CSE 344

JULY 20TH

COST ESTIMATION

ADMINISTRIVIA

- **Midterm in one week**
  - covers the material through today
    - Relational data model & queries
      - SQL, RA, Datalog
    - NoSQL data model and queries
    - Query optimization
  - more details next week...

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

TWO TYPICAL KINDS OF QUERIES

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

- Point queries
  - (or equijoins)
- Hash table or B+ tree

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

- Range queries
- B+ tree only

BASIC INDEX SELECTION GUIDELINES

Consider queries in workload in order of importance

Consider relations accessed by query

- No benefit to indexing other relations

Look at WHERE clause and JOIN .. ON for possible search key

Try to choose indexes that speed-up multiple queries

TO CLUSTER OR NOT

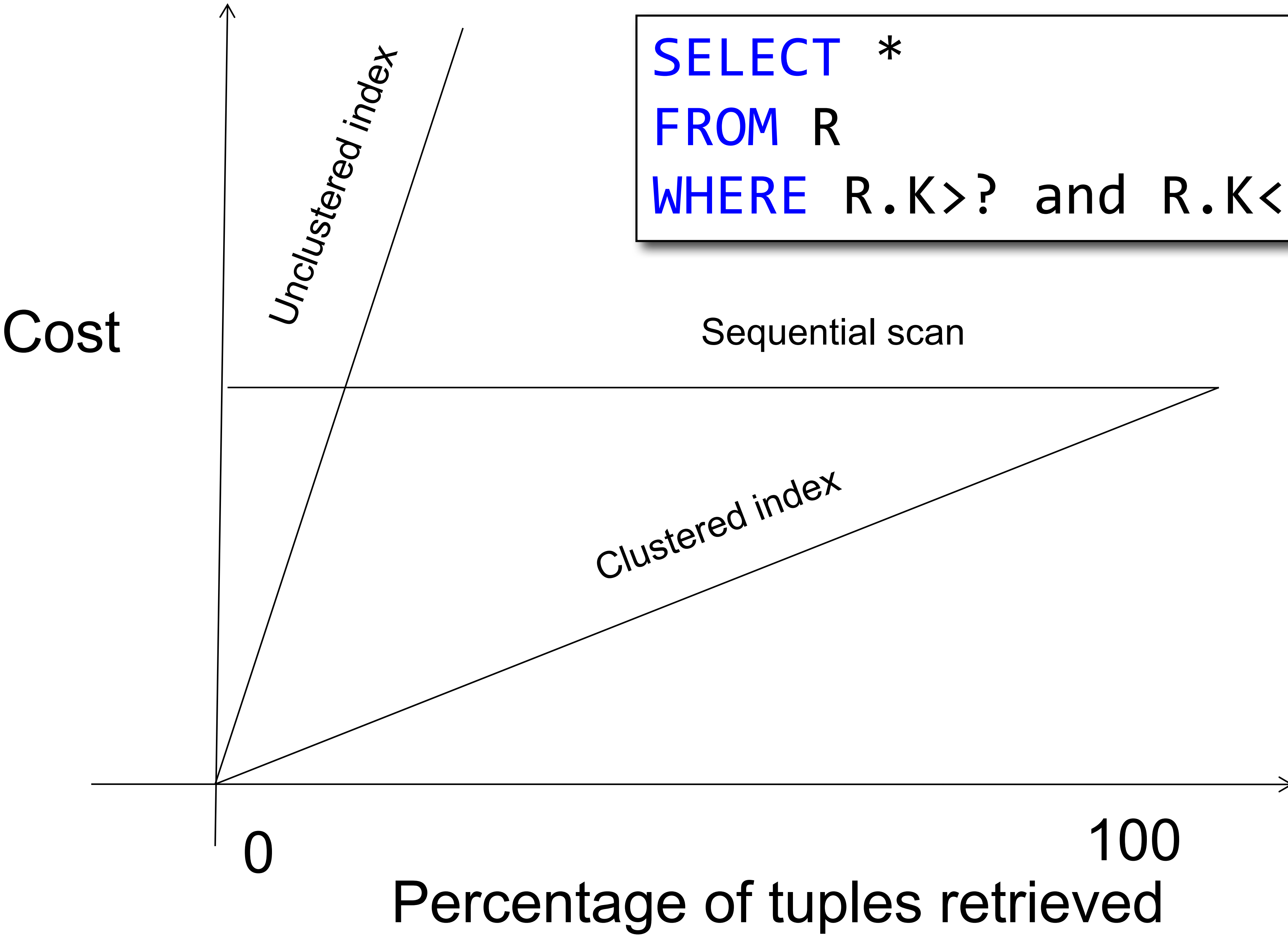
Range queries benefit mostly from clustering

Point indexes on keys do *not* need to be clustered

- will read 1 block whether the index is clustered or not

More generally, cost depends on percent of tuples returned...

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



COST ESTIMATION

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

- How each operator is implemented
- The cost of each operator

Let's start with the basics...

COST PARAMETERS

Cost = Disk I/O + Network I/O + Memory I/O + CPU

- **Disk I/O ≫ Network I/O ≫ Memory I/O ≫ CPU**
- We will focus on Disk I/O for now
  - if the query plan involves disk I/O, that is likely to dominate cost
  - for parallel, in-memory DBs, network costs usually dominate

COST PARAMETERS

Cost = Disk I/O + Network I/O + Memory I/O + CPU

- We will focus on Disk I/O for now

Parameters (a.k.a. statistics):

- **B(R)** = # of blocks for relation R
- **T(R)** = # of tuples in relation R
- **V(R, A)** = # of distinct values of attribute A appearing in relation R

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 = Disk I/O + Network I/O + Memory I/O + CPU

- We will focus on Disk I/O for now

Parameters (a.k.a. statistics):

- **B(R)** = # of blocks for relation R
- **T(R)** = # of tuples in relation R
- **V(R, A)** = # of distinct values of attribute A appearing in relation R

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

- (above information and *more*)
- allows DB to estimate things like “selectivity”...

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

$\sigma_{a=v}$
|
R

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

$\sigma_{a=v}$

|

R

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

$\sigma_{a=v}$

|

R

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

$\sigma_{a=v}$
|
R

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

$\sigma_{a=v}$
|
R

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

$\sigma_{a=v}$

R

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

Key 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 $\bowtie$ S

- Scan S into hash table in main memory
- Then scan R and join

Cost: B(R) + B(S)

Which relation to build the hash table on?

- (either can be done)

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’

