Introduction to Database Systems
CSE 414

Lecture 27: More Operator Costs

Announcements

- HW8 and WQ7 both due tonight!
- Please fill out course evals online!
- Last lecture on Friday

Final Exam

- Thursday 6/7, 2:30-4:20pm
- Location: here
- Can bring 2 letter-size sheets of notes
  - Handwritten or printed
- More info on course website

- Review session:
  - Sunday 6/3, 2:30-5pm, SMI 102

Big Picture

- How to choose the “best” query plan to run? (aka query optimization)
- To answer this question we need to understand:
  - Data organization on the disk
  - Index structures and how they are used in queries
  - A way to model query “costs”
  - Compute cost for each query operator
  - Compute cost for each physical plan

Last topics this quarter!

Big Picture

Why do we care about all these internal details?

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

Join Algorithms

- Nested loop join (short review)
- Hash join
- Sort-merge join

Nested Loop Joins (review)

R		
a	b	c
1	7	4
…	…	…
98	3	2

S		
c	d	e
3	43	7
…	…	…
9	24	9

…

S		
c	d	e
3	43	7
…	…	…
9	24	9

Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- 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 $R \bowtie S$
- 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?

- Cost: $B(R) + T(R)\, B(S)$
- Multiple-pass since S is read many times

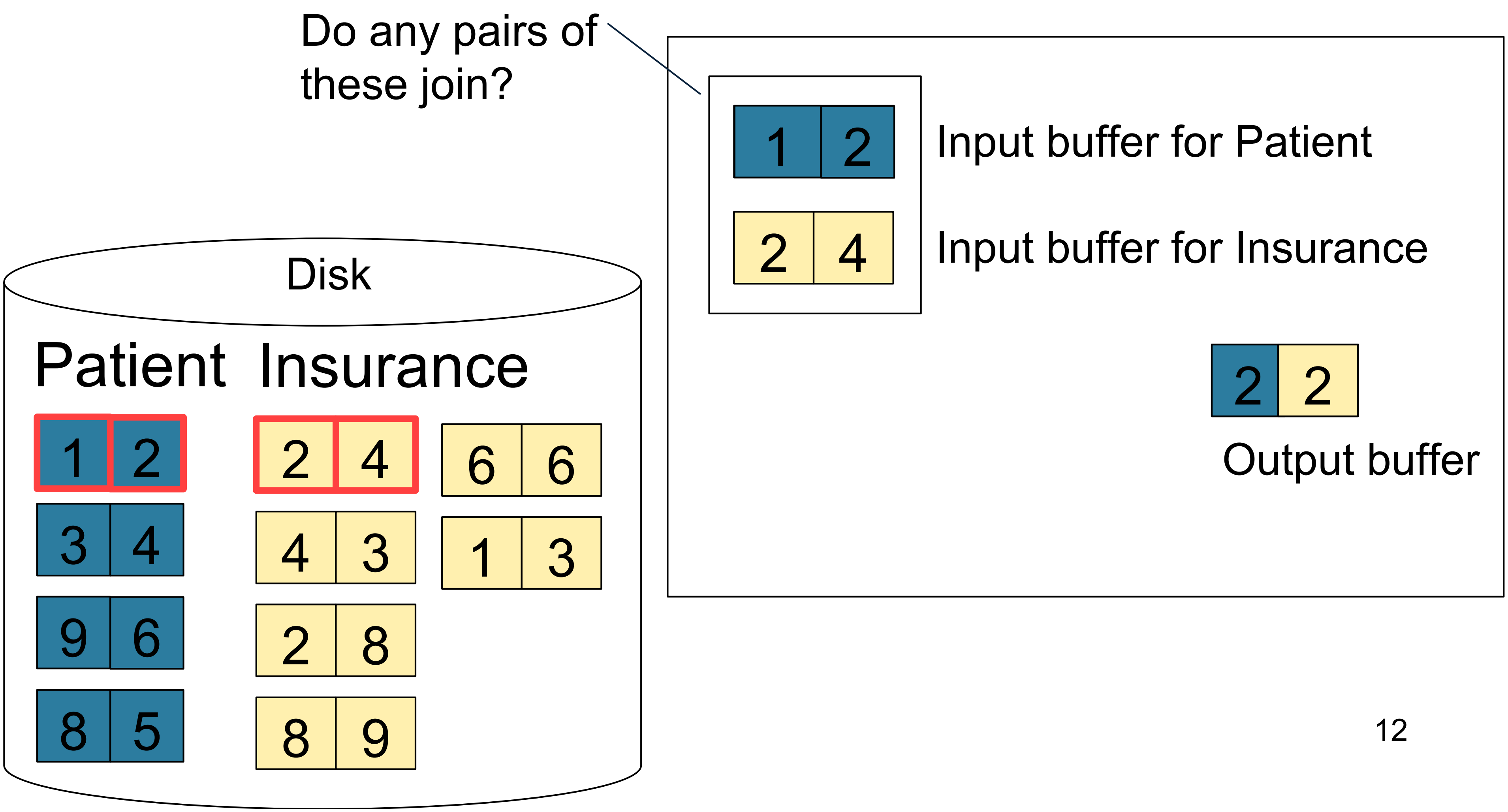
Page-at-a-time Refinement

```
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)
```

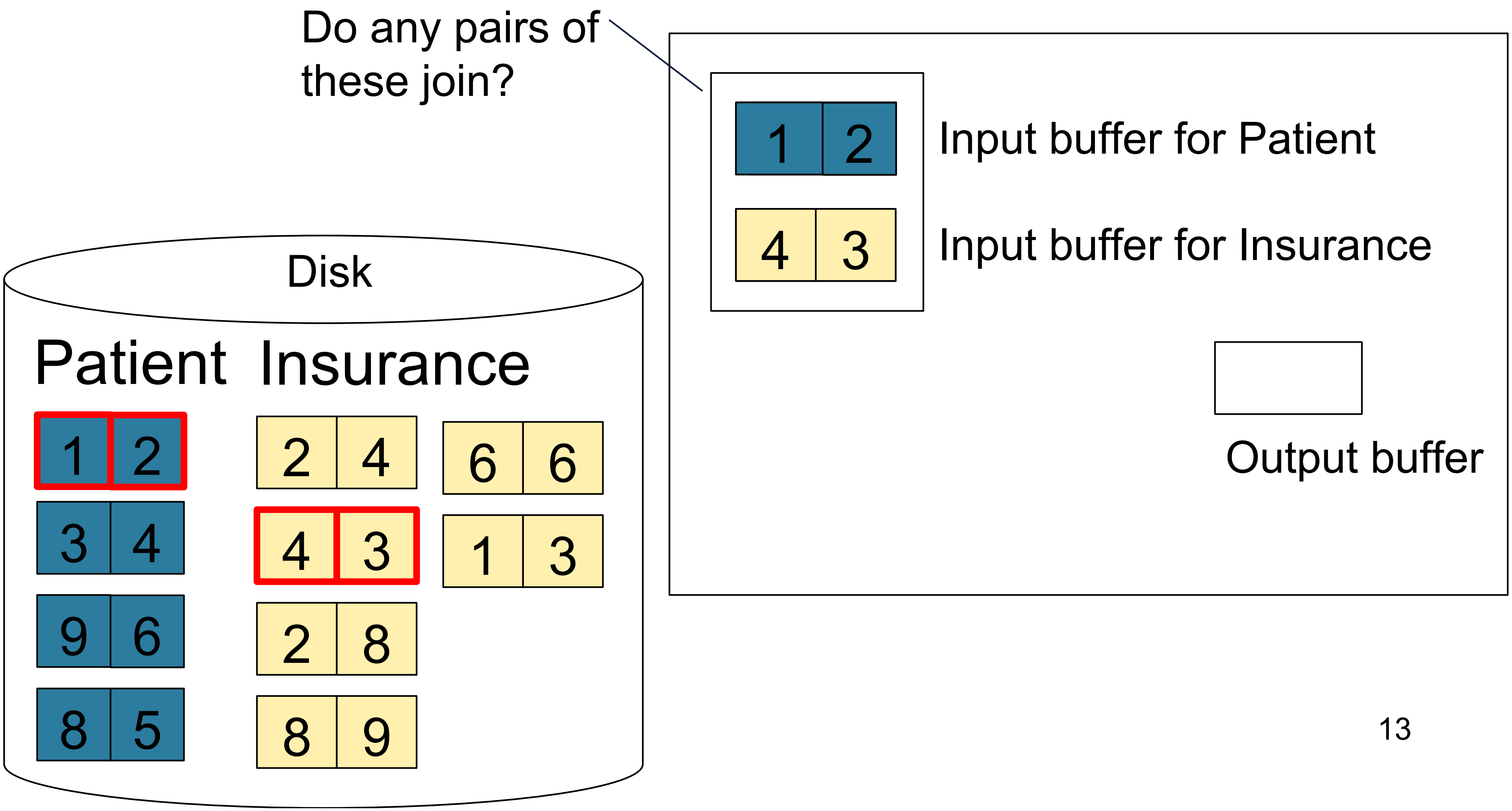
- Cost: B(R) + B(R)B(S)

What is the Cost?

