CSE 344

SECTION 4 – RELATIONAL ALGEBRA

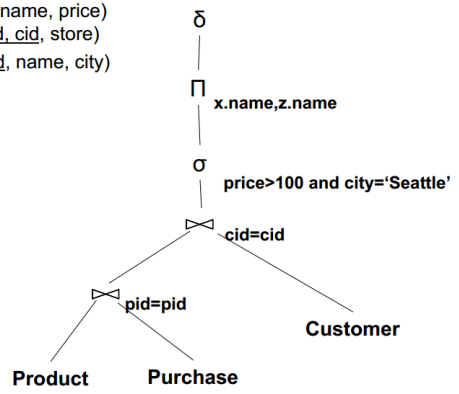
Why RA?

- Formalism for describing queries
- Basis of relational databases
- Will make you a SQL wizard!

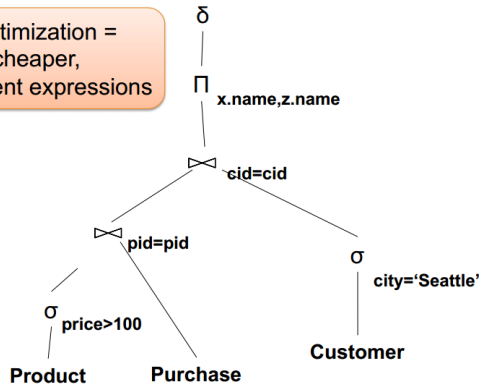
Notes on RA

- Multiple possible query plans

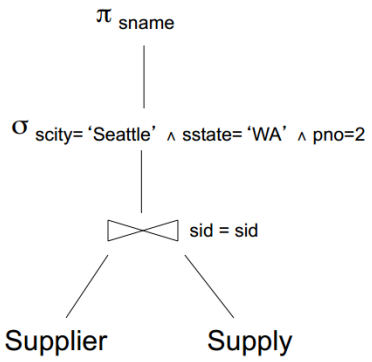
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

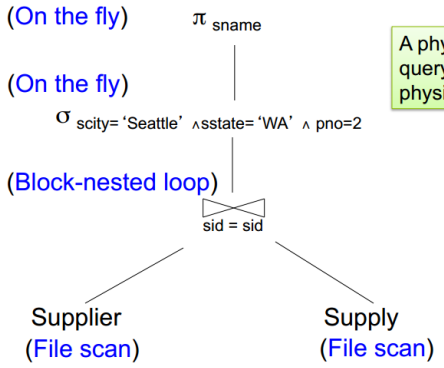


Query optimization = finding cheaper, equivalent expressions



- Logical vs. Physical query plans

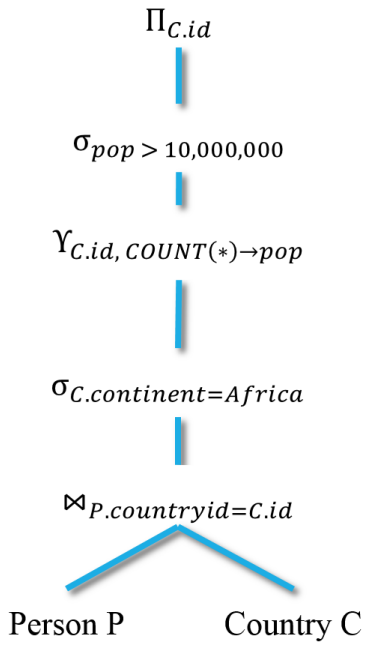




A physical query plan is a logical query plan annotated with physical implementation details

Example: RA-to-SQL

Person(id, name, countryid)
Country(id, name, continent)




```
SELECT C.id
FROM Person P, Country C
WHERE P.countryid = C.id
AND C.continent='Africa'
GROUP BY C.id
HAVING COUNT(*) > 10000000
```

Can we make a more efficient plan?

Equivalently in equation form: $\Pi_{C.id}(\sigma_{pop>10{,}000{,}00}(\Upsilon_{C.id,COUNT(*)\rightarrow pop}(\sigma_{C.continent=Africa}(Person\ P \bowtie_{P.countryid=C.id} Country\ C))))$

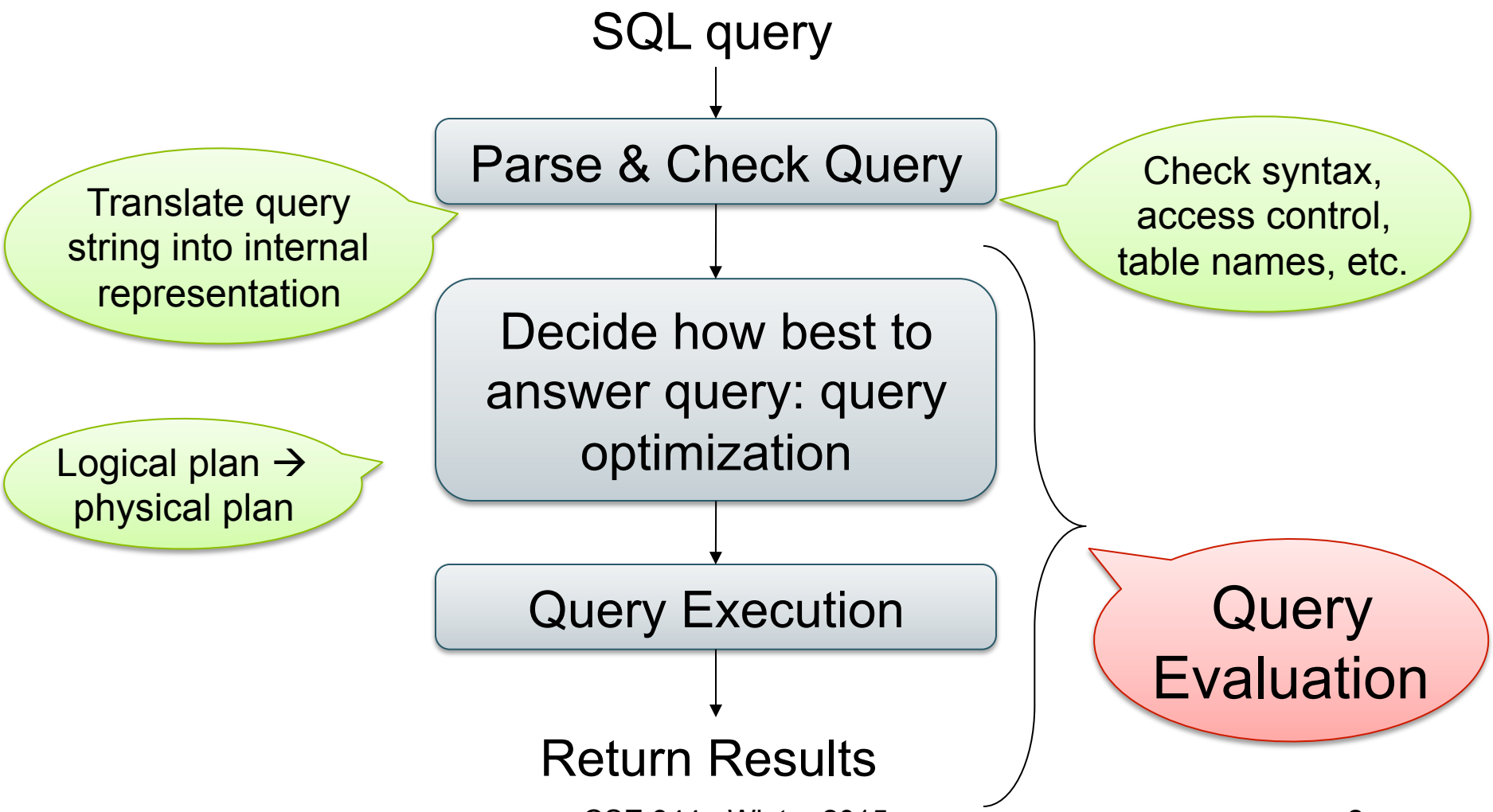
Demo in Azure!

