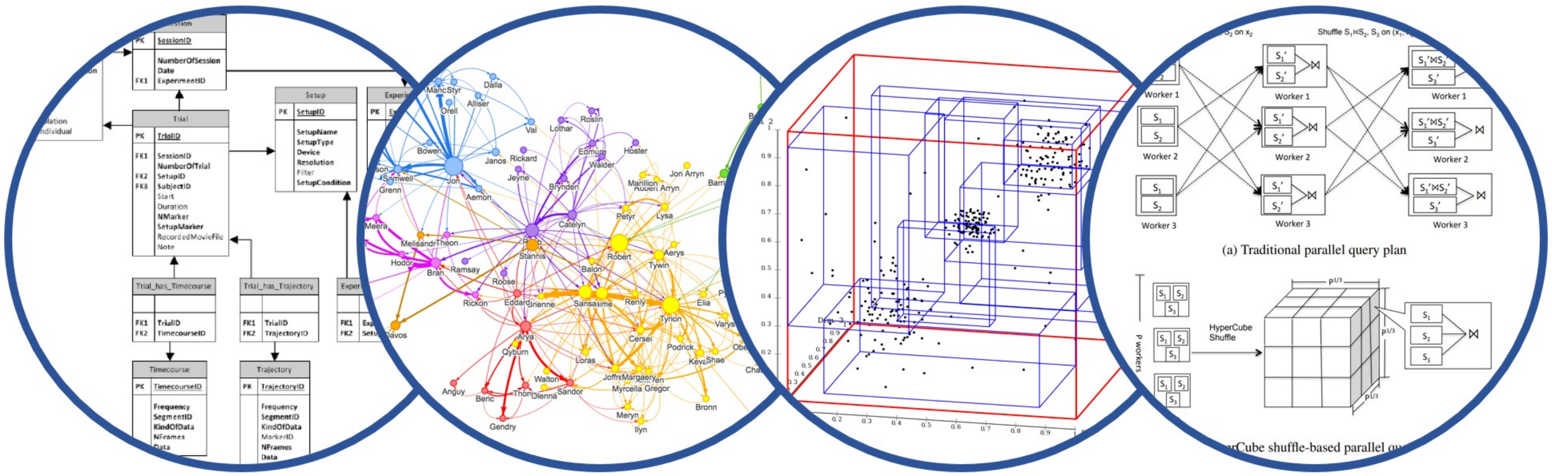


Database System Internals

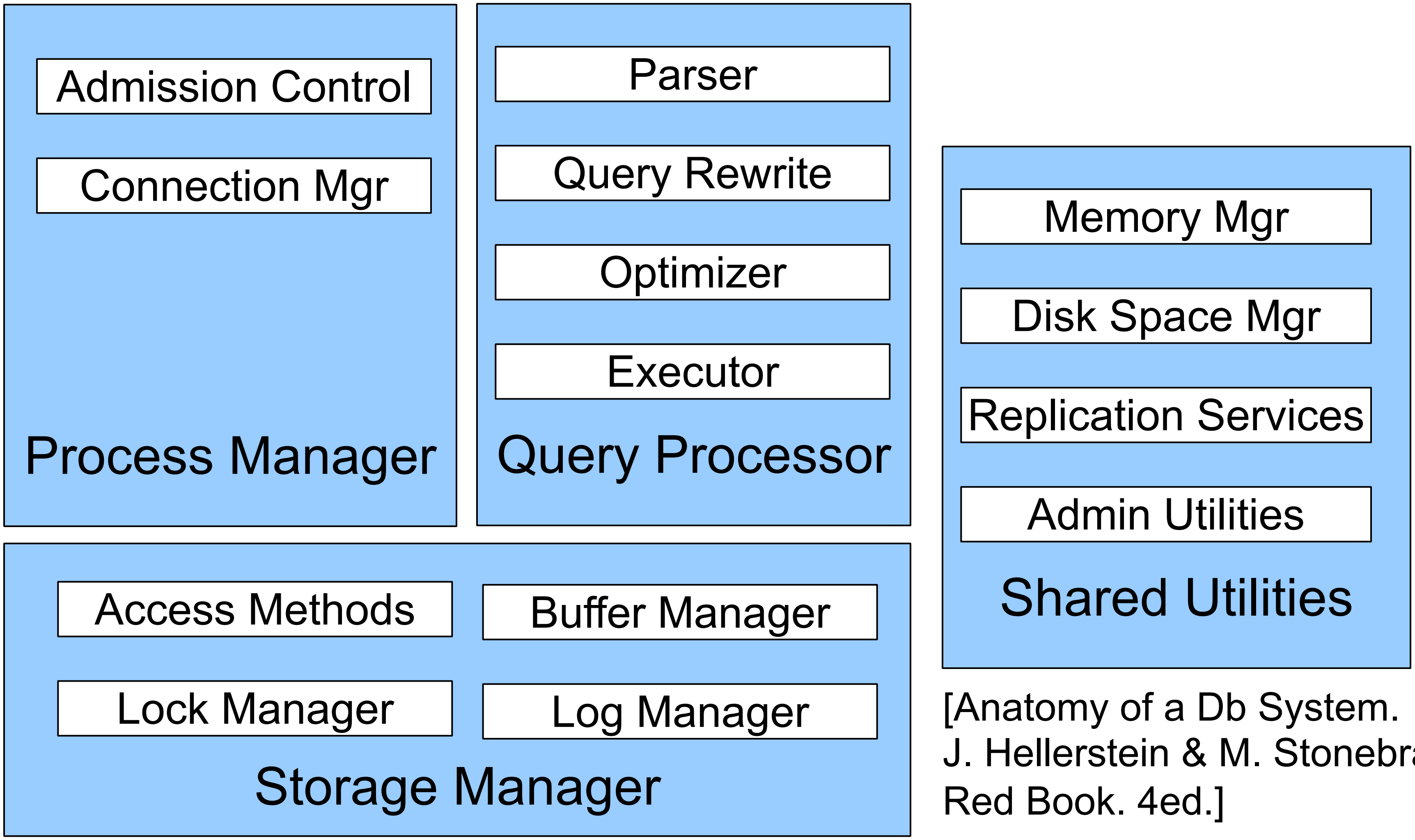
Query Execution and Algorithms

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

What We Have Learned So Far

- Overview of the architecture of a DBMS
- Access methods
  - Heap files, sequential files, Indexes (hash or B+ trees)
- Role of buffer manager
- Practiced the concepts in hw1 and lab1

DBMS Architecture



[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Query Processor

- Query optimization: find a good plan
- Query execution: execute the plan

We start with execution and analyze its cost.
That will inform how to optimize.

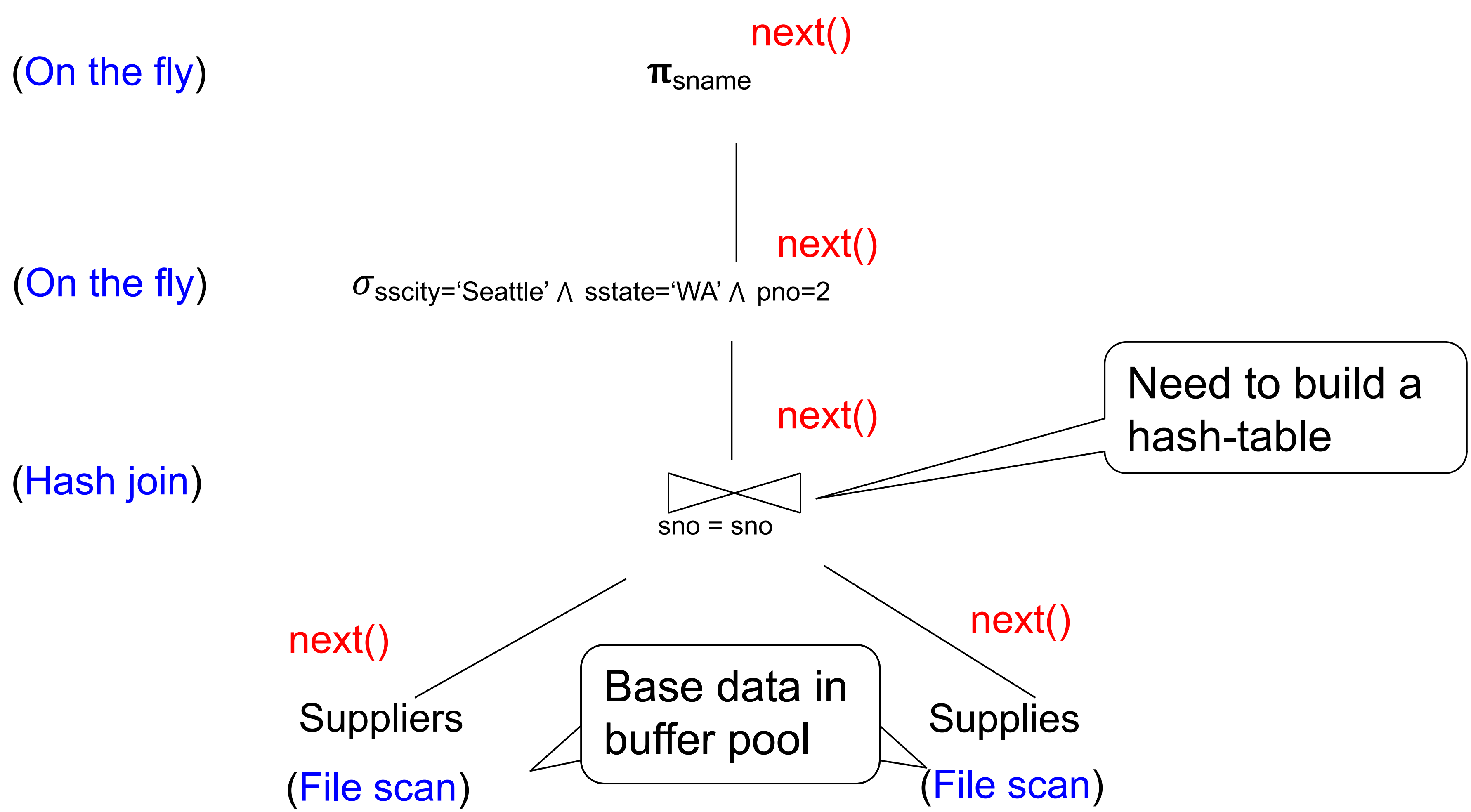
Query Execution Summary

SQL query transformed into physical plan

- Access path selection for each relation
- Implementation choice for each operator
- Scheduling decisions for operators:
  - Single-threaded or parallel
  - Pipelined or materialized

Operators given a limited amount of memory

Pipelined Query Execution



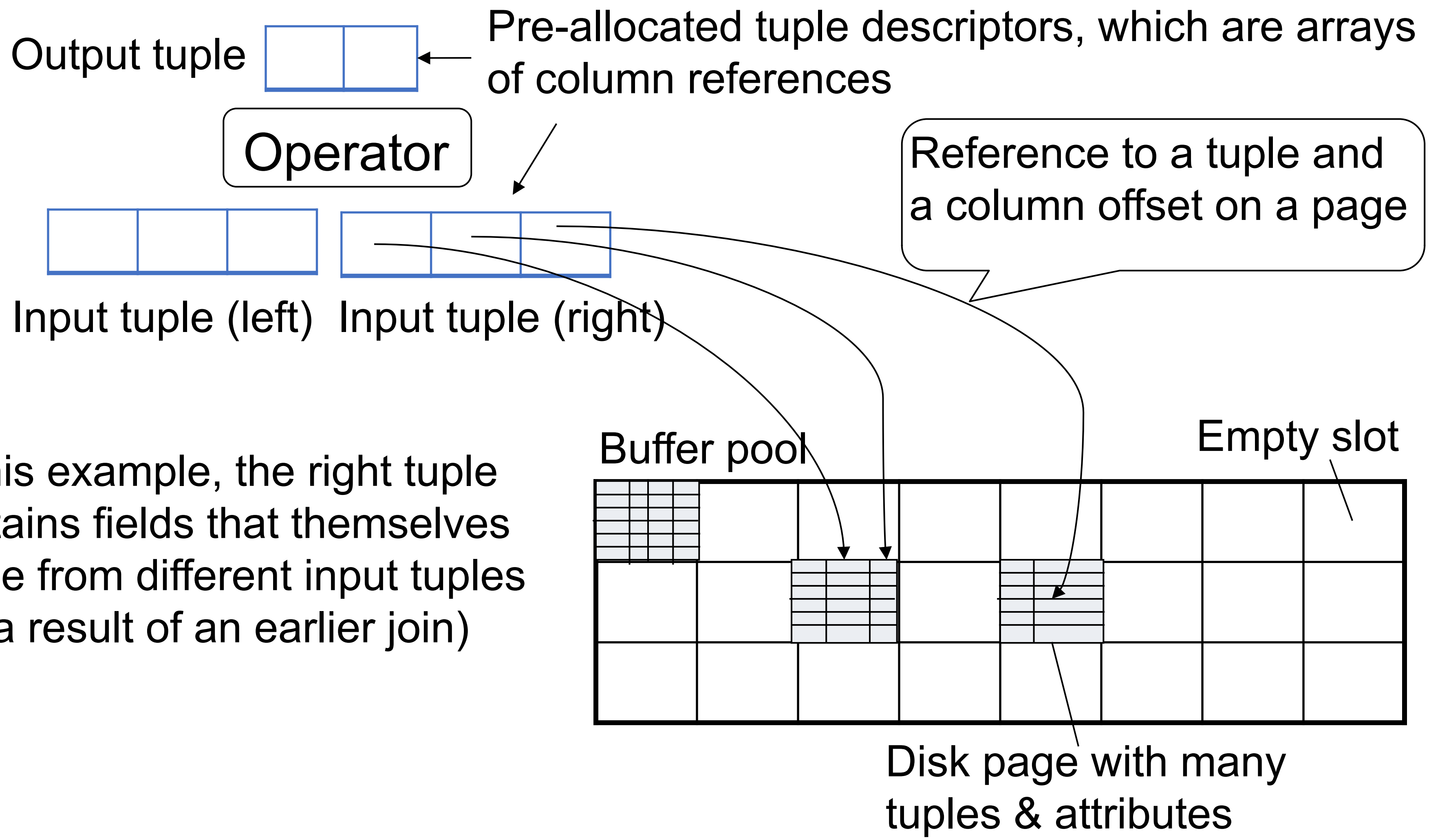
Memory Management

Each operator:

- Pre-allocates heap space for input/output tuples
  - Option 1, BP-tuples: pointers to data in buffer pool
  - Option 2, M-tuples: new tuples on the heap
- Allocates memory for its internal state
  - On heap

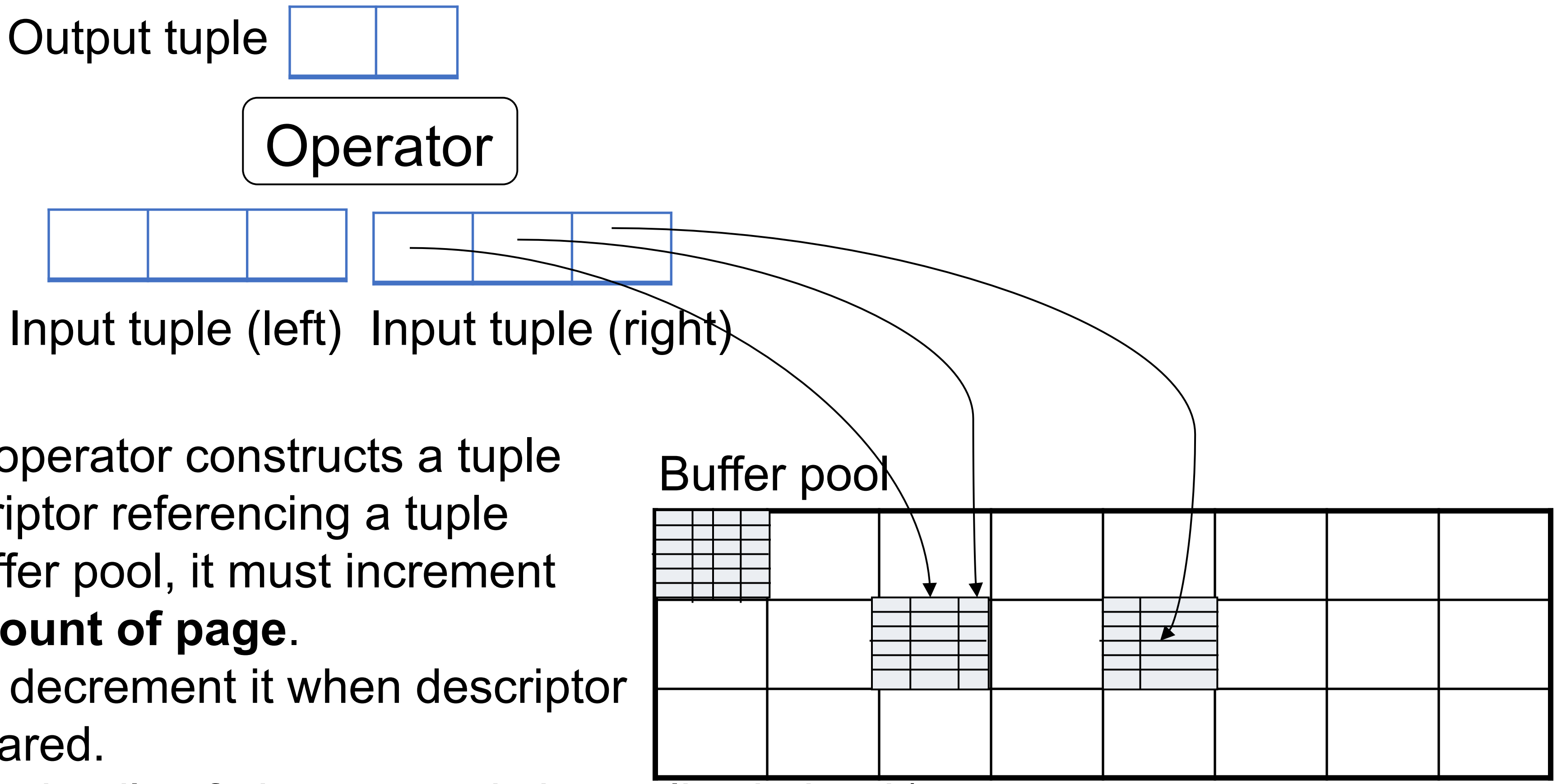
DMBS **limits** how much memory each operator, or each query can use

BP-tuples (option 1)



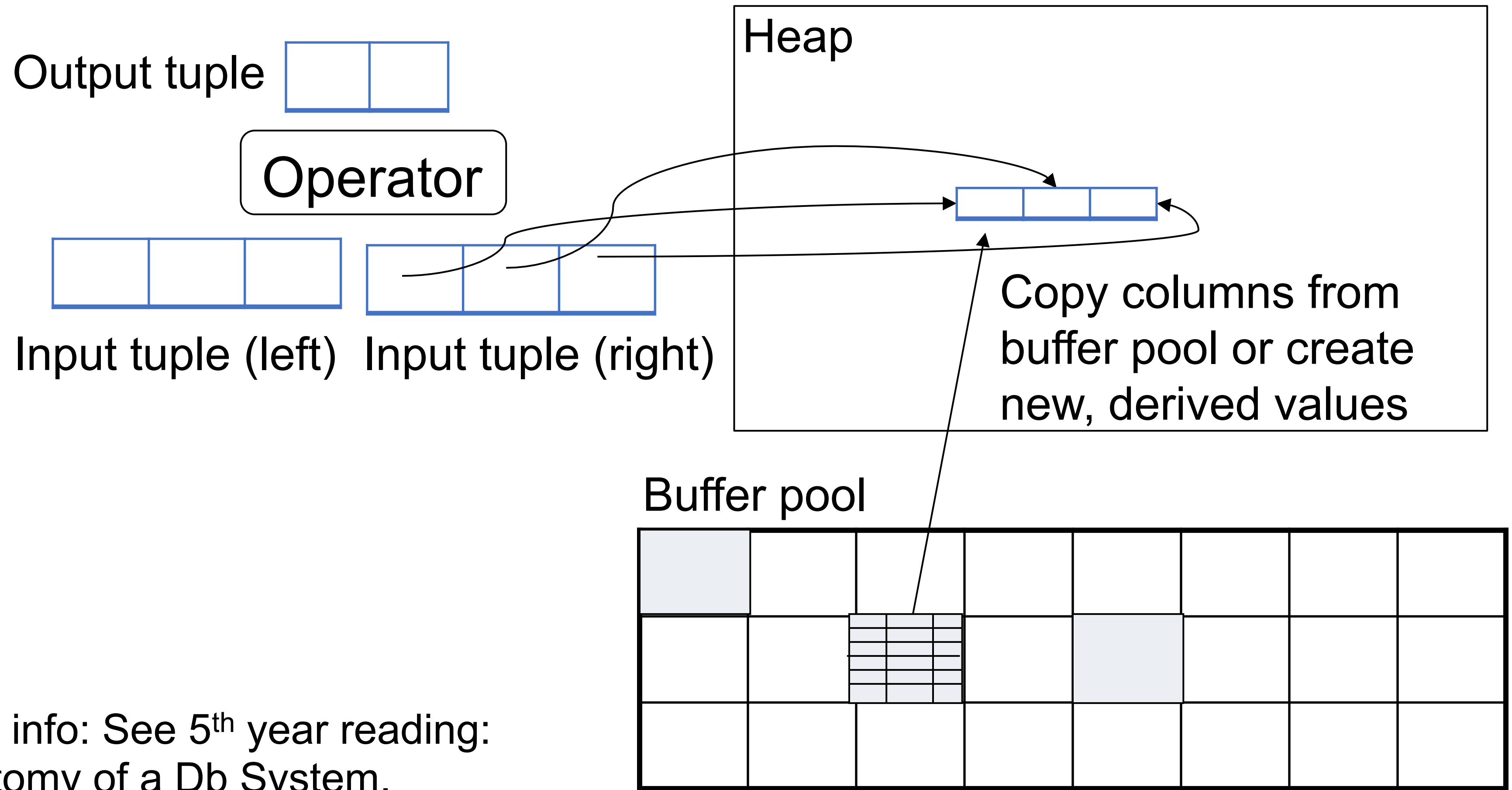
In this example, the right tuple contains fields that themselves come from different input tuples (as a result of an earlier join)

BP-tuples (option 1)



If an operator constructs a tuple descriptor referencing a tuple in buffer pool, it must increment **pin count of page**.
Then decrement it when descriptor is cleared.
(more details of pin count eviction policy in book)

M-Tuples (option 2)



More info: See 5th year reading:
[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Discussion

Buffer-Pool tuples (BP-tuples)

- Pros: don't copy the data (great performance)
- Cons:
  - Need to pin pages in the BP
  - Cannot compute new values:
    SELECT pid, price*quantity FROM …

Heap-tuples, or memory-tuples (M-tuples)

- Pros
  - No need to pin pages (except short period – why?)
  - Can represent new values: price*quanity
- Cons: data copying can degrade performance

Operator Algorithms
(Quick review from 344 today & new algorithms next time)

Operator Algorithms

Design criteria

- Cost: IO, CPU, Network
- Memory utilization
- Load balance (for parallel operators)

Cost Parameters

- **Cost = total number of I/Os**
  - This is a simplification that ignores CPU, network

- **Parameters:**
  - **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)**

Convention

- Cost = the cost of reading operands from disk
- Cost of writing the final result to disk is *not included;* need to count it separately when applicable

Outline

- **Join operator algorithms**
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
  - Two-pass algorithms (Sec 15.4 and 15.5)

- Note about readings:
  - In class, we discuss only algorithms for joins
  - Other operators are easier: book has extra details

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)

- One-pass algorithm when B(R) ≤ M

Note: the *inner* relation is the relation on which we build the hash table
- Usually this is the *right* relation, i.e. S.
- But the following slides choose the *left* relation, i.e. R

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'

