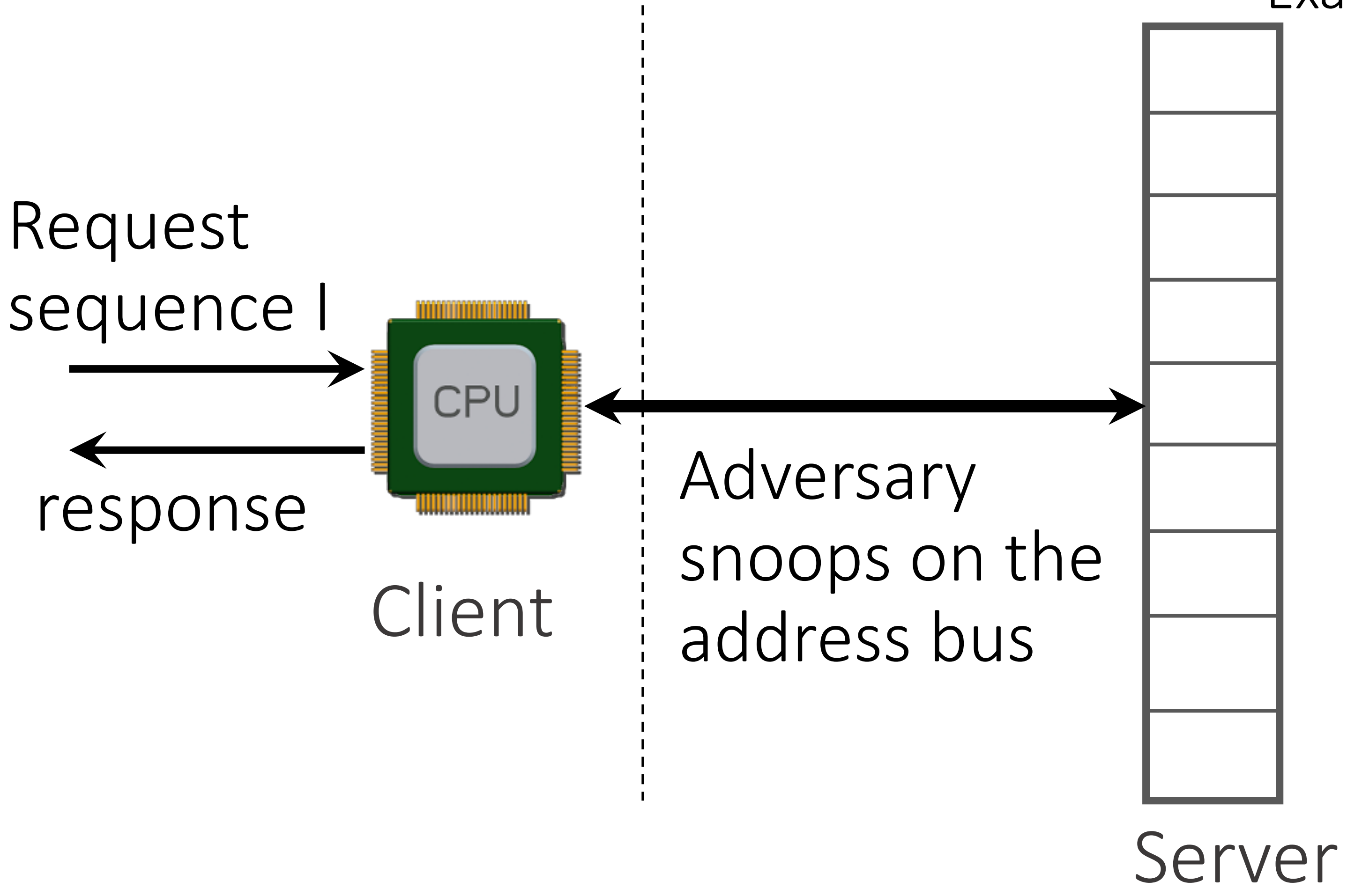
# Perfectly Secure Oblivious Algorithms in the Multi-Server Setting

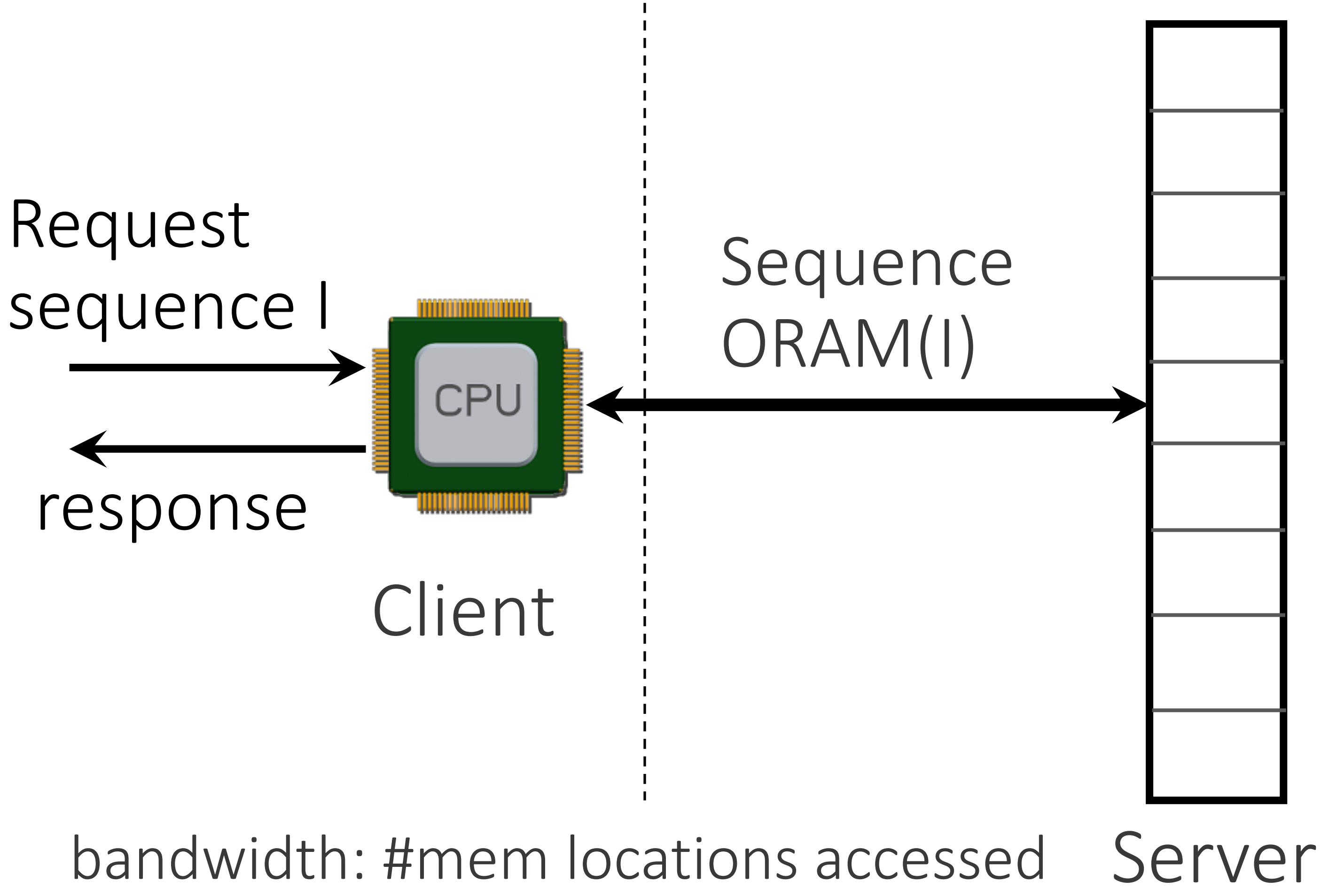
T-H. Hubert Chan, Jonathan Katz, **Kartik Nayak,**
Antigoni Polychroniadou, Elaine Shi

