Efficient Permutation Correlations and Batched Random Access for Two-Party Computation

Stan Peceny

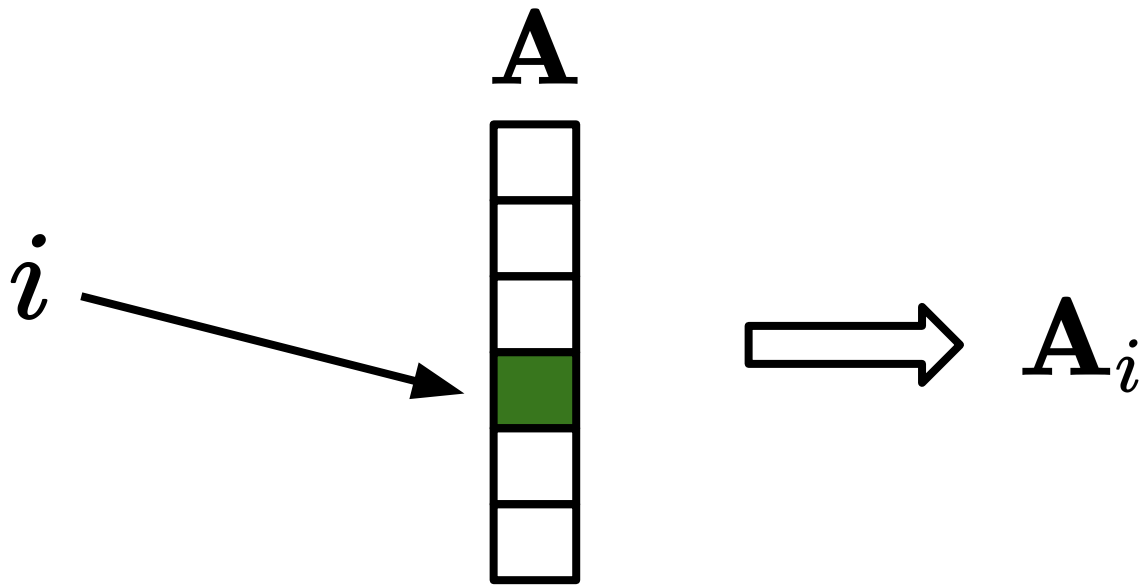
Joint work with

Srini Raghuraman, Peter Rindal, Harshal Shah

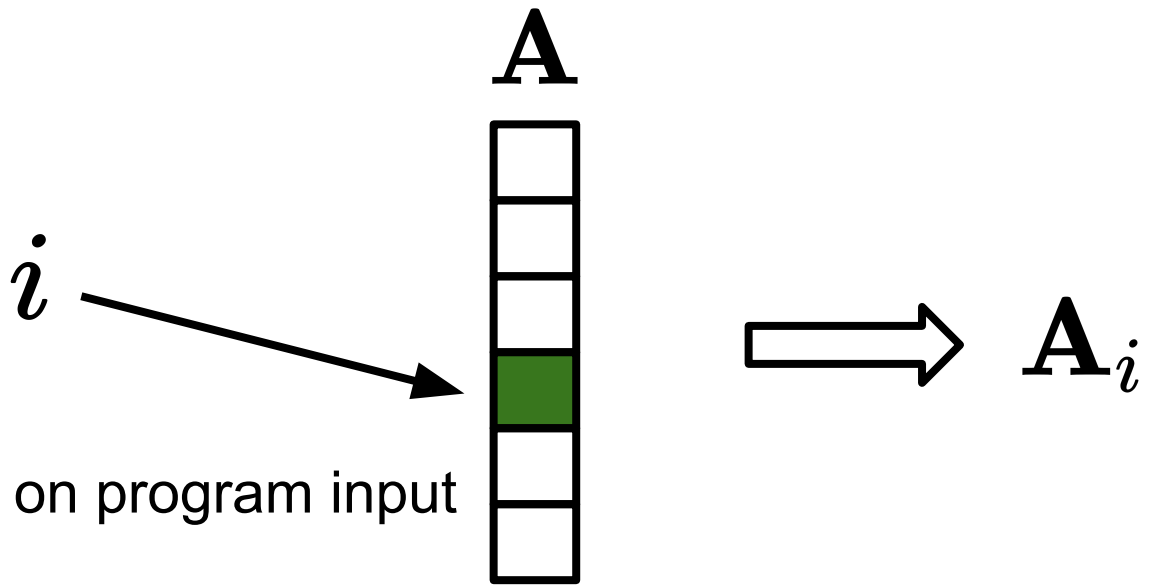
Georgia Institute of Technology

VISA

Array Access

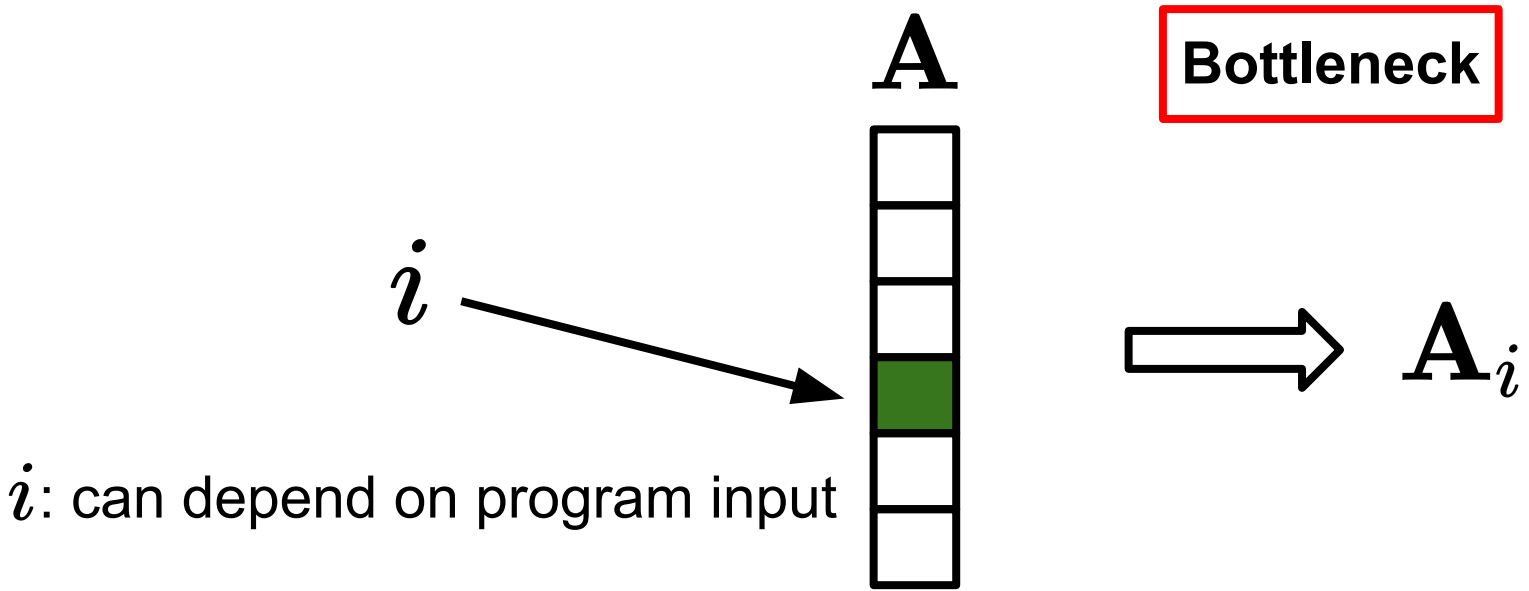


Array Access

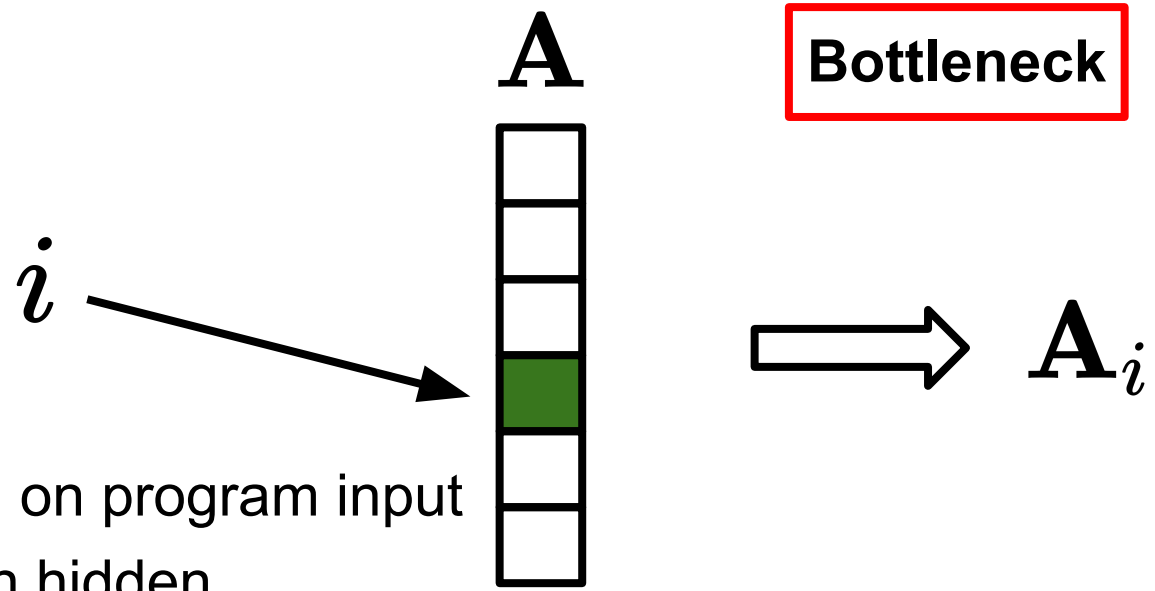


i: can depend on program input

Array Access

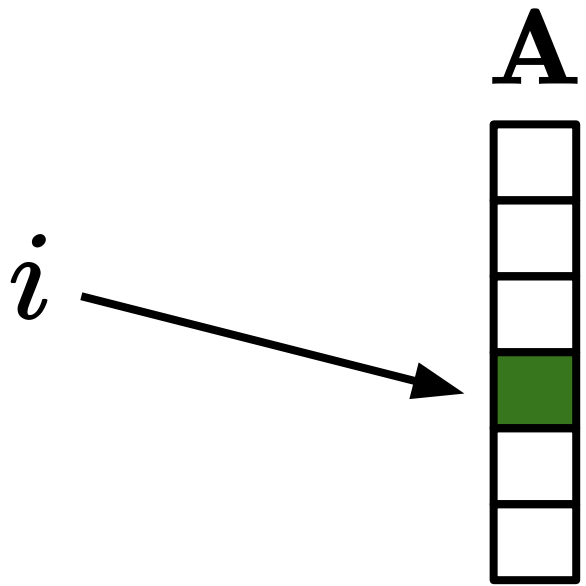


Array Access

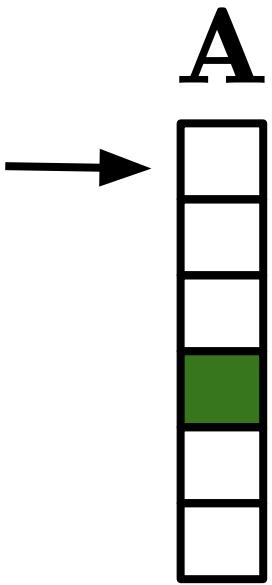


i: can depend on program input
