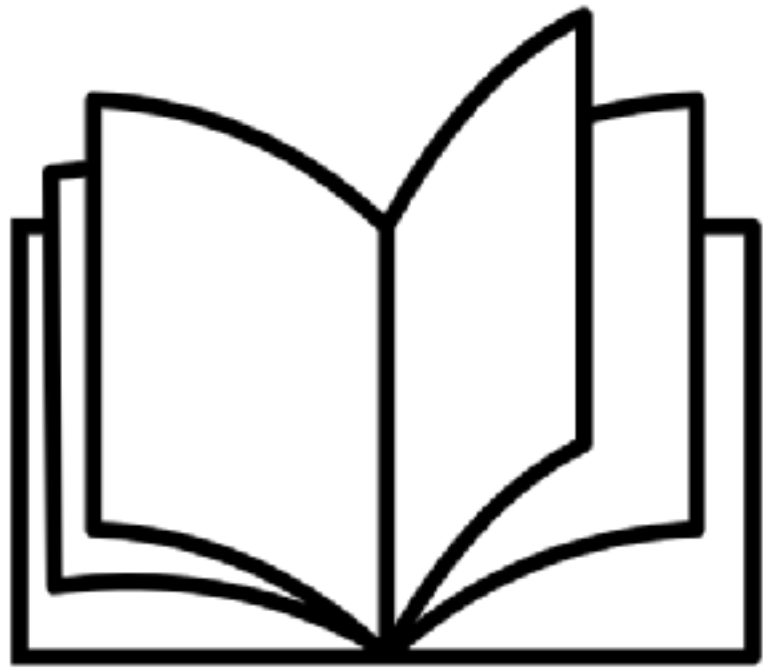
Tutorial on Sorting

Database System Technology

Niv Dayan

Midterm covers all
material so far
(Including this week)