# Defining an Oblivious RAM

Example request sequence I: Read(a1),
Write(a2, d'),
Read(a3)
...



# Defining an Oblivious RAM




- Adversary (server) is semi-honest
- No server computation

Security: for I and I' of the same length,

ORAM(I) ~ ORAM(I')

bandwidth: #mem locations accessed by ORAM(I) for every access

# ORAM(I) ~ ORAM(I')

ORAM(I) ~ ORAM(I') $\xrightarrow{\text{typically}}$ Computationally indistinguishable or Statistically indistinguishable

Statistically indistinguishable: Adversary cannot distinguish with probability > negl(N) $\xrightarrow[N = poly(\lambda)]{}$ negl($\lambda$)

If N = polylog($\lambda$) $\longrightarrow$ negl(N) ≠ negl($\lambda$)

Achieving negl($\lambda$) difference using existing schemes is inefficient;

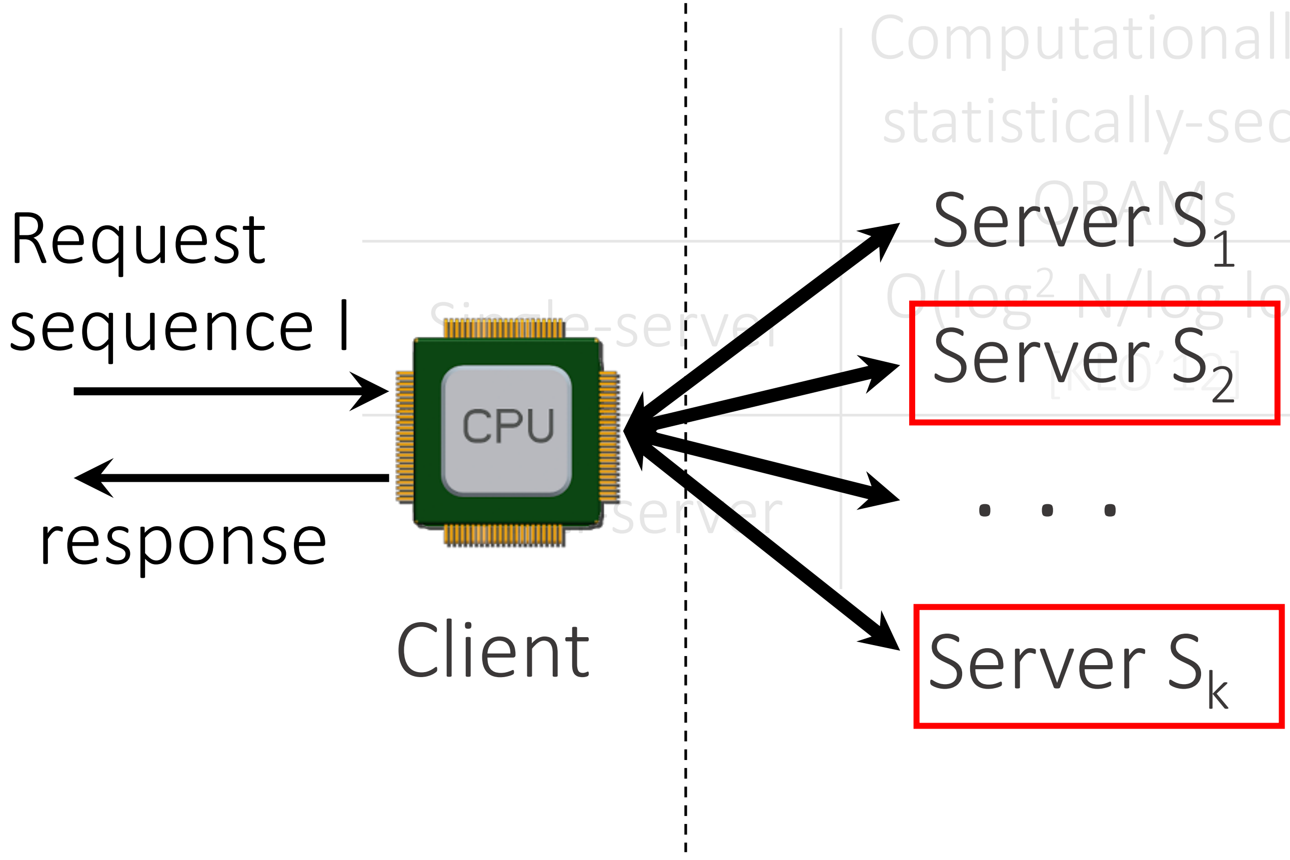
bandwidth of $N^c$, $c < 1$

# Perfectly-Secure ORAM

ORAM(I) ~ ORAM(I') $\longrightarrow$ Identically distributed

Existing perfectly-secure ORAMs: Bandwidth O(log$^3$ N) [DMN'11, CNS'18]

