CS411
Database Systems

07: Indexing

Why Do We Learn This?

Indexing

- Indexing
  - types of indexes
  - B+ trees
  - hash tables

Q: What is "indexing"?

- To build an index.
- But what is an index?

- Examples in the real world?

What is “indexing”?

Indexes

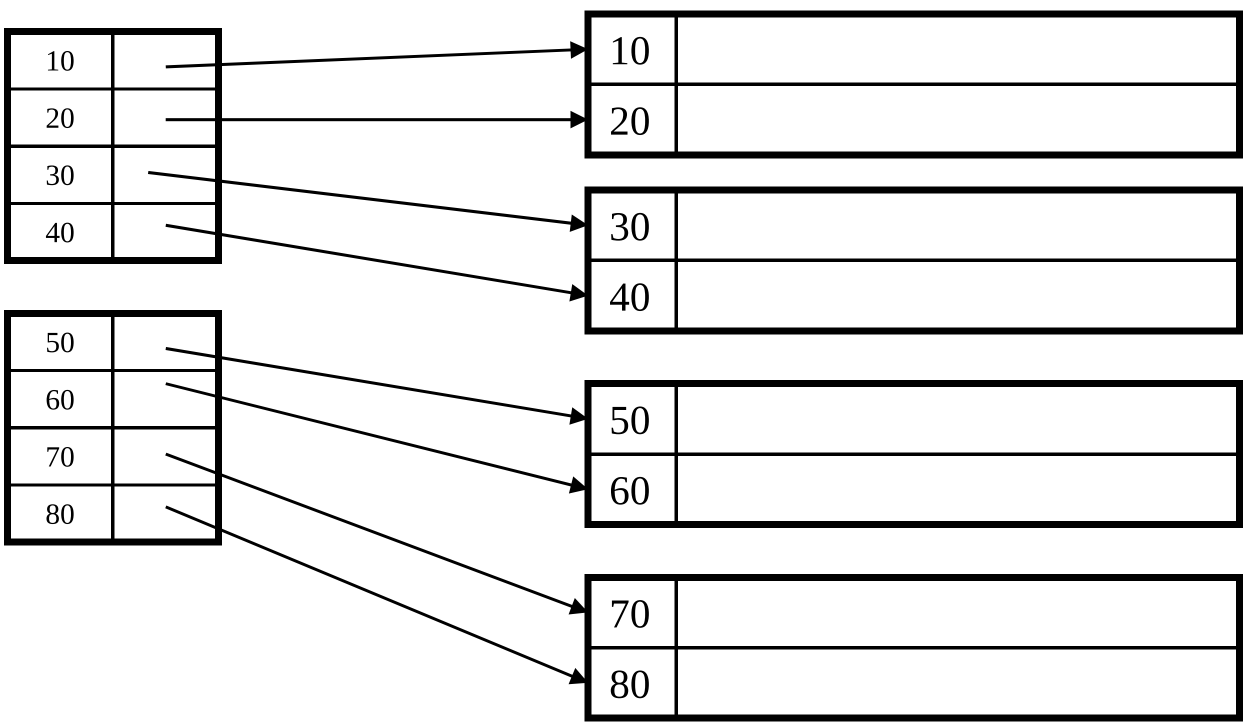
- An *index* on a file speeds up selections on the *search key field(s)*
- Search key = any subset of the fields of a relation
  - *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- Entries in an index: (k, r), where:
  - k = the key
  - r = the record OR record id OR record ids

Types of Indexes

- Clustered/unclustered
  - Clustered = records sorted in the key order
  - Unclustered = no
- Dense/sparse
  - Dense = each record has an entry in the index
  - Sparse = only some records have
- Primary/secondary
  - Primary = on the primary key
  - Secondary = on any key
  - Some textbooks interpret these differently
- B+ tree / Hash table / …

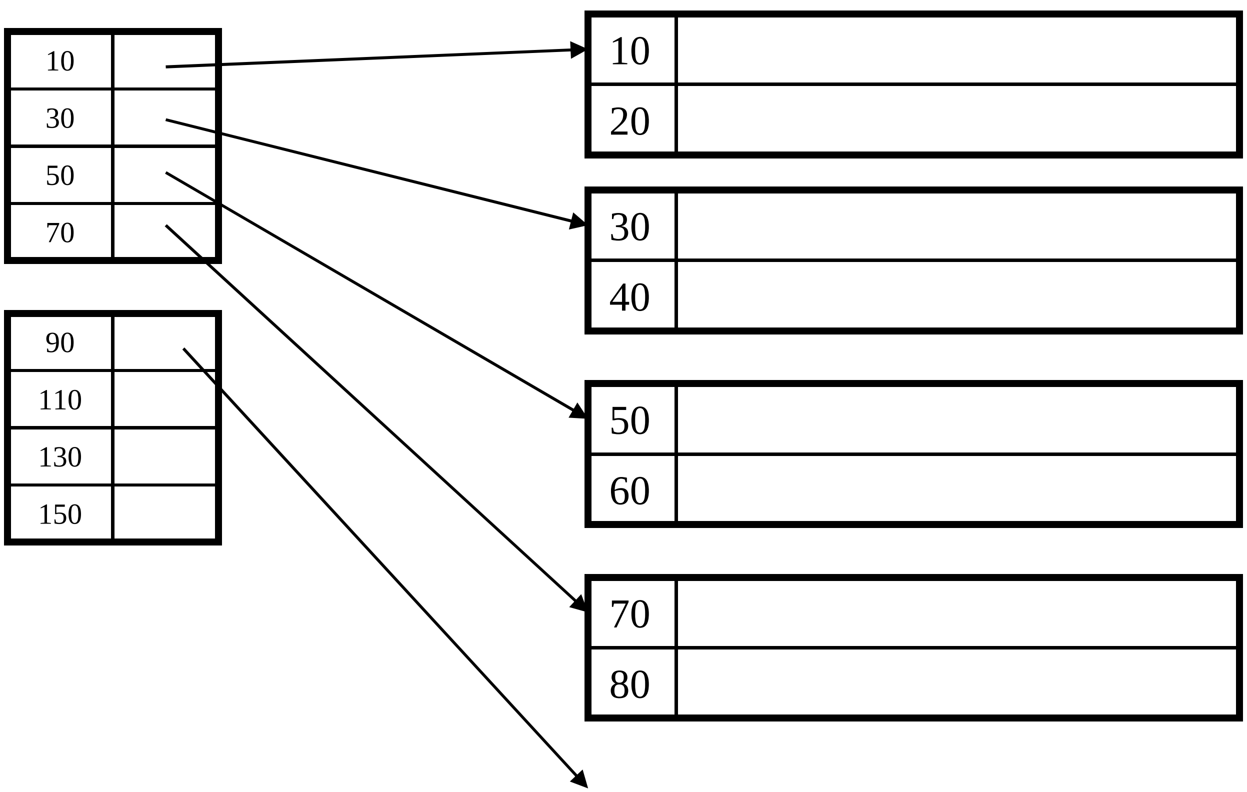
Ex: Clustered, Dense Index

- Clustered: File is sorted on the index attribute
- ***Dense***: sequence of (key,pointer) pairs



Clustered, Sparse Index

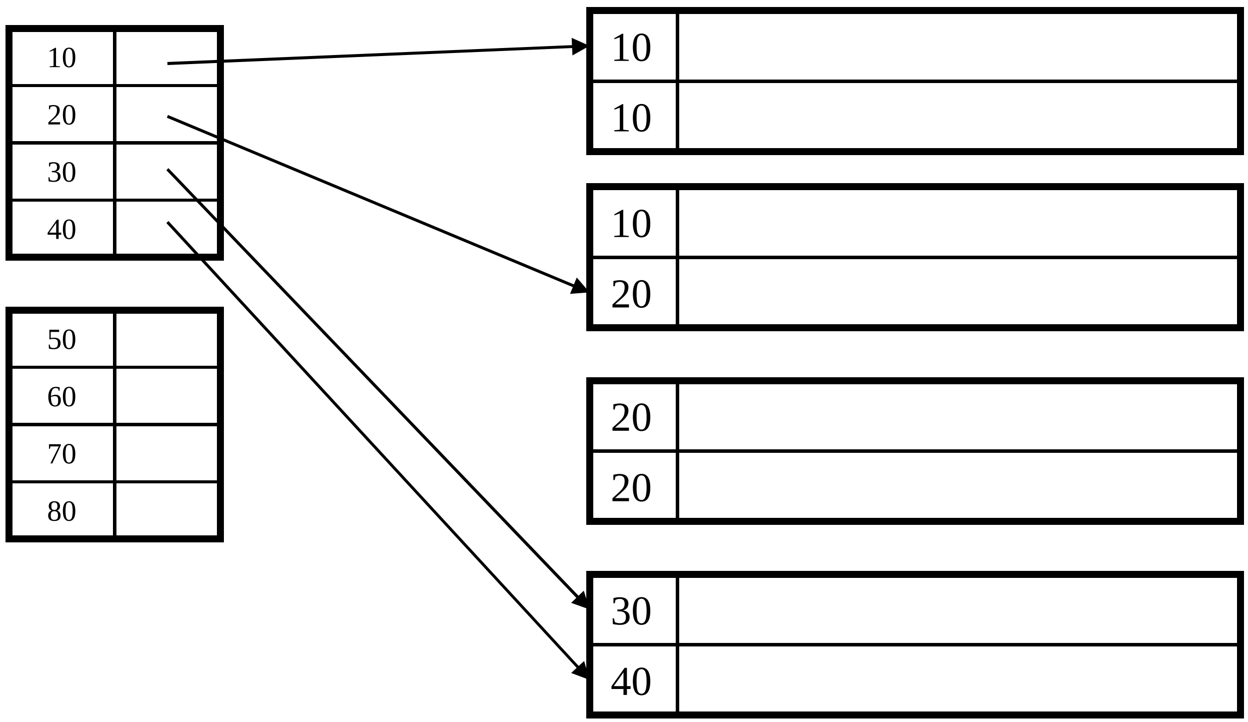
- *<u>Sparse</u>* index: one key per data block



How if duplicate keys?

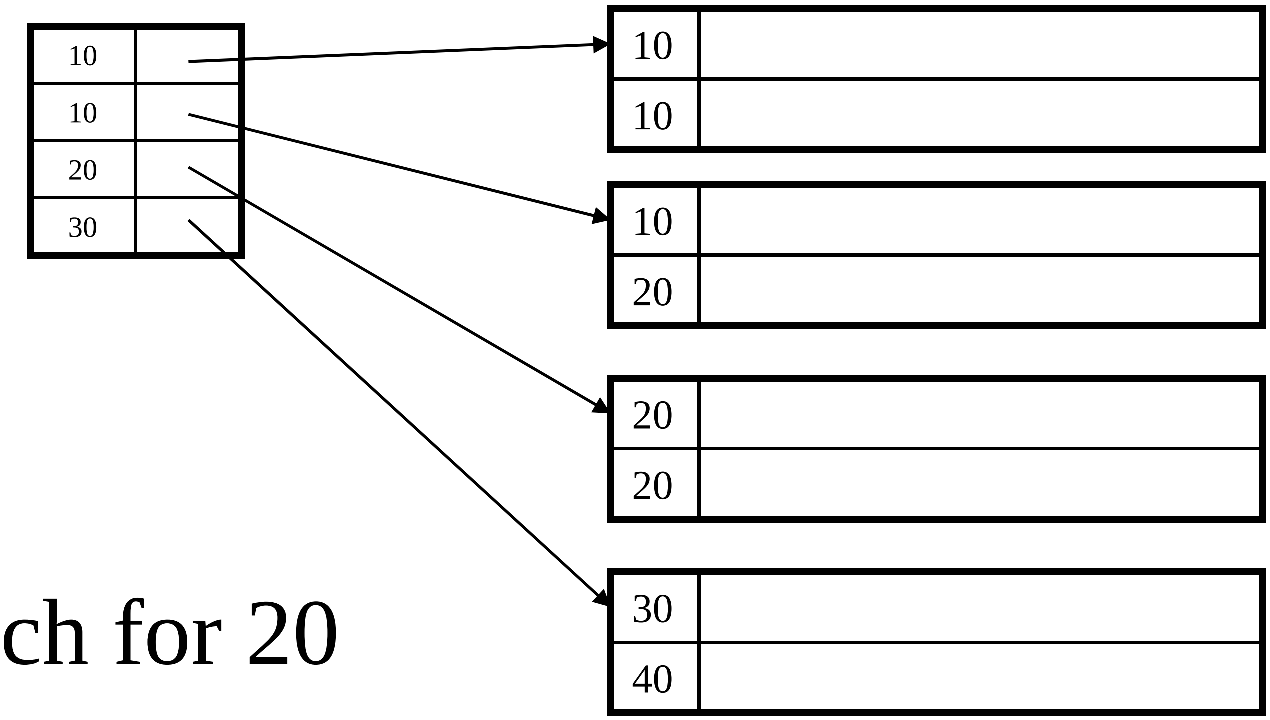
Clustered Index with Duplicate Keys

- Dense index: point to the first record with that key



Clustered Index with Duplicate Keys

- Sparse index: pointer to lowest search key in each block:

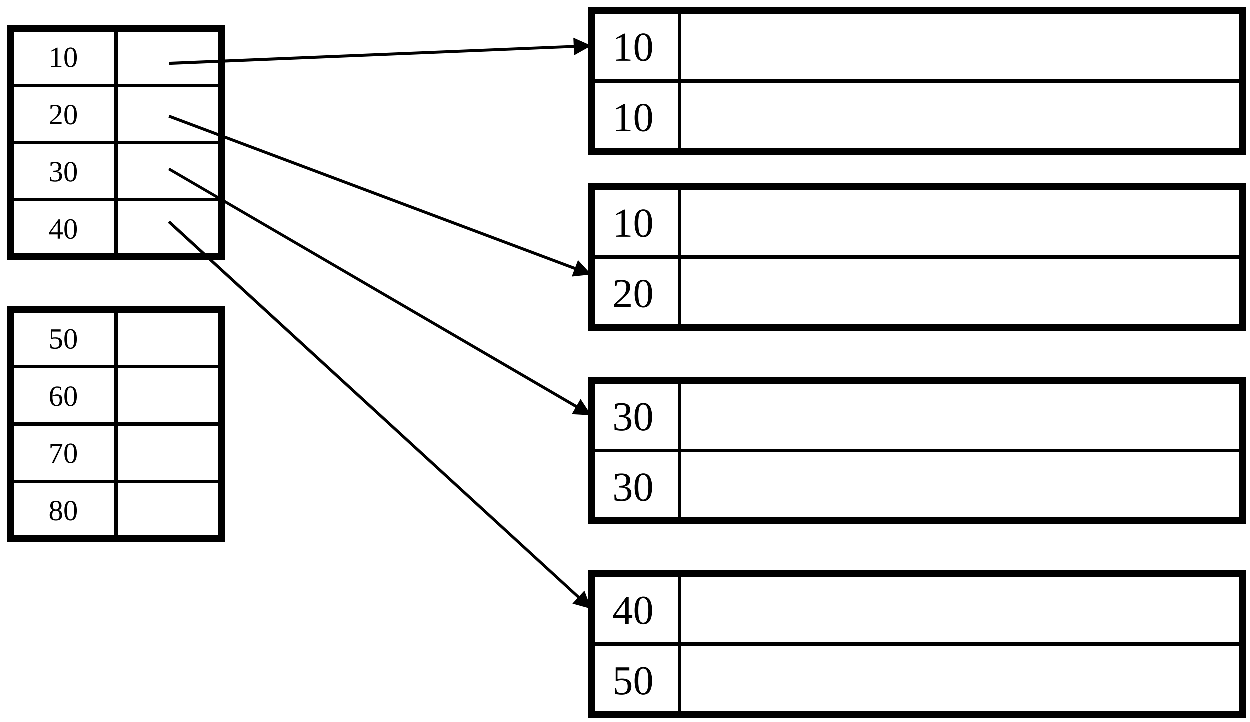


- OK?

