Hierarchical ORAM Revisited, and Applications to Asymptotically Efficient ORAM and OPRAM

Hubert Chan, Yue Guo, Wei-Kai Lin, Elaine Shi

2017/12/5

香 港 大 學
THE UNIVERSITY OF HONG KONG

Cornell University

Random Access Machine, RAM

- Maybe the standard model of algorithms

Memory / Server:
N words, indexed by address

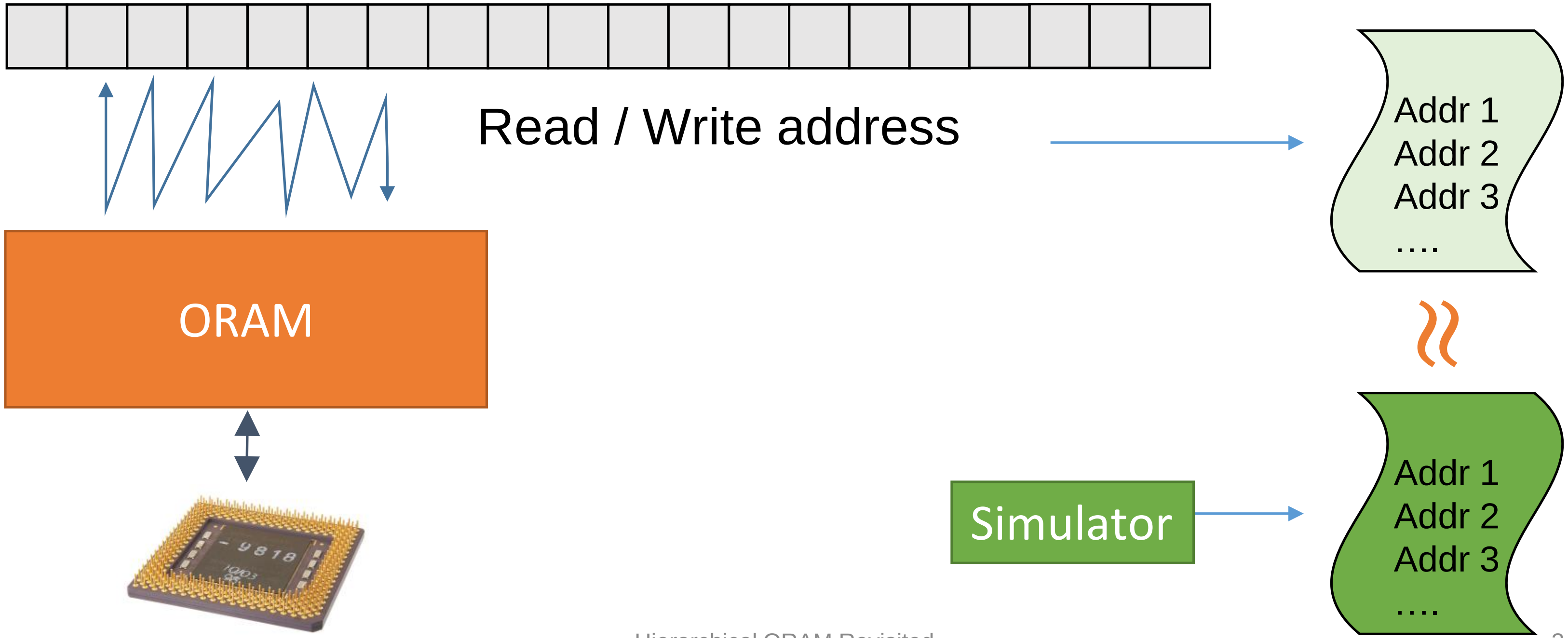
Interface:
- Read / Write **address**

CPU / Client:
Constant num. of registers

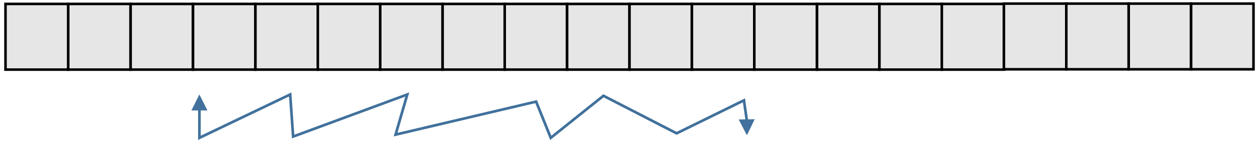
Oblivious RAM (ORAM)

[Goldreich and Ostrovsky]

- Provable security ☺



Hierarchical ORAM Schemes (1)



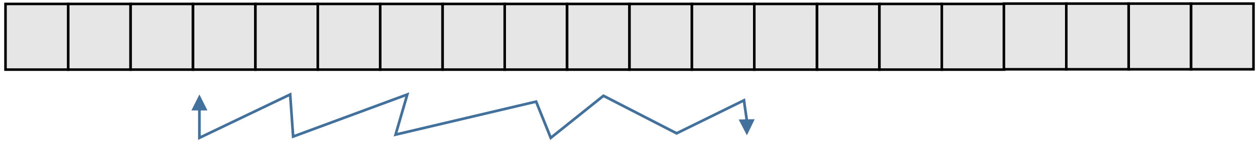
ORAM

[Goldreich and Ostrovsky]

Bandwidth overhead:

$$O(\sqrt{N})$$

Hierarchical ORAM Schemes (2)



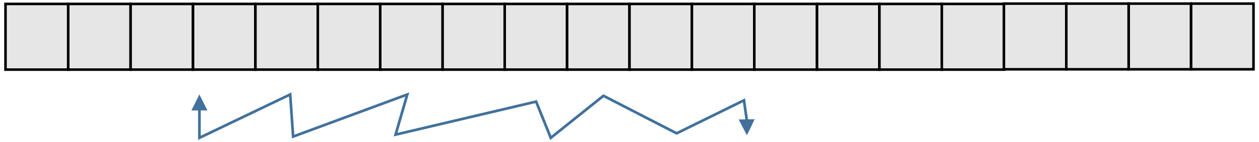
ORAM

[Goldreich and Ostrovsky]

Bandwidth overhead:

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

Hierarchical ORAM Schemes (3)



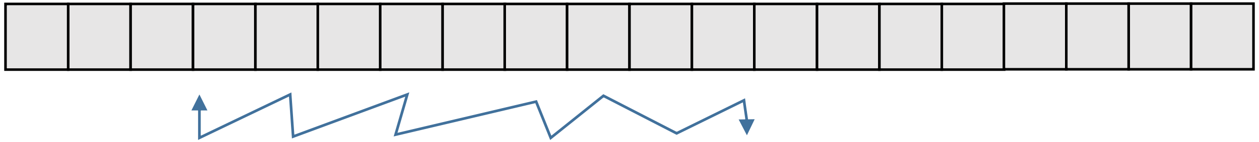
ORAM

[Goodrich and Mitzenmacher]

