

Database System Internals

Operator Algorithms (part 2)

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

Today's Outline

Query Execution Algorithms:

- Catch-up from last lecture
- Finish operator implementation

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, plus cost to **read/write** intermediate results
- 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 $B(R) \leq M$

Note: the inner relation is the relation on which we build the hash table

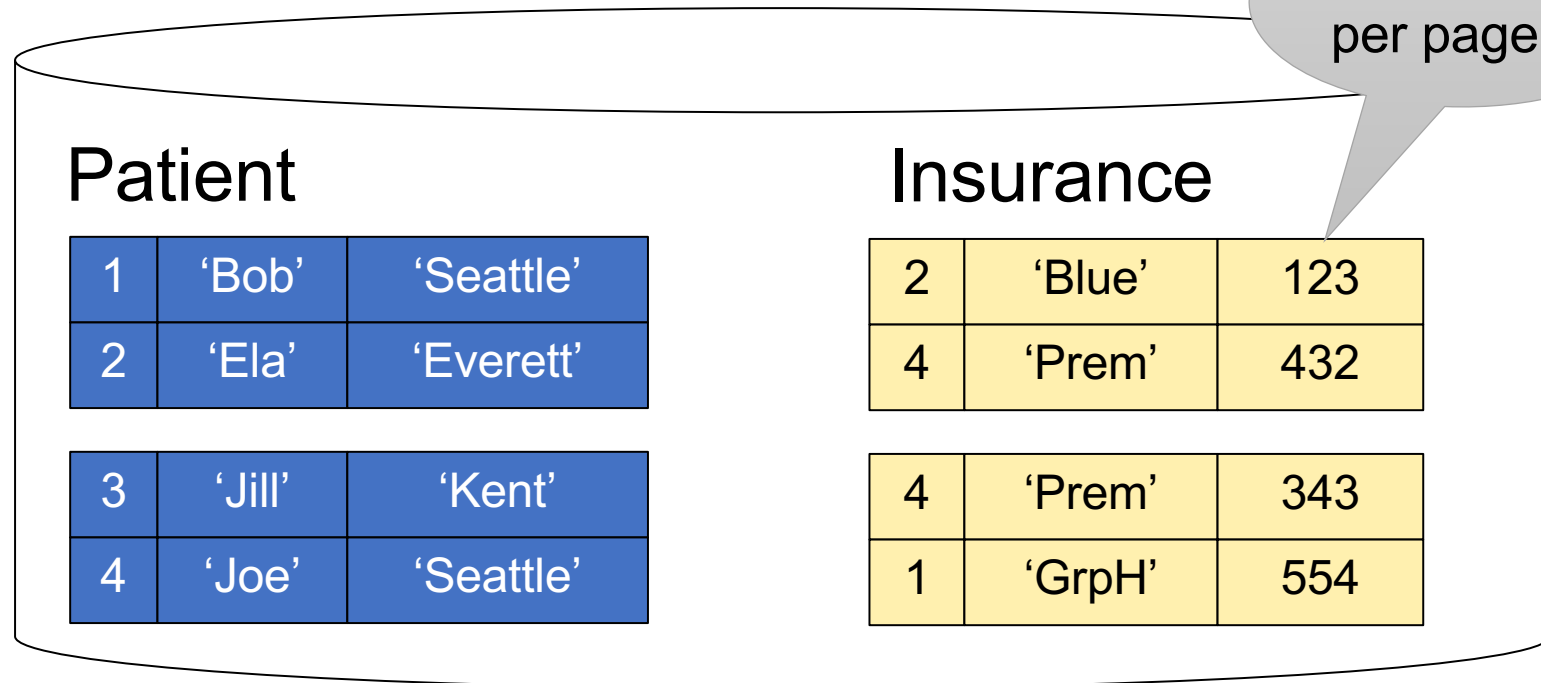
- Usually this is the right relation of $R \bowtie S$, i.e. S .
- But the following slides choose the left relation, i.e. R

Hash Join Example

Patient(pid, name, address)

Insurance(pid, provider, policy_nb)

Patient ⋈ Insurance



Hash Join Example

Patient \bowtie Insurance

Some large-enough nb

Memory M = 21 pages

Showing pid only

Disk

Patient		Insurance			
1	2	2	4	6	6
3	4	4	3	1	3
9	6	2	8		
8	5	8	9		

This is one page with two tuples

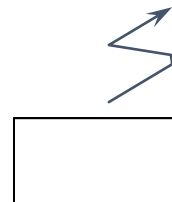
Hash Join Example

Step 1: Scan Patient and **build** hash table in memory
Can be done in method open()

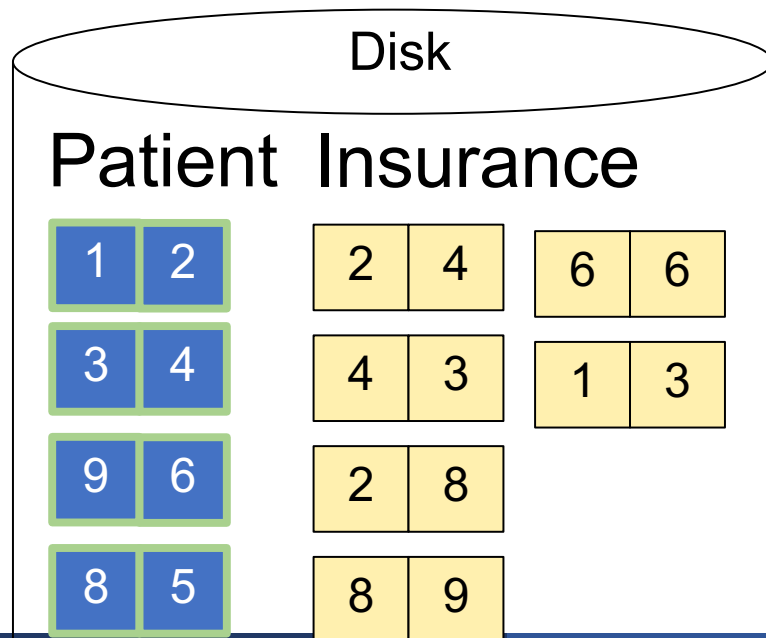
Memory M = 21 pages

Hash h: pid % 5

5		1	6	2		3	8	4	9
---	--	---	---	---	--	---	---	---	---



Input buffer



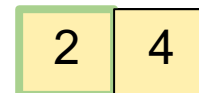
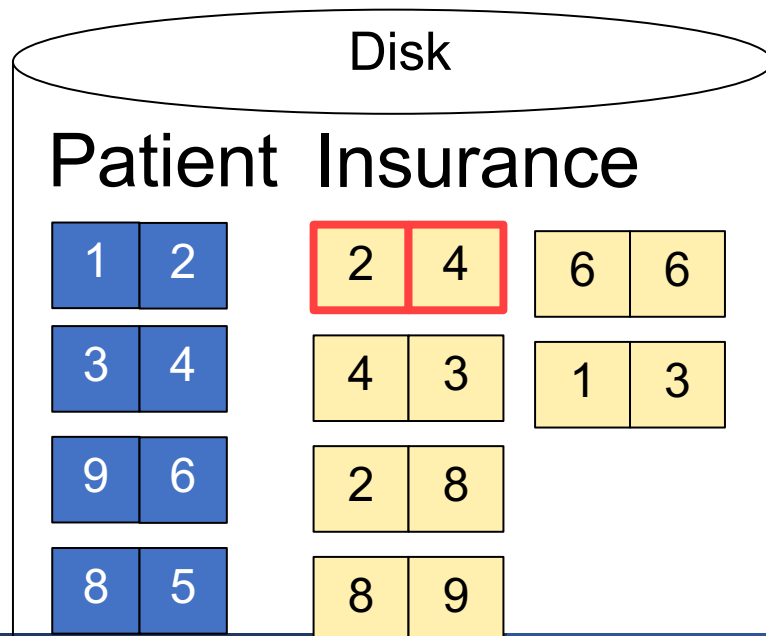
Hash Join Example

Step 2: Scan Insurance and **probe** into hash table
Done during
calls to next()