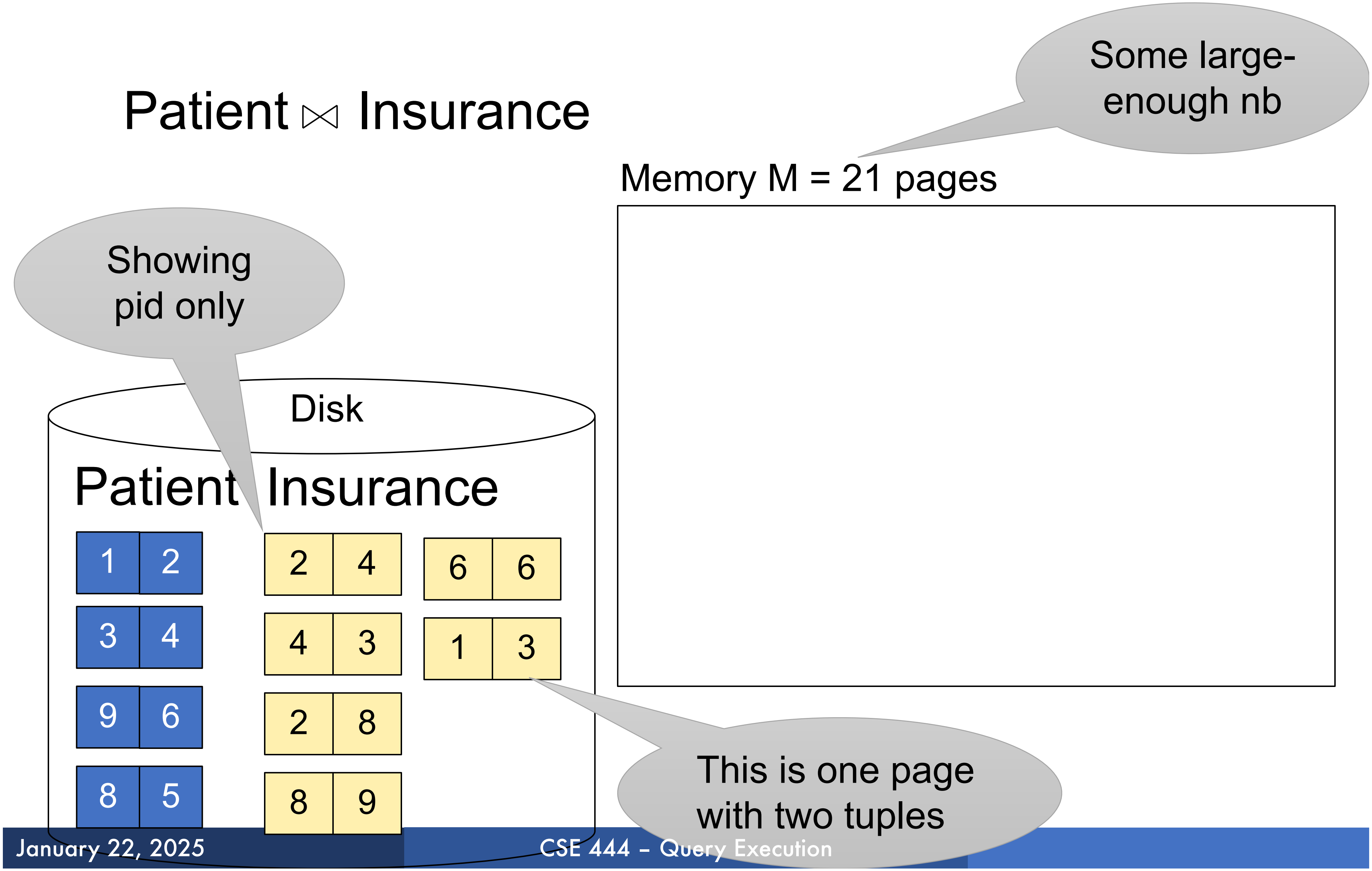
Insurance

2	'Blue'	123
4	'Prem'	432

4	'Prem'	343
5	'GrpH'	554

Hash Join Example

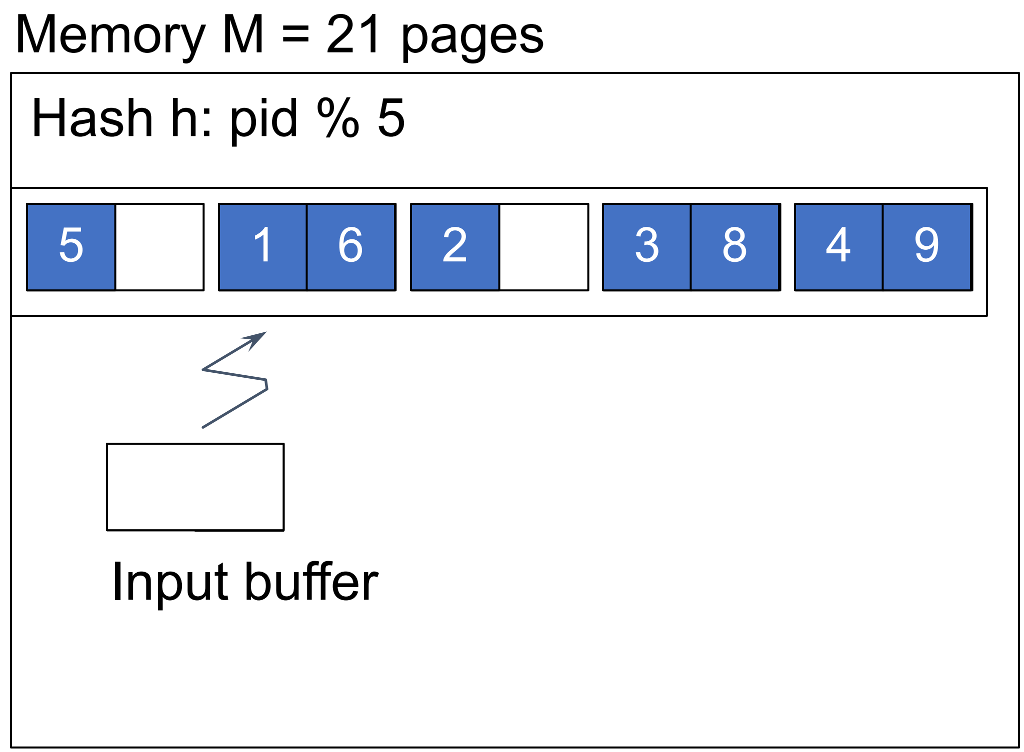
Patient $\bowtie$ Insurance

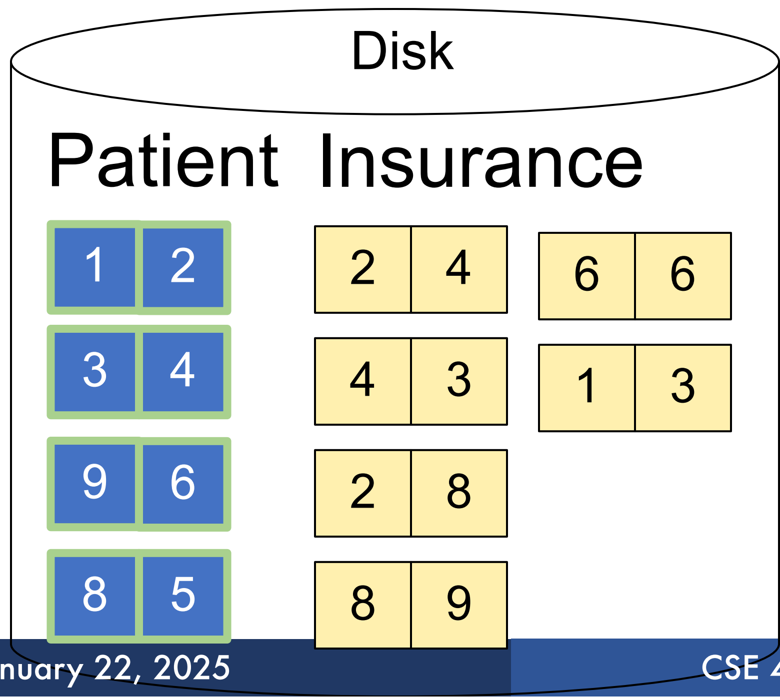


Hash Join Example

Step 1: Scan Patient and build hash table in memory

Can be done in method open()

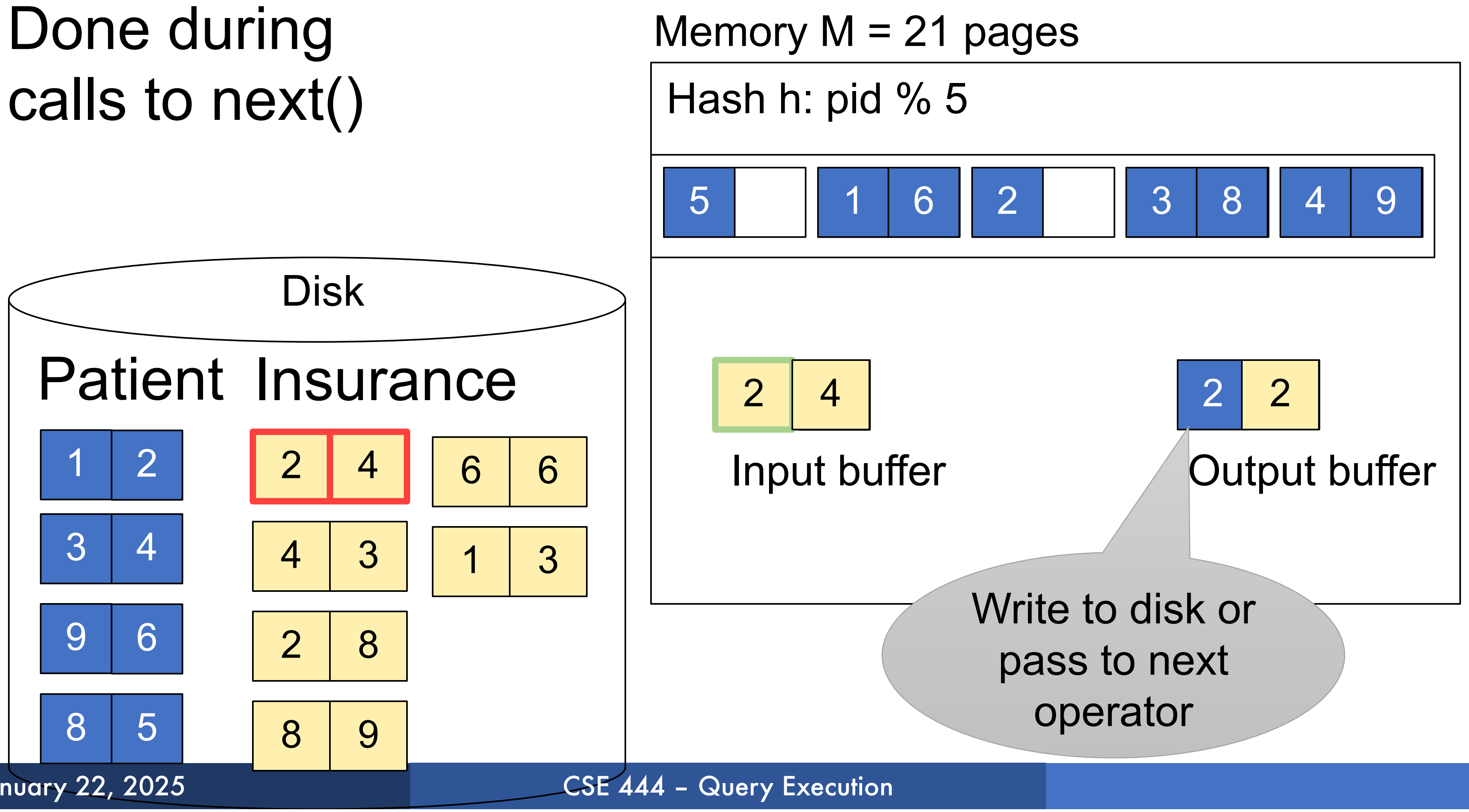




Hash Join Example

Step 2: Scan Insurance and probe into hash table

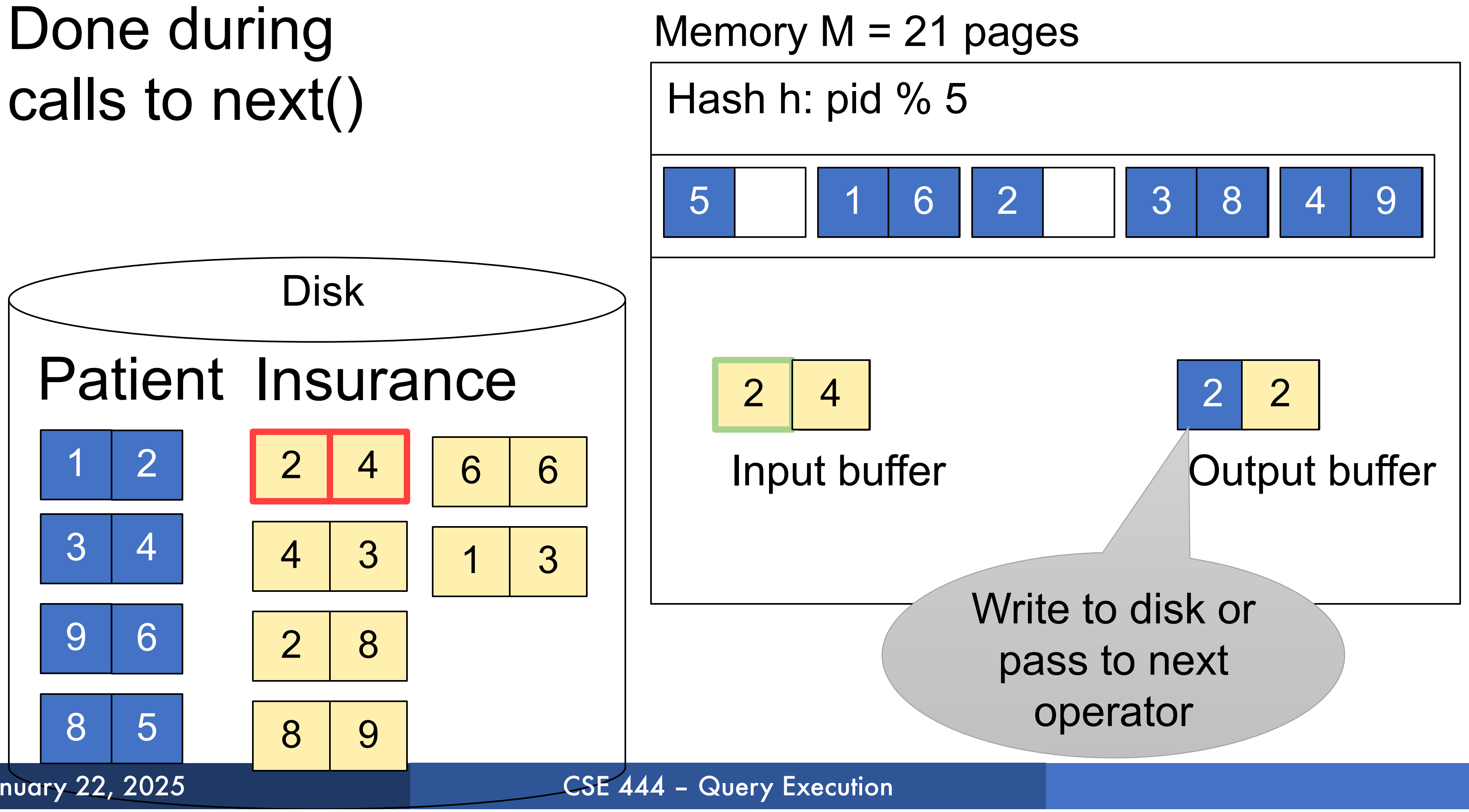
Done during calls to next()