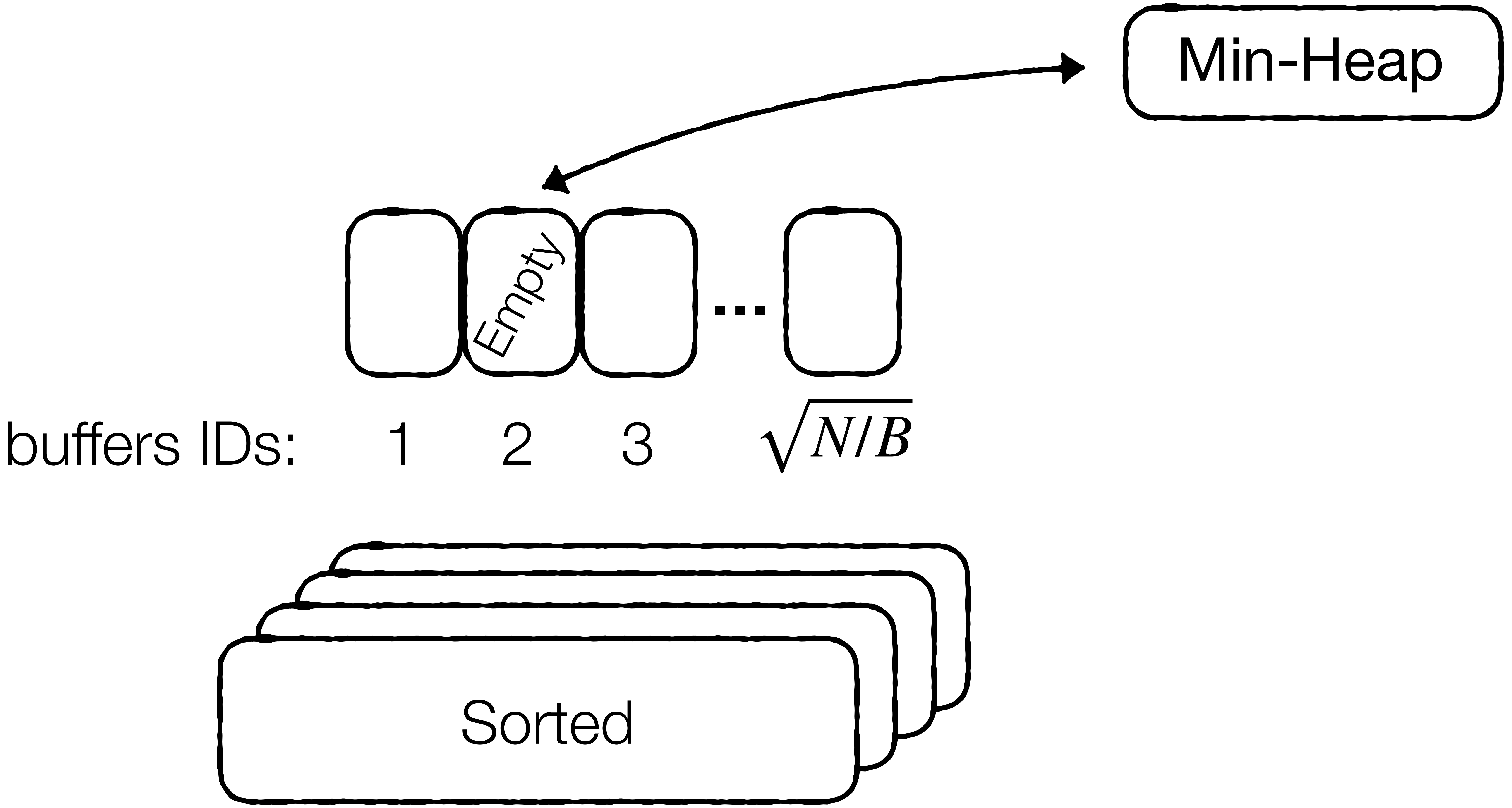
Open book

Office Hours

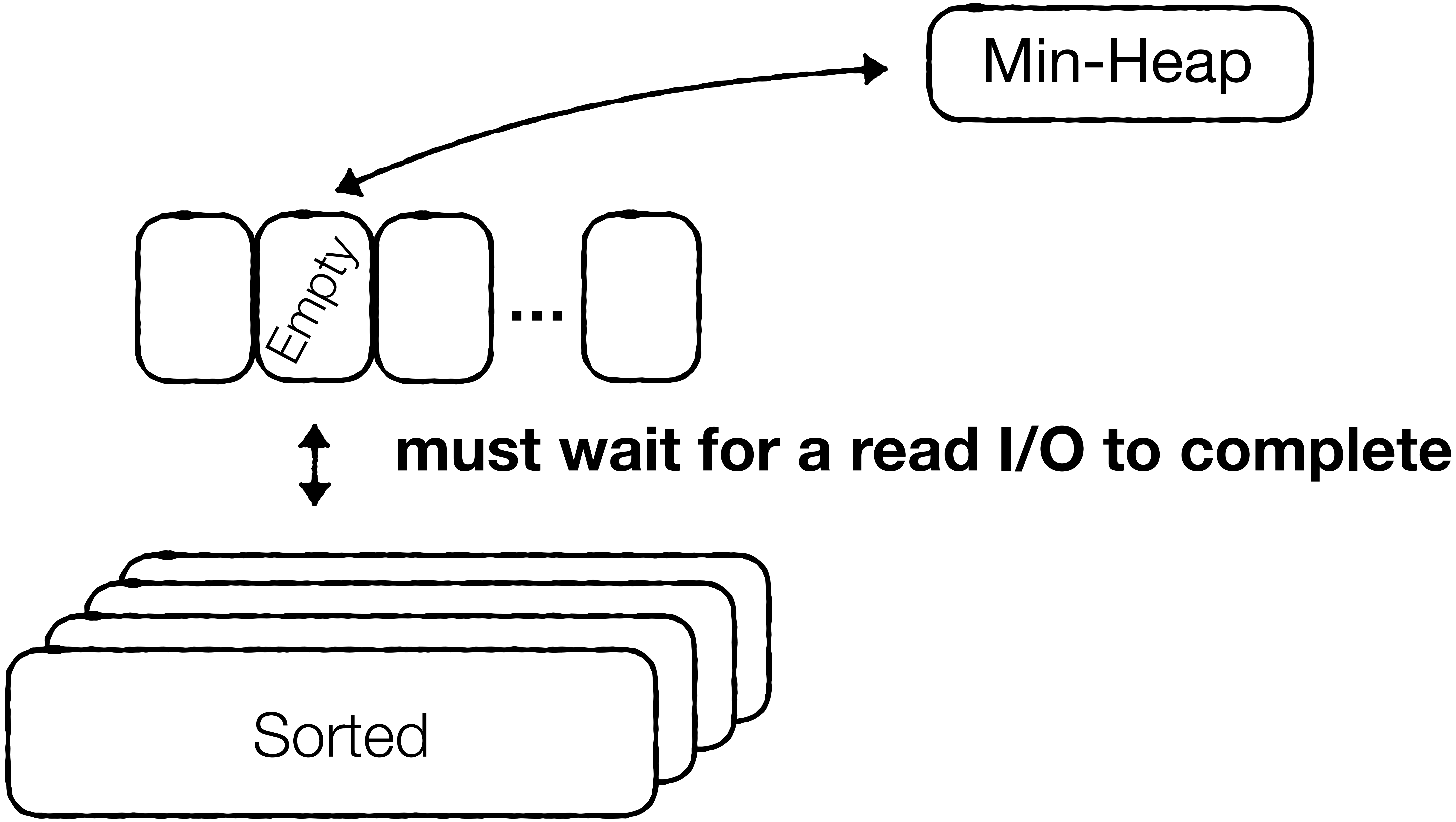
TA Office Hours
4-6 pm
(Project + content)

