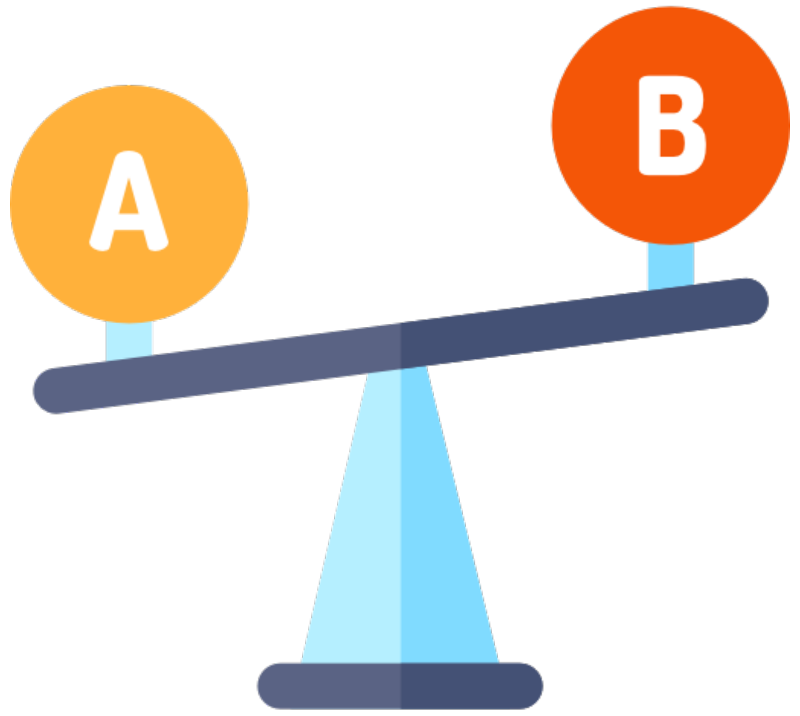
External-Memory Sorting with Comparison Errors

Michael T. Goodrich and Evrim Ozel
Department of Computer Science
University of California, Irvine

Noisy Sorting Framework

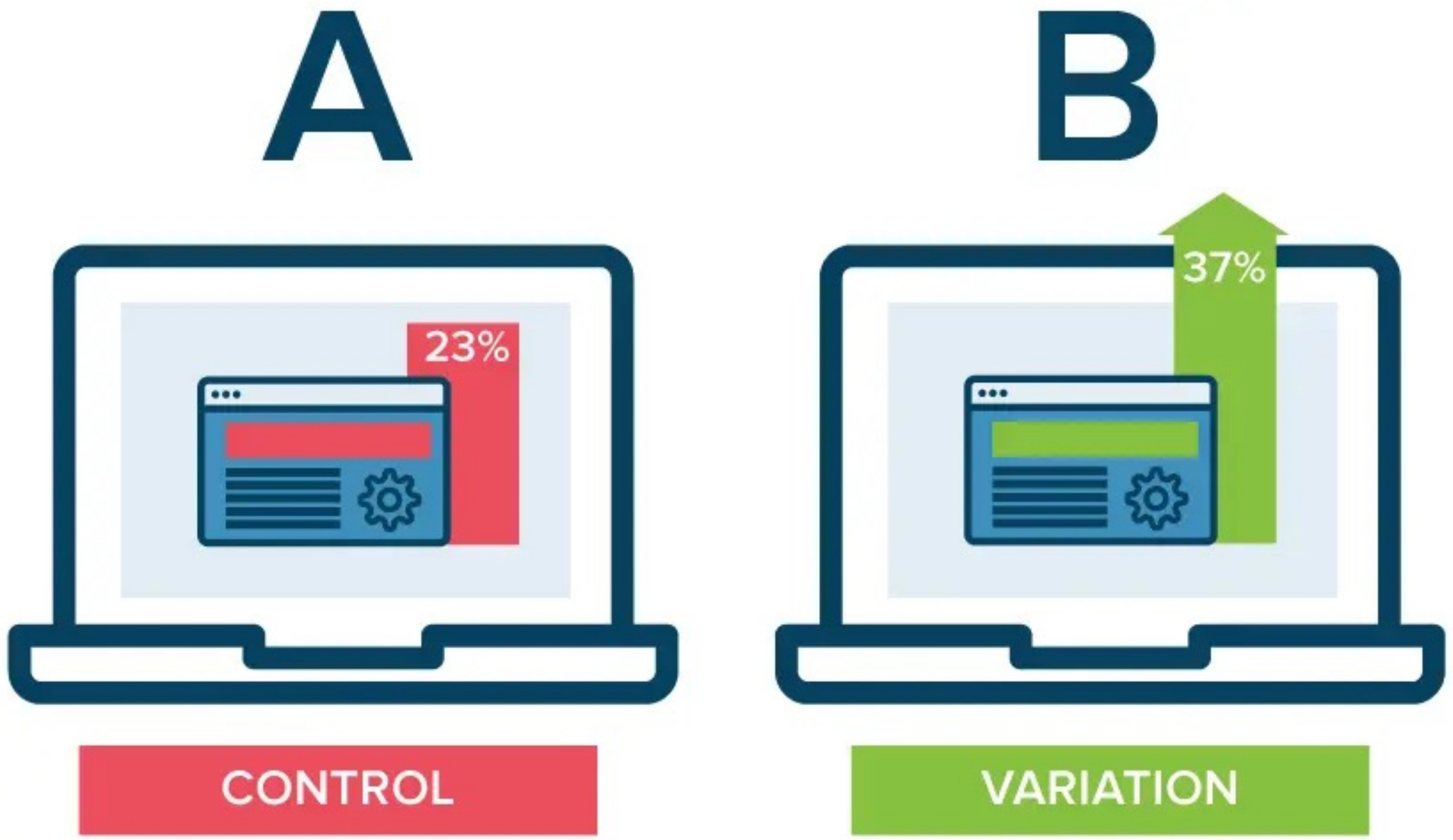
- Sort n distinct comparable elements
- Comparing two elements x, y outputs true result independently according to a fixed probability $p < 1/2$, otherwise outputs false (opposite) result



