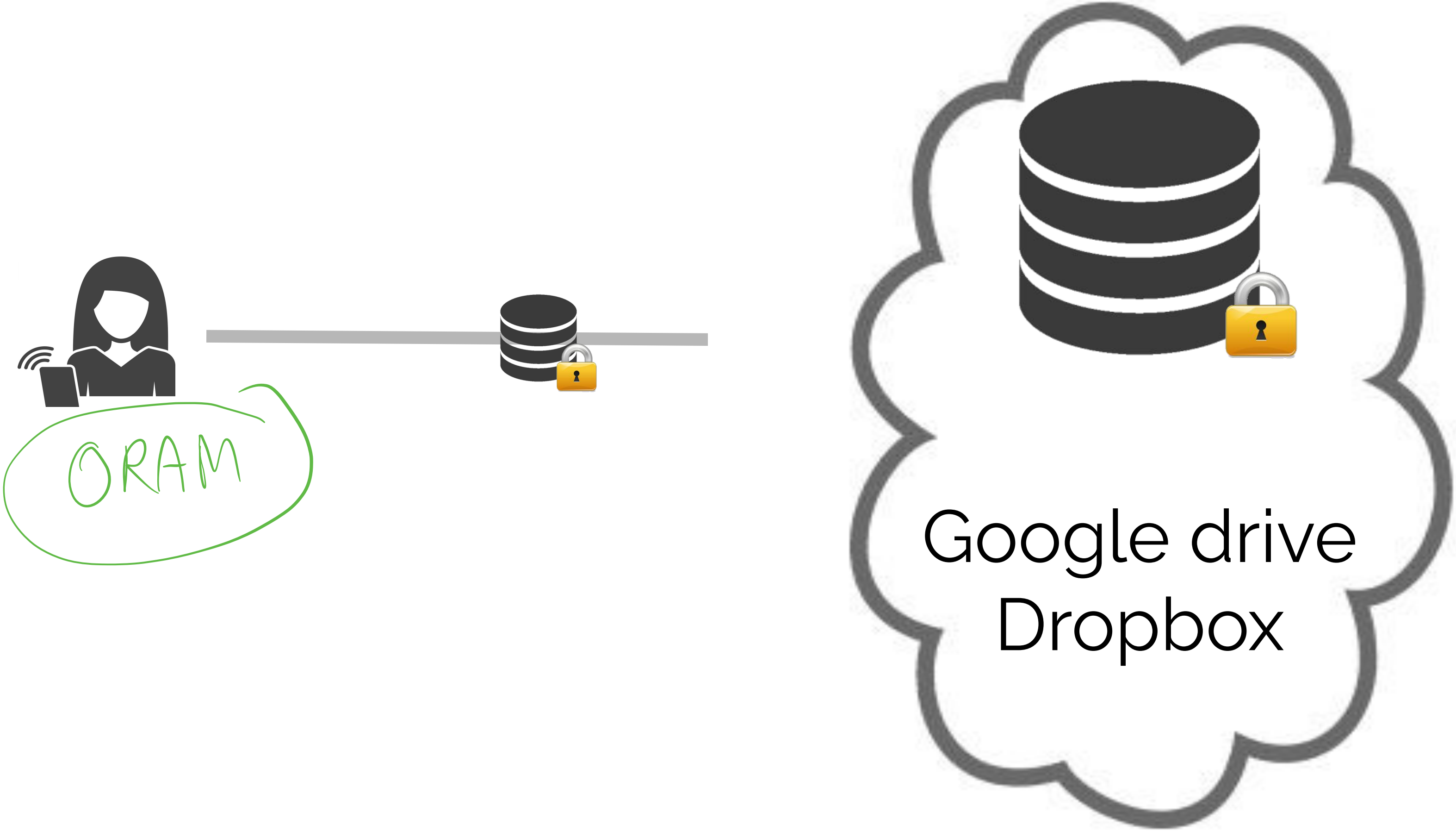
15451

Oblivious RAM

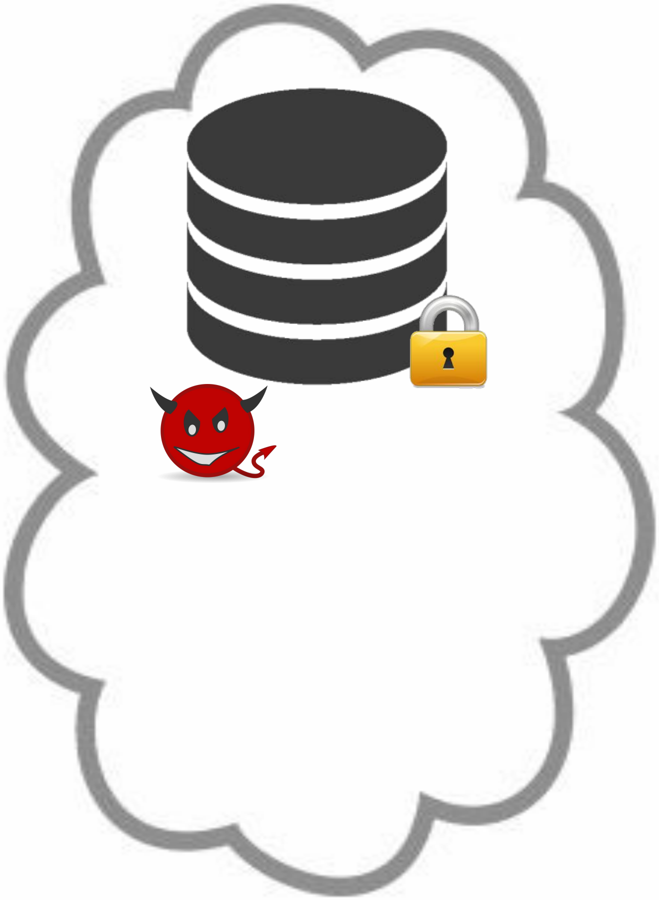
aka. Elaine's favorite data structure

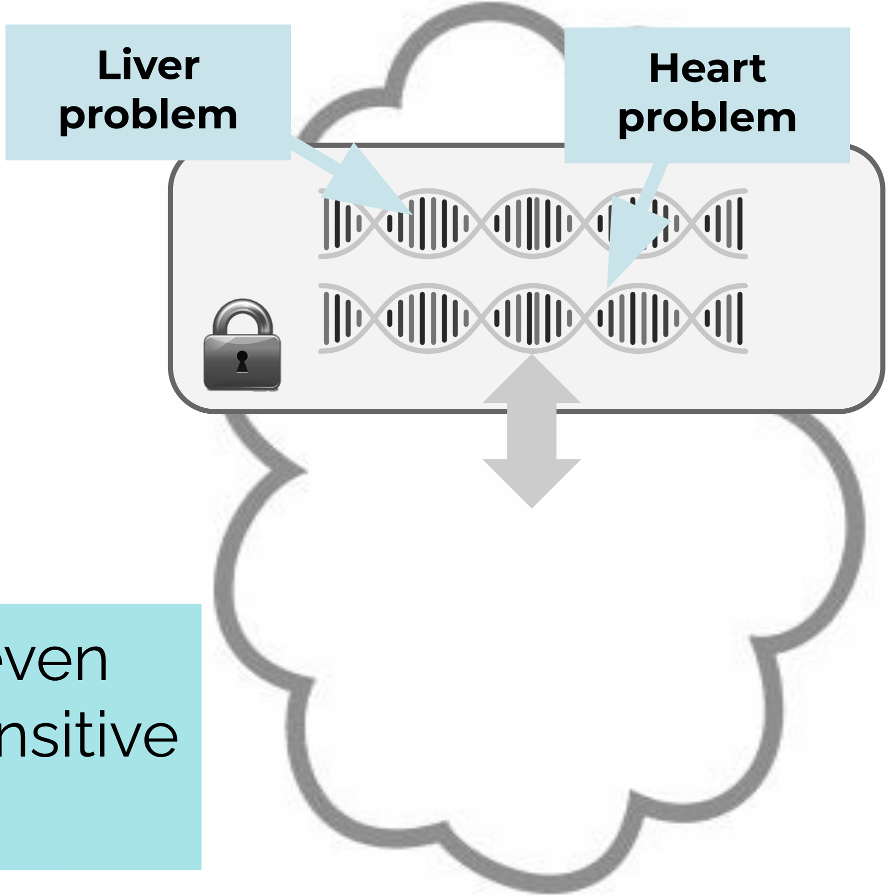
Elaine Shi

Motivating example:
Cloud outsourcing


ORAM
Google drive
Dropbox

Access patterns to even encrypted data leak sensitive information.



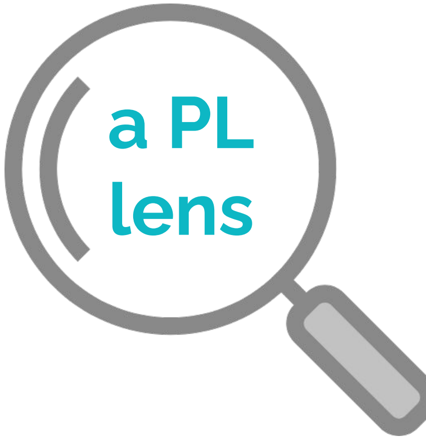


Access patterns to even encrypted data leak sensitive information.

Access patterns of binary search leaks the rank of the number being searched.

```
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 leakage through

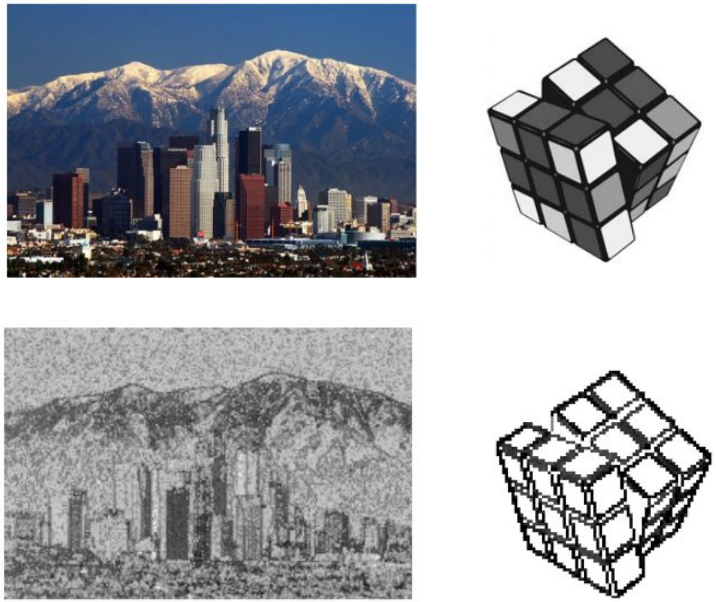



```
if (secret variable)
    read mem[x]
else
    read mem[y]
```

Recovering JPEG images through coarse-grained access patterns

