External Sort

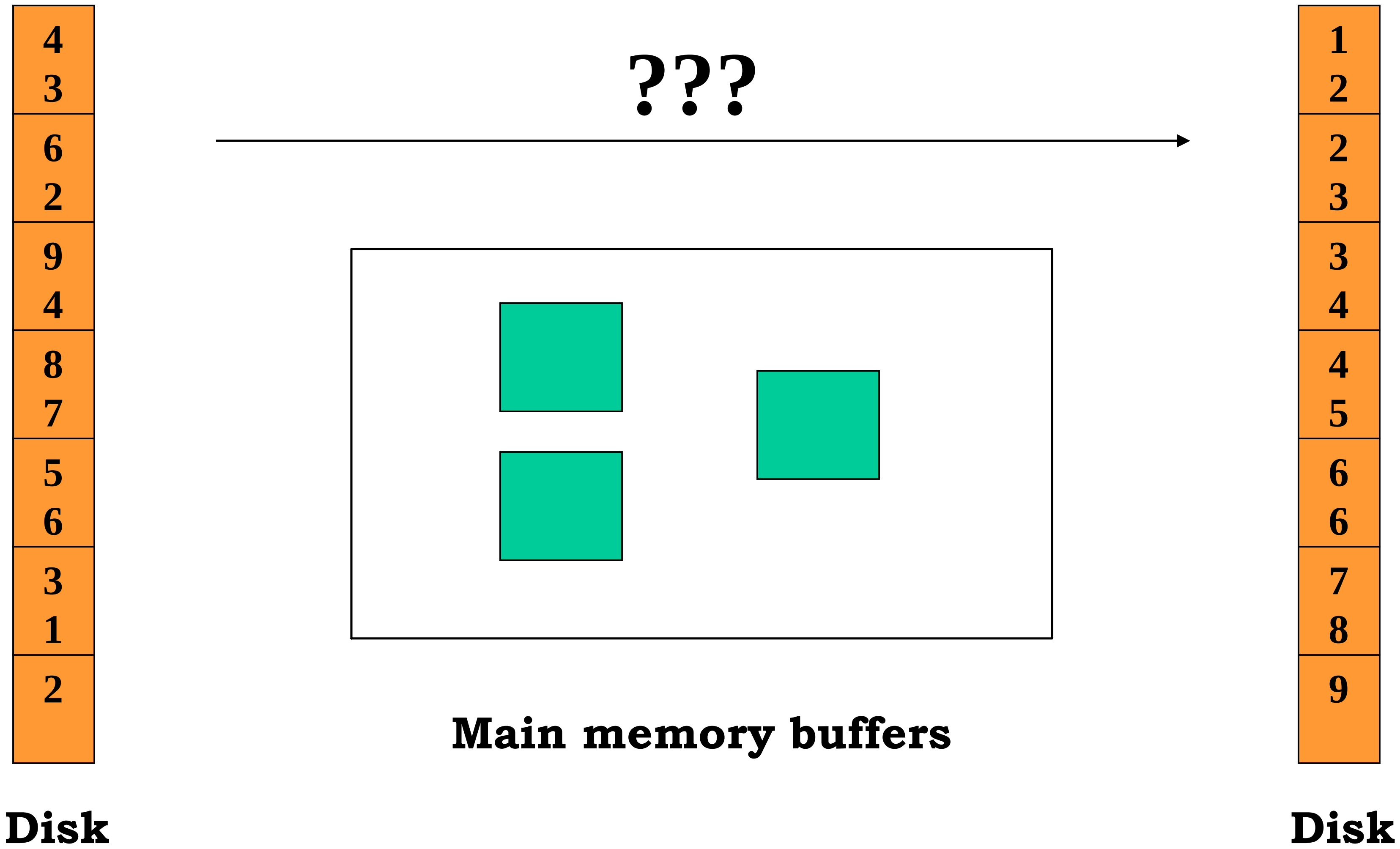
"There it was, hidden in alphabetical order."

Rita Holt

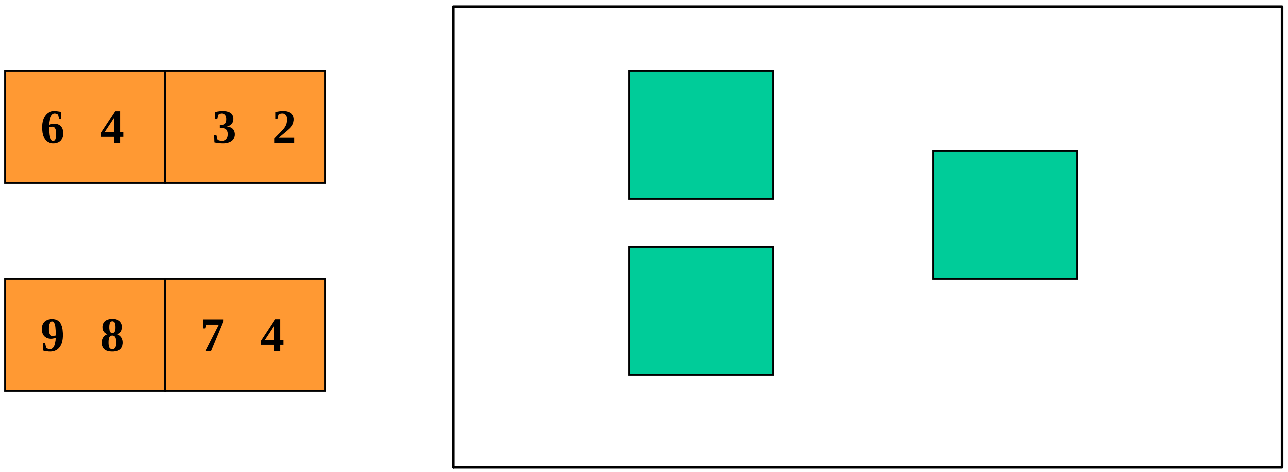
External Sorting

- A classic problem in computer science!
- Data requested in sorted order
  - e.g., find employees in increasing *salary* order
- Sorting is used in many applications
  - First step in *bulk loading* operations
  - Eliminating *duplicate copies* in a collection of records (How?)
  - Evaluating *join* or *set* operations (How?)

