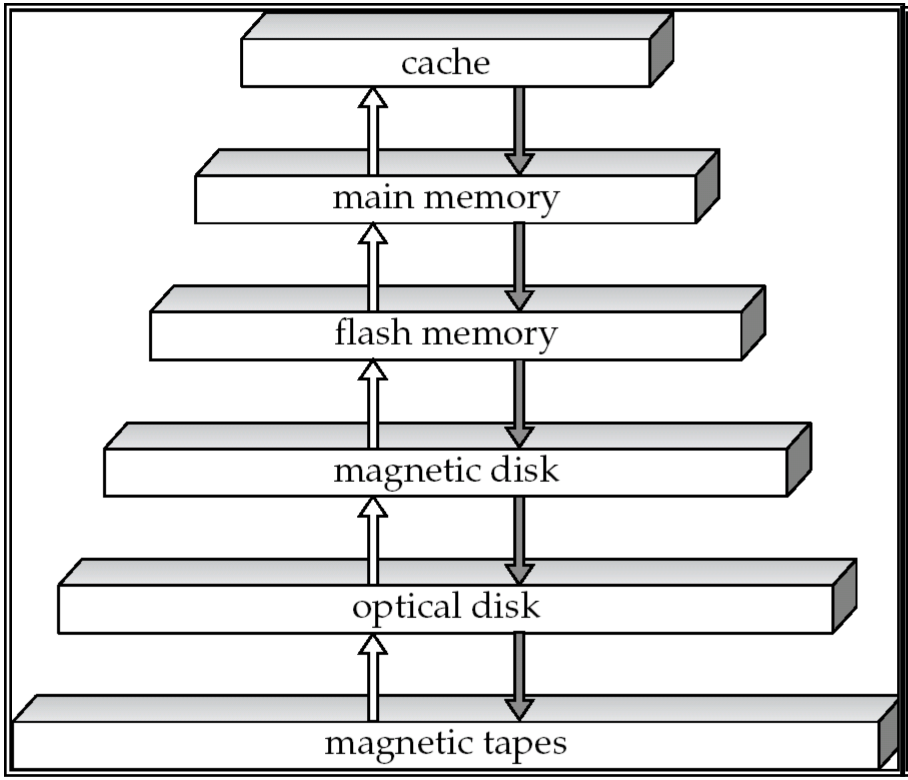
Secondary Storage

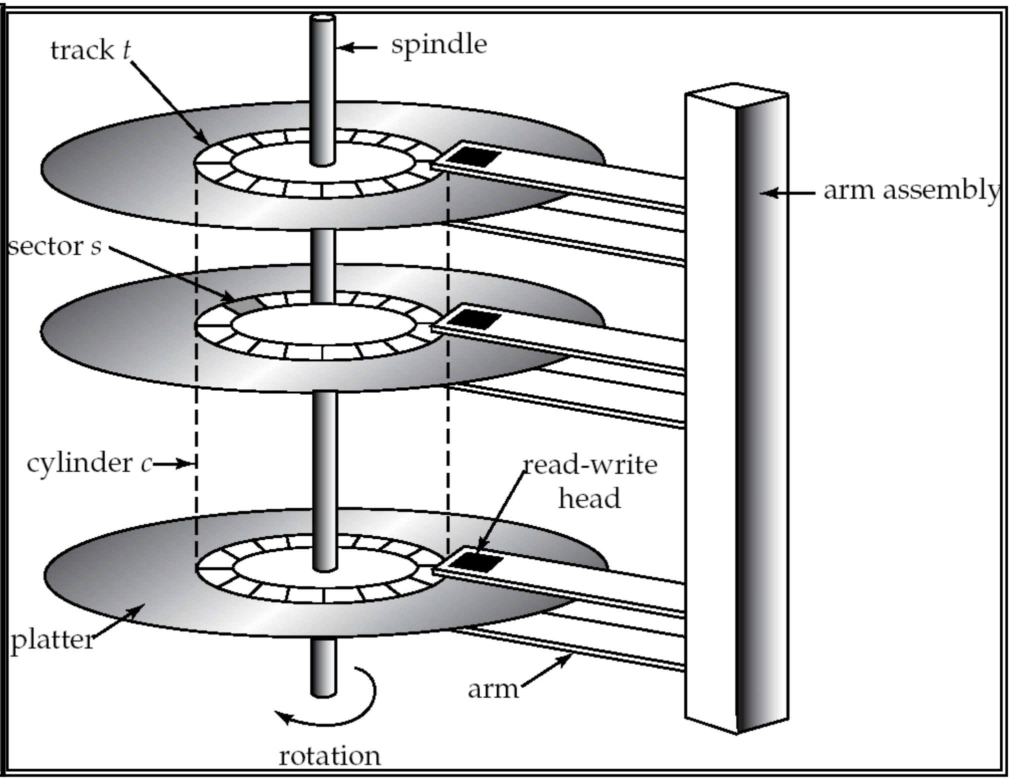
- External Sort
- B-tree Index
- Other Indices

Data Structures

Secondary Storage: *Basics*

- **Main Memory vs. Secondary Storage**
  - CPU time vs. I/O time (seek + latency + transfer)

