External Sorting

External Sorting

- Problem: Sort 1GB of data with 1MB of RAM.
- When a file doesn't fit in memory, there are two stages in sorting:
  1. File is divided into several segments, each of which sorted separately
  2. Sorted segments are *merged.*

  (Each stage involves reading and writing the file at least once)

How to sort segments

- Any efficient sorting algorithm (such as Quicksort) can be used to sort segments
- Each sorted segment is called a **run**.
- Each run will be the size of the available memory.

External sort for Large Files

Basic idea

1. Form runs (i.e. sorted segments):
   - bring as many records as possible to main memory, sort them, save it into a small file on disk.
   - Repeat this until we have read all records from the original file and written them as sorted segments (i.e. runs) to disk.
2. Do a multiway merge of the runs.

External sort for Large Files (cont.)

- How big is each run?
  - As big as the available memory.
- What is the time it takes to create all sorted segments?
  - Ignoring the seek time and assuming *b* blocks in the file

The time for creating the initial sorted segments is **2*b**btt** (read in the file as segments and write out the runs)

- *Note that the entire file has not been sorted yet.*
- *These are just sorted segments, and*
- *the size of each segment is limited to the size of the available memory used for this purpose.*

Multiway Merging

- K-way merge: we want to merge K sorted input lists to create a single sorted output list. (K is the order of a K-way merge)

- We will adapt the 2-way merge algorithm:
  - Instead of two lists, keep an array of lists: list[0], list[1], … list[k-1]

