Secondary Storage

- External Sort
- B-tree Index
- Other Indices

Data Structures

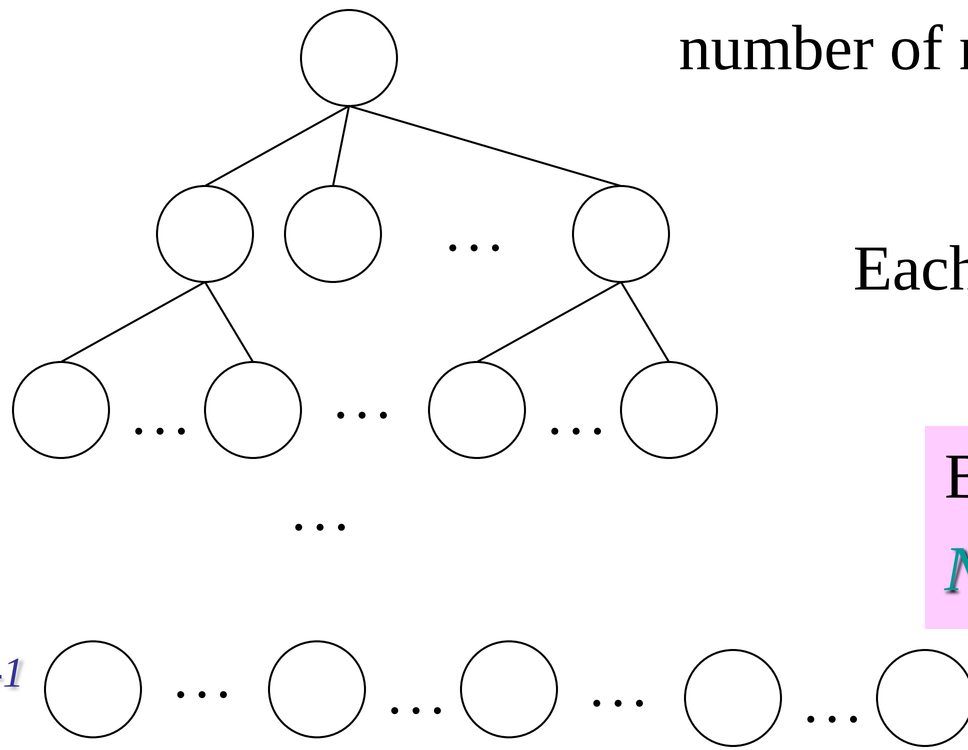
B-tree Index

- **Balanced Search Tree**
  - If the entire tree fits the memory, no file is needed
  - Otherwise, number of node accesses ≅ tree height
  - $O(\log_2 n) > O(\log_m n)$ *for* $m > 2$
- **Balanced *m*-way search tree = B-tree of order *m***
  - A generalization of 2-3 trees and 2-3-4 trees
  - Order m: *maximum number of children* (m-1 keys)
  - Given the order m and tree height h, the number of keys N in the B-tree ≤ $m^h - 1$

B-tree Index

- **Balanced *m*-way search tree = B-tree of order *m***
  - Given the order m and tree height h, the number of keys N in the B-tree $\leq m^h\text{-}1$

level	
1	1
2	m
3	m^2
h	m^{h-1}



number of nodes $\leq \Sigma m^i = (m^h-1) / (m-1)$

for $i=0$ to $h-1$

Each node has at most $m-1$ keys

So, $N \leq m^h\text{-}1$

B-tree of order 40, height 3

$N \leq 40^3-1 = 64{,}000\text{-}1$ keys

B-tree Index

1. **Given N keys and B-tree of order m**

   $N \leq m^h - 1 \Rightarrow \log_m(N+1) \leq h$ … tree height

   – The minimum number of node accesses

   – Expected *best-case* performance under this setting

2. **Given N keys and tree height h**

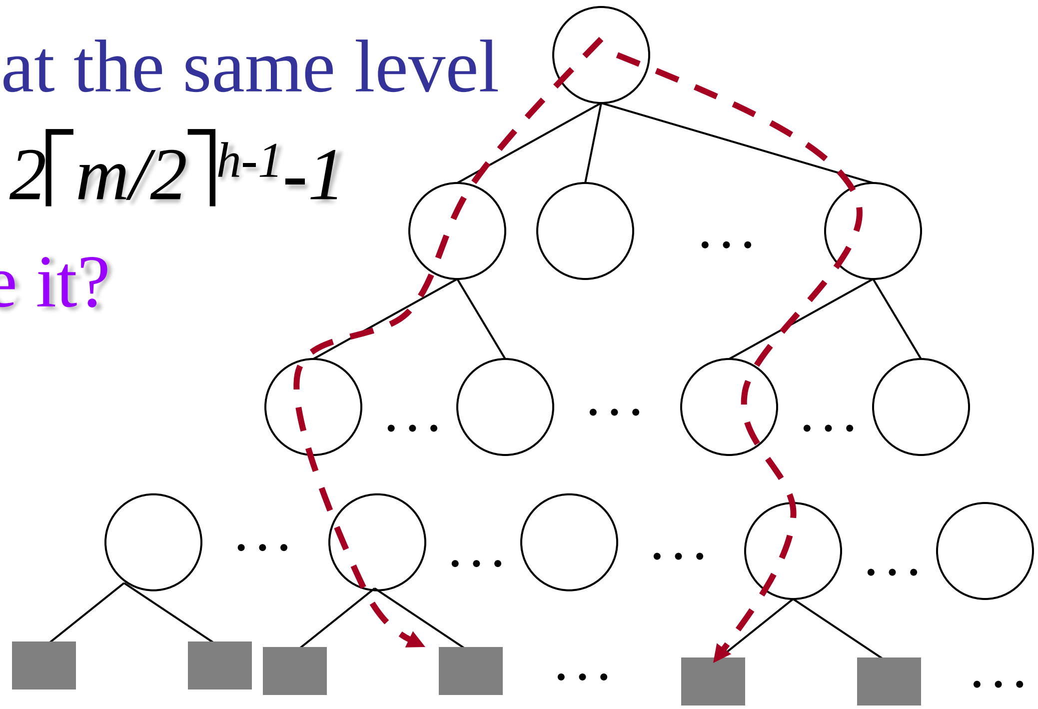
   $N \leq m^h - 1 \Rightarrow (N+1)^{1/h} \leq m$ … the order