Challenge: Sort 1TB of data with 1GB of RAM



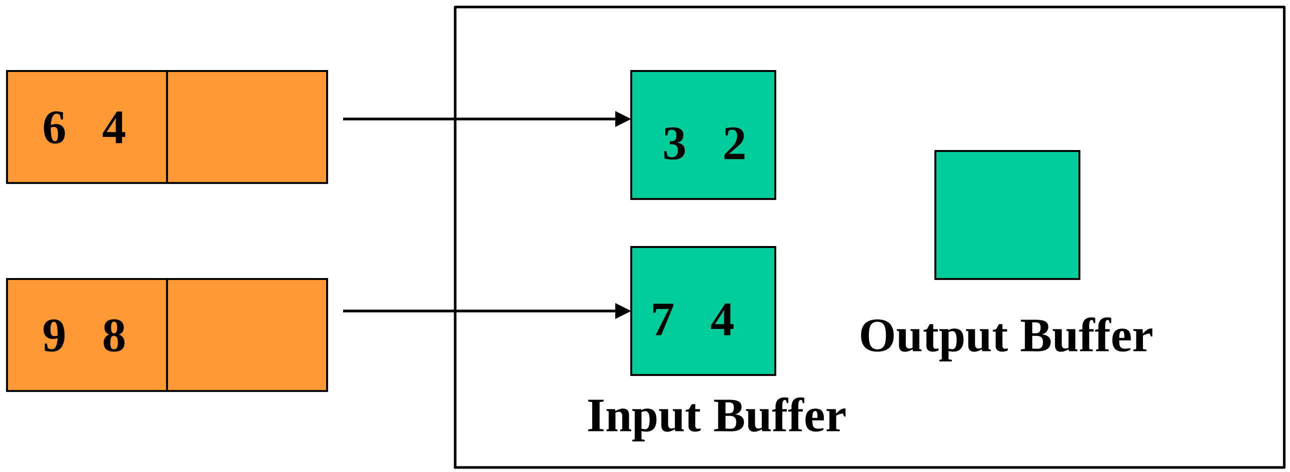
A Simpler Problem: Combine Sorted Files



Main memory buffers

Disk **Disk**

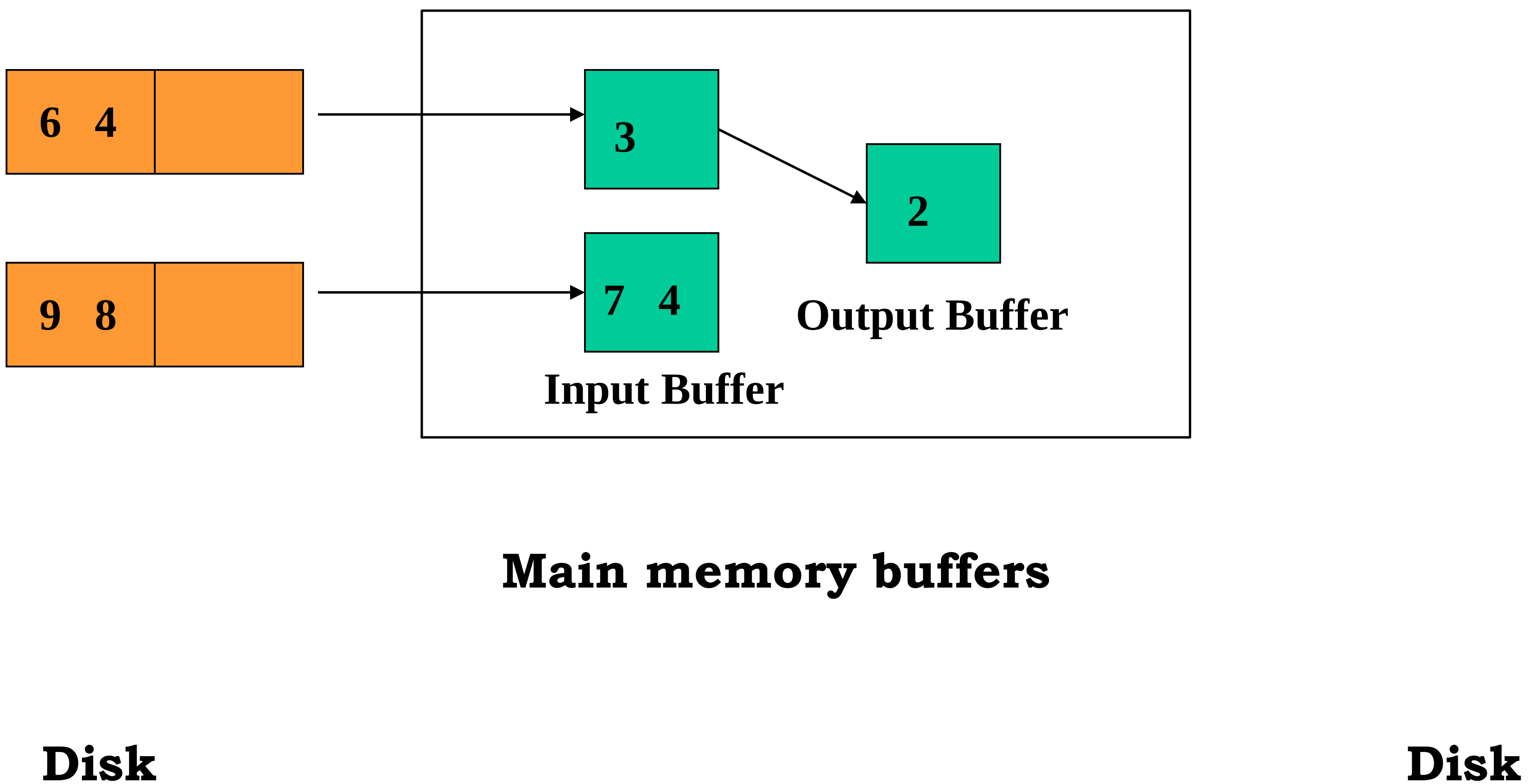
A Simpler Problem: Combine Sorted Files



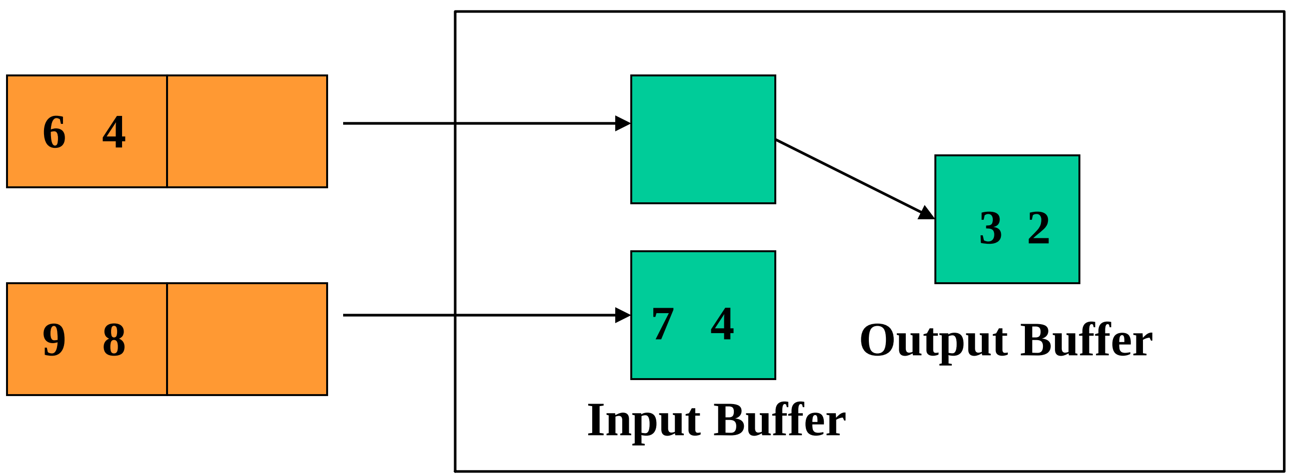
Main memory buffers

Disk **Disk**

A Simpler Problem: Combine Sorted Files



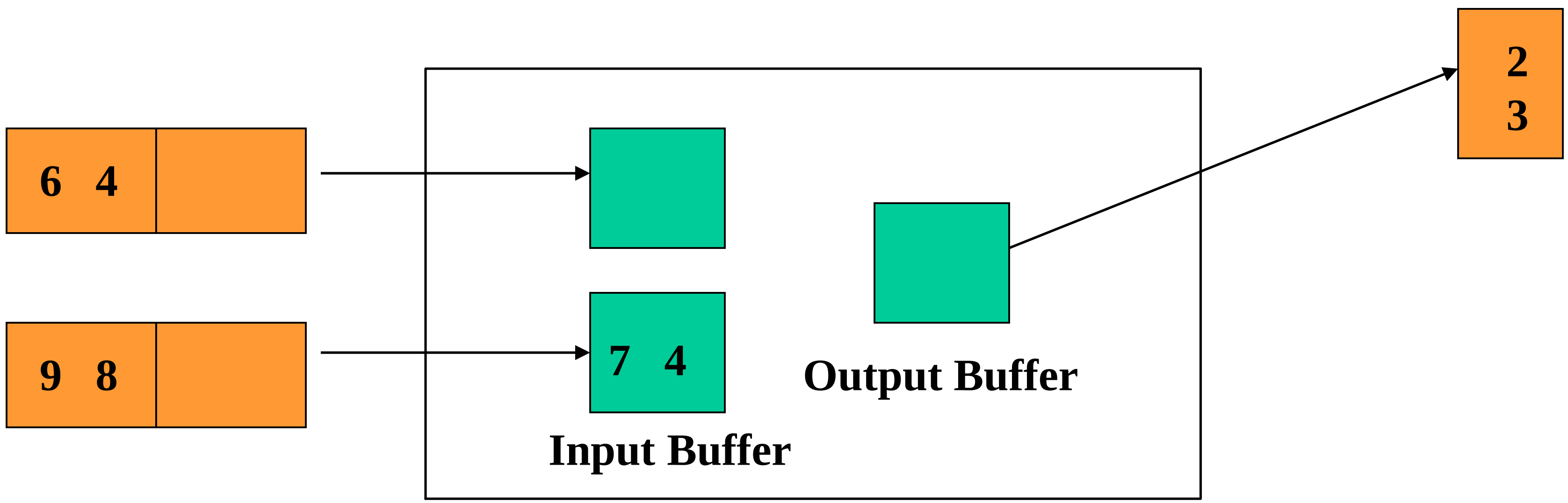
A Simpler Problem: Combine Sorted Files



Main memory buffers

Disk **Disk**

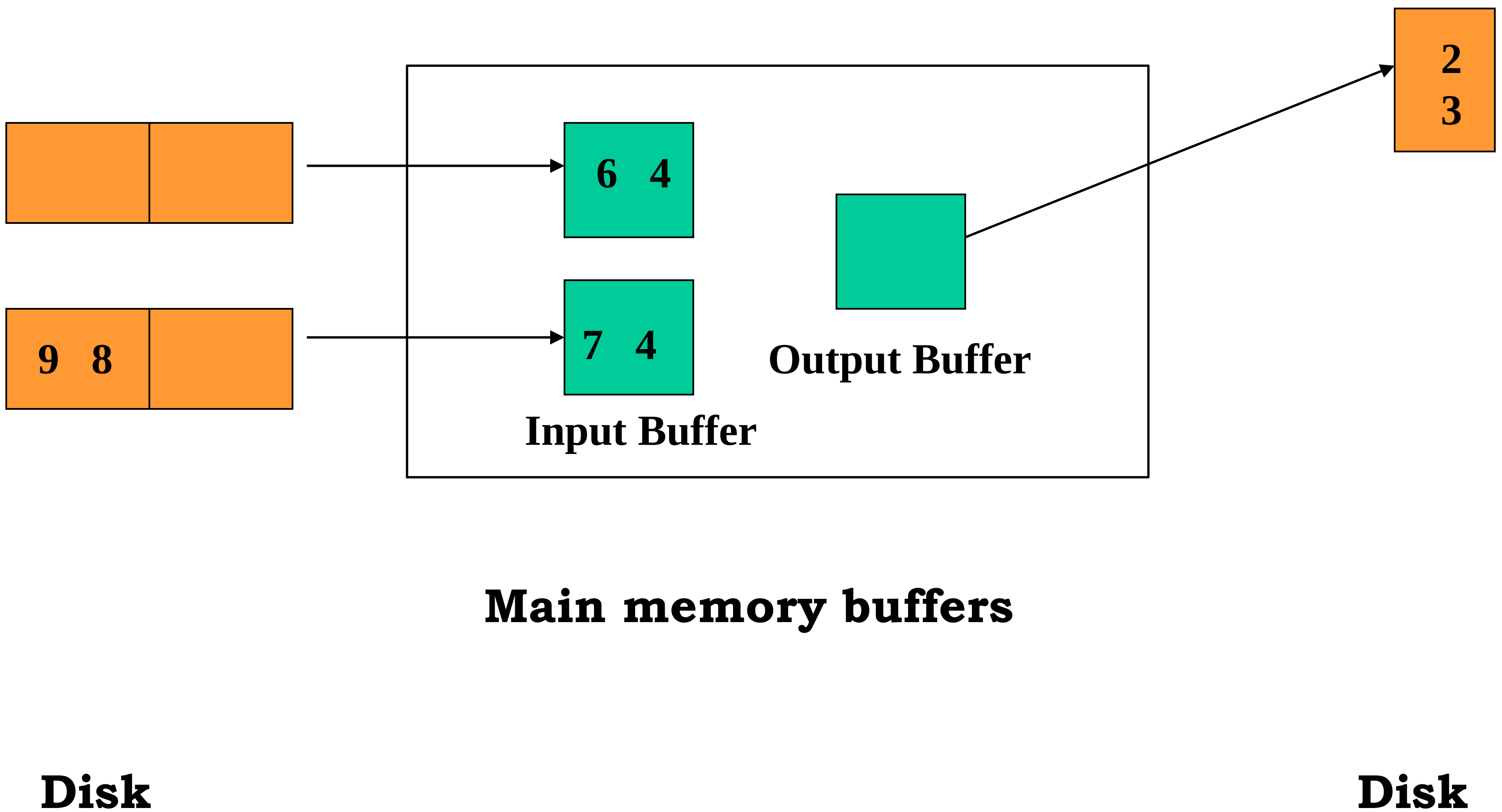
A Simpler Problem: Combine Sorted Files



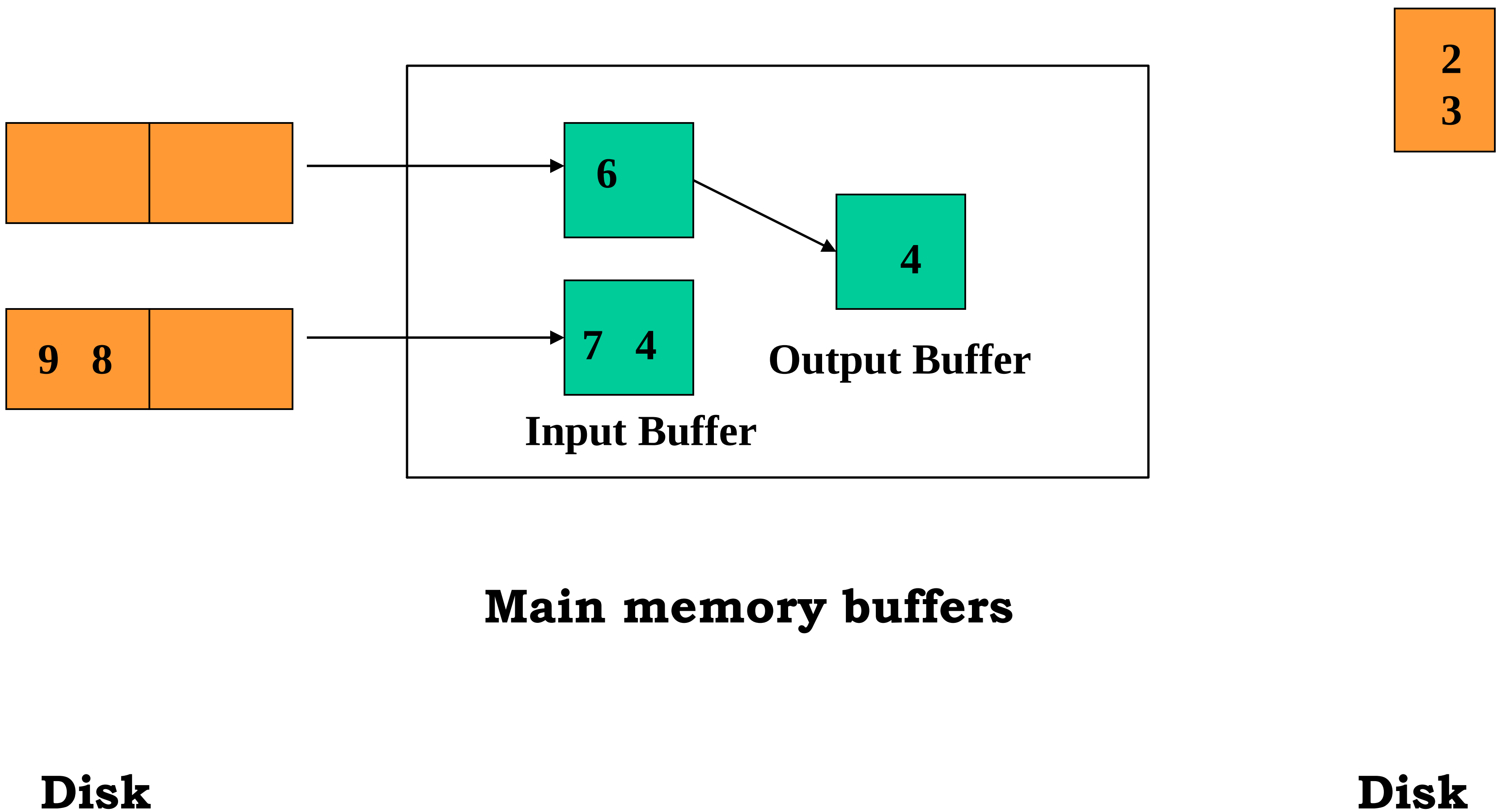
