OptORAMa: Optimal Oblivious RAM

Gilad Asharov
Bar-Ilan University

Ilan Komargodski Wei-Kai Lin Kartik Nayak
Enoch Peserico Elaine Shi

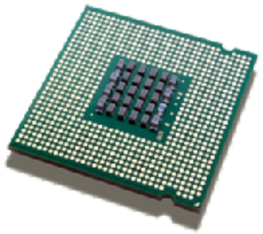
NTTResearch Cornell University Duke UNIVERSITY Università degli Studi di Padova

Roadmap

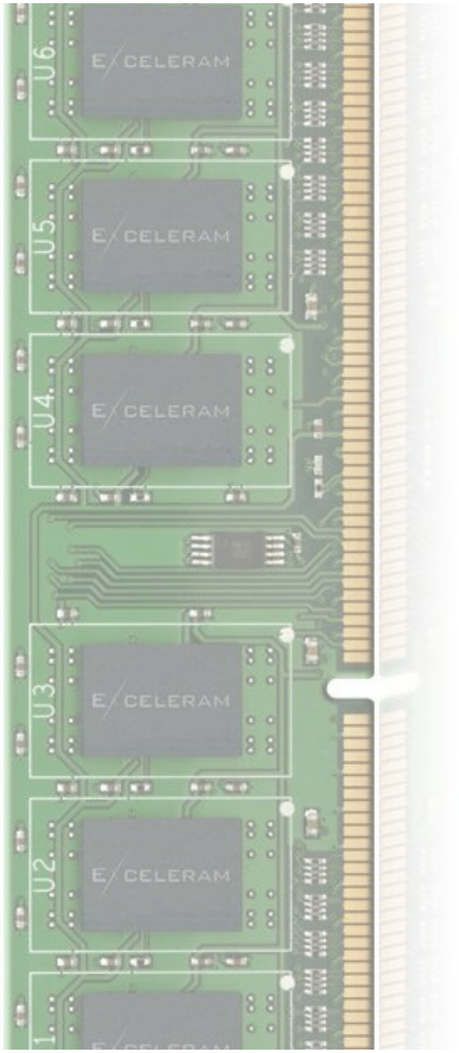
- Introduction
  - Problem definition and our result
- A short tutorial
  - From Square Root ORAM to OptORAMa
- Our techniques

Access Pattern Leakage

(or, why encrypting the data is insufficient?)

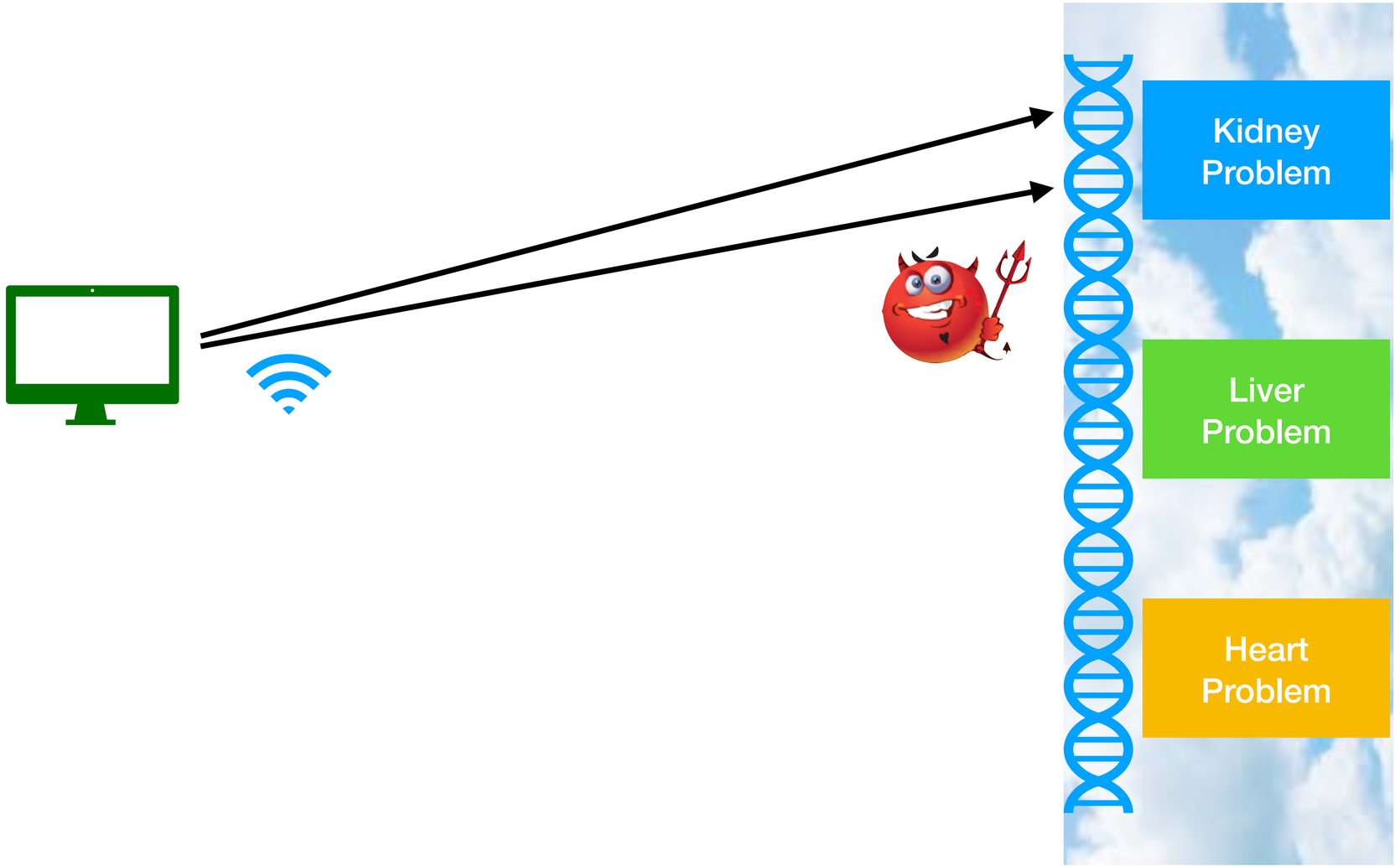


secure processor



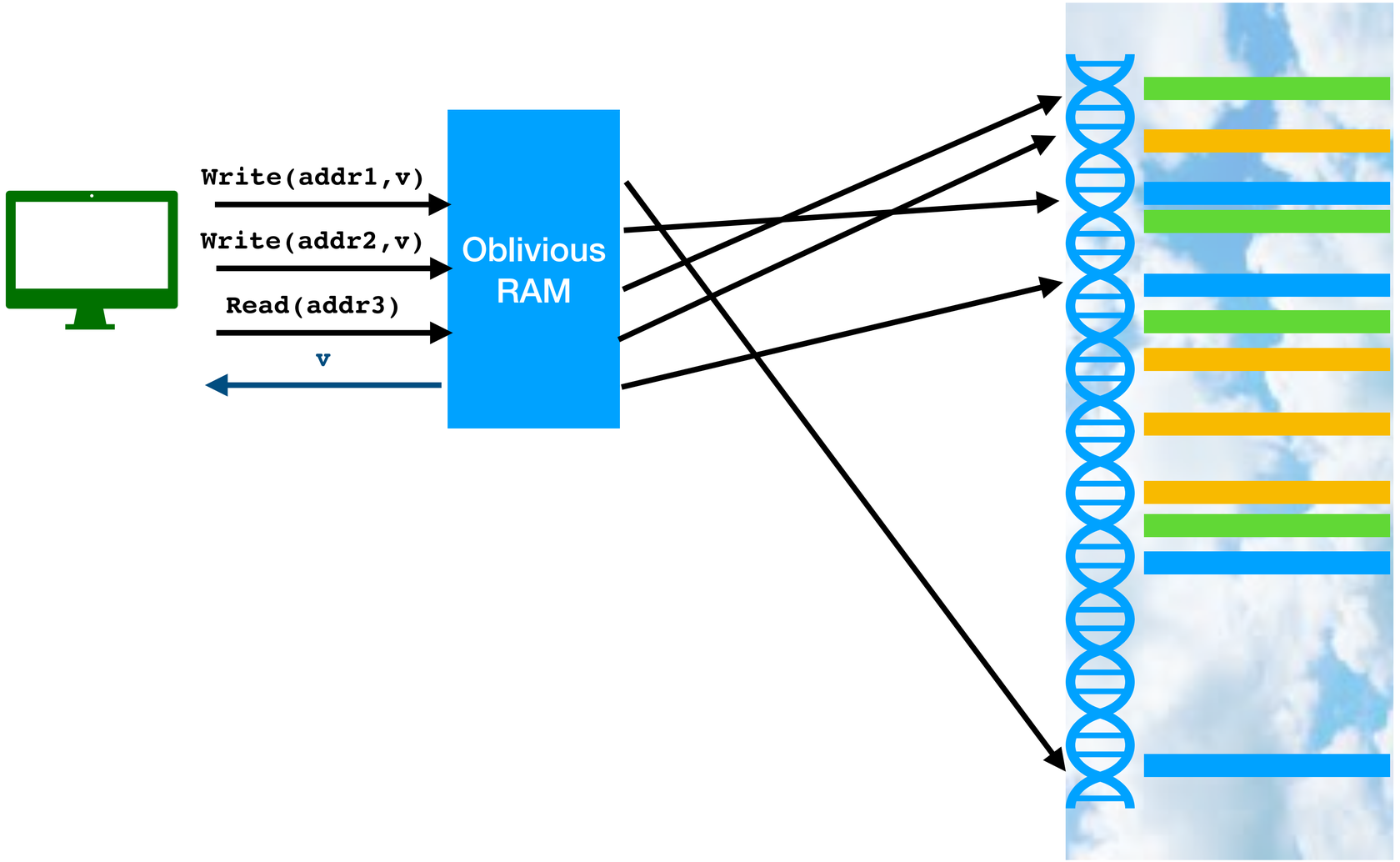
Access Pattern Leakage

(or, why encrypting the data is insufficient?)



