

Secondary Storage

- ❑ External Sort

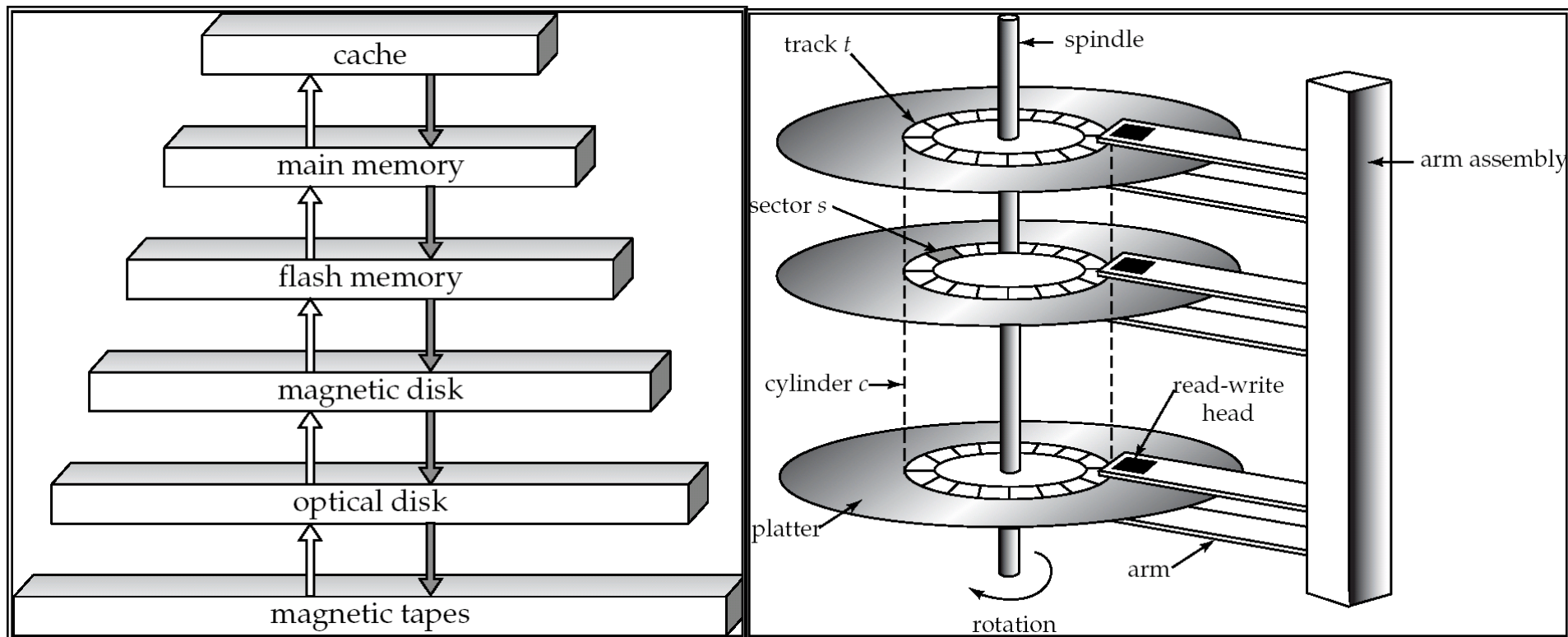
- ❑ B-tree Index

- ❑ Other Indices

Secondary Storage: *Basics*

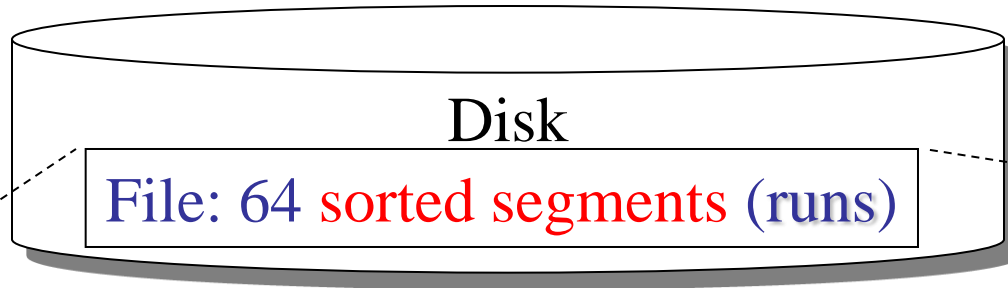
□ Main Memory vs. Secondary Storage

– CPU time vs. I/O time (seek + latency + transfer)



Secondary Storage: *External Sort*

Step 1. Internal sort



Memory space: 10MB

$R_{64,1}, R_{64,2}, \dots, R_{64,100}$

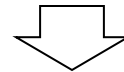
$R_{1,1}, R_{1,2}, \dots, R_{1,100}$

$R_{2,1}, R_{2,2}, \dots, R_{2,100}$

...

