CSE 344

FEBRUARY 21ST – COST ESTIMATION

ADMINISTRIVIA

- **HW5 Due Tonight (11:30)**
- **OQ5 Due Friday (11:00)**
- **HW6 Due next Wednesday (Feb 28)**
- **HW7 Out Friday**
  - Entity Relations
  - Due TBD
- **HW8 Out Monday**
  - Due Mar 9th

BASIC INDEX SELECTION GUIDELINES

Consider queries in workload in order of importance

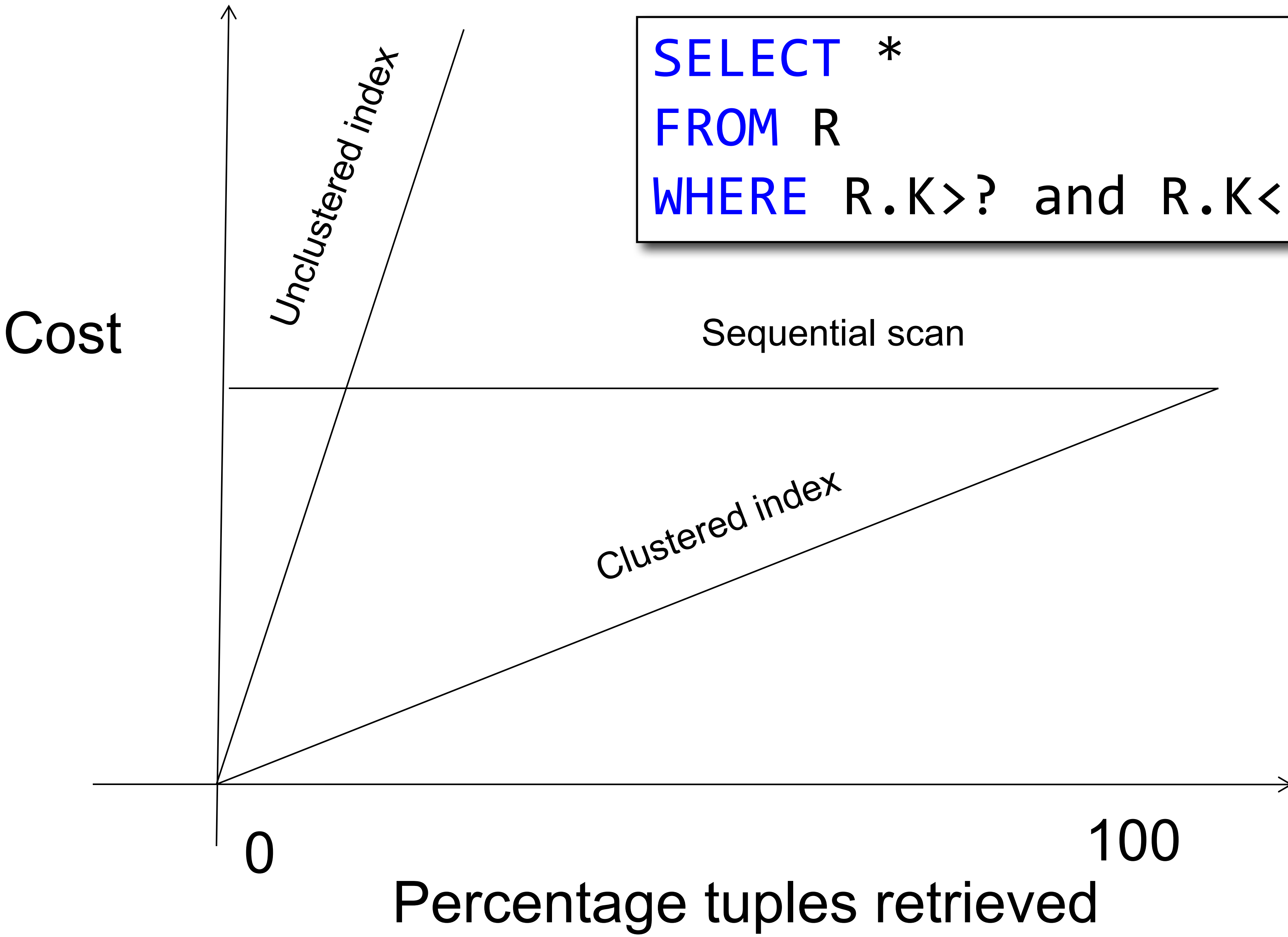
Consider relations accessed by query

- No point indexing other relations

Look at WHERE clause for possible search key

Try to choose indexes that speed-up multiple queries

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



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: 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'