Main memory buffers

Disk **Disk**

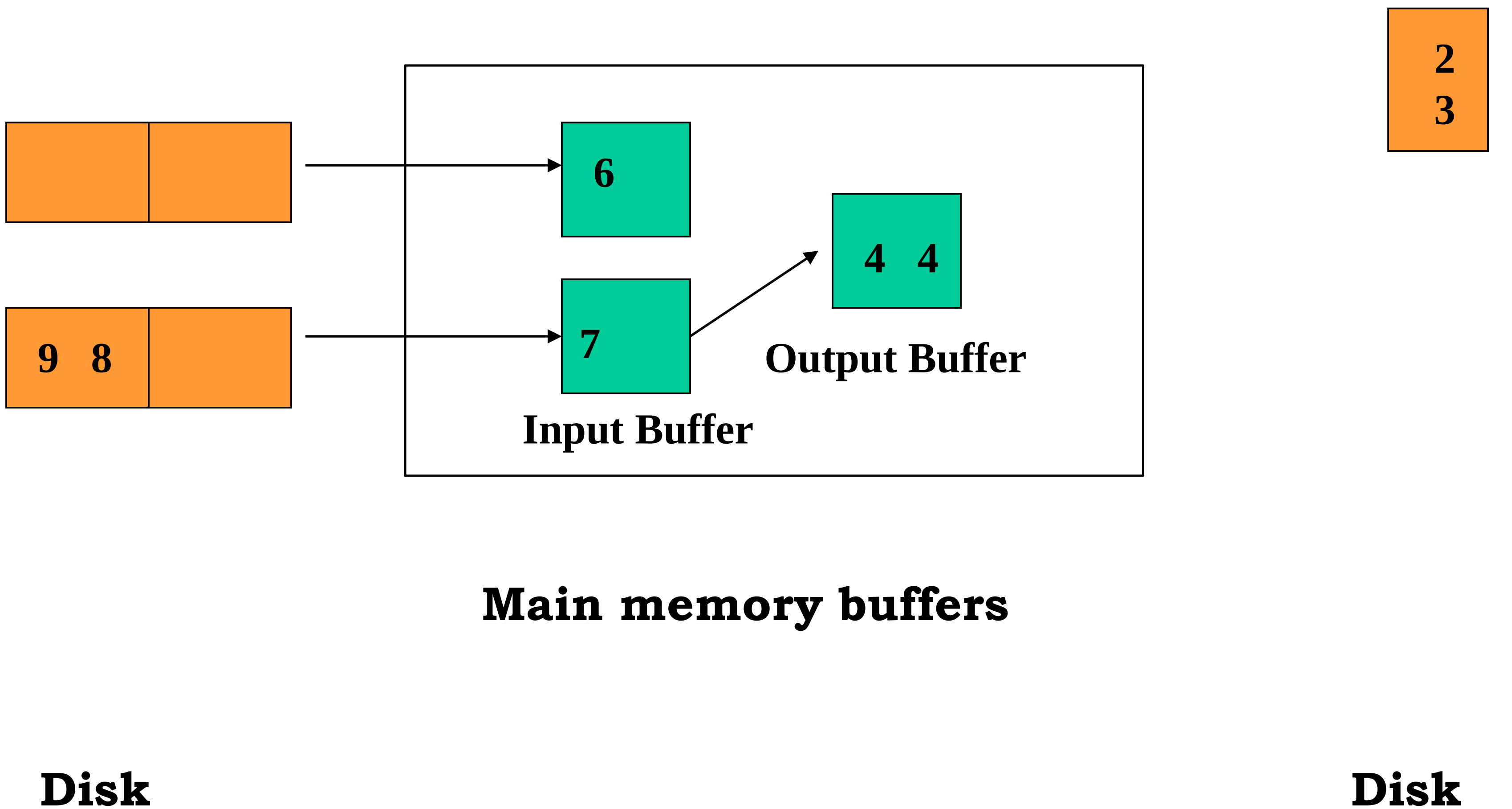
A Simpler Problem: Combine Sorted Files



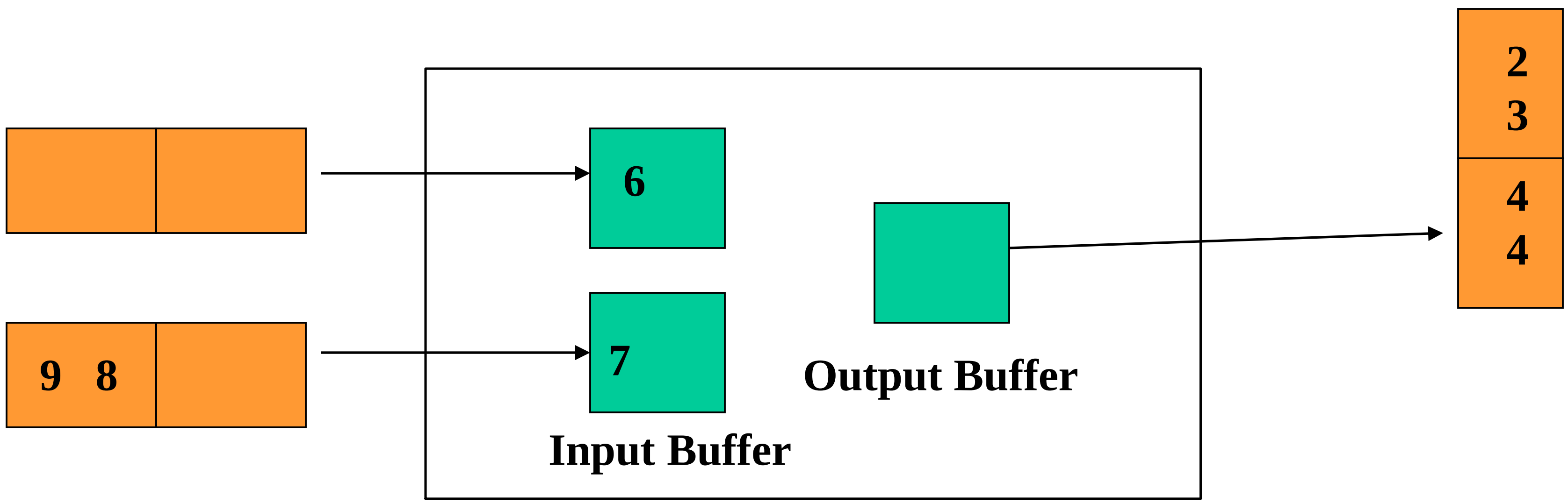
A Simpler Problem: Combine Sorted Files



A Simpler Problem: Combine Sorted Files



A Simpler Problem: Combine Sorted Files

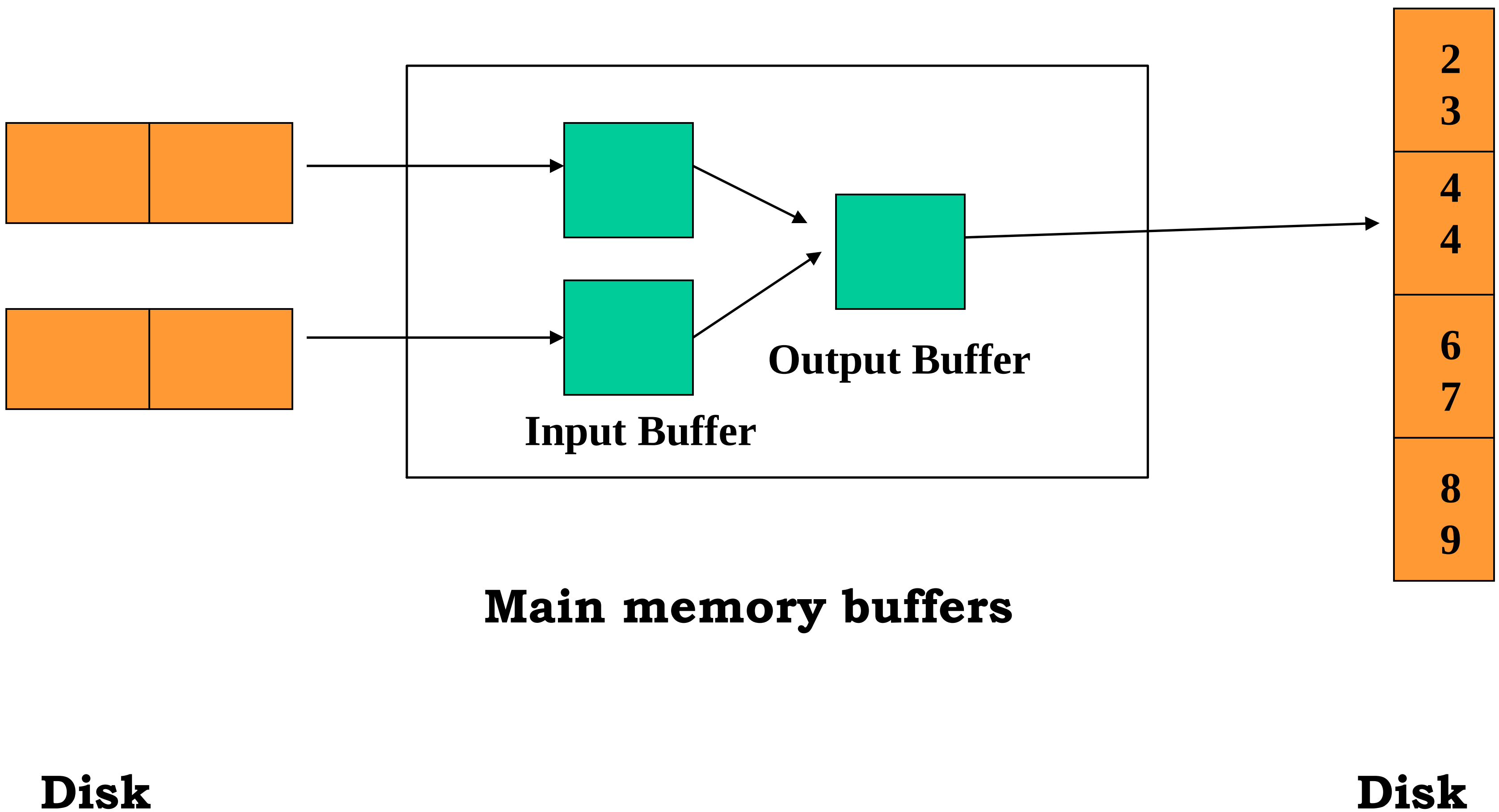


Main memory buffers

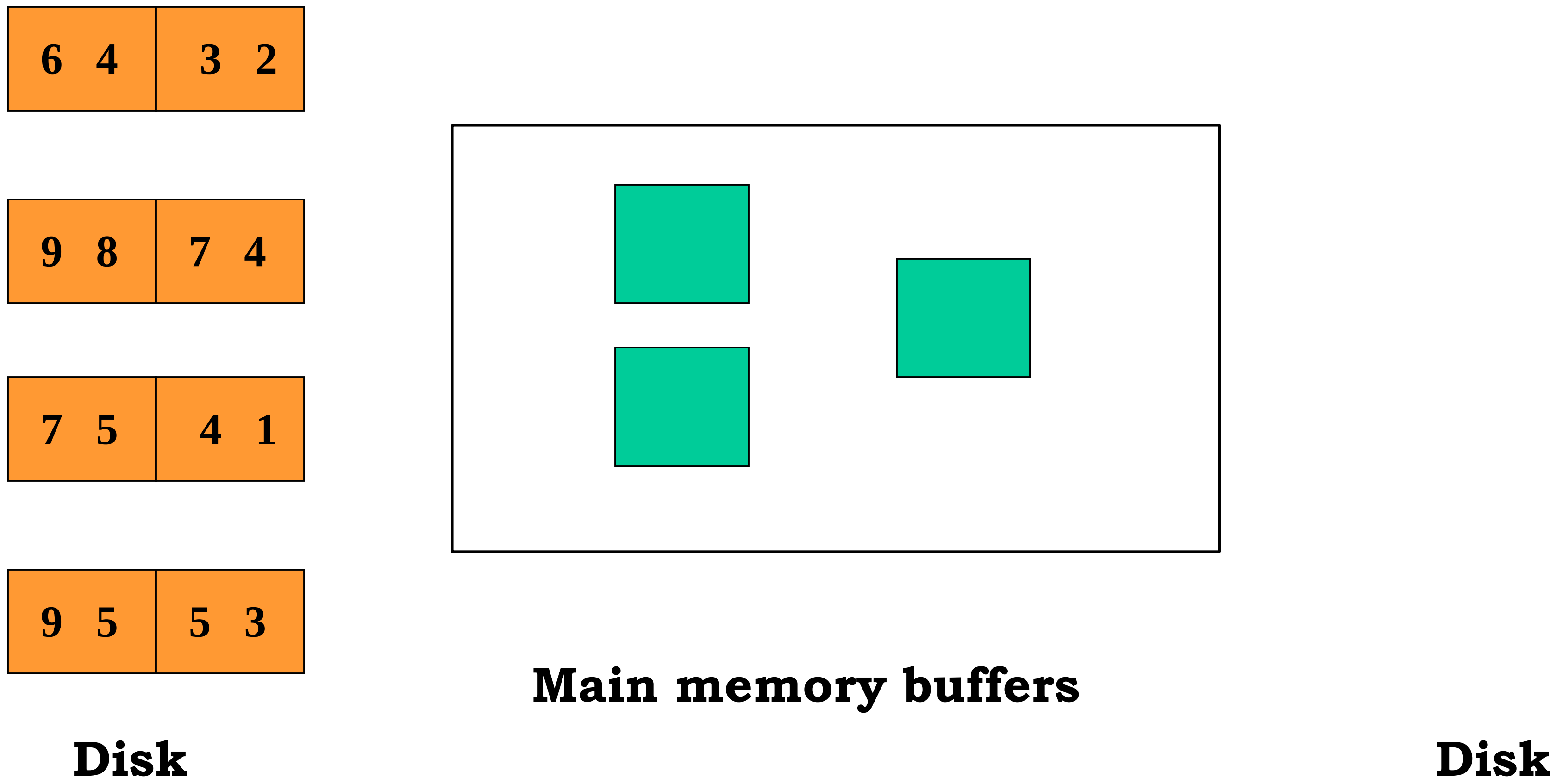
Disk

Disk

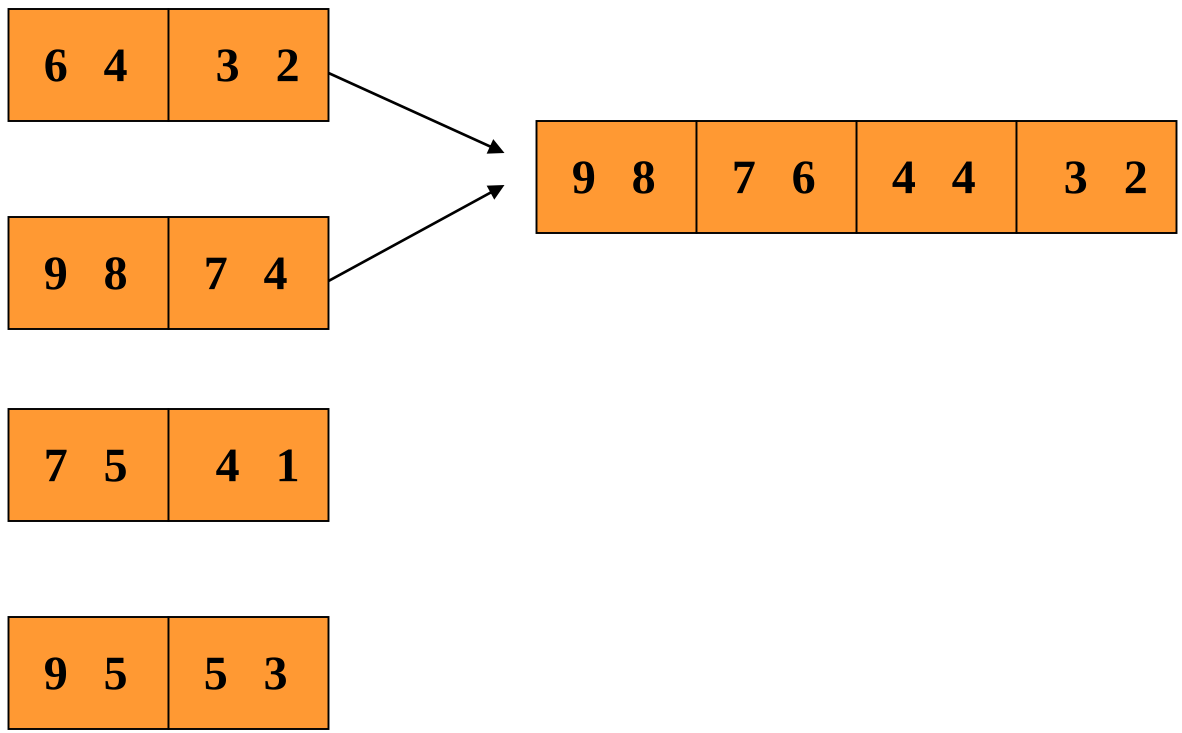
A Simpler Problem: Combine Sorted Files



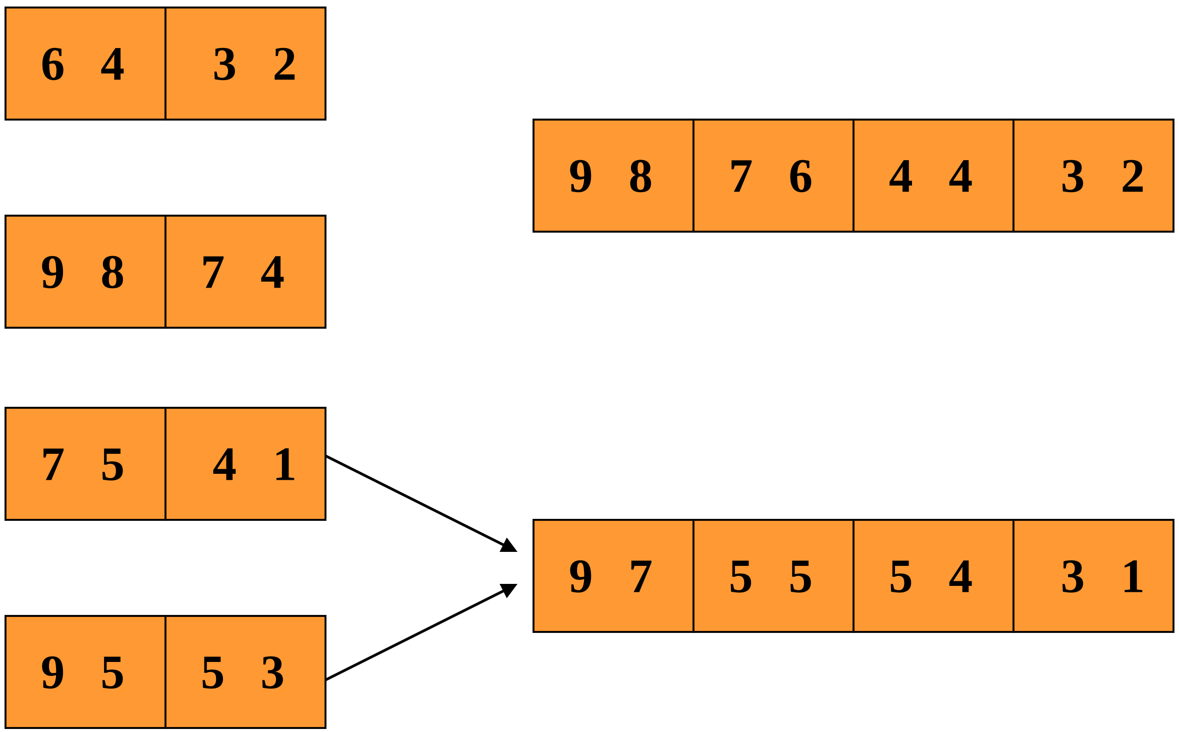
What if there are many more runs (sorted files)?



