

# Introduction to Database Systems

## CSE 344

### Lecture 10: Basics of Data Storage and Indexes

# Reminder

- HW3 is due next Wednesday

# Review

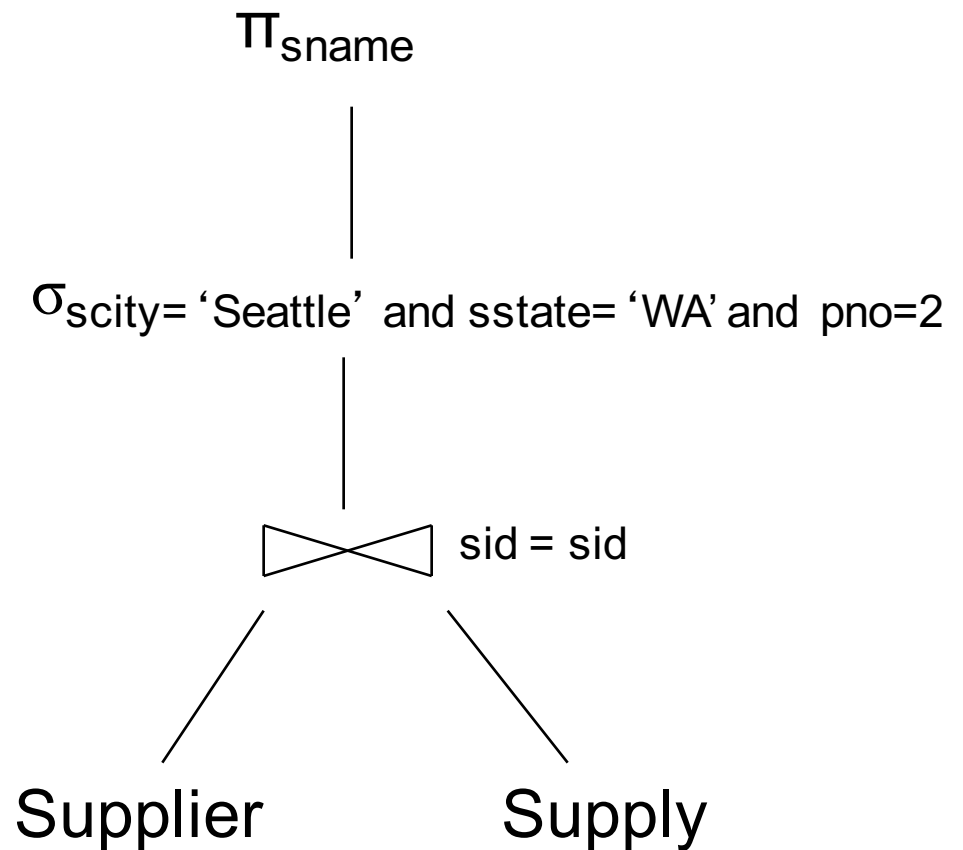
- Logical plans
- Physical plans
- Overview of query optimization and execution

Supplier(sid, sname, scity, sstate)  
Supply(sid, pno, quantity)

# Review: Relational Algebra

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
      and y.pno = 2
      and x.scity = 'Seattle'
      and x.sstate = 'WA'
```

Relational algebra expression is also called the “logical query plan”



Supplier(sid, sname, scity, sstate)  
Supply(sid, pno, quantity)

# Review: Physical Query Plan Example

(On the fly)

$\Pi_{\text{sname}}$

(On the fly)

$\sigma_{\text{scity} = \text{'Seattle'} \text{ and } \text{sstate} = \text{'WA'} \text{ and } \text{pno} = 2}$

(Nested loop)

sid = sid

Supplier  
(File scan)

Supply  
(File scan)

A physical query plan is a logical query plan annotated with physical implementation details

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
      and y.pno = 2
      and x.scity = 'Seattle'
      and x.sstate = 'WA'
```

# Query Optimization Problem

- For each SQL query... many logical plans
- For each logical plan... many physical plans
- How do find a fast physical plan?
  - Will discuss in a few lectures
  - First we need to understand how data is stored on the disk, and how other data structures can be used to optimize query execution

