UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

CS411 - Indexing

illinois.edu

Announcements

- MP2 will be posted tonight
  - About 10 SQL statements to write
  - One week to finish
- Exam scores should all be posted

Review

- We're studying how databases are implemented
- What are the levels of the memory hierarchy?
- Which level are we optimizing for databases?
- What is meant by "volatile" memory?

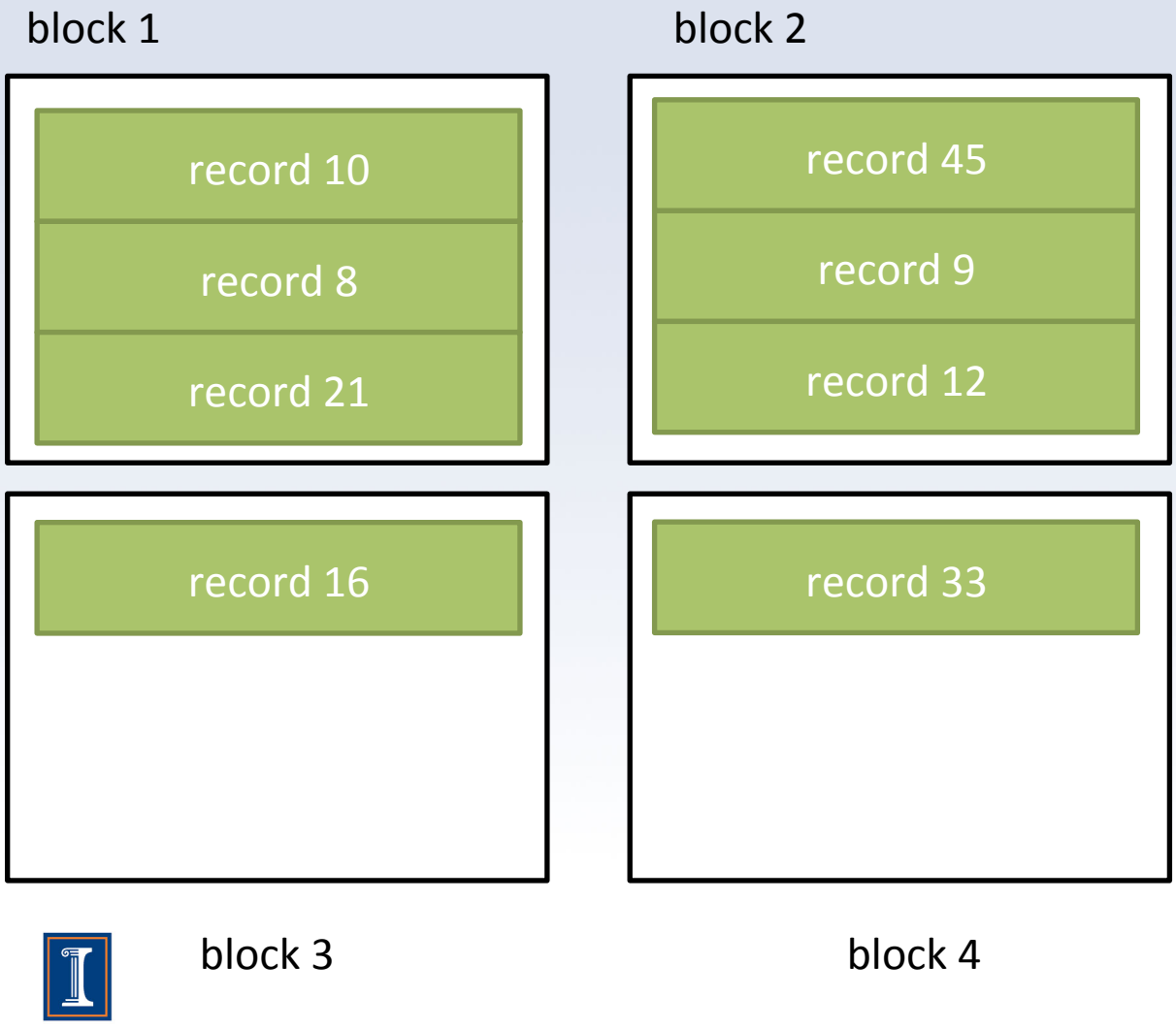
Review

- What steps are required to access a block?
- What is the advantage sequential reads?
- What is a spanned record?
- What are BLOBS?
- What is a tombstone?

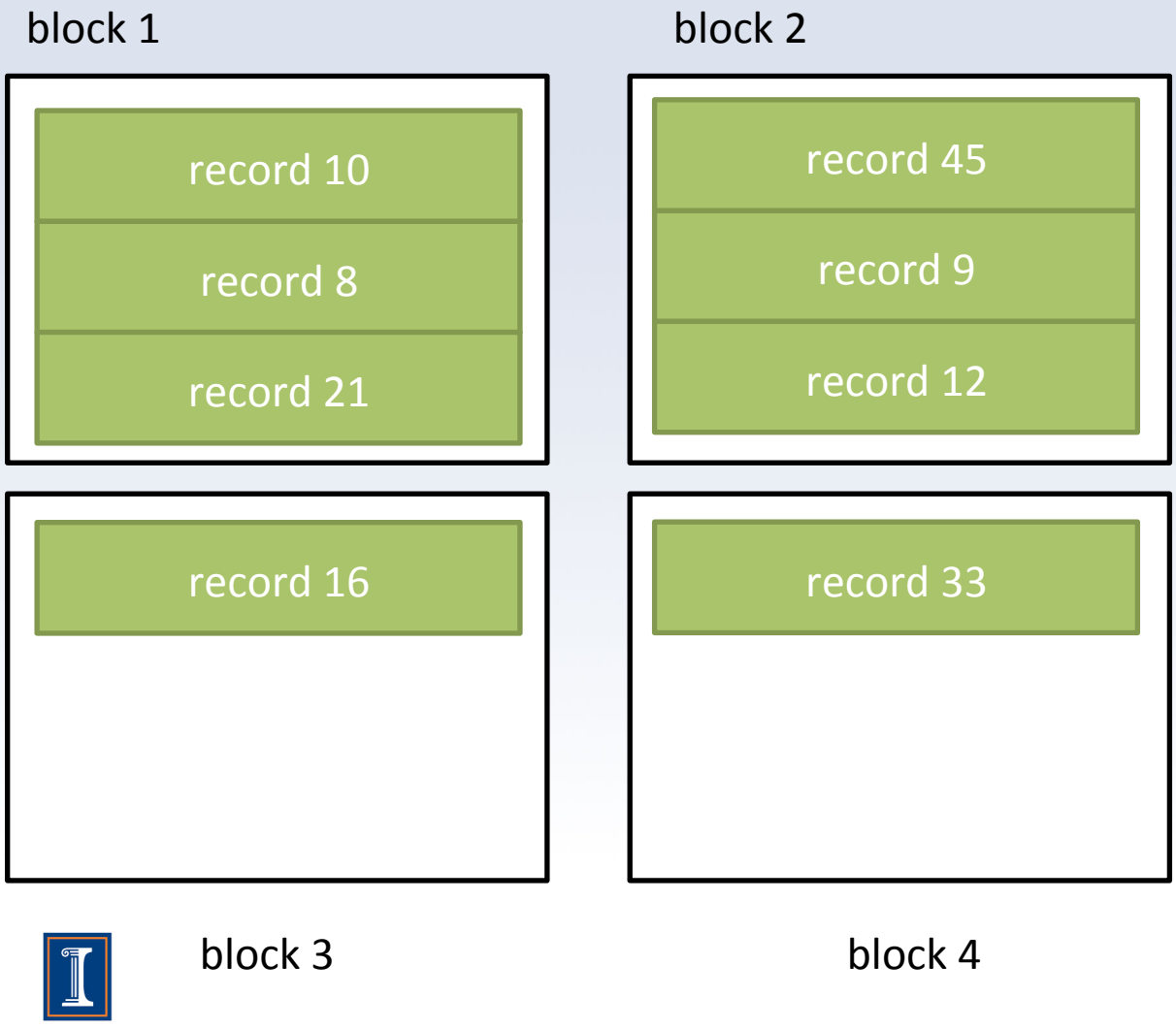
So far…

- We've studied how to arrange data on the disk
- Now we're going to study how to lay out the data so it can be queried more efficiently

A relation on disk

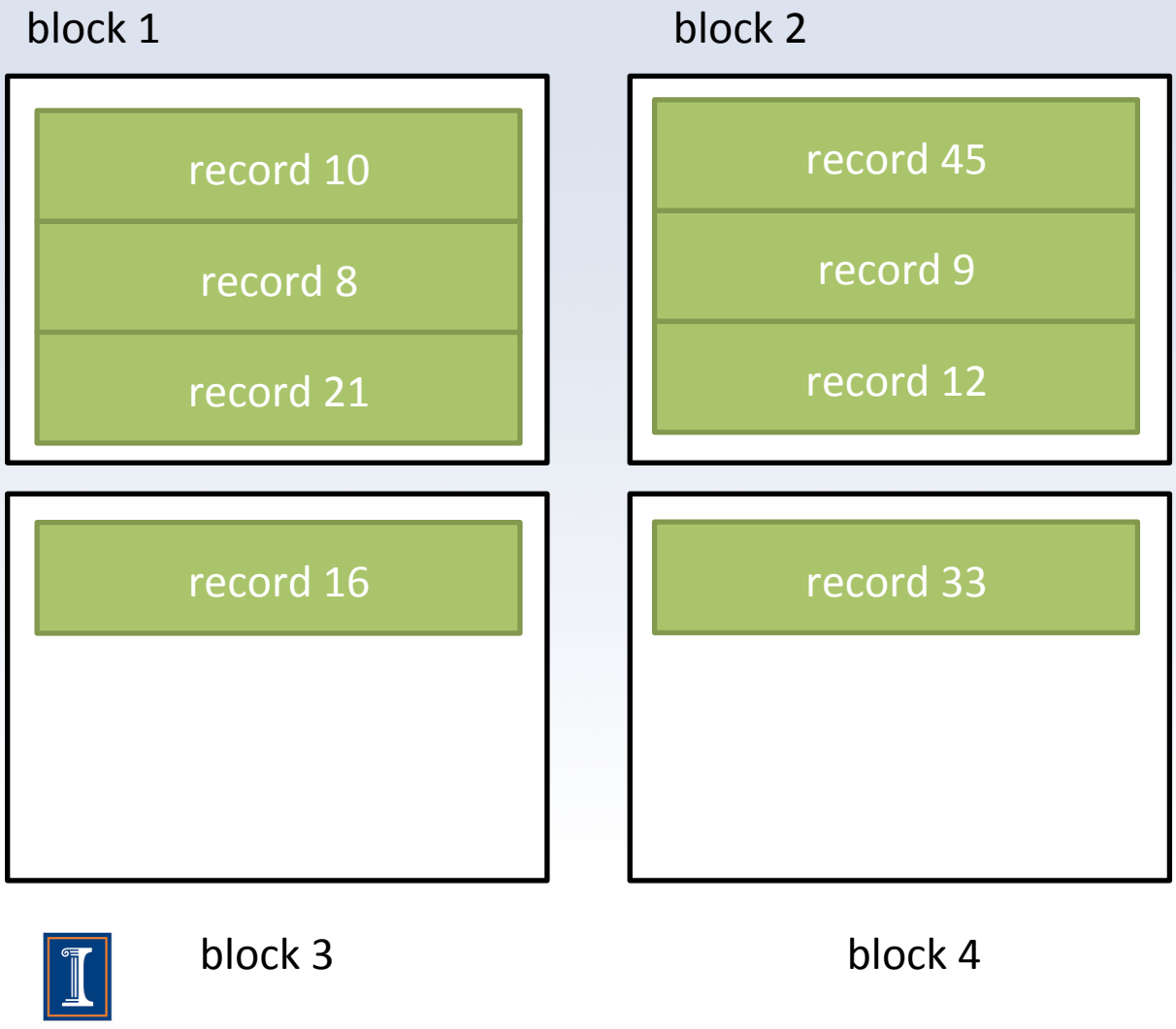


A relation on disk



SELECT *
FROM R;

A relation on disk




```
SELECT *
FROM R
WHERE id=33;
```

Indexes

- An index on a relation speeds up selection on the search key fields
- Search key - any subset of the attributes of a relation
  - ***not*** to be confused with a relation's key

Indexes

- Entries in an index: (k,r)
  - k = the key
  - r = the record OR a pointer to a record OR a pointer to a collection of records
- Stored in an "index file"
  - "data file" is what we call a relation on disk

Index terminology

- Clustered/unclustered
  - Clustered - records sorted in key order
  - Unclustered - not sorted by search key
- Dense/sparse
  - Dense - each record has an entry in index file
  - Sparse - only some records have entries

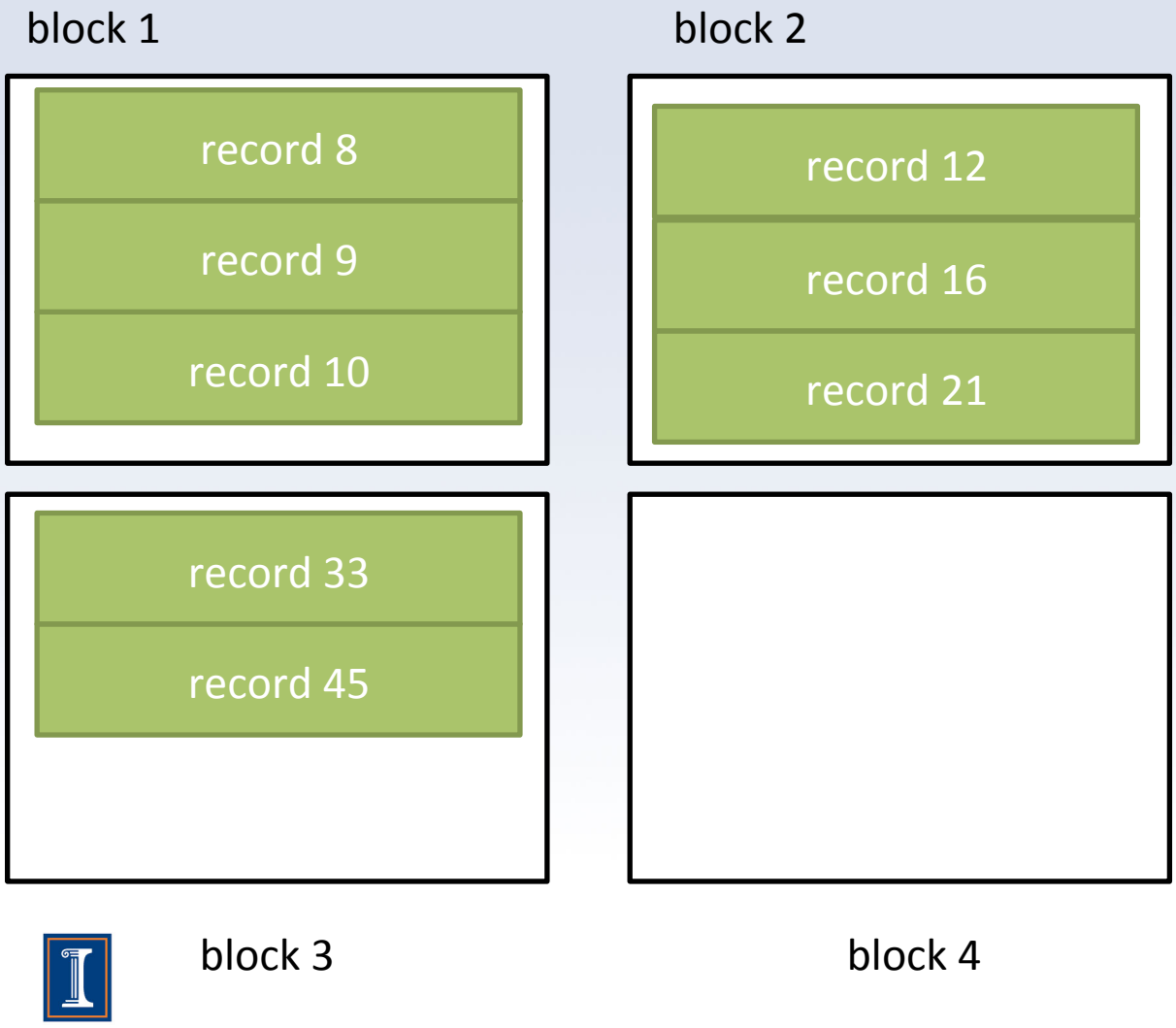
Primary/secondary

- Primary - index is on the primary key of the relation
  - Records are sorted on disk by primary key
  - Clustered index
- Secondary - index is on some other set of attributes

