Discussion 9

Sorting

EECS 484

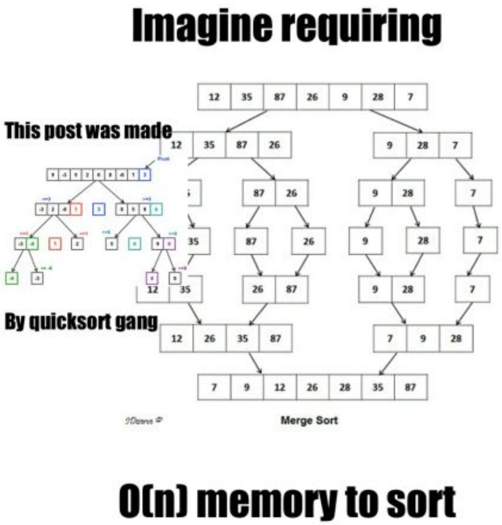
Logistics

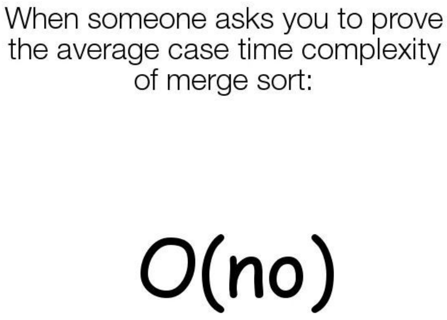
- Homework 5 is due on Thursday, Nov 17 at 11:55 PM
- Project 3 is due this Thursday, Nov 10 at 11:55 PM
- Project 4 is out Nov 10 and due on Dec 8. You will have 4 weeks to complete P4!
- Final exam is on Dec 13, which immediately follows the due date of this project, so please be aware of this when scheduling your time!

External Sorting

External Sorting

- Sorting is nice
  - We have lots of nice algorithms that will sort for us
    - Quicksort, Mergesort, Heapsort, etc.
  - We can do this very quickly with lots of data - O(N*log(N))
- But what if we have too much data to fit in RAM?
  - We can still sort but it will be so so slow :(
  - Need some way to *externally* sort the data on the disk while dealing with limited fast memory





General External Merge Sort

disk　　$N >> B$　　in memory

- Step 1:
  - Have a large dataset of N pages that you would like to sort using B buffer pages

$\lceil \frac{N}{B} \rceil$

- Step 2:
  - Divide the dataset into ceiling(N/B) runs (each of which is B pages long)
- Step 3:
  - Sort each run by itself normally using your favorite algorithm
  - We can fit the entire run of B pages into our RAM so no problem
- Step 4:
  - Sort the runs amongst each other
  - We can merge B-1 runs at a time
    - B-1 pages for each run plus 1 page to store the output
    - Each run is larger than 1 page though!
      - Load the first (sorted) page of each run and once it's empty, read the next page
      - Similarly, write the output buffer each time we run out of space and keep going

output buffer

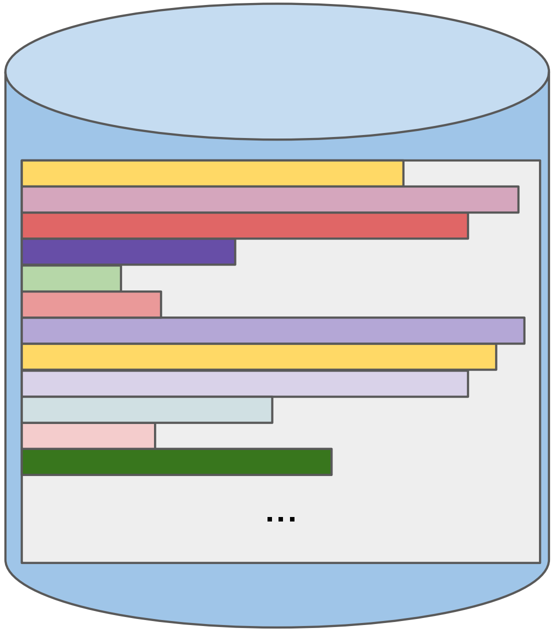
B-1

if all things sorted in this page, load next one

Step 1

- Have a dataset

a bar = one element



Suppose B = 4 and each page can hold 2 bars in full.

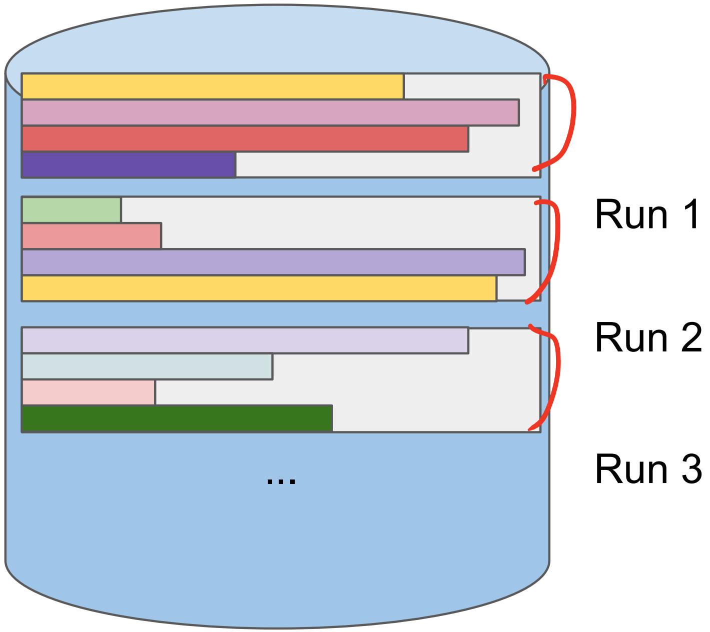
4×2 = 8 bars

Step 2

4 pages (8 bars) - long

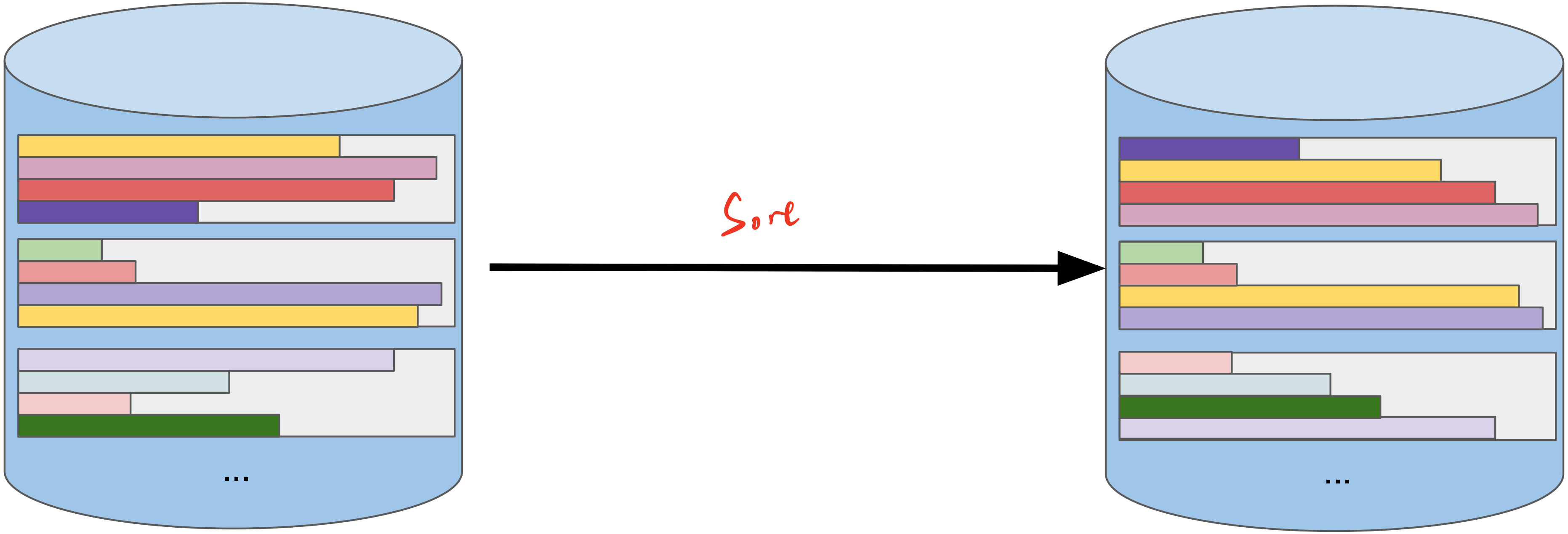
- Divide the data into ceiling(N/B) runs
  - Each is B pages long, i.e. each run is technically supposed to have 8 bars
  - (for simplicity we only show 4 smallest bars in each run)

Suppose B = 4 and each page can hold 2 bars in full.



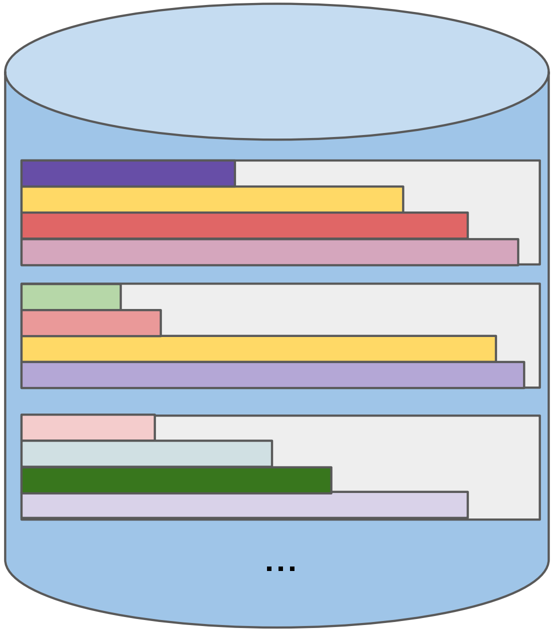
Step 3

- Sort each run individually (for simplicity we only show 4 smallest bars in each run)



Step 4

- Sort the runs with each other
  - B-1 runs at a time



(for simplicity we only show 4
smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

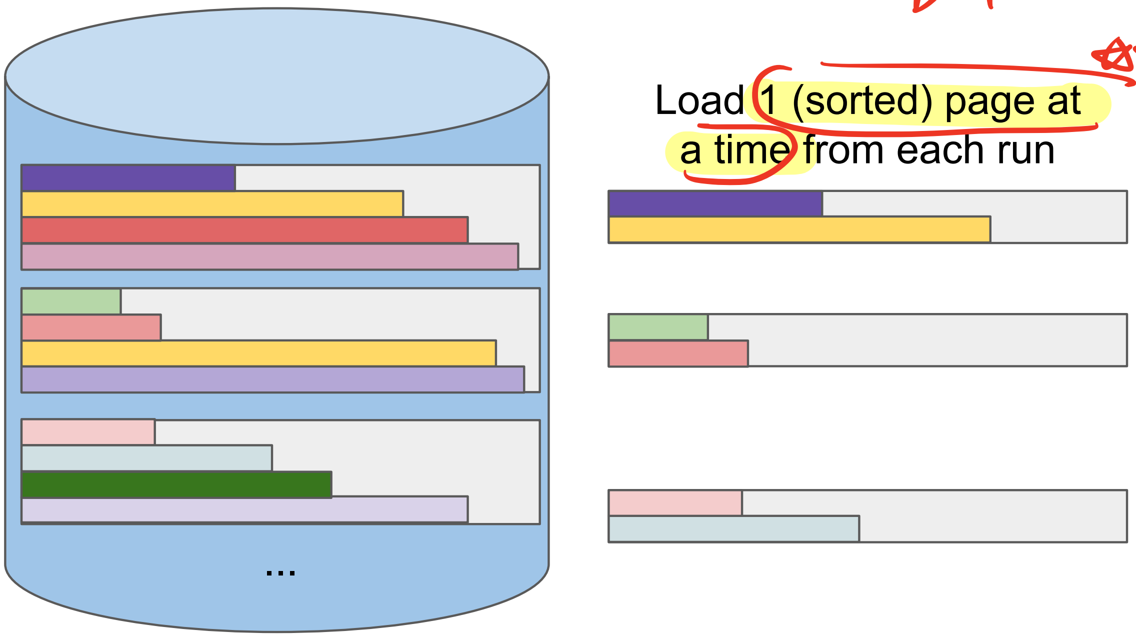
- Sort the runs with each other
  - B-1 runs at a time

B-1

Load 1 (sorted) page at a time from each run

1x2 = 2 bars

Single output page



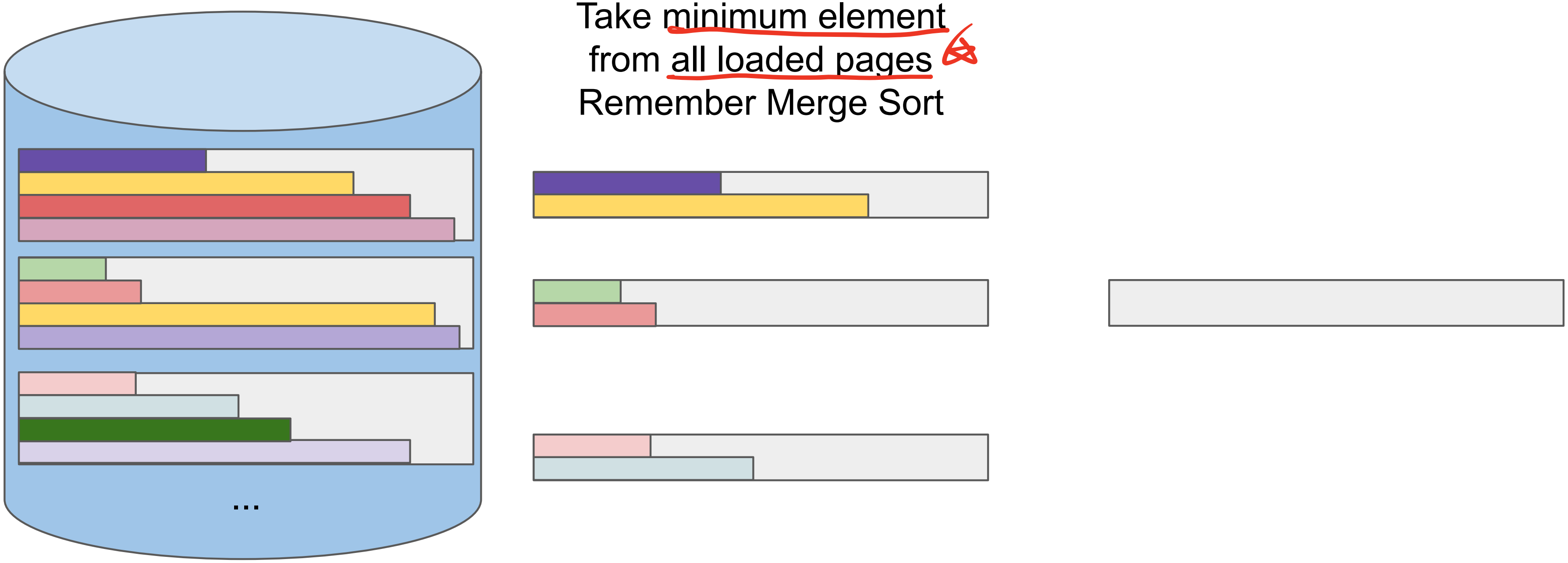
(for simplicity we only show 4 smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time

Take minimum element from all loaded pages
Remember Merge Sort

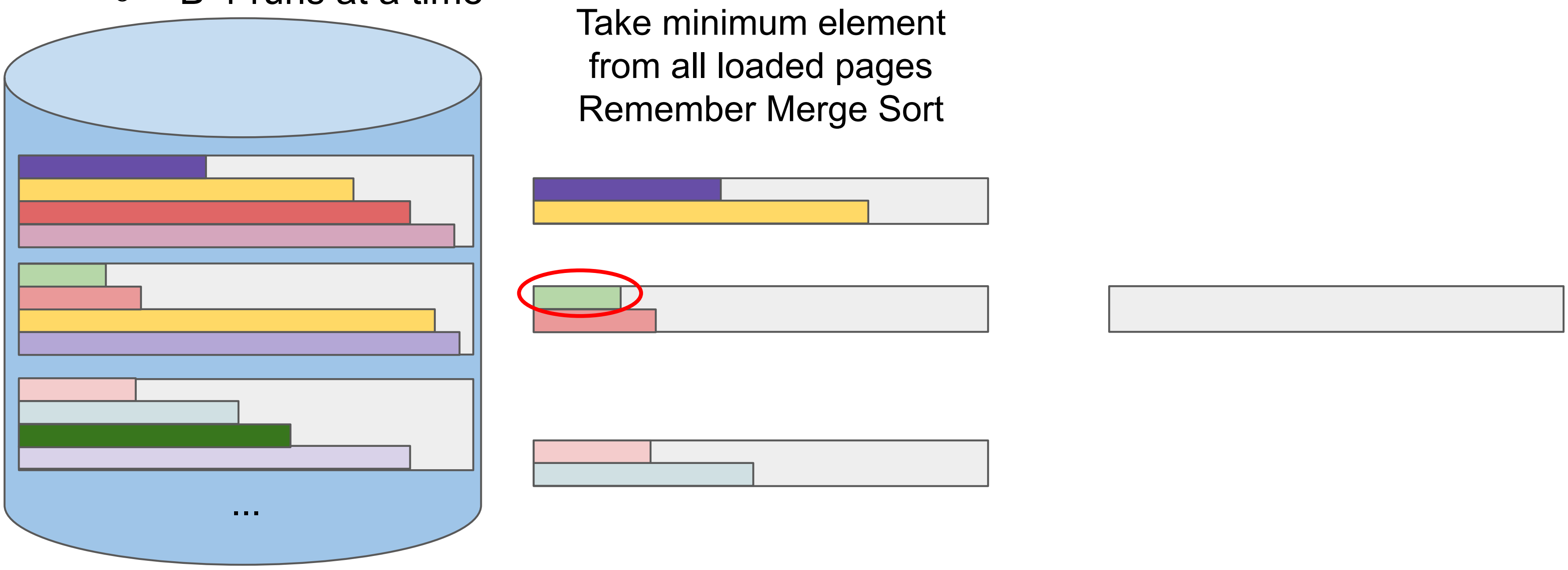


(for simplicity we only show 4 smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time



