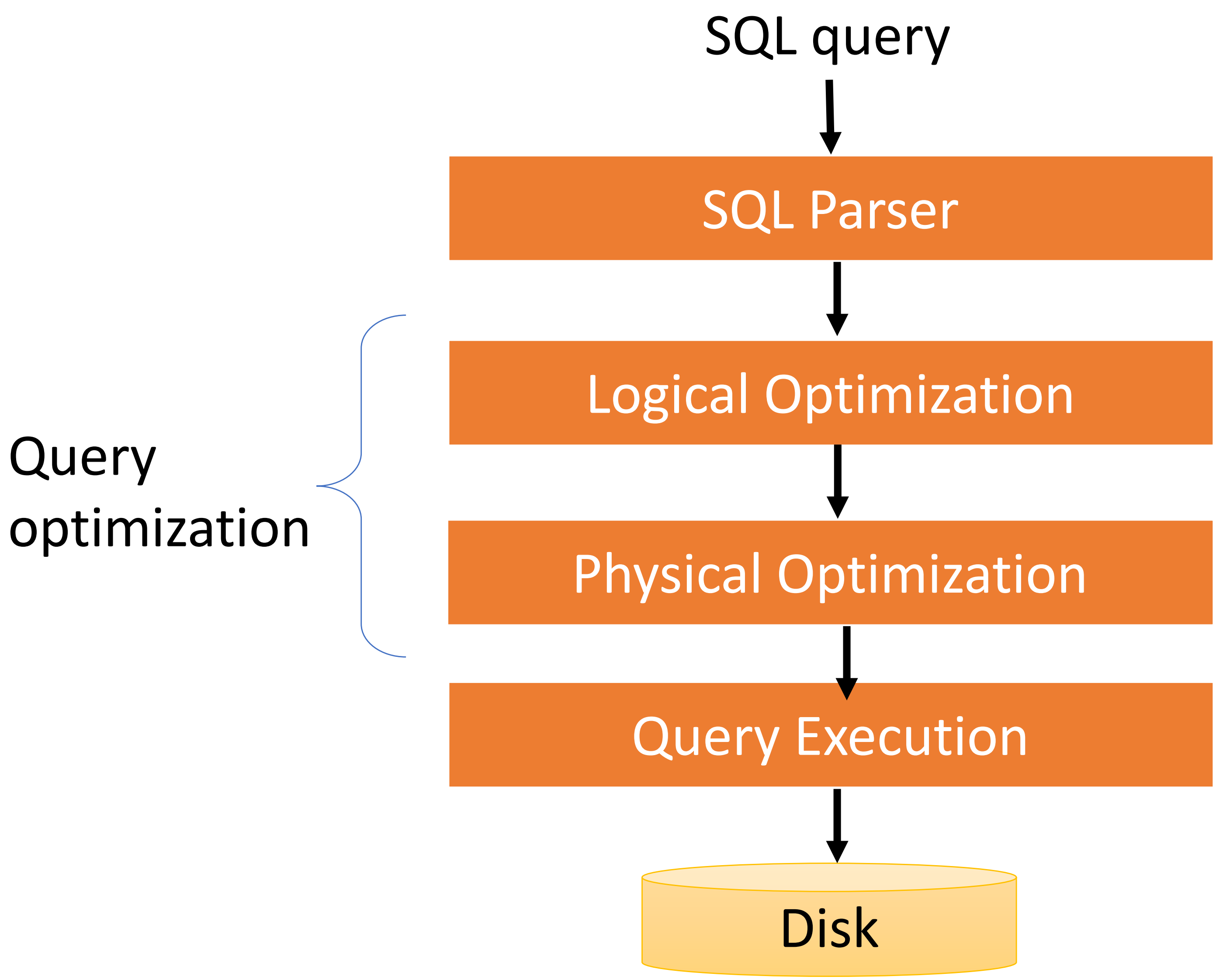
CMPT 354: Database System I

Lecture 6. Basics of Query Processing and Indexing

Outline

- Query Processing
  - What happens when an SQL query is issued?
- Indexing
  - How to speed up query performance?

Query Processing Steps



Example

- **Offering** (oID, dept, cNum, term, instructor)
- **Took** (sID, oID, grade)

Q: Student number of all students who have taken CMPT 354

```
SELECT sID
FROM   Offering O, Took T
WHERE  O.oID = T.oID
      AND O.dept = 'CMPT'
      AND O.cNum = '354'
```

Offering (<u>oID</u>, dept, cNum, term, instructor)

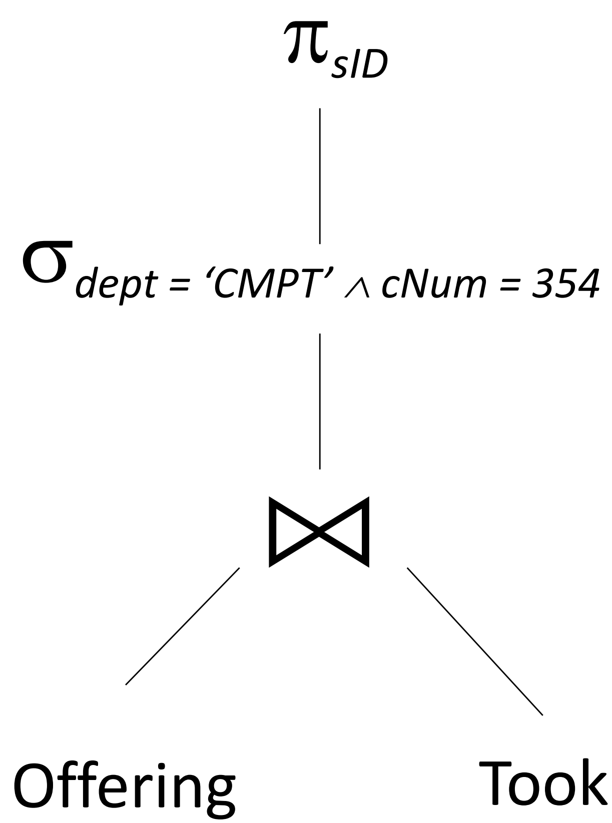
Took (<u>sID, oID</u>, grade)

