# *External Sorting*

# *Why Sort?*

- A classic problem in computer science!
- Data requested in sorted order
  - e.g., find students in increasing *gpa* order
- Sorting is first step in *bulk loading* B+ tree index.
- Sorting useful for eliminating *duplicate copies* in a collection of records
- *Sort-merge* join algorithm involves sorting.

- Problem: sort 10GB of data with 1MB of RAM.

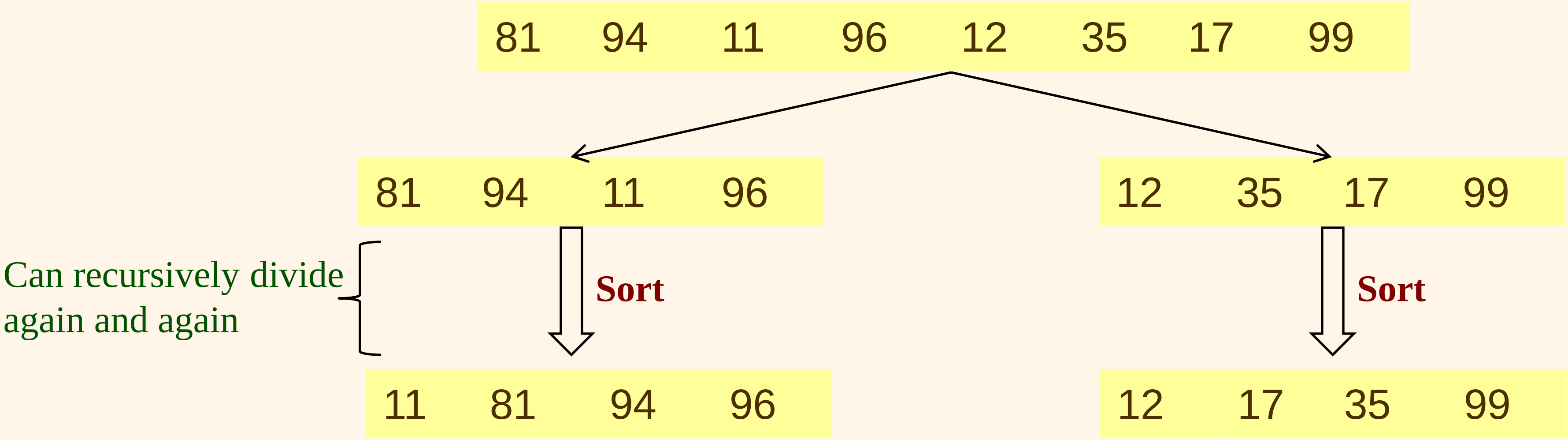
# *Using secondary storage effectively*

- General Wisdom :
  - I/O costs dominate
  - Design algorithms to reduce I/O

# *Overview on Merge-Sort*

- **Merge :** Merge two sorted lists and repeatedly choose the smaller of the two “heads” of the lists

- **Merge Sort:** Divide records into two parts; merge-sort those recursively, and then merge the lists.