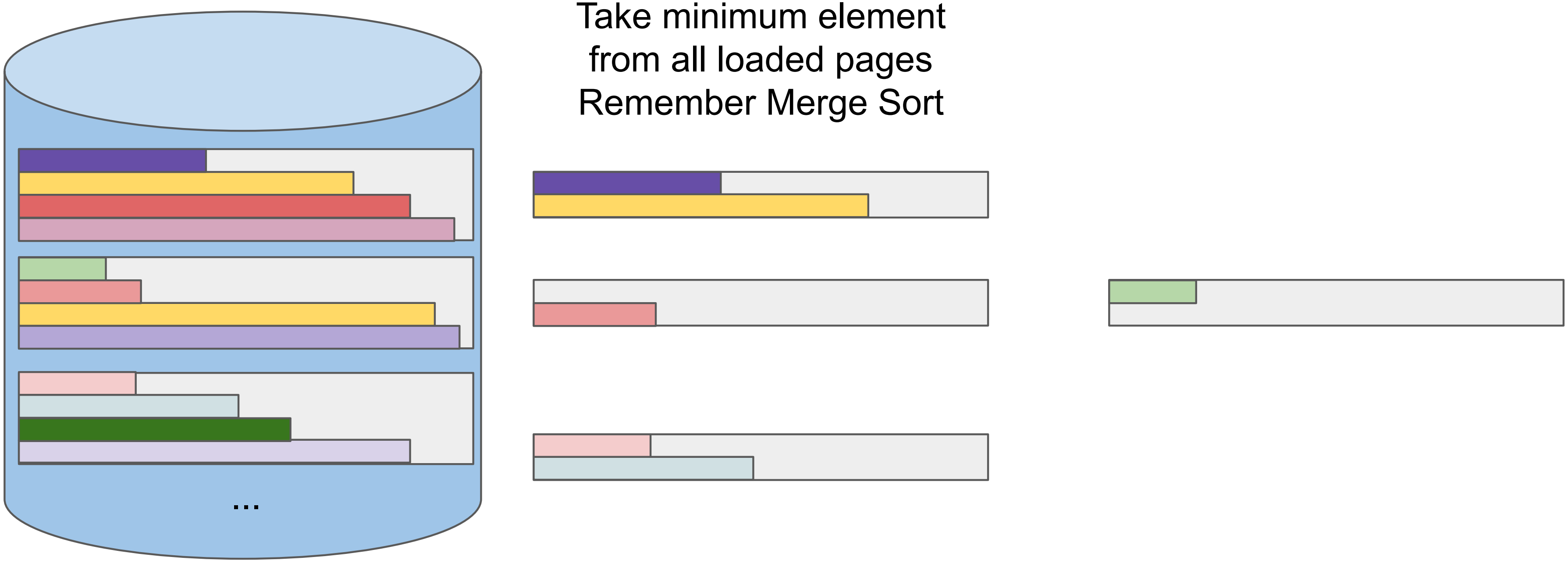
(for simplicity we only show 4 smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
    - B-1 runs at a time

Take minimum element from all loaded pages
Remember Merge Sort



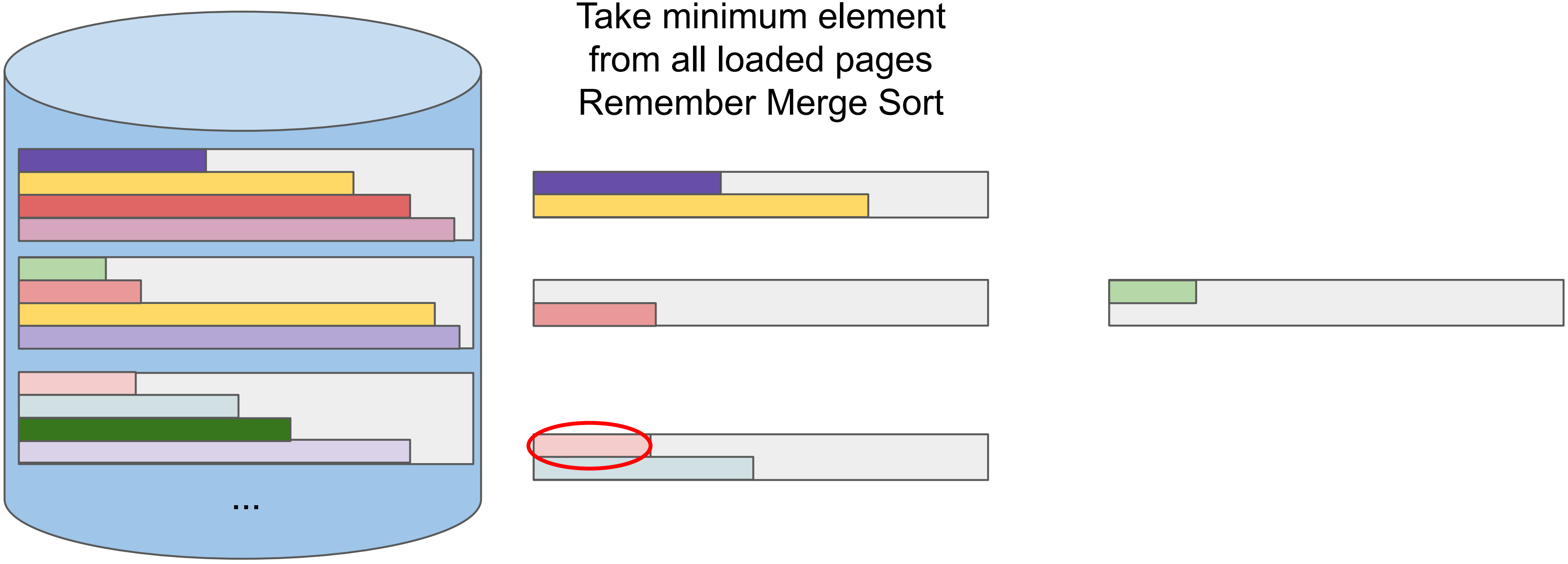
(for simplicity we only show 4 smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time

Take minimum element from all loaded pages
Remember Merge Sort

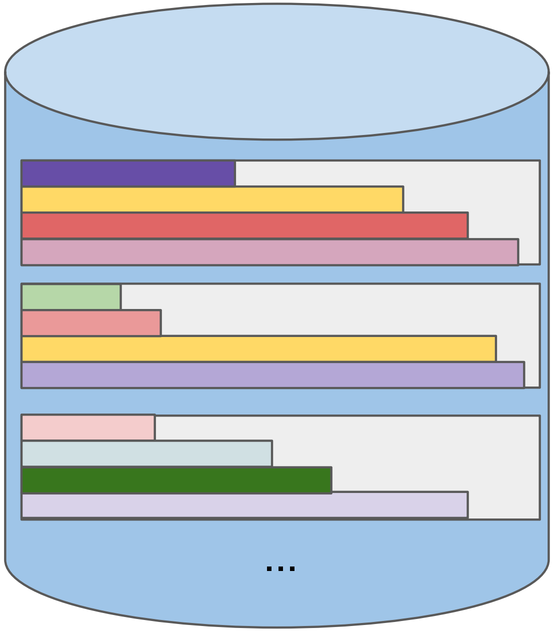


(for simplicity we only show 4 smallest bars in each run)

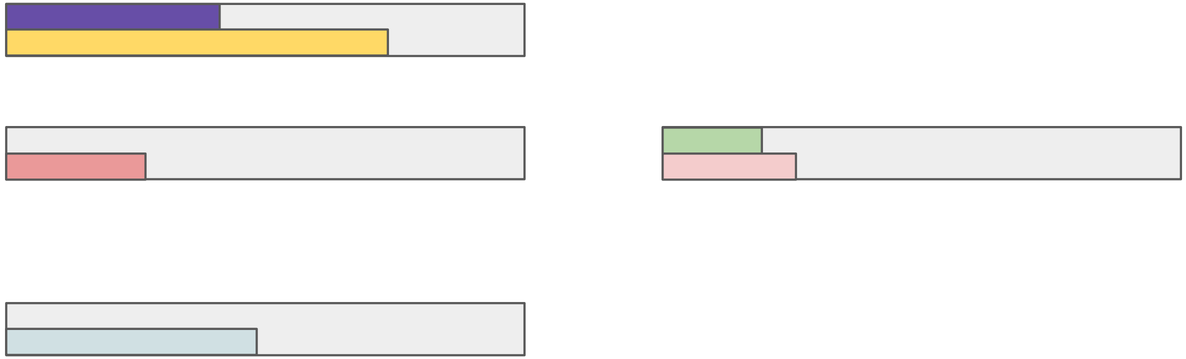
Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time



Take minimum element from all loaded pages
Remember Merge Sort

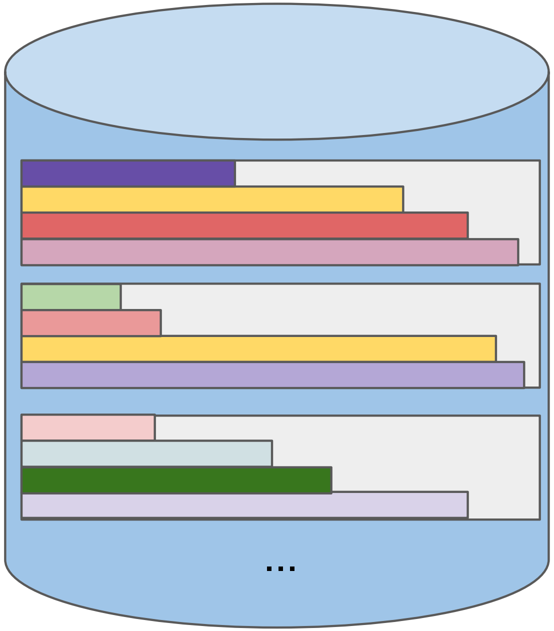


(for simplicity we only show 4 smallest bars in each run)

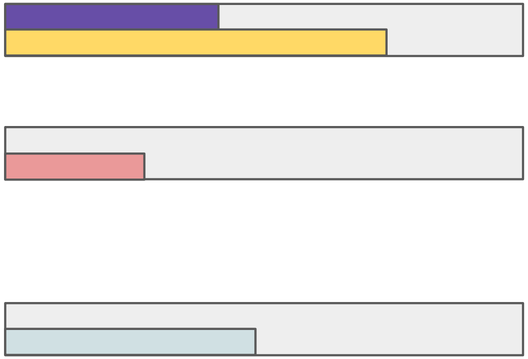
Suppose B = 4 and each page can hold 2 bars in full.

Step 4

- Sort the runs with each other
    - B-1 runs at a time



Take minimum element
from all loaded pages
Remember Merge Sort



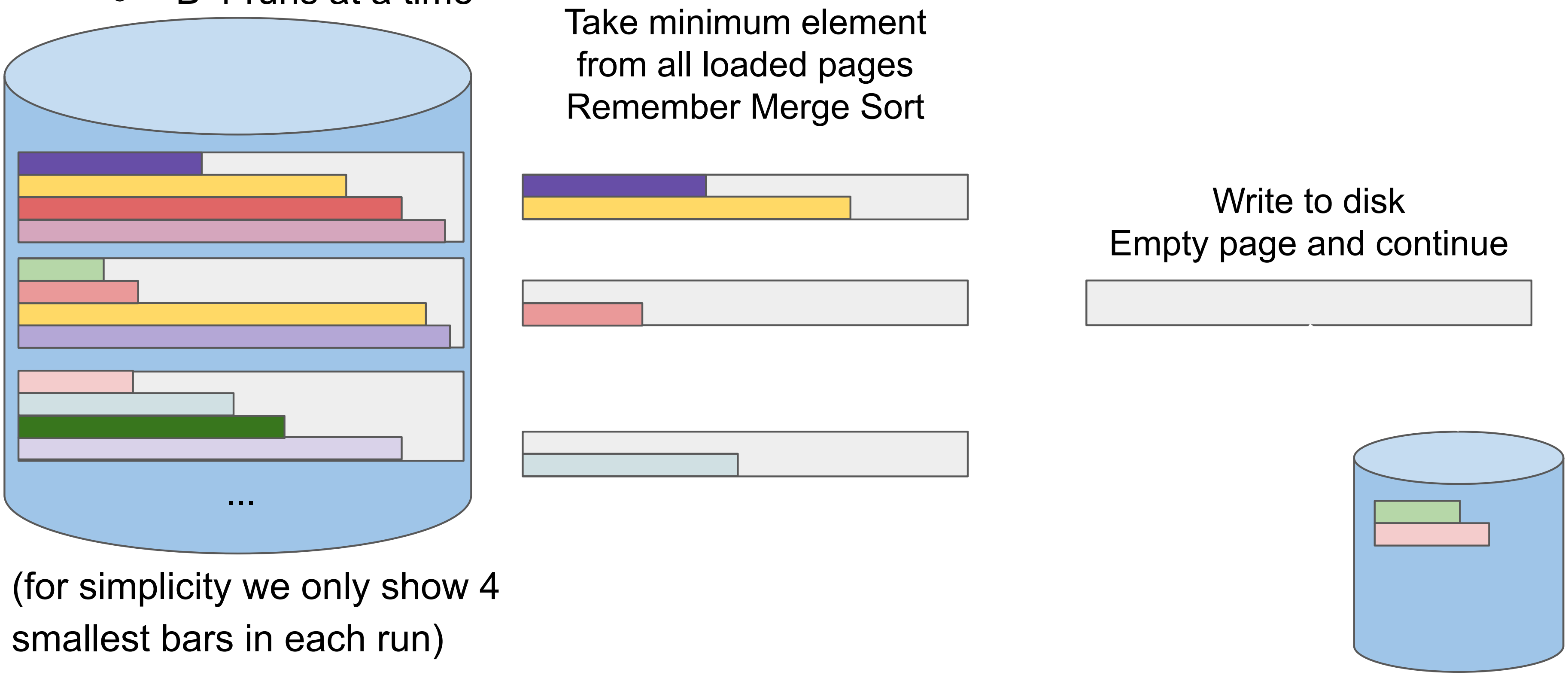
Output page full

(for simplicity we only show 4 smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time

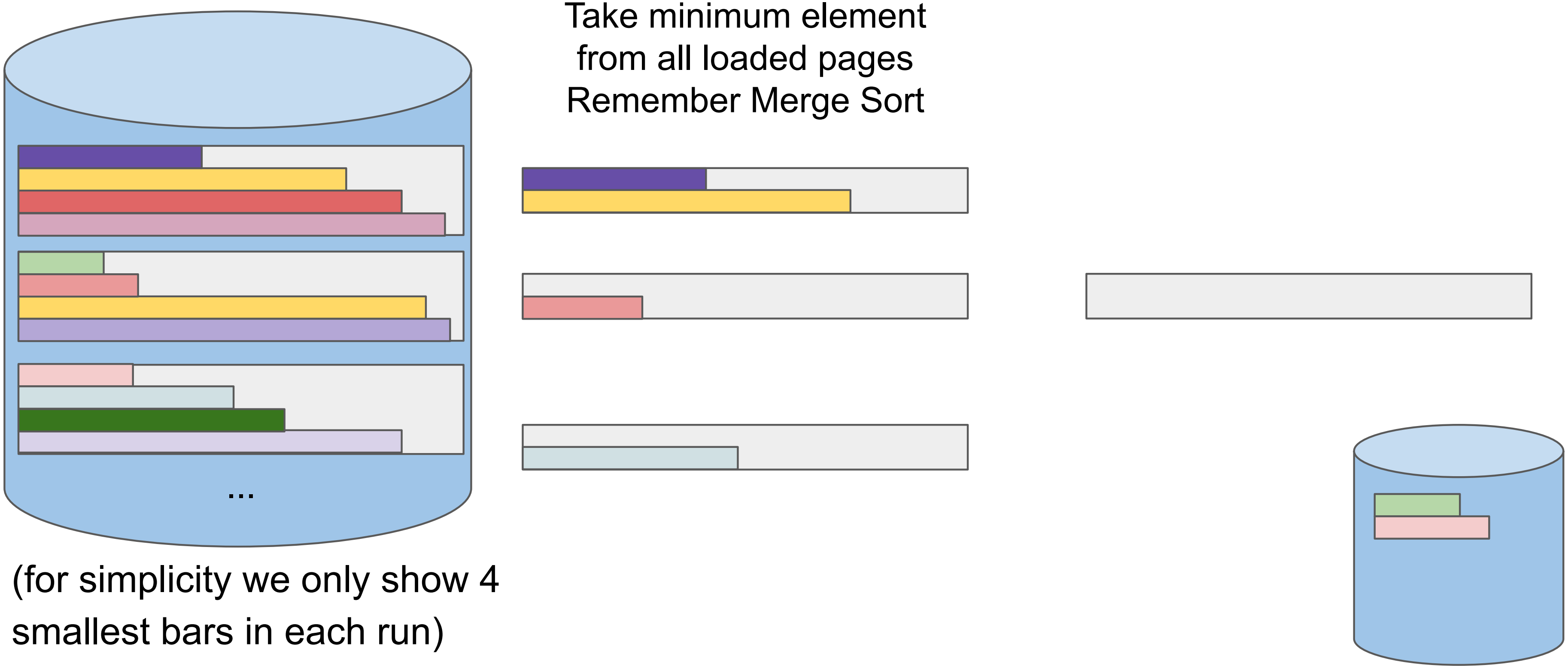


(for simplicity we only show 4 smallest bars in each run)

