Introduction to Database Systems
CSE 414

NOVEMBER 5th

Lecture 17:
Basics of Query Optimization and Query Cost Estimation

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

TYPES OF queries:

point queries (i.e. WHERE X = stuff) —> Hash or Btree index
Range queries (i.e. WHERE X > stuff) —> Btree

Btree preserves ordering of elements, which is why it is good for range queries where the items we want will be adjacent (if indexed by x)

Cost of Reading Data From Disk

Cost Parameters

Disk reads/writes transferring data

processing data

communicating between multiple machines

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class by FAR the slowest; measure efficiency in terms of disk reads
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R

    i.e. cost of a sequential read of entire relation is B(R); have to read every block
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

    function of our relation, and the 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)**

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

i.e. stores info so it doesnt have to recalculate it every time…
Like histogram of all values of attribute a

something that reduces the total amount of data (blocks) you actually have to read

i.e. rough estimate of what proportion of the blocks/tuples of R does this query select

NOT exact calculation

How much does this operation reduce the number of tuples we have to deal with?

Selectivity Factors for Conditions

- A = c /* $\sigma_{A=c}(R)$ */
  - Selectivity = 1/V(R,A)

SELECT *
FROM one_year
WHERE day_id = 32

selects one tuple (day 32) out of 365

SELECT *
FROM one_year
WHERE month_id = Jan

selects about 1/12 of the total number of tuples

i.e. assumes each distinct value of A is equally likely

- A < c /* $\sigma_{A<c}(R)$*/
  - Selectivity = (c - min(R, A))/(max(R,A) - min(R,A))

SELECT *
FROM one_year
WHERE month_id > 4

selectivity = (4 - 1)/ (12 - 1)

- c1 < A < c2 /* $\sigma_{c1<A<c2}(R)$*/
  - Selectivity = (c2 – c1)/(max(R,A) - min(R,A))

Combining selectivity

condition 1 AND condition 2 AND condition 3

Selectivity = Selectivity1*Selectivity2 *Selectivity3 assuming these conditions are indepedent.

Cost of Reading Data From Disk

- Sequential scan for relation R costs **B(R)**

  # blocks in R

  i.e. read entire table in order from data files

- Index-based selection
  - Estimate selectivity factor **f** (see previous slide)
  - Clustered index: f***B(R)** guarenteed that tuples next to each other on index are next to each other on disk, so do not have to read every block
  - Unclustered index f***T(R)** might have to read many blocks; pointers go to differemt blocks sporadically; no adjancency

Note: we ignore I/O cost for index pages

fairly fast to read index pages so fairly negligible compared to other reads.

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

i.e. this last one is worse because we end up accessing the same block multiple times, as we dont know where each index points (our unclustered indexes dont point to adjacent points in data file)

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

although databases can sometimes recognize this mistake for you and just read entire B(R)

Cost of Executing Operators (Focus on Joins)

Outline

some of the most costly operations

- **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 R, build buckets in main memory
- Then scan S and join
- Cost: B(R) + B(S)
- Which relation to build the hash table on?

Does not matter?

- One-pass algorithm when $B(R) \leq M$
  - M = number of memory pages available

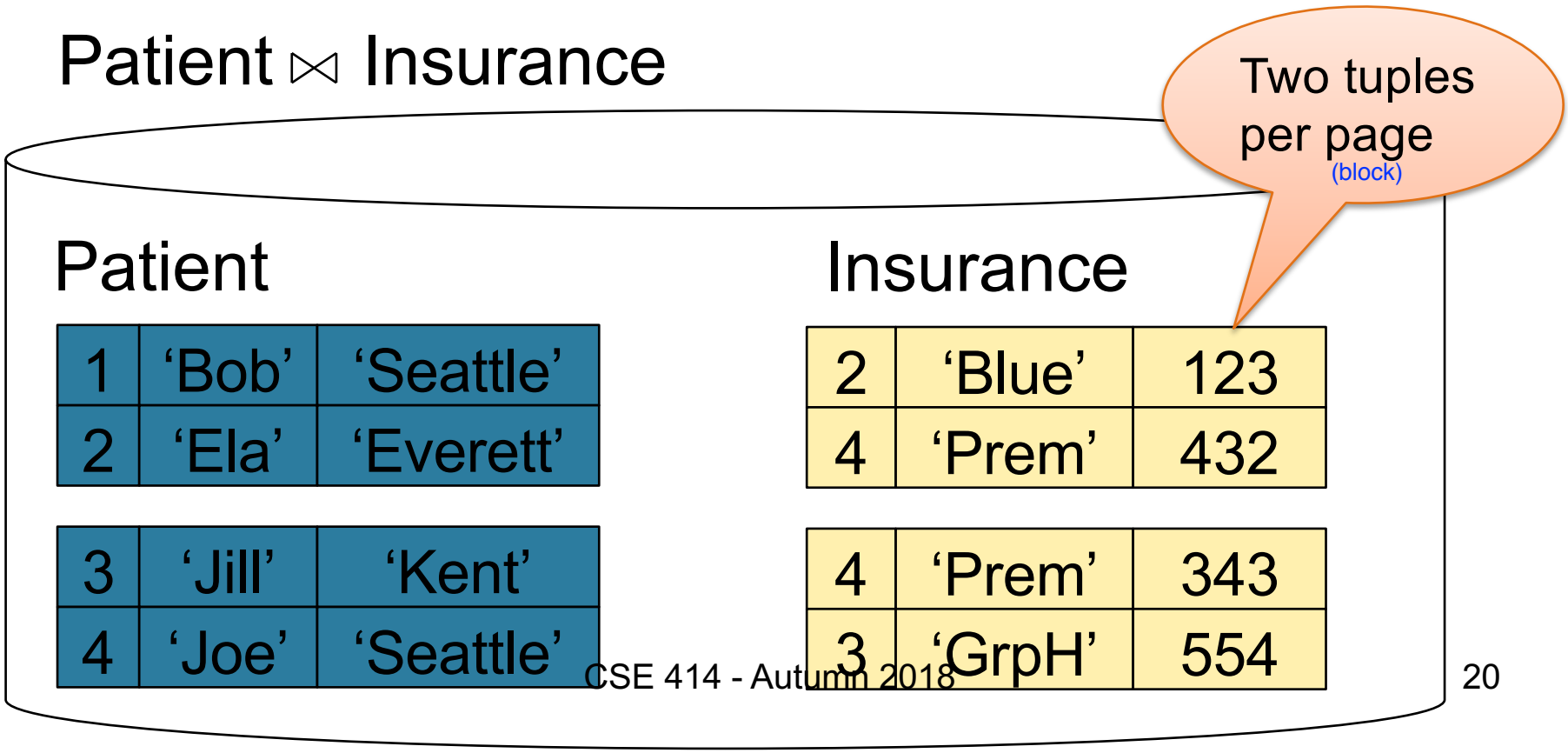
for this to be fast, whole hash table must fit in main memory for fast access

Hash Join Example

Patient(pid, name, address)

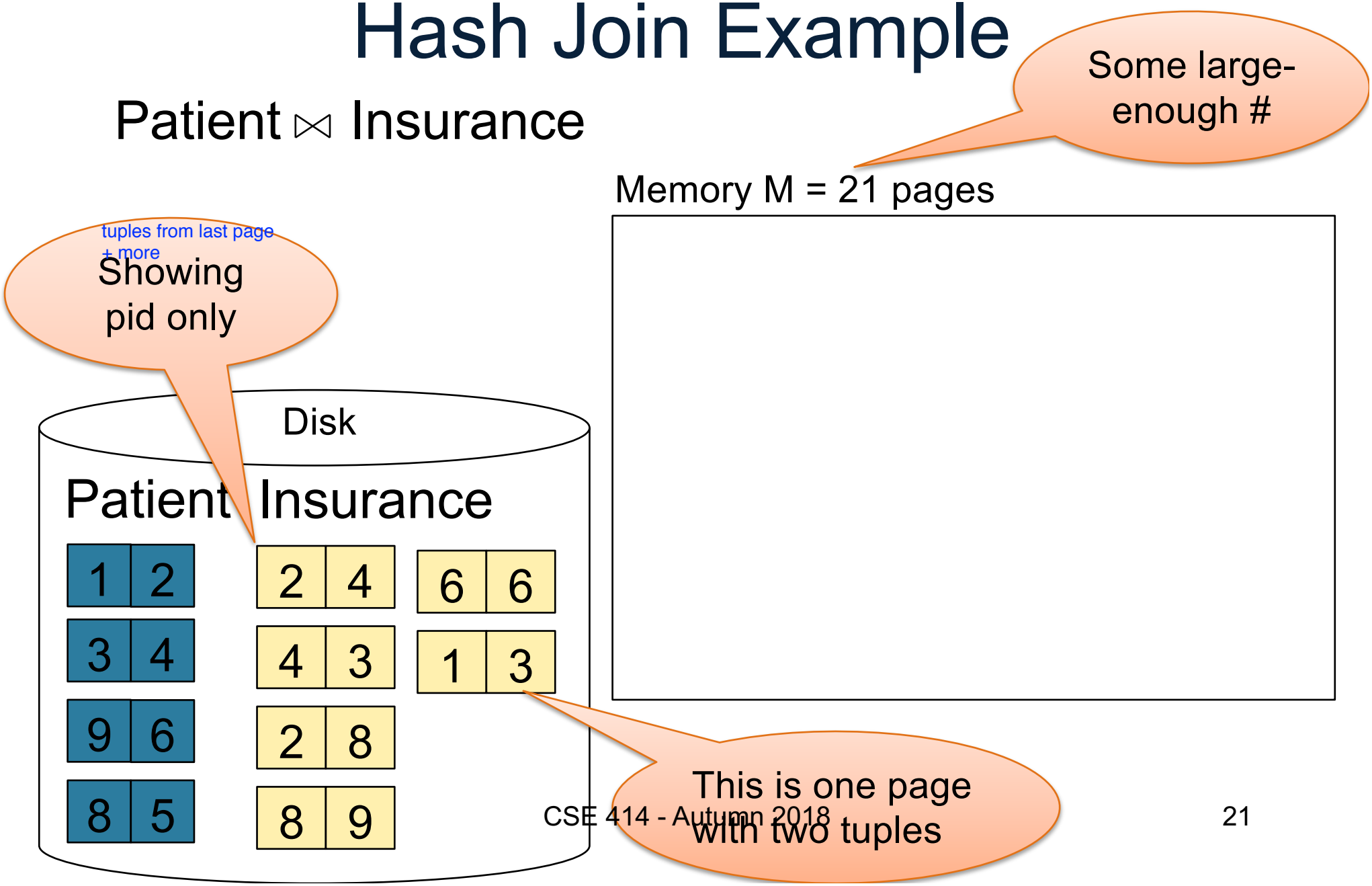
Insurance(pid, provider, policy_nb)

