



---

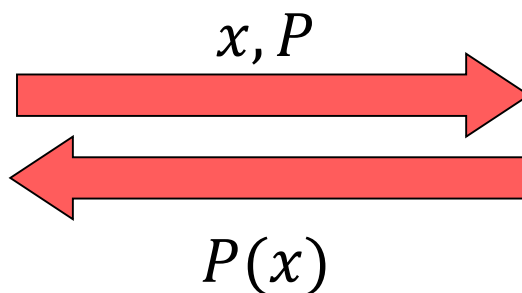
# Path Oblivious-RAM for Secure Processors

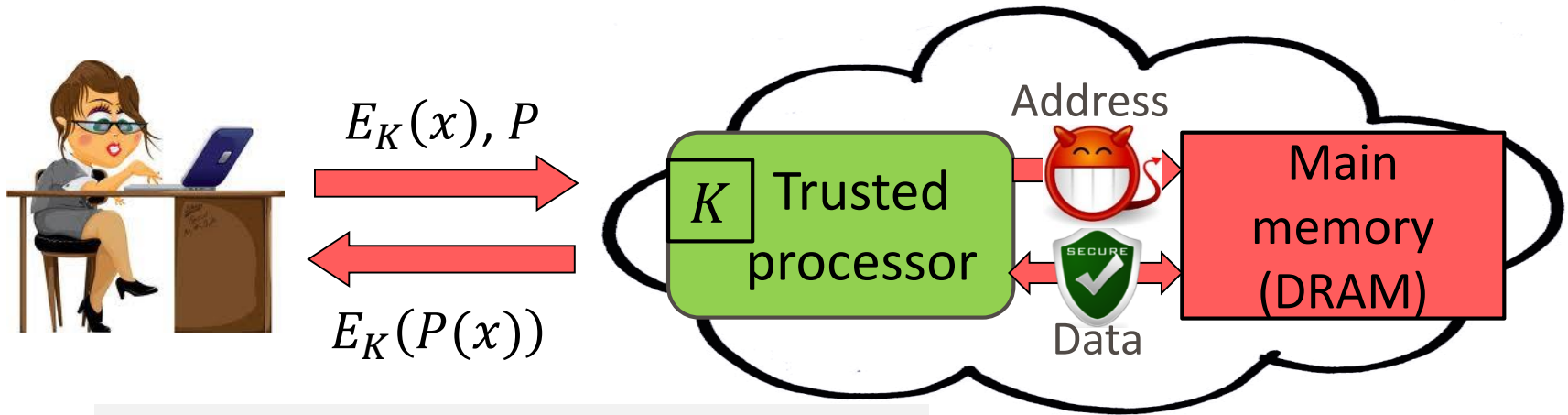
**2013/09/20 Hornet workshop**

**Ling Ren, Xiangyao Yu, Christopher W. Fletcher,**  
Marten van Dijk, Srinivas Devadas

- 
- **Ascend Overview**
  - **Basic Path ORAM**
  - **Background eviction**
  - **Integrity verification**
  - **Summary**

- **Context:** cloud computing
- **Privacy:** user's data not leaked to anyone
- **Integrity:** computation is done correctly (user gets  $P(x)$ )



