CSE 344

FEBRUARY 16TH – DISK I/O AND ESTIMATION

ADMINISTRIVIA

- **HW6/OQ5 out after class**
  - HW6 Due Wednesday, Feb 28th
  - OQ5 Due Friday, Feb 23rd
- **Office hours for exam regrades**
  - Additional HW5 OH on Wednesday

HW6 AWS

- **Making account**
  - Use accurate information (matching academic records)
  - Create full account – not ‘starter code’
  - Be sure to terminate services when done

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

KEYS IN INDEXING

Different keys:

Primary key – uniquely identifies a tuple

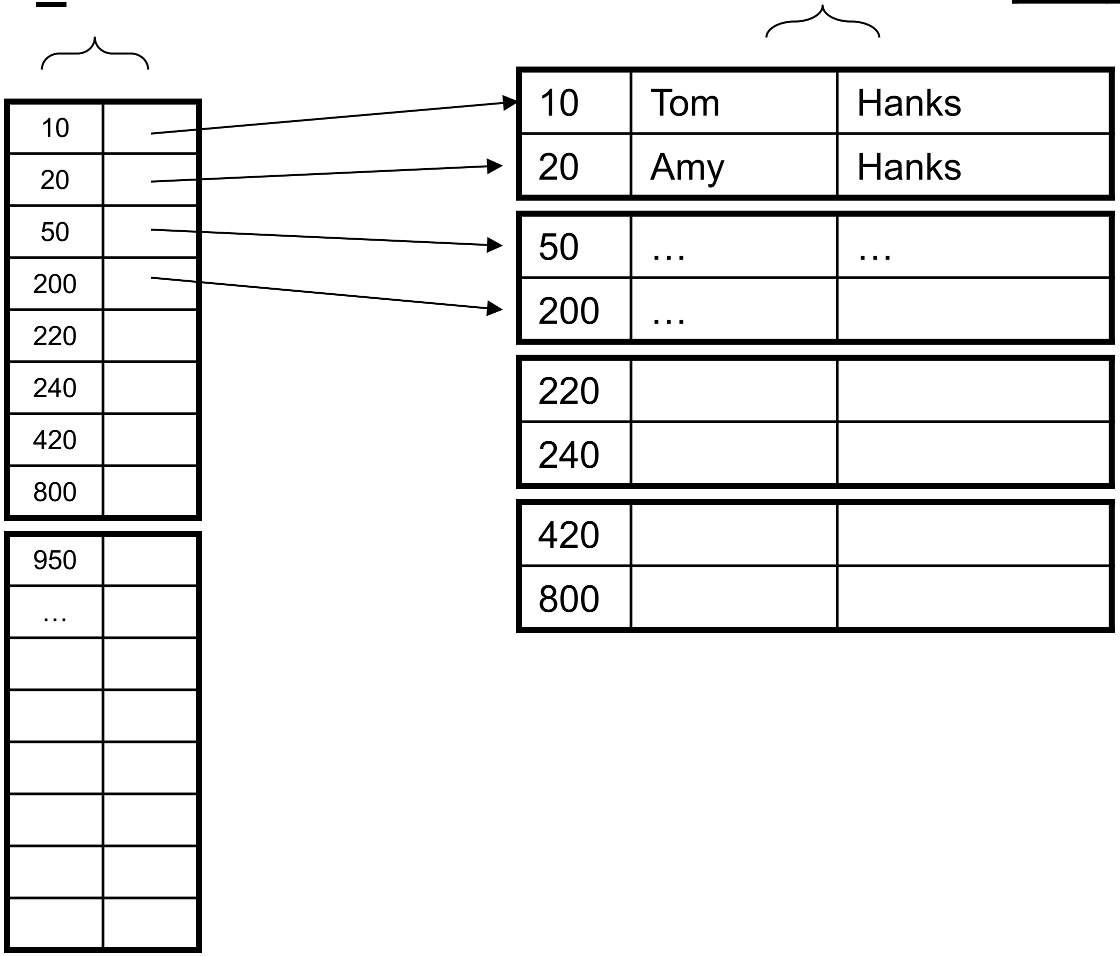
Key of the sequential file – how the data file is sorted, if at all

Index key – how the index is organized

EXAMPLE 1: INDEX ON ID

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		



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

10	
20	
50	
200	
220	
240	
420	
800	

950	
…	

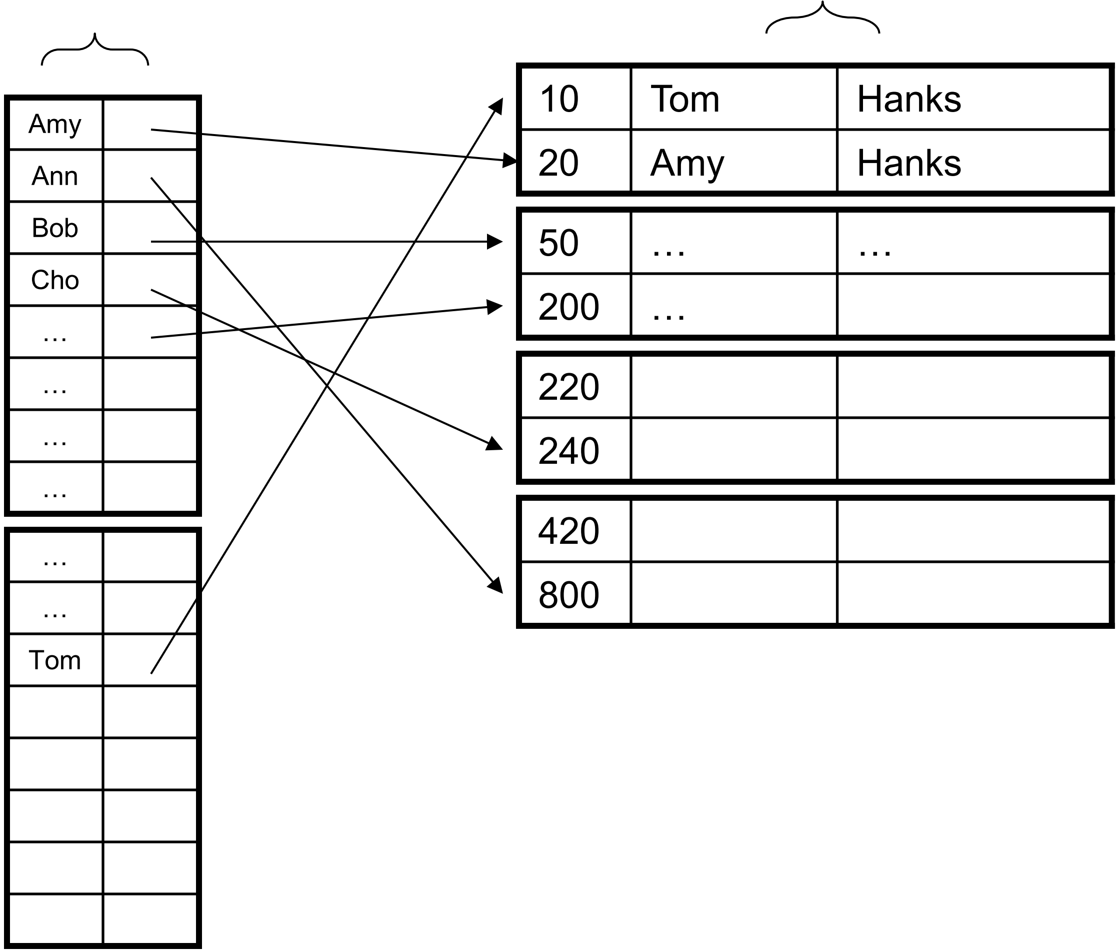
Data File **Student**

10	Tom	Hanks
20	Amy	Hanks
50	…	…
200	…	
220		
240		
420		
800		

EXAMPLE 2: INDEX ON FNAME

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		



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

Amy	
Ann	
Bob	
Cho	
…	
…	
…	
…	

…	
…	
Tom	

Data File **Student**

10	Tom	Hanks
20	Amy	Hanks

50	…	…
200	…	

220		
240		

420		
800		

INDEX ORGANIZATION

We need a way to represent indexes after loading into memory so that they can be used

Several ways to do this:

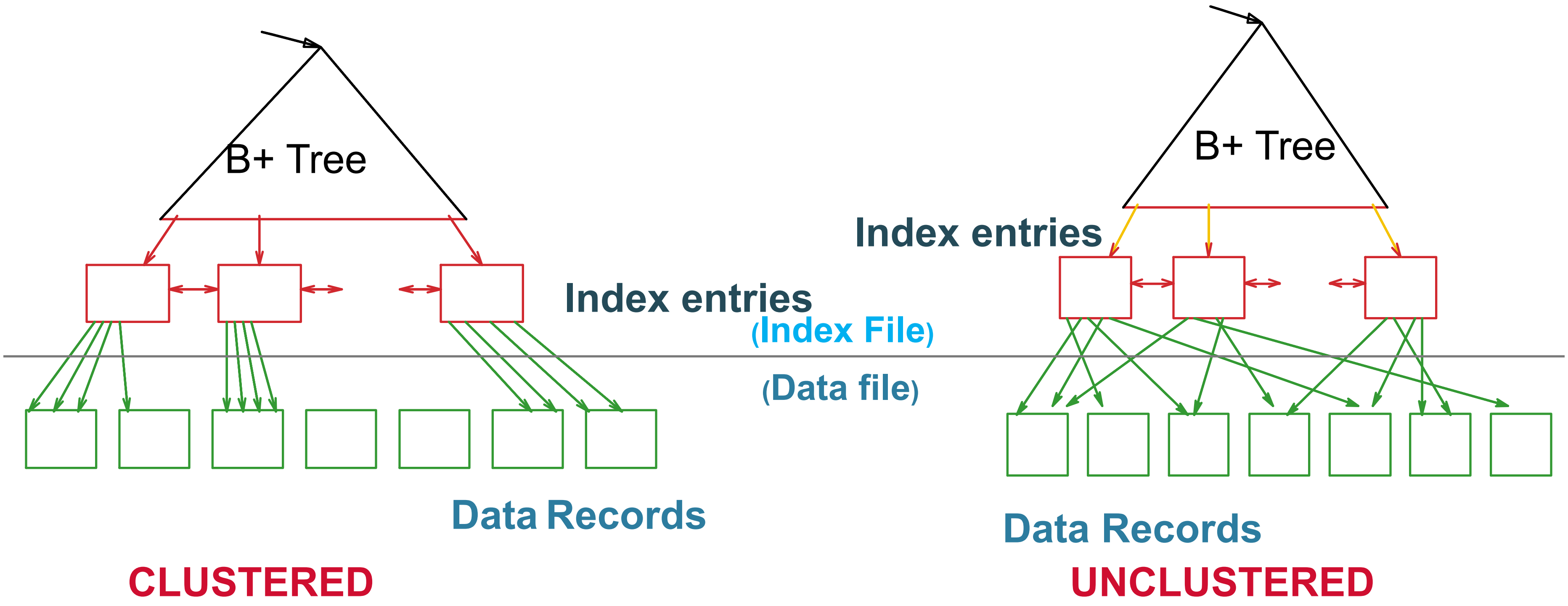
Hash table

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

CLUSTERED VS UNCLUSTERED



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

Disks are mechanical devices!

