Introduction to Database Systems CSE 444

Lecture 16: Database Tuning

About the Midterm

- Open book and open notes
  - But you won’t have time to read during midterm!
  - No laptops, no mobile devices

- Three questions:
  1. SQL
  2. ER Diagrams
  3. Transactions

More About the Midterm

- Review Lectures 1 through 14
  - Read the lecture notes carefully
  - Read the book for extra details, extra explanations

- Review Project 1 (Project 2 not on any exam)
- Review HW1 and HW2

- Take a look at sample midterms

Where We Are?

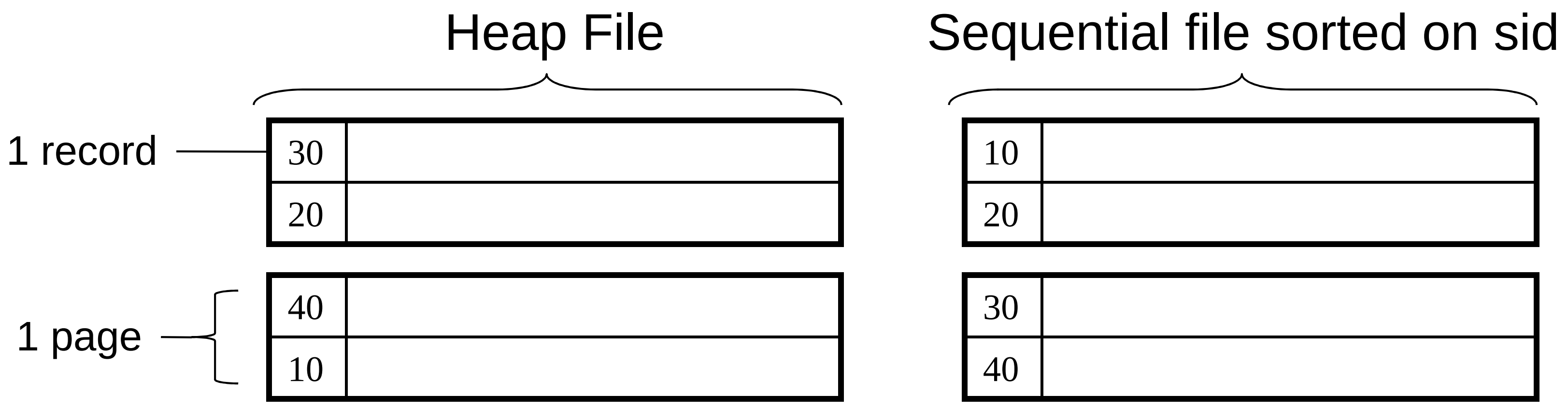
- We just started to learn how a DBMS executes a query…
- … we started with data storage and indexing

Data Storage & Indexing: Review

How does a DBMS store data?

- Typically one relation = one file
- Heap file: tuples inside file are not sorted
- Sequential file: tuples sorted on a key

```
Student(sid: int, age: int, …)
```



Indexes: Motivation

- **Index:** data structure to speed-up selections on ***search key fields*** for the index

- An index contains a collection of ***data entries***, and supports efficient retrieval of all data entries with a given search key value **k**

Indexes

- **Search key** = can be any set of fields
  - not the same as the primary key, nor a key
- **Index** = collection of data entries
- **Data entry** for key k can be:
  - The actual record with key k
    - In this case, **the index is also a special file organization**
  - (k, RID)
    - K is the key
    - RID (Record ID) is a pointer to the record inside the data file

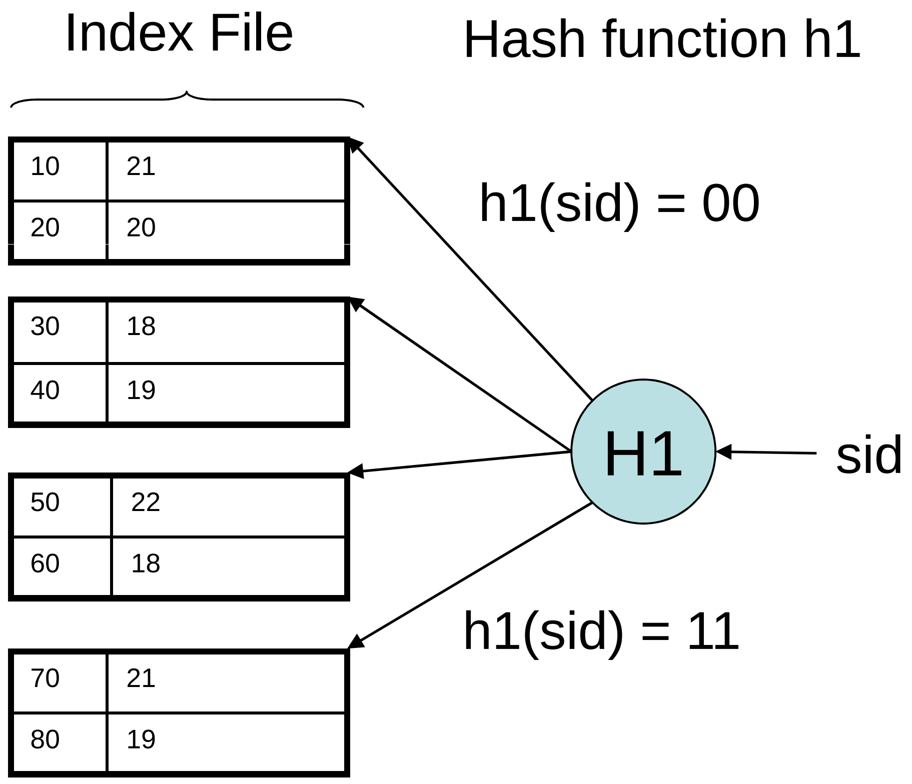
Hash-Based Index Example

Example hash-based index on sid (student id)

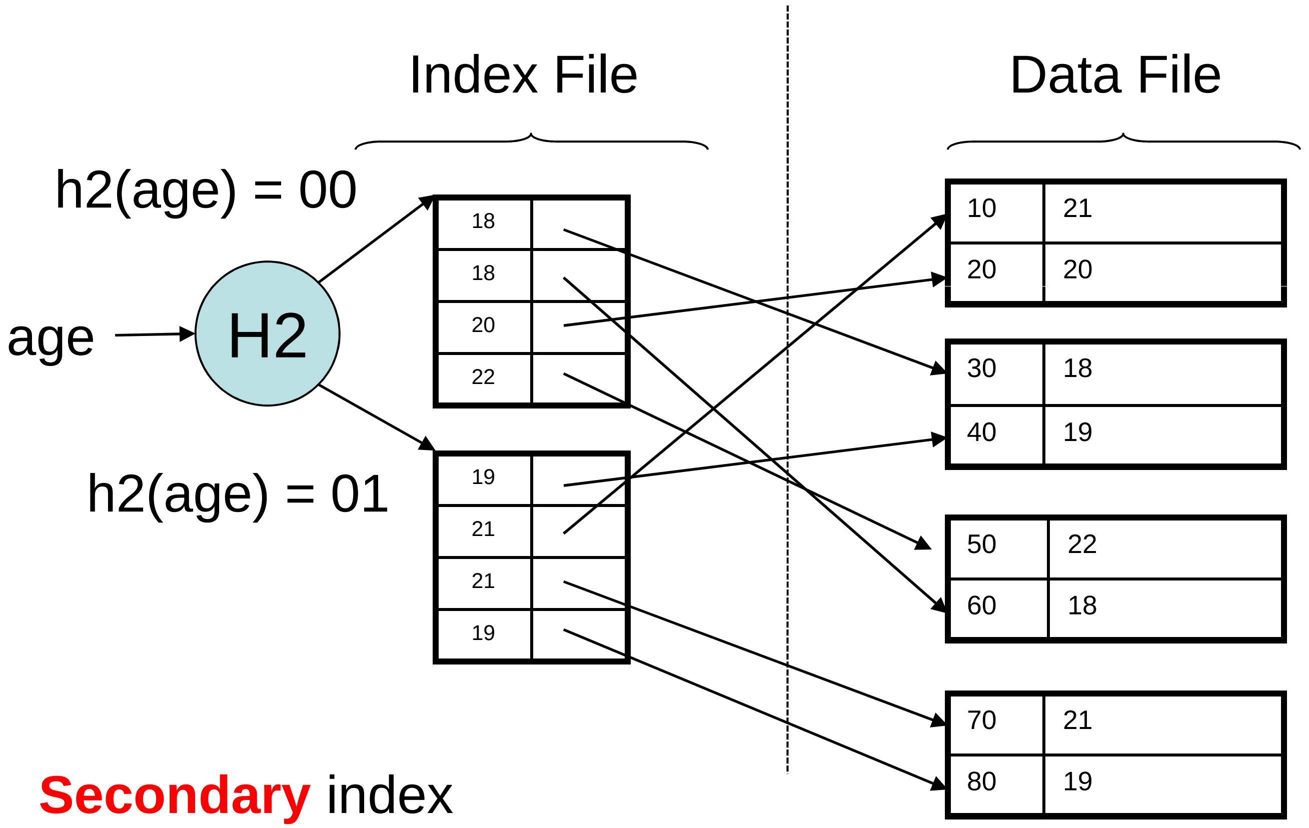
This is a **primary** index because it determines the location of indexed records

In this case, data entries in the index are actual data records
There is no separate data file

This index is also **clustered**



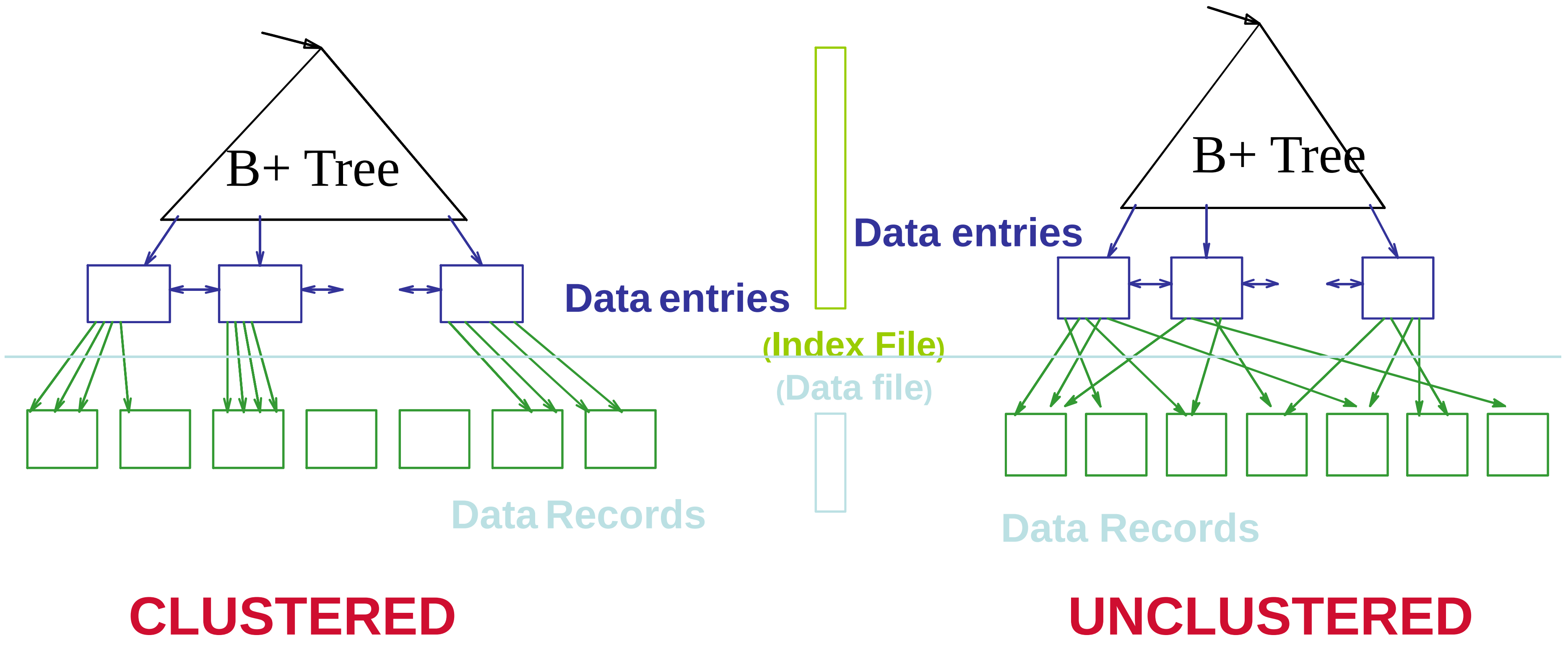
Hash-Based Index Example 2



Secondary index
Data entries in index are (key,RID) pairs

Unclustered index

Tree-Based Indexes



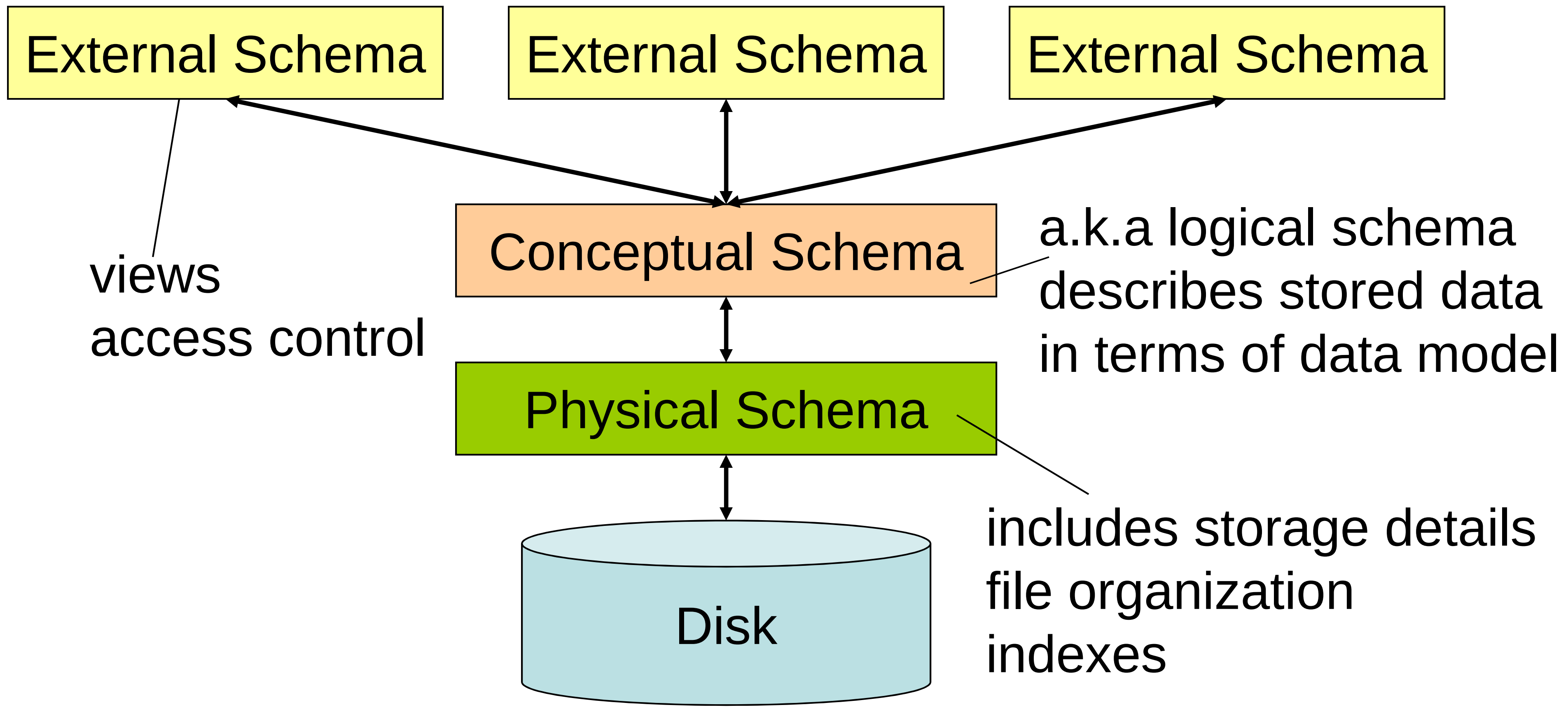
Data entries in index can also be data records

Database Tuning Overview

- The database tuning problem
- Index selection (discuss in detail)
- Horizontal/vertical partitioning (see lecture 4)
- Denormalization (discuss briefly)

This material is partially based on the book: “Database Management Systems” by *Ramakrishnan and Gehrke*, **Ch. 20**

Levels of Abstraction in a DBMS



The Database Tuning Problem

- We are given a workload description
  - List of queries and their frequencies
  - List of updates and their frequencies
  - Performance goals for each type of query
- Perform *physical database design*
  - Choice of indexes
  - Tuning the conceptual schema
    - Denormalization, vertical and horizontal partition
  - Query and transaction tuning

The Index Selection Problem

- Given a database schema (tables, attributes)
- Given a “query workload”:
  - Workload = a set of (query, frequency) pairs
  - The queries may be both SELECT and updates
  - Frequency = either a count, or a percentage
- Select a set of indexes that optimizes the workload

In general this is a very hard problem

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

The Index Selection Problem 1

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

Your workload is this

100000 queries:

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

100 queries:

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

What indexes ?

The Index Selection Problem 1

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

Your workload is this

100000 queries:

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

100 queries:

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

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

The Index Selection Problem 2

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

Your workload is this

100000 queries:

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

100 queries:

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

100000 queries:

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

What indexes ?

The Index Selection Problem 2

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

Your workload is this

100000 queries:

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

100 queries:

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

100000 queries:

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

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

The Index Selection Problem 3

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

Your workload is this

100000 queries:

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

1000000 queries:

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

100000 queries:

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

What indexes ?

The Index Selection Problem 3

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

Your workload is this

100000 queries:

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

1000000 queries:

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

100000 queries:

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

A: V(N, P)

The Index Selection Problem 4

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

Your workload is this

1000 queries:

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

100000 queries:

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

What indexes ?

The Index Selection Problem 4

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

Your workload is this

1000 queries:

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

100000 queries:

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

A: V(N) secondary,   V(P) primary index

The Index Selection Problem

- SQL Server
  - Automatically, thanks to *AutoAdmin* project
  - Much acclaimed successful research project from mid 90's, similar ideas adopted by the other major vendors
- PostgreSQL
  - You will do it manually, part of project 3
  - But tuning wizards also exist

Basic Index Selection Guidelines

- Consider queries in workload in order of importance
- 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
- And then consider the following…

Index Selection: Multi-attribute Keys

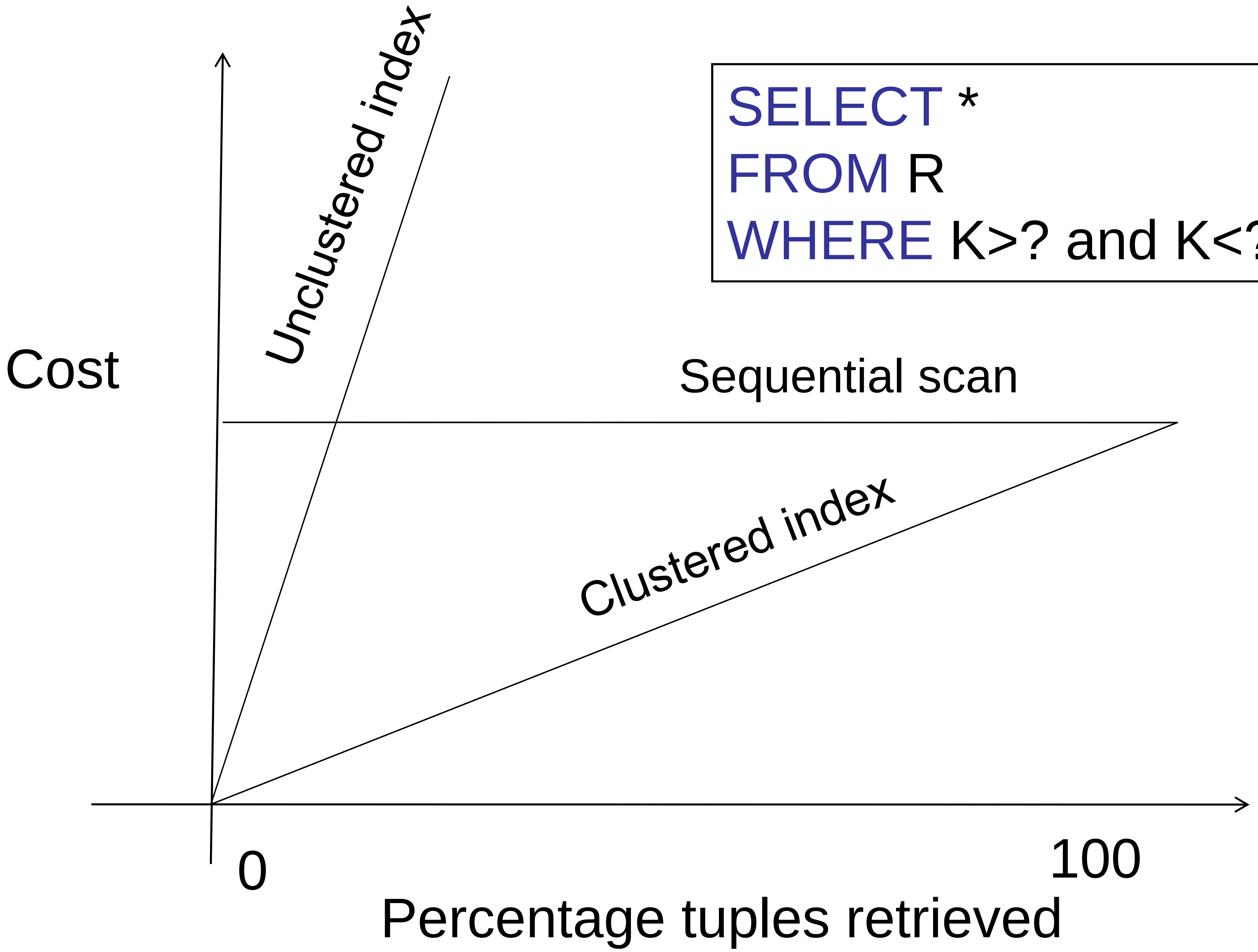
Consider creating a multi-attribute key on K1, K2, … if