# Oblivious RAMs: Bandwidth Trade-offs




view$^{Adv}$: denotes what the adversary can observe from the semi-honest corrupt servers

Security:
for I and I' of the same length, view$^{Adv}$(I) and view$^{Adv}$(I') are identically distributed

# Oblivious RAMs: Bandwidth Trade-offs

| | Computationally or statistically-secure ORAMs | Perfectly-secure ORAMs |
|---|---|---|
| Single-server | $O(\log^2 N/\log \log N)$ [KLO'12] | $O(\log^3 N)$ [DMN'11, CNS'18] |
| Multi-server | $O(\log N)$ [LO'13] | $O(\log^2 N)$ [This paper] |

1. Multi-server ORAMs were only computationally or statistically secure

2. Are there inherent advantages in the multi-server setting?

# Oblivious RAMs: Bandwidth Trade-offs

O(log N) [AKLNS'18]

| | Computationally or statistically-secure ORAMs | Perfectly-secure ORAMs |
|---|---|---|
| Single-server | ~~$O(\log^2 N/\log\log N)$~~ ~~[KLO'12]~~ | $O(\log^3 N)$ [DMN'11, CNS'18] |
| Multi-server | $O(\log N)$ [LO'13] | $O(\log^2 N)$ [This paper] |

1. Multi-server ORAMs were only computationally or statistically secure

2. Are there inherent advantages in the multi-server setting?

# Our Results

There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to perform

1 Oblivious stable compaction and merging with O(N) bandwidth

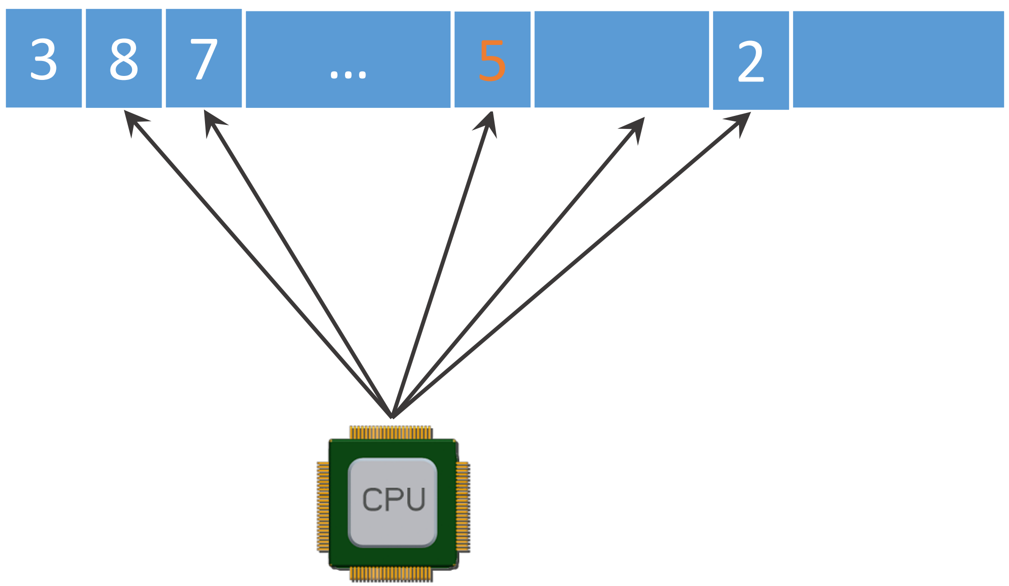
Lower bound: Single-server oblivious stable compaction and merging requires Ω(N log N) bandwidth in the balls-and-bins model [LSX’18]

# Our Results

There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to achieve

1. Oblivious stable compaction and merging with $O(N)$ bandwidth
2. ORAM scheme with $O(\log^2 N)$ bandwidth

# Oblivious Sort Incurs O(N log N) Bandwidth




Typically, shuffle is performed using oblivious sort

# Key Idea:
# Replace Oblivious Sort With Linear Time Operations

# Permutation-Storage-Separation Paradigm

Permute Server

Storage Server


Assumption: Data encrypted using perfectly-secure encryption scheme

# Permutation-Storage-Separation Paradigm

Permute Server

Storage Server


Knows permutation
Fisher-Yates: O(N) bandwidth

Observes accesses
O(1) bandwidth
(assuming position is known)

Lu-Ostrovsky introduced this paradigm [LO'13]
- Built cuckoo hash tables + used PRFs to access data
- Computationally-secure

# O(N) Bandwidth Oblivious Sort?

Can we perform O(N) bandwidth oblivious sort using this paradigm?

- Not aware of a solution
- Comparison-based (non-oblivious) sorts incur O(N log N)

# Our Results

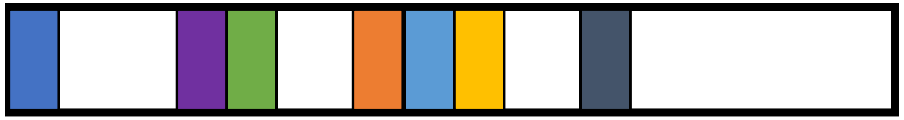
There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to achieve

1. Oblivious stable compaction and merging with $O(N)$ bandwidth

2. ORAM scheme with $O(\log^2 N)$ bandwidth

# Oblivious Tight Stable Compaction

Input: n elements, some real, some dummy



Output: n elements, all real elements at the beginning, order of real elements is preserved