- Technology from the 60s; density much higher now

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)

Solid state (SSD): $$$ expensive; put indexes, other “hot” data there, still too expensive for everything

SUMMARY SO FAR

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 reads are efficint
- Random access less efficient
- Random read 1-2% of data worse than sequential

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)
```

```
CREATE  INDEX V3 ON V(M, N)
```

```
CREATE UNIQUE INDEX V4 ON V(N)
```

```
CREATE CLUSTERED INDEX V5 ON V(N)
```

```
select *
from V
where P=55 and M=77
```

```
select *
from V
where P=55
```

```
select *
from V
where M=77
```

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)
```

yes

```
select *
from V
where P=55 and M=77
```

```
select *
from V
where P=55
```

yes

```
select *
from V
where M=77
```

no

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

```
select *
from V
where P=55 and M=77
```

yes

```
select *
from V
where P=55
```

yes

```
select *
from V
where M=77
```

no

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=?
```

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

TWO TYPICAL KINDS OF QUERIES

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

- Point queries
- What data structure should be used for index?

```
SELECT *
FROM Movie
WHERE year >= ? AND
          year <= ?
```

- Range queries
- What data structure should be used for 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

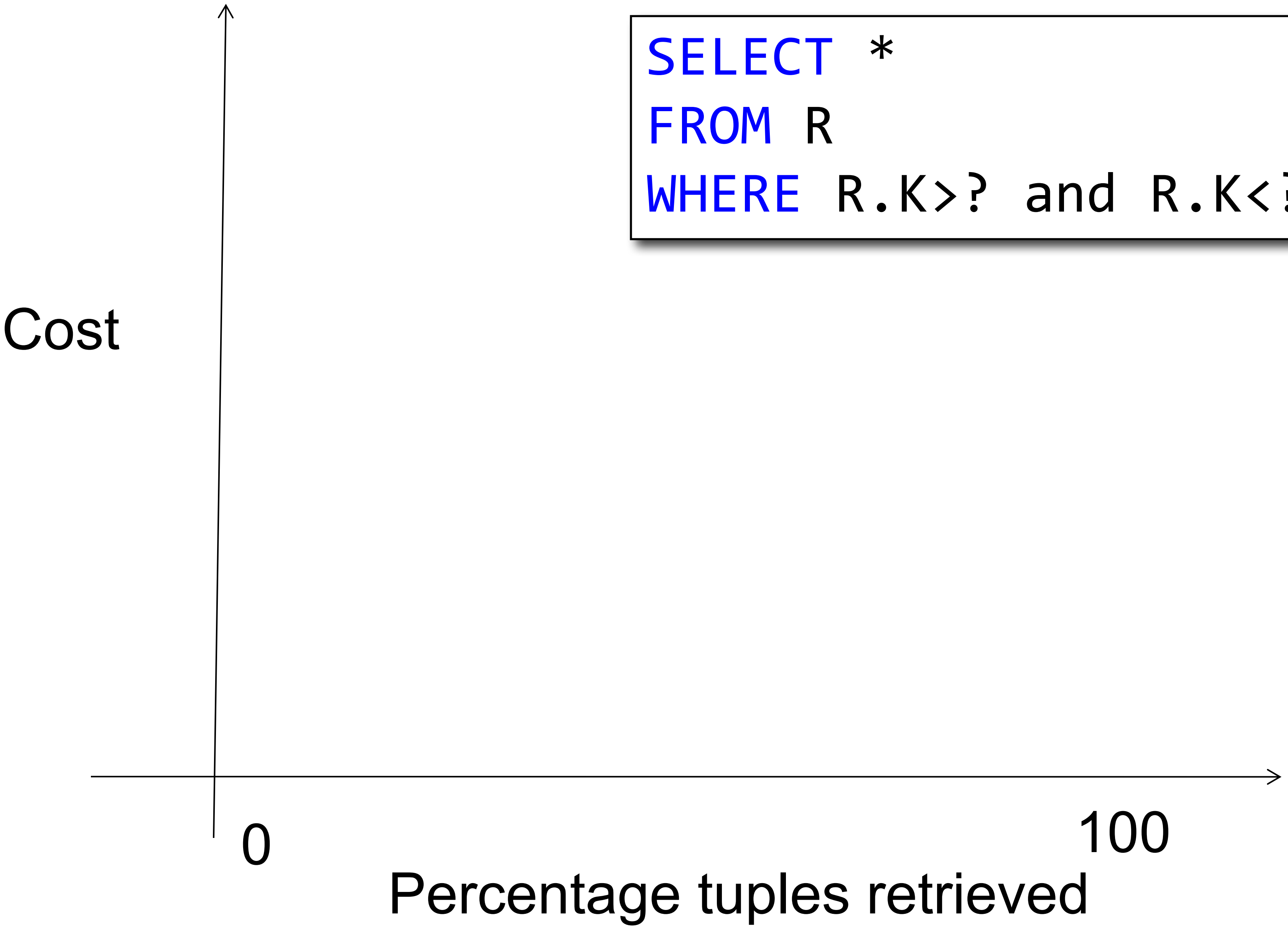
Try to choose indexes that speed-up multiple queries

TO CLUSTER OR NOT

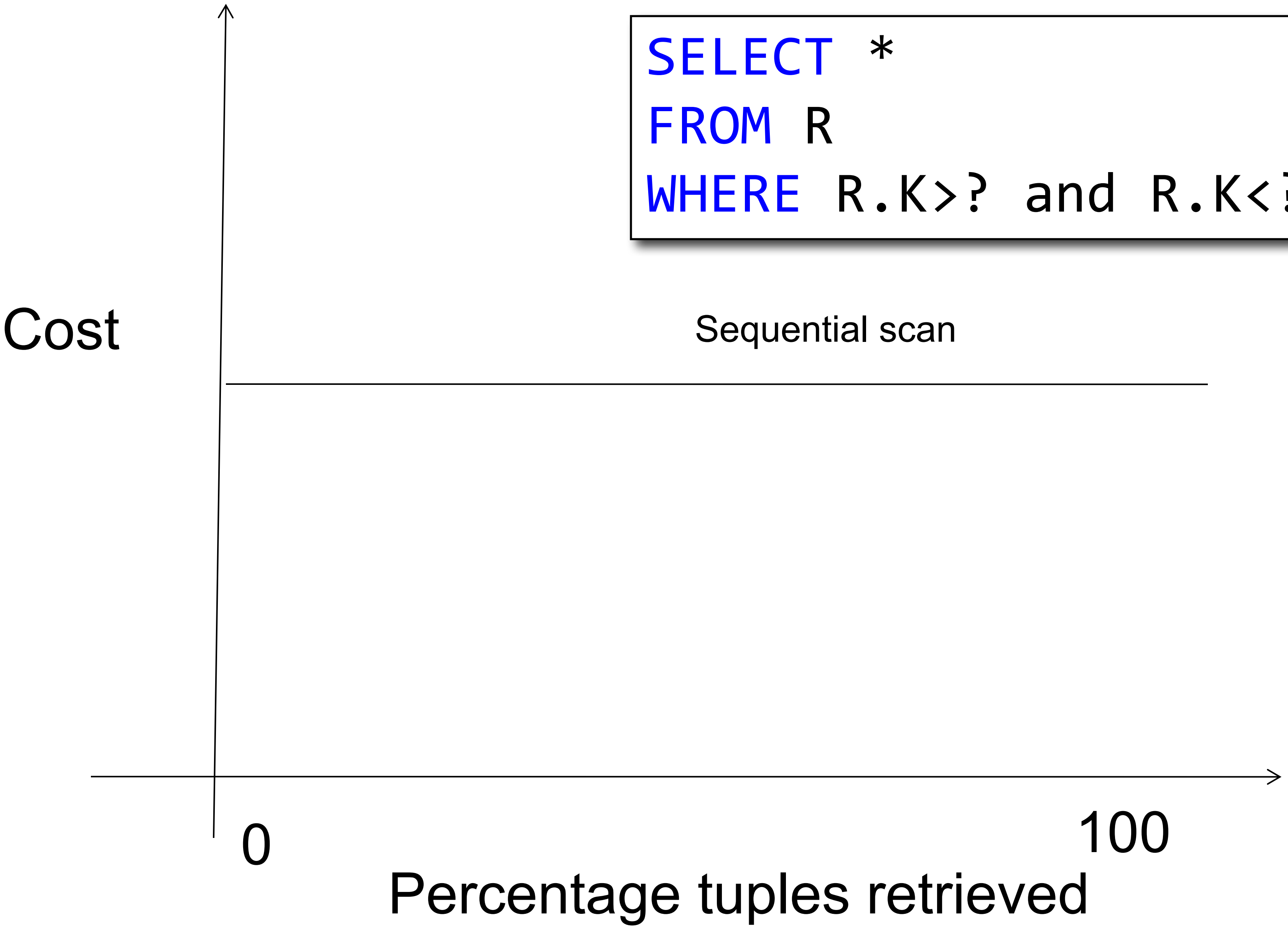
Range queries benefit mostly from clustering

Point indexes do *not* need to be clustered: they work equally well unclustered

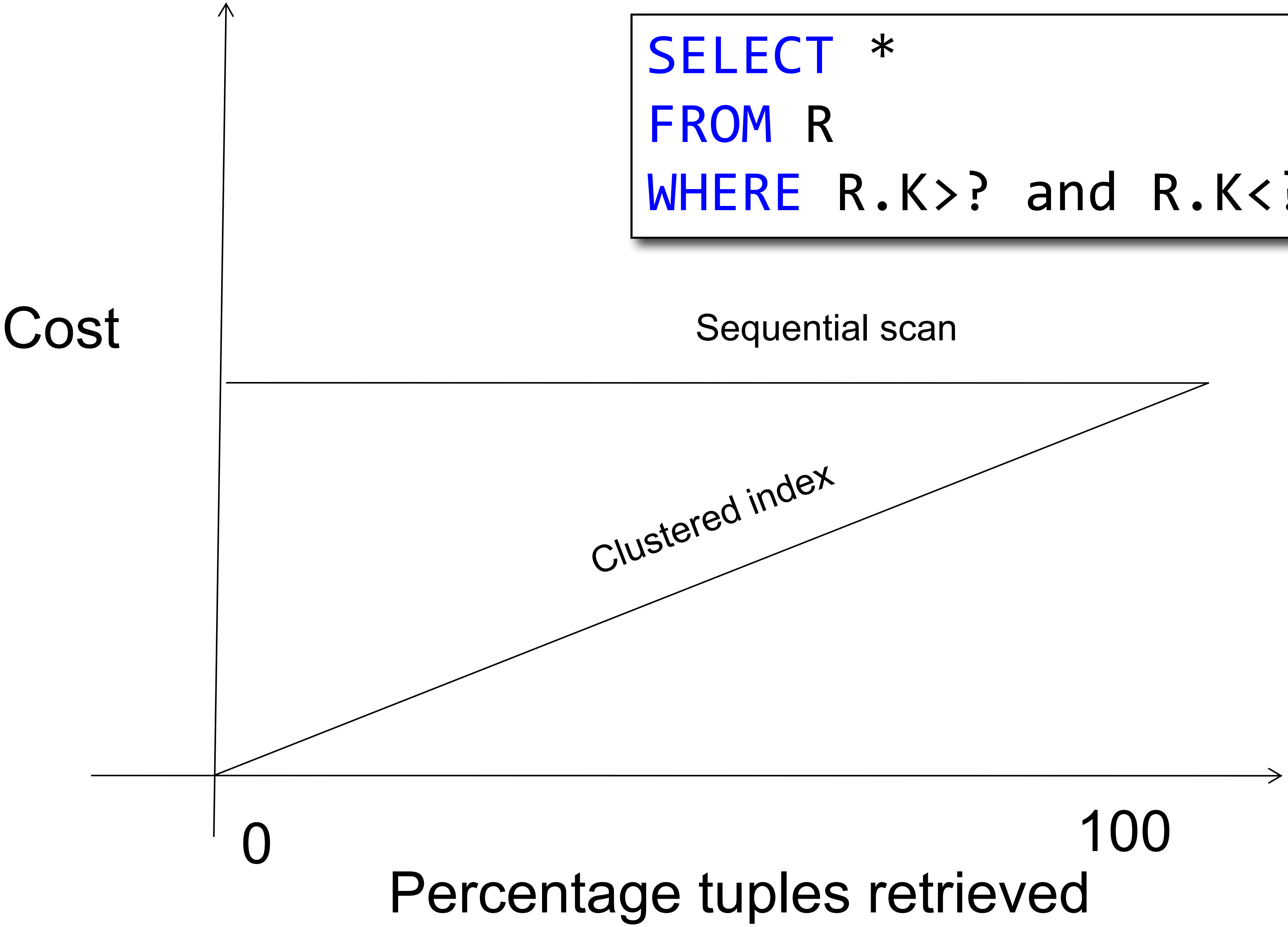
```
SELECT *
FROM R
WHERE R.K>? and R.K<?
```

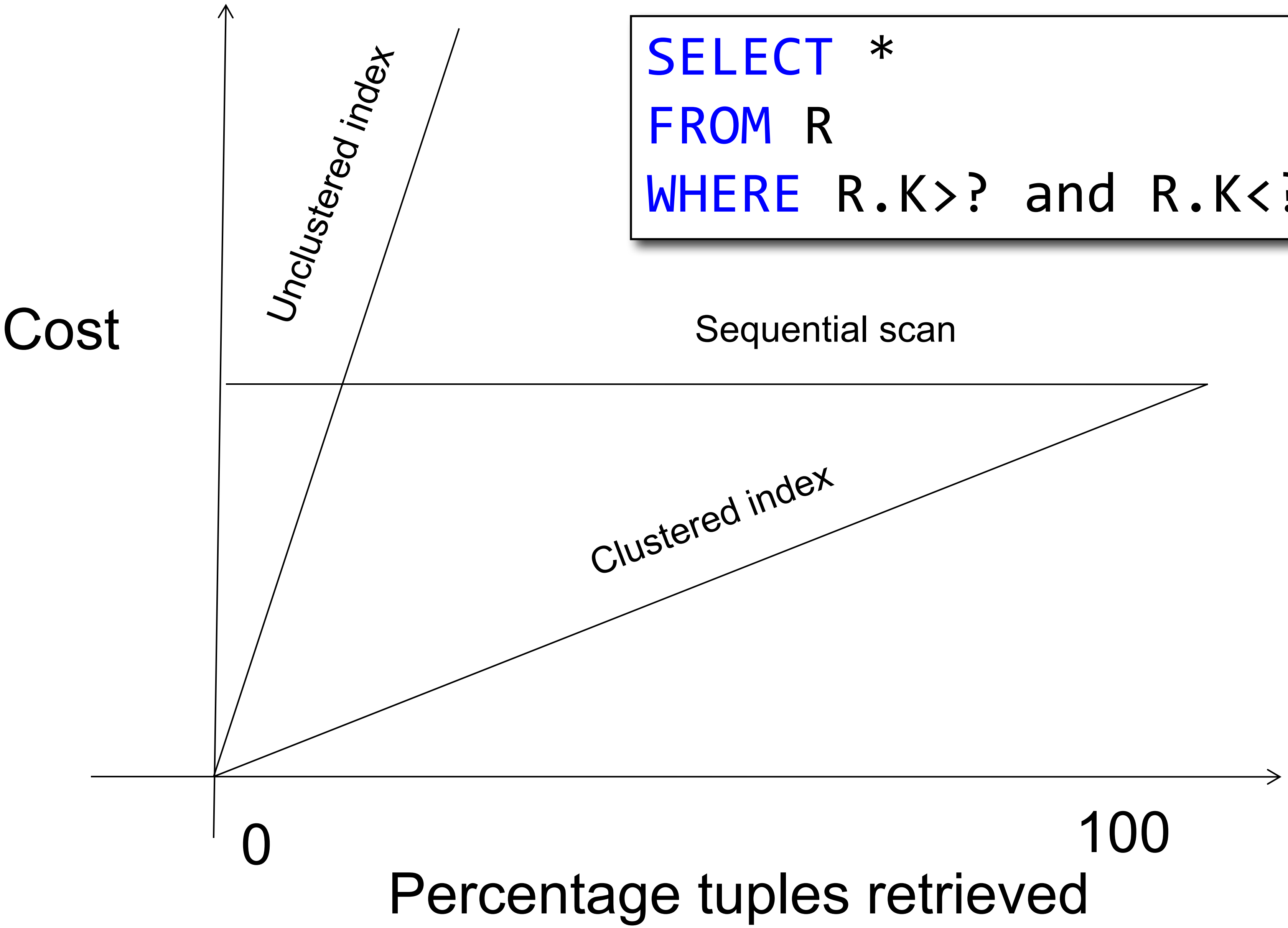


