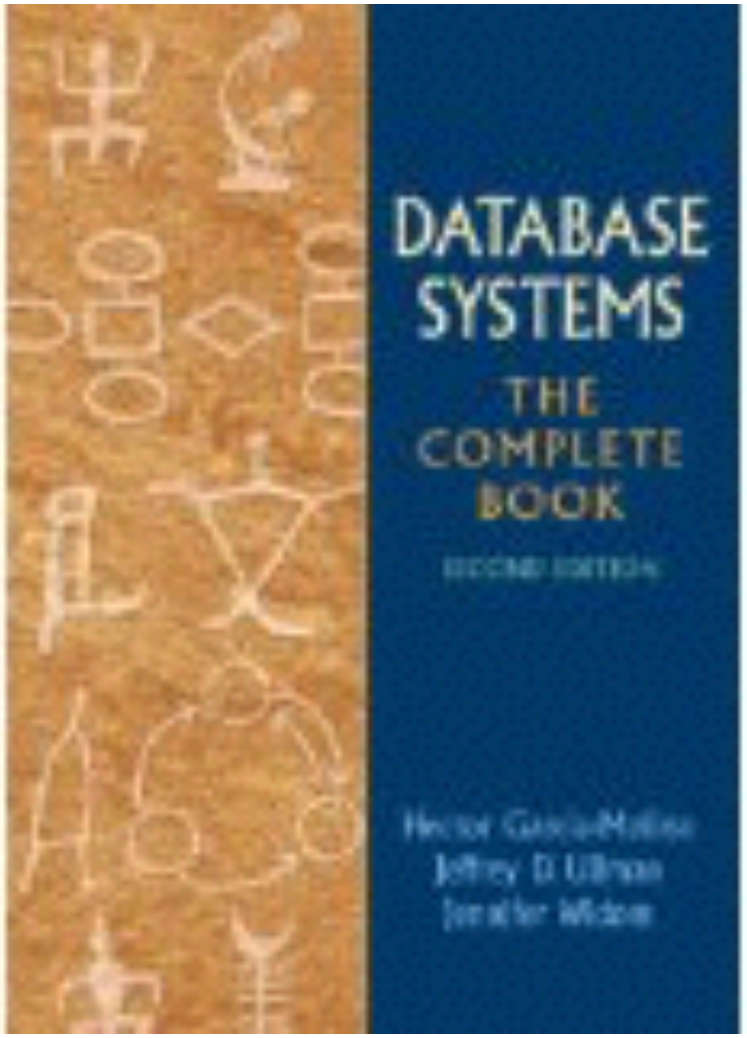
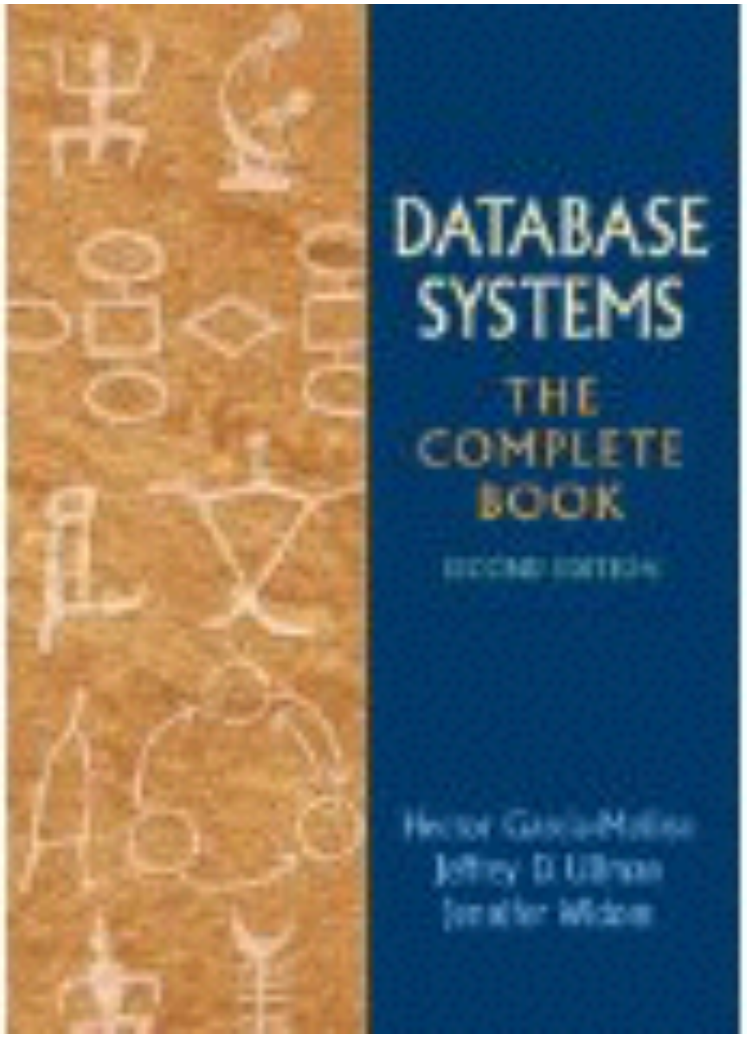
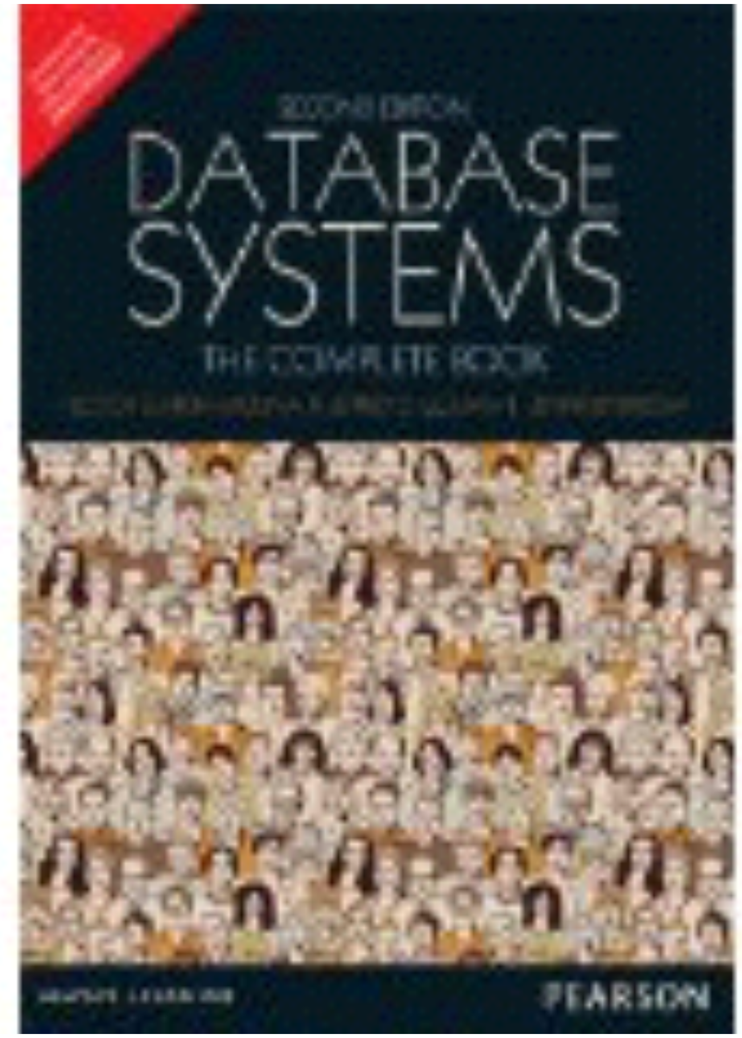
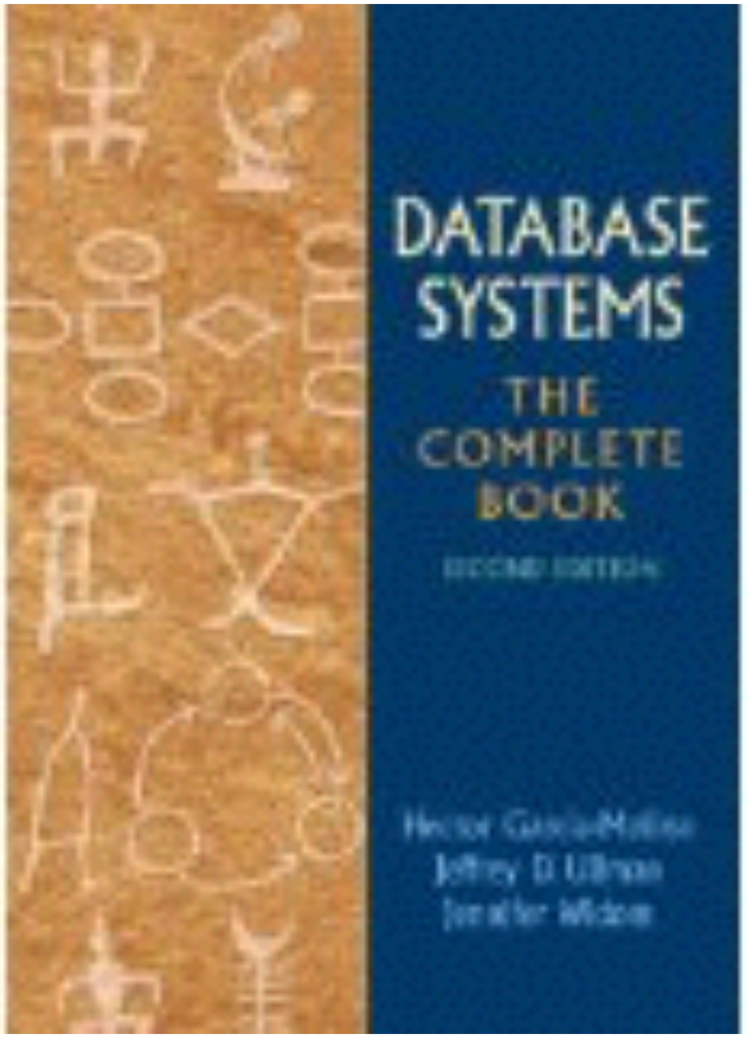
Indexes

Database Systems: The Complete Book
Ch. 13.1-13.3, 14.1-14.2

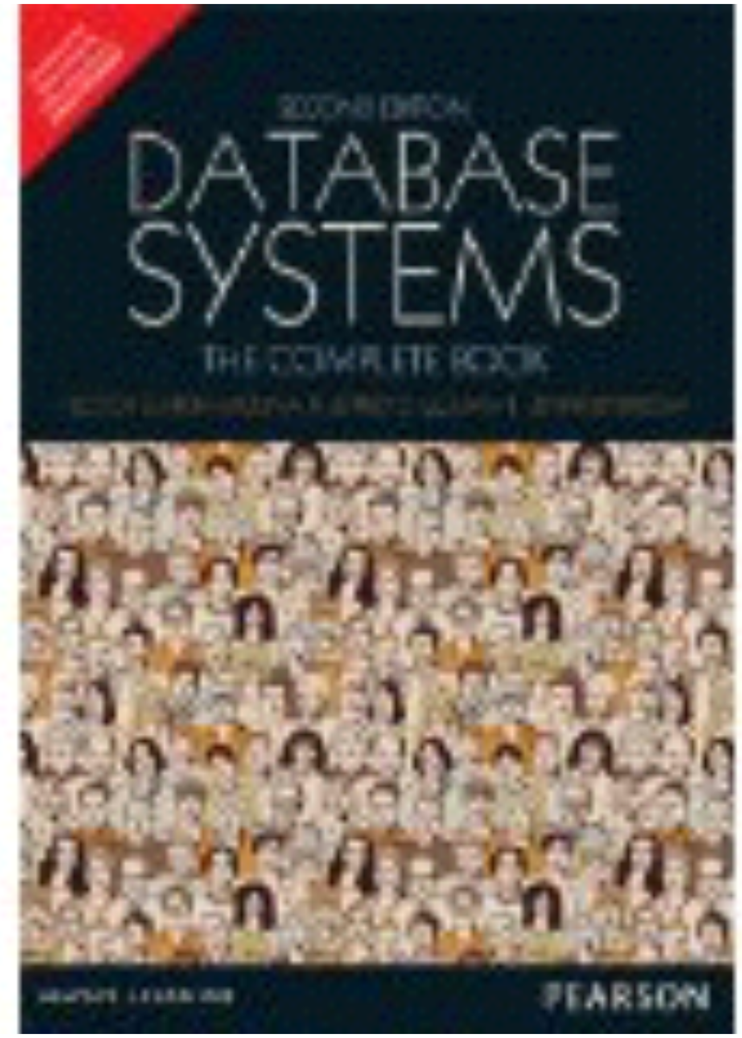

DATABASE
SYSTEMS
THE
COMPLETE
BOOK


DATABASE
SYSTEMS
THE
COMPLETE
BOOK
SECOND EDITION
Hector Garcia-Molina
Jeffrey D. Ullman
Jennifer Widom