   – The *required* setting to achieve the performance

B-tree Index

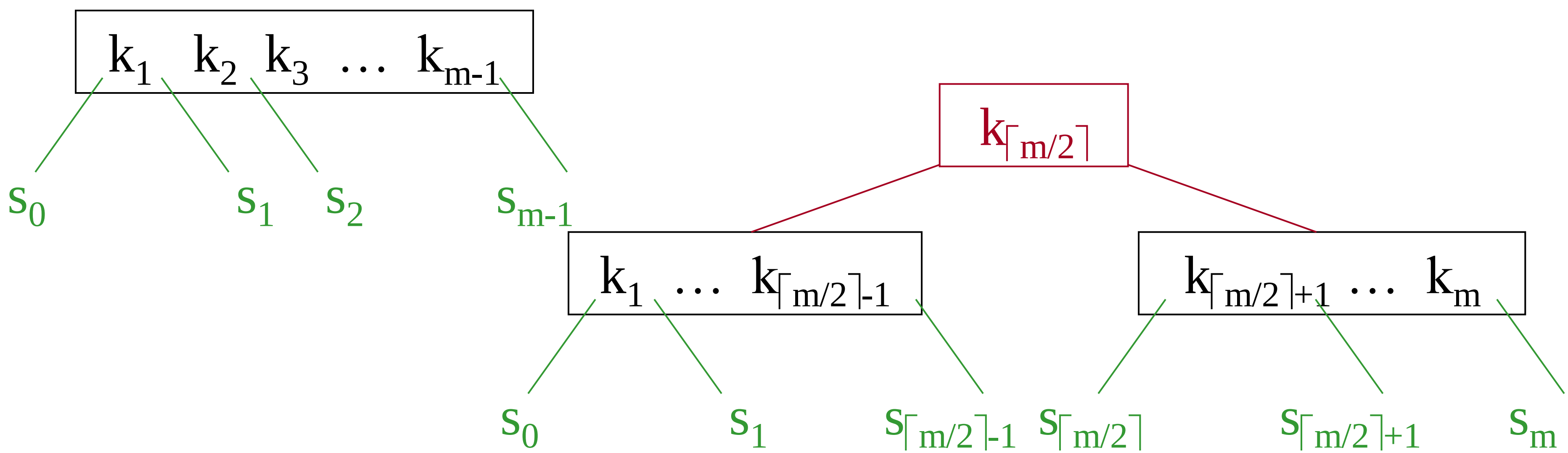
- **Definition**
  - Root has 2~m children
  - The other node has $\lceil m/2 \rceil \sim m$ children
  - Failure nodes are at the same level
- **Given m and h, $N \geq 2\lceil m/2 \rceil^{h-1}-1$**

  Q&A: Can you derive it?



B-tree Index: *Insertion*

- **Similar to 2-3 tree, instead of 2-3-4 tree**
  - Split when the node to insert is full (*m-1* keys)
  - Among the *m* keys (*sorted*), move the $\lceil m/2 \rceil$-th key to the parent node (upward recursion)



m=6: 5 keys + 1 key → $\lceil m/2 \rceil = 3$

$K_1 \; K_2 \; K_3 \; K_4 \; K_5 \; K_6$

B-tree Index: *Examples*

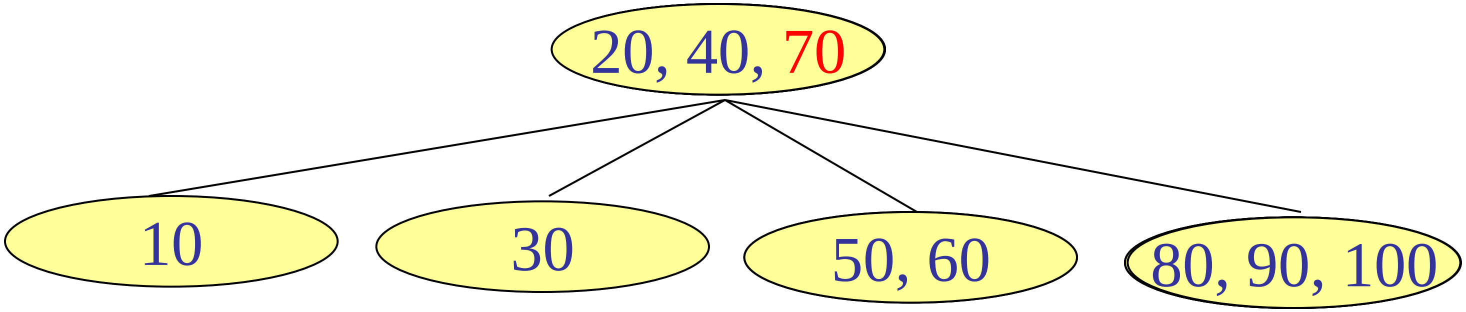
Data Structures

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

B-tree Index: *Examples*

Data Structures

- **B-tree of order 4**
  - ⌈4/2⌉ ~ 4 children ➔ 2~3 keys
  - – Insertions: 10, 20, 30, 40, 50, 90, 80, 70, 60, 100, 110



B-tree Index: *Deletion*

- **B-tree of order 3**

  $\lceil 3/2 \rceil \sim 3$ children ➔ 1~2 keys

  +55, -90, -70, -100, -40, -80

Index vs. Data

- **Insertion**
  1. Add the data record → get the *location* in file
  2. Add the index entry
- **Deletion**
  1. Remove the index entry → get the *location* in file
  2. Remove the data record