  Try search for 20

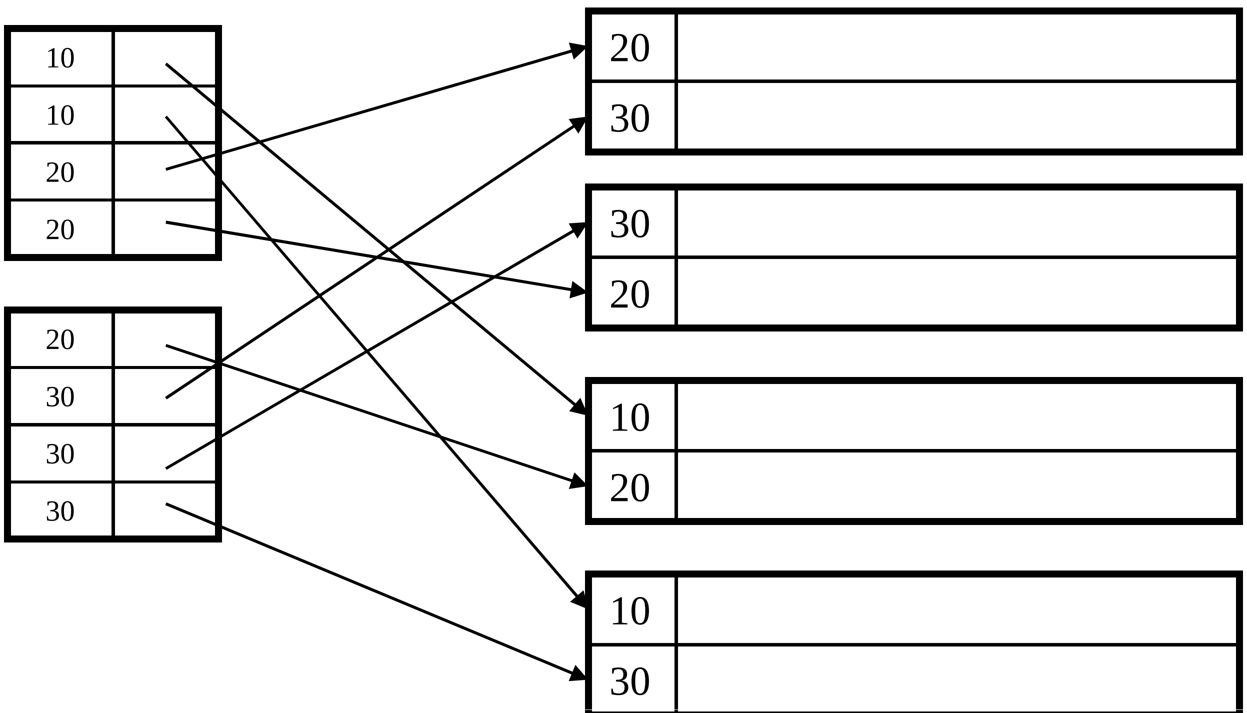
Clustered Index with Duplicate Keys

- Better: pointer to lowest new search key in each block:
- Search for 20

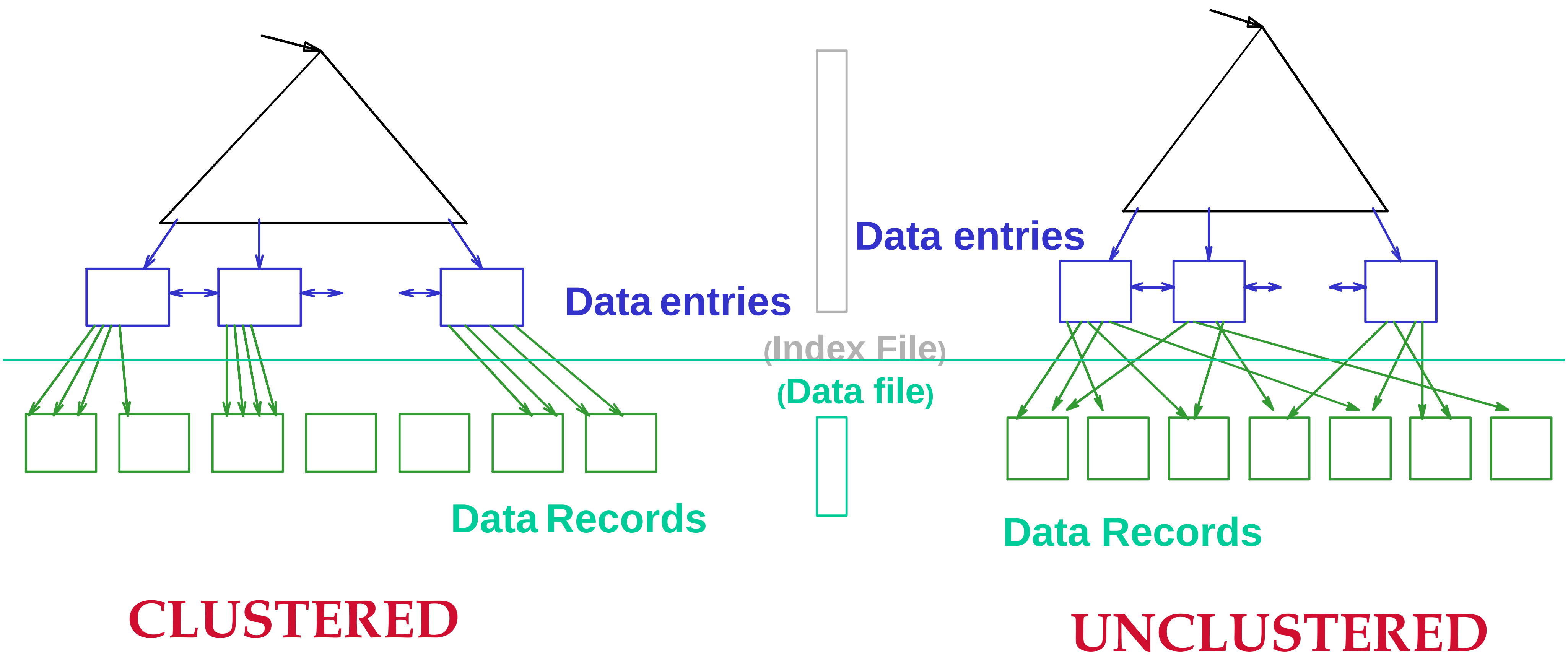


Unclustered Indexes

- Often for indexing other attributes than primary key
- Always dense (why ?)



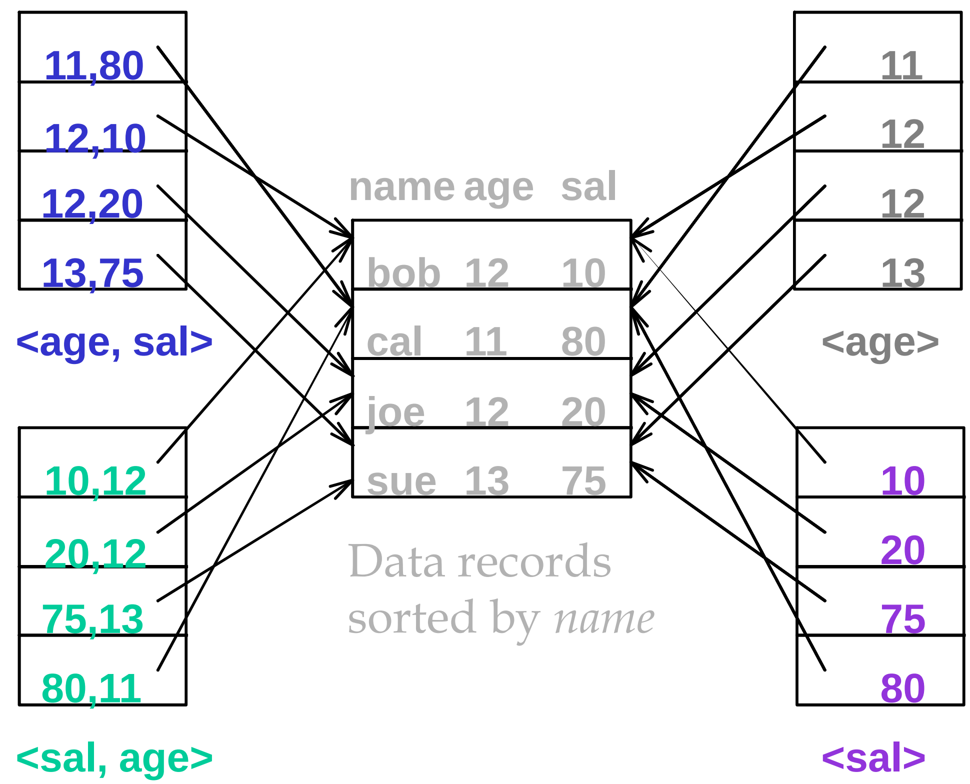
Summary Clustered vs. Unclustered Index



Composite Search Keys

- *Composite Search Keys:* Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g. wrt <sal,age> index:
    - age=20 and sal =75
  - Range query: Some field value is not a constant. E.g.:
    - age =20; or age=20 and sal > 10

Examples of composite key indexes using lexicographic order.

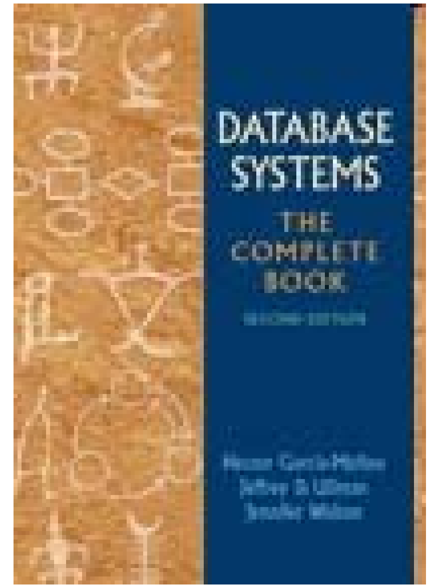


Data records sorted by *name*

Data entries in index sorted by <*sal,age*>

Data entries sorted by <*sal*>

Q: Our textbook as example: Indexes?



- How many indexes? Where?
- What are keys? What are records?
- Clustered?
- Dense?
- Primary?

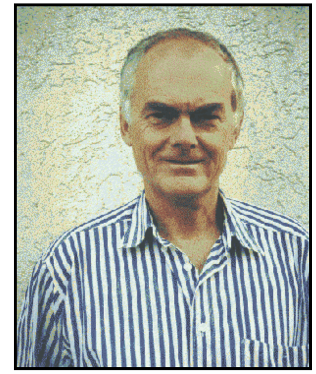
B+ Trees

What's wrong with sequential index?

B-Trees/B+Trees: B_?__?___ Trees

- Intuition:
  - Give up on sequentiality of index
  - Try to get “balance” by dynamic reorganization
- B+trees:
  - Textbook refers to B+trees (a popular variant) as B-trees (as most people do)
  - Distinction will be clear later (ok to confuse now)

Behind the Scene: UIUC (Alumni) Contribution!



Prof. Rudolf Bayer

Rudolf Bayer studied Mathematics in Munich and at the University of Illinois, where he received his Ph.D. in 1966. After working at Boeing Research Labs he became an Associate Professor at Purdue University. He is a Professor of Informatics at the Technische Universität München since 1972 and … …

The 2001 SIGMOD Innovations Award goes to Prof. Rudolf Bayer of the Technical University of Munich, for his invention of the B-Tree (with Edward M. McCreight), of B-Tree prefix compression, and of lock coupling (a.k.a. crabbing) for concurrent access to B-Trees (with Mario Schkolnick). All of these techniques are widely used in commercial database products. …..

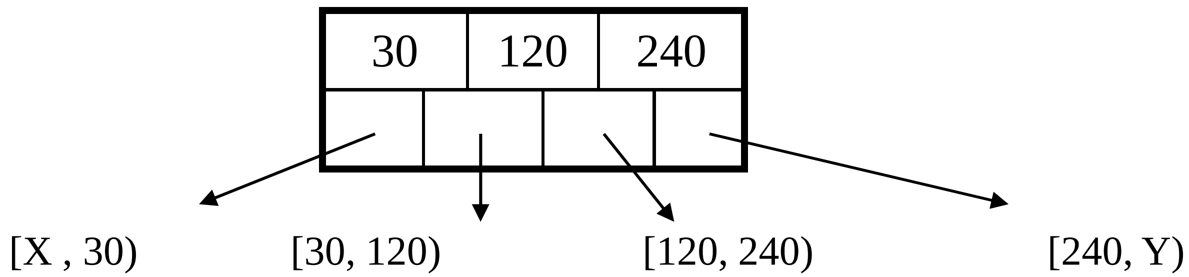
<u>The Original Publication</u>

Rudolf Bayer, Edward M. McCreight: Organization and Maintenance of Large Ordered Indices. Acta Informatica 1: 173-189(1972)

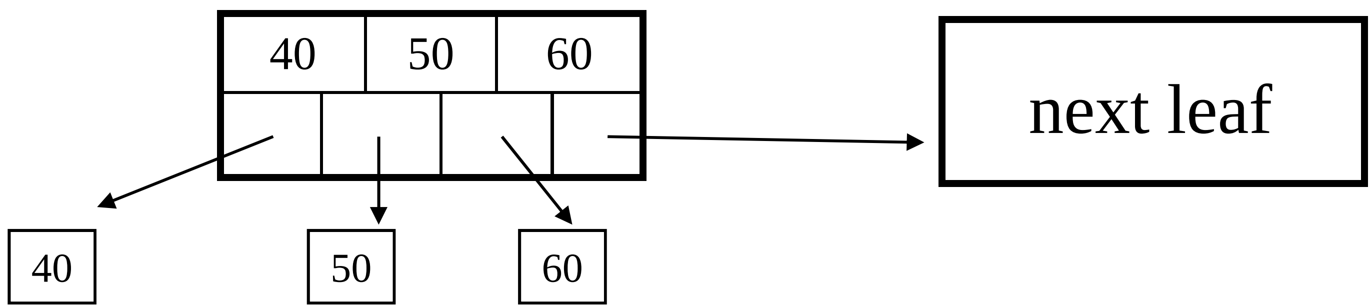
Behind the Scene: And he said Hello!