Example -- Step 1: Create Runs

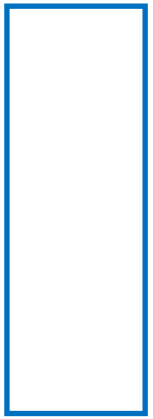
Unsorted file

21
110
27
5
108
9

109
11
25
107
2
4

10
22
3
1
4
21

Memory (buffer) of size 6



Example -- Step 1: Create Runs

Unsorted file

21
110
27
5
108
9
109
11
25
107
2
4
10
22
3
1
4
21

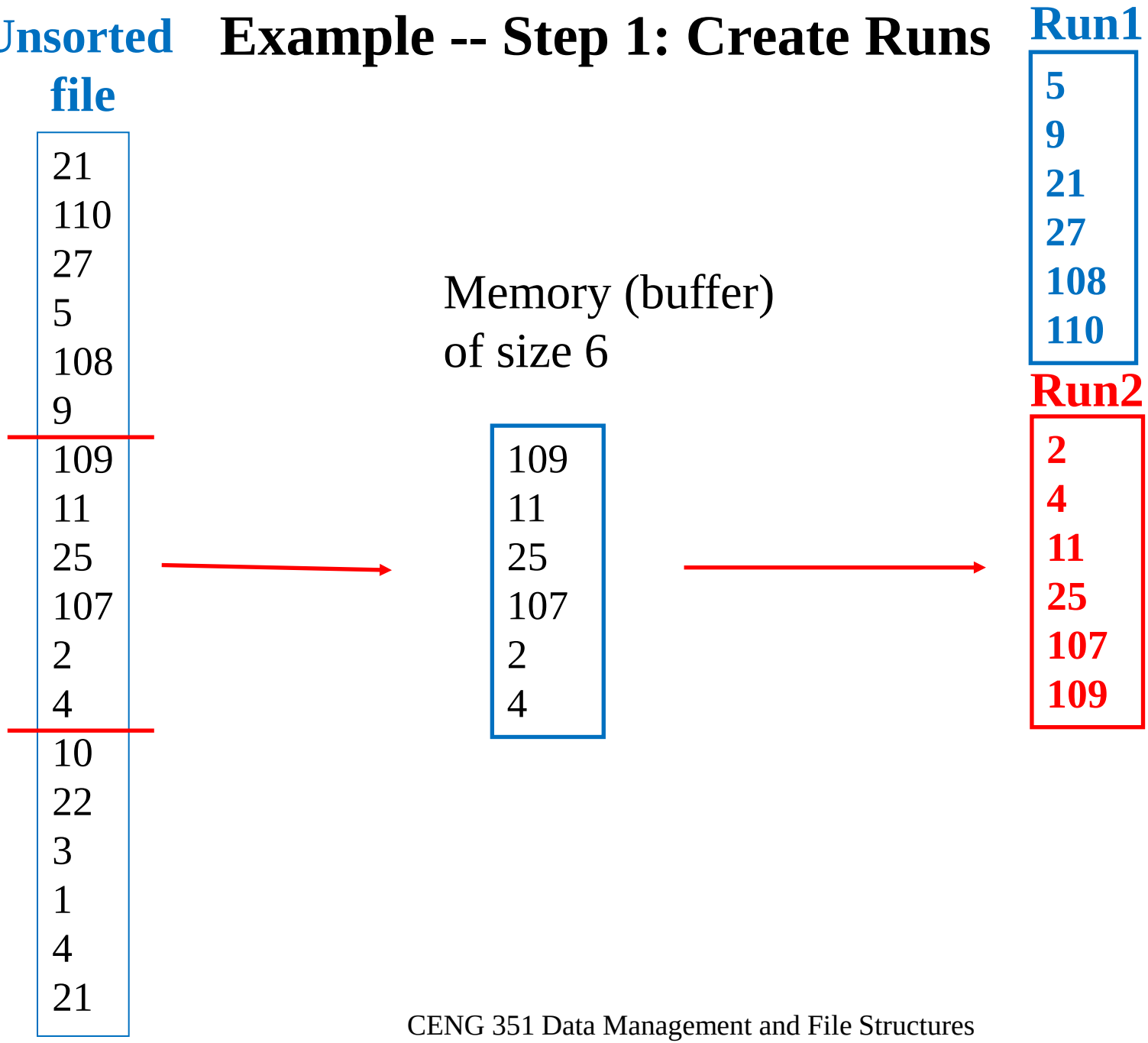
Memory (buffer) of size 6

21
110
27
5
108
9

Run1

5
9
21
27
108
110

Example -- Step 1: Create Runs



Example -- Step 1: Create Runs

Unsorted file

21
110
27
5
108
9
109
11
25
107
2
4
10
22
3
1
4
21

Memory (buffer) of size 6

10
22
3
1
4
21

Run1

5
9
21
27
108
110

Run2

2
4
11
25
107
109

Run3

1
3
4
10
21
22

Example -- Step 2: K-way Merge

Run1

5
9
21
27
108
110

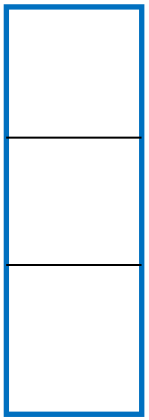
Run2

2
4
11
25
107
109

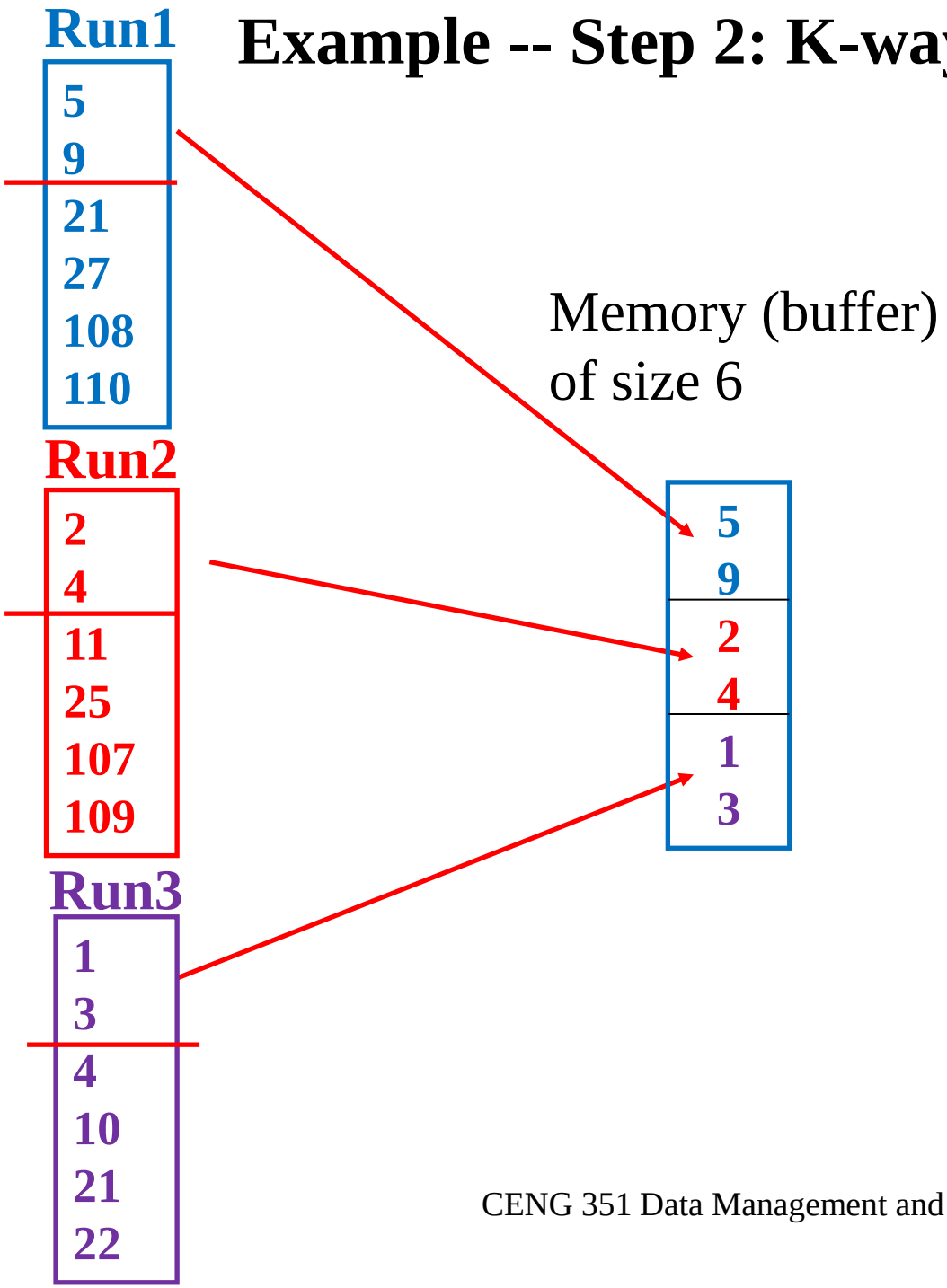
Run3

1
3
4
10
21
22

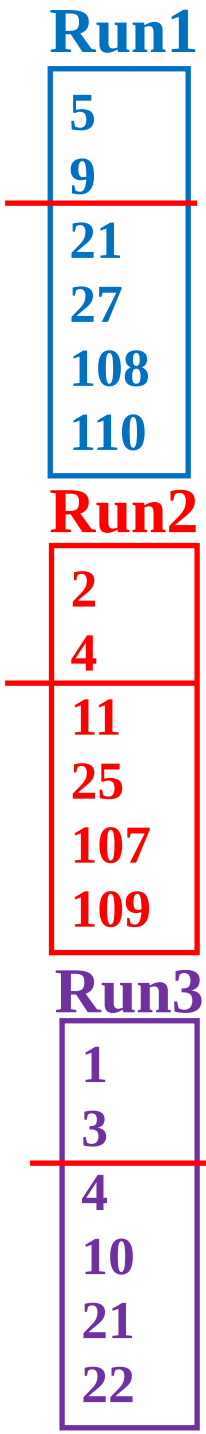
Memory (buffer)
of size 6



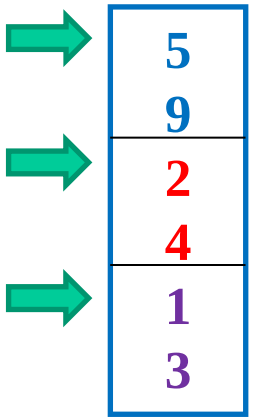
Example -- Step 2: K-way Merge

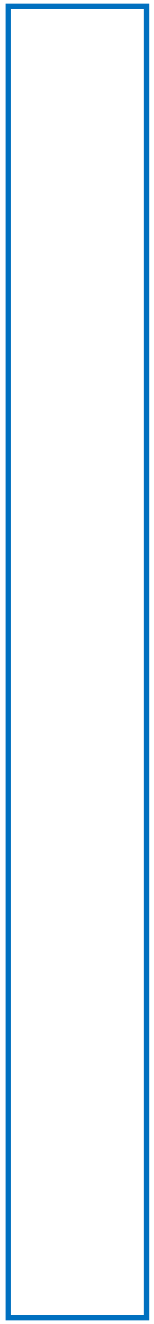


Example -- Step 2: K-way Merge

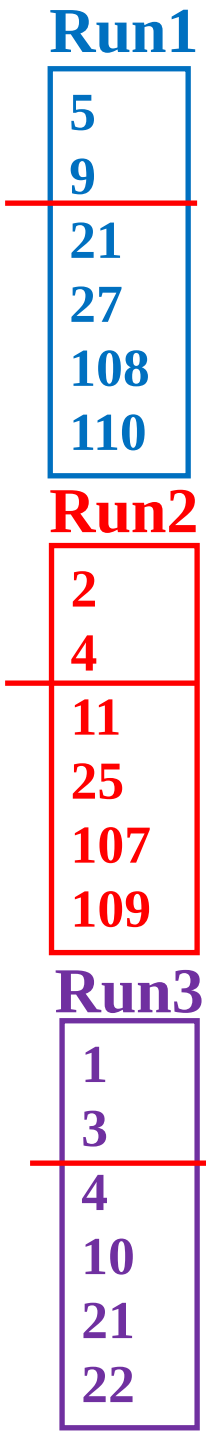


Memory (buffer)
of size 6

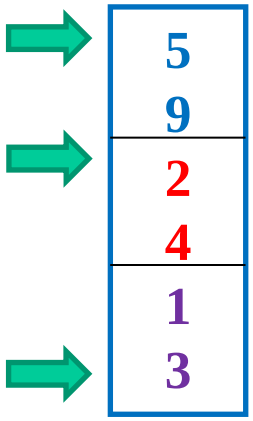


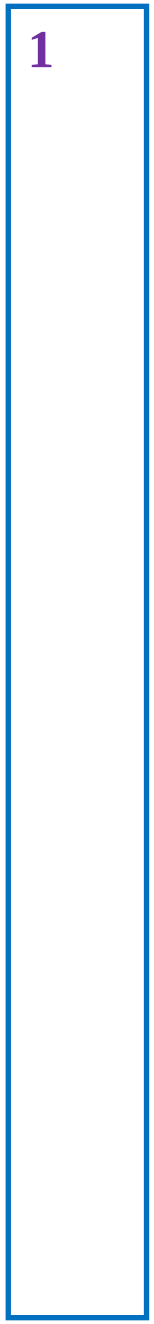


Example -- Step 2: K-way Merge

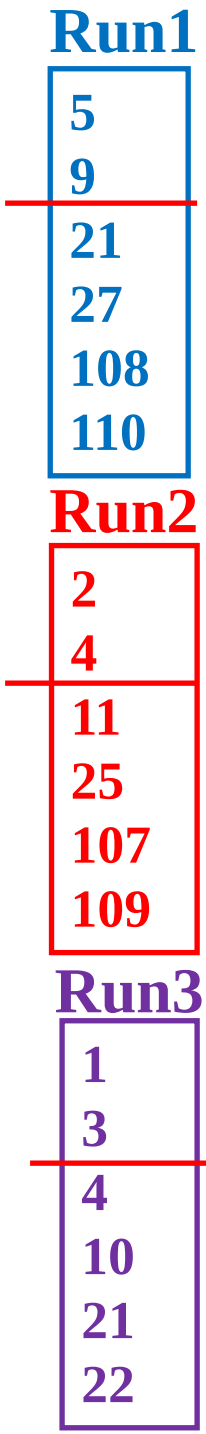


Memory (buffer)
of size 6

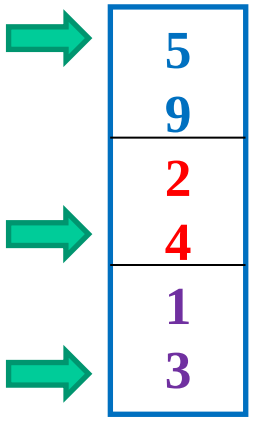


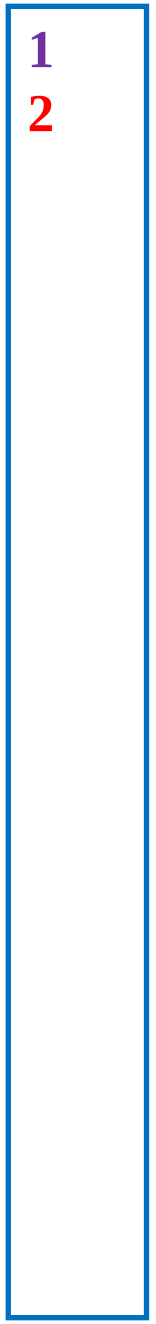


Example -- Step 2: K-way Merge

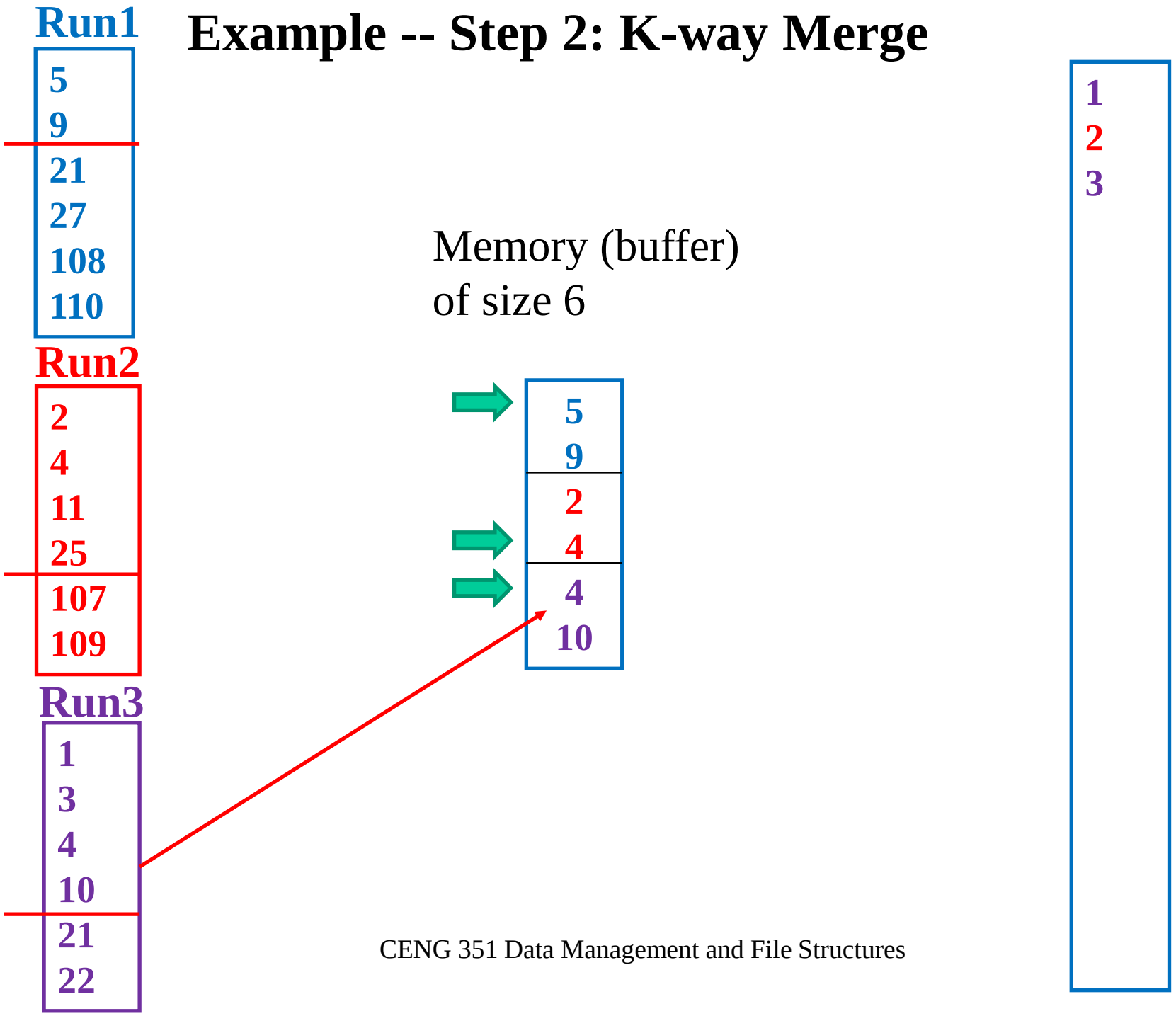


Memory (buffer)
of size 6

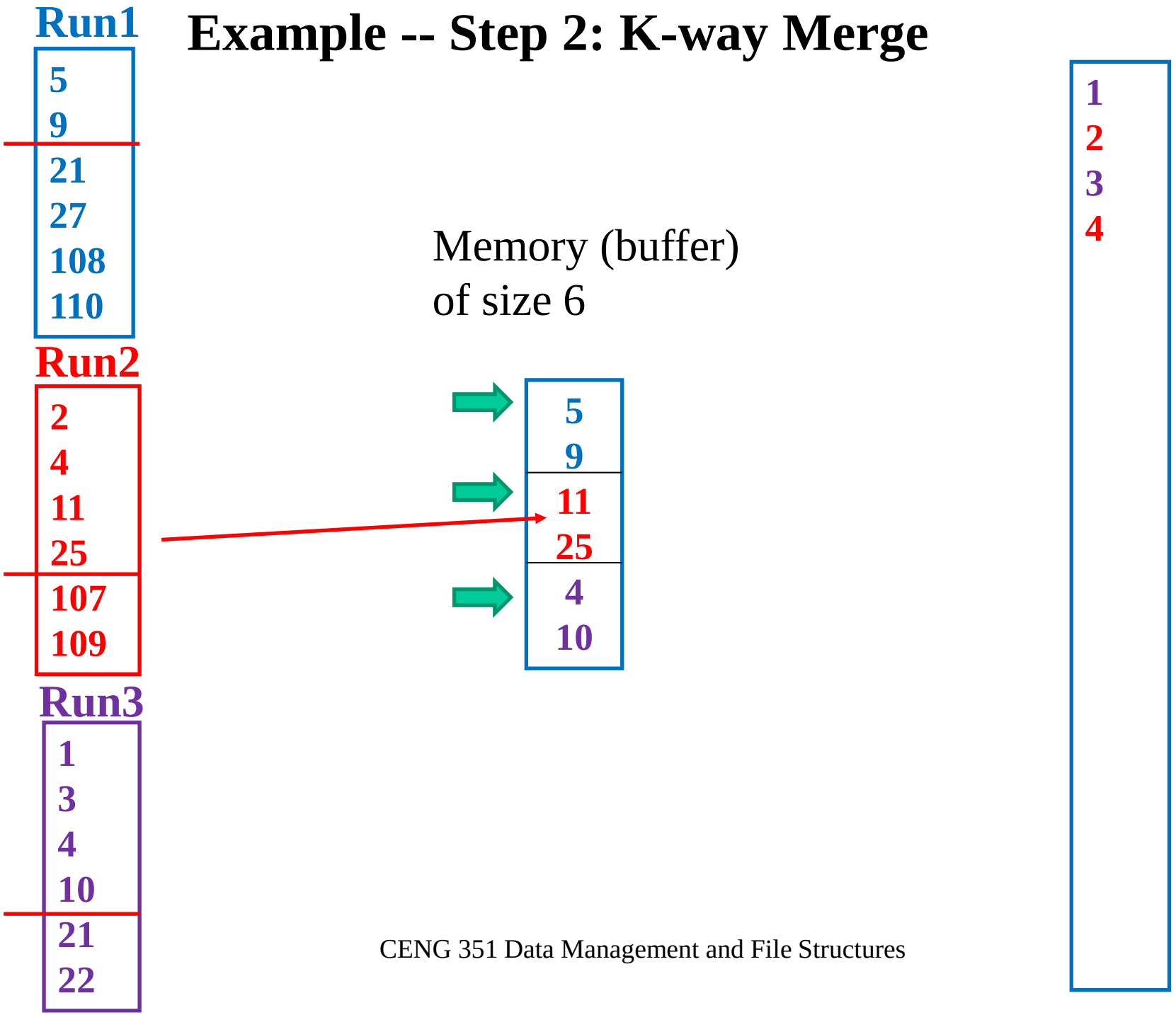




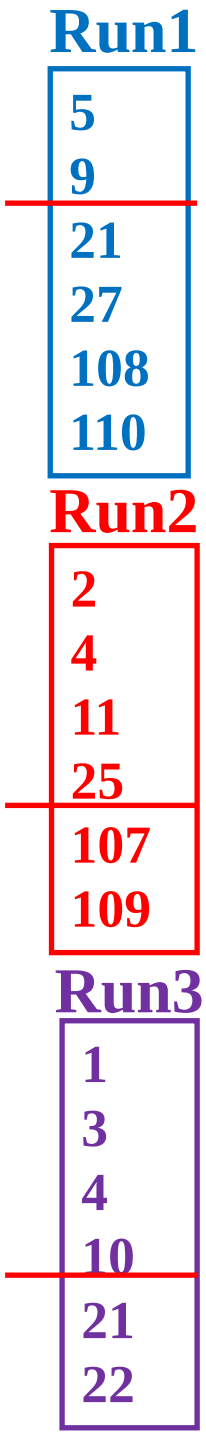
Example -- Step 2: K-way Merge



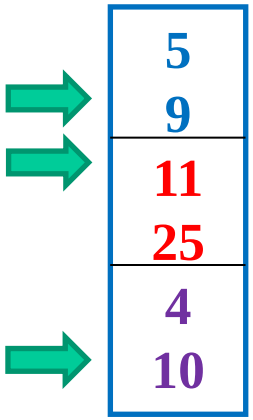
Example -- Step 2: K-way Merge

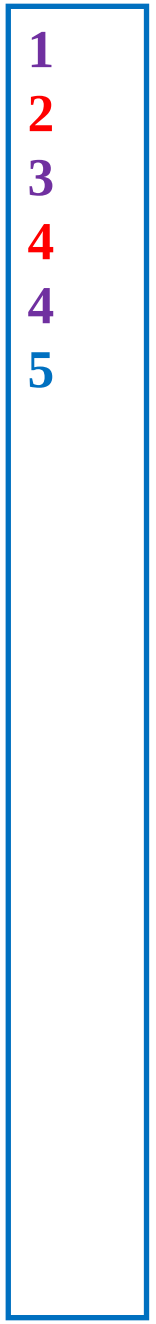


Example -- Step 2: K-way Merge

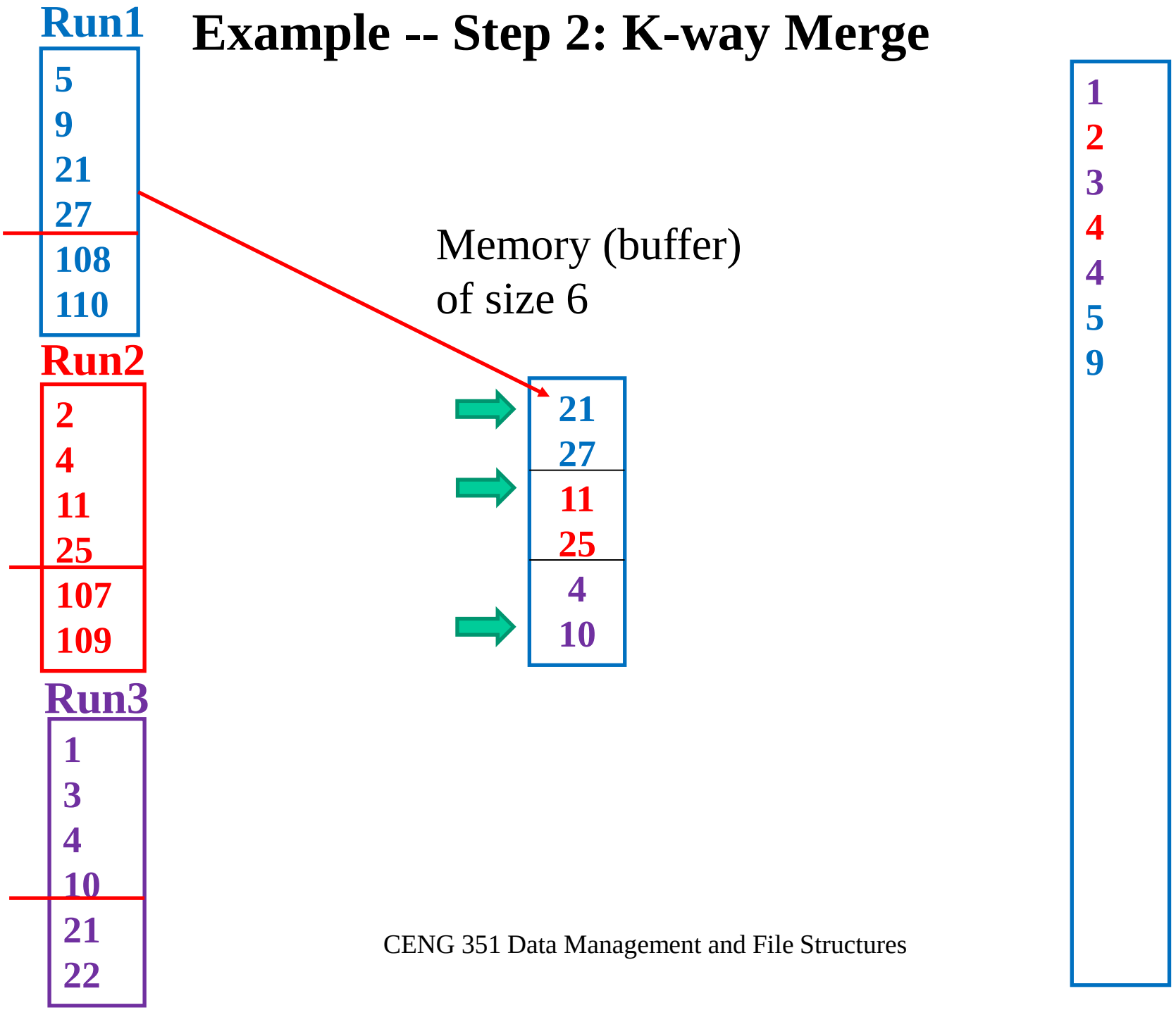


Memory (buffer)
of size 6

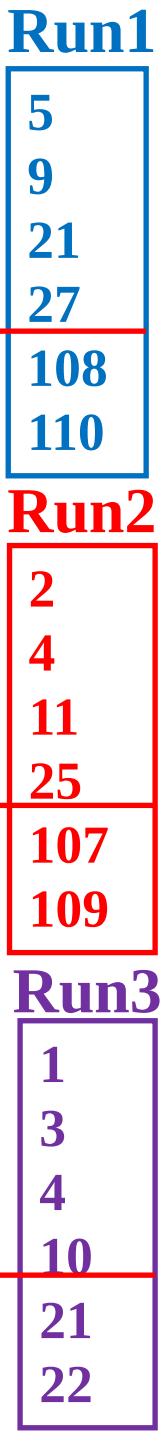




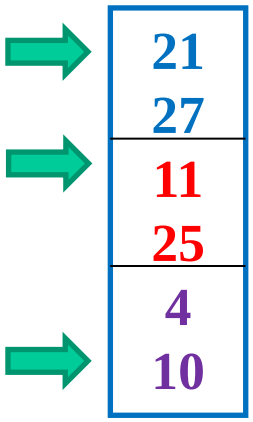
Example -- Step 2: K-way Merge



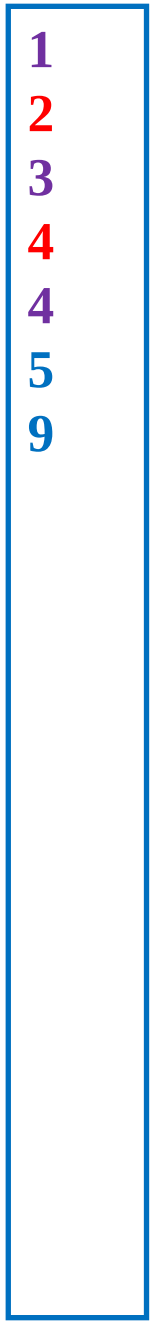
Example -- Step 2: K-way Merge



Memory (buffer)
of size 6



Continues…



Example -- Step 2: K-way Merge

Run1

5
9
21
27
108
110

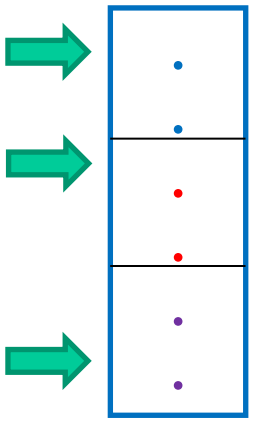
Run2

2
4
11
25
107
109

Run3

1
3
4
10
21
22

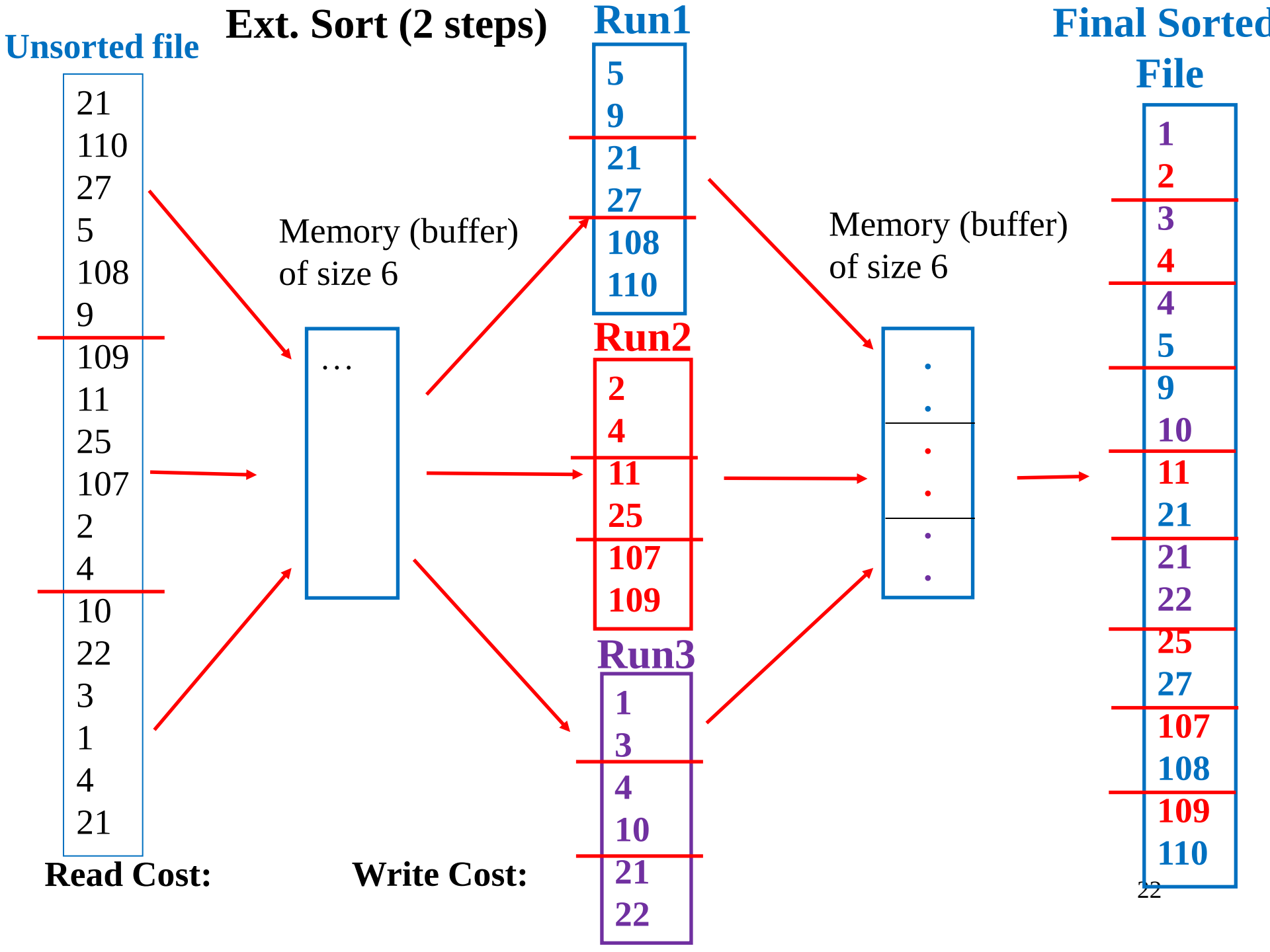
Memory (buffer)
of size 6

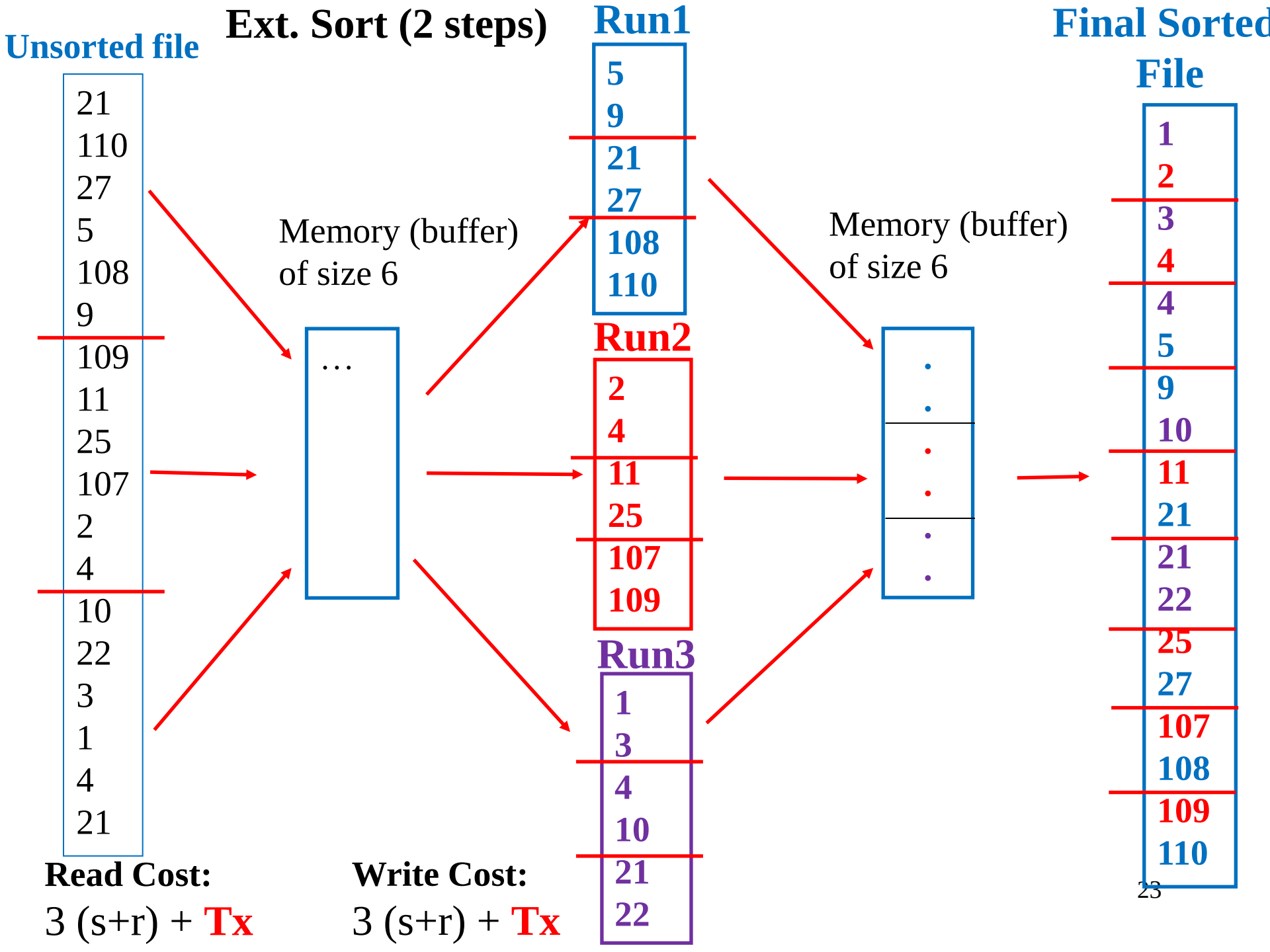


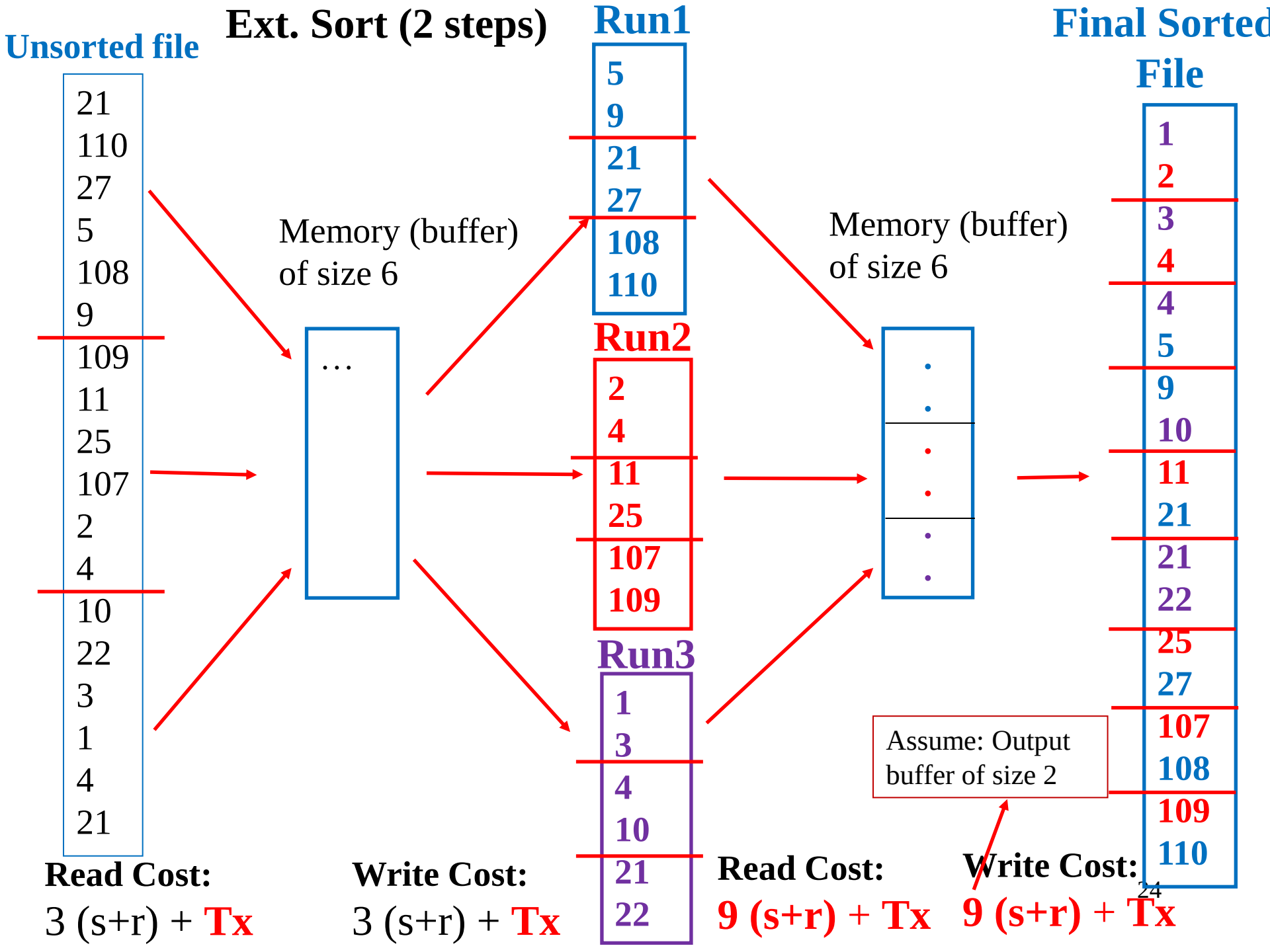
At the end

We obtain the
sorted file

1
2
3
4
4
5
9
10
11
21
21
22
25
27
107
108
109
110


Ext. Sort (2 steps)
Unsorted file
21
110
27
5
108
9
109
11
25
107
2
4
10
22
3
1
4
21
Memory (buffer)
of size 6
…
Run1
5
9
21
27
108
110
Run2
2
4
11
25
107
109
Run3
1
3
4
10
21
22
Memory (buffer)
of size 6
Final Sorted
File
1
2
3
4
4
5
9
10
11
21
21
22
25
27
107
108
109
110
Read Cost:
Write Cost:


Ext. Sort (2 steps)
Unsorted file
21
110
27
5
108
9
109
11
25
107
2
4
10
22
3
1
4
21
Memory (buffer)
of size 6
…
Run1
5
9
21
27
108
110
Run2
2
4
11
25
107
109
Run3
1
3
4
10
21
22
Memory (buffer)
of size 6
Final Sorted
File
1
2
3
4
4
5
9
10
11
21
21
22
25
27
107
108
109
110
Read Cost:
3 (s+r) + Tx
Write Cost:
3 (s+r) + Tx


Ext. Sort (2 steps)
Unsorted file
21
110
27
5
108
9
109
11
25
107
2
4
10
22
3
1
4
21
Memory (buffer) of size 6
…
Run1
5
9
21
27
108
110
Run2
2
4
11
25
107
109
Run3
1
3
4
10
21
22
Memory (buffer) of size 6
Final Sorted File
1
2
3
4
4
5
9
10
11
21
21
22
25
27
107
108
109
110
Assume: Output buffer of size 2
Read Cost:
3 (s+r) + Tx
Write Cost:
3 (s+r) + Tx
Read Cost:
9 (s+r) + Tx
Write Cost:
9 (s+r) + Tx

Multiway Merging to sort Large Files

Let us consider the following example:

- File to be sorted:
  - 8,000,000 records
  - R = 100 bytes

$\Rightarrow$Total file size = 800MB

- Memory available as a work area: 10MB (not counting memory used to hold program, other variables, O.S., I/O buffers etc.)

Step 1: Create Runs

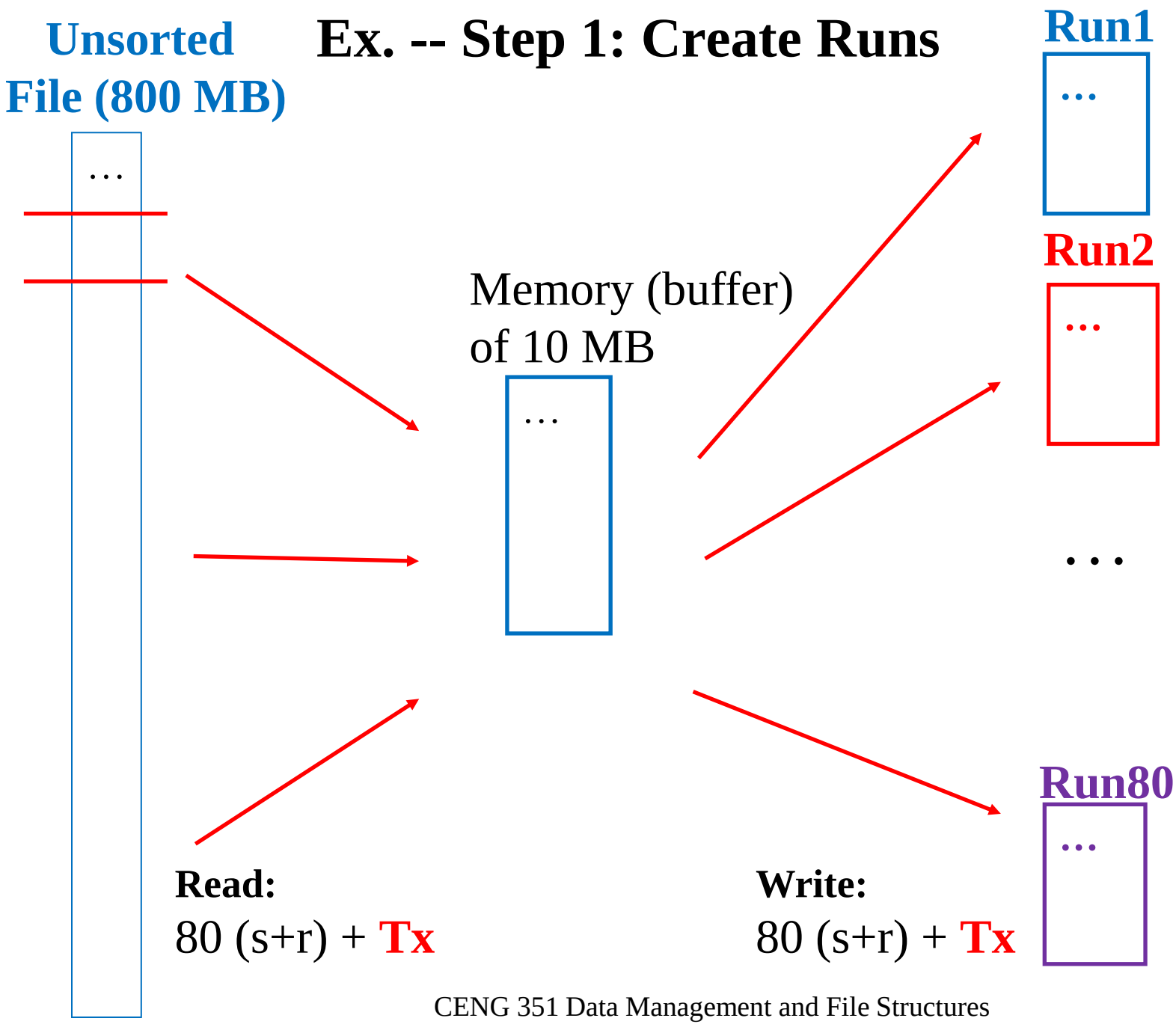
Disk I/Os are performed in the following procedures:

1. Reading records into main memory for sorting
2. Writing runs to disk.

These two steps are done as follows:

- Read 10MB, write 10MB (repeat this 80 times, because file is 800MB). So there will be 80 runs
  - In terms of basic disk operations, this operation costs:
    - For reading: (80 seeks +) a total transfer time of 800 MB **(T_X)**
    - Same for writing: (80 seeks +) a total transfer time of 800 MB **(T_X)**

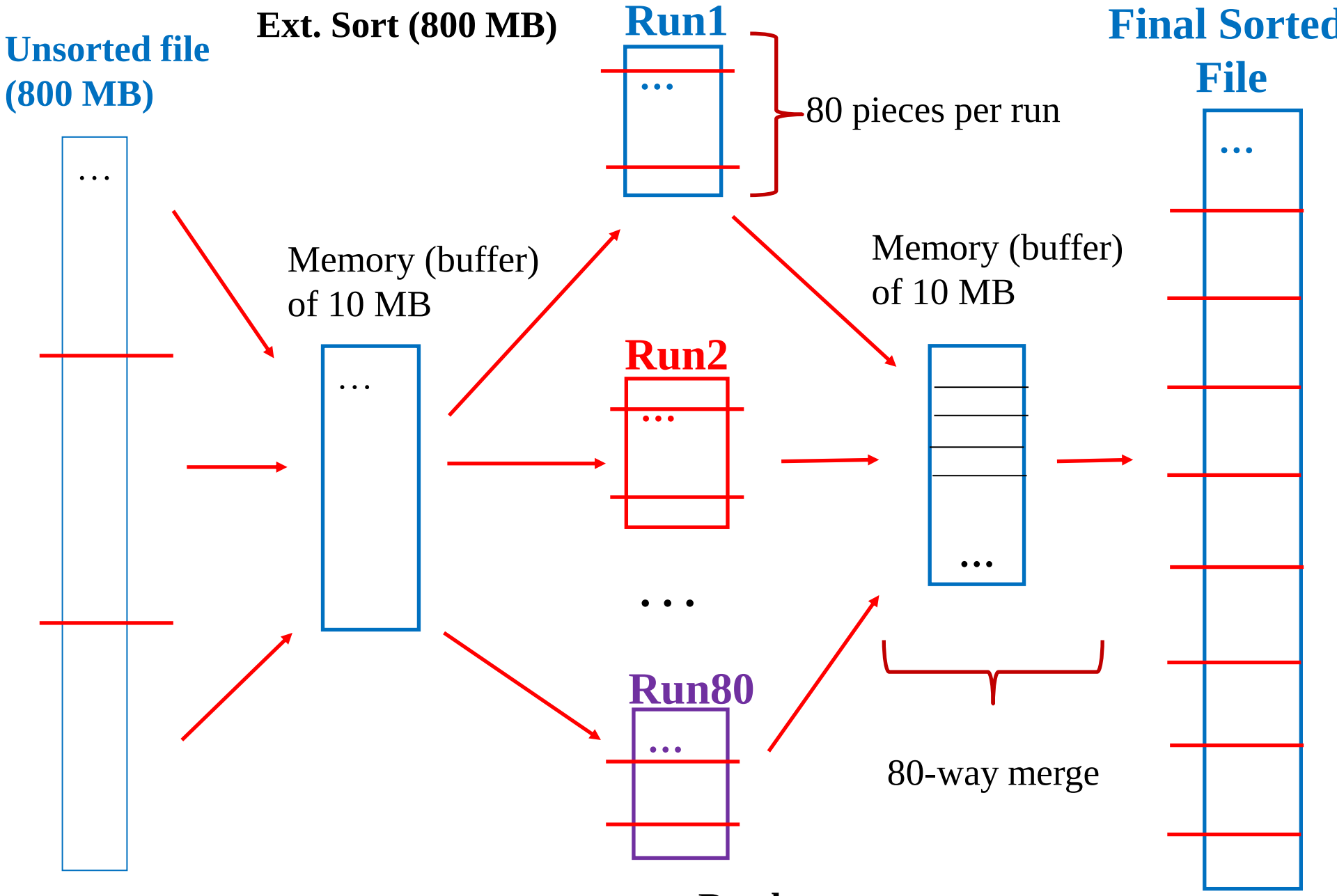
Ex. -- Step 1: Create Runs



Step 2: Multiway Merging

- Read runs into memory for merging.
  - Available memory is 10 M and each run size is 10MB.
  - There are 80 runs.
  - We need to do a 80-way merge.
  - 10M / 80 = 125,000 bytes are available for each list to be merged.
- Read one piece of each run.
  - How many pieces to be read for each run?

    Size of run/size of chunk = 10,000,000/125,000= 80 pieces
- Sorting completed in one round:
  - For reading: a total transfer time of 800 MB **(T_X)**
  - Same for writing: a total transfer time of 800 MB **(T_X)**
  - In addition, time for a **high number** of seeks


Unsorted file
(800 MB)
Ext. Sort (800 MB)
Run1
80 pieces per run
Final Sorted
File
Memory (buffer)
of 10 MB
Memory (buffer)
of 10 MB
Run2
Run80
80-way merge
Read:
80 (s+r) + Tx
Write:
80 (s+r) + Tx
Read:
6400 (s+r) + Tx
Write:
6400 (s+r) + Tx

Step 2 (continues)

- Total # of seeks = Total # of pieces (counting all runs):

  There are 80 runs and 80 pieces in each run:

  ✓ 80^2 pieces = 6400 seeks for reading

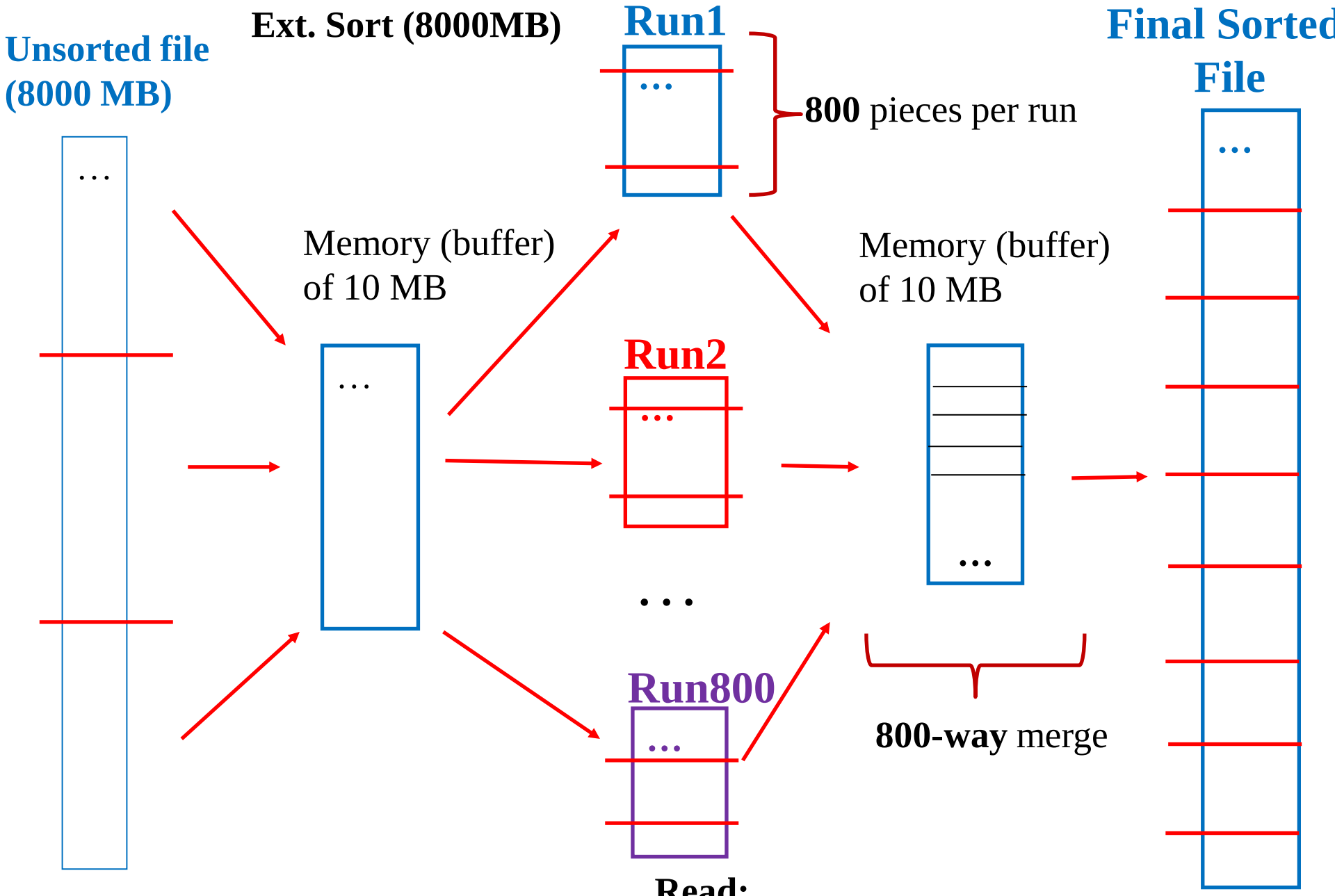
- Writing the merged records to the big sorted file:
- The number of separate writes closely approximates reads:

  ✓ 80^2 pieces = 6400 seeks for writing

- So we estimate two seeks (one for reading and one for writing) for each piece: 80* 80 pieces therefore **a total of 2*6400 = 12800 seeks are necessary.**

Sorting a File that is 10 times larger

- How is the time for merge phase affected if the file is 80 million records?
  - More runs: 800 runs
  - 800-way merge in 10MB memory
    - i.e. divide the memory into 800 pieces (lists).
  - Each piece (list) holds 1/800th of a run
  - So, 800 runs * 800 seeks/run = 640,000 seeks
  - Total = 2 * 640,000 **= 1,280,000** seeks

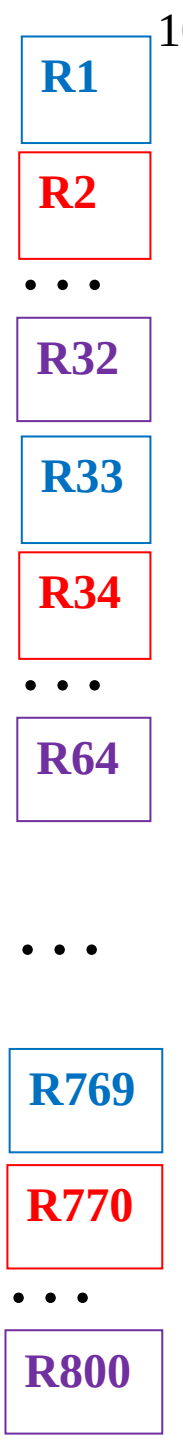

Unsorted file
(8000 MB)
Ext. Sort (8000MB)
Run1
800 pieces per run
Final Sorted
File
Memory (buffer)
of 10 MB
Memory (buffer)
of 10 MB
Run2
Run800
800-way merge
Read:
800 (s+r) + Tx
Write:
800 (s+r) + Tx
Read:
640,000 (s+r) + Tx
Write:
640,000 (s+r) + Tx

The cost of increasing the file size

- In general, for a K-way merge of K runs, the buffer size reserved for each run is
  - (1/K) * size of memory space = (1/K) * size of each run
- So K seeks are required to read all of the records in each run and K seeks to write records.
- Since there are K runs, merge requires a total of **$2K^2$** seeks.
- Because K is directly proportional to N it also follows that the total cost is an **$O(N^2)$** operation.

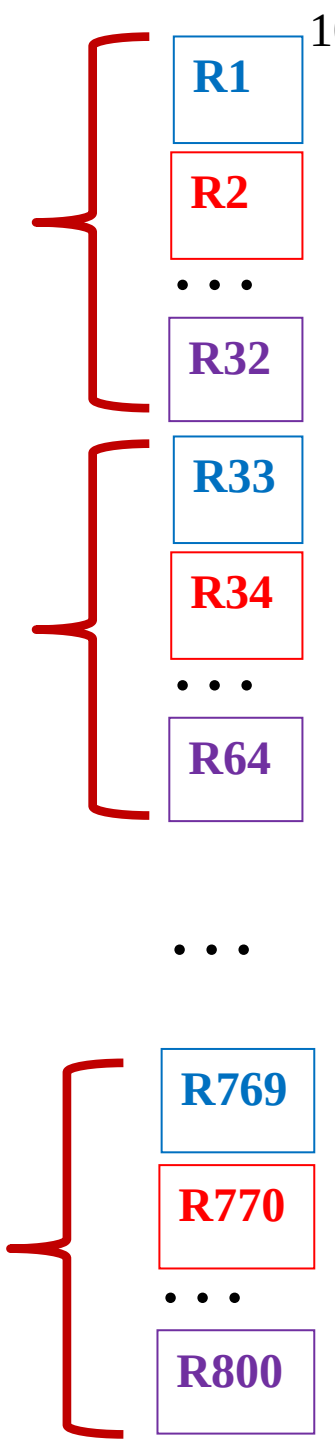
Multiple-pass multiway merges

- Instead of merging all runs at once, we break the original set of runs into small groups and merge the runs in these groups separately.
  - more memory space is available for each run; hence fewer seeks are required per run.
- When all of the smaller merges are completed, a second pass merges the new set of merged runs.



Memory (buffer): 10 MB

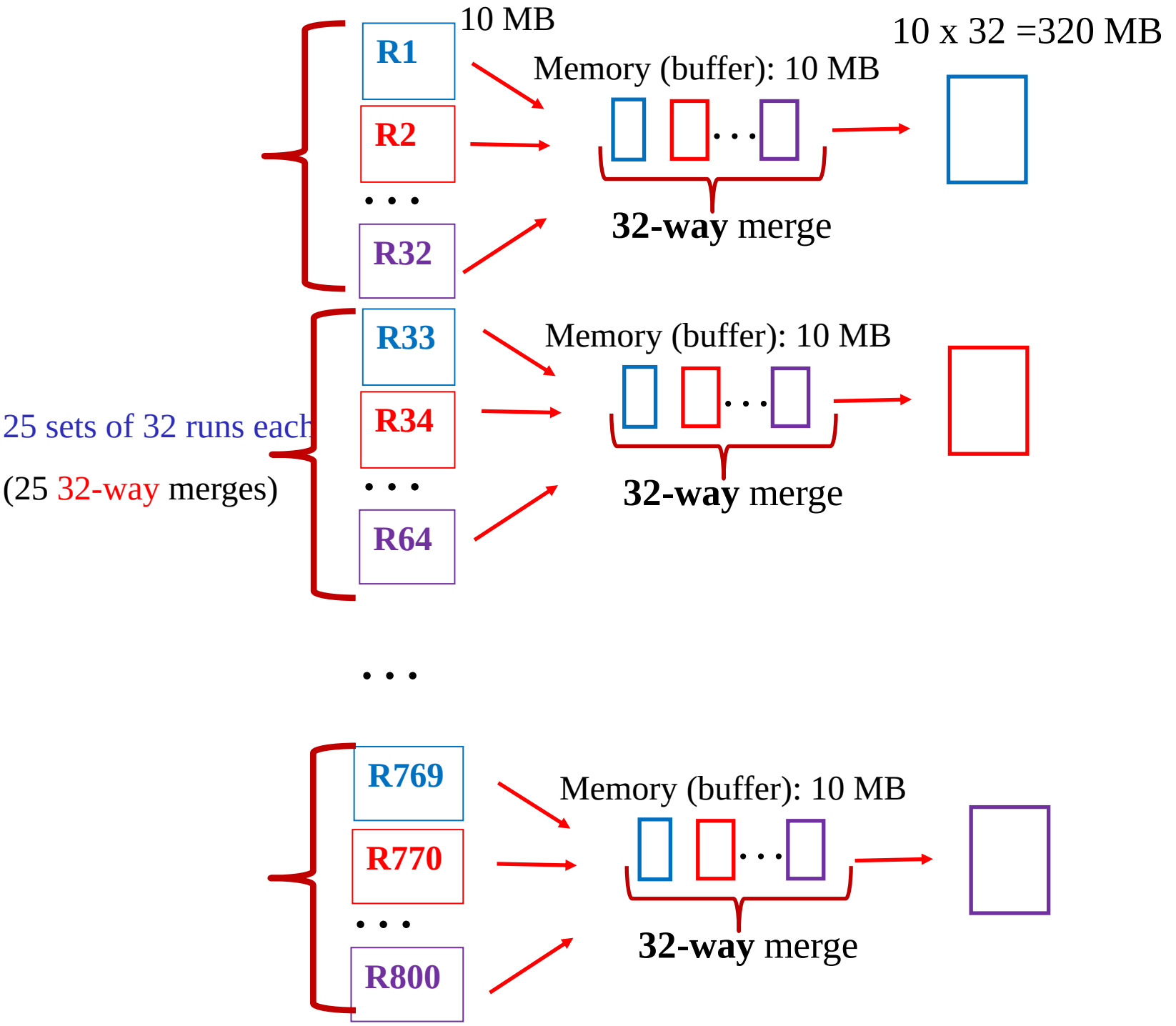
Two-pass merge of 800 runs

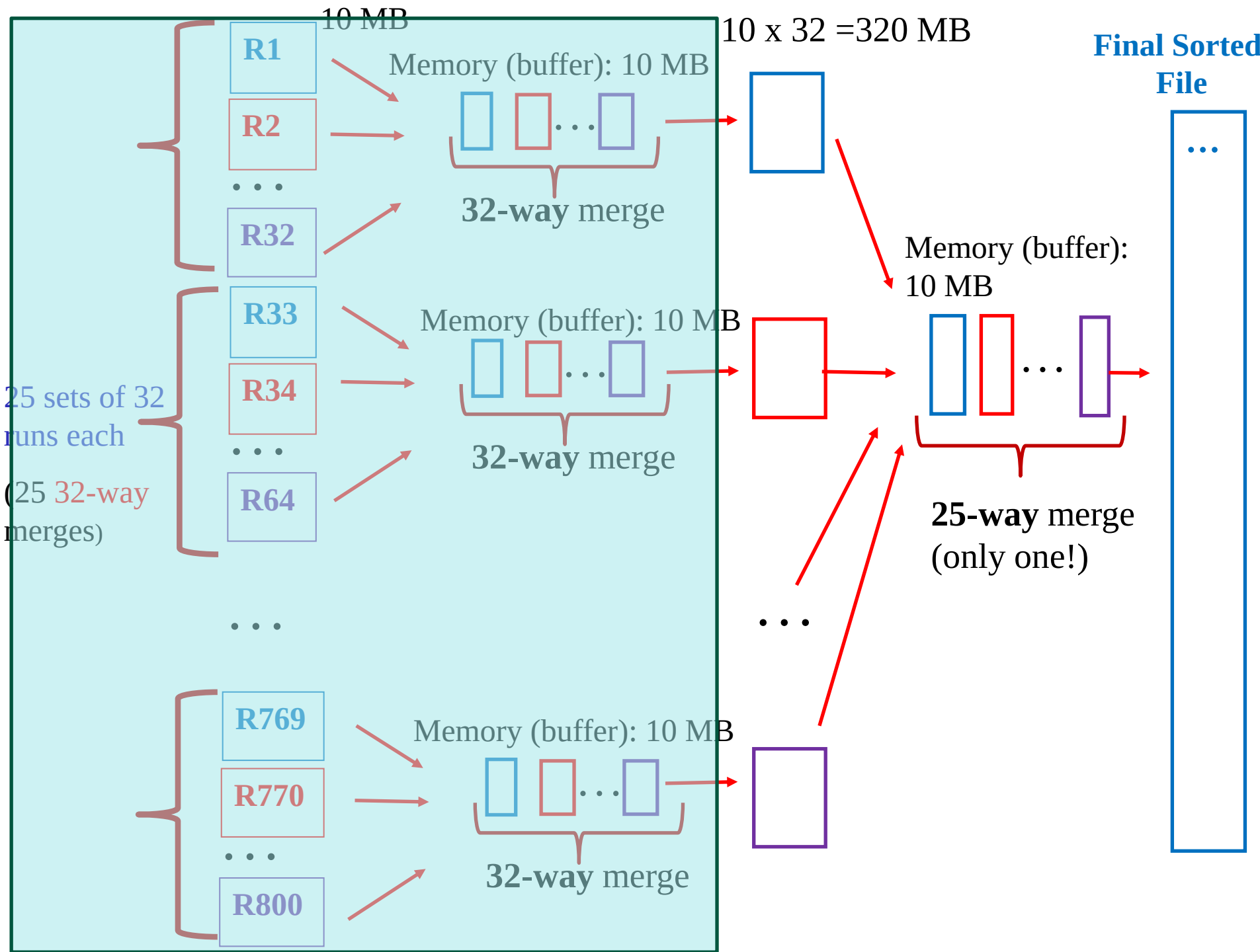


Memory (buffer): 10 MB

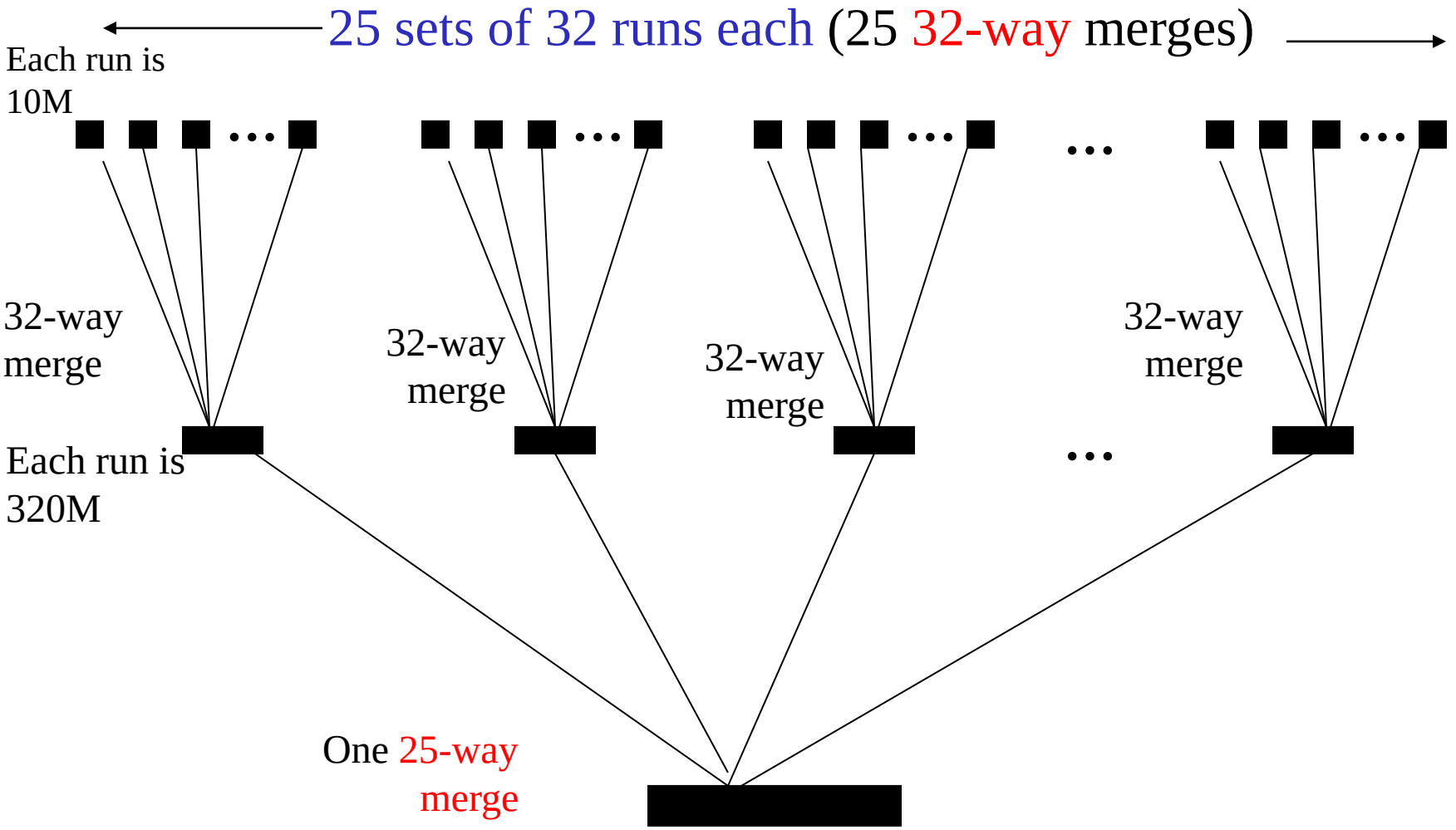
Two-pass merge of 800 runs:

32 way- merge in the first pass

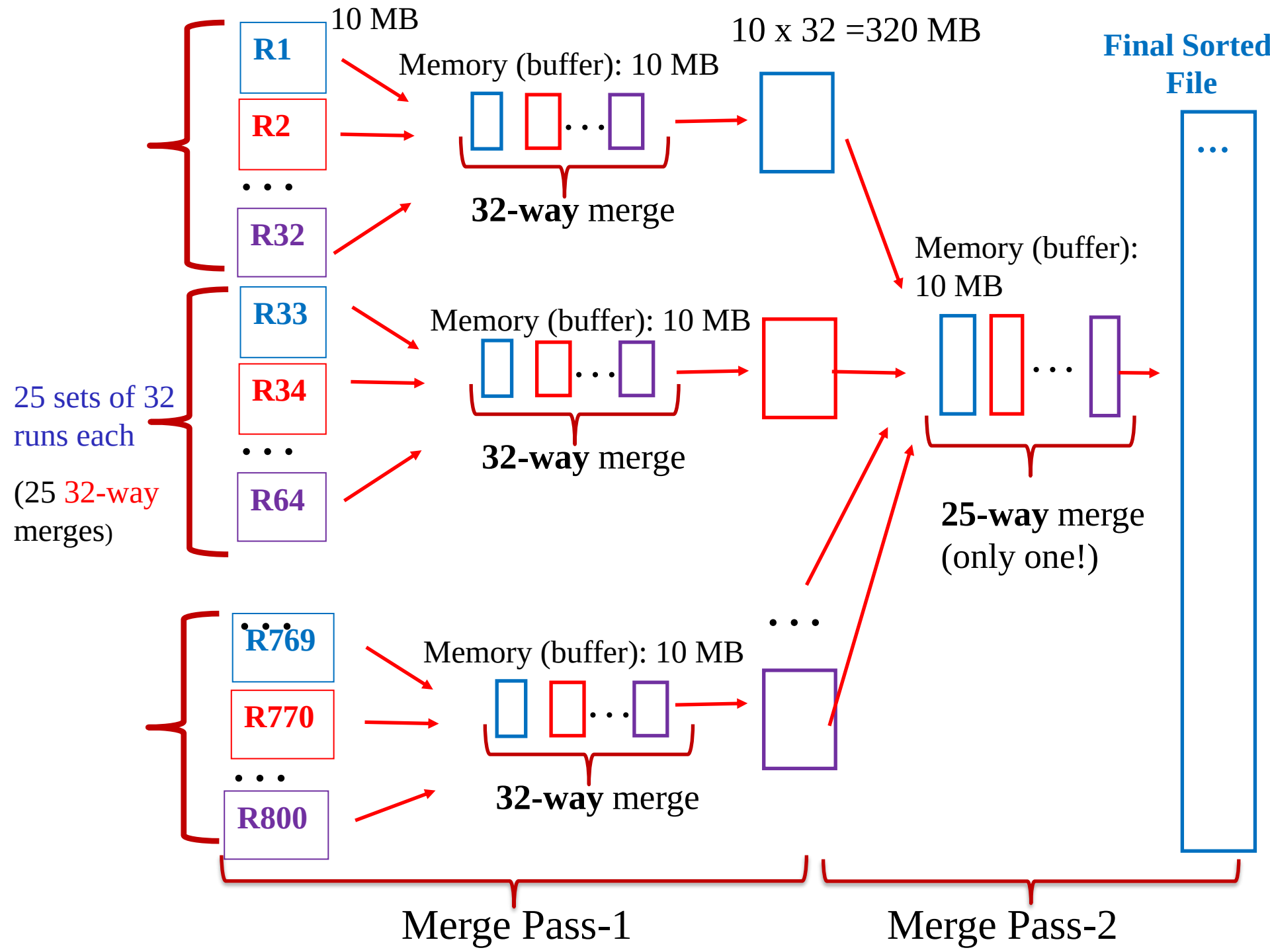

10 MB
10 x 32 =320 MB
R1
R2
R32
Memory (buffer): 10 MB
32-way merge
R33
R34
R64
Memory (buffer): 10 MB
32-way merge
25 sets of 32 runs each
(25 32-way merges)
R769
R770
R800
Memory (buffer): 10 MB
32-way merge

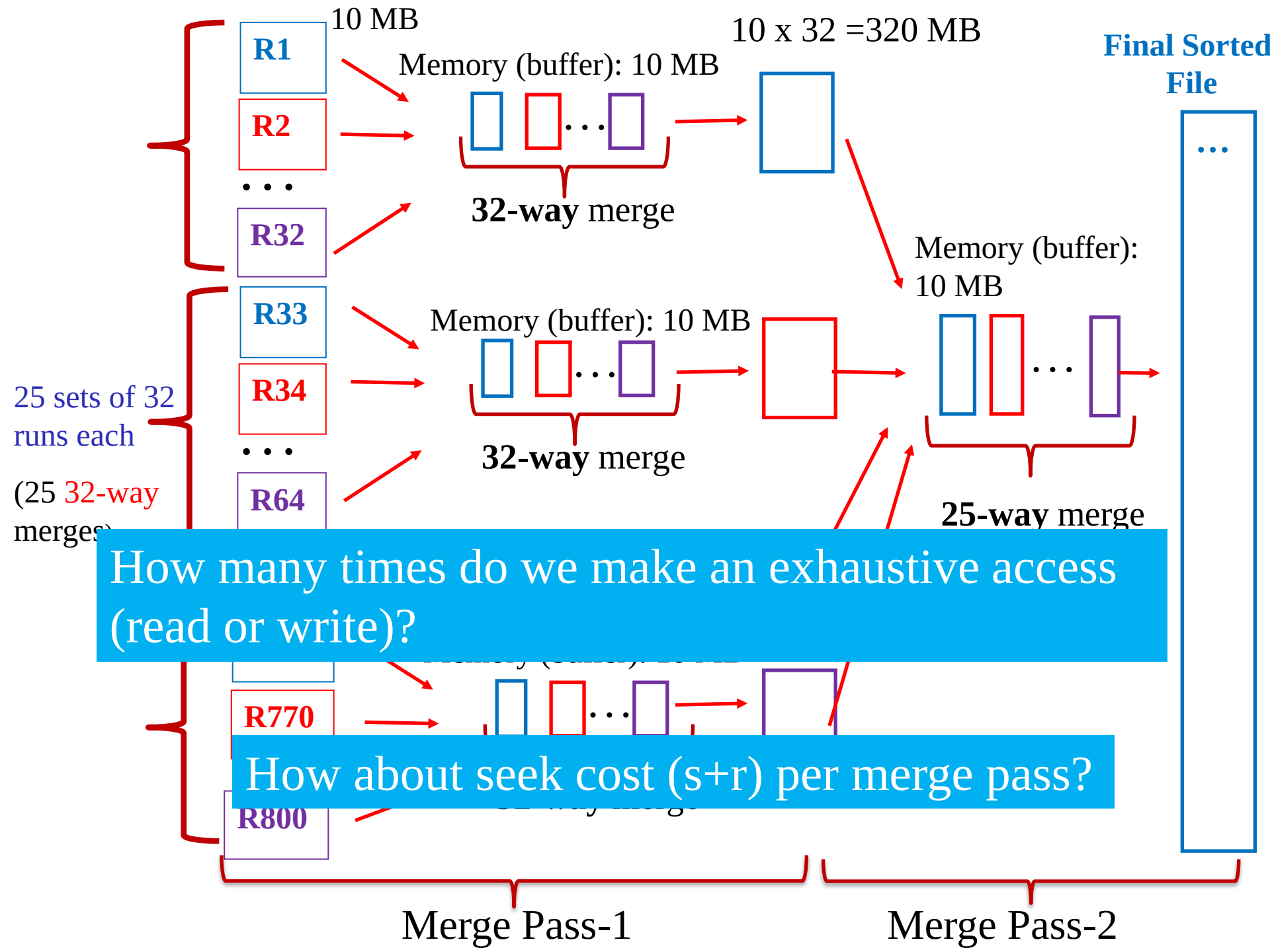

10 MB
R1
R2
R32
Memory (buffer): 10 MB
32-way merge
10 x 32 =320 MB
Final Sorted File
Memory (buffer): 10 MB
25 sets of 32 runs each
(25 32-way merges)
R33
R34
R64
Memory (buffer): 10 MB
32-way merge
25-way merge (only one!)
R769
R770
R800
Memory (buffer): 10 MB
32-way merge