Step 4

Suppose B = 4 and
each page can hold
2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time



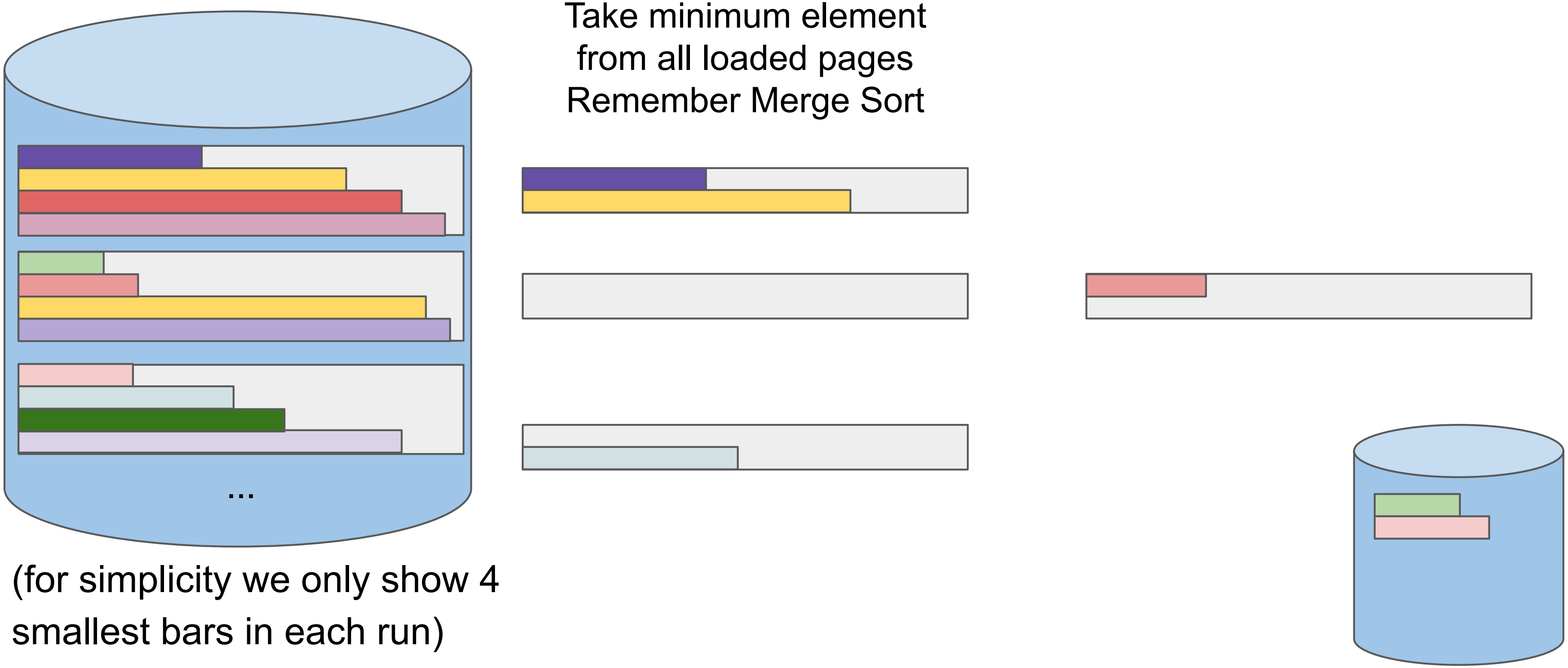
(for simplicity we only show 4
smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time

Take minimum element from all loaded pages
Remember Merge Sort



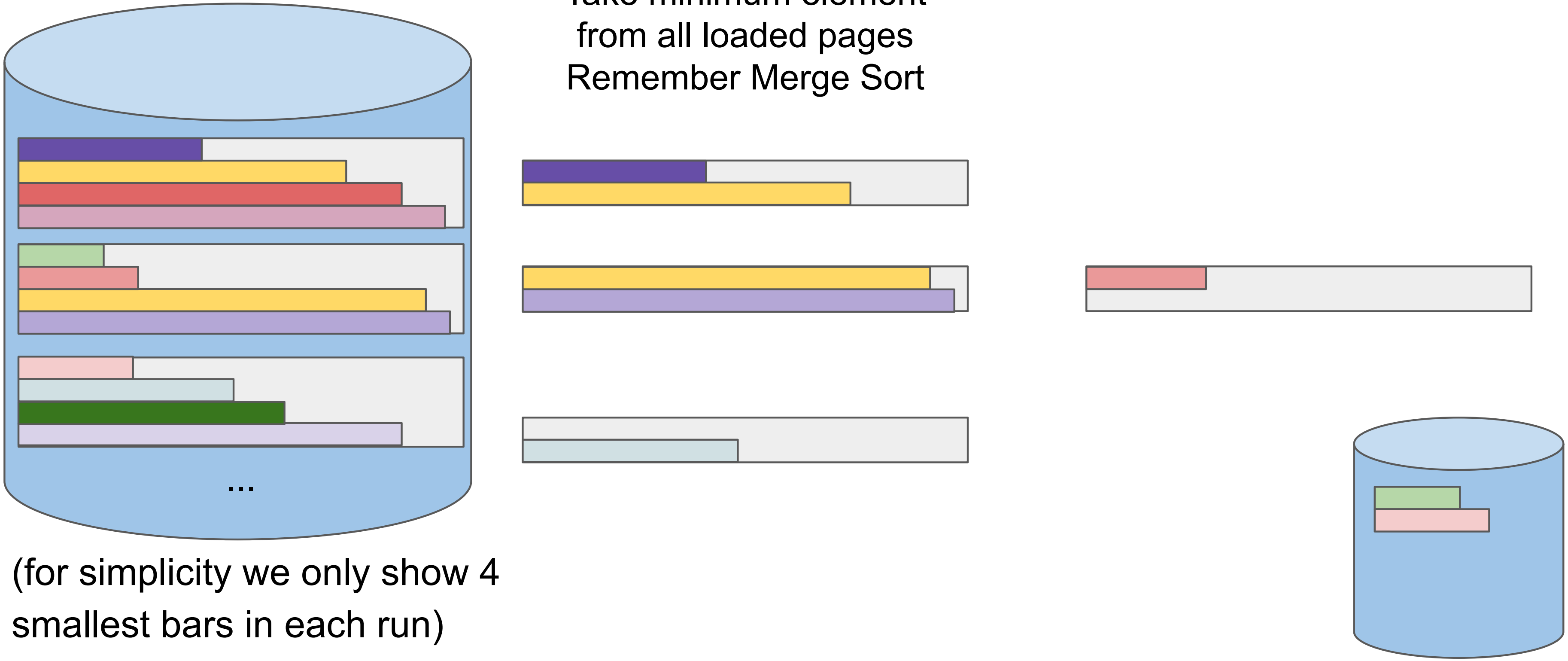
(for simplicity we only show 4 smallest bars in each run)

Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
  - B-1 runs at a time

Take minimum element from all loaded pages
Remember Merge Sort

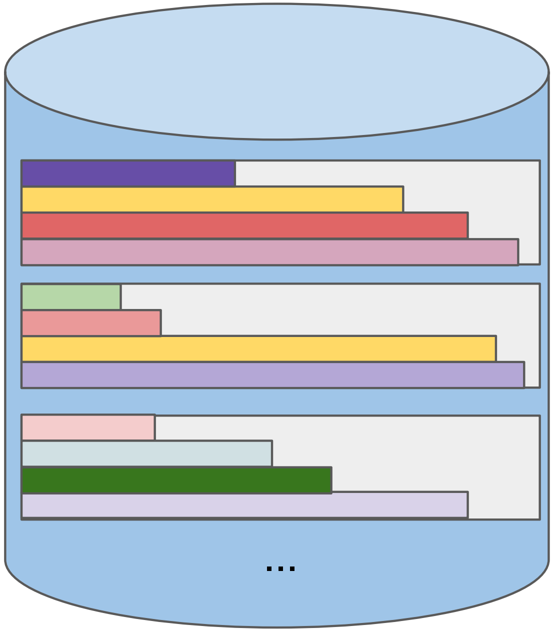


(for simplicity we only show 4 smallest bars in each run)

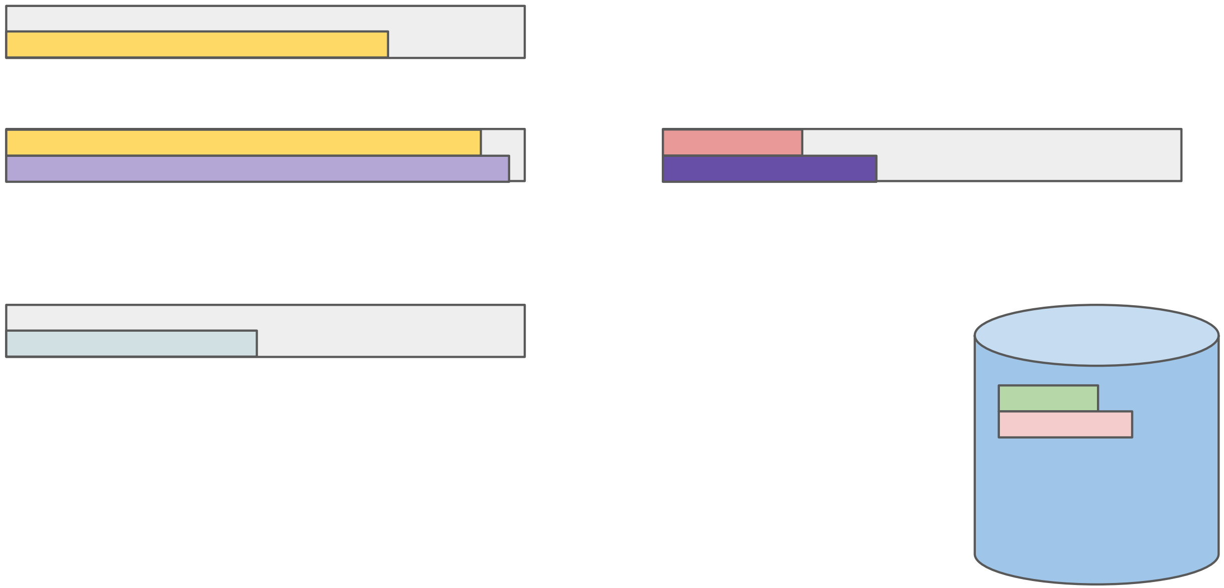
Step 4

Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
    - B-1 runs at a time



Take minimum element from all loaded pages
Remember Merge Sort



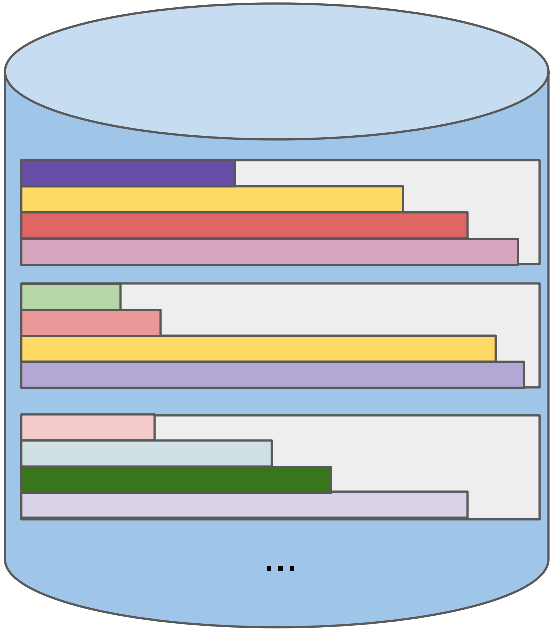
(for simplicity we only show 4 smallest bars in each run)

Step 4

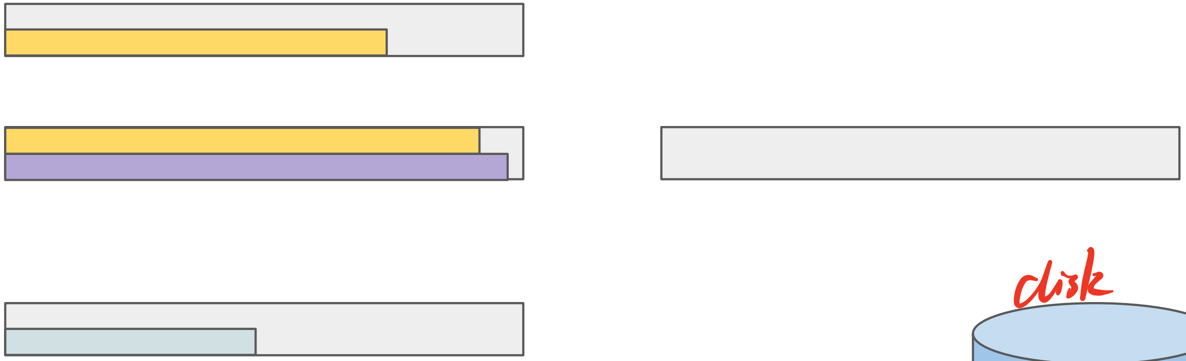
Suppose B = 4 and each page can hold 2 bars in full.

- Sort the runs with each other
    - B-1 runs at a time

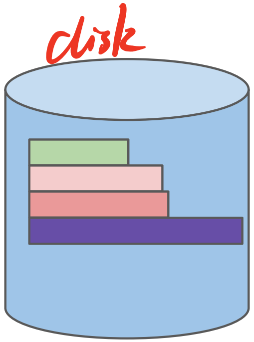
Continue till runs sorted
Will need to make more passes if more than B-1 runs



(for simplicity we only show 4 smallest bars in each run)



$9 \rightarrow 3 \rightarrow 1$



General External Merge Sort Math

merge cost

$$\left\lceil \log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil + 1$$

1 = pass for sorting each run at the beginning

- We have a dataset with N pages
  - We'll use B buffer pages
  - We'll have ceiling(N/B) runs initially
  - We need to make passes over the runs until entire dataset is sorted
    - We merge B-1 runs together at a time
    - That means we have ceiling(ceiling(N/B)/(B-1)) merged runs afterwards
    - Each time we make a pass we've merged all runs in sets of size B-1
    - We must continue to do this till we have 1 output dataset in sorted order
    - Takes 1+ceiling($\log_{B-1}$ceiling(N/B)) passes
  - Total IO cost is #passes * 2N → read/write to output pages
    - Each pass we read each page and write each page in a new sorted order

N read N write

Example

$N = 900$ $B = 18$.

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
  - How many passes will be required while performing a general external merge sort?

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
  - How many passes will be required while performing a general external merge sort?
  - N=900, B=18

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
  - How many passes will be required while performing a general external merge sort?
  - N=900, B=18
  - ceiling(N/B)=50

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
  - How many passes will be required while performing a general external merge sort?
  - N=900, B=18
  - ceiling(N/B)=50
  - B-1=17

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
    - How many passes will be required while performing a general external merge sort?
    - N=900, B=18
    - ceiling(N/B)=50
    - B-1=17
    - #passes = 1+ceiling($\log_{B-1}$(ceiling(N/B)))=1+ceiling($\log_{17}$(50))

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
  - How many passes will be required while performing a general external merge sort?
  - N=900, B=18
  - ceiling(N/B)=50
  - B-1=17
  - #passes = 1+ceiling($\log_{B-1}$(ceiling(N/B)))=1+ceiling($\log_{17}$(50))=1+ceiling(1.38ish)=1+2=3

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
  - How many passes will be required while performing a general external merge sort?
  - N=900, B=18
  - ceiling(N/B)=50
  - B-1=17
  - #passes = 1+ceiling($\log_{B-1}$(N/B))=1+ceiling($\log_{17}$(50))=1+ceiling(1.38ish)=1+2=3
  - How many IO operations?