Bandwidth overhead:

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

Hierarchical ORAM Schemes (4)



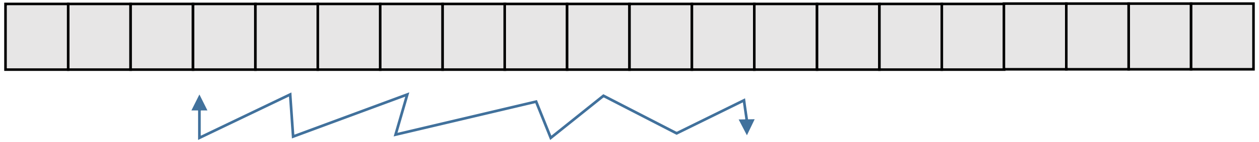
ORAM

[Kushilevitz, Lu, and Ostrovsky]

Bandwidth overhead:

$$O\left(\frac{\log^2 N}{\log\log N}\right)$$

Hierarchical ORAM Schemes (5)



ORAM

[Today]

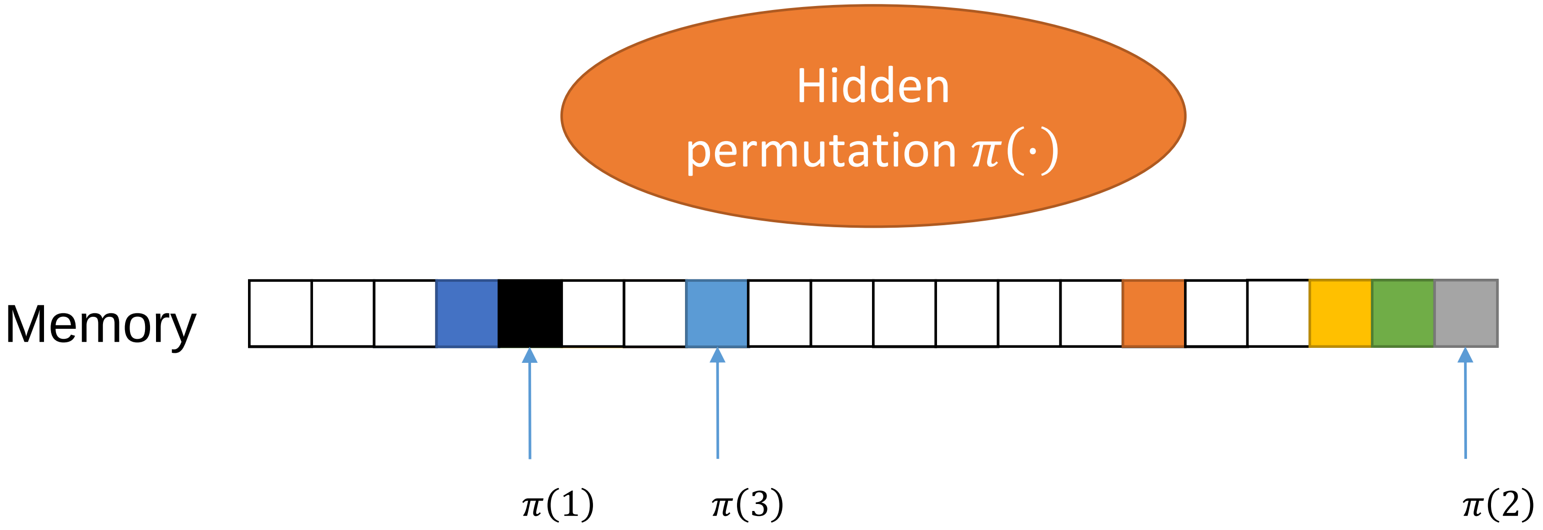
Simple

Bandwidth overhead:

$$O\left(\frac{\log^2 N}{\log\log N}\right)$$

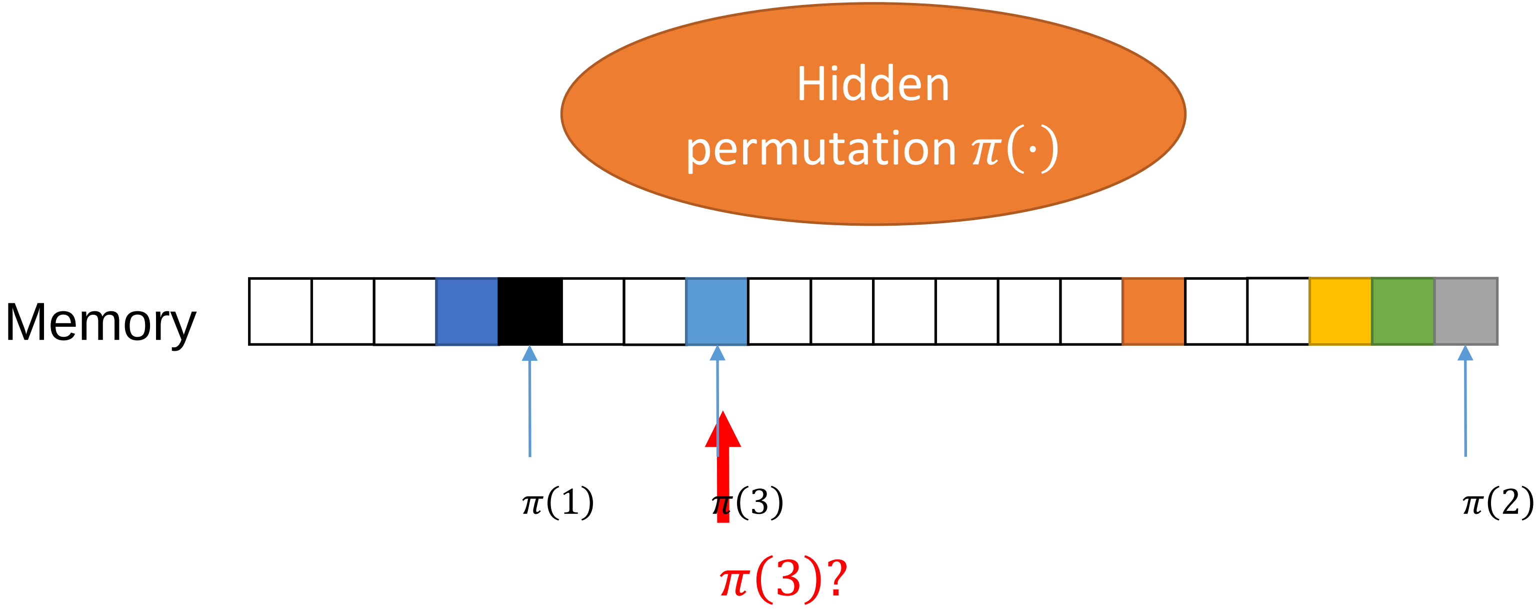
Warmup: Permute and Buffer

[Goldreich and Ostrovsky]



Warmup: Permute and Buffer

[Goldreich and Ostrovsky]



Warmup: Permute and Buffer

[Goldreich and Ostrovsky]

Buffer

3?3?