RA Reference Sheet

Name	Symbol
Selection	σ
Projection	π
Join	$\bowtie$
Group By	γ
Set Difference	$-$
Duplicate Elimination	δ

From Logical Plans to Physical Plans

Query Evaluation Steps



Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Example

```
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'
```

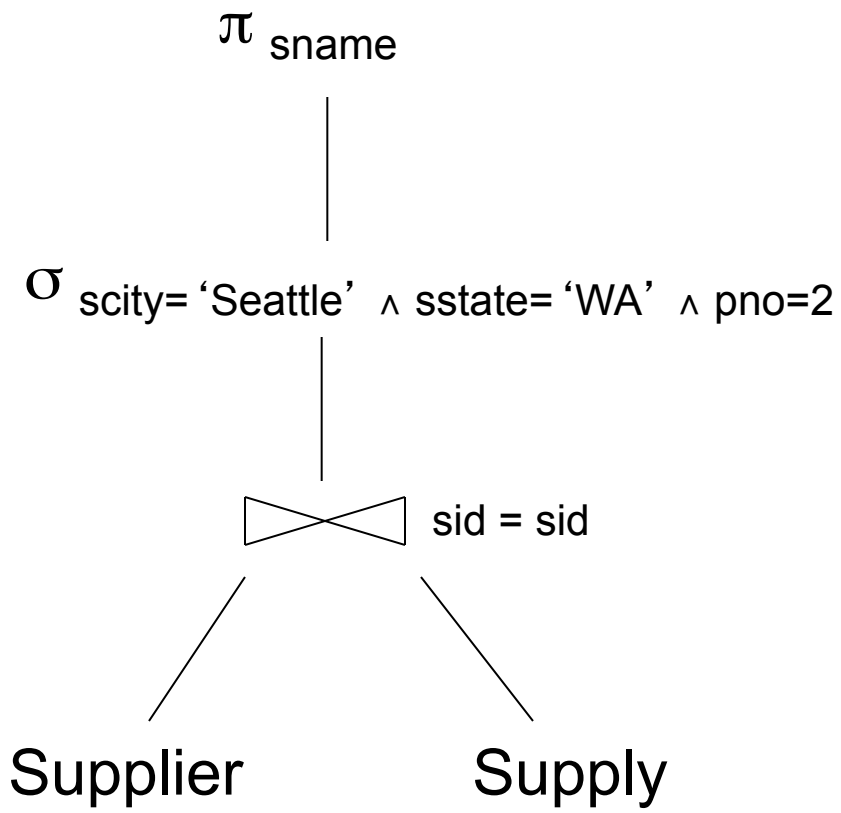
$$\pi_{\text{sname}}(\sigma_{\text{scity= 'Seattle'} \wedge \text{sstate= 'WA'} \wedge \text{pno=2}}(\text{Supplier} \bowtie_{\text{sid = sid}} \text{Supply}))$$

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'
```

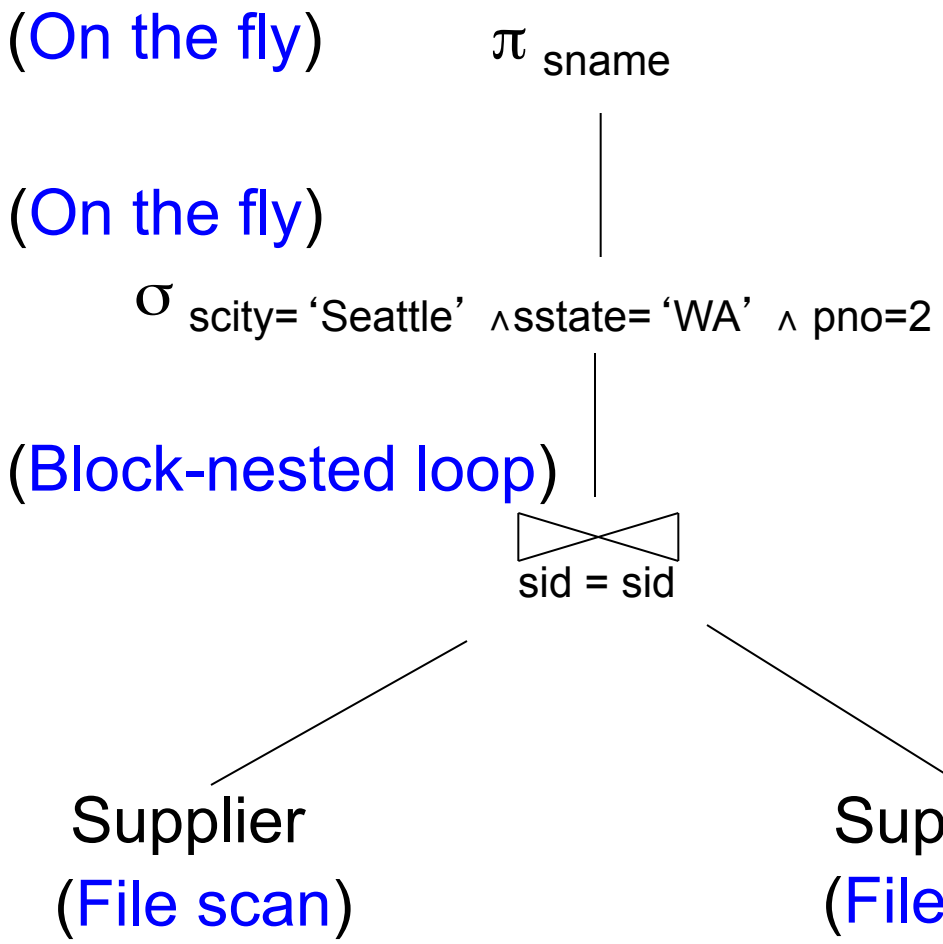


Relational algebra expression is also called the "logical query plan"

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, 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'
```

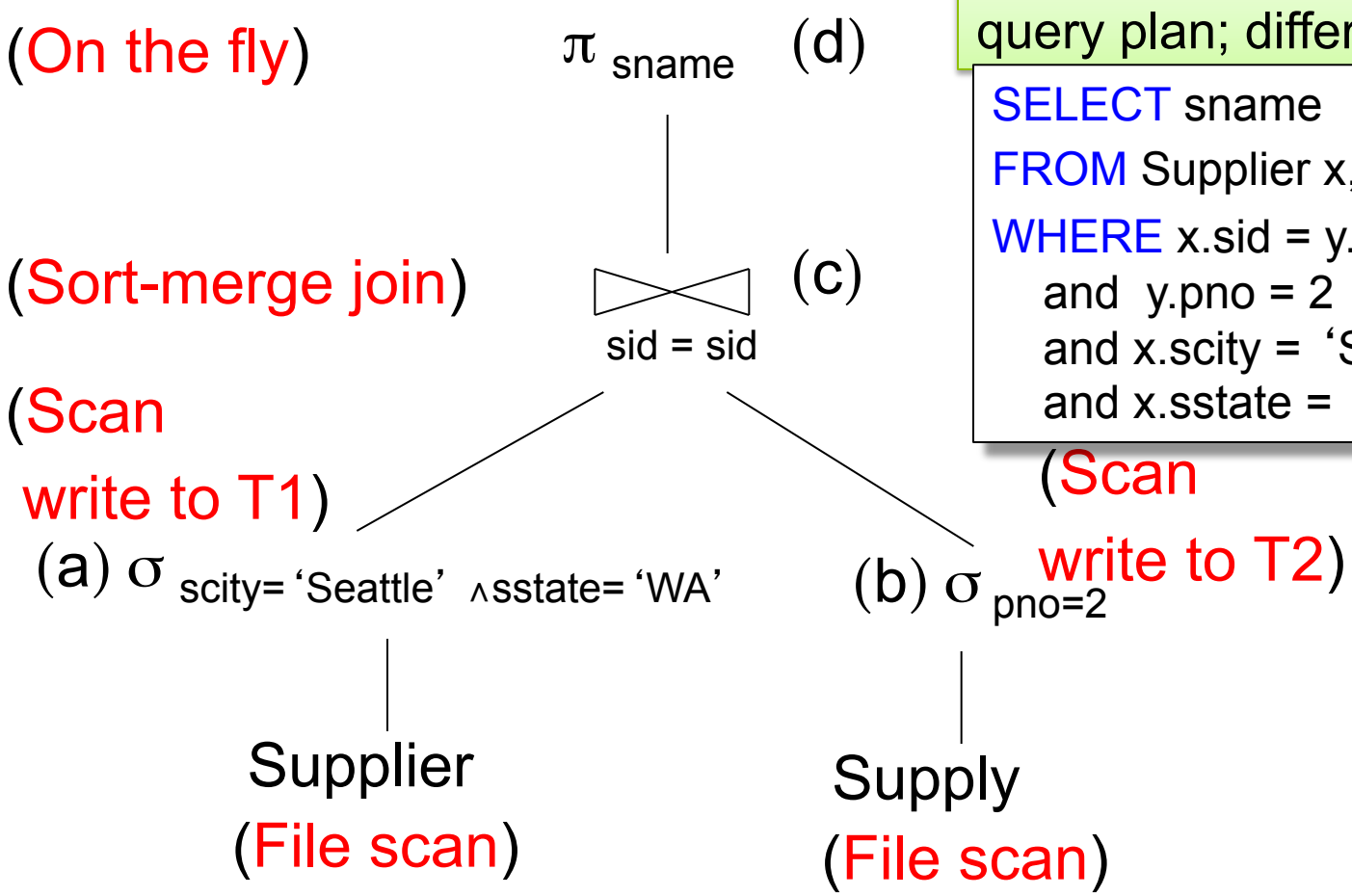
Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 2

Different but equivalent logical query plan; different physical plan

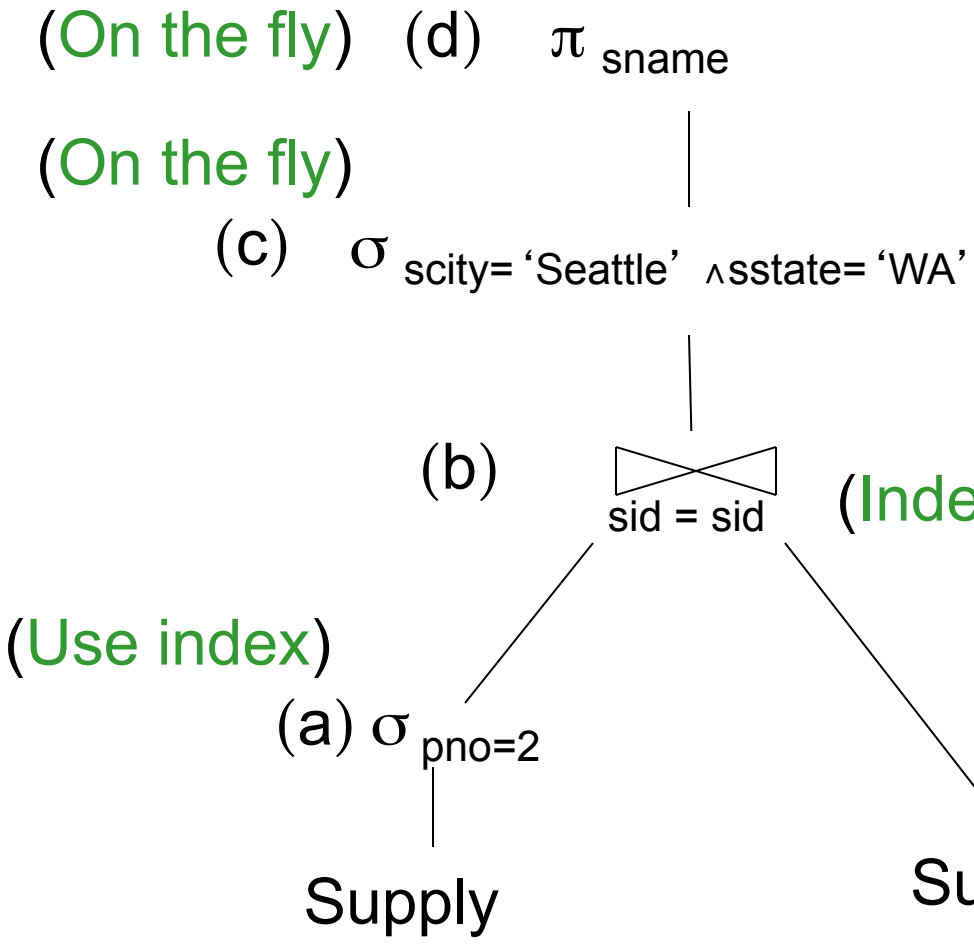
```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```



Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 3



Another logical plan that produces the same result and is implemented with a 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'
```

Physical Data Independence

- Means that applications are insulated from changes in physical storage details
  - E.g., can add/remove indexes without changing apps
  - Can do other physical tunings for performance

- SQL and relational algebra facilitate physical data independence because both languages are "set-at-a-time": Relations as input and output

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

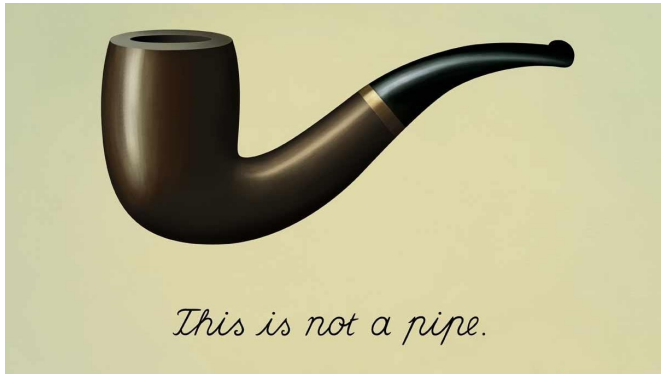
- An **additional** file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - The key = an attribute value (e.g., student ID or name)
  - The value = a pointer to the record
- Could have many indexes for one table

Key = means here search key

This  Is Not A Key

Different keys:

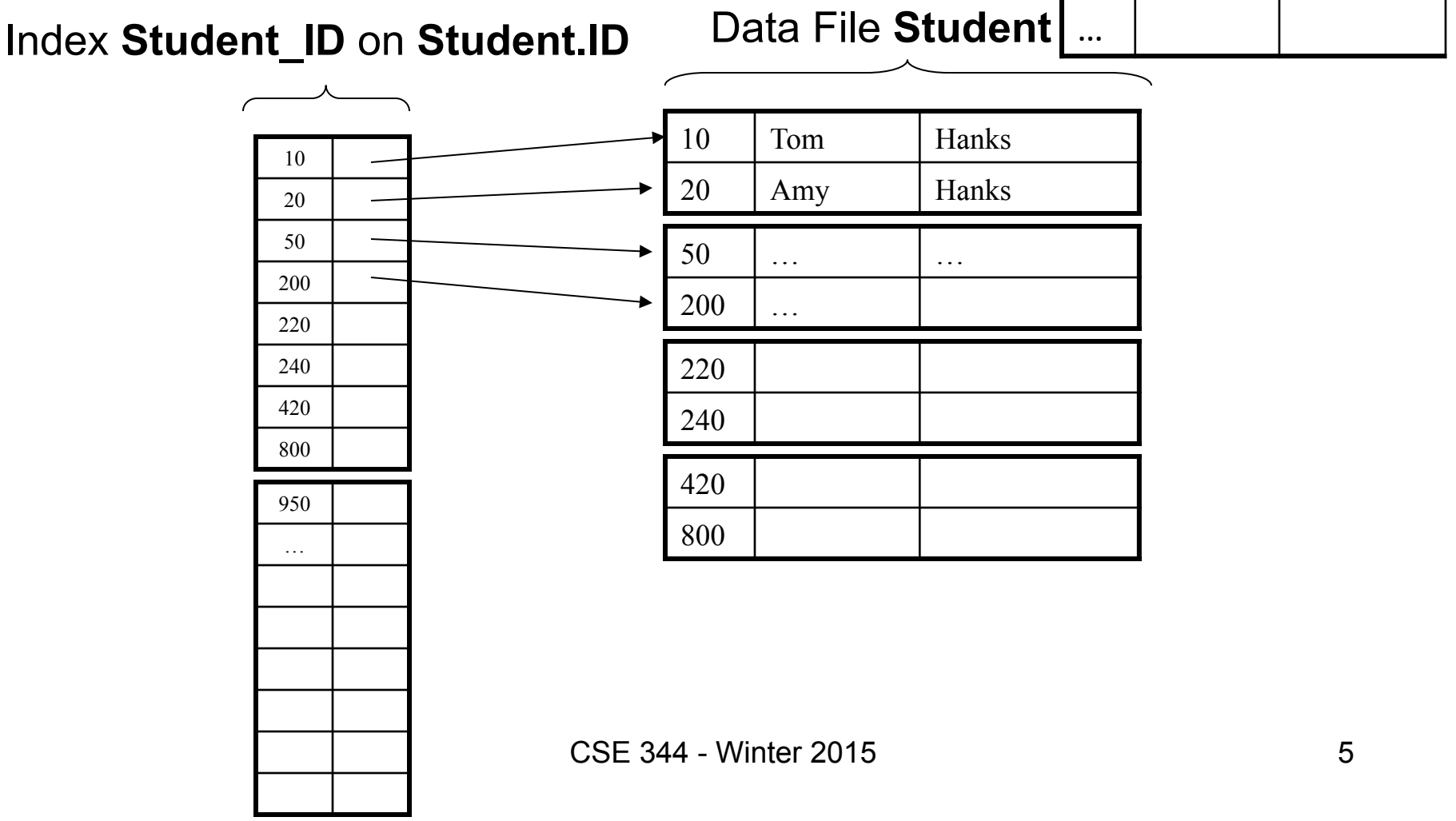
- Primary key – uniquely identifies a tuple
- Key of the sequential file – how the datafile is sorted, if at all
- Index key – how the index is organized