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
    - How many passes will be required while performing a general external merge sort?
    - N=900, B=18
    - ceiling(N/B)=50
    - B-1=17
    - #passes = 1+ceiling($\log_{B-1}$(N/B))=1+ceiling($\log_{17}$(50))=1+ceiling(1.38ish)=1+2=3
    - How many IO operations?
    - #IO=2N*#passes

Example

- We have a large dataset of 900 pages. We are going to use 18 buffer pages
    - How many passes will be required while performing a general external merge sort?
    - N=900, B=18
    - ceiling(N/B)=50
    - B-1=17
    - #passes = 1+ceiling($\log_{B-1}$(N/B))=1+ceiling($\log_{17}$(50))=1+ceiling(1.38ish)=1+2=3
    - How many IO operations?
    - #IO=2N*#passes=2*900*3=5400

Replacement Sort

$2N \cdot \left(1 + \left\lceil \log_{B-1} \boxed{\# \text{ runs}} \right\rceil\right)$

lower this number

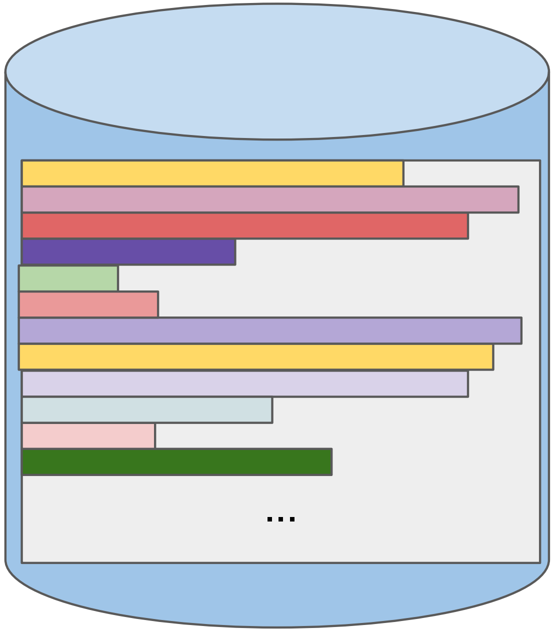
make each run longer

- We have lots and lots of runs
  - More runs → More passes → more IO cost
- What if we had longer runs?
  - Fewer passes!
- Replacement sort helps solve this
  - Let's minimize the number of runs in step 0
  - Rest of the sort is the same
  - We still have B pages
    - Set aside 1 for input and 1 for output
    - B-2 buffer pages for current set
    - Continually read input and add to current set (make sure current set is full at all times)
    - Output smallest element in current set that is larger than largest value in output set
      - Write output buffer to disk when full
      - When no such element exists, end the run and start a new run
    - Continue till everything is in a run. Then continue with external merge sort as normal

Applies to initial pass when creating runs

Step 1

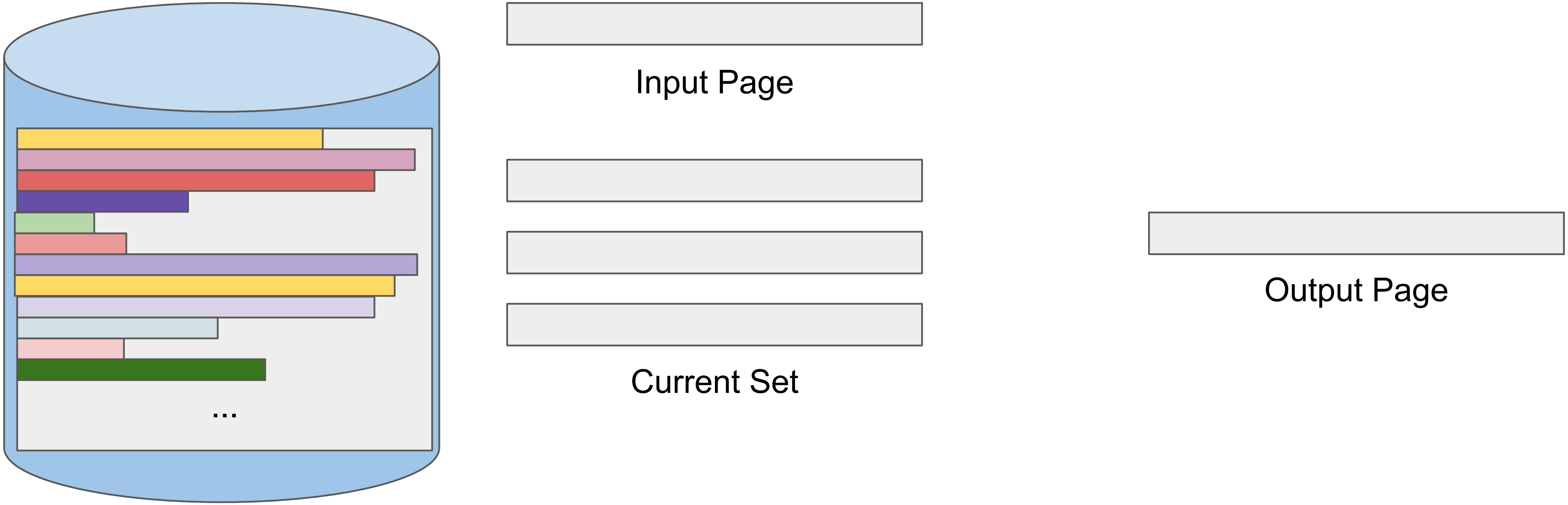
- Have a dataset



Step 2

Suppose B = 5 and each page can hold 2 bars in full.

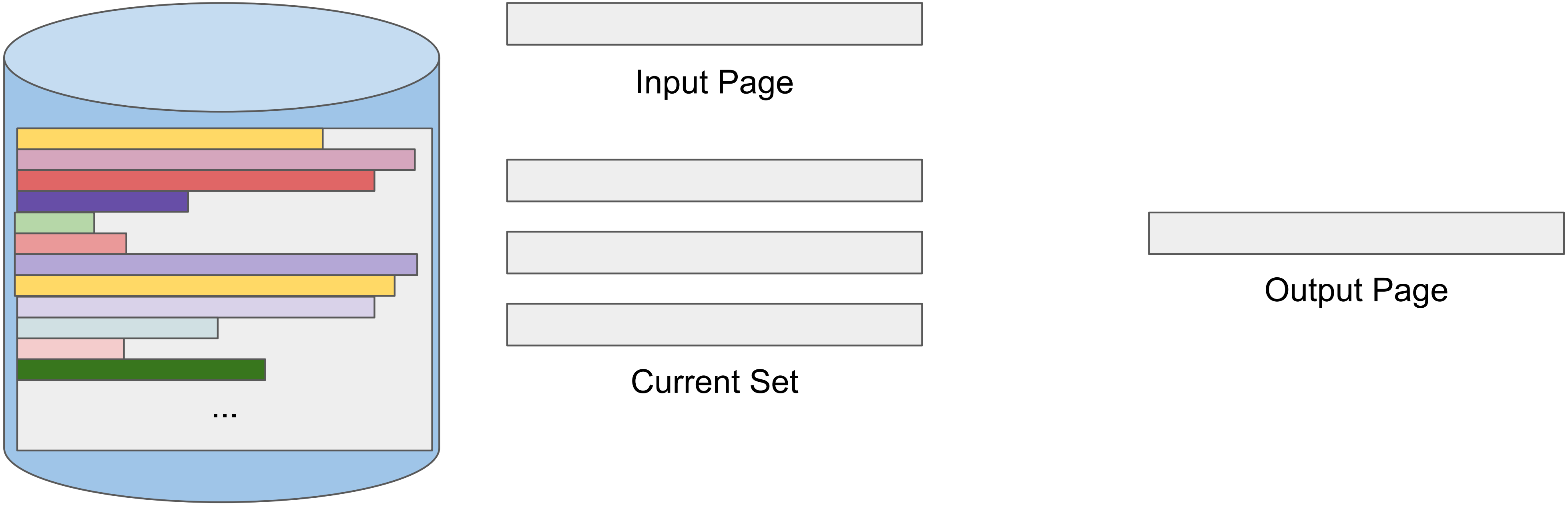
- Allocate pages for current set, input, and output pages



Step 3

Suppose B = 5 and each page can hold 2 bars in full.

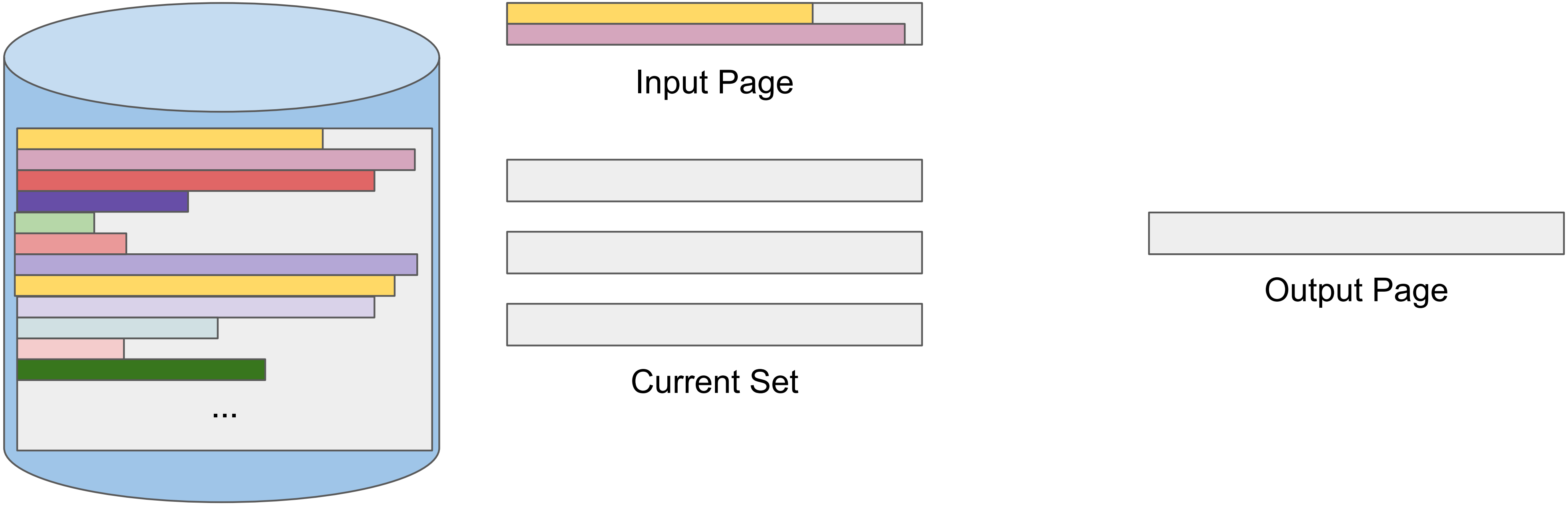
- Read data into current set using input page



Step 3

Suppose B = 5 and
each page can hold
2 bars in full.

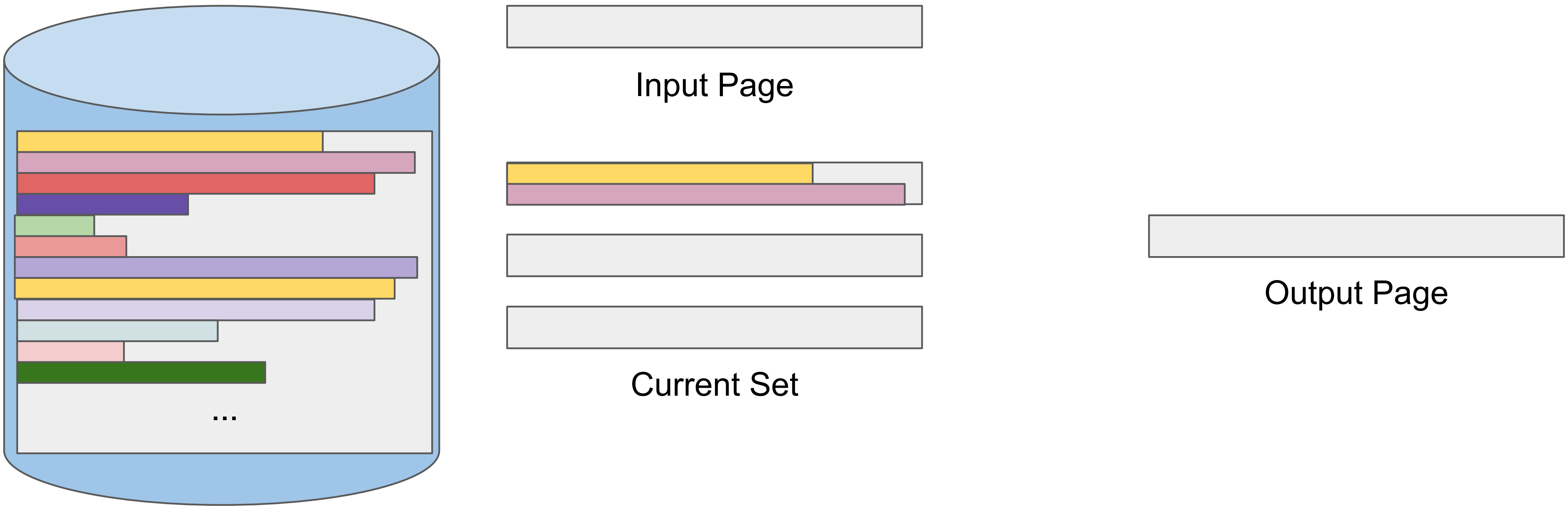
- Read data into current set using input page



Step 3

Suppose B = 5 and each page can hold 2 bars in full.

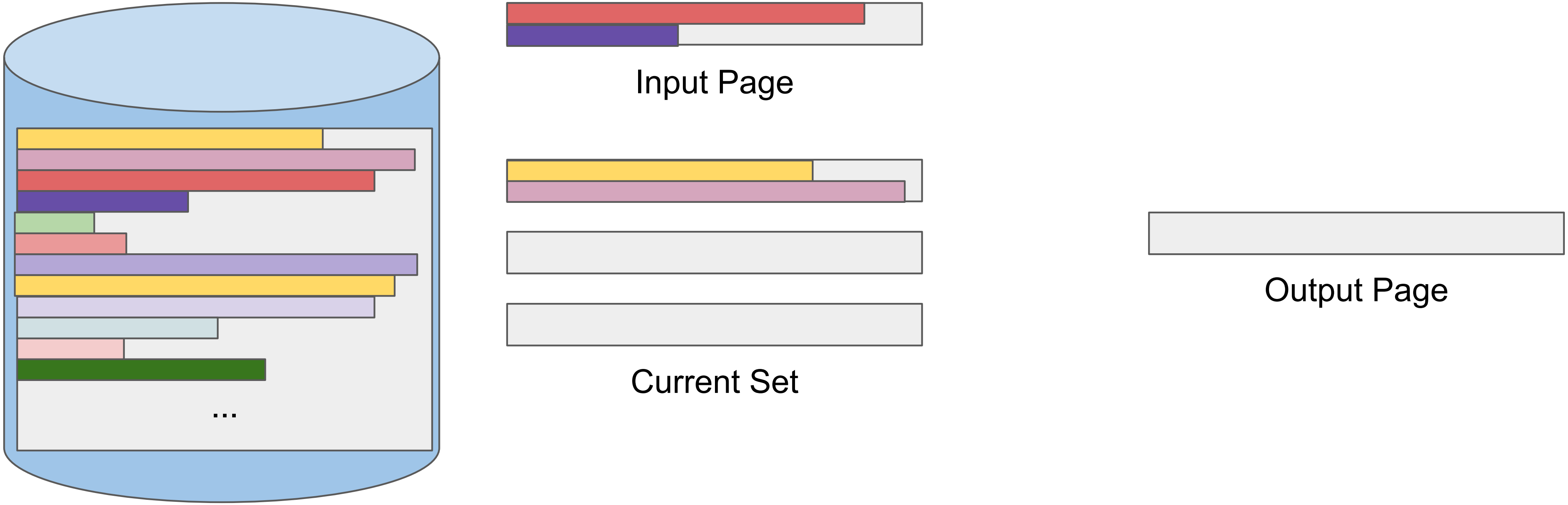
- Read data into current set using input page



Step 3

Suppose B = 5 and each page can hold 2 bars in full.

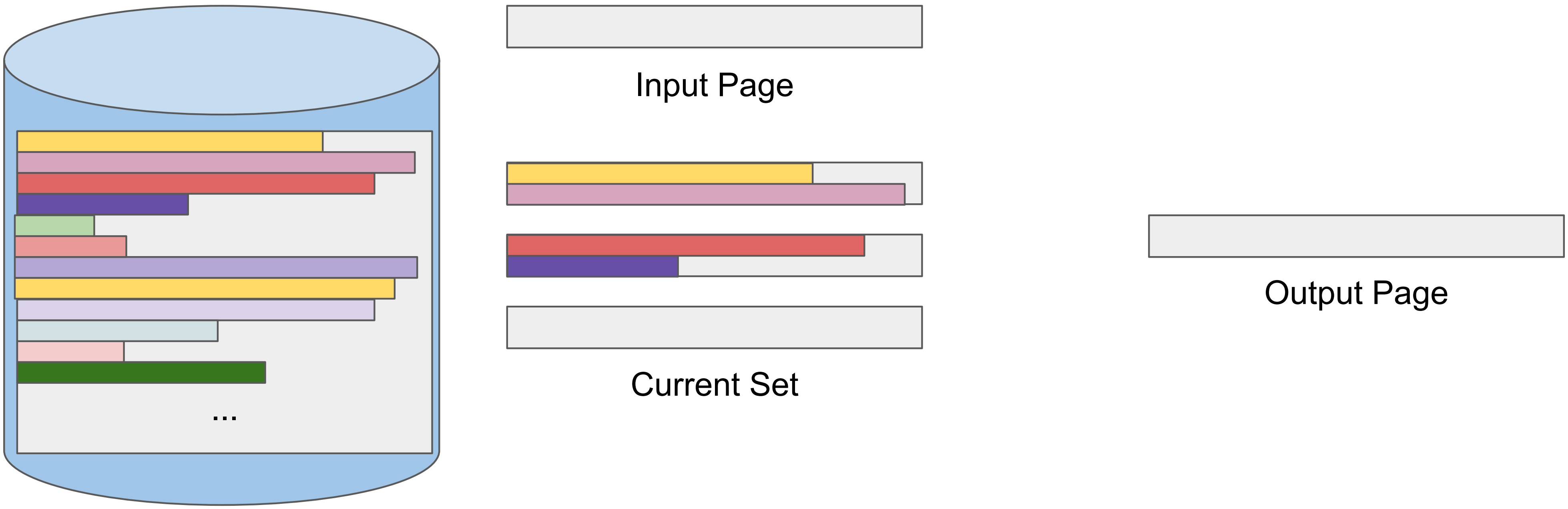
- Read data into current set using input page



Step 3

Suppose B = 5 and each page can hold 2 bars in full.

