Fast Fully Oblivious Compaction and Shuffling

Sajin Sasy, Aaron Johnson, and Ian Goldberg

UNIVERSITY OF WATERLOO

U.S. NAVAL RESEARCH LABORATORY

CrySP

Cryptography, Security, and Privacy
— Research Group @ uWaterloo —

Background

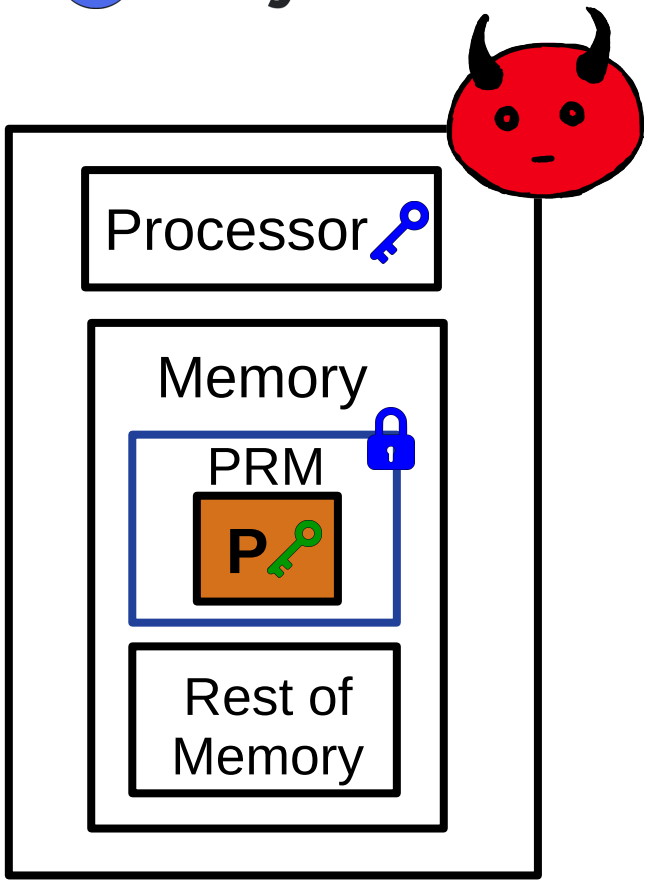
- TEEs enable *secure* execution of programs on a remote server; secure ⟶ confidentiality and integrity guarantees

AMD SEV-SNP

TrustZone® System Security by ARM

intel SGX

Keystone



Background

- TEEs enable *secure* execution of programs on a remote server; secure ⟶ confidentiality and integrity guarantees

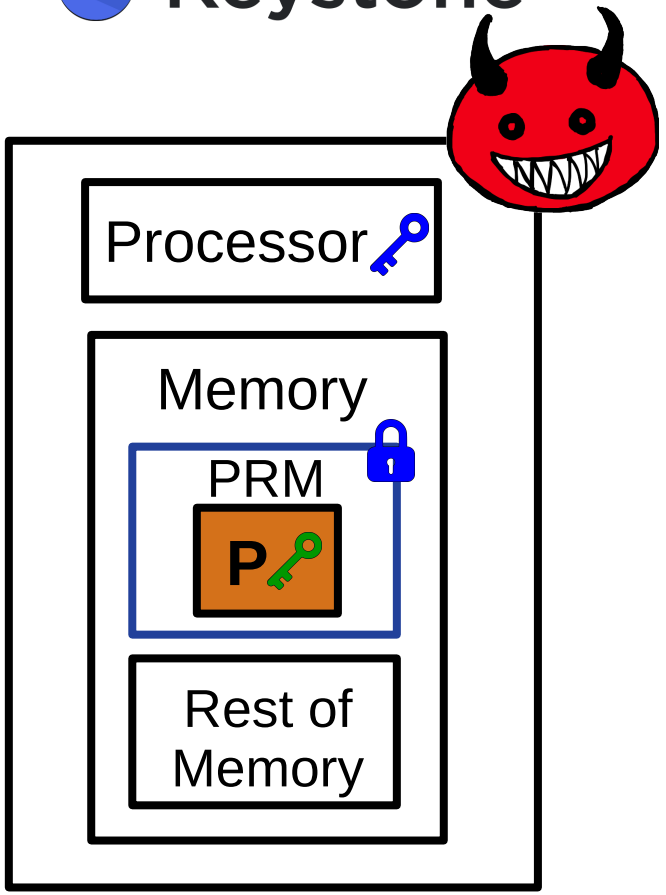
- **Vulnerable to side-channel attacks that violate confidentiality guarantees**

AMD SEV-SNP

TrustZone® System Security by ARM

intel SGX

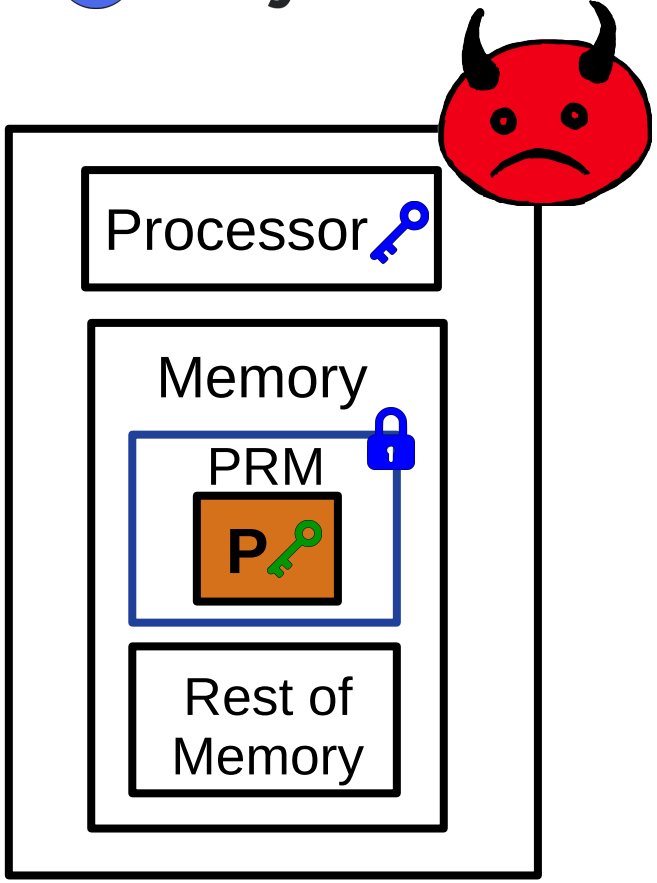
Keystone



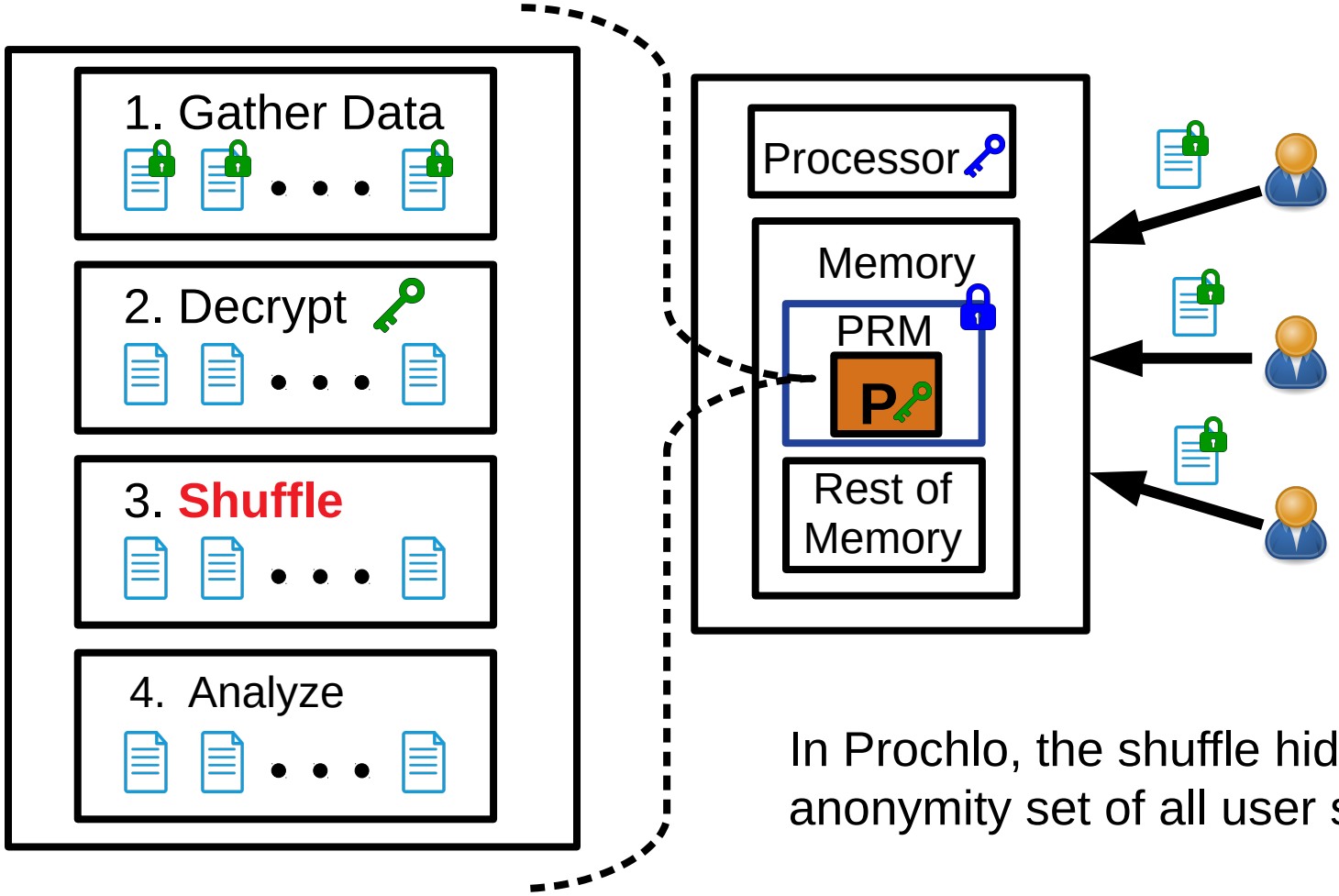
Background

- TEEs enable *secure* execution of programs on a remote server; secure ⟶ confidentiality and integrity guarantees

- **Vulnerable to side-channel attacks that violate confidentiality guarantees**
- **Our goal: Design algorithms that run in TEEs without being vulnerable to such side-channels**



TEE Application: Prochlo

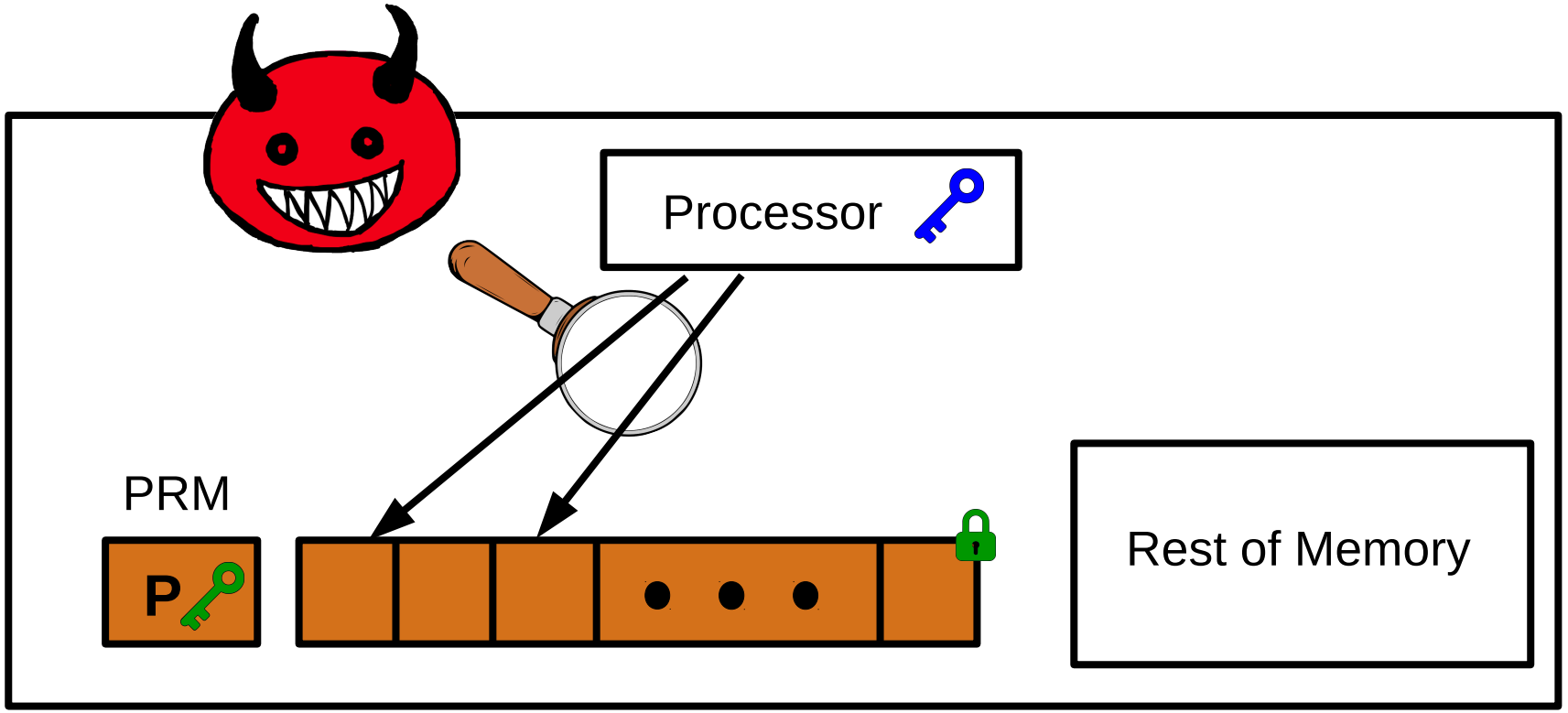


Prochlo [1] is designed for privacy-preserving telemetry, error reporting, and software profiling.

In Prochlo, the shuffle hides individual contributions within the anonymity set of all user submissions.

[1] - Bittau, A. et al., “Prochlo: Strong Privacy for Analytics in the Crowd". SOSP, 2017.

TEE side-channels



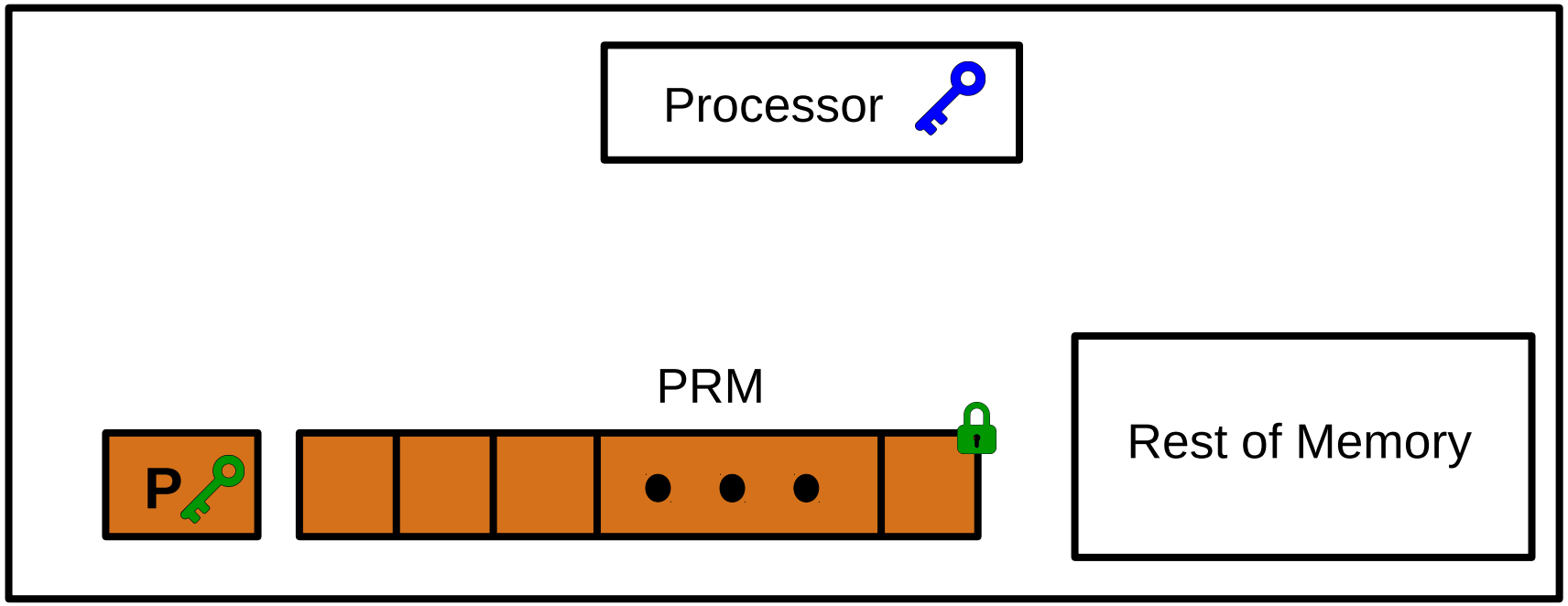
Observing memory access patterns trivially reveal the shuffle permutation