SECOND EDITION
DATABASE
SYSTEMS
THE COMPLETE BOOK
PEARSON



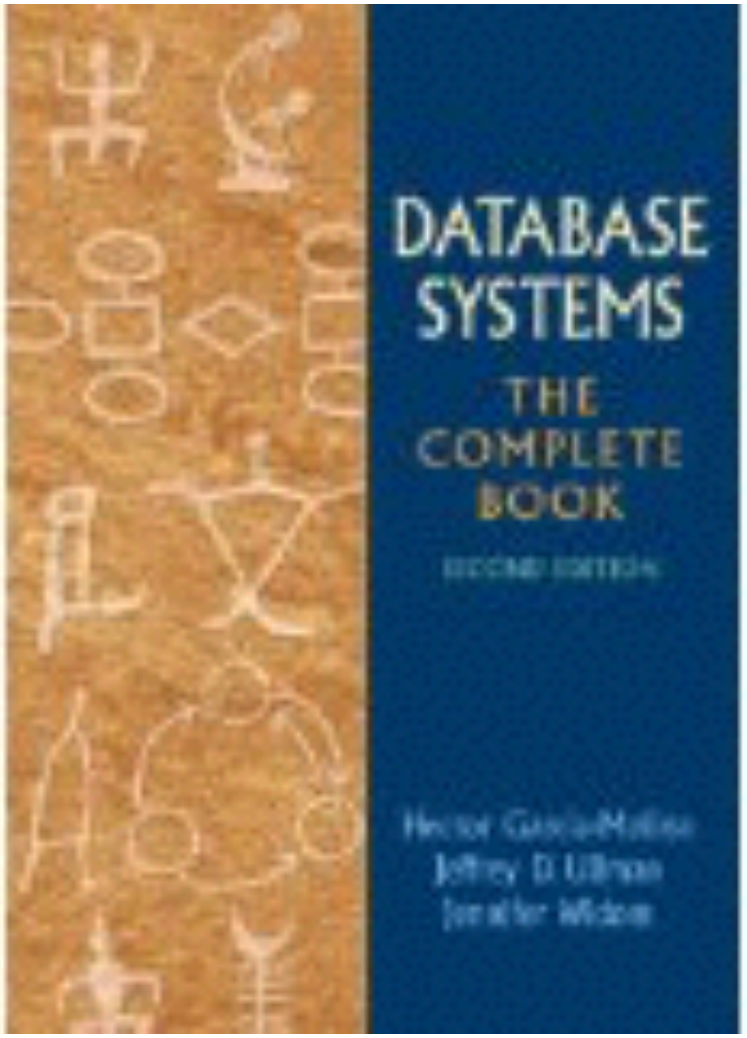
DATABASE SYSTEMS
THE COMPLETE BOOK

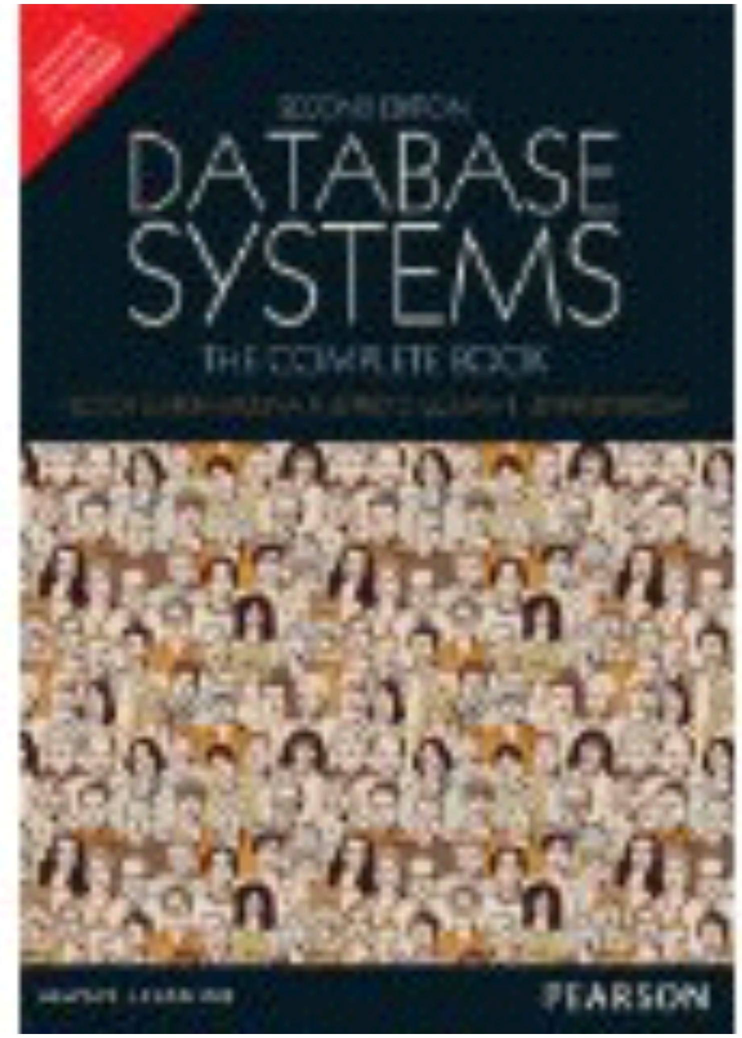


DATABASE SYSTEMS
THE COMPLETE BOOK
PEARSON

$88

$24



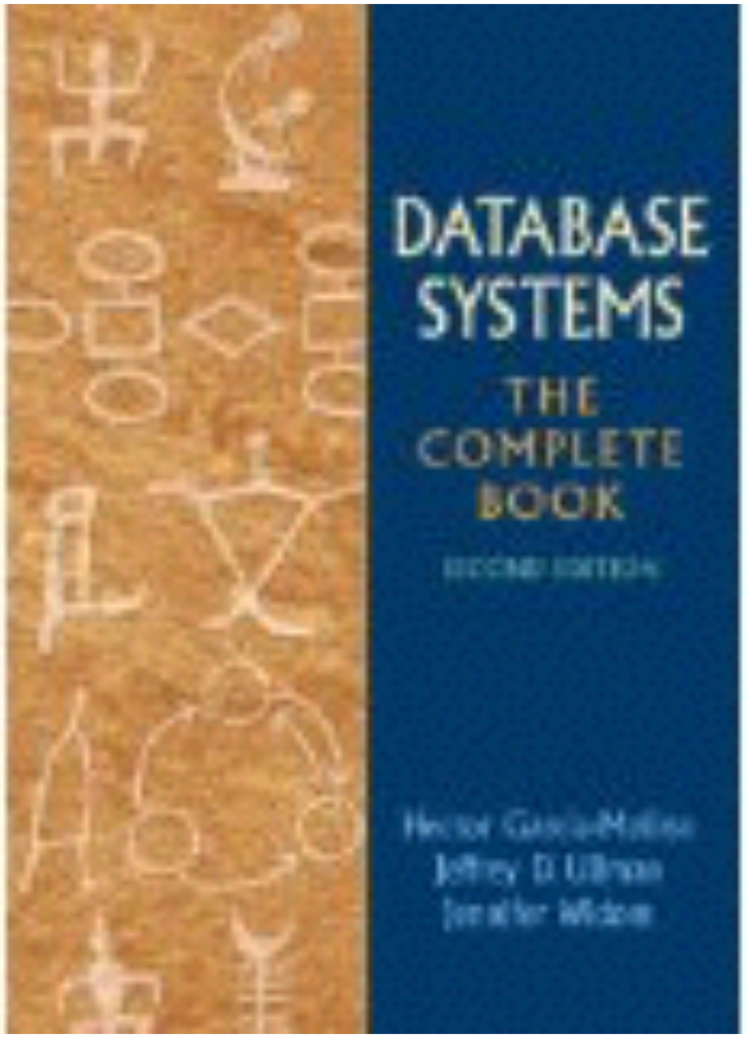


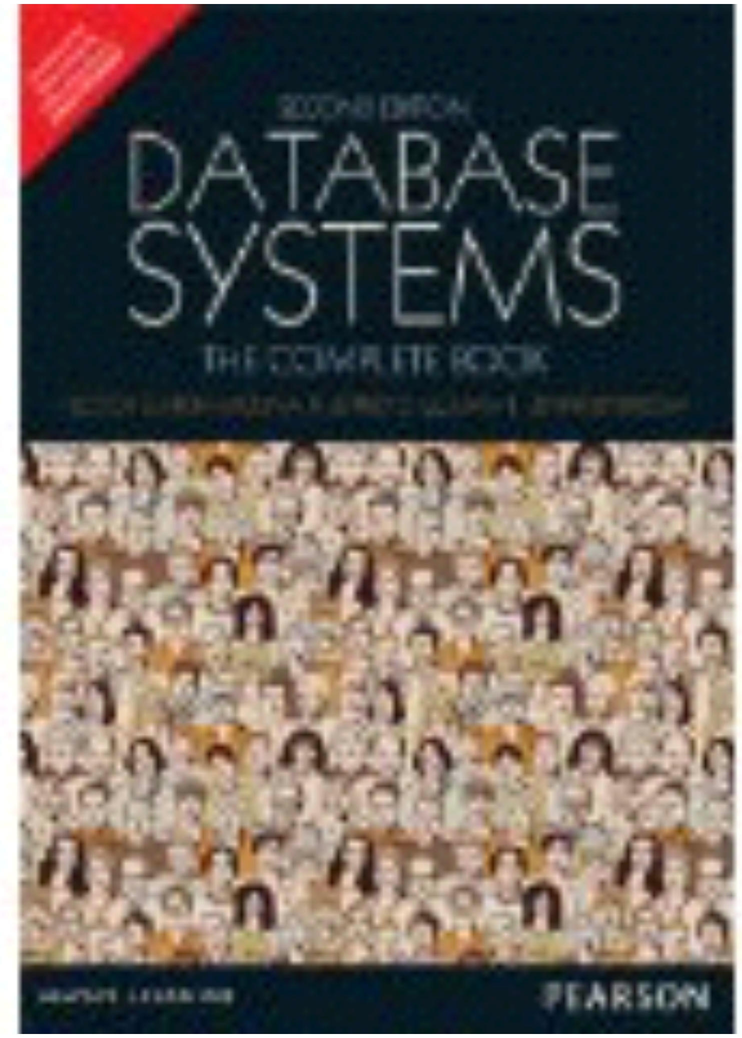
$88

Hardcover (heavy)