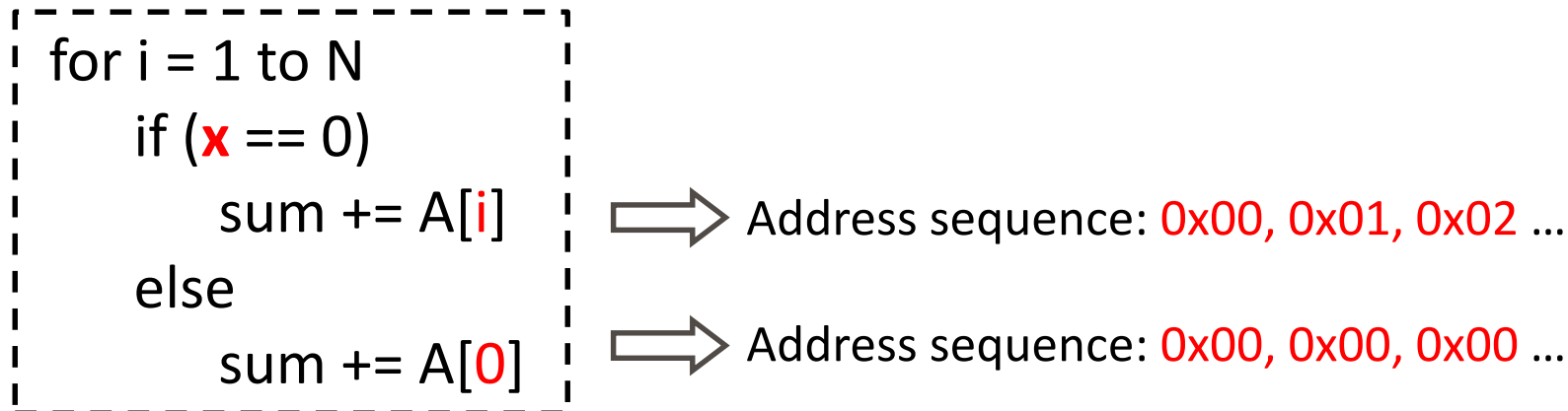


User data is decrypted and computed on inside processor

Data can be encrypted but address cannot

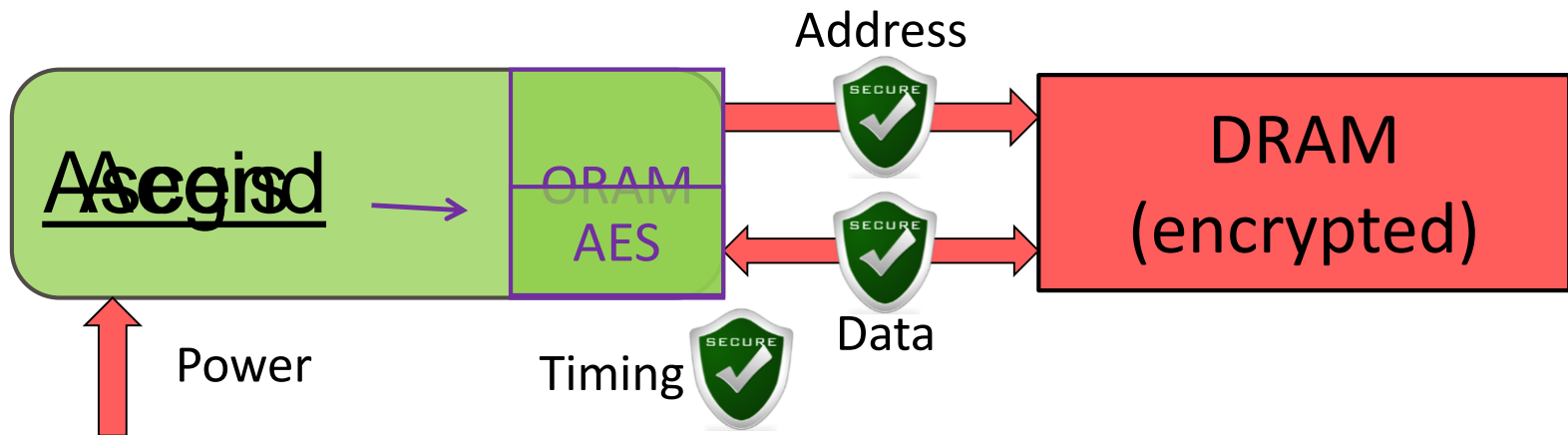
+ Integrity (e.g. Aegis)

– Leakage through address/timing/power

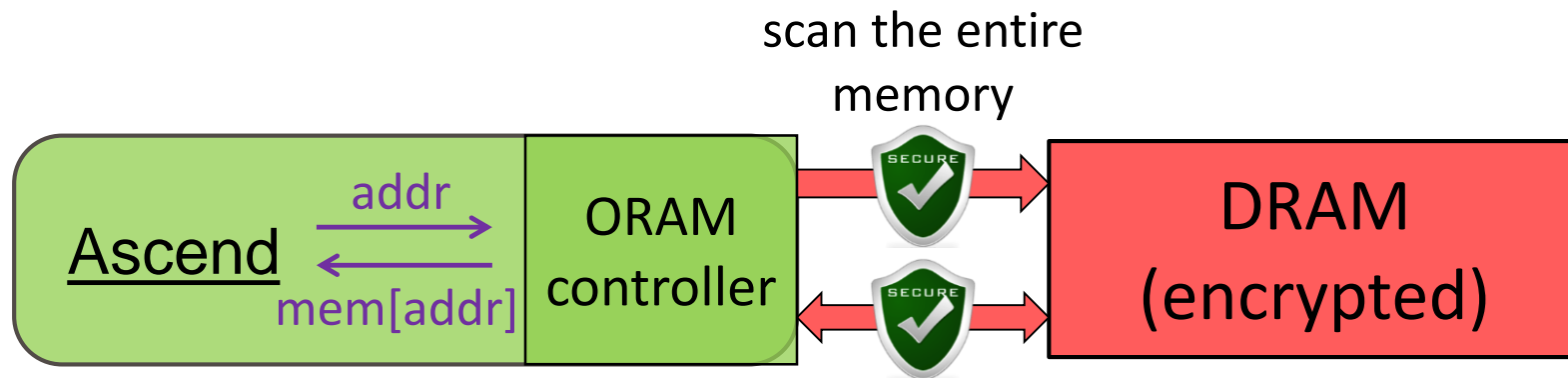


- **Previous work [HIDE, NDSS12] has shown access pattern leakage in practical applications**
- **Addresses can be monitored by software**

