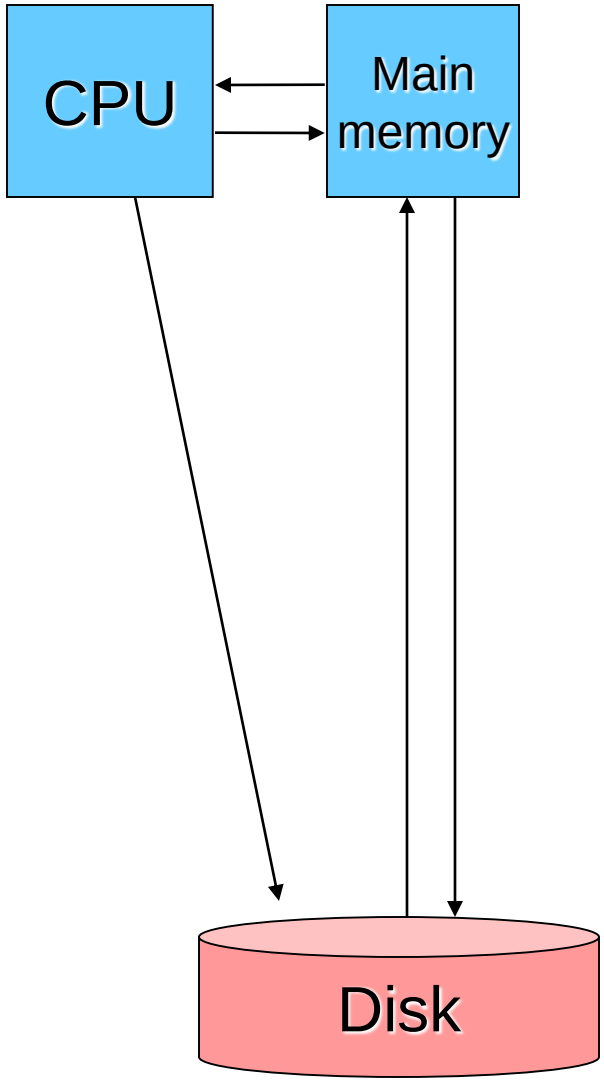
Ch. 13 External Methods

External Storage

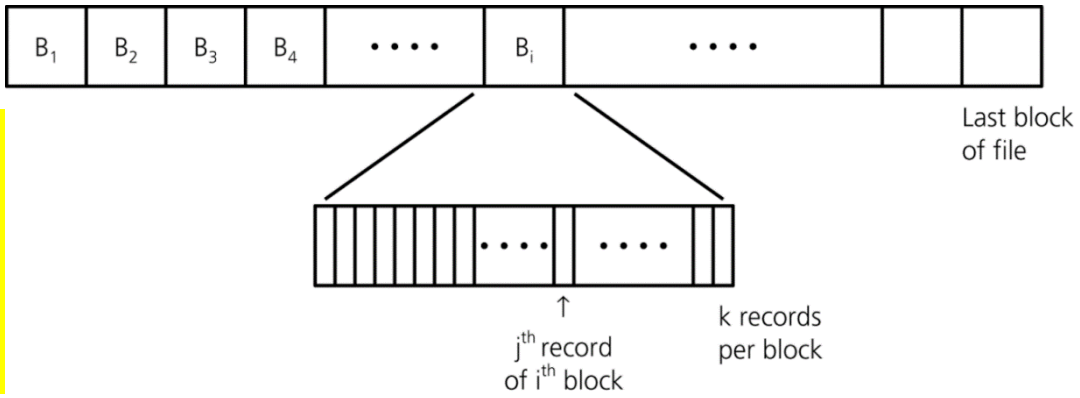
- Disk, CD-ROM, tape, …
- 사이즈가 큰 data는 모두 internal memory에 load 될 수 없다.
  - 전체의 일부만 main memory에 있을 수 있다.
- 한 번의 disk access는 수십만 번의 instruction 수행과 맞먹는다.
- External storage access가 포함된 작업에서는 대부분 external storage access가 시간을 지배한다.
- SSD(Solid-State Drive) 시대가 열려 external storage의 부담은 많이 줄었다. But, 여전히 RAM과의 차이는 크다.



- External storage access의 단위는 block
- Typical block sizes
  - 4K, 8K, 16K, 32K bytes
  - Sector는 물리적 구조, block은 추상적 구조
    - File system 단위로 세팅/변경할 수 있다
    - Unix/Linux의 예:
      - $ mkfs –t ext4 –b 4096 /dev/vdb1 ◄ 파일의 block size를 4K로 세팅
- Each block consists of multiple data records
- Typical access
  - *buf*.readBlock(dataFile, *i*) ◄ dataFile에서 *i*번 block을 *buf*로 읽어들임
    - *buf*: object for memory buffer
    - dataFile: file containing the data
    - *i*: index of block

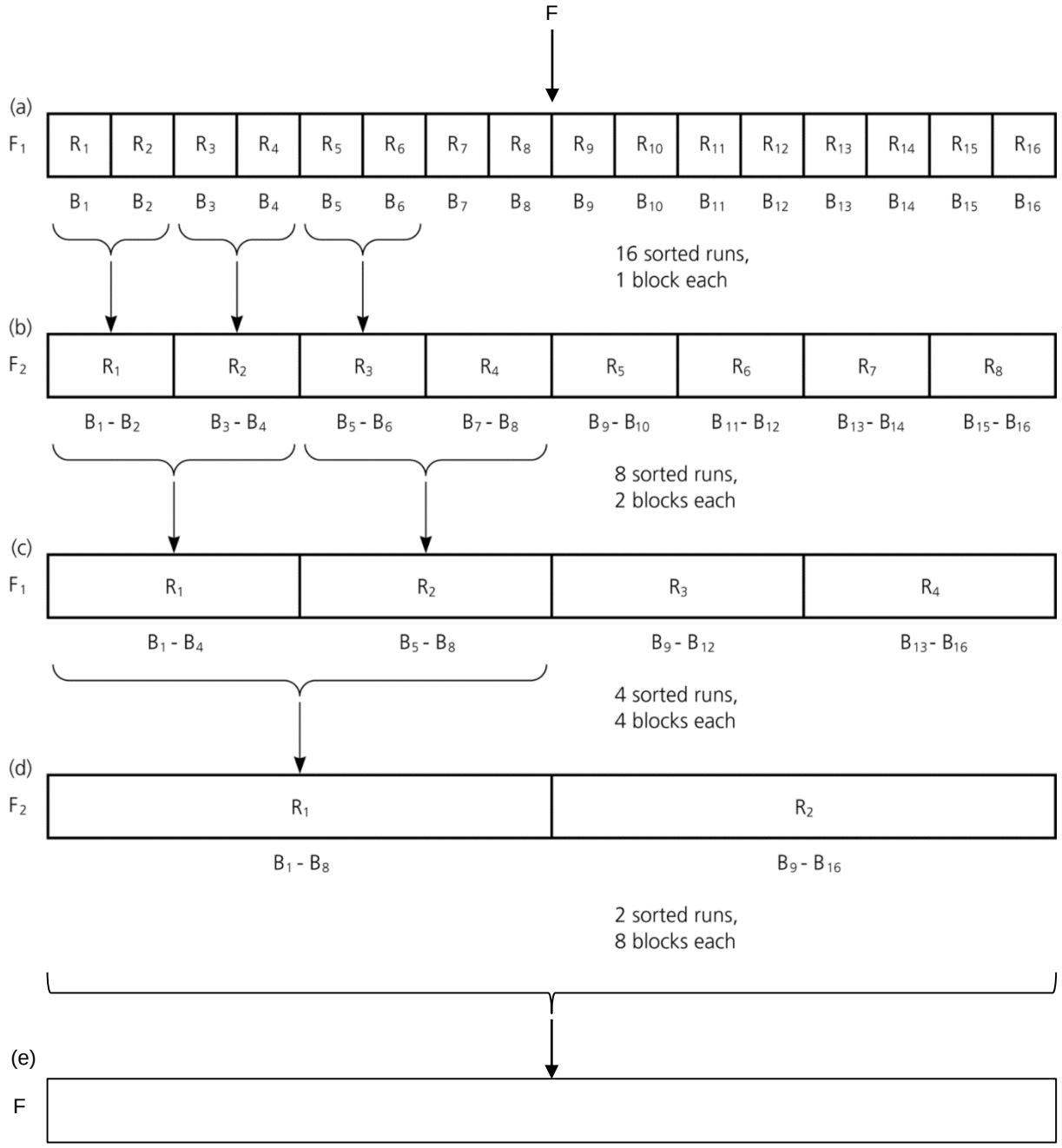
Java

java.lang.Object
  java.io.Reader
    java.io.BufferedReader
  java.io.Writer
    java.io.BufferedWriter



