Path Oblivious-RAM for Secure Processors

2013/09/20 Hornet workshop

Ling Ren, Xiangyao Yu, Christopher W. Fletcher,

Marten van Dijk, Srinivas Devadas

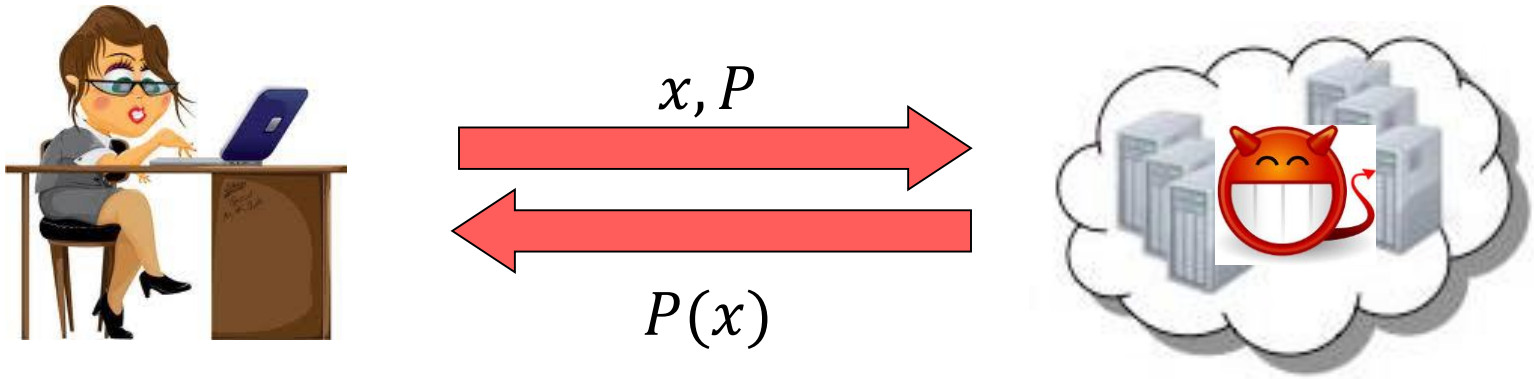
Outline

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

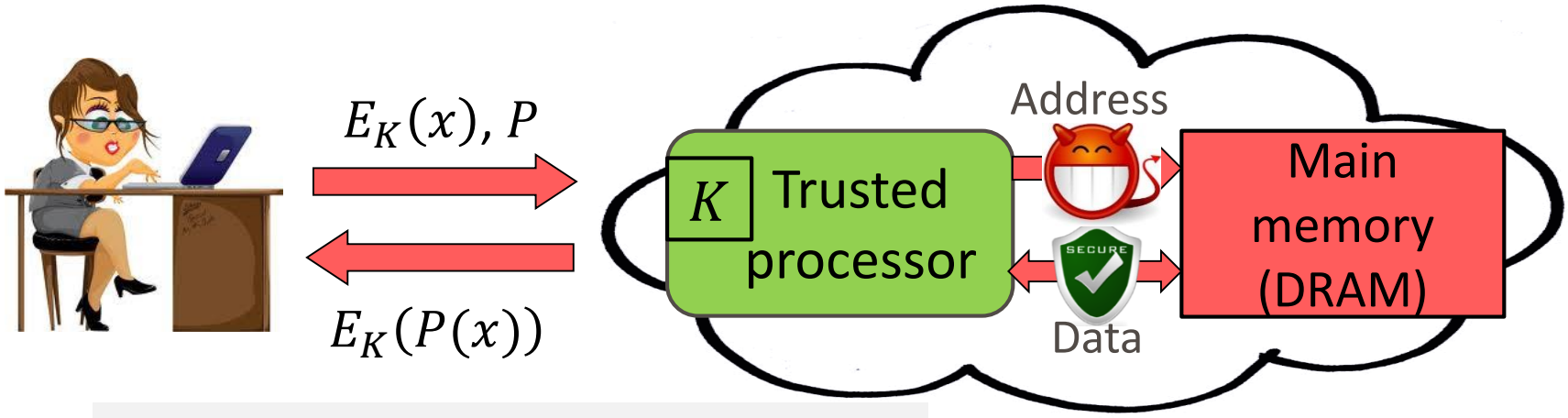
Privacy & Integrity in Cloud

- **Context: cloud computing**

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



Secure Processors



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

Leakage through Addresses

```
for i = 1 to N
    if (x == 0)
        sum += A[i]
    else
        sum += A[0]
```