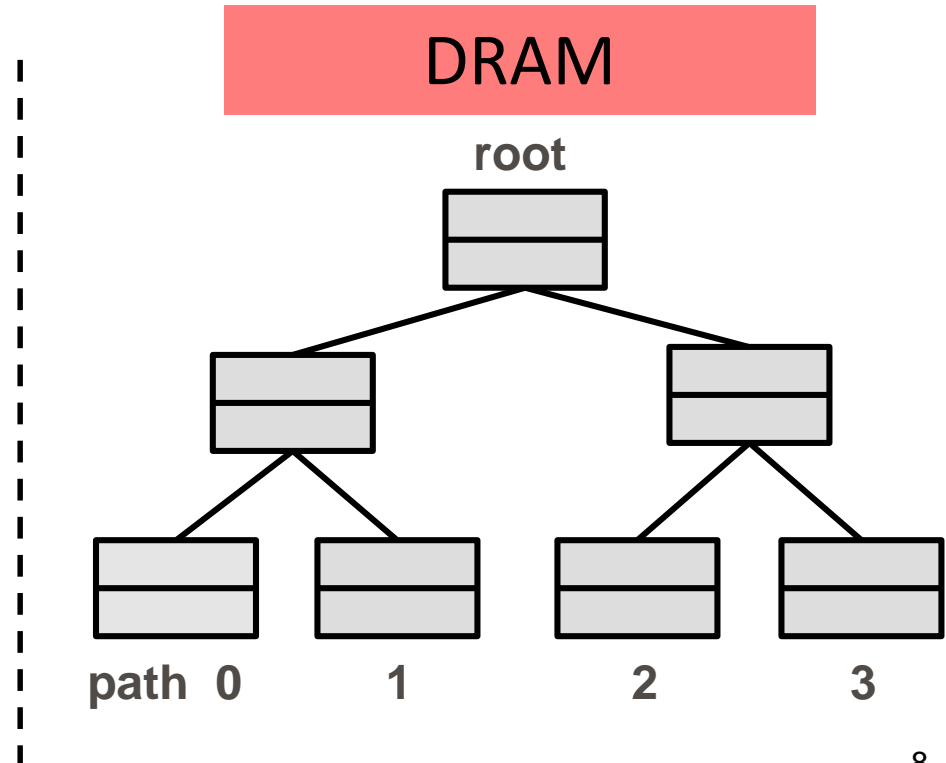
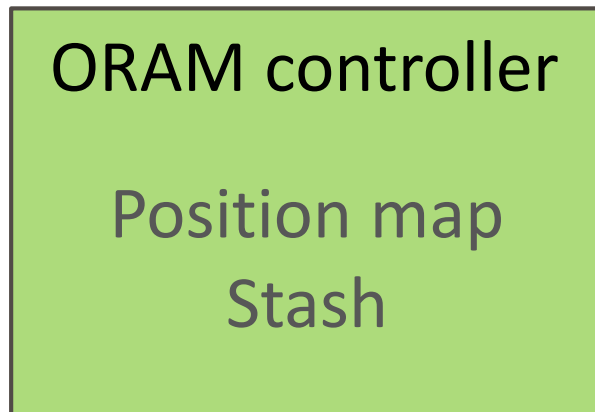
- Existing secure processors (e.g., XOM, Aegis)
  - + Can provide integrity
  - ~~— Leakage through address/timing/power, or trust the program~~
- Ascend: terminate leakage over above channels
  - I/O channel: Oblivious RAM
  - Timing channel: Chris' and Xiangyao' talks
  - Power Channel:



- **Hide access pattern**
  - Read vs. write
  - Make all address sequences indistinguishable
- **Naïve ORAM**
  - Read/write the entire memory on each access
  - Probabilistic encryption  $\rightarrow$  everything always changes
  - $O(N)$  overhead,  $N = \#$  of data blocks (cache lines) in the memory



- **Path ORAM**
  - One of the most efficient ORAMs, simple
- **External DRAM structured as a binary tree**
  - Each node contains  $Z$  blocks ( $Z = 2$  in the example below)





- **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

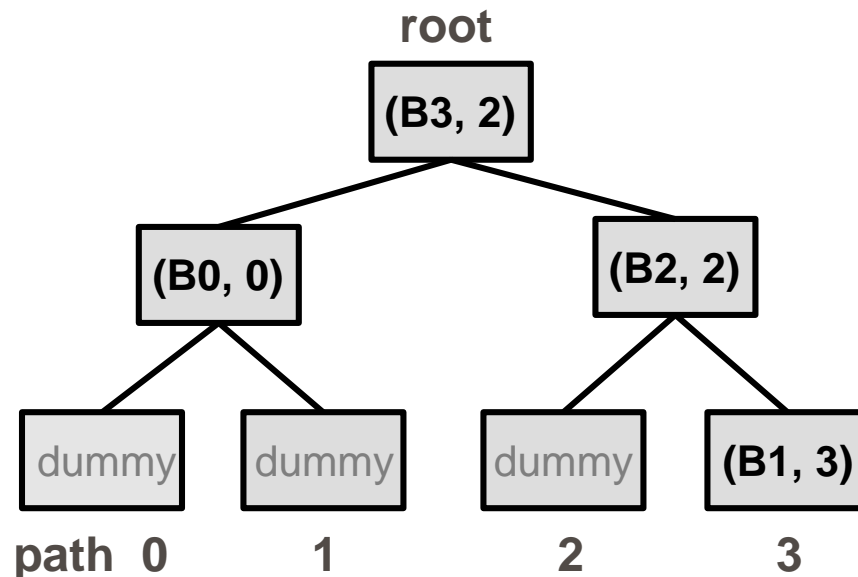
### Stash



### Position Map

Block	Path
B0	0
B1	3
B2	2
B3	2
B4	1

## DRAM



- **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)