Illustration VI: *B-tree Index*

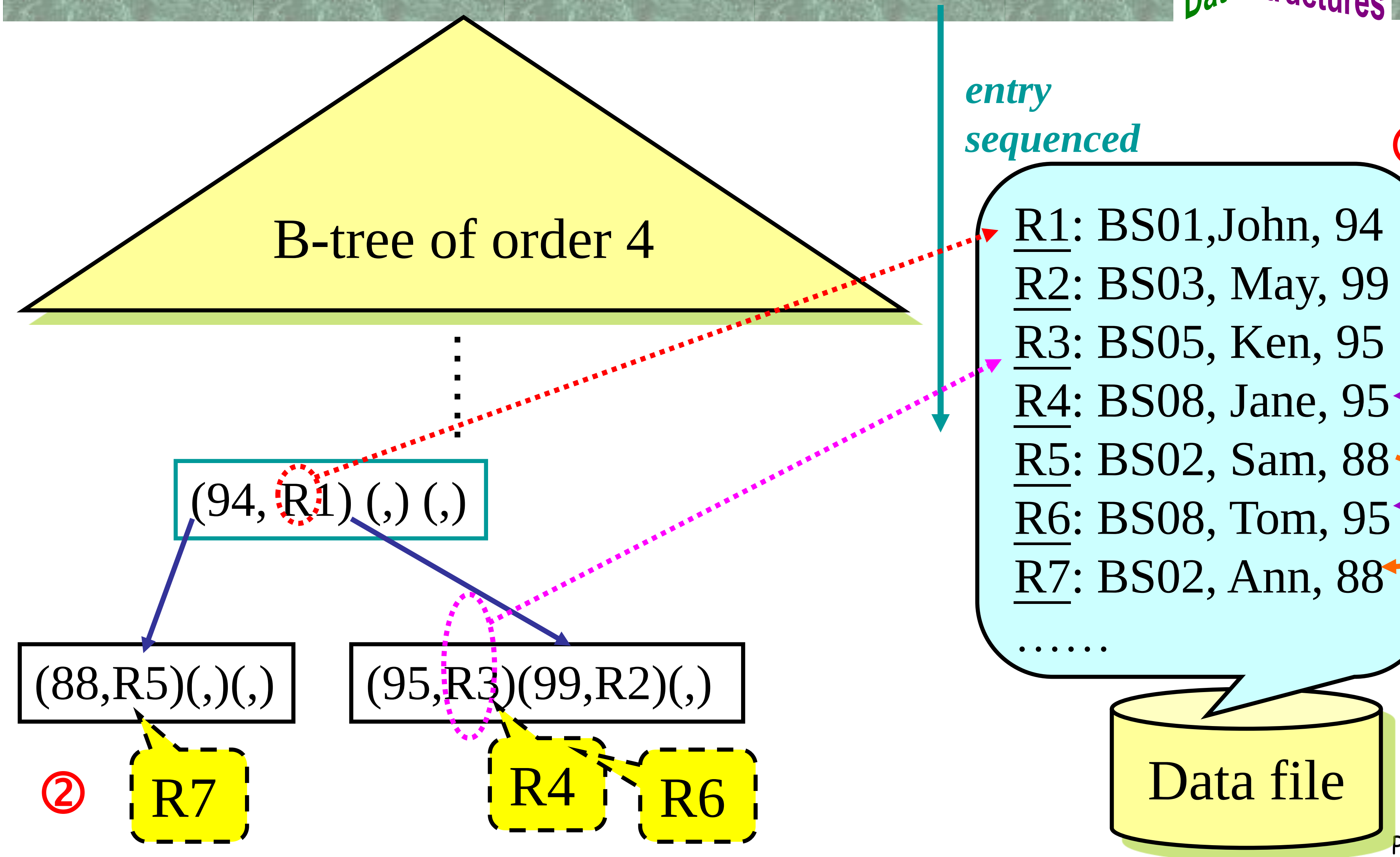


Illustration VII: *B-tree with Buckets*

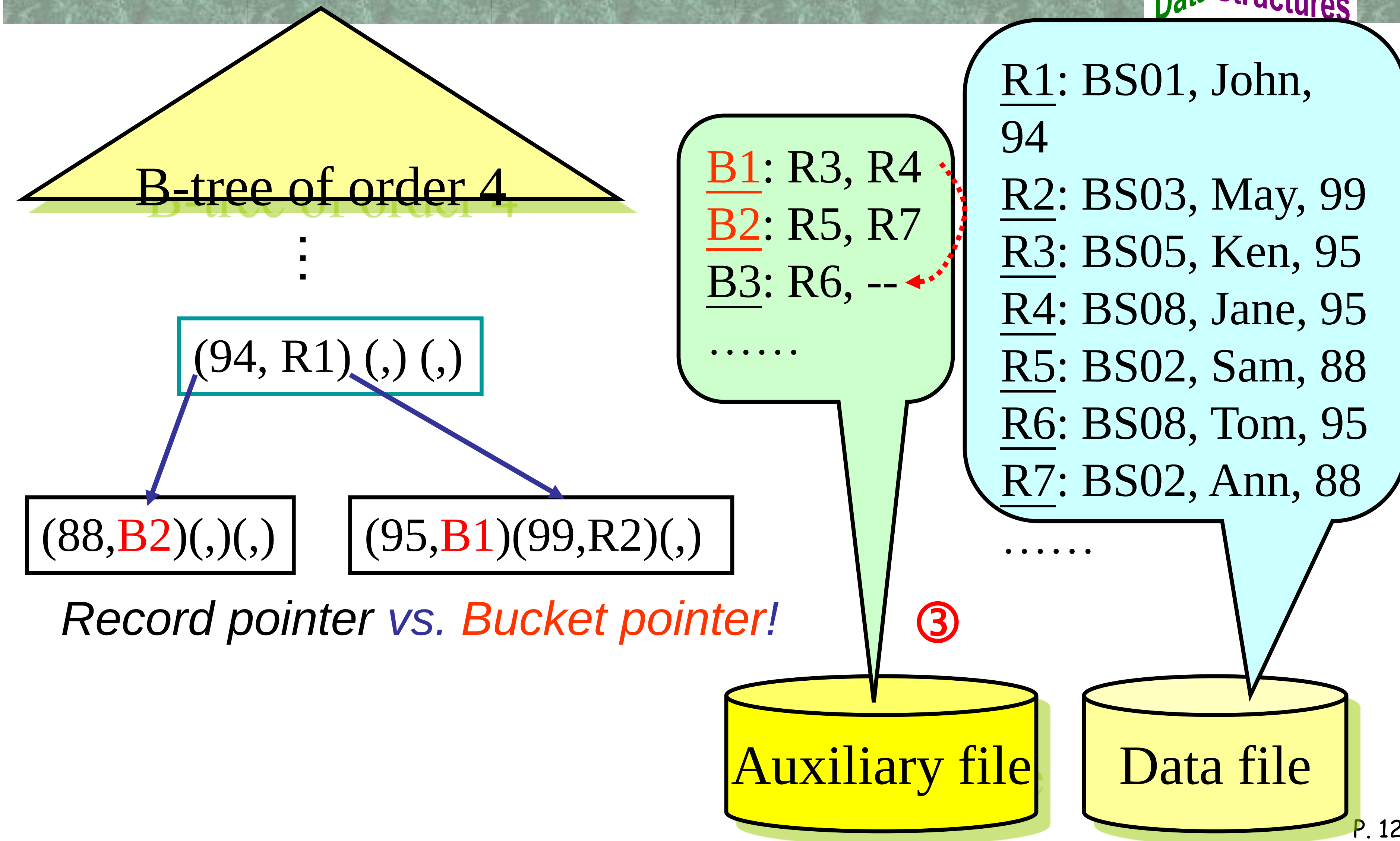


