Data Storage and Query Answering

Indexing and Hashing (2)

Summary So Far

- Conventional index
  - Basic Ideas: sparse, dense, multi-level…
  - Duplicate Keys
  - Secondary Indexes

<u>Advantage:</u>

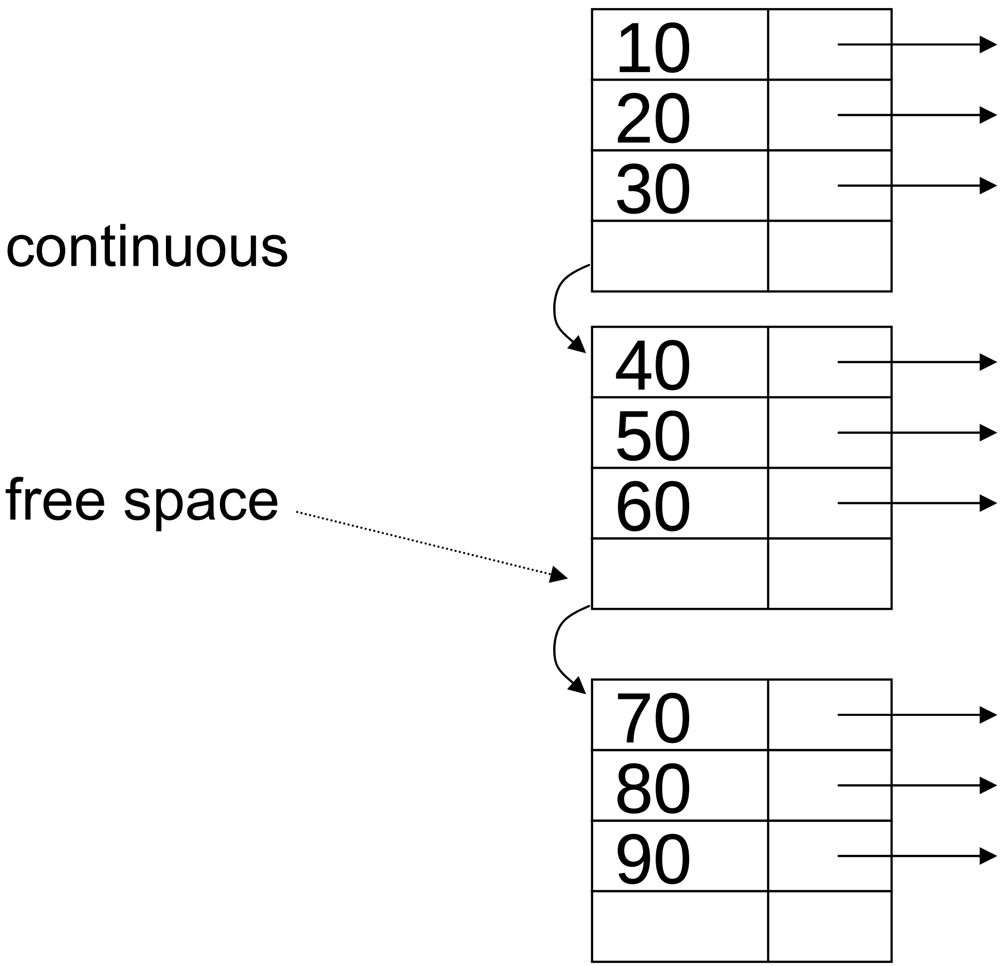
- Simple
- Index is sequential file, good for scans