```
SELECT *
FROM R
WHERE R.K>? and R.K<?
```



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





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

CHOOSING INDEX IS NOT ENOUGH

To estimate the cost of a query plan, we still need to consider other factors:

- How each operator is implemented
- The cost of each operator
- Let’s start with the basics

COST PARAMETERS

Cost = I/O + CPU + Network BW

- We will focus on I/O in this class

Parameters (a.k.a. statistics):

- **B(R)** = # of blocks (i.e., pages) for relation R
- **T(R)** = # of tuples in relation R
- **V(R, a)** = # of distinct values of attribute a

COST PARAMETERS

Cost = I/O + CPU + Network BW

- We will focus on I/O in this class

Parameters (a.k.a. statistics):

- **B(R)** = # of blocks (i.e., pages) for relation R
- **T(R)** = # of tuples in relation R
- **V(R, a)** = # of distinct values of attribute a

When **a** is a key, **V(R,a) = T(R)**
When **a** is not a key, **V(R,a)** can be anything <= **T(R)**

COST PARAMETERS

Cost = I/O + CPU + Network BW

- We will focus on I/O in this class

Parameters (a.k.a. statistics):

- **B(R)** = # of blocks (i.e., pages) for relation R
- **T(R)** = # of tuples in relation R
- **V(R, a)** = # of distinct values of attribute a

When **a** is a key, **V(R,a) = T(R)**
When **a** is not a key, **V(R,a)** can be anything <= **T(R)**

DBMS collects statistics about base tables must infer them for intermediate results

SELECTIVITY FACTORS FOR CONDITIONS

$A = c$ /* $\sigma_{A=c}(R)$ */

- Selectivity = $1/V(R,A)$

$A < c$ /* $\sigma_{A<c}(R)$*/

- Selectivity = $(c - \min(R, A))/(\max(R,A) - \min(R,A))$

$c1 < A < c2$ /* $\sigma_{c1<A<c2}(R)$*/

- Selectivity = $(c2 – c1)/(\max(R,A) - \min(R,A))$

COST OF READING DATA FROM DISK

