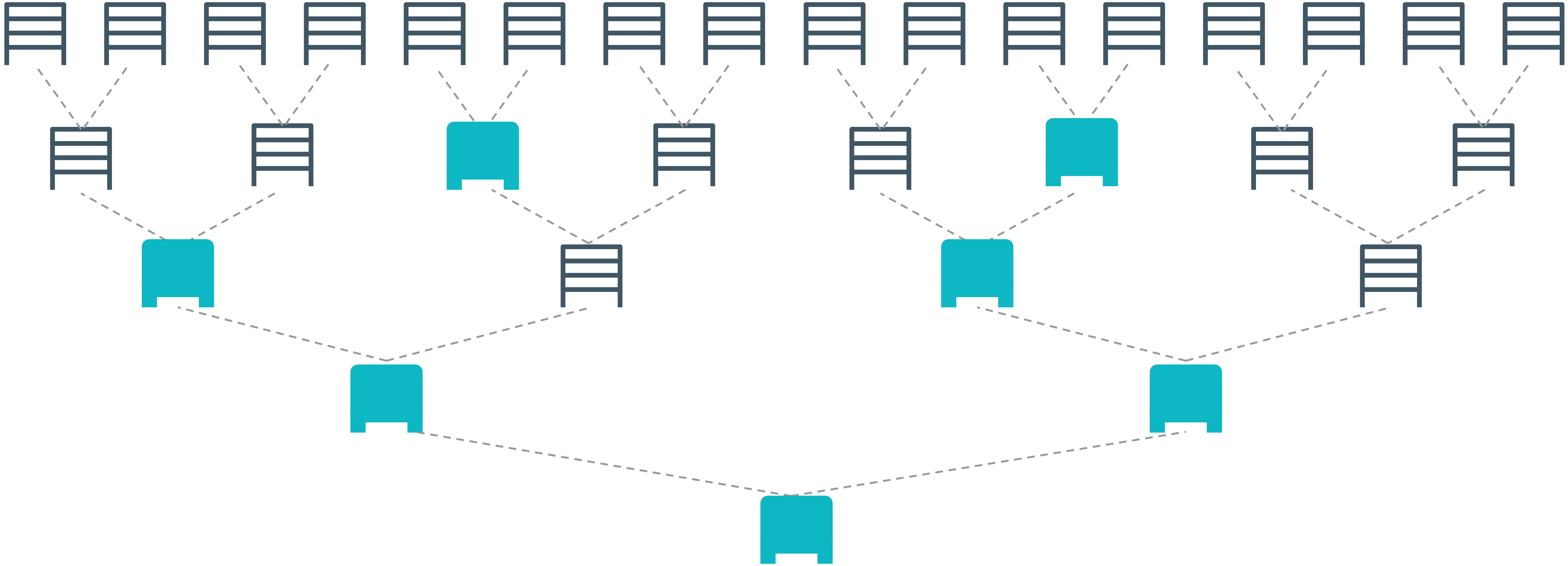


Problem: root will overflow

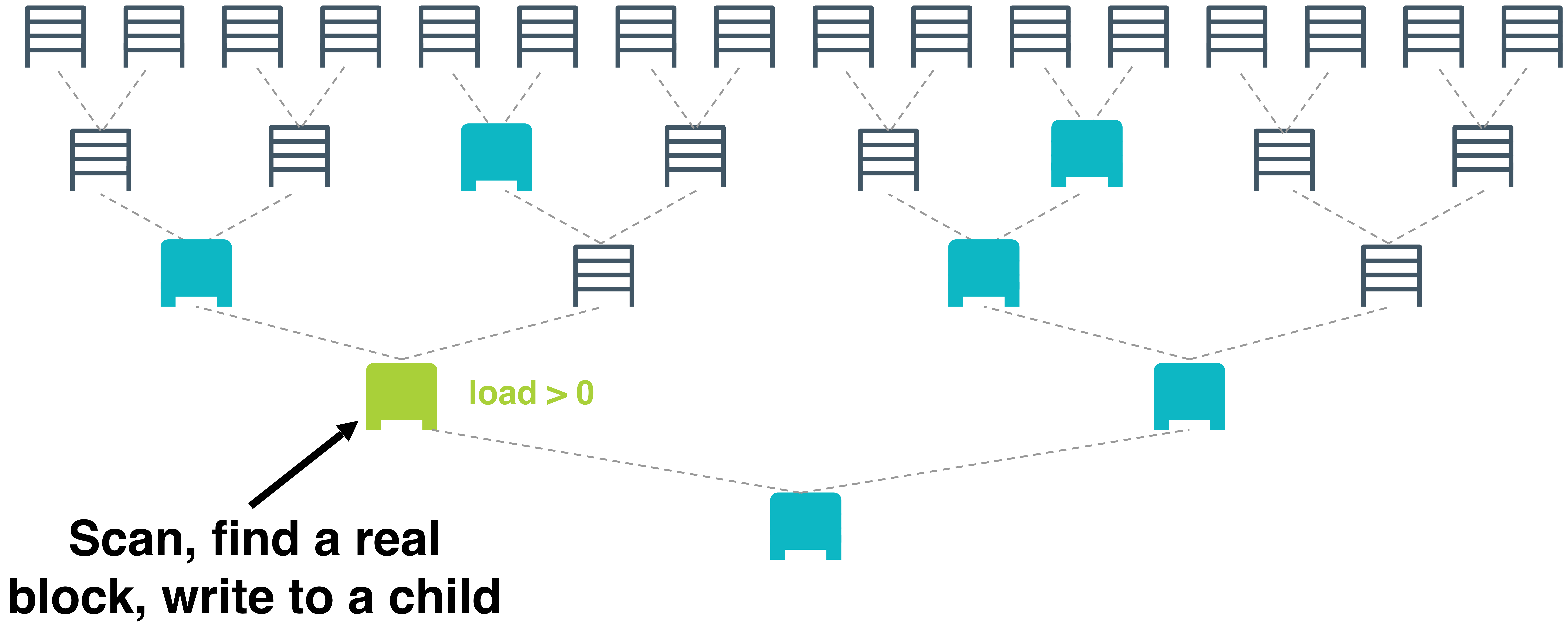
A background eviction process percolates blocks upwards