Original

Recovered



[XCP'15 - IEEE S&P]

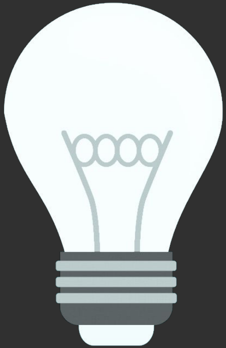
Can we **provably** defeat access pattern leakage

and **preserve efficiency**

Random Access Machine

Oblivious RAM (ORAM)

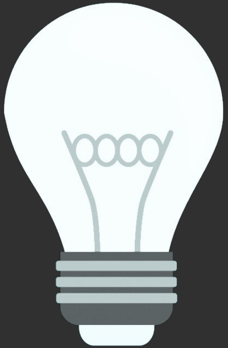
is an algorithmic technique that provably "encrypts" access patterns

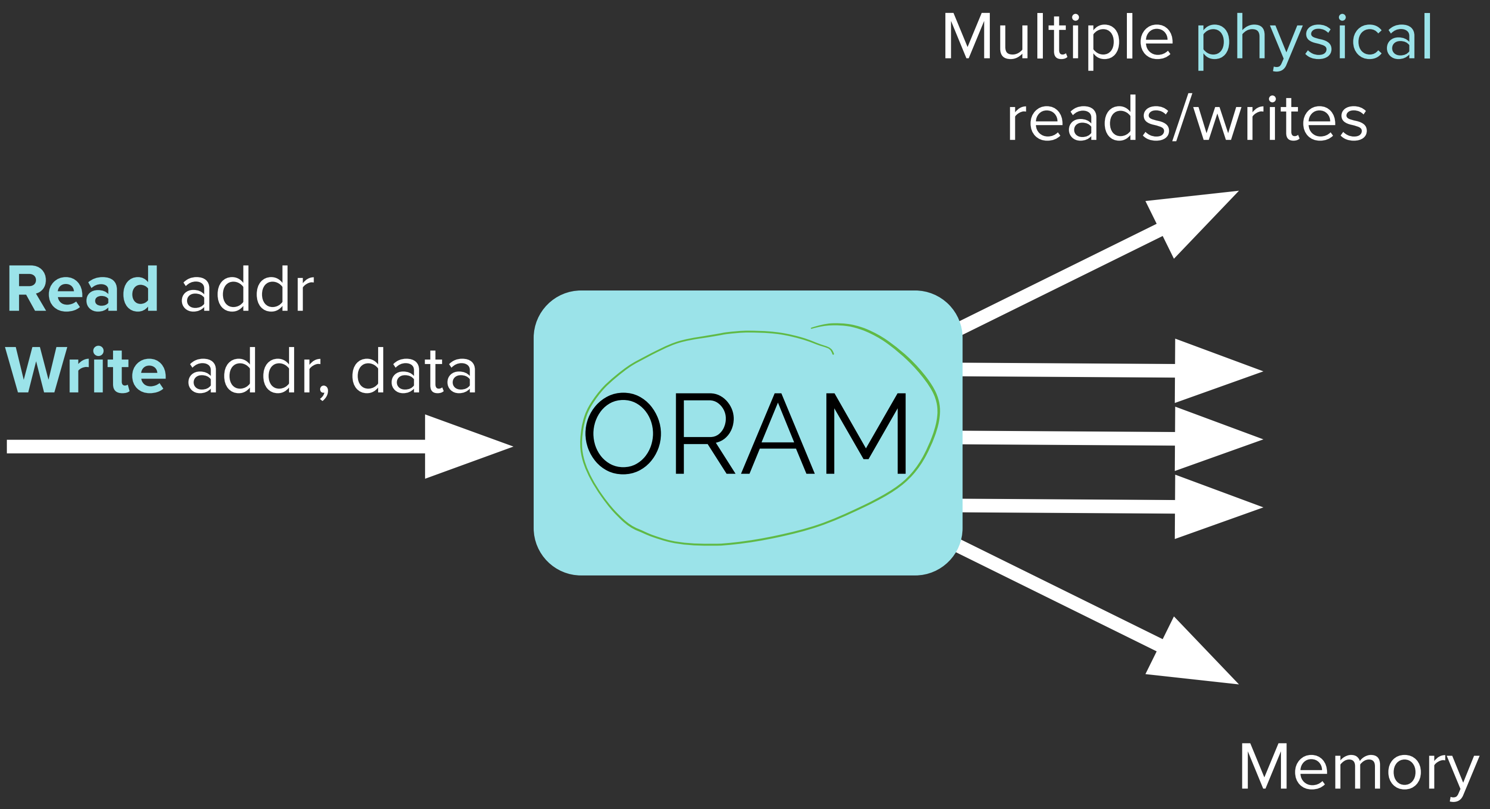


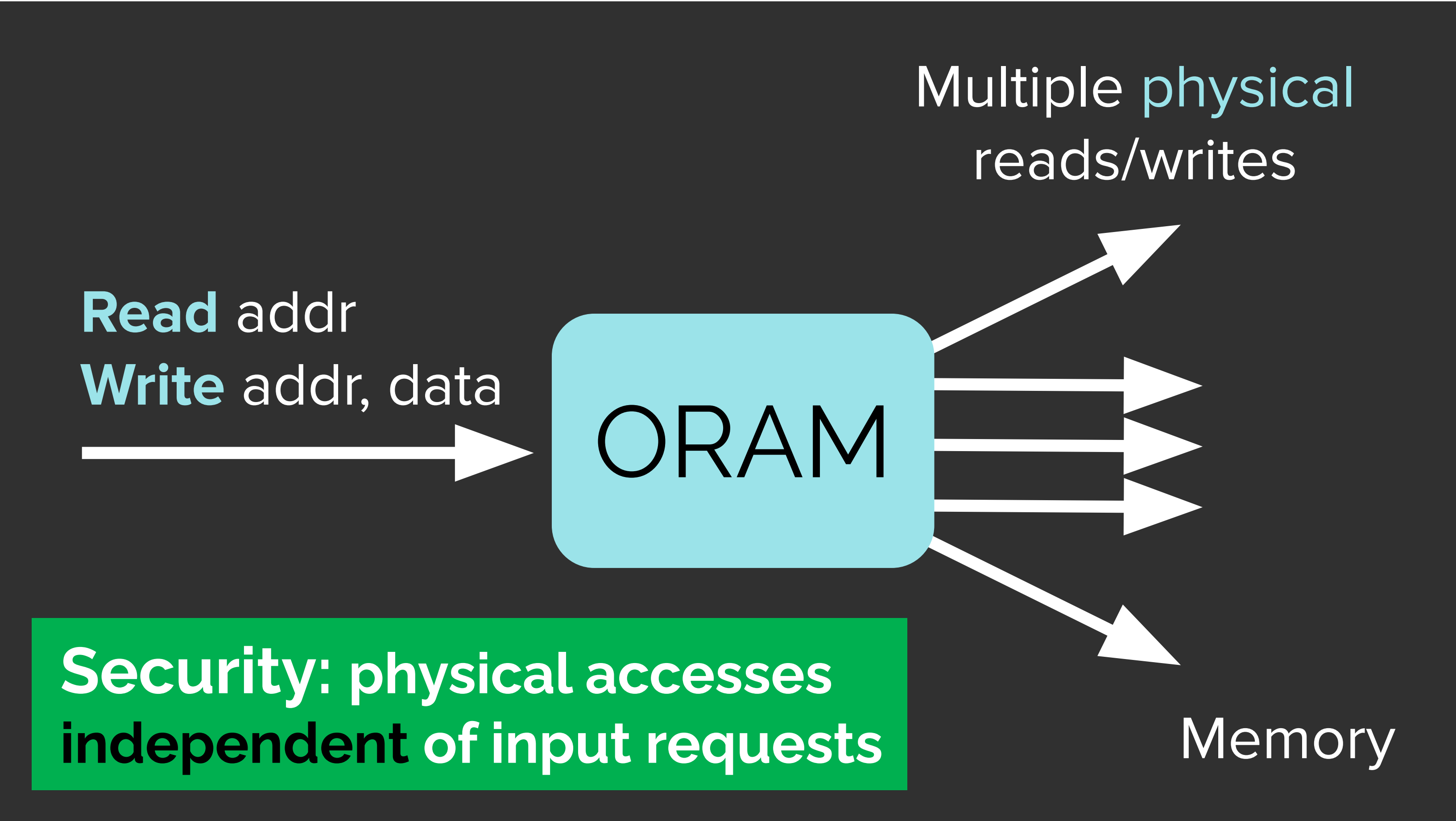
Oblivious RAM (ORAM)

is an algorithmic technique that provably "encrypts" access patterns

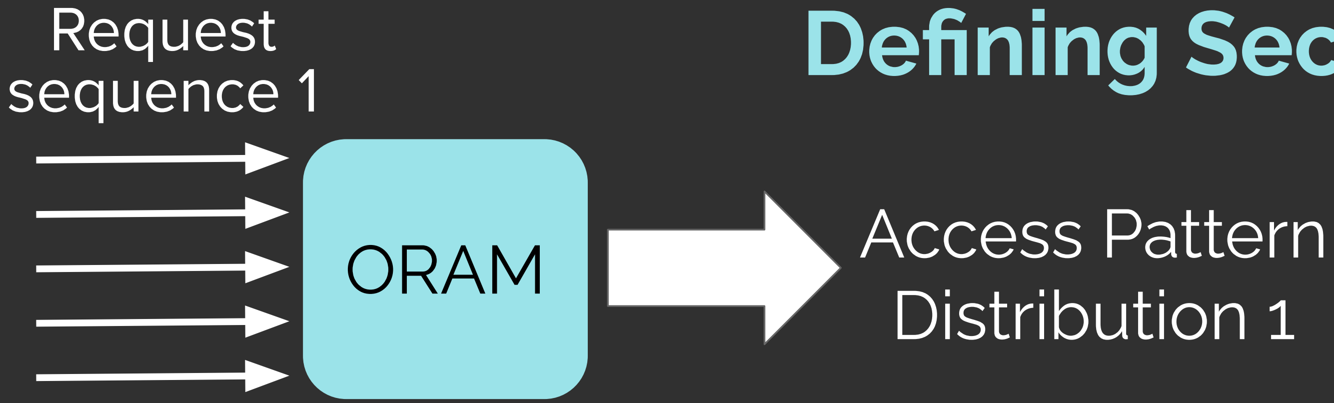
- Permutation
- Shuffling



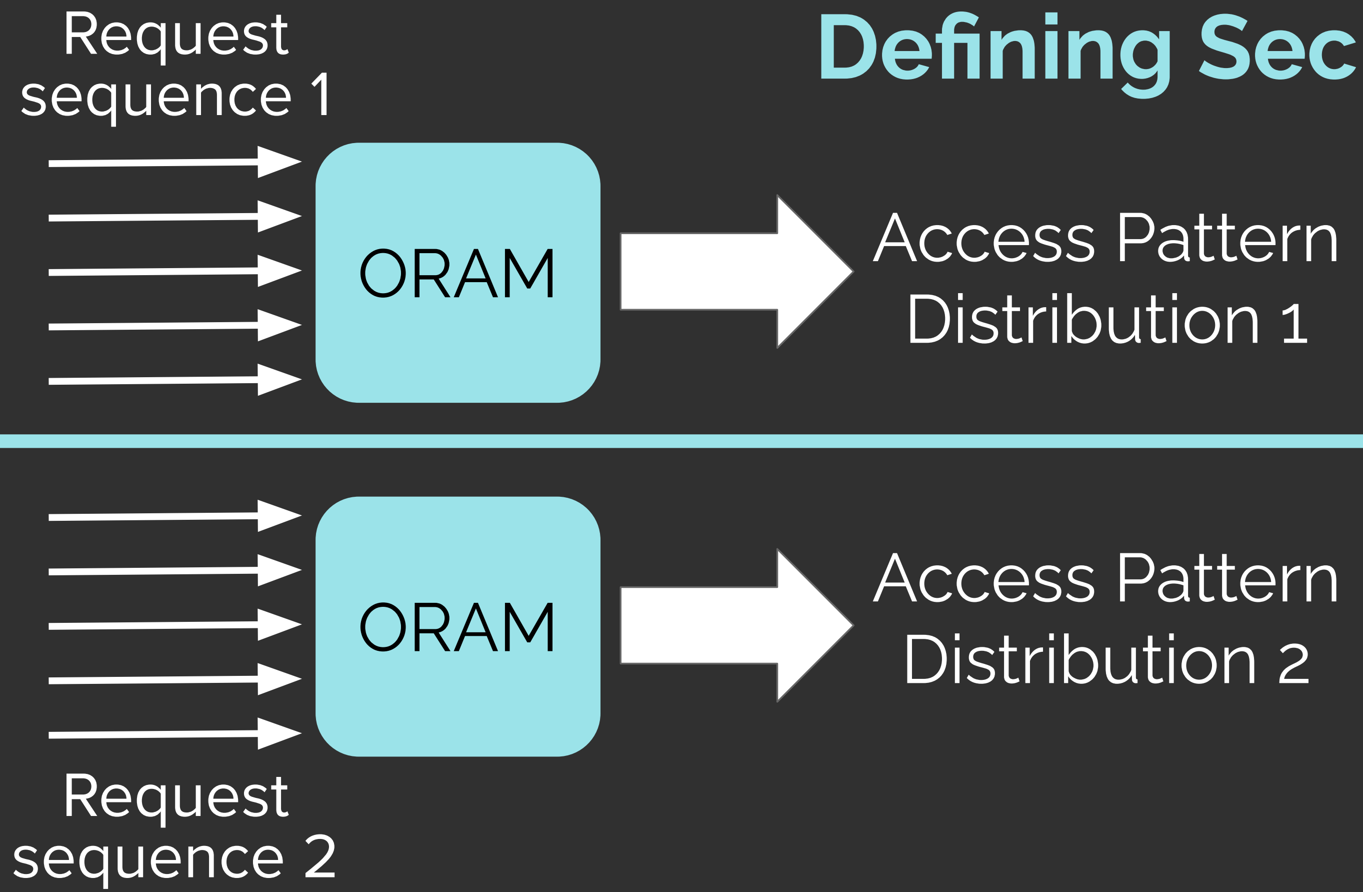

Multiple physical reads/writes
Read addr
Write addr, data
ORAM
Memory


Multiple physical reads/writes
Read addr
Write addr, data
