External Sorting

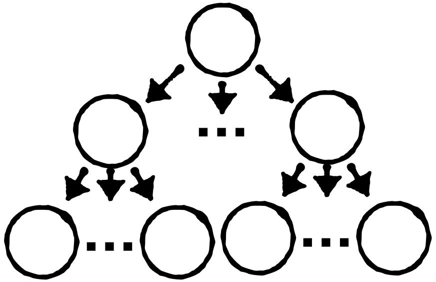
CSC443H1 Database System Technology

Niv Dayan - Feb 14, 2023

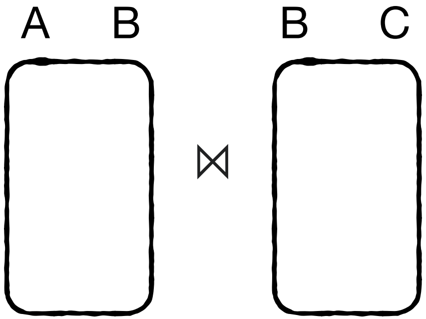
Why do databases need to Sort?

User asking for
sorted output
(Select…order by…)

Creating an Index

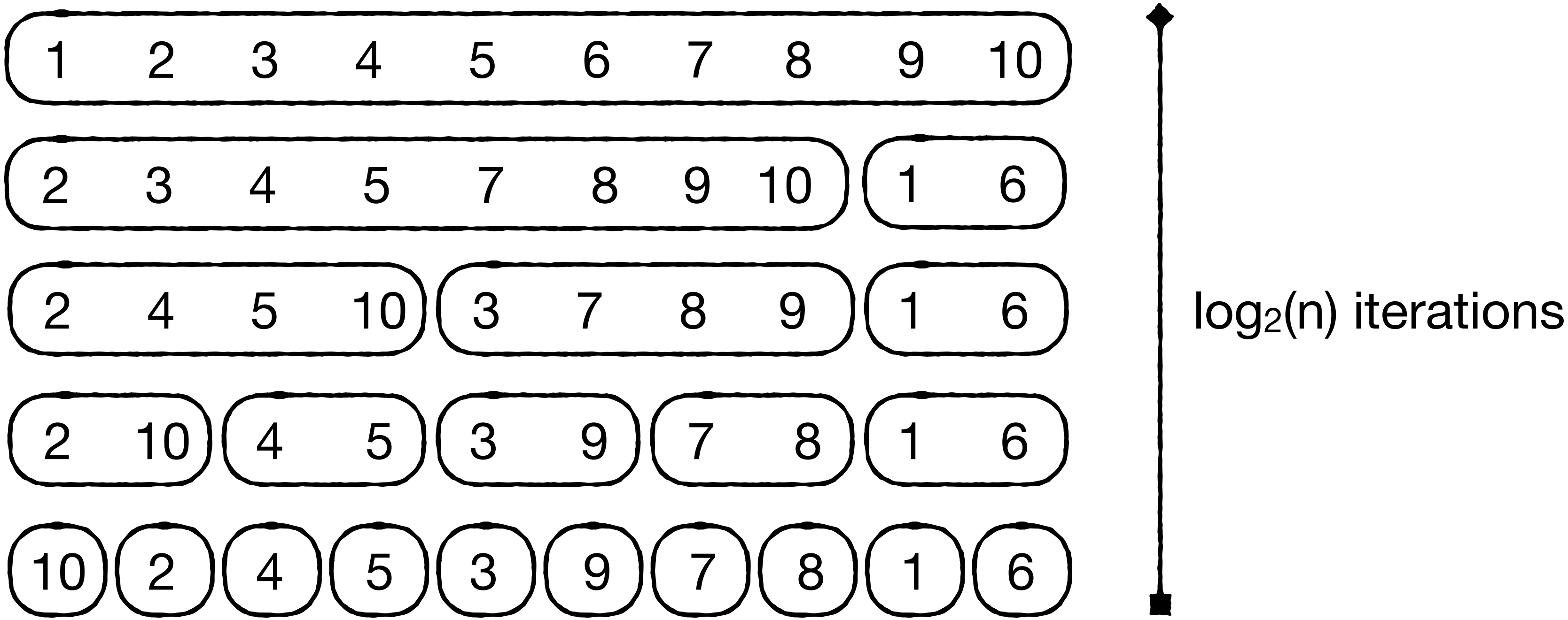


Joining Tables
(More later)



Merge-Sort Analysis

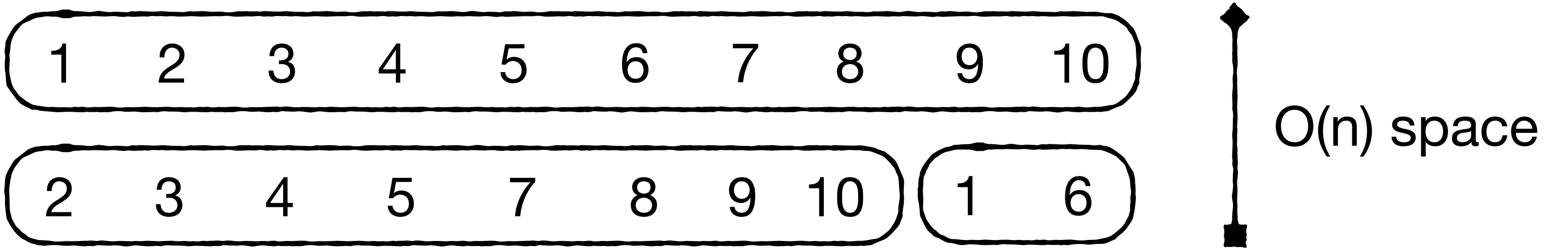
Log(n) iterations, each of which traverses all n elements: O(n $\log_2$(n)) CPU work



Merge-Sort Analysis

Log(n) iterations, each of which traverses all n elements: O(n $\log_2$(n)) CPU work

We need O(n) space by maintaining at most two lists at a time



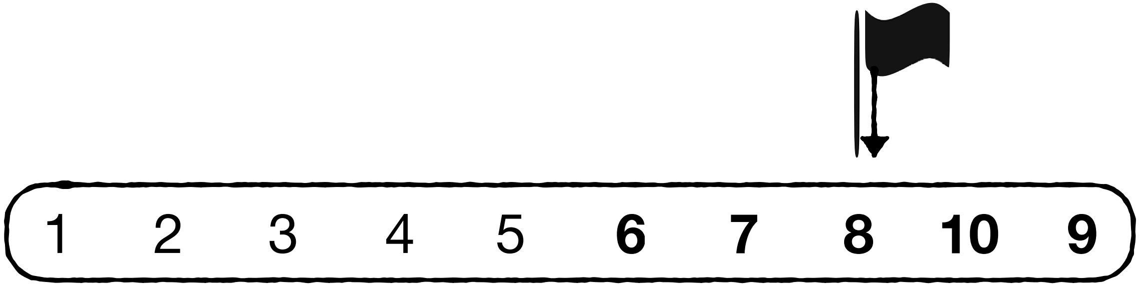
Quick-Sort

1. Pick random pivot point and initialize two pointers at ends

2. Move each pointer towards pivot, stopping at first out-of-order value respect to pivot

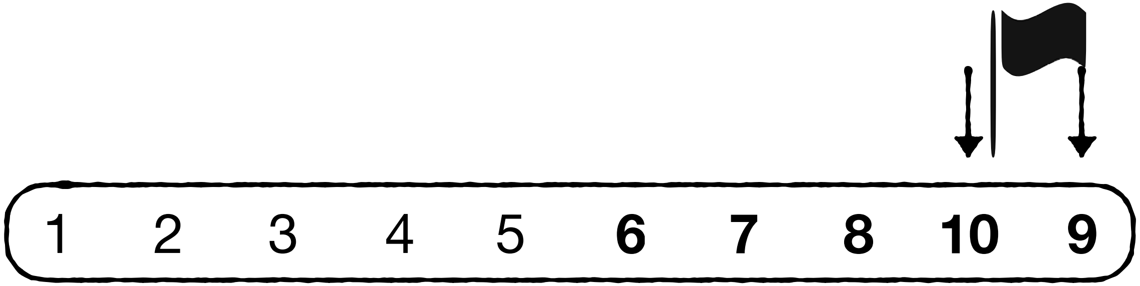
3. Swap pair of out-of-order entries and continue

4. Continue recursively on both partitions around pivot.



Quick-Sort

1. Pick random pivot point and initialize two pointers at ends
2. Move each pointer towards pivot, stopping at first out-of-order value respect to pivot
3. Swap pair of out-of-order entries and continue
4. Continue recursively on both partitions around pivot.



Quick-Sort

Properties: Expected $O(N \log_2 N)$ worst-case

But can be $O(N^2)$ with low probability

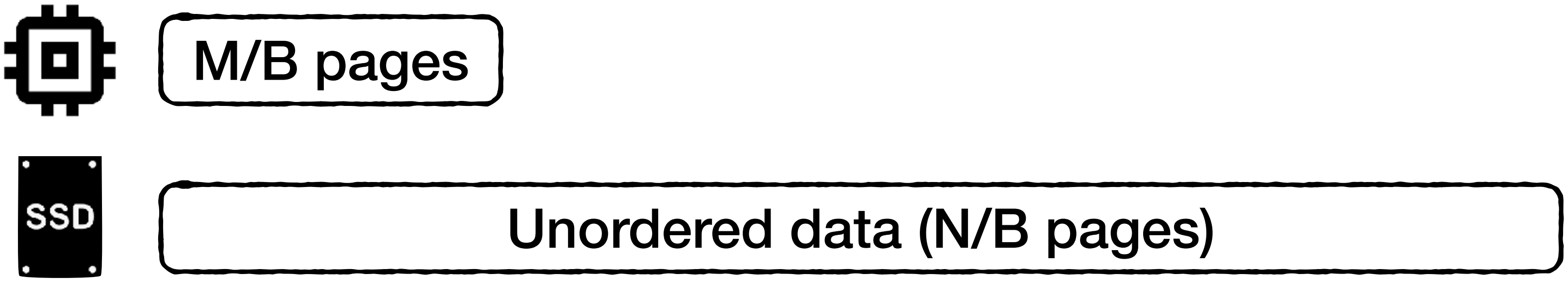
(e.g., always pick minimum key in each partition)

Sequential memory access is fast

In-place algorithm: no need for x2 space like merge-sort

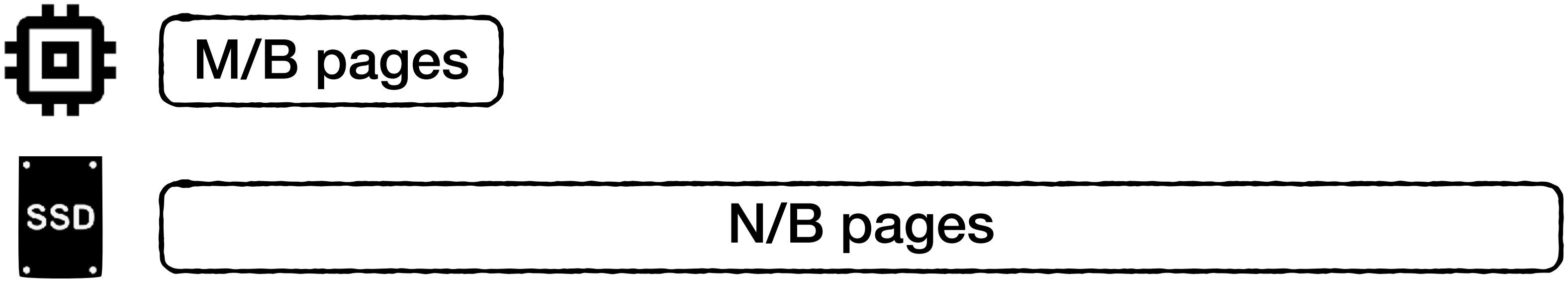
1 2 3 4 5 6 7 8 9 10

But what if data does not fit in memory?



Impact of More Memory

$$O(N/B \cdot \log_{M/B}(N/M))$$



Impact of More Memory

$$O(N/B \cdot \log_{M/B}(\boldsymbol{N/M}))$$

Fewer partitions to merge

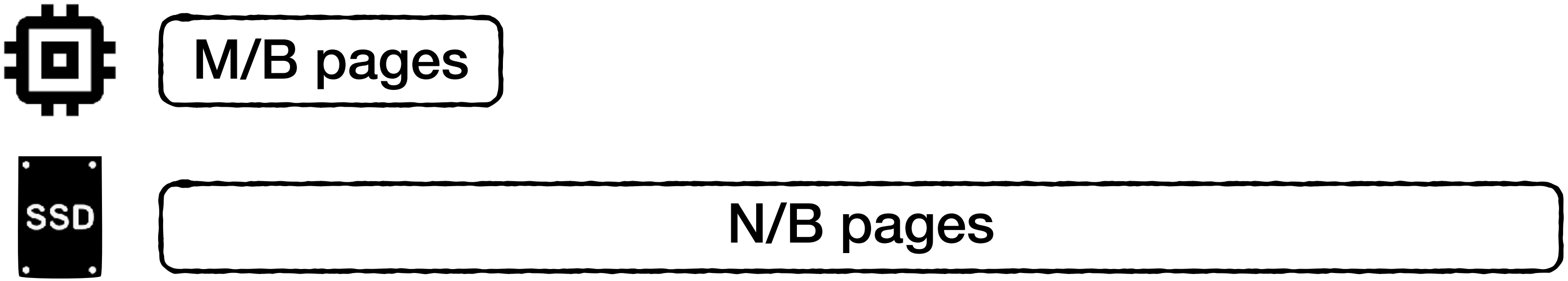
M/B pages

N/B pages

Impact of More Memory

$$O(N/B \cdot \log_{M/B}(N/M))$$

Merging more partitions per pass

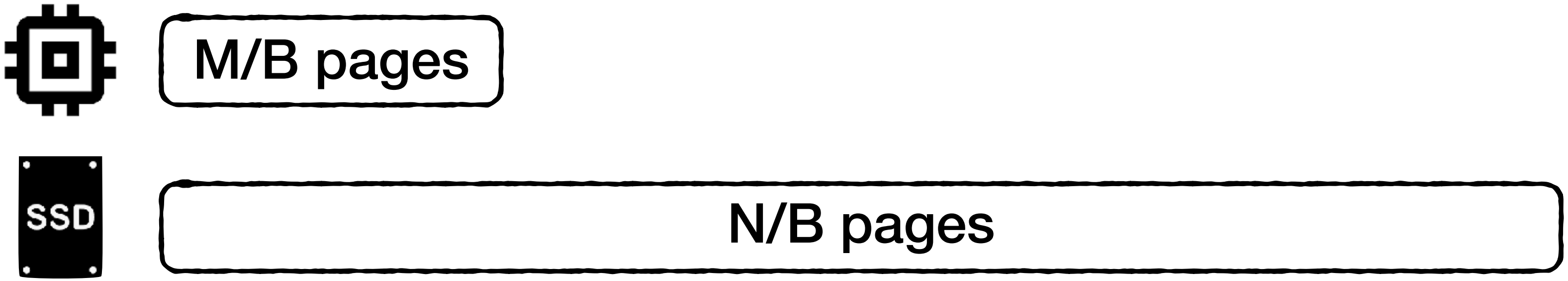


How much memory to merge all partitions in one pass?

