ACM-SIAM Symposium on Discrete Algorithms
January 7-10, 2018

Cache-Oblivious and Data-Oblivious Sorting and Applications

T-H. Hubert Chan, Yue Guo, ***Wei-Kai Lin***, and Elaine Shi

香 港 大 學
THE UNIVERSITY OF HONG KONG

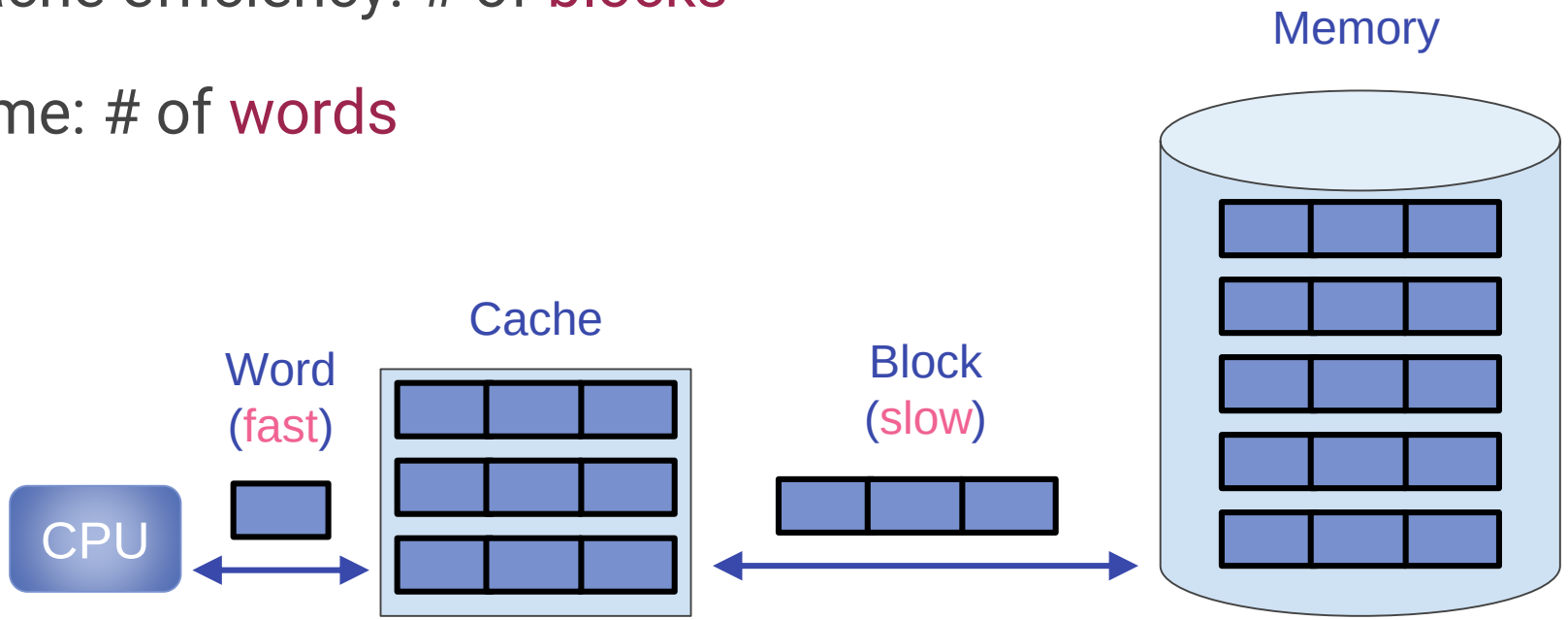
Cornell University

Jan 10, 2018

External Memory Model

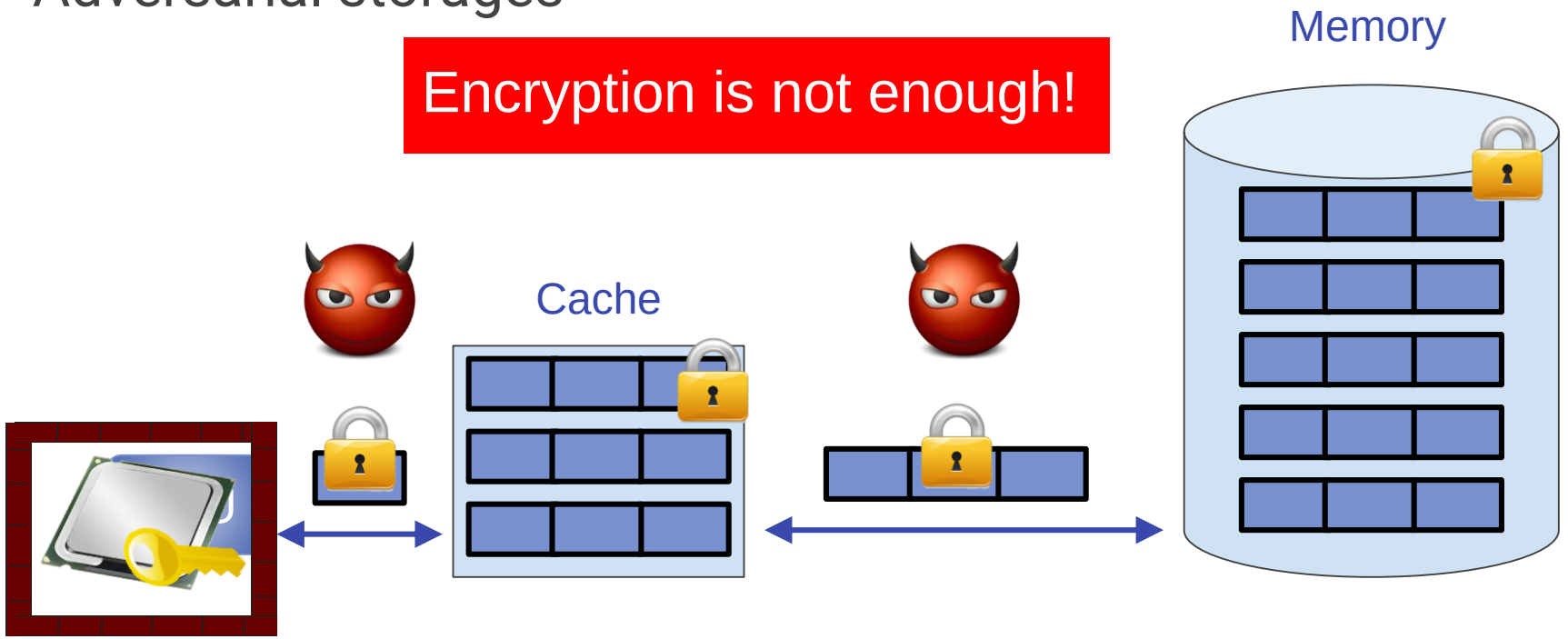
Cache efficiency: # of blocks

Time: # of words



Data-Obliviousness

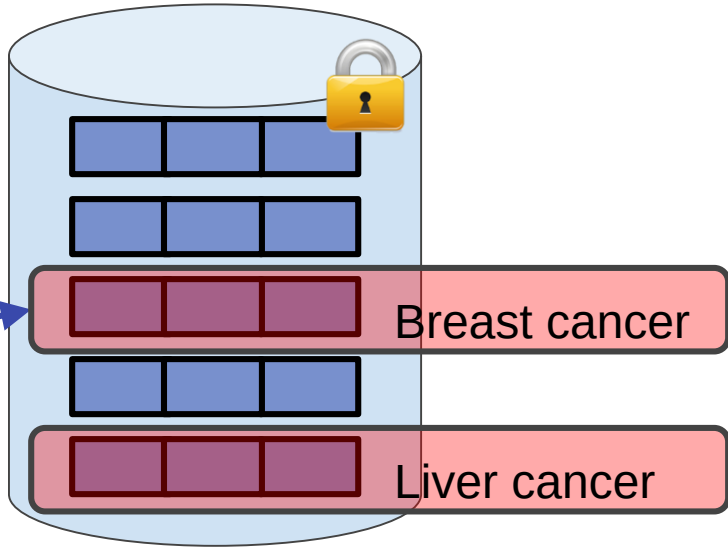
Adversarial storages



Data-Obliviousness

Access patterns leak sensitive information

Read, address
Write, address

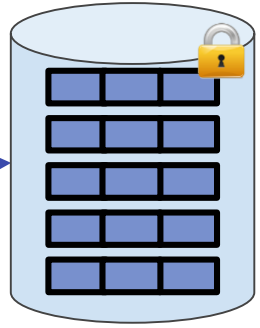


