Version March 15, 2011

Introduction to Database Systems
CSE 444, Winter 2011

Lecture 20: Operator Algorithms

http://www.cs.washington.edu/education/courses/cse444/11wi/

Where we are / and where we go

Feb 7	Transactions: Concurrency Control lecture 14-15 Midterm review on the board	Midterm	Data Storage and Indexing lecture 16 Homework 2 due
Feb 14	Database Tuning lecture 17	Relational Algebra lecture 18	Query Processing Overview lecture 19 Project 3 due
Feb 21	No class (Presidents Day)	Operator Algorithms	Query Optimization Homework 3 due
Feb 28	Query Optimization	Query Optimization	Parallel and Distributed DBMSs
Mar 7	Pig Latin	TBA	Wrap-up Project 4 due
Mar 14	Final Exam Thursday, March 17, 8:30am-10:20am, in class		

Why Learn About Operator Algorithms?

- Implemented in commercial DBMSs
  - DBMSs implement different subsets of known algorithms
- Good algorithms can greatly improve performance
- Need to know about physical operators to understand query optimization

Cost Parameters

- In database systems the data is on disk
- **Cost = total number of I/Os**

- 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)
  - **M** = # of max. pages in main memory

Cost

- Cost of an operation = number of disk I/Os to
  - Read the operands
  - Compute the result

- Cost of writing the result to disk is *not included*
  - Need to count it separately when applicable

Cost of Scanning a Table

- Result may be unsorted: B(R)
- Result needs to be sorted: 3 B(R)
  - We will discuss sorting later

Outline for Today

- 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 will discuss only join operator algorithms (because other operators are easier)
- Read the book to get more details about these algos and about algots for other operators

Basic Join Algorithms

- Logical operator:
  - Product(pname, cname) $\bowtie$ Company(cname, city)
- Propose three physical operators for the join, assuming the tables are in main memory:
  - Hash join
  - Nested loop join
  - Sort-merge join

1. Hash Join

Hash join: R ⋈ 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
  - By "one pass", we mean that the operator reads its operands only once. It does not write intermediate results back to disk.

1. Hash Join Example

Patient ⋈ Insurance

Patient(pid, name, address)
Insurance(pid, provider, policy_nb)

Two tuples per page

Disk

Patient

1	'Bob'	'Seattle'
2	'Ela'	'Everett'

3	'Jill'	'Kent'
4	'Joe'	'Seattle'

...

Insurance

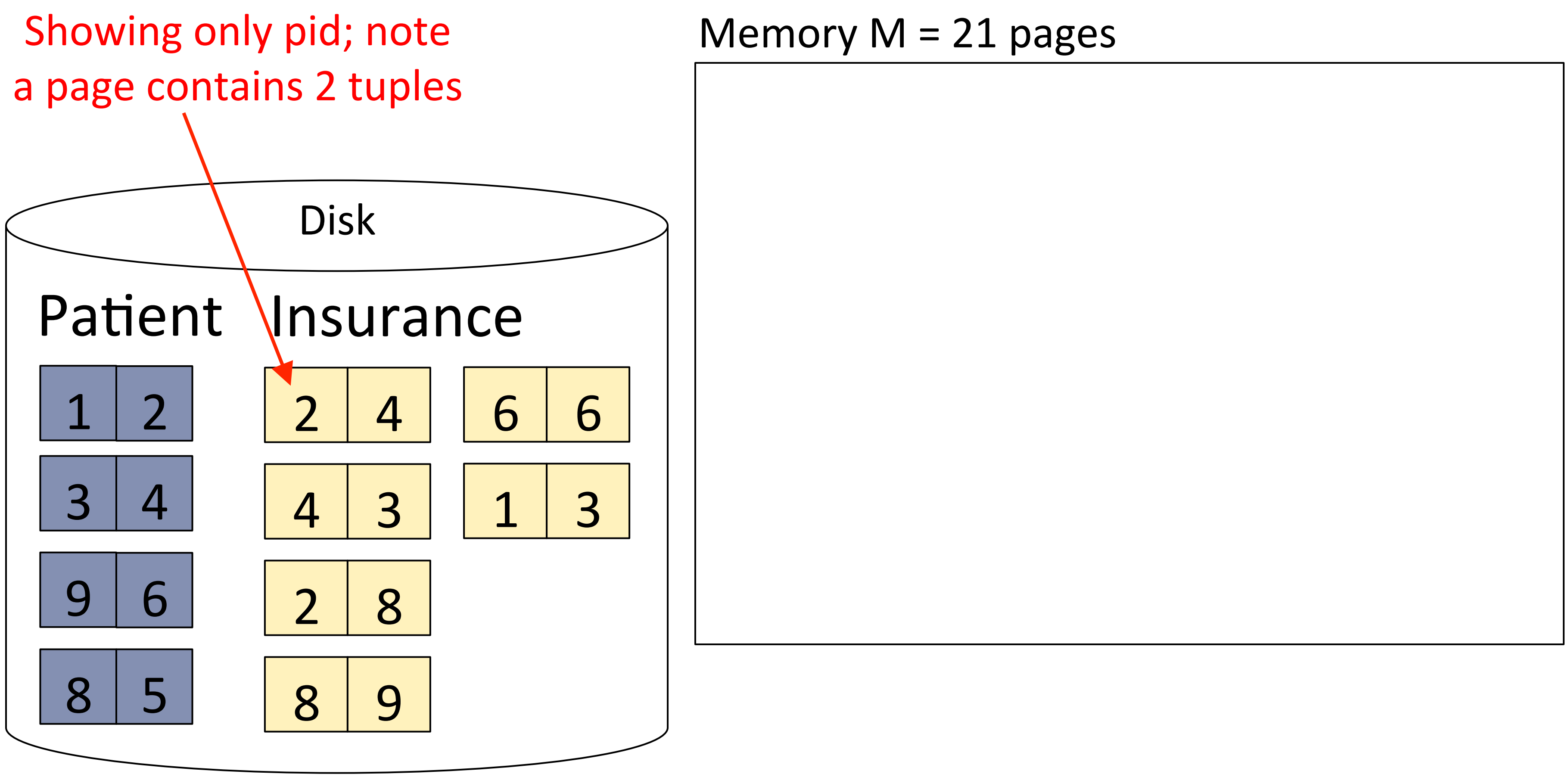
2	'Blue'	123
4	'Prem'	432

4	'Prem'	343
3	'GrpH'	554

...