Example 1: Index on ID

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		



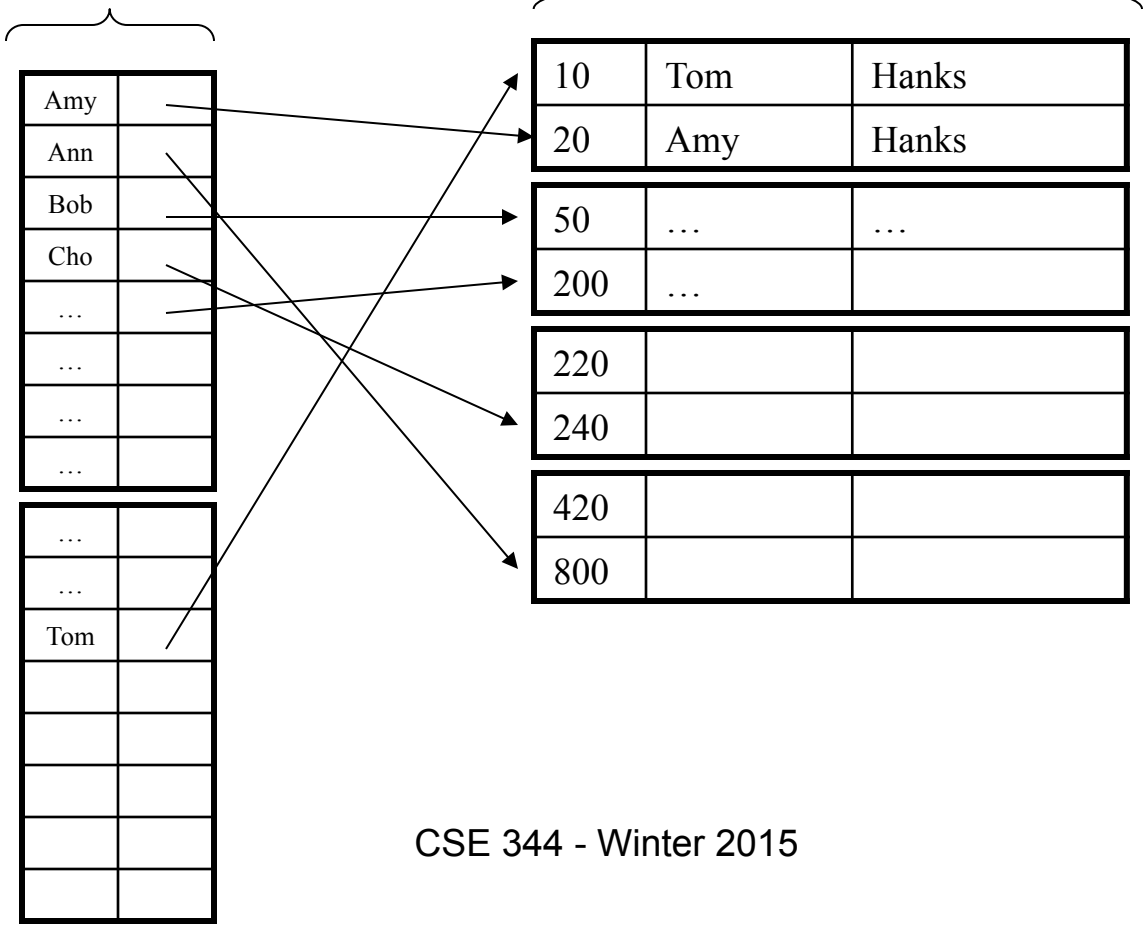
Example 2: Index on fName

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