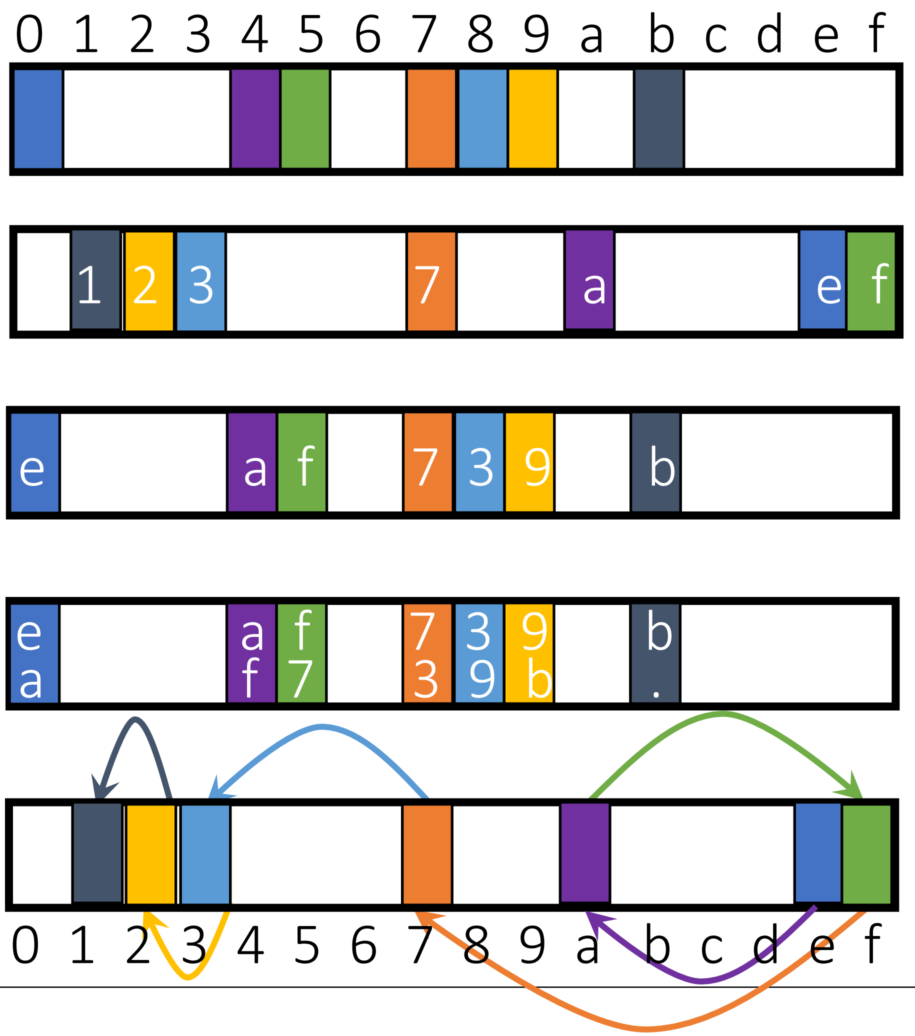
# Attempt 1: Oblivious Tight Stable Compaction


Protocol: Read block, if real, write to storage
Pad with dummies

Obliviousness: Each server observes a linear scan
Server 2 observes write time steps

# Oblivious Tight Stable Compaction

Server 1: Permute




*Remember head of linked-list

*Maintain a dummy linked-list too

Permute using $\pi$, determine destination

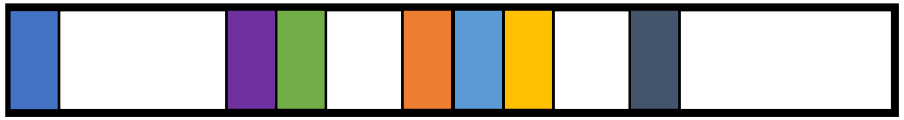
Inverse permute: $\pi^{-1}$

Reverse linear scan to create linked-list

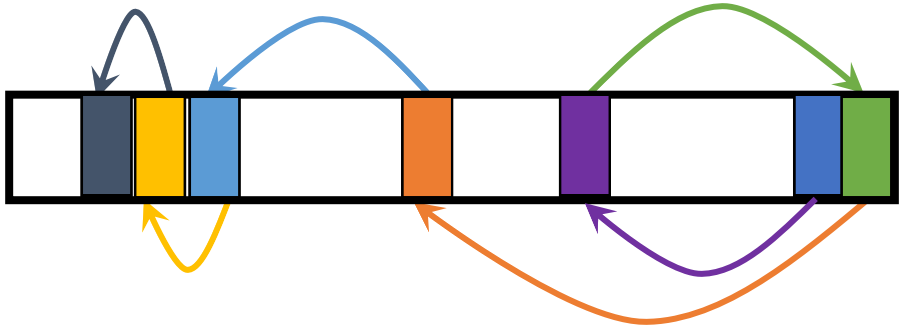
Permute using $\pi$ again

# Oblivious Tight Stable Compaction

Server 1: Permute



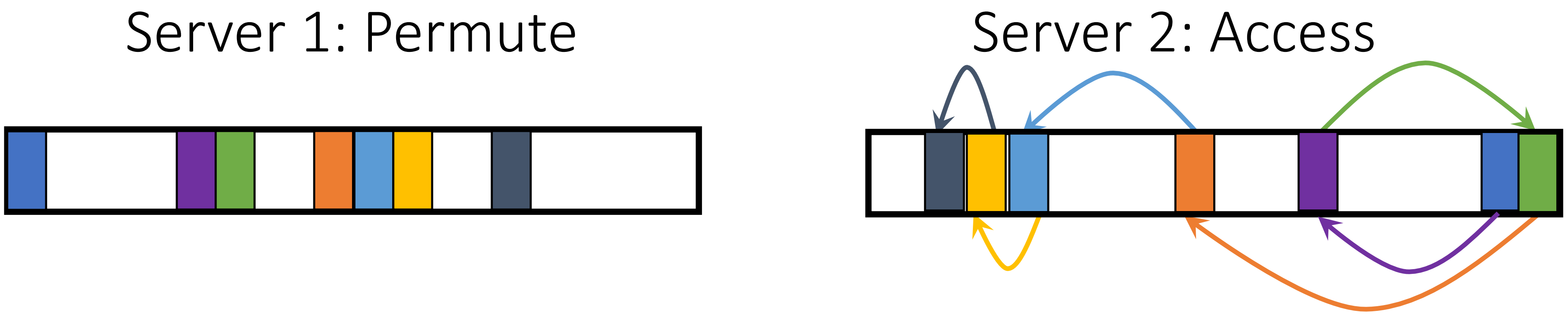
Server 2: Access



Protocol:

- Traverse real linked list followed by dummy linked list

# Oblivious Tight Stable Compaction




Security:

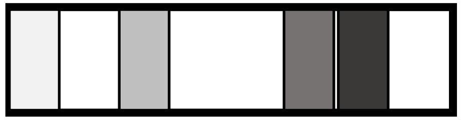
Server 1 permutes and performs linear scan. Does not observe accesses.