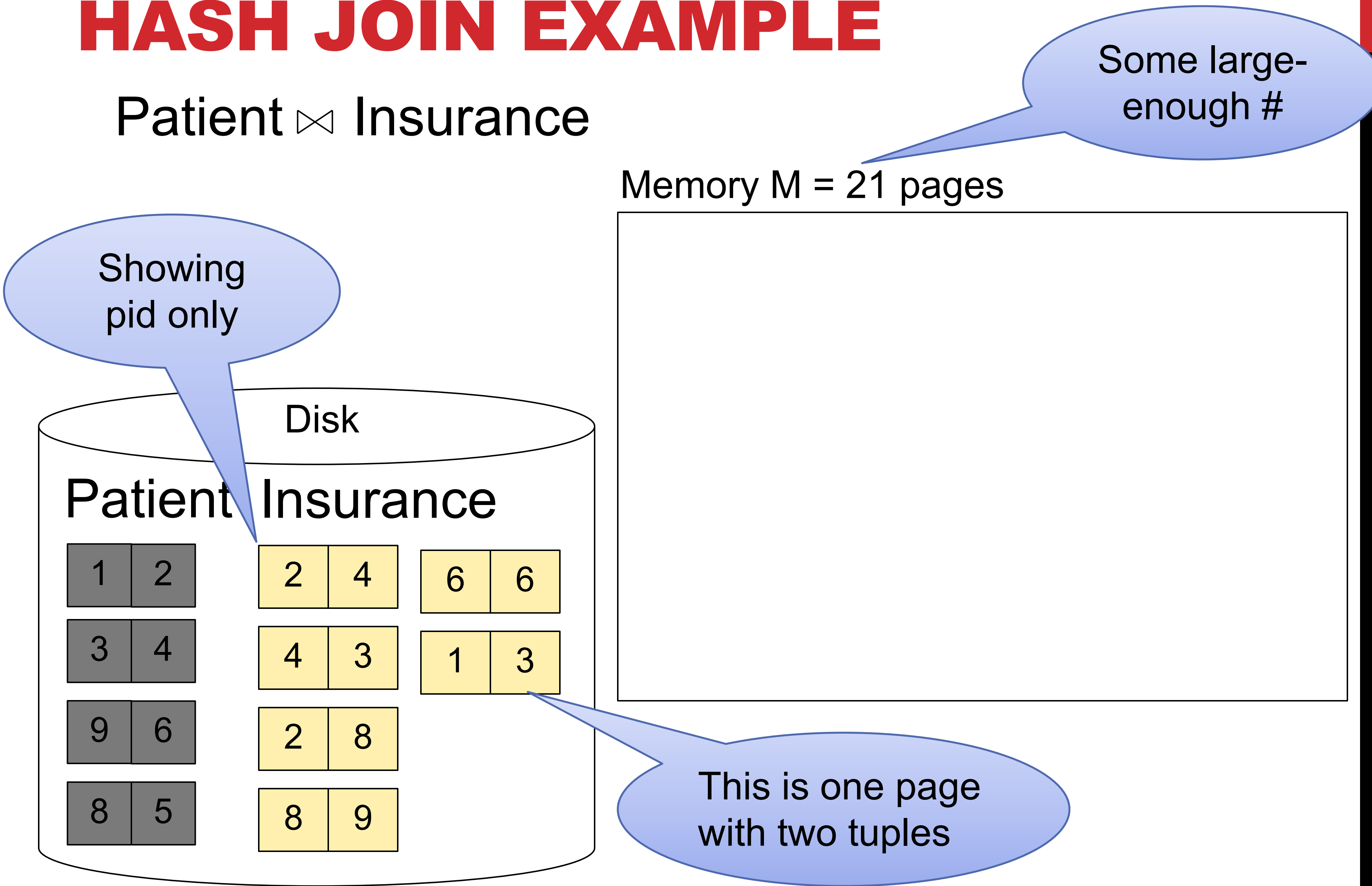
Insurance

2	'Blue'	123
4	'Prem'	432

4	'Prem'	343
3	'GrpH'	554

HASH JOIN EXAMPLE

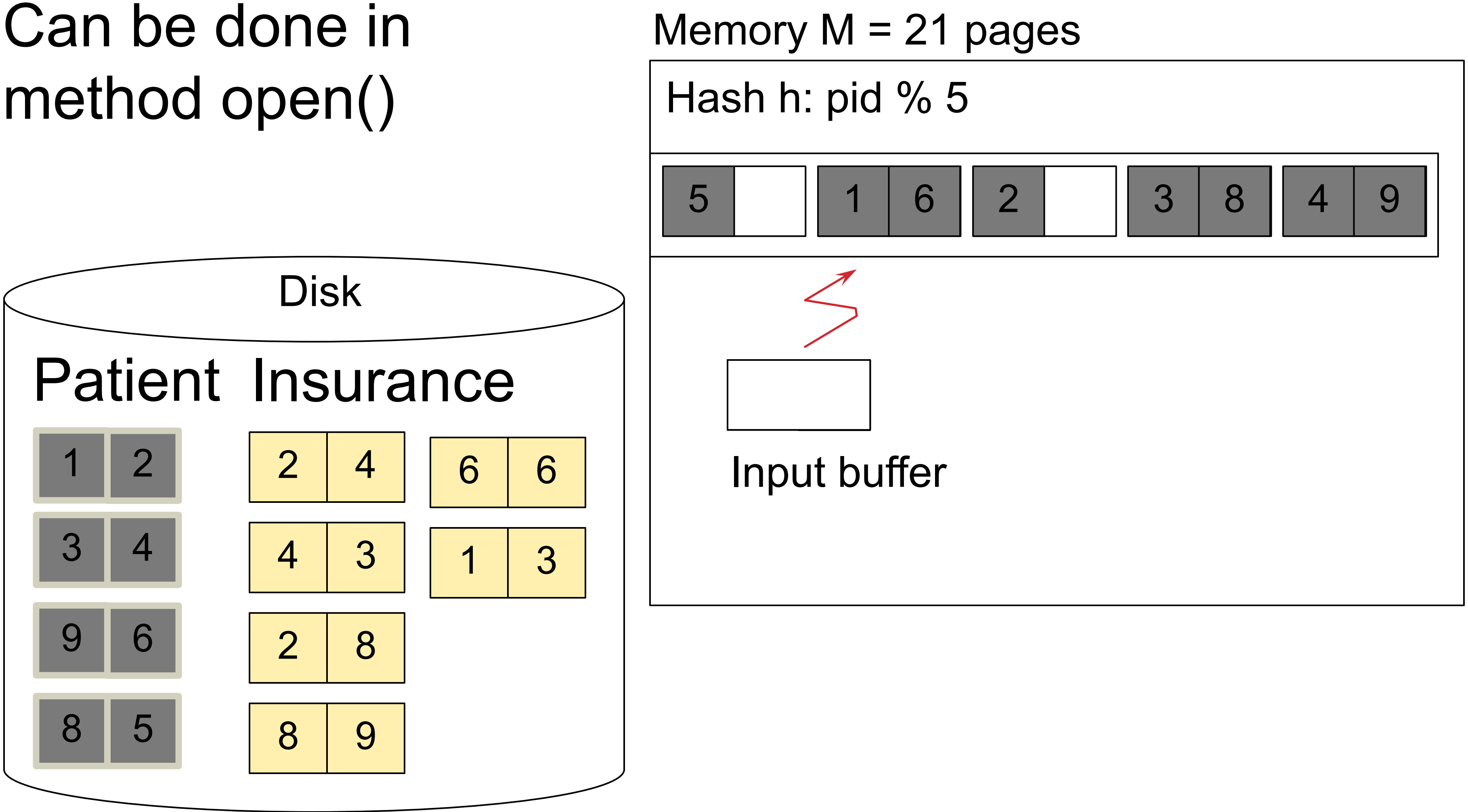
Patient $\bowtie$ Insurance



HASH JOIN EXAMPLE

Step 1: Scan Patient and build hash table in memory

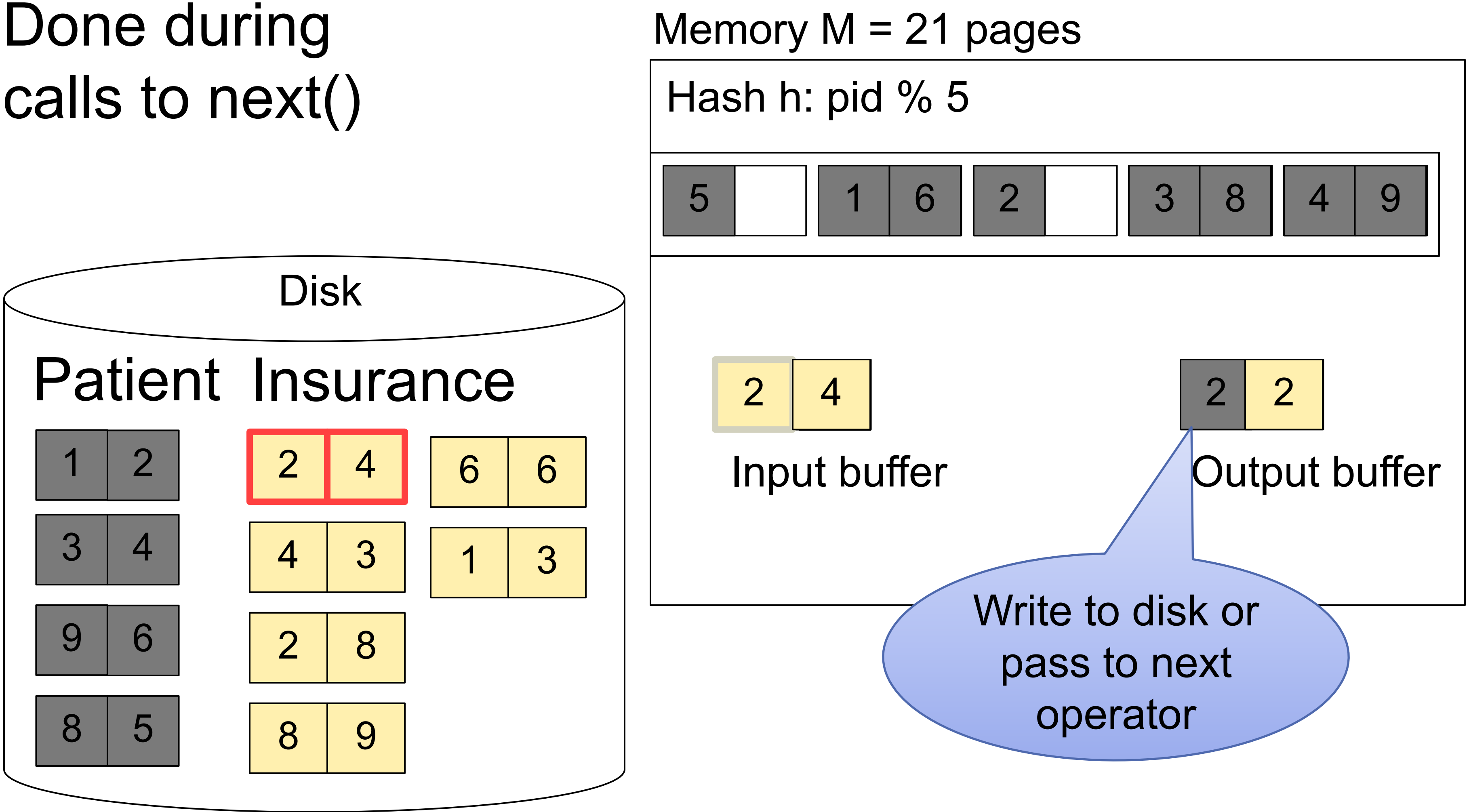
Can be done in method open()



HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table

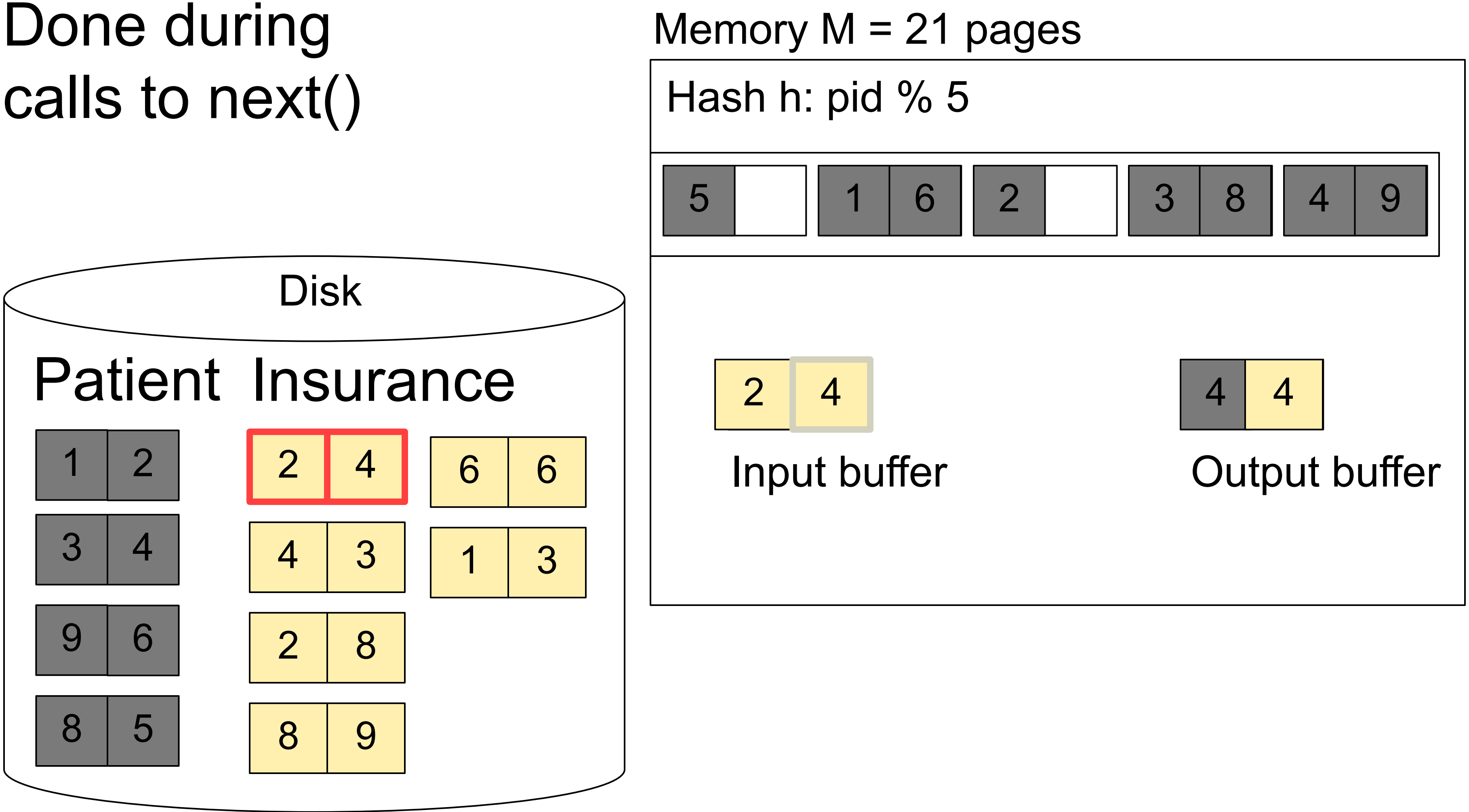
Done during calls to next()



HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

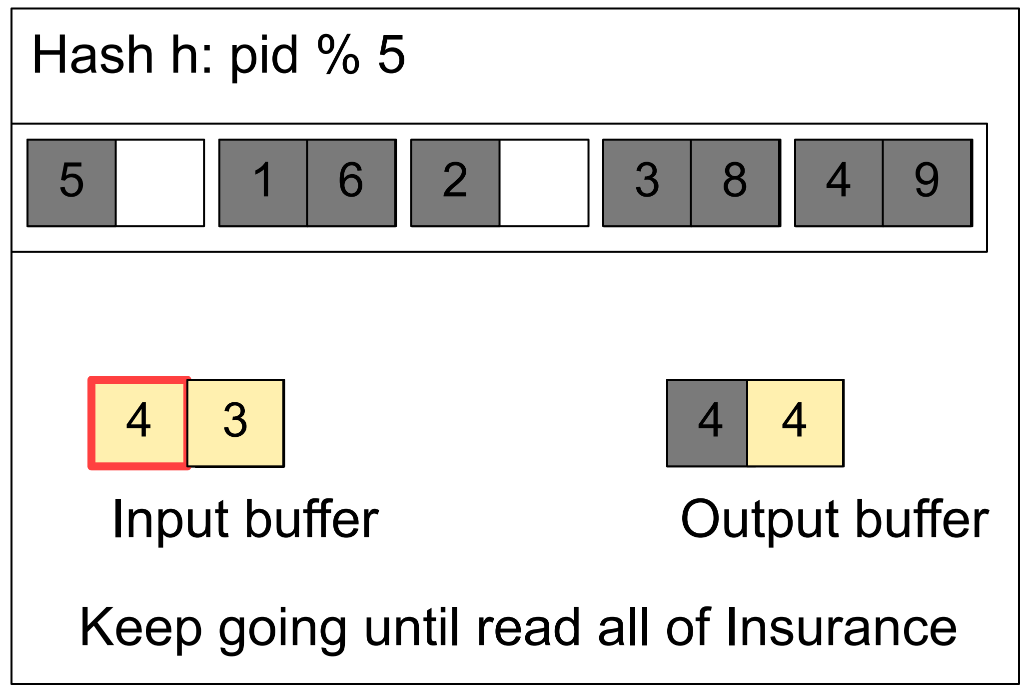


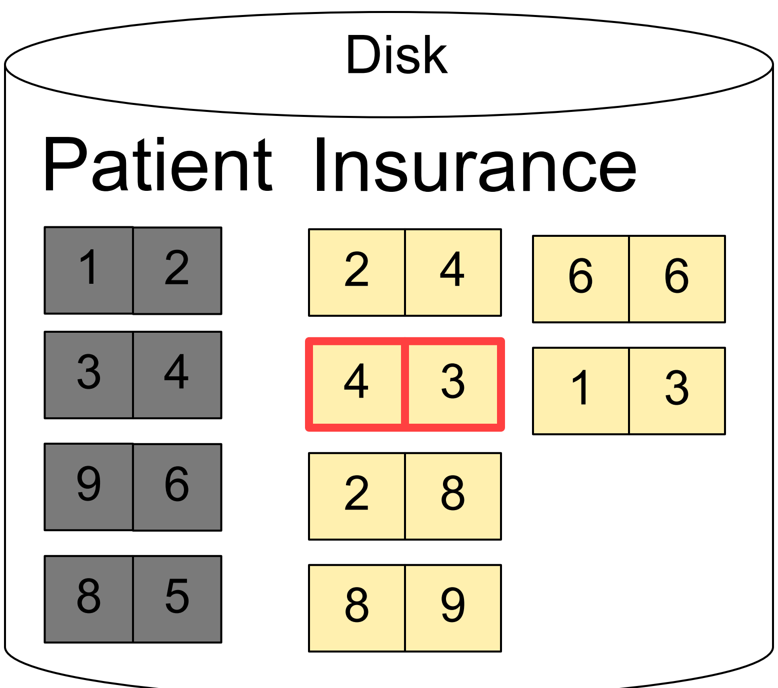
HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

Memory M = 21 pages





Cost: B(R) + B(S)

NESTED LOOP JOINS

Tuple-based nested loop R ⋈ S

R is the outer relation, S is the inner relation

```
for each tuple t1 in R do
   for each tuple t2 in S do
      if t1 and t2 join then output (t1,t2)
```