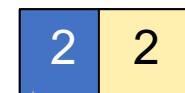
Memory M = 21 pages

Hash h: pid % 5

5		1	6	2		3	8	4	9
---	--	---	---	---	--	---	---	---	---



Input buffer



Output buffer

Write to disk or
pass to next
operator

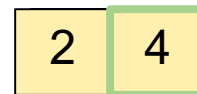
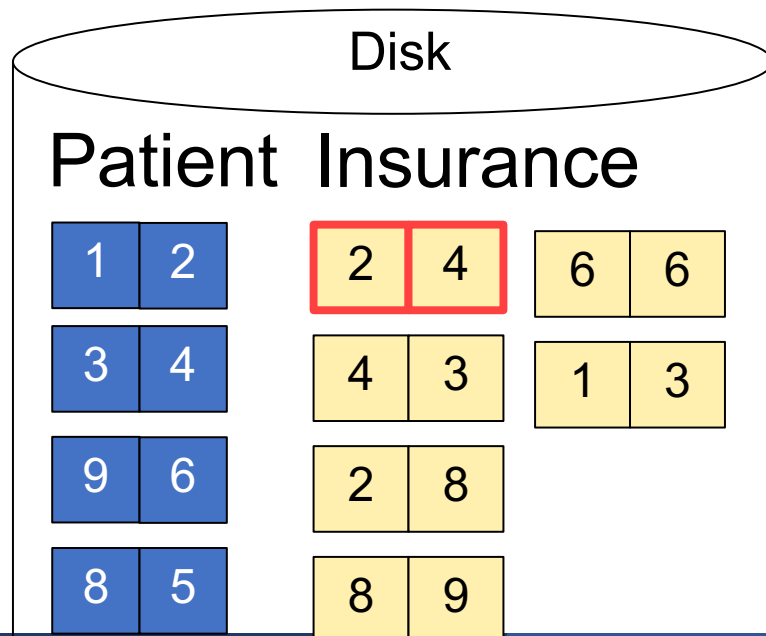
Hash Join Example

Step 2: Scan Insurance and **probe** into hash table
Done during
calls to next()

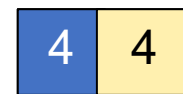
Memory M = 21 pages

Hash h: pid % 5

5		1	6	2		3	8	4	9
---	--	---	---	---	--	---	---	---	---



Input buffer



Output buffer

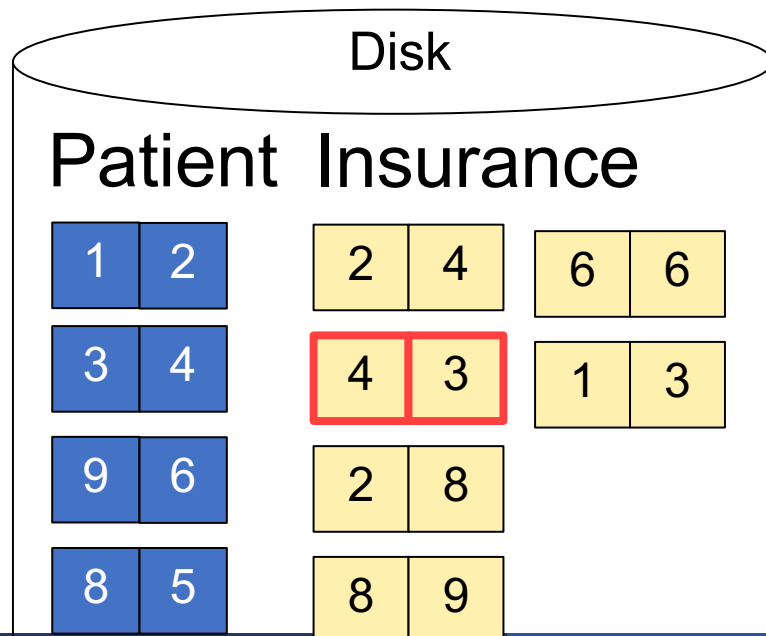
Hash Join Example

Step 2: Scan Insurance and **probe** into hash table
Done during calls to next()

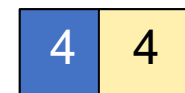
Memory M = 21 pages

Hash h: pid % 5

5		1	6	2		3	8	4	9
---	--	---	---	---	--	---	---	---	---



Input buffer



Output buffer

Keep going until read all of Insurance

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 measure its size
- 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 $R \bowtie S$
- R is the outer relation, S is the inner relation

```
for each tuple  $t_1$  in R do  
  for each tuple  $t_2$  in S do  
    if  $t_1$  and  $t_2$  join then output  $(t_1, t_2)$ 
```

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  $t_1$  in  $R$  do  
  for each tuple  $t_2$  in  $S$  do  
    if  $t_1$  and  $t_2$  join then output  $(t_1, t_2)$ 
```

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

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  $t_1$  in r,  $t_2$  in s  
      if  $t_1$  and  $t_2$  join then output  $(t_1, t_2)$ 
```

What is the **Cost**?

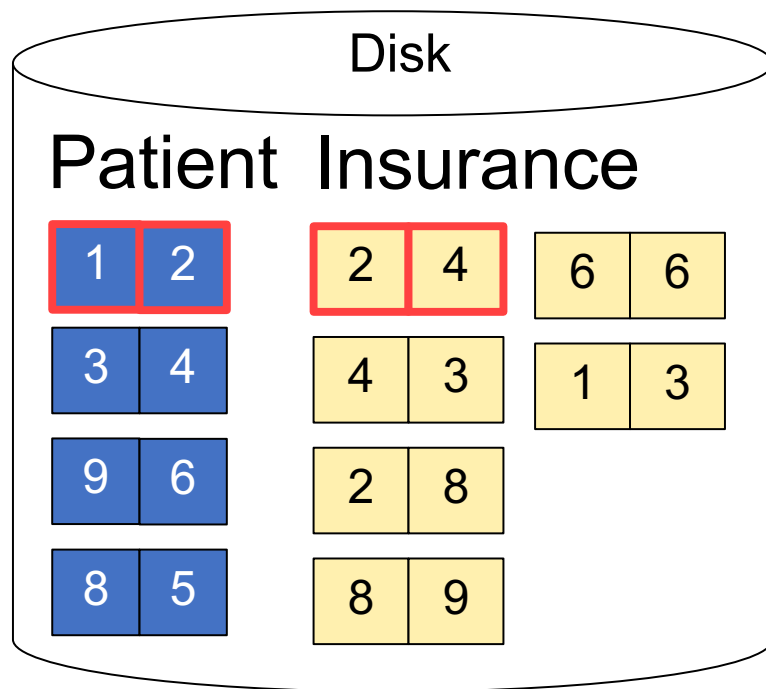
Page-at-a-time Refinement

```
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)
```

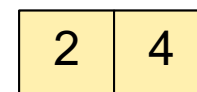
- Cost: $B(R) + B(R)B(S)$

What is the Cost?

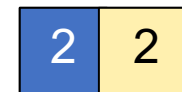
Page-at-a-time Refinement



Input buffer for Patient

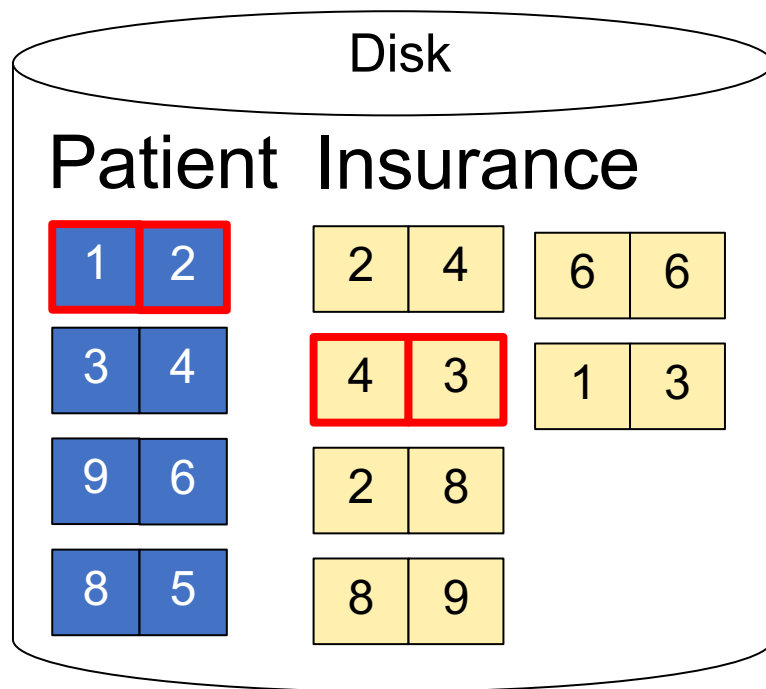


Input buffer for Insurance

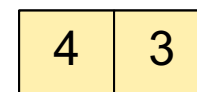


Output buffer

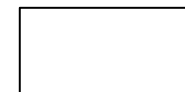
Page-at-a-time Refinement



Input buffer for Patient

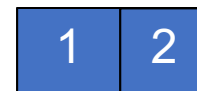
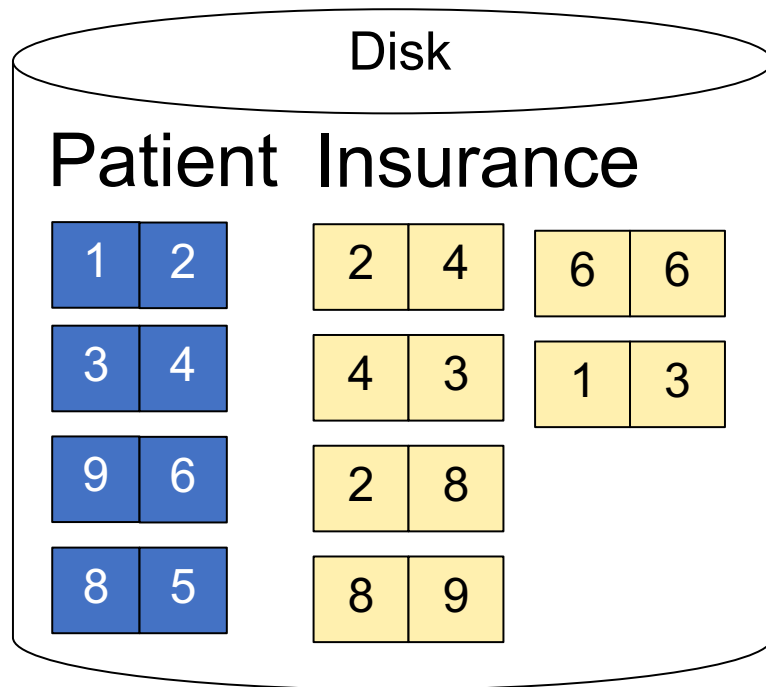