B+ Trees Basics

- Parameter d = the ***degree***
- Each node has [d, 2d] keys (except root)
  - Internal node:

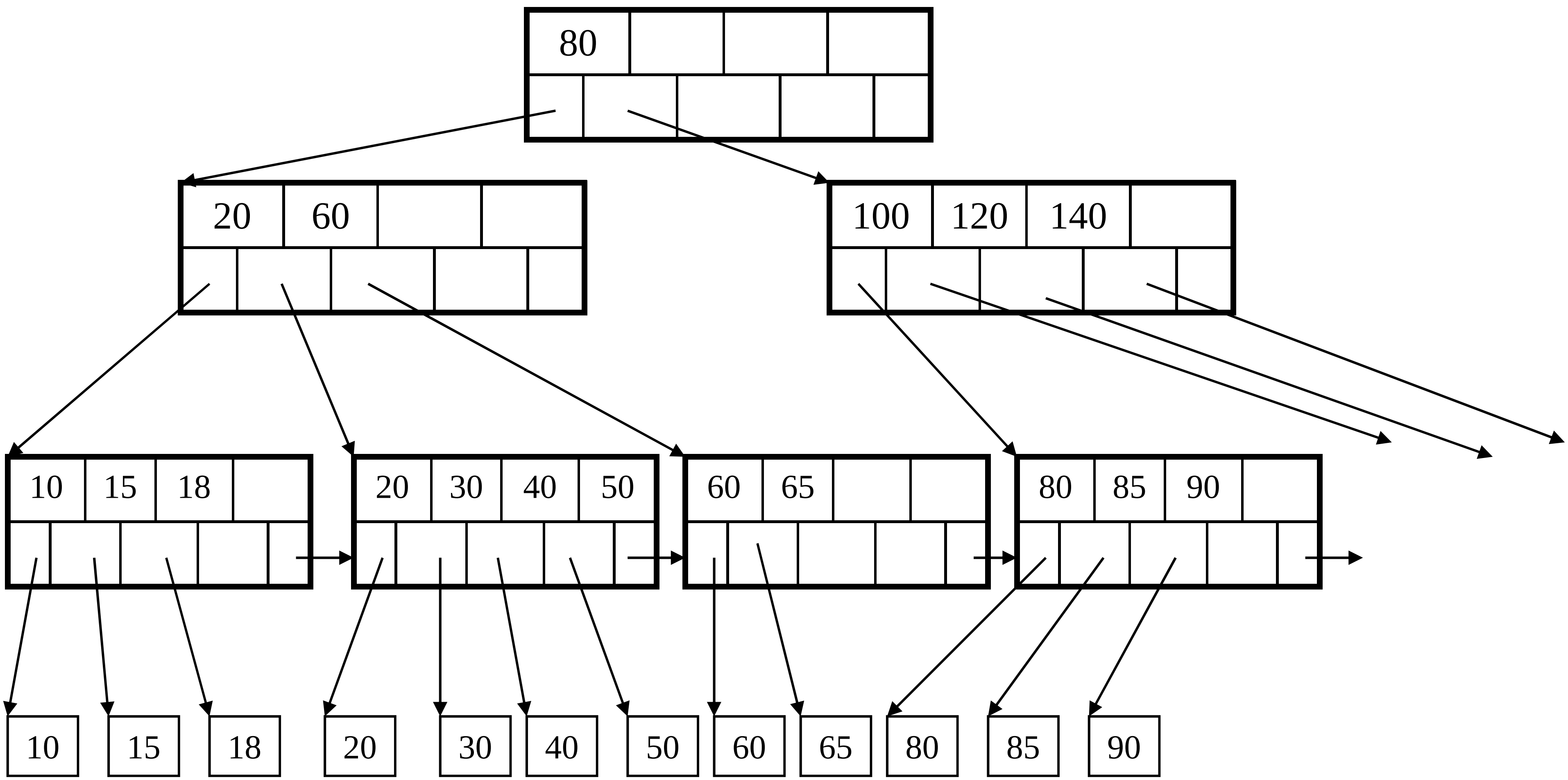


  - Leaf:



B+ Tree Example

d = 2



B+ Tree Design

- How large d ?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 byes
- $2d \times 4 + (2d+1) \times 8 <= 4096$
- $d = 170$

Searching a B+ Tree

- Exact key values:
  - Start at the root
  - Proceed down, to the leaf

```
Select name
From people
Where age = 25
```

- Range queries:
  - As above
  - Then sequential traversal

```
Select name
From people
Where 20 <= age
  and  age <= 30
```

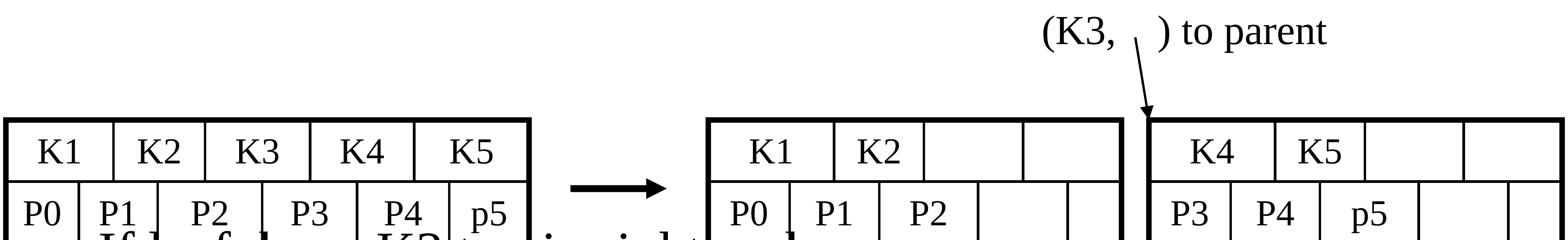
B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 133
- Typical capacities:
  - Height 4: 133^4 = 312,900,700 records
  - Height 3: 133^3 = 2,352,637 records
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes

Insertion in a B+ Tree

Insert (K, P)

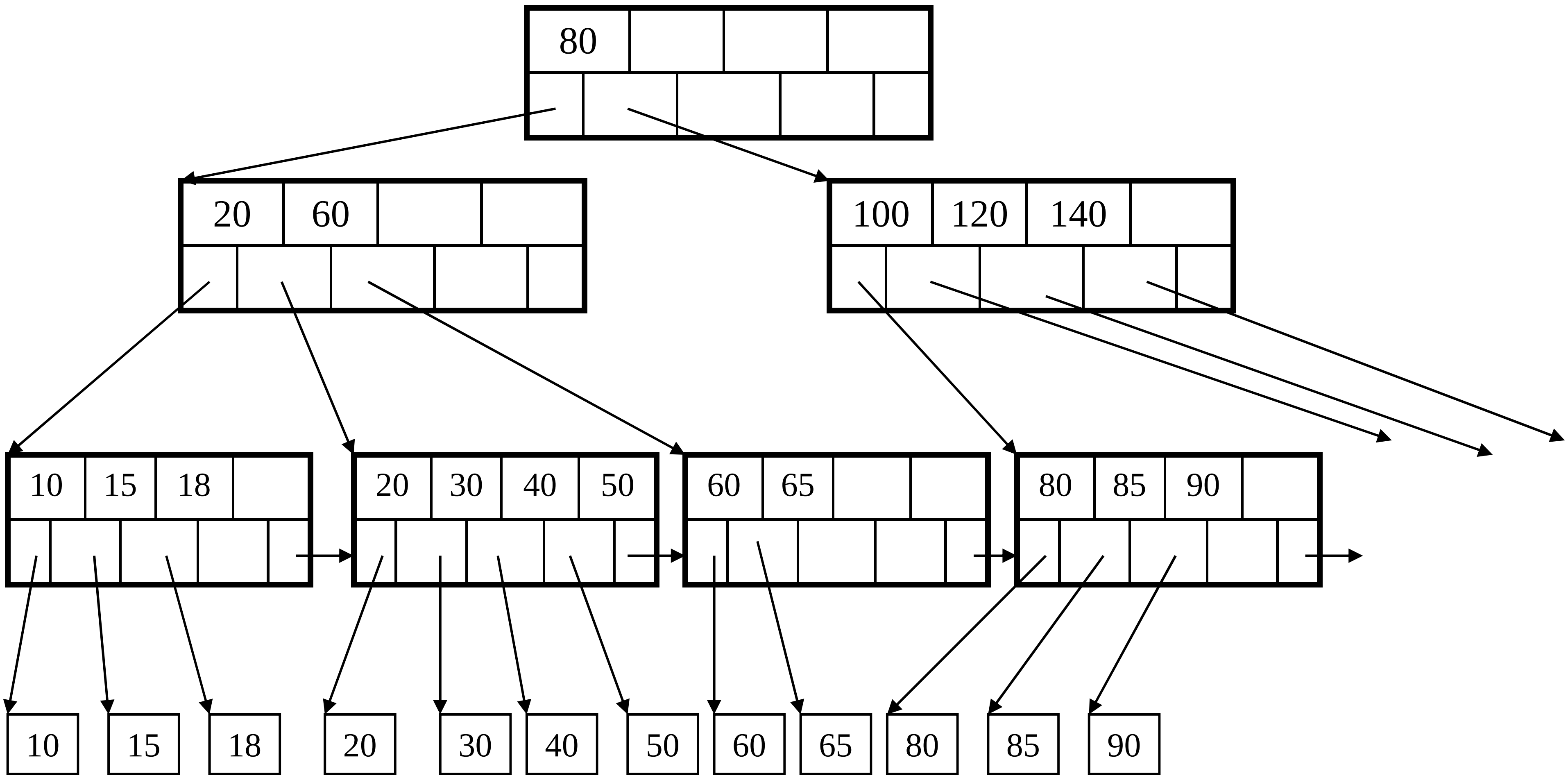
- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:



- If leaf, keep K3 too in right node
- When root splits, new root has 1 key only
  – that’s why root is special for degree satisfaction