Let **$\log_{M/B}(N/M) = 1$** and solve for M

We get: $M = \sqrt{N \cdot B}$ (Measured in entries)

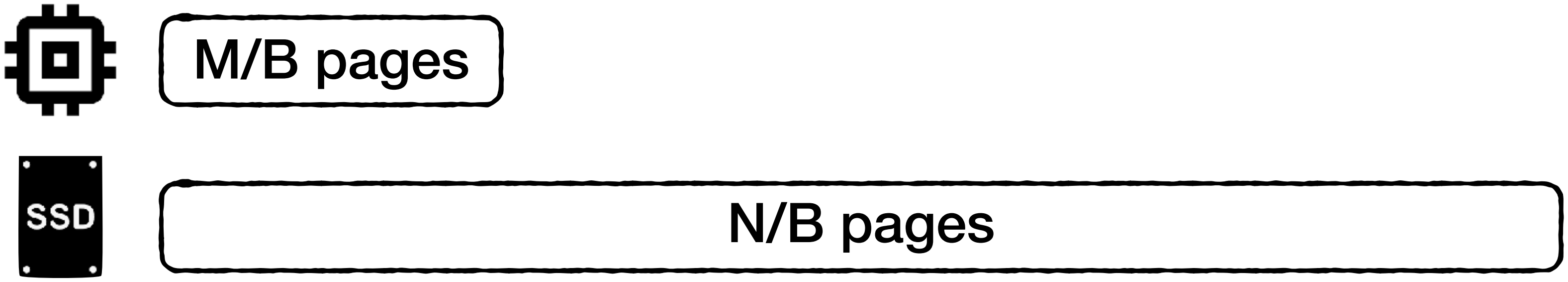
Hence, memory can accommodate $\sqrt{N/B}$ buffers



Two-Pass Merge Sort Algorithm

Use at least $M = \sqrt{N \cdot B}$ memory to partition the data.

This creates at most $N/M = \sqrt{N/B}$ sorted partitons



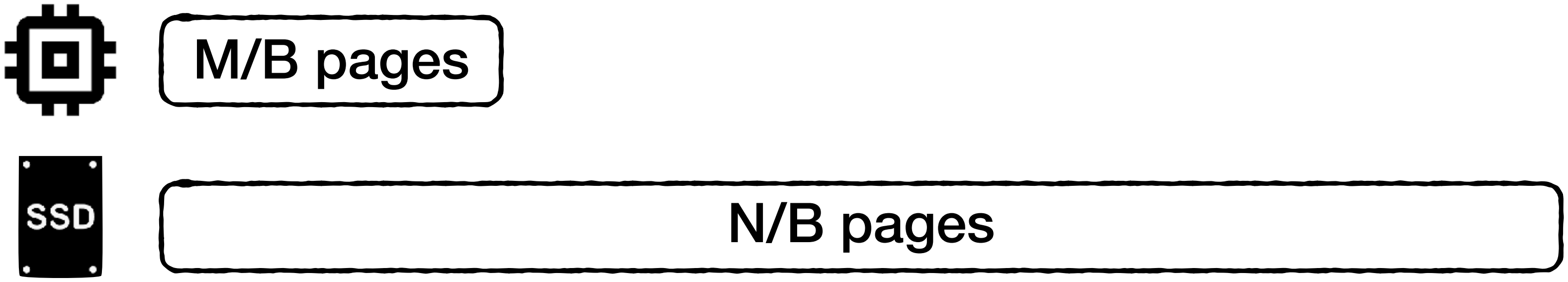
Two-Pass Merge Sort Algorithm

Use at least $M = \sqrt{N \cdot B}$ memory to partition the data.

This creates at most $N/M = \sqrt{N/B}$ sorted partitons

Then merge in one pass using at most $\sqrt{N/B}$ input buffers

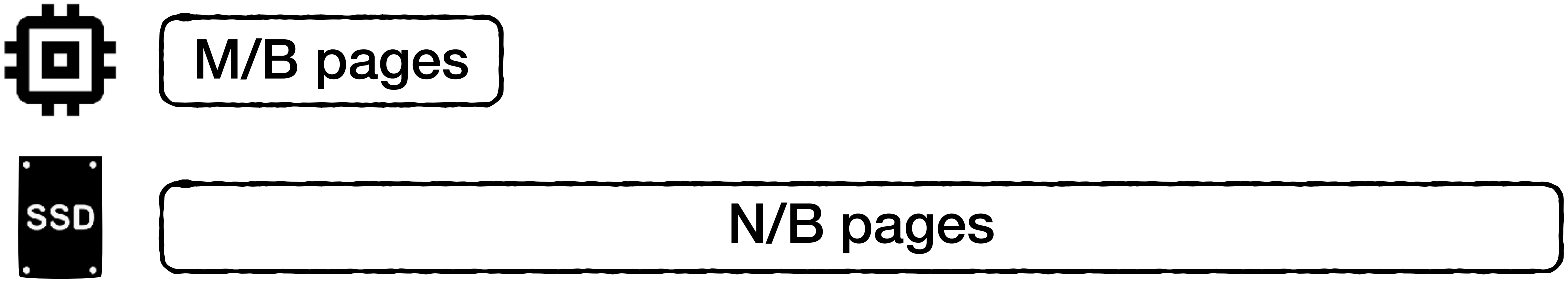
Cost: O(N/B)



How much memory do we need in practice?

Assume 1 TB, 16 byte entries, and 4KB pages
$N=2^{36}$ entries. And with 4KB pages, $B=2^{8}$ entries.
Hence, we need $M = \sqrt{N \cdot B} = \sqrt{2^{44}} = 2^{22}$ entries in memory or 64 MB

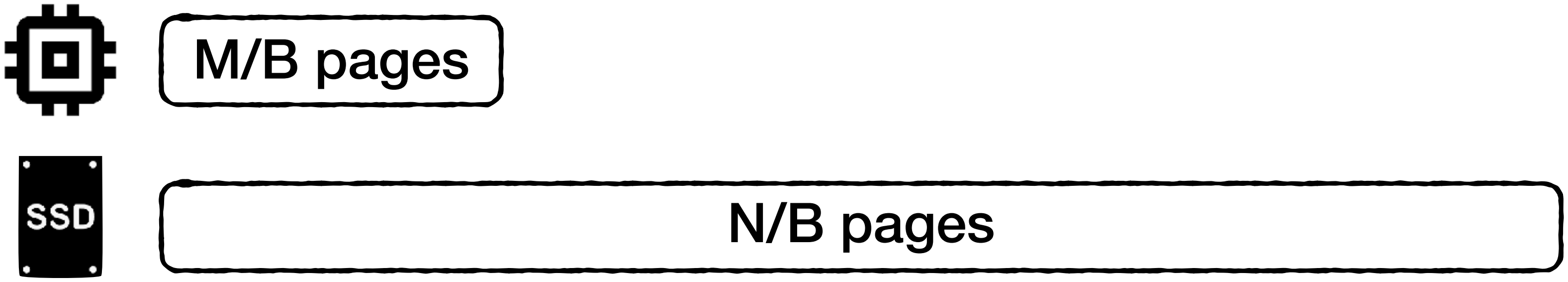
Hence, for all practical purposes, a 2 pass sorting algorithm is practical.



Achieved our goal of sorting using O(N/B) I/Os using little memory :)

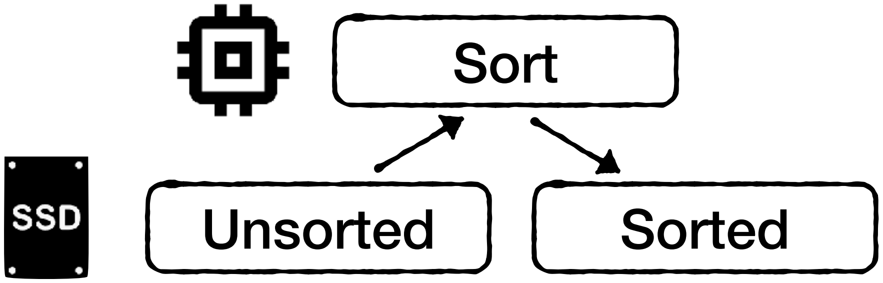
But how about CPU overheads?

We still expect $O(N \log_2 N)$. Do we achieve it?

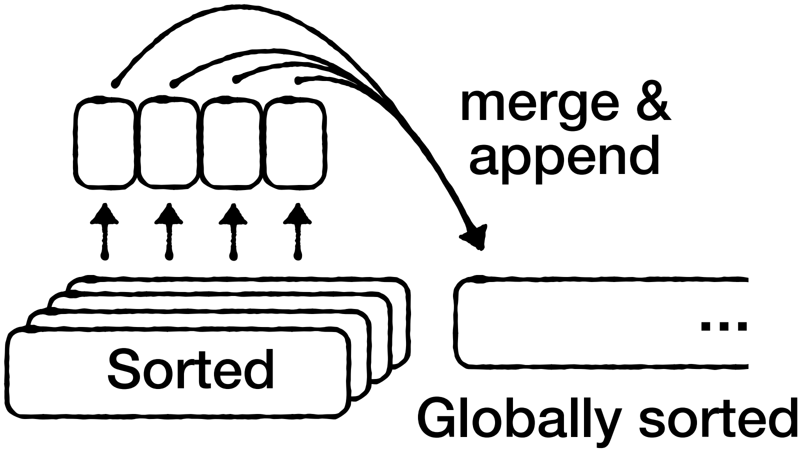


Analyzing CPU

Partitioning Phase



Merging Phase

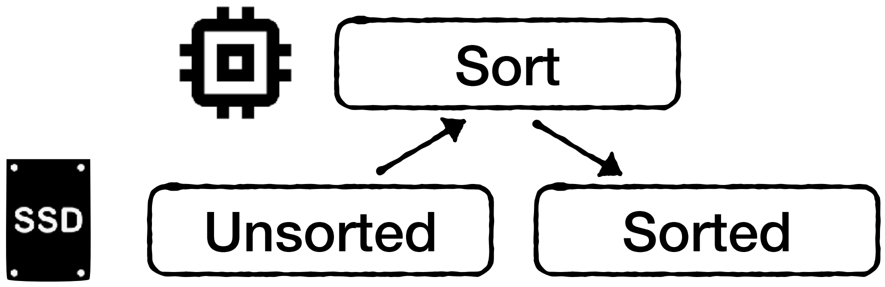


Partitioning Phase

Each chunk contains M entries

Need $O(M \log_2 M)$ CPU cycles to merge-sort it in-memory

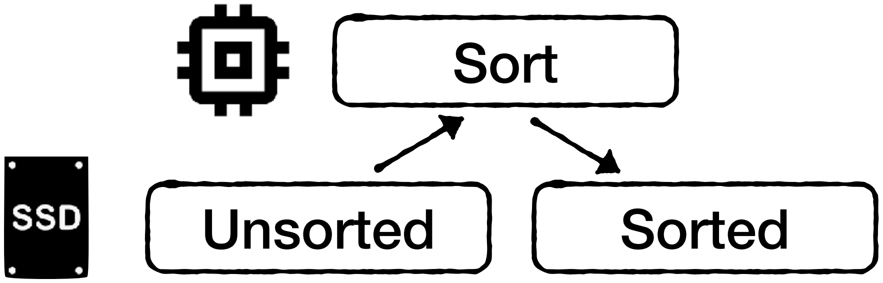
Doing this for all N/M chunk takes $O(N \log_2 M)$ CPU



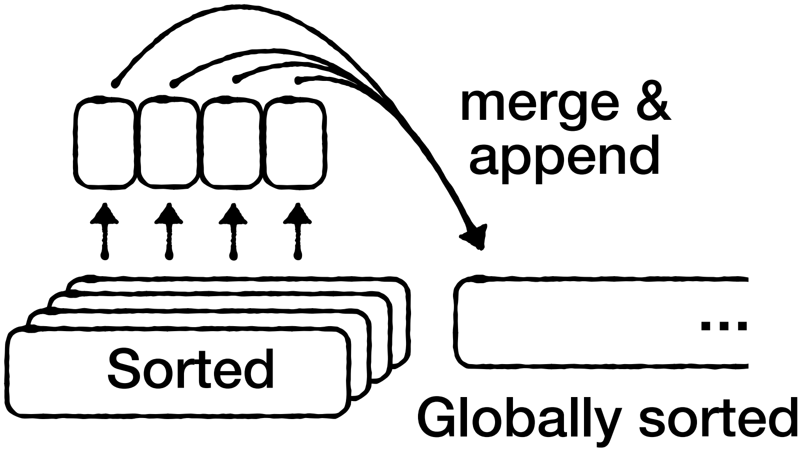
Analyzing CPU

Partitioning Phase

O(N $\log_2$ M)