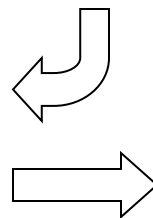
$R_{64,1}, R_{64,2}, \dots, R_{64,100}$

Step 2. Merge sort



Memory space: 40MB

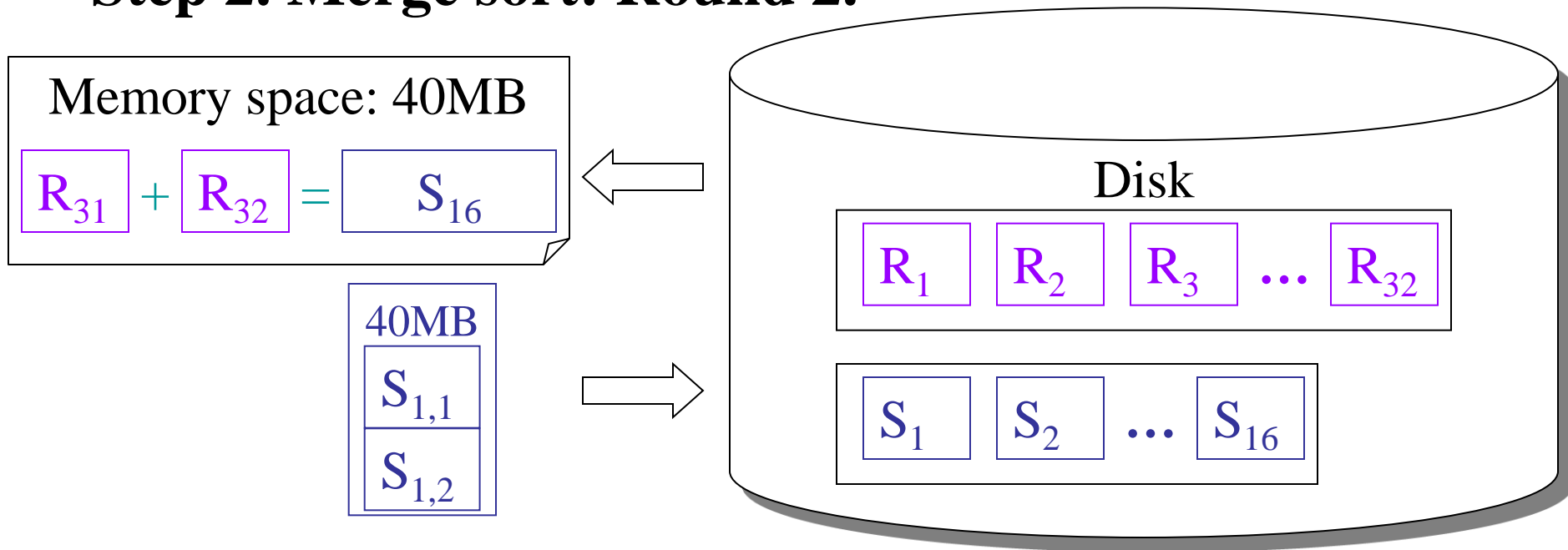
$B_{63} + B_{64} = R_{32}$



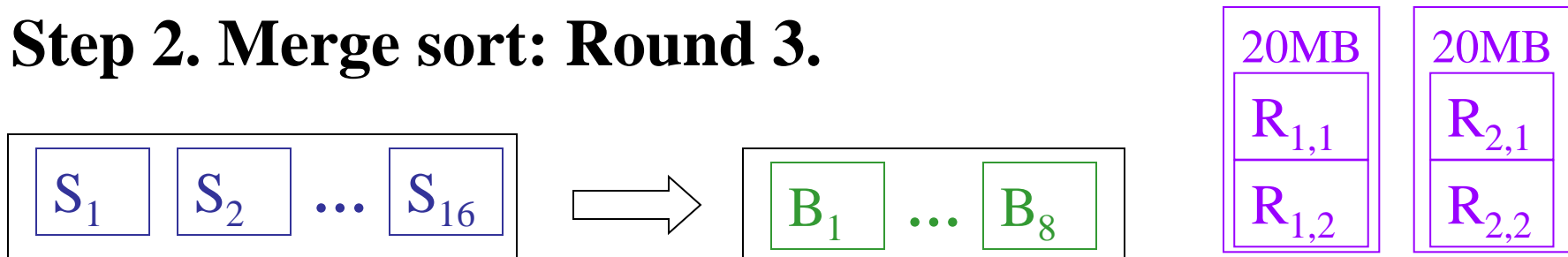
$R_1, R_2, R_3, \dots, R_{32}$

Secondary Storage: *External Sort*

Step 2. Merge sort: Round 2.



Step 2. Merge sort: Round 3.

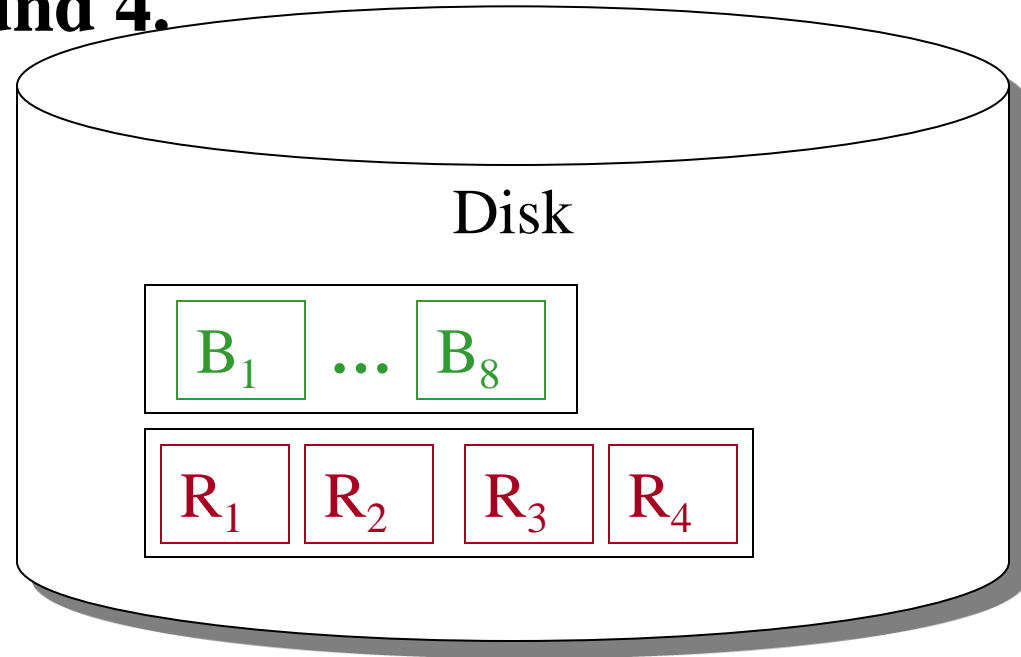
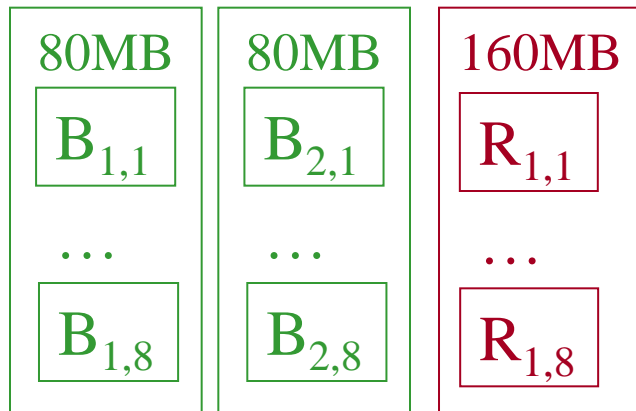