SQL Parser

- From the input SQL text to a logical plan

```sql
SELECT sID
FROM   Offering O, Took T
WHERE  O.oID = T.oID
       AND O.dept = 'CMPT'
       AND O.cNum = '354'
```

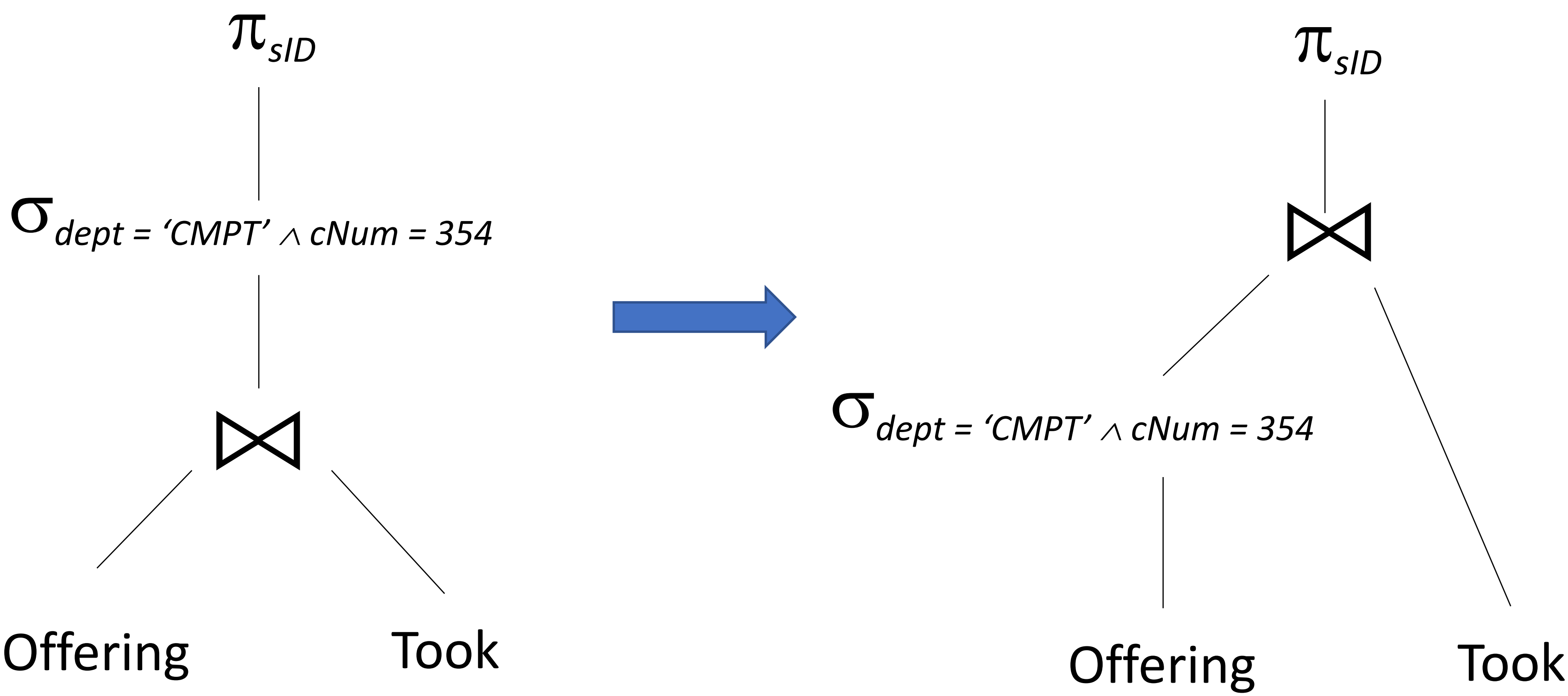
$$\pi_{sID}(\sigma_{dept = \text{'CMPT'} \wedge cNum = 354}(\text{Offering} \bowtie \text{Took}))$$



Relational algebra expression is also called the "**logical query plan**"