Illustration IX: *Reload B-tree*

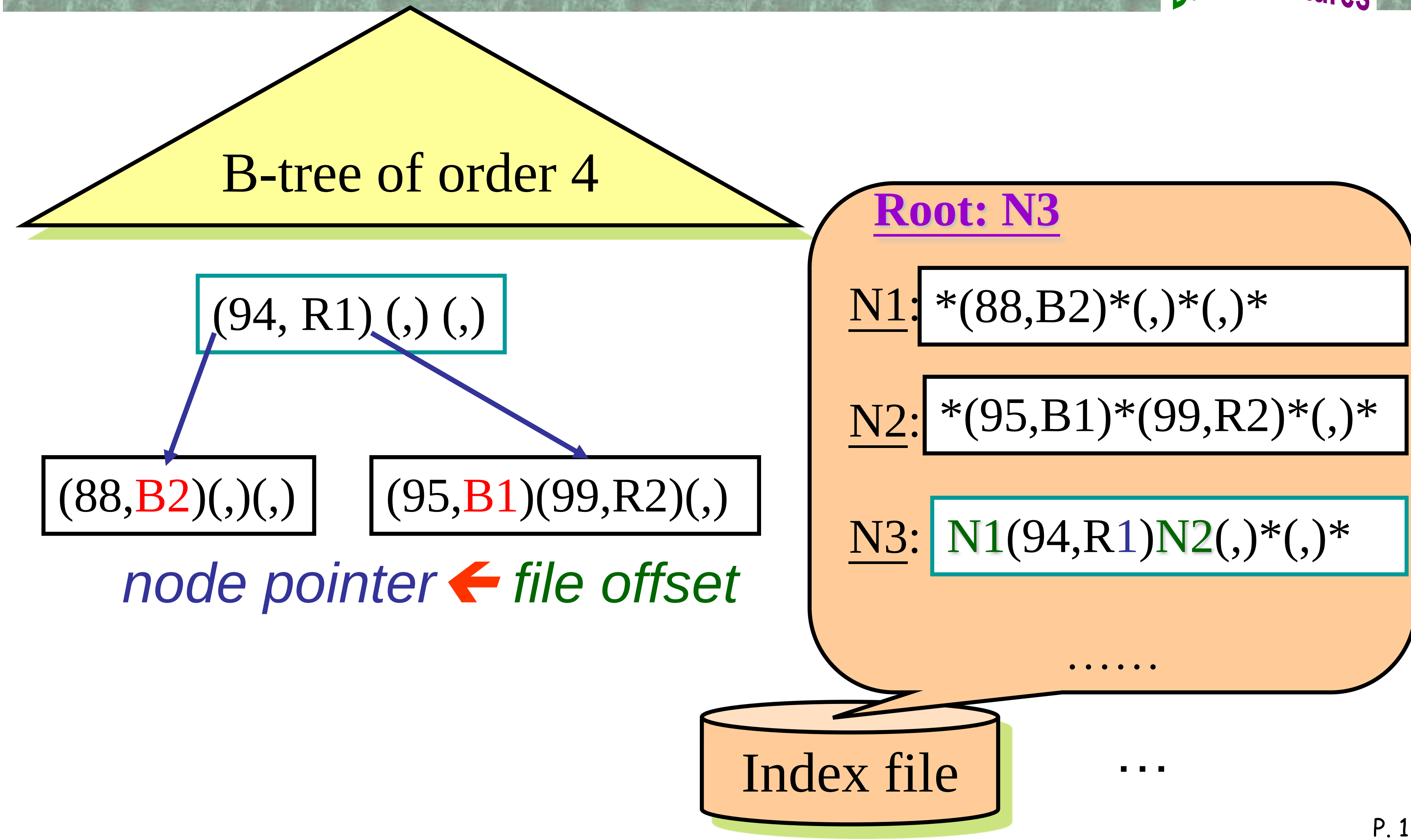
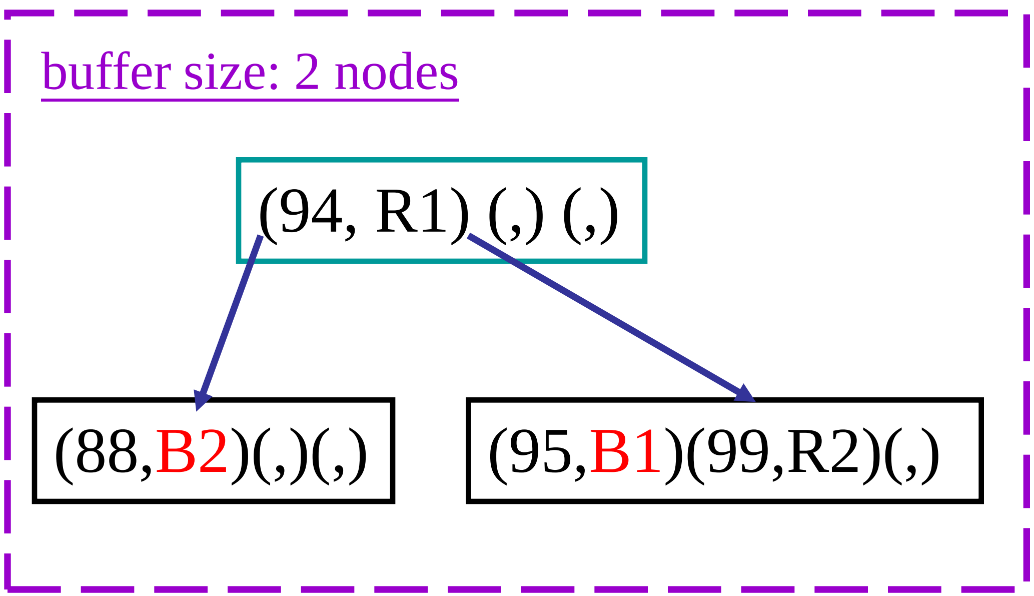
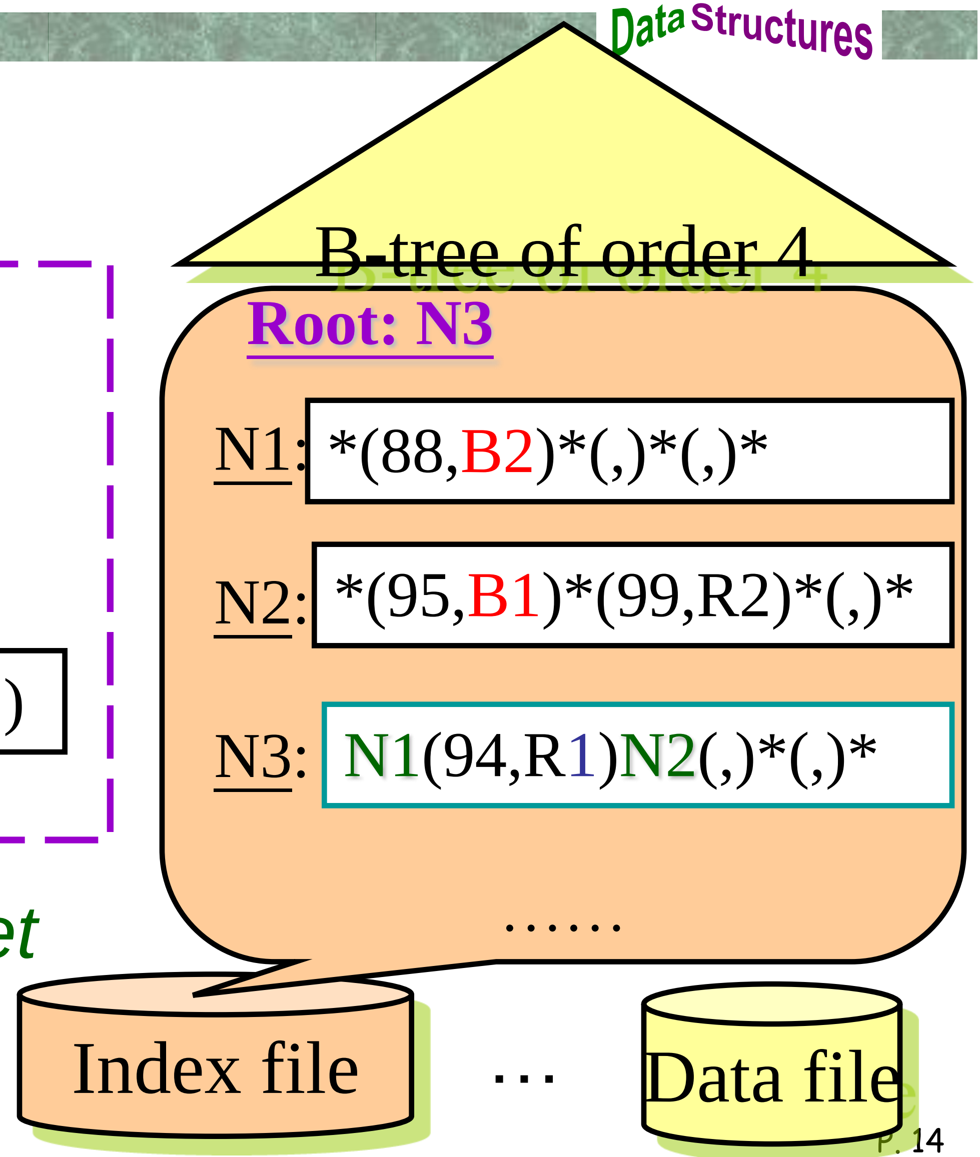