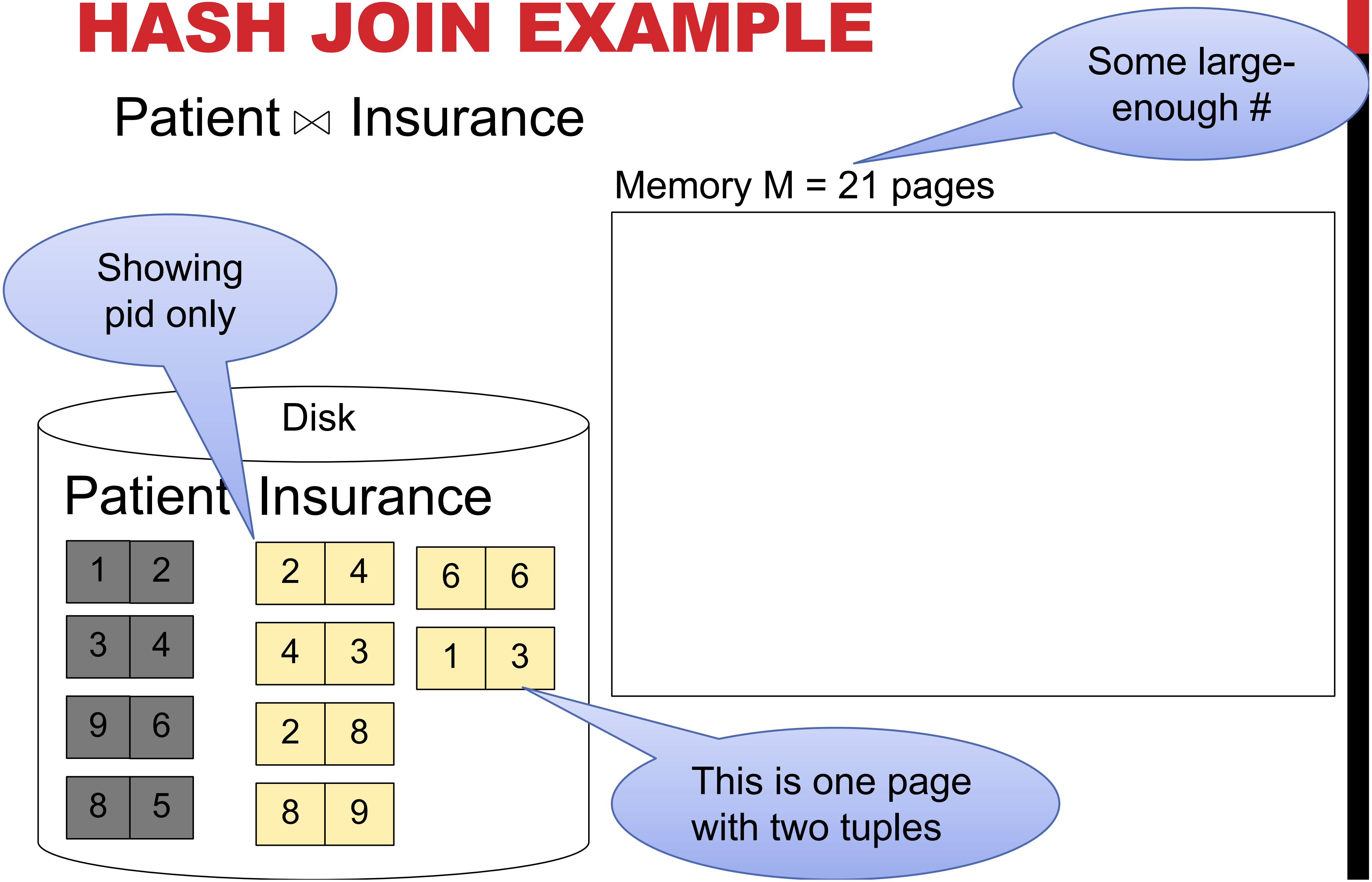
Insurance

2	‘Blue’	123
4	‘Prem’	432

4	‘Prem’	343
3	‘GrpH’	554

HASH JOIN EXAMPLE

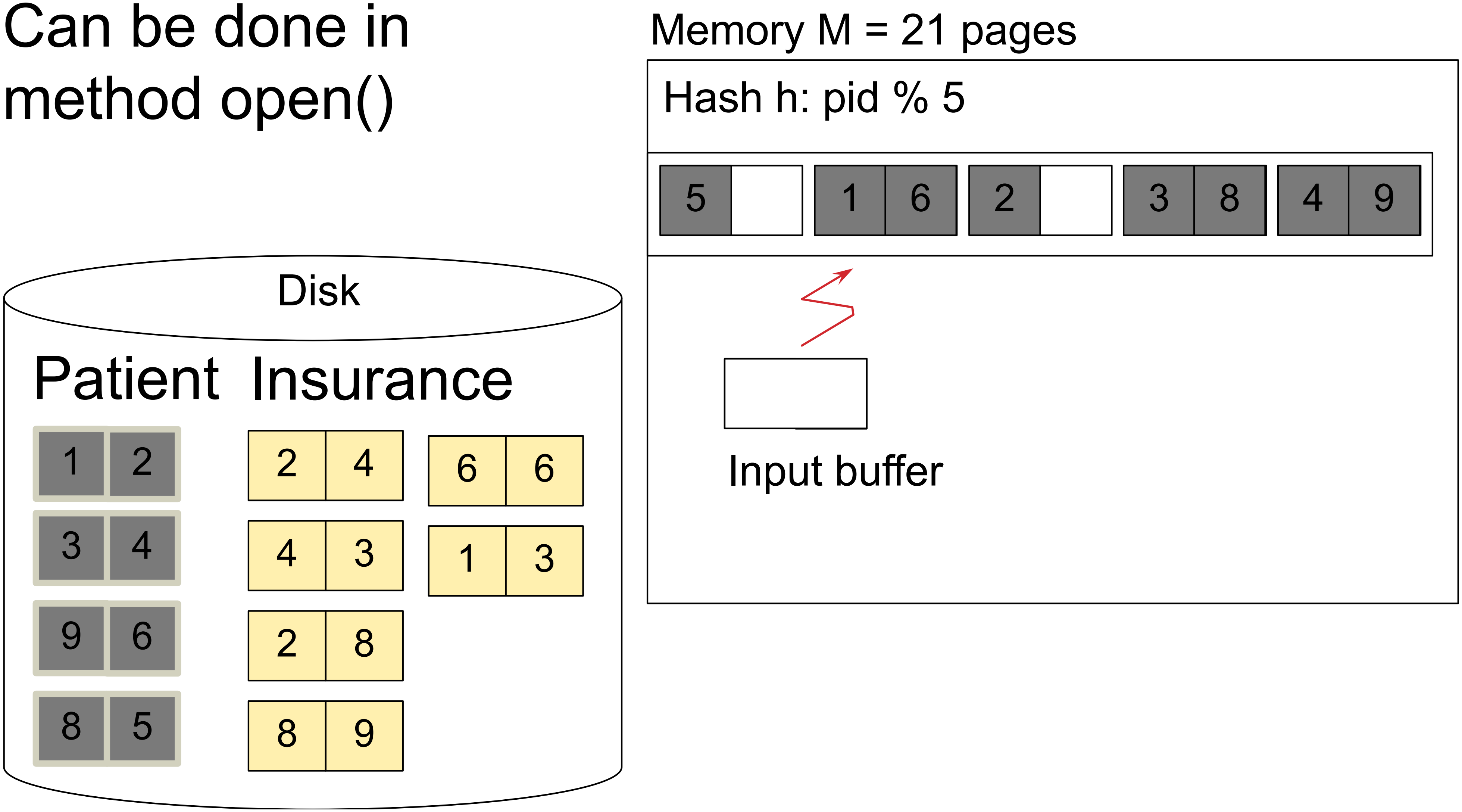
Patient $\bowtie$ Insurance



HASH JOIN EXAMPLE

Step 1: Scan Patient and build hash table in memory

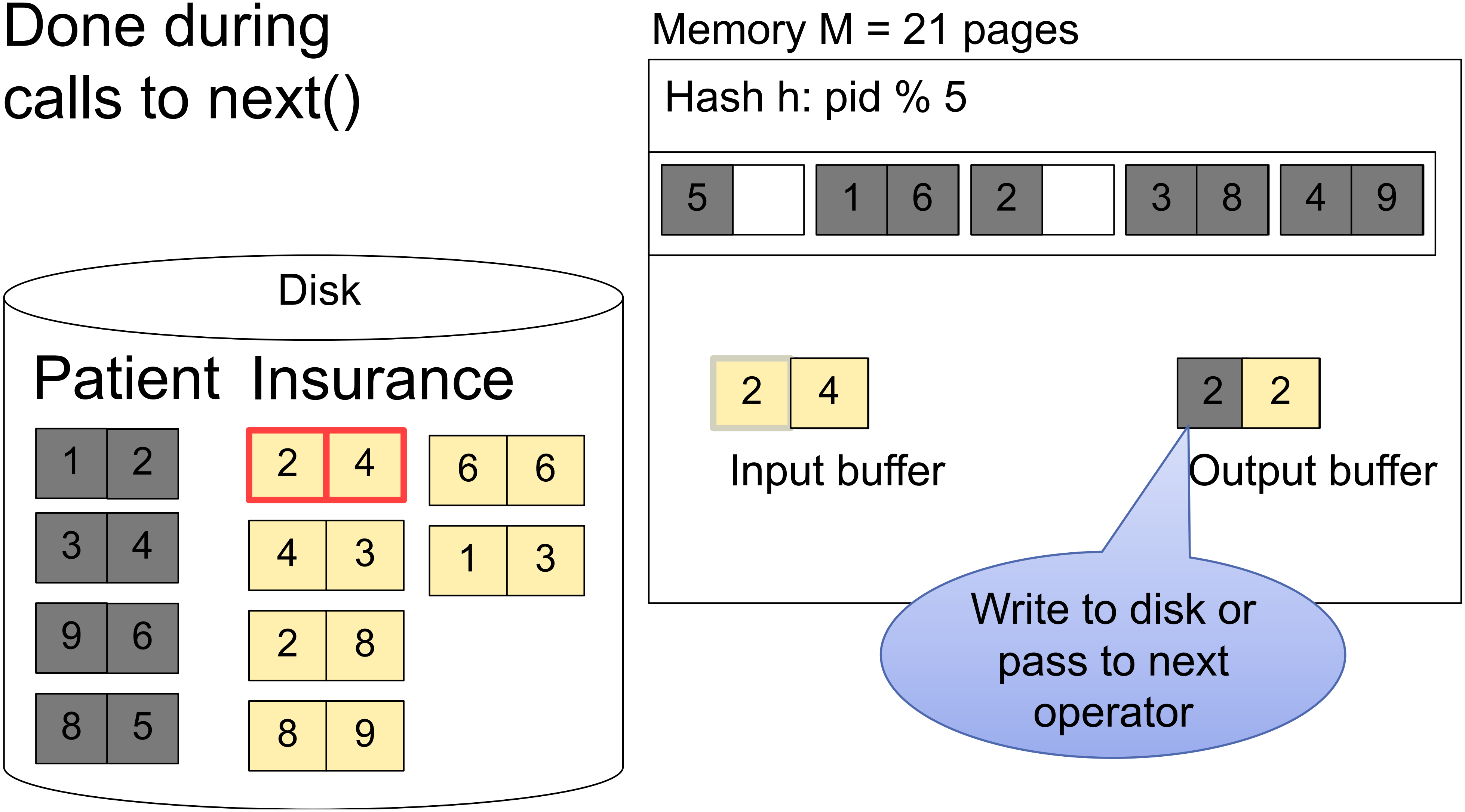
Can be done in method open()



HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table

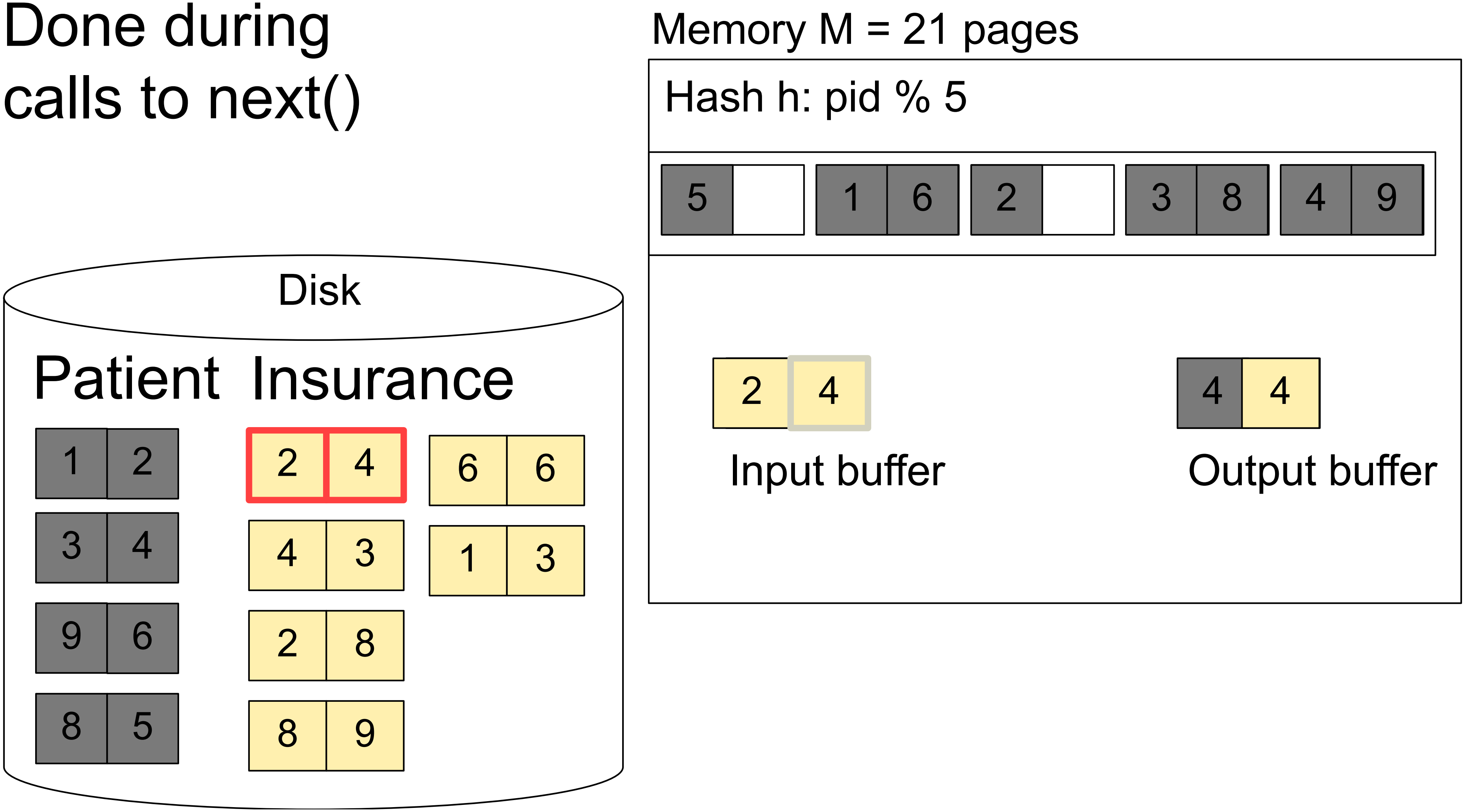
Done during calls to next()



HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

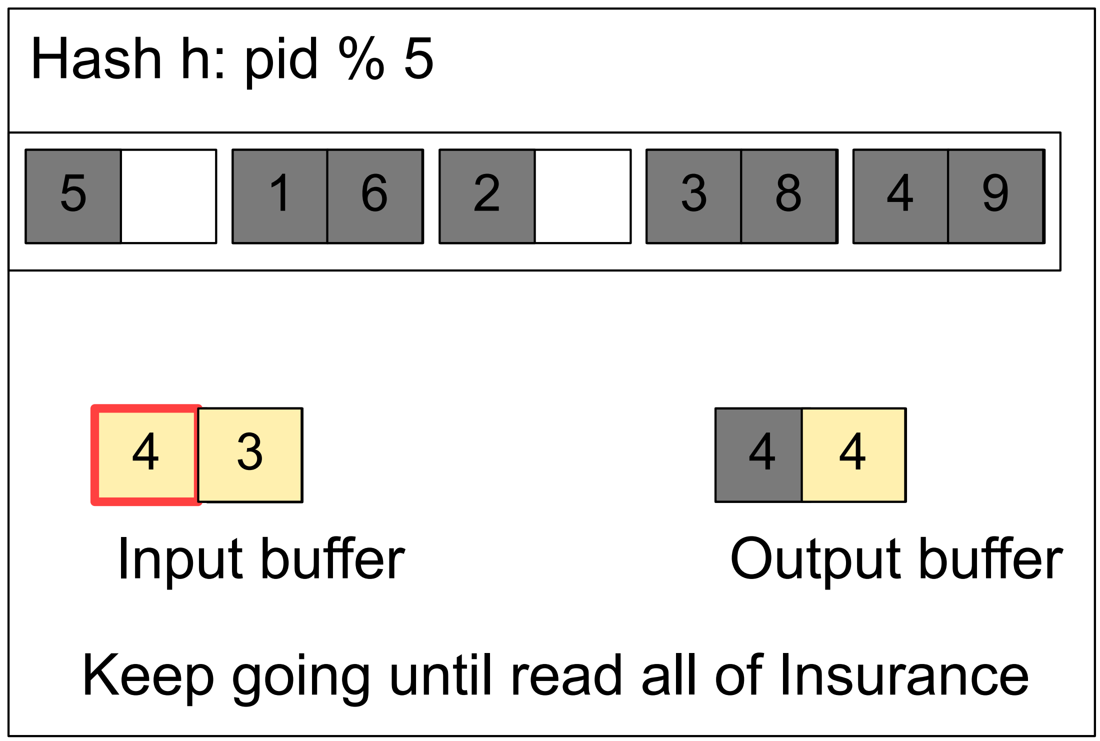


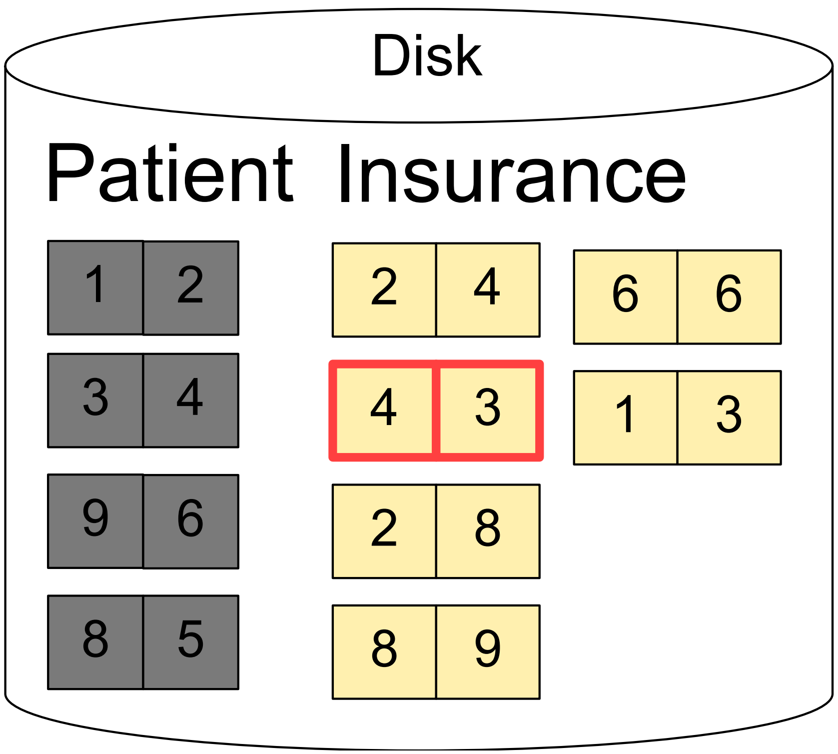
HASH JOIN EXAMPLE

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)
```

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

- multiple-pass since S is read many times
- factor of T(R) is very painful...

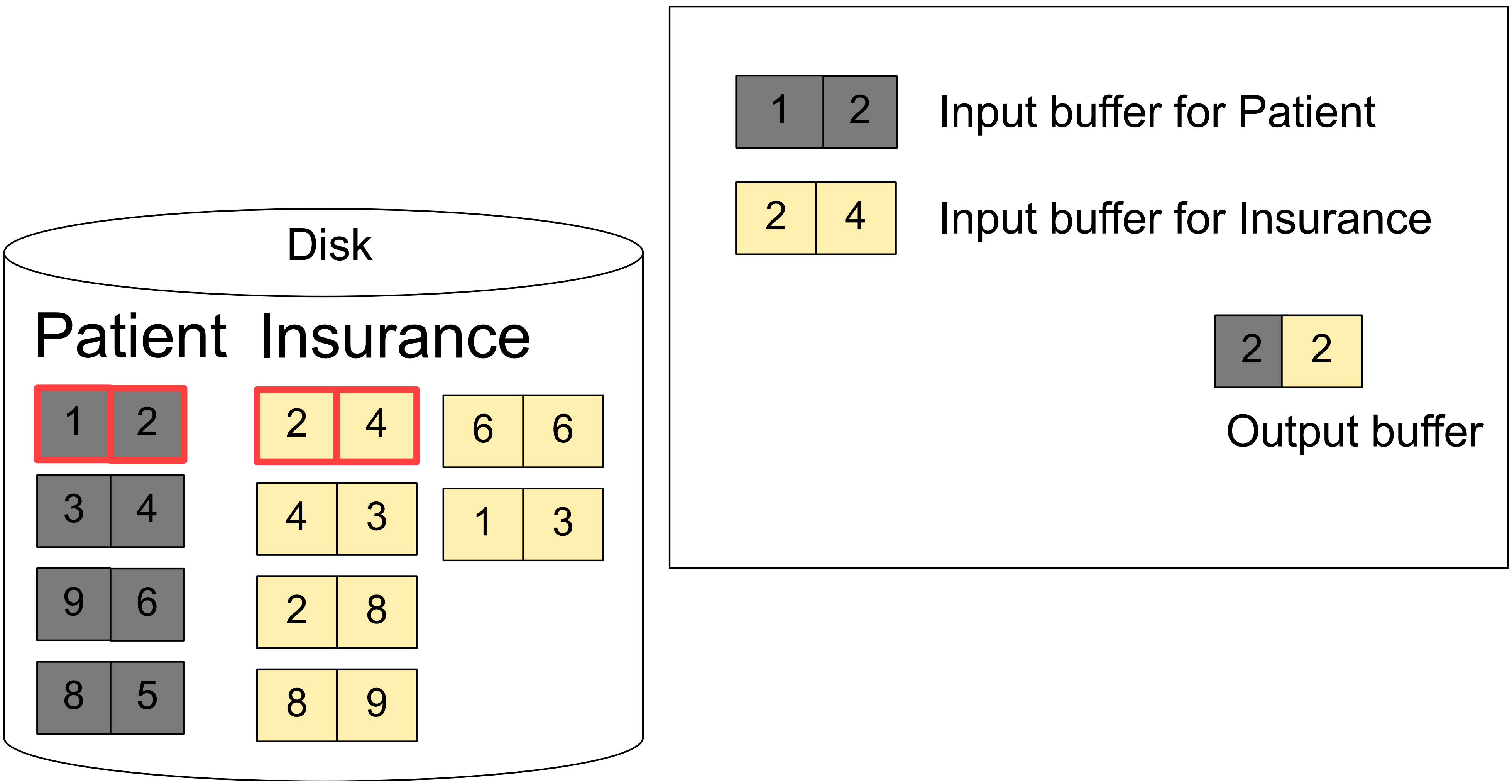
PAGE-AT-A-TIME REFINEMENT