Oblivious RAM

(or - How to Hide the Access Pattern?)



Oblivious RAM

- Introduced by Goldreich and Ostrovsky [STOC'87,STOC'90,JACM'96]

- **Informal definition:**
  - The *access pattern* can be simulated from just the number of data accesses
    - The access pattern is data independent

- **Lower bound:** memory N
  - **Ω(log N)** amortized overhead even with crypto [GoldreichOstrovsky96,LarsenNielsen18]

Overhead of Oblivious RAM

Model: Passive server, word size $O(\log N)$, client memory size $O(1)$

Lower Bound: [Goldreich'87,LarsenNielsen'18]	$\Omega(\log N)$
[Goldreich'87]	$O(\sqrt{N}\log N)$
[Ostr'90/GO'96]	$O(\log^3 N)$
[GoodrichMitzenmacher'11, PathORAM'12]	$O(\log^2 N)$
[KushilevitzLuOstrovsky'12]	$O(\log^2 N/\log\log N)$
PanoRAMa: [Patel,Persiano,Raykova,Yeo'18]	$O(\log N \log\log N)$
Our Result:	$O(\log N)$

Our Main Result

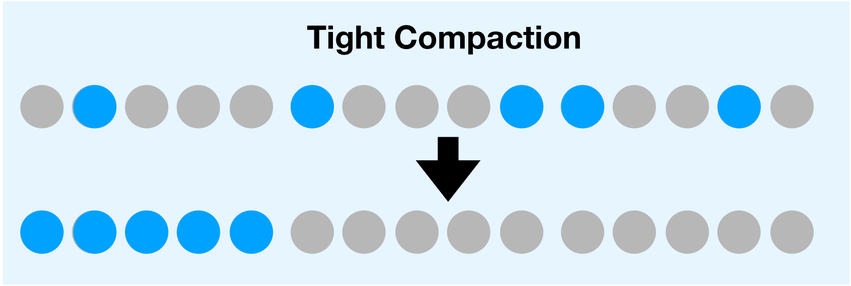
There exists an ORAM with O(log N) amortized overhead

🎉 **Asymptotically Optimal!** 🎊

- Computational Security (OWF)
  - Matches [LN'18]
- PRF -> Random Oracle
  - Statistical security
  - Matches [GO'96]

- **Word size:** log N
- Client's memory size O(1) words
- Passive server
- Balls and bins model
- Large hidden constant
- Based on hierarchical ORAM

Our Result:
Oblivious Tight Compaction



- Oblivious?
  - Best prior deterministic: O(n log n) [AKS'83]
  - Open question from [LeightonMaSuel'95]: Reveals the number of marked elements, O(n log logn) randomized
  - Best prior randomized: O(n log log n) [MZ'14,LST'18] (negligible error prob.)
  - **Lower bound**: Stable, balls-and-bins, Ω(nlogn)

- **Our result:**
  Deterministic, oblivious tight compaction in O(n) (in the balls and bins model)

- [Asharov,Komargodski,Lin,Peserico,Shi] - ITC 2020 : $O(\log N)$ depth
- DittmerOstrovsky: Oblivious tight compaction in $O(n)$ time with smaller constant

A Short Tutorial

Square Root

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

Goldreich 87

Hierarchical Solution

$O(\log^3 N), \ldots, O(\frac{\log^2 N}{\log\log N})$

[Ostrovsky'90],…,[KLO12]