- **Non-persistent** errors: determination of correctness made independently for each comparison
- **Persistent** errors: if previously compared pair of elements (x, y), return that result instead

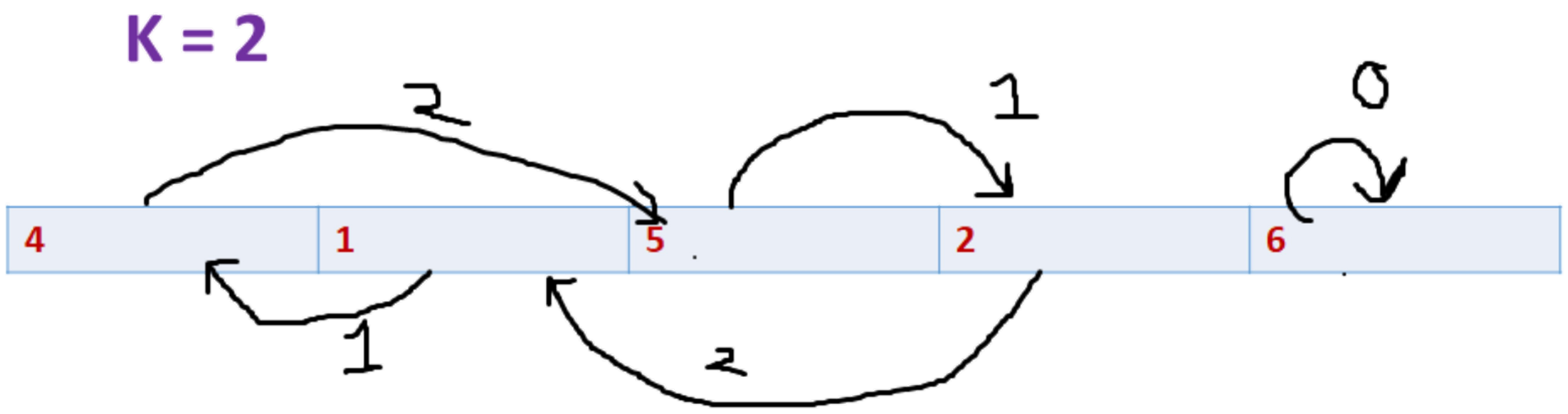
Noisy Sorting Framework: Motivation

- A/B Testing
- Sport ranking



Noisy Sorting Framework: Evaluation

- **Dislocation** of an element x: distance between current position and sorted position
- **Maximum dislocation** of an array
- **Total dislocation** of an array