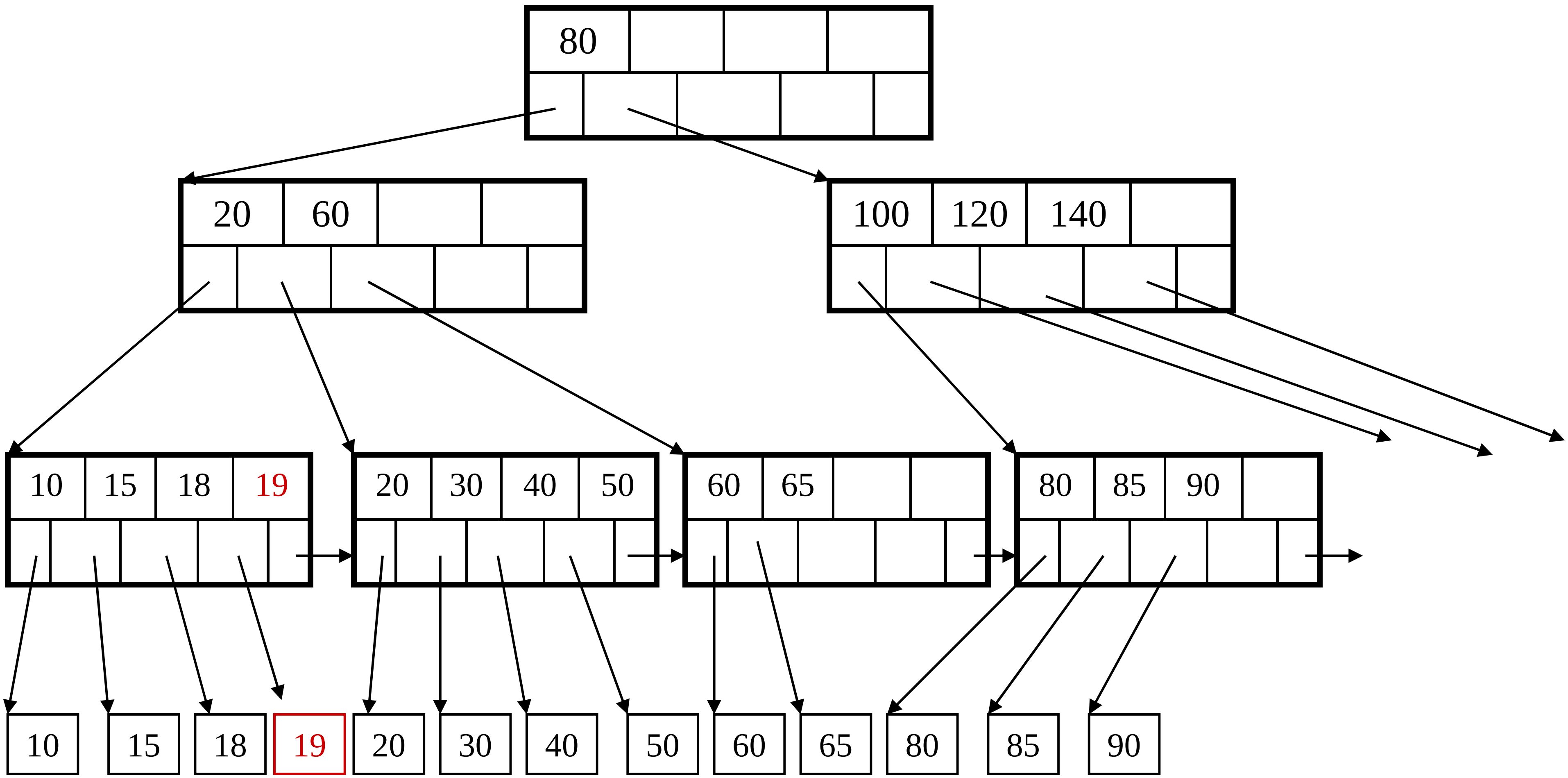
Insertion in a B+ Tree

Insert K=19



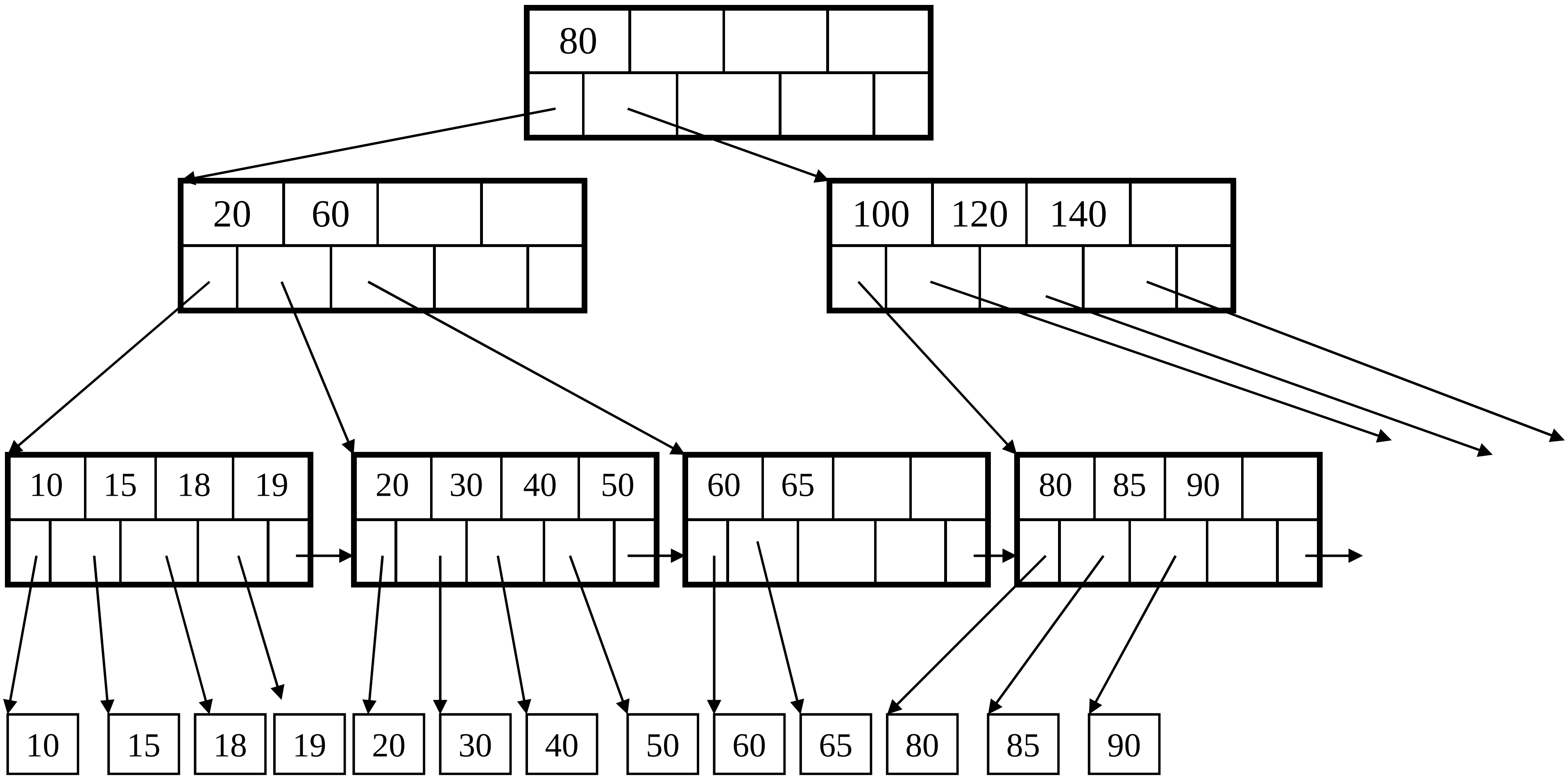
Insertion in a B+ Tree

After insertion



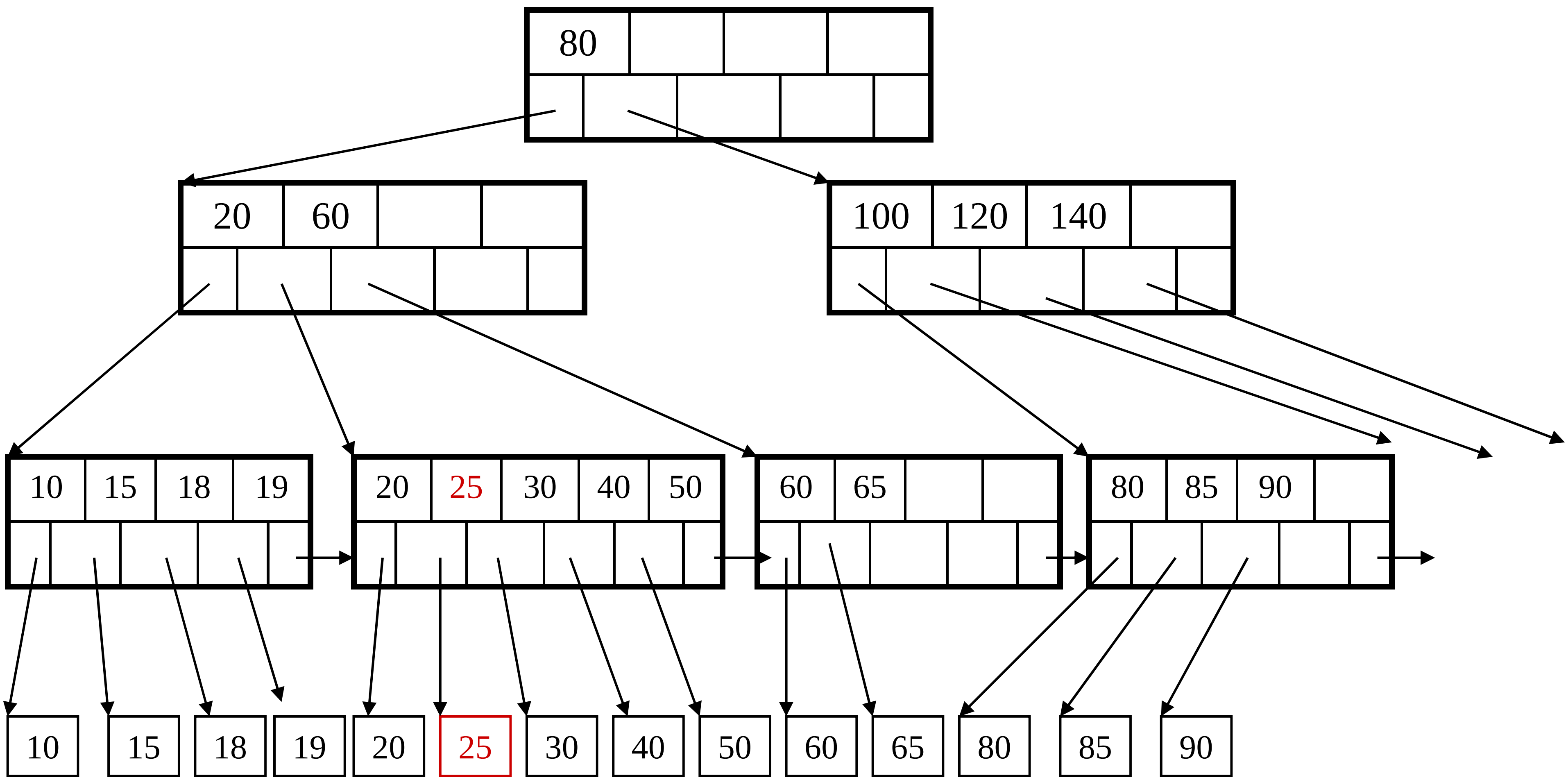
Insertion in a B+ Tree

Now insert 25



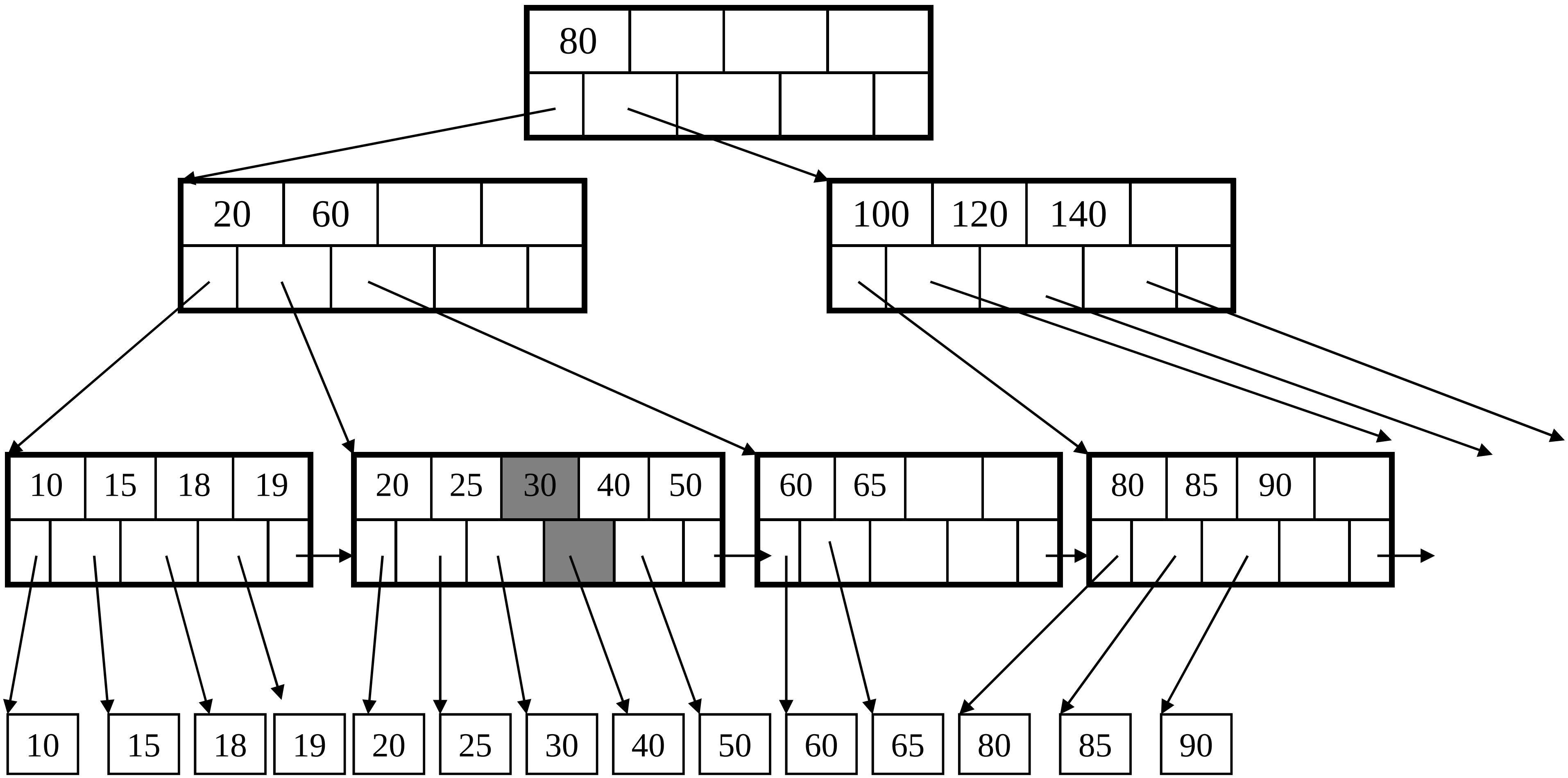
Insertion in a B+ Tree

After insertion



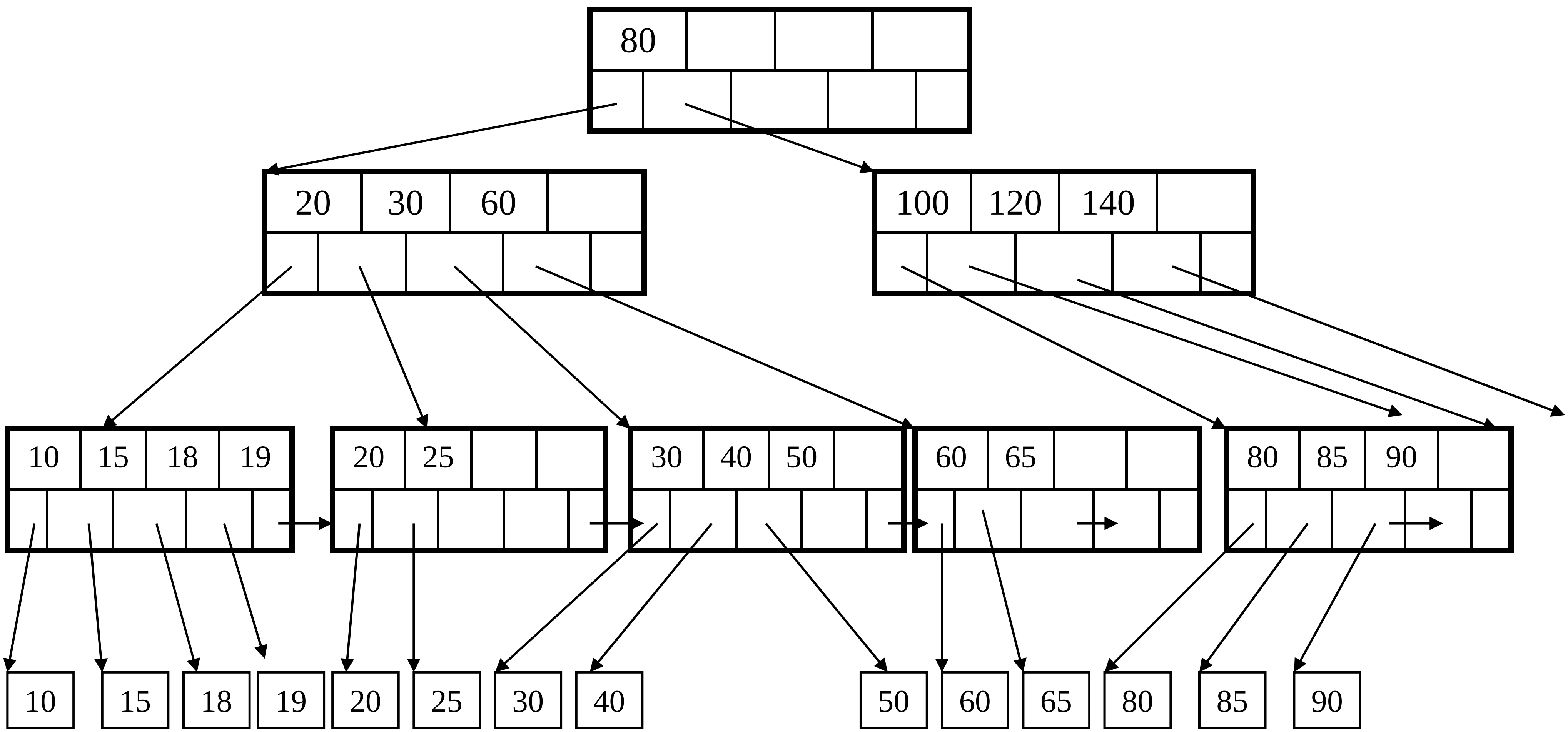
Insertion in a B+ Tree

But now have to split !



