Techniques for Practical ORAM and ORAM in Hardware

Ling Ren

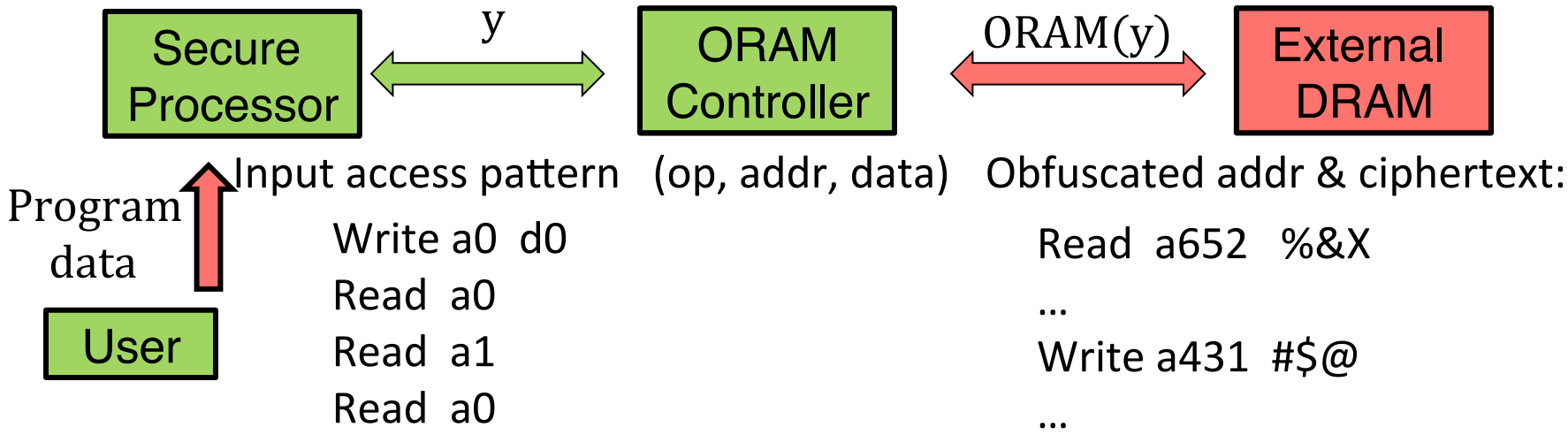
Joint work with

Chris Fletcher, Albert Kwon, Marten van Dijk and Srini Devadas

Oblivious RAM

- **If |y| = |y'|, ORAM(y) and ORAM(y') indistinguishable**

- **Applications**
  - Encrypted computation using secure processor — This talk
  - Remote oblivious storage — Chris' talk
  - Secure RAM computation — Daniel's talk

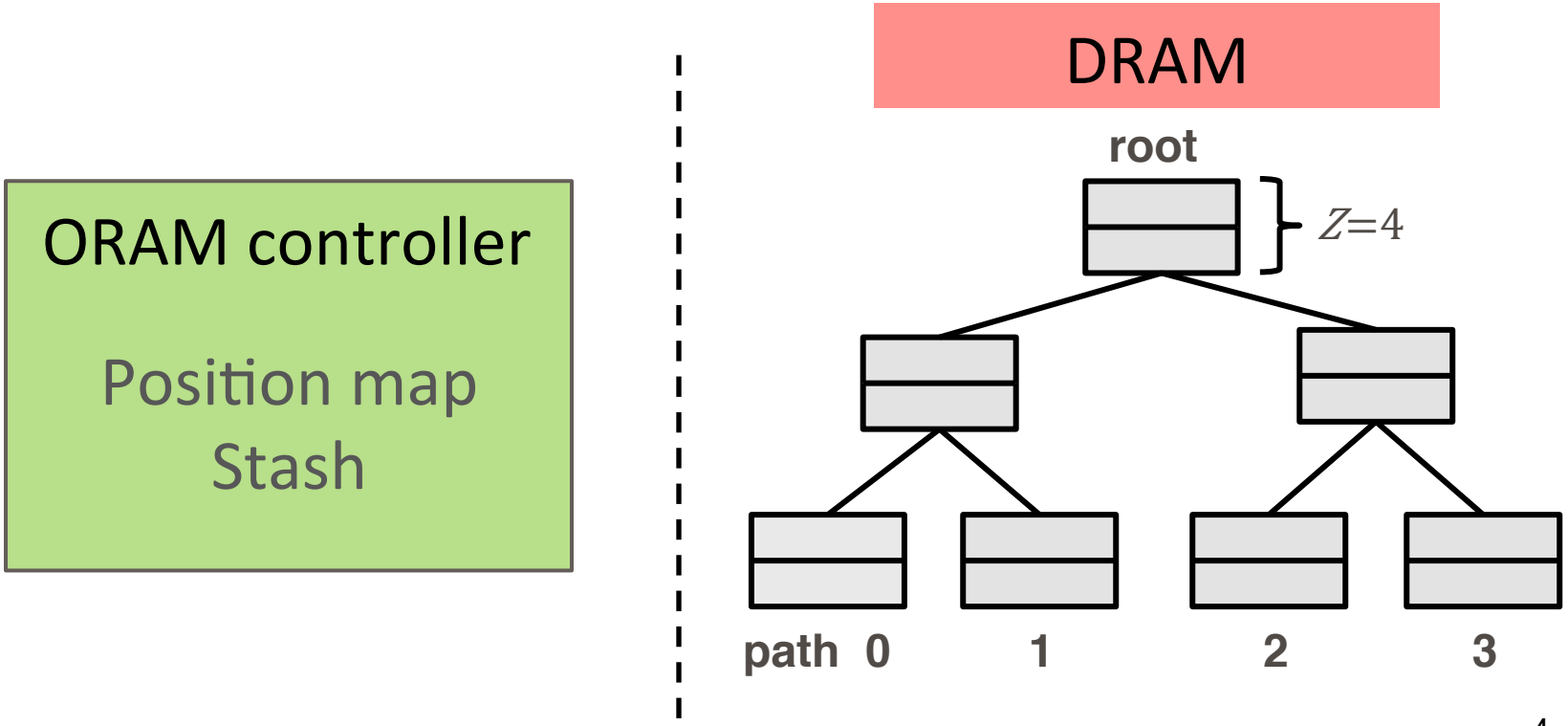


Outline

- Introduction to tree-based ORAMs
- Optimizing recursive ORAM
- Ring ORAM
- Hardware ORAM

Path ORAM

- **Efficient and simple**
- **External DRAM structured as a binary tree**



Path ORAM

- **Position Map: map each block to a random path**
- **Invariant: if a block is mapped to a path, it must be on that path or in the stash**
- **Stash: temporarily hold some blocks**

ORAM controller

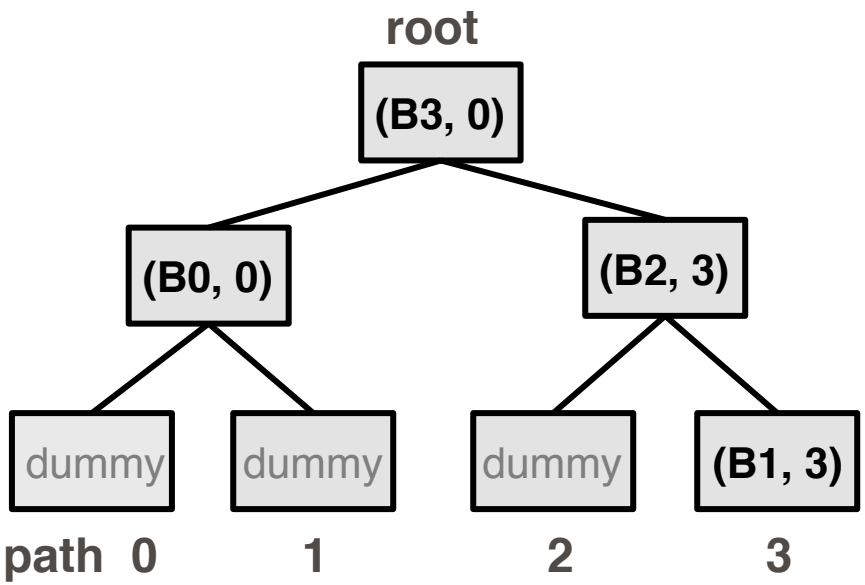
Stash

			(B4, 1)

Position Map

Block	Path
B0	0
B1	3
B2	3
B3	0
B4	1

DRAM



Path ORAM Operation

- **Access Block 1**
  - Read all blocks on path 3
  - Remap B1 to a new random path
  - Write as many blocks as possible back to path 3 (keep the invariant)

ORAM controller

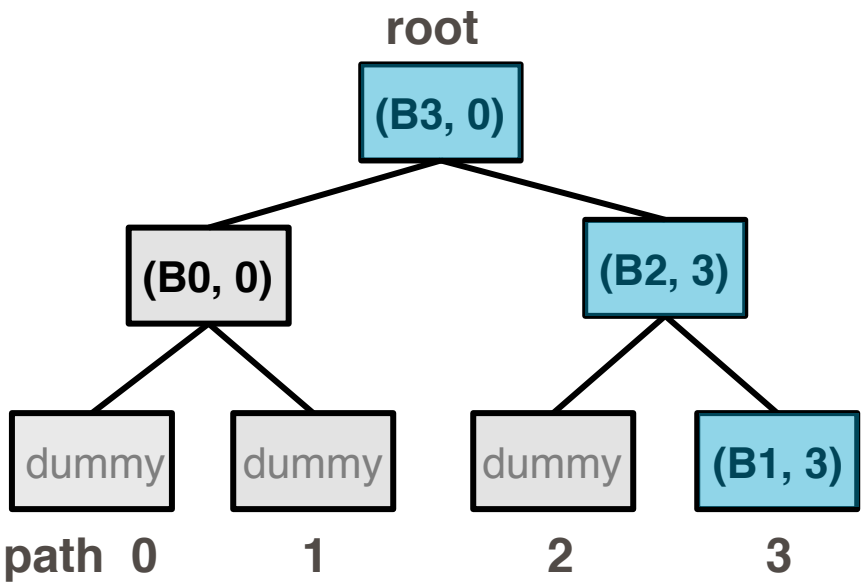
Stash

			dummy
		(B1, 1)	(B4, 1)