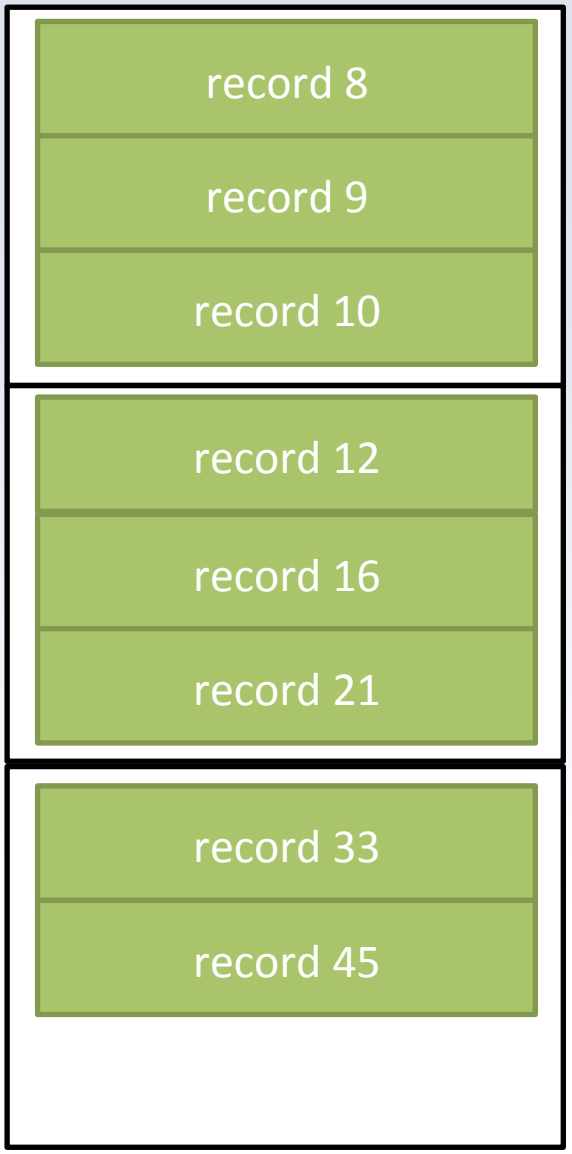
Sequential File

- A data file sorted by the primary key is called a *sequential file*

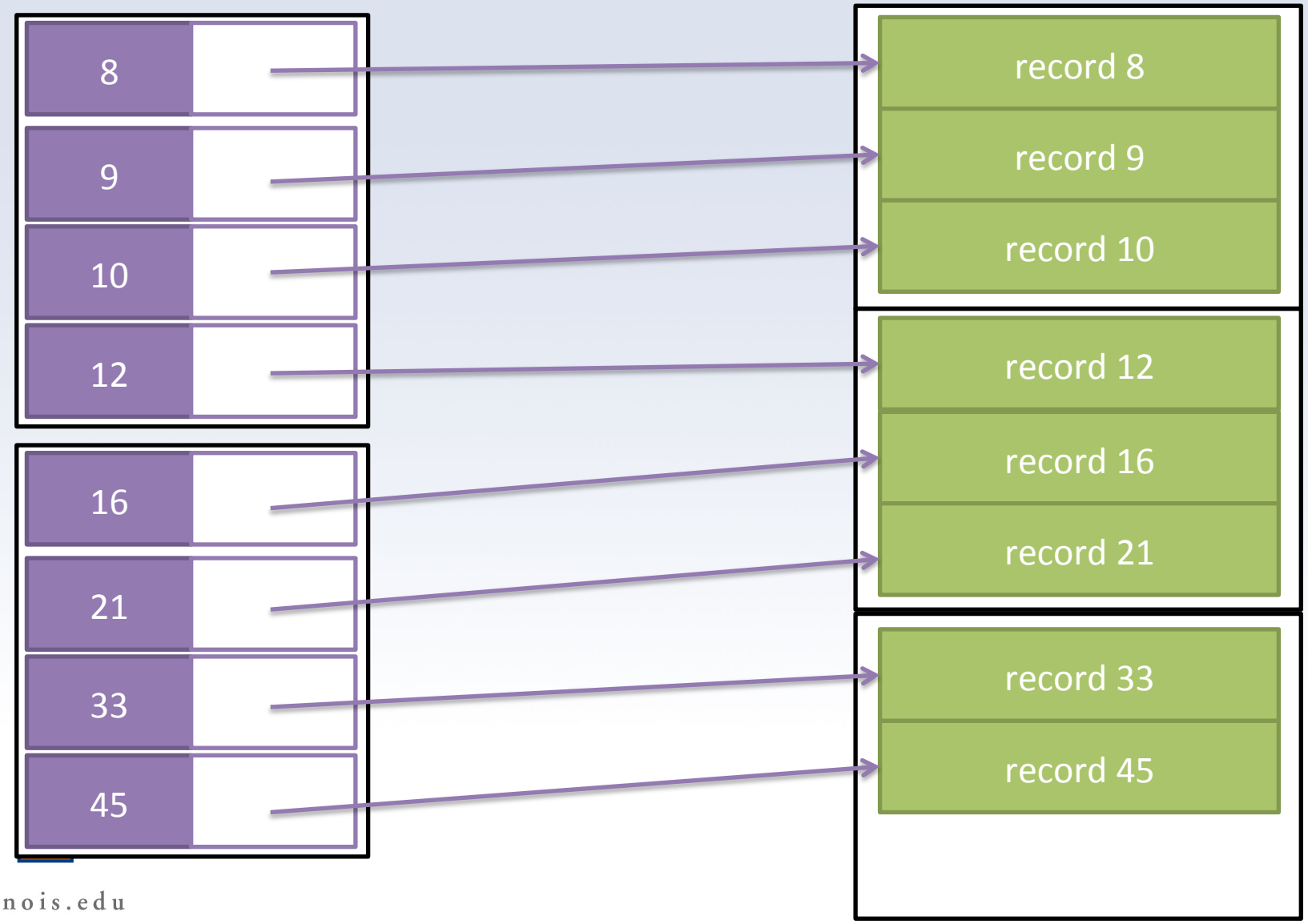
Sequential File



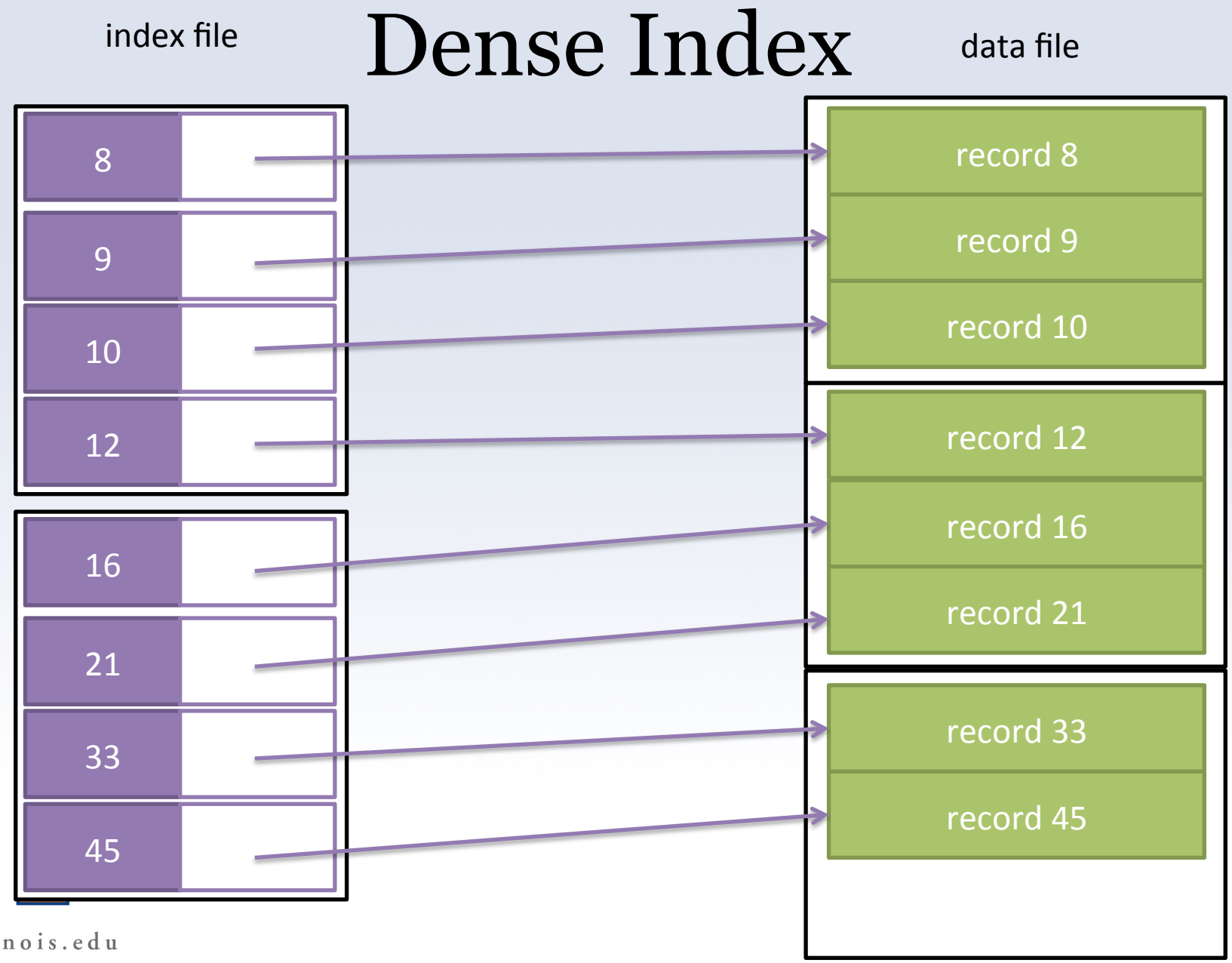
Sequential File



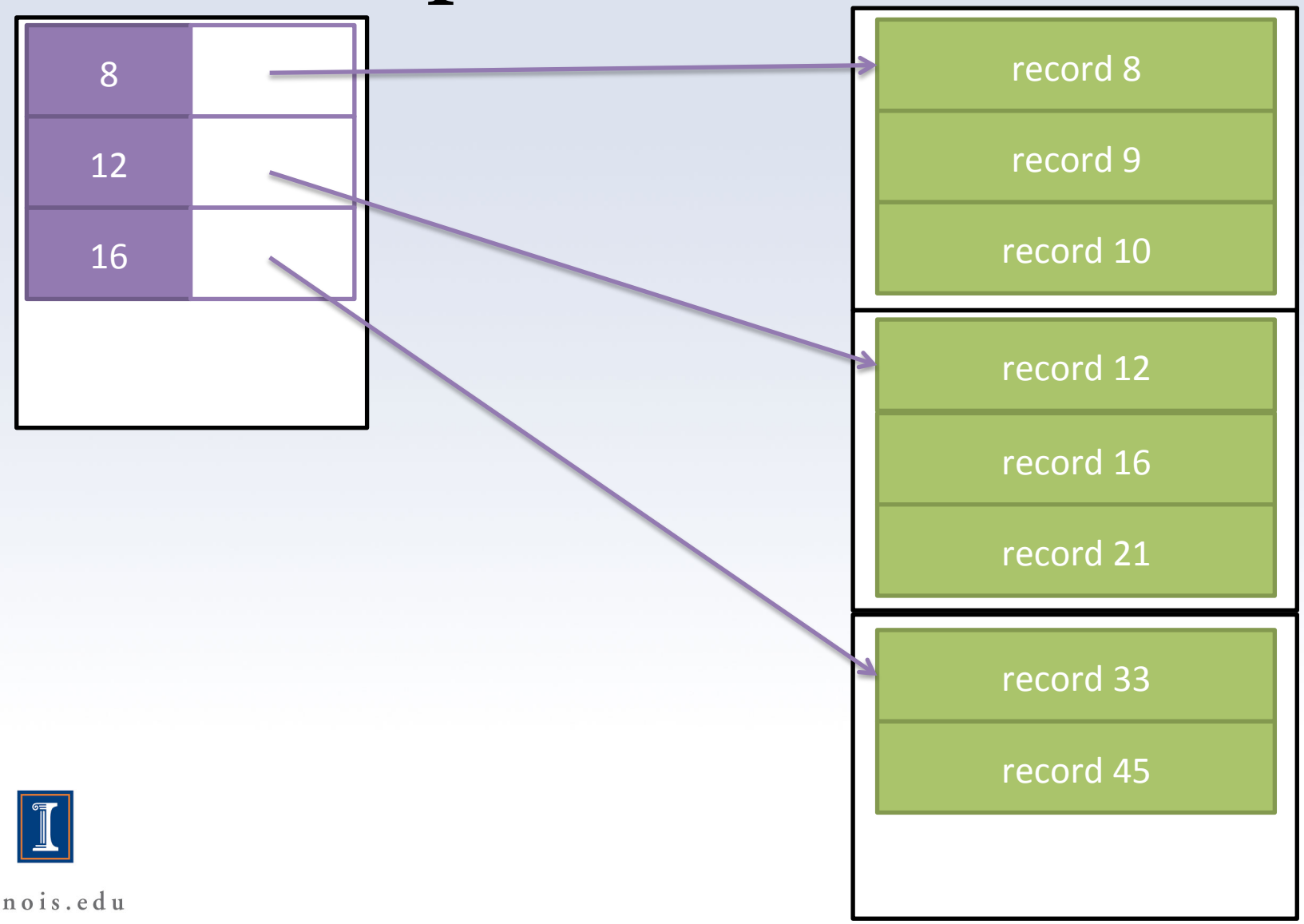
Dense Index


8
9
10
12
16
21
33
45
record 8
record 9
record 10
record 12
record 16
record 21
record 33
record 45

Dense Index


index file
data file
8
9
10
12
16
21
33
45
record 8
record 9
record 10
record 12
record 16
record 21
record 33
record 45

Sparse Index



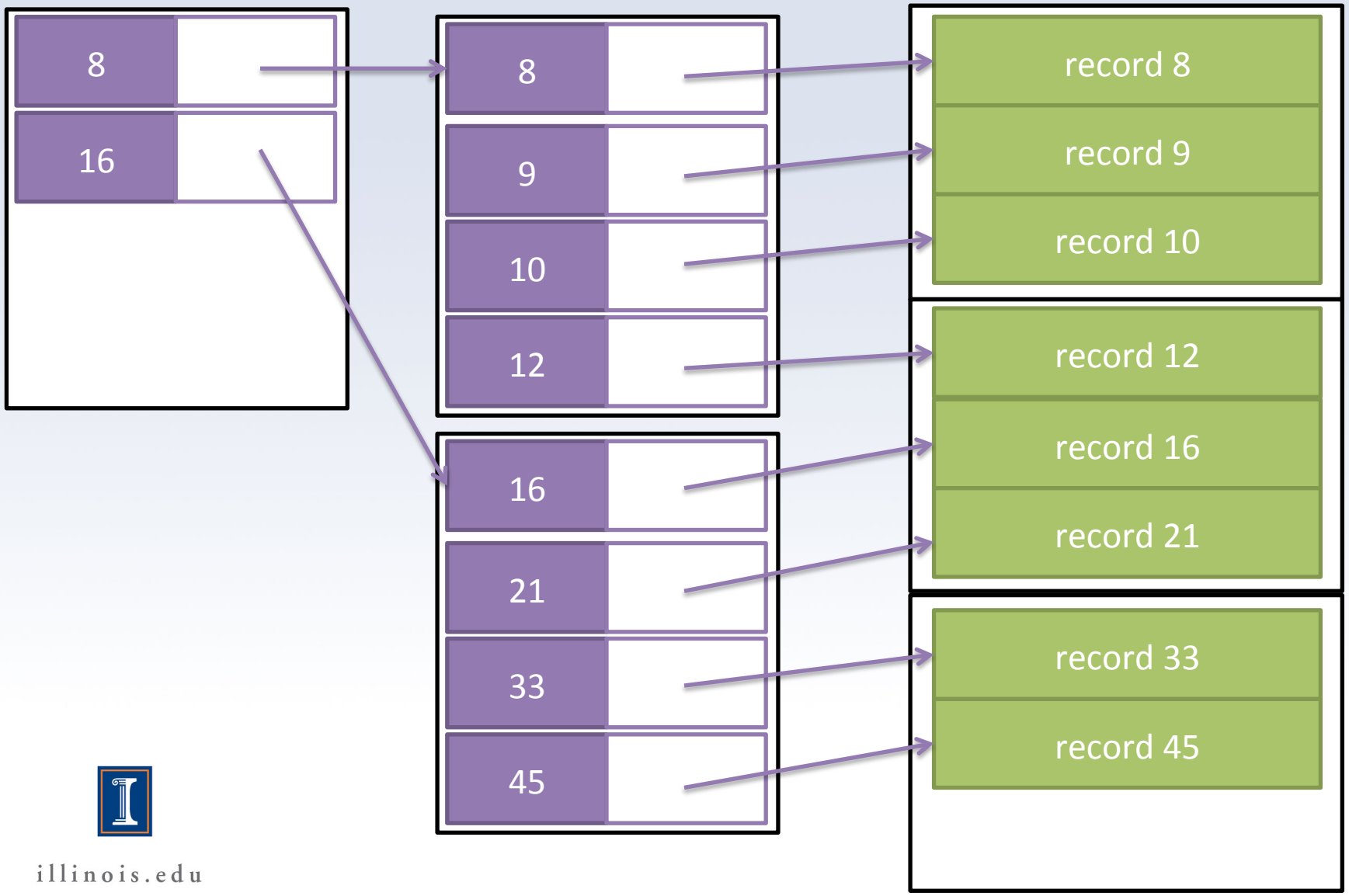
Sparse index

- Note: can only use sparse index if data file is sorted
  - if index is not for primary key, ***must*** use a dense index

Multilevel index

- If index file is large, binary search for key can require many disk operations
- Why?
- To mitigate this, we can put an index on the index
- Obviously, this can be repeated
- B-trees do this in an optimal way

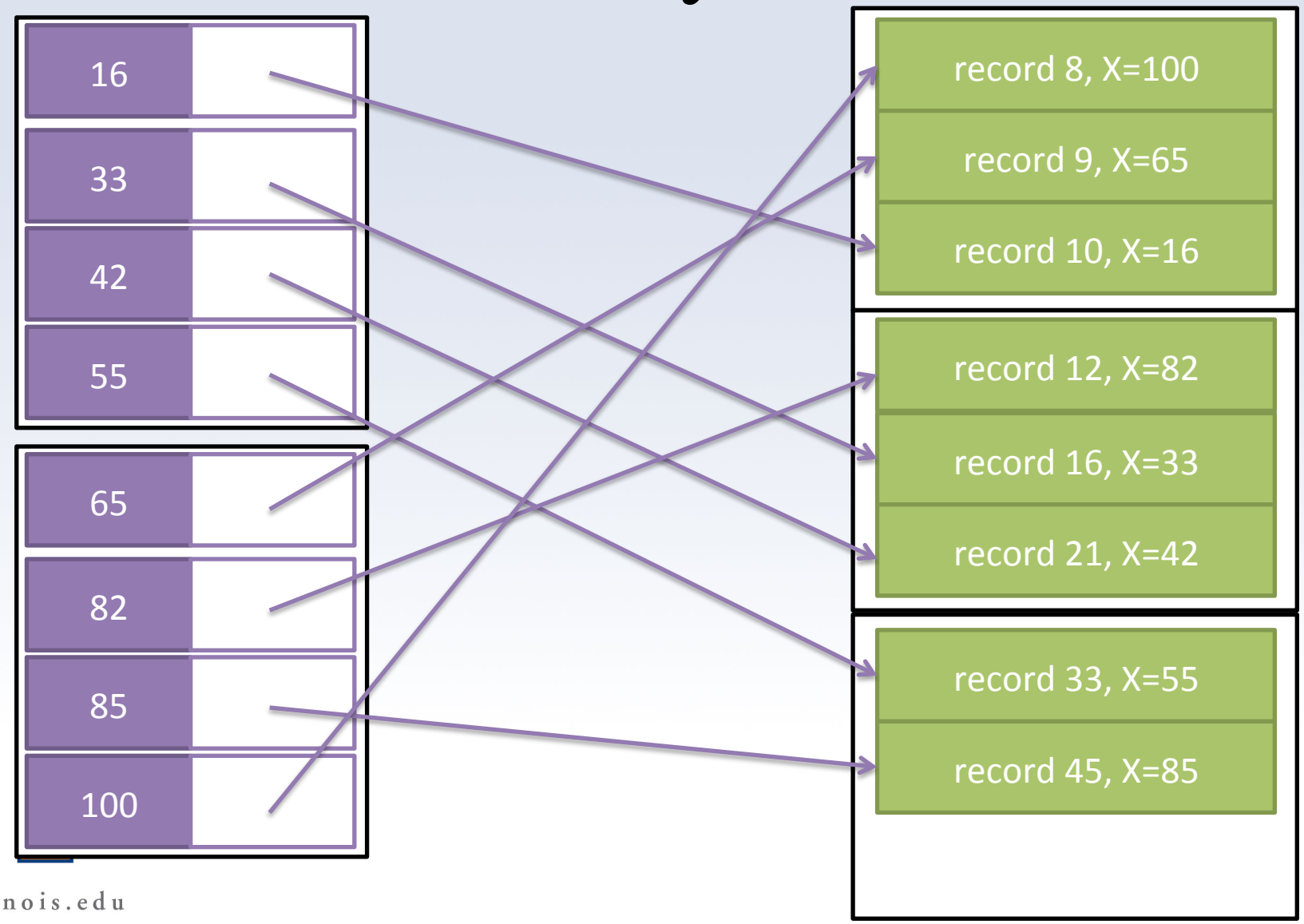
Multilevel Index



Secondary indexes

- To speed up search for non-primary key fields, we create a ***secondary index***
  - Must be unclustered
  - Must be dense