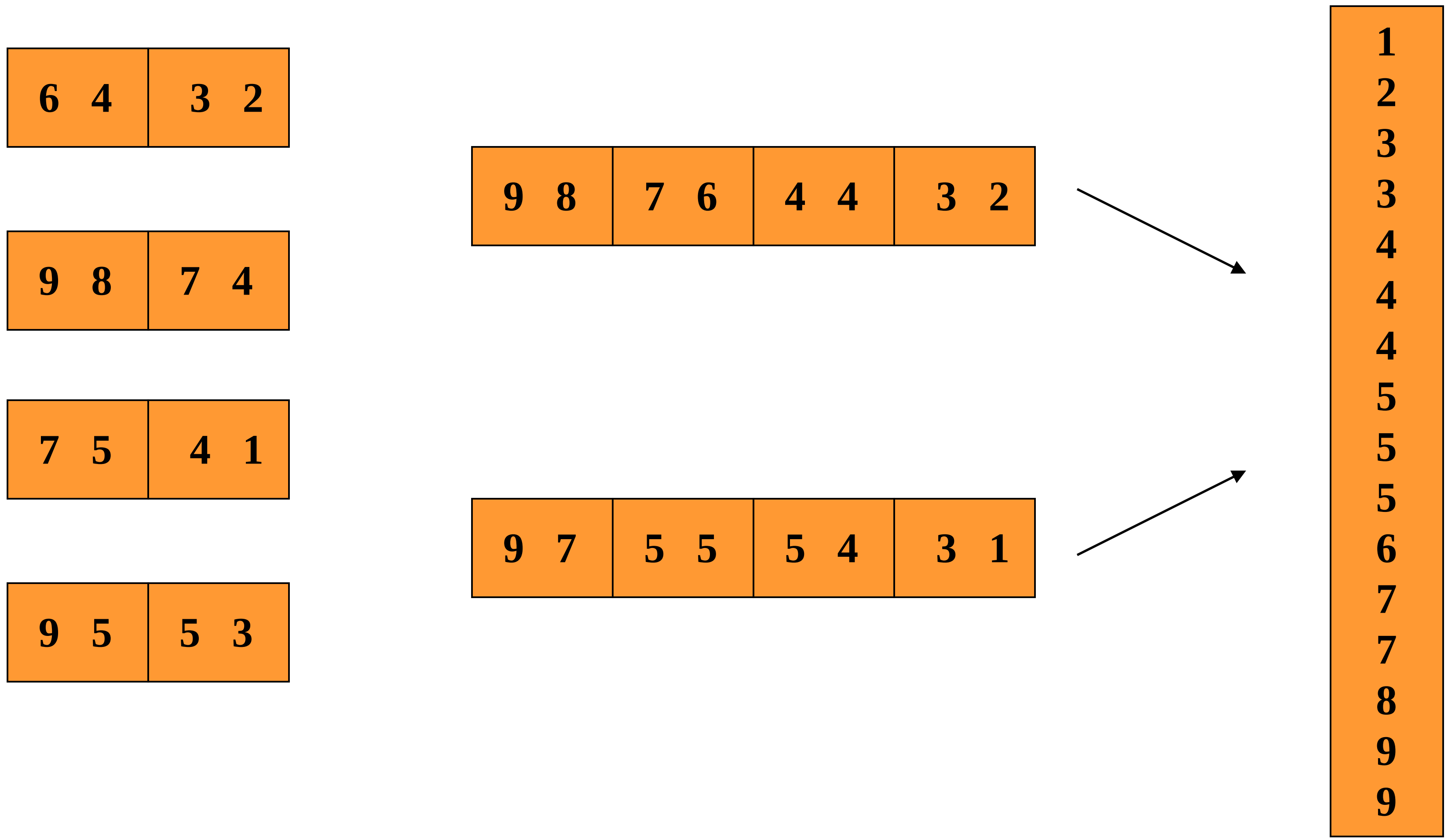
What if there are many more runs?



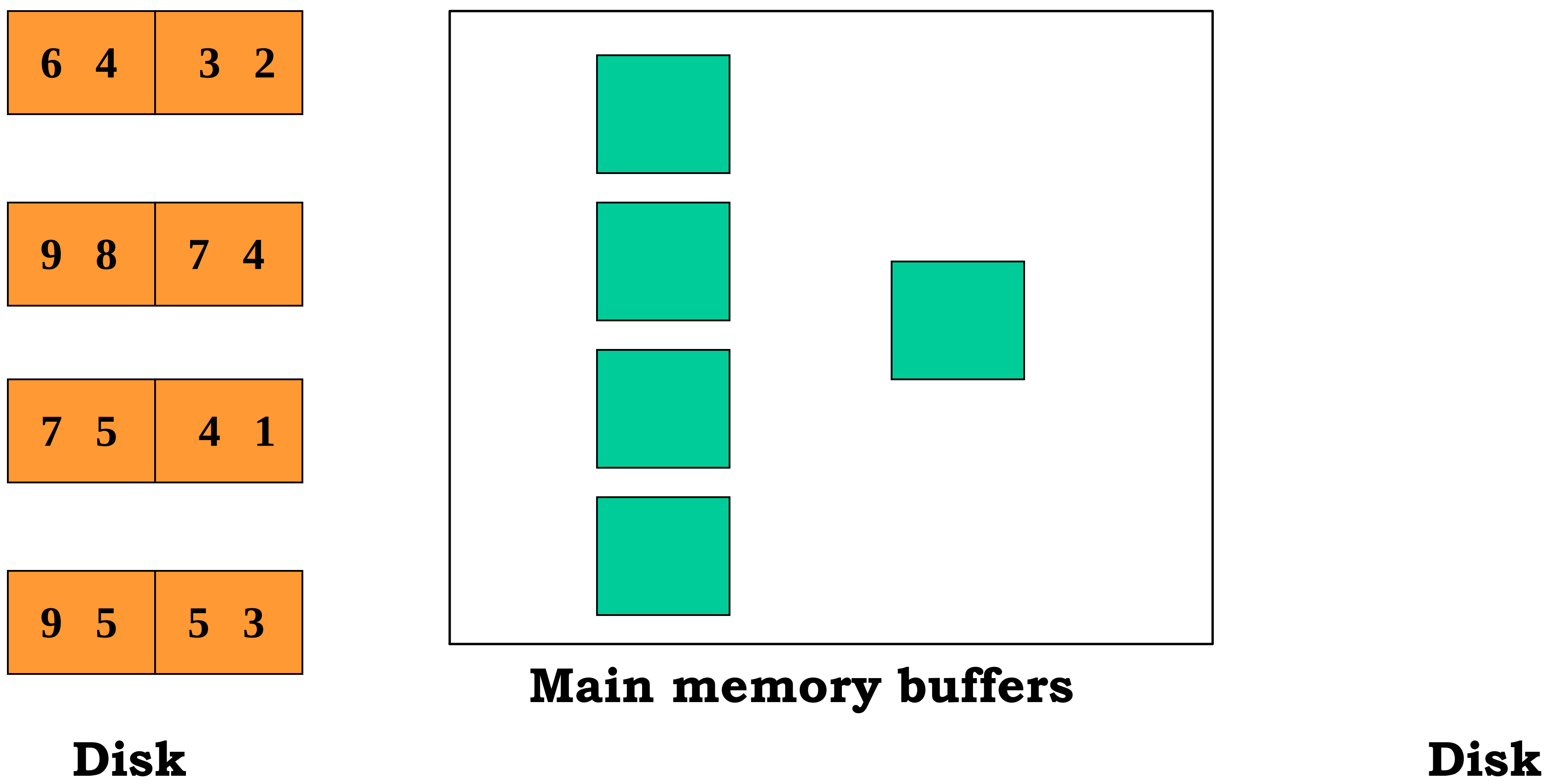
What if there are many more runs?



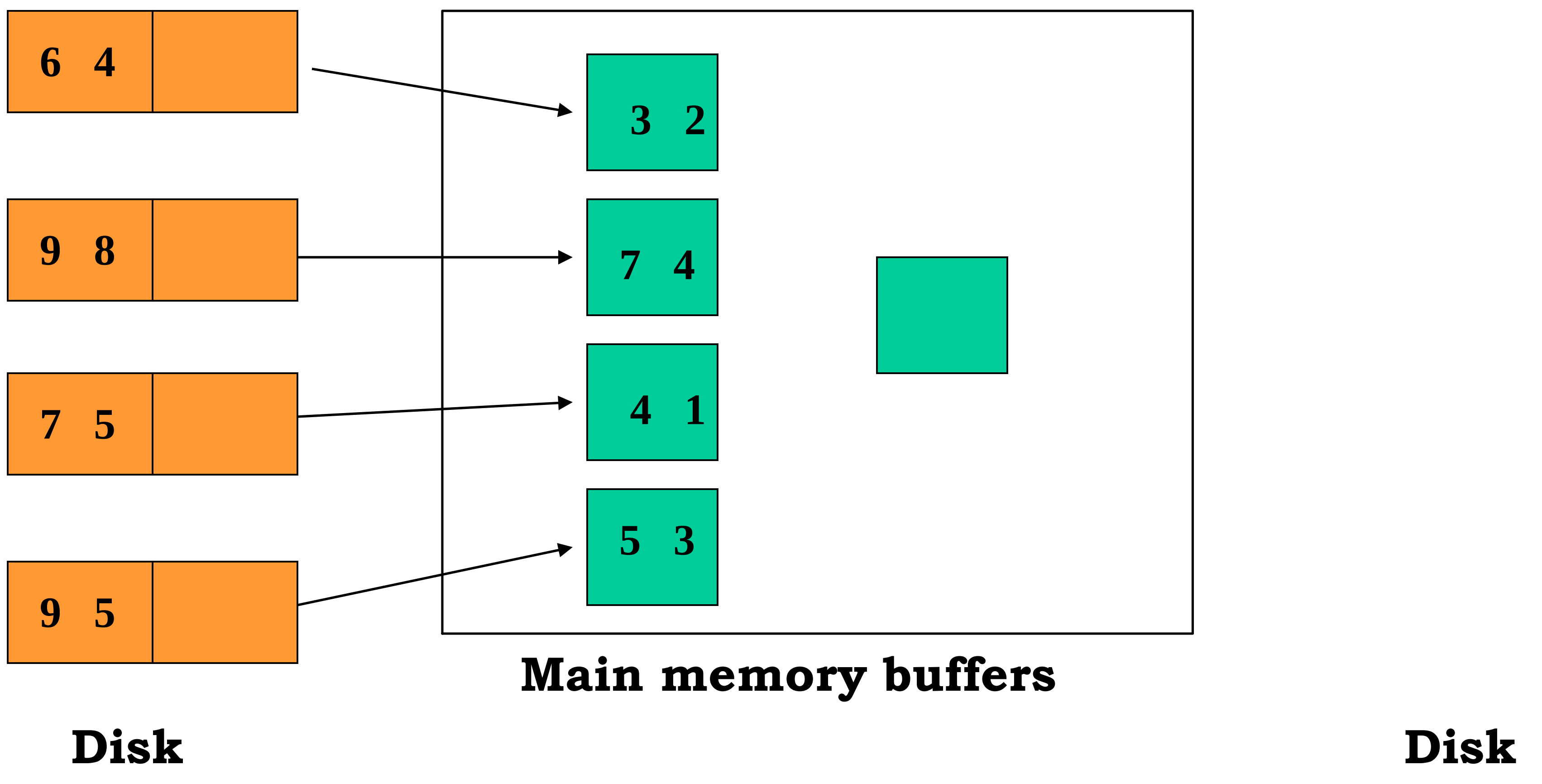
What if there are many more runs?



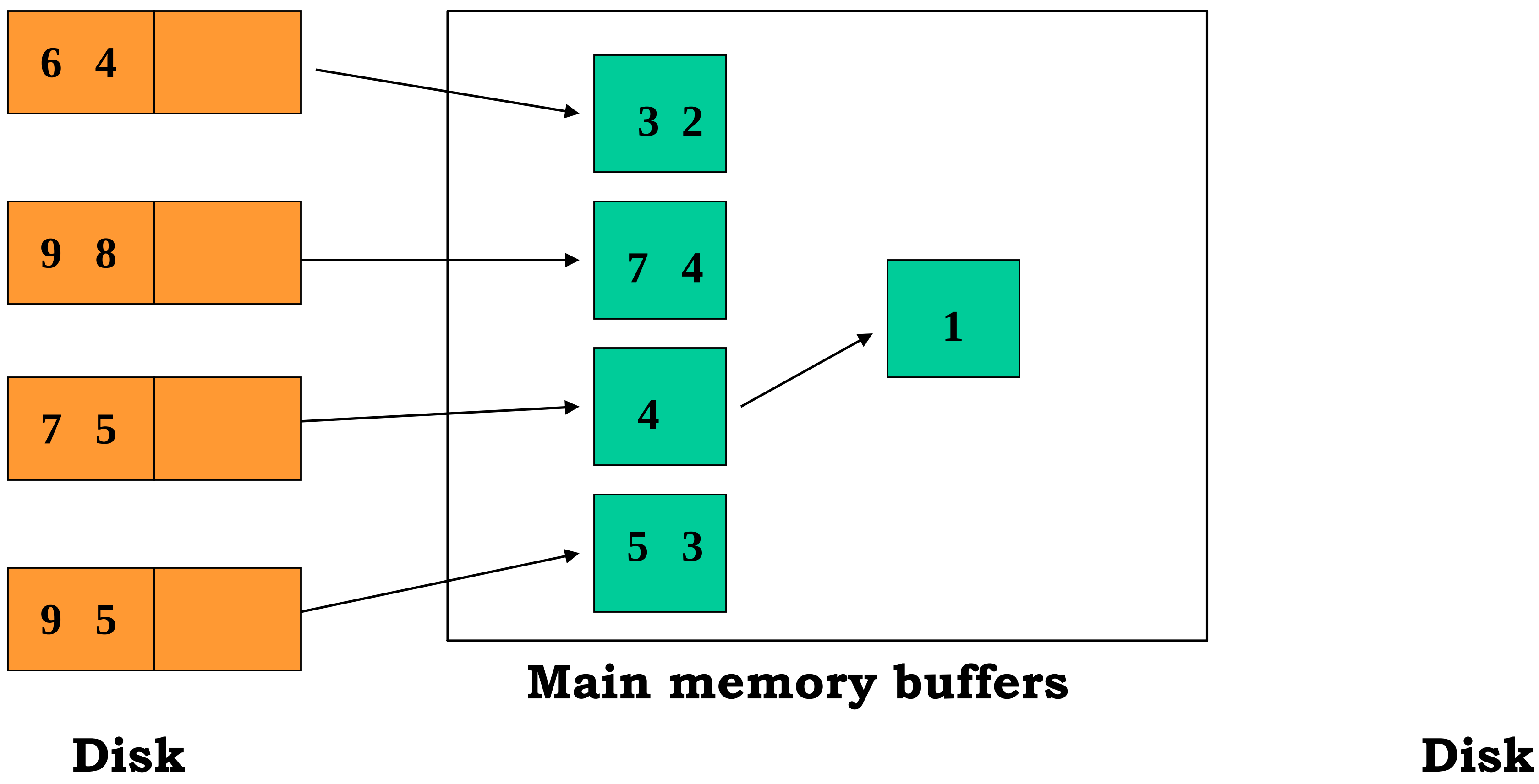
What if there are more memory?