Instructor Office Hours
4-5 pm and 7-8 pm

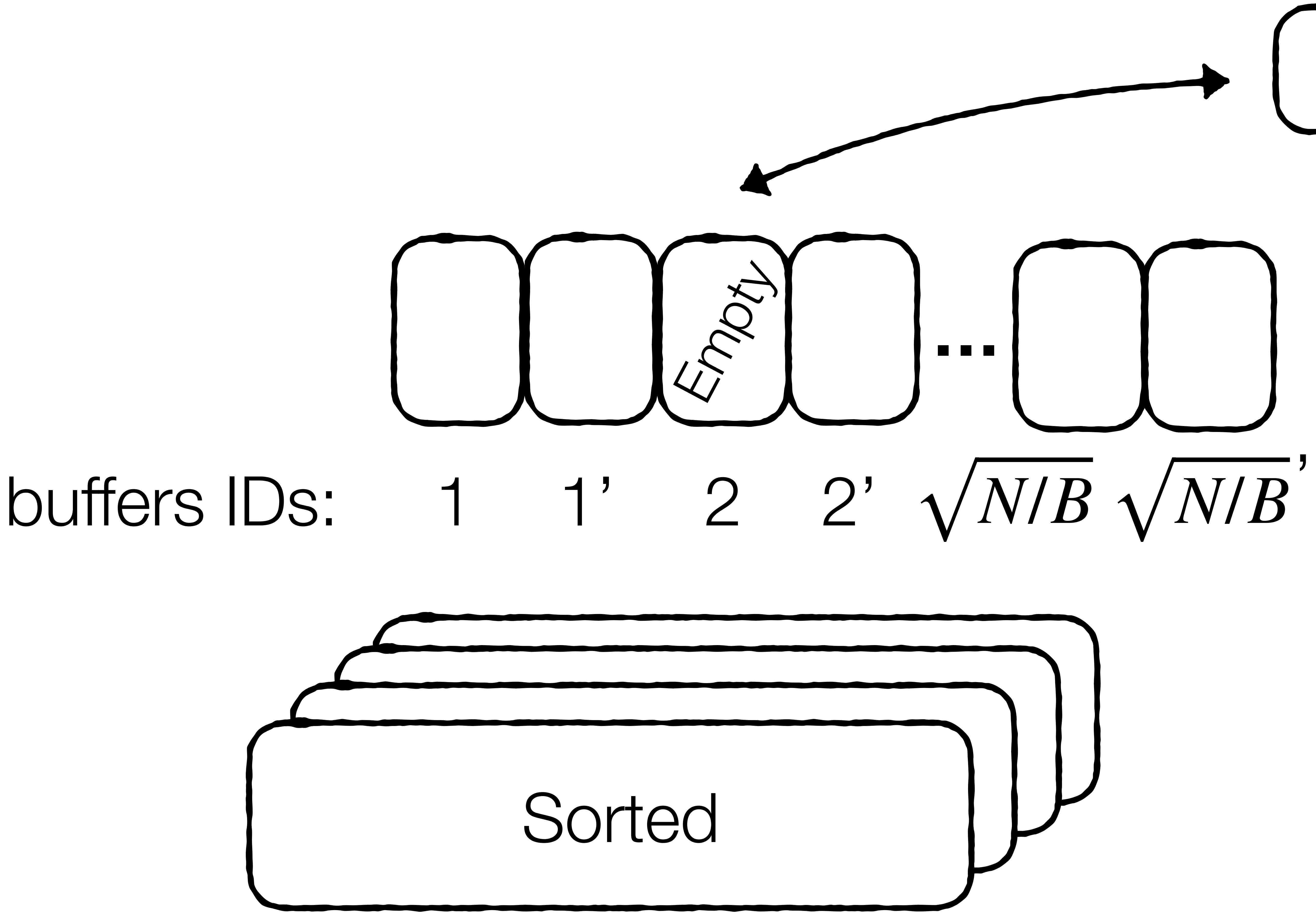
Suppose we need next min entry from buffer 2 but it is empty.



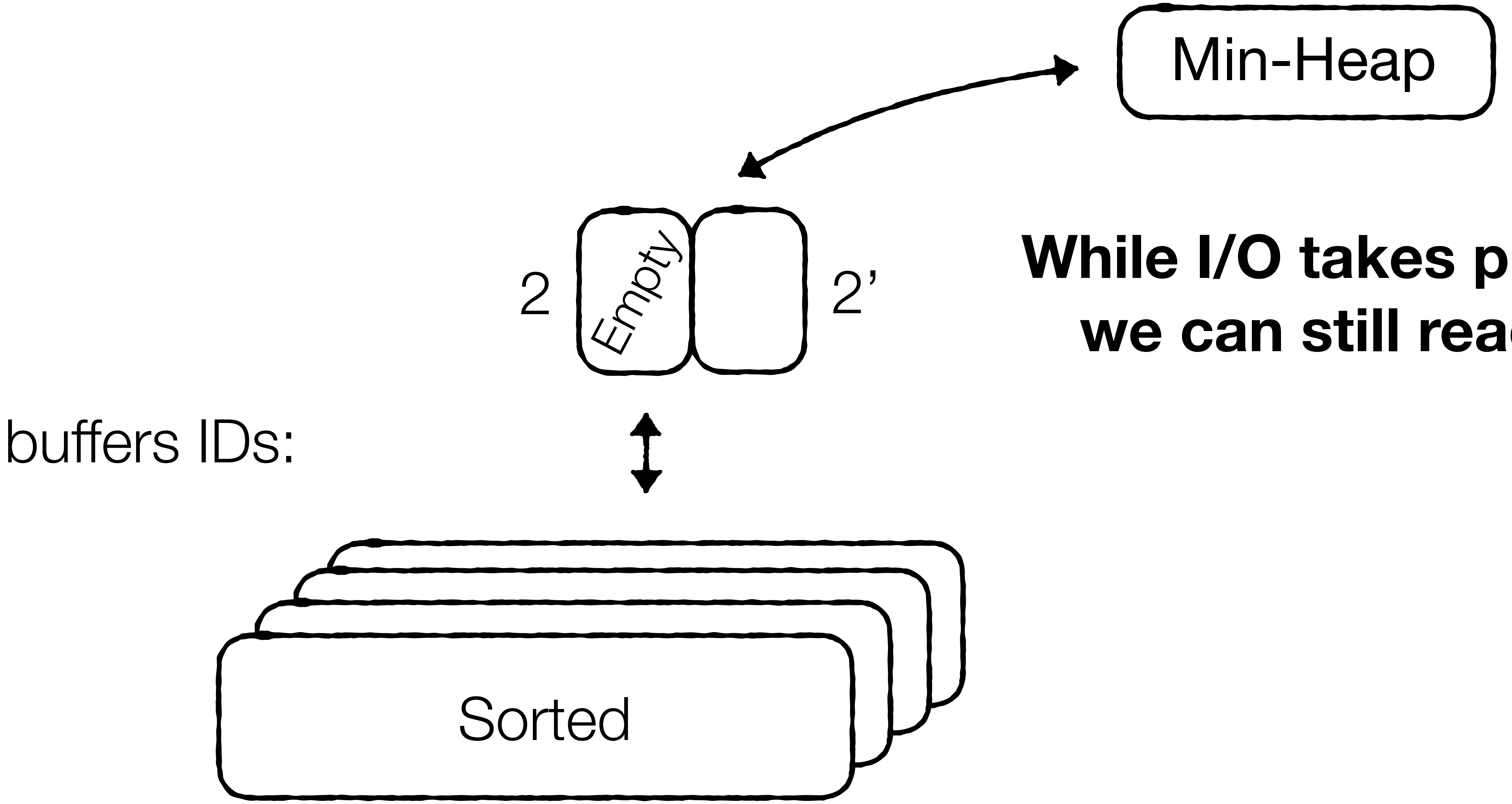
Suppose we need next min entry from buffer 2 but it is empty.