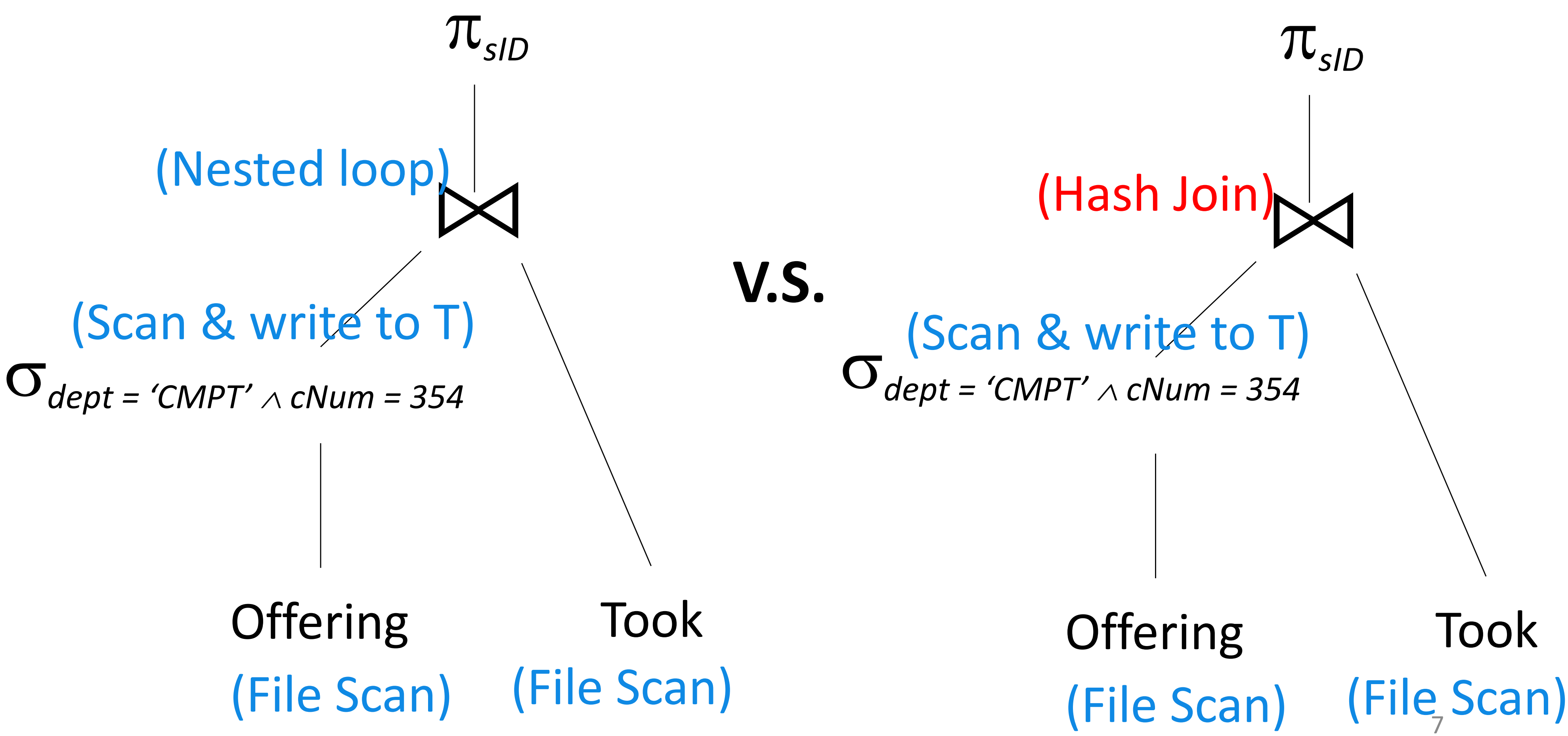
Logical Optimization

- Find the **optimal logical plan**



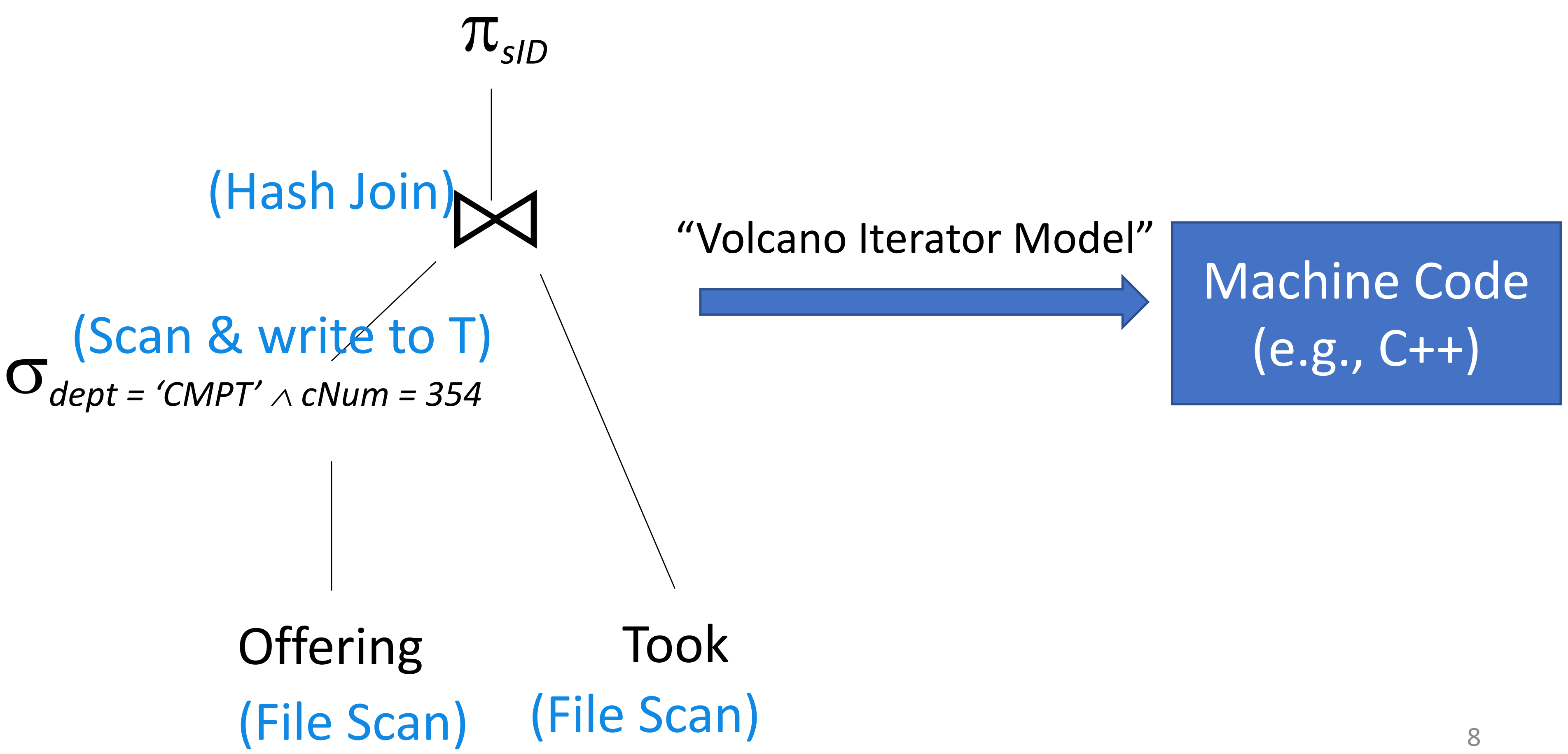
Physical Optimization

- Find the **optimal physical plan**



Query Execution

- From a physical plan to actual machine code



Summary

- **Logical plans:**
  - Created by the parser from the input SQL text
  - Expressed as a relational algebra tree
  - Each SQL query has many possible logical plans
- **Physical plans:**
  - Goal is to choose an efficient implementation for each operator in the RA
  - Each logical plan has many possible physical plans
- **Query Optimization:**
  - Find the optimal logical plan
  - Find the optimal physical plan

Outline

- Query Processing
  - What happens when an SQL query is issued?

- **Indexing**
  - **How to speed up query performance?**

Query Performance

- My database application is too slow… why?
- One of the queries is very slow… why?

- To address these problems, we need to understand:
  - How is data organized on disk
  - What is an index
  - How to select indexes

Data Storage

sID	dept	cNum	Term	instructor
10	CMPT	345	SP 2018	Jiannan
20	CMPT	454	FA 2018	Martin
…	…	…	…	…

- 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	CMPT	345	SP 2018	Jiannan	Block 1
20	CMPT	454	FA 2018	Martin	
30	…	…	…	…	Block 2
40	…				
50					Block 3
60					
70					Block 4
80					

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

Scanning a Data File

- Data file is stored on Disk
- Consequence: Sequential IO is MUCH FASTER than random IO
  - 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

Data File Types

- **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 **sID**. Might as well order by **instructor**, if that seems a better idea for the applications running on our database.

Index Motivation

`Student(name, age)`

- Suppose we want to search for students of a specific age

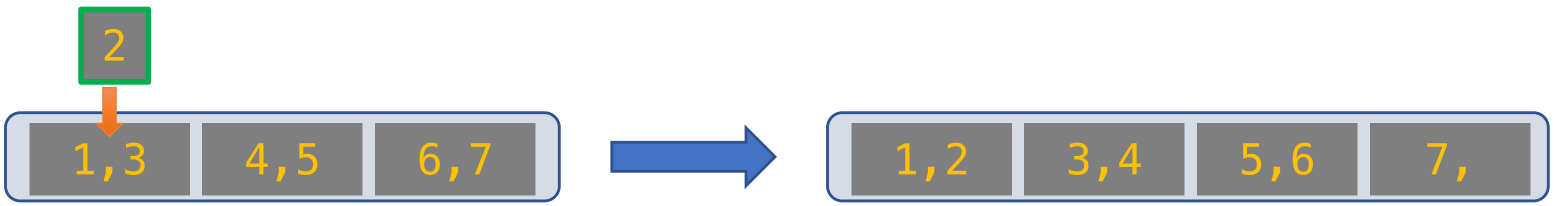
- ***First idea:*** Sort the records by age… we know how to do this fast!

- How many IO operations to search over ***N sorted*** records?
  - Simple scan: ***O(N)***
  - Binary search: ***O(***$\log_2 N$***)***

Could we get even cheaper search? E.g. go from $\log_2 N$ $\rightarrow \log_{200} N$?

Index Motivation

- What about if we want to **insert** a new student, but keep the list sorted?



- We would have to potentially shift ***N*** records, requiring up to **~ 2*N/P** IO operations (where P = # of records per page)!

Could we get faster insertions?

Index Motivation

- What about if we want to be able to search quickly along multiple attributes (e.g. not just age)?
  - We could keep multiple copies of the records, each sorted by one attribute set… this would take a lot of space

Can we get fast search over multiple attribute sets without taking too much space?

We’ll create separate data structures called *indexes* to address all these points

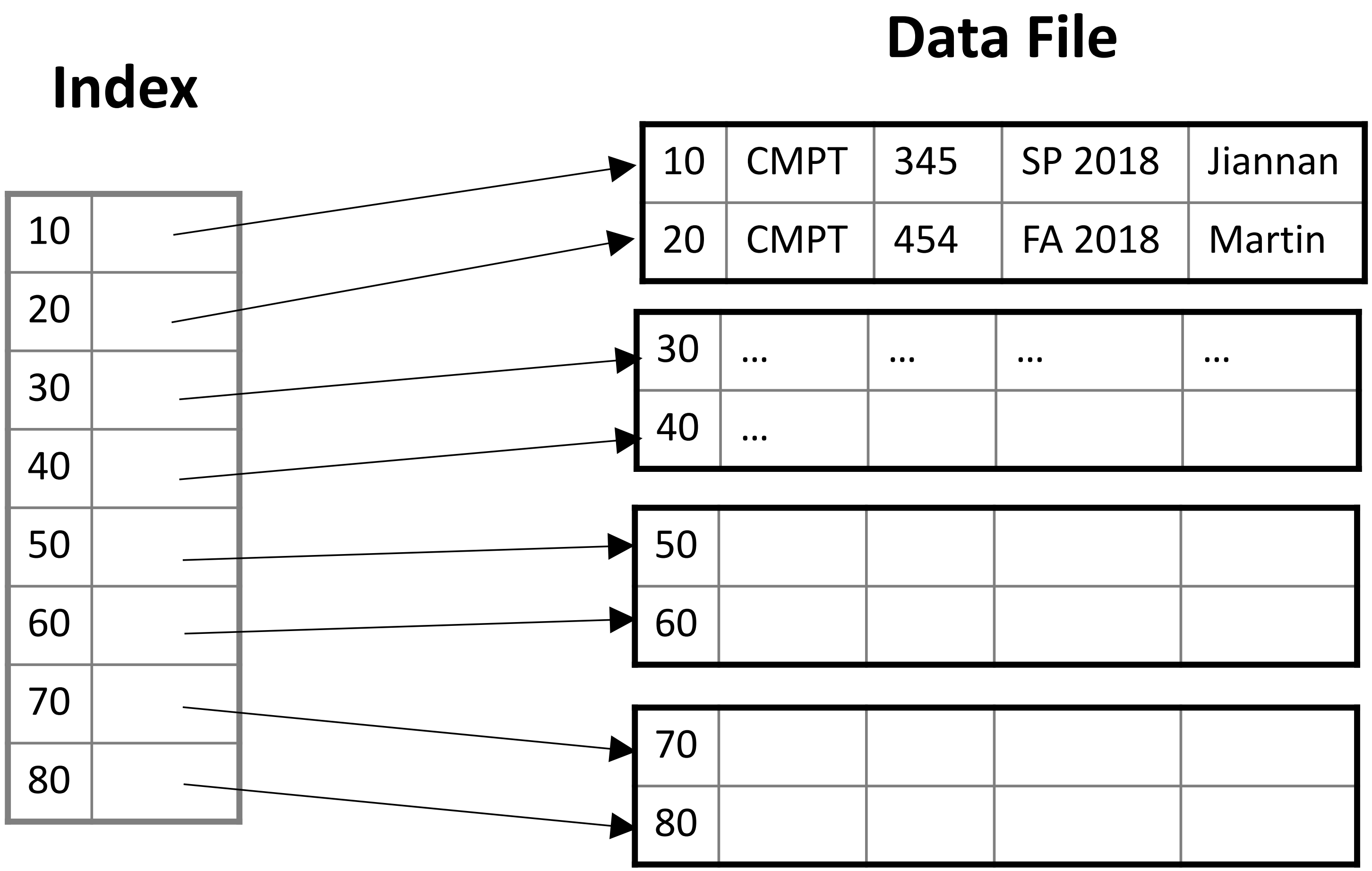
Index

- 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
- An index can store the full rows it points to (*primary index*) or pointers to those rows (*secondary index*)
  - We'll mainly consider secondary indexes
- Could have many indexes for one table

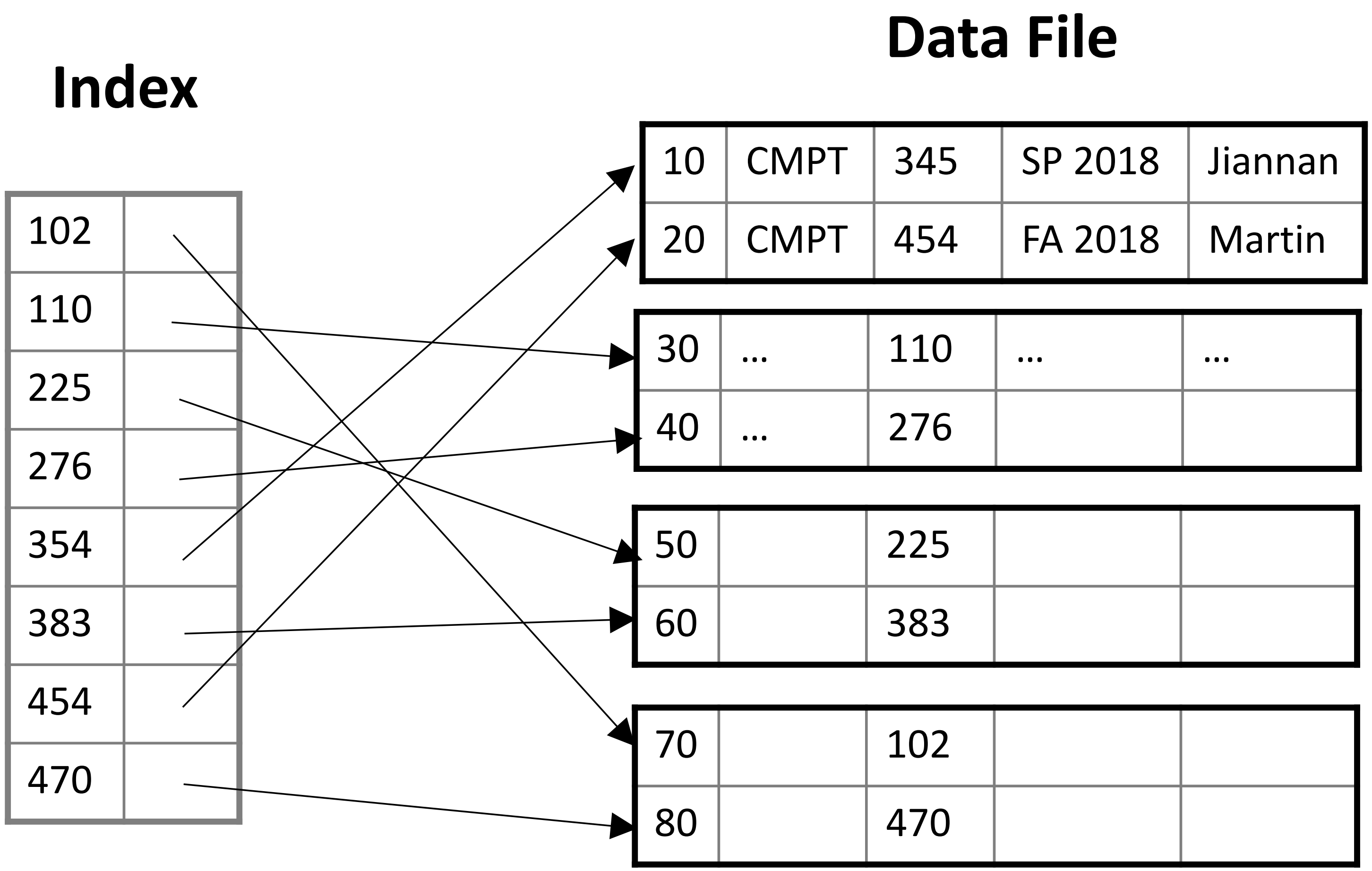
Different Keys

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

Example 1: Index on sID



Example 2: Index on cNum



Index Organization

- Common indexes:
  - Hash tables
  - B+ trees

- Specialized indexes
  - R-trees
  - Inverted index
  - …

B+ Tree Example

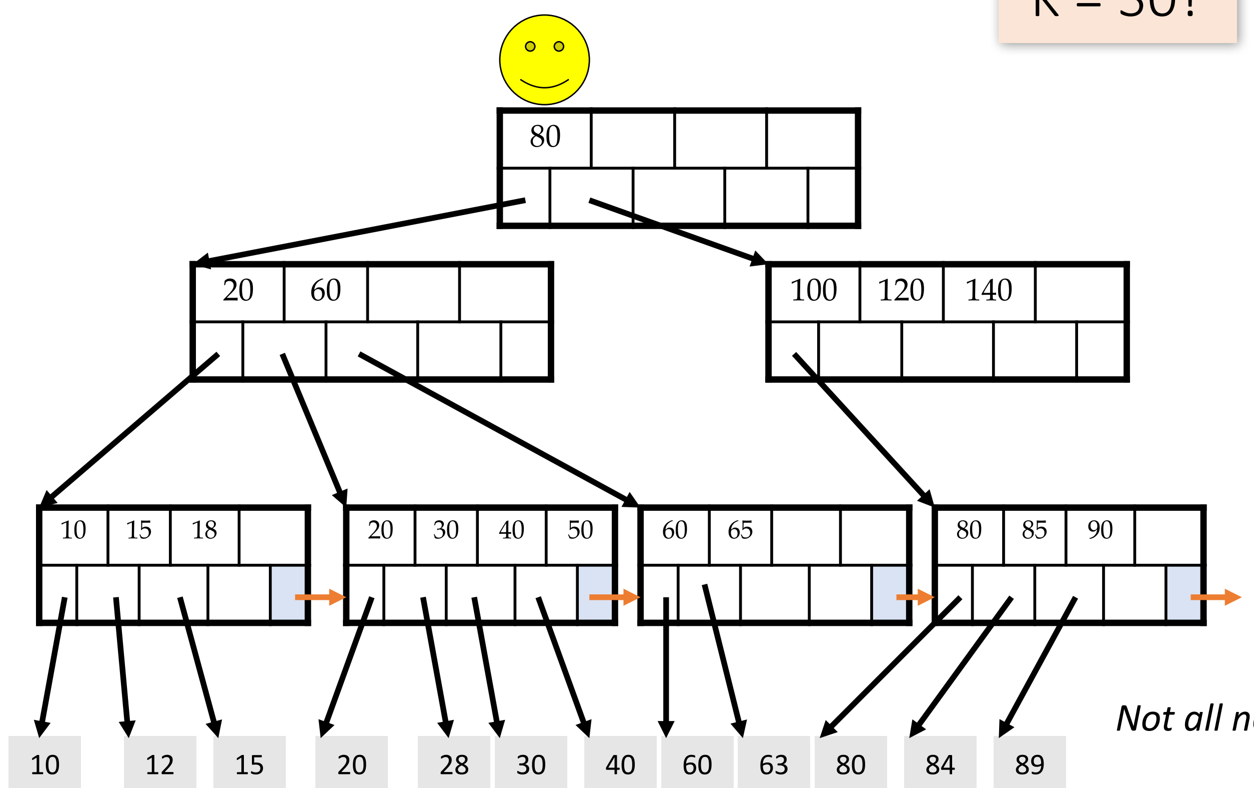
K = 30?