Patient ⋈ Insurance



Hash Join Example

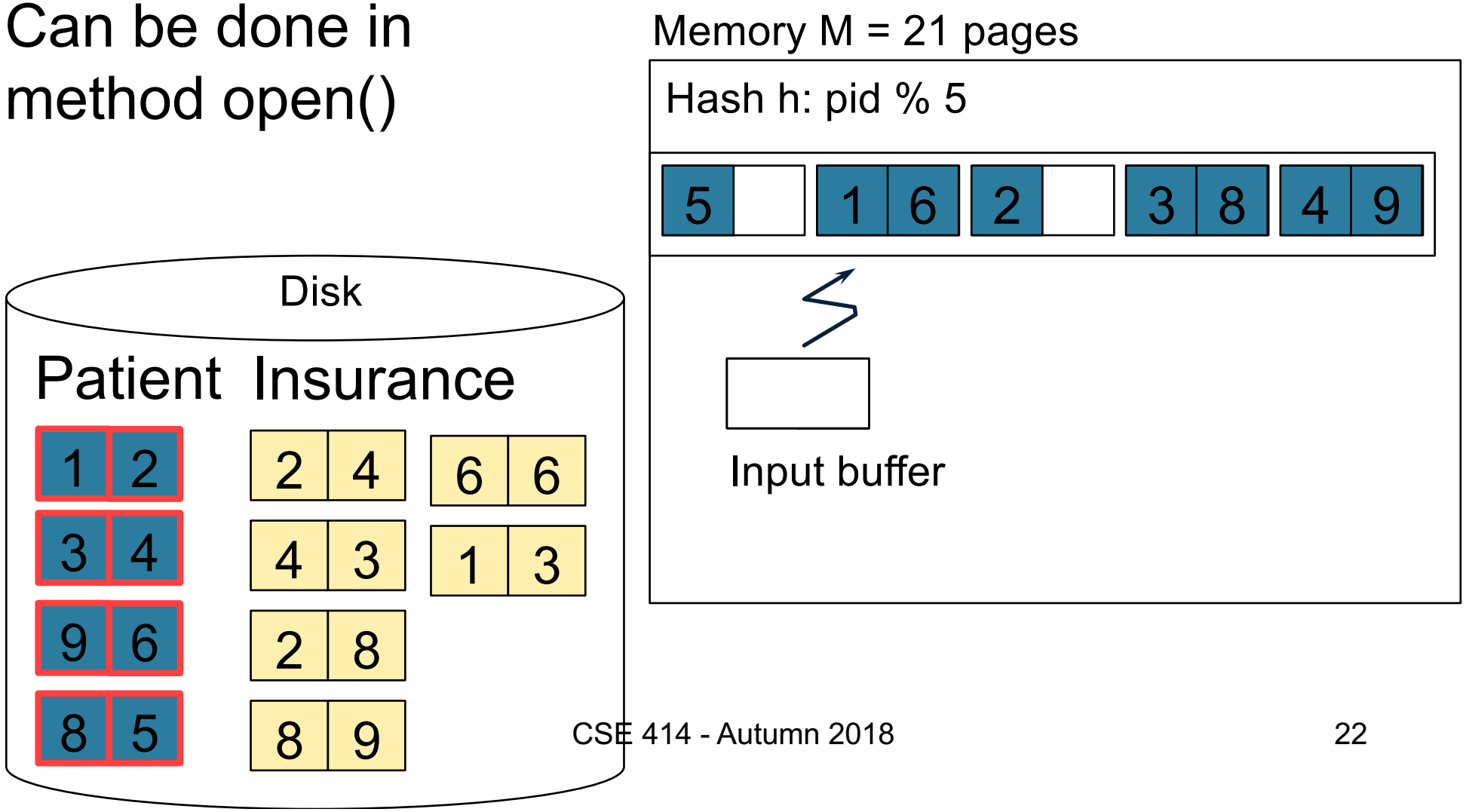
Patient ⋈ Insurance



Hash Join Example

Step 1: Scan Patient and build hash table in memory

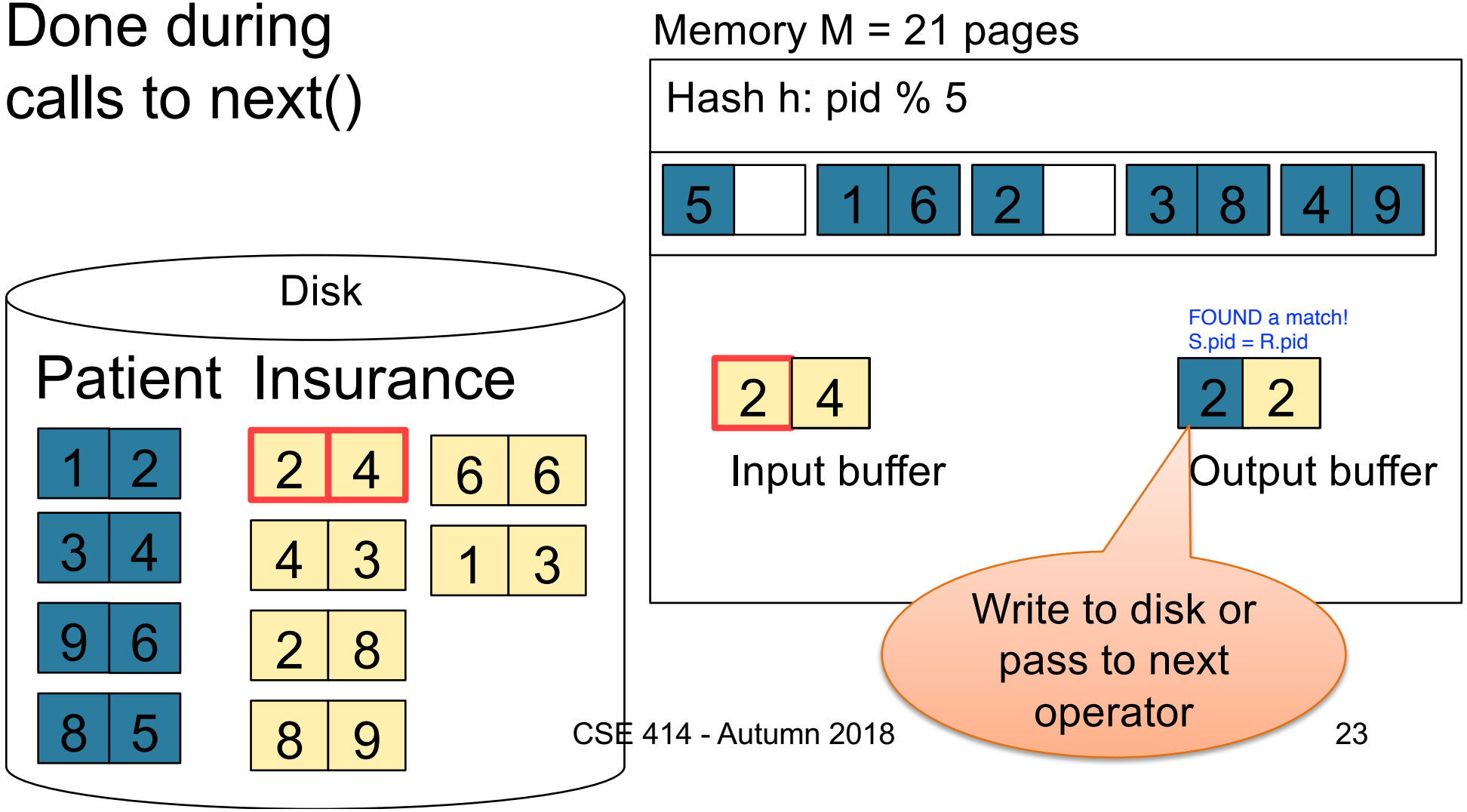
Can be done in method open()



Hash Join Example

Step 2: Scan Insurance and probe into hash table

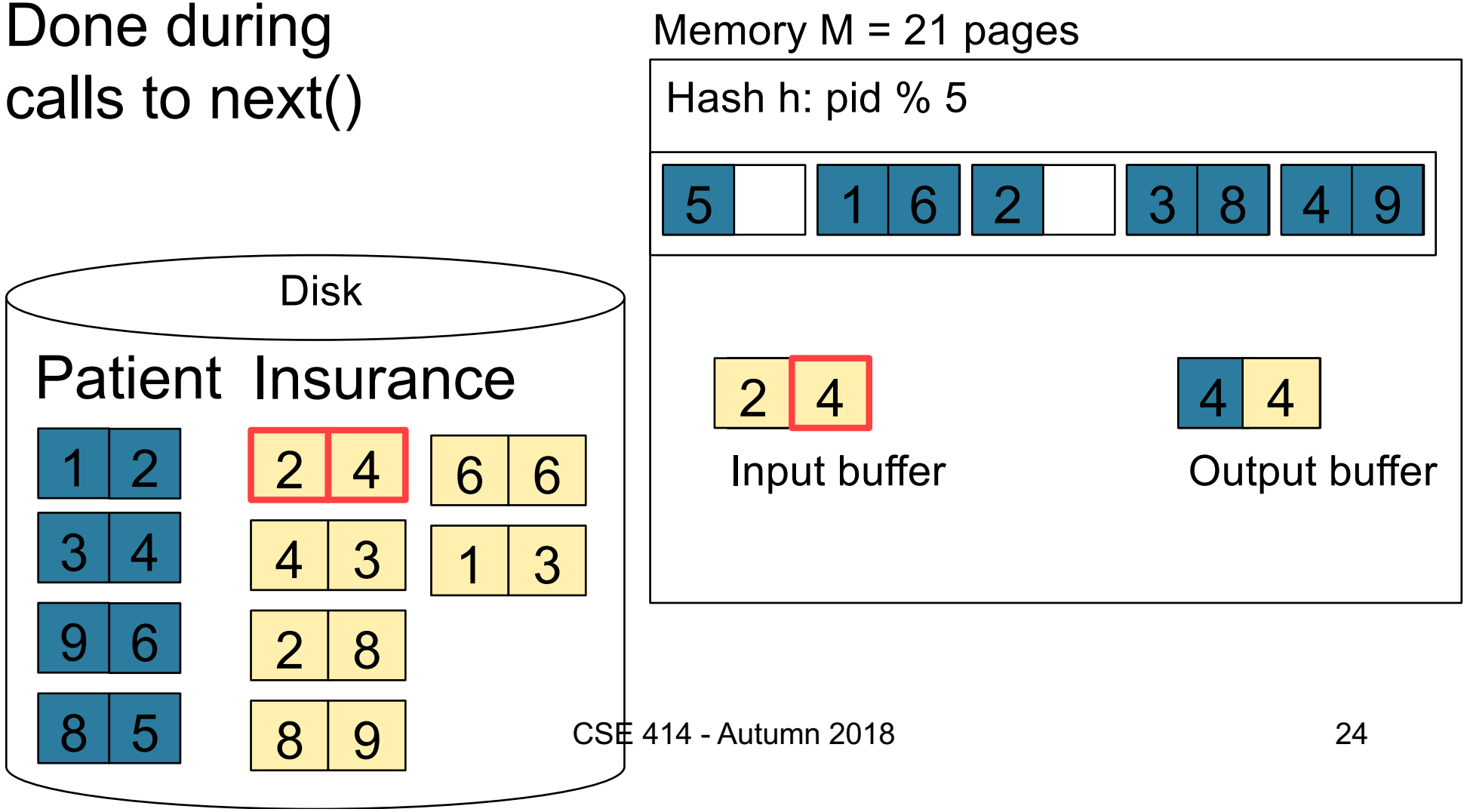
Done during calls to next()



Hash Join Example

Step 2: Scan Insurance and probe into hash table

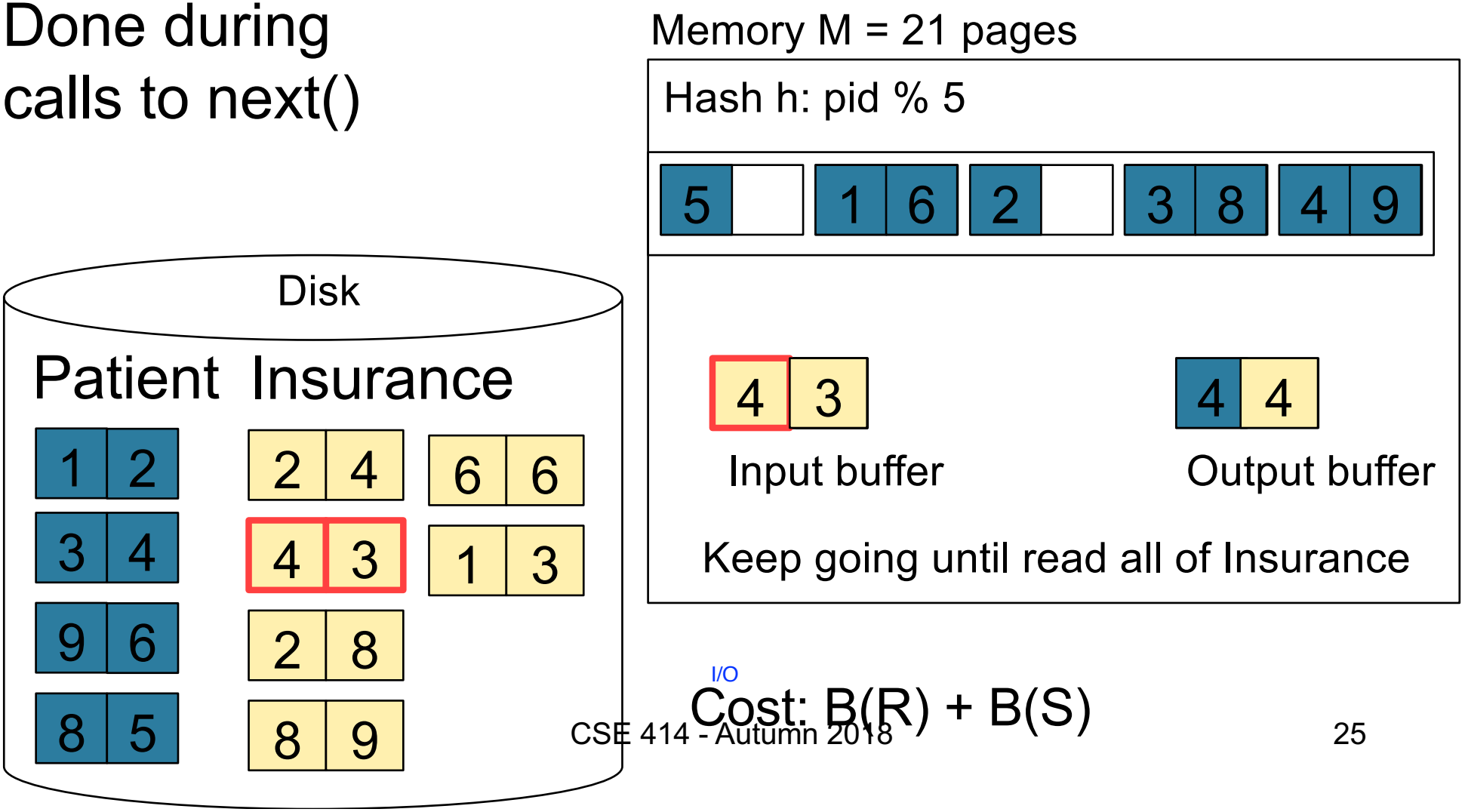
Done during
calls to next()



Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during
calls to next()



I/O 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)
```

Get a block from R
for each tuple in that block,
look through all of S to see if
we can find it in S

scanning over all of R

for each tuple in R, we might have to find all of S
(i.e. in the case there is no match)

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

What is the Cost?

- Multiple-pass since S is read many times

BUT does not require data to be written to main memory

Block-Nested loop join?

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)
```

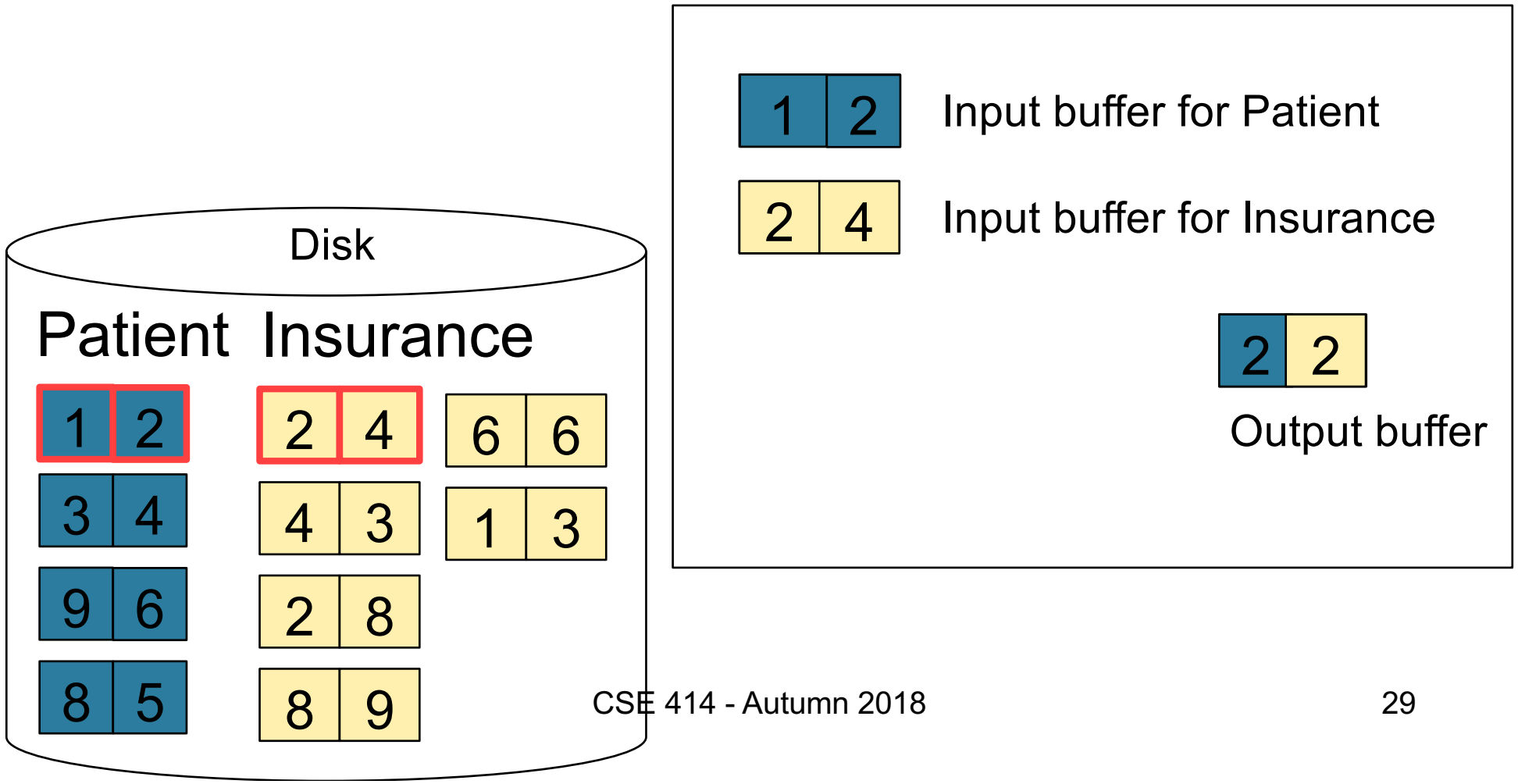
compare the tuples in blocks, rather than comparing a single tuple at a time.

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

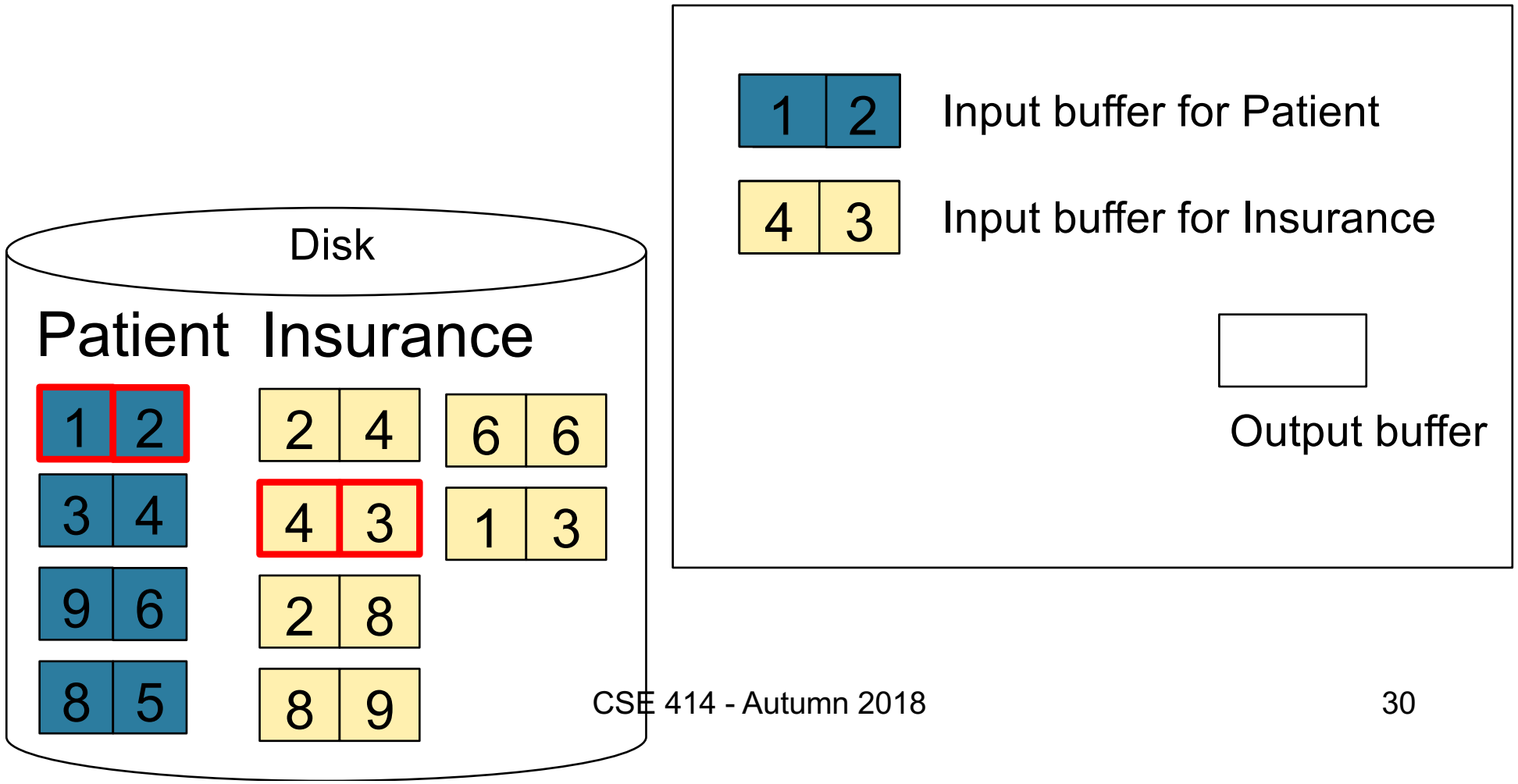
What is the Cost?

Get a block of R.
Get a block of S
compare all these tuples pairwise between the two blocks and search for a match

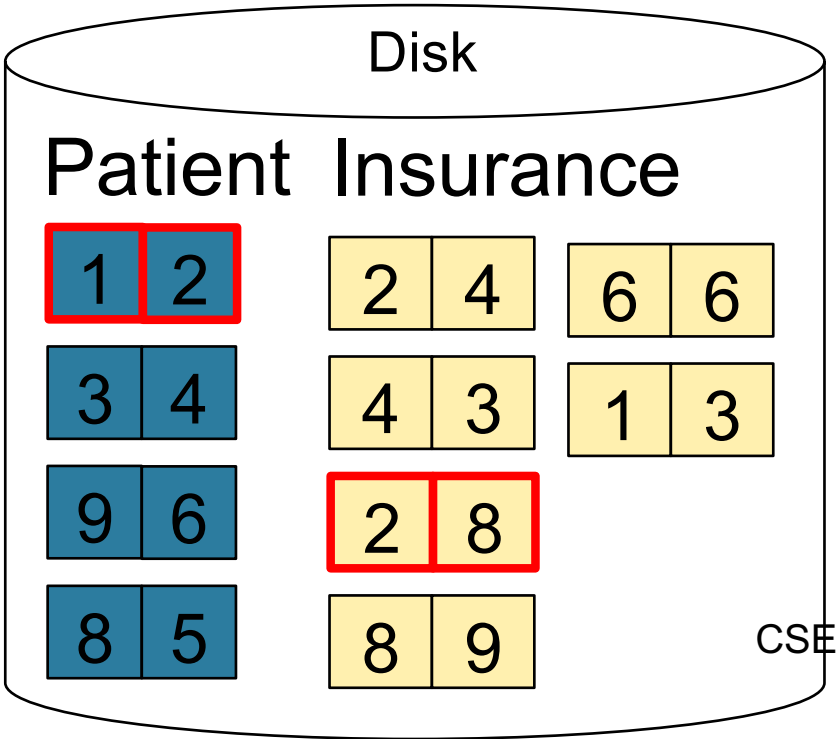
Page-at-a-time Refinement

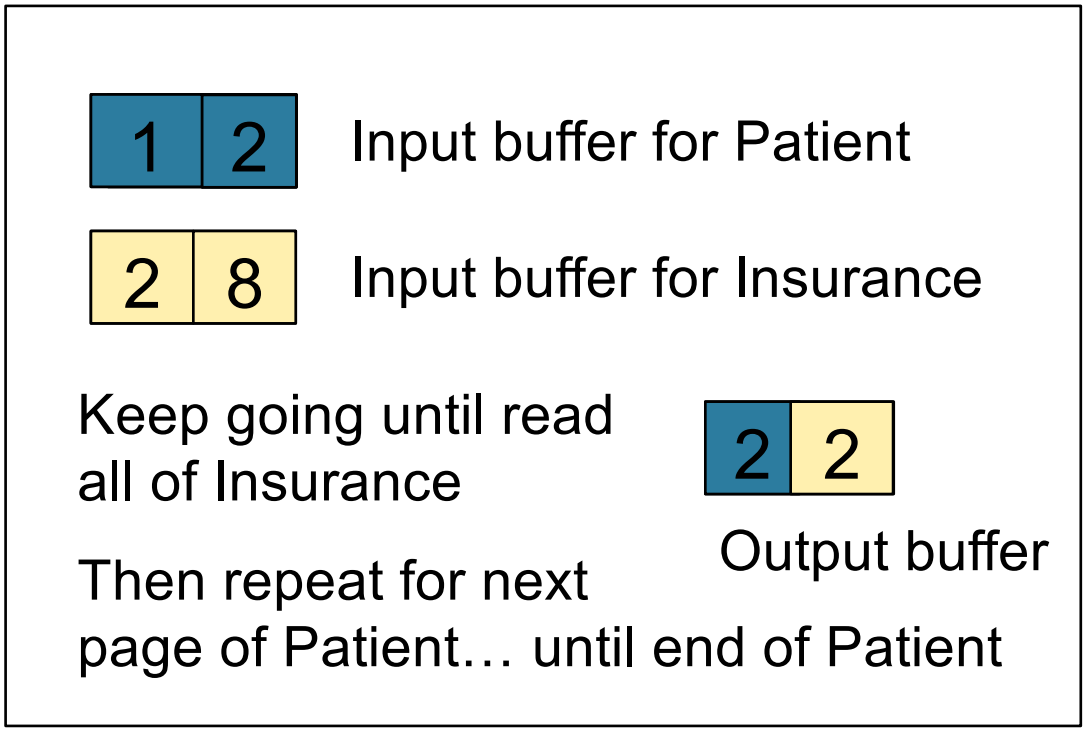


Page-at-a-time Refinement



Page-at-a-time Refinement





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

NOT ON FINAL

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?

