Revisiting Square Root ORAM

Efficient Random Access in Multi-Party Computation

Samee Zahur
Jack Doerner
David Evans

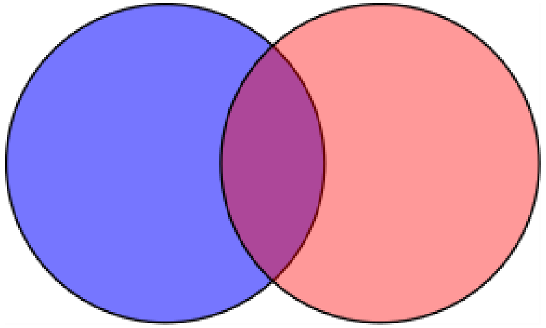
Xiao Wang
Jonathan Katz

Mariana Raykova

Adrià Gascón

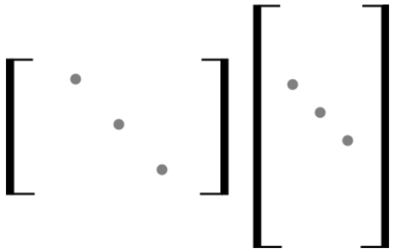
oblivc.org/sqoram

Secure multi-party computation applications

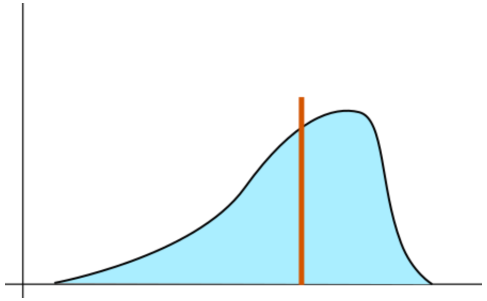


Set intersection
[FNP04]

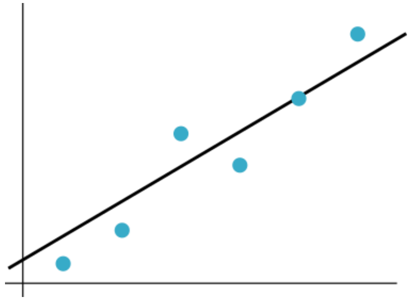
Iris code matching
[LCPLB12]



Matrix factorization for recommendations
[NIWJTB13]

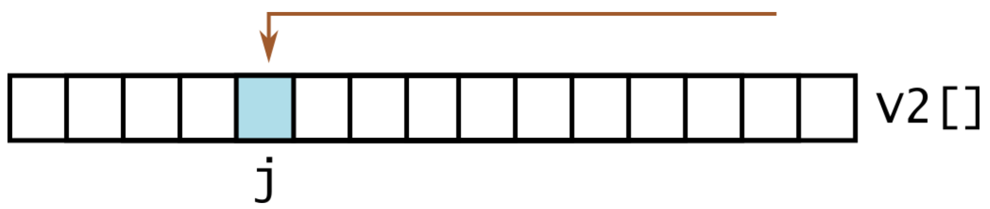


Median computation
[AMP04]



Linear ridge-regression
[NWIJBT13]

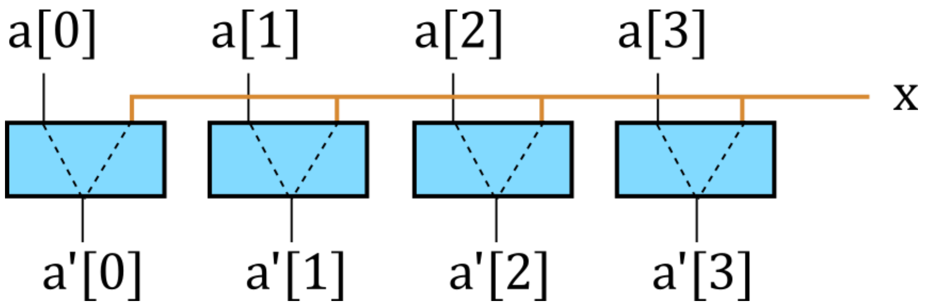
Random Access




```
void oscrypt_smix(obliv uint8_t * B, s
  ...

  for (i = 0; i < N; i += 2) {
    j = integerify(X, r) & (N - 1);
    temp = V2[j];
    xorBits(X,temp,32*r);
    oscrypt_blockmix_salsa8(X, Y, r);

    j = integerify(Y, r) & (N - 1);
    temp = V2[j];

    for (size_t jj = 0; jj < 32 * r; j
      Y[jj] ^= temp[jj];
    }
```