Definition of Data-Oblivious


[Goldreich, Ostrovsky '96]

Access patterns should be simulatable

Memory



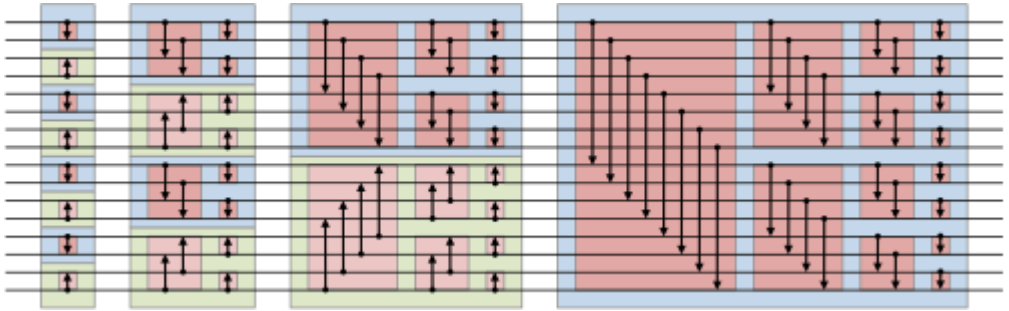
Read, address
Write, address

Statistically close

Read, address
Write, address

Example: Data-Oblivious Sort

Emulated sorting circuits



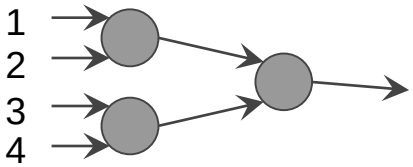
Bitonic sort

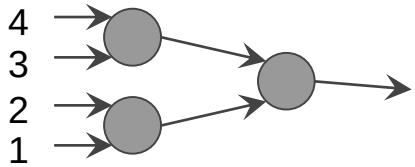
[Batcher ‘68]

AKS sort

[Ajtai, Komlós, Szemerédi ‘83]

~~Merge sort (or Funnel sort)~~



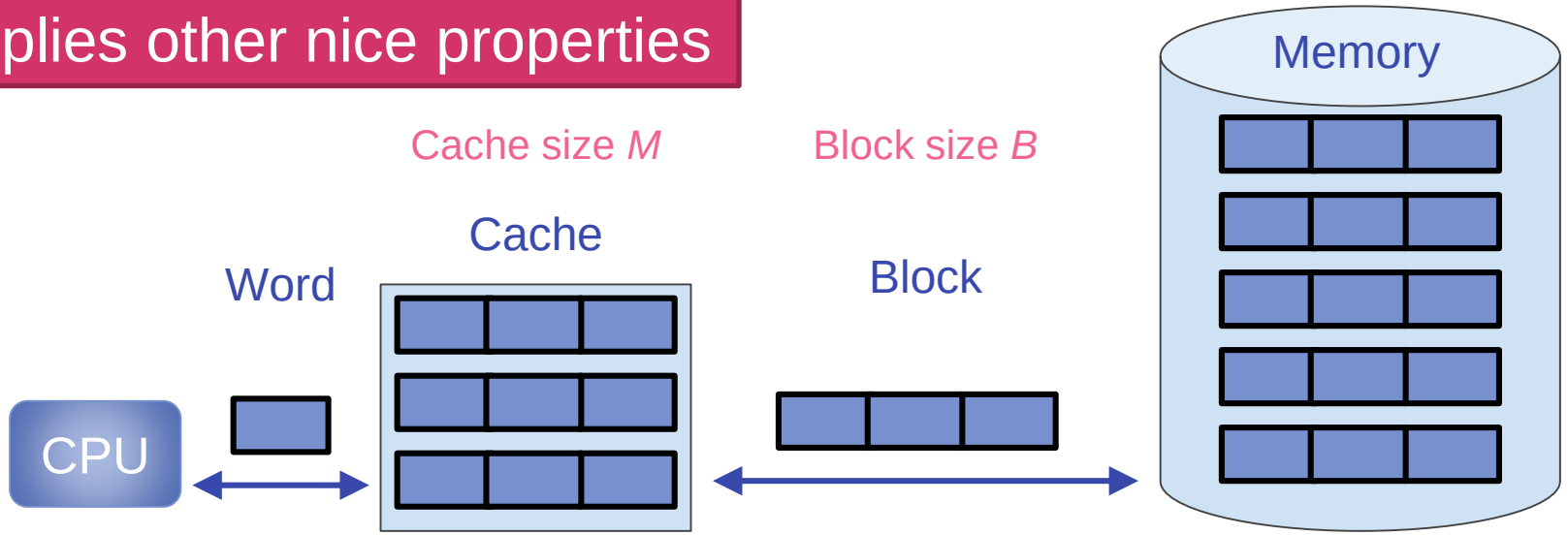


Cache-Oblivious and ***Cache-Efficient***

[Frigo, Leiserson, Prokop, Ramachandran '99]