must remain hidden

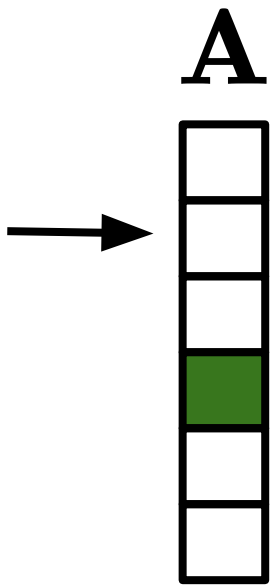
Linear Scan



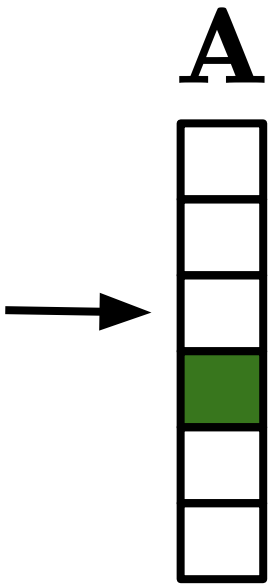
Linear Scan



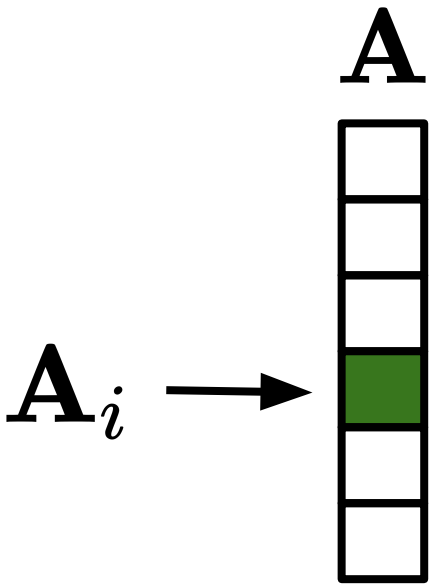
Linear Scan



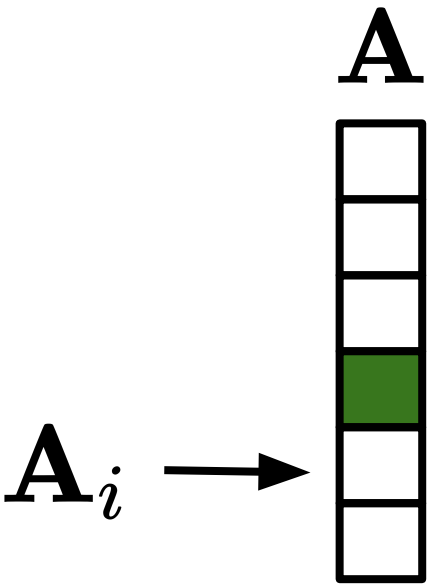
Linear Scan



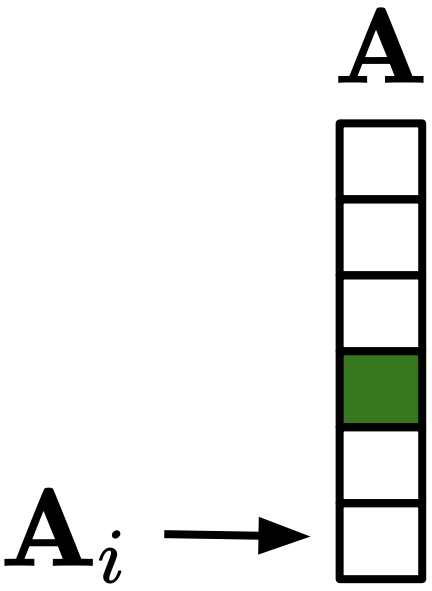
Linear Scan



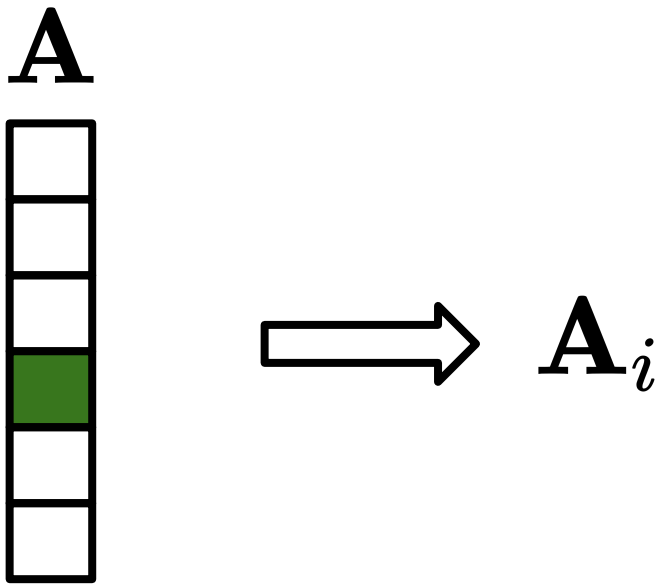
Linear Scan



Linear Scan

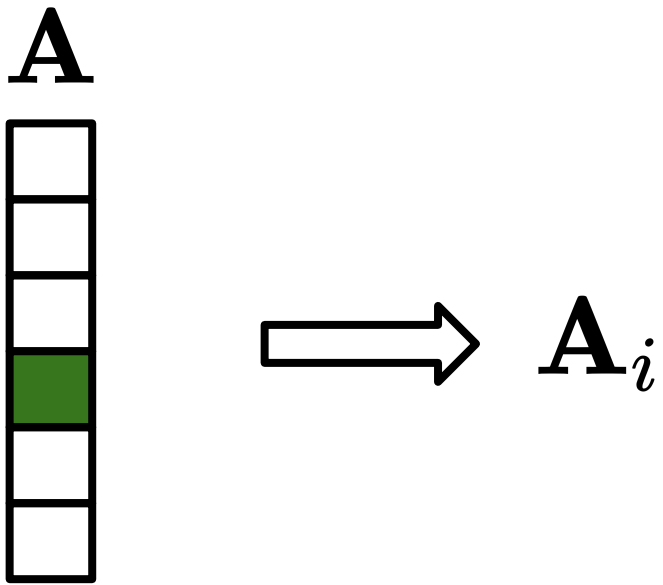


Linear Scan



Linear Scan

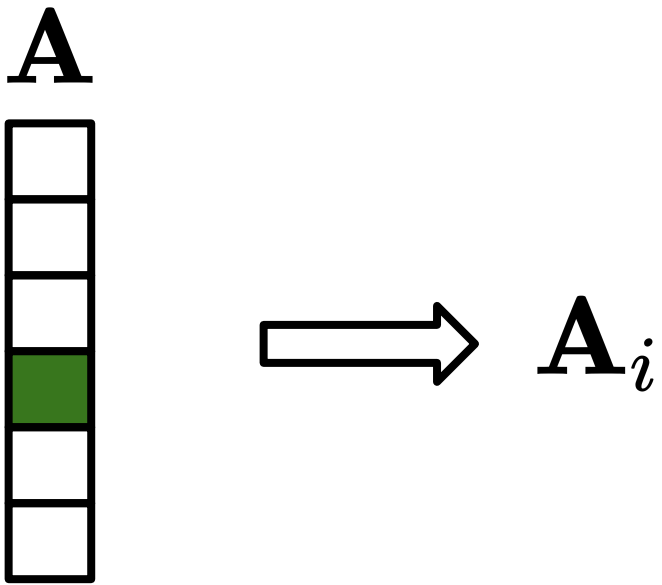
Linear Overhead



Linear Scan

Linear Overhead

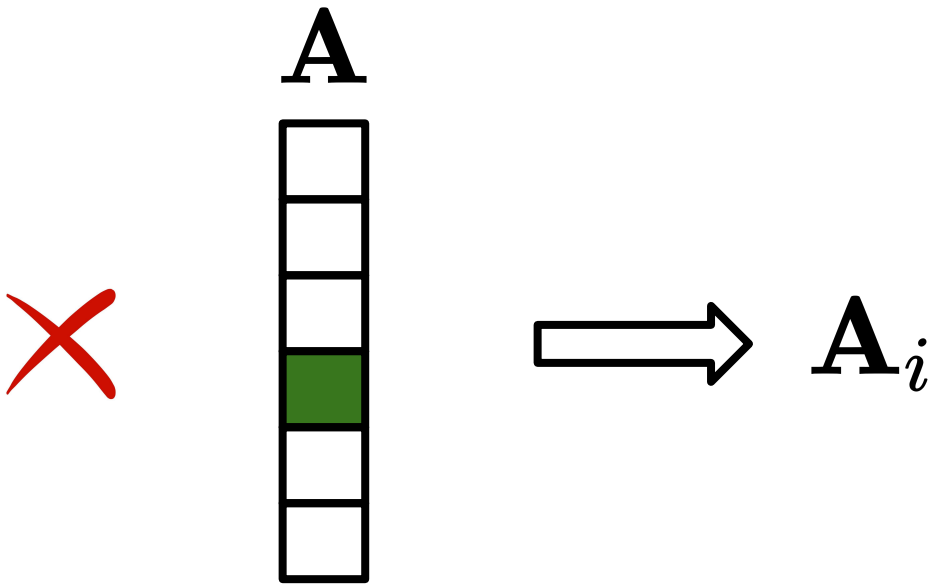