⇨ Address sequence: 0x00, 0x01, 0x02 …

⇨ Address sequence: 0x00, 0x00, 0x00 …

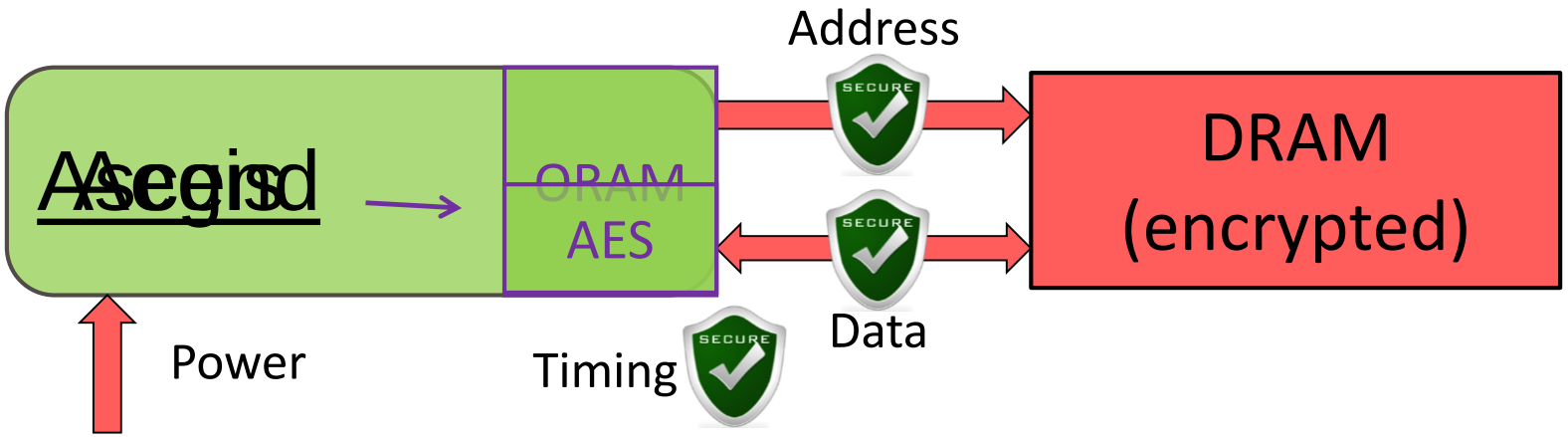
- **Previous work** [HIDE, NDSS12] **has shown access pattern leakage in practical applications**

- **Addresses can be monitored by software**

Ascend secure processor

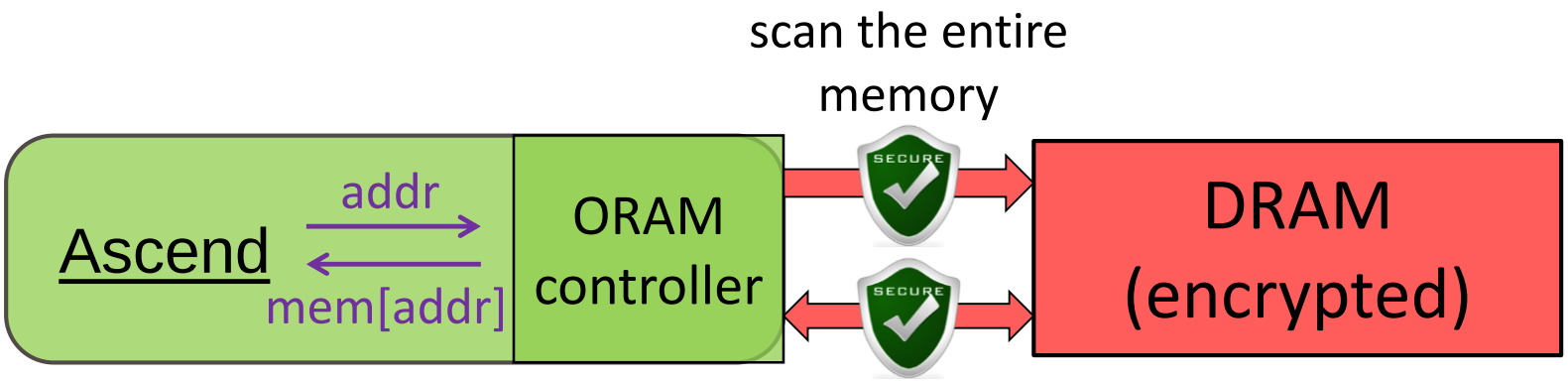
- **Existing secure processors (e.g., XOM, Aegis)**
  - + Can provide integrity
  - ~~– Leakage through address/timing~~/power, ~~or trust the program~~

- **Ascend: terminate leakage over above channels**
  - – I/O channel: Oblivious RAM
  - – Timing channel: Chris' and Xiangyao' talks
  - – Power Channel:



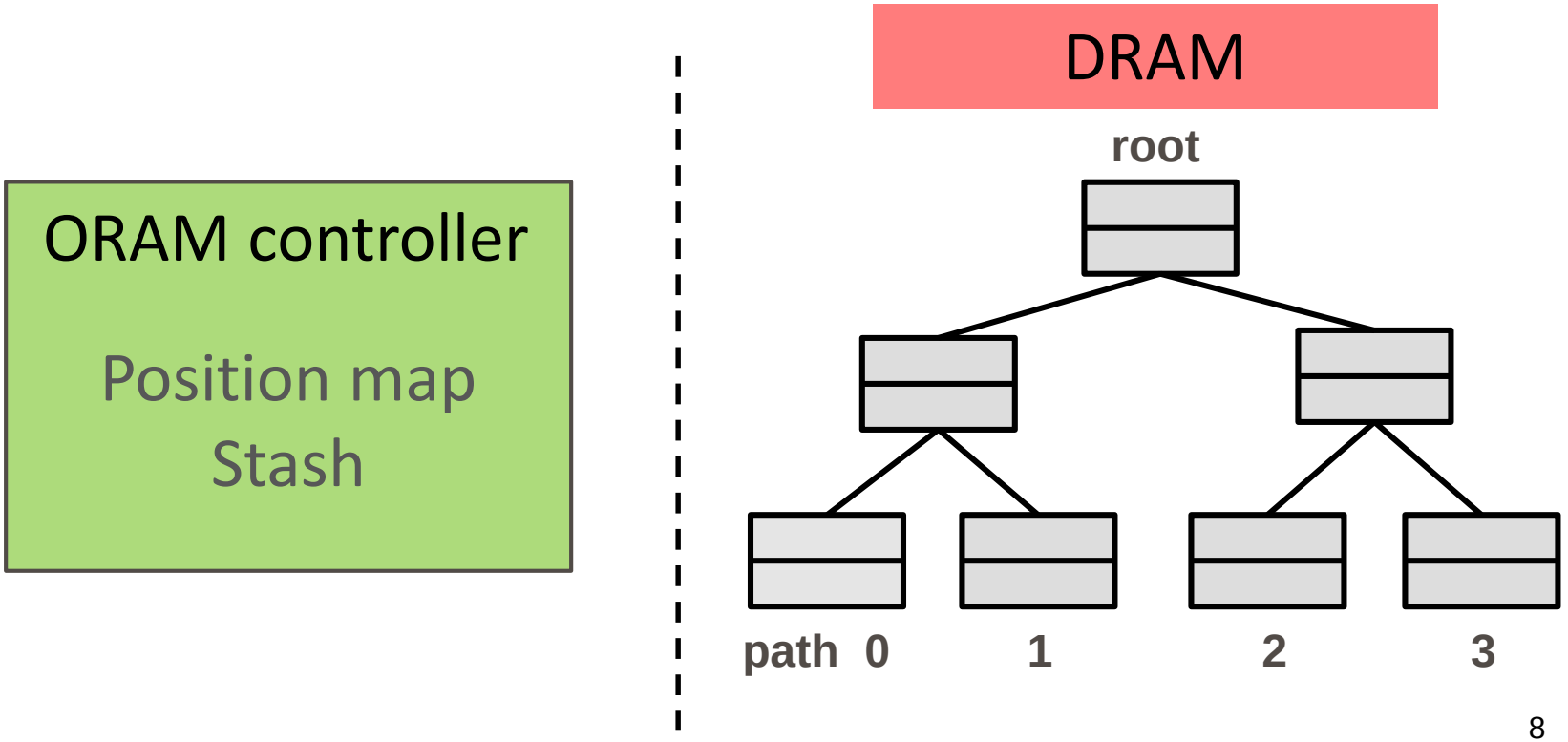
Oblivious RAM (ORAM)

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



Path ORAM

- **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)



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

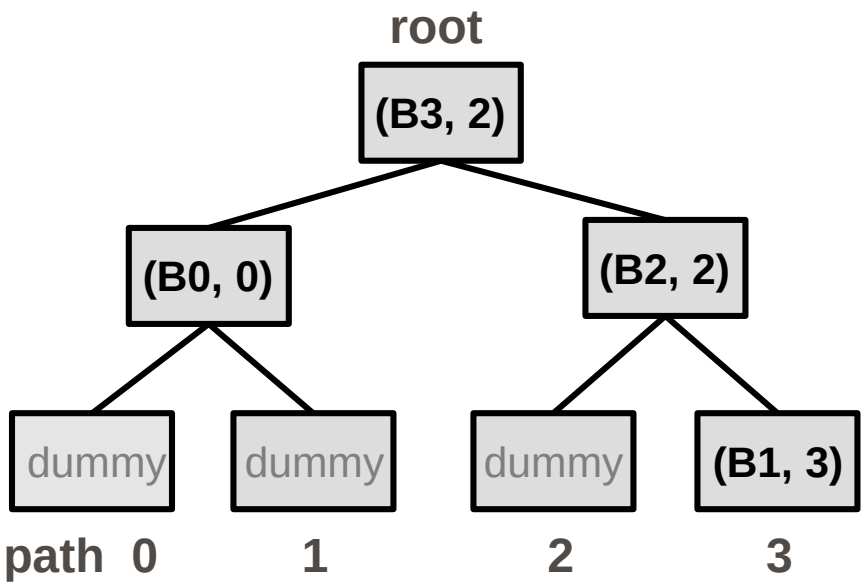
Stash

			(B4, 1)

Position Map

Block	Path
B0	0
B1	3
B2	2
B3	2
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)

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

ORAM controller

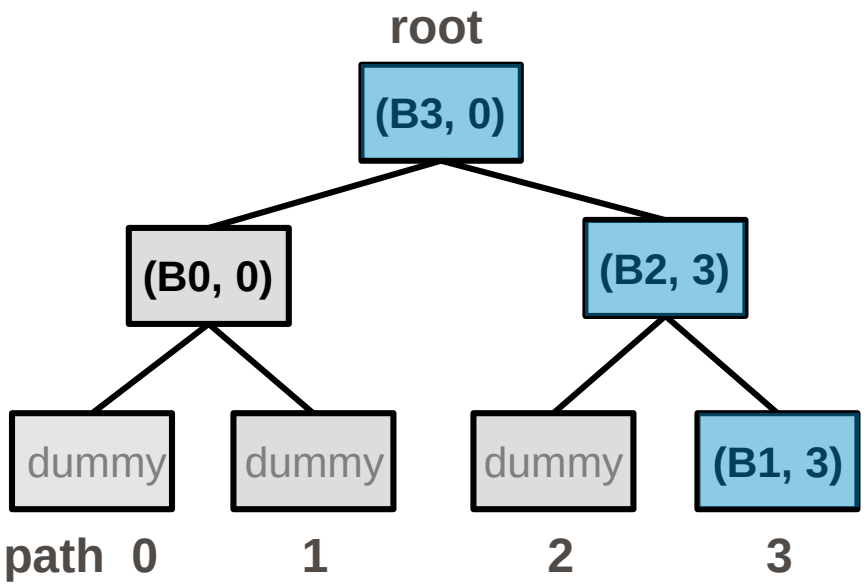
Stash

			dummy
		(B1, 1)	(B4, 1)