Sequential scan for relation R costs B(R)

Index-based selection

- Estimate selectivity factor **f** (see previous slide)
- Clustered index: f***B(R)**
- Unclustered index f***T(R)**

Note: we ignore I/O cost for index pages

INDEX BASED SELECTION

Example:

B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

Table scan:

Index based selection:

INDEX BASED SELECTION

Example:

$B(R) = 2000$
$T(R) = 100{,}000$
$V(R, a) = 20$

cost of $\sigma_{a=v}(R)$ = ?

Table scan: B(R) = 2,000 I/Os

Index based selection:

INDEX BASED SELECTION

Example:

B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

Table scan: B(R) = 2,000 I/Os

Index based selection:

- If index is clustered:
- If index is unclustered:

INDEX BASED SELECTION

Example:

B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

Table scan: B(R) = 2,000 I/Os

Index based selection:

- If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
- If index is unclustered:

INDEX BASED SELECTION

Example:

B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

Table scan: B(R) = 2,000 I/Os

Index based selection:

- If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
- If index is unclustered: T(R) * 1/V(R,a) = 5,000 I/Os

INDEX BASED SELECTION

Example:

B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

Table scan: B(R) = 2,000 I/Os

Index based selection:

- If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
- If index is unclustered: T(R) * 1/V(R,a) = 5,000 I/Os

Lesson: Don't build unclustered indexes when V(R,a) is small !

OUTLINE

Join operator algorithms

- One-pass algorithms (Sec. 15.2 and 15.3)
- Index-based algorithms (Sec 15.6)

Note about readings:

- In class, we discuss only algorithms for joins
- Other operators are easier: read the book

JOIN ALGORITHMS

Hash join