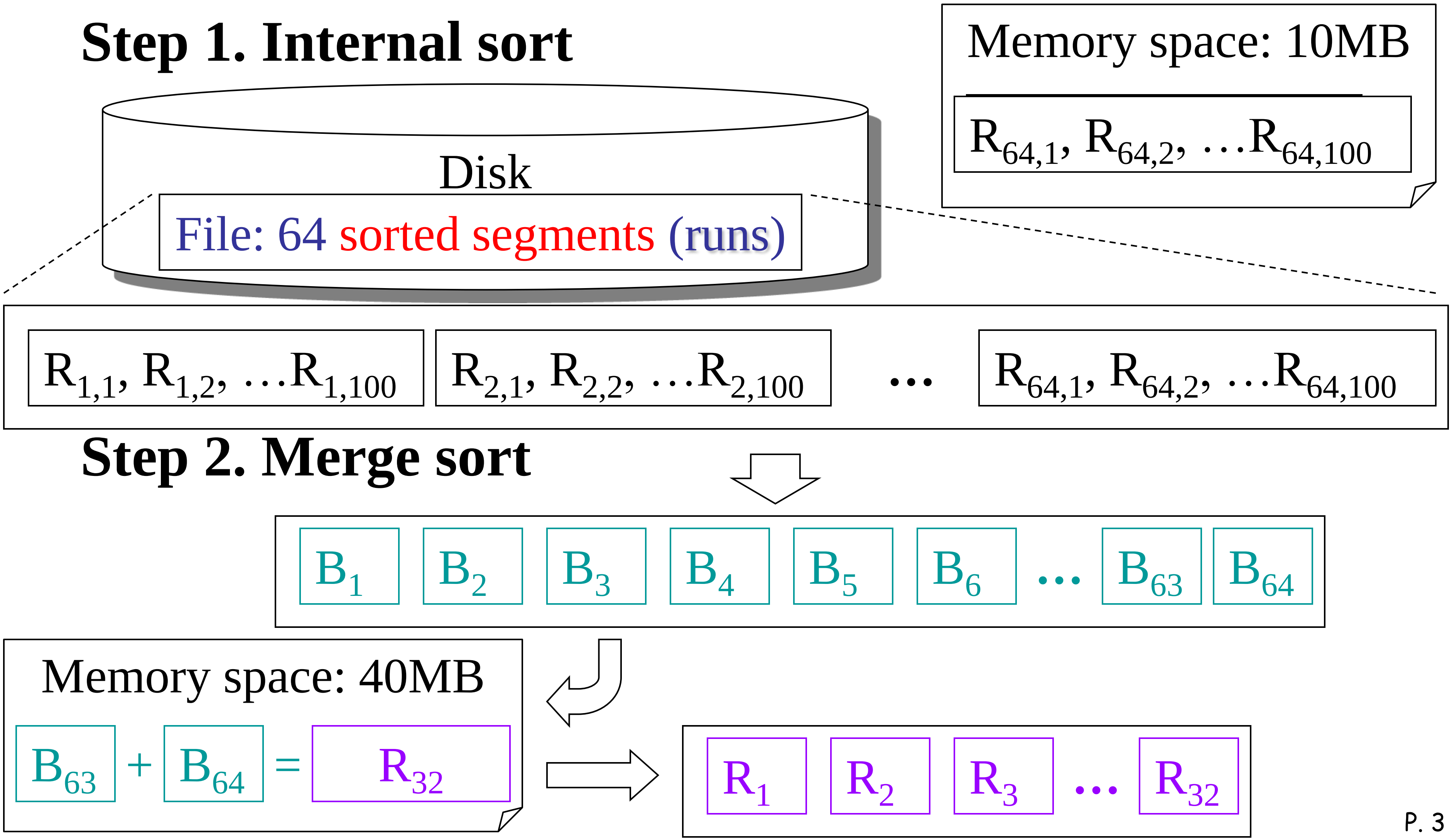




Secondary Storage: *External Sort*

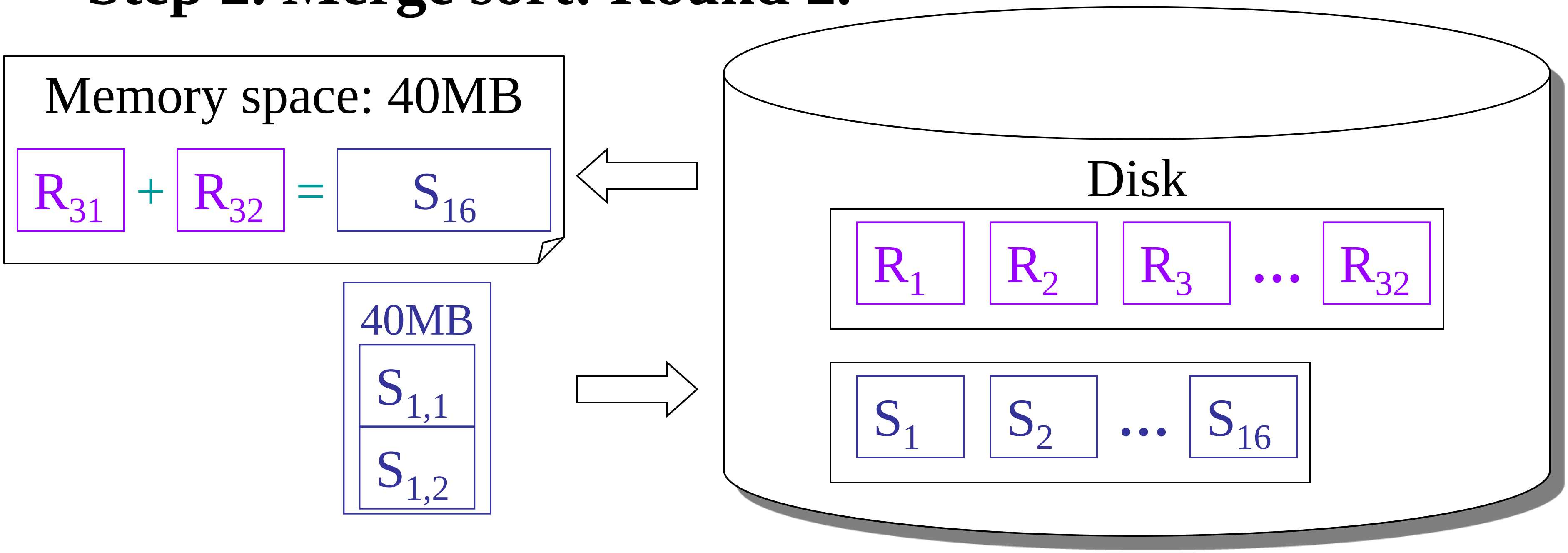
Step 1. Internal sort

Step 2. Merge sort

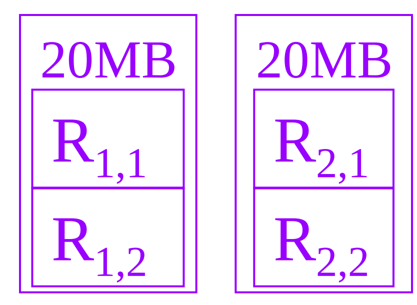


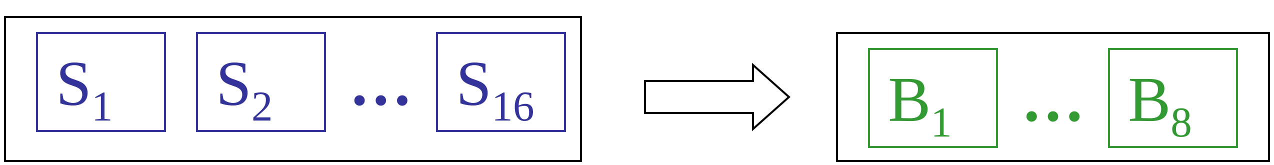
Secondary Storage: *External Sort*

Step 2. Merge sort: Round 2.



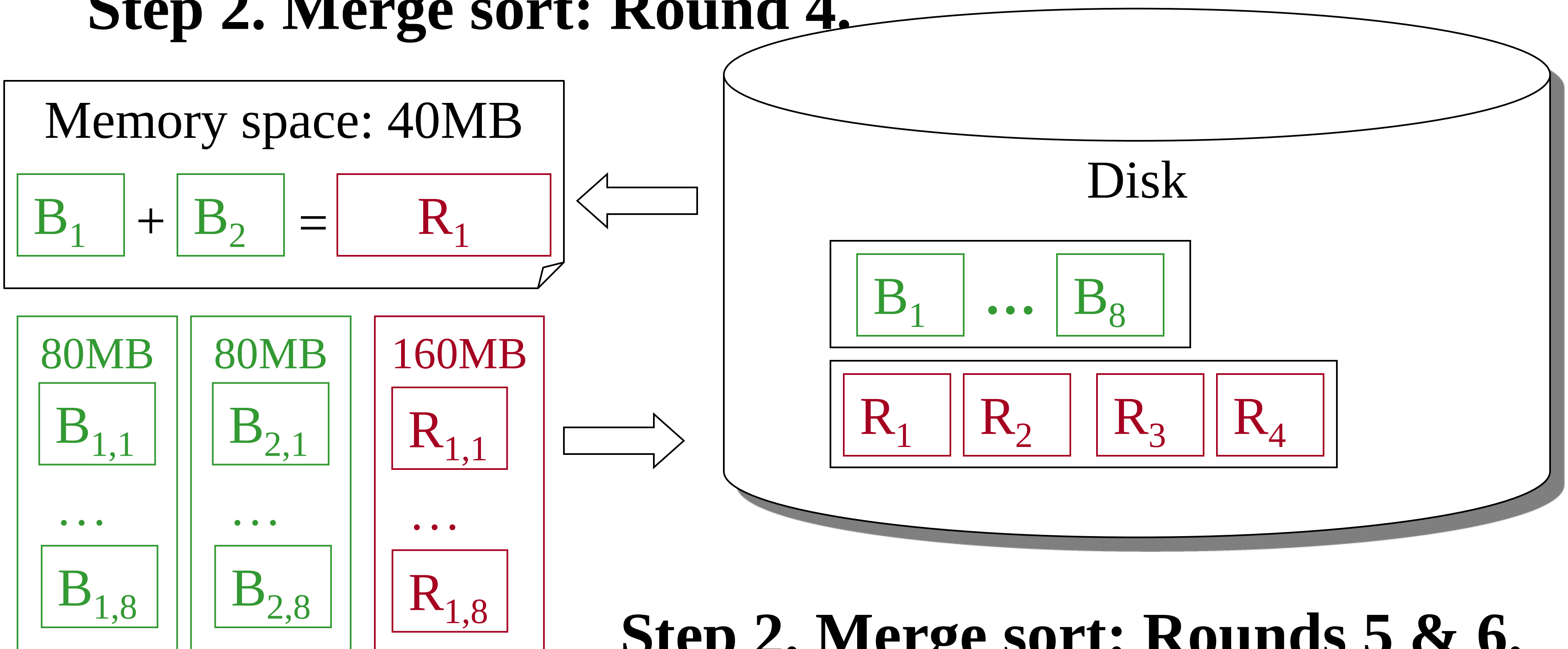
Step 2. Merge sort: Round 3.



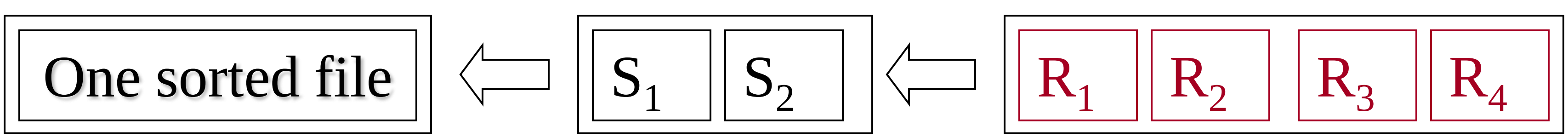


Secondary Storage: *External Sort*

Step 2. Merge sort: Round 4.