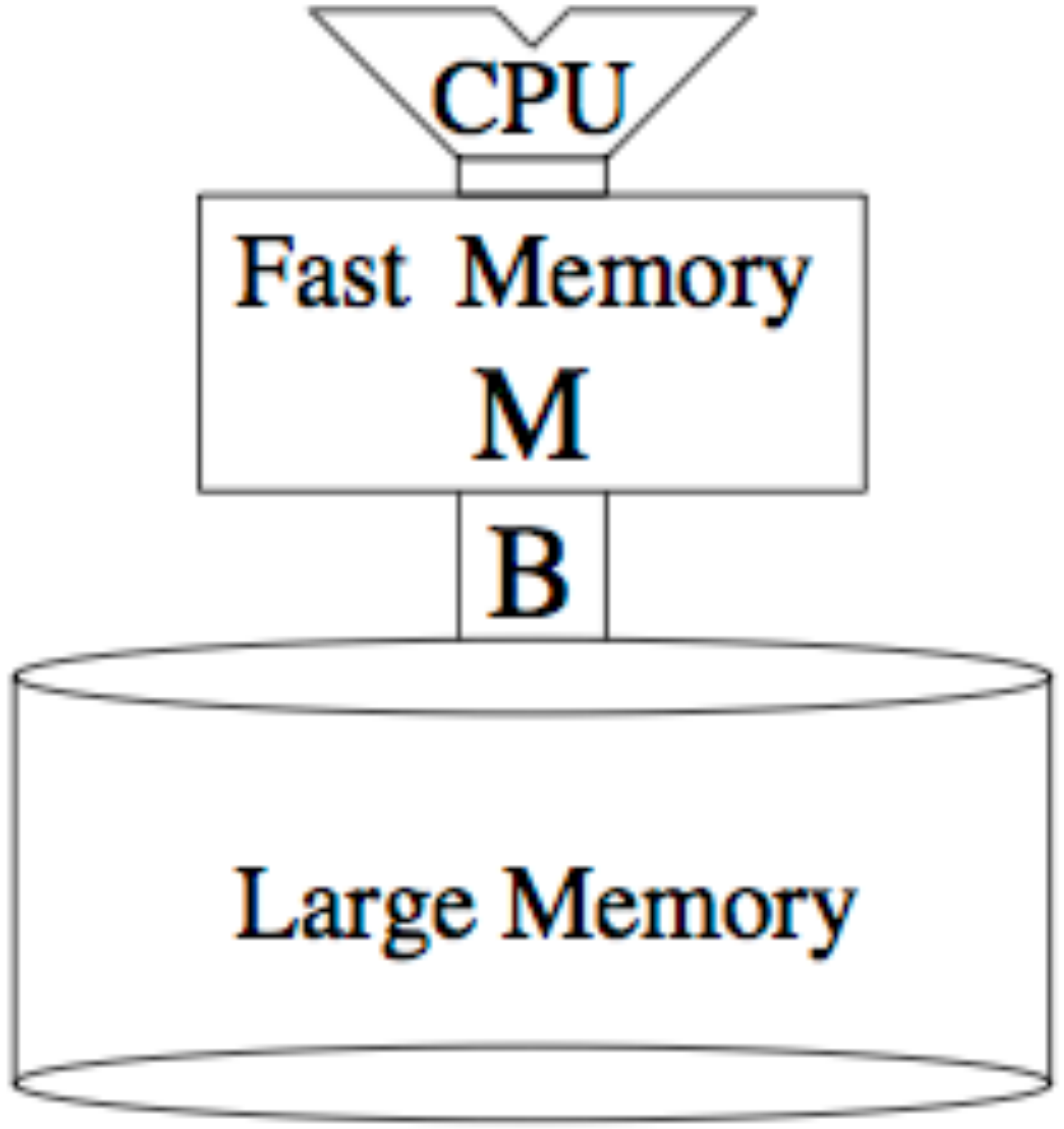


- Worst-case of $O(n)$ and $O(n^2)$ respectively
- Known lower bounds of $\Omega(\log n)$ and $\Omega(n)$ for best-possible max and total dislocation under persistent comparison errors