# Query Performance

- My database application is too slow... why?
- One of the queries is very slow... why?
- To understand performance, we need to understand:
  - How is data organized on disk
  - How to estimate query costs
  - In this course we will focus on **disk-based DBMSs**

# Data Storage

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- DBMSs store data in **files**
- Most common organization is row-wise storage
- On disk, a file is split into **blocks**
- Each block contains a set of tuples

10	Tom	Hanks
20	Amy	Hanks

block 1

50	...	...
200	...	

block 2

220		
240		

block 3

420		
800		

In the example, we have **4 blocks** with 2 tuples each



# Data File Types

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

The data file can be one of:

- **Heap file**
  - Unsorted
- **Sequential file**
  - Sorted according to some attribute(s) called key

# Data File Types

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

The data file can be one of:

- **Heap file**
  - Unsorted
- **Sequential file**
  - Sorted according to some attribute(s) called key

Note: key here means something different from primary key: it just means that we order the file according to that attribute. In our example we ordered by **ID**. Might as well order by **fName**, if that seems a better idea for the applications running on our database.

# Index

- An **additional** file, that allows fast access to records in the data file given a search key

# Index

- An **additional** file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - The key = an attribute value (e.g., student ID or name)
  - The value = a pointer to the record

# Index

- An **additional** file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - The key = an attribute value (e.g., student ID or name)
  - The value = a pointer to the record
- Could have many indexes for one table

Key = means here search key

This



Is Not A Key

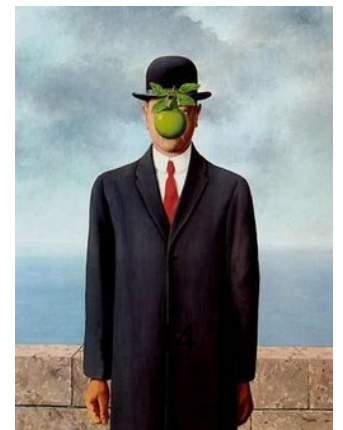
Different keys:

- **Primary key** – uniquely identifies a tuple
- **Key of the sequential file** – how the data file is sorted, if at all
- **Index key** – how the index is organized