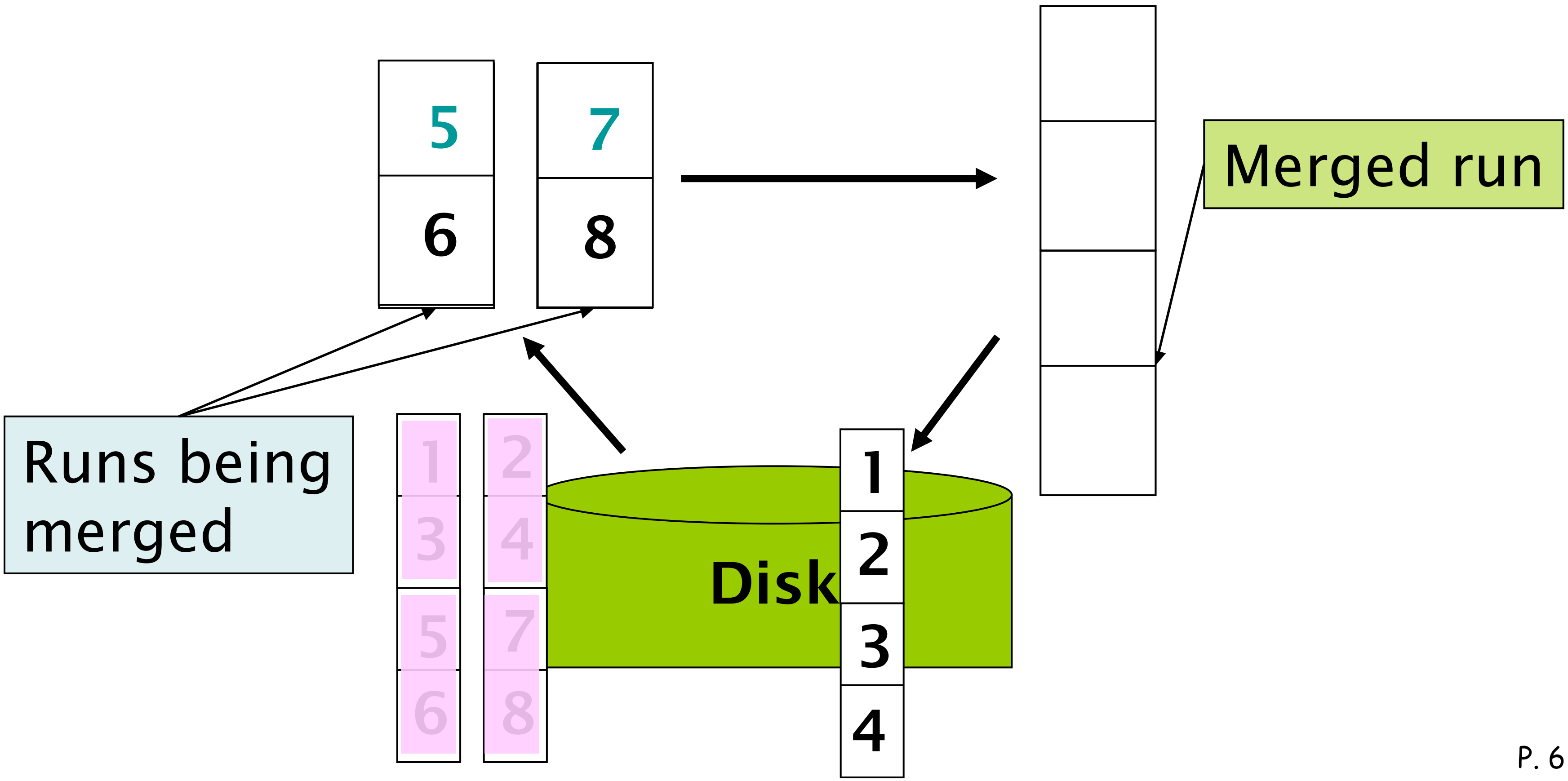
Step 2. Merge sort: Rounds 5 & 6.