$24

Paperback (light)





$88

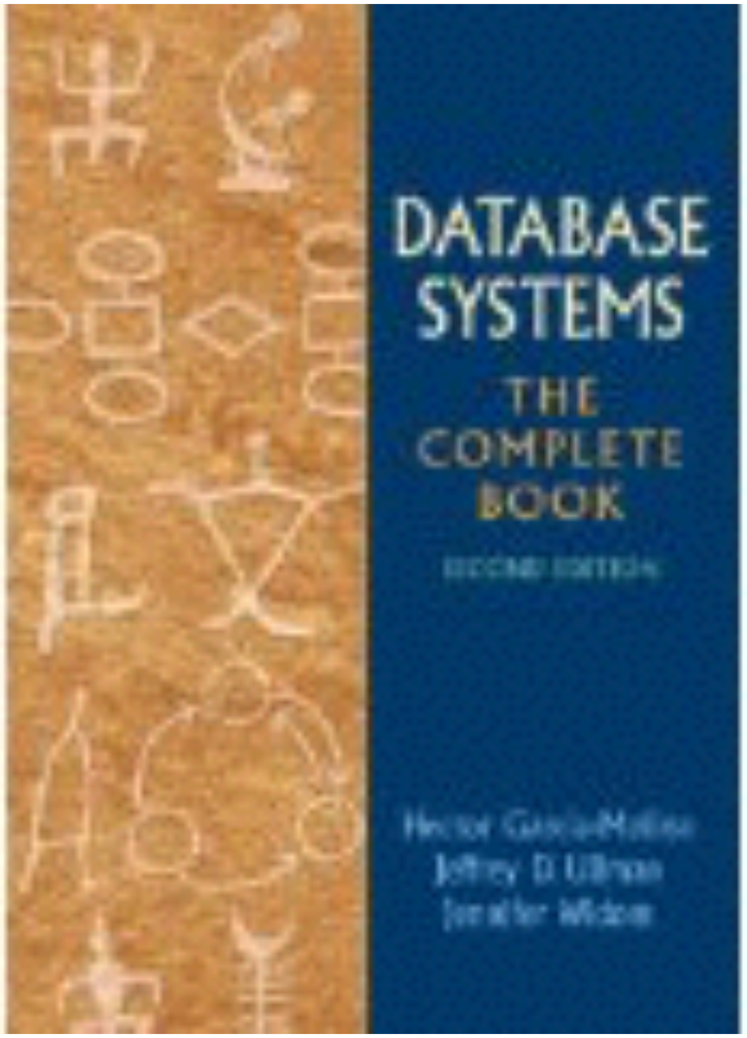
Hardcover (heavy)

Bigger

$24

Paperback (light)

Small

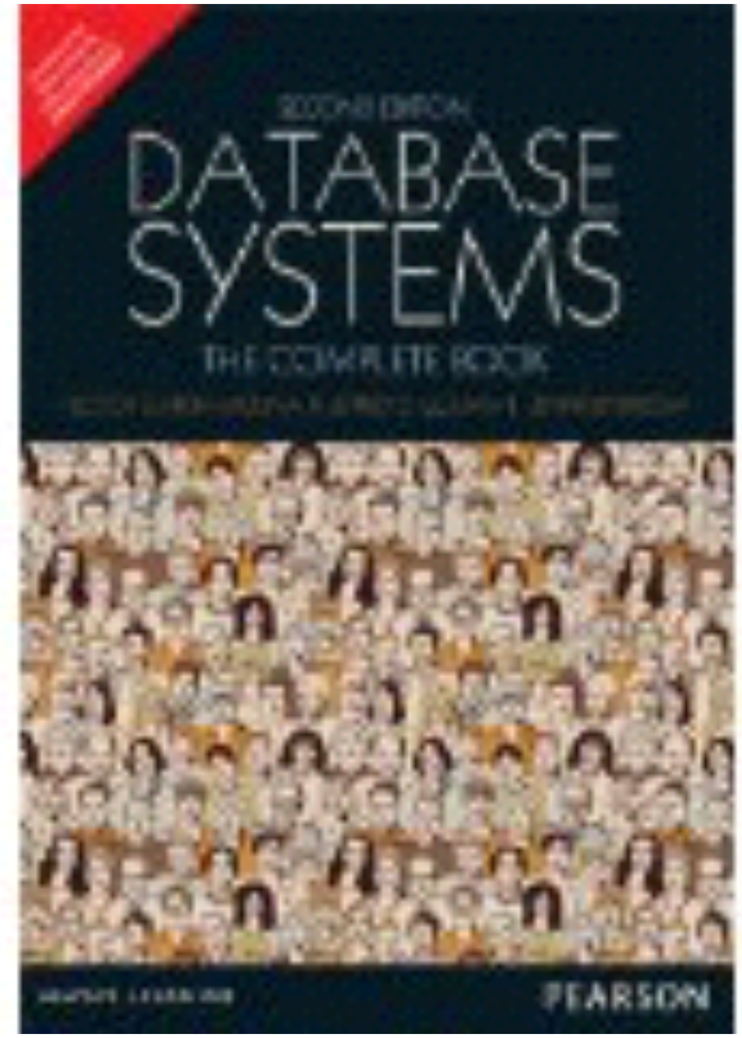


$88

Hardcover (heavy)

Bigger

Good ToC/Index



$24

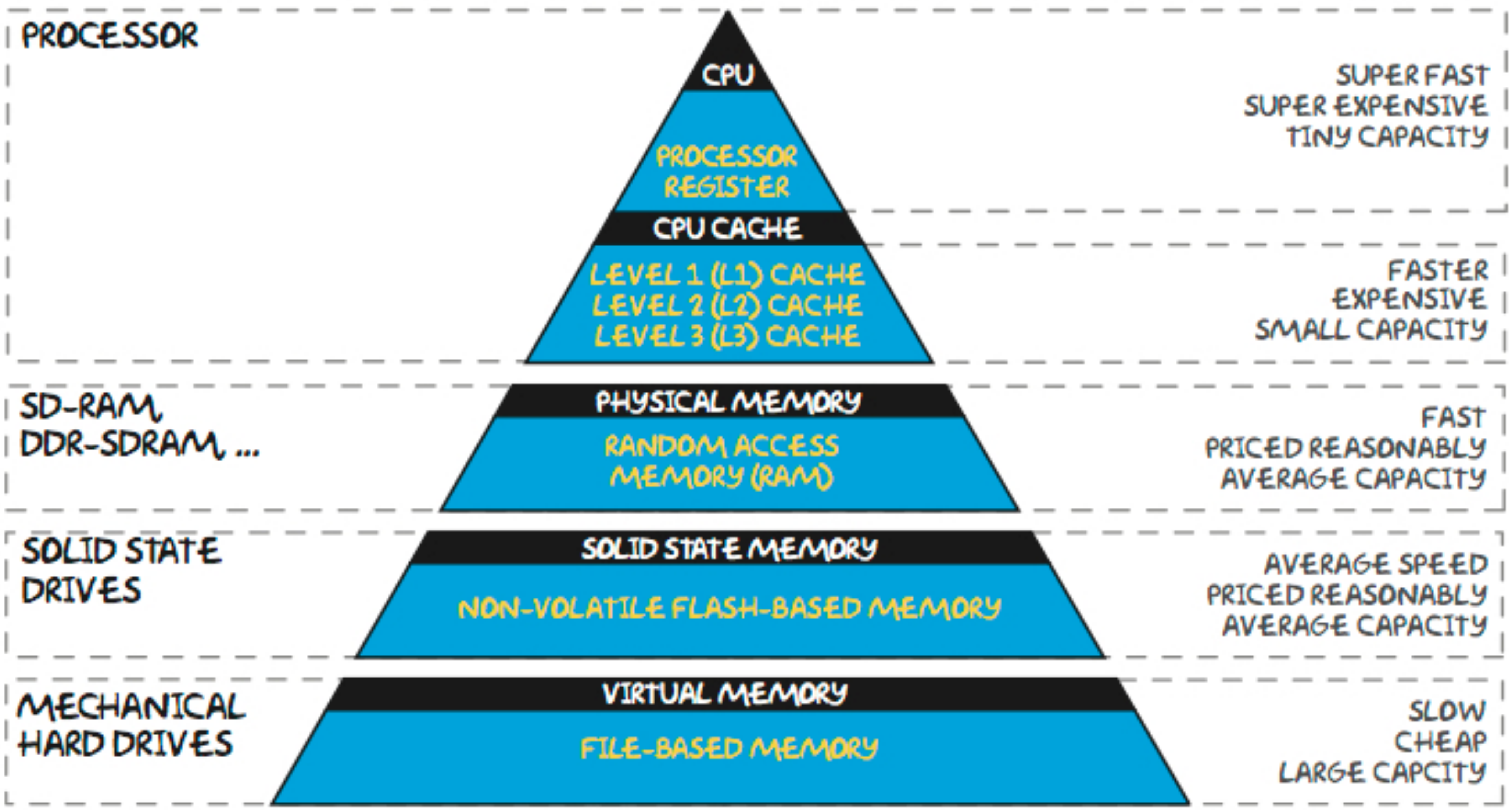
Paperback (light)

Small

Bad ToC/Index

The Memory Hierarchy

Fast (but small)



Big (but slow)

Data Organization

Heap	Clustered/ Sorted	Indexed
Records stored in any order	Records grouped together or stored in sorted order,	Secondary file used to organize data records

What are the benefits/drawbacks of each method?

Does it matter what medium the data is being stored on?

When do we use each method?

IO Operations are Bad

Recap / GroupWork

```
SELECT o.FirstName, o.LastName
FROM Officers o
WHERE o.Rank >= 3
  AND ( o.Ship = 1701
        OR o.Ship = 2000 )
```

What is an equivalent Relational Algebra expression?

What is the maximum working set size?
What is the time complexity?

Query Evaluation

- A query plan identifies the evaluation path.
- Individual operators express primitive operations.
  - Select, project, join, sort, etc…
- Individual operators can be evaluated in isolation.
  - e.g., Select: Drop rows that fail the predicate
- … but sometimes combinations of operators are better.
  - e.g., Select+Cross Product **vs** Join

Let's Consider Select…

```
SELECT o.FirstName, o.LastName
FROM Officers o
WHERE o.Rank >= 3
  AND ( o.Ship = 1701
        OR o.Ship = 2000 )
```

How would you evaluate this query?

How would you organize the data for this query?

Problem

Select searches for data

Checking every data value is *correct*, but not *efficient*

Solution

Organize the data!

What are some ways of organizing the data?