1. Hash Join Example

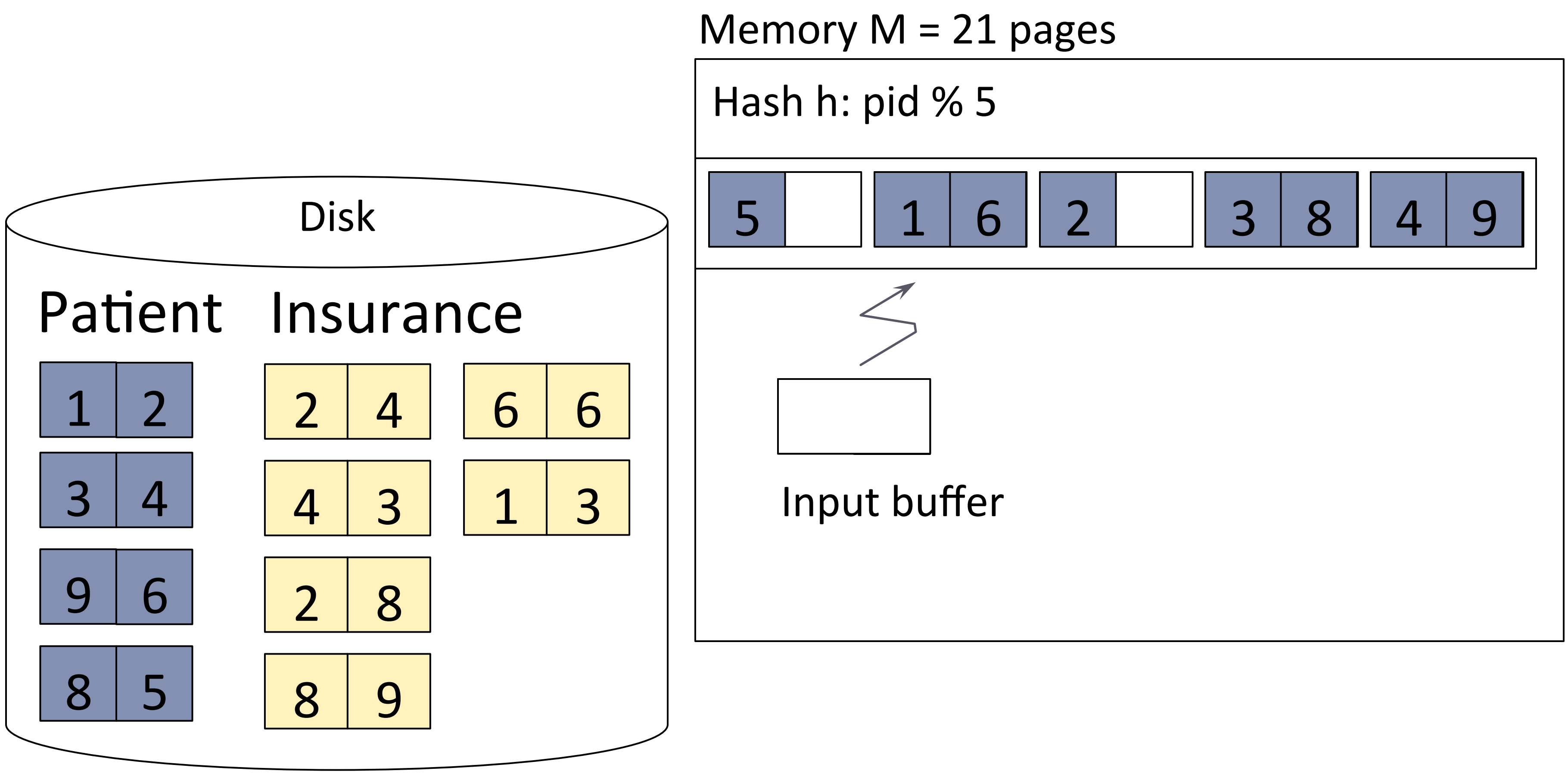
Patient ⋈ Insurance



1. Hash Join Example

Patient ⋈ Insurance

Step 1: Scan Patient and create hash table in memory

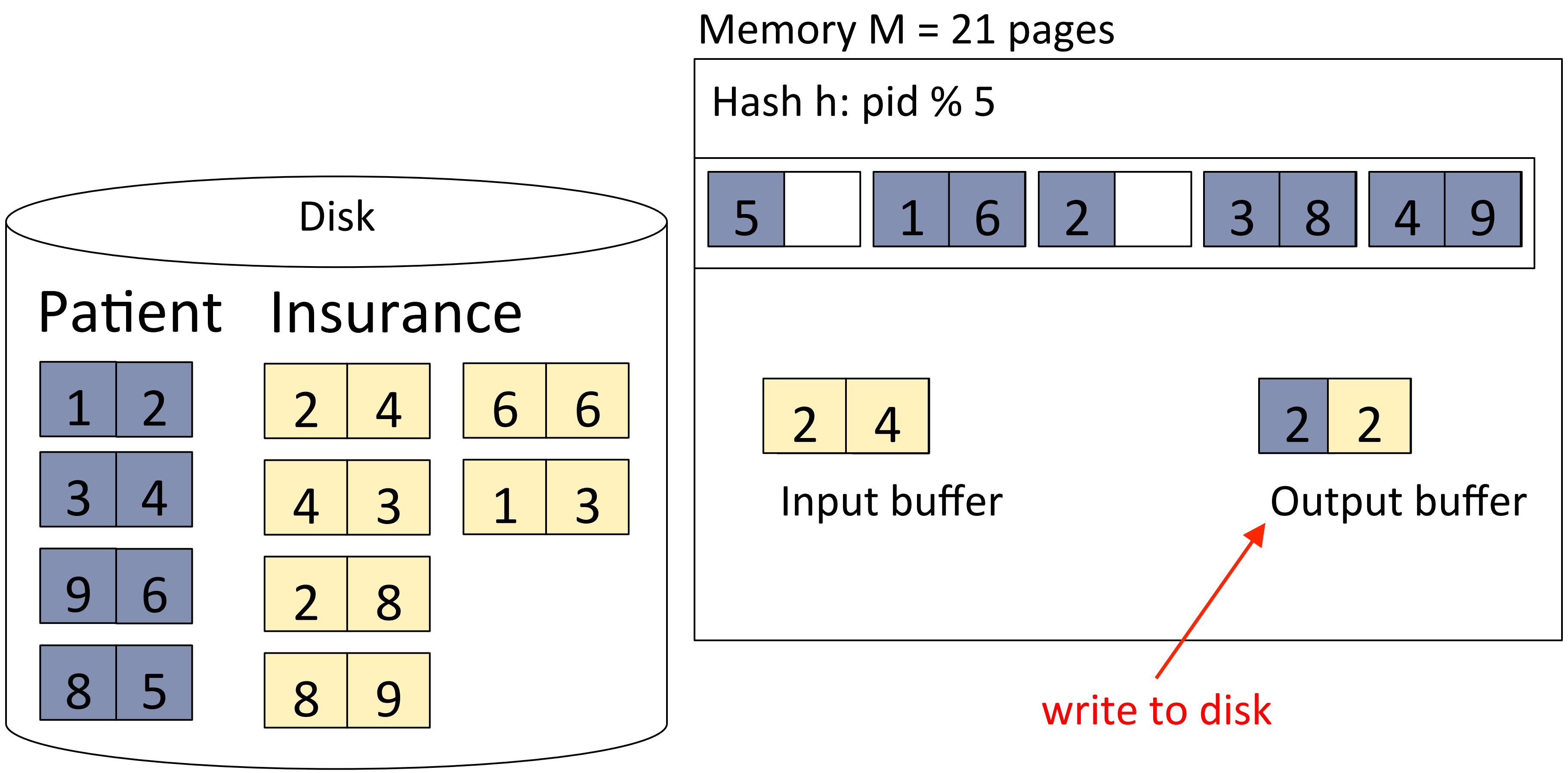


1. Hash Join Example

Patient ⋈ Insurance

Step 1: Scan Patient and create hash table in memory

Step 2: Scan Insurance and probe into hash table

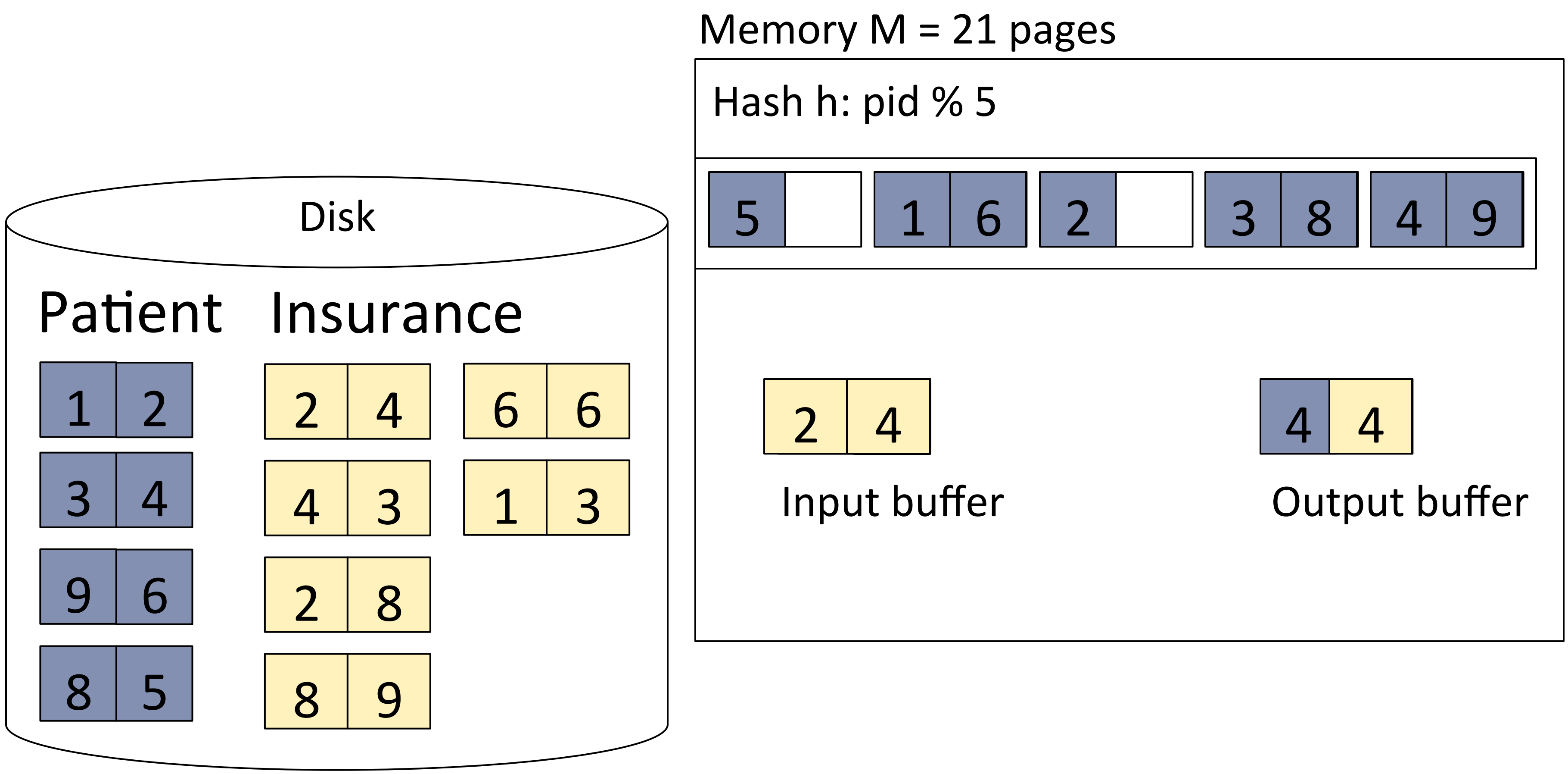


1. Hash Join Example

Patient ⋈ Insurance

Step 1: Scan Patient and create hash table in memory

Step 2: Scan Insurance and probe into hash table

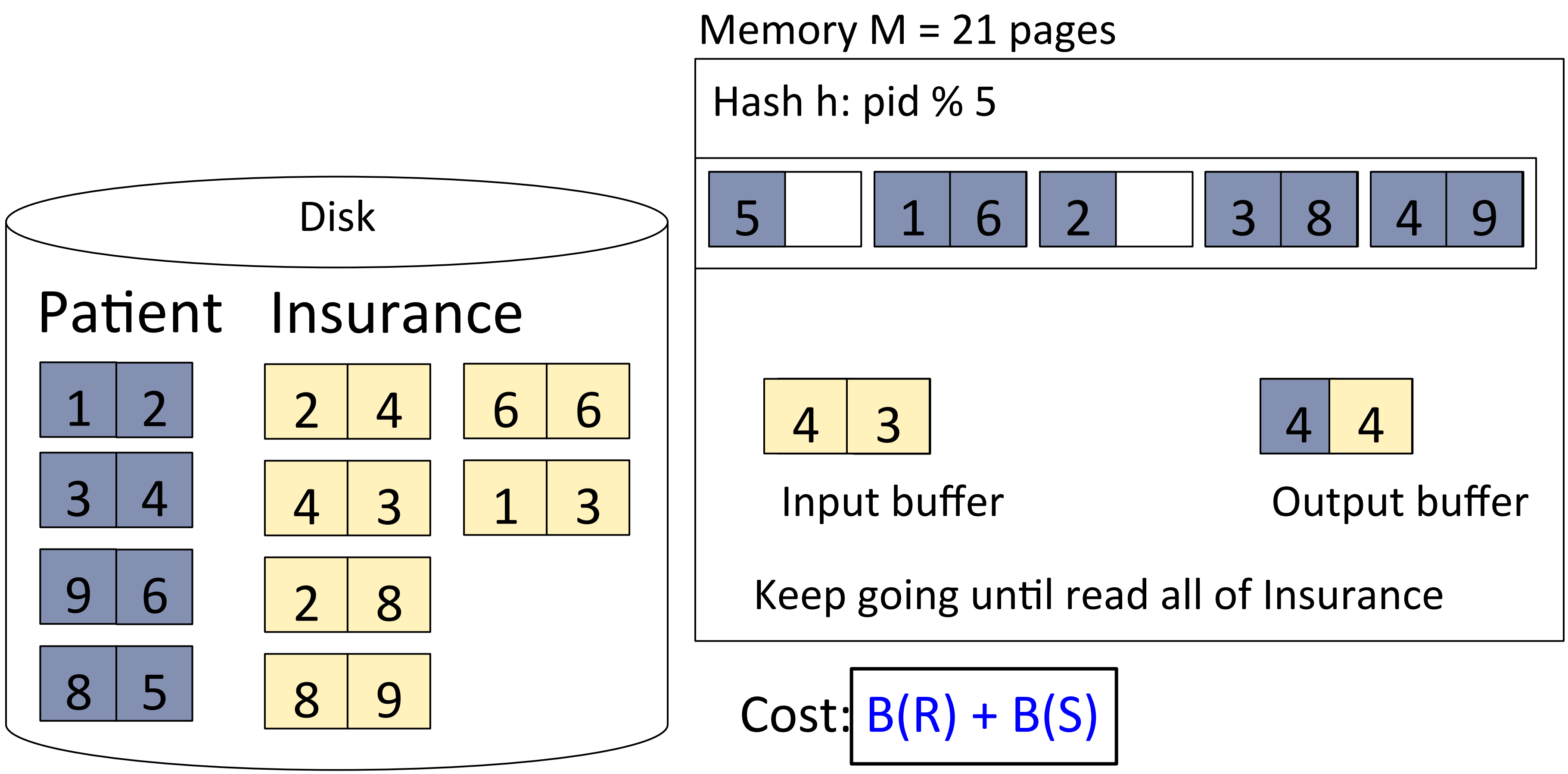


1. Hash Join Example

Patient ⋈ Insurance

Step 1: Scan Patient and create hash table in memory

Step 2: Scan Insurance and probe into hash table



Cost: B(R) + B(S)

1. Hash Join Details

```
Open( ) {
    H = newHashTable( );
    R.Open( );
    x = R.GetNext( );
    while (x != null) {
        H.insert(x);