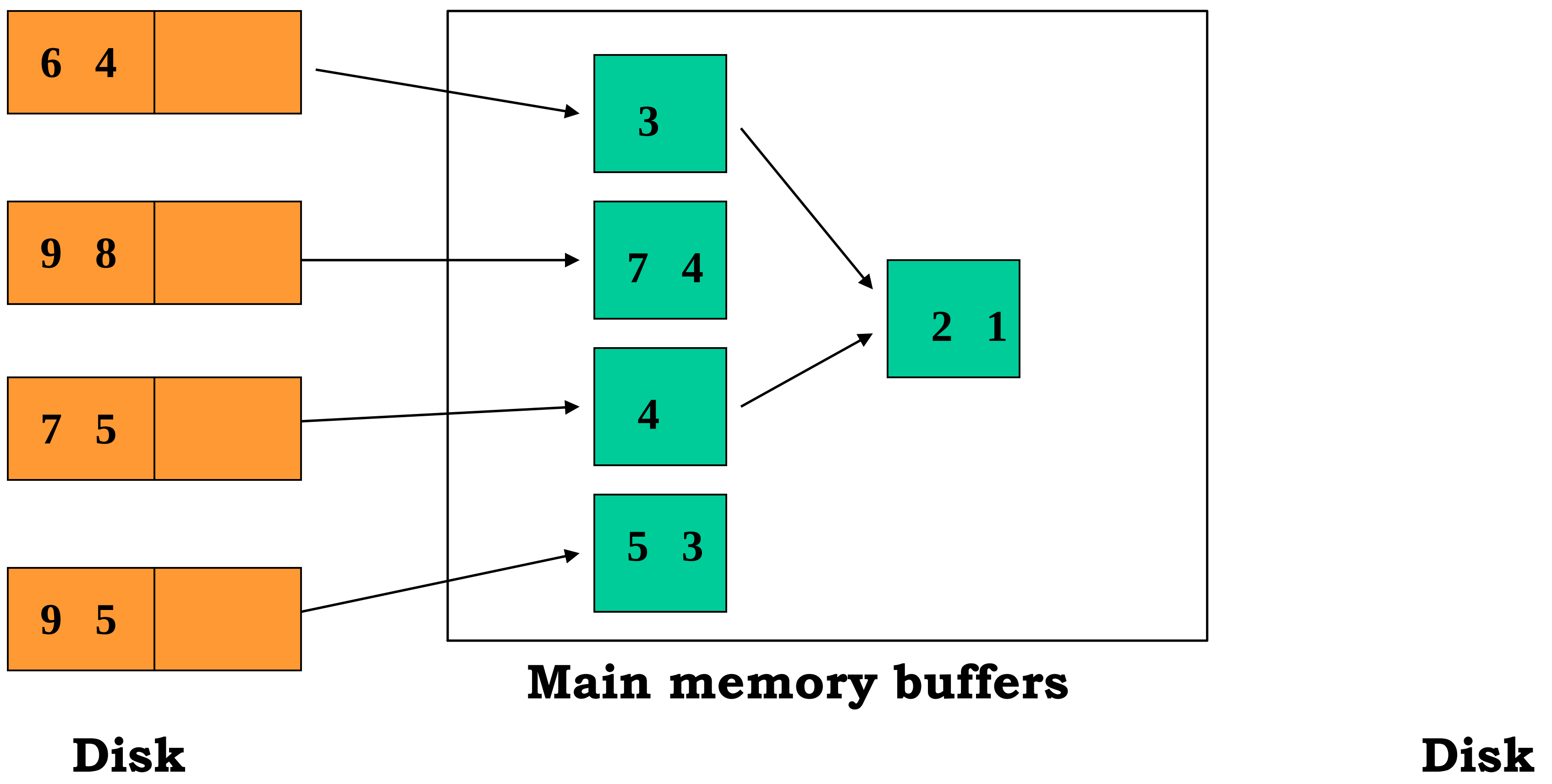
What if there are more memory?



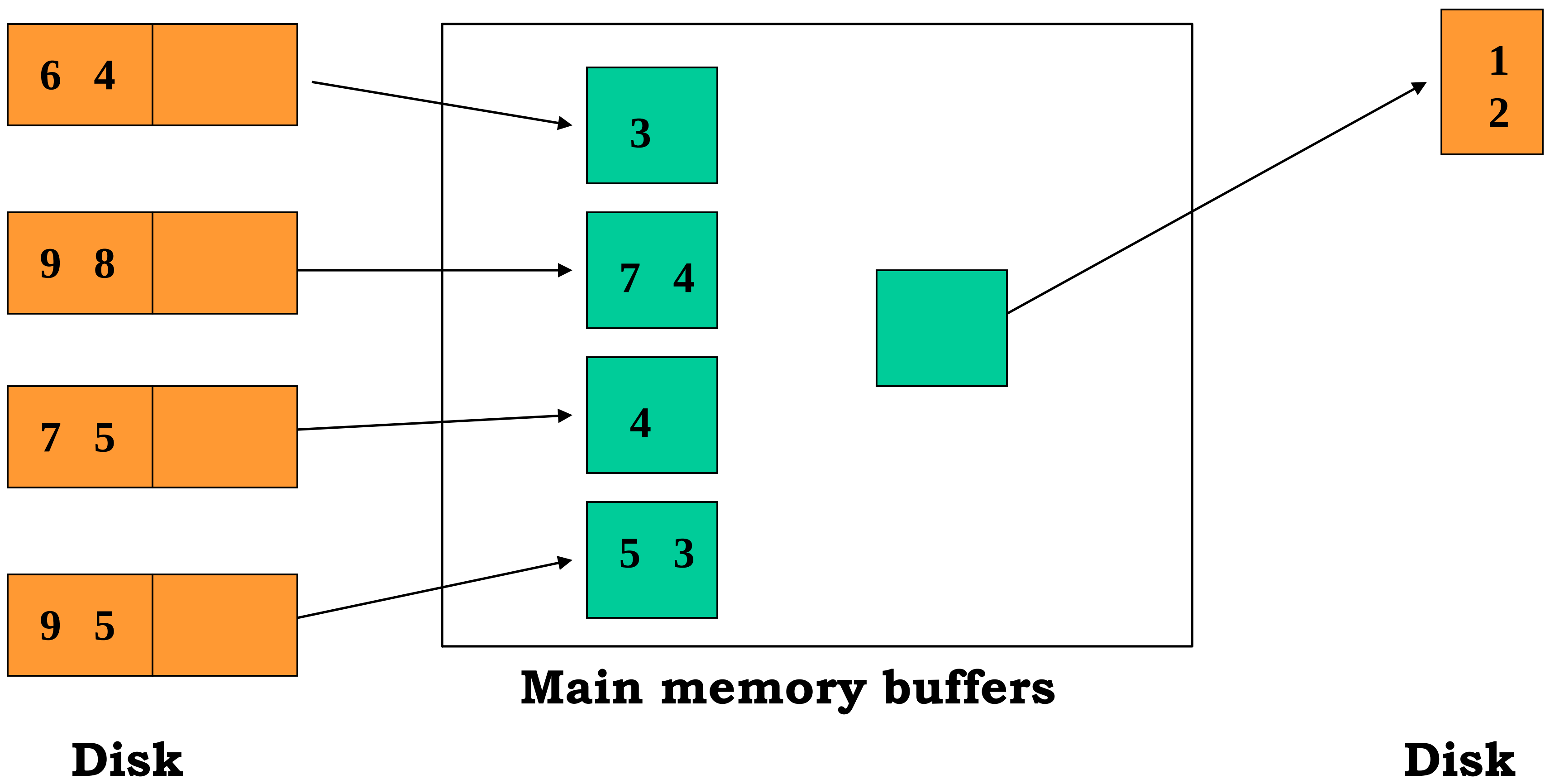
What if there are more memory?



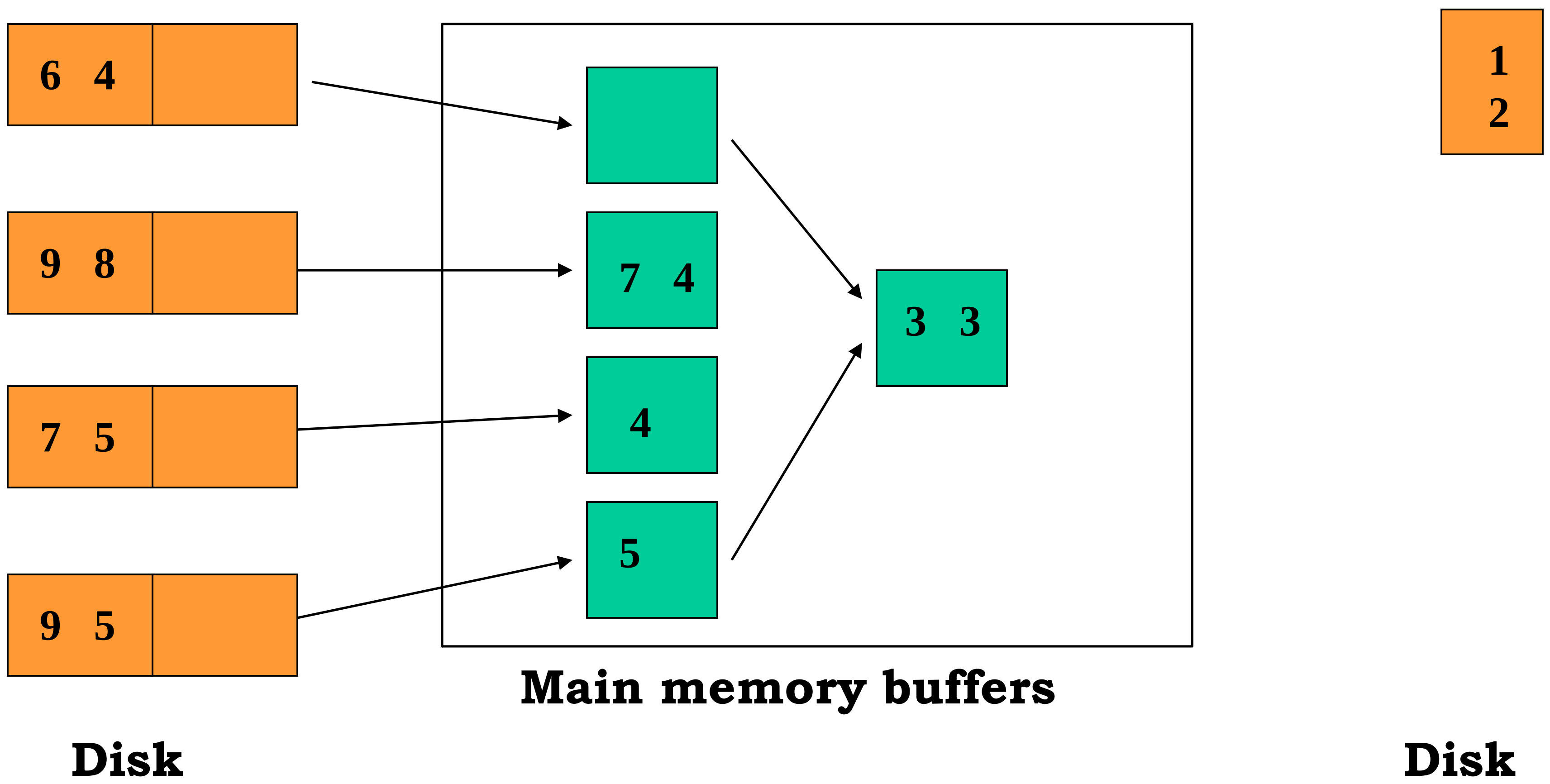
What if there are more memory?



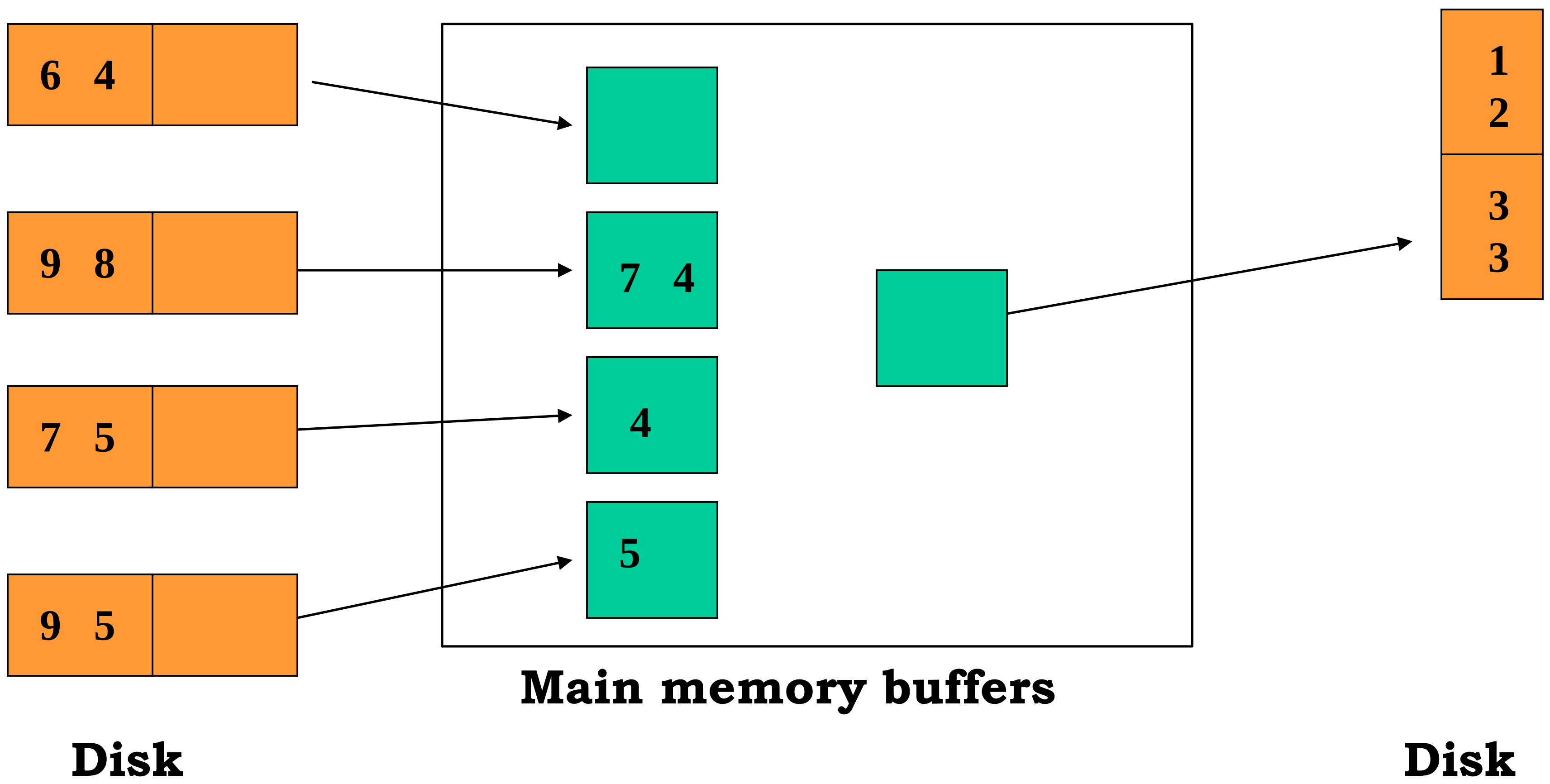
What if there are more memory?