NOT ON FINAL

Sort-Merge Join

Sort-merge join: R ⋈ S

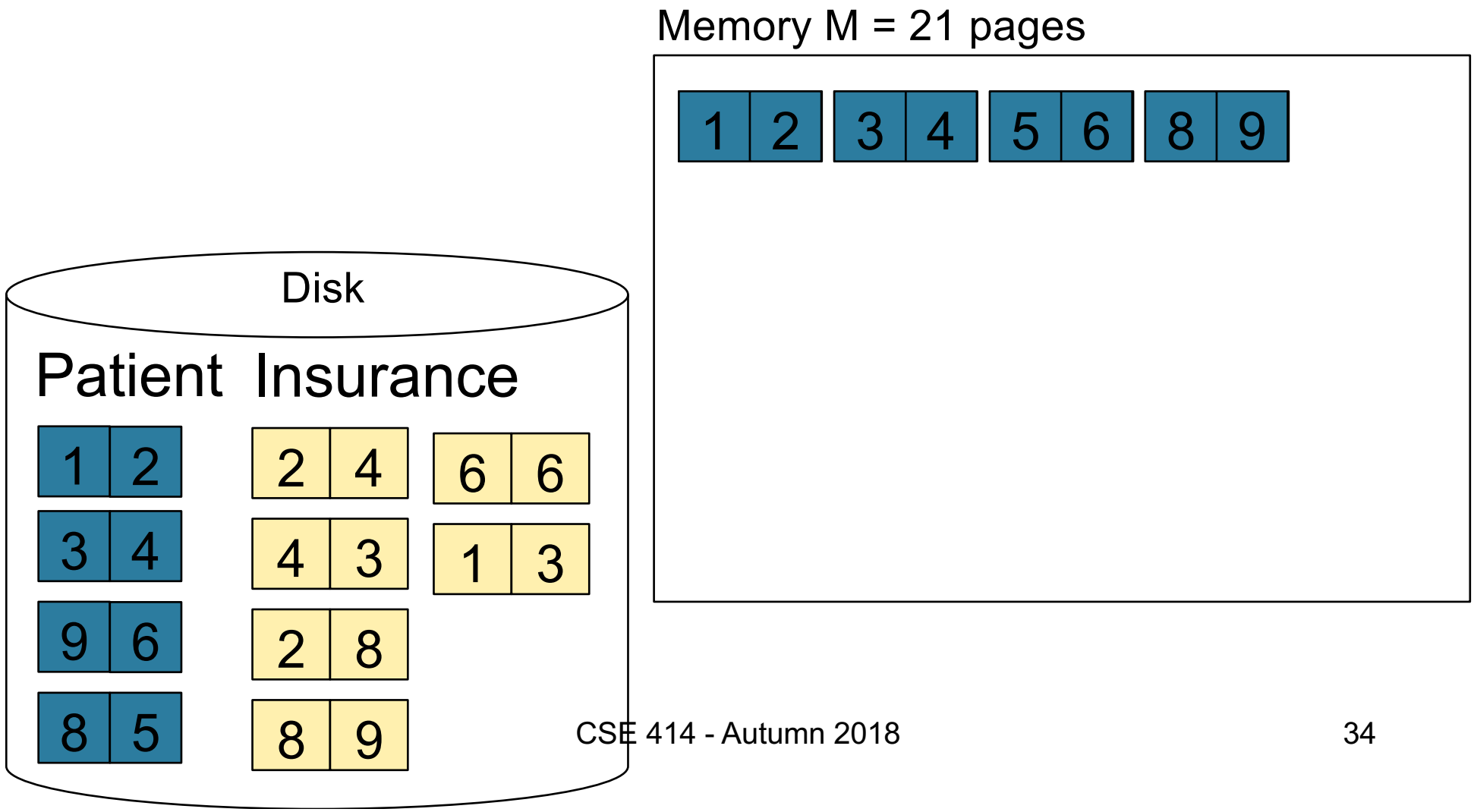
- Scan R and sort in main memory
- Scan S and sort in main memory
- Merge R and S

- Cost: B(R) + B(S)
- One pass algorithm when B(S) + B(R) <= M
- Typically, this is NOT a one pass algorithm

NOT ON FINAL

Sort-Merge Join Example

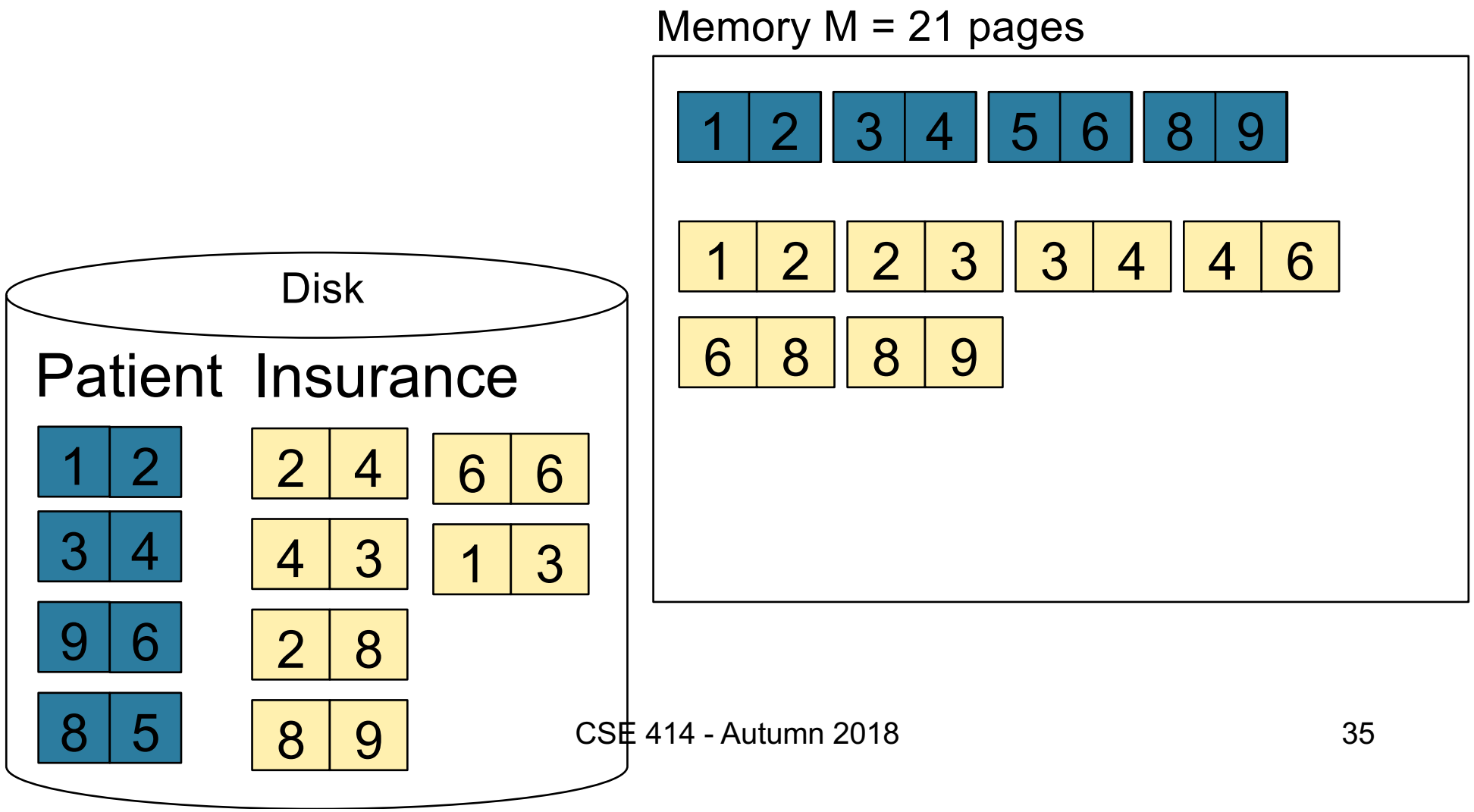
Step 1: Scan Patient and sort in memory



NOT ON FINAL

Sort-Merge Join Example

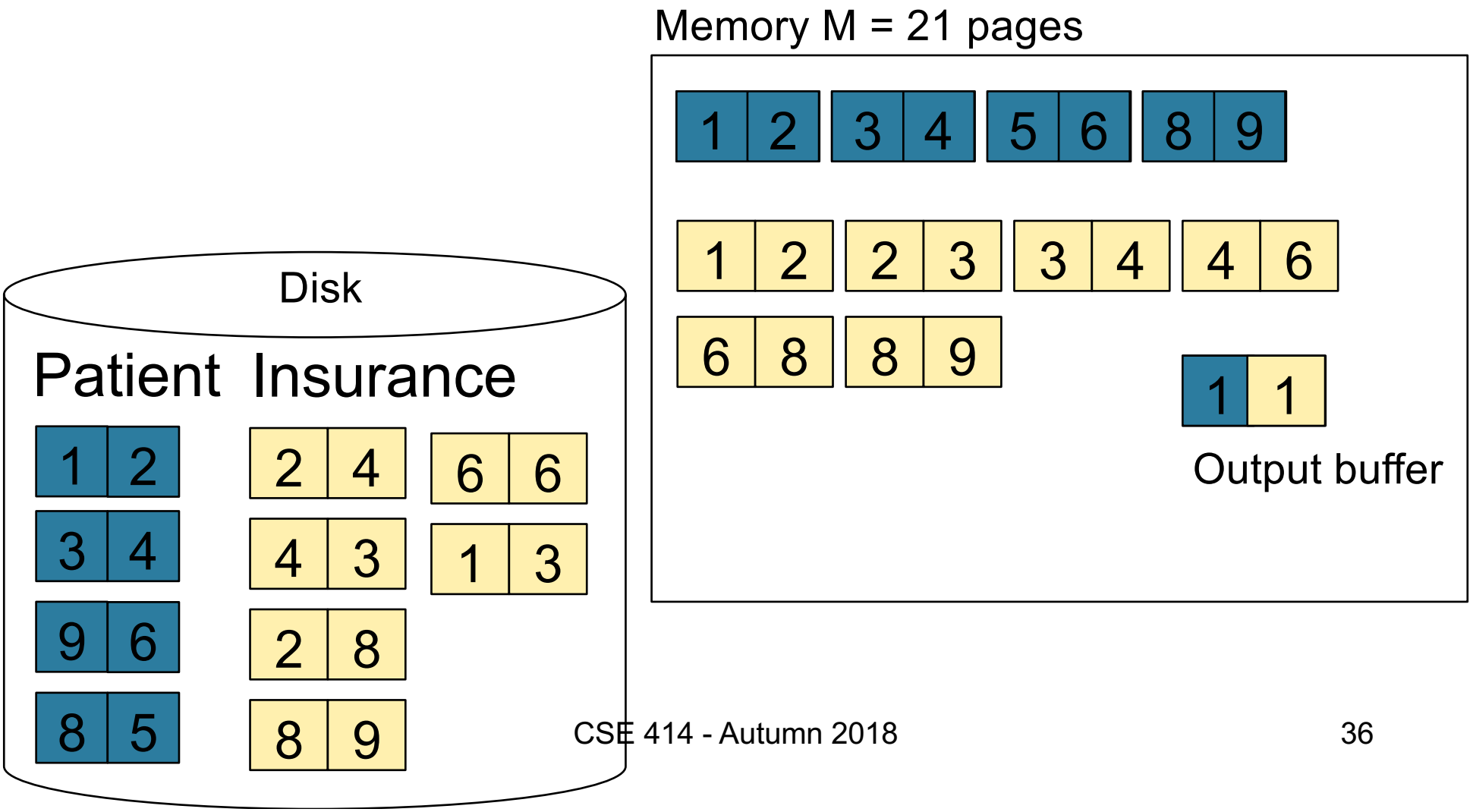
Step 2: Scan Insurance and sort in memory