Secondary Storage: *External Sort*

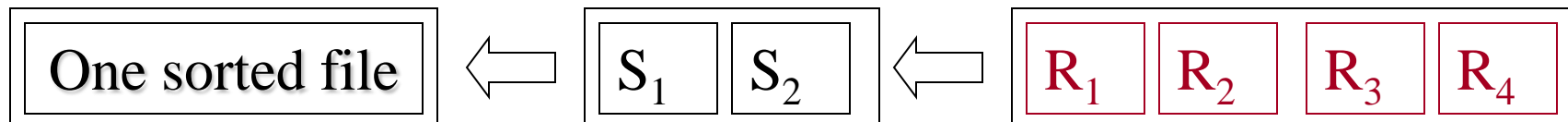
Step 2. Merge sort: Round 4.

Memory space: 40MB

$$B_1 + B_2 = R_1$$

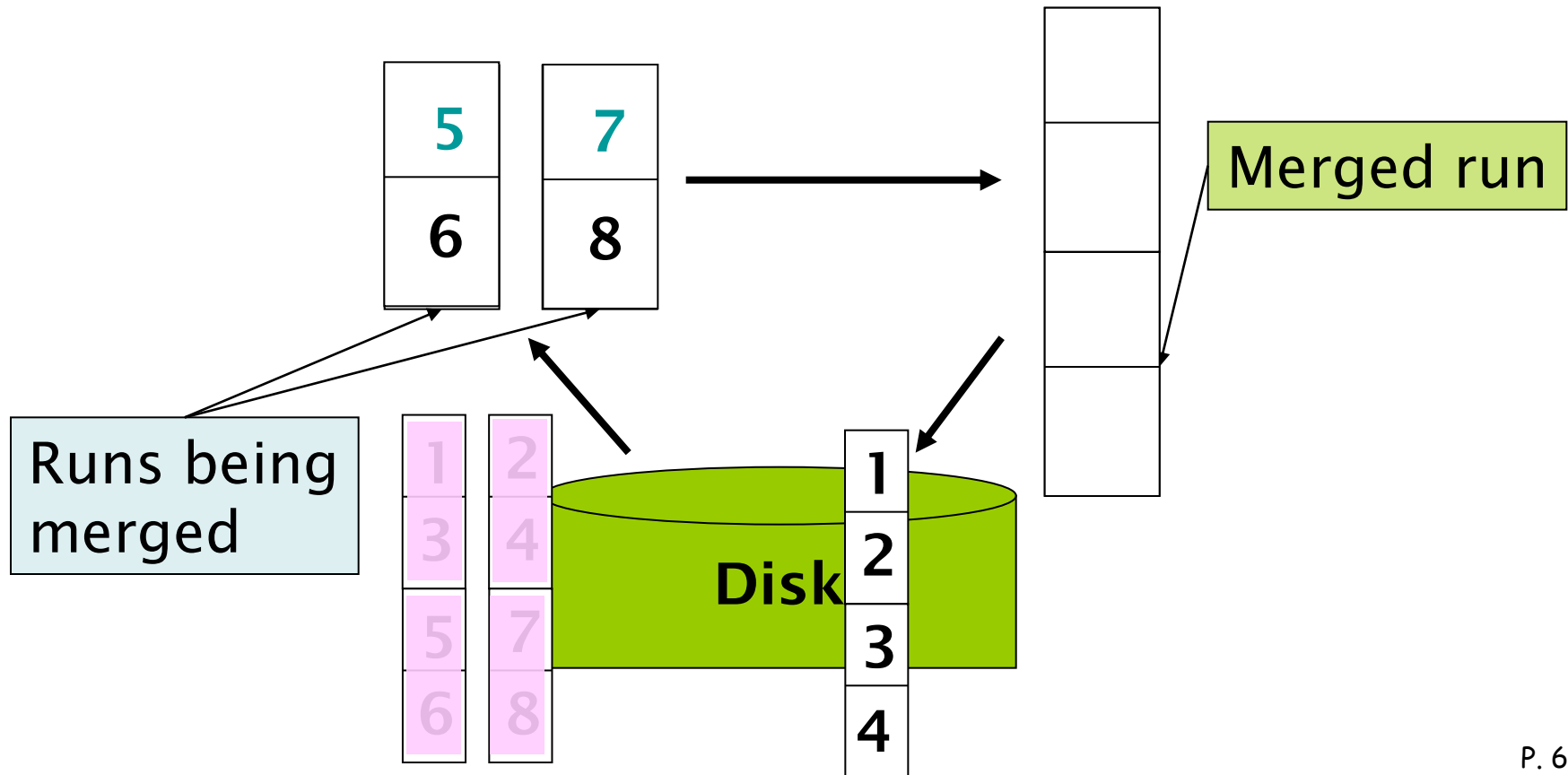


Step 2. Merge sort: Rounds 5 & 6.