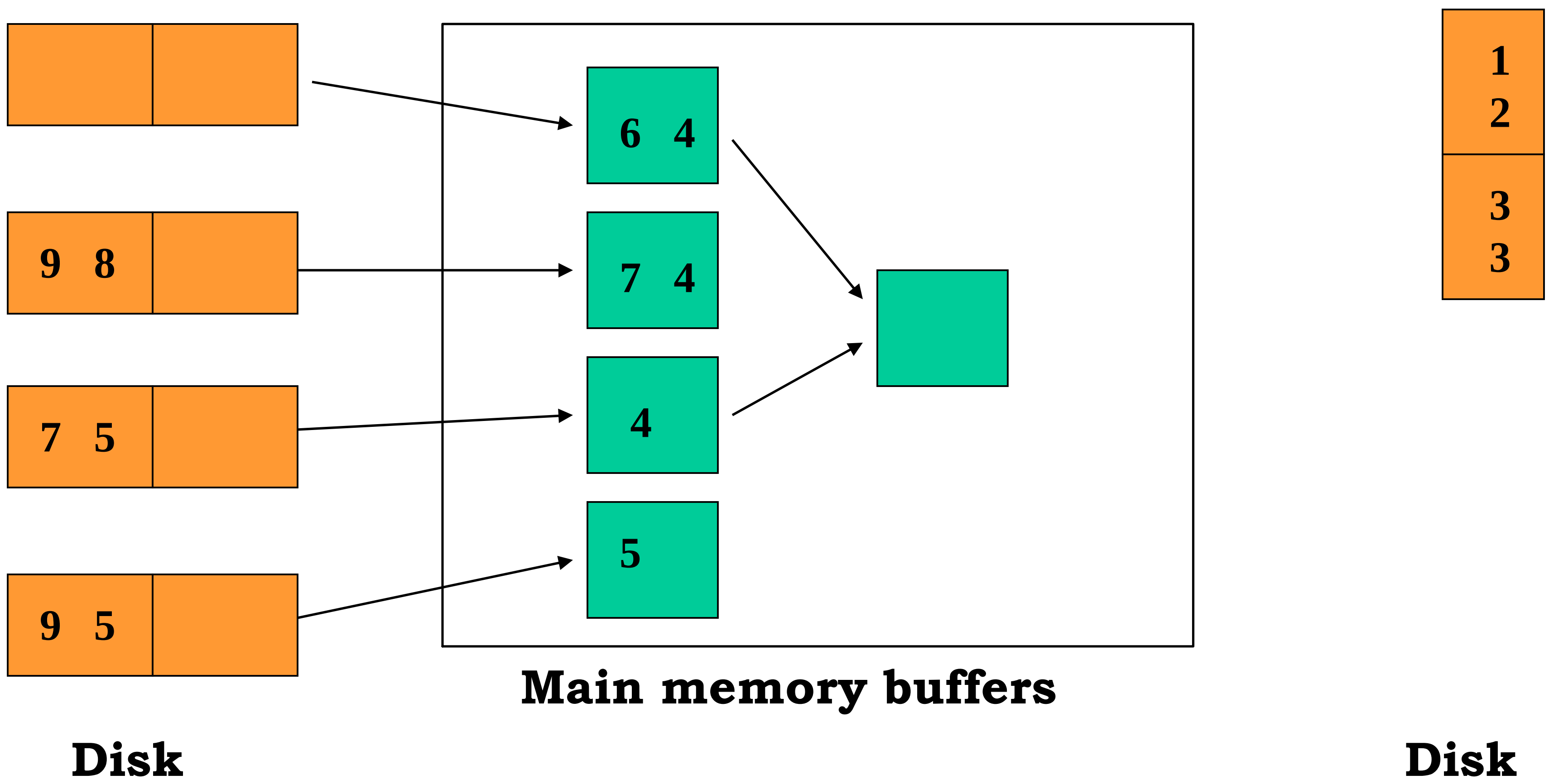
What if there are more memory?



What if there are more memory?



What if there are more memory?



So far …

- Given *k* sorted files (runs), we can *merge* them into larger sorted runs, and eventually produce *one* single sorted file
  - Regardless of the size of the sorted files
  - What about the memory size? What is the *minimum* number of pages needed (regardless of the number of runs)?

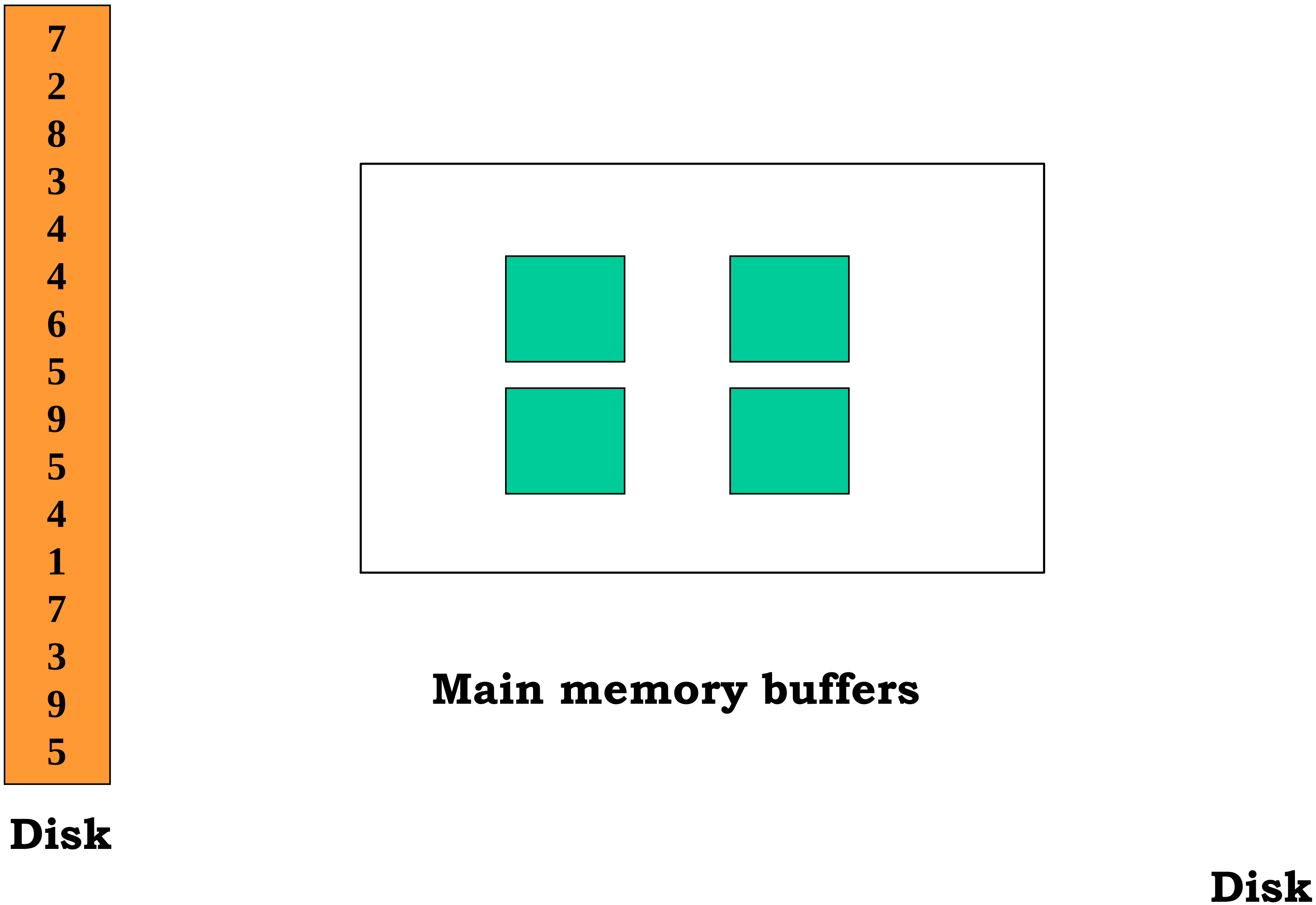
Multi-way Merge Sort

- To sort a very large (unsorted) file, we can do it in 2 phases:
  - Phase 1: Generate sorted runs
  - Phase 2: Merge sorted runs (we already know this)

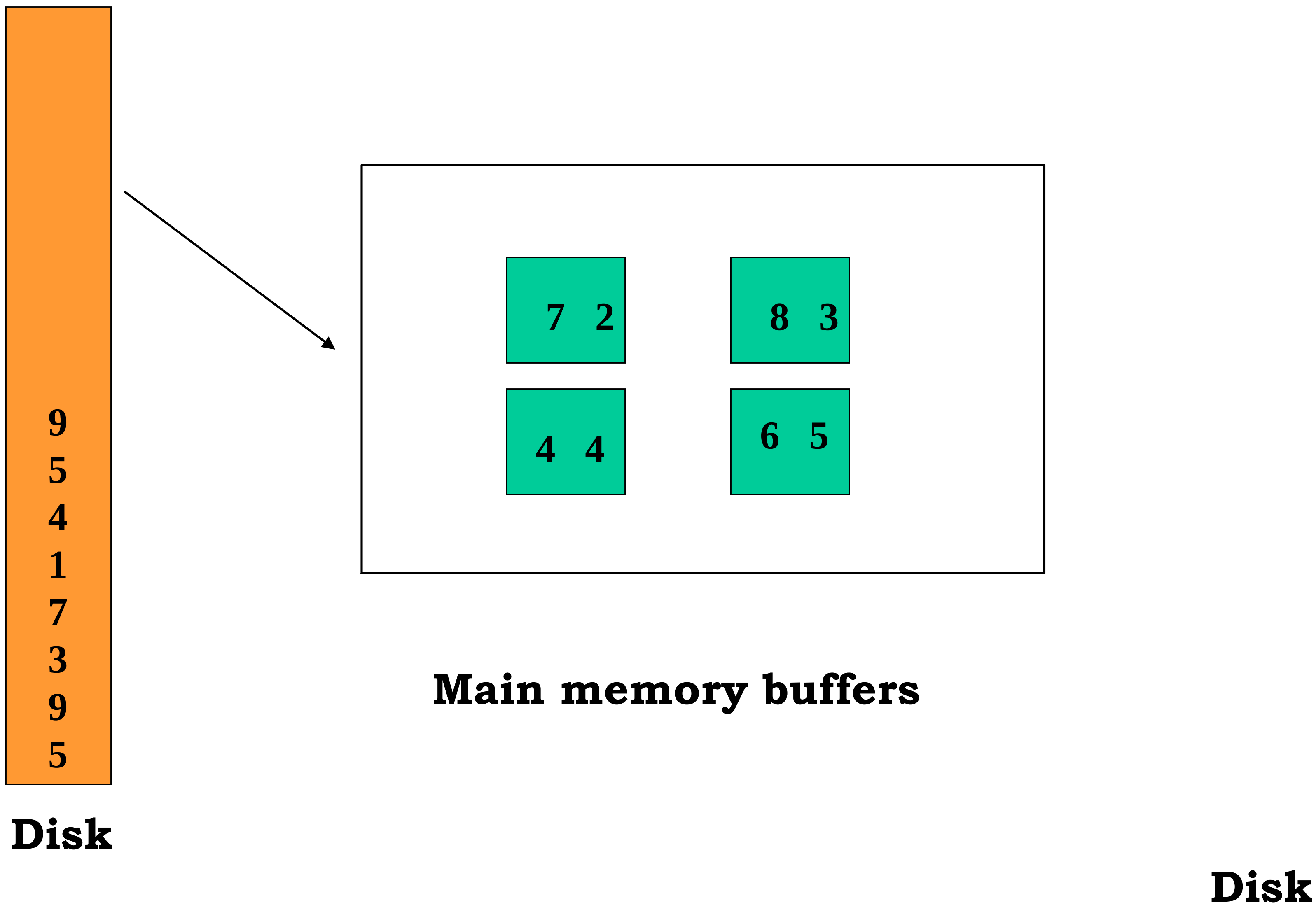
How to generate sorted runs?

- Read as many records as possible into memory
- Perform in-memory sort (you already know this)
- Write out sorted records as a sorted run
- Repeat the process until all records in the unsorted files are read

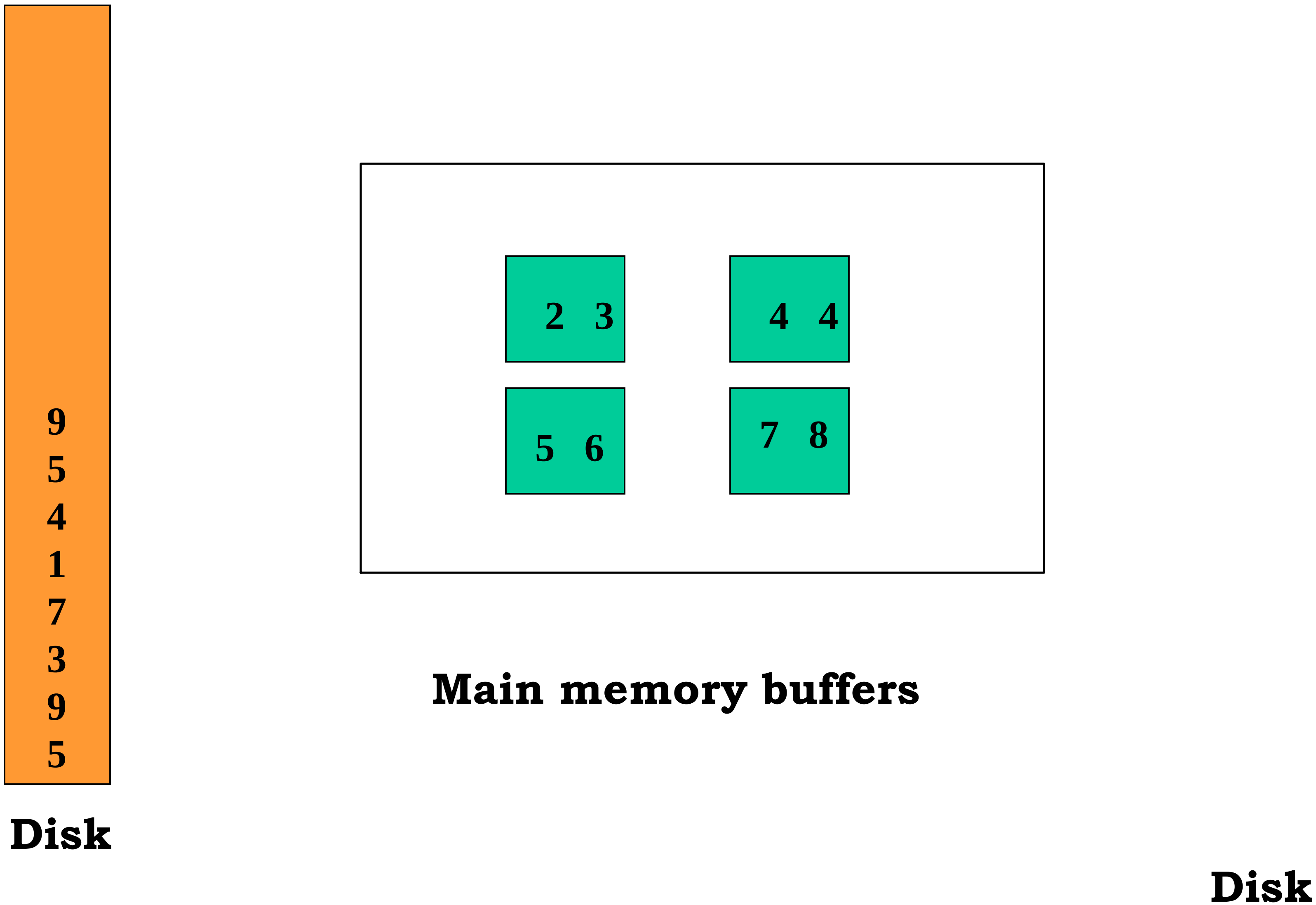
How to generate sorted runs?