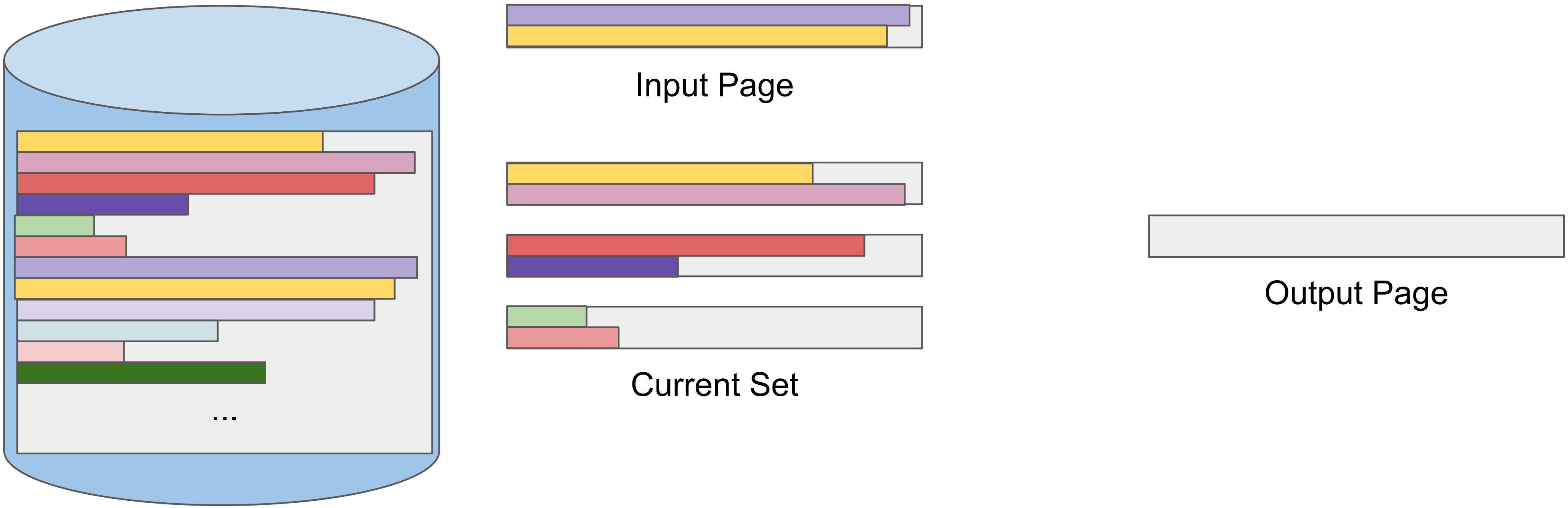
- Read data into current set using input page



Step 3

Suppose B = 5 and each page can hold 2 bars in full.

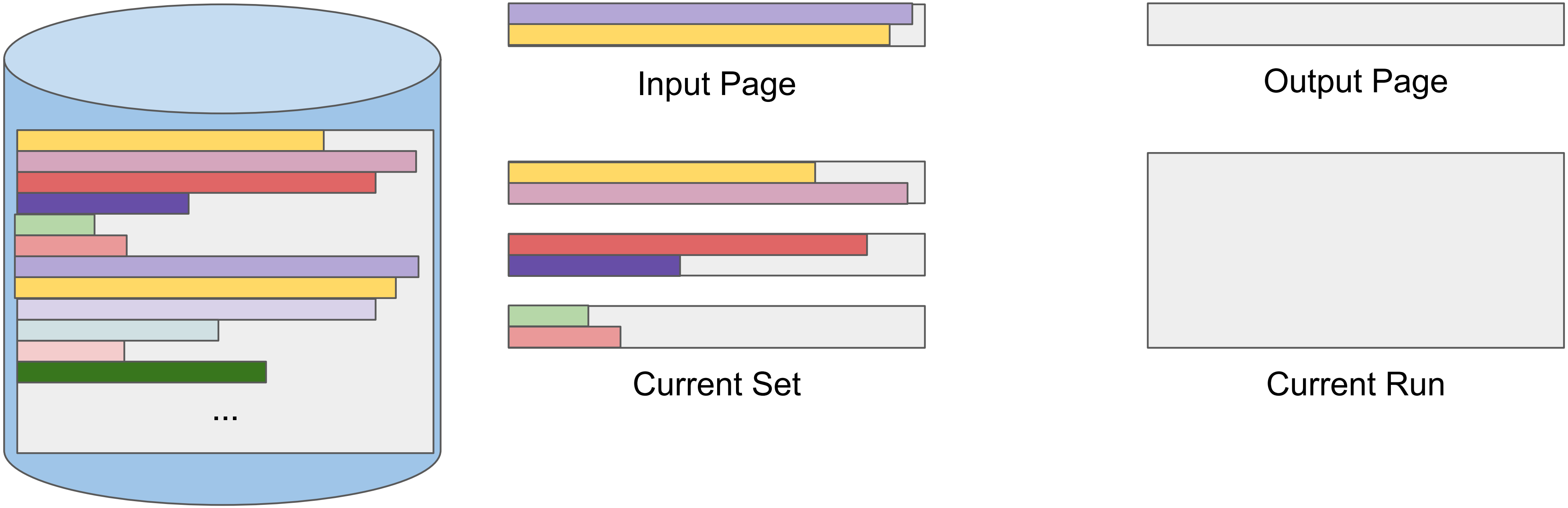
- Read data into current set using input page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

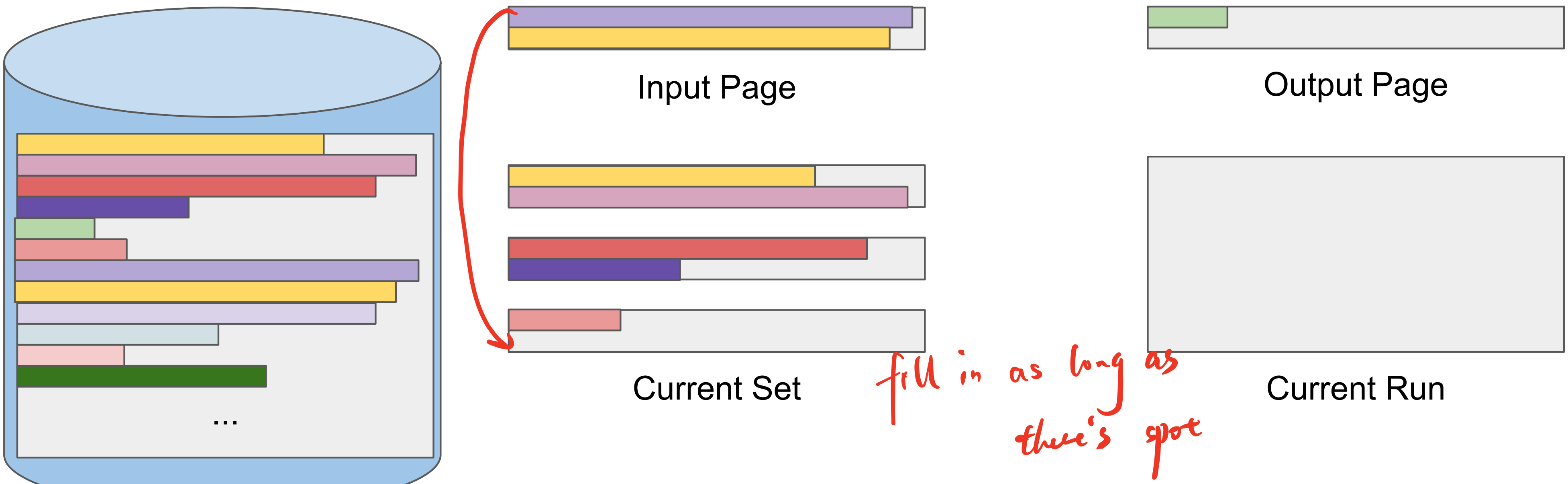
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

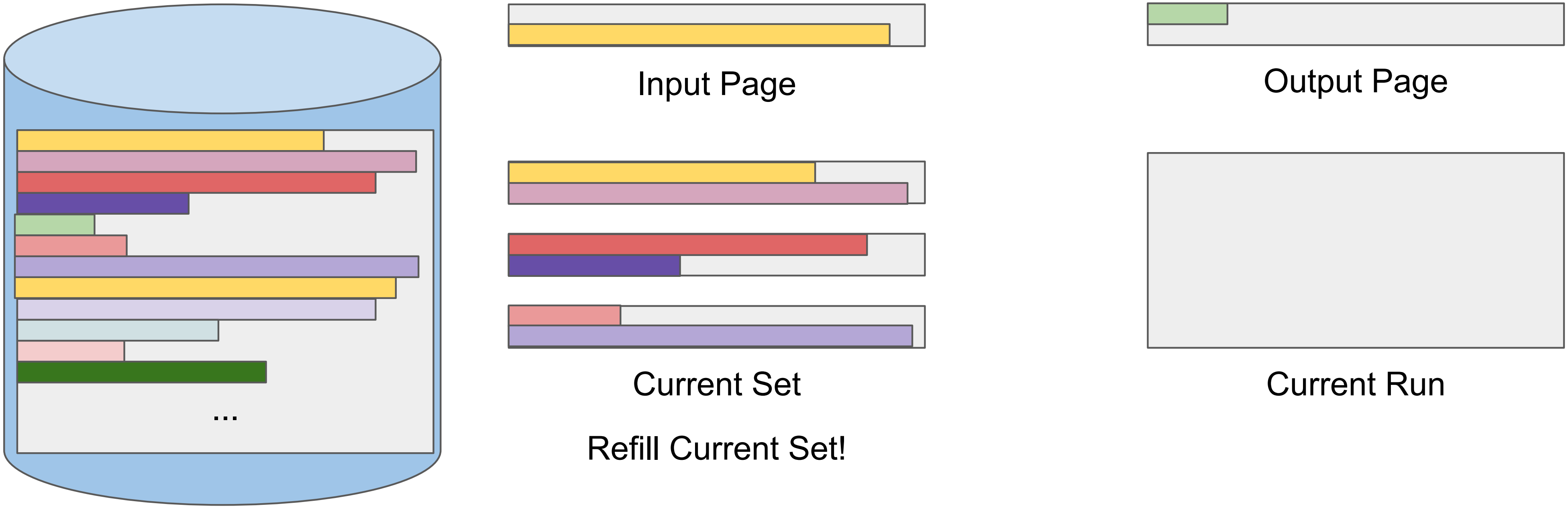
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and
each page can hold
2 bars in full.

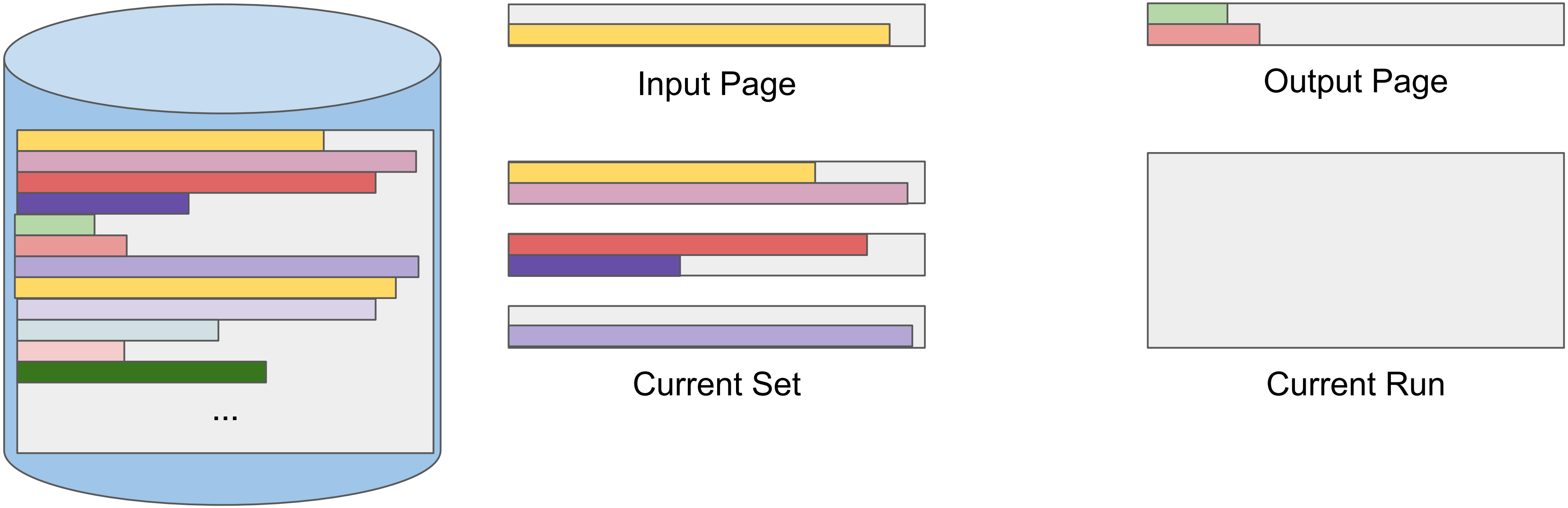
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

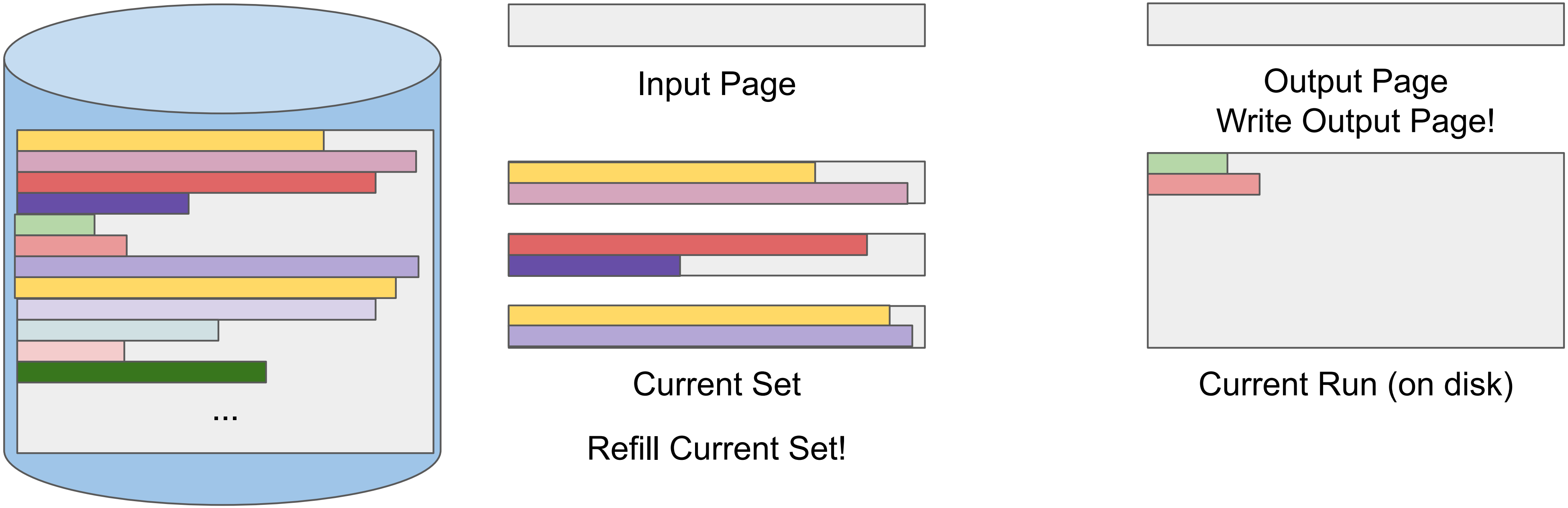
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