External Mergesort

- 비현실적인 예제
  - 1,600 employee records
  - 100 records/block
  - Maximum 300 records on memory
- Files
  - F : Original data
  - F_1, F_2: work files



32 i/o-block accesses

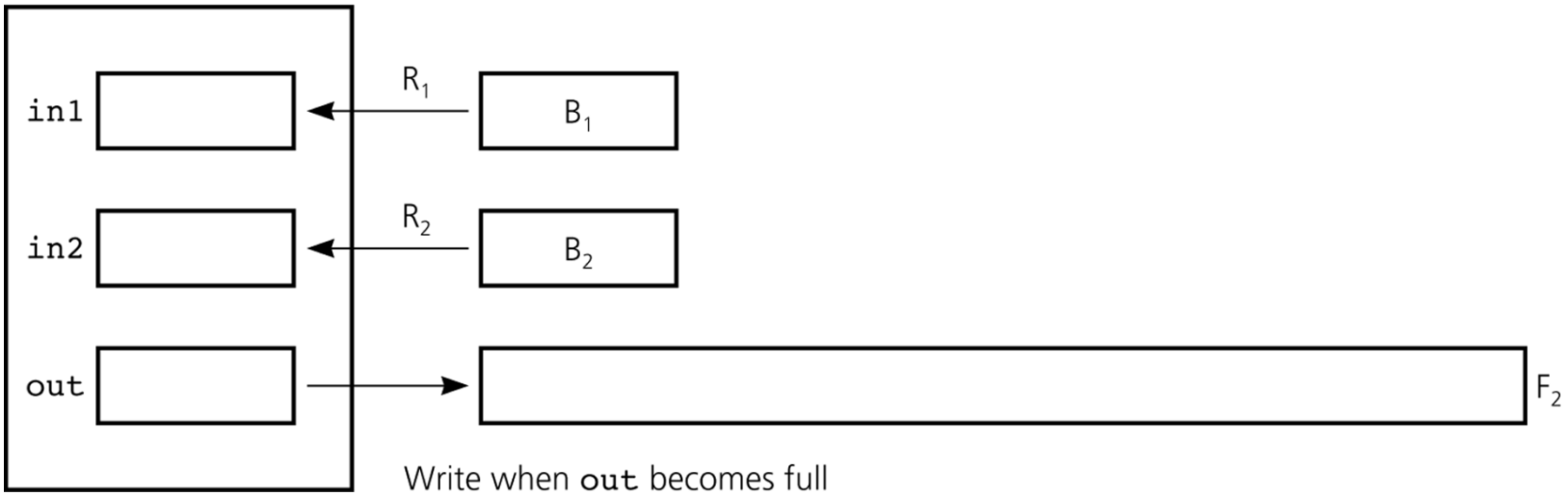
32 i/o-block accesses

32 i/o-block accesses

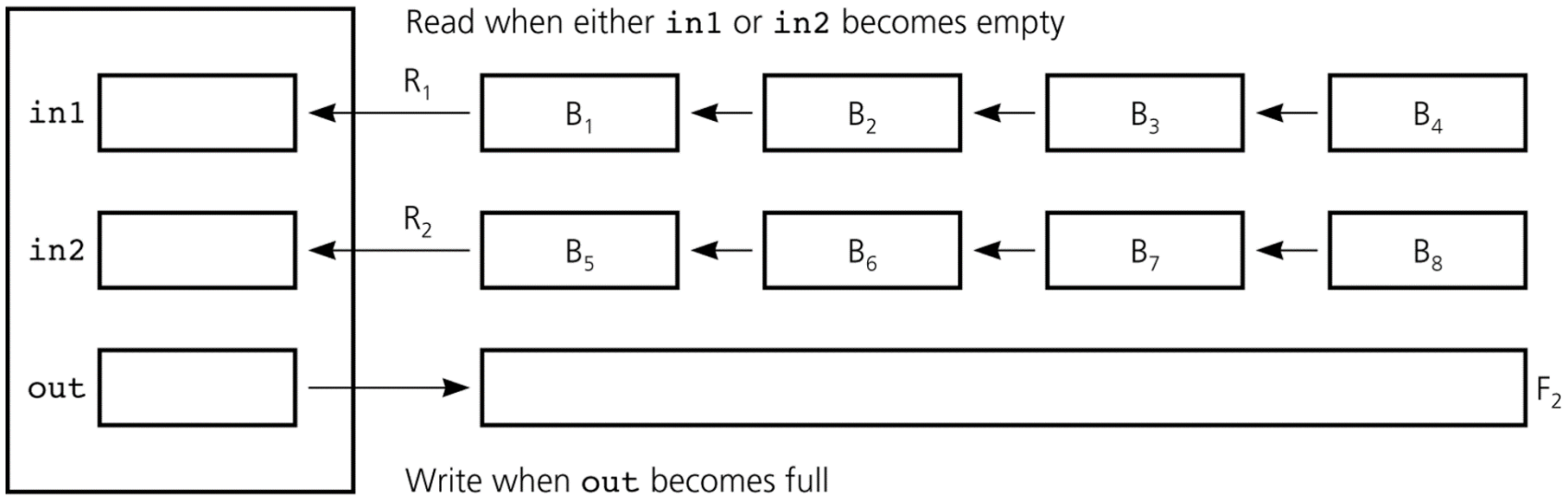