NOT ON FINAL

Sort-Merge Join Example

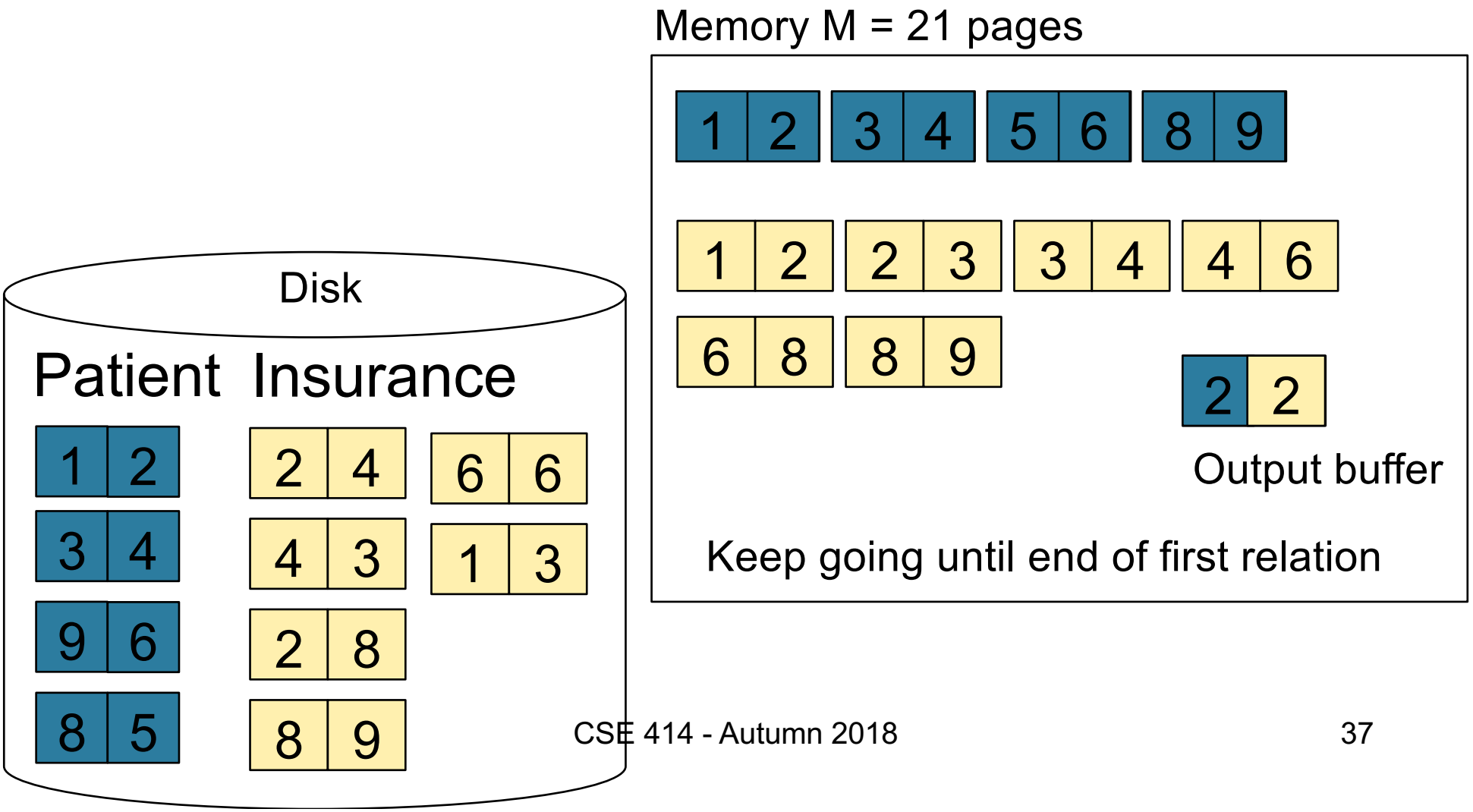
Step 3: Merge Patient and Insurance



NOT ON FINAL

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

since we have an index we can quickly find where given tuples in S are located

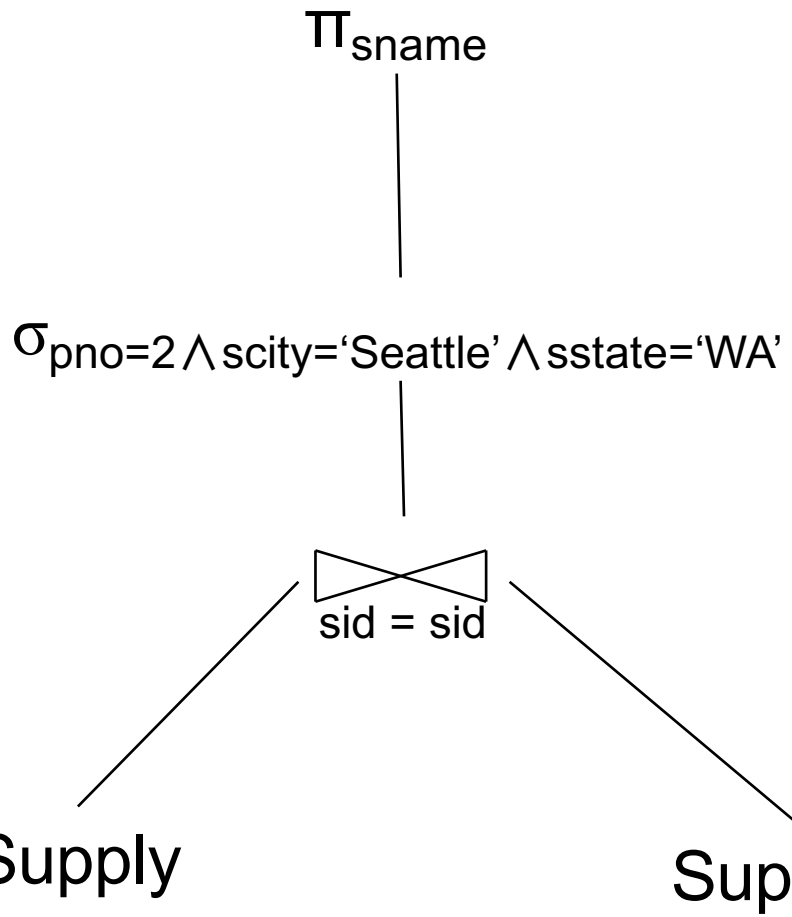
- Cost:
  - If index on S is clustered:
    B(R) + T(R) * (B(S) * 1/V(S,a))
  - If index on S is unclustered:
    B(R) + T(R) * (T(S) * 1/V(S,a))

Nov 5, 4:14pm
gives a detailed explanation of how these formulas are derived

Cost of Query Plans

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'
```

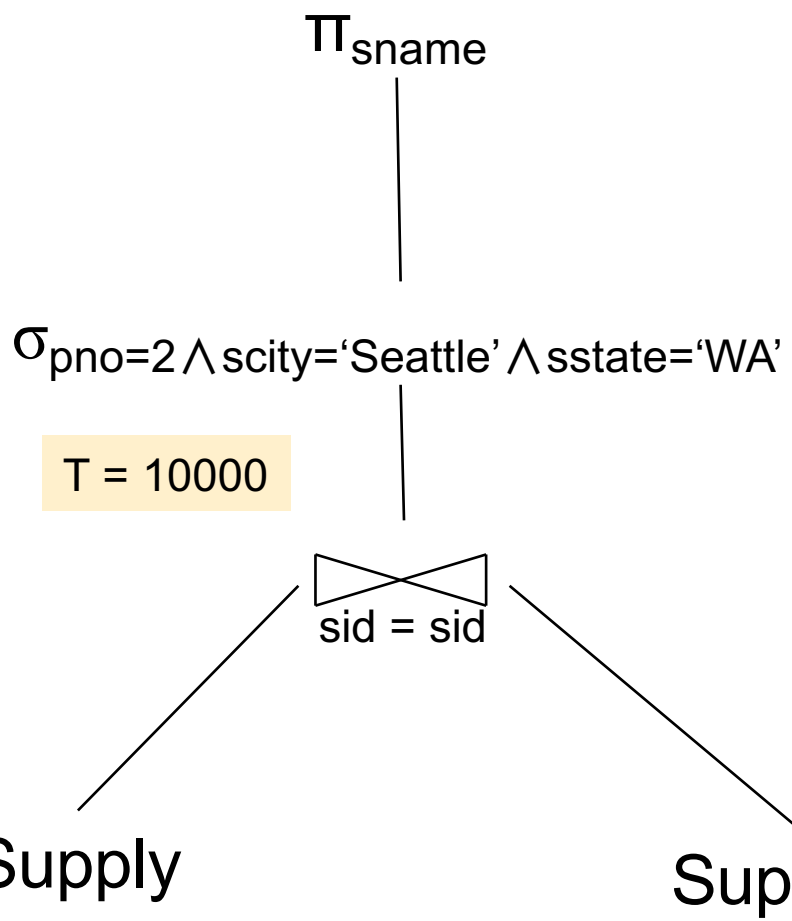
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

M=11

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'
```

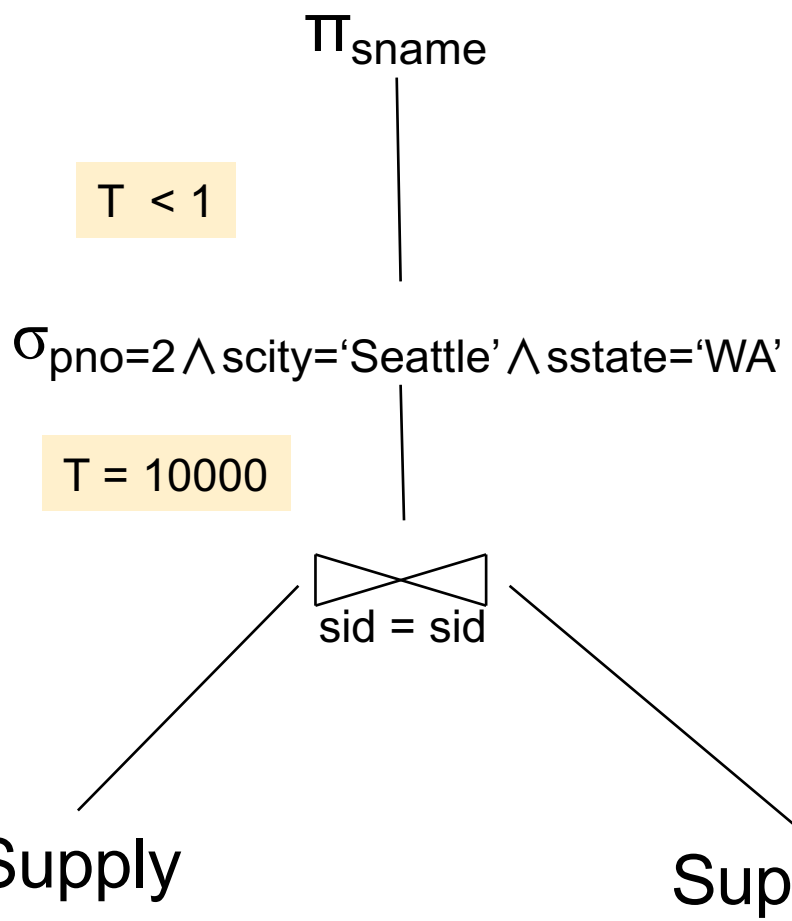
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