Hiding access pattern

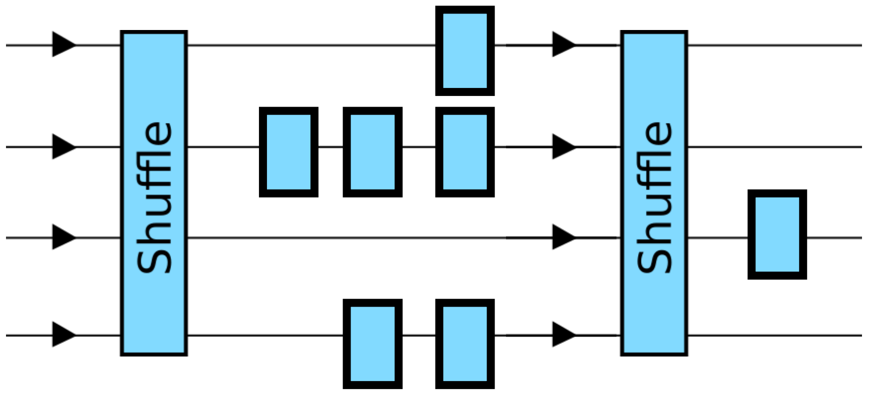
Linear scan



Access every element

Per-access cost: $\Theta(n)$

Oblivious RAM



Continually shuffle elements around

Per-access cost: $\Theta(\log^p n)$

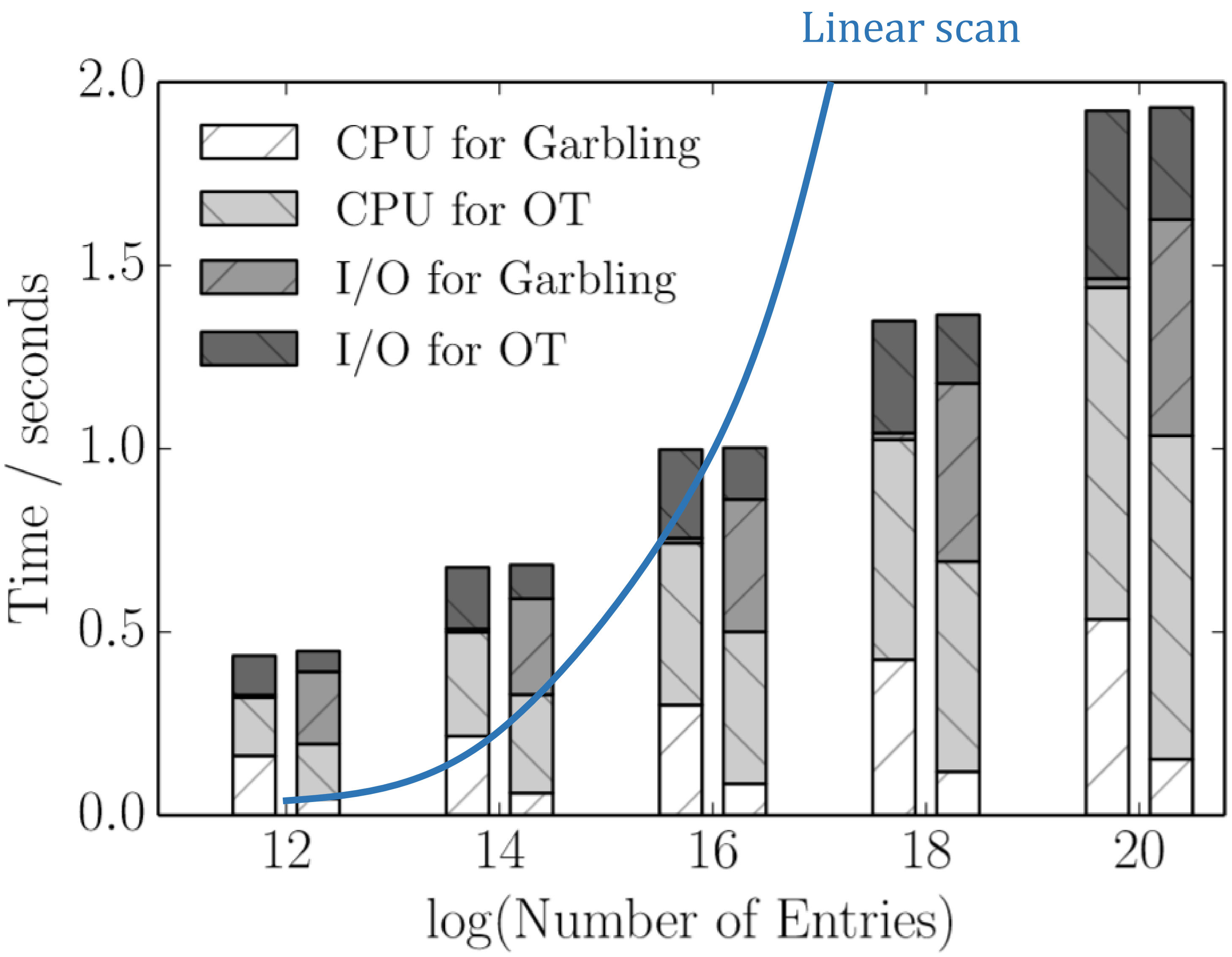
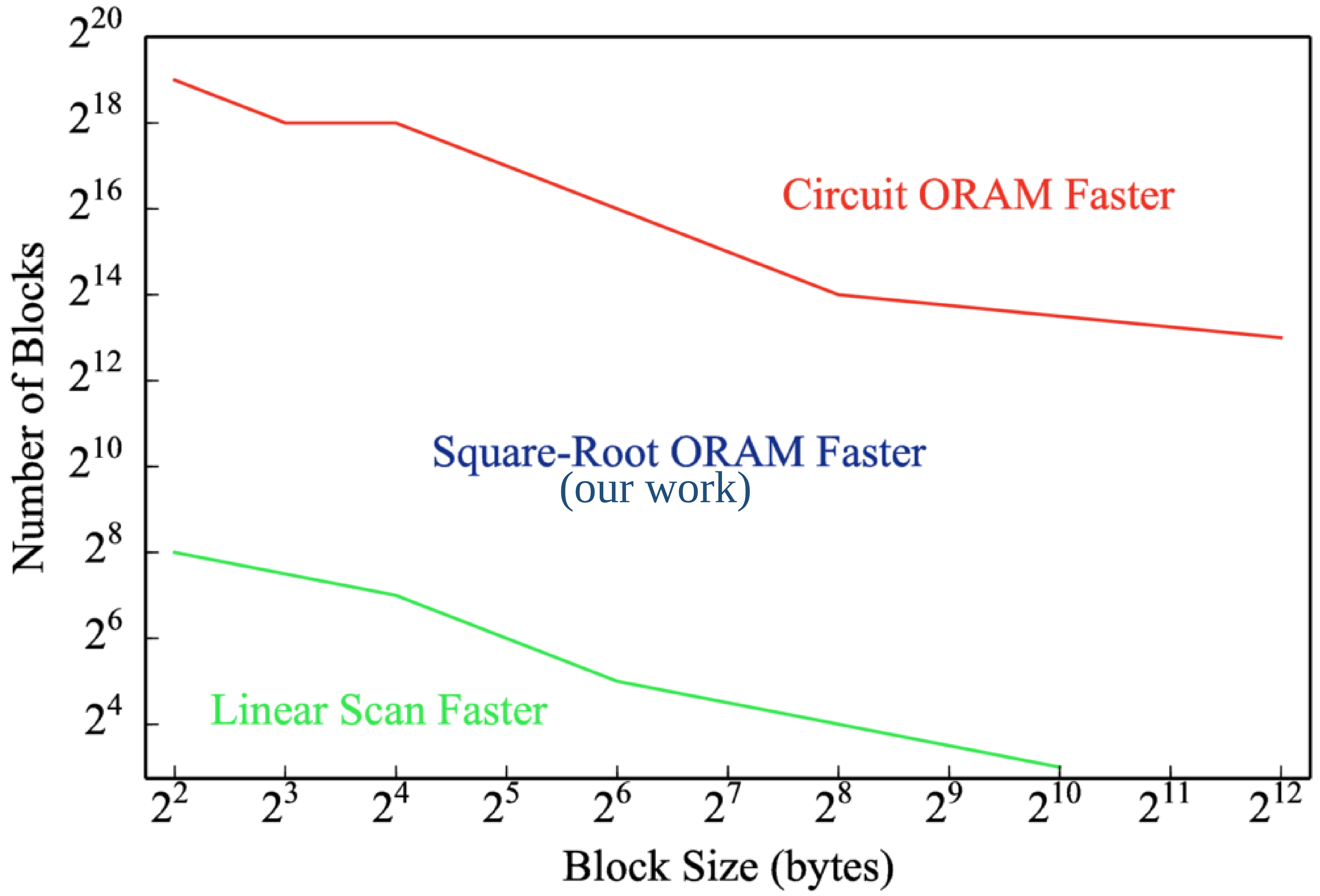