External Sort: *2-way Merge*

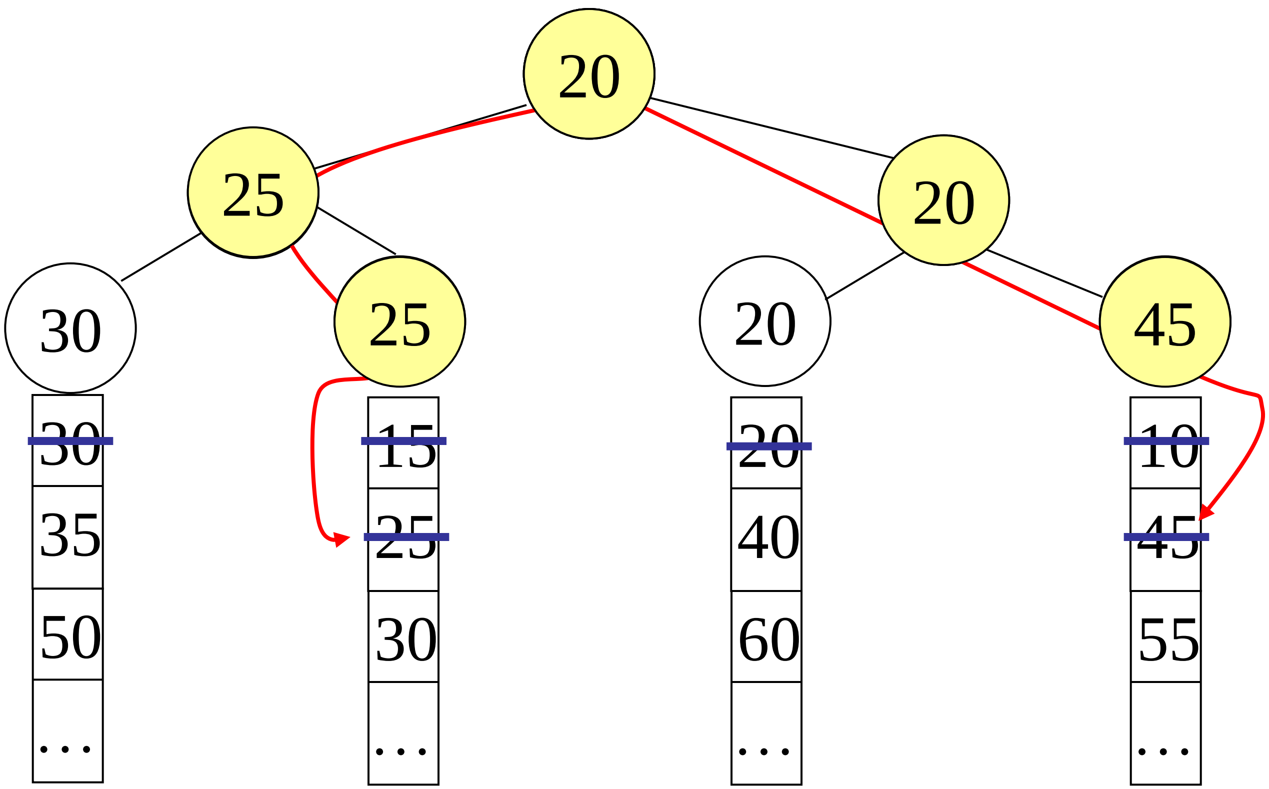
Data Structures

- Input buffer
- Output buffer



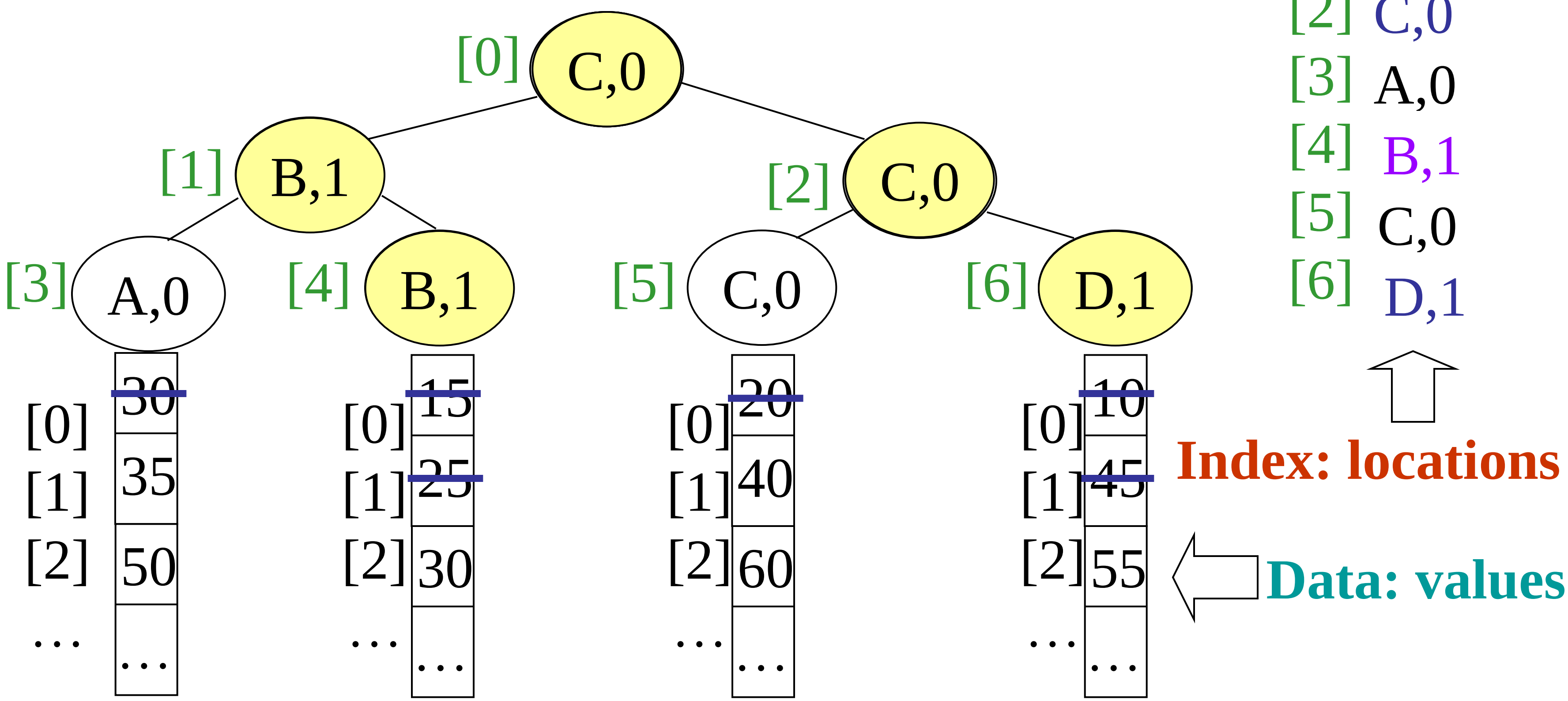
K-way Merge: *Selection Tree*

- Given *k* sorted runs to be merged, a data structure named ***selection tree*** can reduce the number of comparisons for *finding the next smallest element*.



K-way Merge: *Selection Tree*

- **Selection tree on *k* sorted runs**



Secondary Storage: *File Structures*

- **Record**
  - Field
    - **name**
    - **value**

Last Name: ‘Jordan’
First Name: ‘Michael’
Score: 30
Assist: 5
Rebound: 6
Champion: 6
MVP: 5

Last Name: ‘Chamberlain’
First Name: ‘Wilt ’
Score: 30
Assist: 4
Rebound: 22
Champion: 2
MVP: 4

Last Name: ‘Johnson’
First Name: ‘Earvin’
Score: 20
Assist: 11
Rebound: 7
Champion: 5
MVP: 3

Last Name: ‘Abdul-Jabbar’
First Name: ‘Kareem ’
Score: 25
Assist: 4
Rebound: 11
Champion: 6
MVP: 6

Variable-length vs. *Fixed-length*

- **Offset of fixed-length record *RRN = i***