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

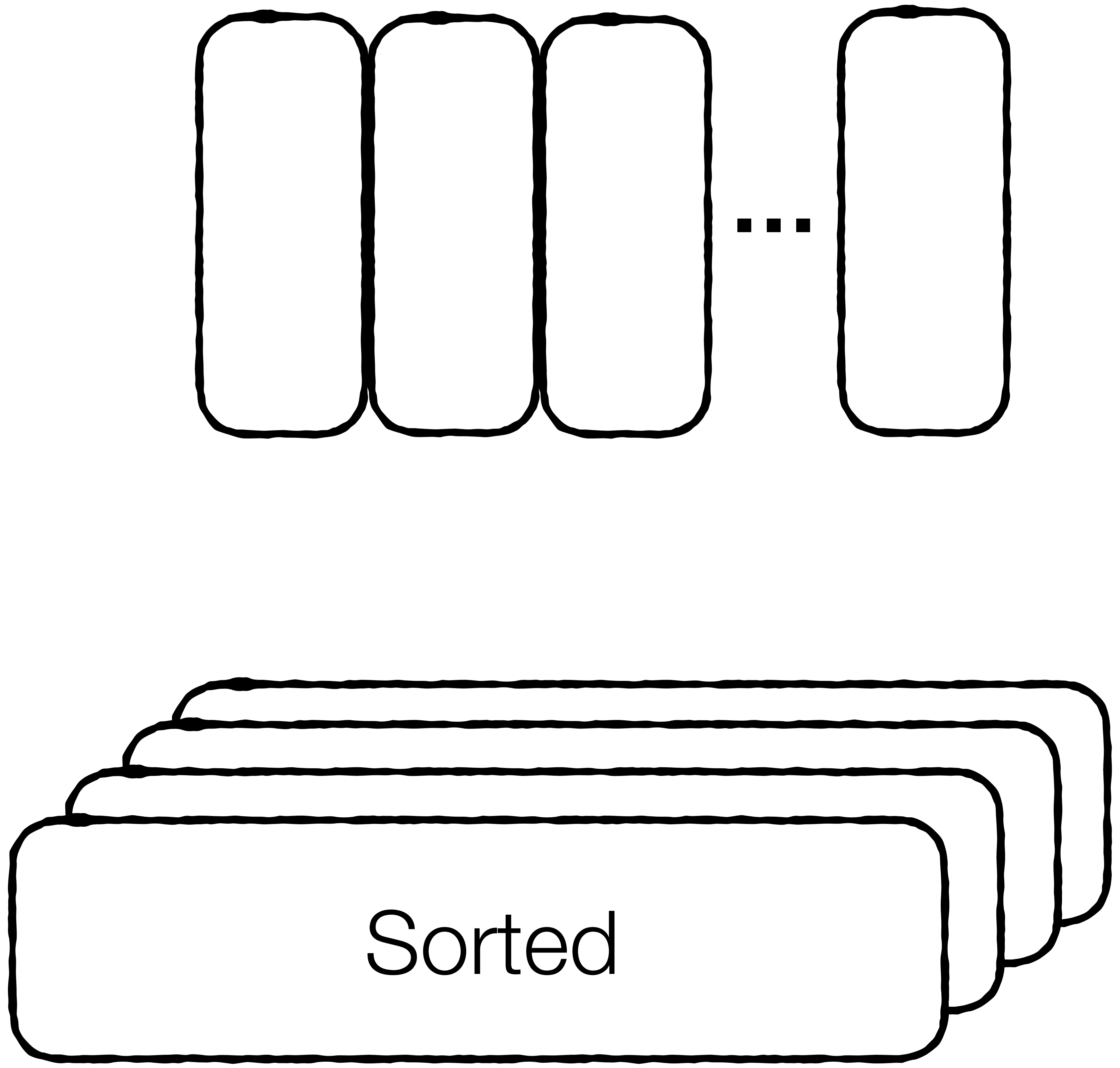


Double buffering: load one additional buffer preemptively for each partition before the first buffer empties



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



Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory, each of them 4KB. We need to externally sort the file.

17 buffers

Table with 10,000 pages

Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory, each of them 4KB. We need to externally sort the file.

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

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

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

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

Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory, each of them 4KB. 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?

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

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

Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory, each of them 4KB. 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?

$$\mathbf{\log_{M/B - 1}(N/B) = \lceil \log_{16}(10{,}000) \rceil = 4}$$

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

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

Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory, each of them 4KB. 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?

$\log_{M/B - 1}(N/B) = \lceil \log_{16}(10{,}000) \rceil = 4$

(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?

Question 1

Suppose you have a file with 10,000 pages and you have 17 buffers in memory, each of them 4KB. We need to externally sort the file.