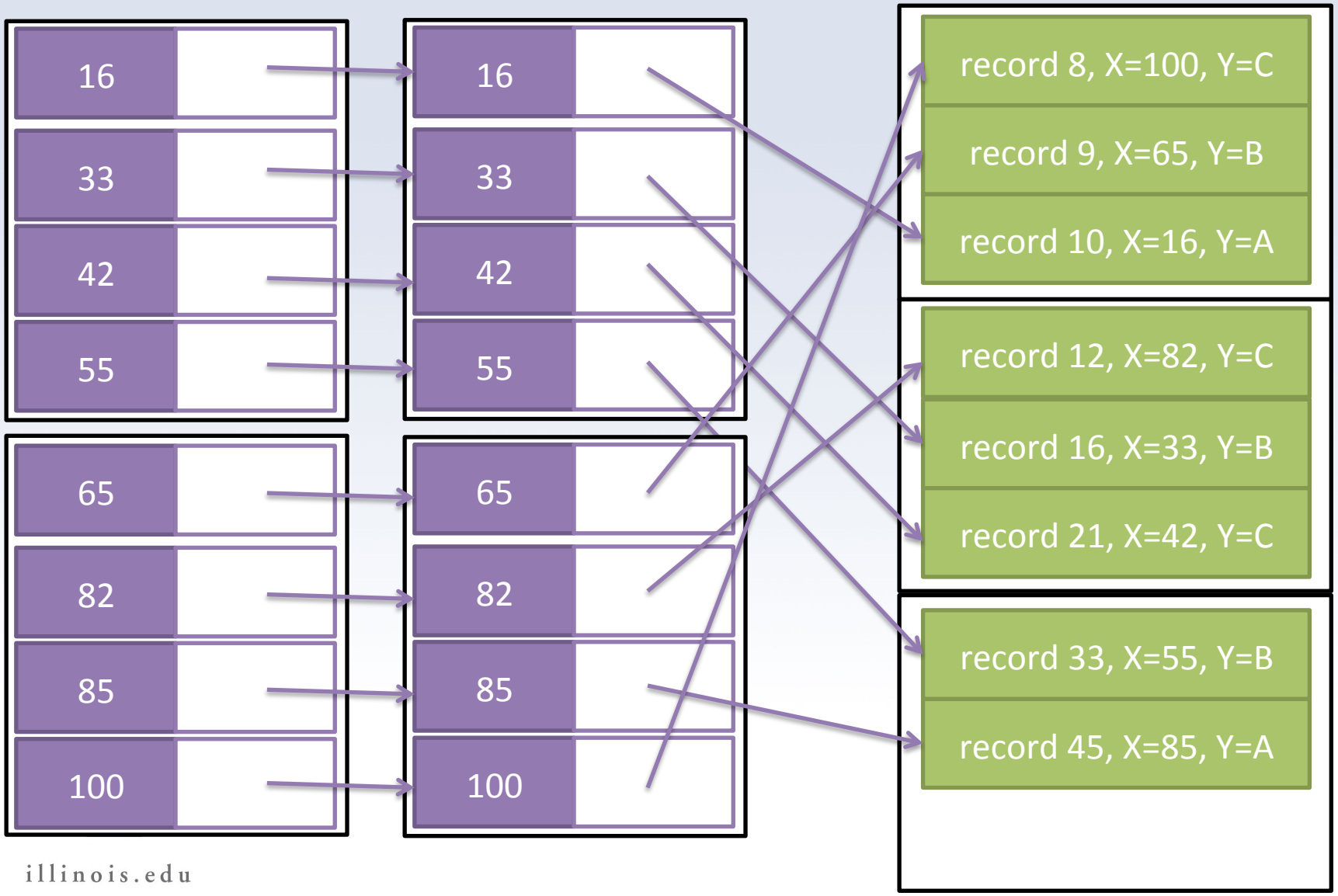
Secondary Index



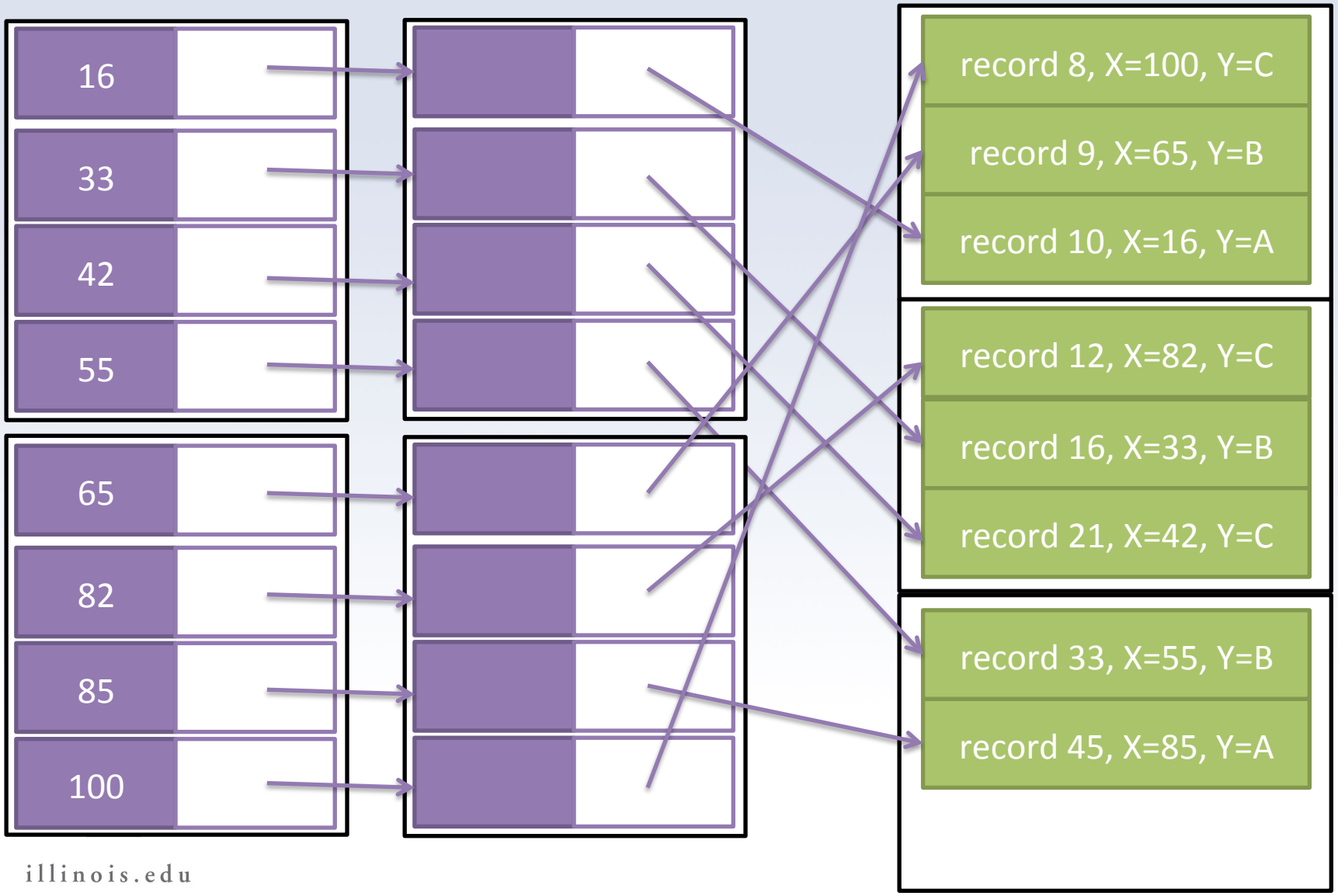
Secondary Indexes

- A layer of indirection can improve a secondary index
  - Eliminates replication of search key values
  - Can be used to perform intersection
- We call this layer of indirection a ***bucket***
  - We call intersection ***bucket intersection***