- WHERE clause has matches on K1, K2, …
  - But also consider separate indexes
- SELECT clause contains only K1, K2, ..
  - A *covering index* is one that can be used exclusively to answer a query, e.g. index R(K1,K2) covers the query:

```
SELECT K2 FROM R WHERE K1=55
```

To Cluster or Not

- Range queries benefit mostly from clustering
- Covering indexes do *not* need to be clustered: they work equally well unclustered



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

Hash Table v.s. B+ tree

- Rule 1: always use a B+ tree ☺

- Rule 2: use a Hash table on K when:
  - There is a very important selection query on equality (WHERE K=?), and no range queries
  - You know that the optimizer uses a nested loop join where K is the join attribute of the inner relation (you will understand that in a few lectures)

Balance Queries v.s. Updates

- Indexes speed up queries
  - SELECT FROM WHERE
- But they usually slow down updates:
  - INSERT, DELETE, UPDATE
  - However some updates benefit from indexes

```
UPDATE R
  SET A = 7
  WHERE K=55
```

Tools for Index Selection

- SQL Server 2000 Index Tuning Wizard
- DB2 Index Advisor

- How they work:
  – They walk through a large number of configurations, compute their costs, and choose the configuration with minimum cost

Tuning the Conceptual Schema

- Index selection
- Horizontal/vertical partitioning (see lecture 4)
- Denormalization

Denormalization

Product(<u>**pid**</u>, pname, price, cid)
Company(<u>**cid**</u>, cname, city)

A very frequent query:

```
SELECT x.pid, x.pname
FROM Product x, Company y
WHERE x.cid = y.cid and x.price < ? and y.city = ?
```

How can we speed up this query workload ?

Denormalization

Product(<u>**pid**</u>, pname, price, cid)
Company(<u>**cid**</u>, cname, city)

Denormalize:

ProductCompany(<u>**pid**</u>, pname, price, cname, city)

```
INSERT INTO ProductCompany
   SELECT x.pid, x.pname, x.price, y.cname, y.city
   FROM Product x, Company y
   WHERE x.cid = y.cid
```

Denormalization

Next, replace the query

```
SELECT x.pid, x.pname
FROM Product x, Company y
WHERE x.cid = y.cid and x.price < ? and y.city = ?
```

```
SELECT pid, pname
FROM ProductCompany
WHERE price < ? and city = ?
```

Issues with Denormalization

- It is no longer in BCNF
  - We have the hidden FD: cid → cname, city
- When Product or Company are updated, we need to propagate updates to ProductCompany
  - Use RULE in PostgreSQL (see PostgreSQL doc.)
  - Or use a trigger on a different RDBMS
- Sometimes cannot modify the query
  - What do we do then ?

Denormalization Using Views

```
INSERT INTO ProductCompany
  SELECT x.pid, x.pname,.price, y.cid, y.cname, y.city
  FROM Product x, Company y
  WHERE x.cid = y.cid;

DROP Product; DROP Company;

CREATE VIEW Product AS
  SELECT pid, pname, price, cid FROM ProductCompany

CREATE VIEW Company AS
  SELECT DISTINCT cid, cname, city FROM ProductCompany
```