Input buffer for Insurance

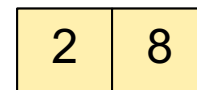


Output buffer

Page-at-a-time Refinement

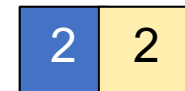


Input buffer for Patient



Input buffer for Insurance

Keep going until read all of Insurance



Output buffer

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

$$\text{Cost: } B(R) + B(R)B(S)$$

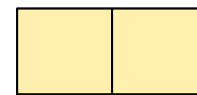
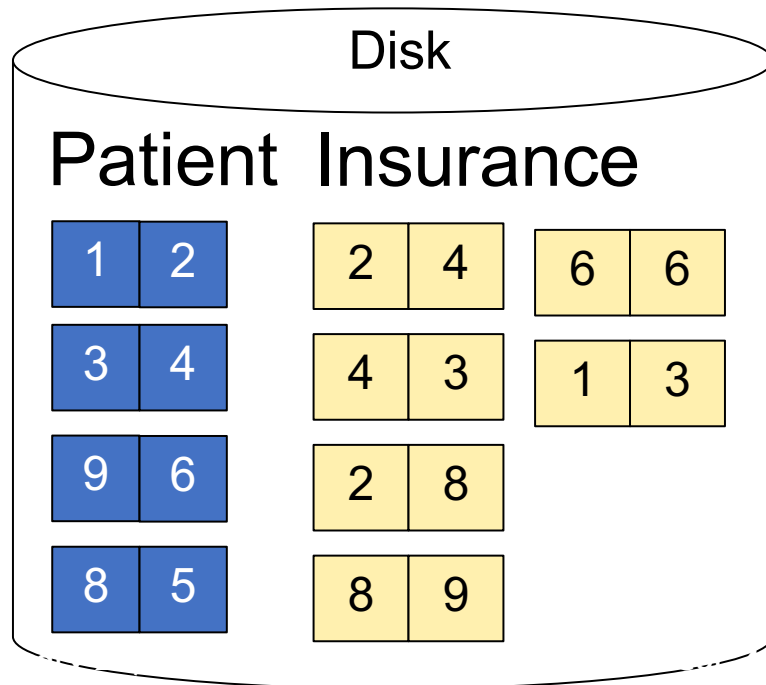
Block-Memory Refinement

```
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

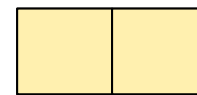
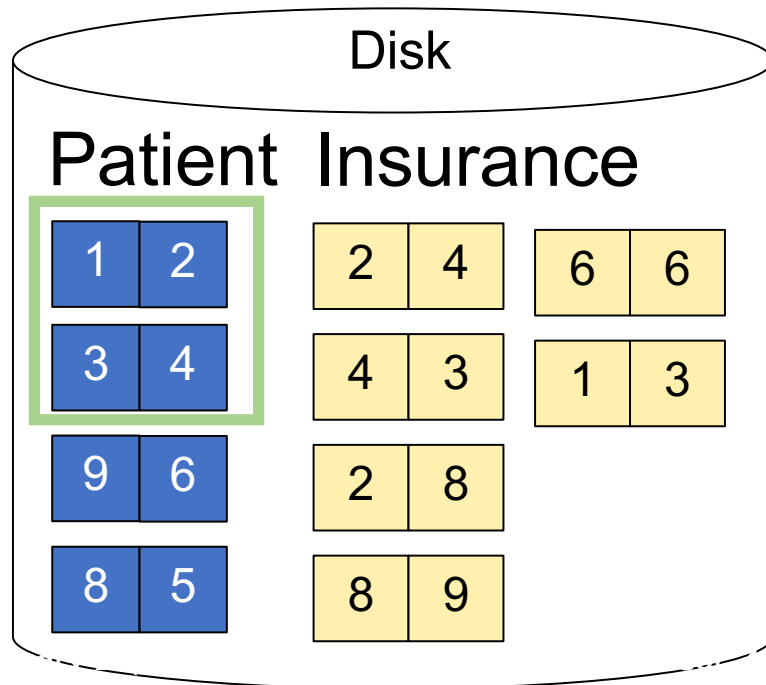
M= 3



No output buffer: stream to output

Block Memory Refinement

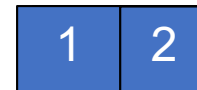
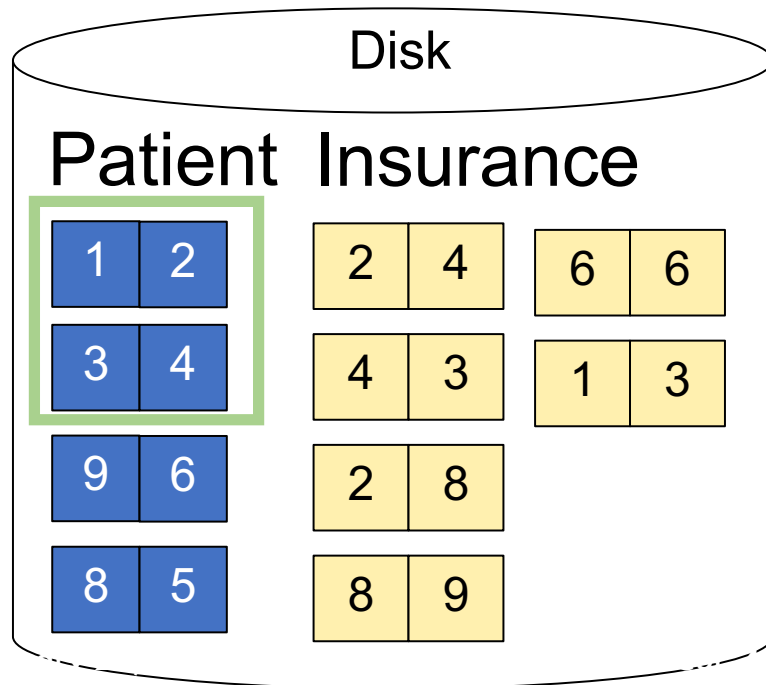
M= 3



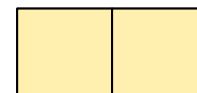
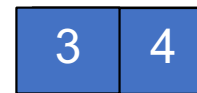
No output buffer: stream to output

Block Memory Refinement

M= 3



Input buffer for Patient

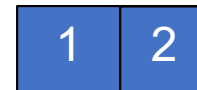
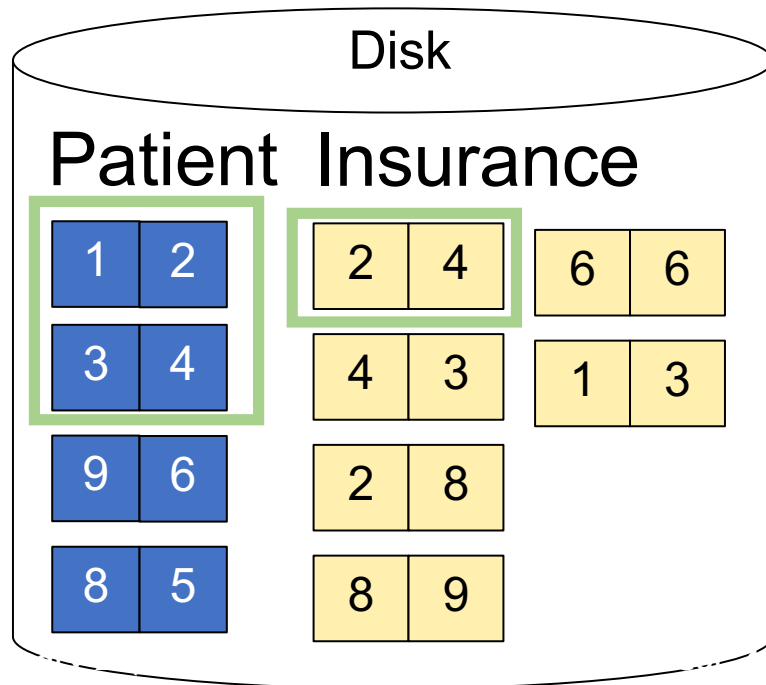


Input buffer for Insurance

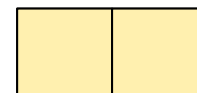
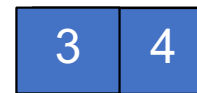
No output buffer: stream to output

Block Memory Refinement

M= 3



Input buffer for Patient

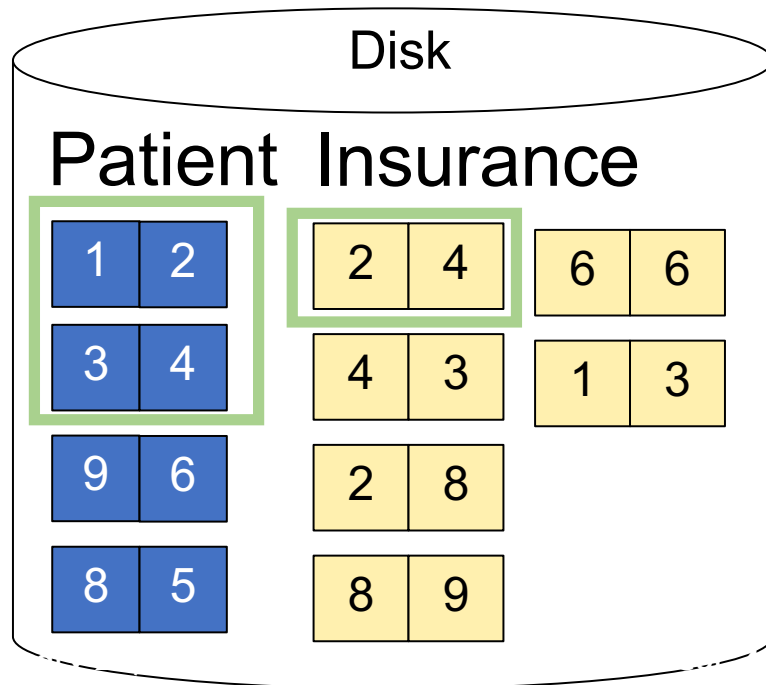


Input buffer for Insurance