81 94 11 96 12 35 17 99

81 94 11 96

12 35 17 99

Can recursively divide again and again

Sort

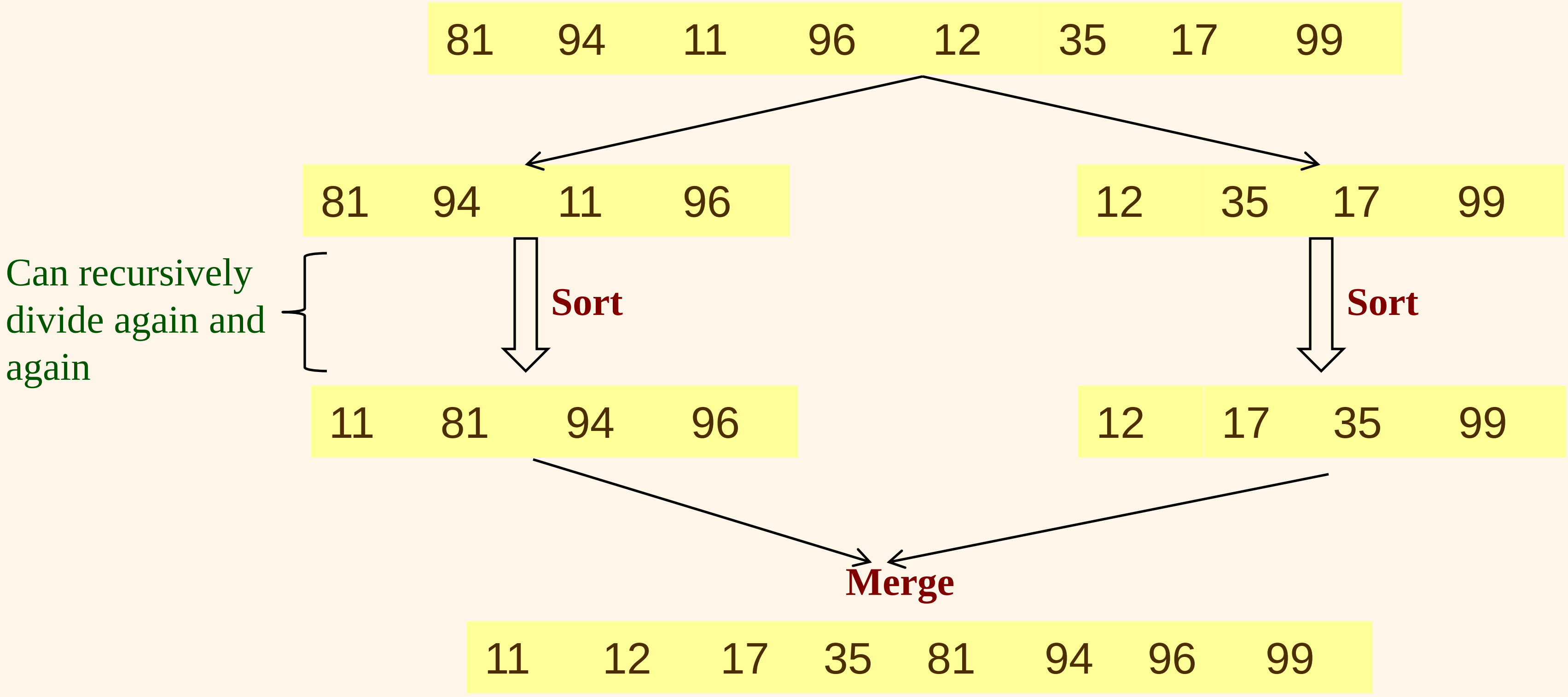
Sort

11 81 94 96

12 17 35 99

# *Overview on Merge-Sort (Cont'd)*

- **Merge :** Merge two sorted lists and repeatedly choose the smaller of the two "heads" of the lists



81 94 11 96 12 35 17 99

81 94 11 96

12 35 17 99

Can recursively divide again and again

Sort

Sort

11 81 94 96

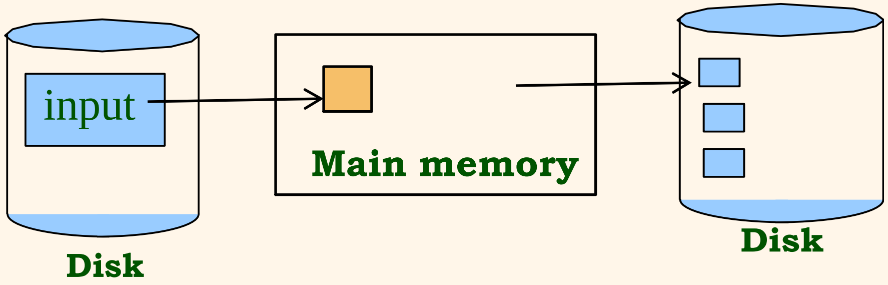
12 17 35 99

Merge

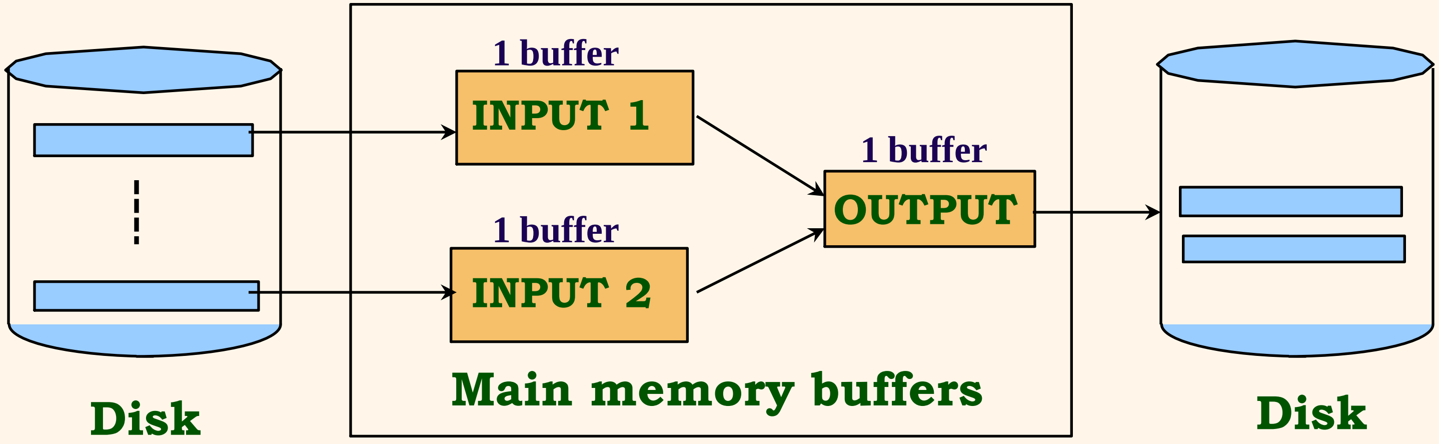
11 12 17 35 81 94 96 99

# 2-Way Sort: Requires 3 Buffers

- **Phase 1: PREPARE.**
  - Read a page, sort it, write it.
  - only one buffer page is used



- **Phase 2, 3, …, etc.: MERGE:**
  - Three buffer pages used.