Unknown cache size and block size

Implies other nice properties

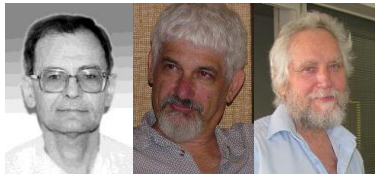


Sorting Alorithms

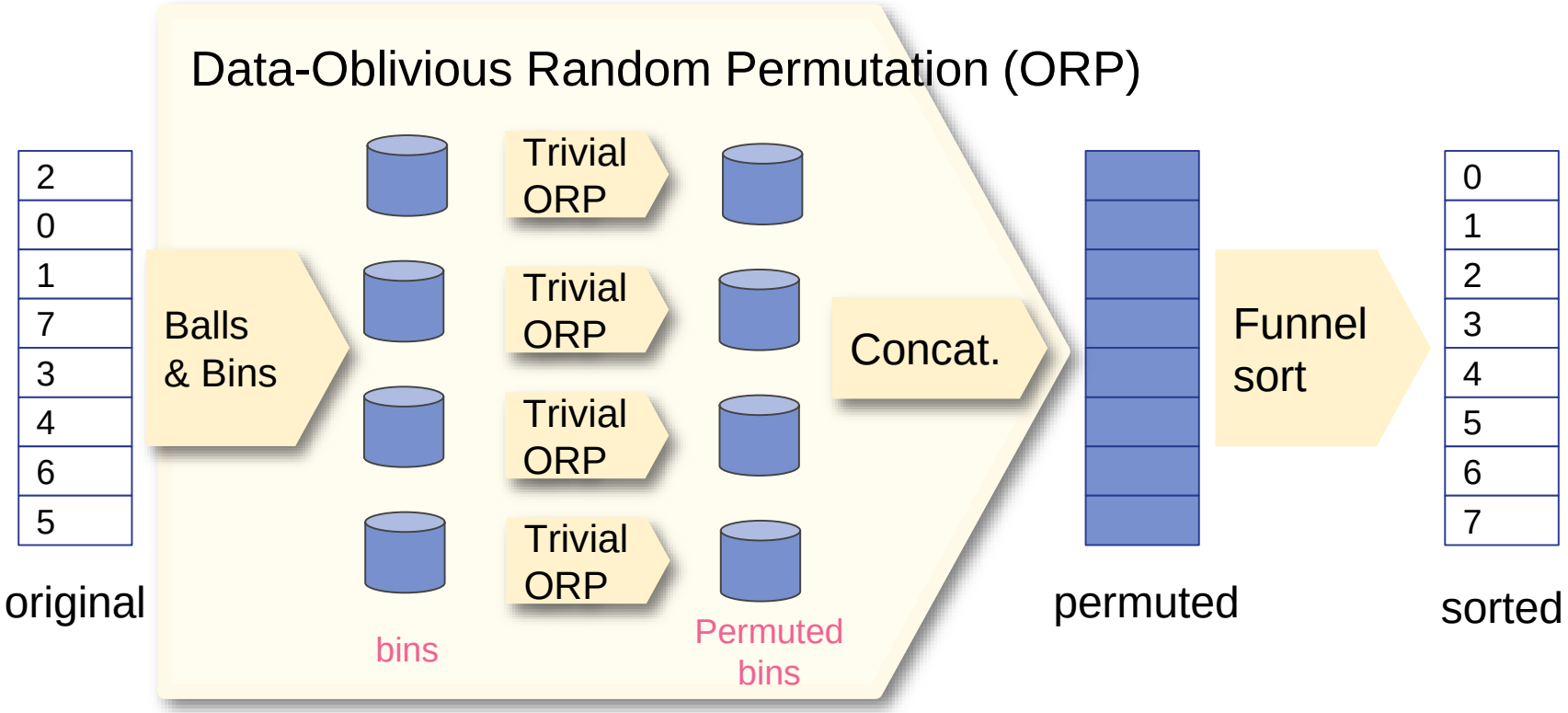
	Data-oblivious	Cache-efficient
 [Ajtai, Komlós, Szemerédi ‘83]	Yes	No
 [Frigo, Leiserson, Prokop, Ramachandran ‘99]	No	$O\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$  Lower bound [Aggarwal, Vitter ‘88]

Can we construct comparison-based, cache-oblivious and *data-oblivious* sorting that is optimal in *cache effcncy*?

Sorting Alorithms

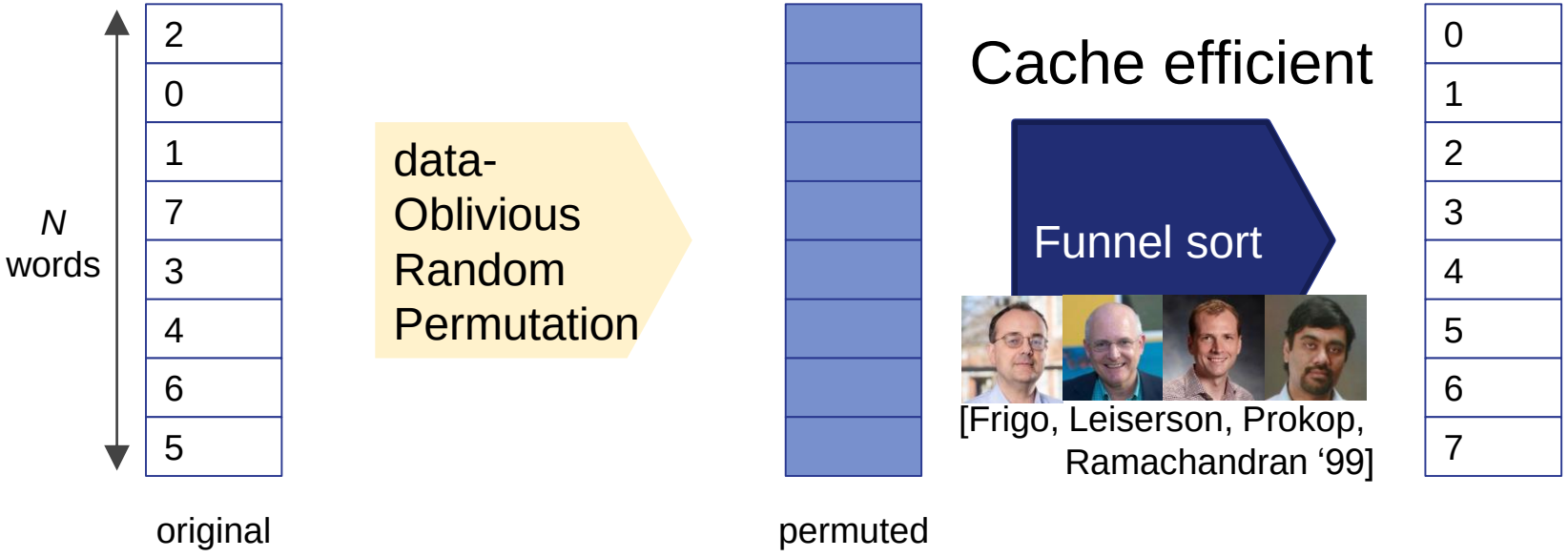
	Data-oblivious	Cache-efficient
 [Ajtai, Komlós, Szemerédi ‘83]	Yes	No
 [Frigo, Leiserson, Prokop, Ramachandran ‘99]	No	$O\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$  Lower bound [Aggarwal, Vitter ‘88]
This work	Yes	$O\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$