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

Data File **Student**

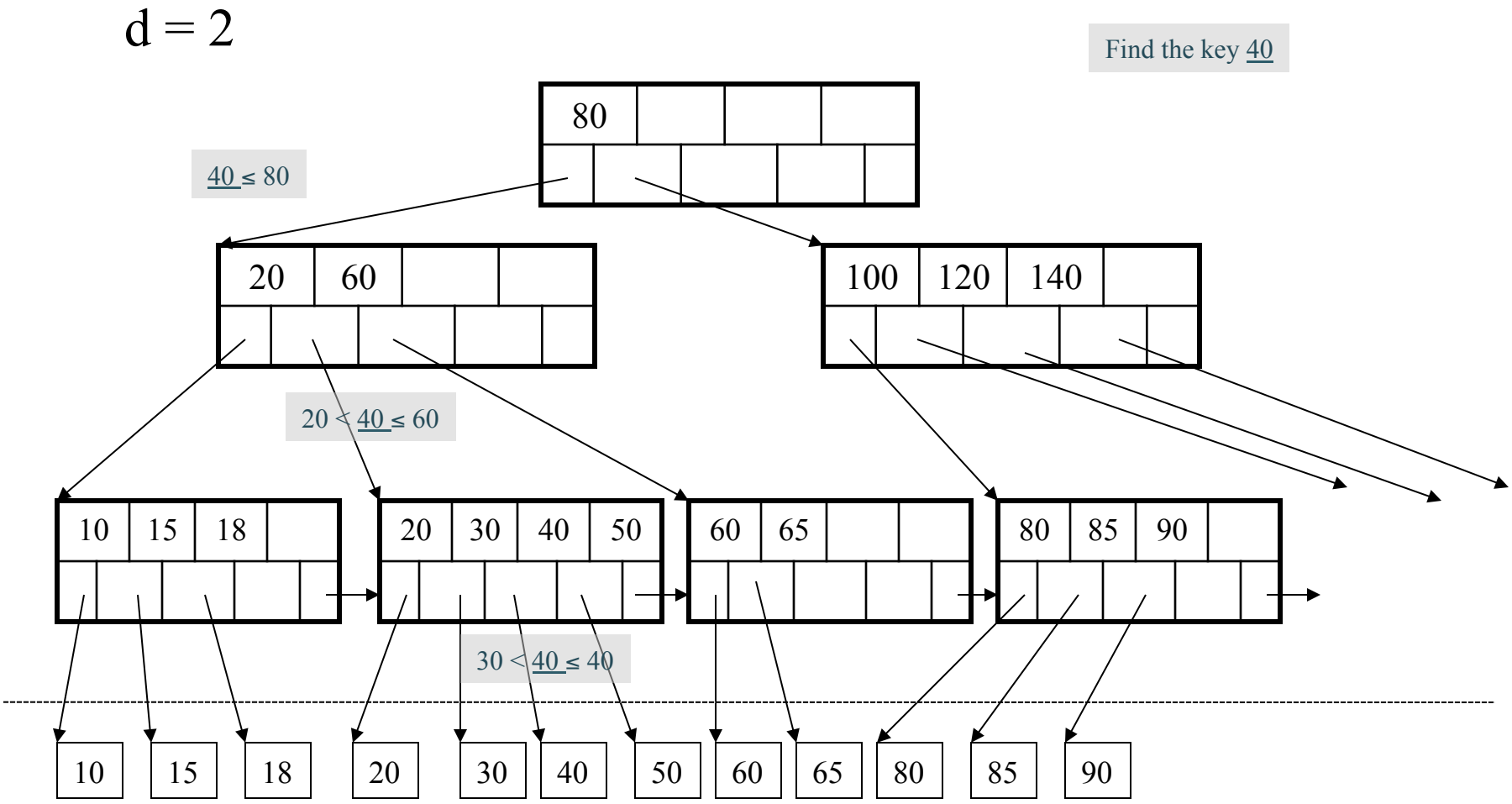


Index Organization

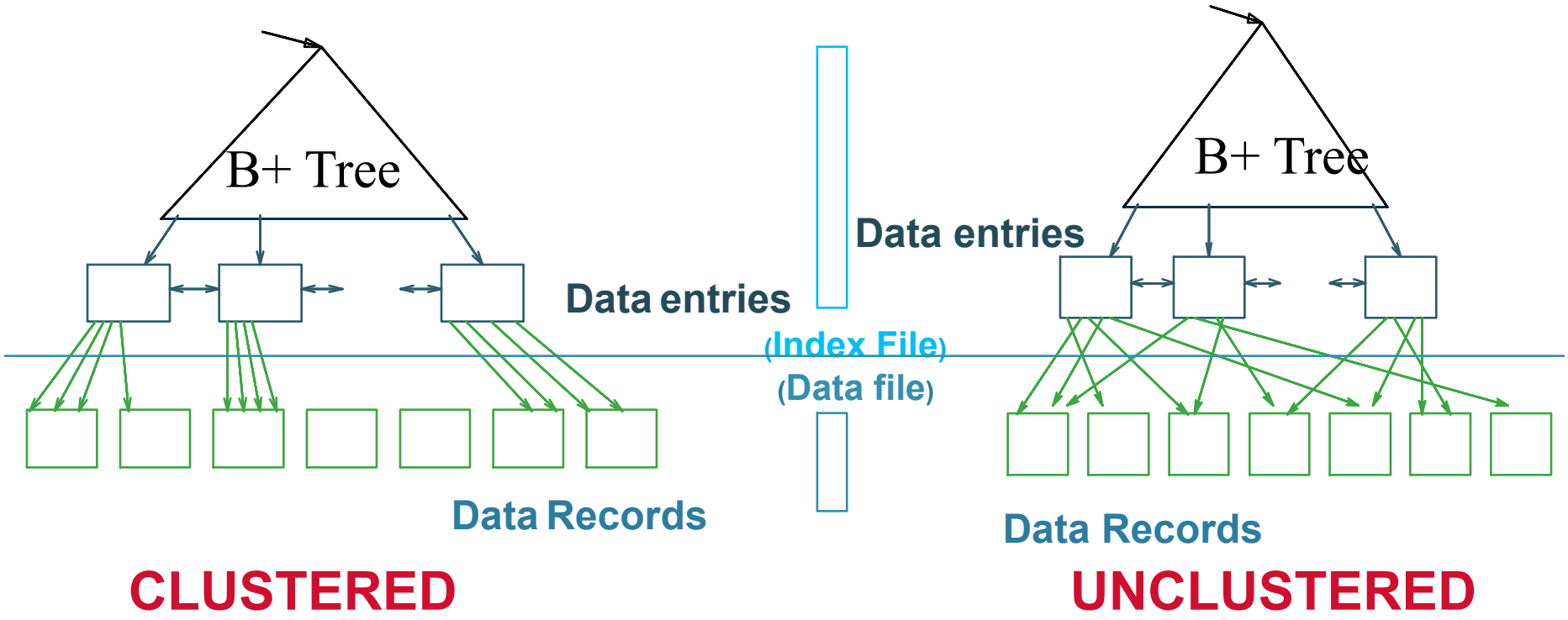
Several index organizations:

- 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+ Tree Index by Example



Clustered vs Unclustered



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

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)
```

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

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)?

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

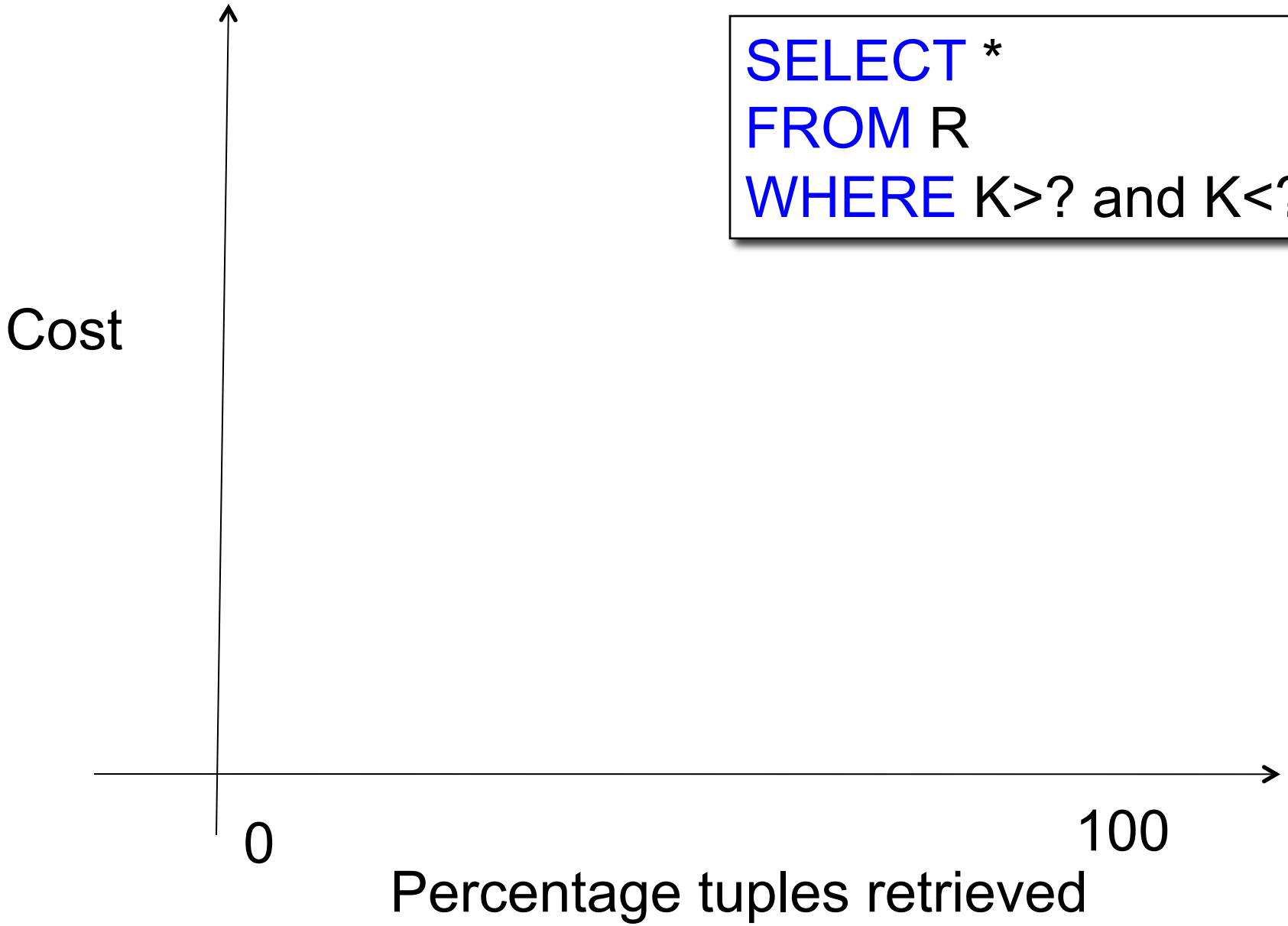
Consider creating a multi-attribute key on K1, K2, … if

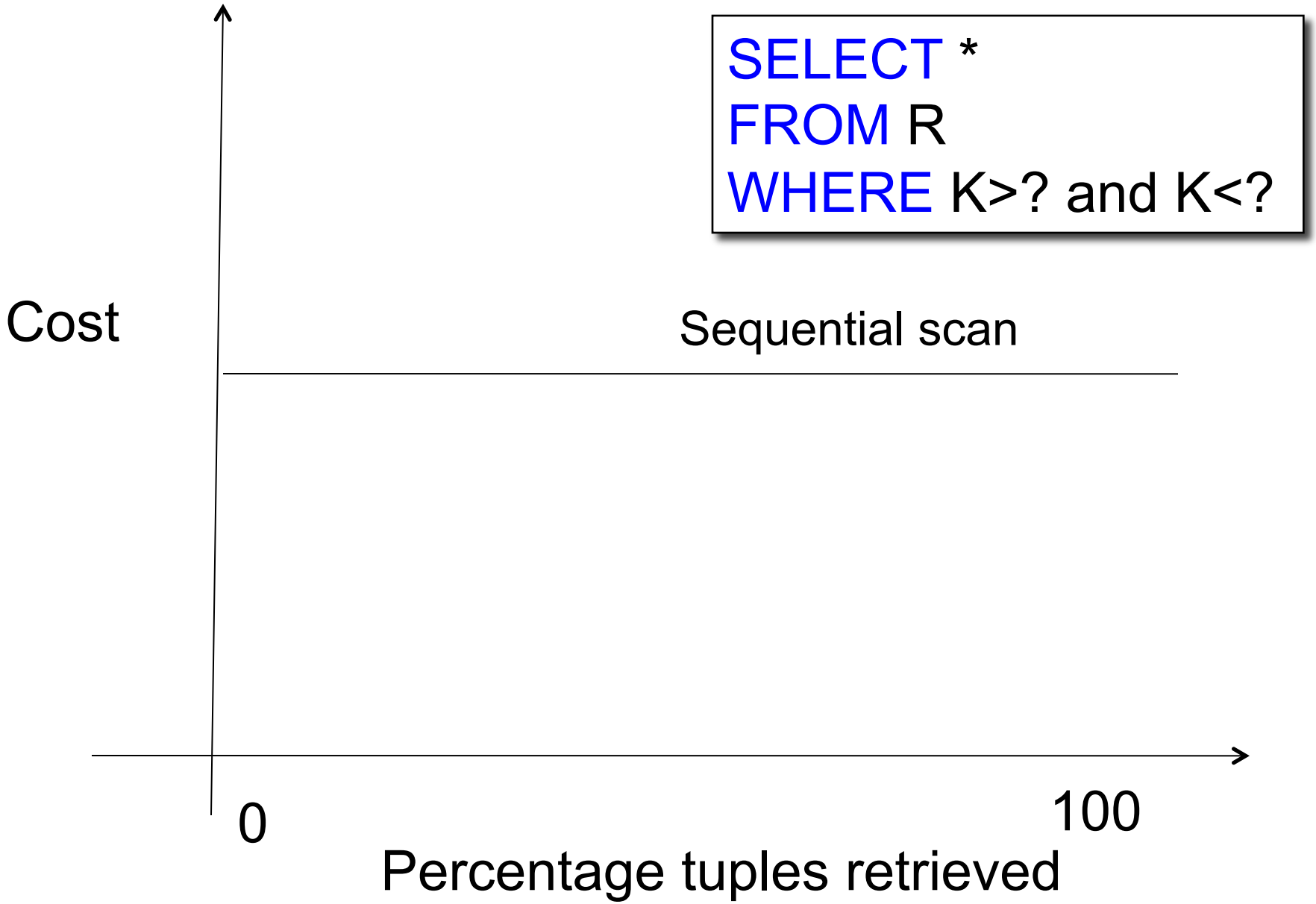
- WHERE clause has matches on K1, K2, …
  - But also consider separate indexes