$$O(L) = O(\log N)$$

## ORAM controller

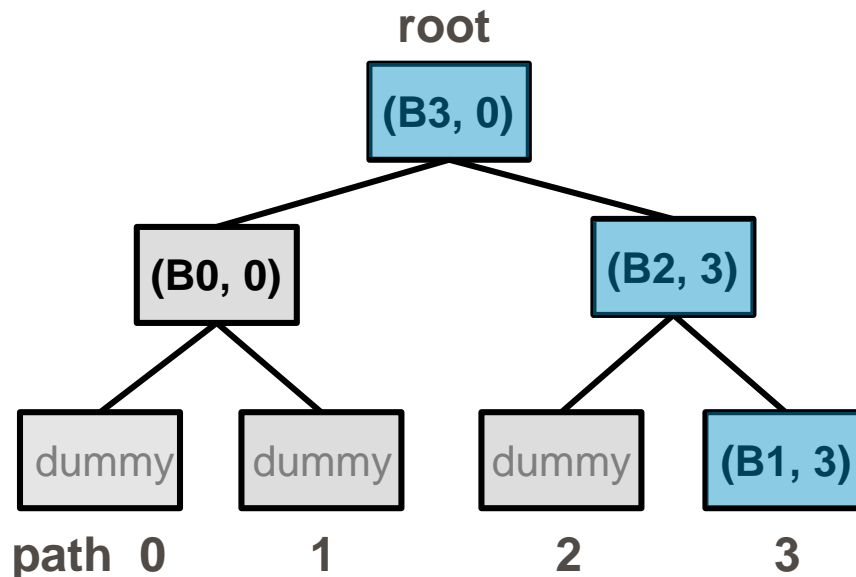
Stash      dummy



### Position Map

Block	Path
B0	0
B1	<del>X</del> 1
B2	3
B3	0
B4	1

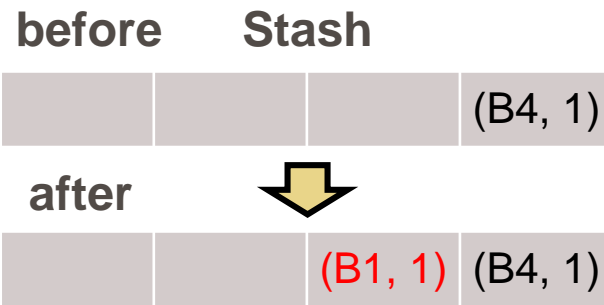
## DRAM



- **A random path is read/written on every access**
  - Extracted from PosMap, which is always random and fresh due to remapping
- **All ciphertexts on the path always change**
  - Due to probabilistic encryption

- **Stash overflow probability?**

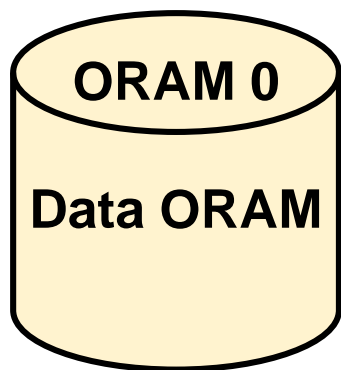
- Negligible (provably or empirically) if  $Z \geq 4$
- Always overflow if  $Z \leq 3$
- Prefer smaller  $Z$  because  $O(ZL)$



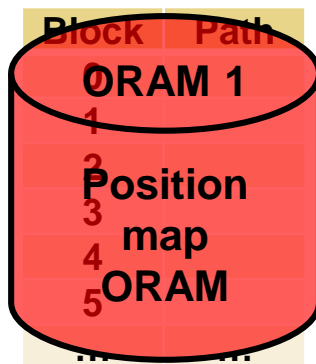
Background eviction prevents overflow

- **Position map too large**

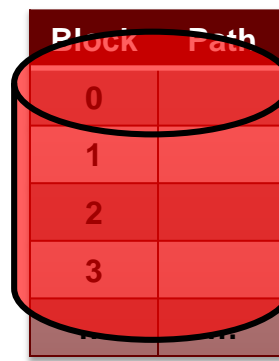
- Recursive Path ORAM → new problem: integrity



4 GB



93 MB



2 MB

Block	Path
0	
1	
...	...

43 KB

- 
- Ascend Overview
  - Basic Path ORAM
  - **Background eviction**
  - **Integrity verification**
  - **Summary**

- When the stash is almost full, read/write a random path
  - Hope there is a dummy and a block in stash can go there
  - If no dummy, all blocks can at least go back (stash will not increase)

## ORAM controller

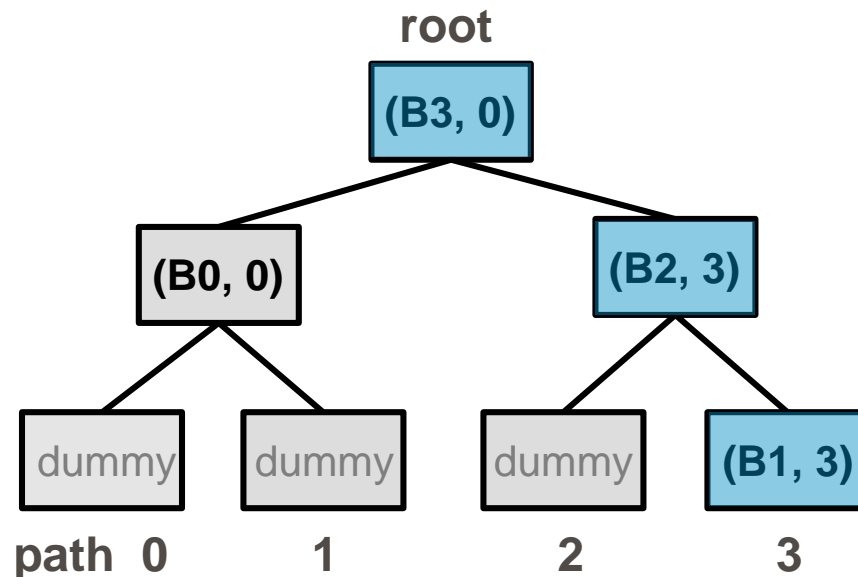
### Stash



### Position Map

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

## DRAM



- When the stash is almost full, read/write a random path
  - Hope there is a dummy and a block in stash can go there
  - Luckily, path 0 has a dummy

