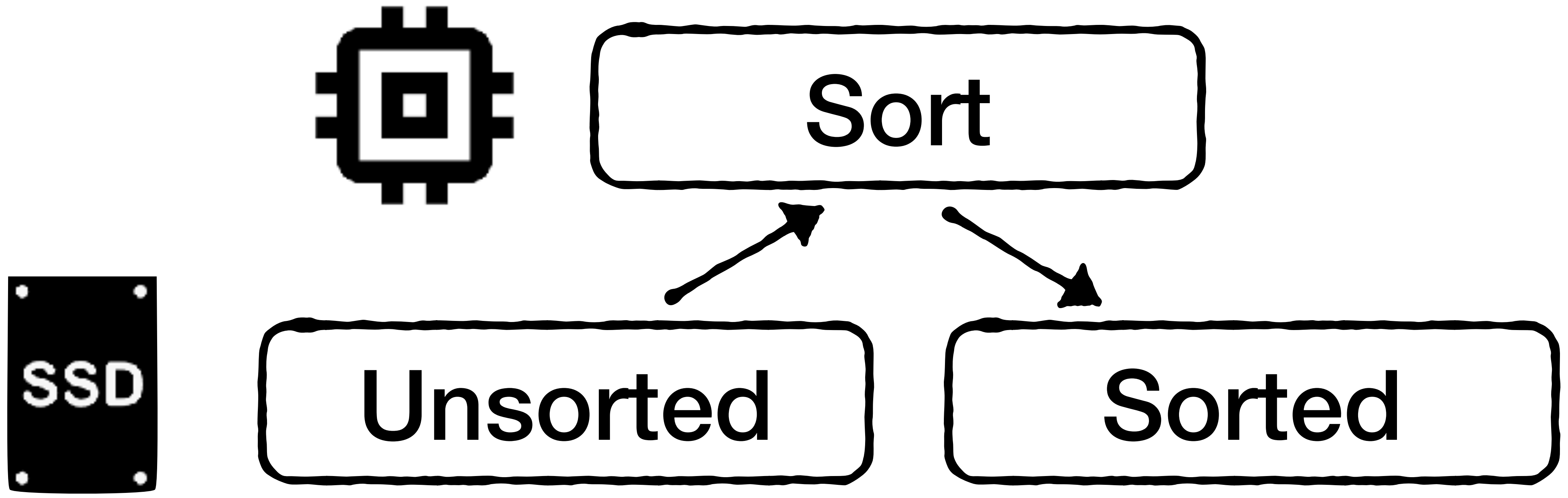
Analyzing I/O for External Sort

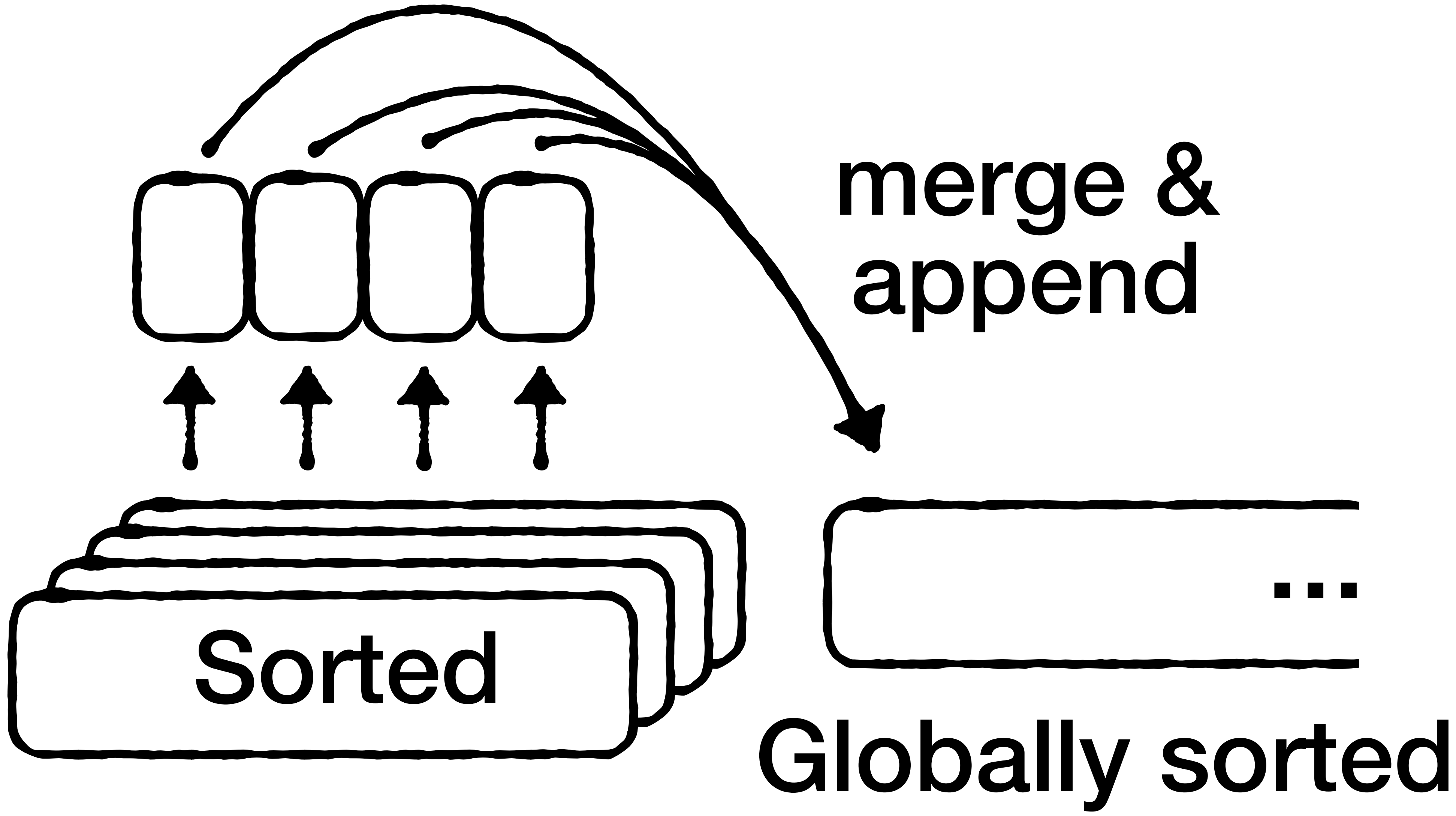
Partitioning Phase

N/B I/Os