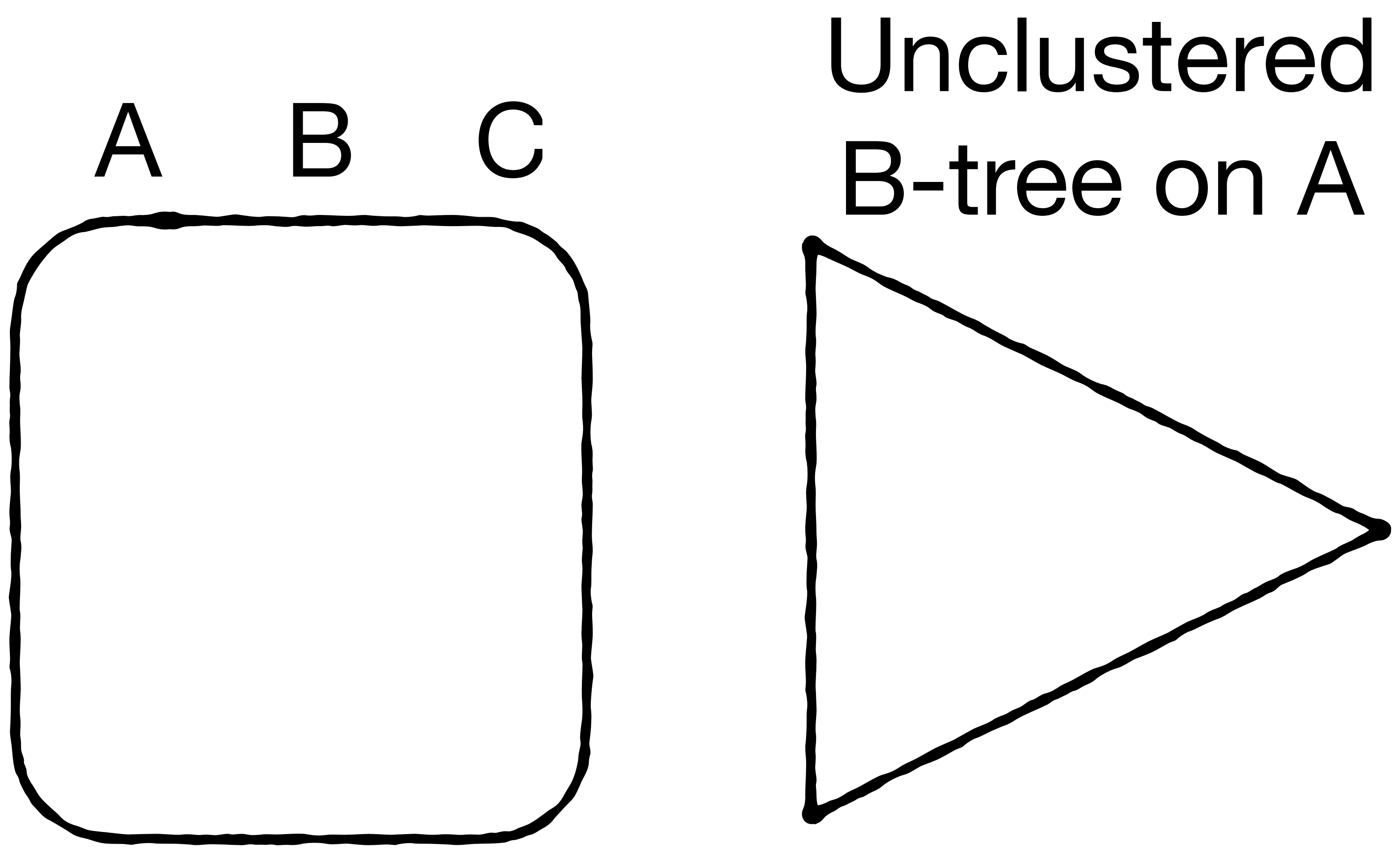
(D) How many buffers would you need to sort the file in just 2 passes?

The total number of passes generally is $\log_{M/B}(N/B)$ where M/B is the number of buffers. We know that N/B = 10,000. So we can equate $\log_{M/B}(N/B)$ to 2 and solve for M/B

$\log_{M/B}(10{,}000) = 2$ -> M/B = 100

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 order 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 would 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 order 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 would 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.

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 order 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 would 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 $O(N/B \cdot \log_{M/B}(N/B))$ I/Os. It is cheaper as long as $\log_{M/B}(N/B) < B$. This holds true for realistic values N, B and M.

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 order 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 would 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 $O(N/B \cdot \log_{M/B}(N/B))$ 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 order 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 would you choose each one?

Follow up: how would your answer change if we have a clustered index on column 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 order 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 would 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 2(N/B) I/Os.

Question 3

You are given M memory and N entries where N >> M >> 3B. 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.