*This is not a pipe.*

CSE 344 - Winter 2017



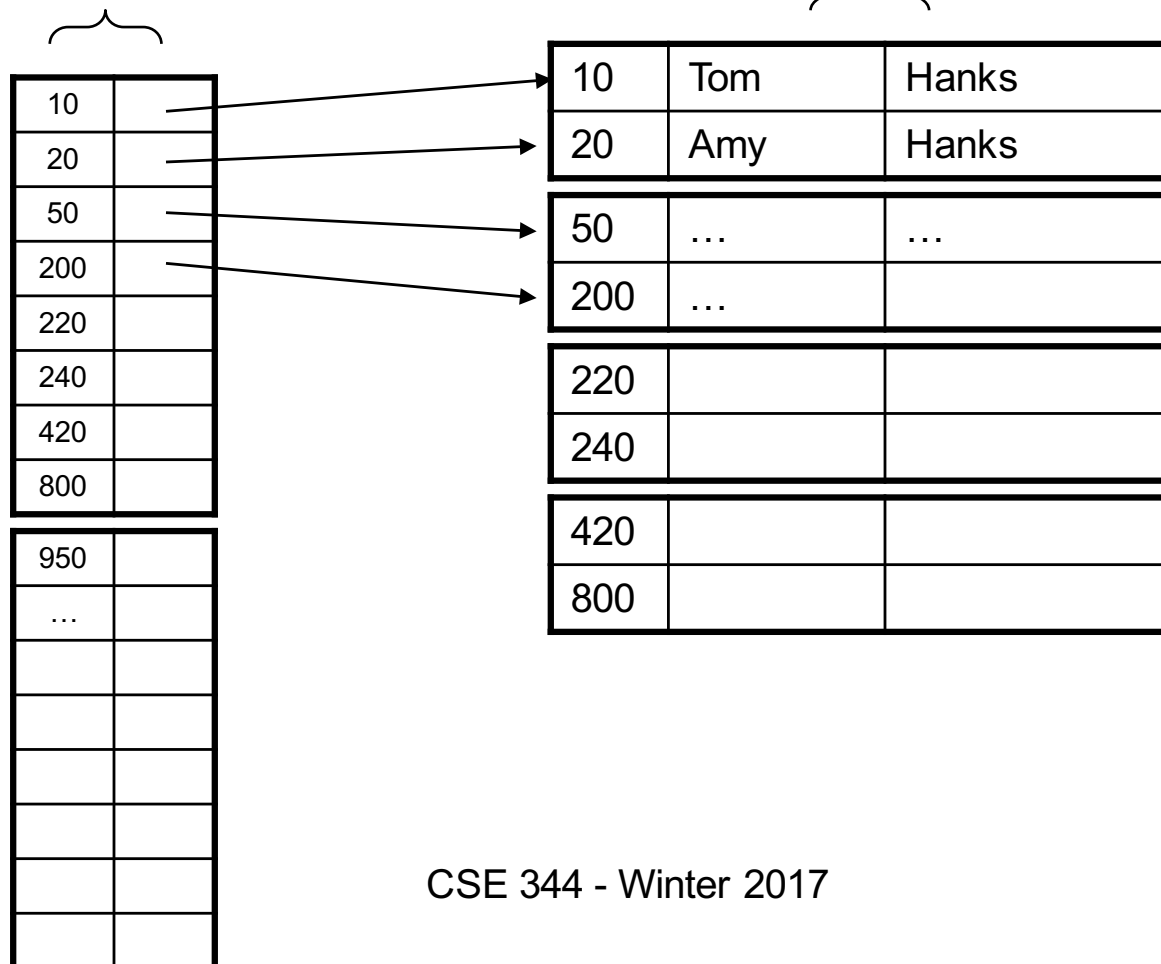
# Example 1: Index on ID

**Student**

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

**Index Student\_ID on Student.ID**

**Data File Student**



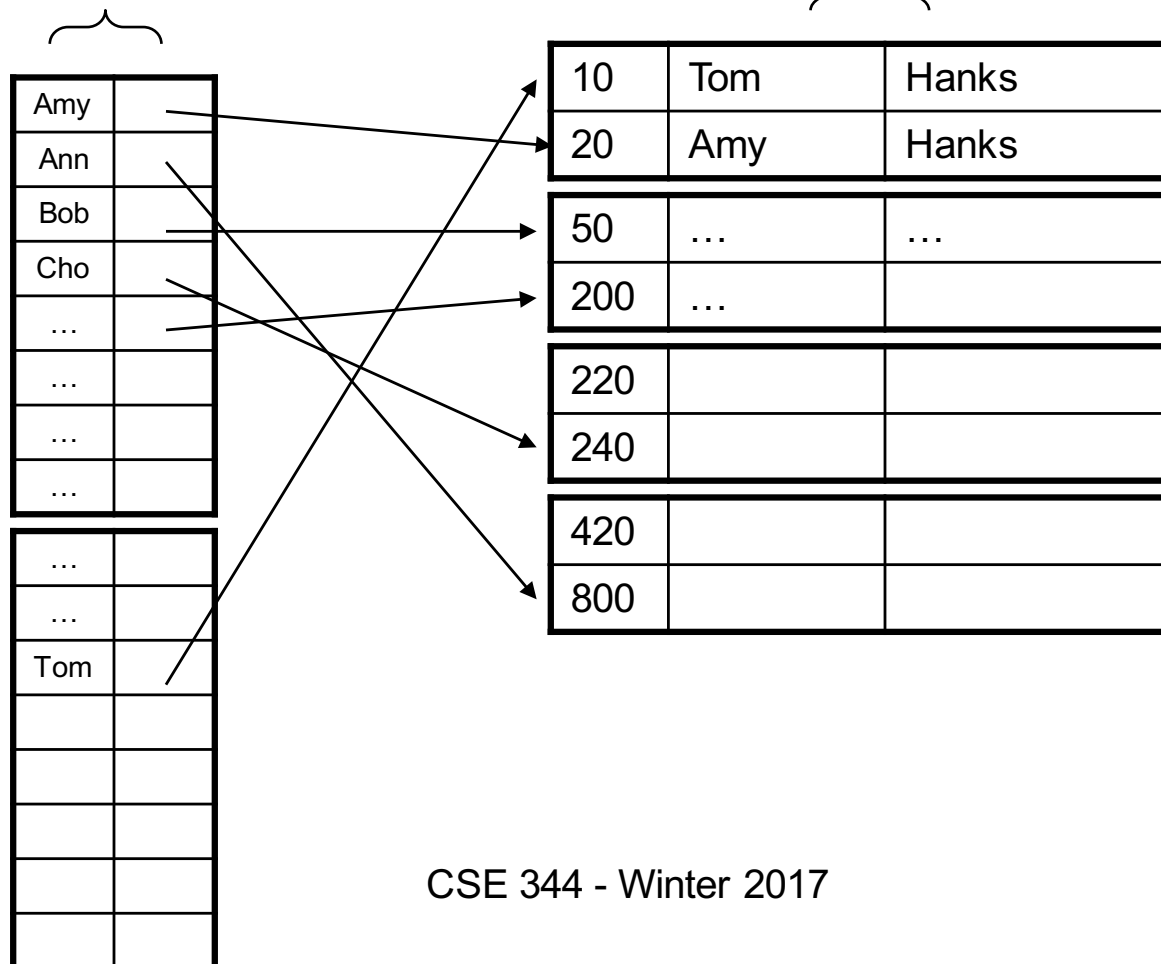
# Example 2: Index on fName

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

Index **Student\_fName**  
on **Student.fName**

Data File **Student**





# Index Organization

We need a way to represent indexes after loading into memory

Several ways to do this:

- Hash table
- B+ trees – most popular
  - They are search trees, but they are not binary instead have higher fanout
  - Will discuss them briefly next
- Specialized indexes: bit maps, R-trees, inverted index