Our Contributions

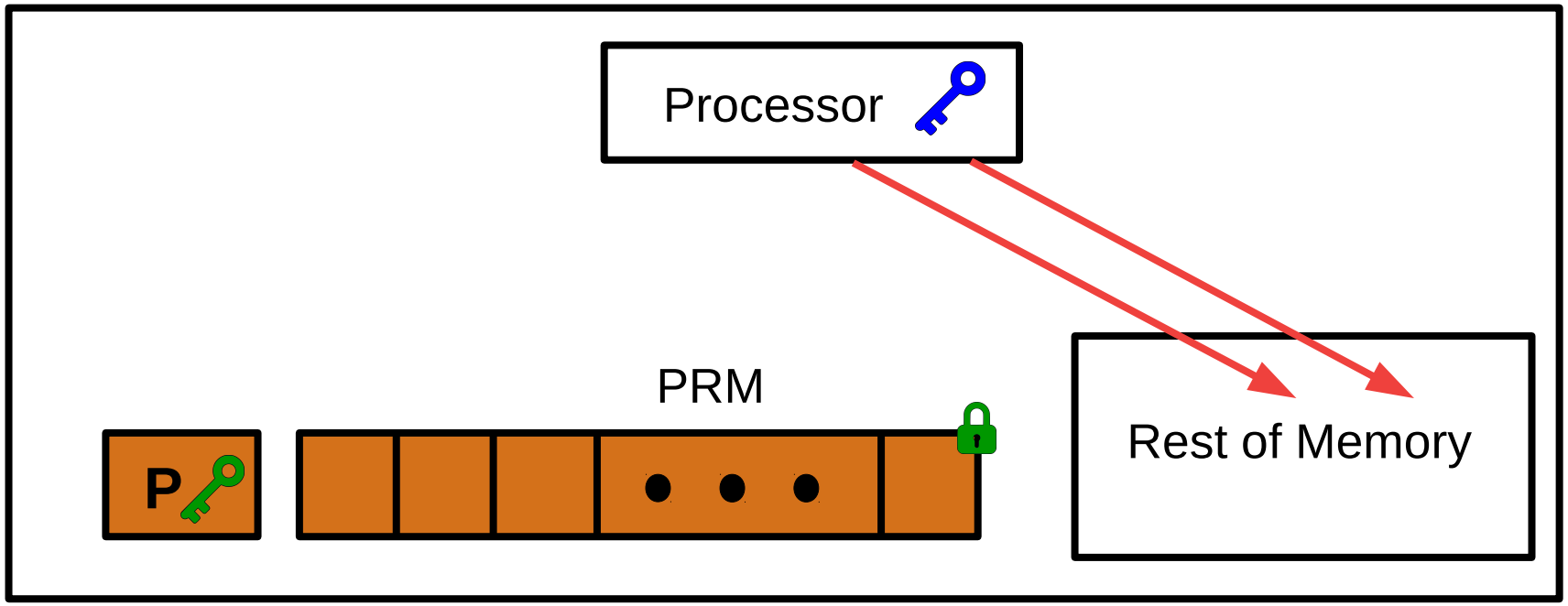
In this work, we present:

- *Fully Oblivious* Algorithms for:
  - Shuffling (**ORShuffle** and **BORPStream**)
  - Order-preserving Tight Compaction (**ORCompact**)
- **FOAV**: A tool to ensure that obliviousness persists in the final binary executed by the processor

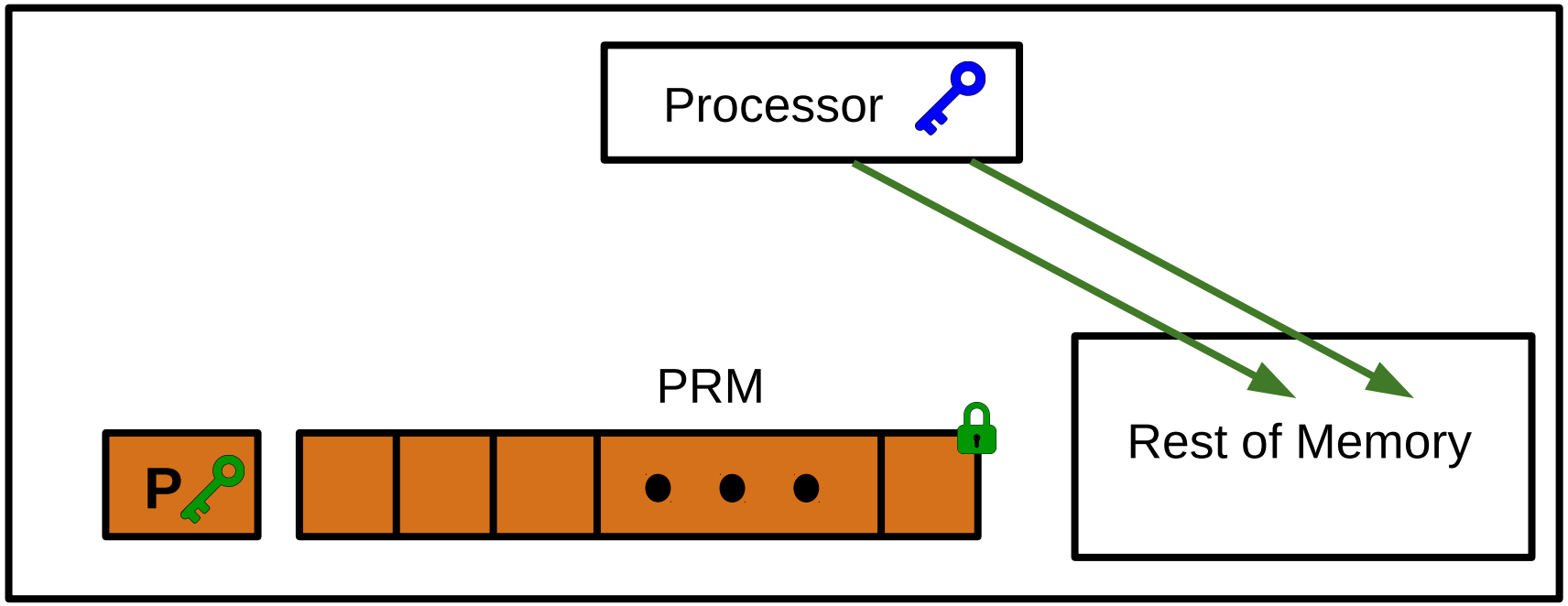
Levels of Obliviousness



Levels of Obliviousness



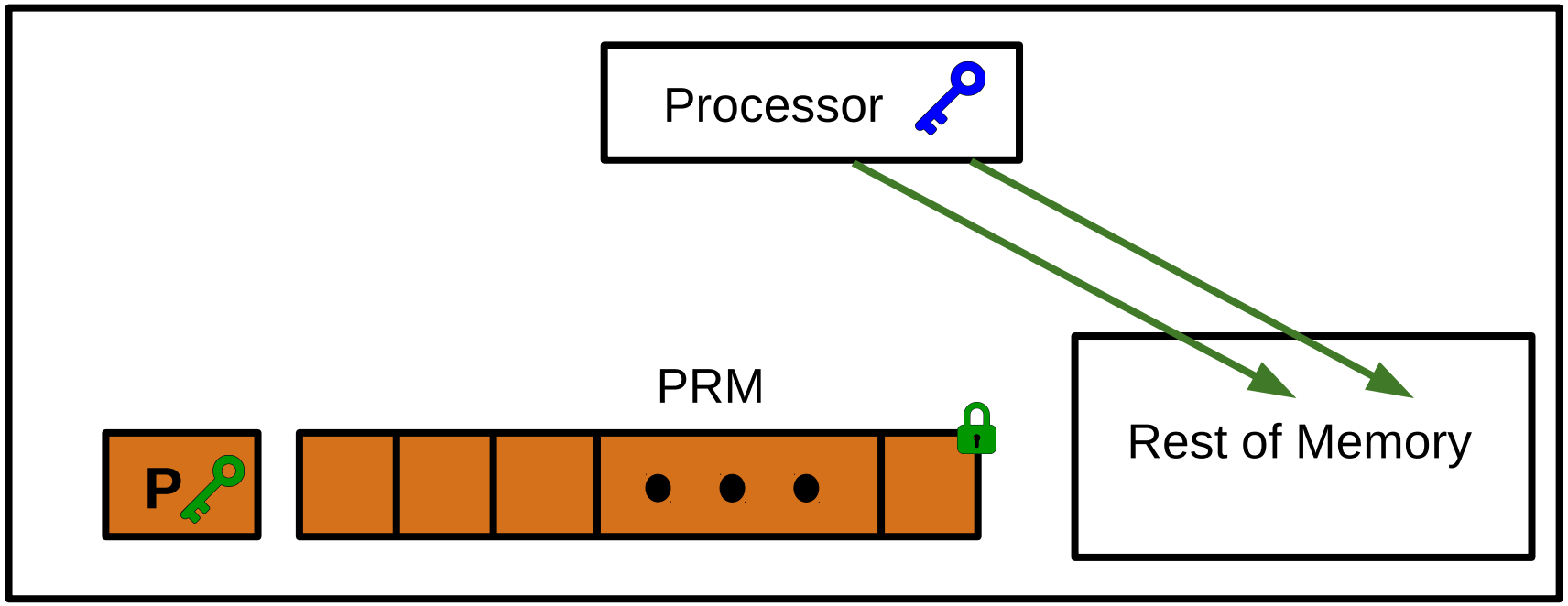
Levels of Obliviousness



External-Memory Oblivious: Access to data outside of the PRM are independent of any secret data.

Levels of Obliviousness

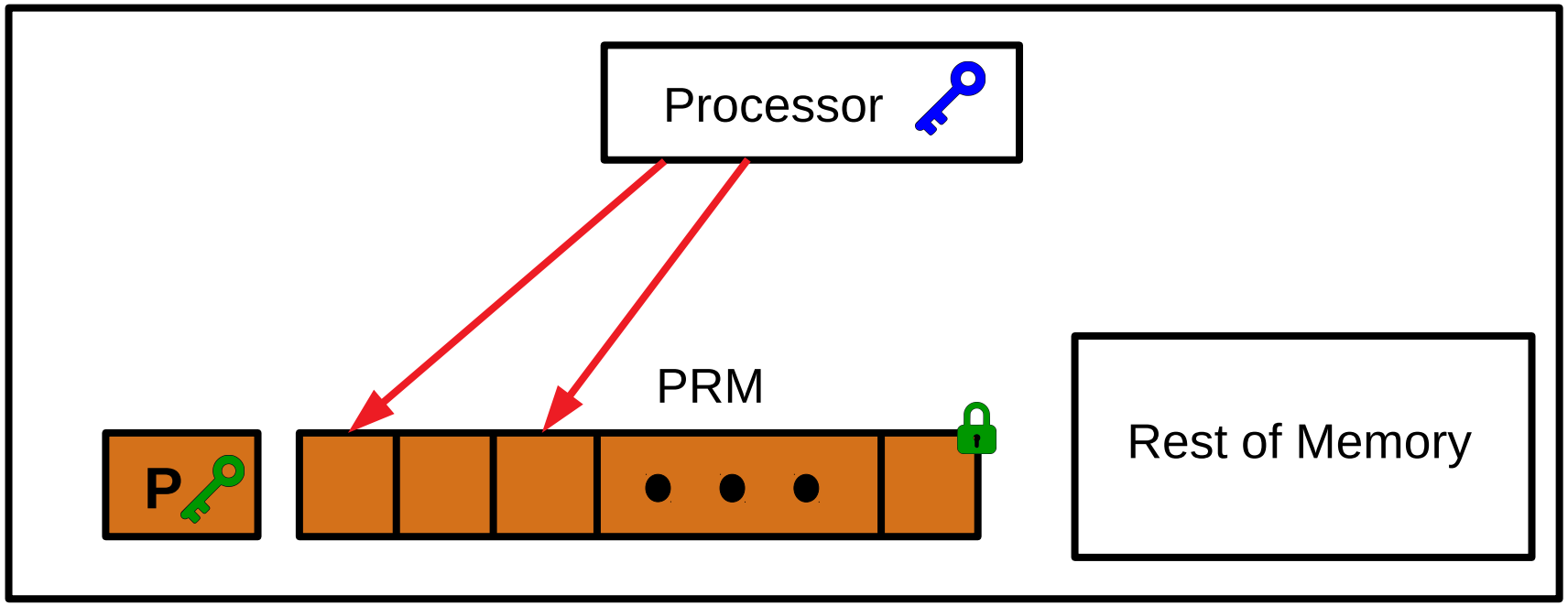
1) External-Memory



External-Memory Oblivious: Access to data outside of the PRM are independent of any secret data.

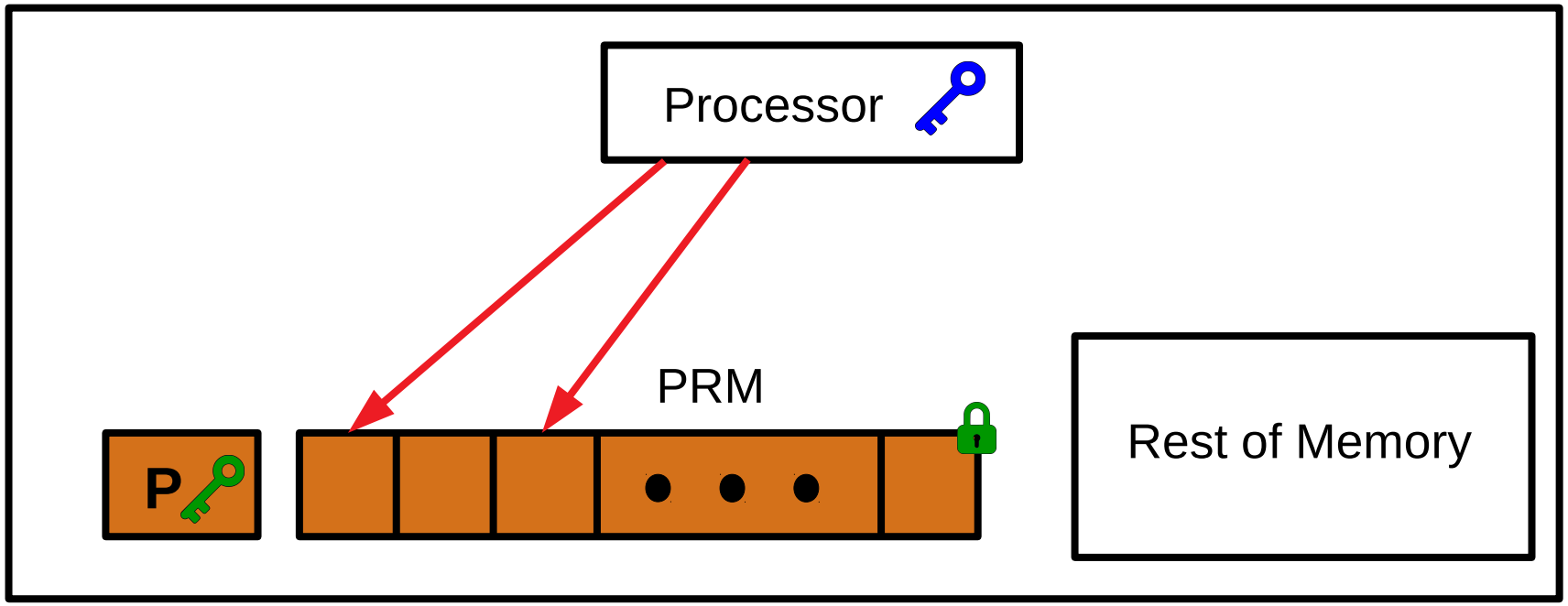
Levels of Obliviousness

1) External-Memory



Protected-Memory Oblivious: Access to data within the PRM are independent of any secret data.

Levels of Obliviousness

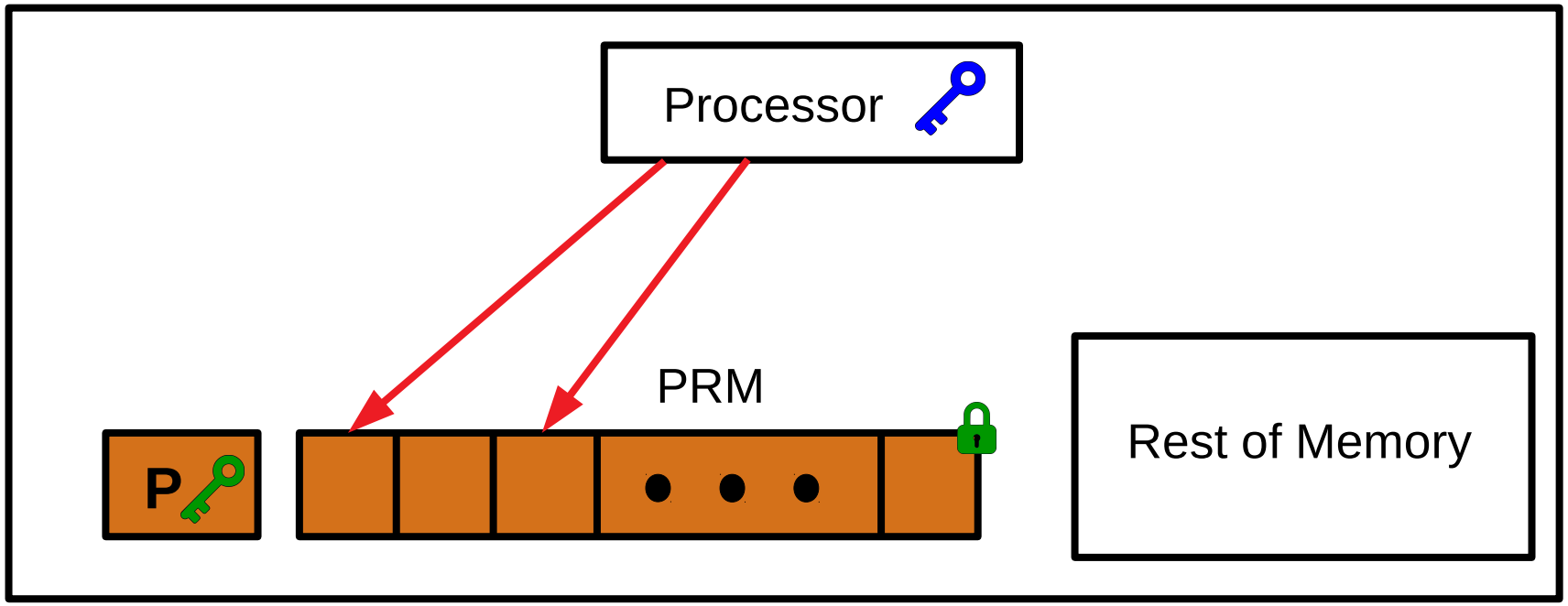


1) External-Memory

2) Protected-Memory

Protected-Memory Oblivious: Access to data within the PRM are independent of any secret data.