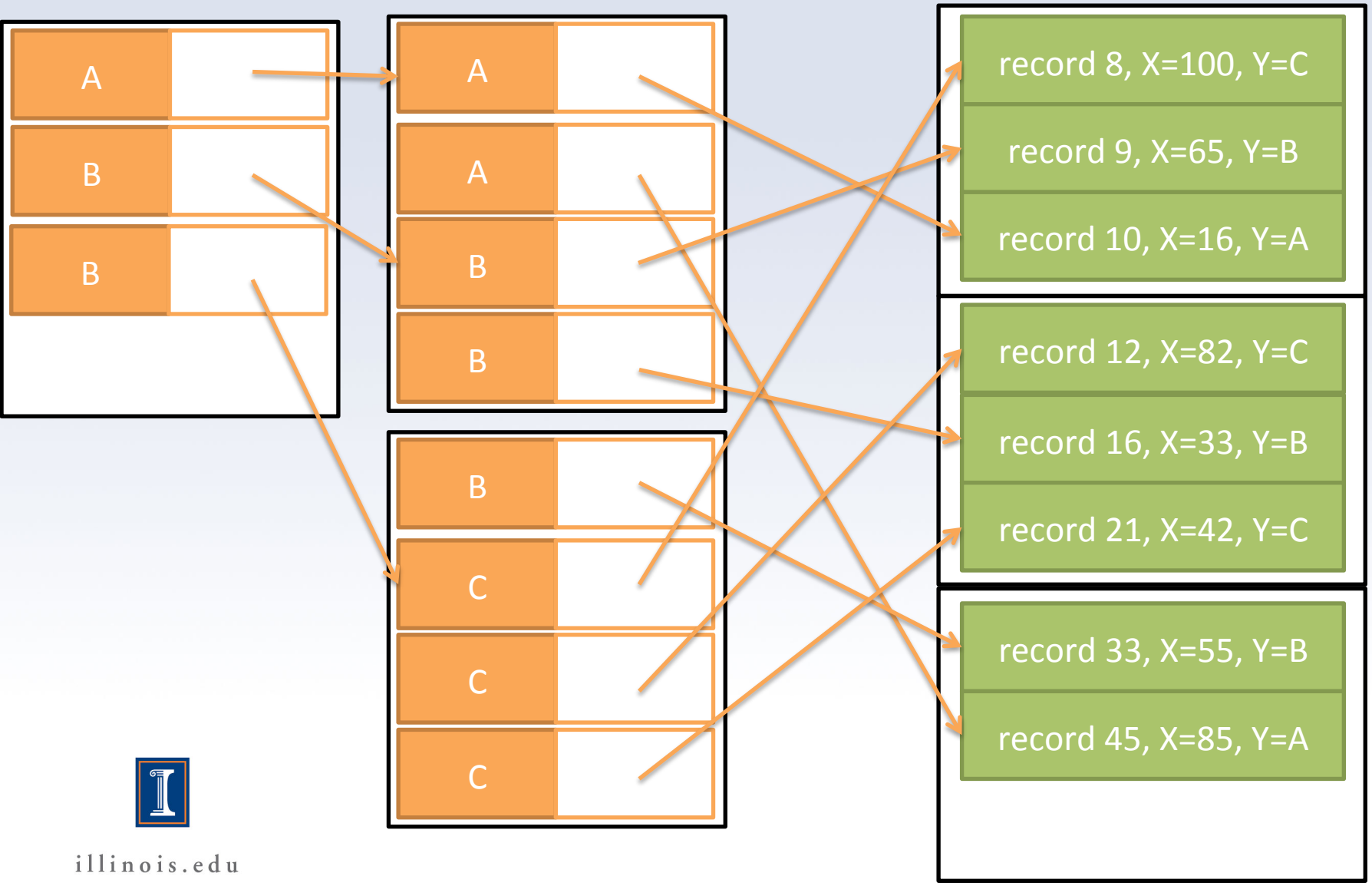
Indirection with Buckets



Indirection with Buckets

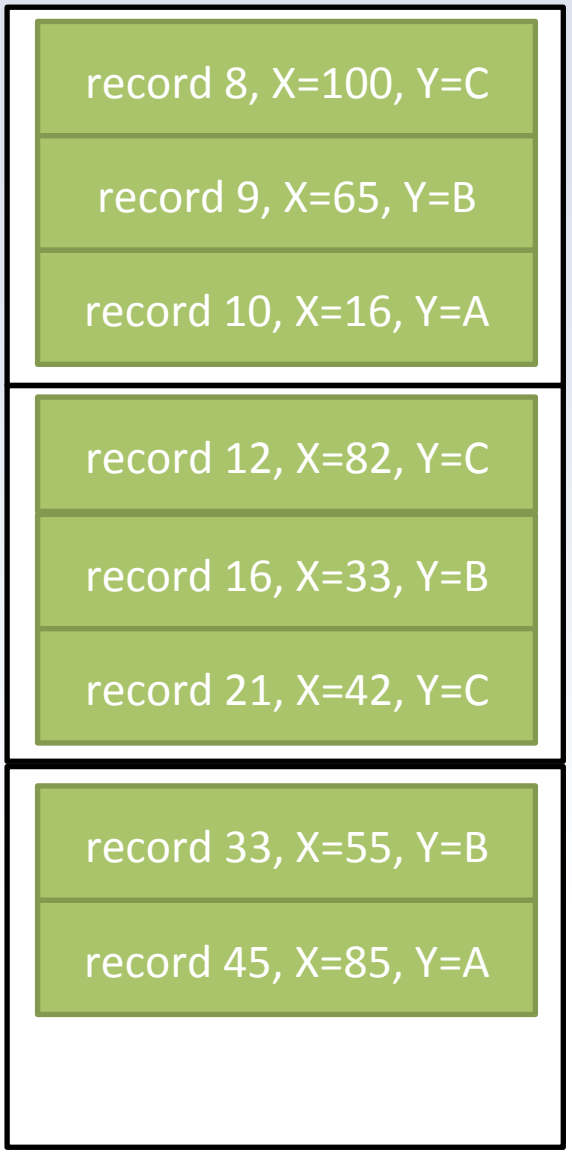


Indirection with Buckets

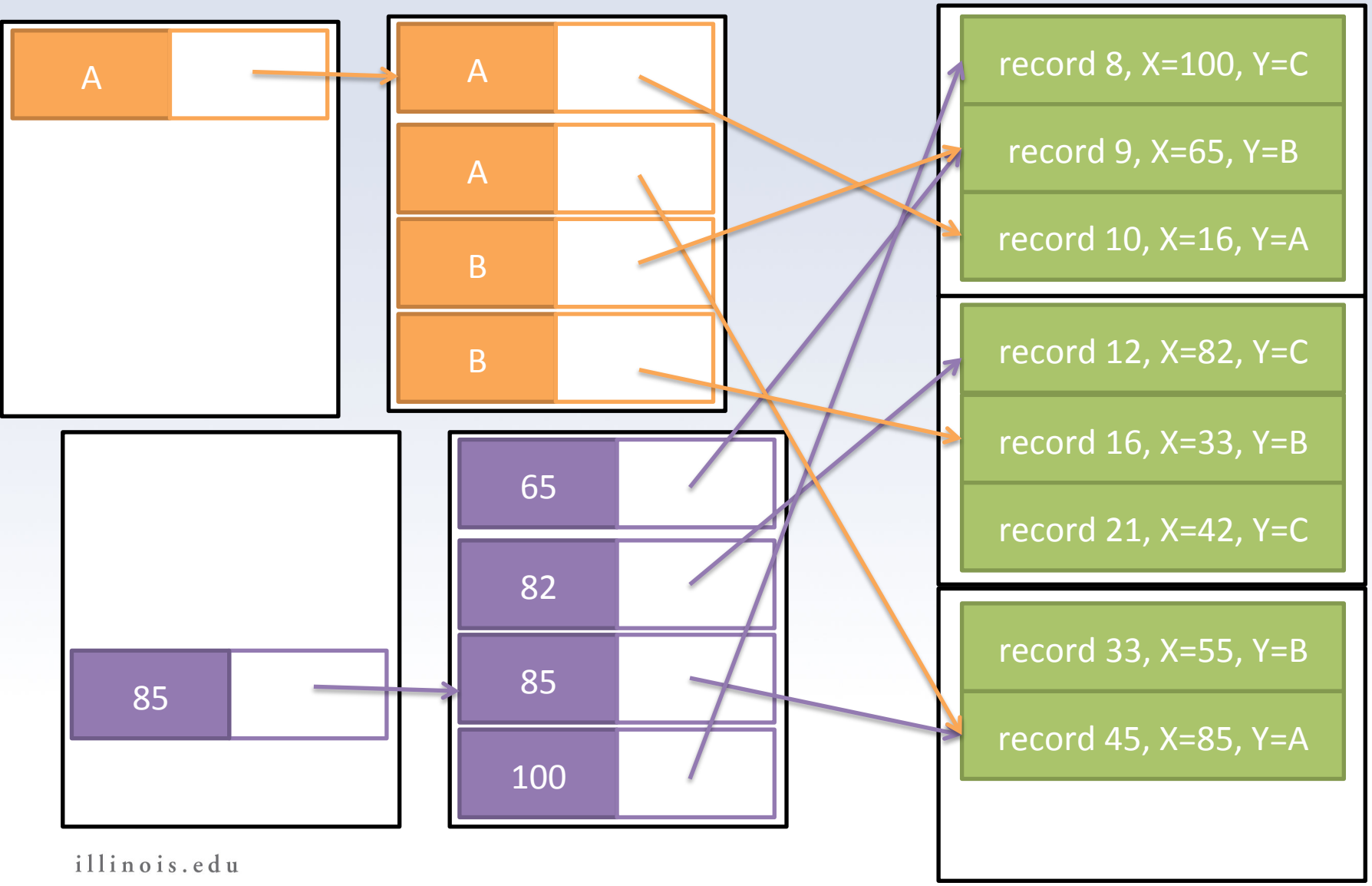


Bucket Intersection

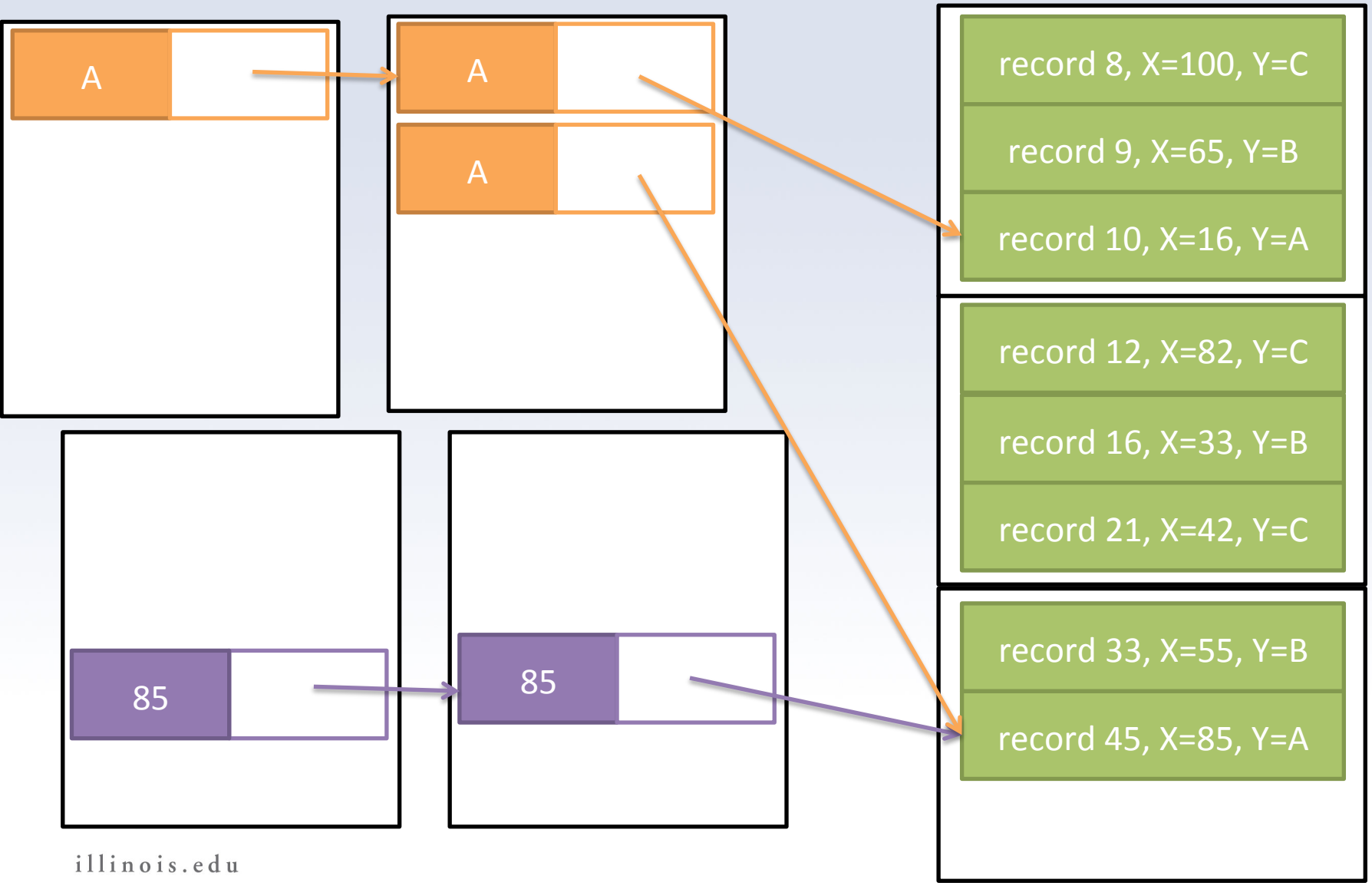
```
SELECT *
FROM R
WHERE X=85 AND Y=A
```



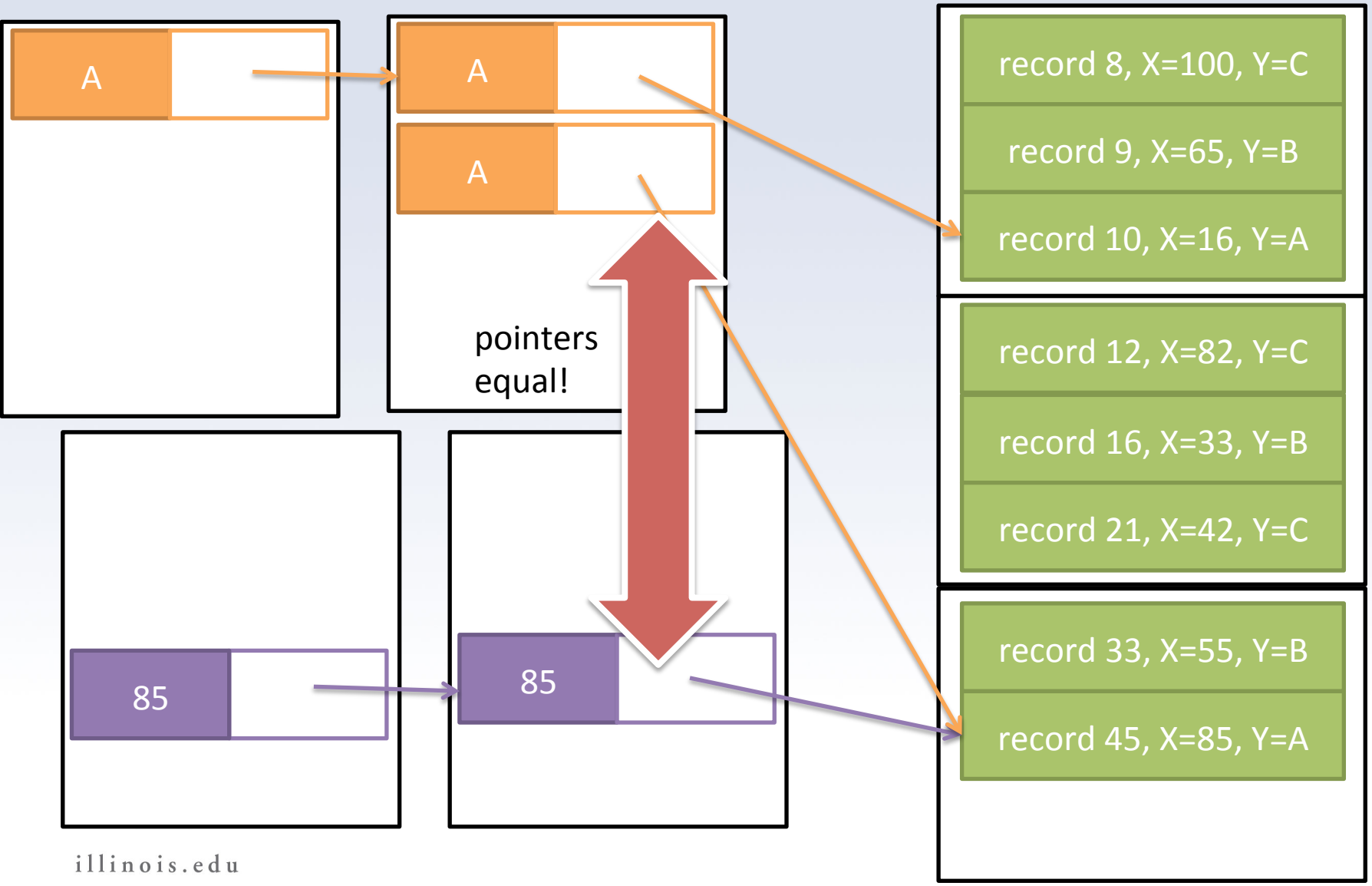
Bucket Intersection



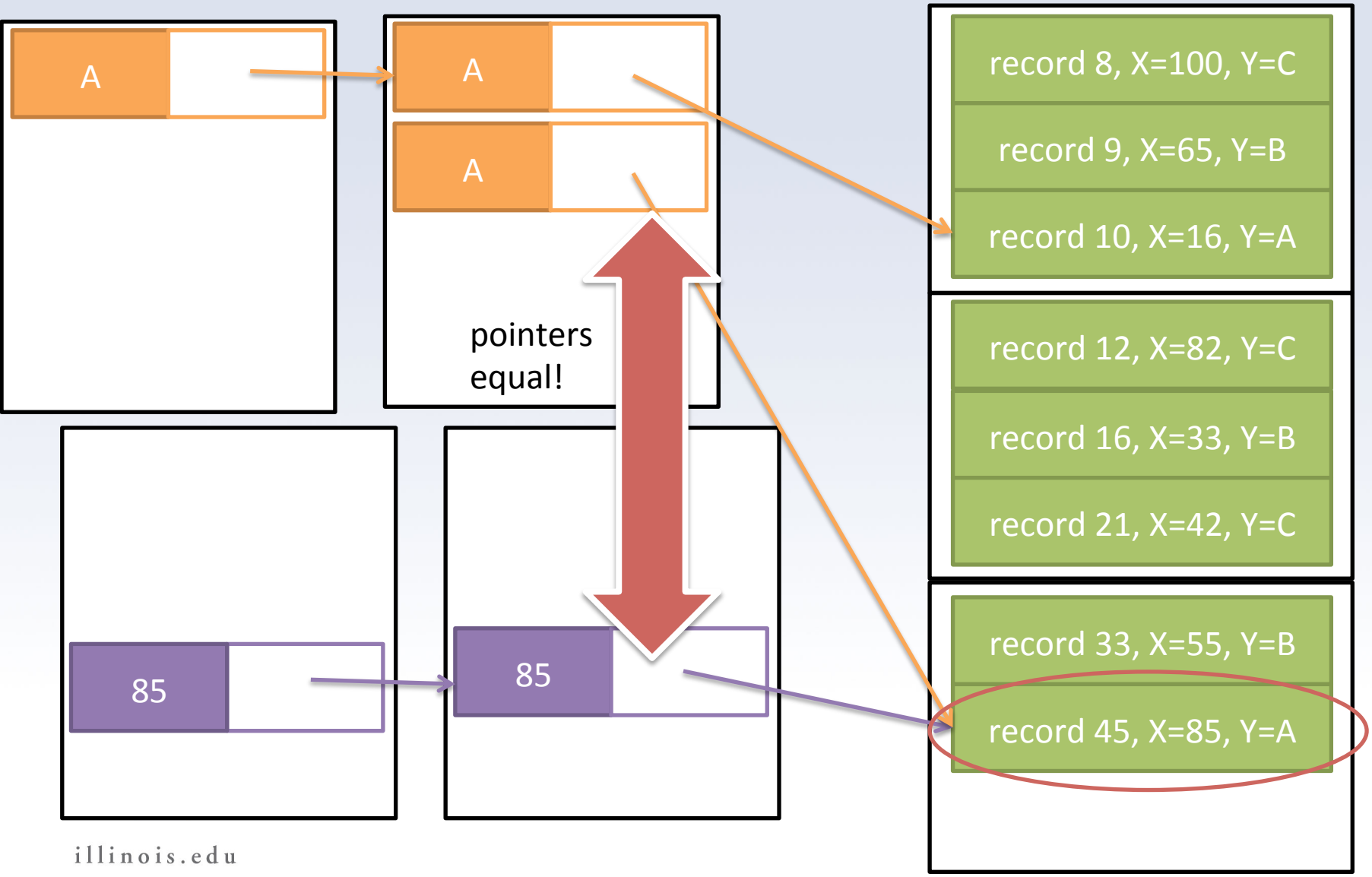
Bucket Intersection



Bucket Intersection



Bucket Intersection



B-trees

- General purpose multi-level indexing structure
  - automatically maintains appropriate number of levels
  - automatically maintains space usage in blocks so they are at least half full
  - automatically *balanced*; all paths from root to leaves are equal

B-trees

- There are many variants of the B-tree
  - What we will call a “B-tree” is actually a “B+ tree”
  - This is the only variation we will discuss, so we’ll just refer to it as a “B-tree” from now on

B-trees

- Has a single parameter *n*
  - Each node in the tree has *n* search key values
  - Each node also has *n+1* pointers
  - We pick *n* to fill up a block as best we can

Example

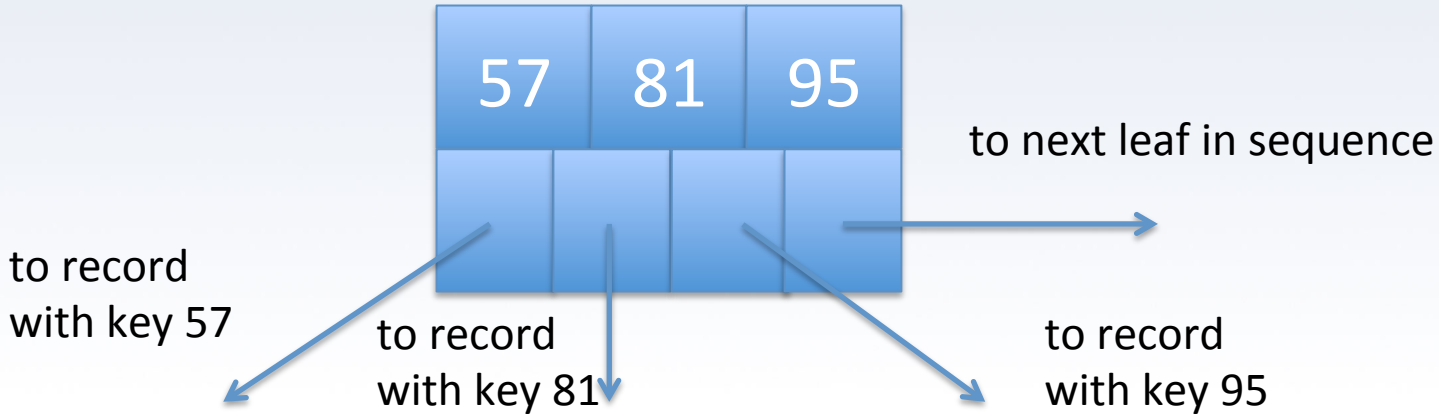
- The search key is a 4 byte integer
- Pointers are 8 bytes
- Pages are 4096 bytes
- Find largest n such that $4n+8(n+1) \leq 4096$
- $12n+8 \leq 4096$
- $n=340$

B-trees

- In our running example
  - Search key will be a single integer
  - we will use the incredibly small parameter setting n=3

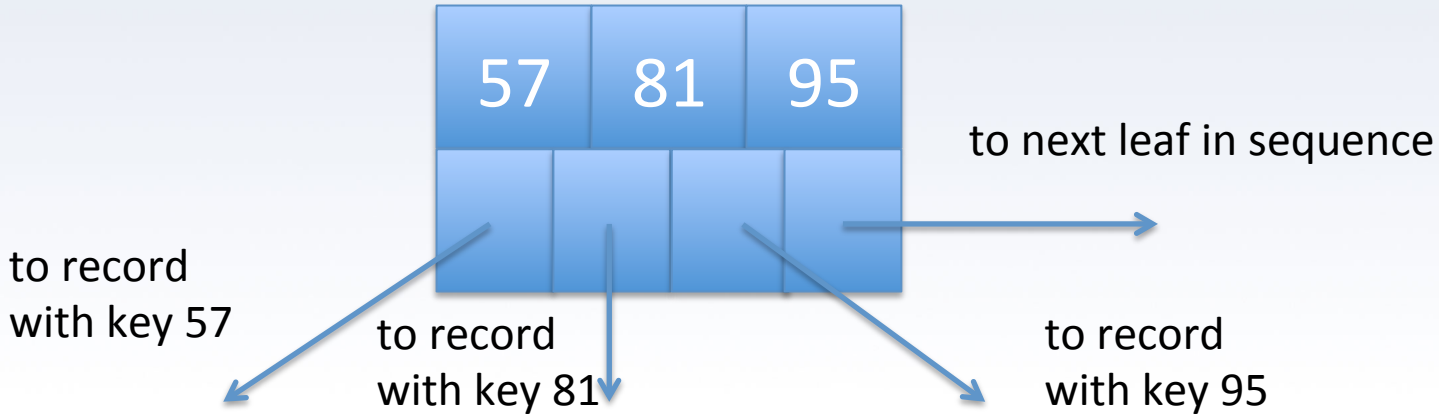
Leaf nodes

- Store key values and pointers to the actual record
- Also keep a pointer to the next leaf



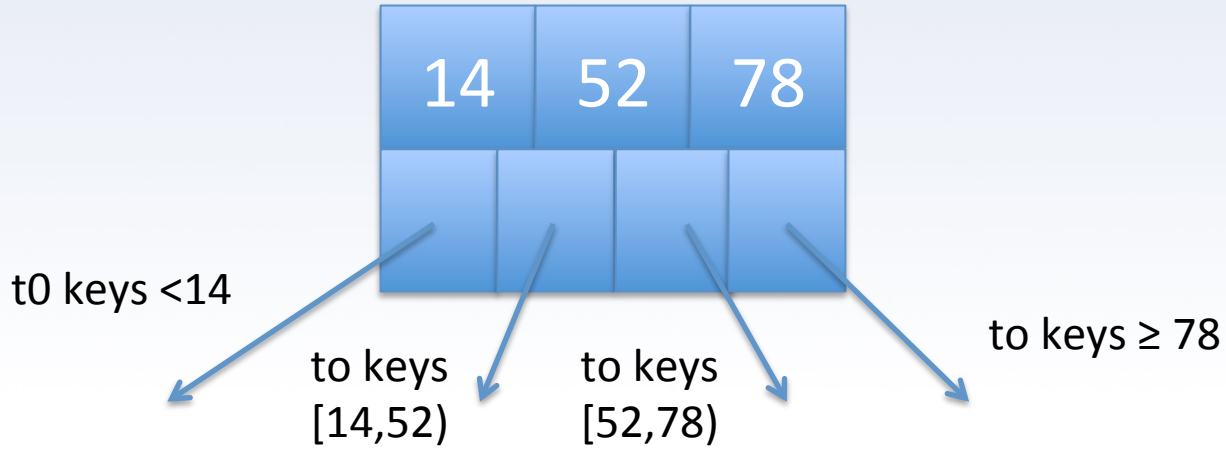
Leaf nodes

- At least $\lfloor (n+1)/2 \rfloor$ of the pointers must be used



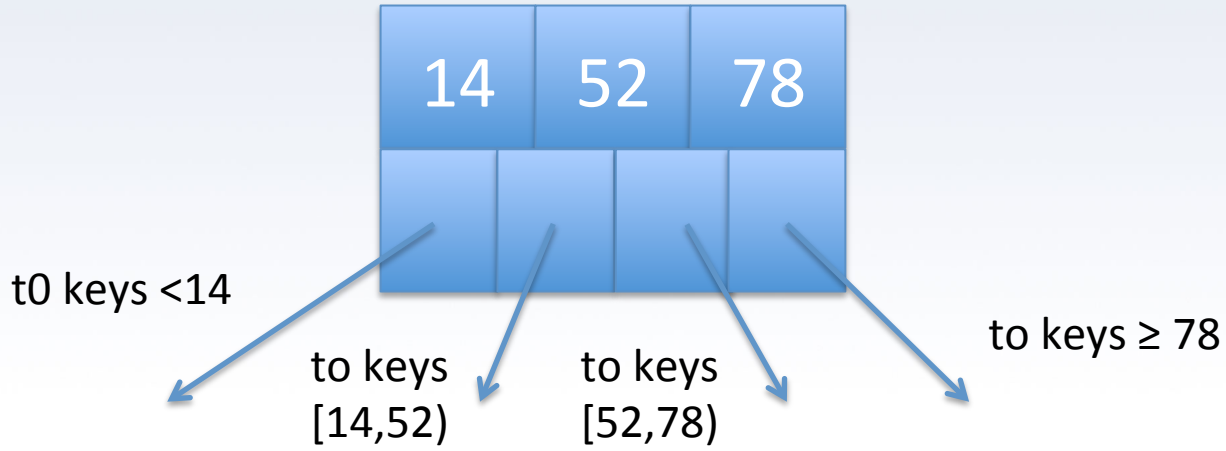
Interior nodes

- Contain pointers to blocks at lower level
- Key values indicate range of values in lower levels