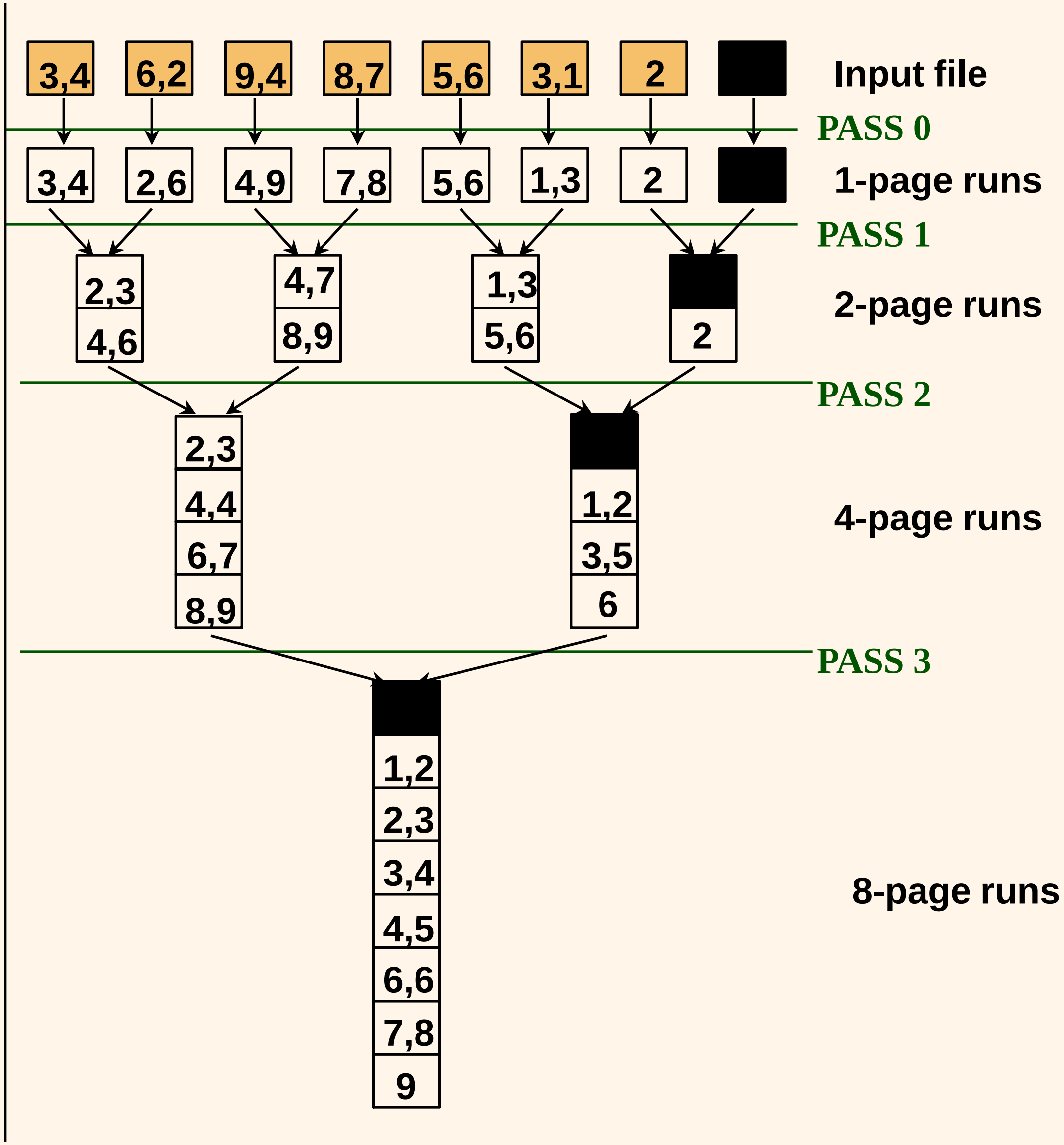


# *Two-Way External Merge Sort*

- *Idea:* ***Divide and conquer:*** sort subfiles and merge into larger sorts

Pass 0 ➔ Only one memory block is needed