- SELECT clause contains only K1, K2, ..
  - A *covering index* is one that can be used exclusively to answer a query, e.g. index R(K1,K2) covers the query:

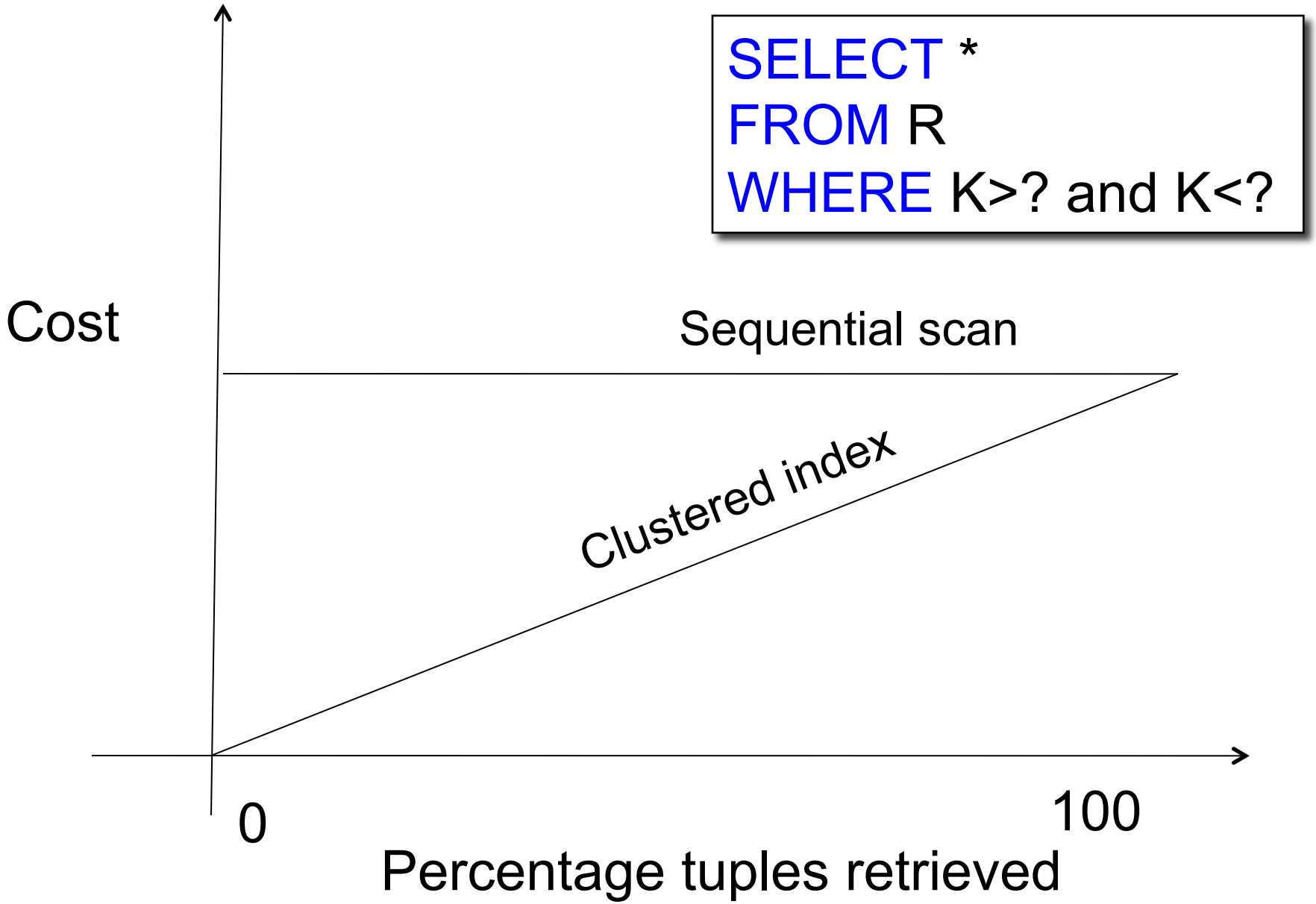
```
SELECT K2 FROM R WHERE K1=55
```

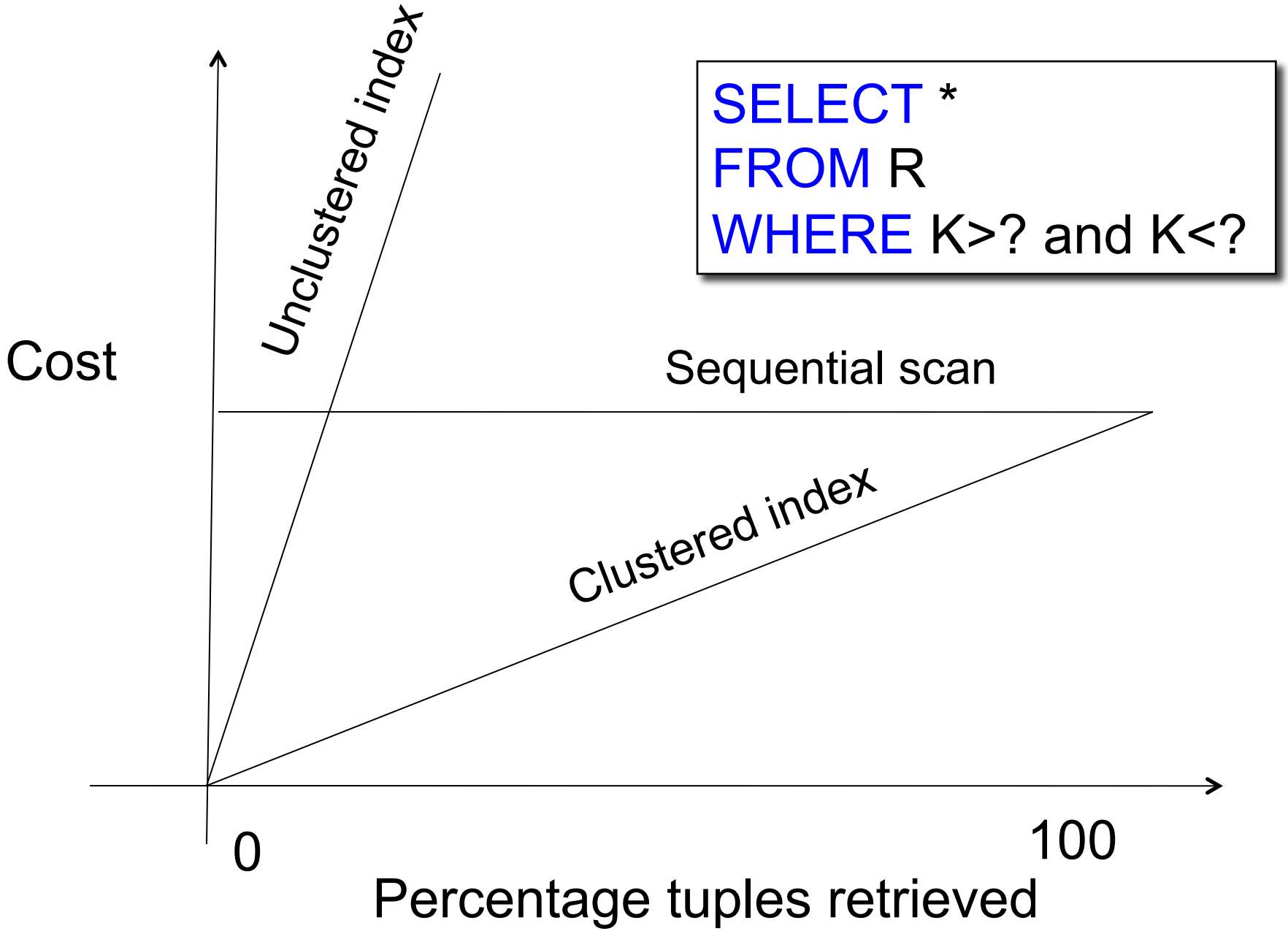
To Cluster or Not

- 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<?
Cost
0
100
Percentage tuples retrieved


SELECT *
FROM R
WHERE K>? and K<?
Cost
Sequential scan
0
100
Percentage tuples retrieved


SELECT *
FROM R
WHERE K>? and K<?
Cost
Sequential scan
Clustered index
0
100
Percentage tuples retrieved


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