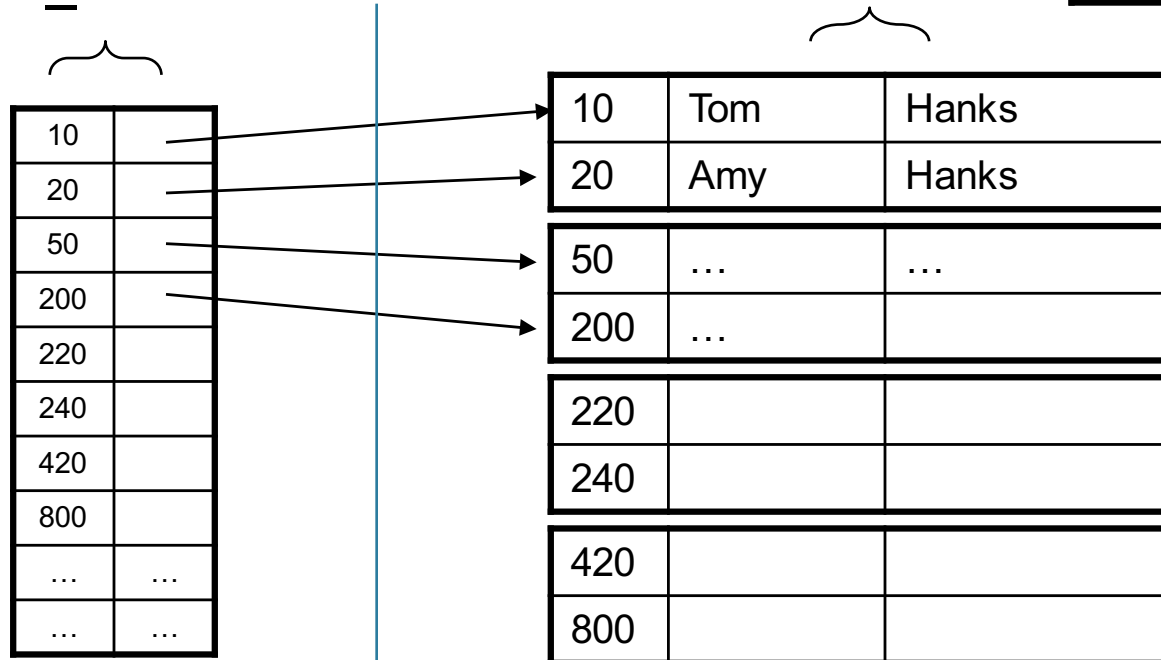
# Hash table example

**Student**

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

**Index Student\_ID on Student.ID**

**Data File Student**



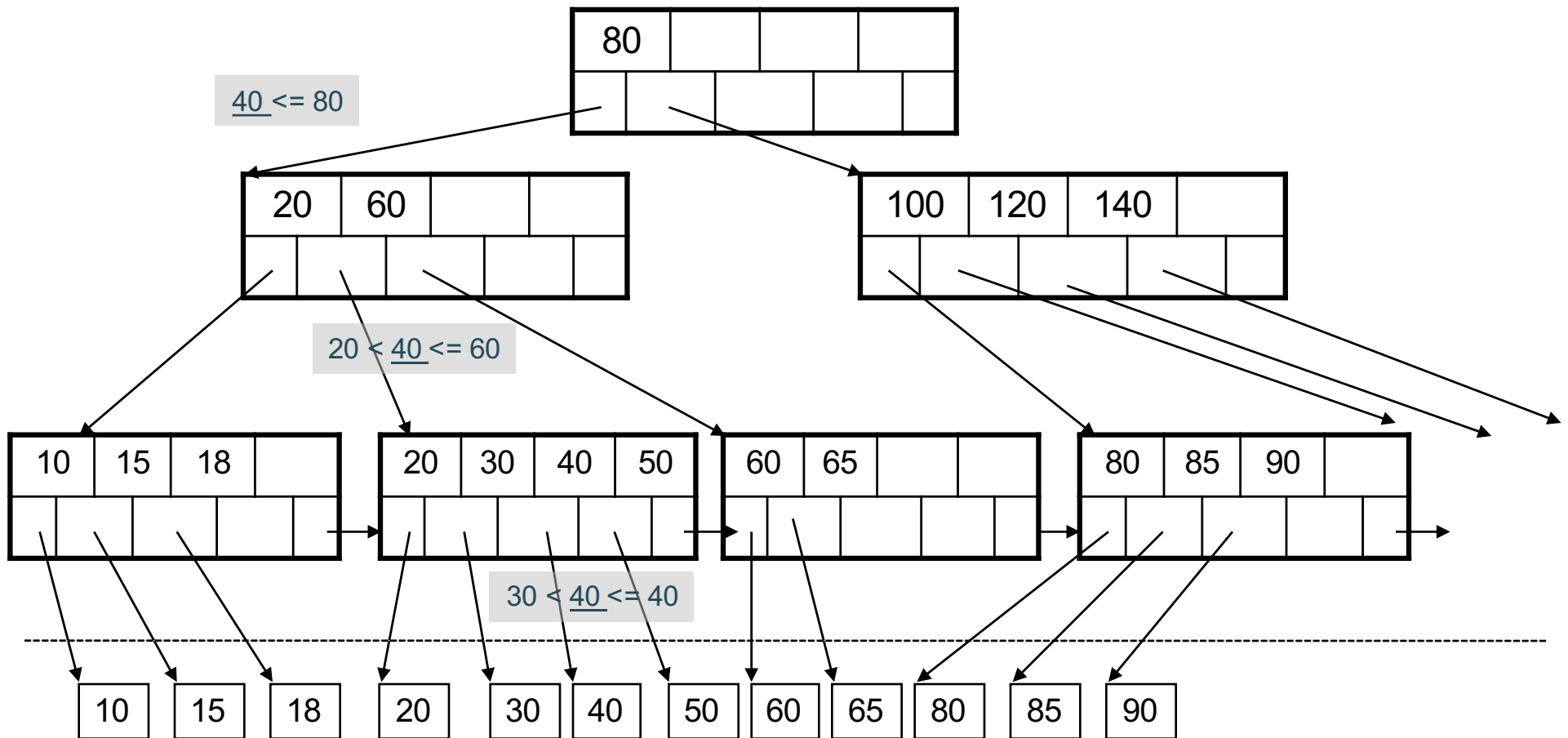
**Index File  
(in memory)**