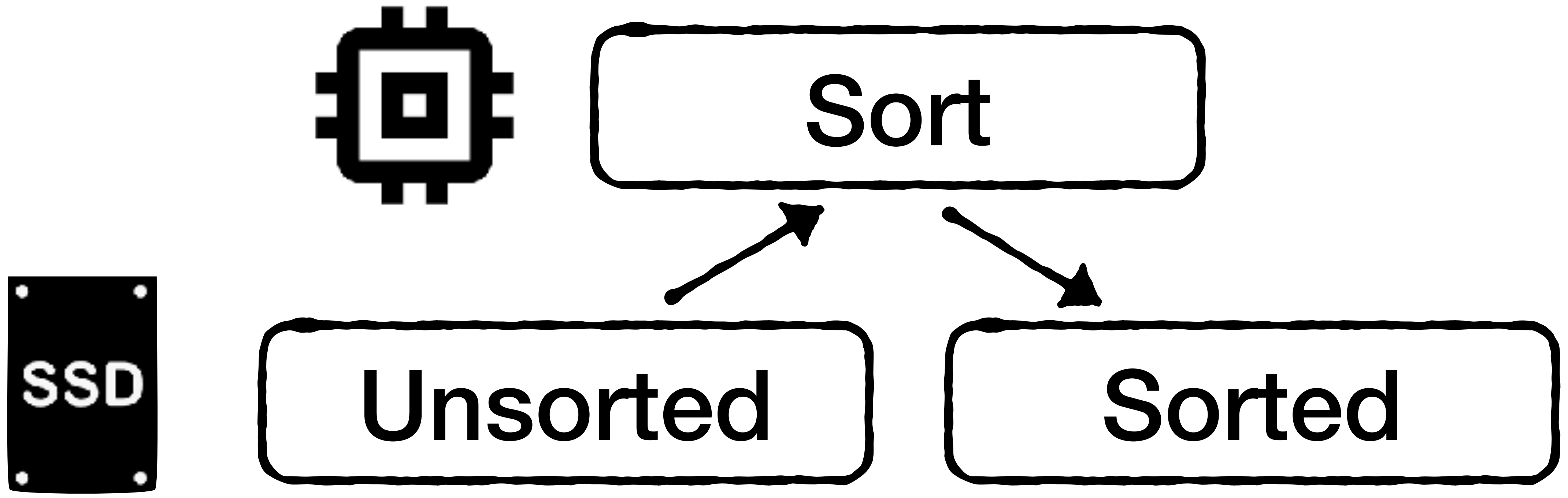
Merging Phase

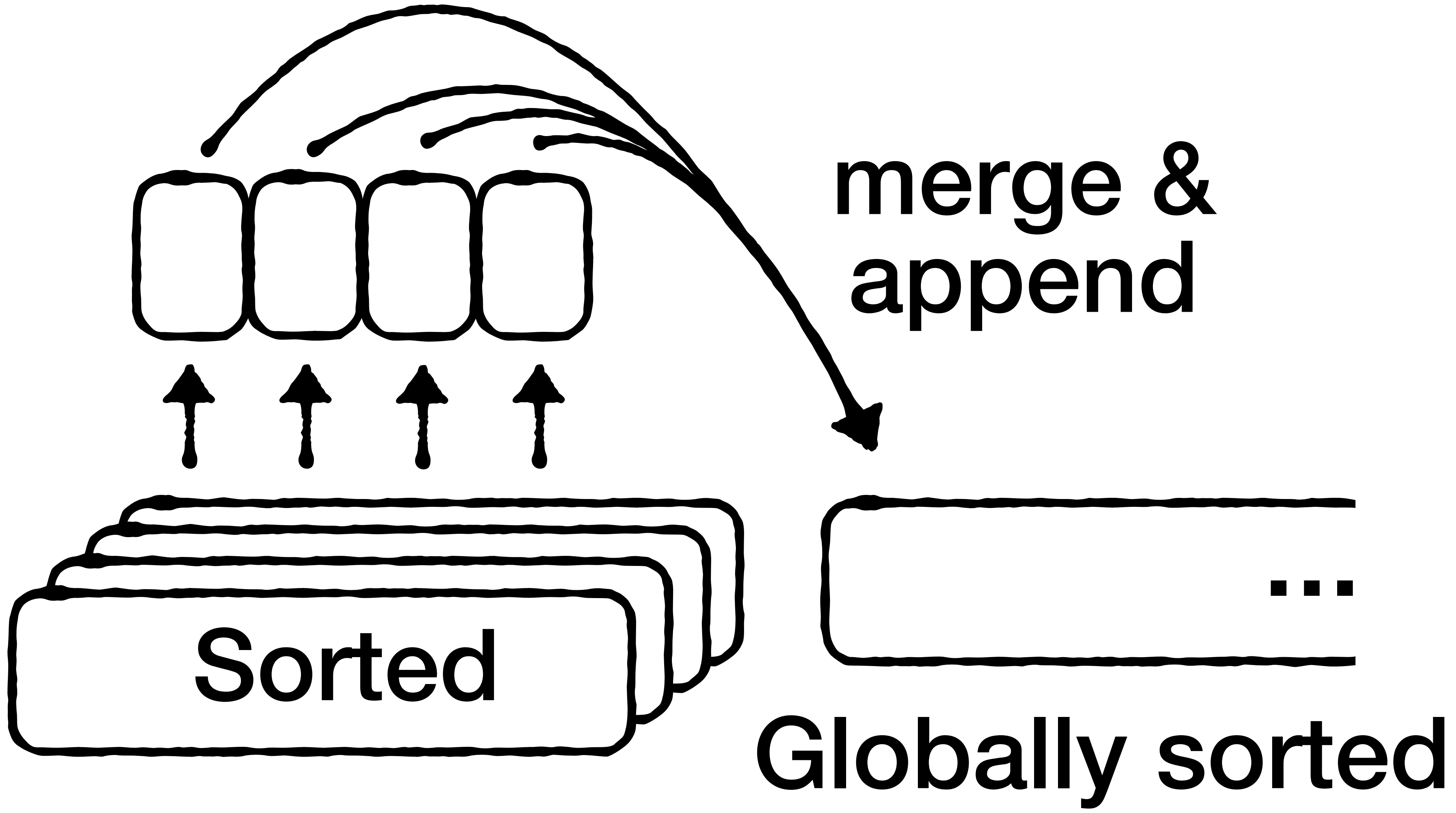
$$N/B * \lceil \log_{M/B}(N/M) \rceil$$



Analyzing I/O for External Sort

Partitioning Phase		Merging Phase		Total
N/B I/Os	+	$N/B * \lceil \log_{M/B}(N/M) \rceil$	=	$\mathbf{O(N/B * \log_{M/B}(N/B))}$