PanORAMa

$O(\log N \log\log N)$

Patel,Persiano,Raykova,Yeo'18

OptORAMa

$O(\log N)$

Our Work

Square Root

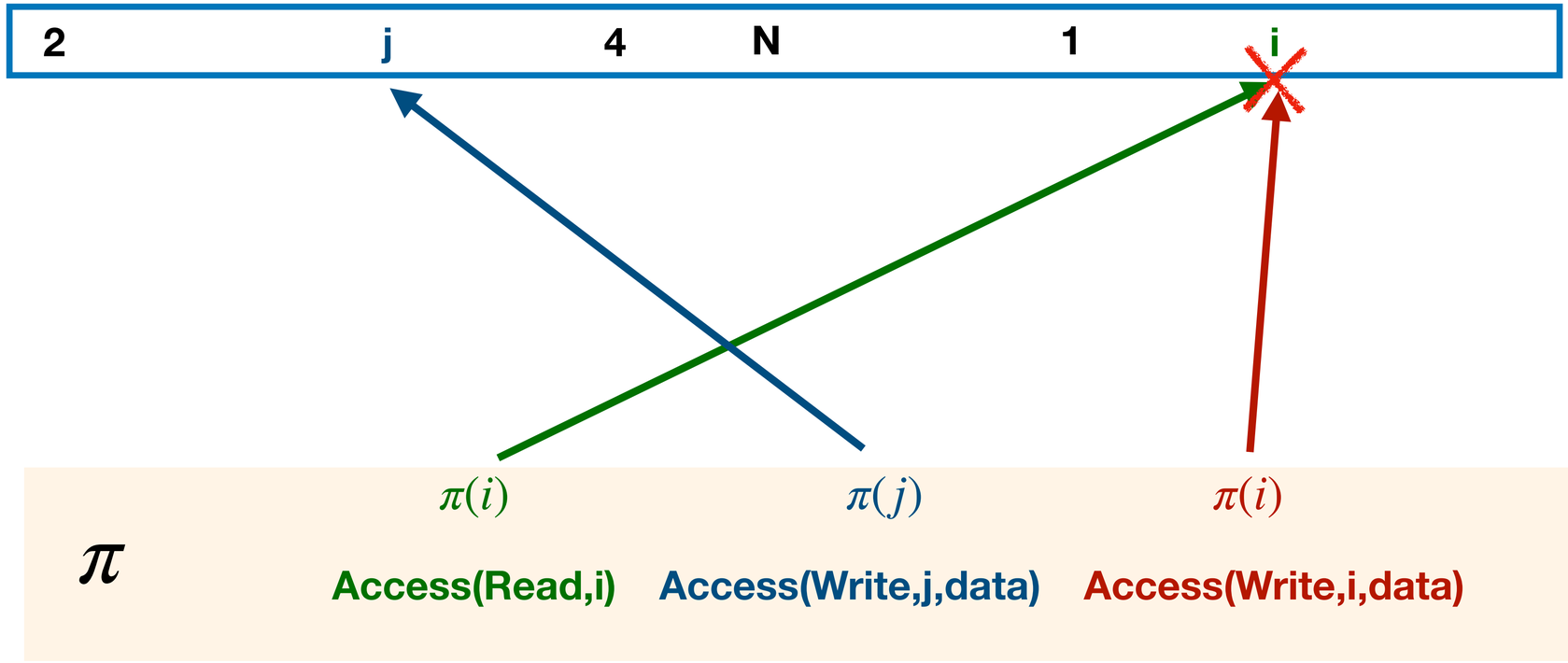
$O(\sqrt{N}\log N)$

Goldreich 87

Warmup: Square Root ORAM

Randomly permute the memory using a random permutation π

N



Square Root

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

Goldreich 87

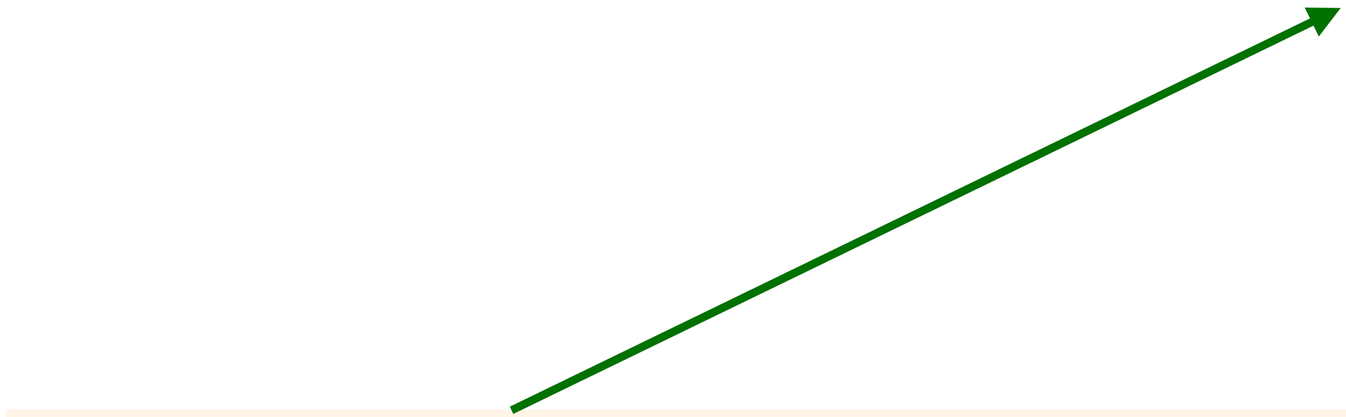
Warmup: Square Root ORAM

Shelter

$\sqrt{N}$

Add $\sqrt{N}$ dummy elements $\perp$

$N+\sqrt{N}$



$\pi(i)$

π

Access(Read,i)

Square Root

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

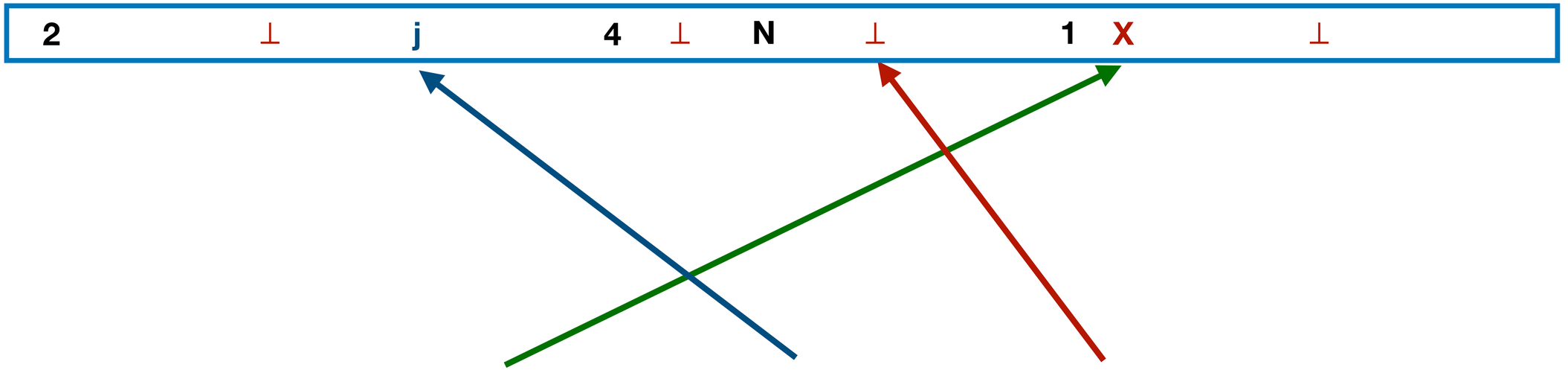
Goldreich 87

Warmup: Square Root ORAM

Shelter $\sqrt{N}$

Add $\sqrt{N}$ dummy elements $\perp$

$N+\sqrt{N}$



π

$\pi(i)$	$\pi(j)$	$\pi(n+3)$
Access(Read,i)	Access(Write,j,data)	Access(Write,i,data)

Square Root

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

Goldreich 87

Warmup: Square Root ORAM

Shelter

$\sqrt{N}$

Add $\sqrt{N}$ dummy elements $\perp$

$N+\sqrt{N}$

After $\sqrt{N}$ accesses the shelter is full -> **Rebuild**

Rebuild uses **oblivious sort**

Hierarchical Solution

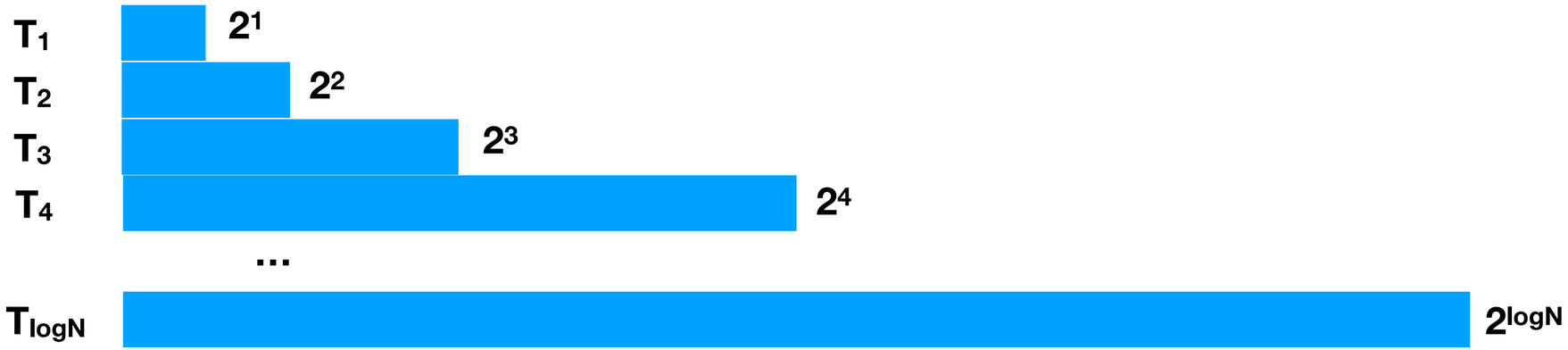
$O(\log^3 N), \ldots, O(\frac{\log^2 N}{\log \log N})$

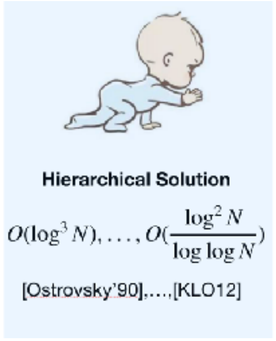
[Ostrovsky'90],...,[KLO12]

Hierarchical ORAM

logN - levels of doubling sizes
Every access — **Lookup** in each table
Every 2^i accesses — **Build** Table T_i

$$\approx \log N \cdot (T_{\text{Build}}(N)/N + T_{\text{Lookup}})$$

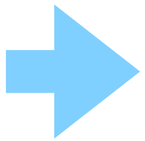


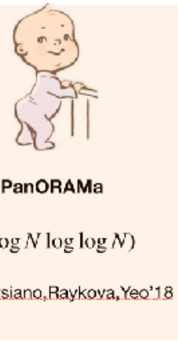


Hierarchical Solution

$O(\log^3 N), \ldots, O(\frac{\log^2 N}{\log \log N})$

[Ostrovsky'90],....,[KLO12]





PanORAMa

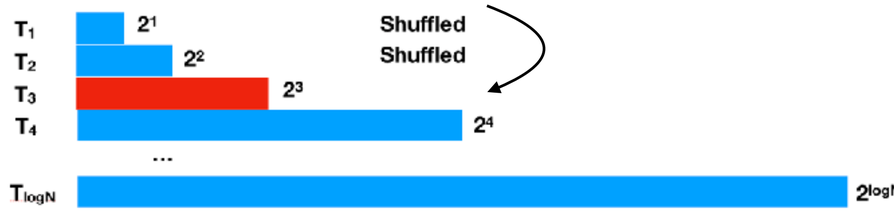
$O(\log N \log \log N)$

Patel,Persiano,Raykova,Yeo'18

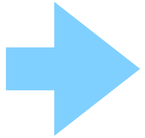
$$\approx \log N \cdot (T_{\text{Build}}(N)/N + T_{\text{Lookup}})$$

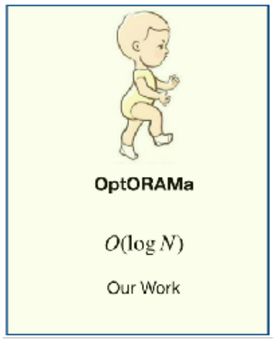
- Previous works: **Rebuild** uses oblivious sort $O(N \log N)$
  - $\approx O(\log N \cdot (N \log N/N + T_{\text{Lookup}})) \Longrightarrow O(\log^2 N)$

- **PanORAMA**: **Rebuild** for a ***randomly shuffled*** input
  - Implementing **Rebuild** in $O(N \log \log N)$



- **But…**
  - Each layer is shuffled, but the concatenation is not shuffled
    - PanORAMa showed how to "intersperse" arrays in $O(N \log \log N)$