## ORAM controller

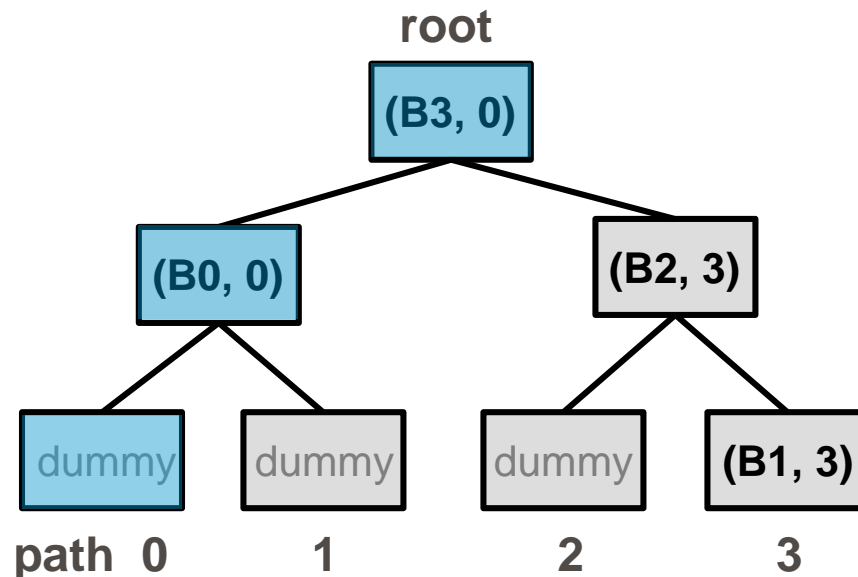
### Stash



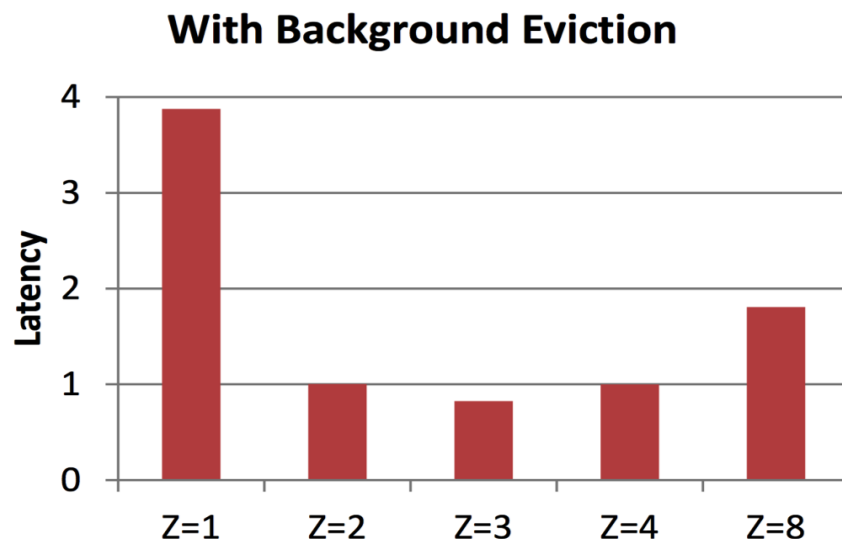
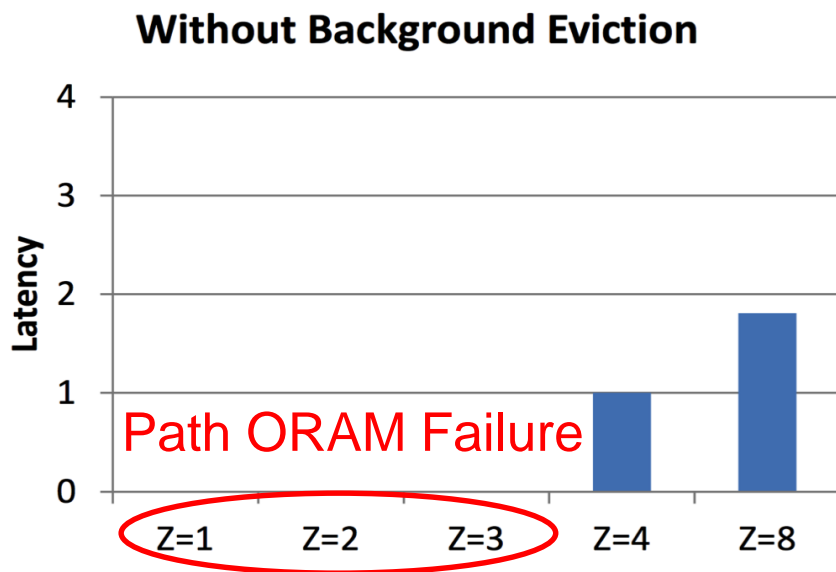
### Position Map

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

## DRAM




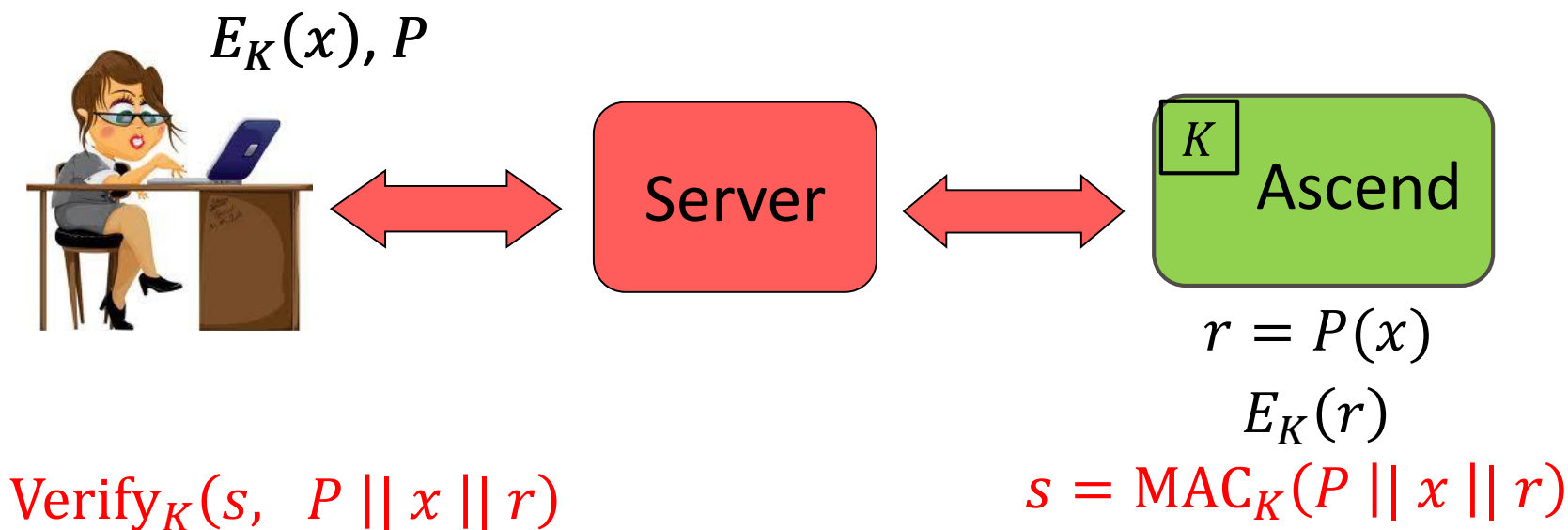
- **Security: indistinguishable from normal accesses**
  - Read/write a random path
  - No need to remap since no leaf label is exposed
- **Impact on performance**