Nested loop join

Sort-merge join

HASH JOIN

Hash join: R ⋈ S

Scan R, build buckets in main memory

Then scan S and join

Cost: B(R) + B(S)

Which relation to build the hash table on?

HASH JOIN

Hash join: R ⋈ S

Scan R, build buckets in main memory

Then scan S and join

Cost: B(R) + B(S)

Which relation to build the hash table on?

One-pass algorithm when B(R) ≤ M

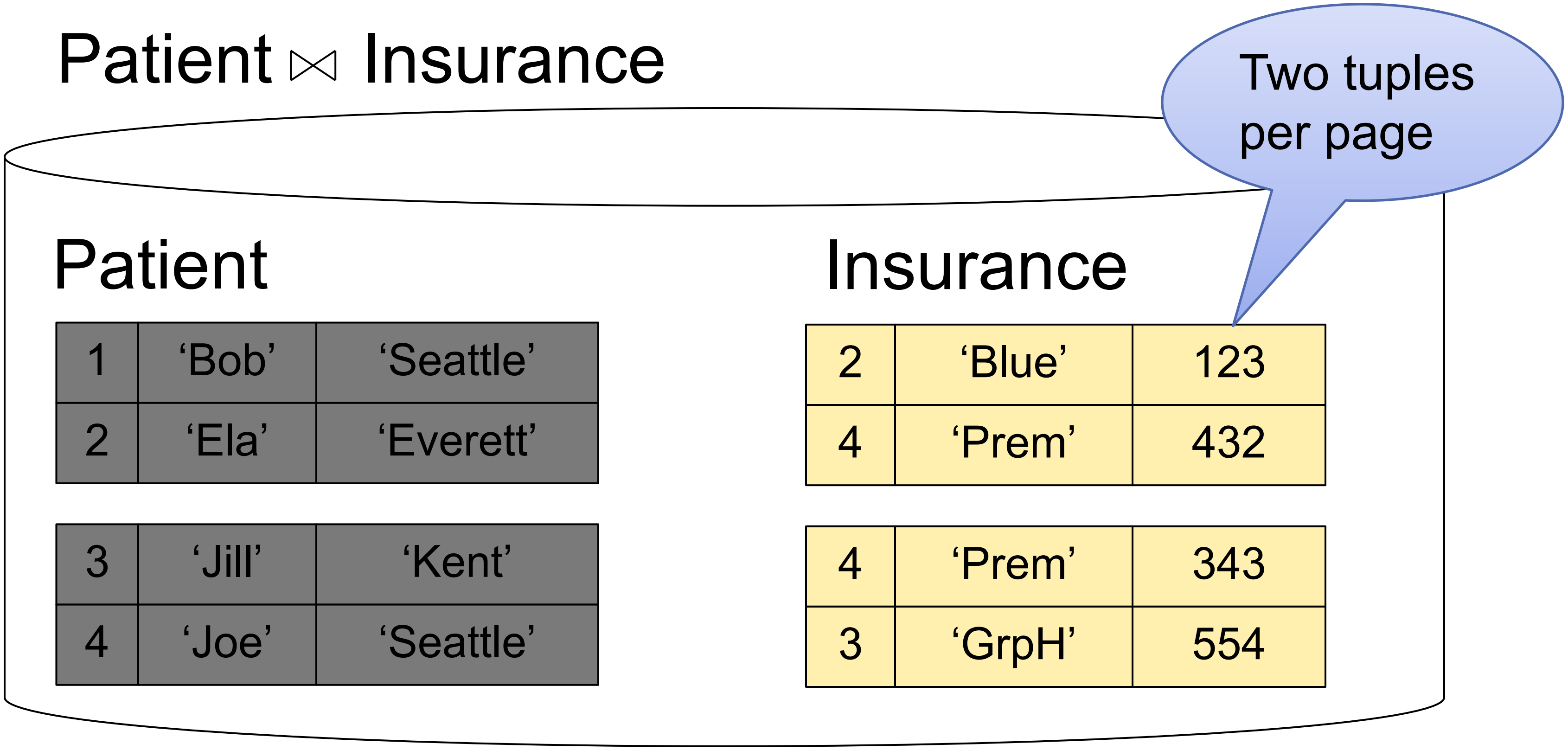
- M = number of memory pages available

HASH JOIN EXAMPLE

Patient(pid, name, address)

Insurance(pid, provider, policy_nb)

Patient $\bowtie$ Insurance



Two tuples per page

Patient

1	'Bob'	'Seattle'
2	'Ela'	'Everett'

3	'Jill'	'Kent'
4	'Joe'	'Seattle'

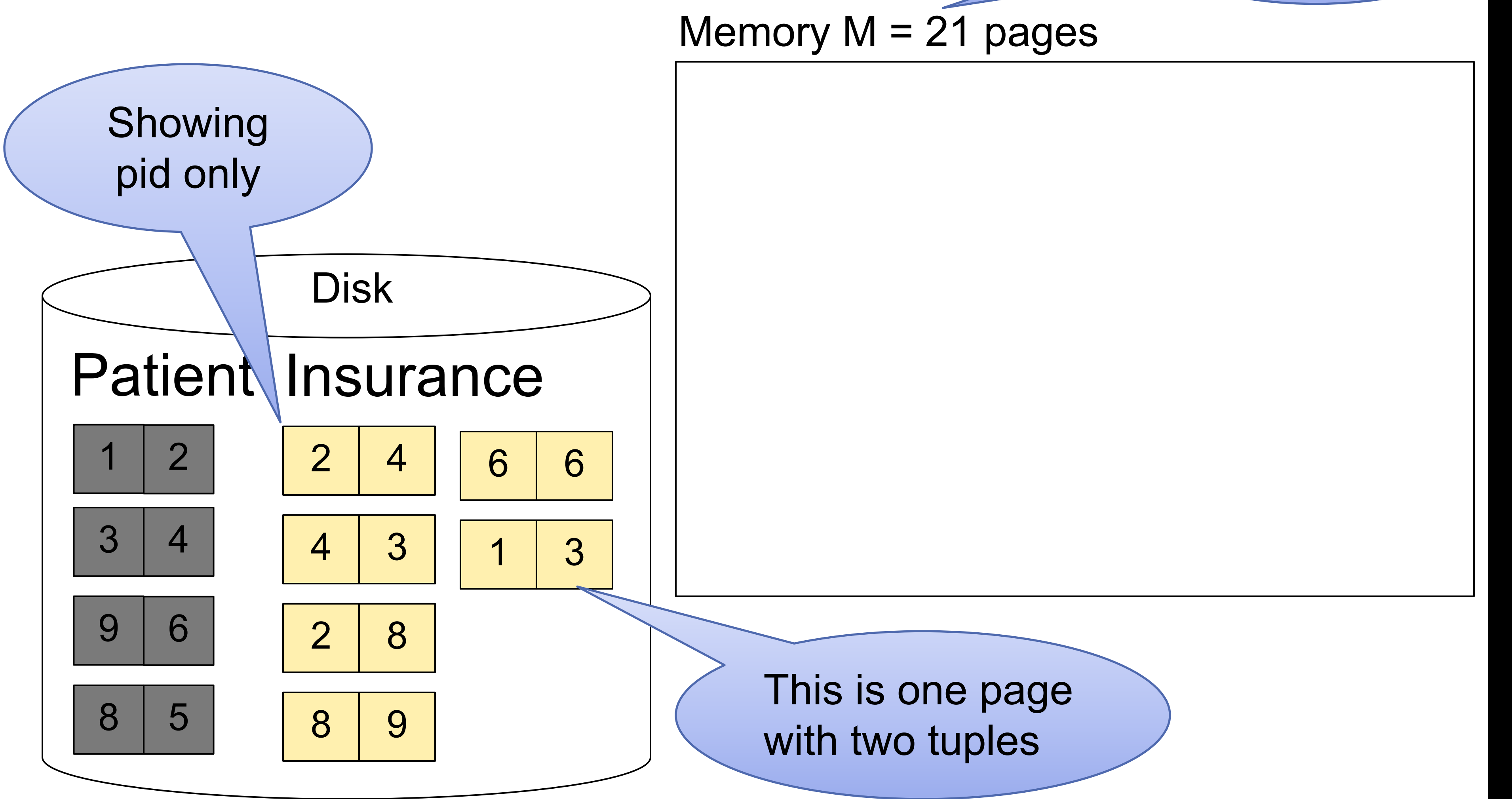
Insurance

2	'Blue'	123
4	'Prem'	432

4	'Prem'	343
3	'GrpH'	554

HASH JOIN EXAMPLE

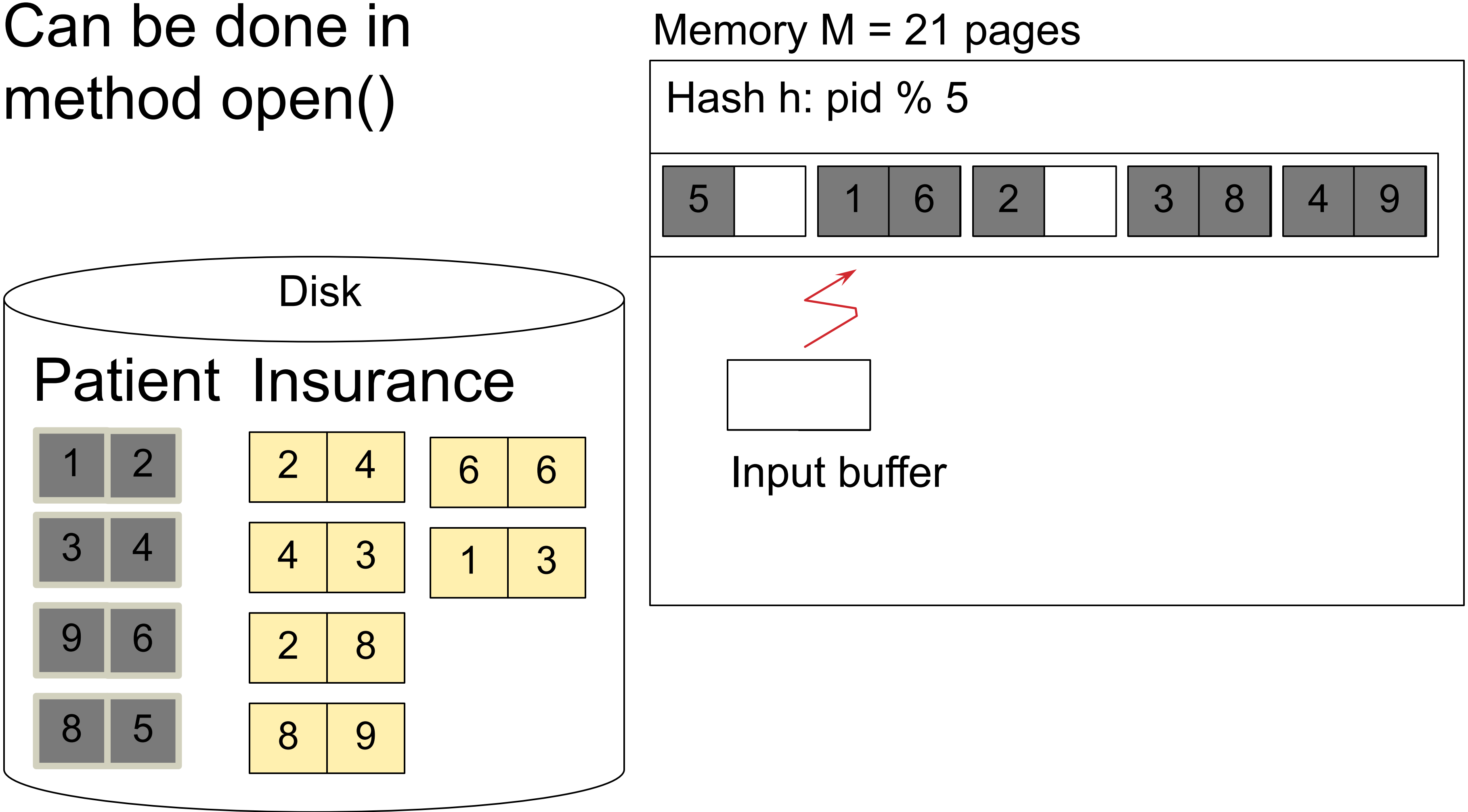
Patient ⋈ Insurance



HASH JOIN EXAMPLE

Step 1: Scan Patient and build hash table in memory

Can be done in method open()



Memory M = 21 pages

Hash h: pid % 5

Input buffer

Disk

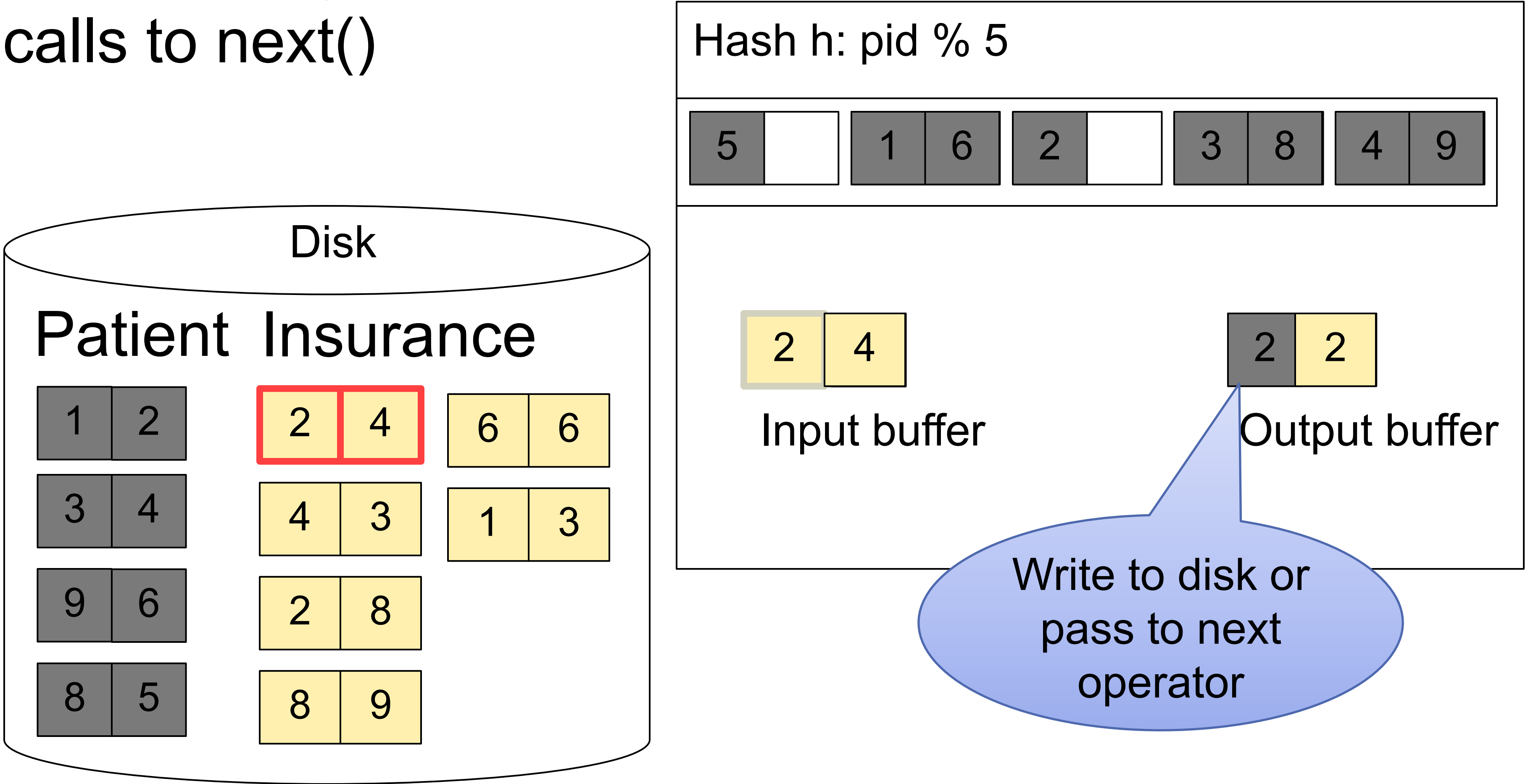
Patient Insurance

HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table

Done during
calls to next()

Memory M = 21 pages



Hash h: pid % 5

Disk

Patient Insurance

Input buffer

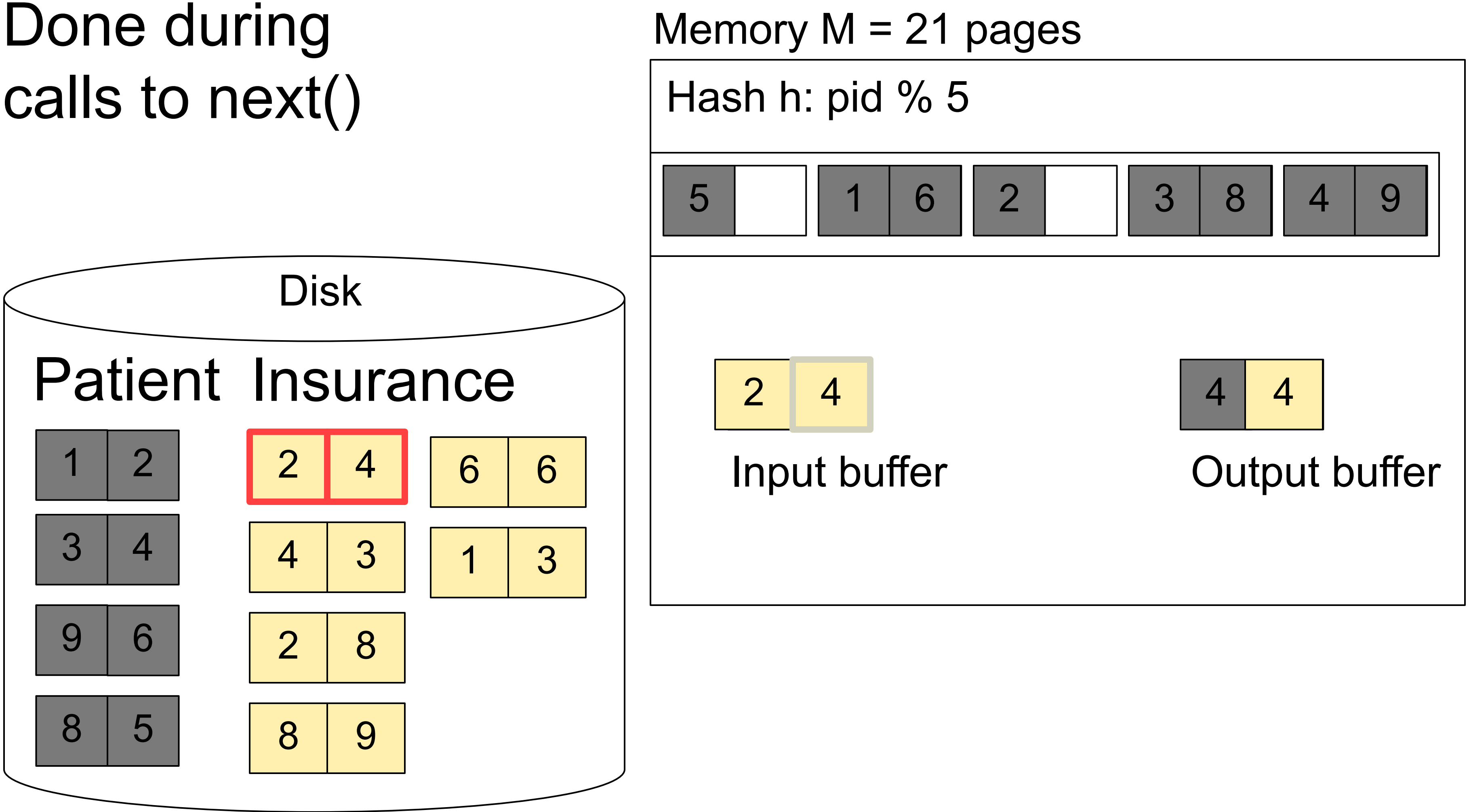
Output buffer

Write to disk or pass to next operator

HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

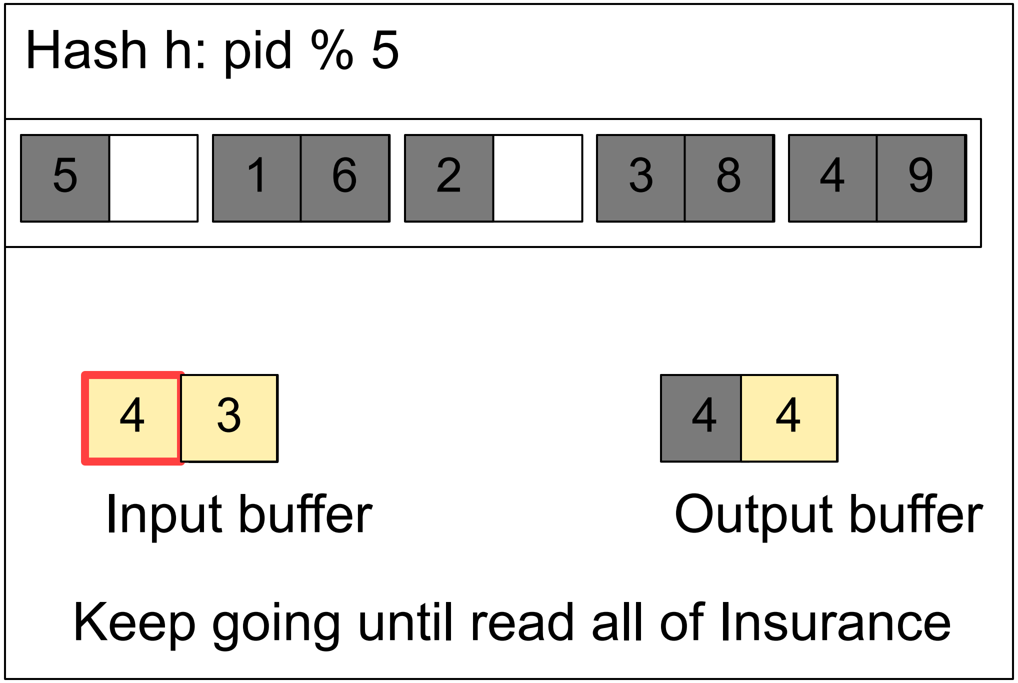


HASH JOIN EXAMPLE

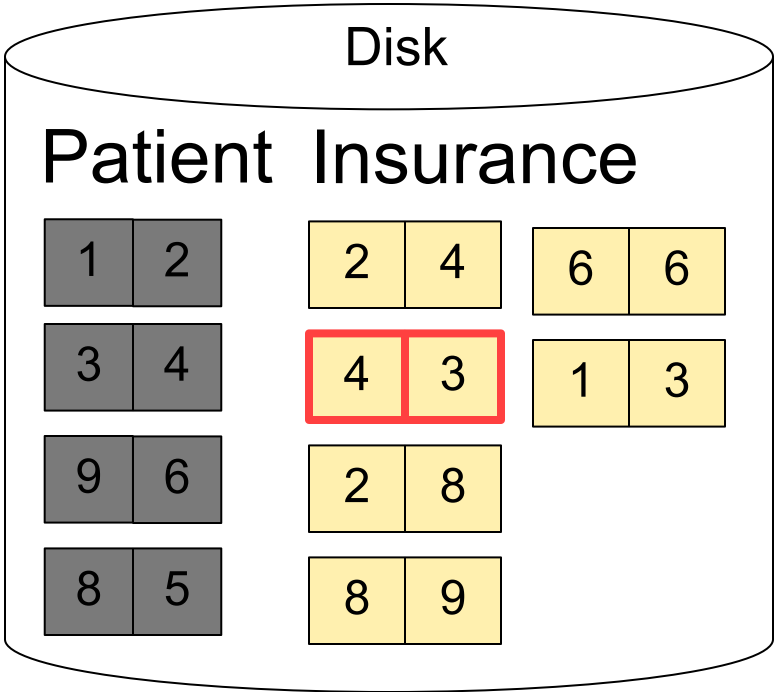
Step 2: Scan Insurance and probe into hash table

Done during
calls to next()

Memory M = 21 pages



Keep going until read all of Insurance



Cost: B(R) + B(S)

NESTED LOOP JOINS

Tuple-based nested loop $R \bowtie S$

R is the outer relation, S is the inner relation