Low Constants



Linear Scan

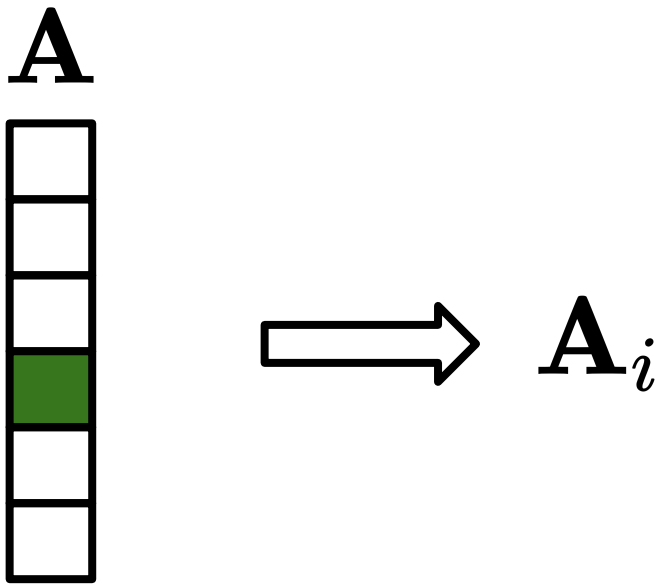
Linear Overhead

Low Constants



Oblivious RAM

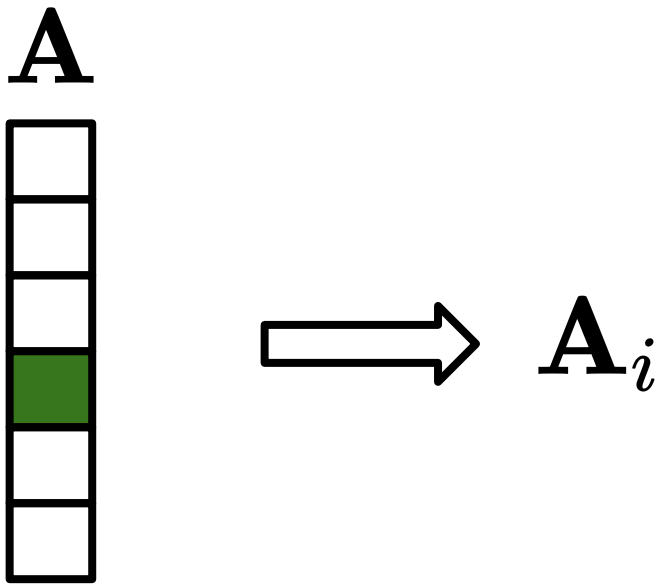
Polylog Overhead



Oblivious RAM

Polylog Overhead

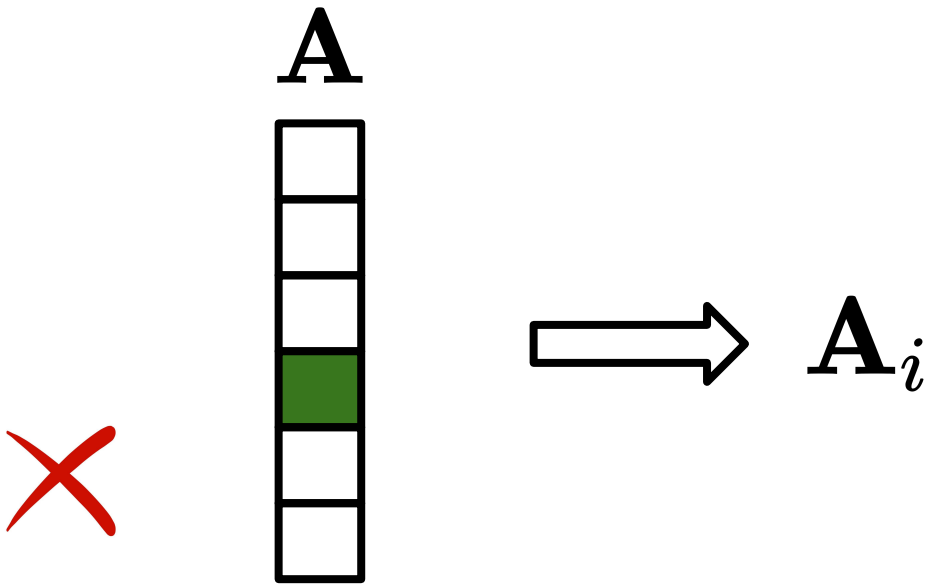
High Constants



Oblivious RAM

Polylog Overhead

High Constants



Oblivious RAM

Secret Sharing Schemes

Garbled RAM

Oblivious RAM

Secret Sharing Schemes

- Mostly sequential accesses
- Adds polylog rounds

Garbled RAM

Oblivious RAM

Secret Sharing Schemes

- Mostly sequential accesses