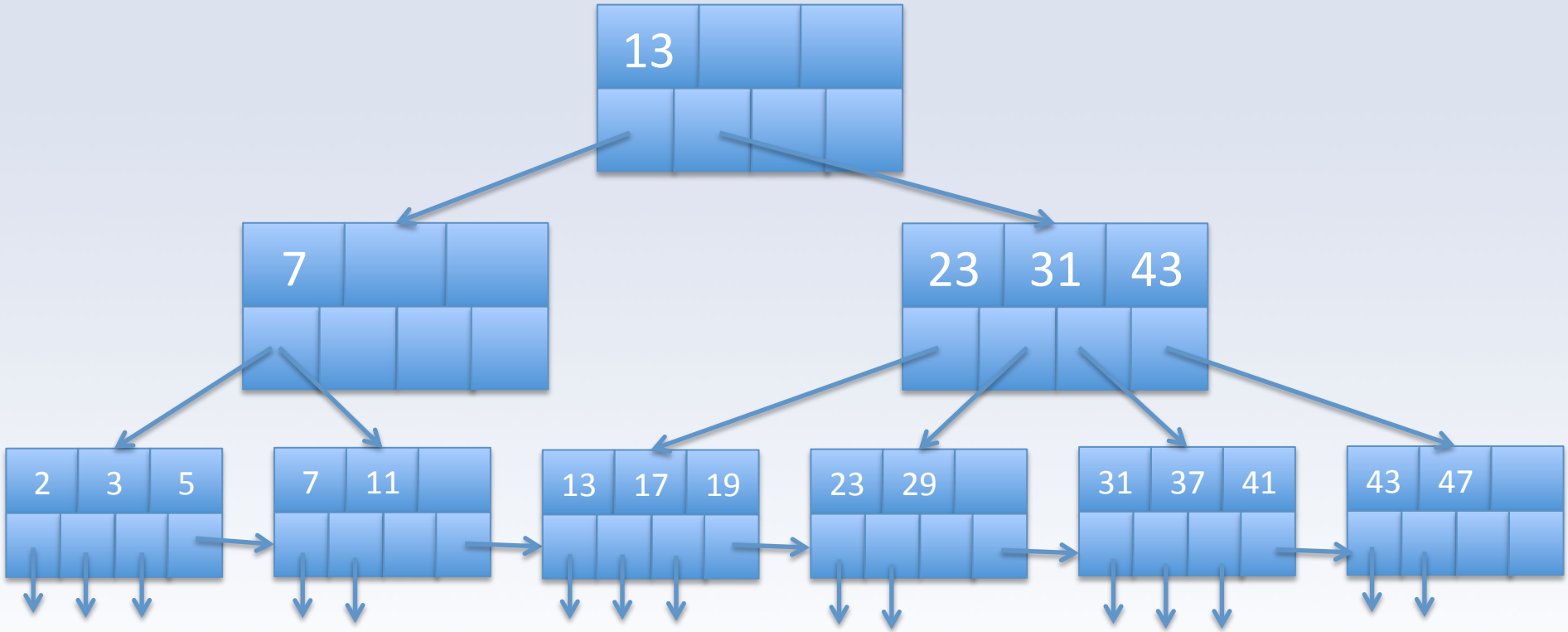


Interior nodes

- At least $\lceil (n+1)/2 \rceil$ of the pointers must be used
- Exception: root must have at least2 children



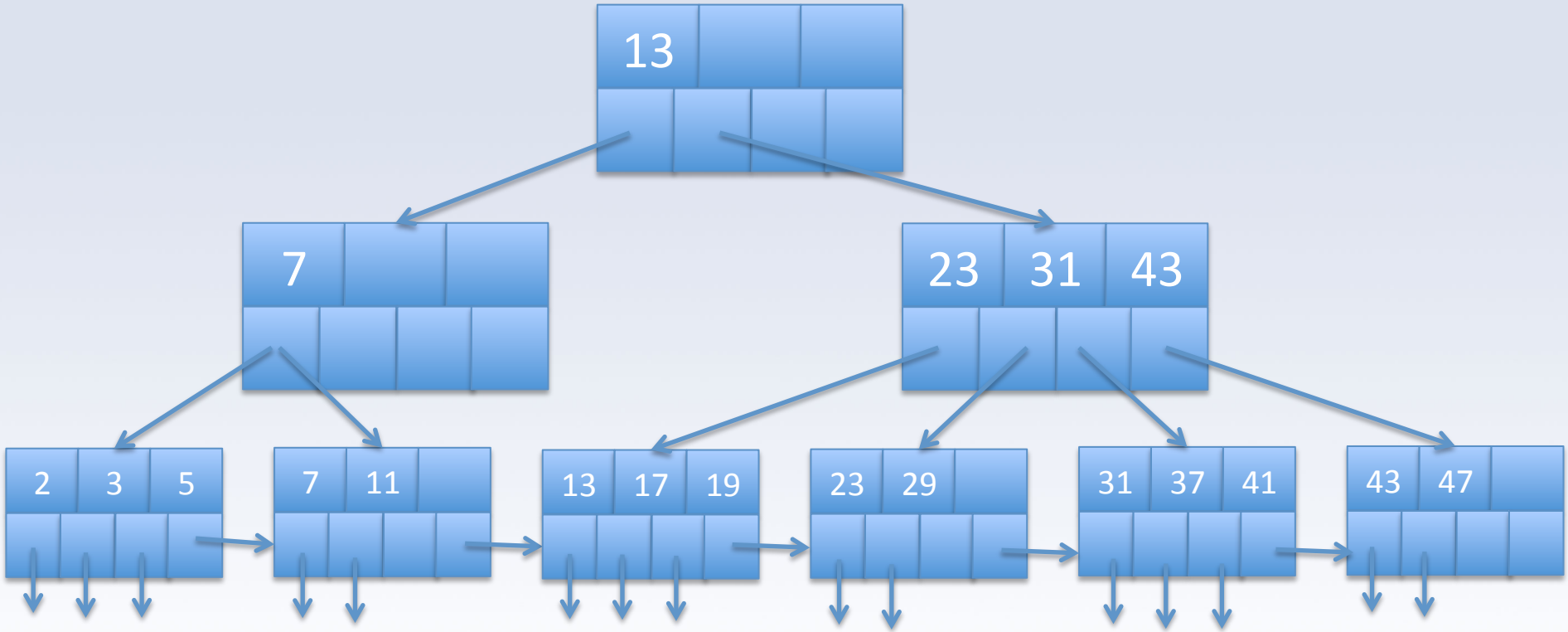
Example B-tree



Lookup

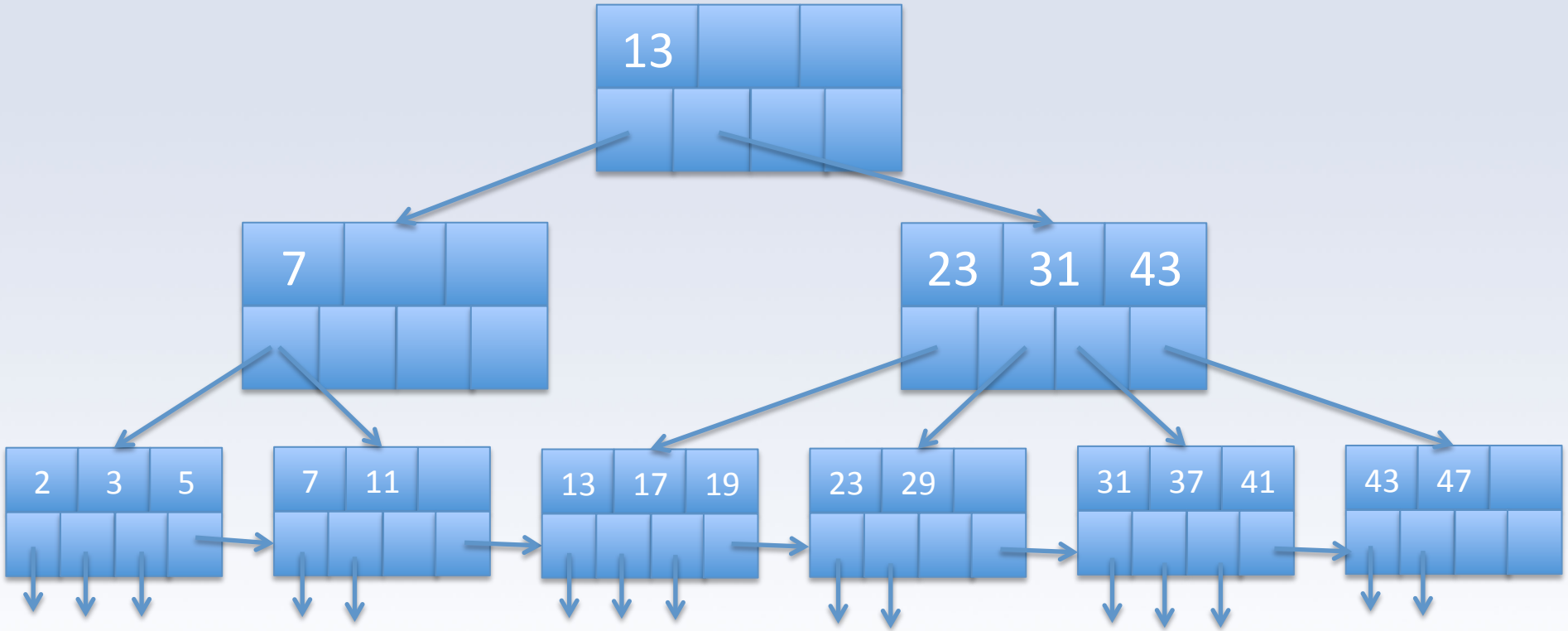
- To find a key K in the tree:
  - Start at root
  - Internal nodes: search keys to find range K belongs in and follow that pointer
  - Leaf nodes: search keys to find K and follow pointer

Lookup 40



illinois.edu

Lookup 17



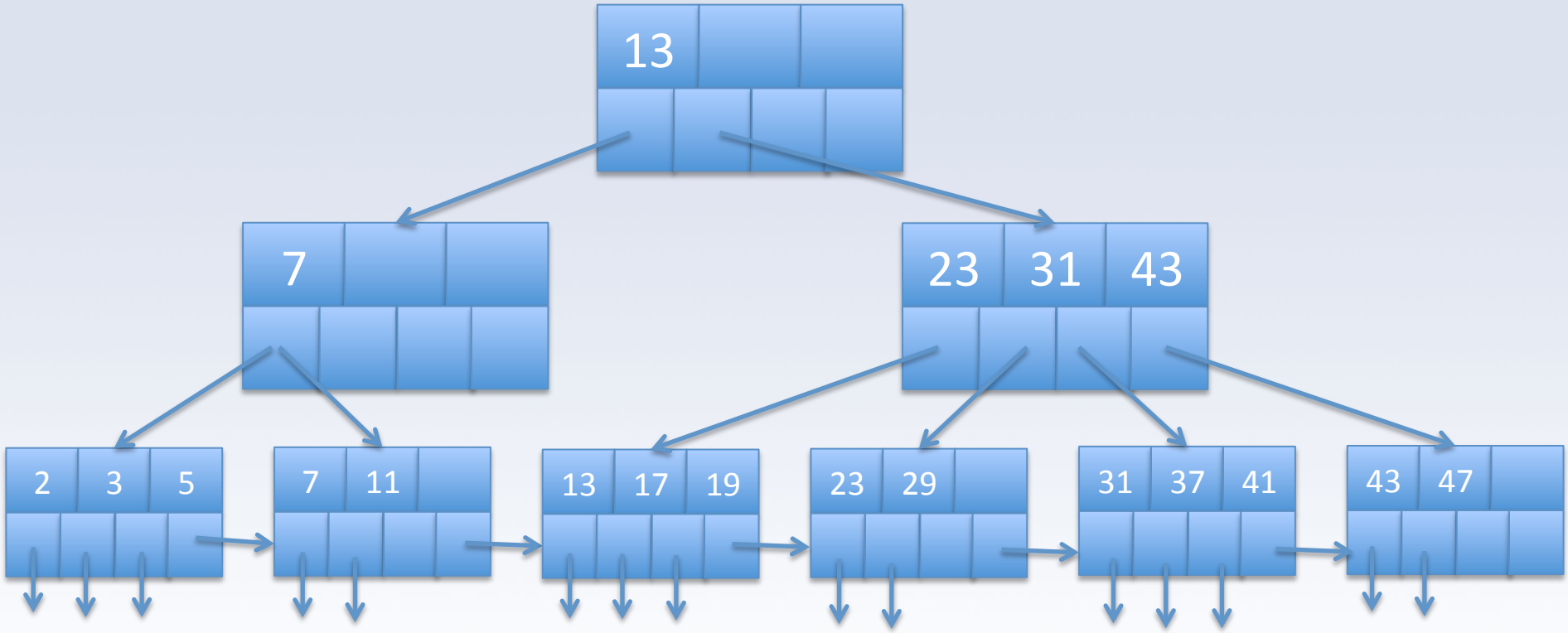
Range lookup

- What about queries like this

SELECT * FROM R

WHERE key>10 AND k<25;

Lookup (10,25)

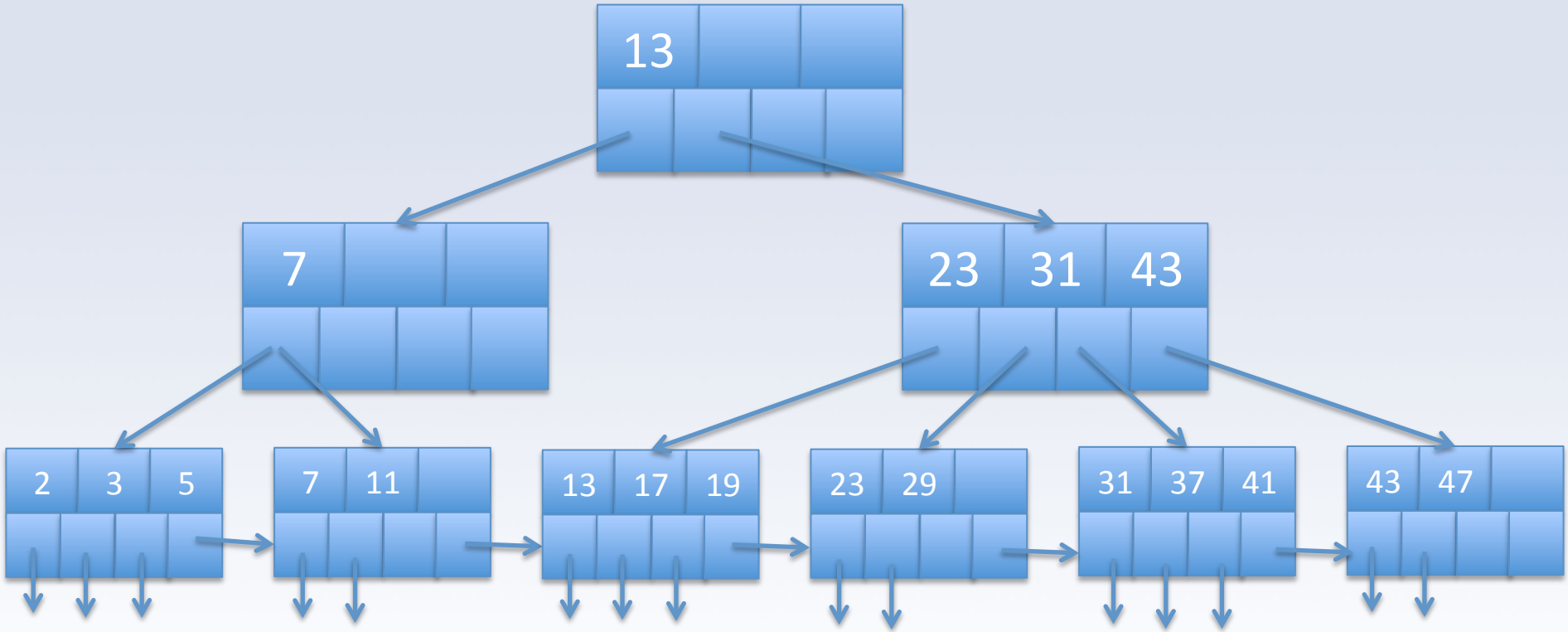


illinois.edu

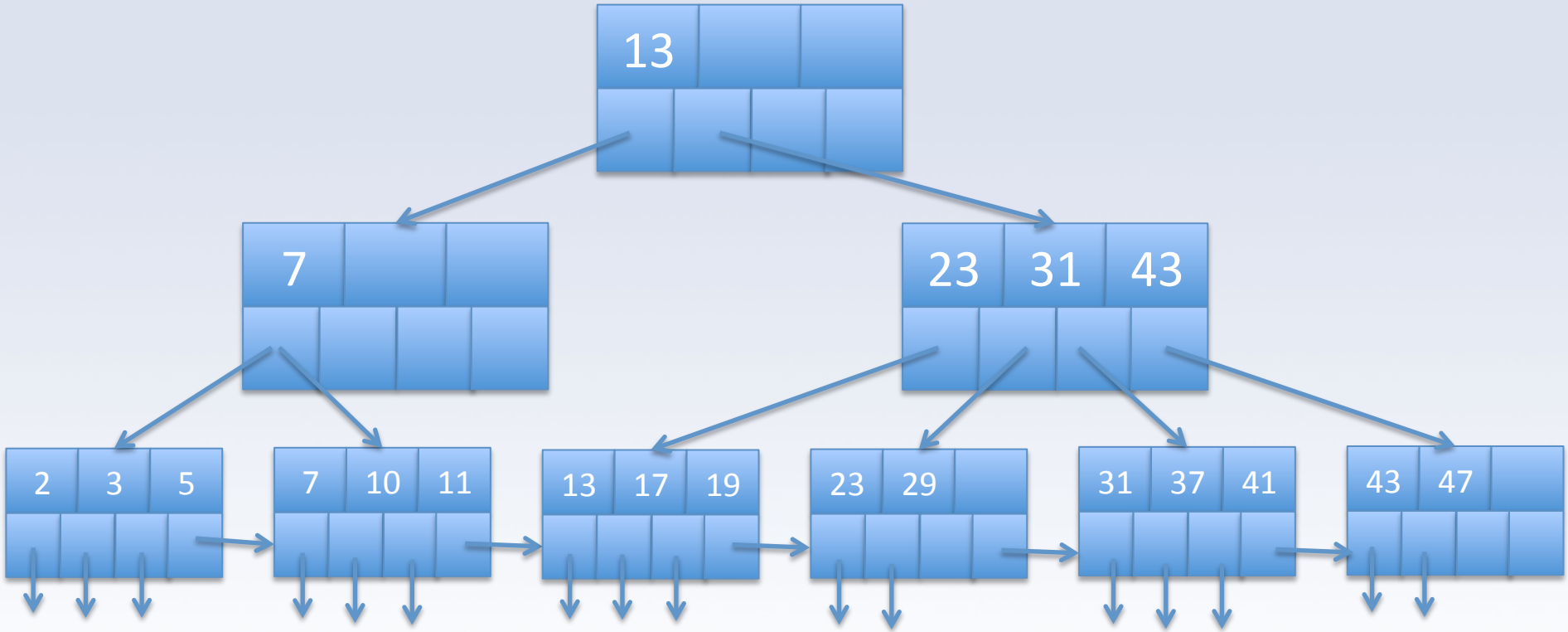
Insertion

- To insert K into a B tree
  - Lookup appropriate leaf node L for K
  - Insert K into L with parent P.
  - If overflow, create new leaf N. Move half to L and half to N. Add N's smallest key to P.
  - If P overflows, split it. Add middle key to P's parent and recurse.
  - If P is root, split it and create a new root.