30 < 80

30 in [20,60)

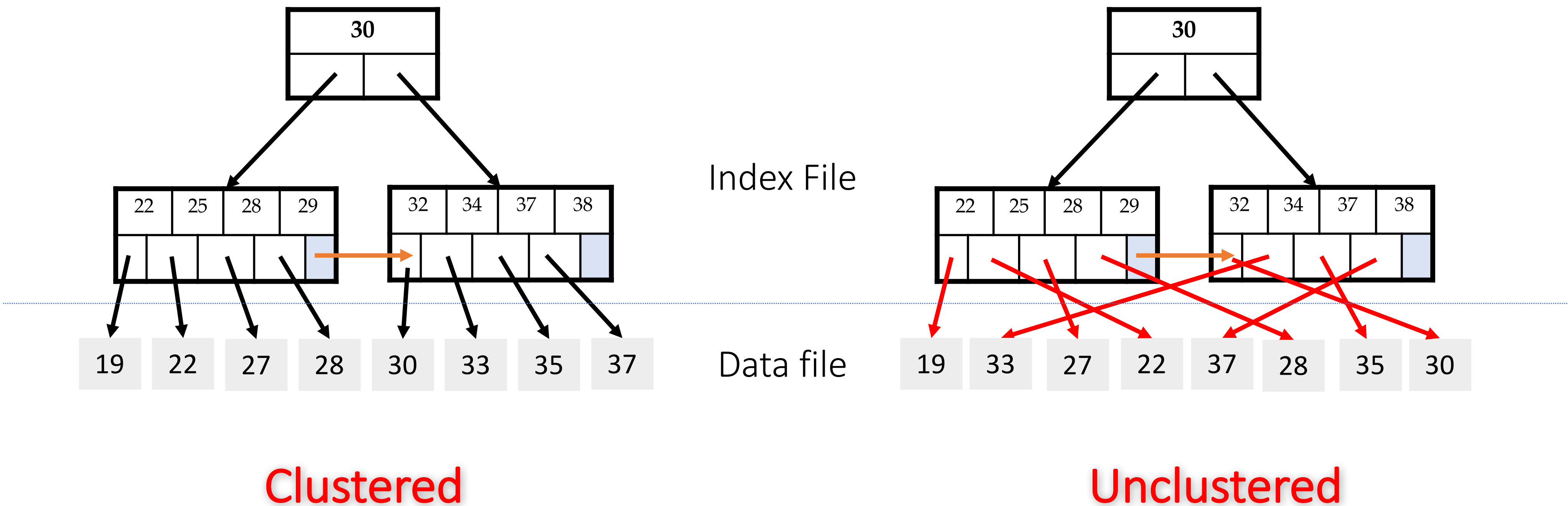
30 in [30,40)

To the data!