```
for each tuple $t_1$ in R do
   for each tuple $t_2$ in S do
       if $t_1$ and $t_2$ join then output ($t_1$,$t_2$)
```

What is the Cost?

NESTED LOOP JOINS

Tuple-based nested loop R ⋈ S

R is the outer relation, S is the inner relation

```
for each tuple $t_1$ in R do
   for each tuple $t_2$ in S do
       if $t_1$ and $t_2$ join then output ($t_1$,$t_2$)
```

Cost: B(R) + T(R) B(S)

Multiple-pass since S is read many times

What is the Cost?

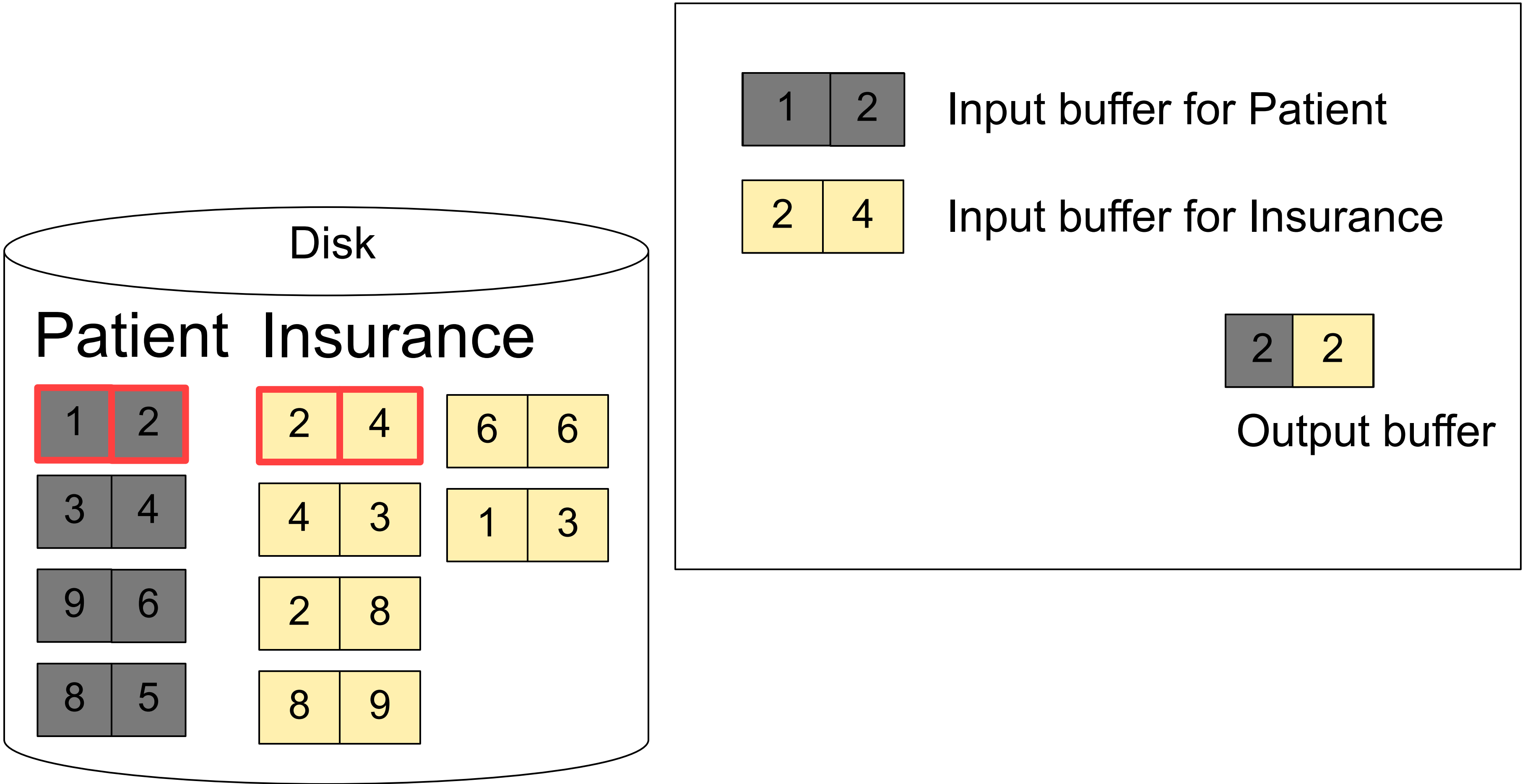
PAGE-AT-A-TIME REFINEMENT