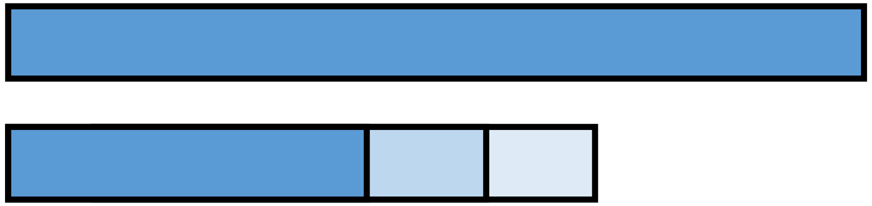
Figure from: Wang, Chan, Shi. **Circuit Oram.** CCS'15



Circuit ORAM Faster
Square-Root ORAM Faster
(our work)
Linear Scan Faster
Number of Blocks
Block Size (bytes)
2^{20} 2^{18} 2^{16} 2^{14} 2^{12} 2^{10} 2^{8} 2^{6} 2^{4}
2^{2} 2^{3} 2^{4} 2^{5} 2^{6} 2^{7} 2^{8} 2^{9} 2^{10} 2^{11} 2^{12}

Approach: revisit old schemes

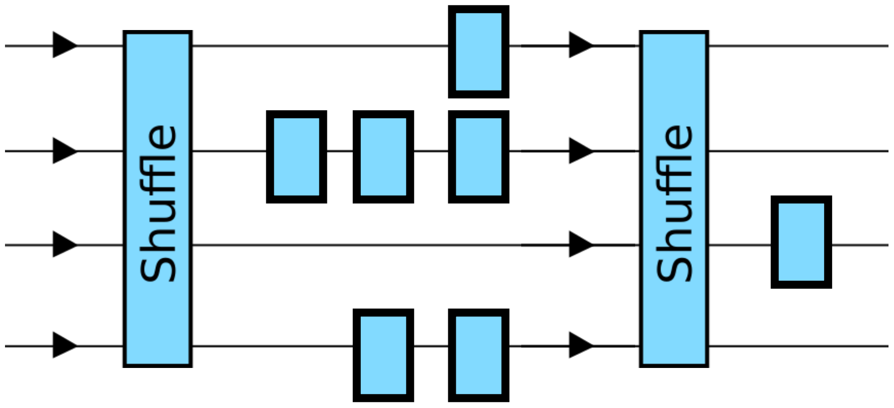
Classic “square root” scheme by Goldreich and Ostrovsky (1996).



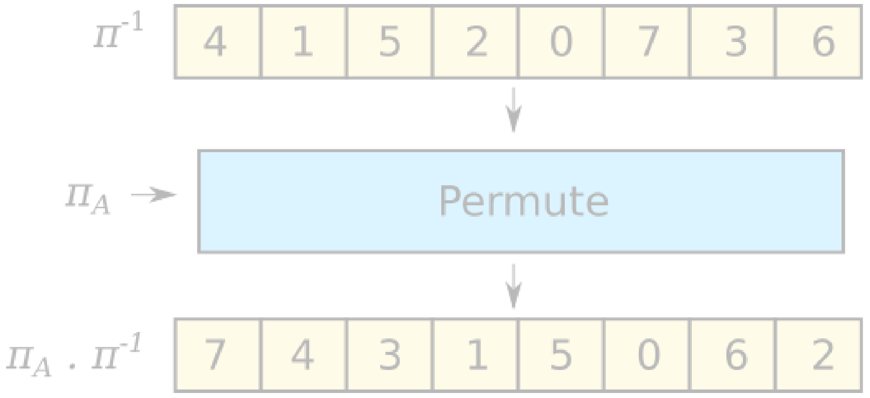
Considered slow for MPC because of per-access hash evaluation.