A background eviction process percolates blocks upwards

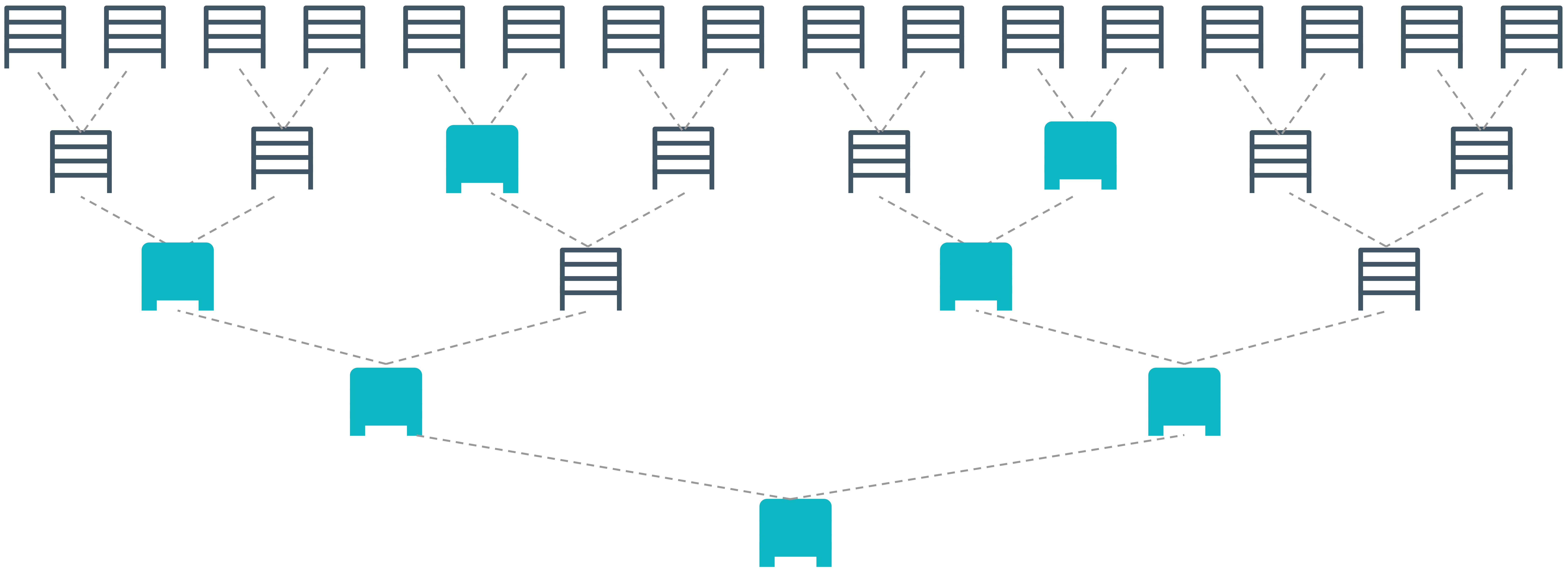
Every request: pick 2 random buckets per level to evict