	PanORAMa	OptORAMa
Rebuild	O(N log log N)	O(N)
Lookup	O(log log N)*	O(1)*
Intersperse	O(N log log N)	O(N)

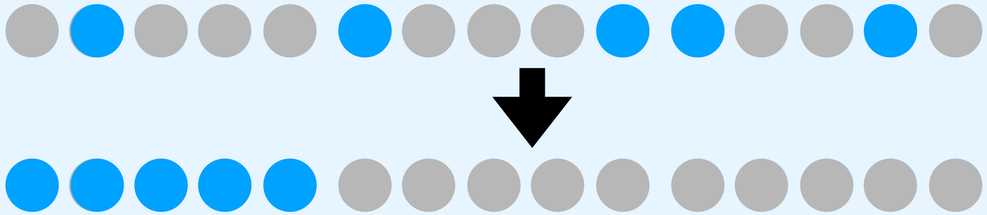
~~$\approx \log N \cdot (T_{\text{Build}}(N)/N + T_{\text{Lookup}})$~~

$\approx \log N \cdot (T_{\text{Rebuild}}(N)/N + T_{\text{Lookup}} + T_{\text{intersperse}}(N)/N)$

***effectively (ignoring stashes)**

Our Techniques

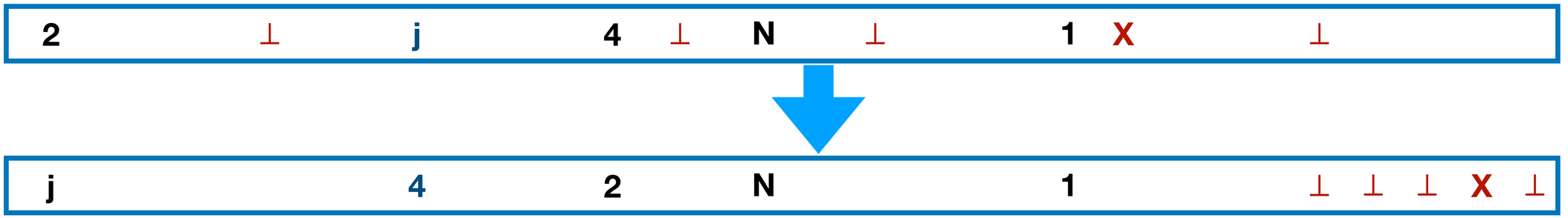
Tight Compaction



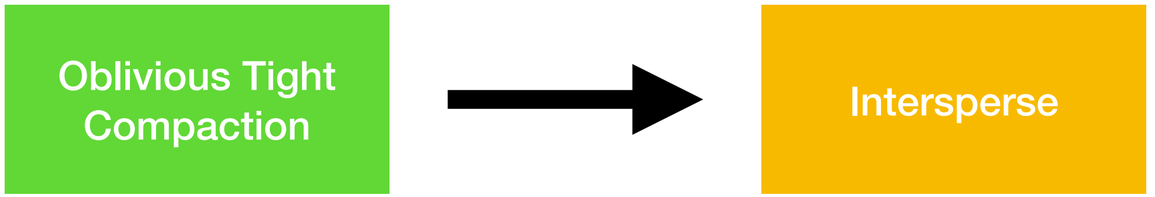
Packing

Tight Compaction: Where Is It Being Used?

1) Getting rid of dummy elements



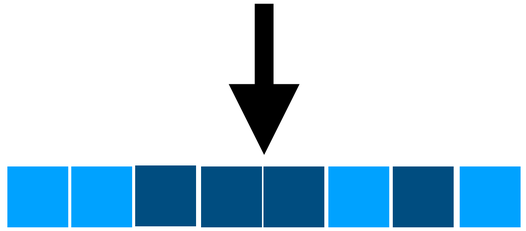
2) Interspersing arrays:



Intersperse [PanORAMa]

Generate random Aux with n_0 zeros, n_1 ones

Oblivious route



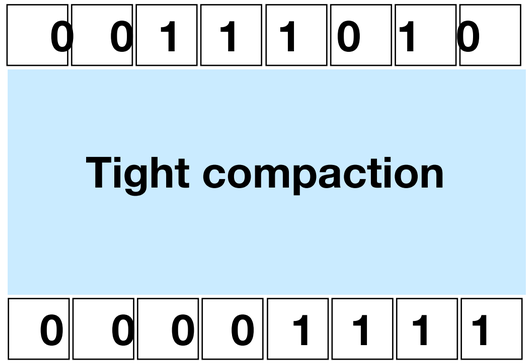
$$\binom{n}{n_0} \cdot n_0! \cdot n_1! = n!$$

$$n = n_0 + n_1$$

Challenge: Move the elements **Obliviously**
PanORAMa: Implemented in O(n log log n)

Intersperse From Oblivious Tight Compaction

Generate random Aux

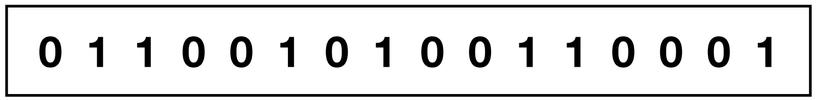


Remember all “move balls”

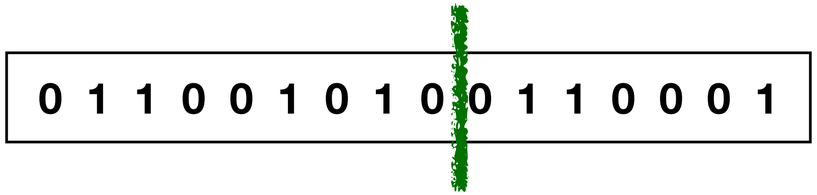
Perform same “swaps”

Oblivious Tight Compaction

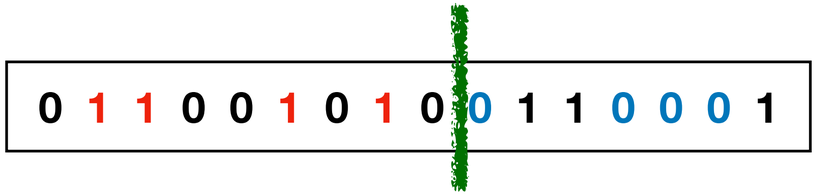
- **Input**: An array of size n where each element is marked 0 or 1
- **Output**: all 0-elements appear before 1-elements



(1) Count the number of balls marked 0

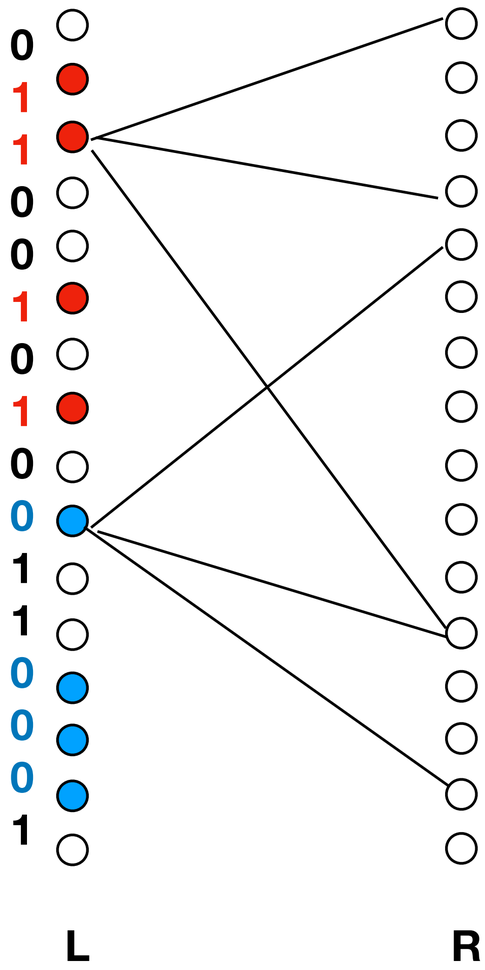


(2) Mark the elements that are “misplaced”



Observation: number of reds always equals number of blues
We just have to swap them!