3? ☺

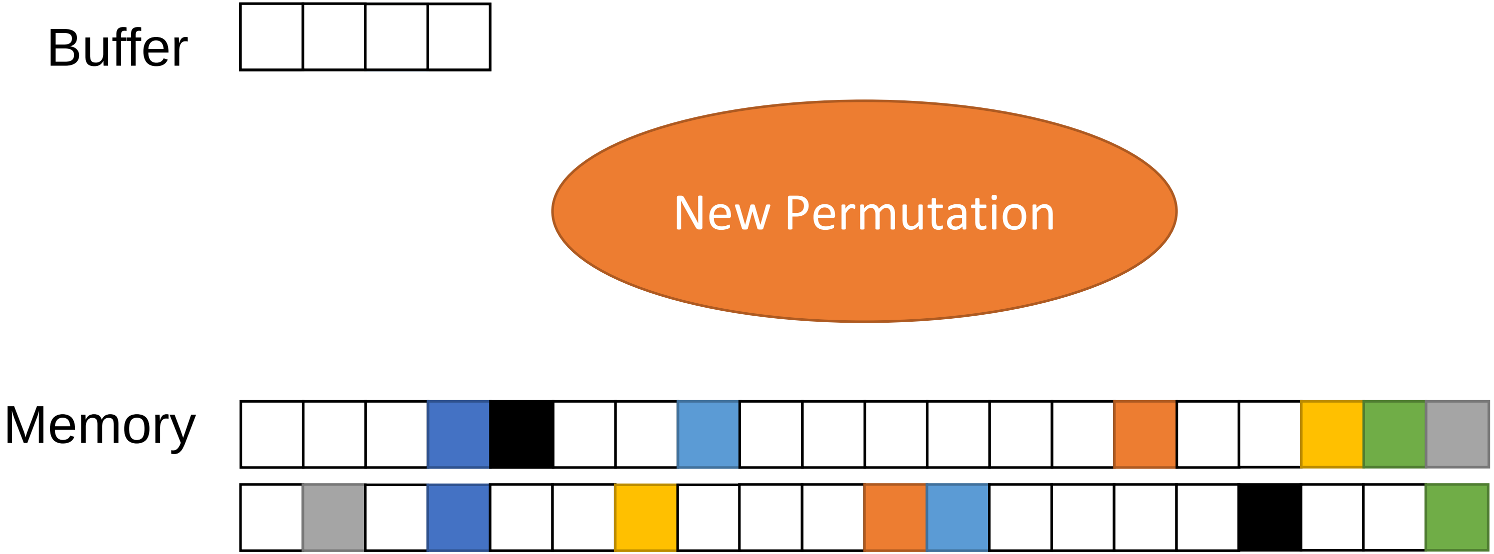
Scan Buffer
Random addr.

Memory

$\pi(1)$ $\pi(3)$ random $\pi(2)$

Warmup: Permute and Buffer

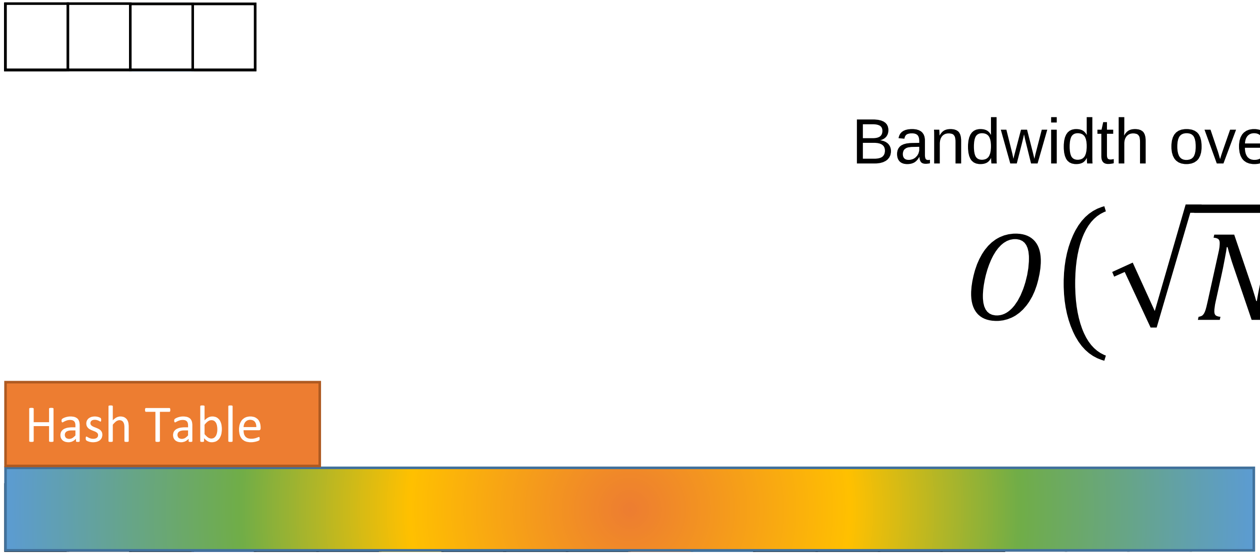
[Goldreich and Ostrovsky]



Warmup: *Hash* and Buffer

[Goldreich and Ostrovsky]