```
for each page of tuples r in R do
  for each page of tuples s in S do
    for all pairs of tuples $t_1$ in r, $t_2$ in s
        if $t_1$ and $t_2$ join then output $(t_1,t_2)$
```

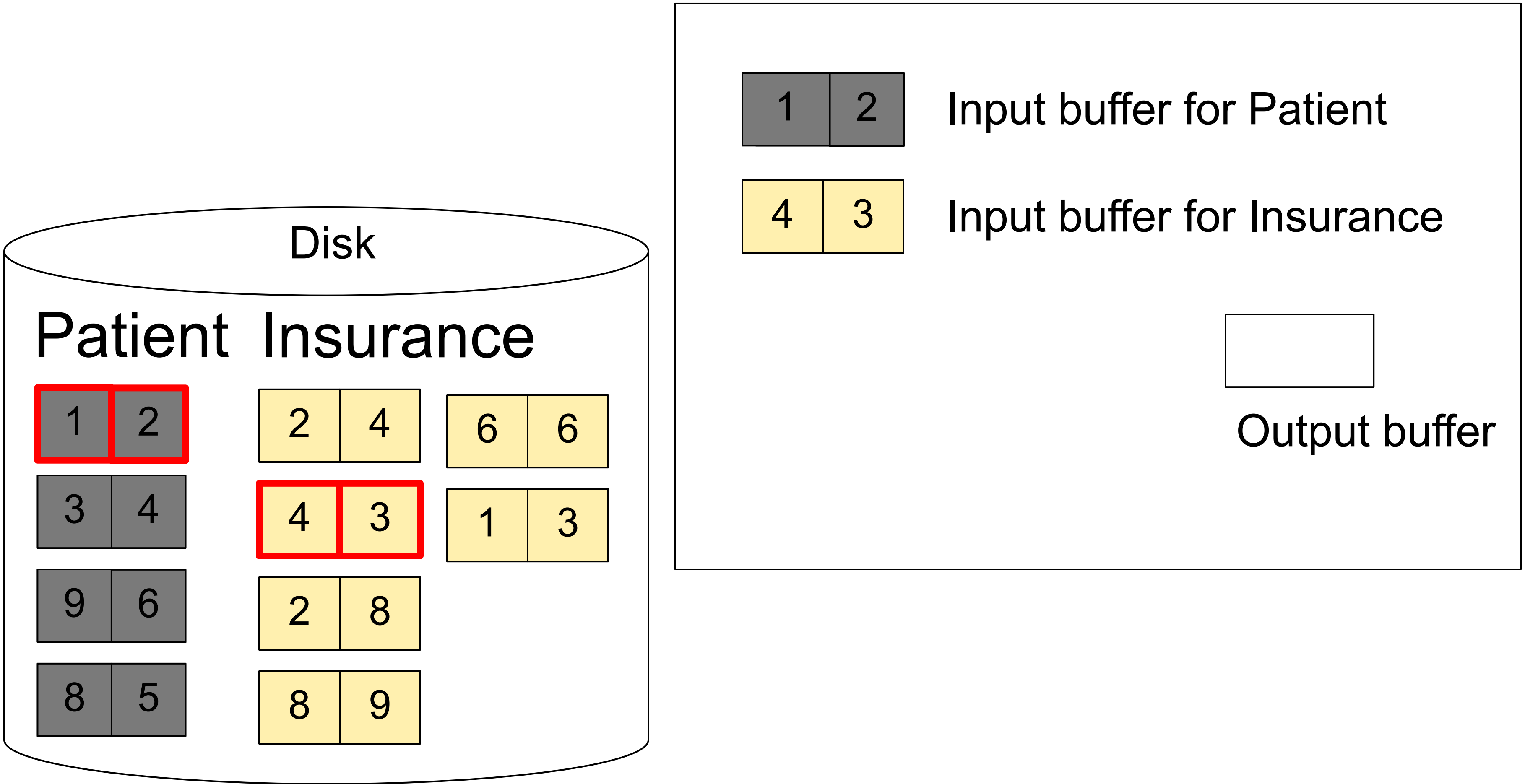
Cost: B(R) + B(R)B(S)

What is the Cost?

PAGE-AT-A-TIME REFINEMENT



PAGE-AT-A-TIME REFINEMENT

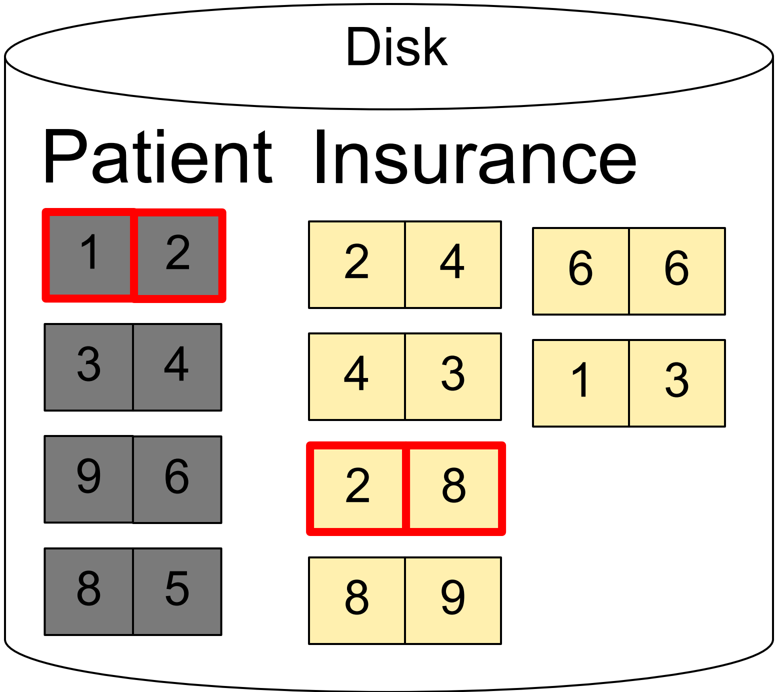


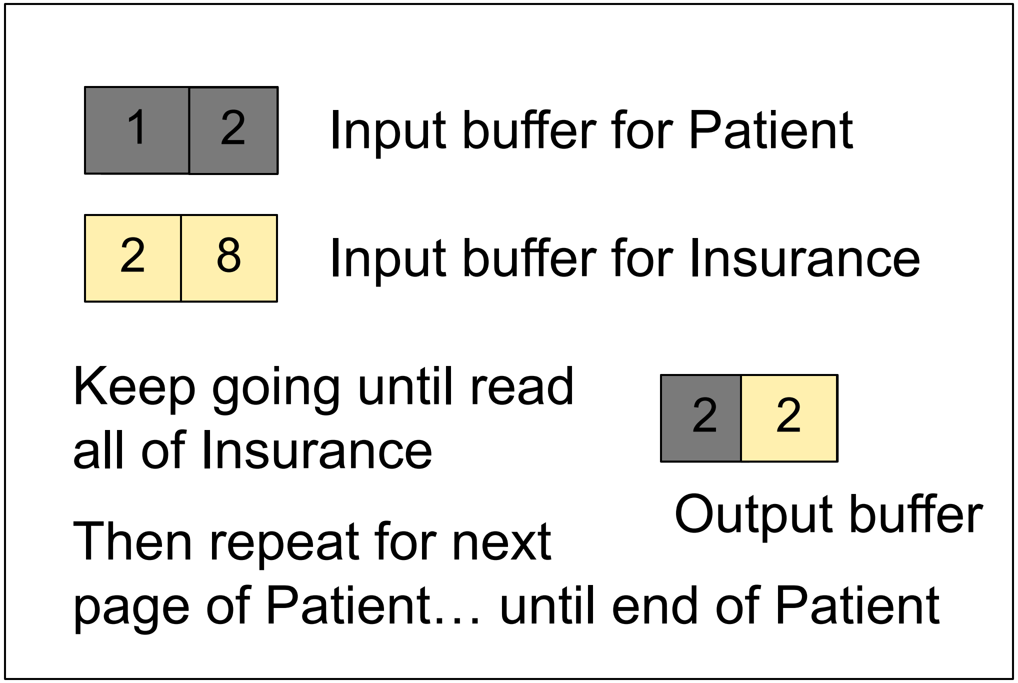
Input buffer for Patient

Input buffer for Insurance

Output buffer

PAGE-AT-A-TIME REFINEMENT





Input buffer for Patient

Input buffer for Insurance

Keep going until read all of Insurance

Output buffer

Then repeat for next page of Patient… until end of Patient

Cost: B(R) + B(R)B(S)

BLOCK-NESTED-LOOP REFINEMENT

