

Modification of Records

How to handle the following operations on the record level?

1. Insertion
2. Deletion
3. Update

1. Insertion

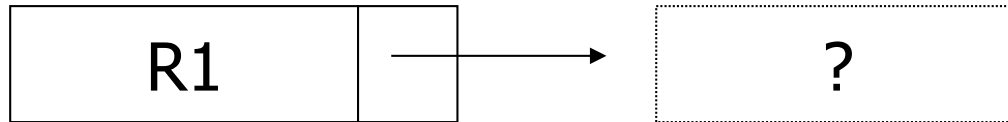
- **Easy case:** records not in sequence
 - Insert new record at end of file
 - If records are fixed-length, insert new record in deleted slot
- **Difficult case:** records are sorted
 - Find position and slide following records
 - If records are sequenced by linking, insert overflow blocks

2. Deletion

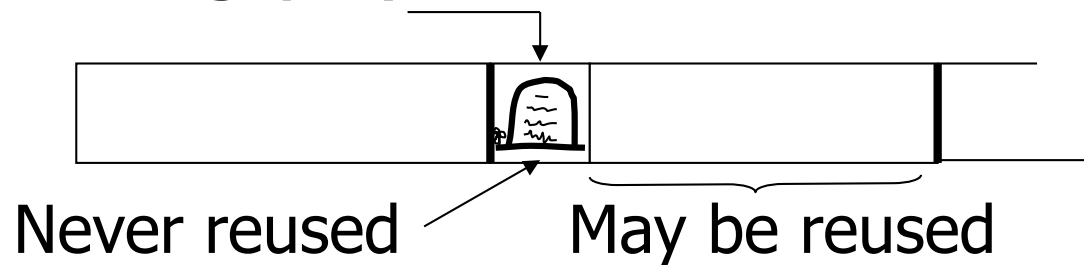
- a. Immediately reclaim space by shifting other records or removing overflows
- b. Mark deleted and list as free for re-use
 - Tradeoffs:
 - How expensive is immediate reclaim?
 - How much space is wasted?

Problem with Deletion


- Dangling pointers:



- When using physical addresses:



- When using logical addresses:

ID	LOC
7788	

Never reuse
ID 7788 nor
space in the map

3. Update

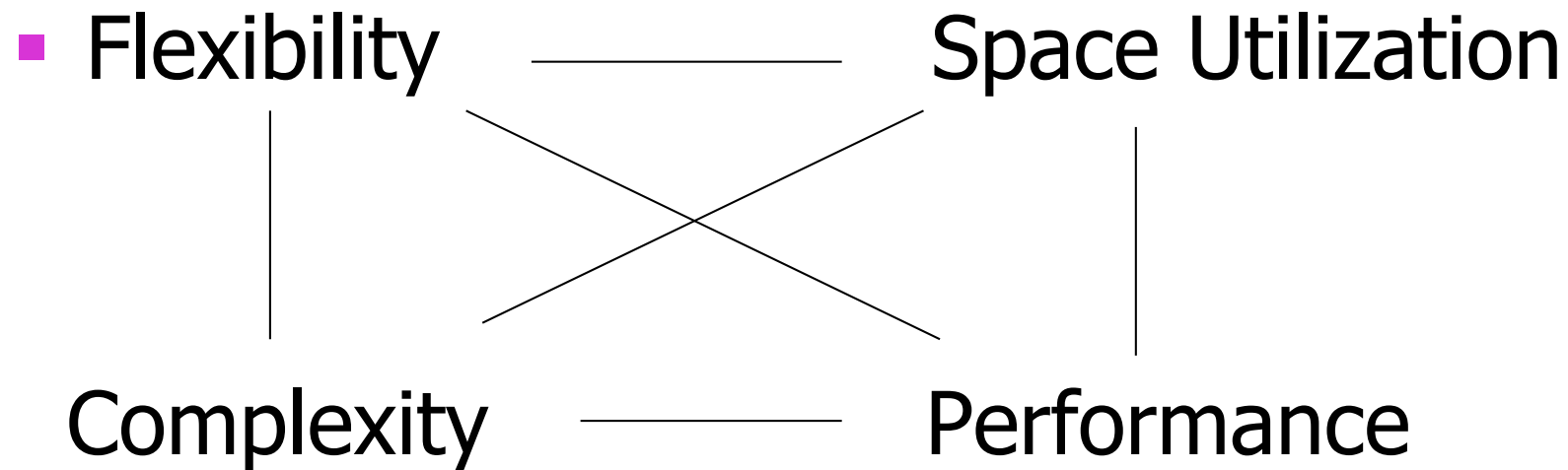
- If records are fixed-length and the order is not affected:
 - Fetch the record, modify it, write it back
- Otherwise:
 - Delete the old record
 - Insert the new record overwriting the tombstones from the deletion

Pointer Swizzling

- *Swizzling* = replacement of physical addresses by memory addresses when loading blocks into memory
- **Automatic Swizzling:** swizzle all addresses when loading a block (need to swizzle all pointer from and to the block)
- **Swizzling on Demand:** use addresses which are invalid as memory addresses

Data Organization

- There are millions of ways to organize the data on disk



Summary 9

More things you should know:

- Memory Hierarchy
- Storage on harddisks
- Values, Records, Blocks, Files
- Storing and modifying records

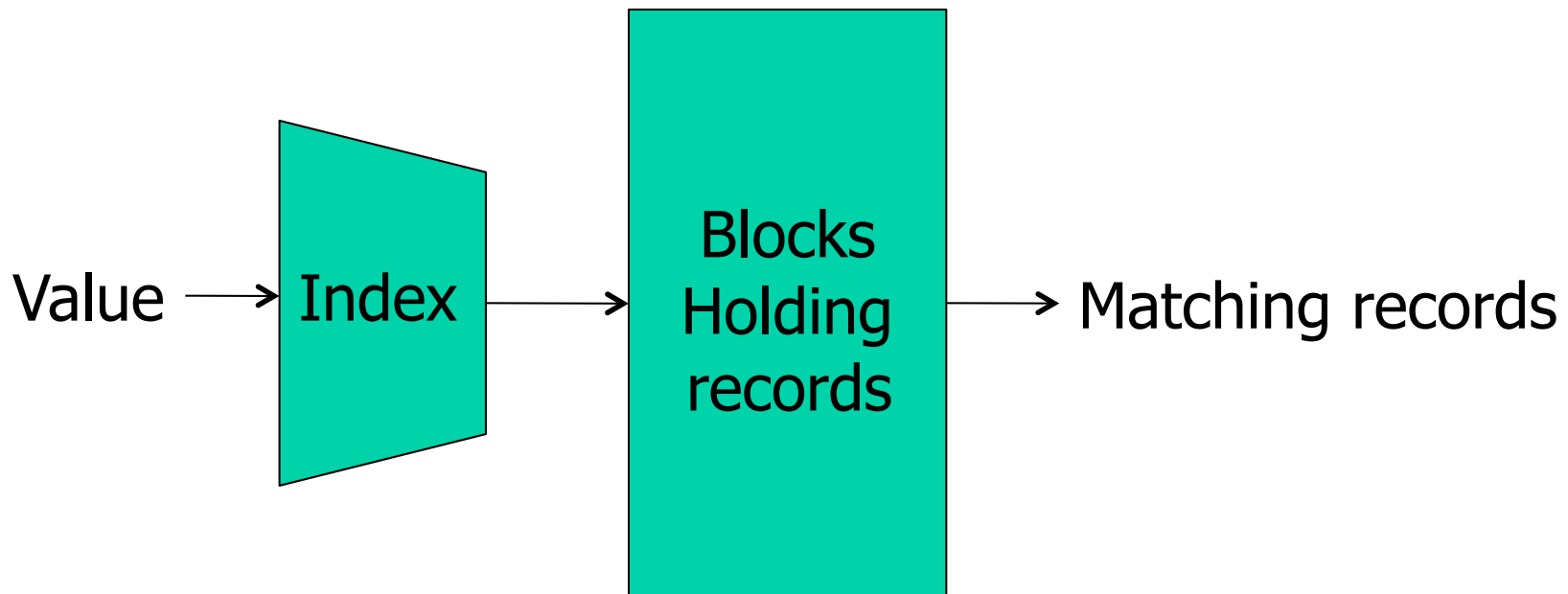
Index Structures

Finding Records

- How do we find the records for a query?
- **Example:** `SELECT * FROM Sells`
- Need to examine every block in every file
- Group blocks into files by relation!
- **Example:** `SELECT * FROM Sells
WHERE price = 20;`
- Need to examine every block in the file