- Adds polylog rounds

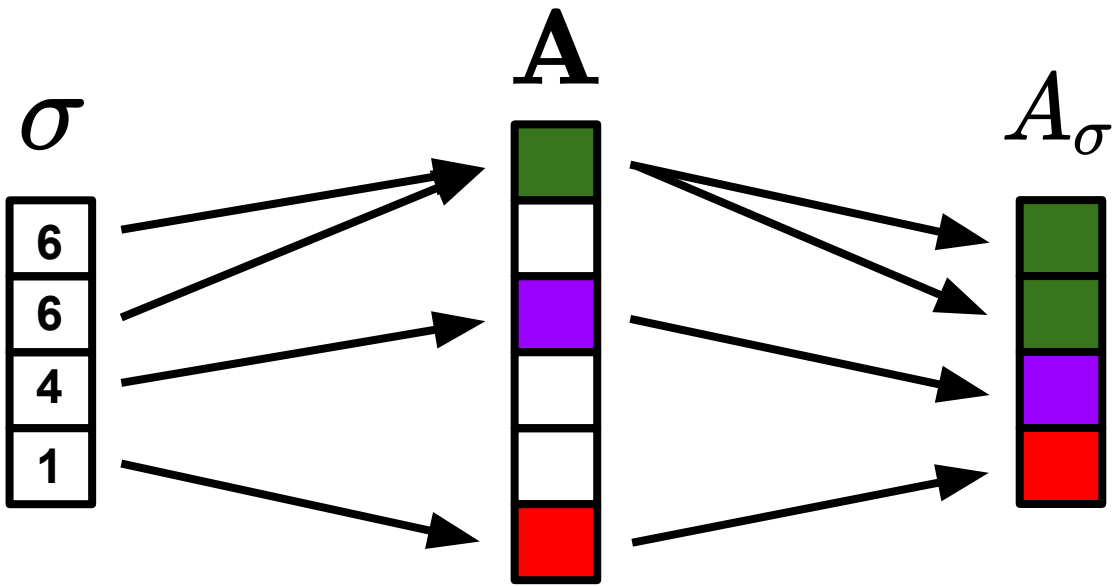
Garbled RAM

- Constant rounds
- κ overhead

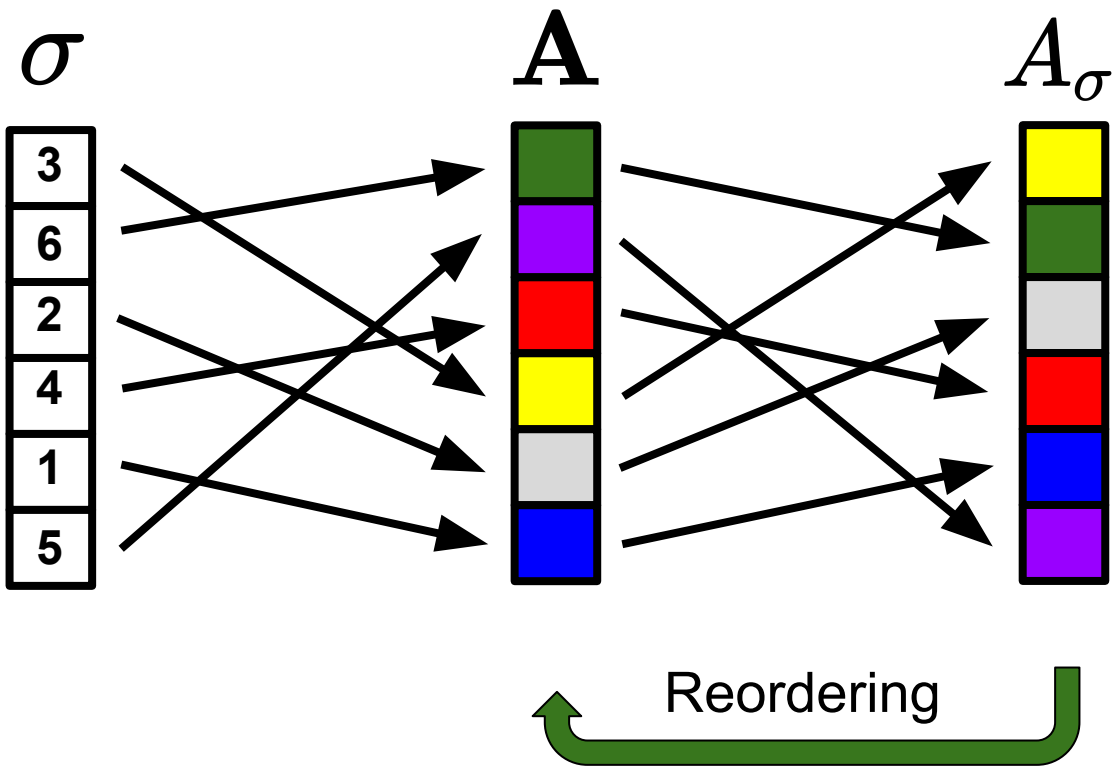
Batched Array Access

Many MPC applications with RAM can be implemented in a **batched** manner

Batched Array Access



σ a **Permutation**



Permutation

A key primitive for batched RAM

We show they allow for more complex batched RAM

Permutation

A key primitive for batched RAM

We show they allow for more complex batched RAM

Amortized O(1) overhead per access

Contribution

Suite of 2-party semi-honest protocols:

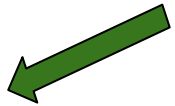
Contribution

Suite of 2-party semi-honest protocols:

- 2 novel permutation correlations
- Expressive batched RAM read/write gates
- Toolbox: derandomization protocols, extraction, sort

Contribution

Suite of 2-party semi-honest protocols:

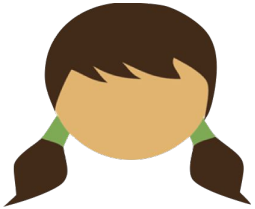


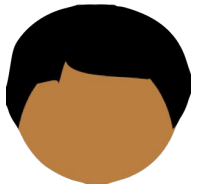
2 novel permutation correlations

Expressive batched RAM read/write gates

Toolbox: derandomization protocols, extraction, sort

Permutation Correlation

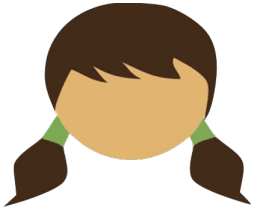


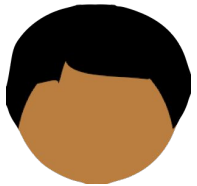


π

$\mathcal{F}_{perm}$

Permutation Correlation





$\mathcal{F}_{perm}$

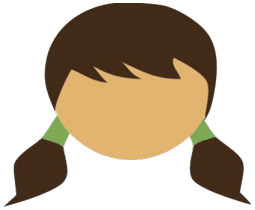
π

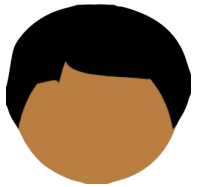
(A, B)

C

$$\pi(A) = B + C$$

Permutation Correlation





π

$\mathcal{F}_{perm}$

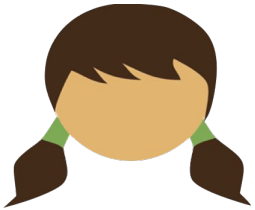
(A, B)

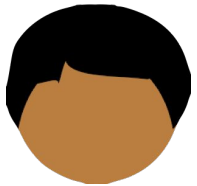
C

$$\pi(A) = B + C$$

Permutation-equivalent to random OLE in MPC

Permutation Correlation





$\mathcal{F}_{perm}$

π

(A, B)

C

$$\pi(A) = B + C$$

Permutation equivalent to random OLE in MPC

‘Share translation’ in [CGP20]

2 Novel Protocols

Weak PRF-based

- O(nl) comm., 3 rounds
- 7.3 seconds and 182MB for $n=2^{20}$, $l=128$

2 Novel Protocols

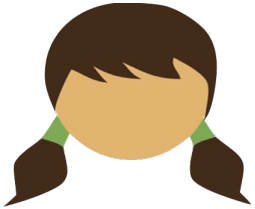
Weak PRF-based

- O(nl) comm., 3 rounds
- 7.3 seconds and 182MB for $n=2^{20}$, $l=128$

PCG-based

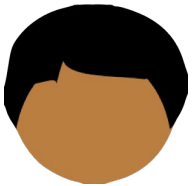
- First sublinear protocol
- O(n log l) comm., O(log (l/κ^2)) rounds
- Higher constants

Weak PRF



k

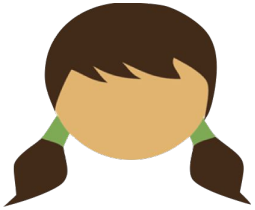
$\mathcal{F}_{wprf}$



x

x is random

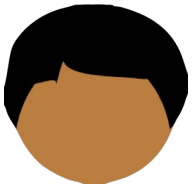
Weak PRF



k

$[F_k(x)]_A$

$\mathcal{F}_{wprf}$

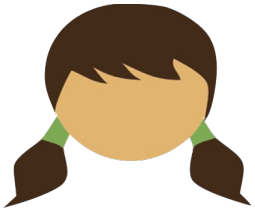


x

$[F_k(x)]_B$

x is random

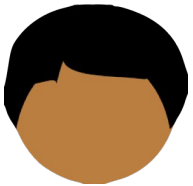
Weak PRF



k

$[F_k(x)]_A$

$\mathcal{F}_{wprf}$



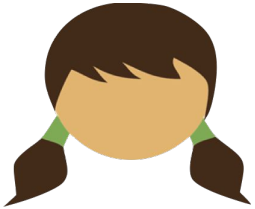
x

$[F_k(x)]_B$

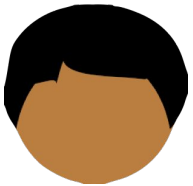
x is random

Alternating Moduli [APRR24], LowMC [ARS+15]