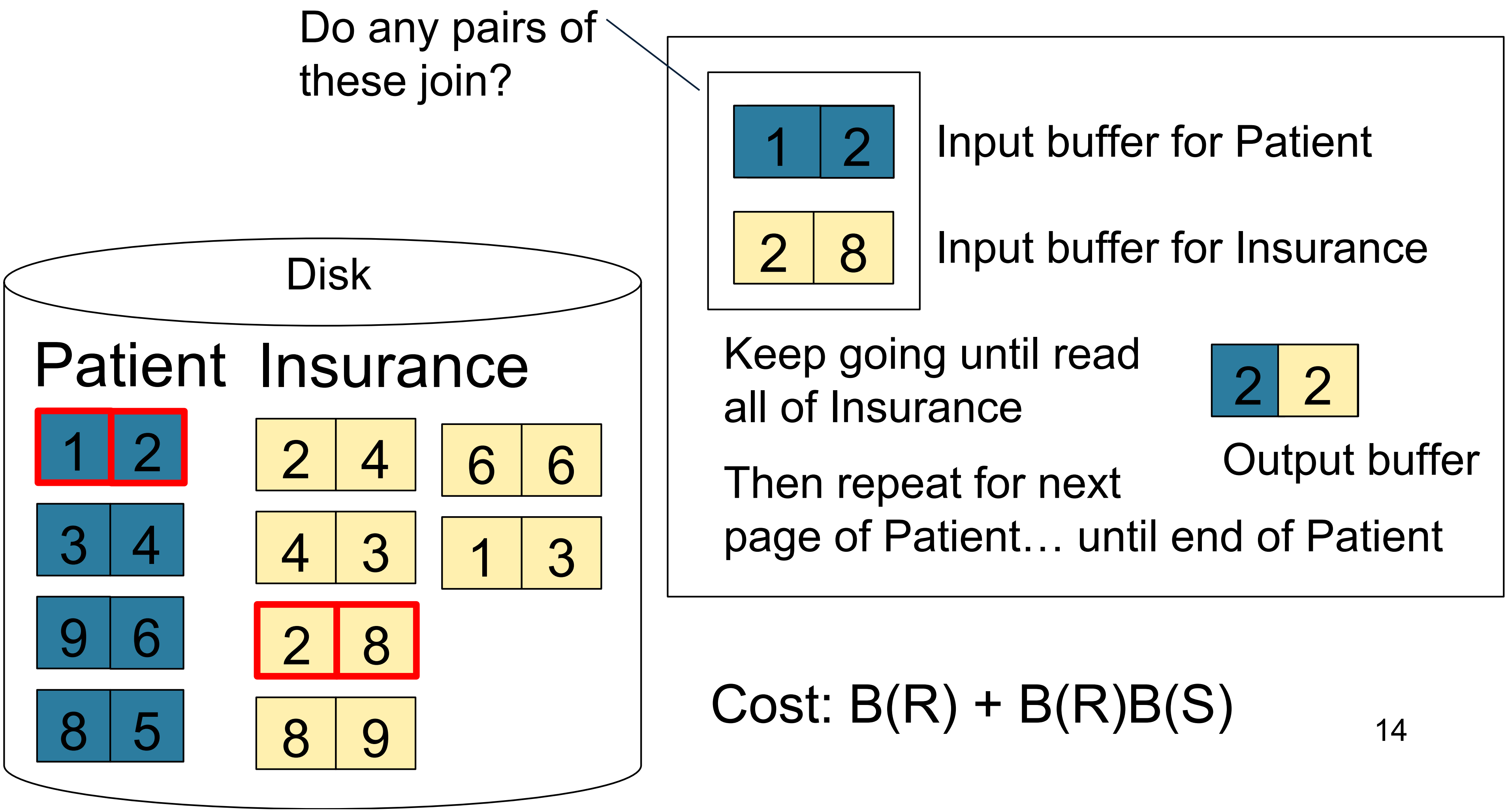
Page-at-a-time Refinement



Page-at-a-time Refinement



Page-at-a-time Refinement



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

S		
c	d	e
3	43	7
...	...	...
9	24	9

R		
a	b	c
1	7	4
…	...	...
98	3	2

Hash Join

Hash Join

Hash join: $R \bowtie S$

- Scan R, build hash table 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 $B(R) \leq M$
  - 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

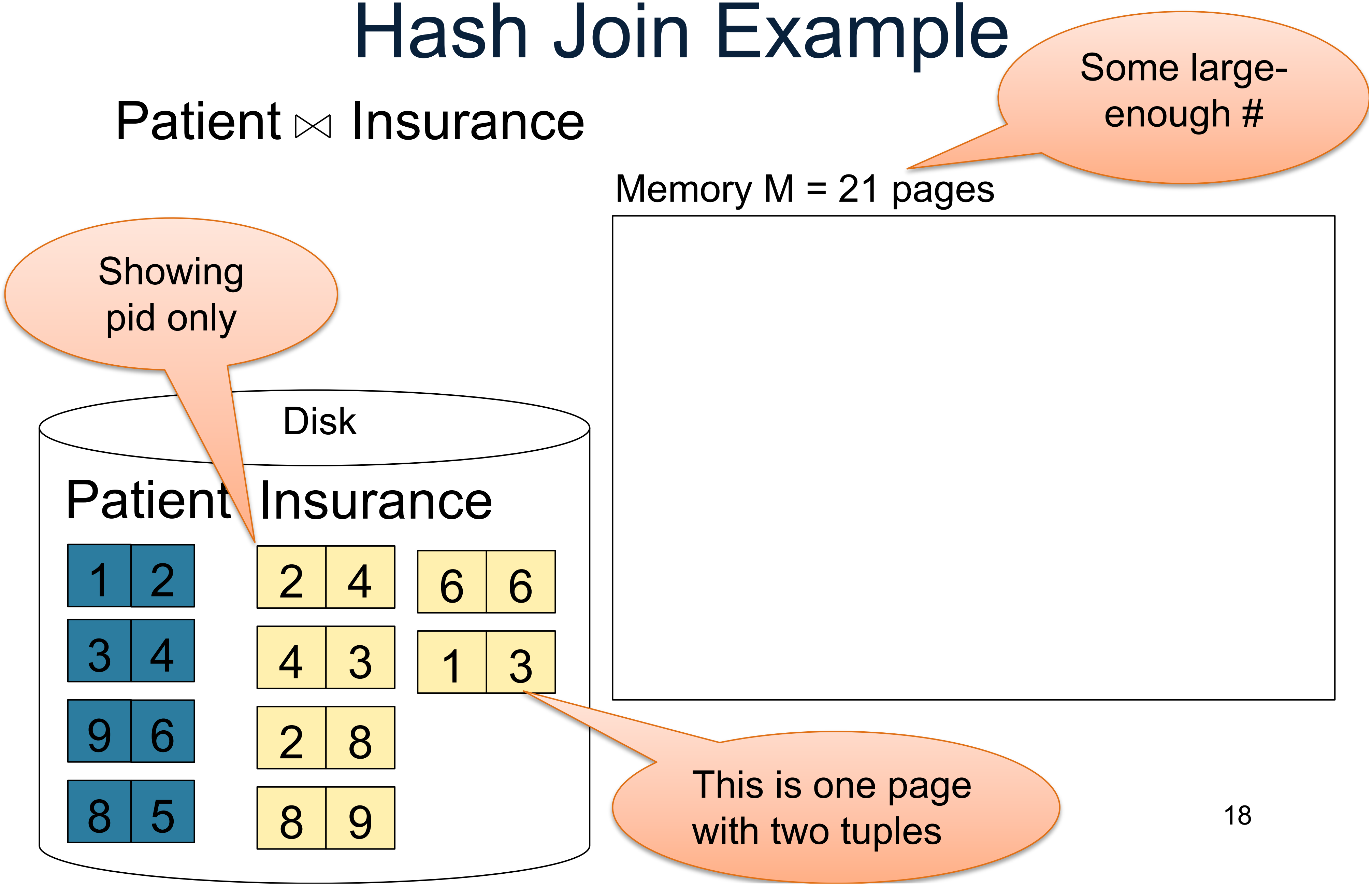
1	'Bob'	'Seattle'
2	'Ela'	'Everett'
3	'Jill'	'Kent'
4	'Joe'	'Seattle'