Server 2 observes accesses, does not know permutation

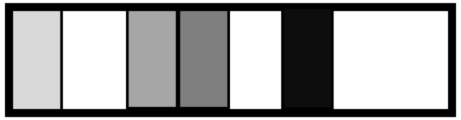
# Oblivious Merge

Input: $S_1$ and $S_2$ have semi-sorted lists with $n_1$ and $n_2$ elements resp.

Server $S_1$



Server $S_2$



Output: Sorted list of $n_1 + n_2$ elements on $S_1$

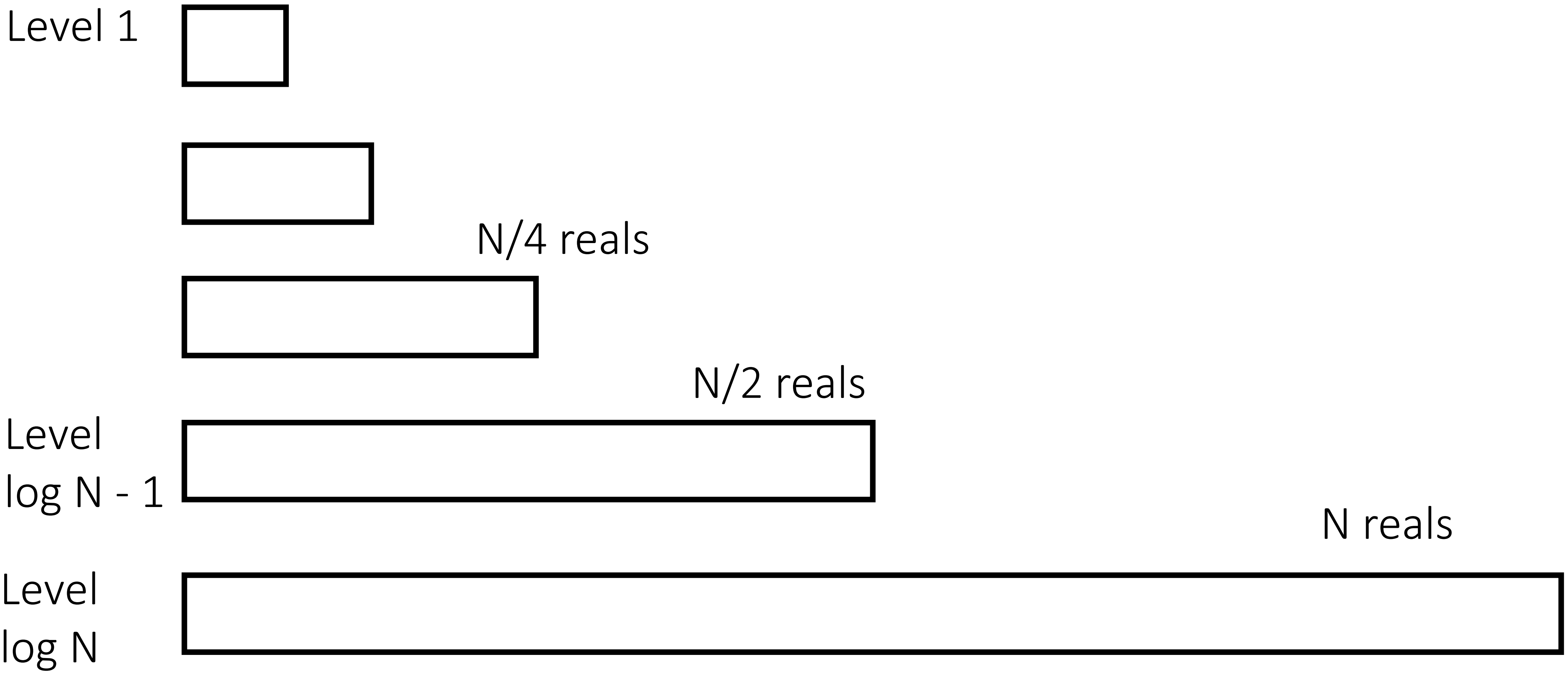
# Our Results

There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to achieve

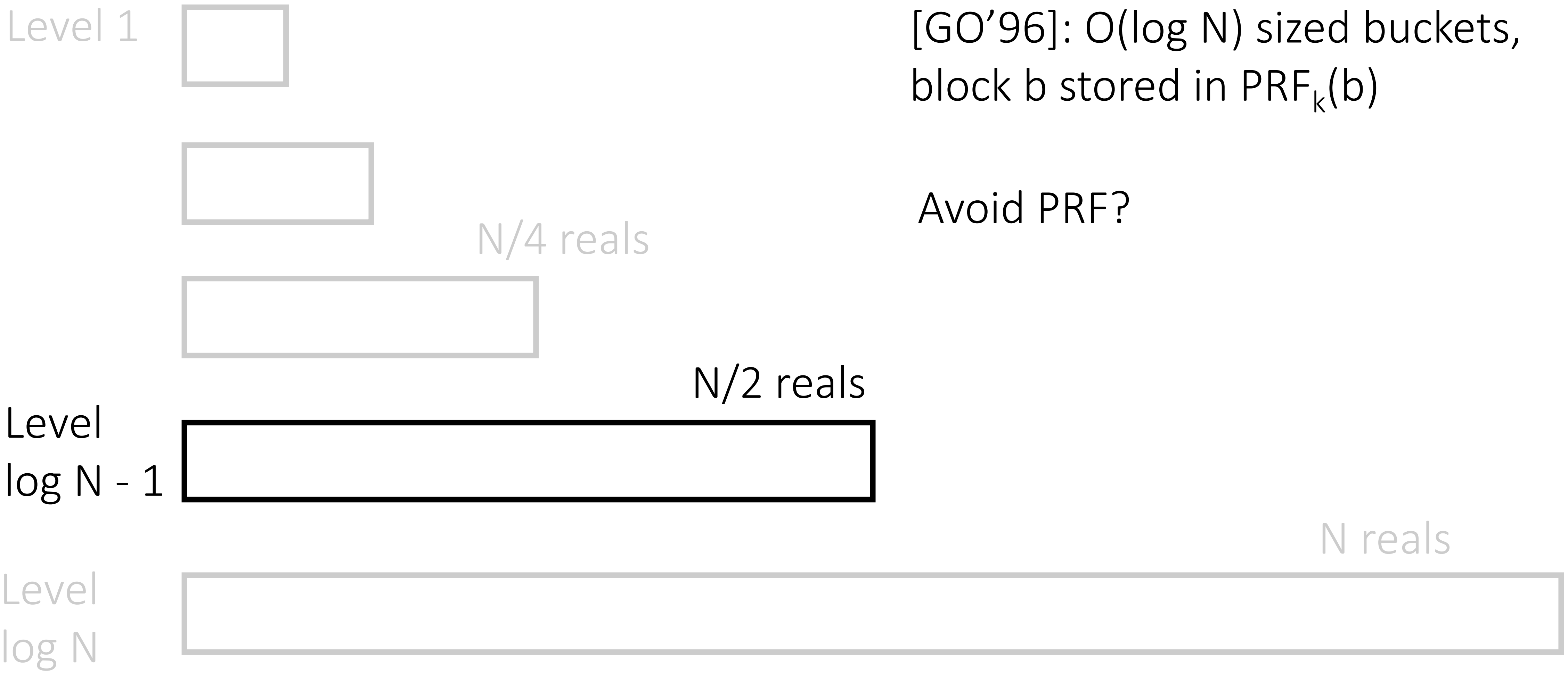
1. Oblivious stable compaction and merging with $O(N)$ bandwidth