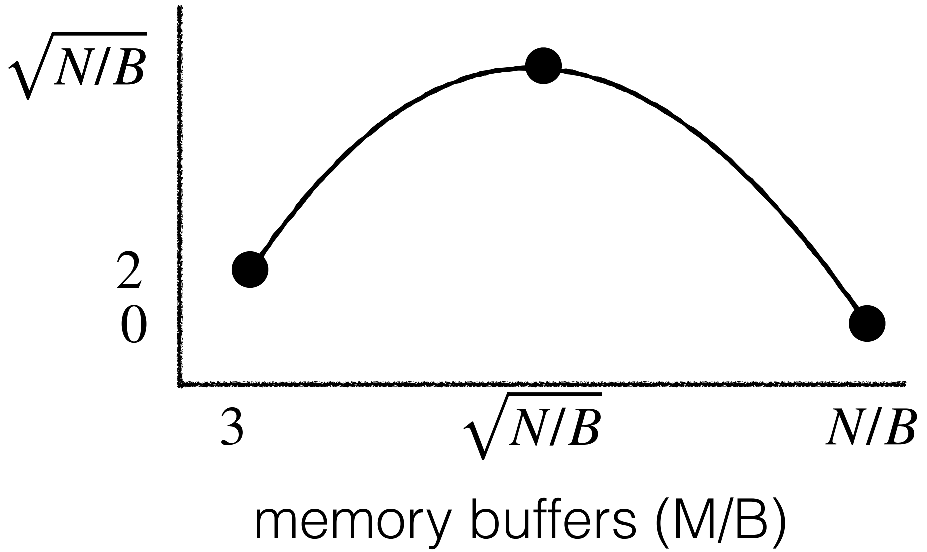


Merging Phase



Merging Phase

maximum # partitions to merge in one go?

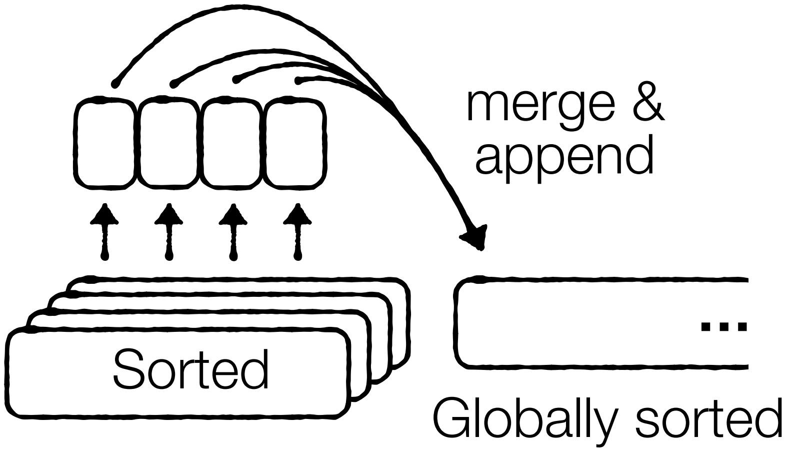


Merging Phase

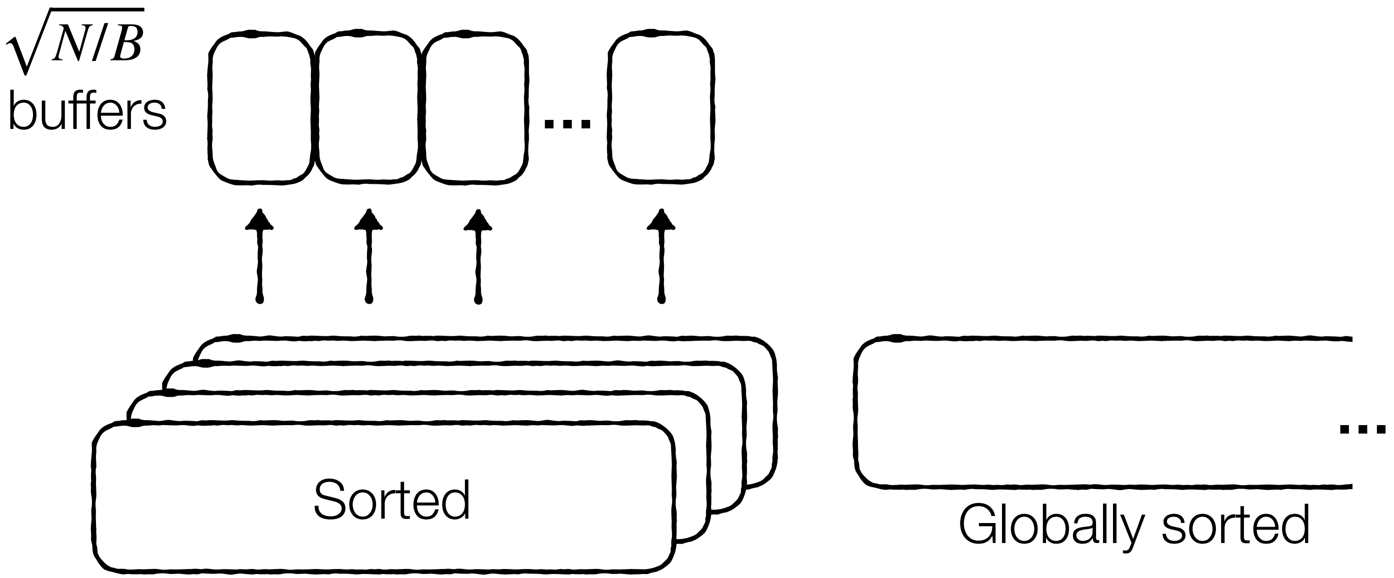
maximum # partitions to
merge in one go?

$$\sqrt{N/B}$$

How to merge partitions?



How to merge partitions?



Binary Min-Heap

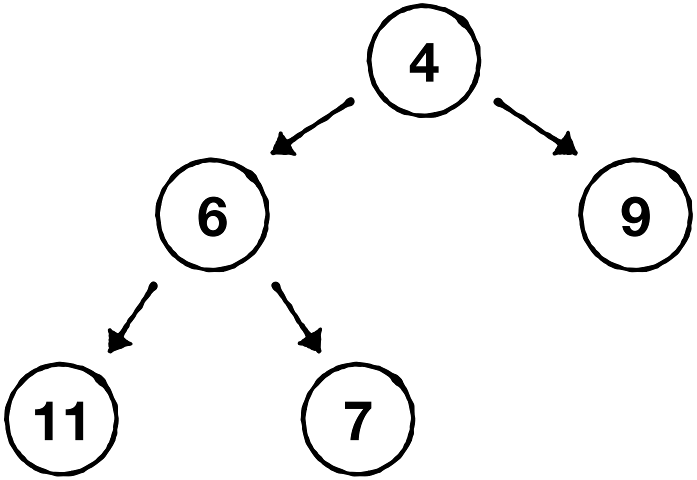
Well-known data structure that efficiently extracts the minimum value in a collection of data items

API	***Runtime***
Insert(key)	$O(\log_2 N)$
Key = extract_min()	$O(\log_2 N)$
min_key = insert_and_extract(new_key) (efficiently combines both operations)	$O(\log_2 N)$

Binary Min-Heap

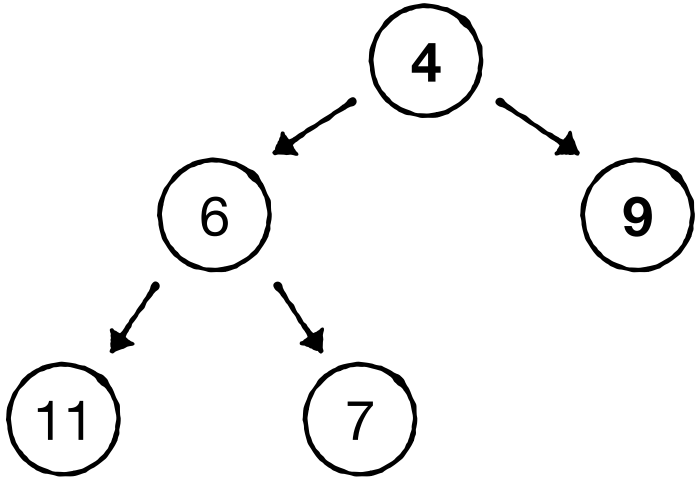
(1) **Complete binary tree**
(All levels are full & largest level is full from left to right)

(2) Parent key always smaller than children'.



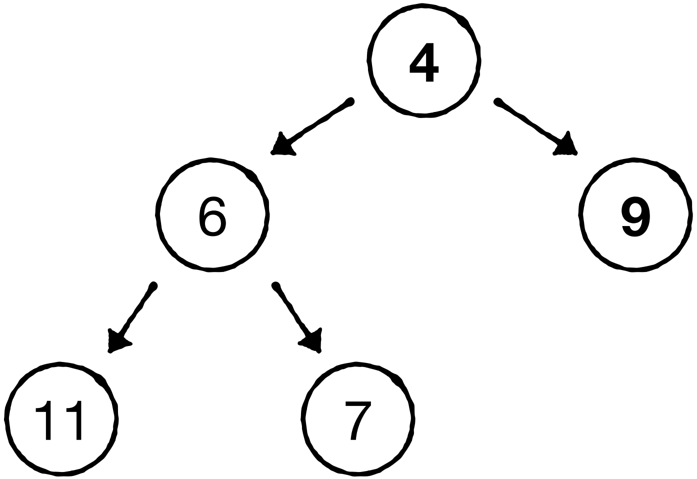
Binary Min-Heap Extract Minimum

$O(\log_2 N)$ min extraction cost

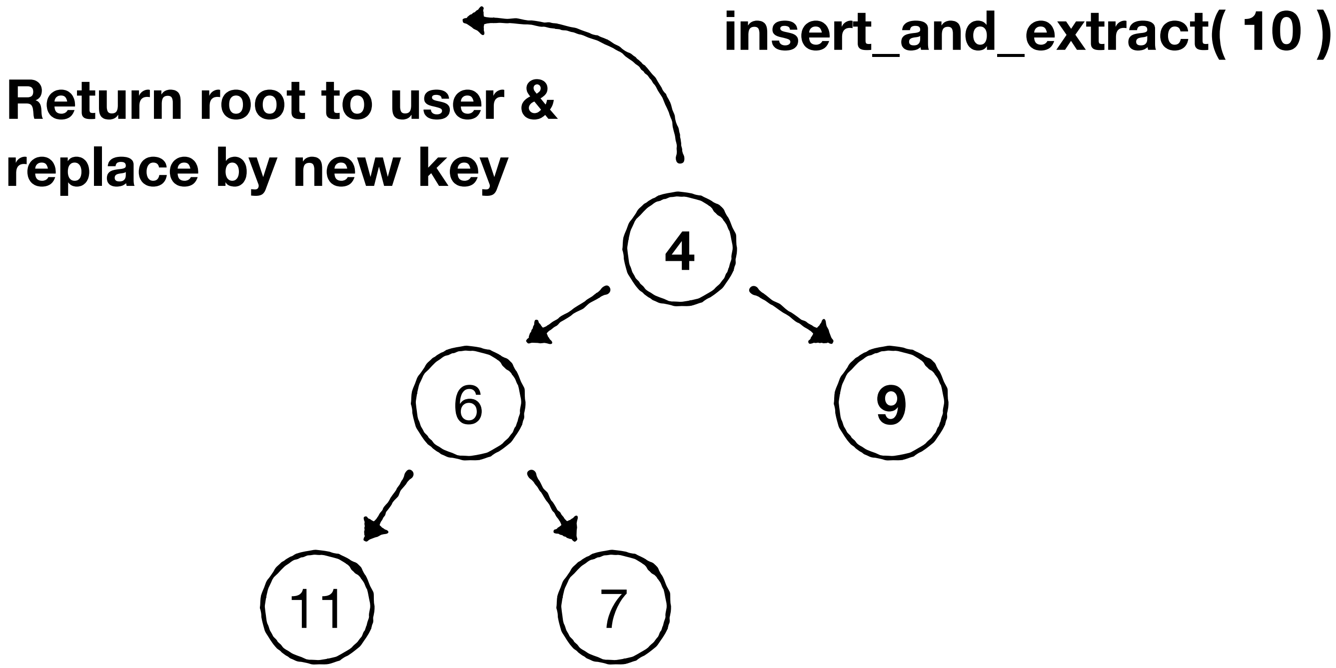


Binary Min-Heap Insert & extract

min_key = insert_and_extract(new_key)