Position Map

Block	Path
B0	0
B1	~~3~~ 1
B2	3
B3	0
B4	1

DRAM



Path ORAM Analysis

- **Bandwidth $O(ZL) = O(Z \log N)$**
- **|Stash| $= O(\log N)$ for $Z \geq 4$**
- **Security: a random path is accessed**
- **Other tree-based ORAMs differ in eviction**

ORAM controller

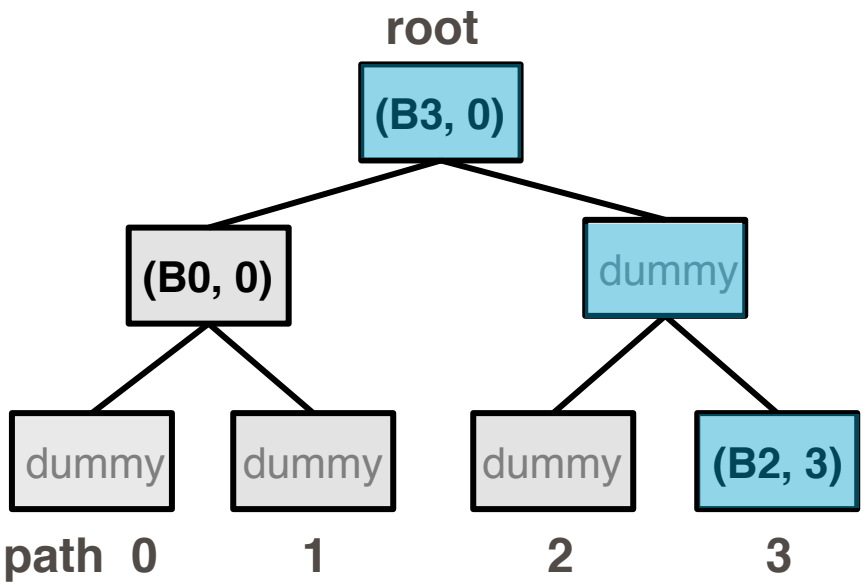
Stash

		(B1, 1)	(B4, 1)

Position Map

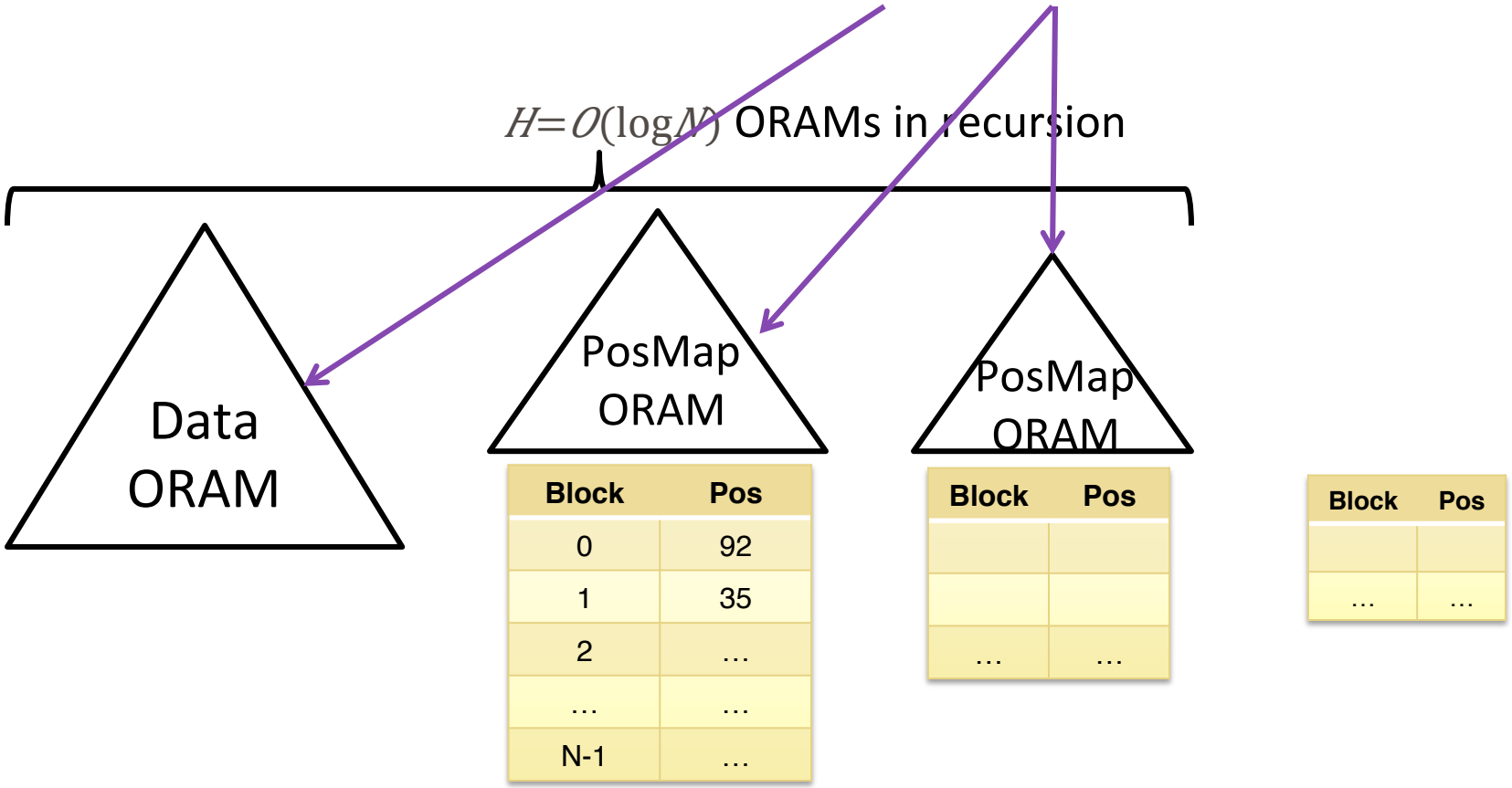
Block	Path
B0	0
B1	~~X~~ 1
B2	3
B3	0
B4	1

DRAM



Recursive ORAM

- **Position map is too large**
- **Bandwidth after recursion** $O(\log N + \log^3 N/B)$



