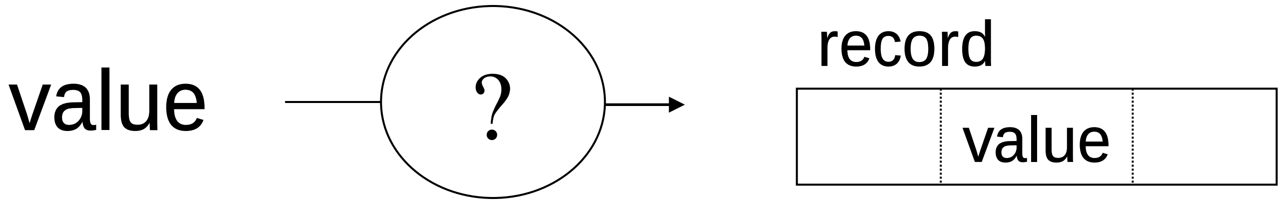
CSCD43: Database Systems Technology

Lecture 6

Wael Aboulsaadat

Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

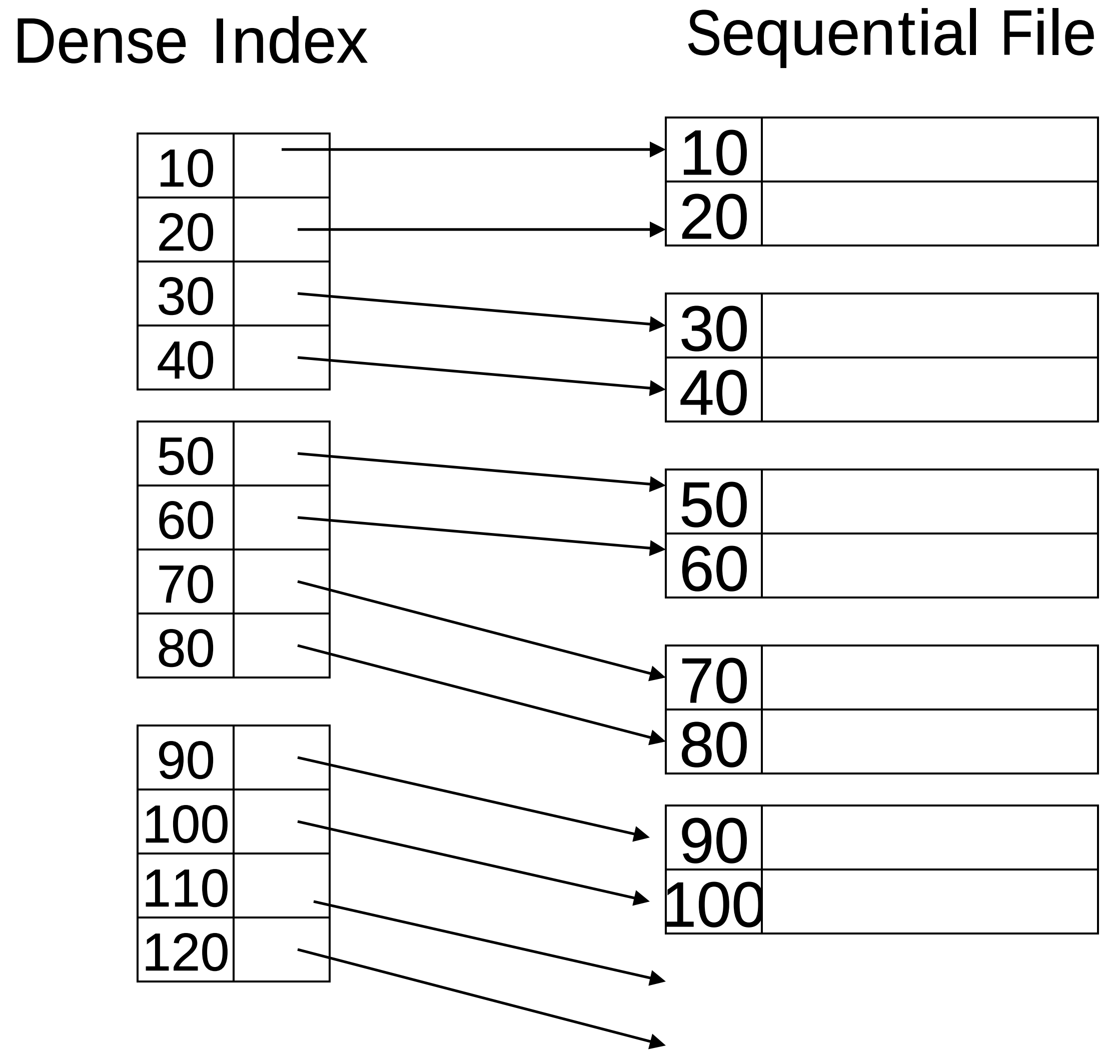

value
?
record
value

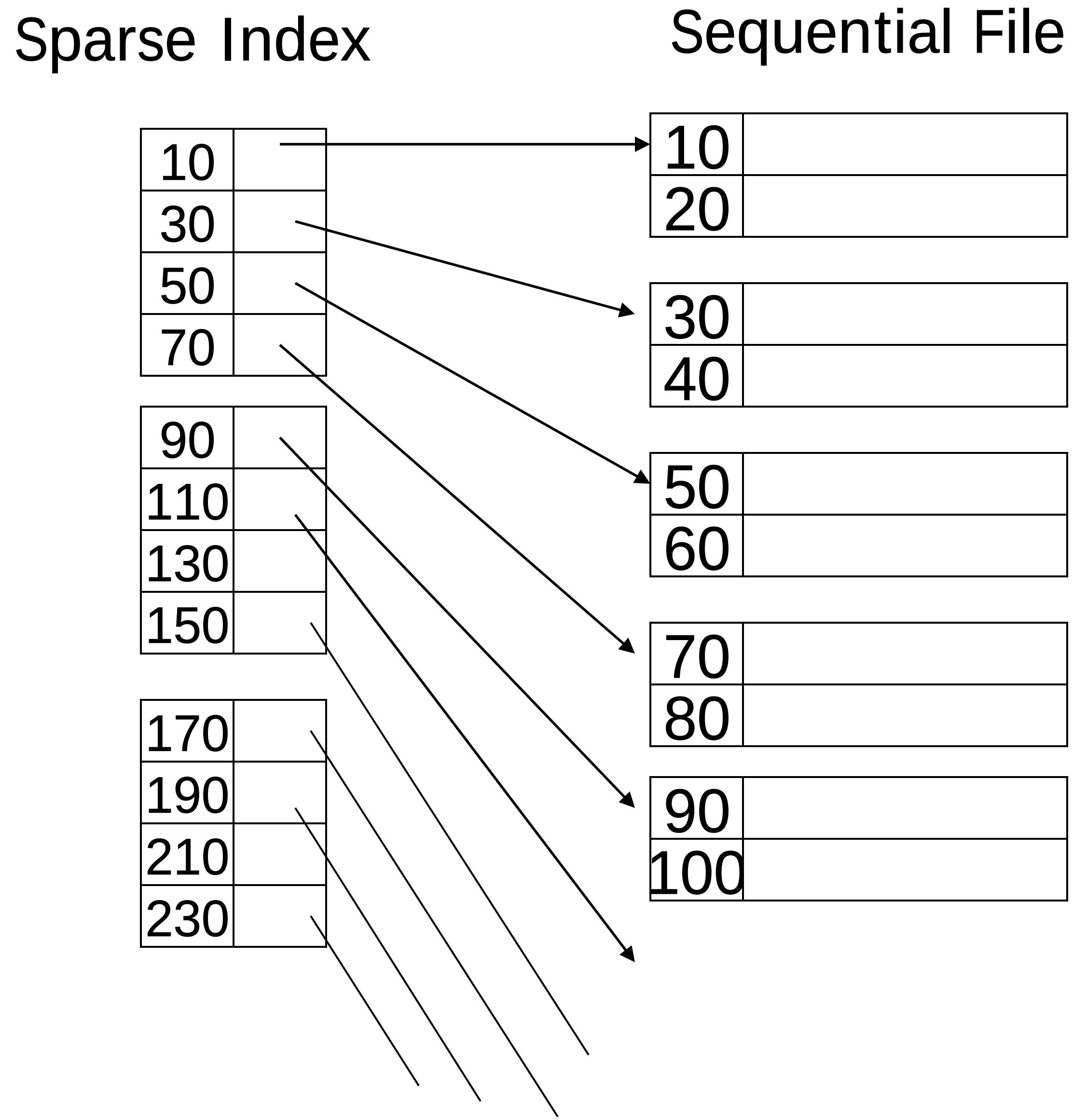
Topics

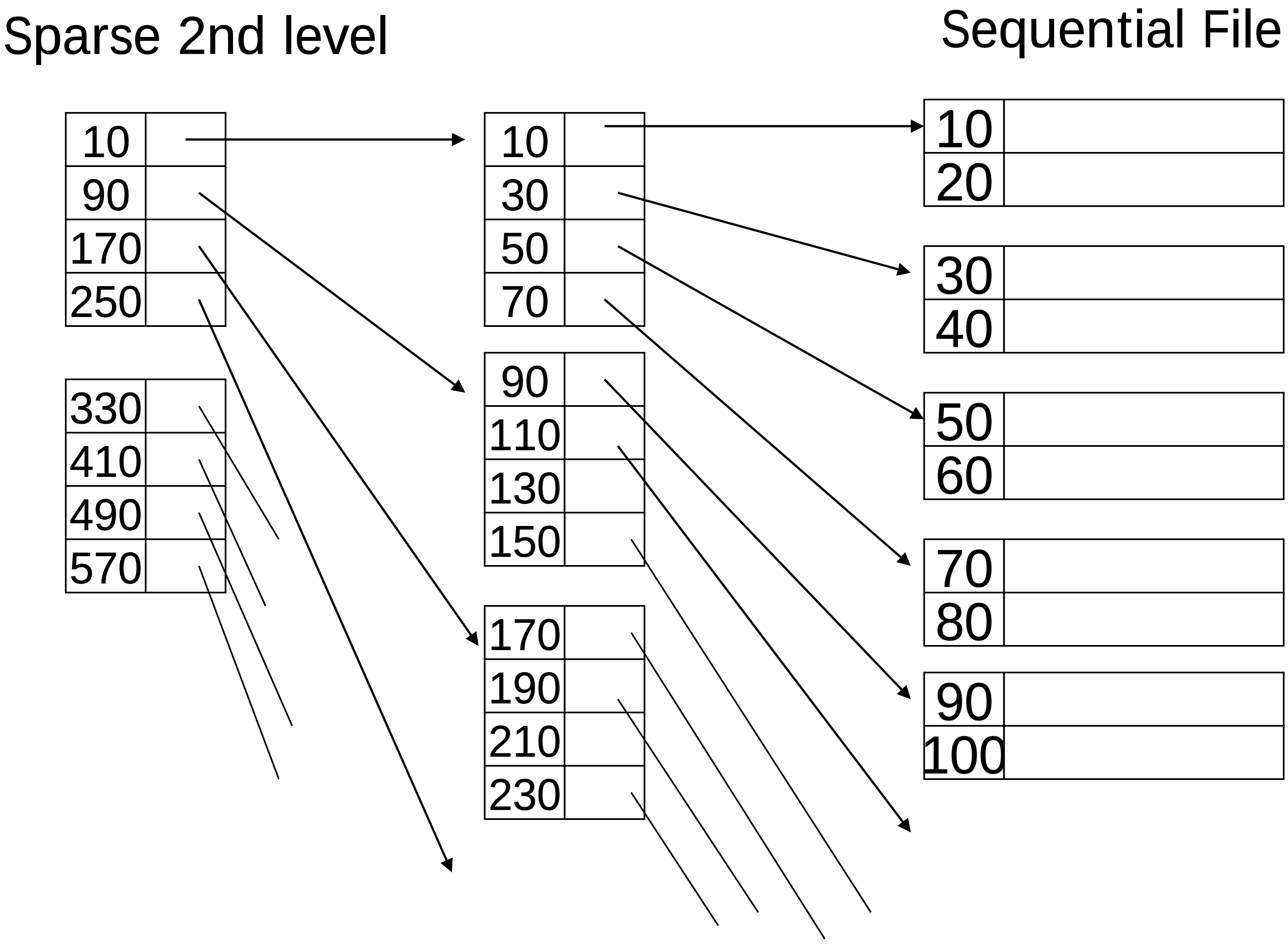
- Conventional Indexes
- B-trees
- Hashing Schemes
- Bitmap Indexes

Sequential File

10	
20	
30	
40	
50	
60	
70	
80	
90	
100	


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

- Comment:

  {FILE,INDEX} may be contiguous

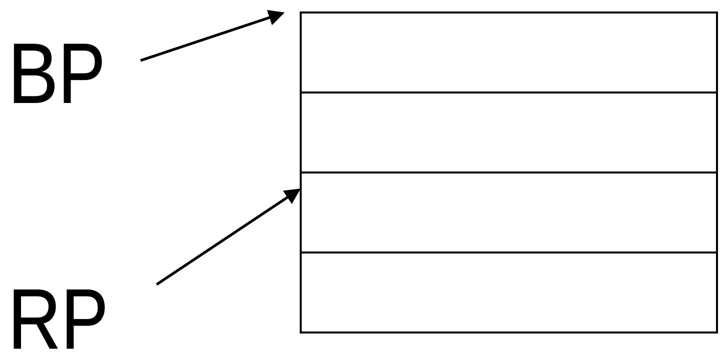
  or not (blocks chained)

Question:

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

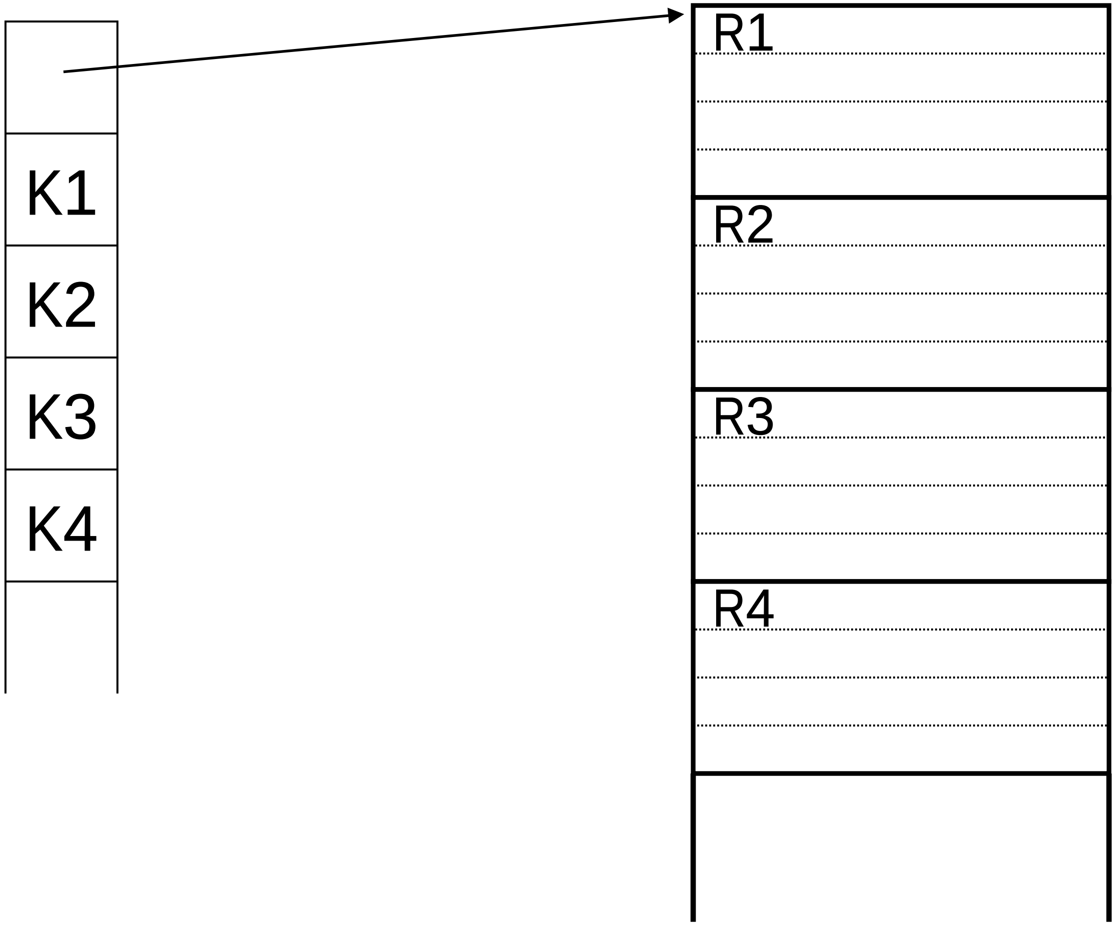
Notes on pointers:

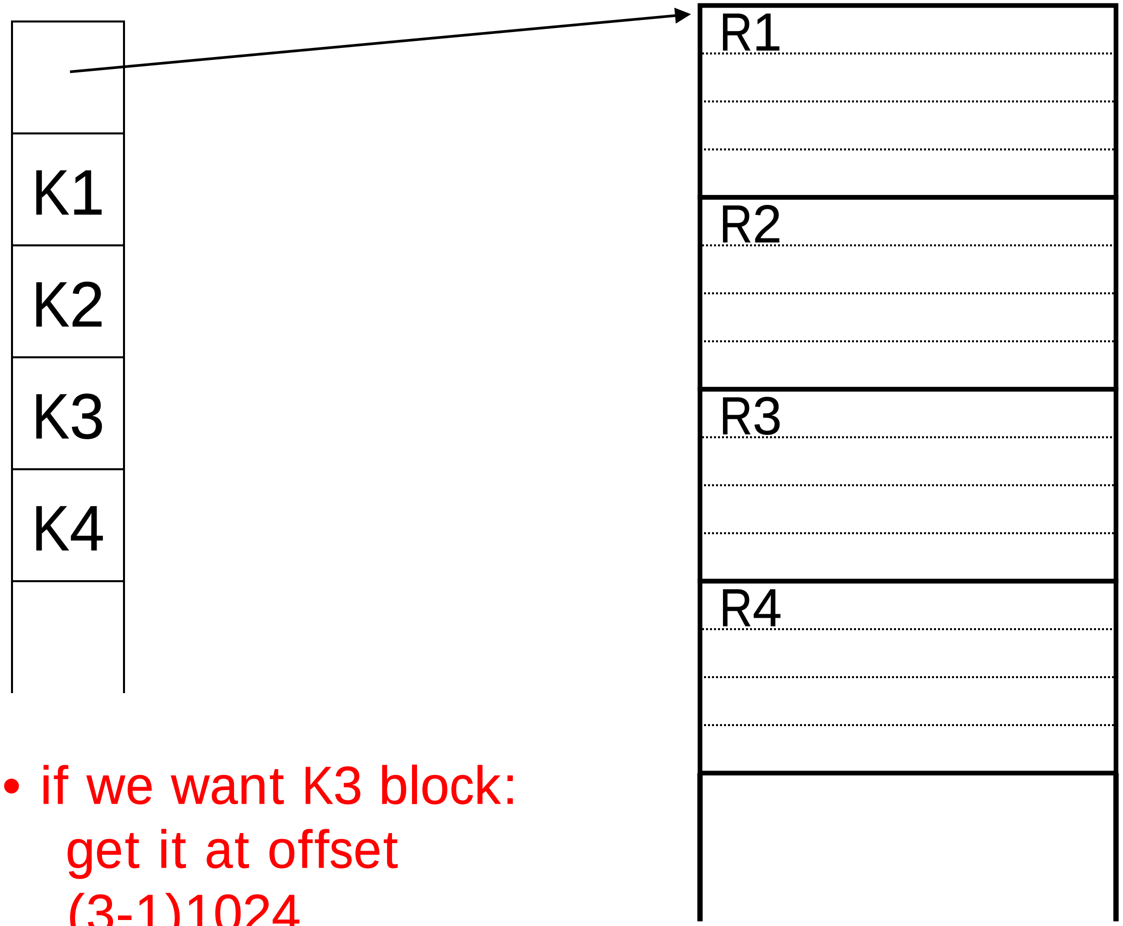
(1) Block pointer (sparse index) can be smaller than record pointer



Notes on pointers:

(2) If file is contiguous, then we can omit pointers (i.e., compute them)


K1
K2
K3
K4
R1
R2
R3
R4



say:
1024 B
per block

- if we want K3 block:
  get it at offset
  (3-1)1024
  = 2048 bytes

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)

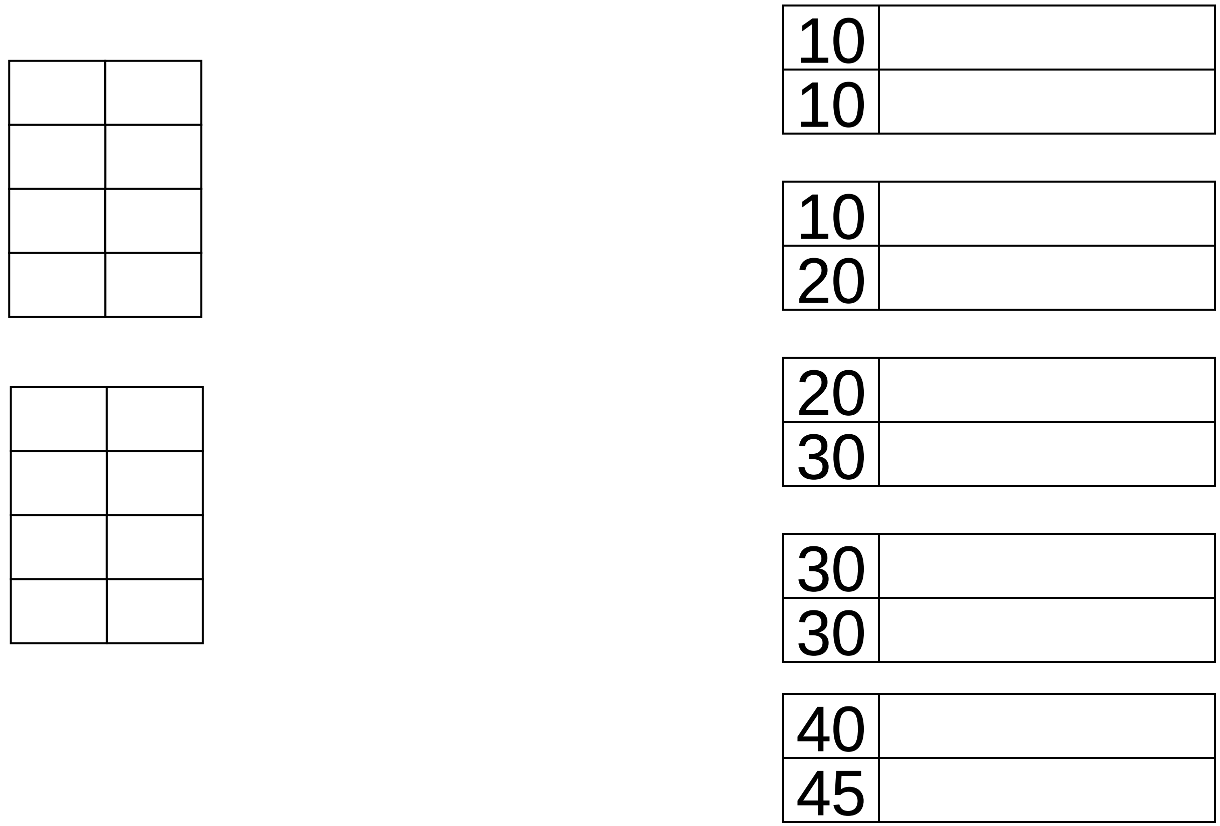
Terms

- Index sequential file
- Search key ($\neq$ primary key)
- Primary index (on Sequencing field)
- Secondary index
- Dense index (all Search Key values in)
- Sparse index
- Multi-level index

Next:

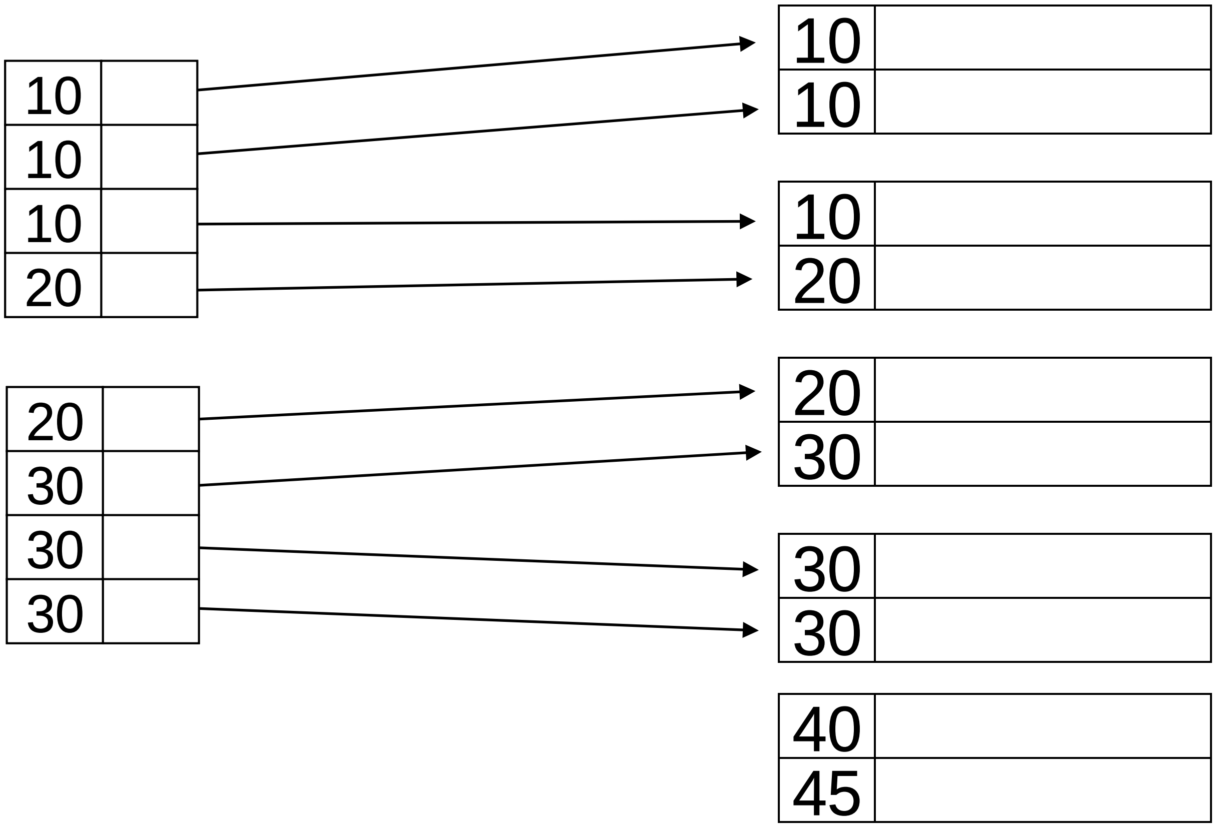
- Duplicate keys
- Deletion/Insertion
- Secondary indexes

Duplicate keys



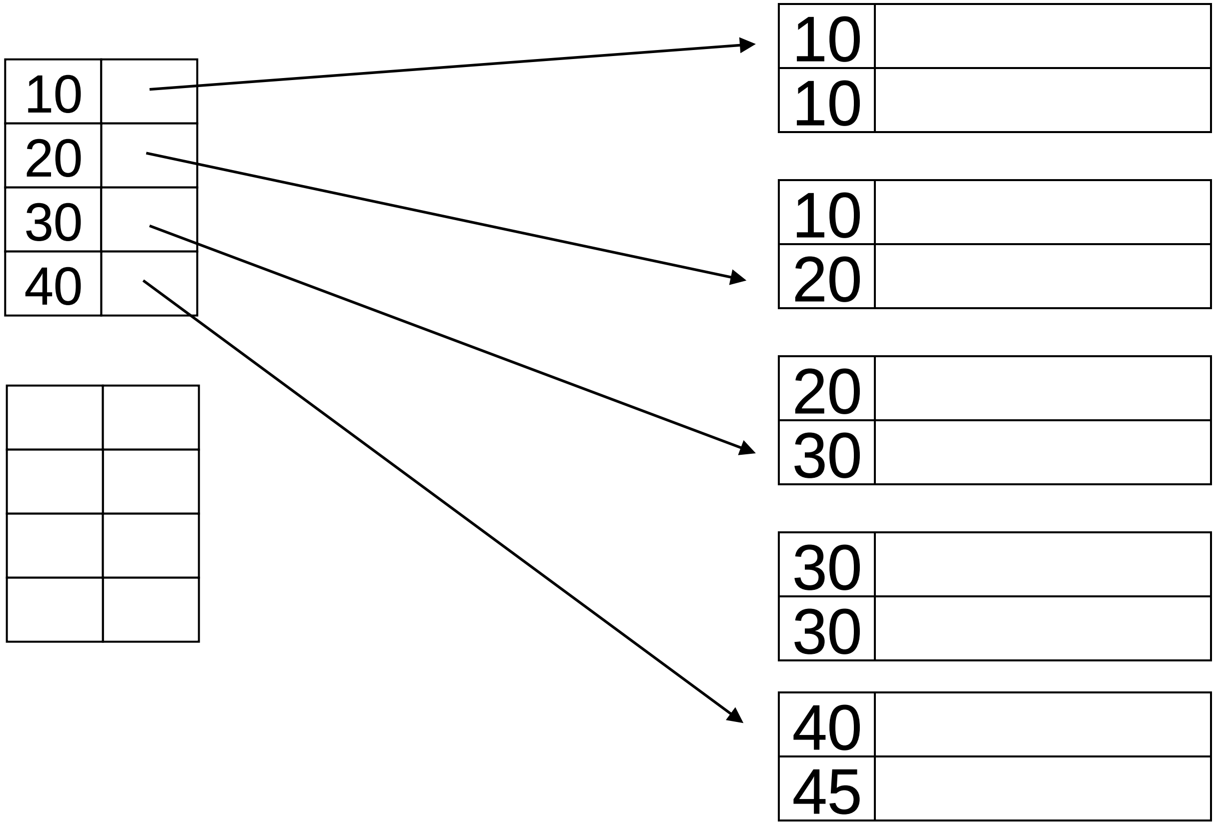
Duplicate keys

Dense index, one way to implement?



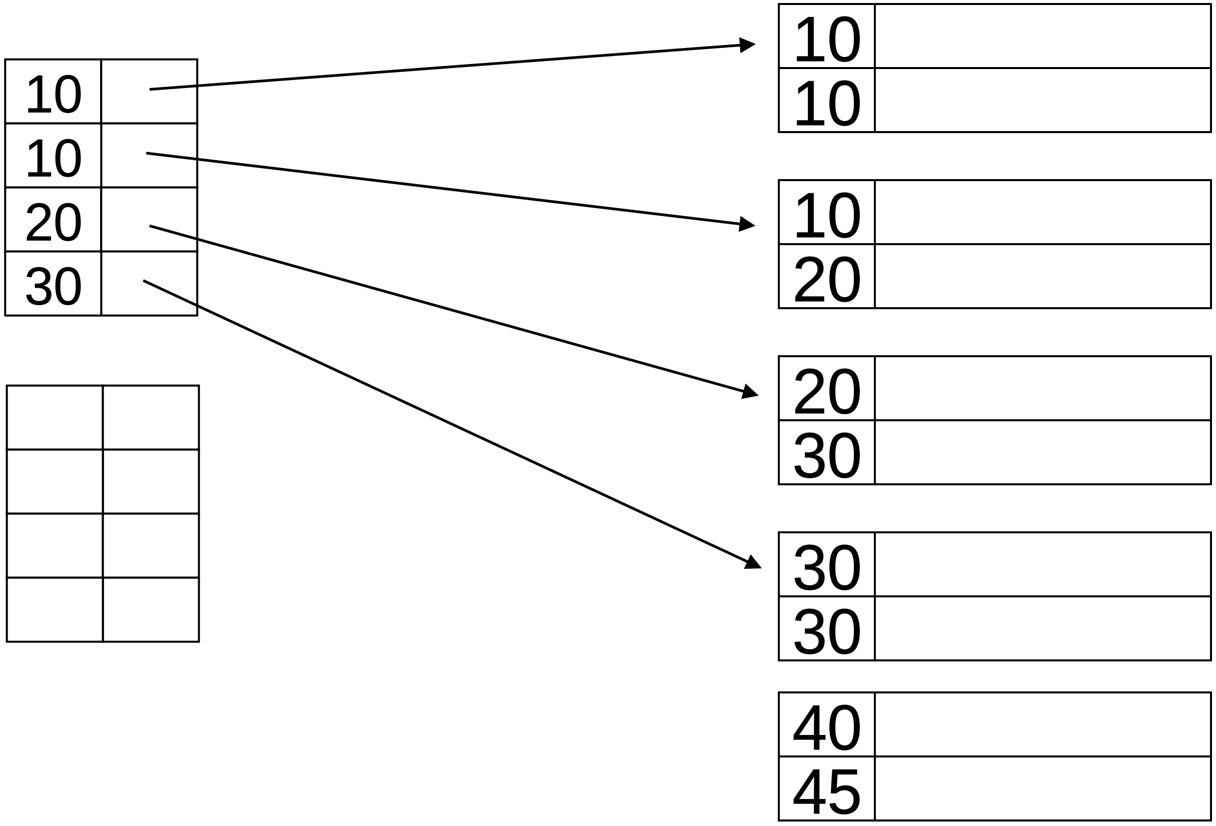
Duplicate keys

Dense index, better way?



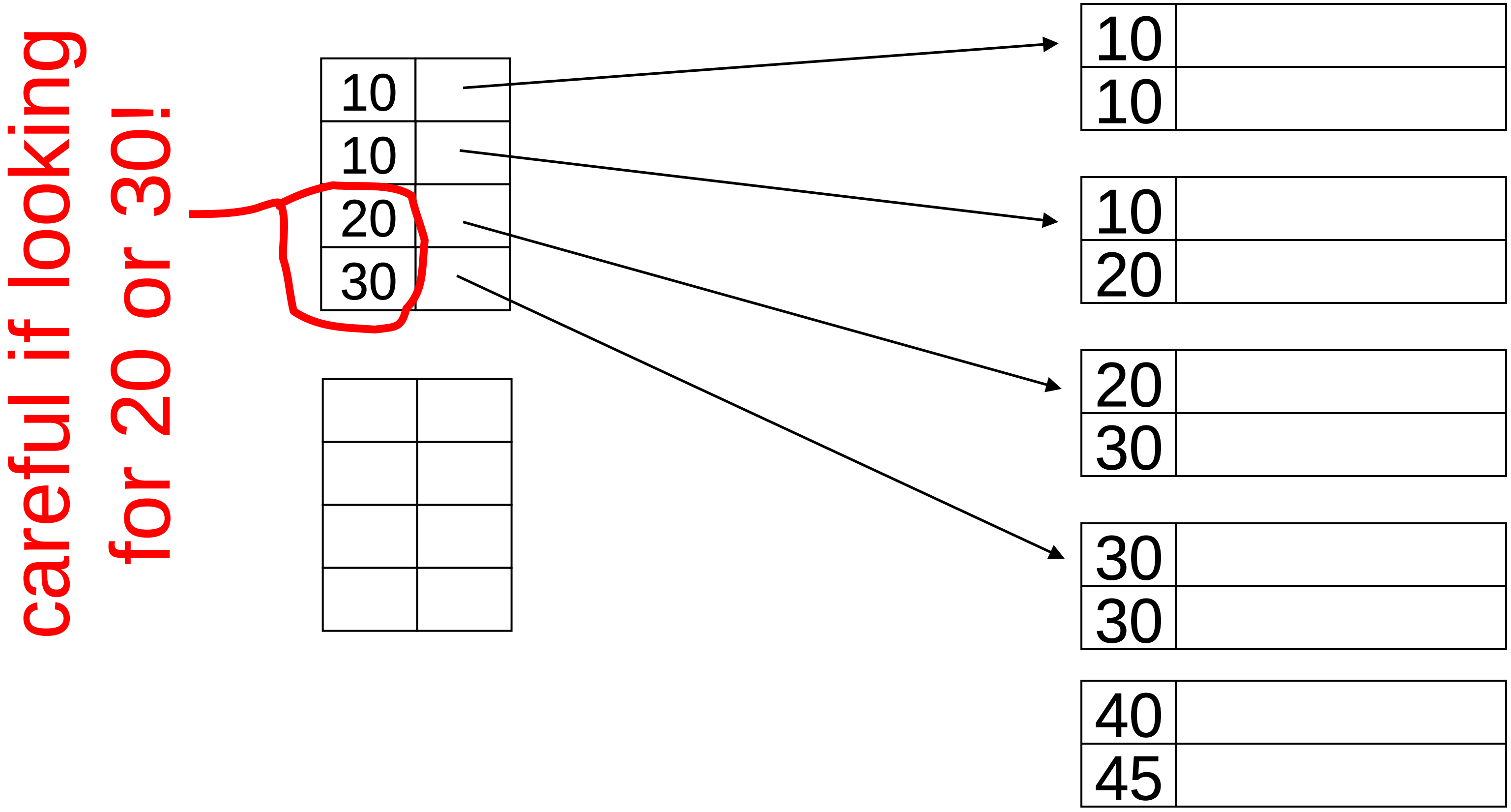
Duplicate keys

Sparse index, one way?



Duplicate keys

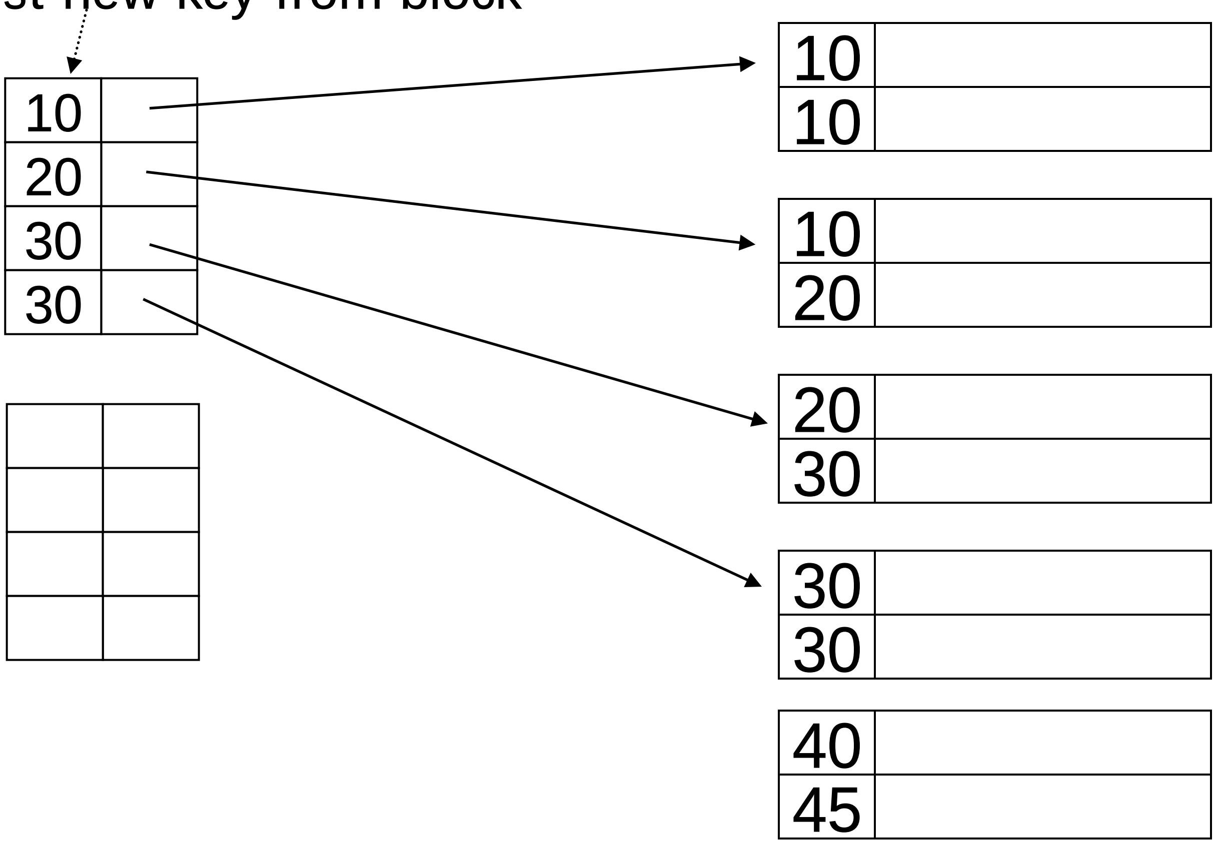
Sparse index, one way?



Duplicate keys

Sparse index, another way?

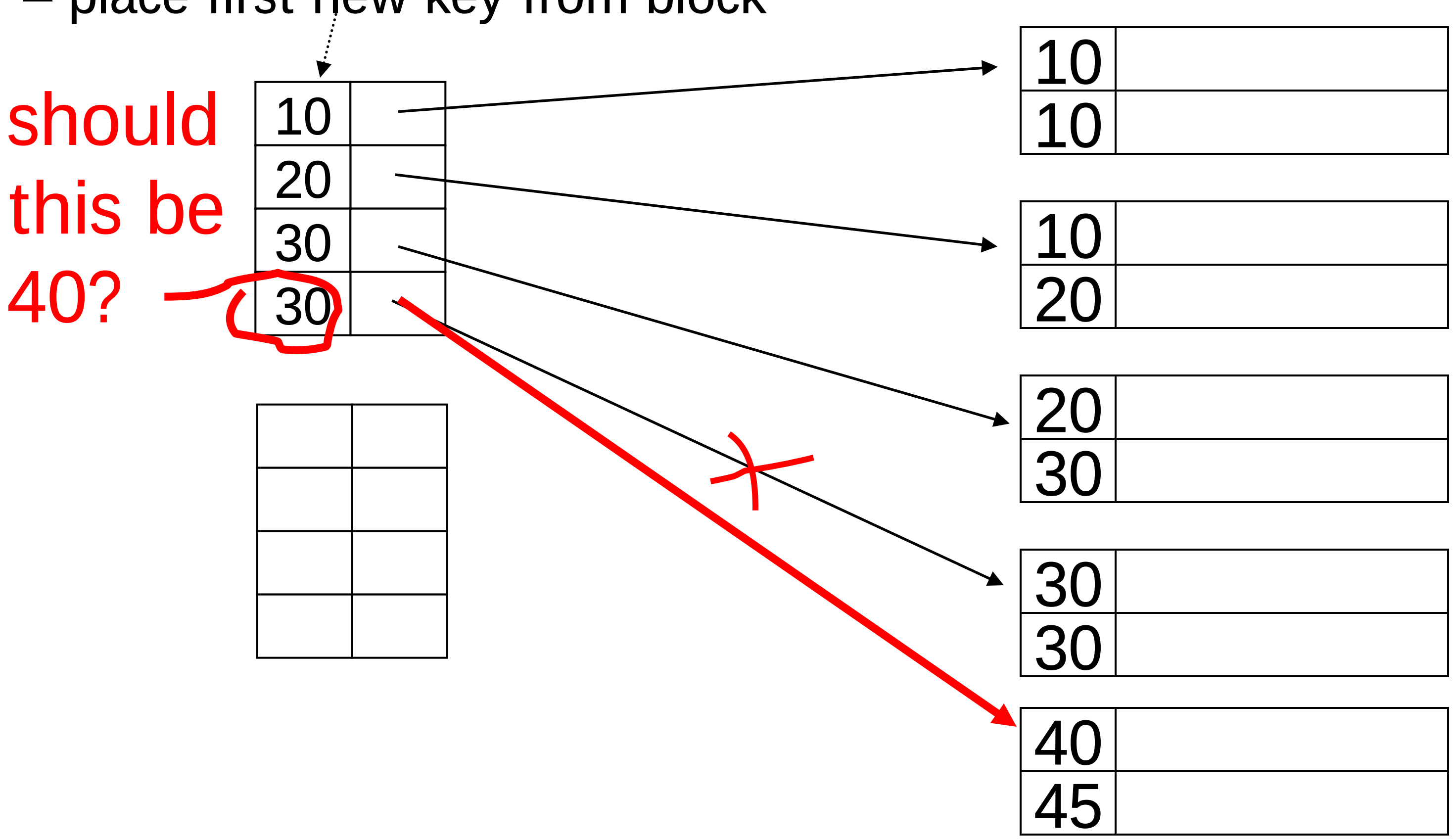
– place first new key from block



Duplicate keys

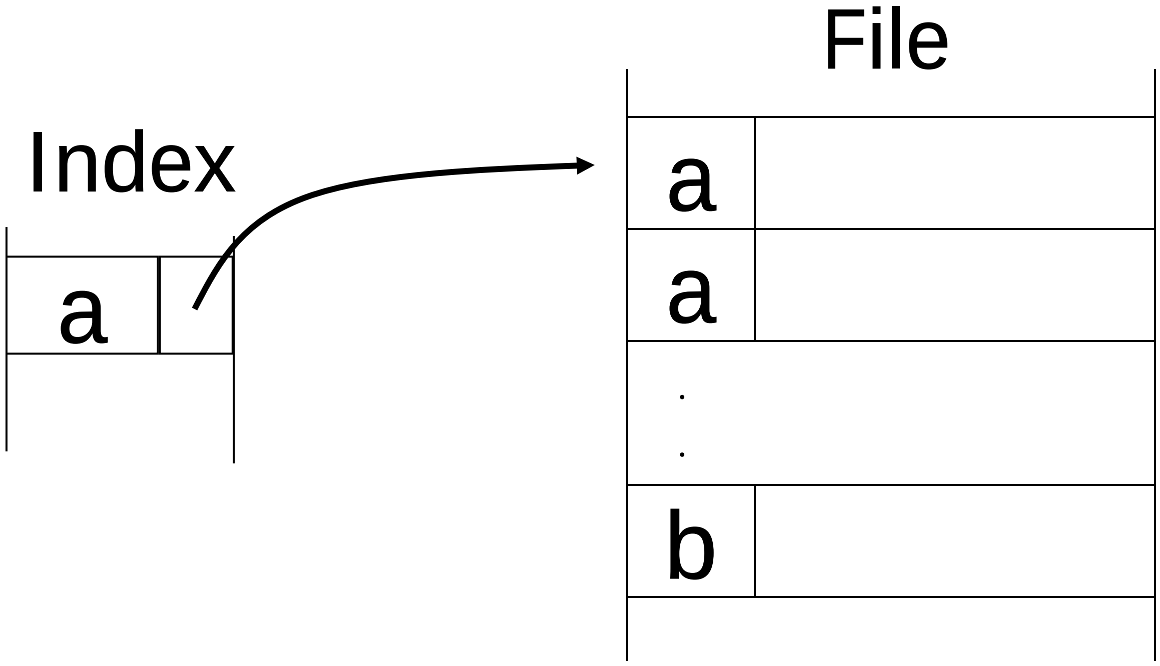
Sparse index, another way?