  (Header record length) + (i – 1) * (record length)

  RRN=2: offset = 1+(2-1)*100 = 101 *bytes*

RRN: relative record number

[1] Jordan,Michael,30,5,6,6,5 ← unused space →

[2] Johnson,Earvin,20,11,7,5,3 ← unused space →

… …

4 Jordan|Michael|30|5|6|6|5|Johnson|Earvin|20|11|7|5|3|…

Magic?

01 27 54 …

Fixed-length Records: *Free List*

- **List head**
  - Stored in the file header
  - Keep the *RRN* of one *deleted record*
- **Use one field of the deleted record to keep the *RRN* of *the next deleted record***
- **Regard these *RRNs* (*offset*) as pointers in the file**

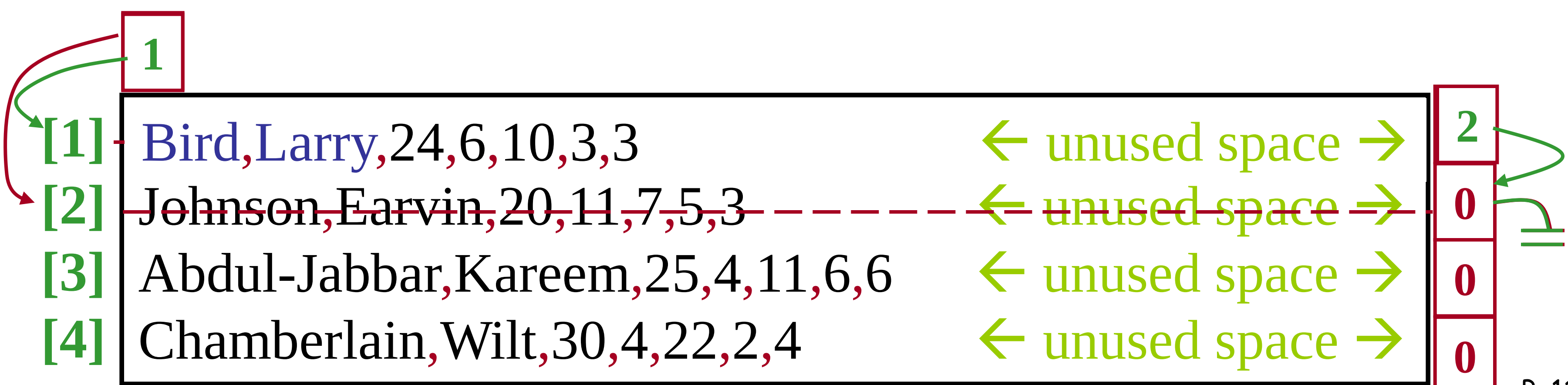


Illustration I: *Key Sort*

(KEY RRN)
(**99**, 1)
(**95**, 2)
(**88**, 3)
(**99**, 4)
(**95**, 5)
…

(KEY RRN)
(99, **1**)
(99, **4**)
…
(95, **2**)
(95, **5**)
…
(88, **3**)
…

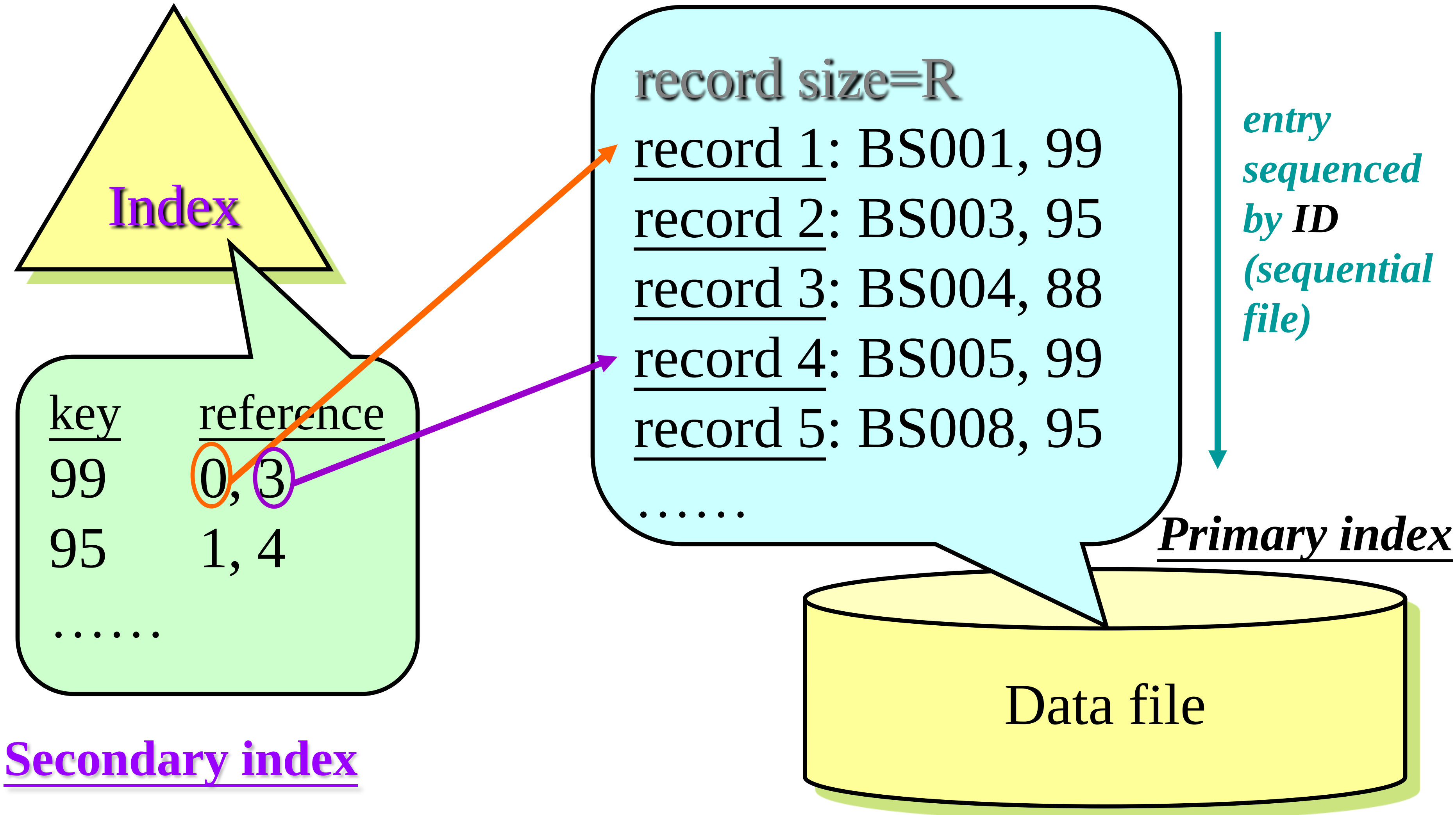
REC_COUNT (number of records)
record 1: BS001, **99**, Lee, …
record 2: BS003, **95**, Lin, …
record 3: BS004, **88**, Wang, …
record 4: BS005, **99**, Liu, …
record 5: BS008, **95**, Chen, …