ORAM
Memory
Security: physical accesses independent of input requests

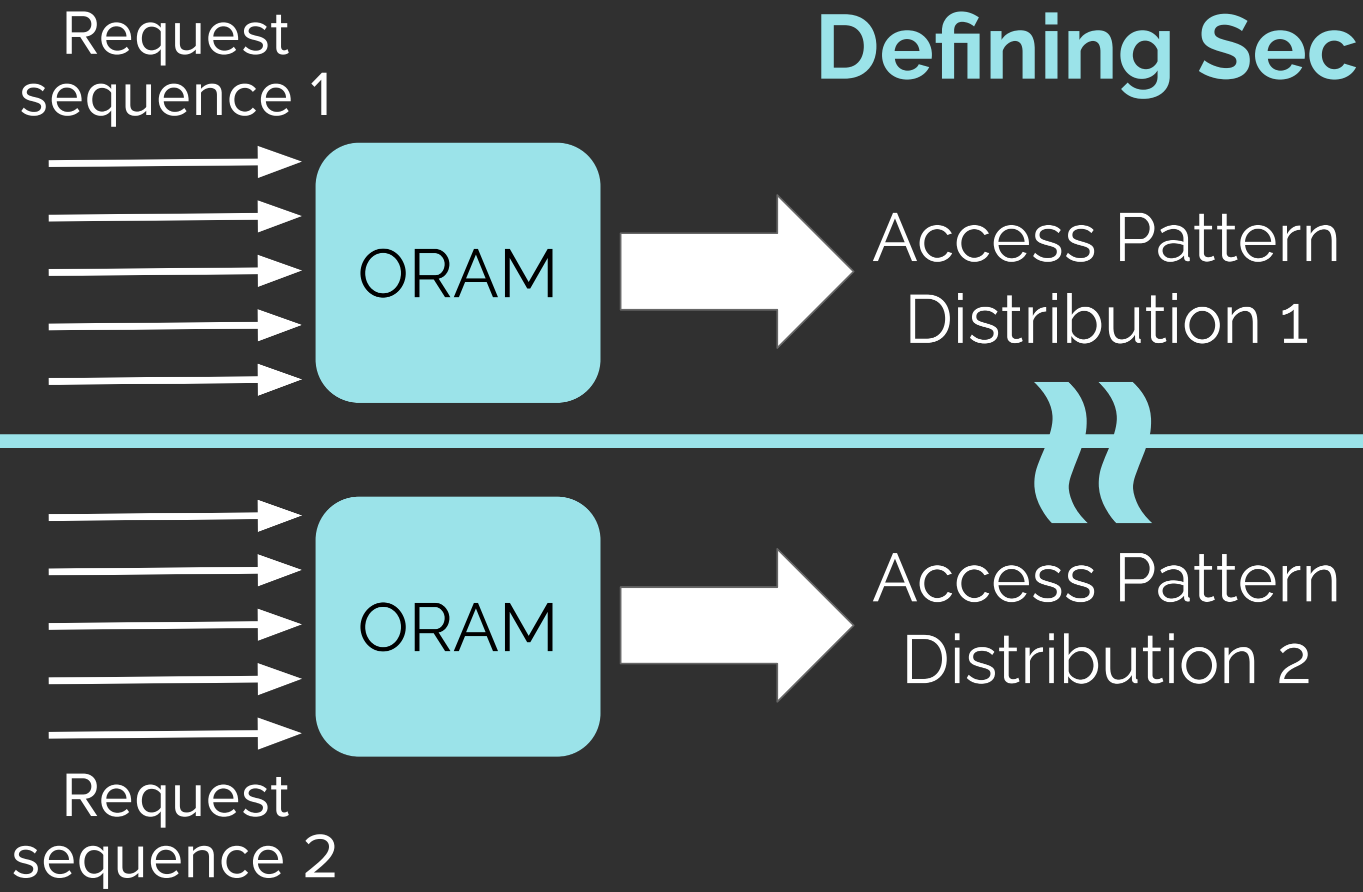
Defining Security

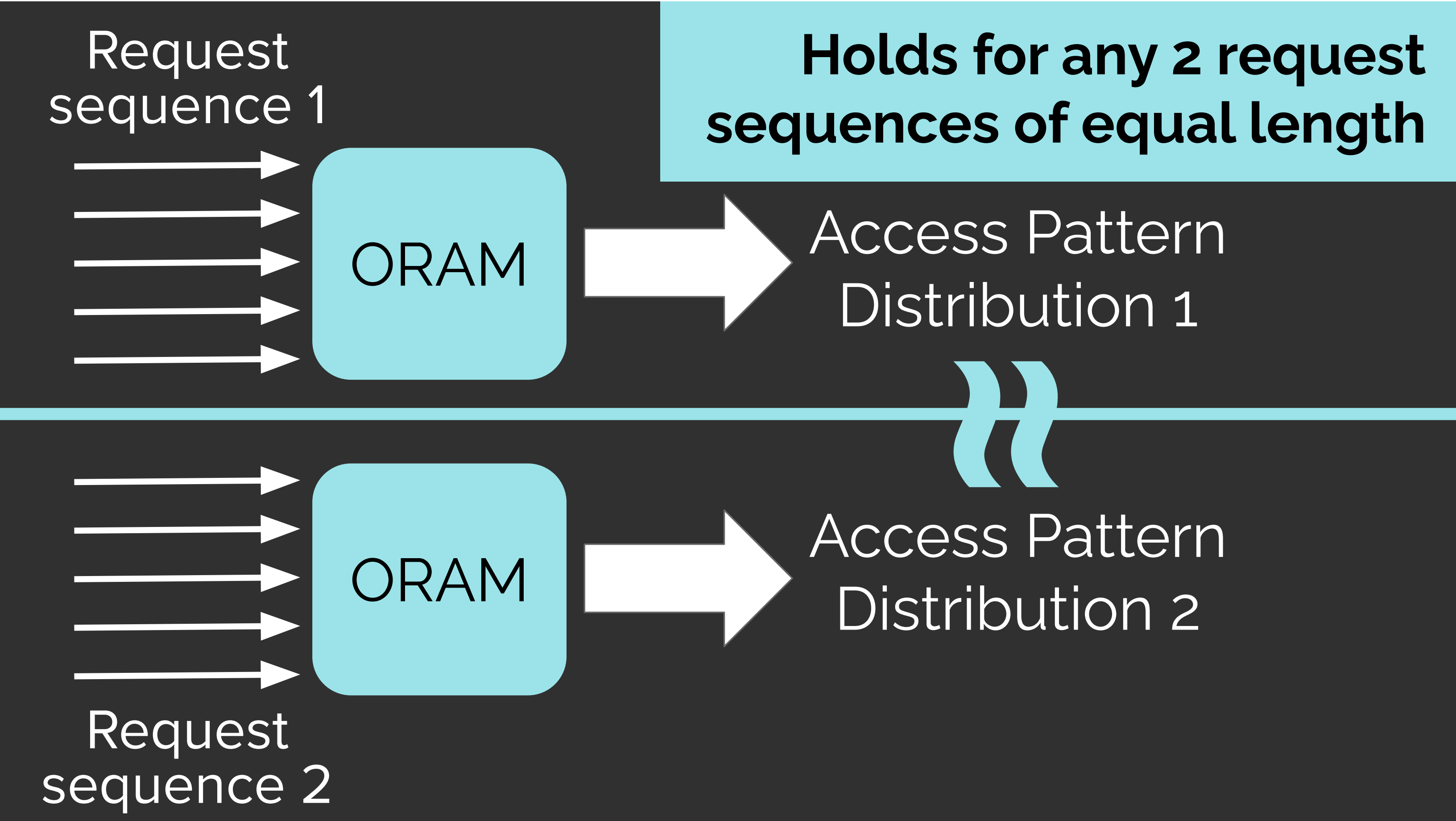


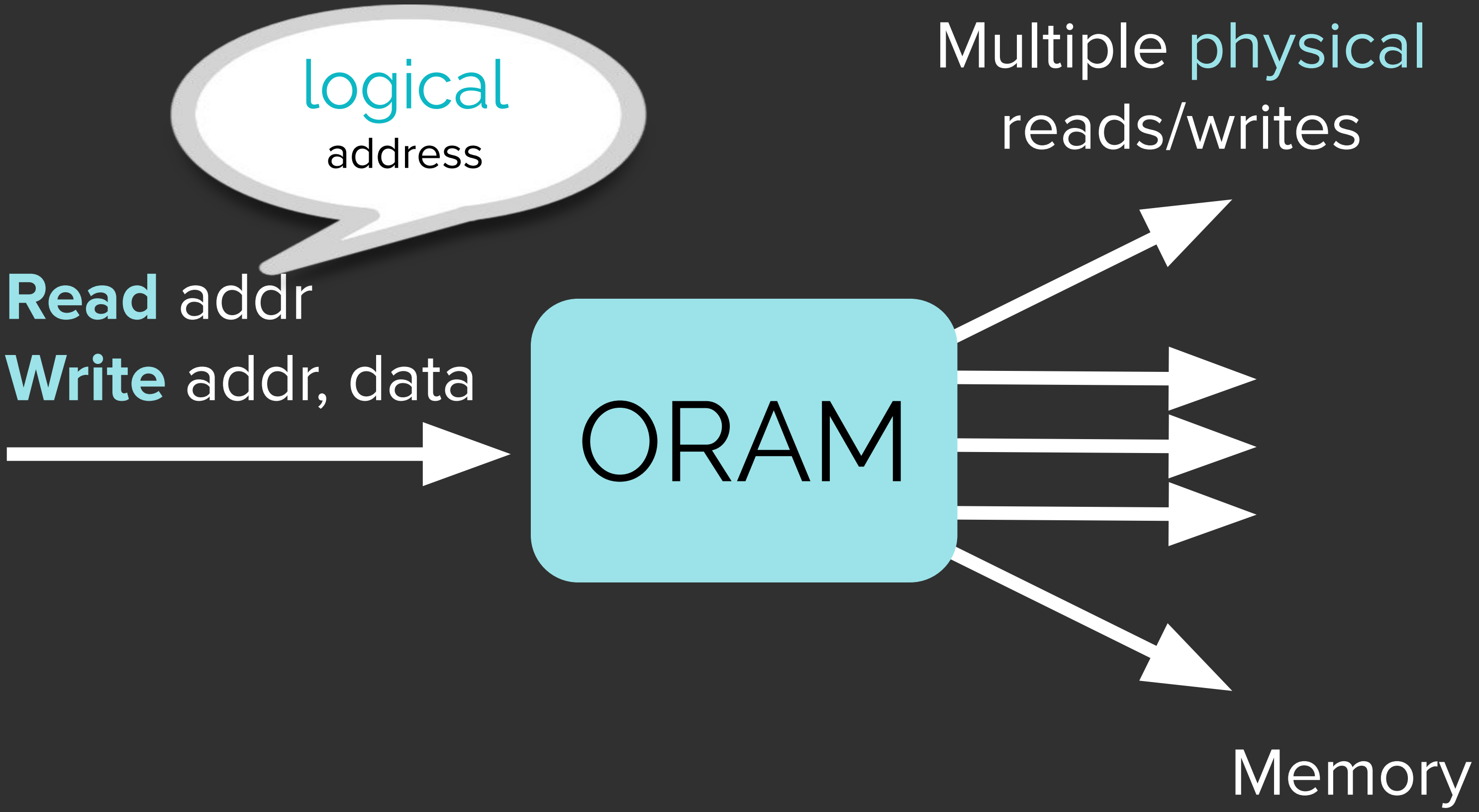
Defining Security

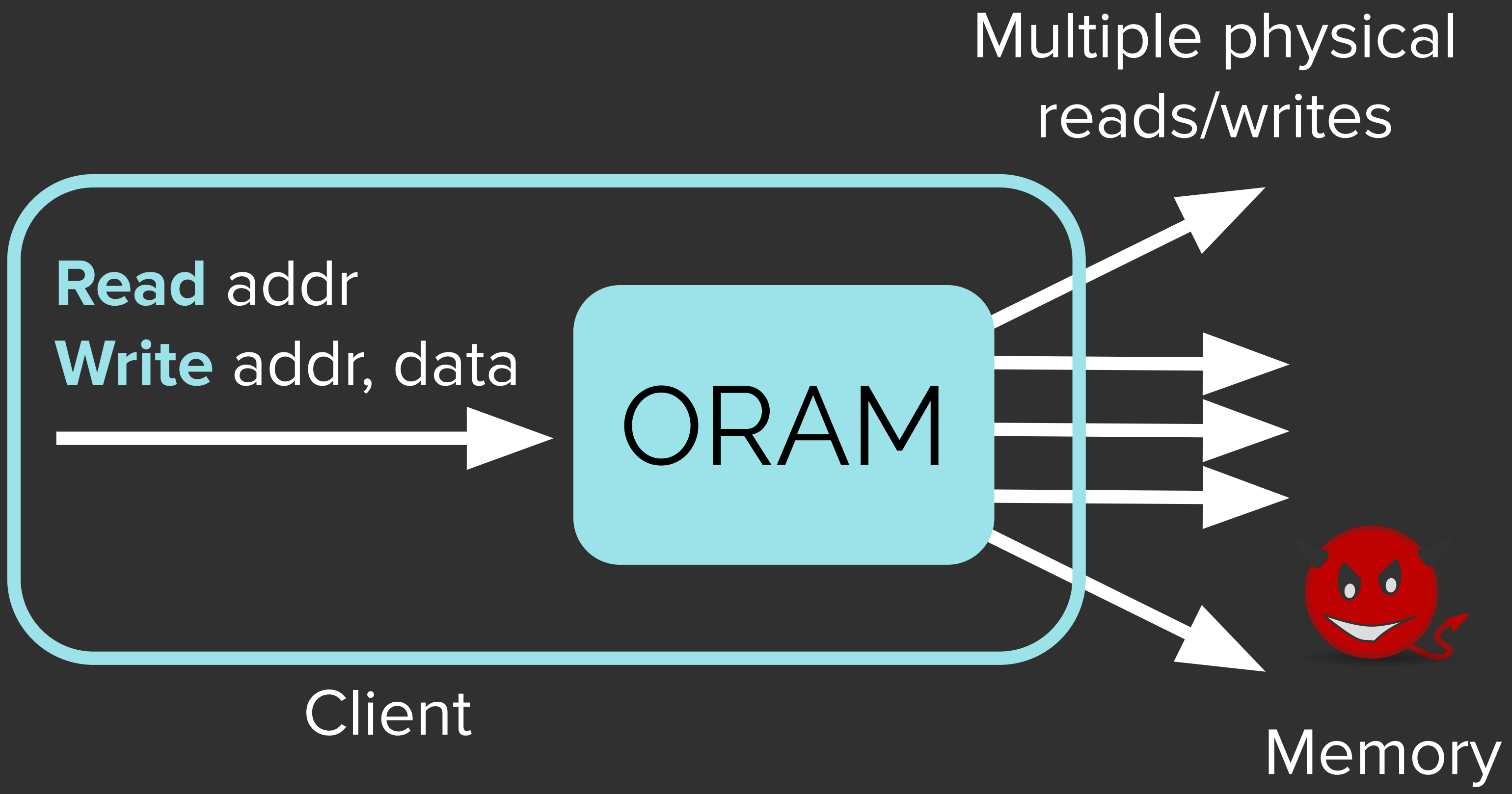


Defining Security




Holds for any 2 request sequences of equal length
Request sequence 1
ORAM
Access Pattern Distribution 1
≈
Access Pattern Distribution 2
ORAM
Request sequence 2


logical
address
Multiple physical
reads/writes
Read addr
Write addr, data
ORAM
Memory


Multiple physical reads/writes
Read addr
Write addr, data
ORAM
Client
Memory

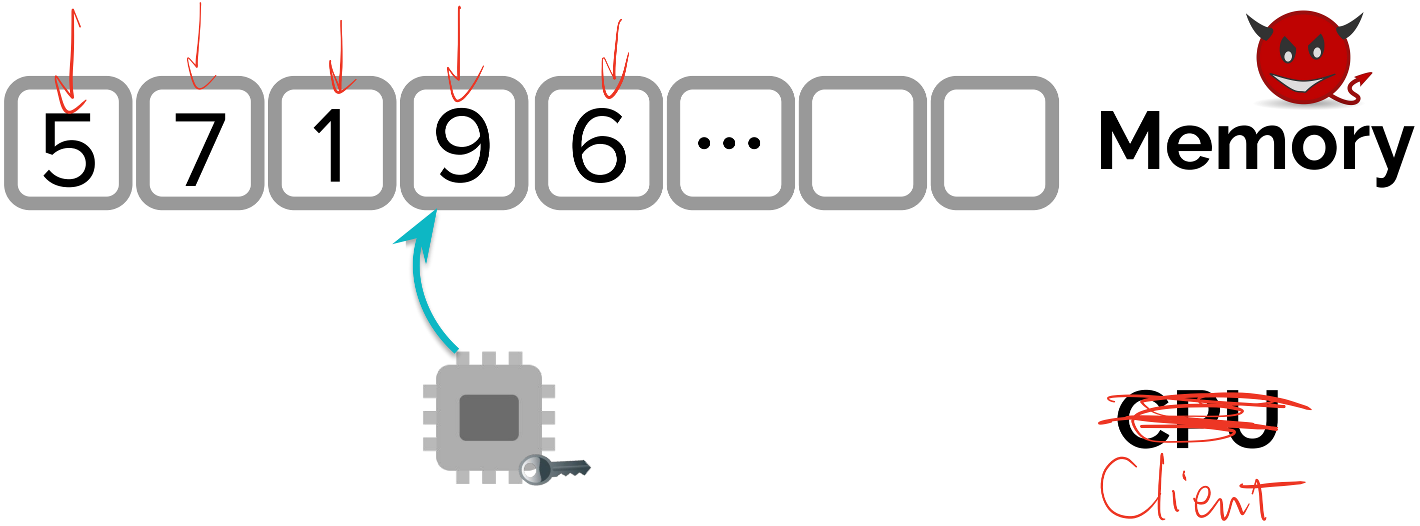
Assumptions

minimal storage unit

- All blocks are encrypted
