Introduction to Database Systems CSE 444

Lecture 19: Operator Algorithms

Why Learn About Op Algos?

- 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)
- Main constraint: **M = # of memory (buffer) pages**

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: 3B(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 algorithms for join operator (because other operators are easier)
    - Read the book to get more details about these algos
    - Read the book to learn about algos for other operators

Basic Join Algorithms

- Logical operator:
  – Product(pname, cname) ⋈ 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**

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 $B(R) <= M$
  – By “one pass”, we mean that the operator reads its operands only once. It does not write intermediate results back to disk.

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’

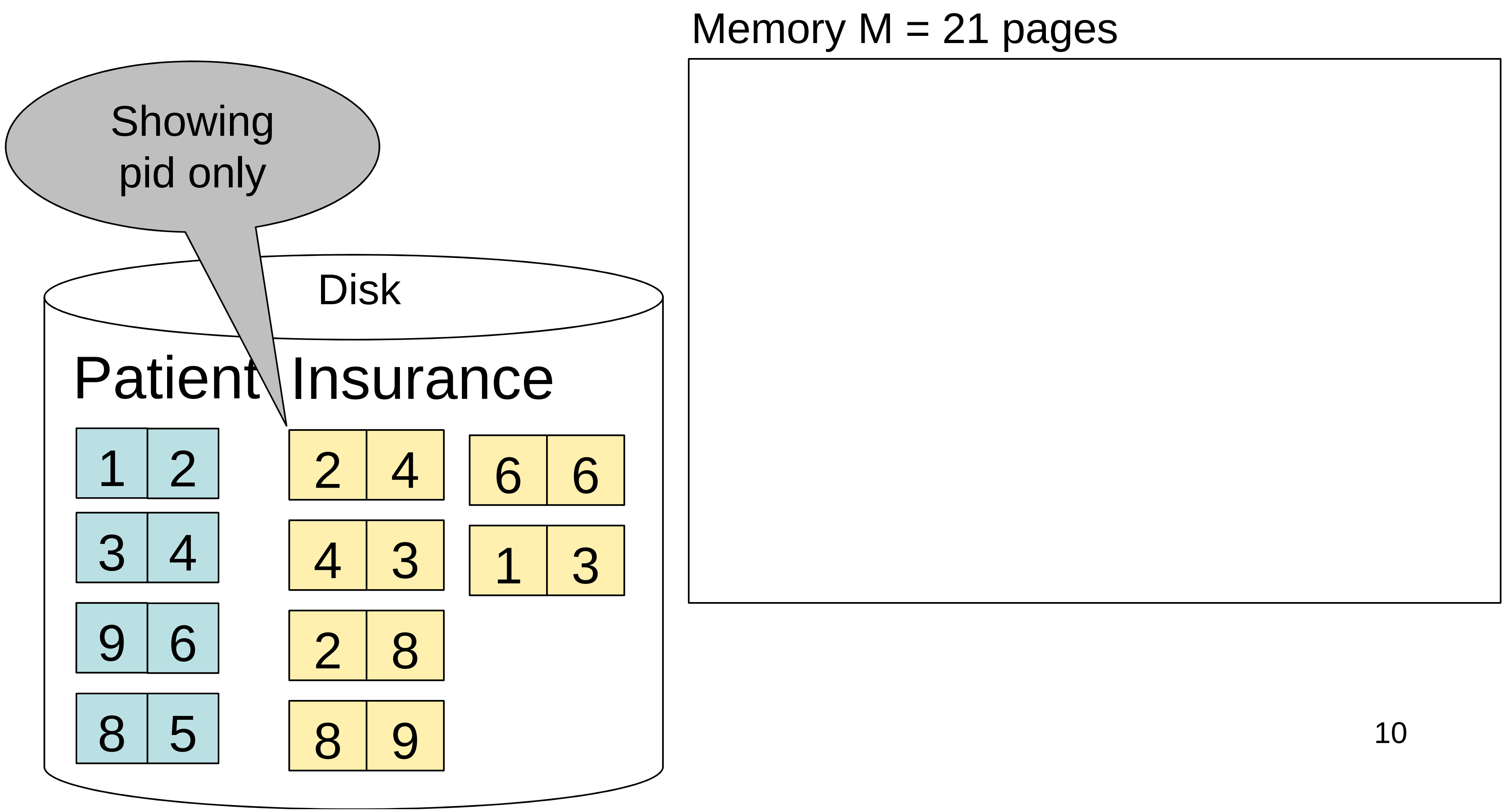
Insurance

2	‘Blue’	123