M=11

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'
```

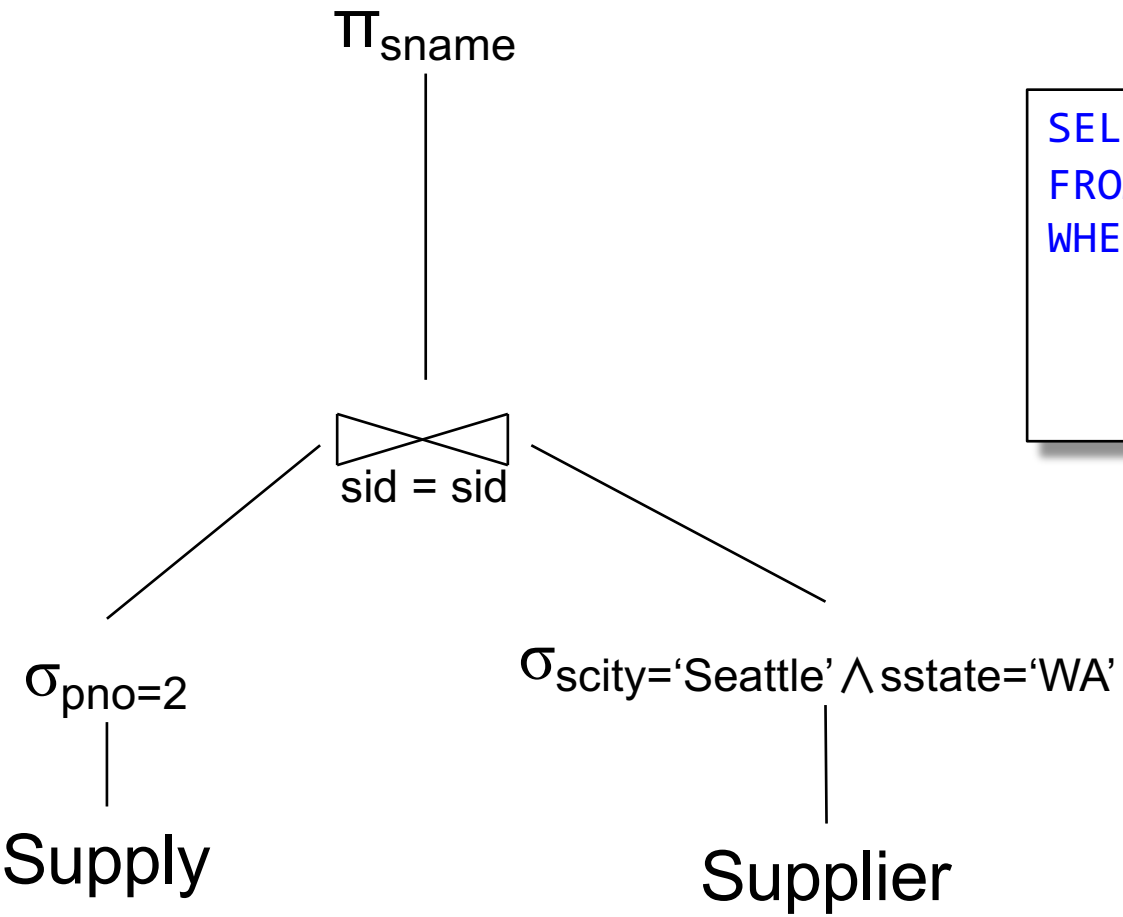
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

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

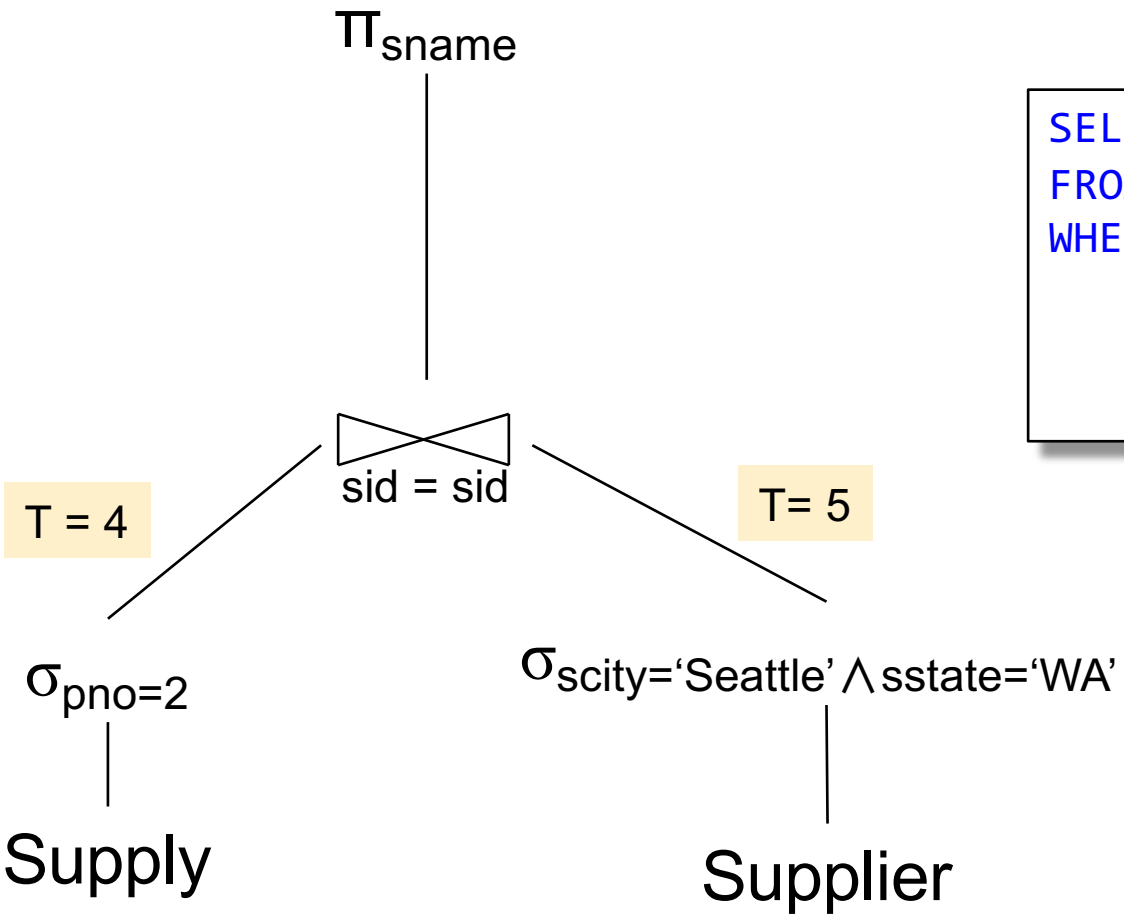
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

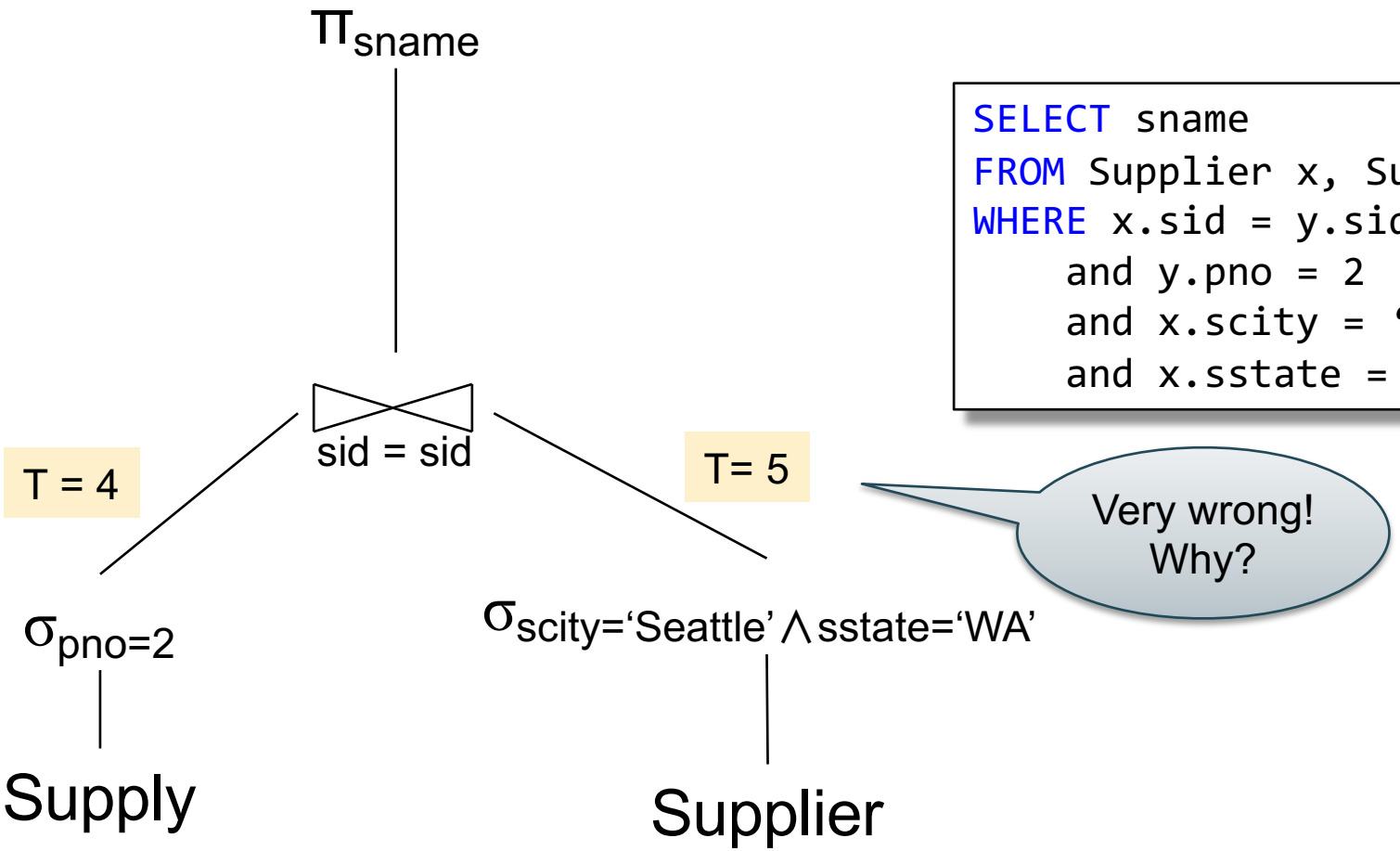
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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'
```

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

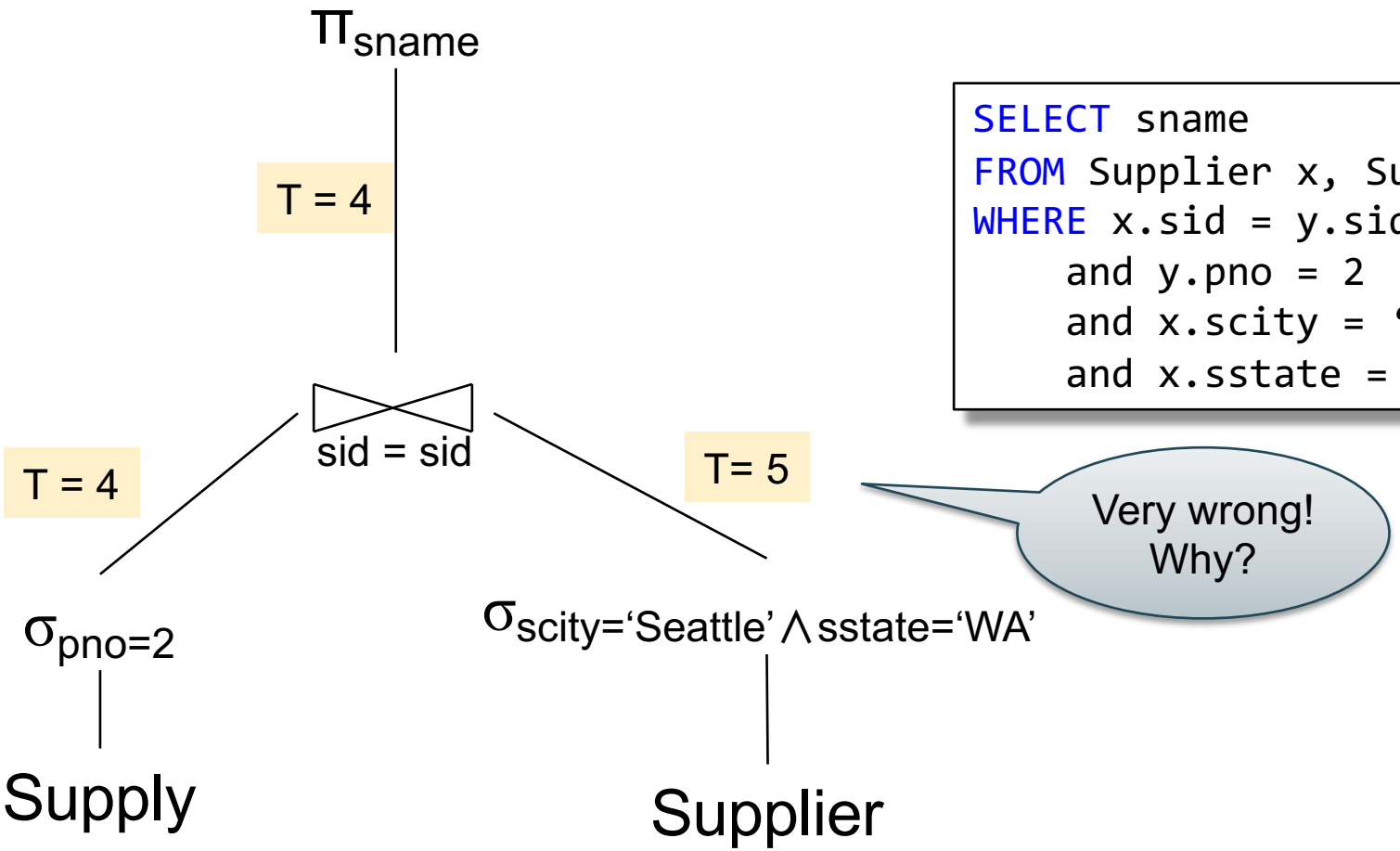
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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?