- Client always reads a block, reencrypts it (possibly updating with new contents), and writes back

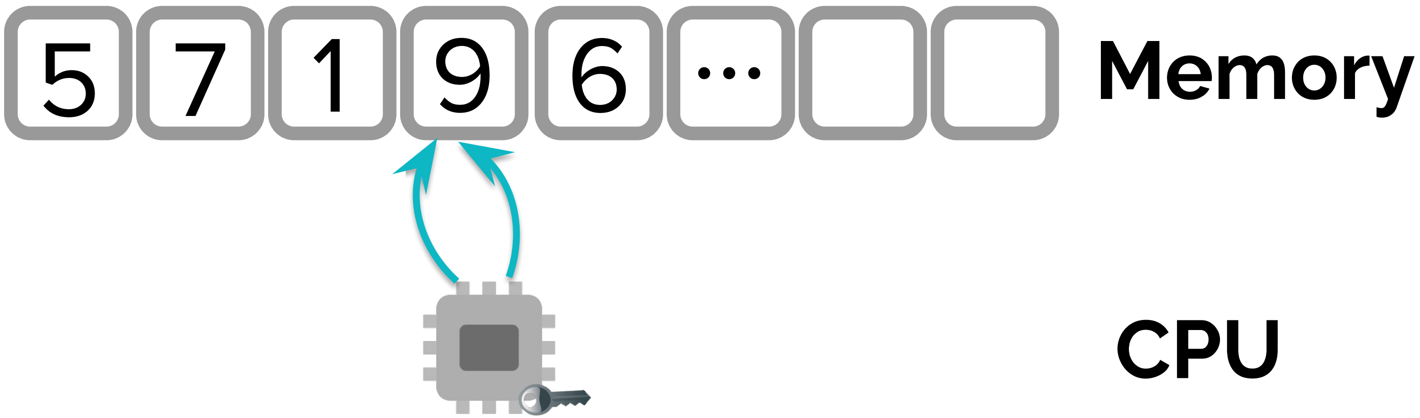
1 2 3 4 5 2 2 2 2 2

Strawman: permute blocks in memory

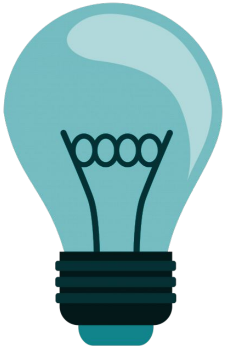


Strawman: provides one-time security!

e.g., leaks frequency, co-occurrence

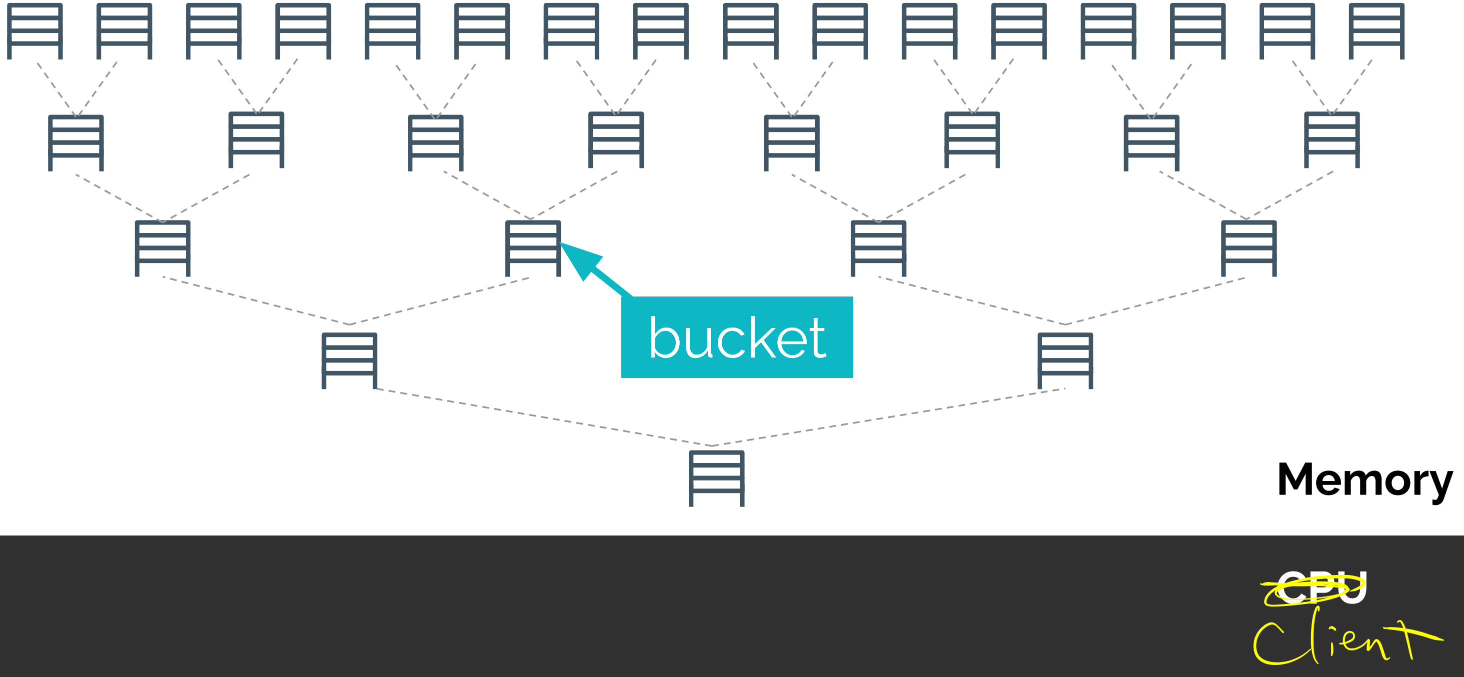


Blocks must move around in memory

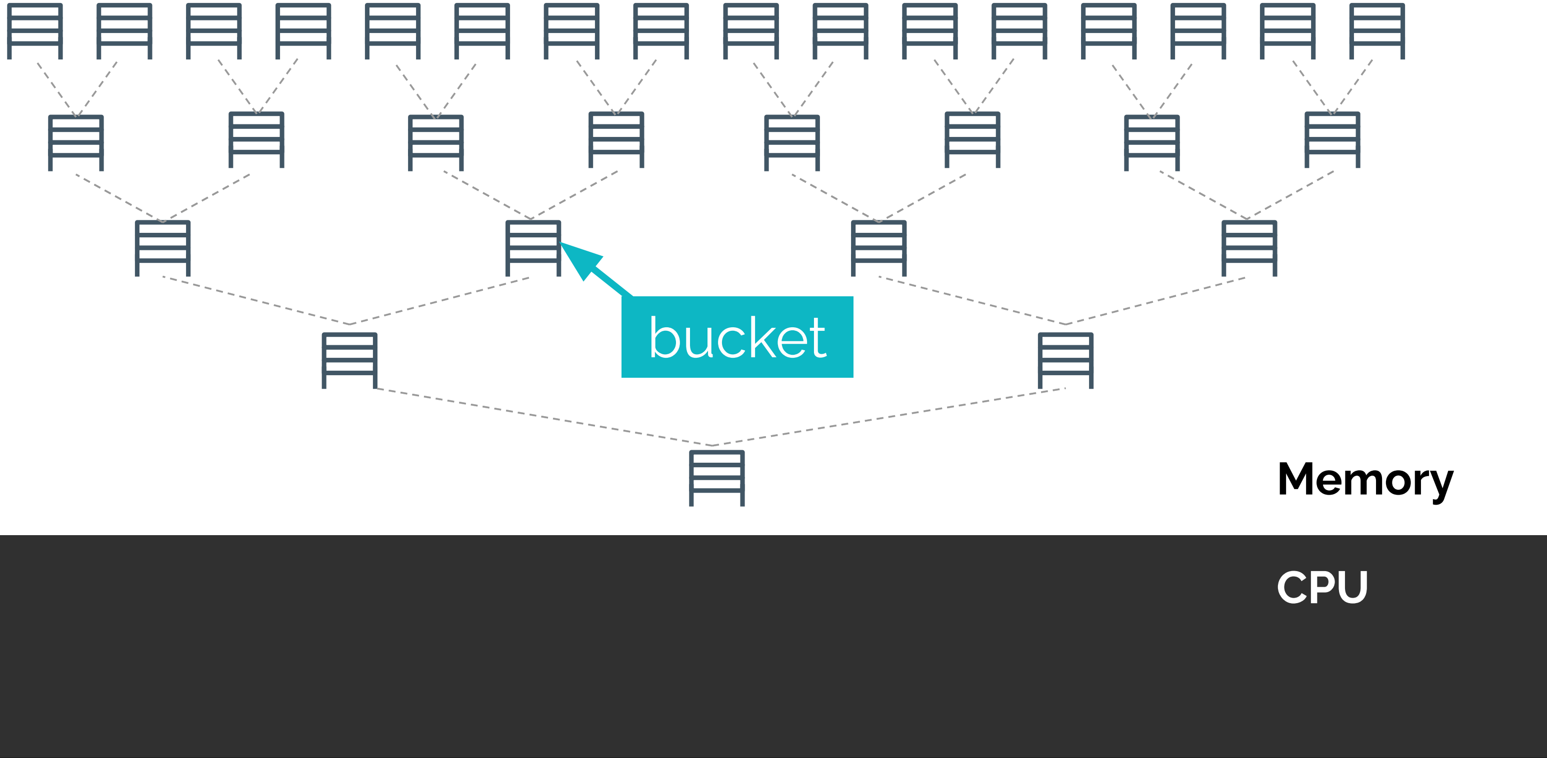


A tree-based paradigm for ORAMs

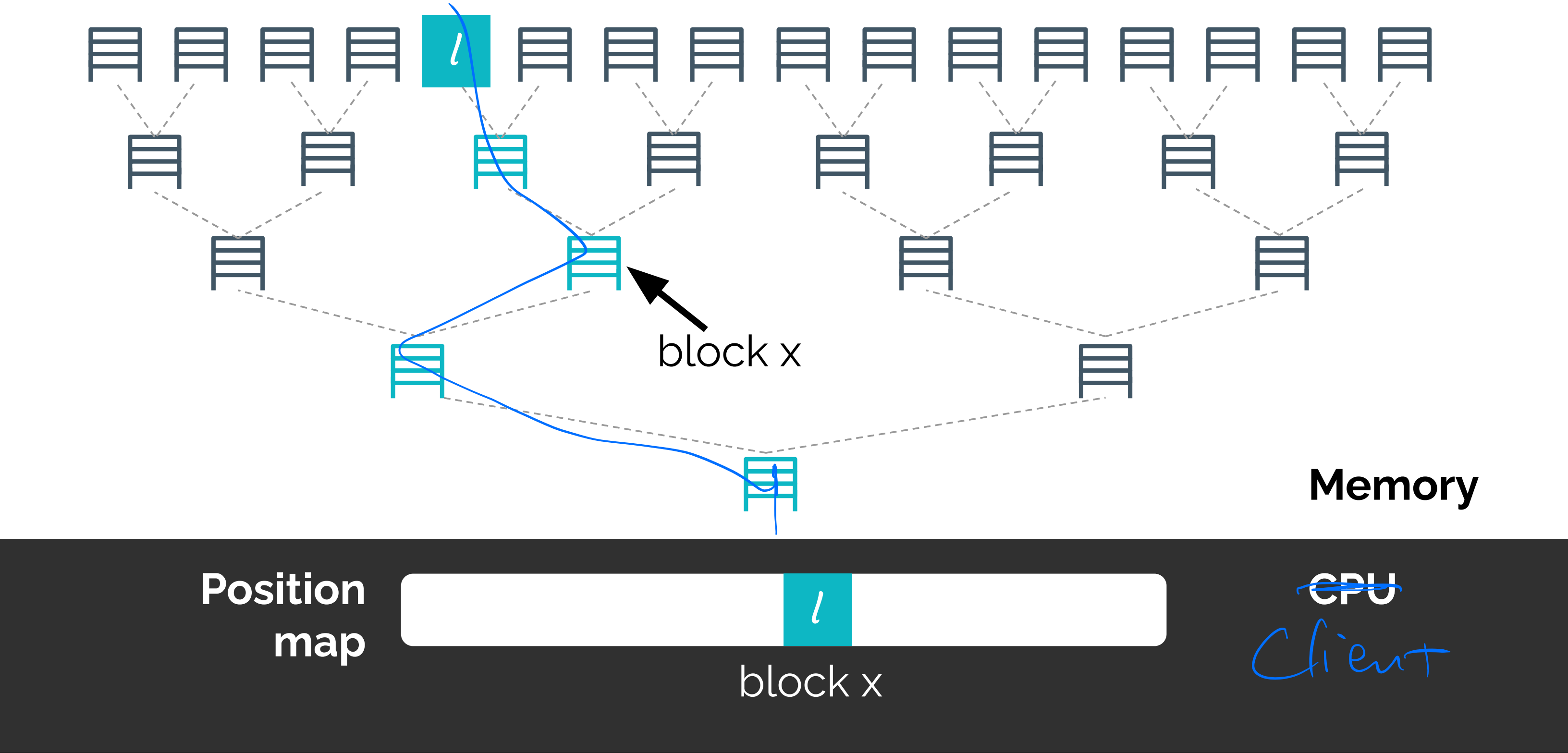
[SCSL'11]



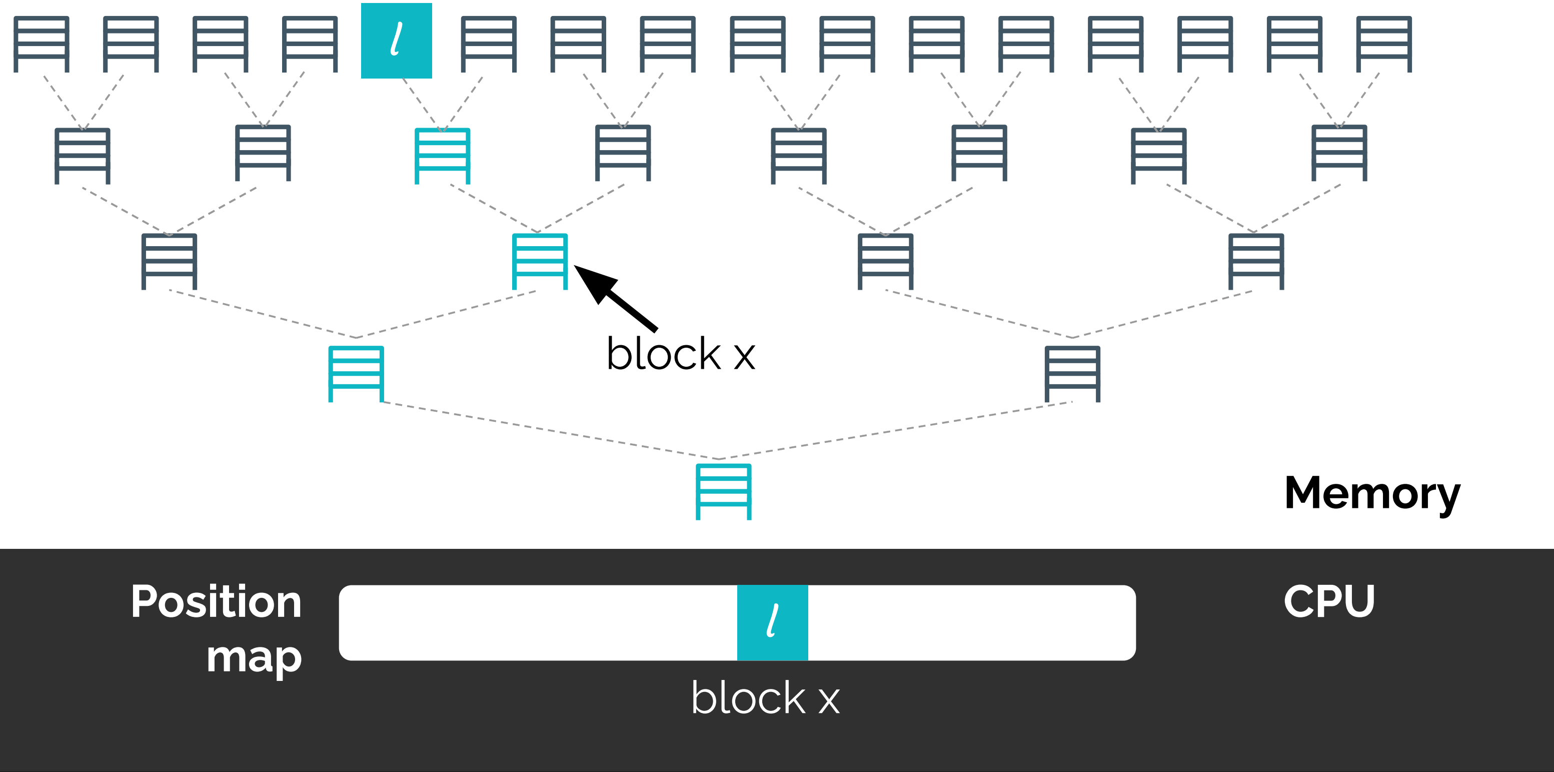
Each bucket stores real and dummy blocks



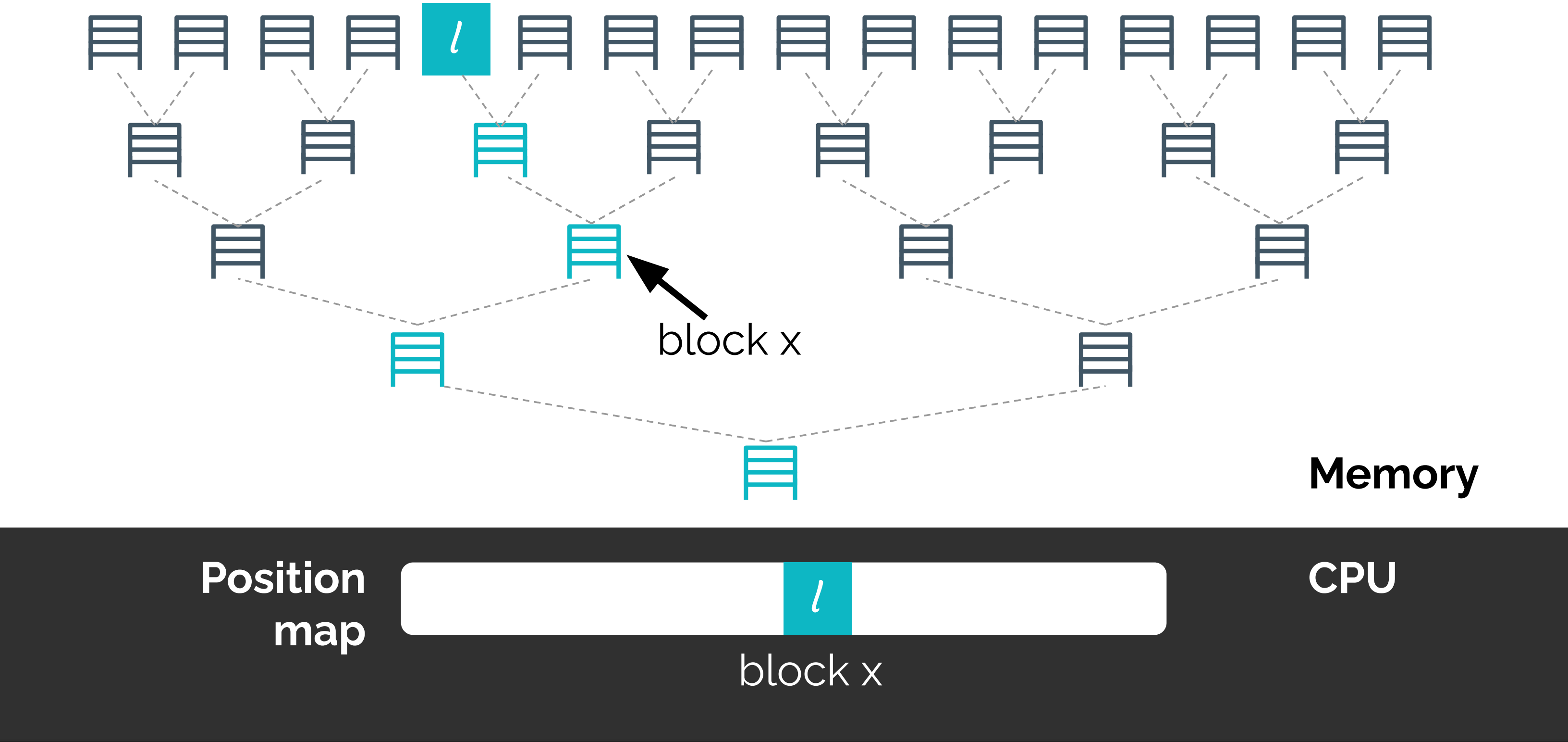
Path invariant: every block mapped to a random path



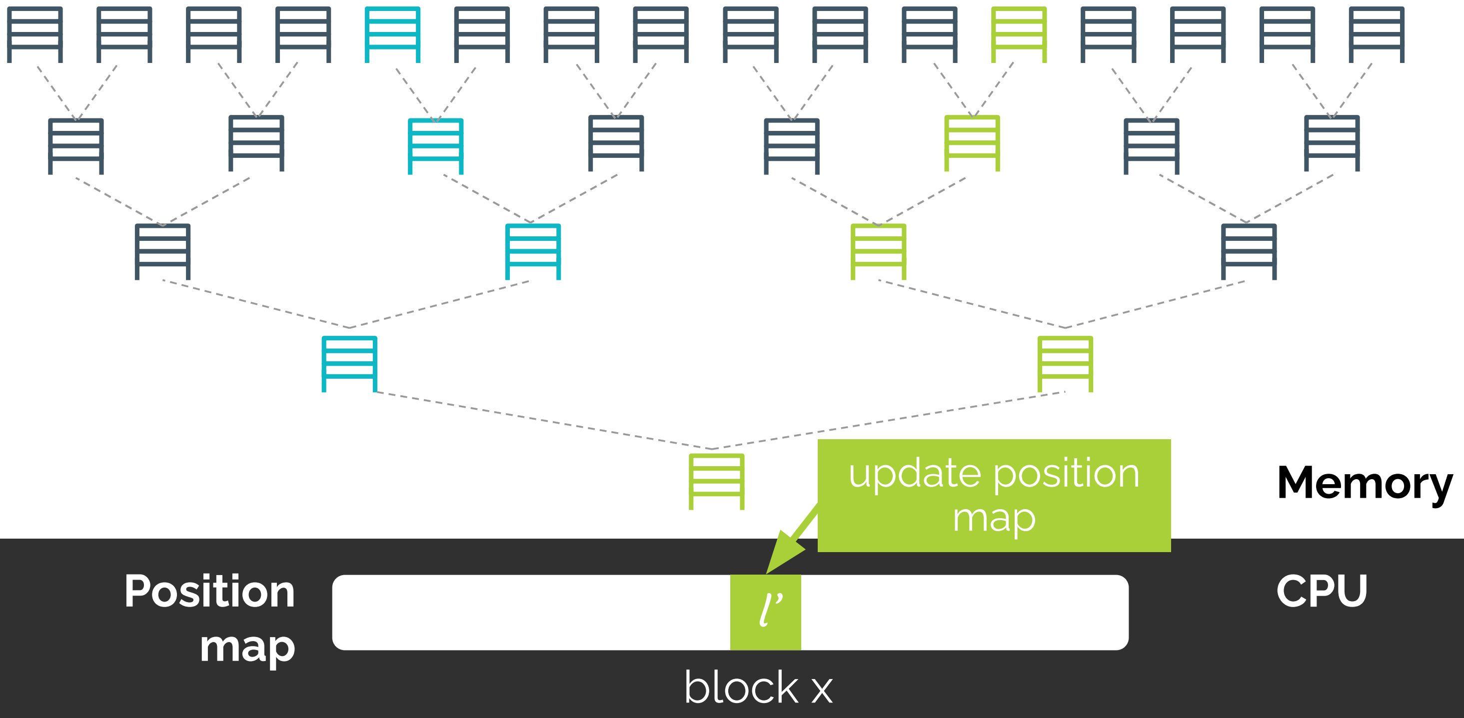
Reading a block is simple!