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 t1 in R do
   for each tuple t2 in S do
      if t1 and t2 join then output (t1,t2)
```

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

Multiple-pass since S is read many times

What is the Cost?

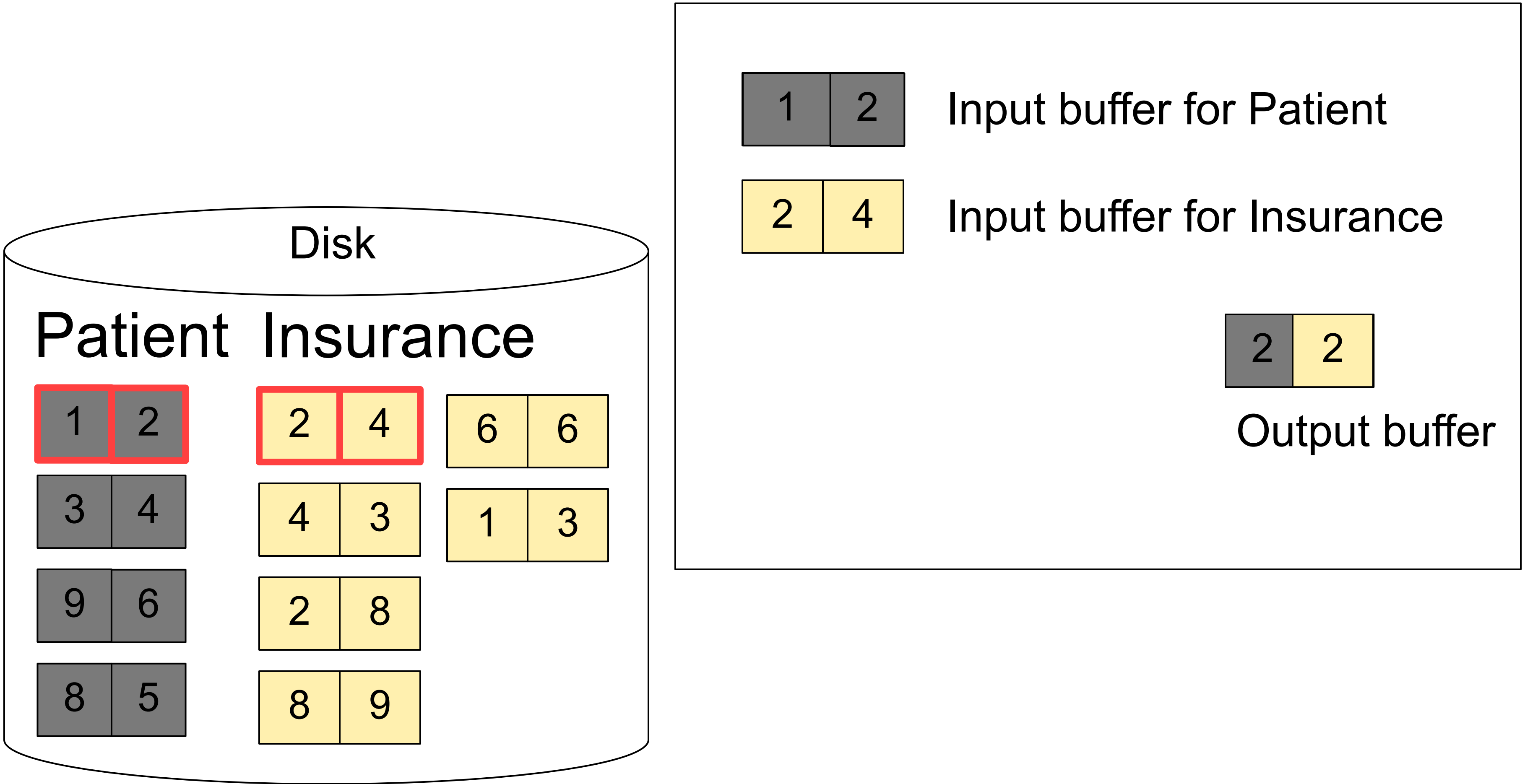
PAGE-AT-A-TIME REFINEMENT

```
for each page of tuples r in R do
   for each page of tuples s in S do
      for all pairs of tuples t1 in r, t2 in s
         if t1 and t2 join then output (t1,t2)
```

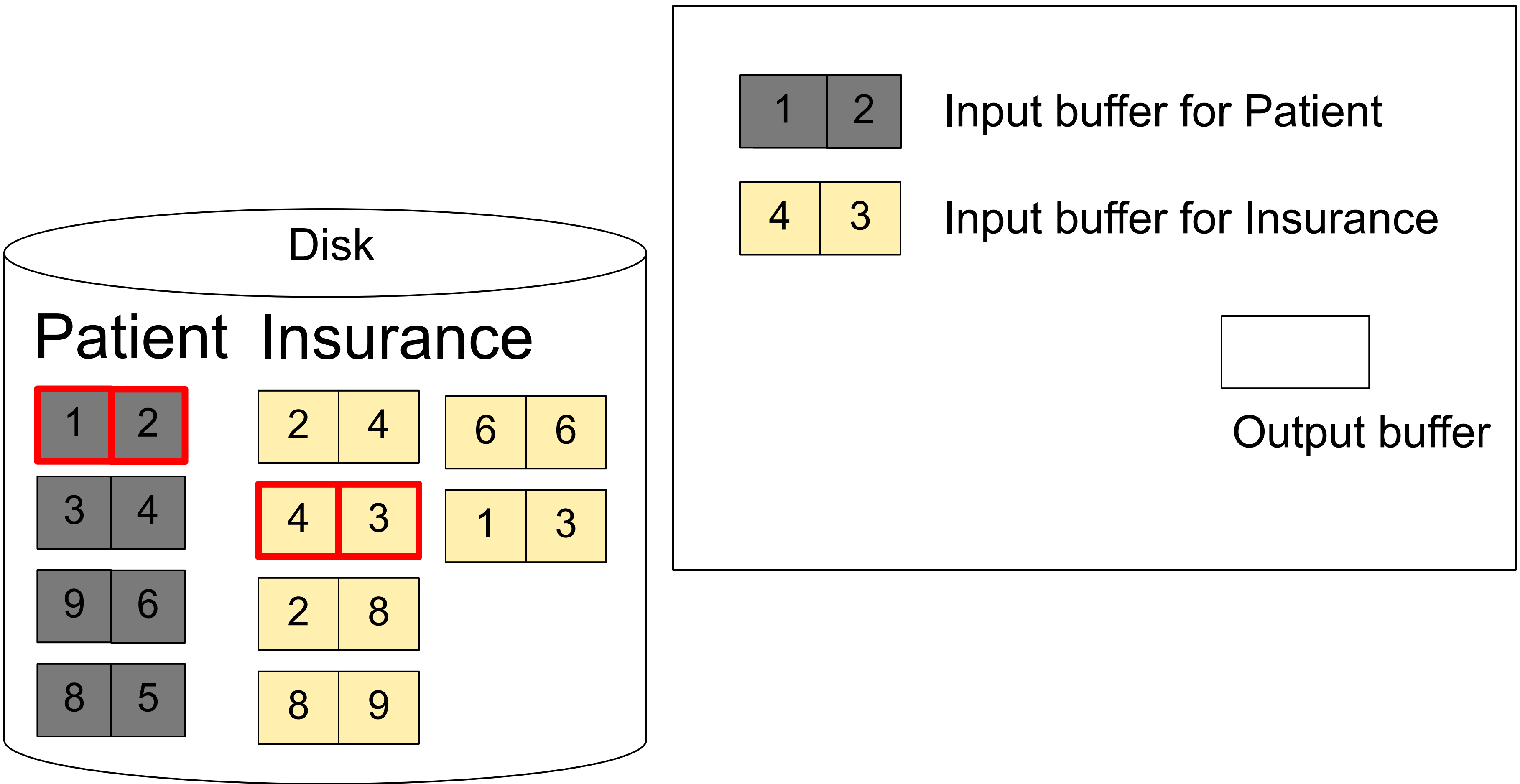
Cost: B(R) + B(R)B(S)

What is the Cost?

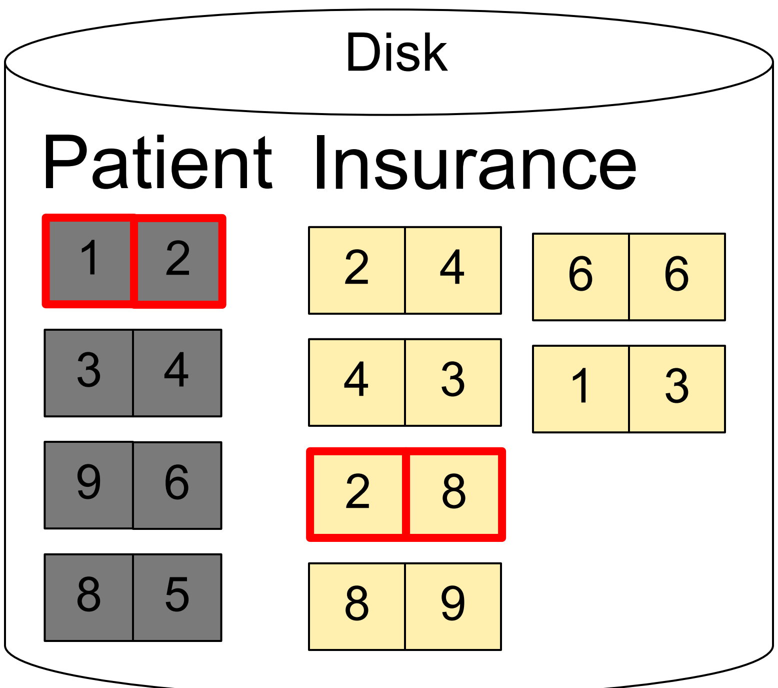
PAGE-AT-A-TIME REFINEMENT



PAGE-AT-A-TIME REFINEMENT



PAGE-AT-A-TIME REFINEMENT



1 2 Input buffer for Patient

2 8 Input buffer for Insurance

Keep going until read all of Insurance

2 2 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 t1 in r, t2 in s
         if t1 and t2 join then output (t1,t2)
```

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

What is the Cost?

SORT-MERGE JOIN

Sort-merge join: R ⋈ S

Scan R and sort in main memory

Scan S and sort in main memory

Merge R and S

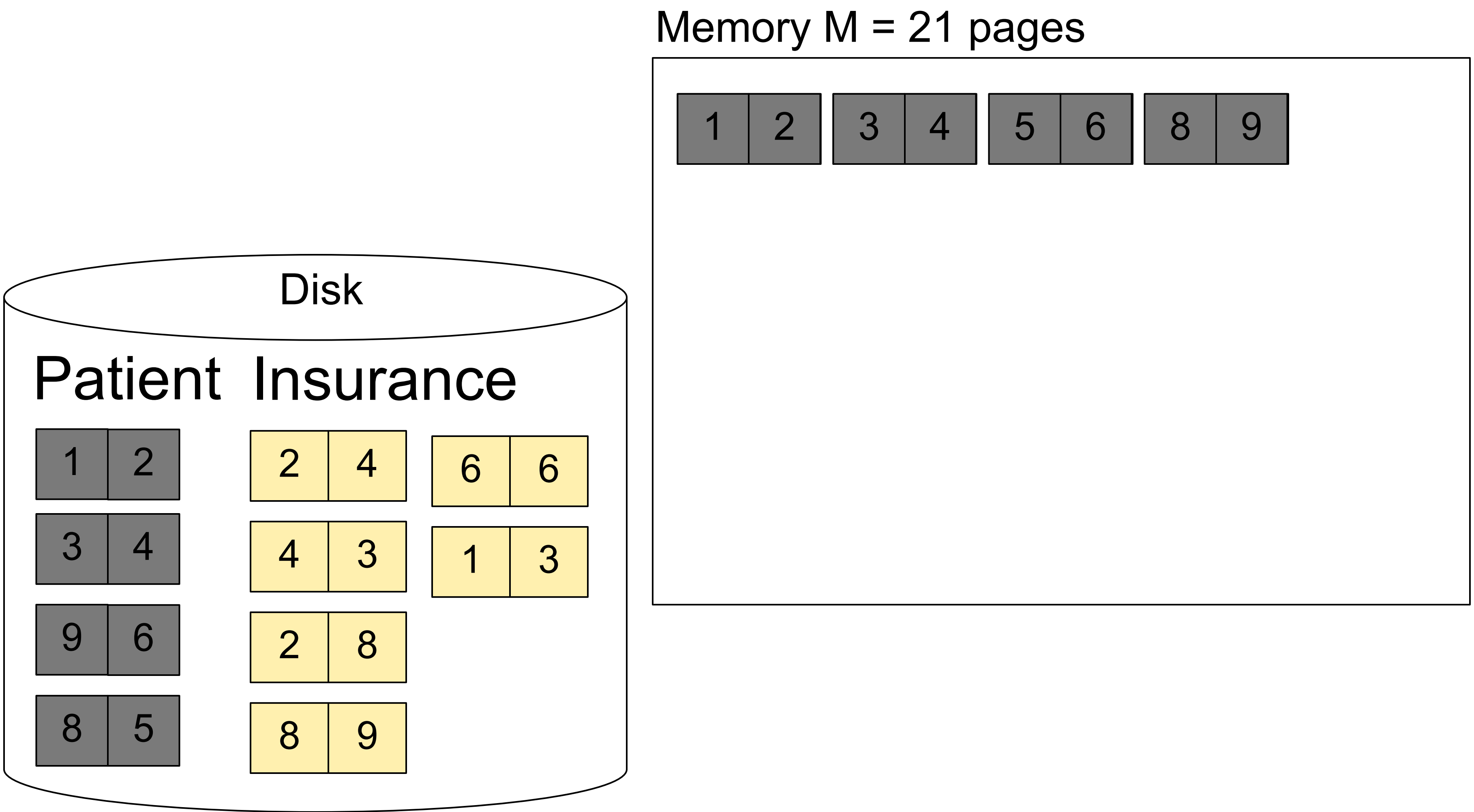
Cost: B(R) + B(S)