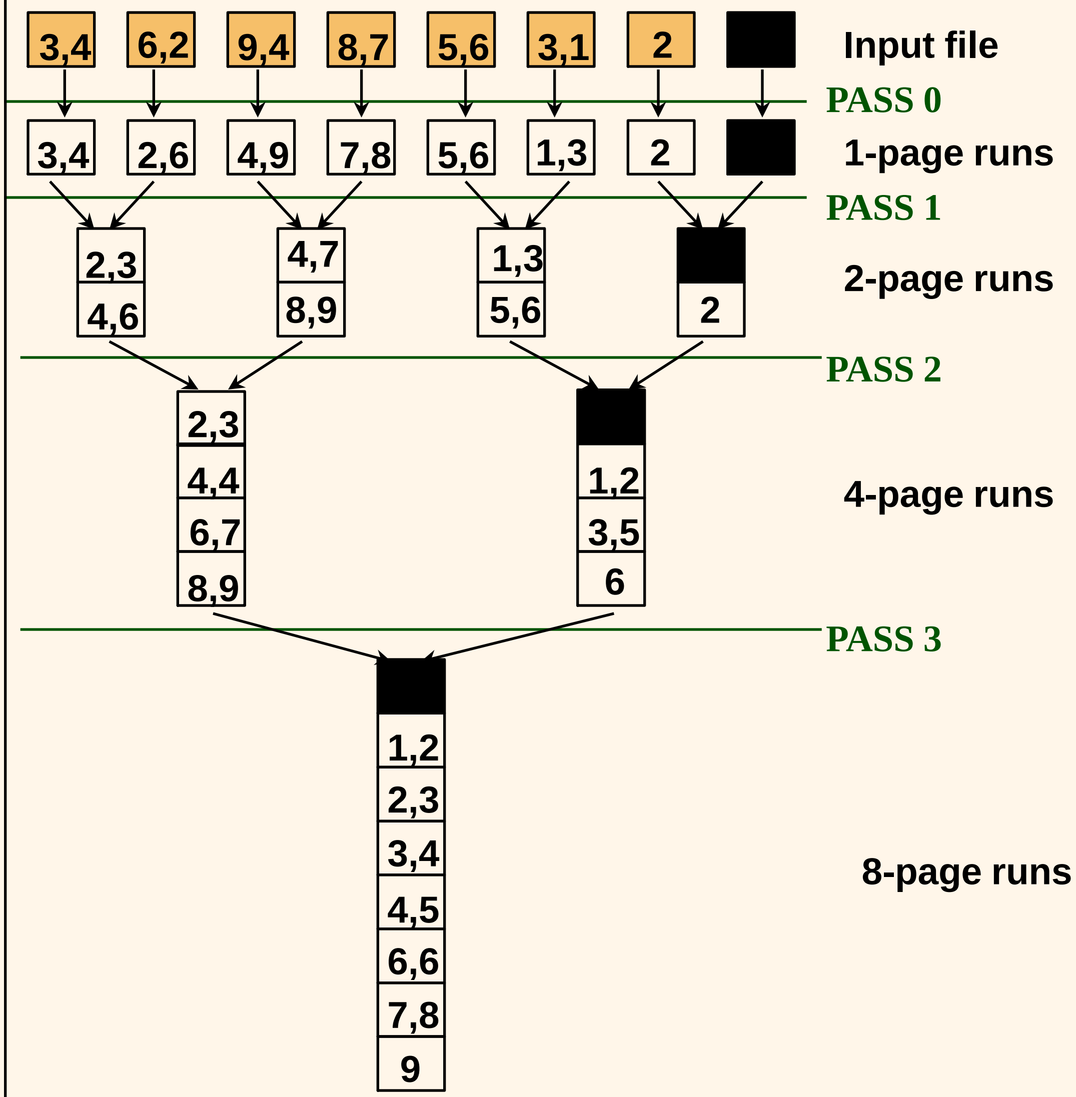
Pass I > 0 ➔ Only three memory blocks are needed



# *Two-Way External Merge Sort*

- Costs for one pass: all pages
- # of passes : height of tree
- Total cost : product of above