Insurance

2	'Blue'	123
4	'Prem'	432
4	'Prem'	343
3	'GrpH'	554

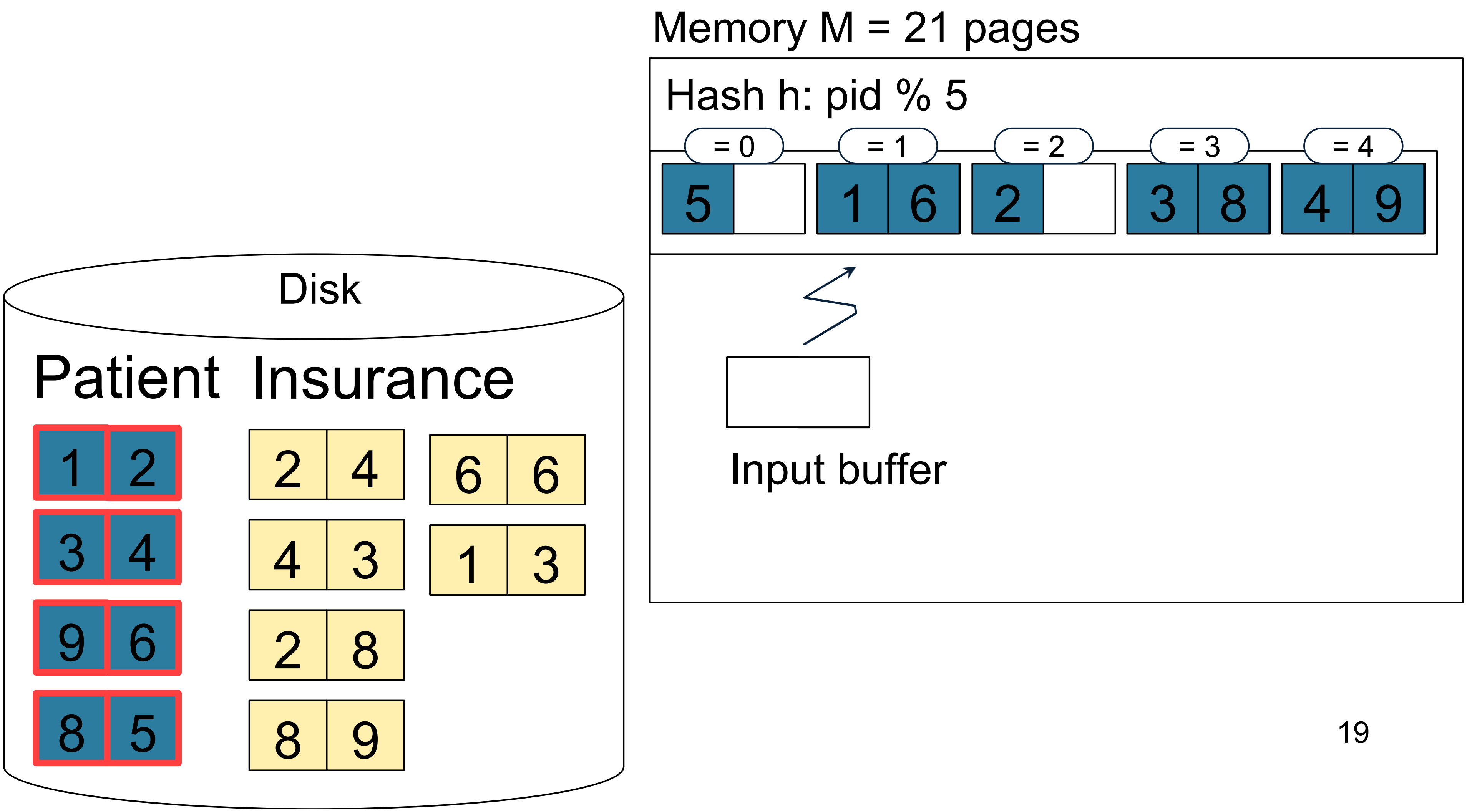
Hash Join Example

Patient $\bowtie$ Insurance



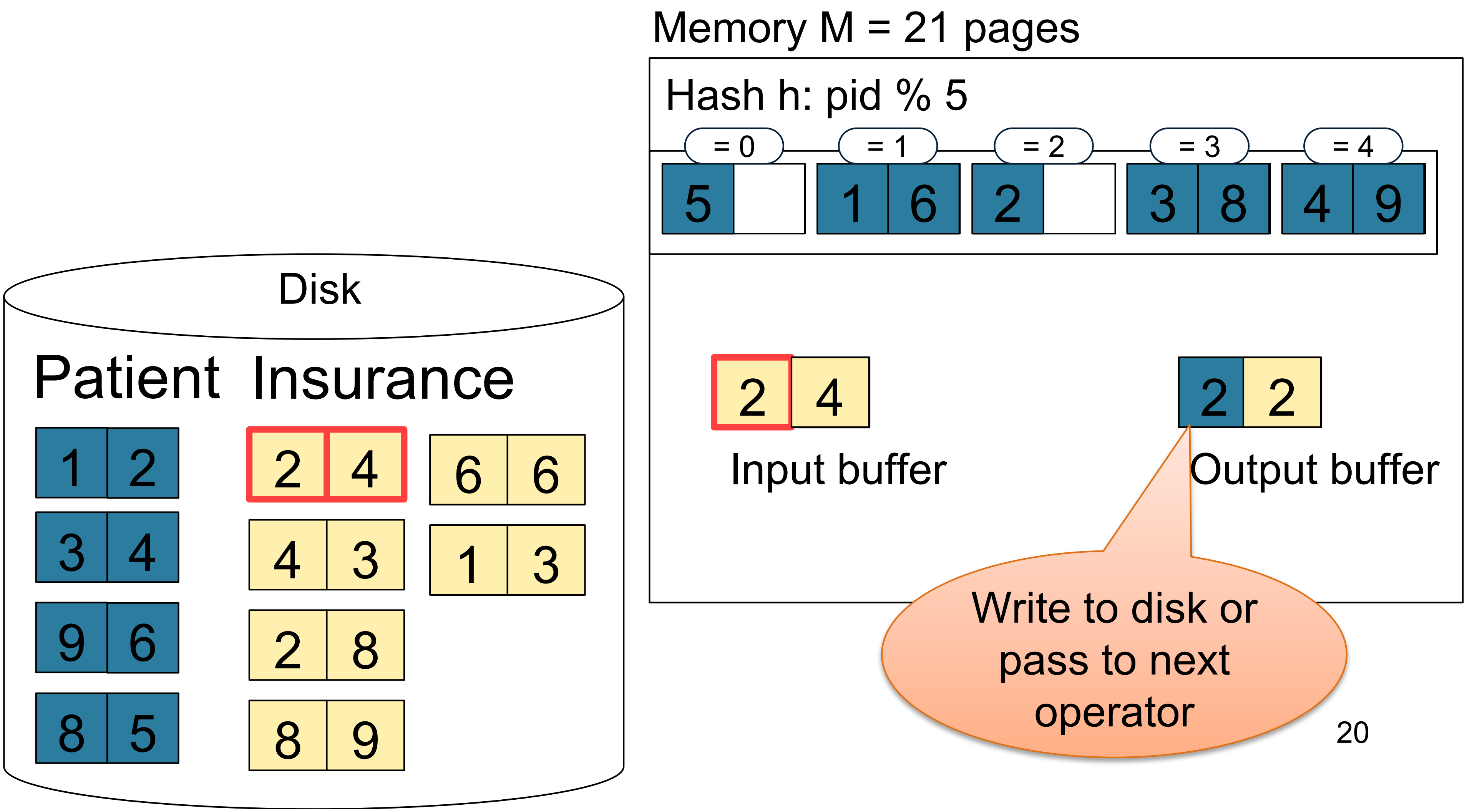
Hash Join Example

Step 1: Scan Patient and build hash table in memory