Levels of Obliviousness

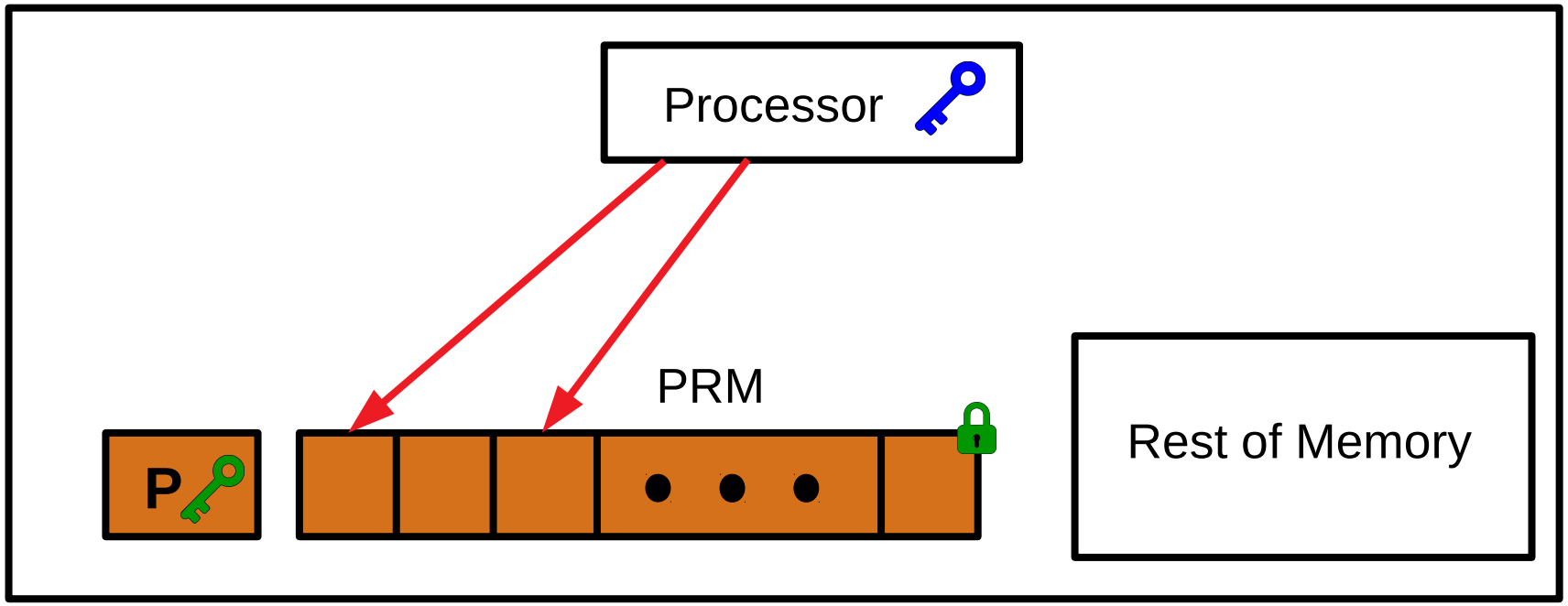


1) External-Memory

2) Protected-Memory
  i. Page

OS is responsible for page table management; Page-granular attacks induce page faults to extract memory locations accessed by the program.

Levels of Obliviousness

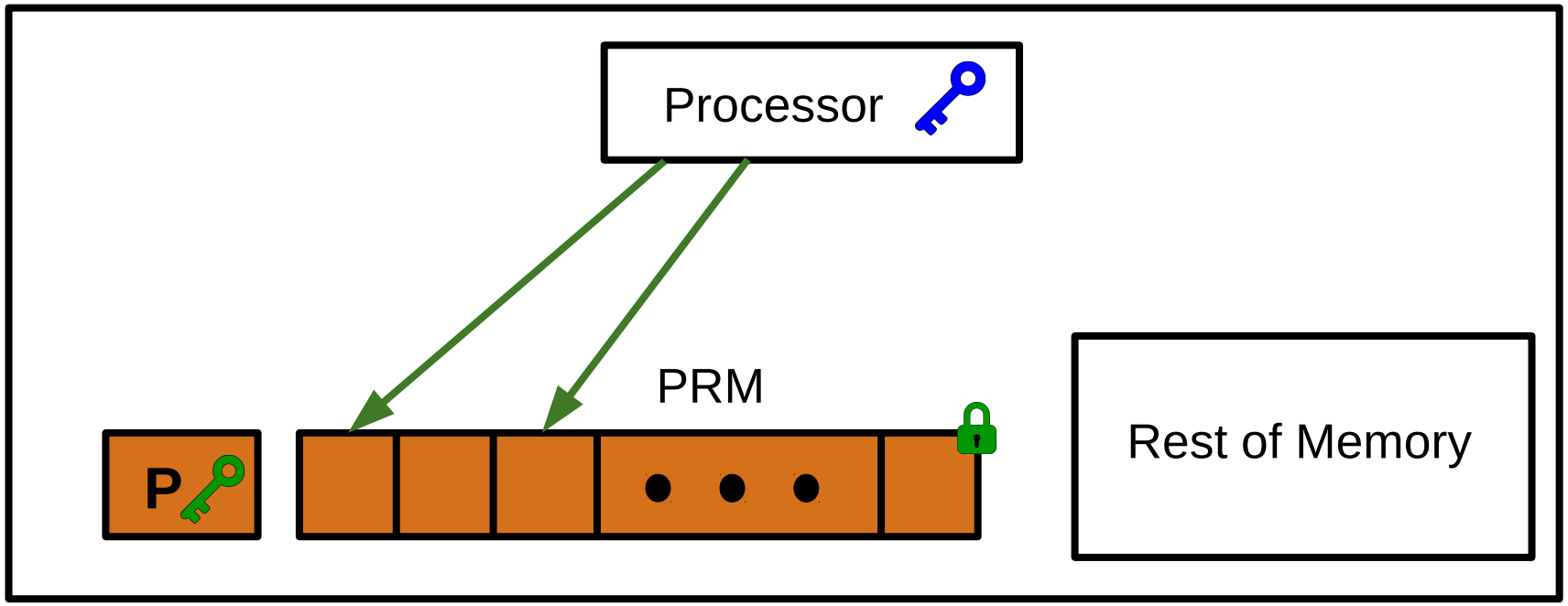


1) External-Memory

2) Protected-Memory
- **i. Page**
- **ii. Cacheline**

Recent attacks extract the precise address of the 64B cacheline loaded during a page fault.

Levels of Obliviousness

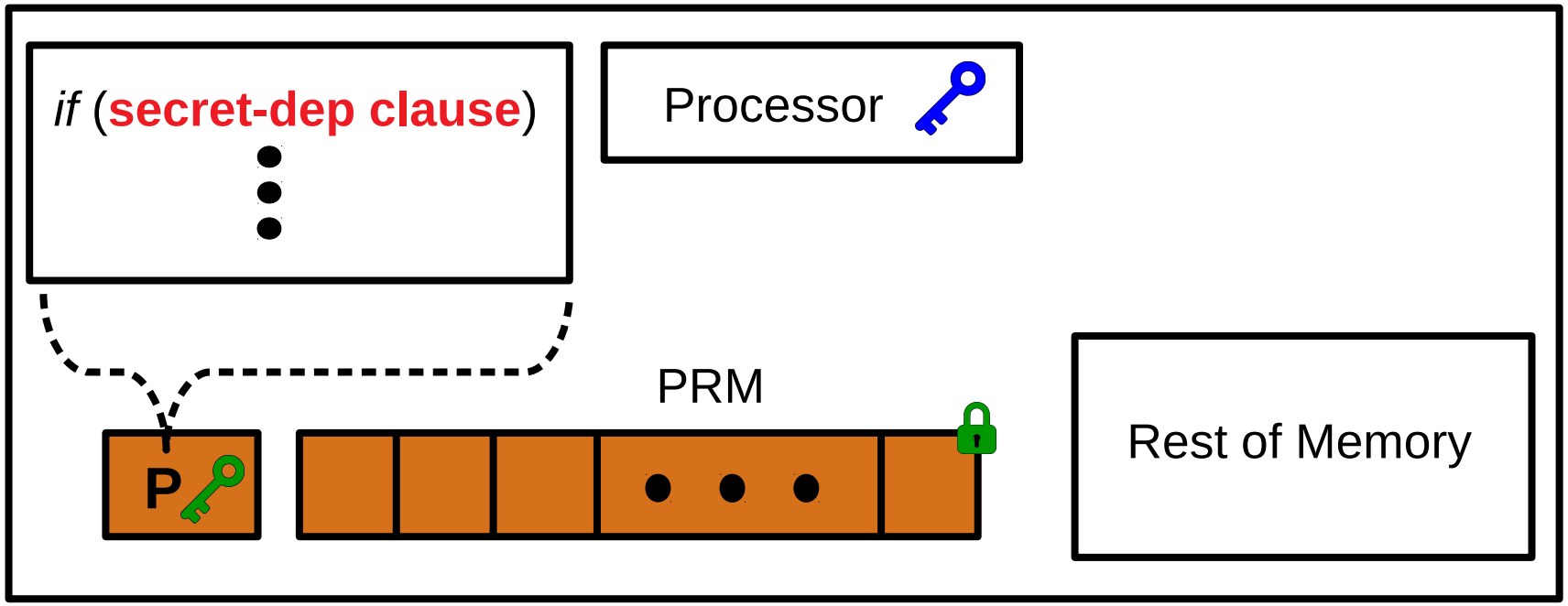


1) External-Memory

2) Protected-Memory
- **i. Page**
- **ii. Cacheline**
- **iii. Subcacheline**

Recent attacks extract the precise address of the 64B cacheline loaded during a page fault.

Levels of Obliviousness

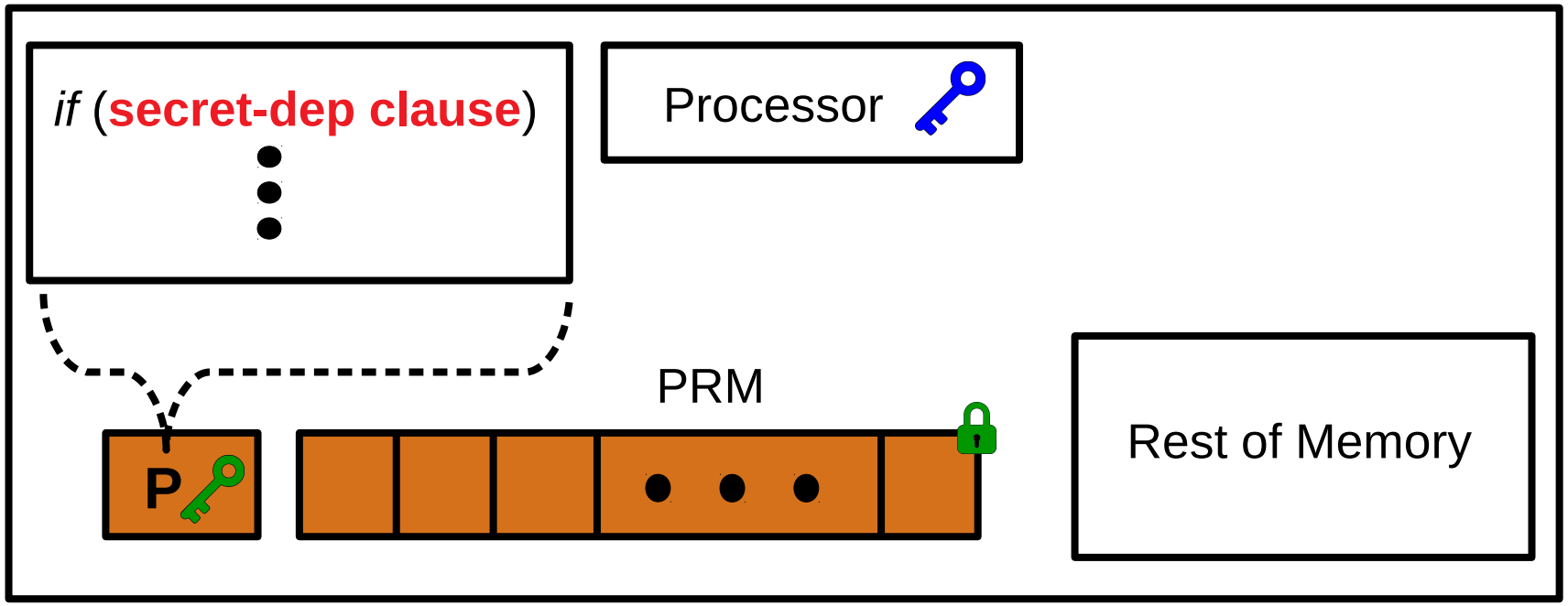


1) External-Memory

2) Protected-Memory
- **i. Page**
- **ii. Cacheline**
- **iii. Subcacheline**

Control-Flow oblivious: Secret-dependent control flow branches leak information about the underlying secret; ensure that the program has no secret-dependent control-flow branches.

Levels of Obliviousness



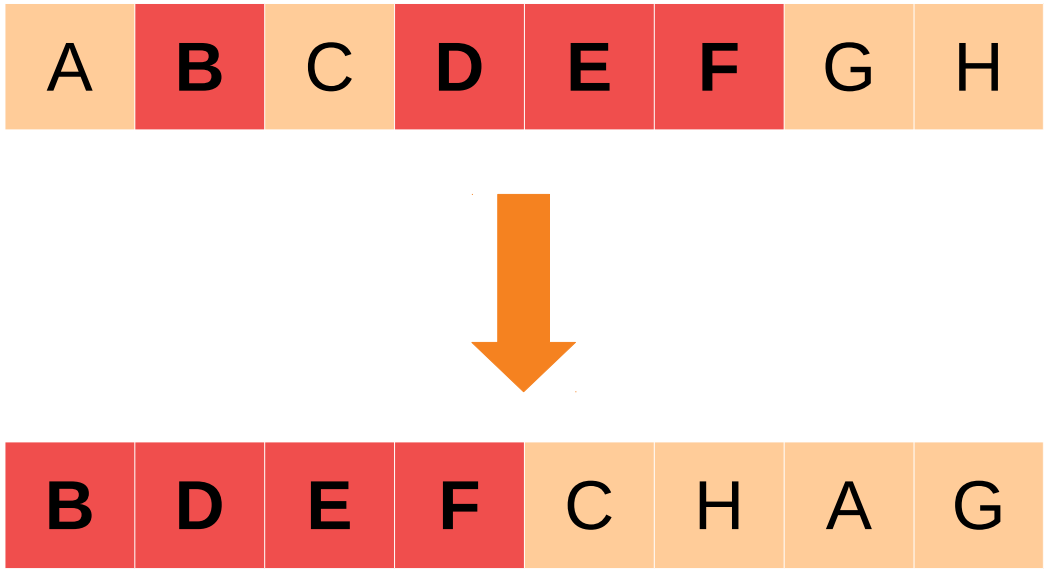
1) External-Memory

2) Protected-Memory
- **i. Page**
- **ii. Cacheline**
- **iii. Subcacheline**

3) Control-flow

Fully Oblivious: A program is fully oblivious if it satisfies all above definitions of obliviousness.

Order-Preserving Tight Compaction



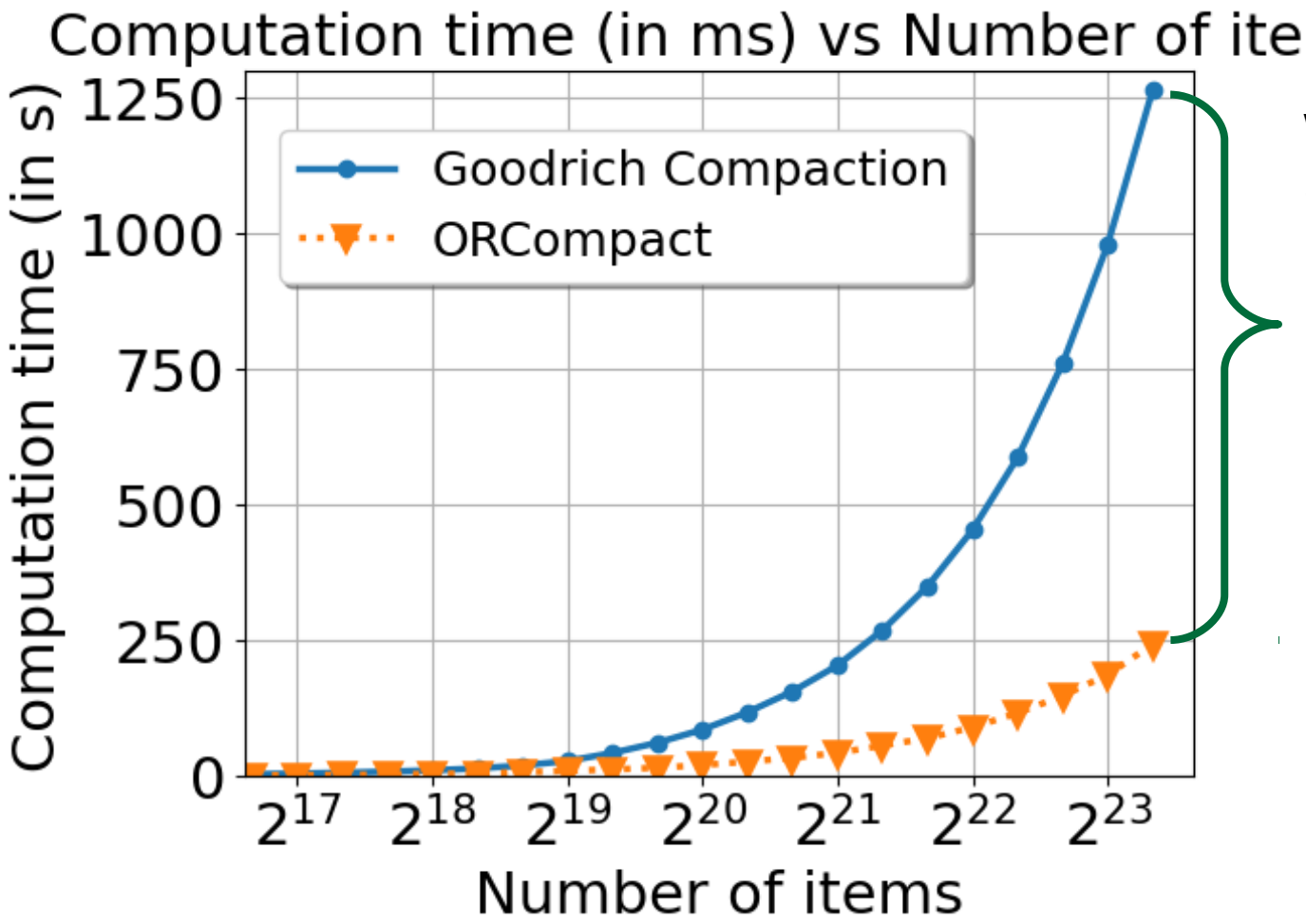
Goal: Bring all the marked items to the start of the array