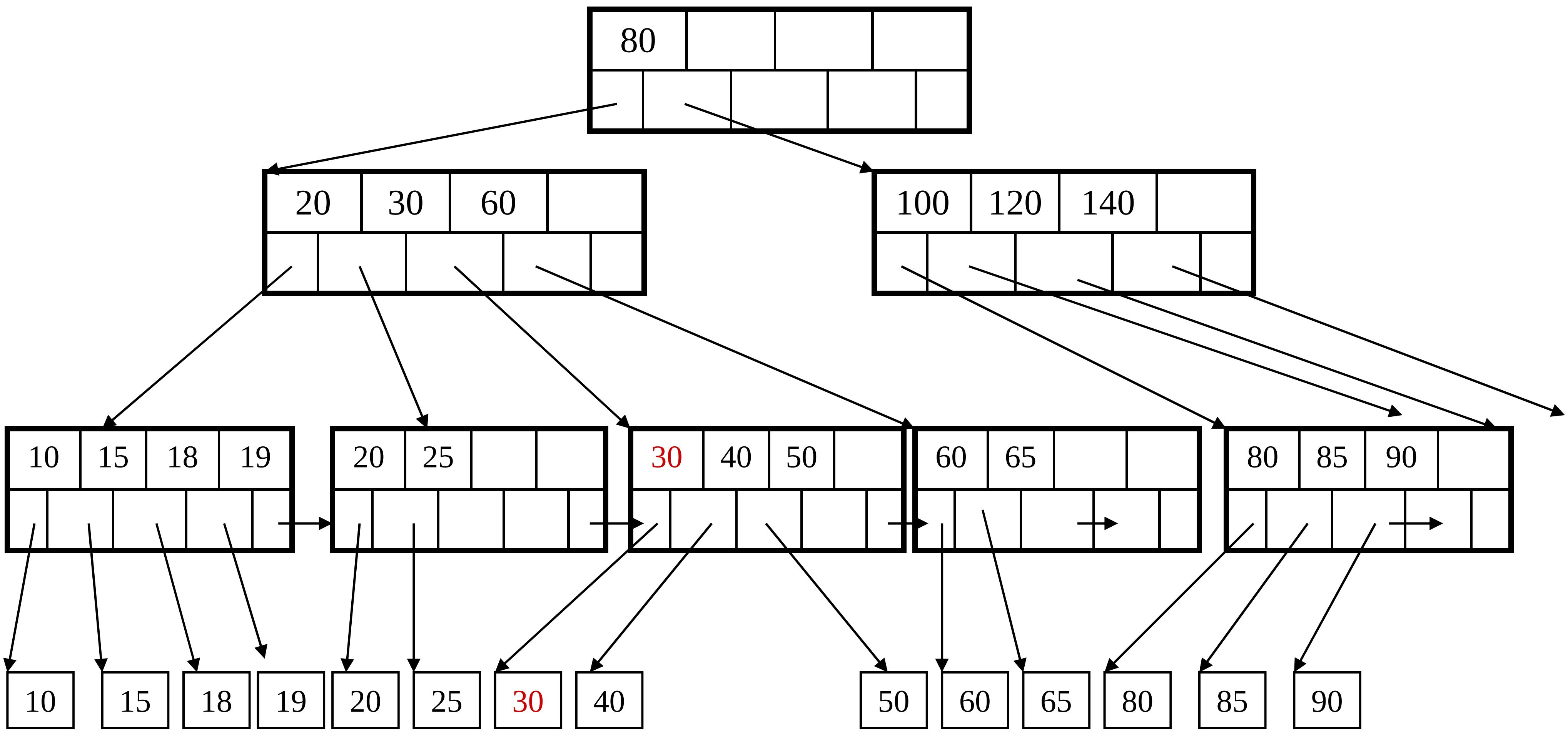
Insertion in a B+ Tree

After the split



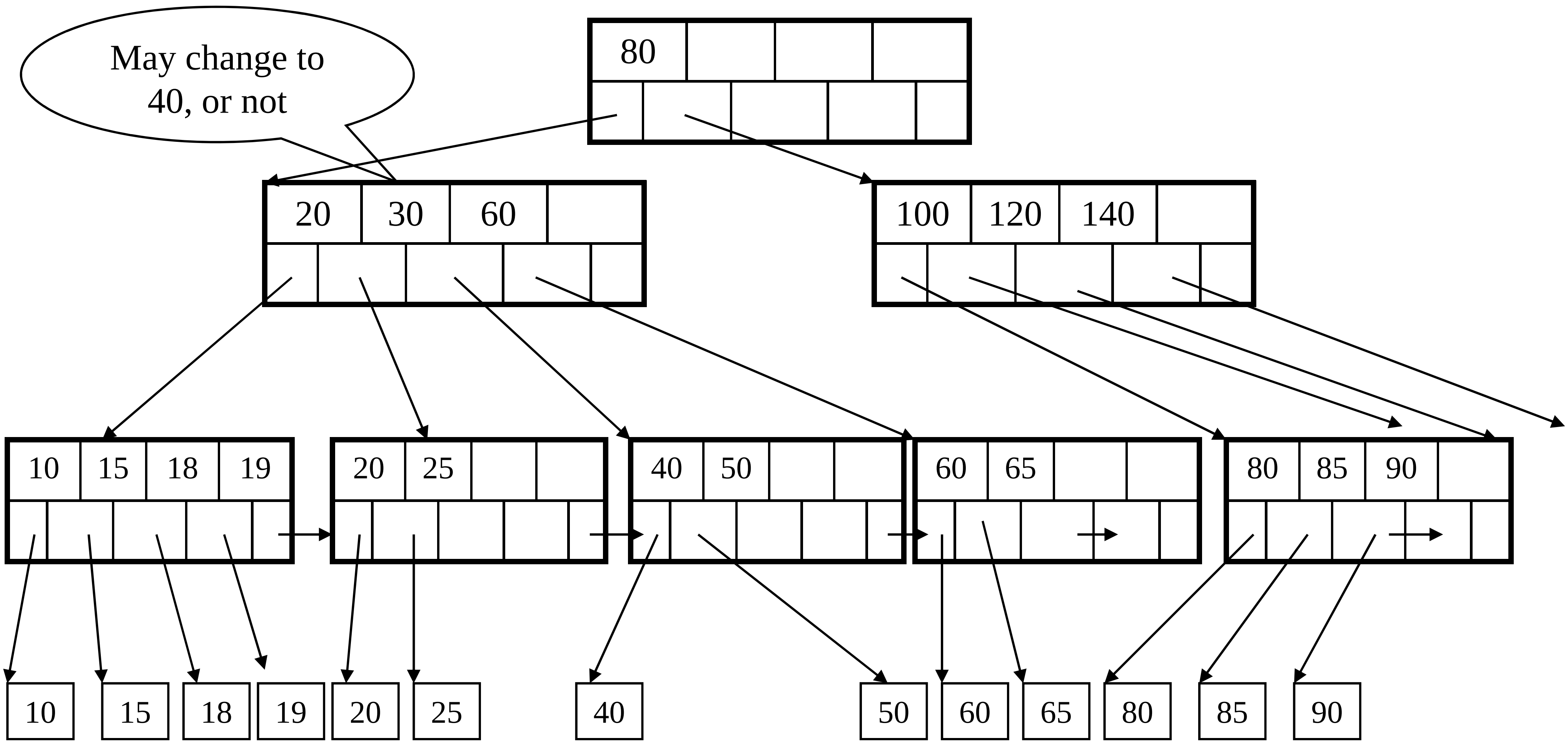
Deletion from a B+ Tree

Delete 30



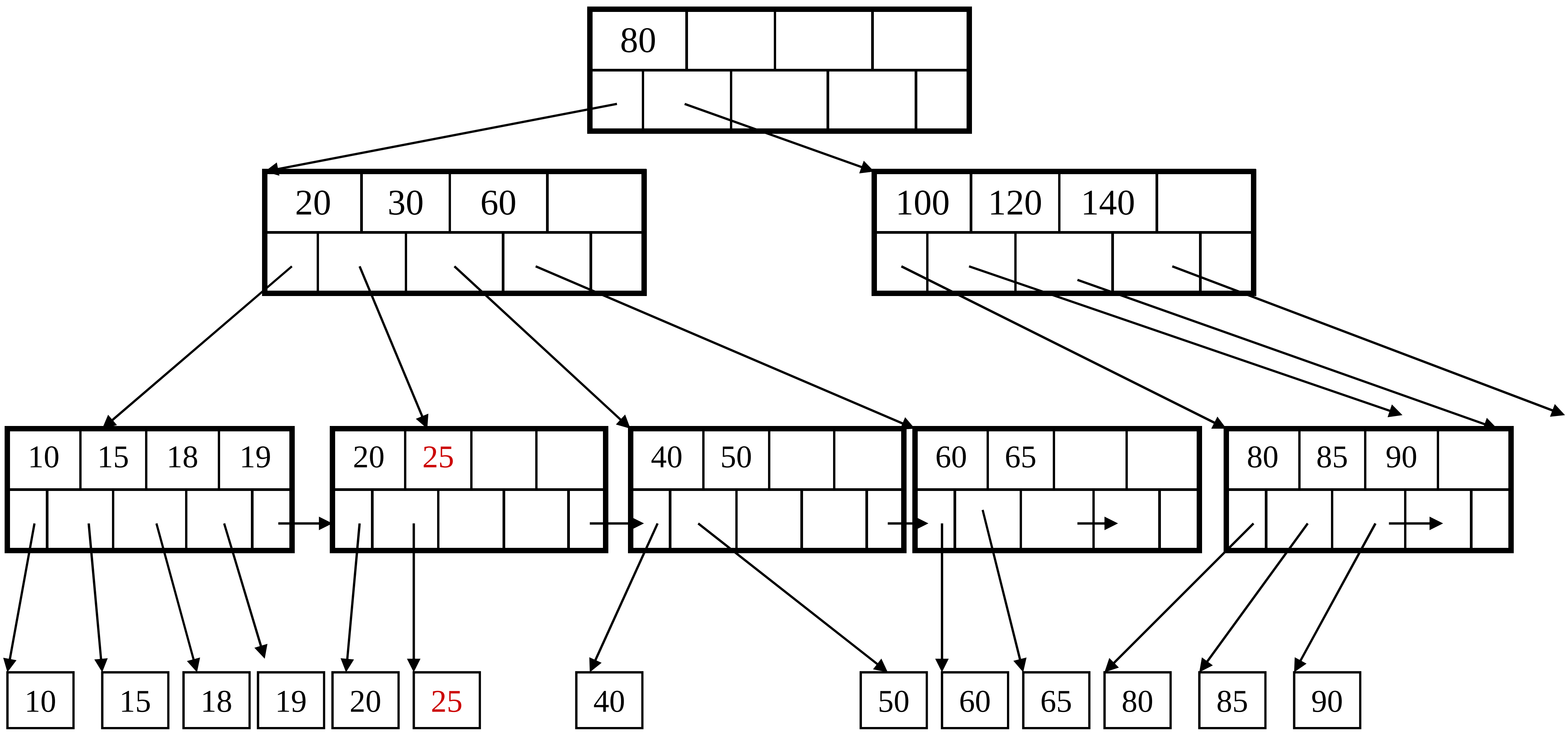
Deletion from a B+ Tree

After deleting 30



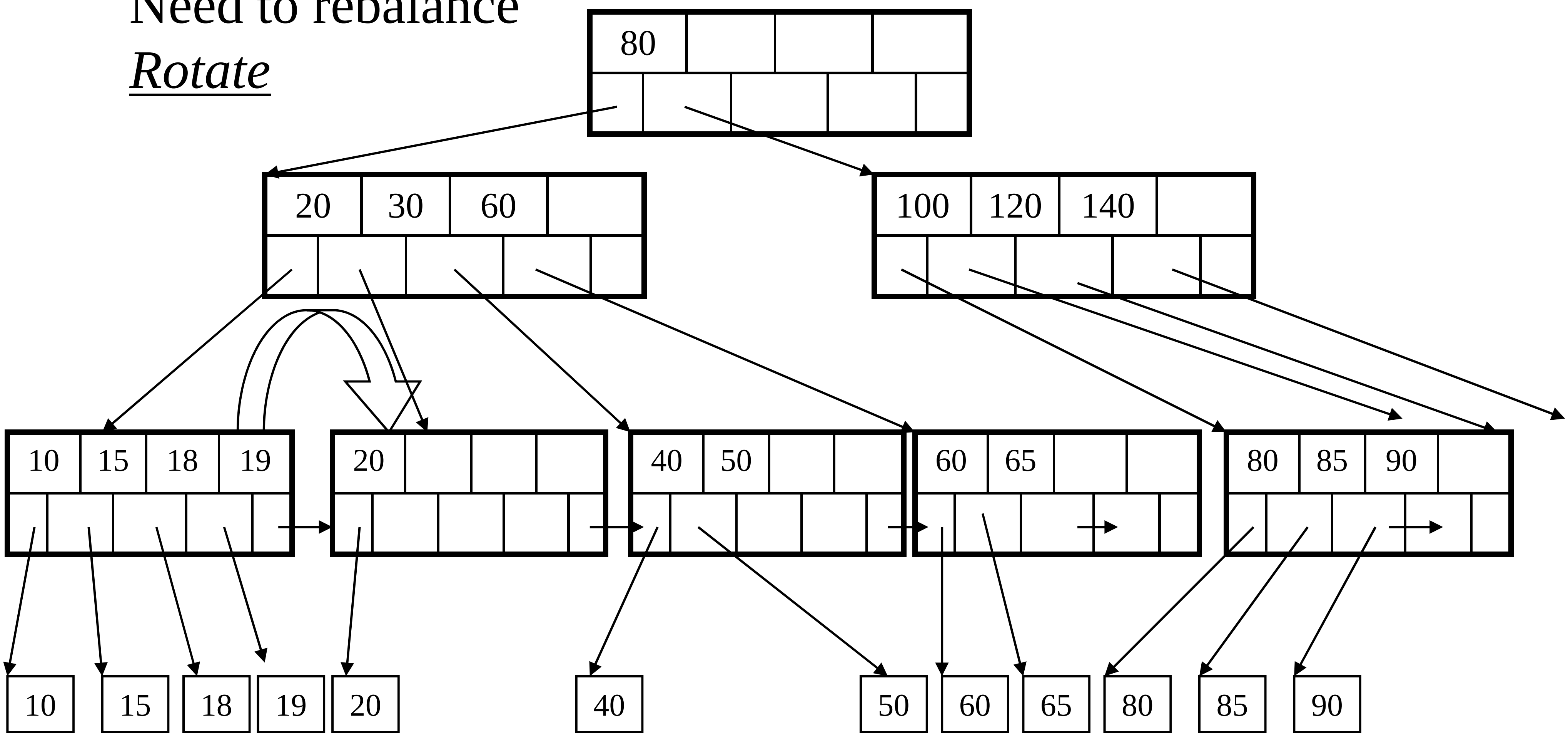
Deletion from a B+ Tree

Now delete 25



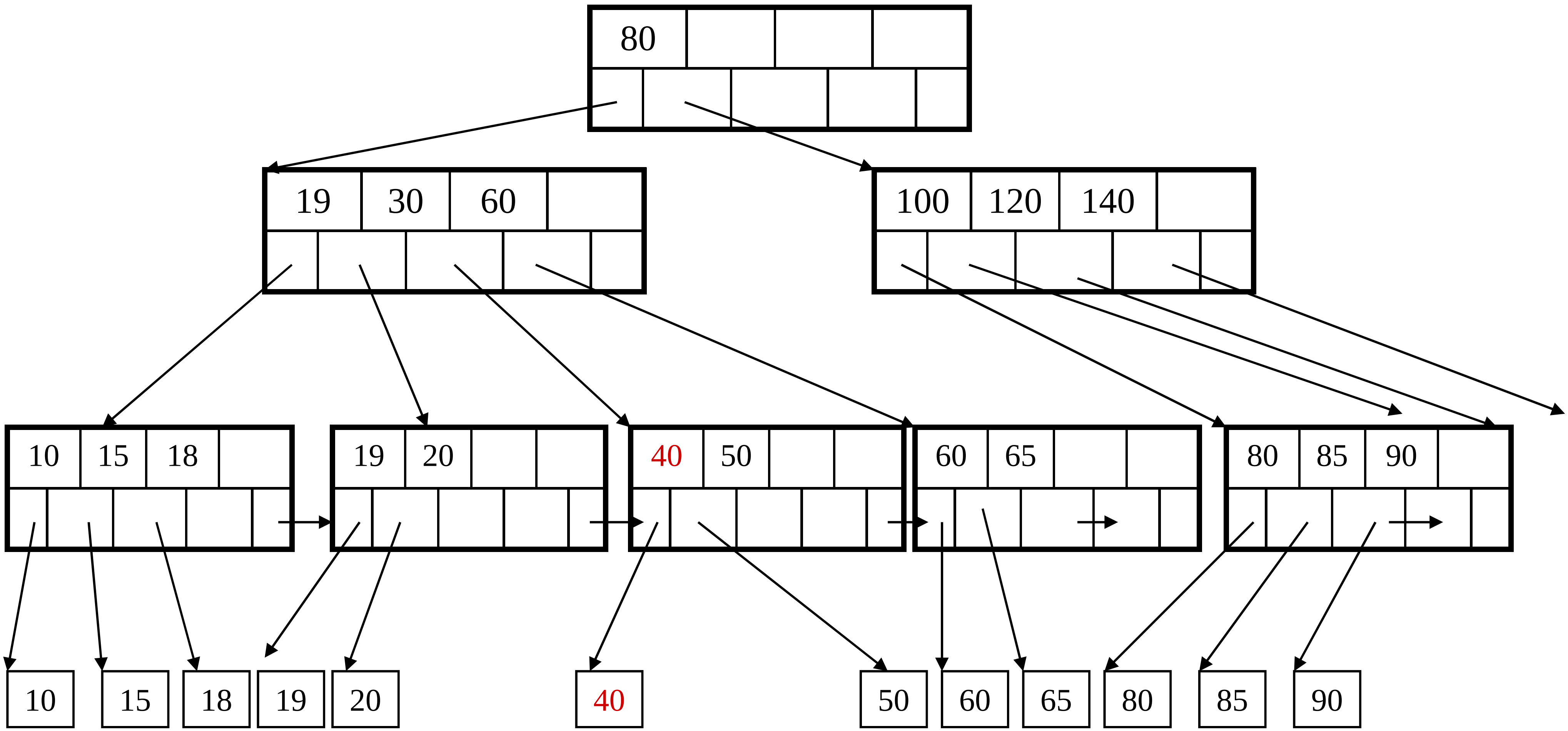
Deletion from a B+ Tree

After deleting 25
Need to rebalance
Rotate



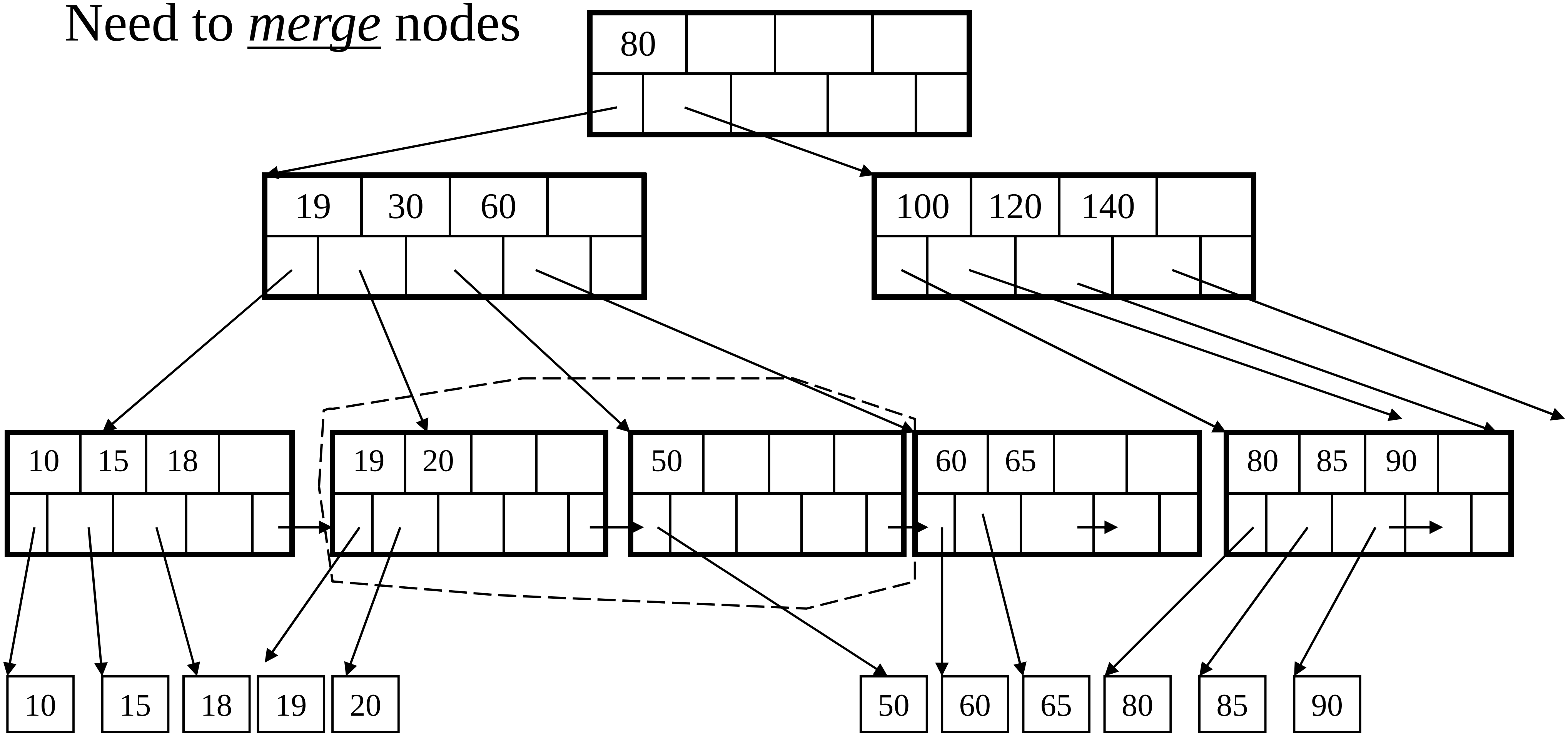
Deletion from a B+ Tree

Now delete 40



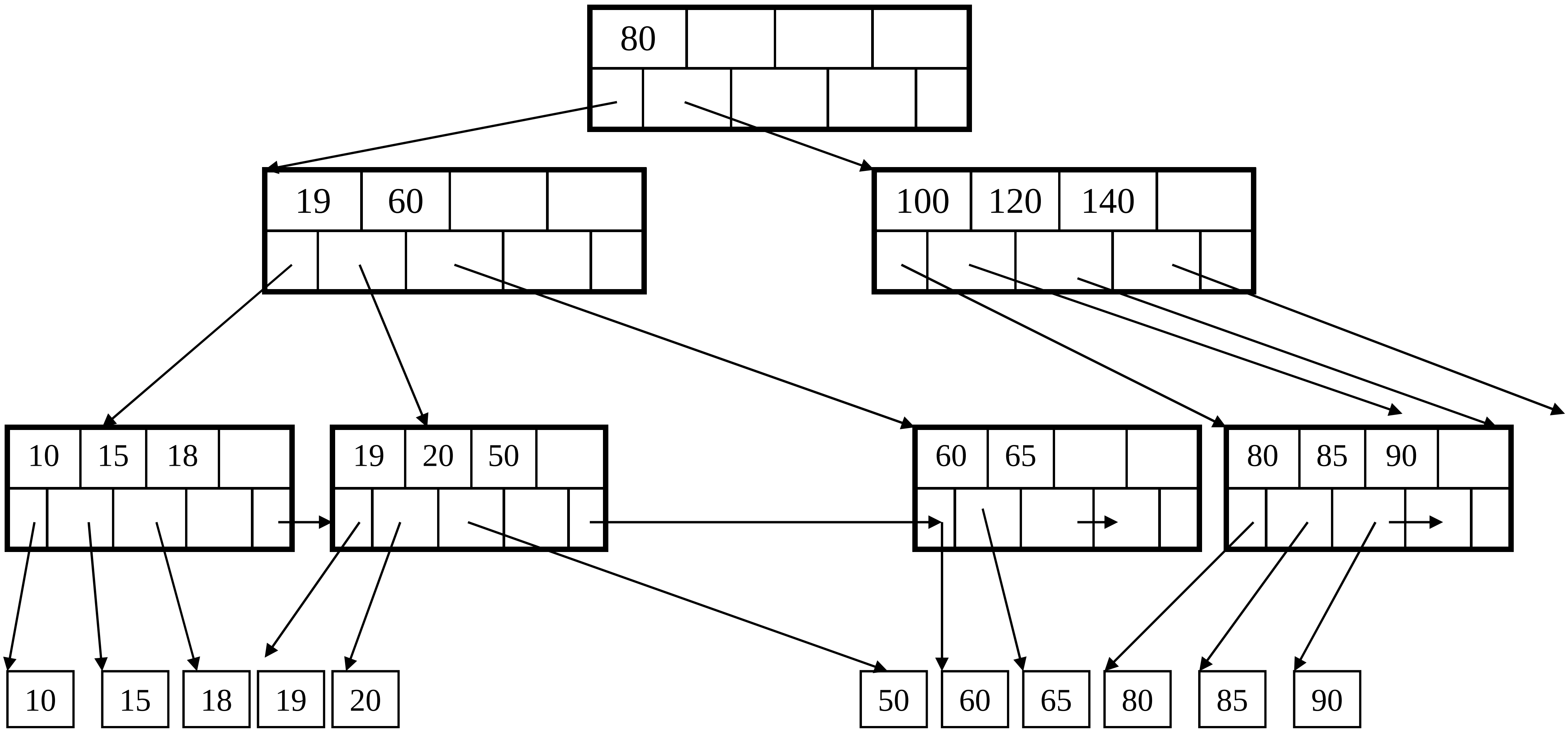