Organizing the Data

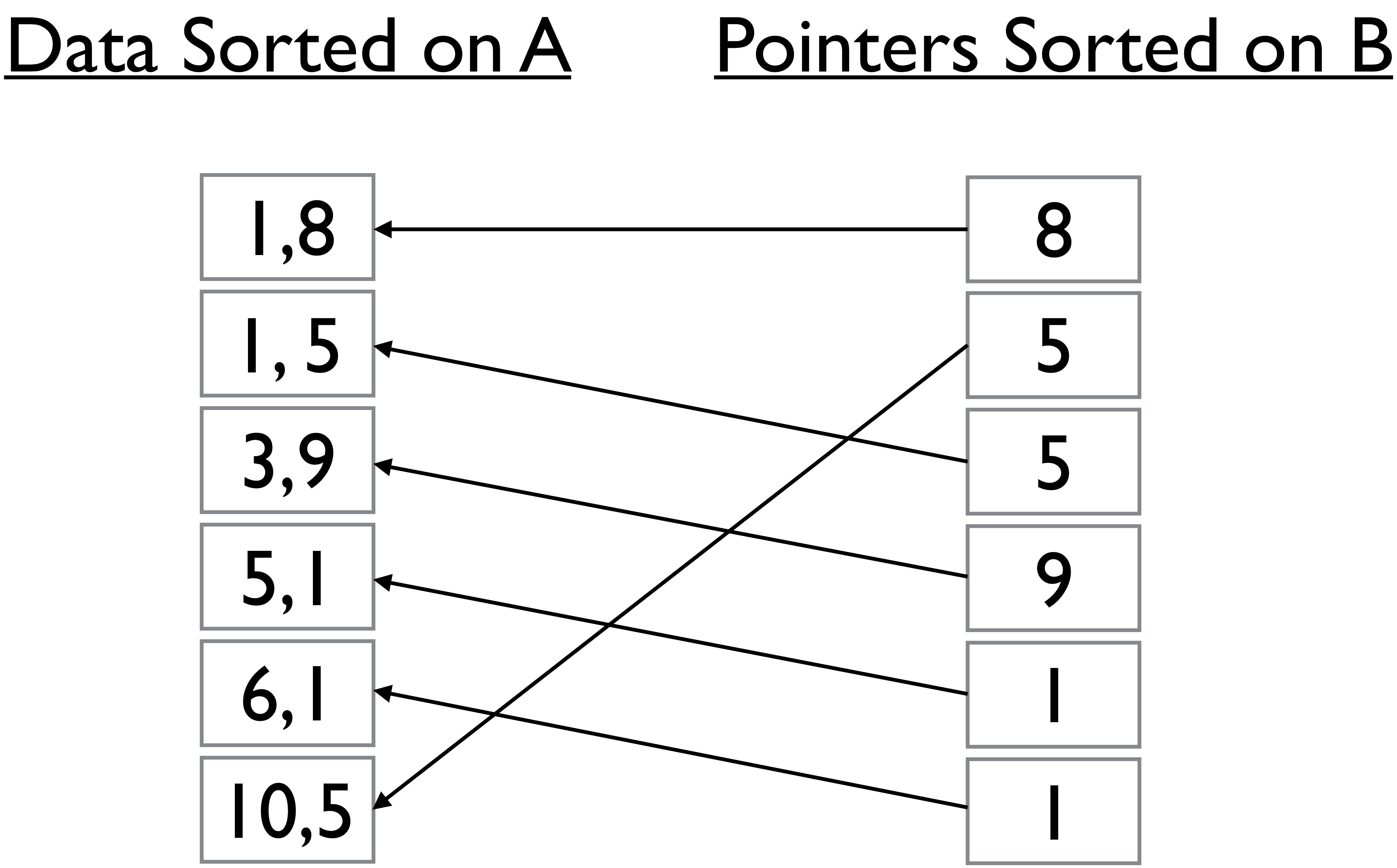
- Solution 1: Sort
  - Store the data sorted
- Solution 2: Partition (e.g., Hash)
  - Deterministically create ‘buckets’ of data.
- Solution 3: Organize References
  - Store/organize ‘pointers’ to the data.

What are some pros and cons for each solution?

Indexing (high level)

<u>**A, B**</u>
10,5
5,1
3,9
1, 5
1,8
6,1

Indexing (high level)



Want Efficient Lookups on Both A and B!

Back to Select

How would you sort your data for…
(and how would you evaluate it)

$$\sigma_{A = 1}$$

$$\sigma_{A < 1}$$

$$\sigma_{A = 1 \text{ AND } B = 2}$$

Data Organization

- Each clause in a CNF boolean formula must be true.
- API: Give me all records (or record IDs) that satisfy this predicate (these predicates)
  - Equality search: All records with field X = ‘Y’
    - **`Officer.Ship = ‘1701A’`**
  - Range search: All records with field X ∈ [Y, Z]
    - **`Officer.Rank ∈ [3, +∞)`**

Problem…

Let's say you have 2^{20} blocks (~4GB) of data sorted on A

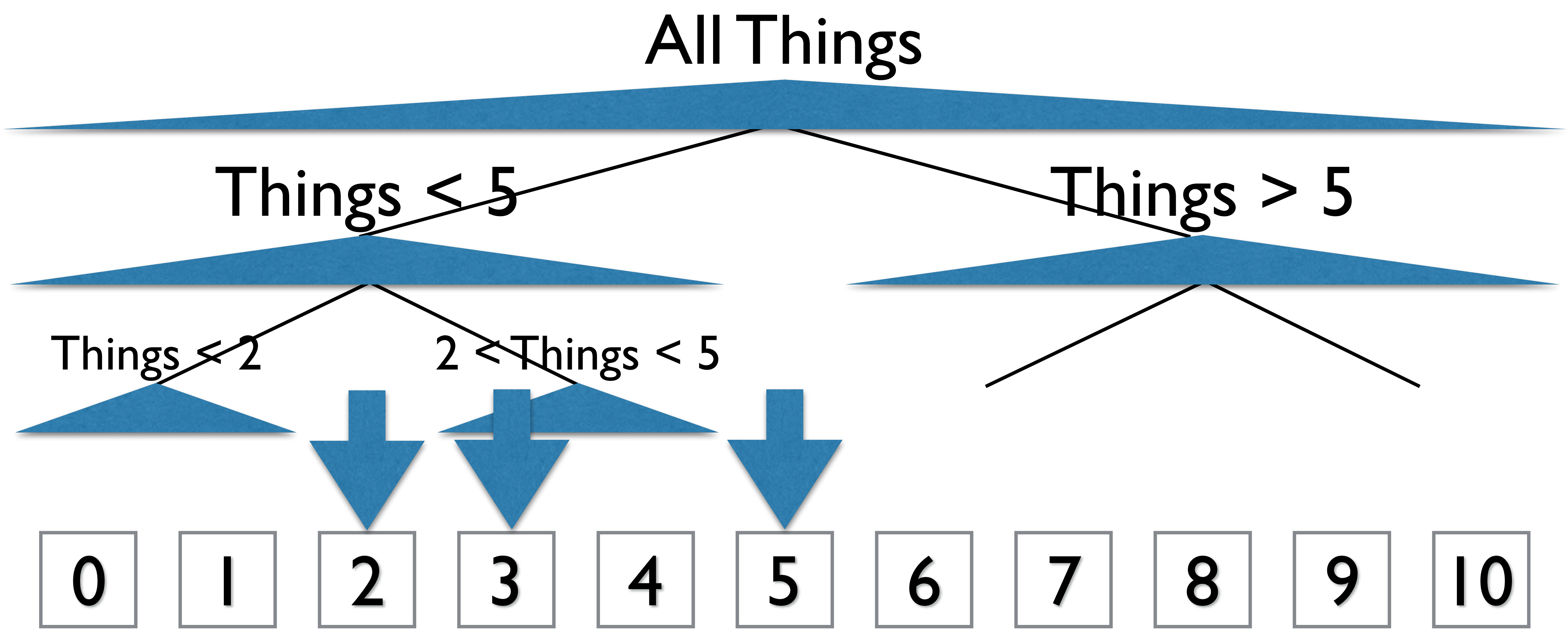
How many IOs are required to find one A?

In general, for N blocks, how many IOs?

$\log_2(N)$

Why?

“Searching”



“Find 3”

As you search, you are effectively building a binary tree.