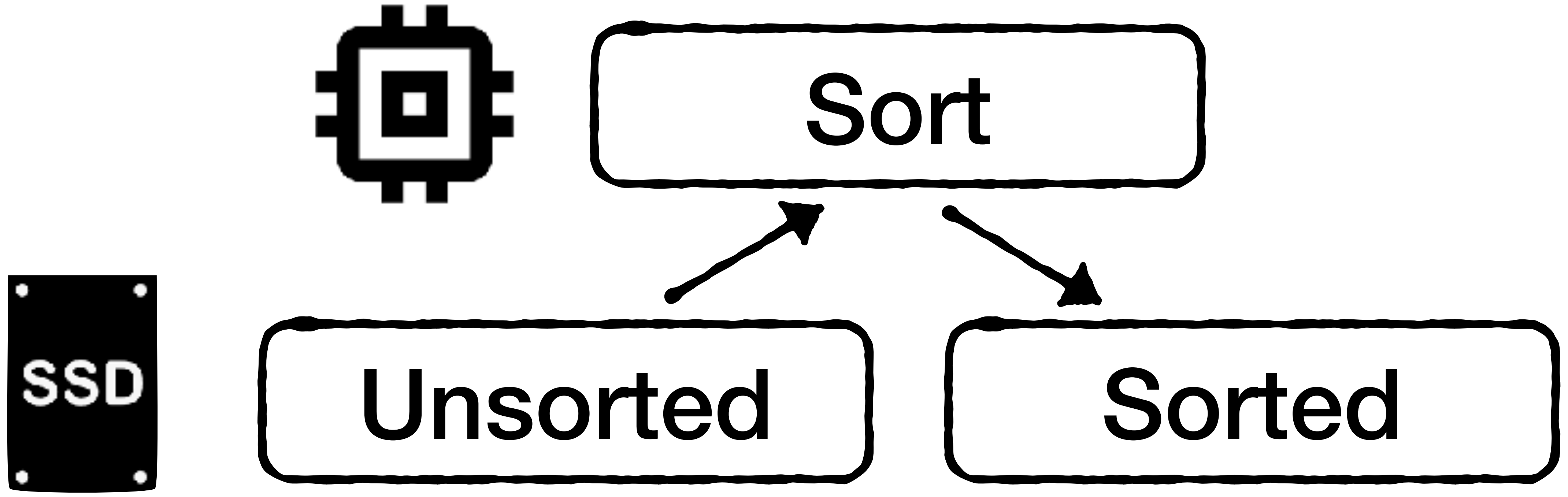


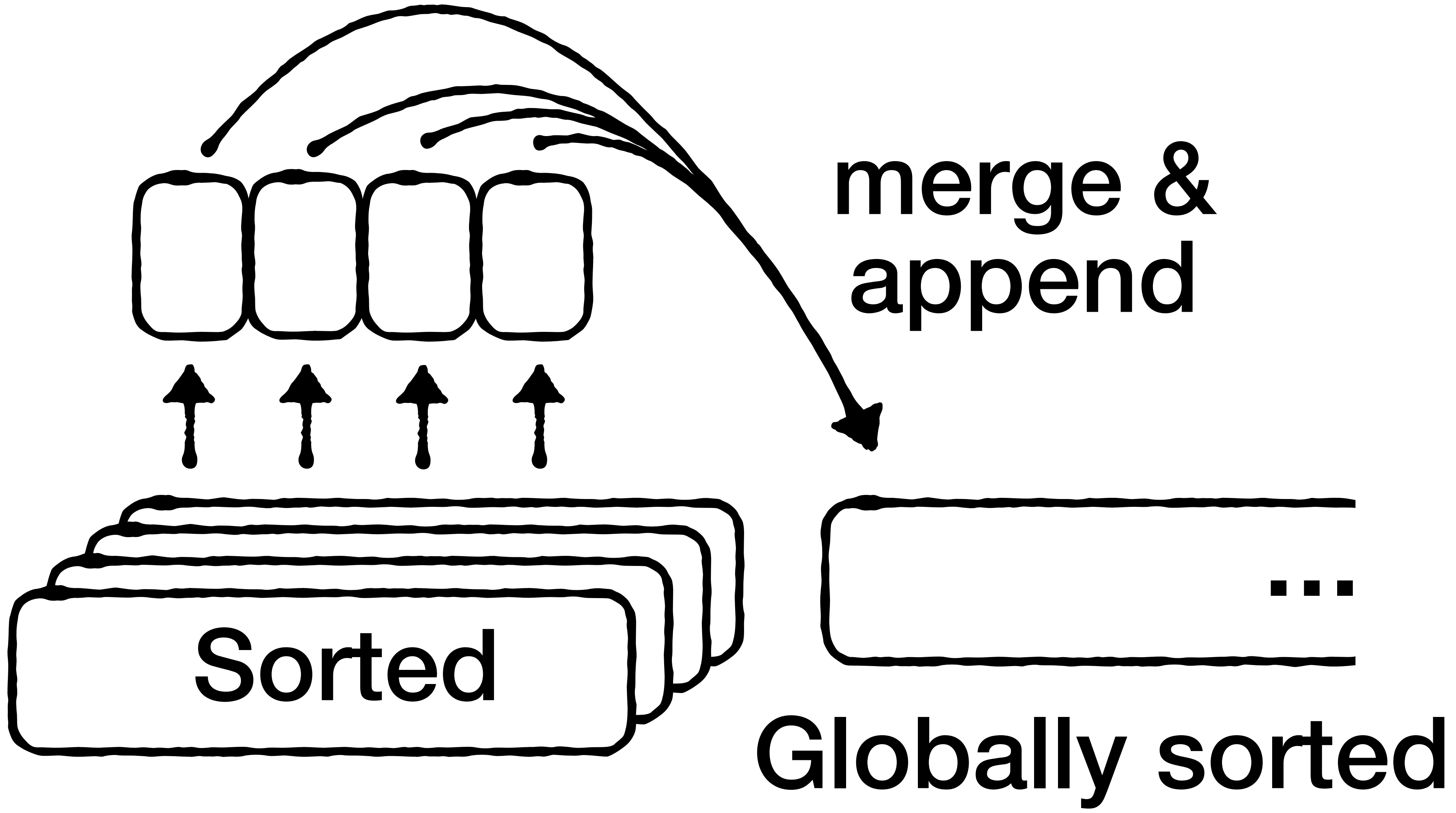


Analyzing I/O for External Sort

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

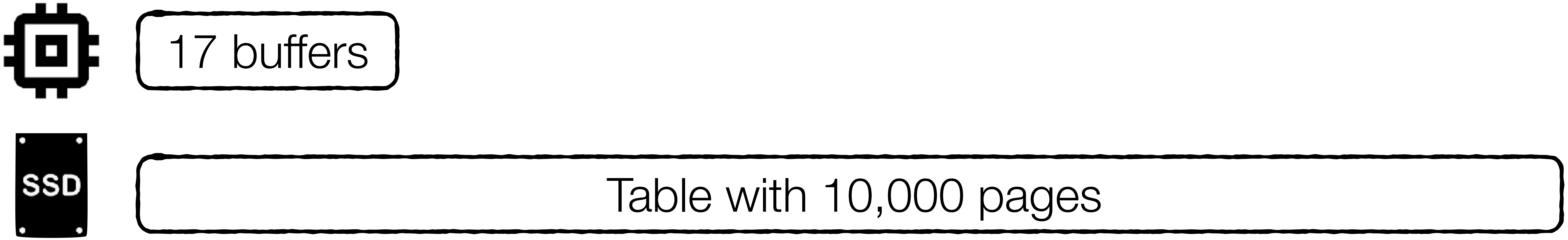
The more common cost expression you'd typically see





Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory. We need to externally sort the file.



Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory. We need to externally sort the file.

(A) How many partitions are created after the first pass?

$10{,}000 / 17 = 589$

(B) How many passes does it take to sort the file completely?

$\lceil 1 \text{ partitioning pass} + \log_{16}(589) \text{ merging passes} \rceil = 4$

(C) How many I/Os are issued in total to sort the file?

$10{,}000 * 4 = 40{,}000$

(D) How many buffers would you need to sort the file in just 2 passes?

$1 + \log_{M-1}(10{,}000/M) = 2$, and solve for M. M = 100

Question 1 - Follow up

Suppose you have a file with 10,000 pages and you have 17 buffers in memory. We need to externally sort the file.

Suppose we are I/O-bound (CPU is not the bottleneck). Would it be better to employ a 2 page buffer for each node? Answer for disk vs. SSD.