Position Map

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

DRAM

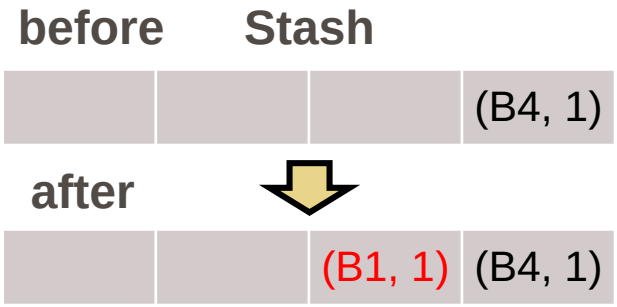


Path ORAM Security

- **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

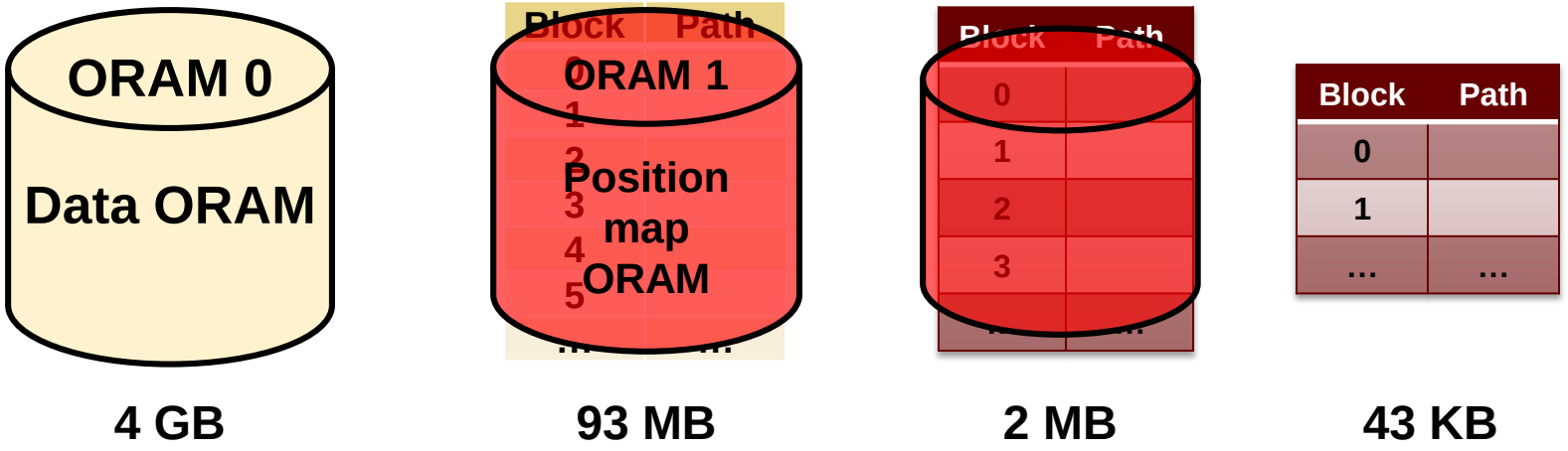
Basic Path ORAM Problems

- **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



Outline

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

Background eviction

- **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

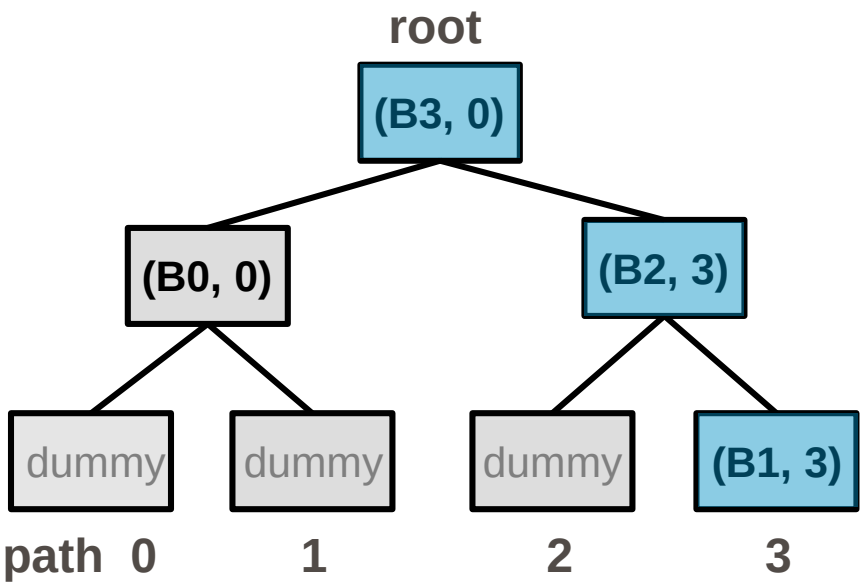
Stash

			(B4, 1)

Position Map

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

DRAM



Background eviction

- **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

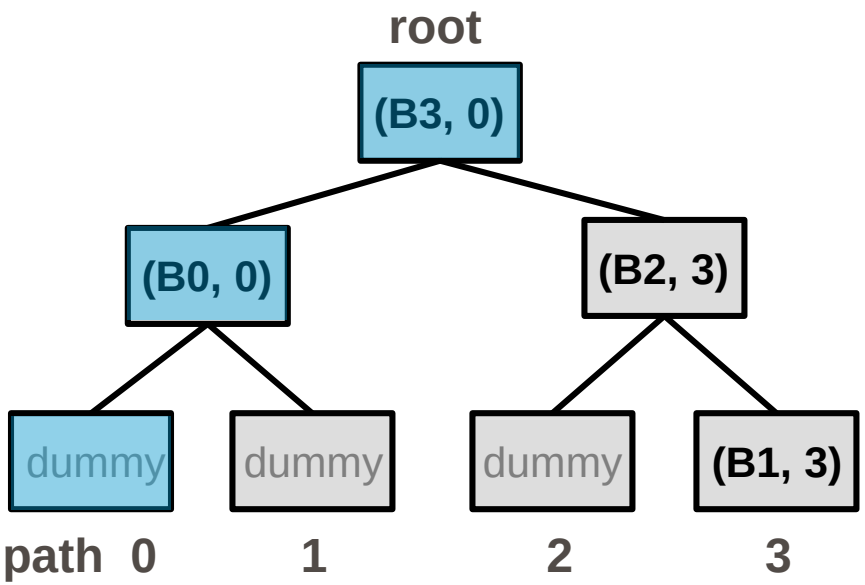
Stash

			(B4, 1)