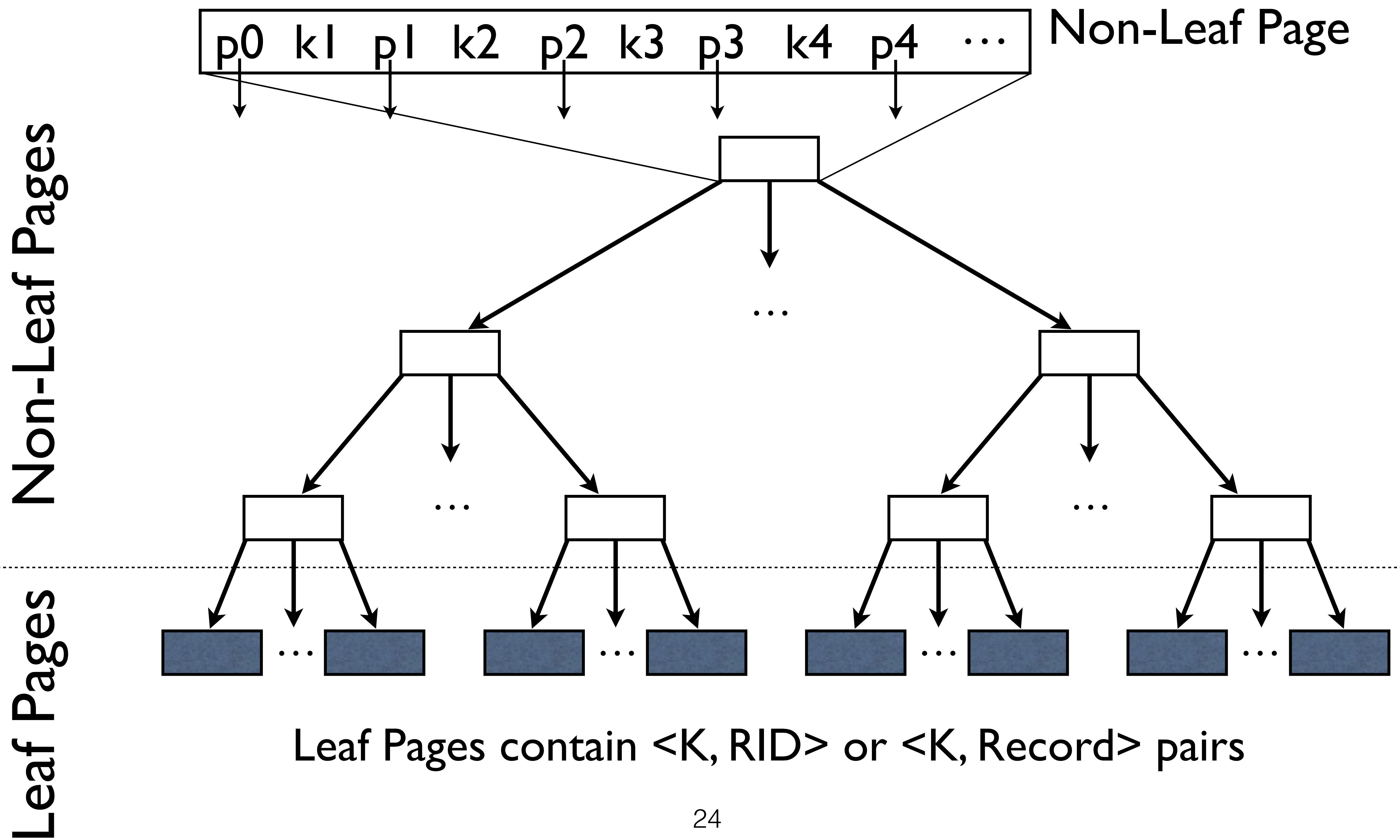
Shorter Trees

Binary Tree → Log 2 Depth

N-ary Tree → Log N Depth

Tree-Based Indexes

The ISAM Datastructure



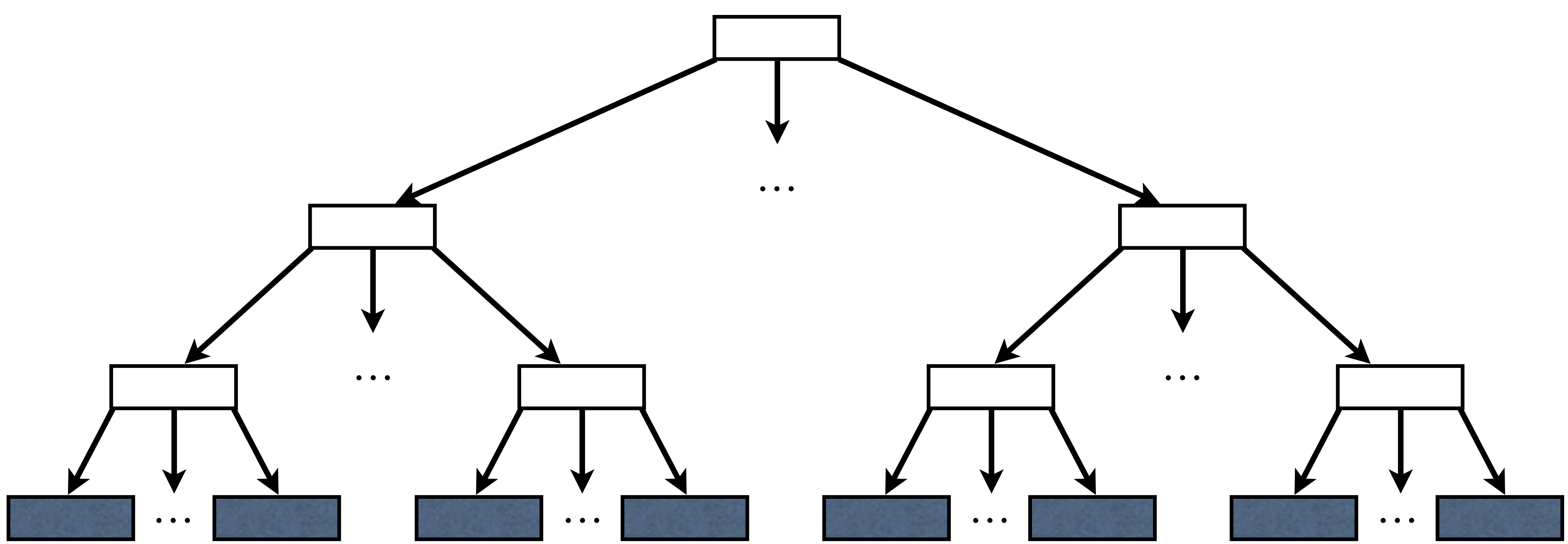
Leaf Pages contain <K, RID> or <K, Record> pairs

Constructing an ISAM Index

1) Allocate (sequential) leaf pages

2) Ensure that the data on the leaf pages is sorted

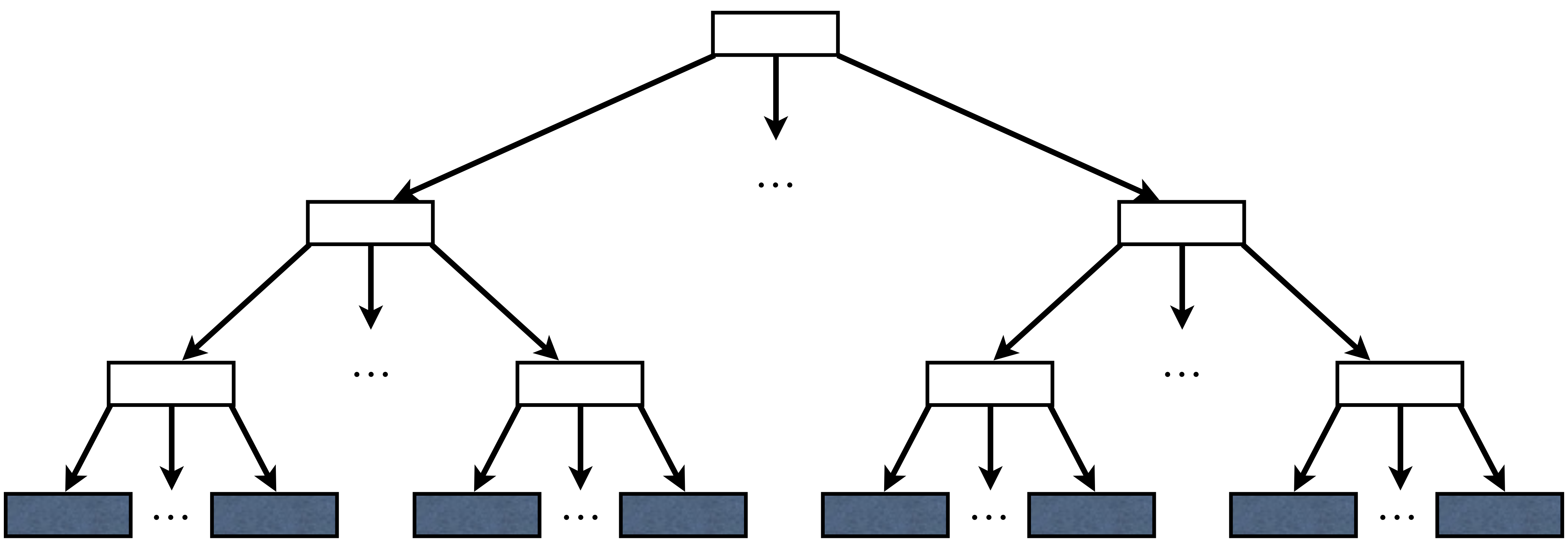
3) Build the non-leaf pages (in arbitrary order)



ISAM Index Searches

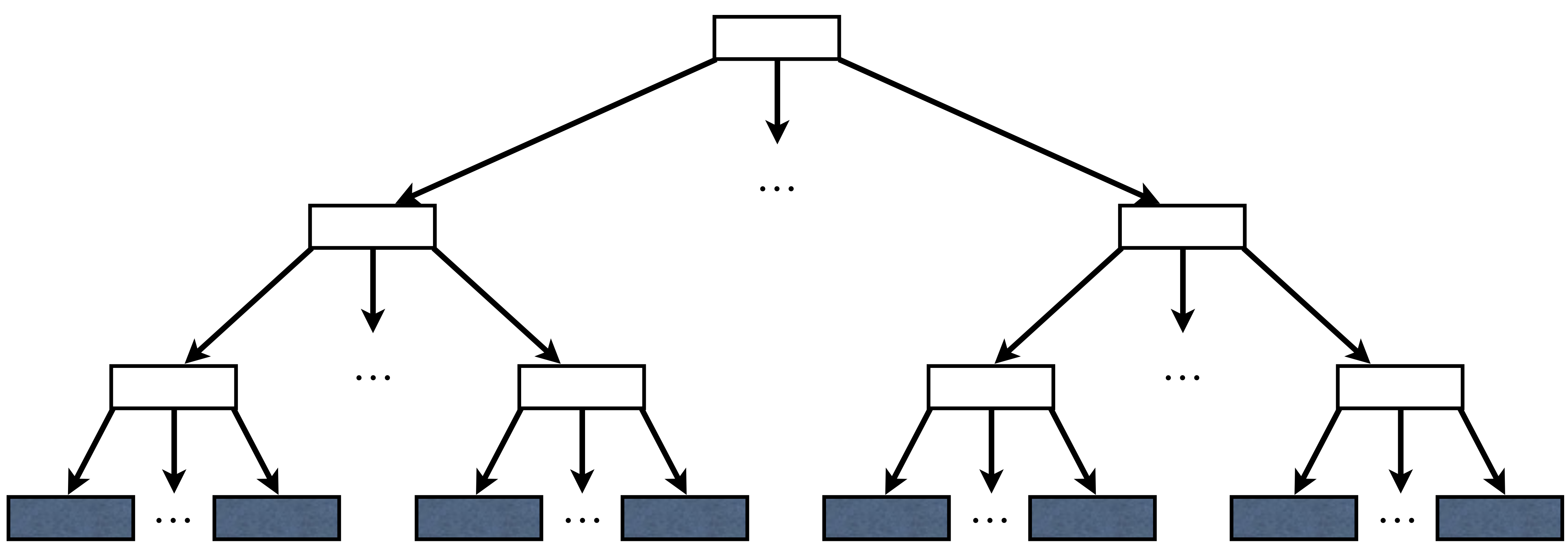
Equality: Start at root, use key comparisons to find leaf

Range: Use key comparisons to find start and end page
Scan all pages in between start/end leaves.



Constructing an ISAM Index

Do you see any problems with this?