Noisy Sorting Framework in External Memory

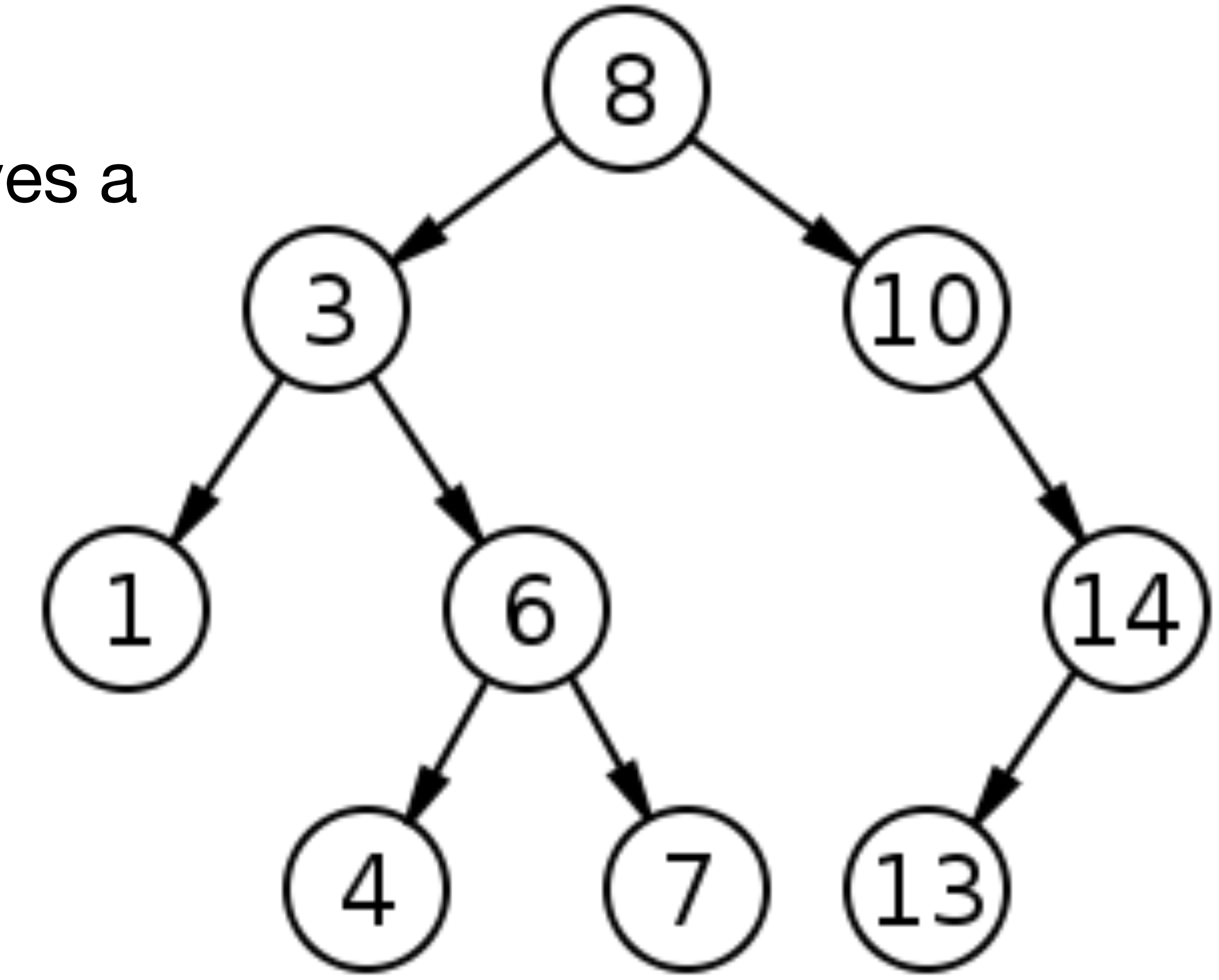
- Goal: sort n items in external memory model using an optimal $\Theta((N/B)\log_{M/B}(N/B))$ I/Os, while also being tolerant to noisy comparisons

- Interested in both cache-aware (parameters involve M and B) and cache-oblivious settings



Noisy Sorting Framework in External Memory

- Existing algorithms cannot be easily converted into an external memory algorithm
  - They make use of **noisy binary search**, which involves a random walk in a BST
  - Not cache efficient and takes $240 \log n$ steps



Related Prior Work

Internal Memory	Time	Max Disloc.	Total Disloc.	I/Os
Braverman, Mossel (2008)	$O(n^{3+f(p)})$	$O(\log n)$	$O(n)$	–
Klein, Penninger, Sohler, Woodruff (2011)	$O(n^2)$	$O(\log n)$	$O(n \log n)$	–
Geissmann, Leucci, Liu, Penna (2019)	$O(n \log n)$	$O(\log n)$	$O(n)$	–
External Memory				
Aggarwal, Vitter (1988) (cache-aware)	$O(n \log n)$	–	–	$O((N/B)\log_{M/B}(N/B))$
Leiserson, Frigo, Prokop (1999) (cache-oblivious)	$O(n \log n)$	–	–	$O((N/B)\log_{M/B}(N/B))$
Our work (cache-oblivious and cache-aware)	$O(n \log^2 n)$	$O(\log n)$	$O(n \log n)$	$O((N/B)\log_{M/B}(N/B))$

Preliminary: Window-Sort

- Takes as input an array of size n with max dislocation at most $d_1 \leq n$ [Geissmann, Leucci, Liu, Penna (2019)]
- Outputs array with max dislocation at most $d_2/2$ w.h.p. as a function of d_2 (typically choose $d_2 = \Theta(\log n)$)
- Takes $O(d_1 n)$ time in internal memory, and uses $O((nd_1/B) + (\log(d_1/d_2))(n/B)\log_{M/B}(n/B))$ I/Os in external memory
- Allows us to achieve noise tolerance for our later algorithms