wPRF Permutation Correlation

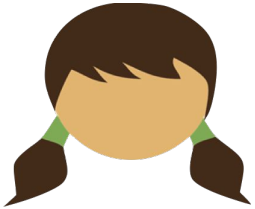


$k \leftarrow \mathcal{K}$



π

wPRF Permutation Correlation

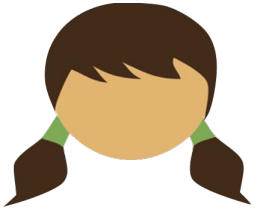


$k \leftarrow \mathcal{K}$

$A_i := F_k(i)$

π

wPRF Permutation Correlation



$k \leftarrow \mathcal{K}$

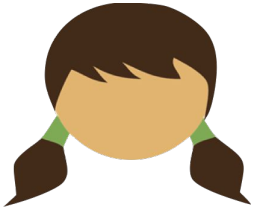
π

$A_i := F_k(i)$

k $\rightleftharpoons$ Π_{wprf} $\rightleftharpoons$ $\pi(i)$

B_i $\rightleftharpoons$ C_i

wPRF Permutation Correlation



$k \leftarrow \mathcal{K}$

π

$A_i := F_k(i)$

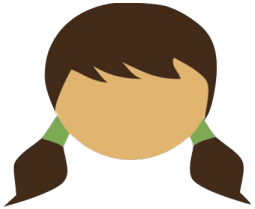
k → ← $\pi(i)$

Π_{wprf}

B_i ← → C_i

$i = 1, \ldots, n$

wPRF Permutation Correlation



$k \leftarrow \mathcal{K}$

π

$A_i := F_k(i)$

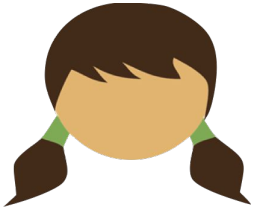
k → ← $\pi(i)$

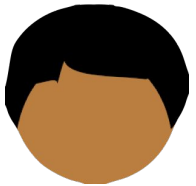
Π_{wprf}

B_i ← → C_i

$$\pi(A) = B + C$$

wPRF Permutation Correlation





$k \leftarrow \mathcal{K}$

π

$A_i := F_k(i)$

k

B_i

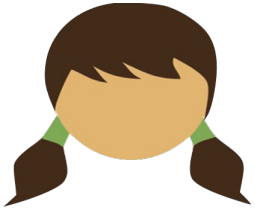
Π_{wprf}

$\pi(i)$

C_i

$$\pi(A) = B + C$$

wPRF Permutation Correlation



$k \leftarrow \mathcal{K}$

π

$A_i := F_k(H(i))$

k

B_i

Π_{wprf}

$H(\pi(i))$

C_i

$$\pi(A) = B + C$$

Ours vs Related Work

Chosen-input, $n=2^{20}$, $l=128$

Ours: 7.3s, 182MB

Chase et al. [CGP20]: 44s, 4GB

Gazelle [JVC18]: 303s, 215MB

Batched Array Access

(1) Batched read and (2) batched write

Multiple reads/write to single index allowed

Batched Array Access

(1) Batched read and (2) batched write

Multiple reads/write to single index allowed

Dominant cost: Sort

O(n log n) time and O(log n) rounds

n accesses with amortized O(1) overhead

Batched Array Access

(1) Batched read and (2) batched write

Multiple reads/write to single index allowed

Dominant cost: Sort

O(n log n) time and O(log n) rounds

n accesses with amortized O(1) overhead

n accesses to $n=2^{16}$ array ~10s and ~160MB

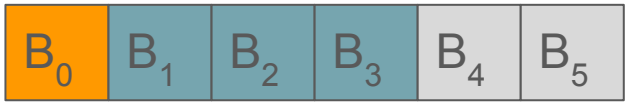
Batched-RAM-Read Tool

Aggregation Tree [BDG+22]:

Input: List of blocks B, List of control bits c

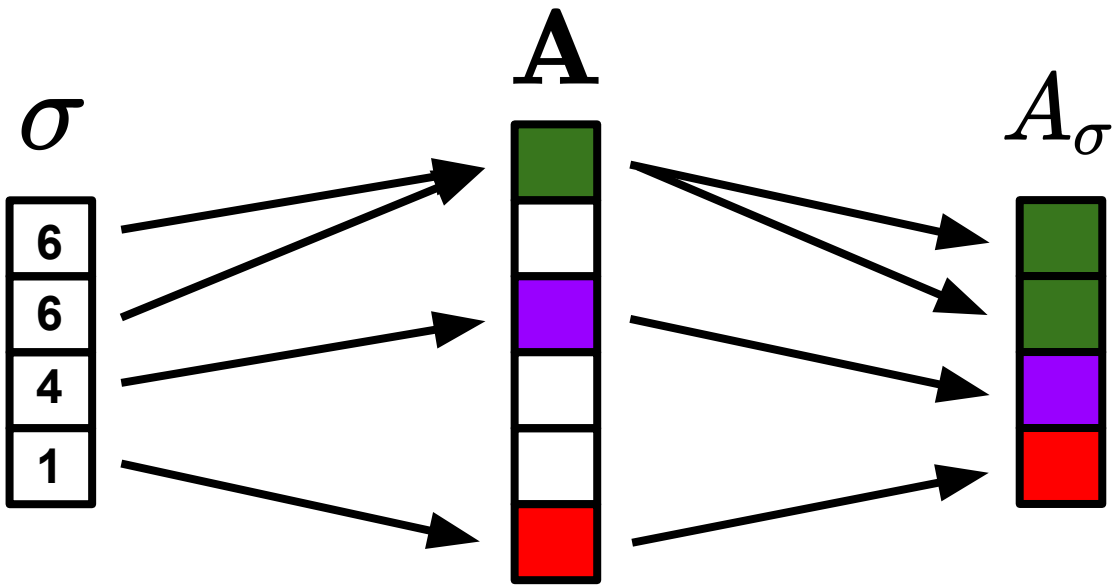
[[B]]: 

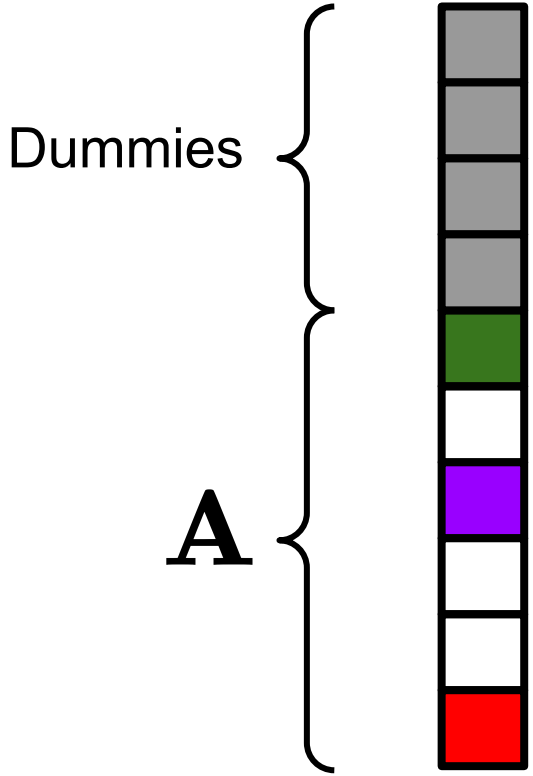
[[c]]: 0 0 1 1 0 1

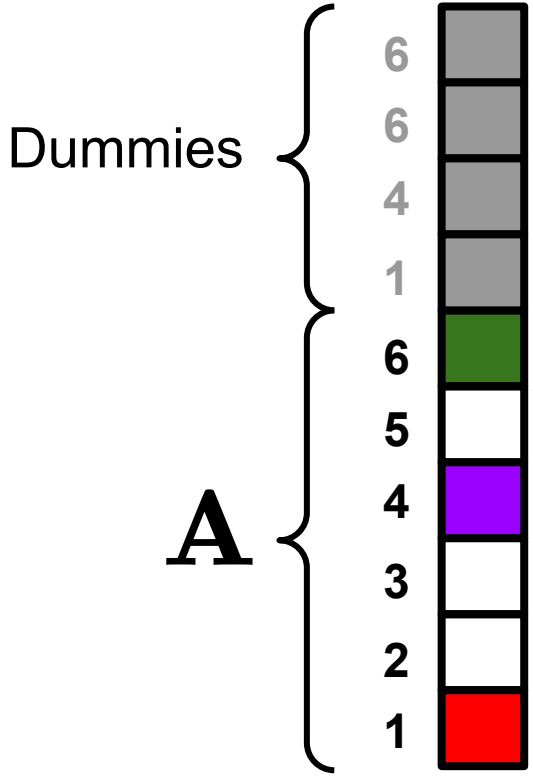
c partitions B into sublists: 

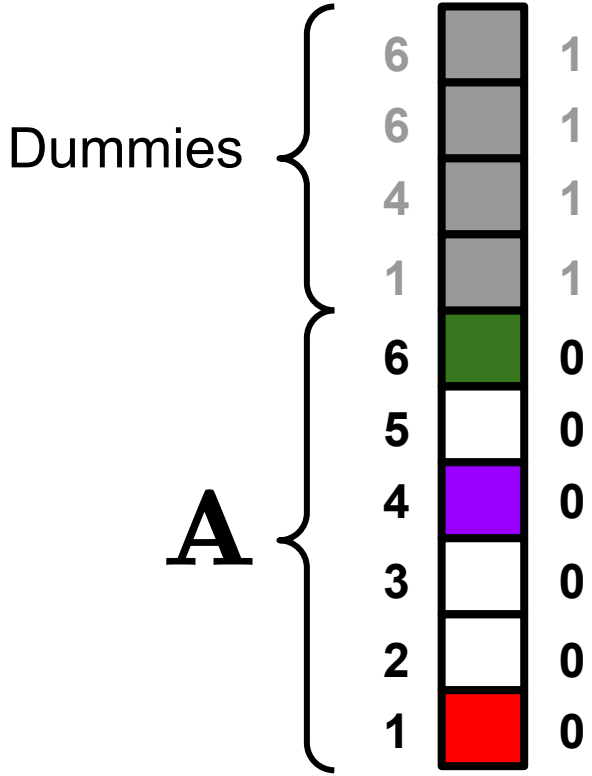
Output: Copy first item of each sublist into rest of sublist