2. ORAM scheme with $O(\log^2 N)$ bandwidth

# Hierarchical ORAM [GO'96]



# Hierarchical ORAM [GO'96]




[GO'96]: O(log N) sized buckets, block b stored in $PRF_k(b)$

Avoid PRF?

# Position-based Hierarchical ORAM [CNS'18]

Level 1

N/4 reals

N/2 reals

Level
log N - 1

N reals

Level
log N

Store blocks shuffled uniformly at random

Access a block:
- Is the block stored at this level?
- If yes, location?
- else, location of a dummy?

# Position-based Hierarchical ORAM [CNS'18]

Level 1

N/4 reals

Level log N - 1

N/2 reals

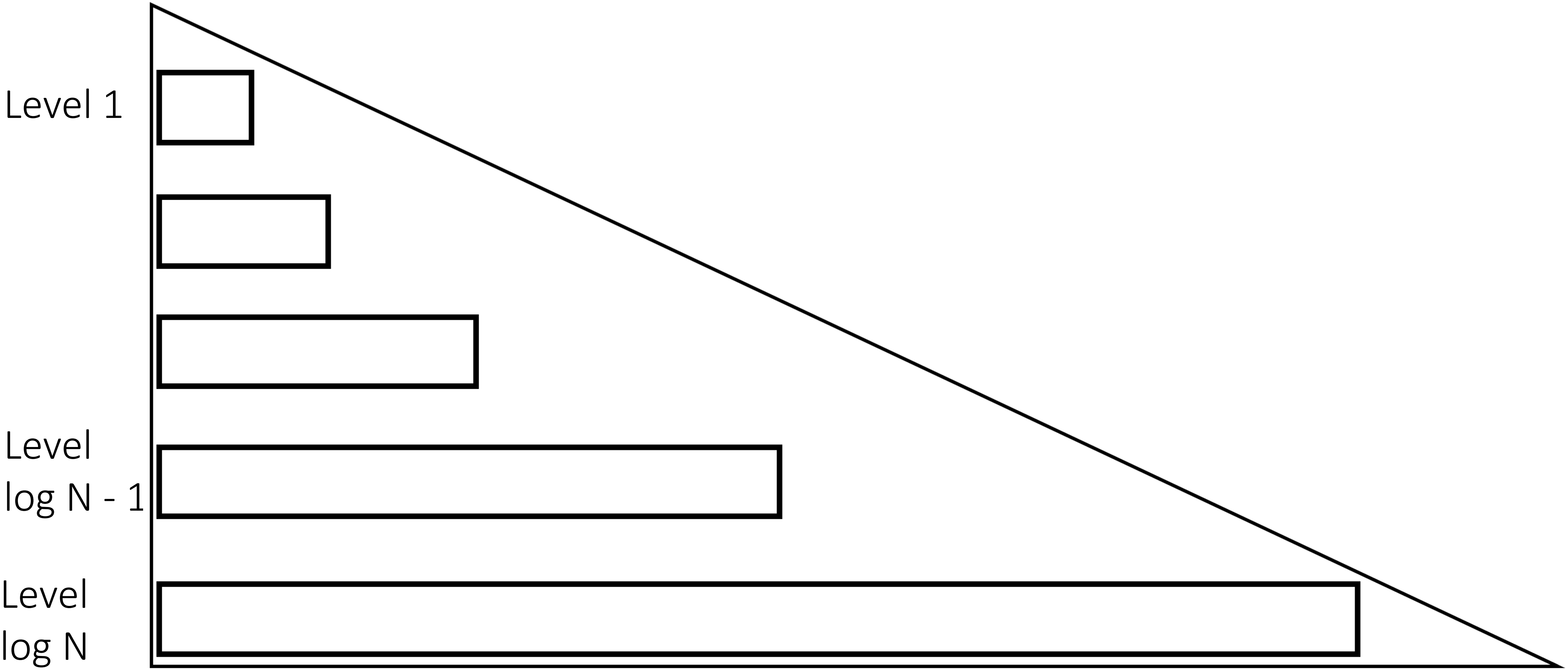
Level log N

N reals

For all levels,

- Is the block stored at this level?
- If yes, location?
- else, location of a dummy?