– place first new key from block

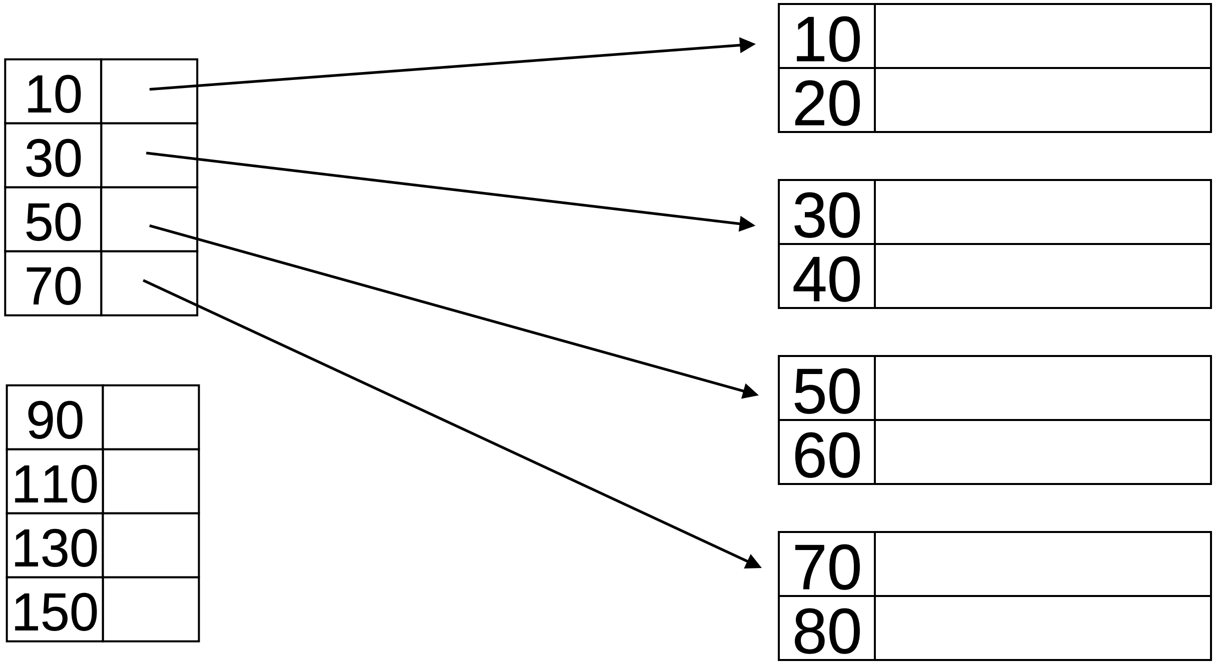


Summary — Duplicate values, primary index

- Index may point to first instance of each value only

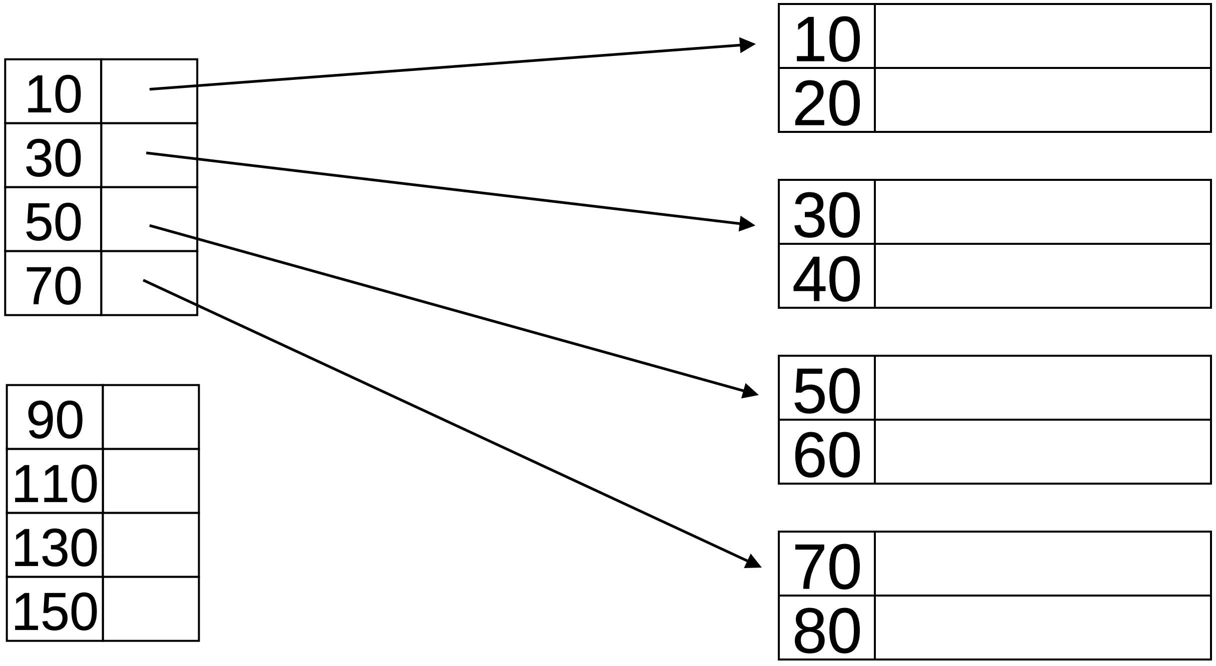


Deletion from sparse index



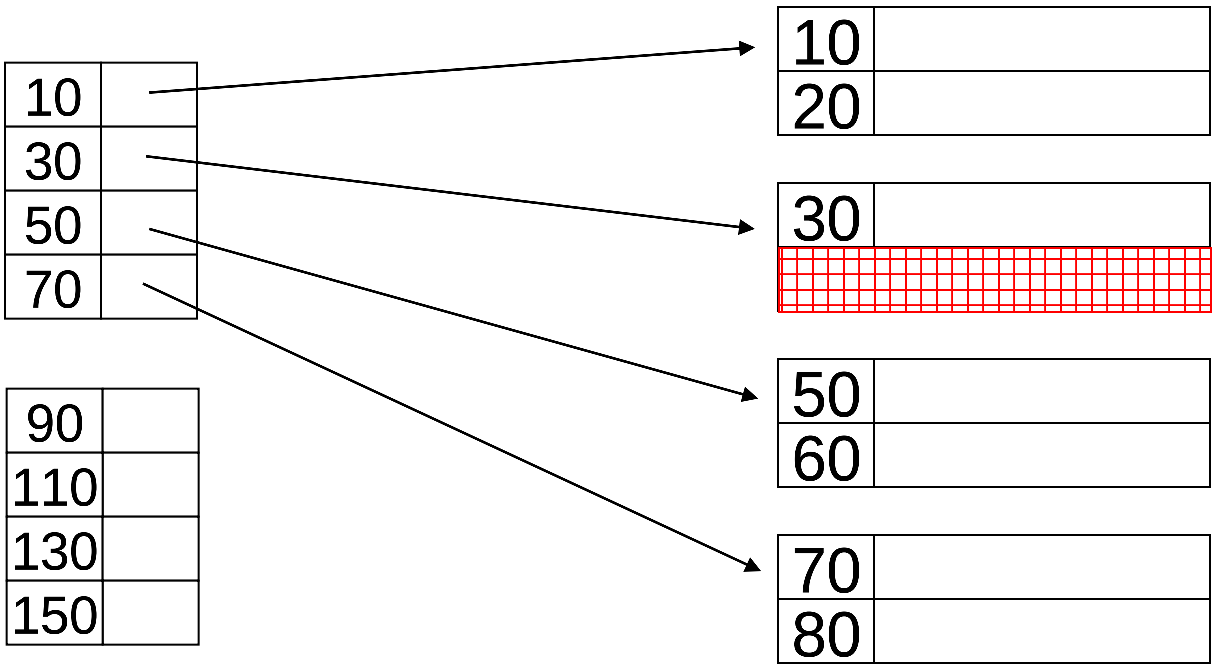
Deletion from sparse index

– delete record 40



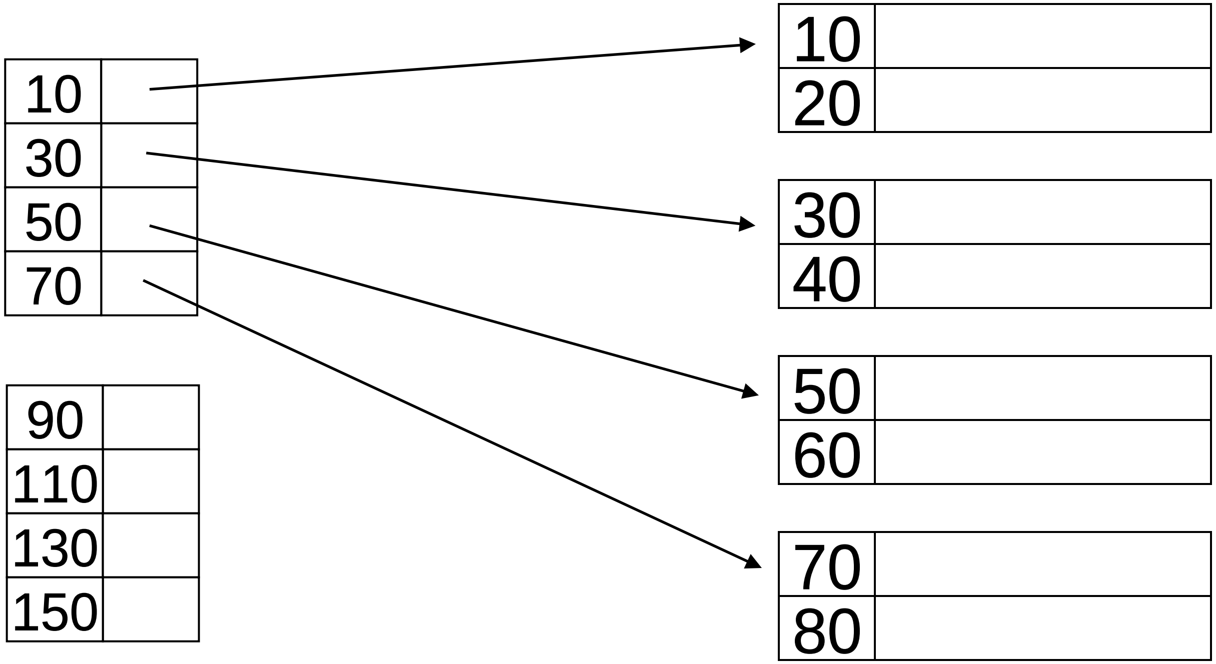
Deletion from sparse index

– delete record 40



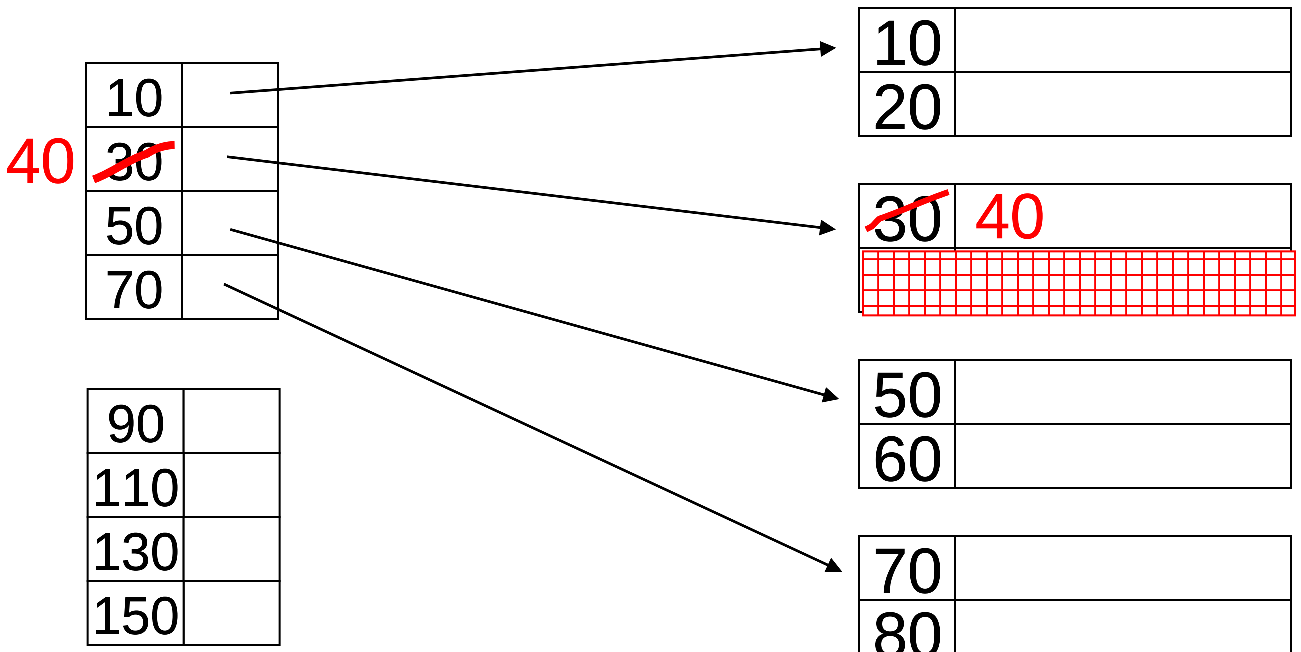
Deletion from sparse index

– delete record 30



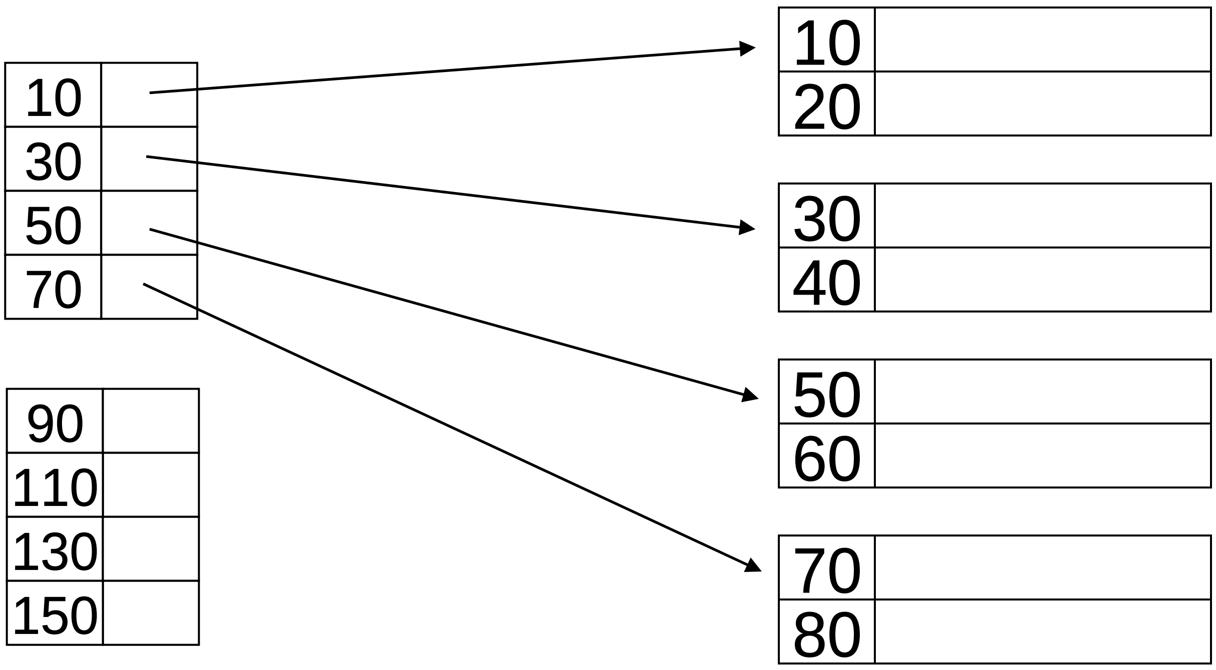
Deletion from sparse index

– delete record 30



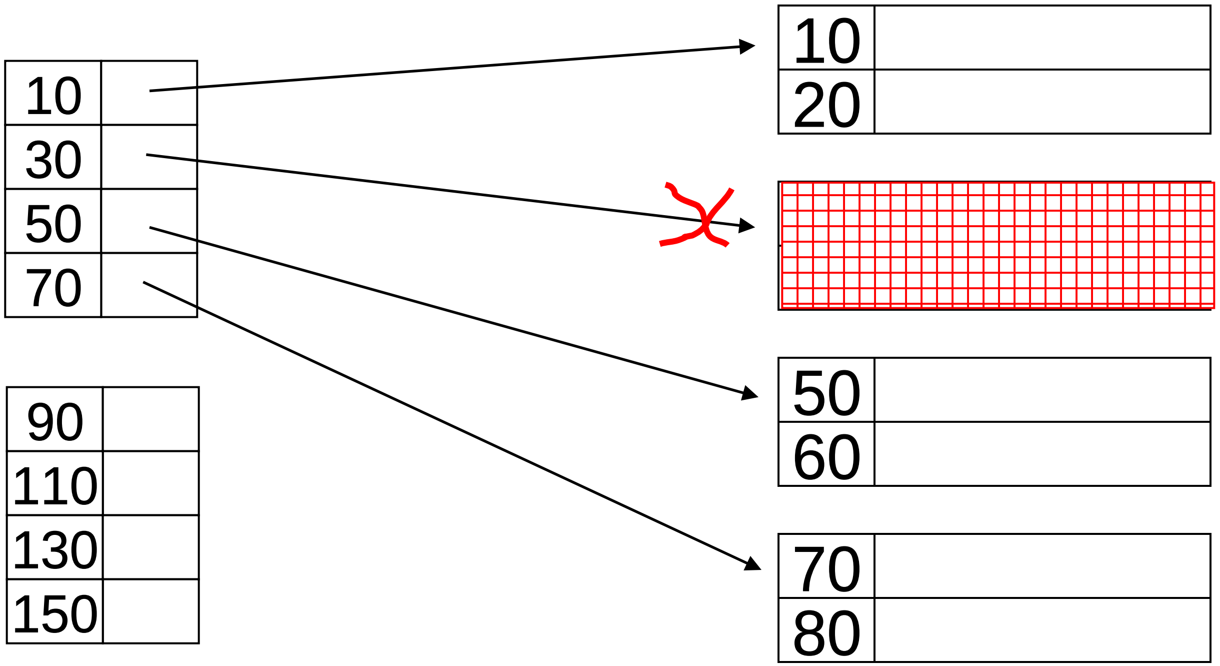
Deletion from sparse index

– delete records 30 & 40



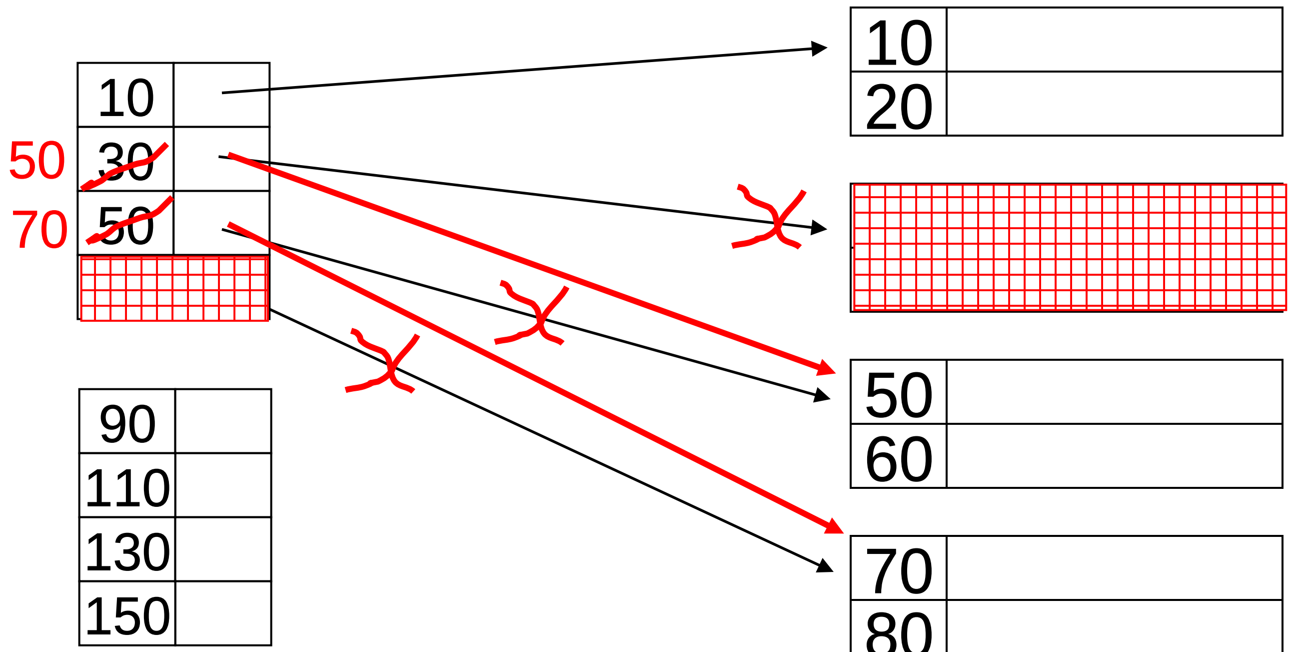
Deletion from sparse index

– delete records 30 & 40

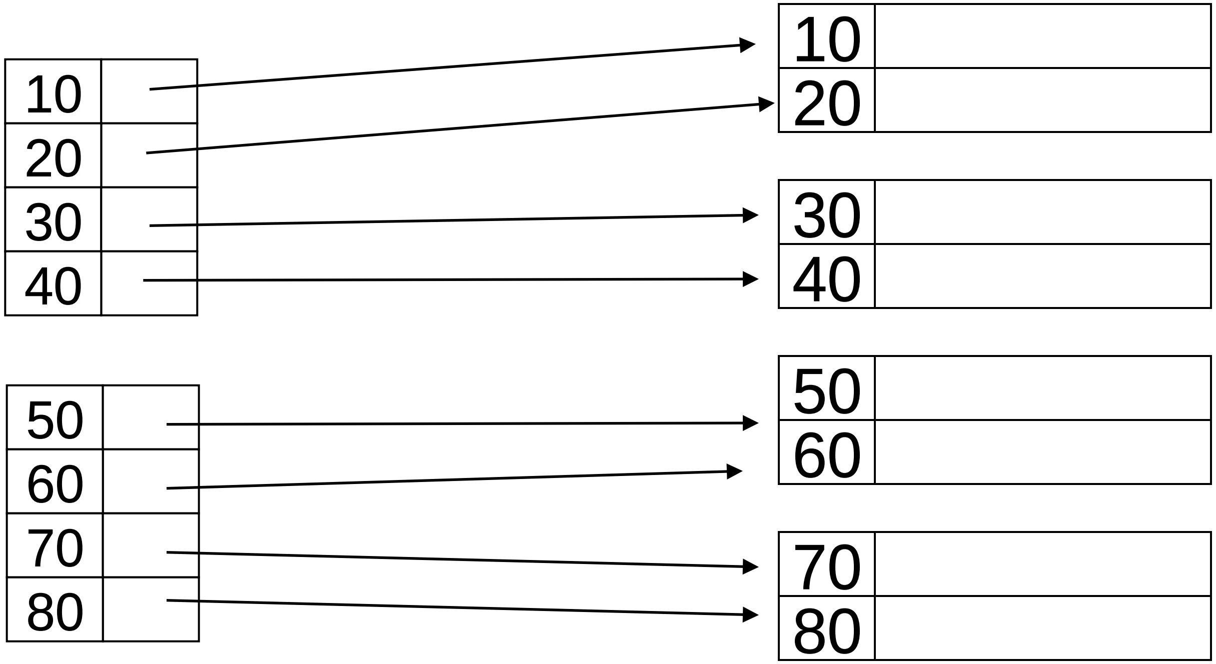


Deletion from sparse index

– delete records 30 & 40

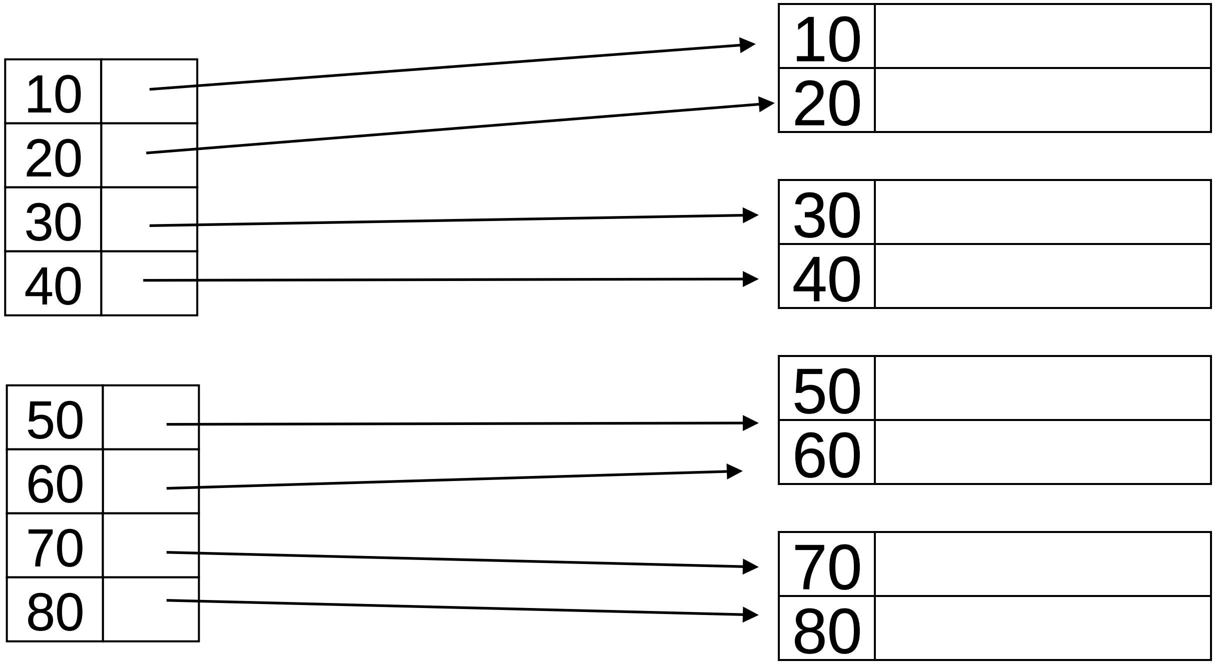


Deletion from dense index



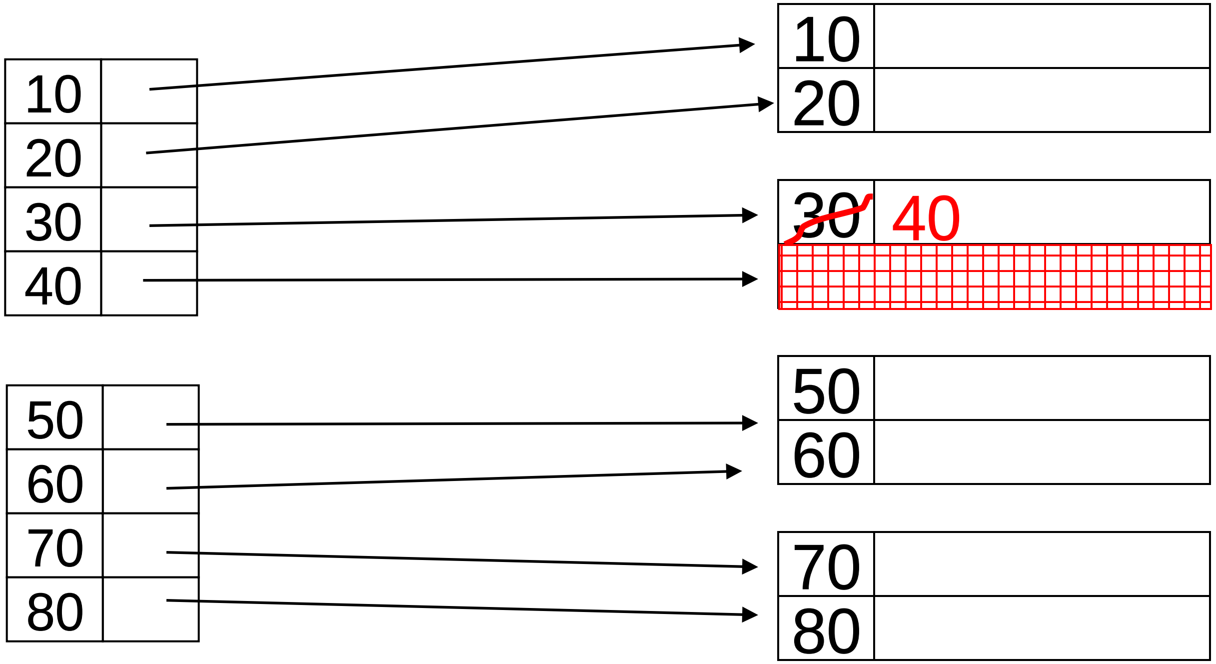
Deletion from dense index

– delete record 30



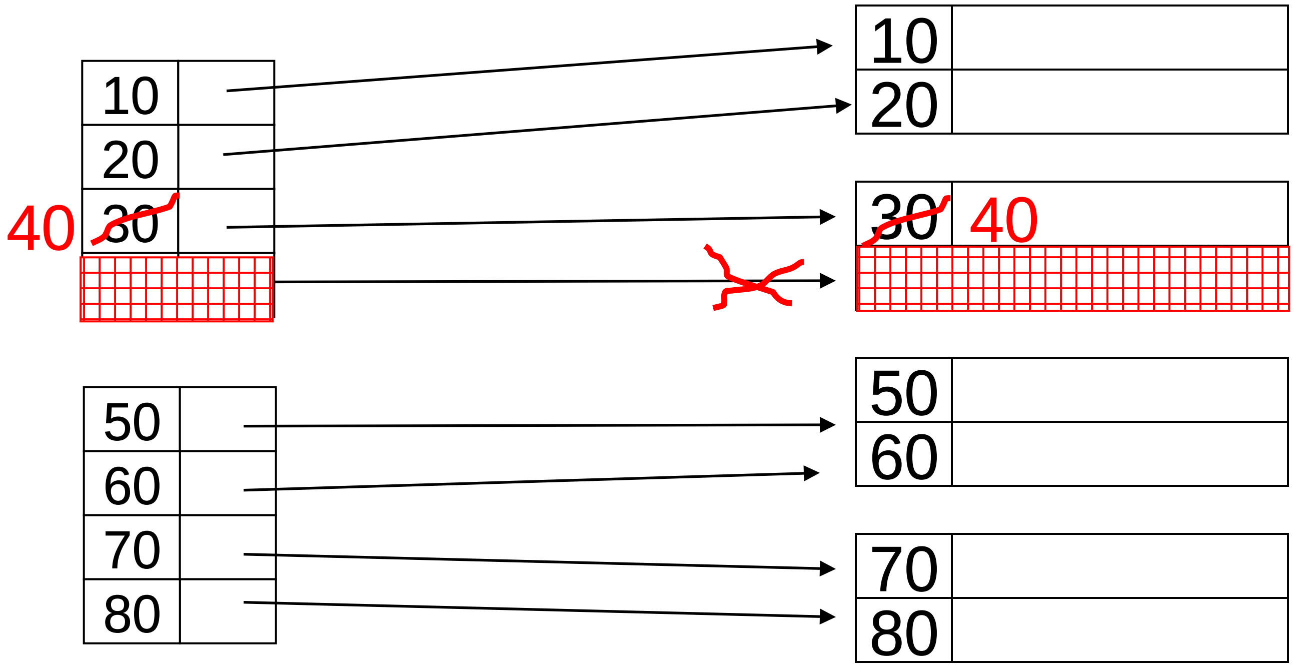