Deletion from a B+ Tree

After deleting 40
Rotation not possible
Need to *merge* nodes



Deletion from a B+ Tree

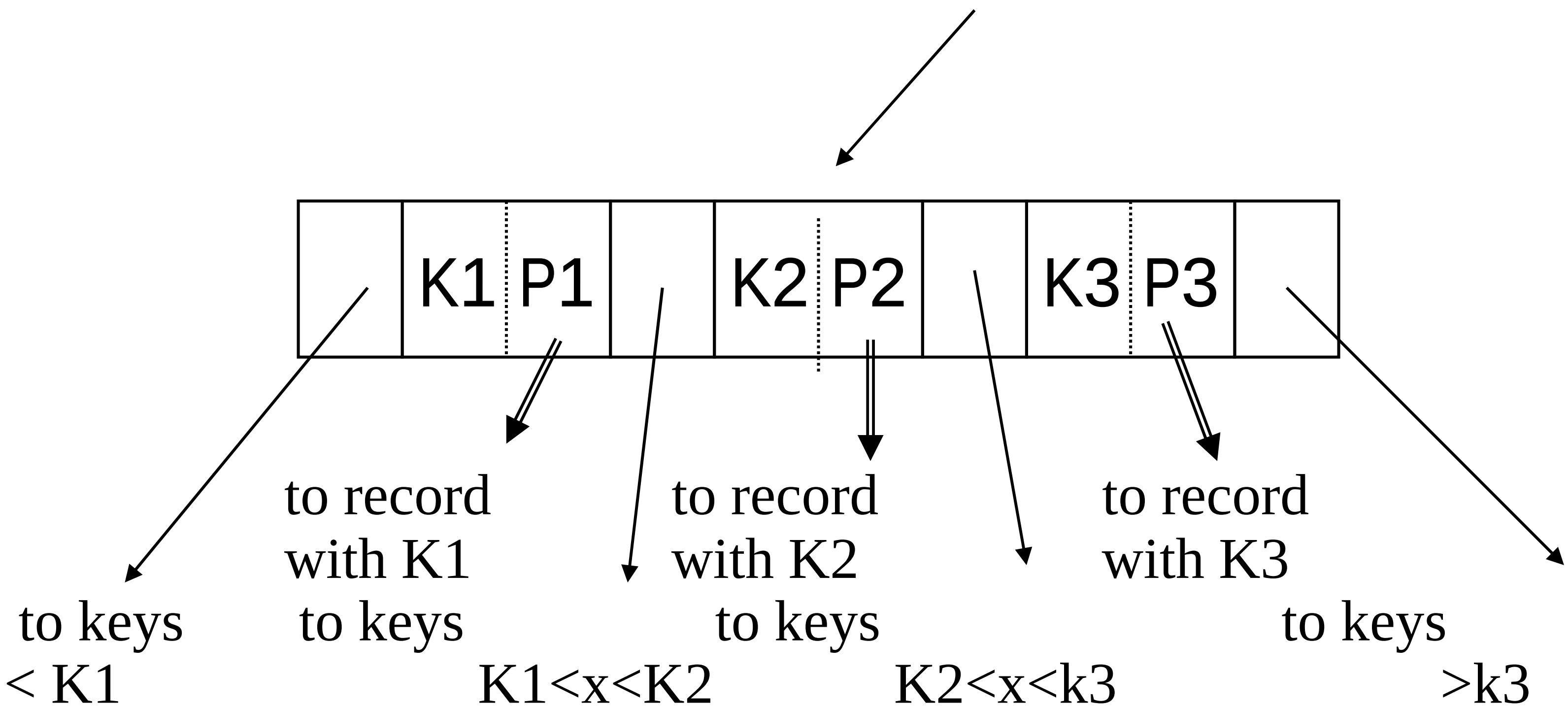
Final tree



Variation on B+tree: B-tree (no +)

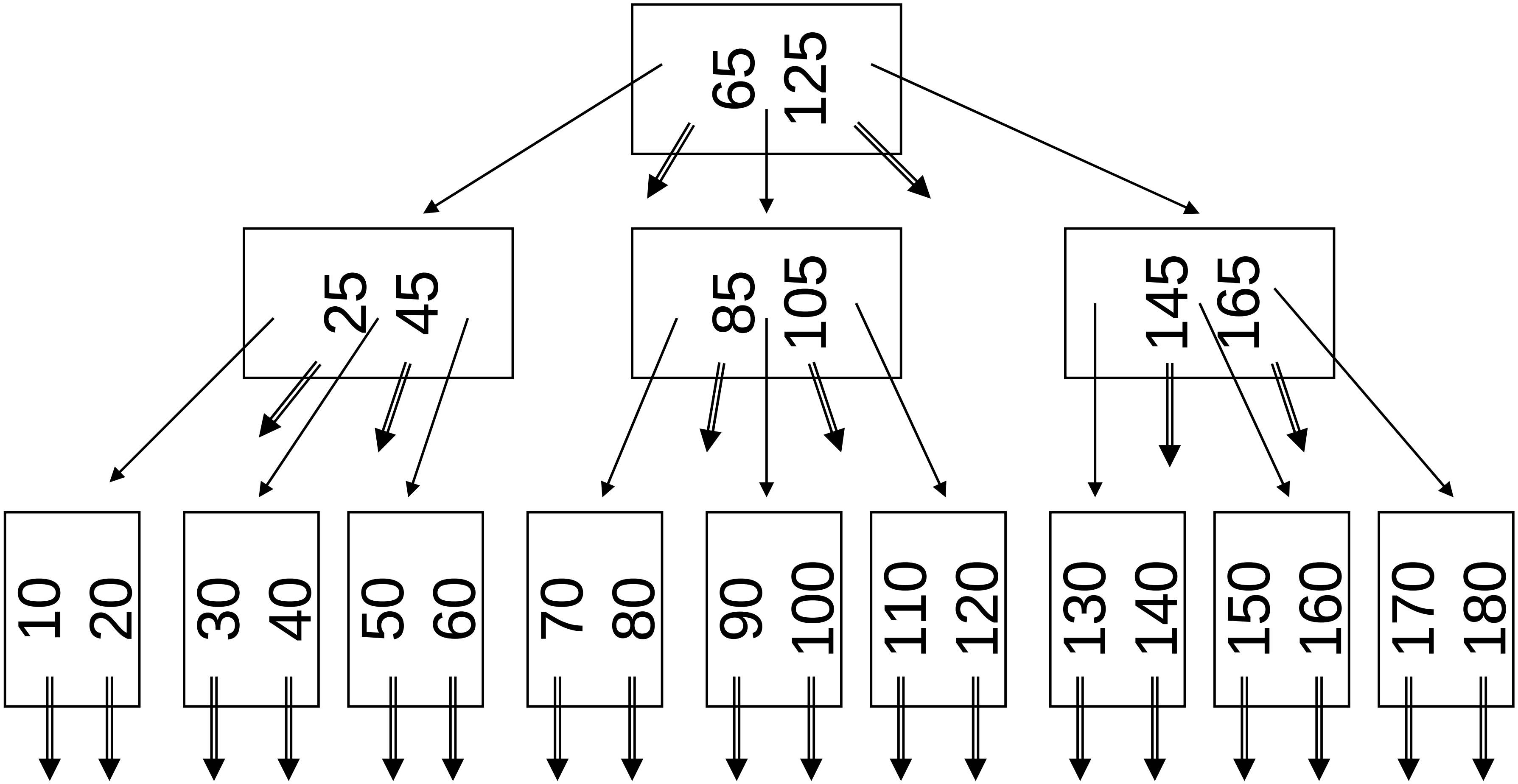
- Idea:
  - Avoid duplicate keys
  - Have record pointers in non-leaf nodes

- Note: Textbook’s B-Tree means B+-tree!


K1 P1
K2 P2
K3 P3
to record with K1
to record with K2
to record with K3
to keys < K1
to keys K1<x<K2
to keys K2<x<k3
to keys >k3

B-tree example n=2

- Sequence pointers not useful now!