Position Map

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

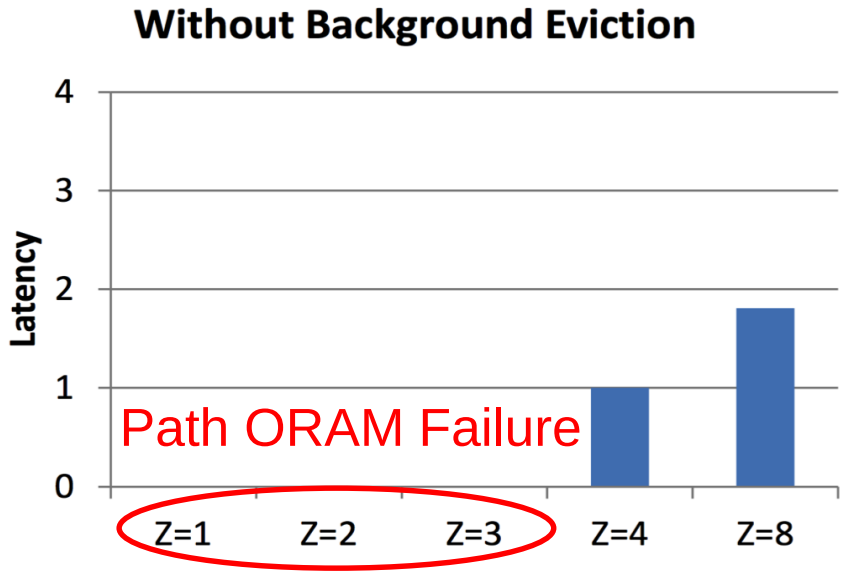
DRAM

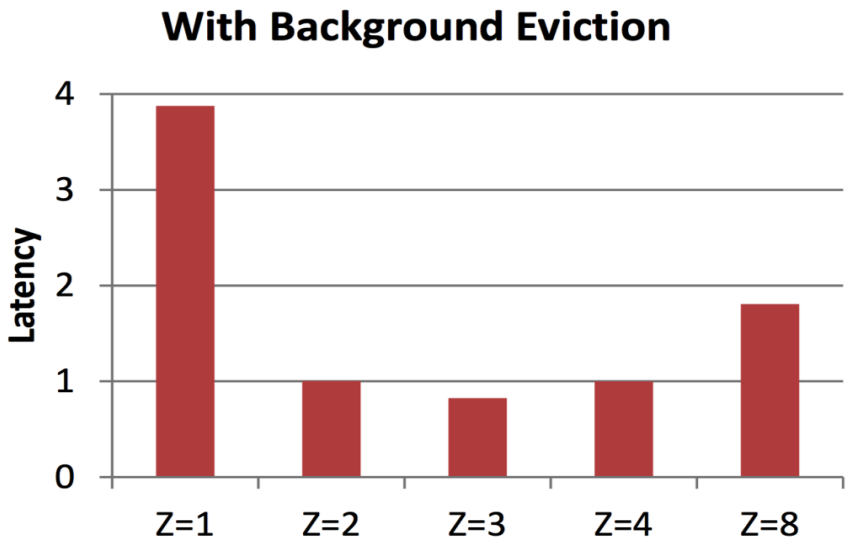


Background eviction

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

- **Impact on performance**





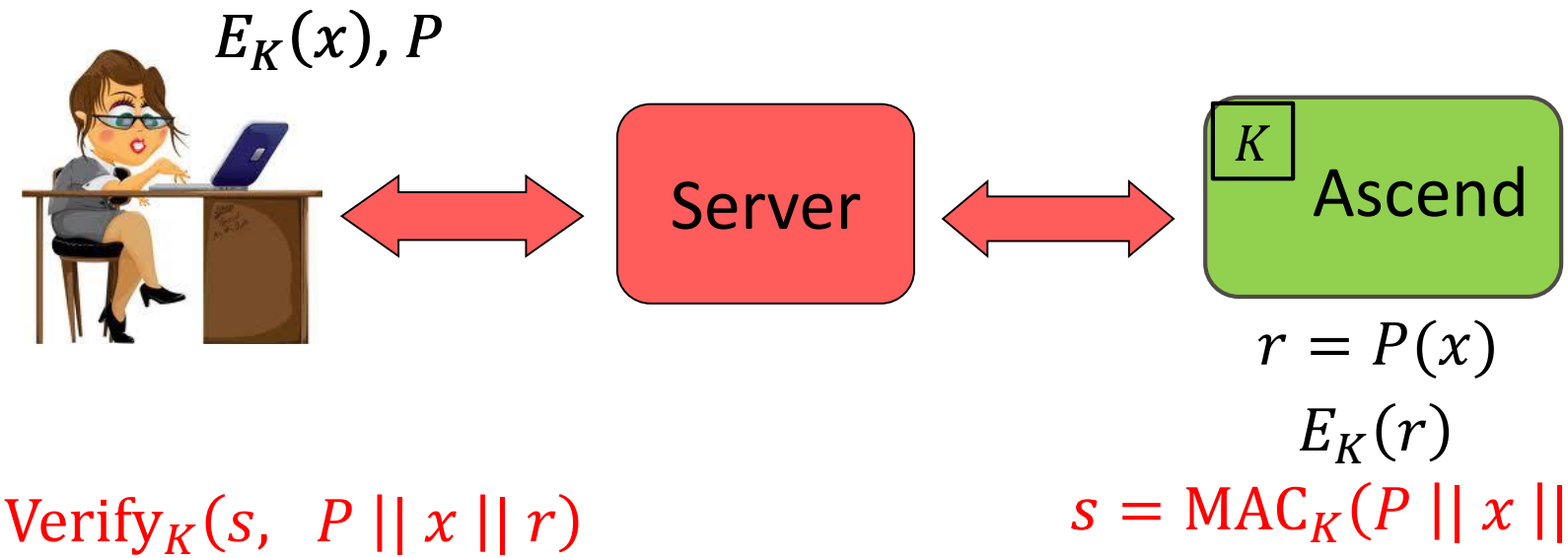
Outline

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

Motivation

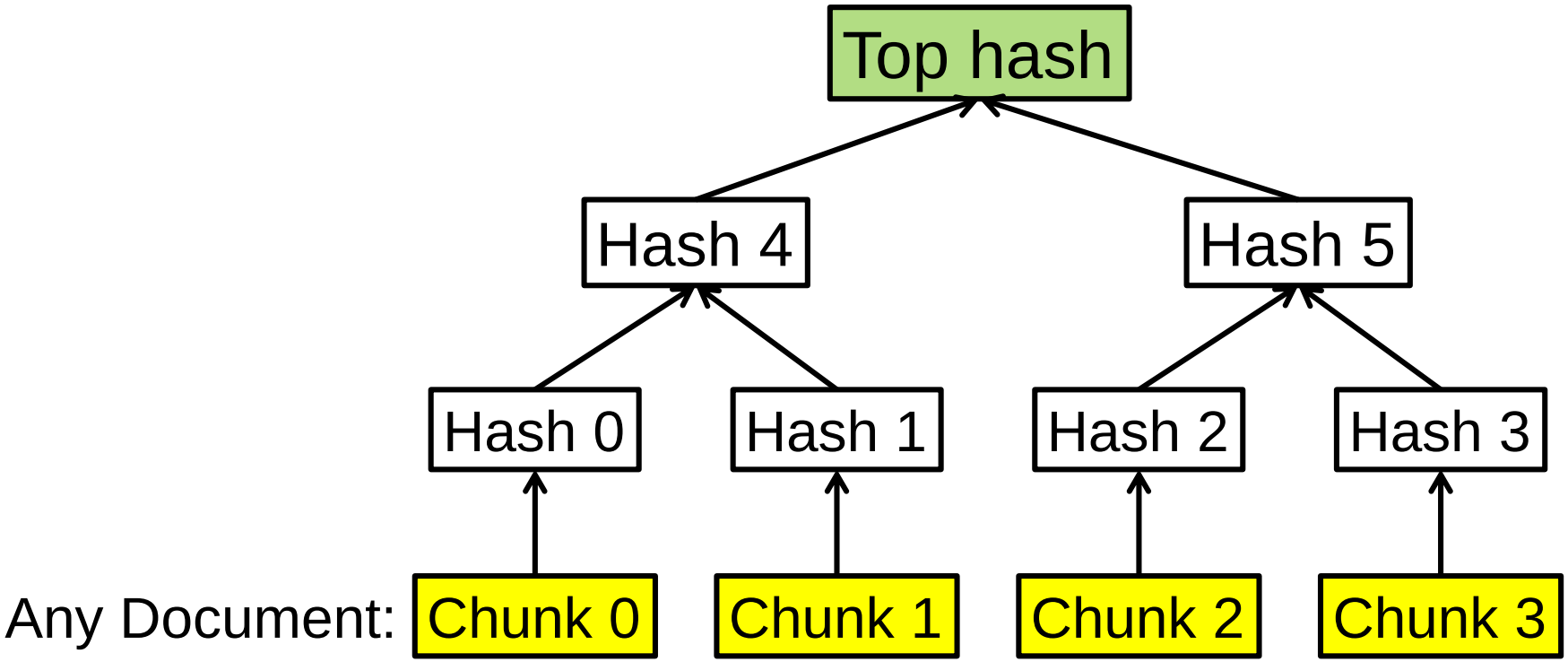
- **Recursive Path ORAM is insecure without integrity verification when attackers can modify ORAM**
  - Revert PosMap ORAMs to force reuse of old leaf labels

- **Another motivation: integrity in Ascend**
  - Need to verify input/output and external memory



Background – Merkle Signature

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



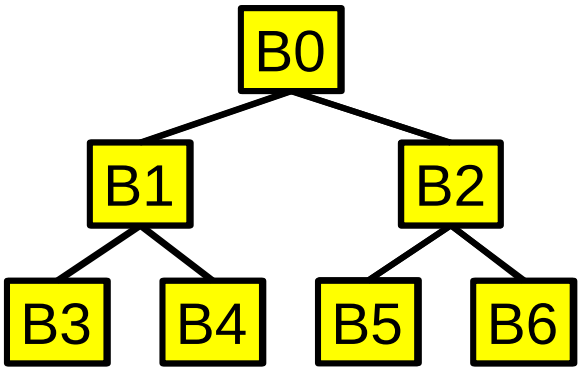
Merkle Signature for Path ORAM?