Per-access amortized cost: $\Theta(\sqrt{n}\log n)$

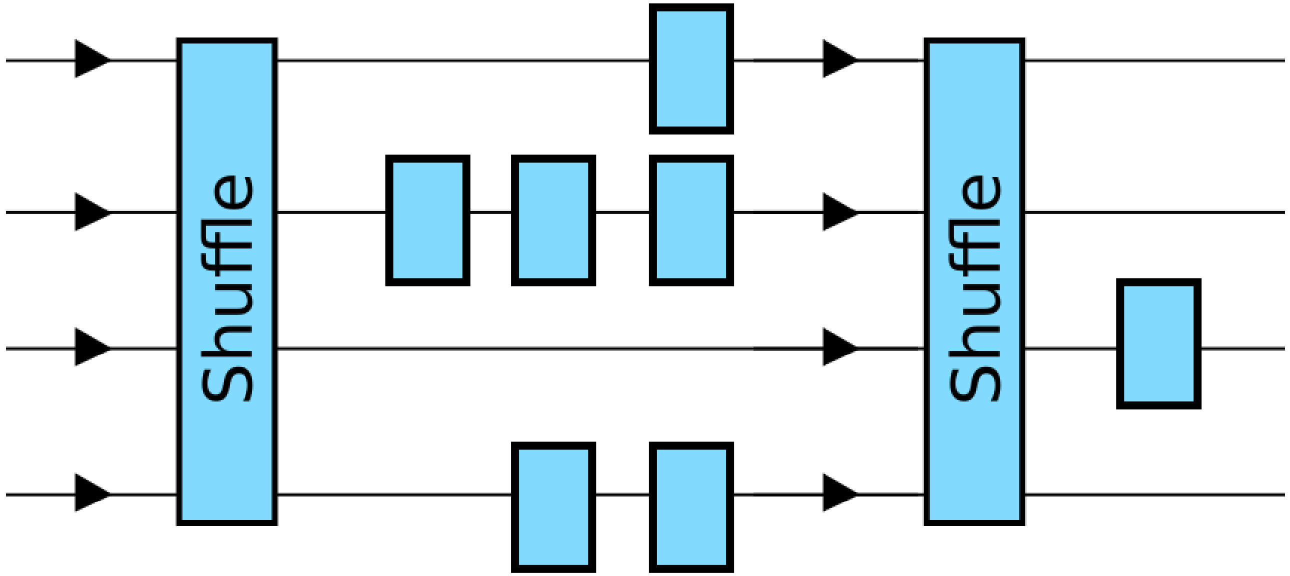
Four-element ORAM



Larger Sizes



4-Block ORAM

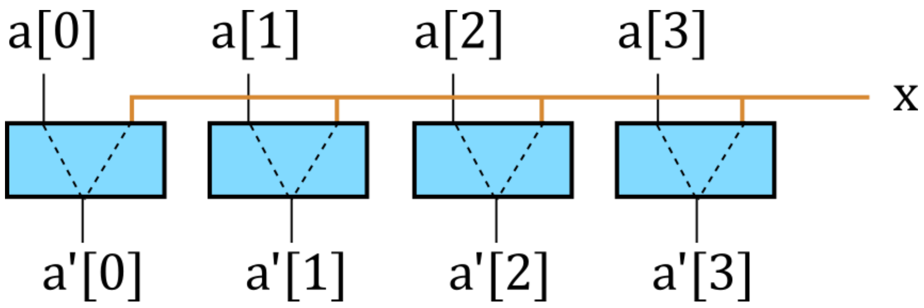


Cost: $5B$ + B +$2B$ +$3B$ + ...