Order-Preserving Tight Compaction Algorithms

Algorithm	Complexity	Constant	Parallel
Goodrich (2011)	n lgn	1	☒
Asharov et al. (2018)	n	$>2^{228}$	☒
Dittmer & Ostrovsky (2020)	n	~9405.7	☒
ORCompact (2022)	**n lgn**	**1/2**	☑

Check out our full paper for details of the ORCompact algorithm, proofs of correctness and obliviousness

ORCompact Time Evaluation



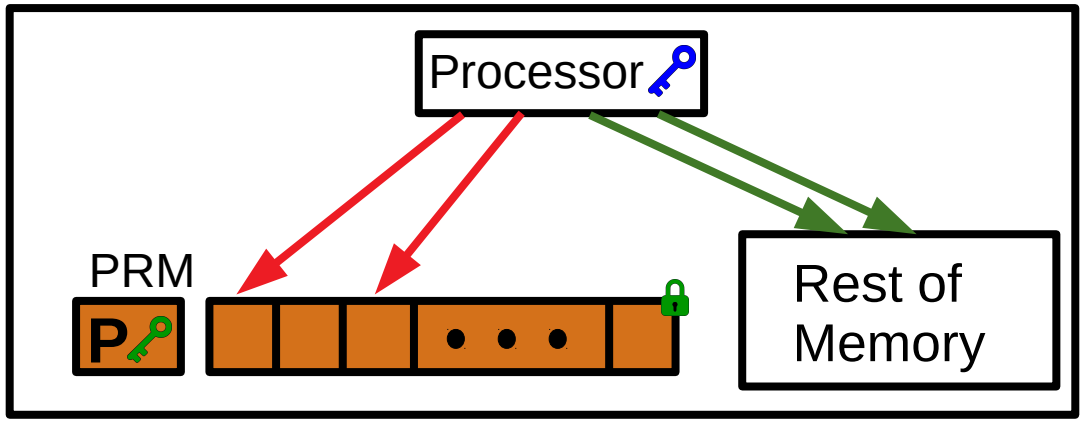
With a fixed block size
b = 8 bytes

5.2x improvement

Oblivious Shuffling

- Melbourne Shuffle
- Stash Shuffle (Prochlo)

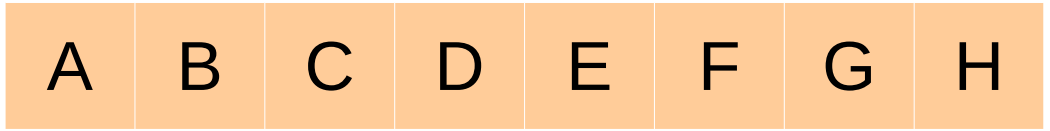
Assumes PRM as private unobservable memory



- In practice, Batcher's Sorting Network (1968) is used
  - To shuffle, attach random tags to the items to sort
  - For simplicity, we call this Bitonic Shuffle

Oblivious Recursive Shuffle (ORShuffle)

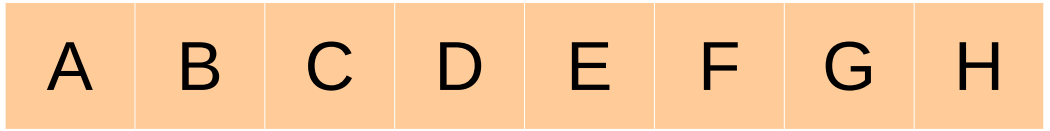
$\textsc{ORShuffle}(D)$



Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)



Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)

Oblivious Recursive Shuffle (ORShuffle)

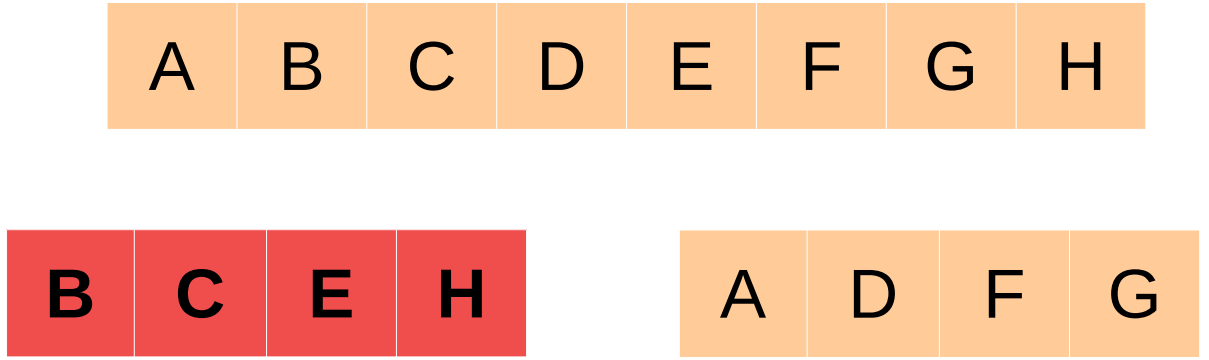
ORShuffle(D)

- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)
  - ORCompact(D, M)

Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

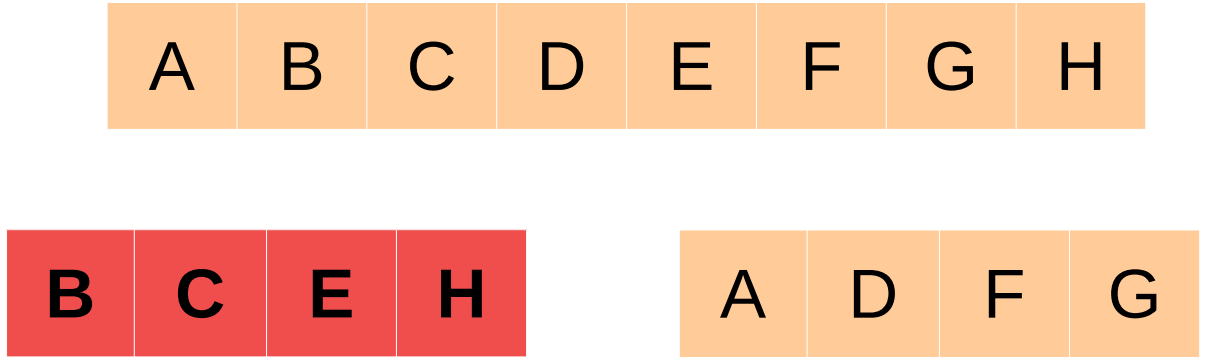
- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)
  - ORCompact(D, M)



Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

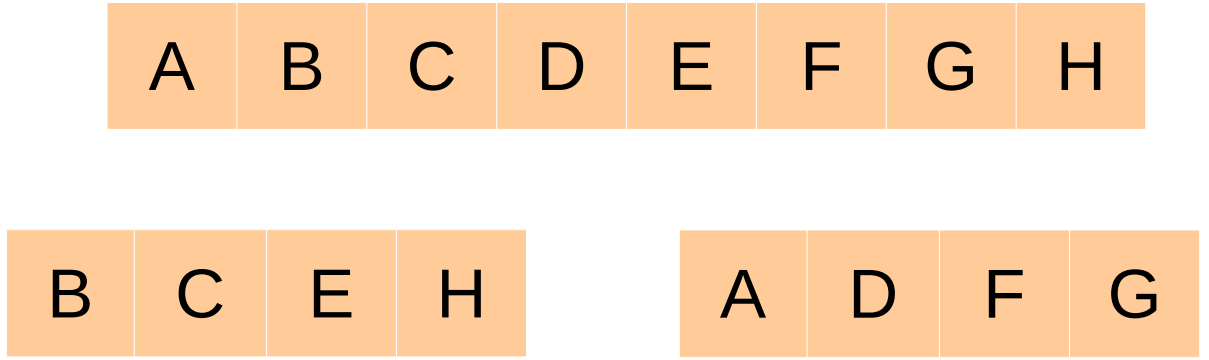
- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)
  - ORCompact(D, M)
  - ORShuffle($D_{0..\lceil \frac{n}{2} \rceil - 1}$)
  - ORShuffle($D_{\lceil \frac{n}{2} \rceil..n-1}$)



Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

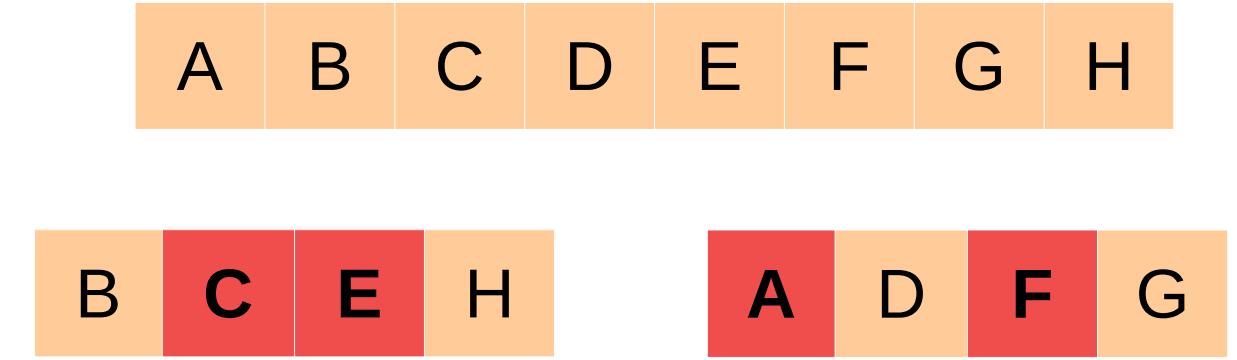
- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)
  - ORCompact(D, M)
  - ORShuffle($D_{0..\lceil \frac{n}{2} \rceil - 1}$)
  - ORShuffle($D_{\lceil \frac{n}{2} \rceil ..n-1}$)



Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

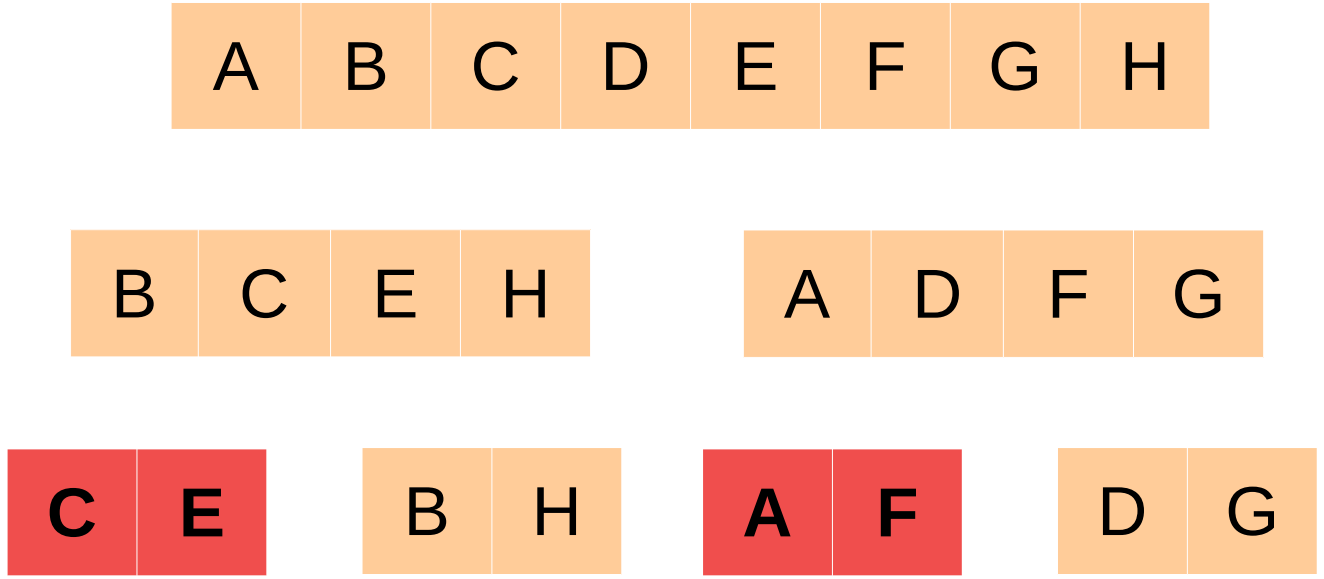
- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)
  - ORCompact(D, M)
  - ORShuffle($D_{0..\lceil \frac{n}{2} \rceil - 1}$)
  - ORShuffle($D_{\lceil \frac{n}{2} \rceil .. n-1}$)



Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

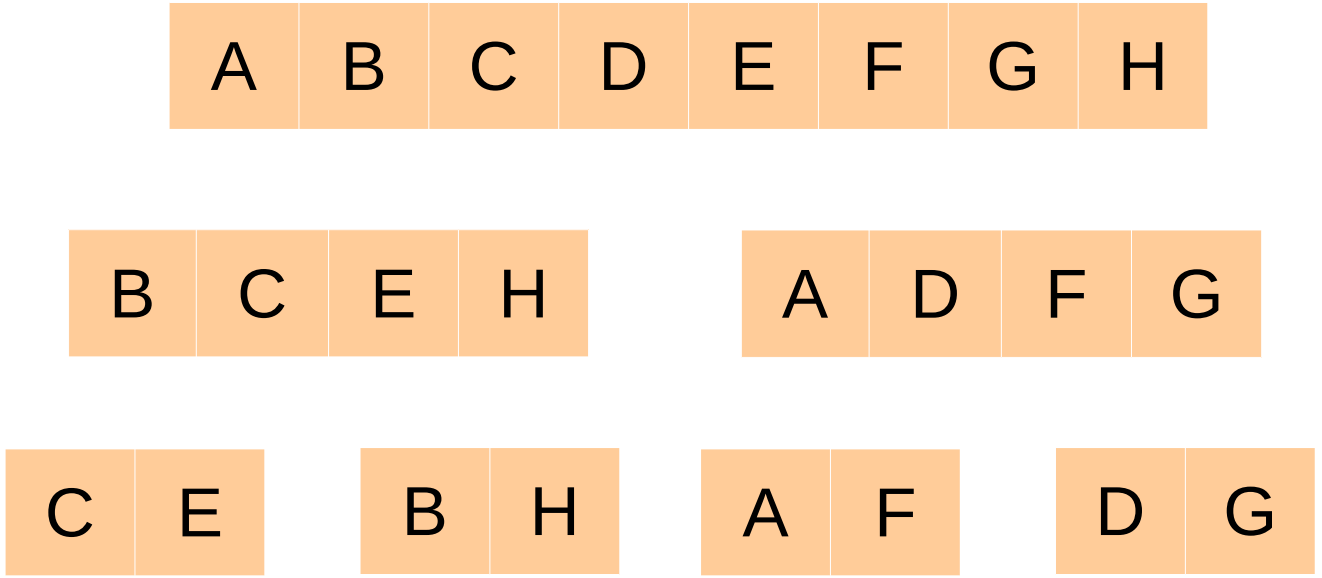
- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)
  - ORCompact(D, M)
  - ORShuffle($D_{0..\lceil \frac{n}{2} \rceil - 1}$)
  - ORShuffle($D_{\lceil \frac{n}{2} \rceil ..n-1}$)



Oblivious Recursive Shuffle (ORShuffle)

ORShuffle(D)

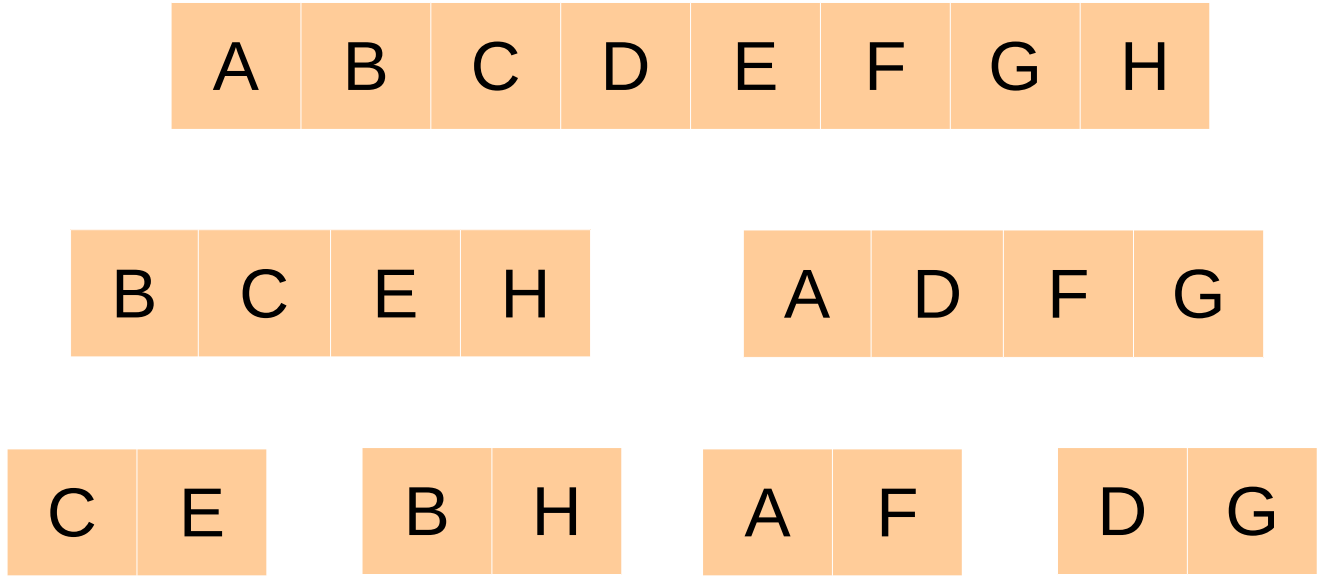
- $n \leftarrow |D|$
- **if** $n > 2$:
  - MarkHalf(n)
  - ORCompact(D, M)
  - ORShuffle($D_{0..\lceil \frac{n}{2} \rceil - 1}$)
  - ORShuffle($D_{\lceil \frac{n}{2} \rceil..n-1}$)