Hash Join Example

Step 2: Scan Insurance and probe into hash table

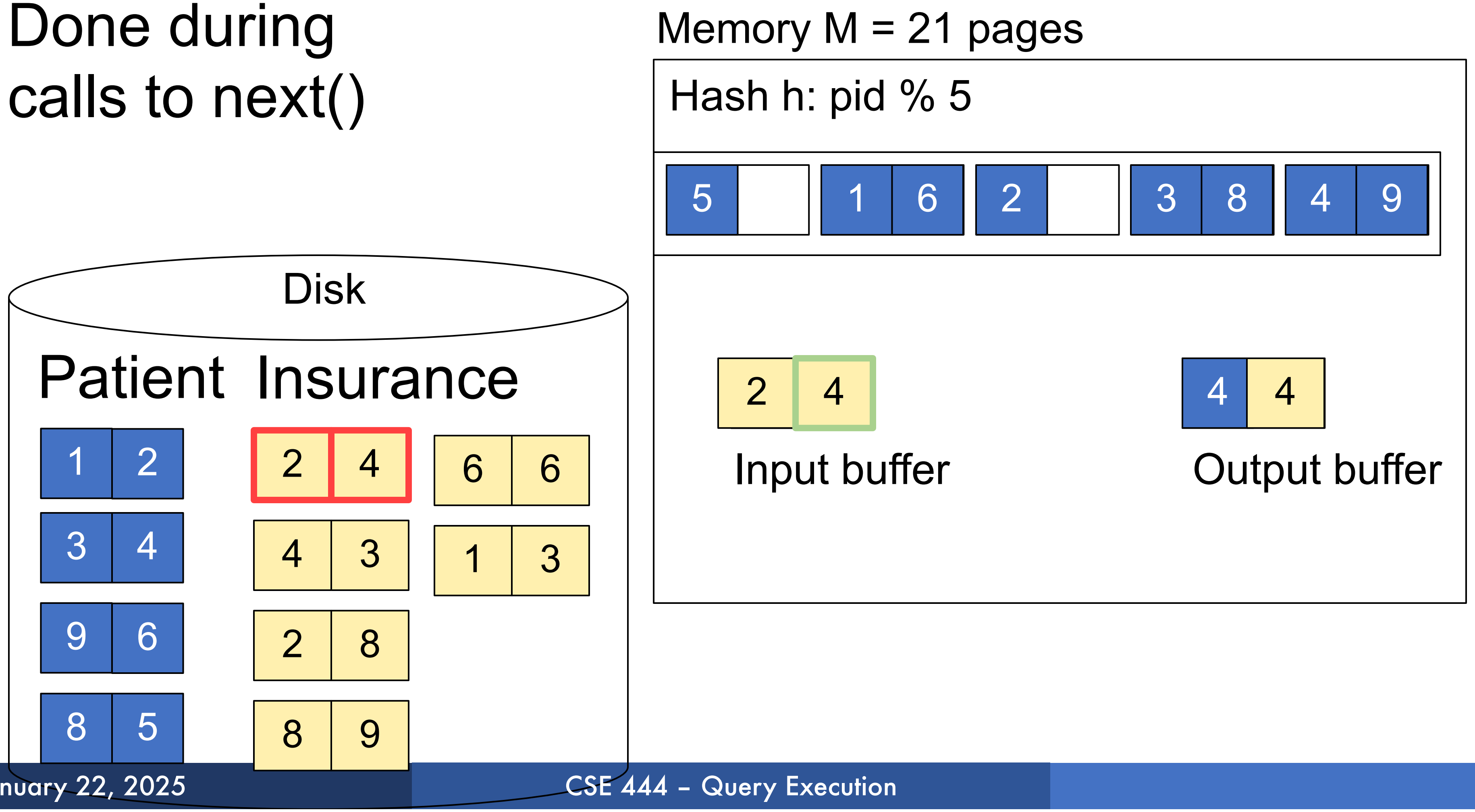
Done during calls to next()



Hash Join Example

Step 2: Scan Insurance and probe into hash table

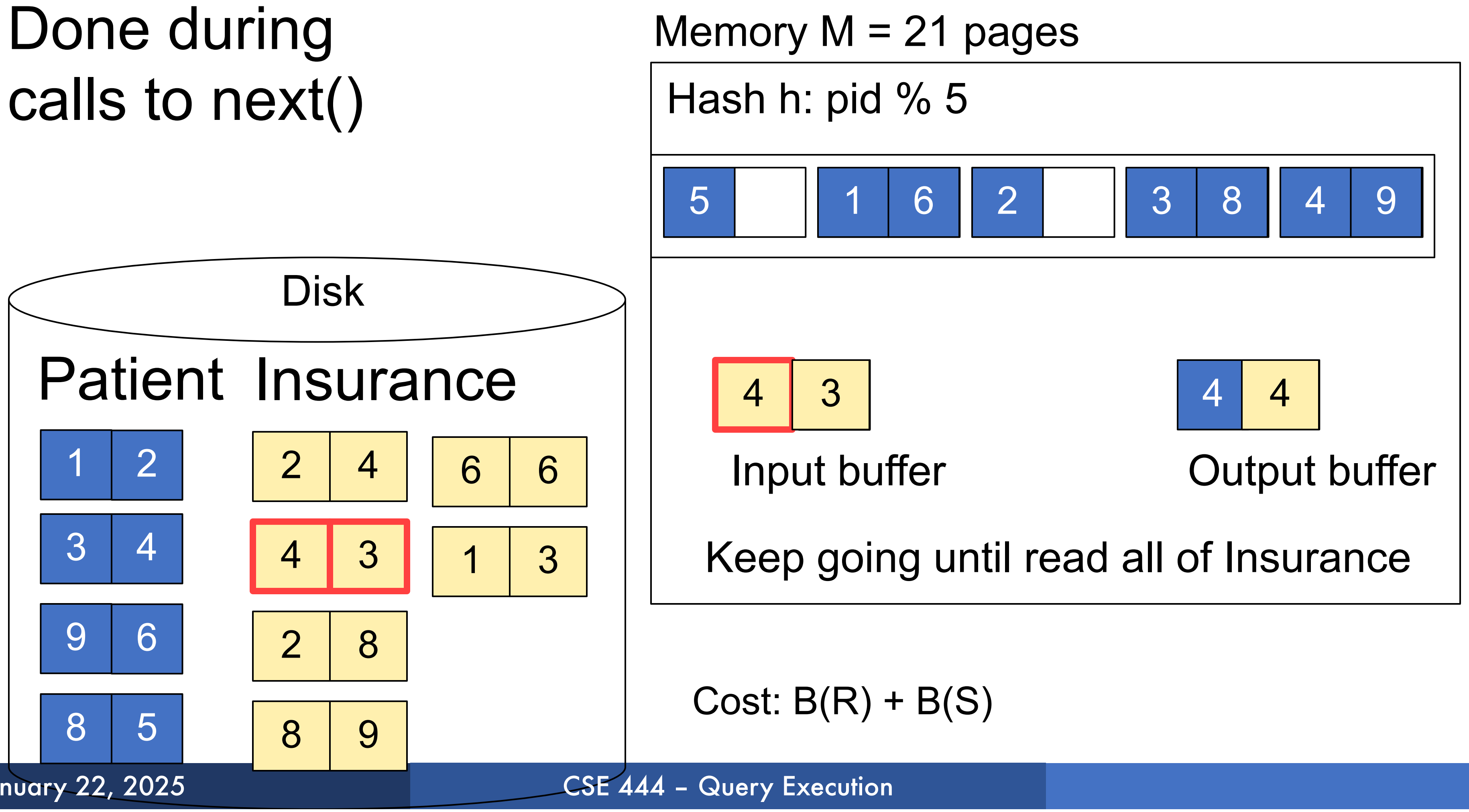
Done during calls to next()



Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during calls to next()



Cost: B(R) + B(S)

Discussion

- Hash-join is the workhorse of database systems
- The hash table is built on the heap, not in BP; hence it is not organized in pages, but pages are still convenient to think about it
- Hash-join works great when:
  - The inner table fits in main memory
  - The hash function is good (never write your own!)
  - The data has no skew (discuss in class...)

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 $R \bowtie S$
- 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)
```

What is the Cost?

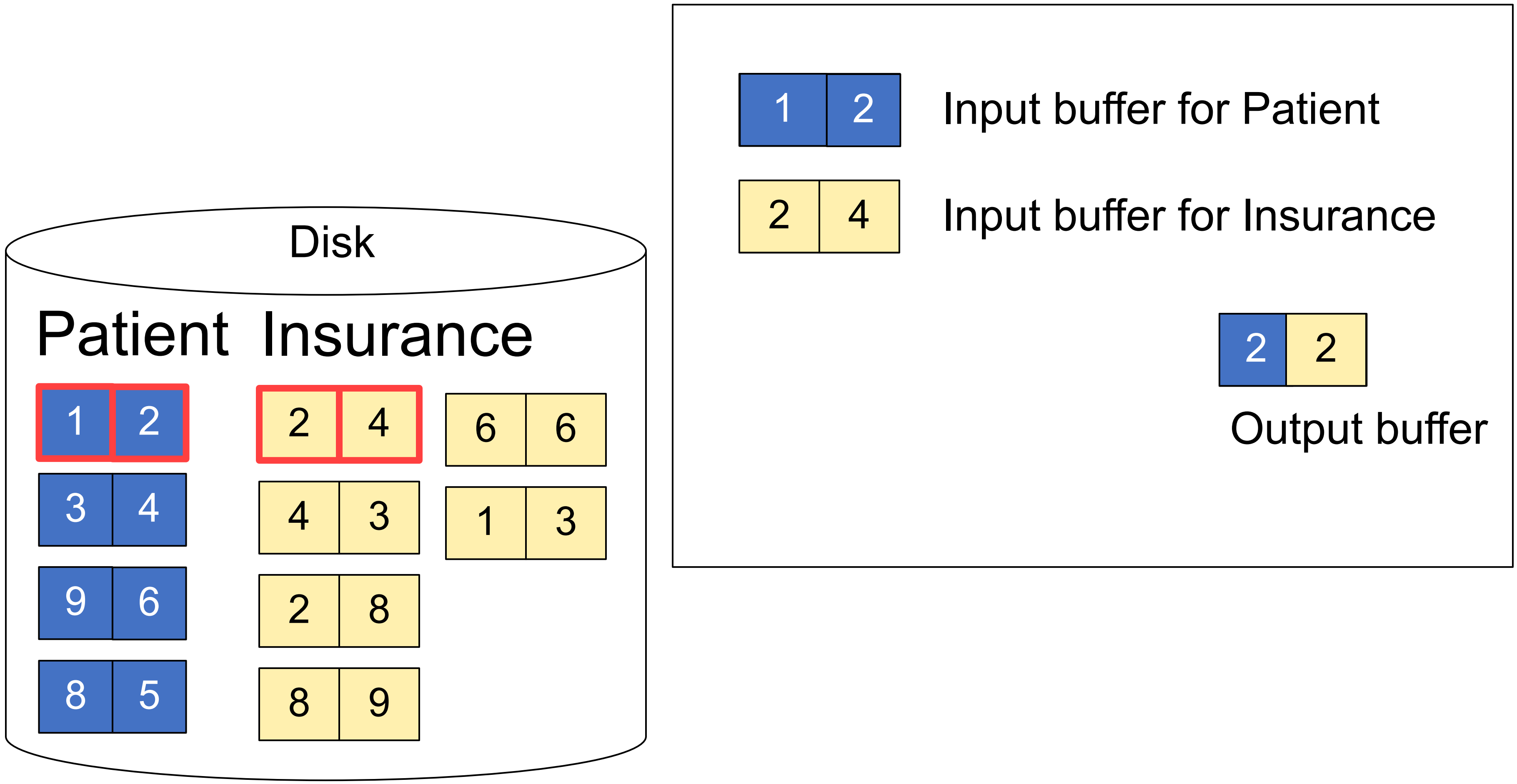
Page-at-a-time Refinement