- **ORAM hides access pattern**

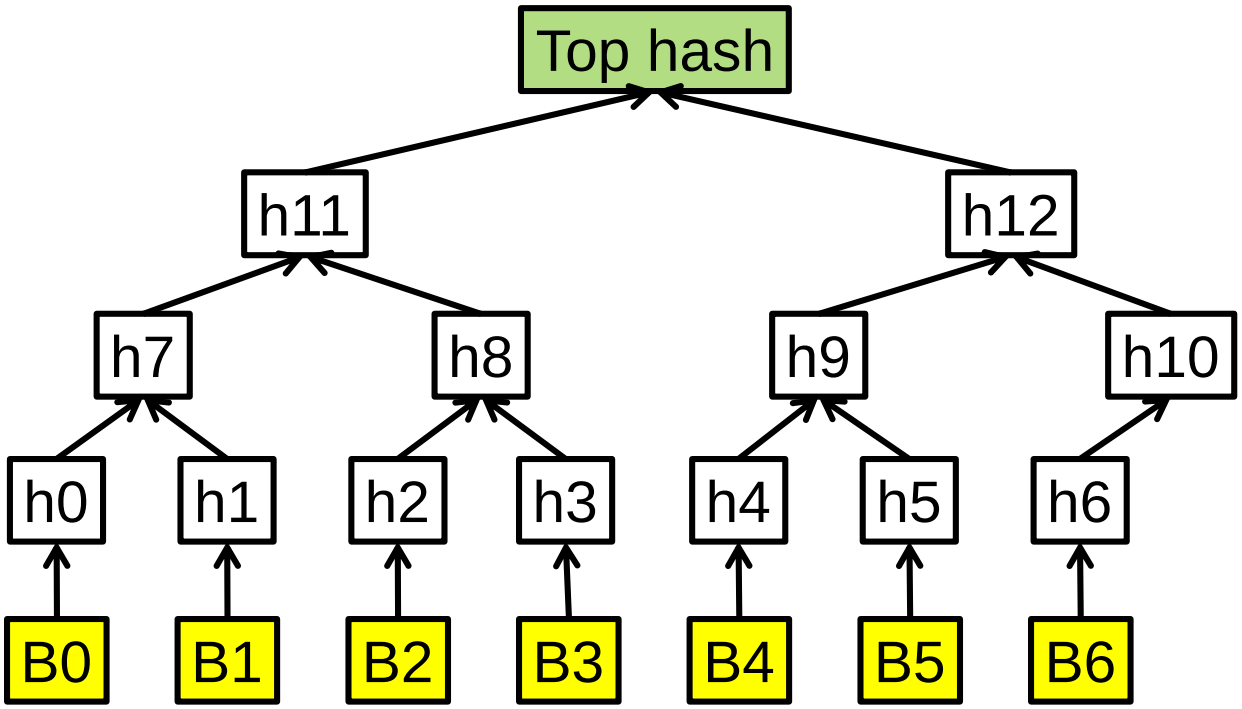
  → (pretend to) verify all buckets on a path