Finding Records

- Use of indexes allows to narrow search to (almost) only the relevant blocks

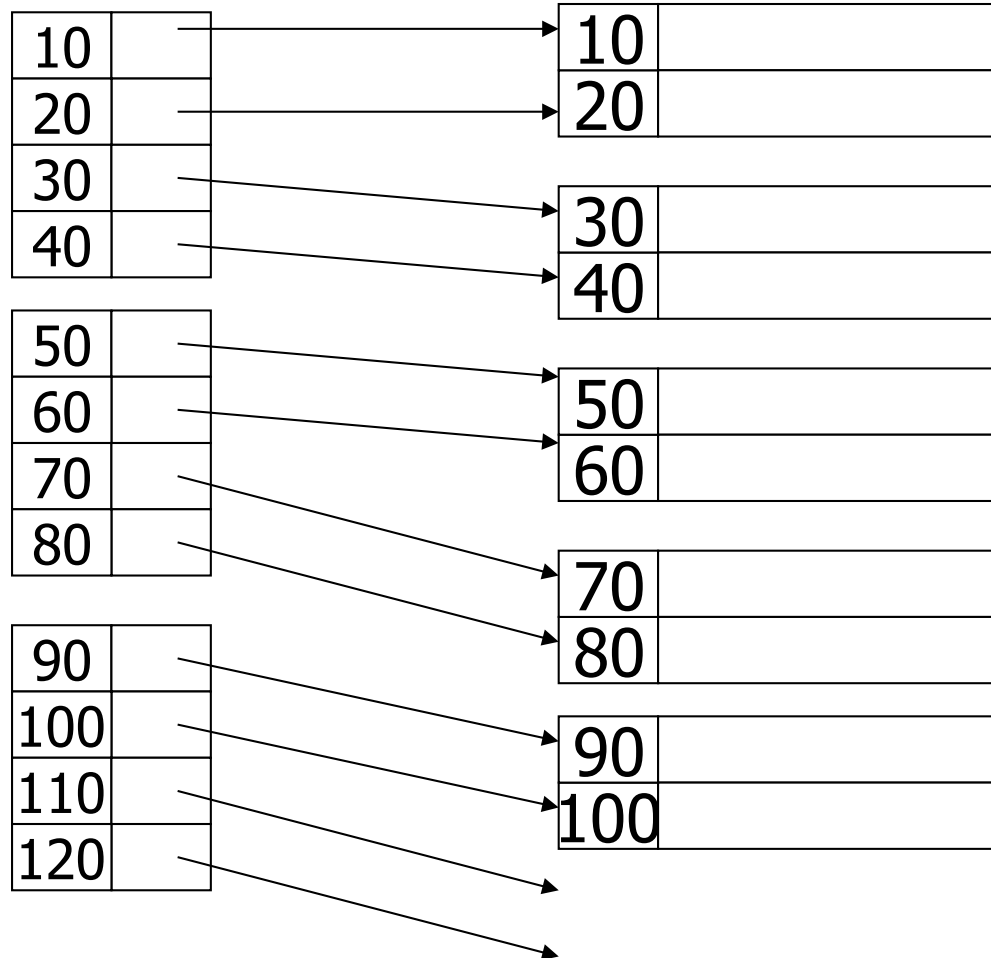


- Indexes can be *dense* or *sparse*

Dense Index

Dense Index

Sequential File

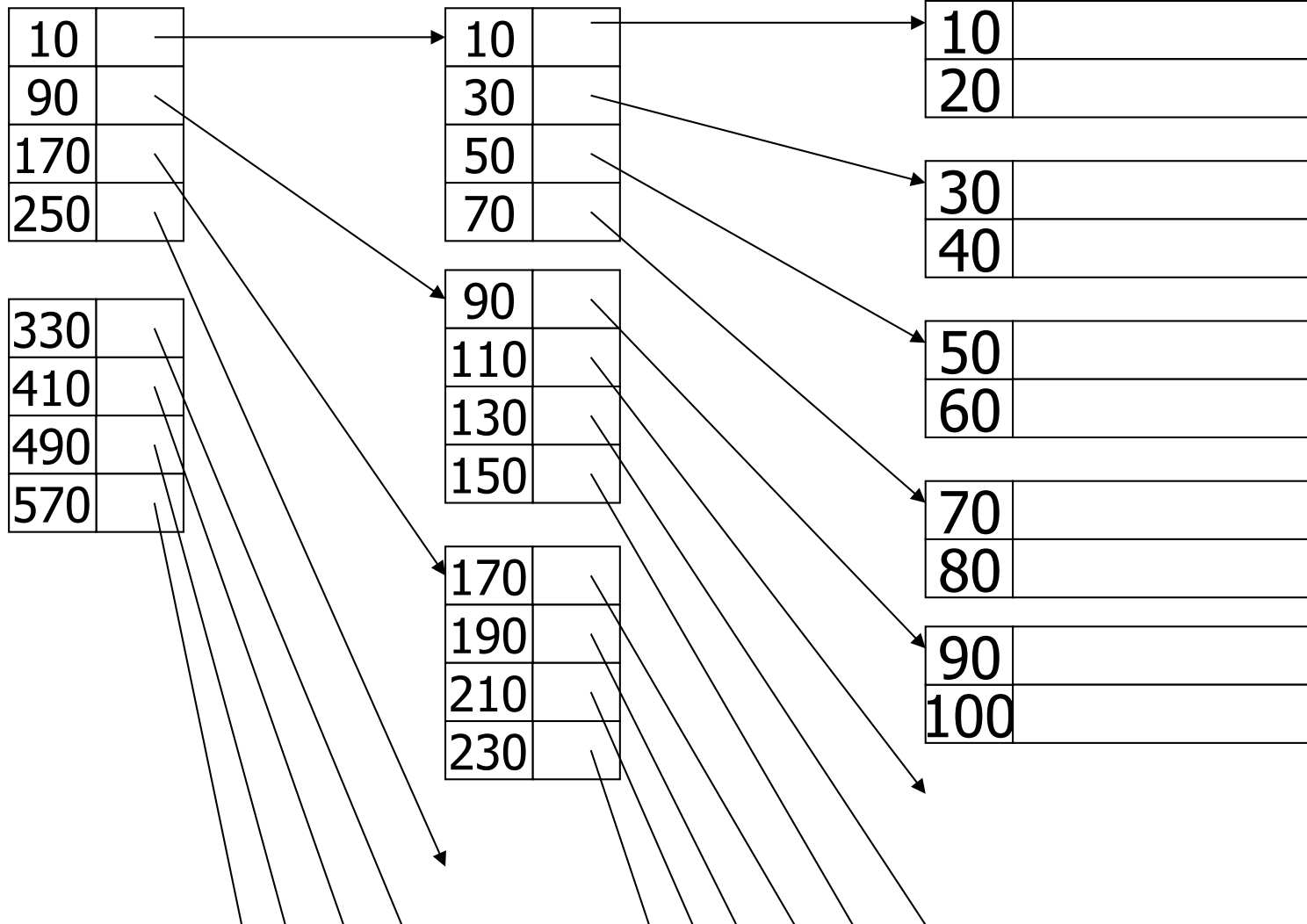


Sparse Index

2nd level

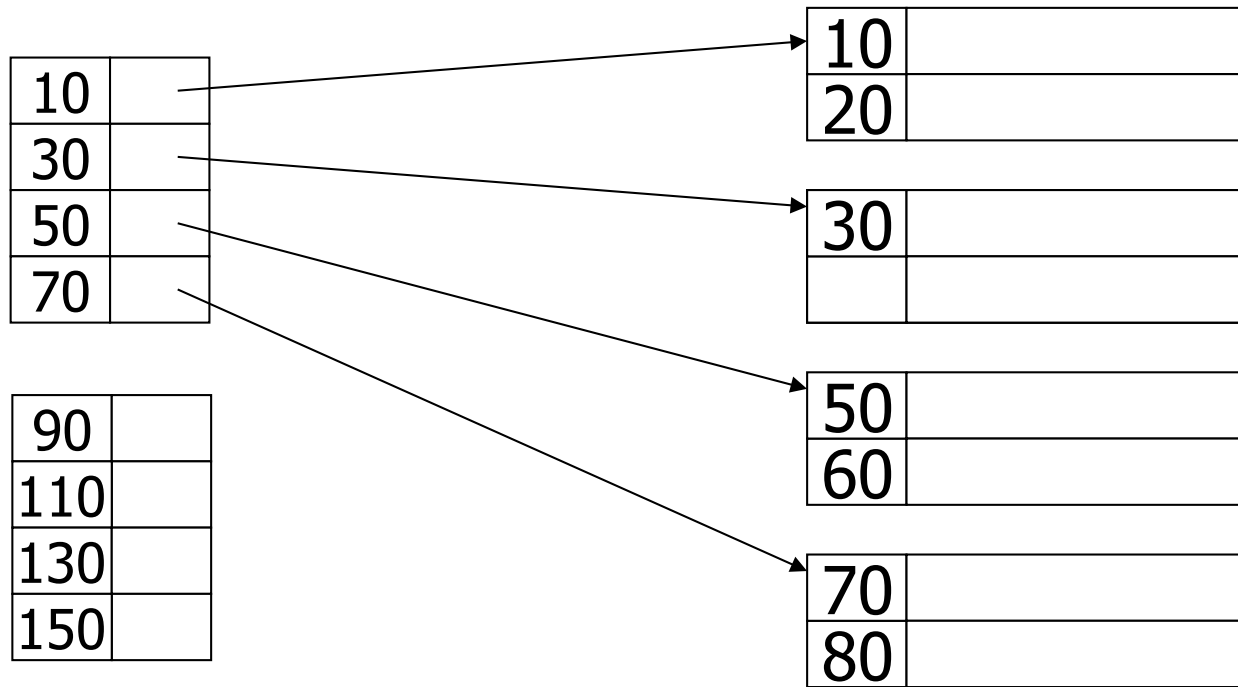
Sparse Index

Sequential File



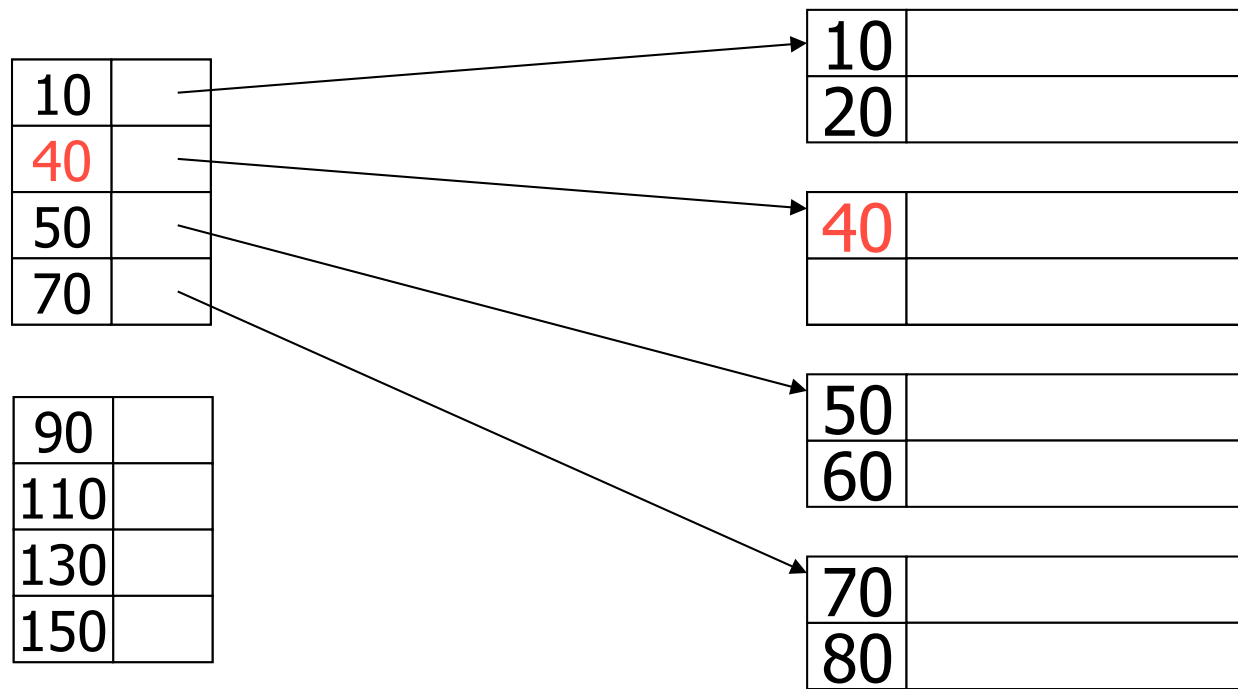
Deletion from Sparse Index

- Delete 40



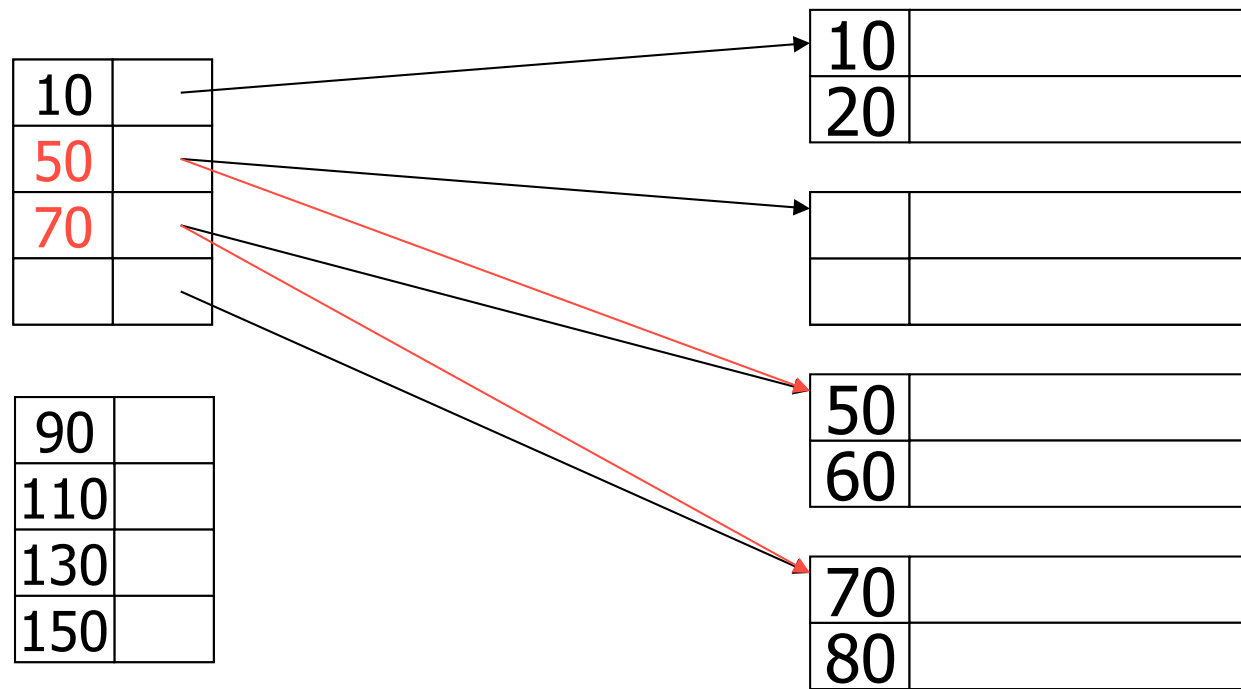
Deletion from Sparse Index

- Delete 30