Notice: We ignored the CPU cost to sort a block in memory or merge two blocks



Input file: 3,4 | 6,2 | 9,4 | 8,7 | 5,6 | 3,1 | 2

PASS 0

1-page runs: 3,4 | 2,6 | 4,9 | 7,8 | 5,6 | 1,3 | 2

PASS 1

2-page runs: 2,3 4,6 | 4,7 8,9 | 1,3 5,6 | 2

PASS 2

4-page runs: 2,3 4,4 6,7 8,9 | 1,2 3,5 6

PASS 3

8-page runs: 1,2 2,3 3,4 4,5 6,6 7,8 9

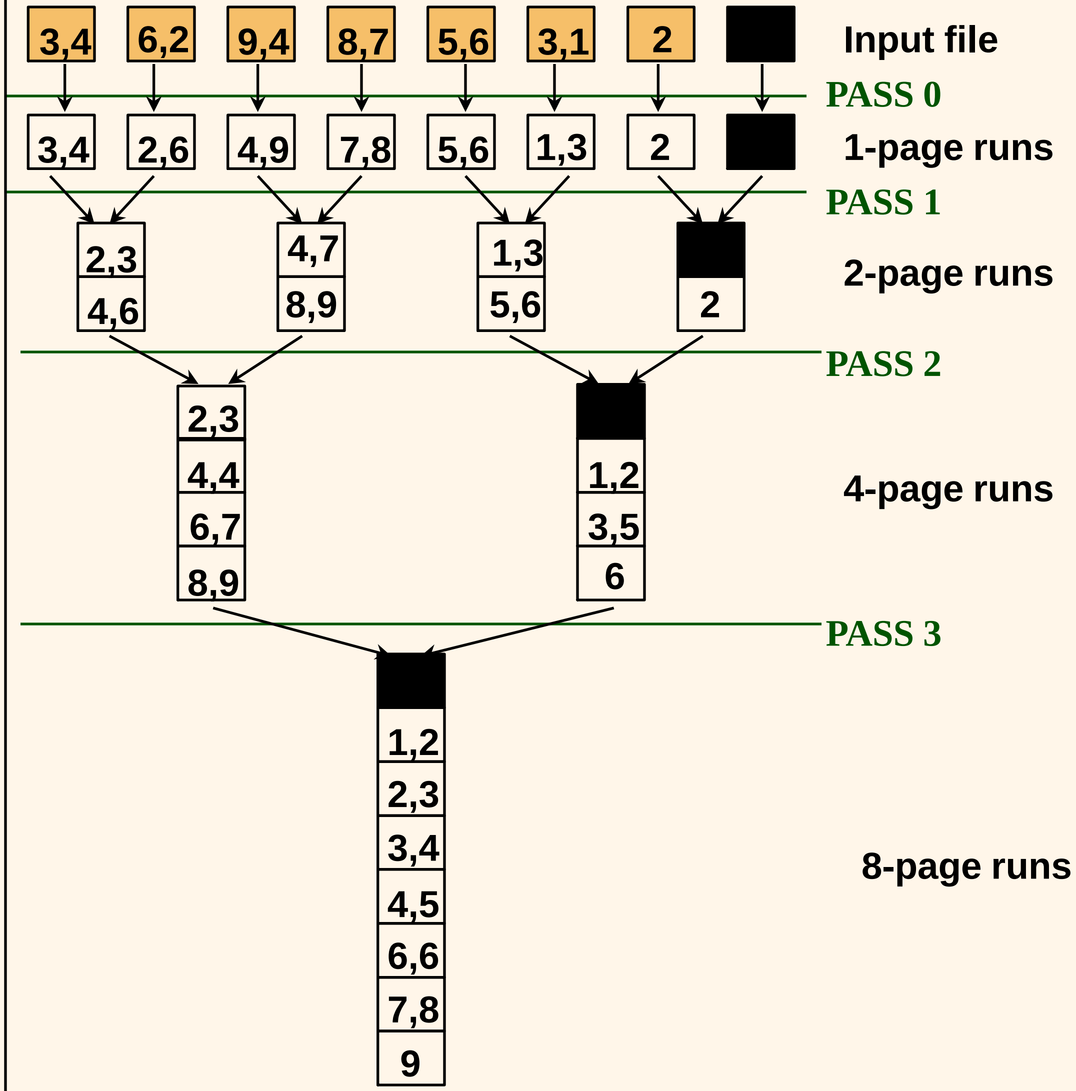
# *Two-Way External Merge Sort*

- Each pass we read + write each page in file.
- N pages in file => 2N
- Number of passes

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

- So total cost is:

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

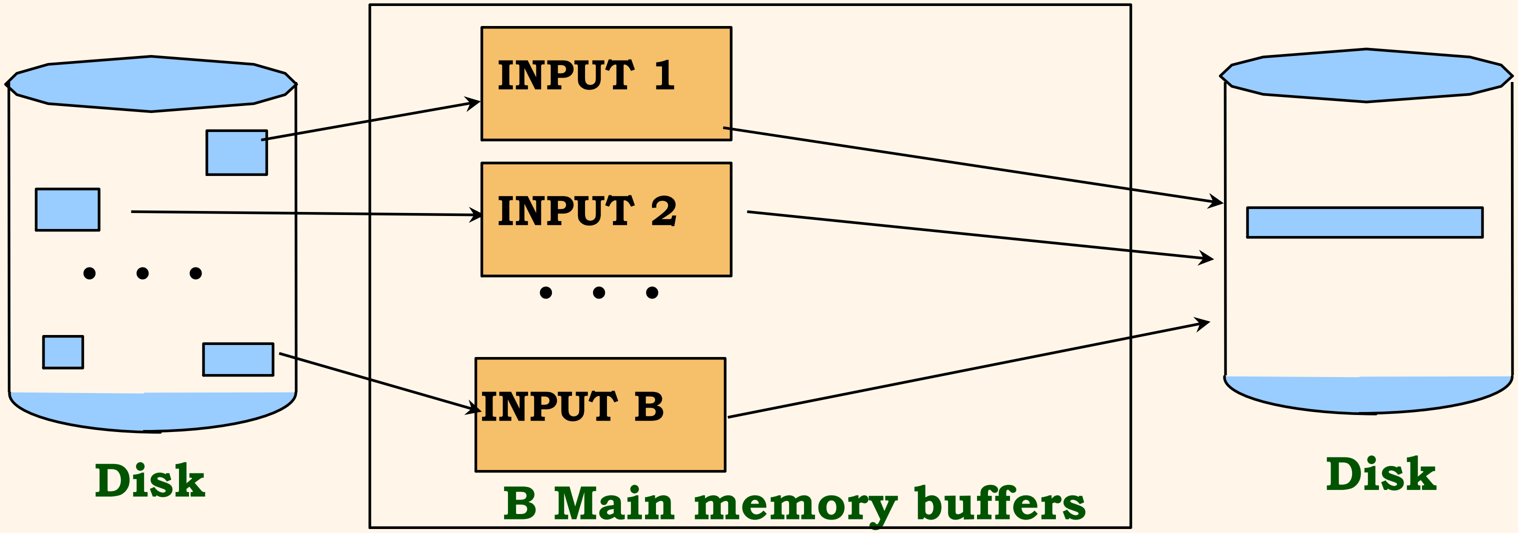


# *External Merge Sort*

- What if we had more buffer pages?
- How do we utilize them wisely ?

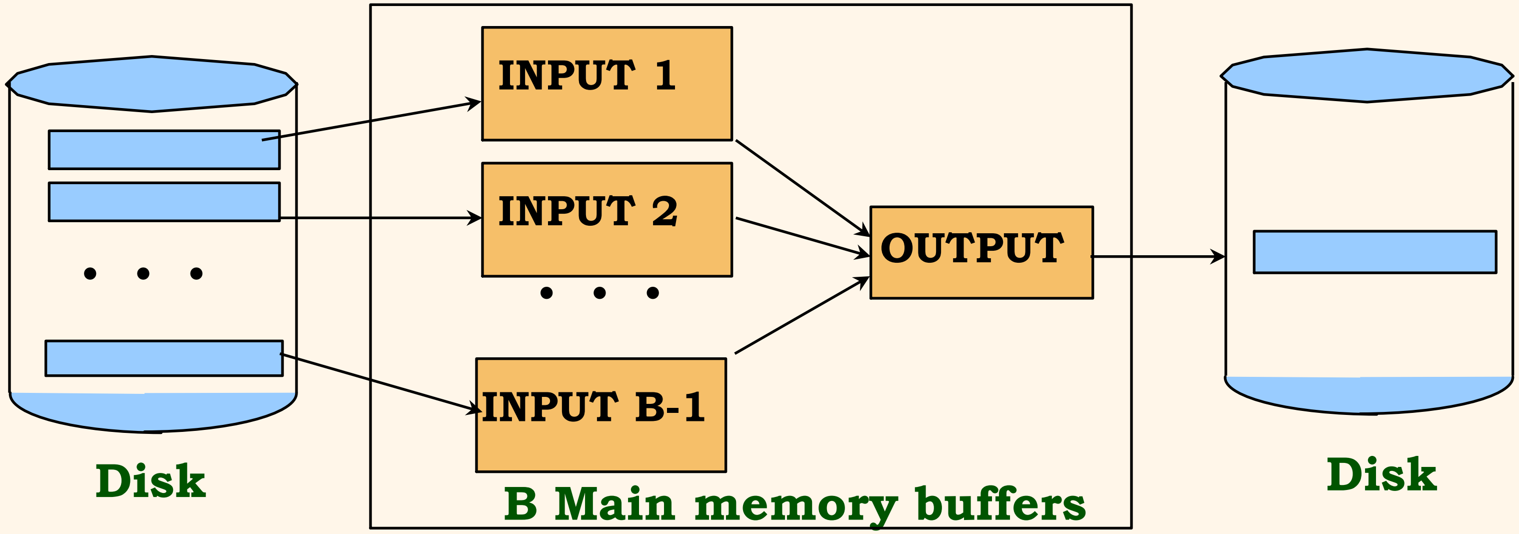
-→ Two main ideas !

# *Phase 1 : Prepare*



- Construct as large as possible starter lists.
- → Will reduce the number of needed passes