4	‘Prem’	432

4	‘Prem’	343
3	‘GrpH’	554

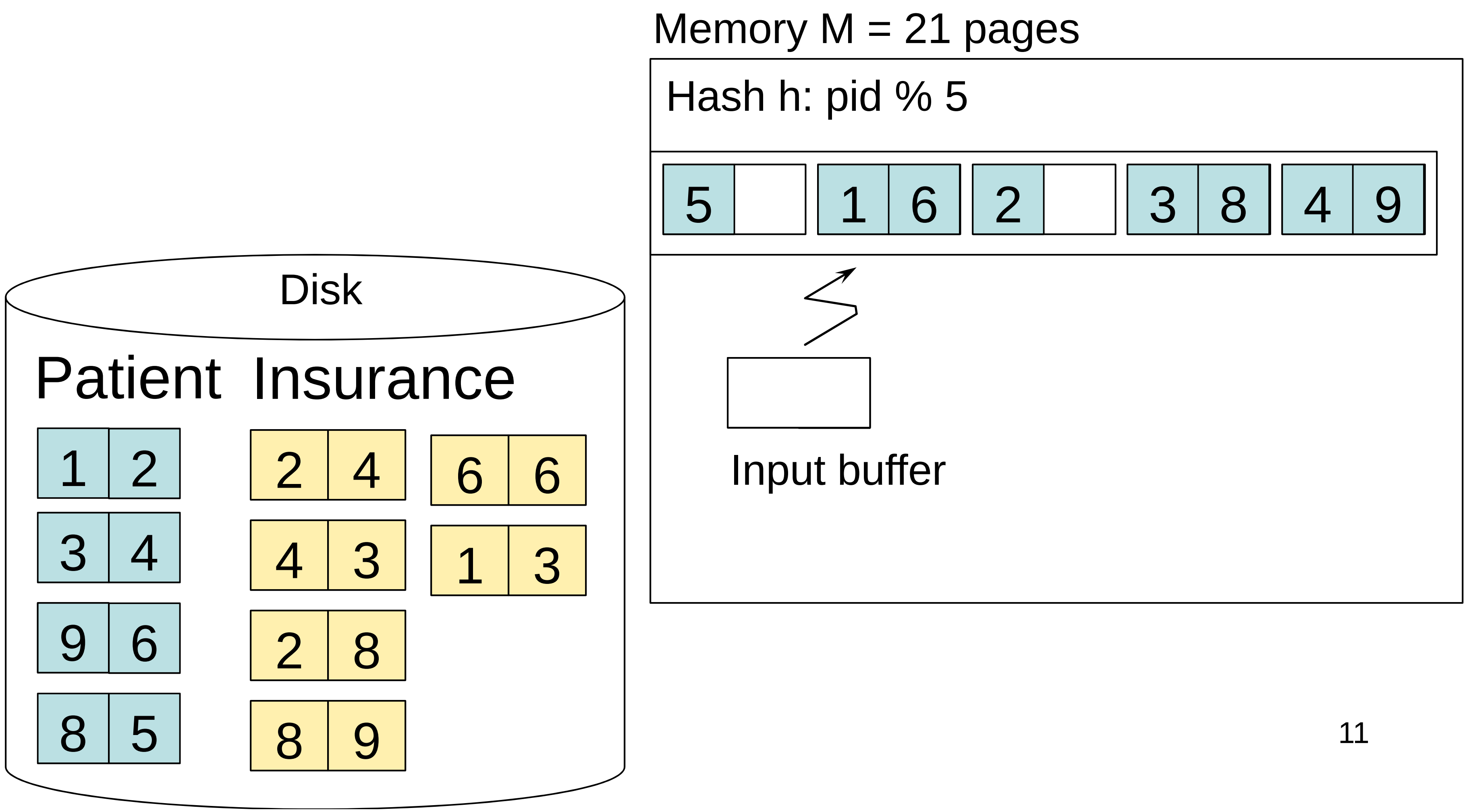
Hash Join Example

Patient $\bowtie$ Insurance



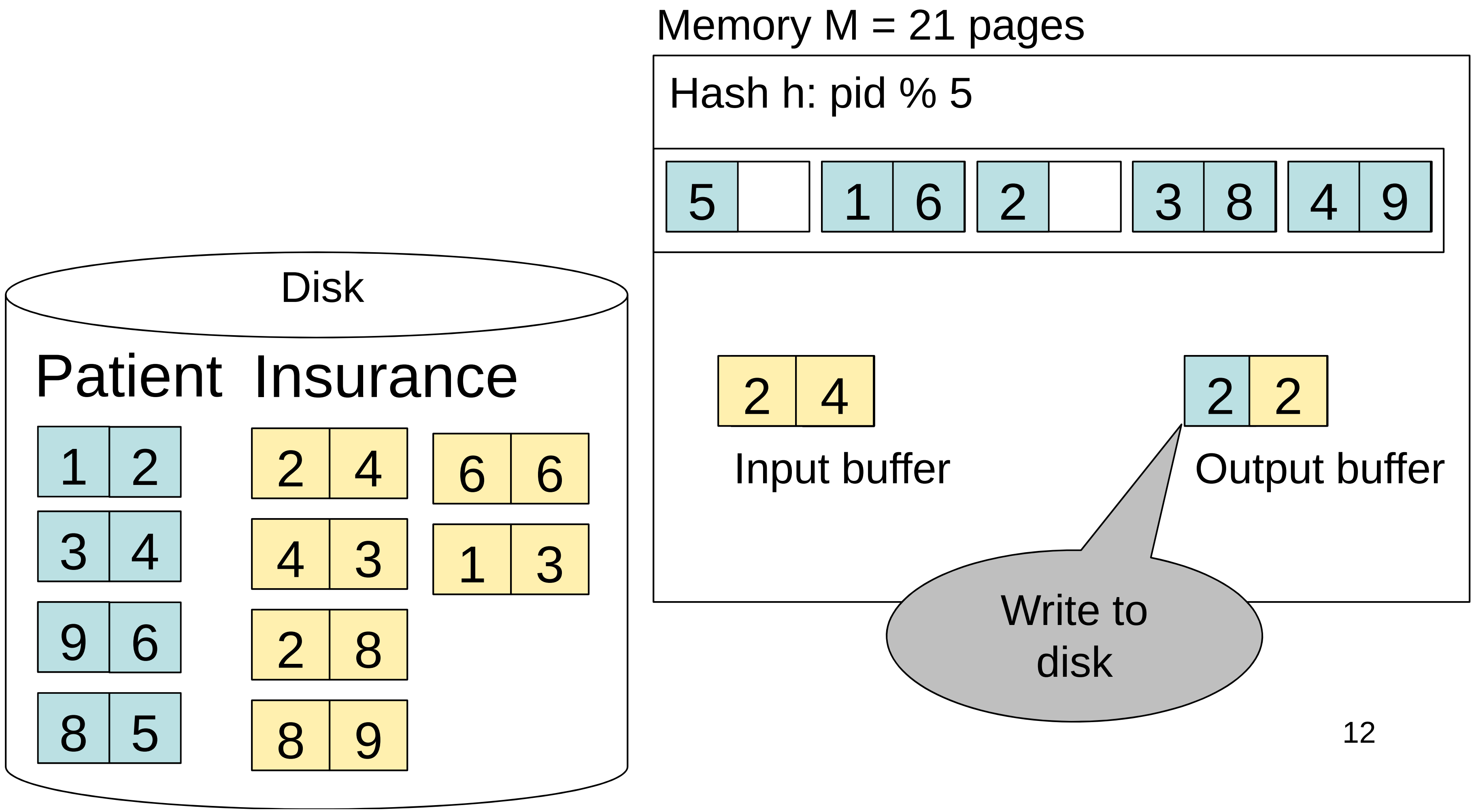
Hash Join Example

Step 1: Scan Patient and create hash table in memory



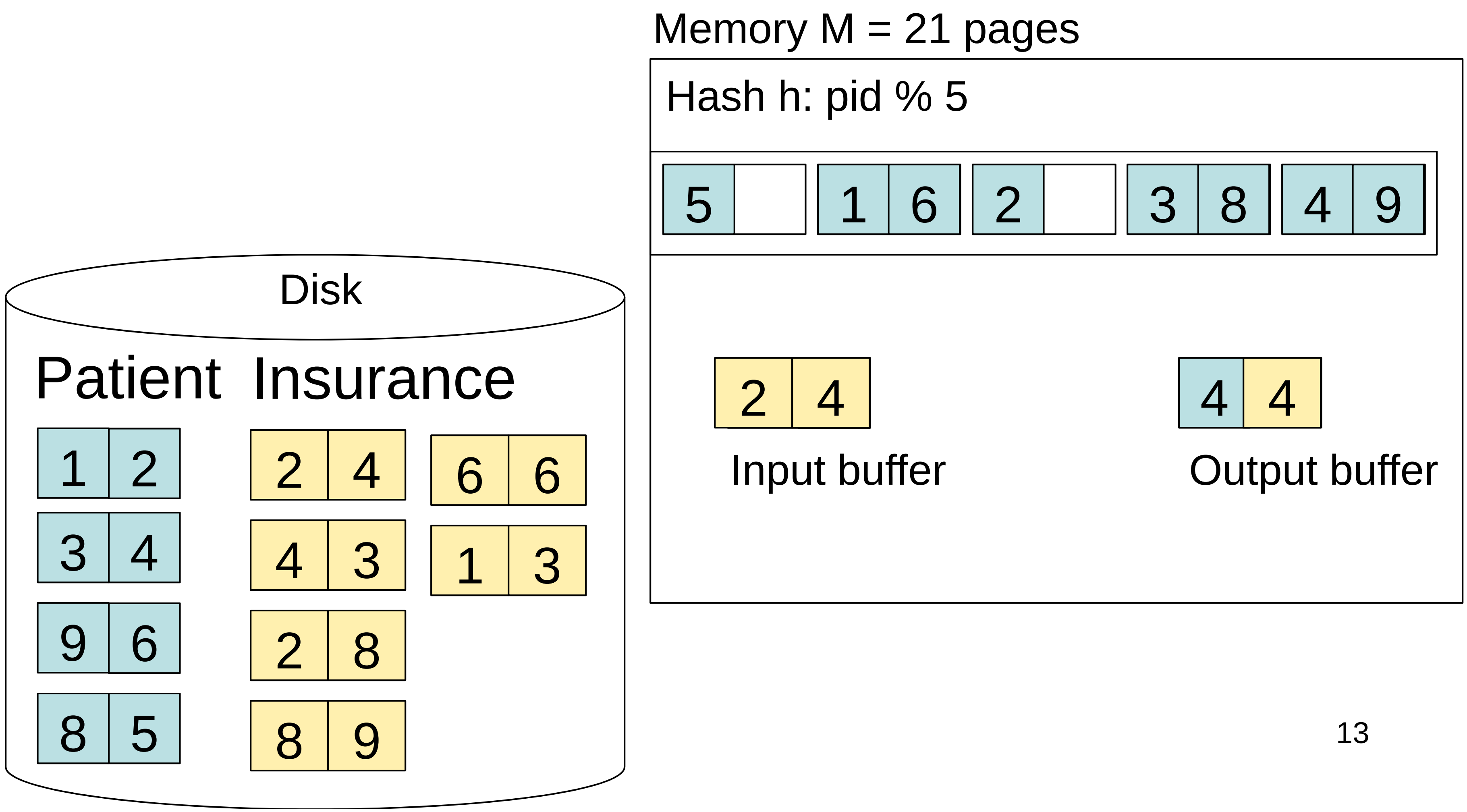
Hash Join Example

Step 2: Scan Insurance and probe into hash table



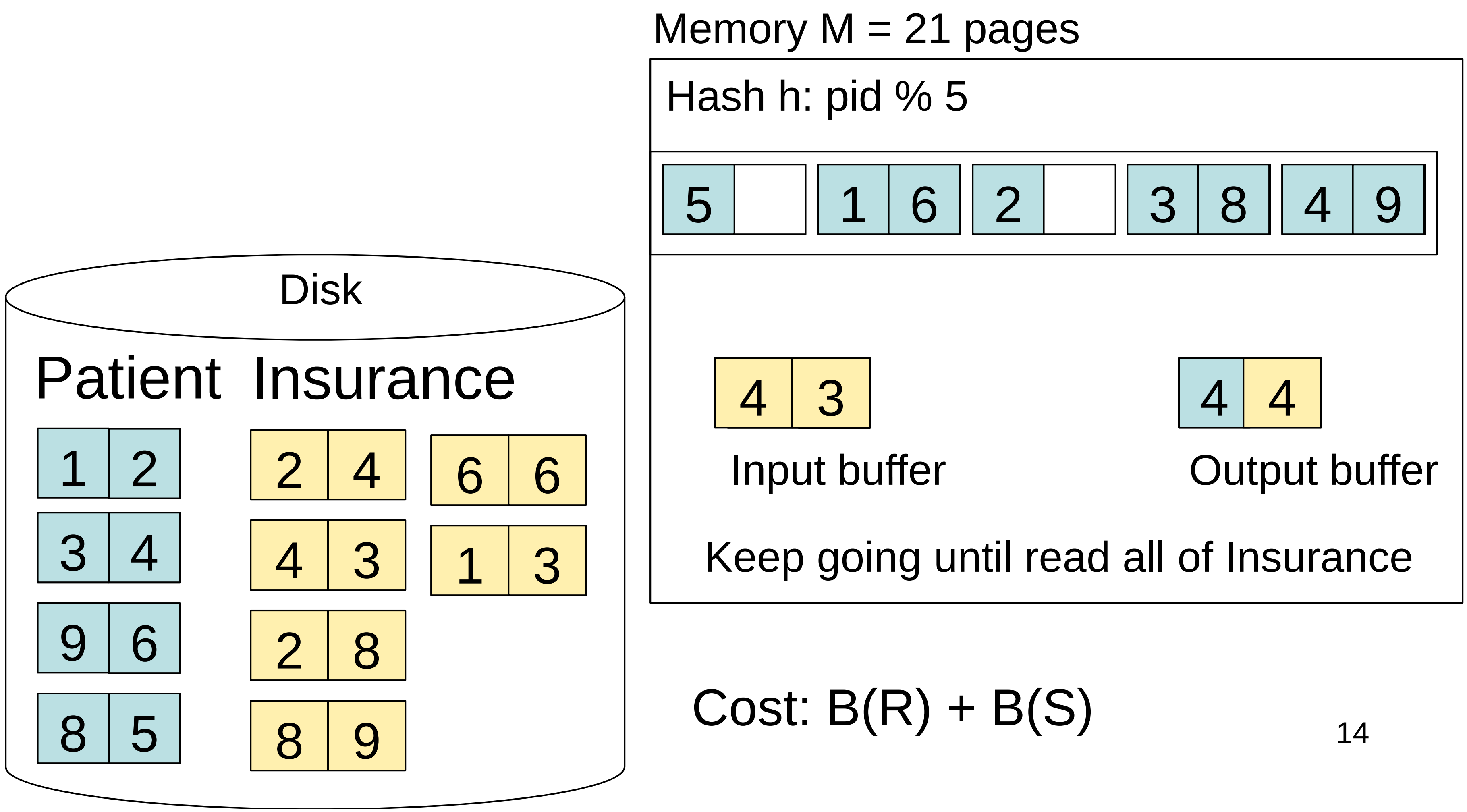
Hash Join Example

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)

Hash Join Details

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

Hash Join Details

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

Hash Join Details

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

Nested Loop Joins

- Tuple-based nested loop R ⋈ 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)
```

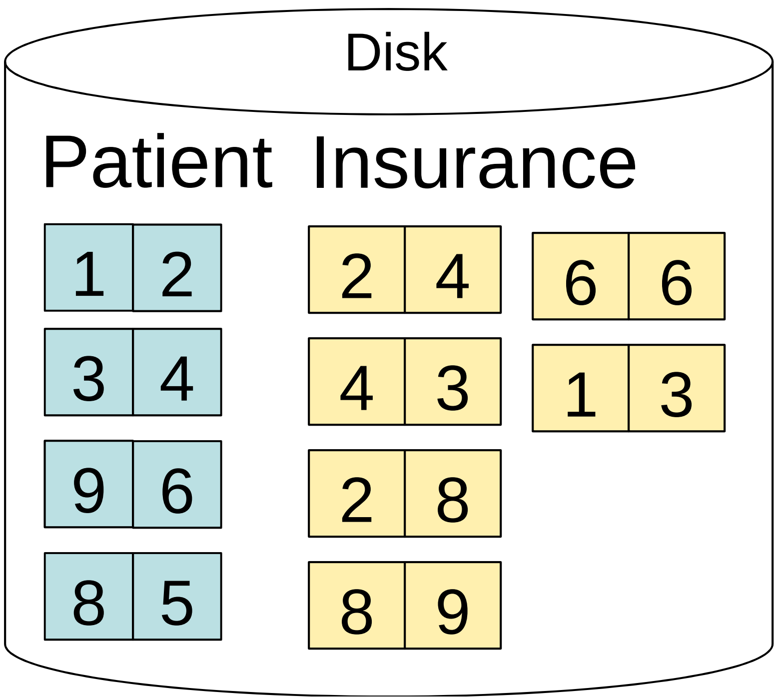
- Cost: B(R) + T(R) B(S)
- Not quite one-pass since S is read many times

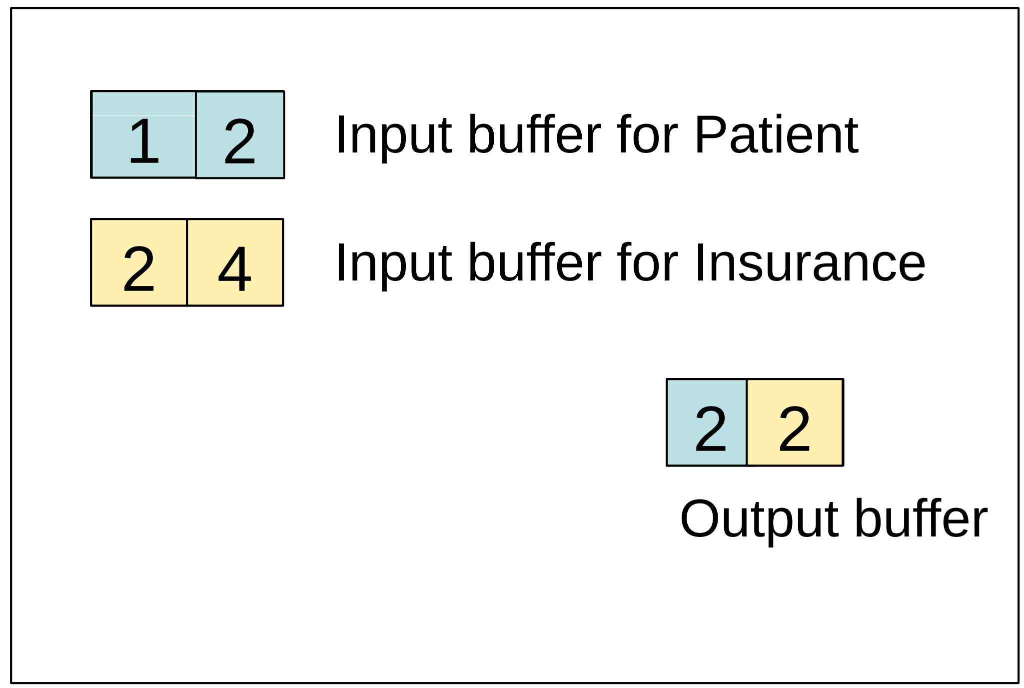
Page-at-a-time Refinement

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

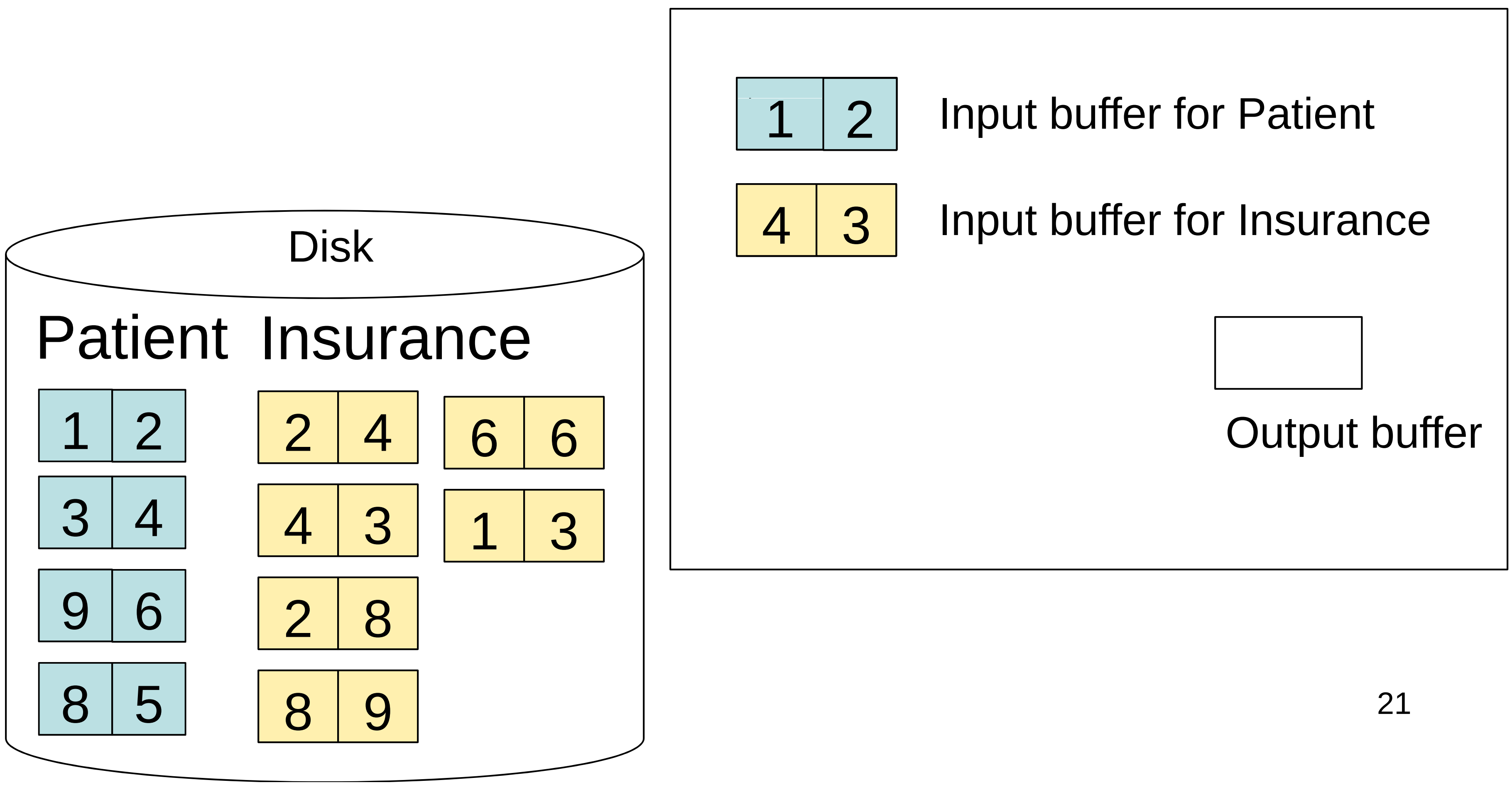
- Cost: B(R) + B(R)B(S)

Nested Loop Example

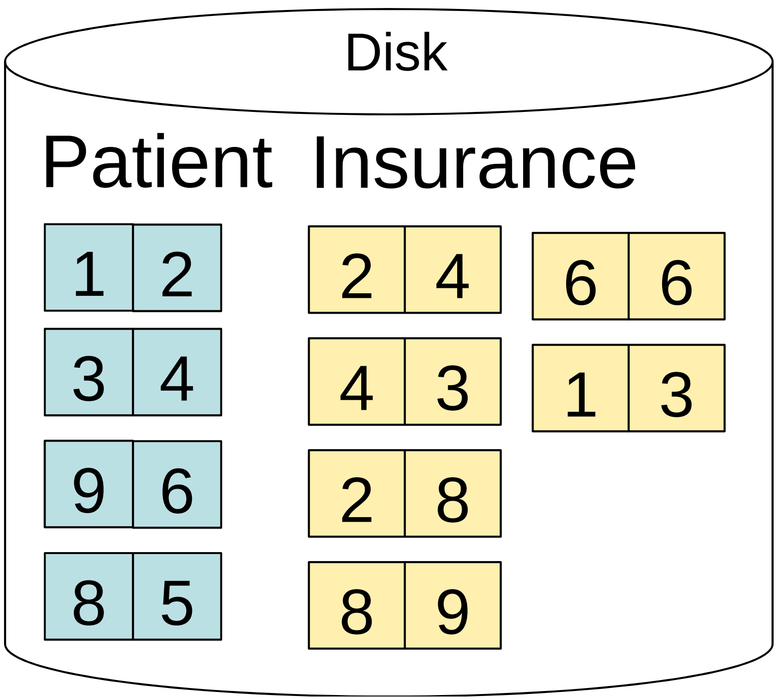


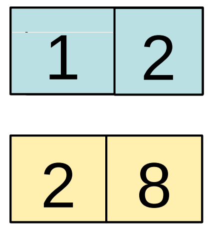


Nested Loop Example



Nested Loop Example

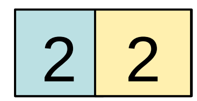




Input buffer for Patient

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

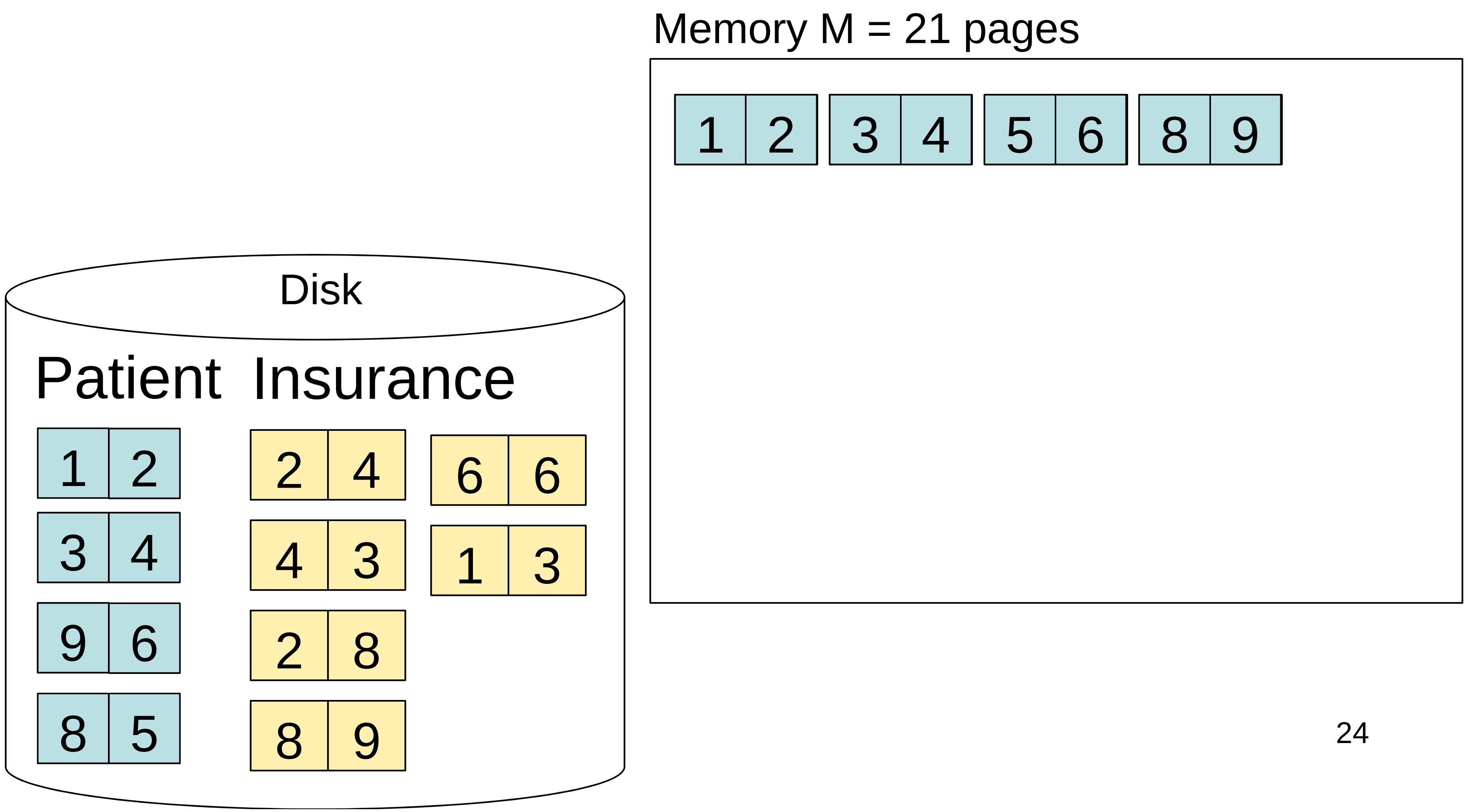
Sort-merge join: R $\bowtie$ S

- 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

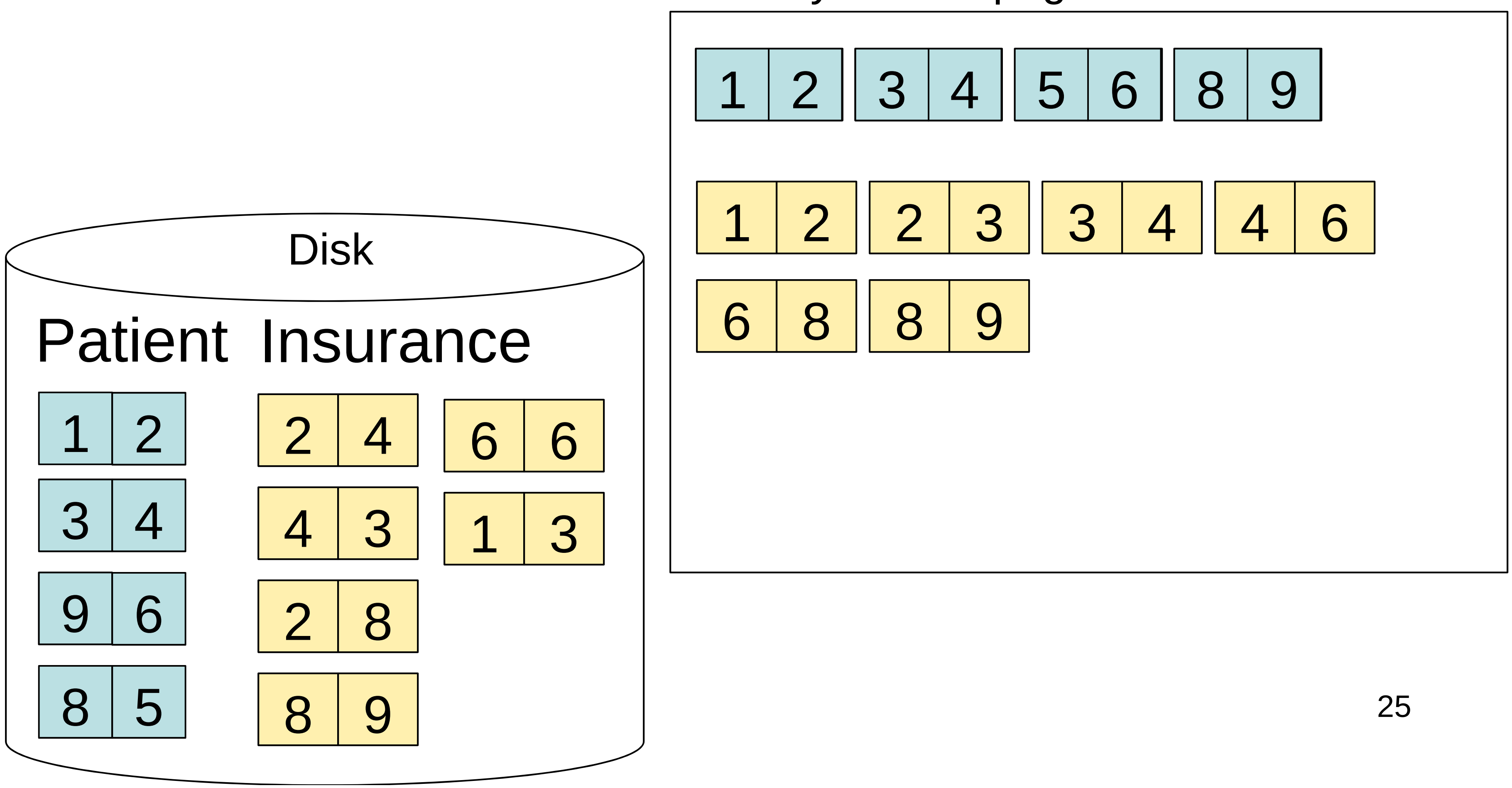
Step 1: Scan Patient and sort in memory



Sort-Merge Join Example

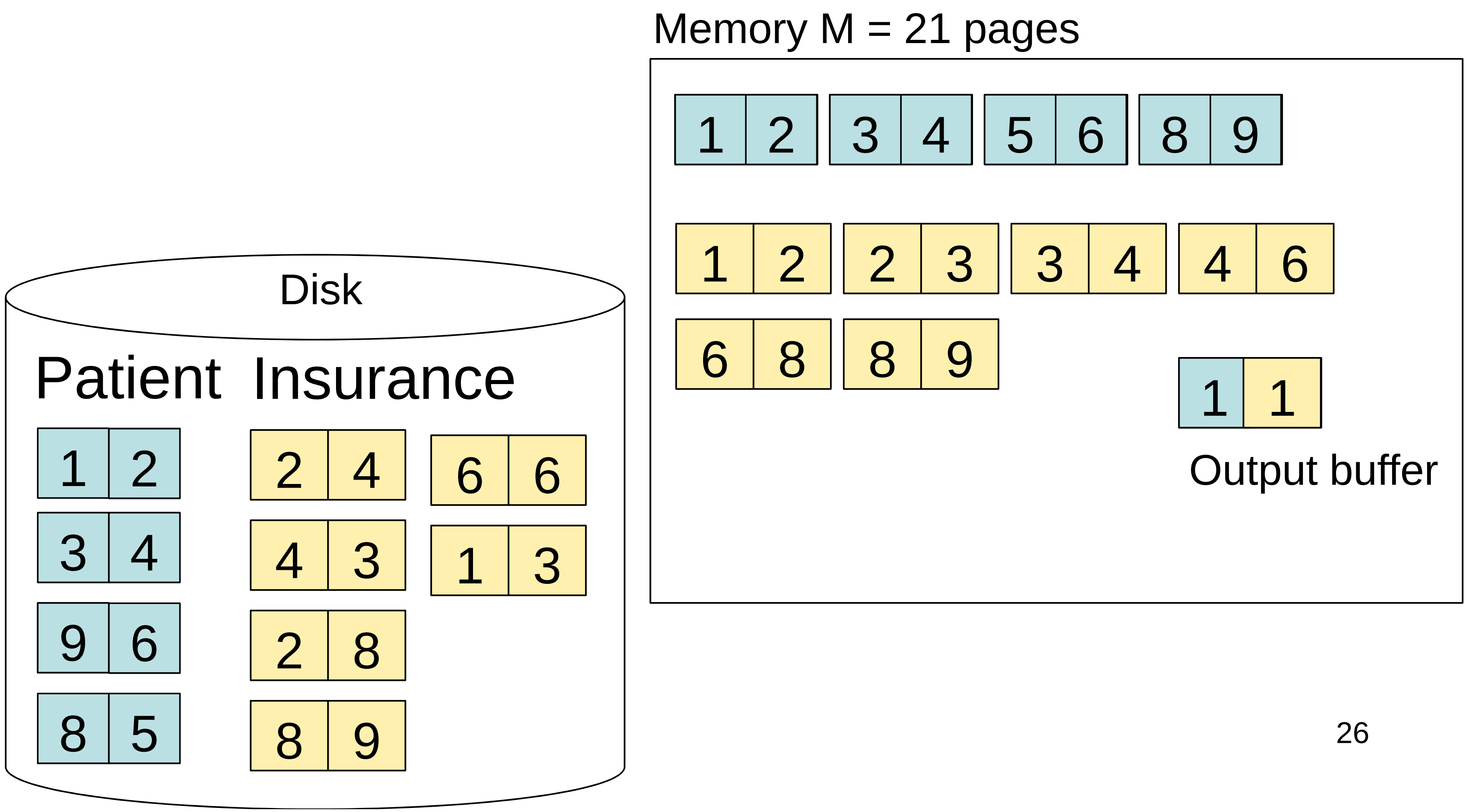
Step 2: Scan Insurance and sort in memory

Memory M = 21 pages



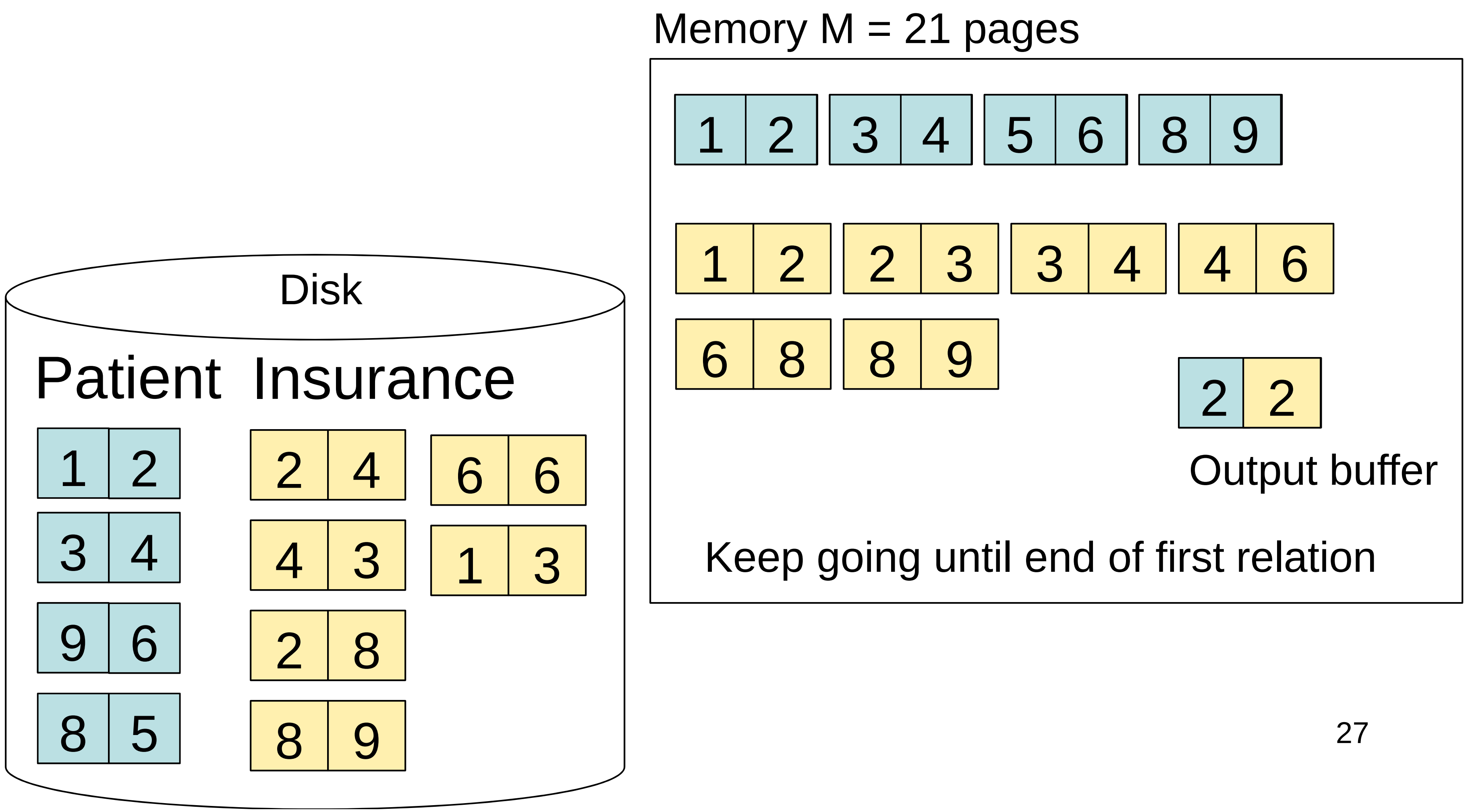
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Sort-Merge Join Example

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
- Clustered index on a: cost B(R)/V(R,a)
- Unclustered index on a: cost T(R)/V(R,a)