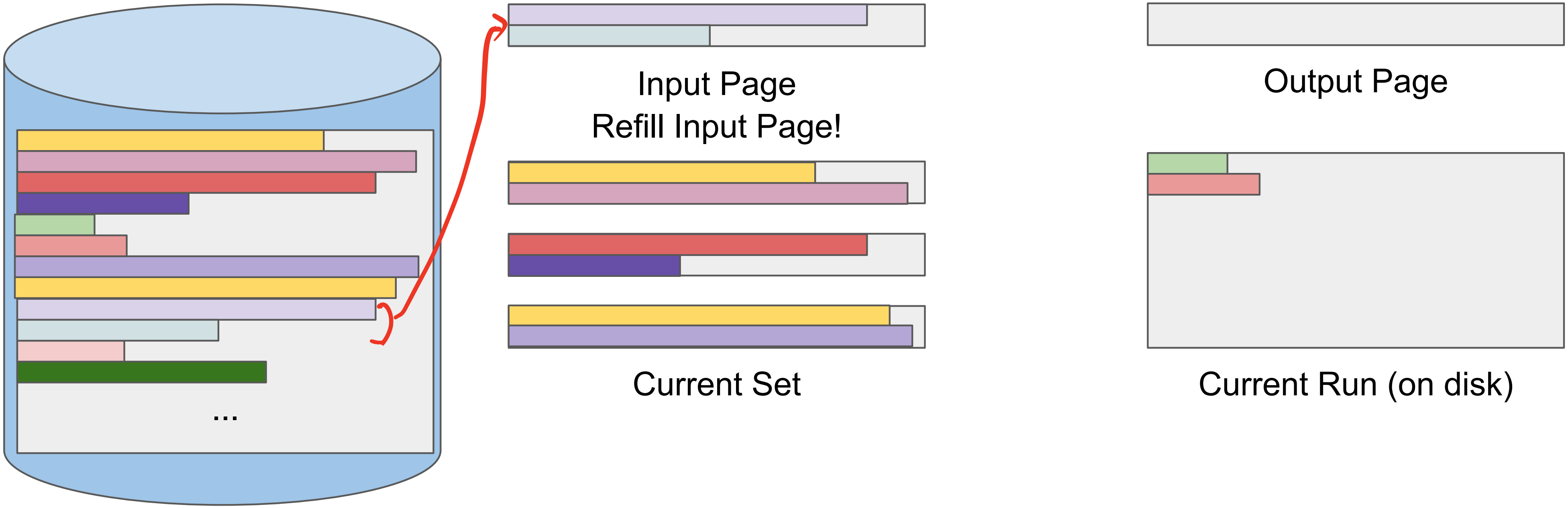
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

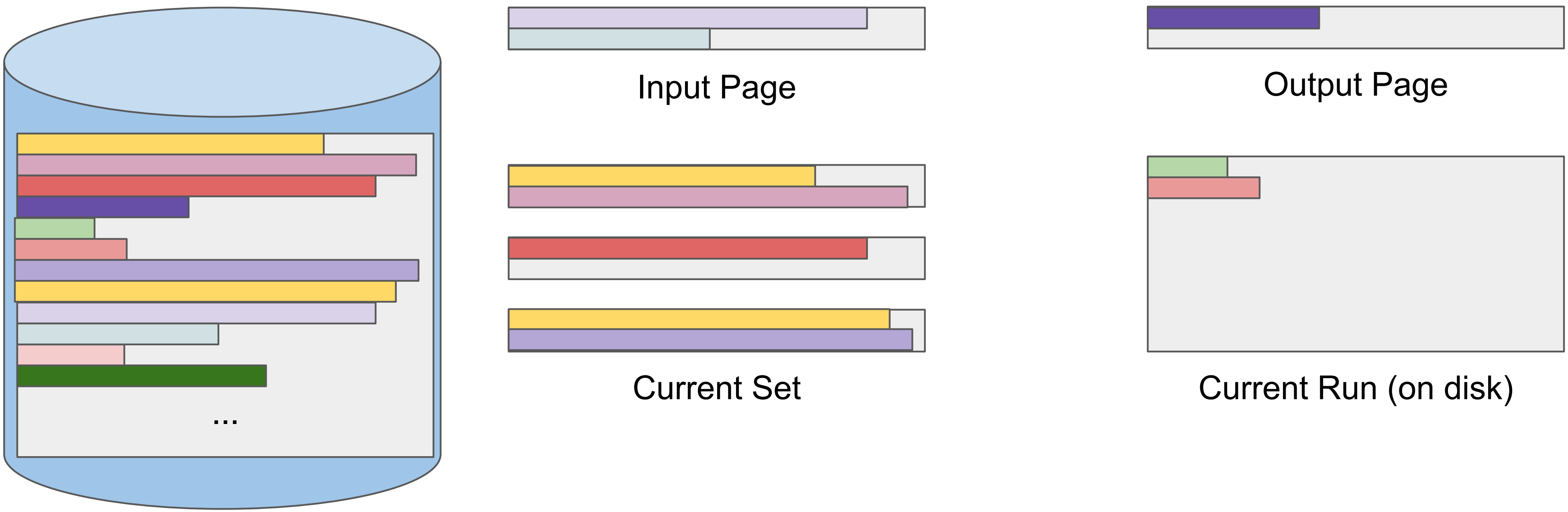
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

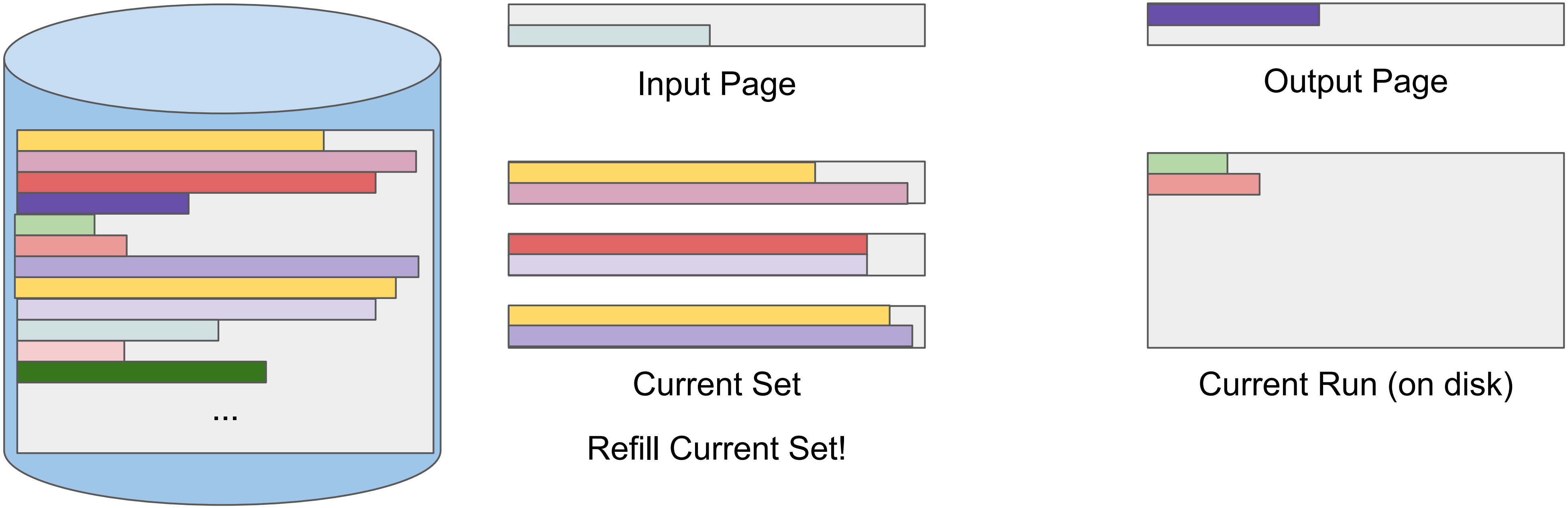
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

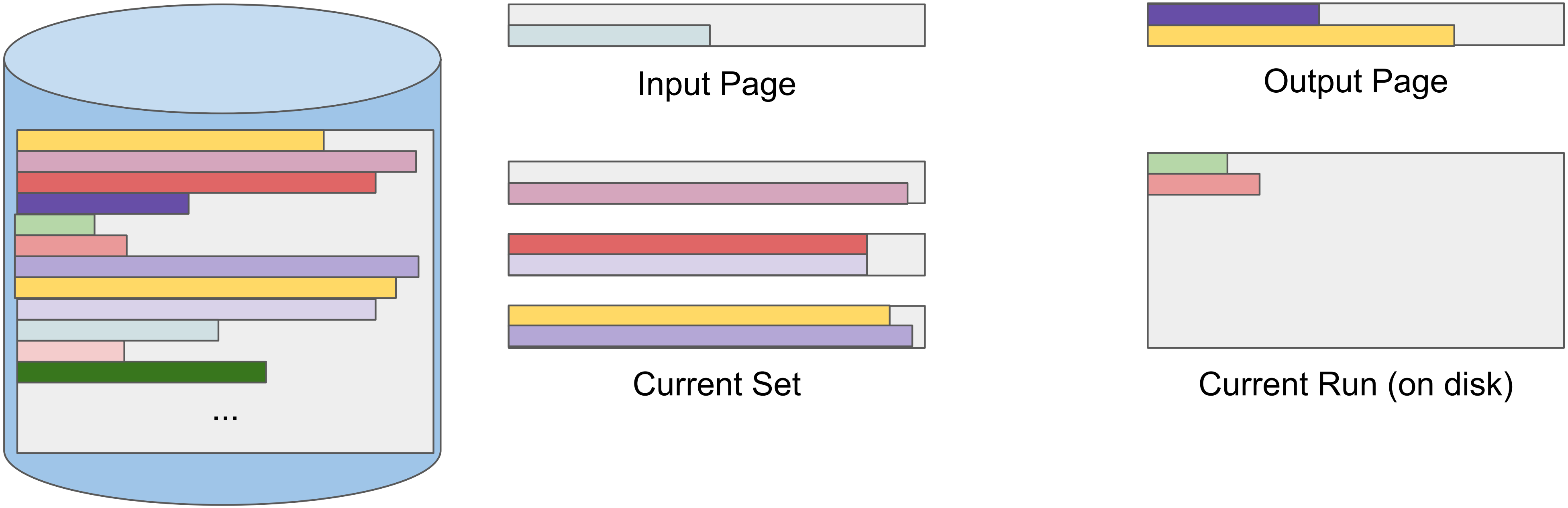
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

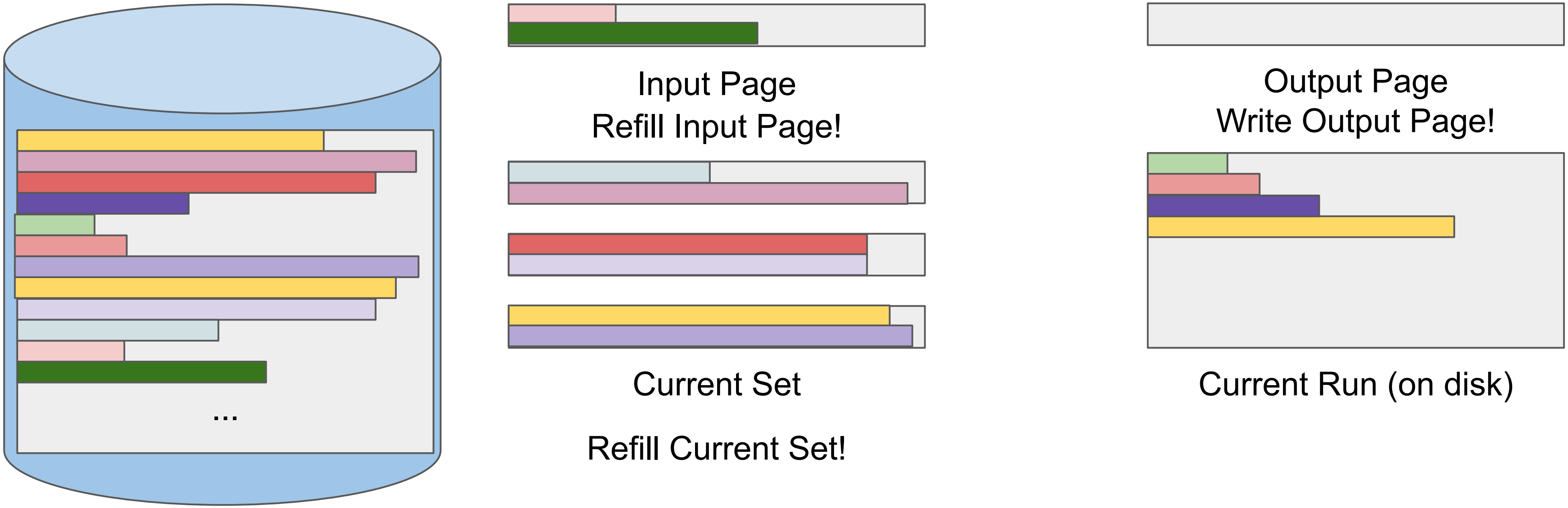
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

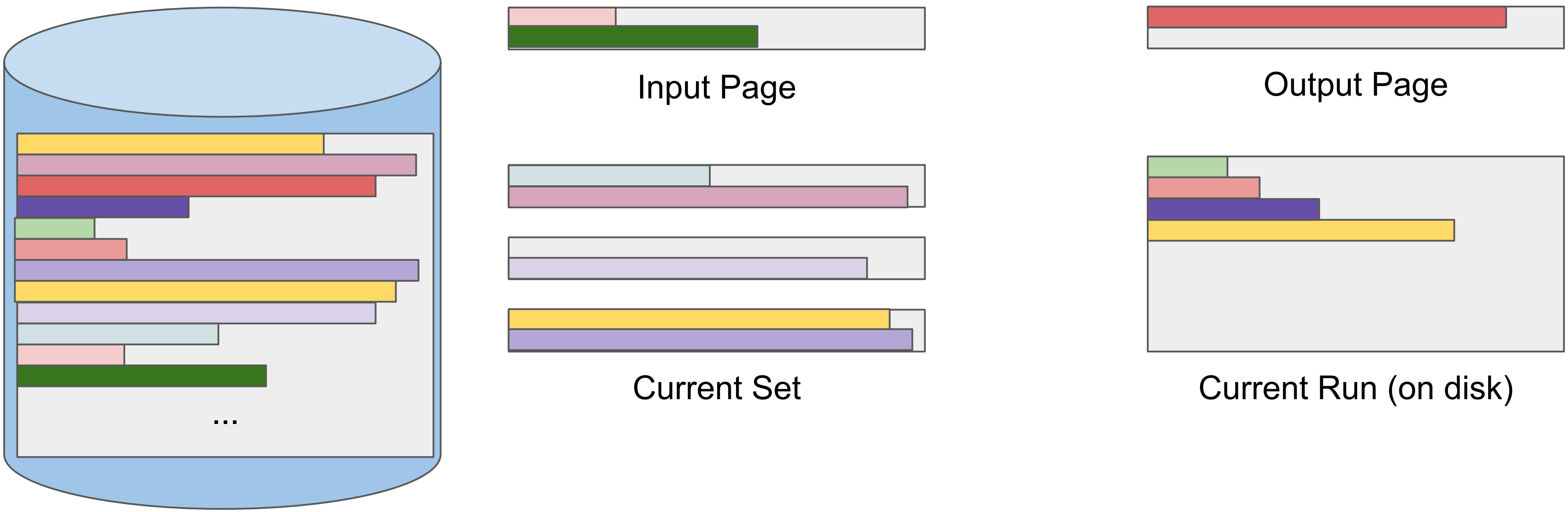
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

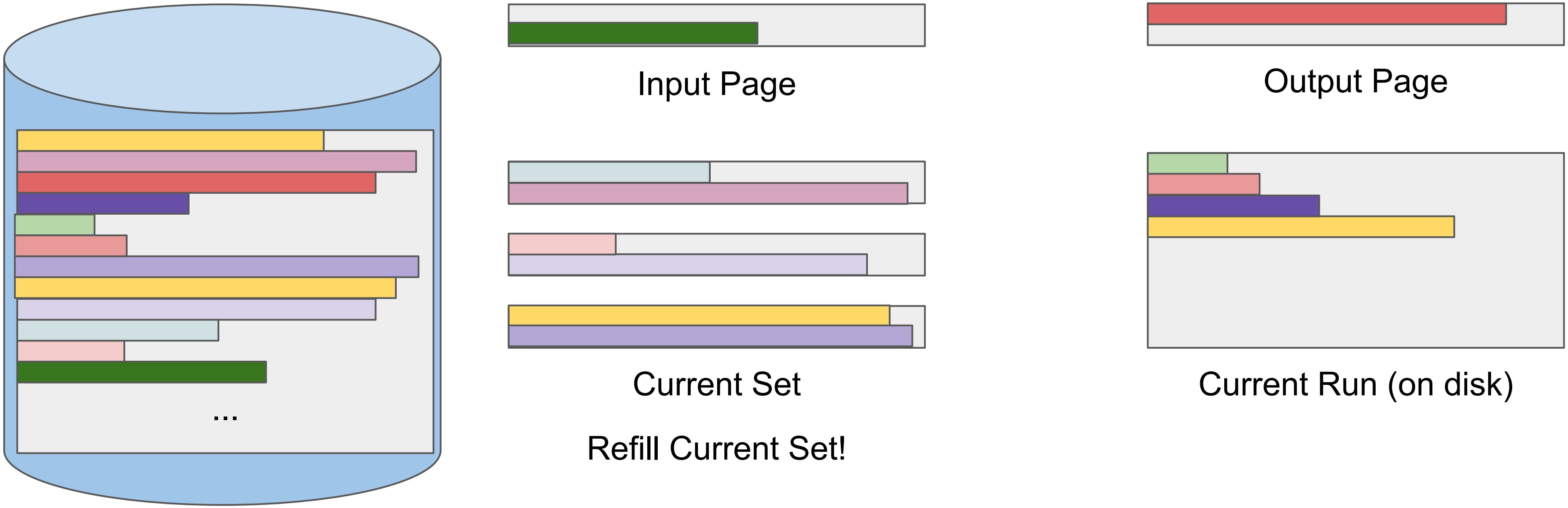
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

