DATA 514

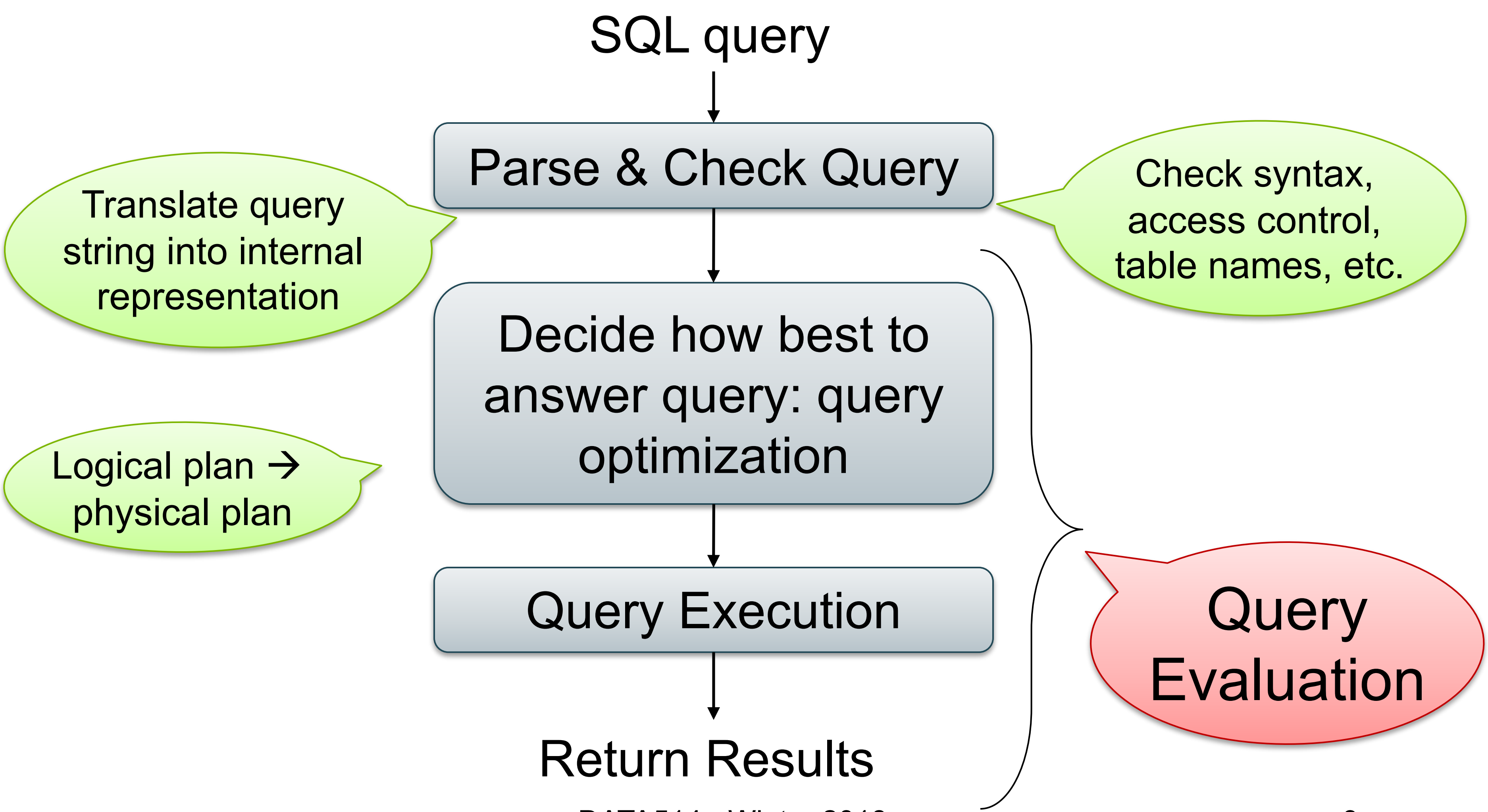
Lecture 4:
Query Execution and Indexes

Announcements

- WQ4 is out – due next Sunday
- HW3- due Feb 3

- Midterm, next Tuesday(Feb. 6)
- Check out past exams in our website
  - Note that the material covered in past exams is not necessarily the same as that covered in our exams

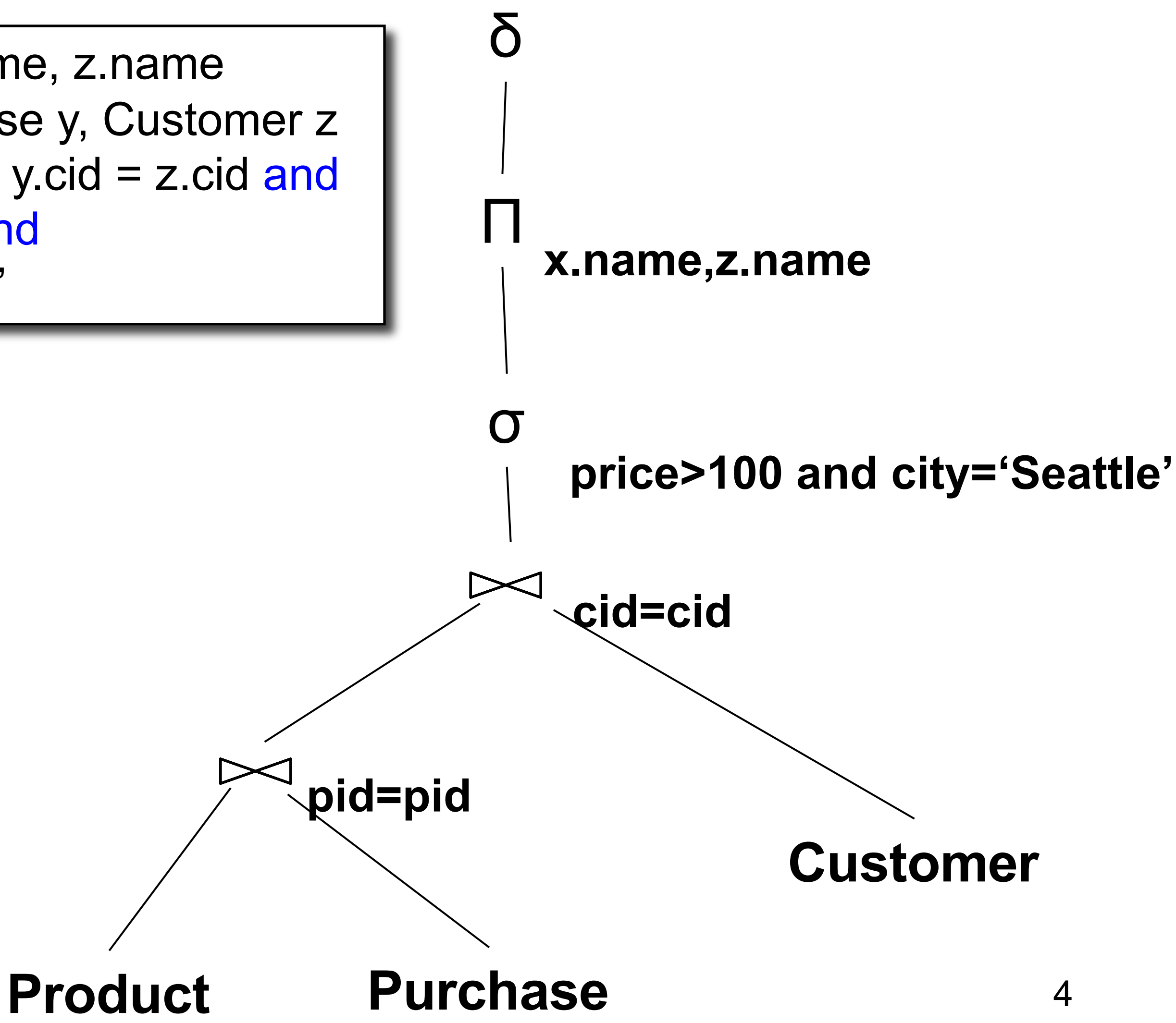
Query Evaluation Steps



Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
           x.price > 100 and
          z.city =  'Seattle'
```

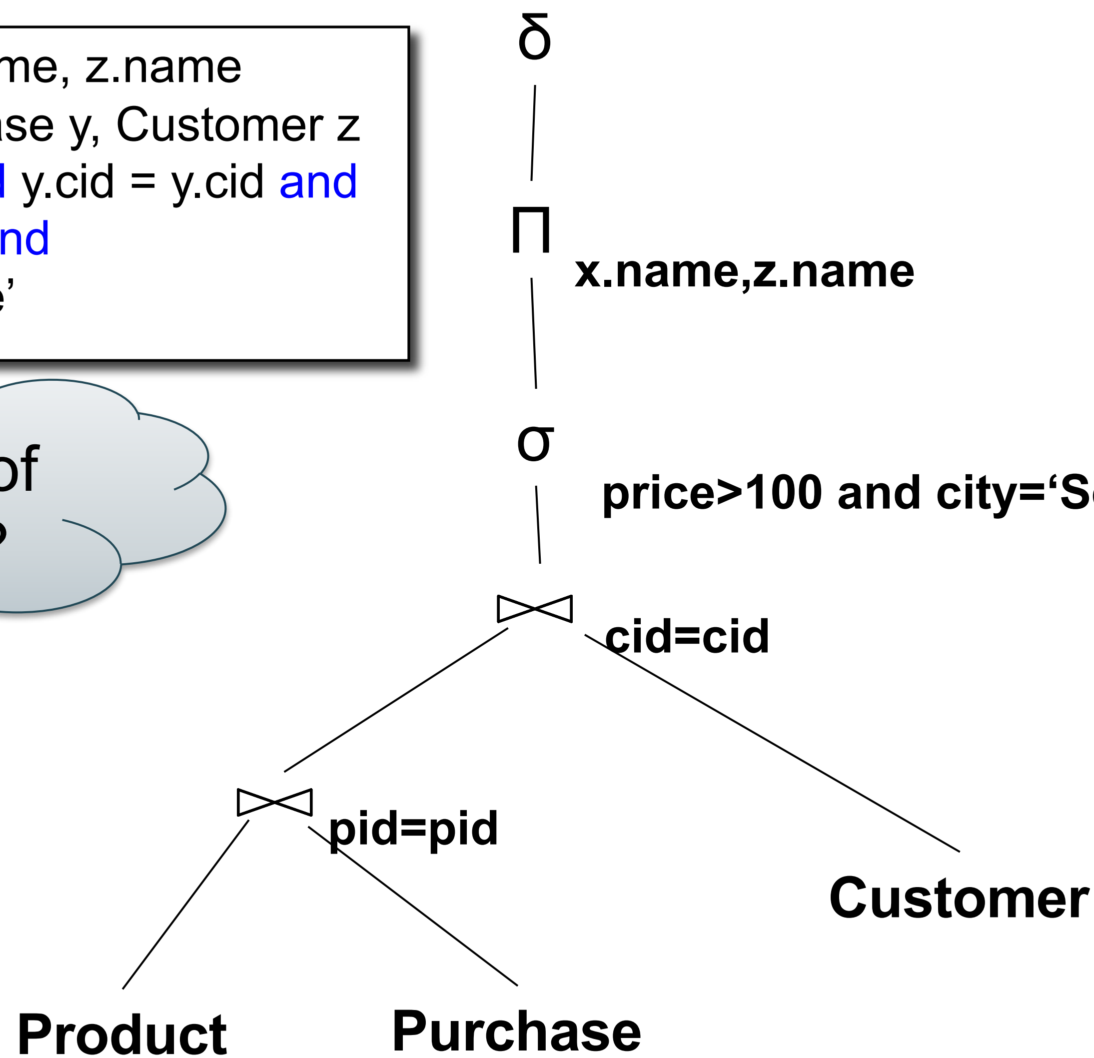


Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
          x.price > 100 and
          z.city =  'Seattle'
```