Oblivious Recursive Shuffle (ORShuffle)

$\textsc{ORShuffle}(D)$

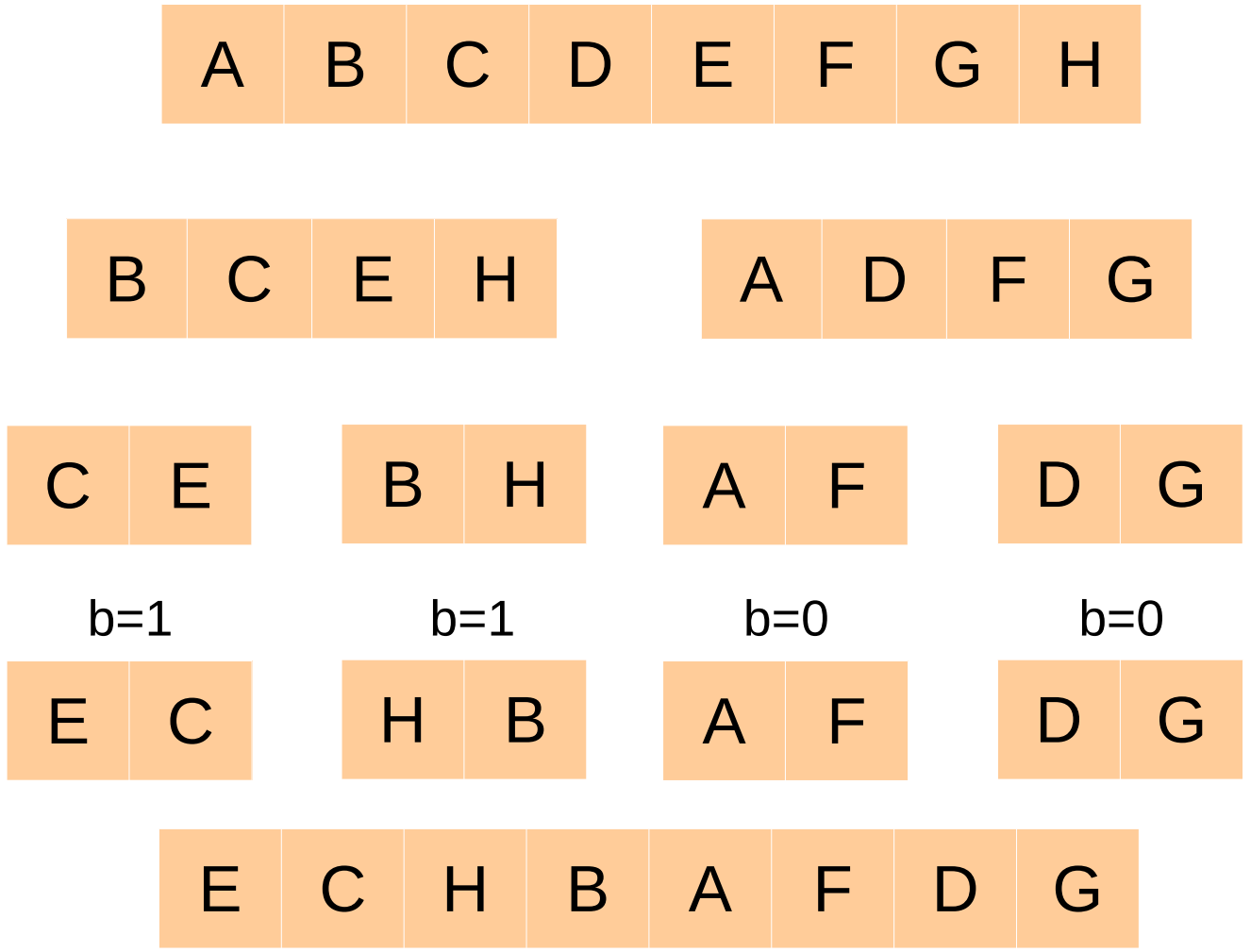
- $n \leftarrow |D|$
- **if** $n > 2$:
  - $\textsc{MarkHalf}(n)$
  - $\textsc{ORCompact}(D, M)$
  - $\textsc{ORShuffle}(D_{0..\lceil \frac{n}{2} \rceil - 1})$
  - $\textsc{ORShuffle}(D_{\lceil \frac{n}{2} \rceil ..n-1})$
- else:
  - Sample $b \in 0, 1$
  - $\textsc{OSwap}(D[0], D[1], b)$



Oblivious Recursive Shuffle (ORShuffle)

$\textsc{ORShuffle}(D)$

- $n \leftarrow |D|$
- **if** $n > 2$:
  - $\textsc{MarkHalf}(n)$
  - $\textsc{ORCompact}(D, M)$
  - $\textsc{ORShuffle}(D_{0..\lceil \frac{n}{2} \rceil - 1})$
  - $\textsc{ORShuffle}(D_{\lceil \frac{n}{2} \rceil..n-1})$
- else:
  - Sample $b \in 0, 1$
  - $\textsc{OSwap}(D[0], D[1], b)$



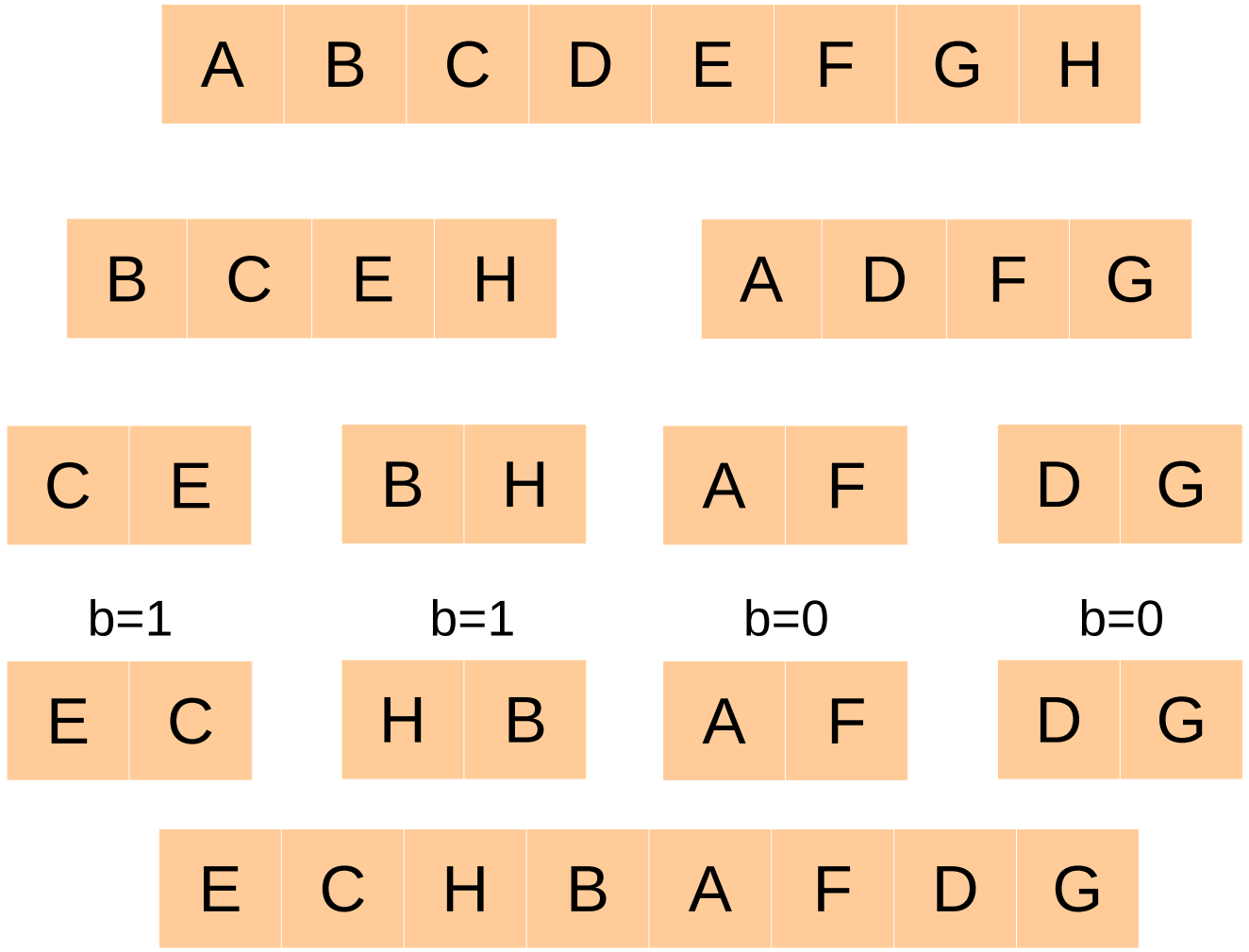
Oblivious Recursive Shuffle (ORShuffle)

$\textsc{ORShuffle}(D)$

- $n \leftarrow |D|$
- **if** $n > 2$:
  - $\textsc{MarkHalf}(n)$
  - $\textsc{ORCompact}(D, M)$
  - $\textsc{ORShuffle}(D_{0..\lceil \frac{n}{2} \rceil - 1})$
  - $\textsc{ORShuffle}(D_{\lceil \frac{n}{2} \rceil..n-1})$
- else:
  - Sample $b \in 0, 1$
  - $\textsc{OSwap}(D[0], D[1], b)$

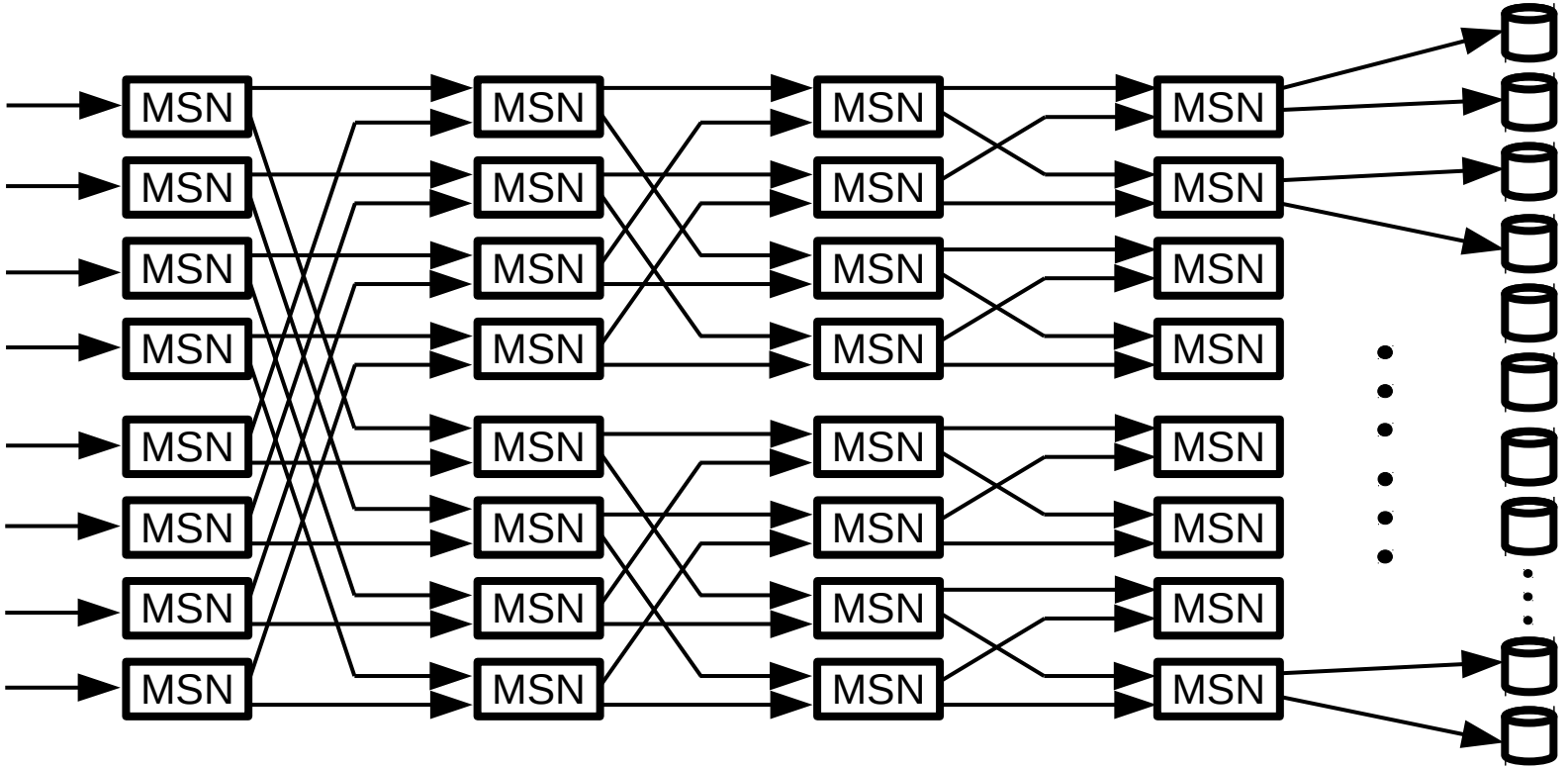
$O(n \log^2 n)$ complexity

$O(\log^2 n)$ parallel step complexity



BORPStream

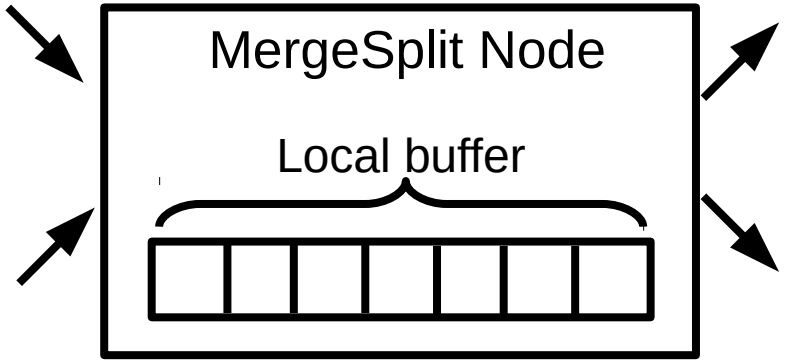
- Inspired by BORP, redesigned for streaming settings; i.e. when packets to shuffle arrive intermittently.



- As items to shuffle arrive, they are assigned a random destination bucket label, and routed through the Butterfly Routing Network (BRN) of MergeSplit Nodes (MSN) to their bucket

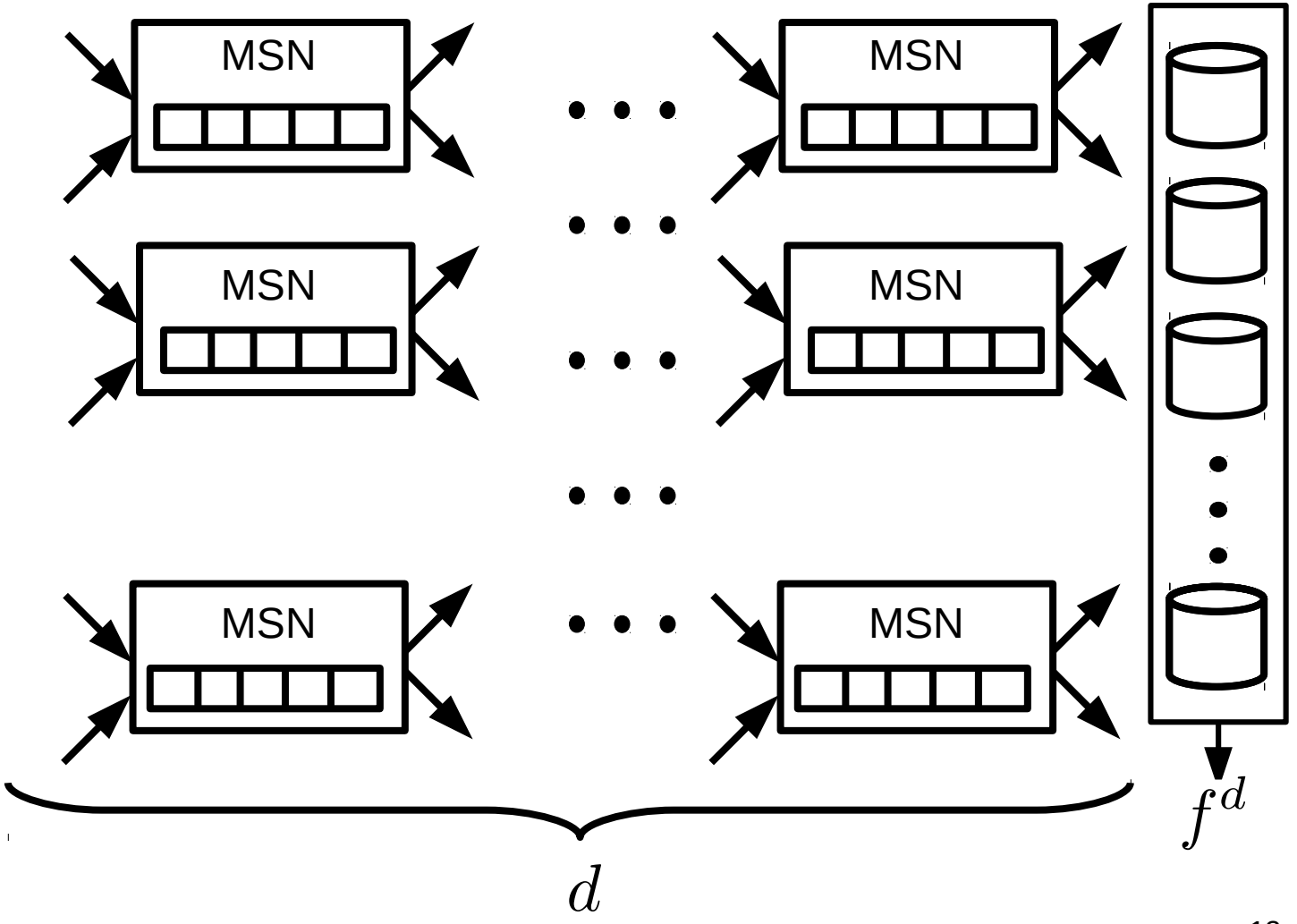
MergeSplitNode

- Each MSN can have a fan of f; i.e. f incoming and outgoing connections
- MSNs evict packet to their f output streams in a round robin fashion
- Maintain a small local buffer to handle routing correctness
- For an incoming packet, the MSN obliviously scans the buffer, and picks an item destined for the current eviction stream, or else sends out a dummy