```
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)
```

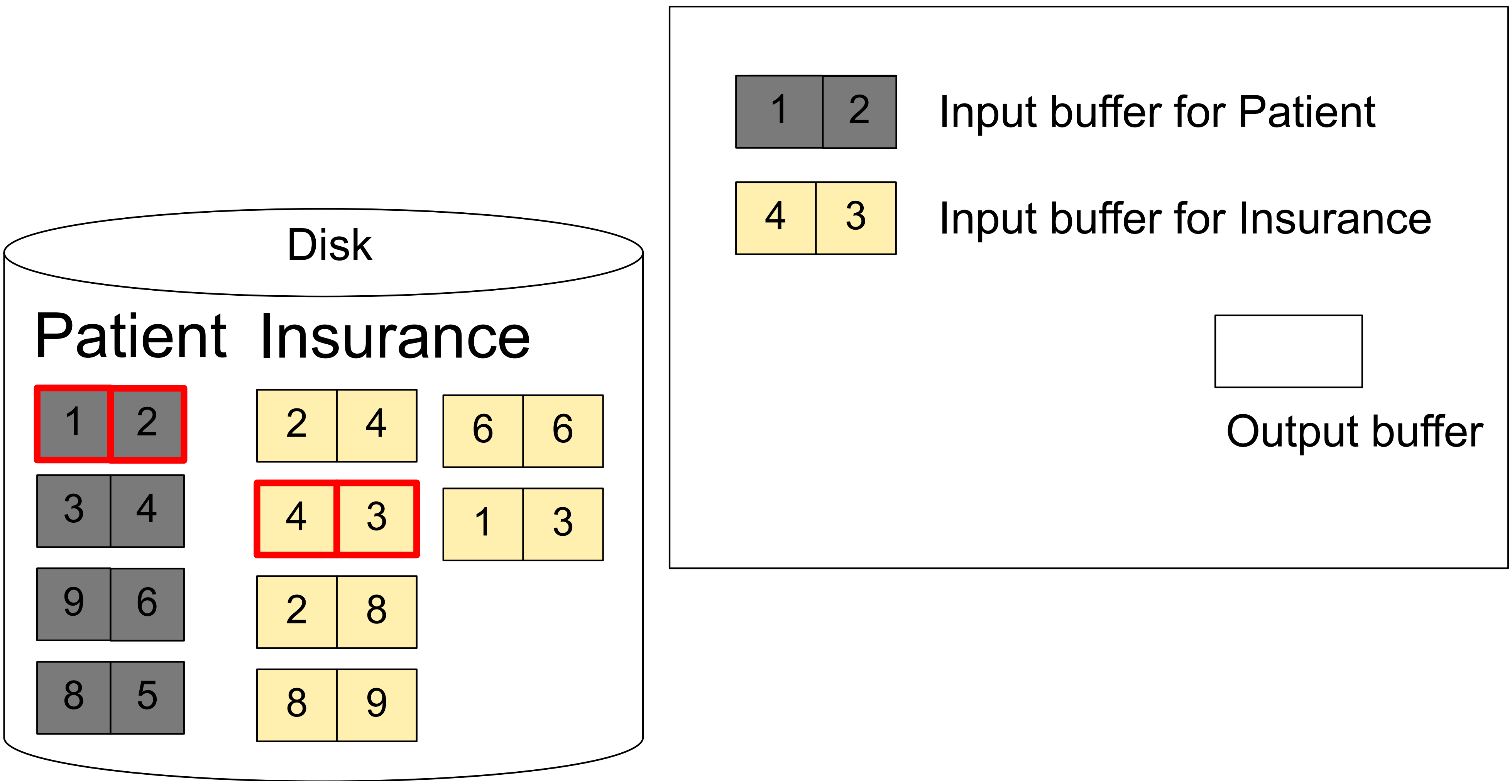
Cost: B(R) + B(R)B(S)

- only outer loops are disk I/O... inner-most loop is “free” (CPU)
- can speed this up even more if more memory is available

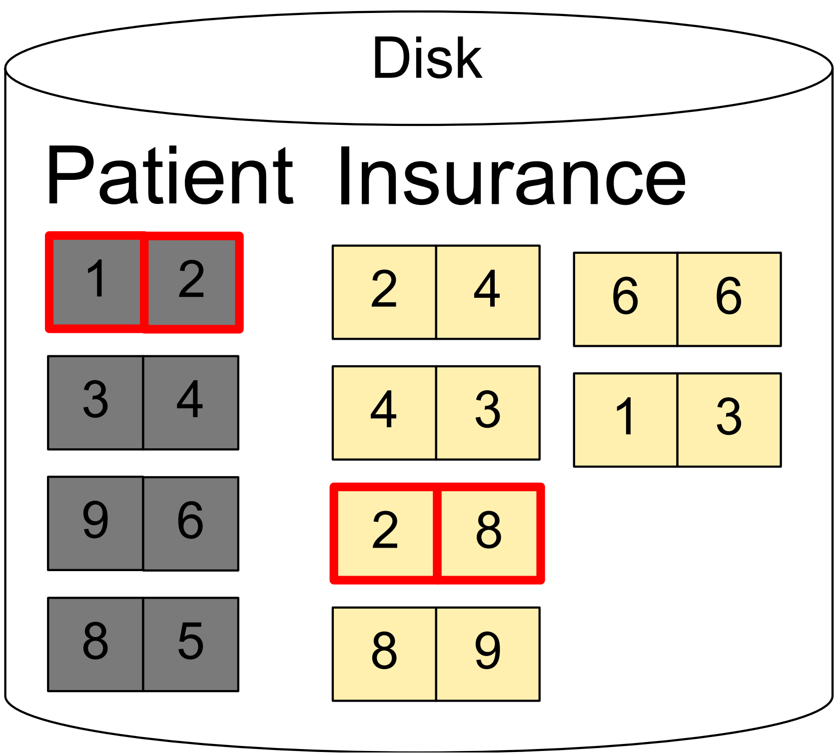
PAGE-AT-A-TIME REFINEMENT

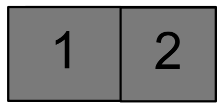


PAGE-AT-A-TIME REFINEMENT

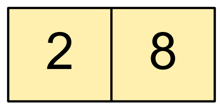


PAGE-AT-A-TIME REFINEMENT



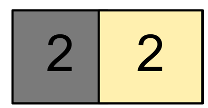


Input buffer for Patient



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)

SORT-MERGE JOIN

Sort-merge join: R ⋈ S

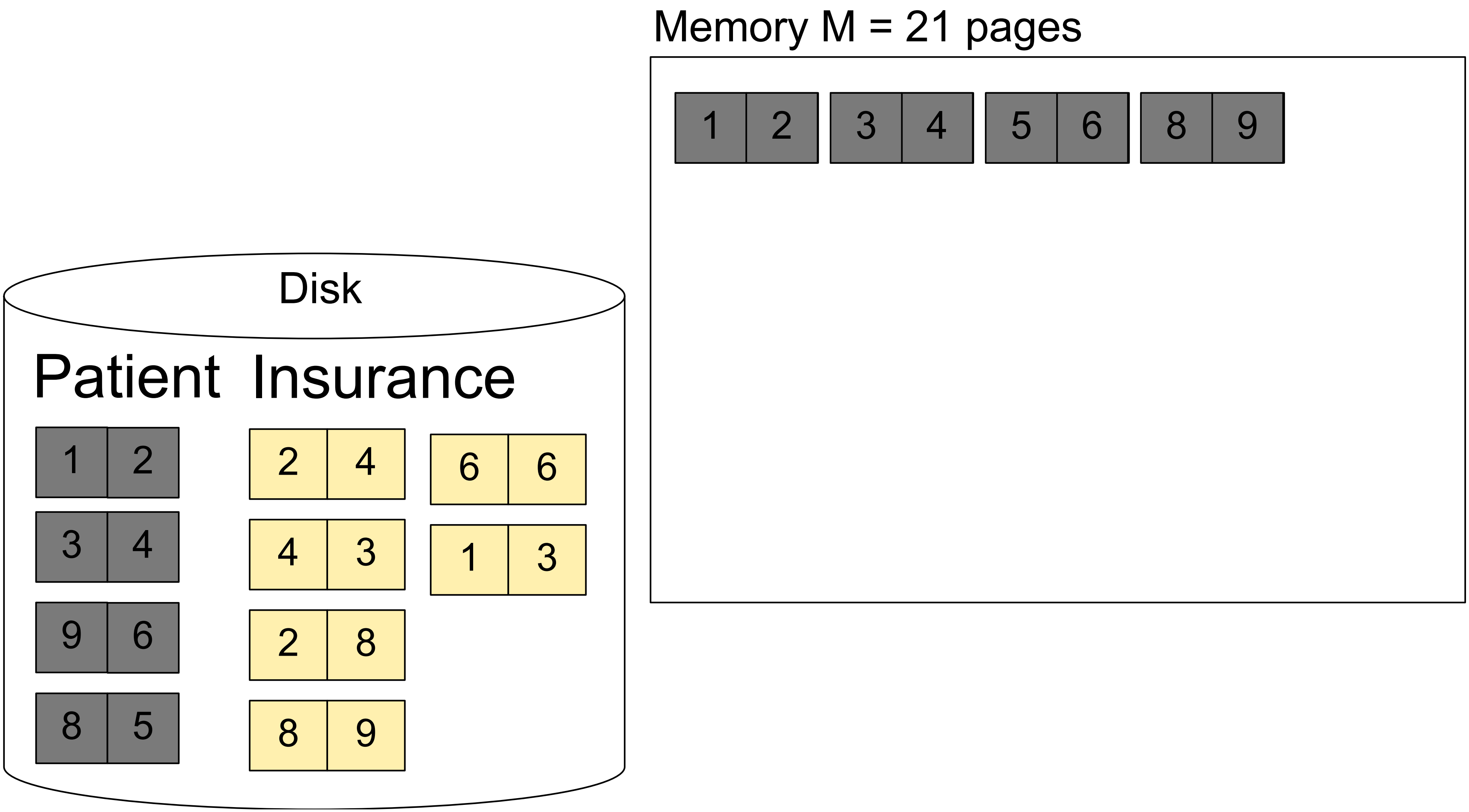
- Scan R and sort (in main memory if possible)
- Scan S and sort (in main memory if possible)
- Merge R and S in one pass

Cost: B(R) + B(S) if sorting done in memory

- only possible if B(R) + B(S) <= M (memory size)
- however, usually no more than 4x this when on disk

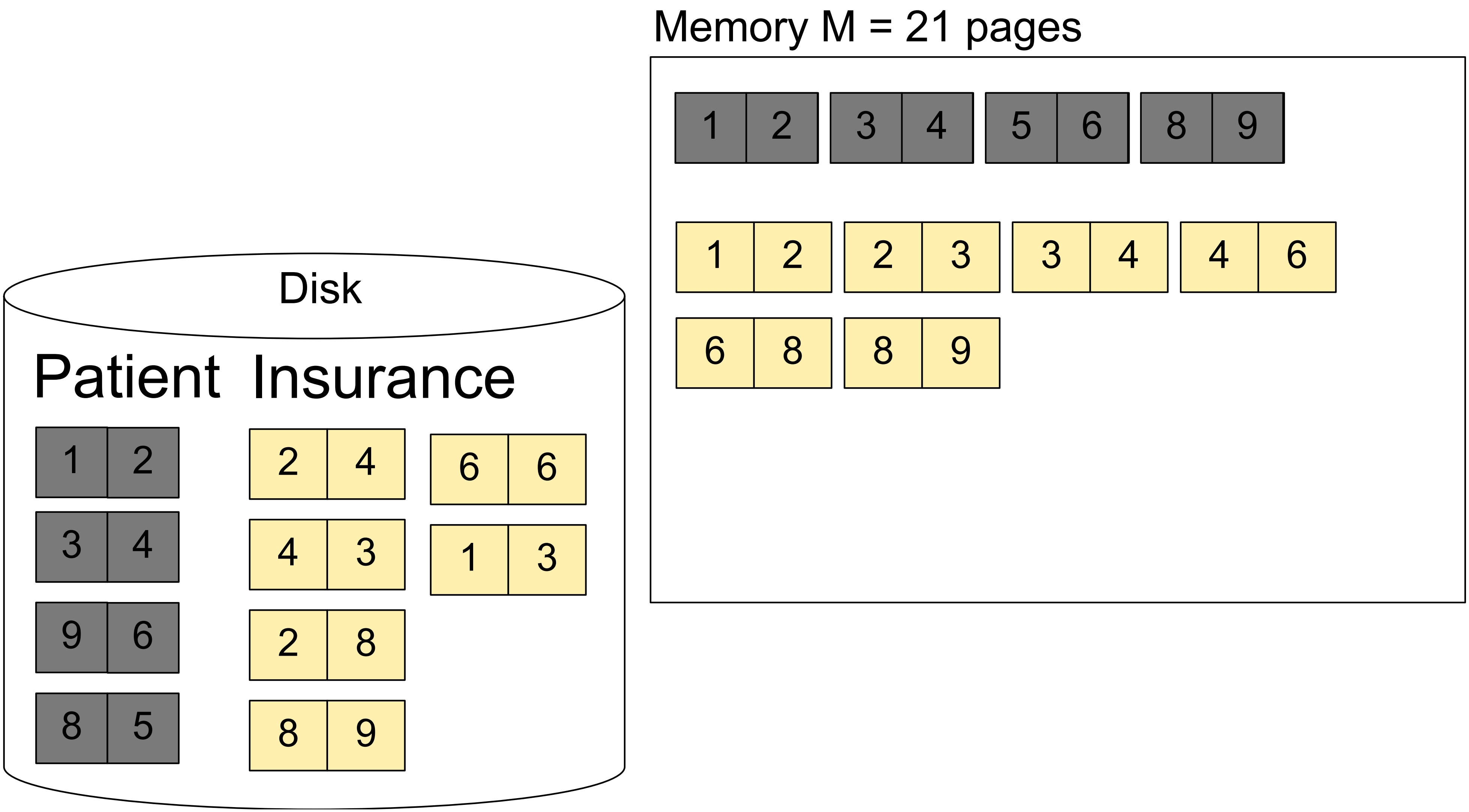
SORT-MERGE JOIN EXAMPLE

Step 1: Scan Patient and sort in memory



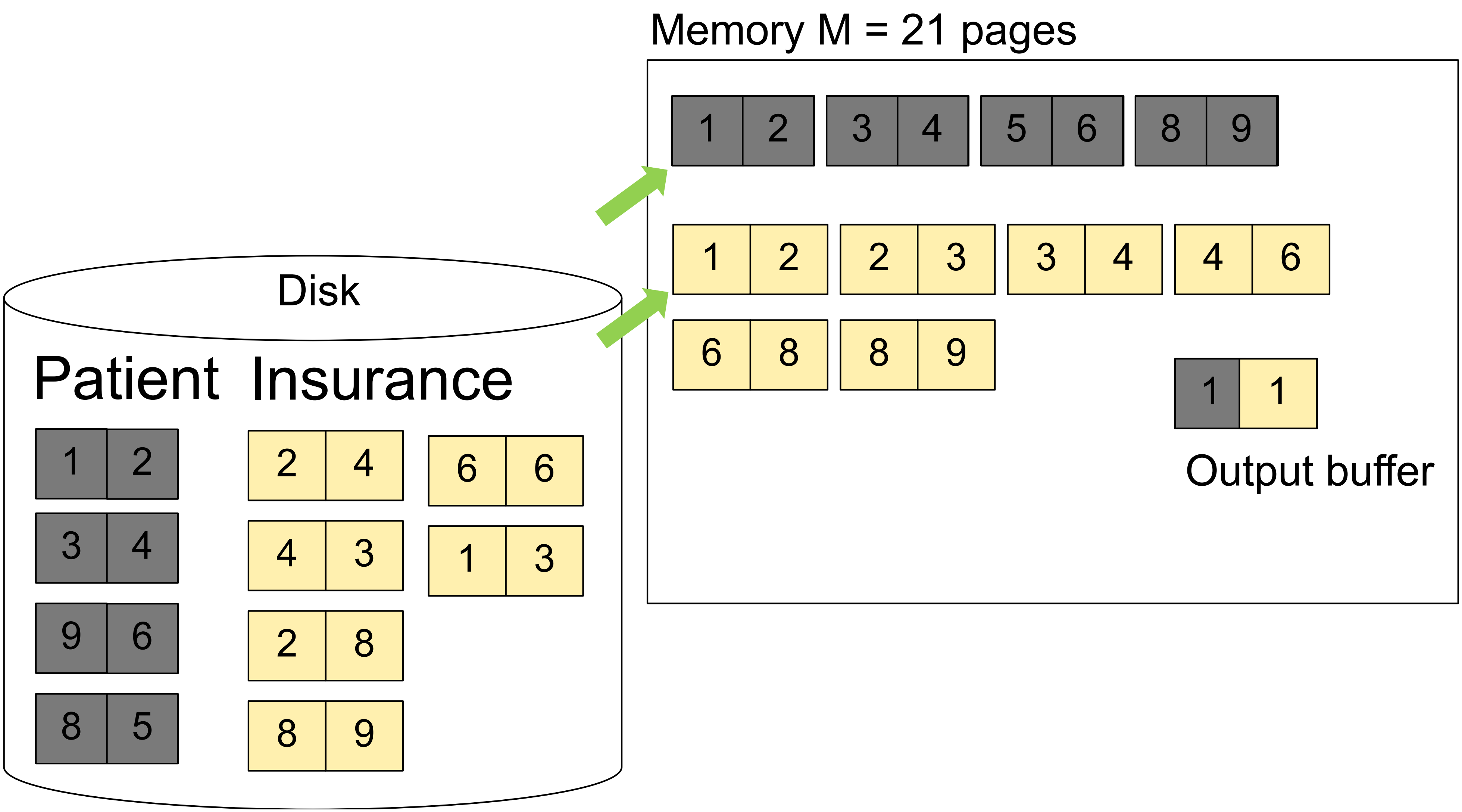
SORT-MERGE JOIN EXAMPLE

Step 2: Scan Insurance and sort in memory