32 i/o-block accesses

32 i/o-block accesses

Merging Single Blocks

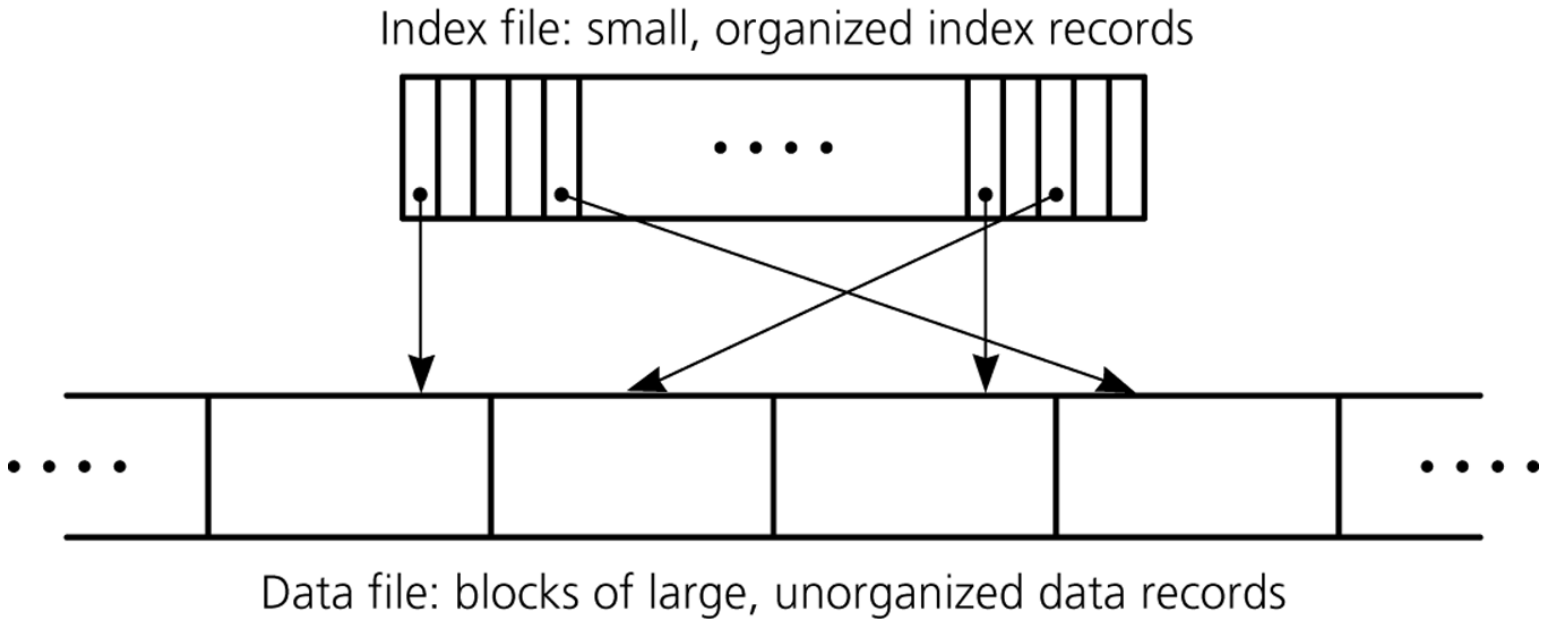


Merging Two 4-Block Runs

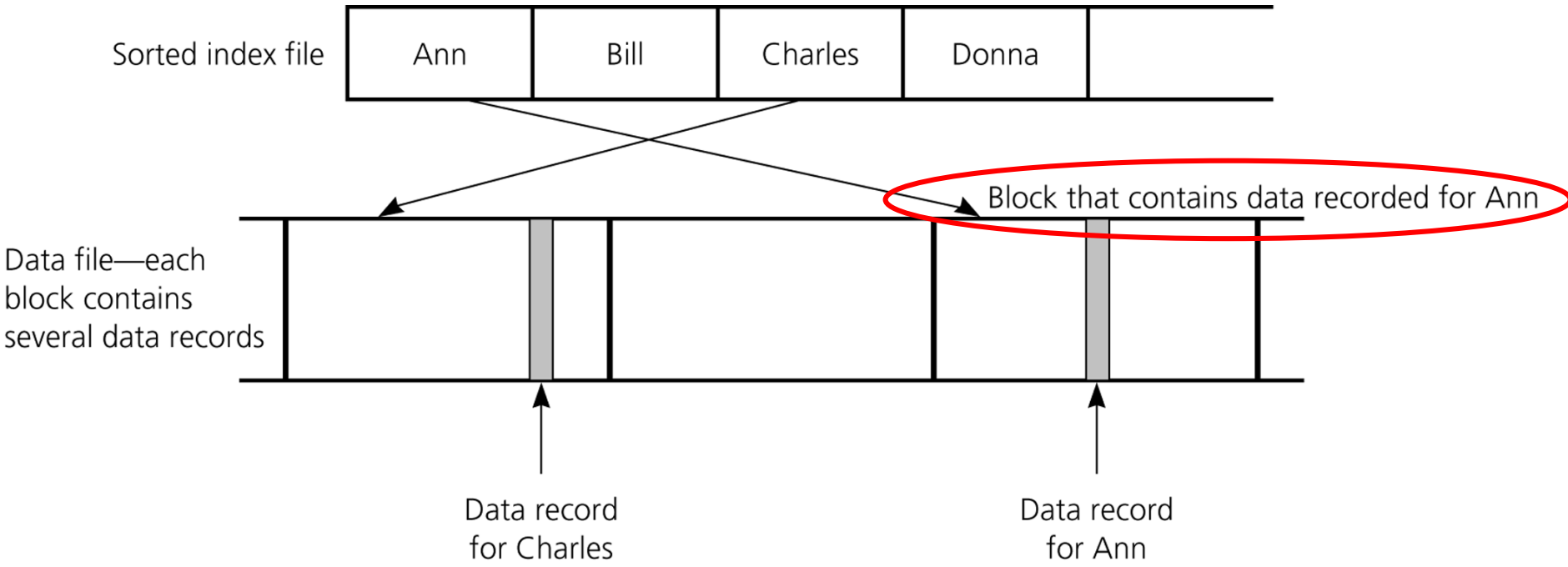


External Tables

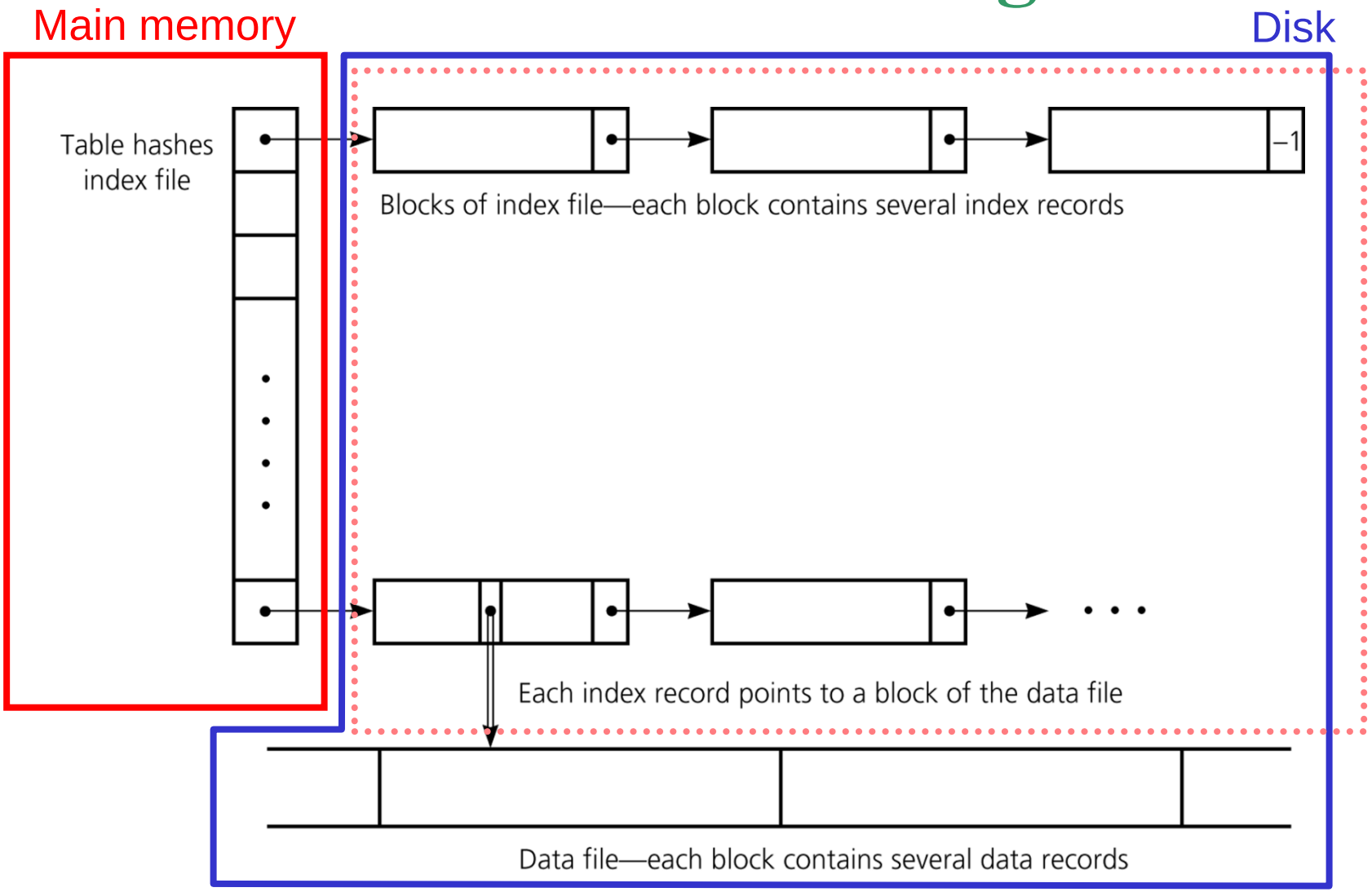
- Data are stored in an external storage
- External hashing
- B-trees