        x = R.GetNext( );
    }
    R.Close( );
    S.Open( );
    buffer = [ ];
}
```


x

1. Hash Join Details

```
GetNext( ) {
    while (buffer == [ ]) {
        x = S.GetNext( );
        if (x==Null) return NULL;
        buffer = H.find(x);
    }
    z = buffer.first( );
    buffer = buffer.rest( );
    return z;
}
```


x

1. Hash Join Details

```
Close( ) {
    release memory (H, buffer, etc.);
    S.Close( )
}
```


x

2. Nested Loop Joins

Tuple-based nested loop $R \bowtie S$

- R is the outer relation, S is the inner relation

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

- Cost: B(R) + T(R) B(S)
- One-pass only over outer relation
  - But S is read many times

x

2. Page-at-a-time Refinement

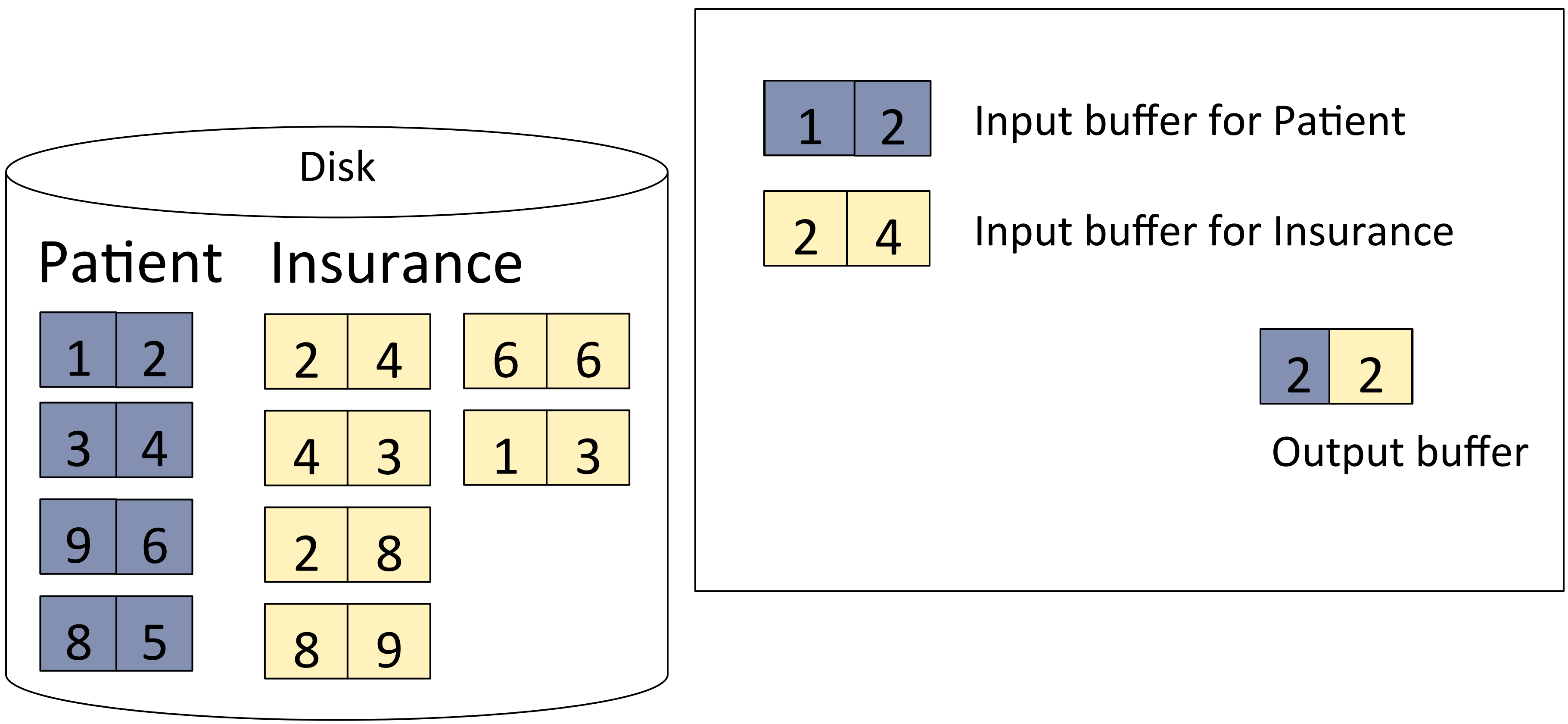
```
for each page of tuples r in R do
  for each page of tuples s in S do
    for all pairs of tuples
      if r and s join then output (r,s)
```