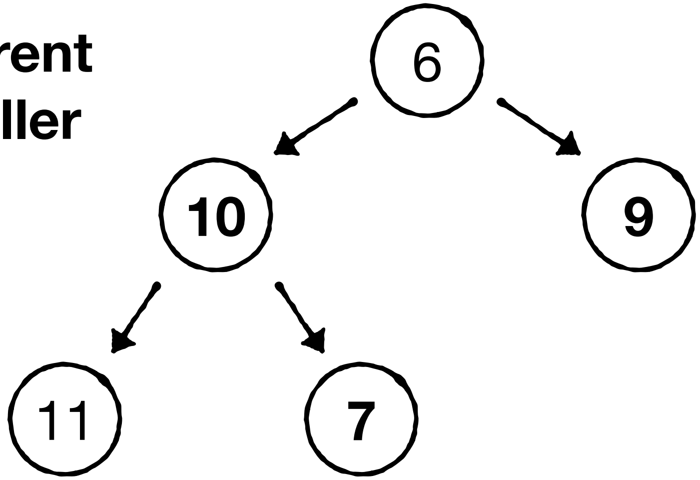


Binary Min-Heap Insert & extract



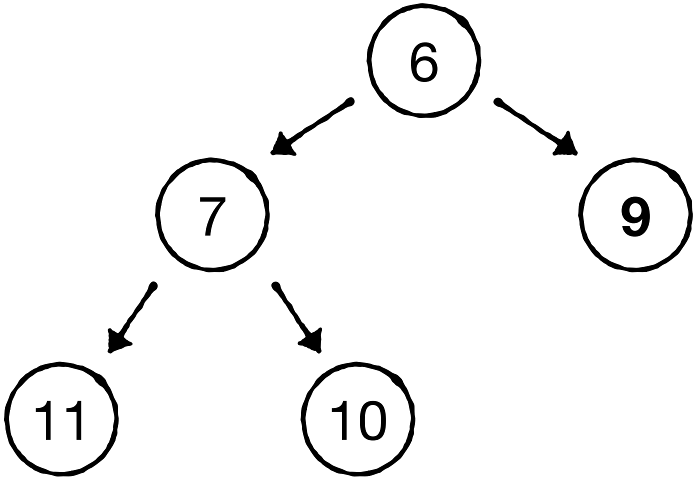
Binary Min-Heap Insert & extract

Swap until parent is always smaller



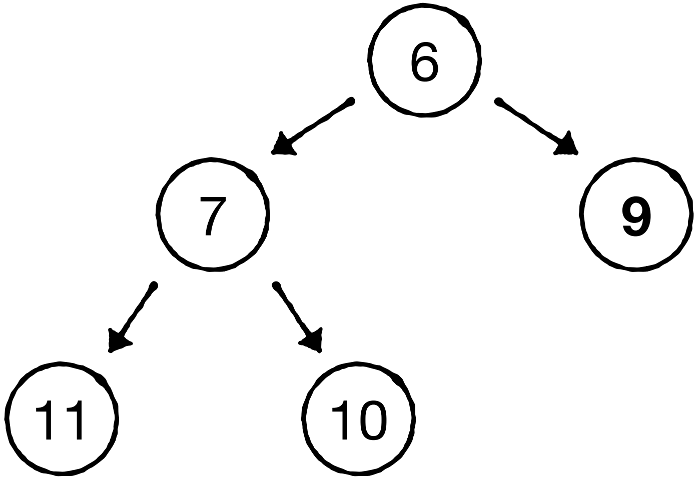
Binary Min-Heap Insert & extract

O($\log_2$ N) for insert_and_extract

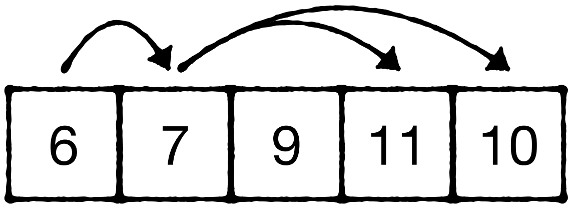


Binary Min-Heap Implementation

Tree with Pointers



Array



Compact since the binary tree is complete. Avoids overhead of pointers.

Binary Min-Heap Construction

We can construct a heap for N entries using normal insertions

takes $O(N \log_2 N)$

Unordered N keys → Min-Heap

Binary Min-Heap Efficient Array Construction

6	8	2	1	5	4	3

Start with Unordered N keys

Binary Min-Heap Efficient Array Construction

1	**5**	2	**8**	**6**	4	3

$\mathbf{O(N)} < \mathbf{O(N \log_2 N)}$ from before with pure insertions

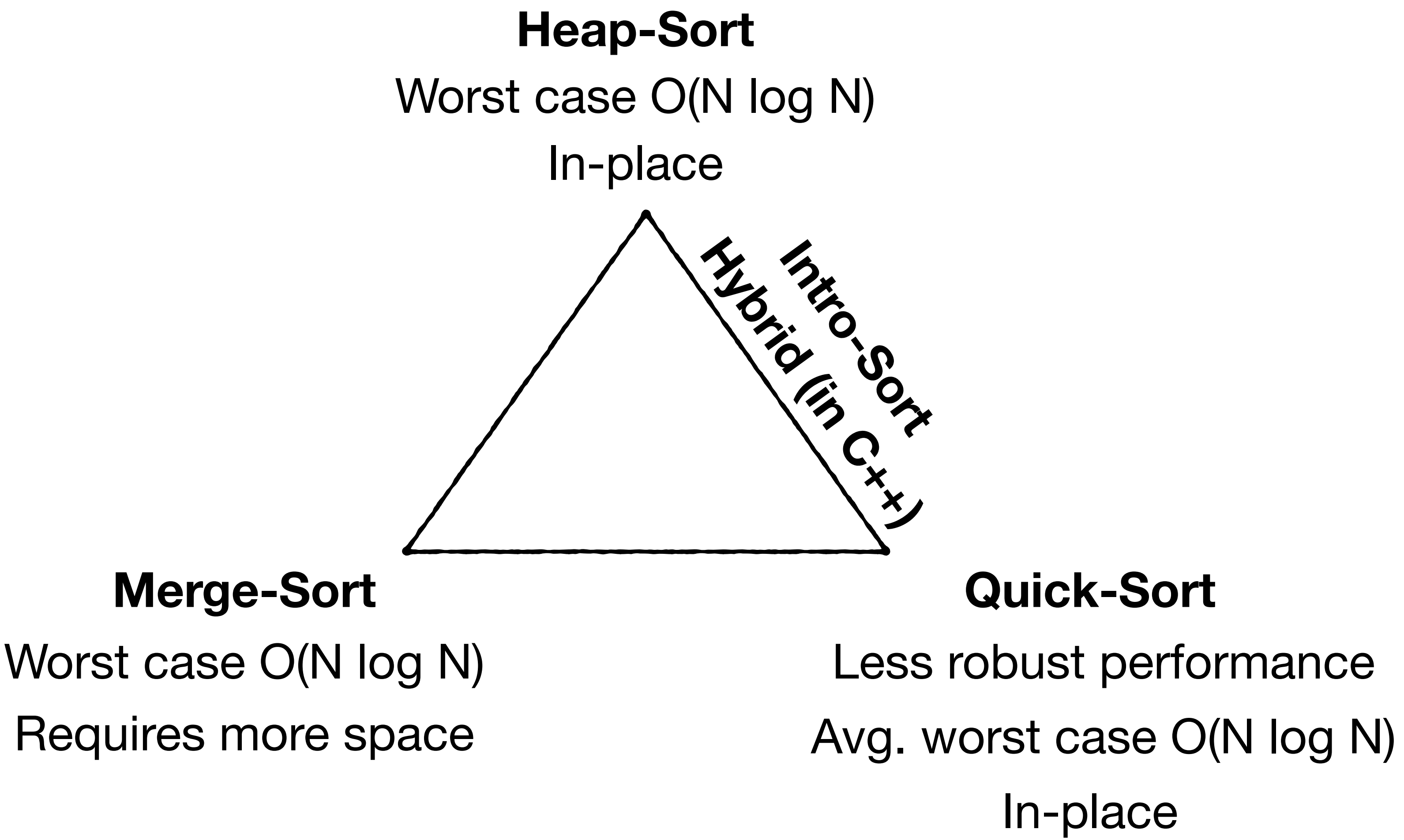
Side-Note on Heap-Sort

If data fits in memory, we can sort it using a heap

Construct heap in-place in O(N)

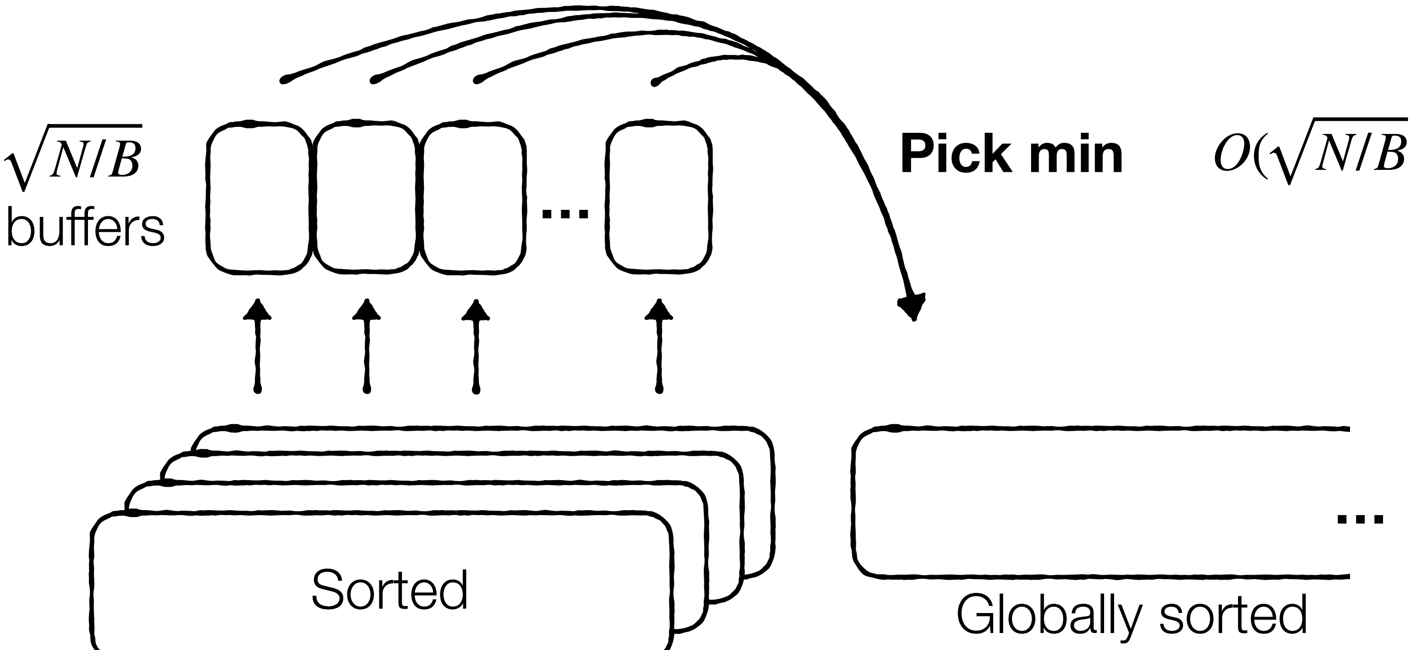
N keys

extract N times until heap is empty in O(N $\log_2$ N)

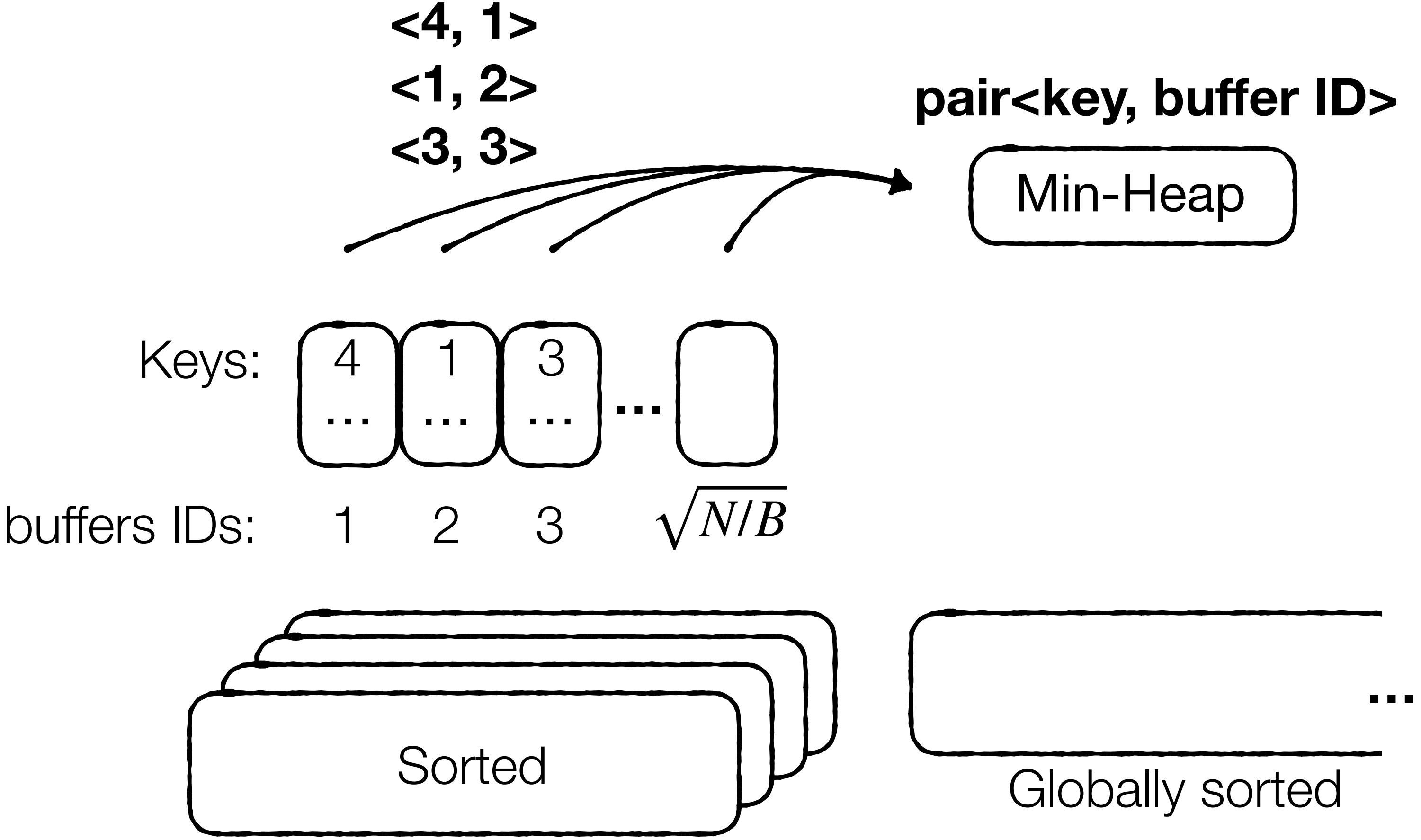

Heap-Sort
Worst case O(N log N)
In-place
Intro-Sort
Hybrid (in C++)
Merge-Sort
Worst case O(N log N)
Requires more space
Quick-Sort
Less robust performance
Avg. worst case O(N log N)
In-place