```
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)
```

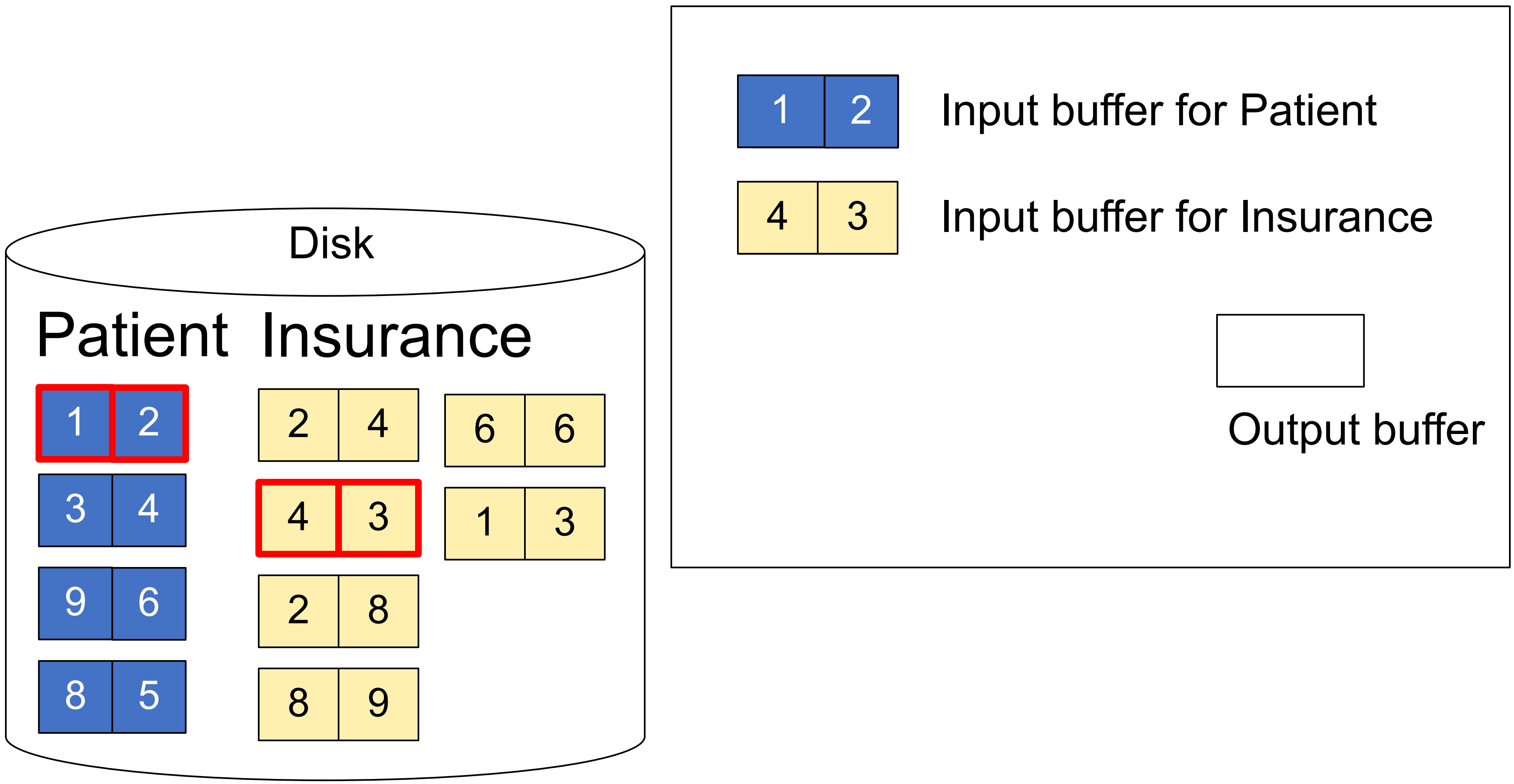
- Cost: B(R) + B(R)B(S)

What is the Cost?

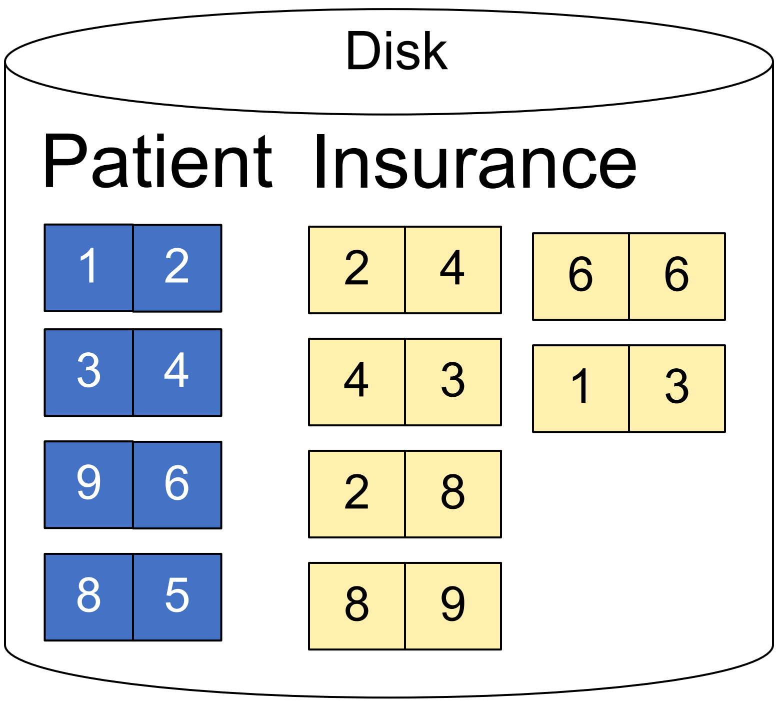
Page-at-a-time Refinement

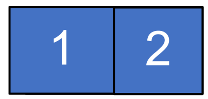


Page-at-a-time Refinement

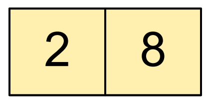


Page-at-a-time Refinement





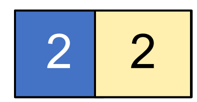
Input buffer for Patient



Input buffer for Insurance

Keep going until read all of Insurance

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



Output buffer

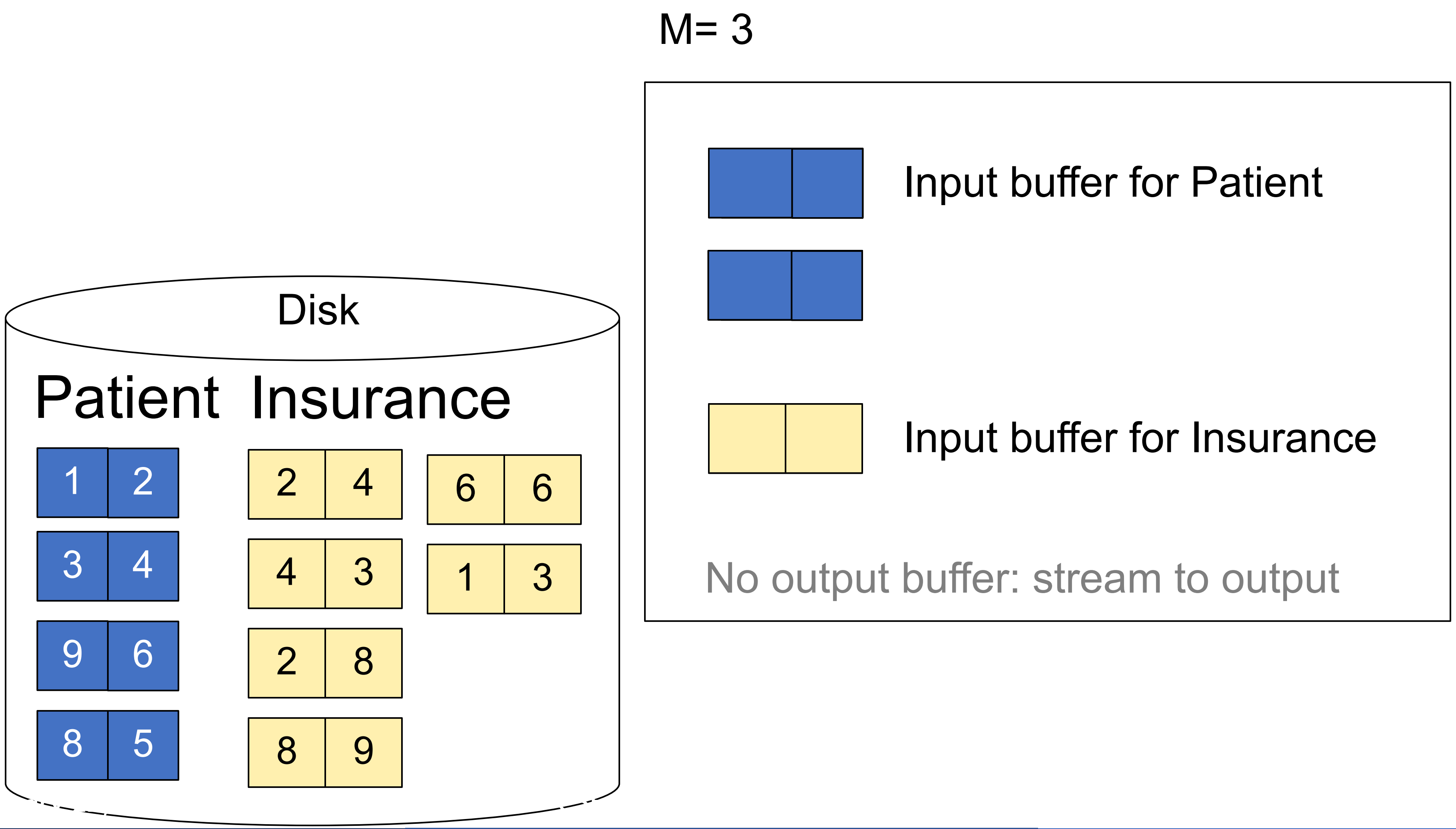
Cost: B(R) + B(R)B(S)

Block-Memory Refinement

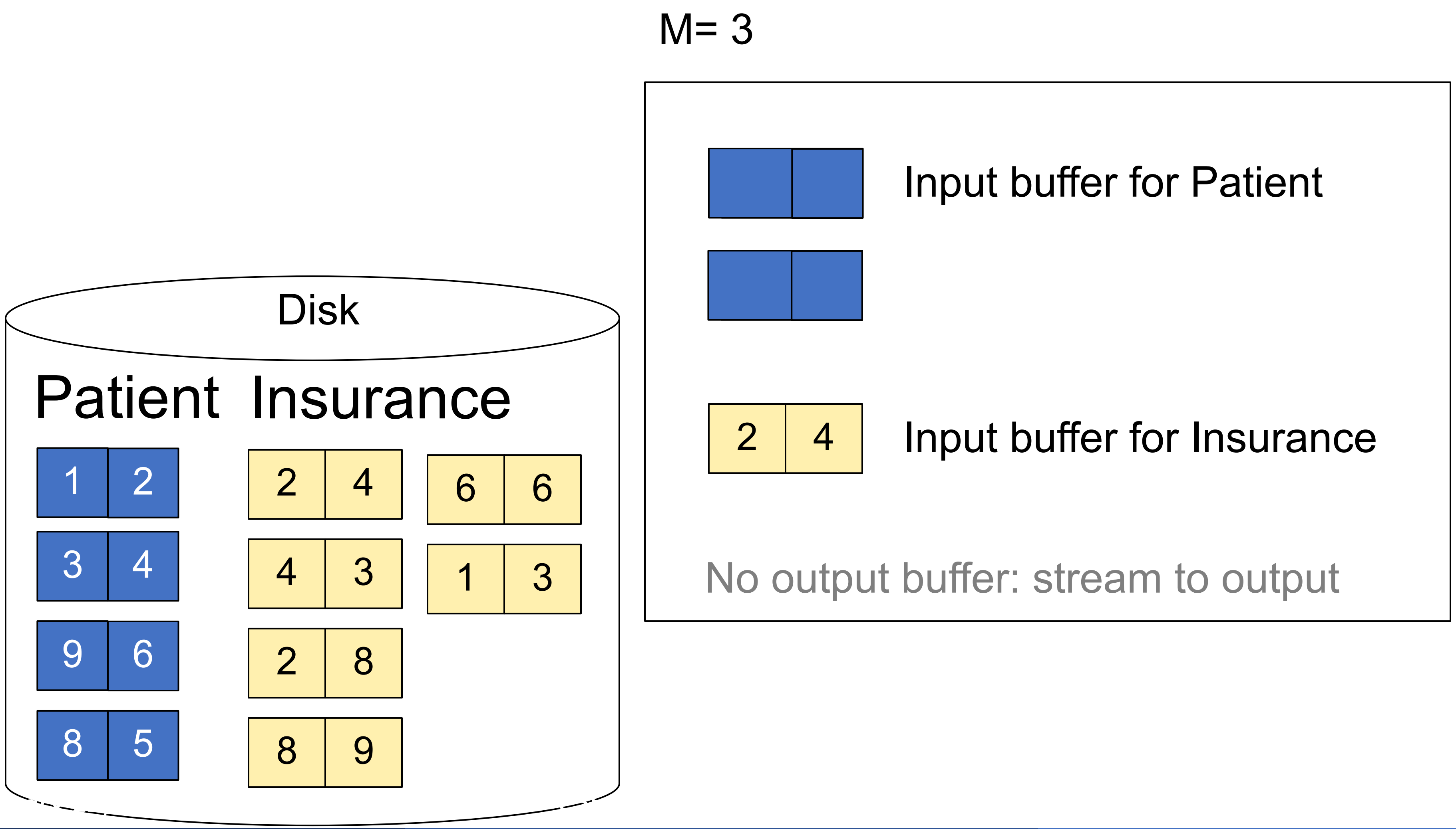