IN_FILE

record 1: BS001, 99, Lee, …
record 4: BS005, 99, Liu, …
…
record 2: BS003, 95, Lin, …

OUT_FILE

Illustration II: *Secondary Index*



B-tree Index: *Examples*

Data Structures

- **B-tree of order 3**
  - $\lceil 3/2 \rceil$ ~ 3 children → 1~2 keys → 2-3 tree
  - – Insertions: 10, 20, 30, 40, 50, 90, 80, 70, 60, 100

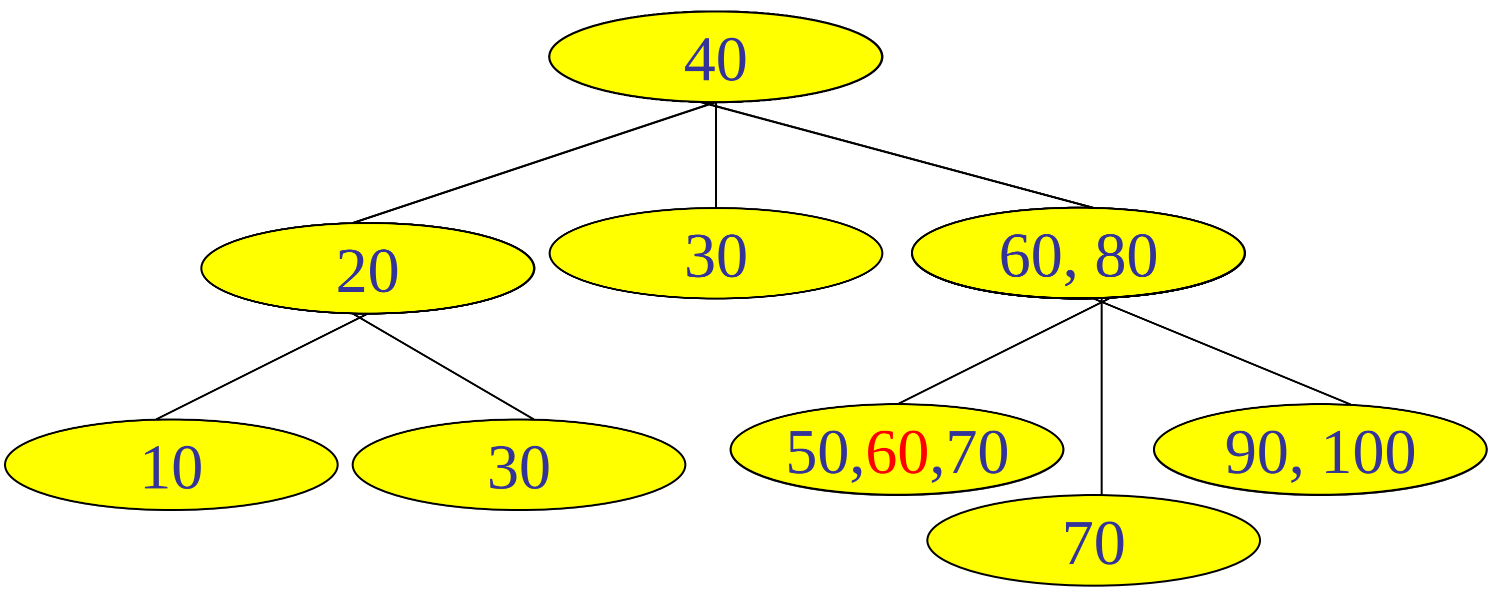


Illustration VI: *B-tree Index*

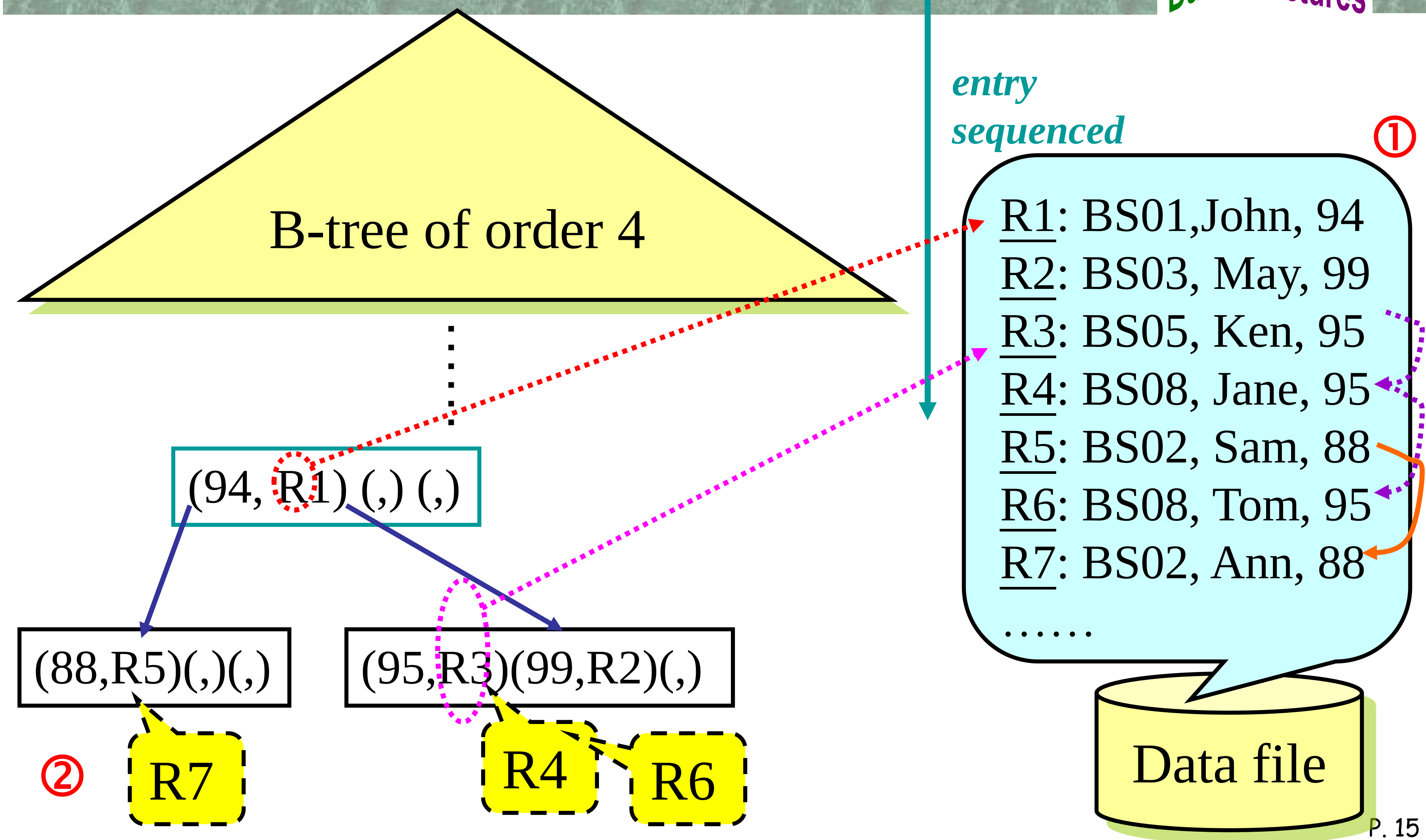
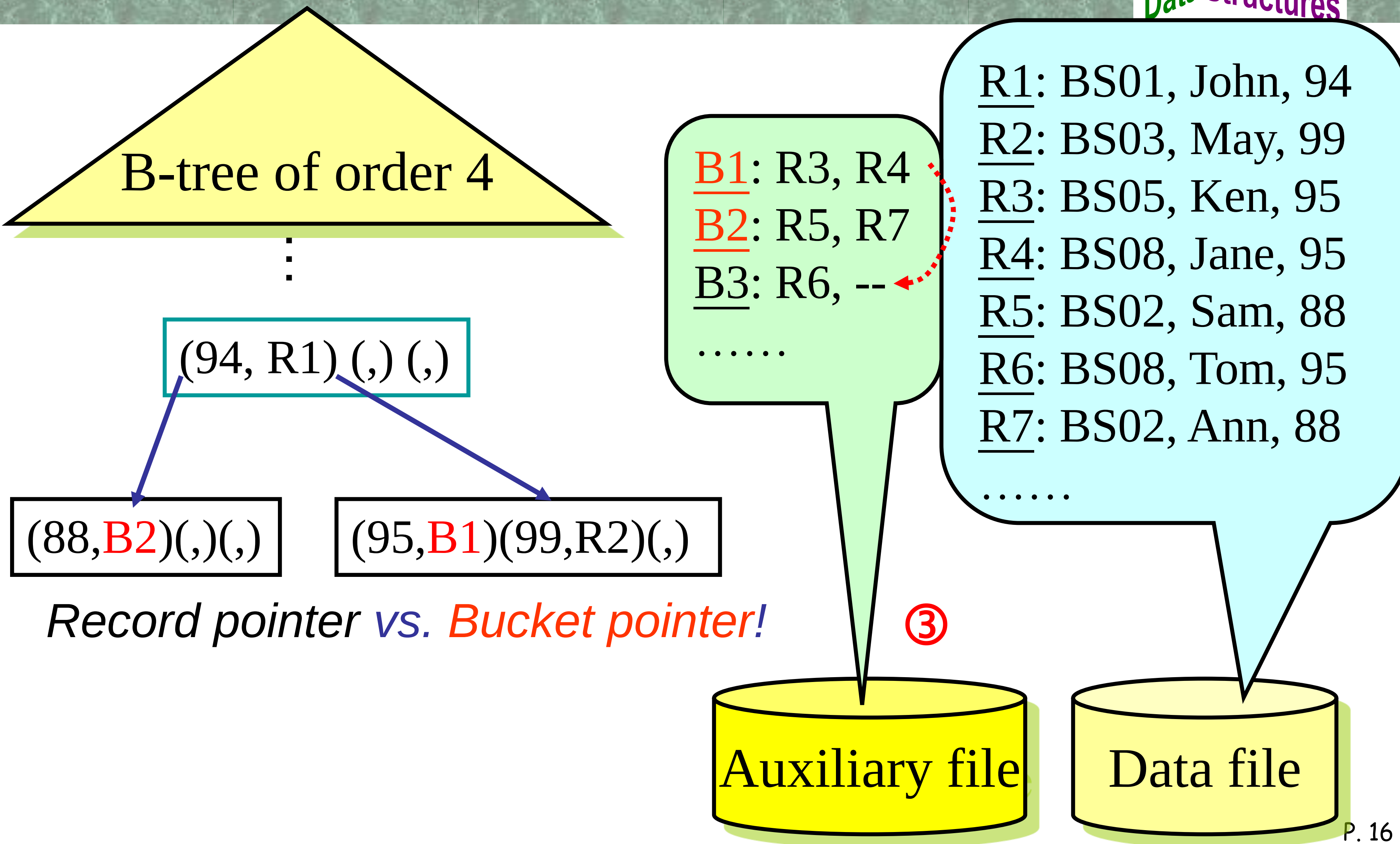
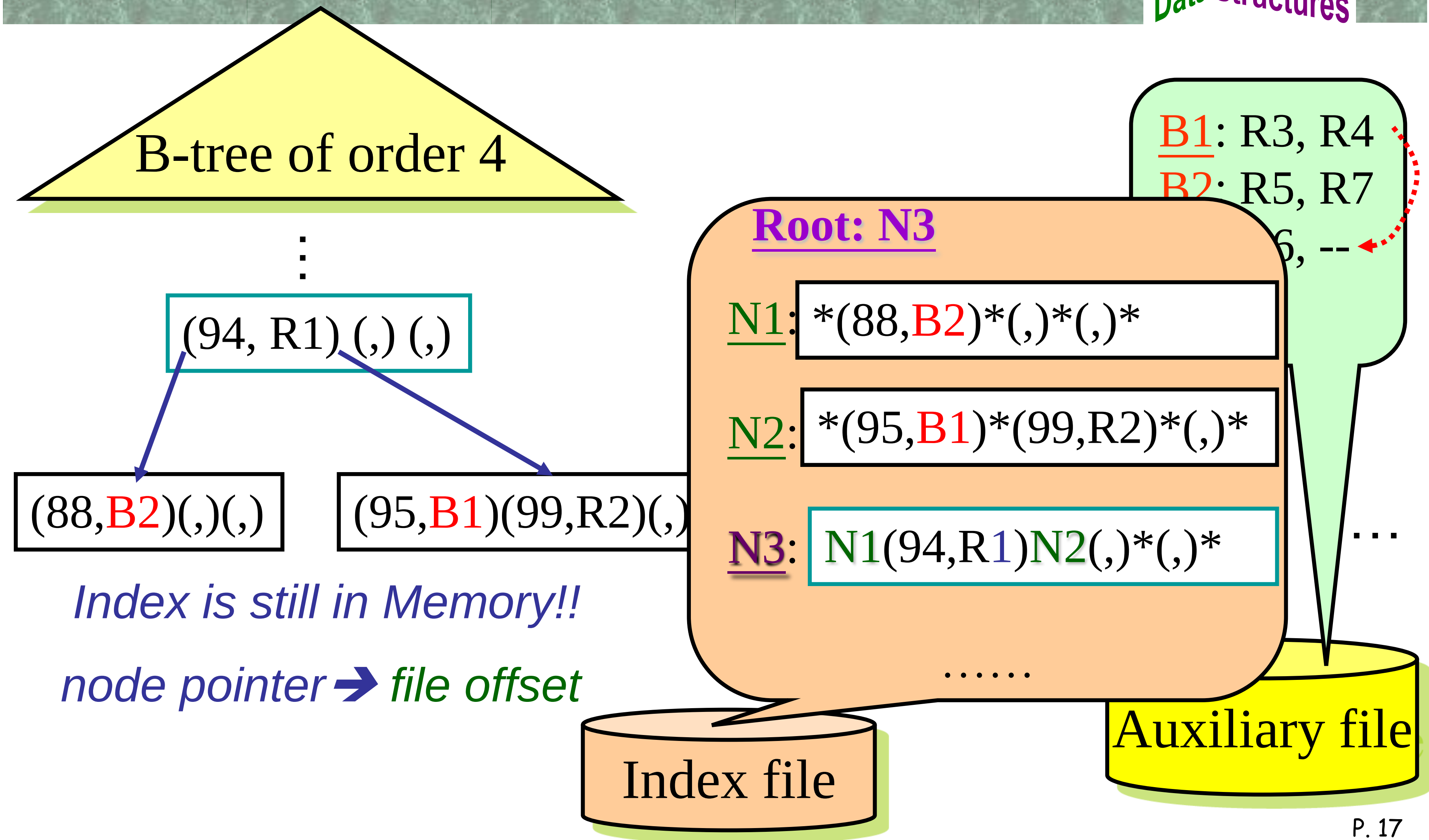


Illustration VII: *B-tree with Buckets*



Record pointer vs. Bucket pointer!