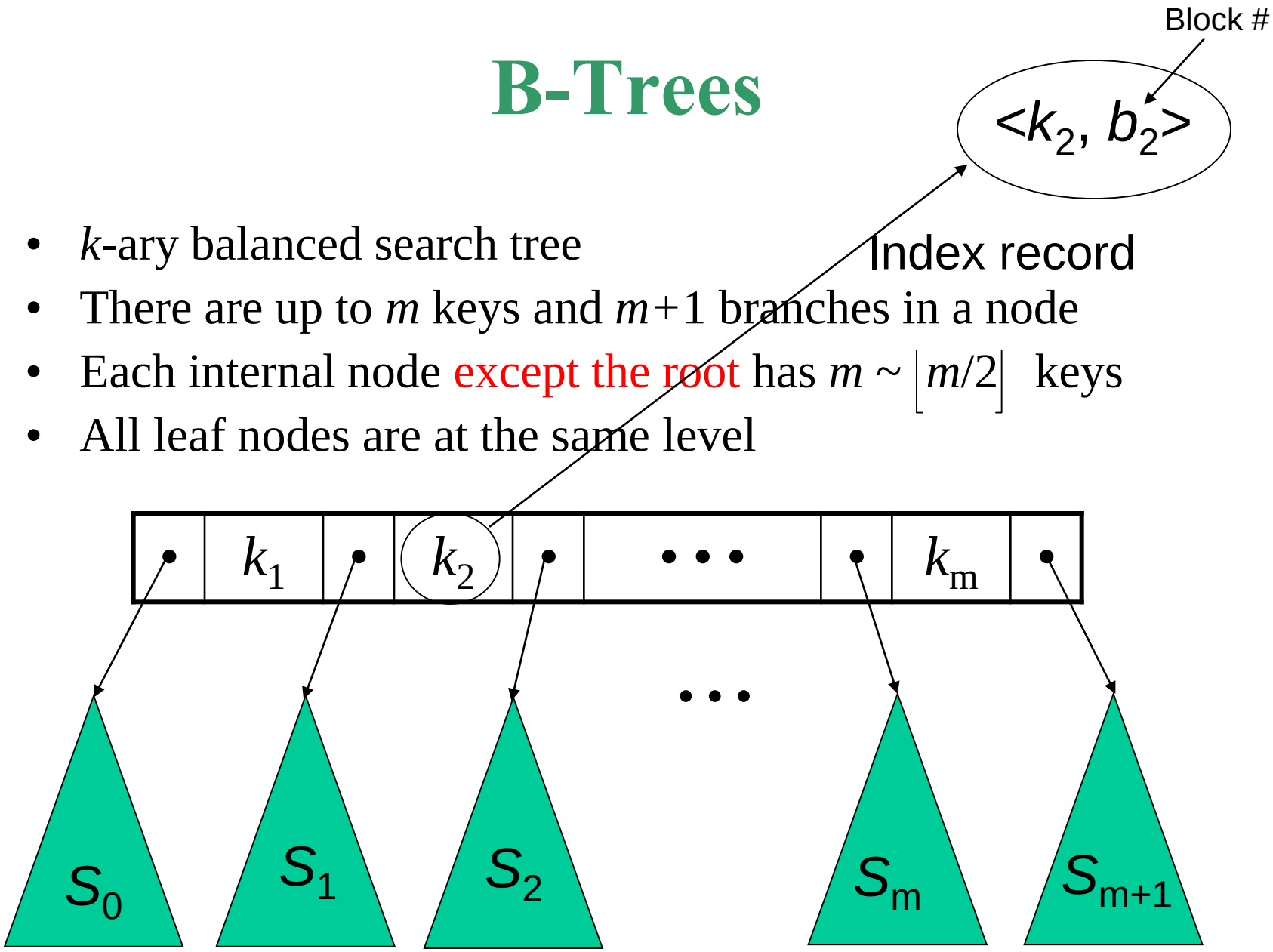
In external table, indices are managed not by references but by block numbers



External Hashing



B-Trees



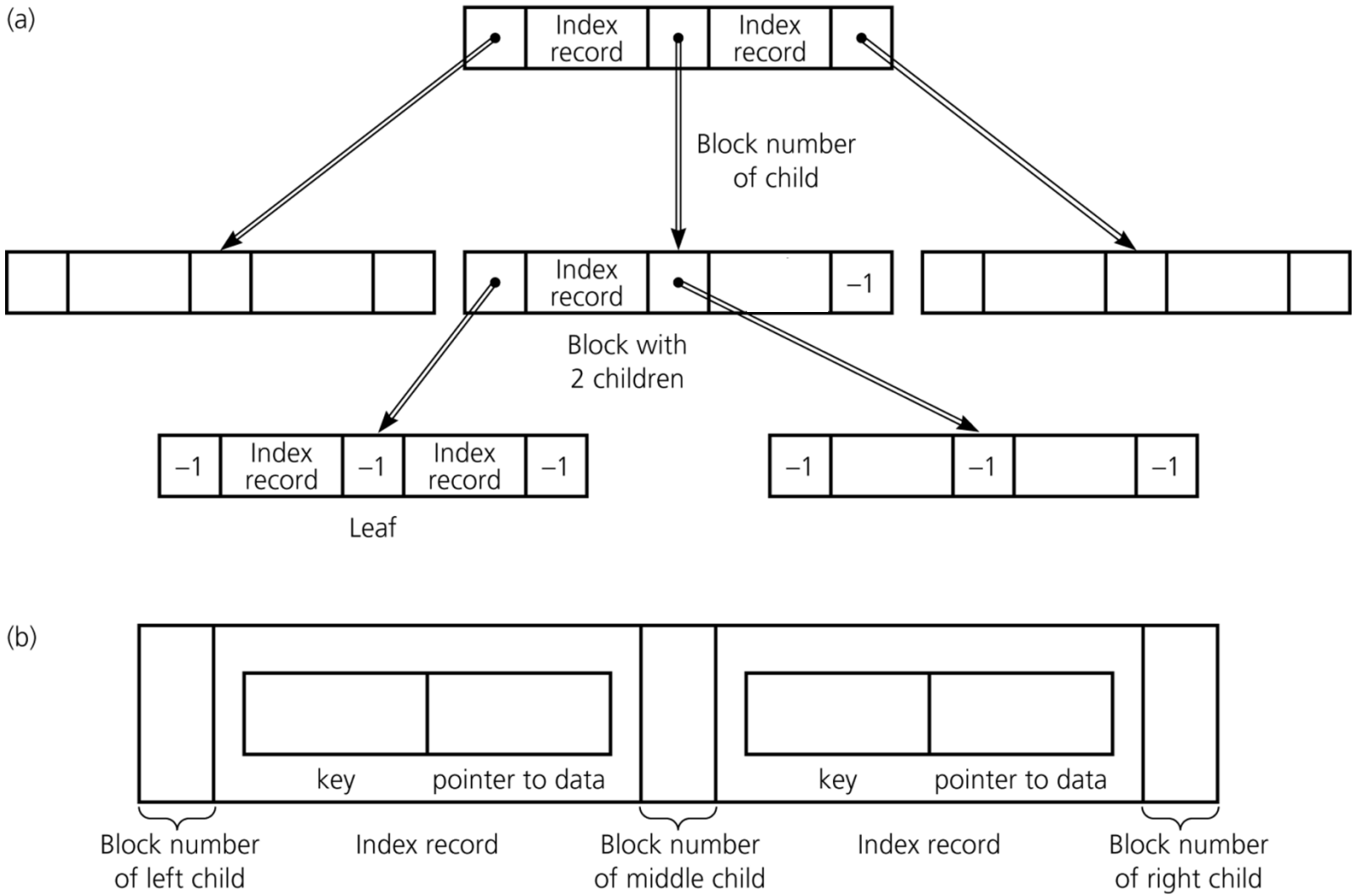
- k-ary balanced search tree
- There are up to m keys and $m+1$ branches in a node
- Each internal node except the root has $m \sim \lfloor m/2 \rfloor$ keys
- All leaf nodes are at the same level