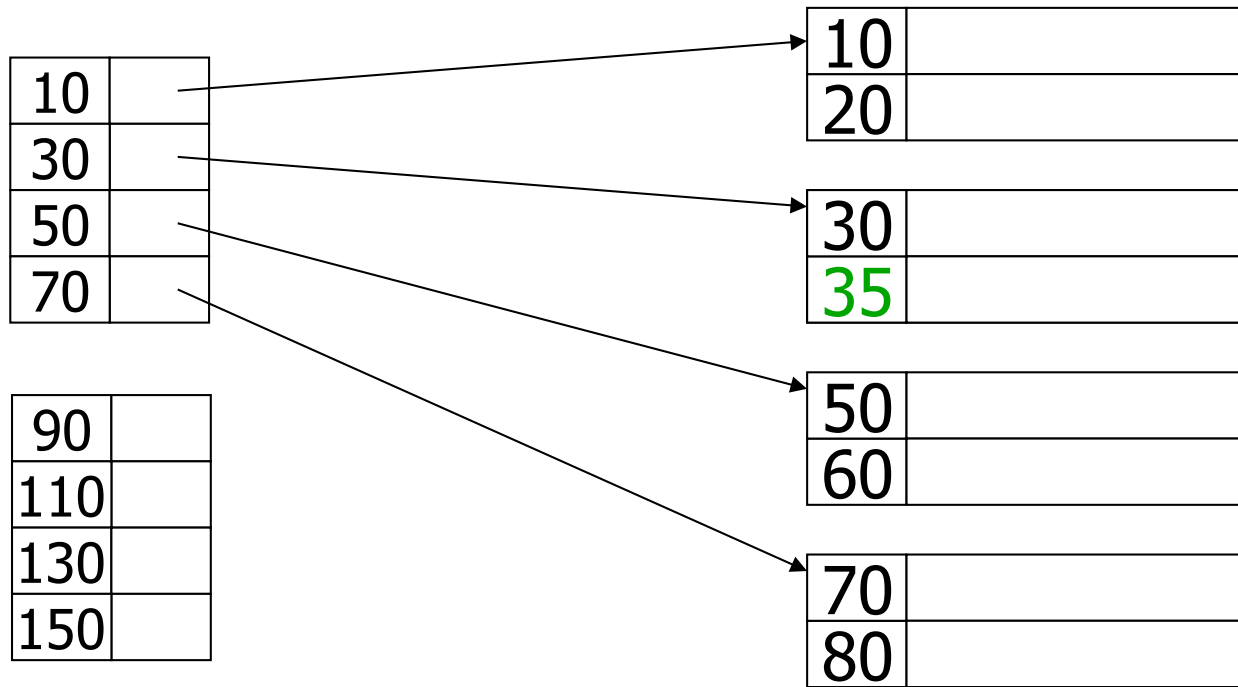
Deletion from Sparse Index

- Delete 30 & 40



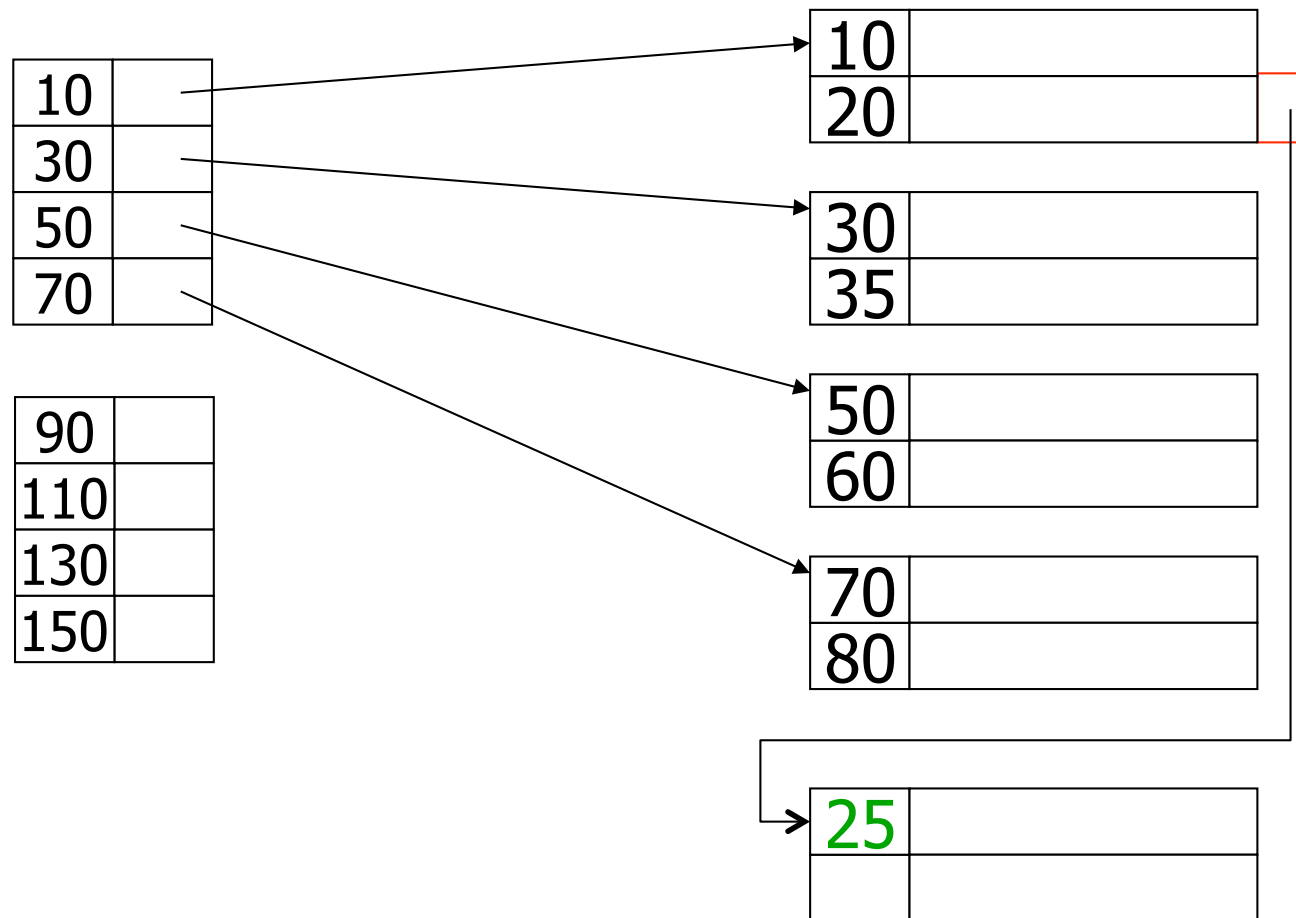
Insertion into **Sparse** Index

- Insert **35**



Insertion into Sparse Index

- Insert 25



Sparse vs Dense

- **Sparse** uses less index space per record (can keep more of index in memory)
- **Sparse** allows multi-level indexes
- **Dense** can tell if record exists without accessing it
- **Dense** needed for secondary indexes
- Primary index = order of records in storage
- Secondary index = impose different order