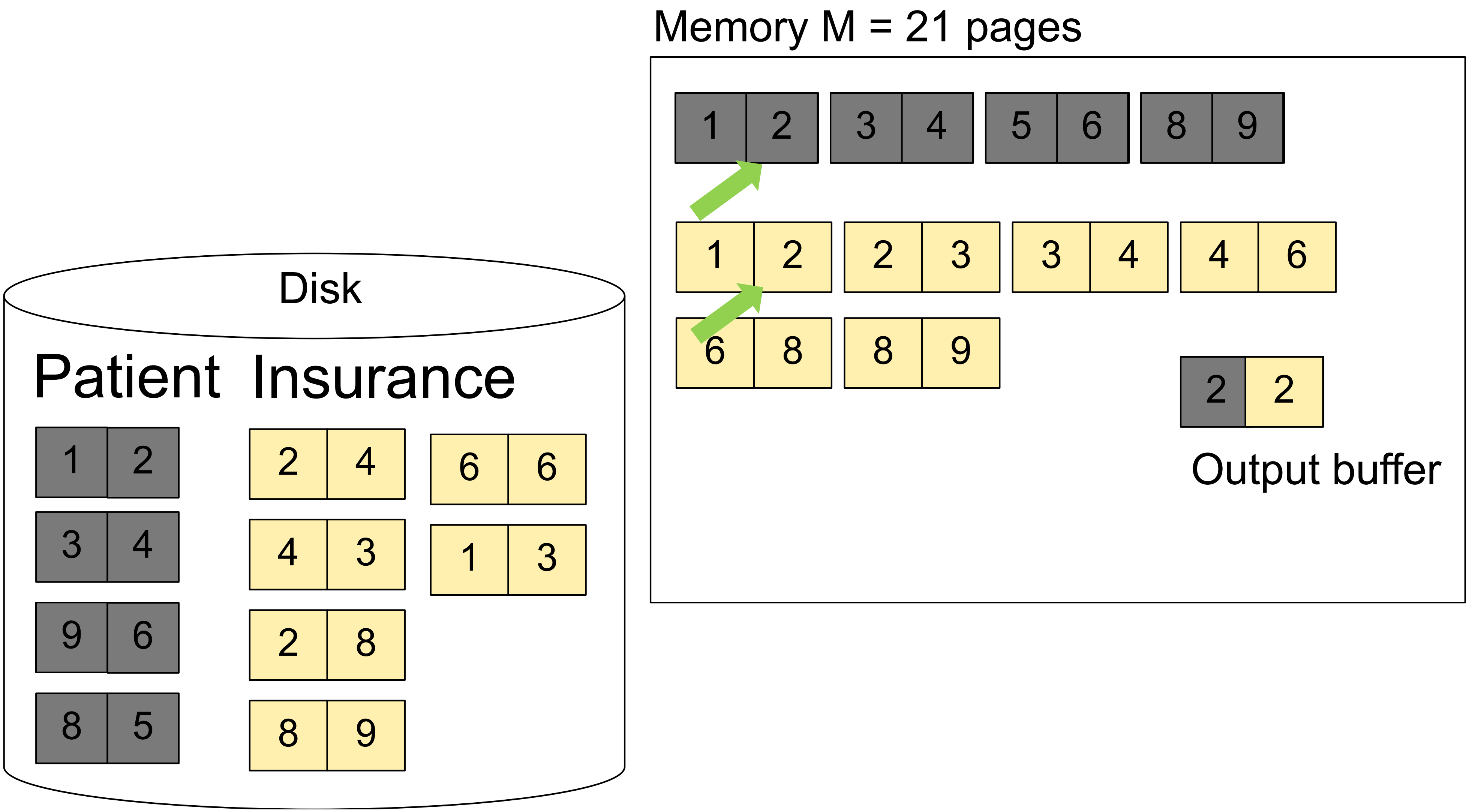
SORT-MERGE JOIN EXAMPLE

Step 3: Merge Patient and Insurance



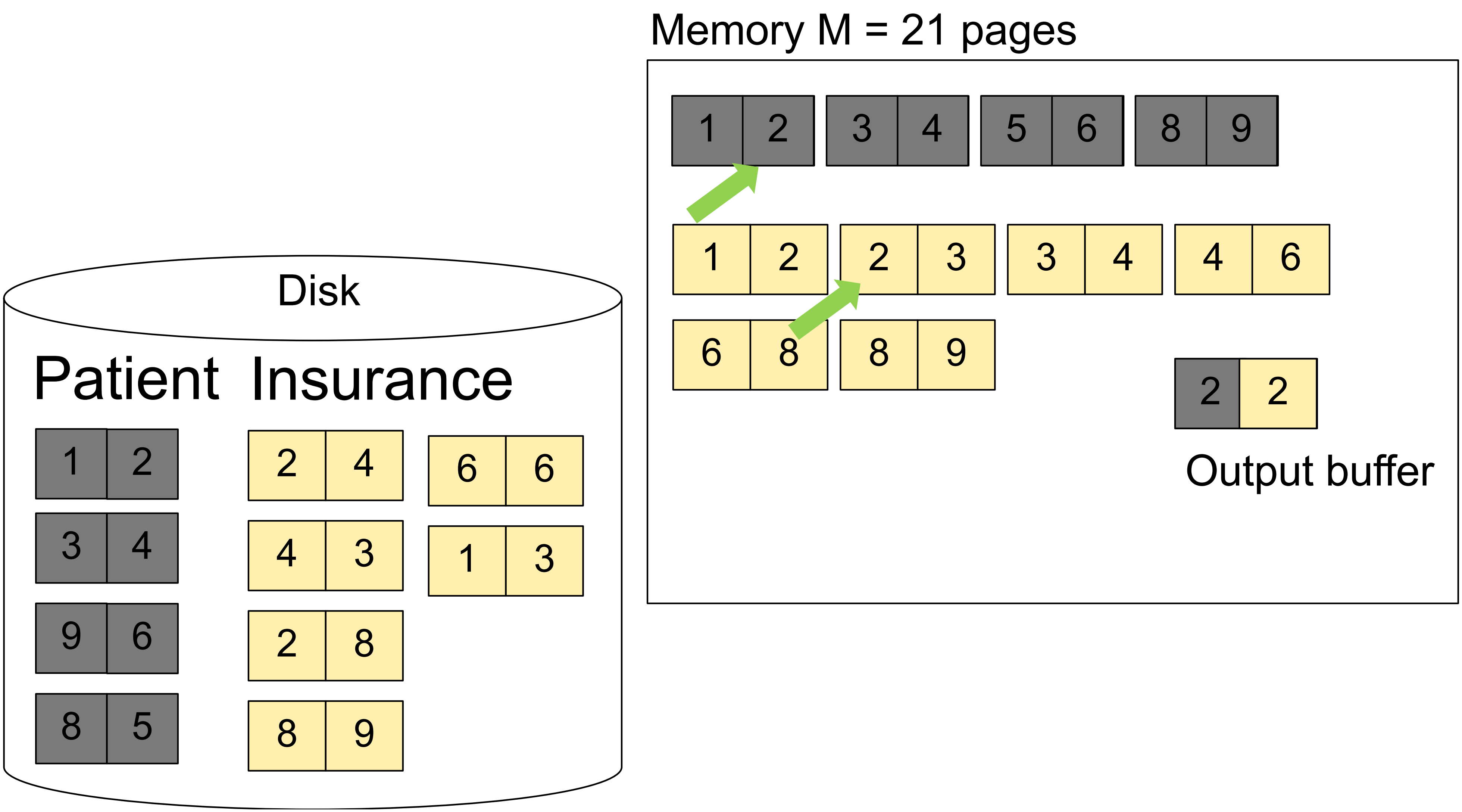
SORT-MERGE JOIN EXAMPLE

Step 3: Merge Patient and Insurance



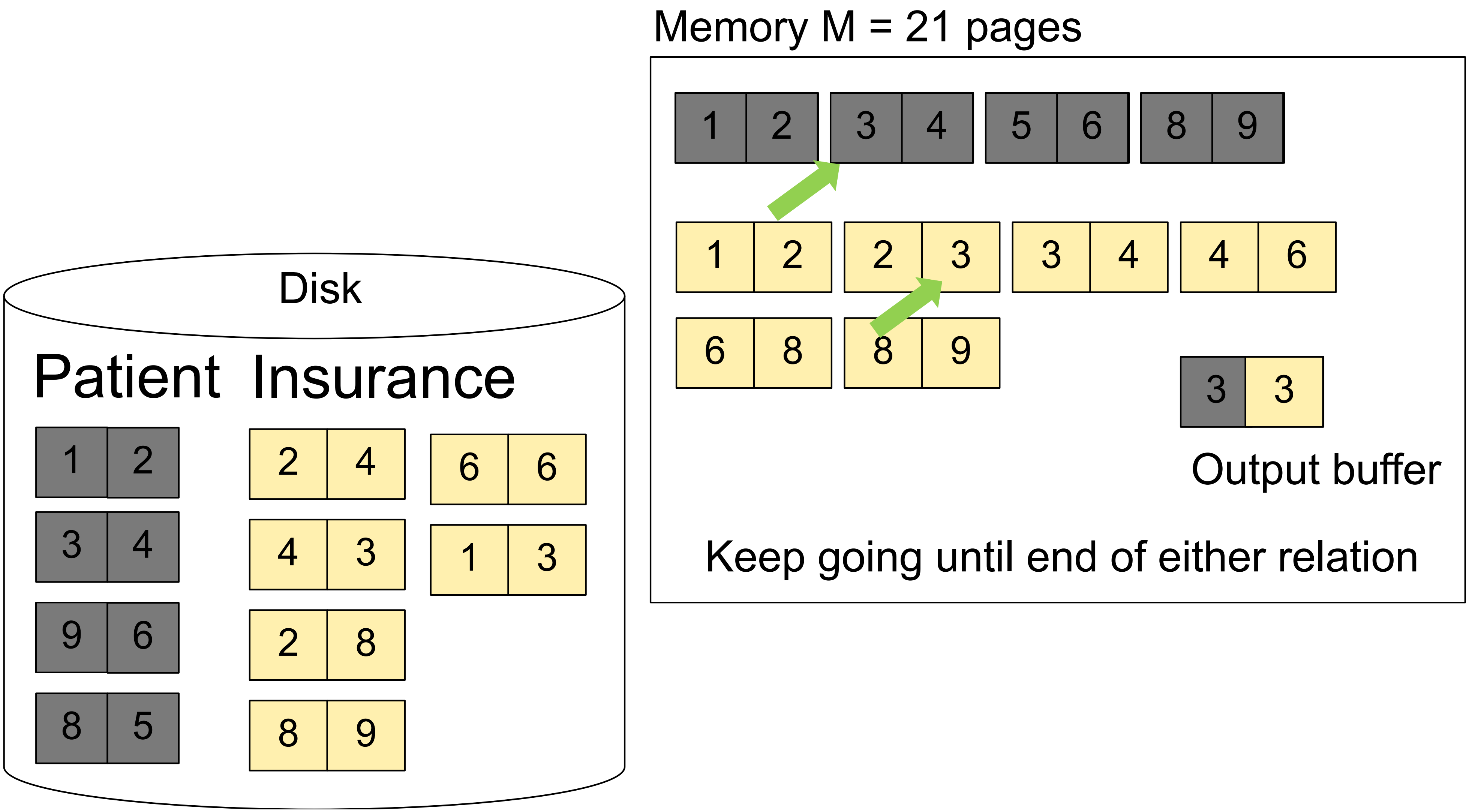
SORT-MERGE JOIN EXAMPLE

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

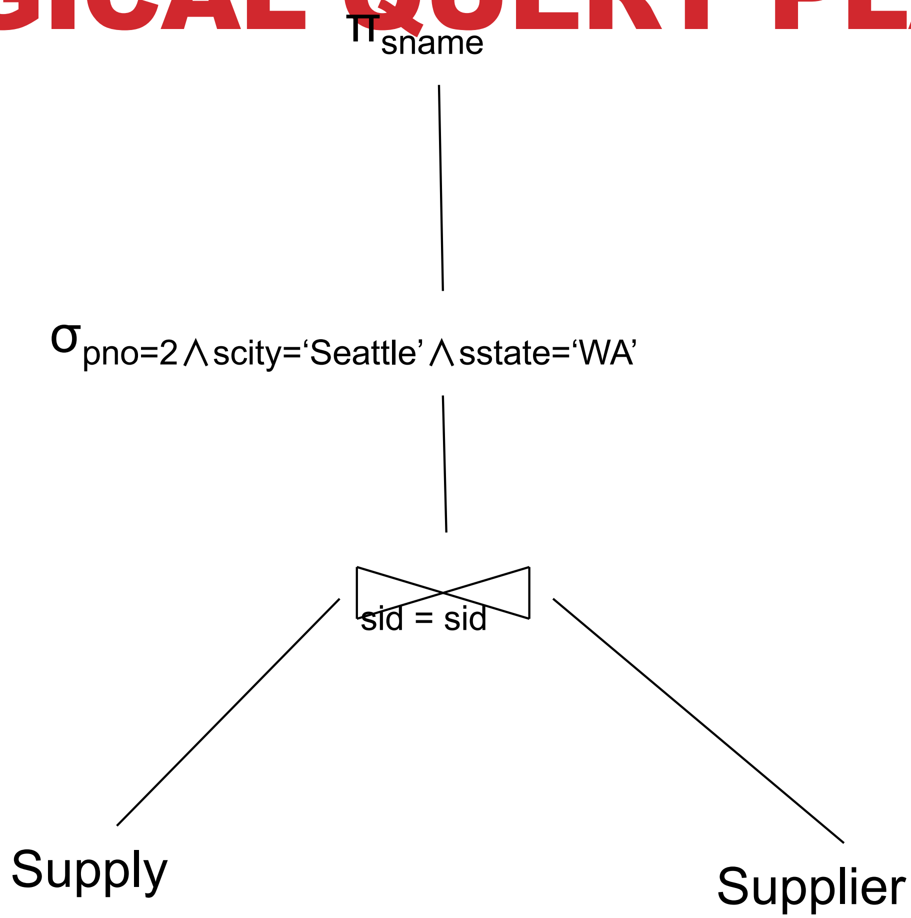
Iterate over R. For each <u>tuple</u>, fetch corresponding tuple(s) from S

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))
- If A is a *key*, then both are B(R) + T(R)

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

LOGICAL QUERY PLAN 1



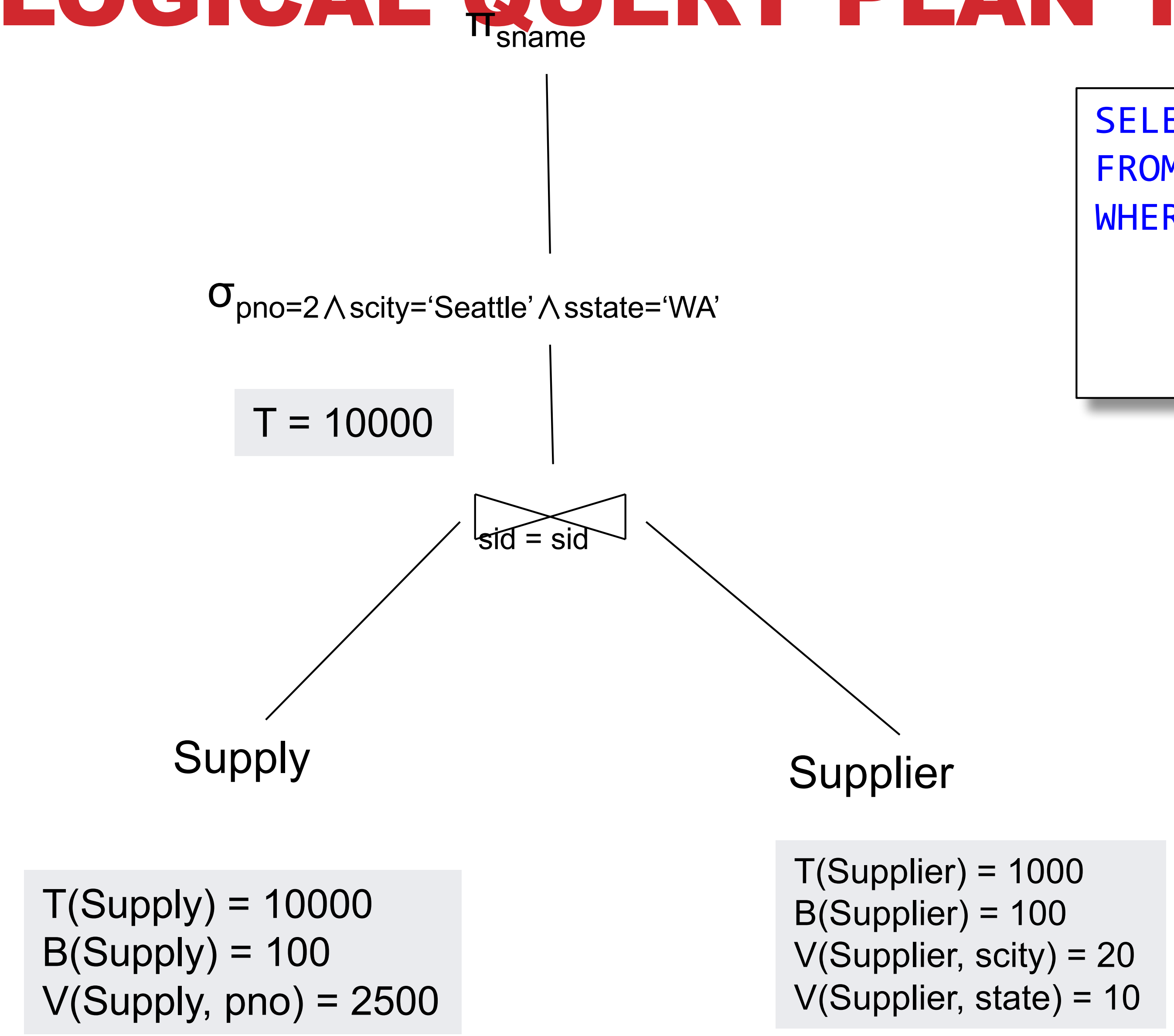

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

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

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

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

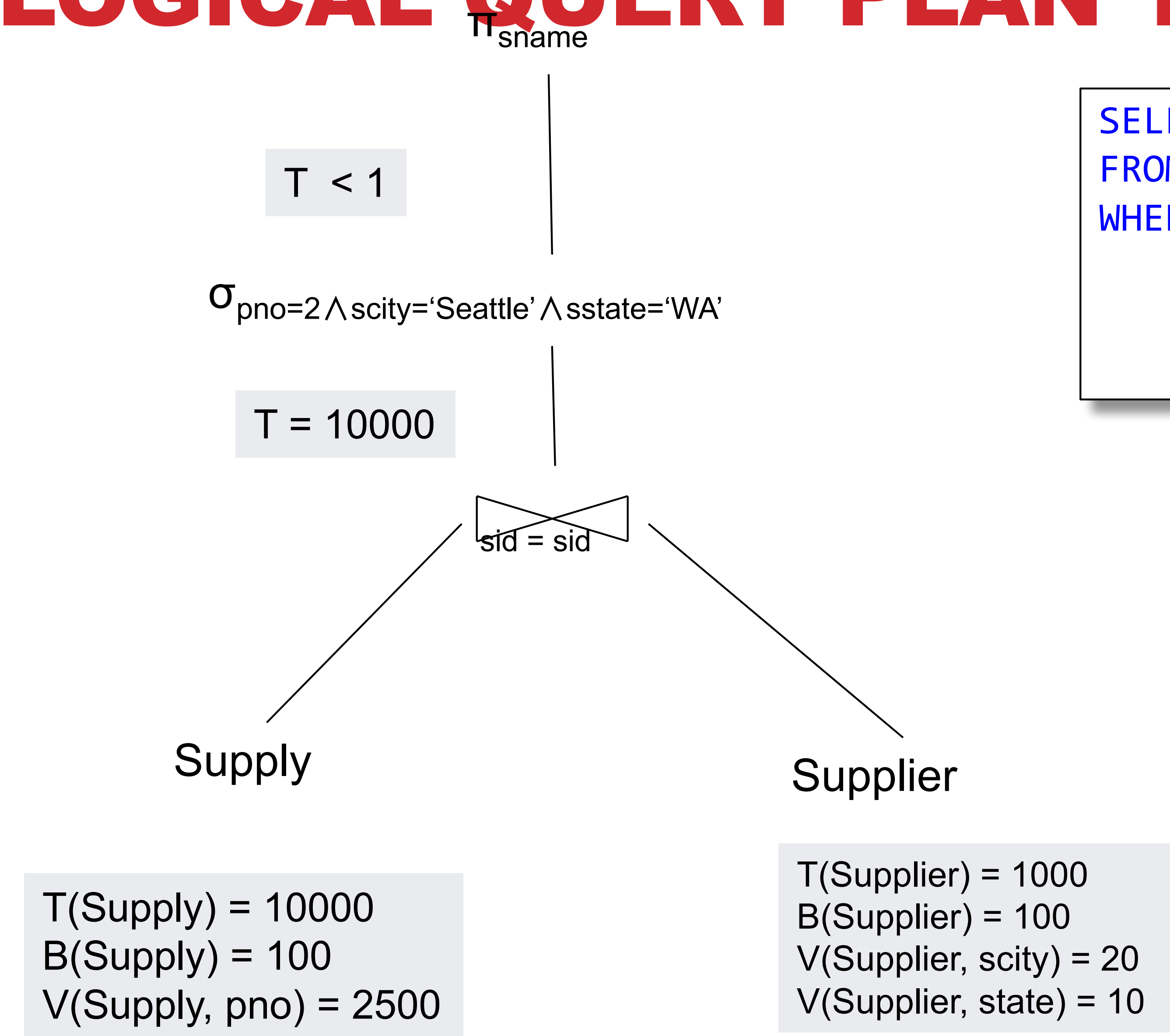
LOGICAL QUERY PLAN 1




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

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

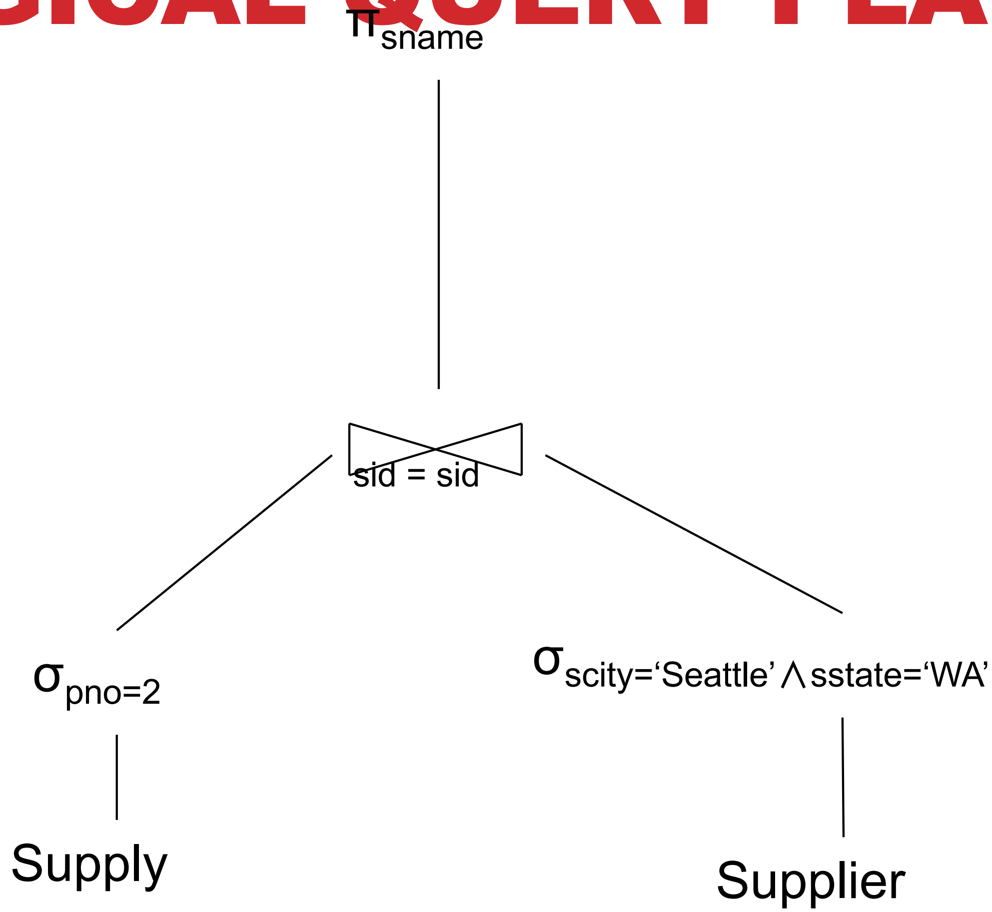
LOGICAL QUERY PLAN 1