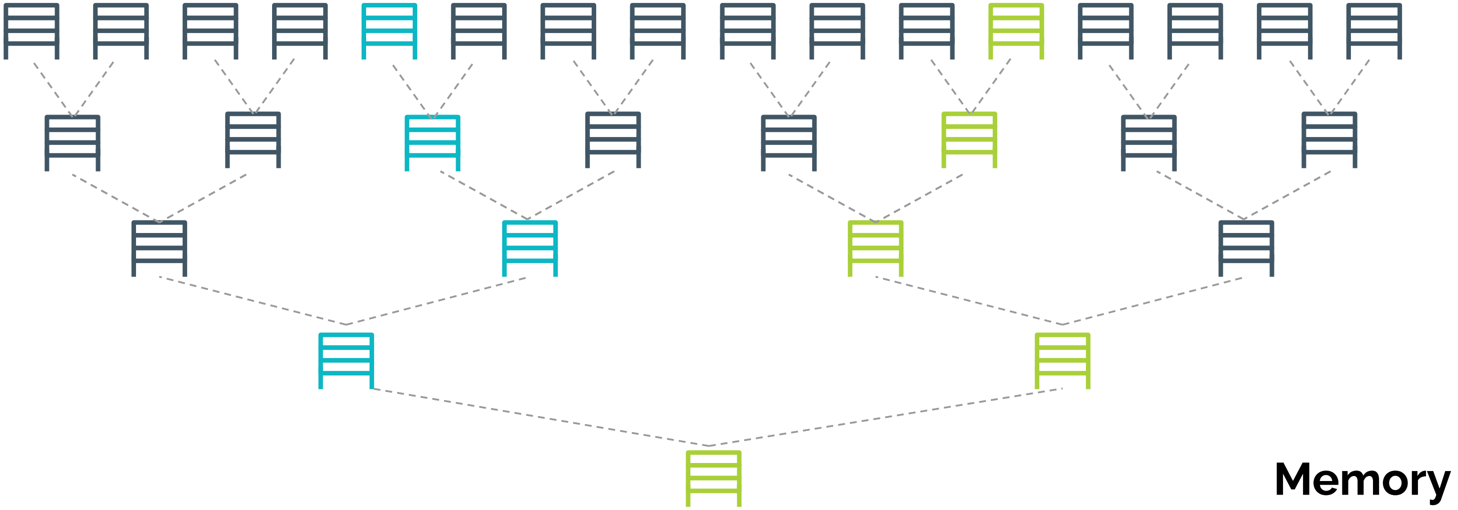
After being read, block x must relocate!



Pick a new random path and move x there



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



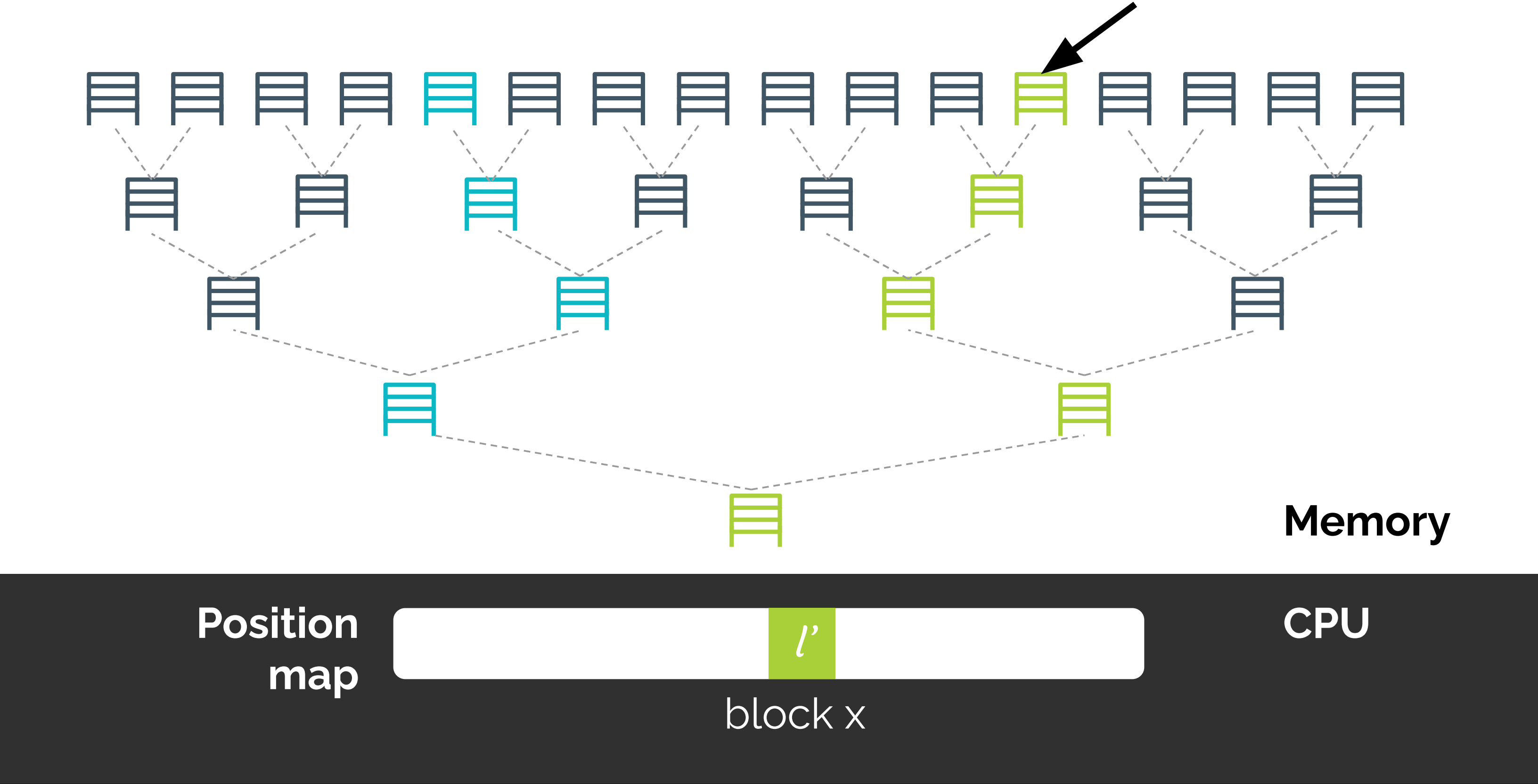
CPU

Position map

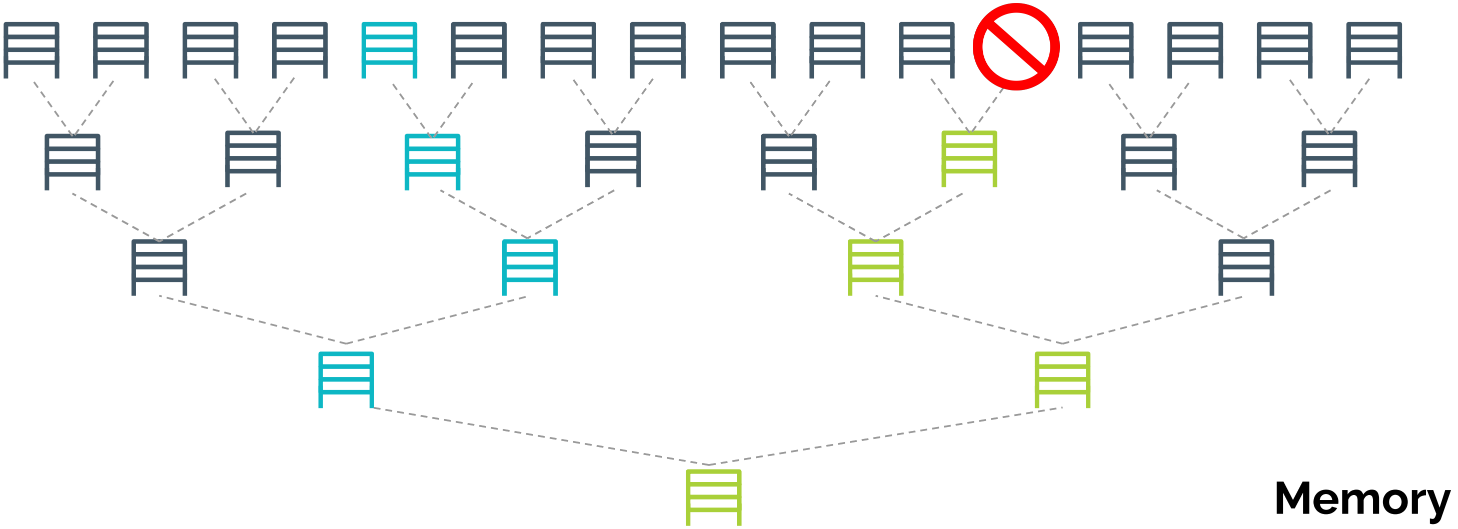
l'

block x

Can we write it to the leaf?



Can we write it to the leaf?



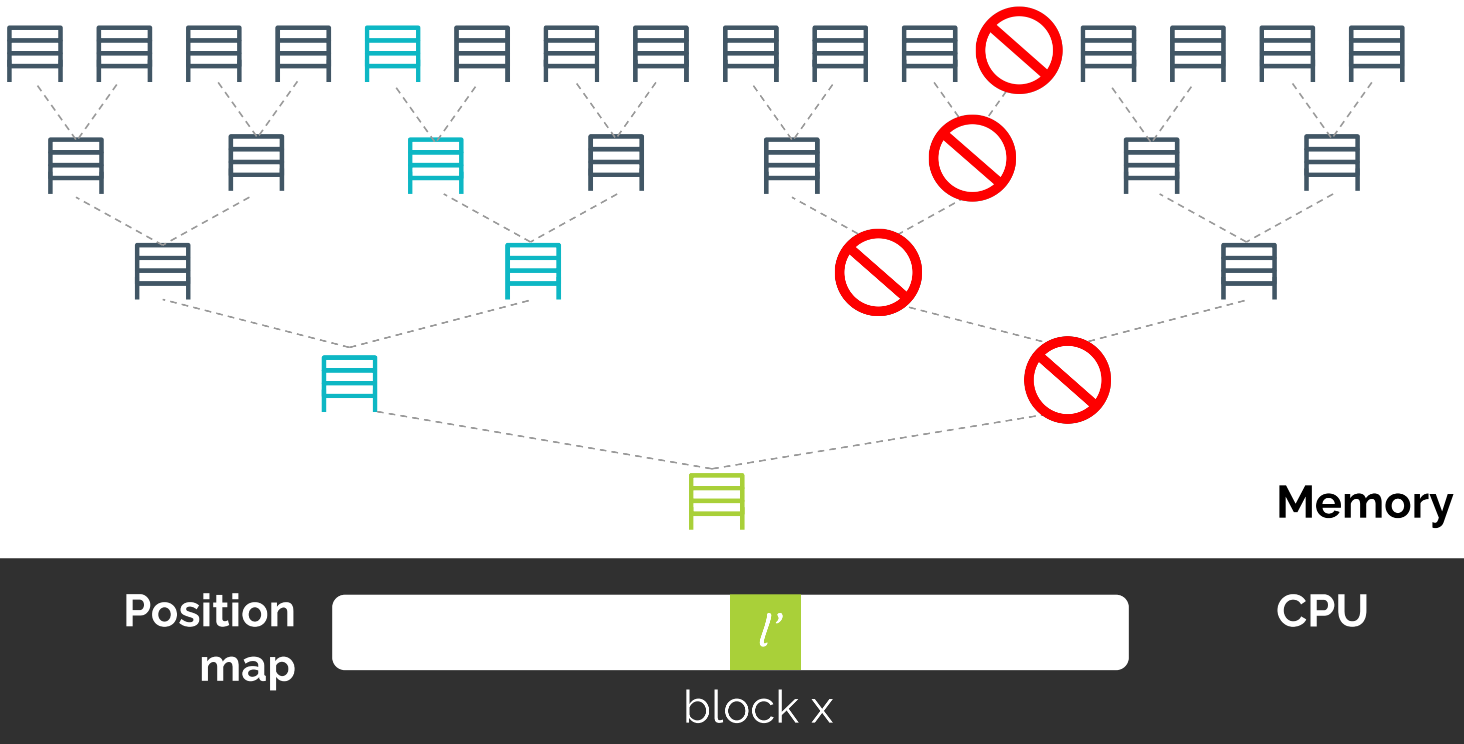
Position map

CPU

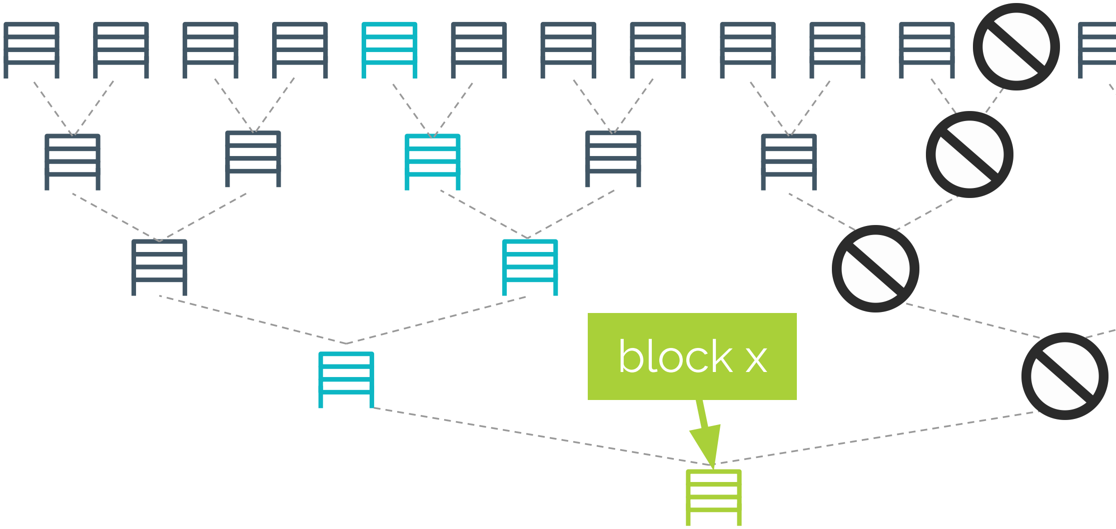
l'

block x

Writing to any non-root bucket leaks information



Write it to the root!