Outline

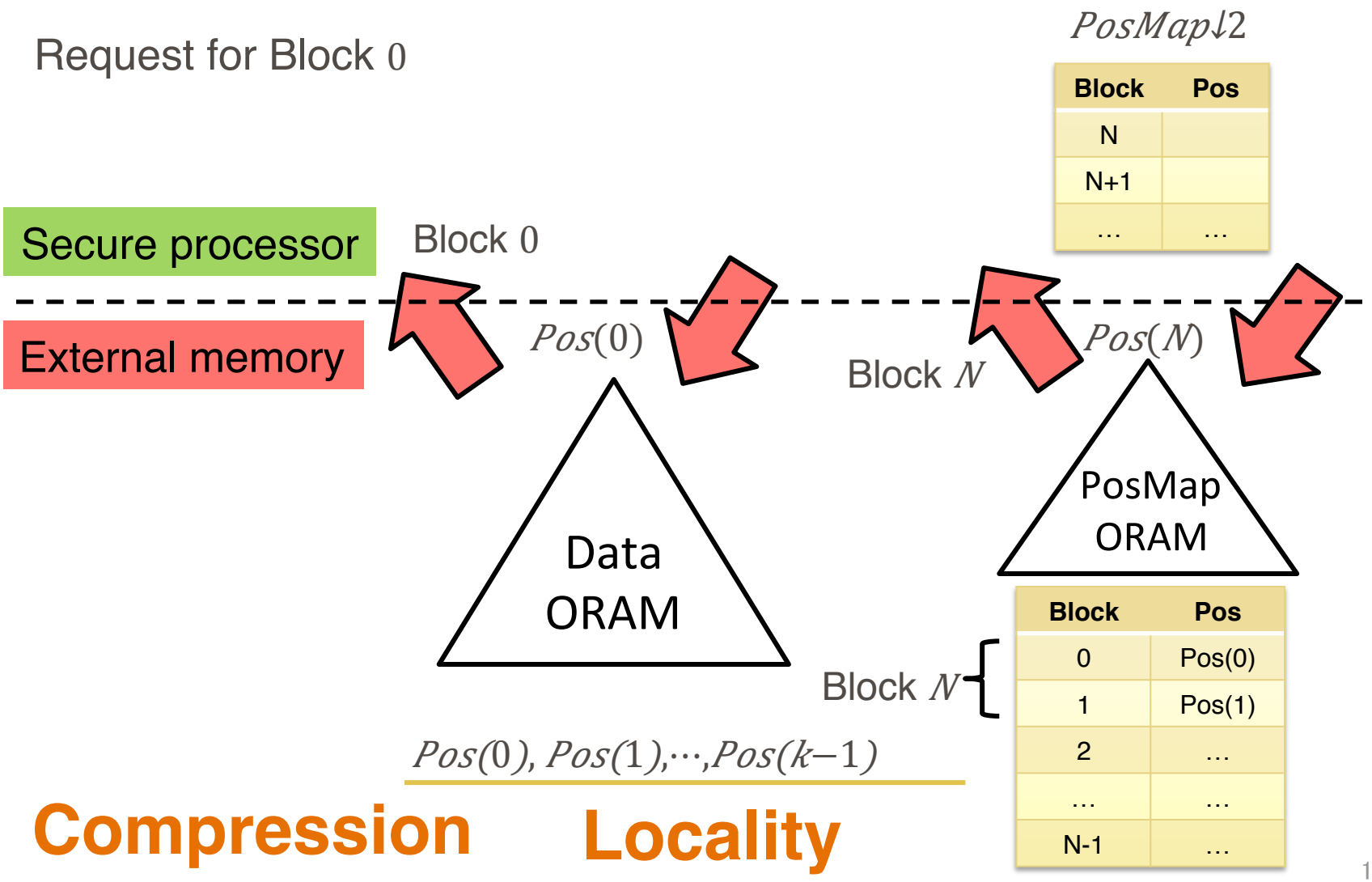
- Introduction to tree-based ORAMs

- **Optimizing recursive ORAM**

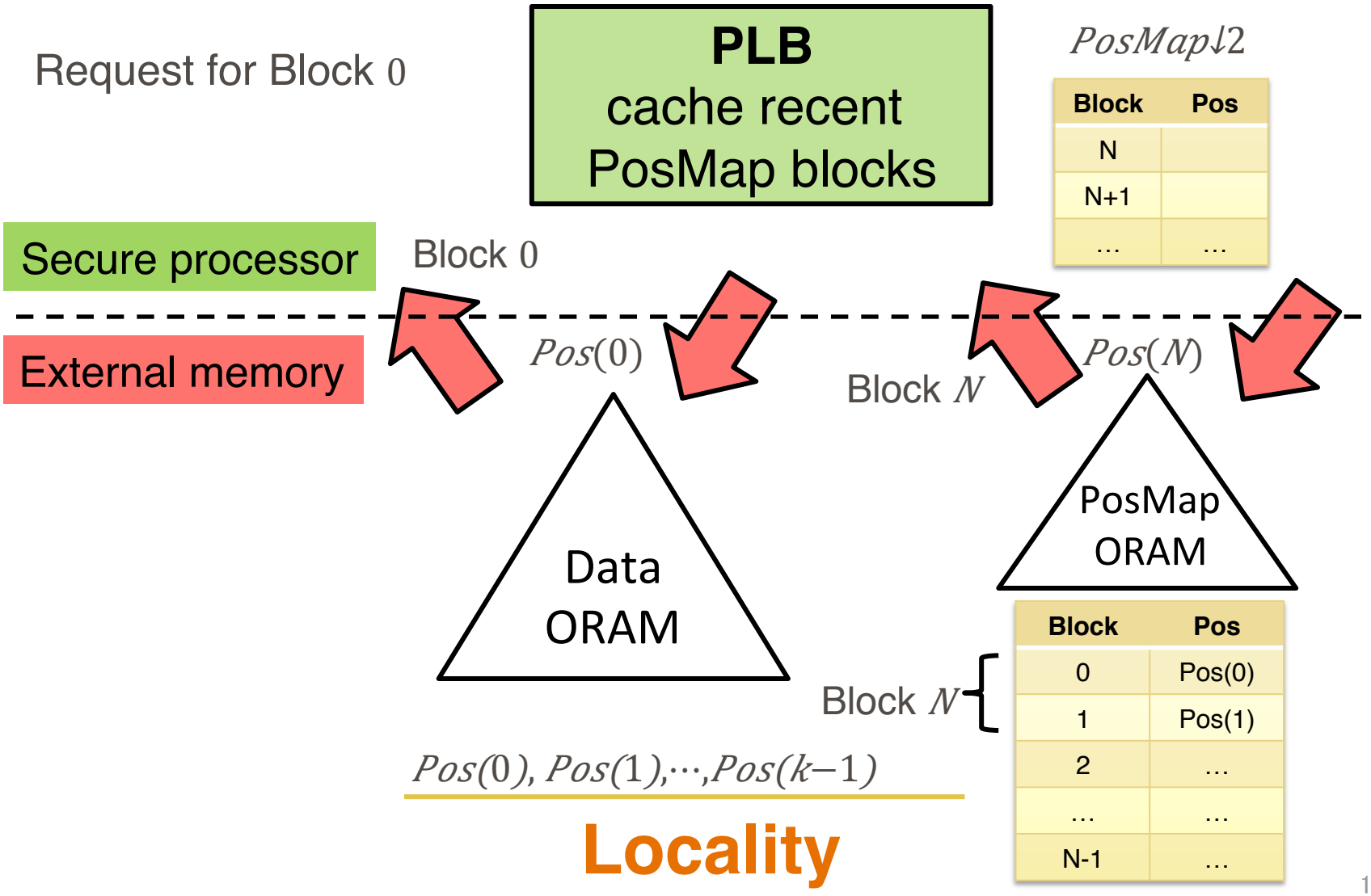
- **Ring ORAM**

- **Hardware ORAM**

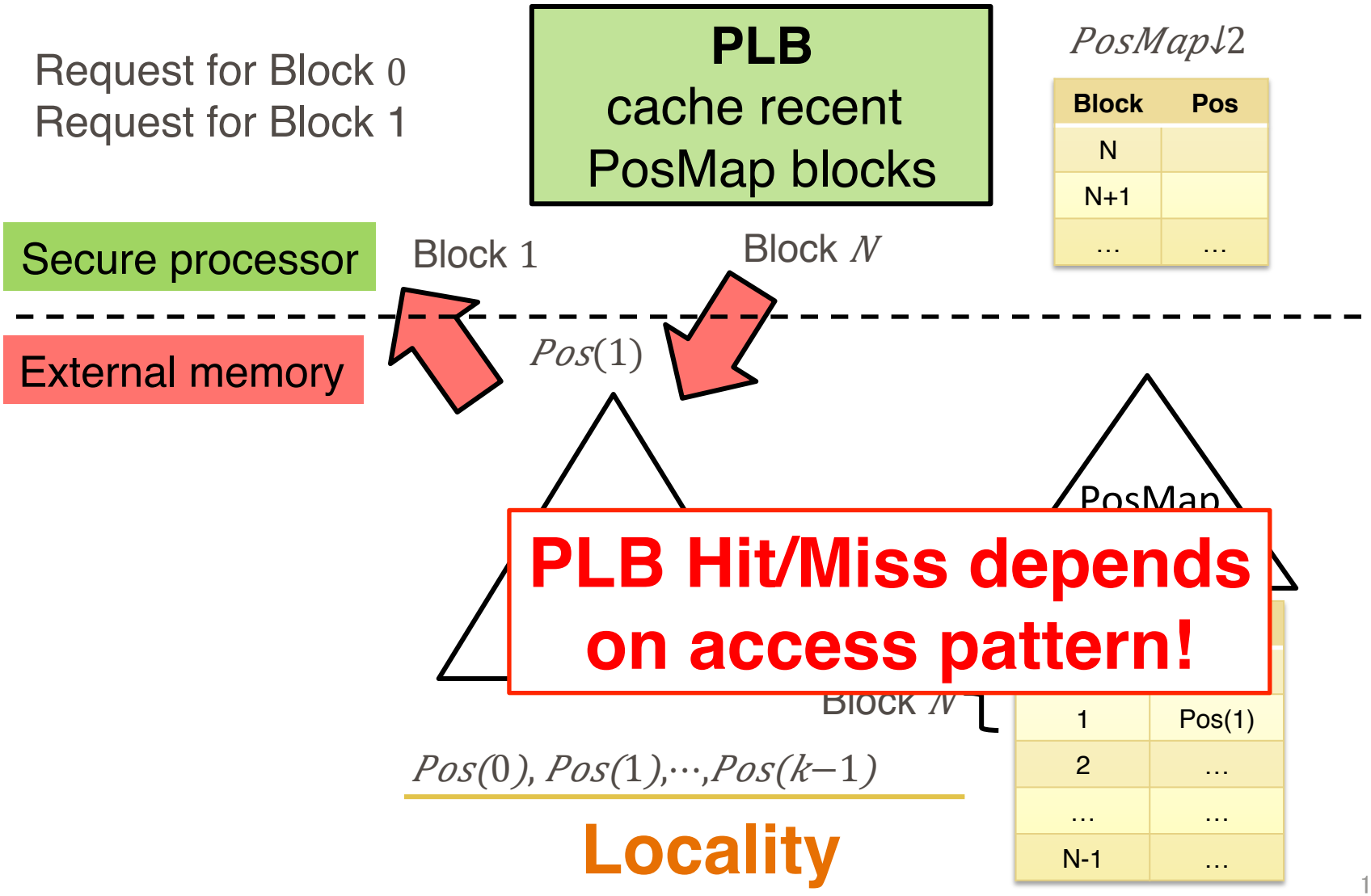
Recursive ORAM Access



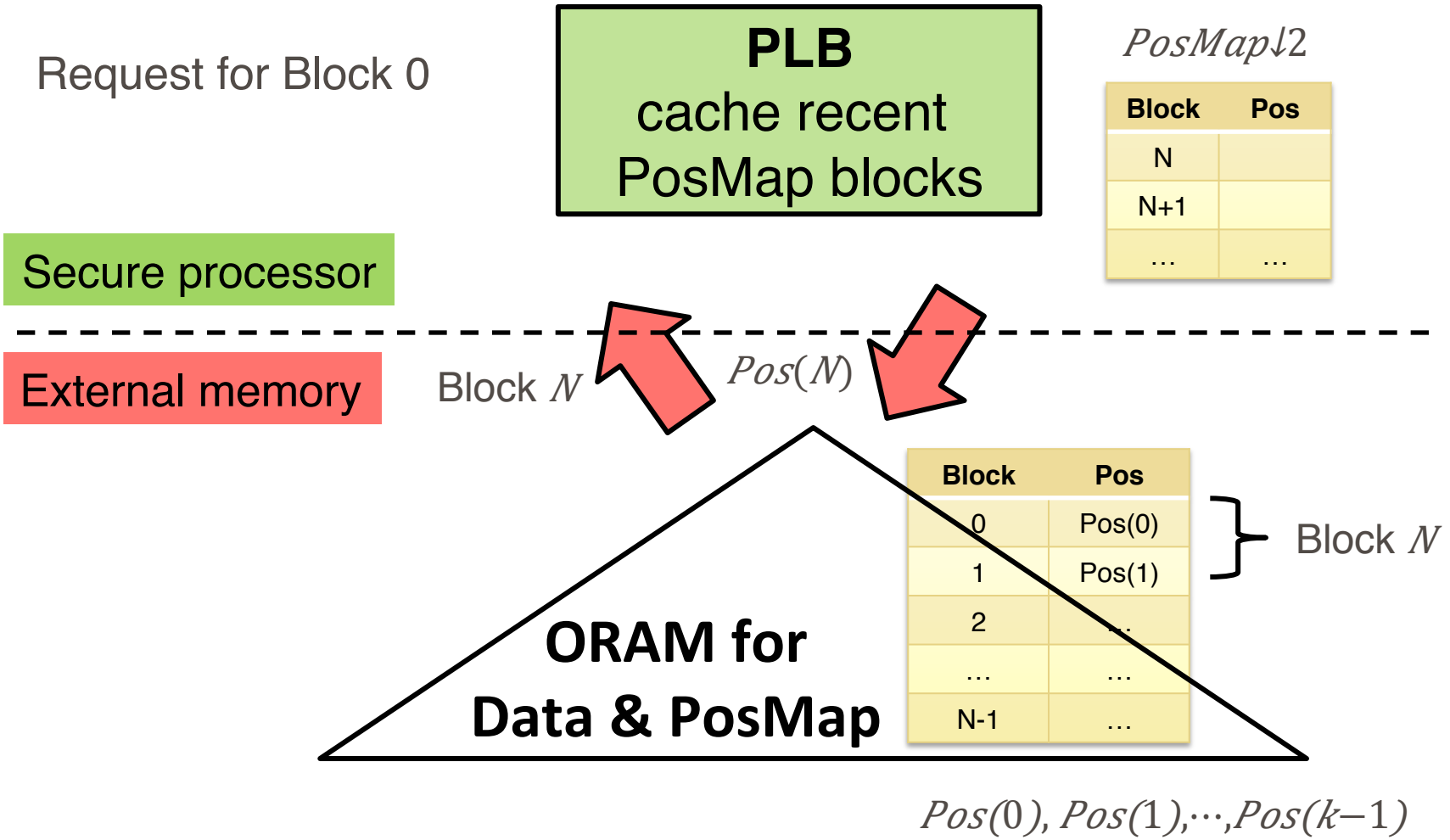
PLB: Cache PosMap



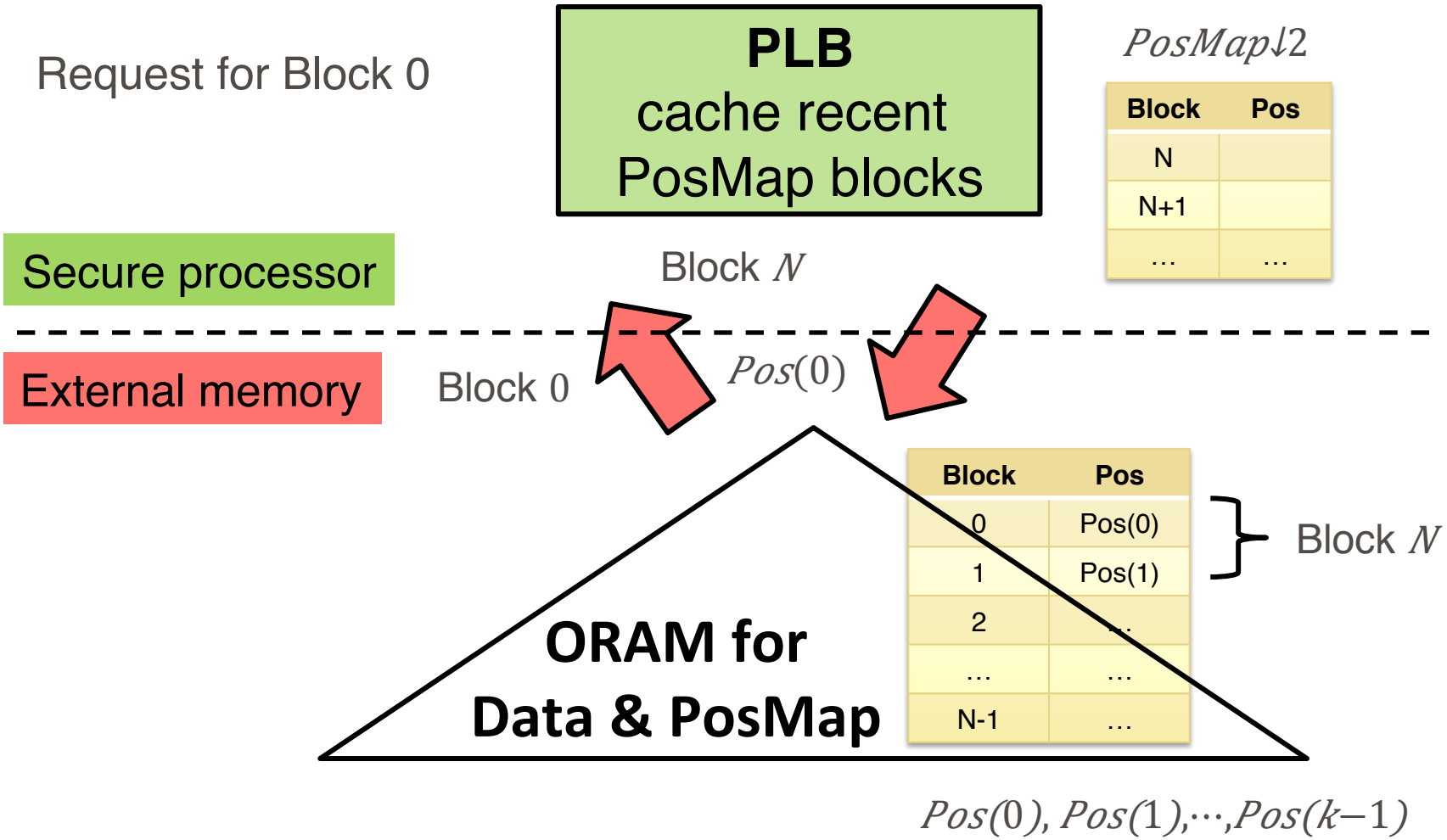
PLB: Cache PosMap



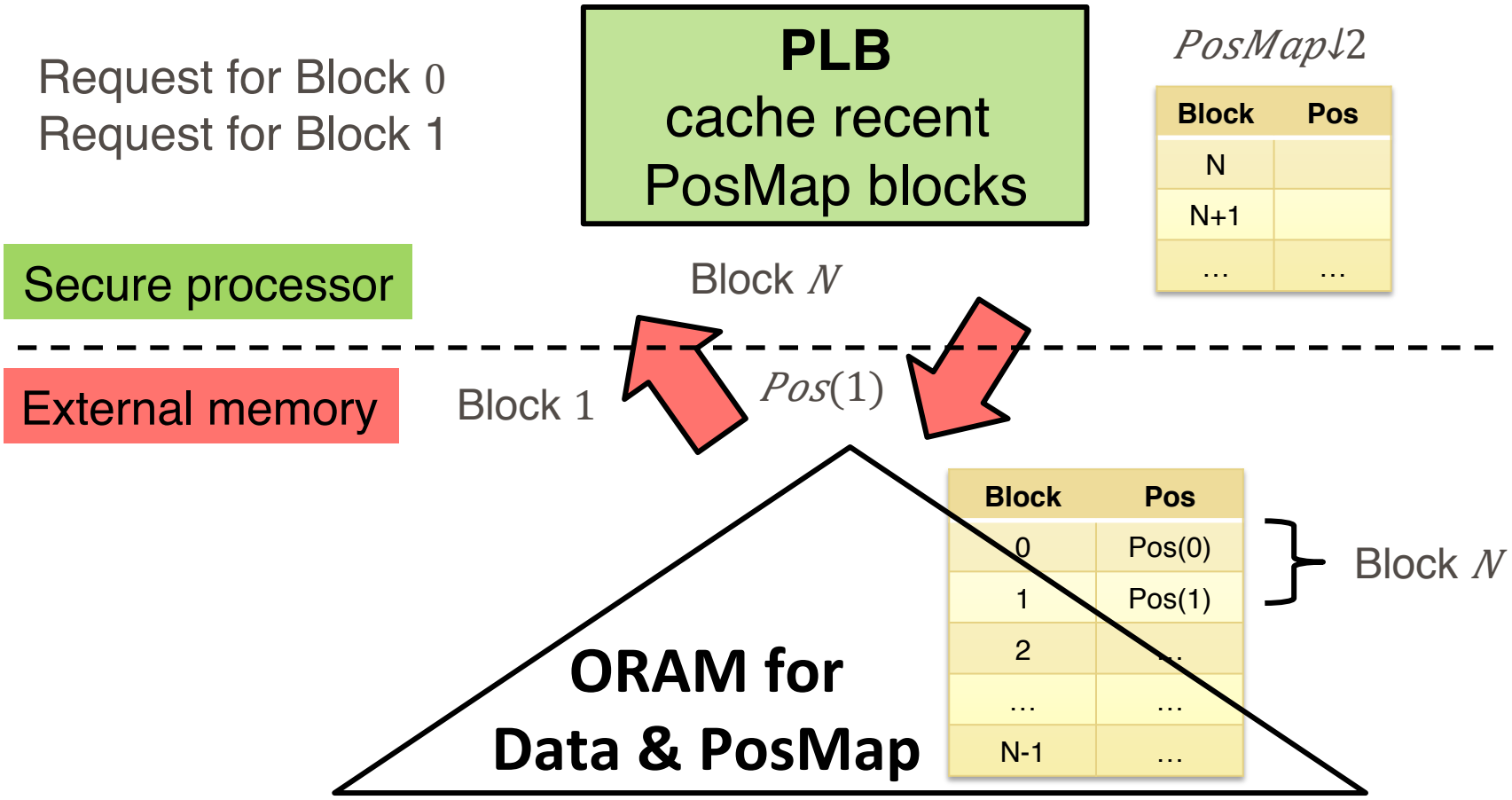
Unified ORAM



Unified ORAM



Unified ORAM



If |ORAM(y)| = |ORAM(y')| ?? Worth discussion $(1),\cdots,Pos(k-1)$

~~If |y| = |y'|,~~ ORAM(y) and ORAM(y') indistinguishable