Deletion from dense index

– delete record 30

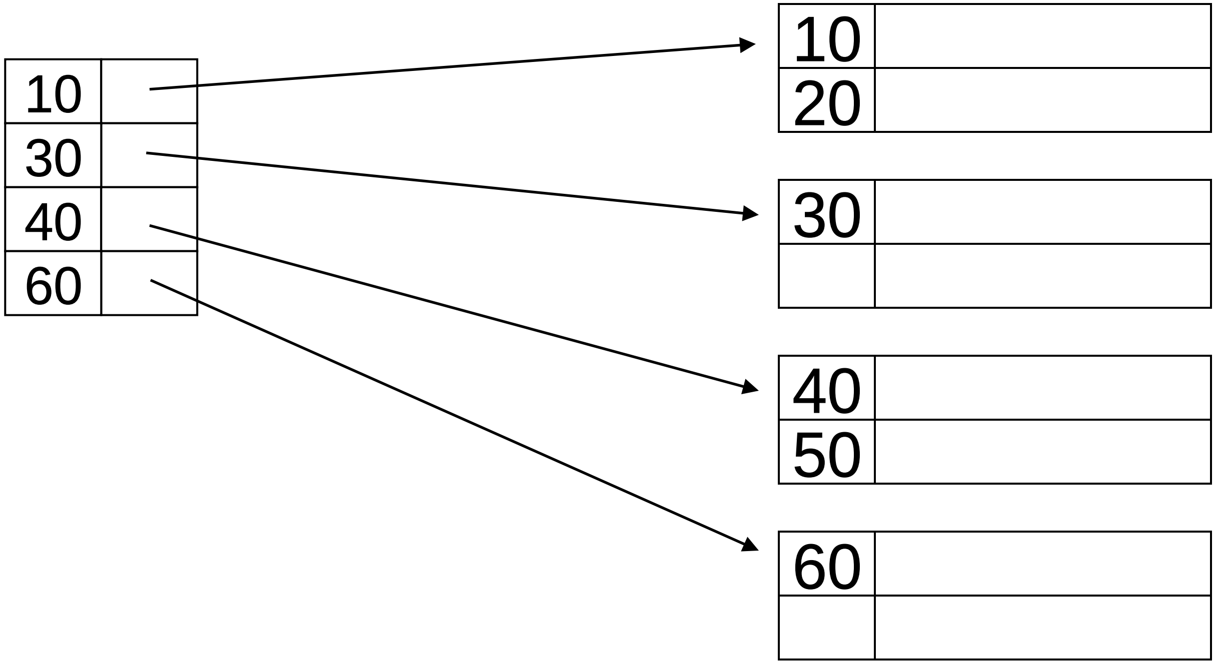


Deletion from dense index

– delete record 30

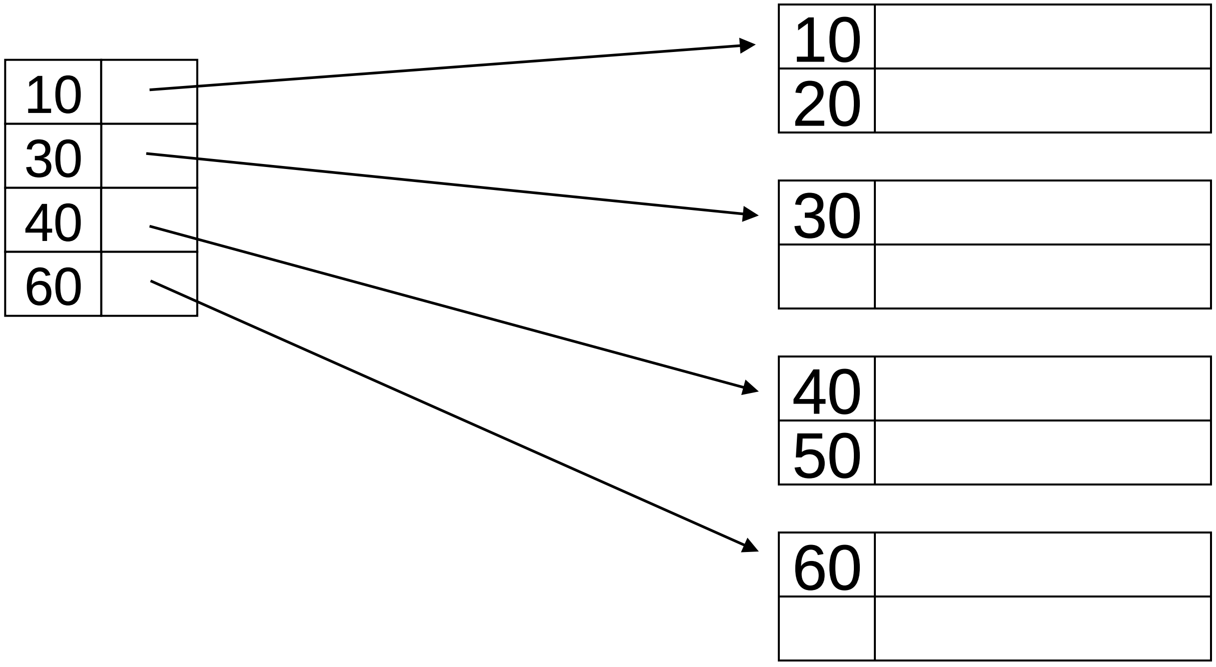


Insertion, sparse index case



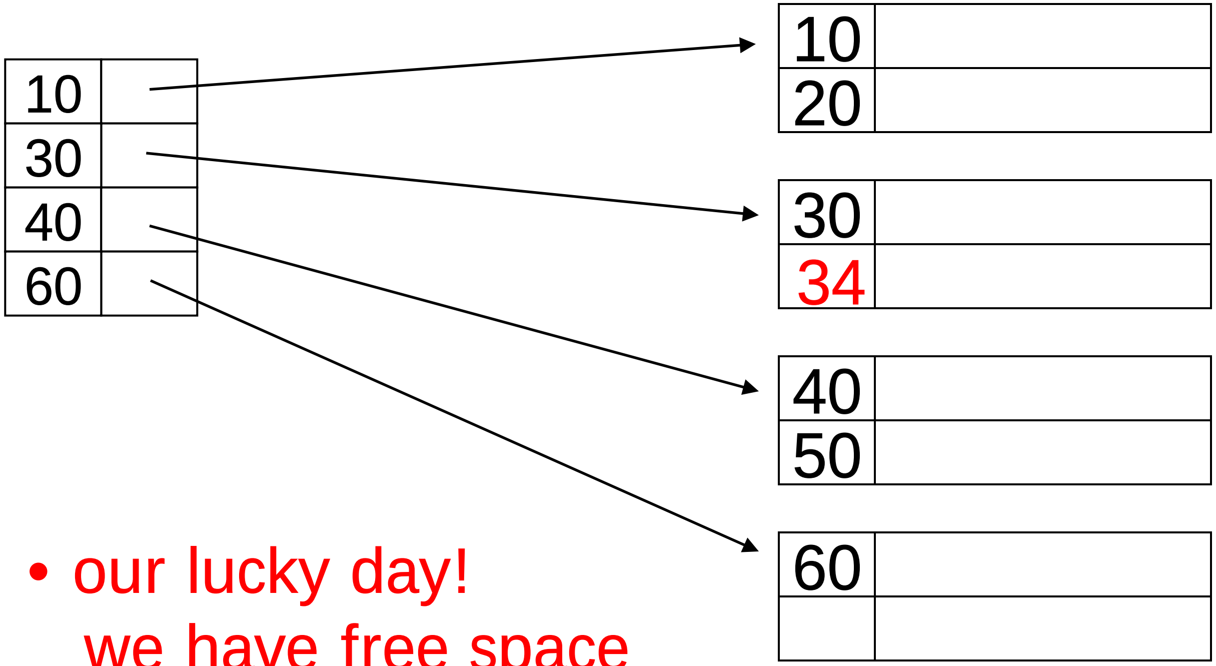
Insertion, sparse index case

– insert record 34



Insertion, sparse index case

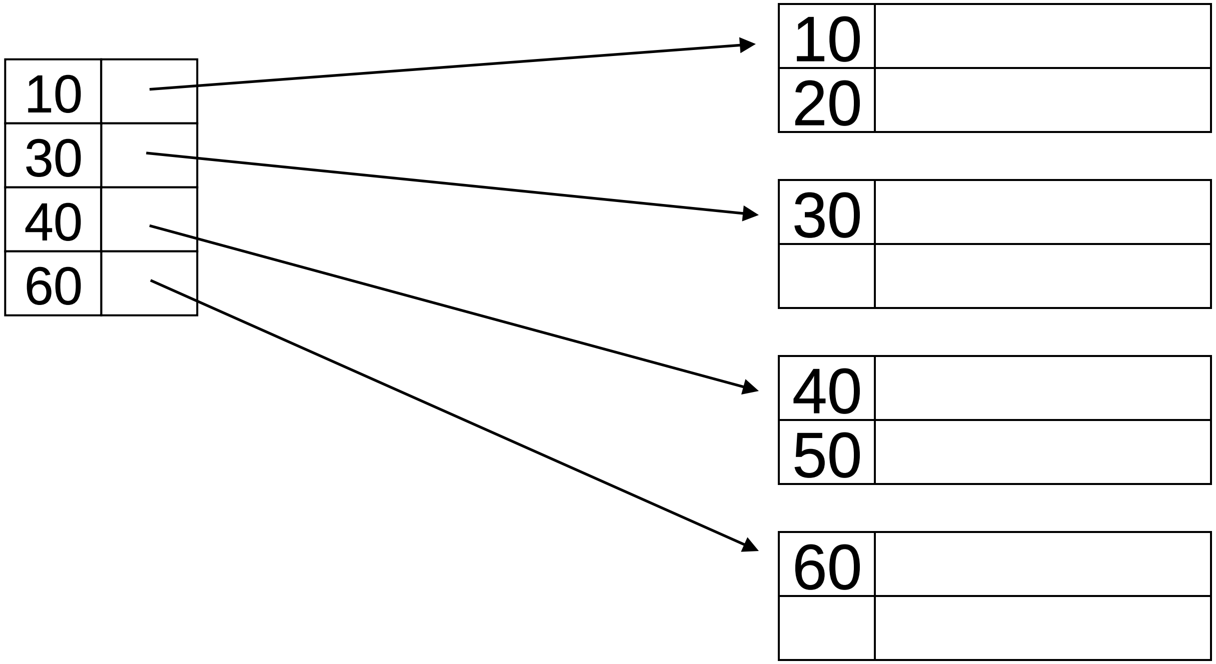
– insert record 34



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

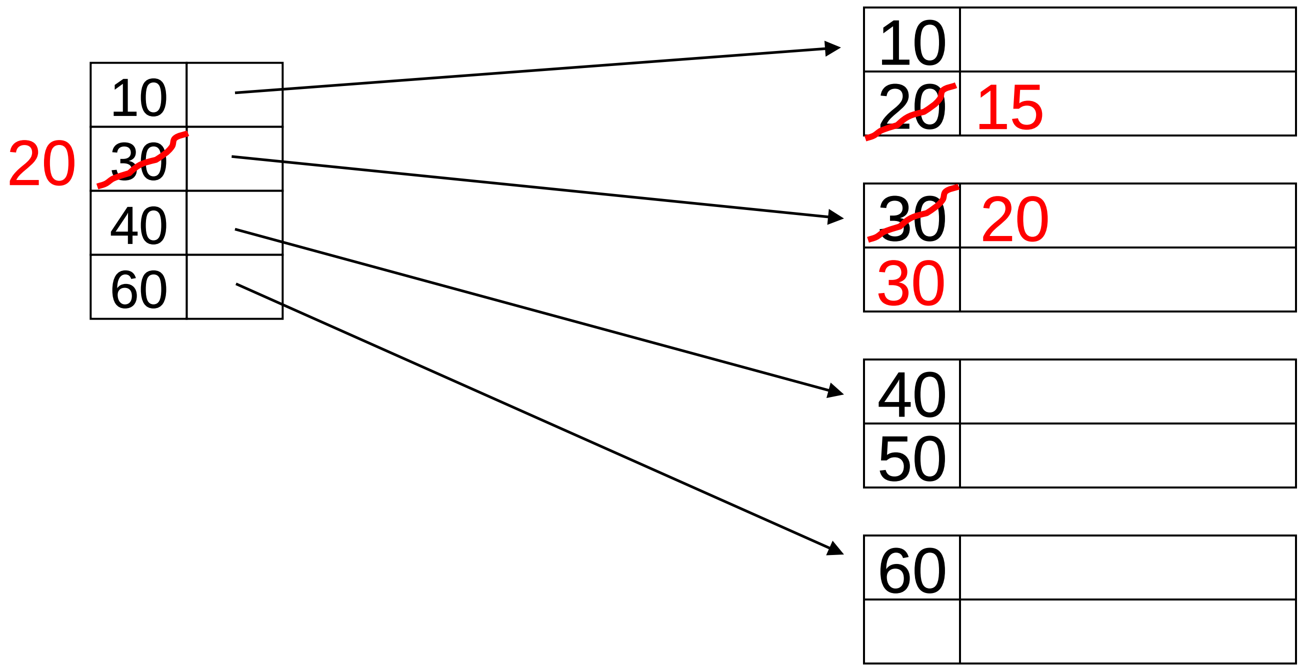
Insertion, sparse index case

– insert record 15



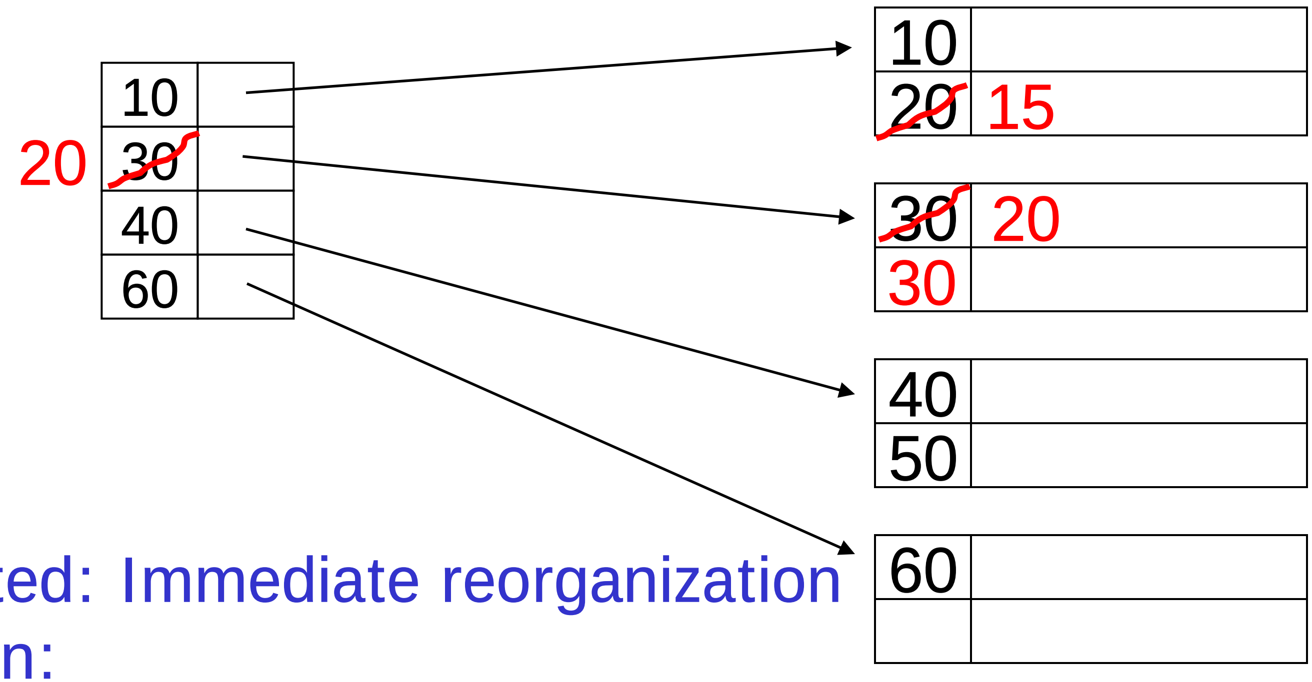
Insertion, sparse index case

– insert record 15



Insertion, sparse index case

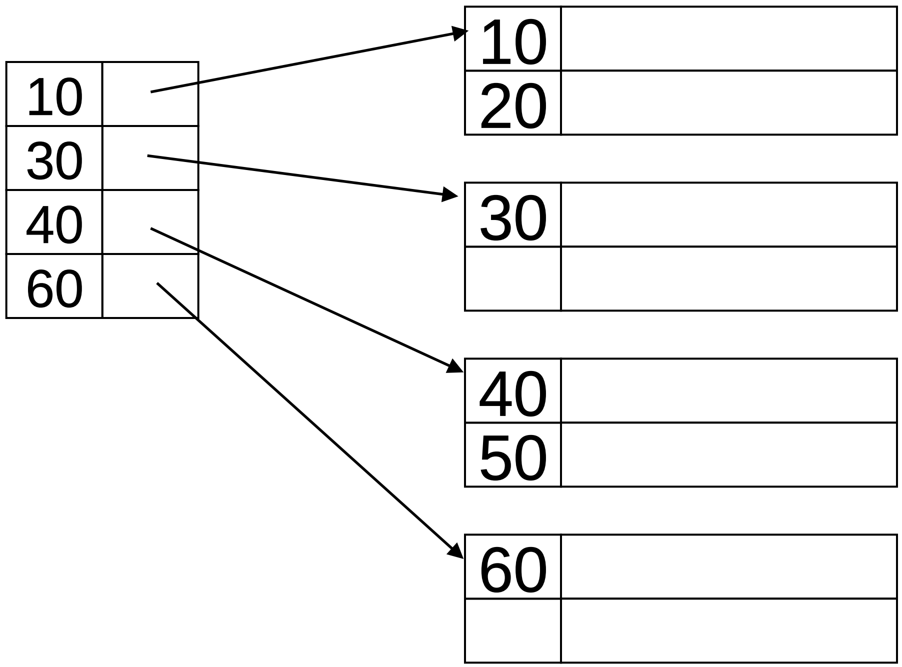
– insert record 15



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

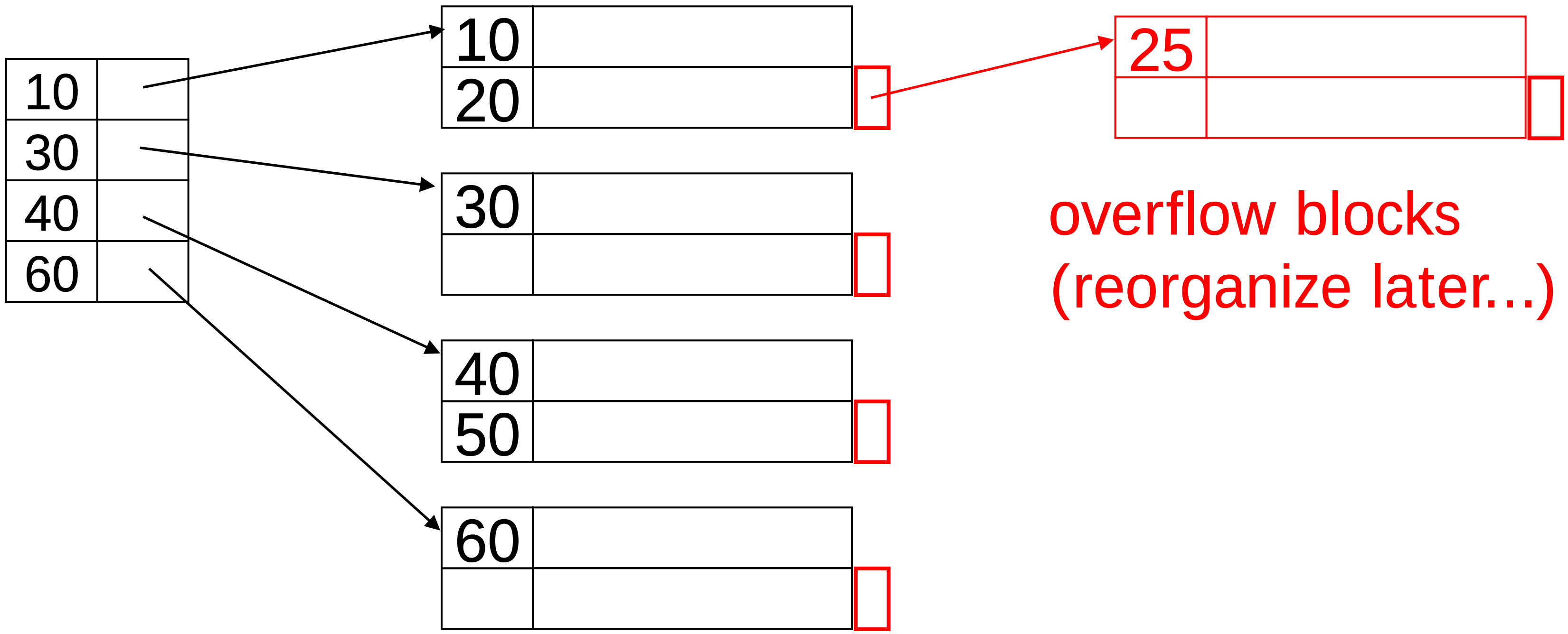
Insertion, sparse index case

– insert record 25



Insertion, sparse index case

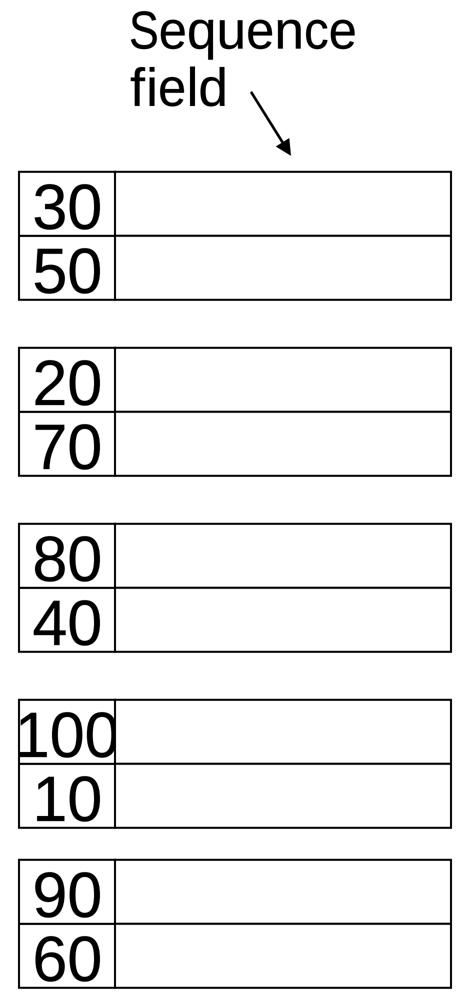
– insert record 25



Insertion, dense index case

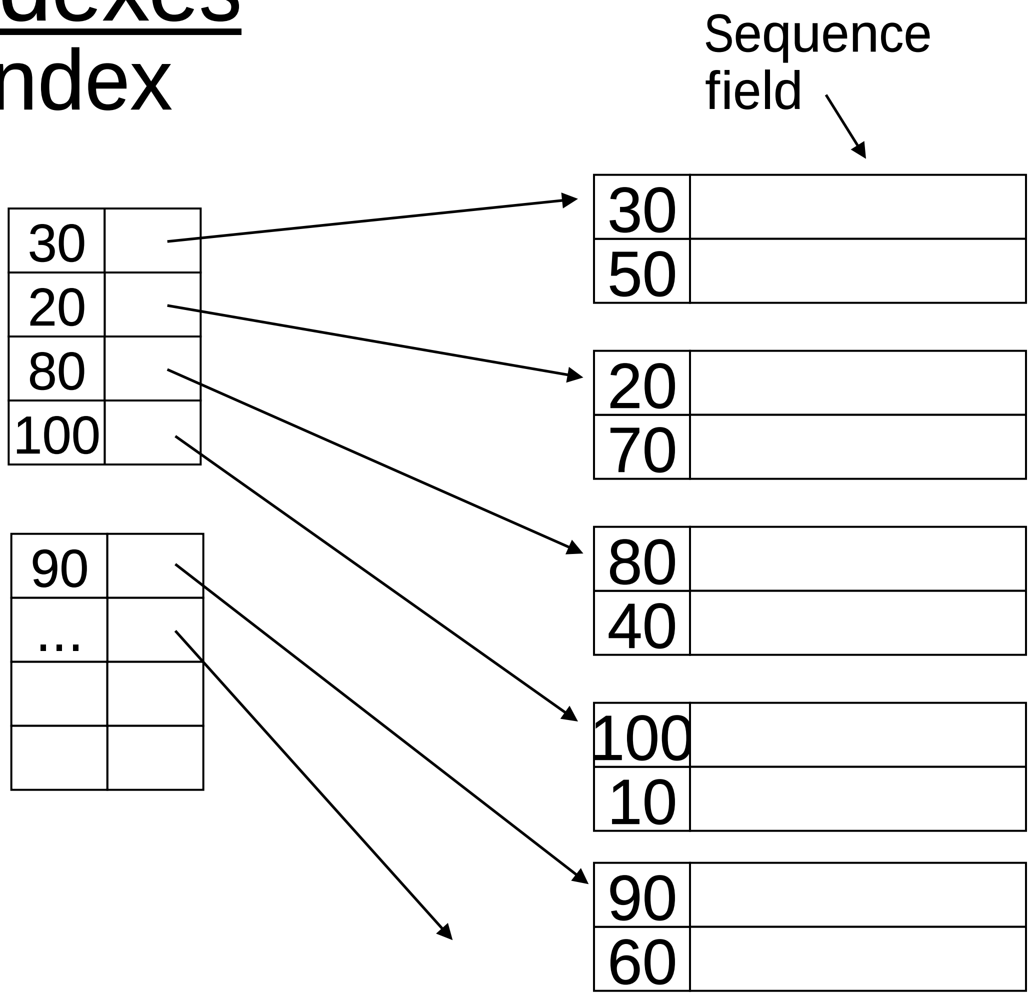
- Similar
- Often more expensive . . .