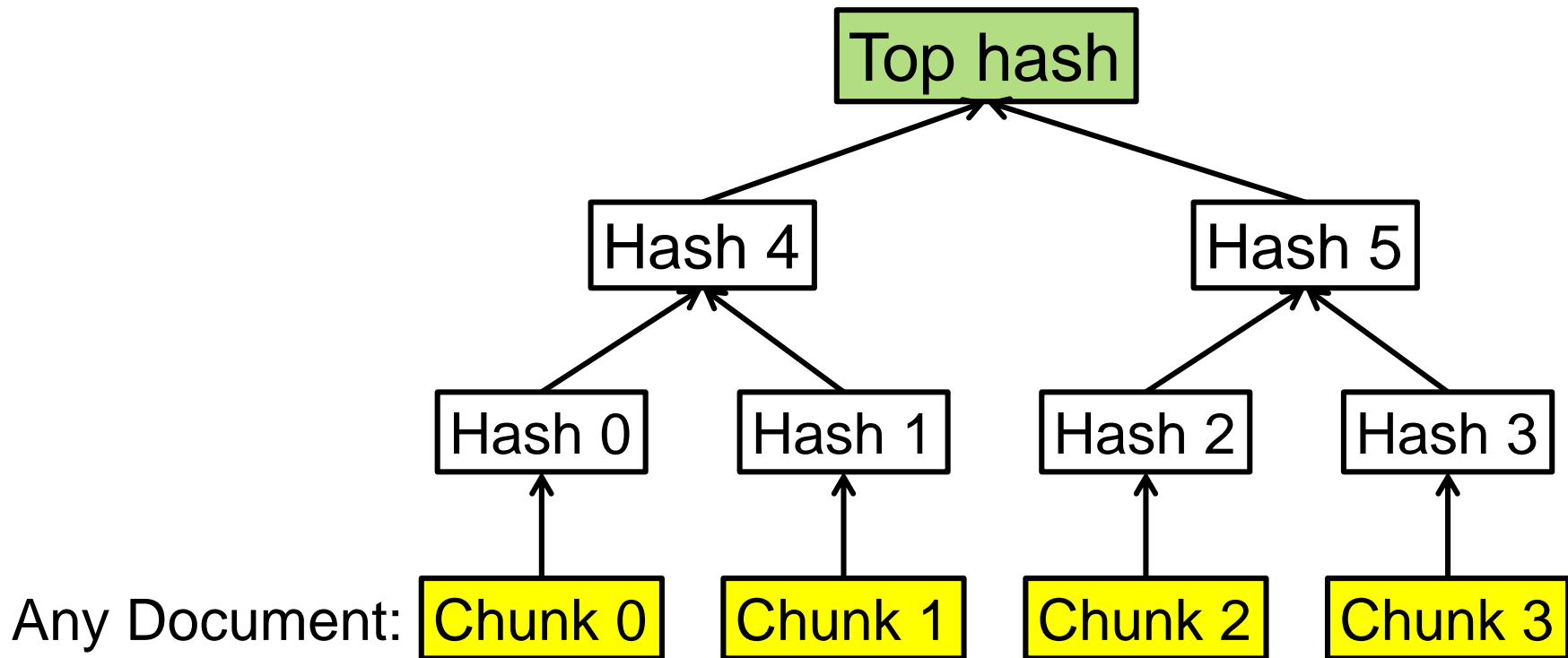


- 
- Ascend Overview
  - Basic Path ORAM
  - Background eviction
  - **Integrity verification**
  - **Summary**

- Recursive Path ORAM is **insecure** without integrity verification when attackers can modify ORAM
  - Revert PosMap ORAMs to force reuse of old leaf labels 
- Another motivation: integrity in Ascend
  - Need to verify input/output and external memory



- **General**, can be used for any document, any ORAM
- **Efficient**  $O(L) = O(\log N)$
- **Security reduced to collision-resistant hash function**