Buffer



Build

Lookup

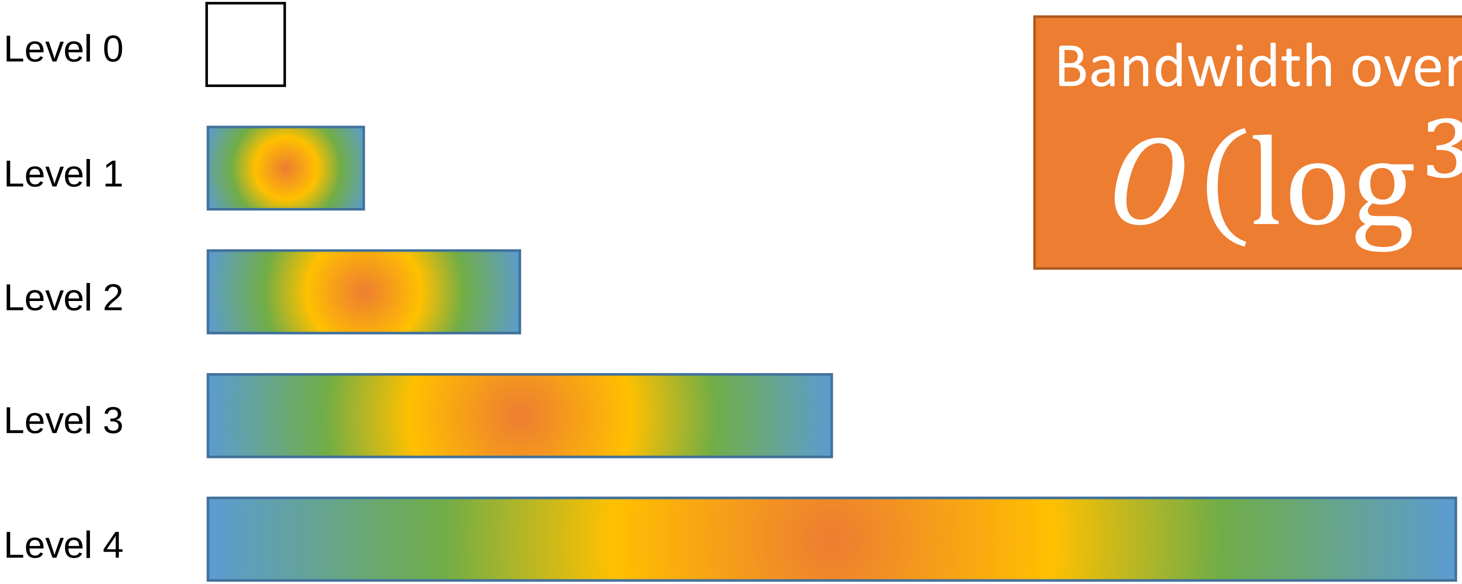
Bandwidth overhead:

$$O(\sqrt{N})$$

Hierarchical ORAM

[Goldreich and Ostrovsky]

- Recursive “buffer” of next level



Bandwidth overhead:

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

Use Cuckoo Hash

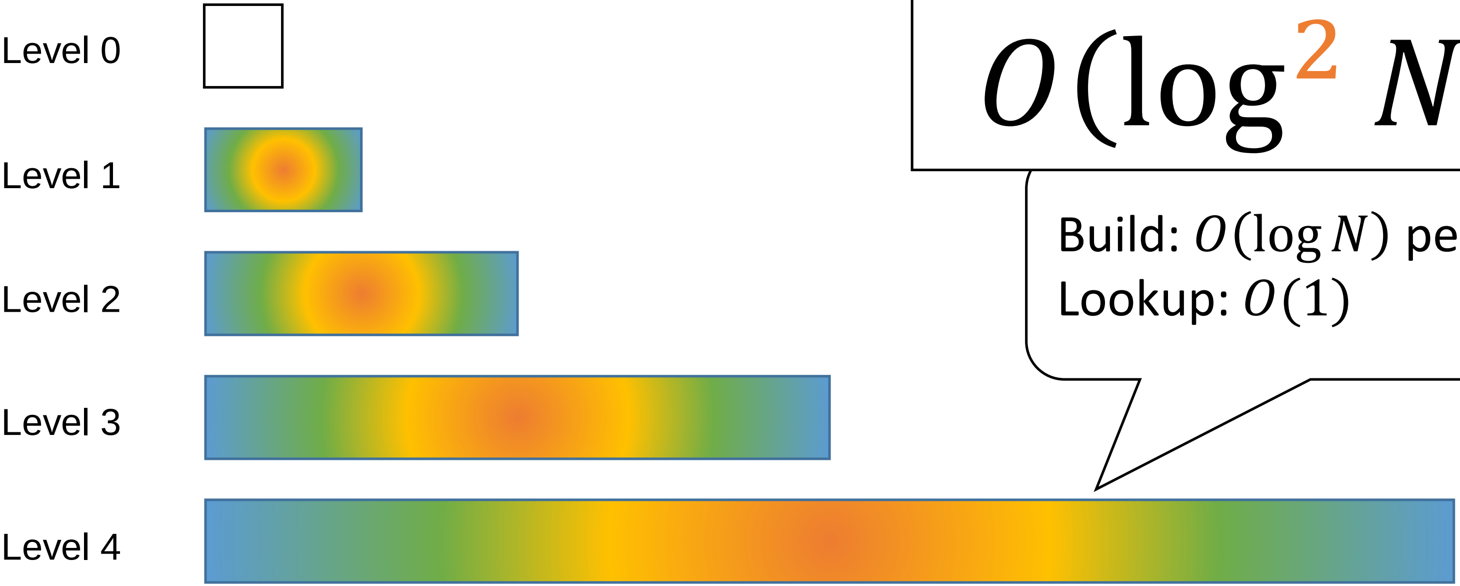
[Goodrich and Mitzenmacher]

- Faster hash table

Bandwidth overhead:

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

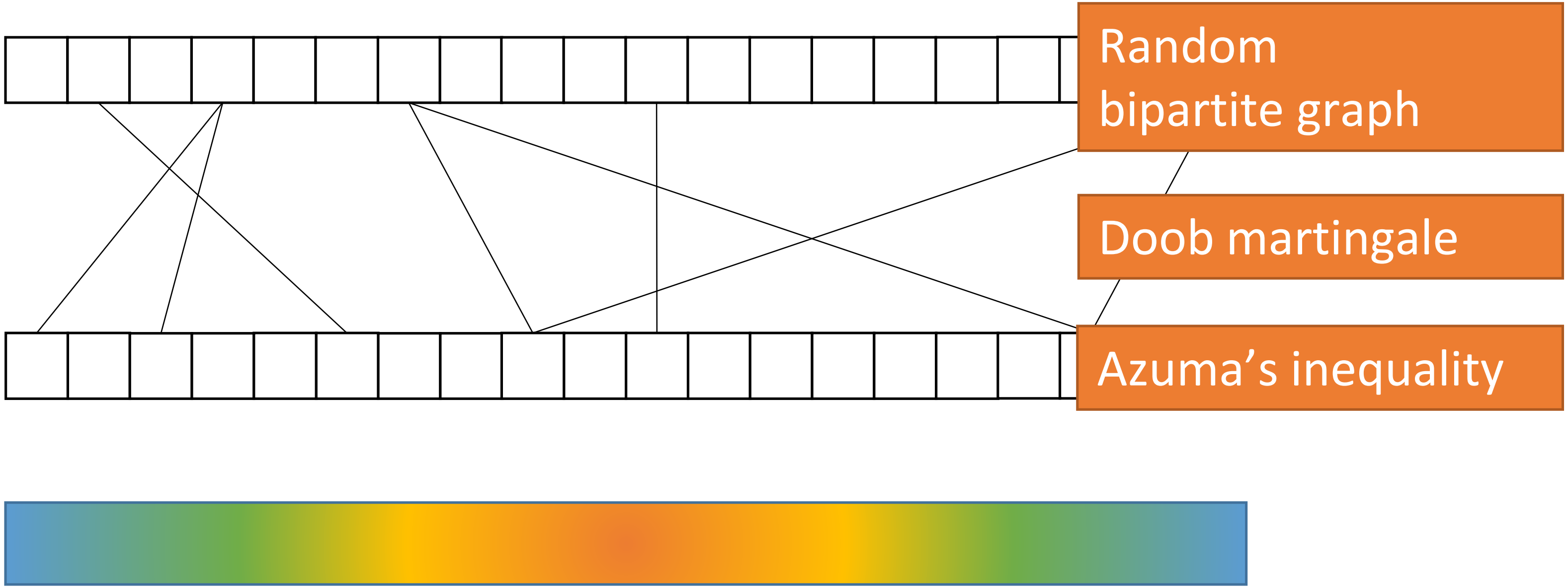
Build: $O(\log N)$ per element
Lookup: $O(1)$



Cuckoo Hash is Involved

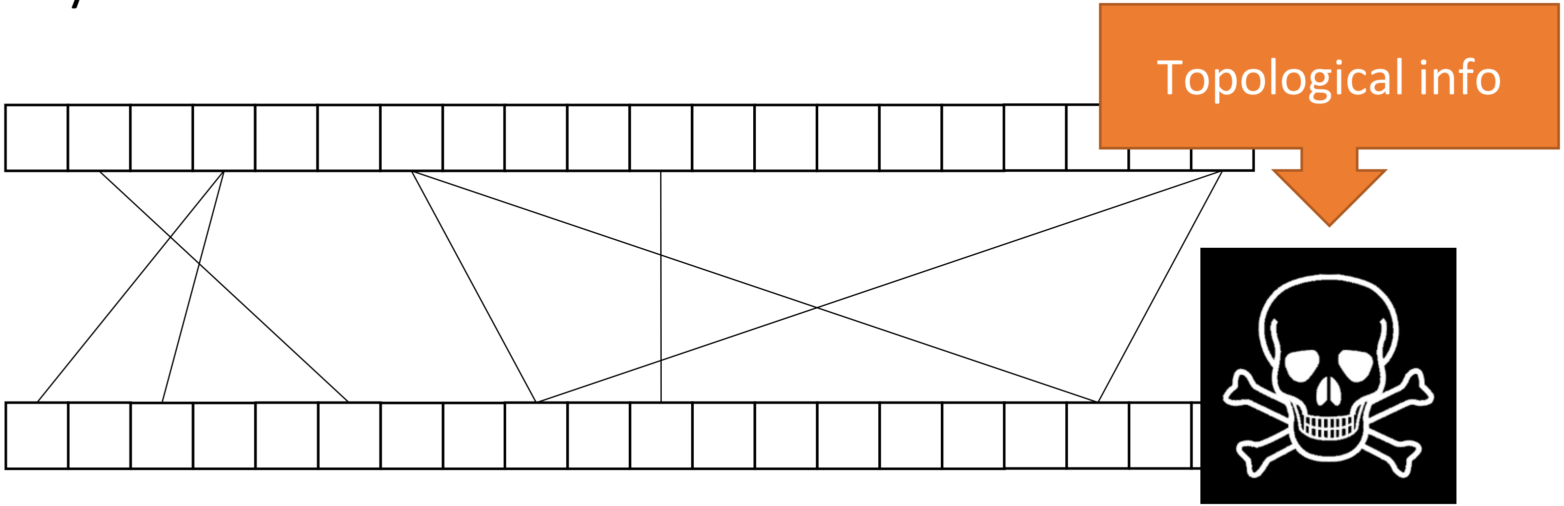
[Goodrich and Mitzenmacher]