Insert 10

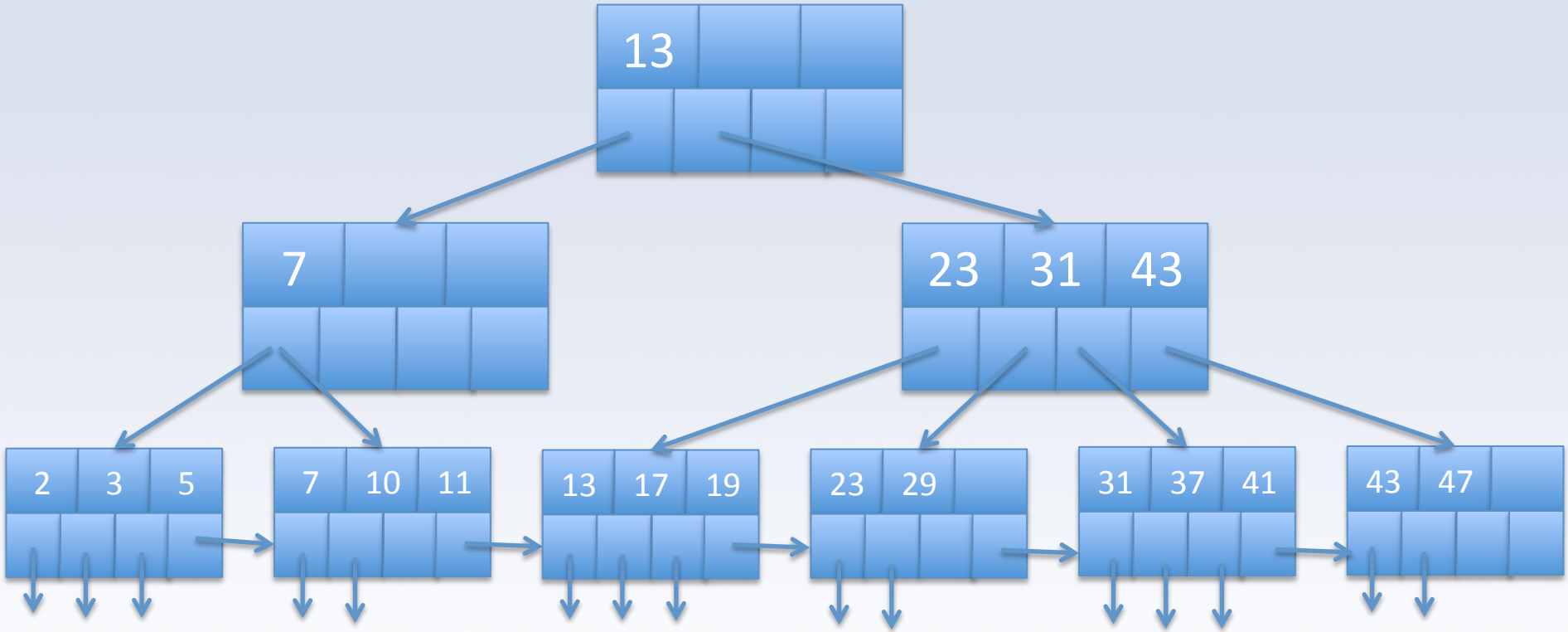


Insert 10

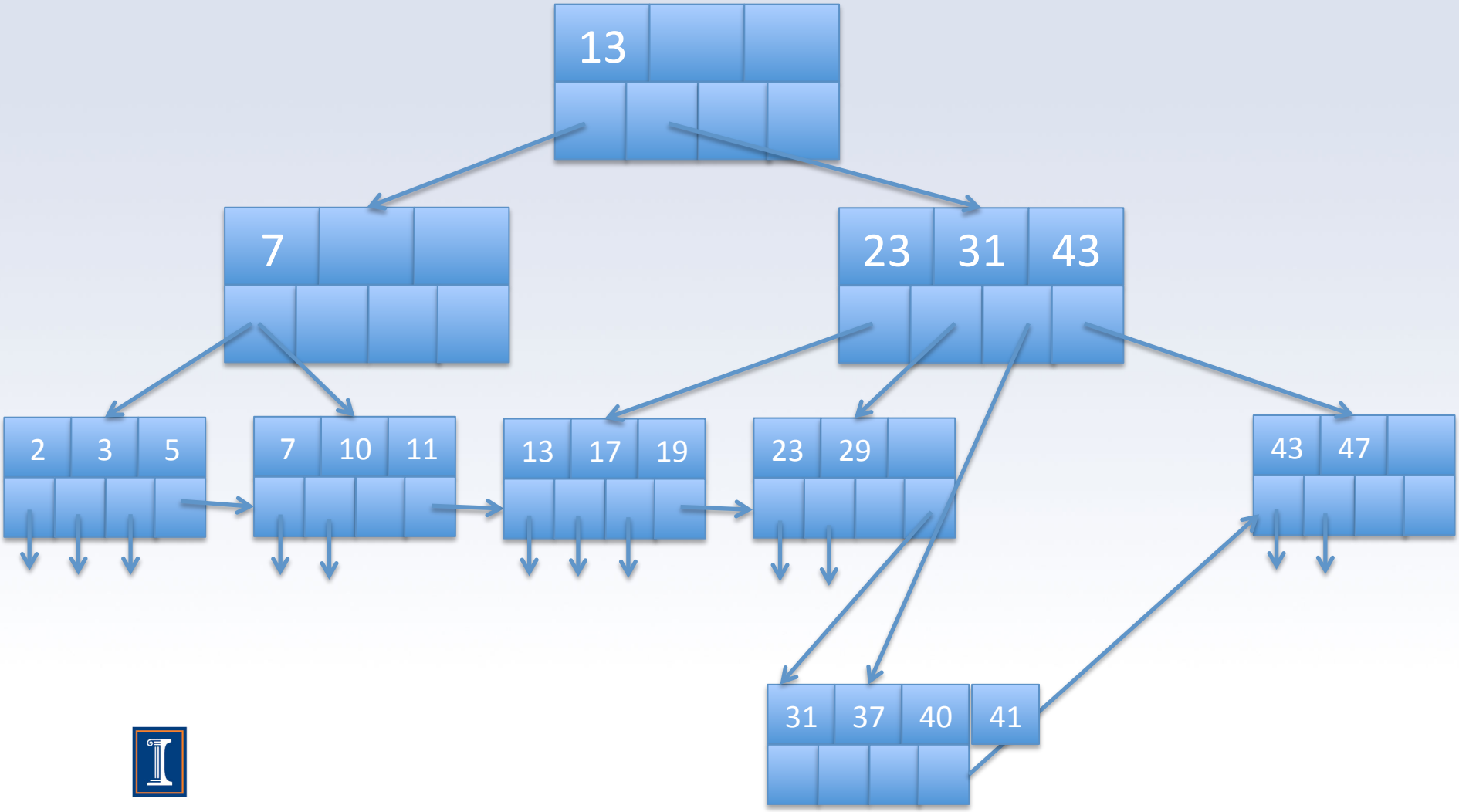


illinois.edu

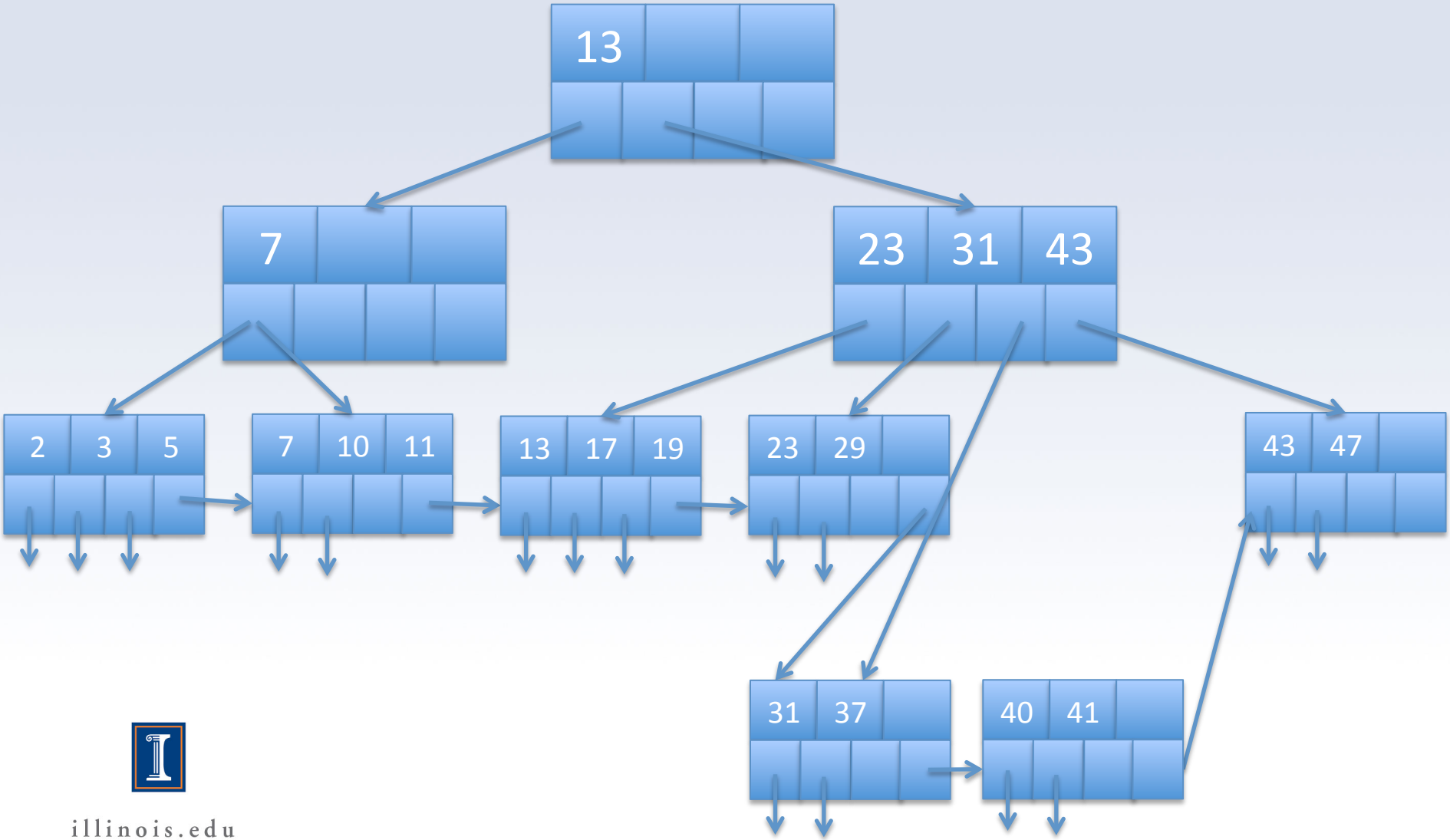
Insert 40



Insert 40

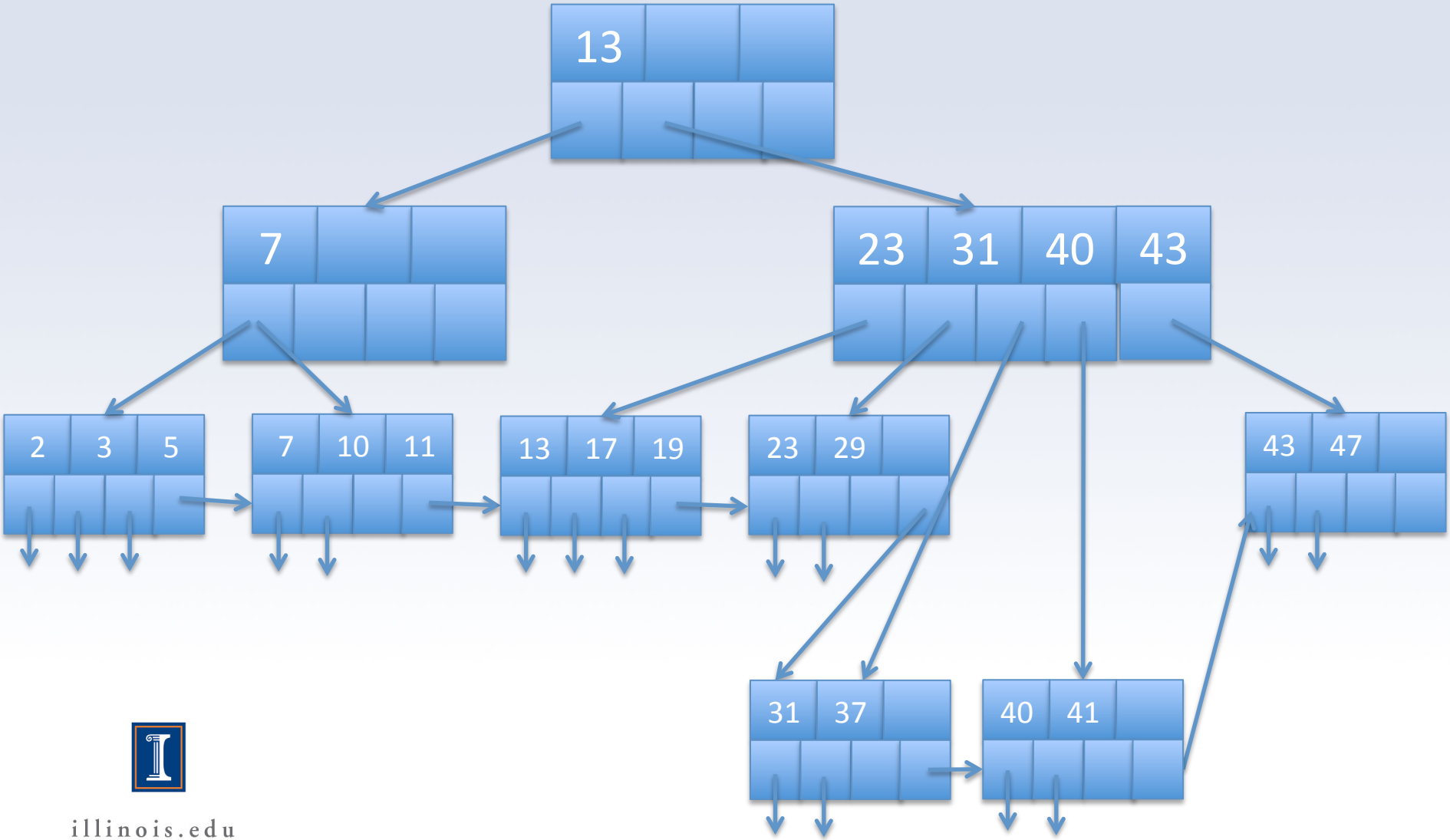


Insert 40

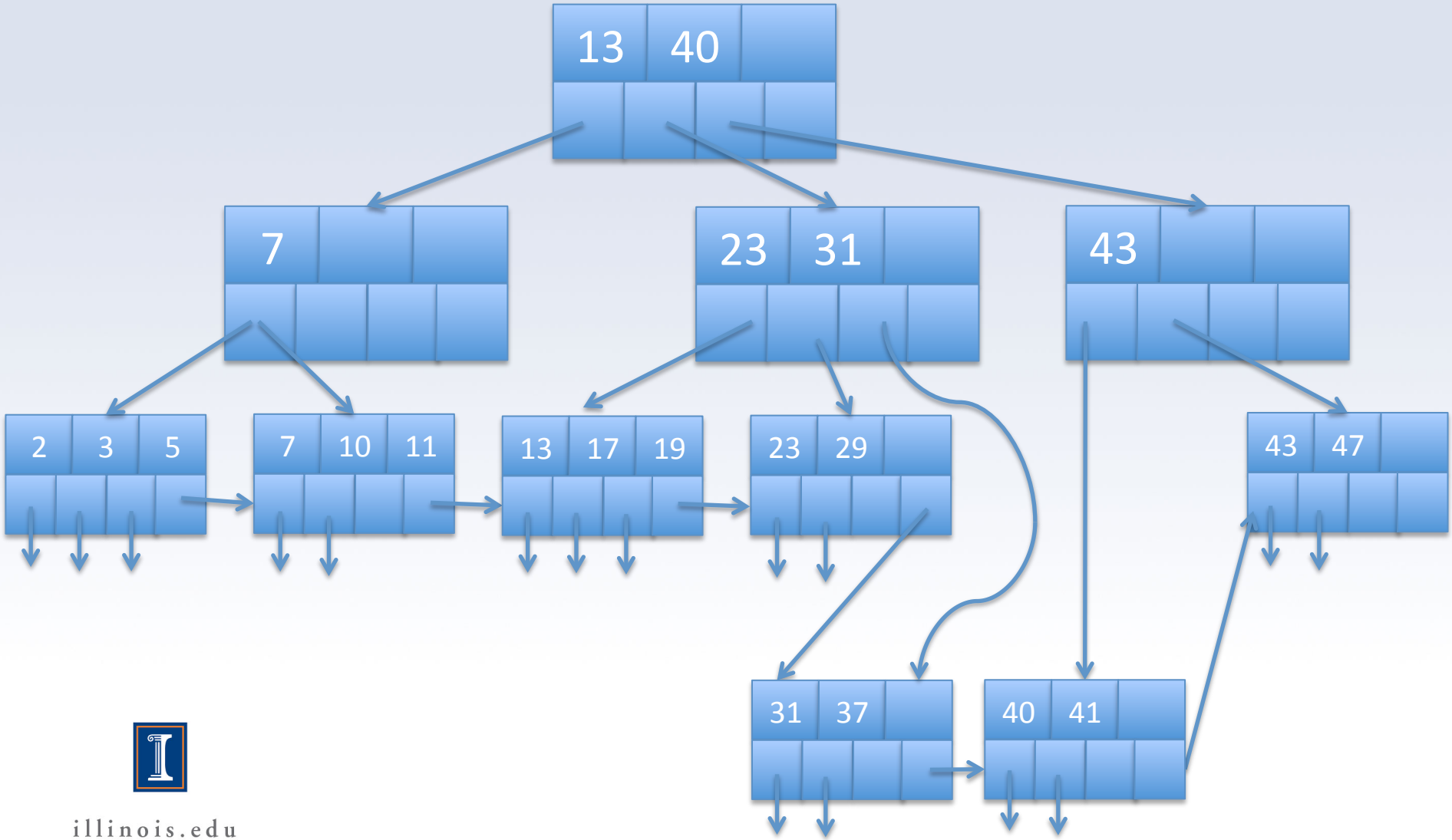


illinois.edu

Insert 40



Insert 40

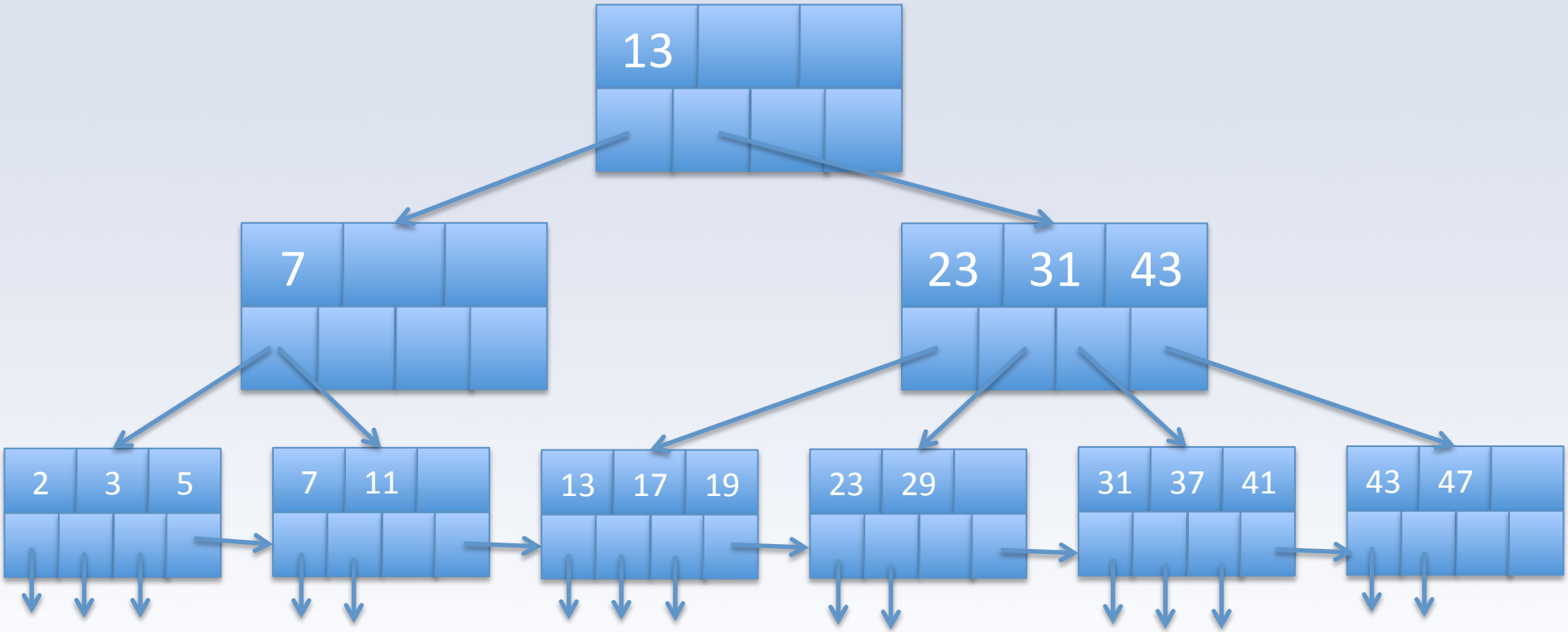


illinois.edu

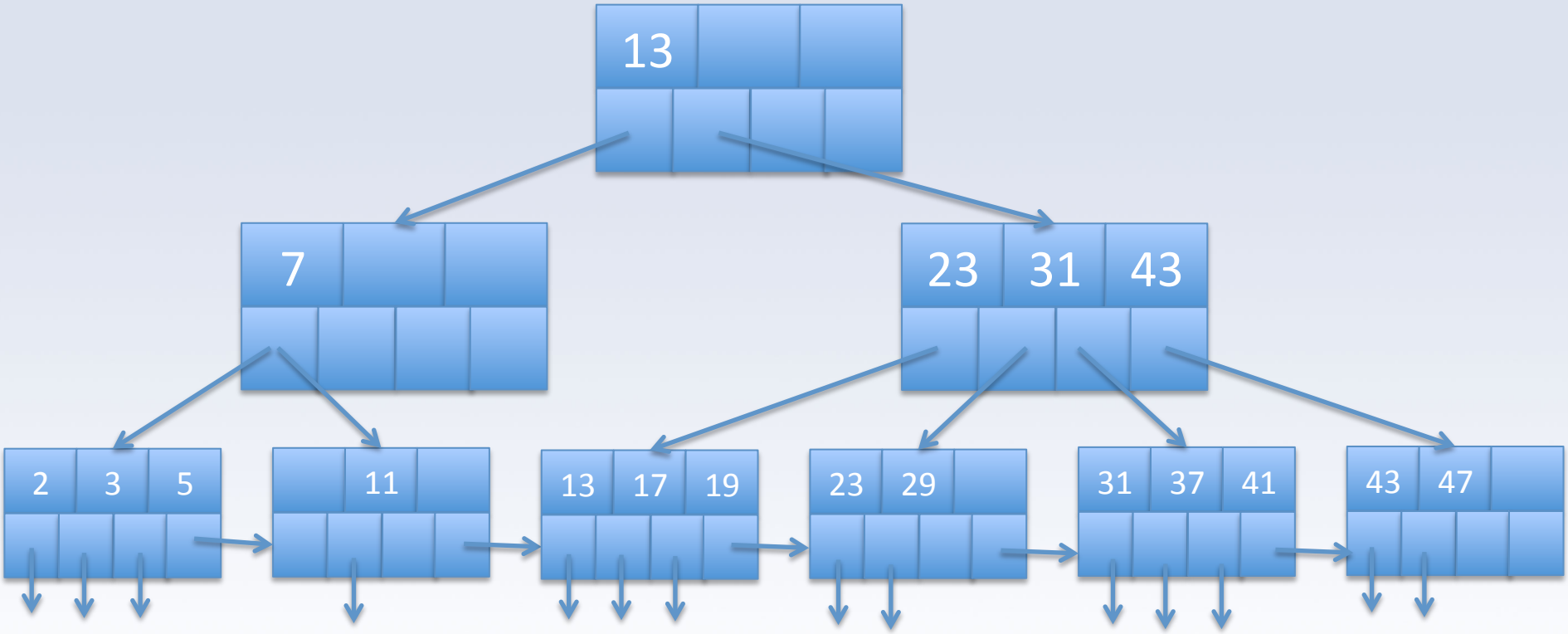
Deletion

- To delete K from a B-tree
  - Lookup leaf L where K belongs
  - Remove K from L
  - If L has too few entries
    - borrow from sibling if possible
    - if cannot borrow, merge with sibling
  - If merged, delete L from parent and recurse
  - Continue until no merges or root is deleted