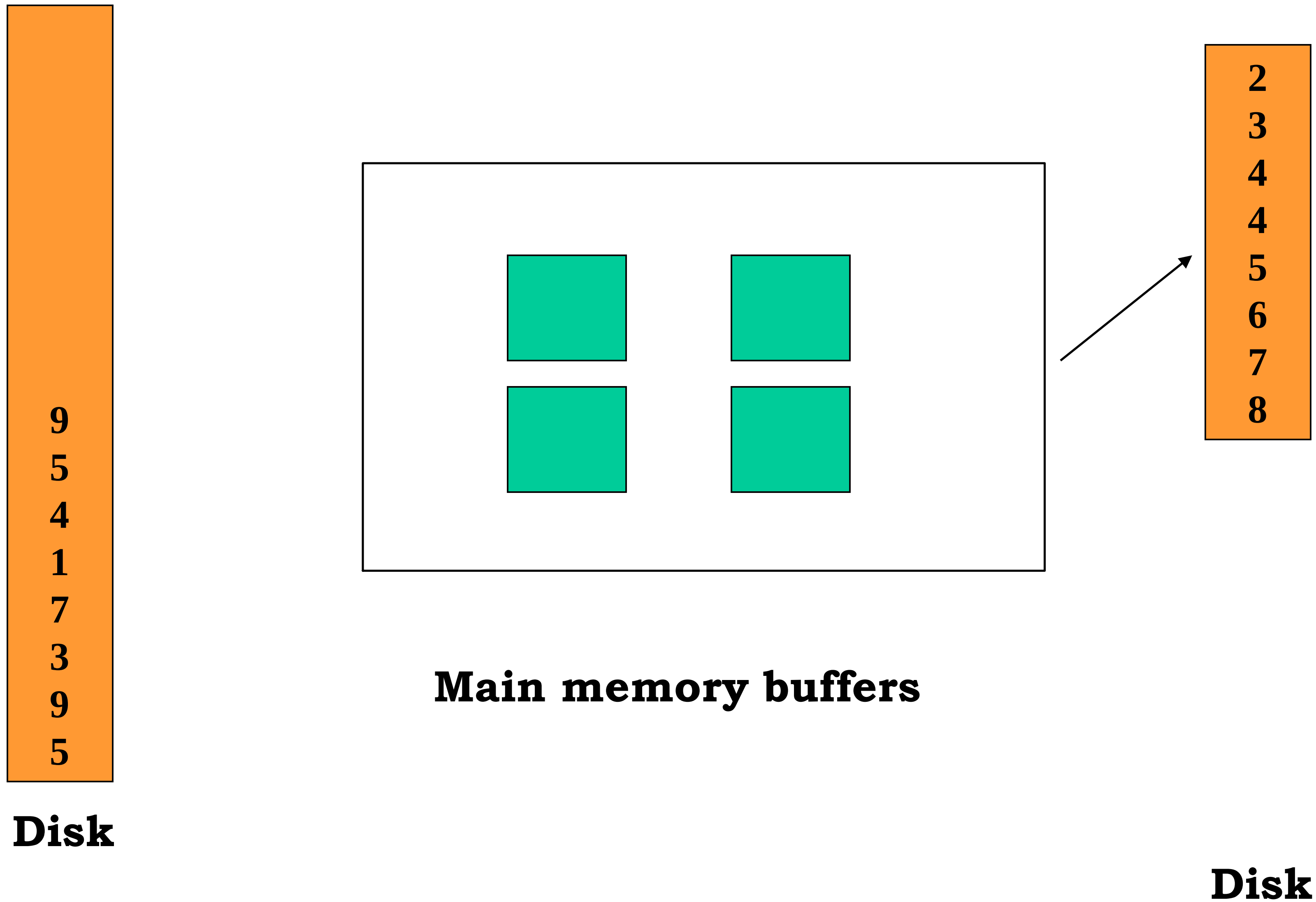
How to generate sorted runs?



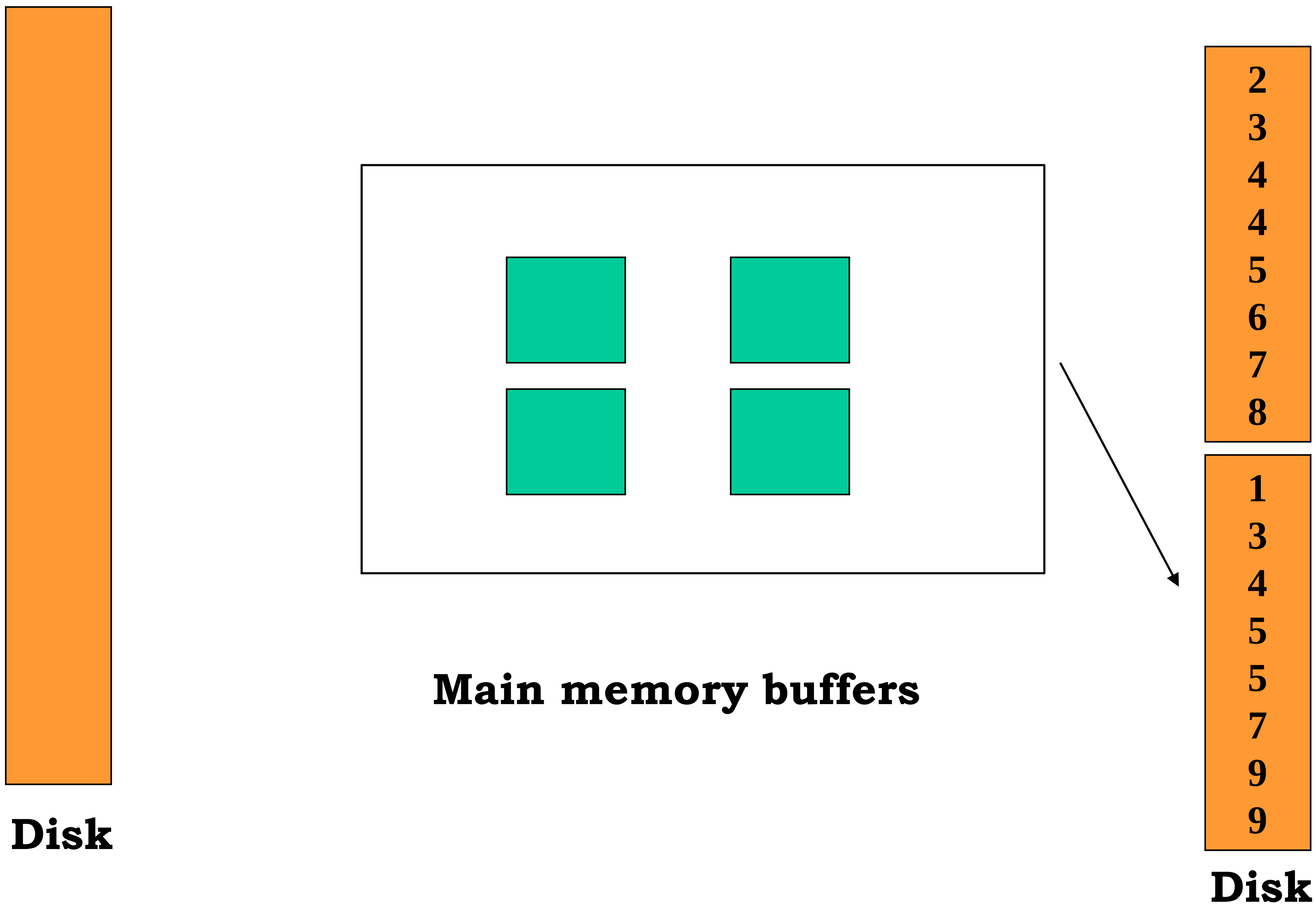
How to generate sorted runs?

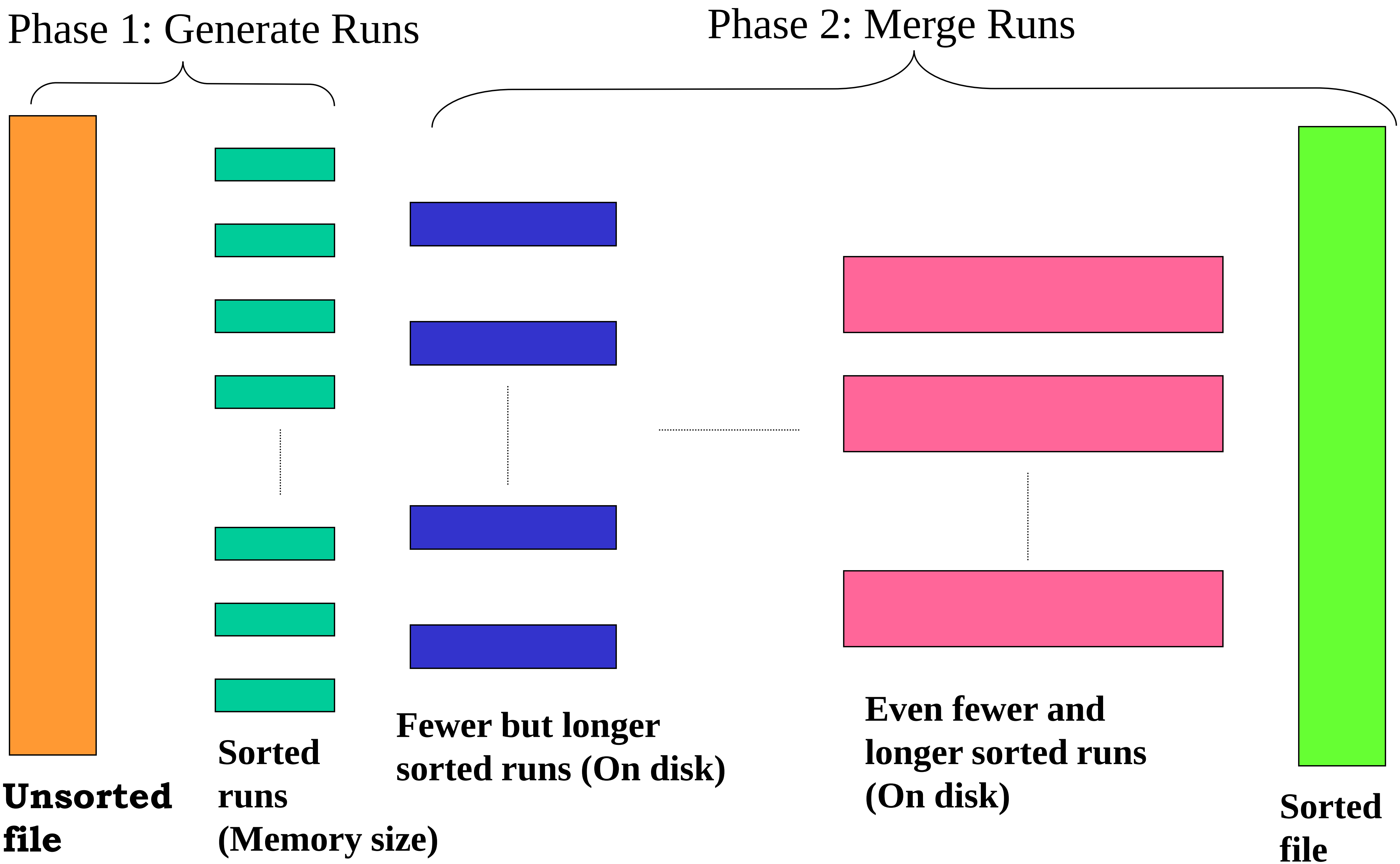


How to generate sorted runs?

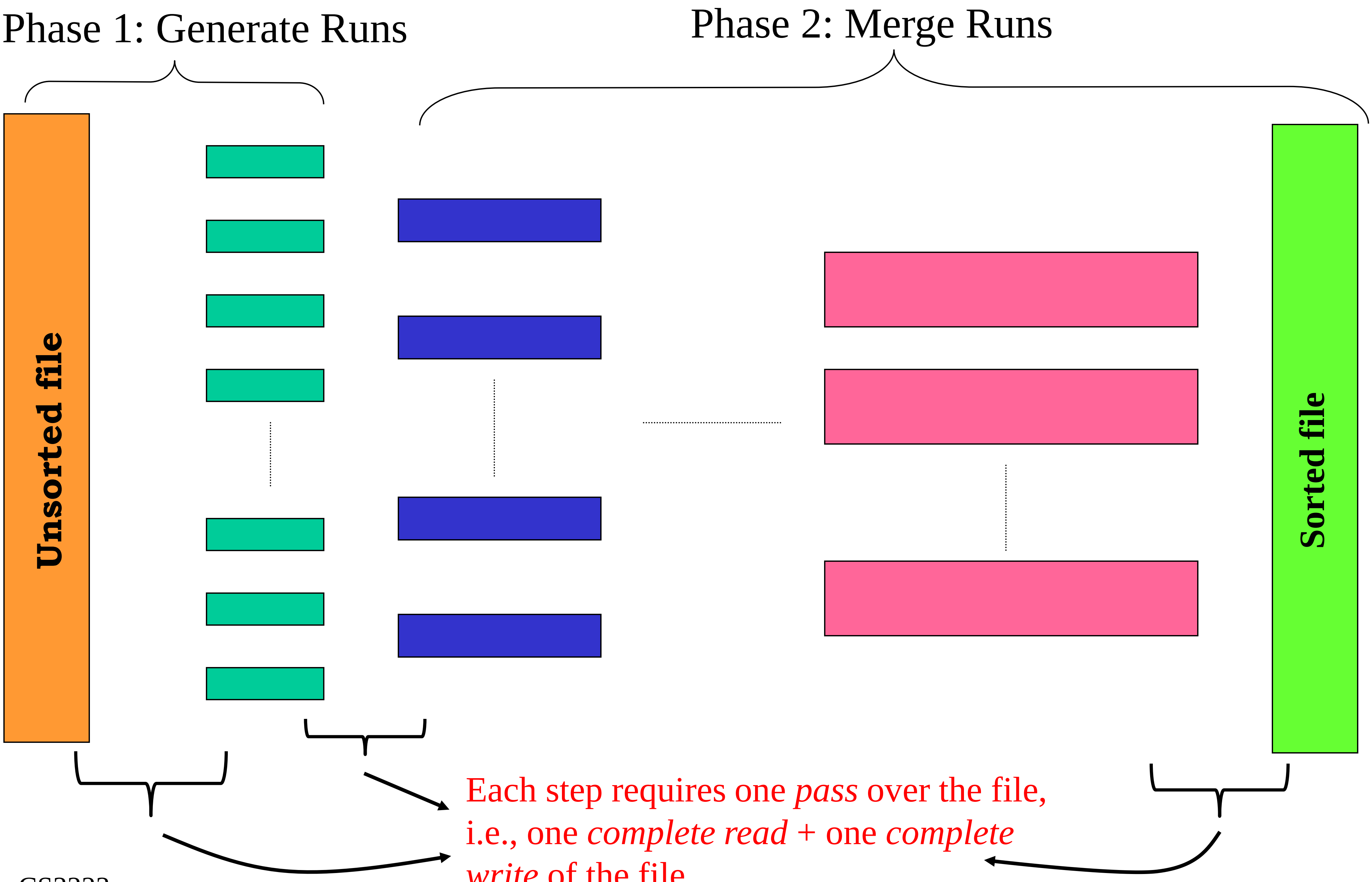


How to generate sorted runs?





Phase 1: Generate Runs
Phase 2: Merge Runs
Unsorted file
Sorted runs (Memory size)
Fewer but longer sorted runs (On disk)
Even fewer and longer sorted runs (On disk)
Sorted file


Phase 1: Generate Runs
Phase 2: Merge Runs
Unsorted file
Sorted file
Each step requires one pass over the file,
i.e., one complete read + one complete
write of the file

Multi-way Merge Sort

- To sort a file with *N* pages using *B* buffer pages ($B \geq 3$):
  - Phase 1: create sorted runs
    - Use *B* buffer pages
      - Read in B pages each time, sort the records, and produce a B page sorted run (except possibly for the last run)
      - Number of sorted runs = $\lceil N / B \rceil$
    - 1 **pass** (read + write) over the file (i.e., equal to 2*N I/Os)
  - Phase 2: Merge sorted runs
    - Use *B*-1 buffer pages for input and one buffer page for output
    - Perform (B-1)-way merge iteratively until one sorted run is produced
      - Each iteration requires 1 pass (read + write) over the file
    - $\lceil \log_{B-1} \lceil N / B \rceil \rceil$ passes

Cost of Multi-way Merge Sort

- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Cost = 2N * (# of passes)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Phase 1: $\lceil 108 / 5 \rceil$ = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Phase 2:
    - Pass 1: $\lceil 22 / 4 \rceil$ = 6 sorted runs of 20 pages each (last run is only 8 pages)
    - Pass 2: 2 sorted runs, 80 pages and 28 pages
    - Pass 3: Sorted file of 108 pages
  - Cost = (2*108) * 4 = 864

Number of Passes of External Sort

N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000						3
1,000,000						3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

A relatively small buffer size can sort a very large file with just a few passes

More buffers may not always be beneficial!