```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'
```

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

LOGICAL QUERY PLAN 2



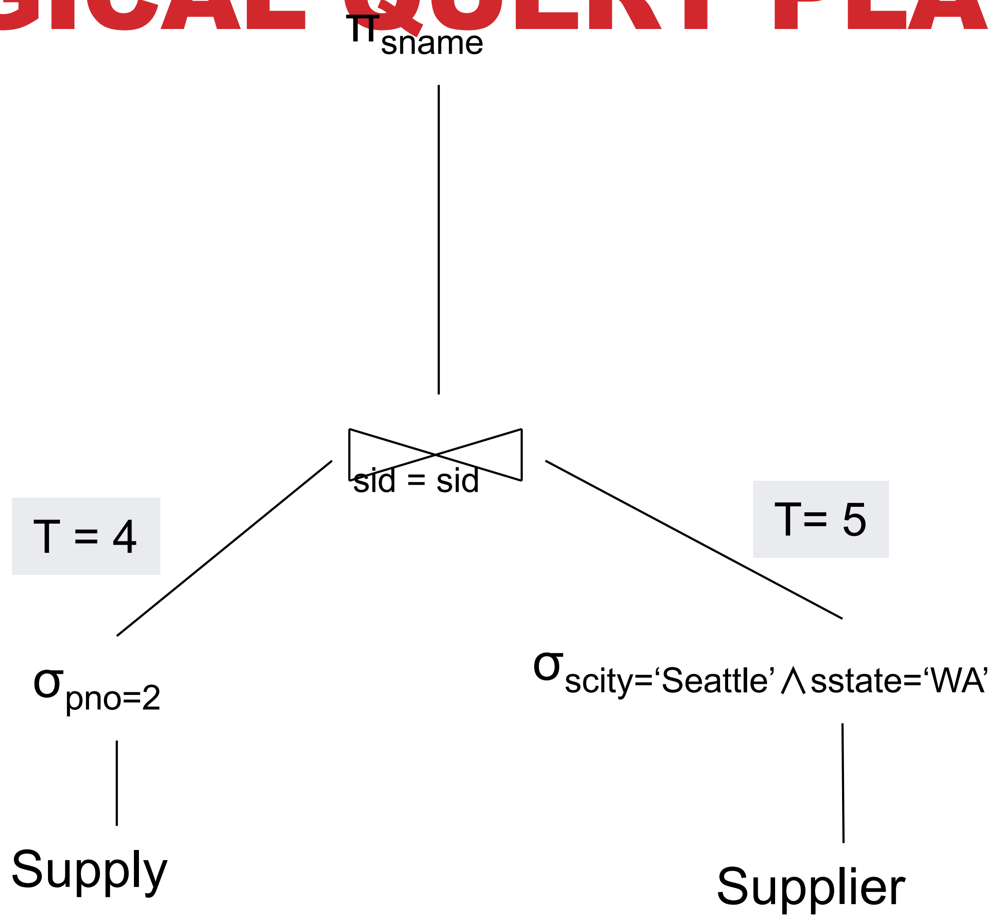

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

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

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

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

LOGICAL QUERY PLAN 2



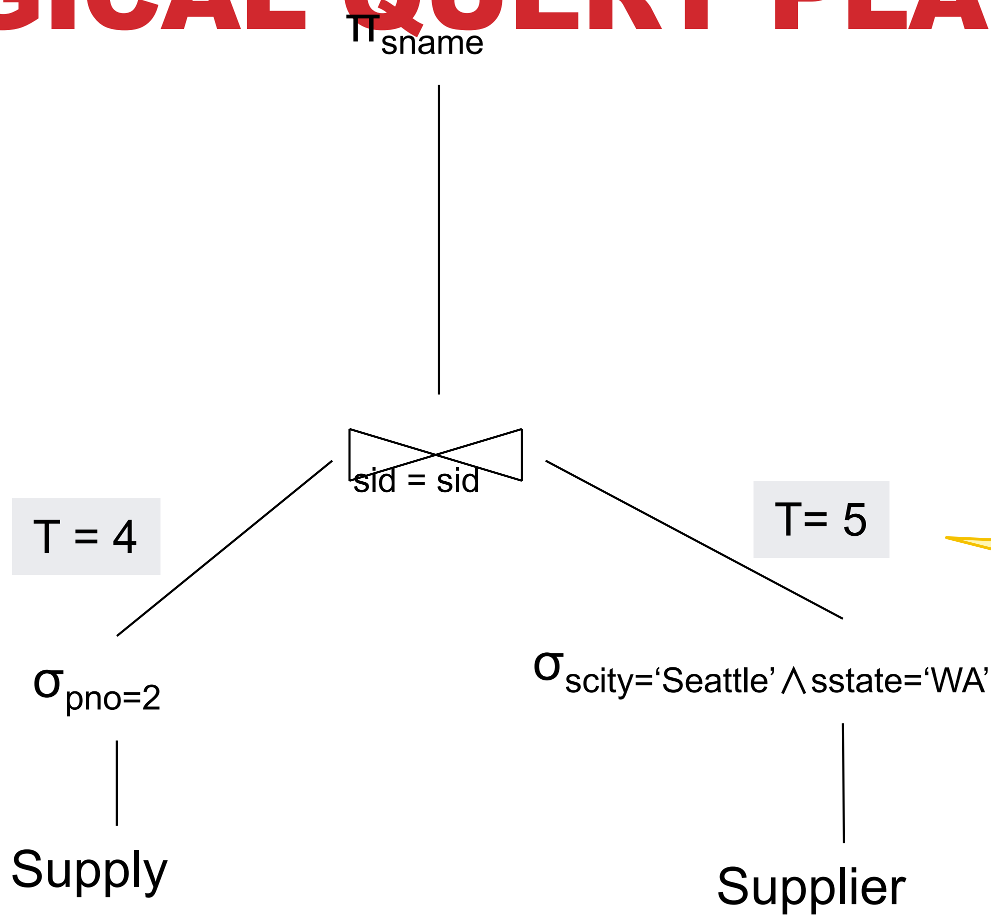

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

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

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

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'
```