Question 3

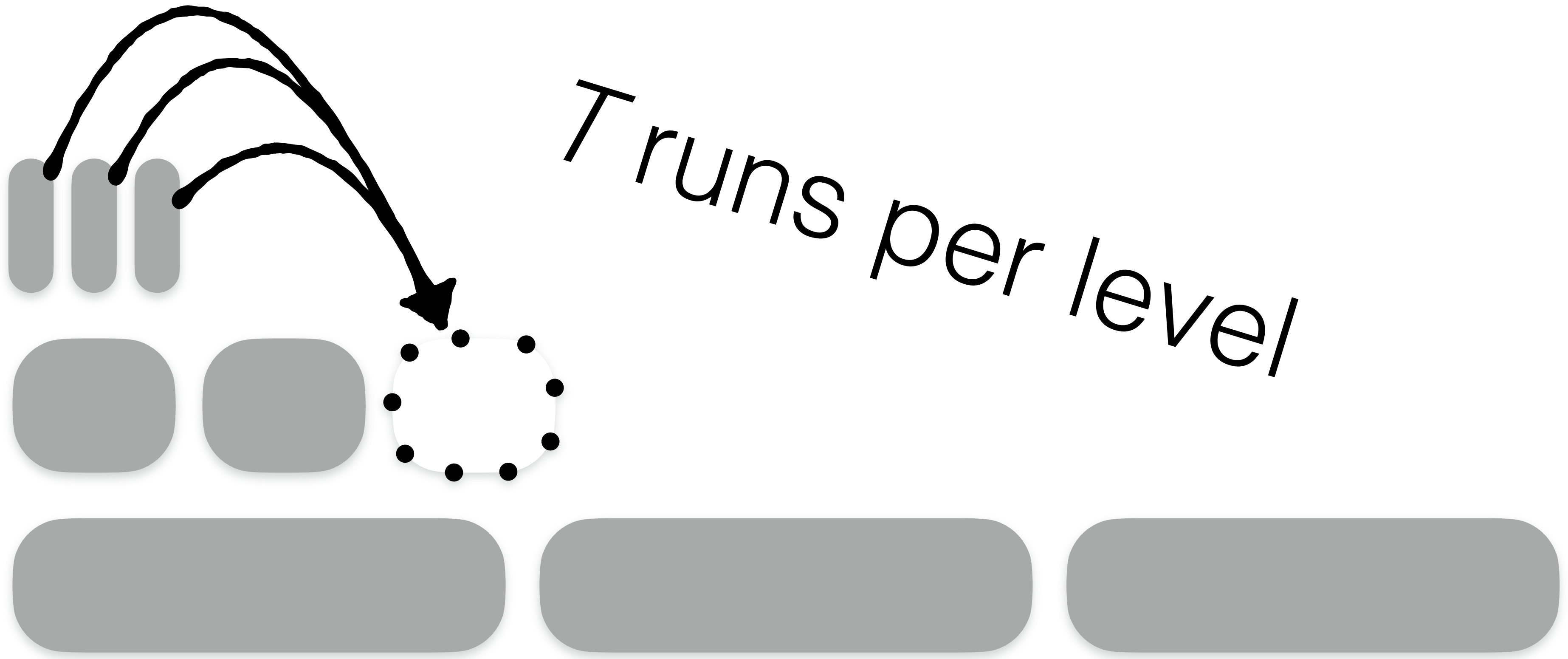
You are given M memory and N entries where N >> M >> 3B. Why is it a bad idea to use the available memory as virtual memory (e.g., to allocate using 'new' space for M entries, and to use in-memory Quicksort? Use cost models to justify your answer.

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

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

Question 4

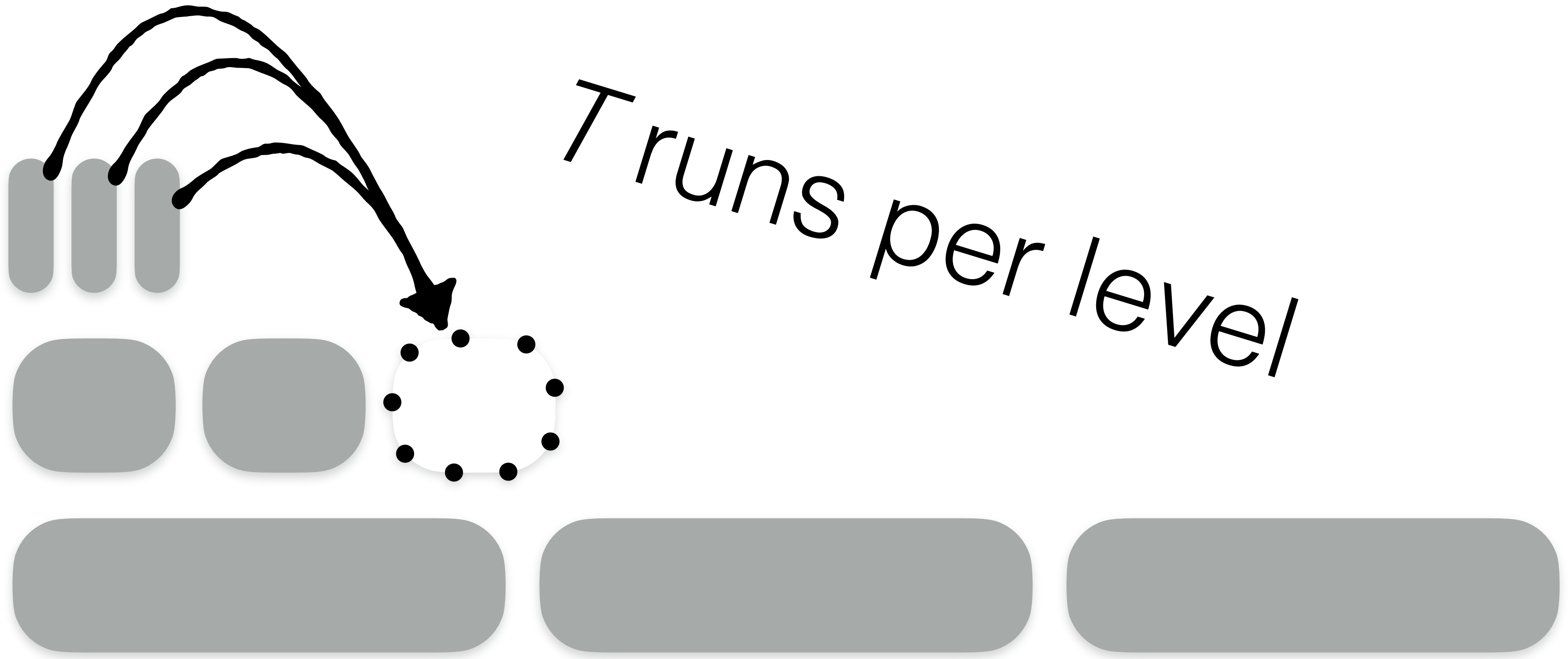
Suppose we have a tiered LSM-tree with a size ratio of T. Analyze the CPU costs of compaction assuming we check all buffers each time to find the minimum key. Then propose a technique to improve CPU costs.