Secondary Index

2nd level

Secondary Index

Sequential File

10	
20	
50	

10	
10	
20	
20	

20	
30	
40	
50	

50	
60	

20	
40	

10	
20	

50	
30	

10	
50	

60	
20	

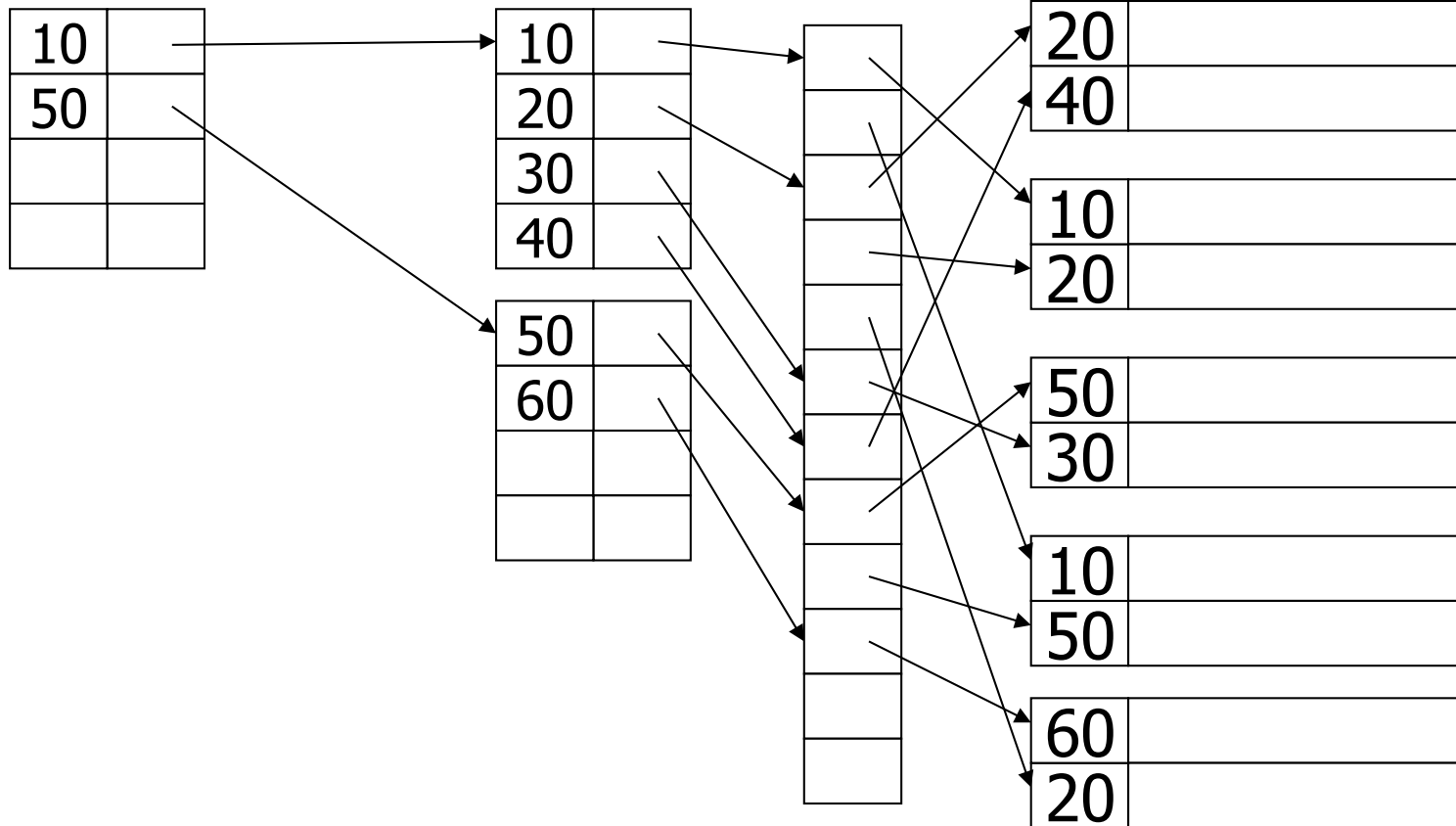
Careful when
Looking for 20

Secondary Index

2nd level

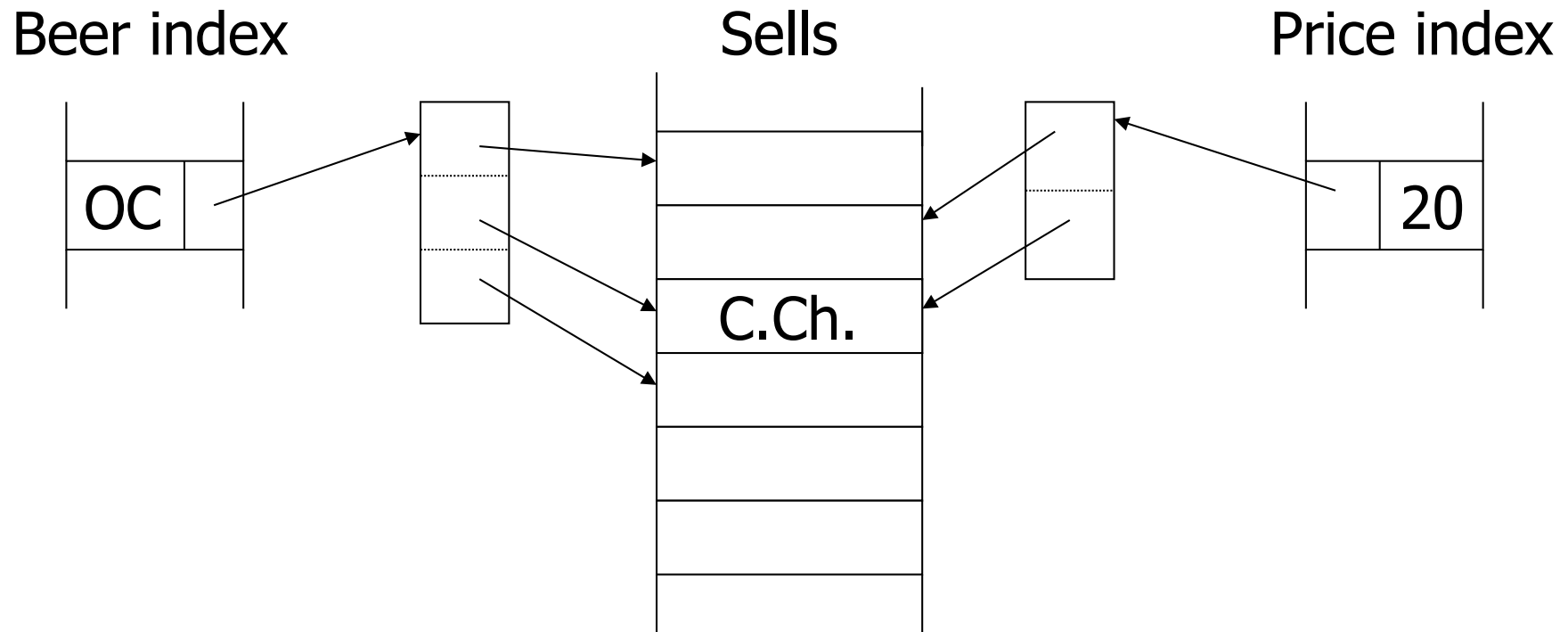
Secondary Index

Sequential File



Combining Indexes

- `SELECT * FROM Sells WHERE beer = "Od.Cl." AND price = "20"`

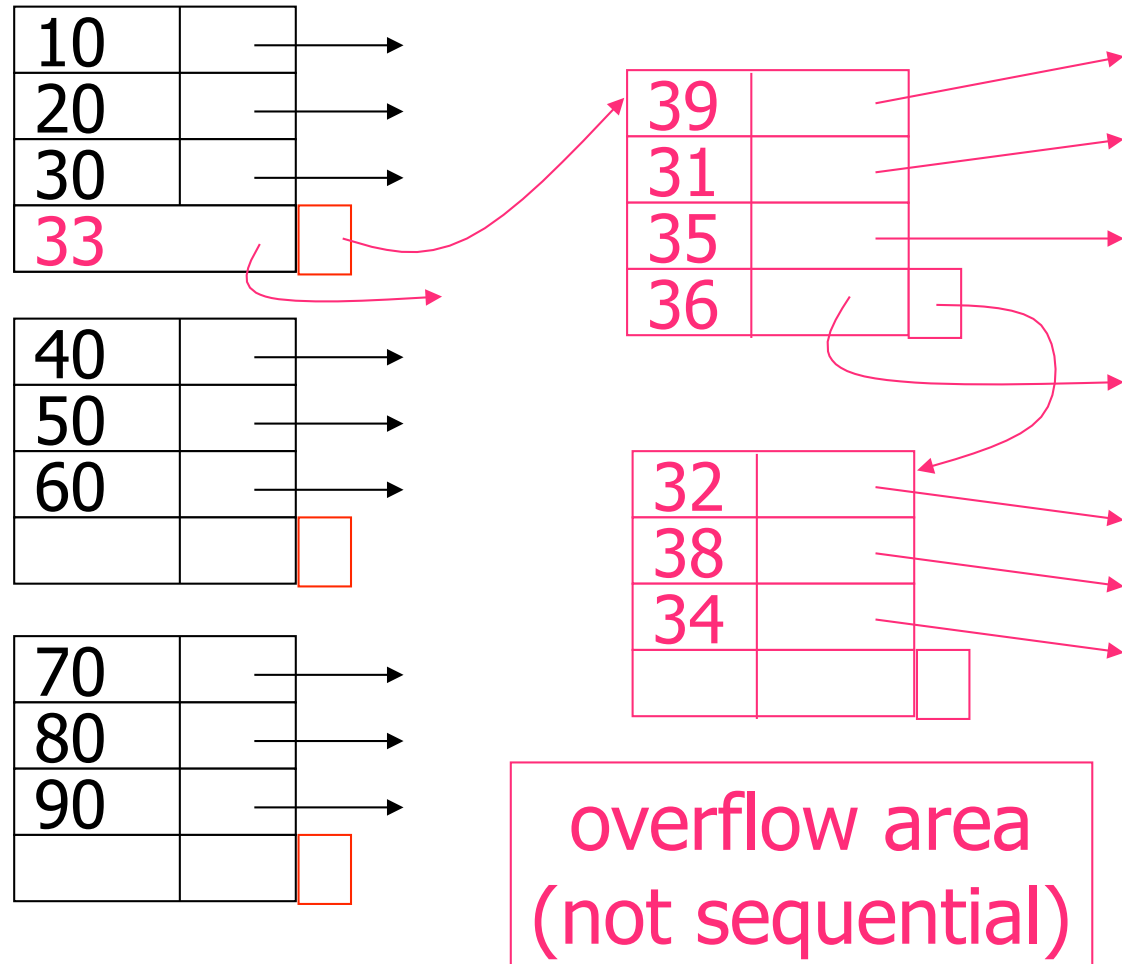


- Just intersect buckets in memory!