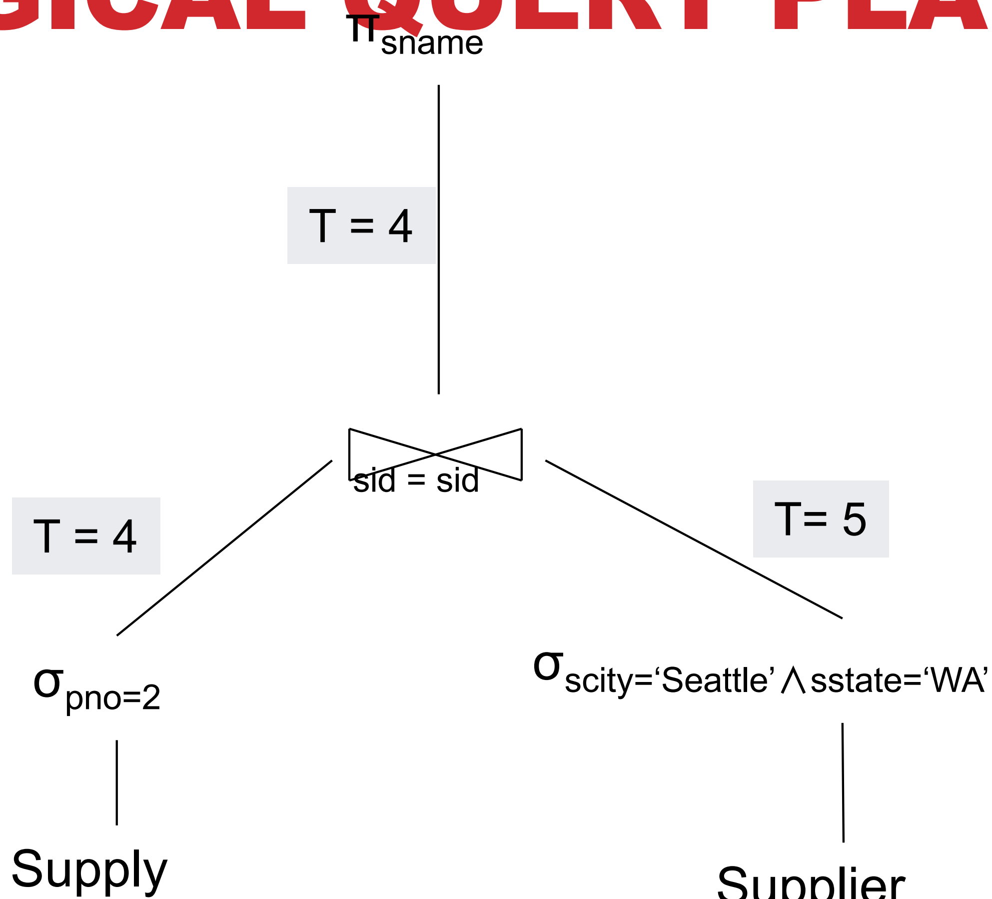
Very wrong!
Why?