This would entail one additional merging pass over the data:
$\lceil 1$ partitioning pass + $\log_8(589)$ merging passes $\rceil = 5$

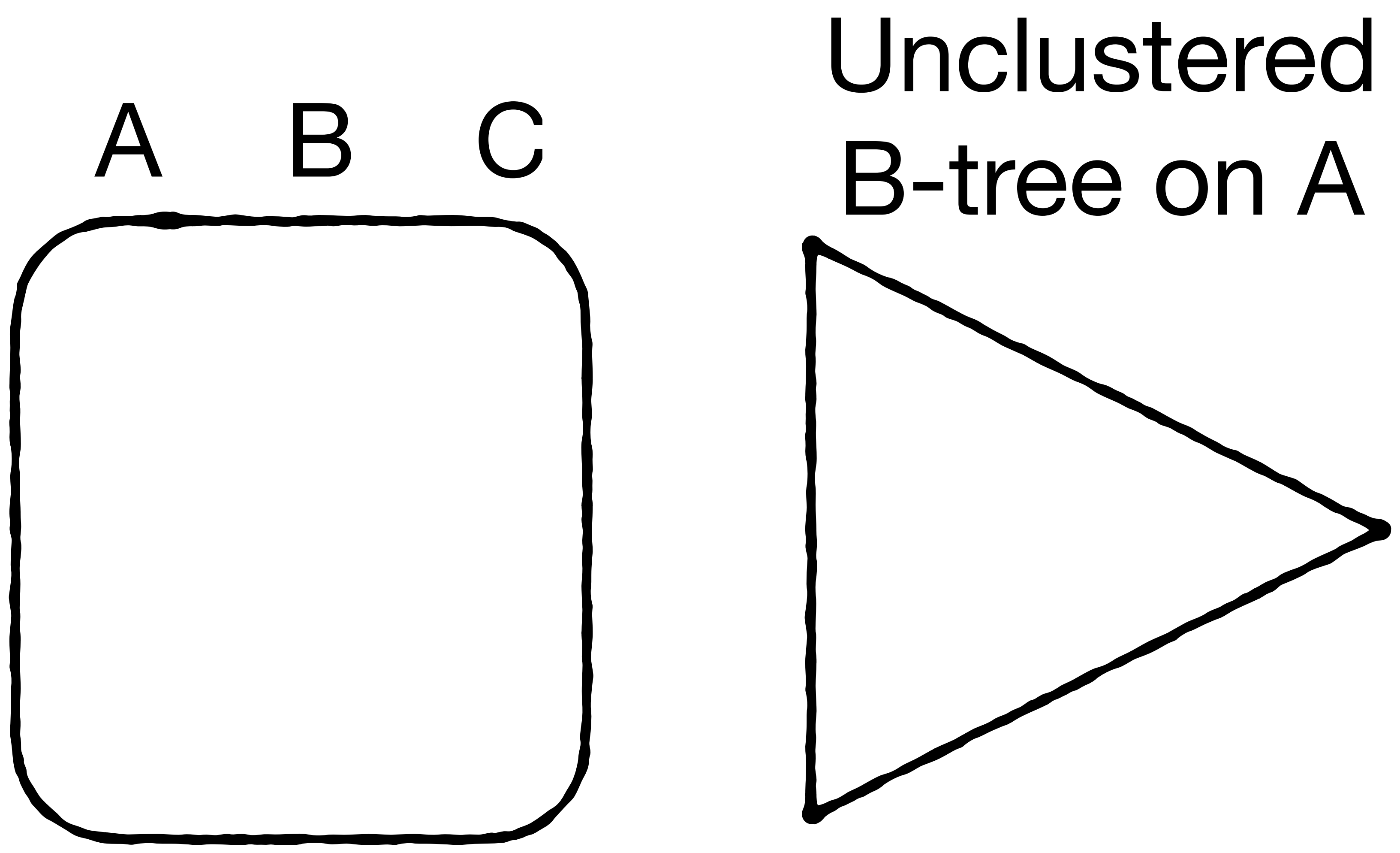
This means we need to do 50,000 read I/Os rather than 40,000 as before.

On SSD, we get a 25% slowdown assuming sequential access costs the same as random access.

On disk, we can do two I/Os at approx. the cost of 1, so the algorithm completes in time needed for 25,000 I/Os. This can lead to a 37% speedup.

Question 2

Consider a table that is too large to fit in memory. We have an unclustered B-tree index over column A. We get the following query: “Select * from table sort by A”. There are two options to process this query: (1) scan the index, or (2) externally sort the file based on column A. What are the costs of these methods, and under which circumstances which you choose each one?



Select * from table sort by A

Question 2

Consider a table that is too large to fit in memory. We have an unclustered B-tree index over column A. We get the following query: “Select * from table sort by A”. There are two options to process this query: (1) scan the index, or (2) externally sort the file based on column A. What are the costs of these methods, and under which circumstances which you choose each one?