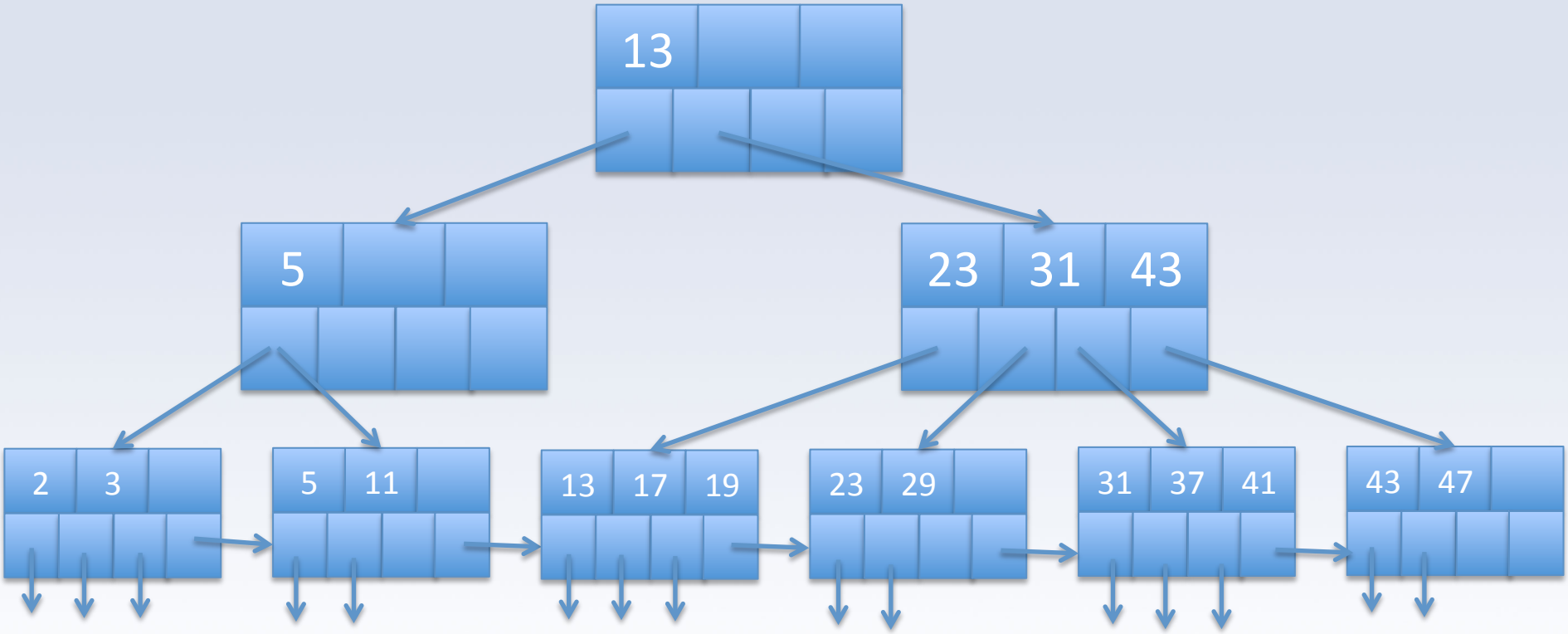
Delete 7



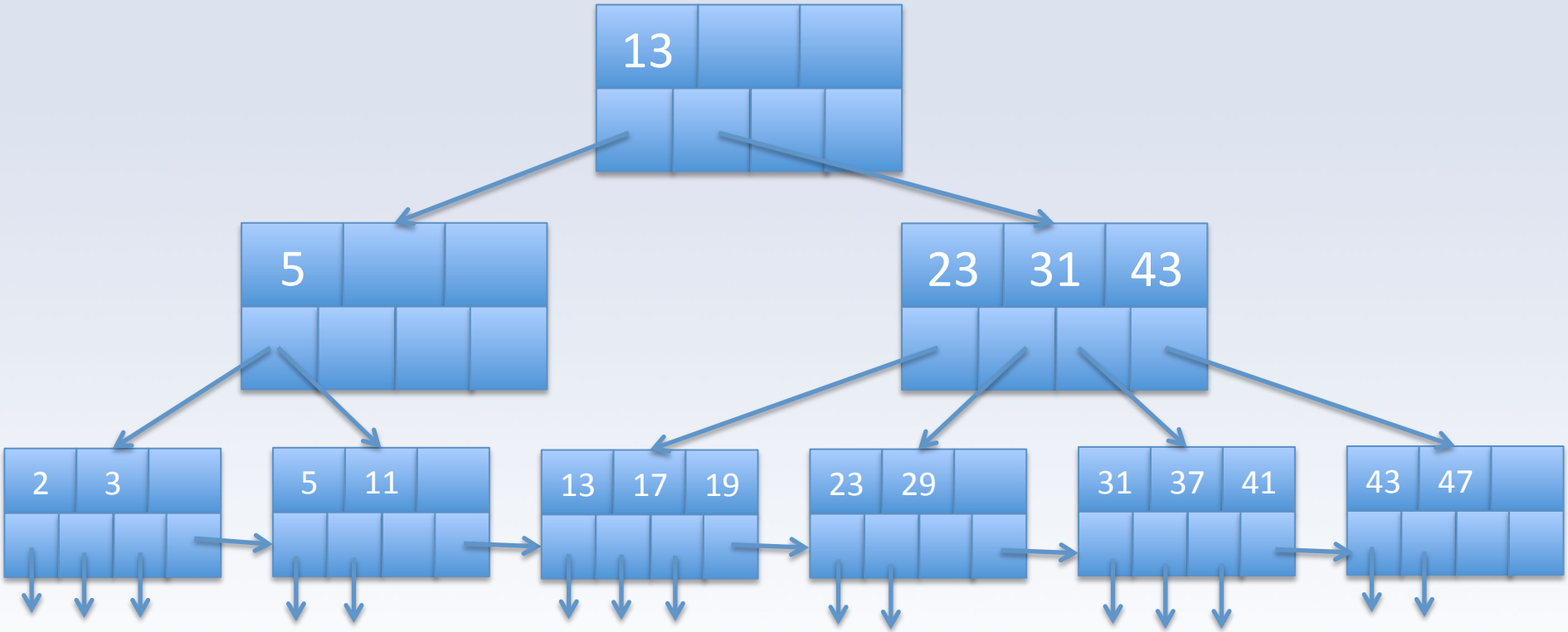
Delete 7



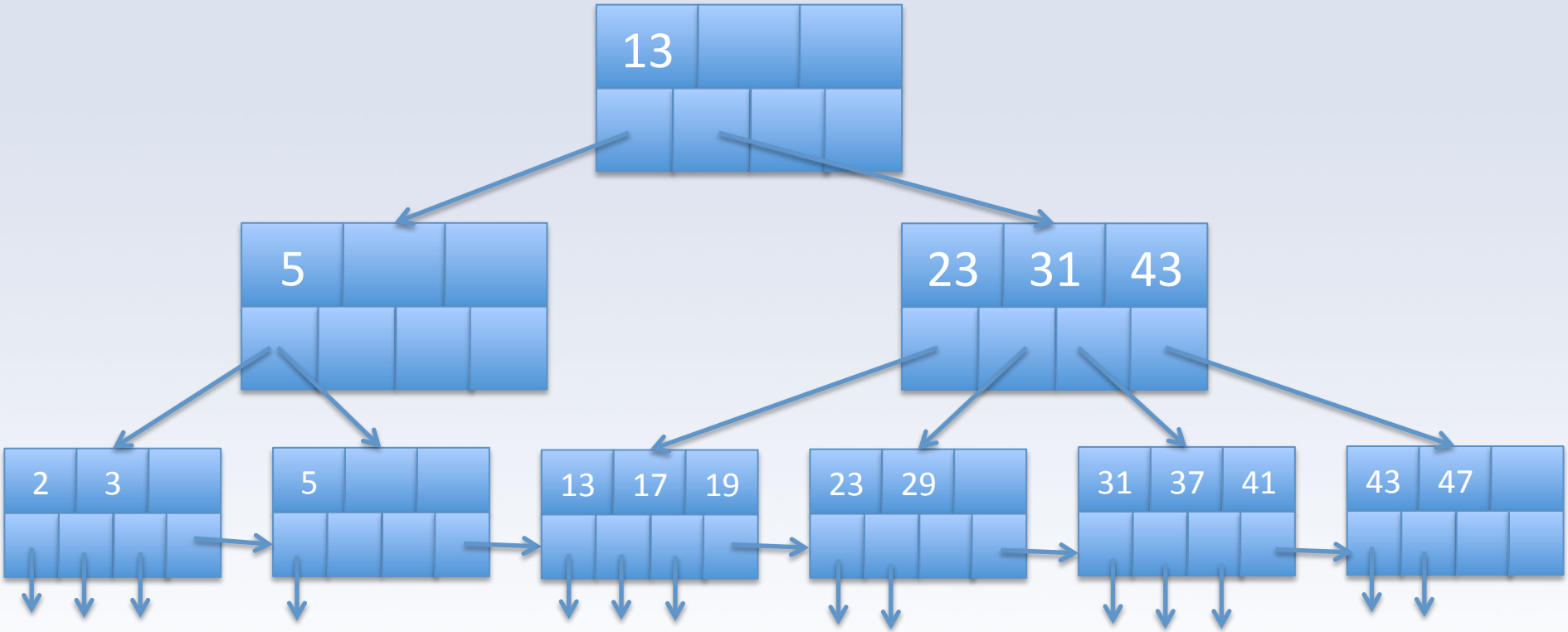
Delete 7



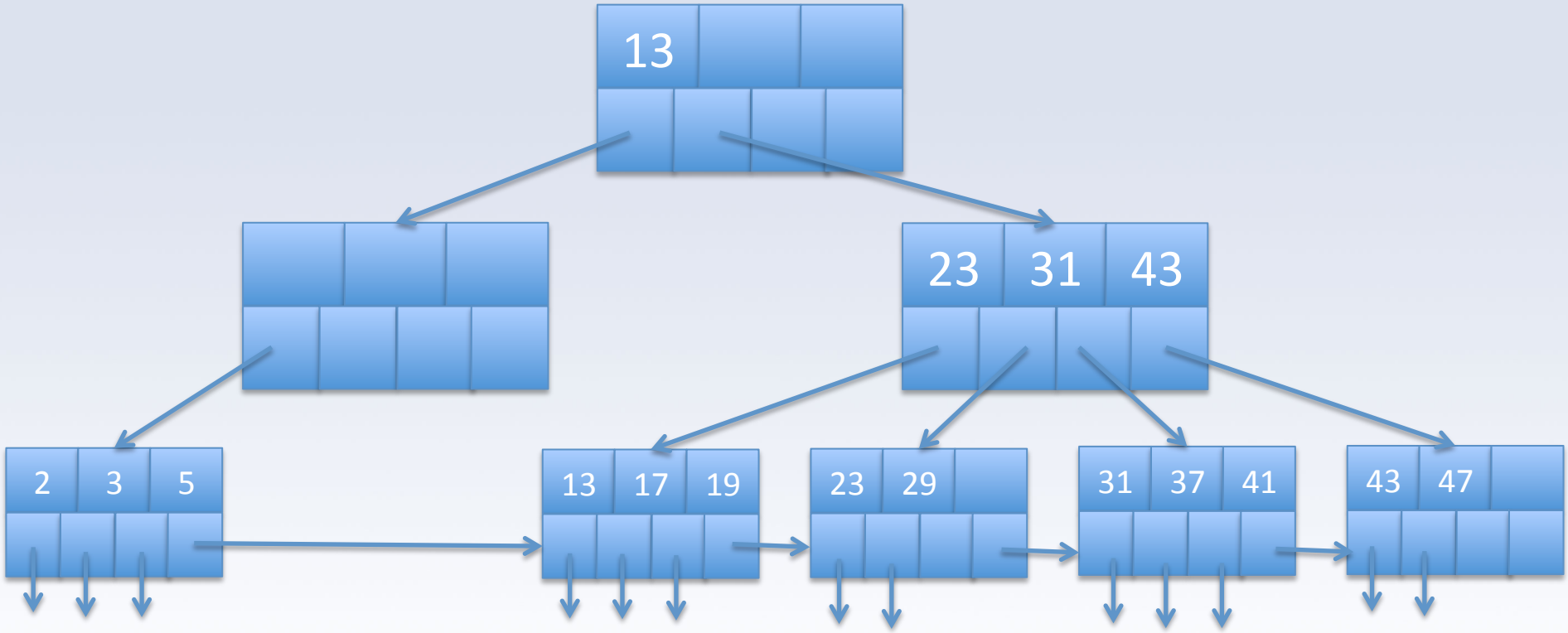
Delete 11



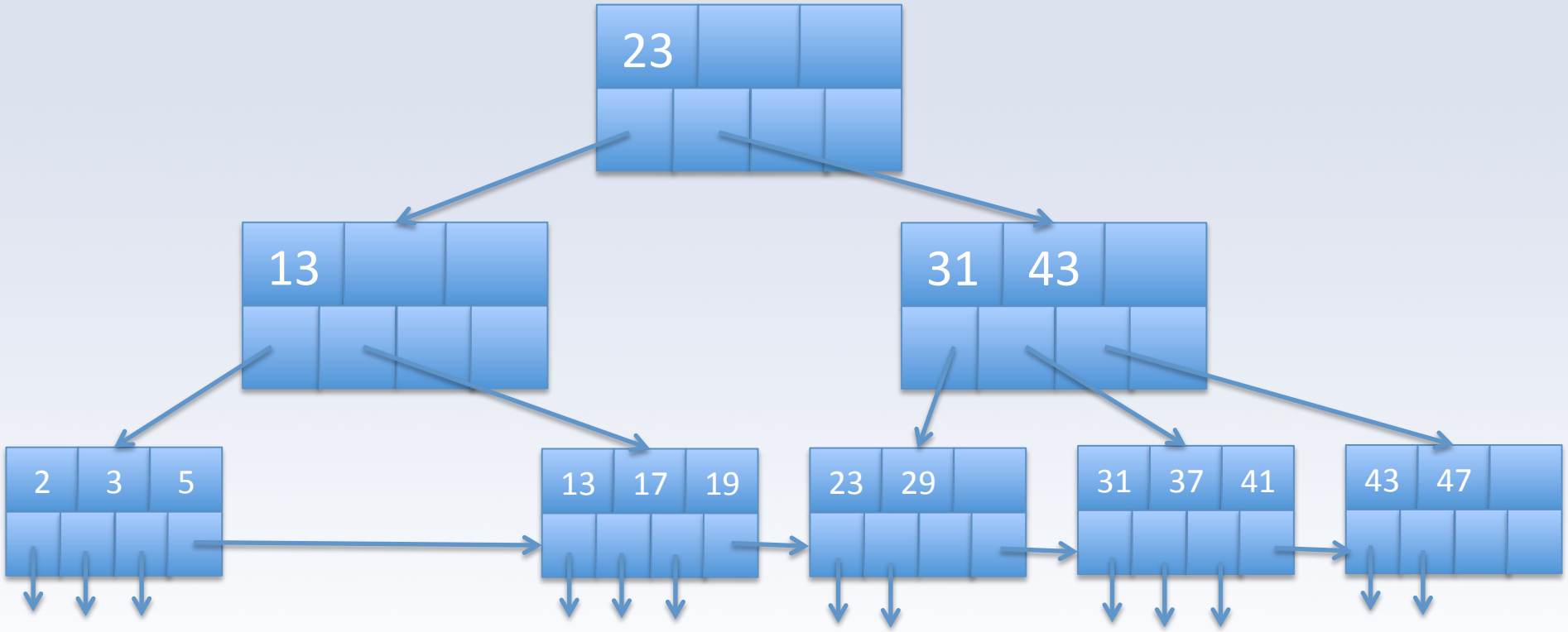
Delete 11



Delete 11

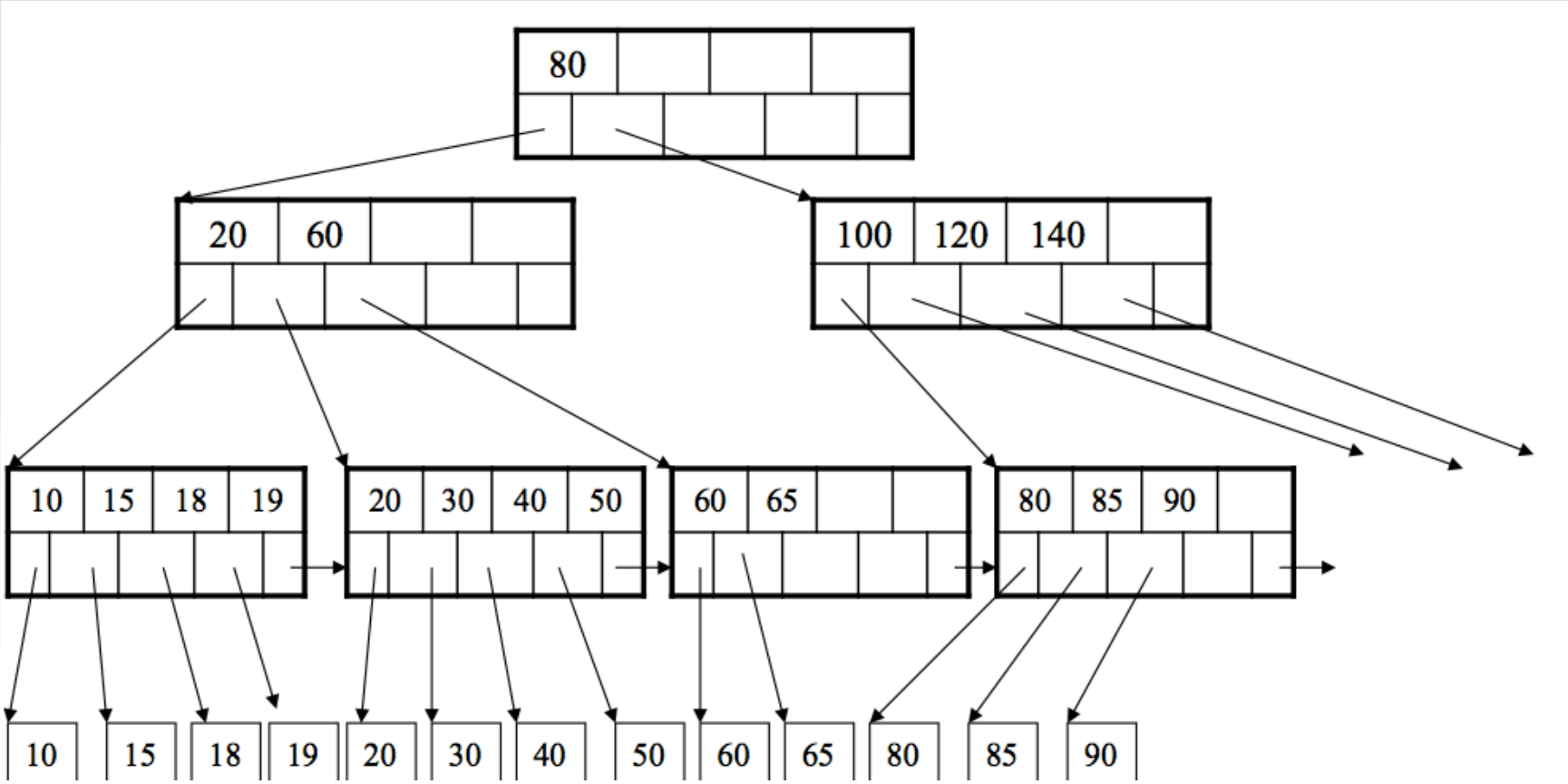


Delete 11

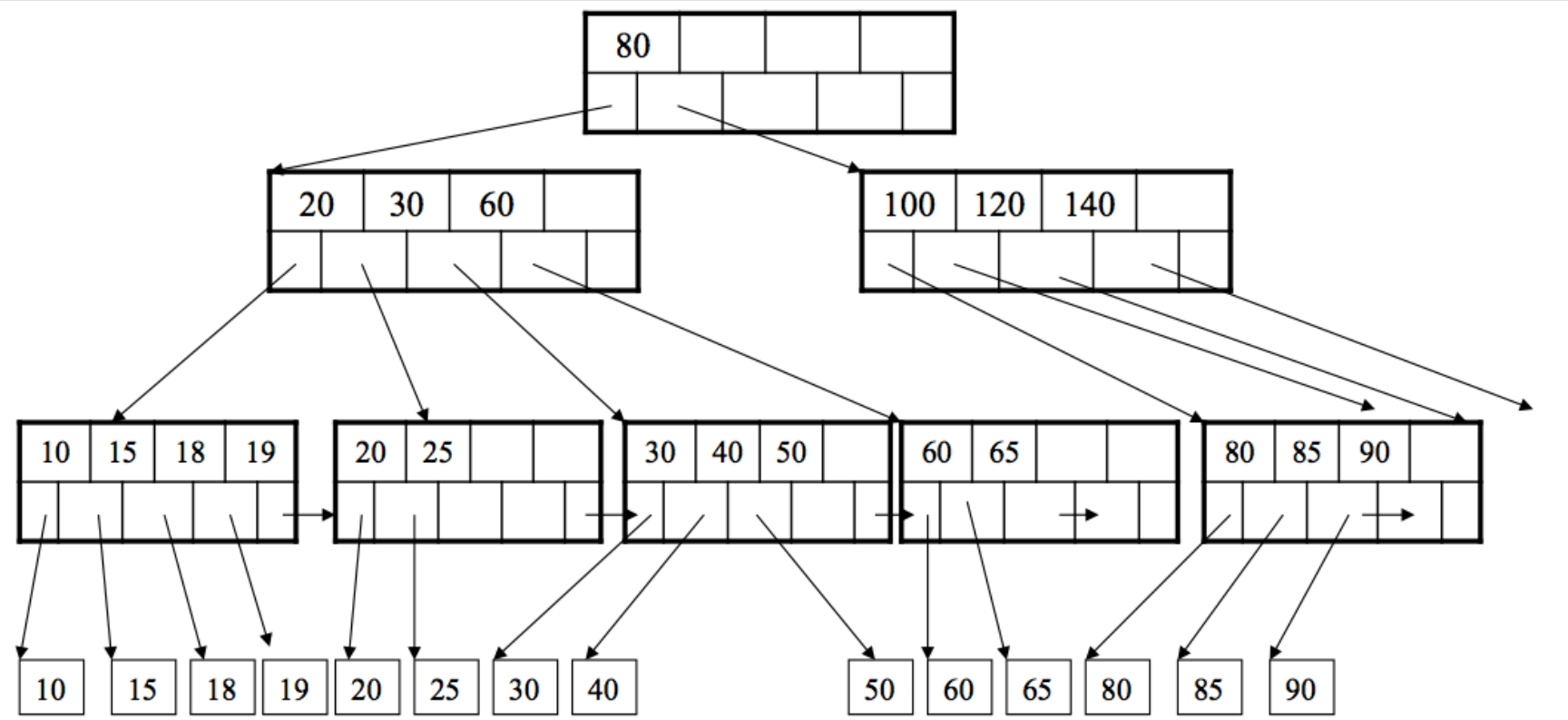