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

~~T(R)~~ ↑

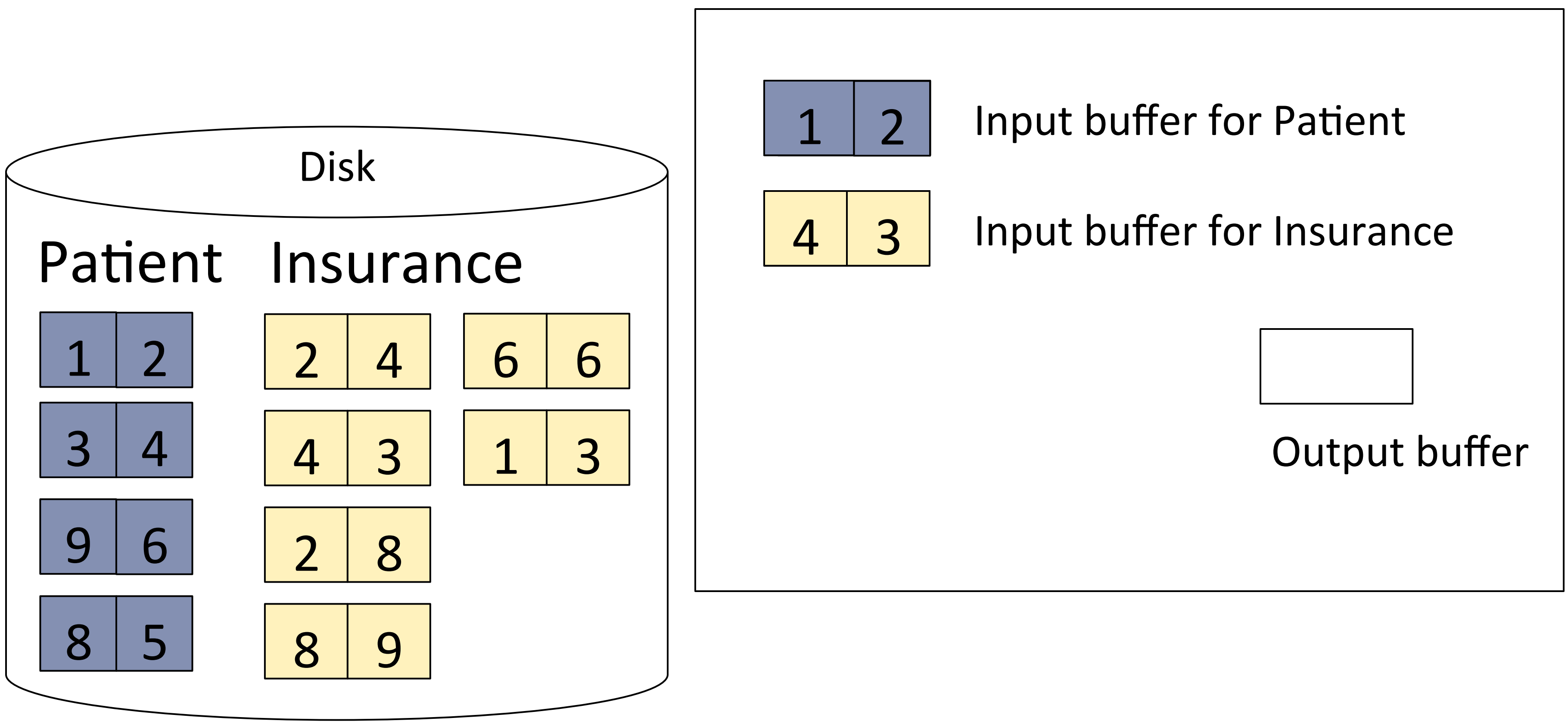
2. Nested Loop Example

Patient ⋈ Insurance



2. Nested Loop Example

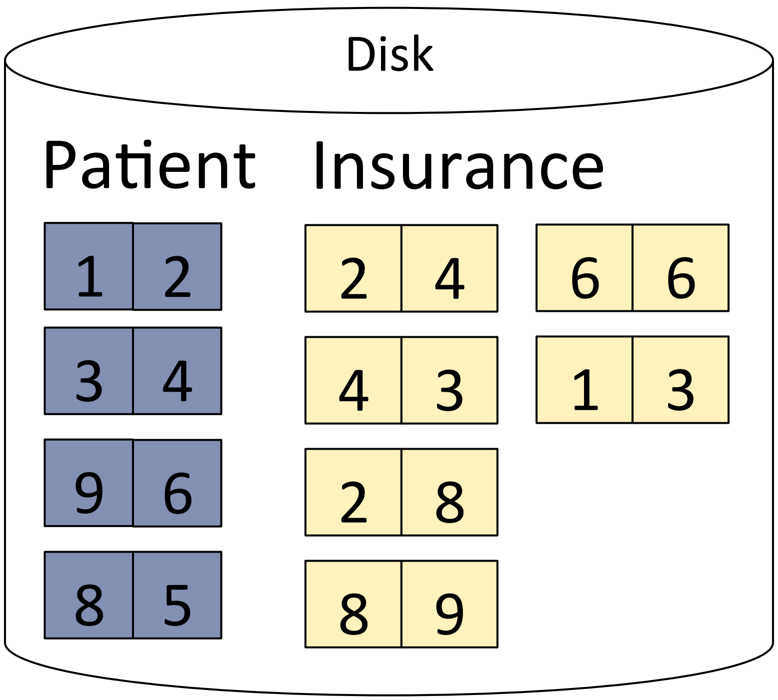
Patient ⋈ Insurance

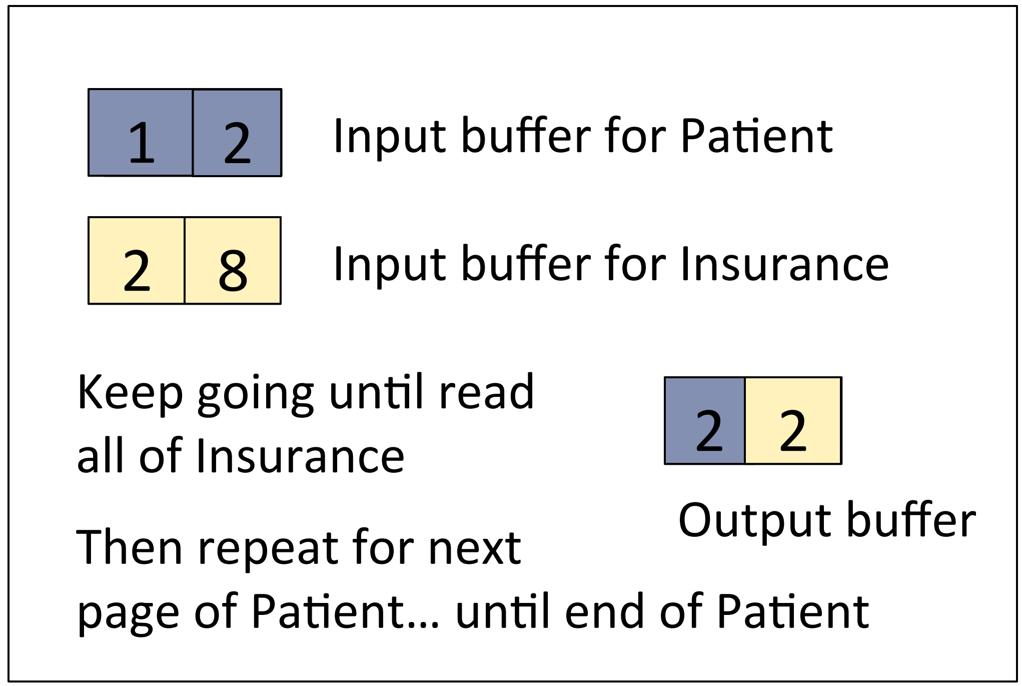


http://www.cs.washington.edu/education/courses/cse444/11wi/

2. Nested Loop Example

Patient ⋈ Insurance

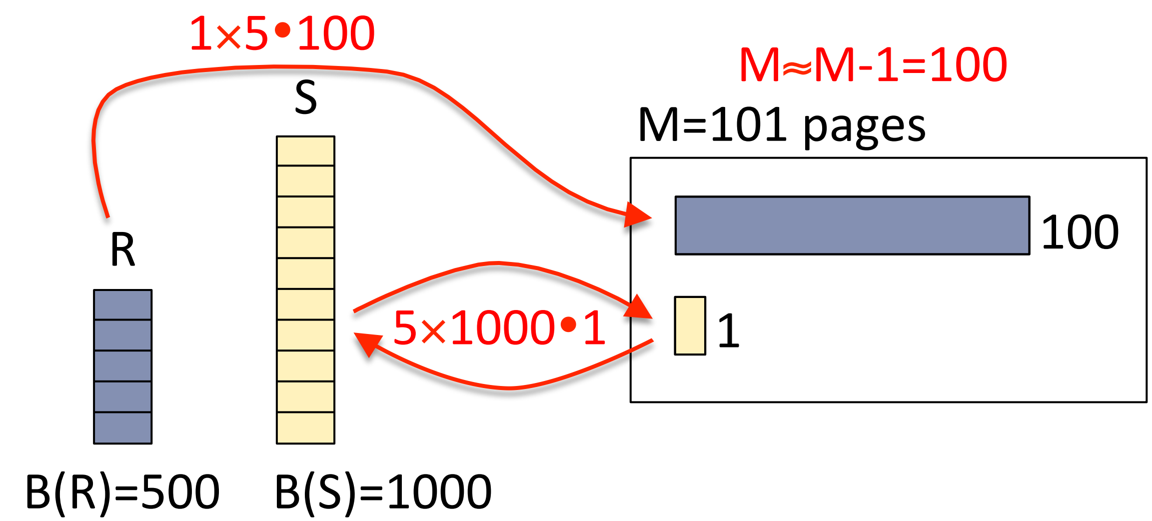




Cost: B(R) + B(R) B(S)