Retrieval in a B-tree

```
BRetrieve(fIndex, fData, rootNum, key)
// fIndex: B-tree file; fData: data file; key: search key
// rootNum: block # that contains the root of the tree(subtree)
{
        if (rootNum == –1) return null;
        buf.readBlock(fIndex, rootNum);
        if (key is one of the k_i's in the root) { // get data item
                b_i = data-file block # that index record specifies;
                buf.readBlock(fData, b_i);
                return the item corresponding to key from buf;
        } else { // branch
                b_i = block # to branch;
                BRetrieve(fIndex, fData, b_i, key);
        }
}
```

✓ 각 노드가 하나씩의 disk block

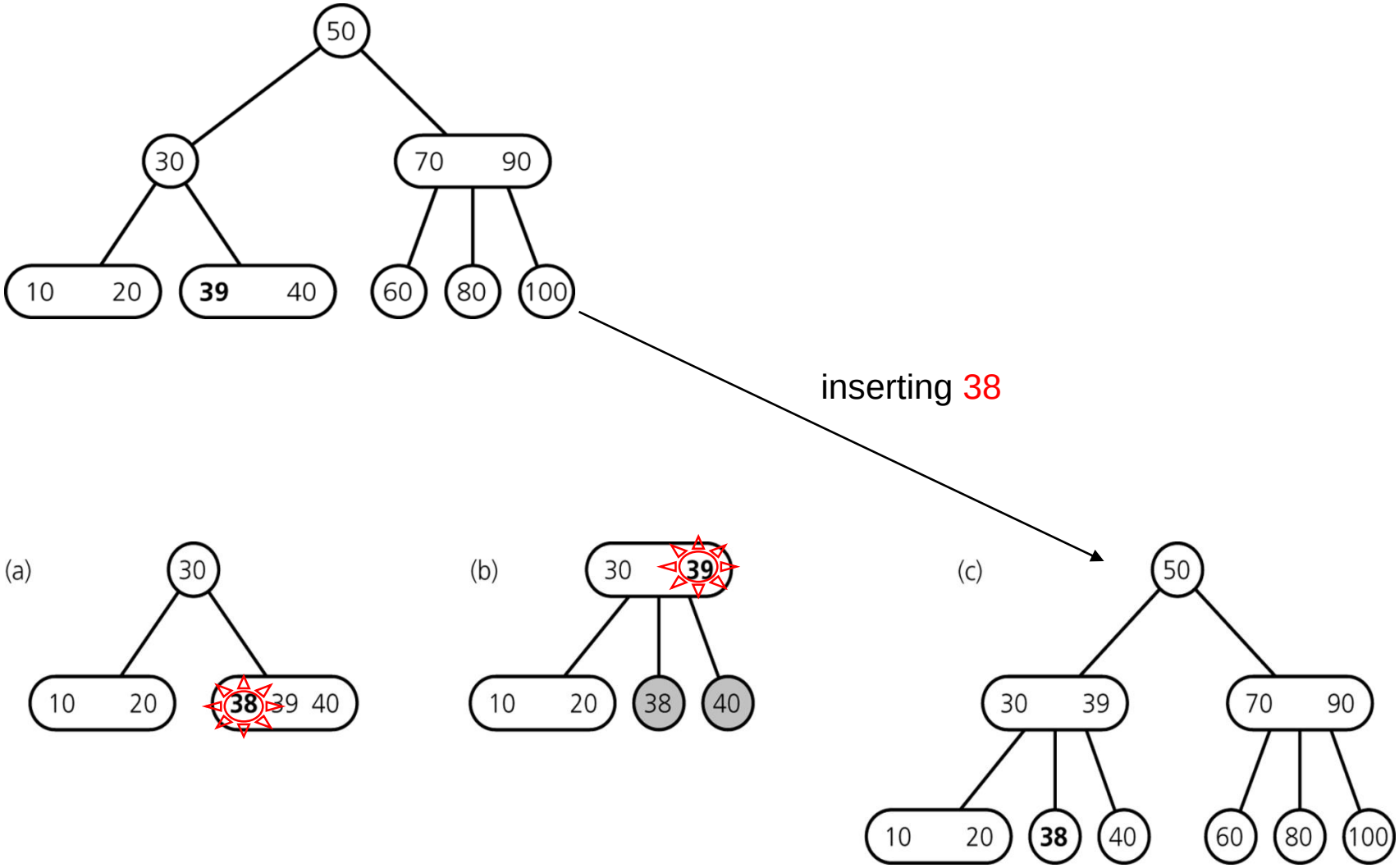
Node Structure of a B-tree



Insertion in a B-Tree

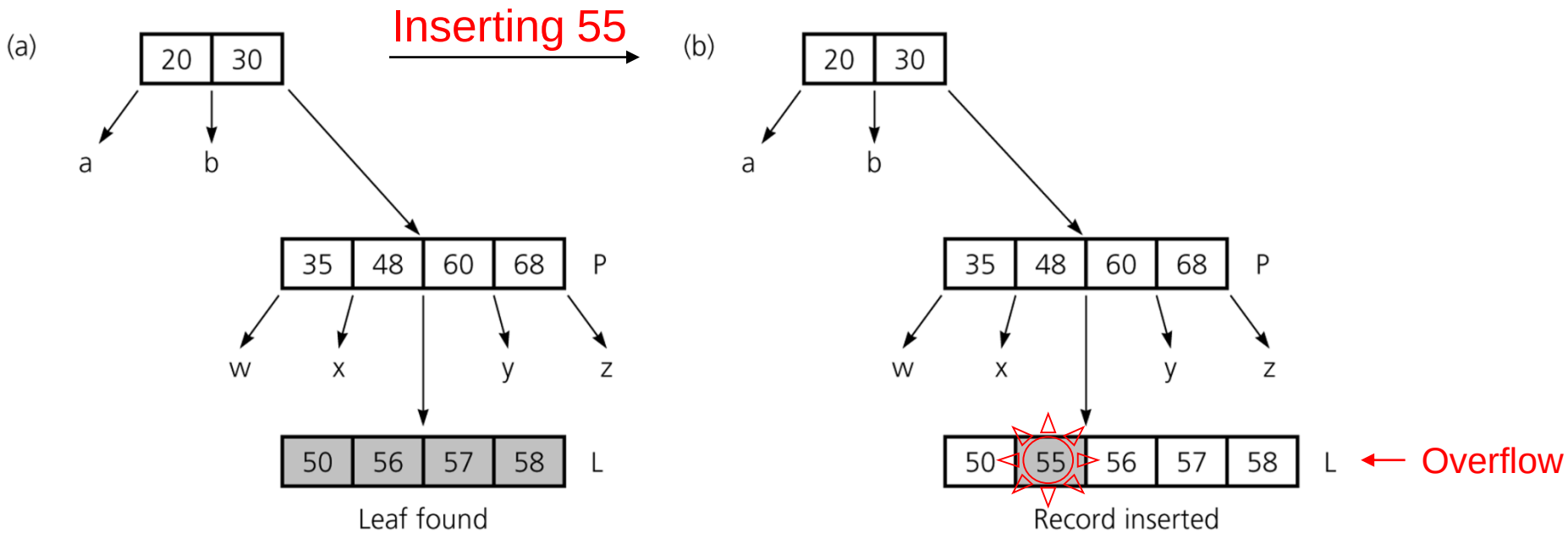
- Insert the record w/ key *x* into the data file
  - Any block *p* with a vacant slot or a new block
- Insert the index record into the index file
  - Insert the index record <*x*, *p*> into the B-tree
  - Analogous to the insertion of 2-3 tree