Question 4

Suppose we have a tiered LSM-tree with a size ratio of T. Analyze the CPU costs of compaction assuming we check all buffers each time to find the minimum key. Then propose a technique to improve CPU costs.

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

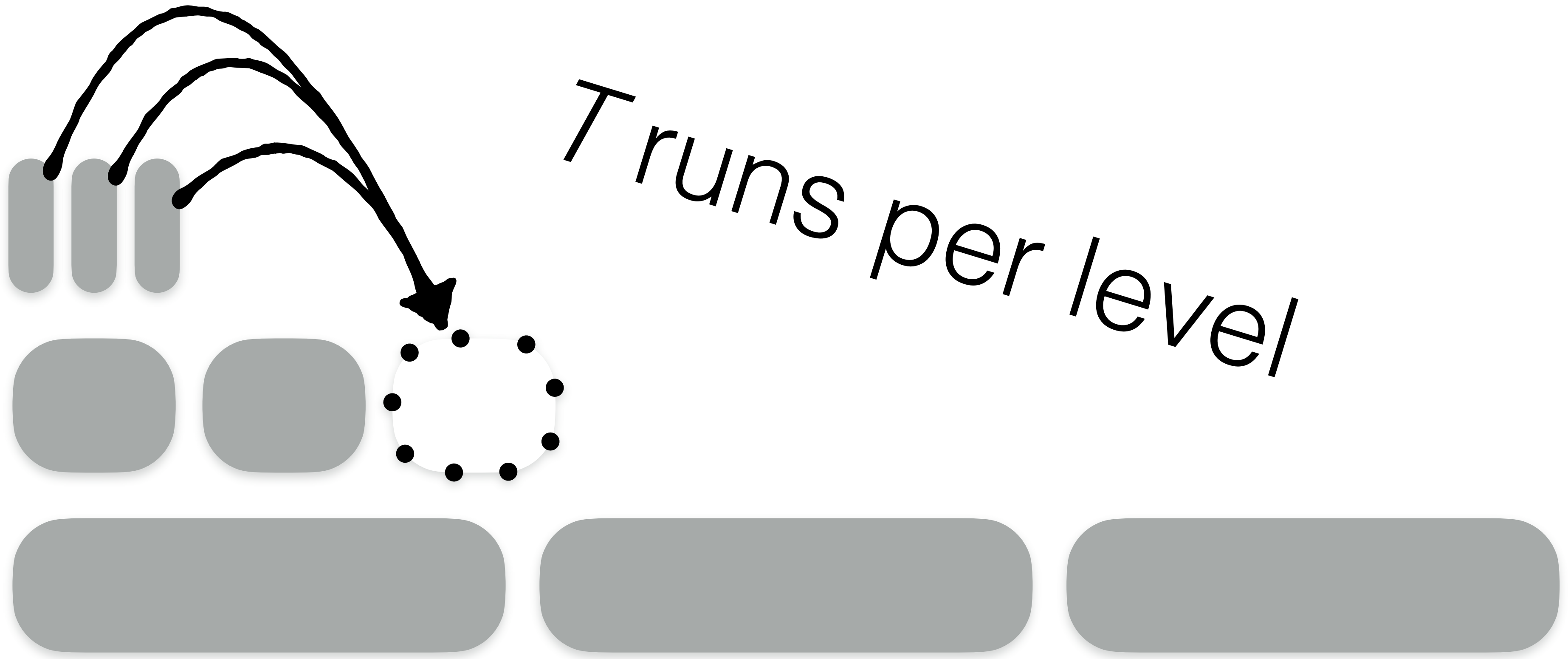


Question 4

Suppose we have a tiered LSM-tree with a size ratio of T. Analyze the CPU costs of compaction assuming we check all buffers each time to find the minimum key. 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(T) buffers.