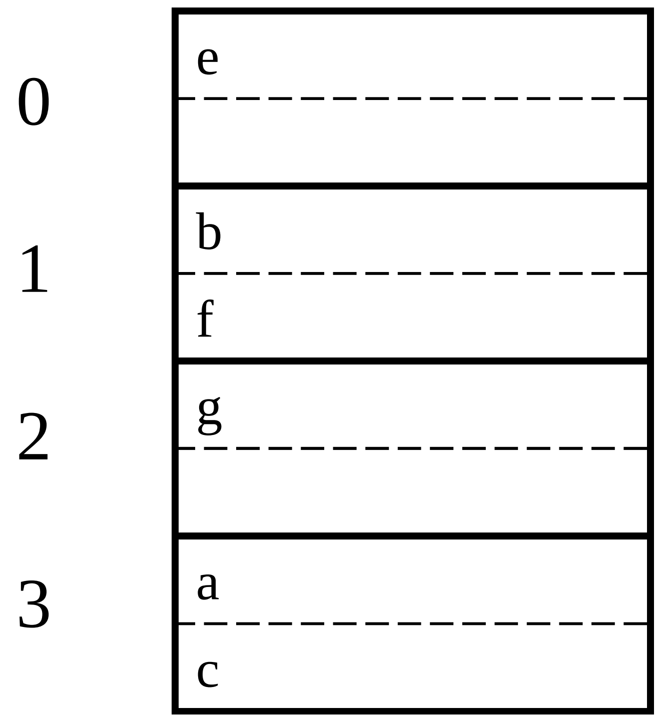
Hash Tables

Hash Tables

- Secondary storage hash tables are much like main memory ones
- Recall basics:
  - There are n *buckets*
  - A hash function f(k) maps a key k to {0, 1, …, n-1}
  - Store in bucket f(k) a pointer to record with key k
- Secondary storage: bucket = block, use overflow blocks when needed

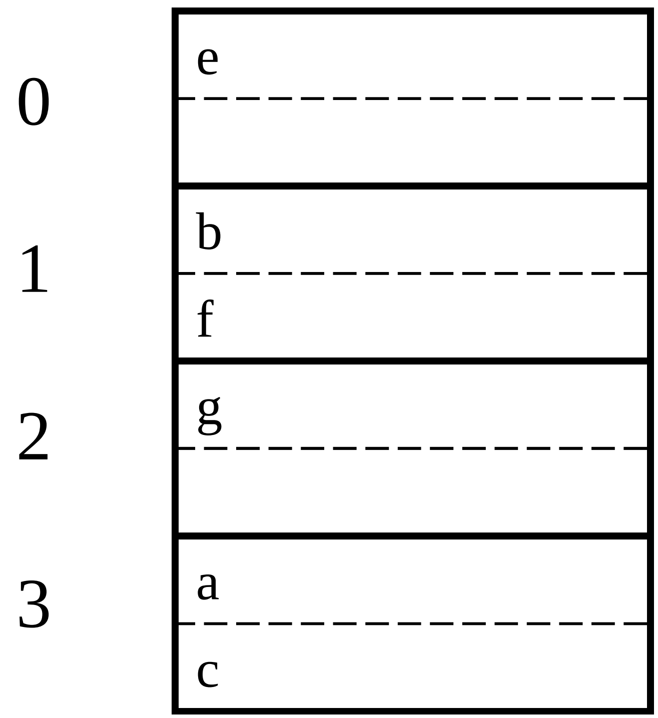
Hash Table Example

- Assume 1 bucket (block) stores 2 keys + pointers
- h(e)=0
- h(b)=h(f)=1
- h(g)=2
- h(a)=h(c)=3



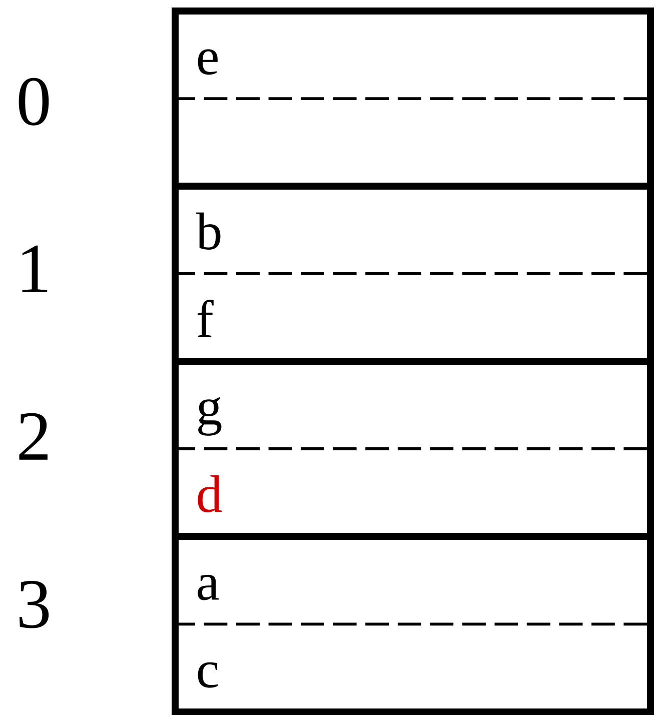
Searching in a Hash Table

- Search for a:
- Compute h(a)=3
- Read bucket 3
- 1 disk access



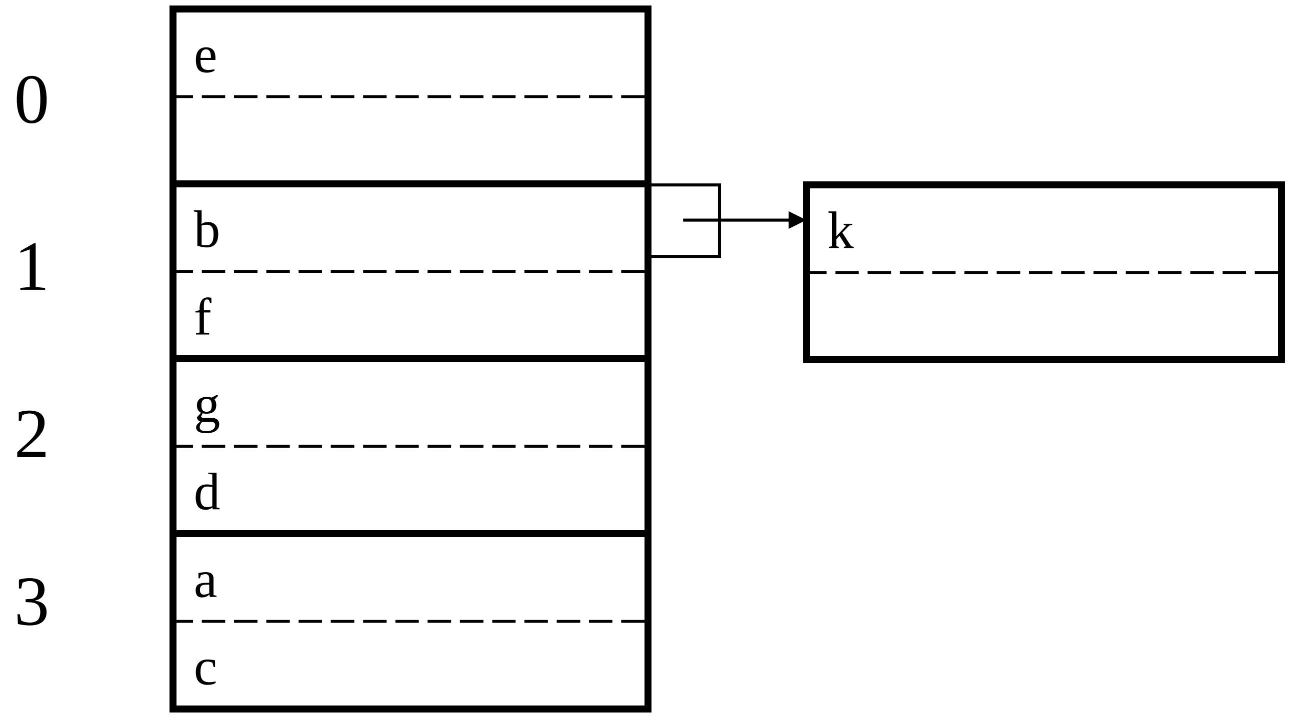
Insertion in Hash Table

- Place in right bucket, if space
- E.g. h(d)=2



Insertion in Hash Table

- Create overflow block, if no space
- E.g. h(k)=1



- More over-flow blocks may be needed

Hash Table Performance

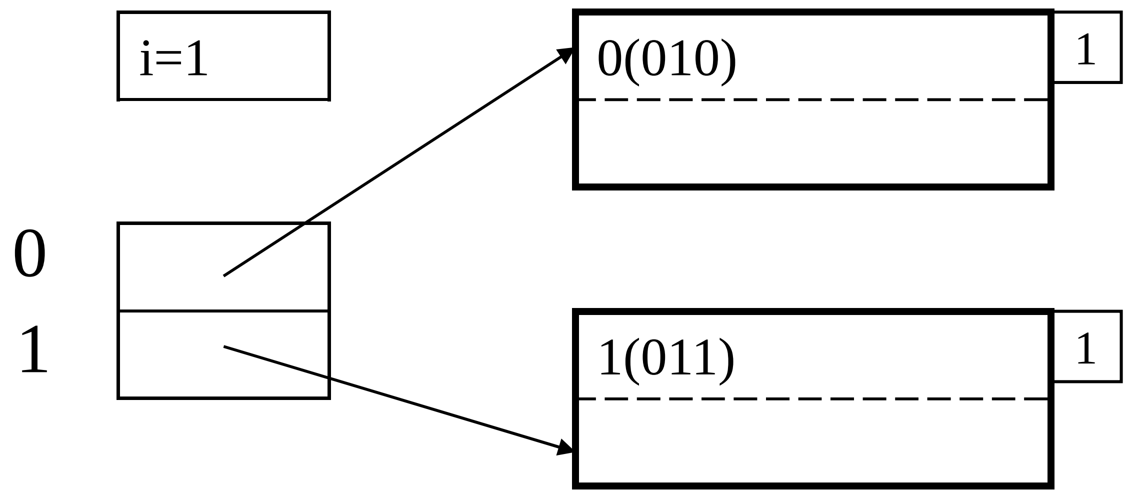
- Excellent, if no overflow blocks
- Degrades considerably when number of keys exceeds the number of buckets (I.e. many overflow blocks).

Extensible Hash Table

- Allows hash table to grow, to avoid performance degradation
- Assume a hash function h that returns numbers in $\{0, \ldots, 2^k - 1\}$
- Start with $n = 2^i << 2^k$, only look at first i most significant bits

Extensible Hash Table

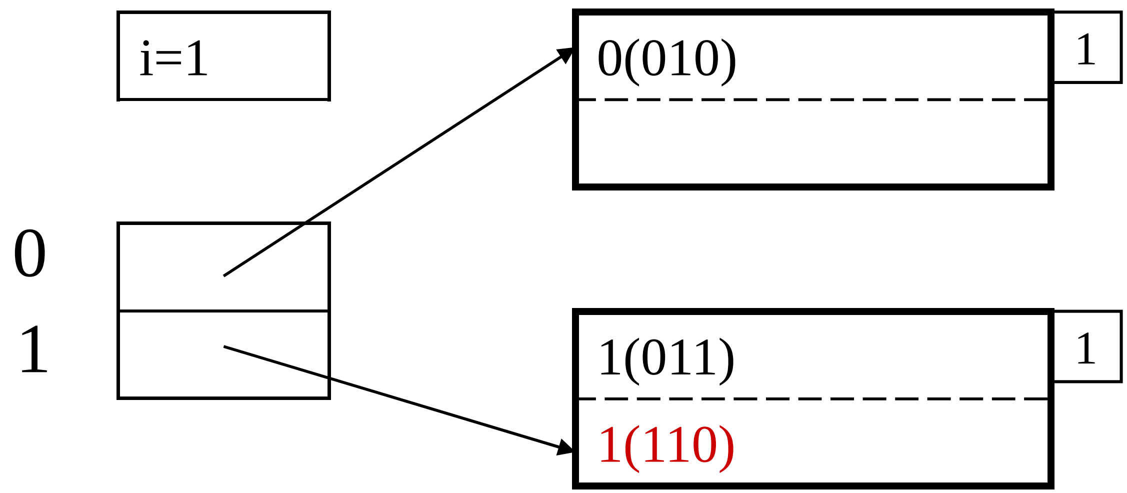
- E.g. i=1, n=2, k=4



- Note: we only look at the first bit (0 or 1)