- Especially make it oblivious



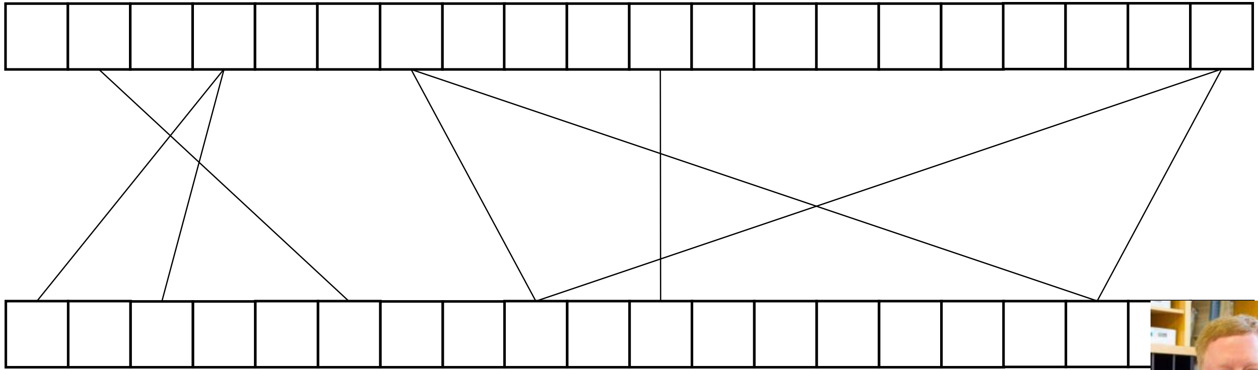
Cuckoo Hash is Involved

- Especially make it oblivious



Cuckoo Hash is Involved

- Especially make it oblivious



Refine algorithm

Var. martingale

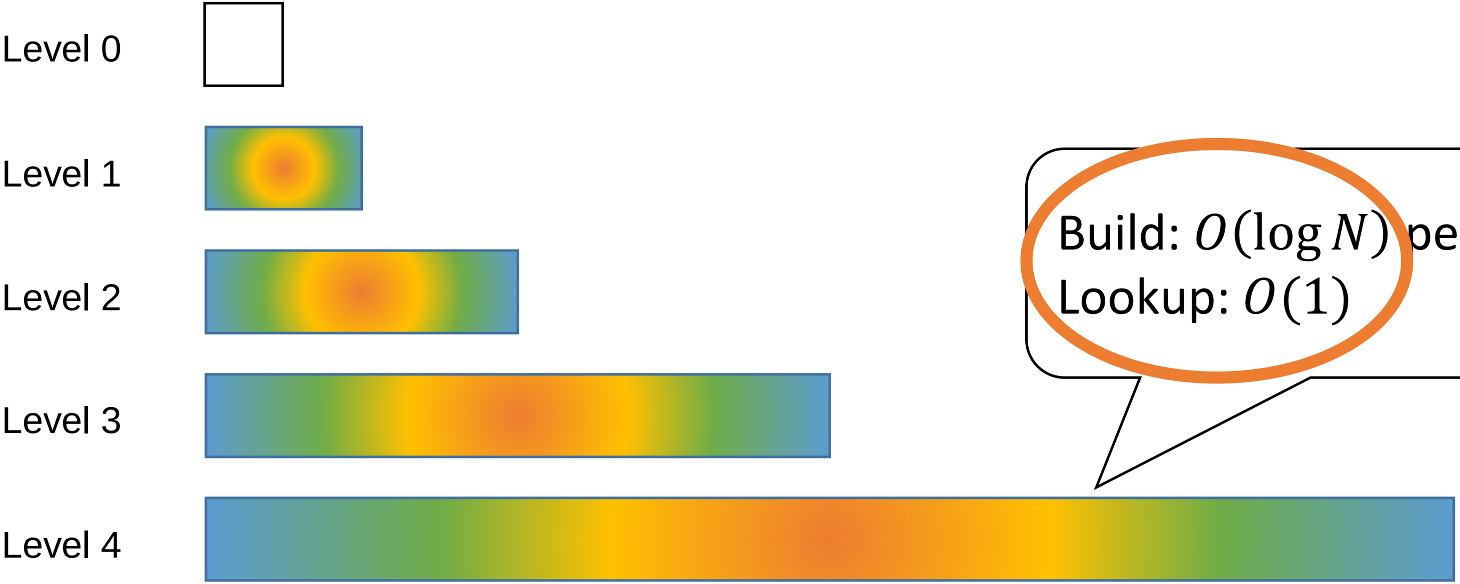
Var. Azume’s ineq.

[Goodrich and Mitzenmacher]

Use Cuckoo Hash

[Goodrich and Mitzenmacher]

- Faster hash table



Build: $O(\log N)$ per element
Lookup: $O(1)$

[Kushilevitz, Lu, and Ostrovsky]

Re-parameterize Hierarchy

- Reduce num. levels

Each level
$\mu = \log N$ hash tables

Level i, capacity μ^i

$\log_\mu N$ levels

Bandwidth overhead:
$$O\left(\frac{\log^2 N}{\log \log N}\right)$$

Level 0

Level 1

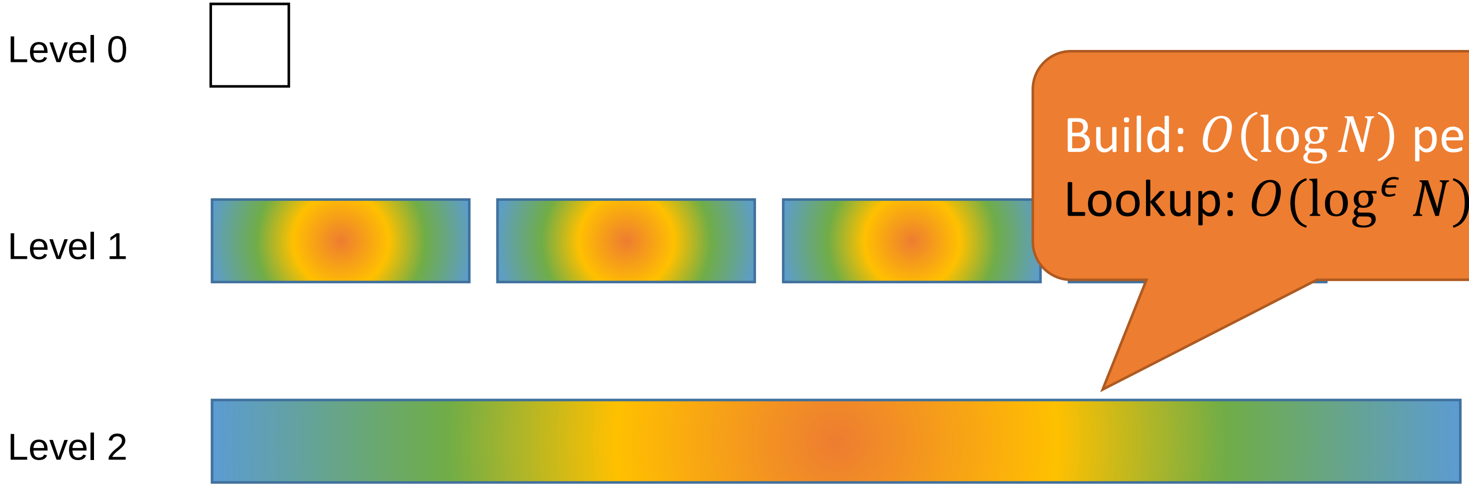
Level 2

Hash Table We Need

Each level
$\mu = \log^{1-\epsilon} N$ hash tables

Cuckoo hash is an overkill!