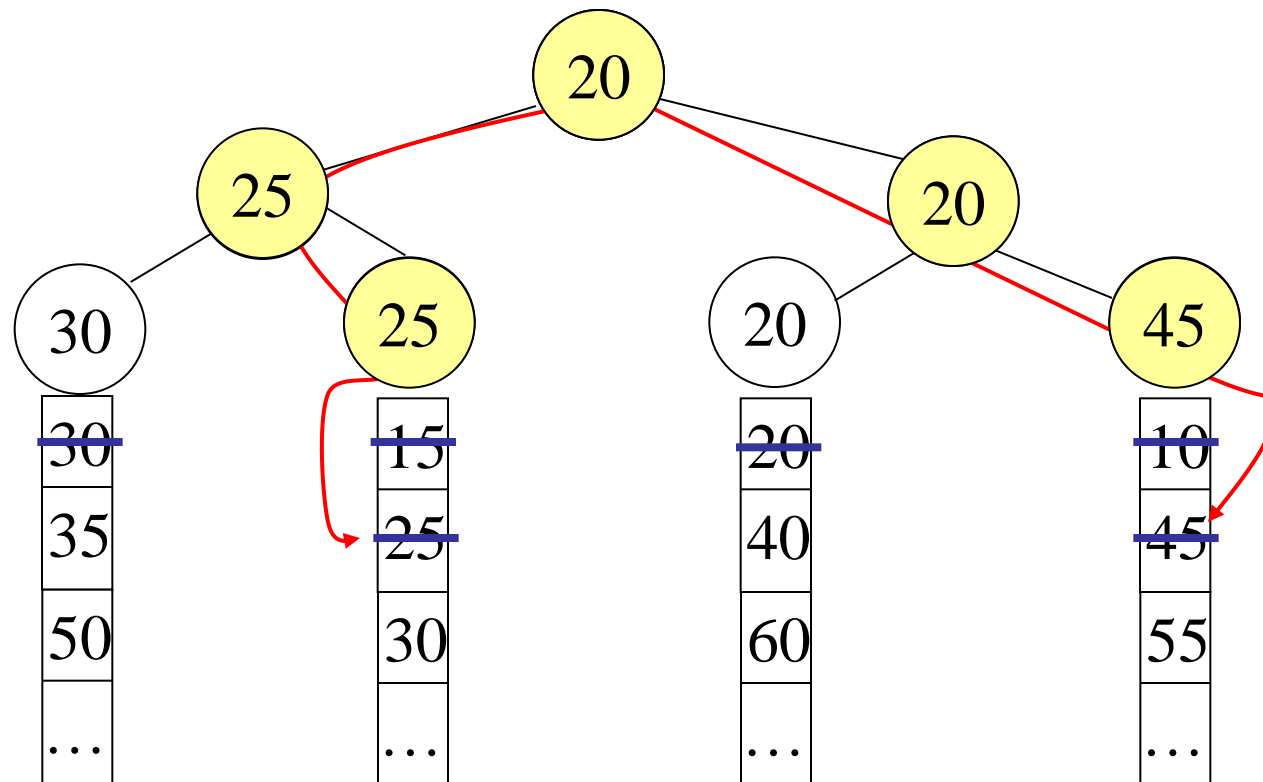
External Sort: 2-way Merge

- ❑ Input buffer
- ❑ Output buffer



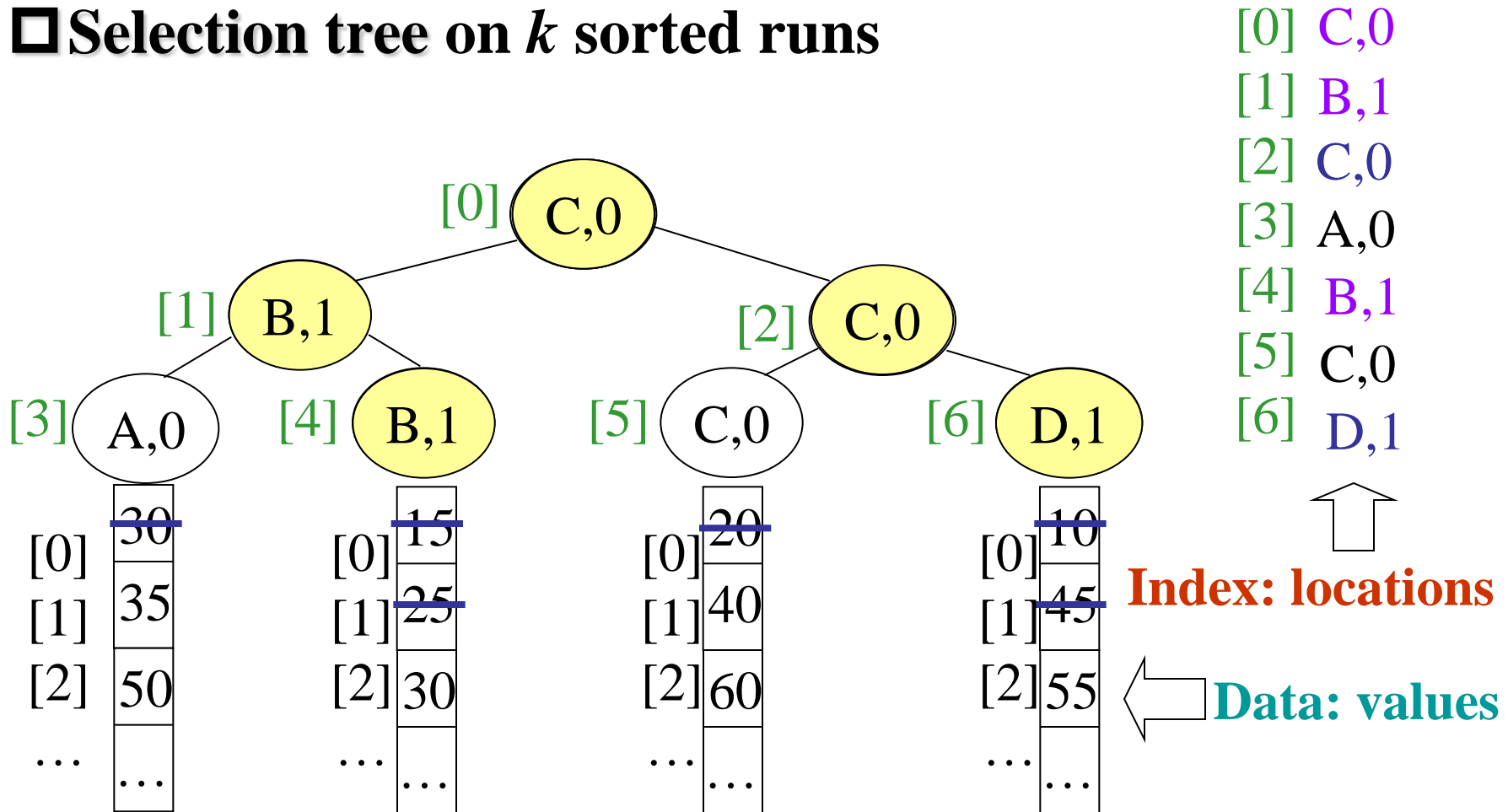
K-way Merge: *Selection Tree*

- 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) + $(i - 1) * (\text{record length})$