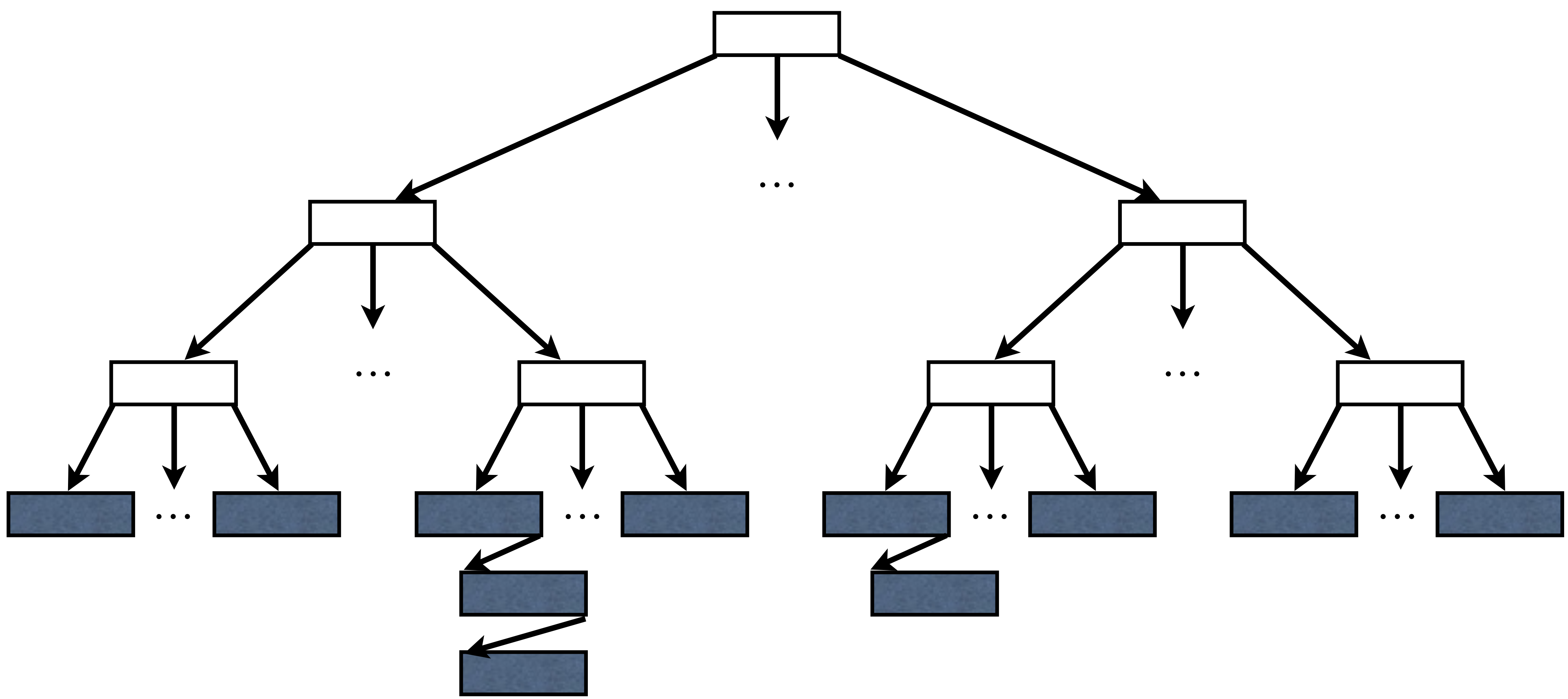


Updating an ISAM Index

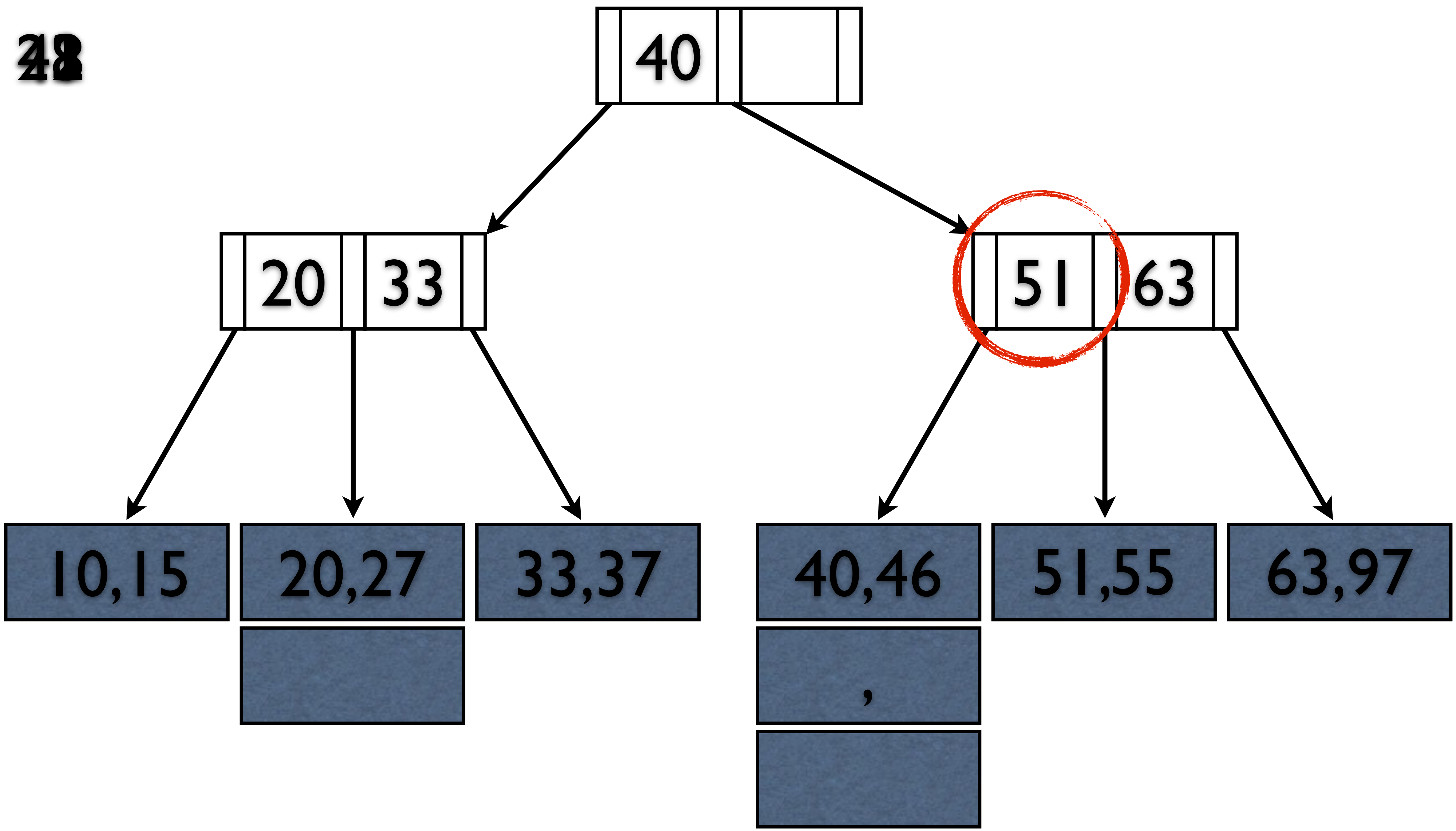
1) When creating the index leave free space in each leaf page

2) The index stays the same, new data is added to the free space

3) If a leaf page overflows, we create an overflow page (or more)



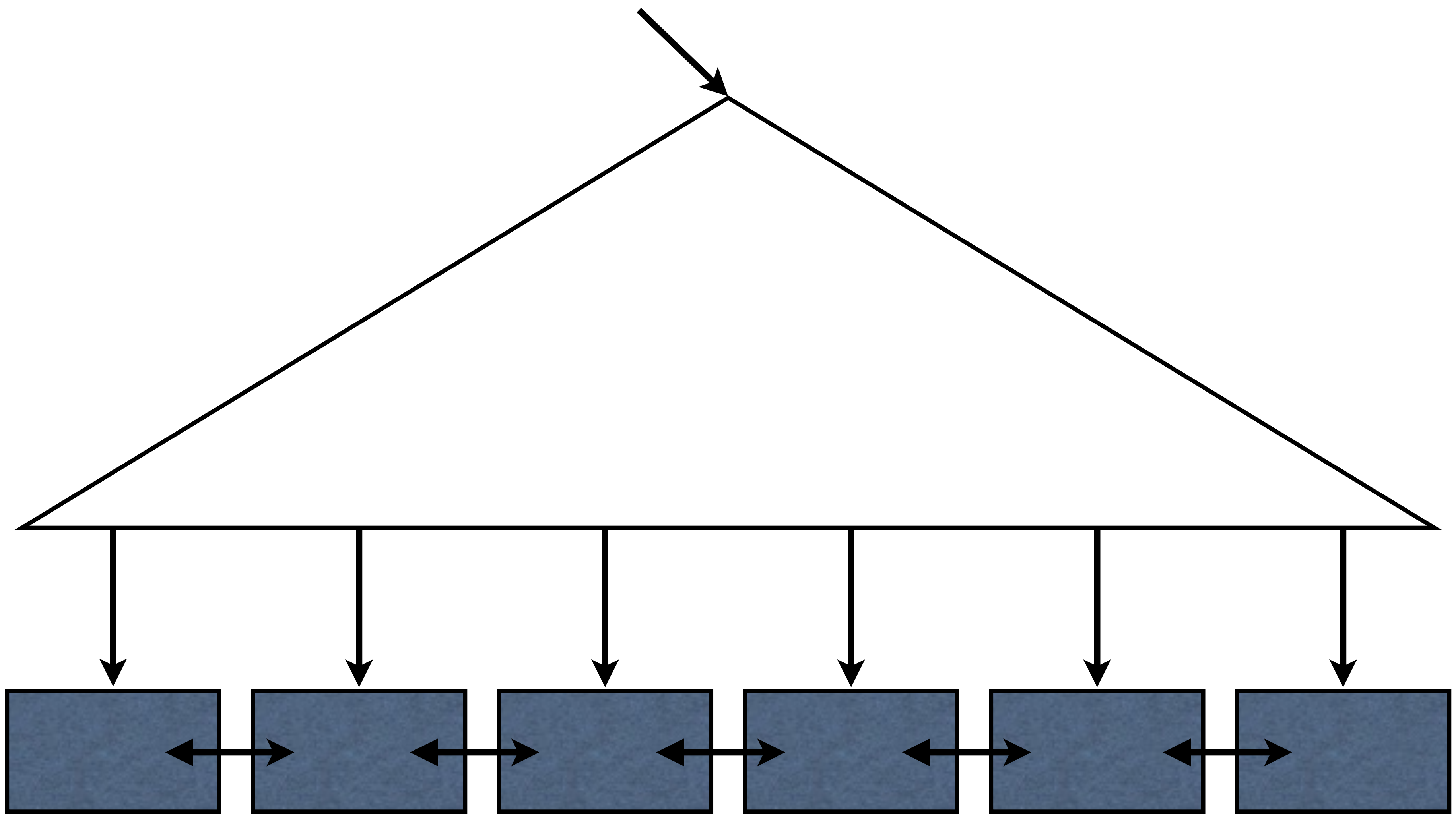
An Example ISAM



B+ Trees

Index Entries

Data Entries

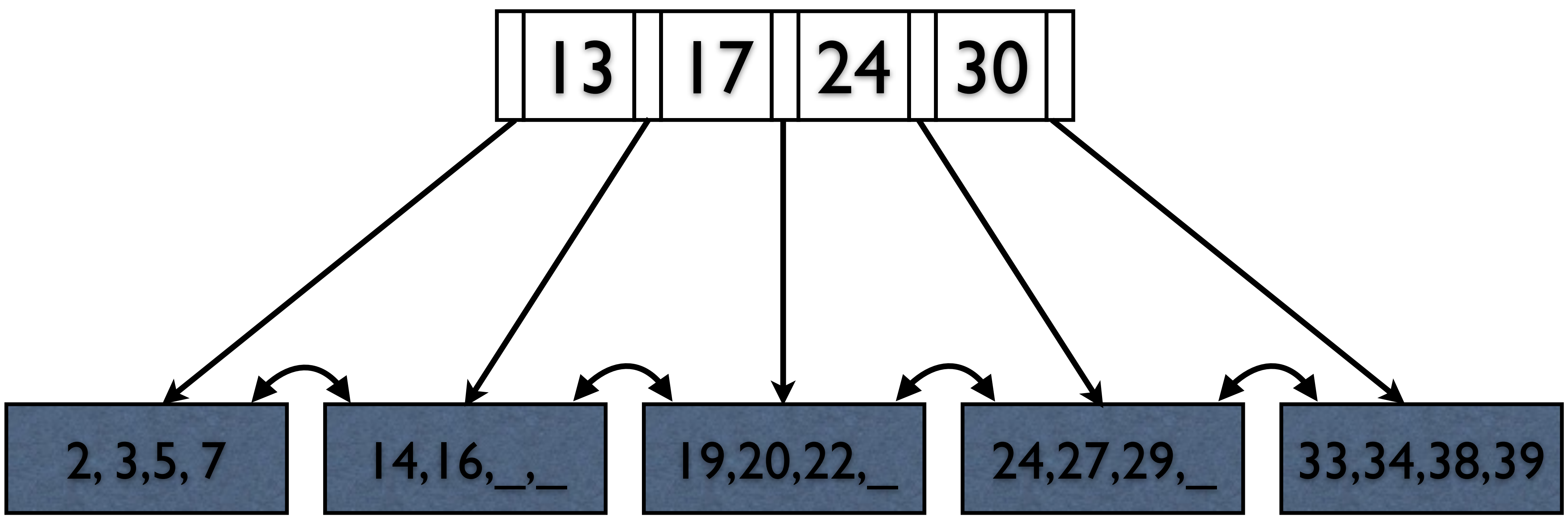


Data pages not sequential - Need linked list for traversals

B+ Trees

Search proceeds as in ISAM via key comparisons

Find 5.   Find 15.   Find [24, ∞)