for each group of M-1 pages r in R do
  for each page of tuples s in S do
    for all pairs of tuples t_1 in r, t_2 in s
      if t_1 and t_2 join then output (t_1,t_2)

What is the Cost?

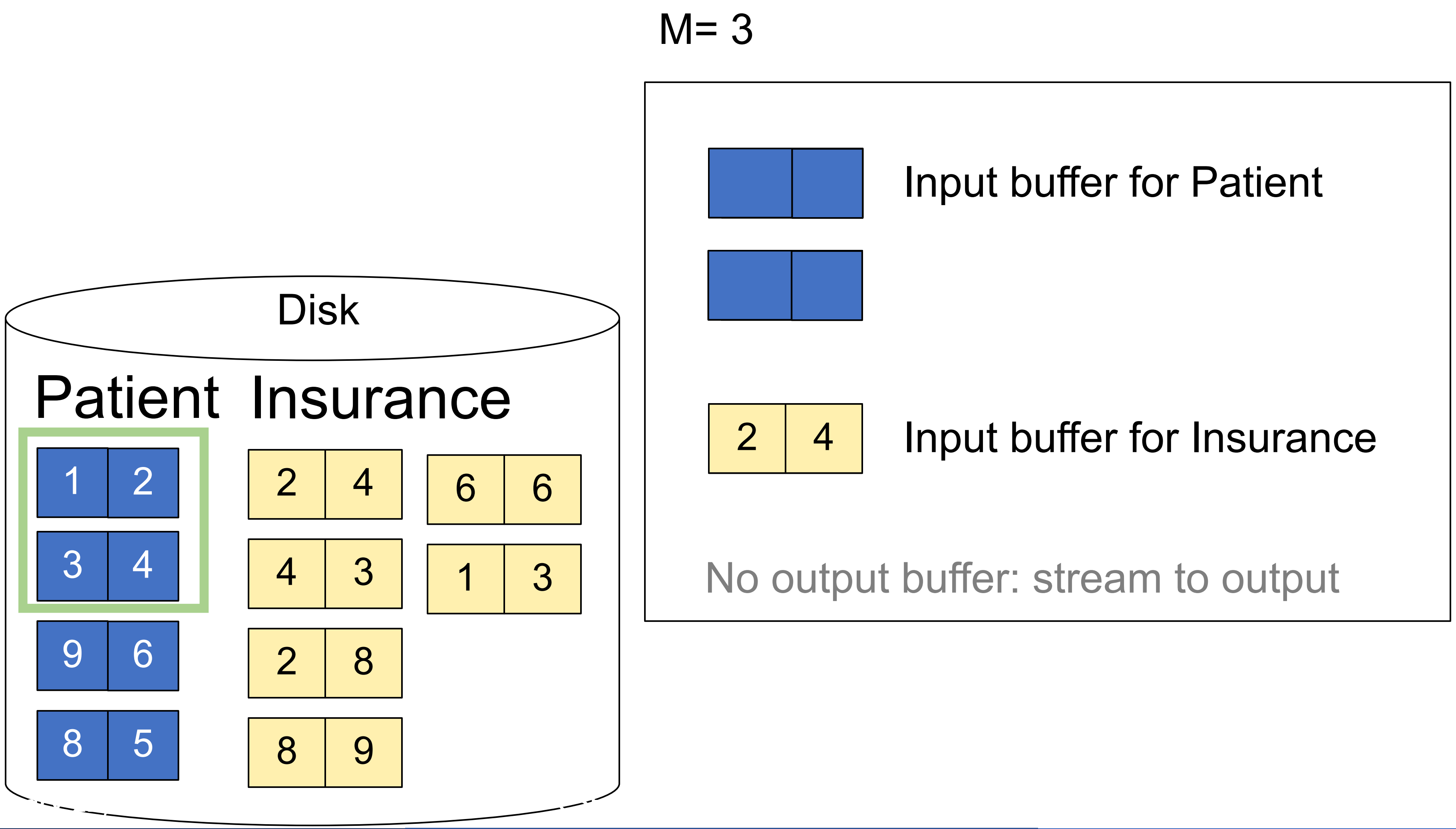
Block Memory Refinement



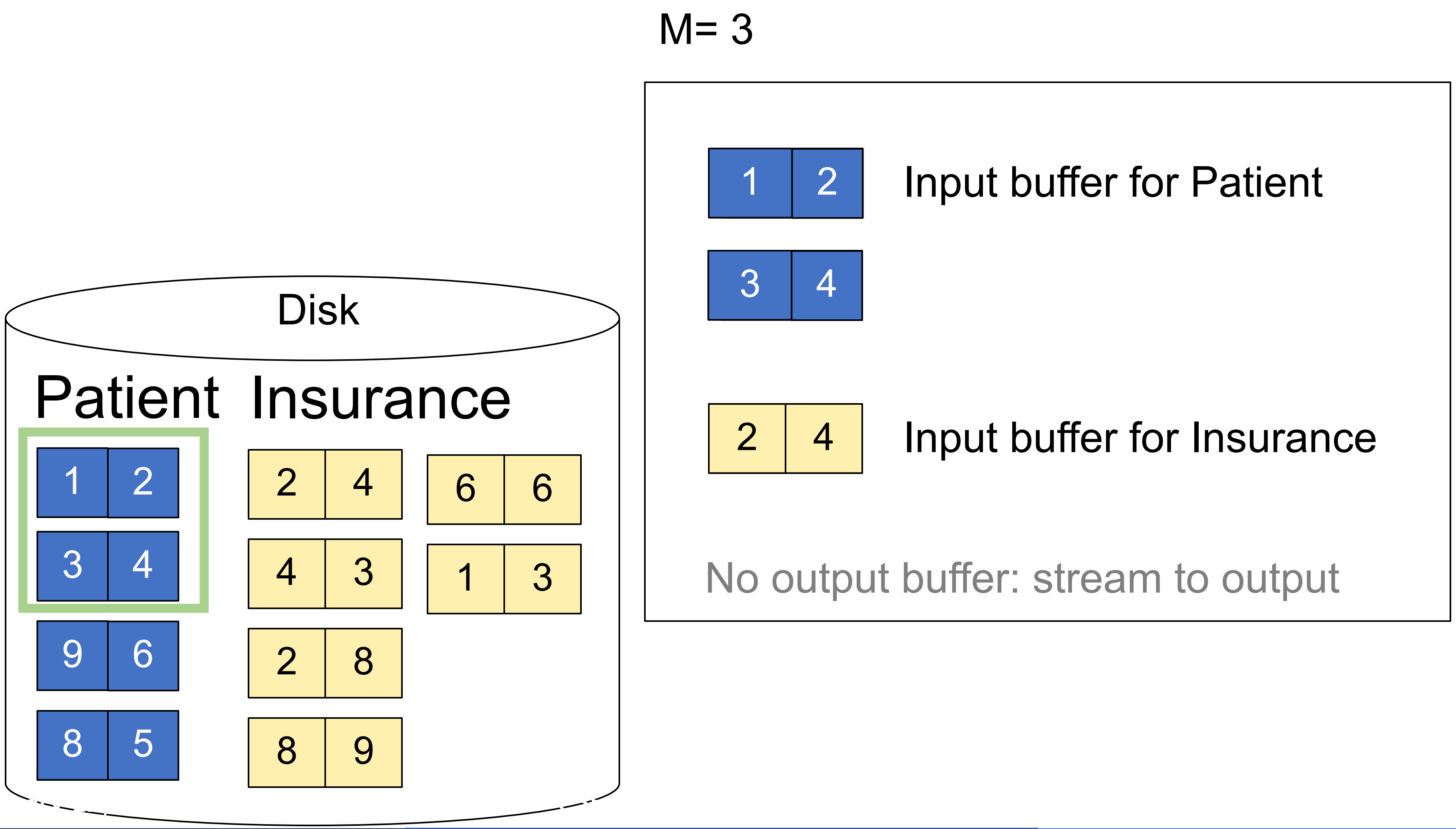
Block Memory Refinement



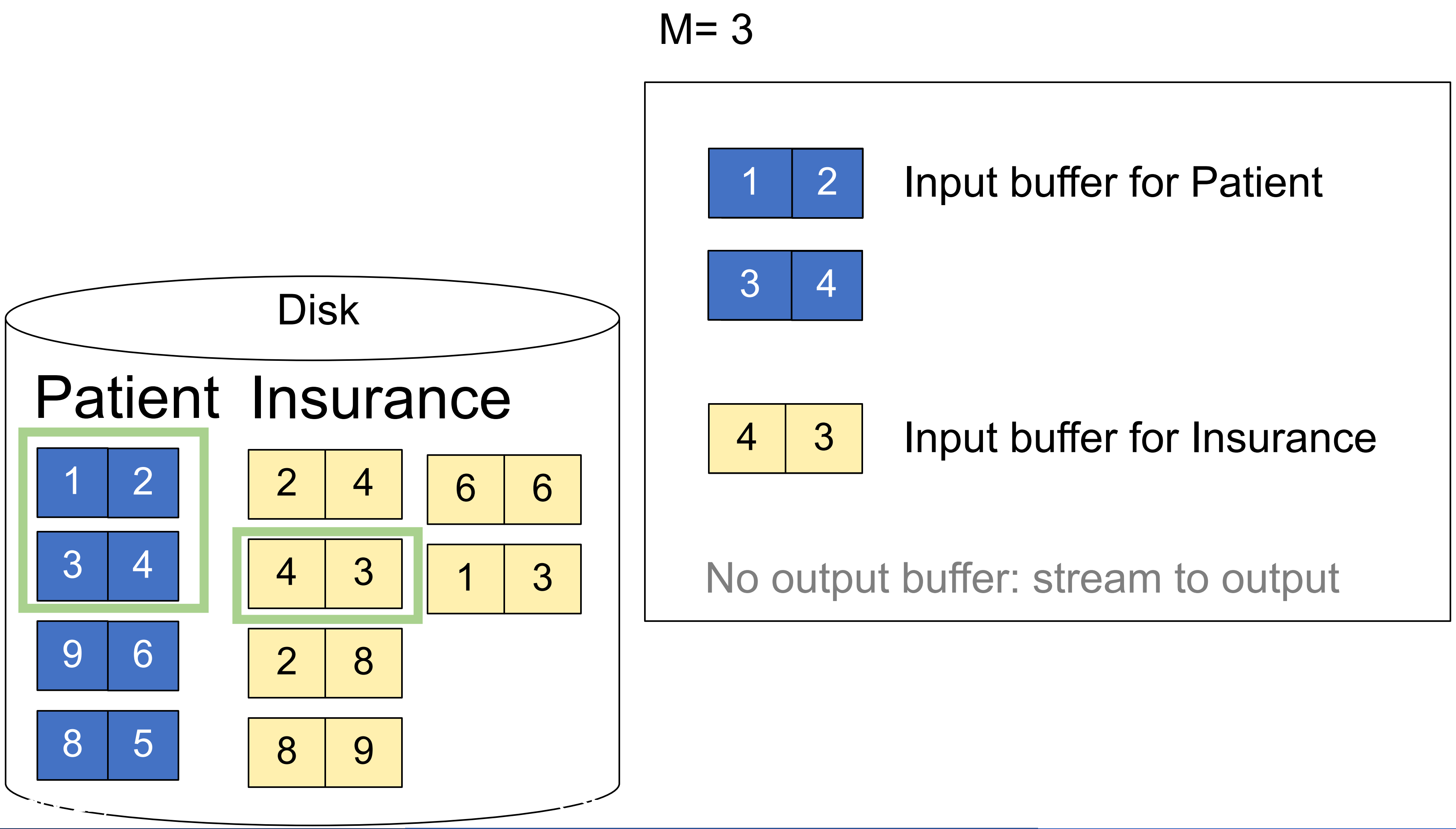
Block Memory Refinement



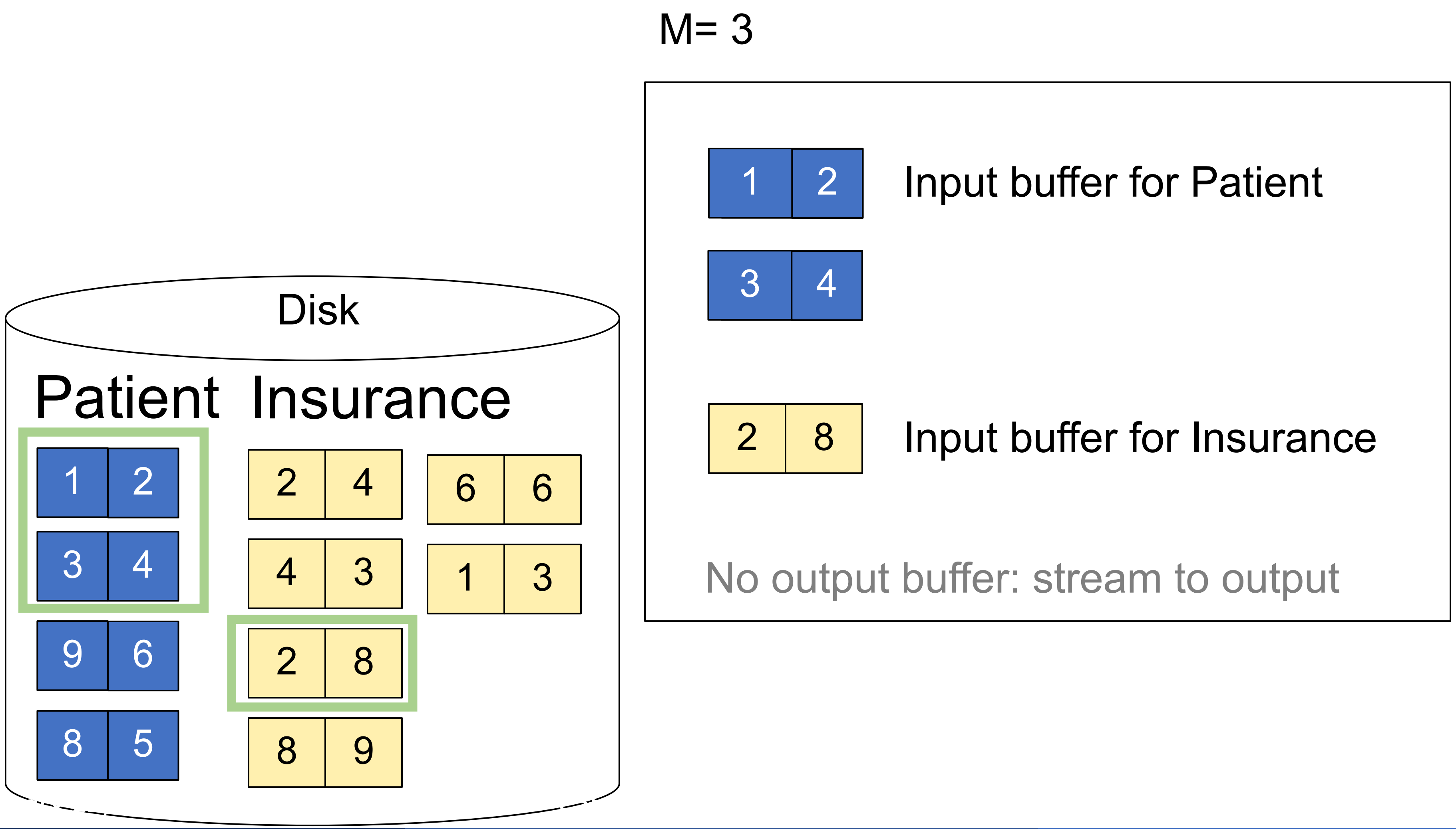
Block Memory Refinement



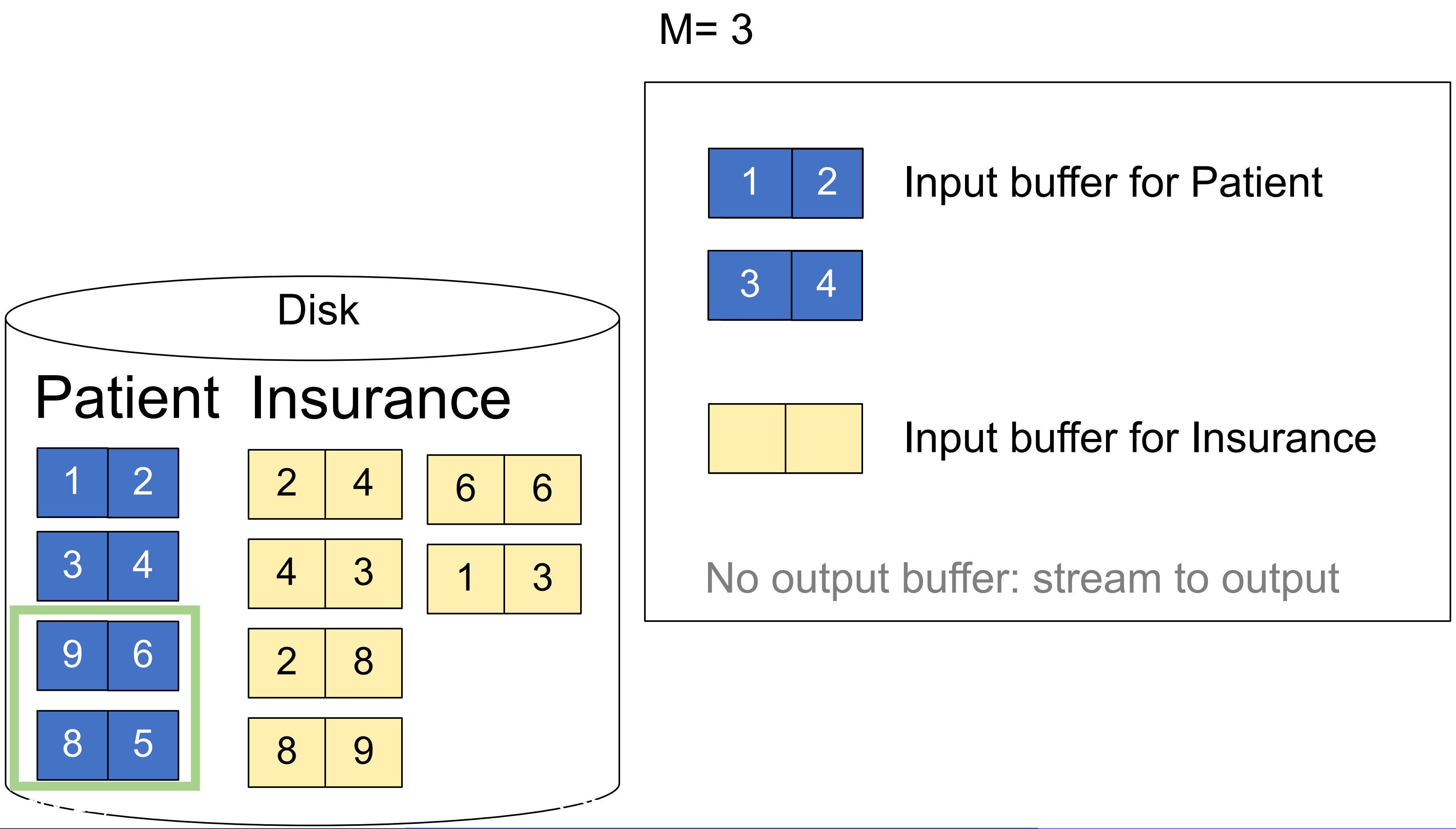
Block Memory Refinement



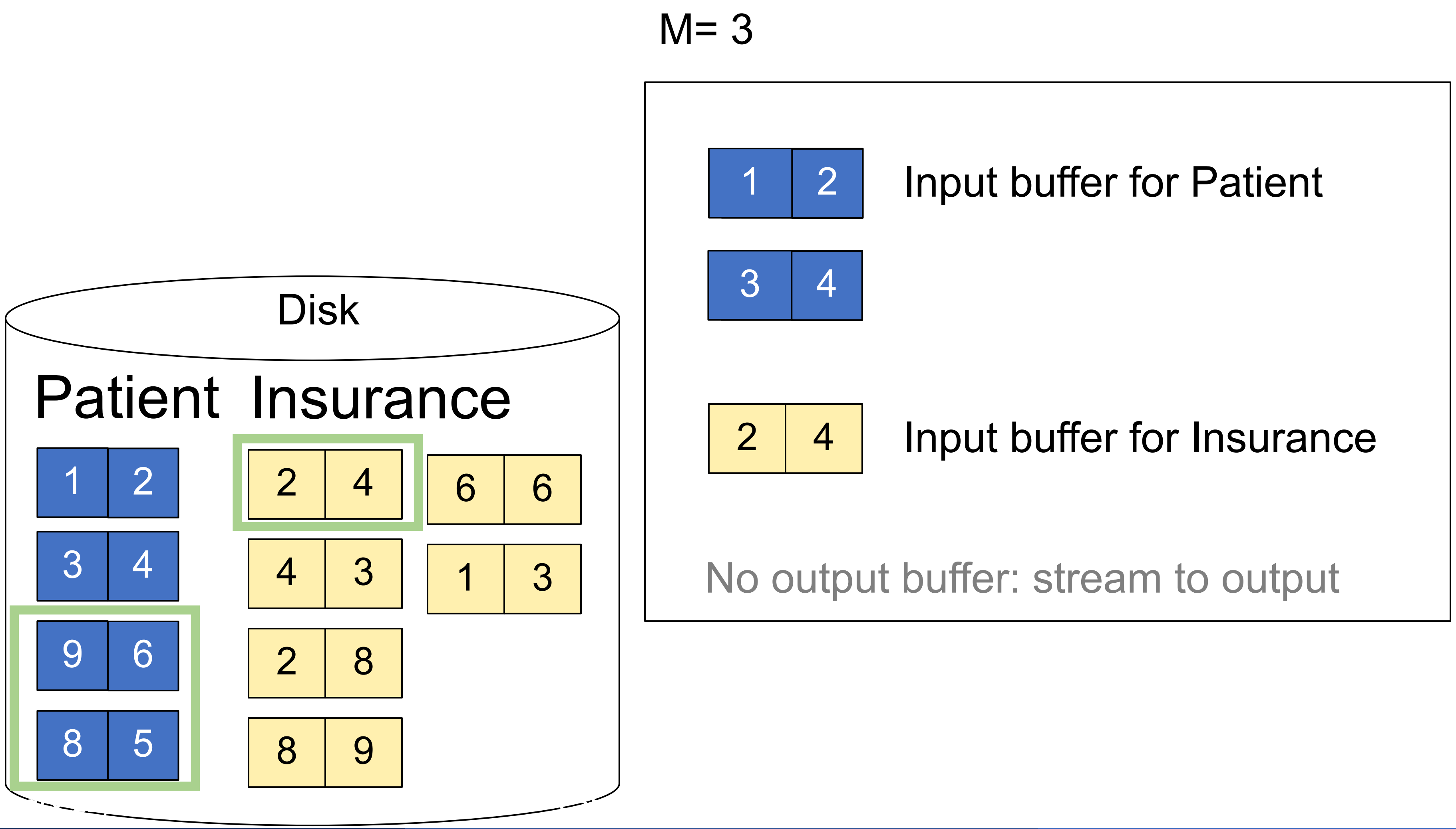
Block Memory Refinement



Block Memory Refinement



Block Memory Refinement



Block Memory 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)
```