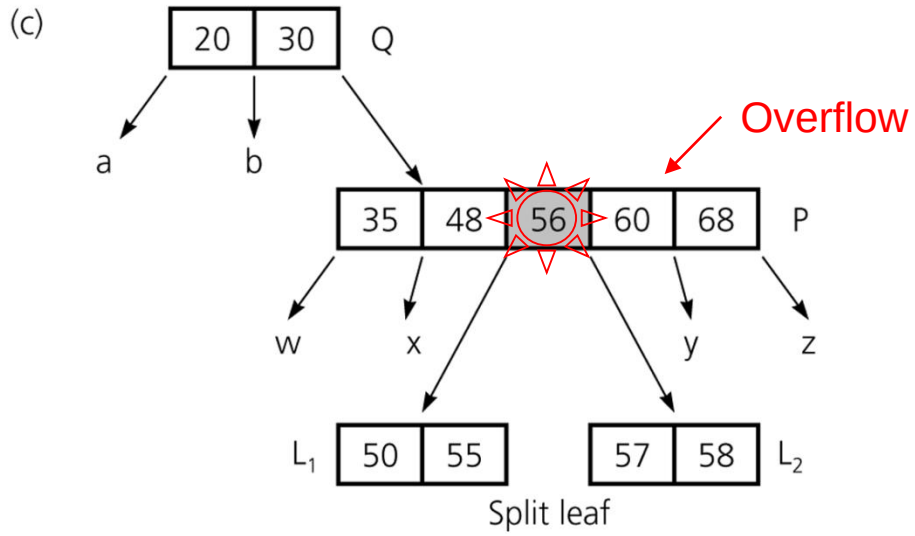
Remind: Insertion in 2-3 Trees

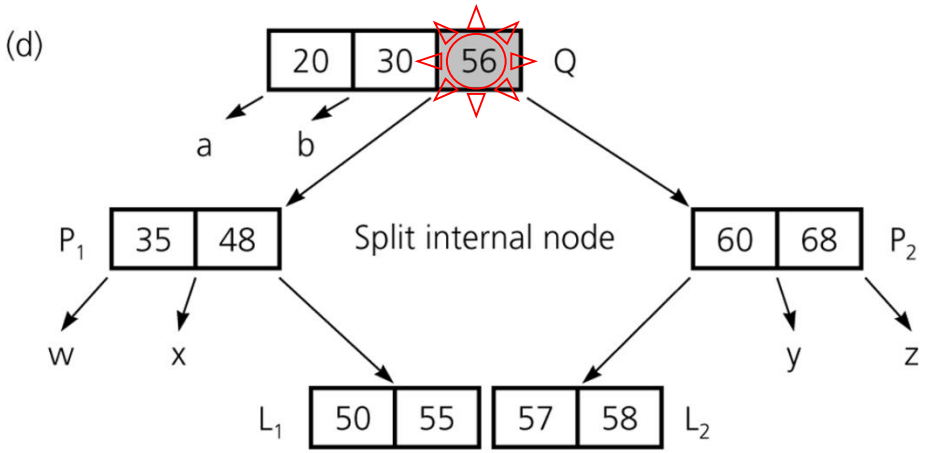


An Example Insertion in a B-Tree

가정: 노드당 최대 4개의 key 허용
(i.e., 2~4개)