<u>Disadvantage:</u>

- Inserts expensive, and/or
- Lose sequentiality & balance

Example

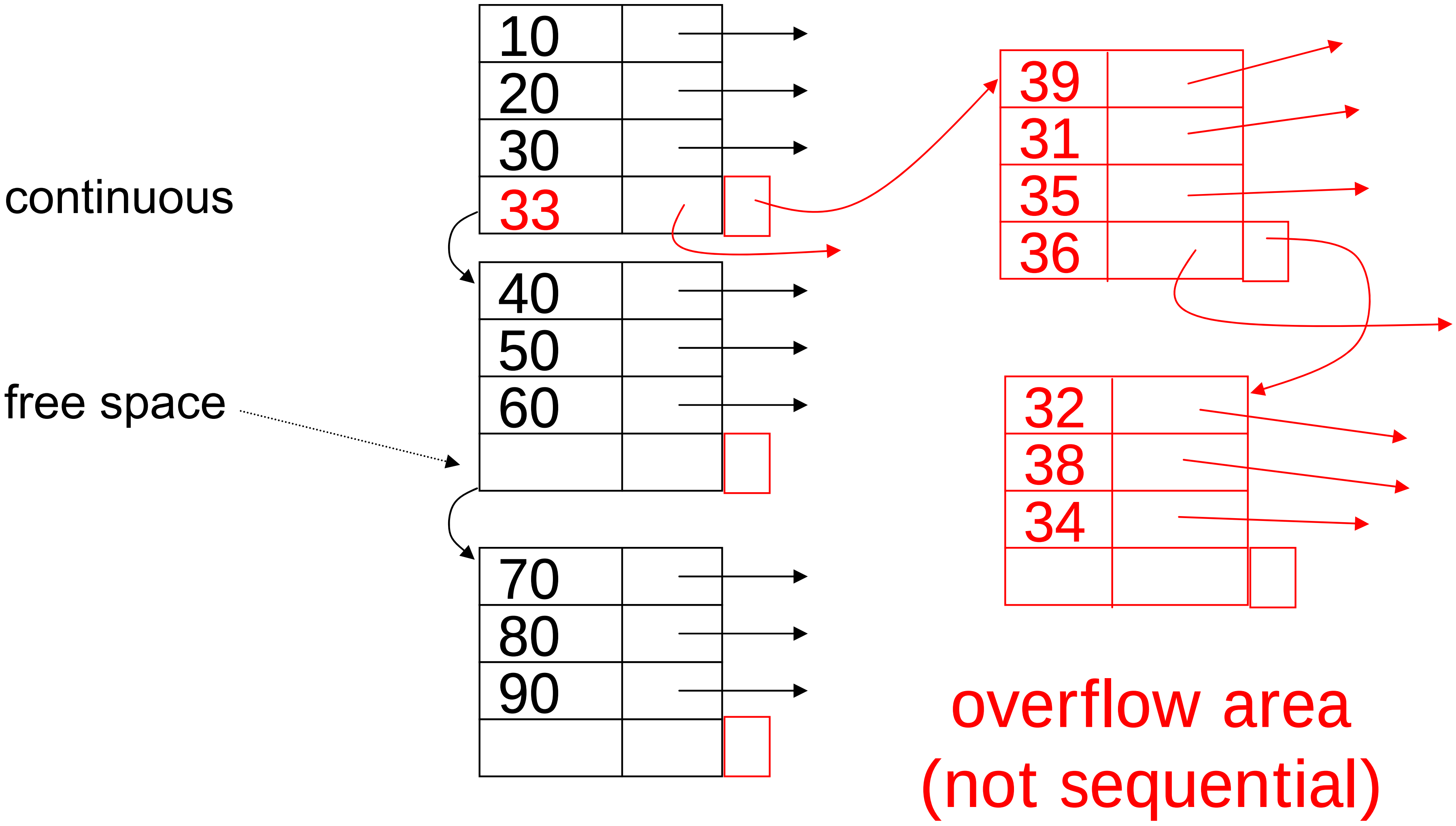
Example Index (sequential)



Example (cont.)

Example

Index (sequential)