2b. Nested-block join (Nested-loop join)

Book example 15.4



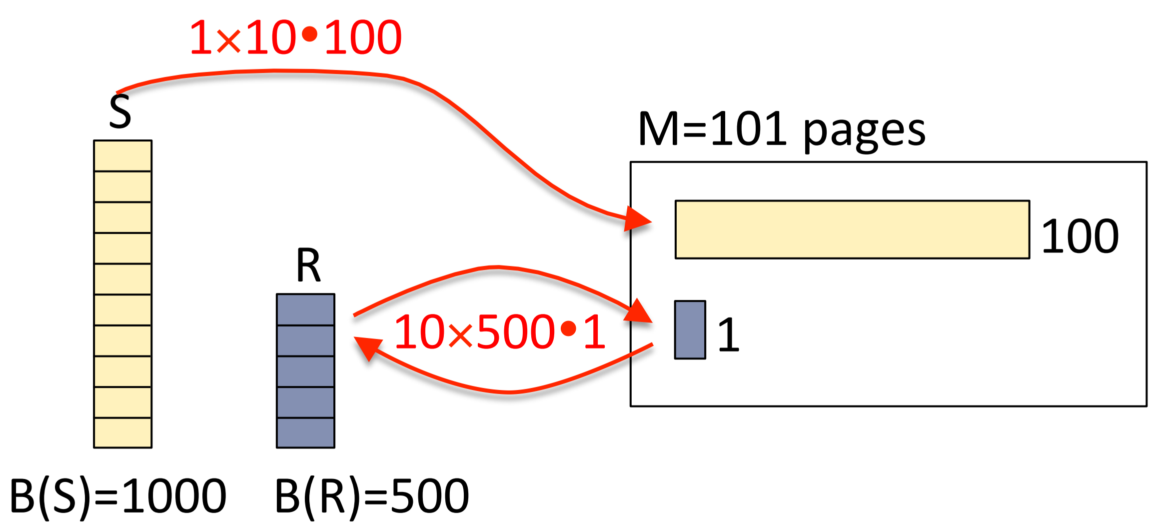
Cost R: 500

Cost S: 5000 = 5•1000

SUM: **5500**

B(R) + B(R)/M•B(S)

500 + (500/100)•1000 = **5500**



Cost S: 1000

Cost R: 5000 = 10•500

SUM: **6000**

B(S) + B(R)/M•B(S)