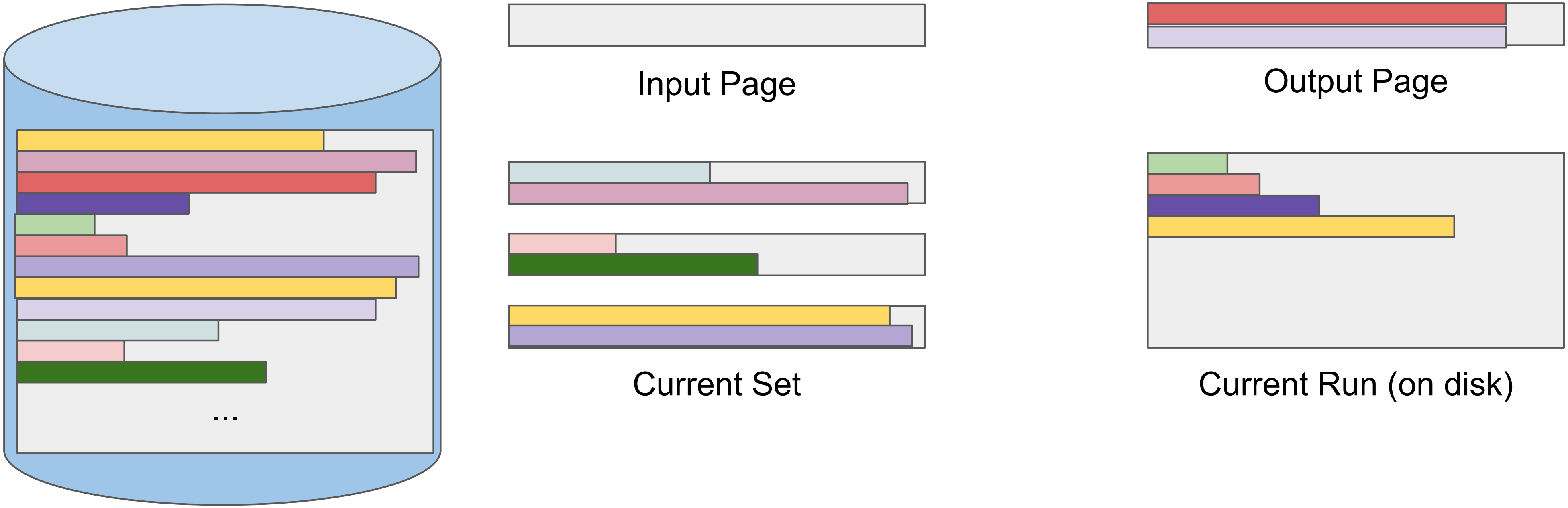
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

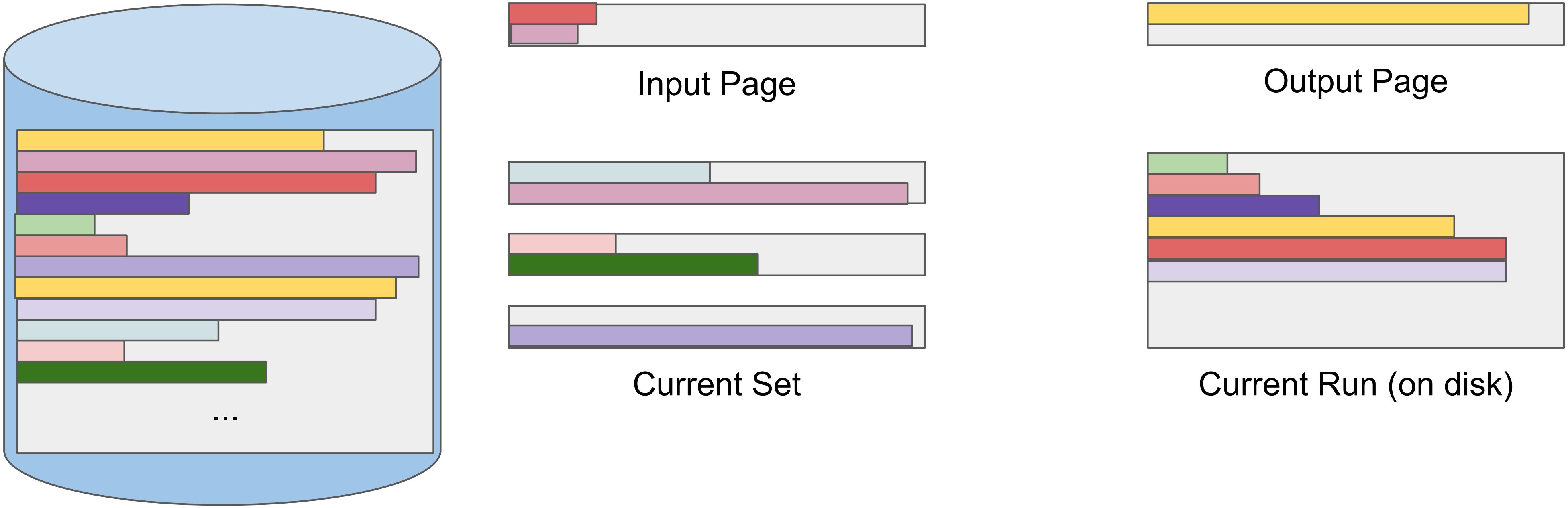
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

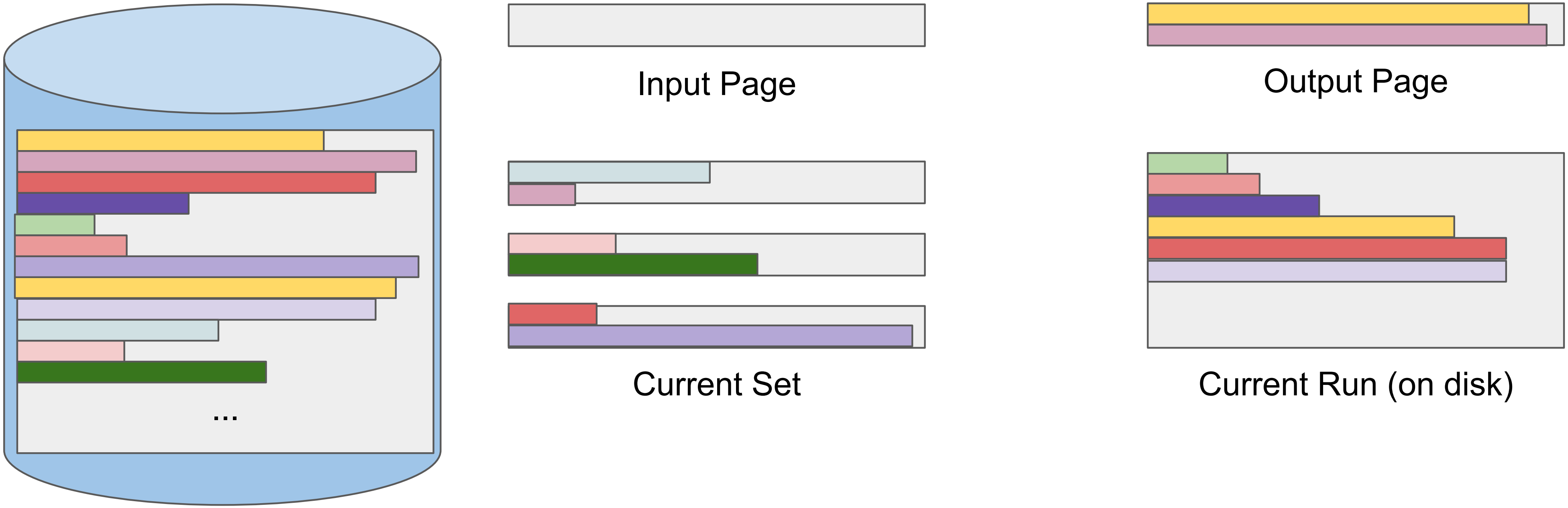
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

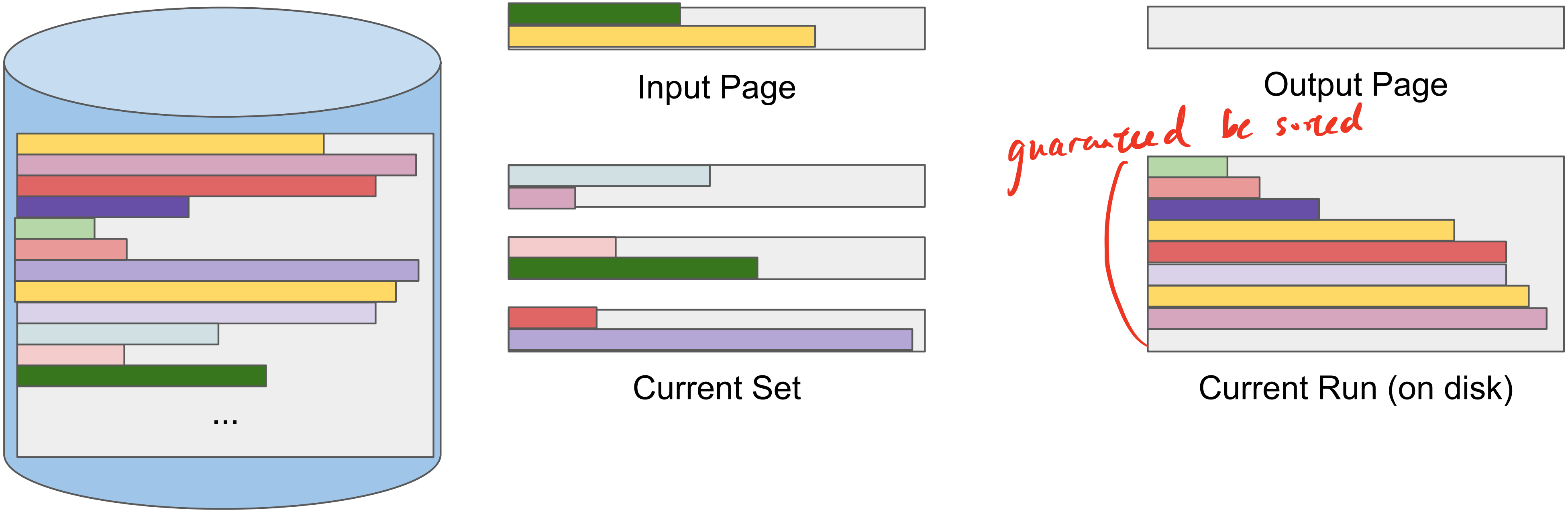
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

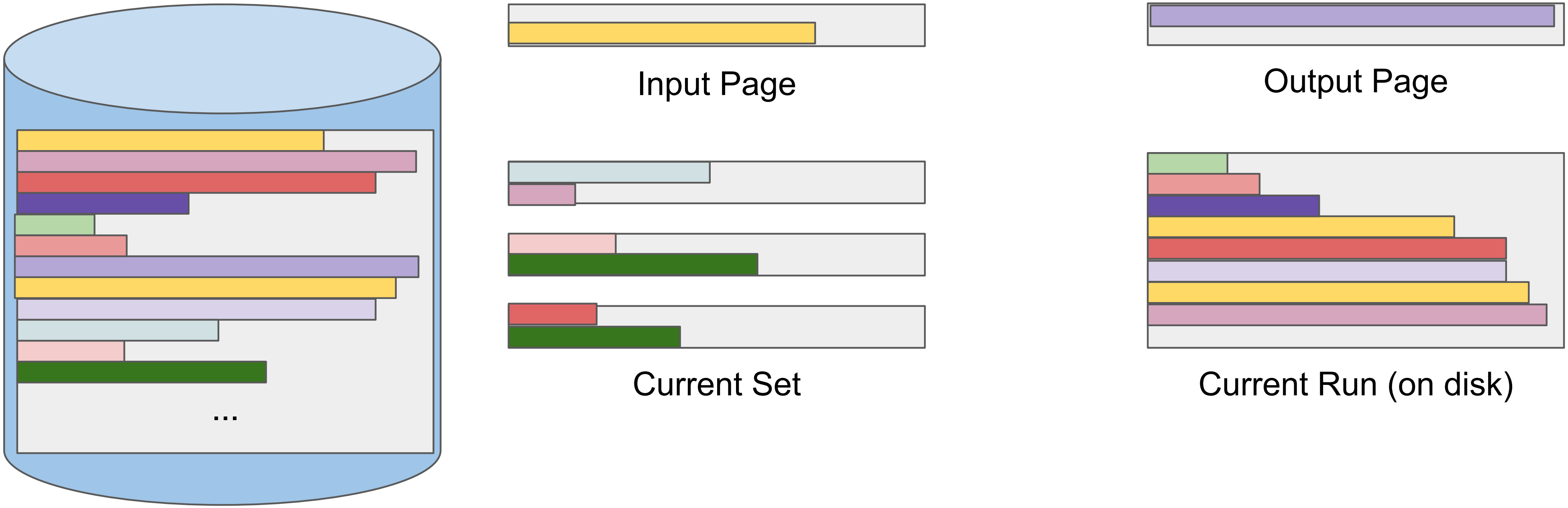
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

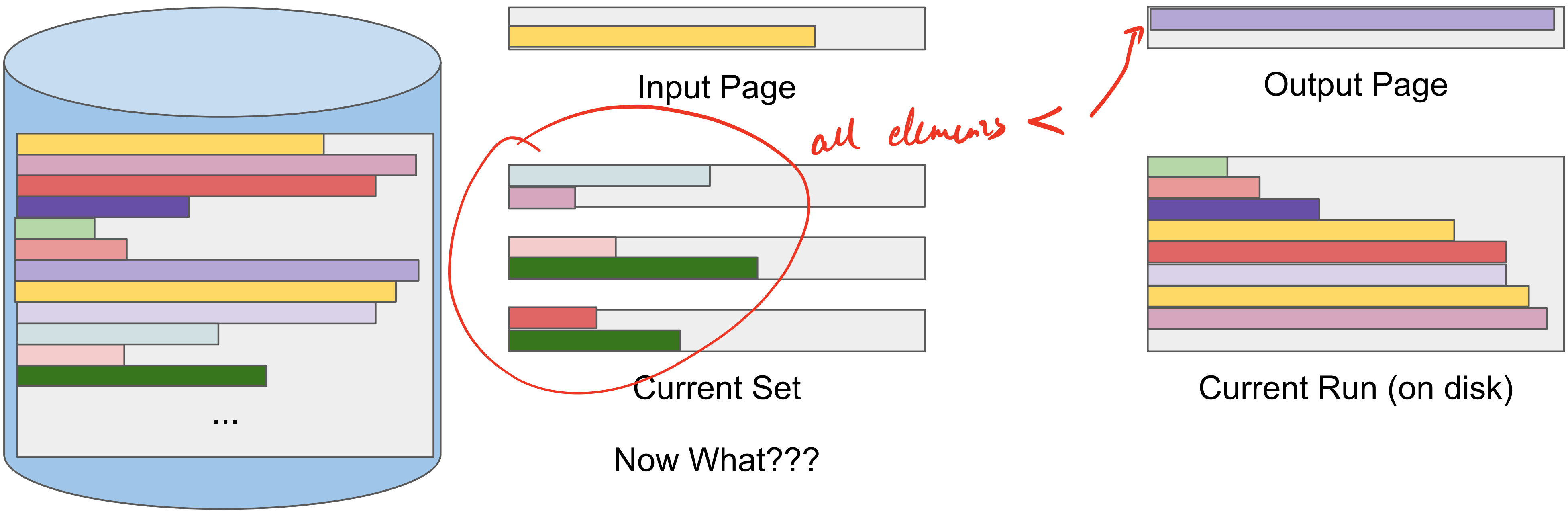
- Take minimum element from current set greater than largest element in current run and put in output page



Step 4

Suppose B = 5 and each page can hold 2 bars in full.

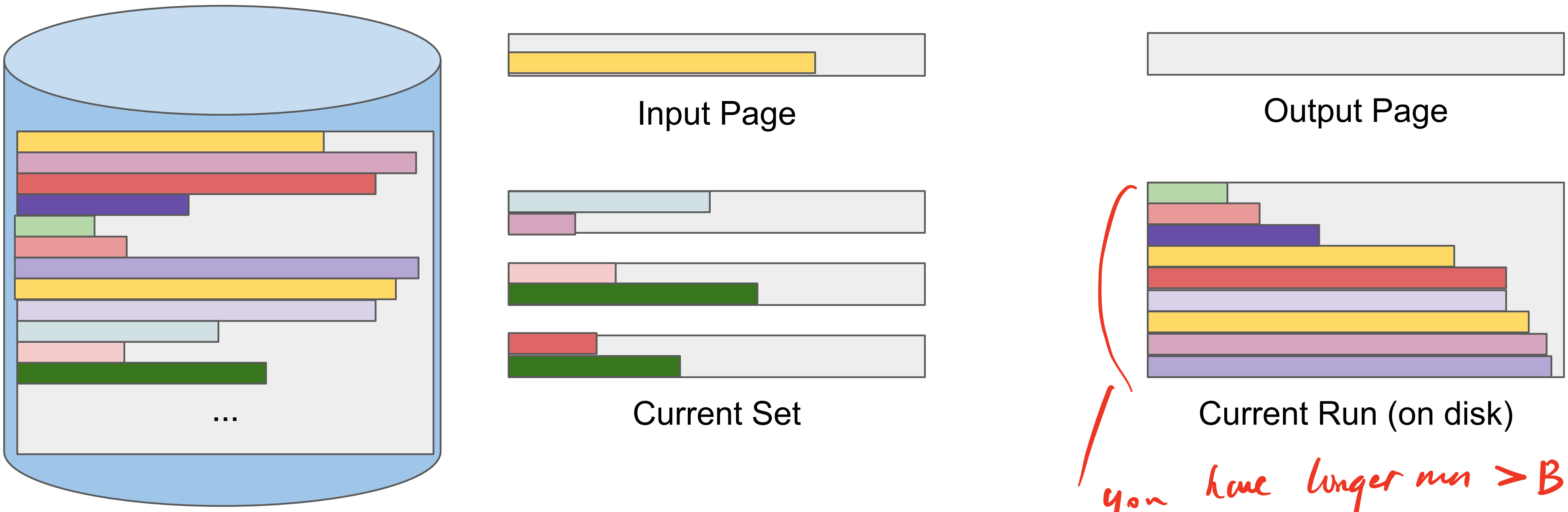
- Take minimum element from current set greater than largest element in current run and put in output page



Step 5

Suppose B = 5 and each page can hold 2 bars in full.