Overview of Cache-Efficient Sort



Access pattern only depends on ordering

New permutation hides ordering



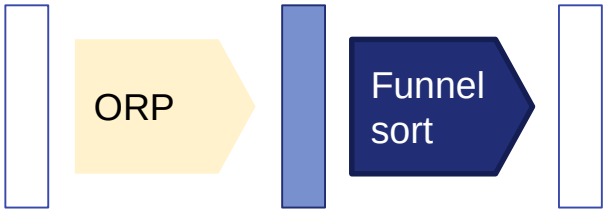
[Frigo, Leiserson, Prokop, Ramachandran ‘99]

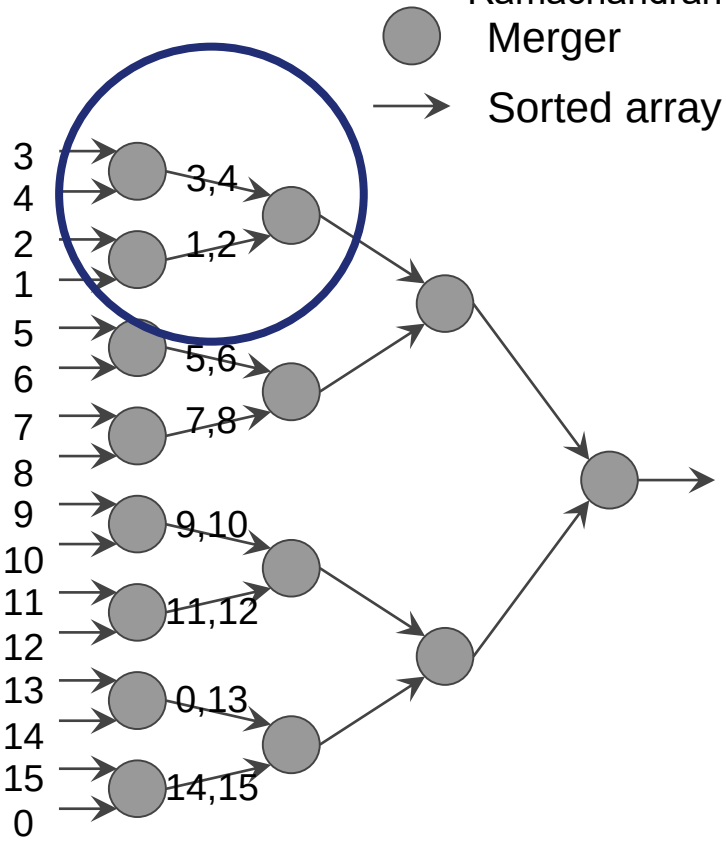
Idea: Funnel Sort

Main body: Divide and Merge

- Merge-sort
- For cache efficiency:
  Solve subproblem in cache

Challenge: unknown cache size





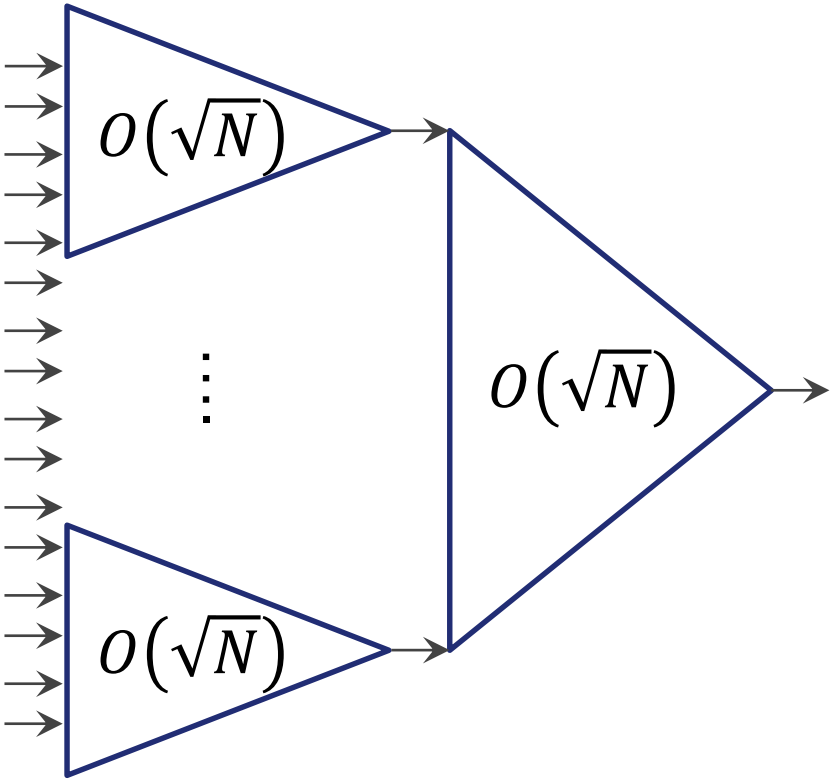
[Frigo, Leiserson, Prokop, Ramachandran ‘99]

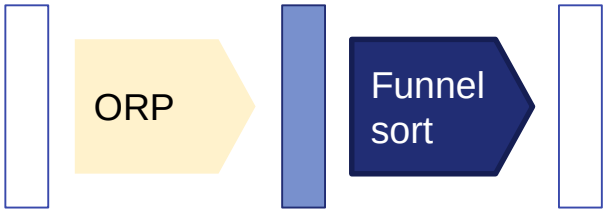
Idea: Funnel Sort

Main body: Divide and Merge

- Merge-sort
- For cache efficiency:
  Solve subproblem in cache

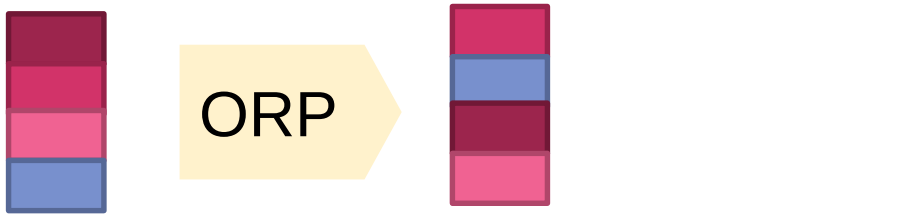
Solution: smart recursion





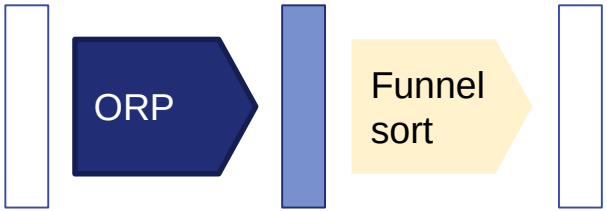
Data-Oblivious Random Permutation (ORP)