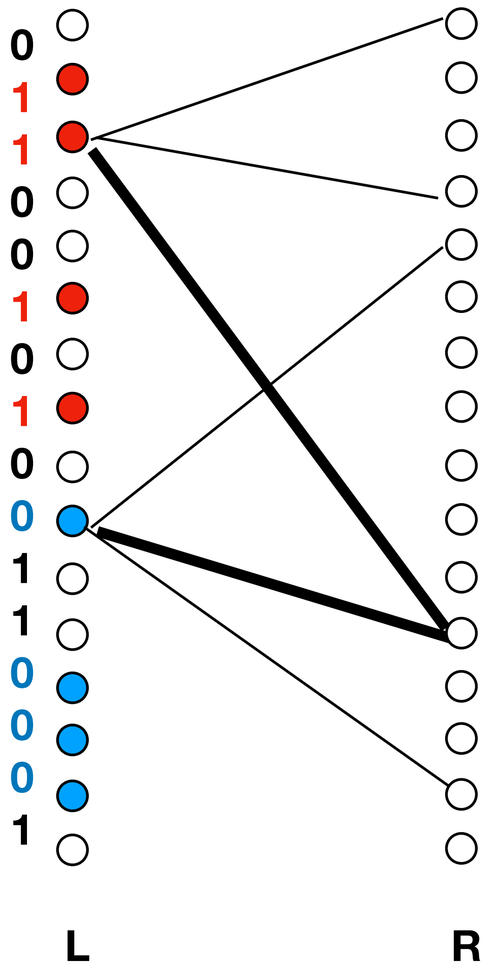
Loose Swap



Bipartite Expander Graph

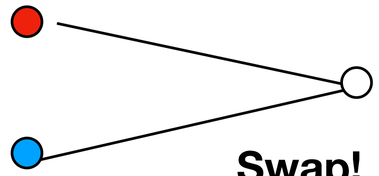
- $O(1)$ degree
- Constant spectral expansion

Loose Swap



- 1 that wants to switch with 0
- 0 that wants to switch with 1

For every pair of

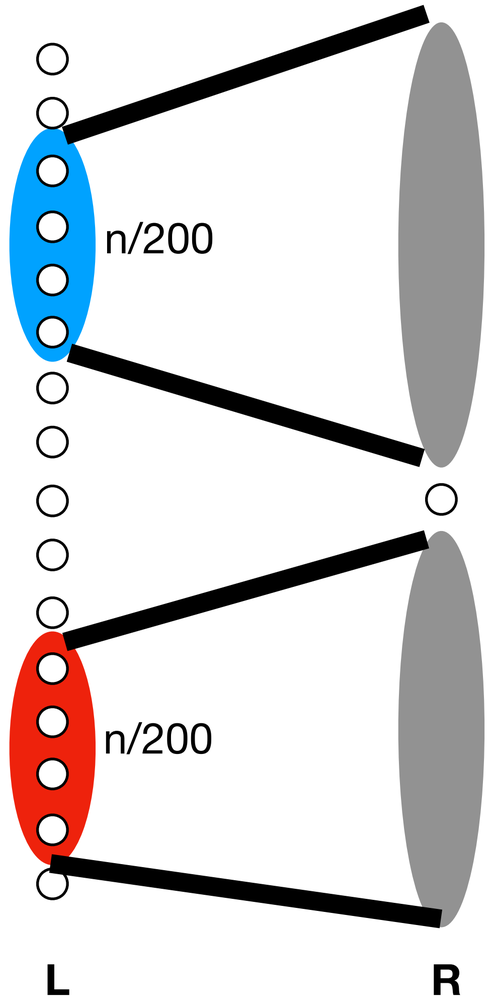


$O(nd^2)$ **work**

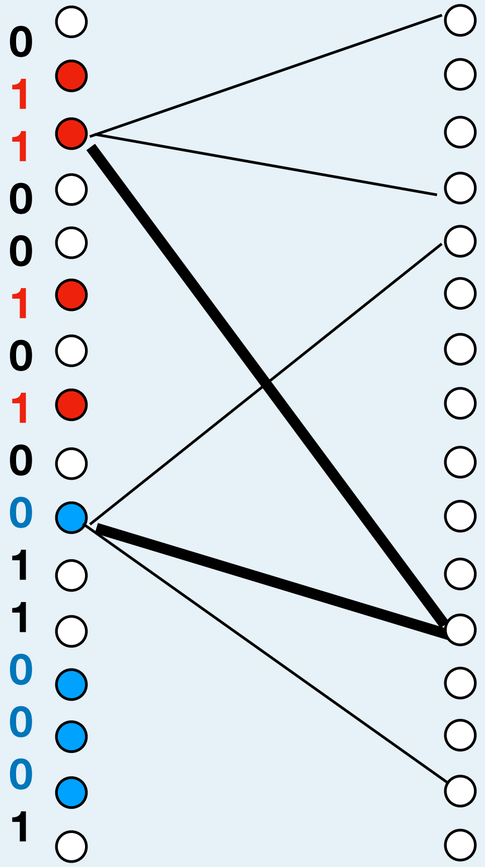
Claim

At the end of this procedure, there are no more than n/100 remaining swaps

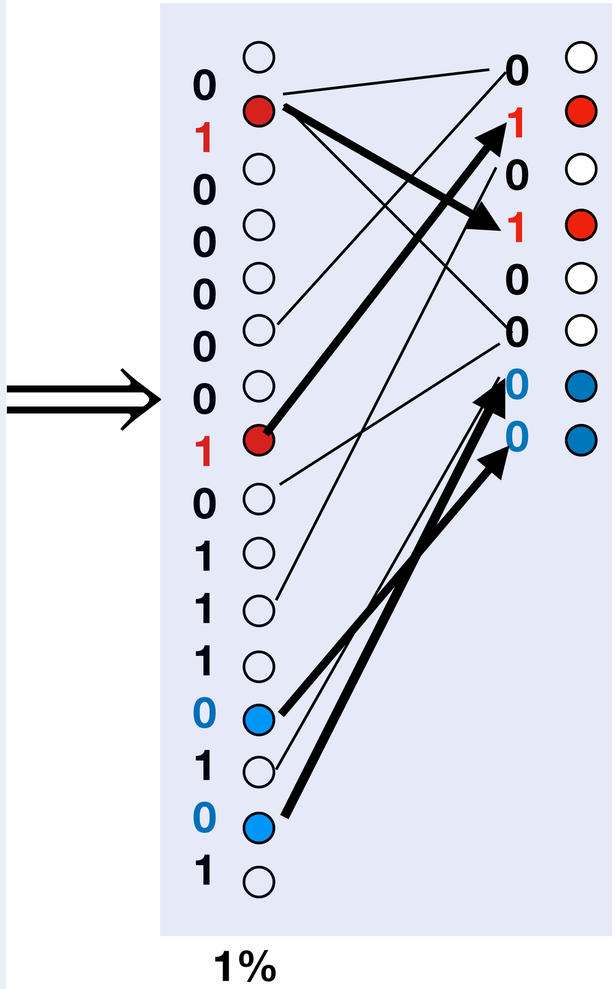
- Consider the sets of “survivors”
  - Each of size > n/200
    (Recall #reds=#blues)
  - Their sets of neighbors must be disjoint
- **Expansion property**:
  for any set of size > n/200,
  number of neighbors > n/2