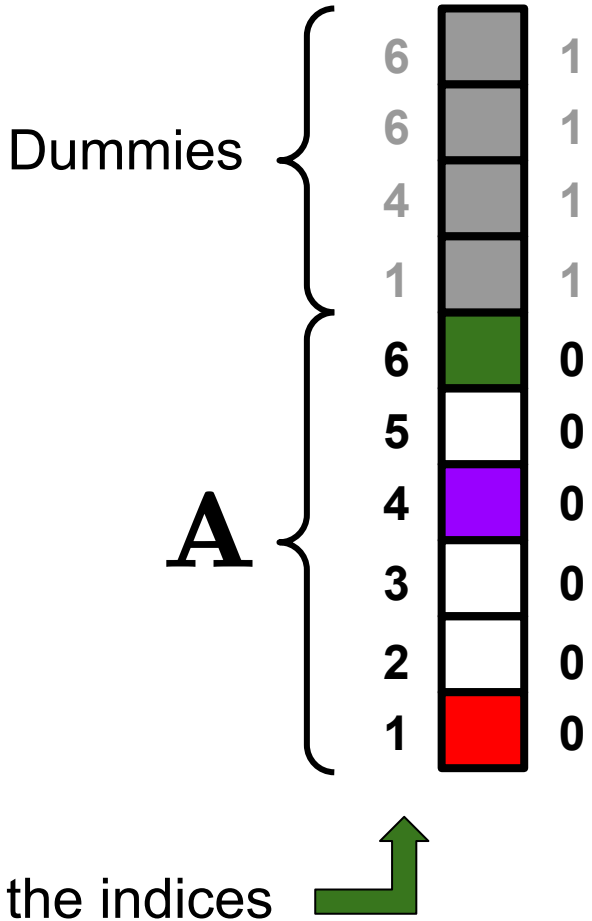
Batched Array Access

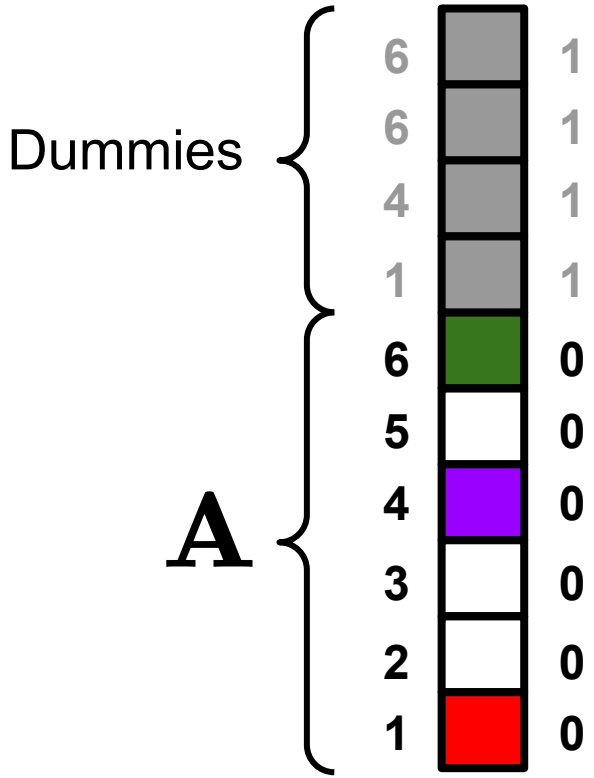



Dummies
A


Dummies
A
6
6
4
1
6
5
4
3
2
1

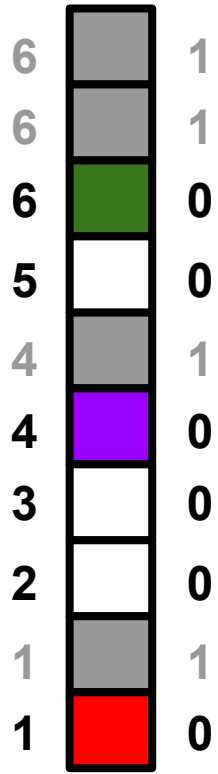

Dummies
A
6
6
4
1
6
5
4
3
2
1
1
1
1
1
0
0
0
0
0
0


Dummies
A
6
6
4
1
6
5
4
3
2
1
1
1
1
1
0
0
0
0
0
0
Stable sort using the indices

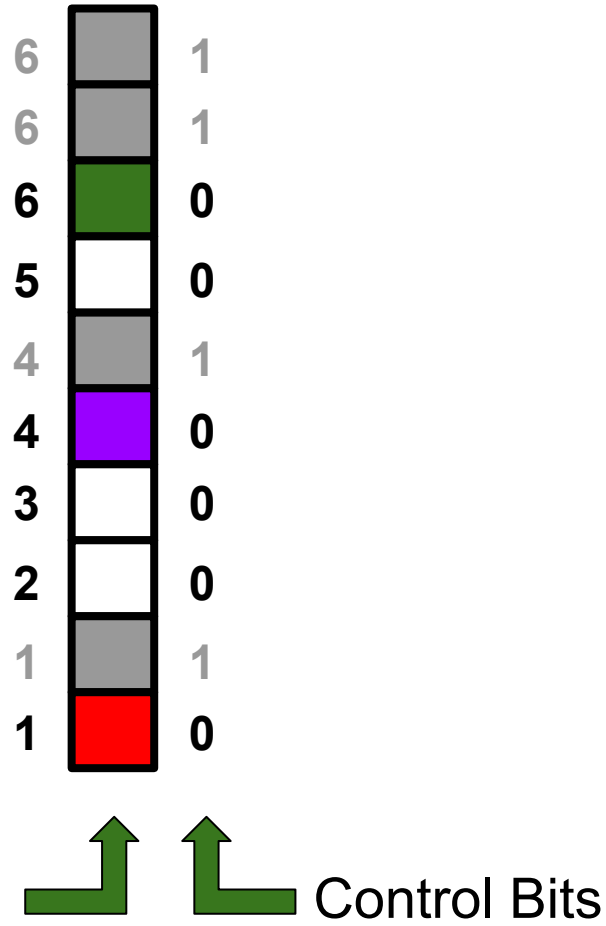


Stable sort using the indices

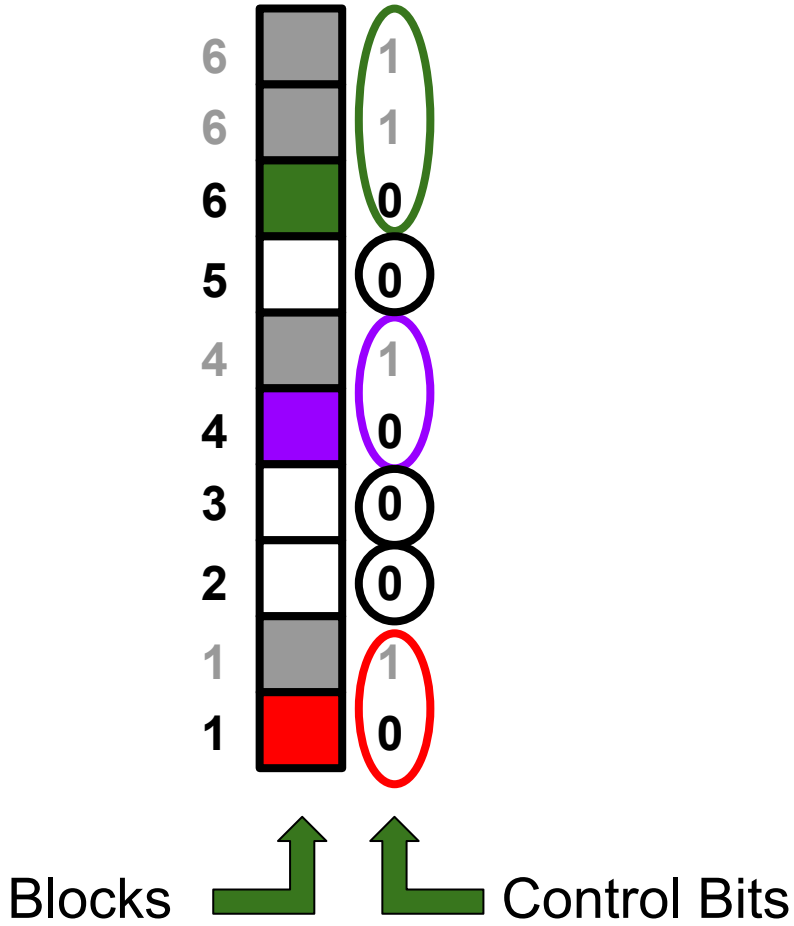
Returns sorting permutation π
Permute with π


6 1
6 1
6 0
5 0
4 1
4 0
3 0
2 0
1 1
1 0

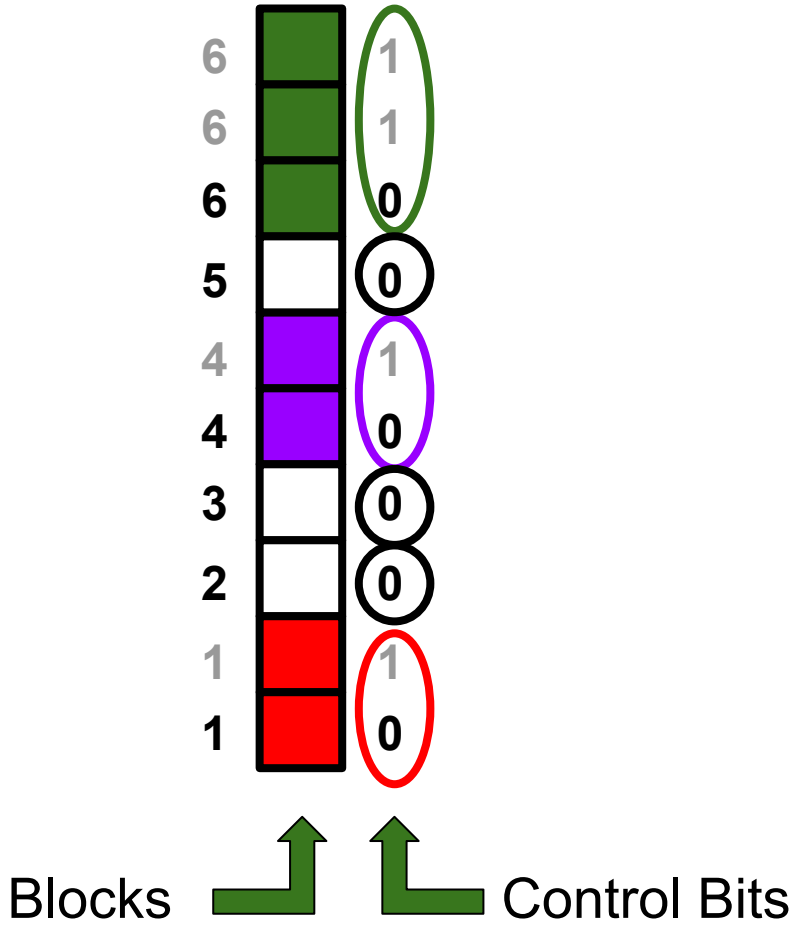
Aggregation Tree

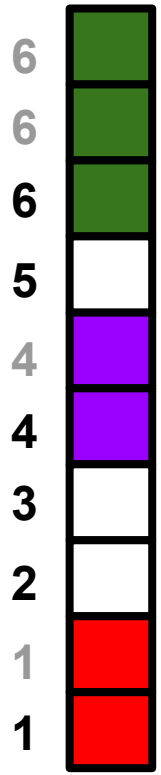


Aggregation Tree

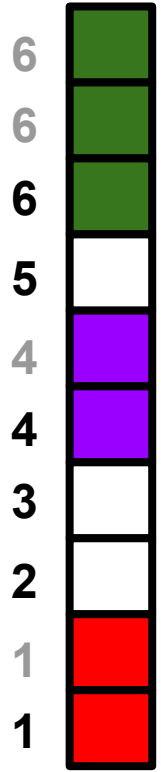


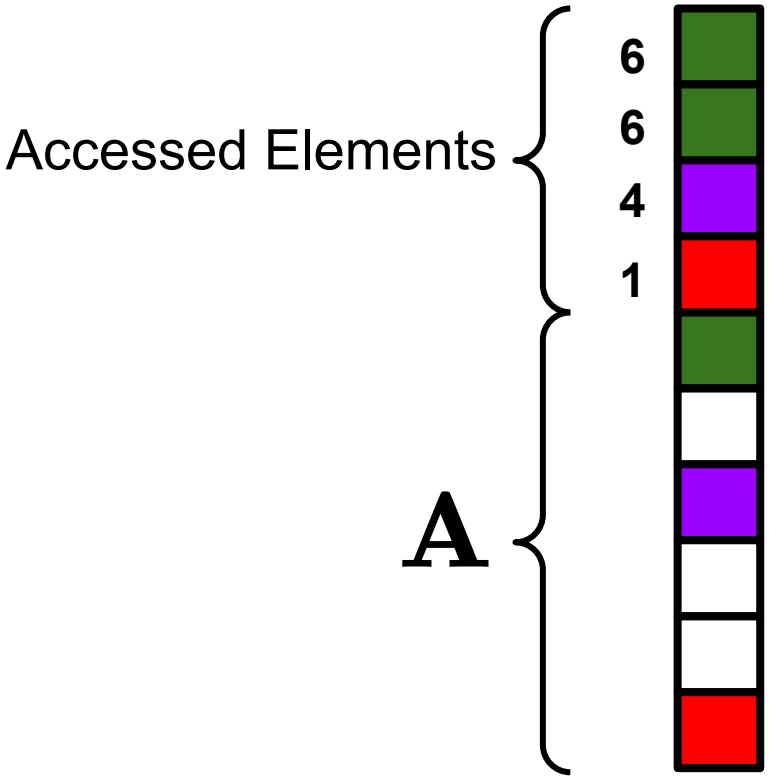
Aggregation Tree




6
6
6
5
4
4
3
2
1
1

Unpermute with π^{-1}




Accessed Elements
6
6
4
1
A

Toolbox

Derandomize:

- Chosen input and permutation
- Input list is secret-shared
- Permutation is secret-shared (as composition)
- Many permutations with 1 correlation
- Inverse permutation

Toolbox

Derandomize:

- Chosen input and permutation
- Input list is secret-shared
- Permutation is secret-shared (as composition)
- Many permutations with 1 correlation
- Inverse permutation

Additive ⇔ composition secret-shared permutation

Sorting and extraction protocols

Efficient Permutation Correlations and Batched Random Access for Two-Party Computation

Suite of 2-party semi-honest protocols:

- 2 novel permutation correlations
- Expressive batched RAM read/write gates
- Toolbox: derandomization protocols, extraction, sort