Now back to how to merge partitions in external merge-sort

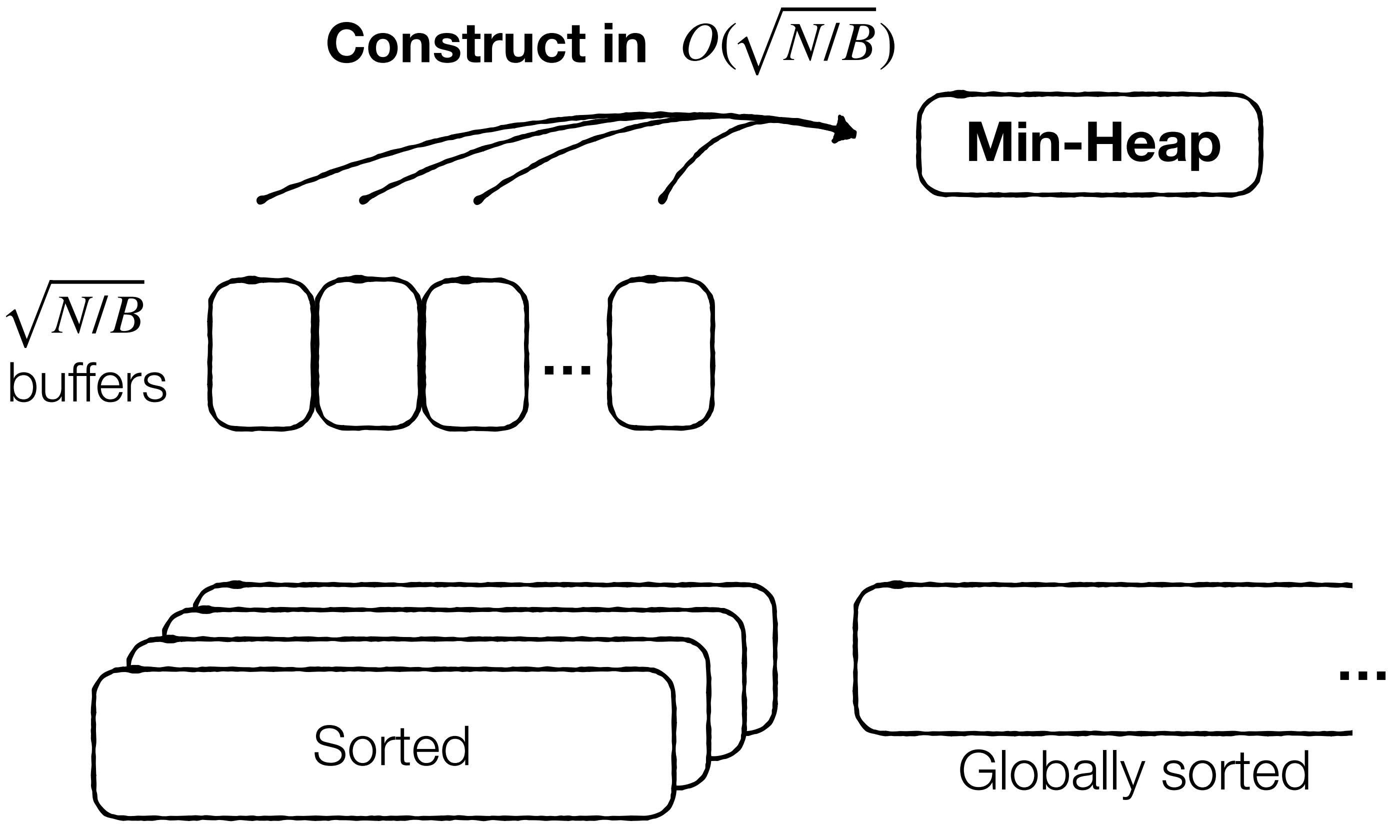
How to merge partitions?

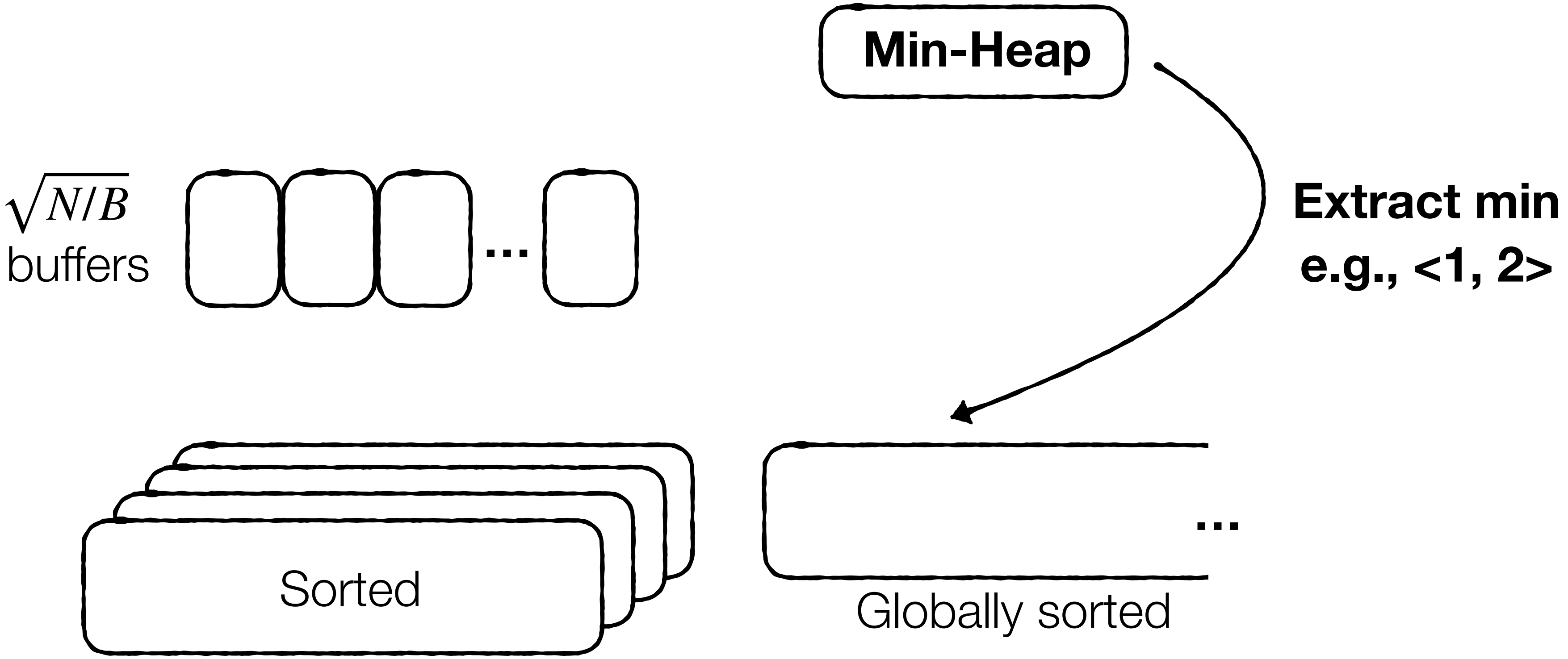


$O(\sqrt{N/B} \cdot N)$ **comparisons**

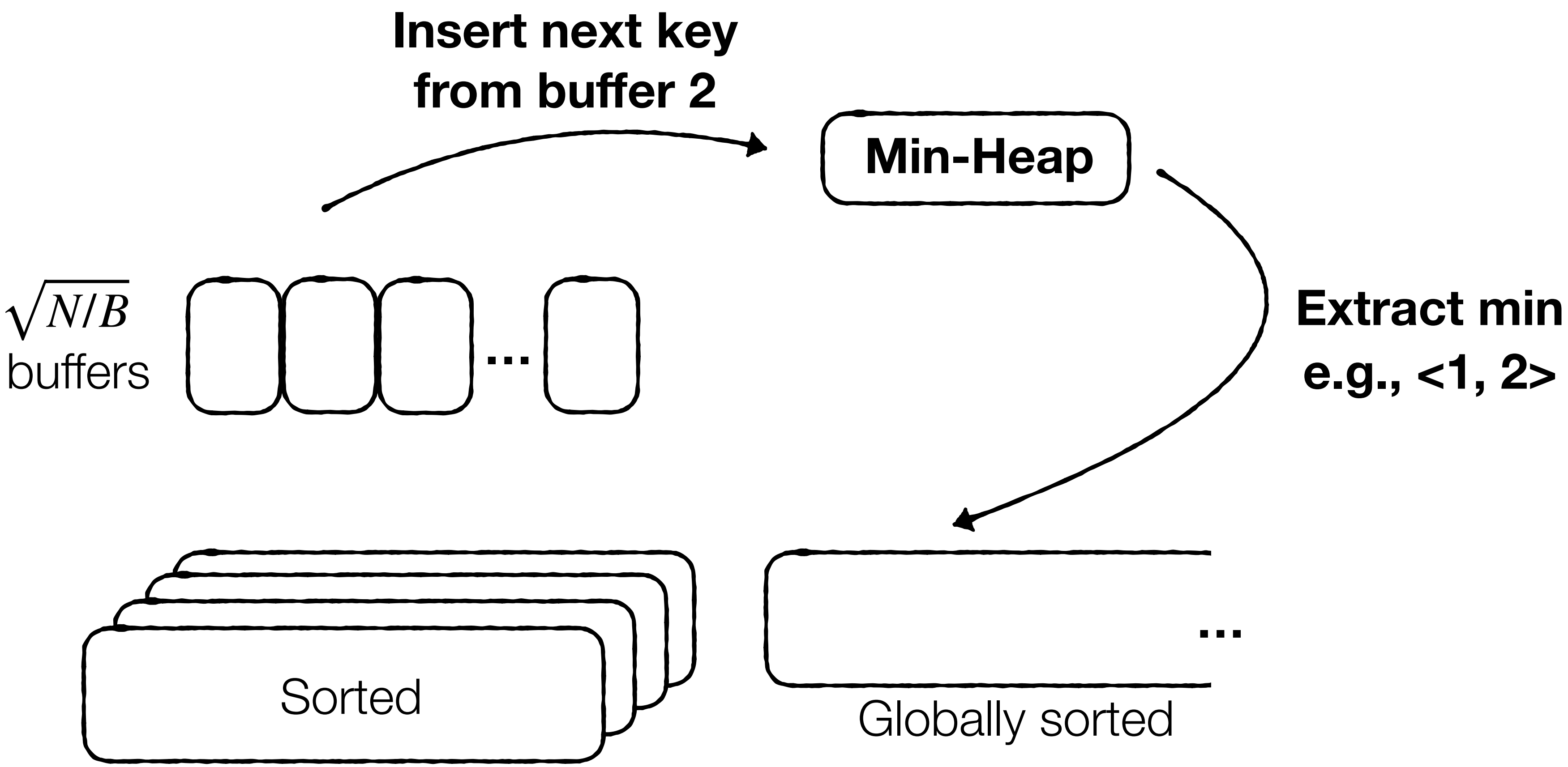


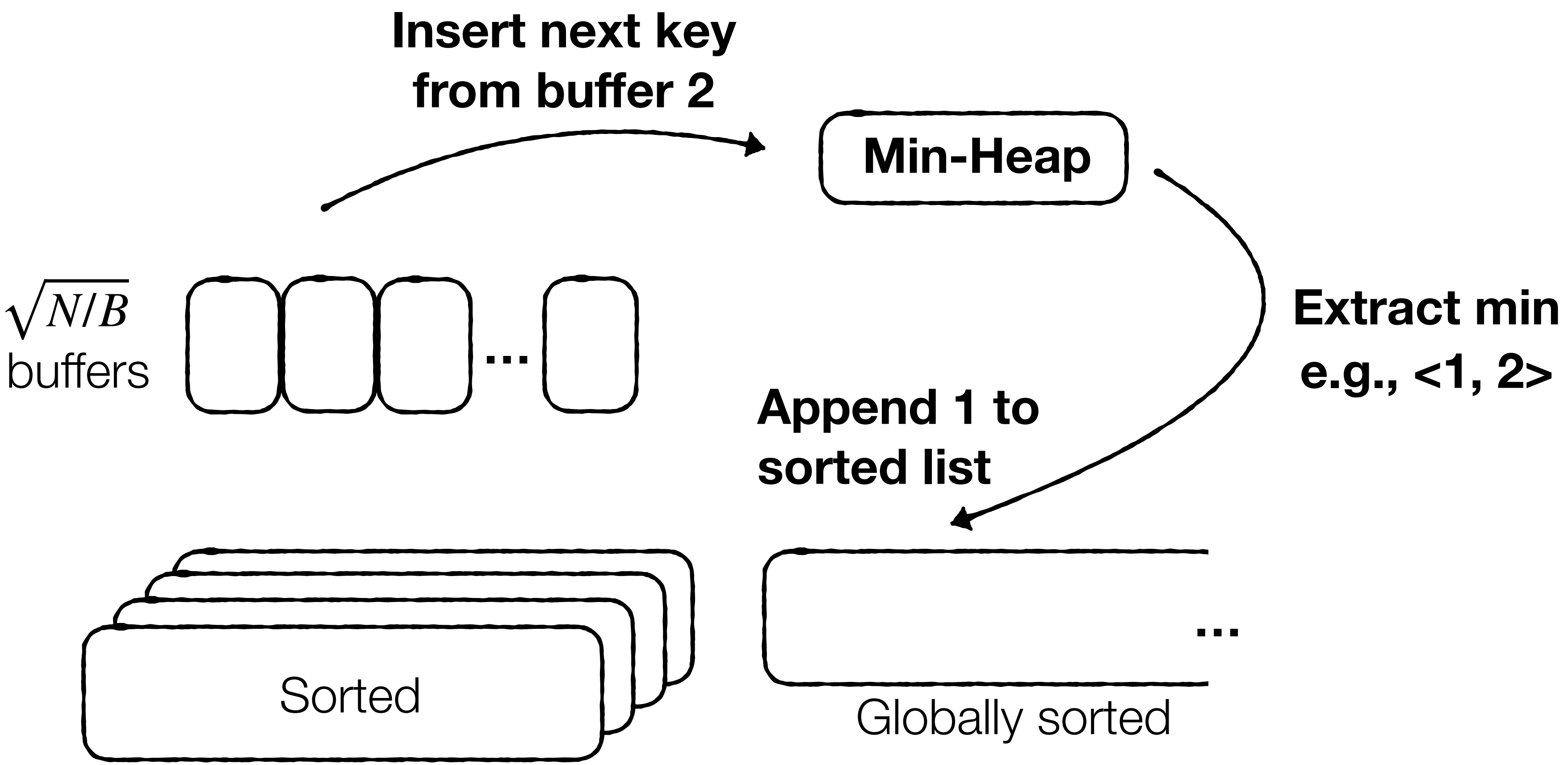
<4, 1>
<1, 2>
<3, 3>
pair<key, buffer ID>
Min-Heap
Keys:
4
1
3
...
buffers IDs:
1
2
3
$\sqrt{N/B}$
Sorted
...
Globally sorted