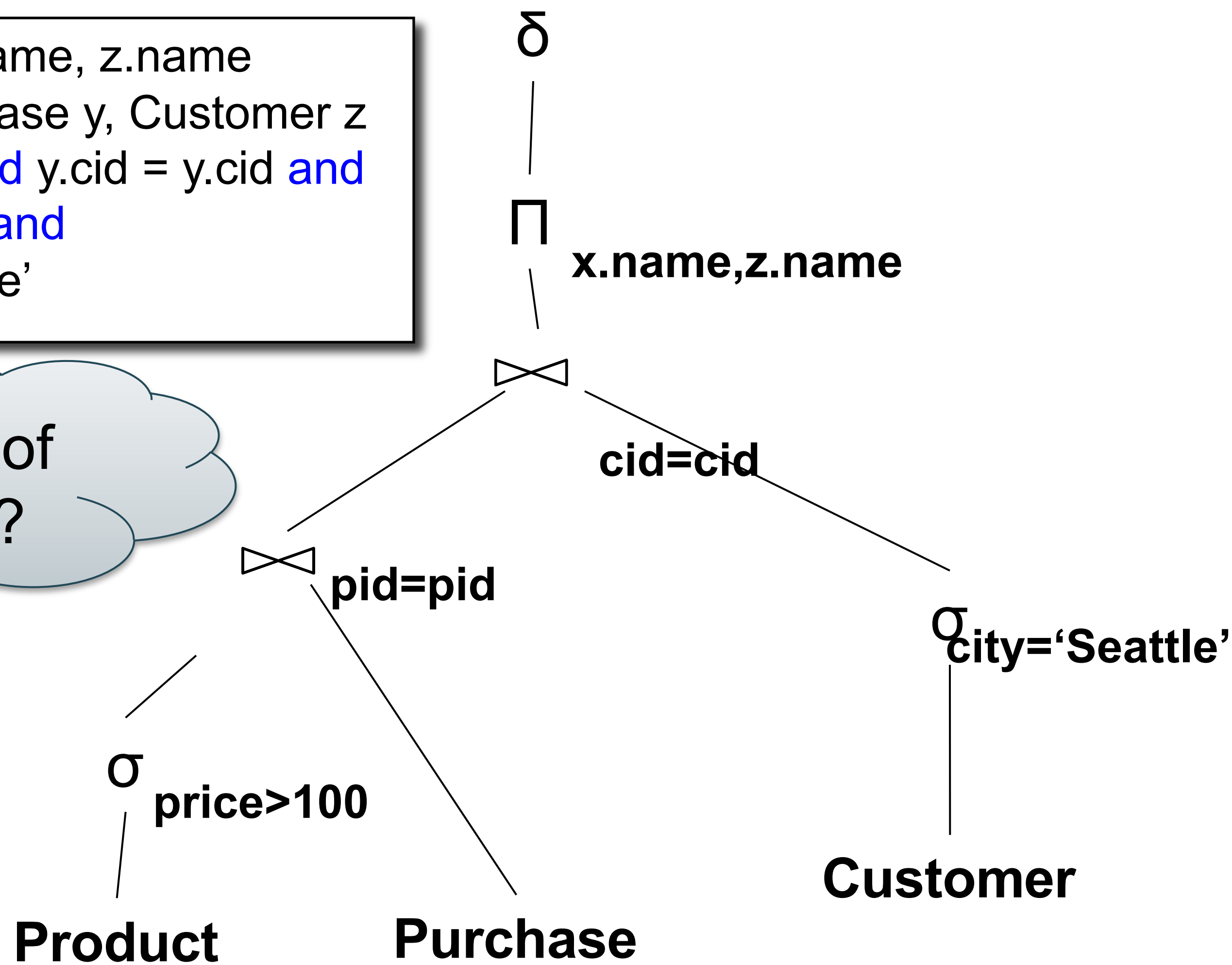
Can you think of a "better" plan?



Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
          x.price > 100 and
          z.city =  'Seattle'
```



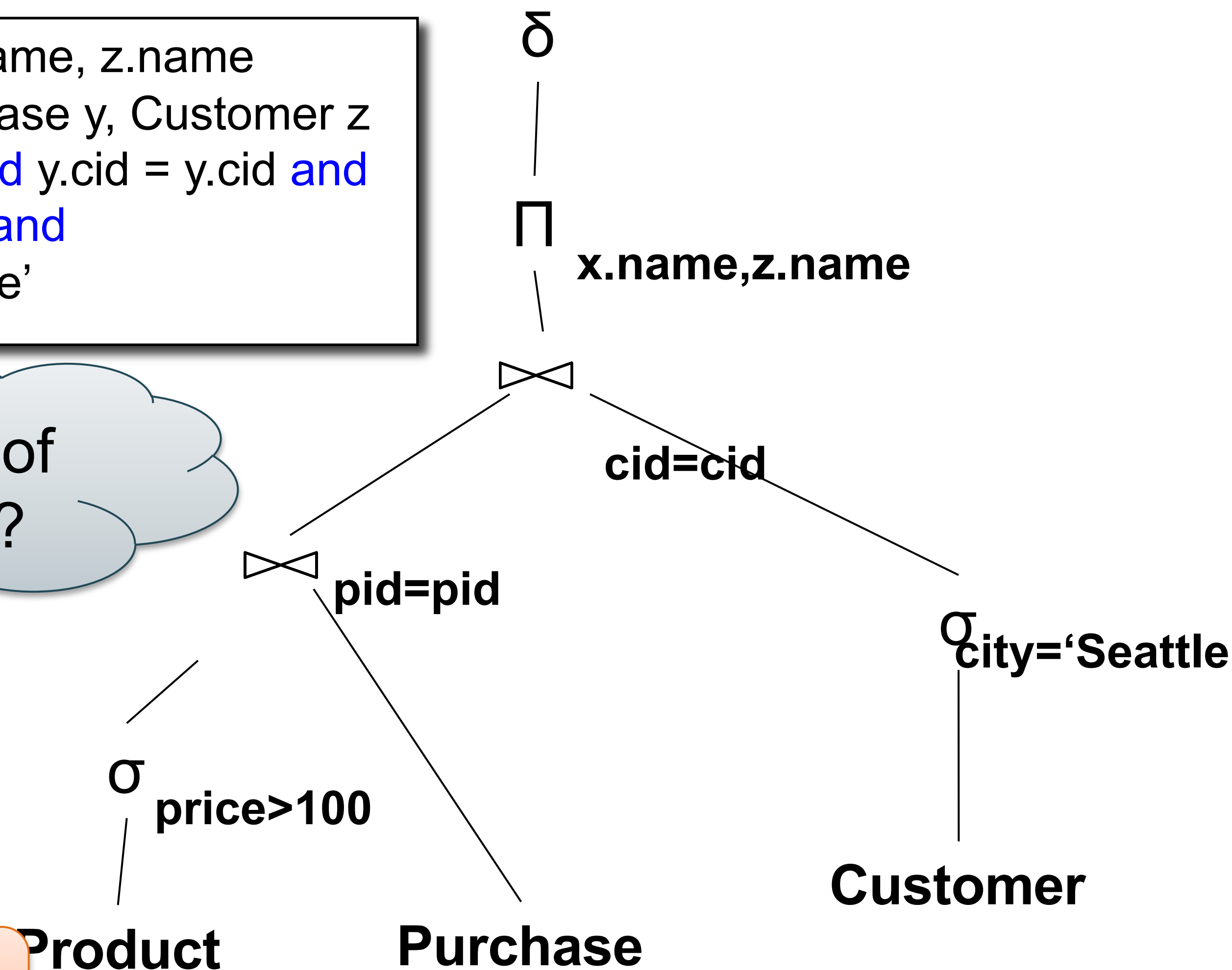
Can you think of a "better" plan?

Push selections down the query plan!

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
        x.price > 100 and
        z.city = 'Seattle'
```



Can you think of a "better" plan?

Push selections down the query plan!

Query optimization: find an equivalent optimal plan

From Logical Plans to Physical Plans

Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join

Main Memory Algorithms

Logical operator:

Product(pid, name, price) $\bowtie_{pid=pid}$ Purchase(pid, cid, store)

Propose three physical operators for the join, assuming the tables are in main memory:

1.
2.
3.

Main Memory Algorithms

Logical operator:

Product(pid, name, price) $\bowtie_{pid=pid}$ Purchase(pid, cid, store)

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join O(??)
2. Merge join O(??)
3. Hash join O(??)

Product(pid, name, price)
Purchase(pid, cid, store)

Main Memory Algorithms

Logical operator:

Product(pid, name, price) $\bowtie_{pid=pid}$ Purchase(pid, cid, store)

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join $O(n^2)$
2. Merge join $O(n \log n)$
3. Hash join $O(n) \ldots O(n^2)$

BRIEF Review of Hash Tables

A (naïve) hash function:

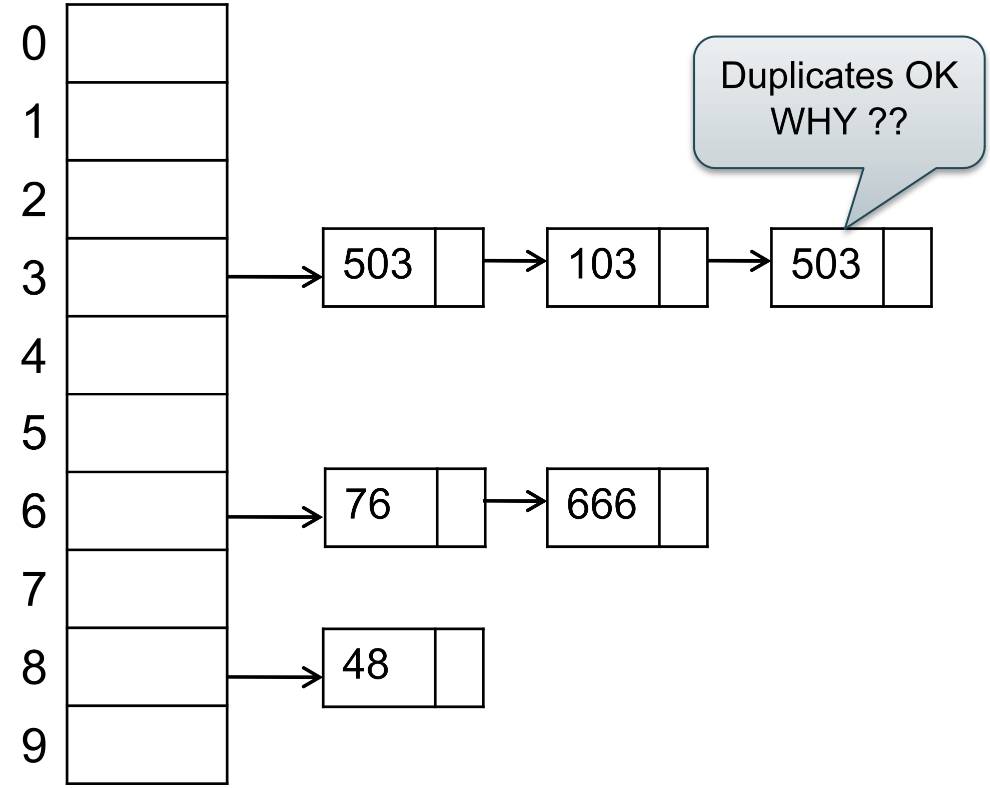
$h(x) = x \bmod 10$

Operations:

find(103) = ??

insert(488) = ??