# *Phase 2 : Merge*

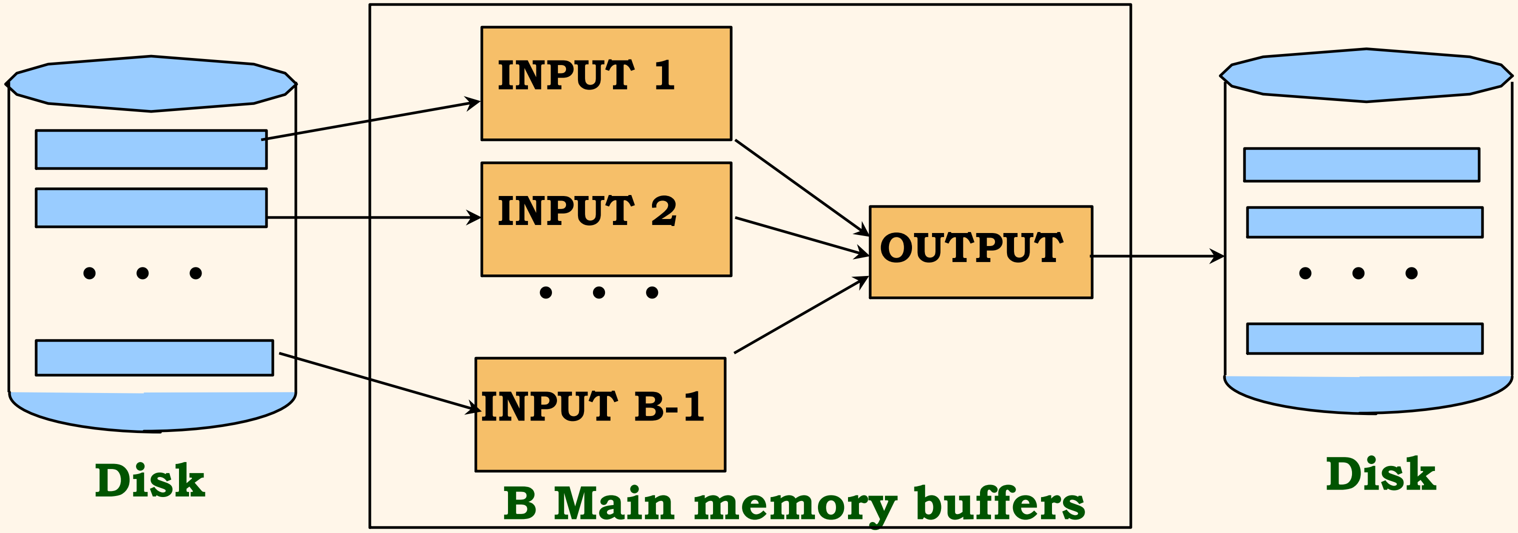


- Merge as many sorted sublists into one long sorted list.
- Keep 1 buffer for the output
- Use B-1 buffers to read from B-1 lists

# *General External Merge Sort*

☒ ***How can we utilize more than 3 buffer pages?***

- To sort a file with *N* pages using *B* 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)

# *Example*

- Buffer : with 5 buffer pages
- File to sort : 108 pages

  - Pass 0:
    - Size of each run?
    - Number of runs?

  - Pass 1:
    - Size of each run?
    - Number of runs?

  - Pass 2:  ???

# *Example*

- Buffer : with 5 buffer pages
- File to sort : 108 pages
  - 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: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

- Total I/O costs: ?