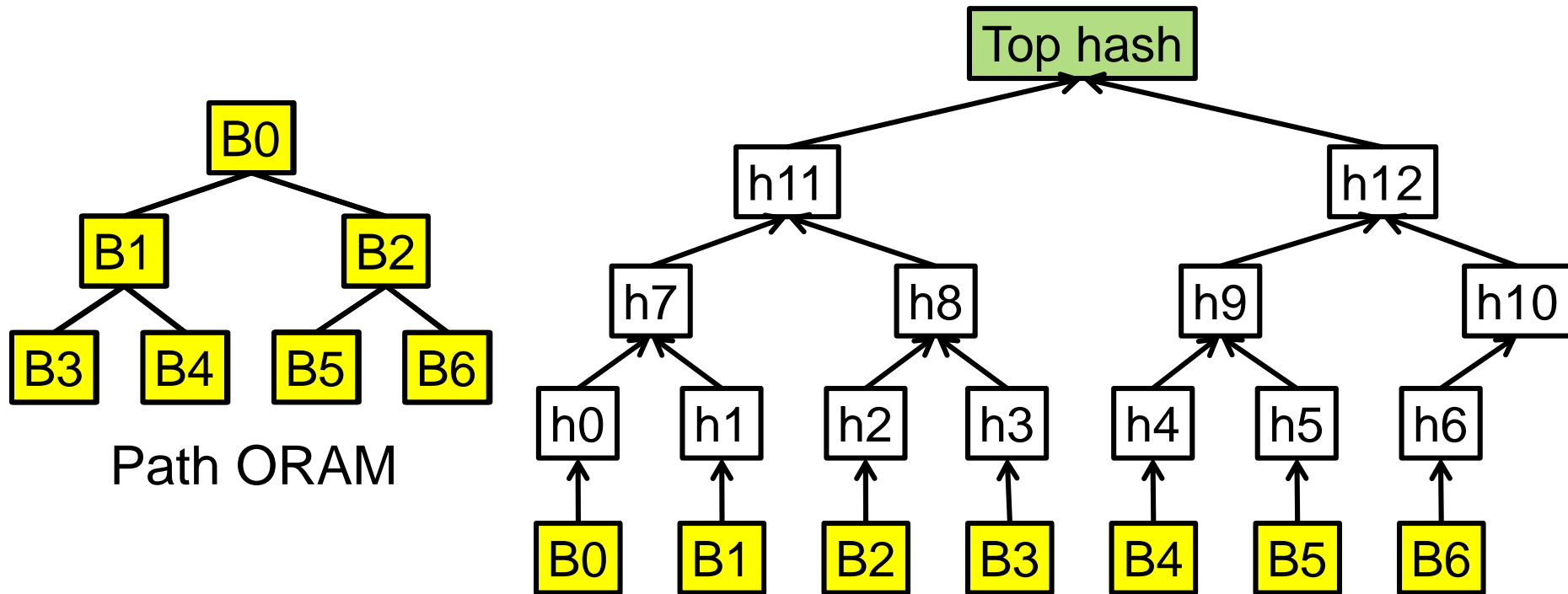




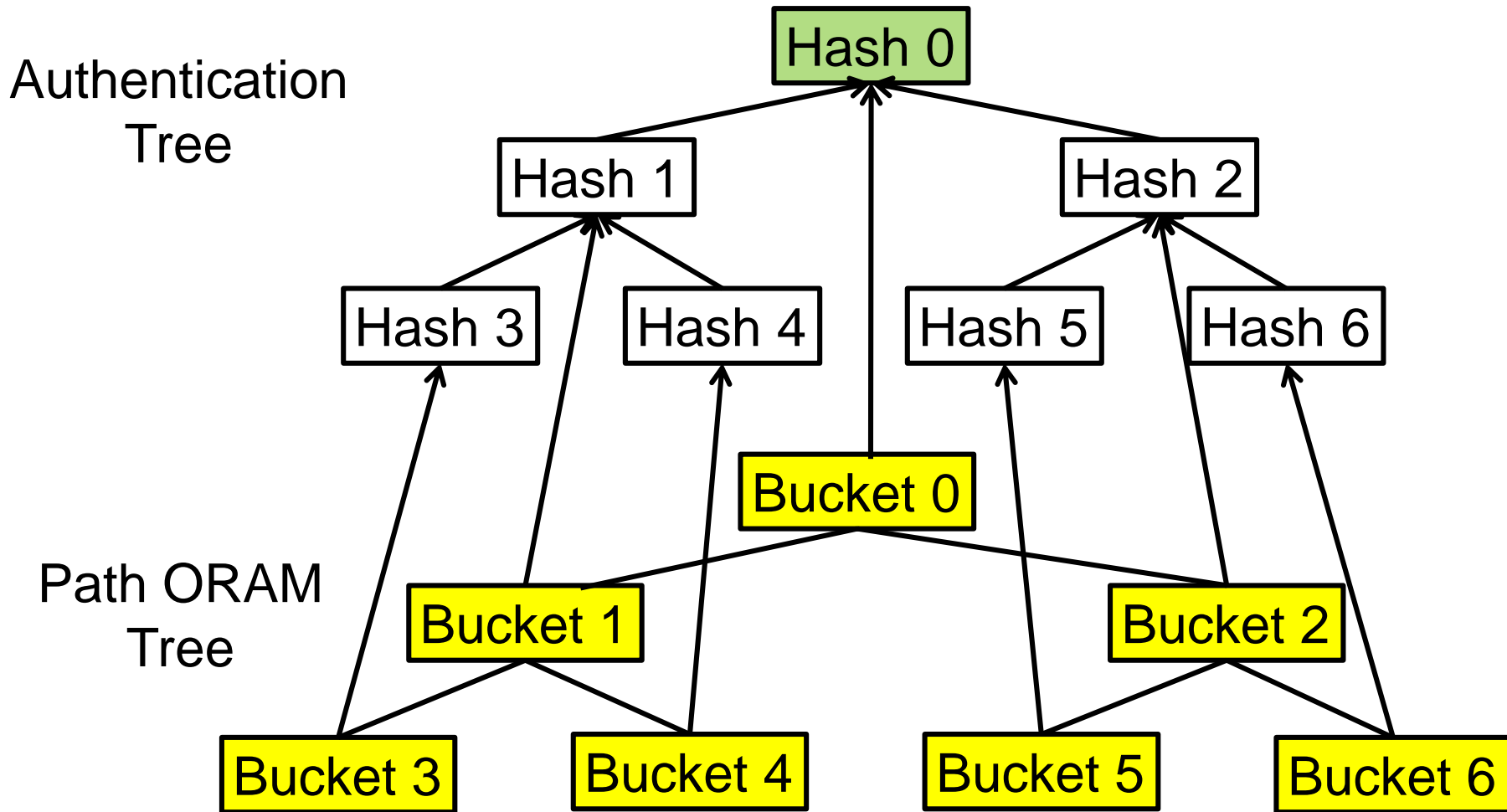
# Merkle Signature for Path ORAM?