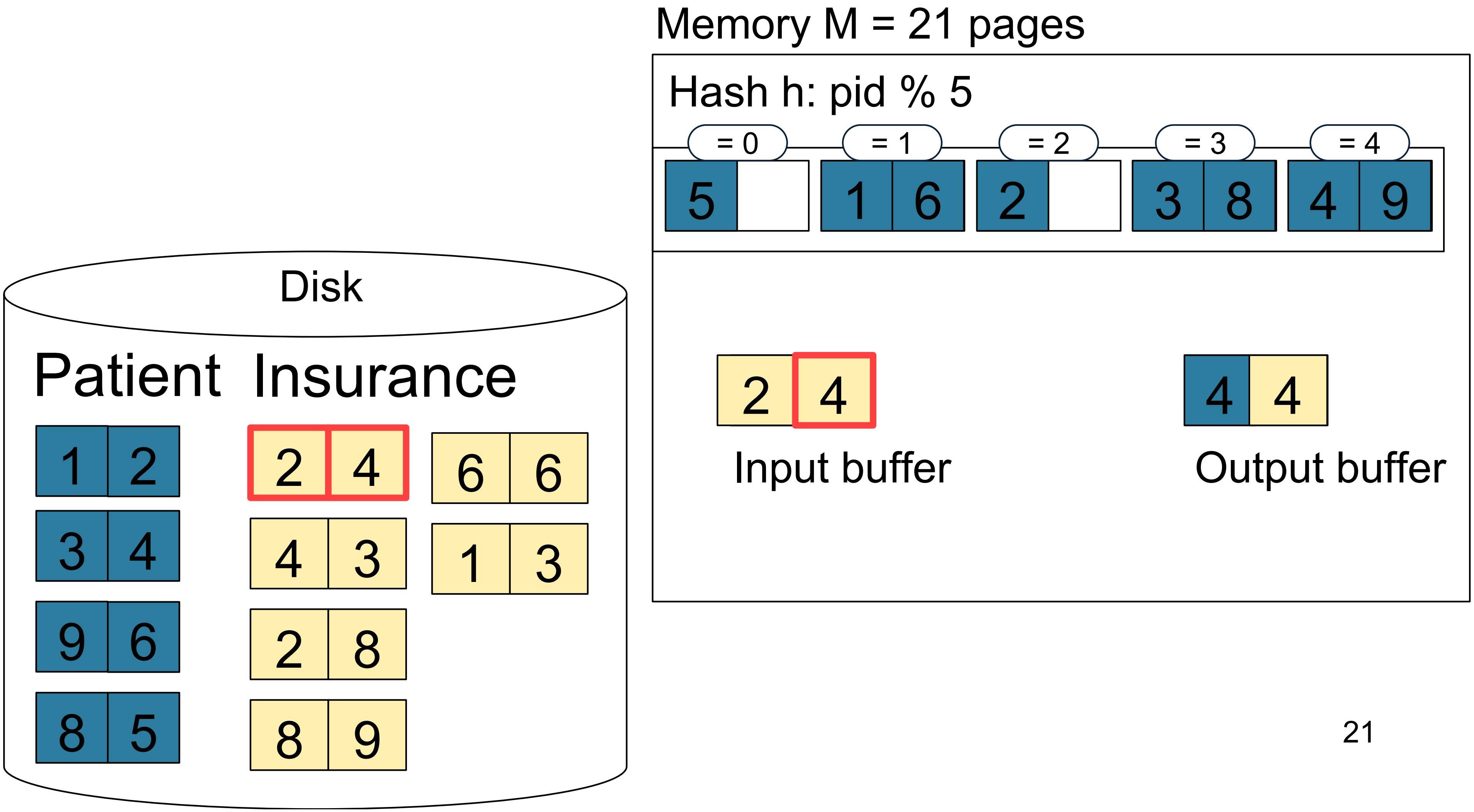
Hash Join Example

Step 2: Scan Insurance and probe into hash table



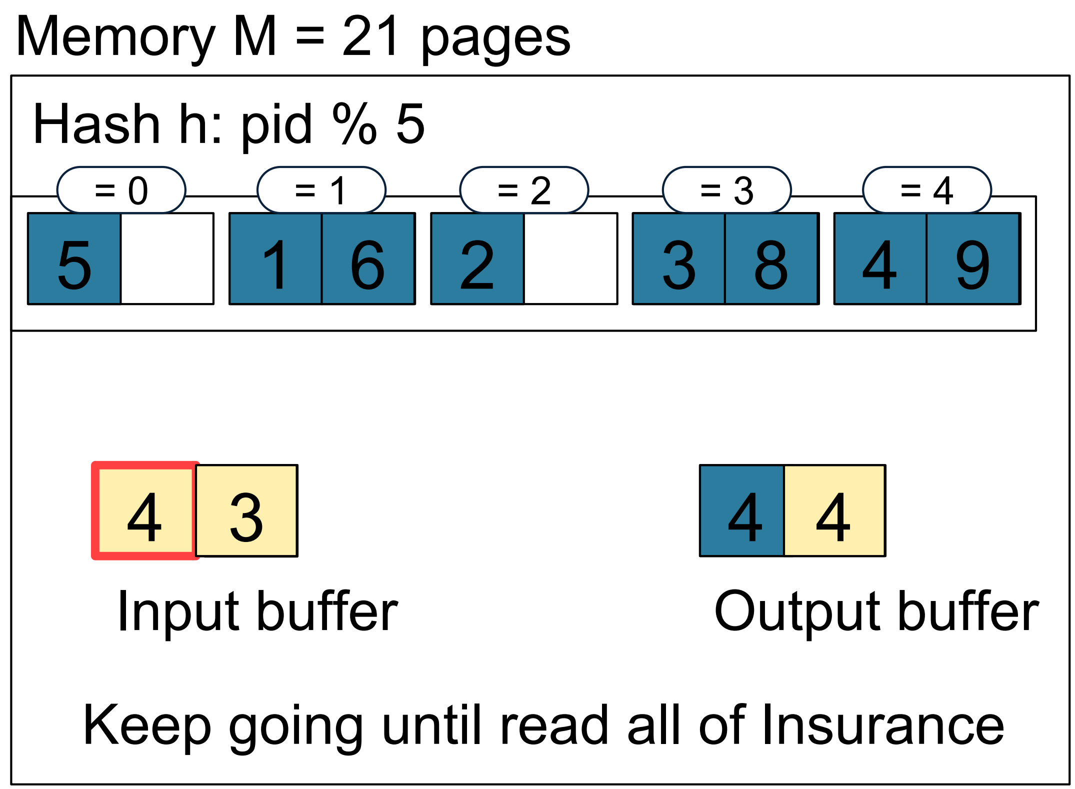
Hash Join Example

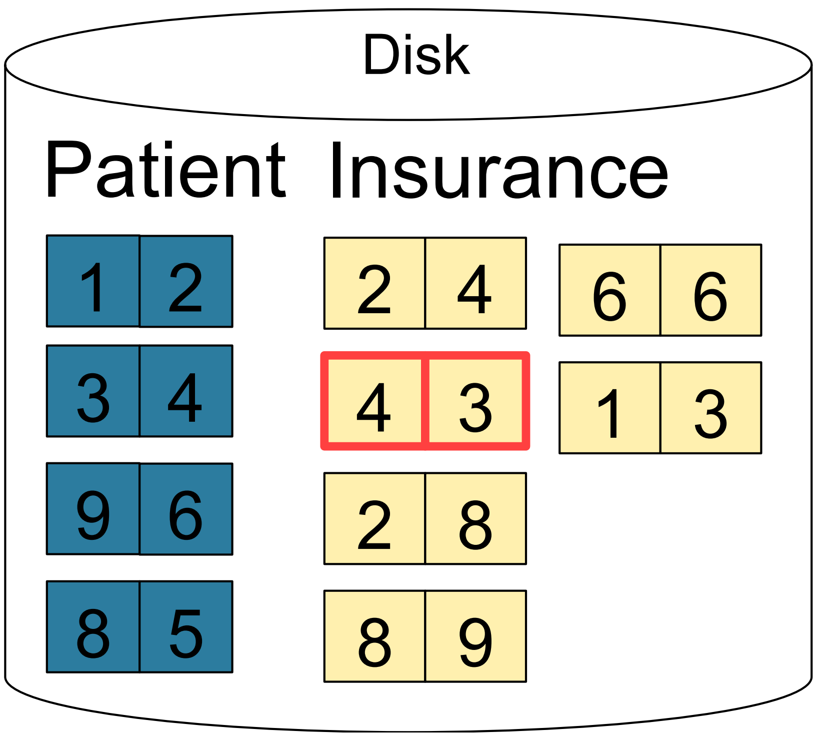
Step 2: Scan Insurance and probe into hash table