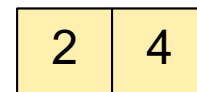
No output buffer: stream to output

Block Memory Refinement

M= 3



Input buffer for Patient

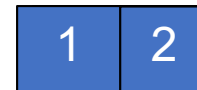
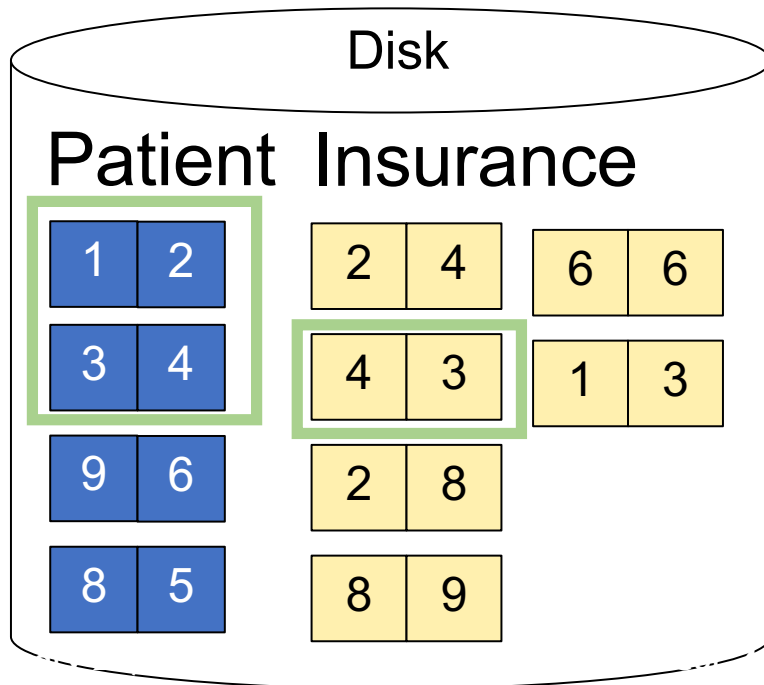


Input buffer for Insurance

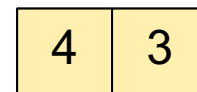
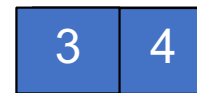
No output buffer: stream to output

Block Memory Refinement

M= 3



Input buffer for Patient

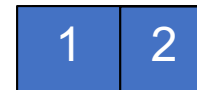
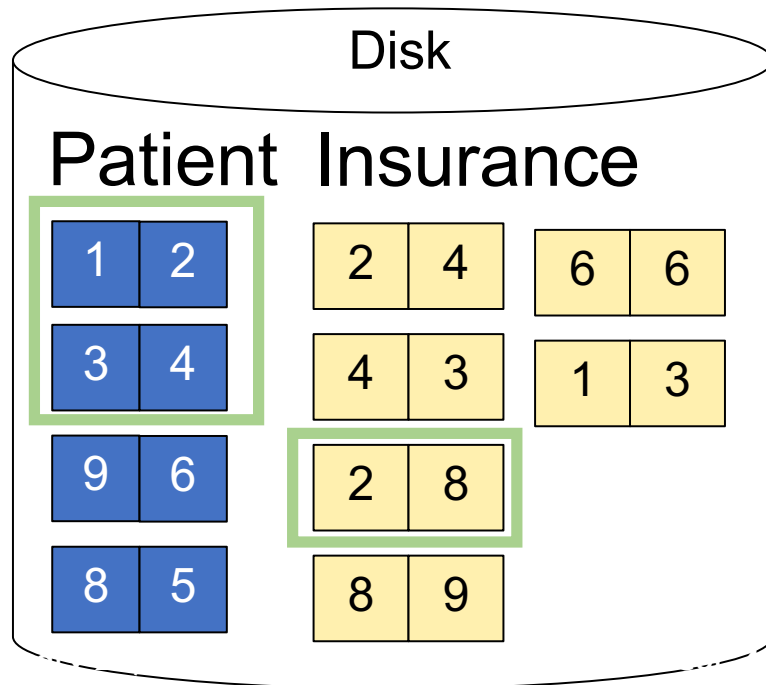


Input buffer for Insurance

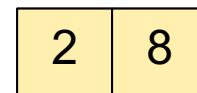
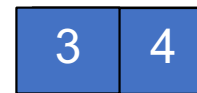
No output buffer: stream to output

Block Memory Refinement

M= 3



Input buffer for Patient

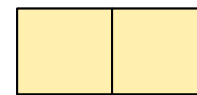
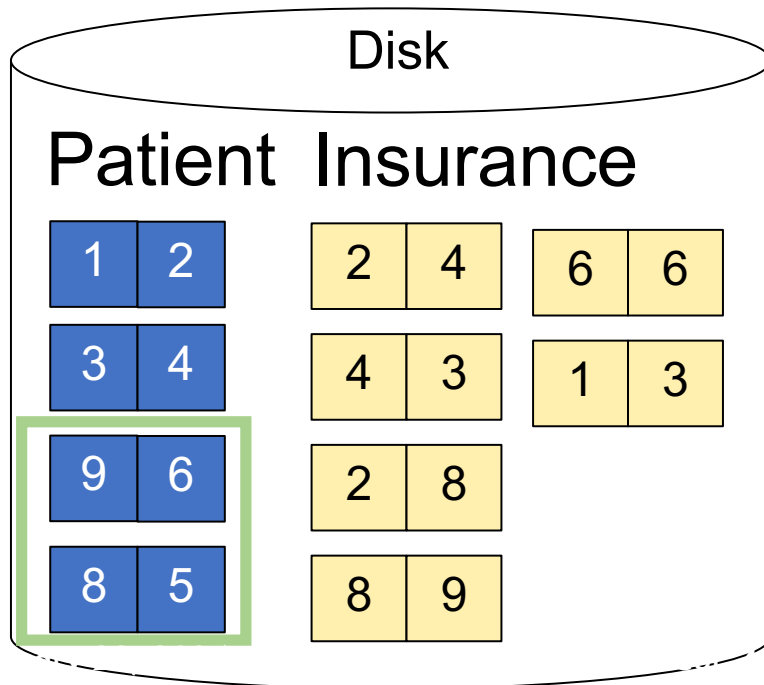


Input buffer for Insurance

No output buffer: stream to output

Block Memory Refinement

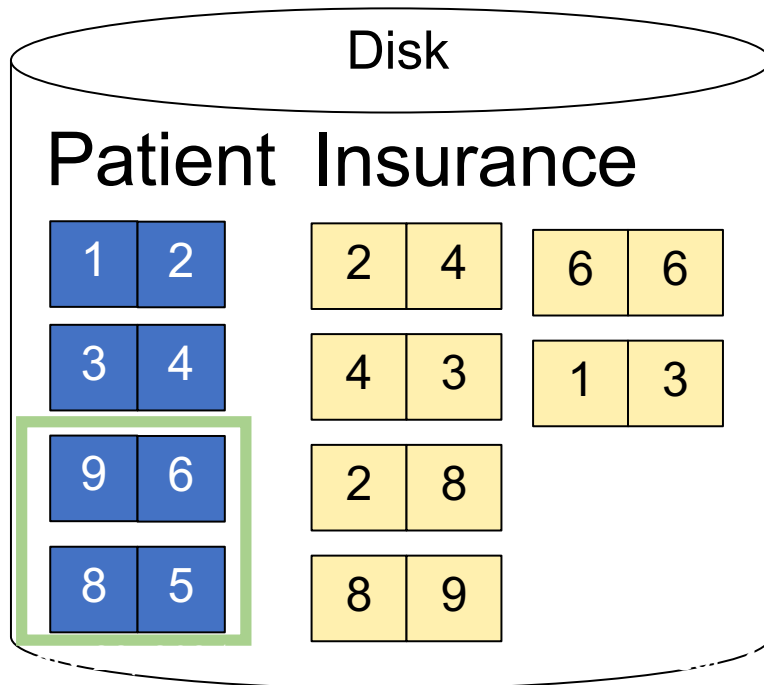
M= 3



No output buffer: stream to output

Block Memory Refinement

M= 3



Input buffer for Patient

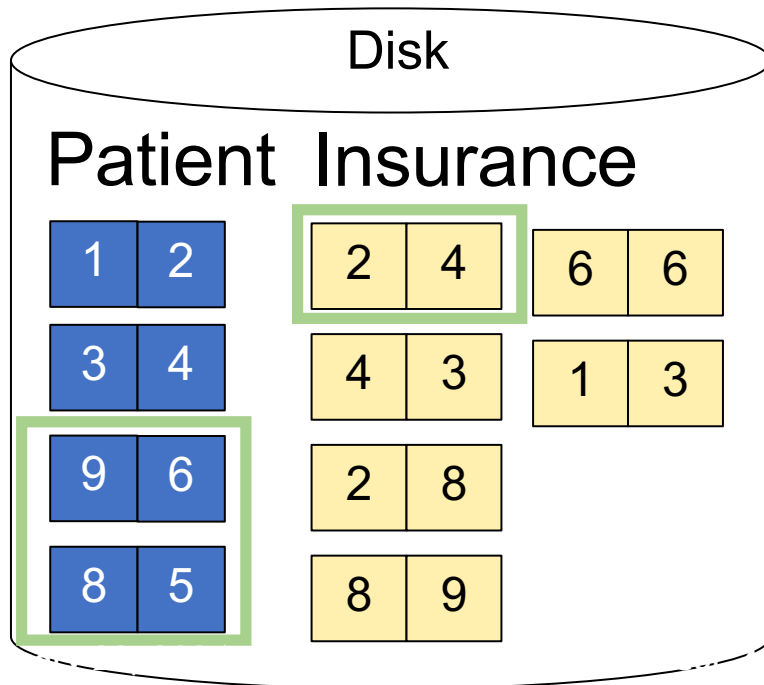


Input buffer for Insurance

No output buffer: stream to output

Block Memory Refinement

M= 3



Input buffer for Patient



Input buffer for Insurance

No output buffer: stream to output

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