Conventional Indexes

- Sparse, Dense, Multi-level, ...
- **Advantages:**
 - Simple
 - Sequential index is good for scans
- **Disadvantage:**
 - Inserts expensive
 - Lose sequentiality and balance

Example: Unbalanced Index



B+Trees

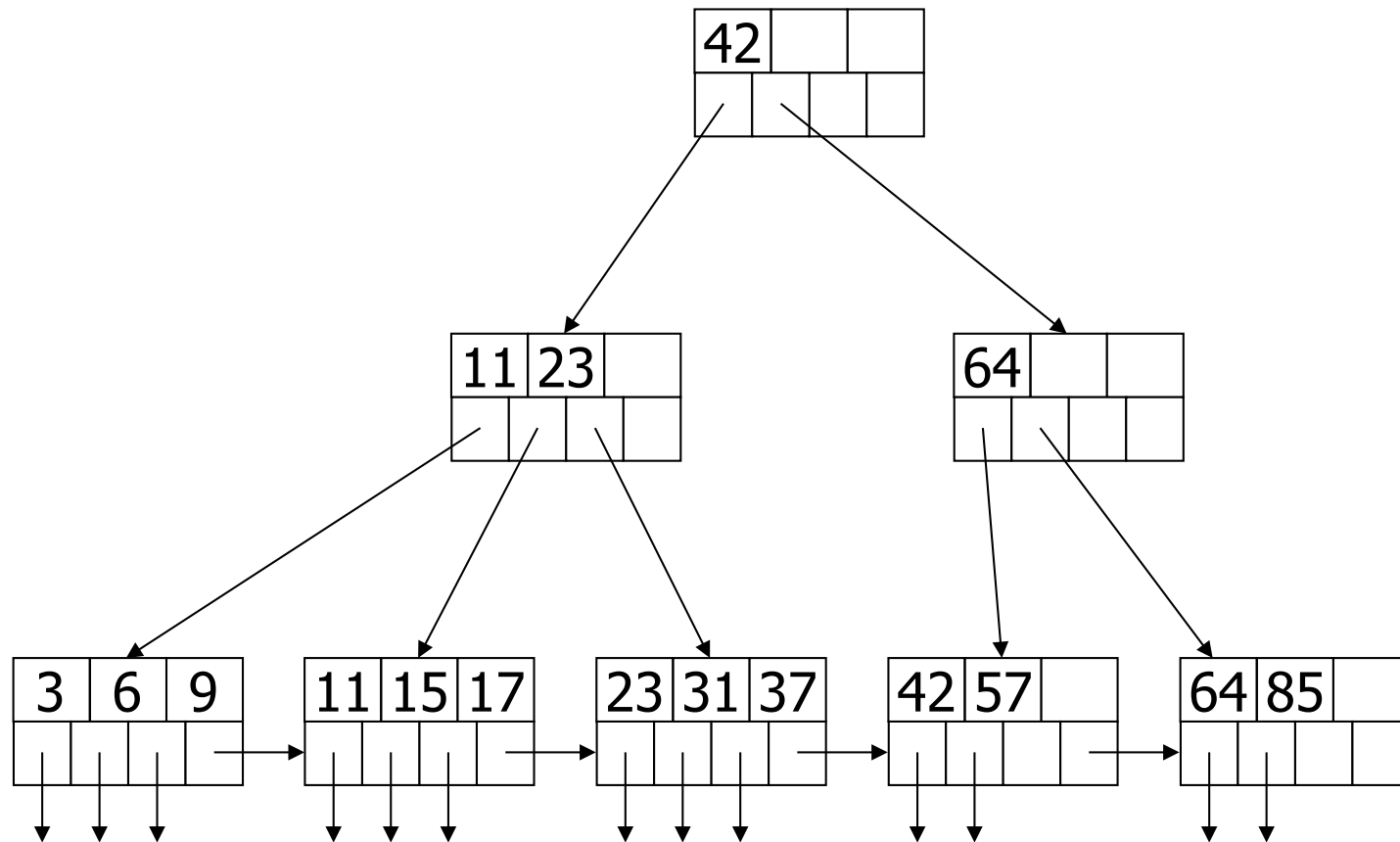
Idea

- Conventional indexes are fixed-level
- Give up sequentiality of the index in favour of balance
- B+Tree = variant of B-Tree
- Allows index tree to grow as needed
- Ensures that all blocks are between half used and completely full

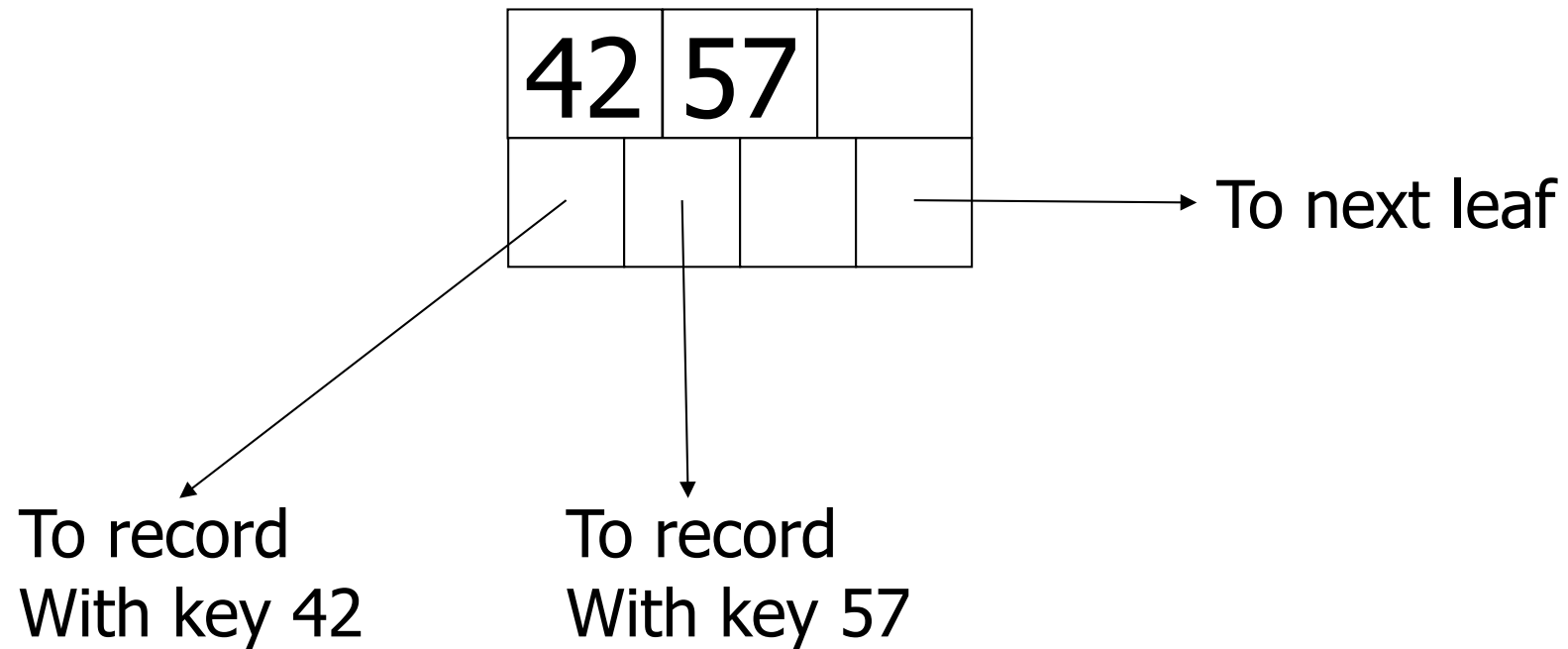
Characteristics

- Parameter n determines number of keys and pointers per node
- Key size 4 and pointer size 8 allows for maximal $n = 340$ ($4n + 8(n+1) < 4096$)
- Leafs contain at least $n/2$ key-pointer pairs to records and a pointer to the next leaf
- Interior nodes contain at least $(n-1)/2$ keys and at least $n/2$ pointers to other nodes
- No restrictions for the root node

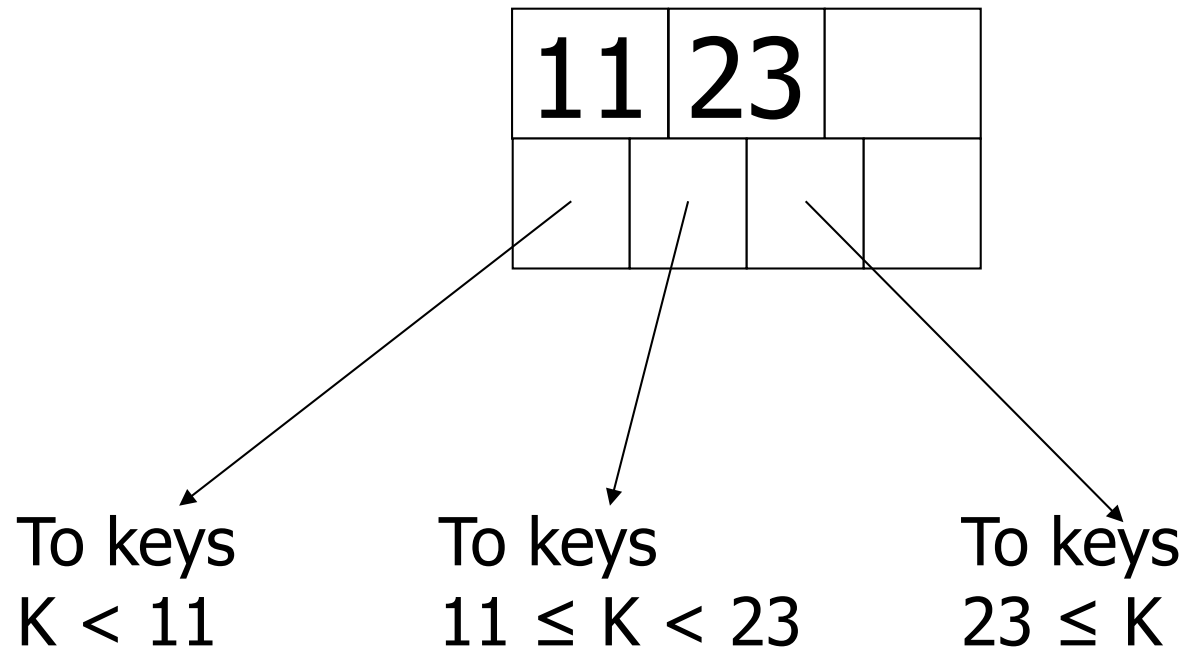
Example: B+Tree (n=3)