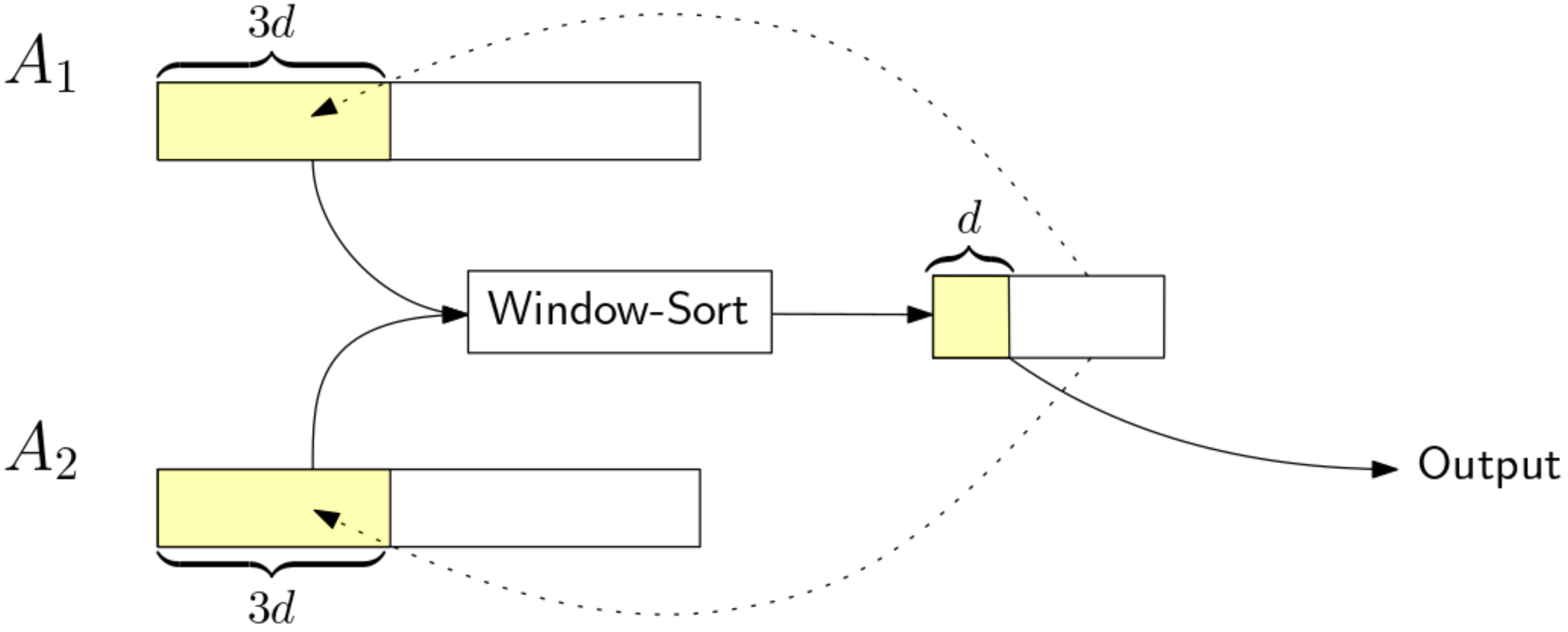
Algorithm 1: Window-Sort($A = \{a_0, a_1, \ldots, a_{n-1}\}, d_1, d_2$)

for $w \leftarrow 2d_1, d_1, d_1/2, \ldots, 2d_2$ **do**
  foreach $i \leftarrow 0, 1, 2, \ldots, n-1$ **do**
    $r_i \leftarrow \max\{0, i - w\} + |\{a_j < a_i \ : \ |j - i| \leq w\}|$
  Sort A (deterministically) by nondecreasing r_i values (i.e., using r_i as the comparison key for a_i)
return A

Window-Merge-Sort

- Variant of merge sort that sorts with max dislocation $O(\log n)$ under persistent errors, w.h.p
- Takes as input parameter d, the desired max dislocation, we choose $d = c\log n$ for some constant $c > 0$
- Not cache-oblivious, but sorts in external memory with optimal #I/Os, assuming $B = \Omega(\log n)$
- We first consider an internal memory version that runs in $O(n \log^2 n)$ time
- Given array A, split it into 2 subarrays A_1, A_2 of roughly equal size and recursively sort them
- Merge sublists by using Window-Sort as subroutine:
  - If $|A_1| + |A_2| \leq 6d$, return Window-Sort$(A_1 \cup A_2, 4d, d)$

Window-Merge-Sort: merge step



Window-Merge-Sort: merge step and key lemma

while $|A_1| + |A_2| > 6d$ **do**
    Let S_1 be the first $\min\{3d, |A_1|\}$ elements of A_1
    Let S_2 be the first $\min\{3d, |A_2|\}$ elements of A_2
    Let $S \leftarrow S_1 \cup S_2$
    Window-Sort$(S, 4d, d)$
    Let B' be the first d elements of (the near-sorted) S
    Add B' to the end of B and remove the elements of B' from A_1 and A_2

Lemma: If A_1 and A_2 each have max dislocation at most $3d/2$, then merging them will result in a sequence with max dislocation at most $3d/2$ w.h.p.