Randomness: uniformly random shuffle



Input array

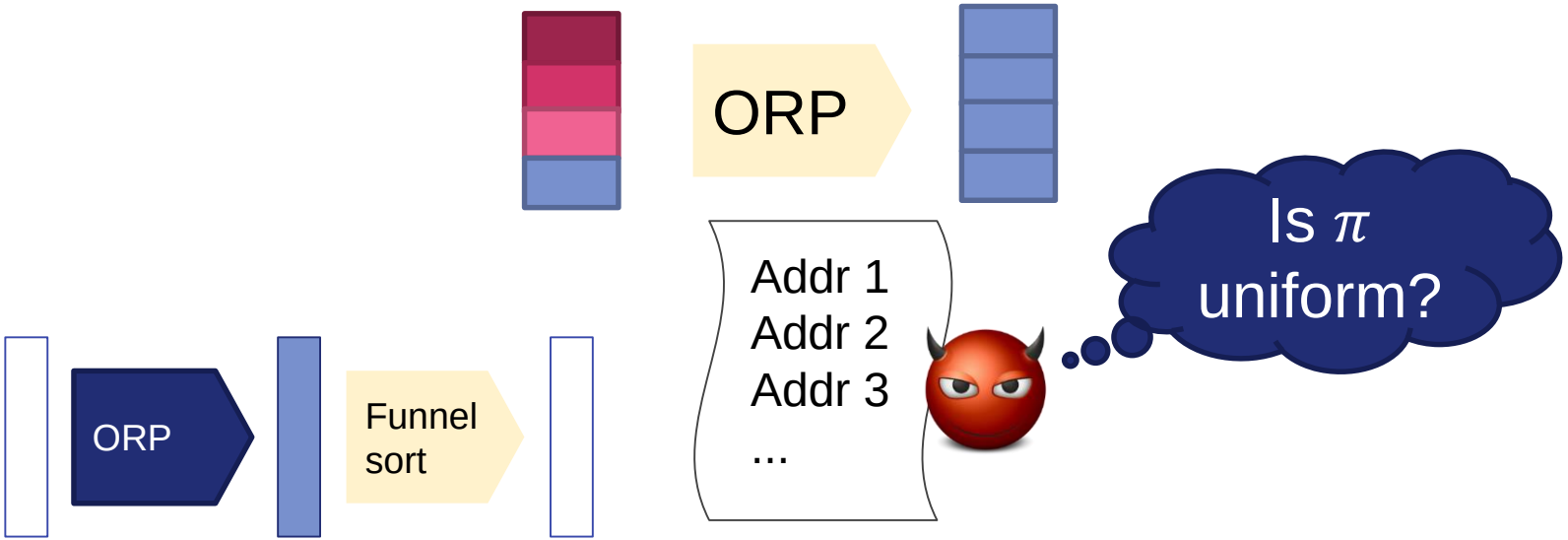
Output: π(Input),
π: permutation



Data-Oblivious Random Permutation (ORP)

Randomness: uniformly random shuffle

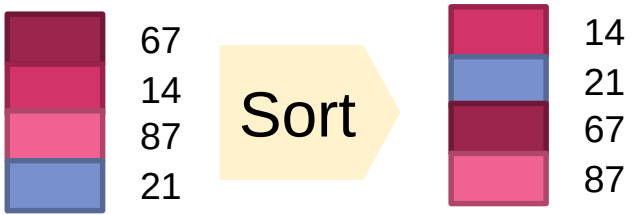
Data-Oblivious: unknown π given access pattern

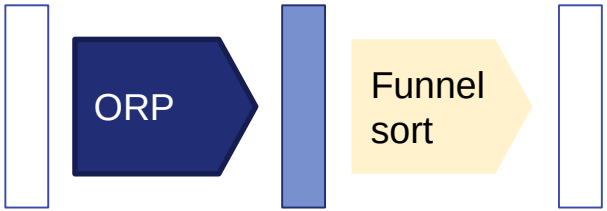


Warmup: Trivial ORP

(Data-oblivious) Sort random keys

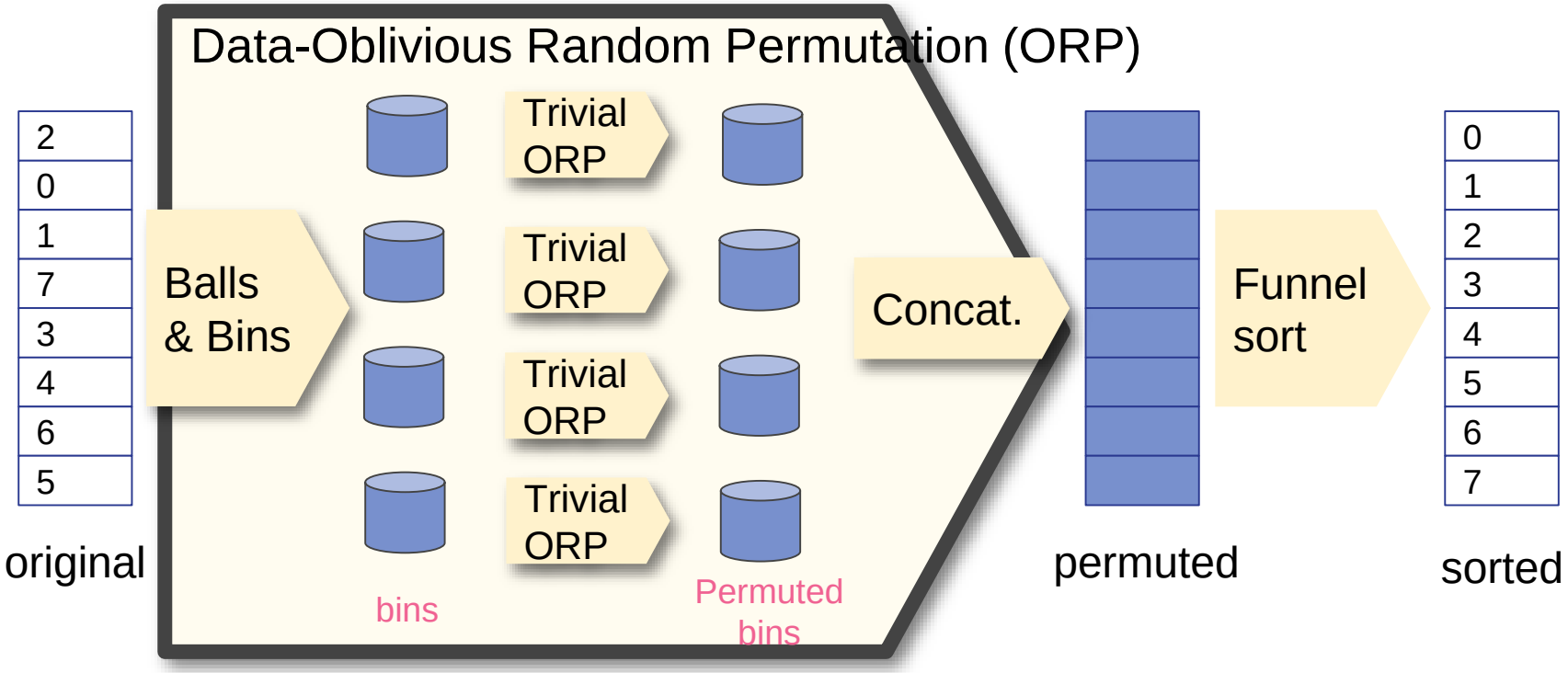
Not cache-efficient!