Internal Sort Algorithm

- Quicksort is a fast way to sort in memory
- Given N data pages, and B buffers, multi-way sort will generate $\lceil N/B \rceil$ runs
- Can we further reduce the number of runs (i.e., generate initial set of sorted runs whose run size is larger than the memory size)?
  - What's the advantage??
- An alternative is ***replacement selection***

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92
- suppose each block contains one record and B=5

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 34 68 45

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 45 $\longrightarrow$ 34

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 45 $\longrightarrow$ 34

60

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 ~~45~~ → 34

60 → 34 45

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 ~~45~~ → 34

60 → 34 45

2

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 ~~49~~ ~~34~~ 68 ~~45~~ ⟶ 34

60 ⟶ 34 45

2 ⟶ 34 45 49

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 ~~49~~ ~~34~~ 68 ~~45~~ → 34

~~60~~ → 34 45

2 → 34 45 49

38 → 34 45 49 60

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 ~~49~~ ~~34~~ ~~68~~ ~~45~~ → 34

~~60~~ → 34 45

2 → 34 45 49

38 → 34 45 49 60

28 → 34 45 49 60 68

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

~~109~~ ~~49~~ ~~34~~ ~~68~~ ~~45~~ $\longrightarrow$ 34

~~60~~ $\longrightarrow$ 34 45

2 $\longrightarrow$ 34 45 49

38 $\longrightarrow$ 34 45 49 60

28 $\longrightarrow$ 34 45 49 60 68

47 $\longrightarrow$ 34 45 49 60 68 109

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

~~109~~ ~~49~~ ~~34~~ ~~68~~ ~~45~~ ⟶ 34

~~60~~ ⟶ 34 45

2 ⟶ 34 45 49

38 ⟶ 34 45 49 60

28 ⟶ 34 45 49 60 68

47 ⟶ 34 45 49 60 68 109

16 38 28 47 2 ⟶ start a new run

Replacement Selection (Cont.)

- Final results:
  - 34 45 49 60 68 109
  - 2 16 19 28 35 38 47 55 76 78 86 98
  - 10 27 35 40 61 92
- Would have been 5 runs using Quicksort (requiring 2 passes in the merge phase)!

Internal Sort Algorithm: Replacement Selection

Read ***B*** blocks into memory

`Output:` move record with smallest sort key value, say *s*, to output buffer

Read in a new record *r*

if *r.key* > *s.key*, then **`GOTO Output`**
else ***freeze*** *r*

if all records in memory are frozen, then all records that have been output constitute a run; unfreeze all records and start a new run

`GOTO Output`

More on Replacement Selection

- Fact: average length of a run is *2B*
- Worst-Case
  - What is the (max) number of runs?
  - How does this arise?
- Best-Case
  - What is the (min) number of runs?
  - How does this arise?
- Quicksort is faster, but longer runs often means fewer passes!

Sequential vs Random I/Os

- Is minimizing passes *optimal*? Is merging as many runs as possible the best solution?
- Suppose we have 80 sorted runs, each 100 pages long and we have 81 pages of buffer space
- We can merge all 80 runs in a single (i.e., one) pass
  - Minimal number of passes!
- We can also merge the 80 runs in two passes
  - Pass 1: merge 16 runs first, resulting in 5 longer runs
  - Pass 2: merge the 5 runs
- Which is better??

Sequential vs Random I/Os

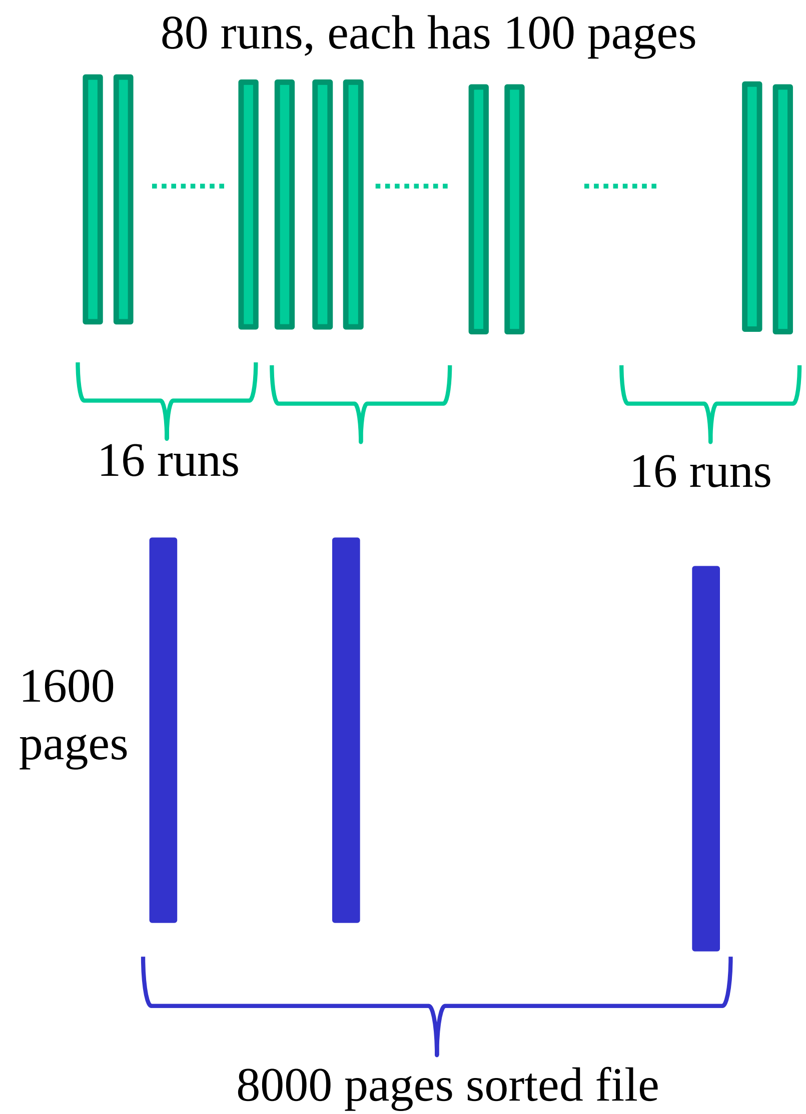
- Sequential I/O
  - If k pages are accessed sequentially, we need only 1 seek for the k pages (assuming the k pages are stored consecutively on the same track) plus retrieving the k pages sequentially
- Random I/O
  - If k pages are stored at k/2 random locations such that 2 pages are stored consecutively, then we need k/2 seeks plus k page accesses

Sequential vs Random I/Os

- Suppose we have 80 sorted runs, each 100 pages long and we have 81 pages of buffer space
- We can merge all 80 runs in a single (i.e., one) pass
  - each page requires a seek to access (Why?)
  - there are 100 pages per run, so 100 seeks per run
  - total cost = 80 runs X 100 seeks = 8000 seeks

Sequential vs Random I/Os (Cont)

- We can merge all 80 runs in two steps
  - 5 sets of 16 runs each
    - read 80/16=5 pages of one run
    - 16 runs result in sorted run of 1600 pages
    - each merge requires 100/5X16 = 320 seeks
    - for 5 sets, we have 5X320 = 1600 seeks
  - merge 5 runs of 1600 pages
    - read 80/5=16 pages of one run
      => 1600/16=100 seeks in total
    - 5 runs => 5X100 = 500 seeks
  - total: 1600+500=2100 seeks!!!
- Number of passes increases,
  but number of seeks decreases!

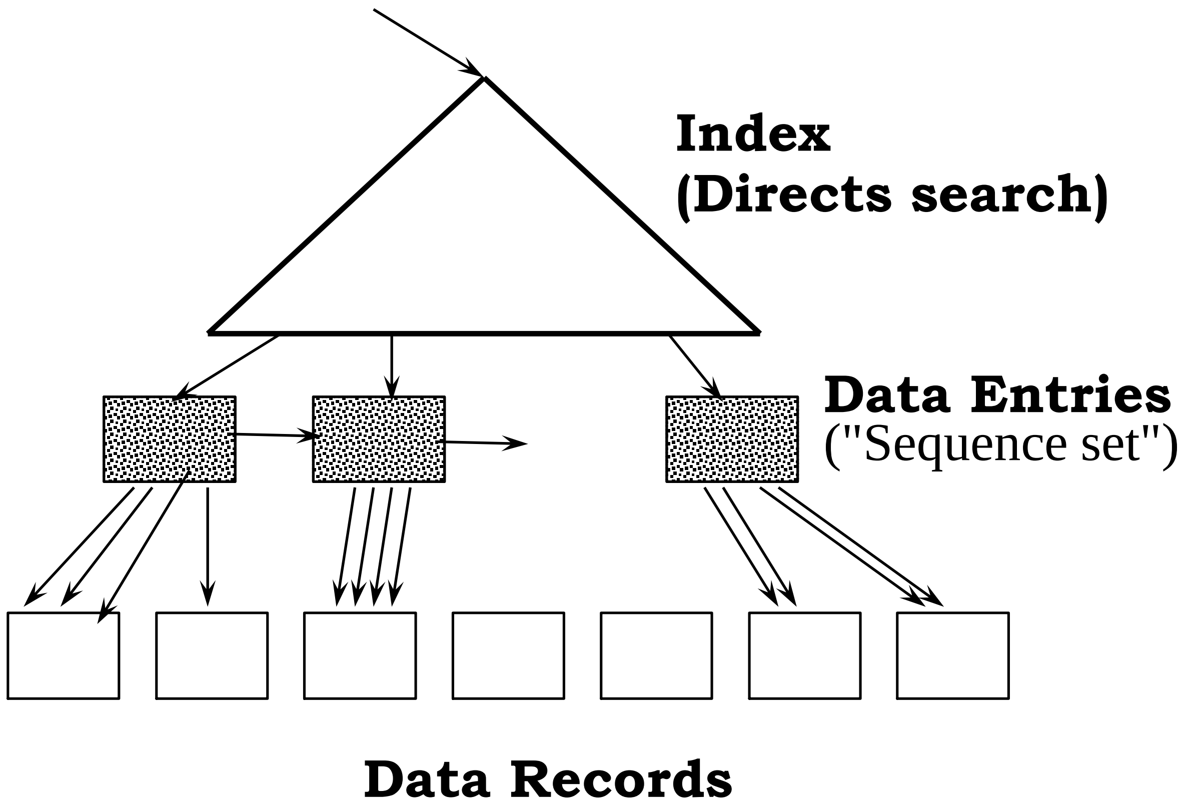


Using B+ Trees for Sorting

- Scenario: Table to be sorted has B^+ tree index on sorting column(s)
- Idea: Can retrieve records in order by traversing leaf pages
- ***Is this a good idea?***

Clustered B+ Tree Used for Sorting

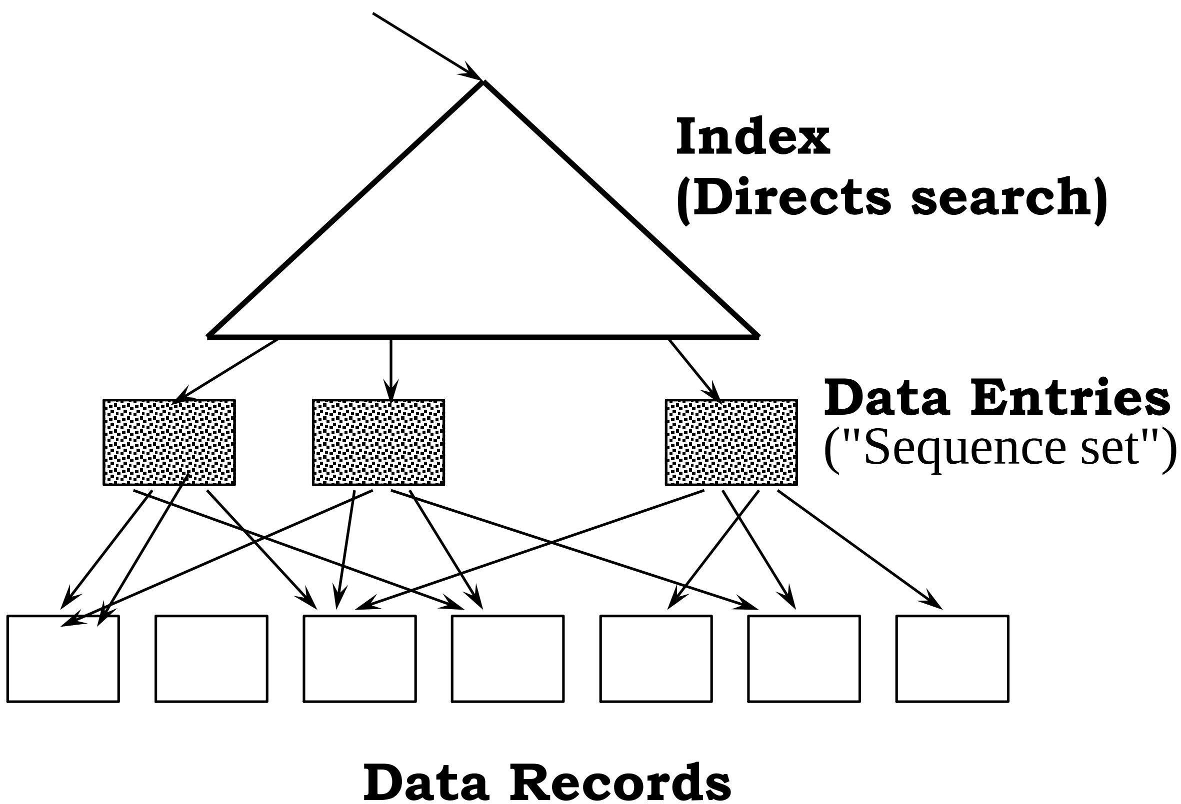
- Cost: root to the left-most leaf, then retrieve all leaf pages (<key,record> pair organization)
- If <key, rid> pair organization is used? Additional cost of retrieving data records: each data page fetched just once



Always better than external sorting!??

Unclustered B+ Tree Used for Sorting

- each data entry contains <key,*rid* > of a data record. In general, ***one I/O per data record!***



Does it mean that this will never be used to retrieve records in sorted order???

External Sorting vs. Unclustered Index

N	Sorting	p=1	p=10	p=100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	400,000	100,000	1,000,000	10,000,000
1,000,000	6,000,000	1,000,000	10,000,000	100,000,000
10,000,000	60,000,000	10,000,000	100,000,000	1,000,000,000

*** *p*: # of records per page**

**** B=1,000***

**** p=100 is the more realistic value***

Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting!
- External merge sort minimizes disk I/O cost:
  - (Phase 1) First pass of file: Produces sorted ***runs*** of size ***B*** (# buffer pages)
    - Replacement selection can reduce the number of runs
  - (Phase 2) Subsequent passes: ***merge*** runs
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad