Database Systems
CSE 414

Lecture 15-16:
Basics of Data Storage and Indexes

(Ch. 8.3-4, 14.1-1.7, & skim 14.2-3)

Announcements

- Midterm on Monday, November 6th, in class
  - Allow 1 page of notes (both sides, 8+pt font)
- WQ4 is due Friday 11pm
- Prof. Gang Luo will be out of town Nov. 3-8
  - No office hour on Nov. 8
  - TAs will handle the midterm in class
  - Prof. Magdalena Balazinska will teach the lecture this Friday (Nov. 3)
  - Prof. Dan Suciu will teach the lecture next Wednesday (Nov. 8)

Midterm

- Content
  - Lectures 1 through 17 (today - Friday)
  - HW 1–5, WQ 1–4
- Closed book. No computers, phones, watches, etc.!
- Can bring one letter-sized piece of paper with notes, but…
  - test will not be about memorization
  - formulas provided for join algorithms & selectivity
- Similar in format & content to CSE 414 17sp midterm
  - CSE 344 tests include some things we did not cover

Motivation

- To understand performance, need to understand a bit about how a DBMS works
  - my database application is too slow… why?
  - one of the queries is very slow… why?
- Understanding query optimization
  - we have seen SQL query ~> logical plan (RA), but not much about RA ~> physical plan
- Choice of indexes is often up to you

Data Storage

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

- DBMSs store data in **files**
- Most common organization is row-wise storage:
  - File is split into blocks
  - Each block contains a set of tuples
- DBMS reads entire block

10	Tom	Hanks	block 1
20	Amy	Hanks	
50	…	…	block 2
200	…		
220			block 3
240			
420			
800			

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

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Data File Types

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 using our DB.

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

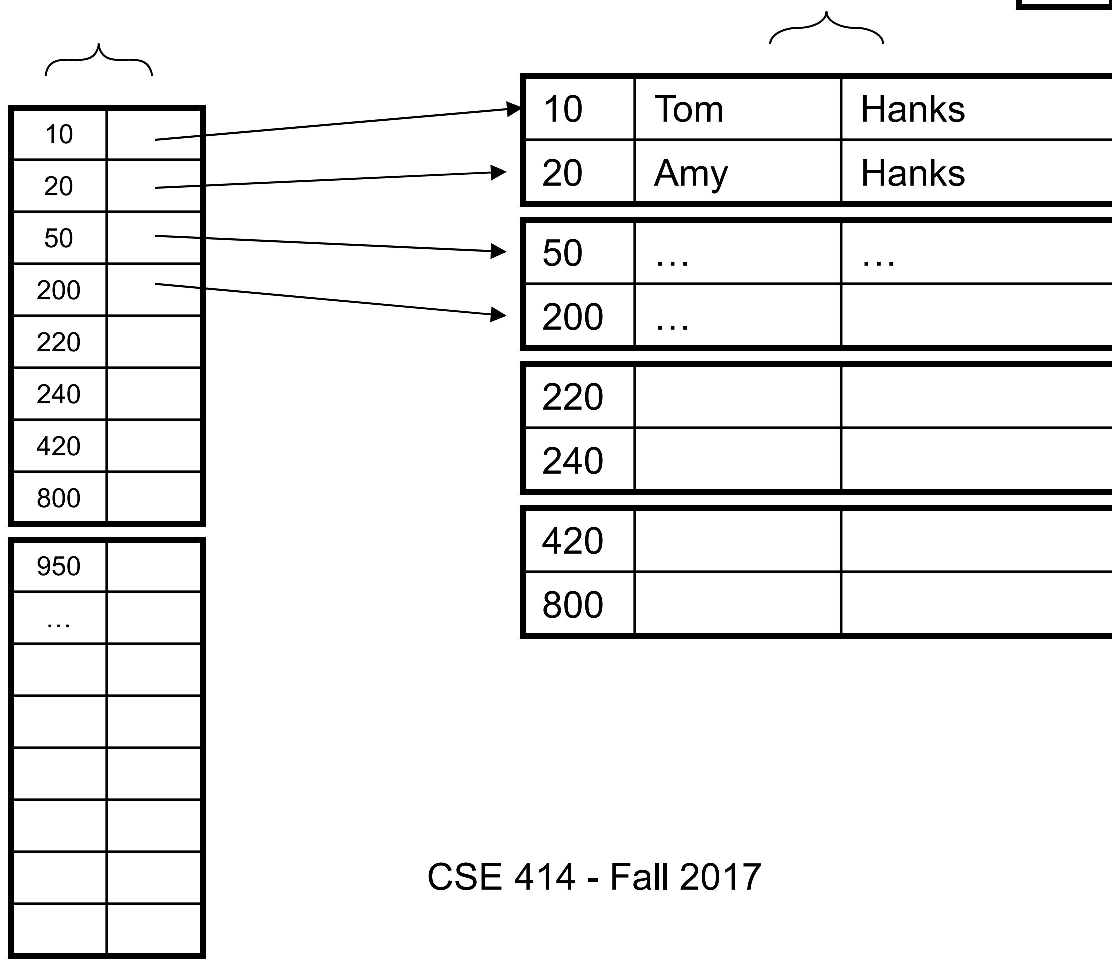
Example 1: Index on ID

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Index on **Student.ID**

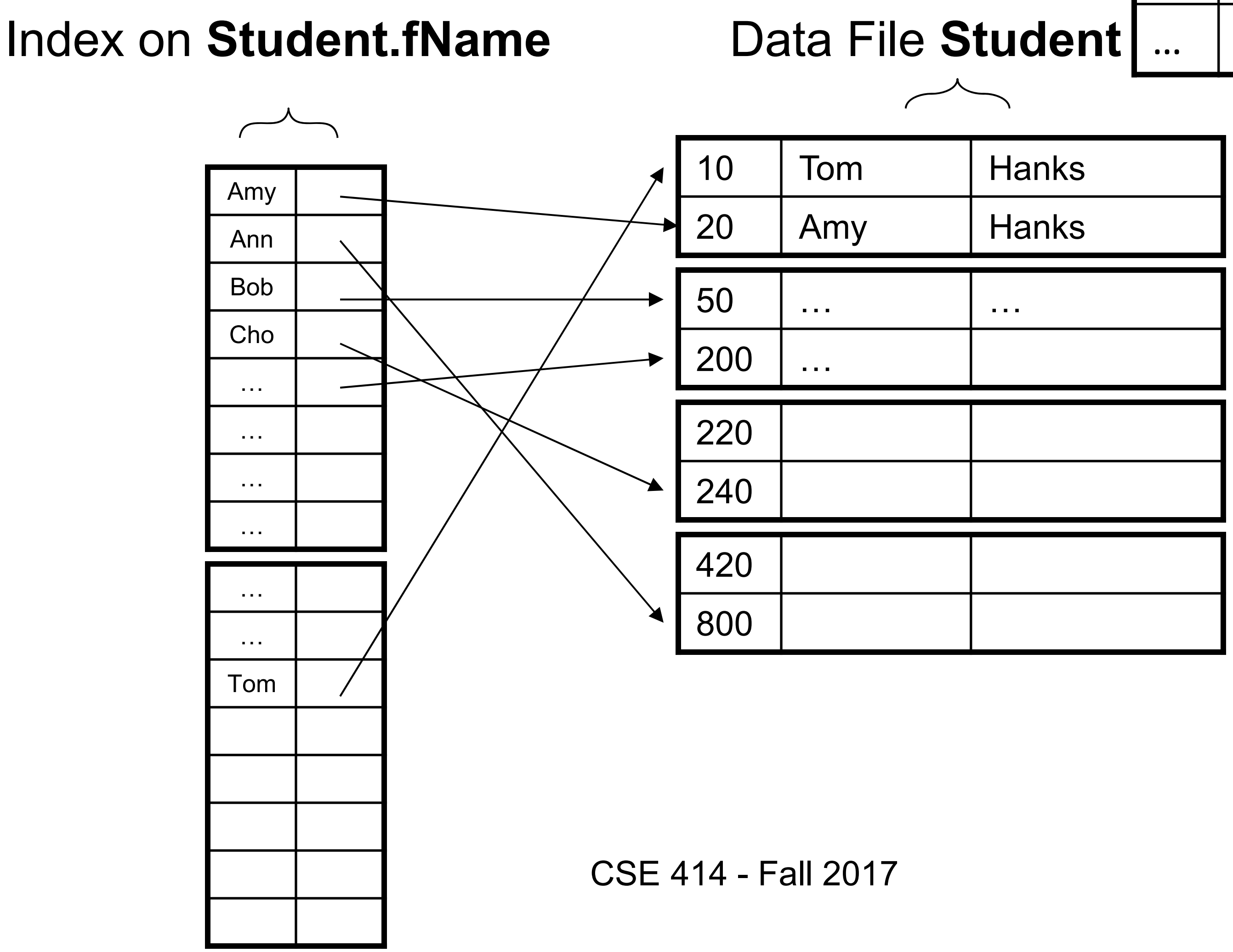
Data File **Student**



Example 2: Index on fName

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		



Index Organization

Several index organizations:

- B+ trees – most popular
  - They are search trees, but they are not binary instead have higher fan-out
- Hash table
- Specialized indexes: bit maps, R-trees, inverted index

(Each level is a fraction of the size of the one below)

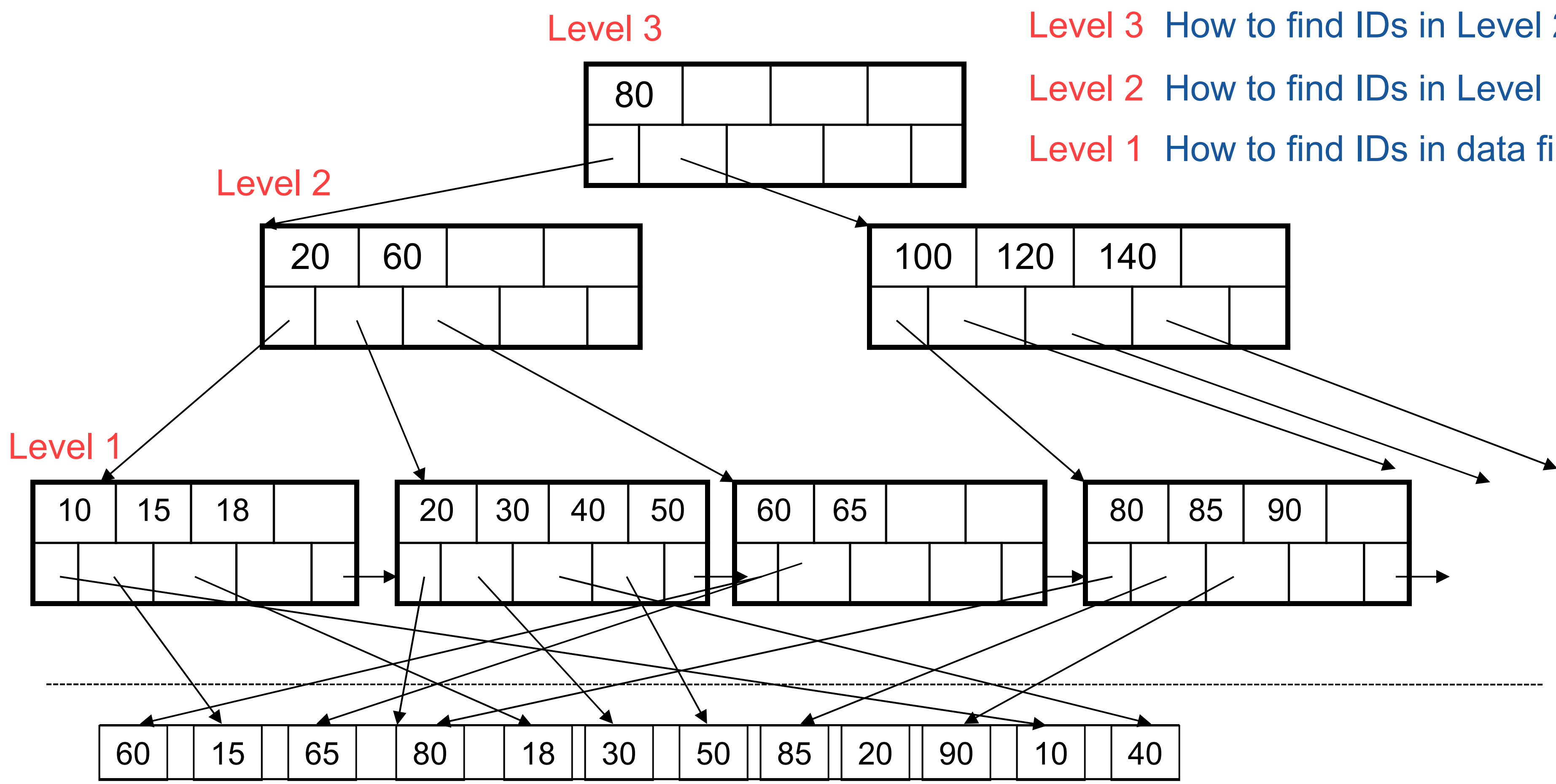
Recap: B+ Tree

…

Level 3 How to find IDs in Level 2

Level 2 How to find IDs in Level 1

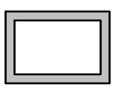
Level 1 How to find IDs in data file



Hash Index

A (naïve) hash function:

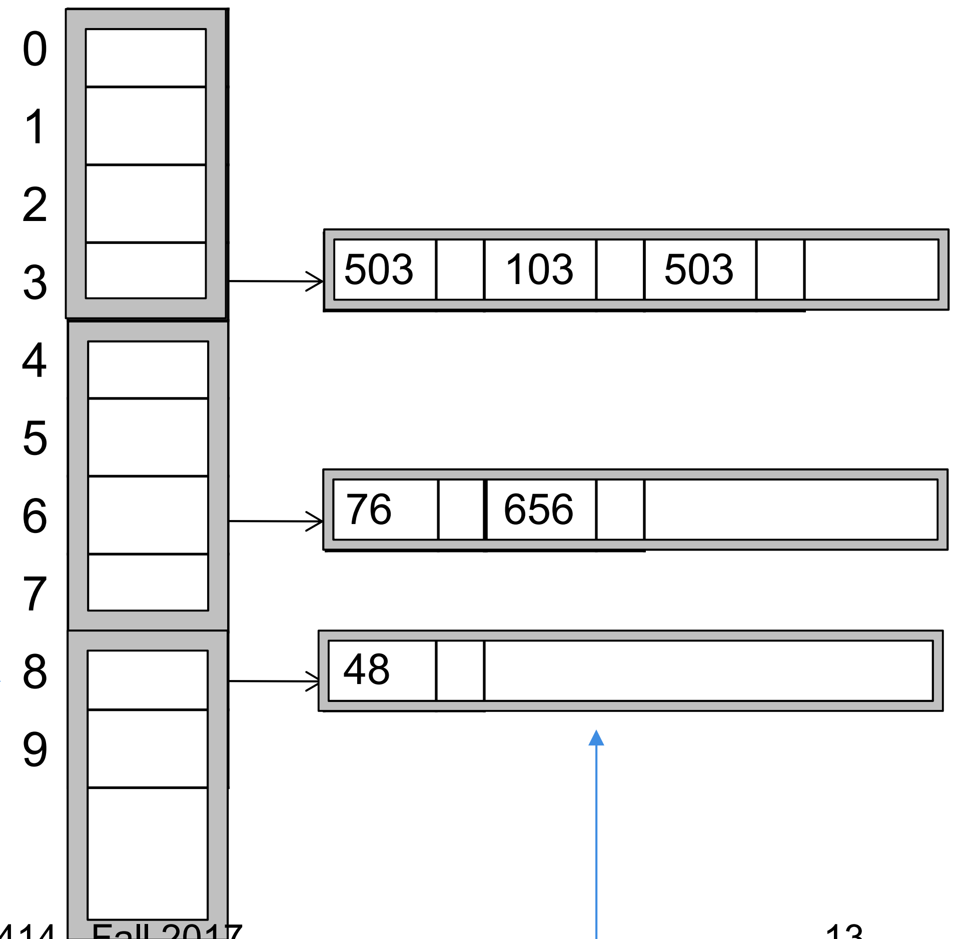
$$h(x) = x \bmod 10$$

 = disk block

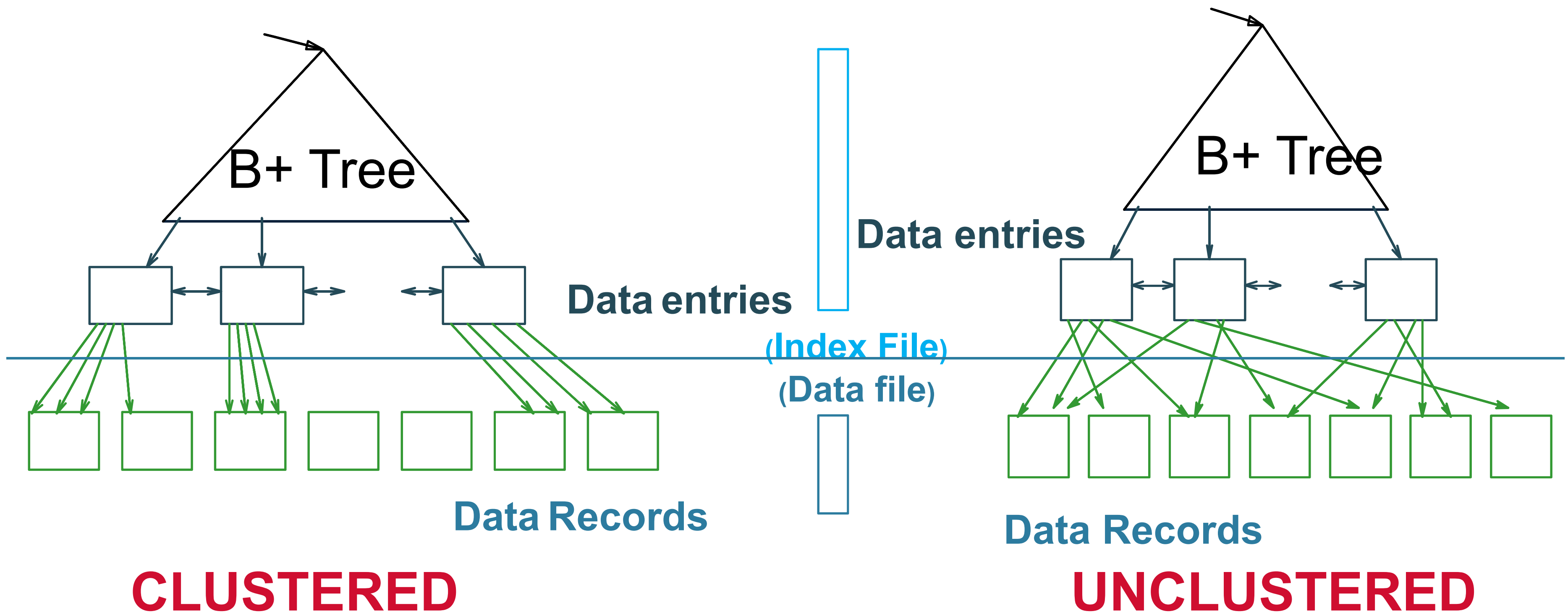
Cost per lookup:

- one access in array
- one access in list

No range queries!



Clustered vs. Unclustered



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

SQL Server defaults to cluster by **primary key**

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

- Hard disks are mechanical devices!
  - Technology from the 60s; density much higher now
- We 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 ≈ sequential scanning the entire file
  - this is decreasing over time (because of increased density of disks)

HDD ~> SSD

- Solid state (SSD): used to be too expensive… not any more
  - entirely different performance characteristics!

Seagate Technology PLC STX | ★★★

Income Statement Balance Sheet Cash Flow

Statement Type: Annual | Data Type: As of Reported | Period: 5 Years | Show Report Dates: Ascending | Data Scroll | View: $ % 1.0 | Rounding: .0 .0 | Export

Fiscal year ends in June

USD in Million except per share data	2012-06	2013-06	2014-06	2015-06	2016-06
Revenue	14,939	14,351	13,724	13,739	11,160
Cost of revenue	10,255	10,411	9,878	9,930	8,545
Gross profit	4,684	3,940	3,846	3,809	2,615
▶ Operating expenses	1,576	1,849	2,070	1,751	2,170
Operating income	3,108	2,091	1,776	2,058	445
Interest Expense	241	214	195	207	193
Other income (expense)	15	(46)	(25)	119	22
Income before taxes	2,882	1,831	1,556	1,970	274
Provision for income t...	20	(7)	(14)	228	26
Net income from contin...	2,862	1,838	1,570	1,742	248
Net income	2,862	1,838	1,570	1,742	248

Takes(studentID, courseID)

Student(ID, name, …)

Example

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

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

Assume the database has indexes on these attributes:

- **index_takes_course** = index on Takes.courseID
- **index_studentID** = index on Student.ID

Index selection

Index join

```
for y1 in index_takes_course where y1.courseID = 300
   for y in y1.Takes
        for x1 in index_studentID where x.ID = y.studentID
            for x in x1.Student
               output x.*, y.*
```

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?

15, namely: (ID), (fName), (lName), (ID,fName), (fName,ID), …

- Which indexes should we create?

Few! Each new index slows down updates to Student

Index selection is a hard problem

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Which Indexes?

- 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:
  - 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

Index Selection Problem

V(M, N, P);

Suppose the database has the index I1 below. Discuss physical query plans for these queries.

INDEX I1 on V(M)

```
SELECT *
FROM V
WHERE V.M = 33
```

```
Scan V
For each record:
  if M=33 then output
```

```
Lookup key 33 in I1
For each record: output
```

```
SELECT *
FROM V
WHERE V.M = 33 and V.P = 55
```

```
Scan V
For each record:
  if M=33 and P=55 then output
```

```
Lookup key 33 in I1
For each record
  if P=55 then output
```

Index Selection Problem 1

V(M, N, P);

Your workload is this (and nothing else)

100,000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

Which indexes ?

Index Selection Problem 1

V(M, N, P);

Your workload is this (and nothing else)

100,000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

A: V(N) and V(P) (hash tables or B-trees)

Index Selection Problem 2

V(M, N, P);

Your workload is this

100,000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100,000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

Which indexes ?

Index Selection Problem 2

```
V(M, N, P);
```

Your workload is this

100,000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100,000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: definitely V(N) (must B-tree); unsure about V(P)

Index Selection Problem 3

V(M, N, P);

Your workload is this

100,000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1,000,000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100,000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

Which indexes ?

Index Selection Problem 3

V(M, N, P);

Your workload is this

100,000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1,000,000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100,000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: V(N, P) (B-tree)

How does this index differ from:

1. Two indexes V(N) and V(P)?
2. An index V(P, N)?

Index Selection Problem 4

V(M, N, P);

Your workload is this

1,000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100,000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

Which indexes ?

Index Selection Problem 4

```
V(M, N, P);
```

Your workload is this

1,000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100,000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

A: V(N) secondary, V(P) primary index (both B-tree)

Index Selection Problem 5

```
V(M, N, P);
```

```
SELECT *
FROM V
WHERE V.M = 33
```

```
SELECT *
FROM V
WHERE V.M = 33 and V.P = 55
```

Suppose the database has these indexes. Which ones can the optimizer use?

INDEX I1 on V(M)

INDEX I2 on V(M, P)

INDEX I3 on V(P, M)

Recap – Indexes

V(M, N, P);

```
SELECT *
FROM V
WHERE V.M = 33
```

Yes

```
SELECT *
FROM V
WHERE V.M = 33 and V.P = 55
```

Suppose the database has these indexes. Which ones can the optimizer use?

INDEX I1 on V(M)

INDEX I2 on V(M, P)

INDEX I3 on V(P, M)

Recap – Indexes

V(M, N, P);

```
SELECT *
FROM V
WHERE V.M = 33
```

```
SELECT *
FROM V
WHERE V.M = 33 and V.P = 55
```

Suppose the database has these indexes. Which ones can the optimizer use?

Yes (why?)

INDEX I1 on V(M)

INDEX I2 on V(M, P)

Yes

INDEX I3 on V(P, M)

Recap – Indexes

V(M, N, P);

```
SELECT *
FROM V
WHERE V.M = 33
```

```
SELECT *
FROM V
WHERE V.M = 33 and V.P = 55
```

Suppose the database has these indexes. Which ones can the optimizer use?

INDEX I1 on V(M) — No! (why?)

INDEX I2 on V(M, P)

INDEX I3 on V(P, M) — Yes

Recap – Indexes

Movie(mid, title, year)

CLUSTERED INDEX I on Movie(id)
INDEX J on Movie(year)

```
SELECT *
FROM Movie
WHERE year = 2010
```

```
SELECT *
FROM Movie
WHERE year = 1910
```

The system uses the index J for one of the queries, but not for the other.

Which and why?

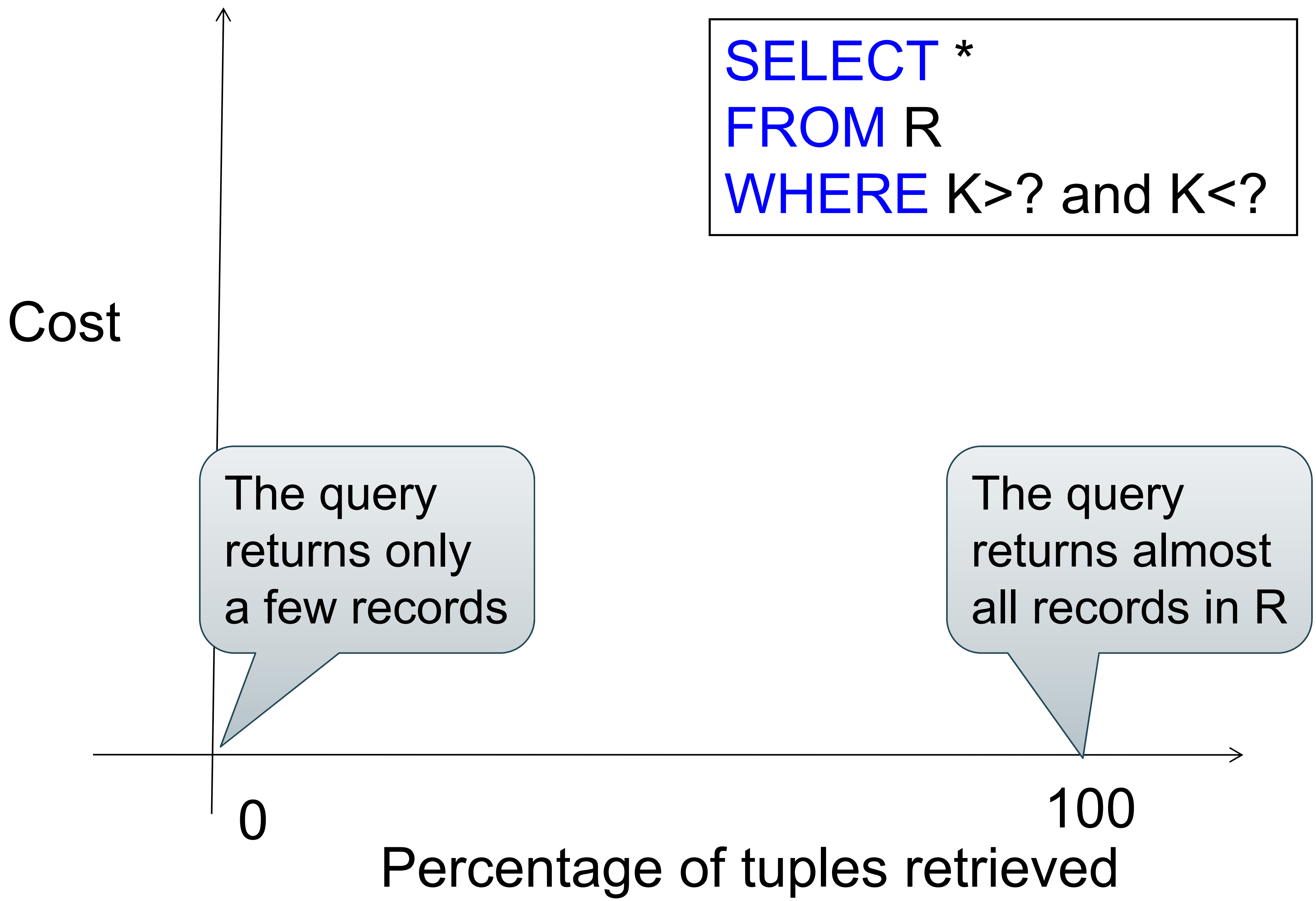
Basic Index Selection Guidelines

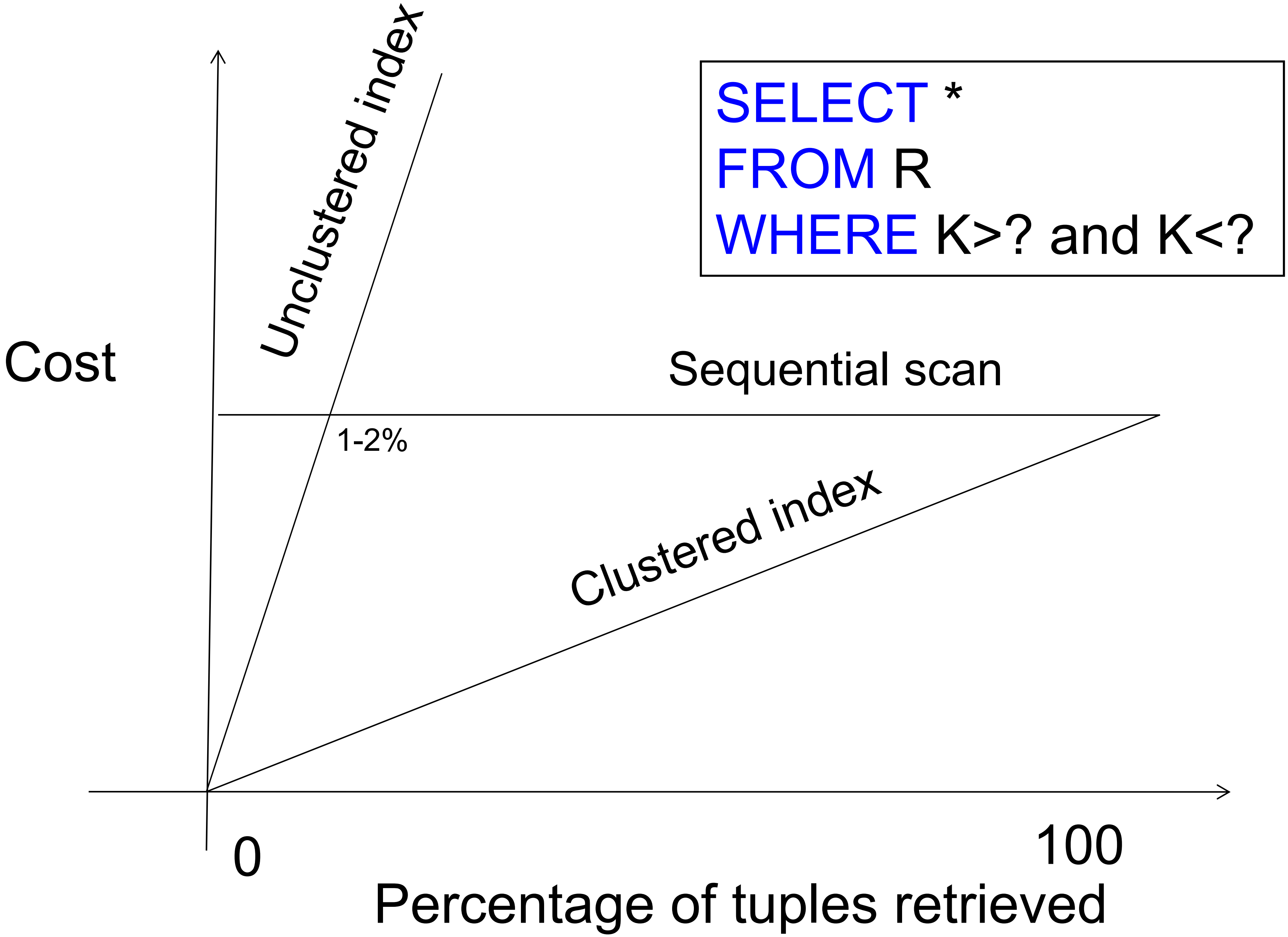
- Consider queries in workload in order of importance
  - ignore infrequent queries if you also have many writes
- Consider relations accessed by query
  - No point indexing other relations
- Look at WHERE clause for possible search key
- Try to choose indexes that speed-up multiple queries

To Cluster or Not

- Range queries benefit mostly from clustering
- Covering indexes do *not* need to be clustered: they work equally well unclustered
  - (a covering index for a query is one where every attribute mentioned in the query is part of the index's search key)
  - in that case, index has all the info you need anyway

```
SELECT *
FROM R
WHERE K>? and K<?
```




SELECT *
FROM R
WHERE K>? and K<?
Unclustered index
Cost
Sequential scan
1-2%
Clustered index
0
100
Percentage of tuples retrieved

Midterm Concept Review I

- relational data model
  - set semantics vs. bag semantics
  - primary & secondary keys
  - foreign keys
  - schemas

- SQL
  - CREATE TABLE
  - SELECT-FROM-WHERE (SFW)
  - joins: inner vs. outer, natural
  - group by & aggregation
  - ordering
  - CREATE INDEX

Midterm Concept Review II

- relational queries
  - languages for writing them:
    - standard relational algebra
    - Datalog (even without recursion)
    - SQL (even without grouping / aggregation)
  - monotone queries are a proper subset
  - SFW queries (i.e., w/out subqueries) are monotone

Midterm Concept Review III

- types of indexes
  - B+ tree vs. hash
    - hash indexes use at most 2 disk accesses
    - B+ tree can be used for < predicates
    - B+ tree index on (X, Y) also allows searching for X=a matches
  - clustered vs non-clustered
    - selectivity above 1-2% => not helped by non-clustered indexes

- cost-based query optimization
  - consider choices over logical and physical query plans
    - most important choice in latter is choice of join algorithm
    - those include nested loop, sorted merge, hash, and indexed joins
  - primary goal of the optimizer is to avoid really bad plans