What is the Cost

Block Memory 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

Discussion

$R \bowtie S$: R=outer table, S=inner table

- Tuple-based nested loop join is never used

- Page-at-a-time nested loop join:
  - Usually combined with index access to inner table
  - Efficient when the outer table is small

- Block memory refinement nested loop
  - Usually builds a hash table on the outer table
  - Efficient when the outer table is small

Sort-Merge Join

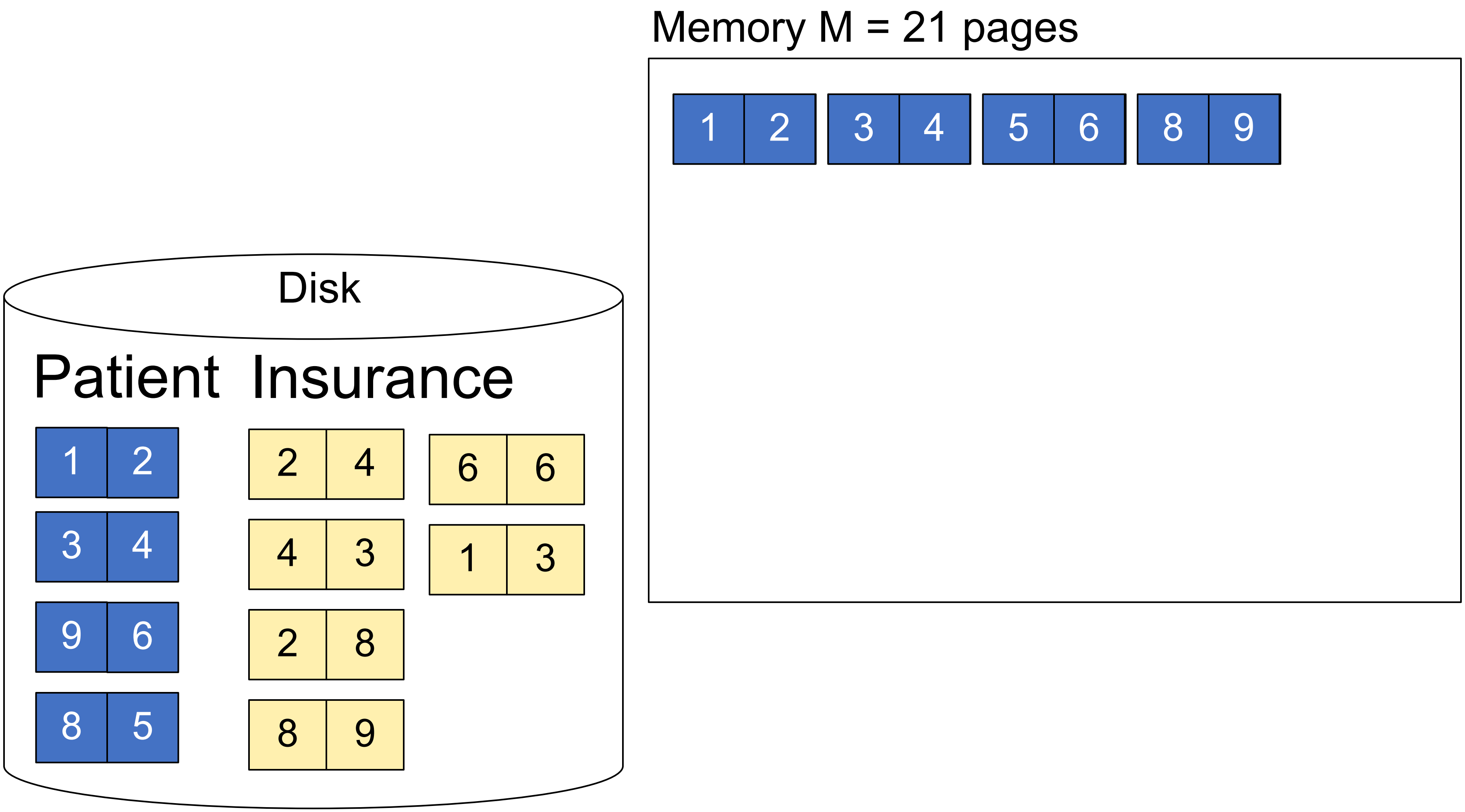
Sort-merge join: $R \bowtie S$

- 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,
  - We'll see the multi-pass version next lecture