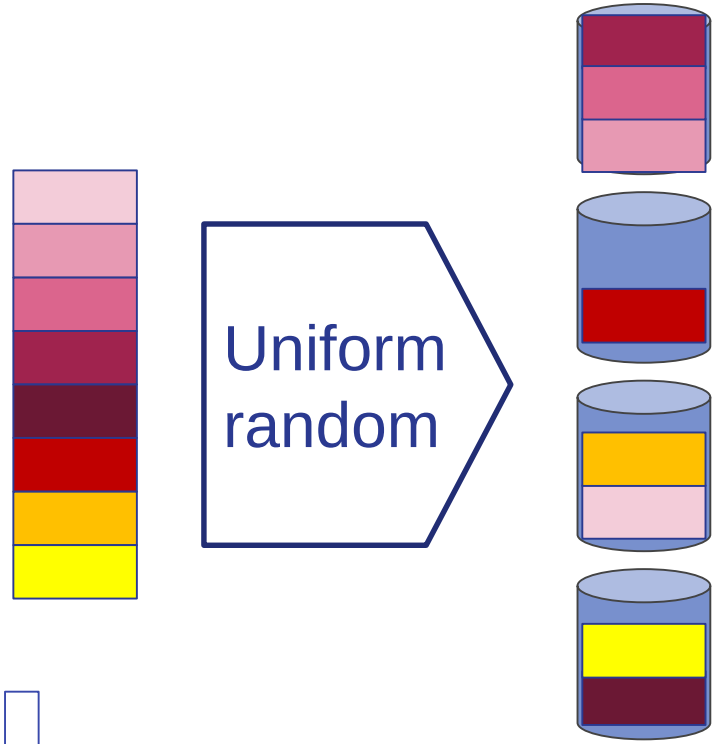
Cache-Efficient ORP

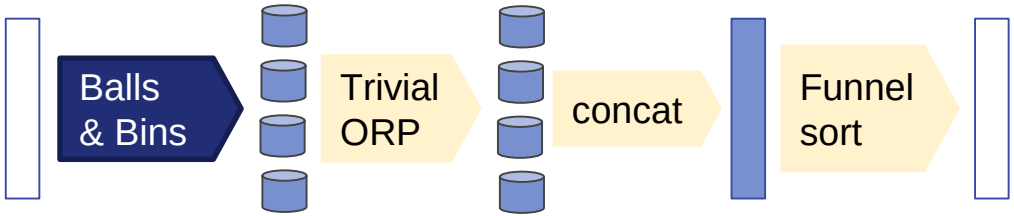
Balls & Bins + Trivial ORP inside each bin



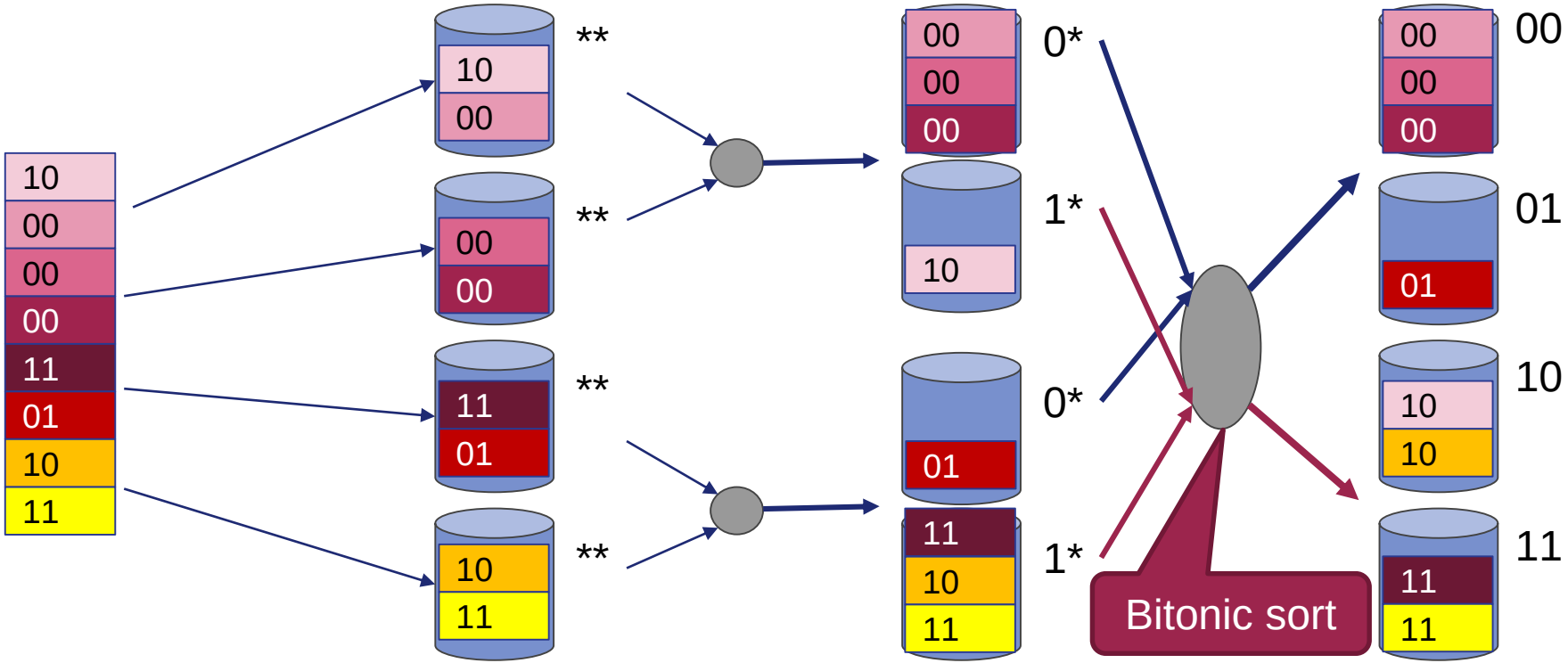
Ball & Bins

- Input: N balls
- Output: N balls in n bins, each with capacity Z





Ball & Bins (Bucket Funnel Sort)