Secondary indexes



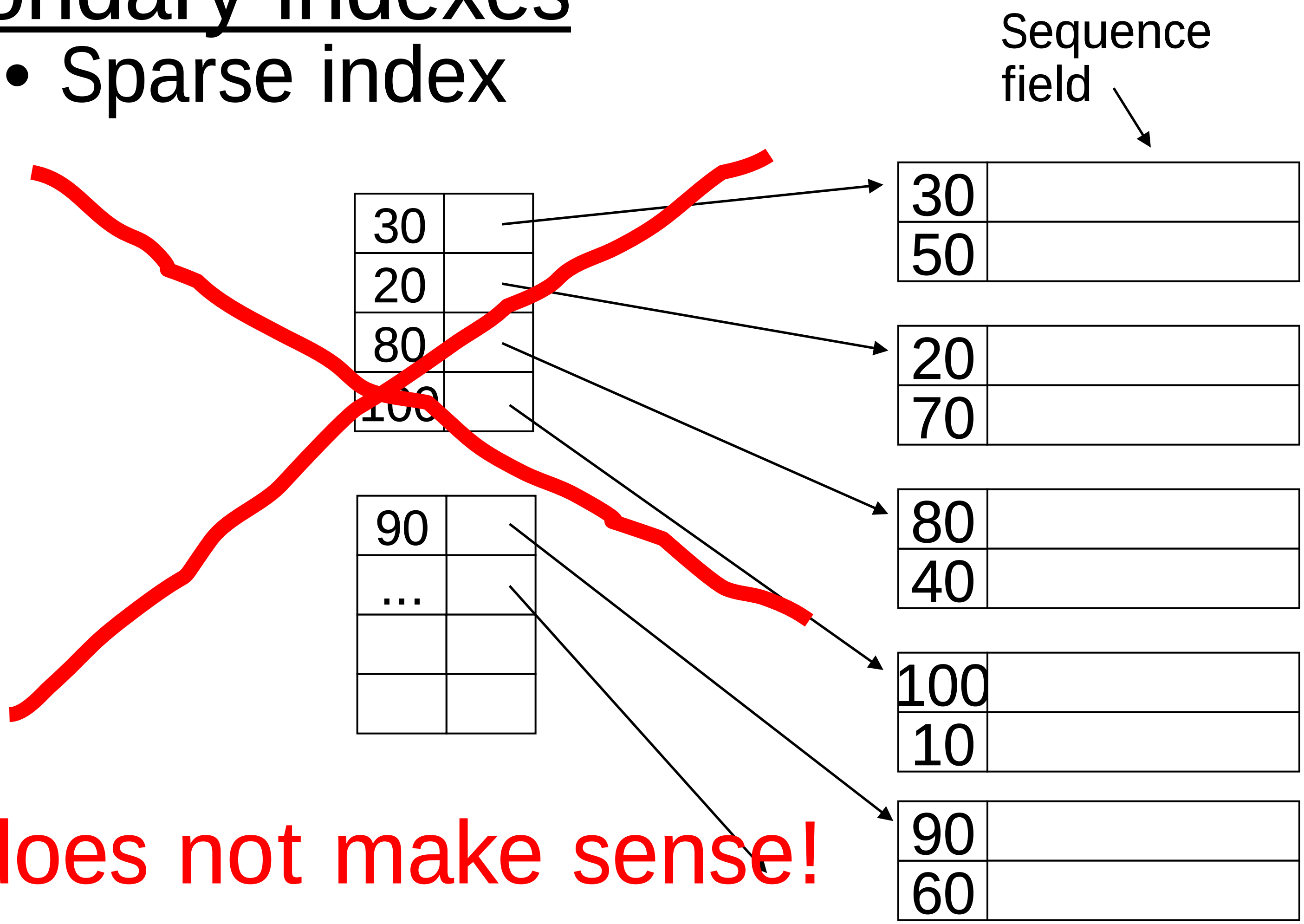
Secondary indexes

- Sparse index



Secondary indexes

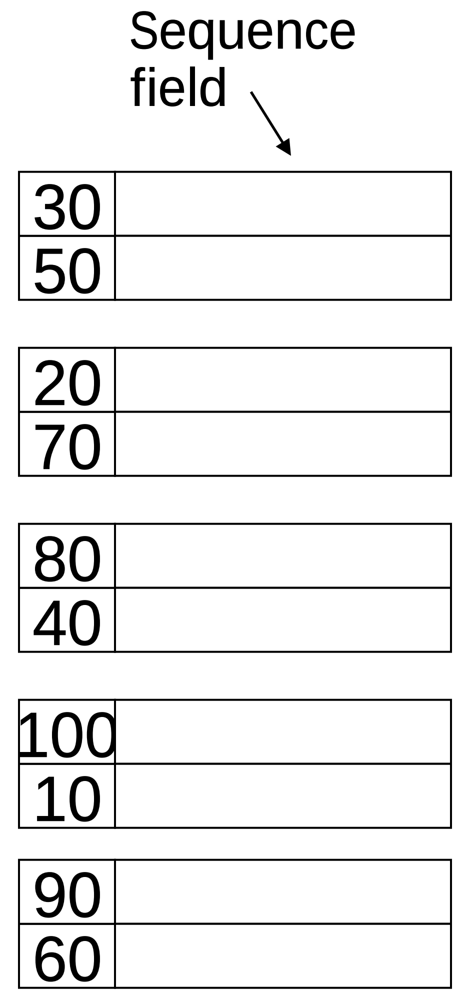
- Sparse index



does not make sense!

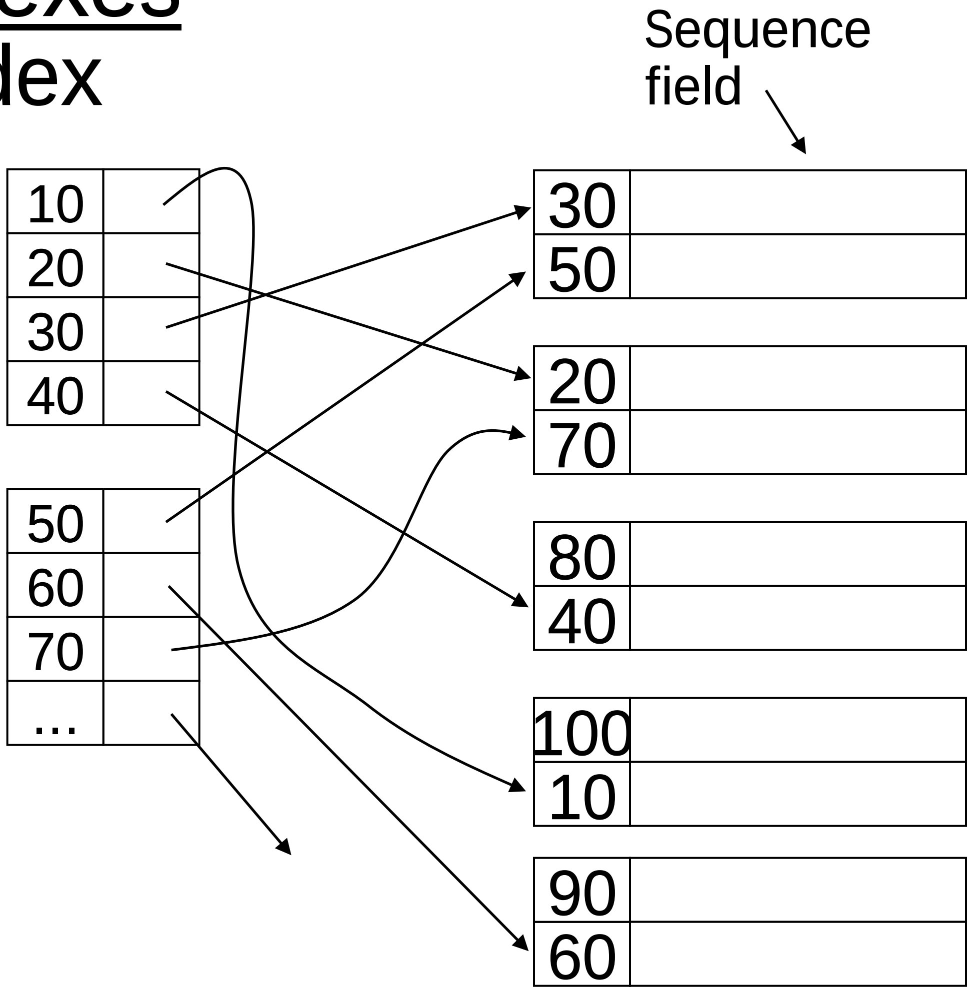
Secondary indexes

- Dense index



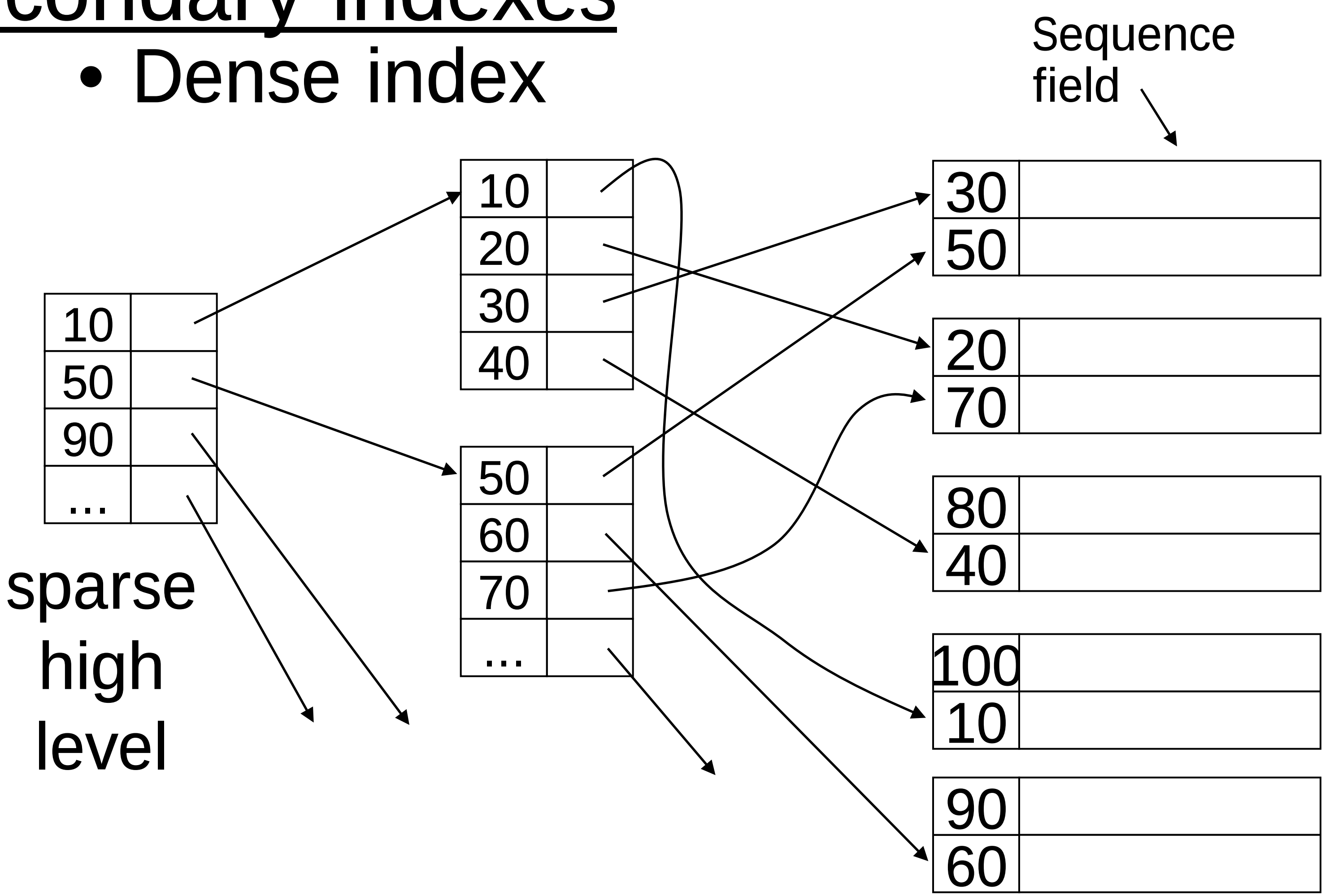
Secondary indexes

- Dense index



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	

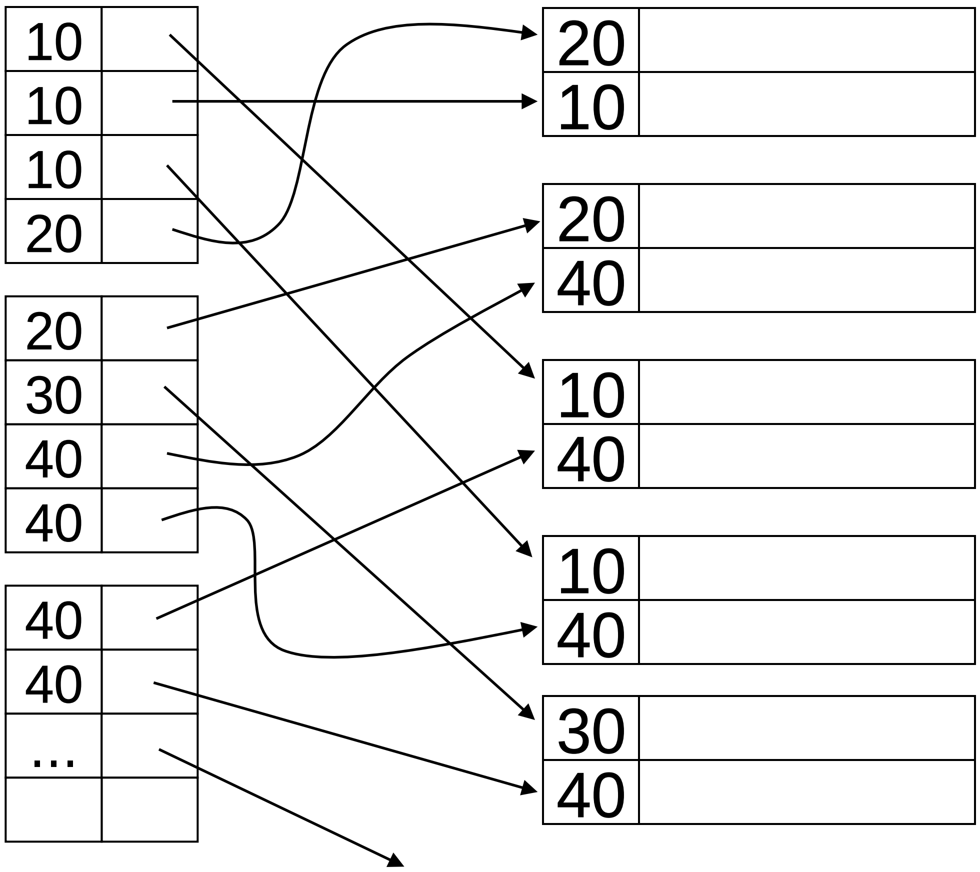
10	
40	

10	
40	

30	
40	

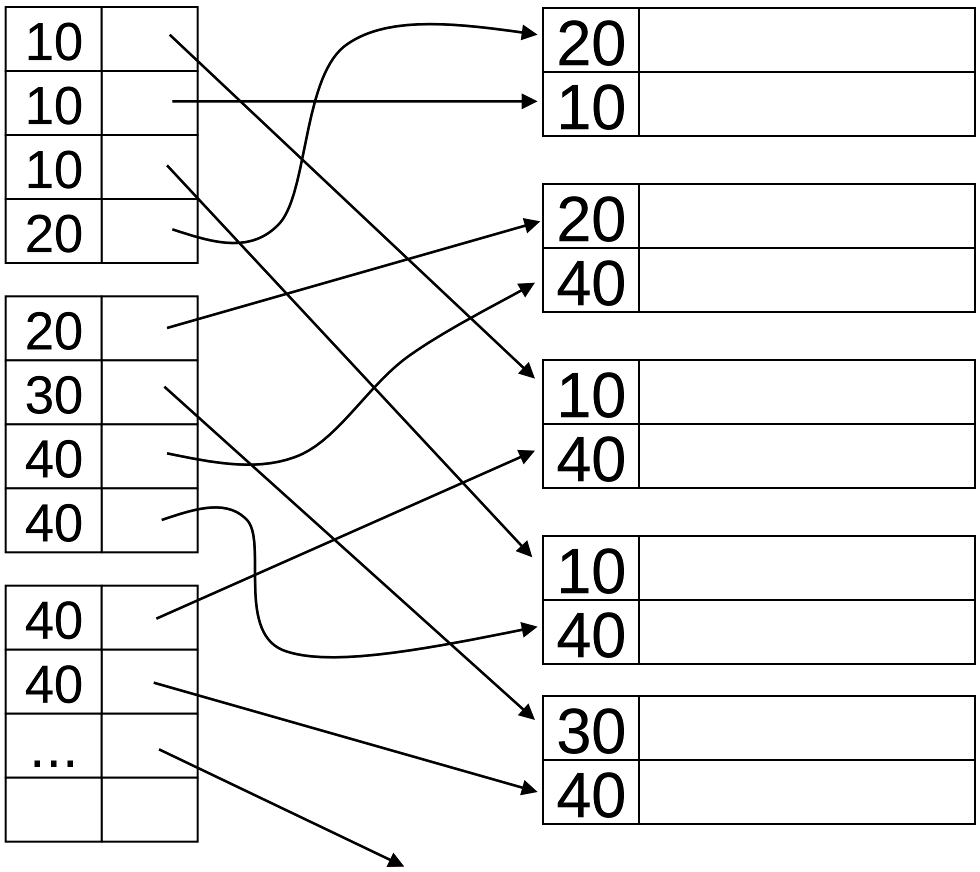
Duplicate values & secondary indexes

one option...



Duplicate values & secondary indexes

one option...



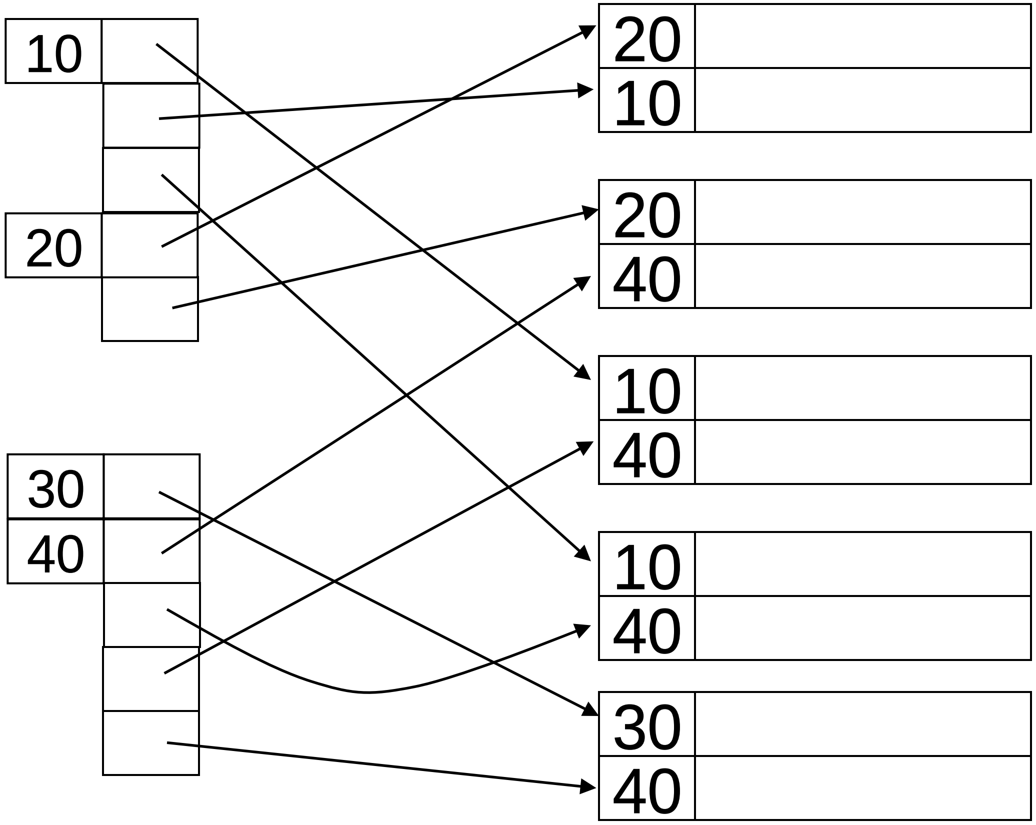
Problem:

excess overhead!