  → $O(L^2)$ complexity

  – Path ORAM $O(L)$ complexity

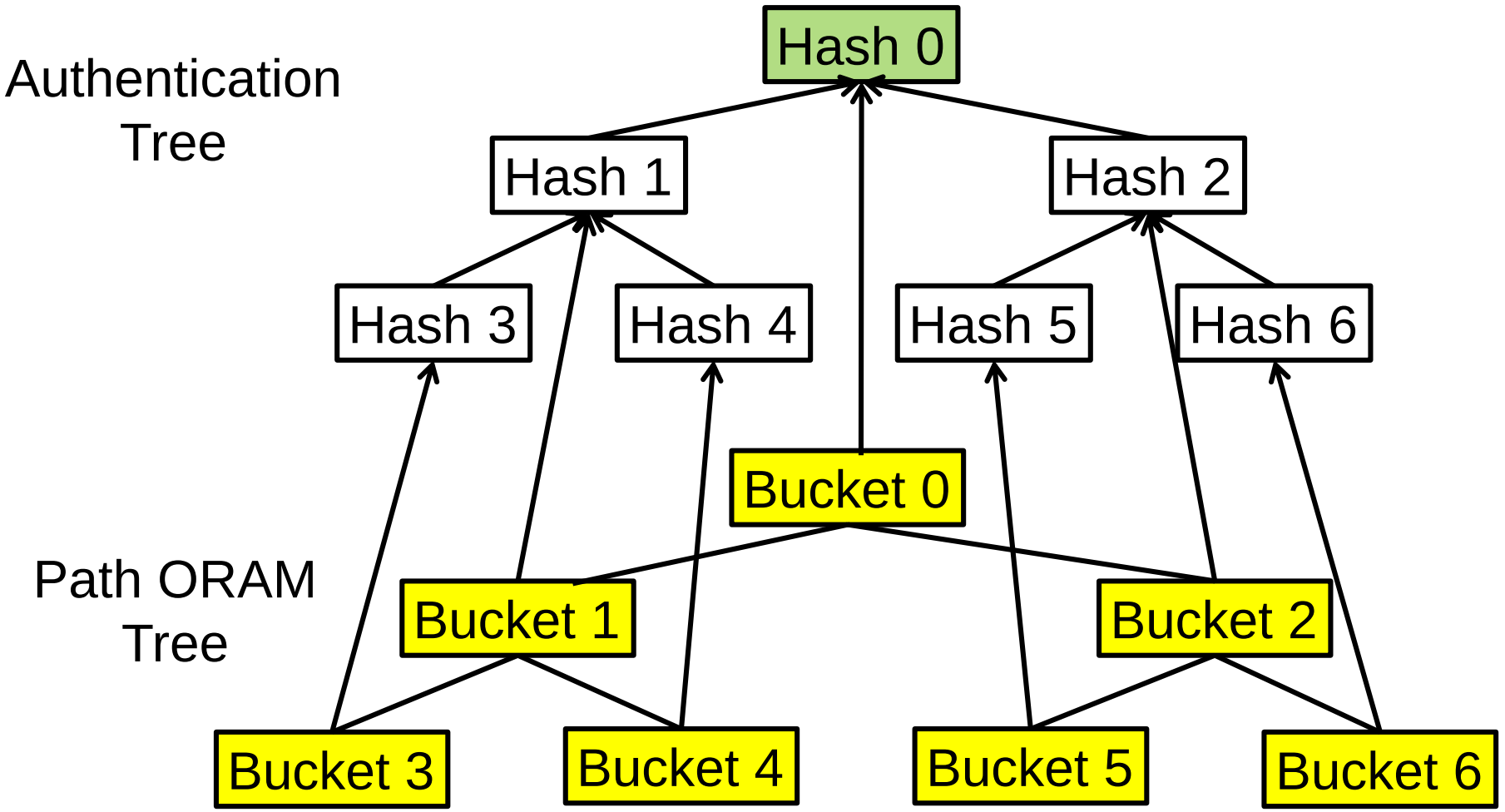


Path ORAM

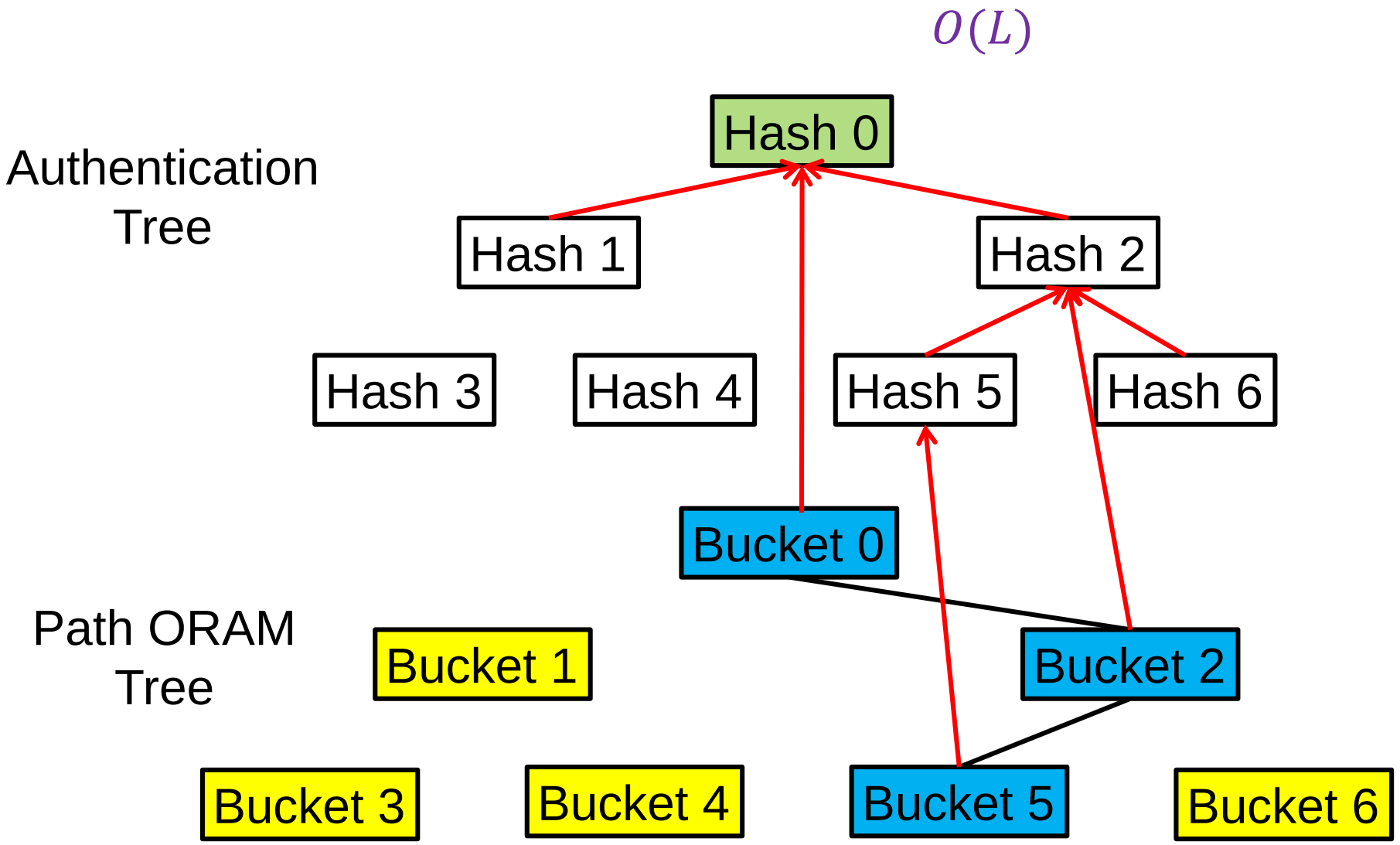


Verify Path ORAM

- **Combine Merkle tree and Path ORAM tree**



Verify Path ORAM



Outline

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

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!**