PosMap compression

- **PosMap: block → random leaf ($\log N$ bits)**
  - A table of random numbers

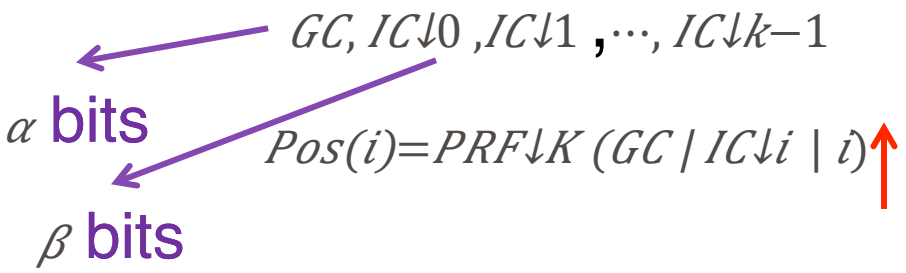
Block	Pos
0	92
1	35
2	…
…	…
N-1	…

- **Block → monotonic counter → pseudorandom leaf**
  - |counter| > $\log N$
  - Reduce |counter| → counter overflows

Replacing true randomness with pseudorandomness improves efficiency

PosMap Compression

- **Let a group of blocks share a big counter**



$PRF\downarrow K\,(GC \mid IC\downarrow j \mid j) \rightarrow PRF\downarrow K\,(GC+1 \mid 0 \mid j)$

$k/2\uparrow\beta$ small α/k

$+\beta<\log N$

= remap in an ORAM access!