Hash Join Example

Step 2: Scan Insurance and probe into hash table

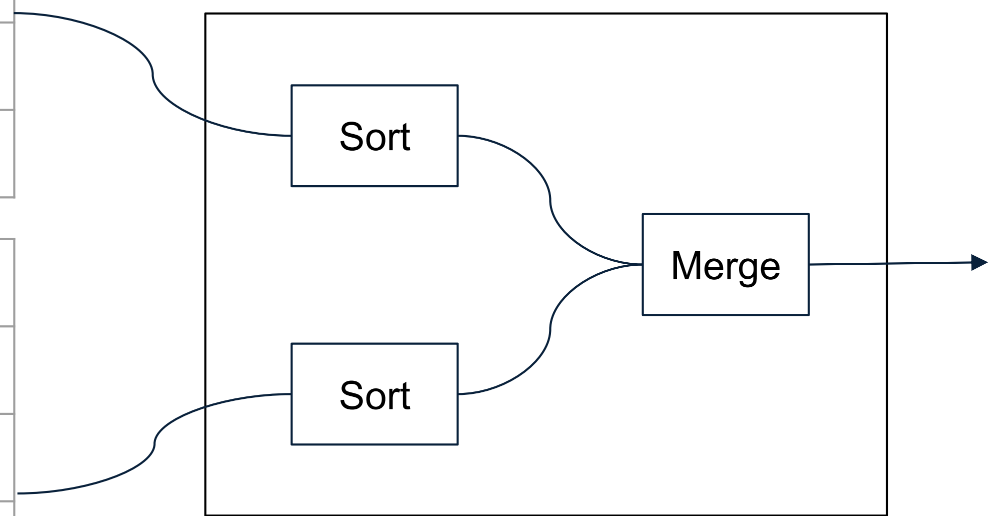




Cost: B(R) + B(S)

R		
a	b	c
1	7	4
…	…	…
98	3	2

S		
c	d	e
3	43	7
…	…	…
9	24	9



Sort-Merge Join

Sort-Merge Join

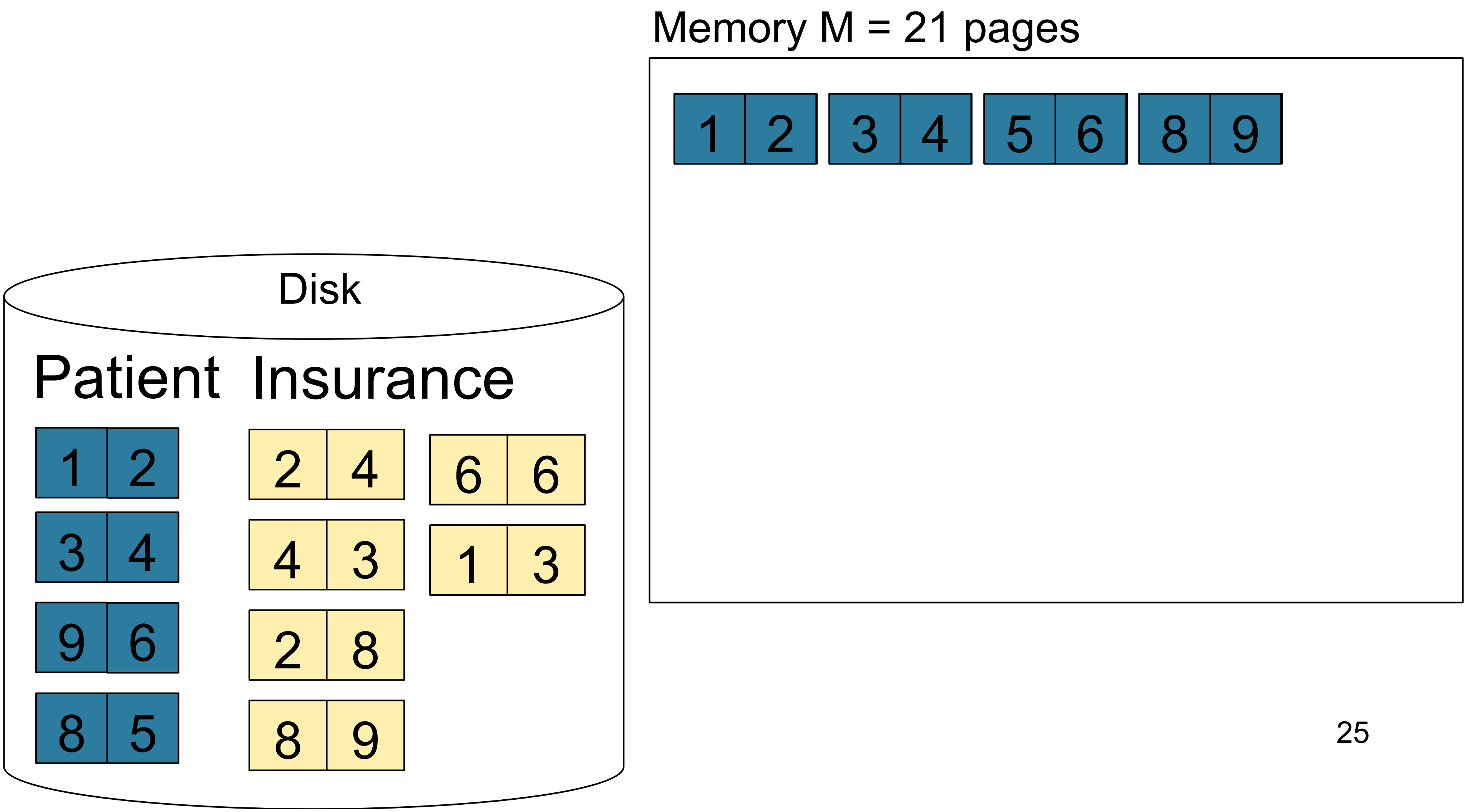
Sort-merge join: $R \bowtie S$

- 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

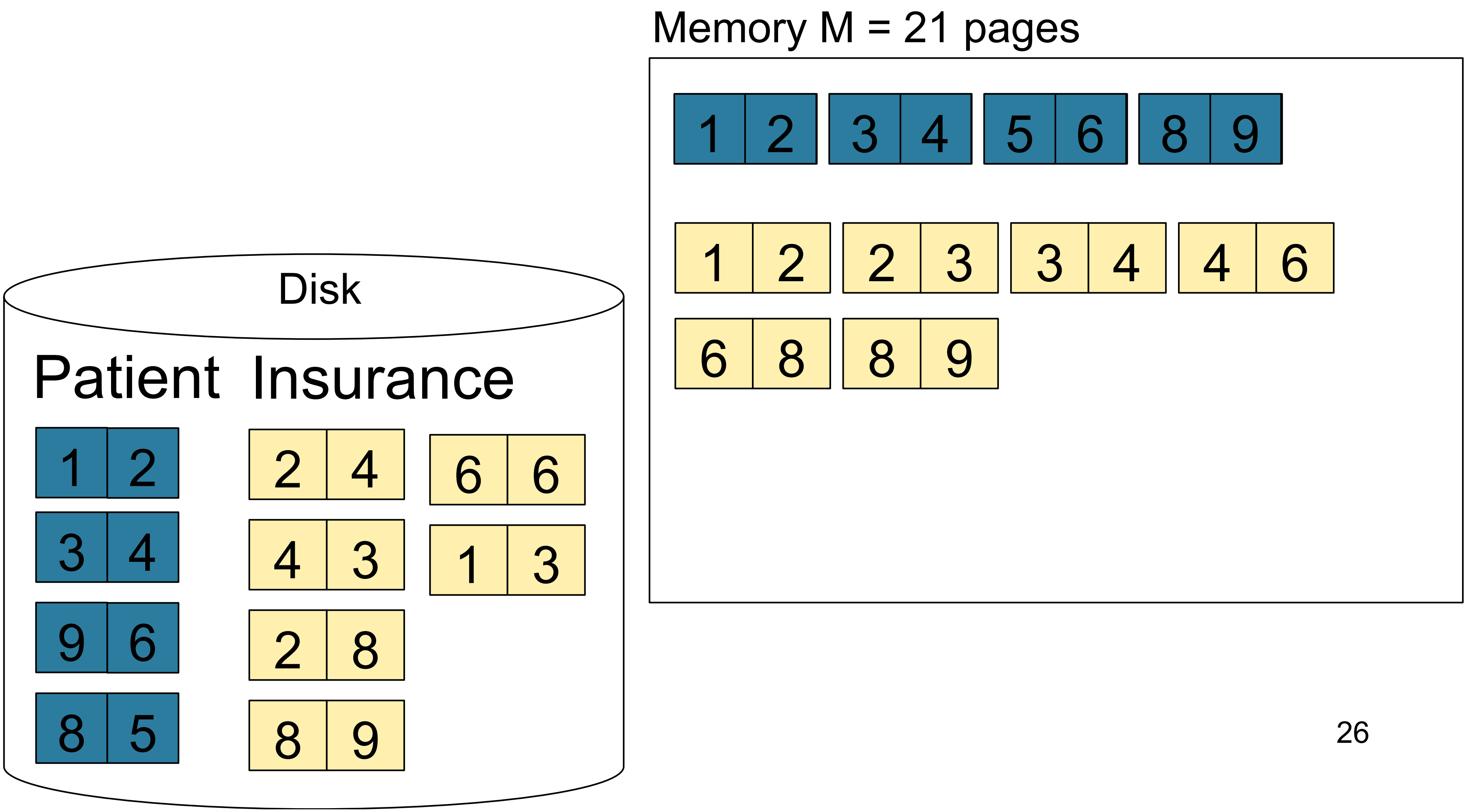
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory



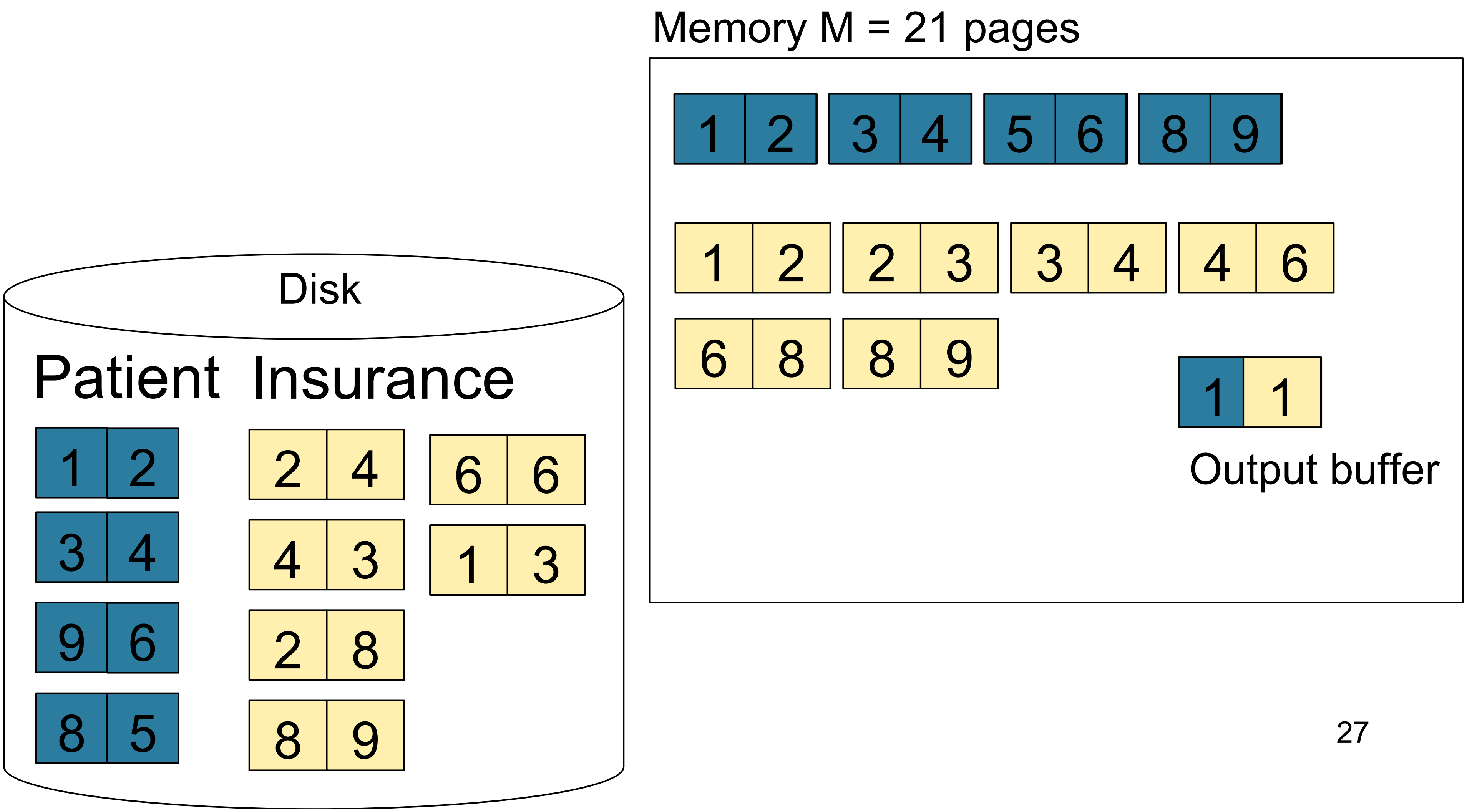
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory



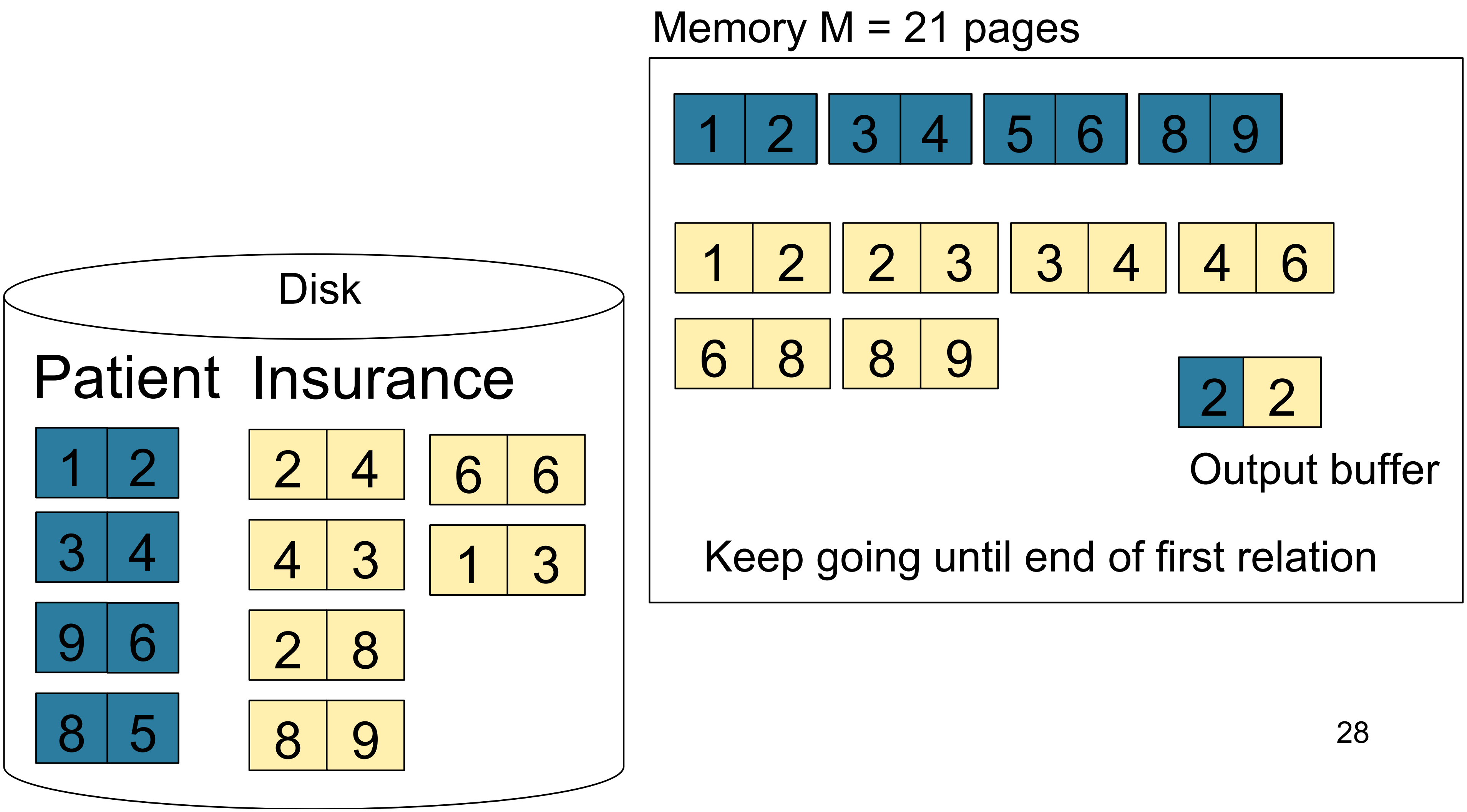
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