Illustration X: *B-tree File*

buffer management



node pointer ⇔ file offset



Variations of B-tree

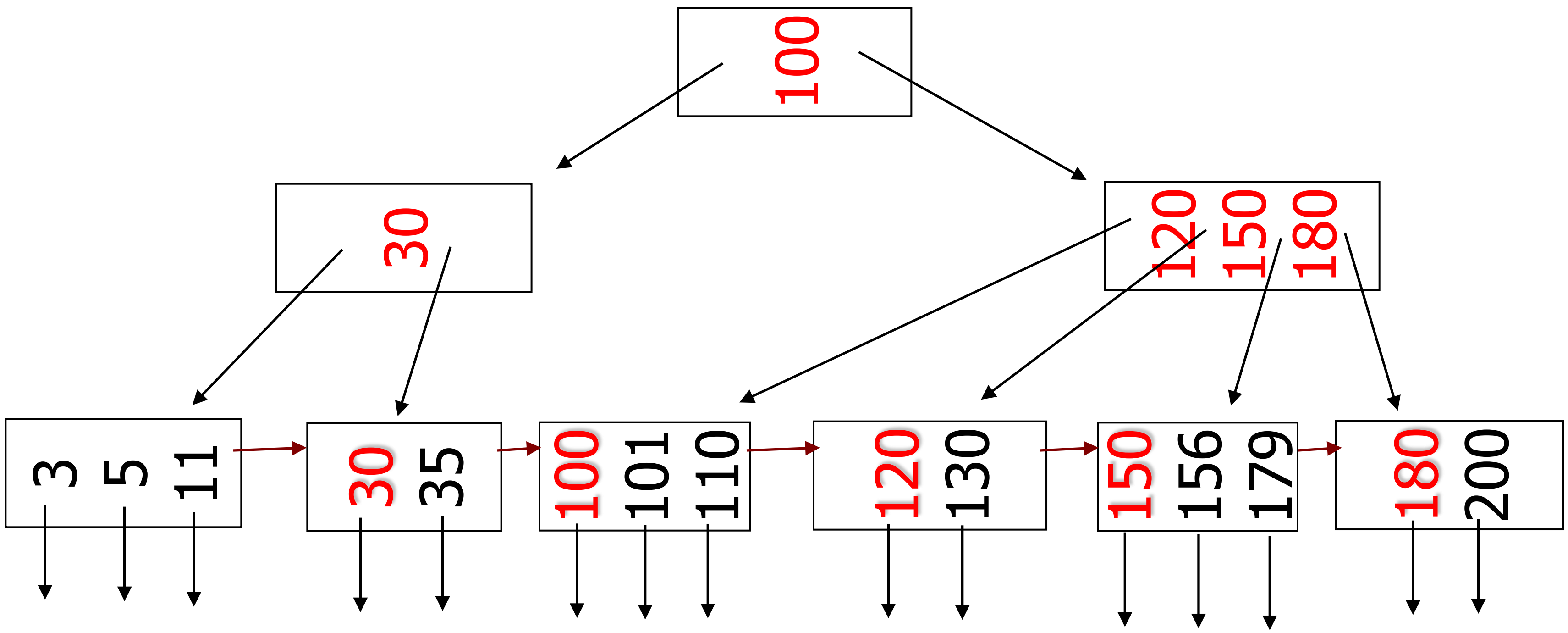
- **B*-tree:** ***delayed split + better space utilization***
  - Root has 2~m children
  - The other node has $\lceil (2m-1)/3 \rceil$ ~ m children
  - Failure nodes are at the same level

    m=6: $\lceil (2m-1)/3 \rceil = 4 \rightarrow$ 3~5 keys

    K_1 K_2 K_3 K_4 K_5

- **B+-tree:** ***fixed-size node + range query***
  - Root has 2~m children
  - The other non-leaf node has $\lceil m/2 \rceil$ ~ m children
  - The leaf has $\lceil (m-1)/2 \rceil$ ~ m-1 keys
  - Failure nodes are at the same level