$\alpha=64,\ k=32,\ \beta=14$

$k/2\uparrow\beta=0.2\%$

$\alpha/k+\beta=16<26=\log N$

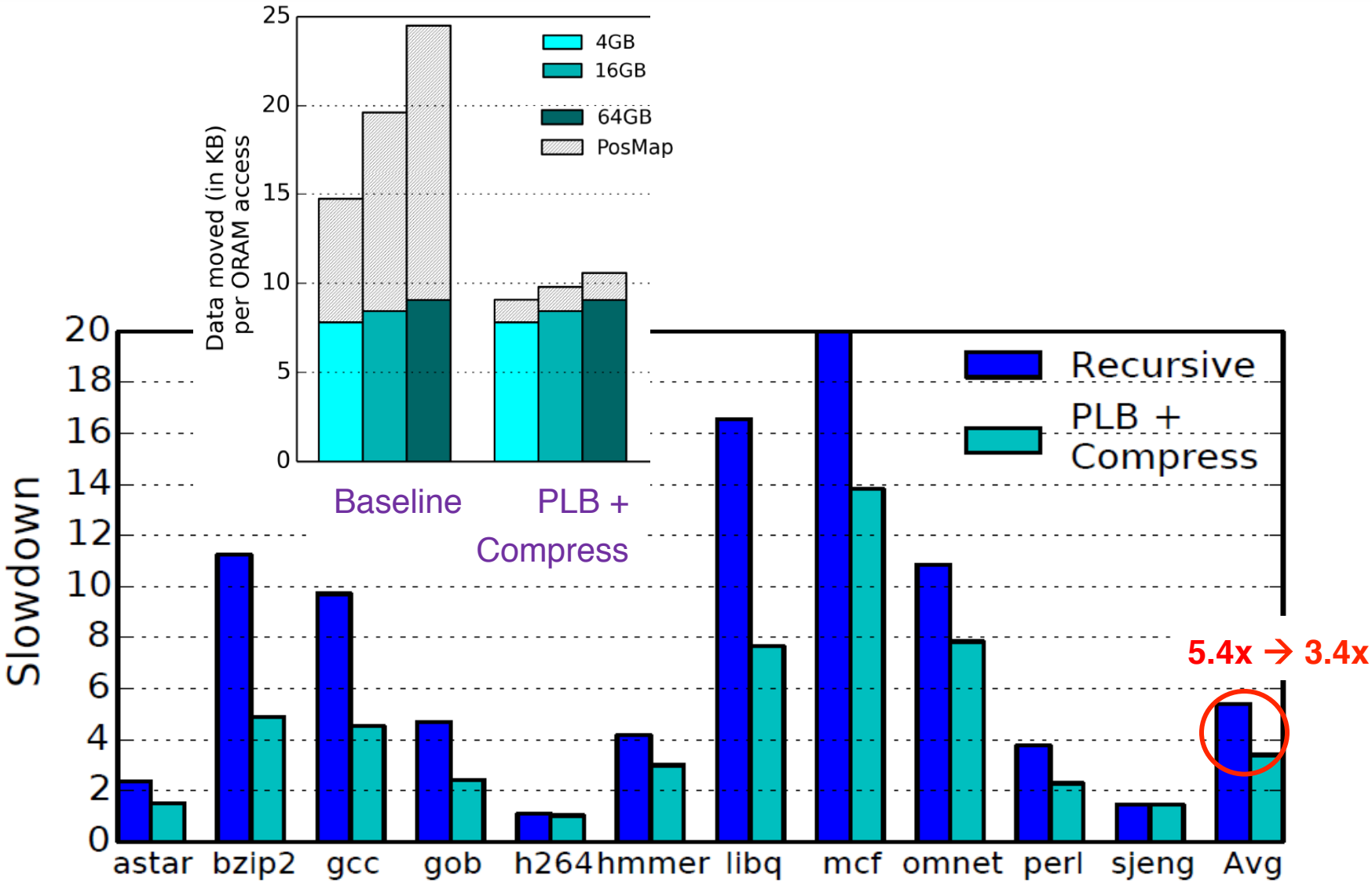
Asymptotic Improvement

$GC, IC\downarrow 0, IC\downarrow 1, \cdots, IC\downarrow k-1$

- $\beta=\log\log N,\ k=\log N/\log\log N,\ \alpha=\Theta(\log N)^{\alpha}$ bits — β bits
- $k/2\uparrow\beta = o(1),\qquad \alpha/k+\beta=\log\log N<\log N$

Algorithm	Asymptotic bandwidth	$B=\Theta(\log N)$
Recursive Path	$O(\log N+\log\uparrow 3\ N/B)$	$O(\log\uparrow 2\ N)$
+ Compression	$O(\log N+\log\uparrow 3\ N/B\log\log N)$	$O(\log\uparrow 2\ N/\log\log N)$

Simulation Results

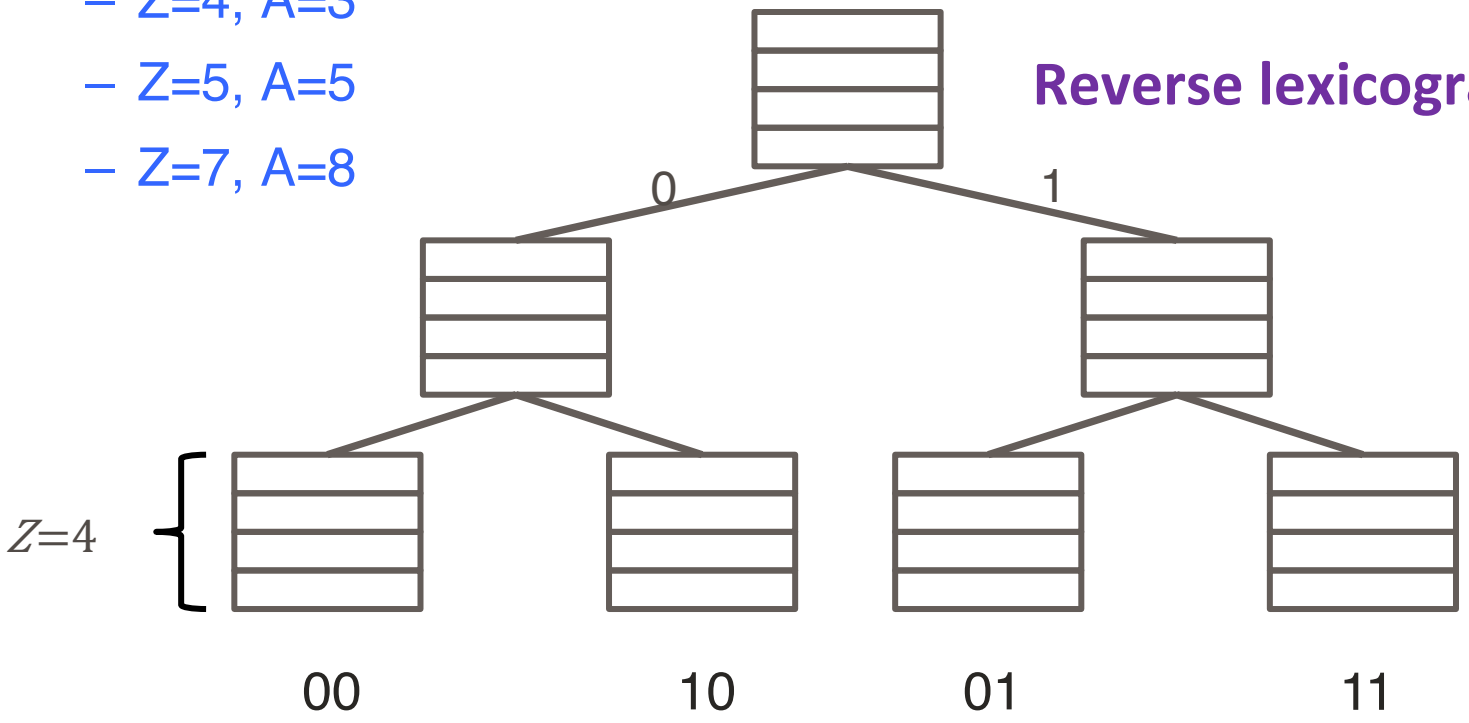


Outline

- Introduction to tree-based ORAMs
- Optimizing recursive ORAM
- **Ring ORAM**
- **Hardware ORAM**