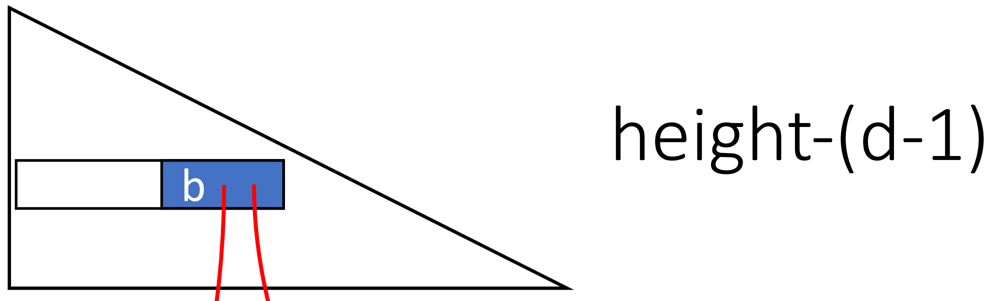
# Position-based Hierarchical ORAM [CNS'18]



# Recursive Position-based Hierarchical ORAM [CNS'18]

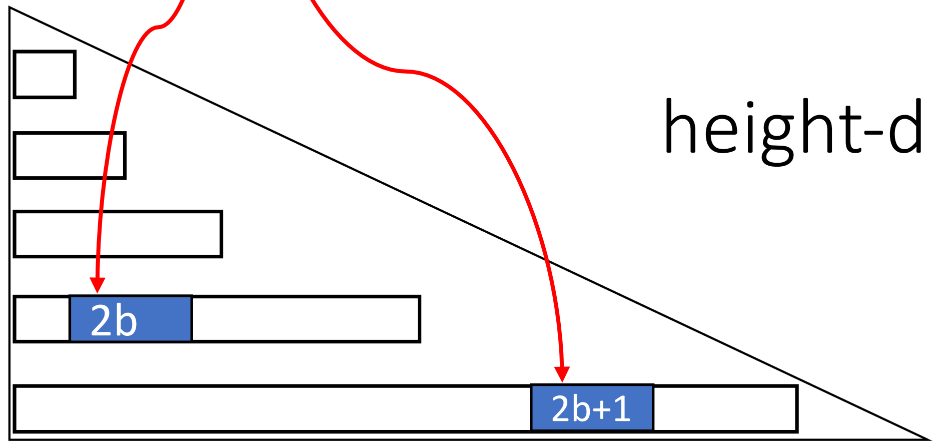
## Position-based ORAM at height-(d-1)

Block b at height-(d-1) stores the level and position of blocks 2b and 2b+1 at height-d




## Position-based ORAM at height-d

For all levels, positions of all blocks



# Recursive Position-based Hierarchical ORAM [CNS’18]

Position-based ORAM at depth-(d-1)

Block b at depth-(d-1) stores the level and position of blocks 2b and 2b+1 at depth-d

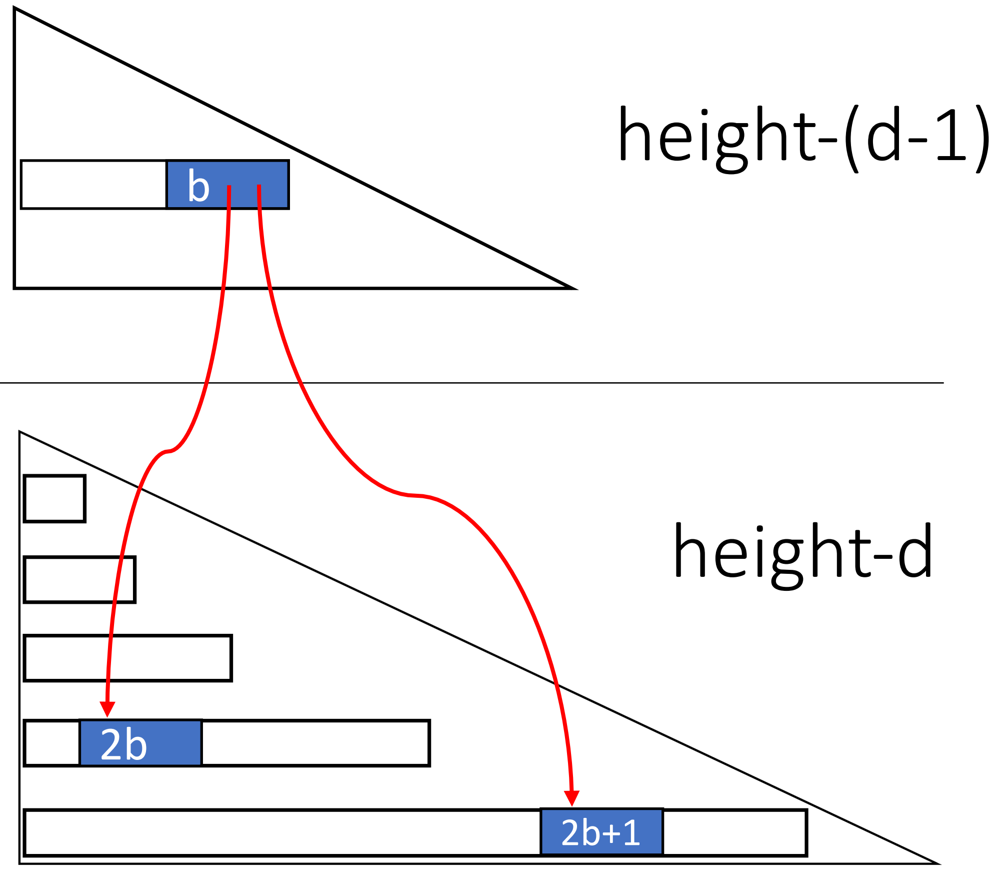
Position-based ORAM at depth-d

For all levels,

- Is the block stored at this level?
- If yes, location?
- Else, location of a dummy

Caveats:

1. Does not handle dummies
2. Cannot be used in a black-box manner



# Co-ordinated Reshuffle Across Hierarchies

Position-based ORAM at height-(d-1)