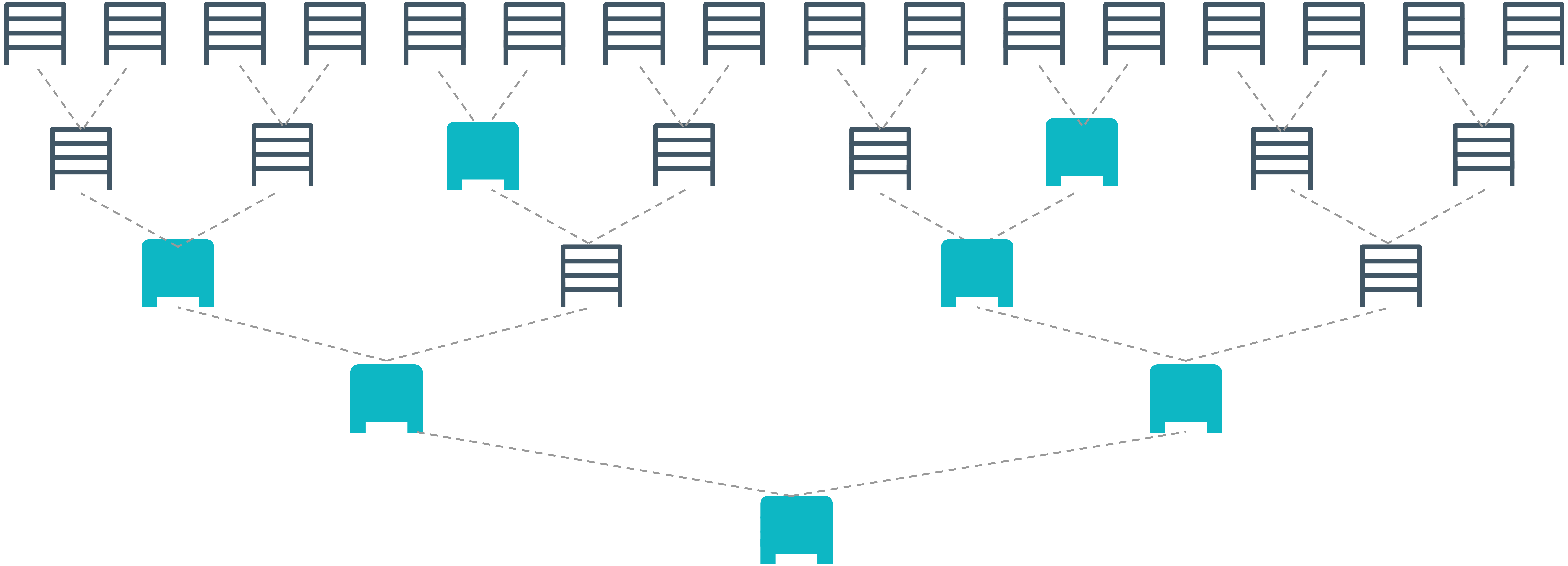
Every request: pick 2 random buckets per level to evict