Memory

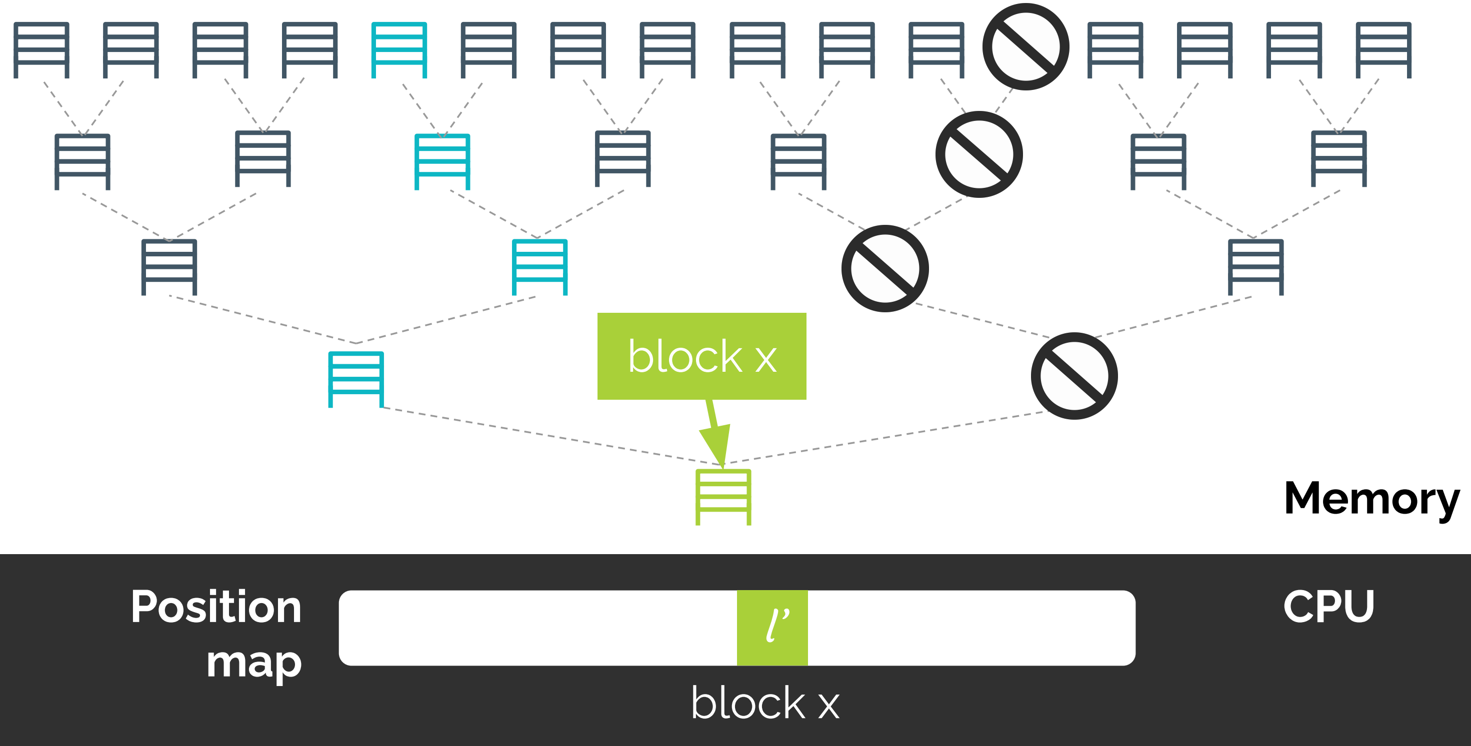
Position map

l'

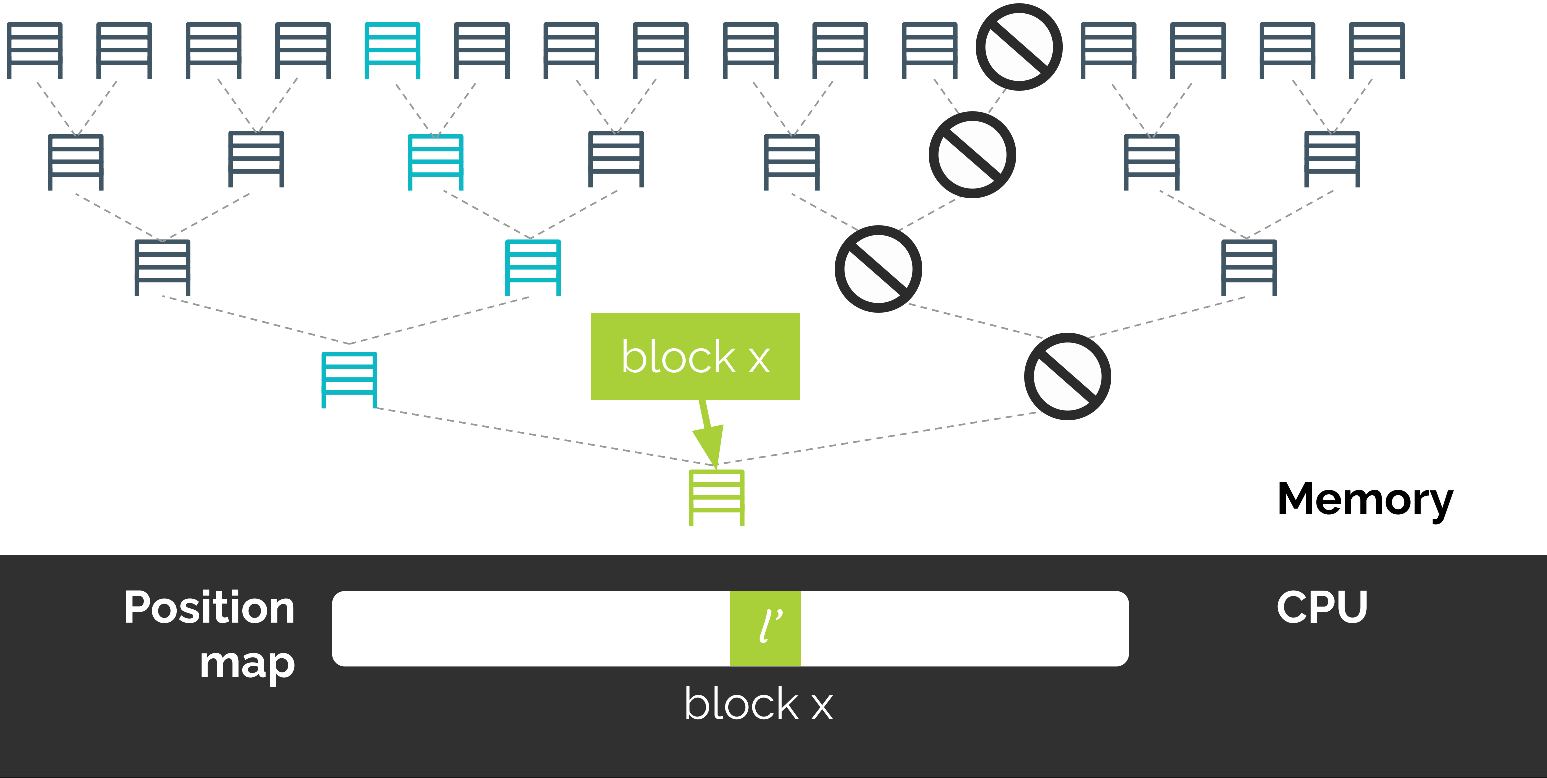
block x

CPU

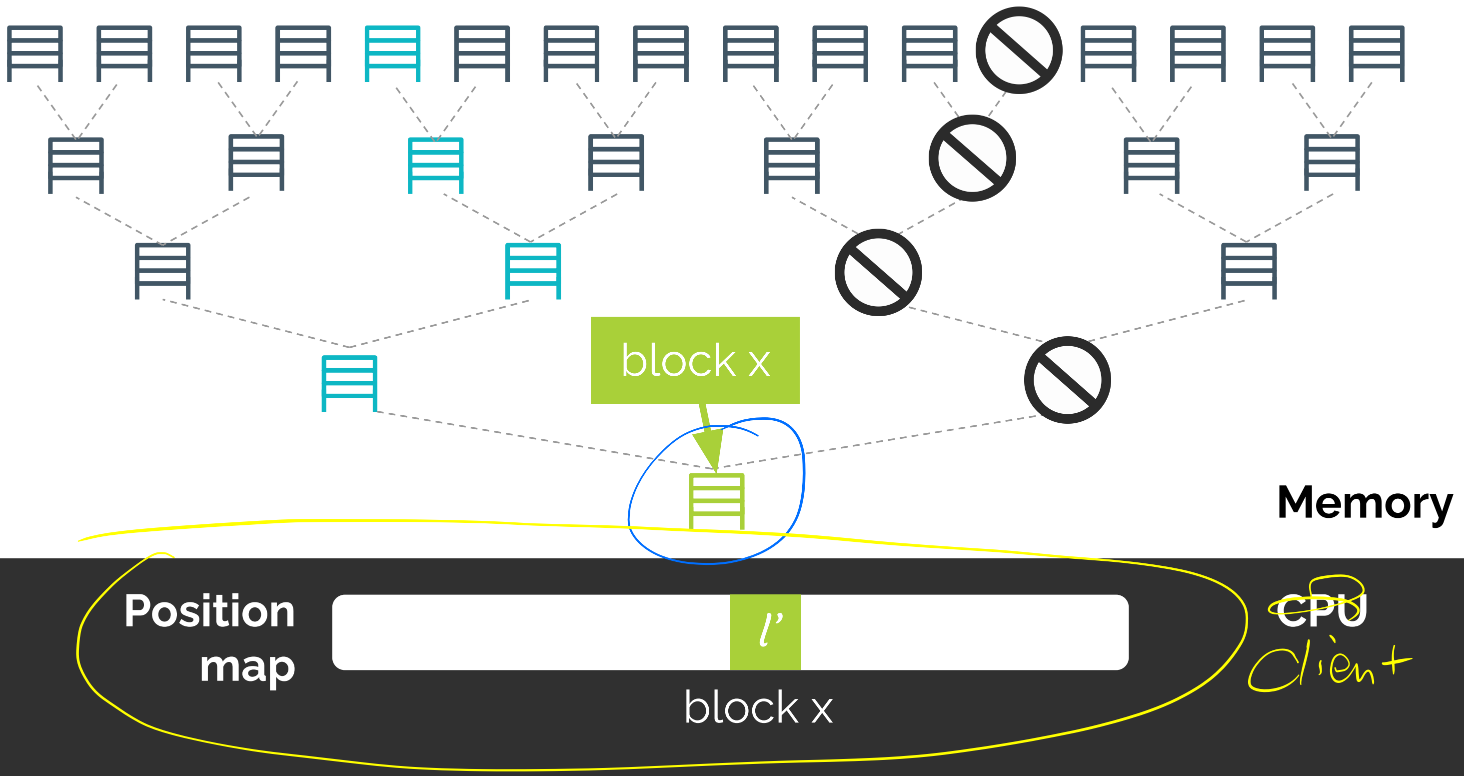
Security: every request, visit a **random** path that has **not** been revealed



Problem?



Problem: root will overflow



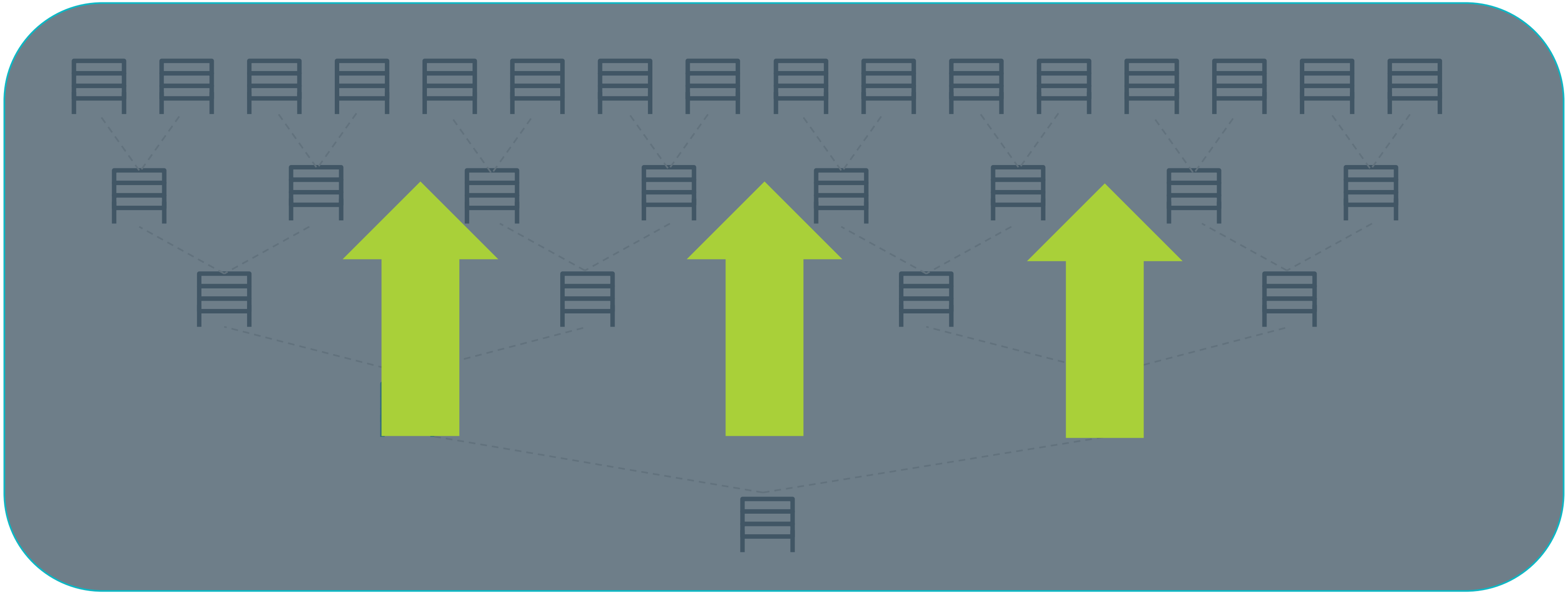
Further improvements

Remove position map

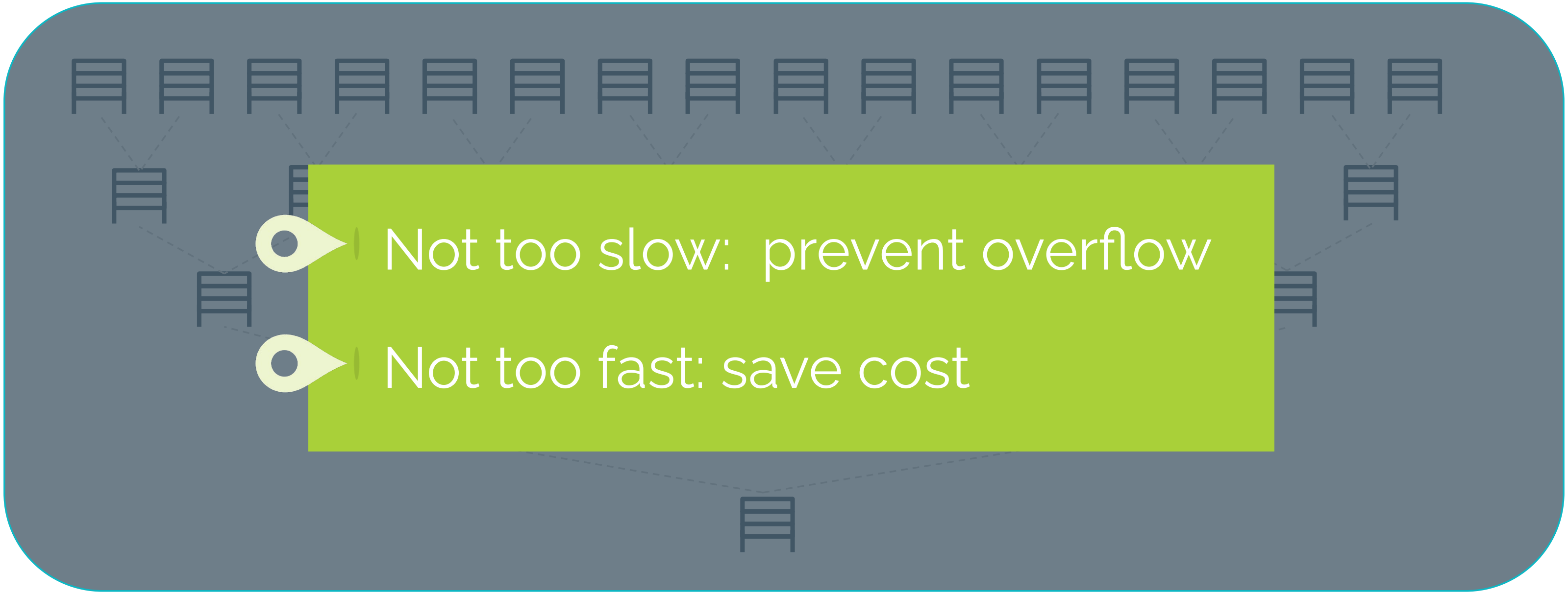
simple

Resolve overflow

A background eviction process percolates blocks upwards

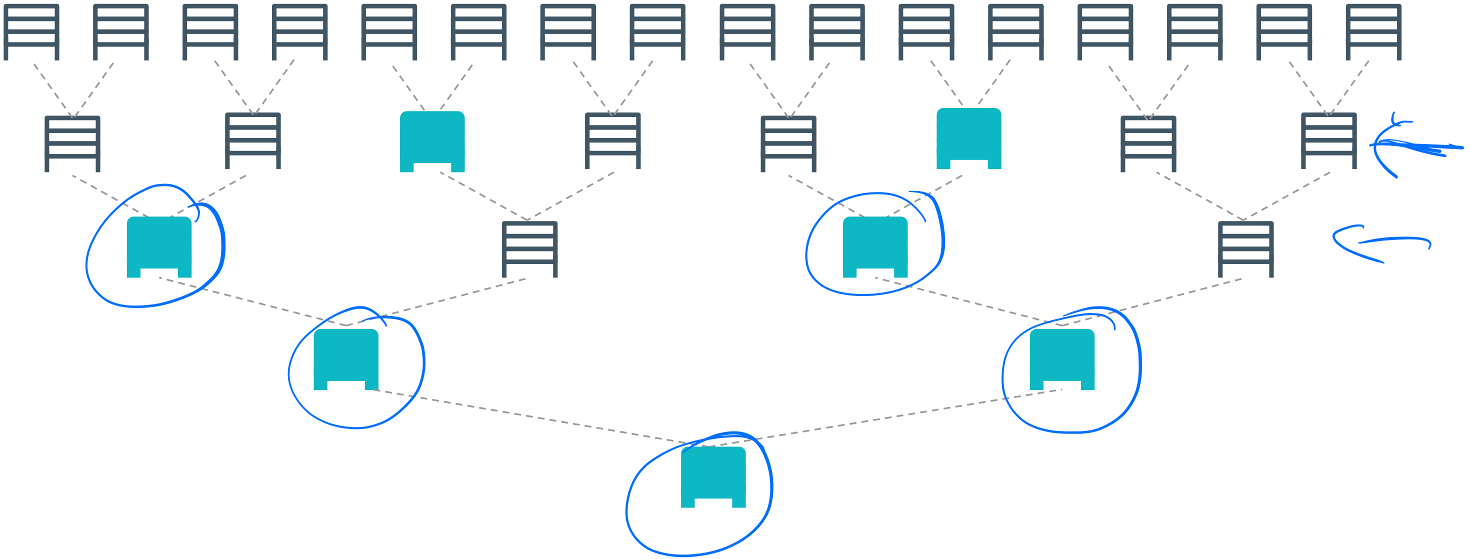


A background eviction process percolates blocks upwards

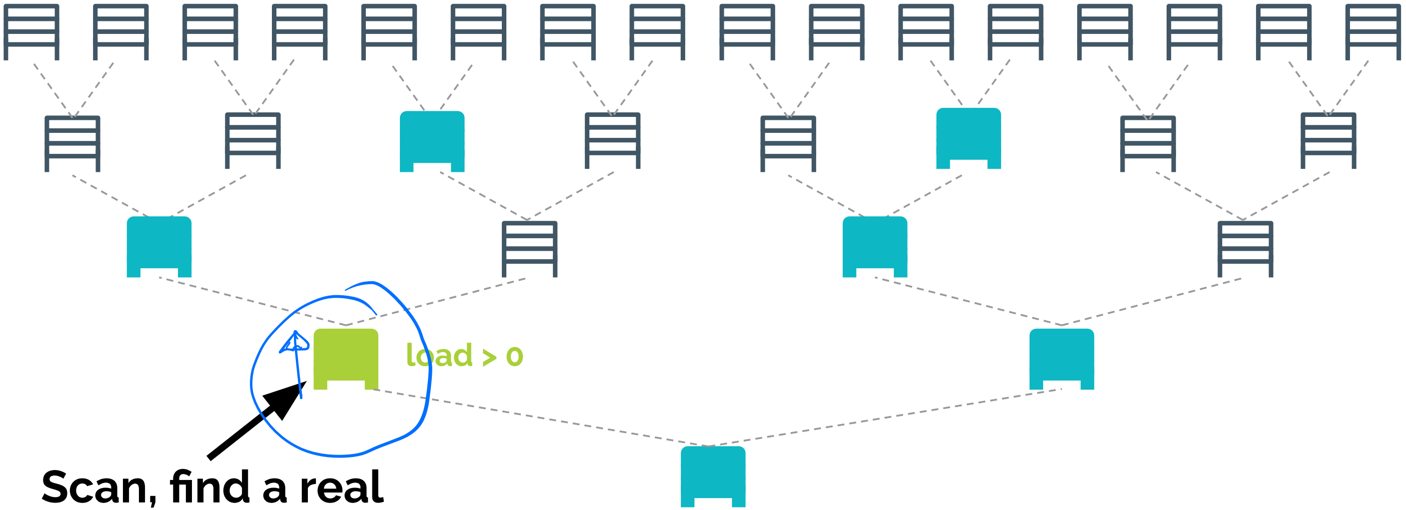


- Not too slow: prevent overflow
- Not too fast: save cost

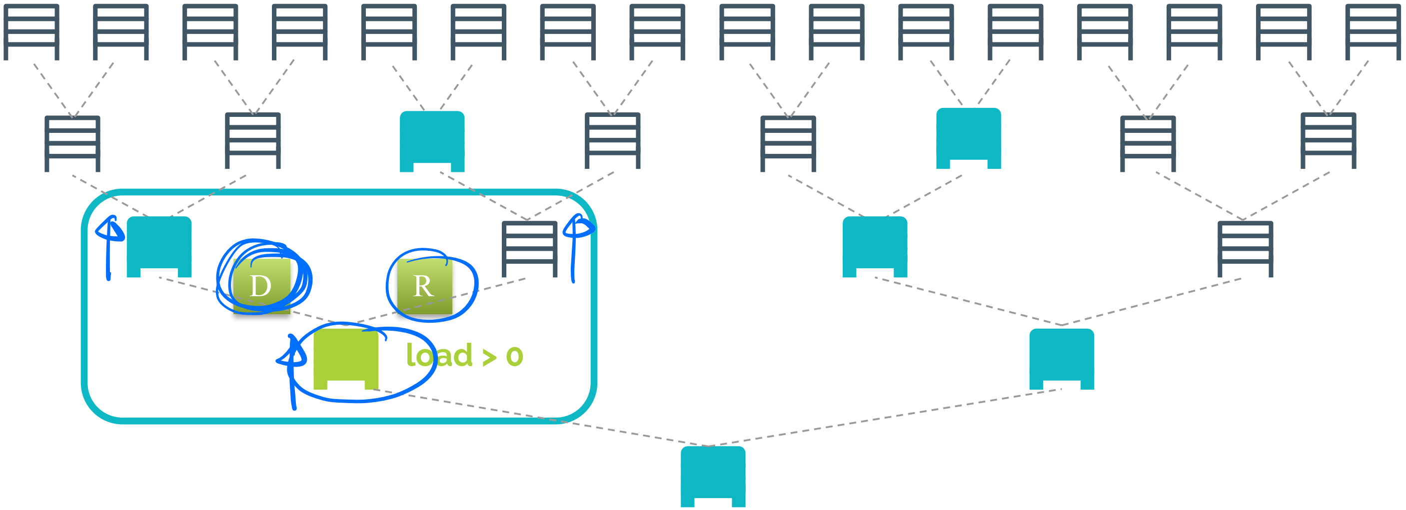
Every request: pick 2 random buckets per level to evict



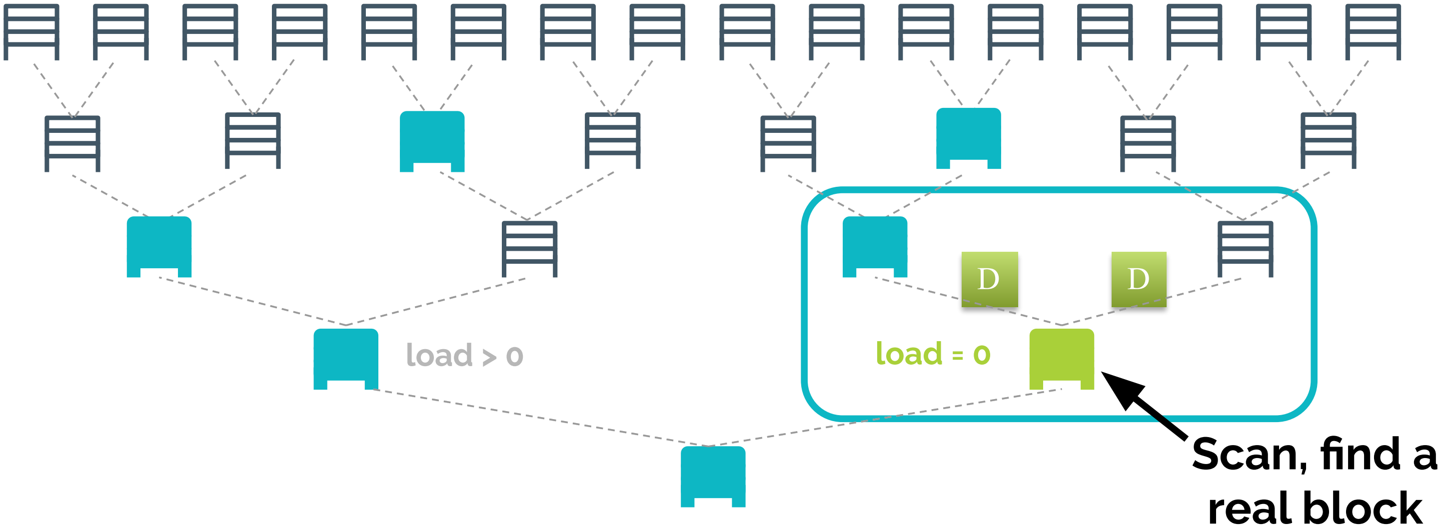