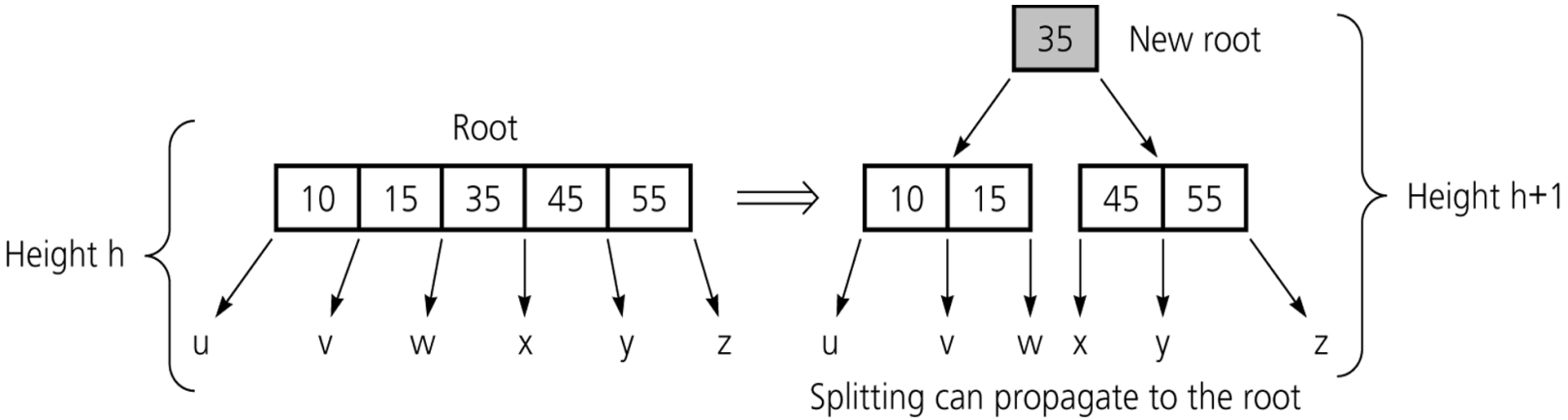






✓ 이 예는 그냥 split을 했지만
일반적으로는 split보다 redistribution을 먼저 시도한다

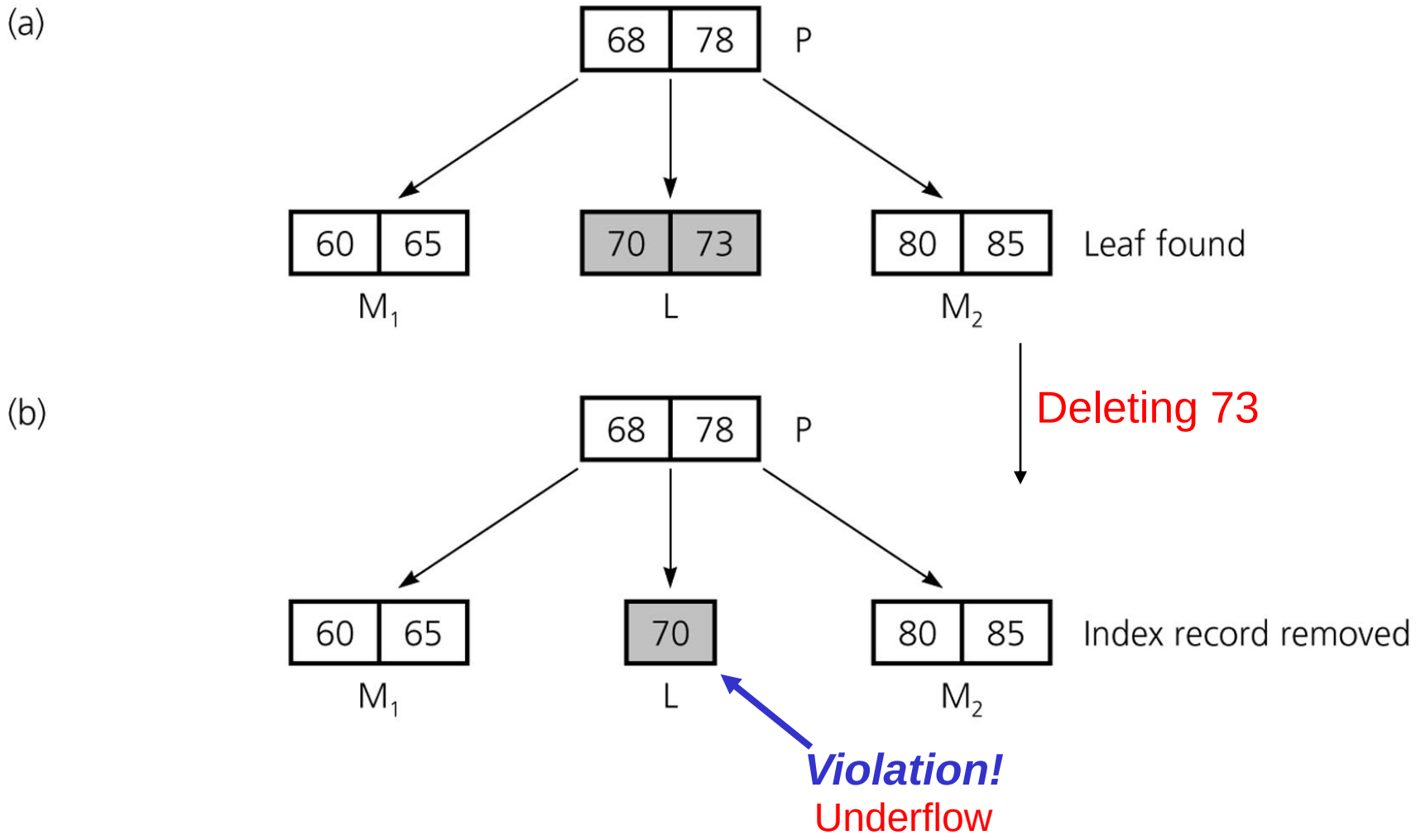
Splitting the Root in a B-Tree



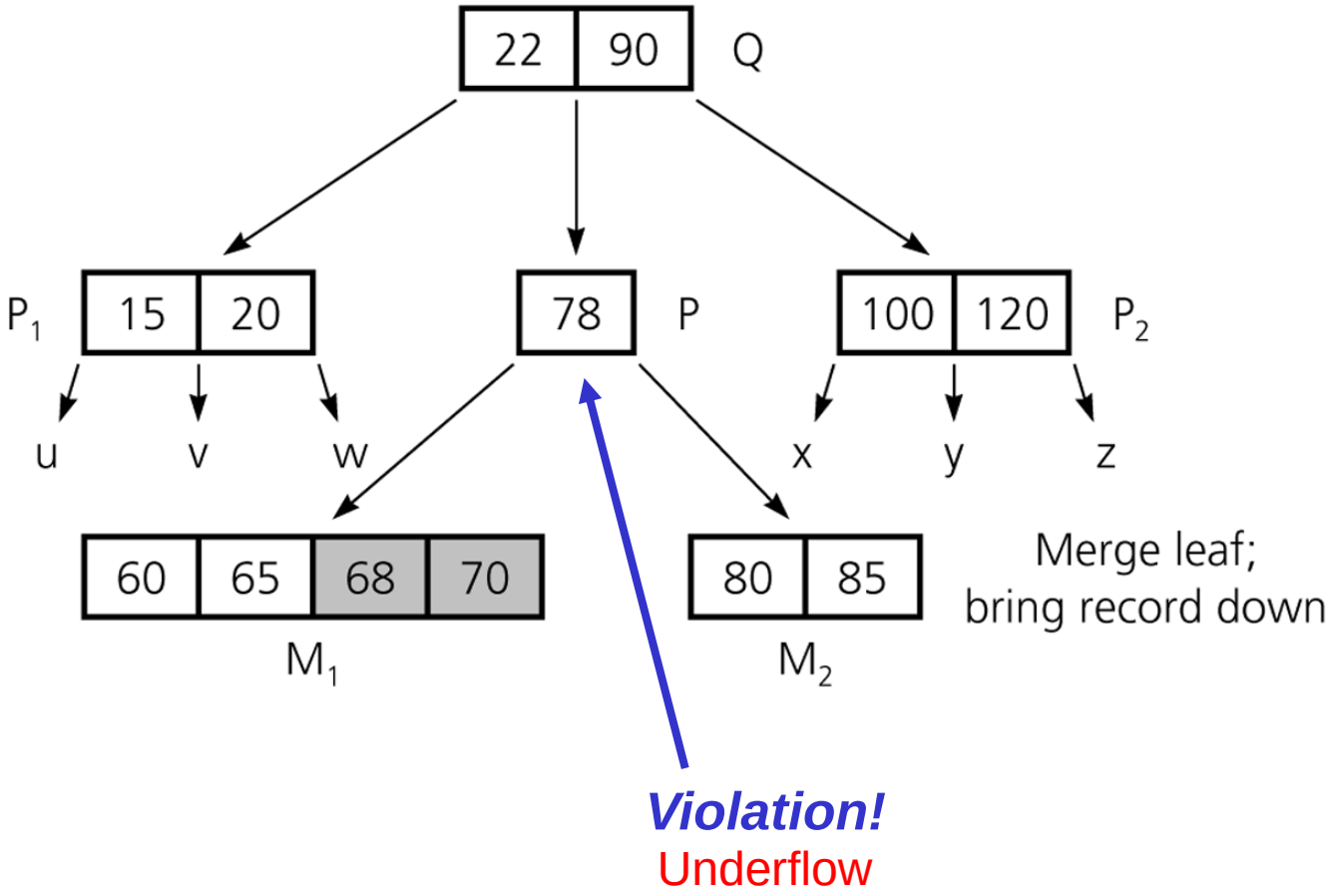
Deletion in a B-Tree

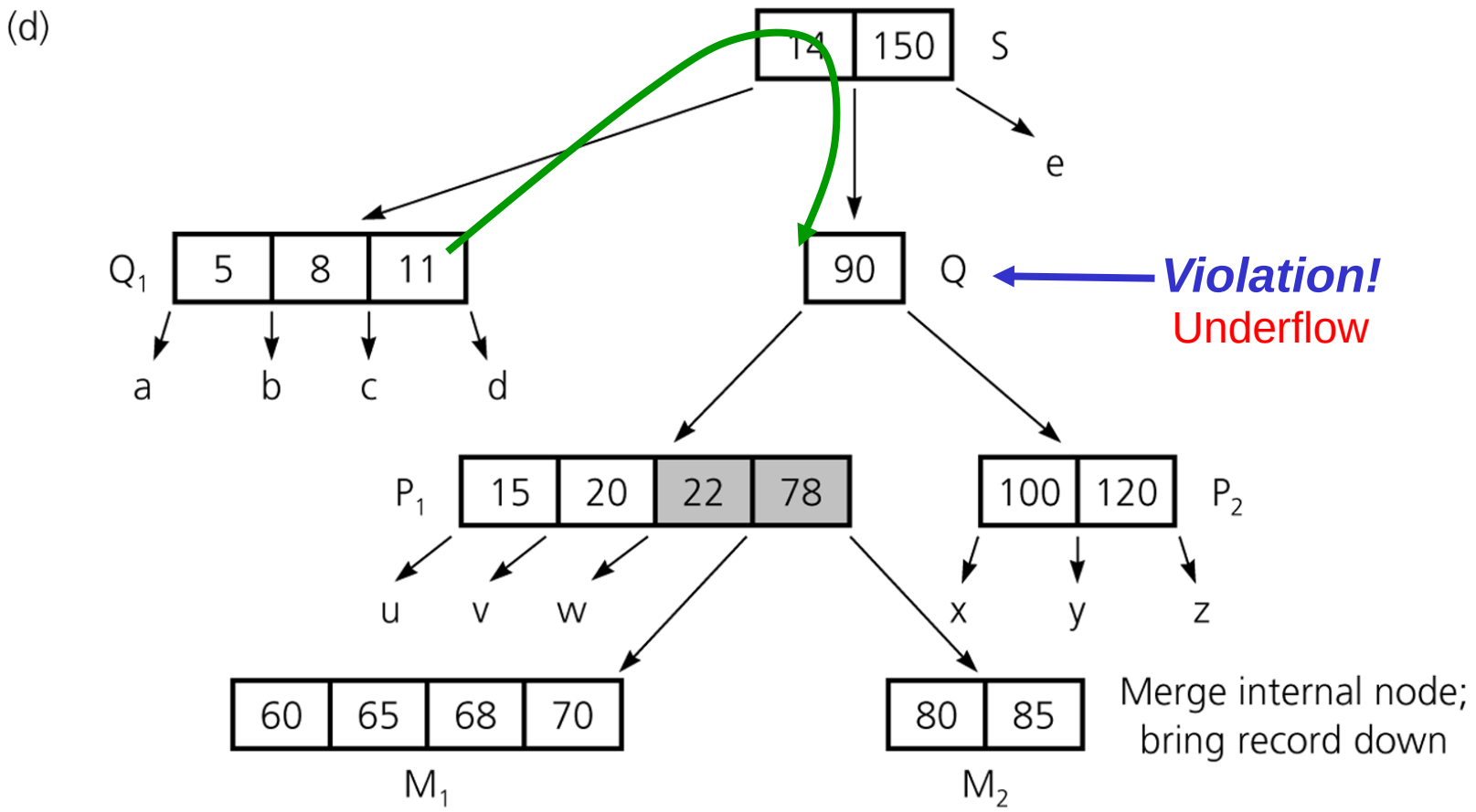
- Find the index record (with the key) and delete in the B-tree
  - Locate the index record
  - If the record is not in a leaf, swap it with its inorder successor
  - Delete the key
  - If the leaf now has fewer than $\lfloor m/2 \rfloor$ keys, merge appropriately