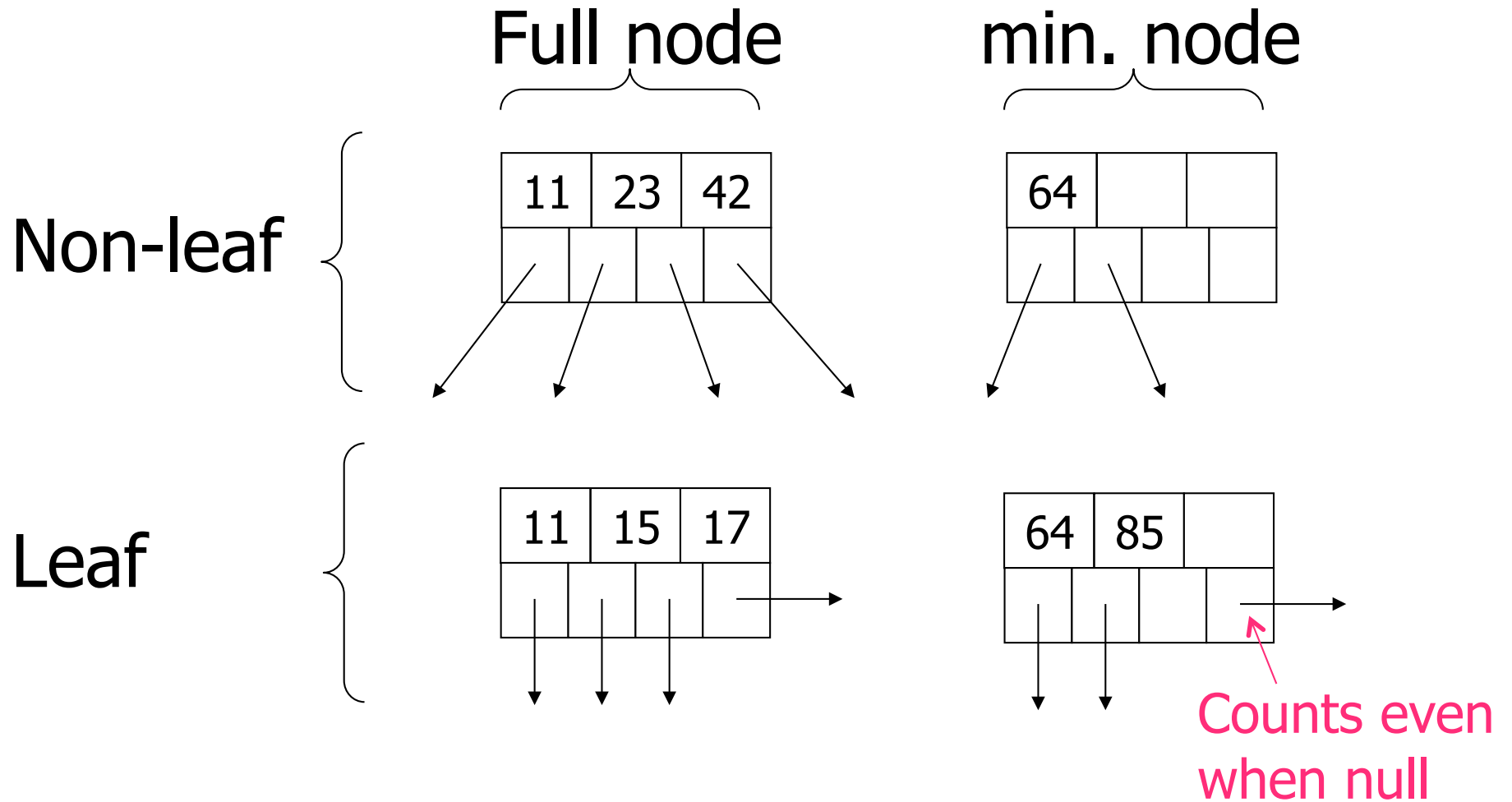
Example: Leaf node



Example: Interior node



Restrictions

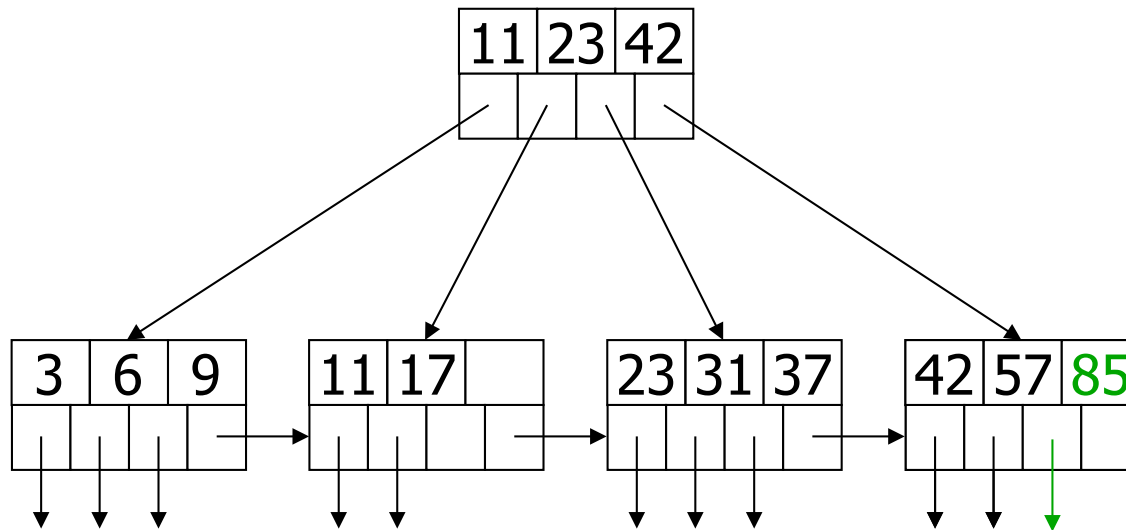


Insertion

- If there is place in the appropriate leaf, just insert it there
- **Otherwise:**
 - Split the leaf in two and divide the keys
 - Insert the smallest value reachable through the right node into the parent node
 - Recurse until there is enough room
- **Special case:** Splitting the root results in a new root