= $11B$ every 3 accesses

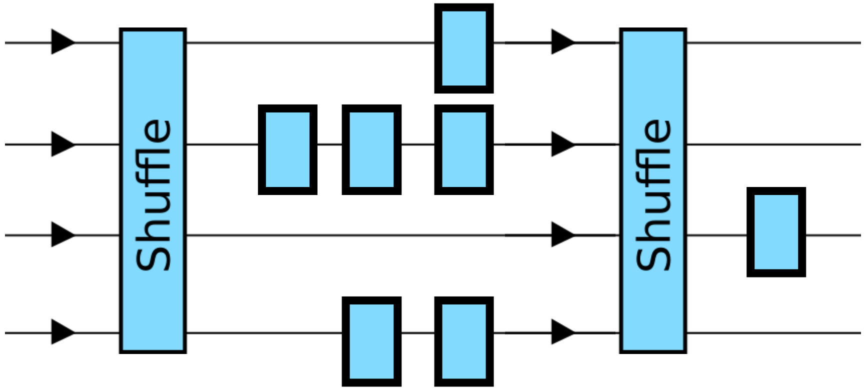
Comparison

Linear scan



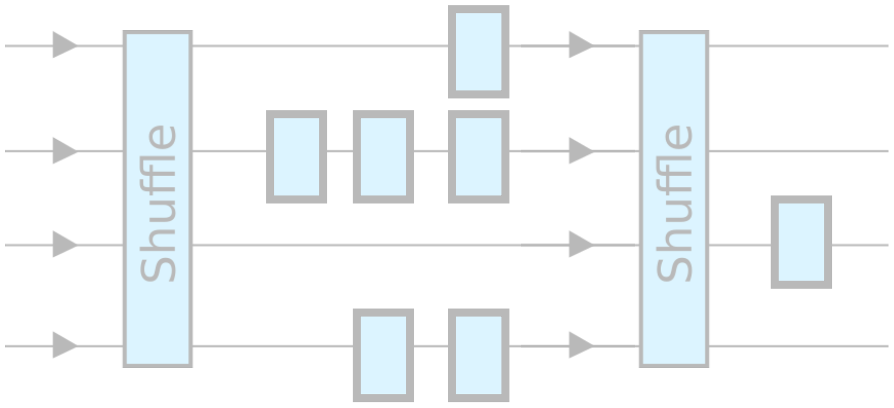
Cost: $4B = 12B/3$

Our scheme

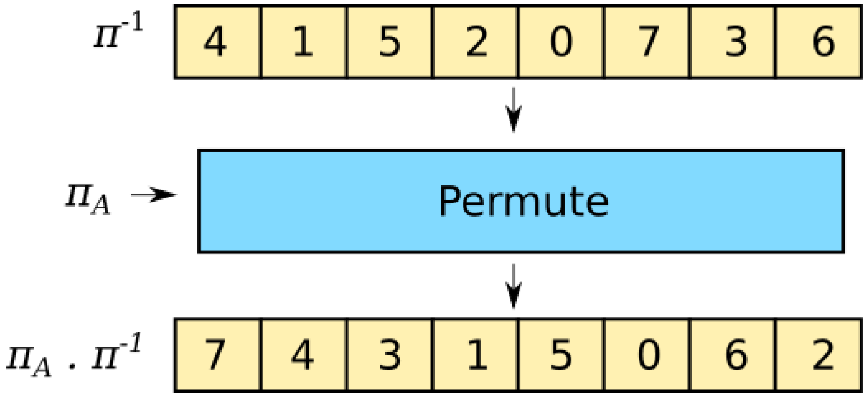


Cost: $11B/3$

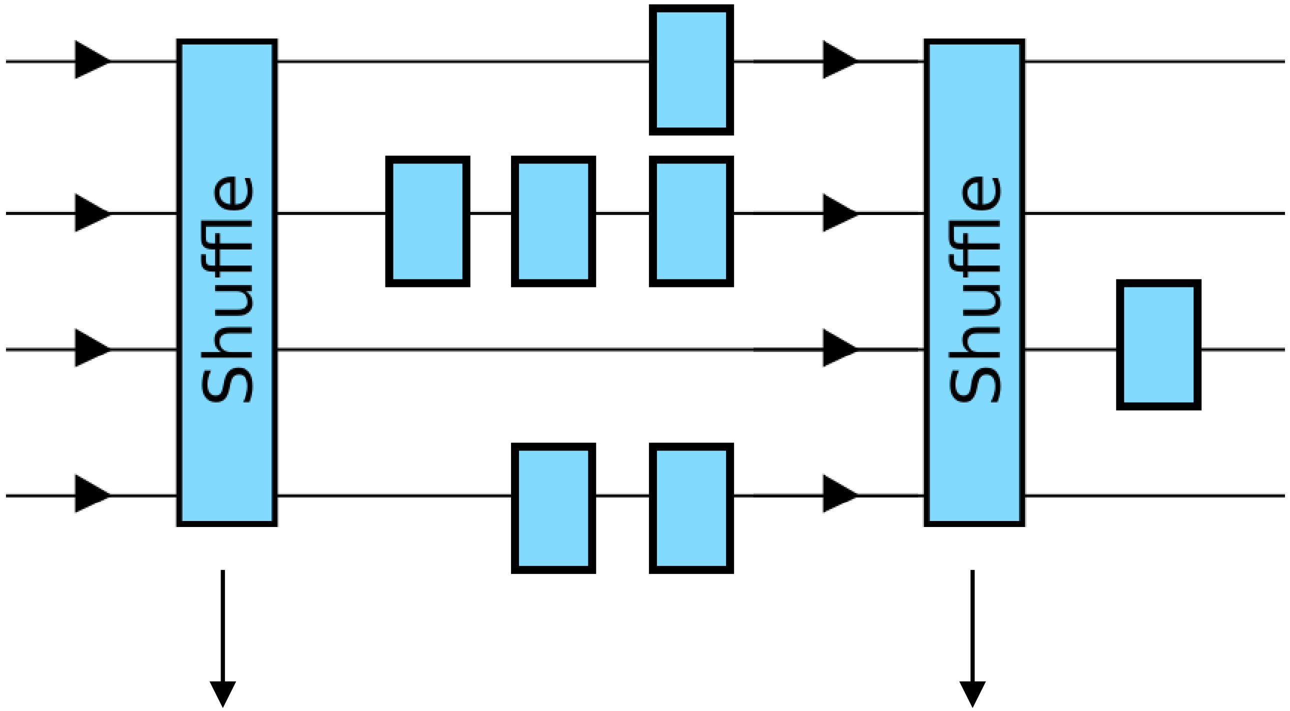
Four-element ORAM



Larger Sizes



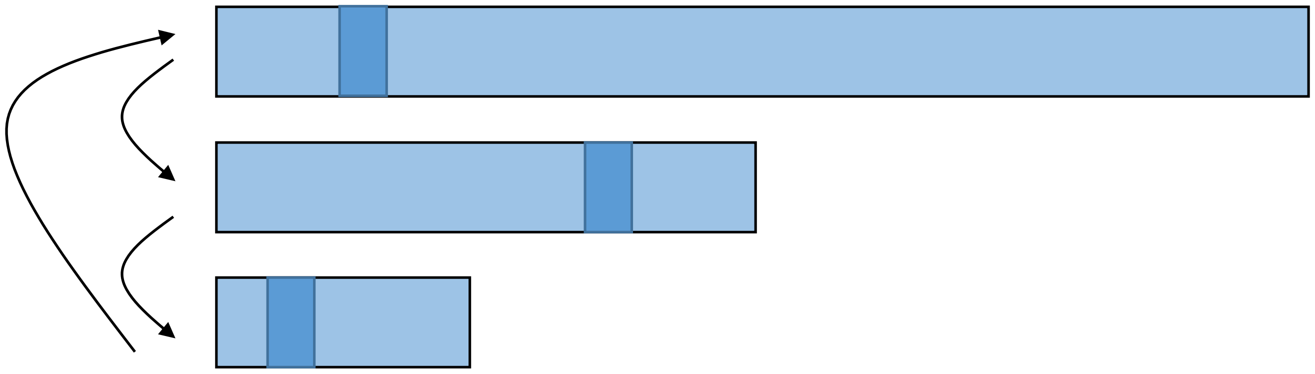
Position map



0	1	2	3
3	0	2	1