illinois.edu

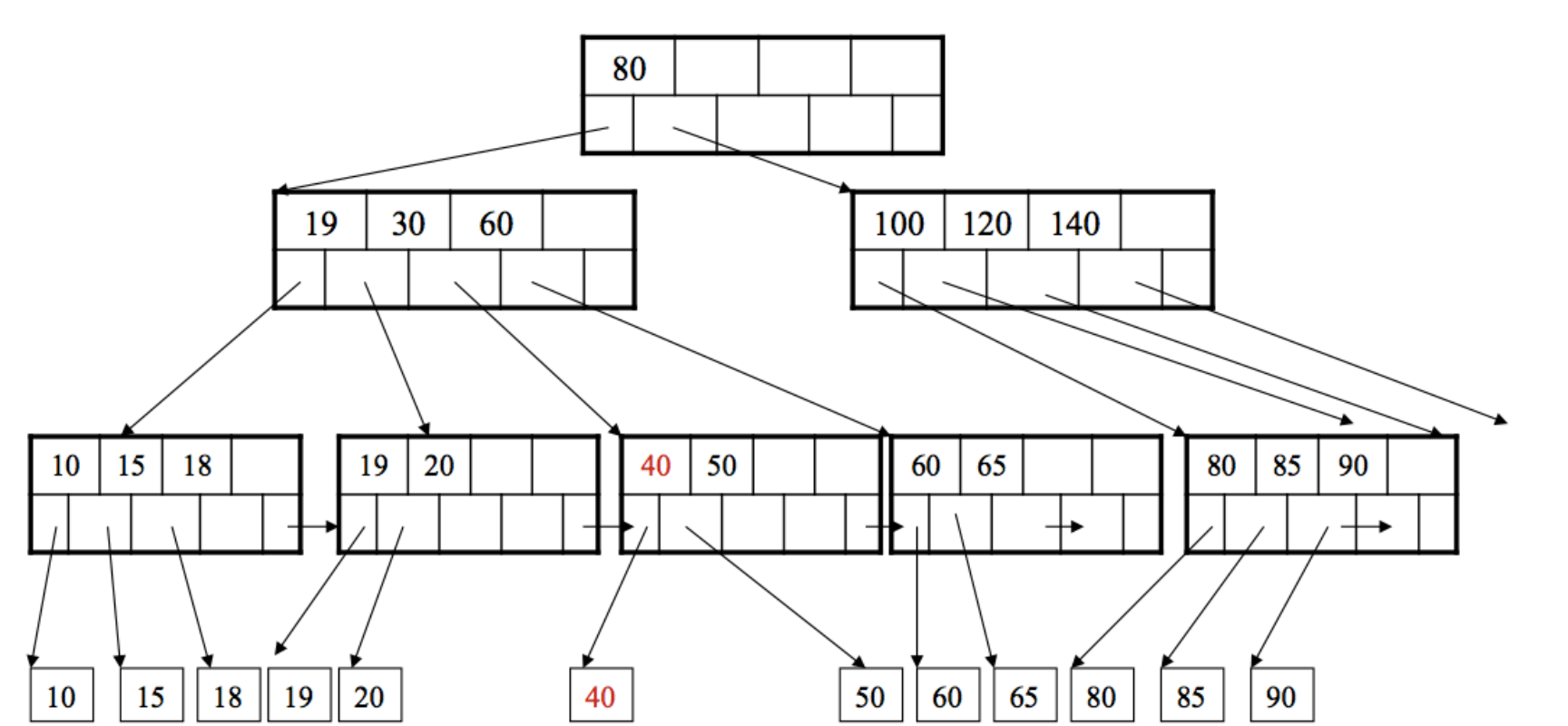
Exercise: Insert 25



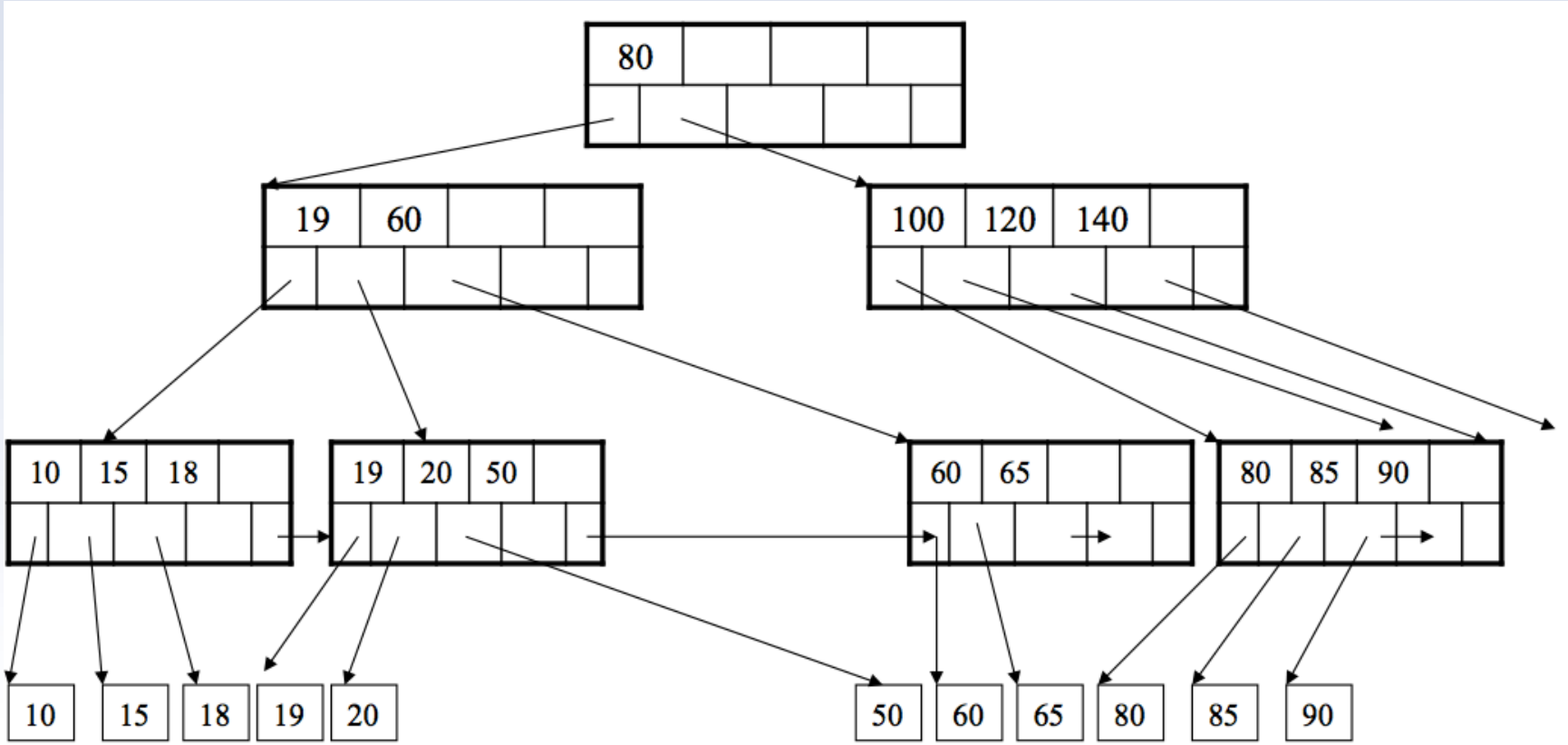
Solution


80
20 30 60
100 120 140
10 15 18 19
20 25
30 40 50
60 65
80 85 90
10 15 18 19 20 25 30 40 50 60 65 80 85 90

Exercise: Delete 40



Solution


80
19
60
100
120
140
10
15
18
19
20
50
60
65
80
85
90
10
15
18
19
20
50
60
65
80
85
90