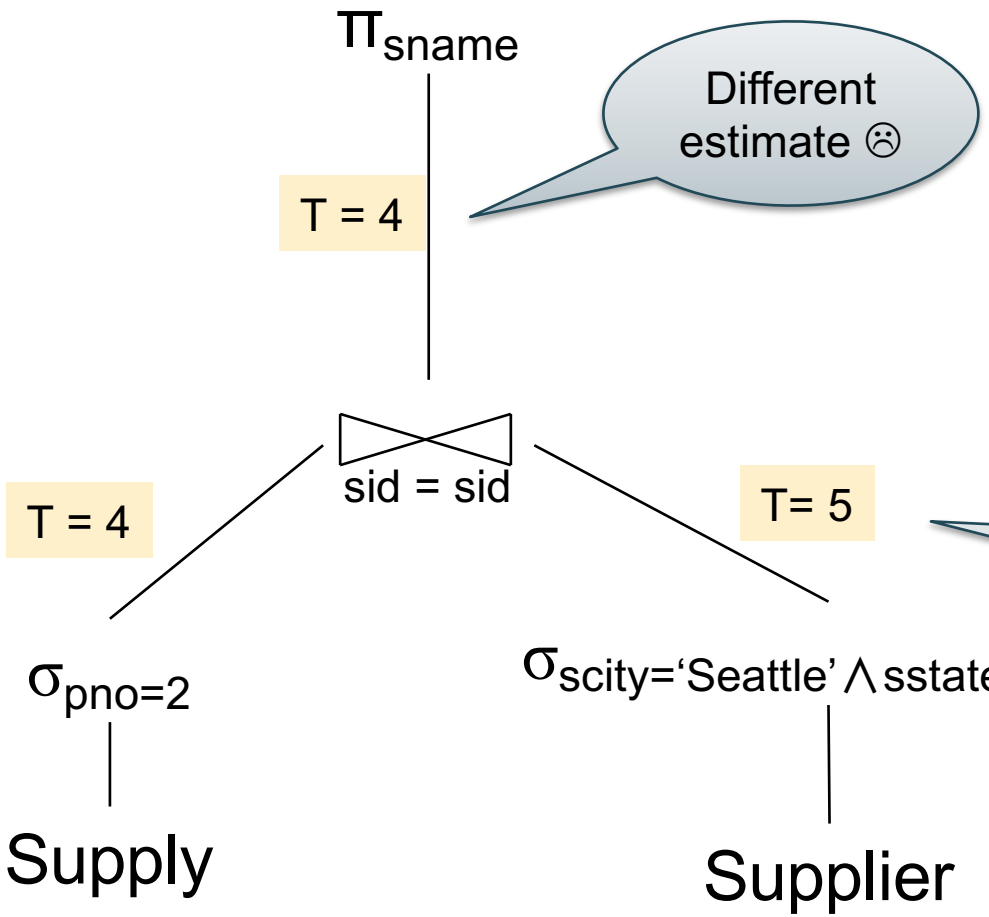
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

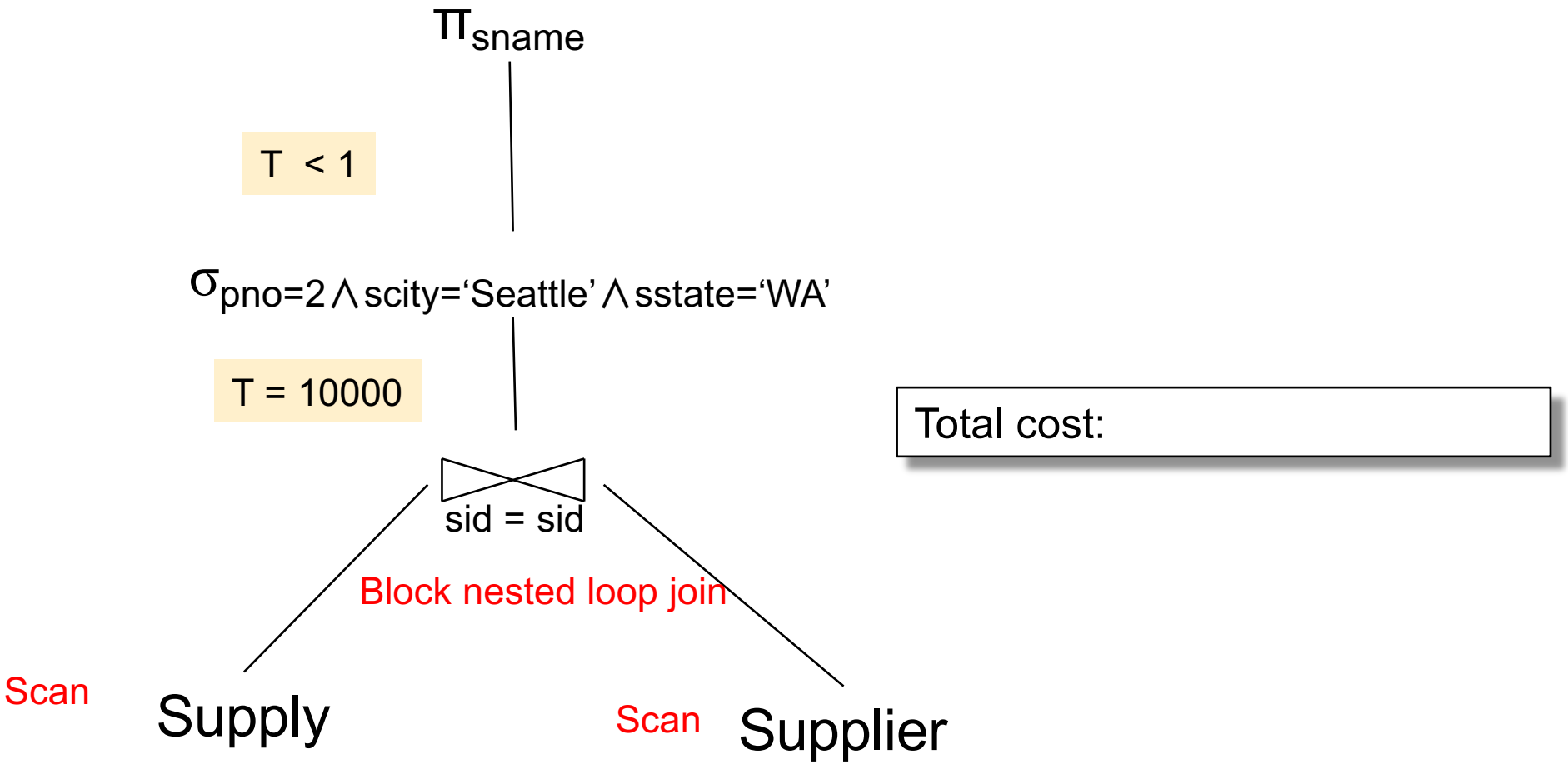
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Plan 1



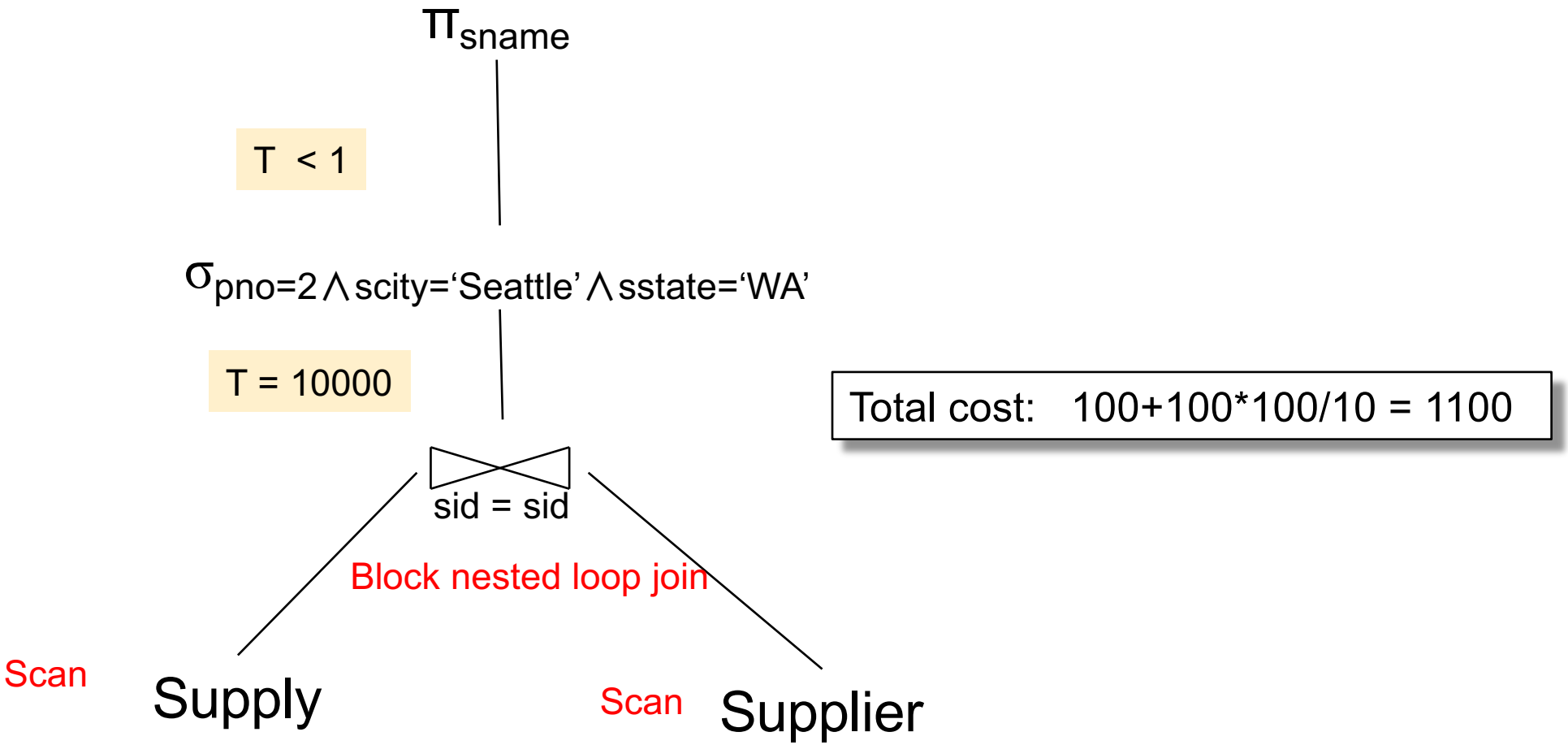
Total cost:

M=11

Supplier(sid, sname, scity, sstate)

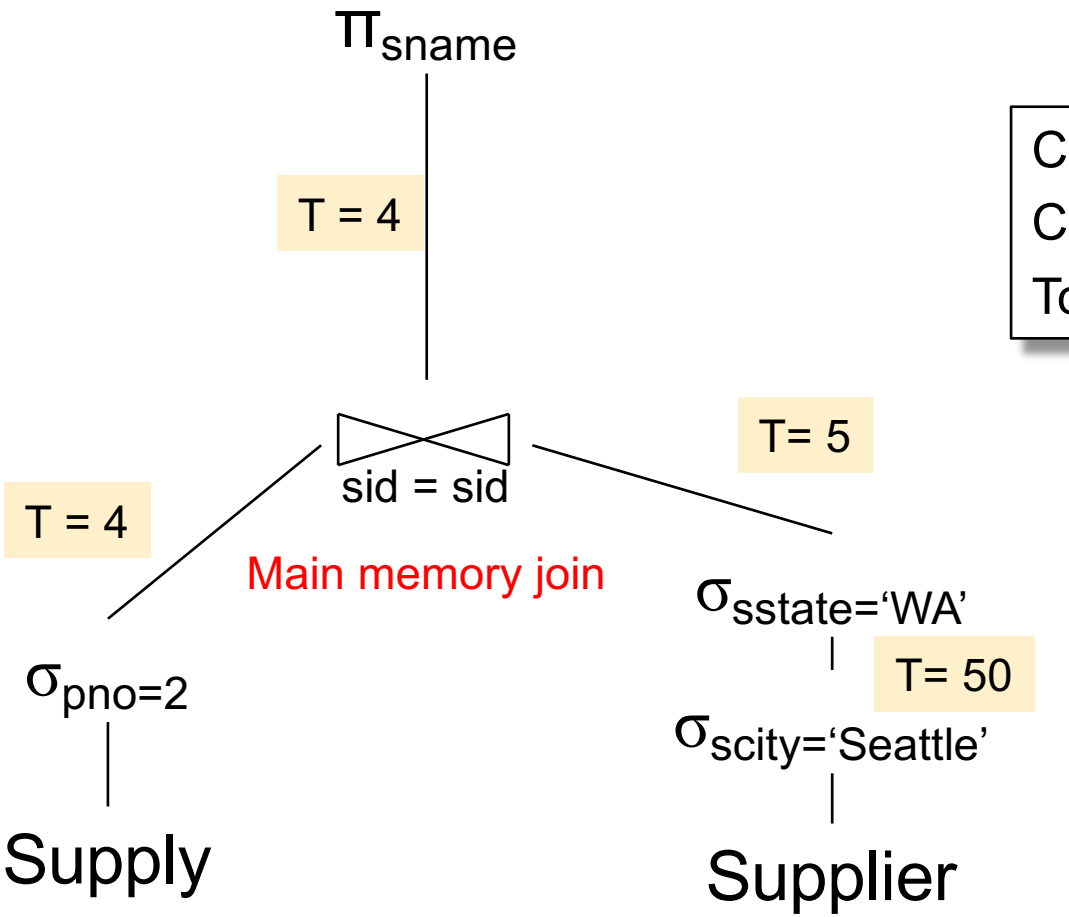
Supply(sid, pno, quantity)

Physical Plan 1



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

Physical Plan 2



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

Unclustered index lookup Supply(pno)

Unclustered index lookup Supplier(scity)