$RRN=2$: 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?

| |
|-----------------------------------|
| <u>01</u> <u>27</u> <u>54</u> ... |
|-----------------------------------|

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

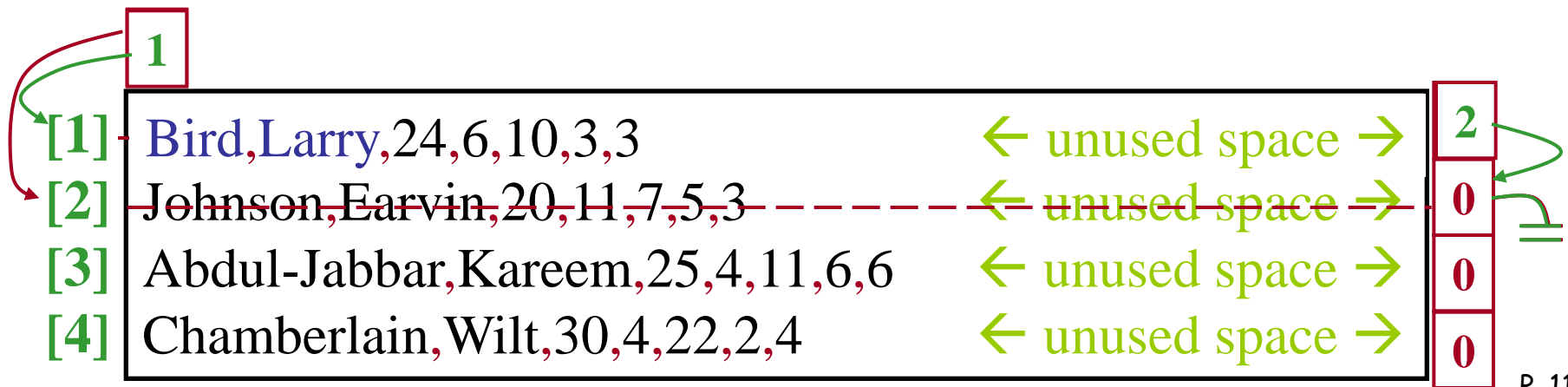


Illustration I: *Key Sort*

(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, ...

record 1: BS001, 99, Lee, ...

record 4: BS005, 99, Liu, ...

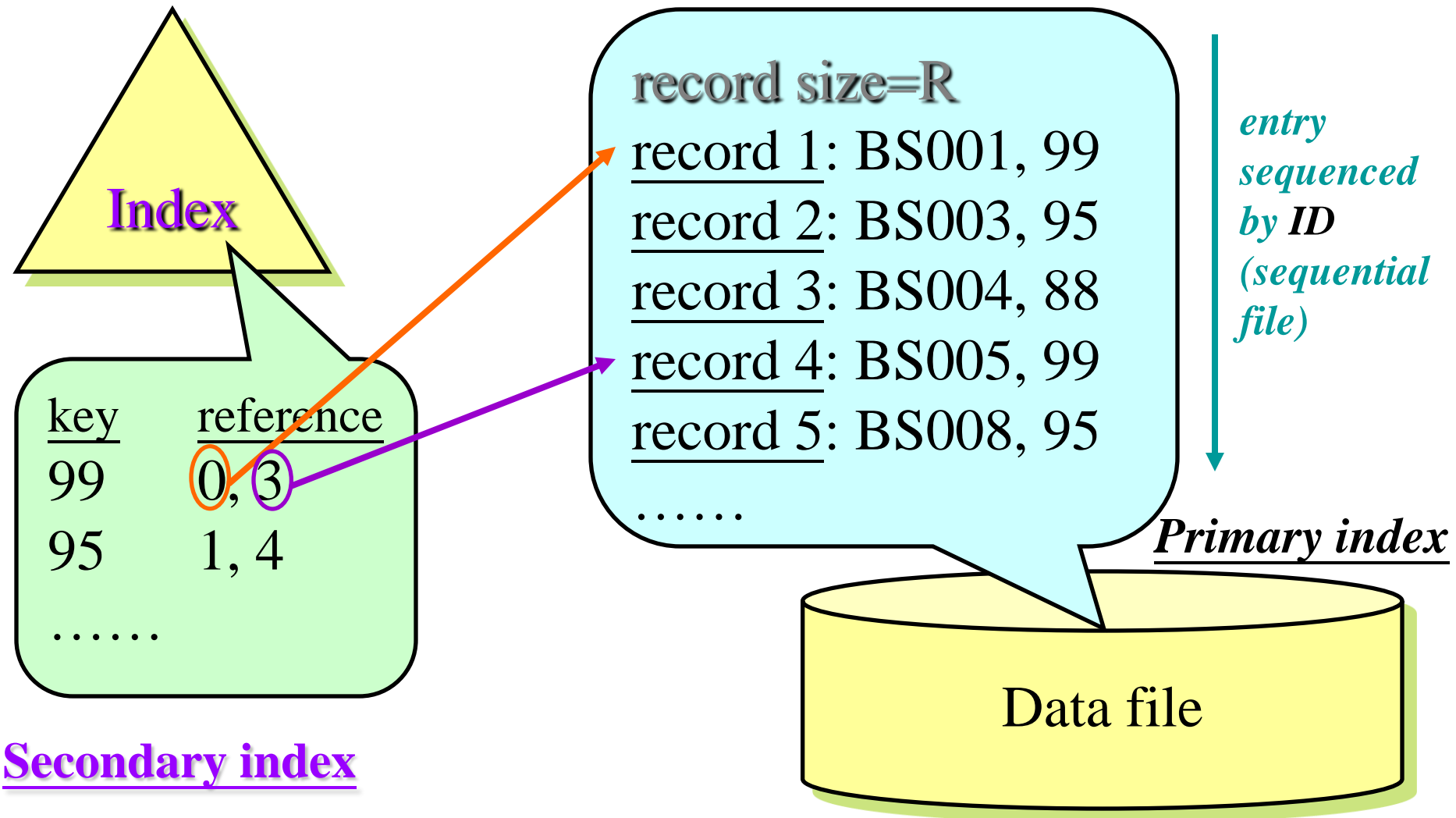
...

record 2: BS003, 95, Lin, ...