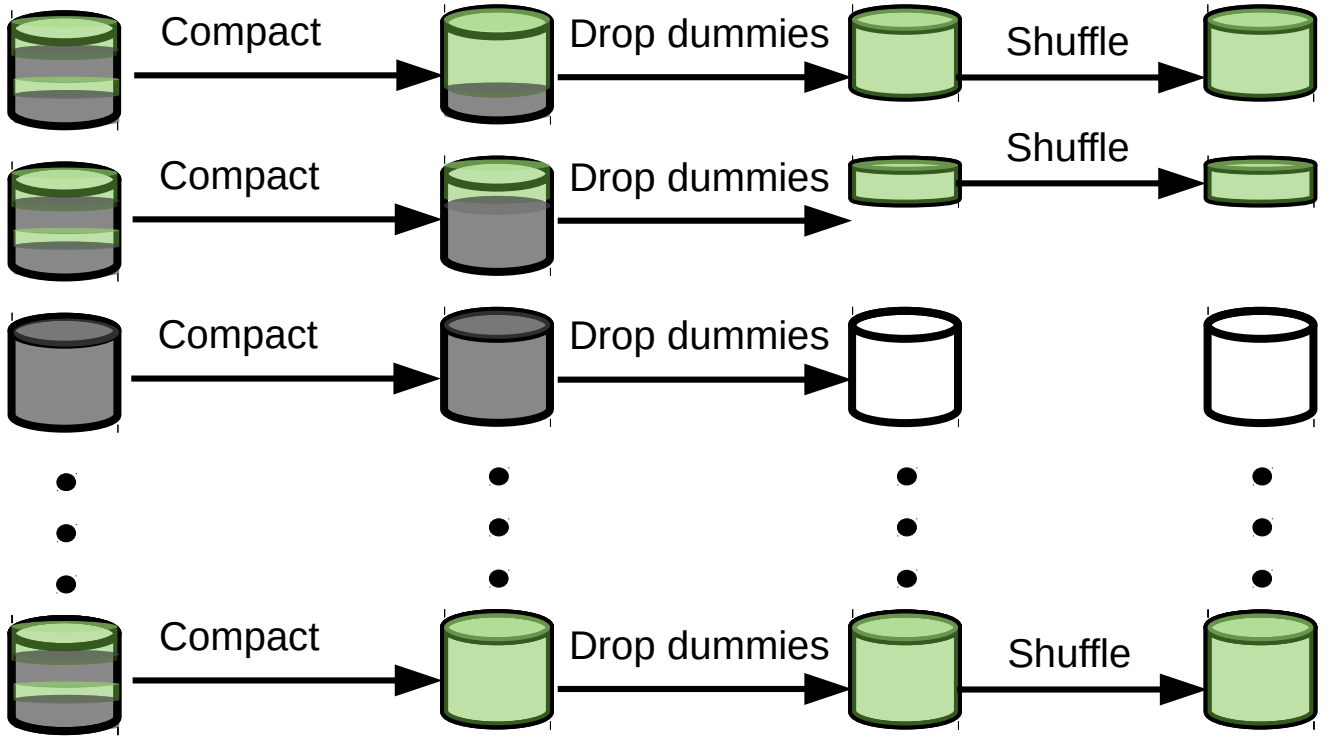
BORPStream Phase 1

- Packets are routed through the BRN of MSNs as they arrive along with a dummy packet
- Failure probability (2^{-80}) depends on if the MSN buffer overflows (Details of computing the failure probability, correctness and obliviousness of BORPStream in the full version of our paper.)
- Note failures are silent and do NOT leak information

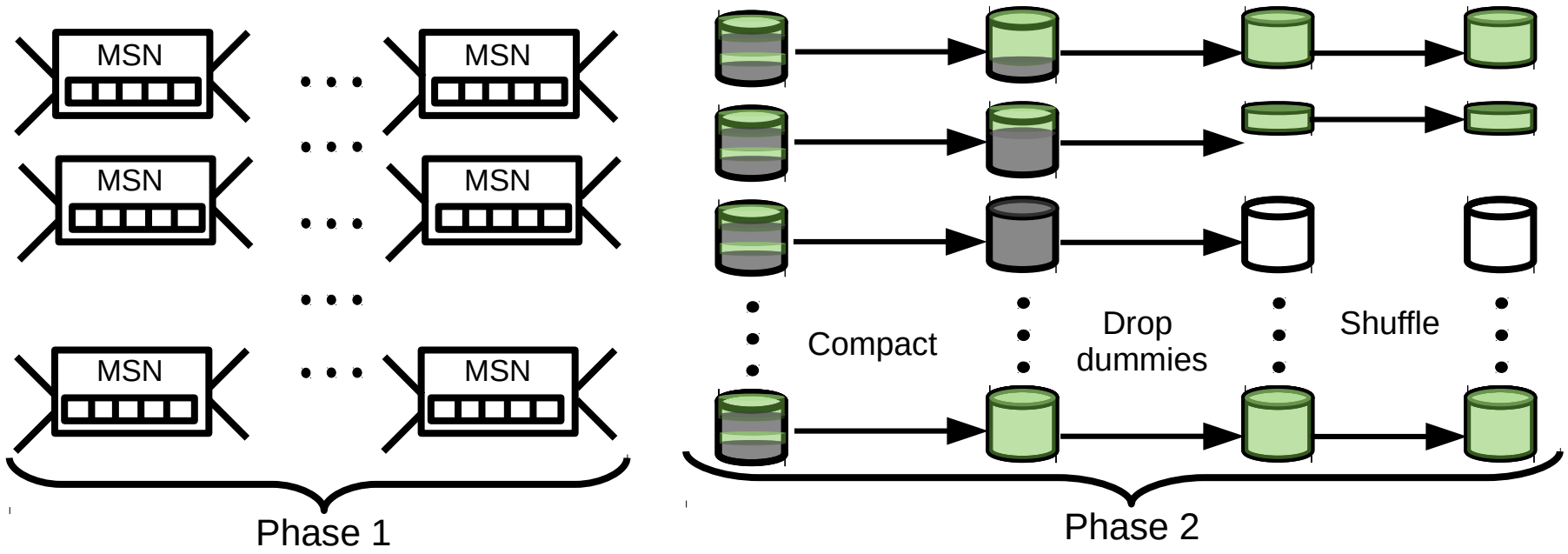


BORPStream Phase 2

- ORCompact to bring all the real items to the start of the bucket
- Drop all the dummy items in each bucket
- ORShuffle to randomize locations of reals in each bucket
- Merge all the shuffled buckets to produce obliviously shuffled output

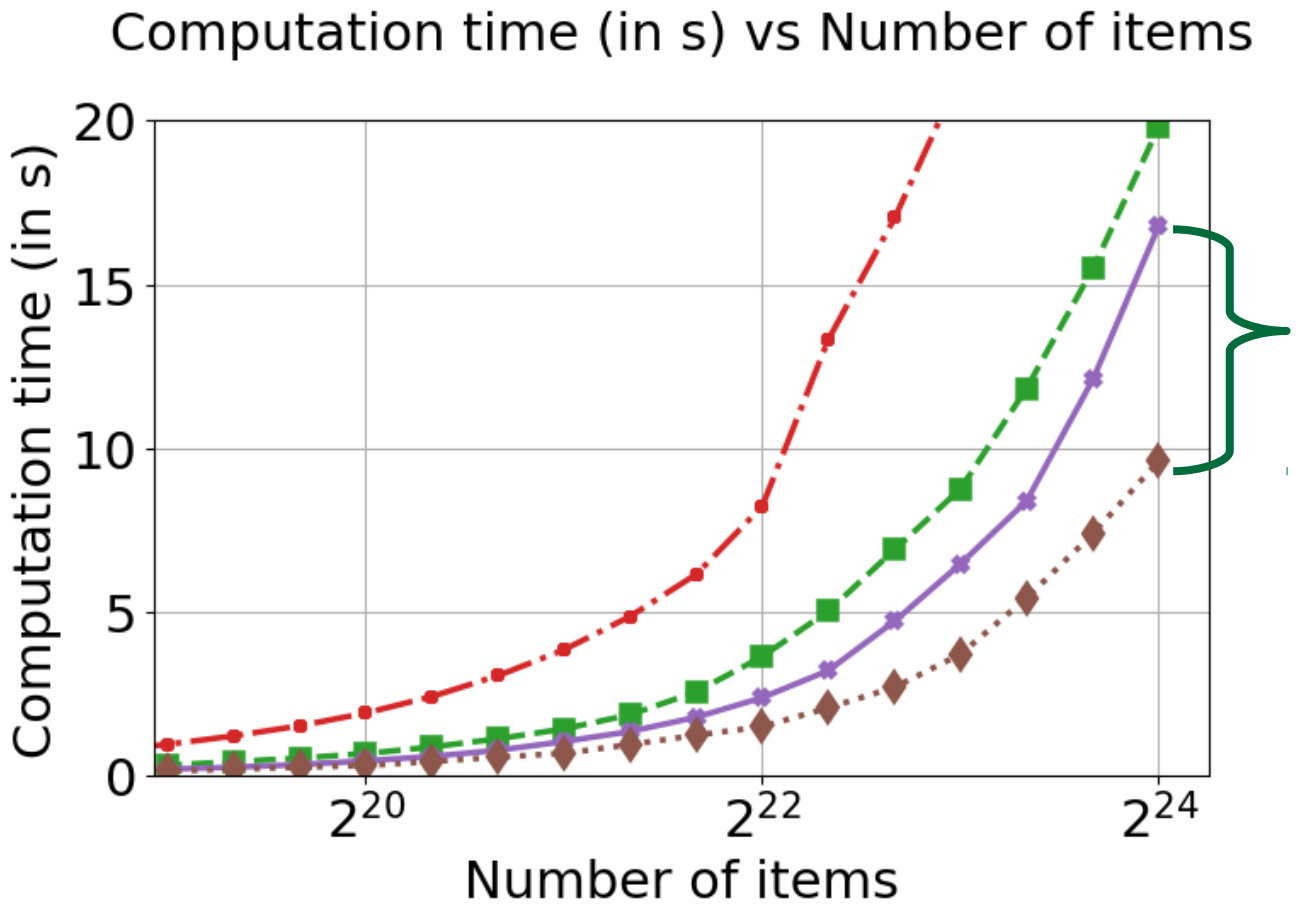


BORPStream



- When data to be shuffled arrive intermittently BORPStream enables partial shuffling of items
- BORPStream can be tuned to minimize total time (V1) or Phase 2 time alone (V2)
- BORPStream has an $O(n \lg^2 n)$ complexity

