Data Representation & External Sorting

DSCI 551

Wensheng Wu

Outline

- Representing data
  - How are tables stored on storage devices?
- External Sorting
  - How to sort 1TB data using 1GB of memory?

In-place sorting

Representing Data Elements

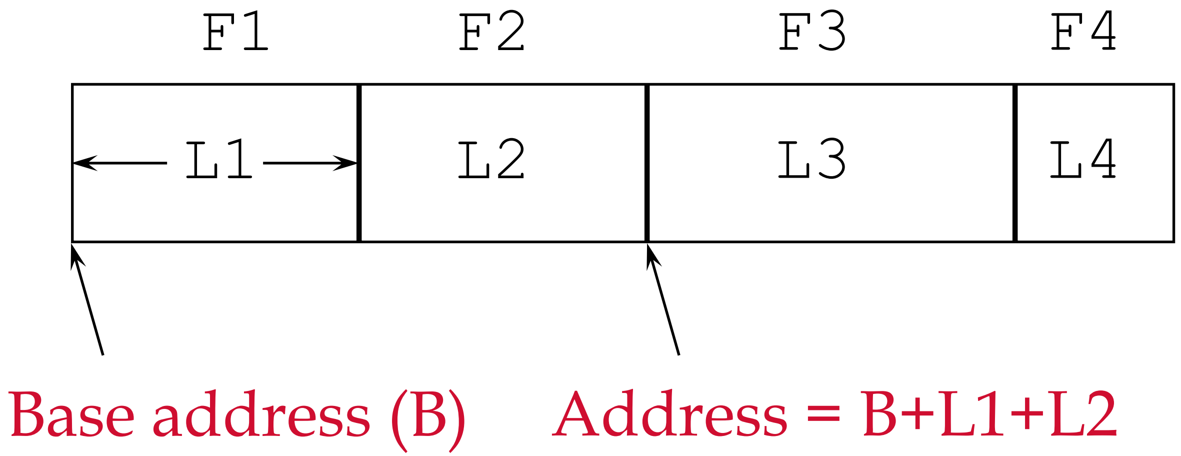
Base address (B)

- Relational database elements:

```
CREATE TABLE Product (
    pid INT PRIMARY KEY, // tinyint, smallint, mediumint
    name CHAR(20),
    description VARCHAR(100),
?   maker CHAR(10) REFERENCES Company(name))
```

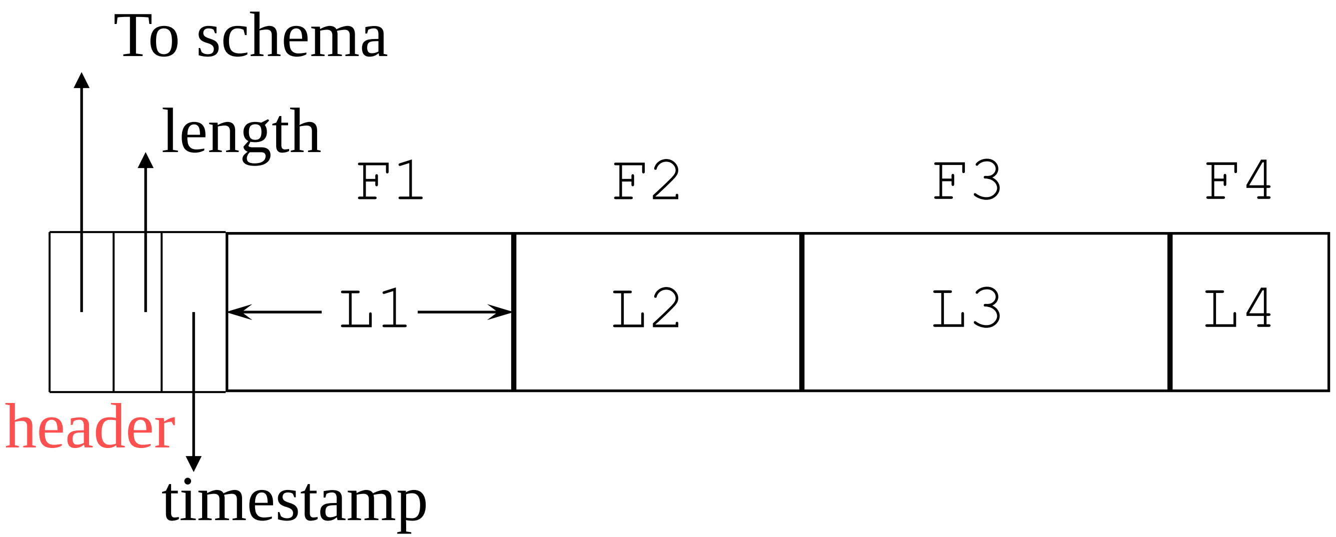
- A tuple/row is stored as a "record"

Record Formats: Fixed Length



- Information about field types is the same for all records in a file; stored in *system catalogs.*
- **Note the importance of schema information!**

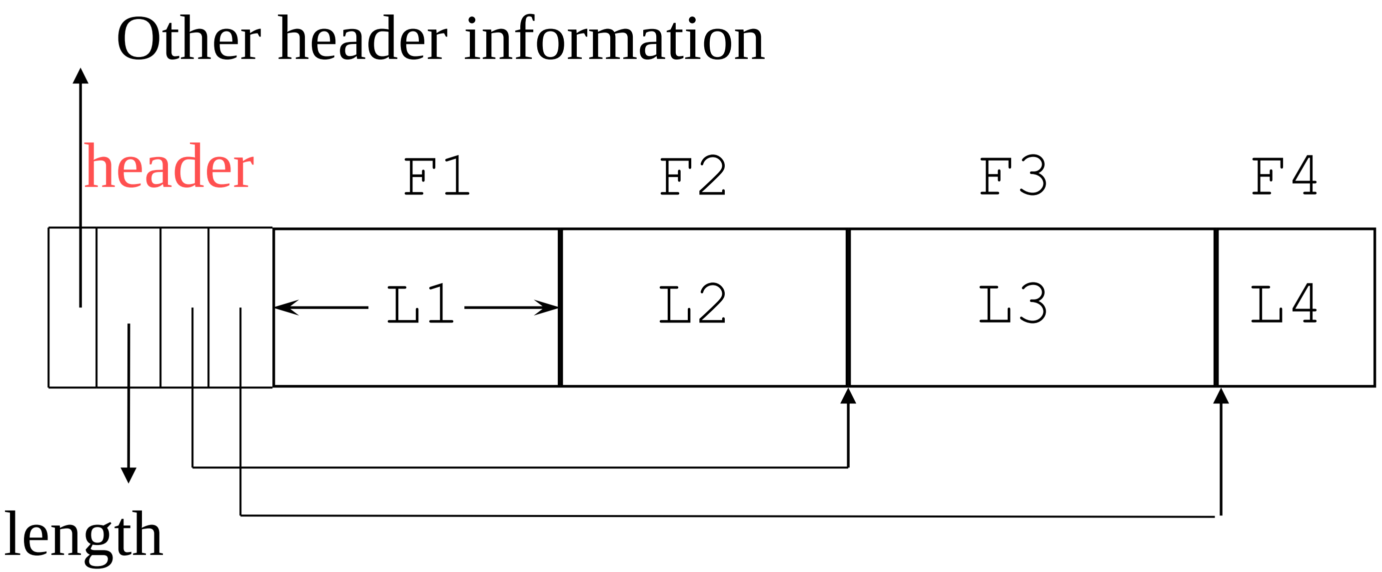
Record Header



Header:

- Pointer to schema: help finding fields
- Length: so we know where the record ends w/o consulting schema
- Timestamp: time when record last modified or read

Variable Length Records



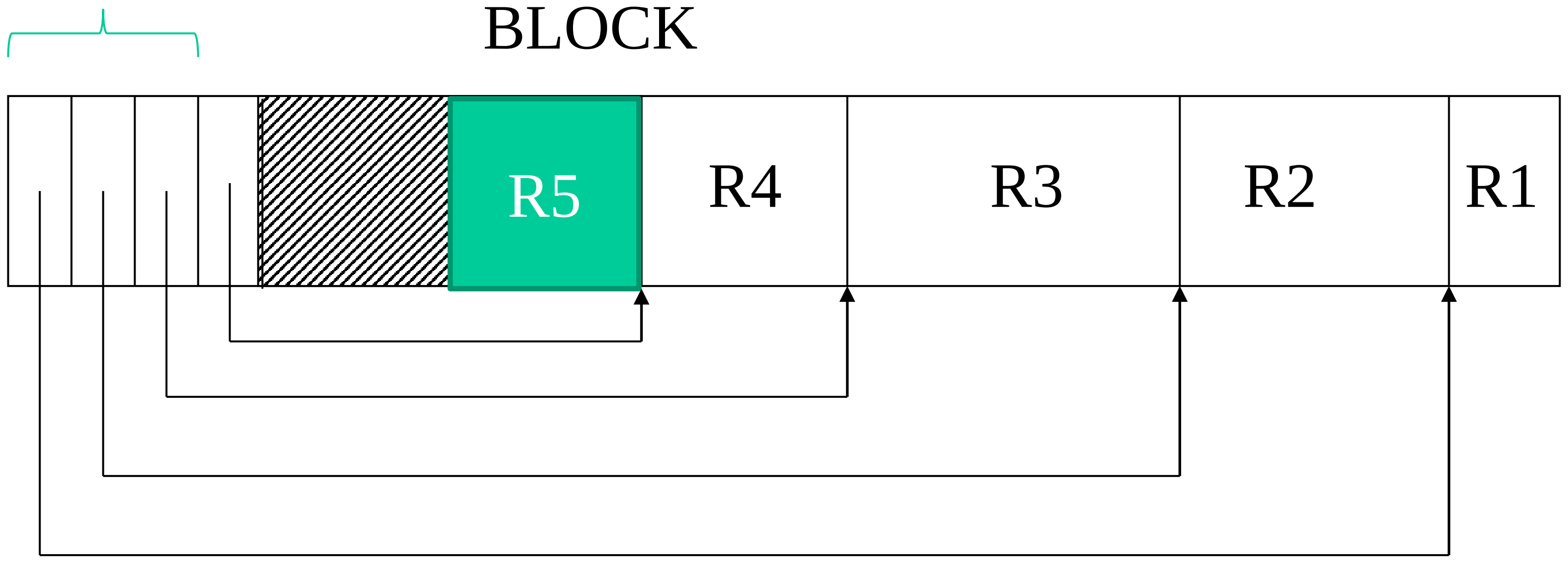
Place the fixed fields first: F1, F2
Then the variable length fields: F3, F4

Note: actually no need for pointer to F3, why?

Storing Records in Blocks

- Blocks have fixed size (typically 4KB)
  - But records may have variable-length

Offset table (slot directory)

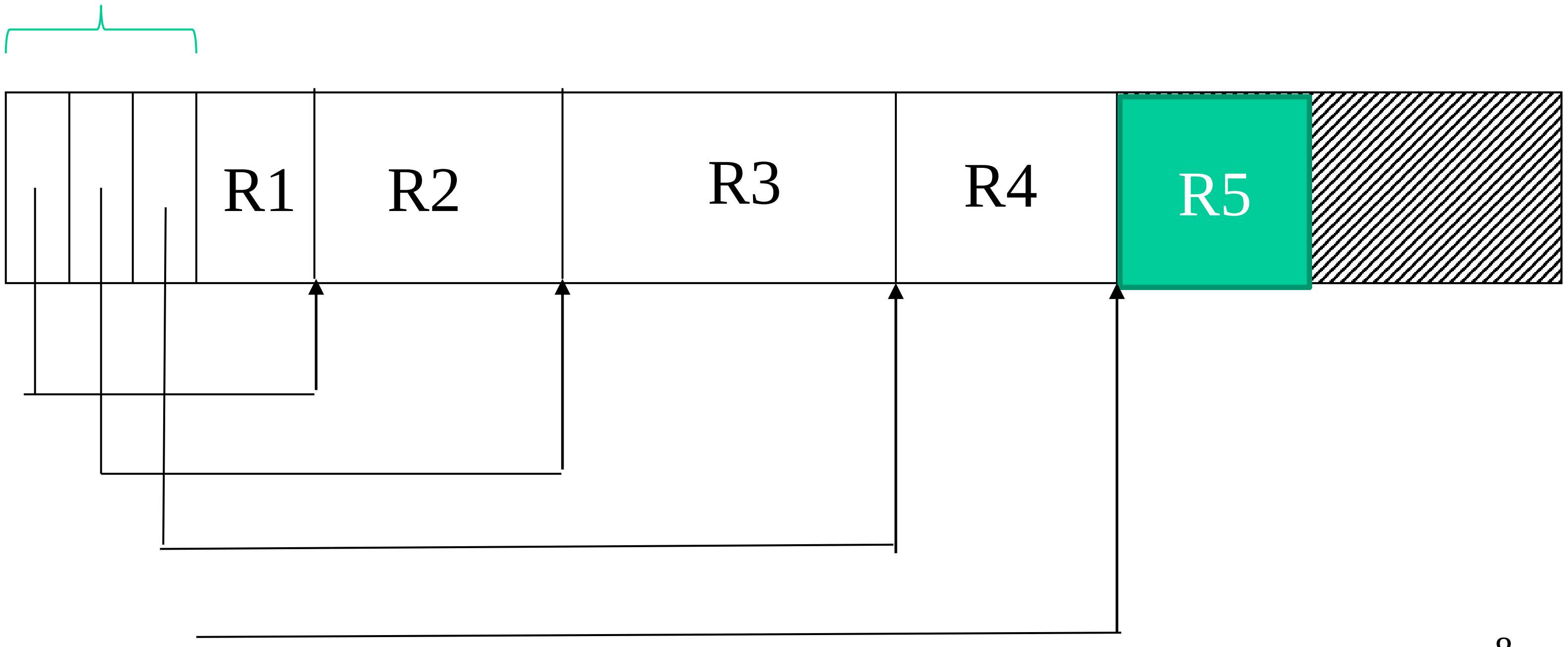


Why are records placed from the end?

Problem with this design?

- Records right after slot directory
- Free space after all records

Offset table (slot directory)



Outline

Mergesort(n)

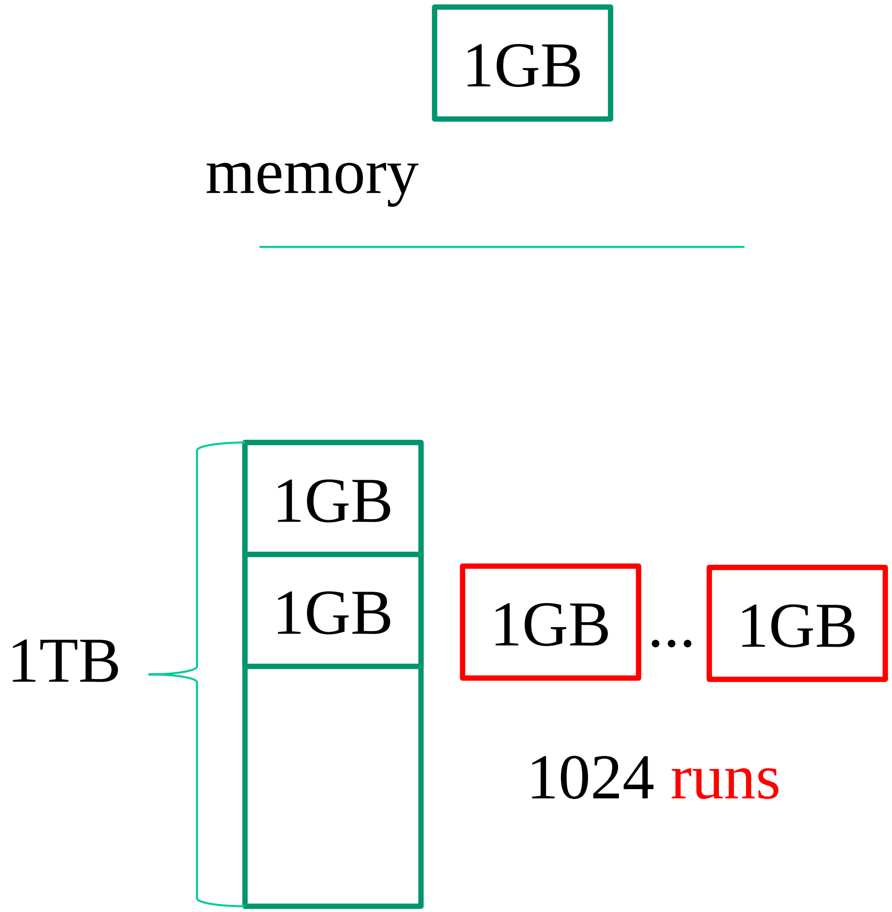
- Representing data
  - How are tables stored on storage devices?

- External Sorting
  - How to sort 1TB data using 1GB of memory?
  - 1GB => memory => 1GB run (sorted)
  - …
  - 1GB => memory => 1GB run

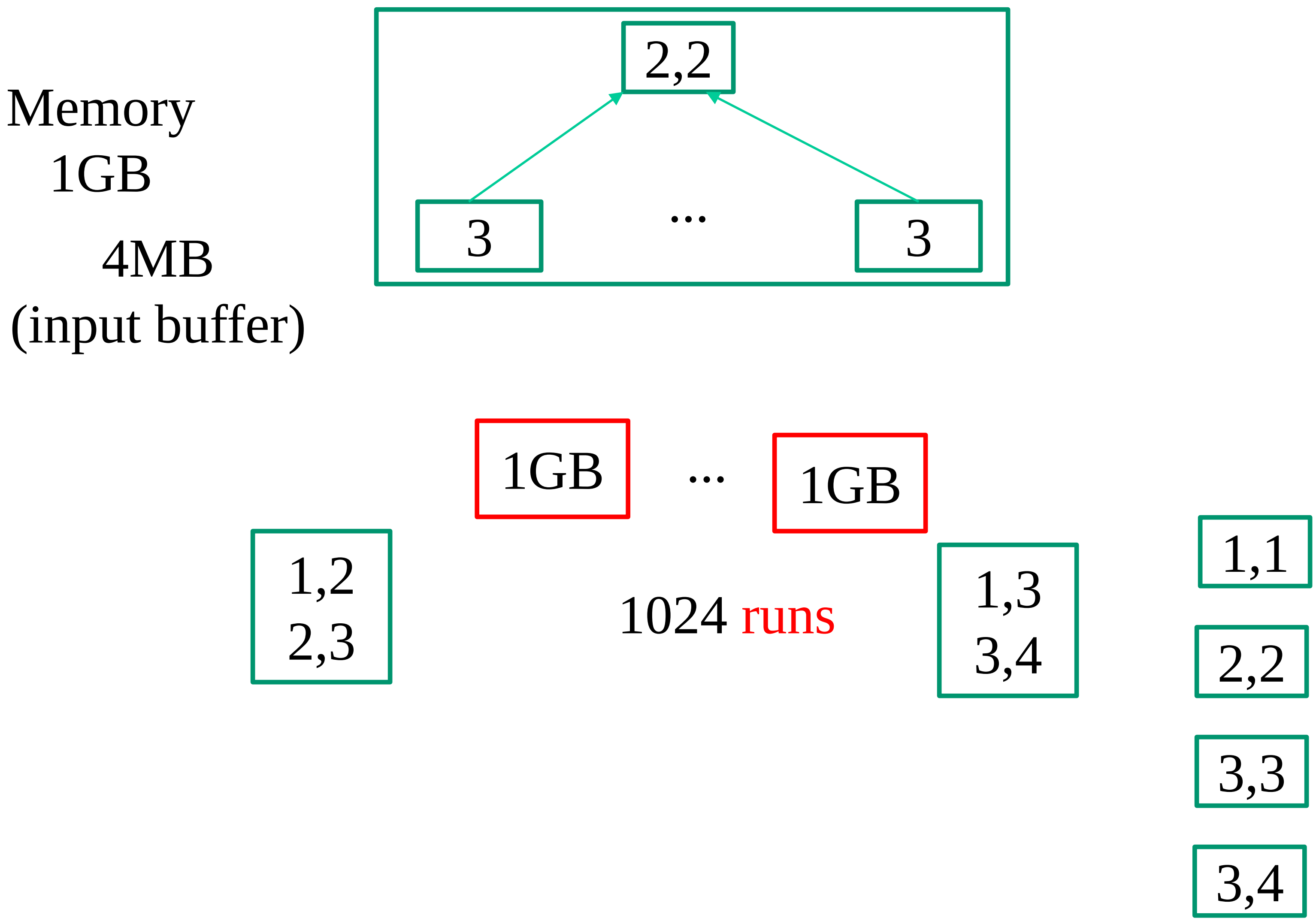
  Sorting (1024)

  - merging: load one block from each run to memory

Notes



Notes



Memory
1GB
4MB
(input buffer)
2,2
3
...
3
1GB
...
1GB
1,2
2,3
1024 runs
1,3
3,4
1,1
2,2
3,3
3,4

The I/O Model of Computation

- In main memory algorithms:
  - we care about CPU time
- In databases
  - time is dominated by I/O cost

- Assumption: cost is given only by I/O
- Consequence: need to redesign certain algorithms, e.g., sorting

Notes

- A block on storage devices loaded into a page in main memory
  - We sometimes interchange page with block
- Buffer pages
  - Often refer to pages in main memory used to store input, output, and intermediate data for an algorithm
- Run: a sorted sublist of input data

Notes

- Make a pass through data:
  - Loading the entire data from disk once

```
select bar, count(*)
from Sells
group by bar
```

Sorting

- Illustrates the difference in algorithm design when your data is not in main memory:
  - Problem: sort 1TB of data with 1GB of RAM

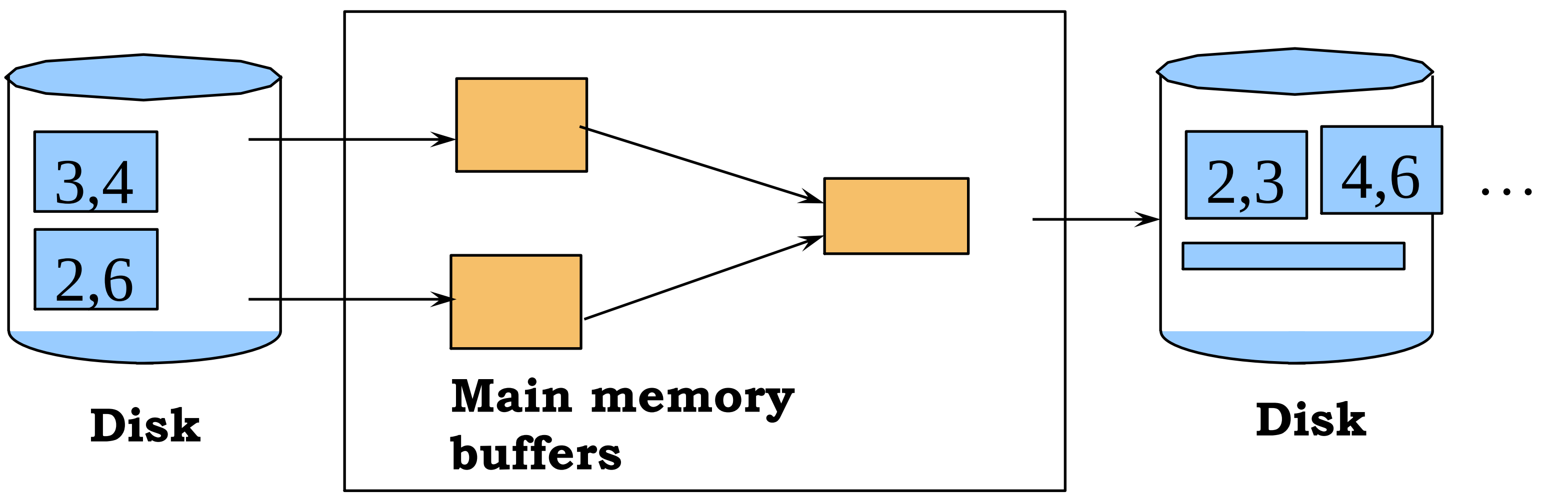
R: 20 20 20 22 22 25 25 25

S: 20 20 23 24 25 26

- Arises in many places in database systems:
  - Data requested in sorted order (ORDER BY)
  - Needed for grouping operations // group by age
  - First step in sort-merge join algorithm (R join_a S)
  - Duplicate removal
  - Bulk loading technique for creating B+-tree indexes

2-Way Merge-sort: Requires 3 Buffers

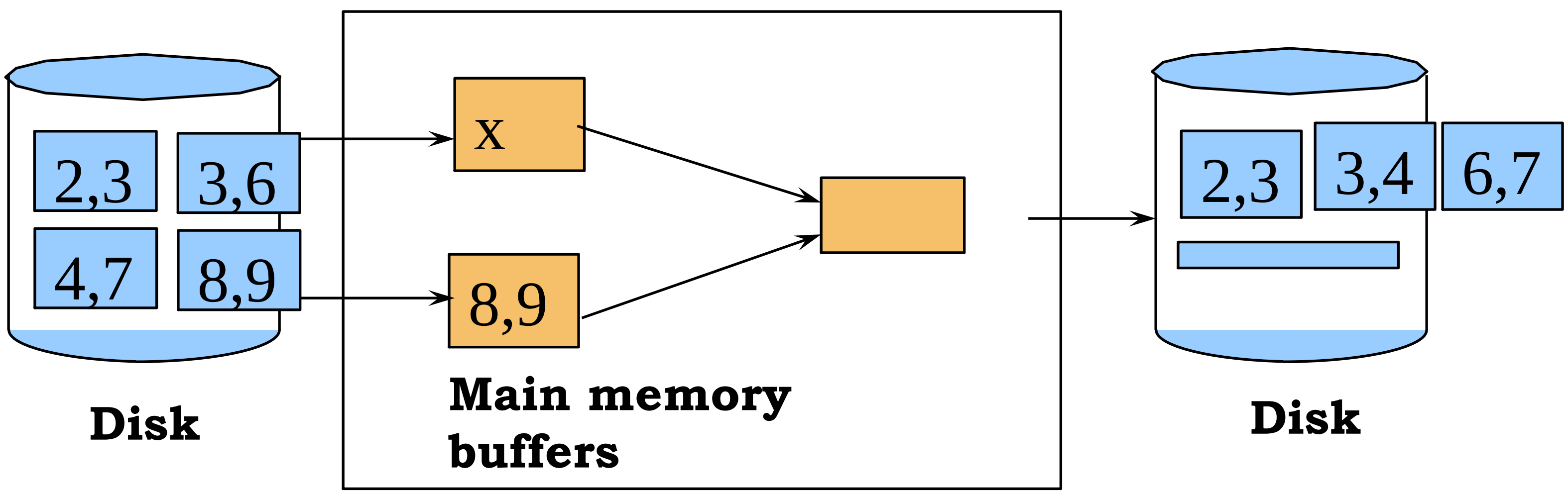
- Pass 0: Read a page, sort it, write it
  - only one buffer page is used
- Pass 1, 2, …, etc.: merging two runs at a time
  - three buffer pages used.



M=3

2-Way Merge-sort: Requires 3 Buffers

- Pass 0: Read a page, sort it, write it
  - only one buffer page is used
- Pass 1, 2, …, etc.: merging two runs at a time
  - three buffer pages used.



M=3

Two-Way External Merge Sort

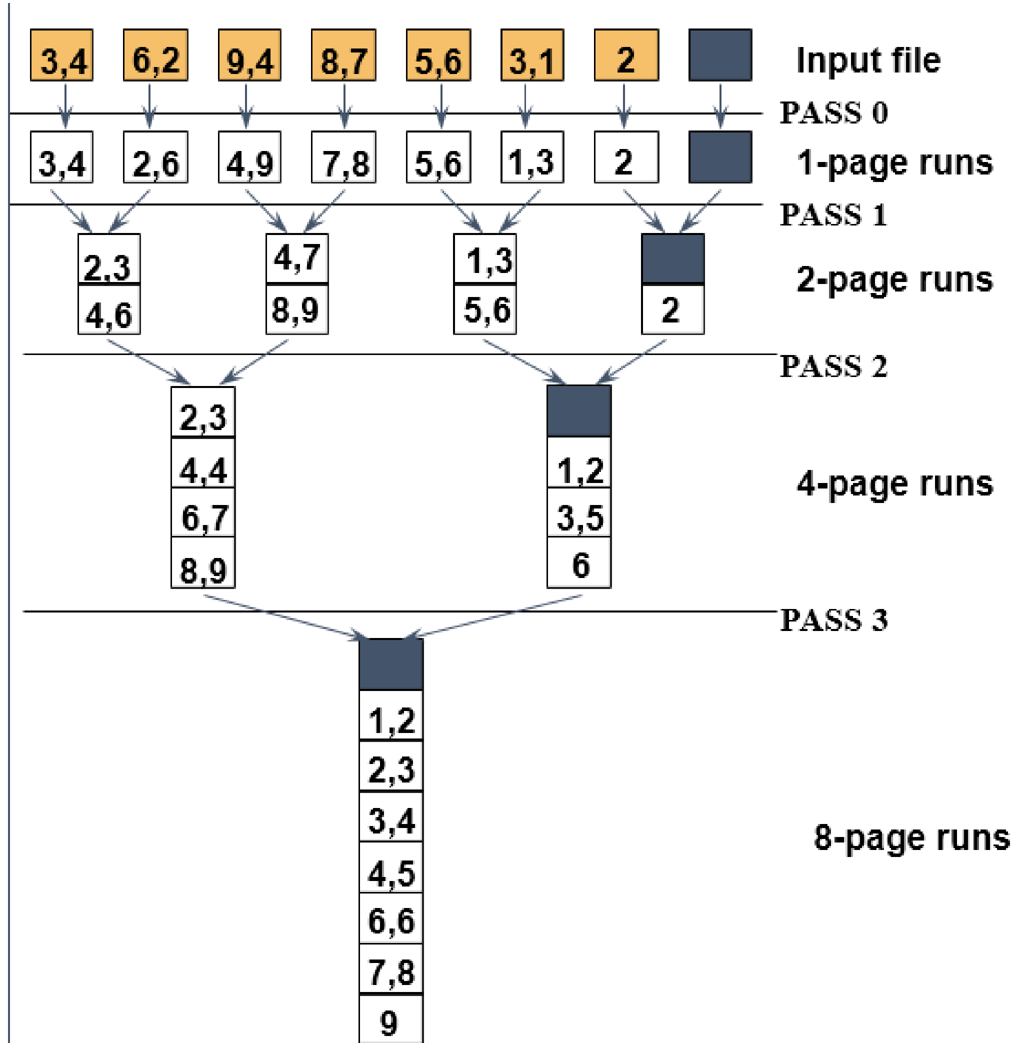
- Each pass we read + write each page in file.
- N pages in the file => the number of passes

$$= \lceil \log_2 N \rceil + 1$$

- So total cost is:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

- Sort 4MB with buffer page size = 4KB: needs11 passes



N=7, k = 3

$1 * 2 * 2 * \ldots * 2 = 1 * 2^k \;\; >= \;\; N \Rightarrow k >= \log_2(N)$

$k = \text{ceil}\,[\log_2 (N)] = \text{ceil}[2.8] = 3$

Notes

N=4MB/4KB
=1024
= 2^10

$\log_2(2^{10}) = 10$

M=5, B(R)=108

sorting:

  # runs: 108/5 = 21.6 = 22

  size of run:

    5 blocks for each of first 21 runs

    3 blocks in last run

merging 1: (4-way)

1. take 4 runs, merge into a single run
   size of run: 4*5 = 20 blocks
2. take next 4 runs => 20 blocks
3. next 4 => 20 blocks
4. next 4 => 20 blocks
5. next 4 => 20 blocks
6. take last two runs => 5+3 = 8 blocks

M=5, B(R)=108

sorting:

runs: 108/5 = 21.6 = 22

size of run:

5 blocks for each of first 21 runs

3 blocks in last run

merging 1: (4-way)

# of runs: 6 = ceiling(22/4) = ceiling(5.5)

size of run:

20 blocks: first 5

8 blocks: last run

merging 2: (4-way)

# of runs: 2 = ceiling(6/4) = 2

1. take first 4 runs => 20 * 4 = 80 blocks
2. take last 2 runs => 20 + 8 = 28 blocks

merging 3:

take 2 runs => a single run

Can We Do Better ?

- We have more main memory
- Should use it to improve performance

- M: # of blocks (i.e., pages) in main memory
- B(R): # of blocks of relation R

  M=5, B(R)=108

  sorting: load 5 pages, sort them, write back as run

    108/5 = ceil(21.6) runs

    21 runs, 5 pages/run => 21*5 = 105 pages

    1 run, 3 pages/run

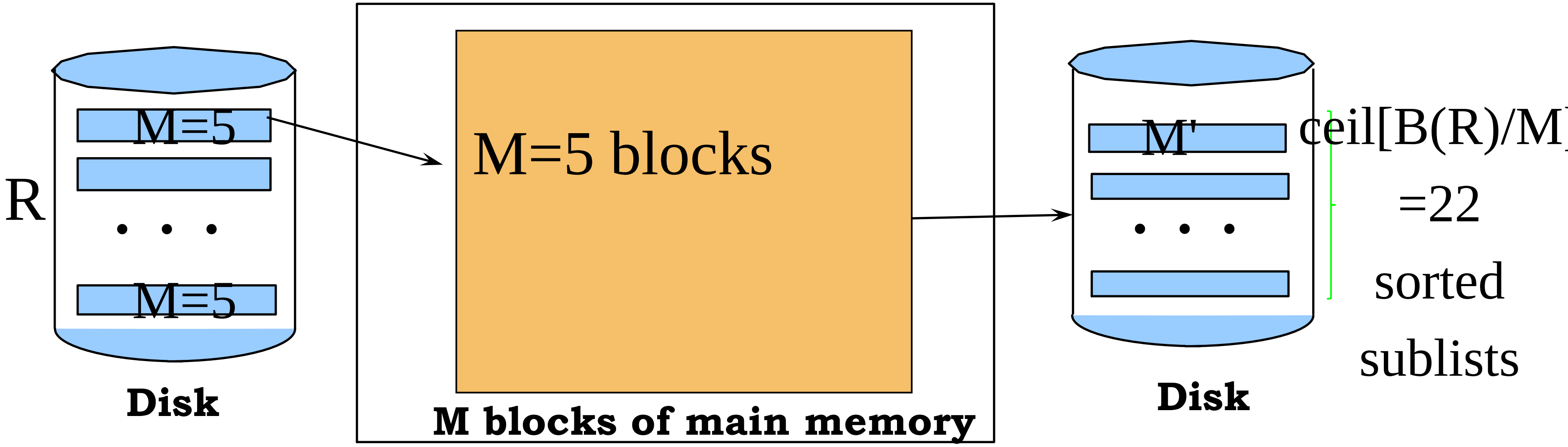
  Merging: (M-1)-way merging => 4-way merging

    take 4 runs, 5 pages/run => 20-page run

    how many runs?

External Merge-Sort

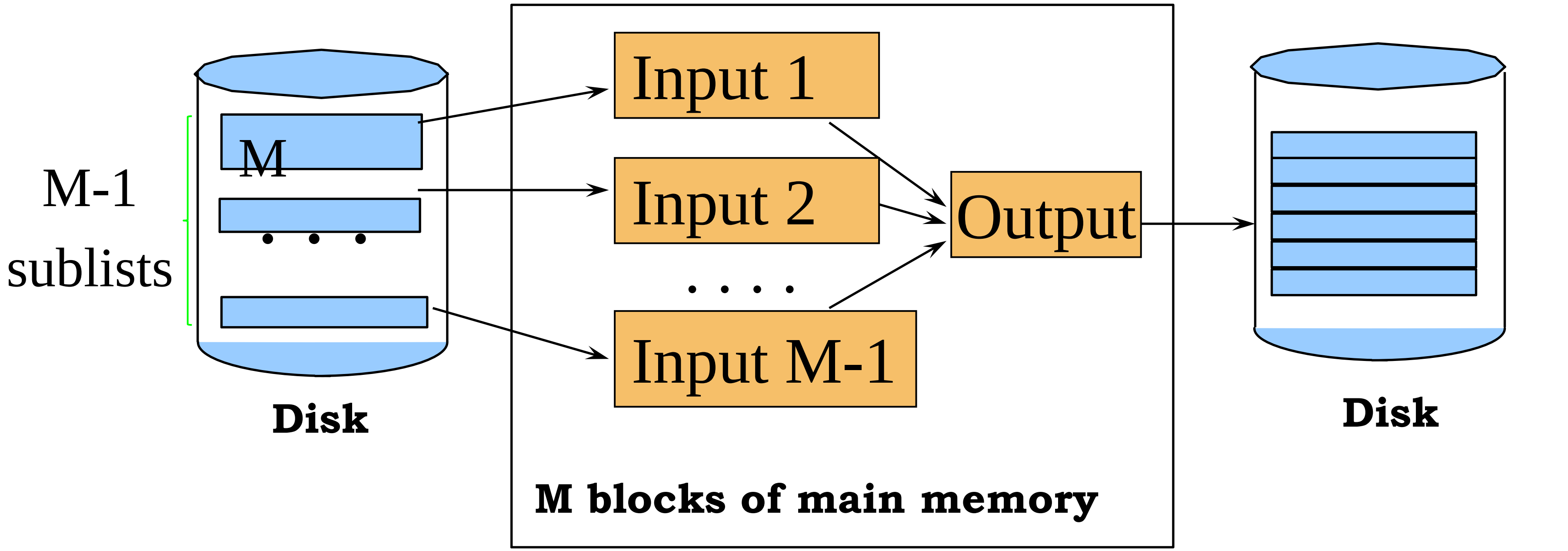
- Pass 0: load M blocks in memory, sort
  - Result: ceil(B(R)/M) sorted sublists of size M
  - Each sorted sublist is a run



B(R)=110

Pass One (merging)

- Merge M – 1 runs into a new run
- Result: each run has now M (M – 1) blocks



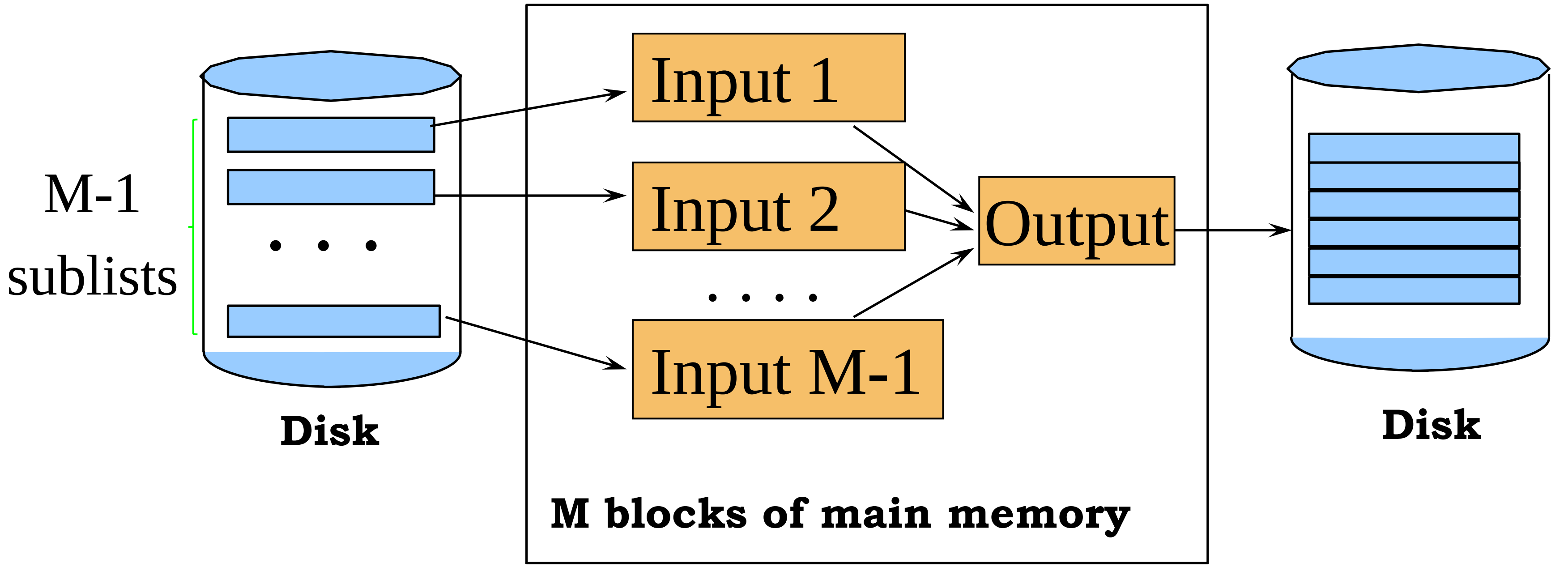
Cost of Two-Pass, Multiway Merge Sort

- Pass 0: sort B/M sublists of size M, write
  - Cost: 2B(R)
- Pass 1: merge B/M sublists, write
  - Cost: 2B(R)

- Total cost: 4B(R)
- Assumption: B(R) <= M^2
  - B/M <= M – 1 or
  - B <= M(M-1) ~ M^2

Generalized to k Passes

- Merge every M – 1 runs into a new run
- Result: each run has now $M (M – 1)^k$ blocks