1000 + (1000/100)•500 = **6000**

x

3. Sort-Merge Join

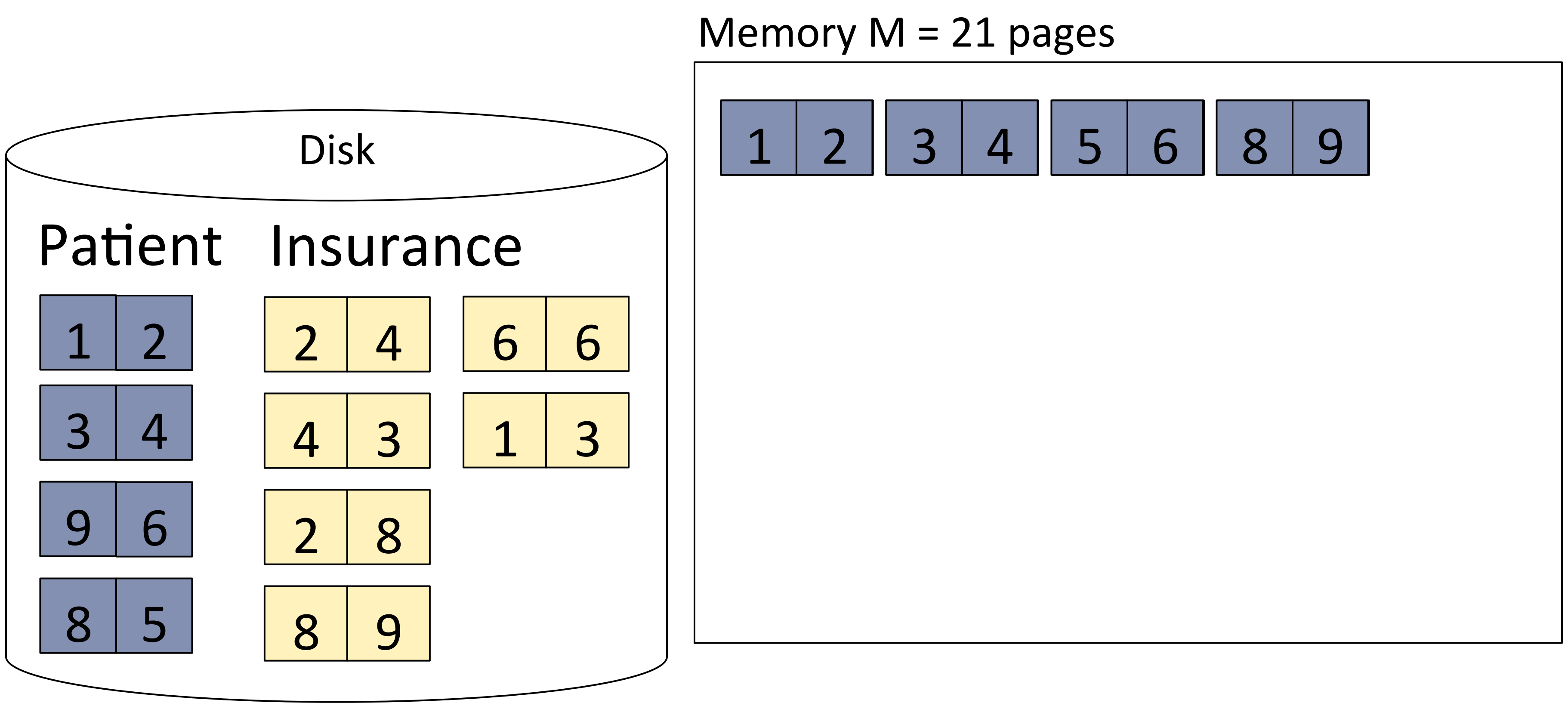
Sort-merge join: R ⋈ S

- 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

3. Sort-Merge Join Example

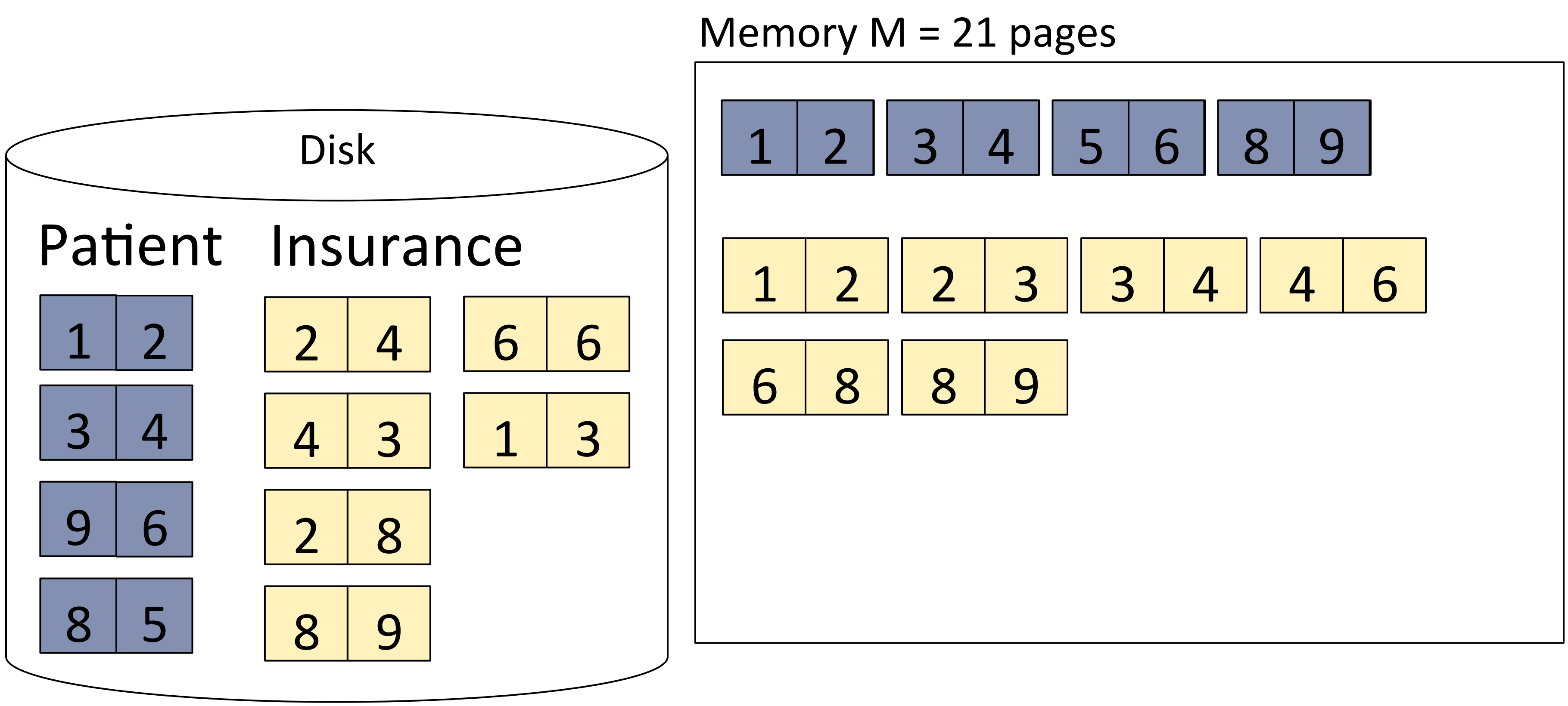
Step 1: Scan Patient and sort in memory



3. Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

Step 2: Scan Insurance and sort in memory

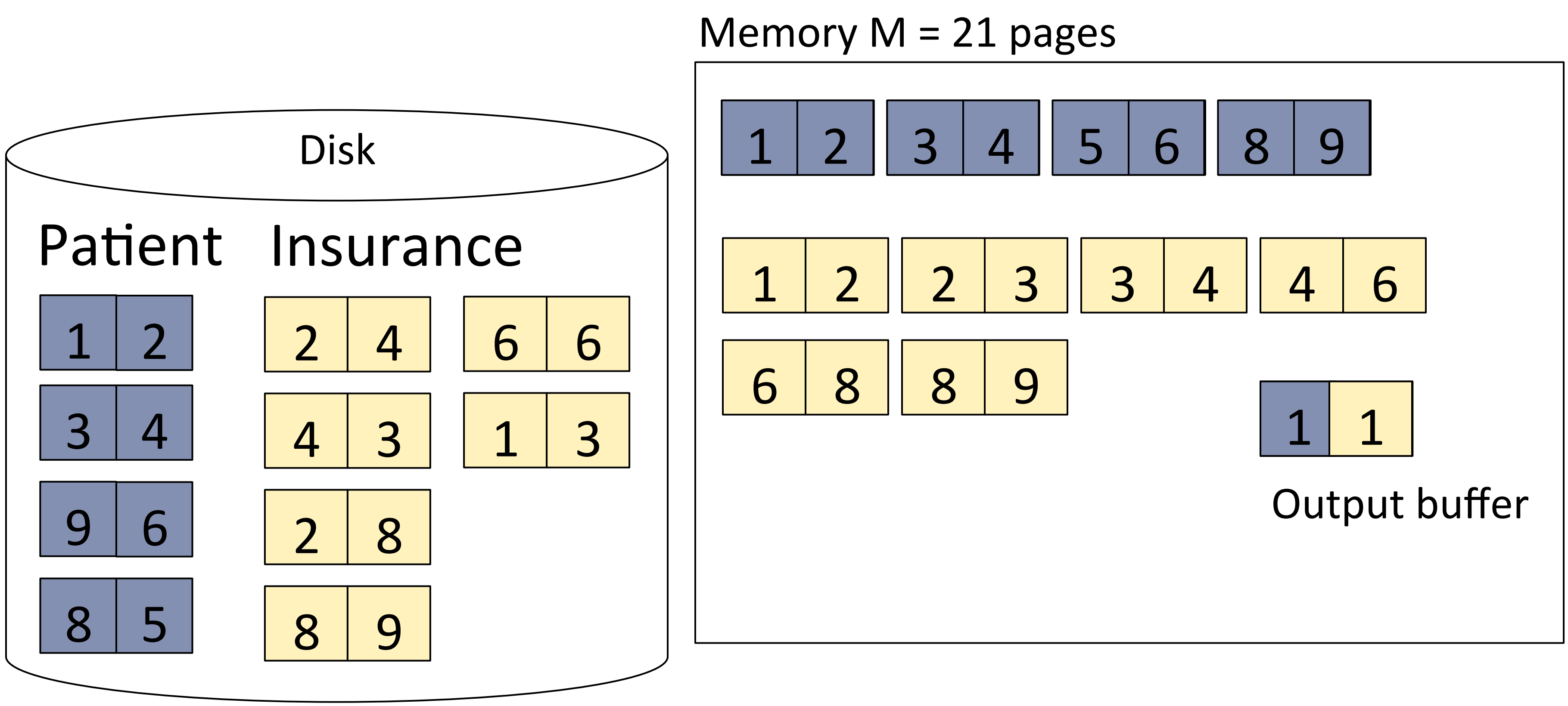


3. Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

Step 2: Scan Insurance and sort in memory

Step 3: Merge Patient and Insurance

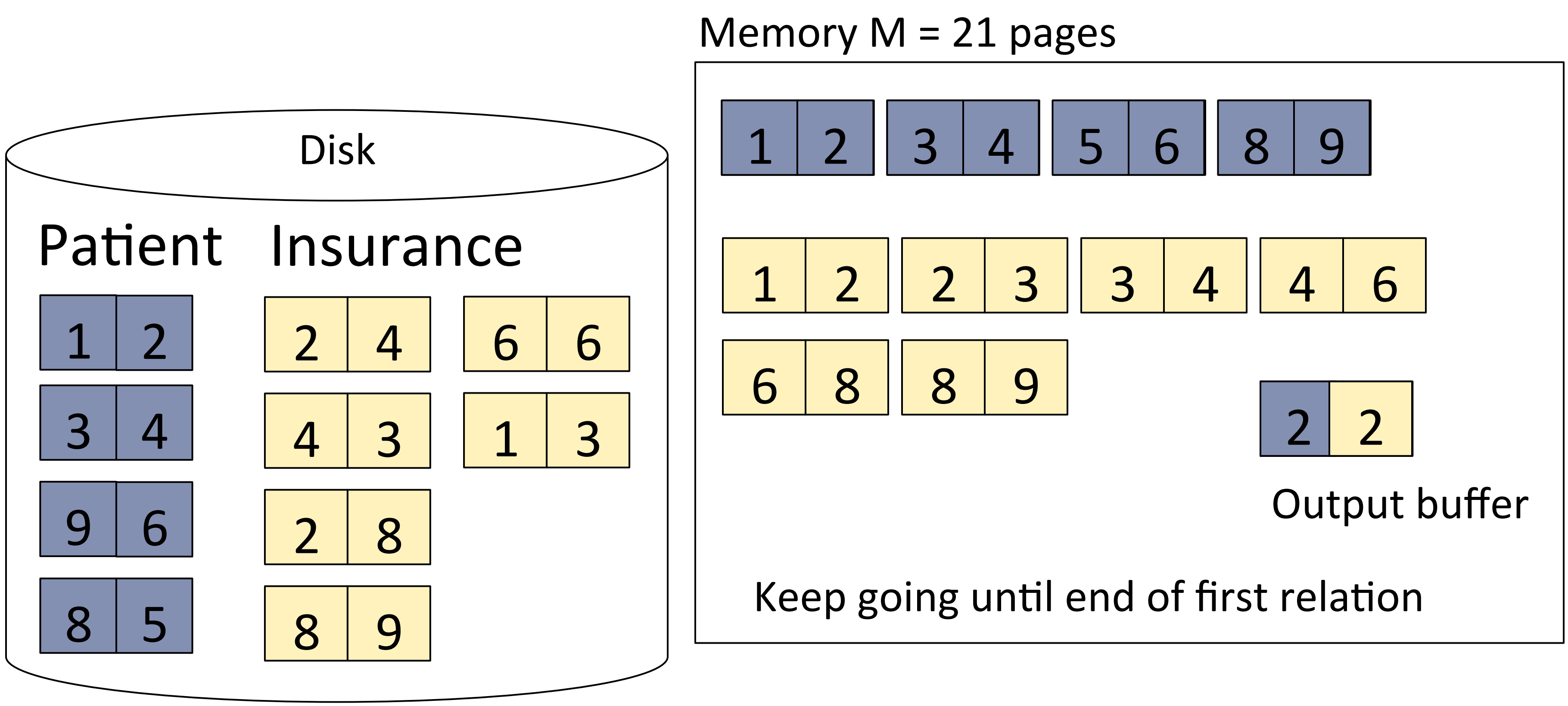


3. Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

Step 2: Scan Insurance and sort in memory

Step 3: Merge Patient and Insurance



Outline for Today

- 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)

Review: Access Methods

- **Heap file**
  - Scan tuples one at the time
- **Hash-based index**
  - Efficient selection on equality predicates
  - Can also scan data entries in index
- **Tree-based index**
  - Efficient selection on equality or range predicates
  - Can also scan data entries in index

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- V(R,a) = # of distinct values of attribute a
- Cost Clustered index on a: B(R)/V(R,a) — Expected number of pages
- Cost Unclustered index on a: T(R)/V(R,a) — Expected number of tuples
- Note: we ignored I/O cost for index pages

x

Index Based Selection

- Example:

T(R) =	100,000
B(R) =	2,000
V(R, a) =	20

Cost of $s_{a=v}(R)$ = ?