Eviction process does not leak information

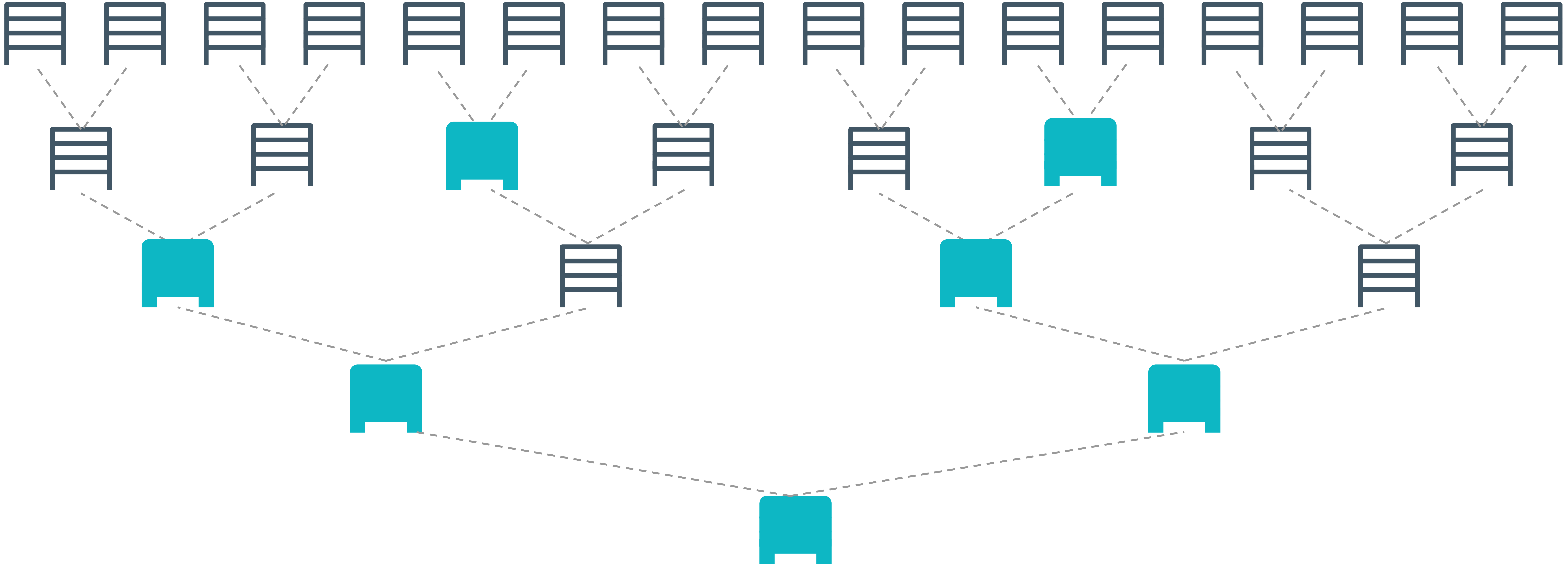


Thm: bucket size = log n ⇨ no overflow w.h.p. [SCSL'11]