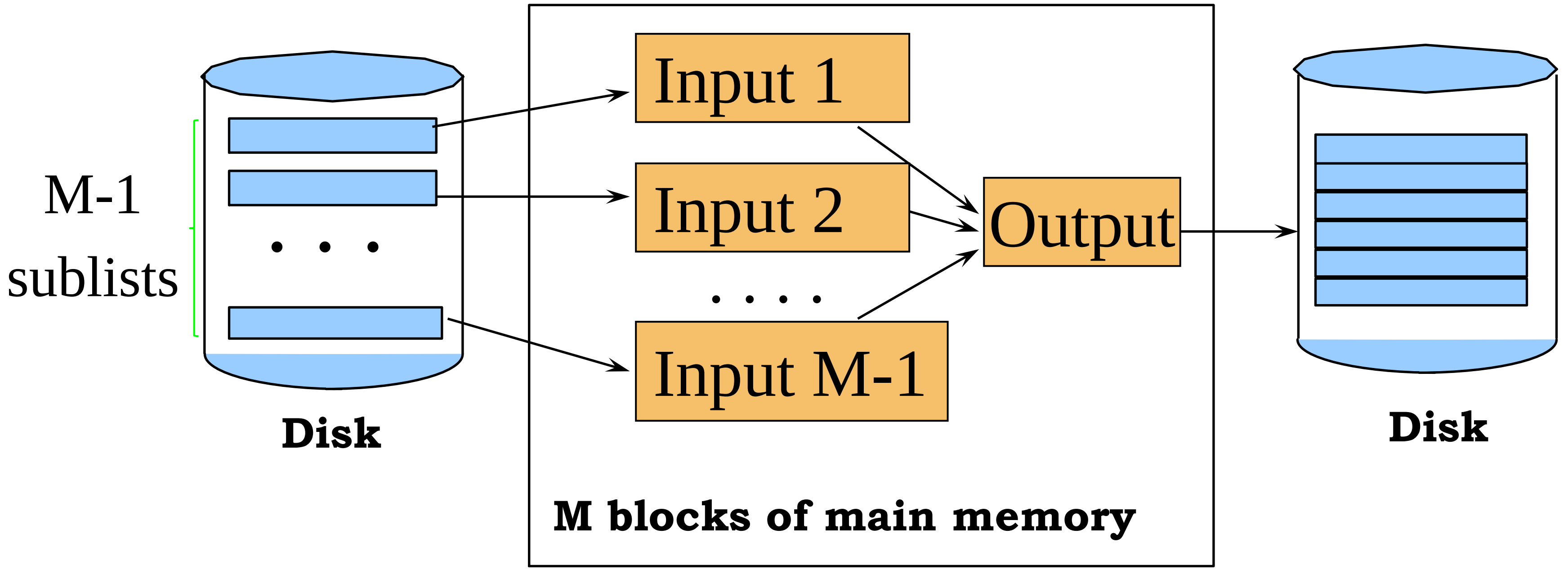


If k is the last pass

1 * 2^k >= N

- Merge M – 1 runs into a single run
- We must have M (M – 1)^k >= B(R)

$$k = \lceil \log_{M-1} \lceil B/M \rceil \rceil$$



Cost of External Merge Sort

- Number of passes: $1 + \lceil \log_{M-1} \lceil B / M \rceil \rceil$
- Cost = 2B * (# of passes)

M=5, B(R)=108

- E.g., with 5 buffer pages, to sort 108-page file:
  - Pass 0: produces $\lceil 108/5 \rceil$ = 22 runs (21 sorted runs of 5 pages each + last run of only 3 pages)
  - Pass 1: $\lceil 22/4 \rceil$ = 6 (5 sorted runs of 20 pages each + last run or only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Example Illustrated

21 runs   1 run

Sorting pass produces: 5 5 5 5 5 … 5 5 3

1^{st} merging pass (4-way merging): …

20 20 20 20 20 8

2^{nd} merging pass

80   28

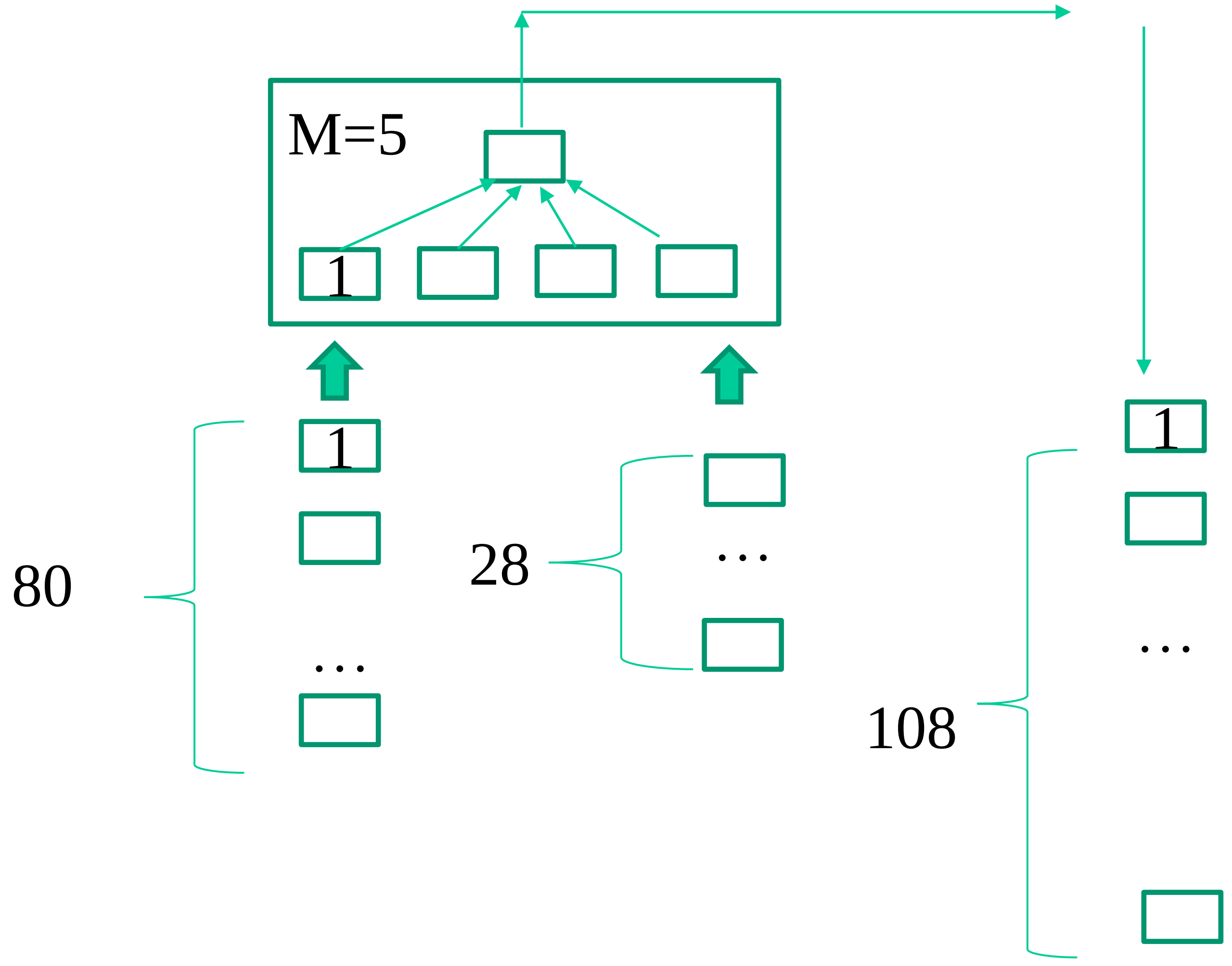
3^{rd} merging pass

108

\# of passes: 1+ceil(log_4(N))

N = # of runs by sorting
= 22
= ceil(108/5)
= ceil(B(R)/M)

Example



Sorting 1TB using 1GB Memory

- B(R) = 1TB/4KB = 256M (blocks), M = 1GB/4KB = 256K (pages)

- Sorting phase produces 1024 runs = 1K runs

- Merging:
  - – Can do: 1GB/4KB-1 = 256K-1 ways of merging
  - – Can we finish merging in one merging pass?

of passes

- 2-way: 1+log_2(N)
  - m = 3
  - m’ = 1
  - B = N

- (m-1)-way of merging:
  - 1 + log_(m-1) (B/m’)