- disk space
- search time

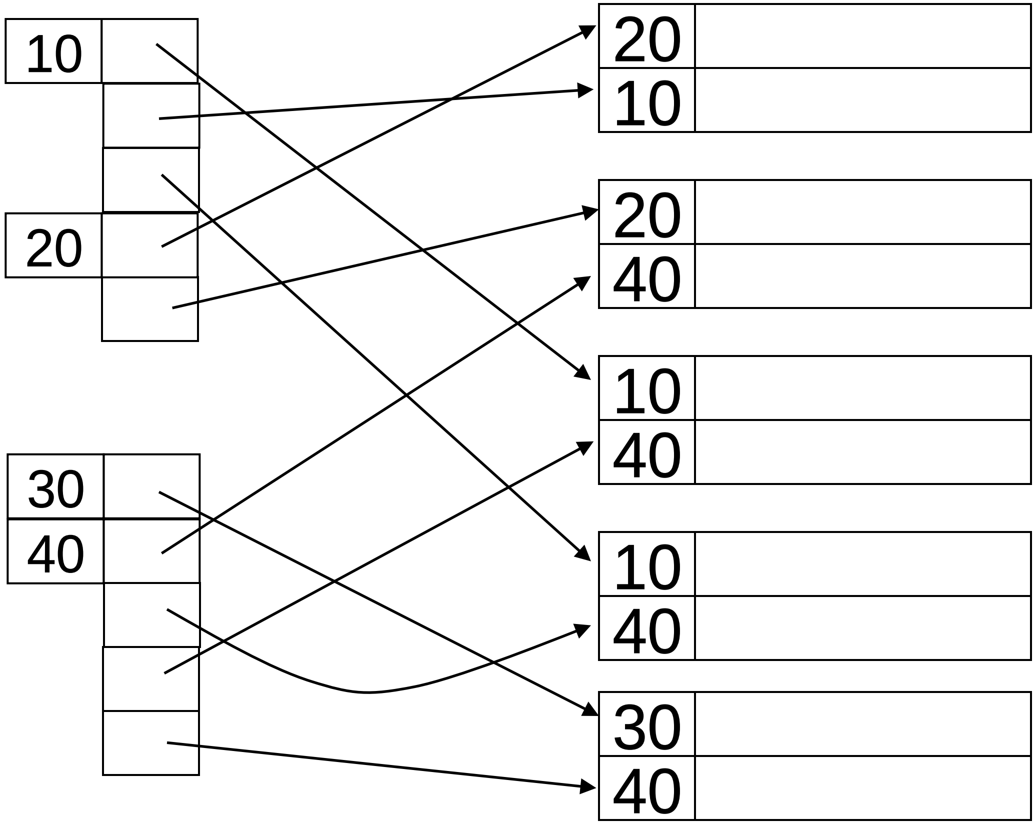
Duplicate values & secondary indexes

another option...



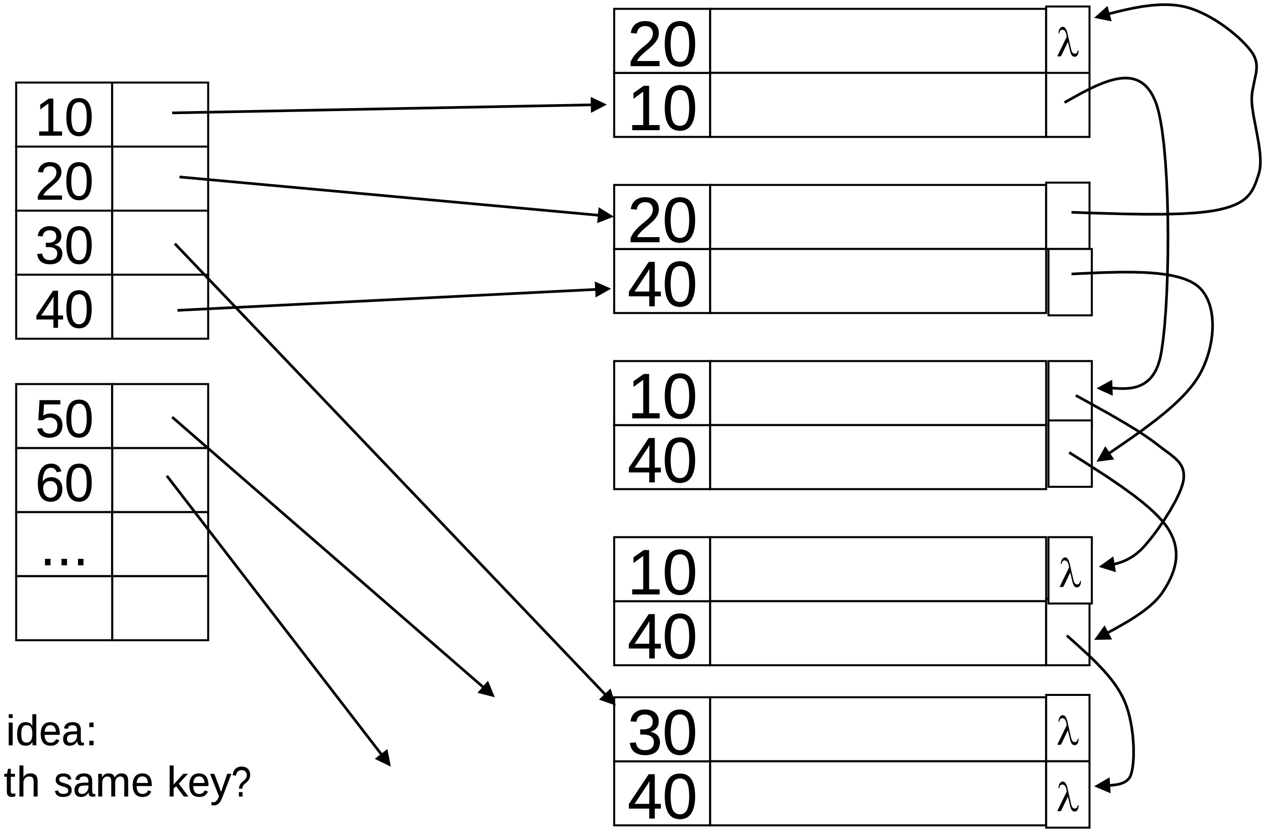
Duplicate values & secondary indexes

another option...



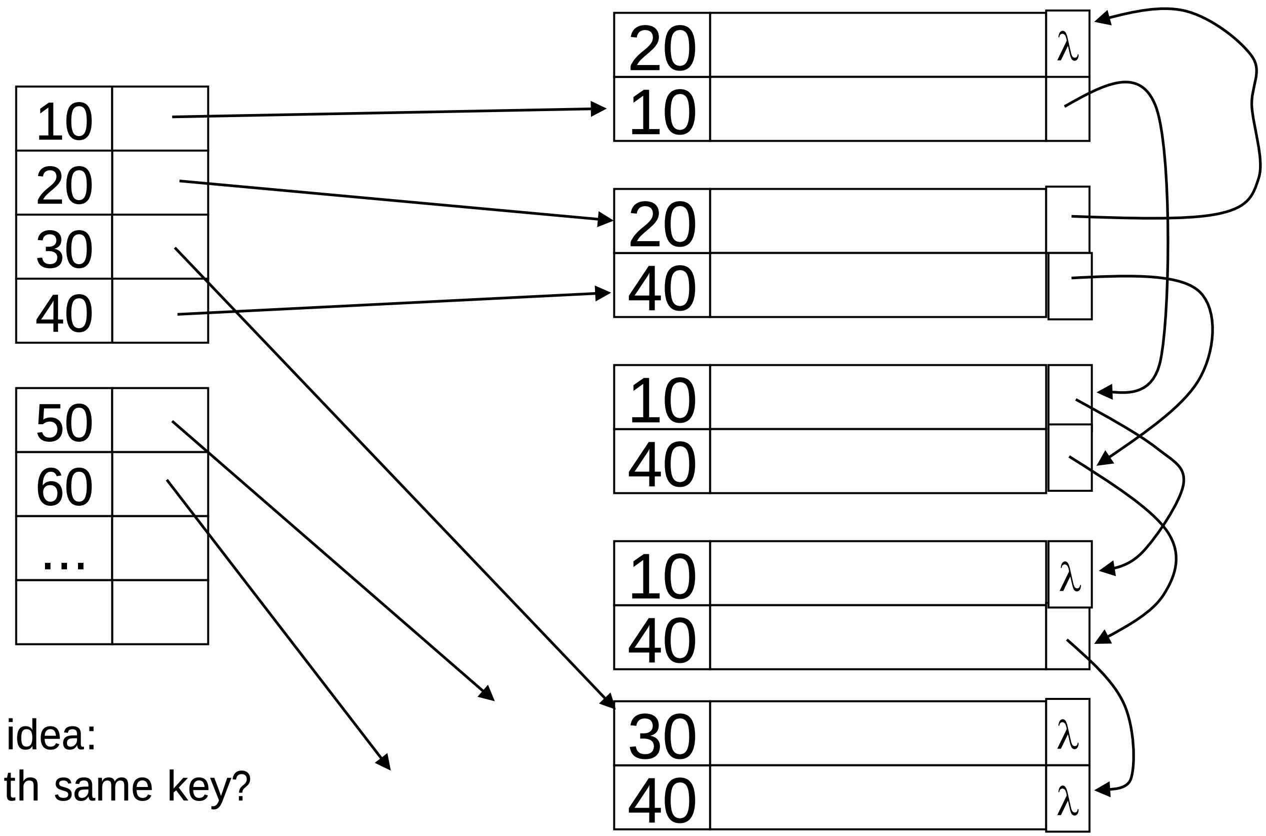
Problem:
variable size records in index!

Duplicate values & secondary indexes



Another idea:
Chain records with same key?

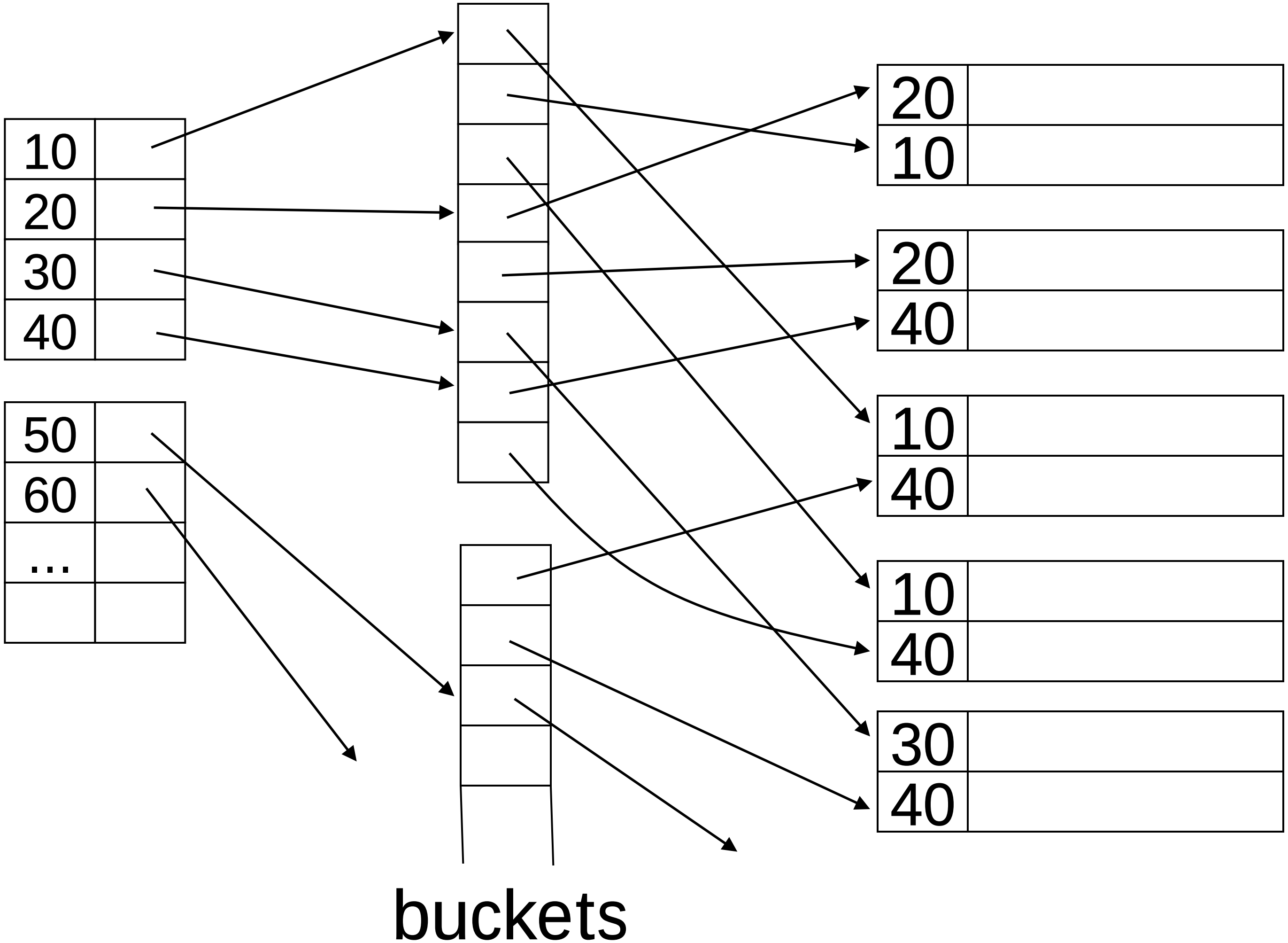
Duplicate values & secondary indexes



Another idea:
Chain records with same key?

Problems:
- Need to add fields to records
- Need to follow chain to know records

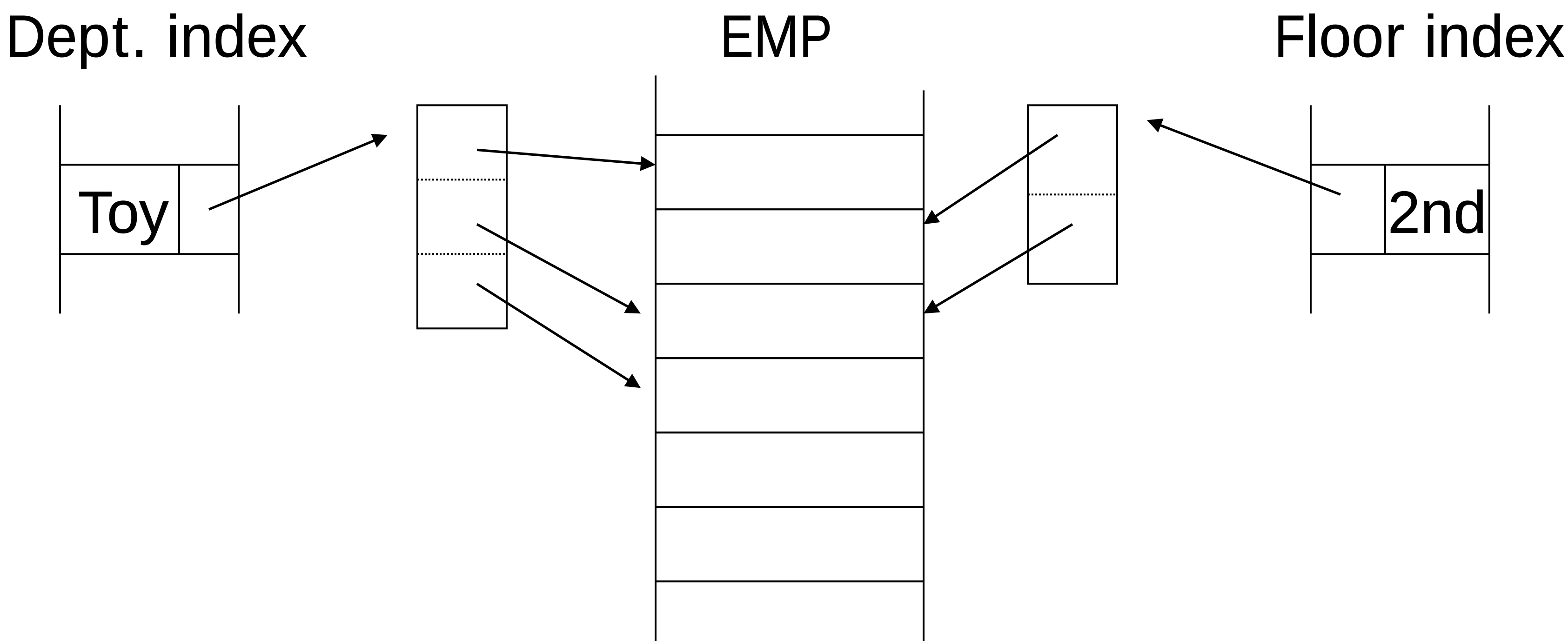
Duplicate values & secondary indexes



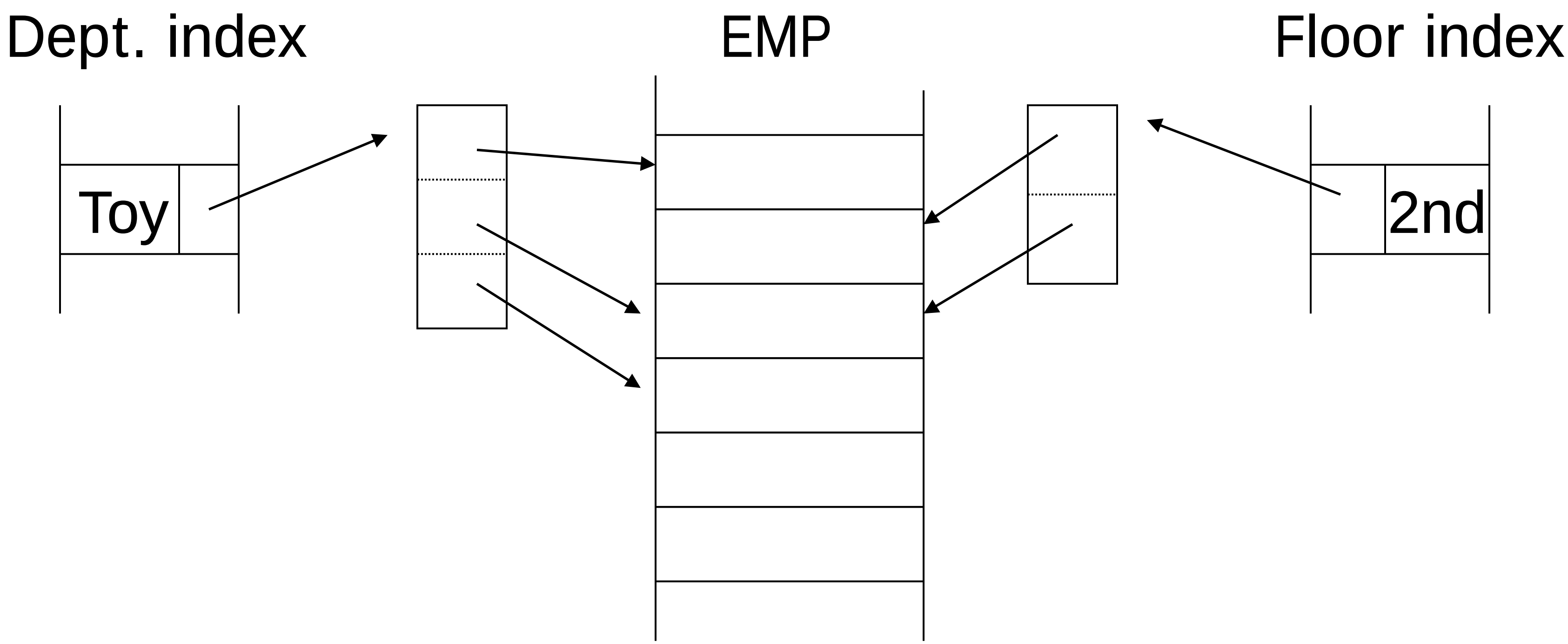
Why "bucket" idea is useful

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

Query: Get employees in (Toy Dept) ∧ (2nd floor)



Query: Get employees in (Toy Dept) ∧ (2nd floor)



→ Intersect toy bucket and 2nd Floor bucket to get set of matching EMP’s

This idea used in text information retrieval

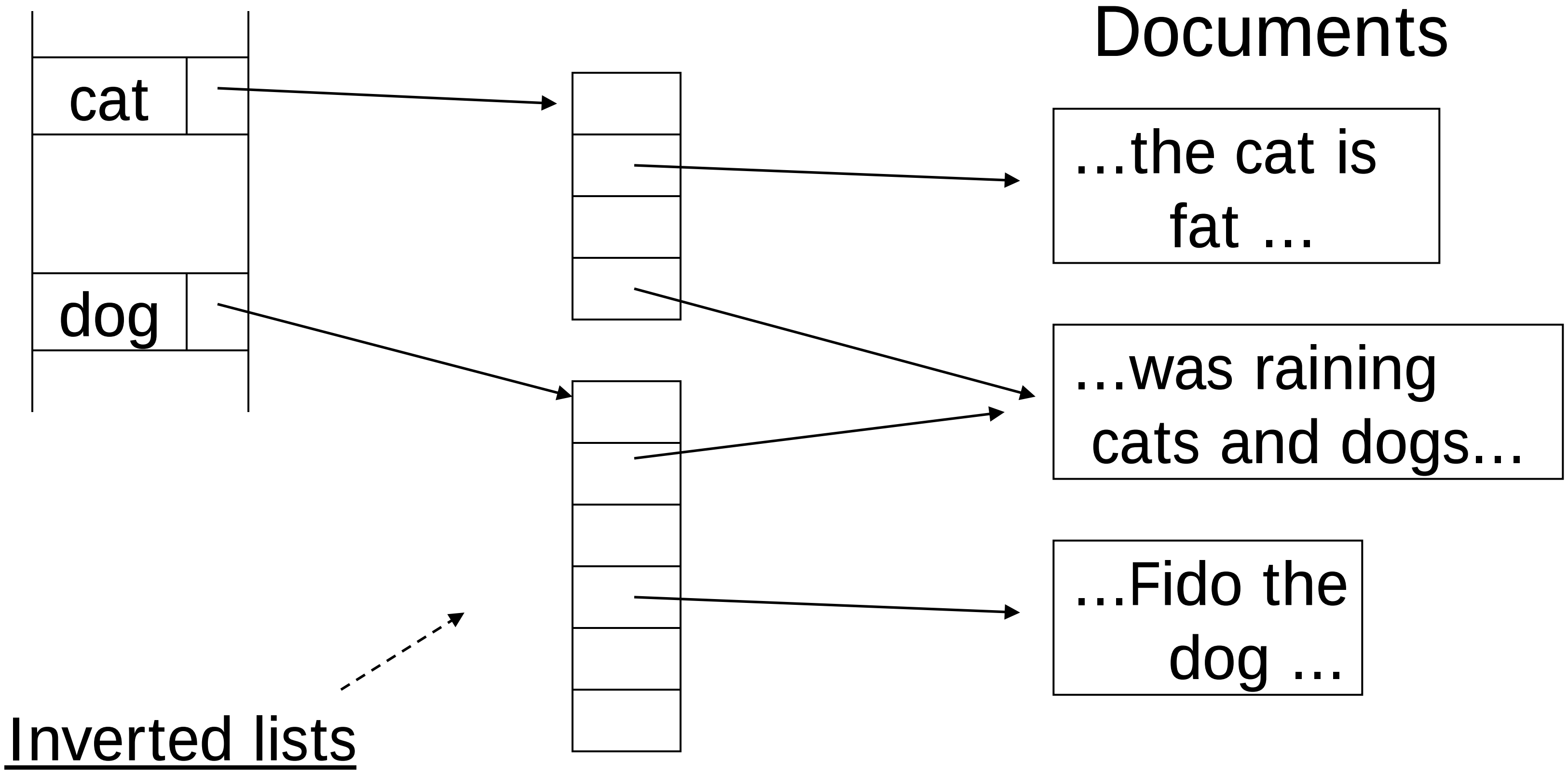
Documents

...the cat is fat ...

...was raining cats and dogs...

...Fido the dog ...