Proof: Omitted

External Window-Merge-Sort

- Divide A into $m = \Theta(M/B) \geq 2$ subarrays $A_1, \ldots, A_m$ instead, each of roughly equal size
- For the merge step, bring in the first $max\{3d, |A_i|\}$ elements from each A_i into a list S
- Call Window-Sort$(S, 4md, d)$. Since $B = \Omega(\log n)$, fits entirely in internal memory
- Output the first d elements from that call and repeat

Lemma: If $A_1, \ldots, A_m$ each have max dislocation at most $3d/2$, then merging them will result in a sequence with max dislocation at most $3d/2$ w.h.p.

External Window-Merge-Sort

Theorem: Given an array A of n distinct comparable elements, one can deterministically sort A in $O(n \log^2 n)$ time in internal memory or in external memory with $O((n/B)\log_{M/B}(n/B))$ I/Os subject to comparison errors with $p \leq 1/16$ so as to have max dislocation at most $O(\log n)$ w.h.p., assuming $B \geq \log n$.

Funnelsort and Cache-Obliviousness

- Introduced by Frigo, Leiserson and Prokop in 1999
- Cache-Oblivious algorithms do not contain parameters dependent on M or B that can be tuned to optimize performance (e.g. by loop tiling: breaking problem into optimally sized blocks for a given cache)
- Such algorithms usually use divide-and-conquer: divide problem into smaller pieces until subproblem fits into cache, regardless of cache size
- Analysis: work complexity $W(n)$, cache complexity $Q(n)$
- Funnelsort: cache-oblivious sorting algorithm, $W(n) = O(nlogn)$ and $Q(n) = \Theta((N/B)\log_{M/B}(N/B))$, requires tall-cache assumption, $M = \Omega(B^2)$

Preliminary: Funnelsort

- Split input into $n^{1/3}$ arrays of size $n^{2/3}$, sort them recursively
- Merge the $n^{1/3}$ sorted sequences using a $n^{1/3}$-merger
- k-merger: recursive data structure that merges k sorted sequences
- Idea: to achieve noise tolerance, use Window-Sort for base case k-mergers