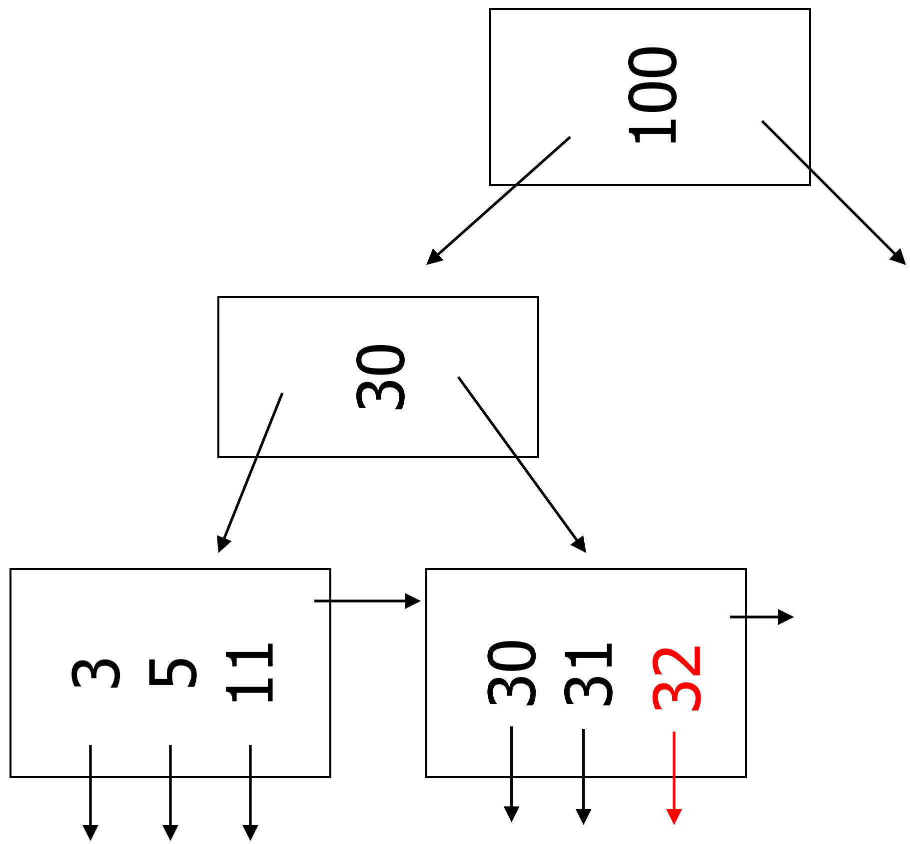
B+-tree Index: *Example of m=4*



B+-tree Index: *Insertion for m=4*

Data Structures

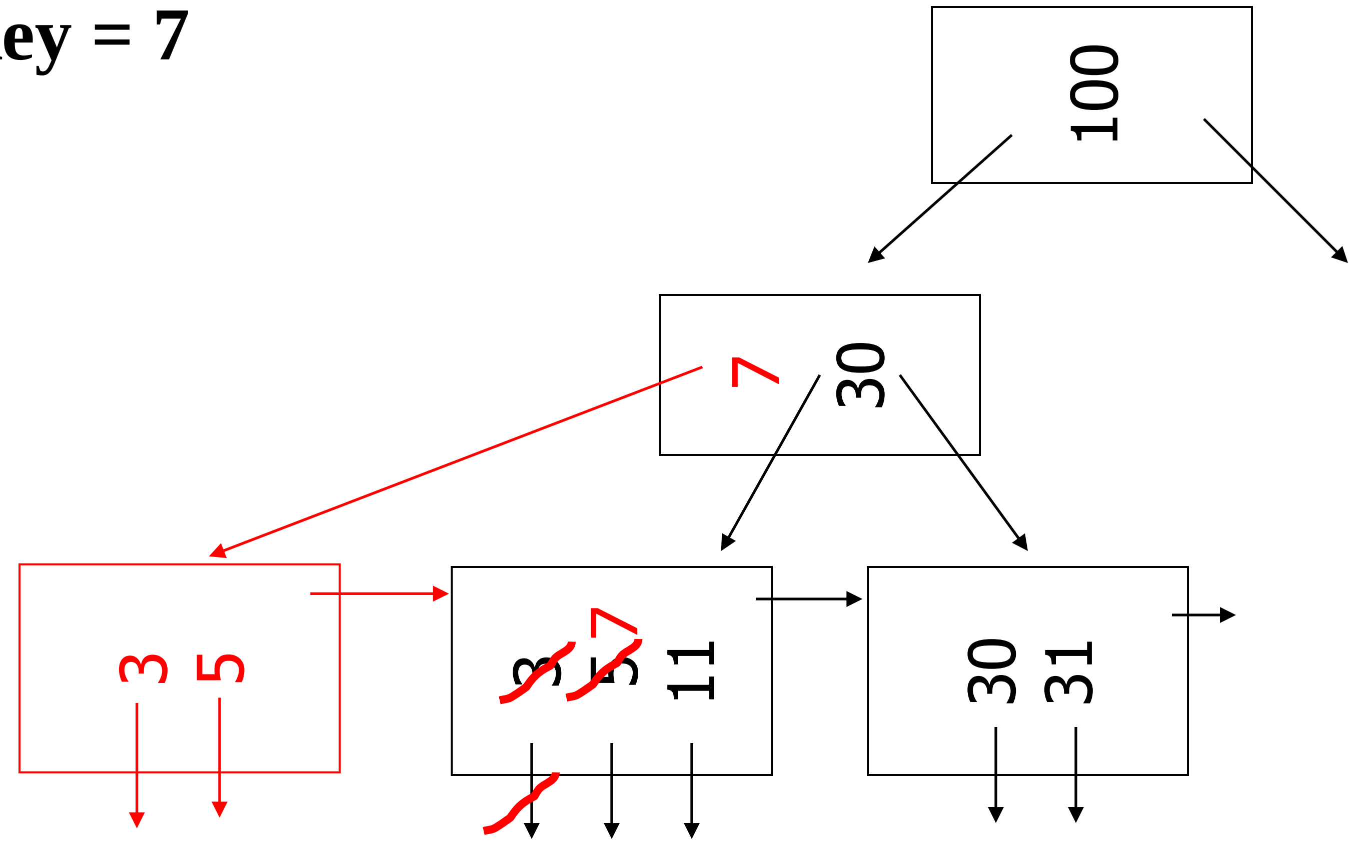
(a) Insert key = 32



B+-tree Index: *Insertion for m=4*

Data Structures

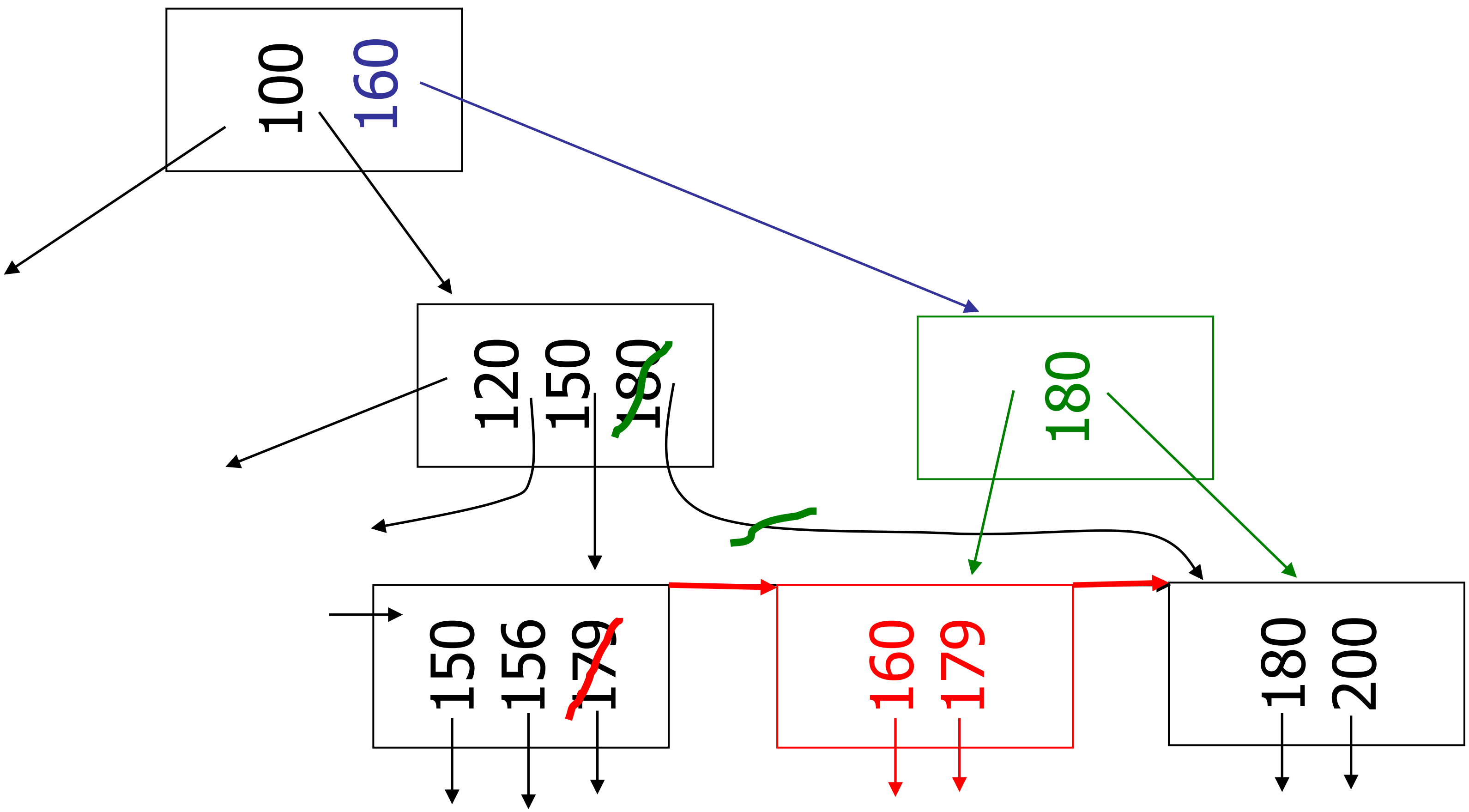