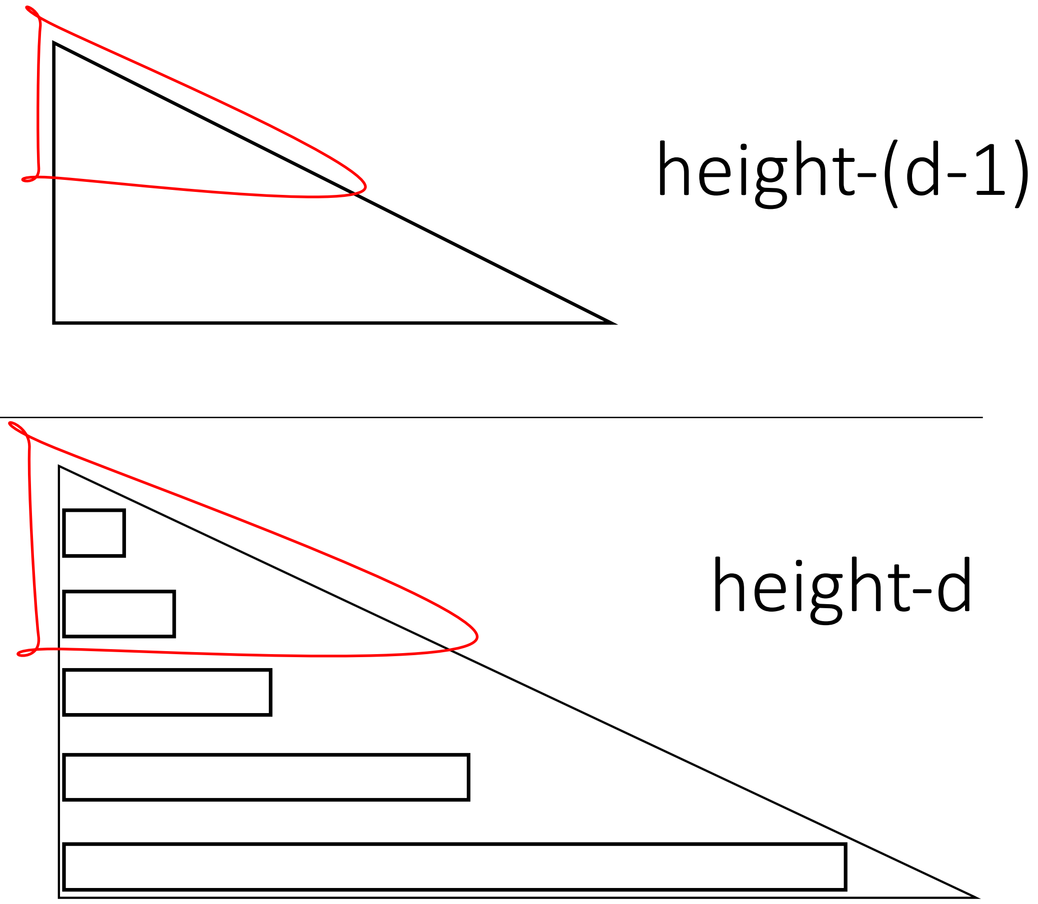
Block b at height-(d-1) stores the level and position of blocks 2b and 2b+1 at depth-d

**Co-ordinated reshuffle:**

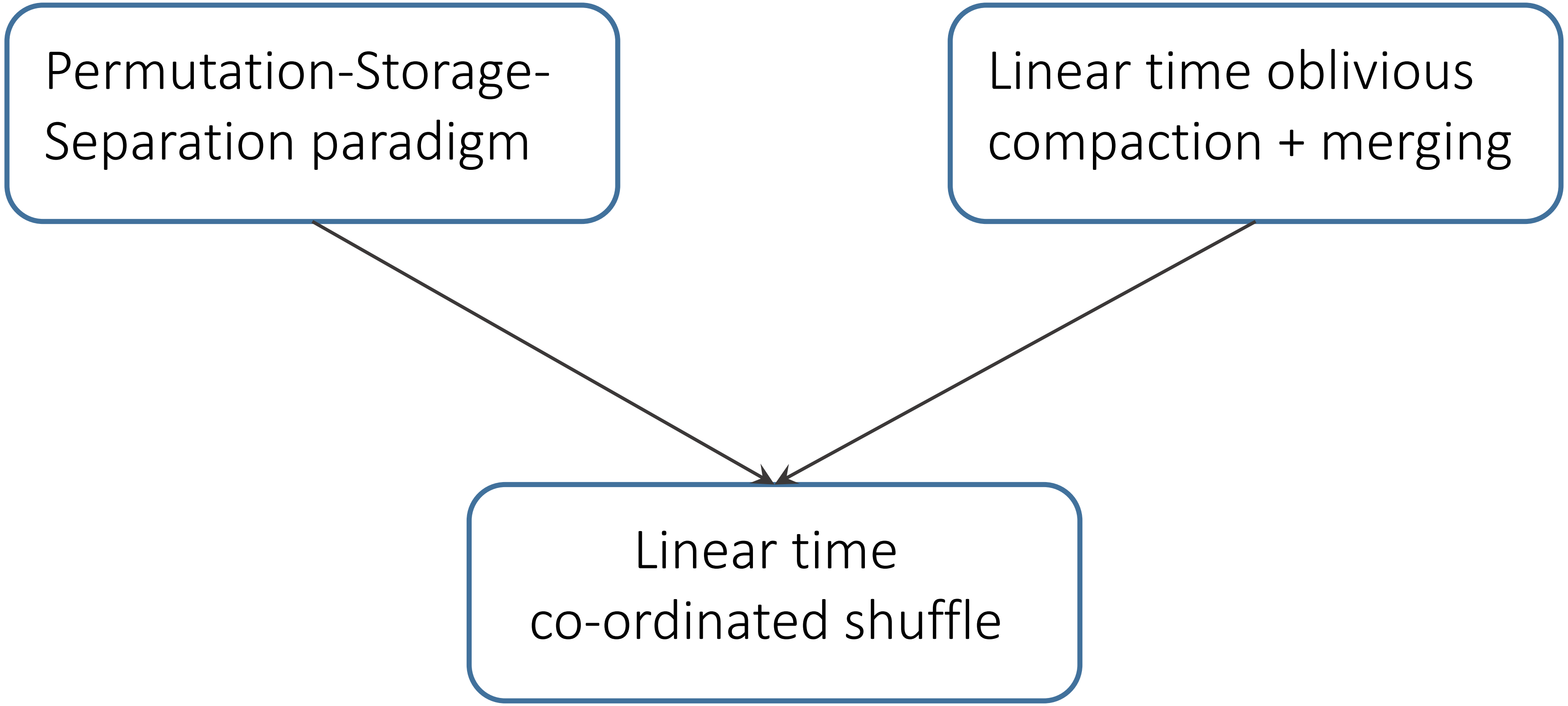
When level l at height-d is reshuffled, all levels ≤ l at height < d are reshuffled

Position-based ORAM at height-d

For all levels, positions of all blocks



# Co-ordinated Shuffle in the Multi-Server Setting



# Conclusion

- Oblivious stable compaction and merging can be performed with $O(N)$ bandwidth using 3 servers

- 3-server ORAM scheme with $O(\log^2 N)$ amortized bandwidth

Thank You!

kartik@cs.duke.edu