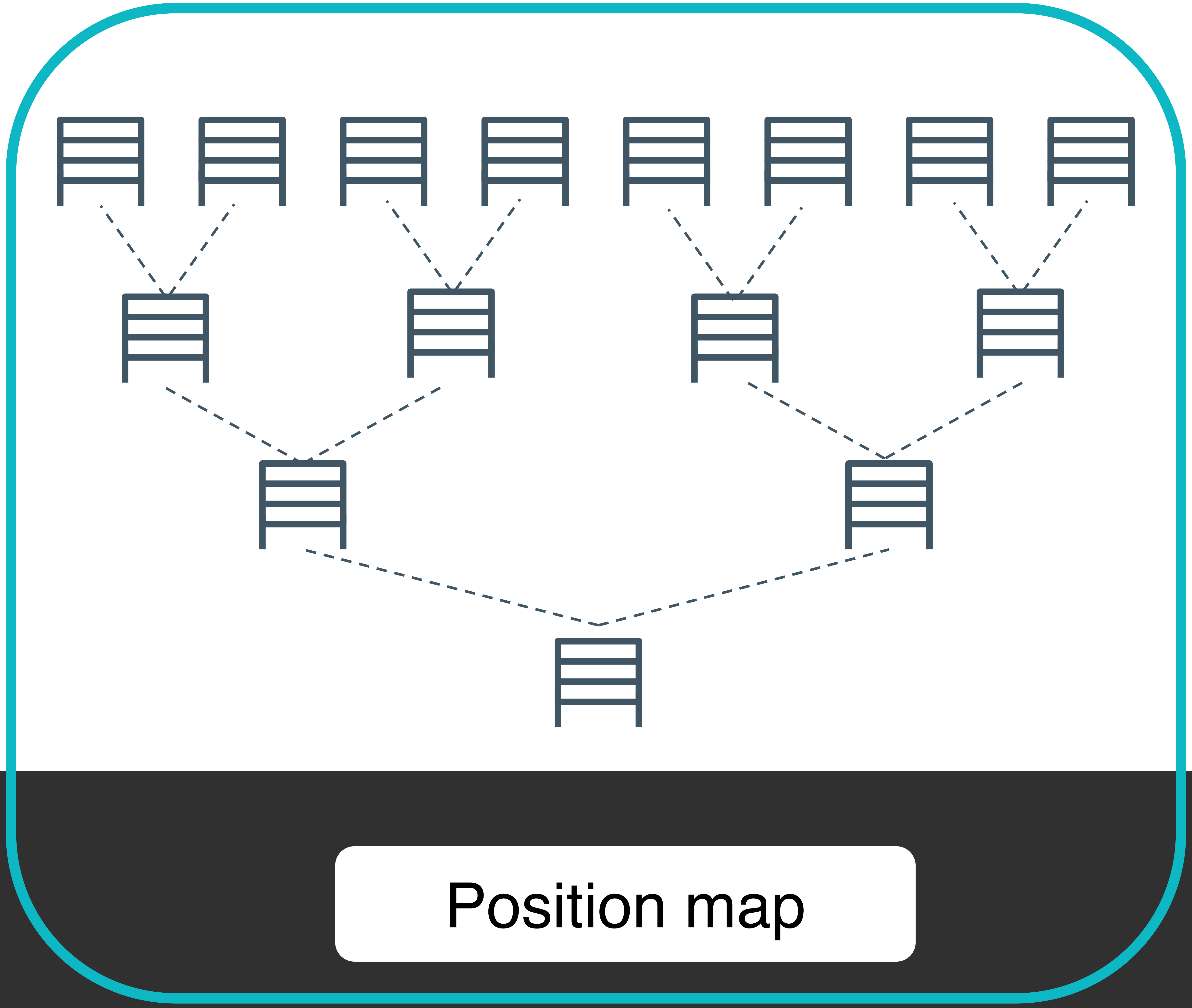


Proof: use queuing theory and measure concentration bounds.