Construct in $O(\sqrt{N/B})$
Min-Heap
$\sqrt{N/B}$ buffers
...
Sorted
...
Globally sorted



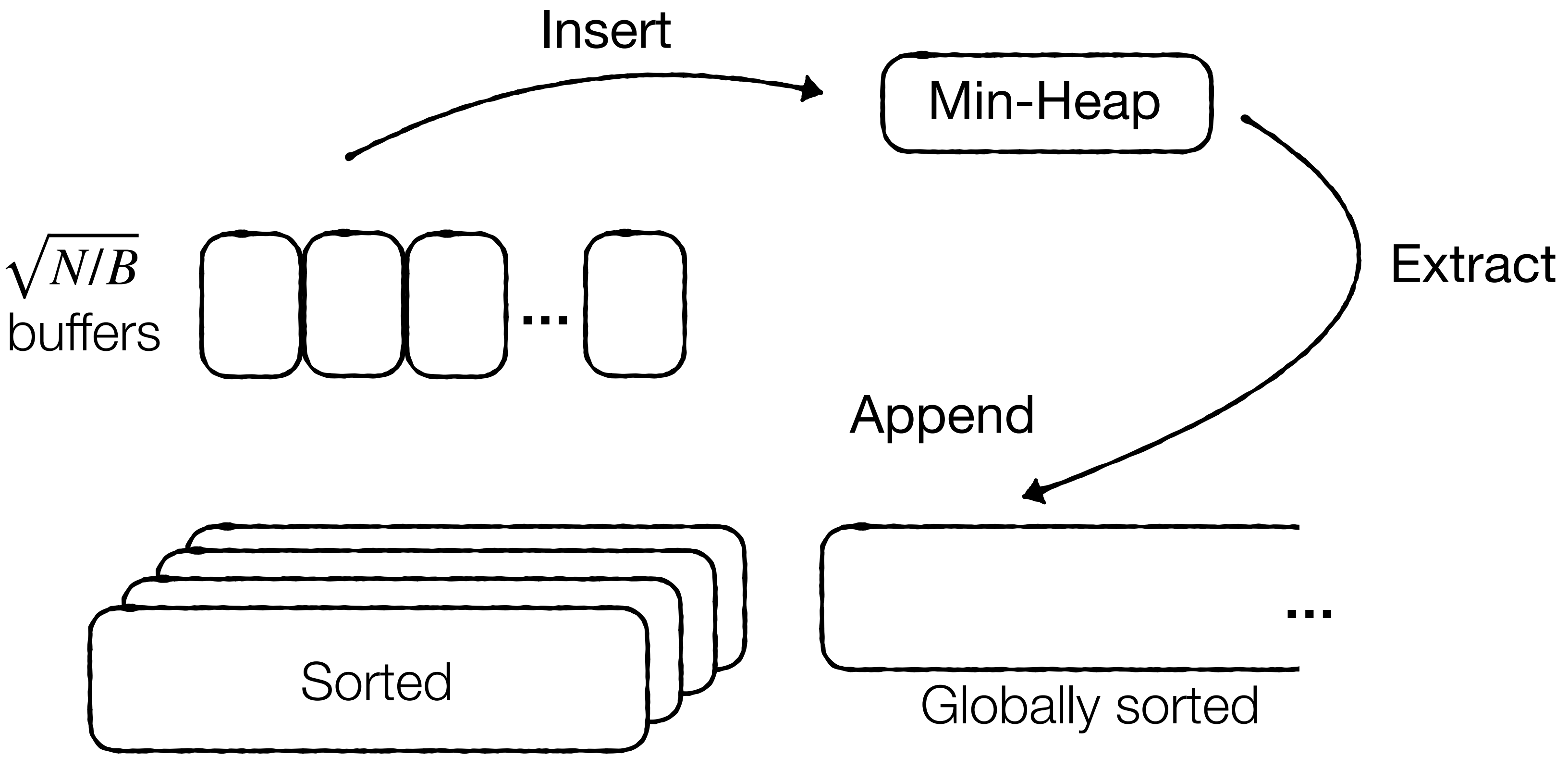
Min-Heap
Extract min
e.g., <1, 2>
$\sqrt{N/B}$ buffers
...
Sorted
...
Globally sorted


Insert next key
from buffer 2
Min-Heap
$\sqrt{N/B}$
buffers
...
Extract min
e.g., <1, 2>
Sorted
...
Globally sorted

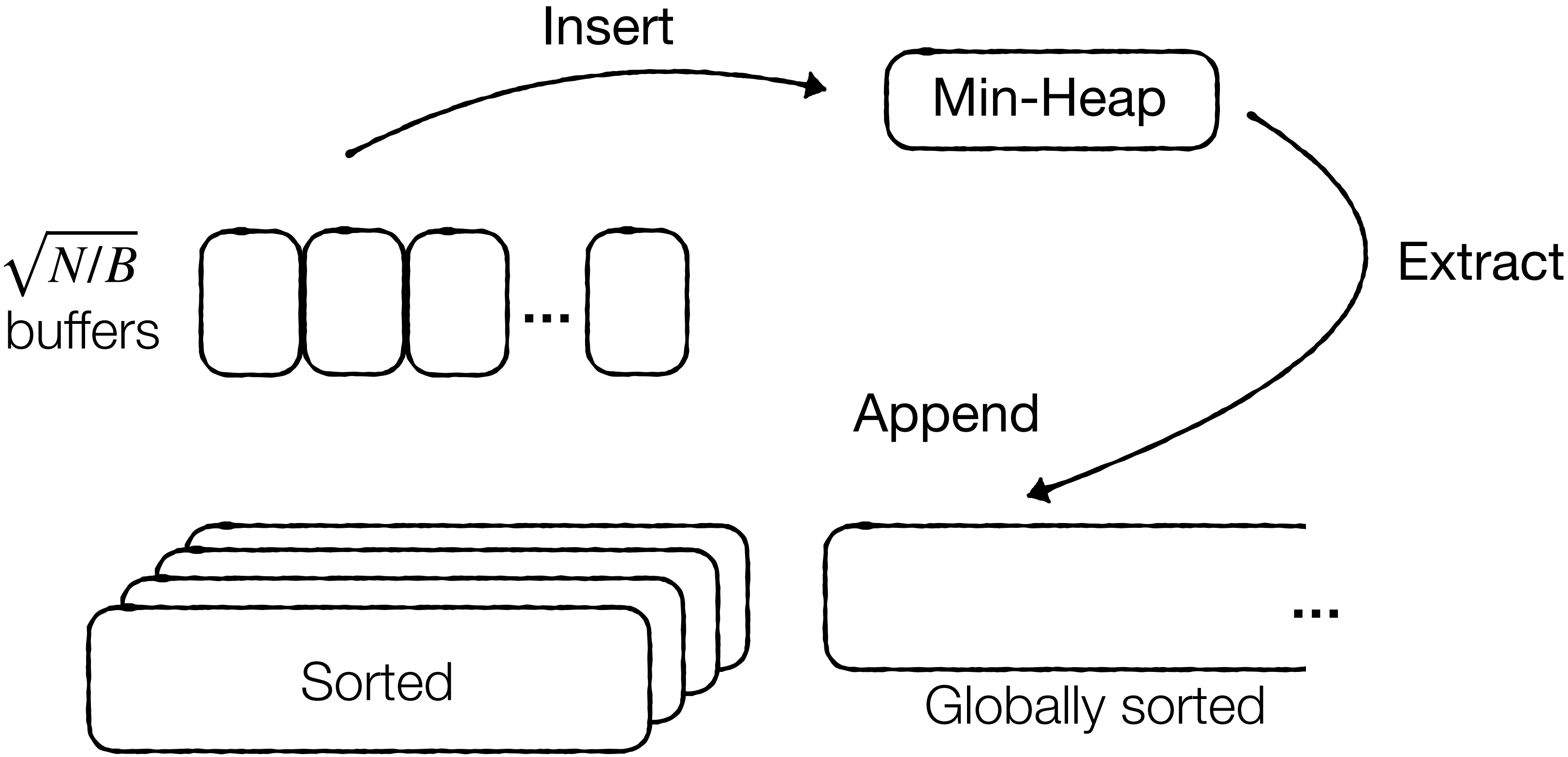


Insert next key
from buffer 2
Min-Heap
$\sqrt{N/B}$
buffers
...
Extract min
e.g., <1, 2>
Append 1 to
sorted list
Sorted
...
Globally sorted

can do this with insert_and_extract

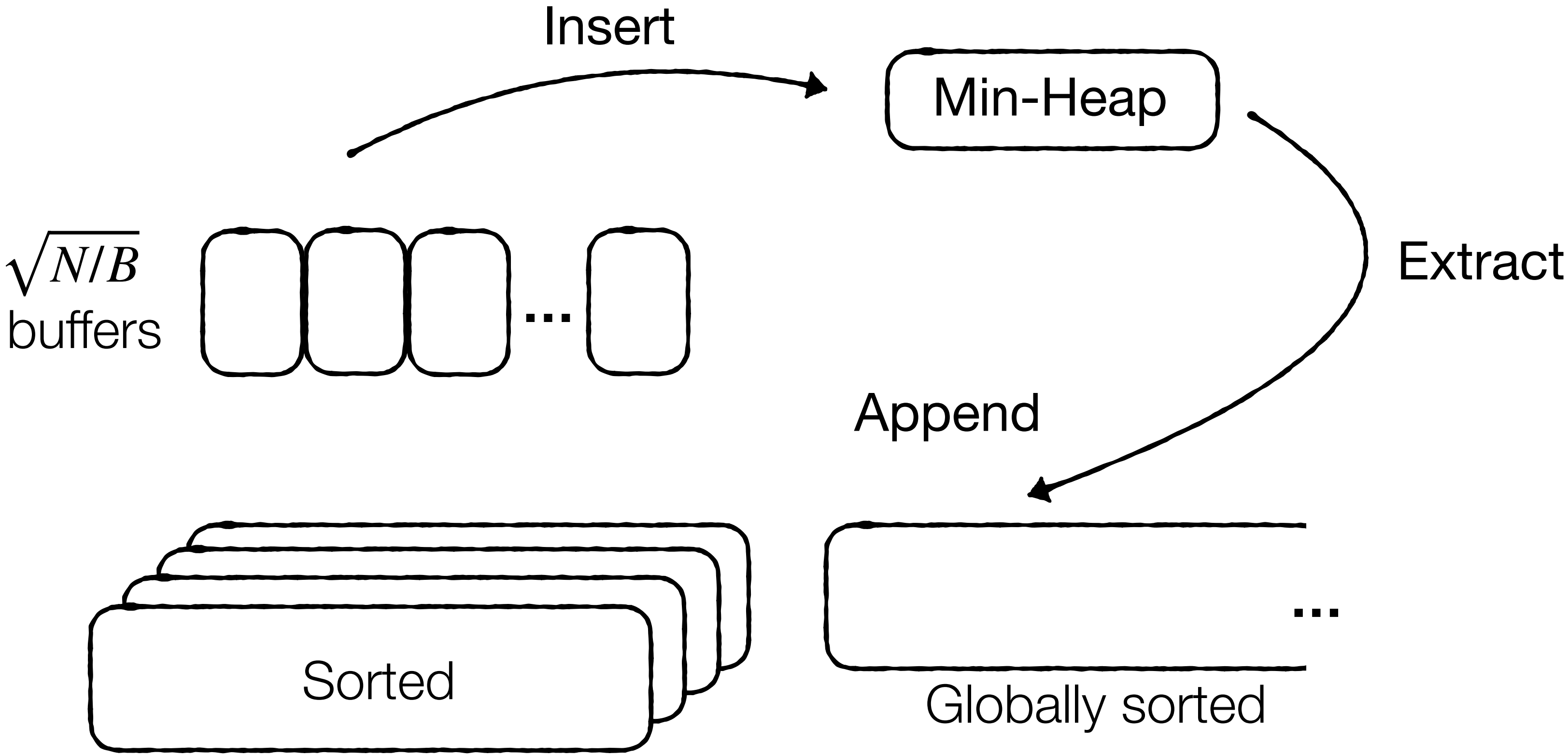


can do this with insert_and_extract: $O(log_2\sqrt{N/B})$ **per entry**



can do this with insert_and_extract: $O(log_2\sqrt{N/B})$

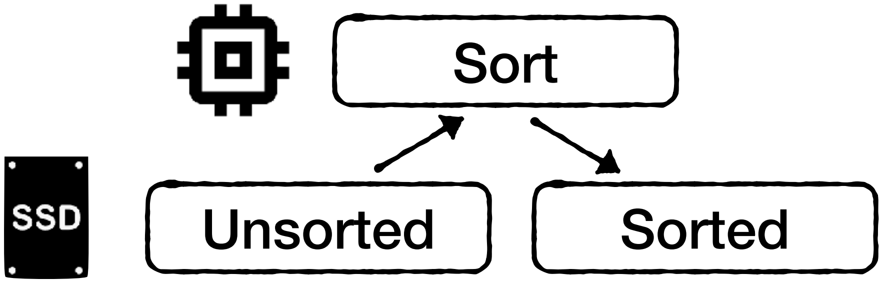
$O(N \cdot log_2\sqrt{N/B})$ **overall**