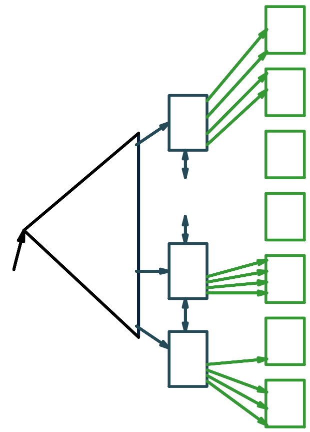
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



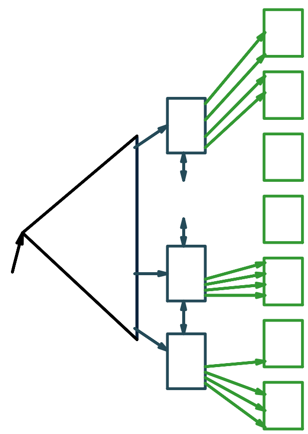
Index Joins

R		
a	b	c
1	7	4
…	...	...
98	3	2



S		
c	d	e
3	43	7
...	...	...
9	24	9

...



S		
c	d	e
3	43	7
...	...	...
9	24	9

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))

Index Nested Loop Join

If index on S is clustered:

$$B(R) + T(R) * (B(S) * 1/V(S,a))$$

Still have to scan in R

Why is the multiplier term T(R)?

What does 1/V(S,a) represent?

T(R) must be used because we cannot assume that a whole block of R (B(R)) will have the same attribute to join on, and thus use the same index access on S for.

1/V(S,a) represents the nature of the B+ Tree index. We are only scanning as much as we need. Note that the performance of the index join will decrease as V decreases.

Index Nested Loop Join

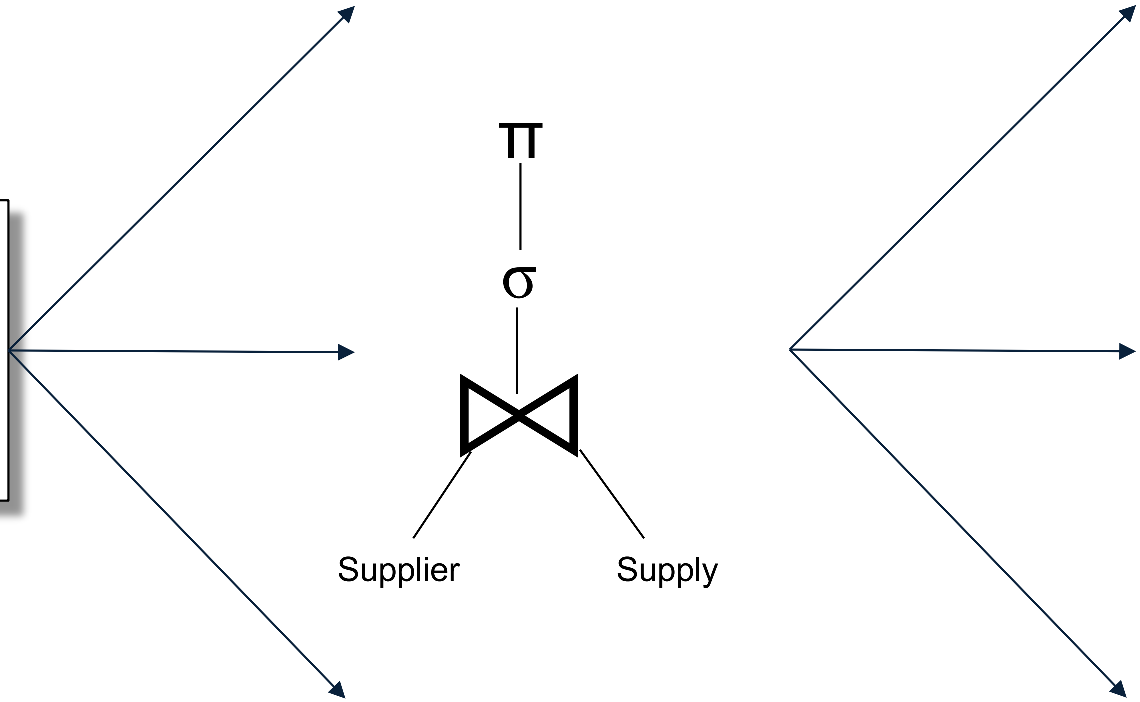
If index on S is unclustered:

$$B(R) + T(R) * (T(S) * 1/V(S,a))$$

Why did this change from B(R) to T(R)?

Remember that tuples are stored on contiguous blocks. In a clustered index from before we know we can scan a single chunk of the disk to get the entire desired range. In an unclustered index we no longer can assume contiguous access. Thus we estimate that every tuple needs its own I/O operation.

```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’
```



Generating Query Plans (review)

Review: Logical vs Physical Plans

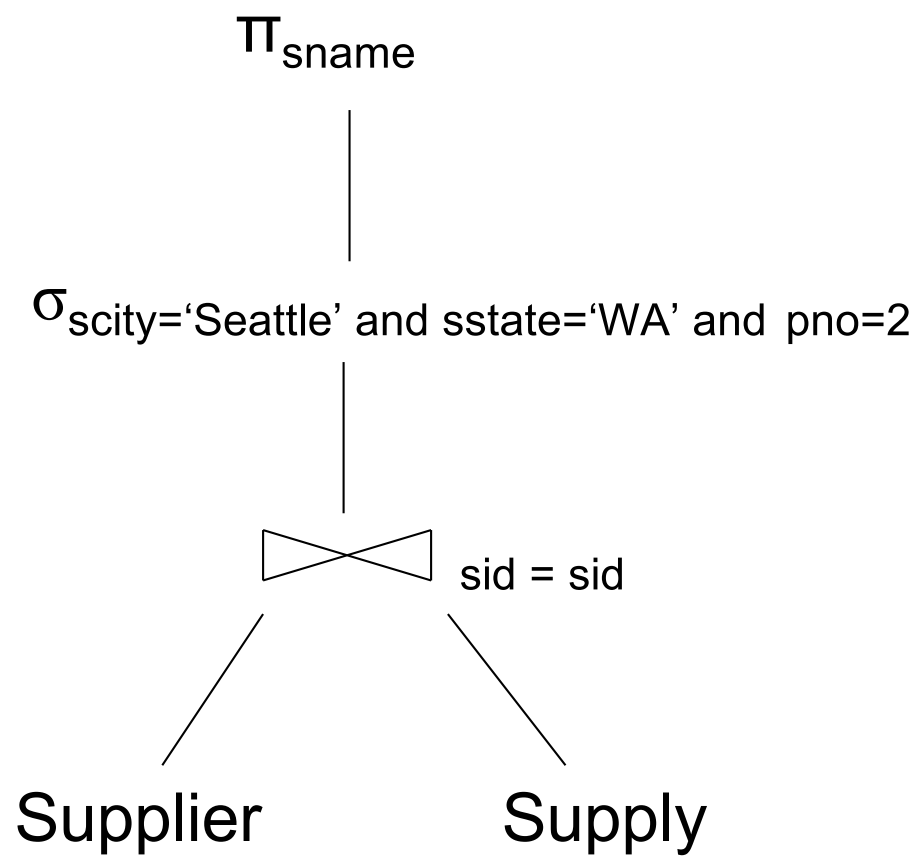
- Logical plans:
  - Created by the parser from the input SQL text
  - Expressed as a relational algebra tree
  - Each SQL query has many possible logical plans

- Physical plans:
  - Goal is to choose an efficient implementation for each operator in the RA tree
  - Each logical plan has many possible physical plans