IN_FILE

OUT_FILE

Illustration II: *Secondary Index*



B-tree Index: *Examples*

□ B-tree of order 3

$\lceil 3/2 \rceil \sim 3$ children \rightarrow 1~2 keys \rightarrow 2-3 tree

– Insertions: 10, 20, 30, 40, 50, 90, 80, 70, 60, 100

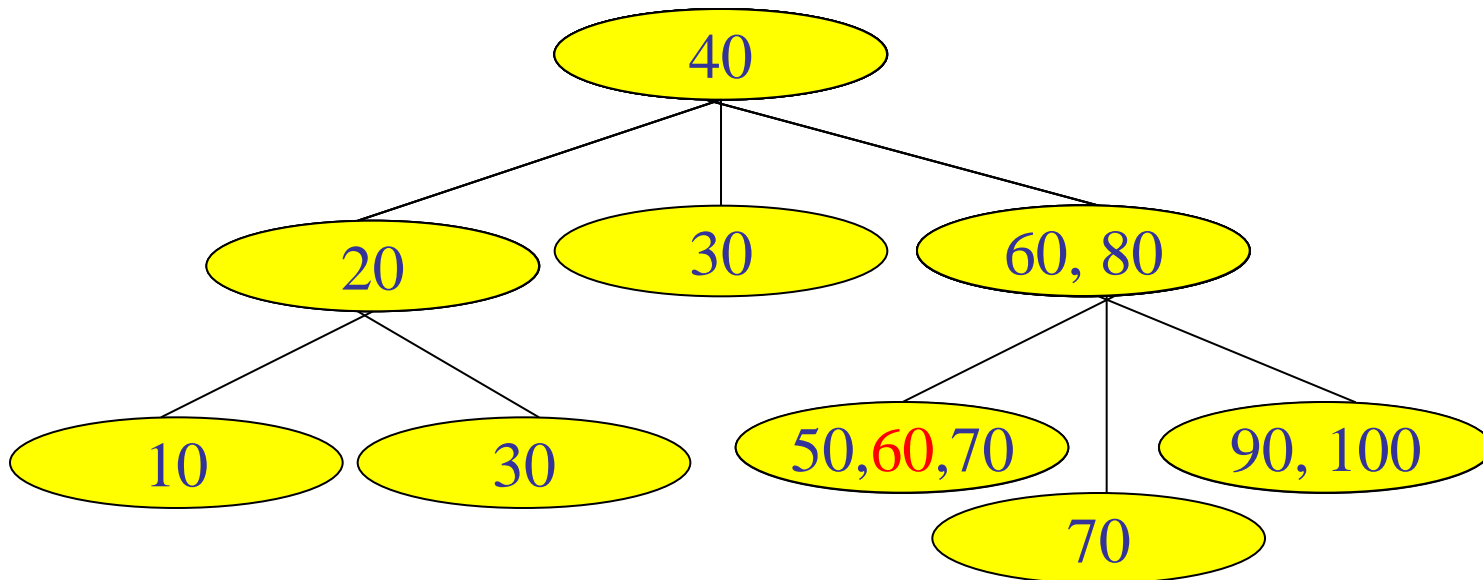


Illustration VI: *B-tree Index*

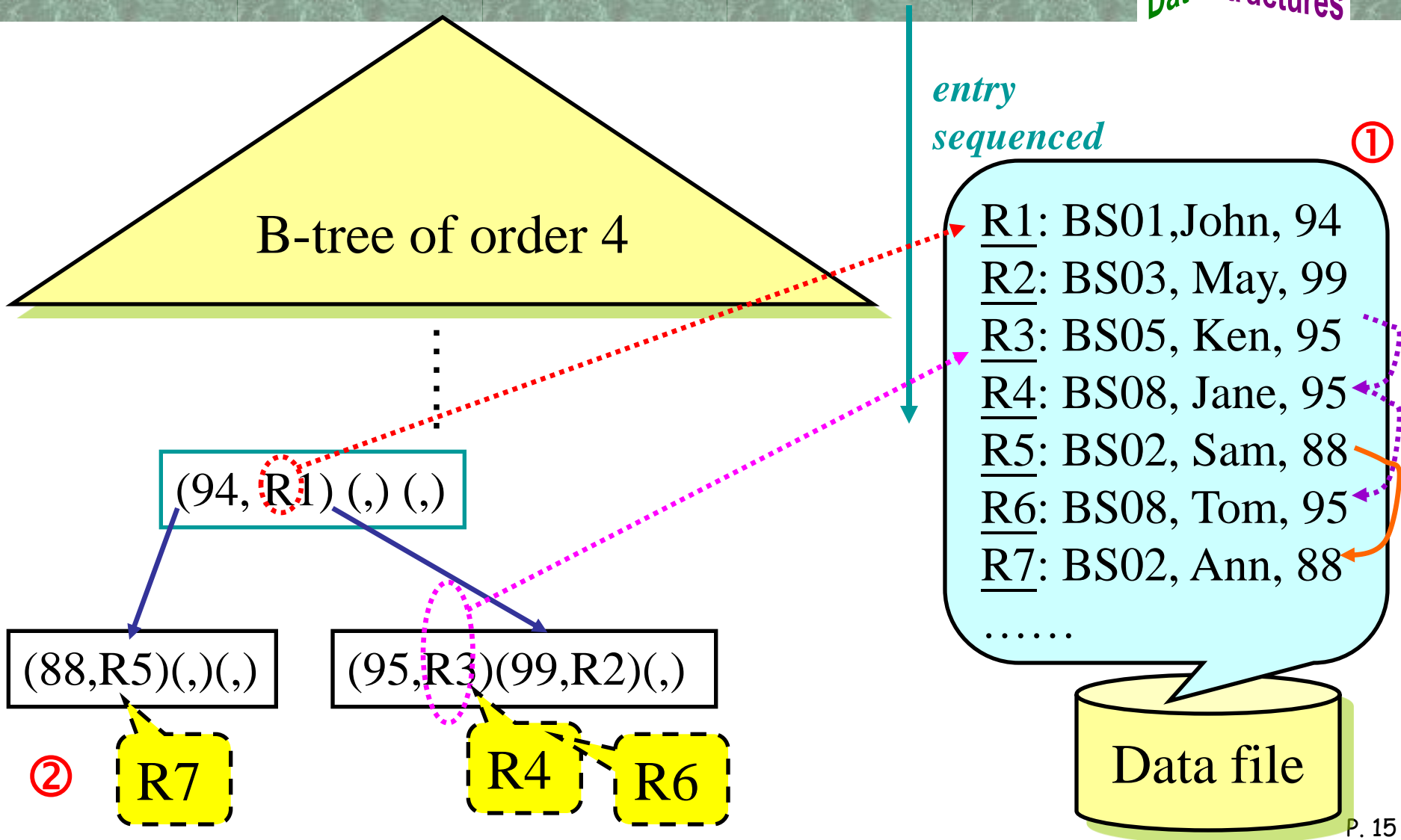


Illustration VII: *B-tree with Buckets*

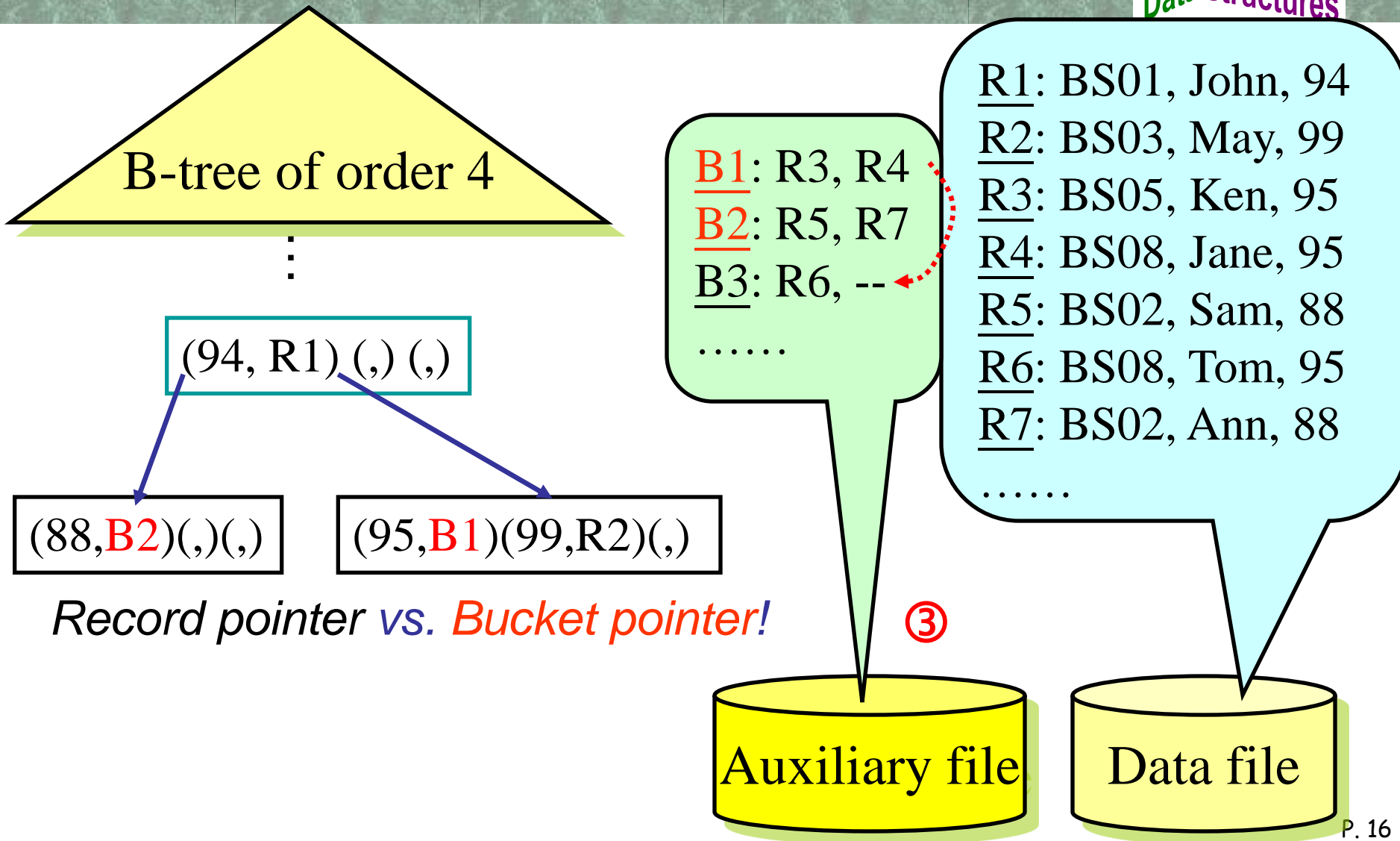


Illustration VIII: *B-tree in File*

B-tree of order 4

⋮

(94, R1) (,) (,)

(88, **B2**) (,) (,)

(95, **B1**) (99, R2) (,)

Index is still in Memory!!

node pointer → file offset

Root: N3

N1: *(88, **B2**)* (,)* (,)*

N2: *(95, **B1**)* (99, R2)* (,)*

N3: N1 (94, R1) N2 (,)* (,)*

.....

B1: R3, R4

B2: R5, R7

6, --

...

Index file

Auxiliary file

Illustration IX: *Reload B-tree*

B-tree of order 4

(94, R1) (,) (,)

(88, **B2**) (,) (,)

(95, **B1**) (99, R2) (,)

node pointer  *file offset*

Root: N3

N1: *(88, B2)* (,)* (,)*

N2: *(95, B1)* (99, R2)* (,)*

N3: **N1**(94, **R1**)**N2**(,)* (,)*

.....