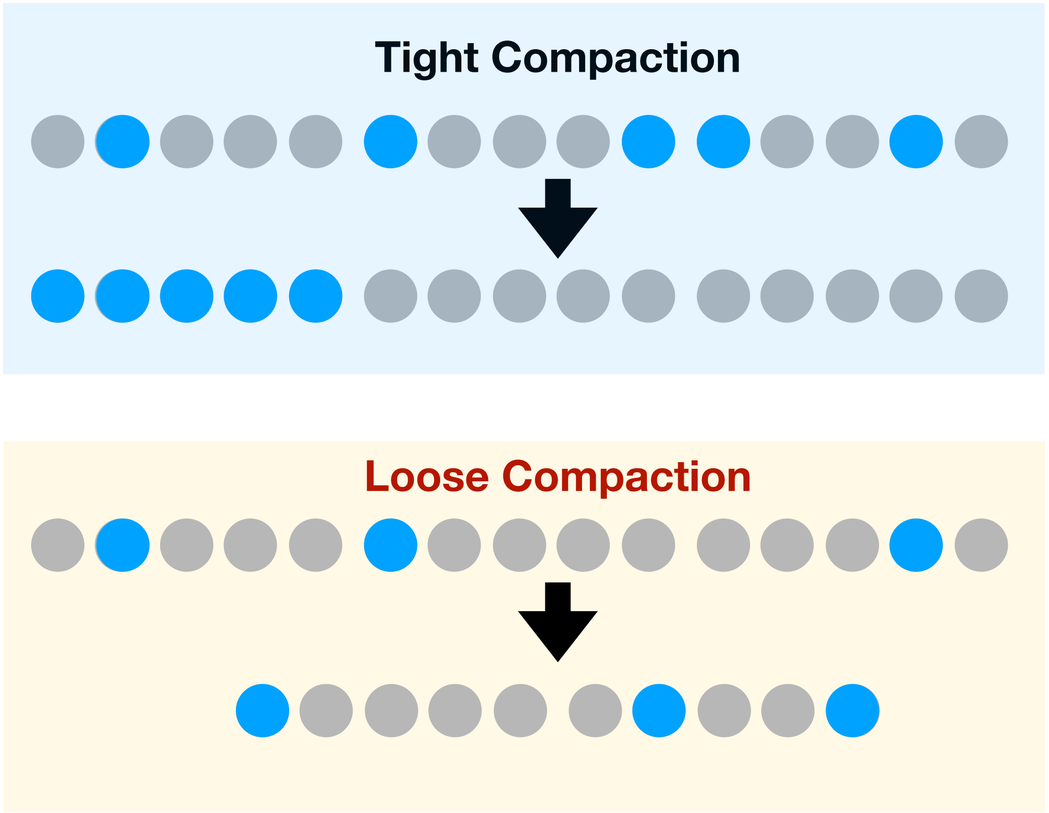


Loose Swap

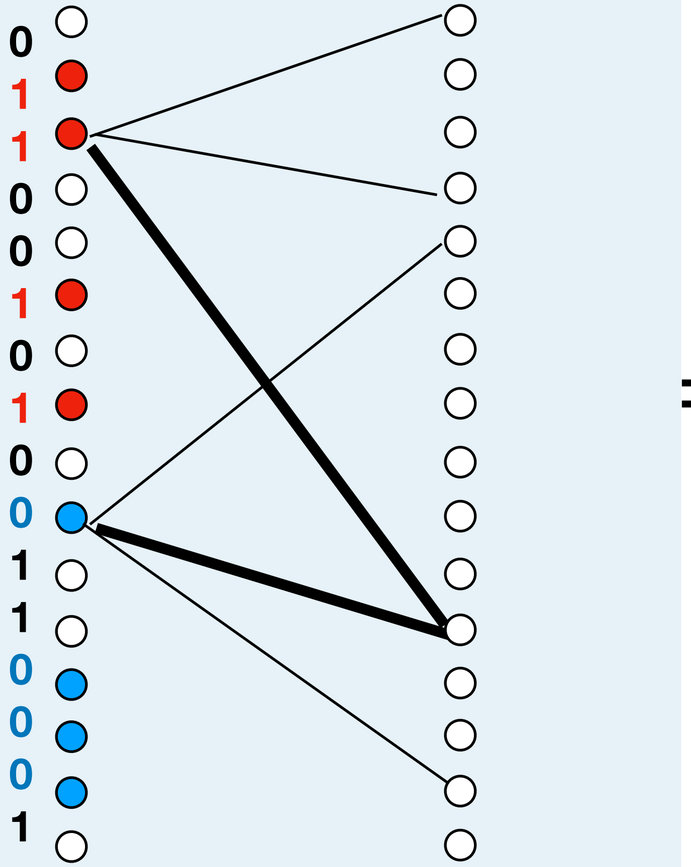


Loose Compactor

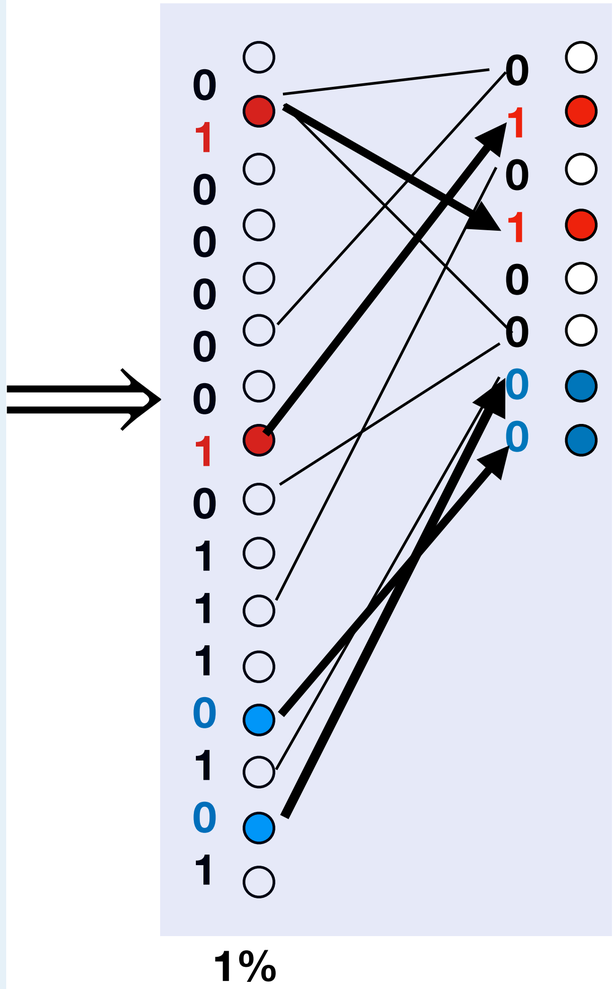




Loose Swap

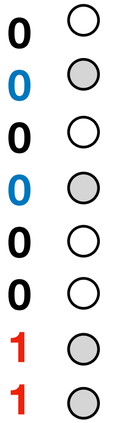


Loose Compactor

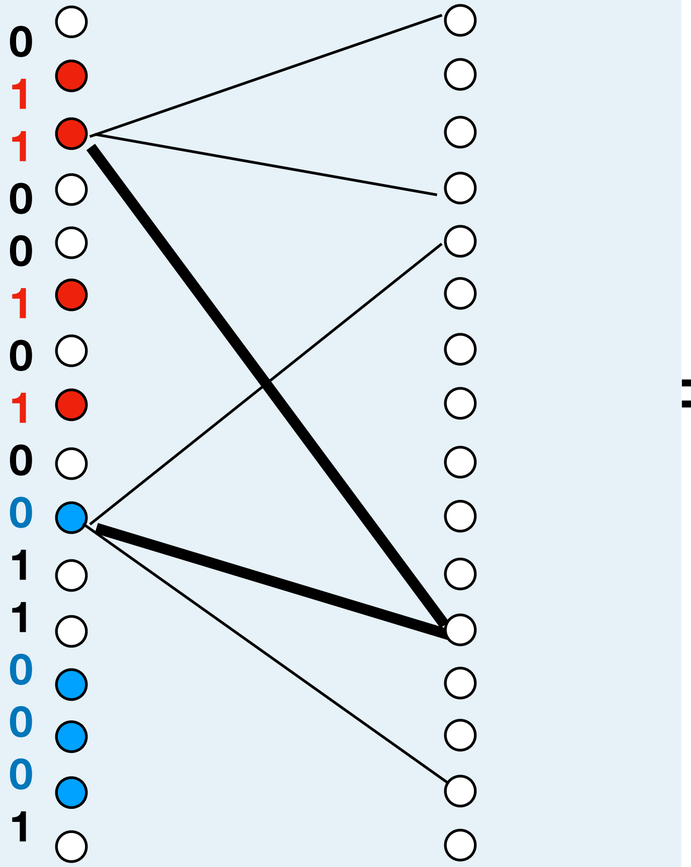


Recurse

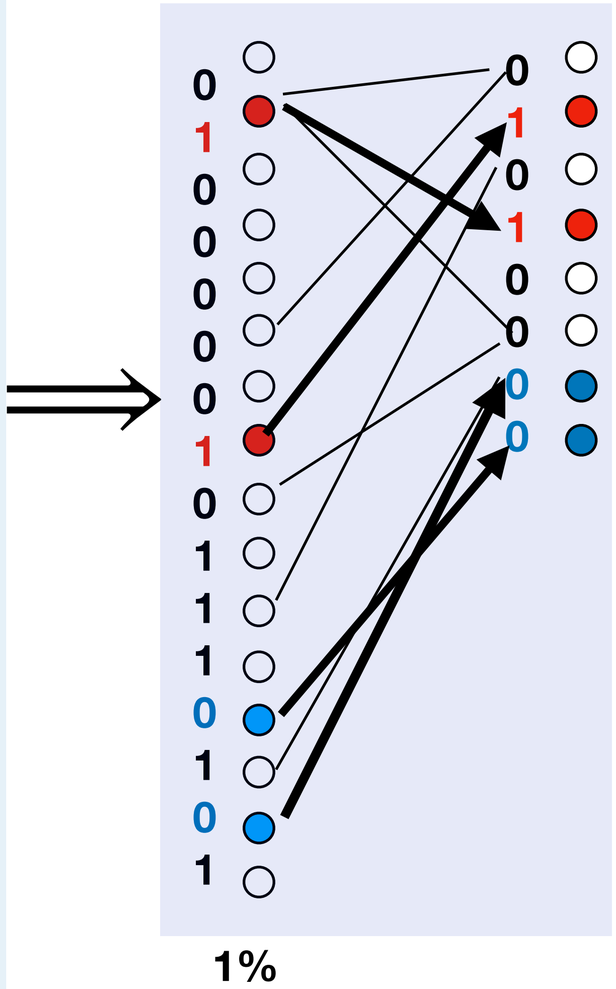
...