```
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

Sort-Merge Join

Sort-merge join: $R \bowtie S$

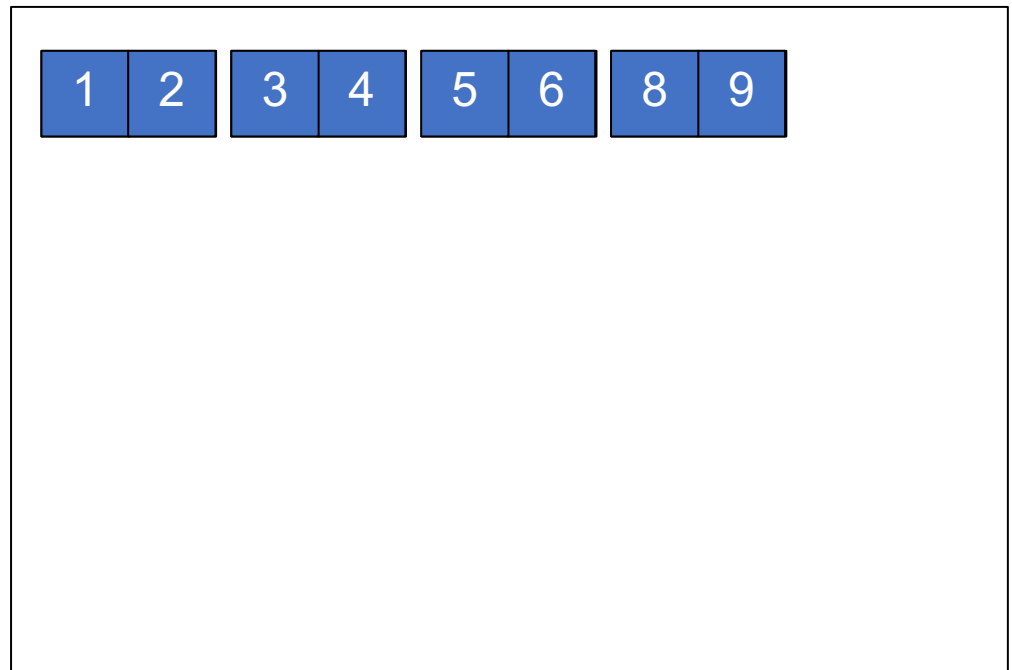
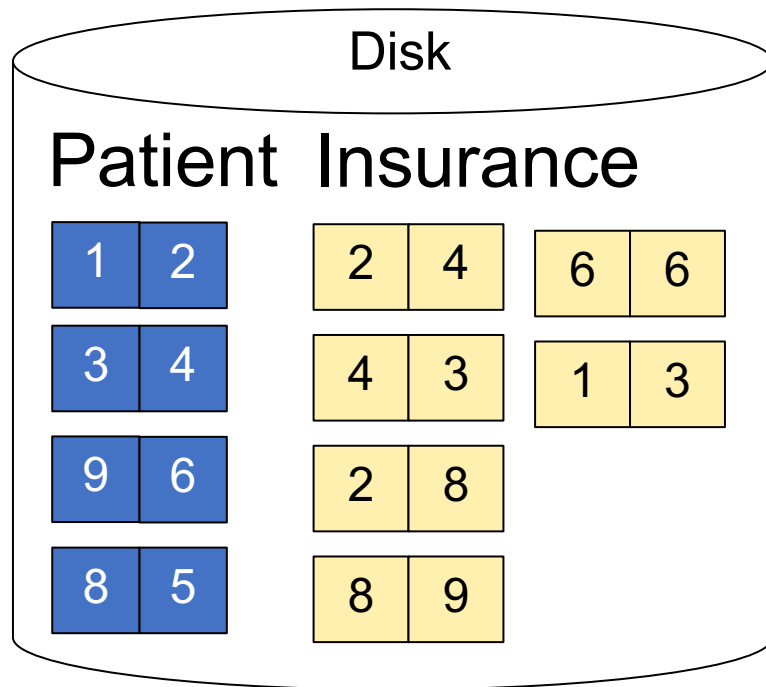
- Scan R and sort in main memory
- Scan S and sort in main memory
- Merge R and S

- Cost: $B(R) + B(S)$
- One pass algorithm when $B(S) + B(R) \leq M$
- 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