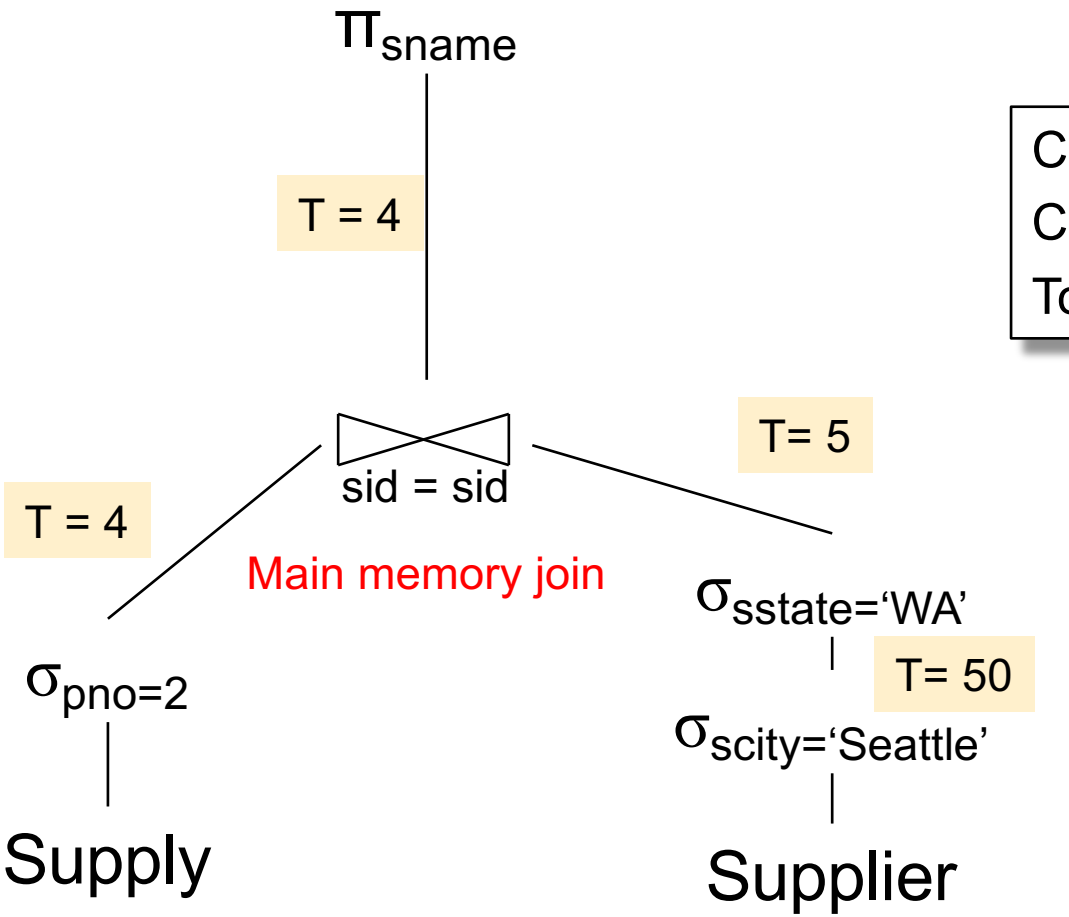
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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:

Unclustered index lookup Supply(pno)

Unclustered index lookup Supplier(scity)

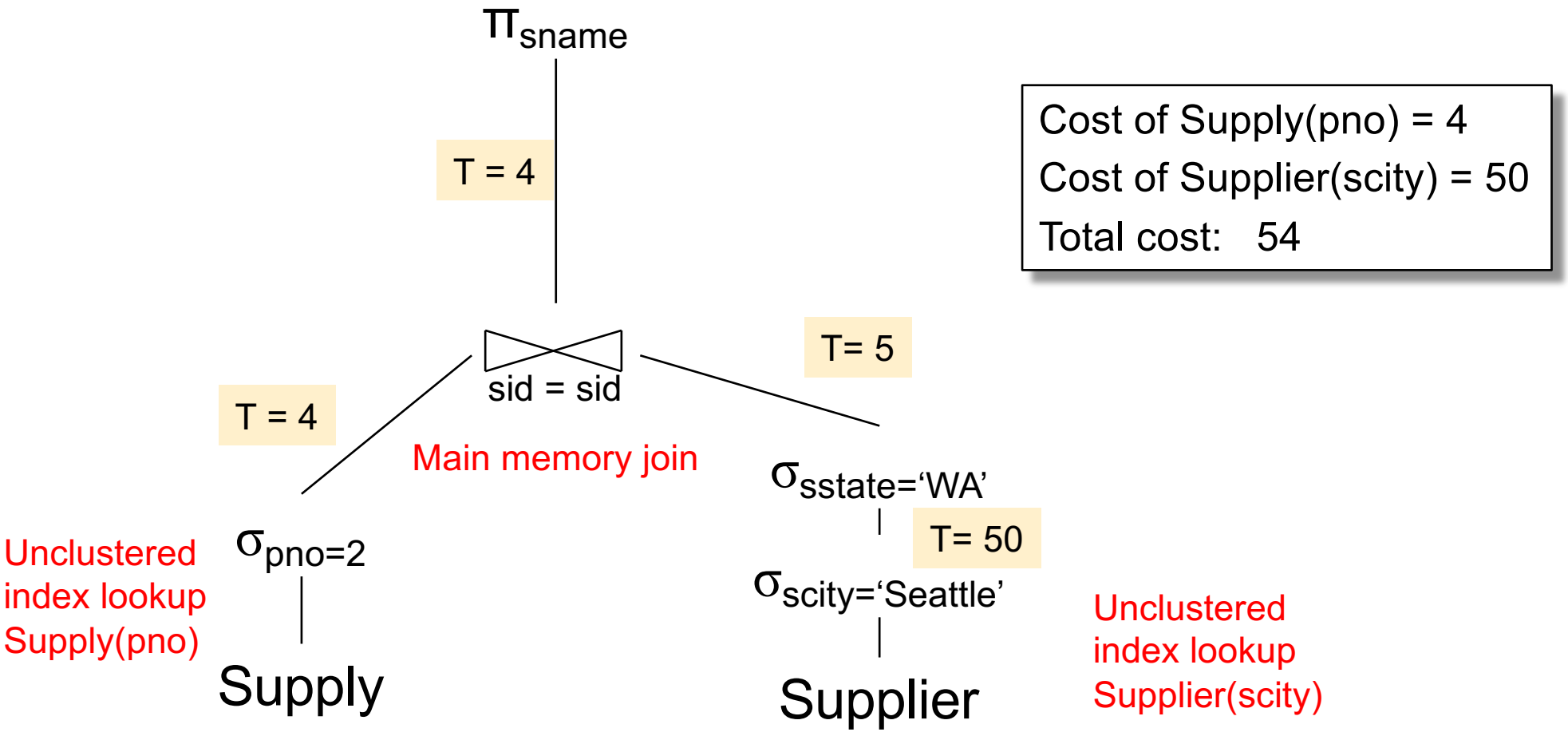
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

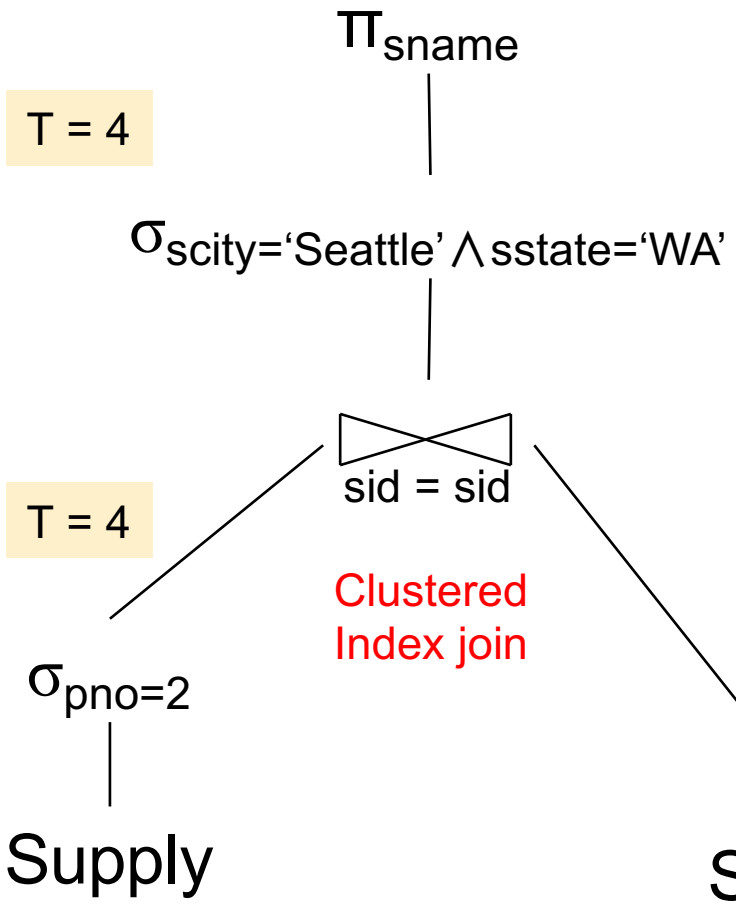
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

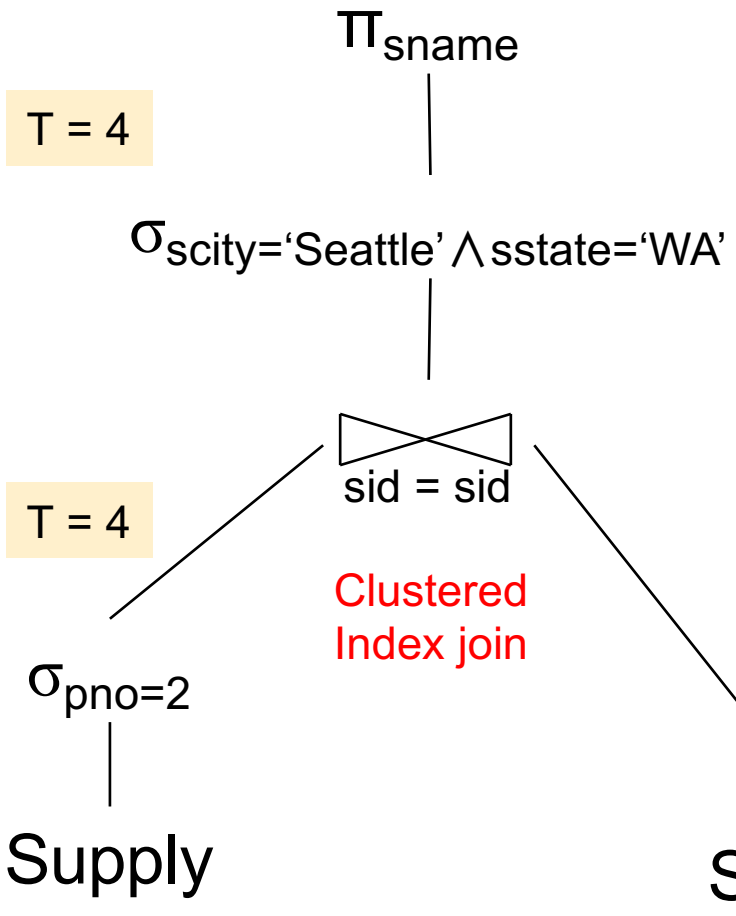
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

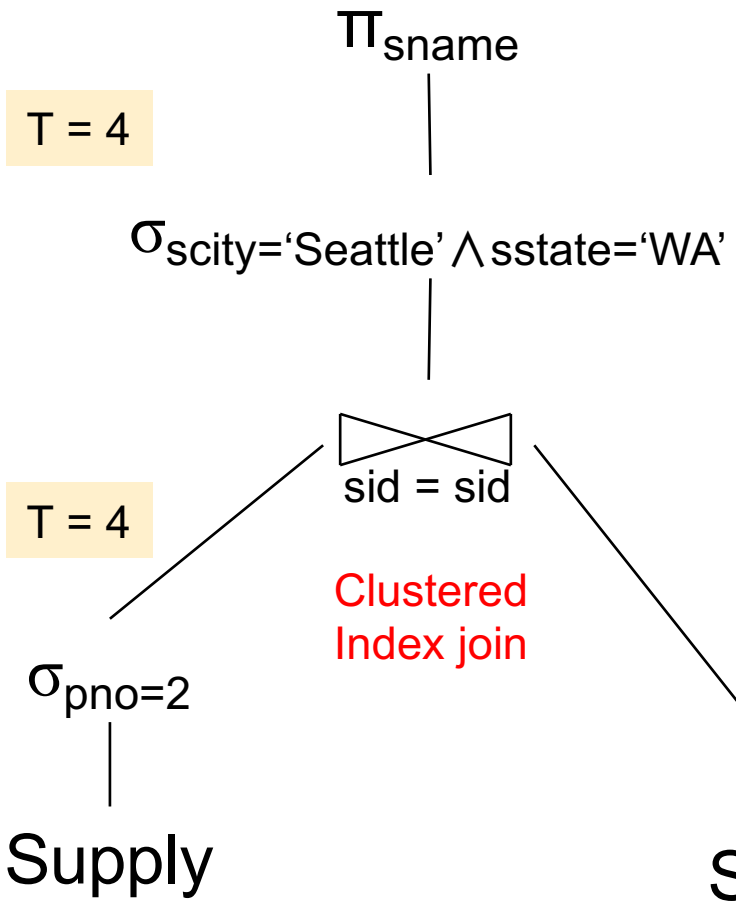
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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