```
for each group of M-1 pages r in R do
   for each page of tuples s in S do
      for all pairs of tuples $t_1$ in r, $t_2$ in s
            if $t_1$ and $t_2$ join then output ($t_1$,$t_2$)
```

Cost: B(R) + B(R)B(S)/(M-1)

What is the Cost?

SORT-MERGE JOIN

Sort-merge join: $R \bowtie S$

Scan R and sort in main memory

Scan S and sort in main memory

Merge R and S

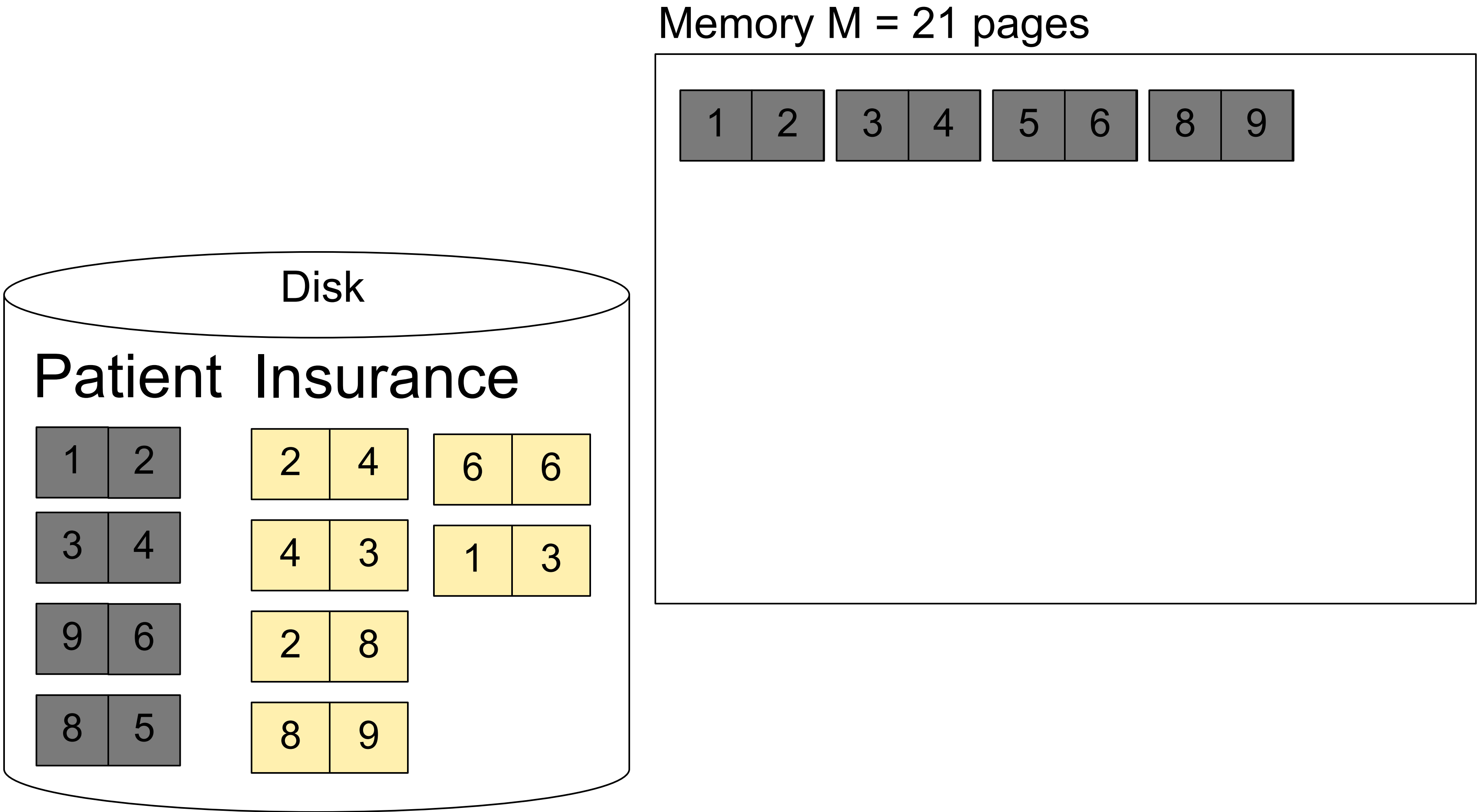
Cost: B(R) + B(S)

One pass algorithm when B(S) + B(R) <= M

Typically, this is NOT a one pass algorithm

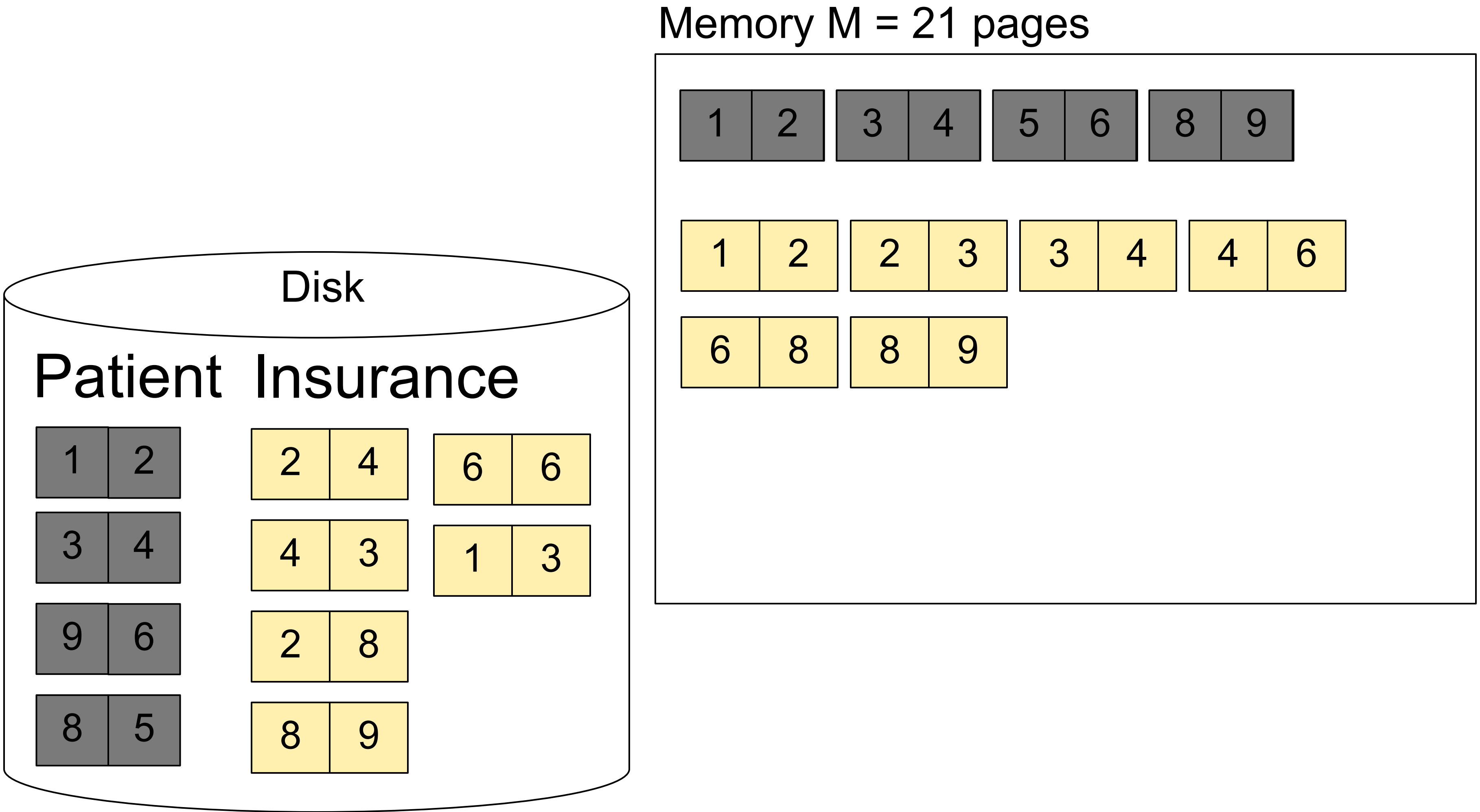
SORT-MERGE JOIN EXAMPLE

Step 1: Scan Patient and sort in memory



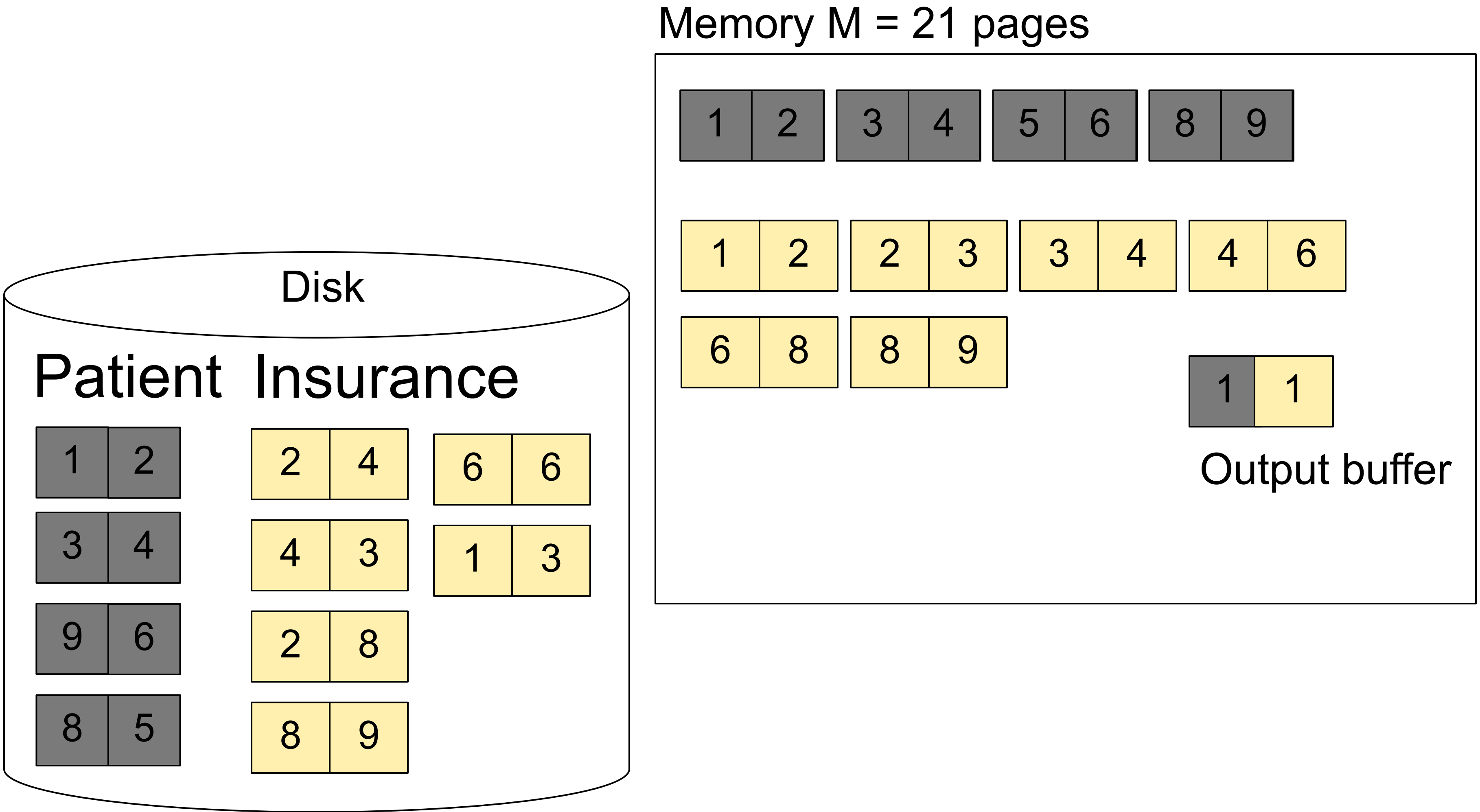
SORT-MERGE JOIN EXAMPLE

Step 2: Scan Insurance and sort in memory



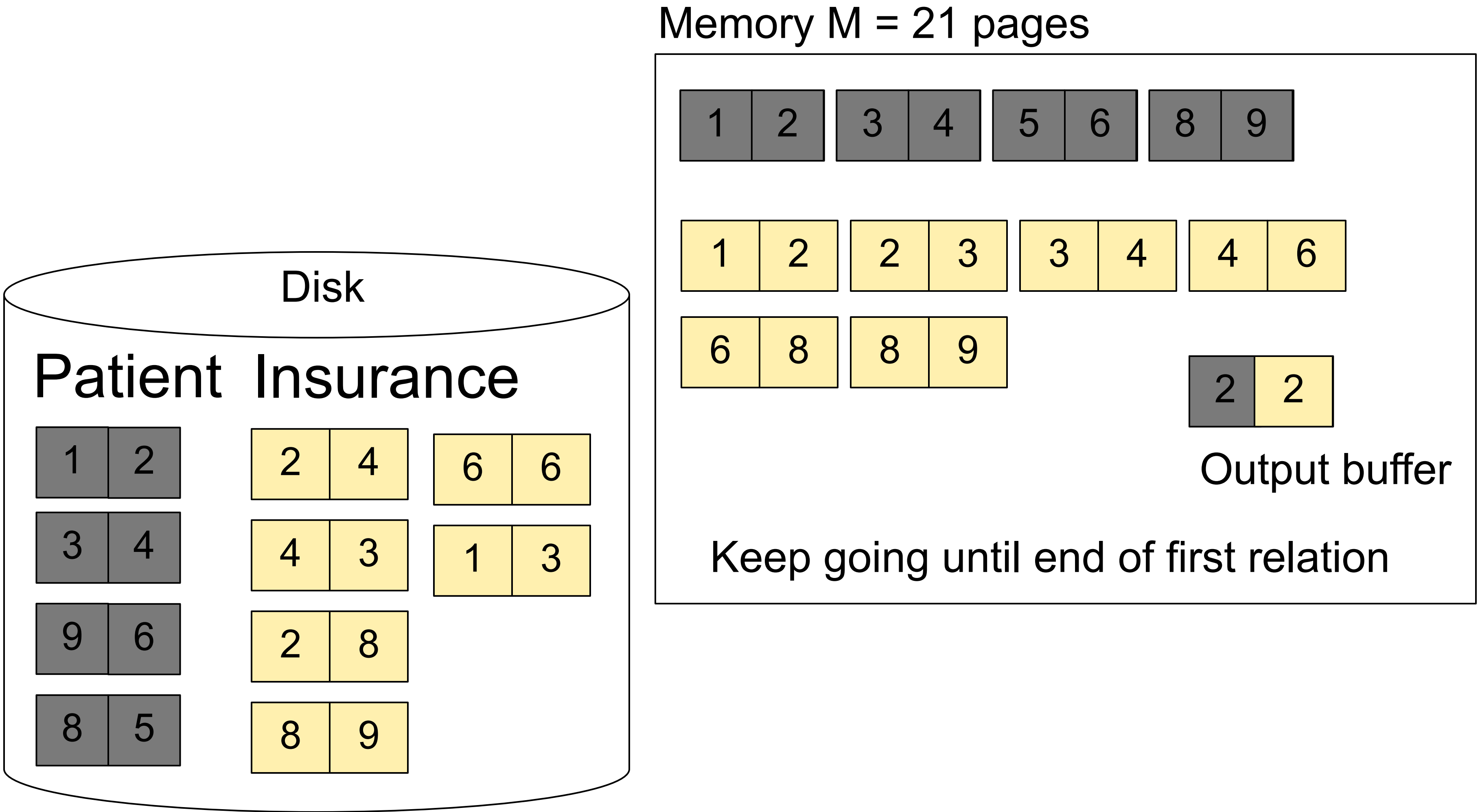
SORT-MERGE JOIN EXAMPLE

Step 3: Merge Patient and Insurance



SORT-MERGE JOIN EXAMPLE

Step 3: Merge Patient and Insurance



Memory M = 21 pages

Output buffer

Keep going until end of first relation

Disk

Patient Insurance

INDEX NESTED LOOP JOIN

$R \bowtie S$

Assume S has an index on the join attribute

Iterate over R, for each tuple fetch corresponding tuple(s) from S

Cost:

- If index on S is clustered:
  $B(R) + T(R) * (B(S) * 1/V(S,a))$
- If index on S is unclustered:
  $B(R) + T(R) * (T(S) * 1/V(S,a))$