- Delete the corresponding item in the data file

An Example Deletion in a B-Tree

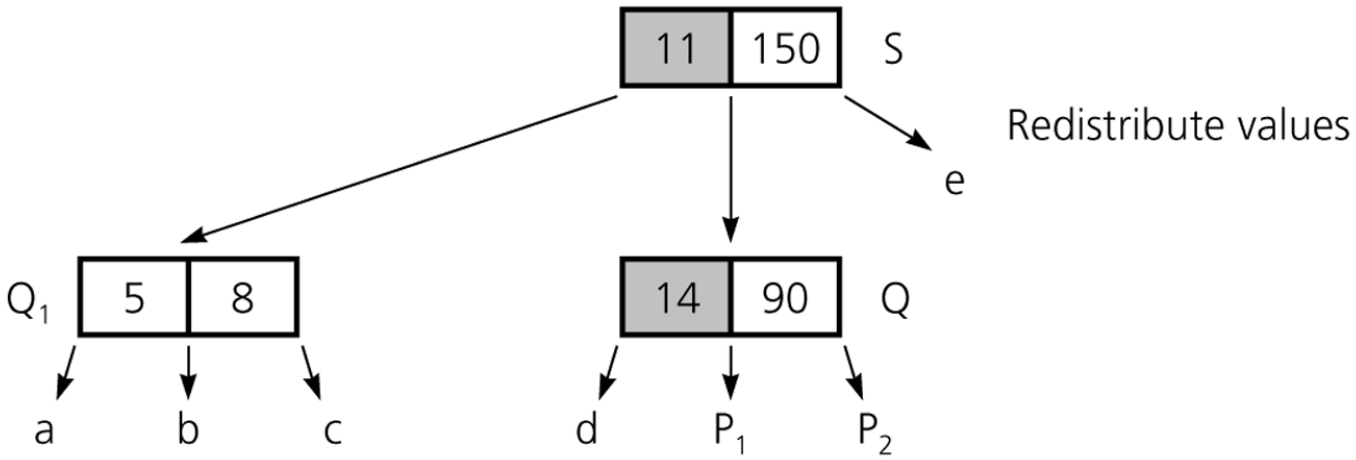


(c)


22
90
Q
P1
15
20
78
P
100
120
P2
u
v
w
x
y
z
60
65
68
70
M1
80
85
M2
Merge leaf;
bring record down
Violation!
Underflow

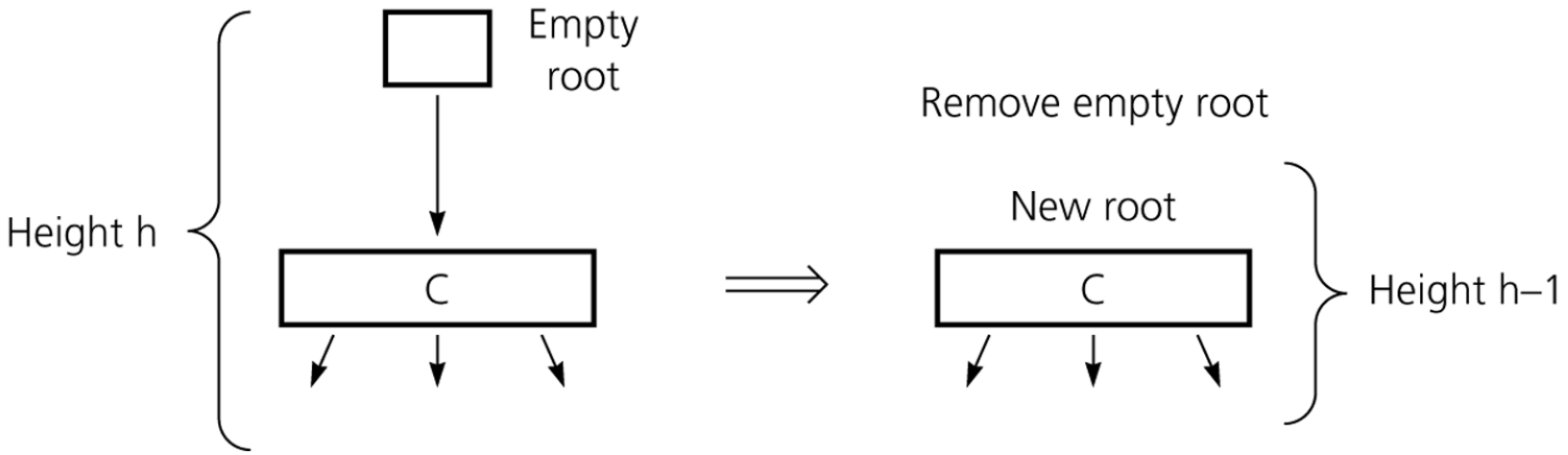

(d)
14
150
S
e
Q_1
5
8
11
a
b
c
d
90
Q
Violation!
Underflow
P_1
15
20
22
78
u
v
w
100
120
P_2
x
y
z
60
65
68
70
M_1
80
85
M_2
Merge internal node;
bring record down

(e)



Removing the Root in a B-Tree

Situation: merging w.r.t. the root w/ only one key



Multiple Indexing