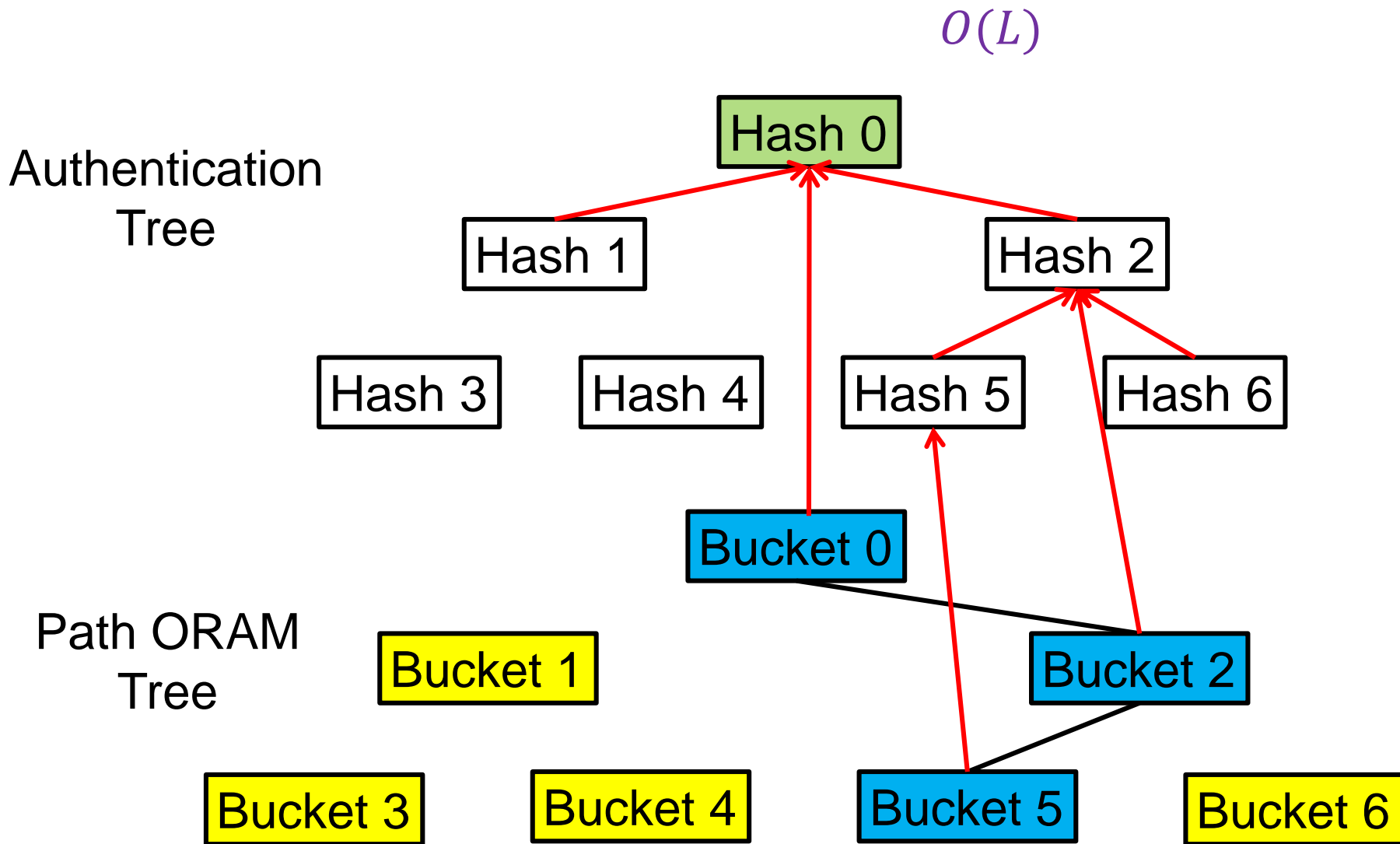


- ORAM hides access pattern
  - (pretend to) verify all buckets on a path
  - $O(L^2)$  complexity
  - Path ORAM  $O(L)$  complexity



- Combine Merkle tree and Path ORAM tree





- 
- Ascend Overview
  - Basic Path ORAM
  - Background eviction
  - Integrity verification
  - **Summary**

- **Background eviction prevents stash overflow, enables smaller  $Z$ , and improve performance by 20%**
- **Overhead relative to DRAM**
  - Bit movement: ~300x
  - Latency: ~50x (assuming 2 channels)
  - On SPEC: 1x ~ 10x
- **Integrity verification adds 17% latency on top of recursive Path ORAM**
- **We are designing the on-chip Path ORAM controller!**