(b) Insert key = 7



B+-tree Index: *Insertion for m=4*

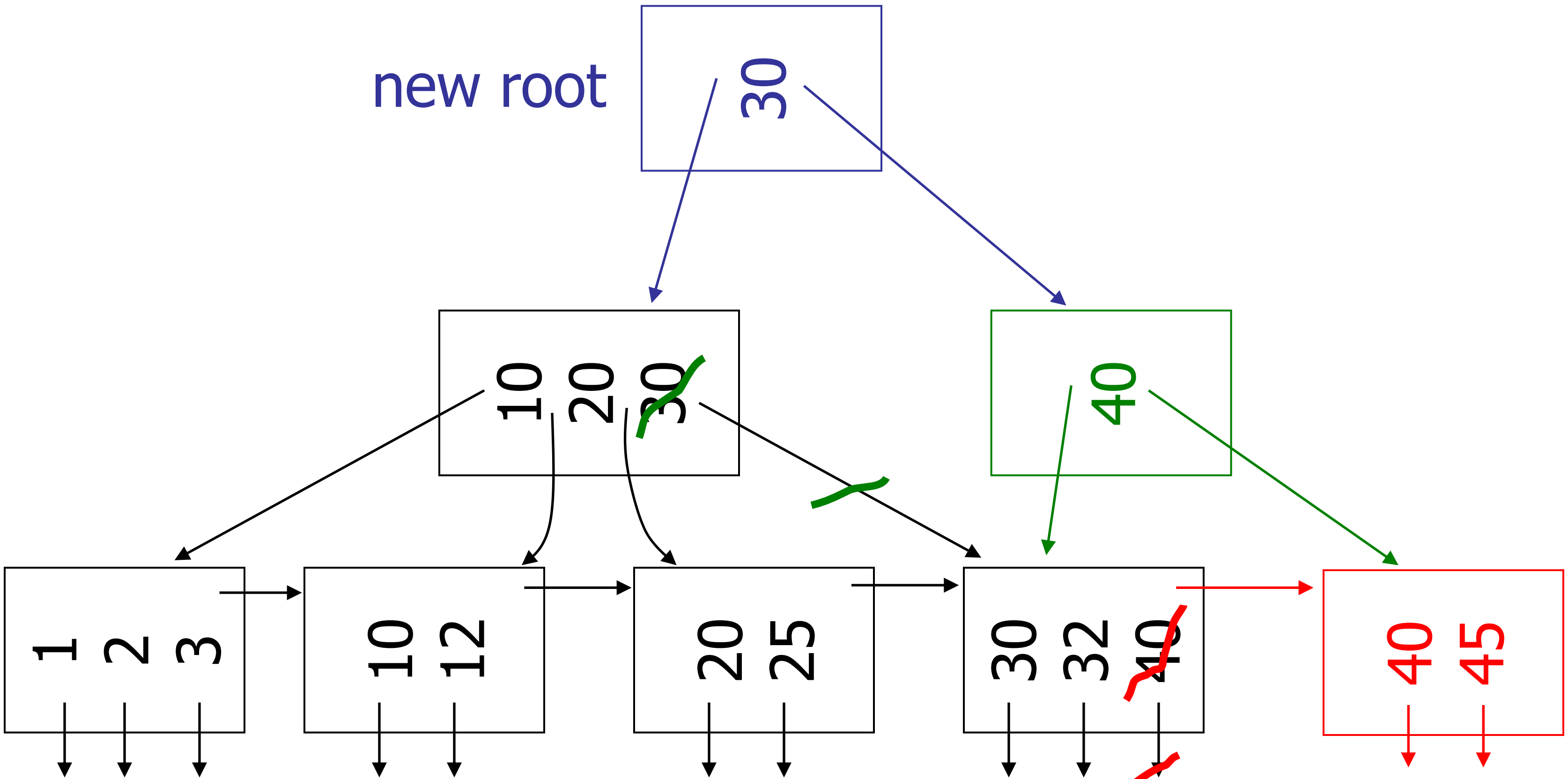
Data Structures

(c) Insert key = 160



B+-tree Index: *Insertion for m=4*

(d) New root, insert 45

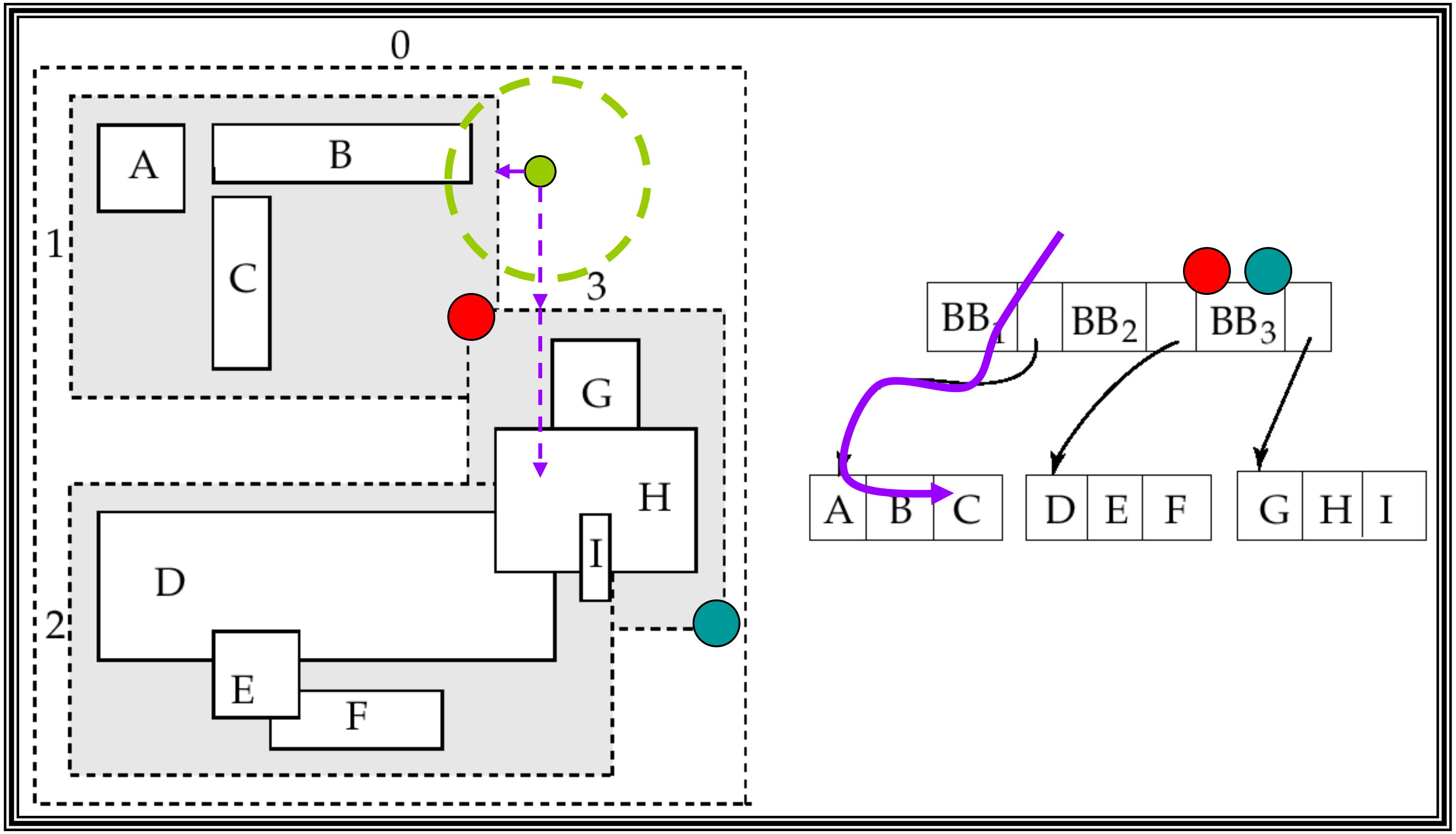


Variations of Hash File

- **Hash Indexing Methods**
  - Static Hash
    - **Fixed-length hash table**
  - Extensible Hash
    - **Hash table size is doubled if necessary**
  - Linear Hash
    - **Hash table size grows linearly**

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