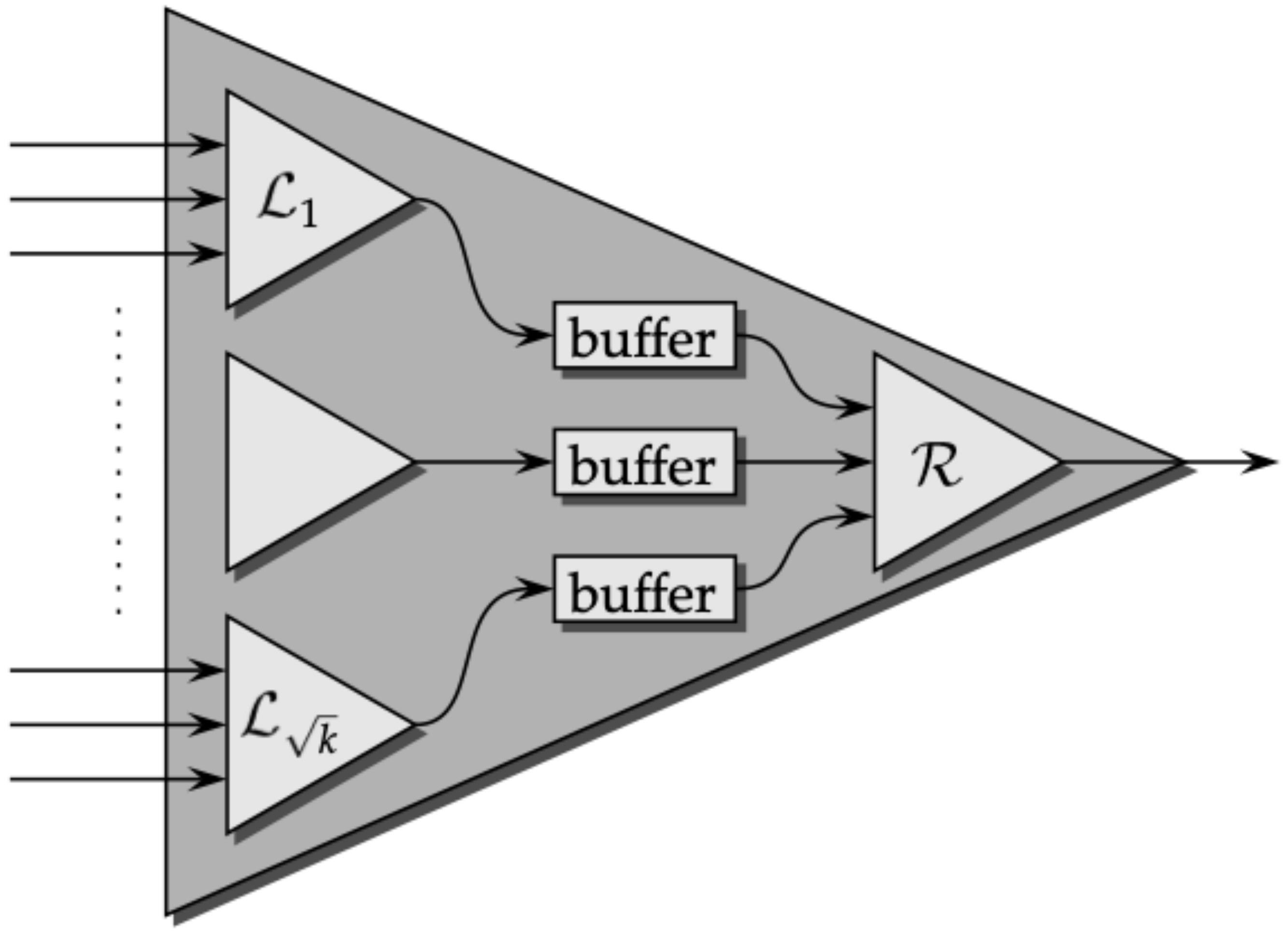


Figure 4-1: Illustration of a k-merger. A k-merger is built recursively out of $\sqrt{k}$ left $\sqrt{k}$-mergers $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_{\sqrt{k}}$, a series of buffers, and one right $\sqrt{k}$-merger $\mathcal{R}$.

Constructing a k-merger

- Invariant: one k-merger invocation outputs k^3 elements of merged sequence
- To output k^3 elements, R gets invoked $k^{3/2}$ times
- Each left merger output connected to buffer of size $2k^{3/2}$
- Before each invocation of R, if any buffer i has $< k^{3/2}$ elements, invoke L_i once so that buffer has $\geq k^{3/2}$ elements