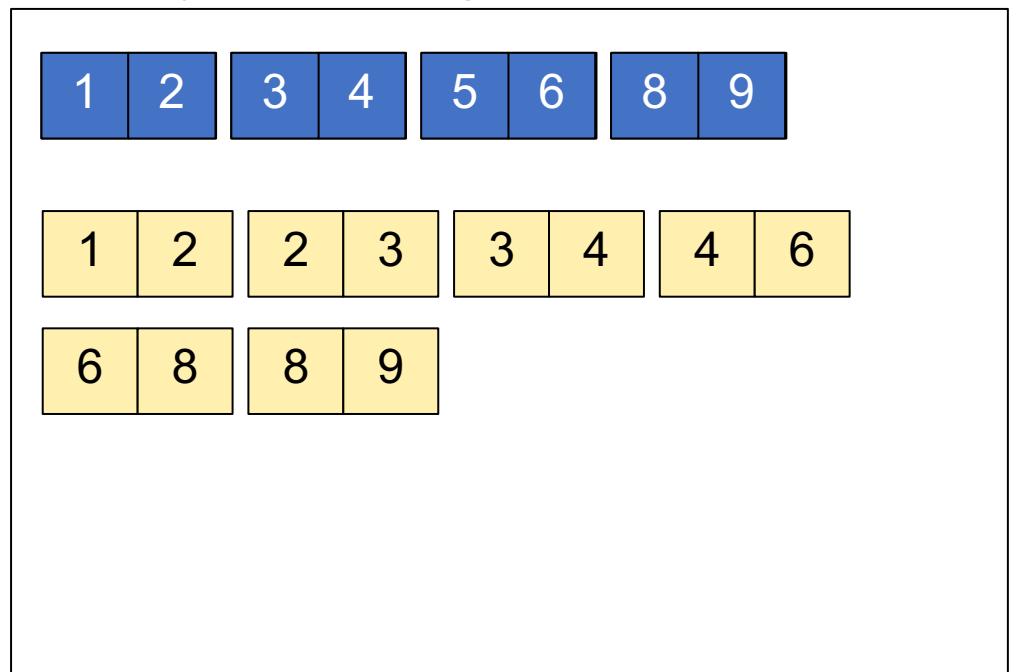
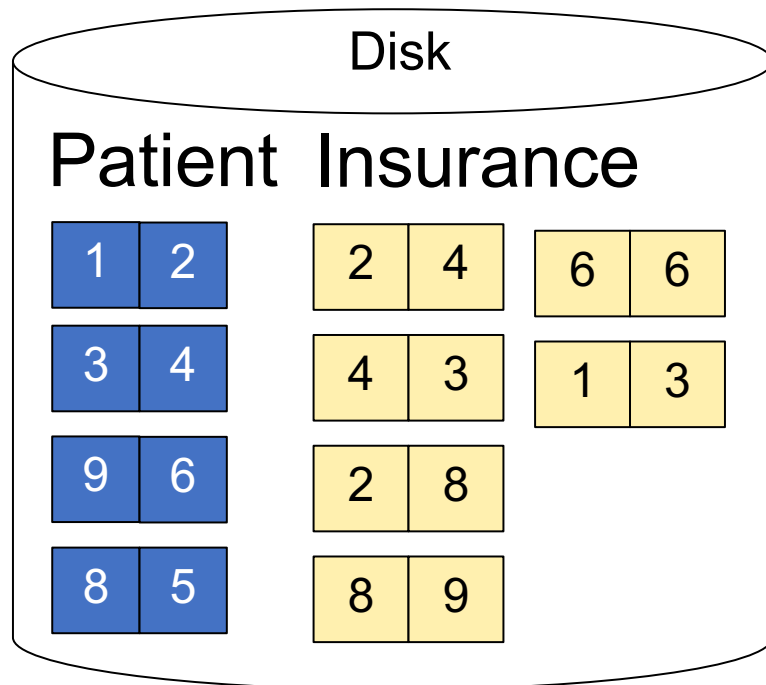
Memory M = 21 pages



Sort-Merge Join Example

Step 2: Scan Insurance and **sort** in memory

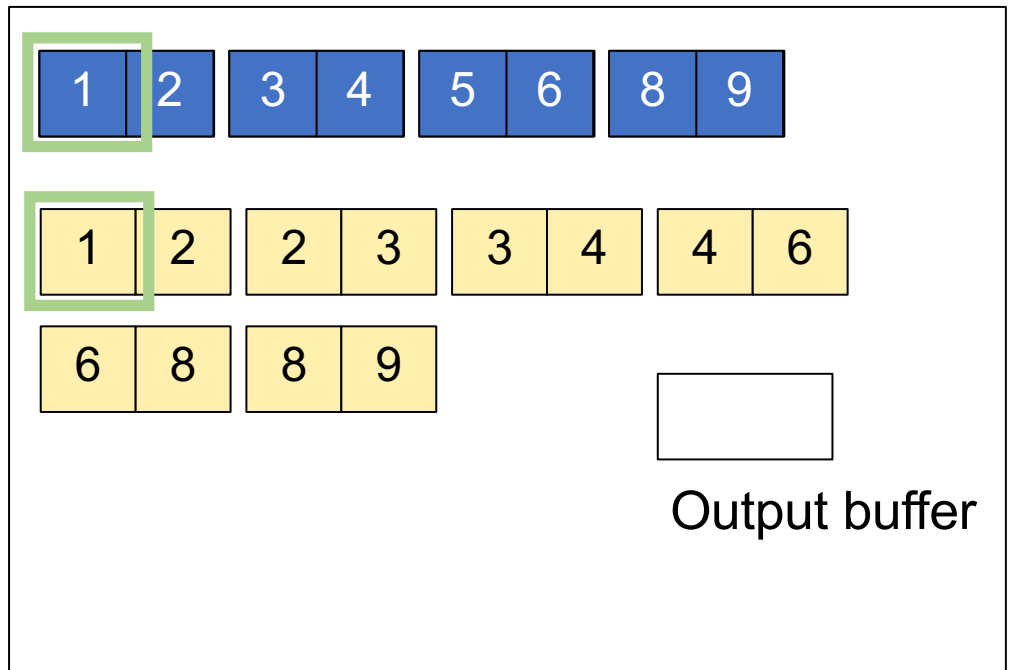
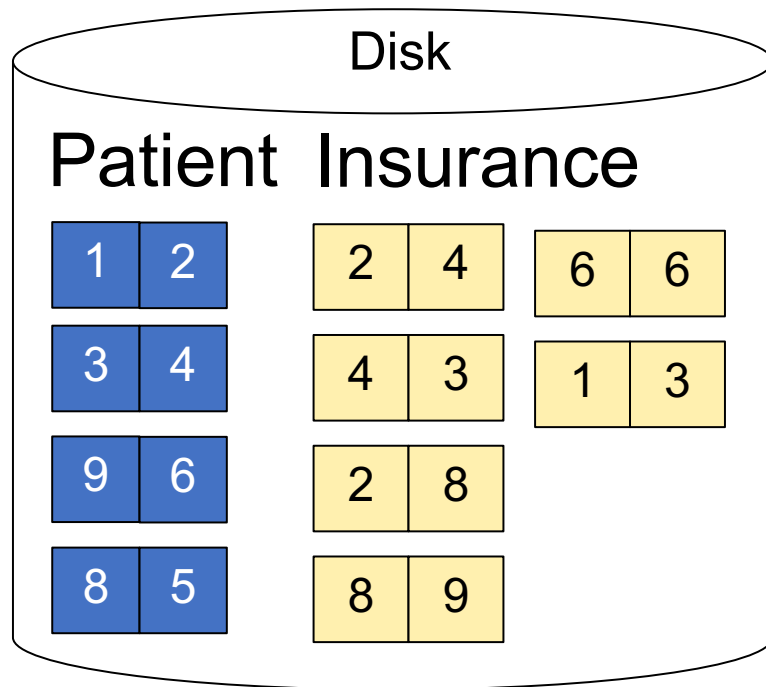
Memory M = 21 pages



Sort-Merge Join Example

Step 3: Merge Patient and Insurance

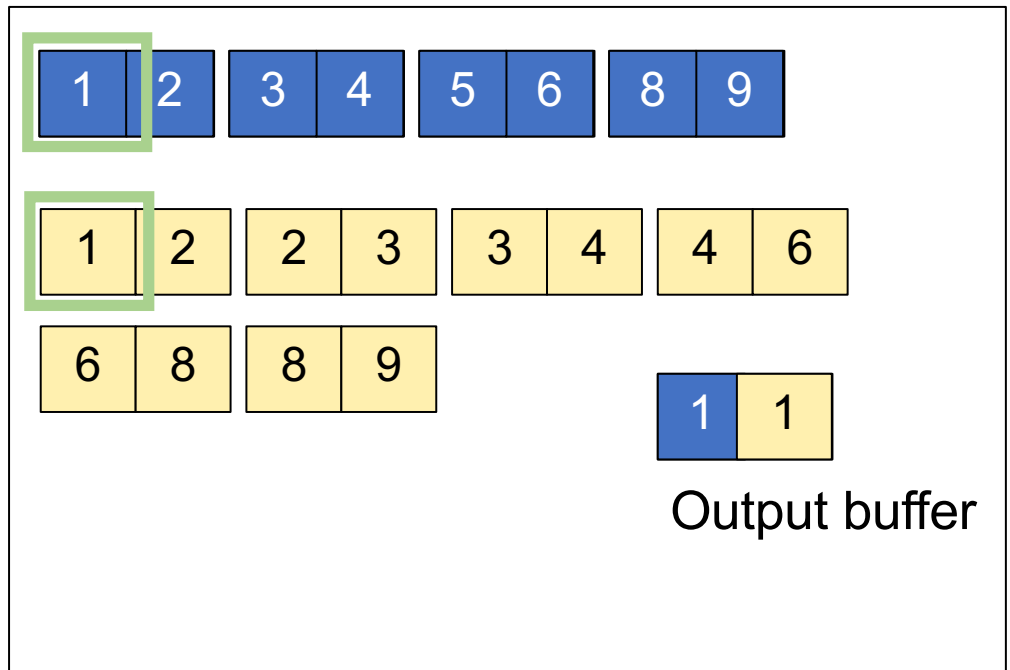
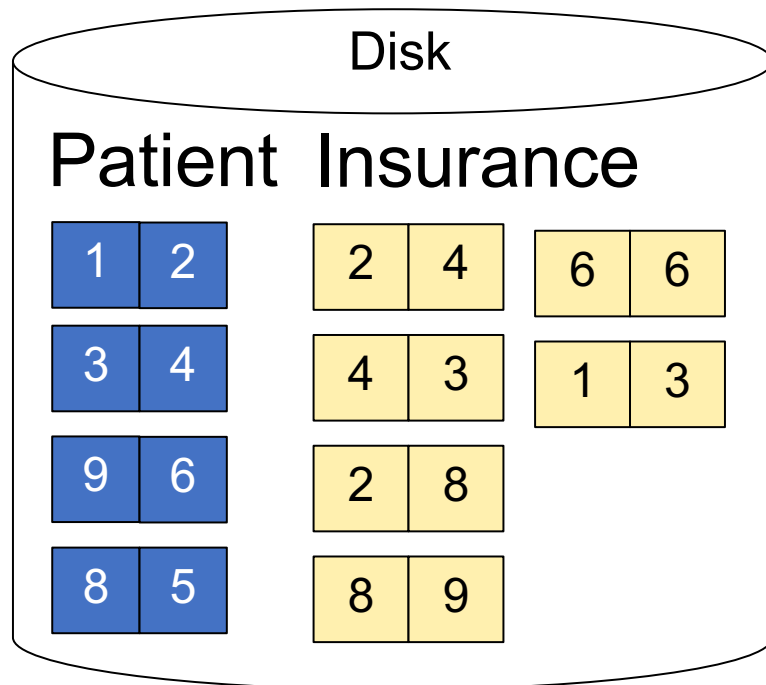
Memory M = 21 pages