- Note: we ignored 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: 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: $B(R) + T(R)T(S) / V(S,a)$

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)

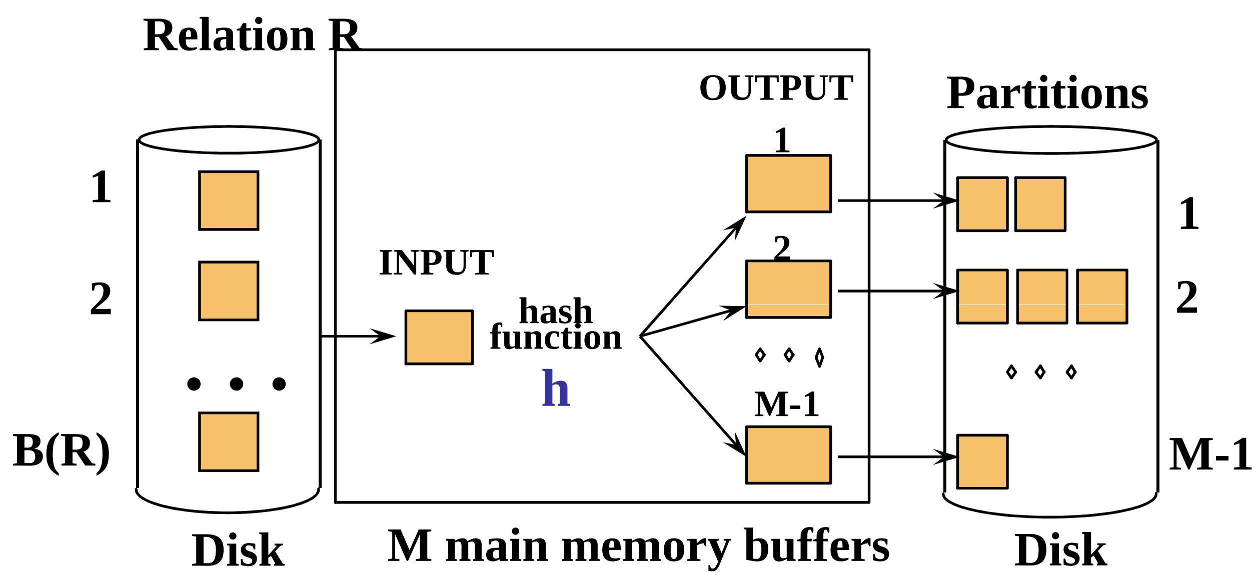
Two-Pass Algorithms

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

- Two key techniques
  - Hashing
  - Sorting

Two Pass Algorithms Based on Hashing

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



- Does each bucket fit in main memory ?
  - Yes if B(R)/M <= M, i.e. B(R) <= M^2

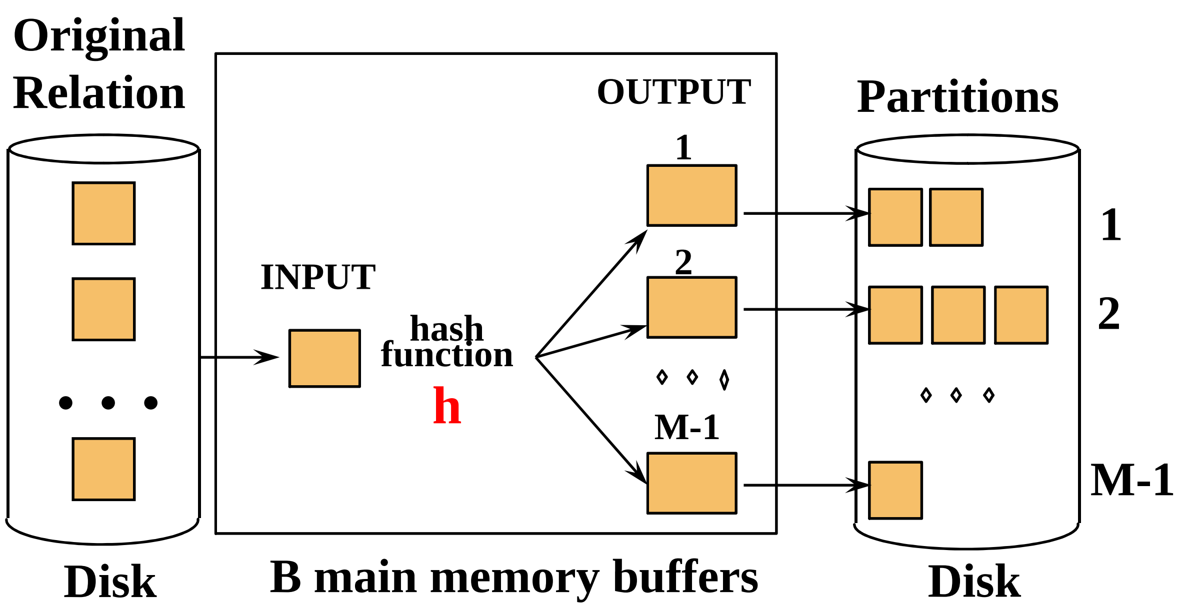
Partitioned (Grace) Hash Join

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

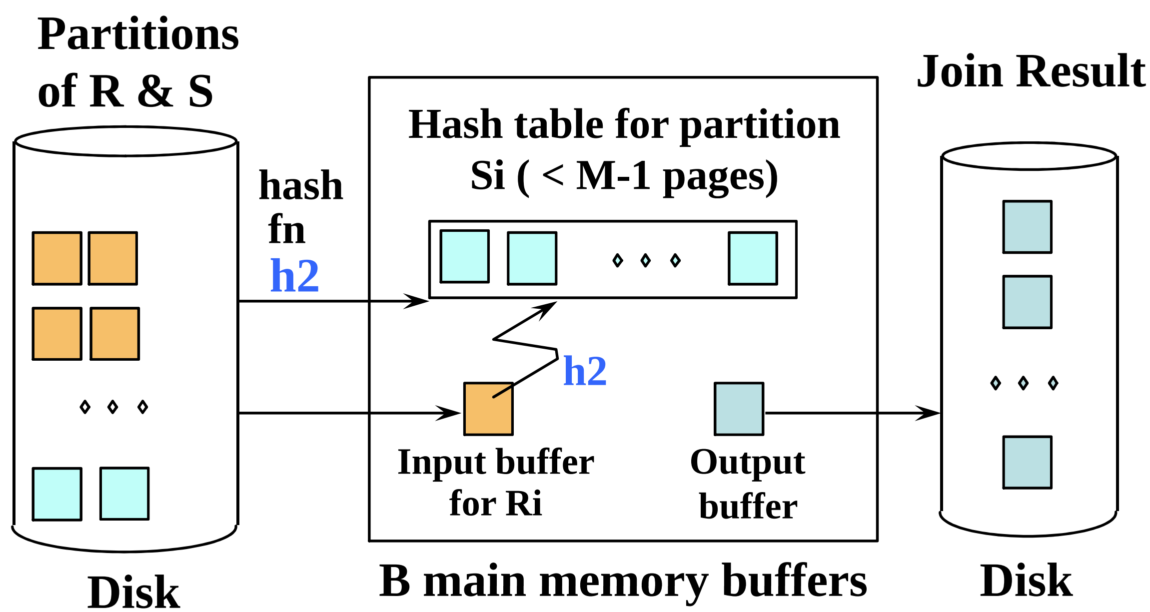
Partitioned Hash Join

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



Partitioned Hash Join