Loose Swap

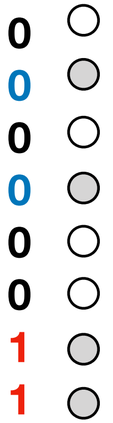


Loose Compactor

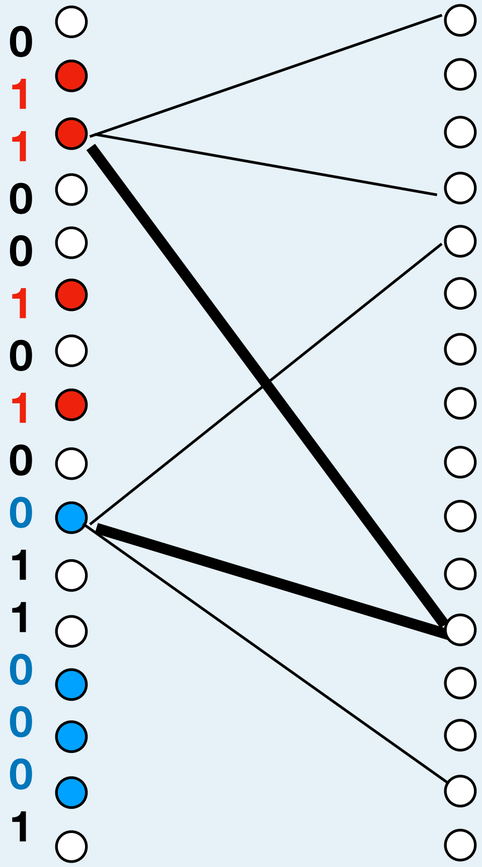


Recurse

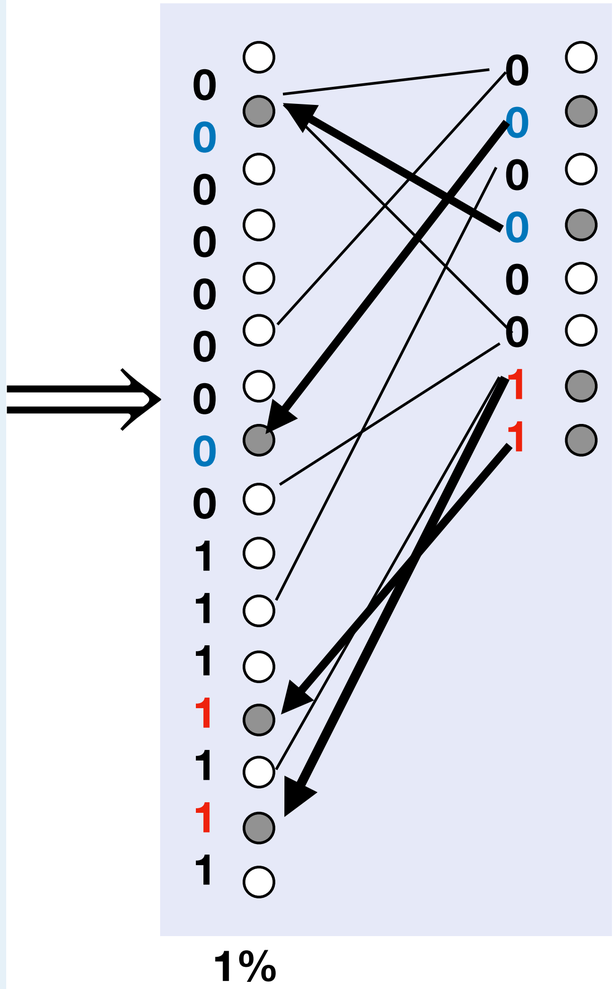
...



Loose Swap

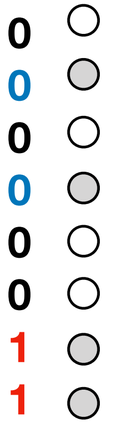


Reverse Route

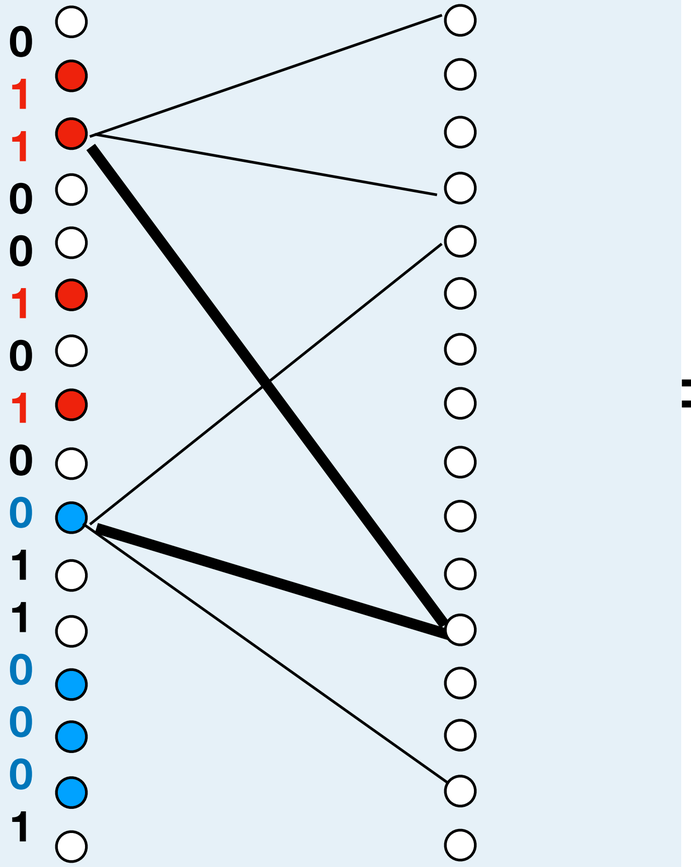


Recurse

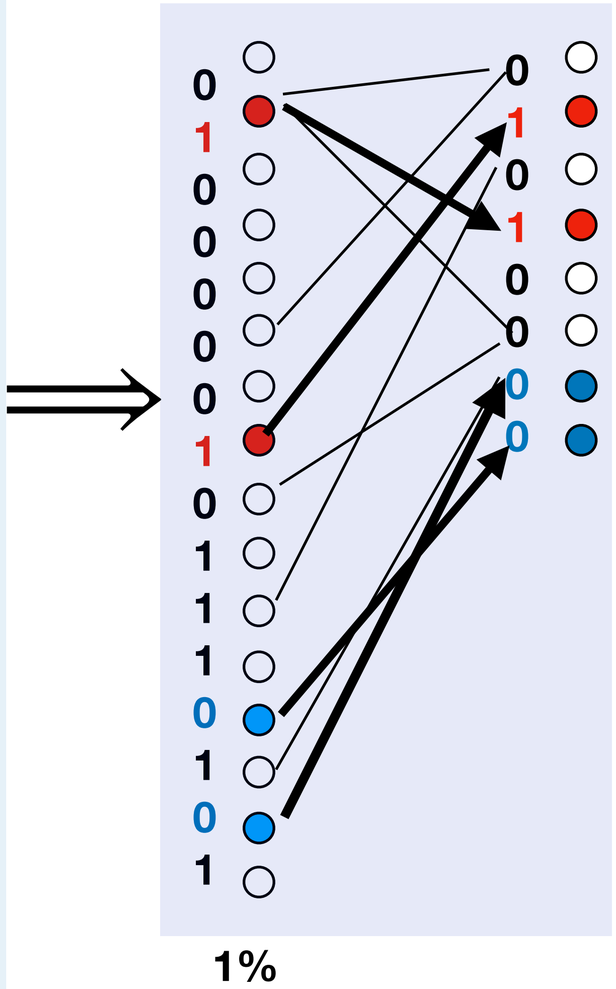
...