Sort-Merge Join Example

Step 3: Merge Patient and Insurance

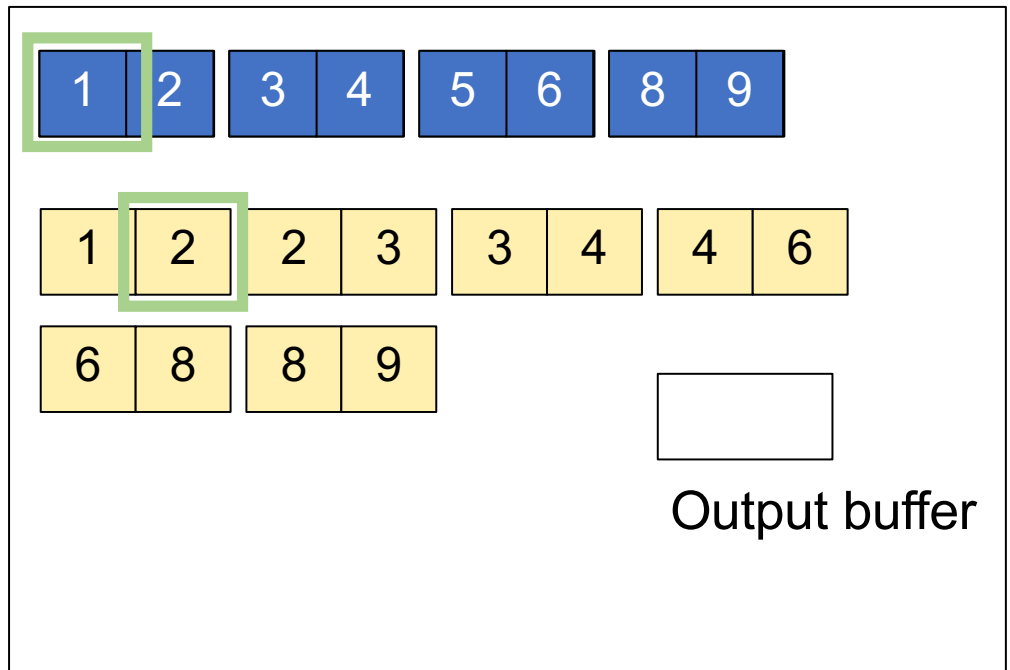
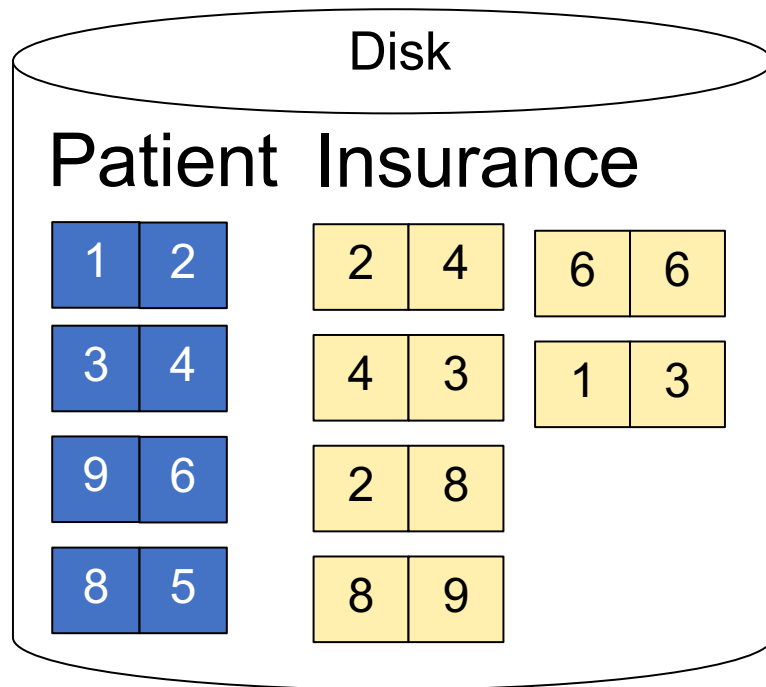
Memory M = 21 pages



Sort-Merge Join Example

Step 3: Merge Patient and Insurance

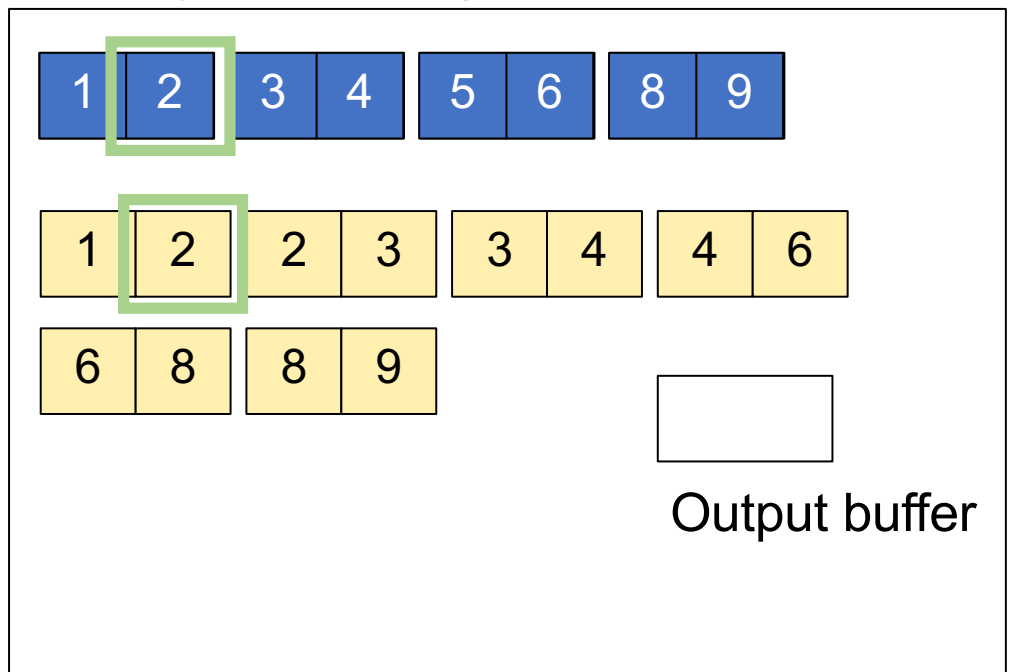
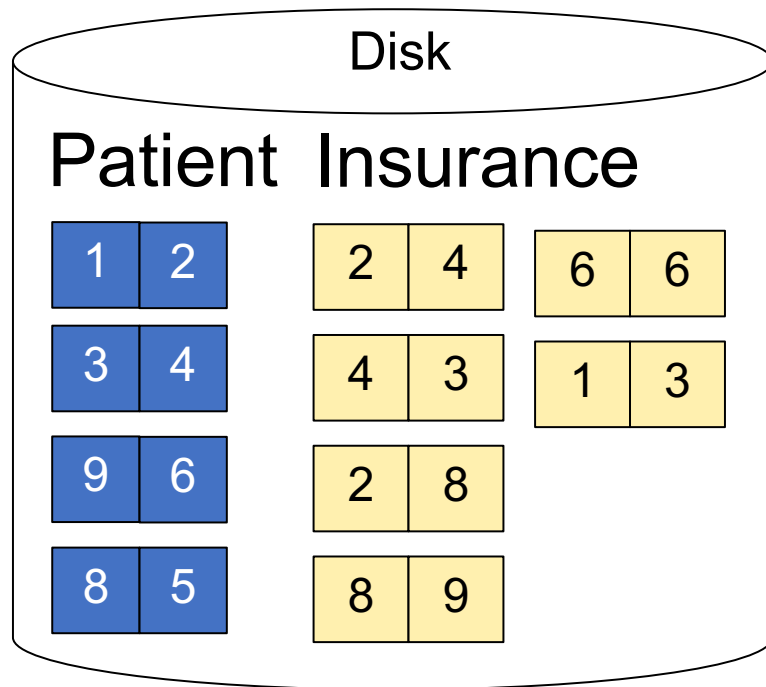
Memory M = 21 pages