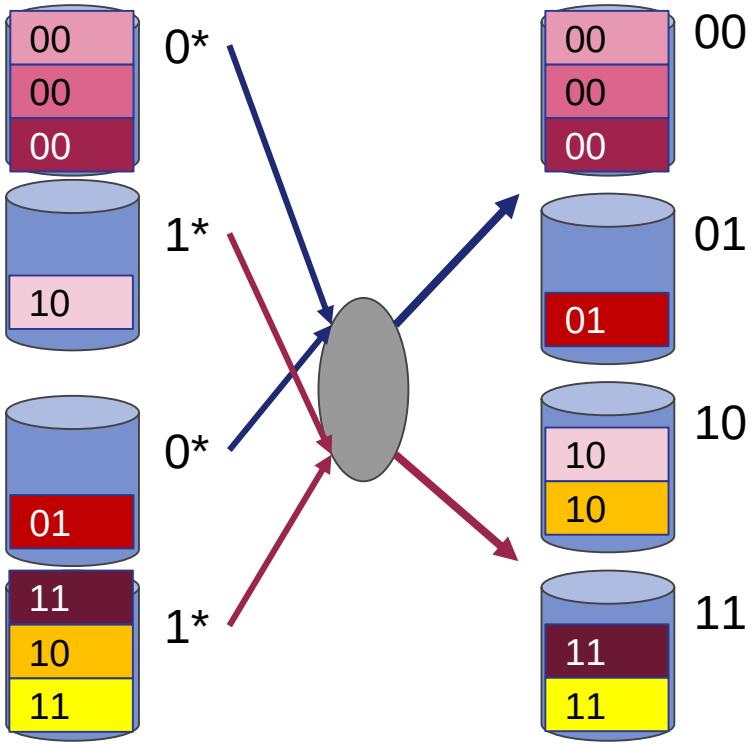
Merge two same-prefix bins

Ball & Bins: Data-Oblivious

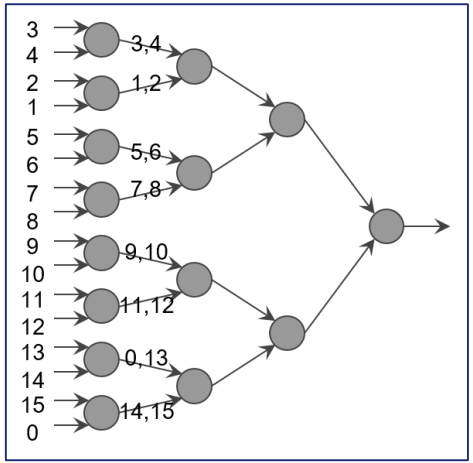
- Bitonic sort: data-oblivious
- Merging bins: fixed pattern

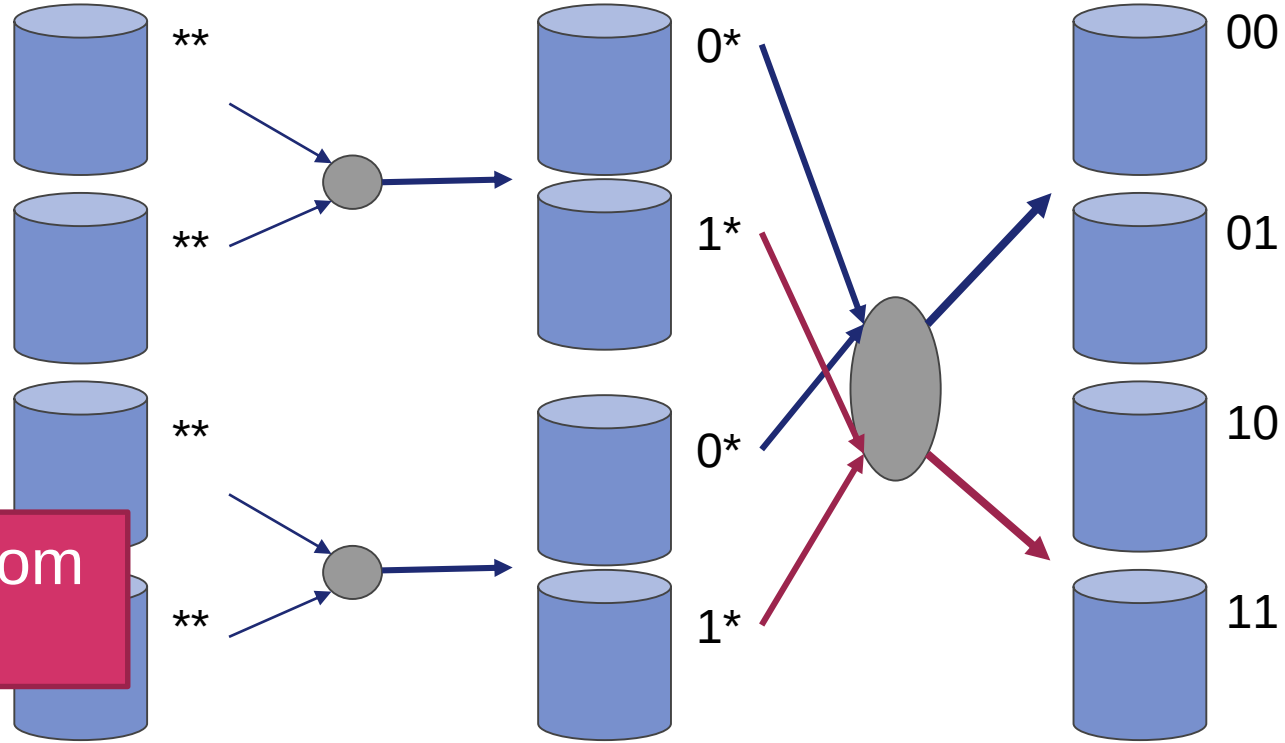
Data-Oblivious ☺

Negligible overflow probability:
Bin size $Z = \omega(\log \lambda)$
λ: security parameter, ~1,000