Illustration VIII: *B-tree in File*



Index is still in Memory!!

node pointer ➔ file offset

Illustration IX: *Reload B-tree*

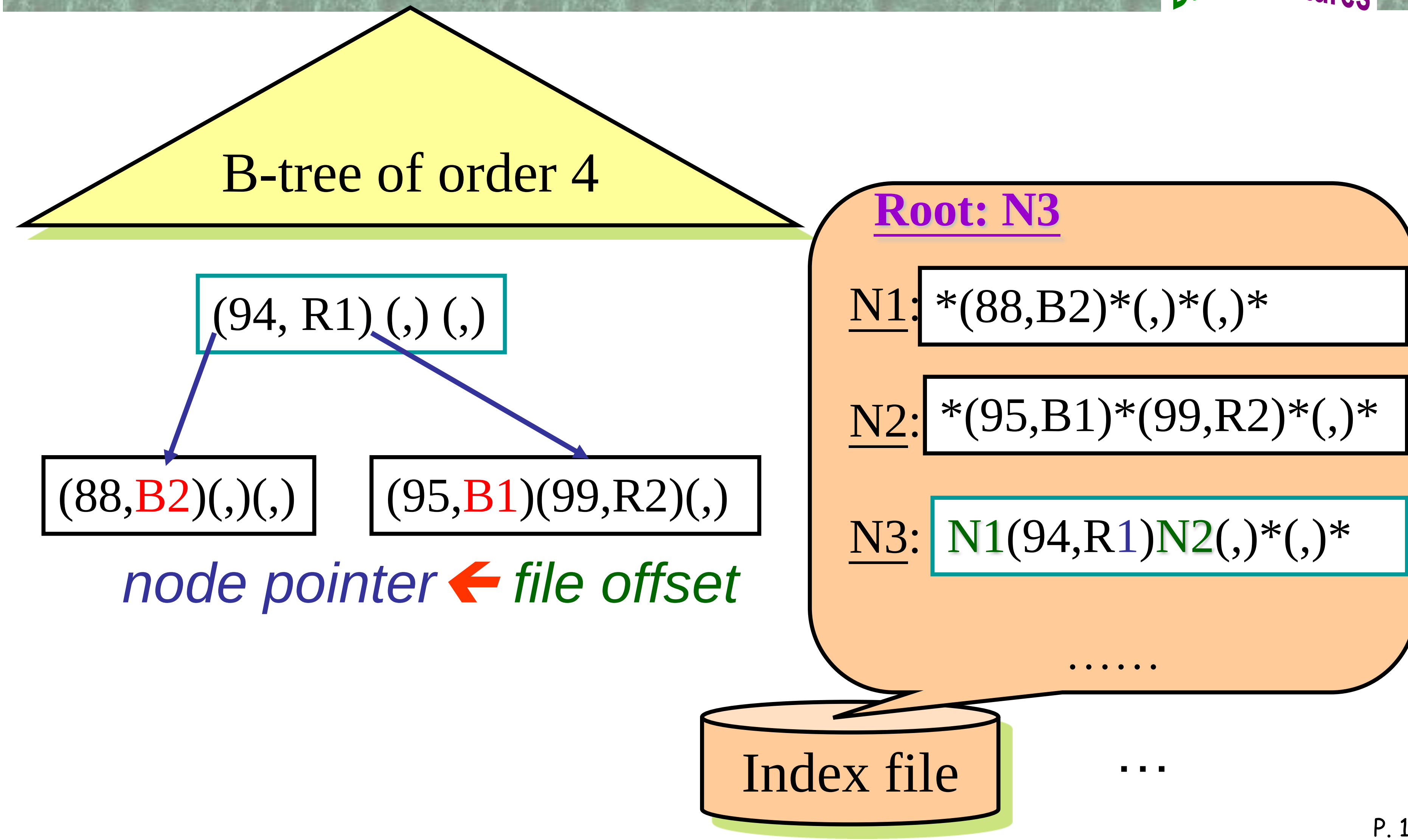
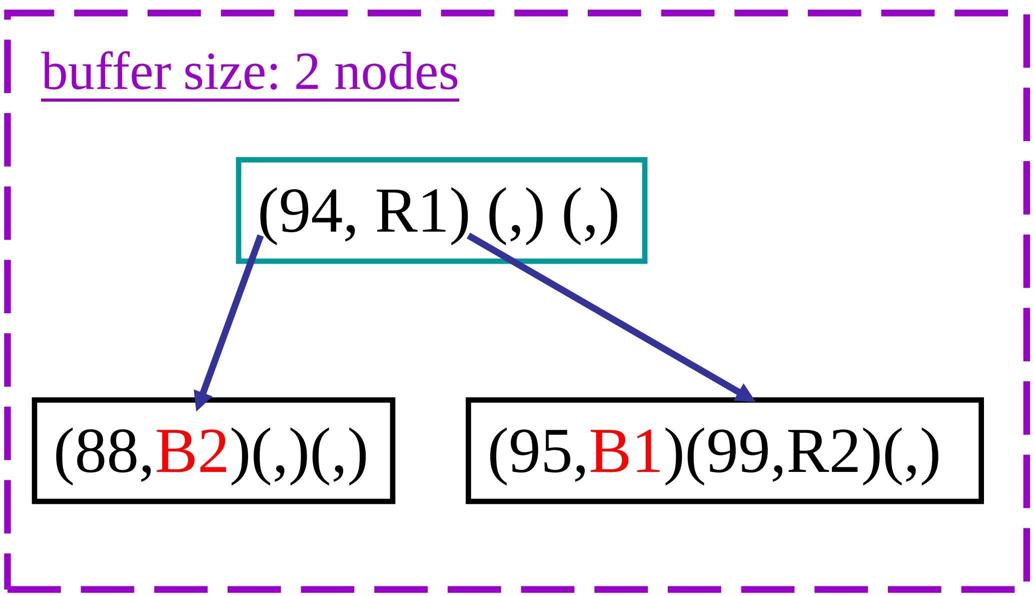