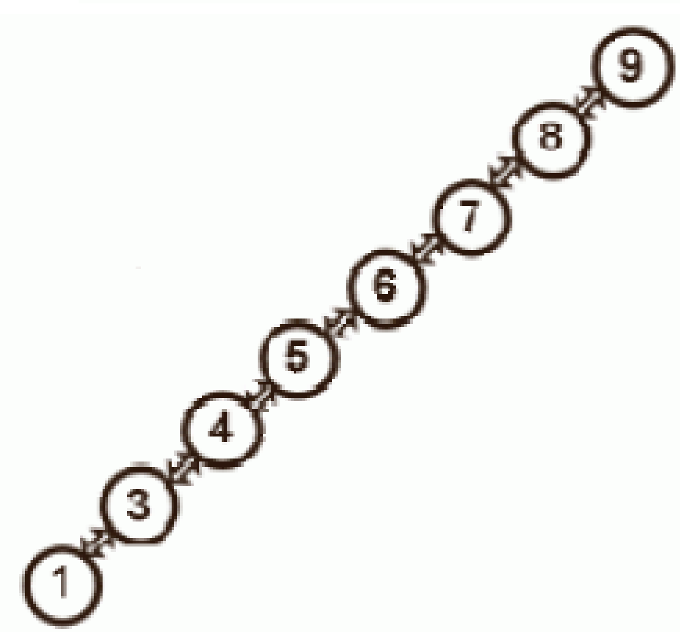


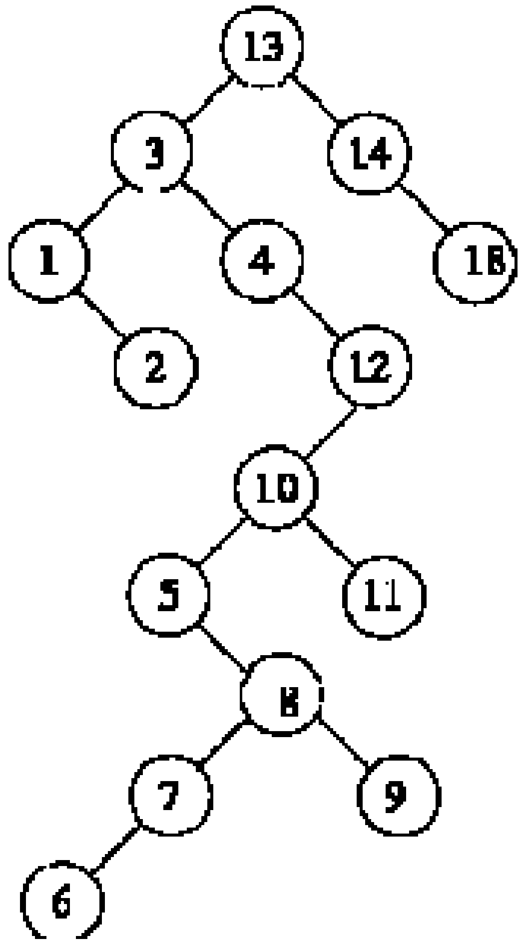
B+-Tree

- NEXT: Another type of index
  - Give up on sequentiality of index
  - Try to get “balance”

Balanced versus unbalanced tree

- Searching a balanced search tree O(log n)
- Searching an unbalanced search tree can be O(n)