Analyzing CPU

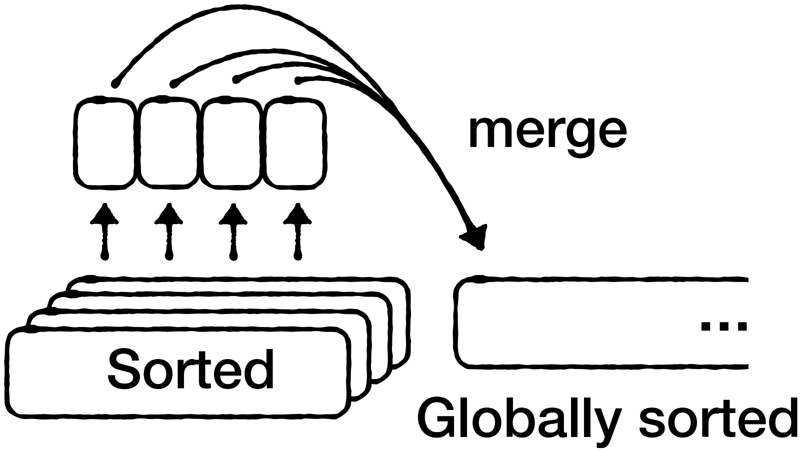
Partitioning Phase

$$O(N \cdot log_2 M)$$



Merging Phase

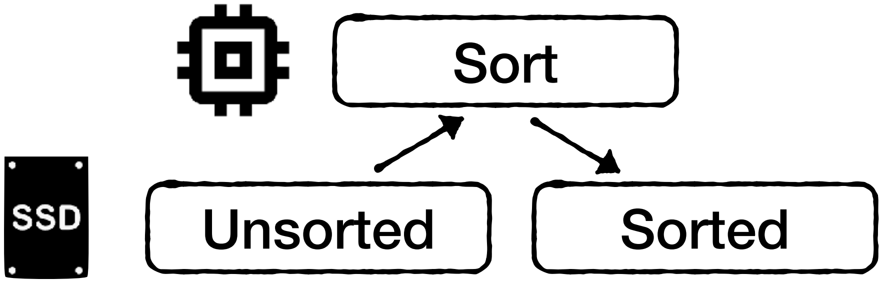
$$O(N \cdot log_2 \sqrt{N/B})$$



Analyzing CPU

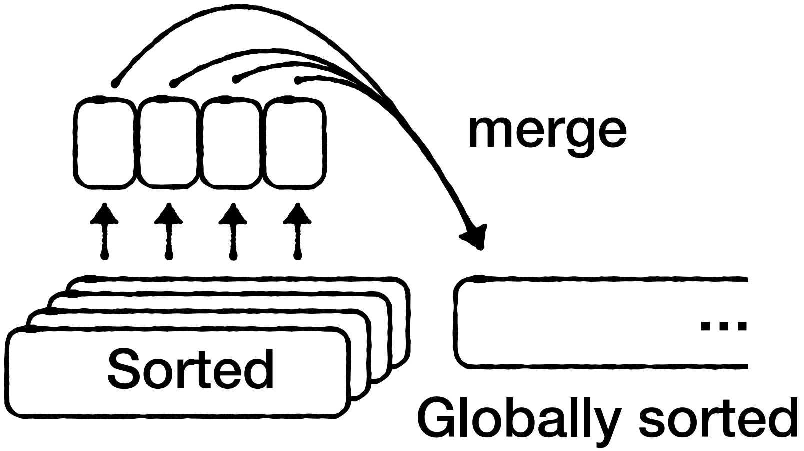
Partitioning Phase

$$O(N \cdot log_2 M)$$
$$= O(N \cdot log_2 \sqrt{N \cdot B})$$



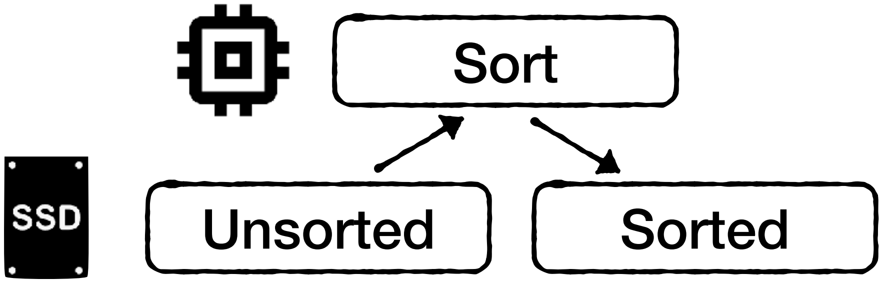
Merging Phase

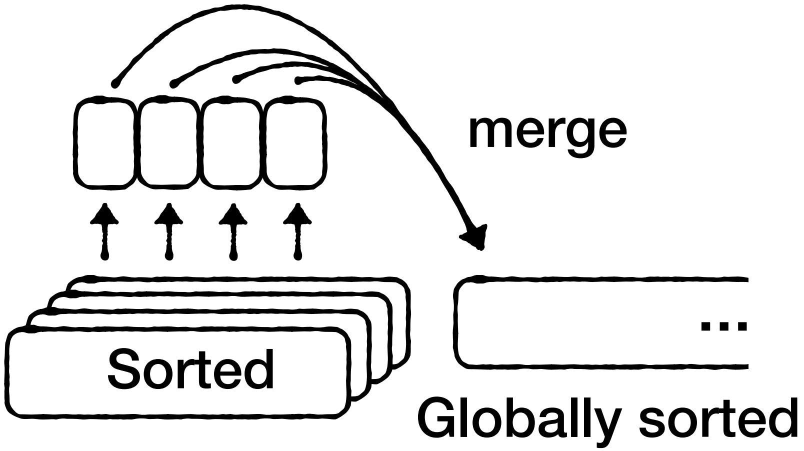
$$O(N \cdot log_2 \sqrt{N/B})$$



Analyzing CPU

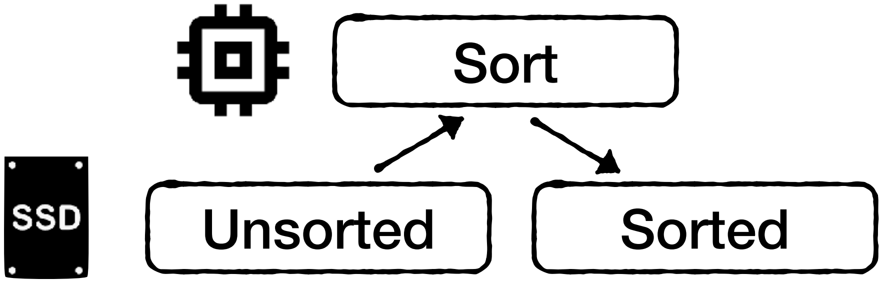
Partitioning Phase		Merging Phase		**Total cost**
$O(N \cdot log_2\sqrt{N \cdot B})$	+	$O(N \cdot log_2\sqrt{N/B})$	+	$O(N \cdot log_2 N)$

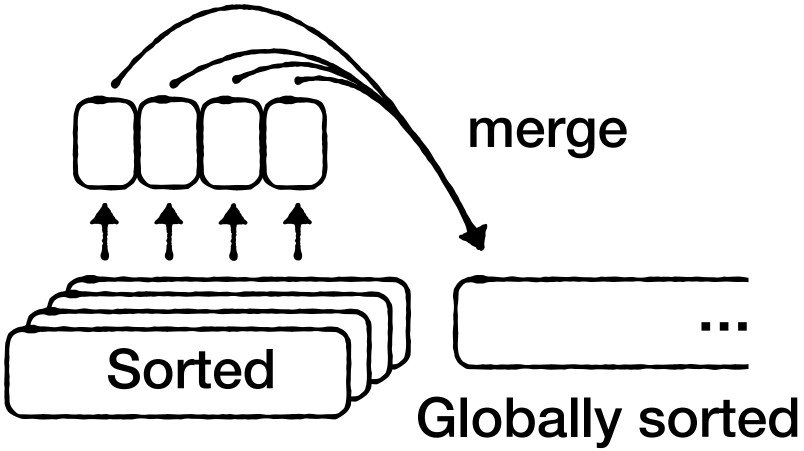




Same as in-memory merge-sort :)