Illustration X: *B-tree File*

buffer management



node pointer ⇔ file offset

④

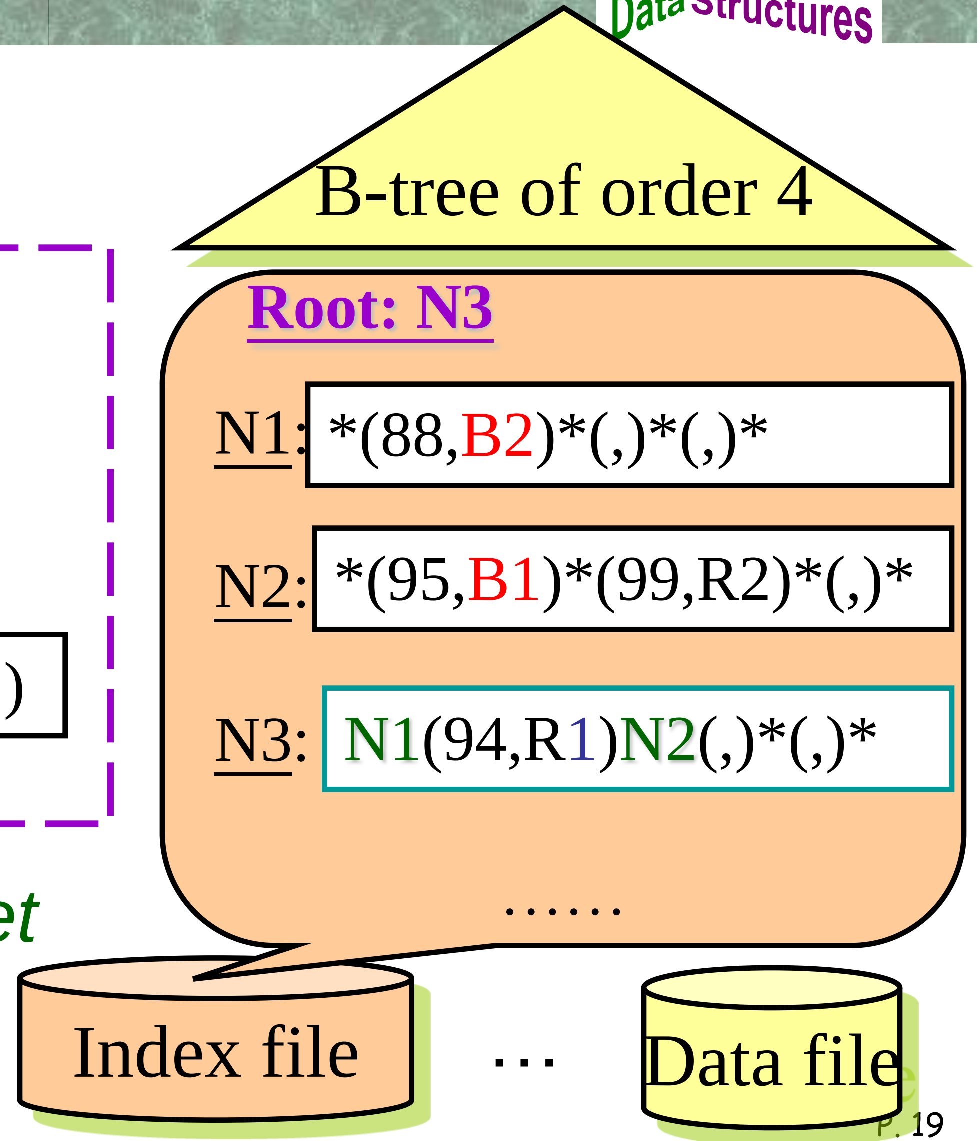
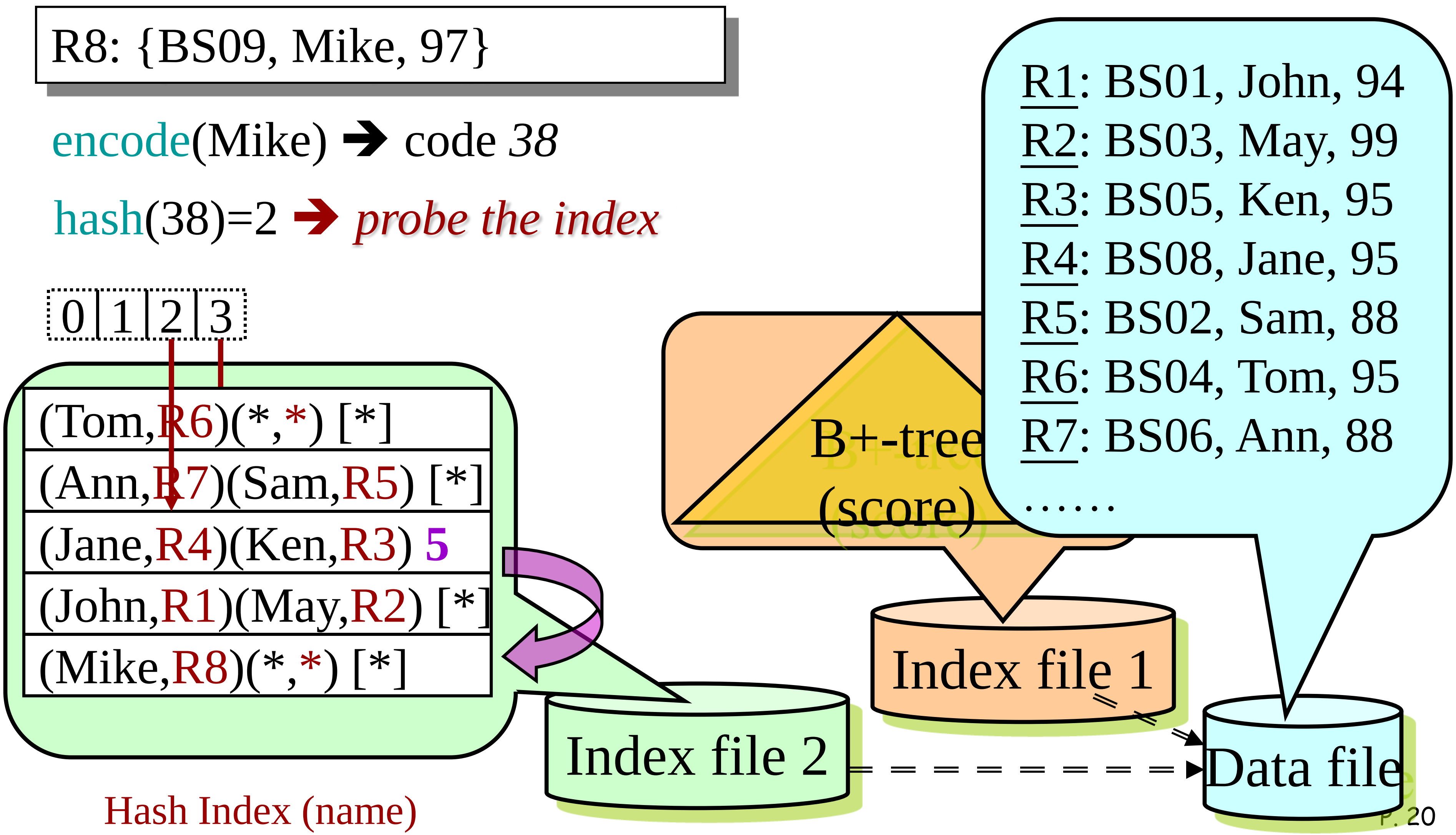
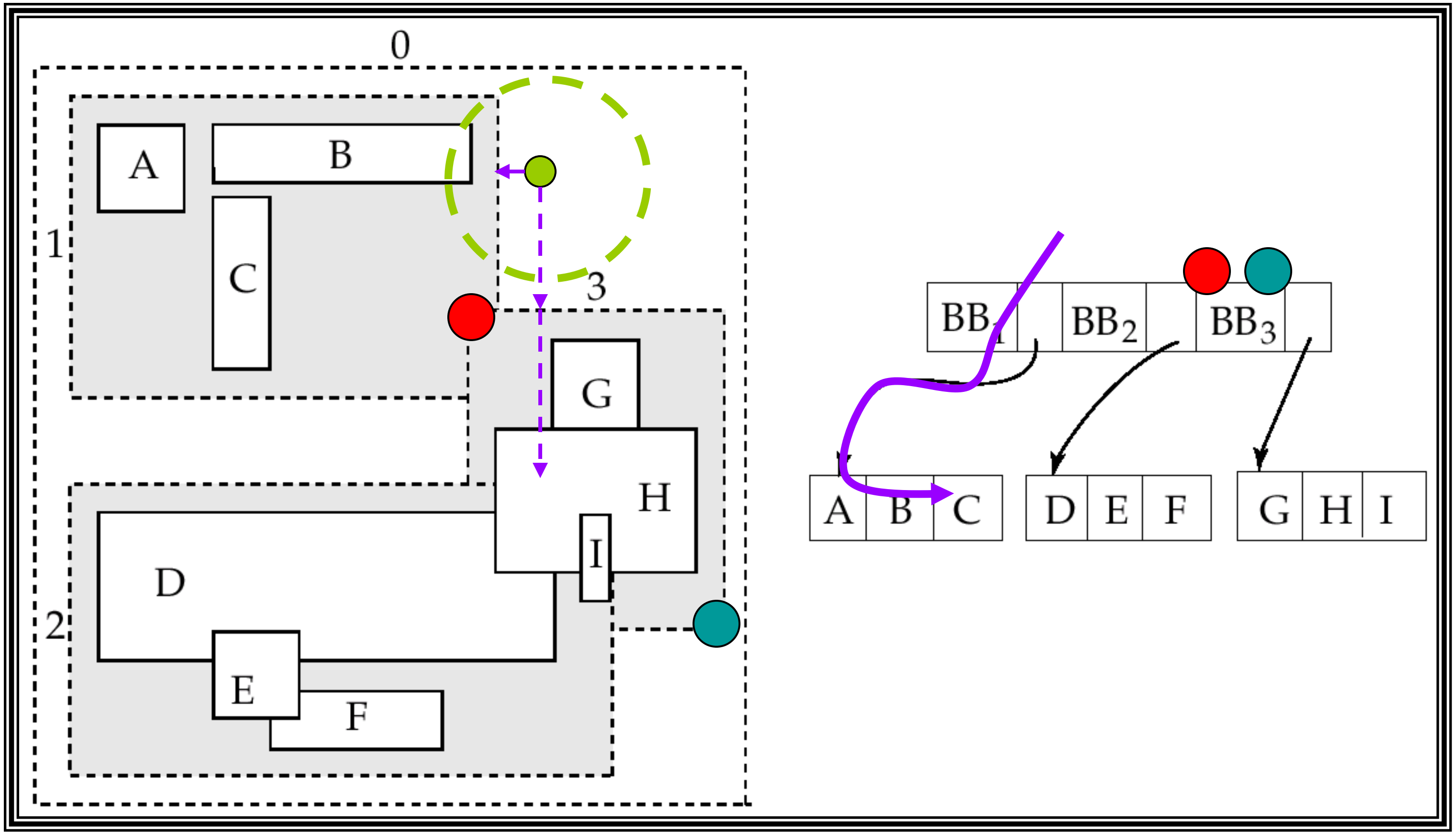


Illustration XII: *Hash File*



Multi-dimensional B+-tree: *R-tree*

Data Structures



Concluding Remarks

1. Recursion
2. Data Abstraction
3. Linked Lists
4. Recursion for Problem Solving
5. Stacks
6. Queues
7. Sorting Algorithms
8. Trees
9. Priority Queues
10. Balanced Search Trees
11. Hashing
12. Graph Basics
13. Graph Apps
14. Secondary Storage

Good Luck!