Separate chaining:



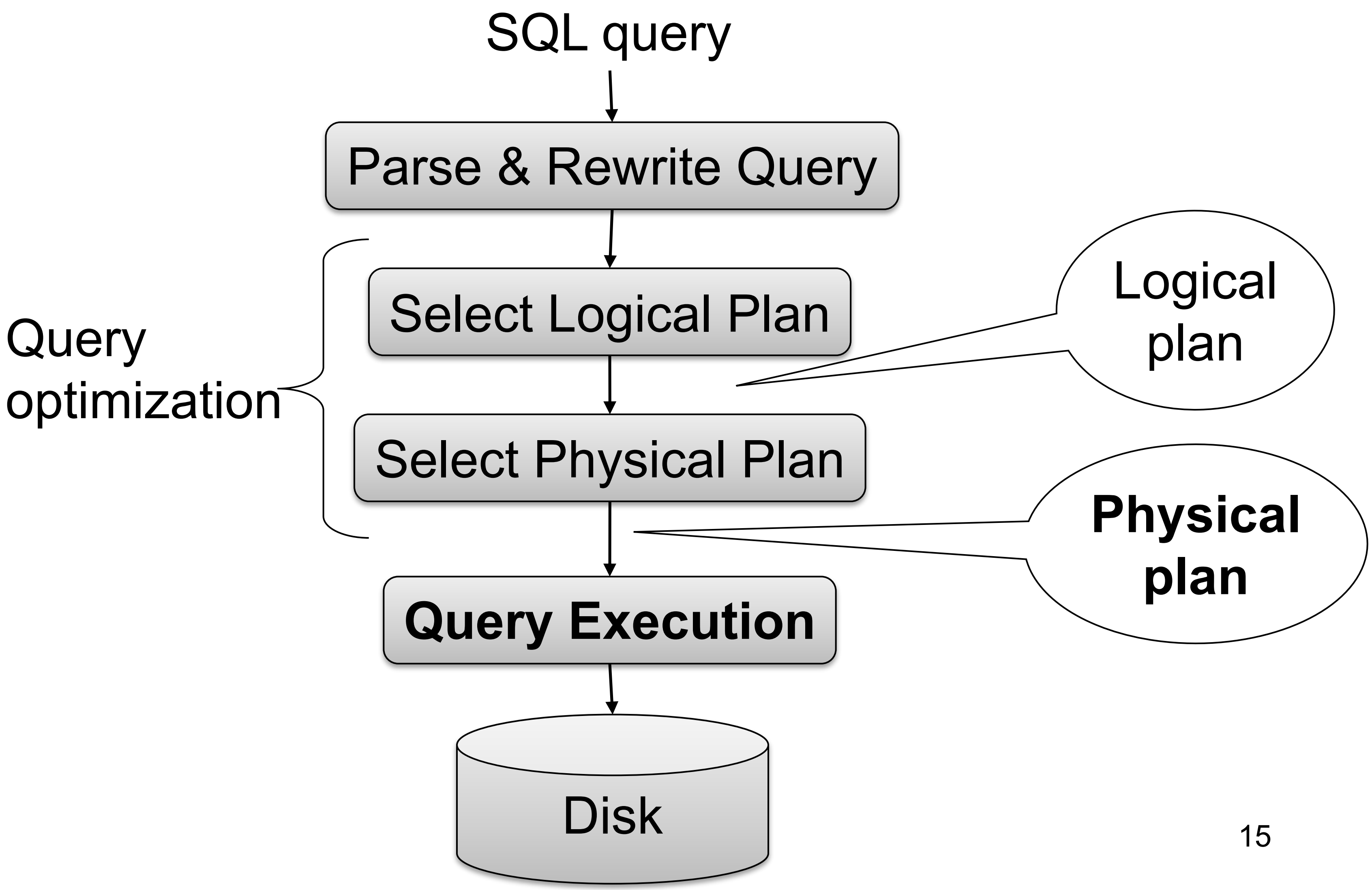
BRIEF Review of Hash Tables

- insert(k, v) = inserts a key k with value v

- Many values for one key
  – Hence, duplicate k's are OK

- find(k) = returns the ***list*** of all values v associated to the key k

Query Evaluation Steps Review



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

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'
```

Give a relational algebra expression for this query

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

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'
```

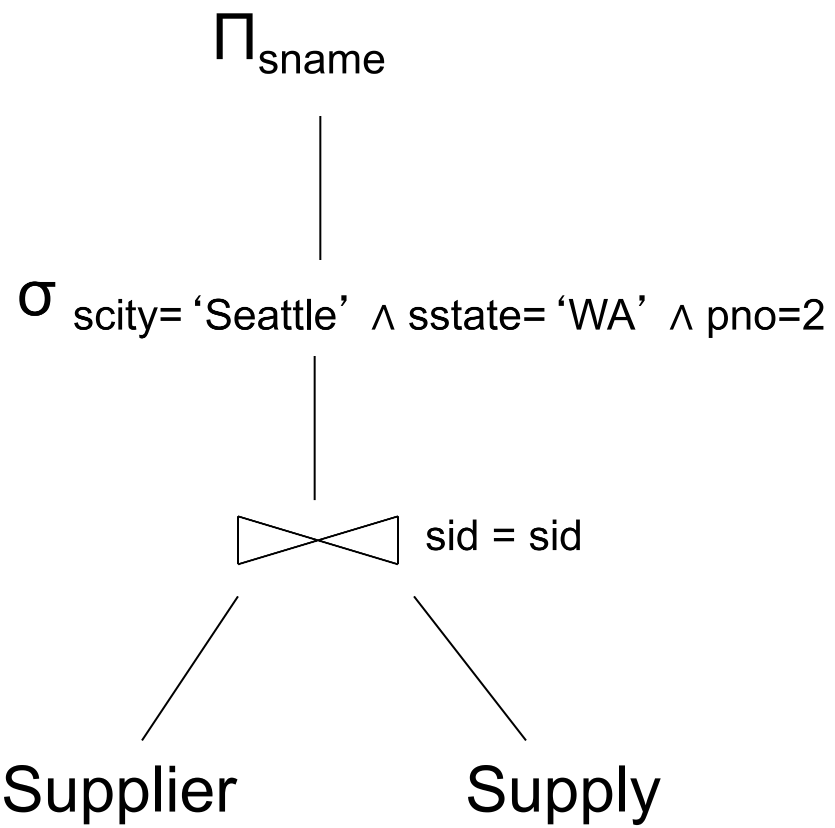
$$\Pi_{sname}(\sigma_{scity=\text{'Seattle'} \wedge sstate=\text{'WA'} \wedge pno=2}(\text{Supplier} \bowtie_{sid = sid} \text{Supply}))$$

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

Relational Algebra

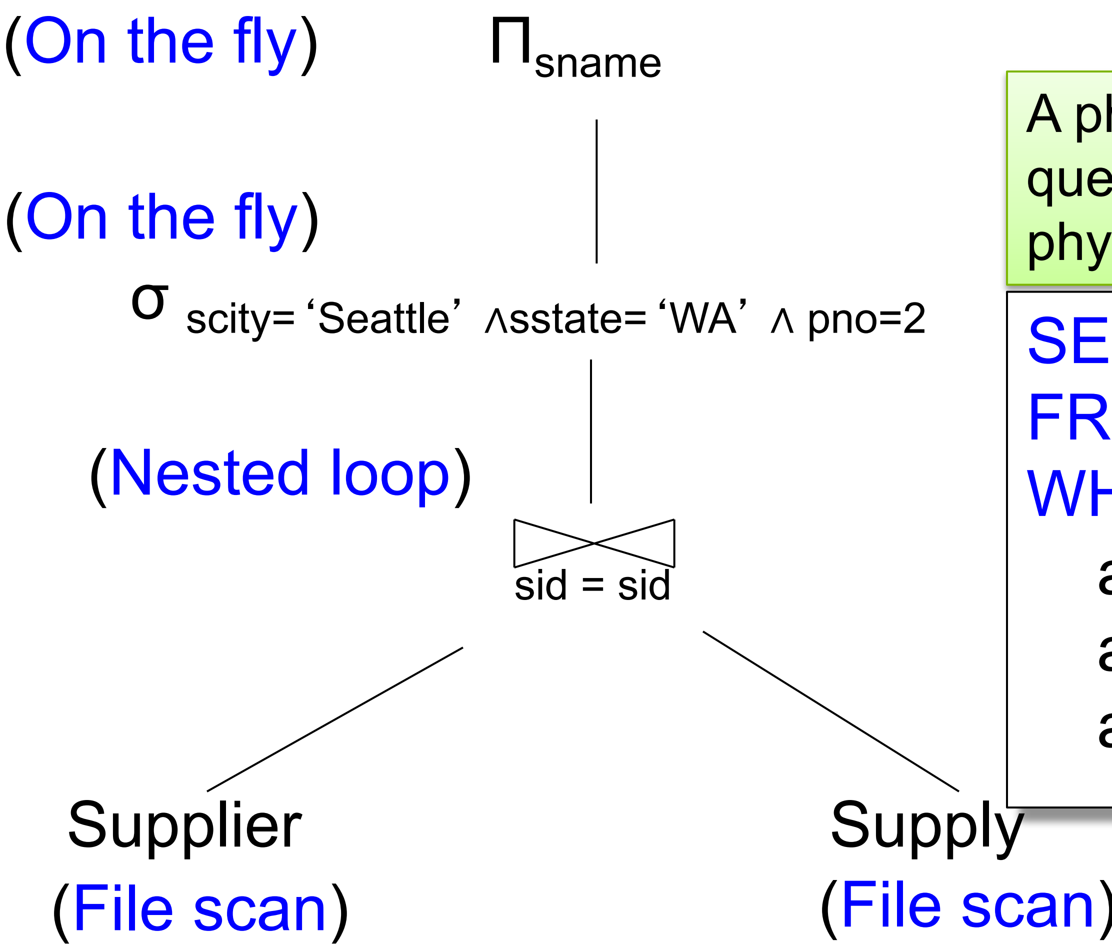
```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity =  'Seattle'
    and x.sstate =  'WA'
```

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



Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Physical Query Plan 1

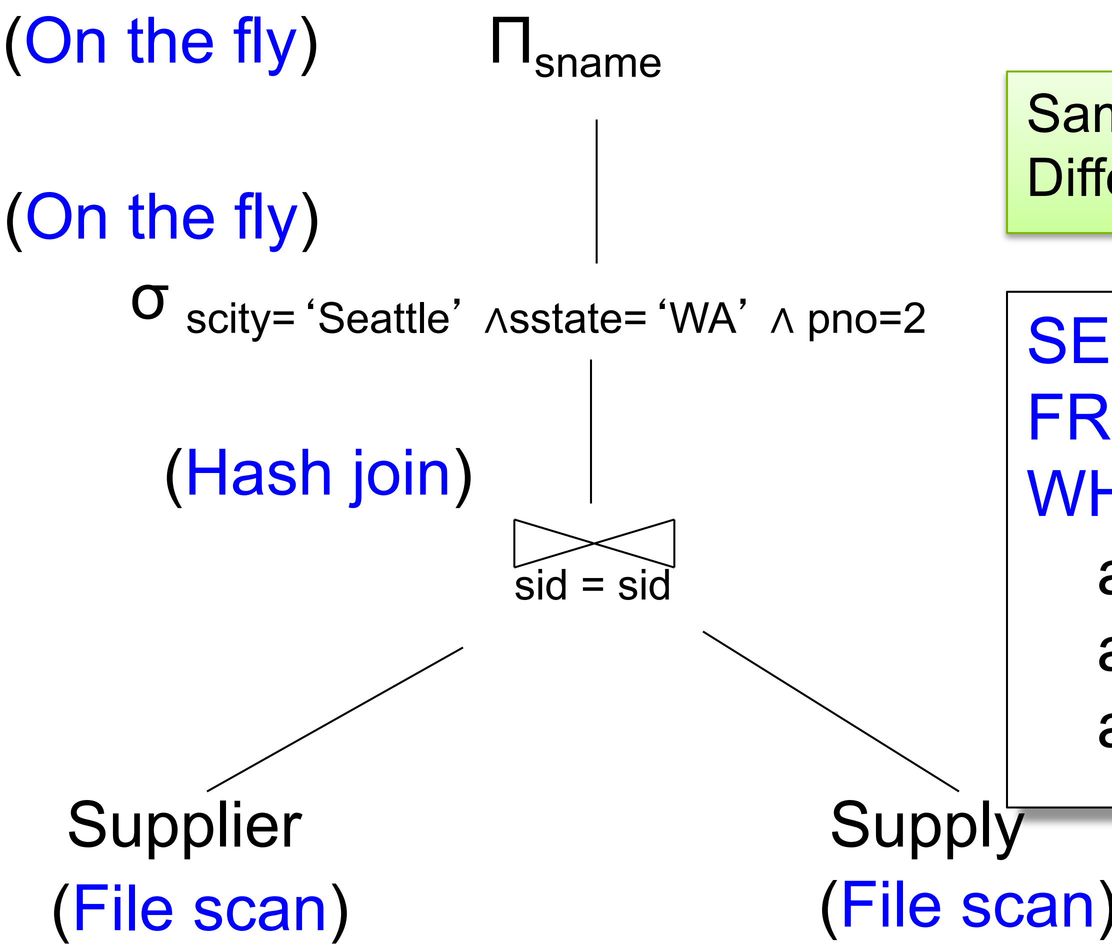


A physical query plan is a logical query plan annotated with physical implementation details