**Data file  
(on disk)**

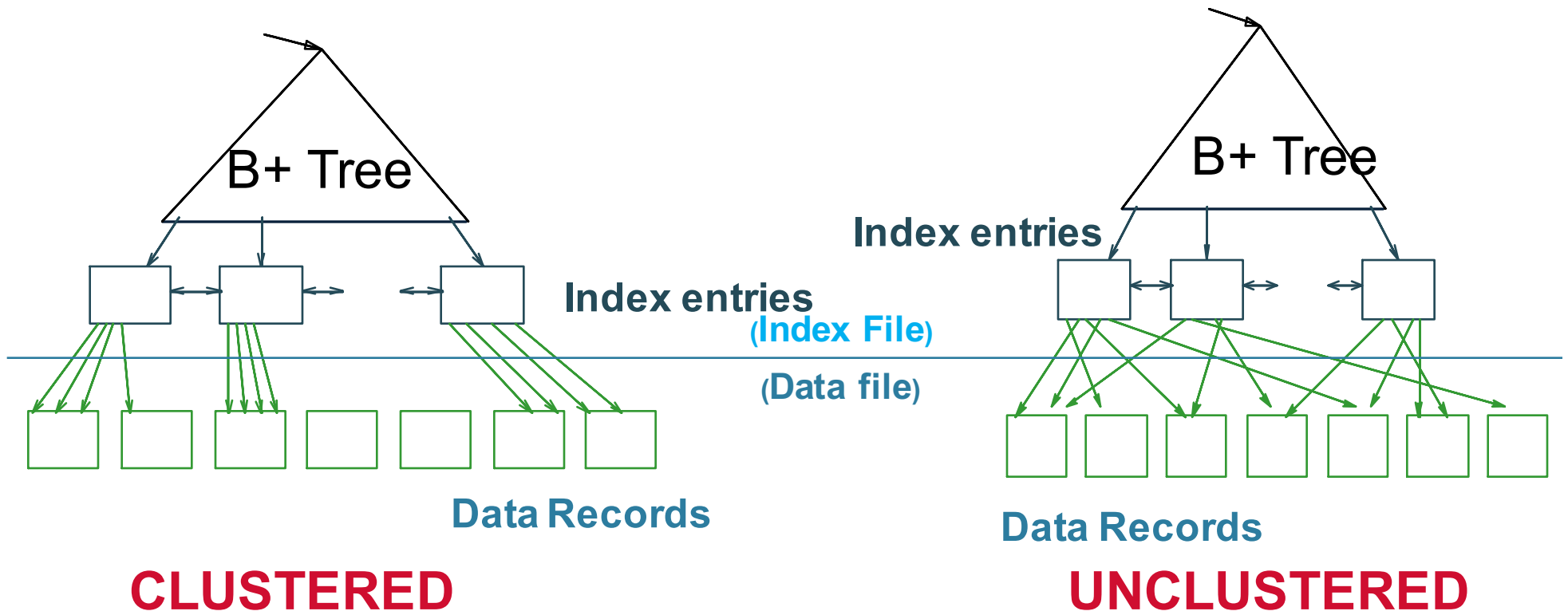
# B+ Tree Index by Example

$d = 2$

Find the key 40



# Clustered vs Unclustered



Every table can have **only one** clustered and **many** unclustered indexes  
Why?