Build: $O(\log N)$ per element
Lookup: $O(\log^{\epsilon} N)$



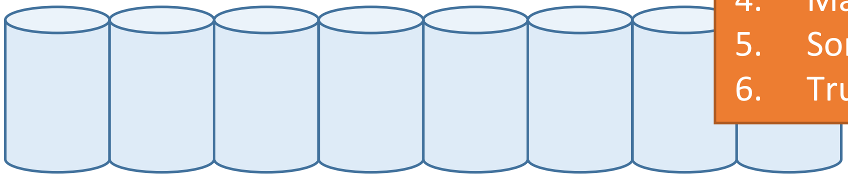
Recall: Balls-and-Bins (1-tier) Hash Table

- Standard hash table
- Oblivious: sort and scan

Not efficient enough

Secure implementation [GO96]
1. Add hash values
2. Add dummy words
3. Sort by hash values using AKS
4. Mark duplicates
5. Sort by duplicates
6. Truncate

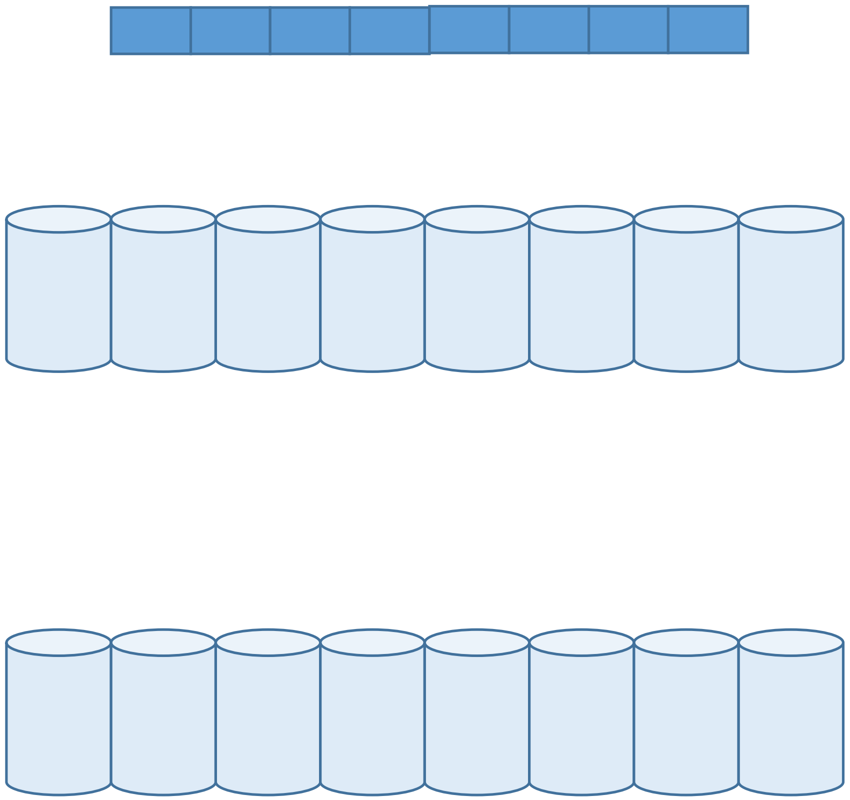
Bucket size: Z words



Num. of buckets B

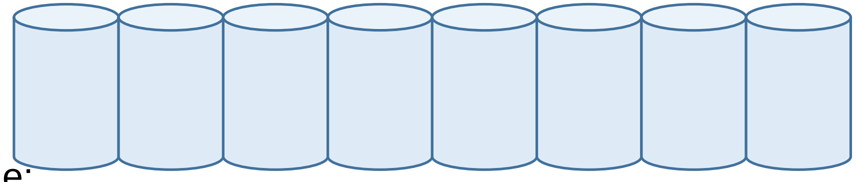
Two-Tier Hash Table

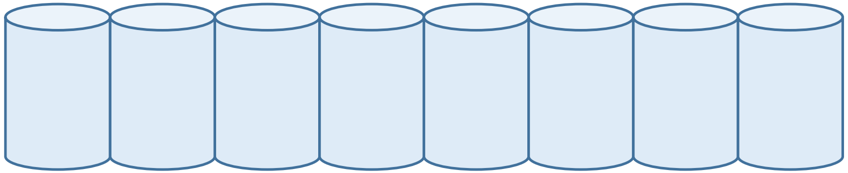
- Repeats standard hash twice
- Diff. B and Z



Parameters

- To store n elements, choose
  - $Z = 5 \log^{\epsilon} \lambda$
  - $B = \frac{n}{\log^{\epsilon} \lambda}$
  - $\epsilon \in (0.5, 1)$ is a constant



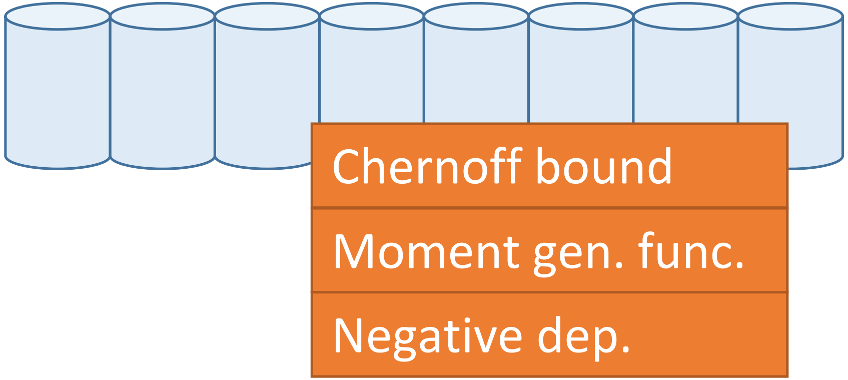


Overflow Probability

Theorem (1st tier):
If $n \geq 3\exp(\log^{\epsilon} N)$,
then total overflow is at most

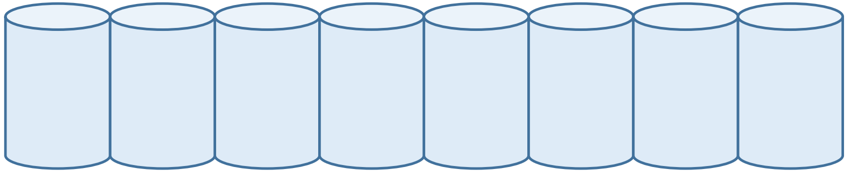
$$k' = 288B\exp\left(\frac{-Z}{6}\right)$$

Except with negl. prob. $\boxed{o(n^{1-\alpha})}$



Theorem (2nd tier):
If $n \geq 3\exp(\log^{\epsilon} N)$,
then no overflow

Except with negl. prob.