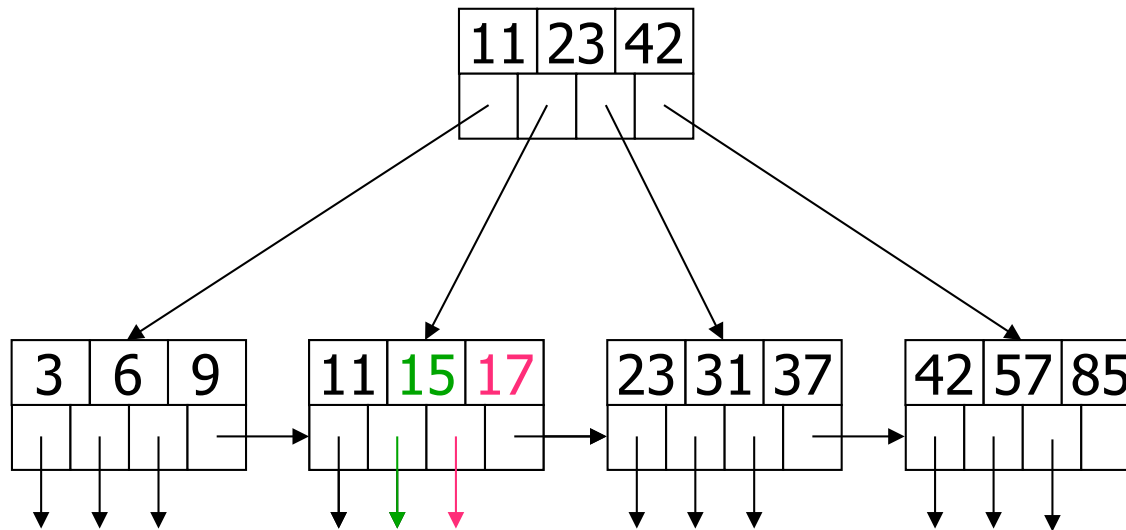
Example: Insertion

- Insert 85



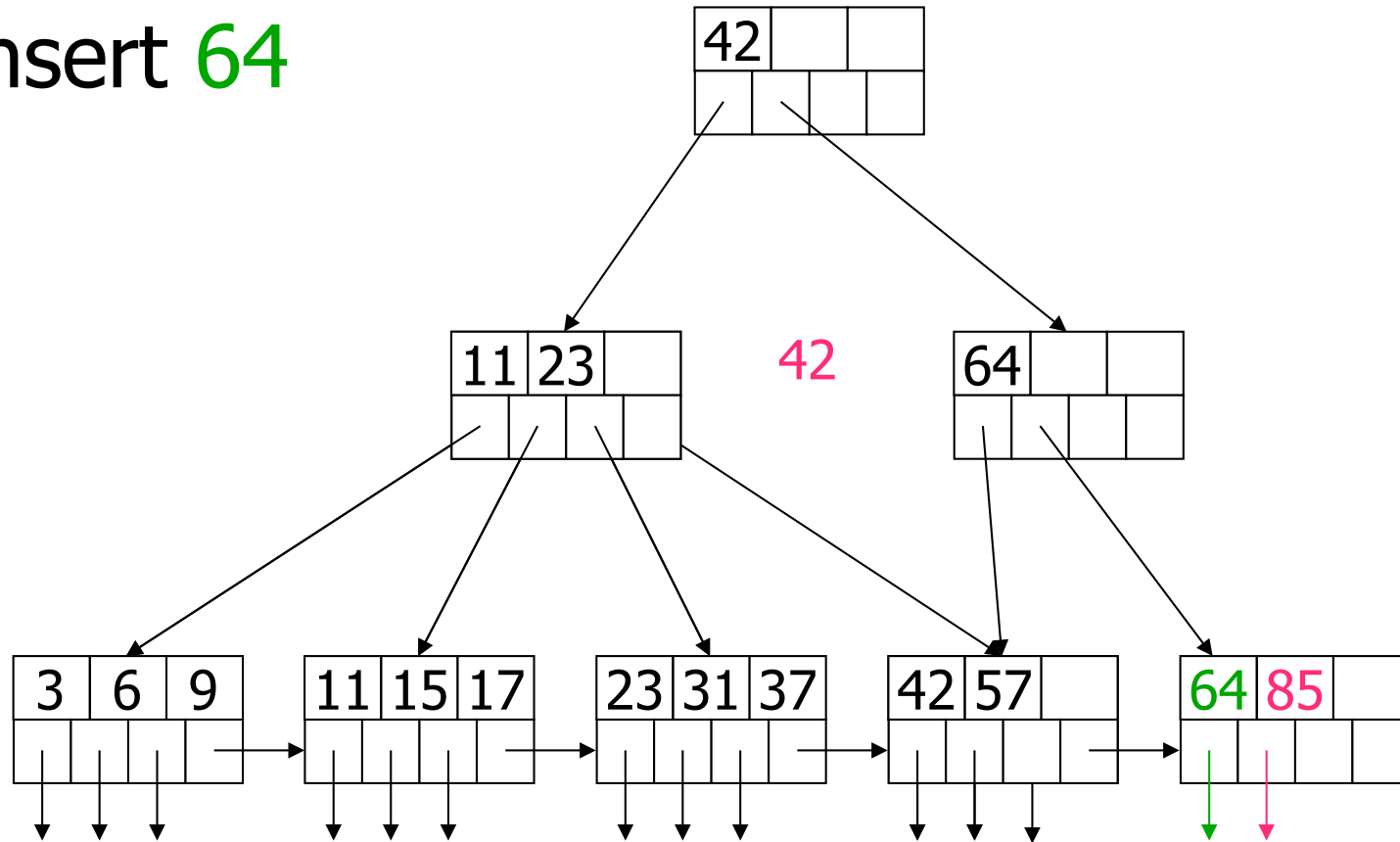
Example: Insertion

- Insert 15



Example: Insertion

- Insert 64

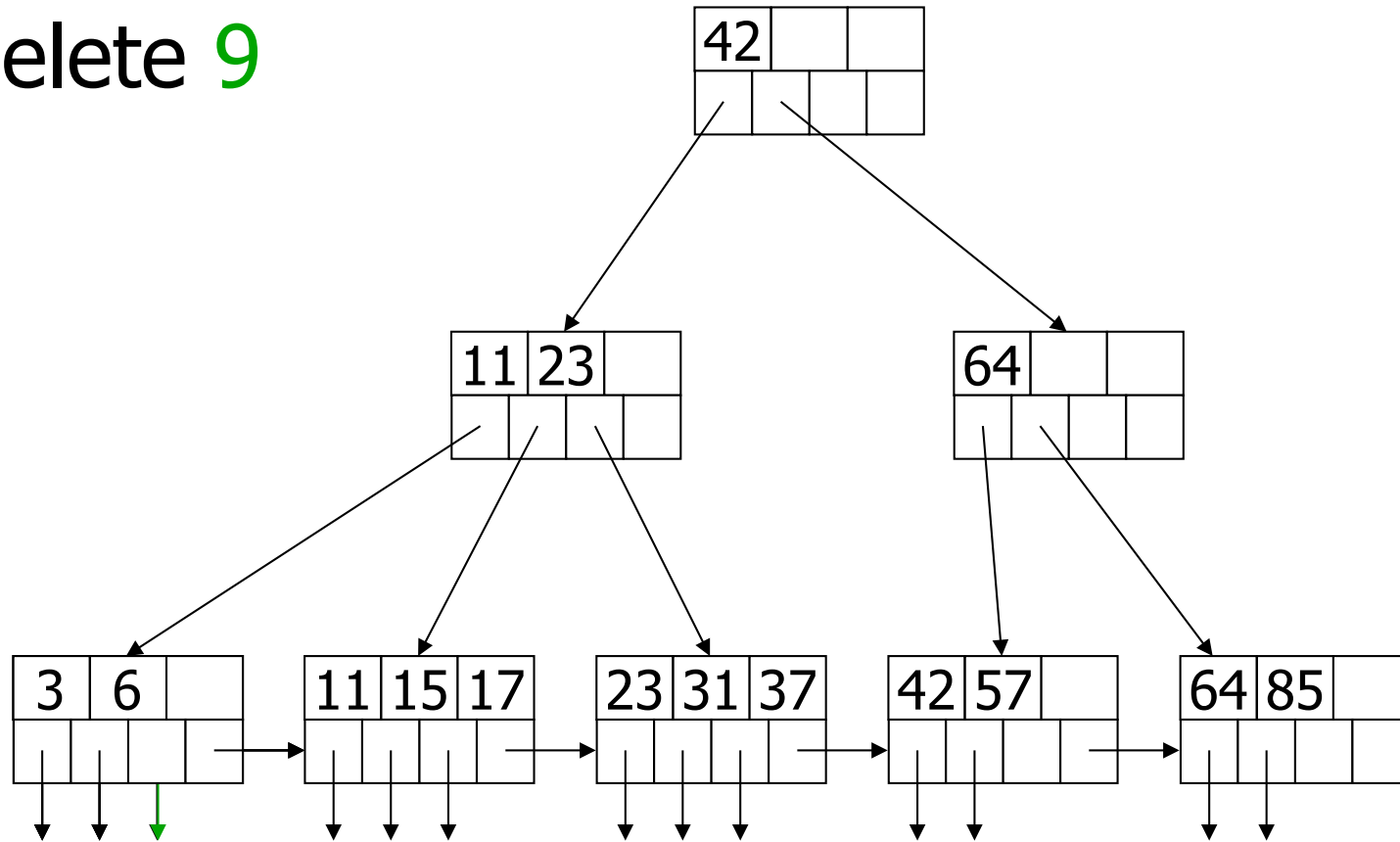


Deletion

- If there are enough keys left in the appropriate leaf, just delete the key
- **Otherwise:**
 - If there is a direct sibling with more than minimum key, steal one!
 - If not, join the node with a direct sibling and delete the smallest value reachable through the former right sibling from its parent
- **Special case:** If the root contains only one pointer after deletion, delete it