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

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

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

LOGICAL QUERY PLAN 2



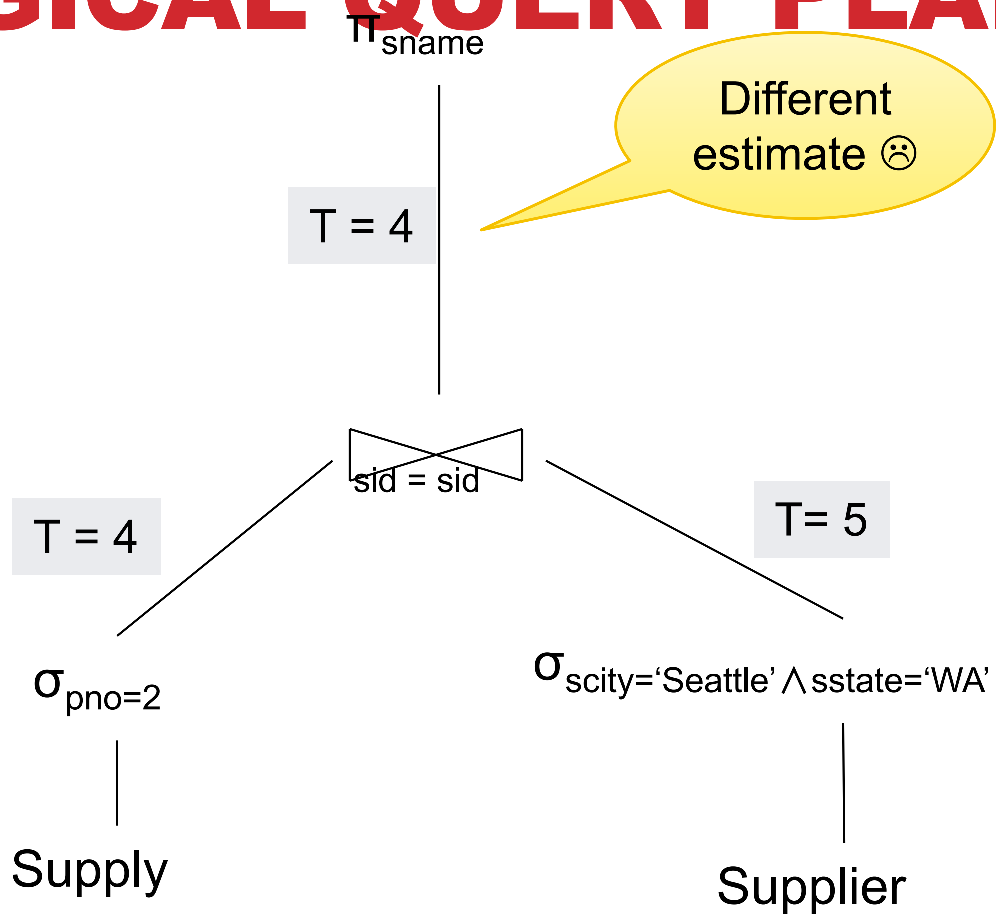

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

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

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

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

LOGICAL QUERY PLAN 2



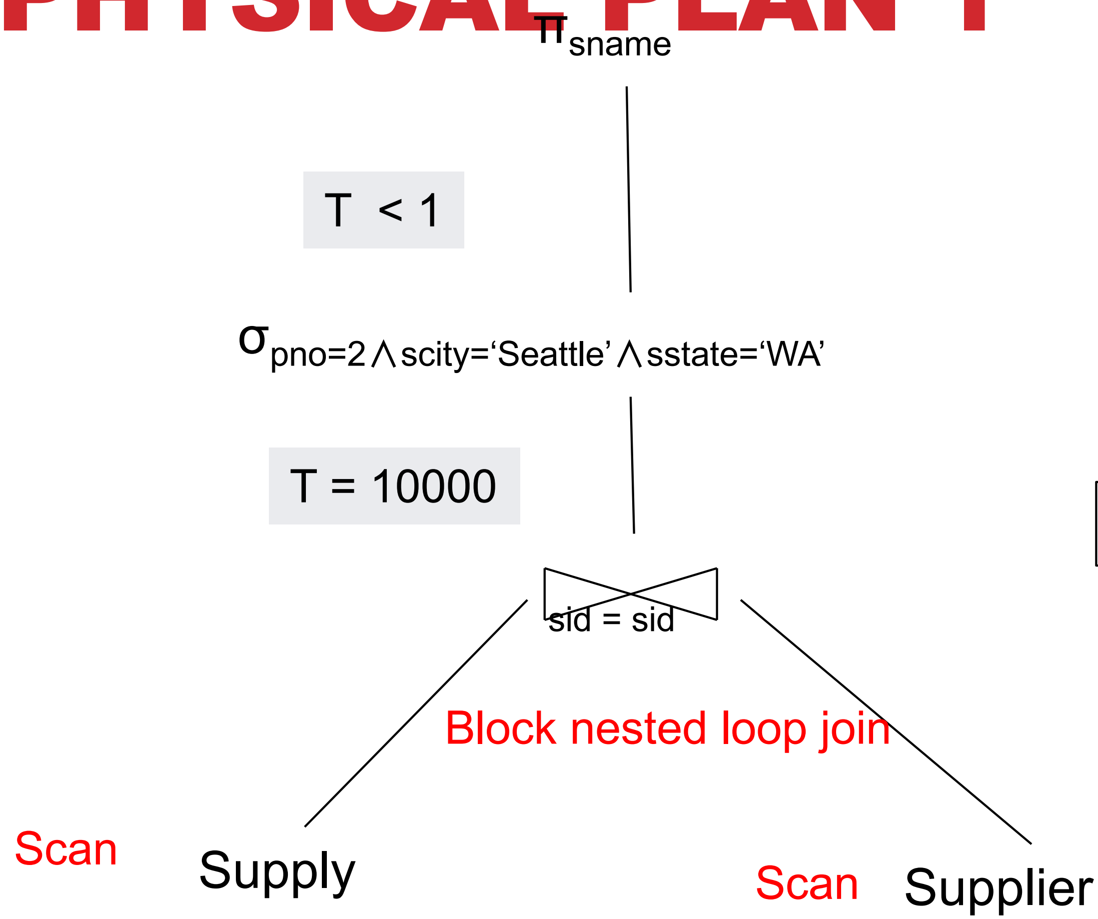

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

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

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

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

PHYSICAL PLAN 1



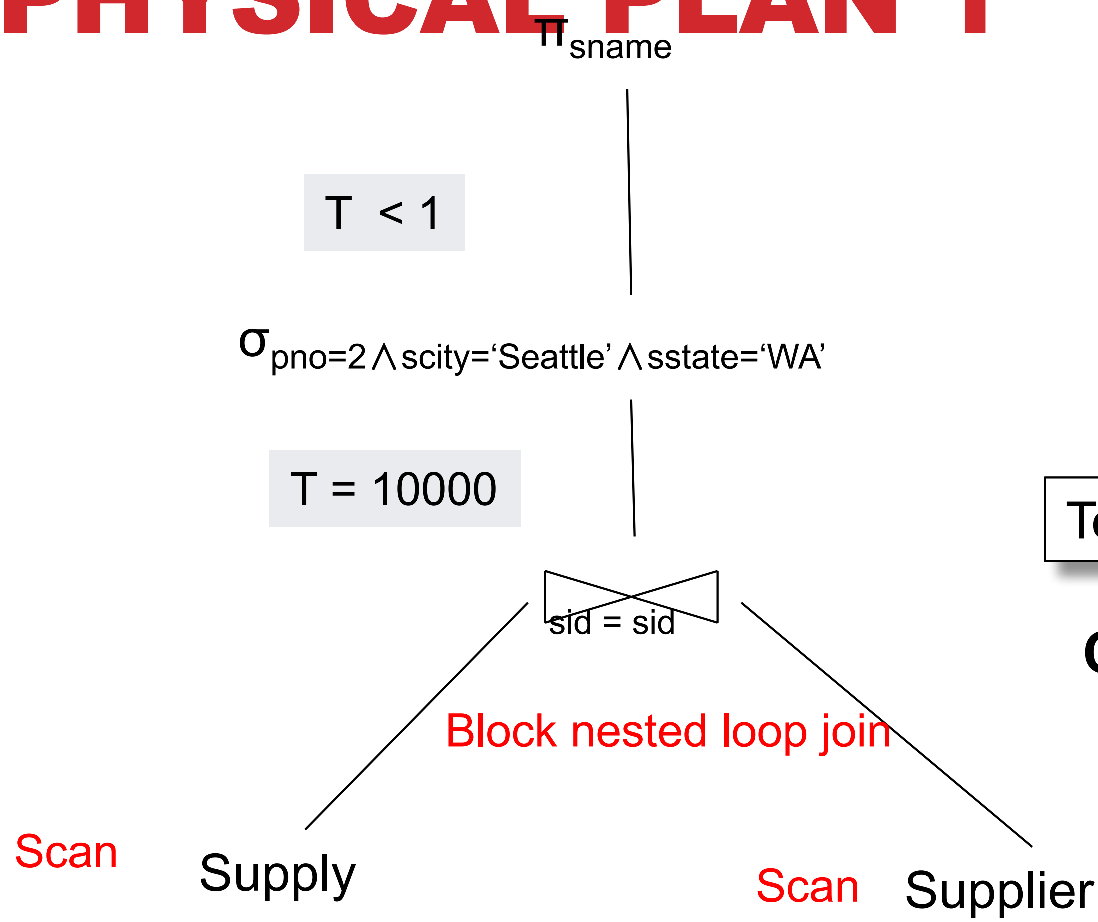
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

Total cost:

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

PHYSICAL PLAN 1



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

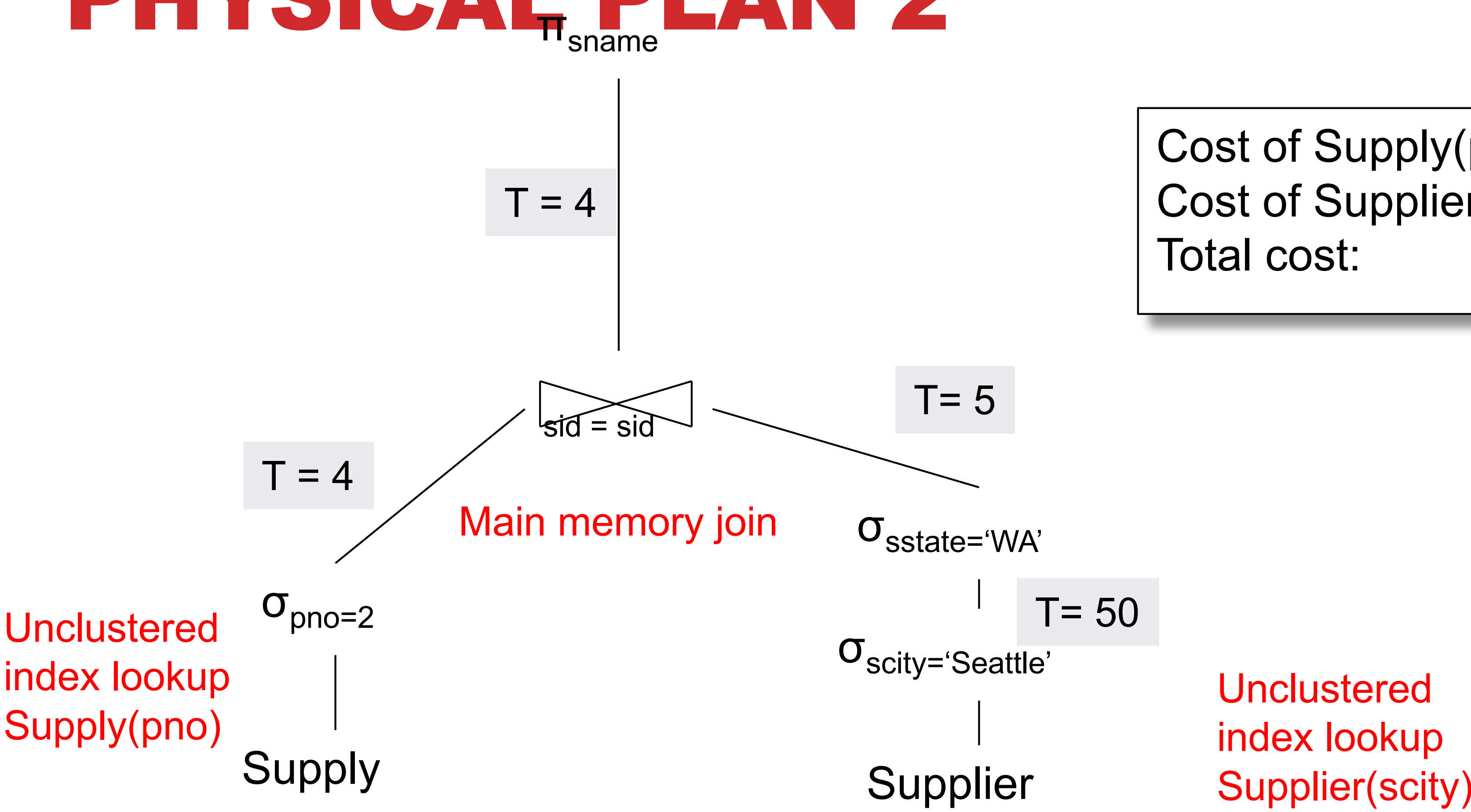
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

Total cost: 100+100*100 = 10,100

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

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

PHYSICAL PLAN 2



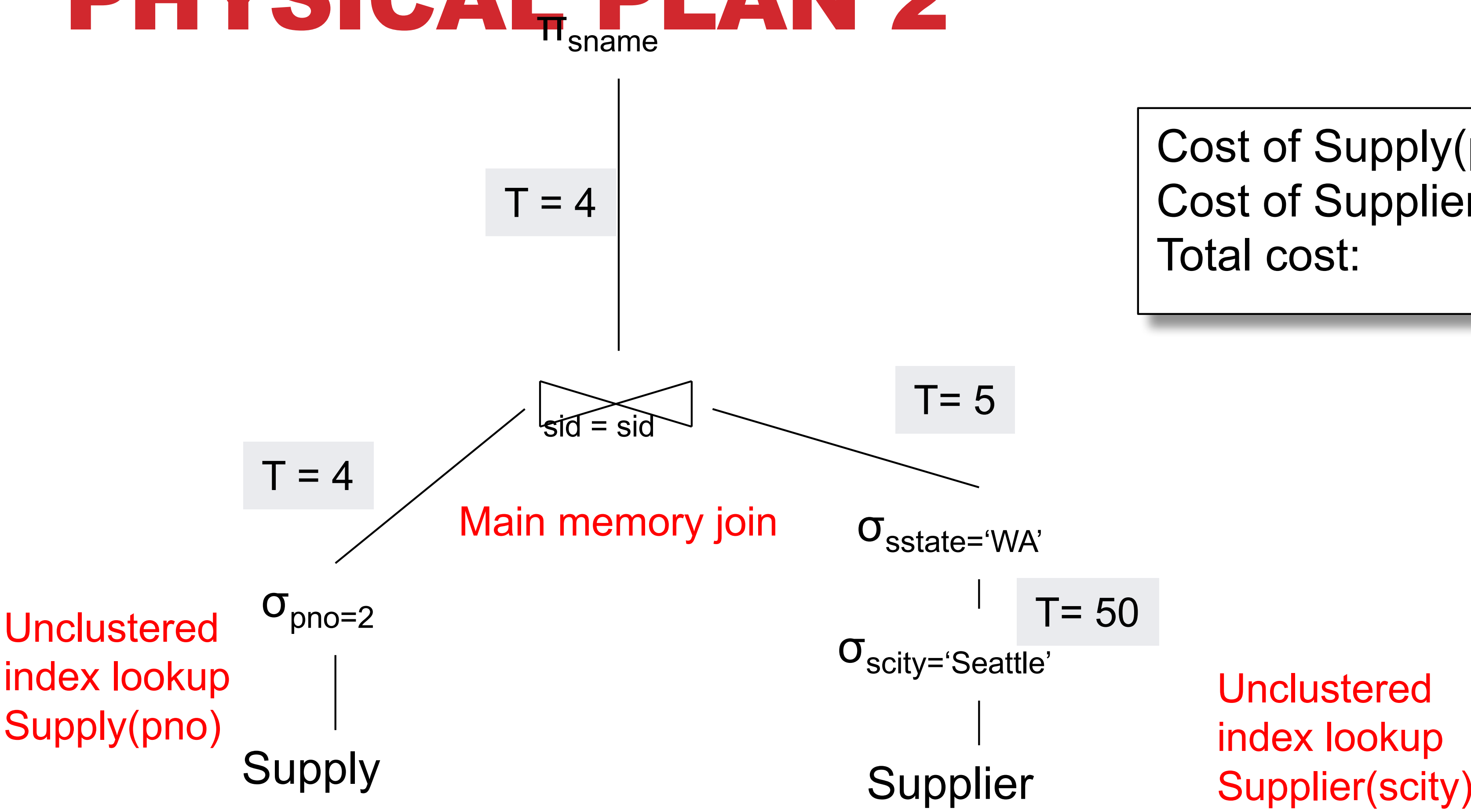
Cost of Supply(pno) =
Cost of Supplier(scity) =
Total cost:

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

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

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

PHYSICAL PLAN 2



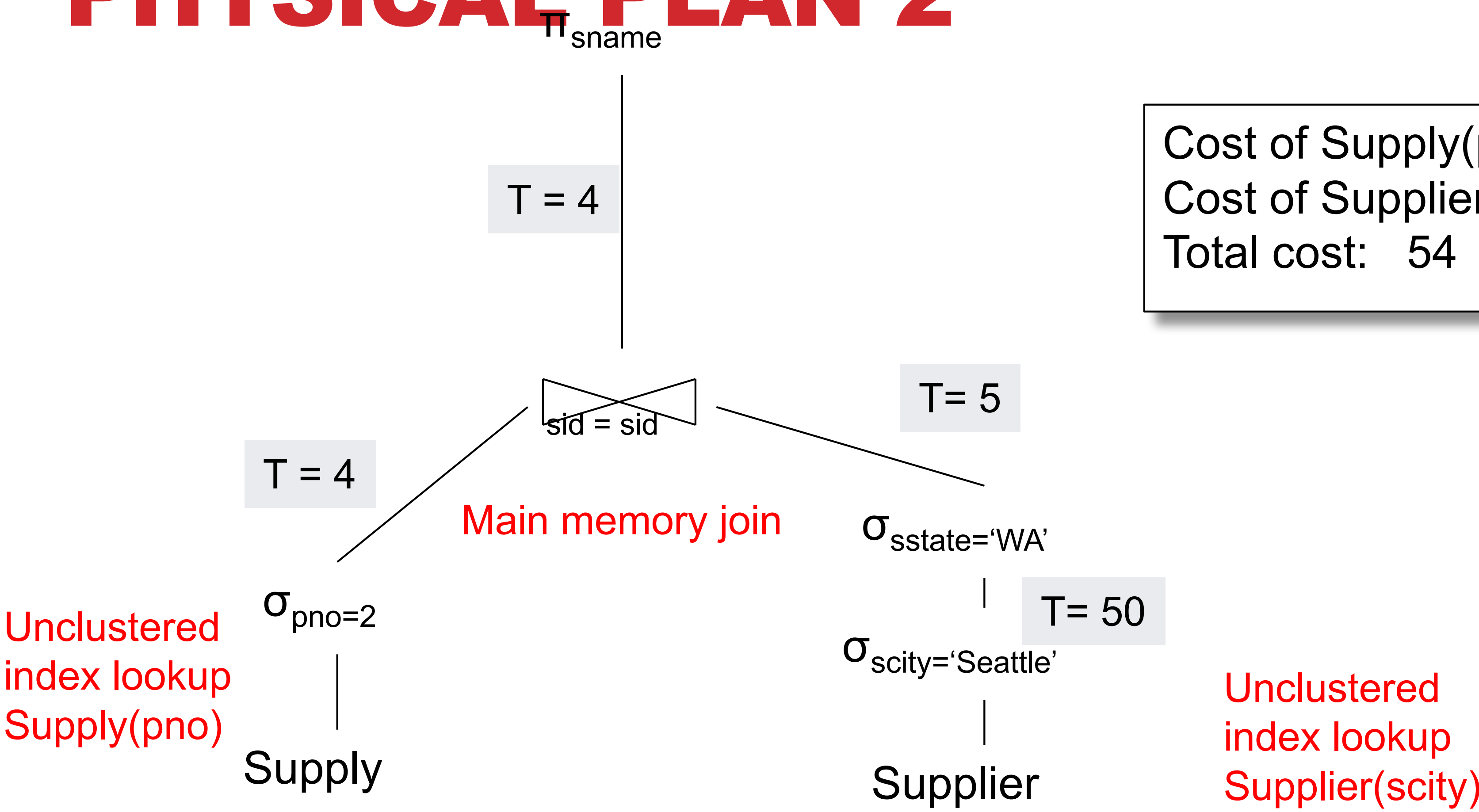
Cost of Supply(pno) = 4
Cost of Supplier(scity) =
Total cost:

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

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

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

PHYSICAL PLAN 2



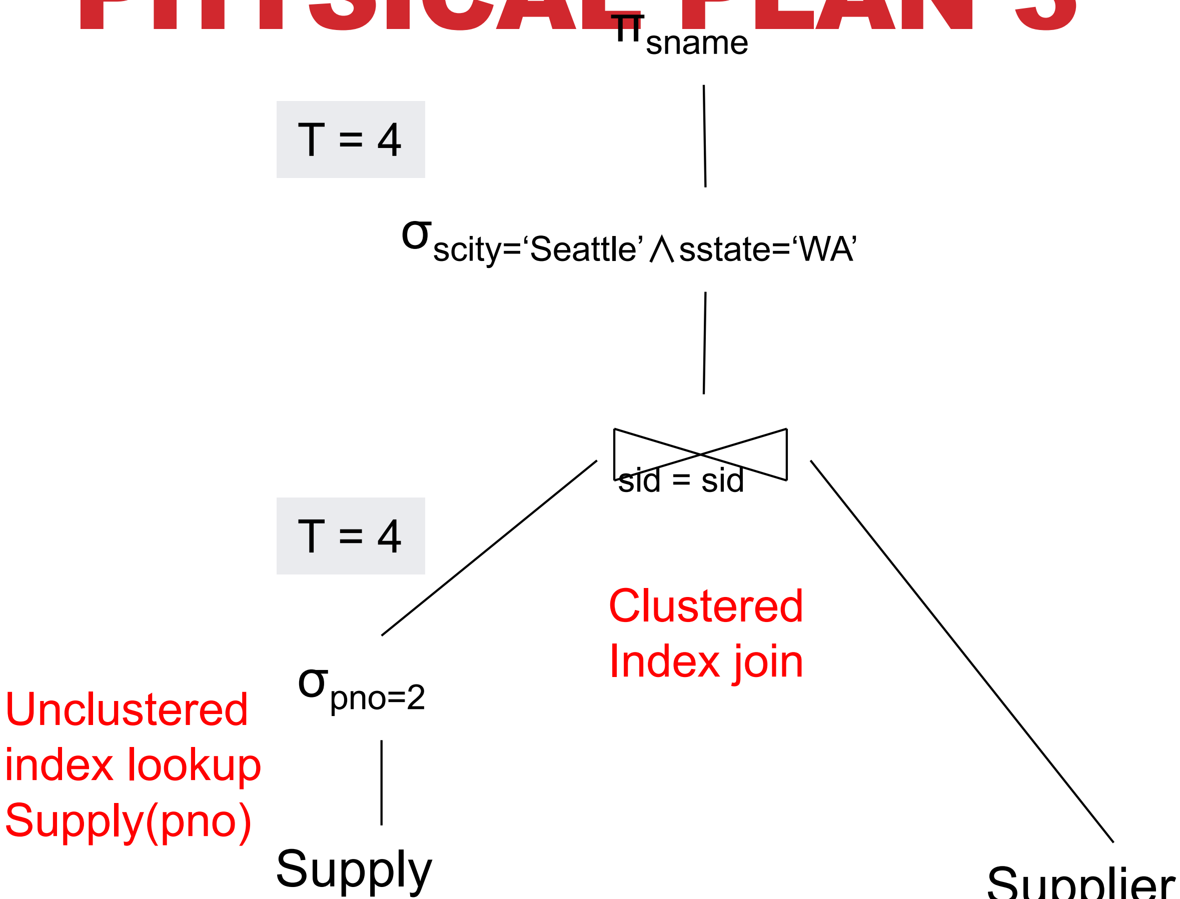
Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54

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

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

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

PHYSICAL PLAN 3



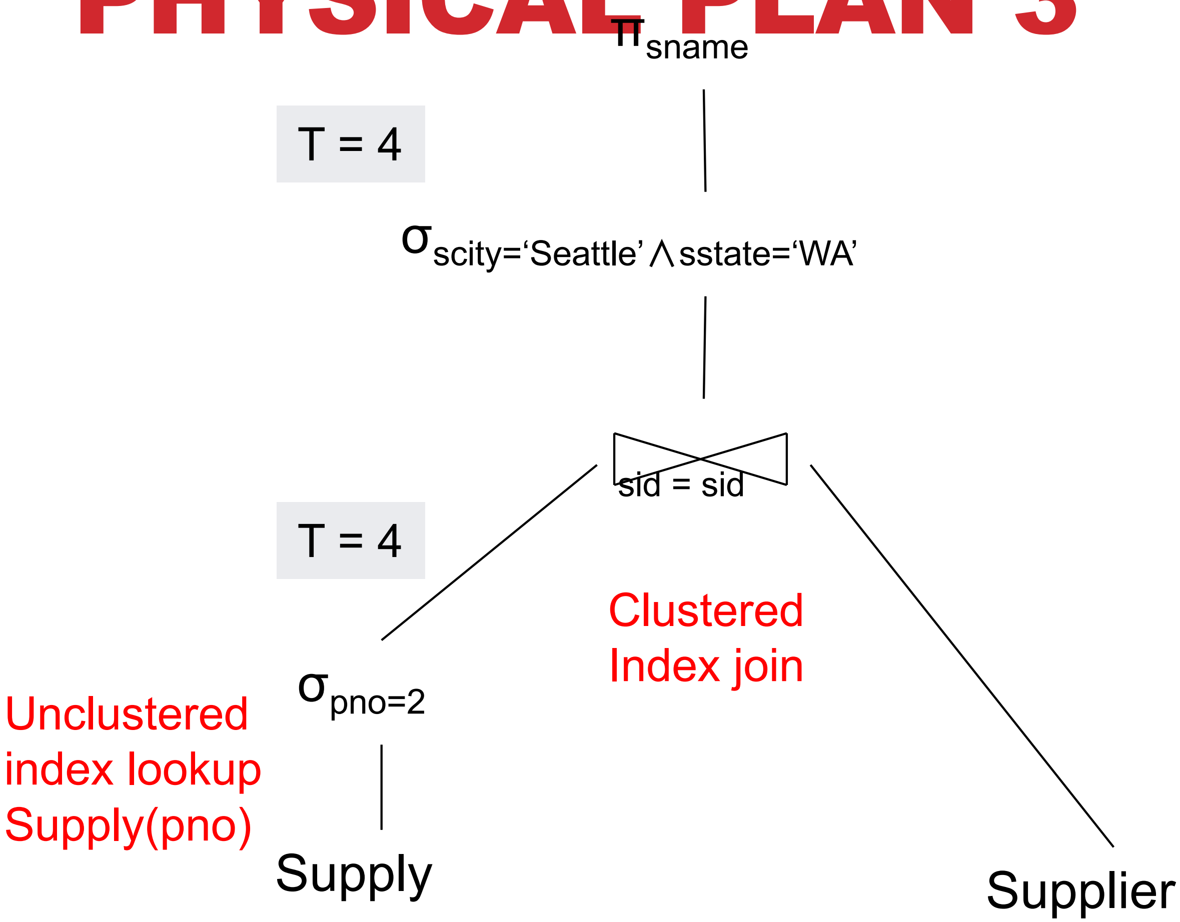
Cost of Supply(pno) =
Cost of Index join =
Total cost:

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

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

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

PHYSICAL PLAN 3



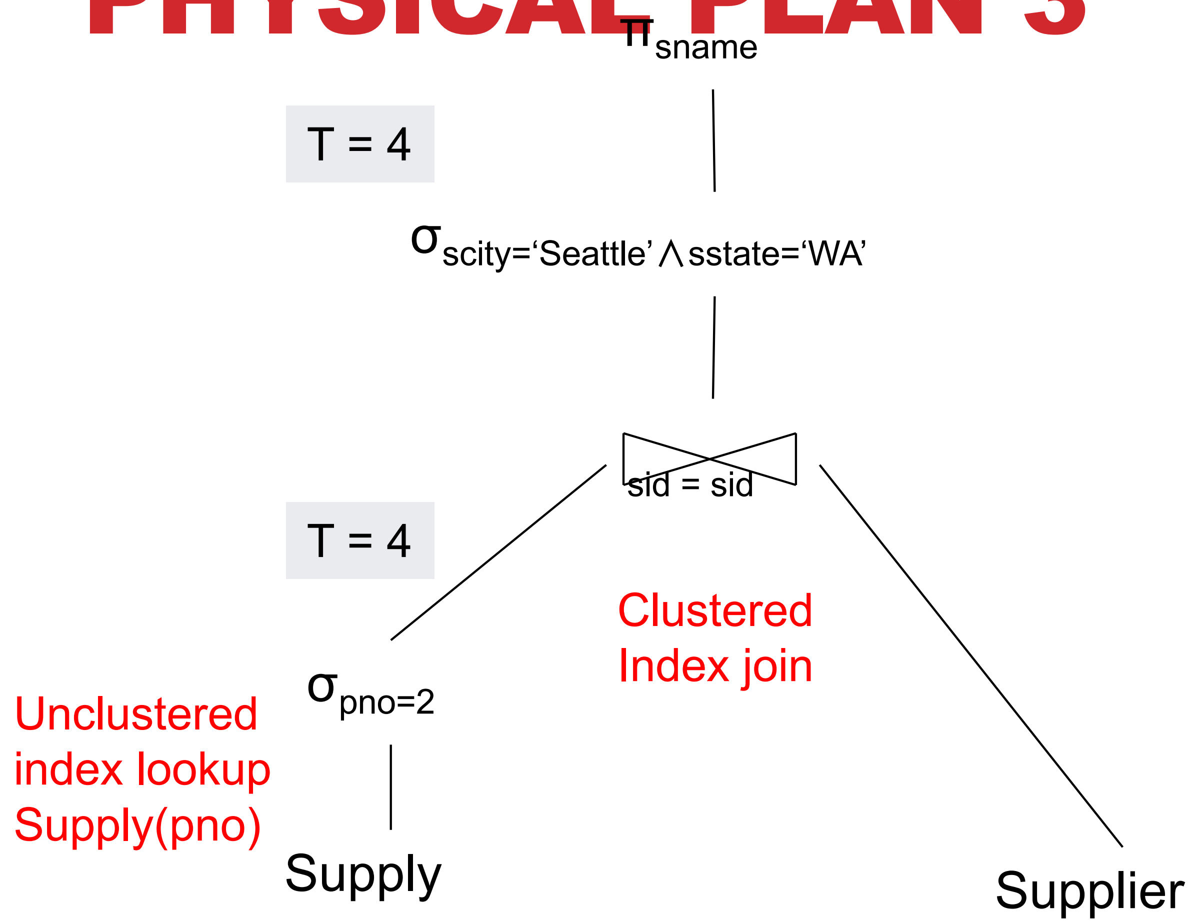
Cost of Supply(pno) = 4
Cost of Index join =
Total cost:

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

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

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

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