- Read in partition of R, hash it using h2 ($\neq$ h)
  - Build phase
- Scan matching partition of S, search for matches
  - Probe phase



Partitioned Hash Join

- Cost: 3B(R) + 3B(S)
- Assumption: min(B(R), B(S)) <= M^2

External Sorting

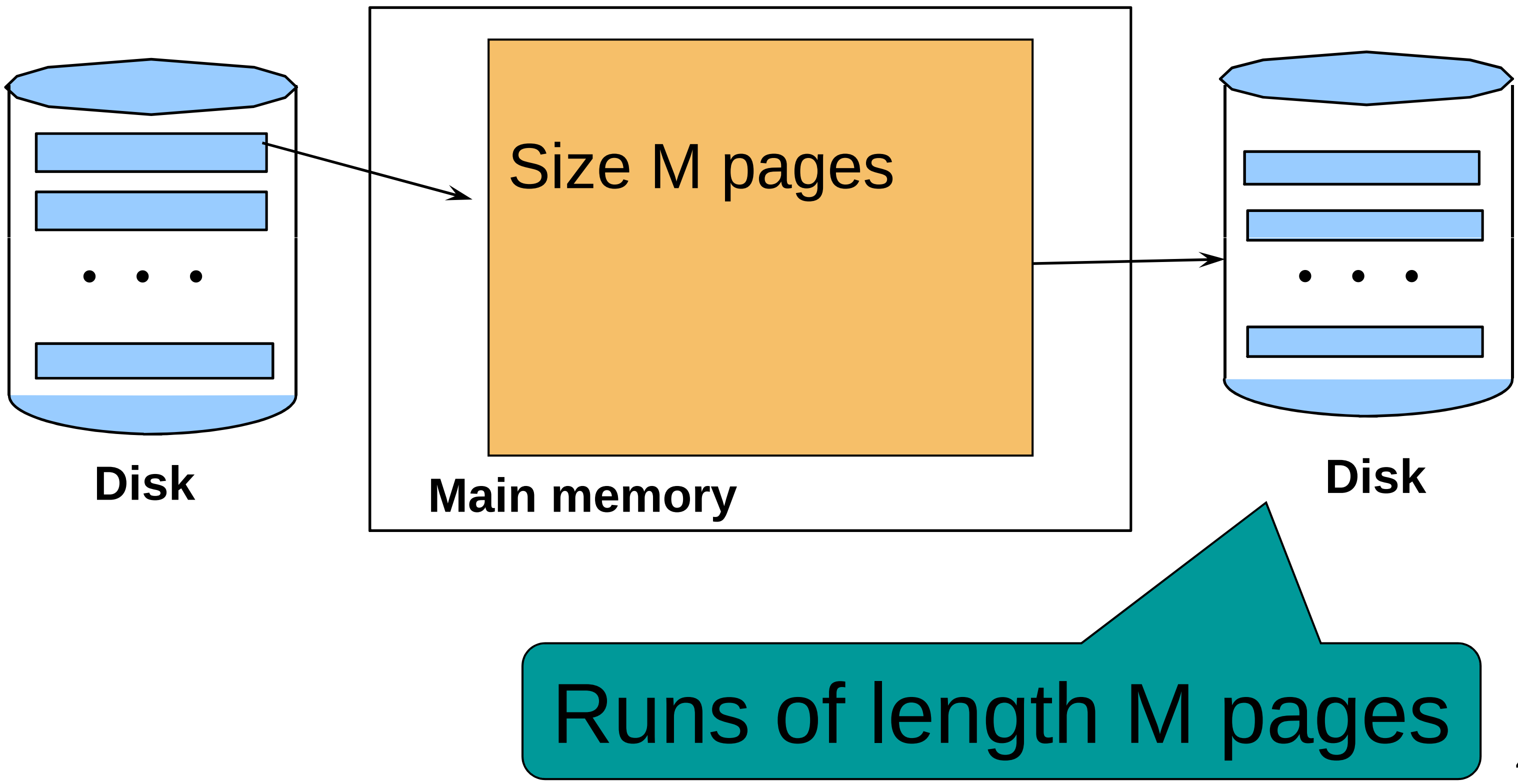
- Problem: Sort a file of size B with memory M

- Where we need this:
  - ORDER BY in SQL queries
  - Several physical operators
  - Bulk loading of B+-tree indexes.

- Sorting is two-pass when $B < M^2$

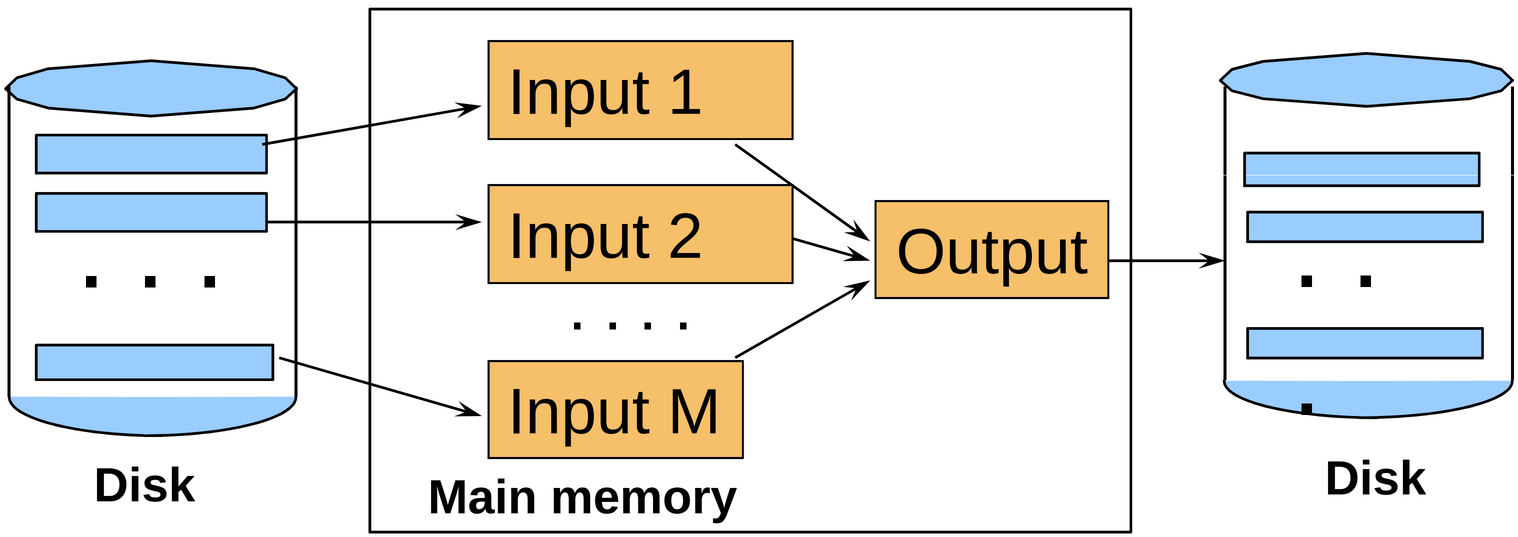
External Merge-Sort: Step 1

- Phase one: load M pages in memory, sort



External Merge-Sort: Step 2

- Merge M – 1 runs into a new run
- Result: runs of length M (M – 1)≈ M^2



If B <= M^2 then we are done

External Merge-Sort

- Cost:
  – Read+write+read = 3B(R)
  – Assumption: $B(R) <= M^2$
- Other considerations
  – In general, a lot of optimizations are possible

Two-Pass Join Algorithm Based on Sorting

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: $B(R) <= M^2$, $B(S) <= M^2$

Two-Pass Join Algorithm Based on Sorting

Join R ⋈ S

- If B(R) + B(S) <= M^2
  - Or if use a priority queue to create runs of length 2|M|
- 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)

Summary of Join Algorithms

- Nested Loop Join: B(R) + B(R)B(S)
  - Assuming page-at-a-time refinement
- 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 a

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 lecture: query optimization
  - How to compute the cost of a complete plan?
  - How to pick a good query plan for a query?