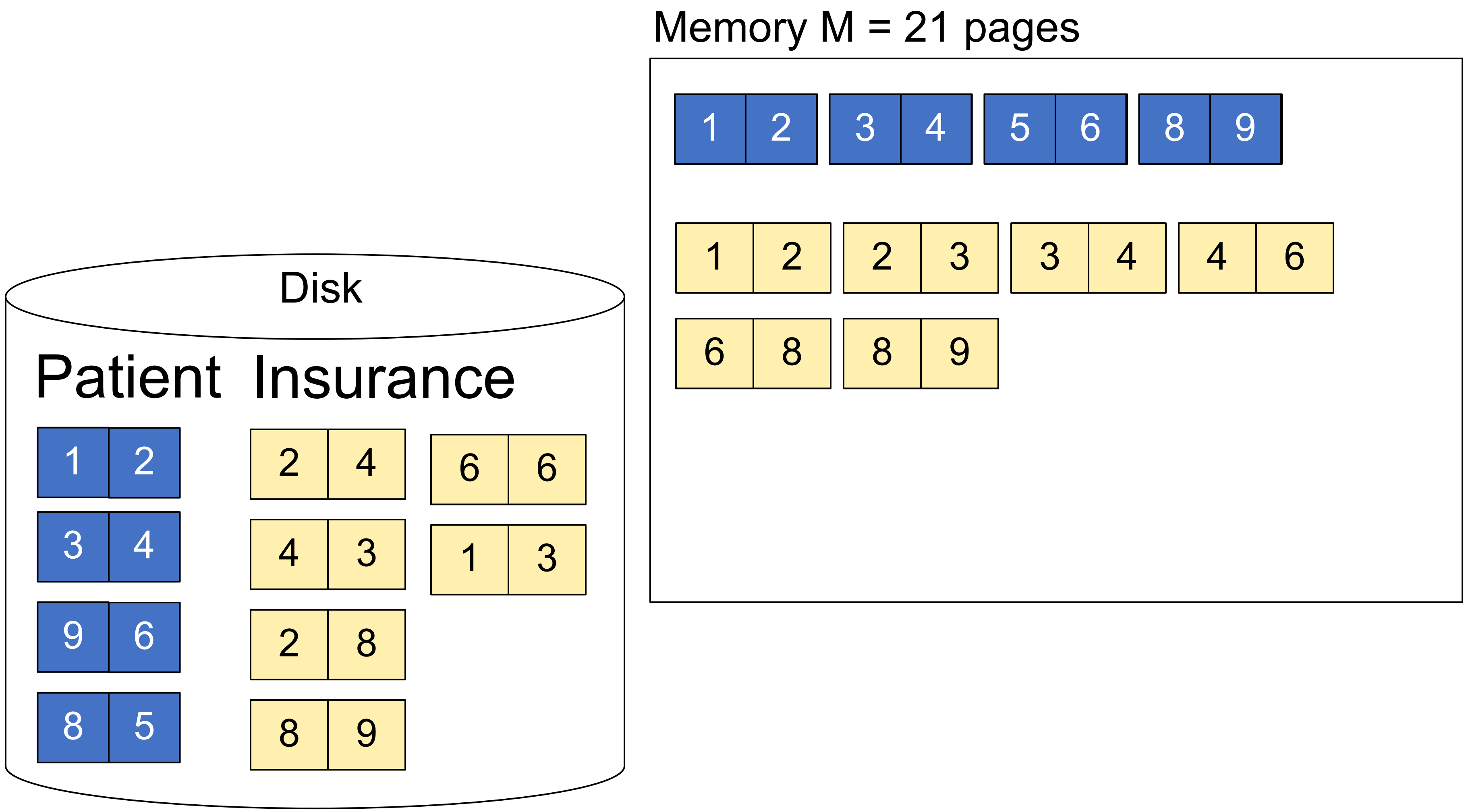
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory



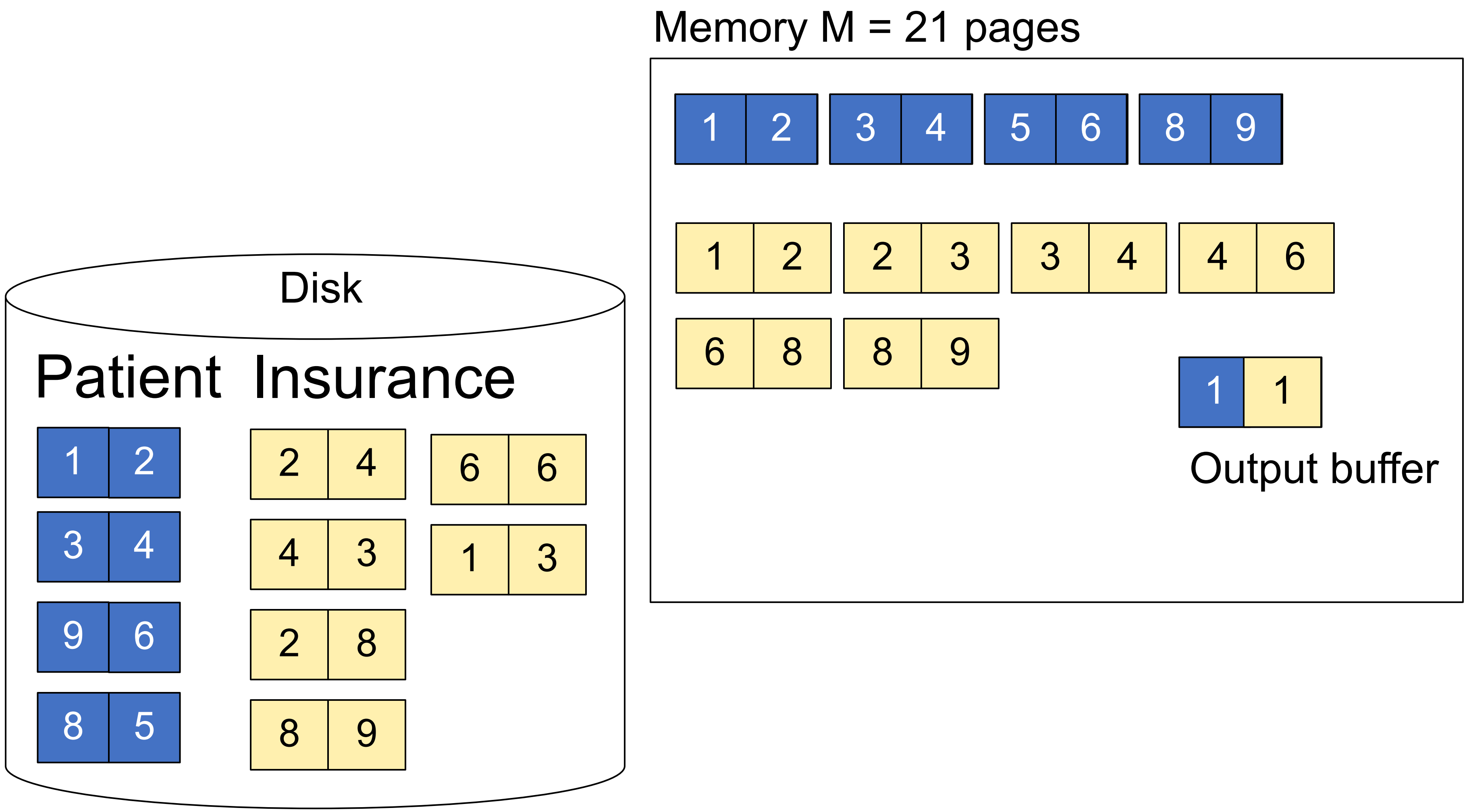
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory



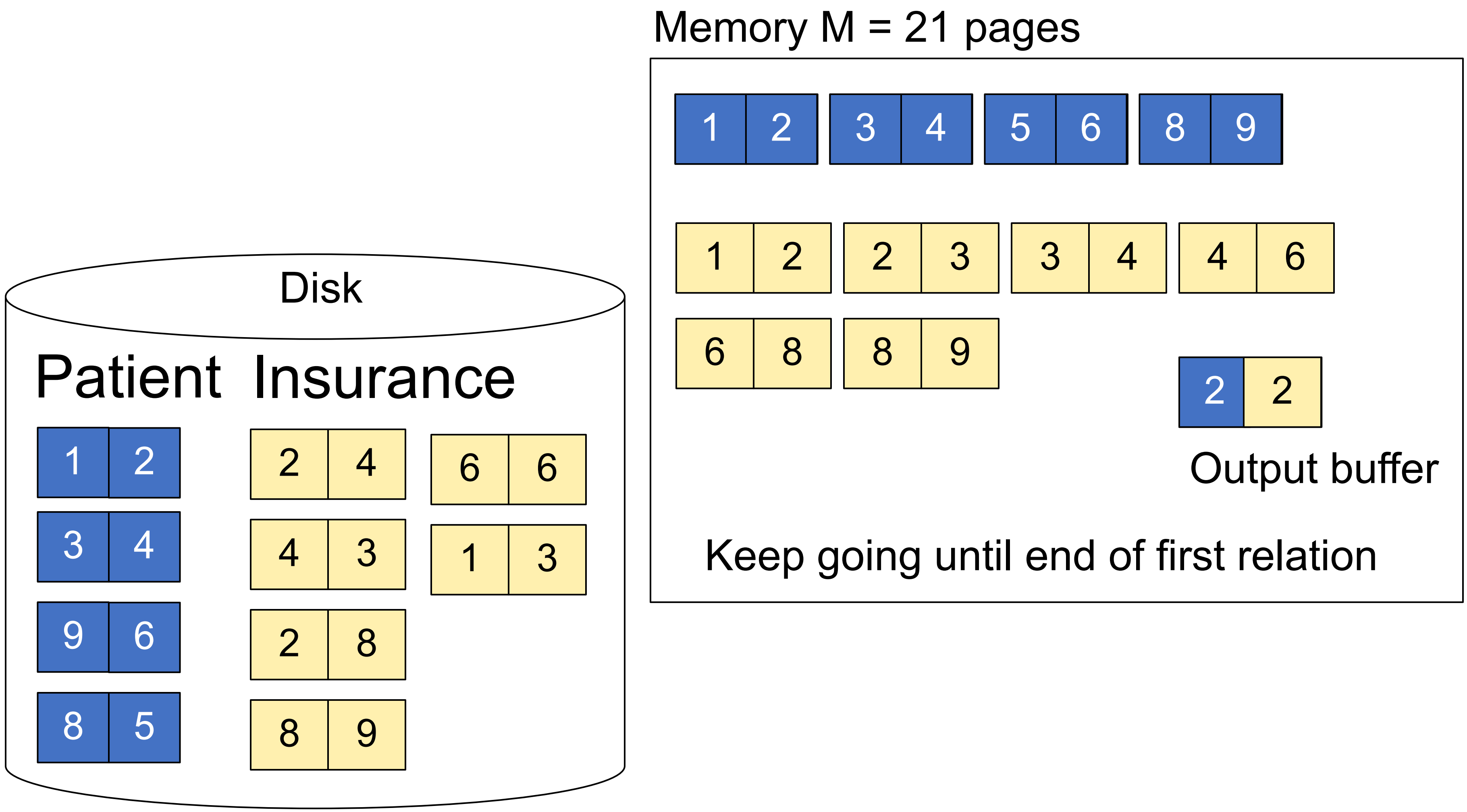
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Outline

- **Join operator algorithms**
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
  - Two-pass algorithms (Sec 15.4 and 15.5)

Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$

- B(R)= size of R in blocks
- T(R) = number of tuples in R
- V(R, a) = # of distinct values of attribute a

Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$

- B(R)= size of R in blocks
- T(R) = number of tuples in R
- V(R, a) = # of distinct values of attribute a

What is the cost in each case?

- Clustered index on a:
- Unclustered index on a:

Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$

- B(R)= size of R in blocks
- T(R) = number of tuples in R
- V(R, a) = # of distinct values of attribute a

What is the cost in each case?

- Clustered index on a: B(R)/V(R,a)
- Unclustered index on a: T(R)/V(R,a)

Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$

- B(R)= size of R in blocks
- T(R) = number of tuples in R
- V(R, a) = # of distinct values of attribute a

What is the cost in each case?

- Clustered index on a: B(R)/V(R,a)
- Unclustered index on a: T(R)/V(R,a)

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

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

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