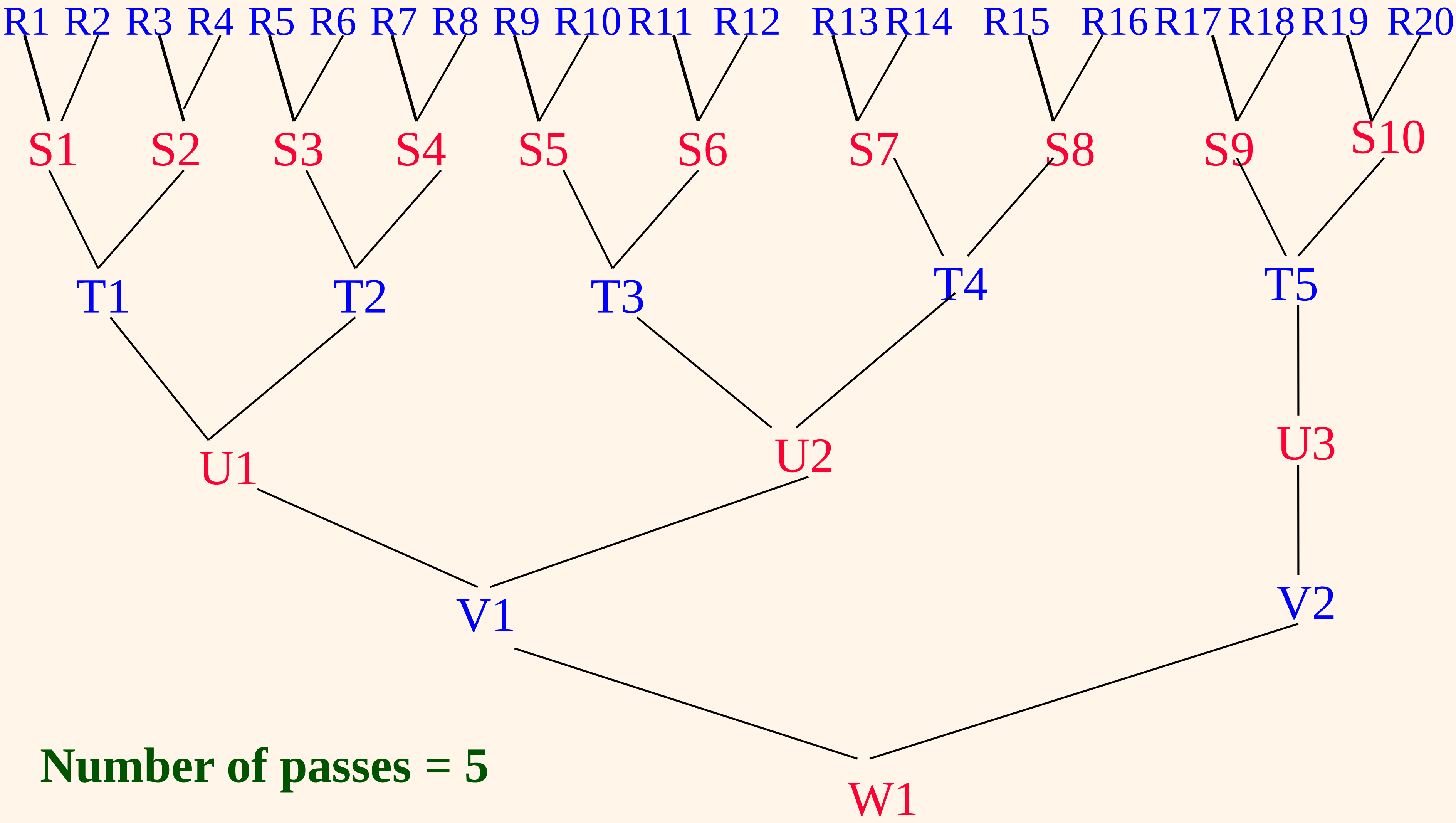
# *Example*

- Buffer : with 5 buffer pages
- File to sort : 108 pages

  - 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: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

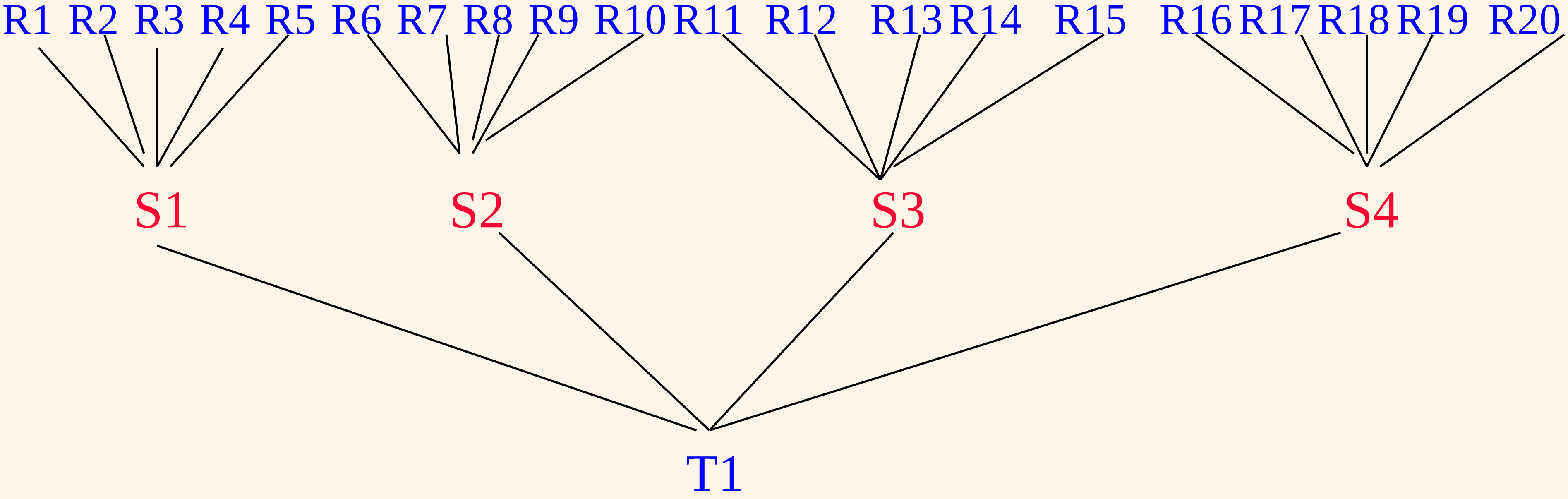
- Total I/O costs: 2*N * (4 passes)

# *Example: 2-Way Merge for 20 Runs*


R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R17 R18 R19 R20
S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
T1 T2 T3 T4 T5
U1 U2 U3
V1 V2
W1

**Number of passes = 5**

# *Example: 5-Way Merge for 20 Runs*


R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R17 R18 R19 R20
S1
S2
S3
S4
T1

**Number of passes = 2**

# *Number of Passes of External Sort*

- gain of utilizing all available buffers
- importance of a high fan-in during merging

**#Buffers available in main-memory**

**#Pages in File**

| 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 |