B+ Tree Invariants

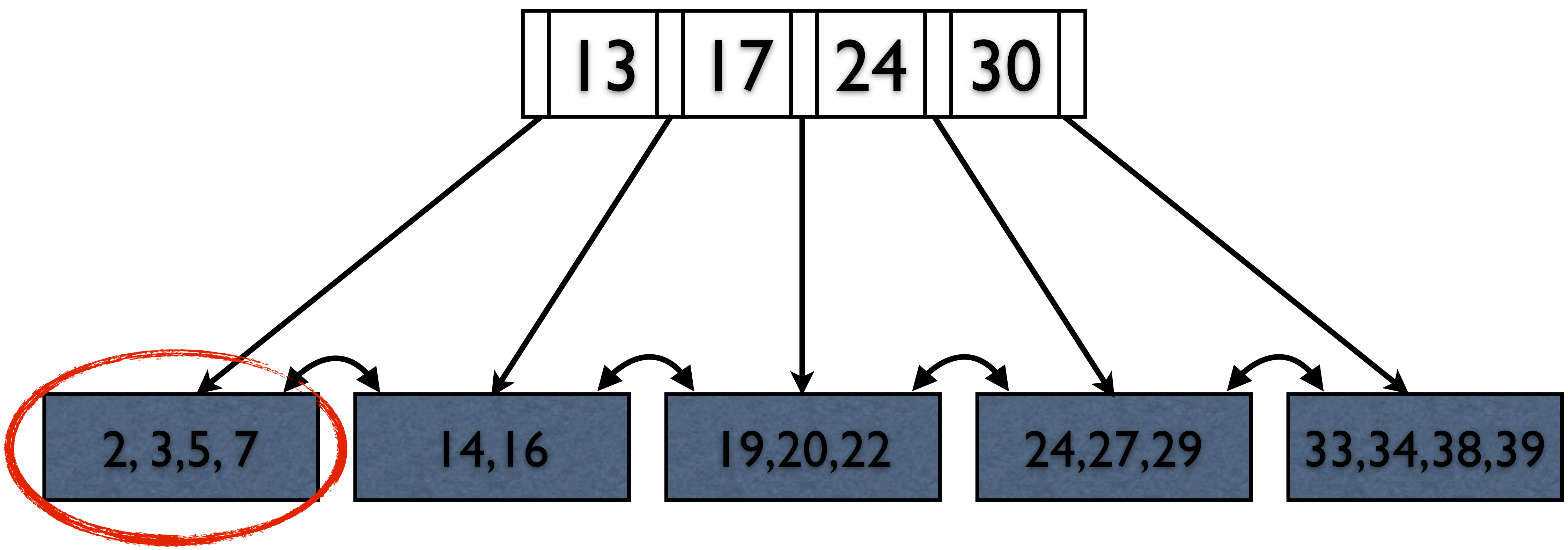
- Keep space open for insertions in inner/data nodes.
  - ‘Split’ nodes when they’re full
- Avoid under-using space
  - ‘Merge’ nodes when they’re under-filled
- **Maintain Invariant**: All Nodes $\geq$ 50% Full
  - (Exception: The Root)

Example

Inner Nodes: 4 values, 5 pointers

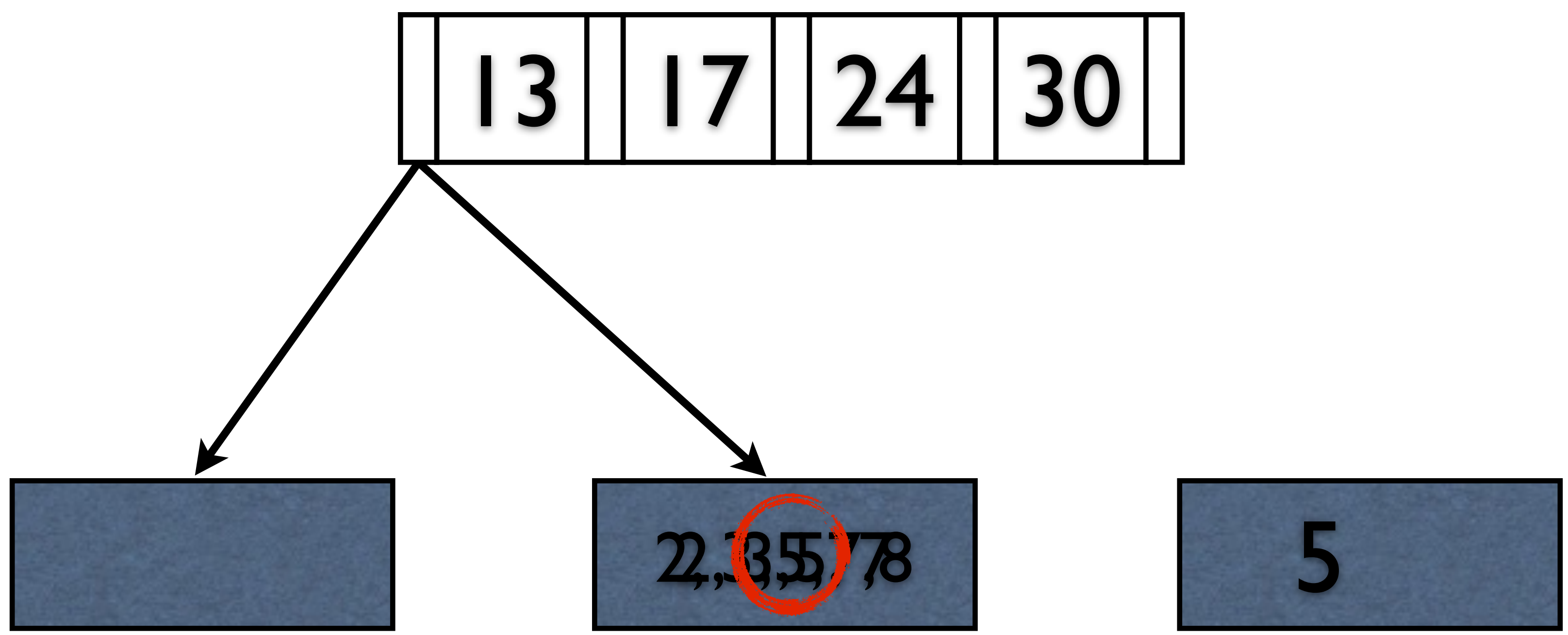
Data Nodes: 4 values

Inserting into B+ Trees



Insert 8

Inserting into B+ Trees



Copy <5> into parent index

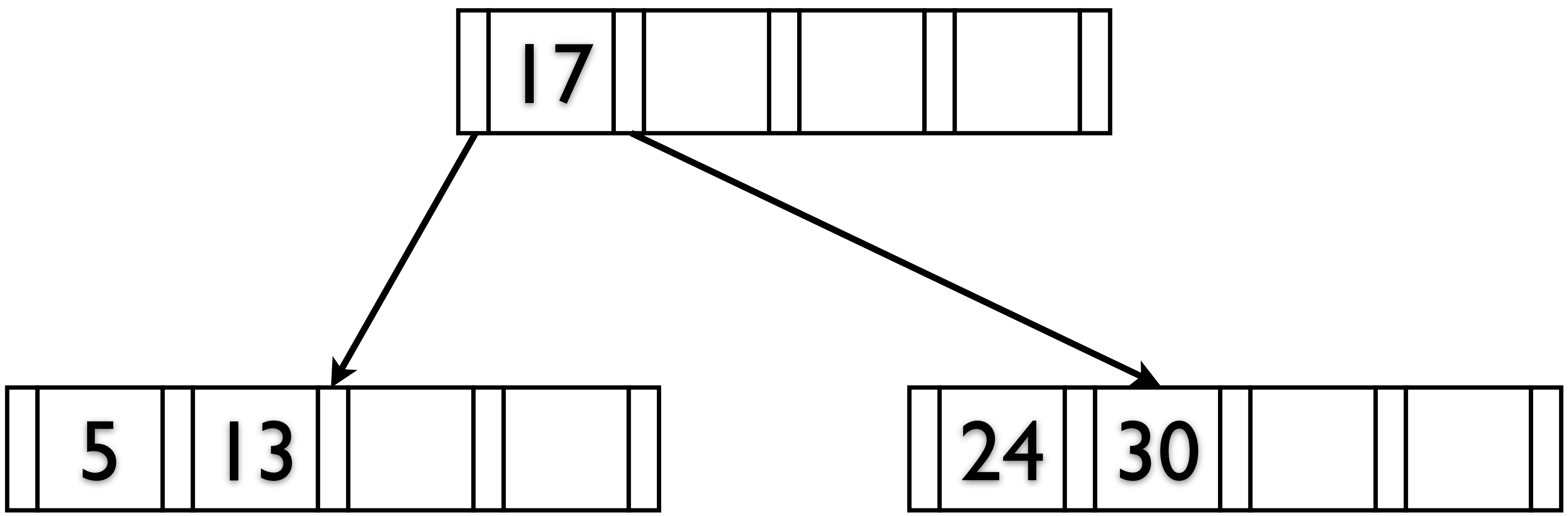
Insert 8

Inserting into B+ Trees

Move <17> into parent index : Root Split!

Copy <5> into parent index

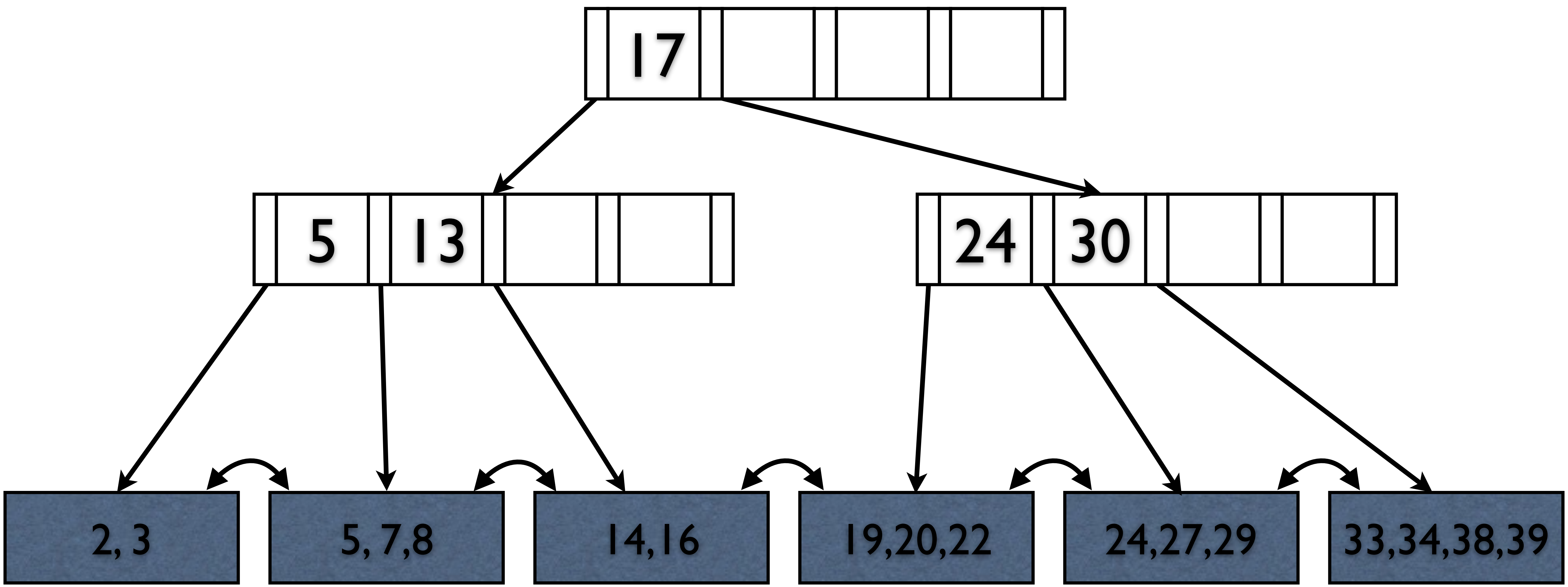
Inserting into B+ Trees



Move <17> into parent index : Root Split!