Loose Swap

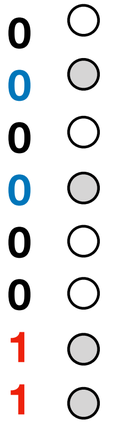


Loose Compactor

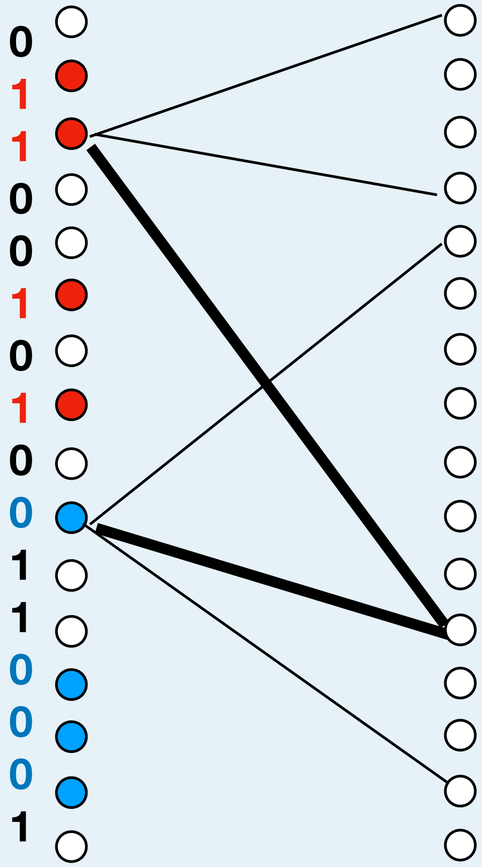


Recurse

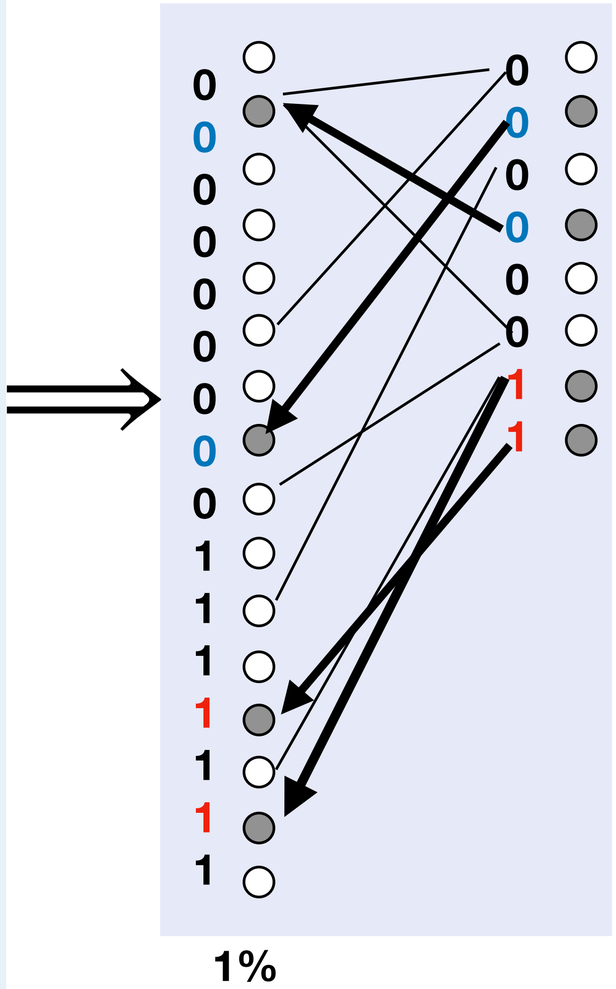
...



Loose Swap

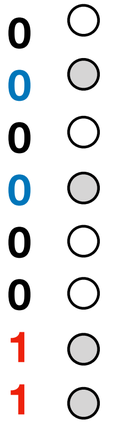


Reverse Route



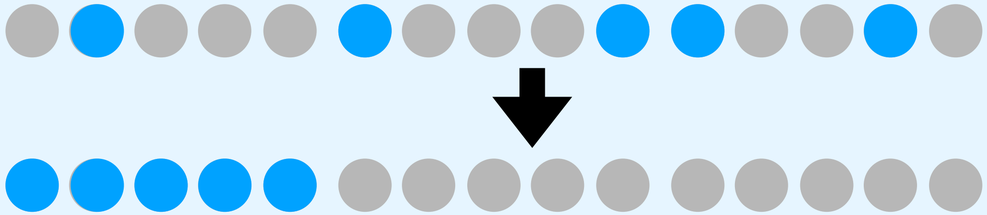
Recurse

...



Our Techniques

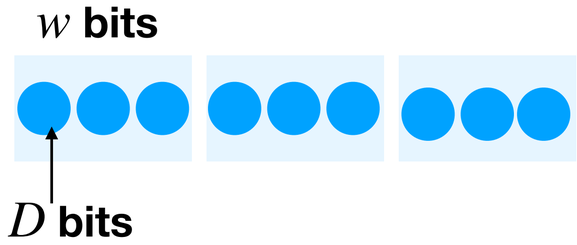
Tight Compaction



Packing

Packing - The Idea

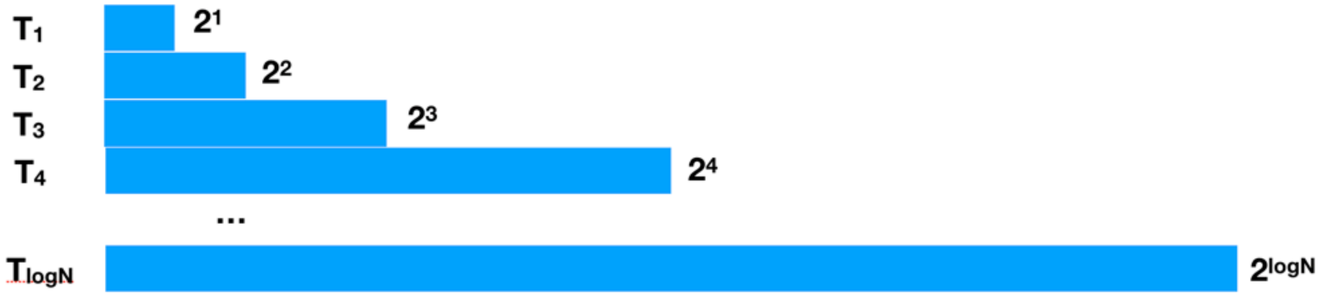
- Given n balls each of size D bits, word size w
- Classical oblivious sort costs $O(\lceil D/w \rceil \cdot n \cdot \log n)$
- What if $D \ll w$?
- **Packing**: put w/D balls in one memory word!



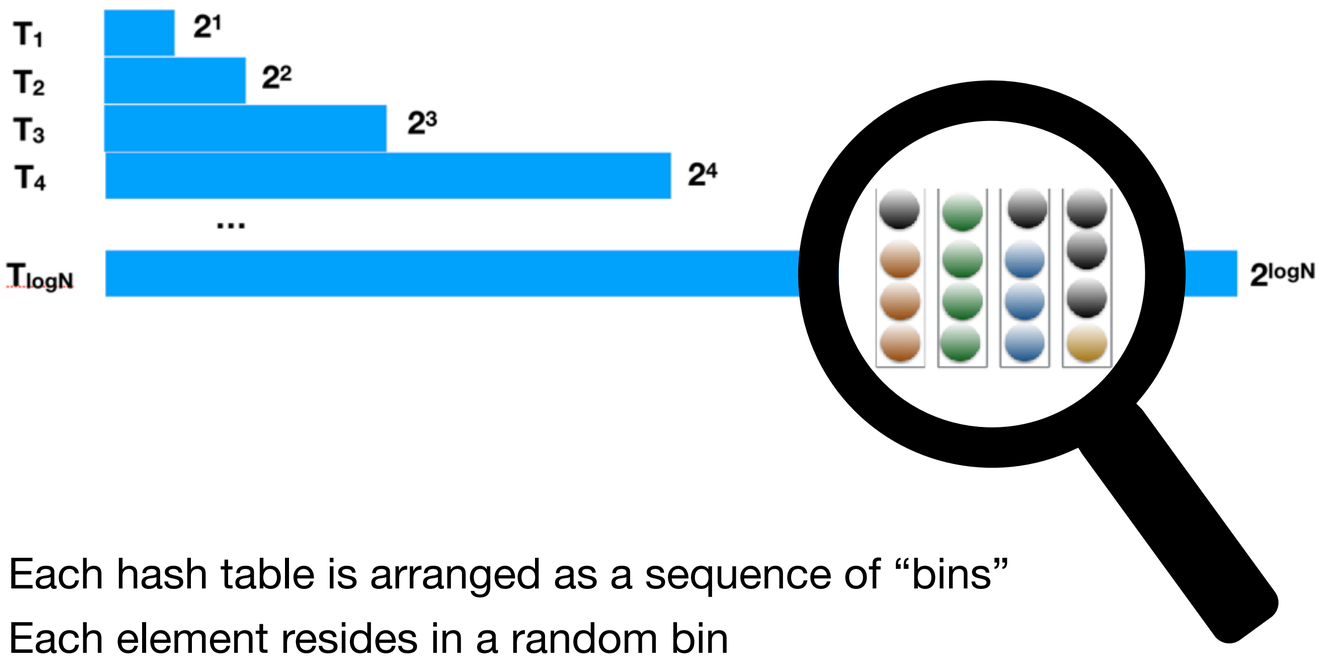
- Can sort in time $O(D/w \cdot n \cdot \log^2 n)$

- When n and D are small (say $n = w^4$ and $D = \log w$), we can sort in linear time! ($\frac{n \log^2 n}{w} \leq n$ vs. $n \cdot \log n$)

Where is it Being Used?



Where is it Being Used?



Each hash table is arranged as a sequence of “bins”

Each element resides in a random bin

The size of each bin is $n = \log^4 N$

Previously: build a structure on a bin using oblivious sort $n \log n$ -> $\log \log N$ overhead

We can remove it using the packing trick

Conclusions

There exists an ORAM with O(log N) blowup
(where N is the size of the logical memory)

🎉 **Asymptotically Optimal!** 🎊

Hash Table
Build in O(n) on permuted input, Lookup in O(1)

Oblivious Tight Compaction in O(n)

Thank you!