This idea used in text information retrieval



IR QUERIES

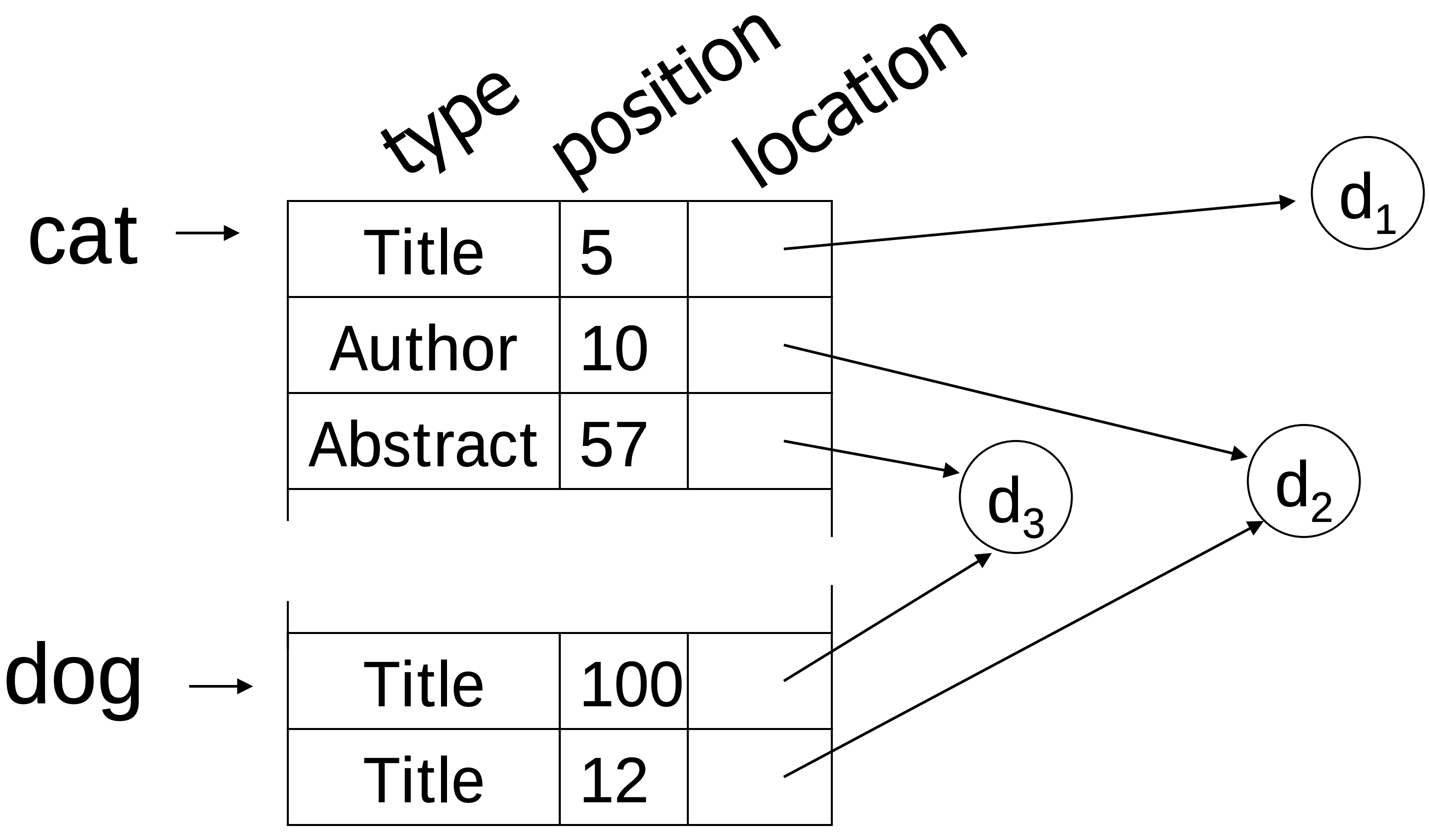
- Find articles with “cat” and “dog”
- Find articles with “cat” or “dog”
- Find articles with “cat” and not “dog”

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

Common technique:
more info in inverted list



Posting: an entry in inverted list. Represents occurrence of term in article

Size of a list: 1 Rare words or miss-spellings
(in postings)

↓

10^6 Common words

Size of a posting: 10-15 bits (compressed)

IR DISCUSSION

- Stop words
- Truncation
- Thesaurus
- Full text vs. Abstracts
- Vector model

Vector space model

	w1	w2	w3	w4	w5	w6	w7	...
DOC =	<1	0	0	1	1	0	0	...>
Query=	<0	0	1	1	0	0	0	...>

Vector space model

	w1	w2	w3	w4	w5	w6	w7	...
DOC =	<1	0	0	1	1	0	0	...>
Query=	<0	0	1	1	0	0	0	...>

PRODUCT = 1 + ... = score

- Tricks to weigh scores + normalize

e.g.: Match on common word not as useful as match on rare words...

- How to process V.S. Queries?

	w1	w2	w3	w4	w5	w6	...	
Q = <	0	0	0	1	1	0	...	>

- Try Stanford Libraries
- Try Google, Yahoo, ...

Summary so far

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

Conventional indexes

Advantage:

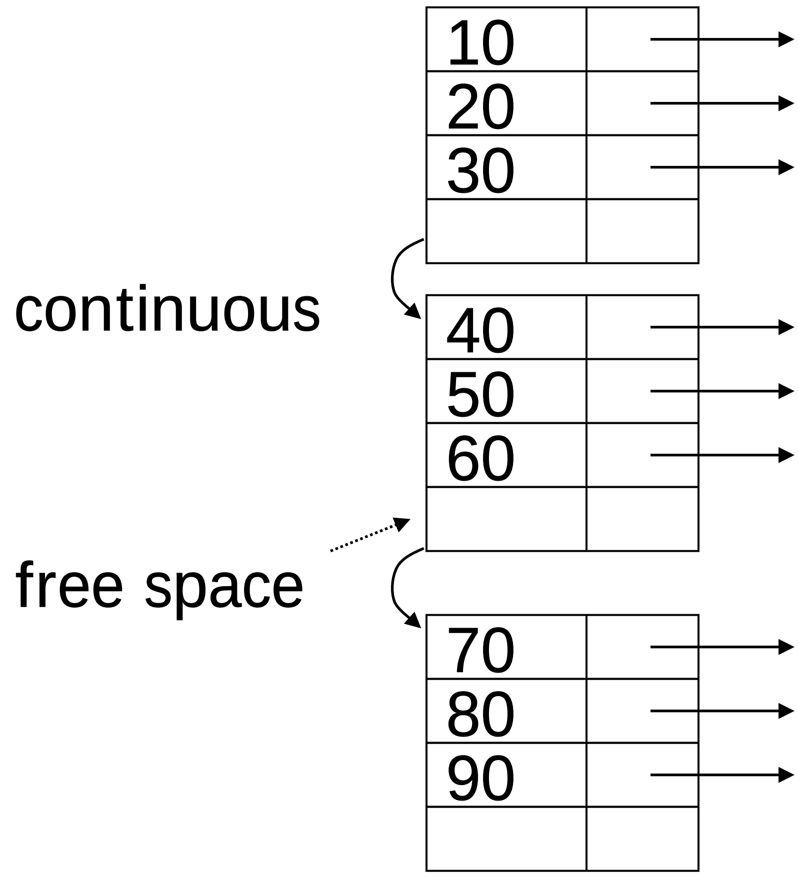
- Simple
- Index is sequential file
  good for scans

Disadvantage:

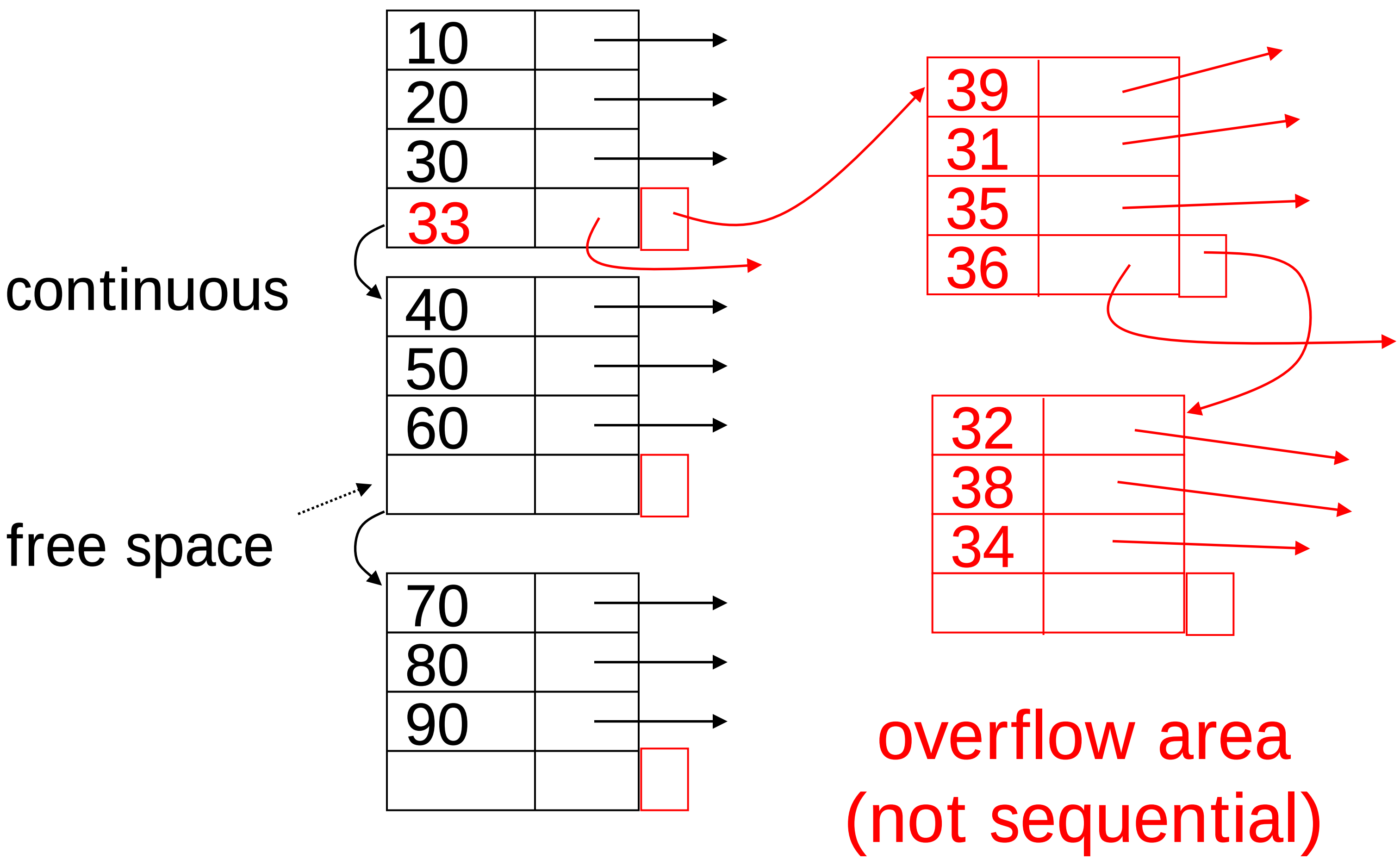
- Inserts expensive, and/or
- Lose sequentiality & balance

Example

Index (sequential)



Example Index (sequential)

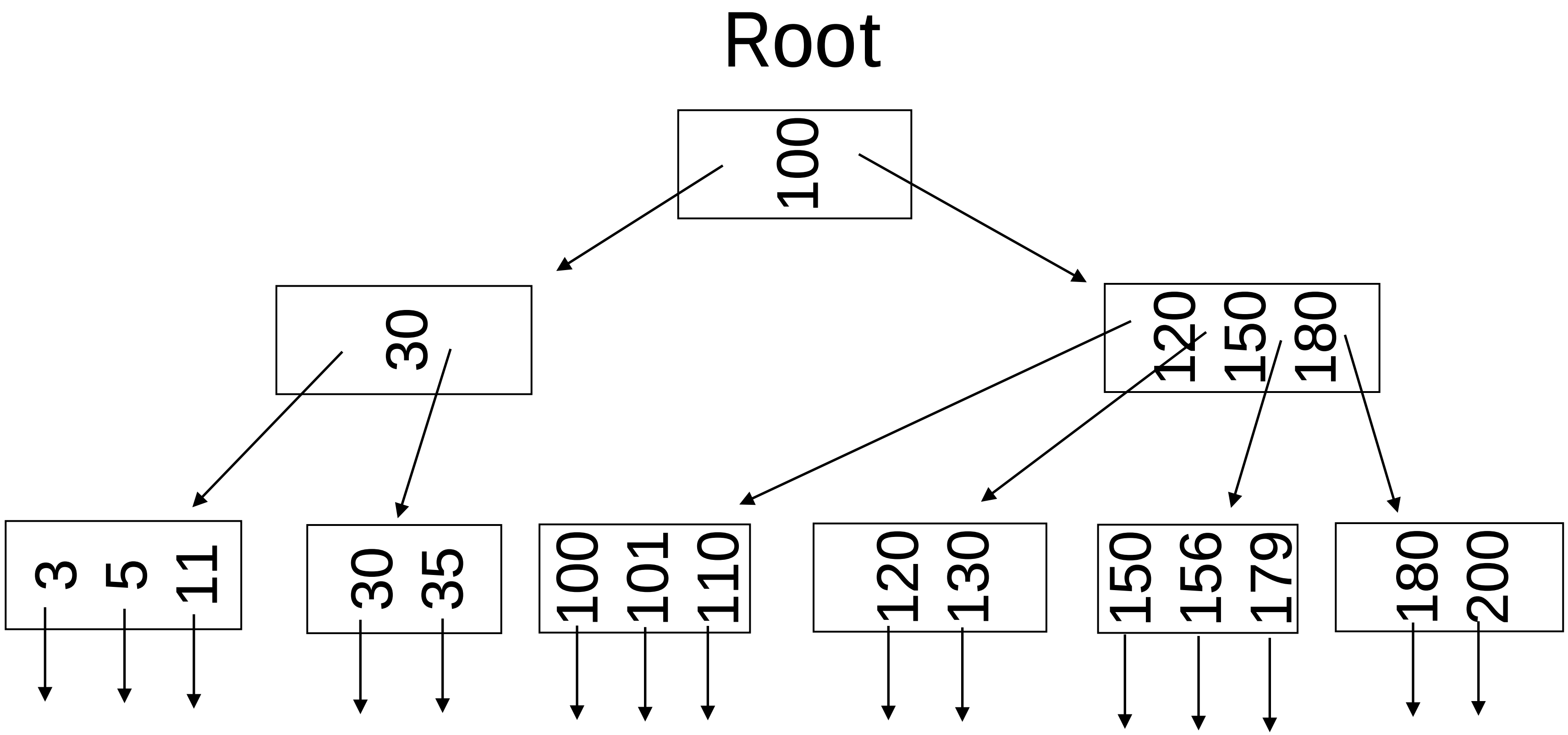


Outline:

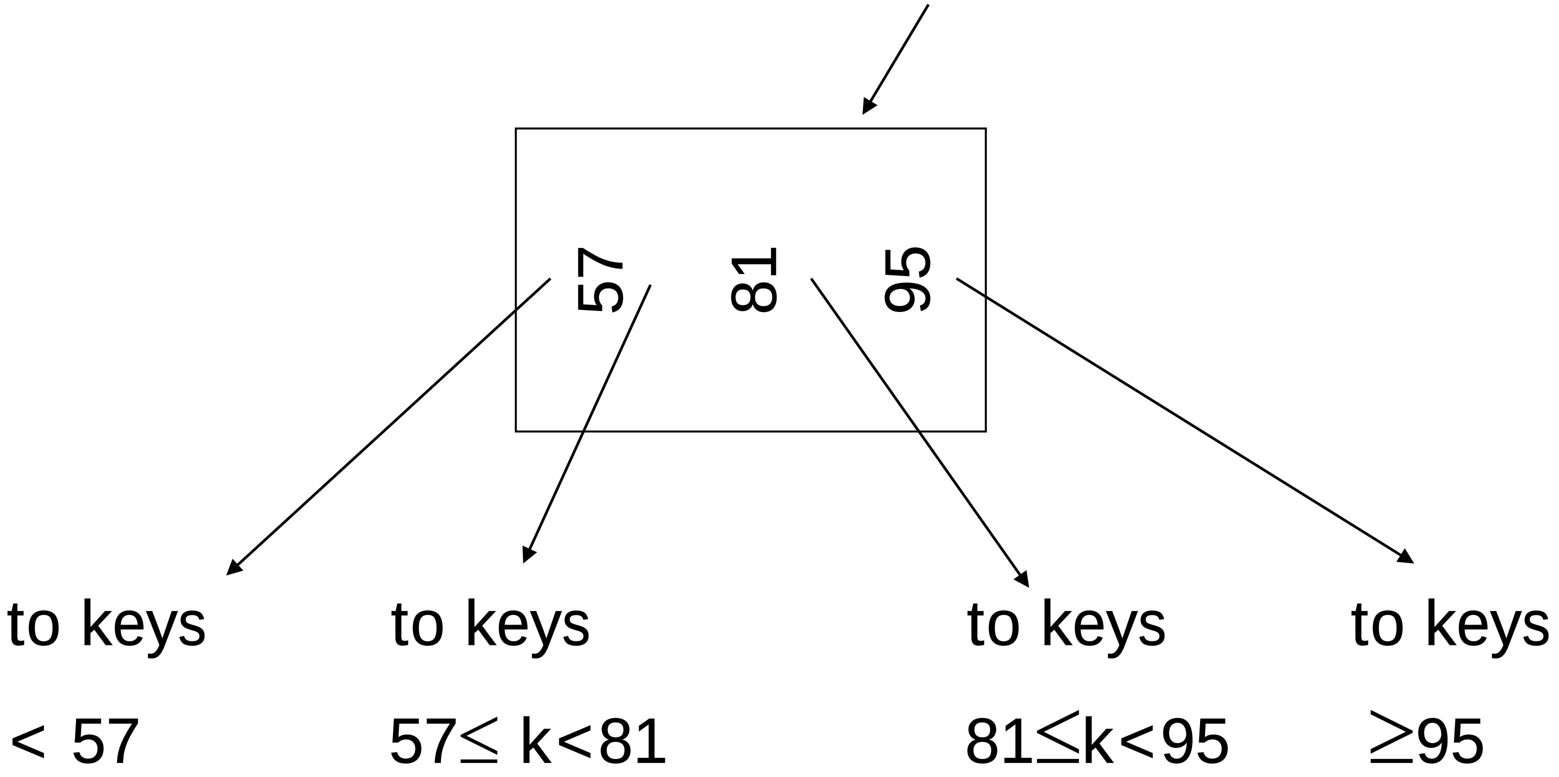
- Conventional indexes
- B-Trees $\Rightarrow$ NEXT
- Hashing schemes

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

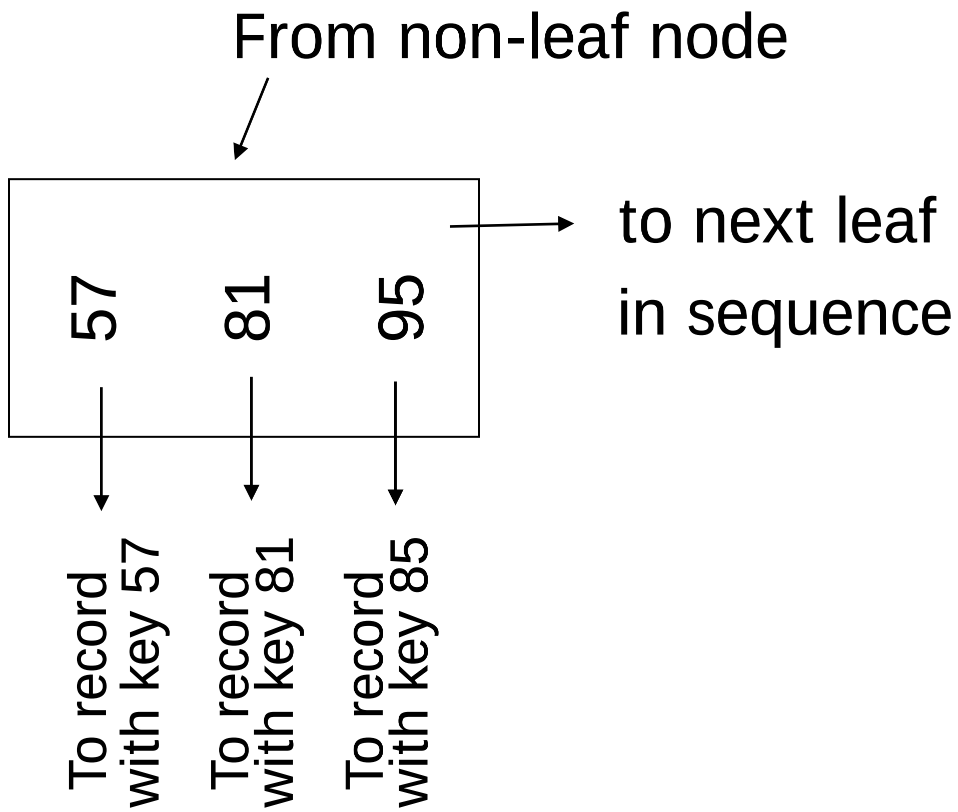
B+Tree Example n=3



Sample non-leaf

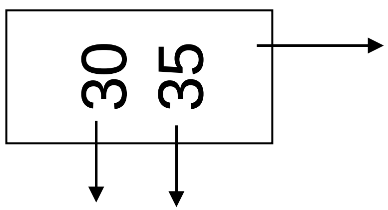


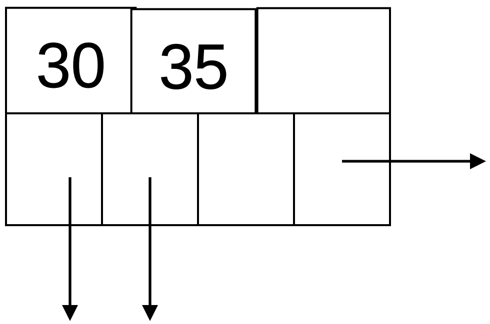
Sample leaf node:



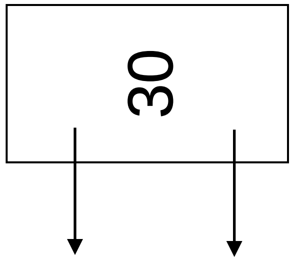
In textbook's notation n=3

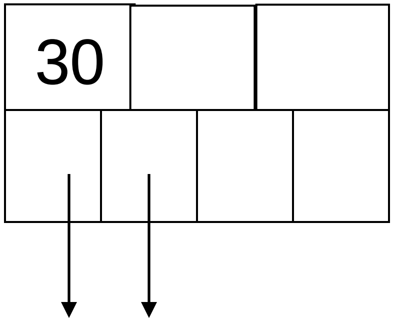
Leaf:





Non-leaf:





Size of nodes:

- n+1 pointers
- n keys

(fixed)