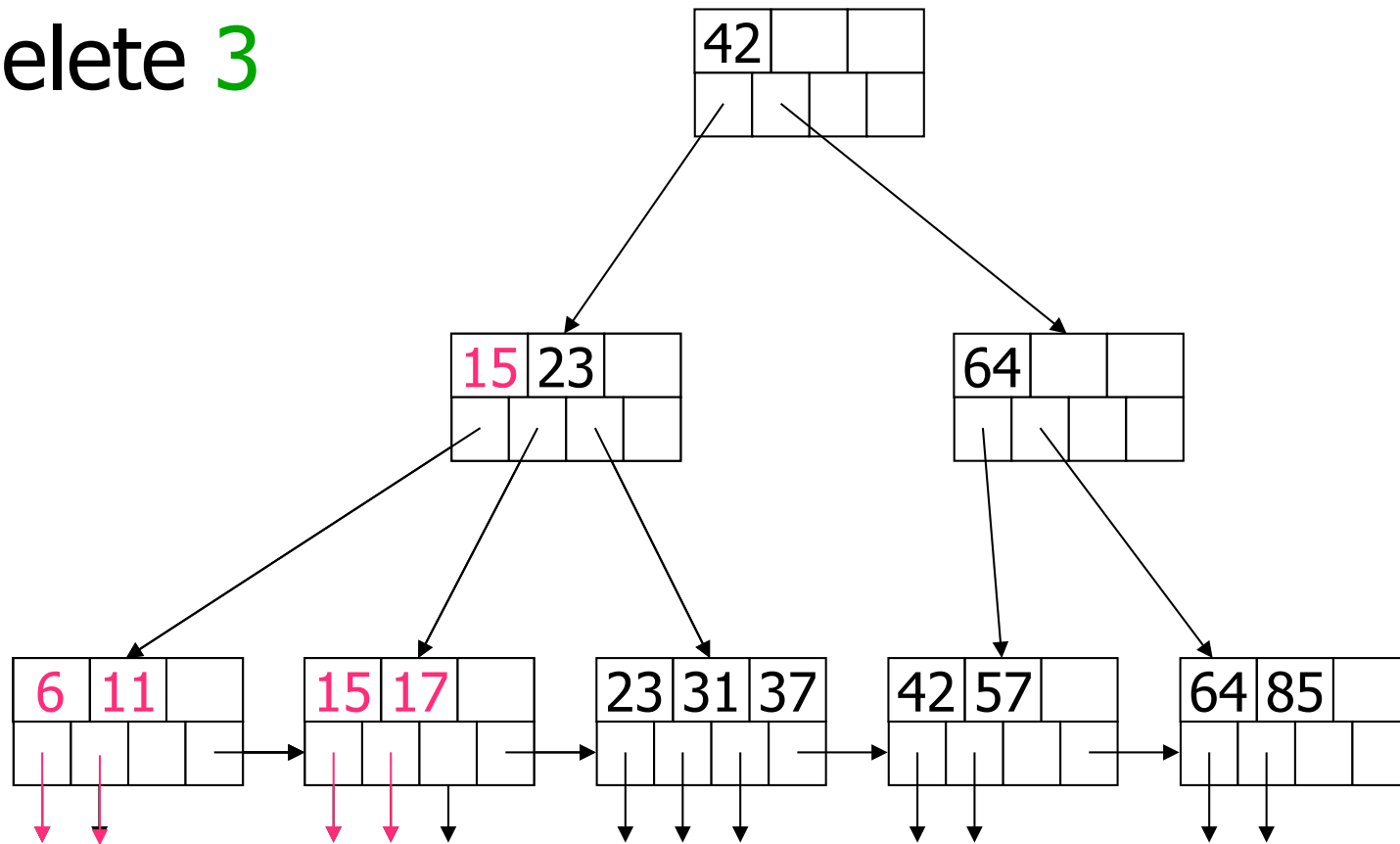
Example: Deletion

- Delete 9



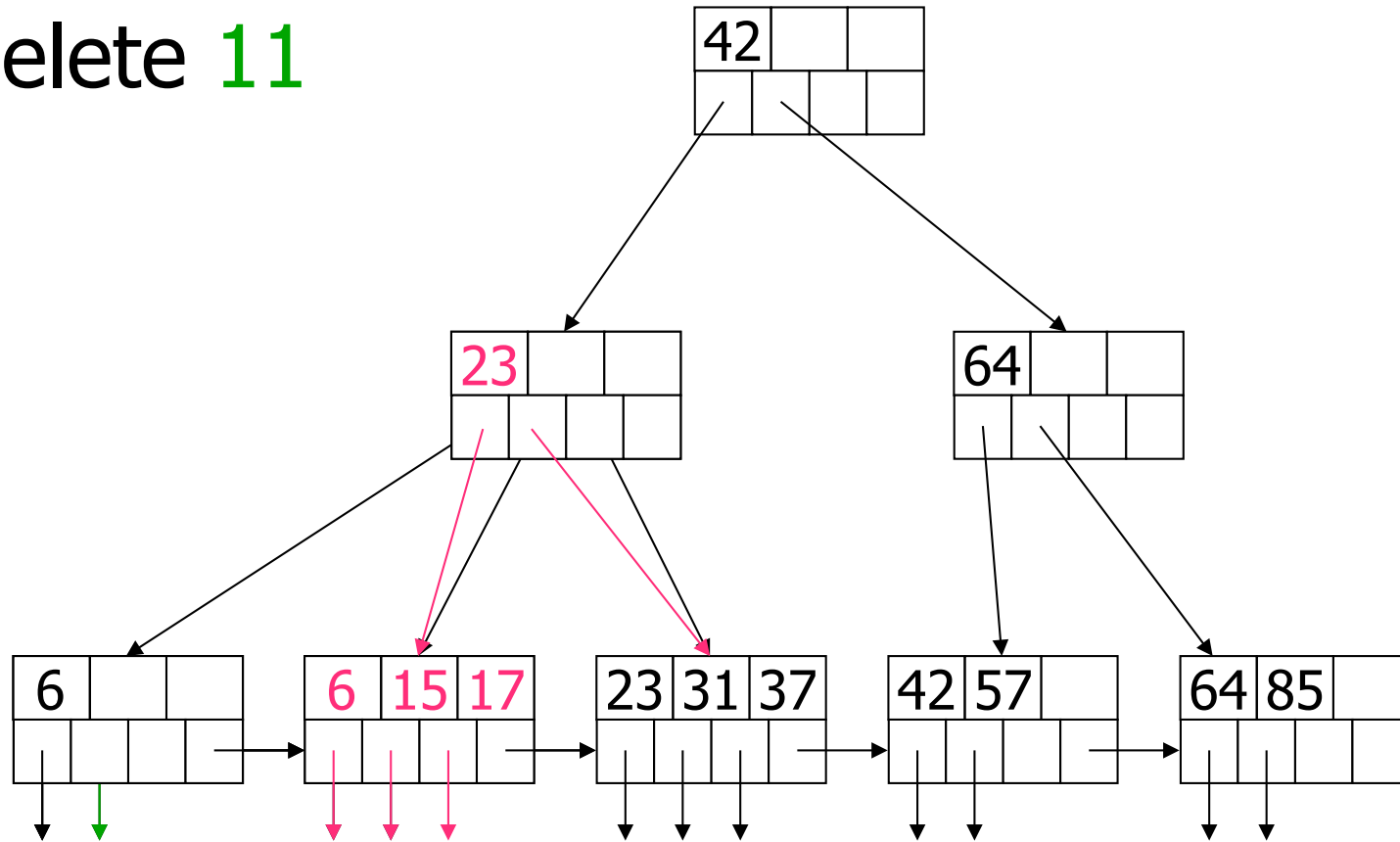
Example: Deletion

- Delete 3



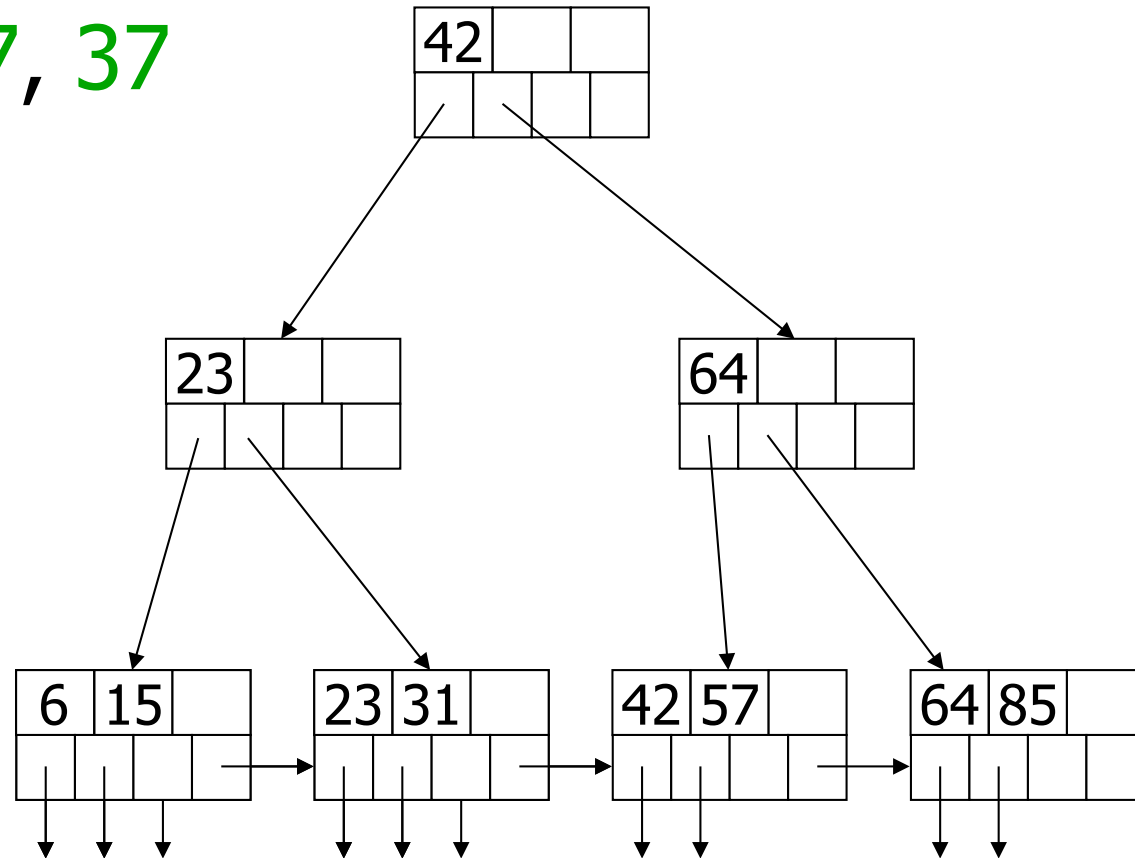
Example: Deletion

- Delete 11



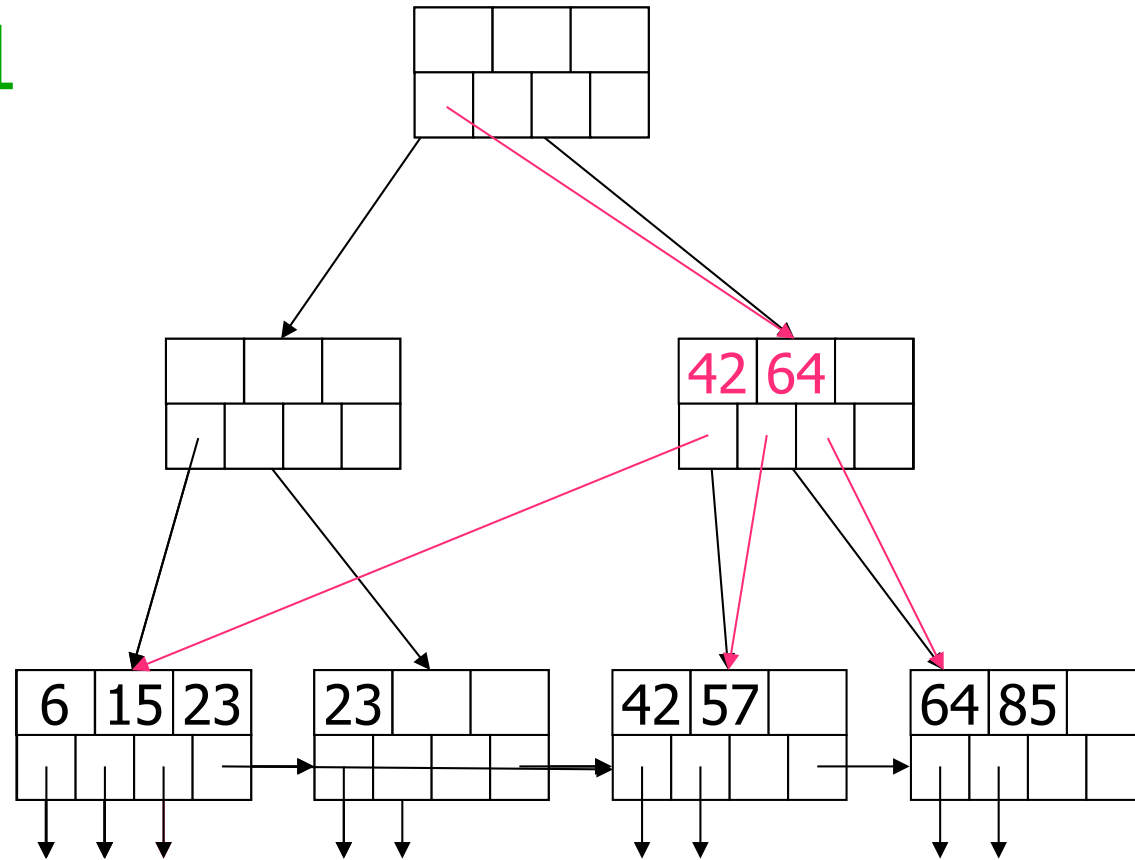
Example: Deletion

- Delete 17, 37



Example: Deletion

- Delete 31



Efficiency

- Need to load one block for each level!
- With $n = 340$ and an average fill of 255 pointers, we can index $255^3 = 16.6$ million records in only 3 levels
- There are at most 342 blocks in the first two levels
- First two levels can be kept in memory using less than 1.4 Mbyte
- Only need to access one block!

Range Queries

- Queries often restrict an attribute to a range of values

- **Example:**

```
SELECT * FROM Sells  
WHERE beer > 20;
```

- Records are found efficiently by searching for value 20 and then traversing the leafs
- Can also be used if there is both an upper and a lower limit

Summary 10

More things you should know:

- Dense Index, Sparse Index
- Multi-Level Indexes
- Primary vs Secondary Index
- Structure of B+Trees
- Insertion and Deletion in B+Trees