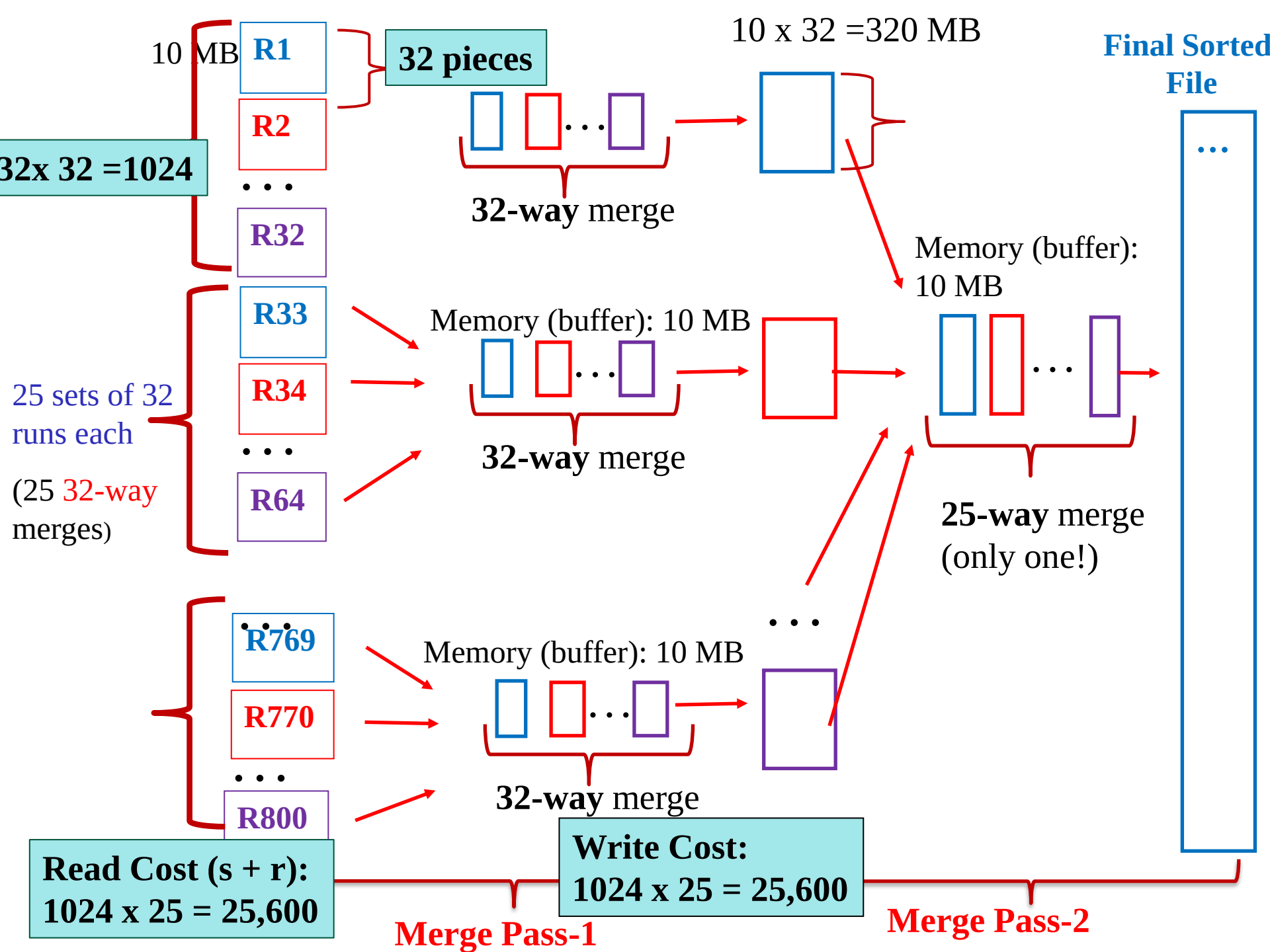
Two-pass merge of 800 runs

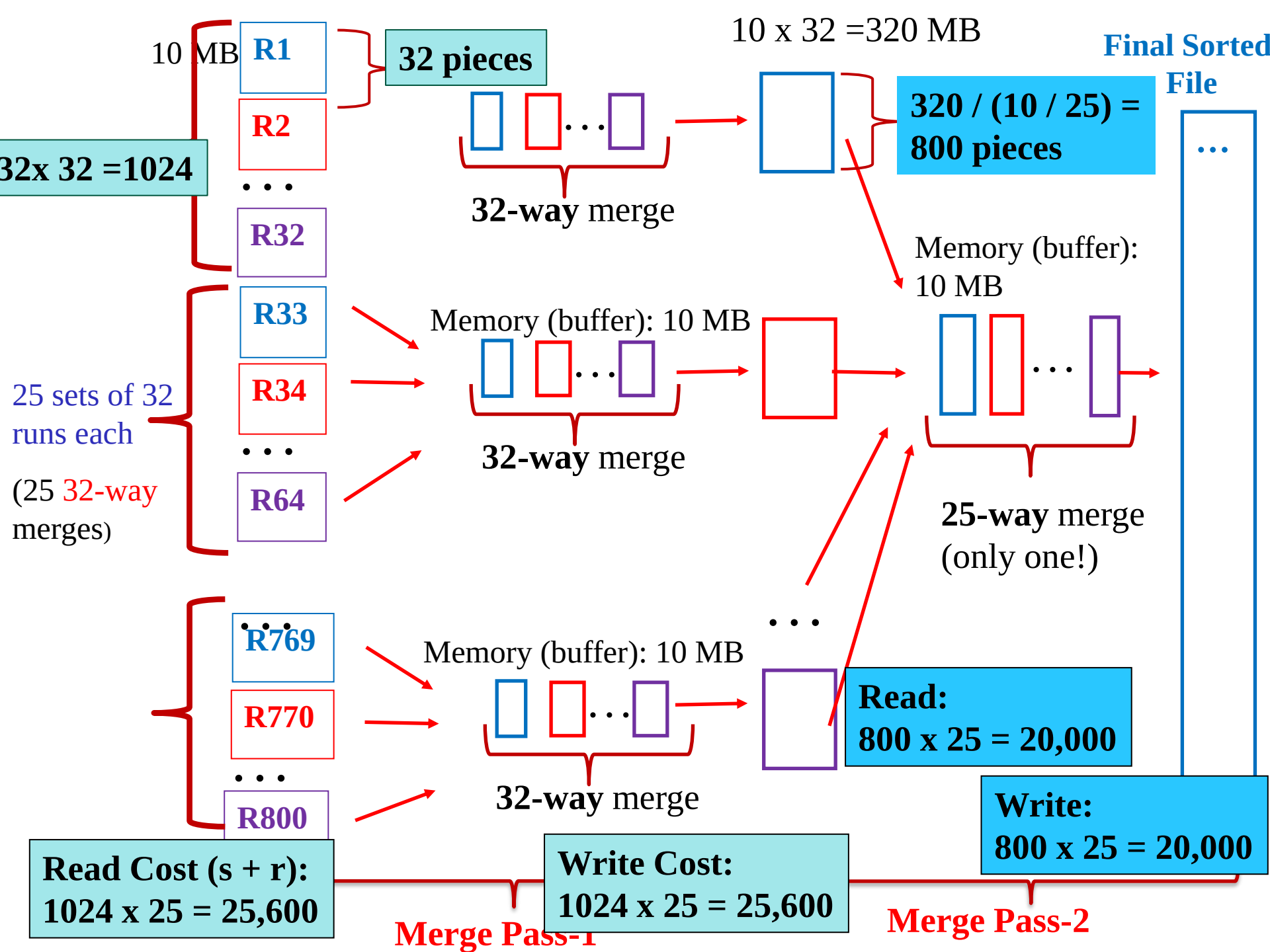


Two-pass merge of 800 runs


10 MB
R1
R2
R32
Memory (buffer): 10 MB
32-way merge
10 x 32 =320 MB
Final Sorted File
Memory (buffer): 10 MB
25 sets of 32 runs each
(25 32-way merges)
R33
R34
R64
Memory (buffer): 10 MB
32-way merge
25-way merge (only one!)
R769
R770
R800
Memory (buffer): 10 MB
32-way merge
Merge Pass-1
Merge Pass-2


10 MB
R1
R2
R32
Memory (buffer): 10 MB
32-way merge
10 x 32 =320 MB
Final Sorted File
Memory (buffer): 10 MB
R33
R34
R64
Memory (buffer): 10 MB
32-way merge
25 sets of 32 runs each
(25 32-way merges)
25-way merge
How many times do we make an exhaustive access (read or write)?
R770
R800
How about seek cost (s+r) per merge pass?
Merge Pass-1
Merge Pass-2


10 MB
R1
R2
R32
32x 32 =1024
32 pieces
32-way merge
10 x 32 =320 MB
Final Sorted File
Memory (buffer): 10 MB
R33
R34
R64
25 sets of 32 runs each
(25 32-way merges)
Memory (buffer): 10 MB
32-way merge
25-way merge (only one!)
R769
R770
R800
Memory (buffer): 10 MB
32-way merge
Read Cost (s + r): 1024 x 25 = 25,600
Write Cost: 1024 x 25 = 25,600
Merge Pass-1
Merge Pass-2


10 MB
R1
R2
R32
32 pieces
32x 32 =1024
32-way merge
10 x 32 =320 MB
320 / (10 / 25) = 800 pieces
Final Sorted File
Memory (buffer): 10 MB
R33
R34
R64
25 sets of 32 runs each
(25 32-way merges)
Memory (buffer): 10 MB
32-way merge
25-way merge (only one!)
R769
R770
R800
Memory (buffer): 10 MB
32-way merge
Read: 800 x 25 = 20,000
Write: 800 x 25 = 20,000
Read Cost (s + r): 1024 x 25 = 25,600
Write Cost: 1024 x 25 = 25,600
Merge Pass-1
Merge Pass-2

Cost of multi-pass merging

- 25 sets of 32 runs, followed by 25-way merge:
  - Disadvantage: we read every record twice.
  - Advantage: we can use larger lists and avoid a large number of disk seeks.
- Calculations:

  First Merge Pass:
  - List size = 1/32 run => 32*32 = 1024 seeks
  - For 25 32-way merges=> 25 * 1024 = 25,600 seeks

Second Merge Pass:

- For each 25 final runs, 1/25 memory space is allocated.
- So each input list is 0.4M and it can hold 4000 records (or 1/800 run)
- Hence, 800 seeks per run, so we end up making 25 * 800 = 20,000 seeks.

Total number of seeks for reading in two passes:

25600 + 20000 = 45,600

- What about the total time for merge?
  - We now have to transmit all of the records 4 times instead of two.
  - We also write the records twice, requiring an extra 45,600 seeks.
  - **Total number of seeks : 91200 seeks**
- Still the trade is profitable.

External sorting of 800 run (8000 MB) file

With one-pass merge:

Create runs (Pass-0)	2 x 800 (s+r)	+ **2 Tx**
Merge Pass-1	**2 x 640,000 (s+r)**	+ **2 Tx**

With two-pass merge:

Create runs (Pass-0)	2 x 800 (s+r)	+ **2 Tx**
Merge Pass-1	**2 x 25,600 (s+r)**	+ **2 Tx**
Merge Pass-2	**2 x 20,000 (s+r)**	+ **2 Tx**

Total Cost

Total # of block transfers= $2b*btt + (\lceil \log_k m \rceil) * 2b*btt$

$= 2b*btt * (\lceil \log_k m \rceil + 1)$

(when **k-way multi-pass merge** is used for **m runs**.)

Total cost including # of seeks:

$= 2b*btt + (\lceil \log_k m \rceil) * [2k*m*(s+r) + 2b*btt]$

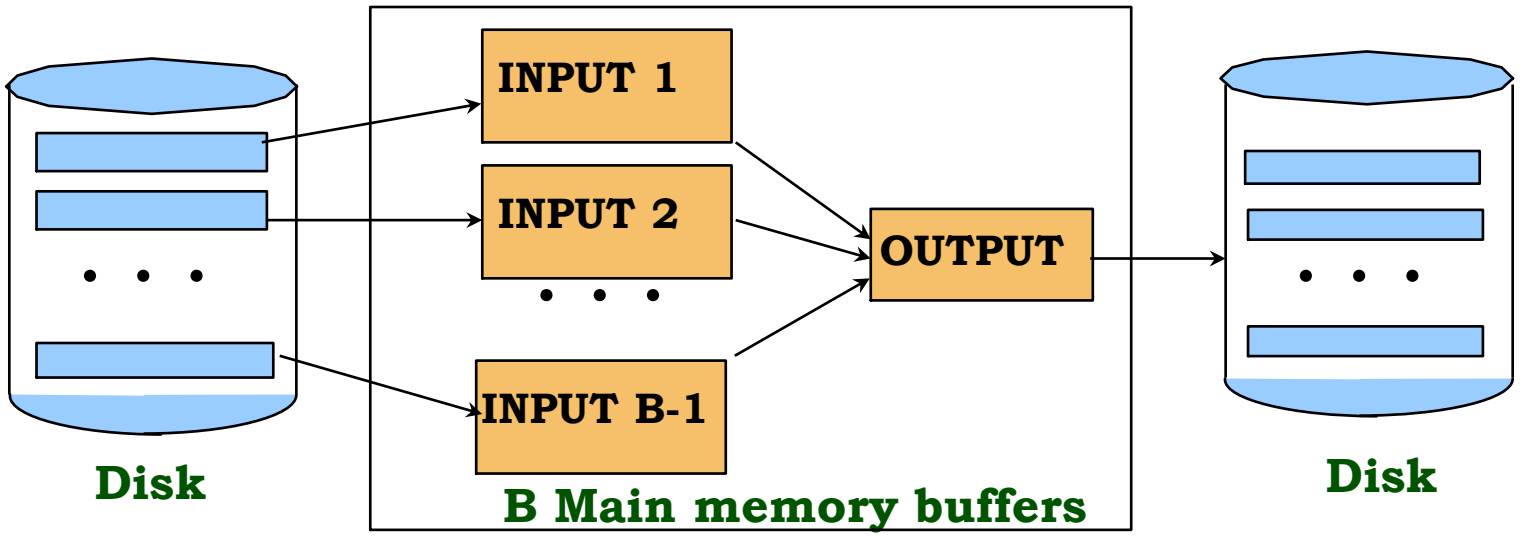
creating sorted runs

k-way multi-pass merge

(handled in $\lceil \log_k m \rceil$ passes (steps))

DBMS view (from Raghu's book)

- Sort a file with *N* blocks (pages)
- Available memory *B* blocks (buffer pages):
  - Pass 0: use *B* buffer pages. Produce $\lceil N / B \rceil$ sorted runs of *B* pages each.
  - Pass 1, 2, …, etc.: merge *B-1* runs.



Cost of External 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:
  - Pass 0: $\lceil 108 / 5 \rceil$ = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: $\lceil 22 / 4 \rceil$ = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: $\lceil 6 / 4 \rceil$ = 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

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	17	9	6	5	3	3
1,000,000	20	10	7	5	3	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

Problem

Assume you are given a file of 800,000 records, 400 bytes each, a block size of 4000 bytes with btt of 1 msec, and the available memory size is 8MB. You are asked to sort this file using 4-way merge with one disk drive. Assume s = 5 ms, r = 4 ms, btt= 1ms.

1. How many runs (sorted segments) are created ?
2. What is the size of each run?
3. Estimate the cost of creating these runs.

Solution

1. How many runs (sorted segments) are created?

   Number of runs = $file\ size / size\ of\ avail.memory$ = 800000∗400/ 8000000 = 40 sorted segments

2. What is the size of each run?

   Size of each run = size of available memory = 8MB

3. Estimate the cost of creating these runs

- Number of blocks = b = 800000 ∗400/4000=80000 $blocks$
- Cost of creating the runs = $2*b*btt$=2∗80000∗(1ms)=160000 ms

Problem (cont.)

4. What is the number of passes necessary to merge the sorted segments using 4-way merge?

5. Estimate the cost of one pass of merging.

6. Estimate the total cost of sorting the file, including the creation of sorted runs and the multi- step merging.

Solution (cont.)

4. What is the number of passes necessary to merge the sorted segments using 4-way merge?

Number of passes = $\lceil \log_4 40 \rceil = 3$

5. Estimate the cost of one pass of merging.

Cost of one pass of merging =

= 2 * ***k*** * (# of runs) * (s+r) + 2 * b * btt

where *k* is the number in k-way merging (= no of pieces in each run)

= 2 * 4 * 40 * (9 ms) + 2 * 80000 * (1ms)

= 2880 ms + 160000 ms

= 162880 ms

Solution (cont.)

6. Estimate the total cost of sorting the file, including the creation of sorted runs and the multi- step merging.

Total cost of sorting the file =

# of passes * cost for one pass of merging + cost of creating runs (Pass-0)

= 3 * (162880 ms) + 160000 ms = 648640 ms