Copy <5> into parent index

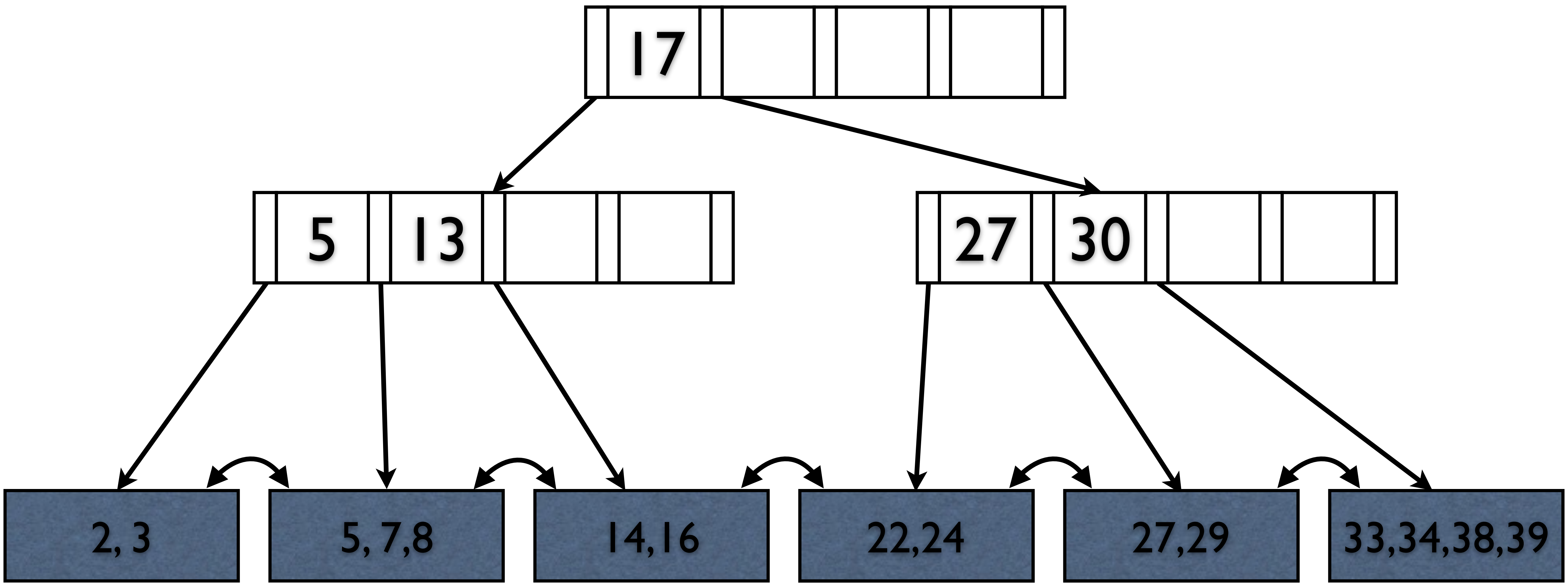
Inserting into B+ Trees



Why do we move, rather than copy the 17?

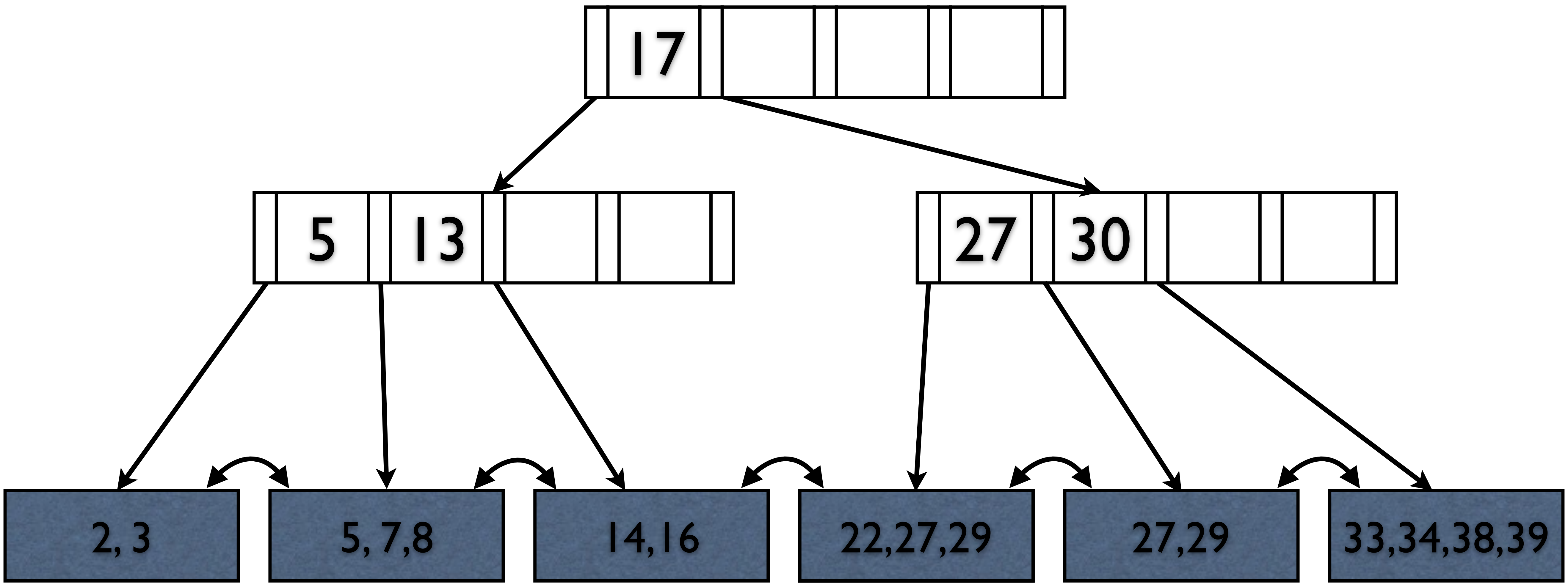
Are we guaranteed to satisfy our occupancy guarantee?

Deleting from B+ Trees



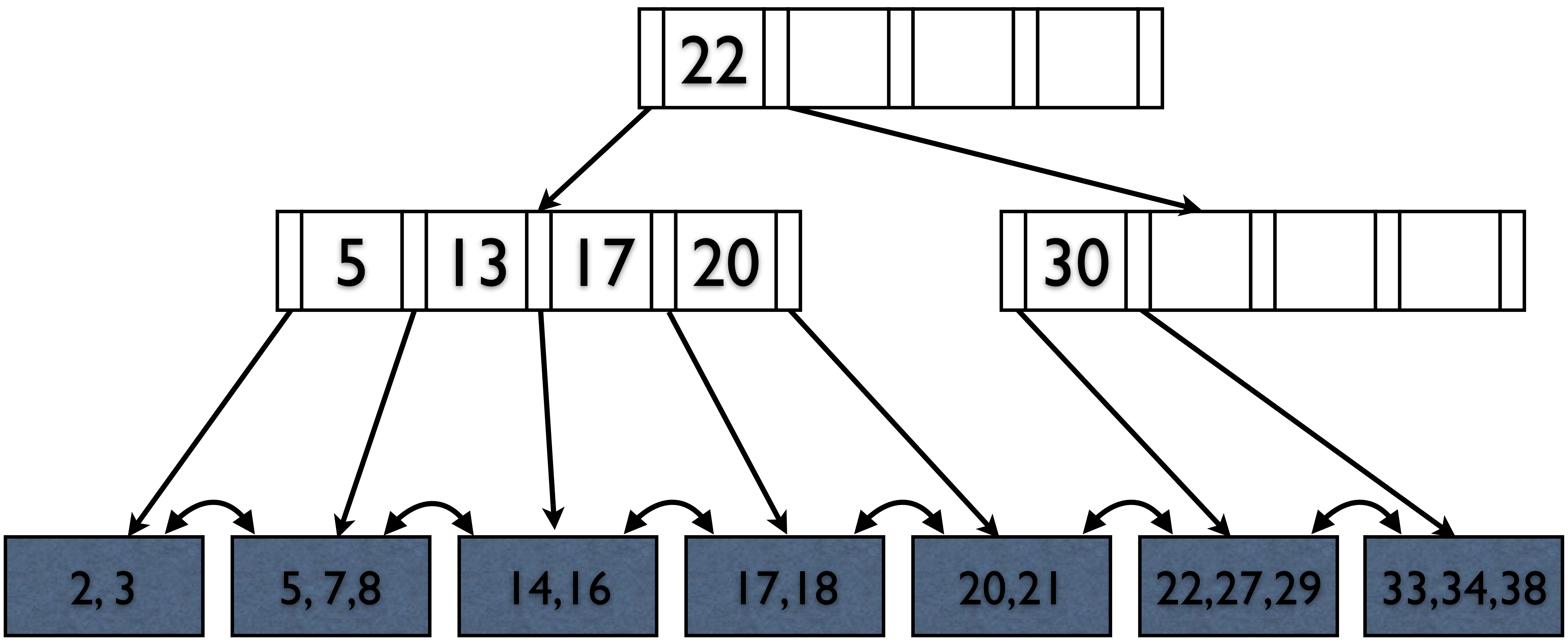
Delete 19 Delete 20

Deleting from B+ Trees

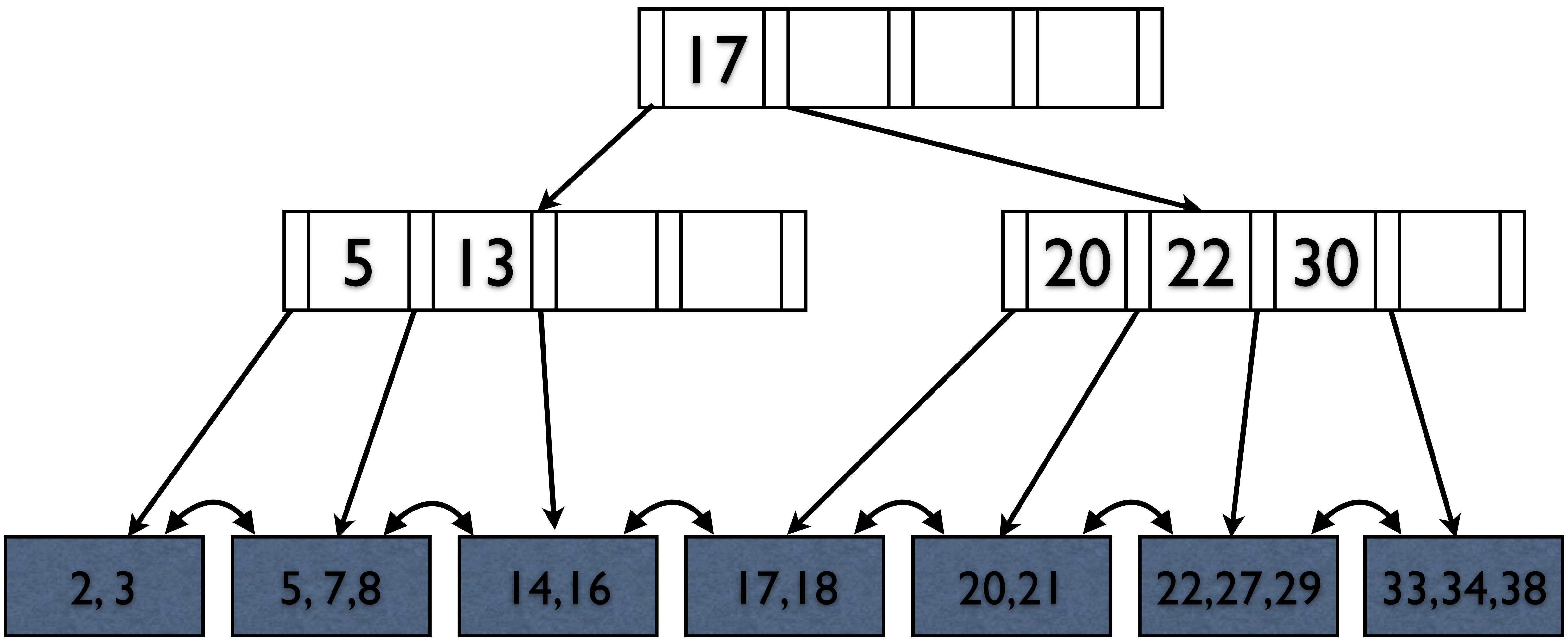


Delete 24

Non-Leaf Redistribution



Non-Leaf Redistribution



Intuitively, we rotate index entries 17-22 through the root