- 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

Expected # of pages for expected # of tuples

Expected # of tuples

- Lesson
  - Don't build unclustered indexes when V(R,a) is small, i.e. many tuples with same attribute values a (here 5,000)!

x

4. Index Nested Loop Join

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: $B(R) + T(R)\, T(S)/V(S,a)$

Expected number of tuples from S that join with a tuple from R

x

Outline for Today

- 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)

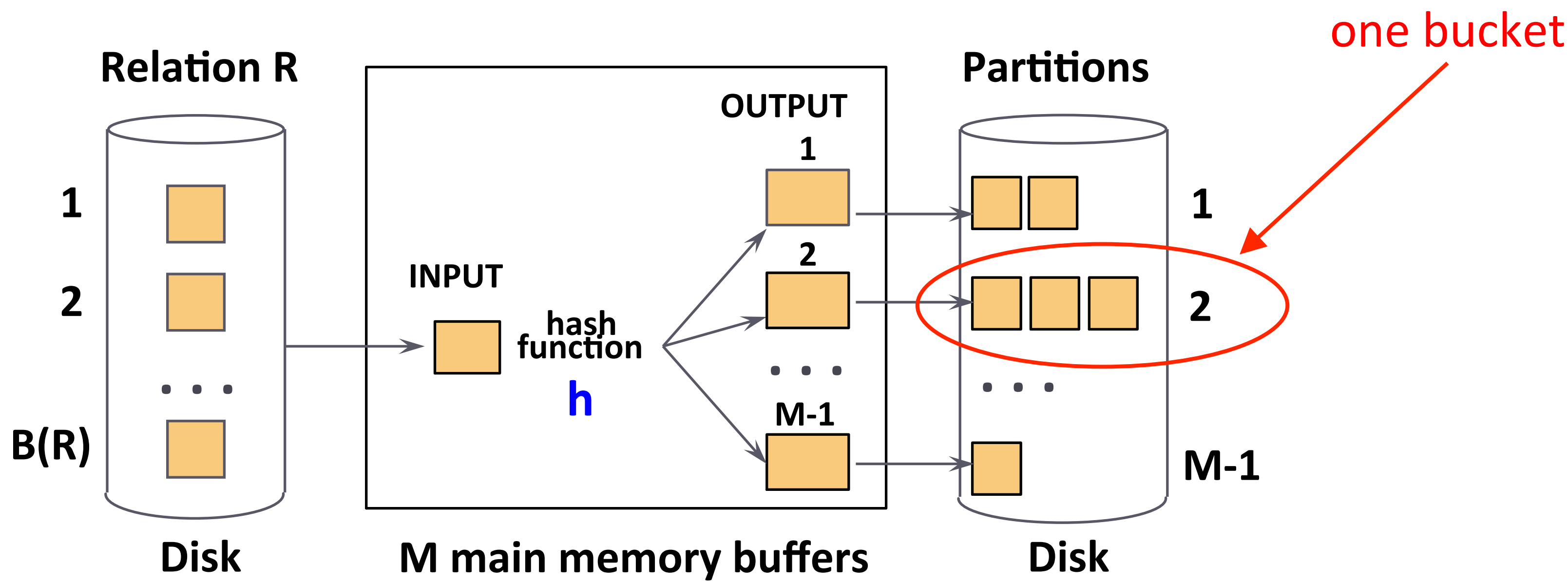
Two-Pass Algorithms

- What if data does not fit in memory?
- Need to process it in multiple passes

- Two key techniques
  - 1. Hashing
  - 2. Sorting

5. Two-Pass Join Alg. based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size ≈ B(R)/M pages



Does each bucket fit in main memory ?

Yes if $B(R)/M \leq M$, i.e. $B(R) \leq M^2$

x

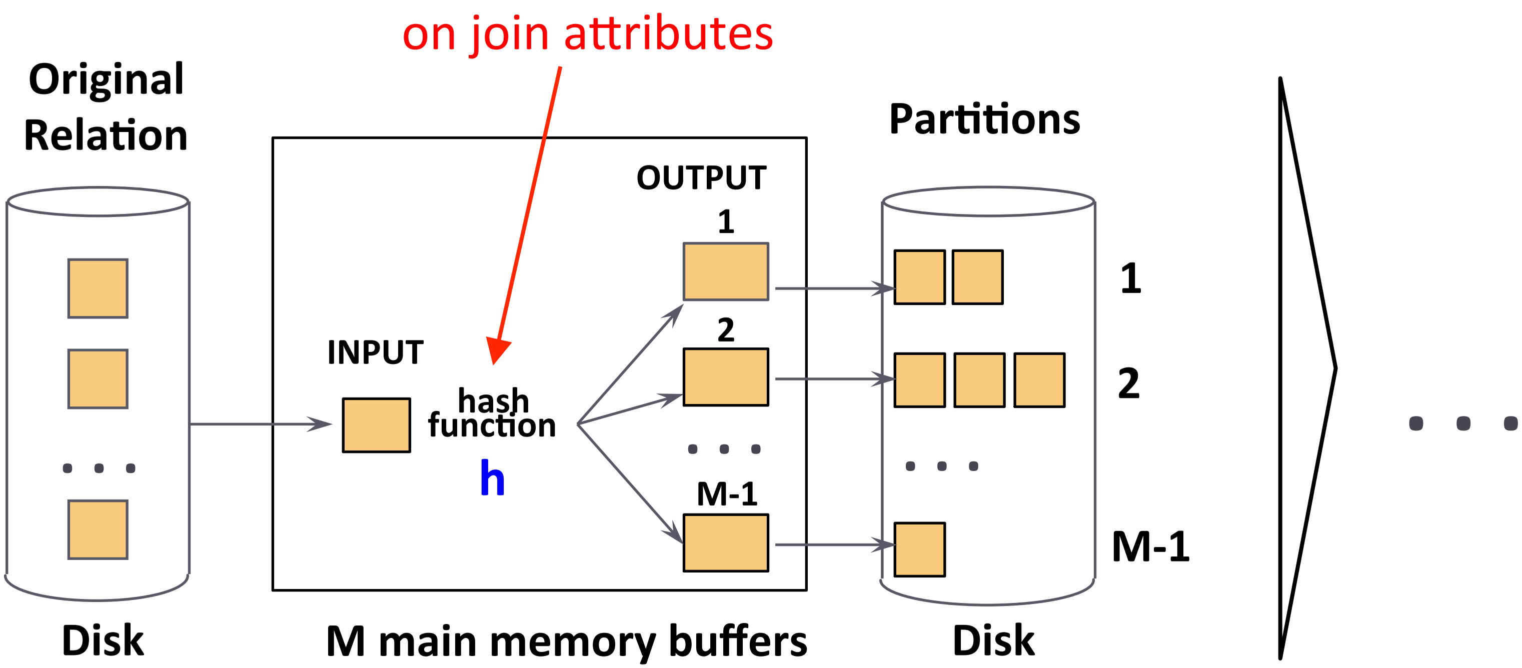
5. Partitioned (Grace) Hash Join

Hash Join $R \bowtie S$

- Step 1:
  - Hash S into M-1 buckets
  - Send all buckets to disk
- Step 2
  - Hash R into M-1 buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

5. Partitioned Hash Join

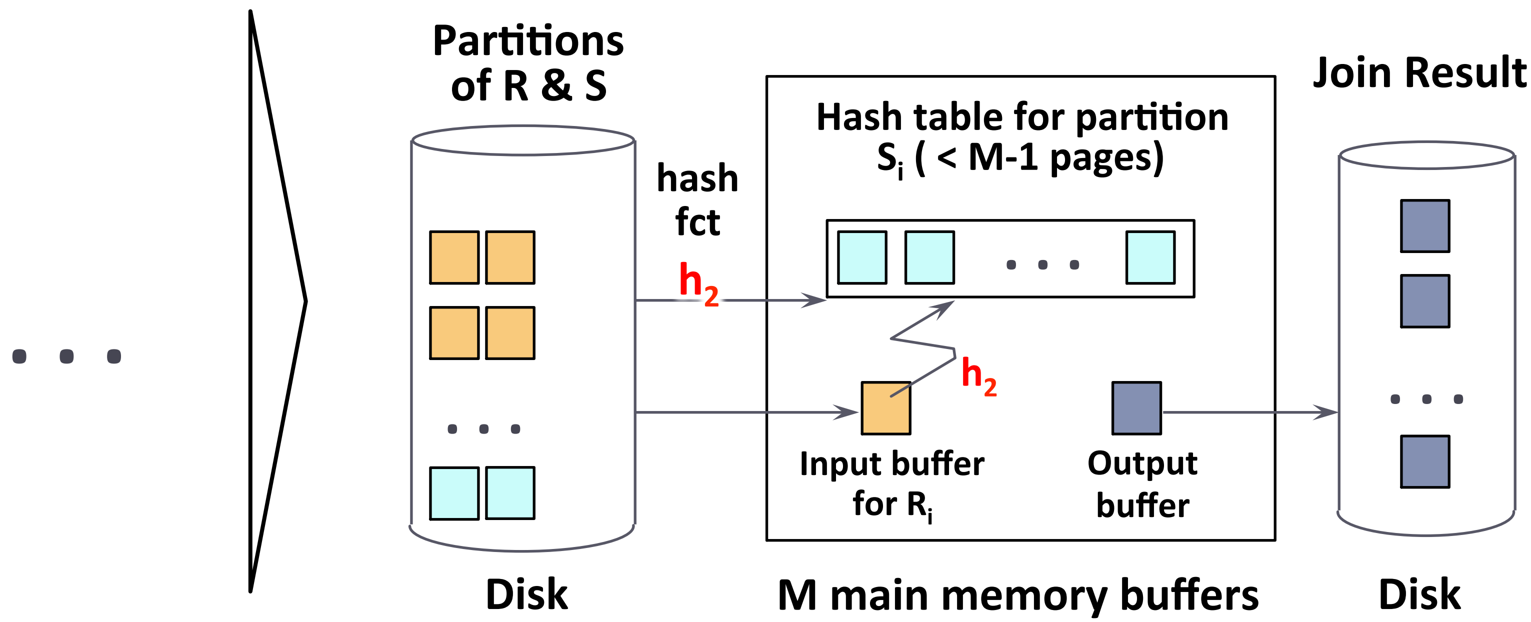
- Partition both relations using hash function **h**
- R tuples in partition *i* will only match S tuples in partition *i*.