0	1	2	3
1	3	0	2

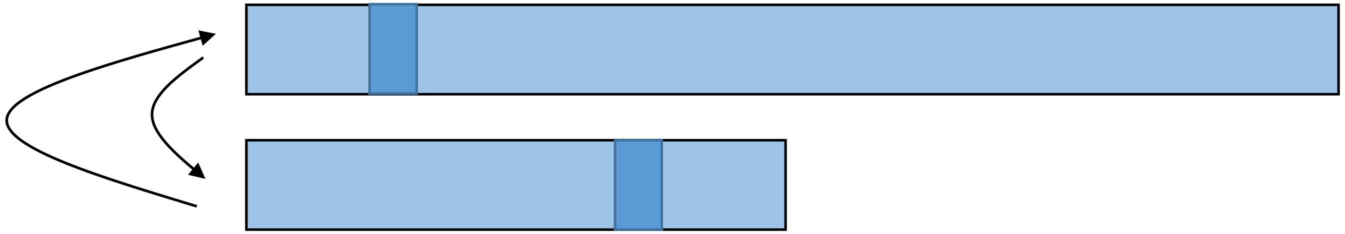
Keeping position map updated



4	1	5	2	0	7	3	6

Position map

Keeping position map updated



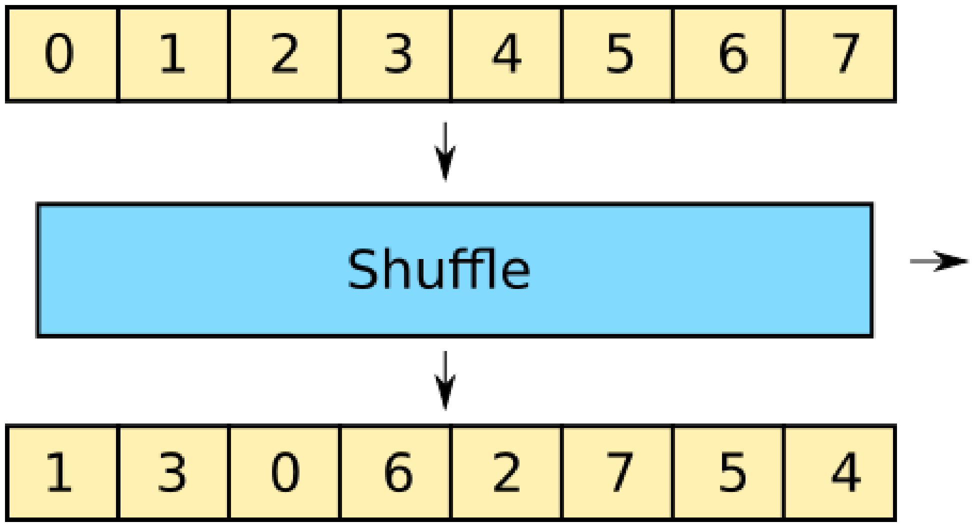
4	1	5	2	0	7	3	6

Position map

Rinse and repeat

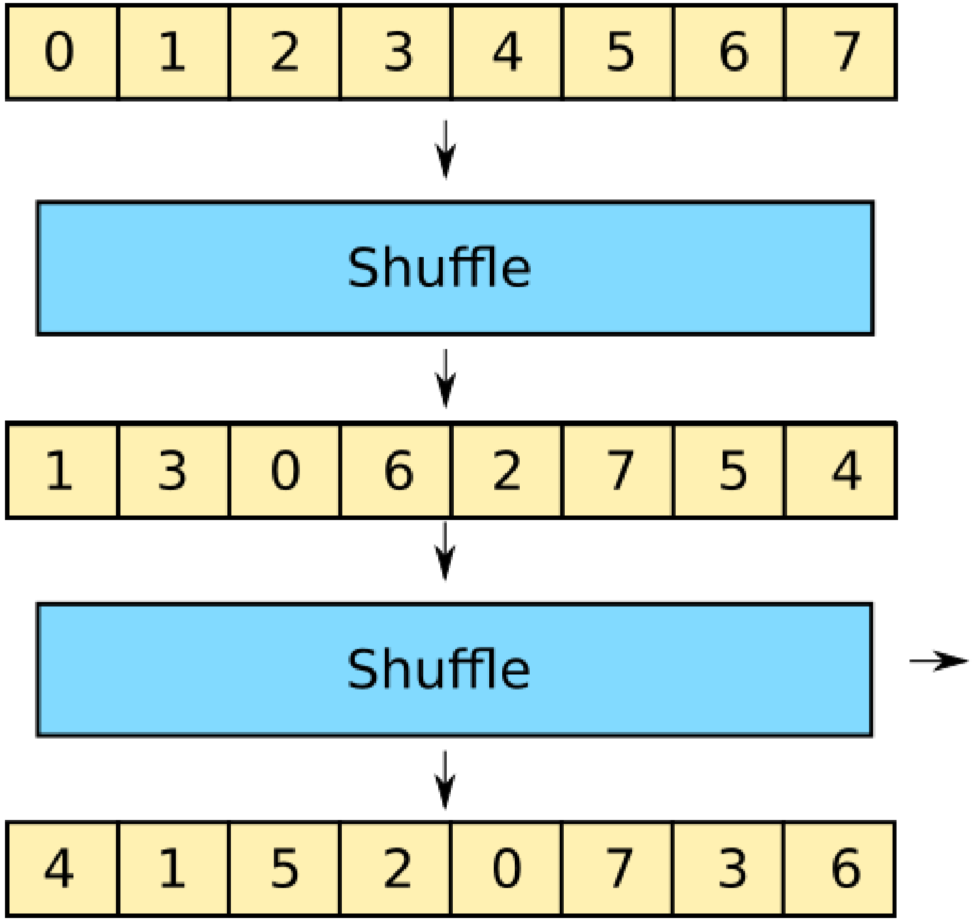
1. Shuffle elements
2. Recreate position map
3. Service $T = \sqrt{n \log n}$ accesses

