Database System Architecture

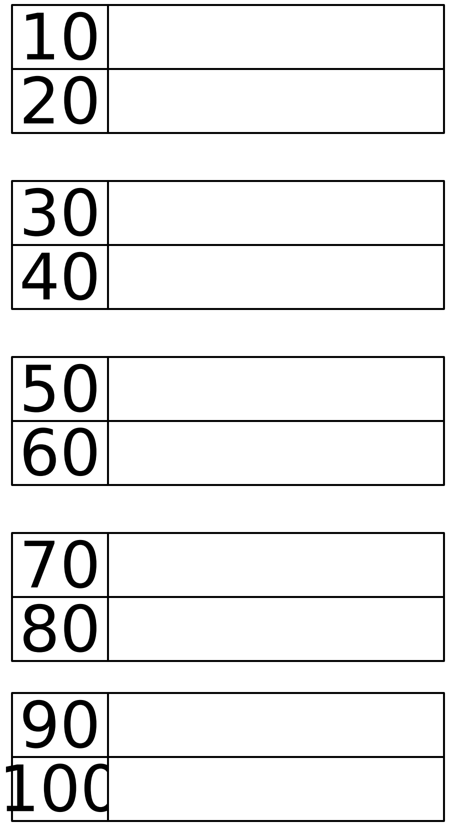
Index Structures

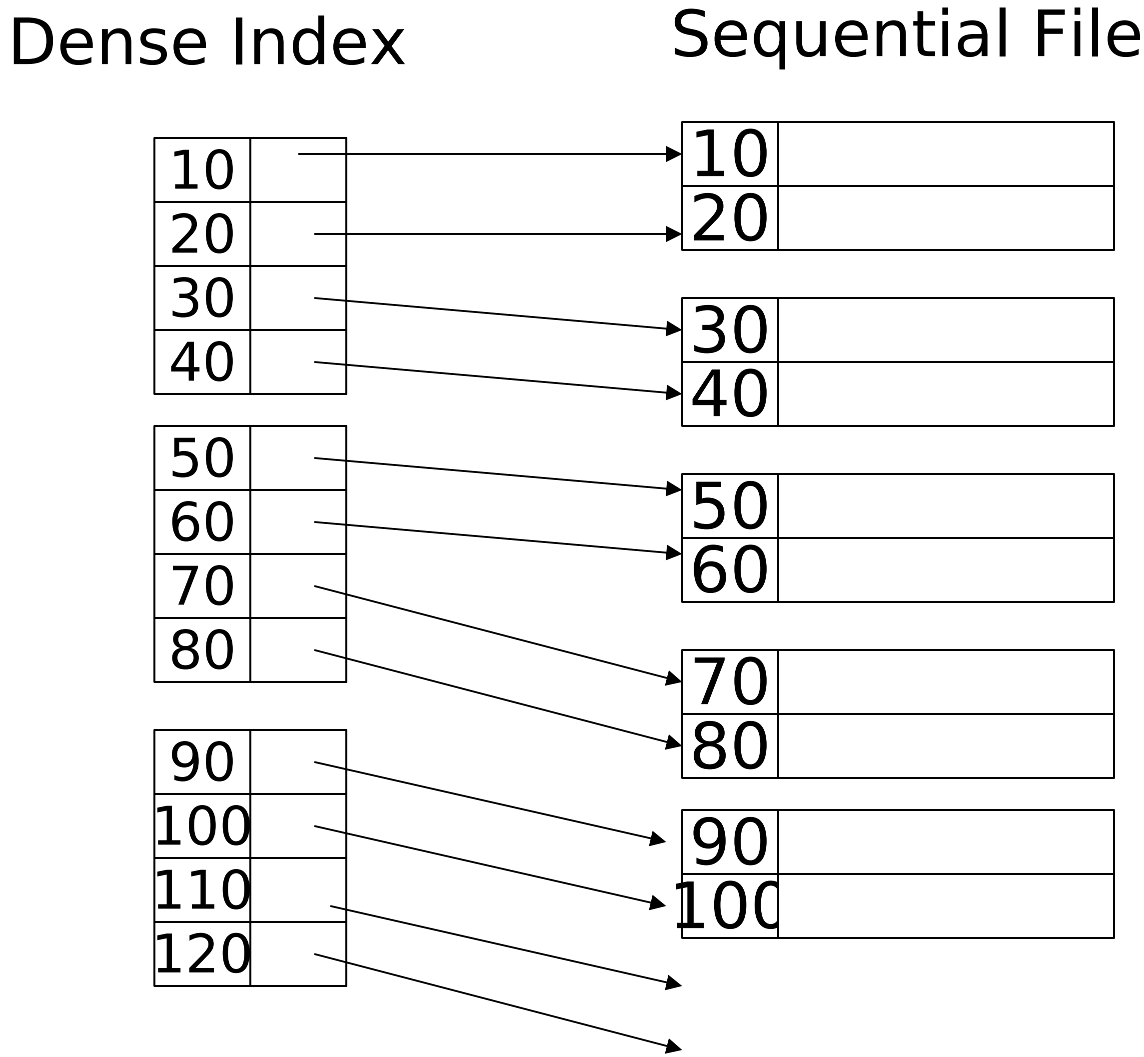
Hector Garcia-Molina
Stijn Vansummeren

Index structure

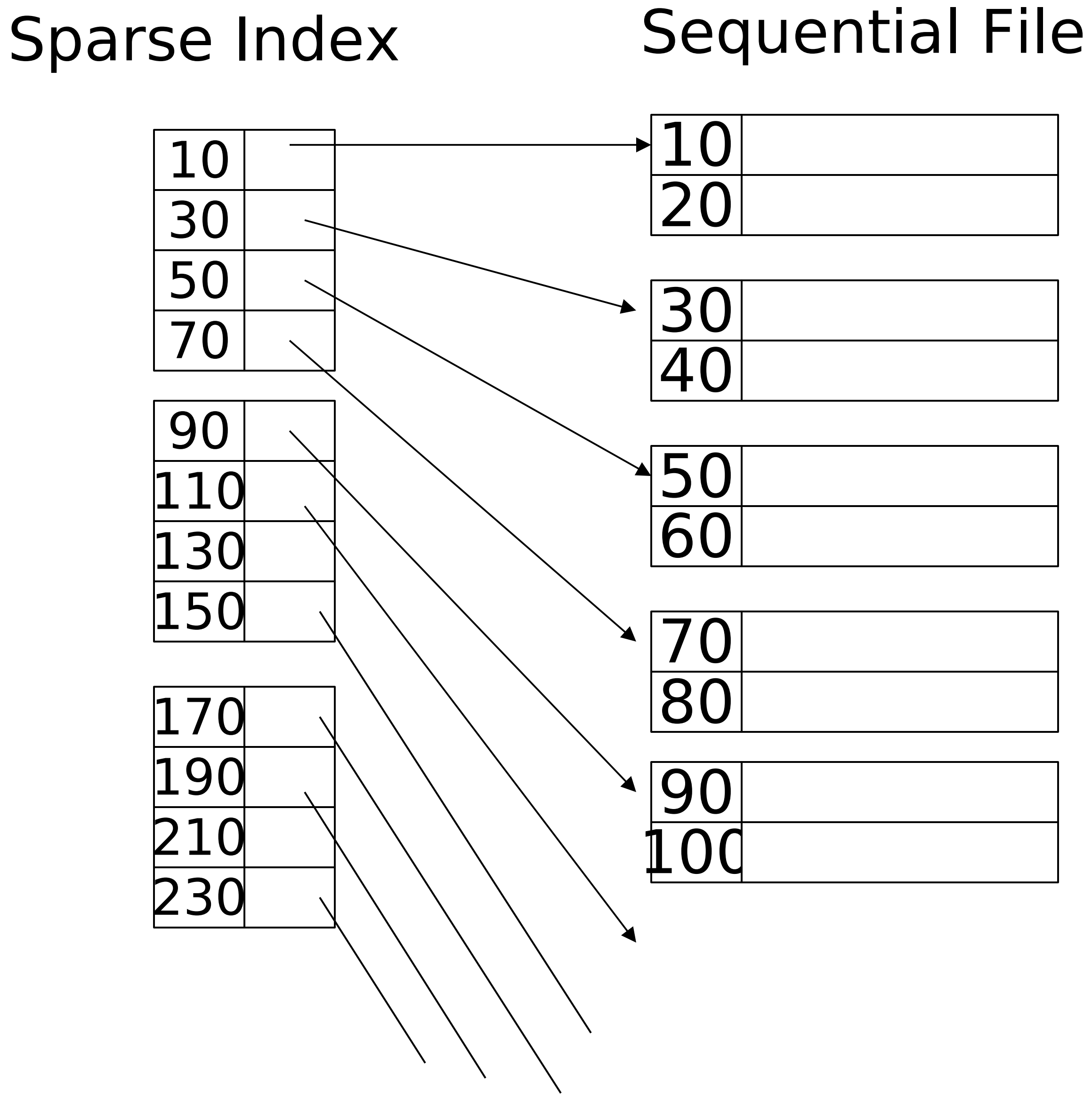
- Any data structure that takes as input a search key and ***efficiently*** returns the collection of matching records

Sequential File

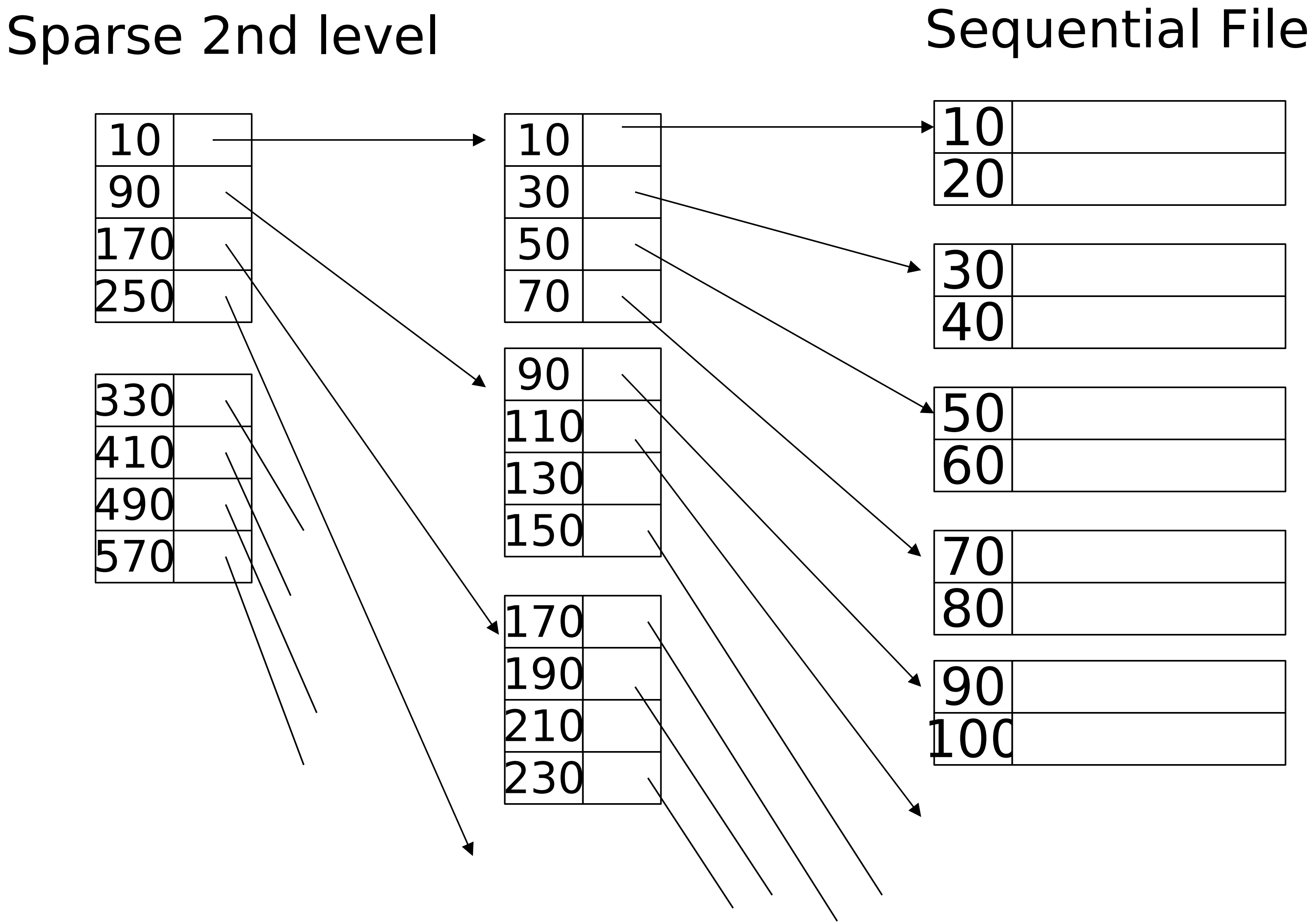




Dense Index
Sequential File
10
20
30
40
50
60
70
80
90
100
110
120
10
20
30
40
50
60
70
80
90
100



Sparse Index
Sequential File
10
30
50
70
90
110
130
150
170
190
210
230
10
20
30
40
50
60
70
80
90
100



Sparse 2nd level
Sequential File
10
90
170
250
330
410
490
570
10
30
50
70
90
110
130
150
170
190
210
230
10
20
30
40
50
60
70
80
90
100

Question:

- Can we build a dense, 2nd level index for a dense index?

Sparse vs. Dense Tradeoff

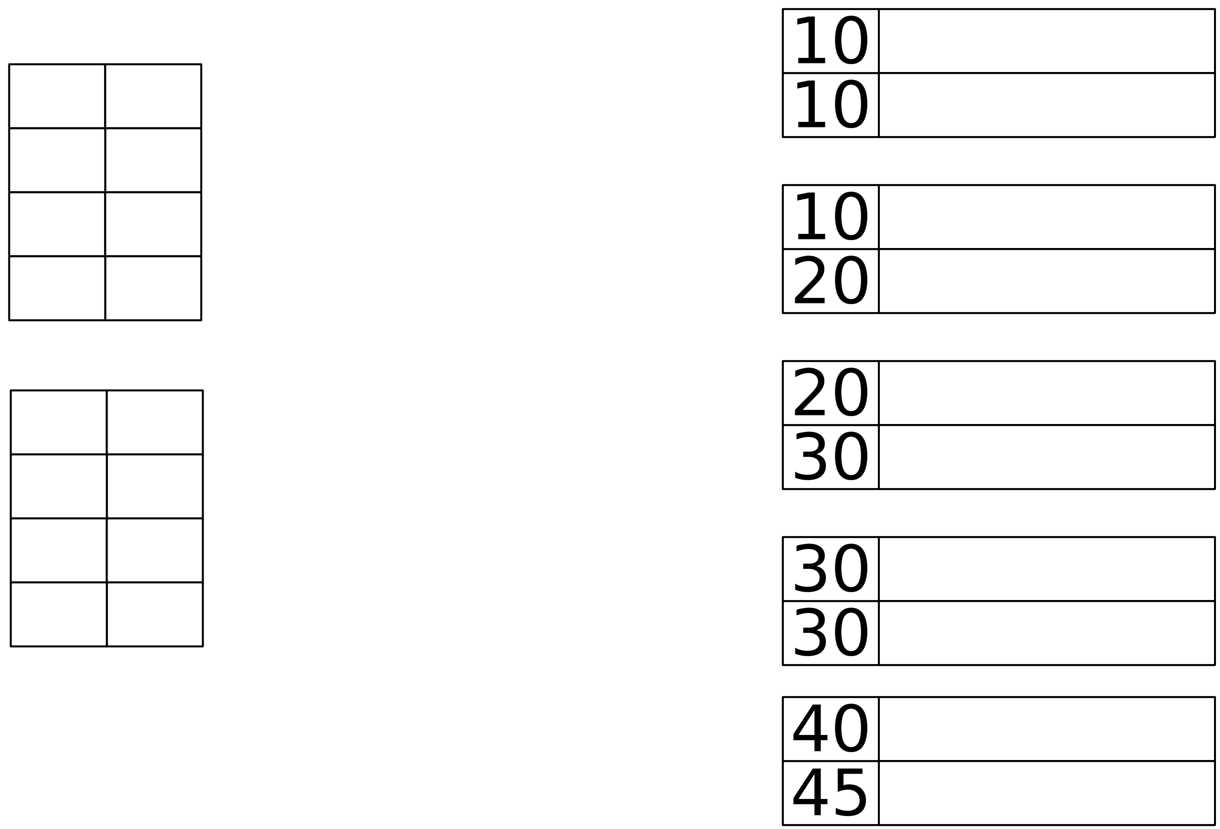
- Sparse: Less index space per record
  can keep more of index in
  memory
- Dense: Can tell if any record exists
  without accessing file

(Later:
  - sparse better for insertions
  - dense needed for secondary indexes)

Next:

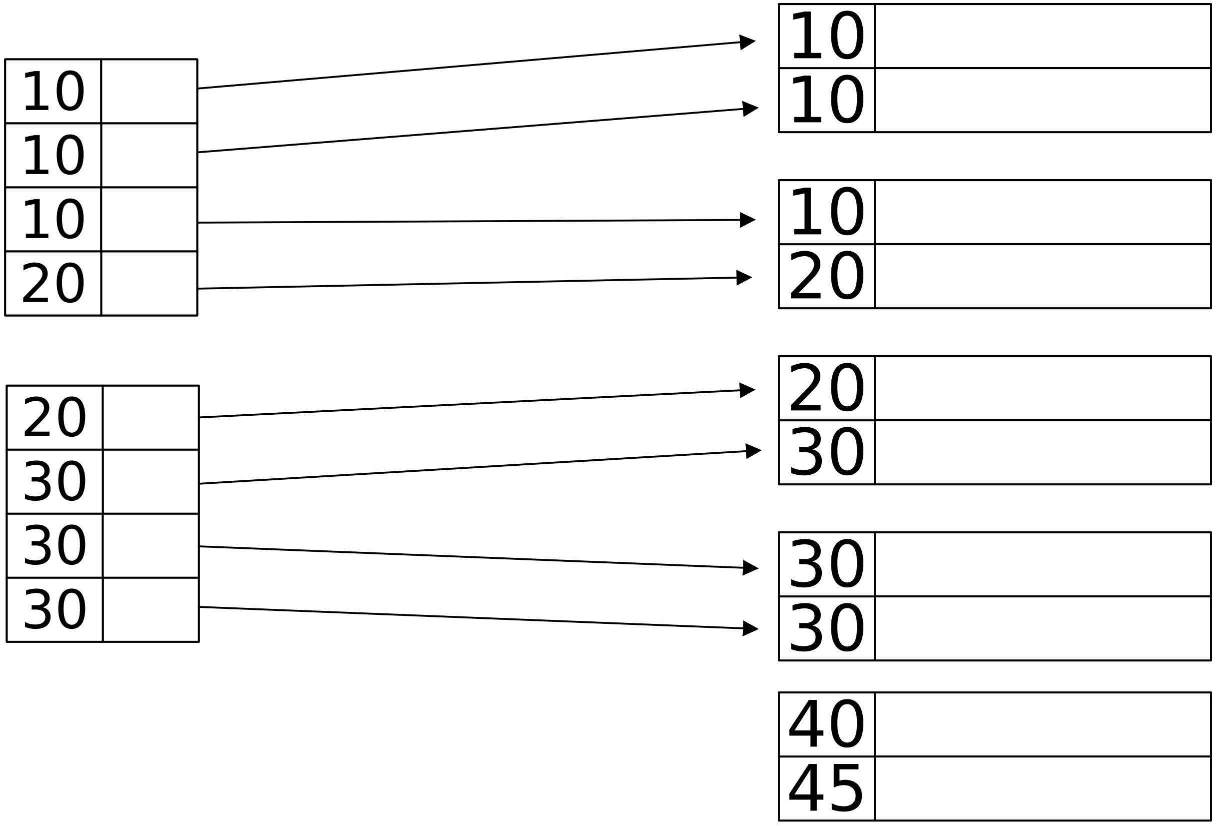
- Duplicate keys
- Deletion/Insertion
- Secondary indexes

Duplicate keys



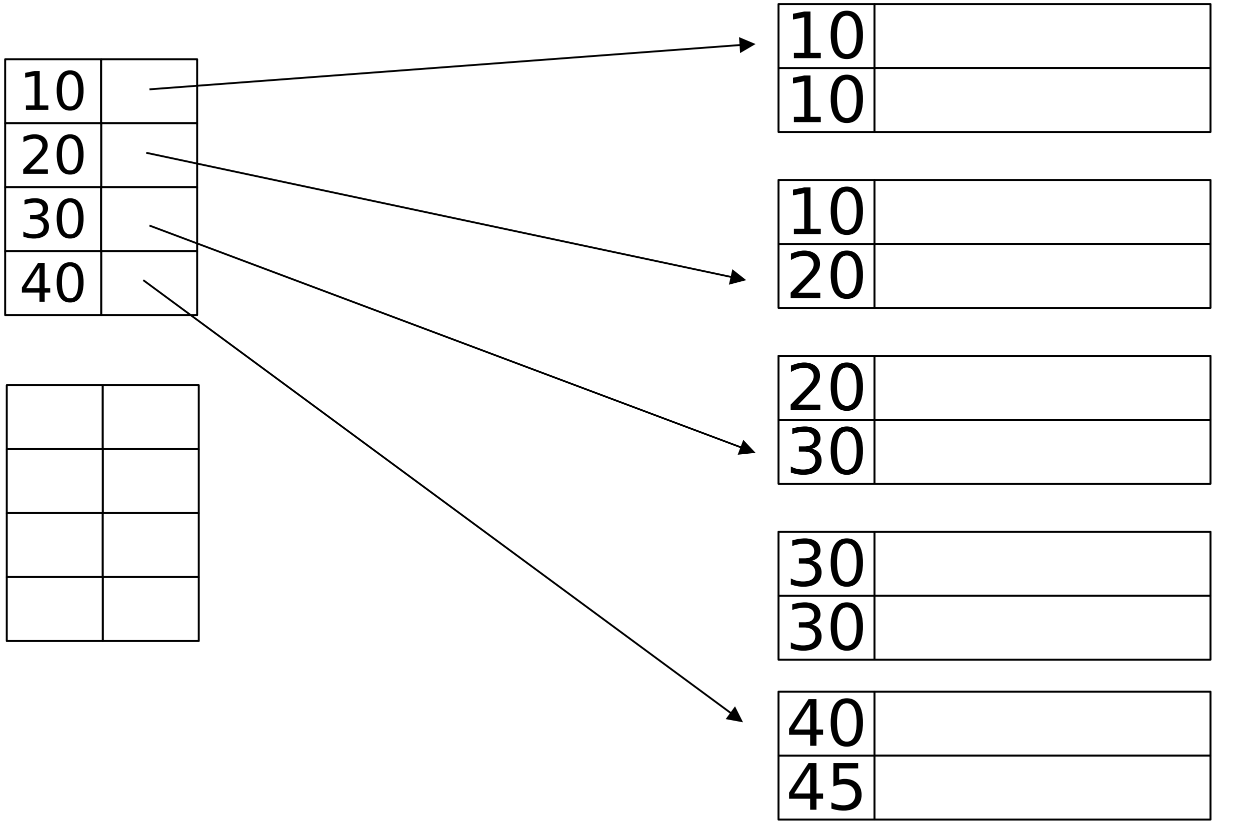
Duplicate keys

Dense index, one way to implement?



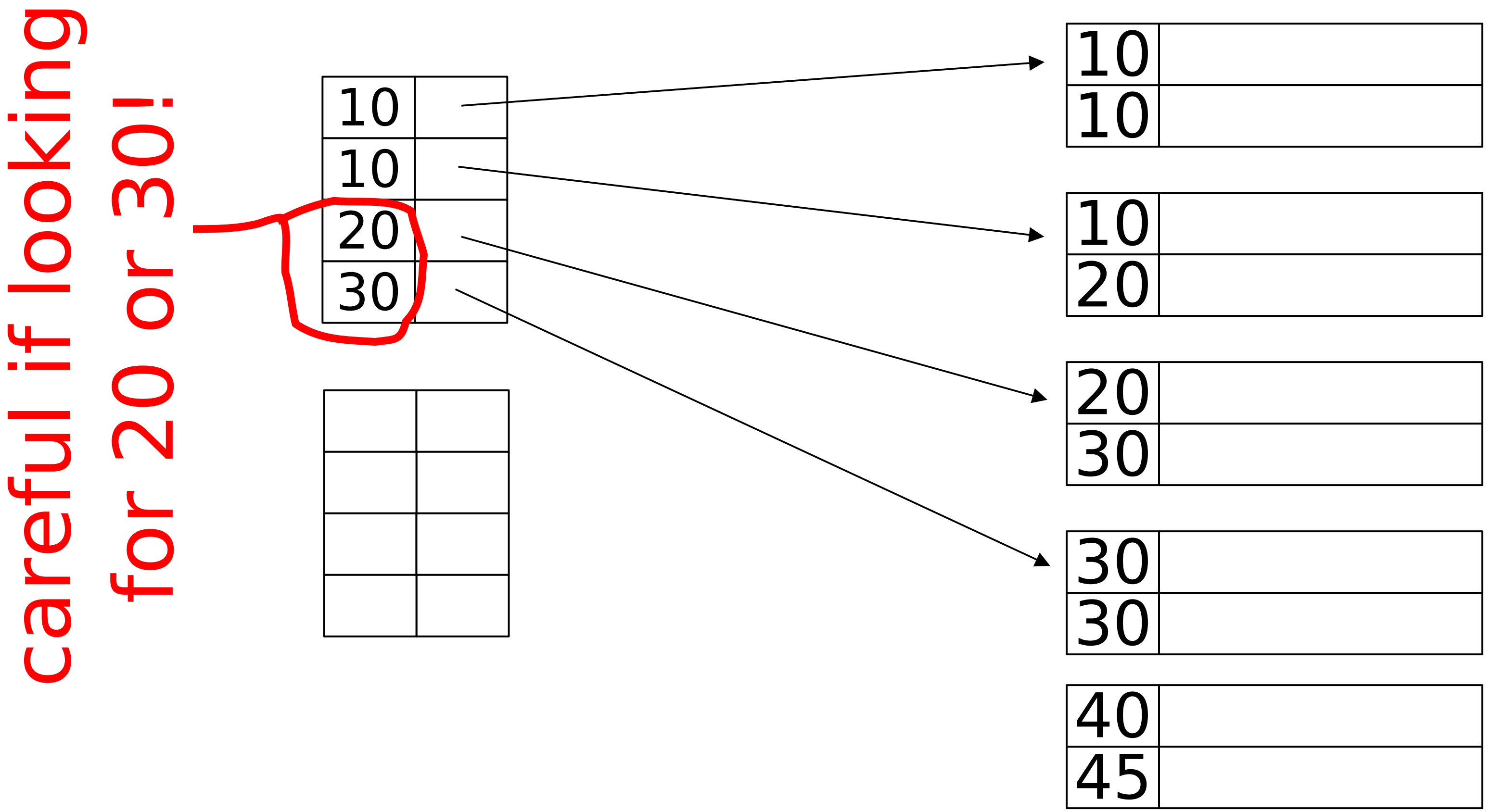
Duplicate keys

Dense index, better way?



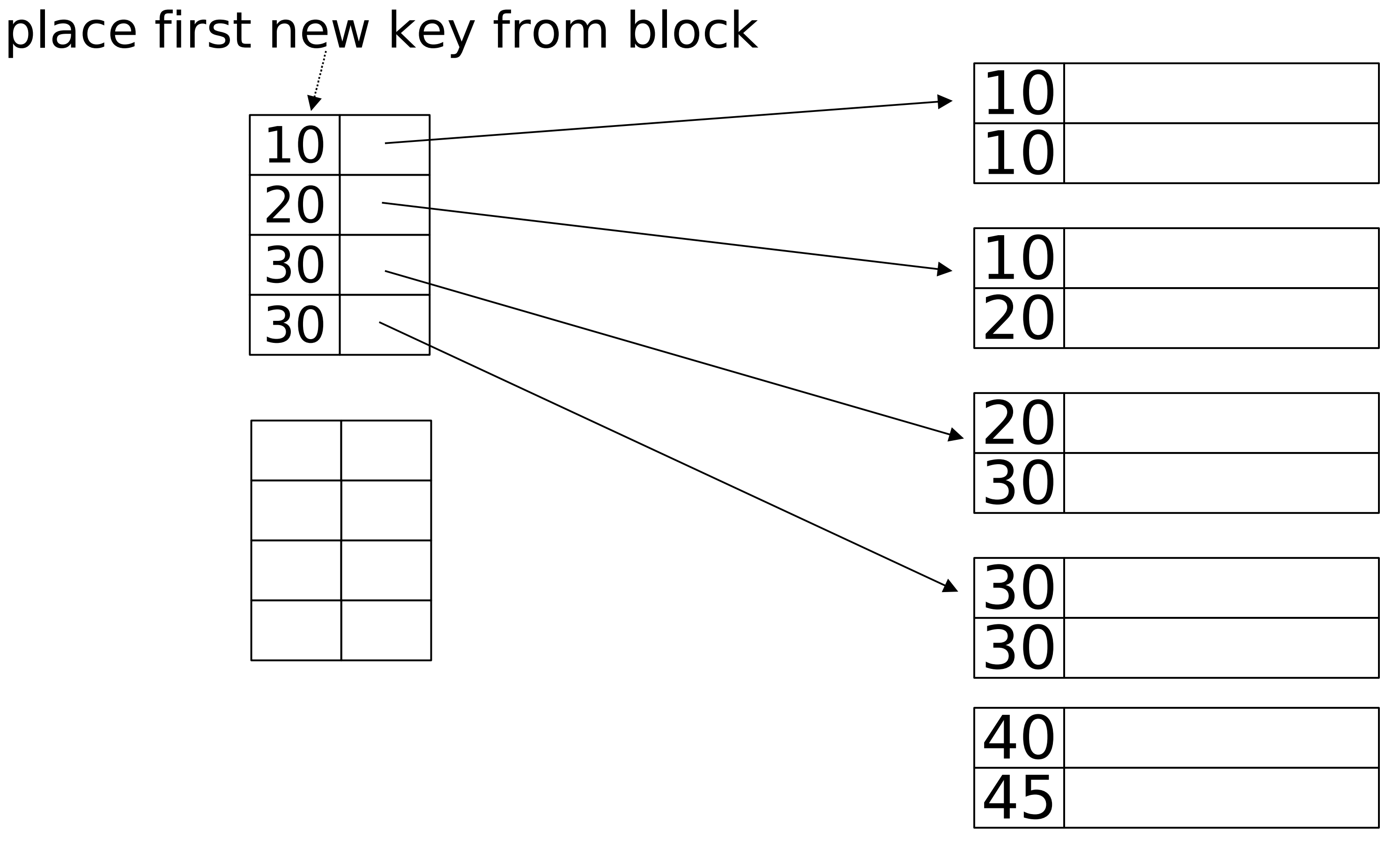
Duplicate keys

Sparse index, one way?

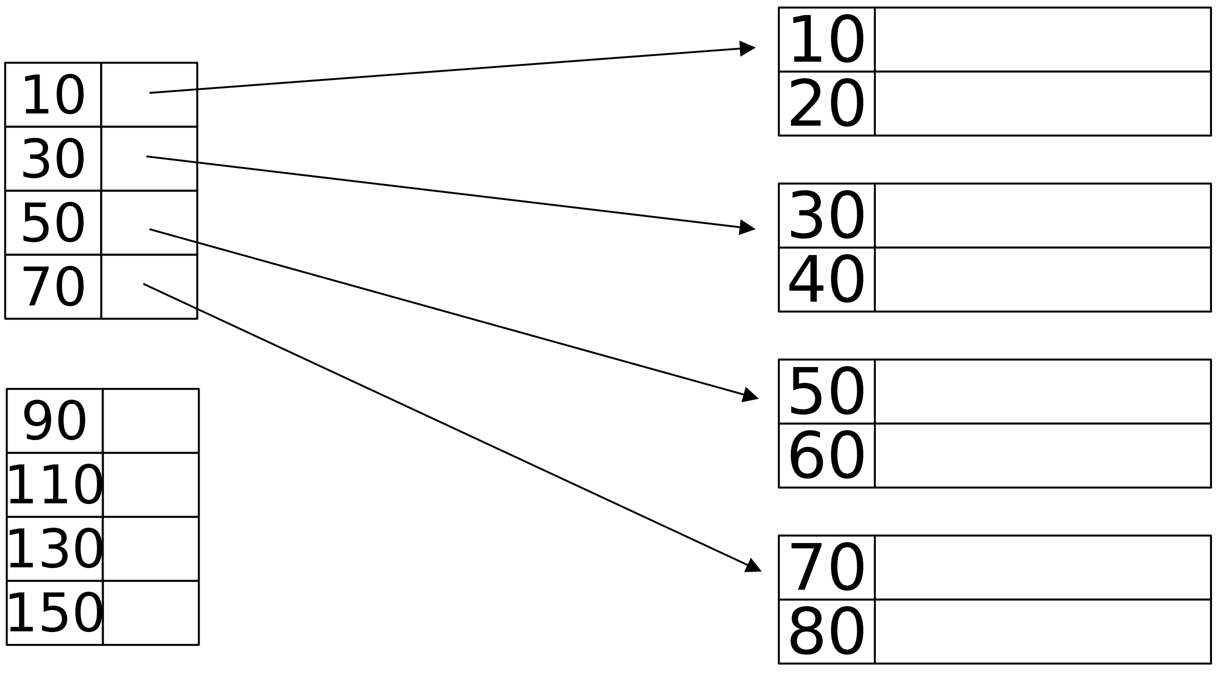


Duplicate keys

Sparse index, another way?

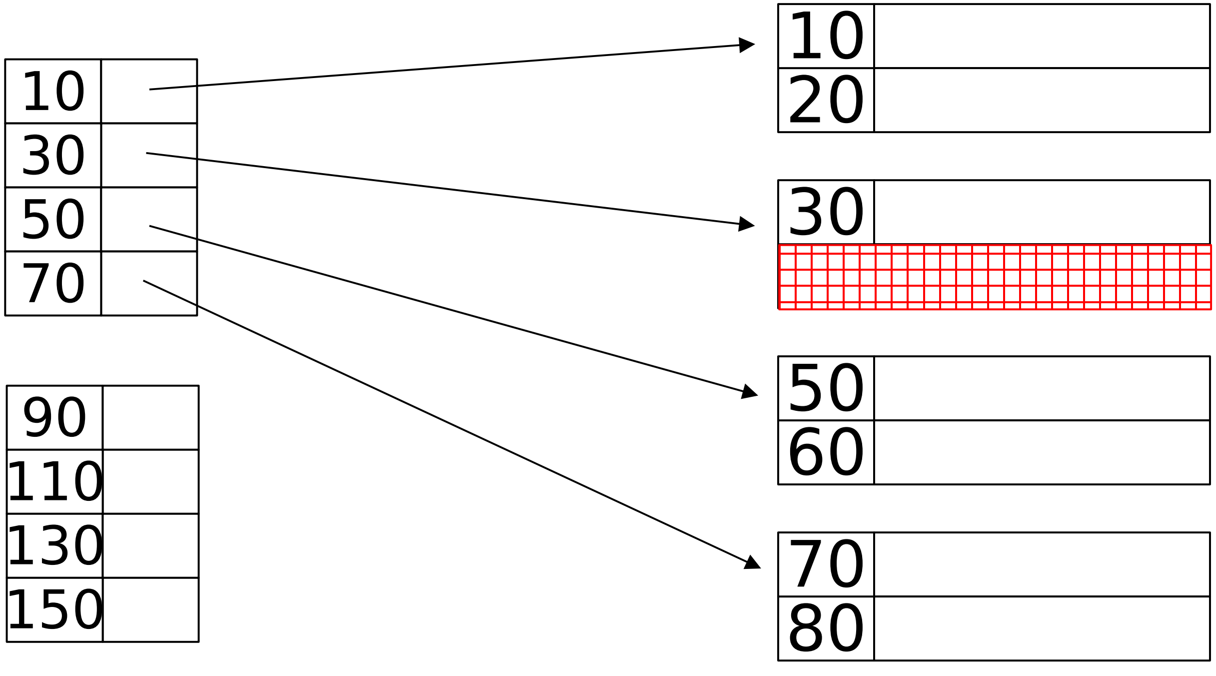


Deletion from sparse index



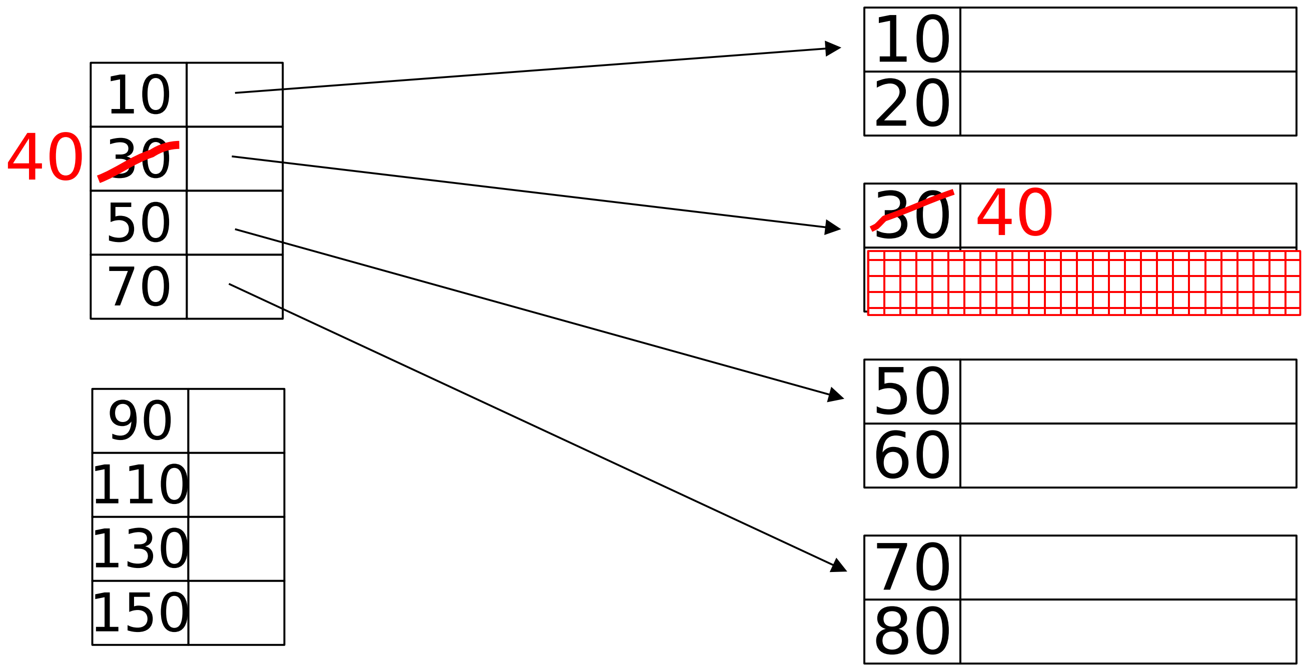
Deletion from sparse index

- delete record 40



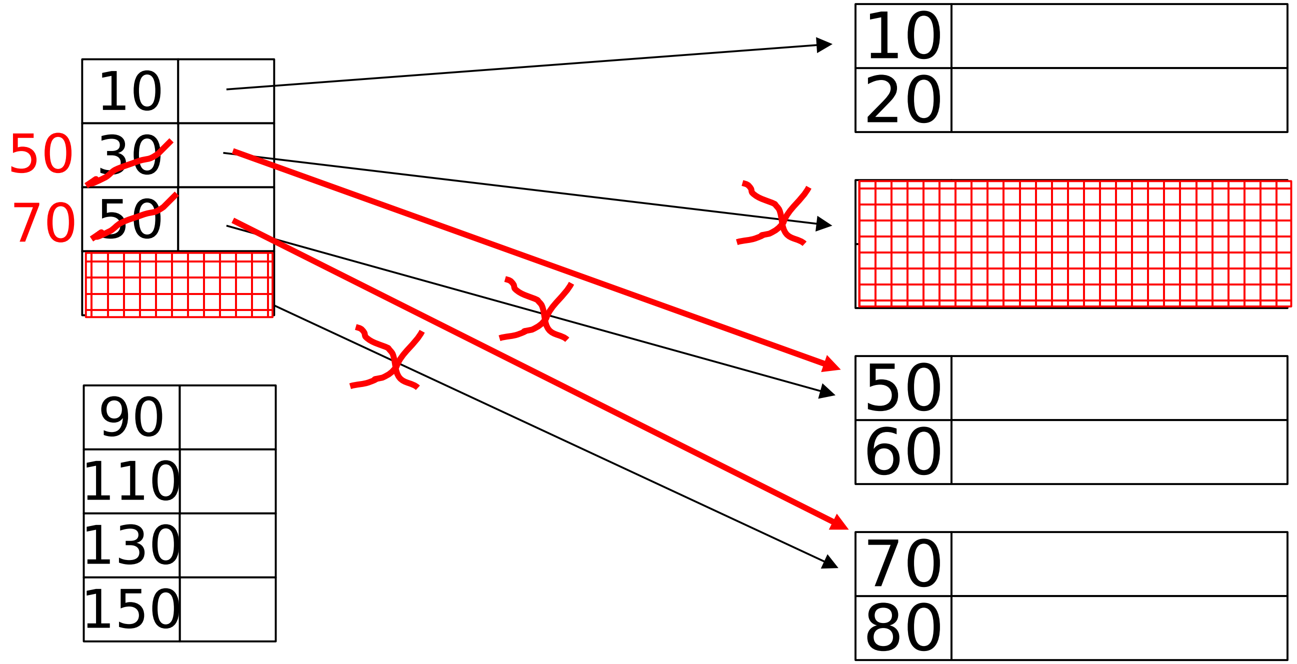
Deletion from sparse index

- delete record 30

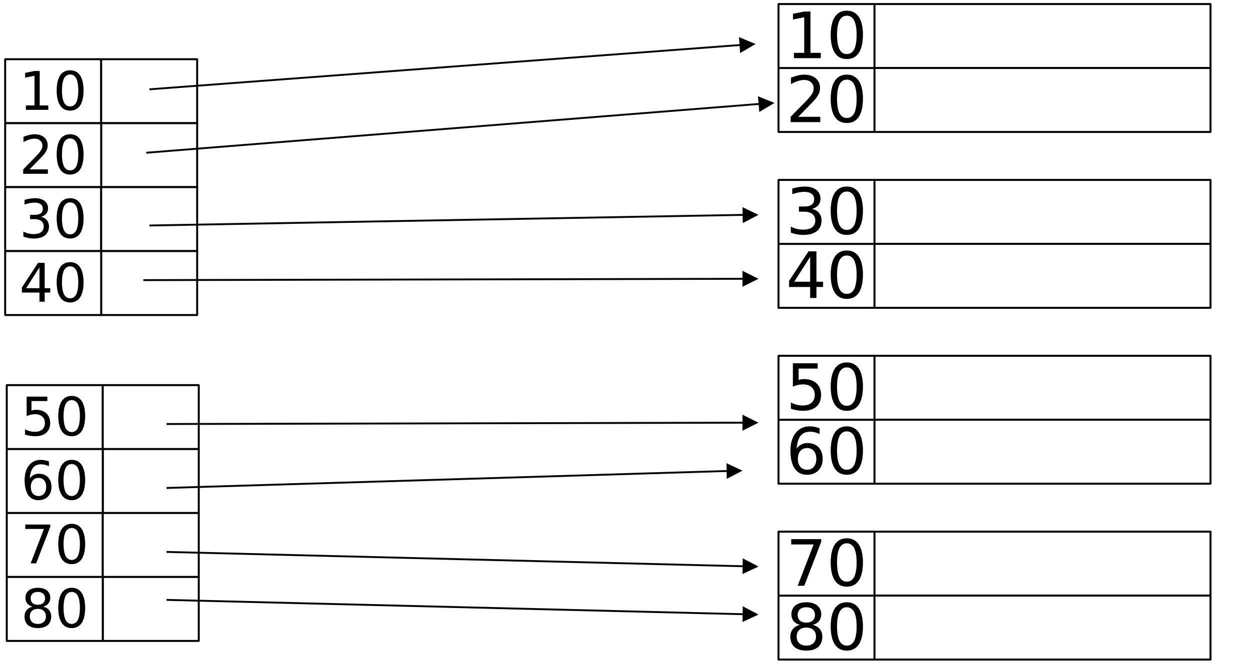


Deletion from sparse index

- delete records 30 & 40

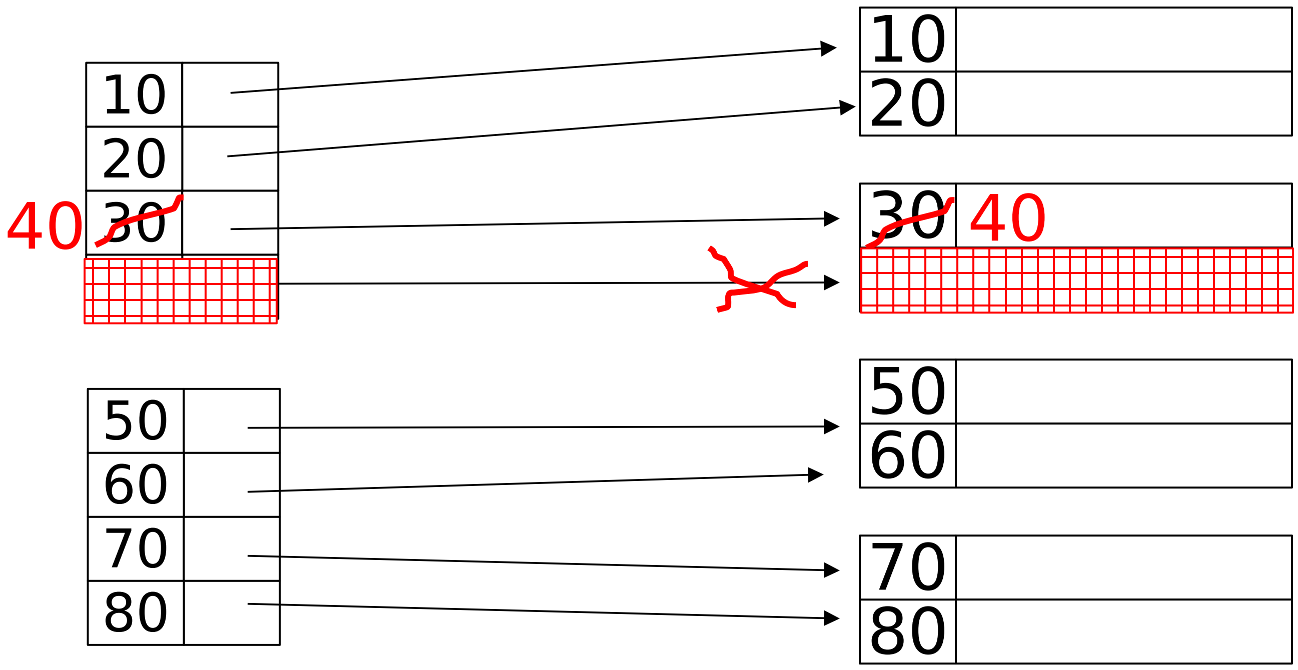


Deletion from dense index

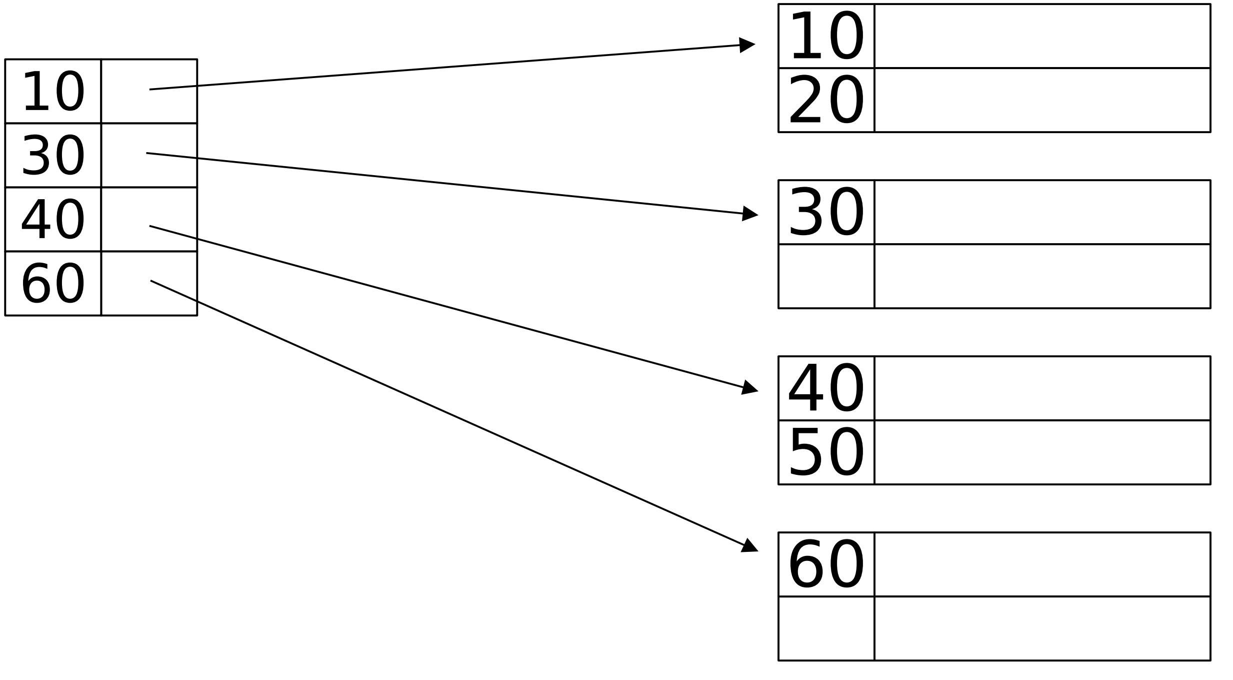


Deletion from dense index

- delete record 30

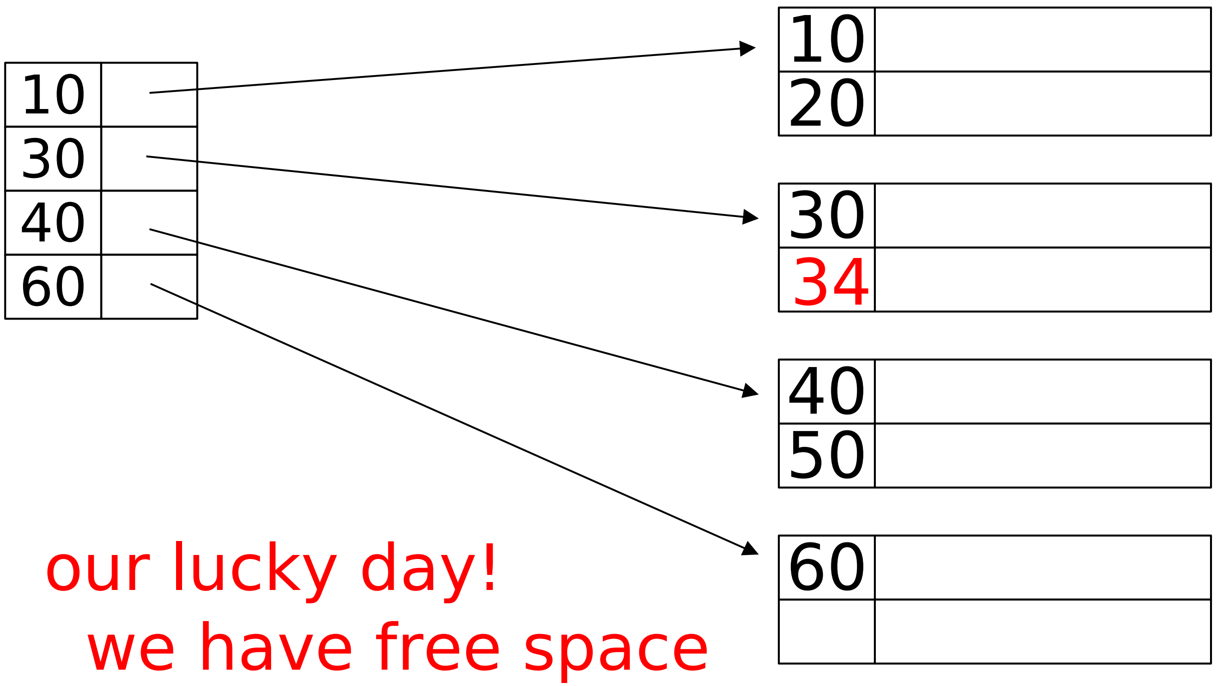


Insertion, sparse index case



Insertion, sparse index case

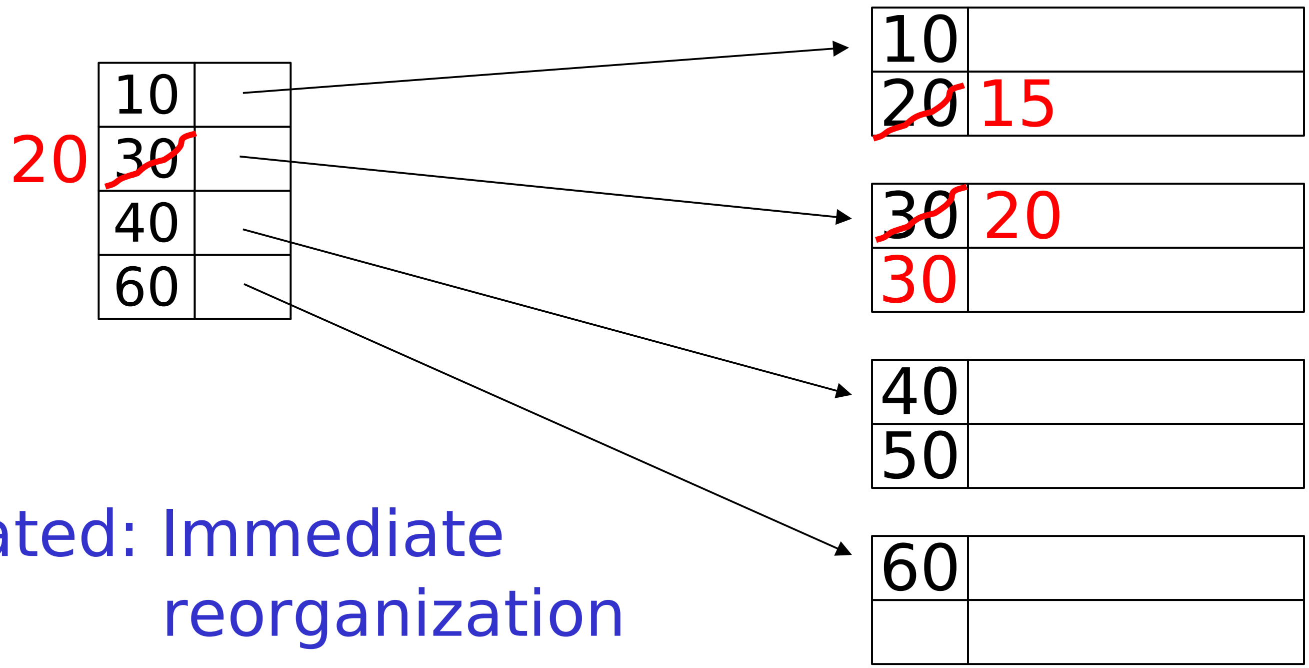
- insert record 34



our lucky day!
we have free space
where we need it!

Insertion, sparse index case

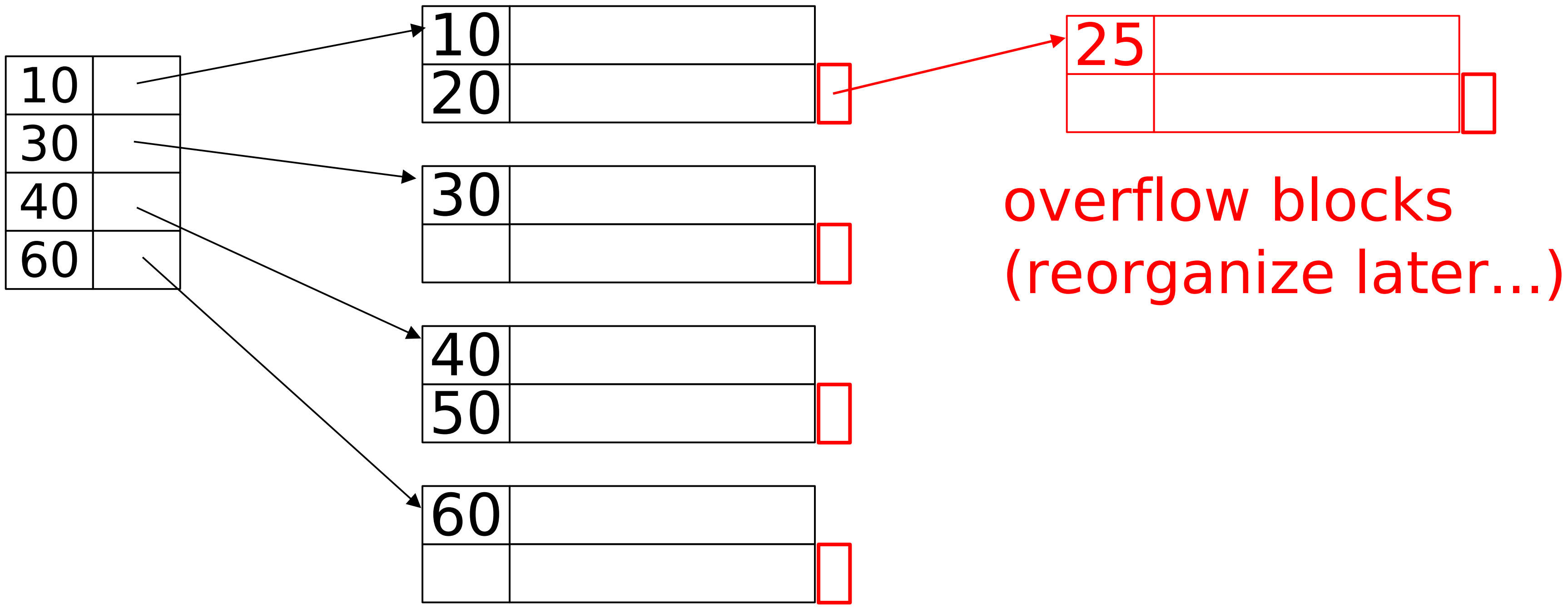
- insert record 15



- Illustrated: Immediate reorganization
- Variation:
  - insert new block (chained file)
  - update index

Insertion, sparse index case

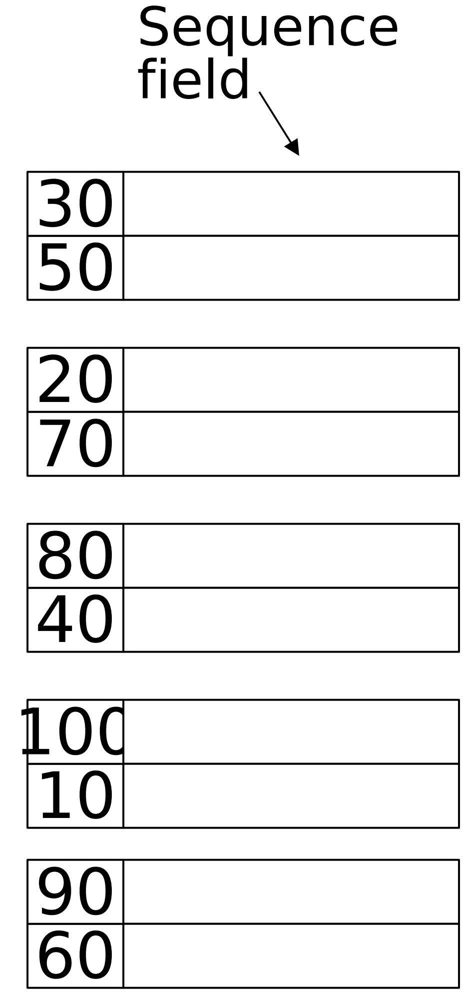
- insert record 25



Insertion, dense index case

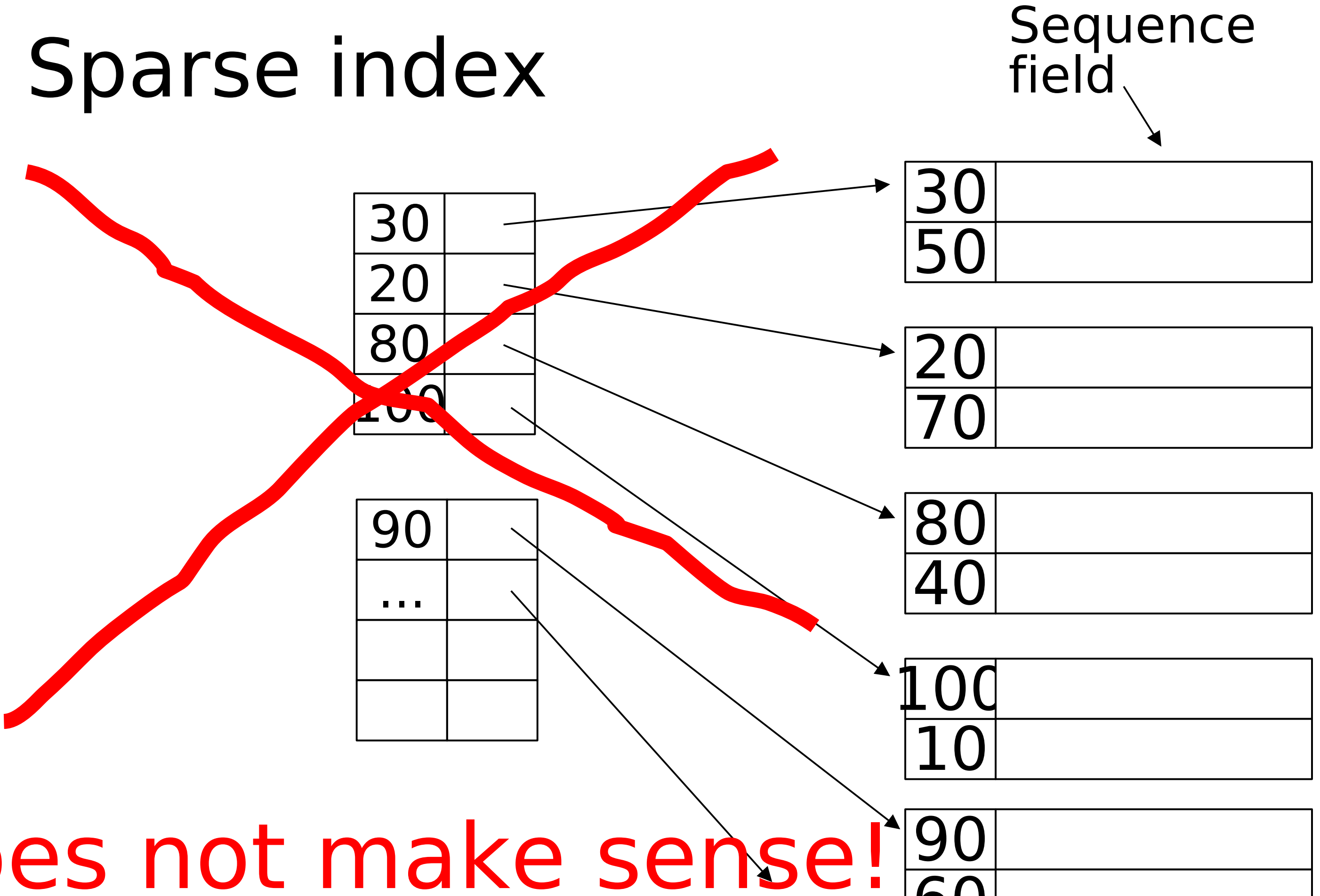
- Similar
- Often more expensive . . .

Secondary indexes



Secondary indexes

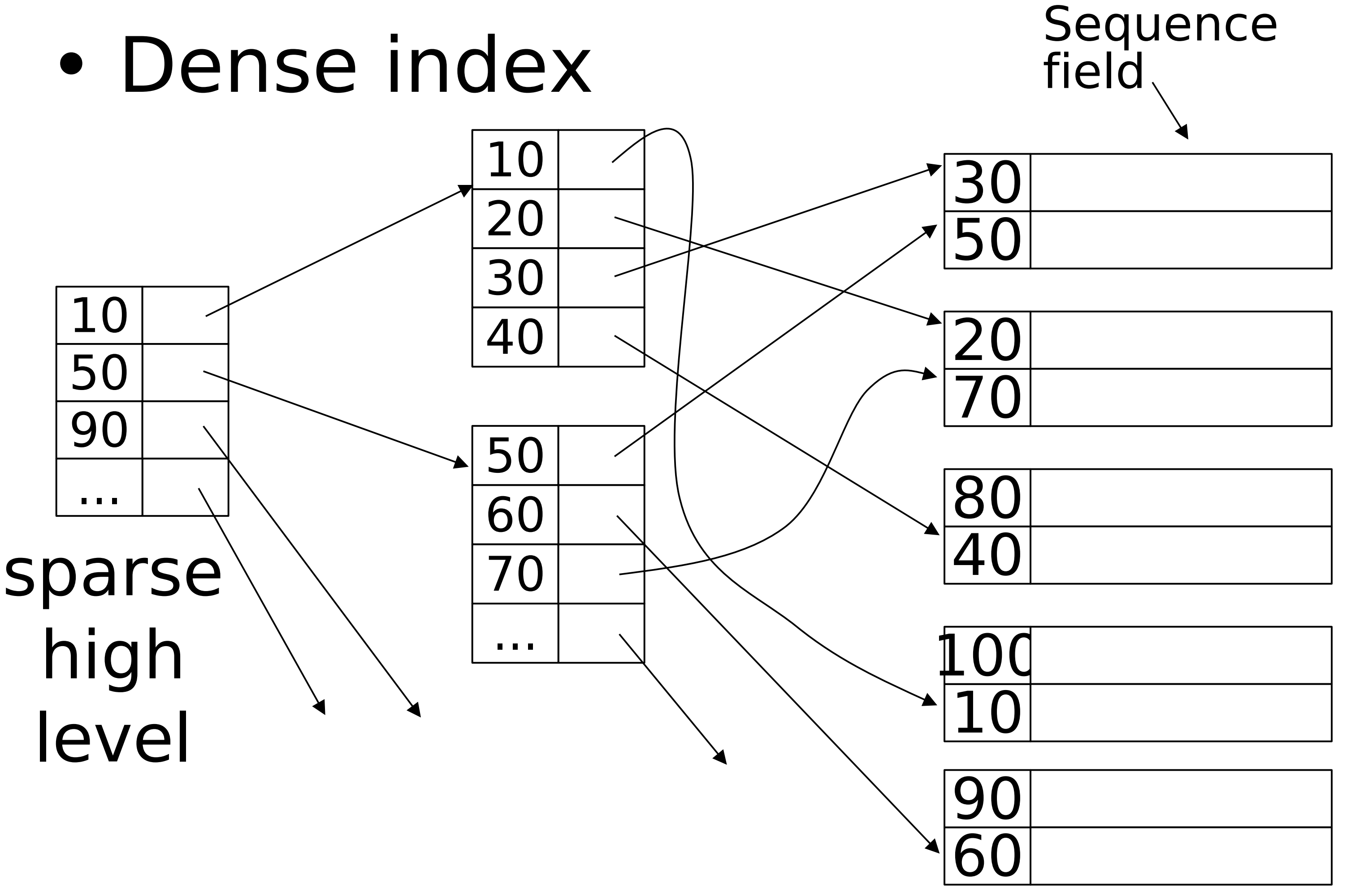
- Sparse index



does not make sense!

Secondary indexes

- Dense index



With secondary indexes:

- Lowest level is dense
- Other levels are sparse

Also: Pointers are record pointers

(not block pointers; not computed)

Duplicate values & secondary indexes

20	
10	

20	
40	

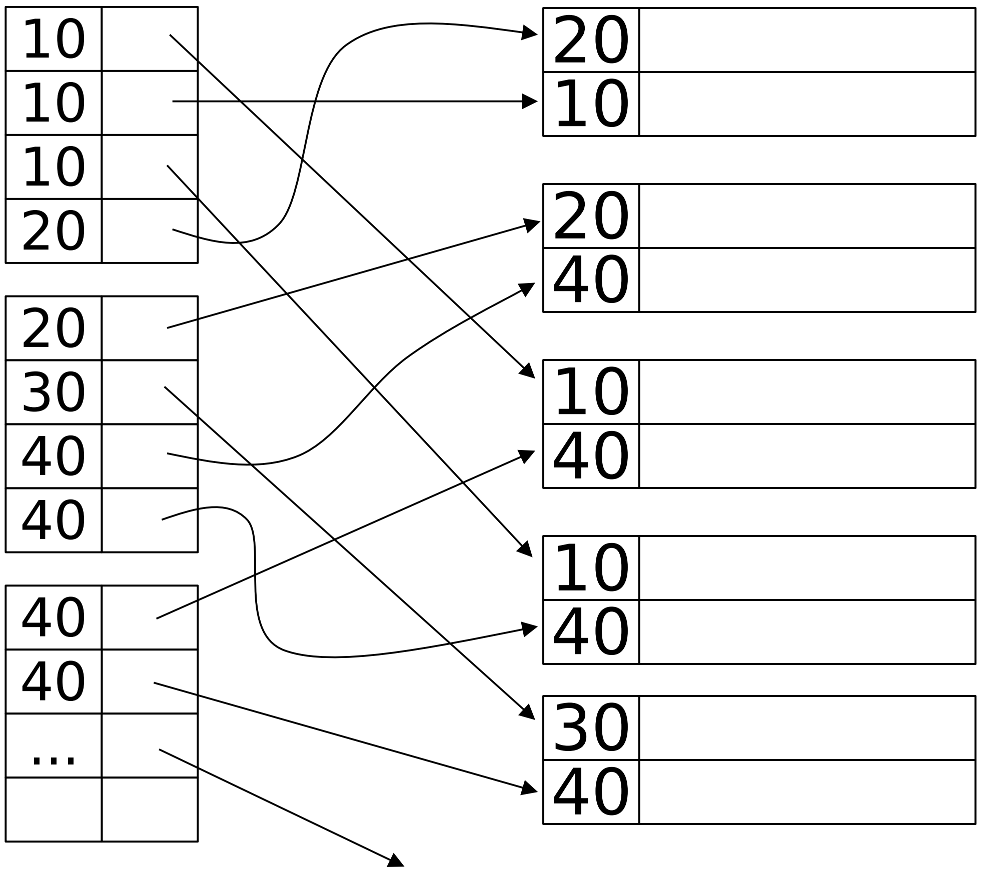
10	
40	

10	
40	

30	
40	

Duplicate values & secondary indexes

one option...



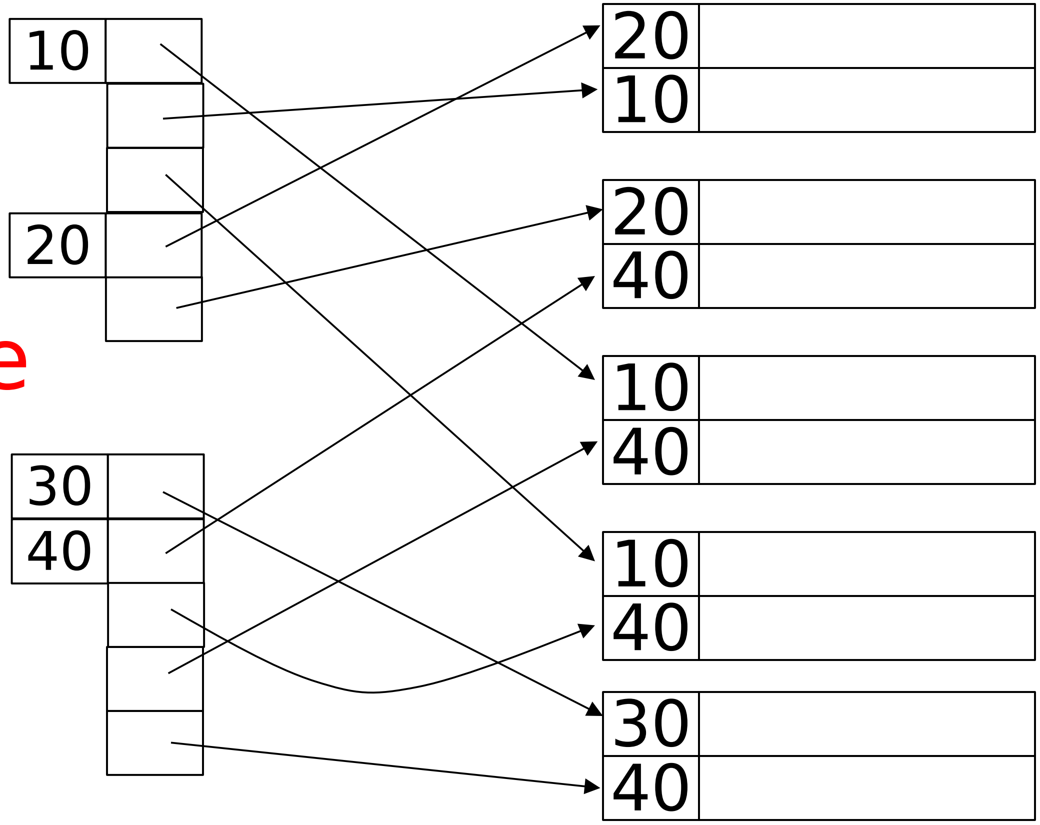
Problem:

excess overhead!

- disk space
- search time

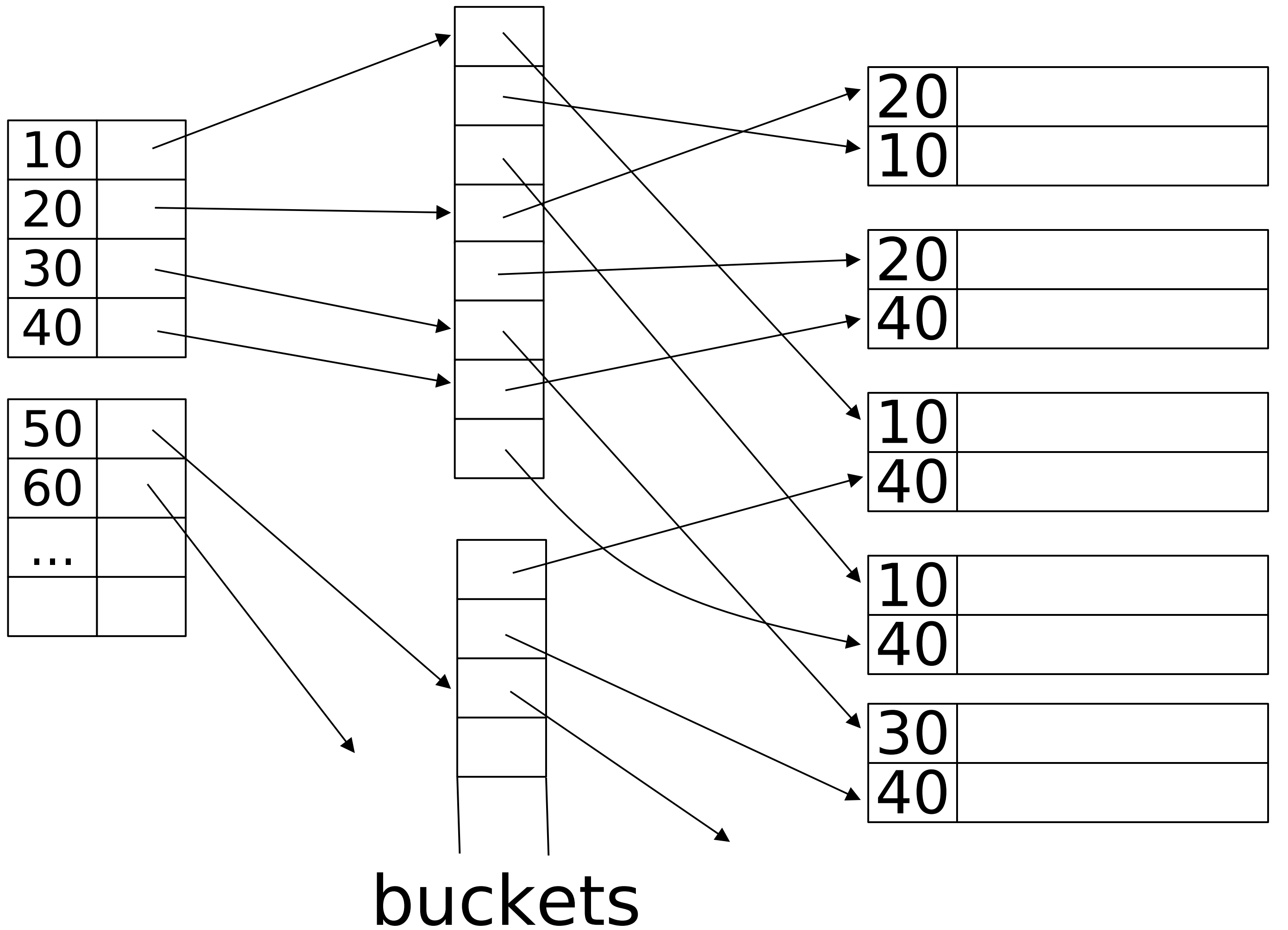
Duplicate values & secondary indexes

another option...



Problem: variable size records in index!

Duplicate values & secondary indexes

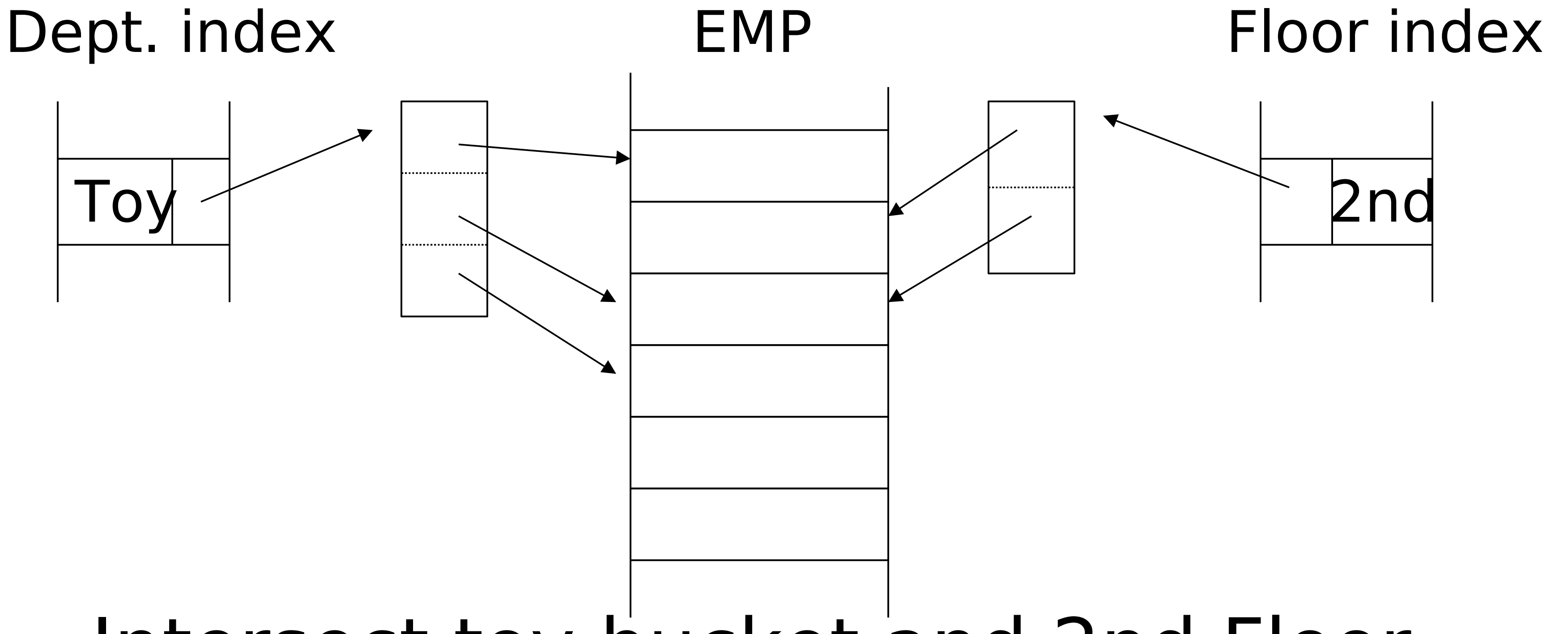


Why “bucket” idea is useful

Indexes	Records
Name: primary	EMP(name,dept,floor,...)
Dept: secondary	
Floor: secondary	

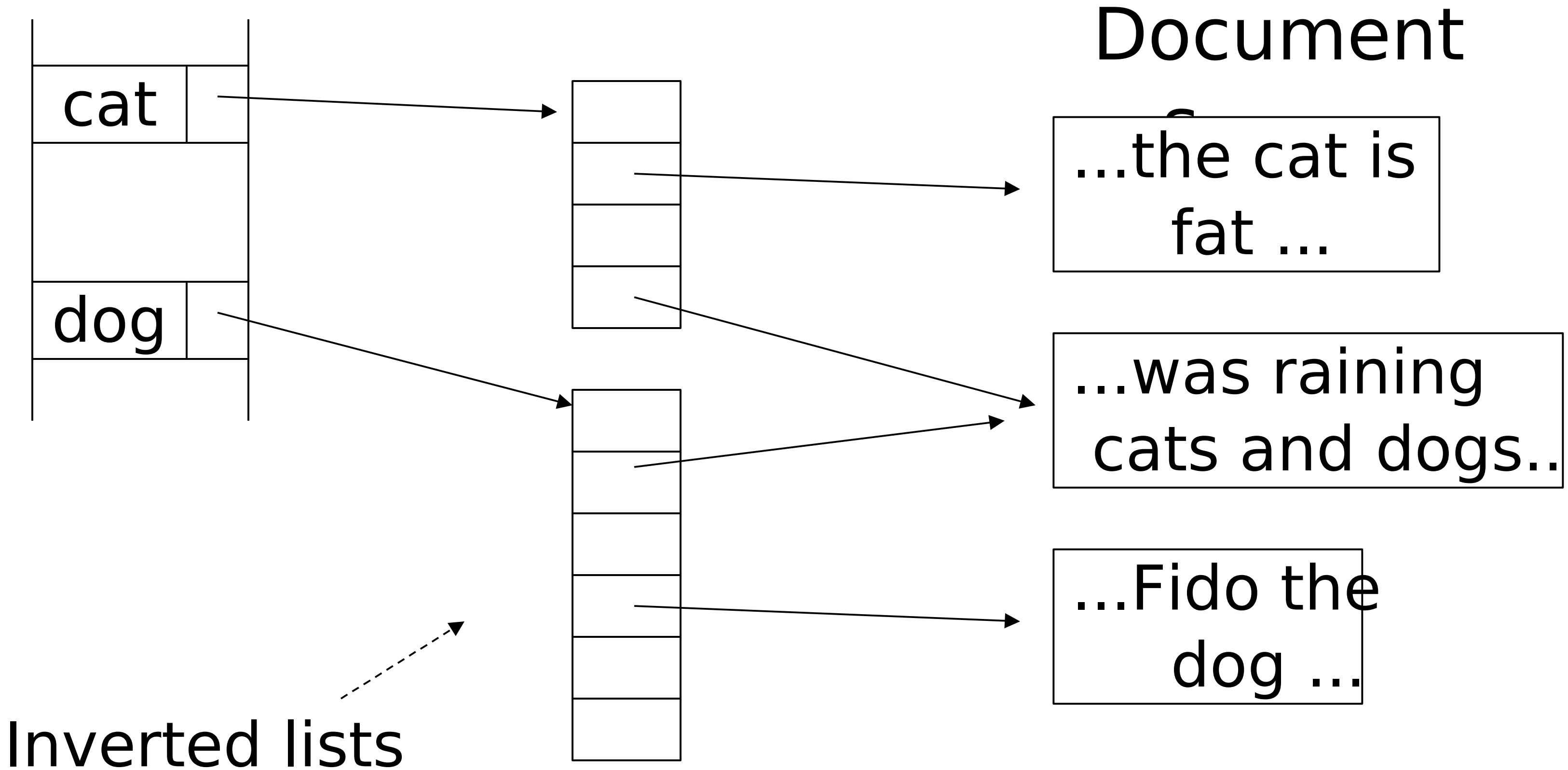
Query: Get employees in

(Toy Dept) $\wedge$ (2nd floor)



$\rightarrow$ Intersect toy bucket and 2nd Floor bucket to get set of matching EMP's

This idea used in text information retrieval



IR QUERIES

- Find articles with “cat” and “dog”
- Find articles with “cat” or “dog”
- Find articles with “cat” and not “dog”
- Find articles with “cat” in title
- Find articles with “cat” and “dog” within 5 words

Summary so far

- Conventional index
  - Basic Ideas: sparse, dense, multi-level…
  - Duplicate Keys
  - Deletion/Insertion
  - Secondary indexes
    - Buckets of Postings List

Outline/summary

- Conventional Indexes
    - Sparse vs. dense
    - Primary vs. secondary
- B trees --> Next
    - B+trees vs. indexed sequential
- Hashing schemes

Conventional indexes

Advantage:

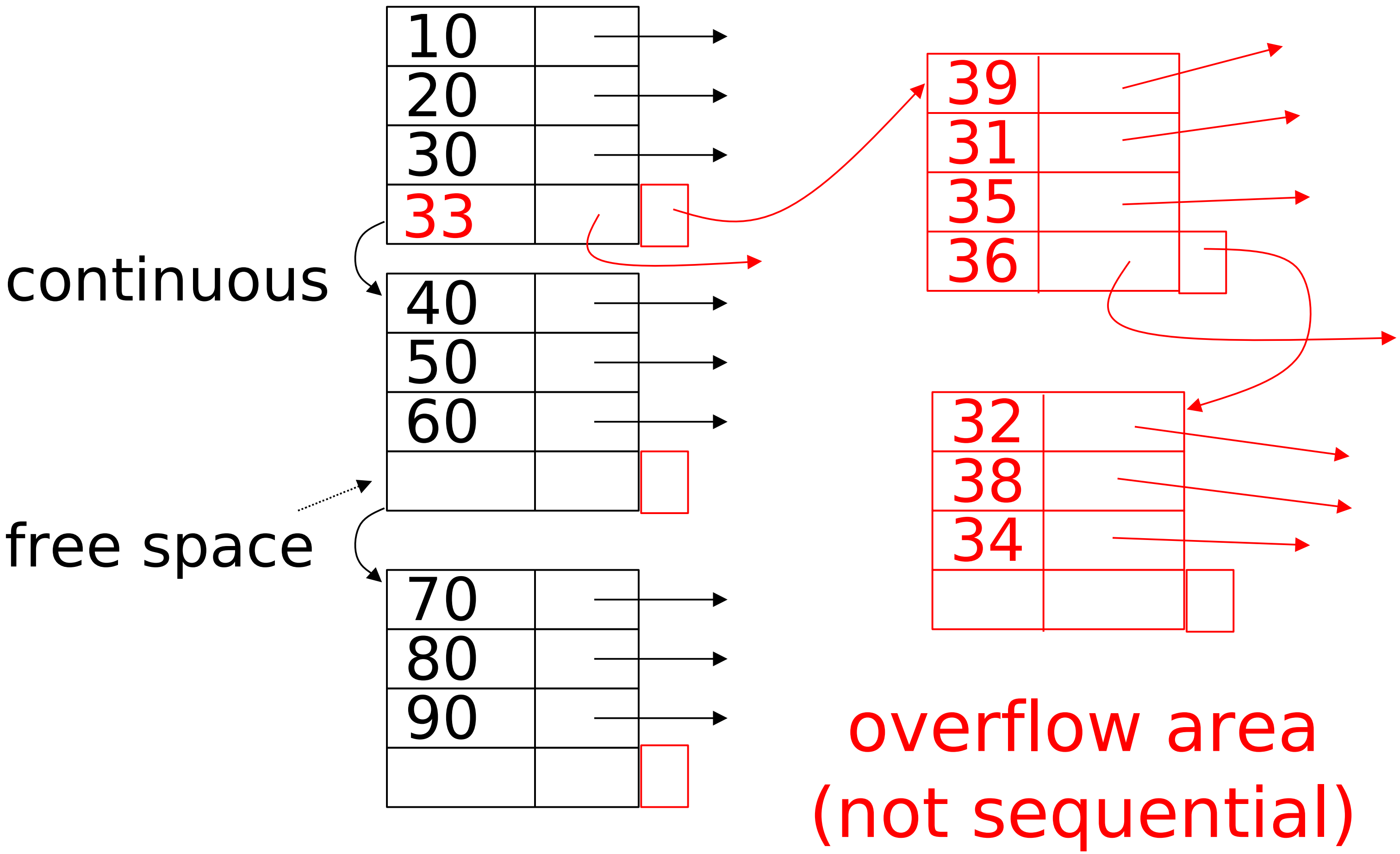
- Simple
- Index is sequential file
  good for scans

Disadvantage:

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

Example

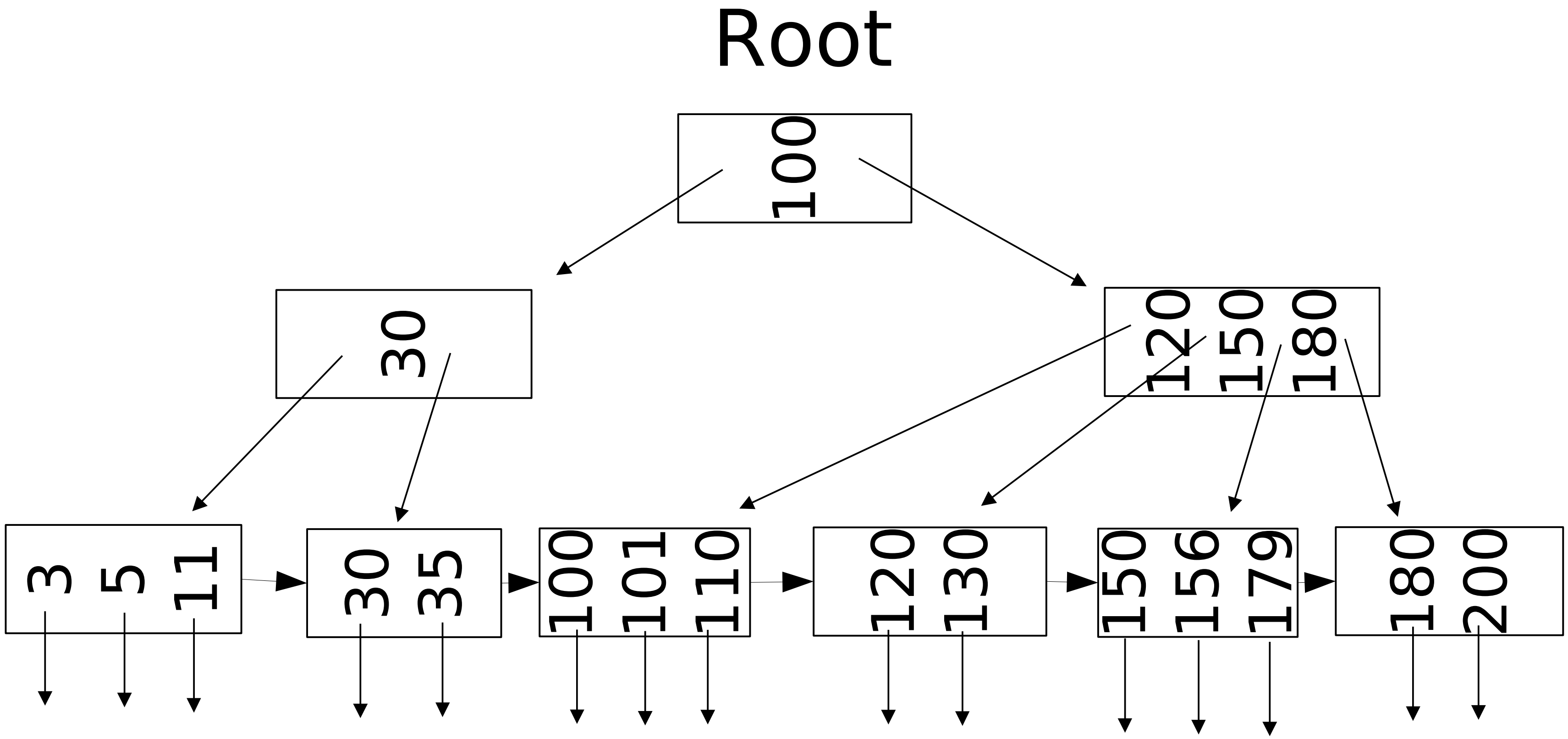
Index (sequential)



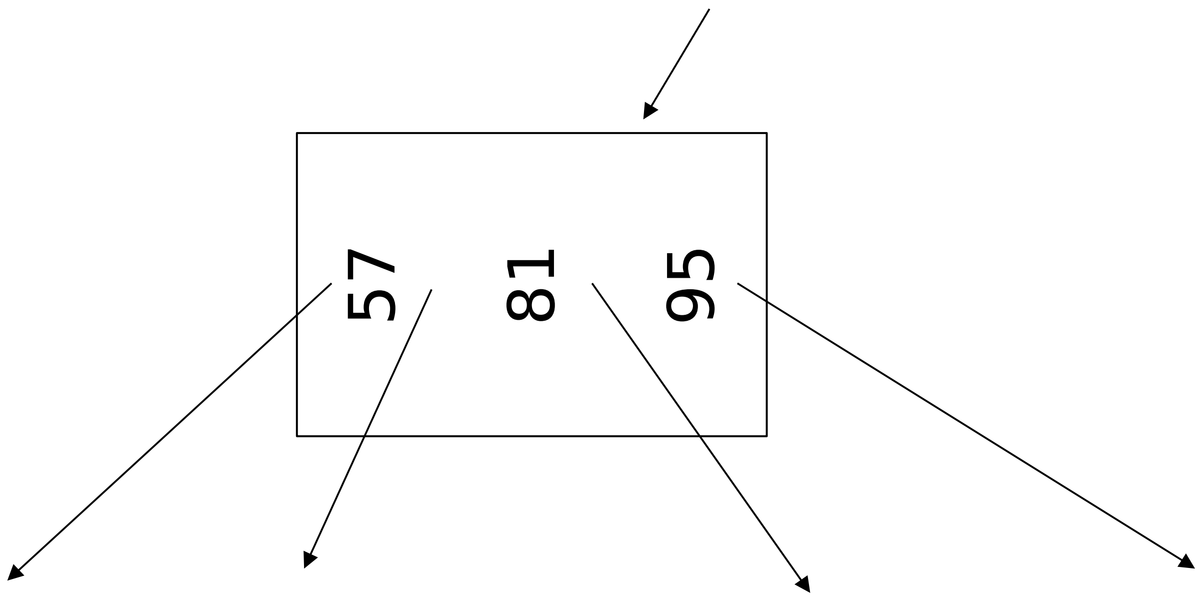
- NEXT: Another type of index
  - Give up on sequentiality of index
  - Try to get “balance”

B+Tree Example

n=3



Sample non-leaf



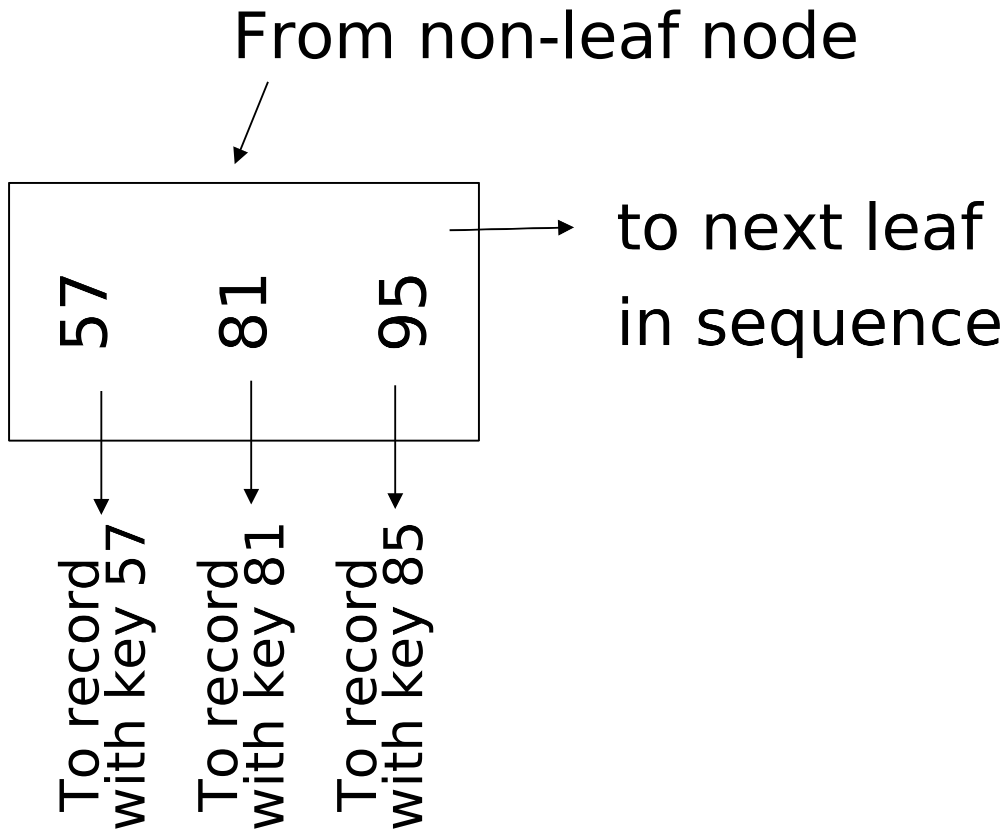
to keys < 57

to keys $57 \leq k < 81$

to keys $81 \leq k < 95$

to keys ≥ 95

Sample leaf node:



In textbook's notation

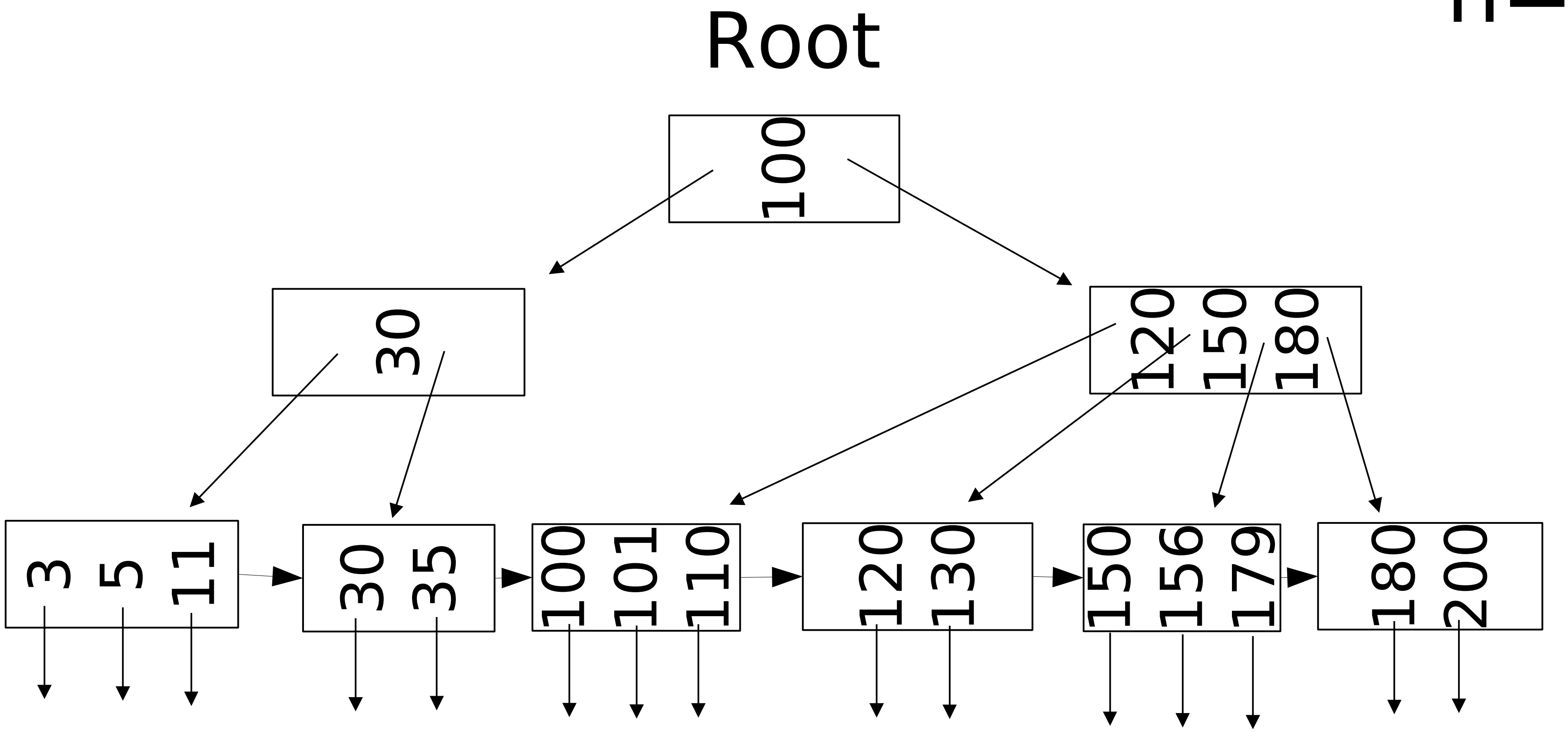
n=3

Leaf:

Non-leaf:

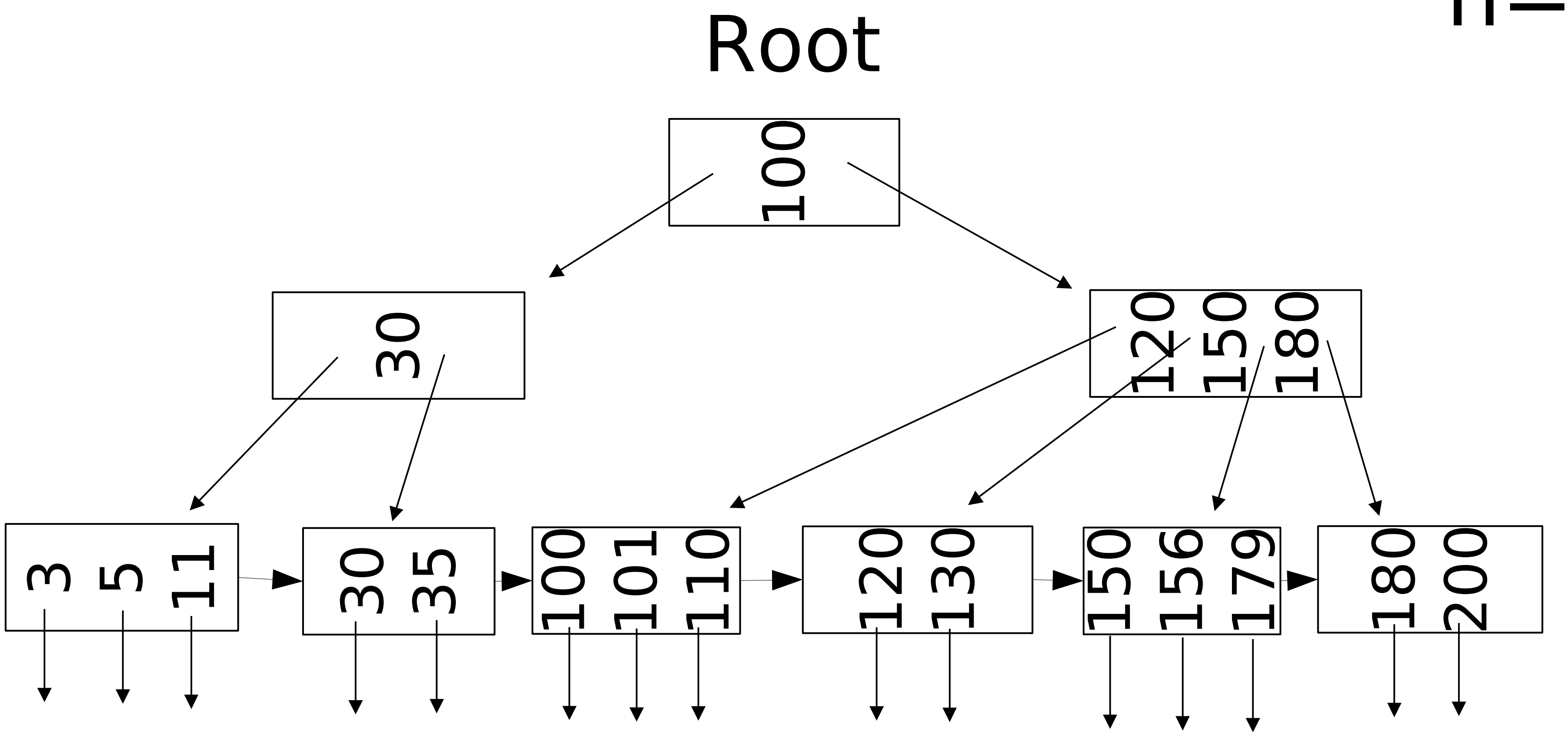
Lookup record(s) with key = 35

n=3



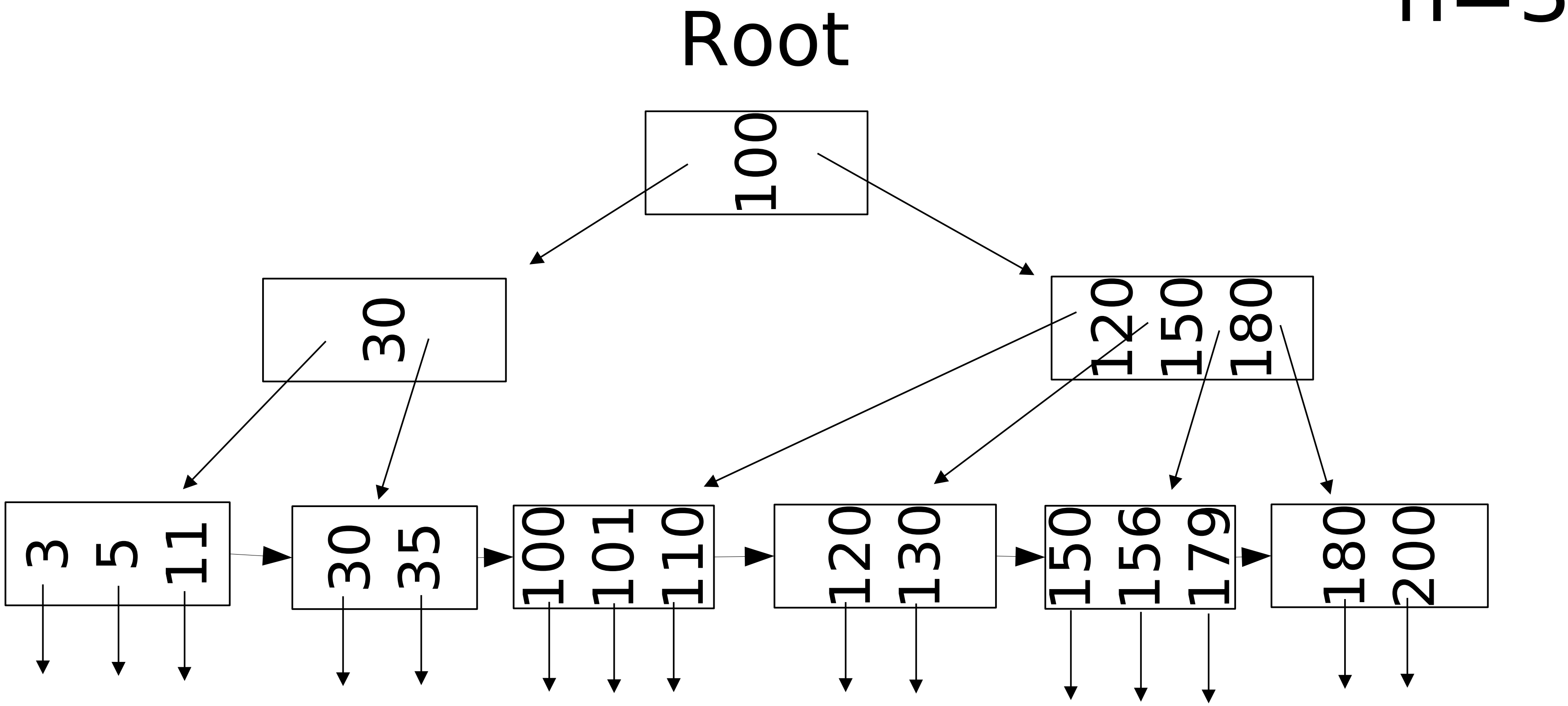
Lookup record(s) with key = 40

n=3



Range query: lookup record(s) with 35 <= key <= 150

n=3



- The I/O cost of a lookup in a BTree is equal to longest path of the root to a leaf
- Hence, the goal is to keep this longest path as short as possible
- In particular: we want all leafs to be at the same depth in the tree (and hence want a balanced tree)

Size of nodes: { n+1 pointers, n keys } (fixed)

- n+1 pointers
- n keys

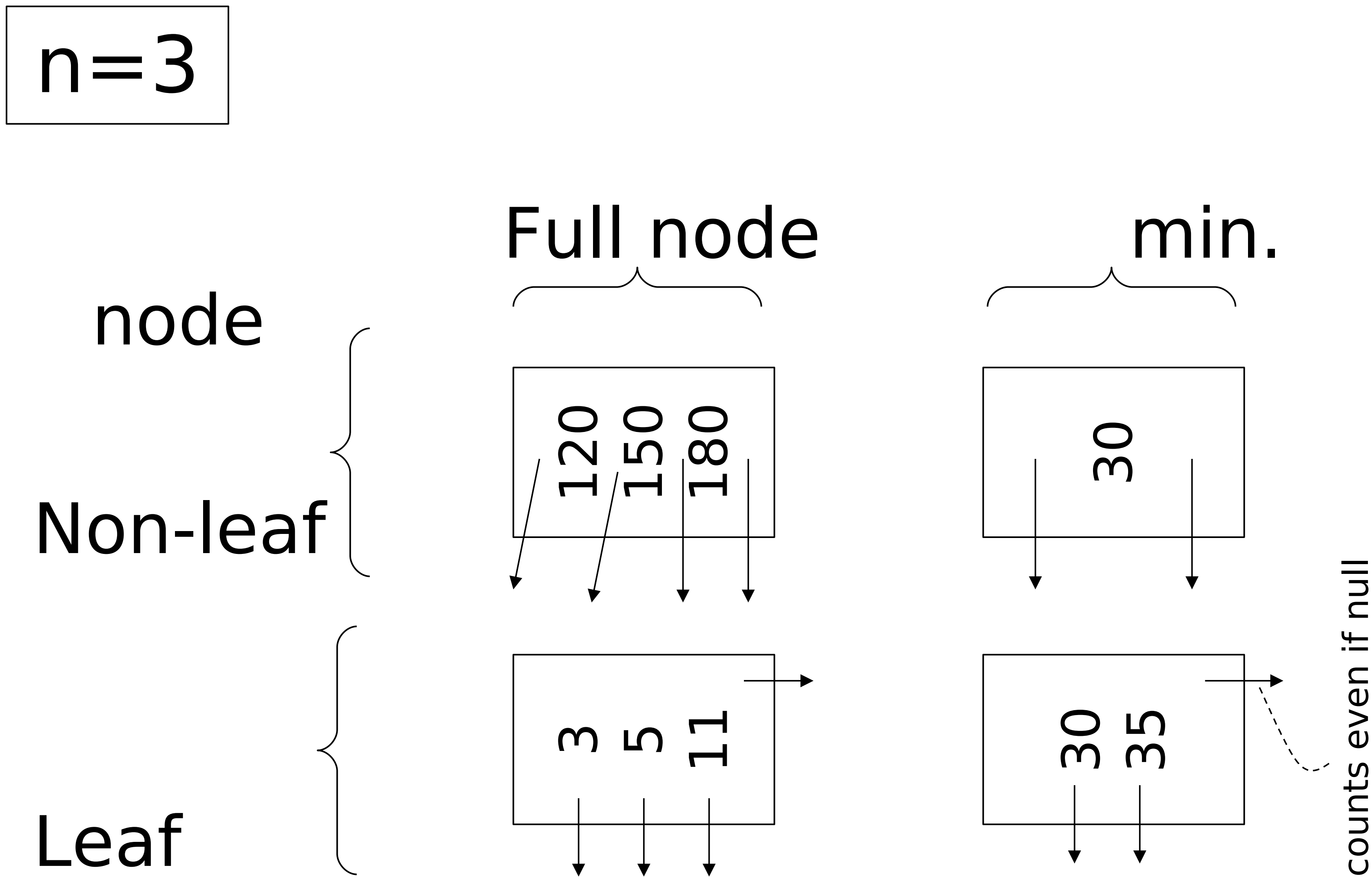
(fixed)

Don’t want nodes to be too empty

- Use at least

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

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


n=3
Full node
min. node
Non-leaf
Leaf
120
150
180
30
3
5
11
30
35
counts even if null

B+tree rules tree of order *d*

(1) All leaves at same lowest level (balanced tree)

(2) Pointers in leaves point to records except for "sequence pointer"

(3) 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	2	1

Insert into B+tree

(a) simple case

- space available in leaf

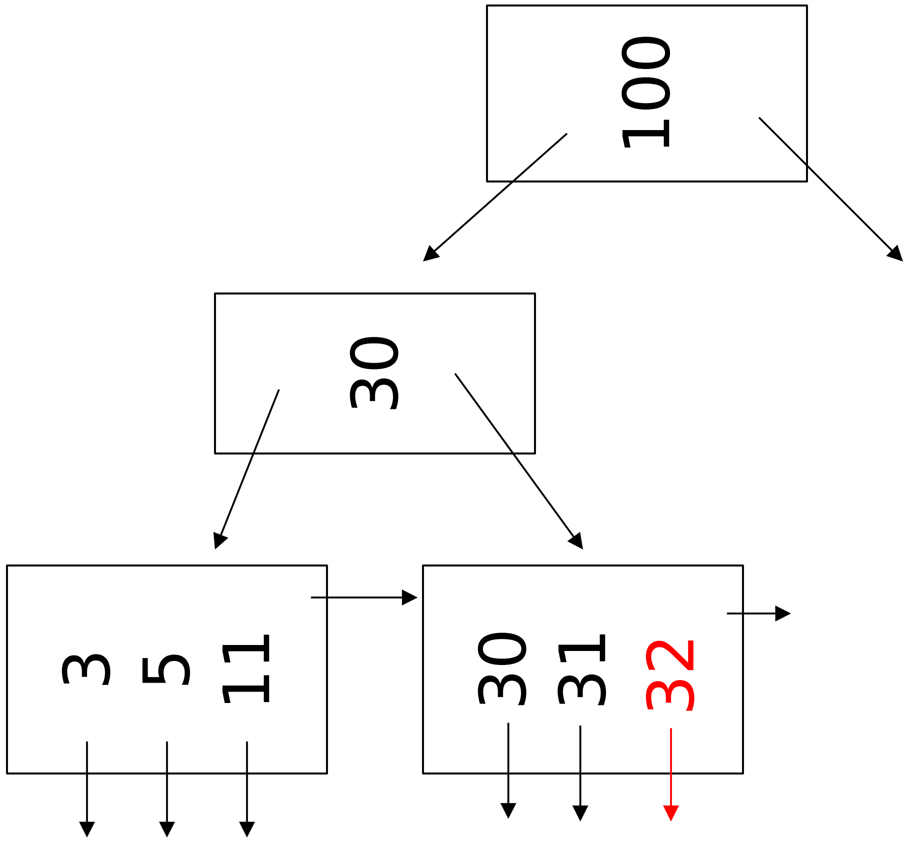
(b) leaf overflow

(c) non-leaf overflow

(d) new root

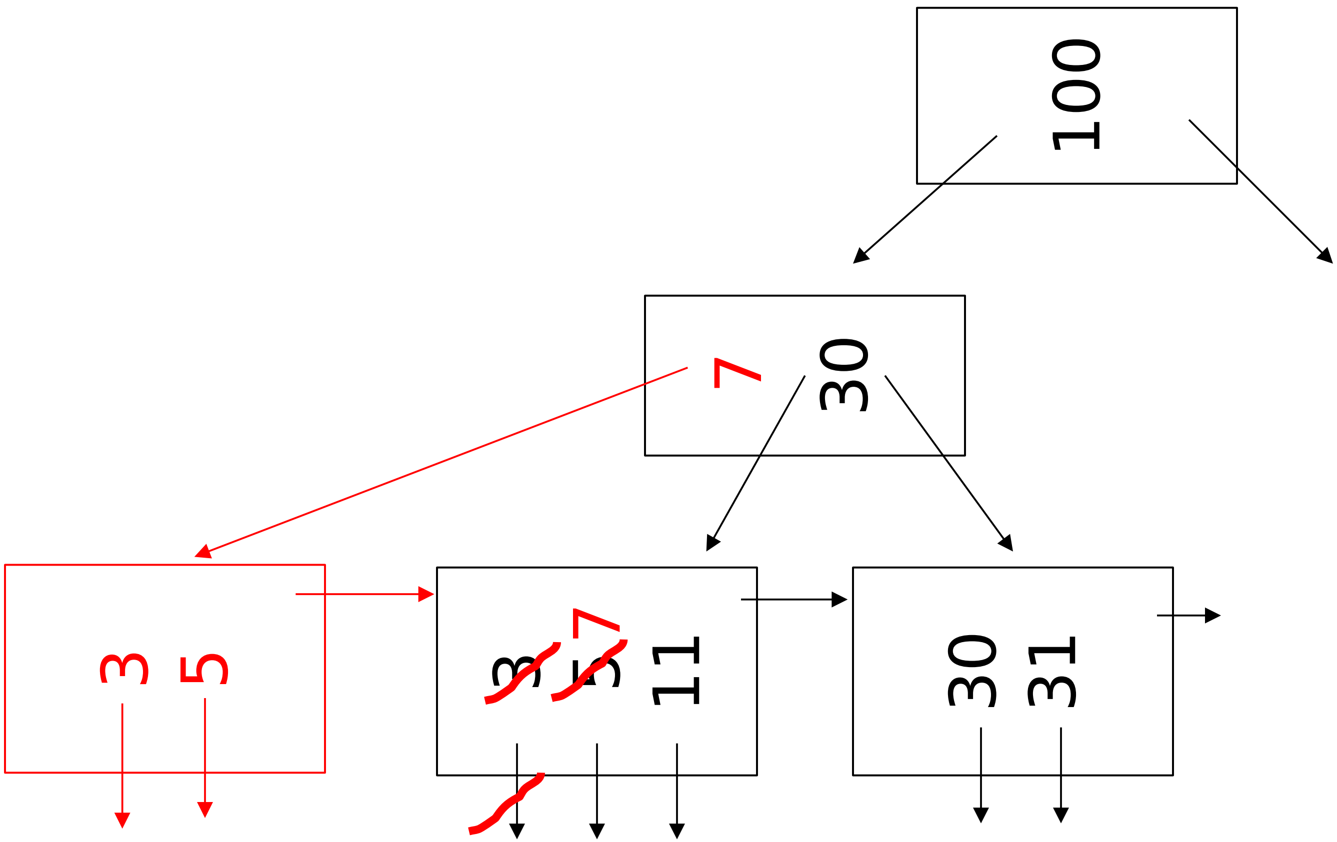
(a) Insert key = 32

n=3



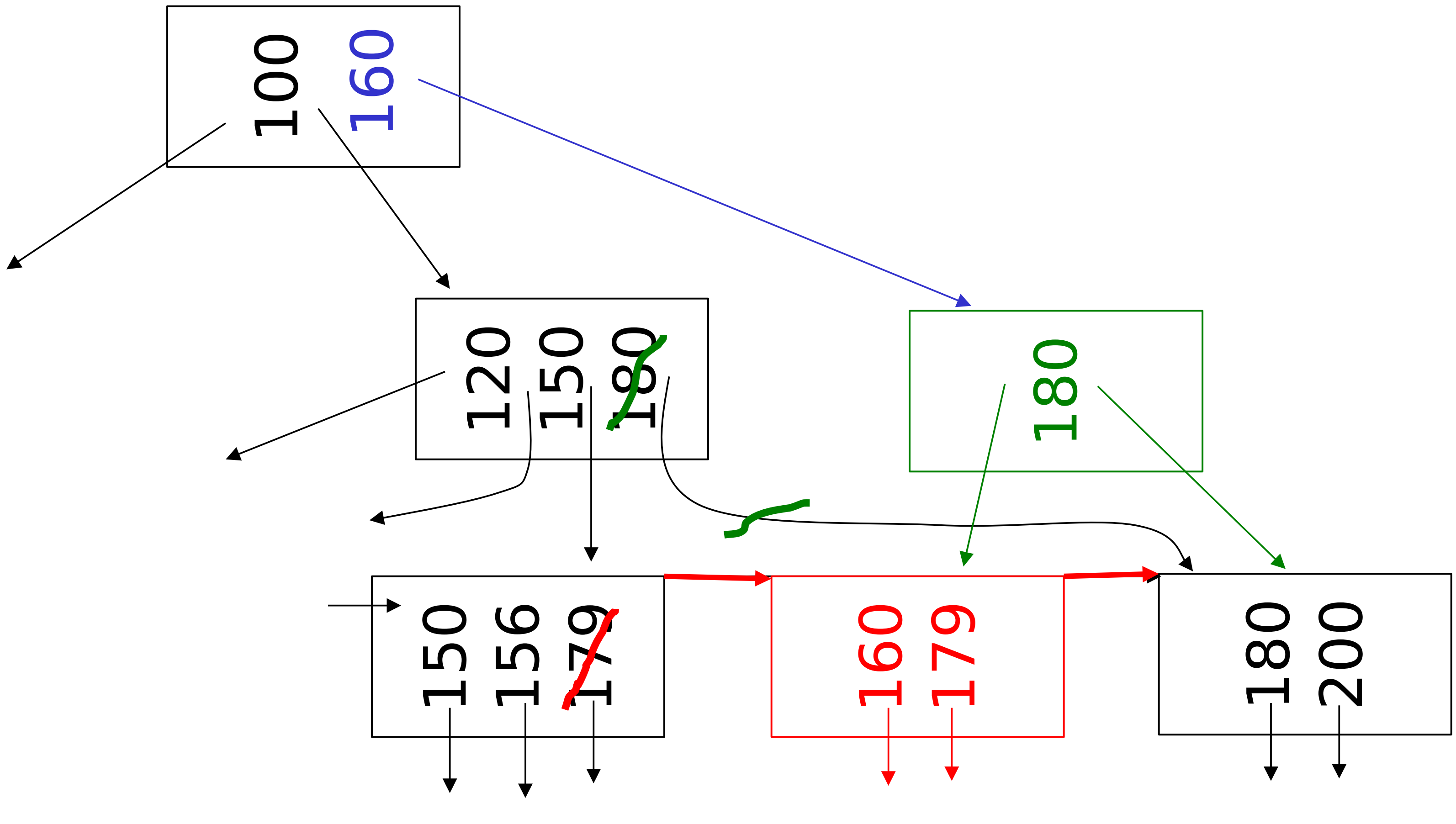
(a) Insert key = 7

n=3



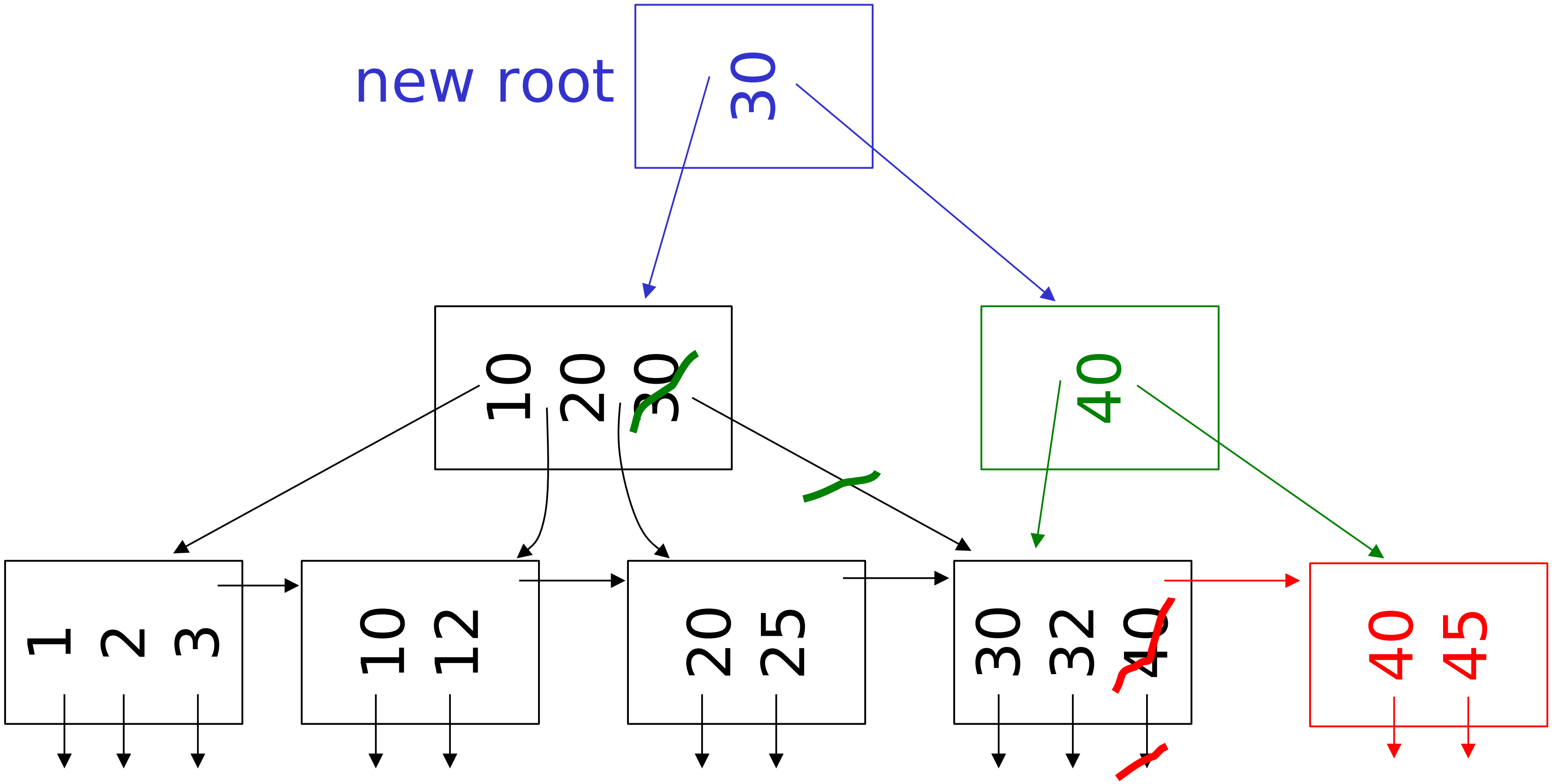
(c) Insert key = 160

n=3



(d) New root, insert 45

n=3



Deletion from B+tree

(a) Simple case - no example

(b) Coalesce with neighbor (sibling)

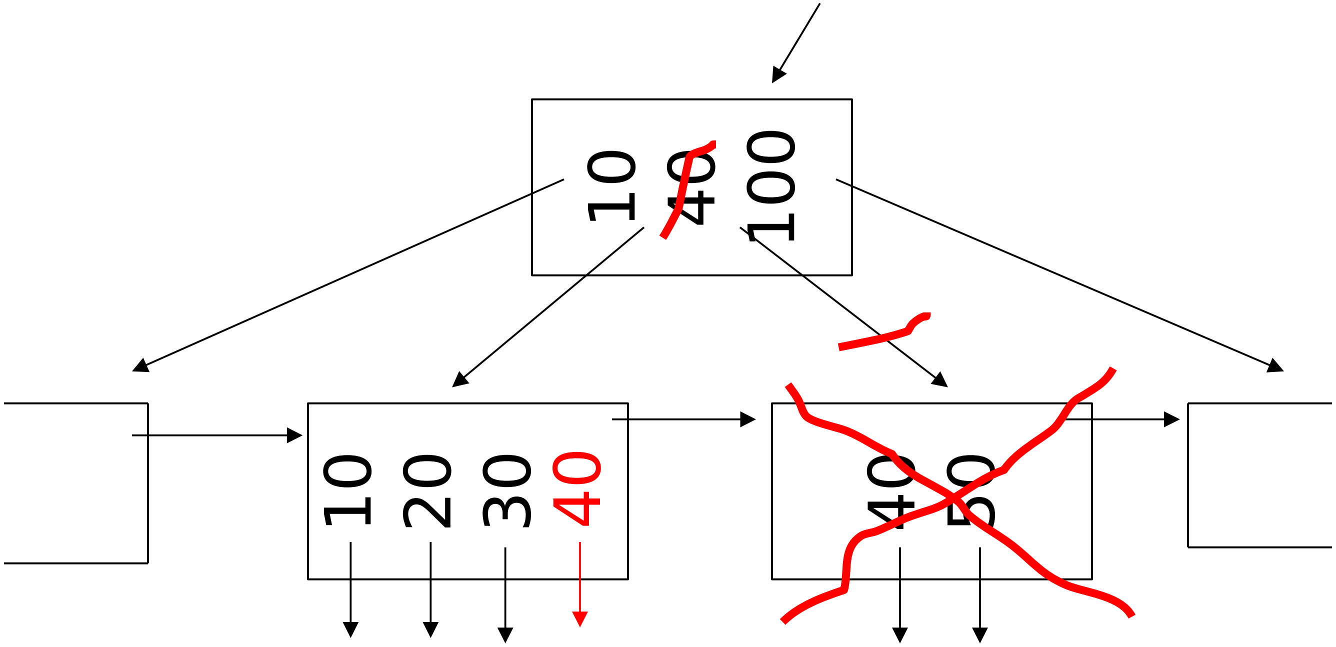
(c) Re-distribute keys

(d) Cases (b) or (c) at non-leaf

(b) Coalesce with sibling

– Delete 50

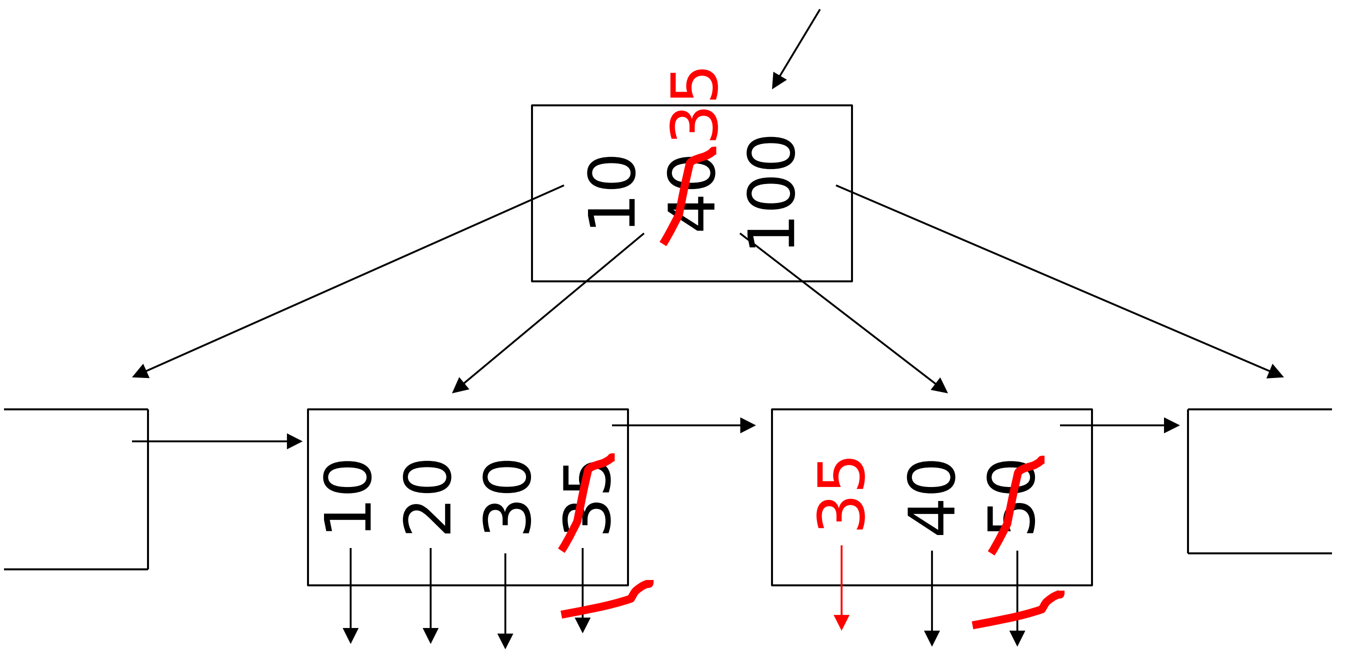
n=4



(c) Redistribute keys

– Delete 50

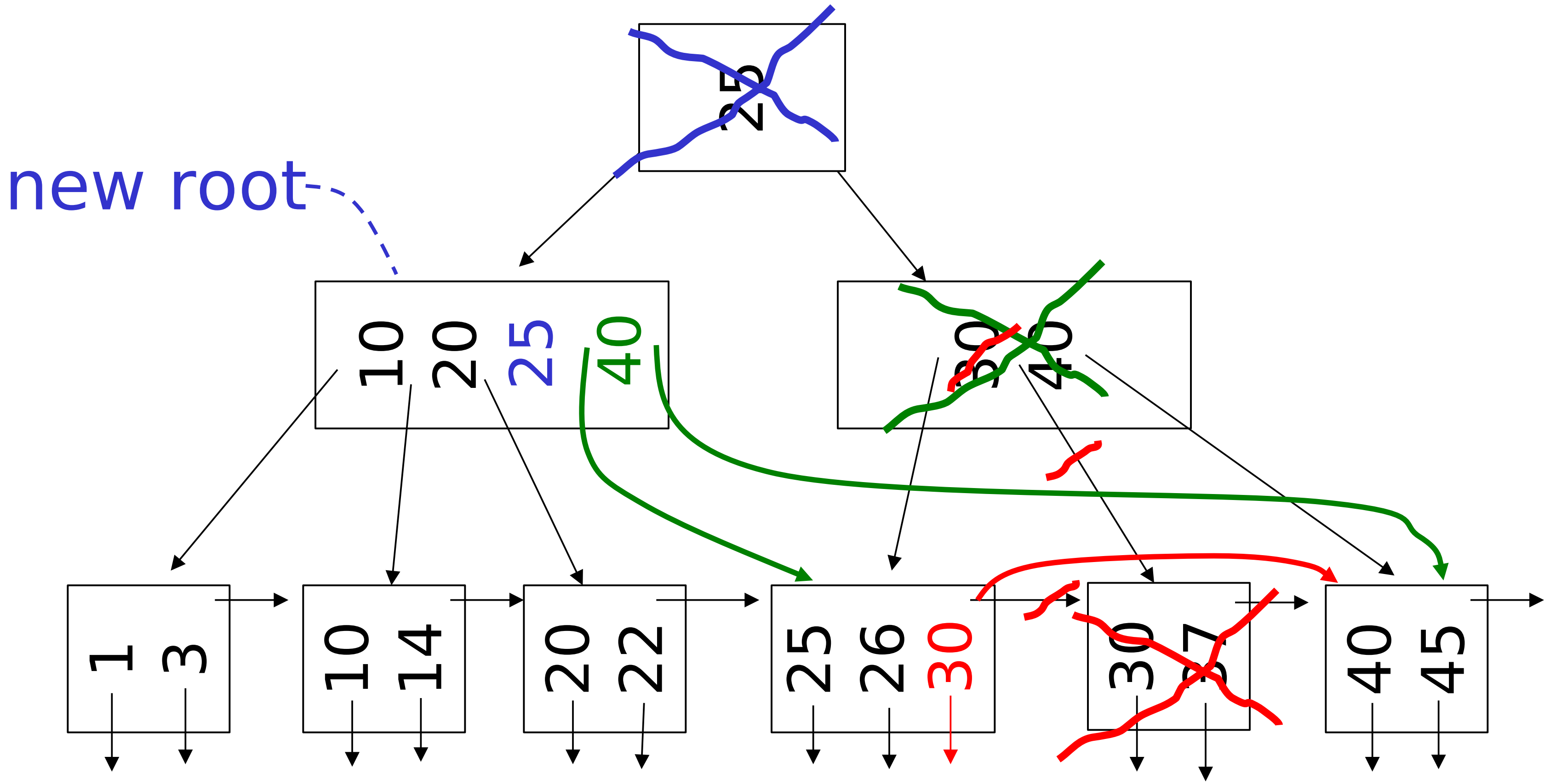
n=4



(d) Non-leaf coalese

– Delete 37

n=4



Outline/summary

- Conventional Indexes
  - Sparse vs. dense
  - Primary vs. secondary
- B trees
  - B+trees vs. indexed sequential
- Hashing schemes --> Next