(1) An unclustered index requires issuing one I/O into the table for each entry. Retrieving the table in a sorted order therefore takes O(N) I/Os.

(2) External sorting takes $O(N/B * \log_{M/B}(N/B))$ I/Os. It is cheaper as long as $\log_{M/B}(N/B) < B$. This holds true for realistic values N, B and M.

Method (2) is generally better, but if we have large entry sizes (small B), little memory, and/or astronomical data, approach (1) may be better.

Question 2

Consider a table that is too large to fit in memory. We have an unclustered B-tree index over column A. We get the following query: “Select * from table sort by A”. There are two options to process this query: (1) scan the index, or (2) externally sort the file based on column A. What are the costs of these methods, and under which circumstances which you choose each one?

Follow up: how would your answer change if we have a clustered index on column A?

Scanning the clustered index to return sorted data now costs N/B I/Os, which is cheaper than even even a two-pass external sort, which costs 2N/B.

Question 3

You are given M memory and N entries where N >> M >> 3. Why is it a bad idea to use the available memory as virtual memory (e.g., to allocate using 'new' space for M entries, and to use in-memory Quicksort? Use cost models to justify your answer.

With virtual memory, swapping would kick in. Quicksort scans the data sequentially, so we would expect $O(N/B * \log_2 N)$ I/Os as it performs $\log_2 N$ iterations.

In contrast, external sort provides $O(N/B * \log_{M/B}(N/B))$, which dominates.

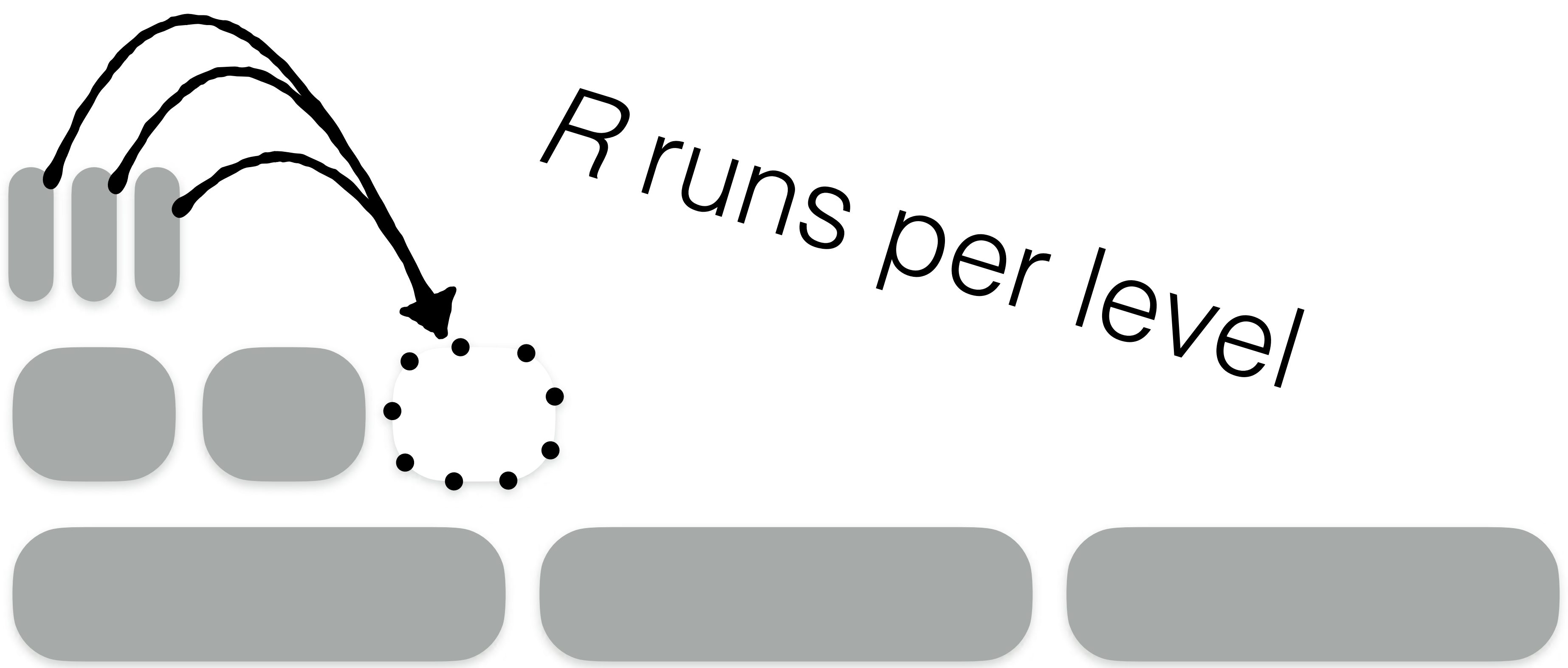
Question 4

Suppose we have a tiered LSM-tree with a size ratio of R. Analyze the CPU costs of compaction. Then propose a technique to improve CPU costs.