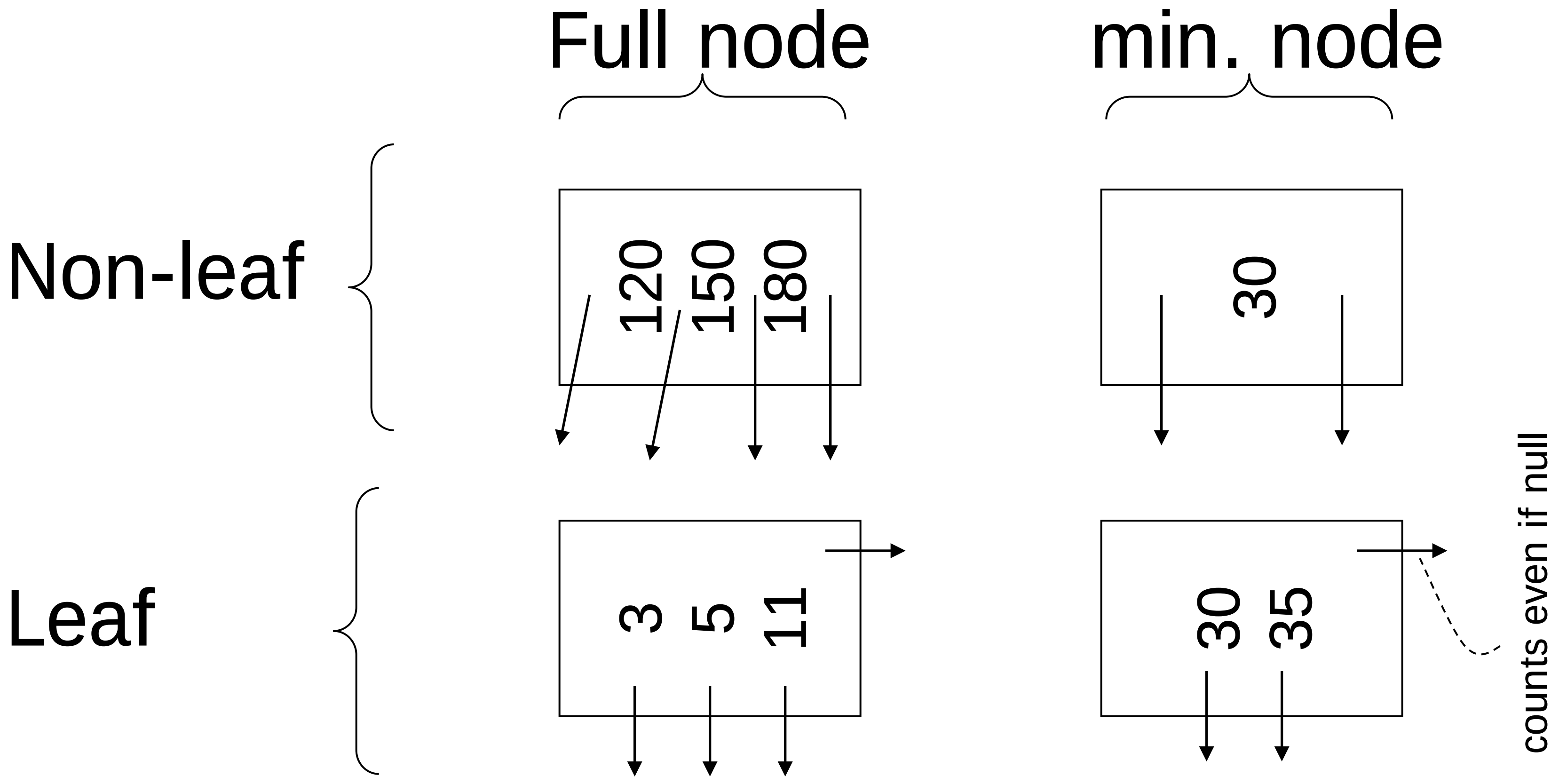
Don't want nodes to be too empty

- 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 *n*

(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	1	1

Insert into B+tree

(a) simple case
- space available in leaf

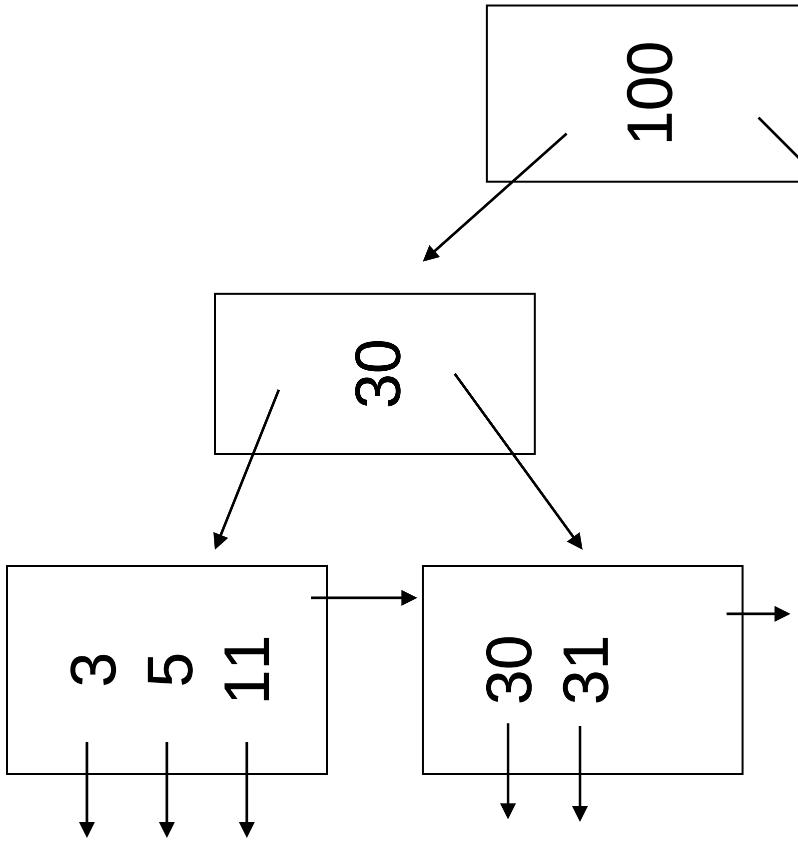
(b) leaf overflow

(c) non-leaf overflow

(d) new root

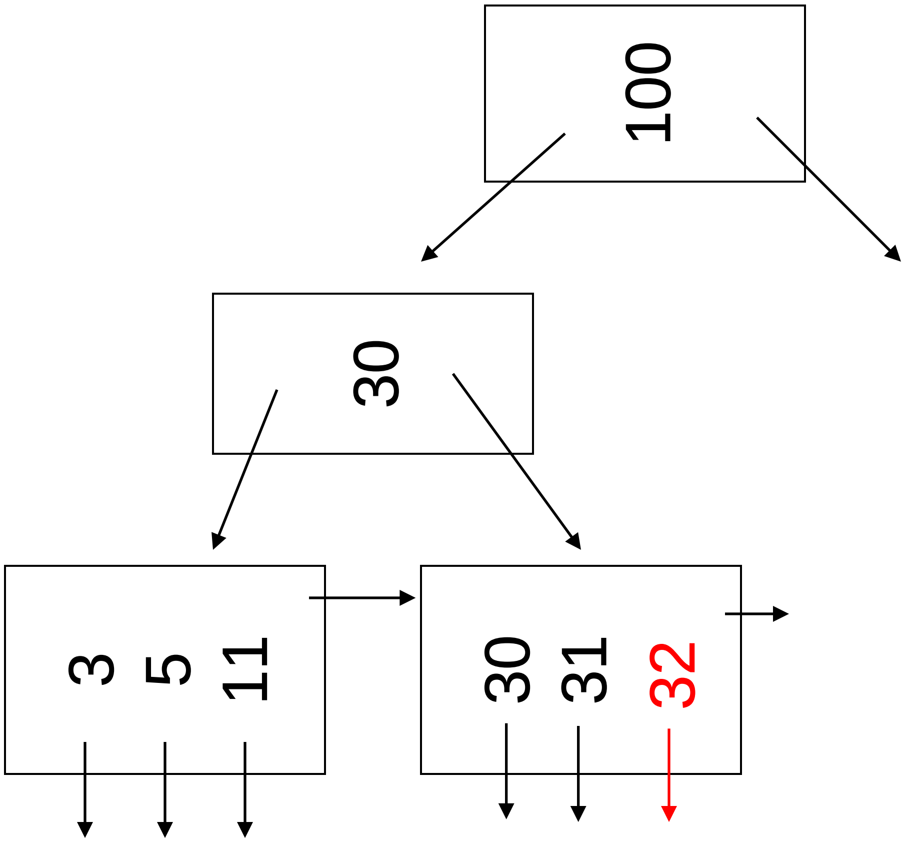
(a) Insert key = 32

n=3



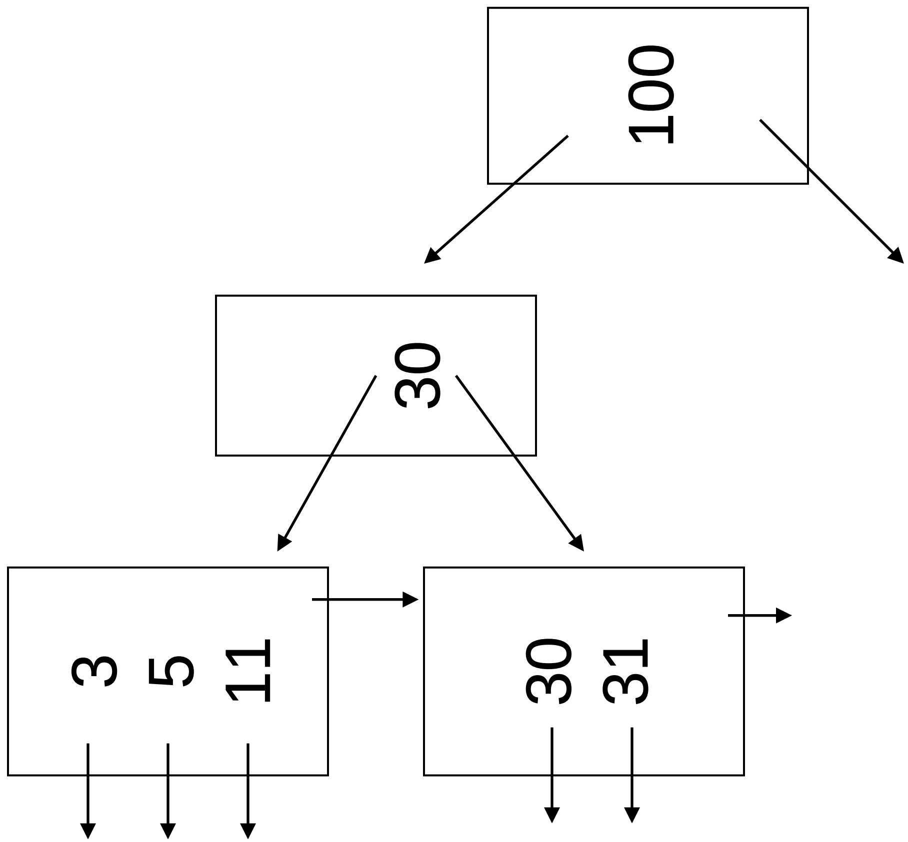
(a) Insert key = 32

n=3

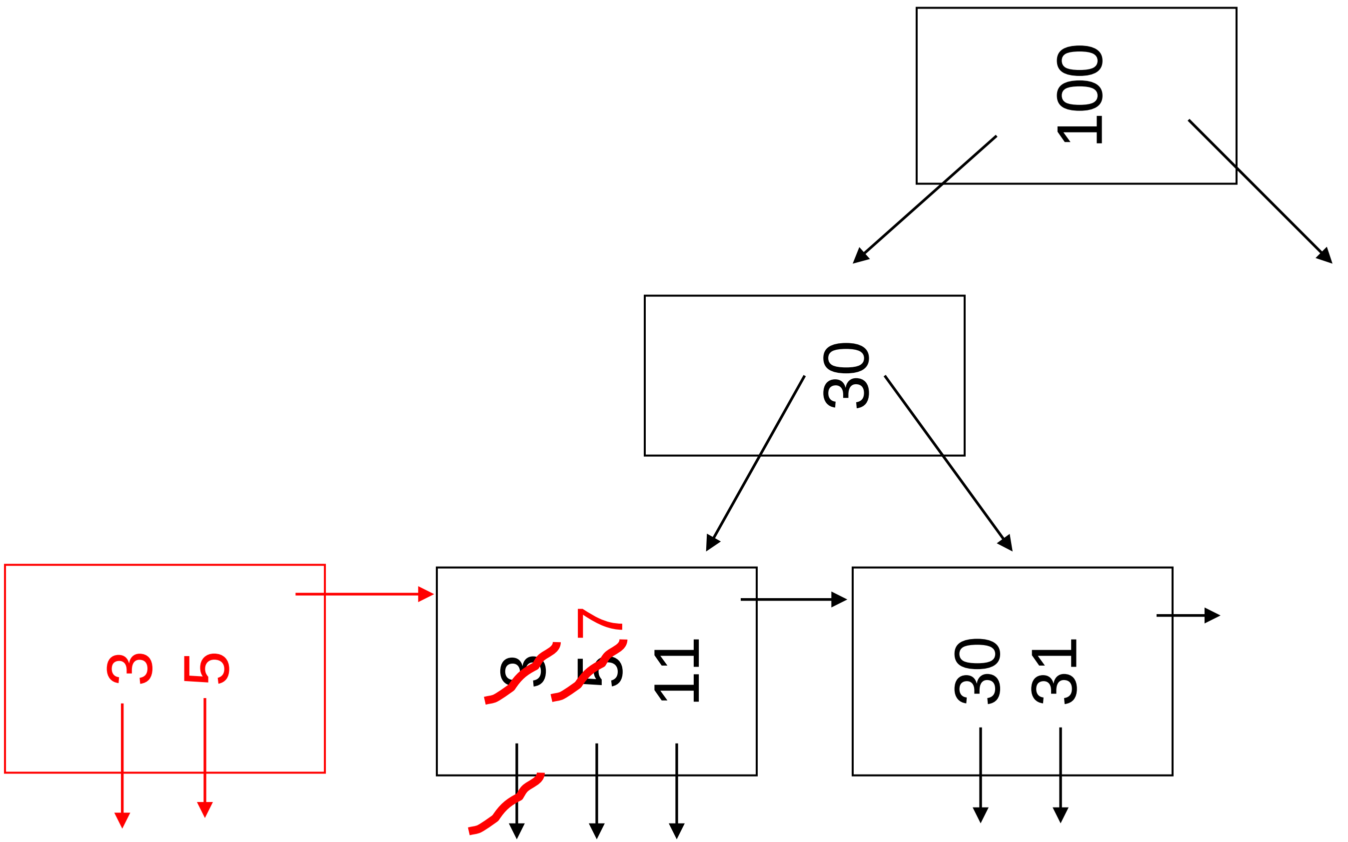


(a) Insert key = 7

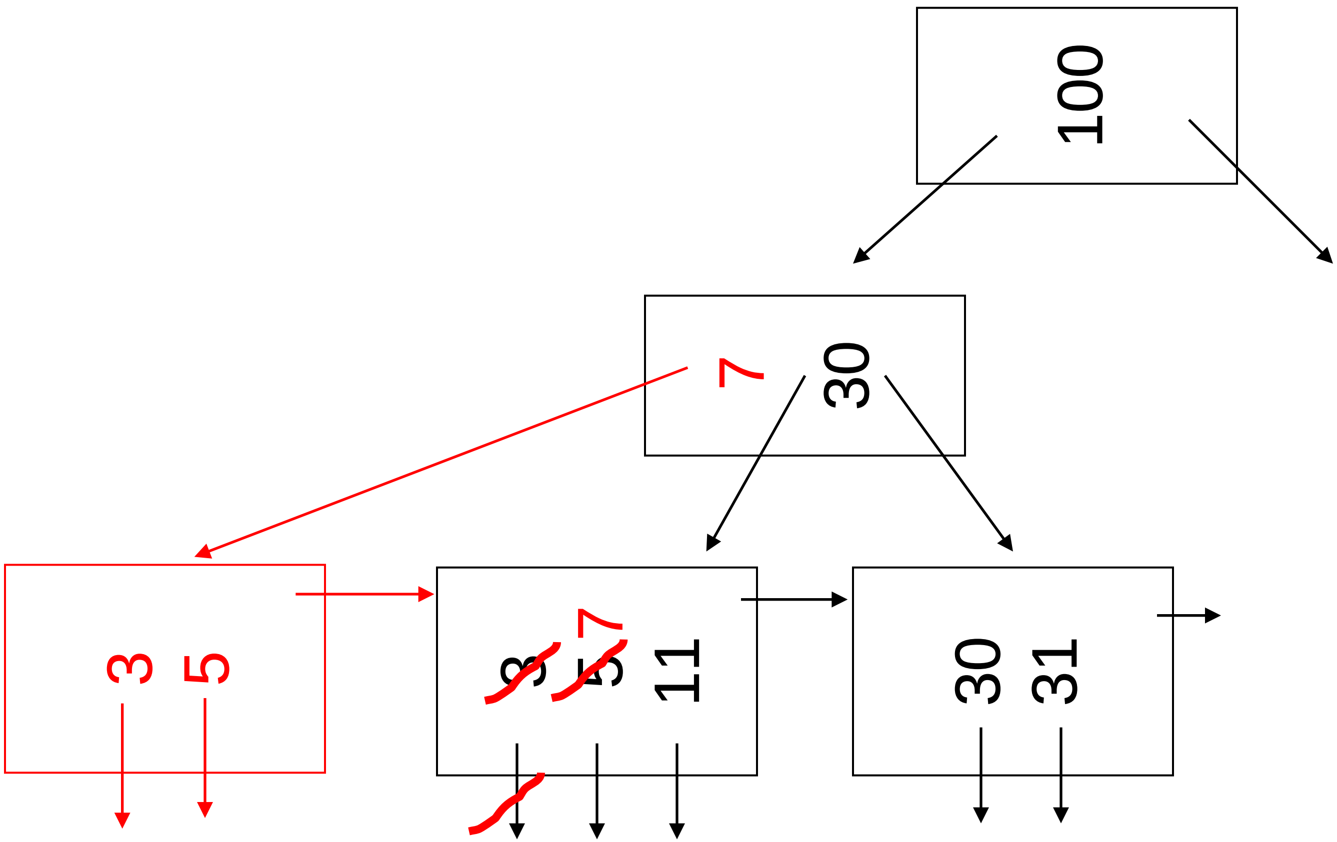
n=3



(a) Insert key = 7 n=3

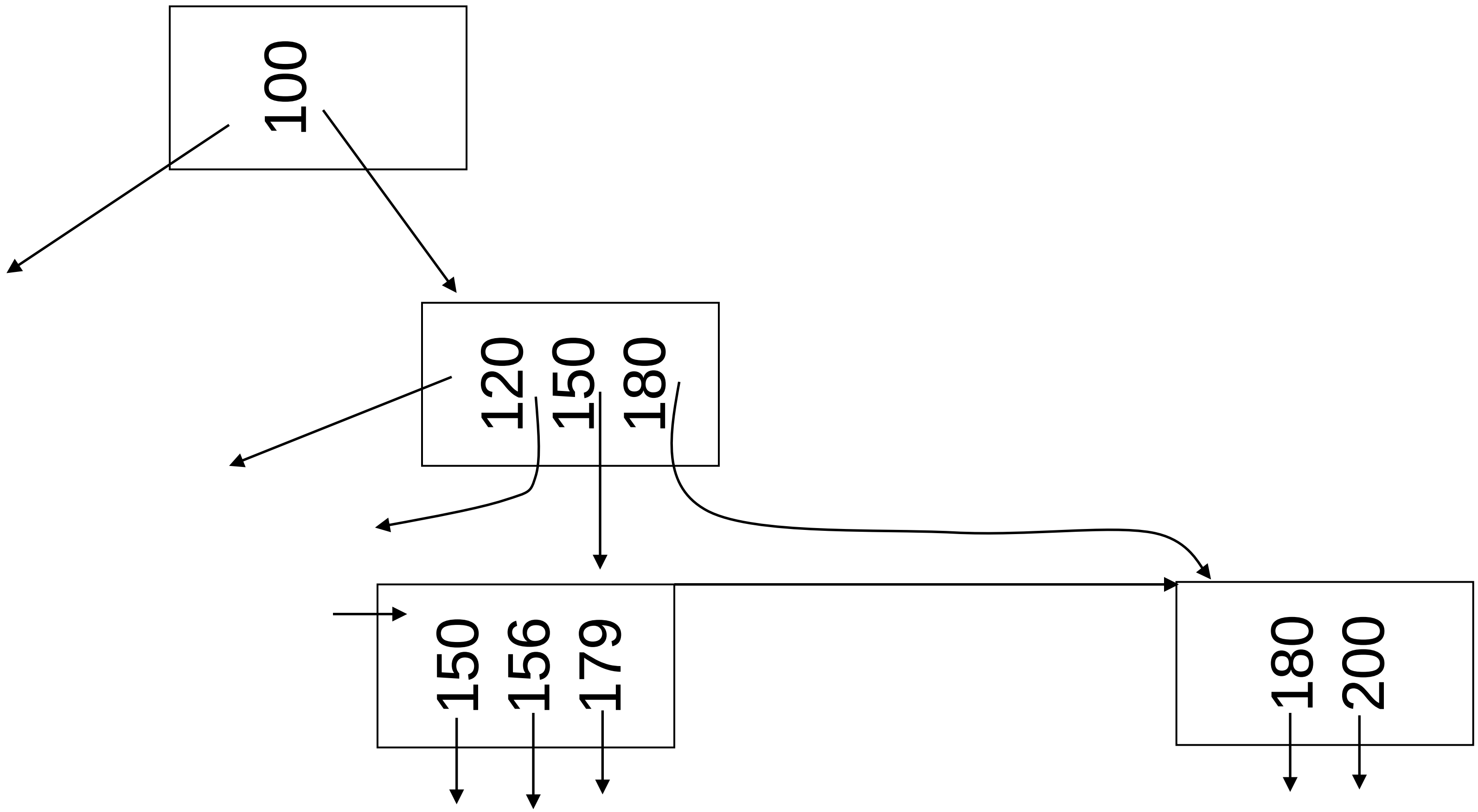


(a) Insert key = 7 n=3



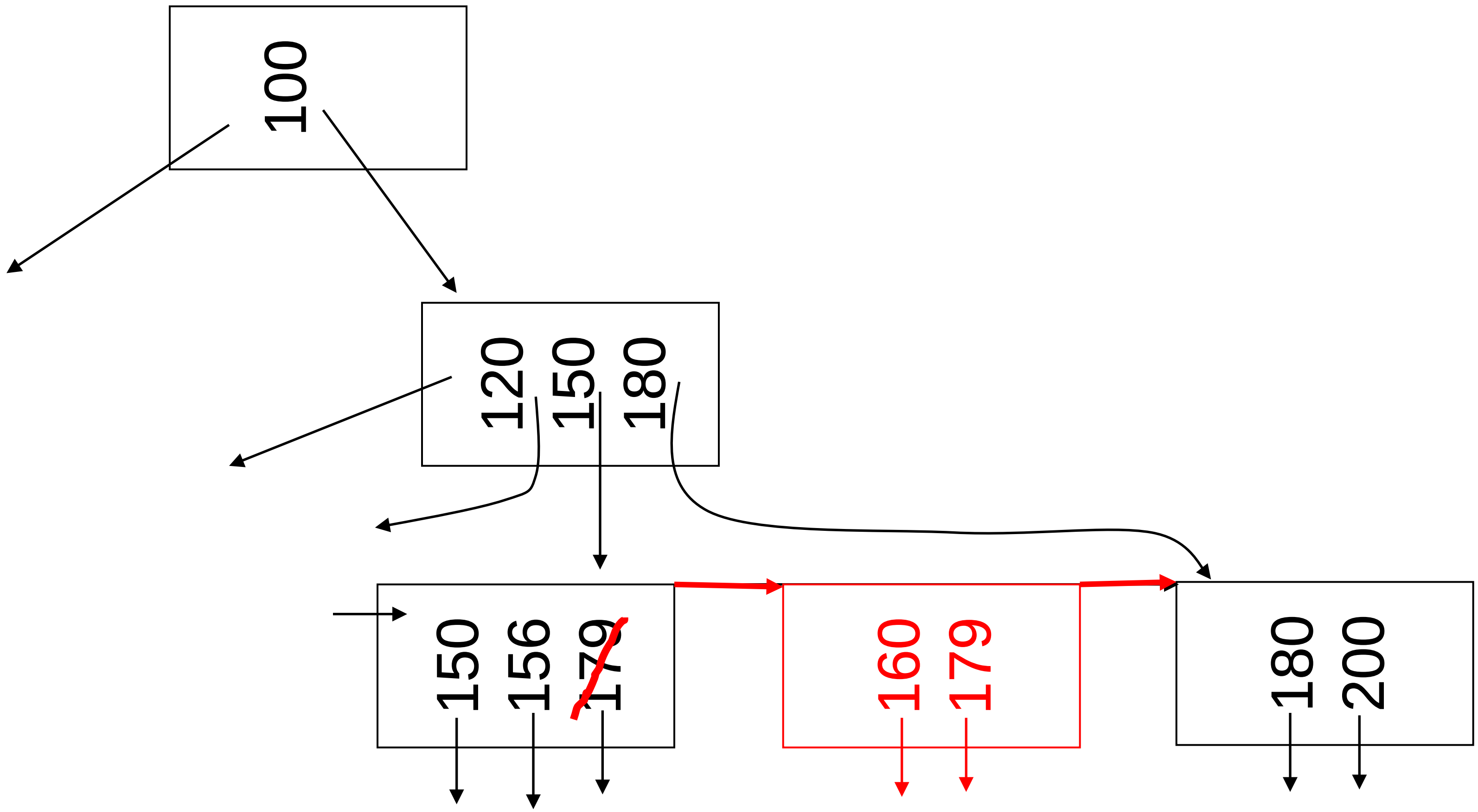
(c) Insert key = 160

n=3



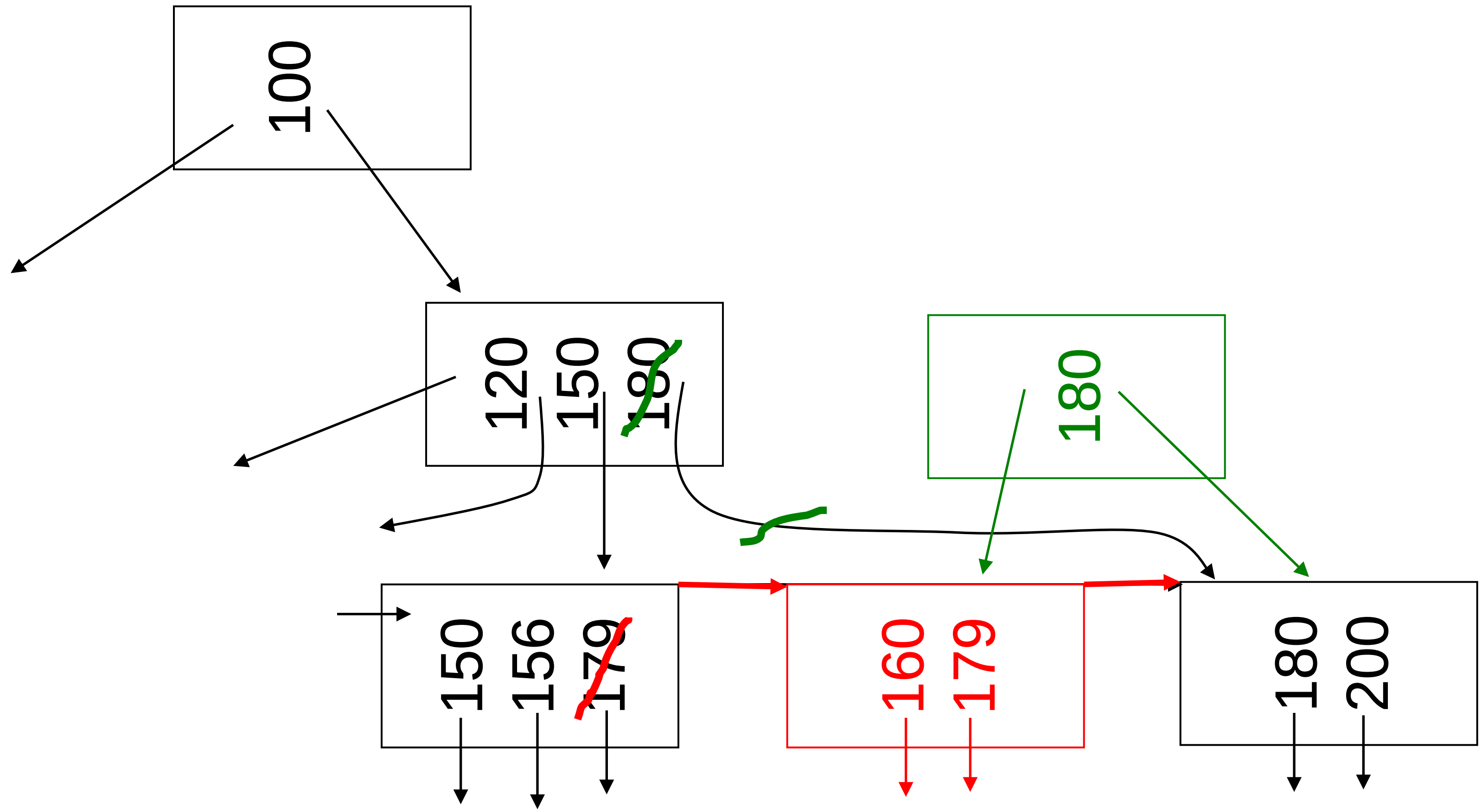
(c) Insert key = 160

n=3



(c) Insert key = 160

n=3



(c) Insert key = 160

n=3