$$O(N \cdot log_2 N)$$

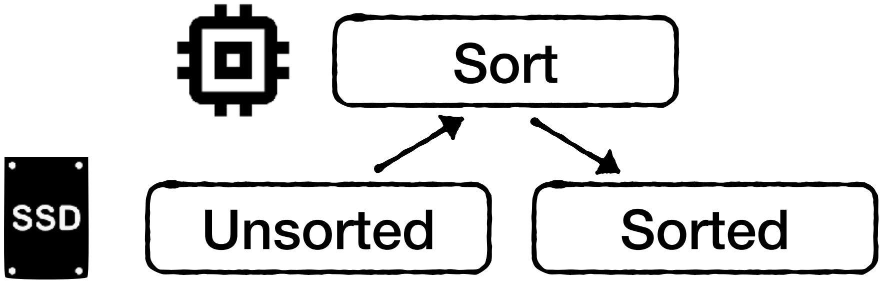


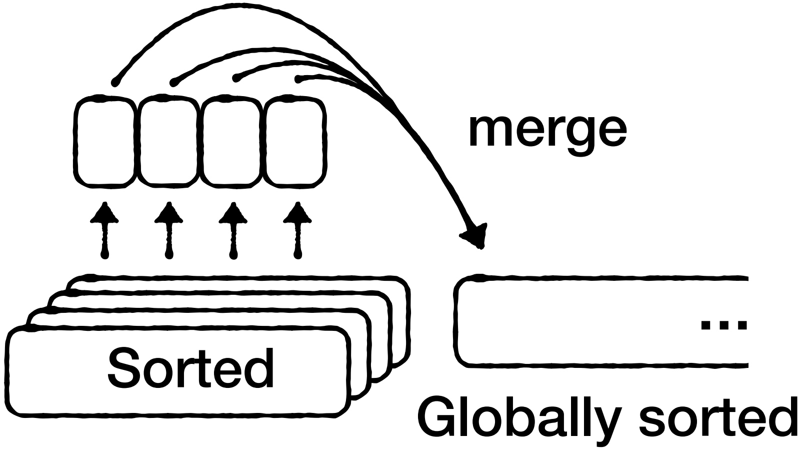


Overall costs

$O(N \cdot log_2 N)$ **CPU**

$O(N/B \cdot log_{M/B}(N/M))$ **I/O**

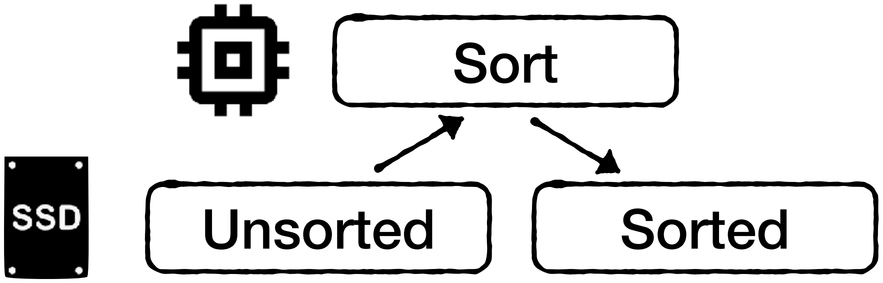


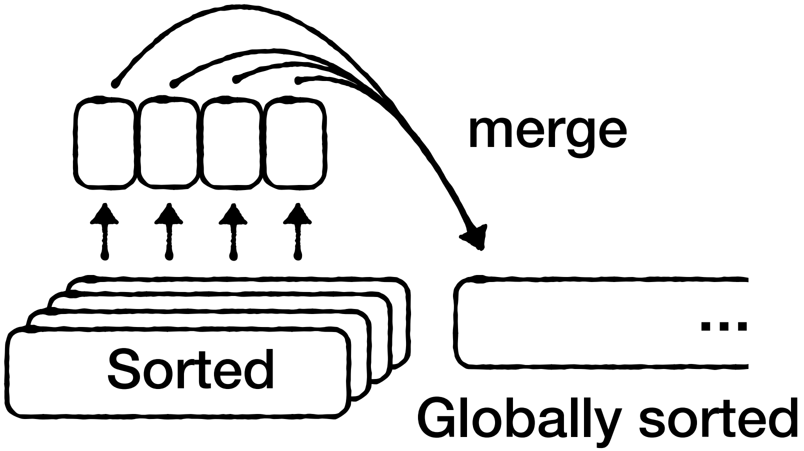


Overall costs

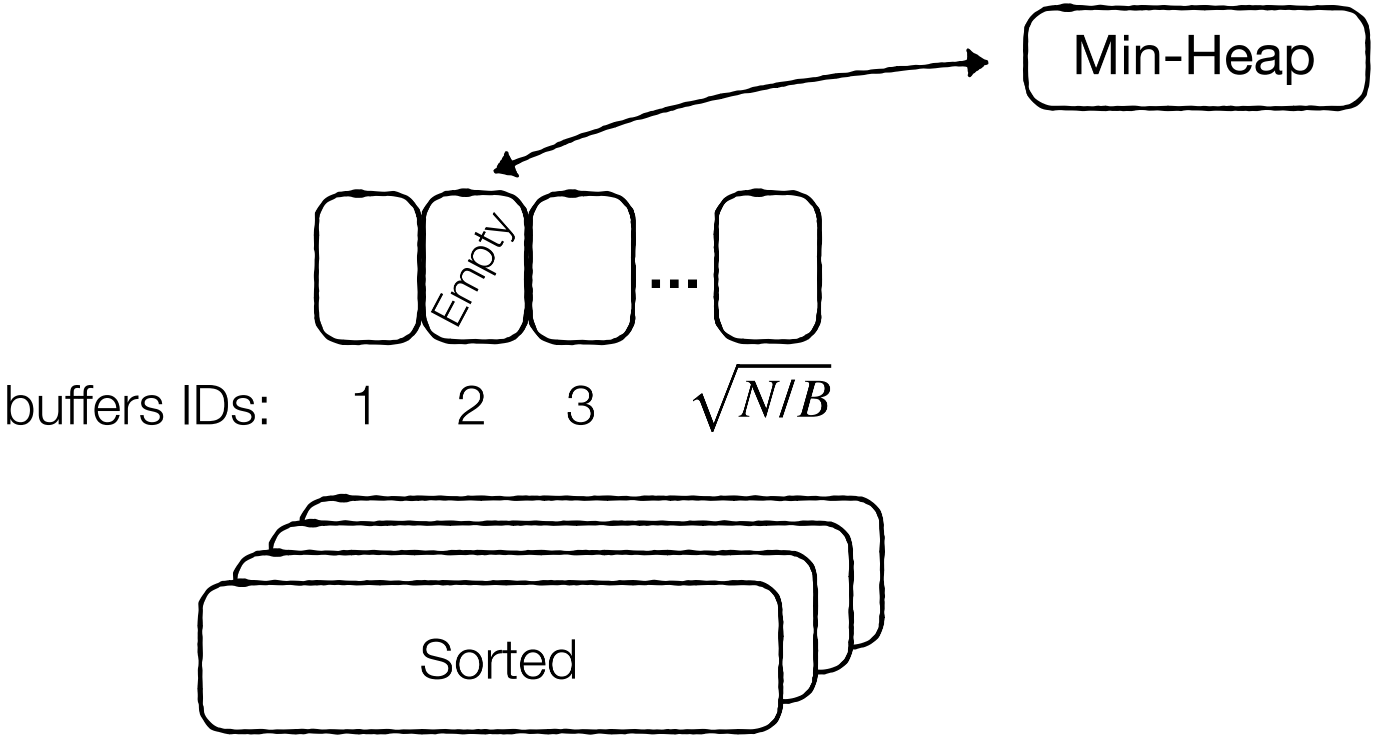
$O(N \cdot log_2 N)$ **CPU**

$O(N/B \cdot log_{M/B}(N/M))$ **I/O** **or** $O(N/B)$ **when** $M > \sqrt{N \cdot B}$

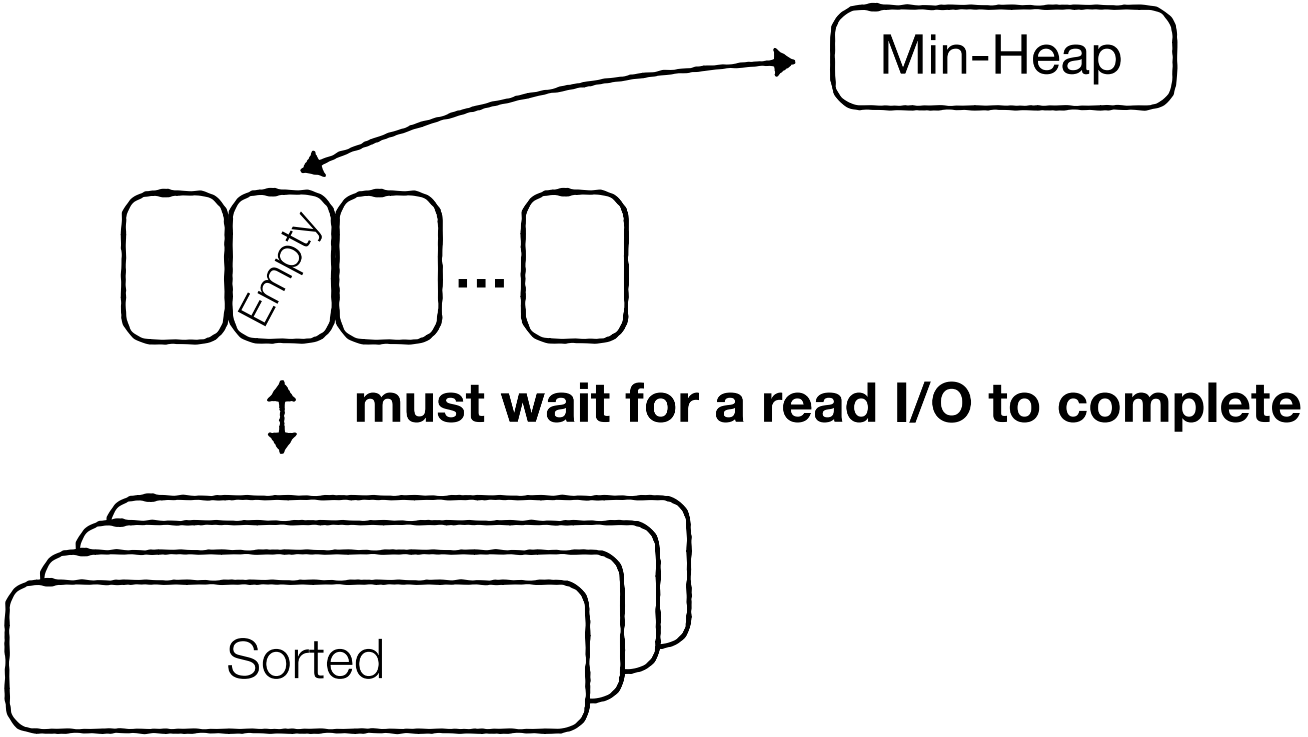




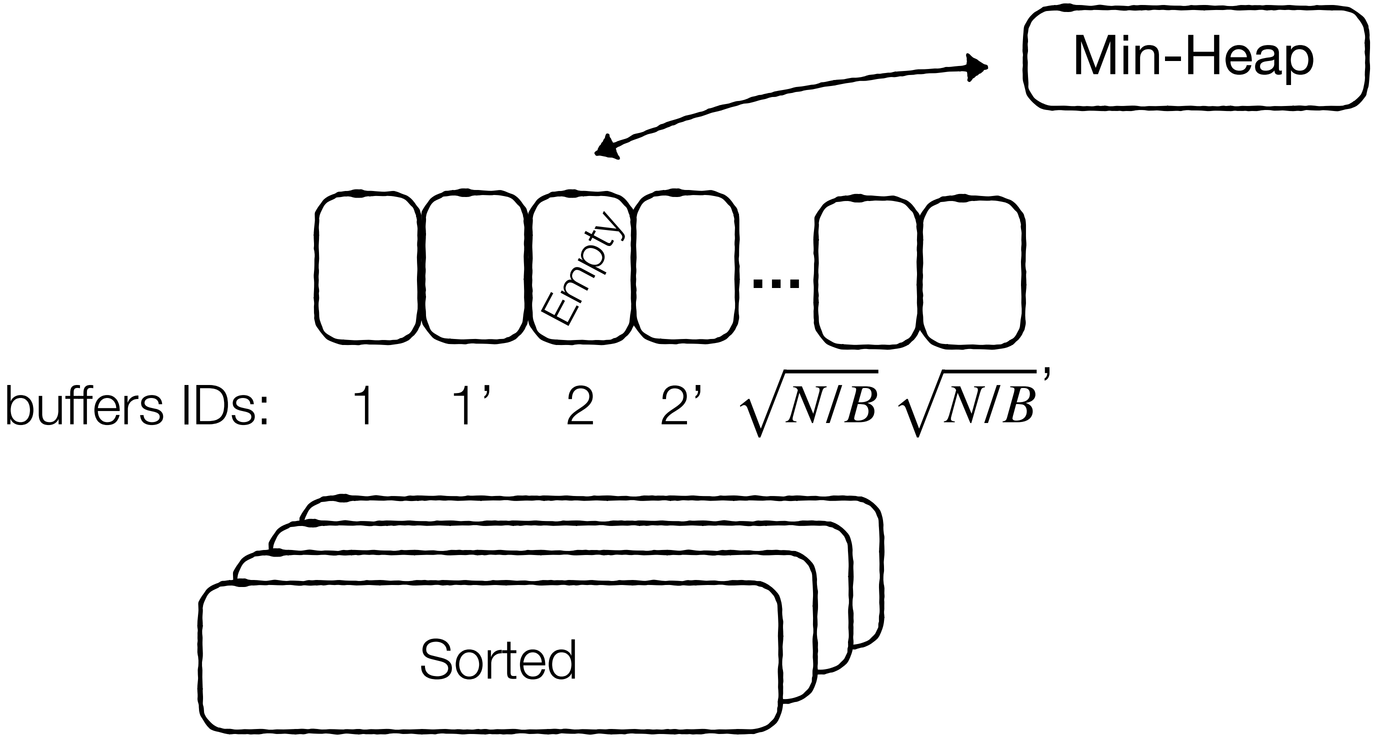
Suppose we need next min entry from buffer 2 but it is empty.



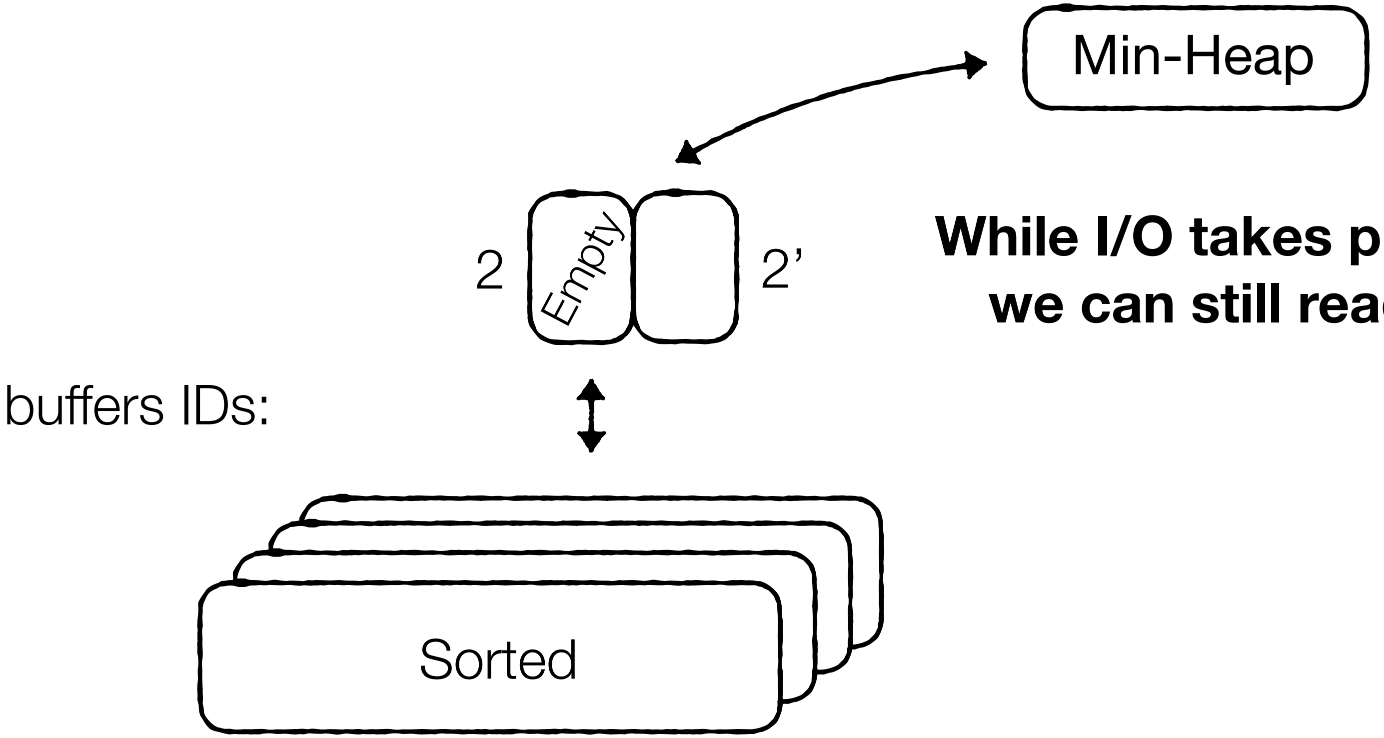
Suppose we need next min entry from buffer 2 but it is empty.



Double buffering: load one additional buffer preemptively for each partition before the first buffer empties



Double buffering: load one additional buffer preemptively for each partition before the first buffer empties



While I/O takes place into buffer 2, we can still read from buffer 2’

Larger though fewer buffers: more groups, so potentially more iterations, but each I/O reads more data