Improve Eviction Quality

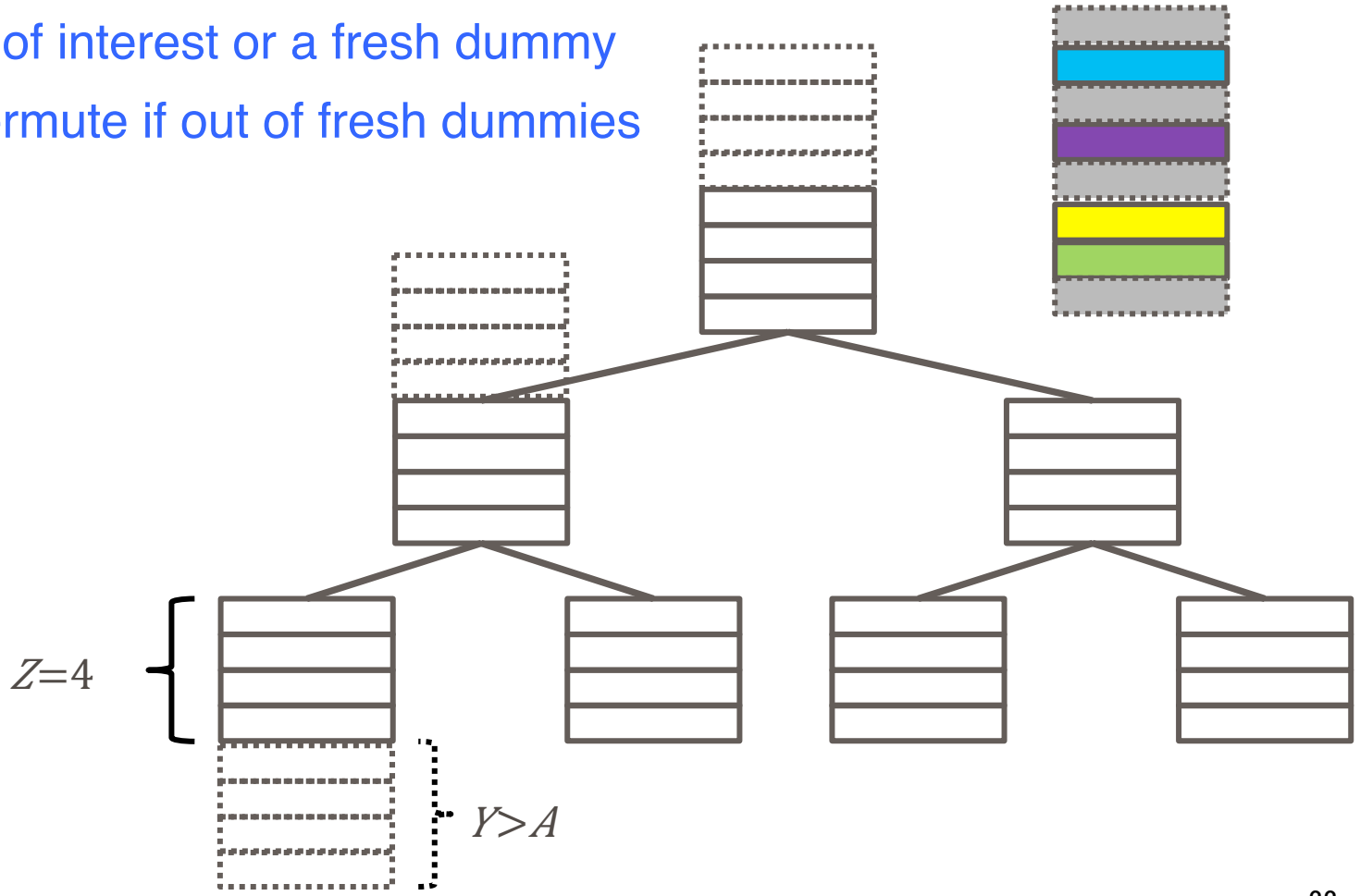
- **Path ORAM: $8\log N$ evict to the [random] path being read**
- **Load balance between paths**
- **Allow less frequent evictions (1 per A accesses)**
  - Z=4, A=3
  - Z=5, A=5
  - Z=7, A=8



Craig Gentry, Kenny Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova and Daniel Wichs. Optimizing ORAM and Using it Efficiently for Secure Computation

Permuted buckets

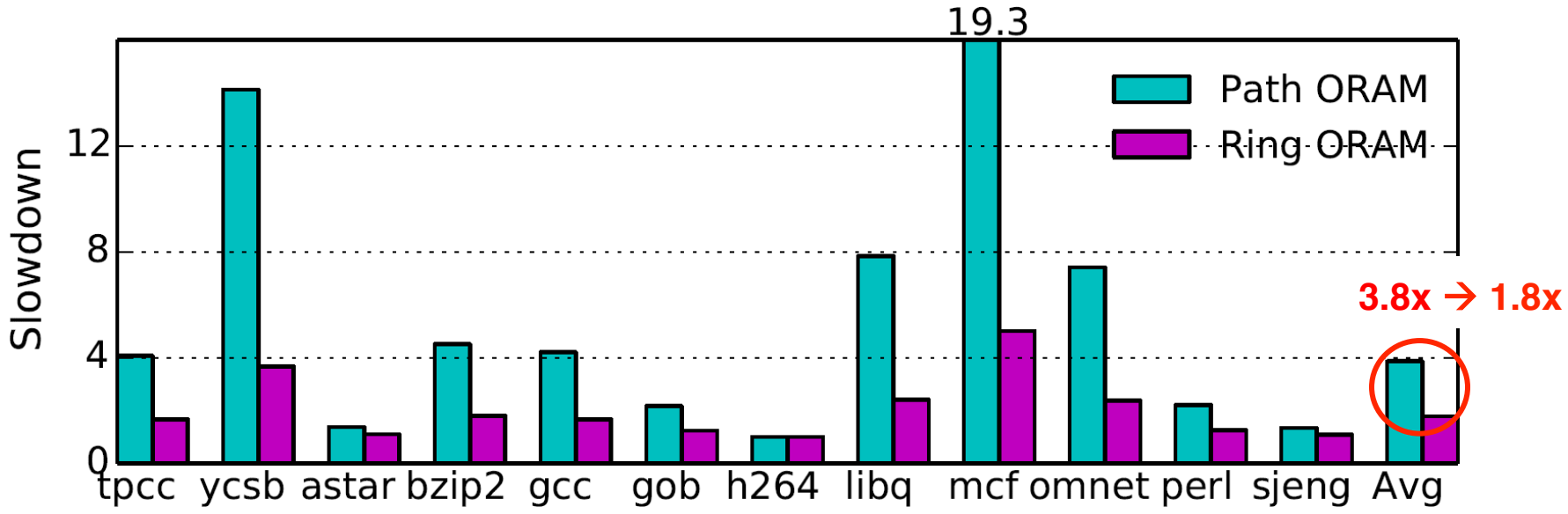
- **Goal: read only 1 block per bucket**
  - Add Y reserved dummy slots, and permute buckets
  - Block of interest or a fresh dummy
  - Re-permute if out of fresh dummies



Ring ORAM Performance

- **Eviction overhead (Z + Z + Y) / A per bucket**

Scheme and Parameters	Read	Eviction	Total
Path ORAM $Z = 4$	$4L$	$4L$	$8L$
Ring ORAM Z=16,A=22,Y=28	$\boldsymbol{L}$	$\boldsymbol{2.8L}$	$\boldsymbol{3.8L}$
Ring ORAM Z=5, A=5,Y=7	$\boldsymbol{L}$	$\boldsymbol{3.8L}$	$\boldsymbol{4.8L}$



Outline

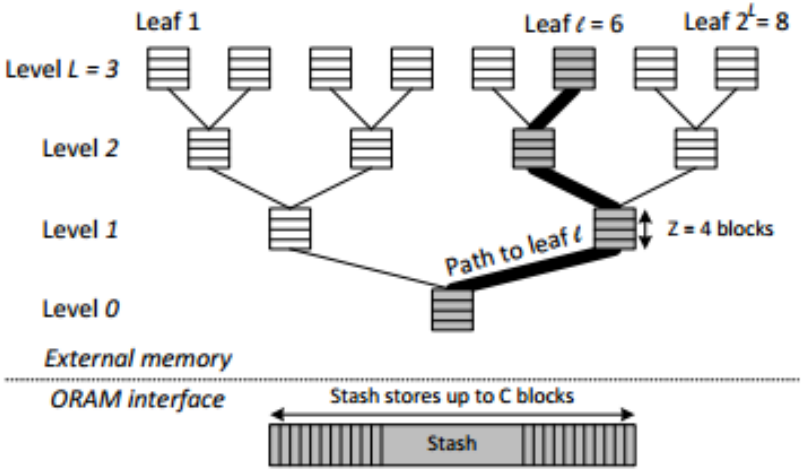
- Introduction to tree-based ORAMs
- Optimizing recursive ORAM
- Ring ORAM
- **Hardware ORAM**

Hardware ORAM

- **Things we take for granted in ORAM algorithms**
  - **Computation is cheap (e.g. ORAM eviction, hashing)**
  - **RAM has uniform latency**

Stash Management

- **Computation is cheap?**



function PUSHBACK(l, l')

$t_1 \leftarrow (l \oplus l')||0$ ▷ Bitwise XOR

$t_2 \leftarrow t_1 \;\&\; -t_1$ ▷ Bitwise AND, 2C negation

$t_3 \leftarrow t_2 - 1$ ▷ 2C subtraction

$\text{full} \leftarrow \{(\text{Occupied}[i] \stackrel{?}{=} Z) \text{ for } i = 0 \text{ to } L\}$