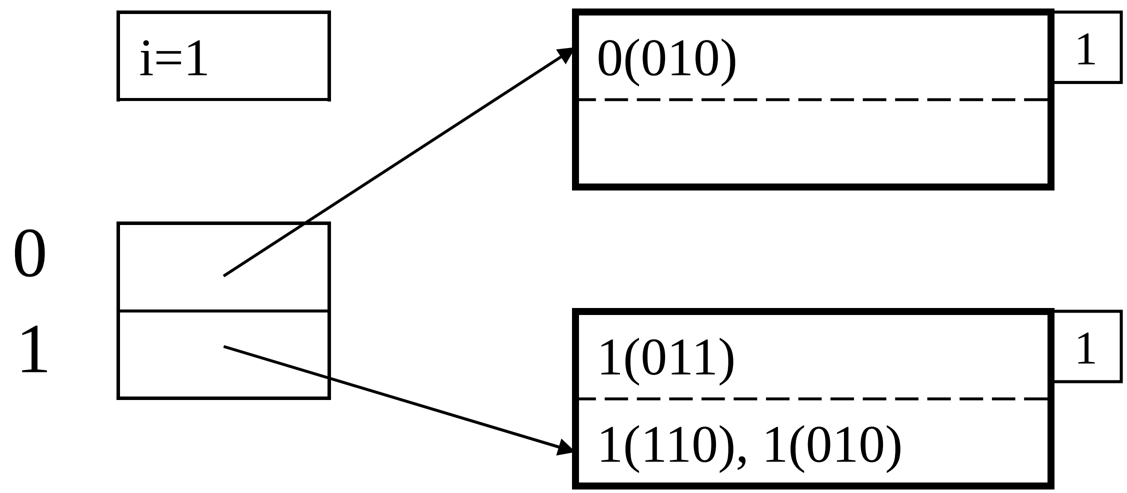
Insertion in Extensible Hash Table

- Insert 1110



Insertion in Extensible Hash Table

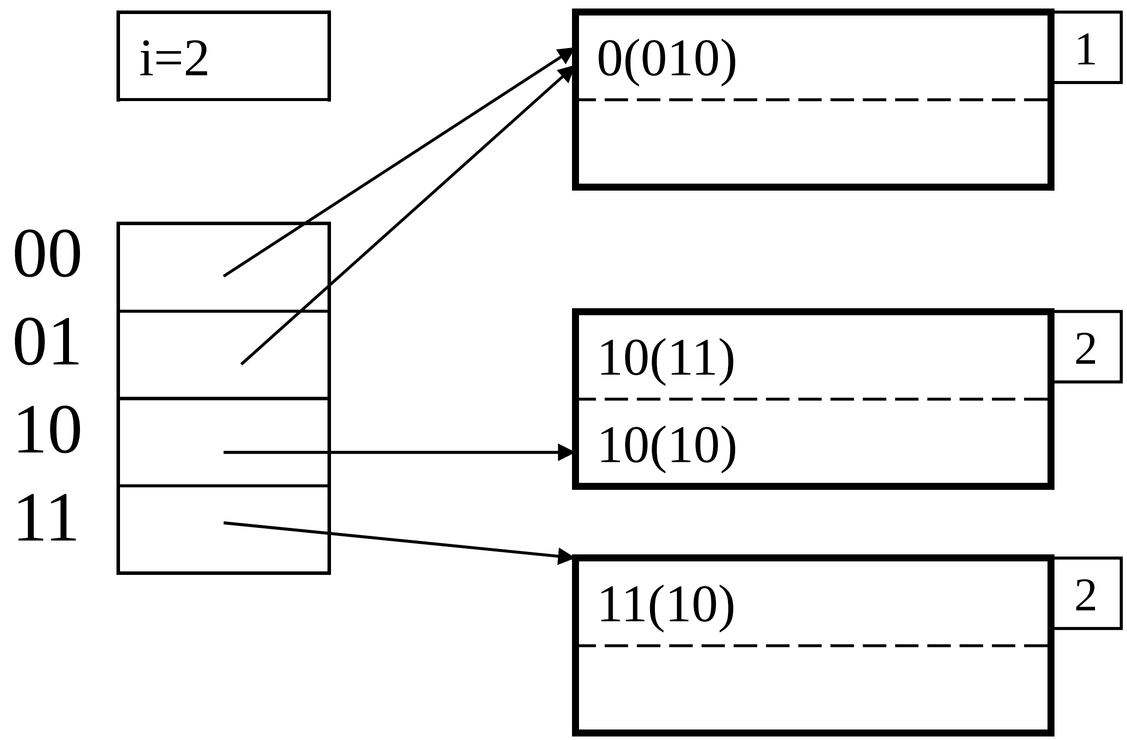
- Now insert 1010



- Need to extend table, split blocks
- i becomes 2

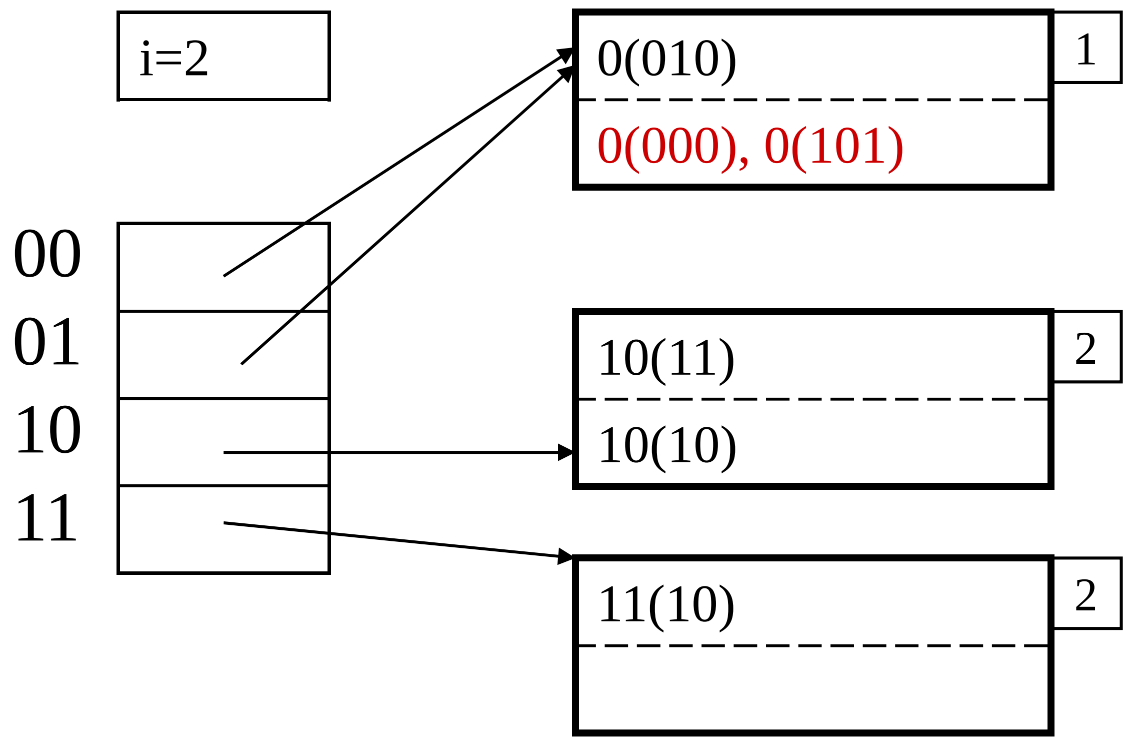
Insertion in Extensible Hash Table

- Now insert 1010 (cont.)



Insertion in Extensible Hash Table

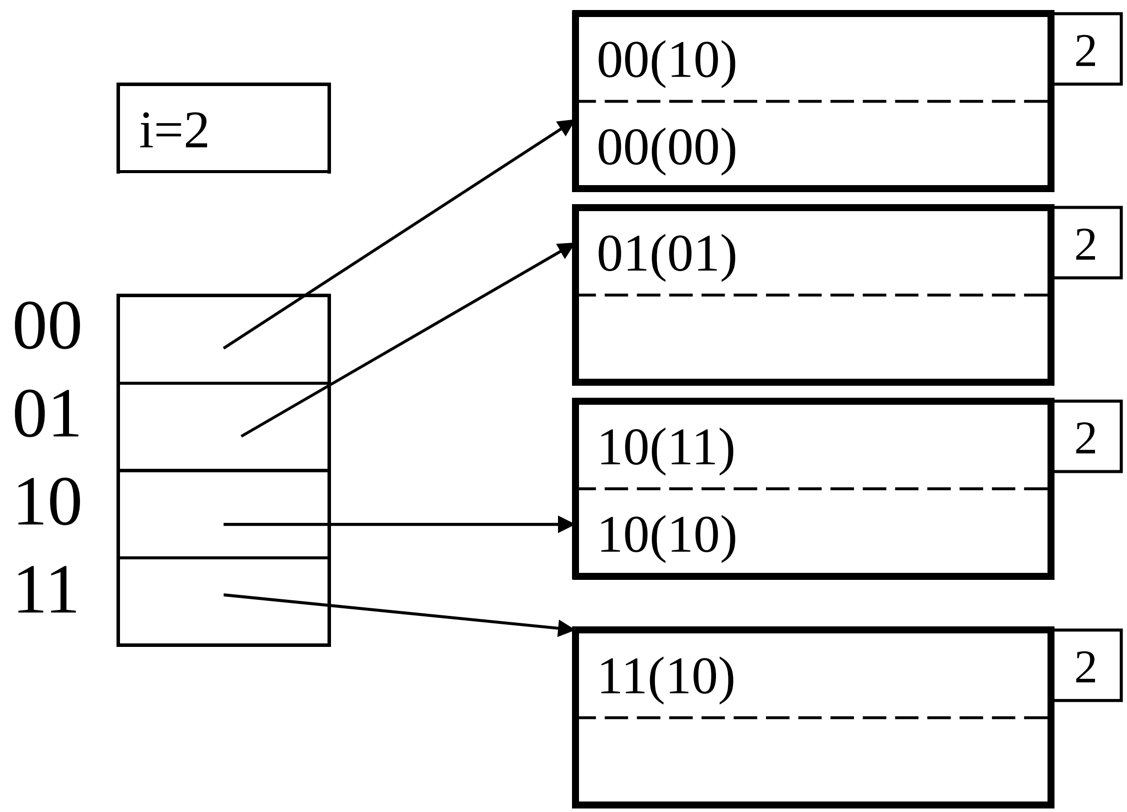
- Now insert 0000, then 0101



- Need to split block

Insertion in Extensible Hash Table

- After splitting the block



Performance Extensible Hash Table

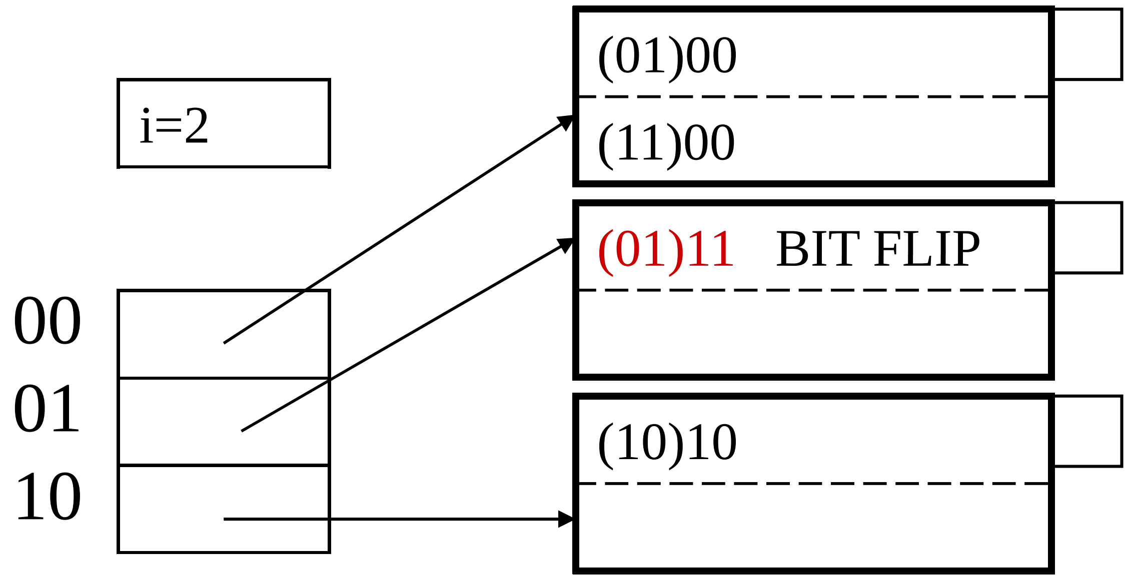
- No overflow blocks: access always one read
- BUT:
  - Extensions can be costly and disruptive
  - After an extension table may no longer fit in memory

Linear Hash Table

- Idea: extend only one entry at a time
- Problem: n= no longer a power of 2
- Let i be #bits necessary to address n buckets.
  - $2^{i-1} < n <= 2^i$
- After computing h(k), use last i bits:
  - If last i bits represent a number >= n, change msb from 1 to 0 (get a number < n)

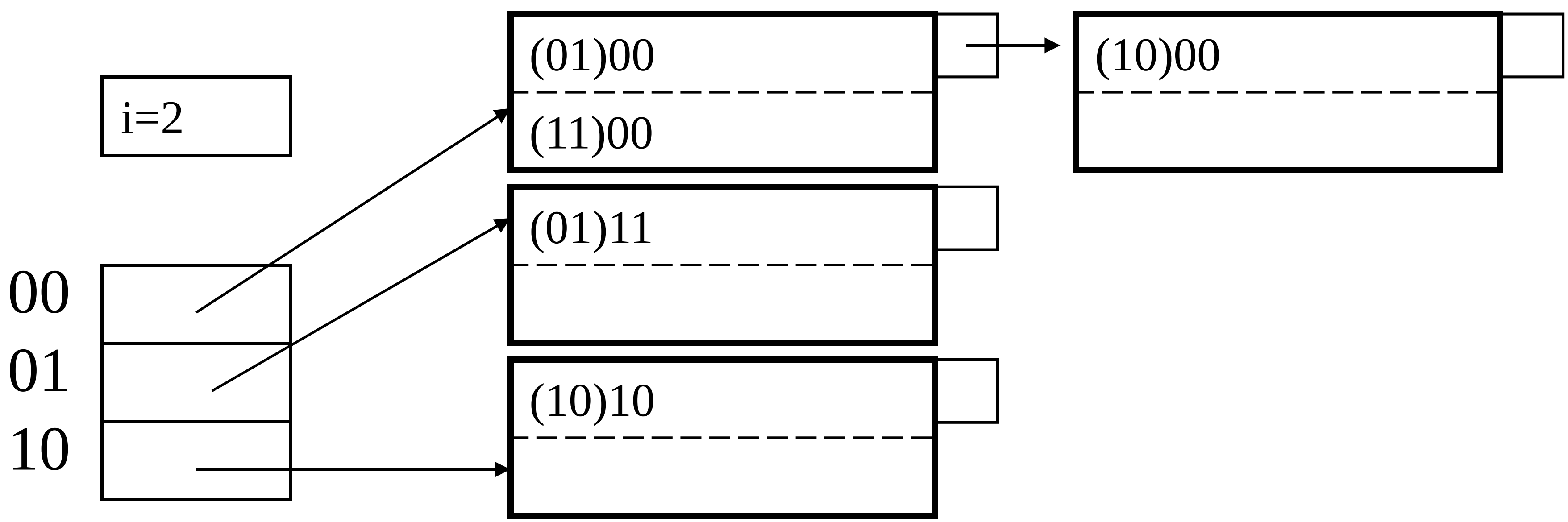
Linear Hash Table Example

- N=3



Linear Hash Table Example

- Insert 1000: overflow blocks…

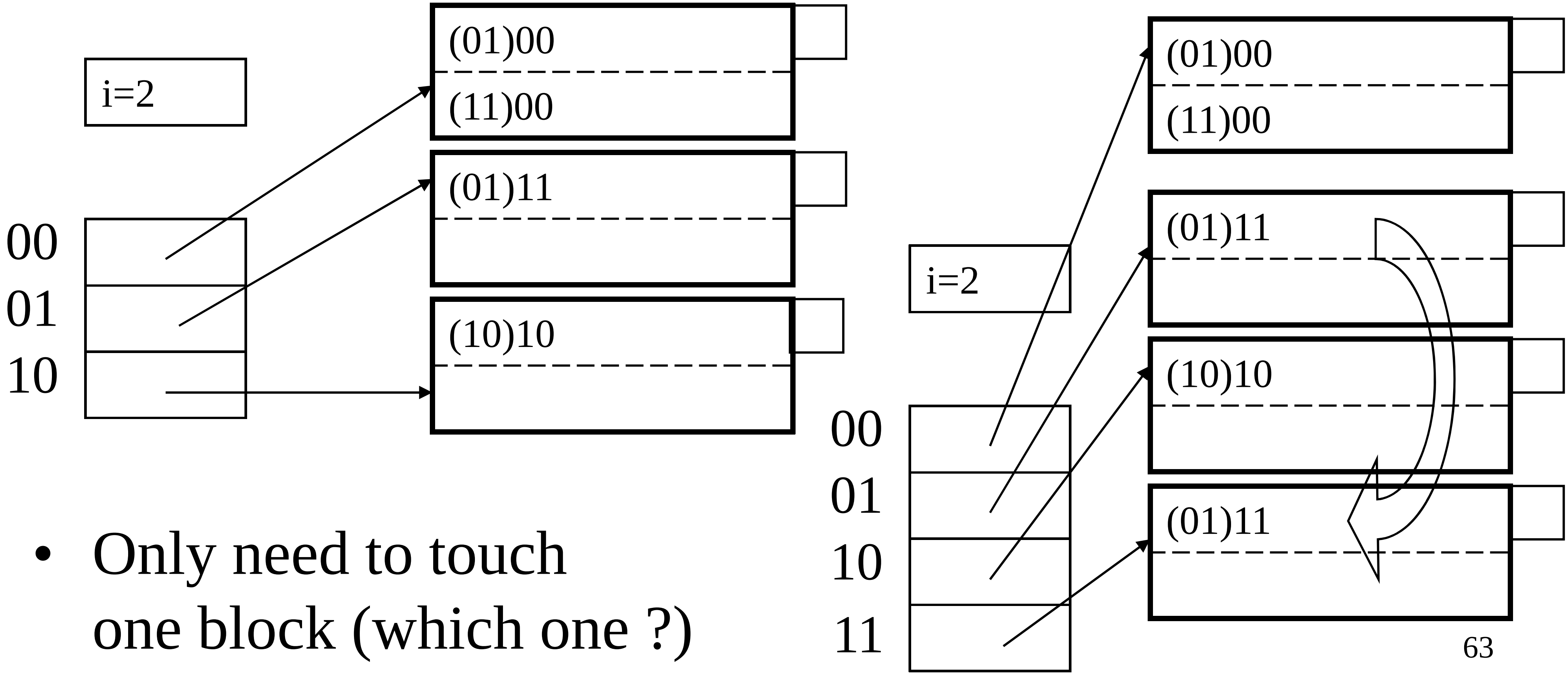


Linear Hash Tables

- Extension: independent on overflow blocks
- Extend n:=n+1 when average number of records per block exceeds (say) 80%

Linear Hash Table Extension

- From n=3 to n=4



- Only need to touch one block (which one ?)

Linear Hash Table Extension

- From n=3 to n=4 finished

- Extension from n=4 to n=5 (new bit)
- Need to touch every single block (why ?)