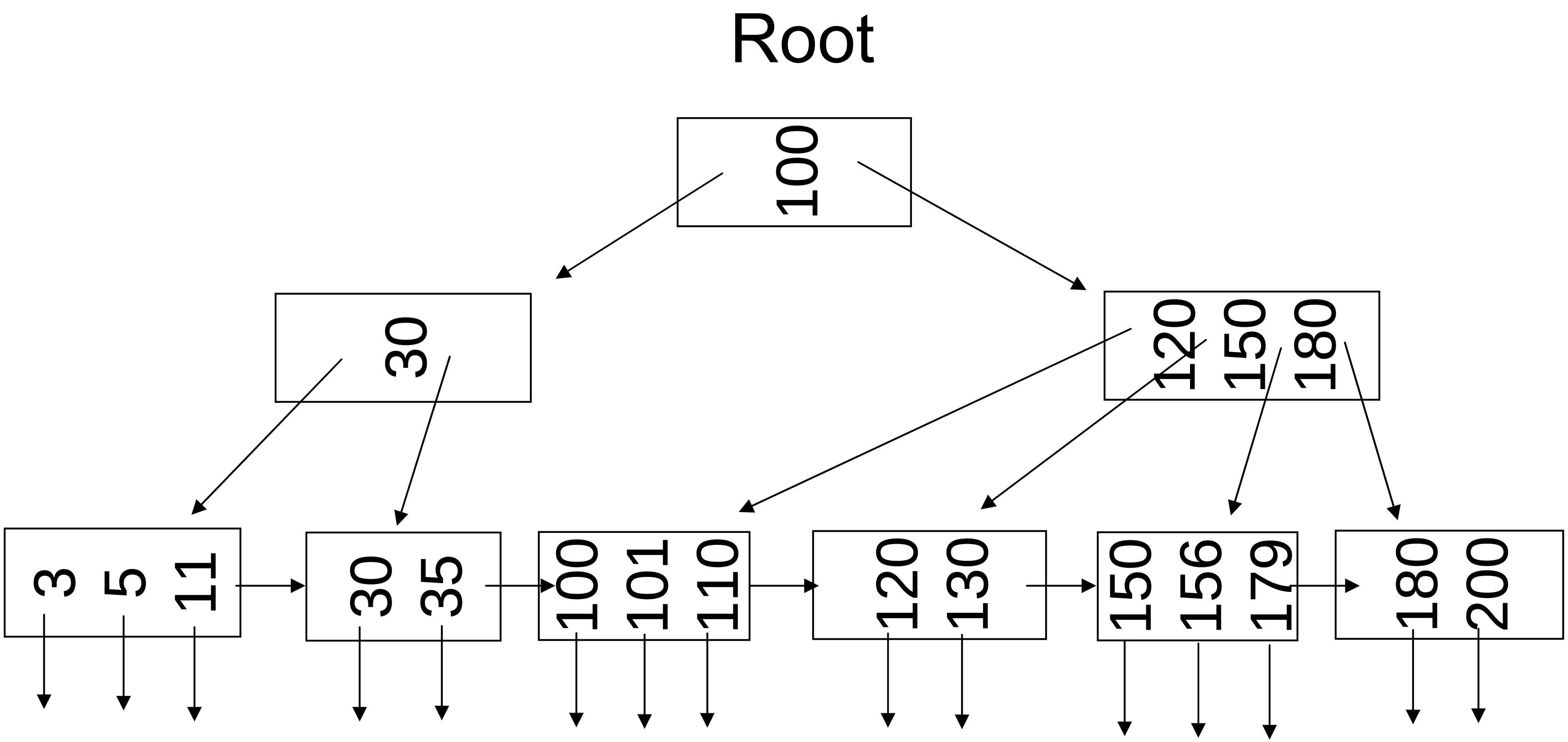




B+-Tree Example

B+Tree Example

n=3



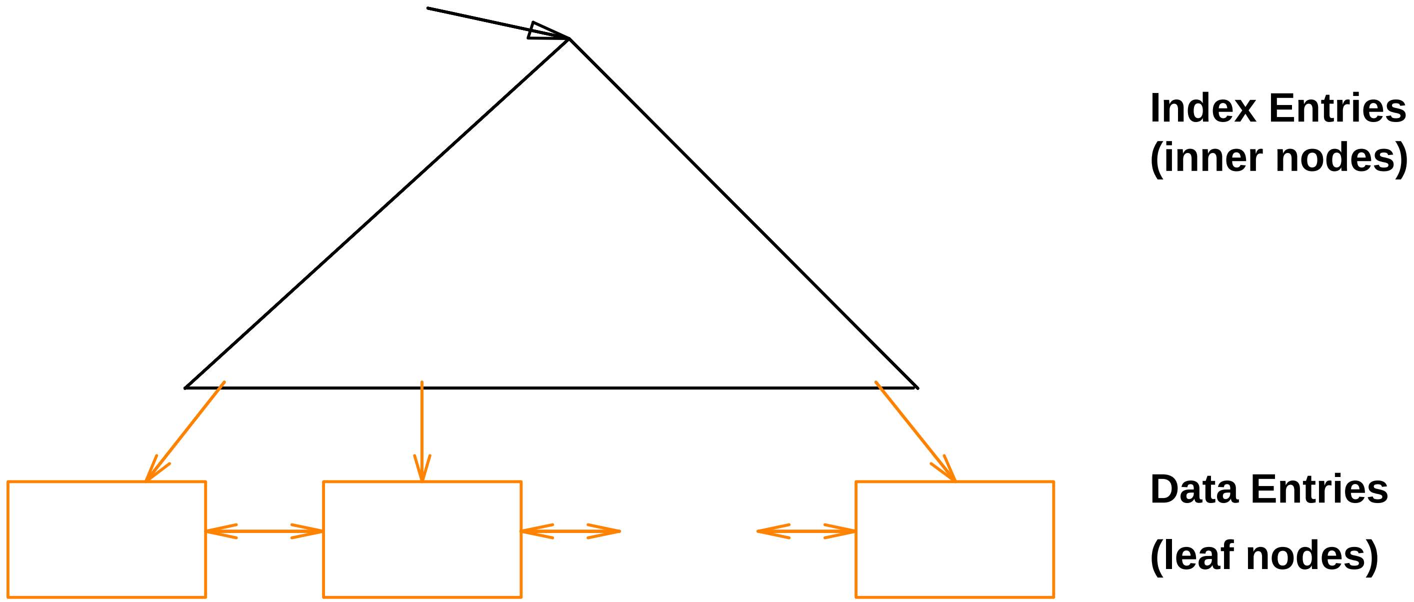
B+-Trees

Introduction

- B+-trees are *balanced*, i.e. all leaves at same level. This guarantees efficient access.
- B+-trees use small space utilization.
- n (*order*): maximum number of keys per node, minimum number of keys is roughly n/2.
- Exception: root may have one key only.
- $m + 1$ pointers in node, m actual number of keys.