Ball & Bins: Cache Efficiency



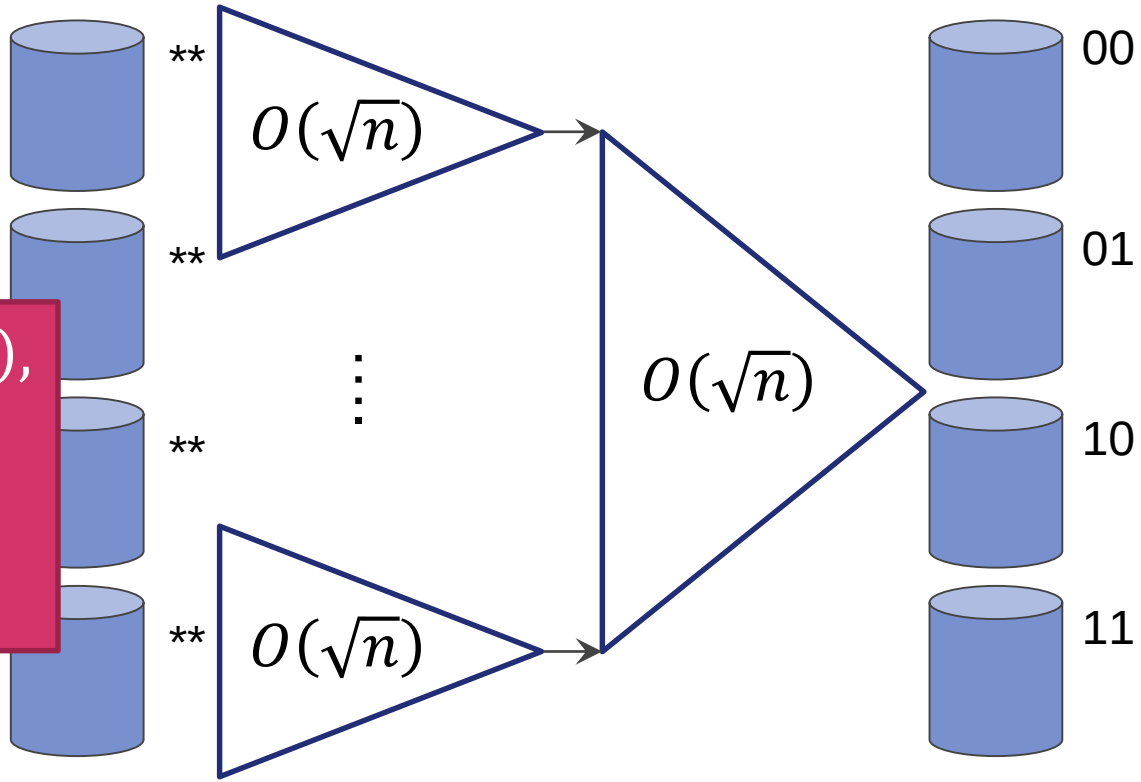


Borrow recursion from Funnel Sort ☺

Ball & Bins: Cache Efficiency

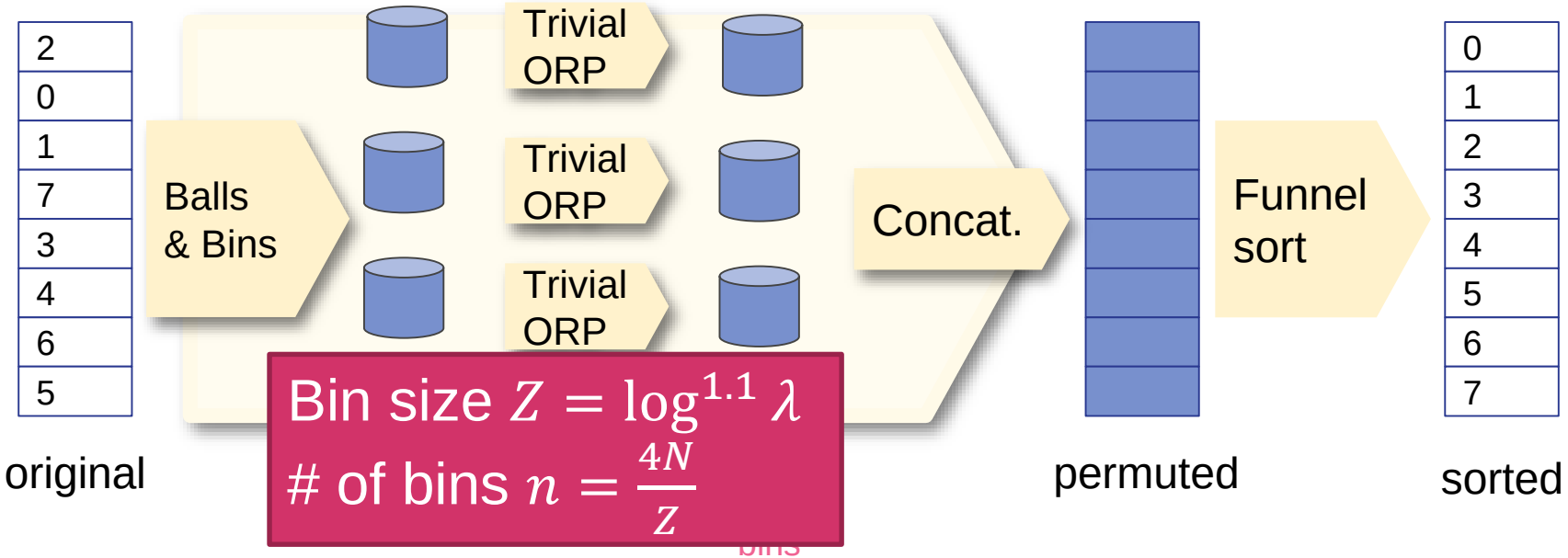
Assume $M = \Omega(\log^{1.1} \lambda)$,
Cache efficiency is

$$O\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$$

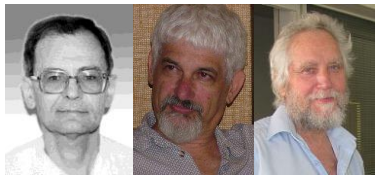


Cache-Efficient and Data-Oblivious Sort

Simple algo: Balls & Bins + Trivial ORP + Funnel sort



Sorting Alorithms

	Data-oblivious	Cache-efficient
 [Ajtai, Komlós, Szemerédi ‘83]	Yes	No
 [Frigo, Leiserson, Prokop, Ramachandran ‘99]	No	$O\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$
This work	Yes	$O\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$
Comparison-based, cache-oblivious		Time: $O(N\log N\cdot\log^2\log\lambda)$

Applications

- Cache-efficient data-oblivious RAM compiler: $O(\log N)$
- Other data-oblivious frameworks
  - MapReduce
  - GraphSC

Summary

This work:

Cache-efficient, data-oblivious sort

Open:

Cache-efficient, data-oblivious sort and optimal in time?

Followup:

Data-oblivious sorting in $o(N \log N)$ time (non-comparison-based)

Thank you!