x

5. Partitioned Hash Join

- Read in partition of R, hash it using h_2 ($\neq$ h)
  - Build phase
- Scan matching partition of S, search for matches
  - Probe phase



x

5. Partitioned Hash Join

- Cost: $3B(R) + 3B(S)$
- Assumption: $\min(B(R), B(S)) \leq M^2$

What is max. size of smaller table?
- 1 Gb main memory = 2^{30} b
- 64 Kb block size = 2^{16} b

Then M (# blocks) = $2^{14} = 16$ K

Then $B \leq M^2 =$ 2^{28}

Then total size = 2^{44} b = 16 Tb

Calculate cost with nested block join for two 16 Tb tables: Cost = $B + B^2/M$
- B(R) = 16 Tb / 64 Kb = 2^{28}
- Cost = $2^{28} + 2^{42} \approx 2^{42}$
- Cost $\approx 2^{14}$ B(R) = 16 K B(R)

  vs. 6 B(R) for Part. Hash Join

Example 15.5 in the book says 4 Tb, b/c "2^{14} = 64 K" ??? ☹

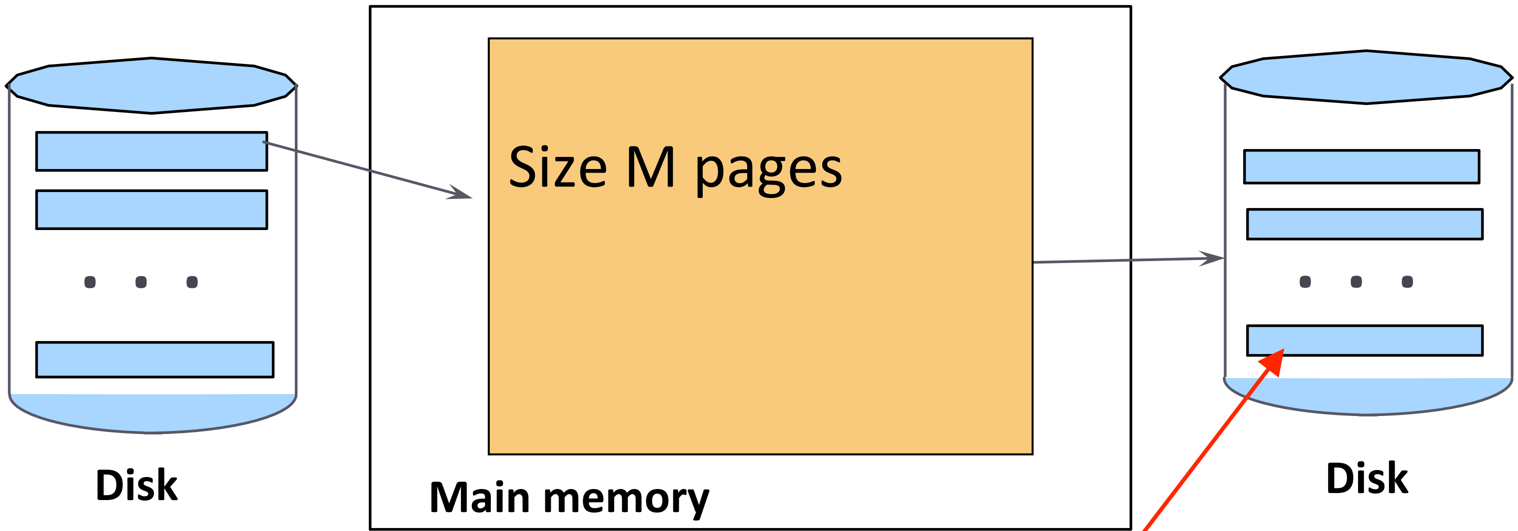
x

External Sorting

- Problem: Sort a file of size B with memory M
- Where we need this:
  - ORDER BY in SQL queries
  - Several physical operators
  - Bulk loading of B+-tree indexes.
- Sorting is two-pass when $B < M^2$

External Merge-Sort: Step 1

Step 1: Load M pages in memory, sort



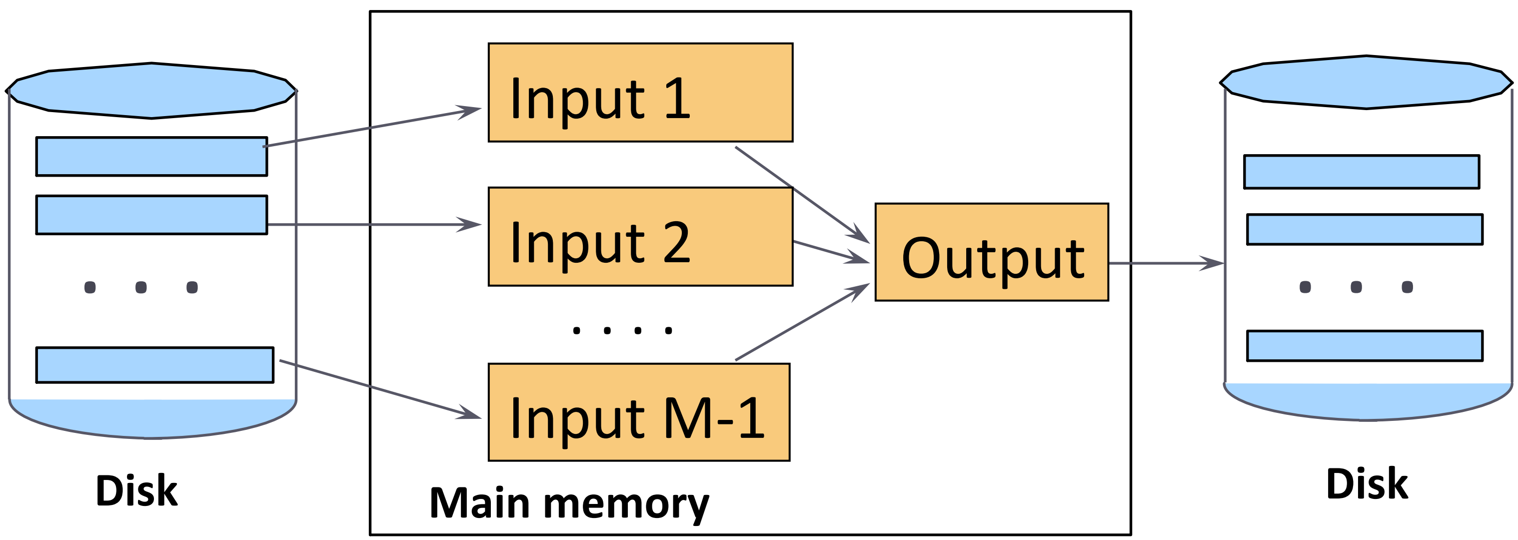
Each run is of length M pages
(maximal M-1 runs)

External Merge-Sort: Step 2

Step 1: Load M pages in memory, sort

Step 2: Merge M – 1 runs into a new run

Result: runs of length M (M – 1) ≈ M^2



Cost: read + write + read: 3B(R)

Assumption: B(R) ≤ M^2

x

6. Two-Pass Join Alg. based on Sorting

Sort-based Join $R \bowtie S$

- Step 1: sort both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Step 2: Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Total cost: 5B(R)+5B(S)
- Assumption: $B(R) \leq M^2$, $B(S) \leq M^2$

x

6. Two-Pass Join Alg. based on Sorting

Sort Merge Join $R \bowtie S$

- If $B(R) + B(S) \leq M^2$
- If the number of tuples in R matching those in S is small (or vice versa)
- We can compute the join during the merge phase

- Total cost: $3B(R)+3B(S)$

x

Summary of Join Algorithms

- Nested Loop Join: B(R) + B(R)B(S)/M
  - Assuming block-at-a-time refinement
  - With page-at a time, the formula would be: B(R) + B(R)B(S)
- Hash Join: 3B(R) + 3B(S)
  - Assuming: min(B(R), B(S)) ≤ M^2
- Sort-Merge Join: 3B(R) + 3B(S)
  - Assuming B(R)+B(S) ≤ M^2
- Index Nested Loop Join: B(R) + T(R)B(S)/V(S,a)
  - Assuming S has clustered index on attribute attribute a

x

Summary of Query Execution

- For each logical query plan
  - There exist many physical query plans
  - Each plan has a different cost
  - Cost depends on the data
- Additionally, for each query
  - There exist several logical plans
- Next 3 lectures: query optimization
  - How to compute the cost of a complete plan?
  - How to pick a good query plan for a query?