$t_4 \leftarrow t_3 \;\&\; \sim\text{full}$ ▷ Bitwise AND/negation

$t_5 \leftarrow \text{reverse}(t_4)$ ▷ Bitwise reverse

$t_6 \leftarrow t_5 \;\&\; -t_5$

$t_7 \leftarrow \text{reverse}(t_6)$

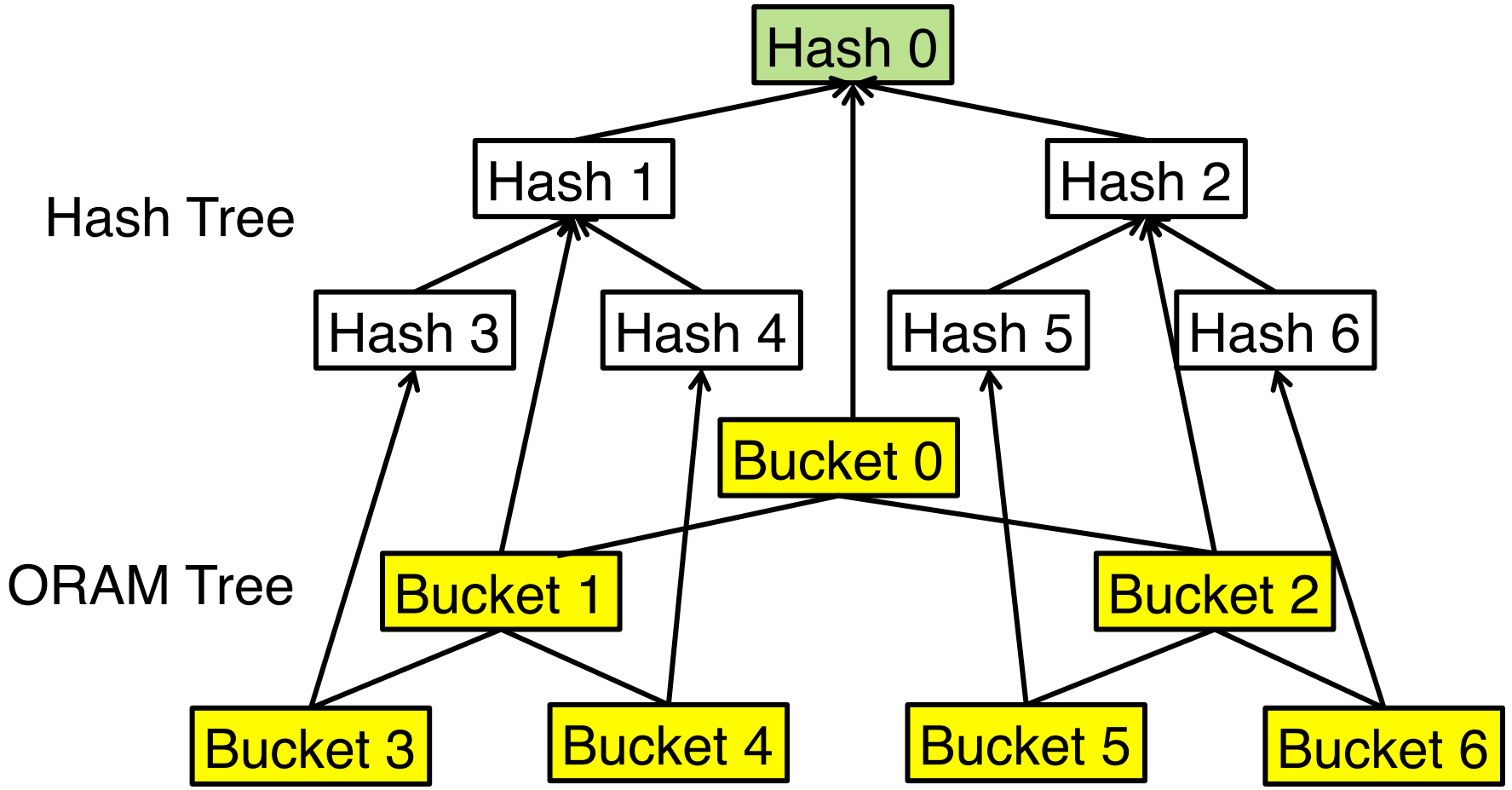
if $t_7 \stackrel{?}{=} 0$ **then**

  return -1 ▷ Block is stuck in stash

return $\log_2(t_7)$ ▷ Note: t_7 must be one-hot

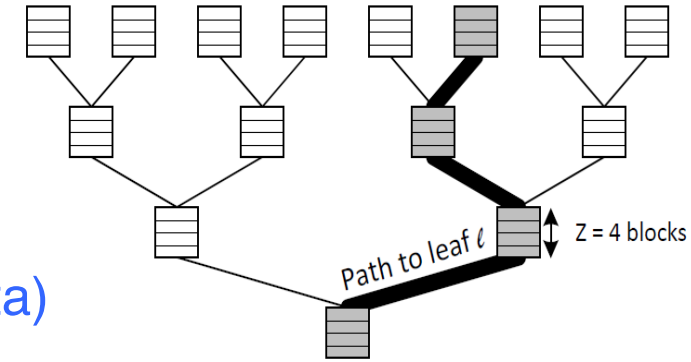
ORAM Authenticity

- **Just use hash tree?**
  - A serialization problem. Hashing becomes the bottleneck



PMMAC for Authenticity

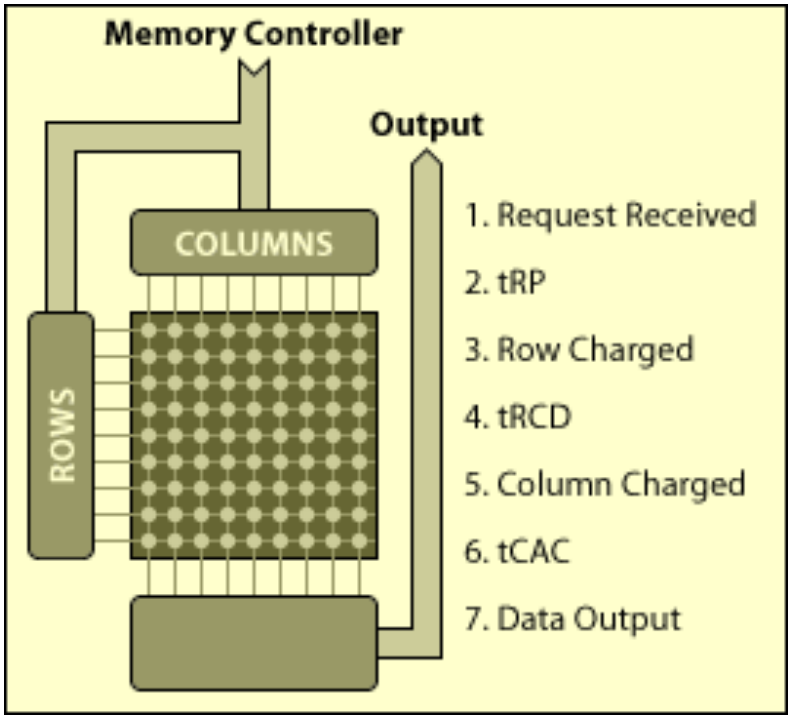
- **Construction**
  - Block → monotonic counter
  - MAC(Block address || counter || block data)

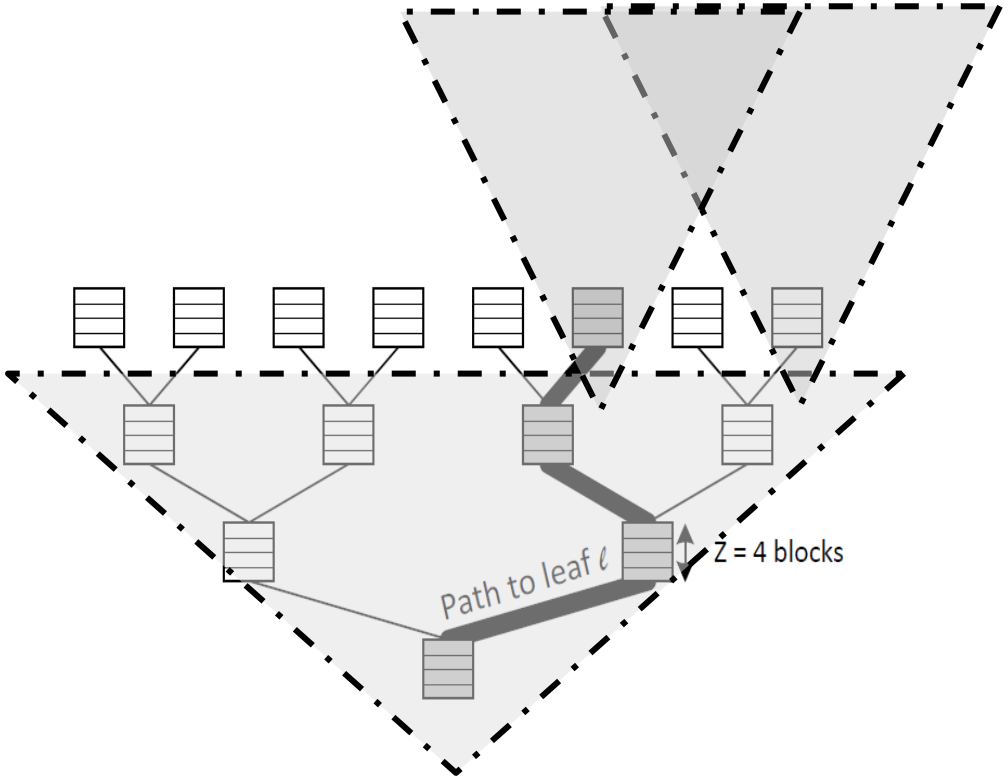


- **Problem: privacy under malicious server**
  - Is it fixable? Bounding the leakage?