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

Typically, this is NOT a one pass algorithm

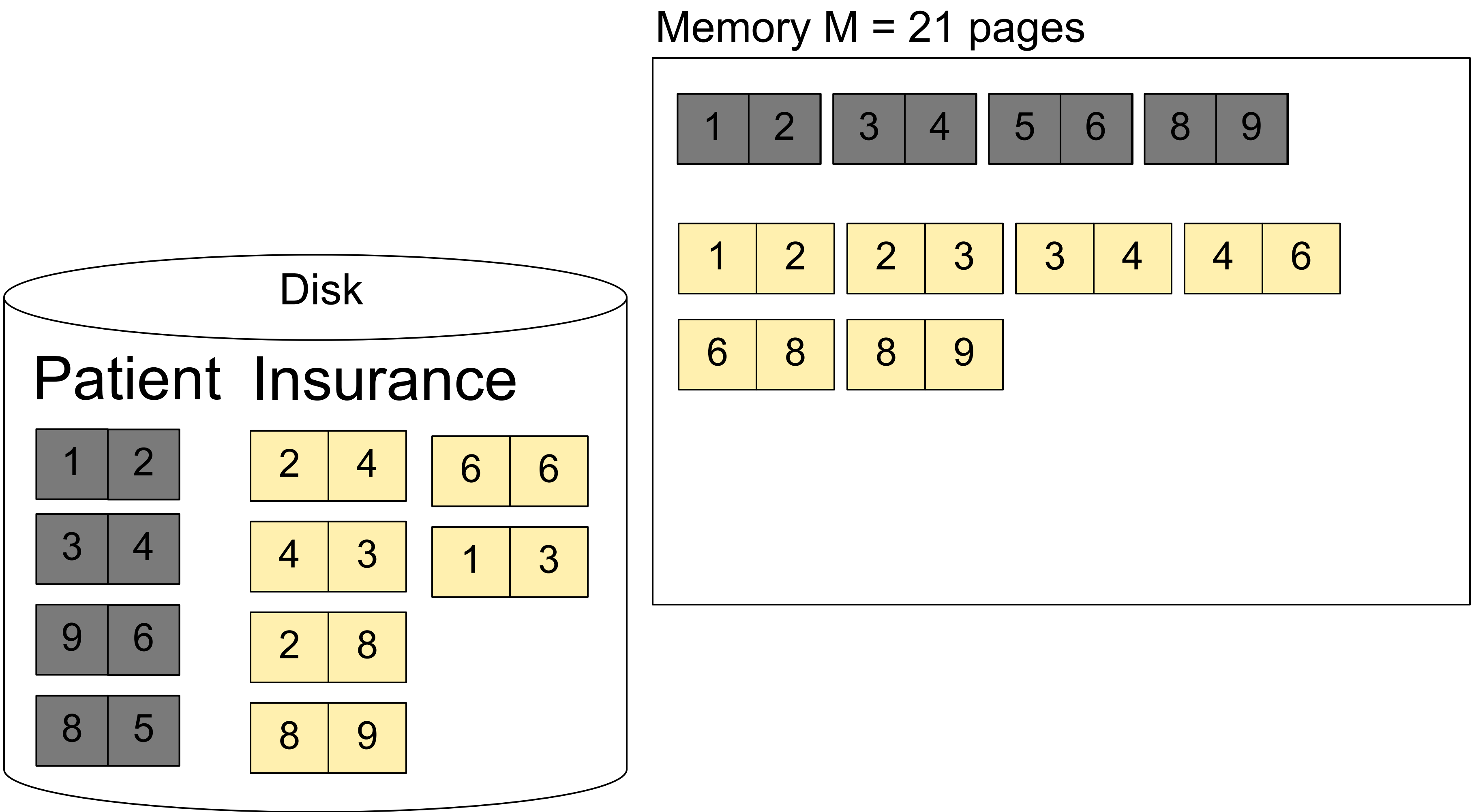
SORT-MERGE JOIN EXAMPLE

Step 1: Scan Patient and sort in memory



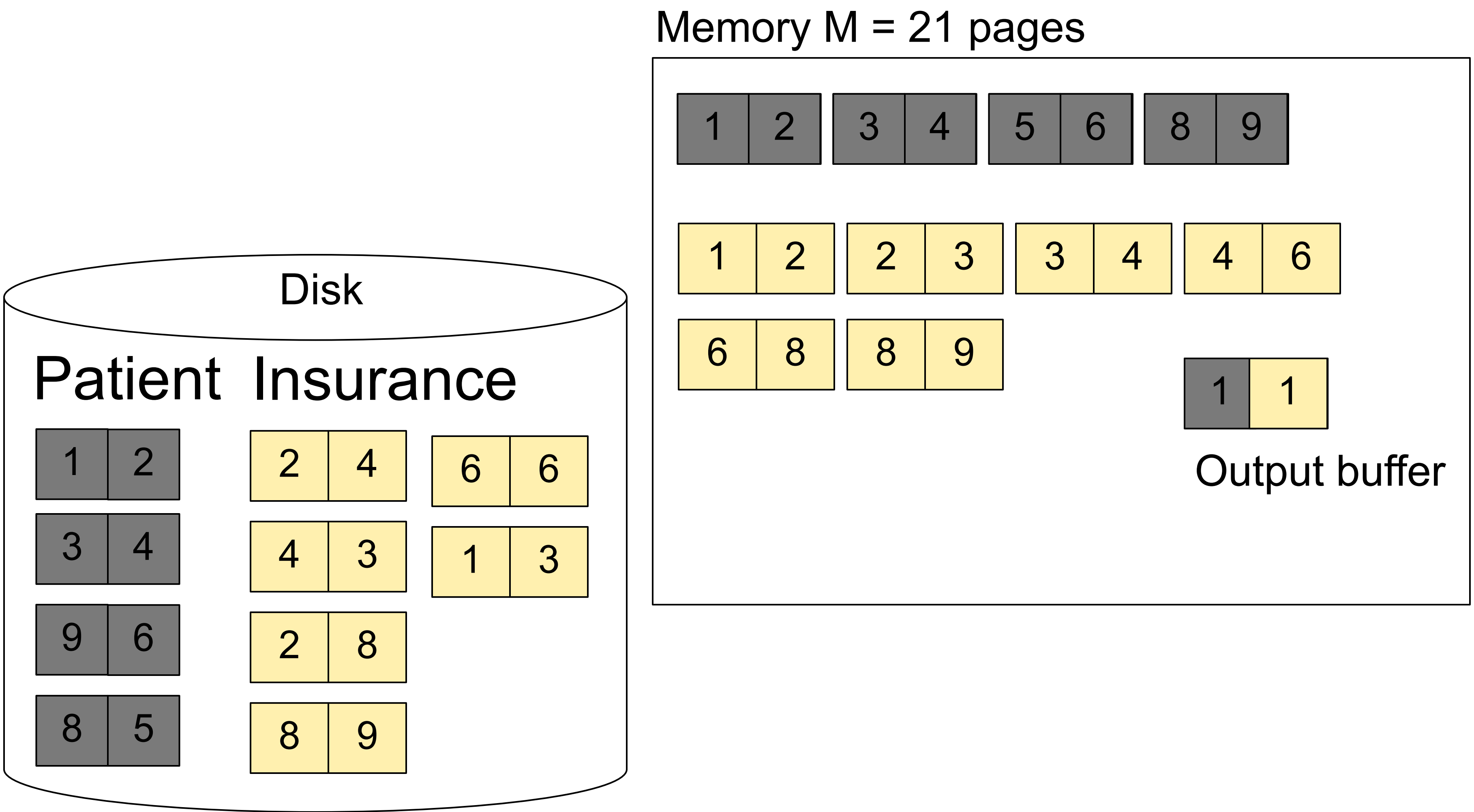
SORT-MERGE JOIN EXAMPLE

Step 2: Scan Insurance and sort in memory



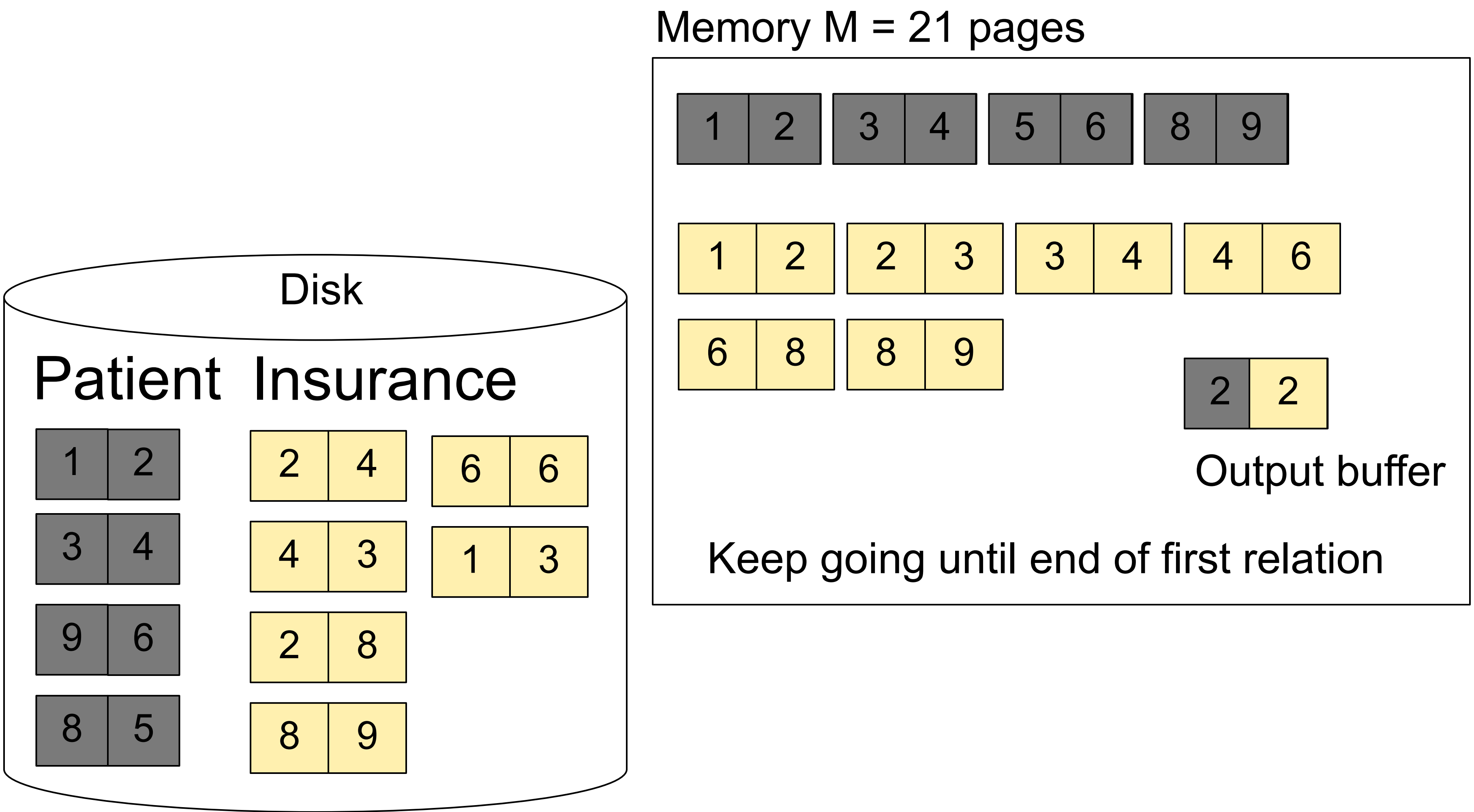
SORT-MERGE JOIN EXAMPLE

Step 3: Merge Patient and Insurance



SORT-MERGE JOIN EXAMPLE

Step 3: Merge Patient and Insurance



INDEX NESTED LOOP JOIN

R ⋈ S

Assume S has an index on the join attribute

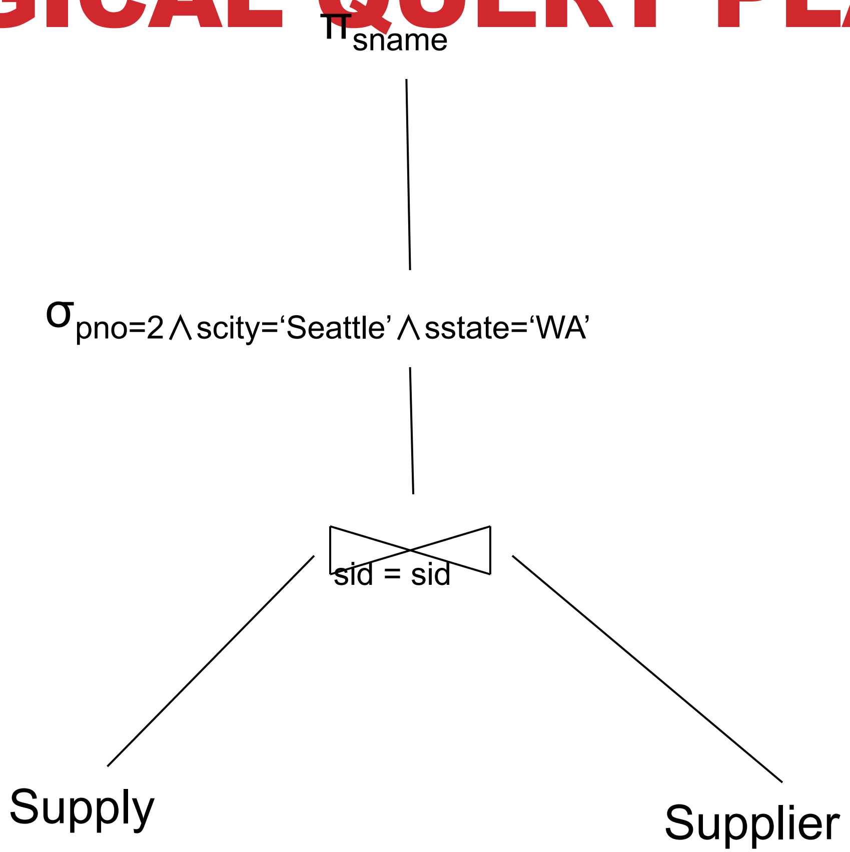
Iterate over R, for each tuple fetch corresponding tuple(s) from S

Cost:

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

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

LOGICAL QUERY PLAN 1




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

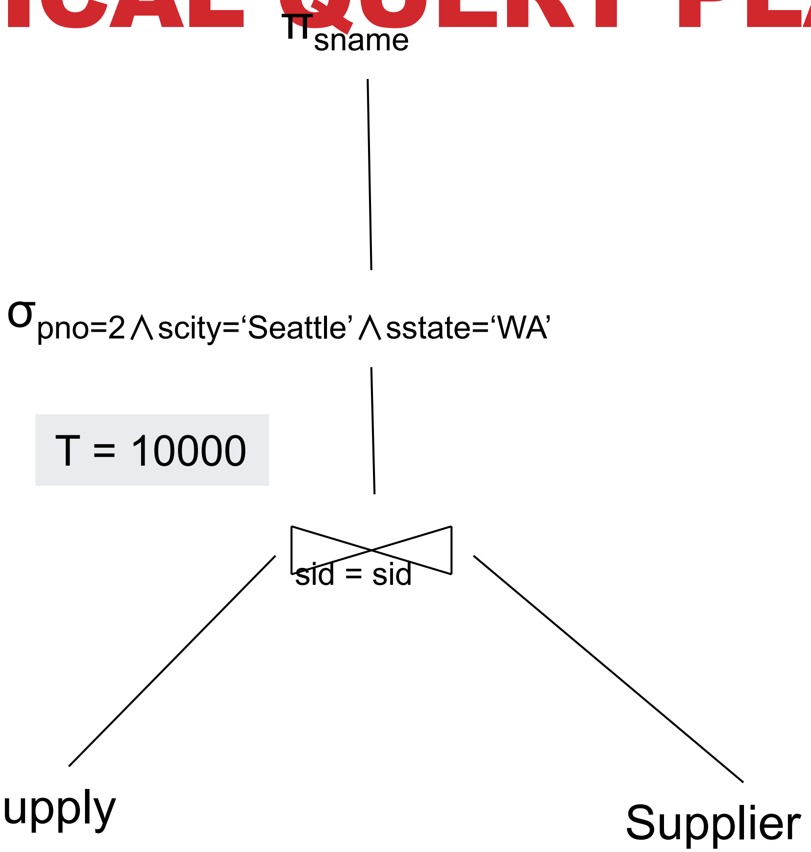
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

LOGICAL QUERY PLAN 1




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

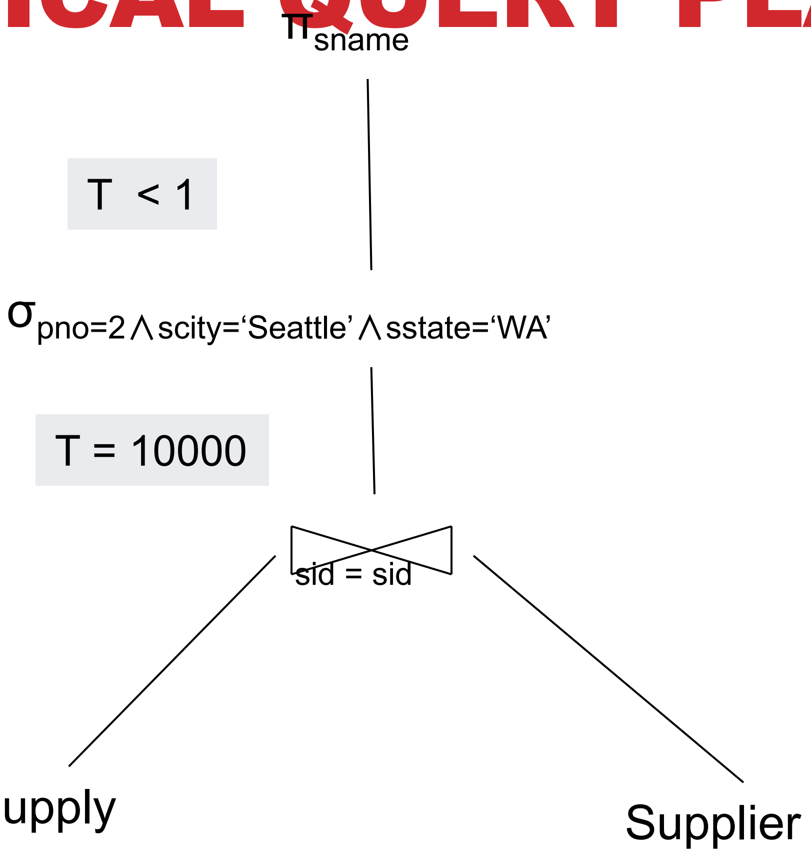
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

LOGICAL QUERY PLAN 1




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

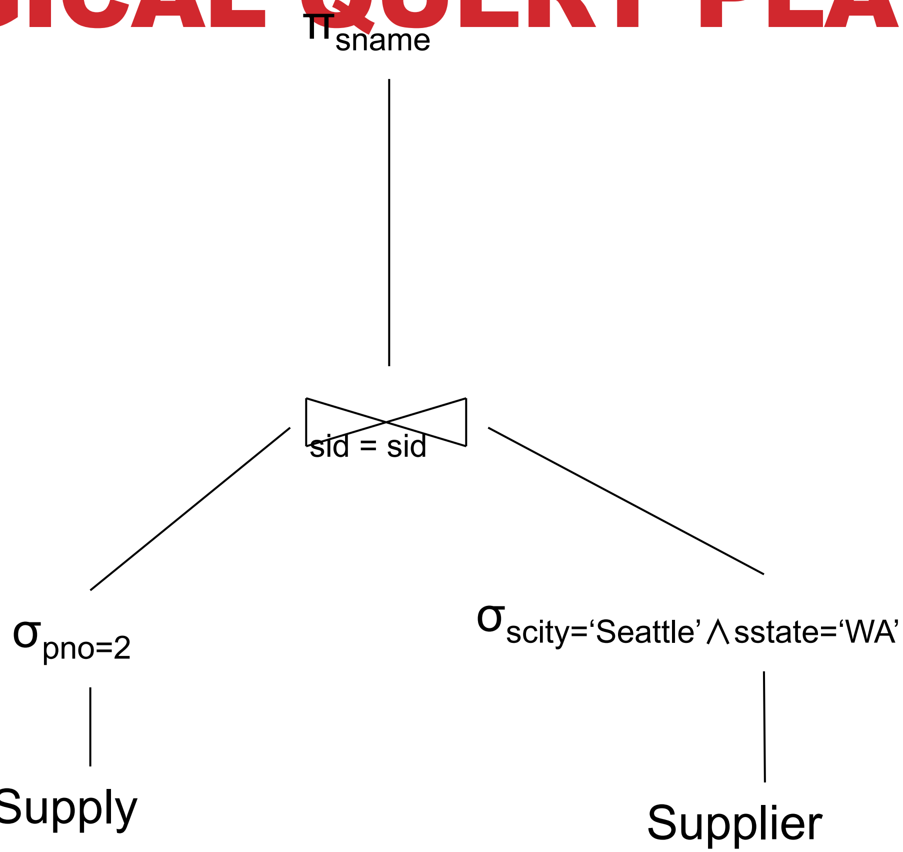
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

LOGICAL QUERY PLAN 2




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

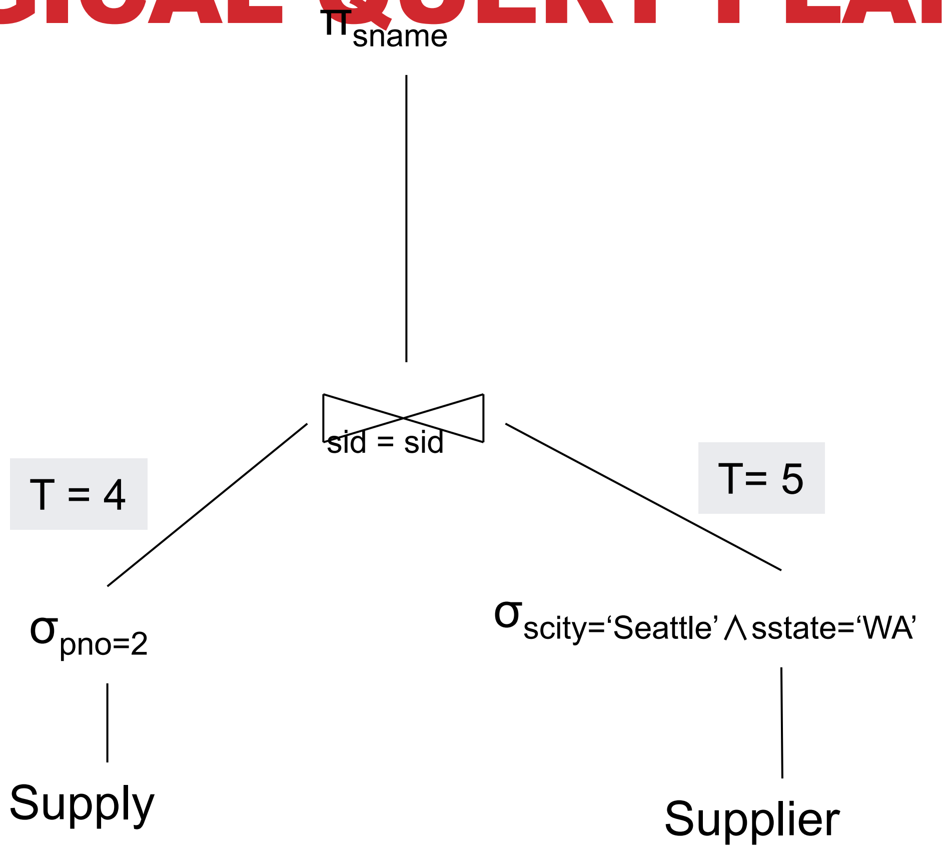
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