Creating position map



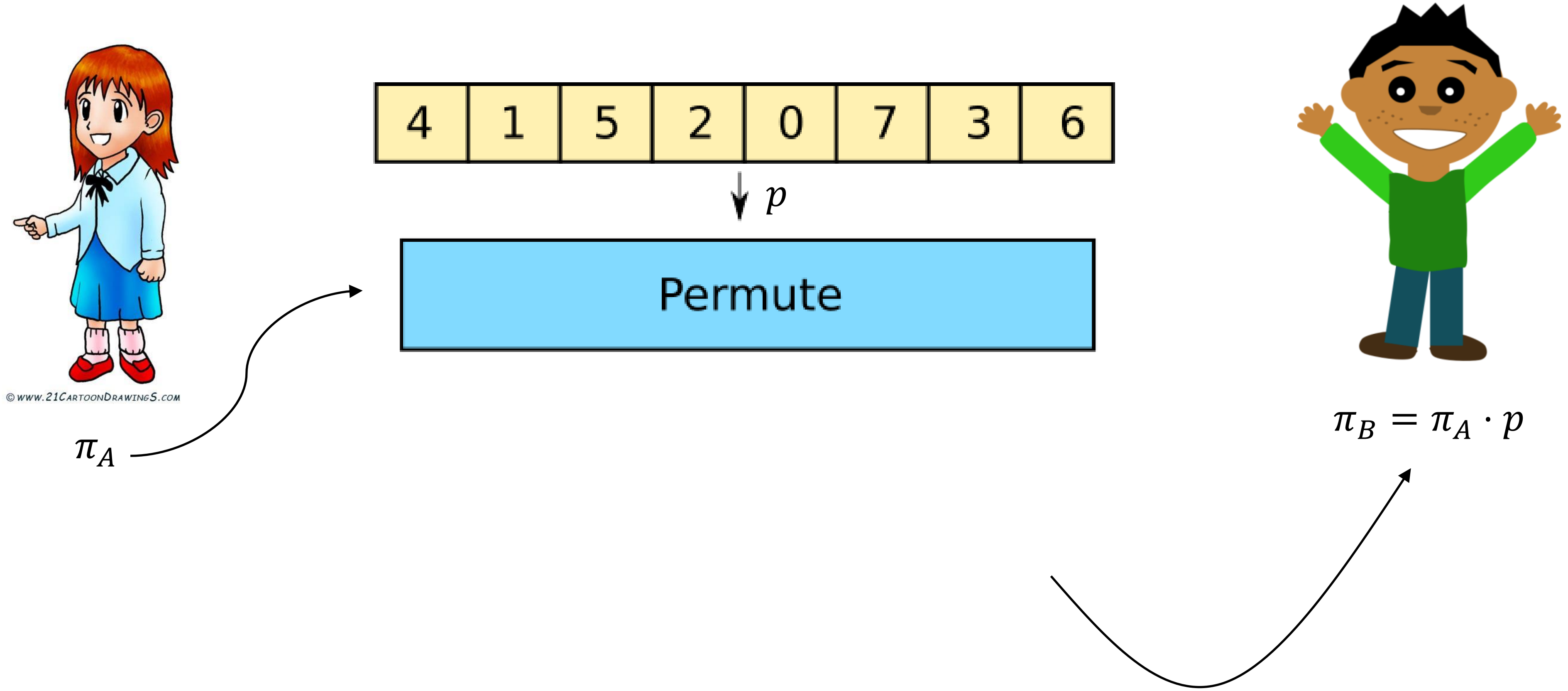
x	$\pi(x)$
0	2
1	0
2	4
3	1
4	7
5	6
6	3
7	5

Creating position map



x	$\pi(x)$
0	4
1	1
2	3
3	6
4	0
5	2
6	7
7	5

Inverse permutation



Inverse permutation

π_A

π_A

Permute

π_B

$\pi_B = \pi_A \cdot p$

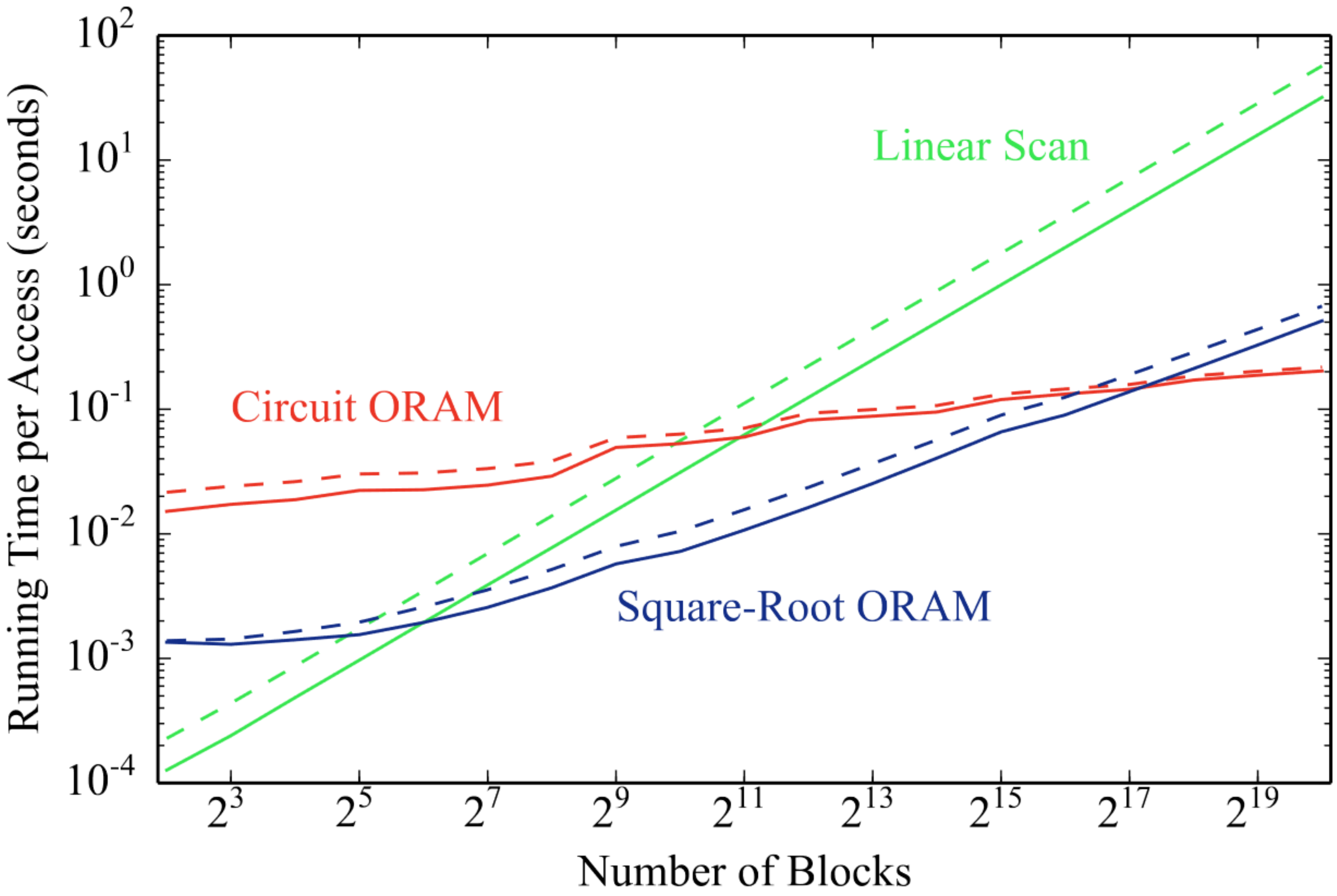
Bob computes
$\pi_B^{-1} = p^{-1} \cdot \pi_A^{-1}$