Shuffle Algorithms Evaluation

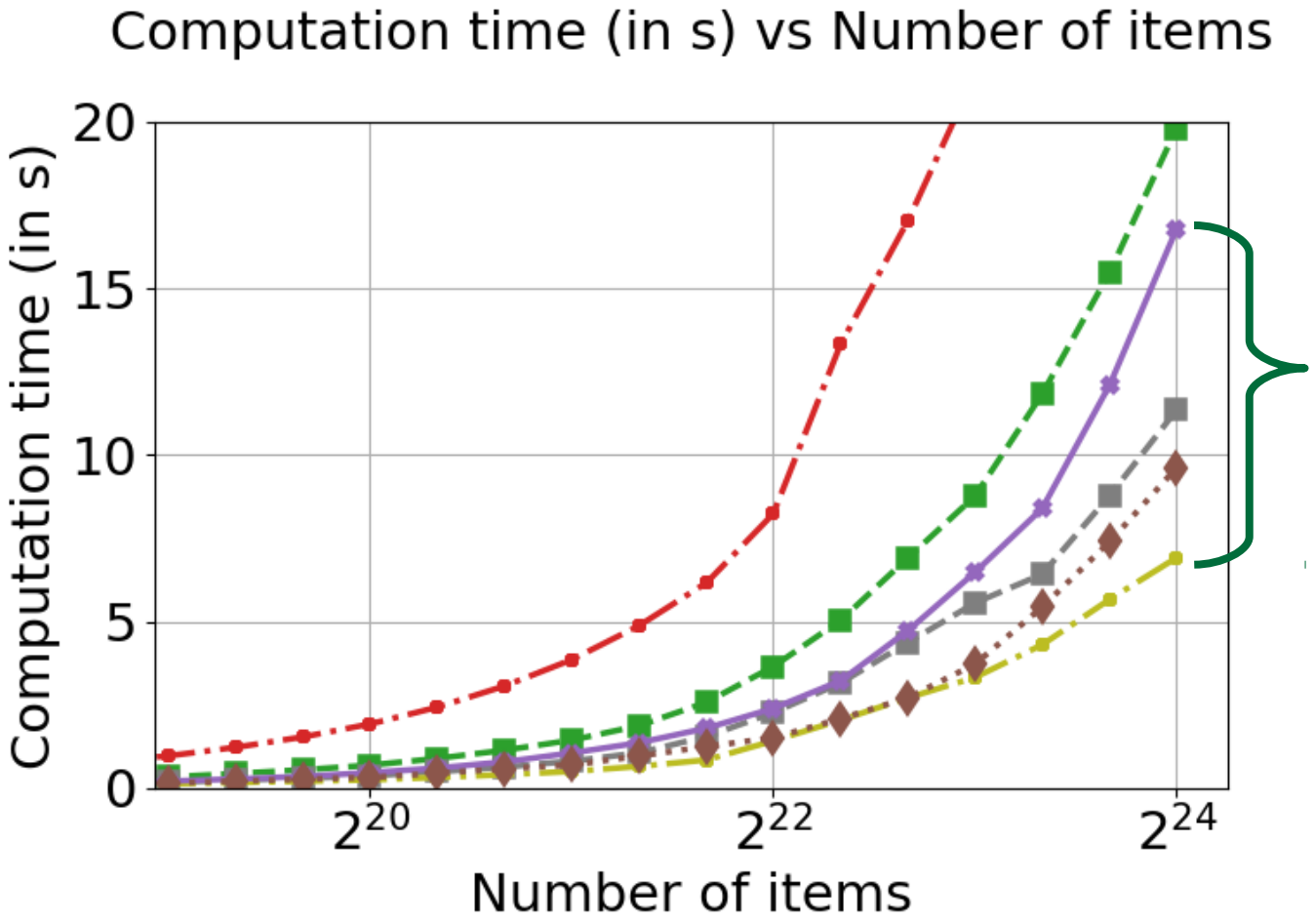


With a fixed block size
b = 8 bytes

1.8x improvement

BORPStream V1 Total | BORPStream V2 Total | Bitonic Shuffle | ORShuffle

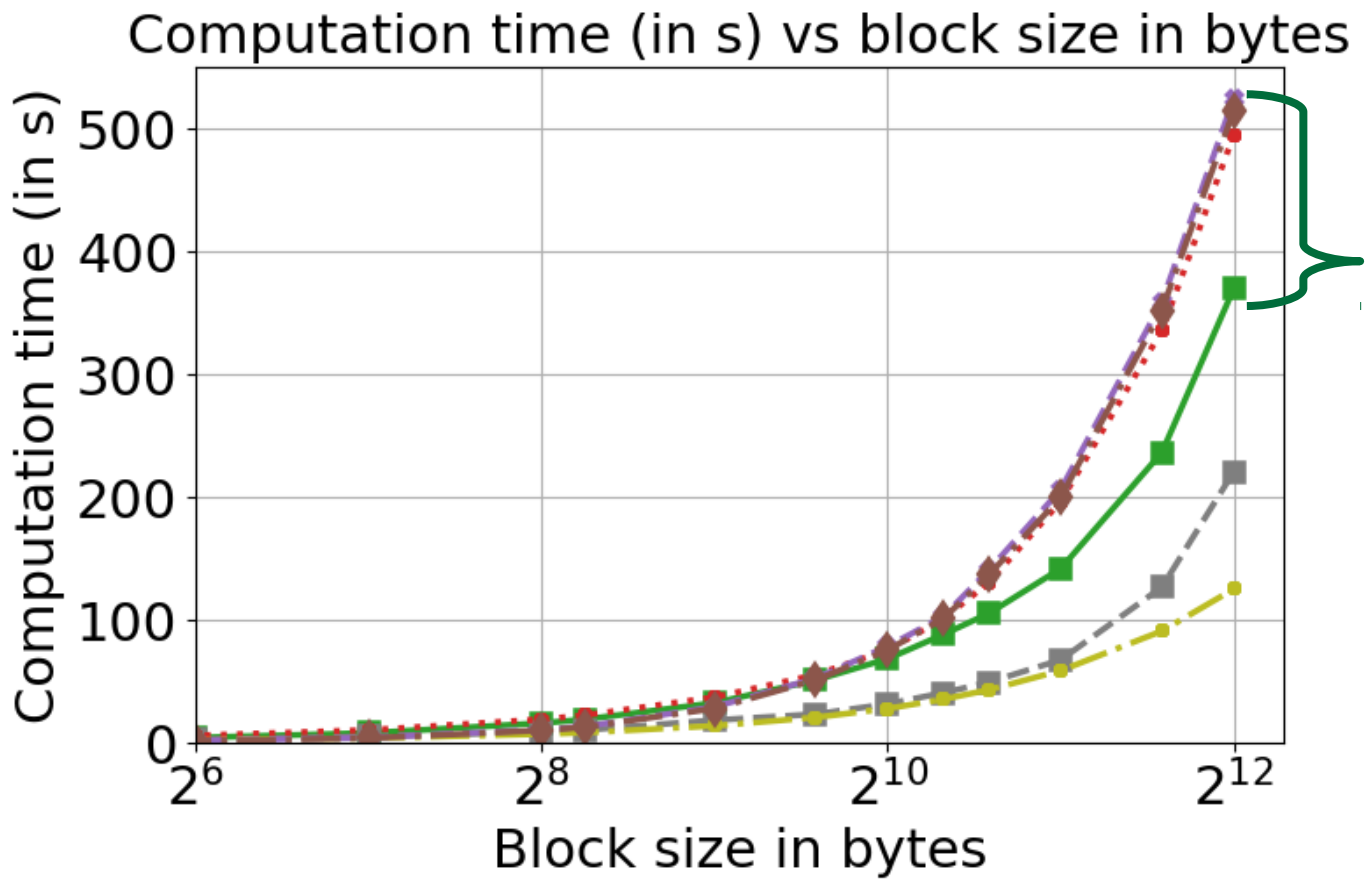
Shuffle Algorithms Evaluation



With a fixed block size
b = 8 bytes

2.5x improvement

Shuffle Algorithms Evaluation

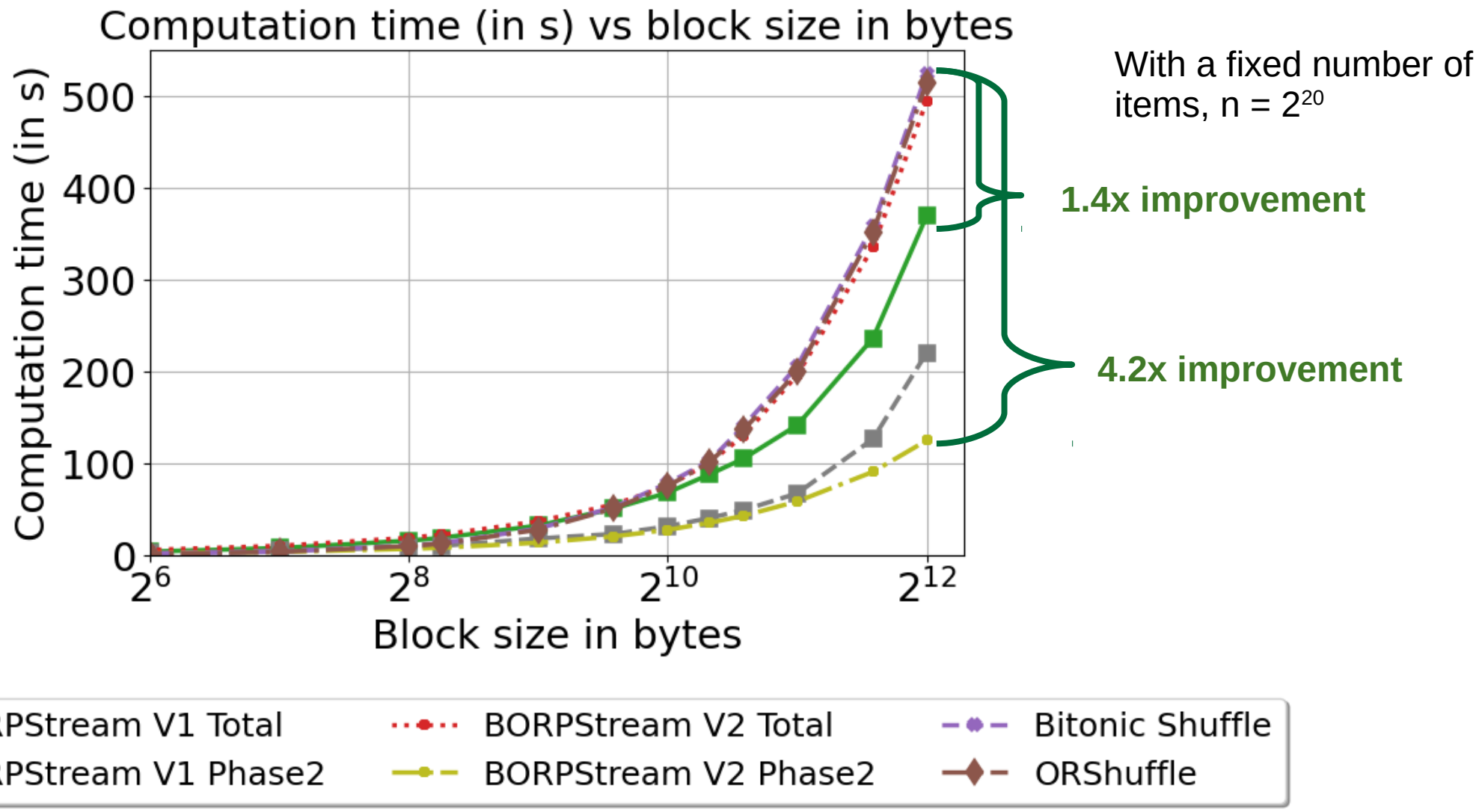


With a fixed number of items, $n = 2^{20}$

1.4x improvement

BORPStream V1 Total
BORPStream V1 Phase2
BORPStream V2 Total
BORPStream V2 Phase2
Bitonic Shuffle
ORShuffle

Shuffle Algorithms Evaluation



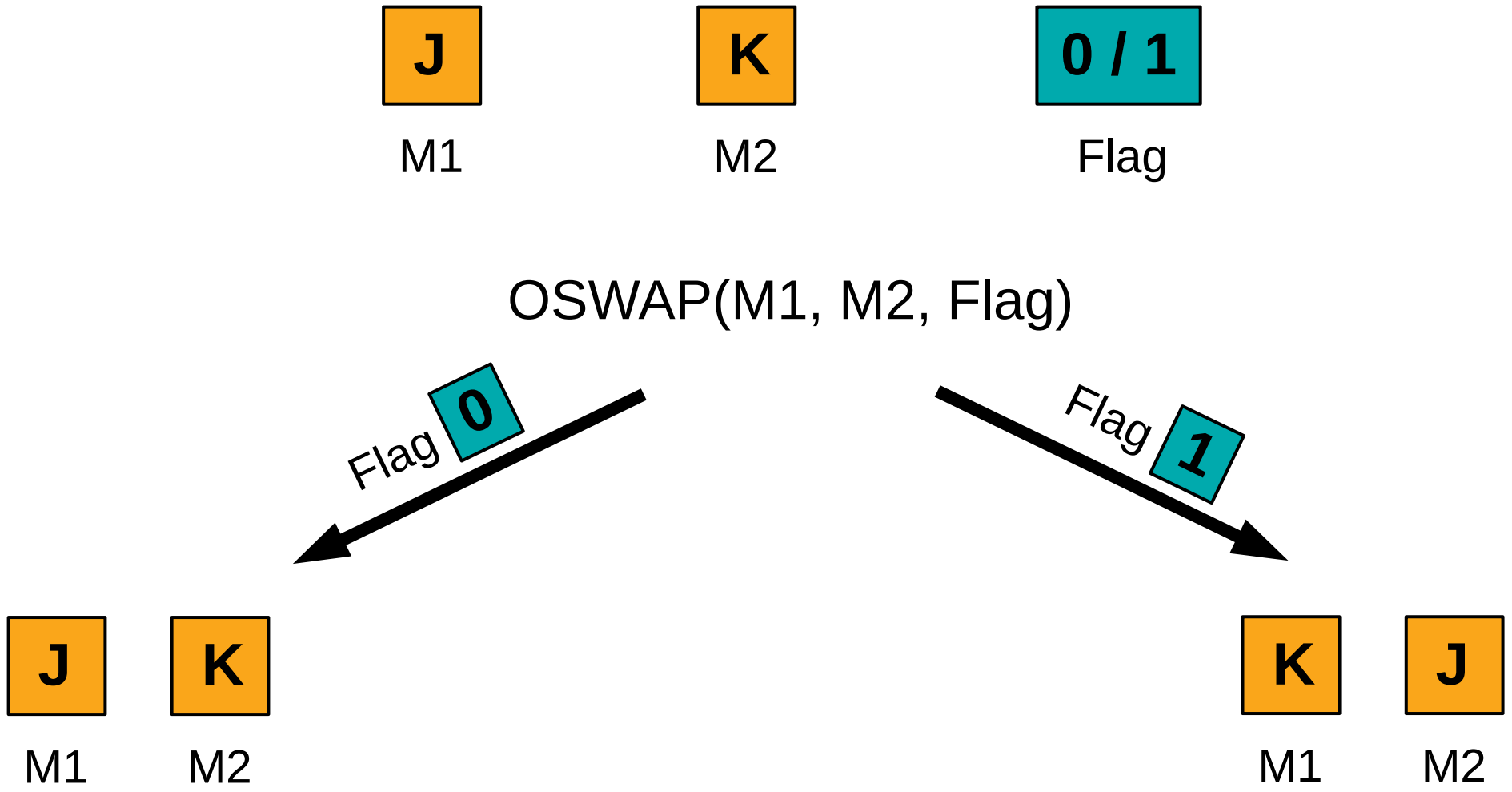
Fully Oblivious Assembly Verifier (FOAV)

- Instrument C/C++ source to insert assembly comments to state if a variable is “safe”
- FOAV analyzes the output assembly to verify control flow obliviousness
- It tracks all the conditional jump instructions, and their flag-manipulating instruction (FMI) to ensure that operands of the FMI are marked safe

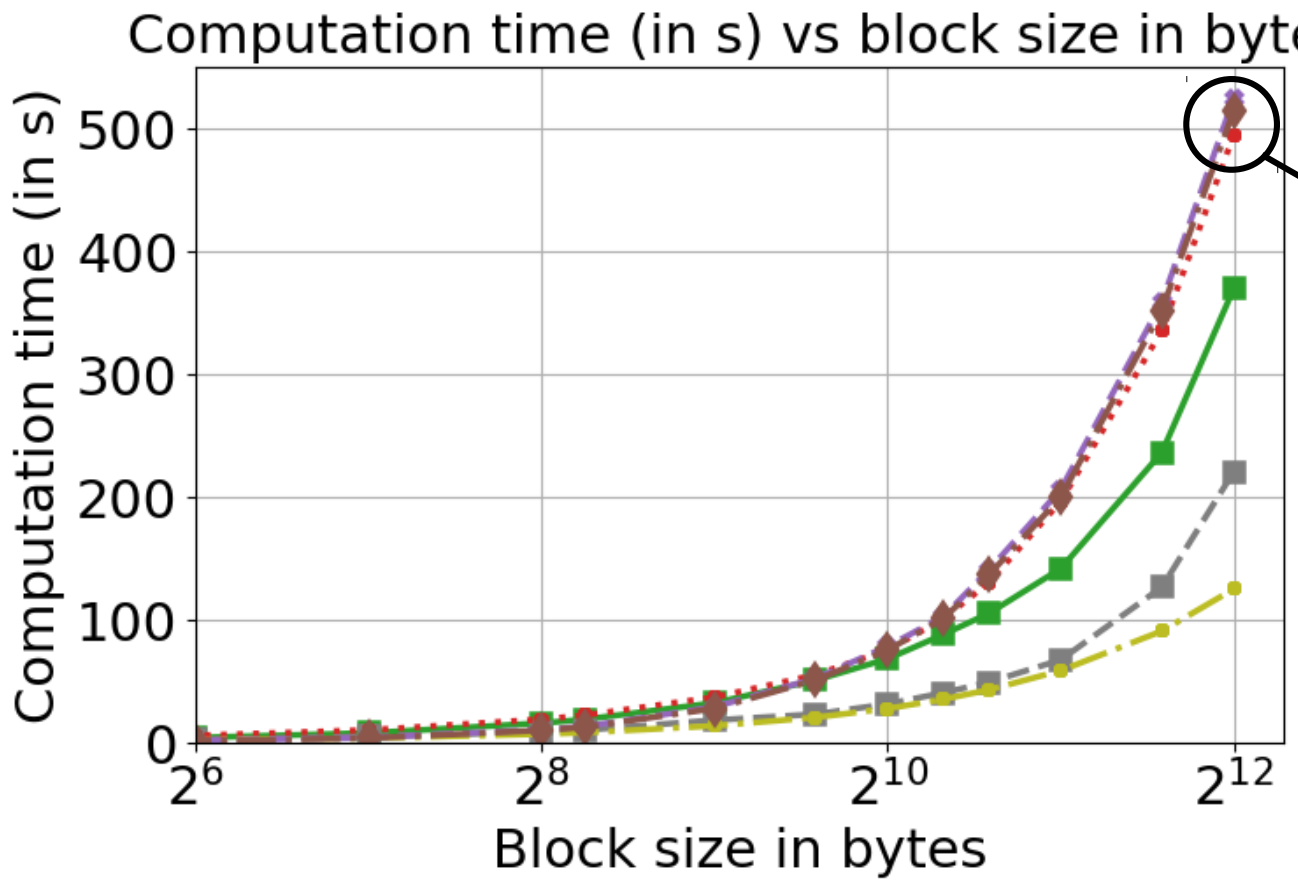
Summary

- Introduced efficient fully oblivious algorithms for:
  - Order-preserving tight compaction: **ORCompact**
  - Shuffling: **ORShuffle** and **BORPStream**
- Ensure obliviousness at assembly level: **FOAV**
- Code: https://crysp.uwaterloo.ca/software/obliv/

Oblivious Swap (OSWAP)



Shuffle Algorithms Evaluation



With a fixed number of items, $n = 2^{20}$

Bitonic, ORShuffle, and BORPStream V2 Total time

BORPStream V1 Total
BORPStream V2 Total
Bitonic Shuffle
BORPStream V1 Phase2
BORPStream V2 Phase2
ORShuffle

ORCompact

ORCompact(D,M):

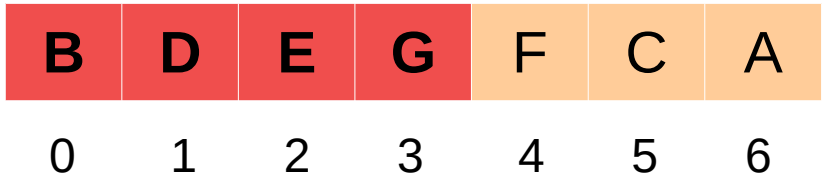
- **if** $|D| = 0$ **return**
- $n \leftarrow |D|, n_2 \leftarrow 2^{\lfloor log(n) \rfloor}, n_1 \leftarrow n - n_2$
- $m \leftarrow \sum_{i=0}^{n_1-1} M_i$
- $\text{ORCompact}(D_{0 \ldots n_1-1}, M_{0..n_2-1})$
- $\text{OROffCompact}(D_{n_1..n-1}, M_{n_1..n-1}, n_2 - n_1 + m)$
- **for** $i \leftarrow 0 \ldots n_1$:
  - $\text{OSwap}(D_i, D_{i+n_2}, [i >= m])$

ORCompact (,)

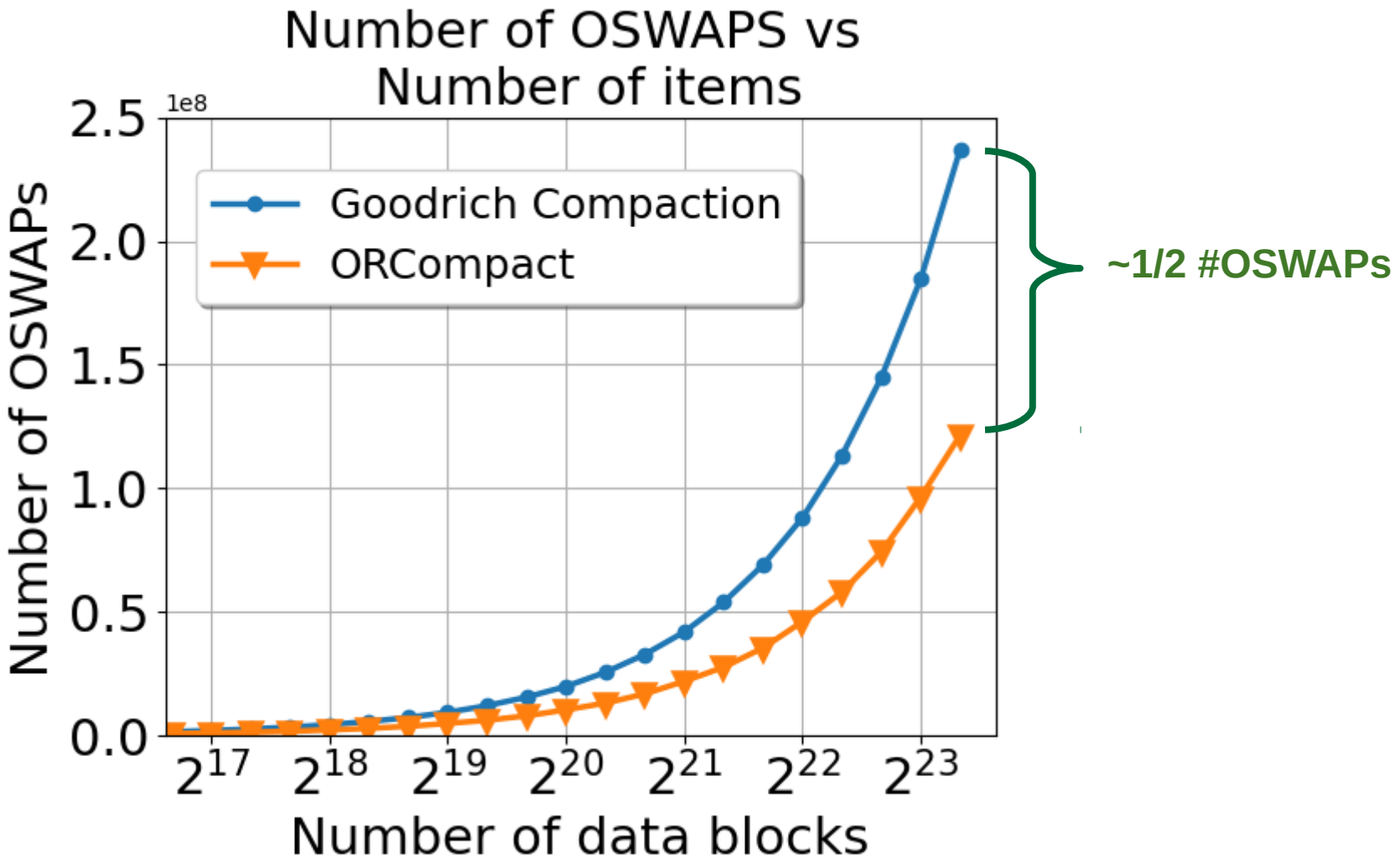
$n_1 = 3$
$m = 1$

OROffCompact (,,z)