Every request: pick 2 random buckets per level to evict



Scan, find a real block, write to a child

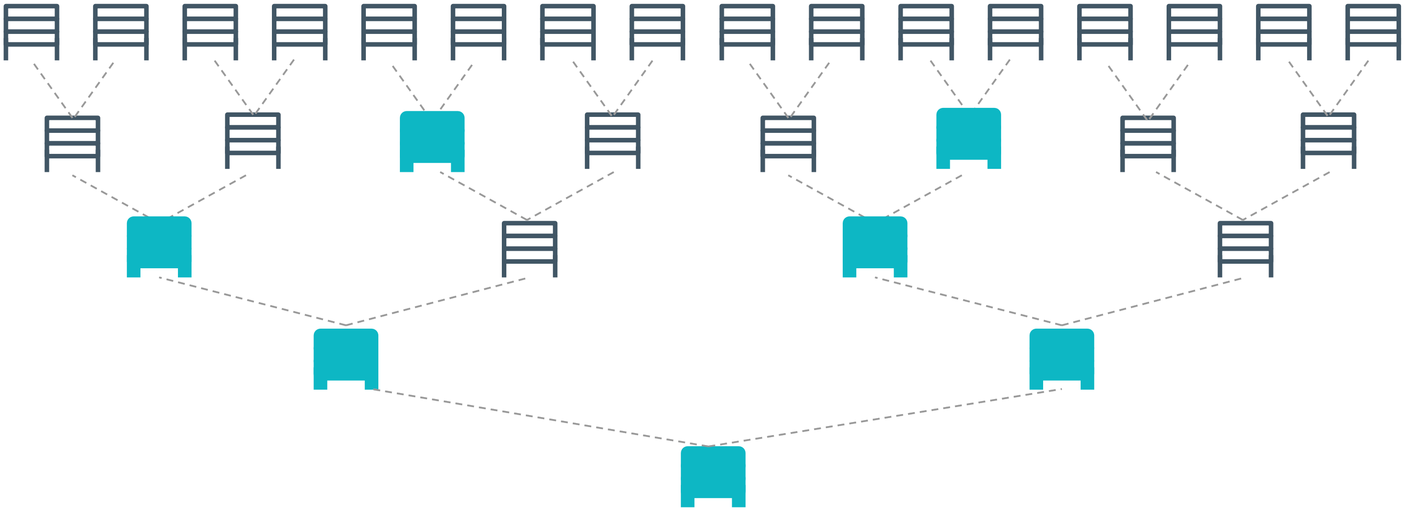


Dummy eviction ensures security

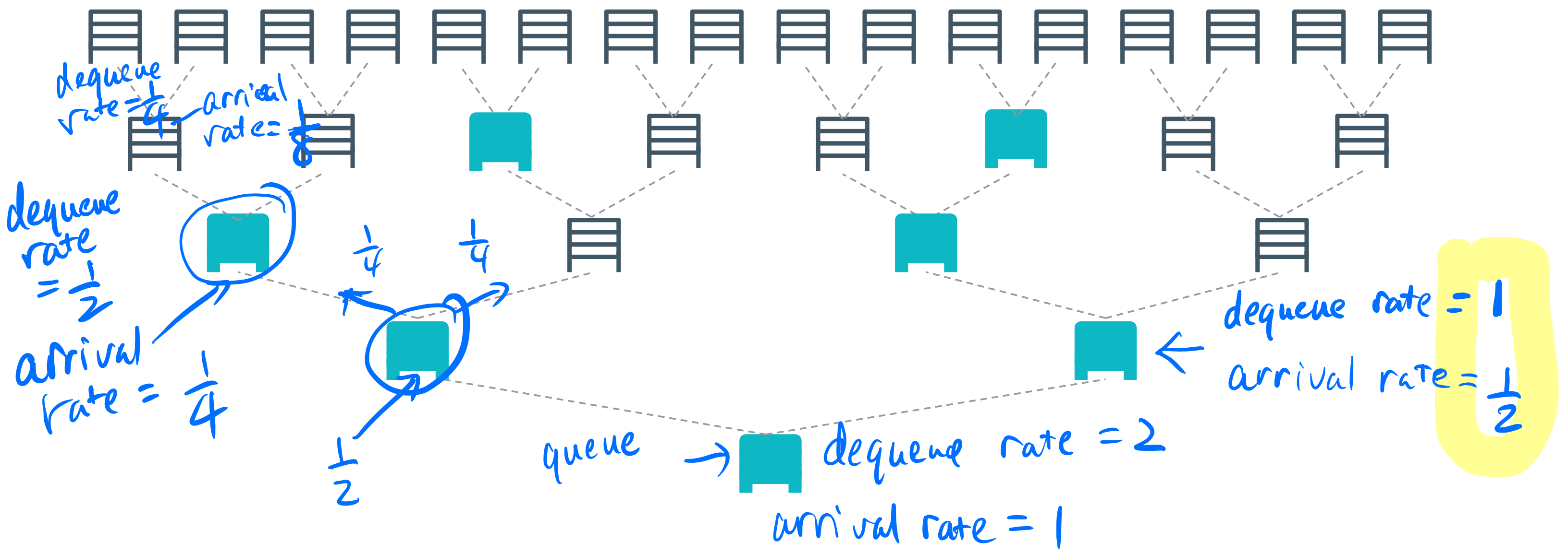


Dummy eviction ensures security

Eviction process does not leak information



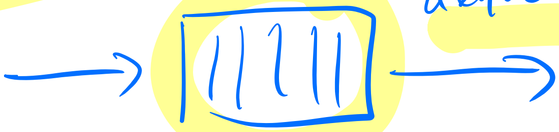
Thm: bucket size = log n $O(\log n)$ ⇒ no overflow w.h.p. $1 - \frac{1}{n^c}$ [SCSL'11]



Proof: use queuing theory and measure concentration bounds.

From 15259:

arrival rate p



dequeue rate $= 2p$

$$\Pr[\text{queue length} > R] \le \exp(-C \cdot R)$$

C: suitable constant.

in particular, suppose $R = C' \cdot \log N$

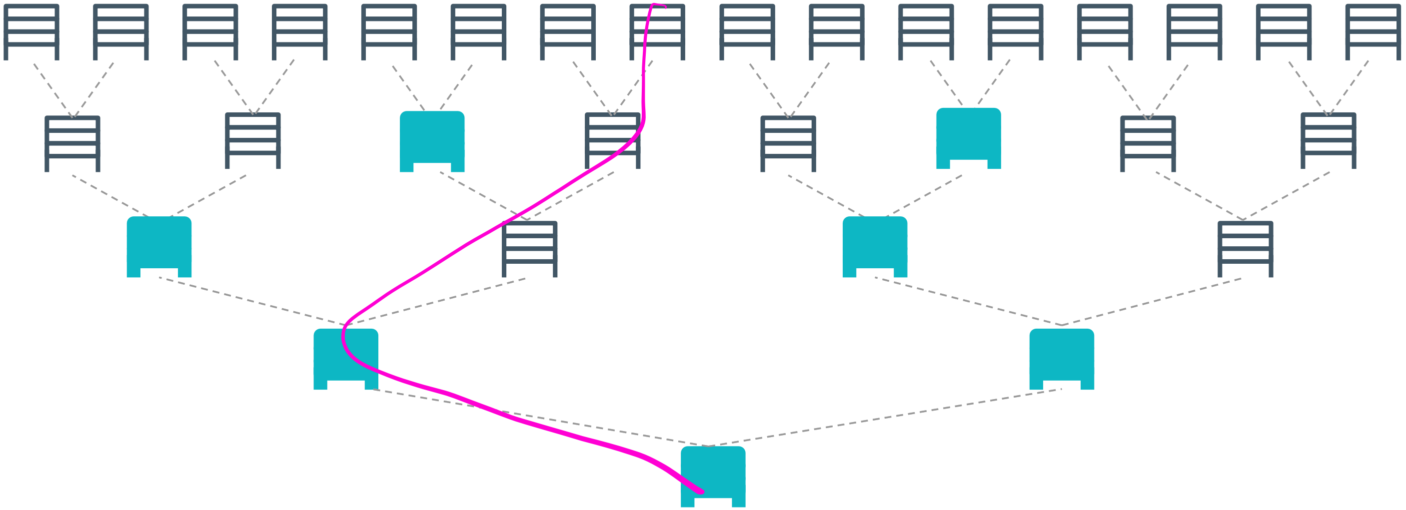
C': sufficiently large constant

$$\Pr[\text{queue length} > R] \le \frac{1}{N^{\varepsilon}}$$

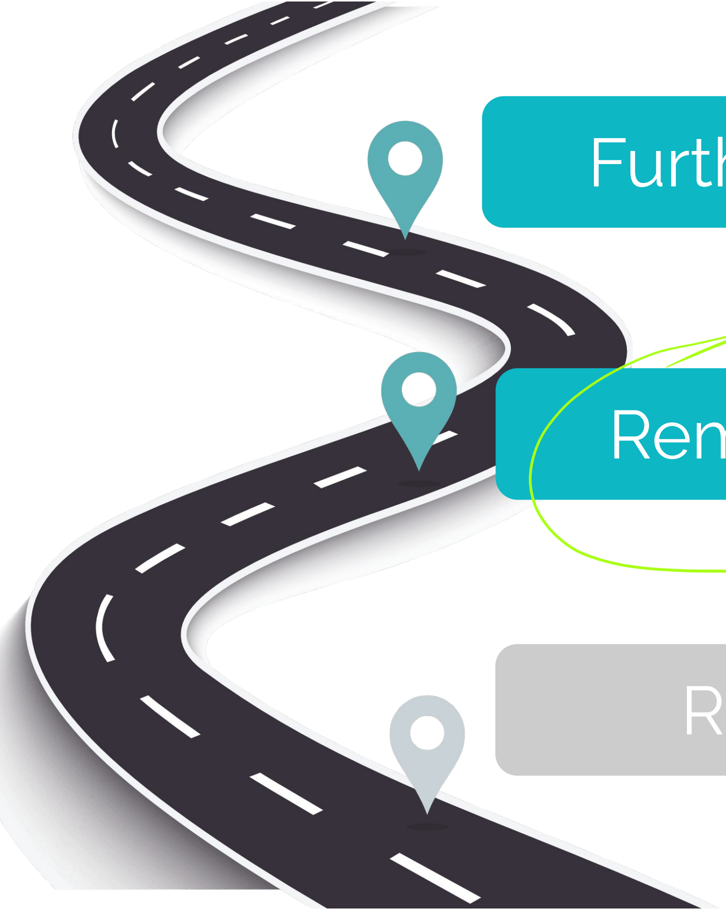
ε: constant.