$$\begin{aligned} &\pi_B^{-1} \cdot \pi_A \\ &= p^{-1} \boxed{\pi_A^{-1} \cdot \pi_A} \\ &= p^{-1} \end{aligned}$$

Rinse and repeat

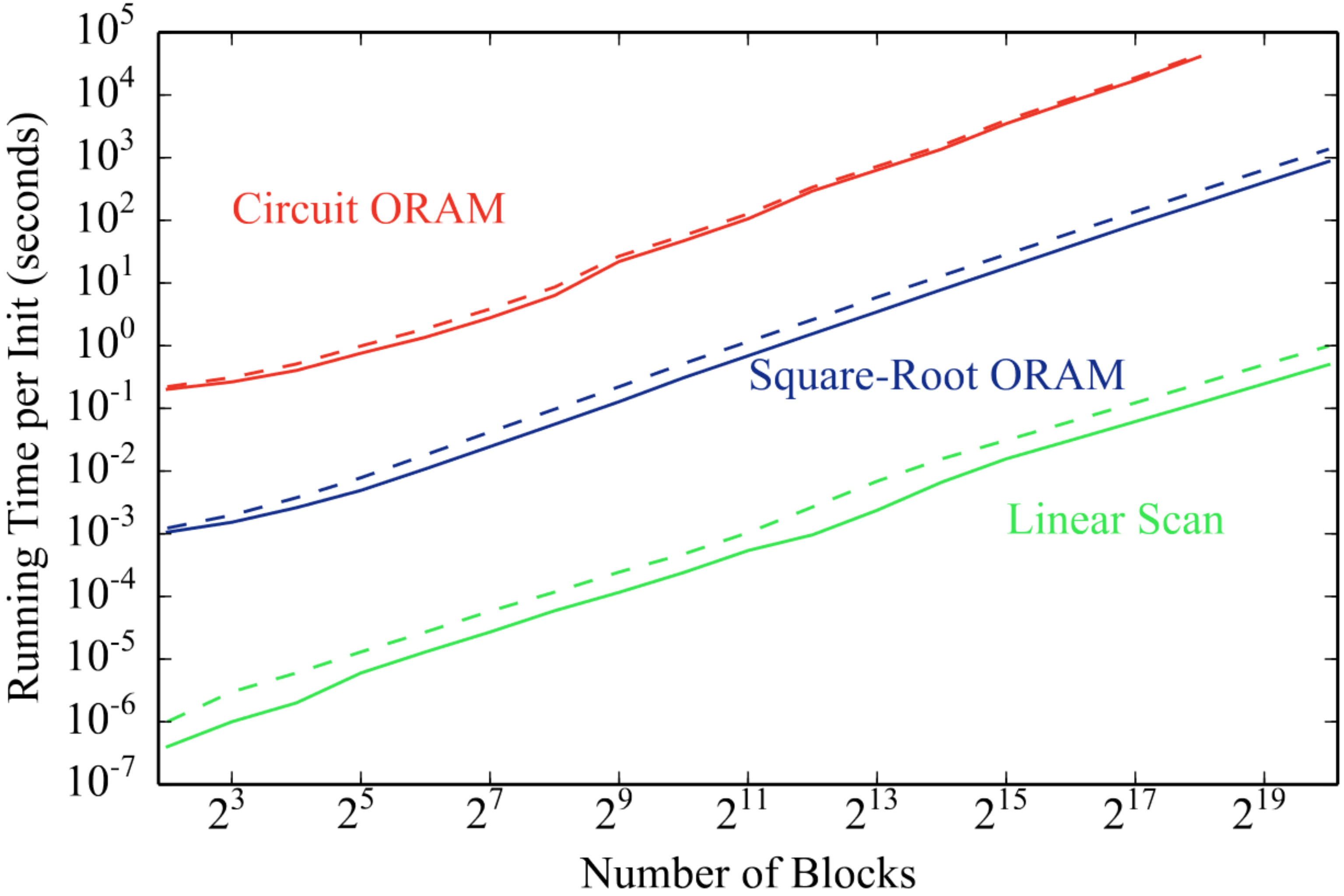
1. Shuffle elements
2. Recreate position map at $\Theta(n \log n)$
3. Service $T = \sqrt{n \log n}$ accesses

Access time



Running Time per Access (seconds)
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
Linear Scan
Circuit ORAM
Square-Root ORAM
2^3
2^5
2^7
2^9
2^{11}
2^{13}
2^{15}
2^{17}
2^{19}
Number of Blocks

Initialization cost



Benchmarks

Task	Parameters	Linear scan	Circuit ORAM	Square-root ORAM
Binary search	2^{10} searches 2^{15} elements	1020 s	5041 s	825 s
Breadth-first search	2^{10} vertices 2^{13} edges	4570 s	3750 s	680 s
Stable matching	2^9 pairs	-	189000 s	119000 s
scrypt hashing	$N = 2^{14}$	≈ 7 days	2850 s	1920 s

Conclusion

We revisited a well-known scheme and used it to

- Lower initialization cost
- Improve breakeven point

Shows that asymptotic costs are not the final word, concrete costs require more consideration.

Download

oblivc.org/sqoram

Contact for help:
Samee Zahur <samee@virginia.edu>