Index file

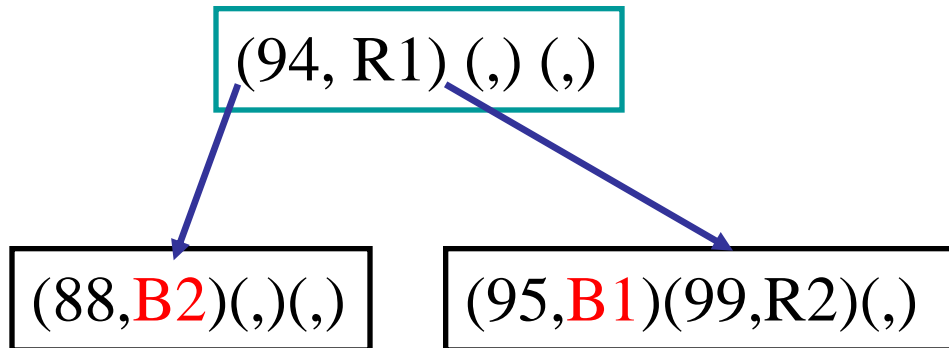
...

Illustration X: *B-tree File*

Data Structures

buffer management

buffer size: 2 nodes



node pointer \Leftrightarrow *file offset*

④

B-tree of order 4

Root: N3

N1: $*(88, \textcolor{red}{B2}) * (,) * (,) *$

N2: $*(95, \textcolor{red}{B1}) * (99, R2) * (,) *$

N3: $\textcolor{green}{N1}(94, \textcolor{blue}{R1}) \textcolor{green}{N2} (,) * (,) *$

.....

Index file

...

Data file

Illustration XII: Hash File

R8: {BS09, Mike, 97}

encode(Mike) → code 38

hash(38)=2 → *probe the index*

0 | 1 | 2 | 3

| | |
|-------------------------------------|----------|
| (Tom, R6)(*,*) | [*] |
| (Ann, R7)(Sam, R5) | [*] |
| (Jane, R4)(Ken, R3) | 5 |
| (John, R1)(May, R2) | [*] |
| (Mike, R8)(*,*) | [*] |

Hash Index (name)

B+-tree
(score)

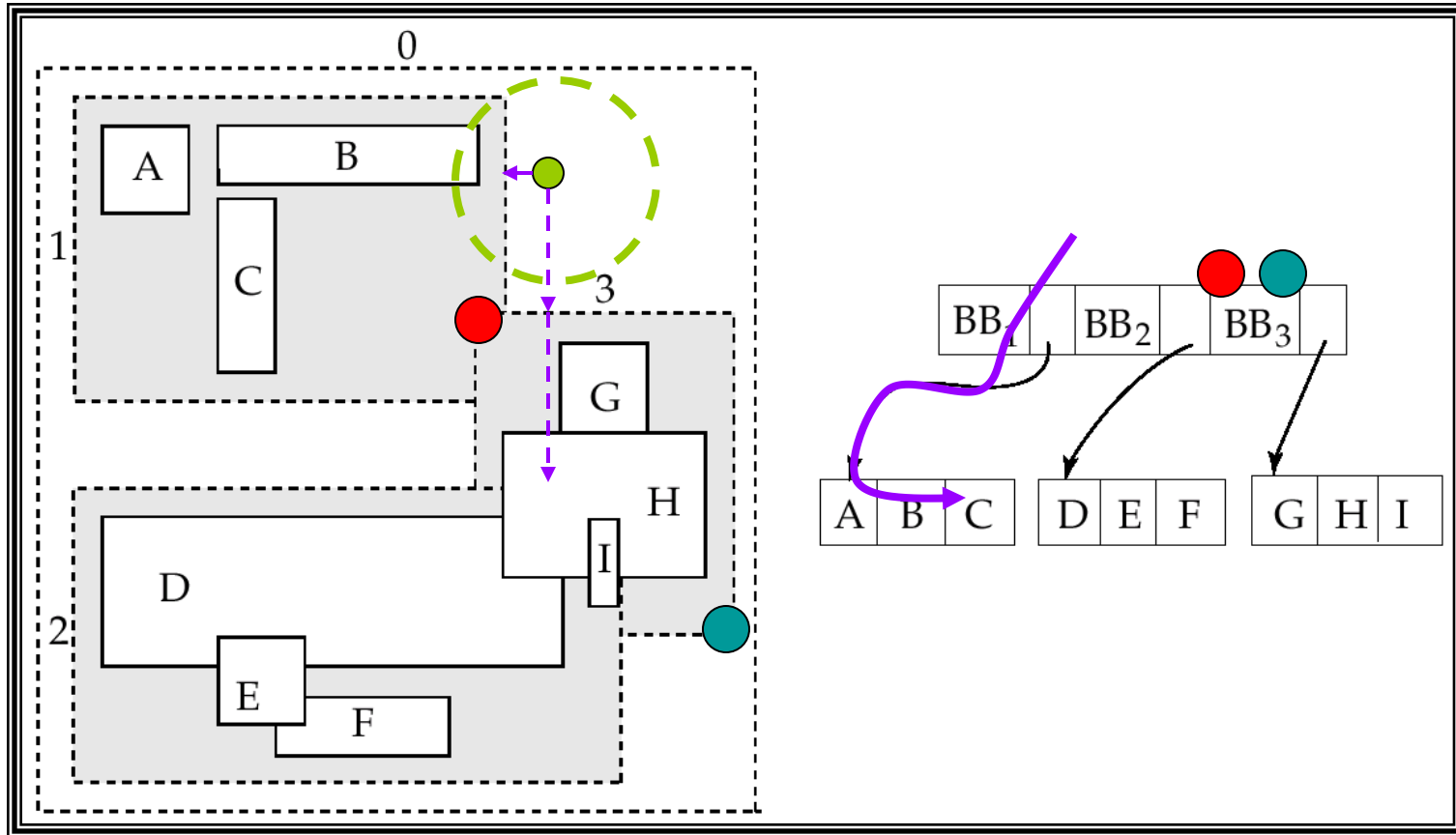
R1: BS01, John, 94
R2: BS03, May, 99
R3: BS05, Ken, 95
R4: BS08, Jane, 95
R5: BS02, Sam, 88
R6: BS04, Tom, 95
R7: BS06, Ann, 88
.....

Index file 1

Index file 2

Data file

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



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!