discrete version of Burke's Thm

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



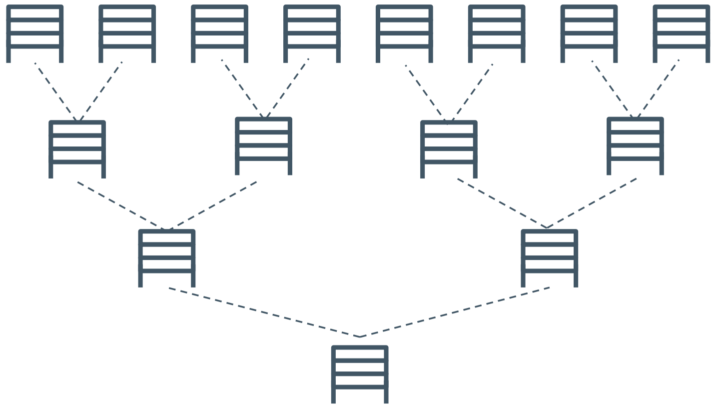
Every request incurs $O(\log^2 n)$ cost



Further improvements

Remove position map

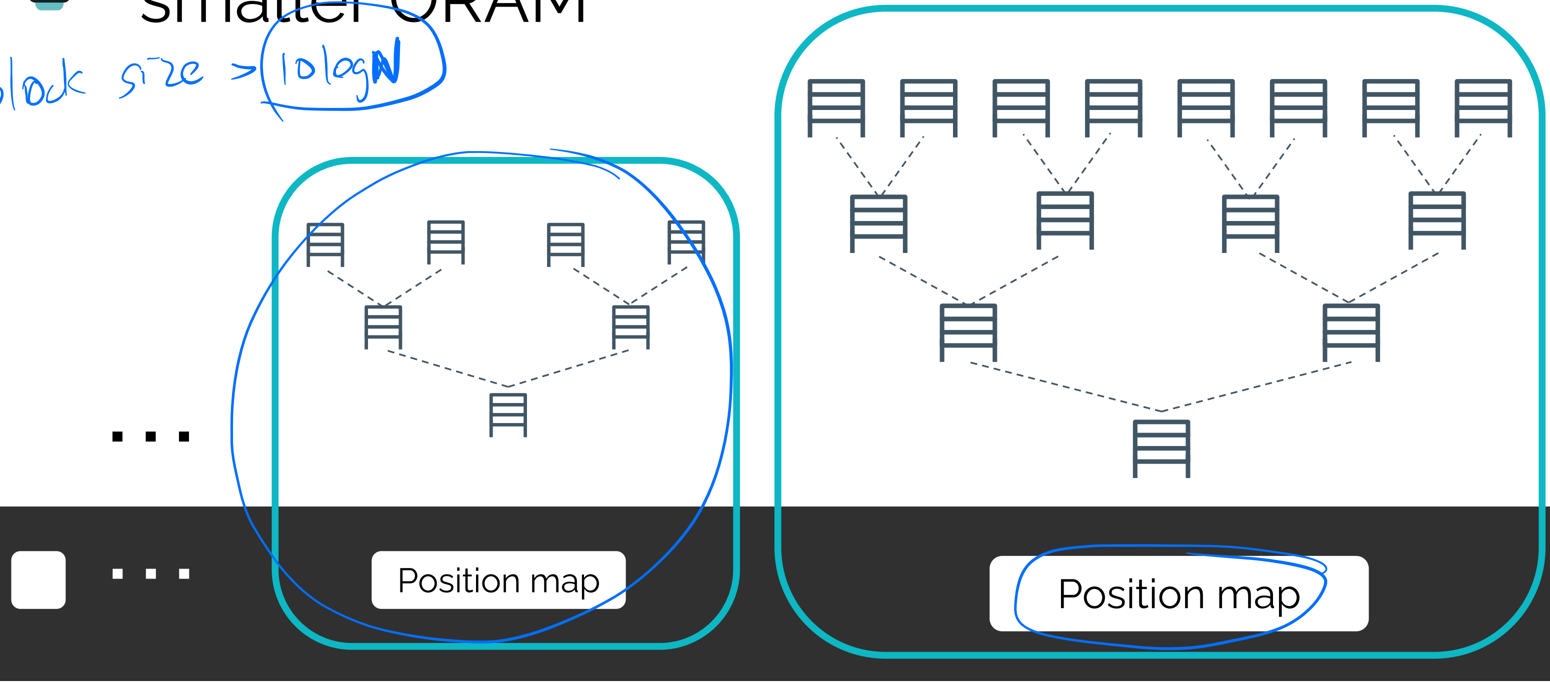
Resolve overflow



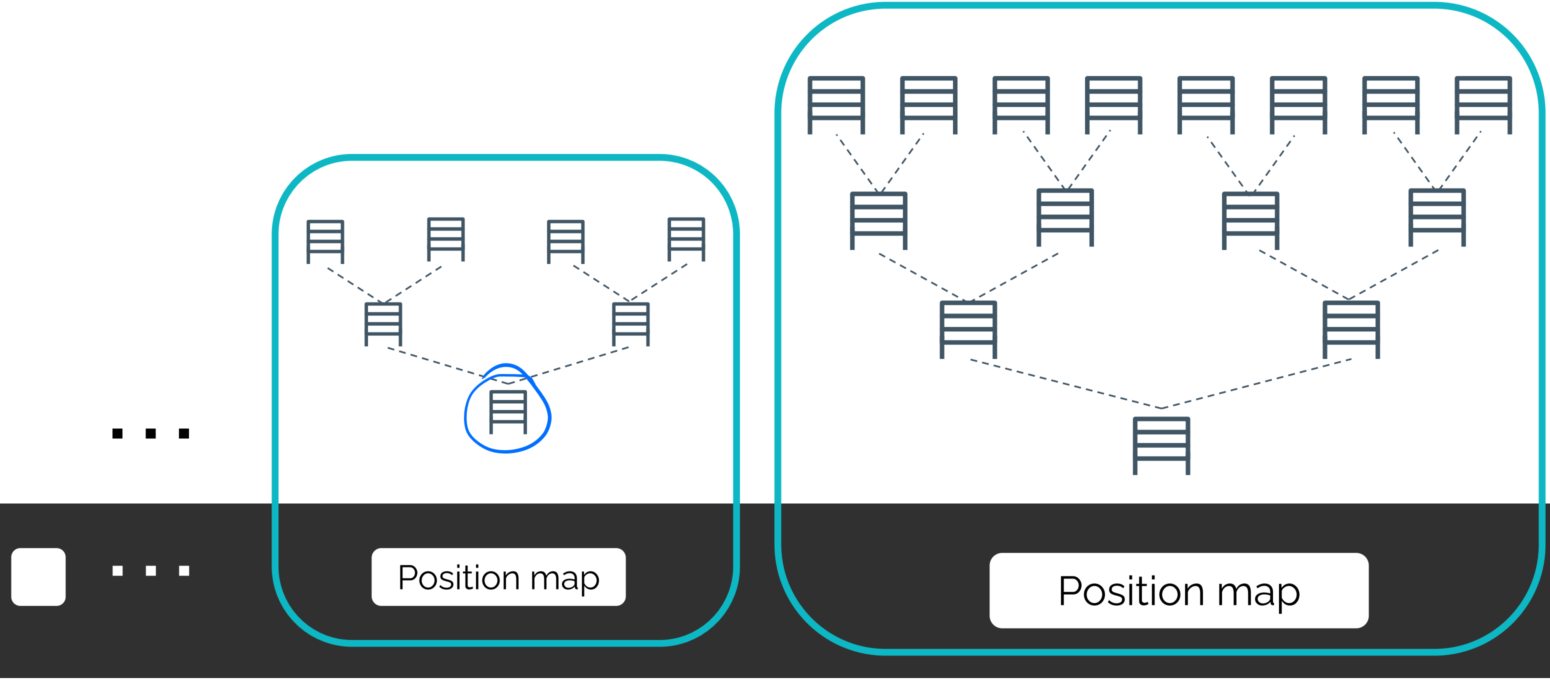
Position map

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

block size > 10logN

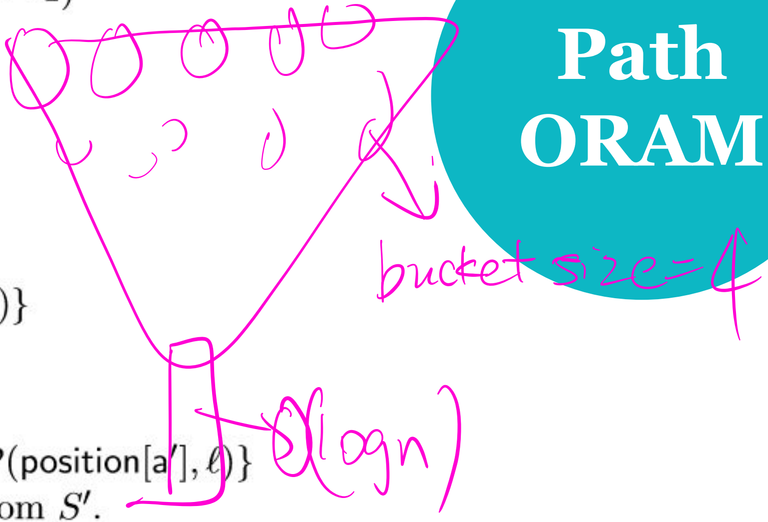


Cost with eviction: $O(\log^3 n)$



Path ORAM

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



Achieves O(log² n) cost with recursion

Perform eviction only on the read path

The **most aggressive** eviction algorithm

- Pack blocks as close to leaves as possible subject to path invariant

Blocks that cannot be evicted in **stash**

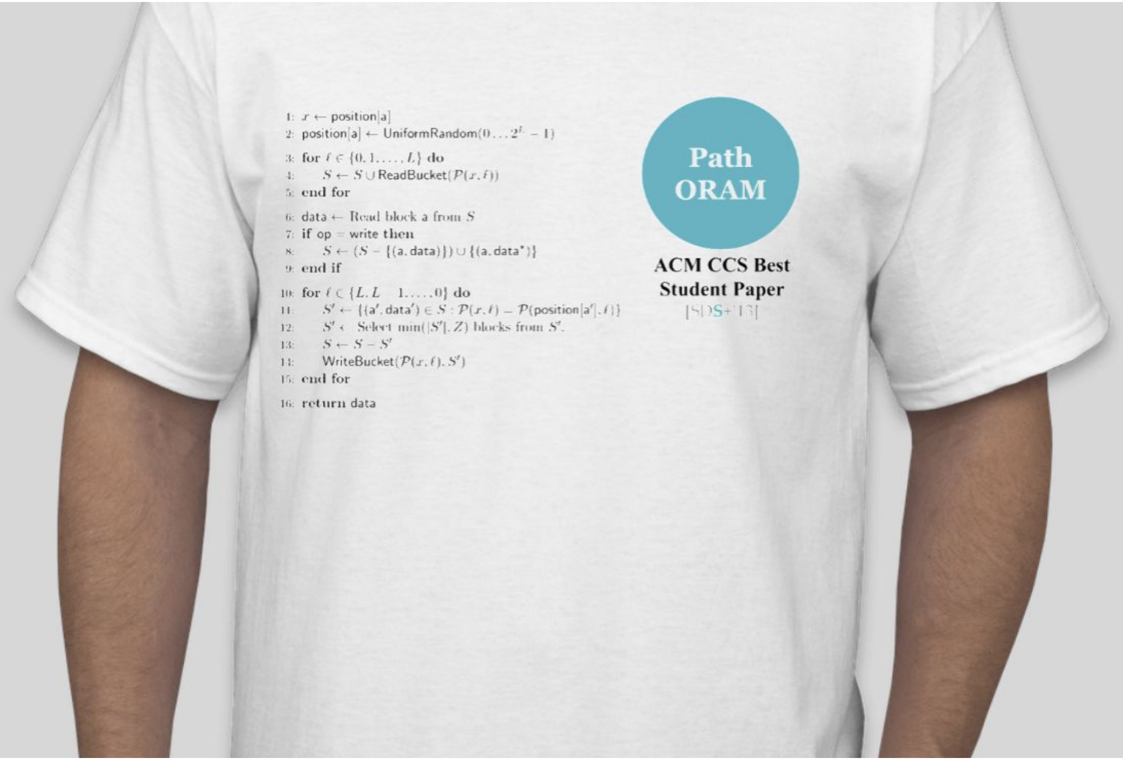
Path ORAM

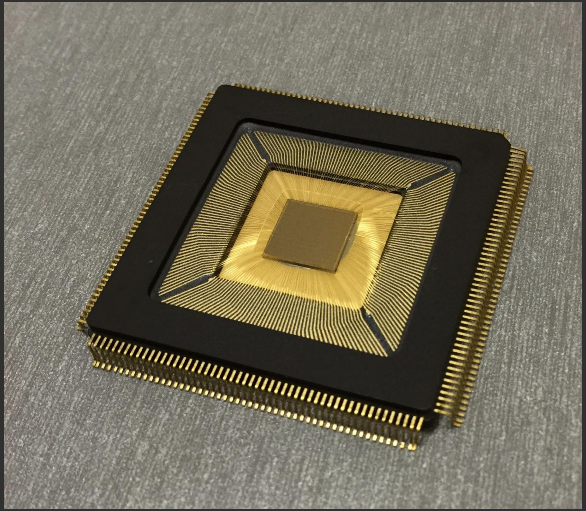
$x \leftarrow$ position[a]
position[a] $\leftarrow$ UniformRandom$(0 \ldots 2^L - 1)$
for $\ell \in \{0, 1, \ldots, L\}$ **do**
    $S \leftarrow S \cup$ ReadBucket$(\mathcal{P}(x, \ell))$
end for
data $\leftarrow$ Read block a from S
if op = write **then**
    $S \leftarrow (S - \{(\mathsf{a}, \mathsf{data})\})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

Path ORAM

ACM CCS Best Student Paper
[SDS+'13]

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

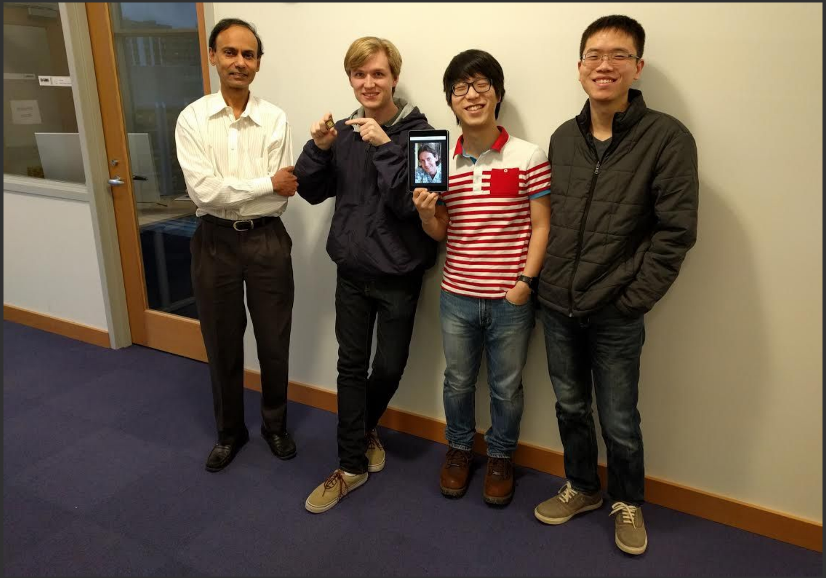

x ← position[a]
position[a] ← UniformRandom(0 … 2^L − 1)
for ℓ ∈ {0, 1, …, L} do
S ← S ∪ ReadBucket(P(x, ℓ))
end for
data ← Read block a from S
if op = write then
S ← (S − {(a, data)}) ∪ {(a, data*)}
end if
for ℓ ∈ {L, L − 1, …, 0} do
S′ ← {(a′, data′) ∈ S : P(x, ℓ) = P(position[a′], ℓ)}
S′ ← Select min(|S′|, Z) blocks from S′.
S ← S − S′
WriteBucket(P(x, ℓ), S′)
end for
return data
Path ORAM
ACM CCS Best Student Paper
[SDS+13]



~2X average overhead on standard benchmarks

(ORAM not always on)

"Ascend" processor running a variant of our ORAM



Thank you!