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

Review: Relational Algebra

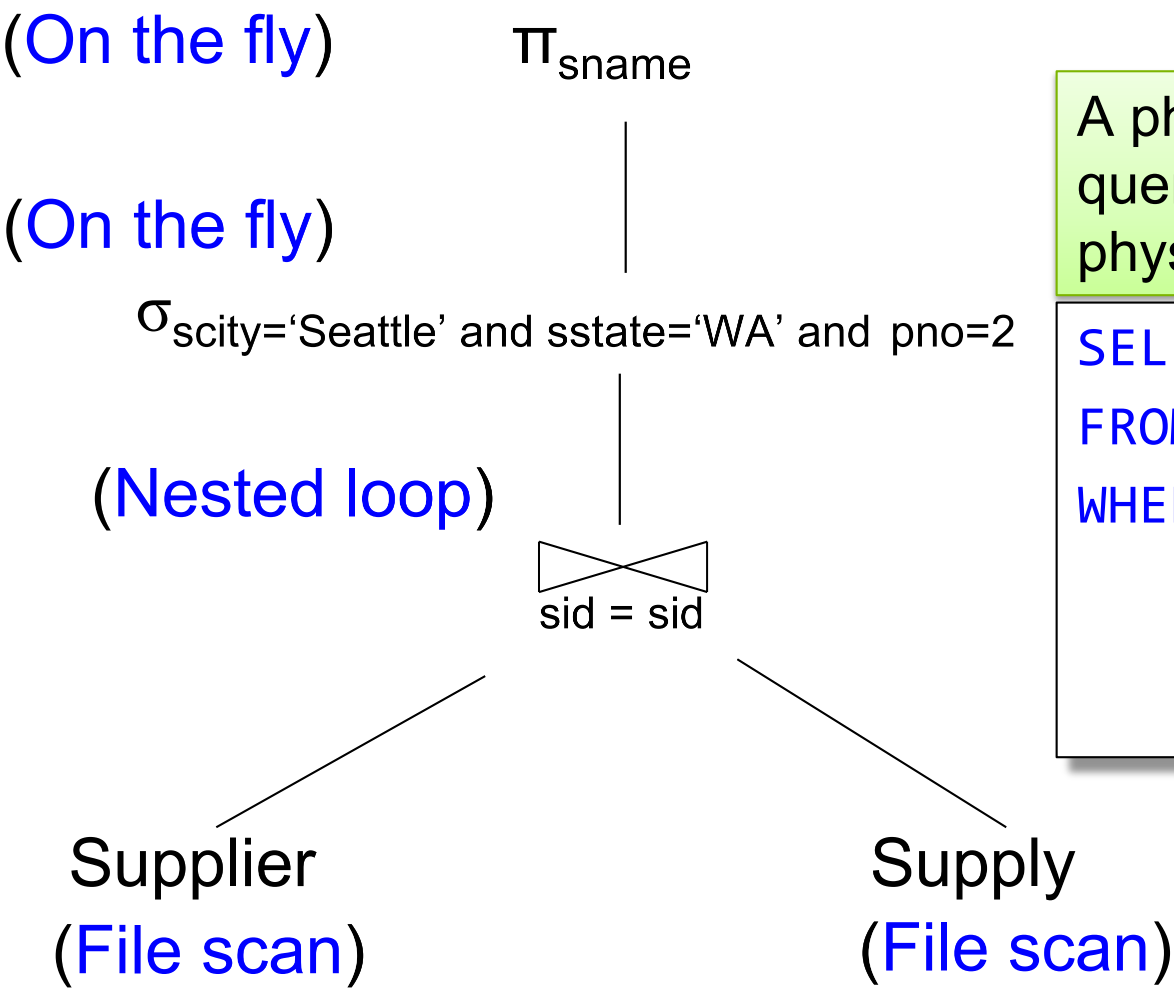
```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```



Relational algebra expression is also called the “logical query plan”

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

Review: Physical Query Plan 1



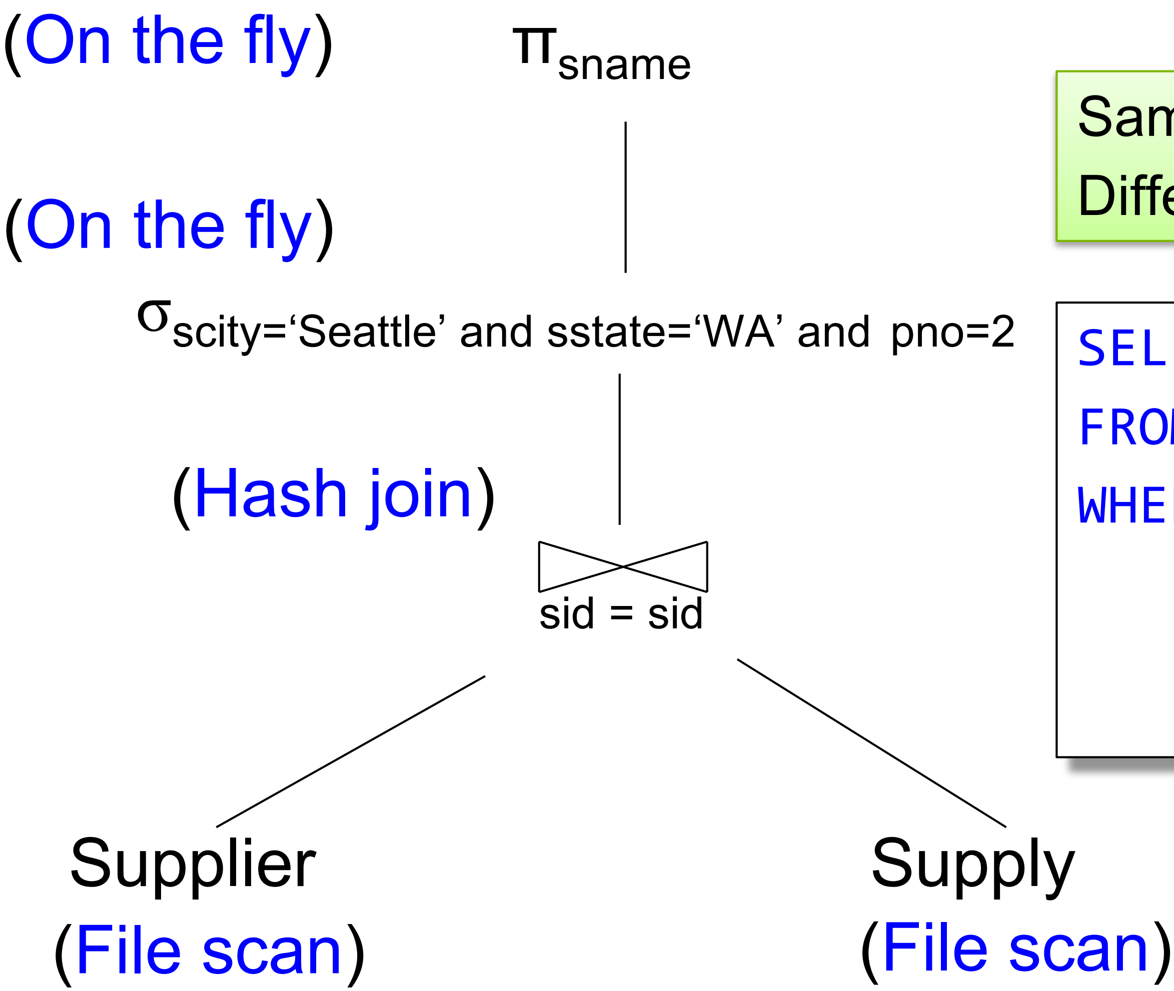
A physical query plan is a logical query plan annotated with physical implementation details

```
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)

Review: Physical Query Plan 2



Same logical query plan

Different physical plan

```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)

Review: Physical Query Plan 3

Different but equivalent logical query plan; different physical

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

(On the fly) π_{sname} (d)

(Sort-merge join) $\bowtie_{sid = sid}$ (c)

(Scan & write to T1) (a) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'}}$

(b) $\sigma_{pno=2}$ (Scan & write to T2)

Supplier (File scan)

Supply (File scan)

Query Optimization: Overview

- Compute cost of each operator
  - This depends on:
    - Table statistics (# of tuples etc)
    - Algorithm used

- Cost of a physical plan =
  sum(each operator cost)

- Cost each plan and choose the one with lowest cost