B+-Trees

Introduction



→ leaf nodes are linked in sequential order

B+-Trees

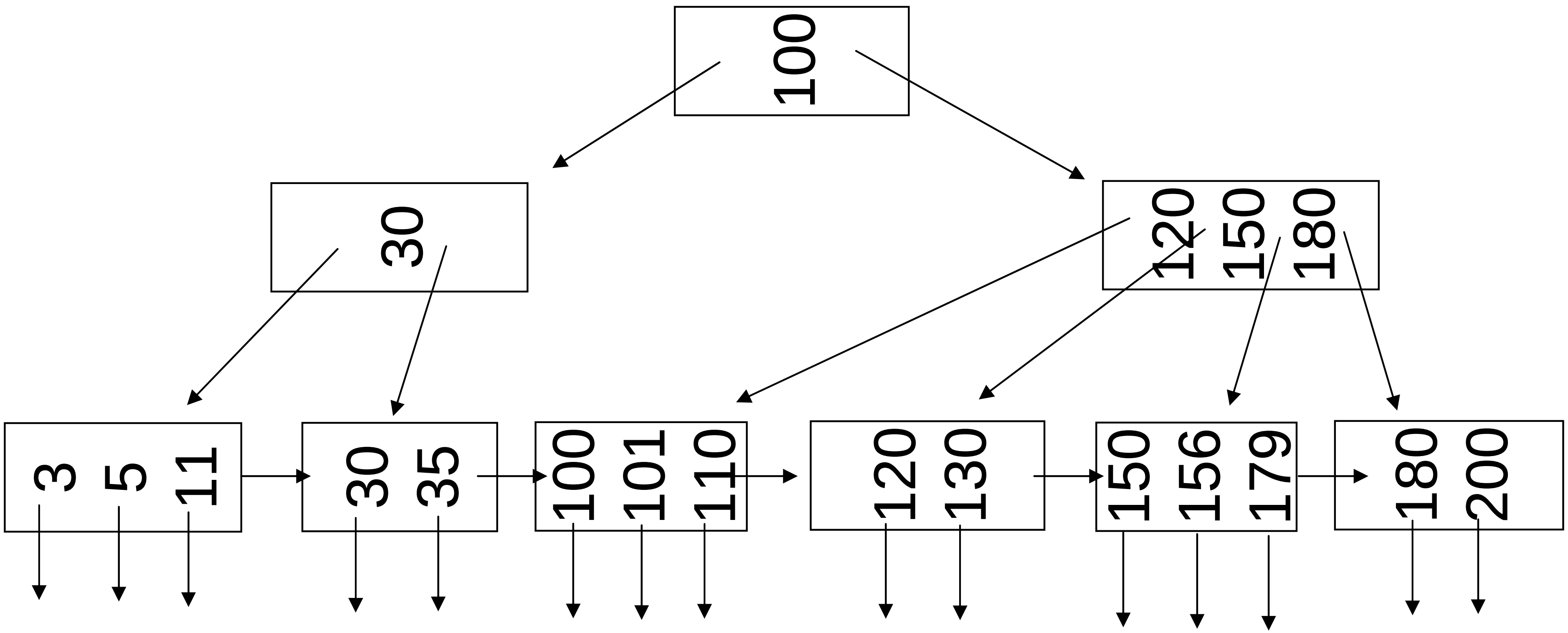
Introduction

- Node format: $(p_1, k_1, \ldots, p_n, k_n, p_{n+1})$
  p_i: pointer, k_i: search key
- Node with *m* pointers has *m* children and corresponding sub-trees.
- *n+1*-th index entry has only pointer. At leaf level, this pointer references the next leaf node.
- *Search key property*: *i*-th subtree contains data entries with search key $k < k_i$, i+1-th subtree contains data entries with search key $k >= k_i$.

B+-Trees

Example

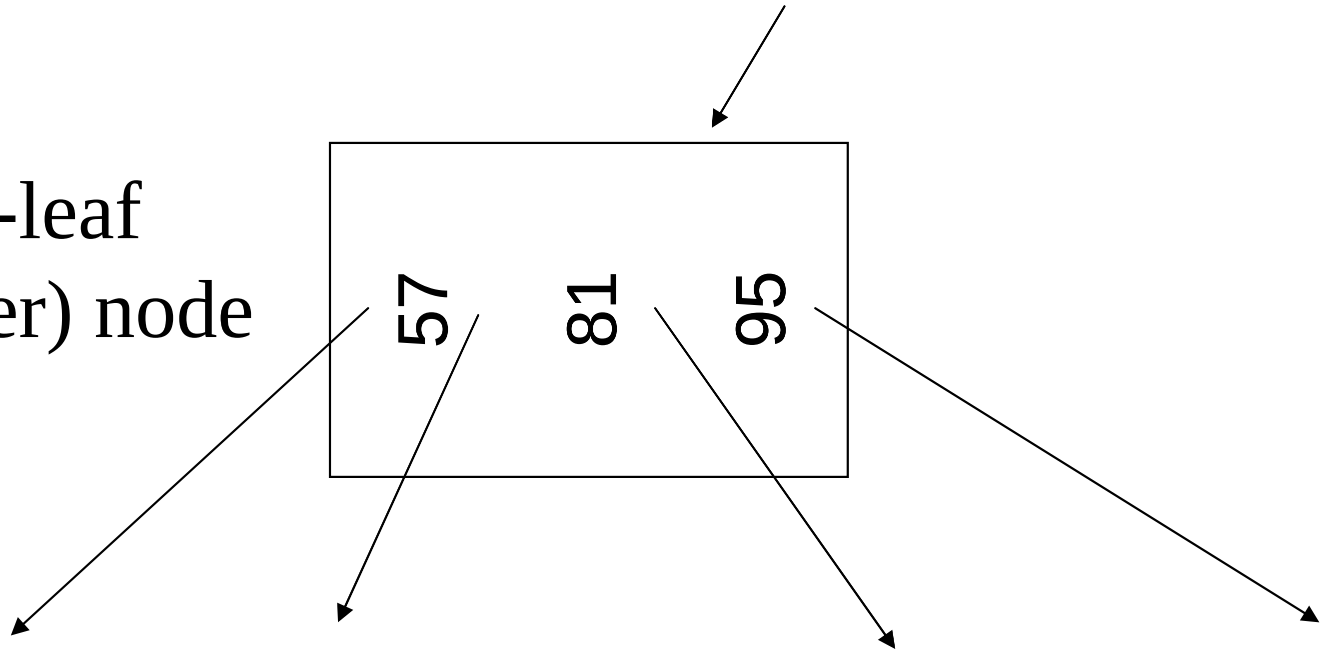
Root n = 3



B+-Trees

Example

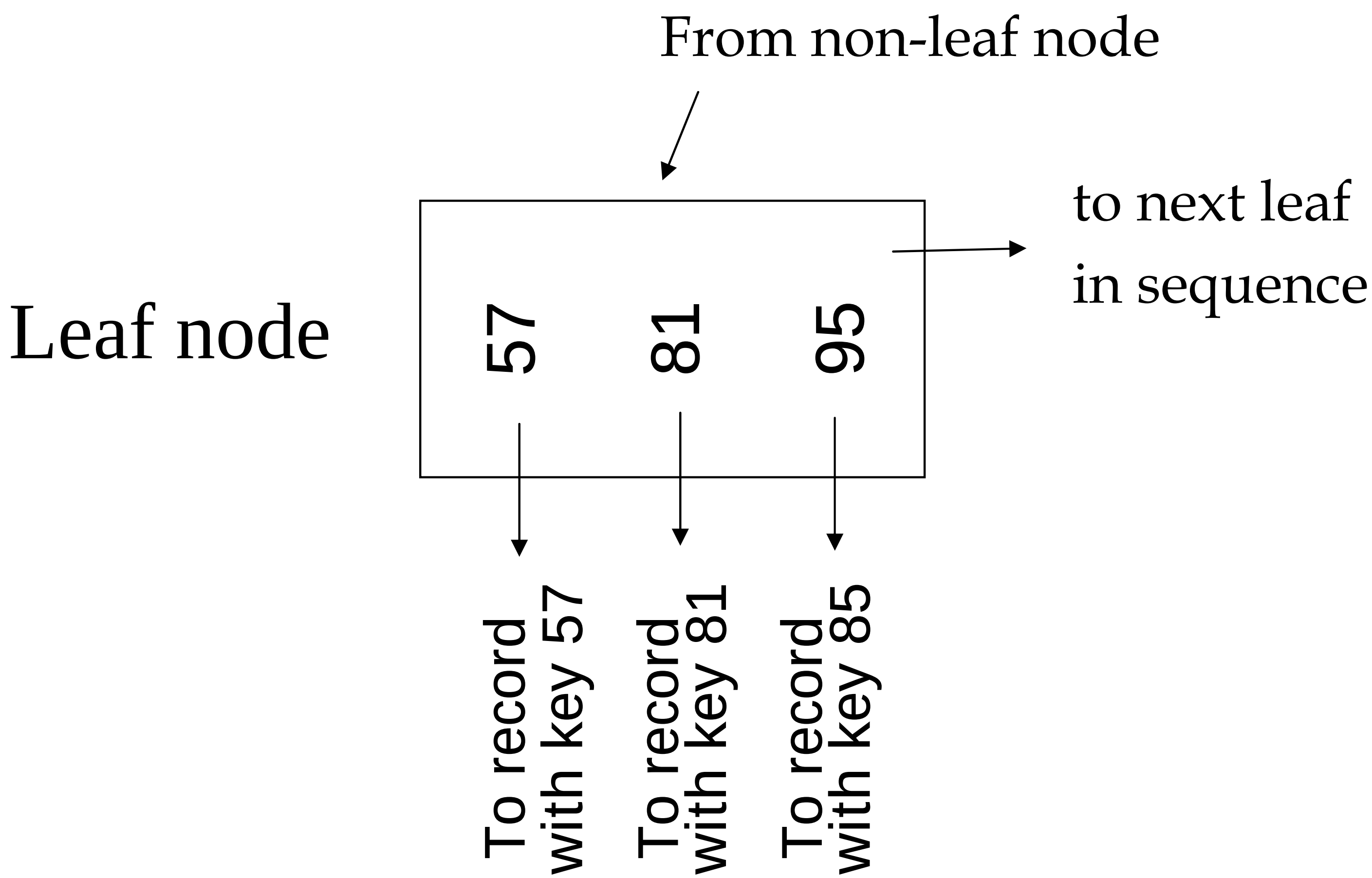
Non-leaf
(inner) node



to keys	to keys	to keys	to keys
< 57	57≤ k<81	81≤k<95	≥95

B+-Trees

Example



In Textbook's Notation

N=3

Leaf:

Non-leaf:

Don't want nodes to be too empty

- Size of nodes: $\begin{cases} n+1 \text{ pointers} \\ n \text{ keys} \end{cases}$

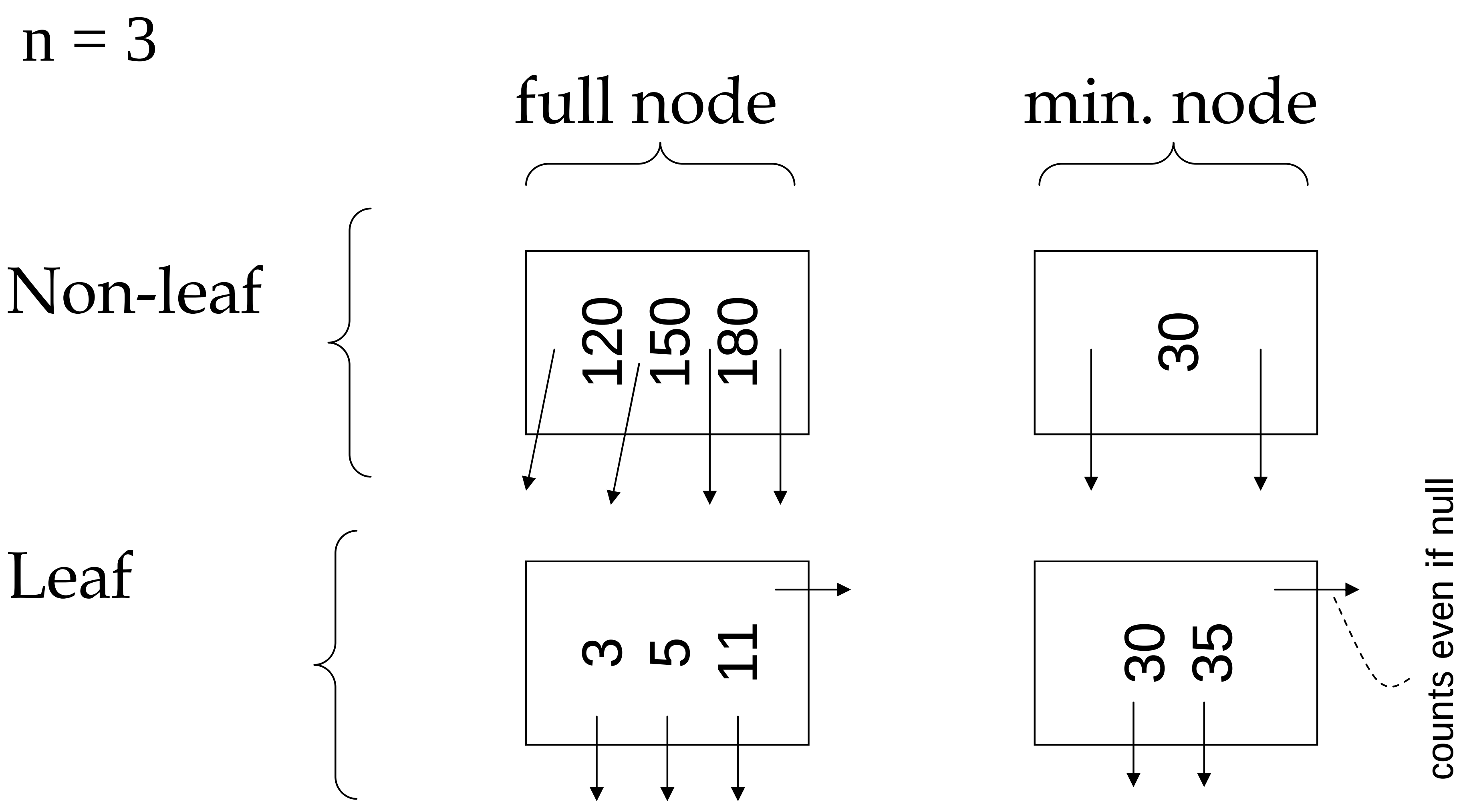
- Use at least

  Non-leaf: $\lceil (n+1)/2 \rceil$ pointers

  Leaf: $\lfloor (n+1)/2 \rfloor$ pointers to data

B+-Trees

Space utilization



B+-Trees

Space utilization

Number of pointers/keys for B-tree

	Max ptrs	Max keys	Min ptrs→data	Min keys
Non-leaf (non-root)	n+1	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil$- 1
Leaf (non-root)	n+1	n	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root	n+1	n	1	1

B+-Trees

Equality Queries

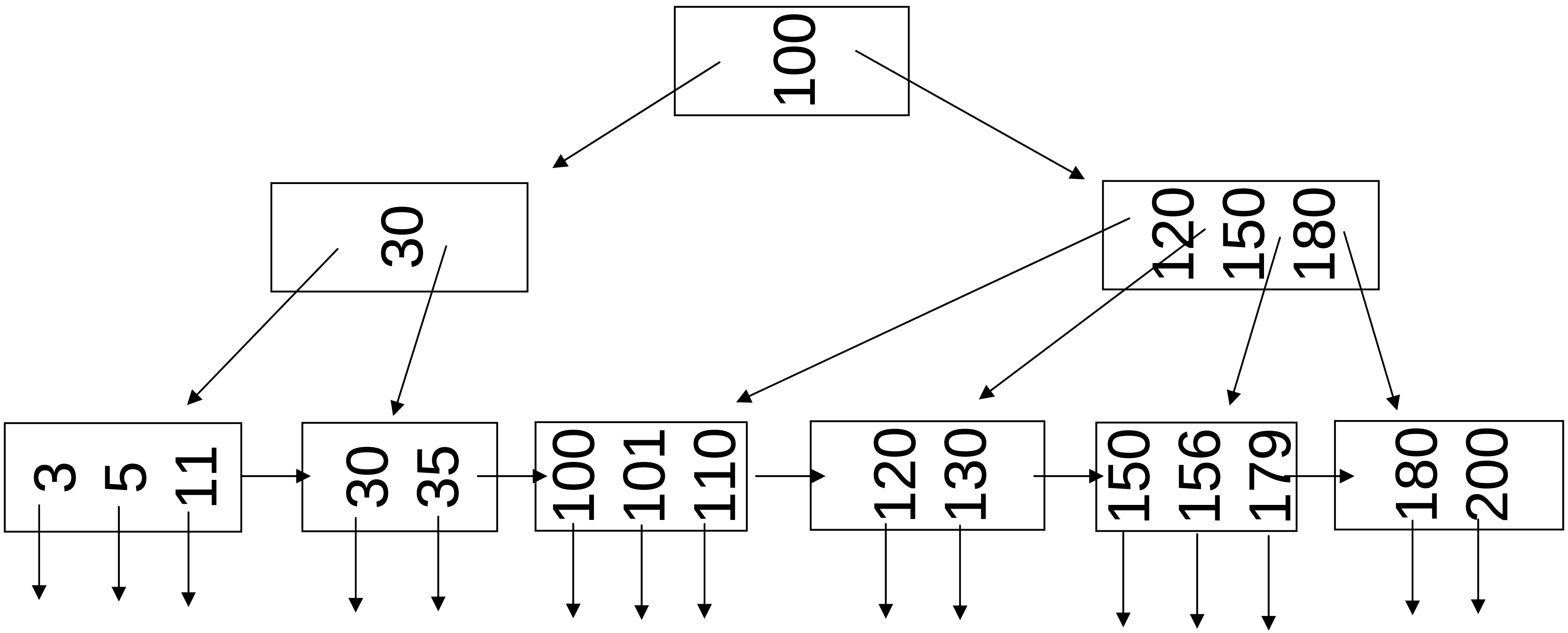
- To search for key k, start from root.
- At a given node, find "nearest key" k_i and follow left (p_i) or right (p_{i+1}) pointer depending on comparison of k and k_i.
- Continue, until leaf node reached.
- Explores one path from root to leaf node.
- Height of B-tree is $O(\log_{n/2} N)$ where N: number of records indexed

→ runtime complexity $O(\log N)$

B+-Trees

Example

Root n = 3



To Discuss

- How to construct a B+-Tree
  - Insertion?
  - Deletion?