- Write last of output page to disk and save run

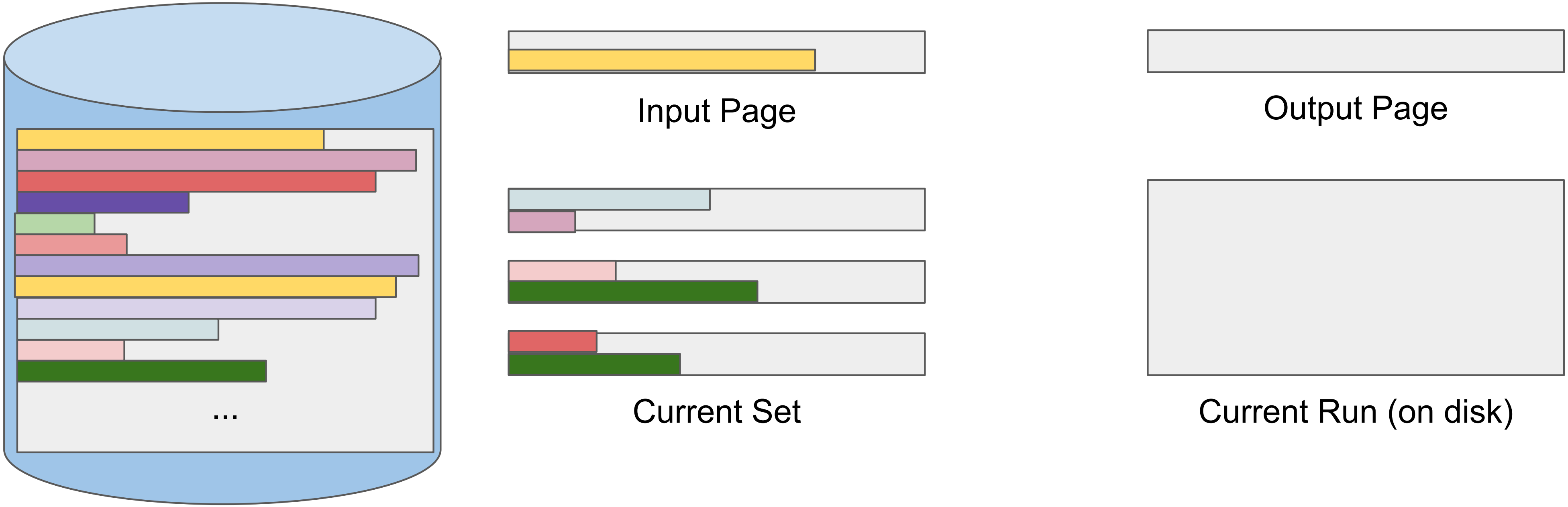


you have longer run > B page long

Step 5

Suppose B = 5 and each page can hold 2 bars in full.

- Create new run and repeat




DID YOU CREATE
THE ULTIMATE EXAMPLE?
YES
WHAT DID IT COST?
EVERYTHING
imgflip.com

Replacement Sort Math

$>B$

- Average length of a run is 2*(B-2) pages
  - Proof left as an exercise to the reader

$2(B-2)$-page-long runs

- Fewer but longer runs
  - ceiling(N/(2*(B-2))) runs

$$1+\left\lceil \log_{B-1}\left\lceil \frac{N}{2B-4} \right\rceil \right\rceil$$

- Takes 1+ceiling($\log_{B-1}$ceiling(N/(2*(B-2)))) passes
  - Should be a smaller number!
  - Not always in all cases though
- Total IO cost is #passes * 2N
  - Smaller because fewer passes!

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?

Input

output

current set

x2 (2x2 elements).

Example

- Suppose we have 4 buffer pages and the following dataset:
    - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {12, 15, 5, 30}

Example

- Suppose we have 4 buffer pages and the following dataset:
    - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {12, 15, 5, 30}
- Run 1 = {}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {12, 15, 30, 33}
- Run 1 = {5}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {15, 30, 33, 51}
- Run 1 = {5, 12}

Example

- Suppose we have 4 buffer pages and the following dataset:
    - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {30, 33, 51, 8}
- Run 1 = {5, 12, 15}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {33, 51, 8, 1}
- Run 1 = {5, 12, 15, 30}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {51, 8, 1, 2}
- Run 1 = {5, 12, 15, 30, 33}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {8, 1, 2, 7}
- Run 1 = {5, 12, 15, 30, 33, 51}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {8, 1, 2, 7}
- Run 1 = {5, 12, 15, 30, 33, 51}
- No more elements in current set greater than largest element in run :(
  - Flush output and start a new run

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {8, 1, 2, 7}
- Run 2 = {}
- Run 1 = {5, 12, 15, 30, 33, 51}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {8, 2, 7} - No more data!
- Run 2 = {1}
- Run 1 = {5, 12, 15, 30, 33, 51}

Example

- Suppose we have 4 buffer pages and the following dataset:
    - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {8, 7} - No more data!
- Run 2 = {1, 2}
- Run 1 = {5, 12, 15, 30, 33, 51}

Example

- Suppose we have 4 buffer pages and the following dataset:
    - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {8} - No more data!
- Run 2 = {1, 2, 7}
- Run 1 = {5, 12, 15, 30, 33, 51}

Example

- Suppose we have 4 buffer pages and the following dataset:
  - 12, 15, 5, 30, 33, 51, 8, 1, 2, 7
- Each page can store 2 elements
- How many runs will we have?
- 4 buffer pages -> 1 input, 1 output page -> 2 pages for current set
- Current set = {} - No more data!
- Run 2 = {1, 2, 7, 8}
- Run 1 = {5, 12, 15, 30, 33, 51}
- Done :)
- 2 Runs

lead runs with different length

on avg : $2(B-2)$ - length runs.

Blocked I/O

$1 + \left\lceil \log_{\lfloor \frac{B}{b} \rfloor - 1} \left\lceil \frac{N}{B} \right\rceil \right\rceil$

Sequential I/O cheaper than random I/O

- Pages in memory live in segments called Blocks
  - Often cheaper to load an entire Block into memory at a time (contain multiple pages)
- How to handle
  - Instead of caring about B buffer pages lets take into account our blocks
  - Block consists of b pages
  - We instead have floor(B/b) buffer “blocks”
    - Floor since we can’t have half a block and we’re given an upper bound on the number of pages (and blocks) that can fit into memory at the same time
  - #passes = 1+ceiling($\log_{\text{floor}(B/b)-1}$(ceiling(N/B)))
    - More passes but less per page costs
  - Everything else is the same as usual

$\lfloor \frac{B}{b} \rfloor$

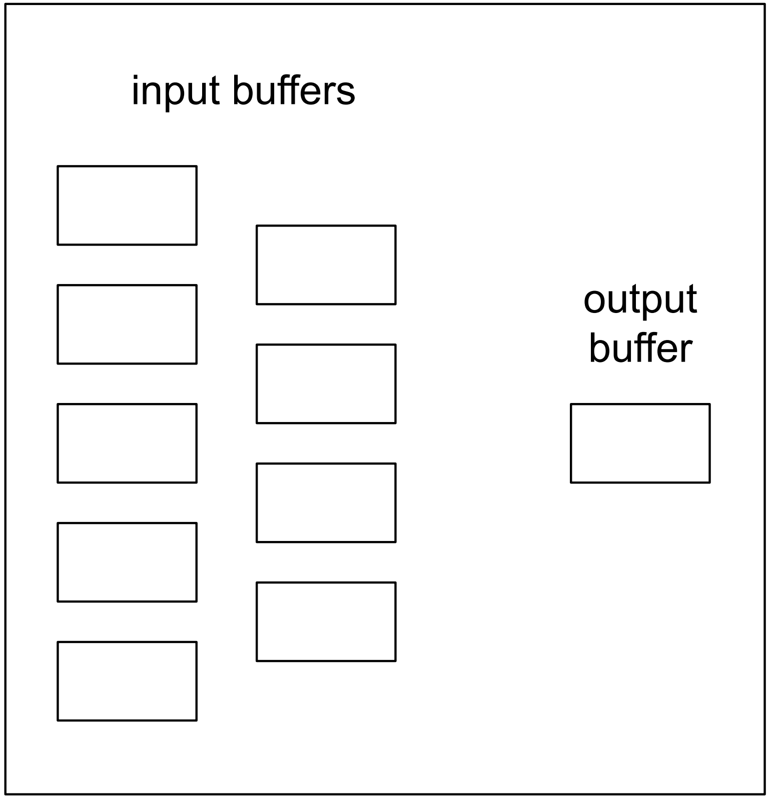
floor: can't read $\frac{1}{3}$ of block

$\lfloor \frac{B}{b} \rfloor - 1$ output buffer block

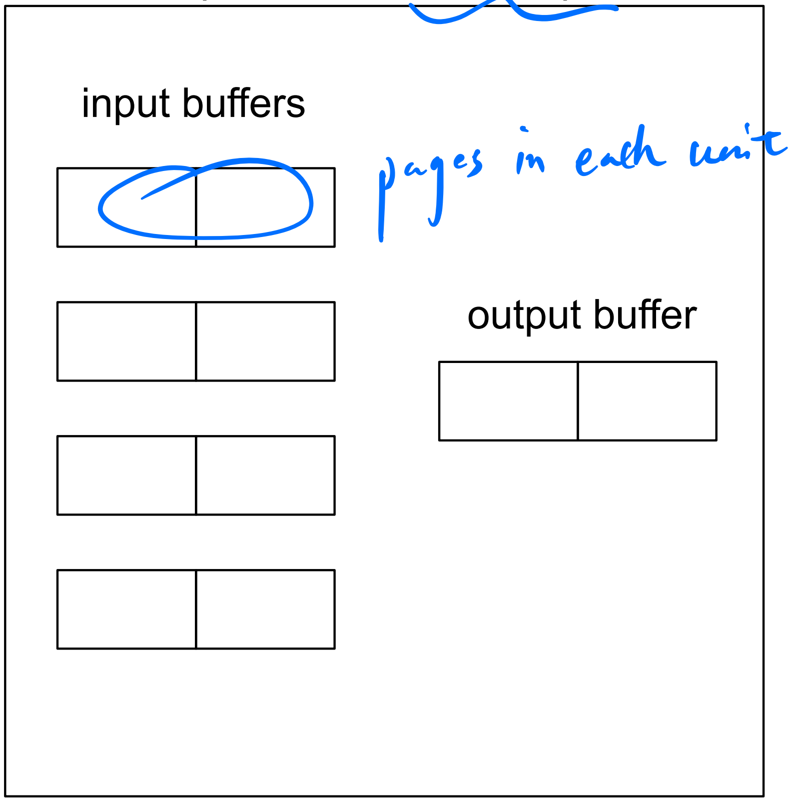
log base ↓ pass ↑ ⇒ I/O ↑

do $\lfloor \frac{B}{b} \rfloor$-way merge

non-blocked I/O



blocked I/O
(suppose b = 2)

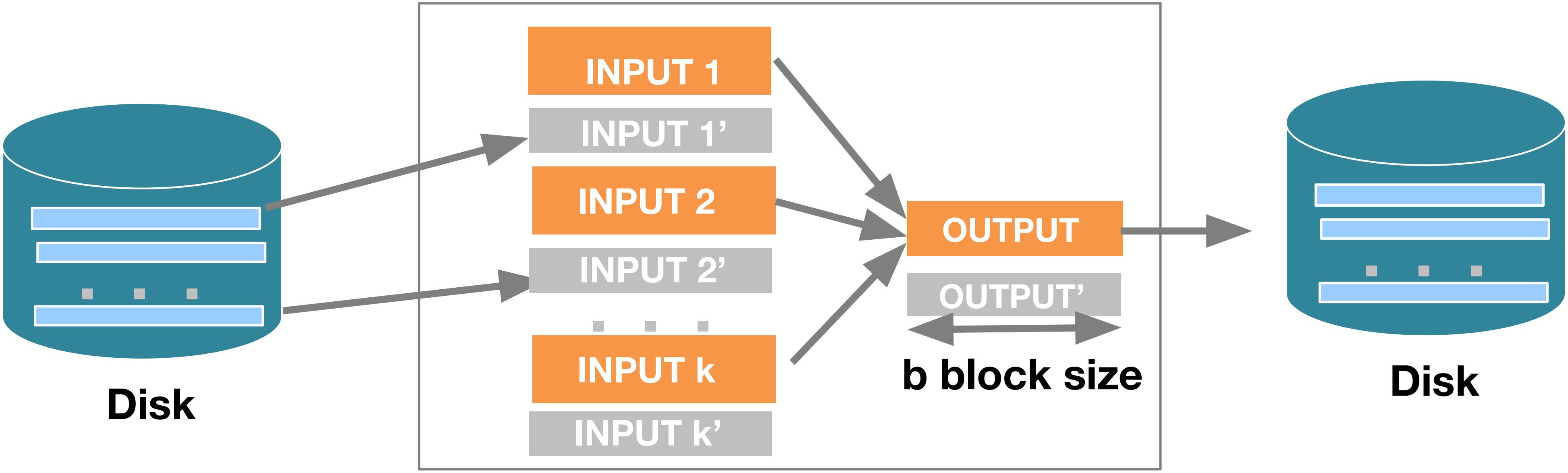


Double Buffering

keep cpu and disk working all the time

disk read in orange | grey
cpu do grey | orange

- I/O processing needs time
- Overlap CPU and I/O processing



2*(K+1)*b main memory buffers, k-way merge

Extra Examples

- We have a 565 GB dataset with page size of 8 KB and 2 GB RAM
  - How many IOs required to sort?
  - Hint: figure out how many buffer pages we can fit in RAM along with how many pages our dataset spans
- Using replacement sort at step 0, how many IOs do we need now?
- How many IOs will we need if we use blocking (but not replacement sort)?
  - Block size of 64KB
- How many IOs will we need if we use blocking and replacement sort at step 0?
  - Block size of 64KB
- Answers on last slide

Extra Example Answers

- We have a 565 GB dataset with page size of 8 KB and 2 GB RAM. How many IOs required to sort?
  - Number of pages in dataset (N) = 565GB / (8KB / page) = 74,055,680 data pages
  - Number of buffer pages (B) = 2GB / (8KB / page) = 262,144 buffer pages
  - # passes = 1+ceiling($\log_{B-1}$(ceiling(N/B))) = 2
  - # IOs = # passes * 2N = 296,222,720 IOs
- Using replacement sort at step 0, how many IOs do we need now?
  - # passes = 1+ceiling($\log_{B-1}$(ceiling(N/(2*(B-2))))) = 1+ceiling($\log_{B-1}$(142))=2
  - # IOs = # passes * 2N = 296,222,720 IOs
  - The same??? (Take a look at the log to see why)   fewer passes not guaranteed
    - Think about which number would I have to decrease (N or B) to have replacement sort result in fewer passes than the previous question

Extra Example Answers

- How many IOs will we need if we use blocking (but not replacement sort)?
  - Block size of 64KB
  - # pages in a block (b) = (64 KB/block)/(8 KB/page) = 8 pages/block   b = 8
  - # passes = 1+ceiling($\log_{\text{floor}(B/b)-1}$(ceiling(N/B)))= 2
    - Again ???
  - # IOs = # passes * 2N = 296,222,720 IOs   total I/O is same but each page is cheaper
- How many IOs will we need if we use blocking and replacement sort at step 0?
  - Block size of 64KB
  - # passes = 1+ceiling($\log_{\text{floor}(B/b)-1}$(ceiling(N/(2*(B-2)))))= 1+ceiling($\log_{\text{floor}(B/b)-1}$(142))=2
    - Again ???
  - # IOs = # passes * 2N = 296,222,720 IOs   B is large here ⇒ pass # doesn't change much
- We can sort a CRAZY amount of data with just 2GB of RAM in just 2 passes
  - Yay modern computing :)   B smaller now ⇒ # pass may change
  - Try reducing the RAM to 128MB and see how many passes you have to make with each attempt

block I/O + replacement sort $\quad 1 + \left\lceil \log_{\lfloor \frac{B}{b} \rfloor - 1} \left\lceil \frac{N}{2(B-2)} \right\rceil \right\rceil$

Get started on HW5!