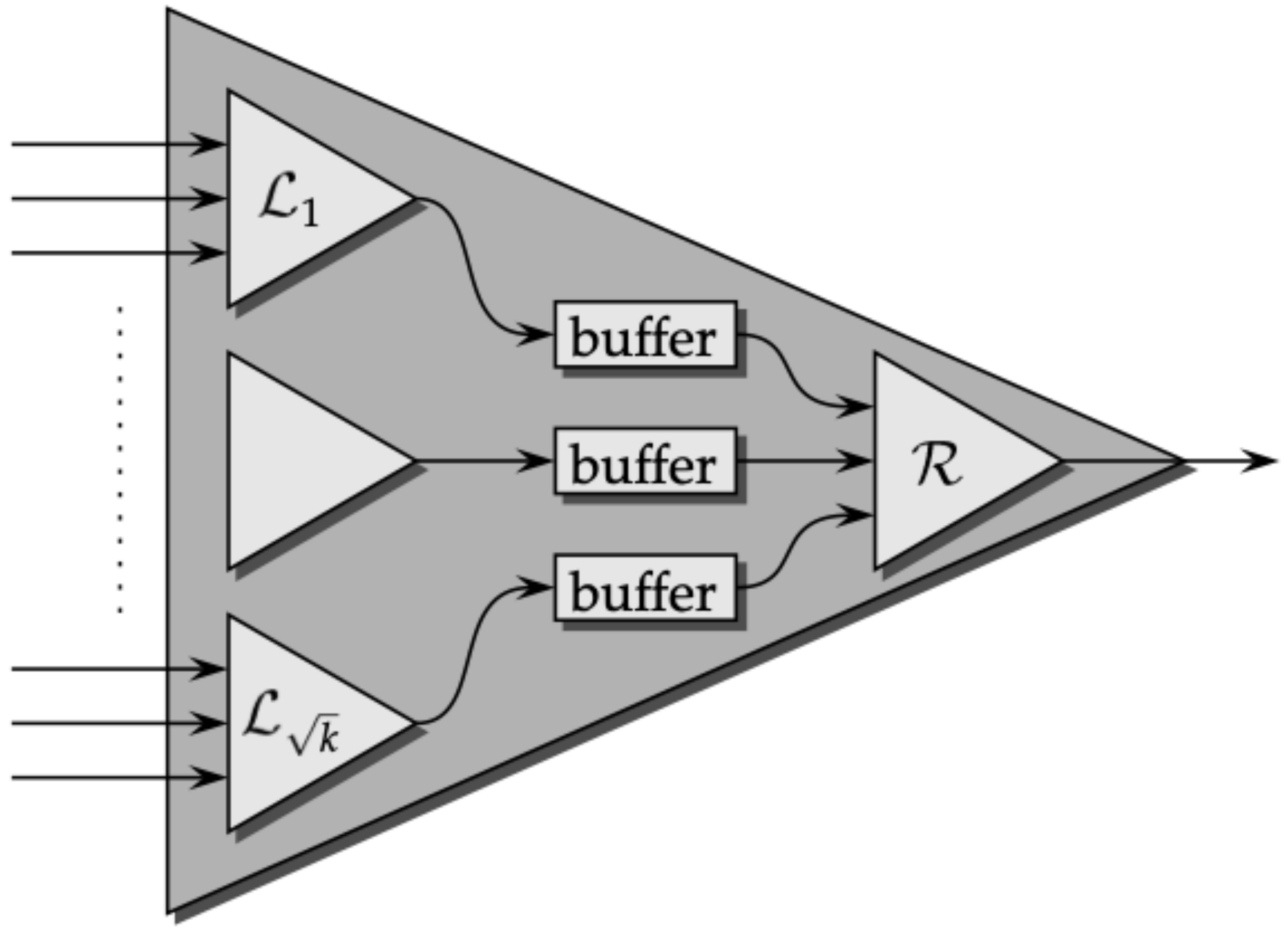


Figure 4-1: Illustration of a k-merger. A k-merger is built recursively out of $\sqrt{k}$ left $\sqrt{k}$-mergers $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_{\sqrt{k}}$, a series of buffers, and one right $\sqrt{k}$-merger $\mathcal{R}$.

Window-Funnelsort

- Cache-oblivious, but requires $B = \Omega(\log n)$ in addition to tall-cache assumption
- Same general structure as Funnelsort: recursively sort $n^{1/3}$ sequences of size $n^{2/3}$, feed into $n^{1/3}$-merger
- Modify k-merger construction such that base-case merges are done using Window-Merge
- Original Funnelsort: base case at $k = 2$, Window-Funnelsort: base cases $\sqrt{c \log n} \leq k < c \log n$ for constant $c > 0$
- k-mergers with $k > c \log n$ work the same (recursive) way as original Funnelsort

Window-Funnelsort

- We prove that Window-Funnelsort:
  - has optimal cache complexity $Q(n) = O((N/B)\log_{M/B}(N/B))$
  - has work complexity $W(n) = O(n\log^2 n)$
  - sorts with optimal maximum dislocation $O(\log n)$ under persistent errors, w.h.p.
- We provide a proof sketch for $Q(n)$, and omit the remaining two proofs for this talk

Cache complexity of a k-merger

Lemma: One invocation of a k-merger incurs $O(k + k^3/B + k^3 \log_M k/B)$ cache misses.

Proof sketch: From the previous lemma, and assuming $B = \Omega(\log n)$ and $M = \Omega(B^2)$, any k-merger with $\sqrt{c \log n} \leq k \leq \alpha\sqrt{M}$ will fit entirely in the cache

In this case, since $B = O(\sqrt{M})$, there are at least $M/B = \Omega(k)$ blocks available for buffers

$\Longrightarrow O(k + k^3/B)$ cache misses for reading/writing k^3 elements

We incur an additional $O(k^2/B)$ cache misses due to the $O(k^2)$ space used by k-merger

Cache complexity of a k-merger

Lemma: One invocation of a k-merger incurs $O(k + k^3/B + k^3 \log_M k/B)$ cache misses.

Proof sketch (cont.): For k-mergers with $k > \alpha\sqrt{M}$, we invoke the internal $\sqrt{k}$-mergers a total of at most $2k^{3/2} + 2\sqrt{k}$ times

The k-merger also needs to check before each invocation of R whether any buffers are empty, which incurs at most $\sqrt{k}$ misses and is repeated $k^{3/2}$ times

We have $Q_k \leq (2k^{3/2} + 2\sqrt{k})Q_{\sqrt{k}} + k^2$, which has solution $Q_k \leq O(k^3 \log_M k/B)$

Cache complexity of Window-Funnelsort

Theorem: Window-Funnelsort incurs at most $Q(n) = O((N/B)\log_{M/B}(N/B))$ cache misses.

Proof sketch: We have $Q(n) = n^{1/3}Q(n^{2/3}) + Q_{n^{1/3}}$, where $Q_{n^{1/3}}$ is # of cache misses of $n^{1/3}$-merger

From the previous lemma we know that $Q_{n^{1/3}} = O(n^{1/3} + n/B + n\log_M n/B)$

So we get $Q(n) = n^{1/3}Q(n^{2/3}) + O(n\log_M n/B)$, which has solution $Q(n) = O((N/B)\log_{M/B}(N/B))$

Conclusions/Future work

- We provided cache-optimal and noise tolerant sorting algorithms in the external memory setting (cache-aware and cache-oblivious)
- Improvements can be made for total dislocation and work complexity
- Can we do without the assumption that $B = \Omega(\log n)$?
- Can other cache-oblivious algorithms be made noise-tolerant? (e.g. cache-oblivious distribution sort)

	Time	Max Disloc.	Total Disloc.	I/Os
Our work (cache-oblivious and cache-aware)	$O(n\log^2 n)$	$O(\log n)$	$O(n\log n)$	$O((N/B)\log_{M/B}(N/B))$