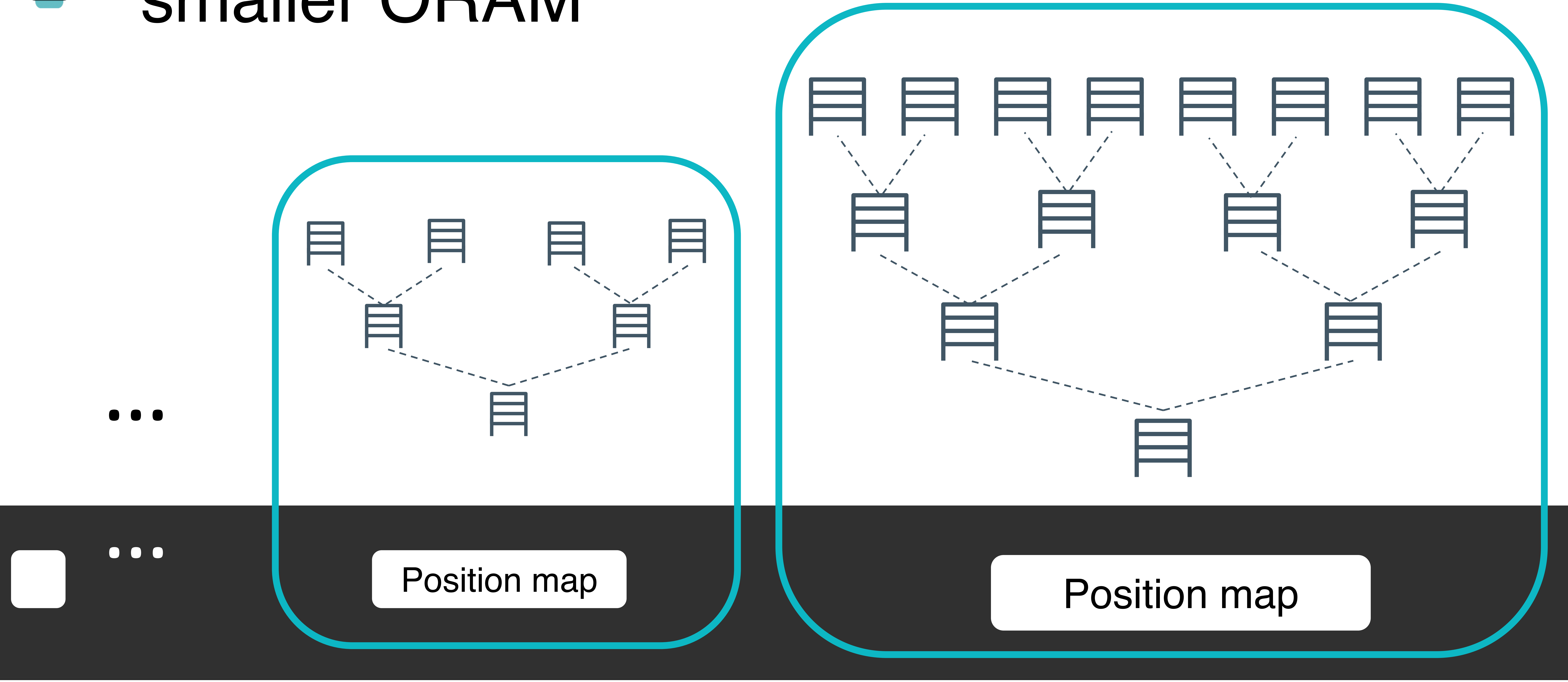
Thm: bucket size = log n ⇨ no overflow w.h.p. [SCSL'11]



Every request incurs $O(\log^2 n)$ cost


Position map

Store position map **recursively** in a smaller ORAM



Cost with eviction: $O(\log^3 n)$

Previous construction - $O(\log^3 N)$ overhead:

- Each path has $O(\log N)$ nodes
- Each node has a bucket of size $O(\log N)$
- Recursion adds another $O(\log N)$

Improvement: Path ORAM ($O(\log^2 N)$ overhead)

- Each node has a bucket of size $O(1)$
- Client has local stash of size poly $\log N$

Path ORAM

ACM CCS '13

[SDS+'13]

$x \leftarrow \mathsf{position}[\mathsf{a}]$
$\mathsf{position}[\mathsf{a}] \leftarrow \mathsf{UniformRandom}(0 \ldots 2^L - 1)$
for $\ell \in \{0, 1, \ldots, L\}$ **do**
    $S \leftarrow S \cup \mathsf{ReadBucket}(\mathcal{P}(x, \ell))$
end for
$\mathsf{data} \leftarrow$ Read block a from S
if $\mathsf{op} = \mathsf{write}$ **then**
    $S \leftarrow (S - \{(\mathsf{a}, \mathsf{data})\}) \cup \{(\mathsf{a}, \mathsf{data}^*)\}$
end if
for $\ell \in \{L, L-1, \ldots, 0\}$ **do**
    $S' \leftarrow \{(\mathsf{a}', \mathsf{data}') \in S : \mathcal{P}(x, \ell) = \mathcal{P}(\mathsf{position}[\mathsf{a}'], \ell)\}$
    $S' \leftarrow$ Select $\min(|S'|, Z)$ blocks from S'.
    $S \leftarrow S - S'$
    $\mathsf{WriteBucket}(\mathcal{P}(x, \ell), S')$
end for
return data

Path ORAM

ACM CCS '13

[SDS+'13]

Achieves $O(\log^2 n)$ cost with recursion

Summary: tree-based ORAMs

- A block is re-mapped to a new random path upon being read.
- The block must be relocated to the new path without revealing the new path
- Key challenge: design eviction process and prove no overflow.