Each entry is merged O($\log_T(N/P)$) times across the tree

For each entry we merge, we must check the minimum across O(R) runs.

Total CPU costs: O(N * $\log_T(N/P)$ * R)



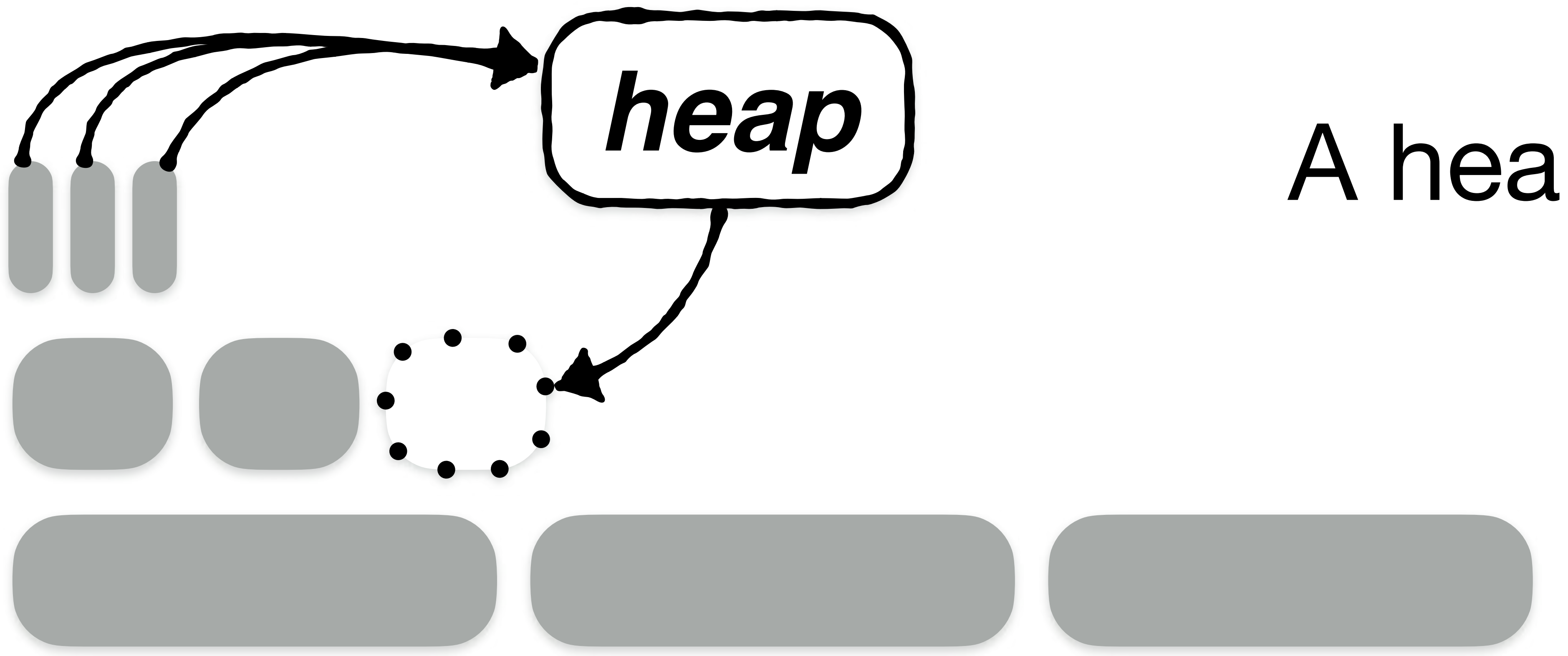
Question 4

Suppose we have a tiered LSM-tree with a size ratio of R. Analyze the CPU costs of compaction. Then propose a technique to improve CPU costs.

Each entry is merged O($\log_T(N/P)$) times across the tree

For each entry we merge, we must check the minimum across O(R) runs.

Total CPU costs: $O(N * \log_T(N/P) * R)$



A heap brings this down to $O(N * \log_T(N/P) * \log_2(R))$