$n_2 = 4$
$z = 2$

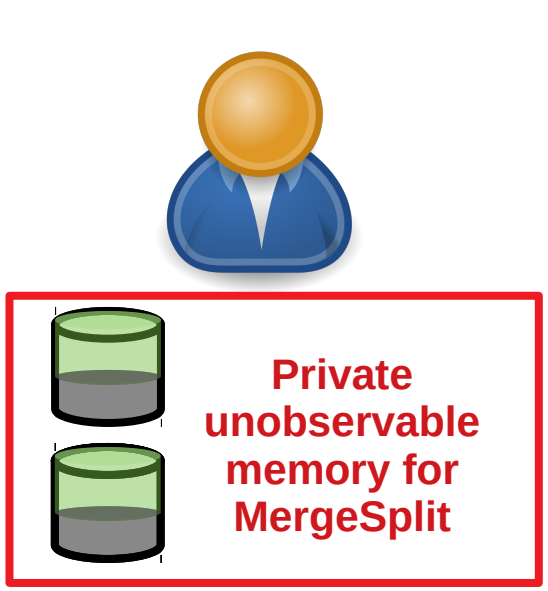


ORCompact OSWAP Evaluation

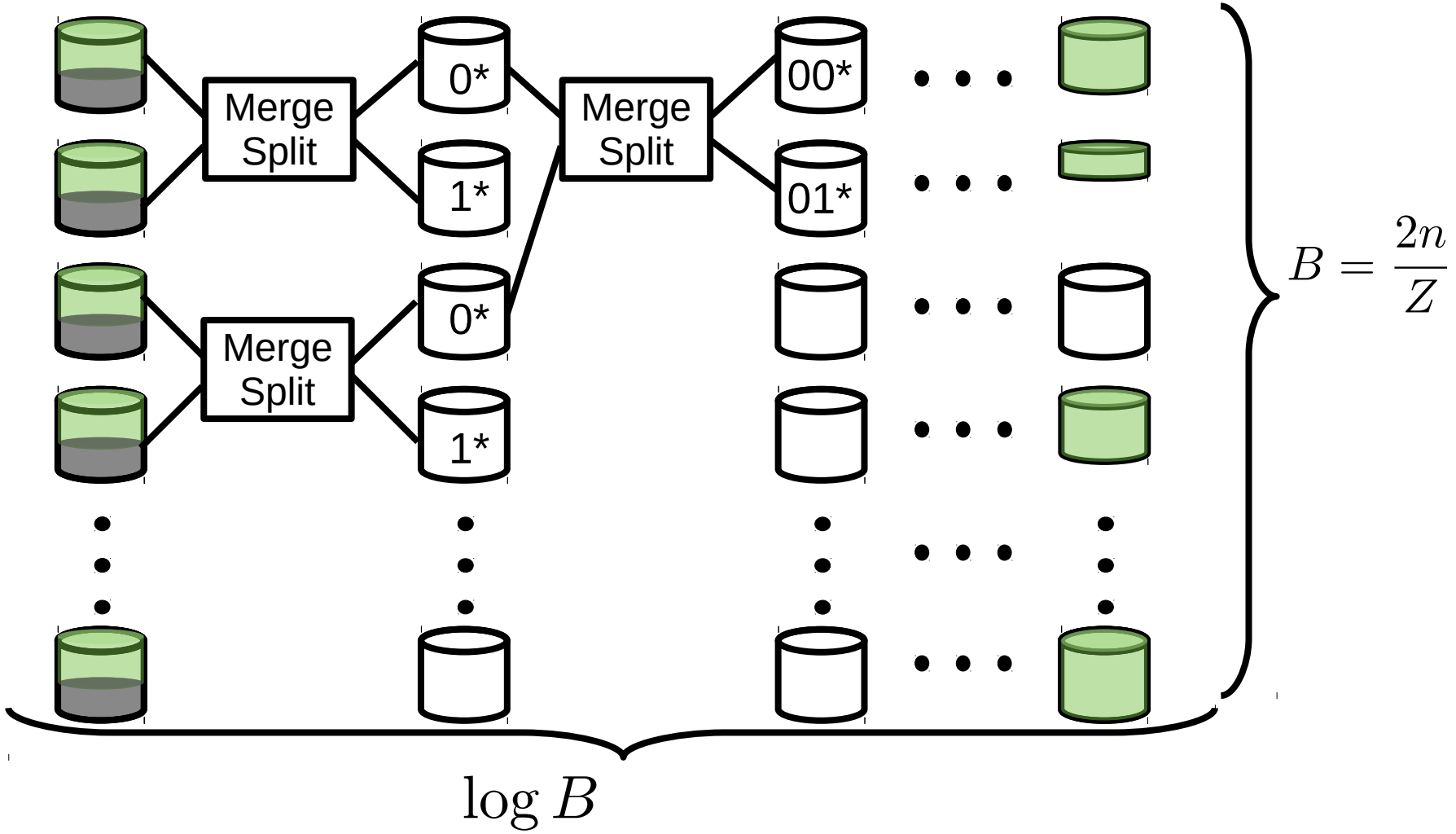


Bucket Oblivious Random Permutation (BORP)

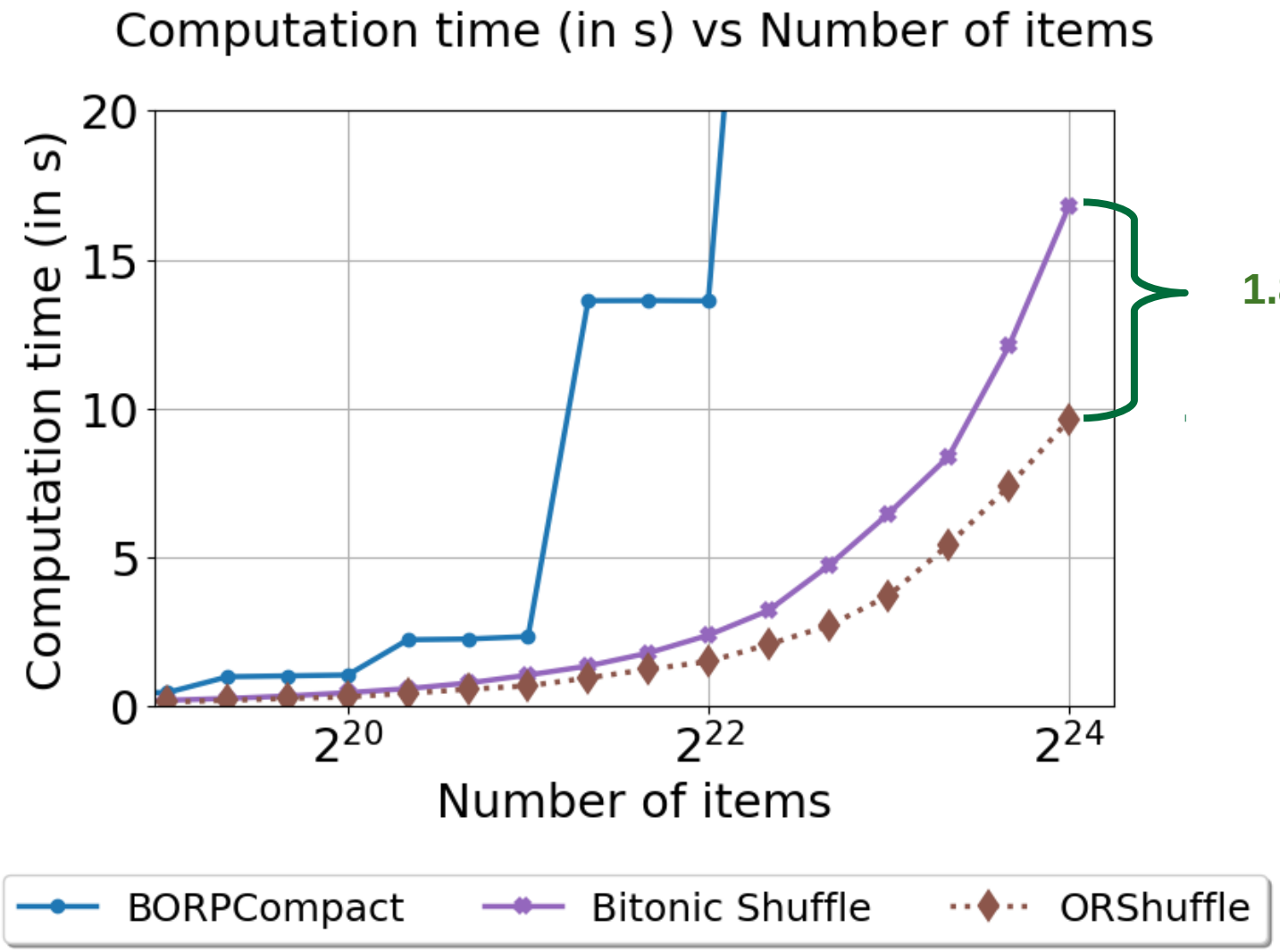
- O (n logn) shuffle algorithm
- Client-Server model, with private unobservable memory on client side



At layer i, MergeSplits are performed on buckets that are 2^i apart, based on the i^{th} MSB



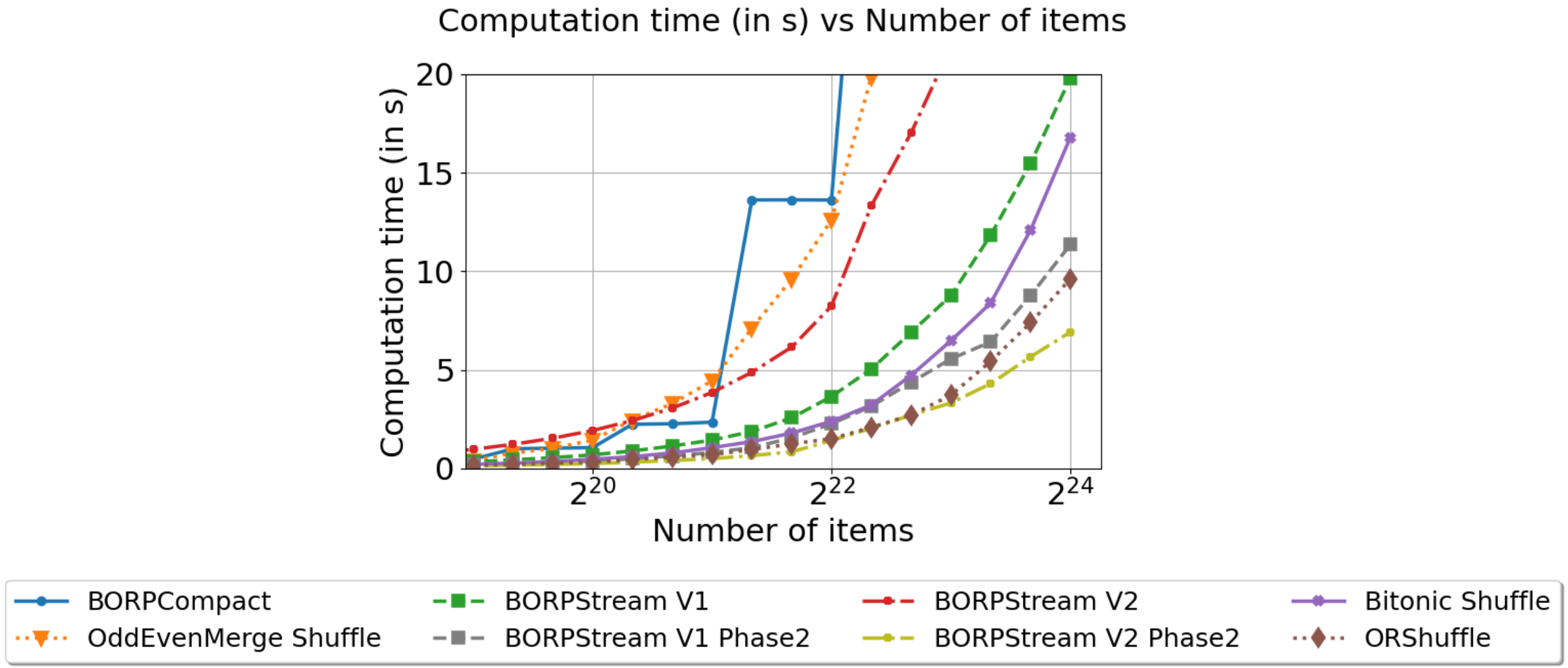
Shuffle Algorithms Evaluation



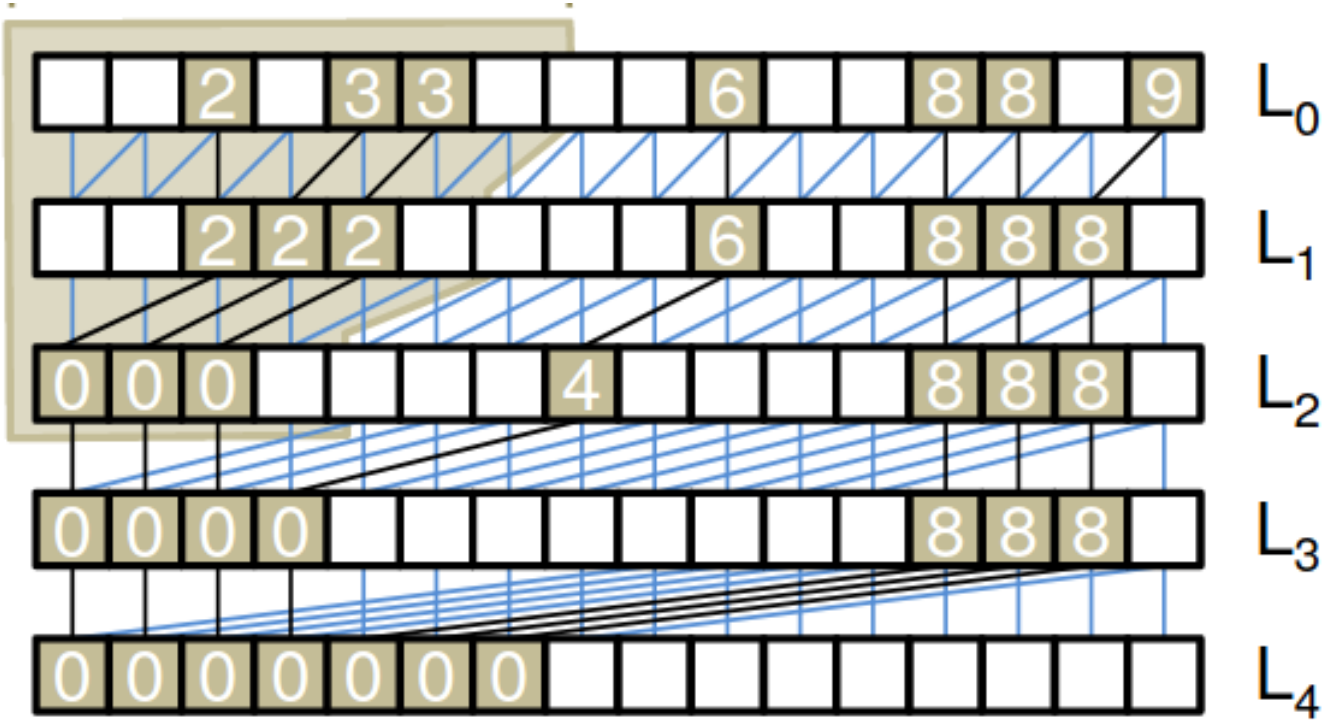
With a fixed block size
b = 8 bytes

1.8x improvement

Shuffle Algorithms Evaluation



Goodrich Oblivious Compaction



BORPStream Parameters

Table 1: BORPStream parameters produced by the optimizer for V1 (minimize total time) and V2 (minimize phase 2 time) modes across sample problem sizes n and block sizes b.

Mode	n	b	f	d	s	λ
V1	2^{20}	8	2	1	32	81
V1	2^{20}	1024	4	3	47	82
V1	2^{24}	8	4	1	48	82
V2	2^{20}	8	4	4	47	80
V2	2^{20}	1024	4	4	47	80
V2	2^{24}	8	4	5	49	82

BORPStream OSWAP and Efficiency

- BORPStream V2 incurs in total 4x more OSWAPS than BitonicShuffle
  - ➔ Phase2 alone only incurs about 1/2 the number of OSWAPS as Bitonic and ORShuffle

- PRM is 90 MB; using more incurs expensive PRM paging overheads
- BORPStream is more locality efficient than BitonicShuffle/ORShuffle
  - ➔ Phase1: PRM stores just memory corresponding to the MSNs
  - ➔ Phase2: BORPStream operates over each bucket at a time

Trusted Execution Environments (TEEs)

- TEEs enable *secure* execution of programs on a remote server; secure ⟶ confidentiality and integrity guarantees

- Data within PRM remain encrypted at all times
- P can have its own key pair enabling users to send private data to P, that only P can decrypt.
- Enables mutually mistrusting parties to share and compute on data with privacy guarantees

AMD SEV-SNP

TrustZone® System Security by ARM

intel SGX

Keystone