Not all nodes pictured

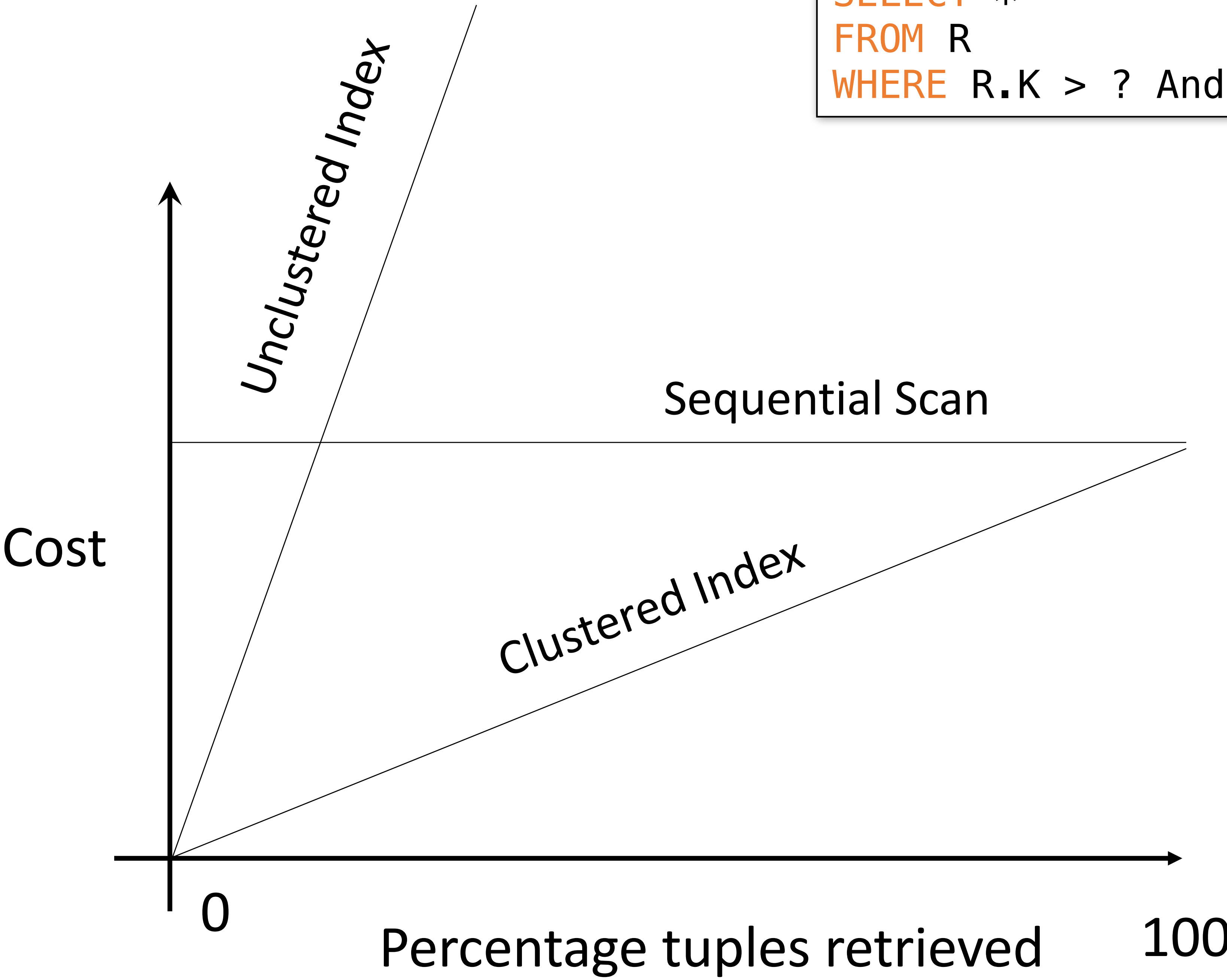
Clustered vs. Unclustered Index



Clustered vs. Unclustered Index

- Recall that for a disk with block access, **sequential IO is much faster than random IO**
- For exact search, no difference between clustered / unclustered
- For range search over R values: difference between **1 random IO + R sequential IO**, and **R random IO**

```
SELECT *
FROM R
WHERE R.K > ? And R.K < ?
```



Summary

- **Logical plans:**
  - Created by the parser from the input SQL text
  - Expressed as a relational algebra tree
  - Each SQL query has many possible logical plans
- **Physical plans:**
  - Goal is to choose an efficient implementation for each operator in the RA
  - Each logical plan has many possible physical plans
- **Query Optimization:**
  - Find the optimal logical plan
  - Find the optimal physical plan

Summary

- Index = a file that enables direct access to records in another data file
  - B+ tree / Hash table
  - Clustered/unclustered
- Data resides on disk
  - Organized in blocks
  - Sequential IO is more efficient than random IO
  - Random read 1-2% of data worse than sequential scan of the entire file

Creating Indexes in SQL

- **Offering** (<u>oID</u>, dept, cNum, term, instructor)

```
CREATE INDEX IDX1 ON Offering(dept)
```

Which query(s) could be affected by IDX1?

(A)
```
SELECT oID FROM Offering
WHERE  dept = ‘CMPT’
```

(B)
```
SELECT oID FROM Offering
WHERE  cNum = ‘354’
```

(C)
```
SELECT oID FROM Offering
WHERE  dept = ‘CMPT’ AND cNum = ‘354’
```

Creating Indexes in SQL

- **Offering** (<u>oID</u>, dept, cNum, term, instructor)

```
CREATE INDEX IDX2 ON Offering(dept, cNum)
```

Which query(s) could be affected by IDX2?

(A)
```
SELECT oID FROM Offering
WHERE  dept = ‘CMPT’
```

(B)
```
SELECT oID FROM Offering
WHERE  cNum = ‘354’
```

(C)
```
SELECT oID FROM Offering
WHERE  dept = ‘CMPT’ AND cNum = ‘354’
```

Which Indexes?

- How many indexes could we create?

- Which indexes should we create?

Which Indexes?

- The index selection problem
  - Given a table, and a “workload” (SFU CourSys 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

The Index Selection Problem 1

- Your workload is

100000 queries

```
SELECT sID
FROM Student
WHERE  name = ?
```

100000 queries

```
SELECT sID
FROM Student
WHERE  gender = ?
```

Which one is better?

A. Index on name

B. Index on gender

The Index Selection Problem 2

- Your workload is

100000 queries

```
SELECT sID
FROM Student
WHERE  name like ?
```

100000 queries

```
SELECT sID
FROM Student
WHERE  age = ?
```

Which one is better?

A. Index on name

B. Index on age

The Index Selection Problem 3

- Your workload is

100000 queries

```
SELECT sID
FROM Student
WHERE  name = ?
```

100 queries

```
SELECT sID
FROM Student
WHERE  age = ?
```

Which one(s) are useful?

A. Index on name

B. Index on age

C. Index on name, age

D. Index on age, name

The Index Selection Problem 4

- Your workload is

100000 queries

```
SELECT sID
FROM Student
WHERE  fname = ?
```

100000 queries

```
SELECT sID
FROM Student
WHERE fname = ? AND age > ?
```

Which one is better?

A. Index on (fname, age)

B. Index on (age, fname)

The Index Selection Problem 5

- Your workload:

100000 queries

```
SELECT sID
FROM Student
WHERE  name = ?
```

100 queries

```
SELECT sID
FROM Student
WHERE  age = ?
```

100000 queries

```
INSERT INTO Student
VALUES (?, …, ?)
```

Which one(s) are useful?

A. Index on name

B. Index on age

C. Index on name, age

D. Index on age, name

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

Summary

- Query Processing
  - SQL Parser
  - Logical Optimization
  - Physical Optimization
  - Query Execution

- Indexing
  - Data Storage
  - Index motivation
  - Index Selection

Acknowledge

- Some lecture slides were copied from or inspired by the following course materials
  - “W4111: Introduction to databases” by Eugene Wu at Columbia University
  - “CSE344: Introduction to Data Management” by Dan Suciu at University of Washington
  - “CMPT354: Database System I” by John Edgar at Simon Fraser University
  - “CS186: Introduction to Database Systems” by Joe Hellerstein at UC Berkeley
  - “CS145: Introduction to Databases” by Peter Bailis at Stanford
  - “CS 348: Introduction to Database Management” by Grant Weddell at University of Waterloo