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

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

Physical Query Plan 2

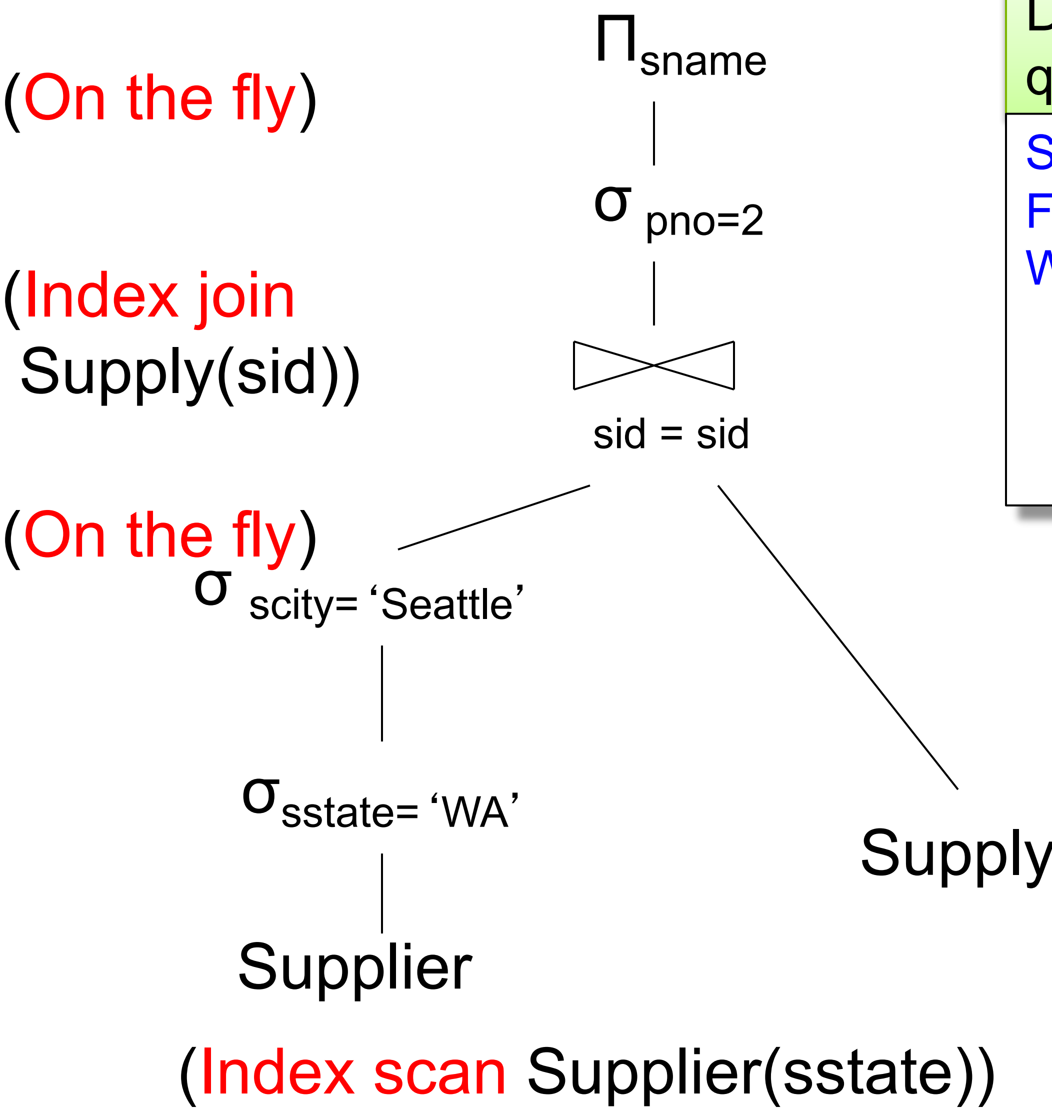


Same logical query plan
Different physical plan

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

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

Physical Query Plan 3

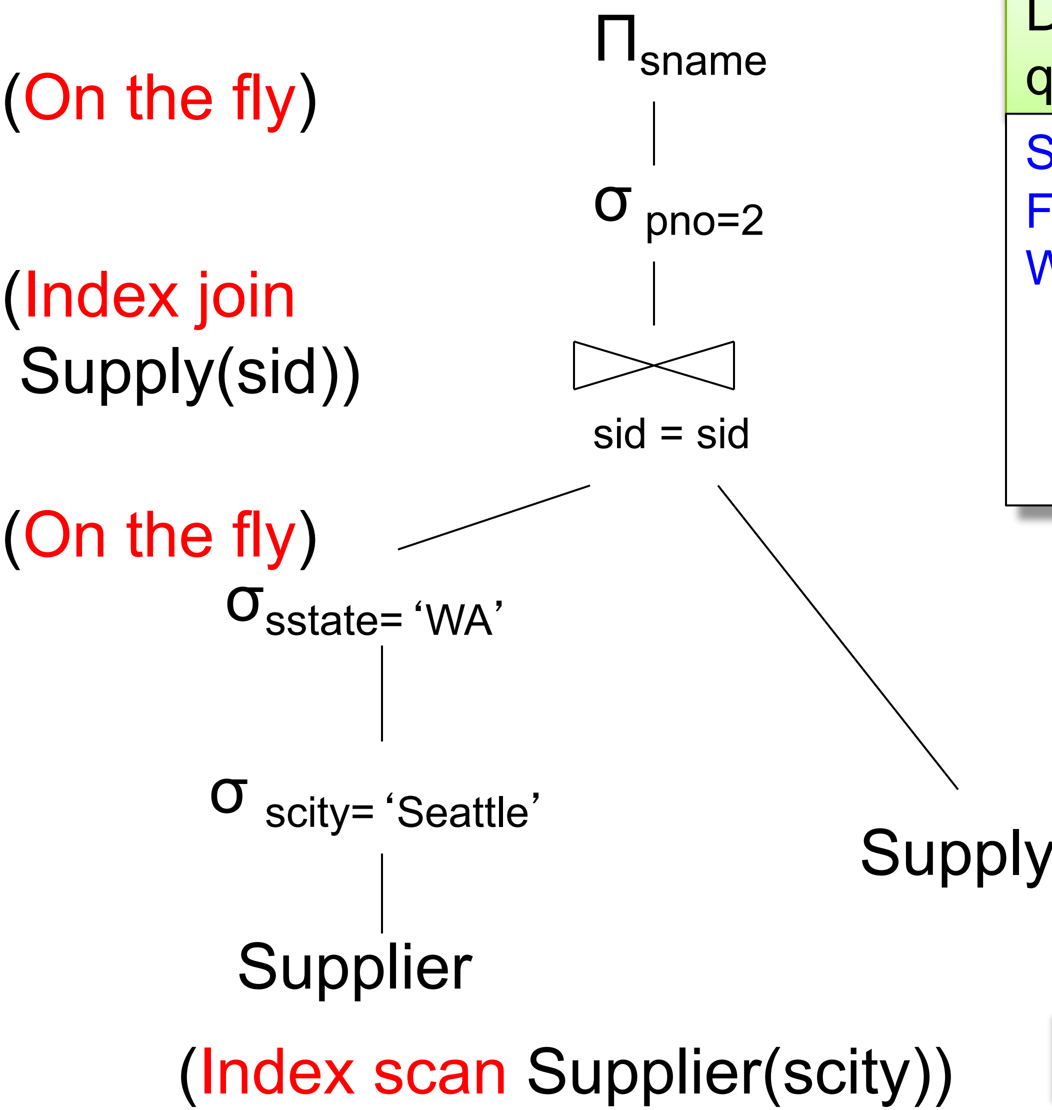


Different but equivalent logical query plan; different physical plan

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

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

Physical Query Plan 4



Different but equivalent logical query plan; different physical plan

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

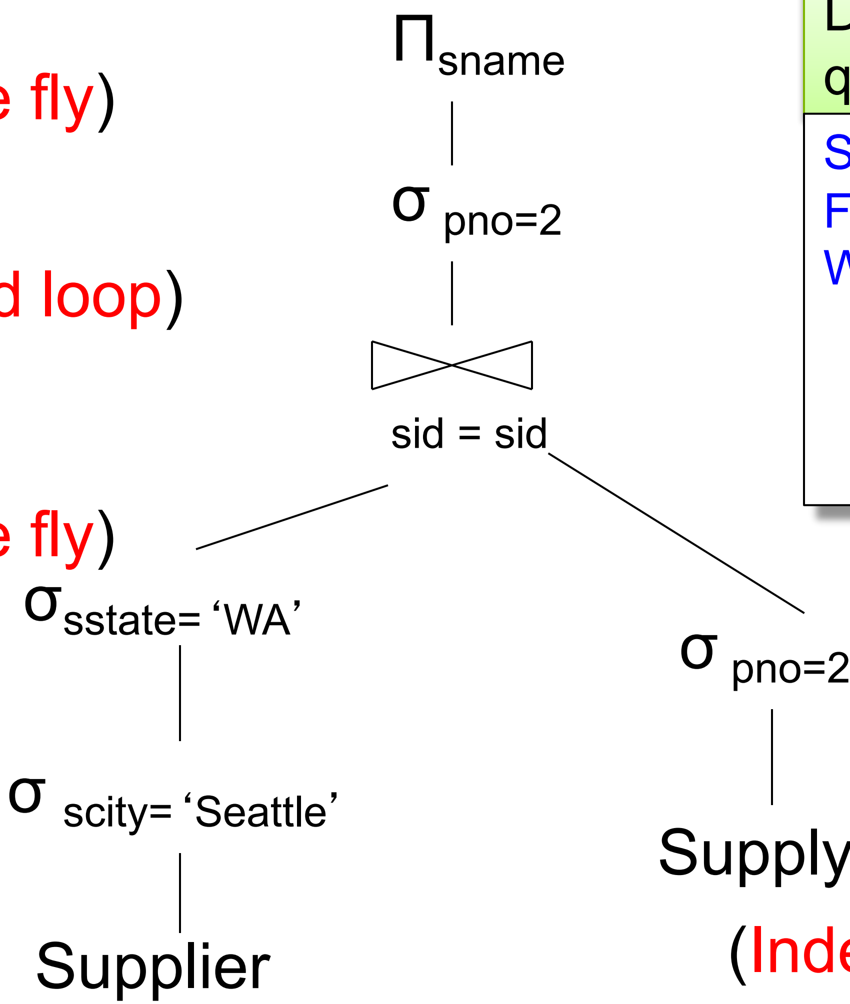
← Note difference from Plan 3

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

Physical Query Plan 5

Different but equivalent logical query plan; different physical plan

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



(On the fly)

(Nested loop)

(On the fly)

(Index scan Supply(pno))

(Index scan Supplier(scity))

Query Optimization Problem

- For each SQL query… many logical plans
- For each logical plan… many physical plans
- Optimizer examines multiple equivalent plans, chooses one with minimum cost

Query Execution

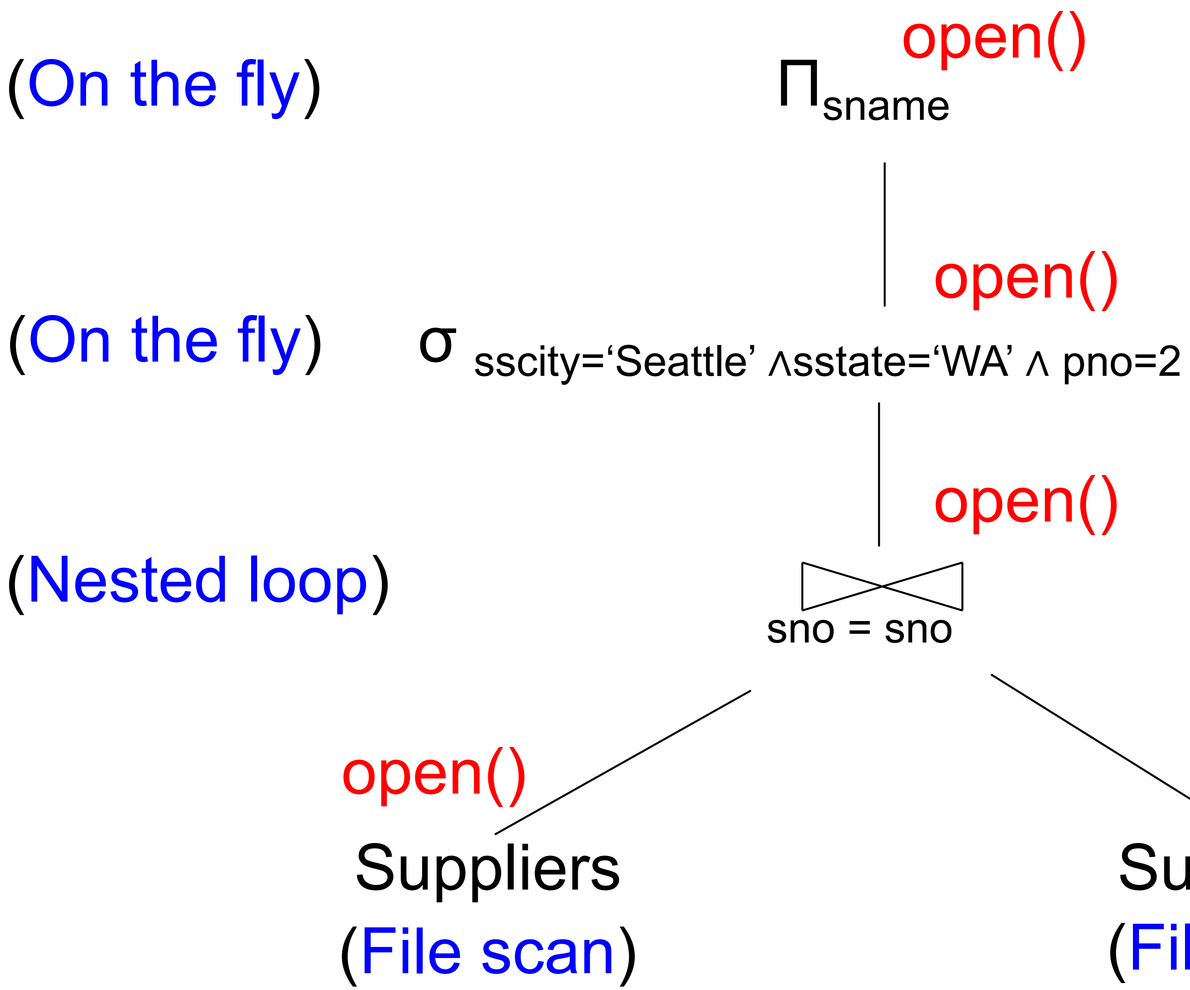
Iterator Interface

- Iterators:
  - Do not materialize intermediate results
  - Children pipeline their results to parents
- Every physical operator maintains its own execution state and implements the following methods
  - open(): Initialize state and get ready for processing
  - next() Operator invokes get_next() recursively on its inputs; Performs processing and produces an output tuple
  - close(): clean-up state

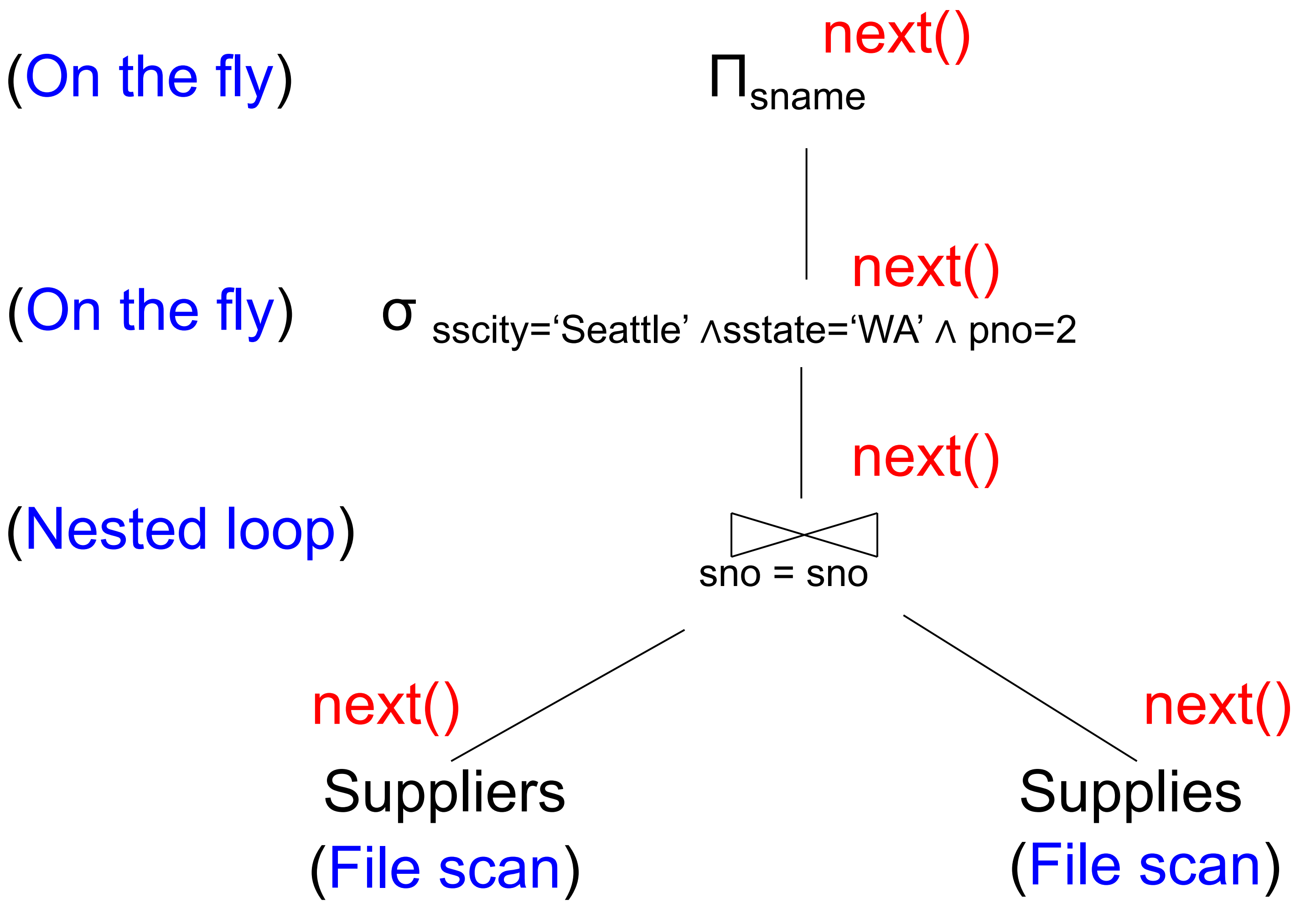
An iterator for file scan

- state: a block of memory for buffering input a pointer to a tuple within the block
- open(): allocate a block of memory
- next():
  - If no block of has been read yet, read the first block from the disk and return the first tuple in the block
  - If there is no more tuple left in the current block, read the next block of from the disk and return the first tuple in the block
  - Otherwise, return the next tuple in the memory block
- close(): deallocate the block of memory

Pipelined Query Execution



Pipelined Query Execution



Pipelined Execution

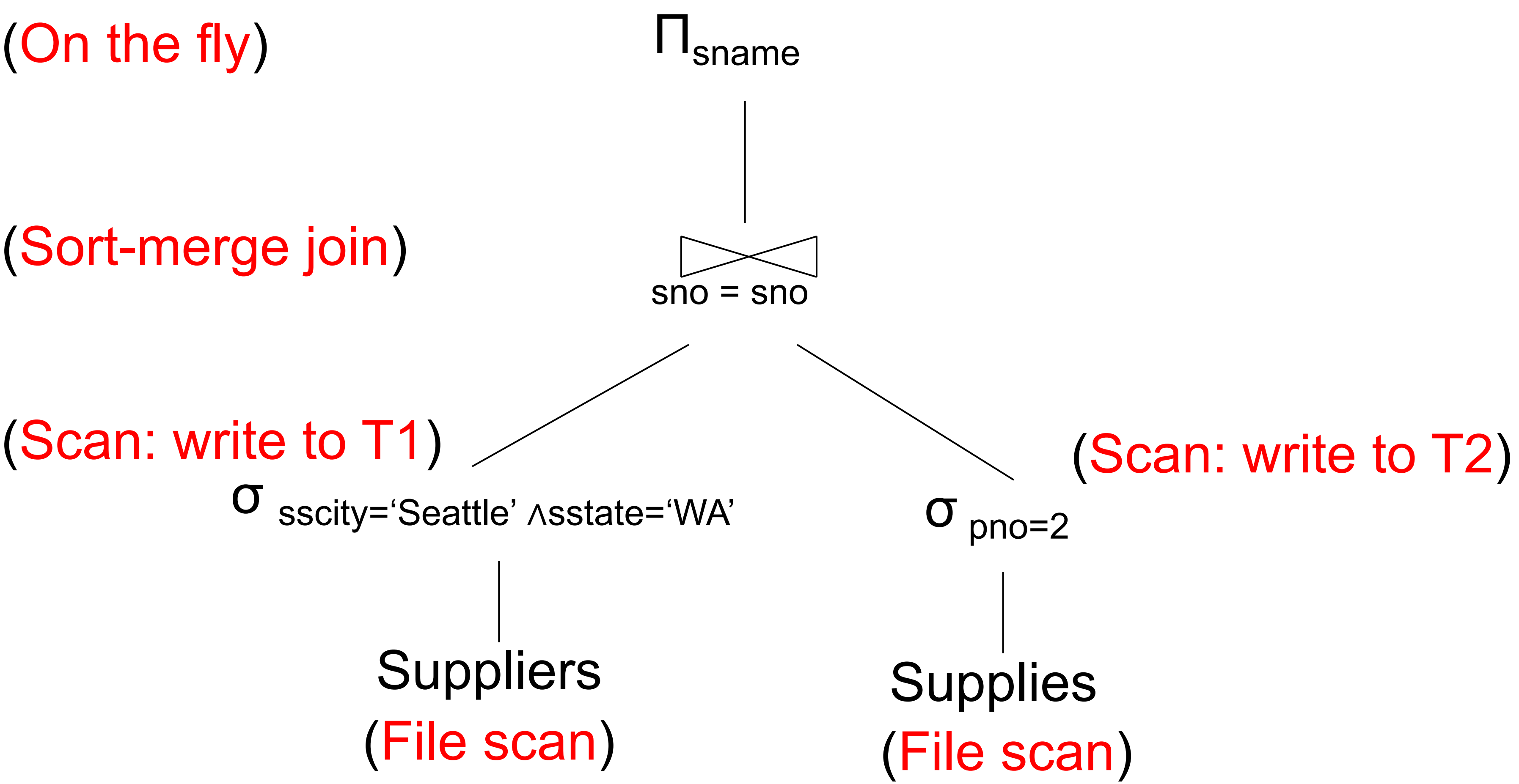
- Tuples generated by an operator are immediately sent to the parent
- Benefits:
  - No operator synchronization issues
  - No need to buffer tuples between operators
  - Saves cost of writing intermediate data to disk
  - Saves cost of reading intermediate data from disk
- This approach is used whenever possible

Intermediate Tuple Materialization

- Tuples generated by an operator are written to disk an in intermediate table

- No direct benefit
- Necessary:
  - For certain operator implementations
  - When we don’t have enough memory

Intermediate Tuple Materialization



Query Execution Bottom Line

- SQL query transformed into physical plan
  - **Access path selection** for each relation
    - Scan the relation or use an index (rest of this lecture)
  - **Implementation choice** for each operator
    - Nested loop join, hash join, etc.
  - **Scheduling decisions** for operators
    - Pipelined execution or intermediate materialization

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Data Storage

- 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	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*

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

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

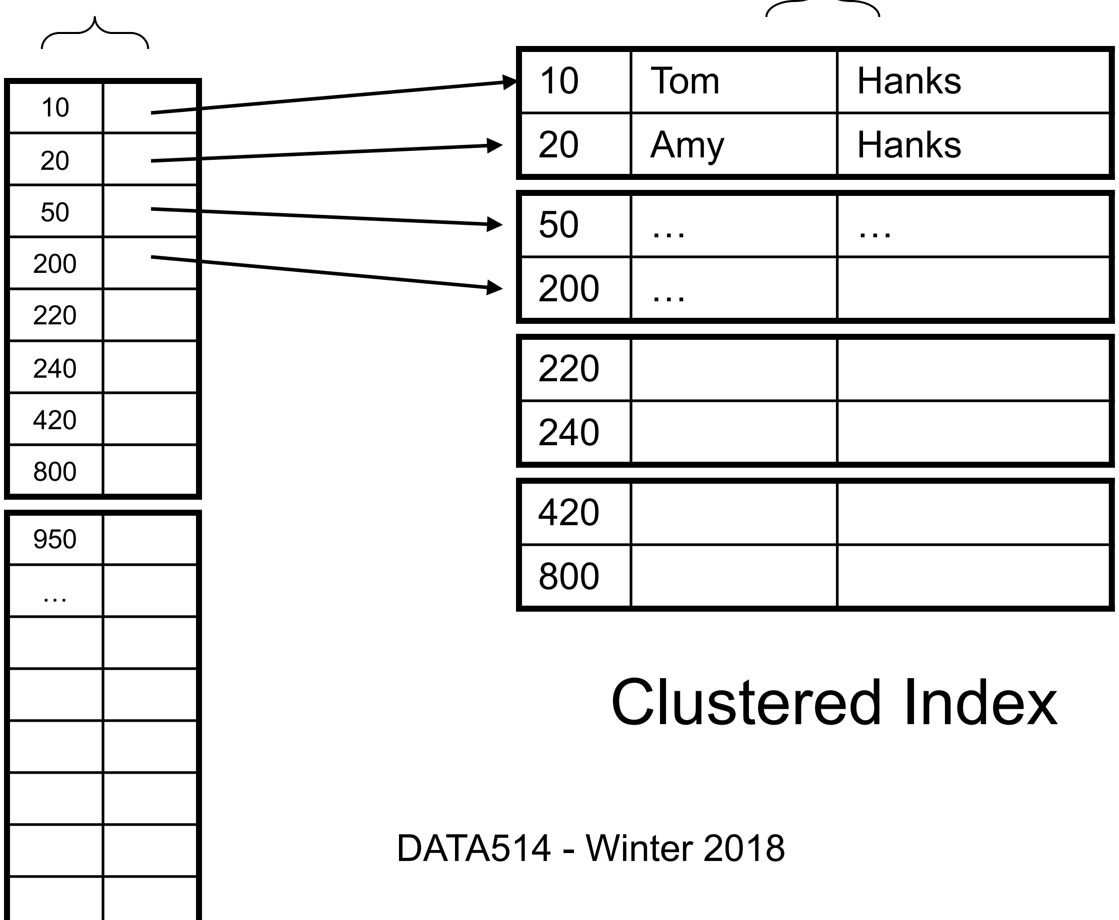
Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Example 1: Index on ID

Index **Student_ID** on **Student.ID**

Data File **Student**



Clustered Index

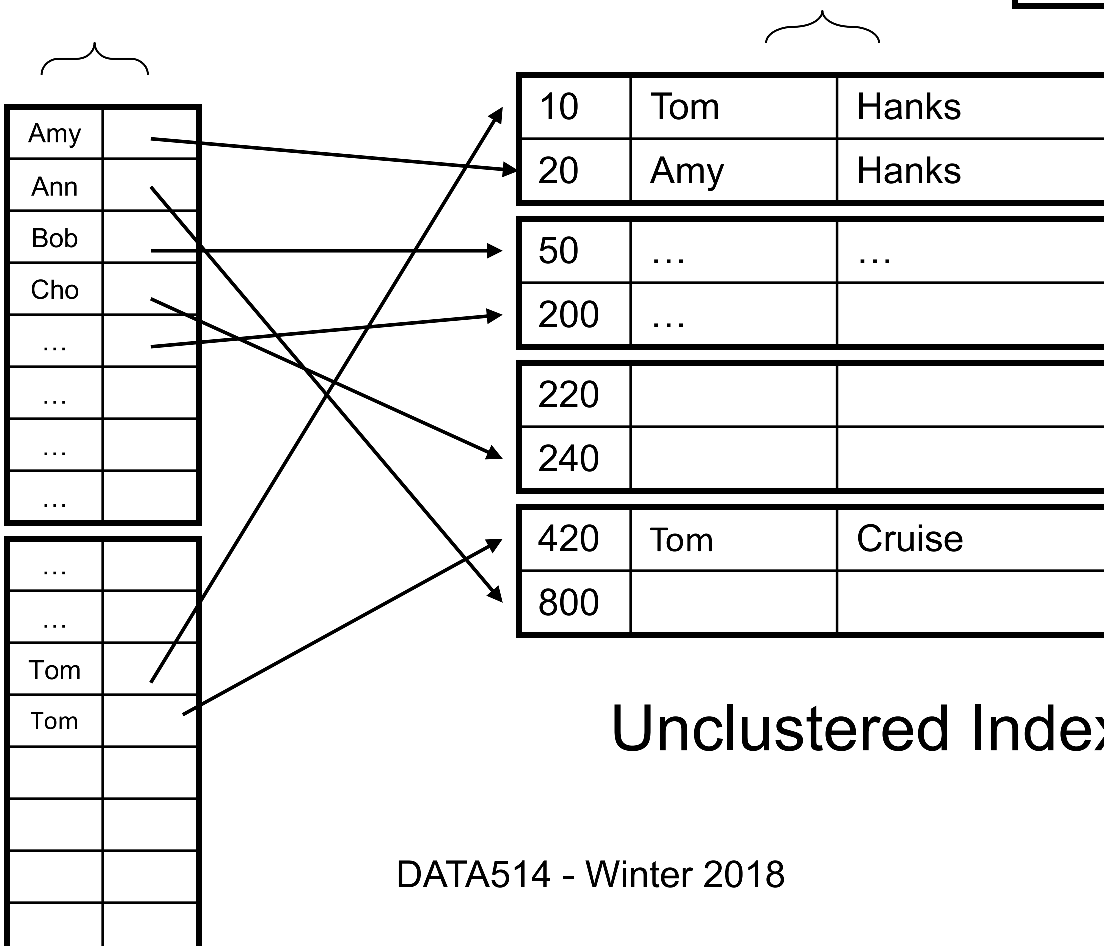
Example 2: Index on fName

Student

ID	fName	lName
10	Tom	Hanks
20	Tom	Cruise
…	Amy	Hanks

Index **Student_fName** on **Student.fName**

Data File **Student**



Unclustered Index

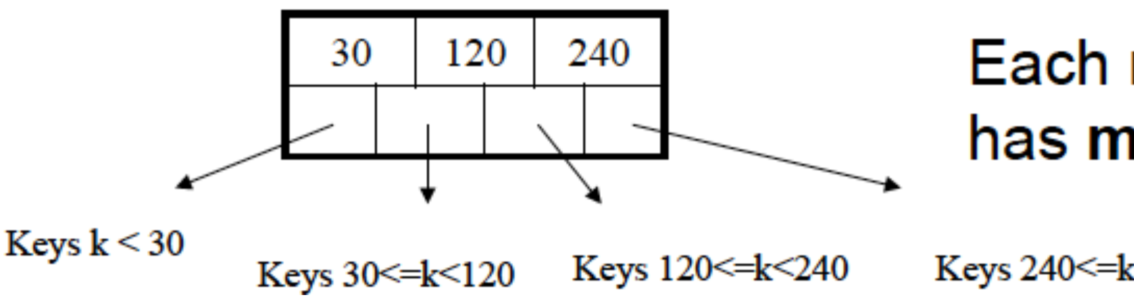
Index Organization

Several index organizations:

- 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

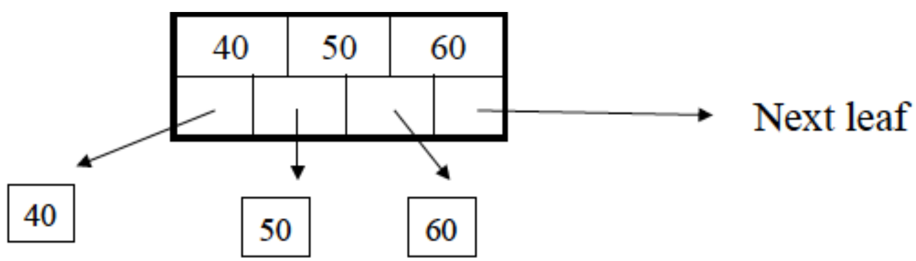
B+ Trees Basics

- Parameter d = the degree
- Each node has d<= m<= 2d keys (except root)



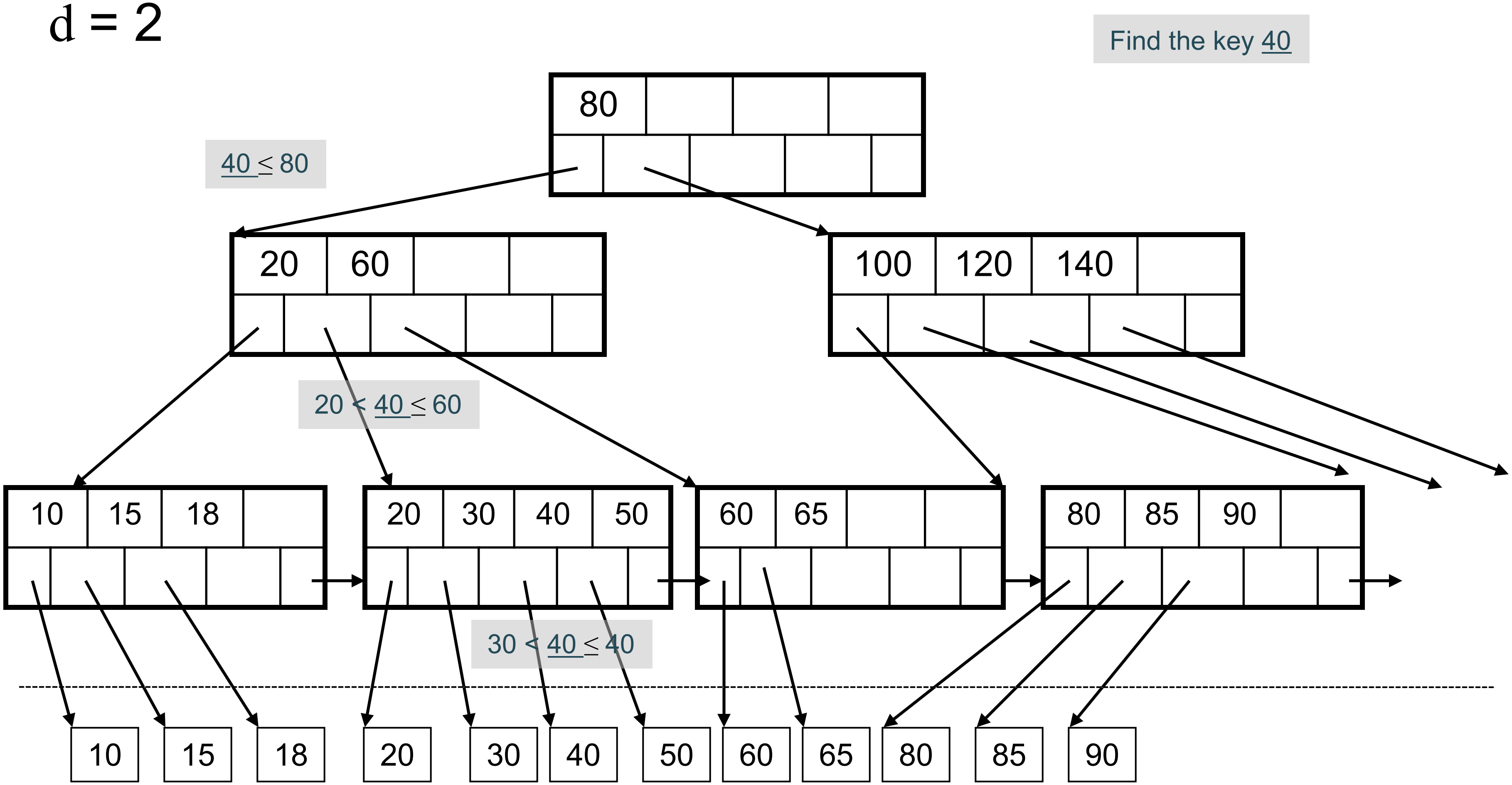
Each node also has **m+1 pointers**

- Each leaf has d<= m<= 2d keys



B+ Tree Index by Example

d = 2



Searching a B+ Tree

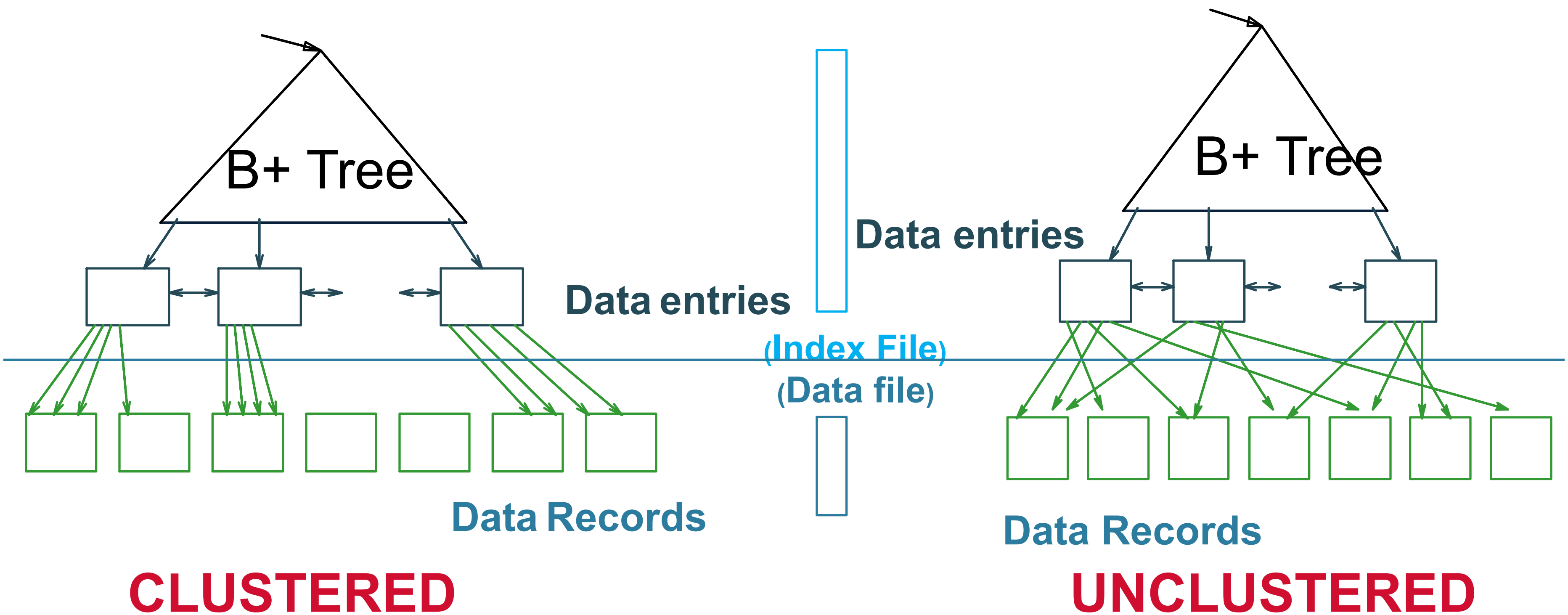
- Exact key values:
  - Start at the root
  - Proceed down, to the leaf

```
Select name
From people
Where age = 25
```

- Range queries:
  - As above
  - Then sequential traversal

```
Select name
From people
Where 20 <= age
  and  age <= 30
```

Clustered vs Unclustered



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

Scanning a Data File

- 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, …

Scanning a Data File

- 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
  - Solid state (SSD): $$$ expensive; put indexes, other "hot" data there, not enough room for everything

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

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Which Indexes?

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

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Which Indexes?

- How many indexes could we create?

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

- Which indexes should we create?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Which Indexes?

- 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

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

Which Indexes?

- How many indexes could we create?

18, 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:
  - The database administrator DBA
  - Semi-automatically, using a database administration tool

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:
  - 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

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

Your workload is this

100000 queries:

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

100 queries:

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

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)

How does this index differ from:

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

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:

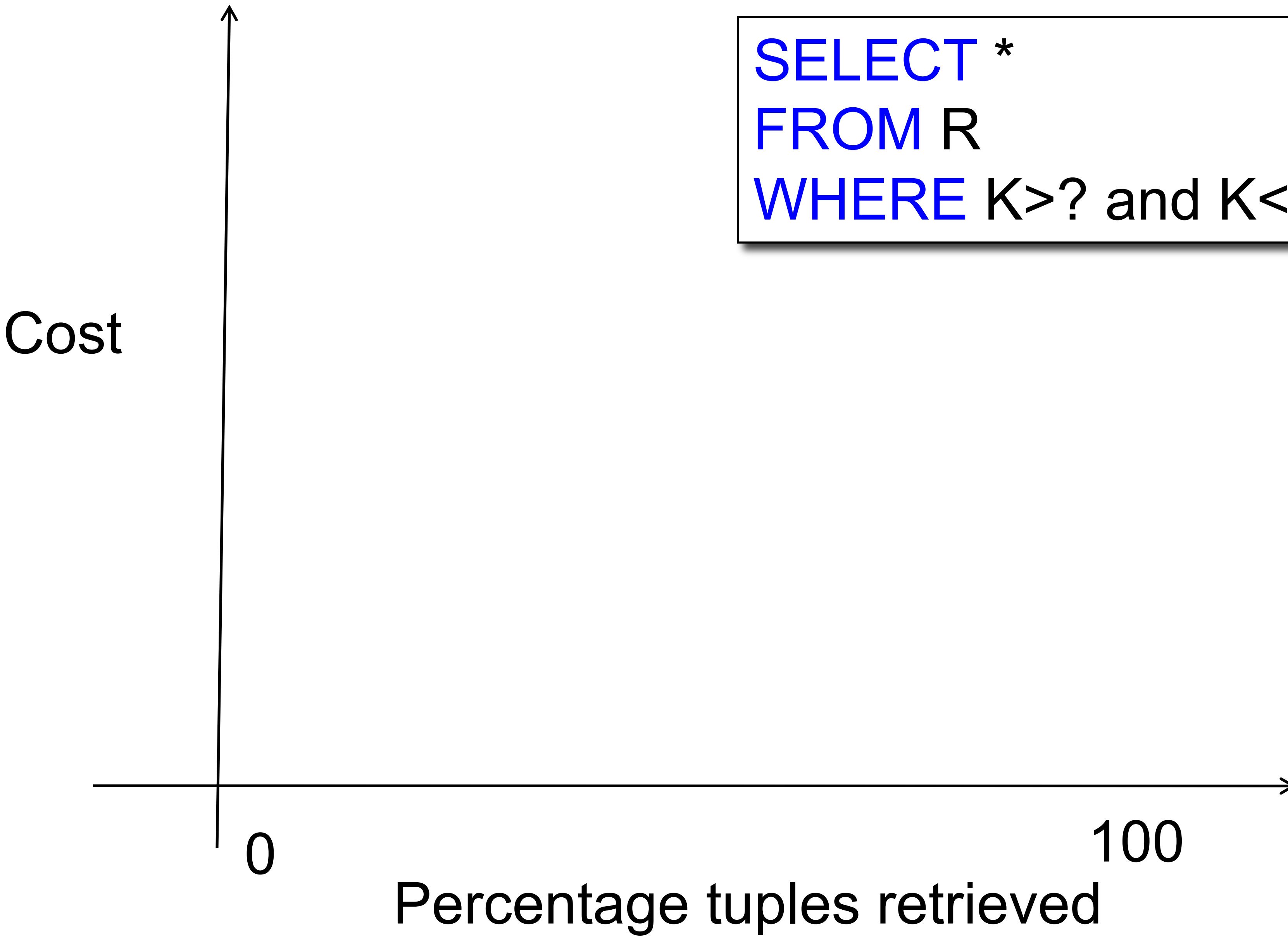
```
SELECT *
FROM V
WHERE P>? and P<?
```

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

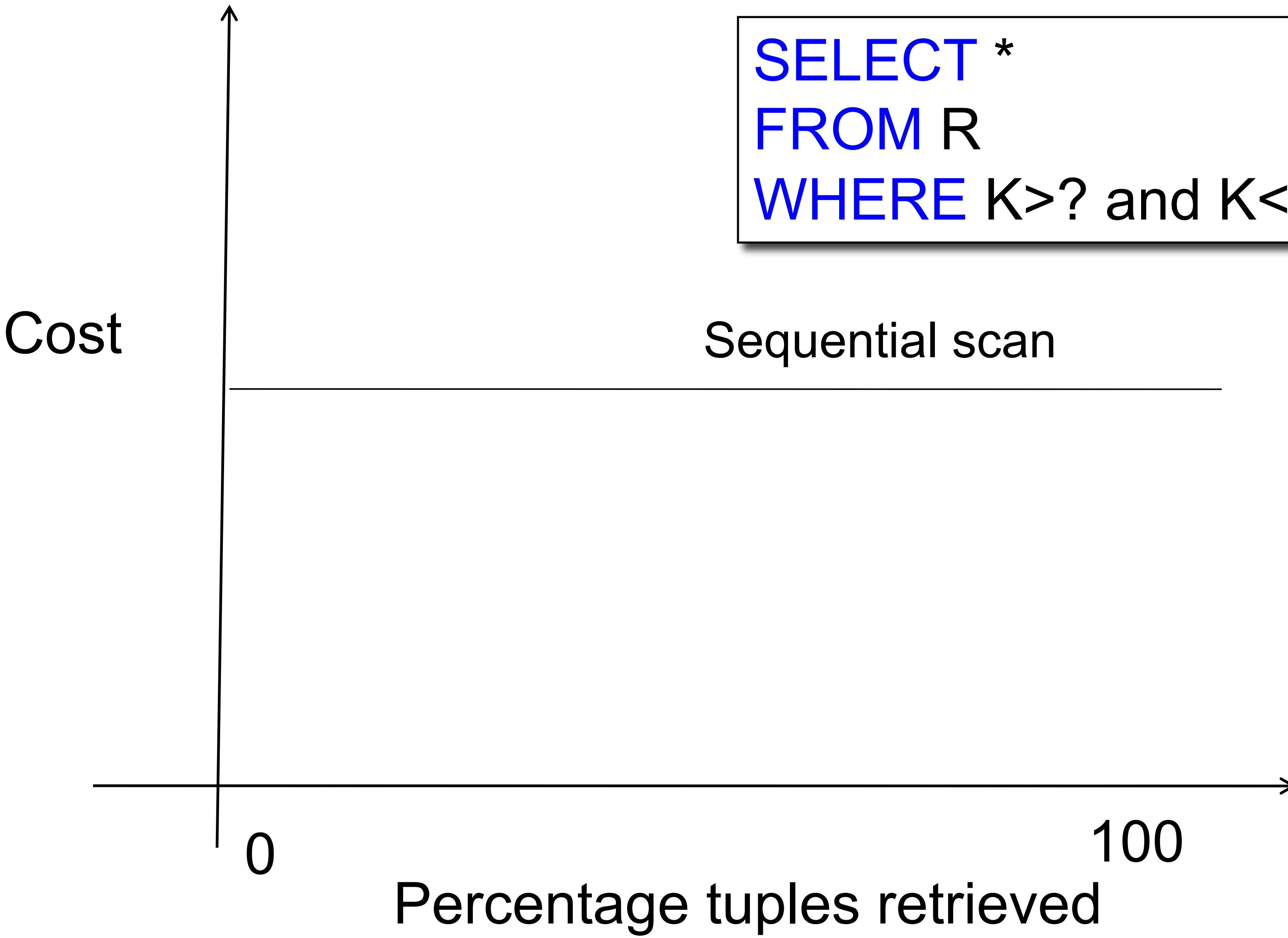
Basic Index Selection Guidelines

- 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
- To Cluster or Not?
  - Range queries benefit mostly from clustering

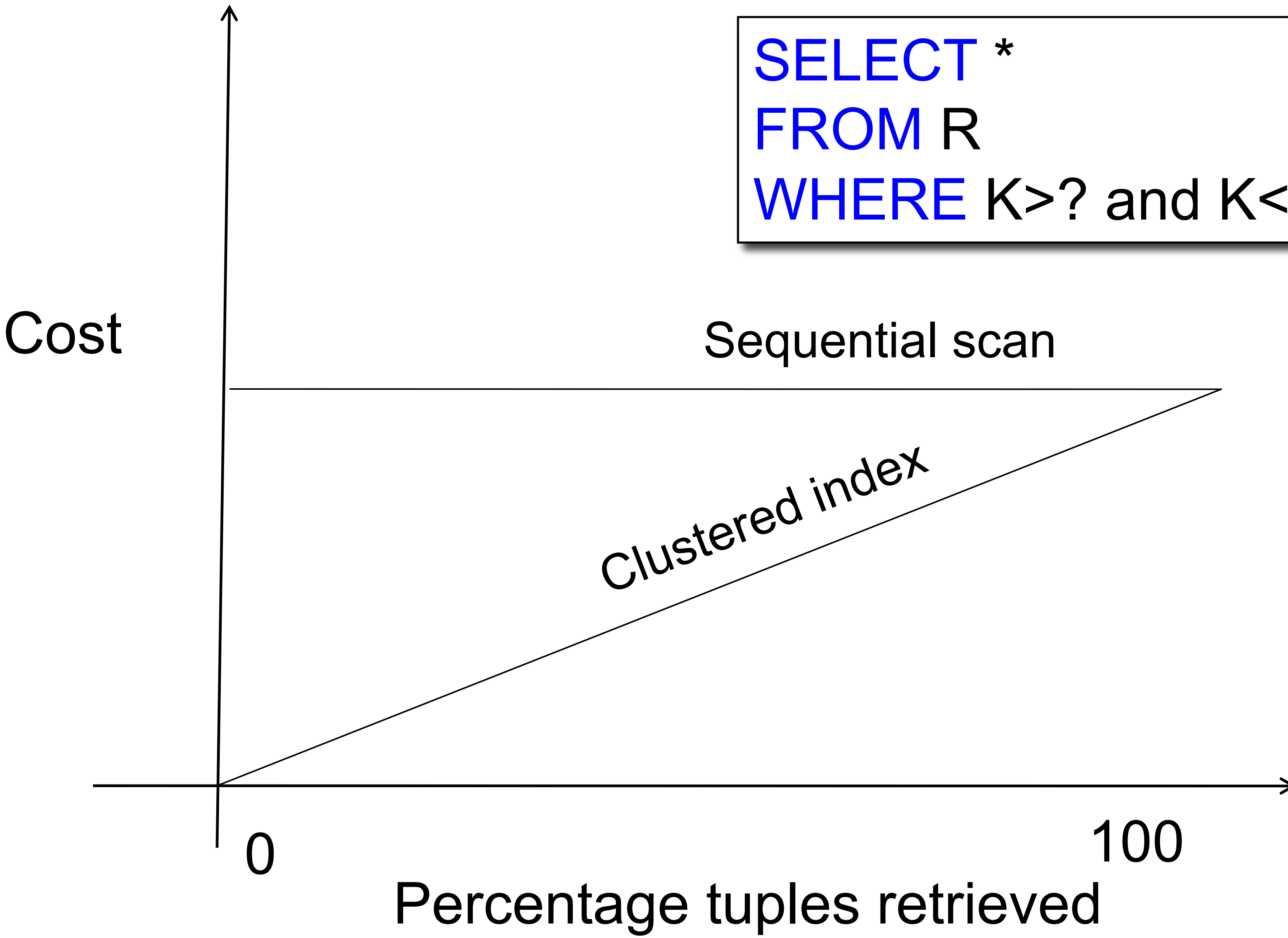
```
SELECT *
FROM R
WHERE K>? and K<?
```

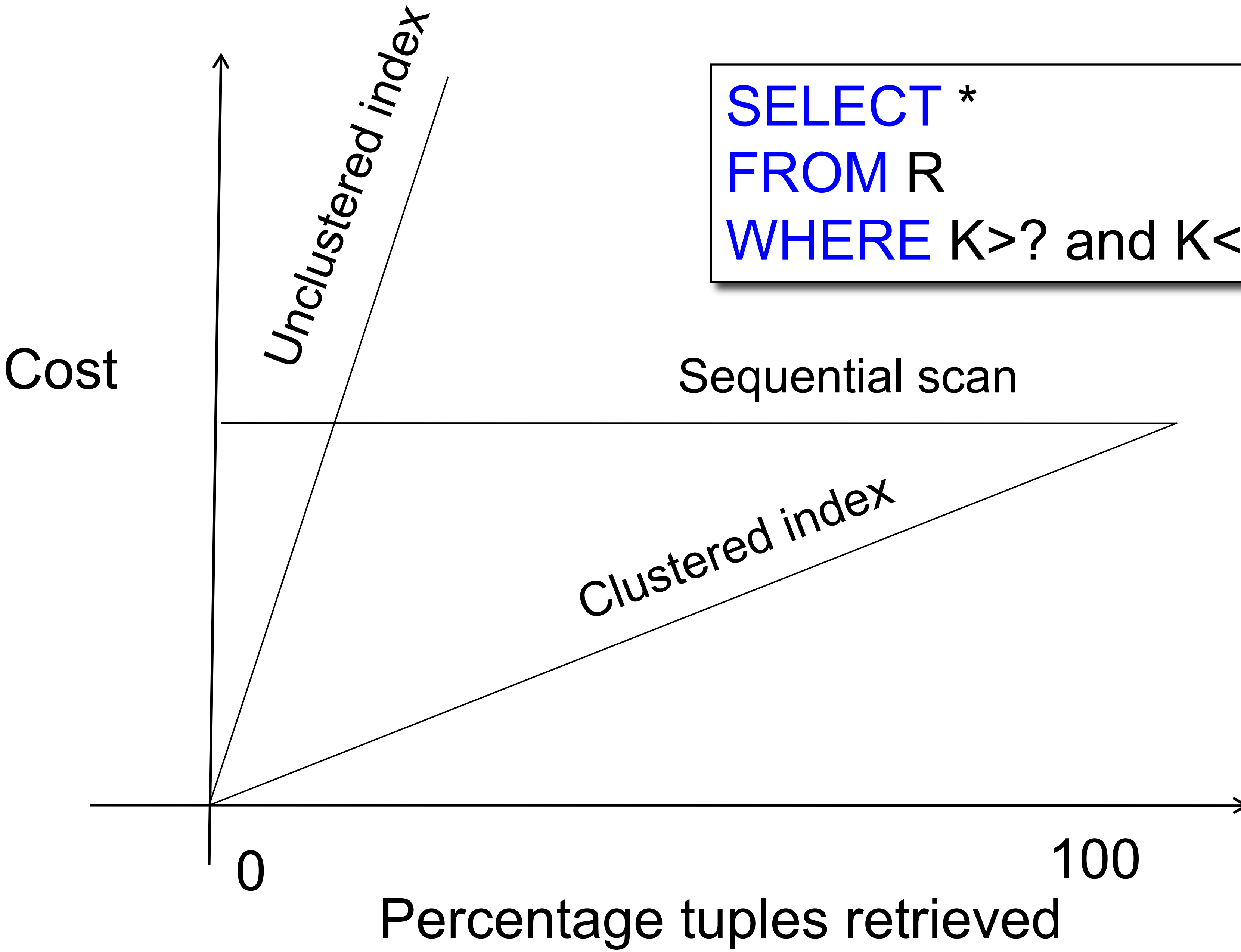


```sql
SELECT *
FROM R
WHERE K>? and K<?
```



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






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

Using Subqueries to Solve Problems: More Examples

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: Which employees have salaries greater than Richard's salary?

– What is Richard's salary?

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

```
SELECT names
FROM Employees
WHERE  Salary> (SELECT Salary
                FROM Employees
                WHERE  name='Richard')
```

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

- Which employee receives the highest salary ?

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

- Which employee receives the highest salary ?
- Let's find the other employees, i.e., those who do not receive the highest salary.

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

- Which employee receives the highest salary ?
- Let's find the other employees, i.e., those who do not receive the highest salary
- Let's find the maximum salary among those who do not receive the highest salary

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

```
SELECT * FROM Employee WHERE Salary
IN ( SELECT max(Salary) FROM Employee)
```

This will return the second record in our case

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

```
SELECT * FROM Employee WHERE Salary NOT
IN ( SELECT max(Salary) FROM Employee)
```

This will return all records except for the second one in our case

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

```
SELECT max(Salary) FROM Employee WHERE Salary NOT
IN ( SELECT max(Salary) FROM Employee)
```

This will return 130 in our case.

Example 0

Employees(id, name, salary)

id	name	salary
54	Richard	50
20	Roger	150
33	David	130
23	Nick	30

Main query: get the second highest salary?

Exercise: use a correlated subquery!

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 1

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 1

Find drinkers that frequent some bar that serves some beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 1

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

```
SELECT DISTINCT X.drinker
FROM Frequents X
WHERE EXISTS (SELECT bar
              FROM Serves Y, Likes Z
              WHERE X.bar=Y.bar
                    AND X.drinker=Z.drinker
                    AND Y.beer = Z.beer)
```

```
SELECT DISTINCT X.drinker
FROM Frequents X, Serves Y, Likes Z
WHERE X.bar = Y.bar
AND Y.beer = Z.beer
AND X.drinker = Z.drinke
```

Negation of Quantifiers

The statement: "It is not true that all **x** have the property P"

is equivalent to: "There is some **x** for which ~P is true".

Example:

– Not all people are honest = Some people are not honest.

Negation of Quantifiers

The statement: "It is not true that there is some **x** with the property P"

is equivalent to: "No **x** has the property P" or "All **x** have the property ~P."

Example:

- This bar does not serve some beer that I like = This bar only severs beer that I don’t like.
- I only frequent bars that serve some beer I like= I do not frequent bars that only serves beer I don’t like

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 2

Find drinkers that frequent some bar that serves only beers they don't like

Likes

drinker	beer
Roger	Bud
David	Michelob
Nick	Bud Lite
Richard	Bud

Frequents

name	bar
Roger	Joe's
Roger	adam's
Nick	Sue's
Richard	Sue's

Serves

bar	beer
Joe's	Bud
adam's	Michelob
adam's	Bud Lite
Sue's	Bud

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 2

Find drinkers that frequent some bar that serves only beers they don't like

Likes

drinker	beer
Roger	Bud
David	Michelob
Nick	Bud Lite
Richard	Bud

Frequents

name	bar
Roger	Joe's
Roger	adam's
Nick	Sue's
Richard	Sue's

Serves

bar	beer
Joe's	Bud
adam's	Michelob
adam's	Bud Lite
Sue's	Bud

Find drinkers that frequent some bar that DO NOT serves some beers they like

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 2

Find drinkers that frequent some bar that DO NOT serves some beers they like

```
SELECT DISTINCT X.drinker
FROM Frequents X
WHERE NOT EXISTS (SELECT bar
                  FROM Serves Y, Likes Z
                  WHERE X.bar=Y.bar
                     AND X.drinker=Z.drinker
                     AND Y.beer = Z.beer)
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 3

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

Likes

drinker	beer
Roger	Bud
David	Michelob
Nick	Bud Lite
Richard	Bud

Frequents

name	bar
Roger	Joe's
Roger	adam's
Nick	Sue's
Richard	Sue's
David	Sue's

Serves

bar	beer
Joe's	Bud
adam's	Michelob
adam's	Bud Lite
Sue's	Bud
Sue's	Michelob

Find drinkers that DO NOT *frequent <u>some</u> bar that <u>only</u> serves beers they <u>don't like</u>*!