DRAM Timing

- An access to a new row is slow
- Very bad for trees
- Solution: subtree layout





GIF from http://www.pcmag.com/article2/0,2817,1153175,00.asp

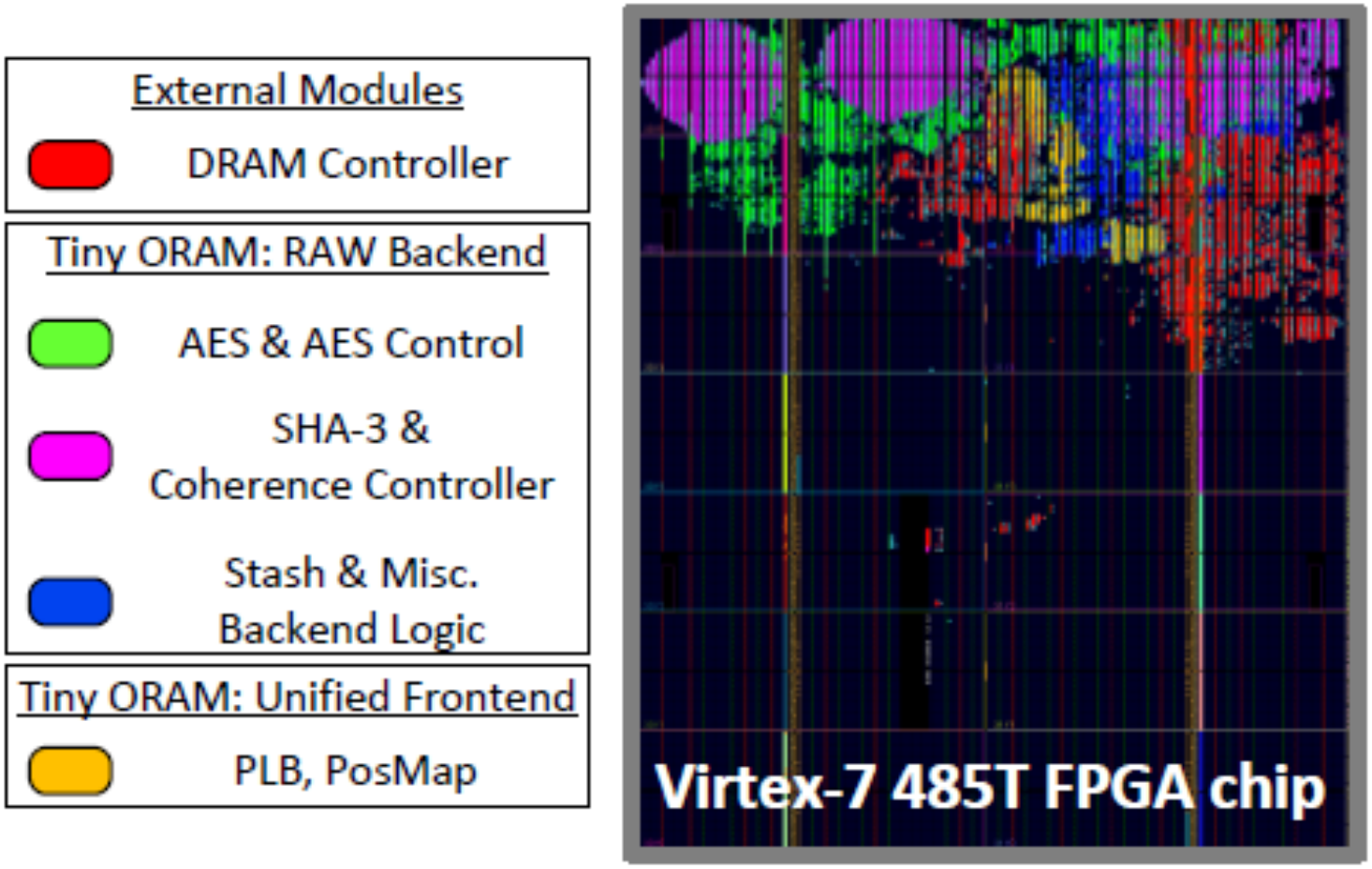
Our Hardware ORAM

- **Techniques we have implemented**
  - PLB + Unified ORAM tree
  - Load balance eviction order (1/A eviction frequency)
  - Efficient stash management
  - PMMAC
  - Subtree layout

- **Techniques we have not implemented**
  - Compressed counter, permuted buckets

FPGA Die Photo

- **Area: 6% logic, 14% memory of a mid-range FPGA**
- **Performance: 250 cycles (200 MHz clock) = 1.25 us**



Take-aways

- **Recursion accounts for about 50% overhead; PLB (+ Compression) makes it almost free**
- **Ring ORAM achieves another 2x improvement**
- **Design challenges in hardware ORAM: computation, hashing, DRAM locality**

Thank you! Questions?

Backup

What block size?

- **Data block size** B_d
- **PosMap block size** $B_p = \alpha + k\beta = \Theta(\log N)$

$\beta = \log\log N$, $k = \log N / \log\log N$, $\alpha = \Theta(\log N)$

- **Will use B_p (using B_d is suboptimal)**
  - Break a data block into $\lceil B_d / B_p \rceil$ sub-blocks of size B_p
  - Treat them as separate blocks in backend, with position being
  - $PRF_K(GC \| IC_j \| j \| t)$ where t is sub-block index
- $(\lceil B_d / B_p \rceil + \log N / \log k) O(\log^2 N) 1/B_d = O(\log N + \log^3 N / B_d \log\log N)$
  - $N \rightarrow N\lceil B_d / B_p \rceil$

Theoretical Construction

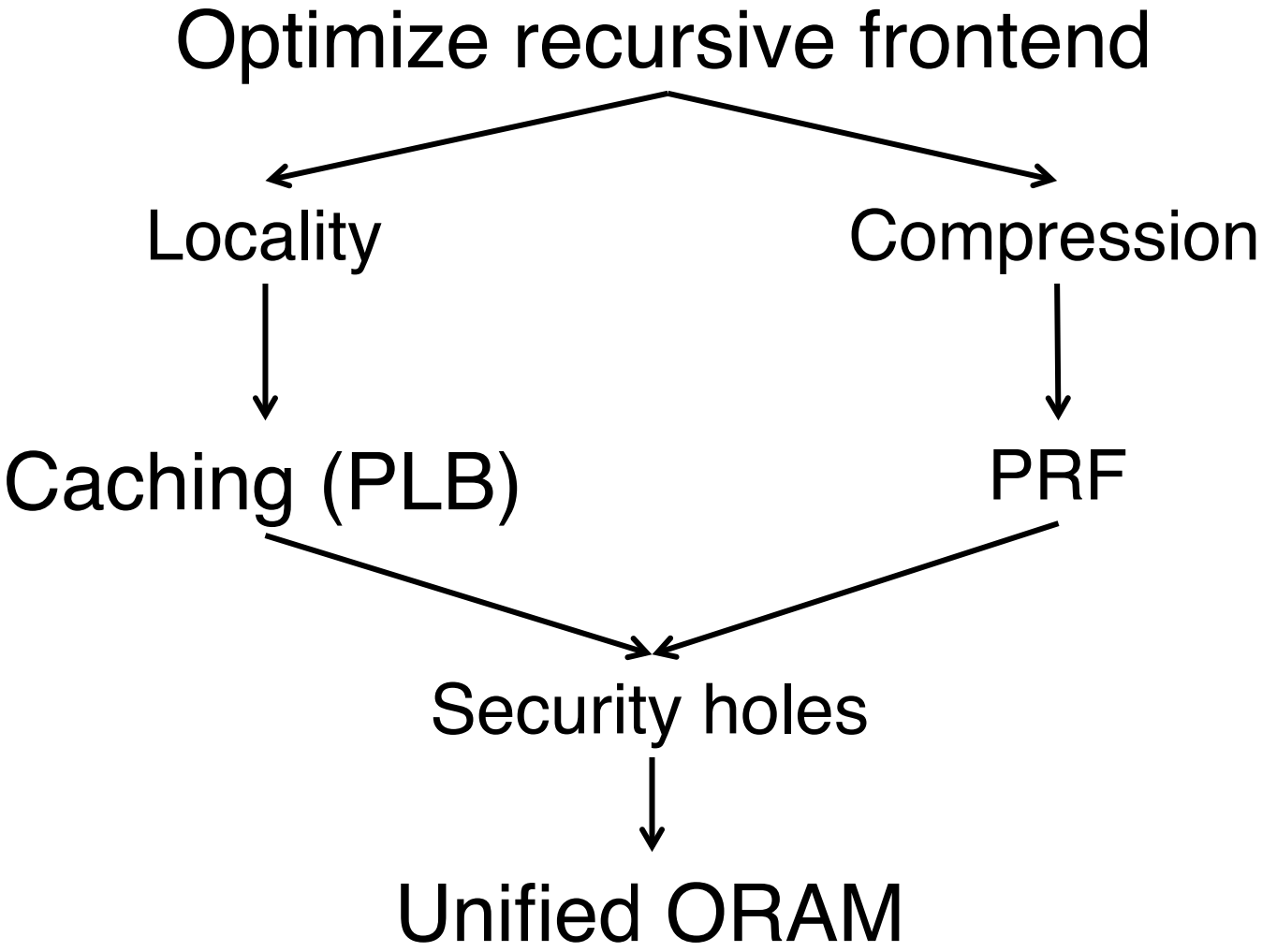
$GC, IC\downarrow 0, IC\downarrow 1, \cdots, IC\downarrow k-1$

α bits β bits

- $\beta = \log\log N,\ k = \log N / \log\log N,\ \alpha = \Theta(\log N)$
- $k/2\uparrow\beta = o(1),\qquad \alpha/k + \beta = \log\log N < \log N$

Algorithm	Asymptotic bandwidth
Recursive Path	$O(\log N + \log\uparrow 3\ N/B)$
+ Compression	$O(\log N + \log\uparrow 3\ N/B \log\log N)$

Outline


Optimize recursive frontend
Locality
Compression
Caching (PLB)
PRF
Security holes
Unified ORAM

Ring ORAM

- **Ring ORAM also performs well in oblivious storage**
  - Online bandwidth: $\mathcal{L}$ blocks →1 block

- **The XOR trick**
  - B, d1, d2, d3, …
  - E(B), E(d1), E(d2), E(d3) …
  - E(B, r), E(d1, r1), E(d2, r2), E(d3, r3) …
  - E(B, r) ⊕ E(d1, r1) ⊕ E(d2, r2) ⊕ E(d3, r3) ⊕ ...

  - A property not present in previous tree-based ORAMs

Computation is cheap?

Block size
= 512 bits

Platform			
FPGA			
ASIC			

DRAM bandwidth
100Gbits / second

AES/SHA BW
128 bits / cycle