LOGICAL QUERY PLAN 2




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

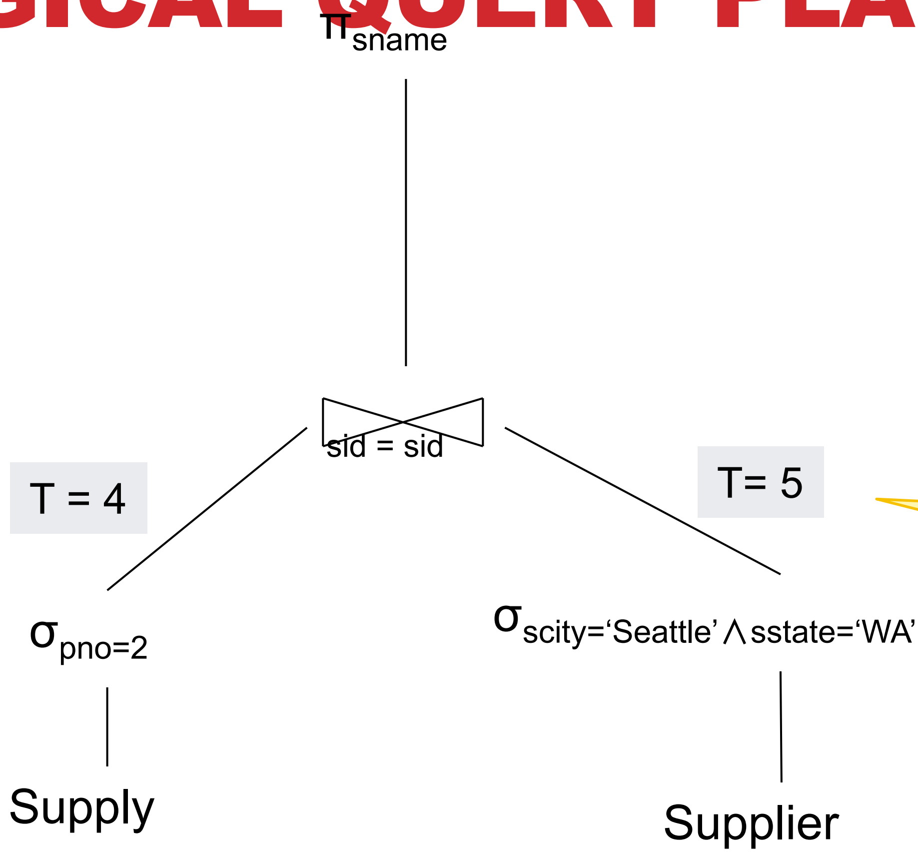
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

LOGICAL QUERY PLAN 2




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

Very wrong!
Why?

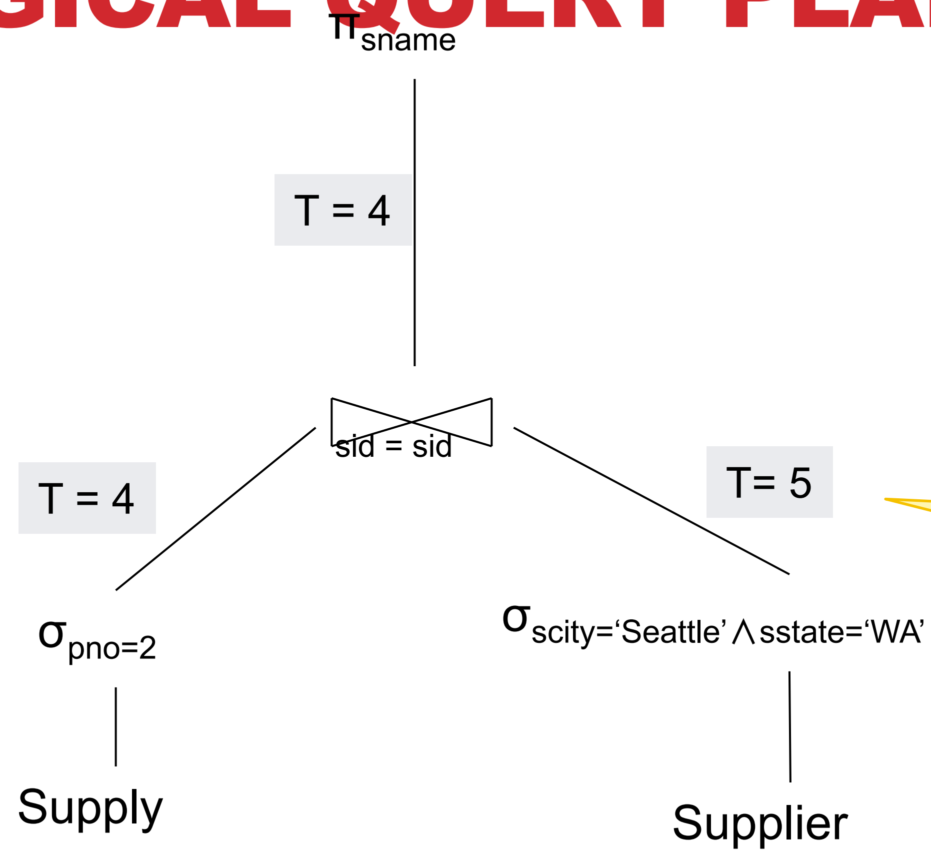
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

LOGICAL QUERY PLAN 2




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

Very wrong!
Why?

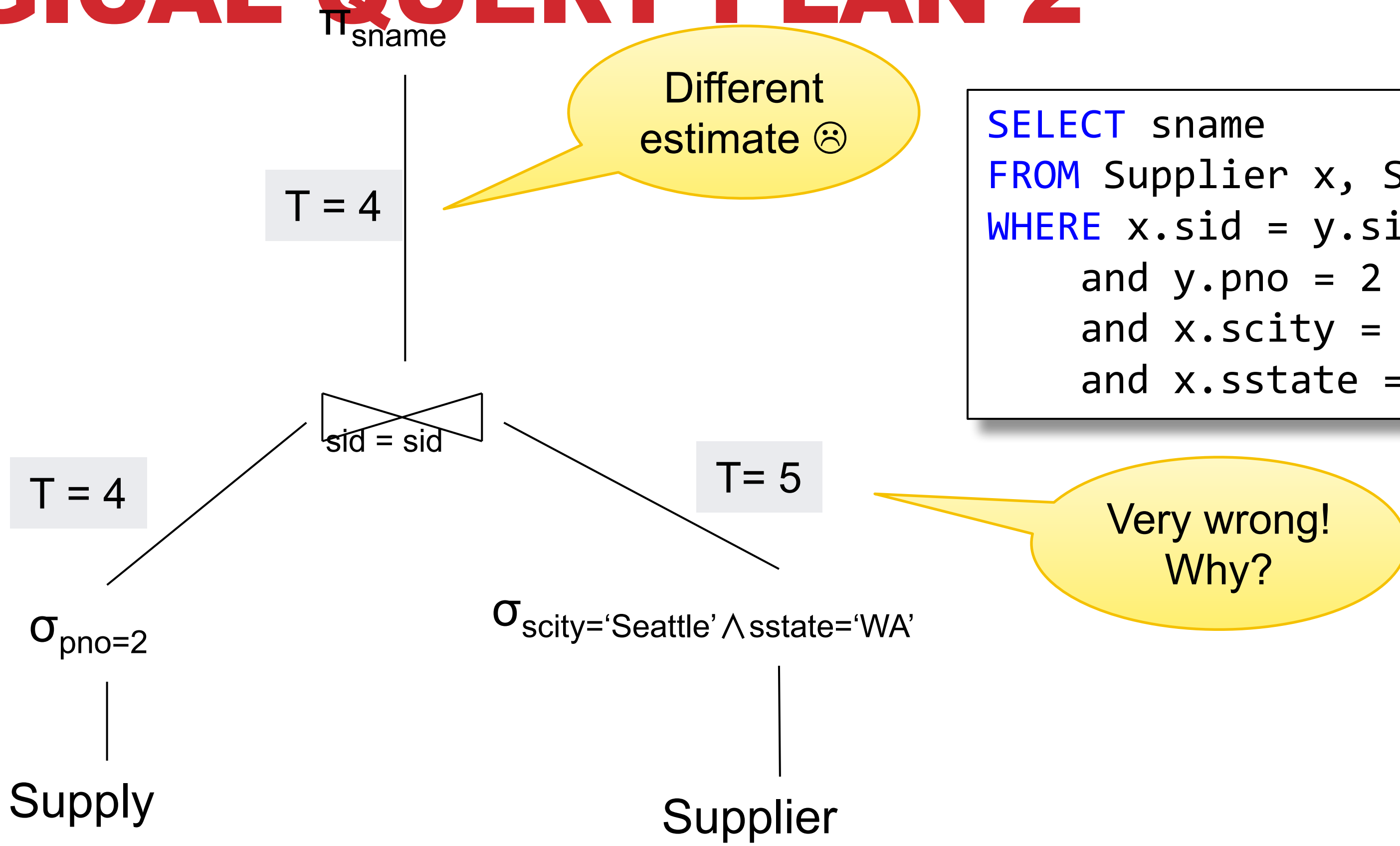
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

LOGICAL QUERY PLAN 2




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

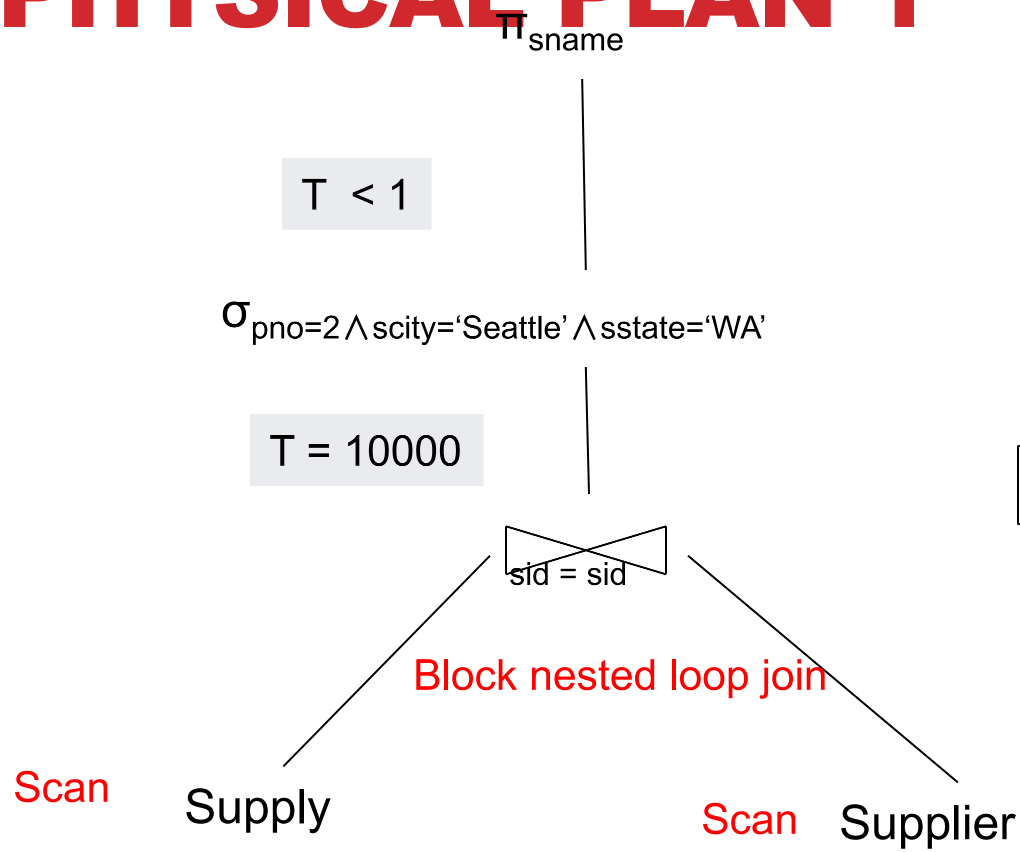
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 1