Simple Hierarchical ORAM

Each level $\mu = \log^{1-\epsilon} N$ hash tables

Two-tier hash

Build: $O(\log N)$ per element

Lookup: $O(\log^{\epsilon} N)$

Bandwidth overhead:

$$O\left(\frac{\log^2 N}{\log\log N}\right)$$

Level 0

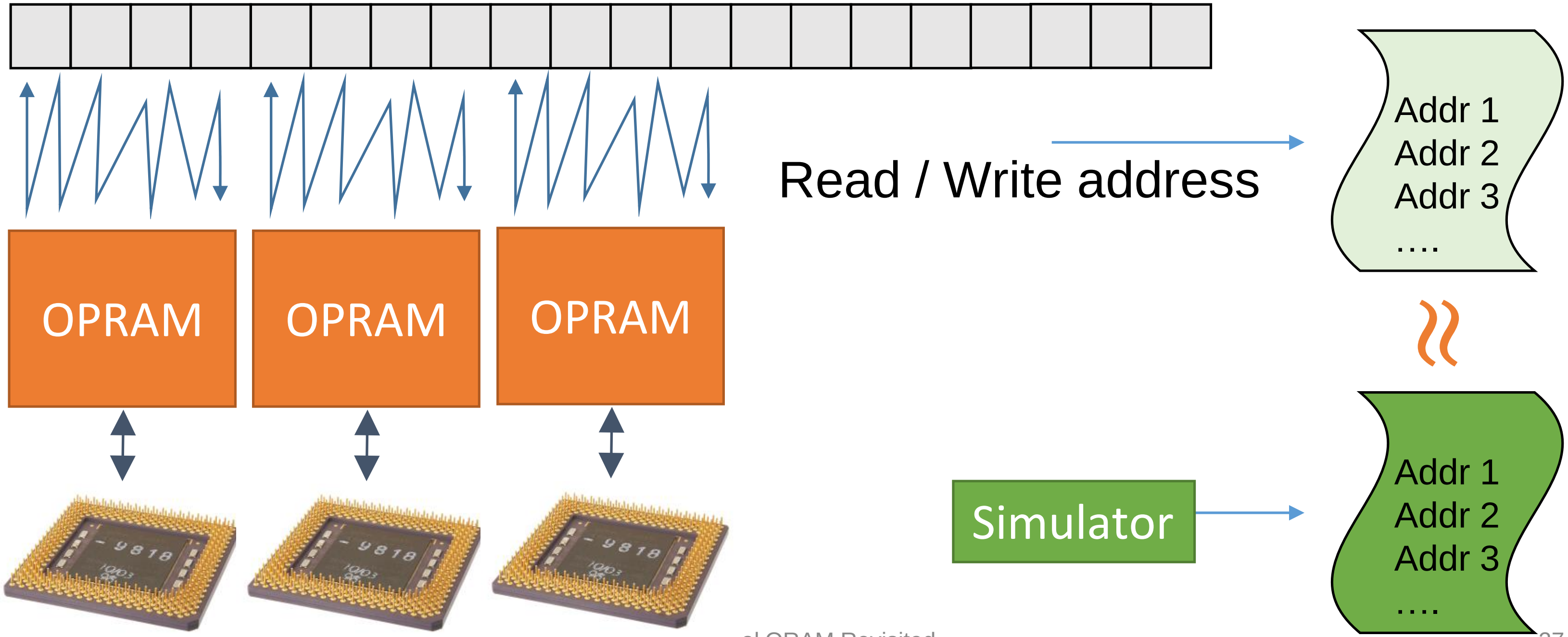
Level 1

Level 2

Parallel: Oblivious PRAM

[Boyle, Chung, and Pass]

- Provable security ☺



Improved OPRAM

Two-tier hash table is easy to parallelize

Just sort and scan

Previous result

Work & parallel time:

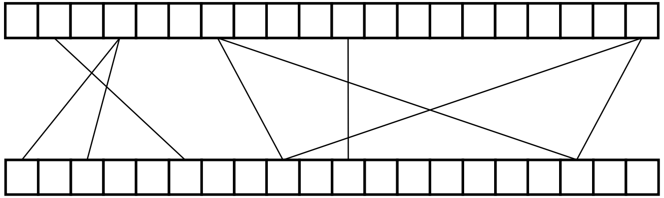
$$O\left(\frac{\log^2 N}{\log\log N}\right)$$

Level 0

Level 1

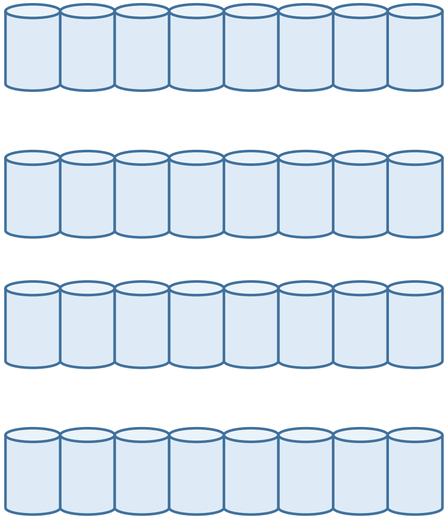
Level 2

Conclusions



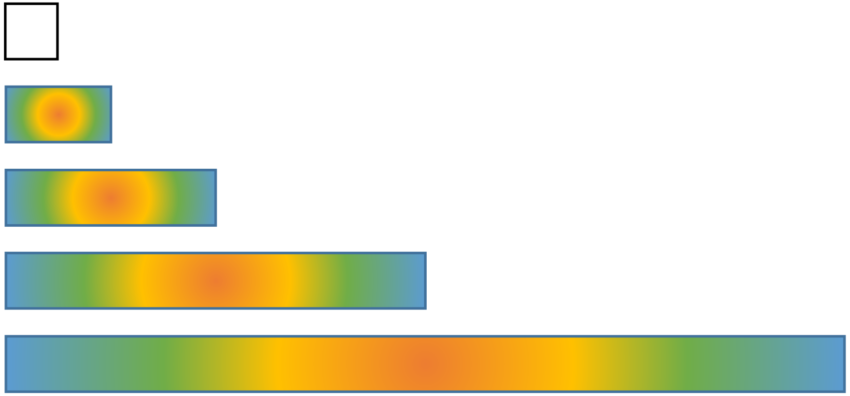
ORAM

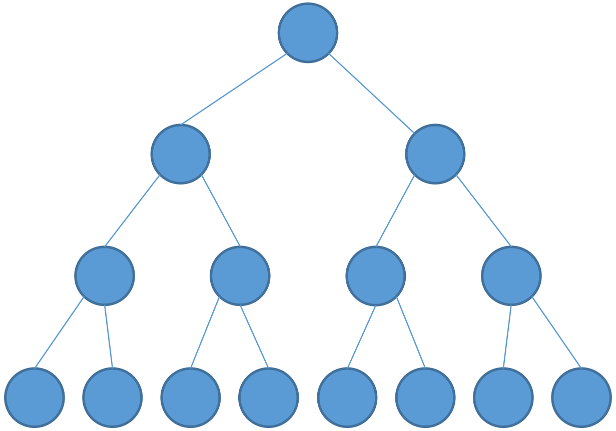
OPRAM



Followup: Cache-Efficiency

Contiguous memory





Thank you!

Questions?

wklin@cs.cornell.edu