# Index Classification

- **Clustered/unclustered**
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data

# Index Classification

- **Clustered/unclustered**
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data
- **Primary/secondary**
  - Meaning 1:
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2: means the same as clustered/unclustered

# Index Classification

- **Clustered/unclustered**
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data
- **Primary/secondary**
  - Meaning 1:
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2: means the same as clustered/unclustered
- **Organization** B+ tree or Hash table

# Scanning a Data File

- Disks are mechanical devices!
  - Technology from the 60s; density much higher now
- Read only at the rotation speed!
- Consequence:  
Sequential scan is MUCH FASTER than random reads
  - **Good**: read blocks 1,2,3,4,5,...
  - **Bad**: read blocks 2342, 11, 321,9, ...
- **Rule of thumb**:
  - Random reading 1-2% of the file  $\approx$  sequential scanning the entire file; this is decreasing over time (because of increased density of disks)
- Solid state (SSD): \$\$\$ expensive; put indexes, other “hot” data there, not enough room for everything (NO LONGER TRUE)





```
SELECT *  
FROM Student x, Takes y  
WHERE x.ID=y.studentID AND y.courseID > 300
```

# Example

```
for y in Takes  
  if courseID > 300 then  
    for x in Student  
      if x.ID=y.studentID  
        output *
```

Assume the database has indexes on these attributes:

- **index\_takes\_courseID** = index on Takes.courseID
- **index\_student\_ID** = index on Student.ID

```
for y in index_Takes_courseID where y.courseID > 300  
  for x in Student where x.ID = y.studentID  
    output *
```

```
SELECT *  
FROM Student x, Takes y  
WHERE x.ID=y.studentID AND y.courseID > 300
```

# Example

```
for y in Takes  
  if courseID > 300 then  
    for x in Student  
      if x.ID=y.studentID  
        output *
```

Assume the database has indexes on these attributes:

- **index\_takes\_courseID** = index on Takes.courseID
- **index\_student\_ID** = index on Student.ID

Index selection

```
for y in index_Takes_courseID where y.courseID > 300  
  for x in Student where x.ID = y.studentID  
    output *
```

```
SELECT *  
FROM Student x, Takes y  
WHERE x.ID=y.studentID AND y.courseID > 300
```

# Example

```
for y in Takes  
  if courseID > 300 then  
    for x in Student  
      if x.ID=y.studentID  
        output *
```

Assume the database has indexes on these attributes:

- **index\_takes\_courseID** = index on Takes.courseID
- **index\_student\_ID** = index on Student.ID

Index selection

```
for y in index_Takes_courseID where y.courseID > 300  
  for x in Student where x.ID = y.studentID  
    output *
```

Index join

# Getting Practical: Creating Indexes in SQL

```
CREATE TABLE V(M int, N varchar(20), P int);
```

```
CREATE INDEX V1 ON V(N)
```

```
CREATE INDEX V2 ON V(P, M)
```

```
CREATE INDEX V3 ON V(M, N)
```

```
CREATE UNIQUE INDEX V4 ON V(N)
```

```
CREATE CLUSTERED INDEX V5 ON V(N)
```

# Getting Practical: Creating Indexes in SQL

```
CREATE TABLE V(M int, N varchar(20), P int);
```

```
CREATE INDEX V1 ON V(N)
```

```
CREATE INDEX V2 ON V(P, M)
```

What does this mean?

```
CREATE INDEX V3 ON V(M, N)
```

```
CREATE UNIQUE INDEX V4 ON V(N)
```

```
CREATE CLUSTERED INDEX V5 ON V(N)
```

# Getting Practical: Creating Indexes in SQL

```
CREATE TABLE V(M int, N varchar(20), P int);
```

```
CREATE INDEX V1 ON V(N)
```

```
CREATE INDEX V2 ON V(P, M)
```

What does this mean?

```
CREATE INDEX V3 ON V(M, N)
```

```
CREATE UNIQUE INDEX V4 ON V(N)
```

```
CREATE CLUSTERED INDEX V5 ON V(N)
```

Not supported  
in SQLite

# Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- How many indexes **could** we create?
- Which indexes **should** we create?

# Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- How many indexes **could** we create?
- Which indexes **should** we create?

In general this is a very hard problem



# Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- The *index selection problem*
  - Given a table, and a “workload” (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)
- Who does index selection:
  - The database administrator DBA
  - Semi-automatically, using a database administration tool

# Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- The *index selection problem*
  - Given a table, and a “workload” (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)
- Who does index selection:
  - The database administrator DBA
  - Semi-automatically, using a database administration tool



# Index Selection: Which Search Key

- Make some attribute K a search key if the WHERE clause contains:
  - An exact match on K
  - A range predicate on K
  - A join on K