Total cost:

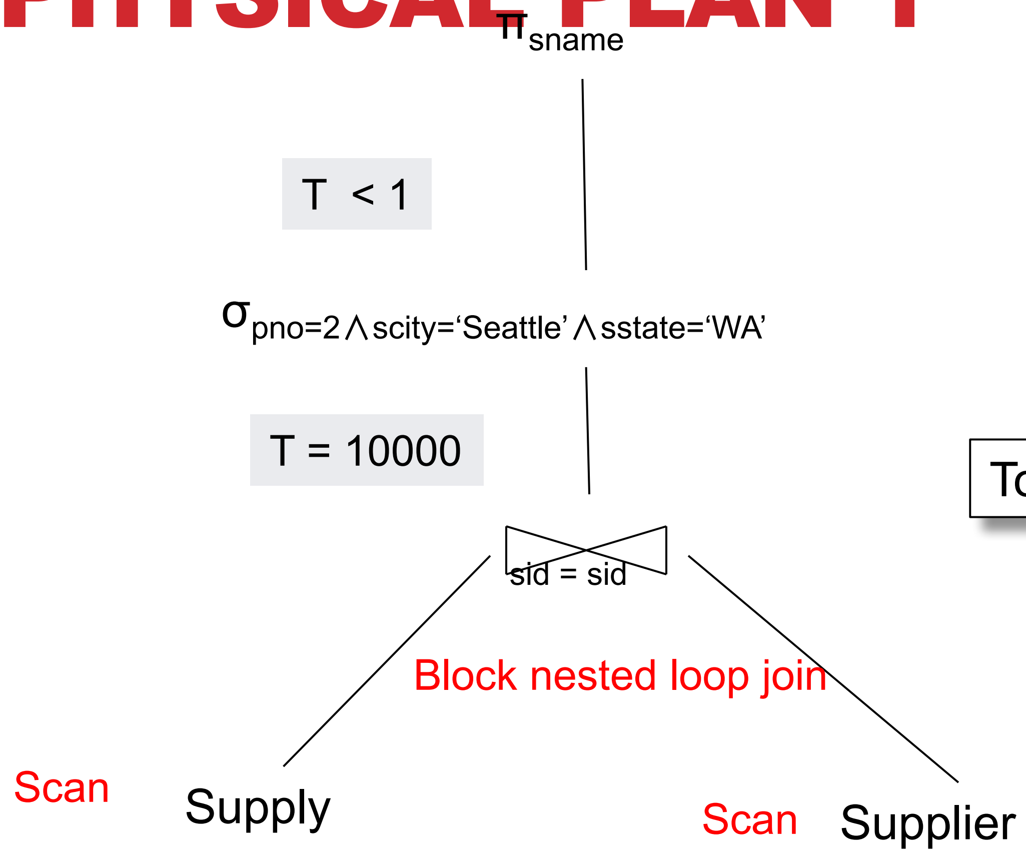
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 1



Total cost: 100+100*100/10 = 1100

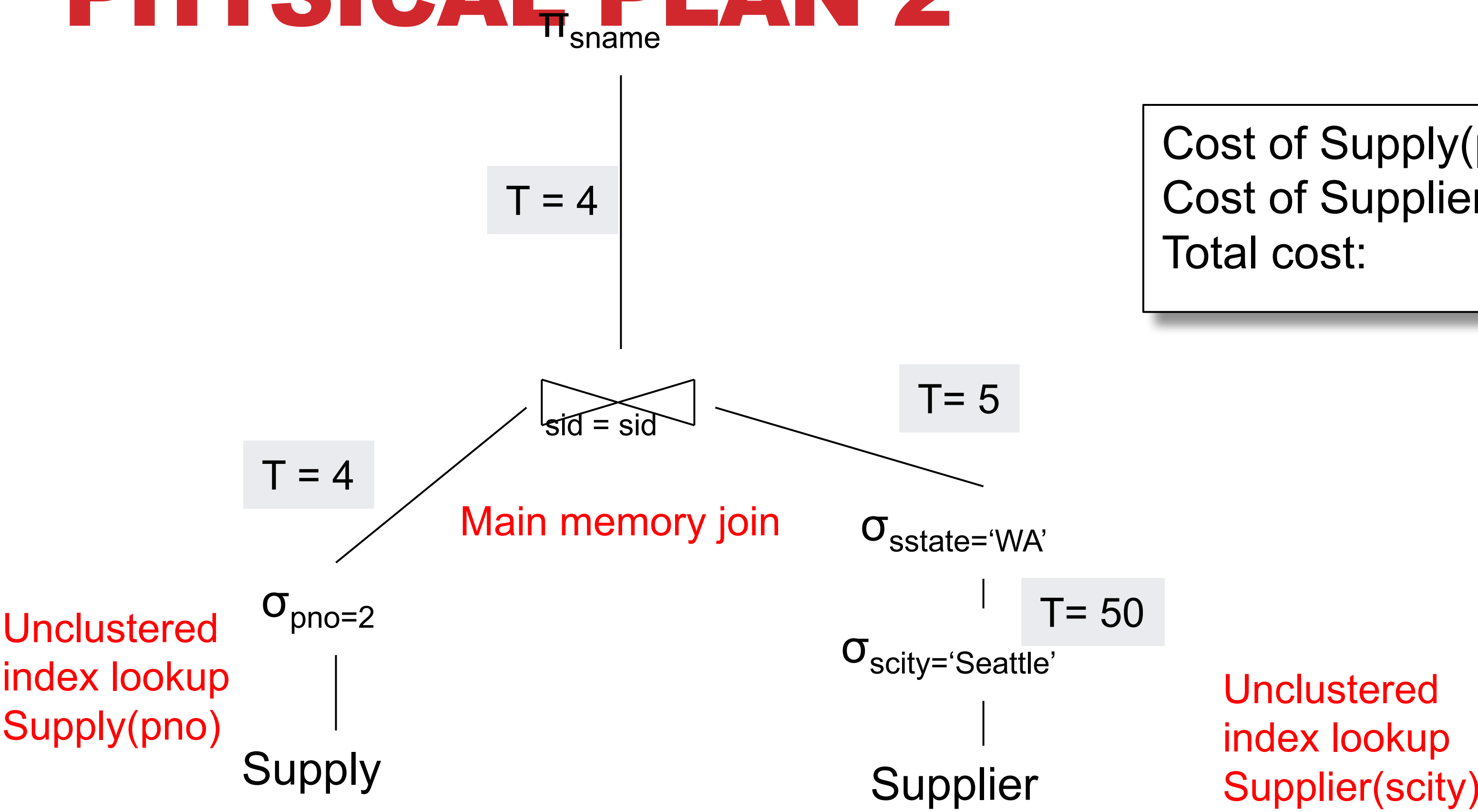
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 2



Cost of Supply(pno) =
Cost of Supplier(scity) =
Total cost:

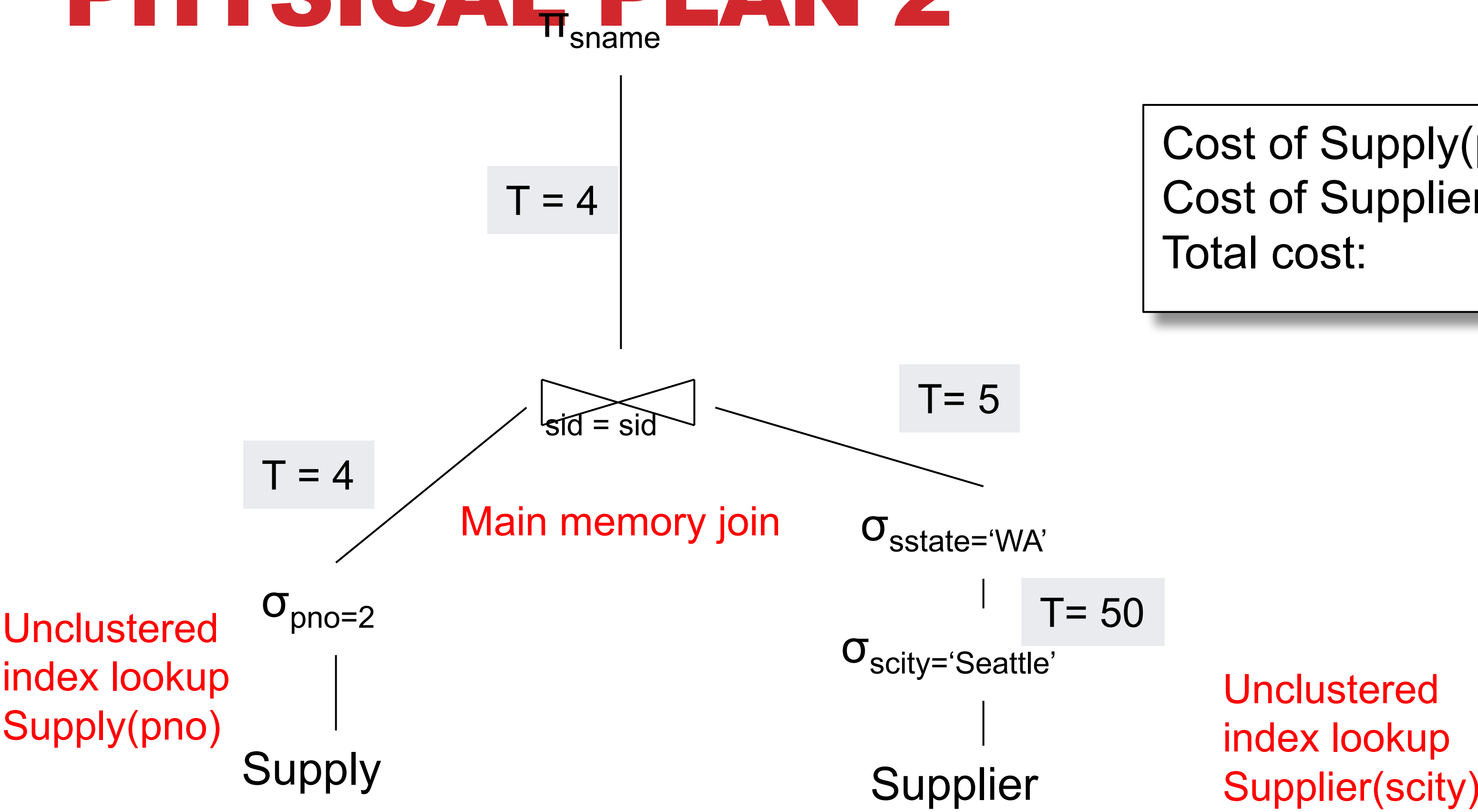
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 2



Cost of Supply(pno) = 4
Cost of Supplier(scity) =
Total cost:

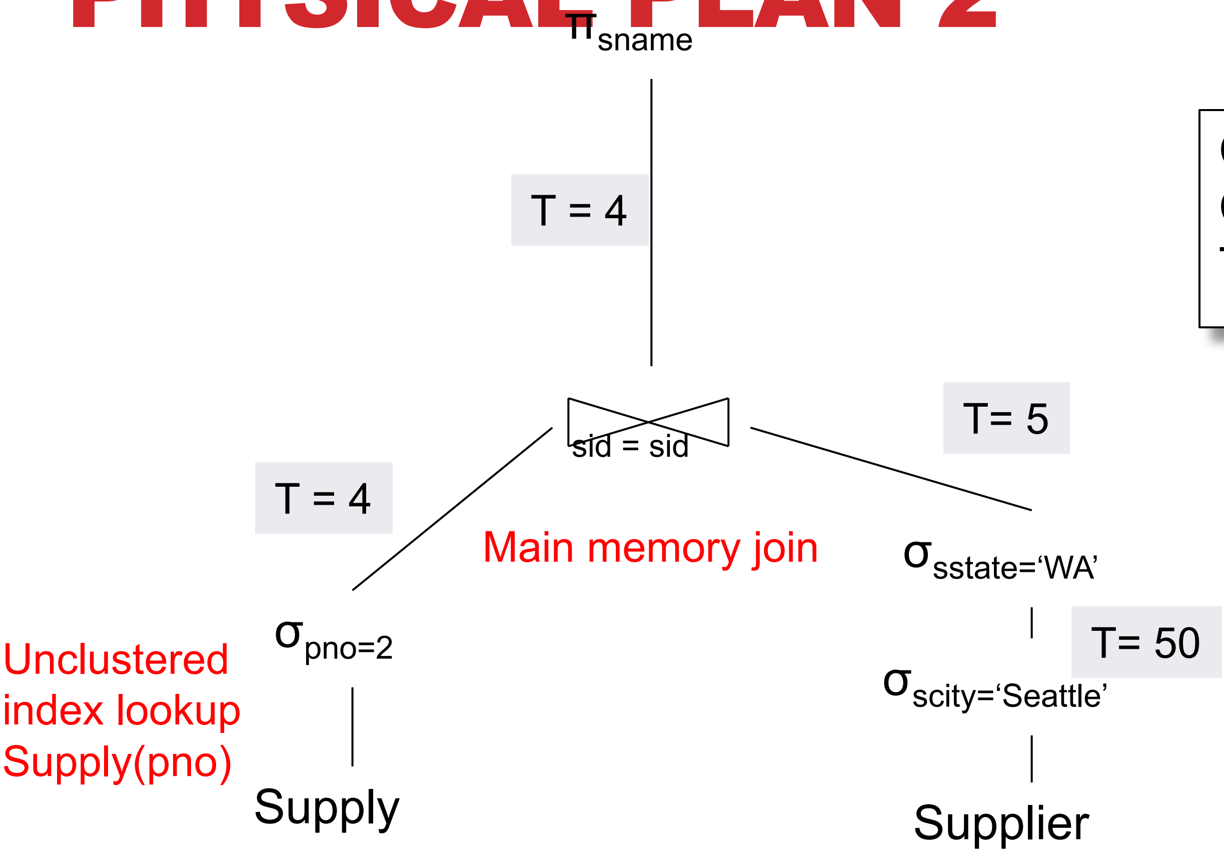
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 2



Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54

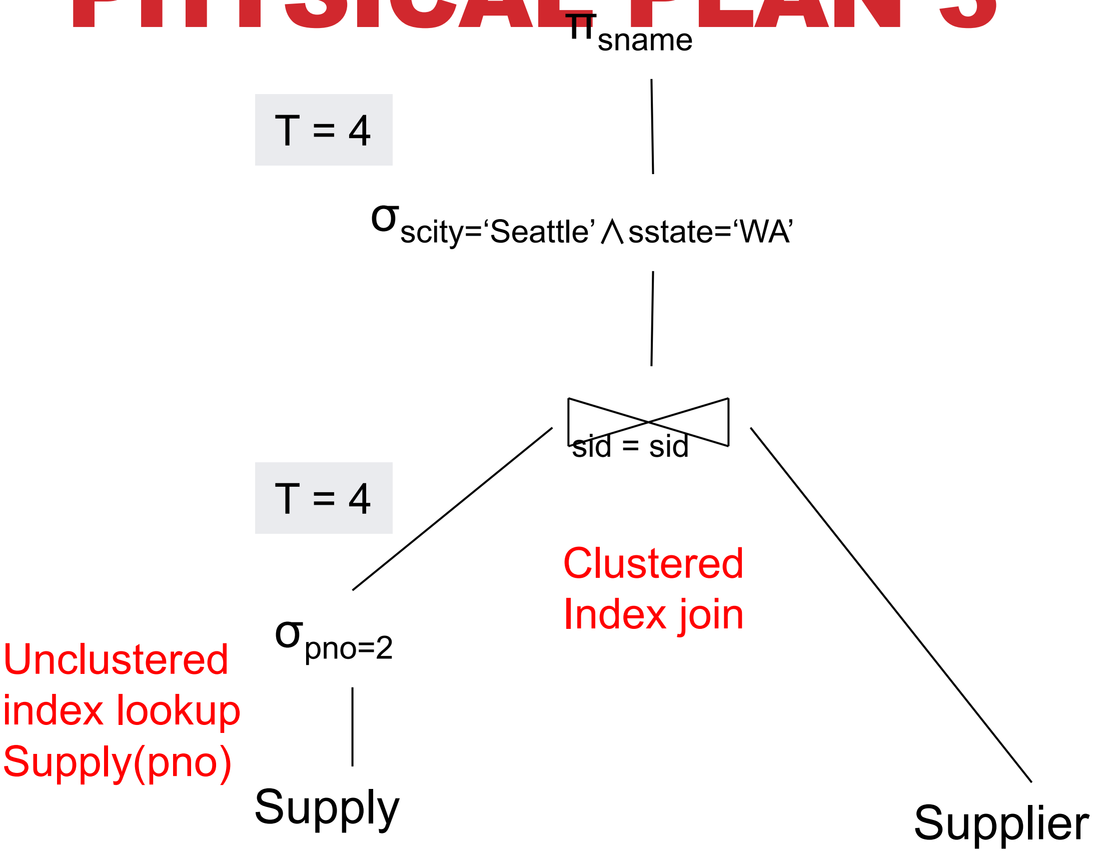
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 3



Cost of Supply(pno) =
Cost of Index join =
Total cost:

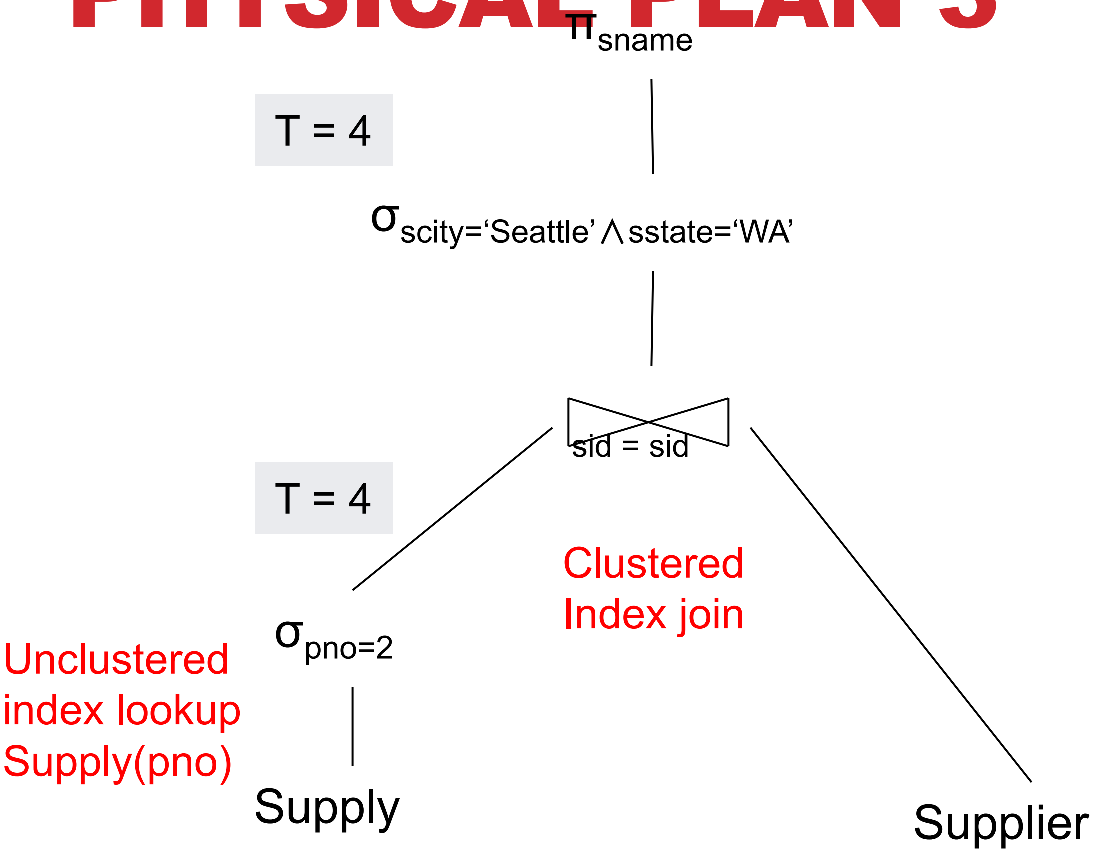
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 3



Cost of Supply(pno) = 4
Cost of Index join =
Total cost:

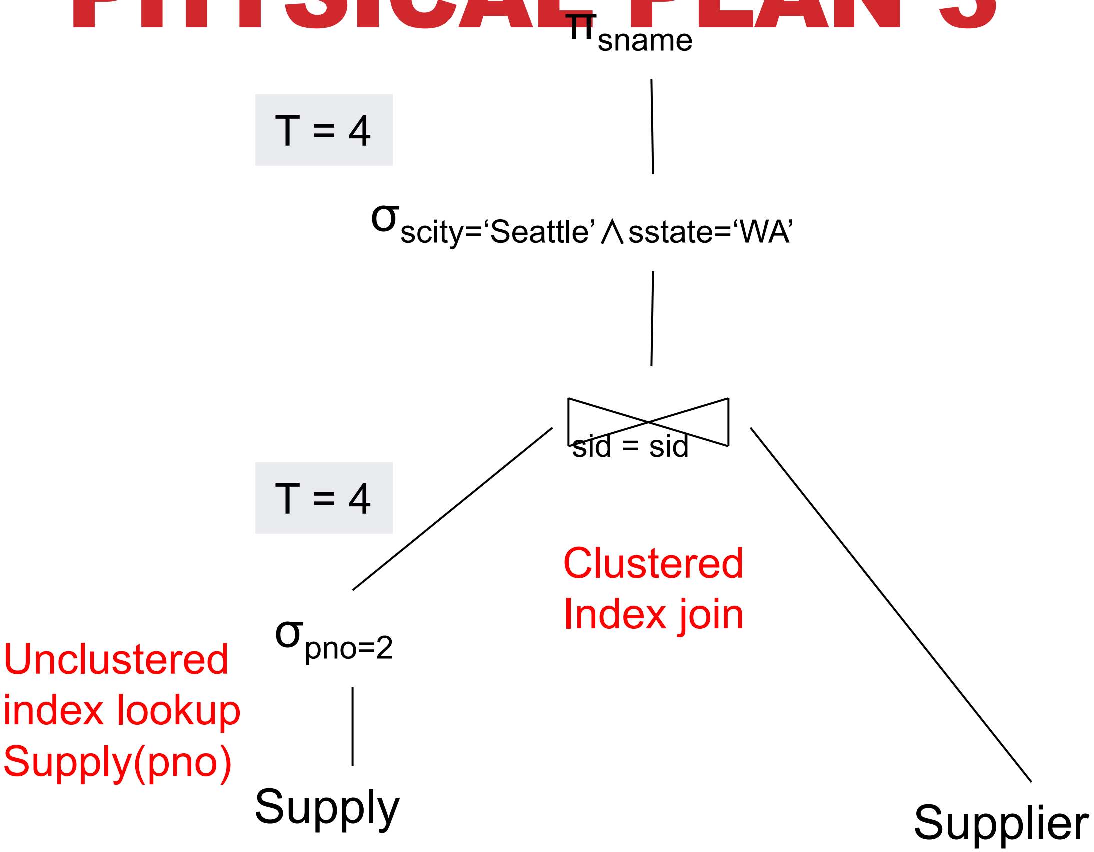
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

PHYSICAL PLAN 3



Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

QUERY OPTIMIZER SUMMARY

Input: A logical query plan

Output: A good physical query plan

Basic query optimization algorithm

- Enumerate alternative plans (logical and physical)
- Compute estimated cost of each plan
- Choose plan with lowest cost

This is called cost-based optimization