Tree Based ORAM

Shi, Chan, Stefanov, Li: **Oblivious RAM with $O(\log^3 N)$ Worst-Case Cost**, ASIACRYPT 2011

Stefanov, van Dijk, Shi, Fletcher, Ren, Yu, Devadas: **Path ORAM: an Extemrely Simple Oblivious RAM Protocol**, CCS 2013

Gentry, Goldman, Halevi, Jutla, Raykova, Wichs: **Optimizing ORAM and Using it Efficiently for Secure Computation**, PETS 2013

Chung, Pass: **A Simple ORAM**, 2013

Wang, Chan, Shi: Circuit ORAM: **On Tightness of the Goldreich-Ostrovsky Lower Bound**, CCS 2015

Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$

[GoldreichOstrovsky'96, LarsenNeilsen'18]

Tomorrow!

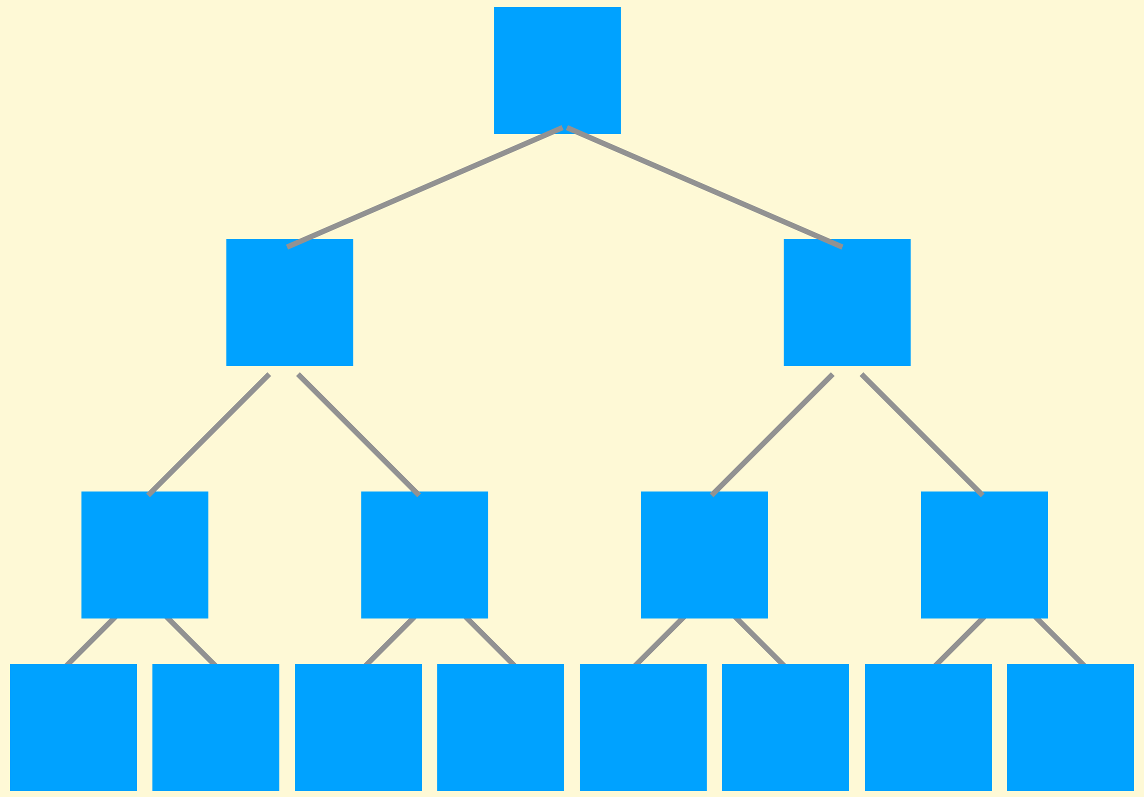
Hierarchical

[O90,GO96]

$O(\log N)$

Computational security

[OptORAMa,AKLNPS'20]



Tree based ORAM

[Shi,Chan,Stefanov11]

$O(\log^2 N)$

Statistical security

[PathORAM,CircuitORAM]

Thank You!