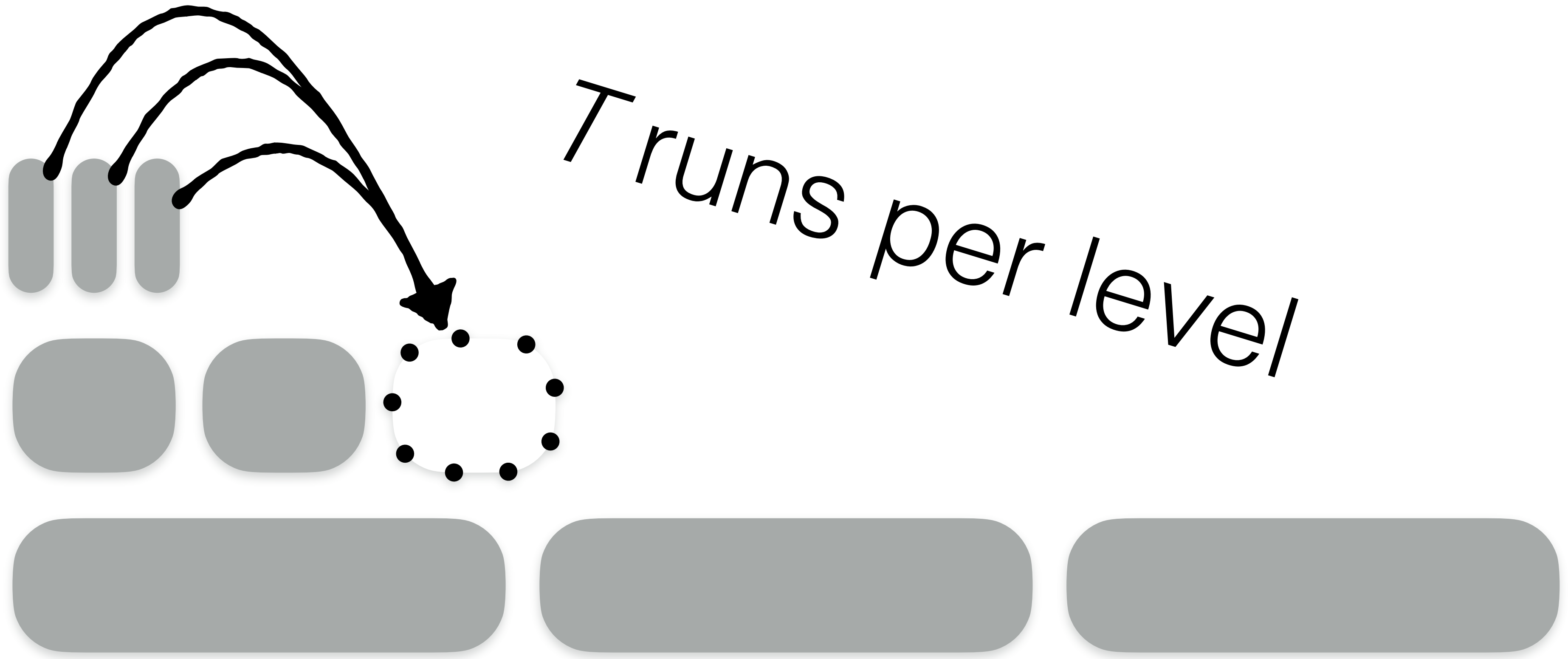
Question 4

Suppose we have a tiered LSM-tree with a size ratio of T. Analyze the CPU costs of compaction assuming we check all buffers each time to find the minimum key. 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(T) buffers.

Total CPU costs: $O(N \cdot \log_T(N/P) \cdot T)$



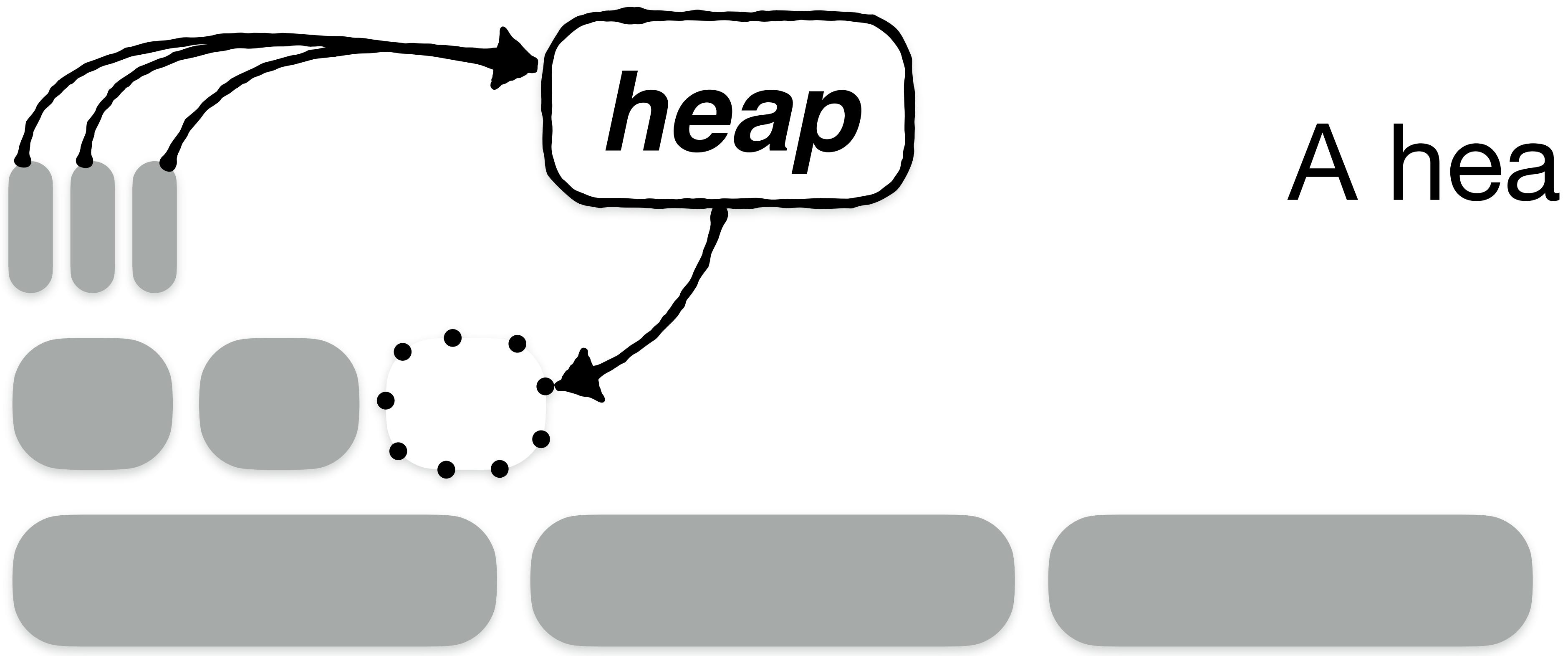
Question 4

Suppose we have a tiered LSM-tree with a size ratio of T. Analyze the CPU costs of compaction assuming we check all buffers each time to find the minimum key. 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 $O(T)$ buffers.

Total CPU costs: $O(N \cdot \log_T(N/P) \cdot T)$



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