Sort-Merge Join Example

Step 3: Merge Patient and Insurance

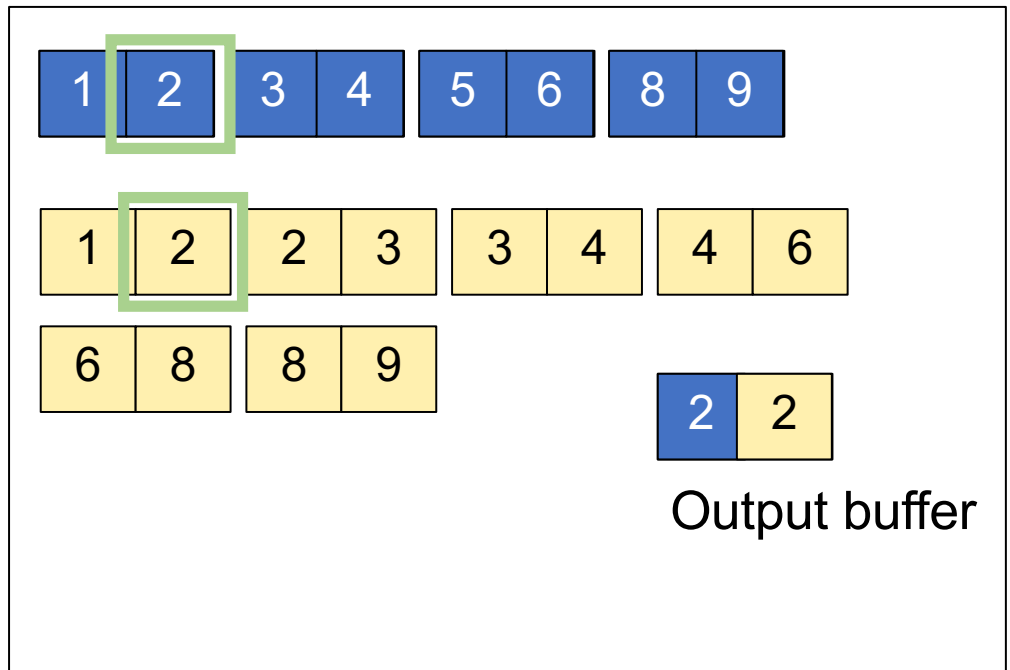
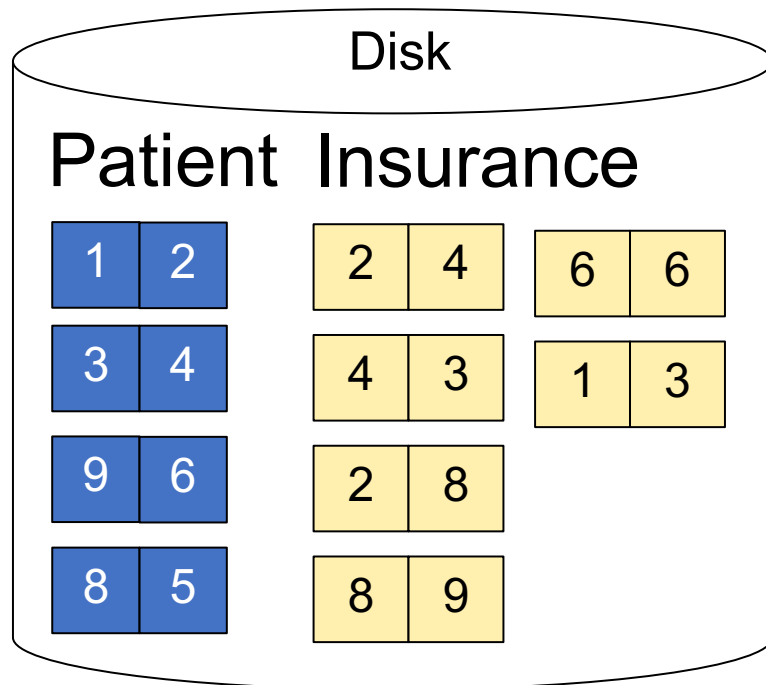
Memory M = 21 pages



Sort-Merge Join Example

Step 3: Merge Patient and Insurance

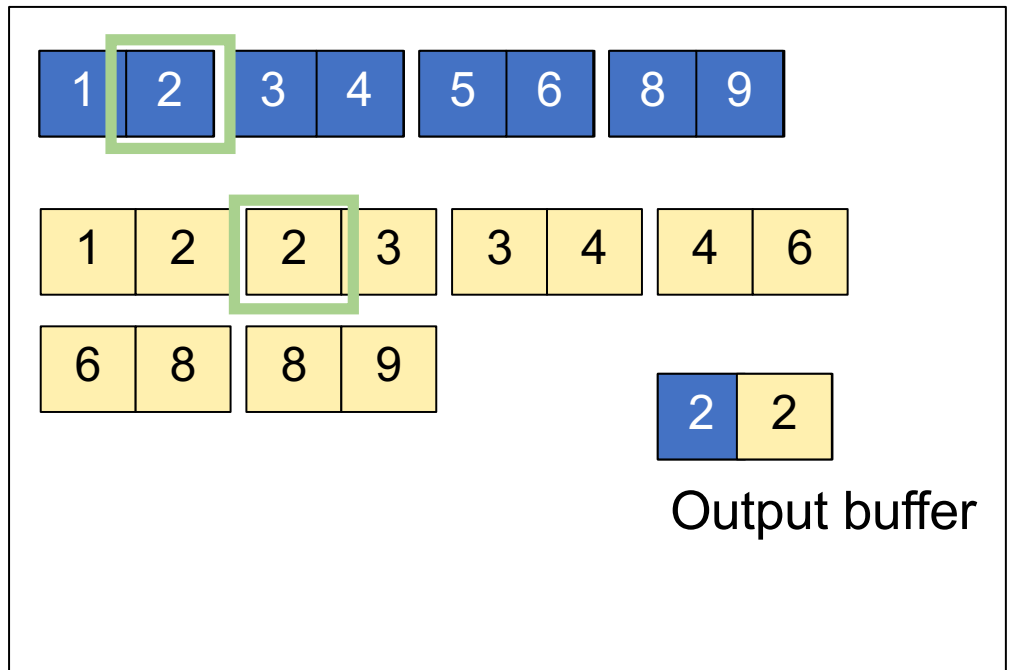
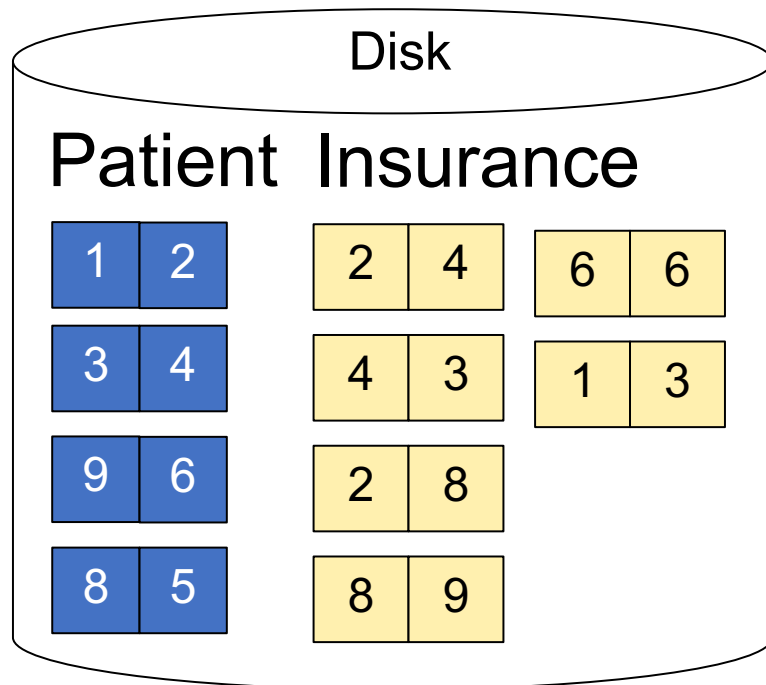
Memory M = 21 pages



Sort-Merge Join Example

Step 3: Merge Patient and Insurance

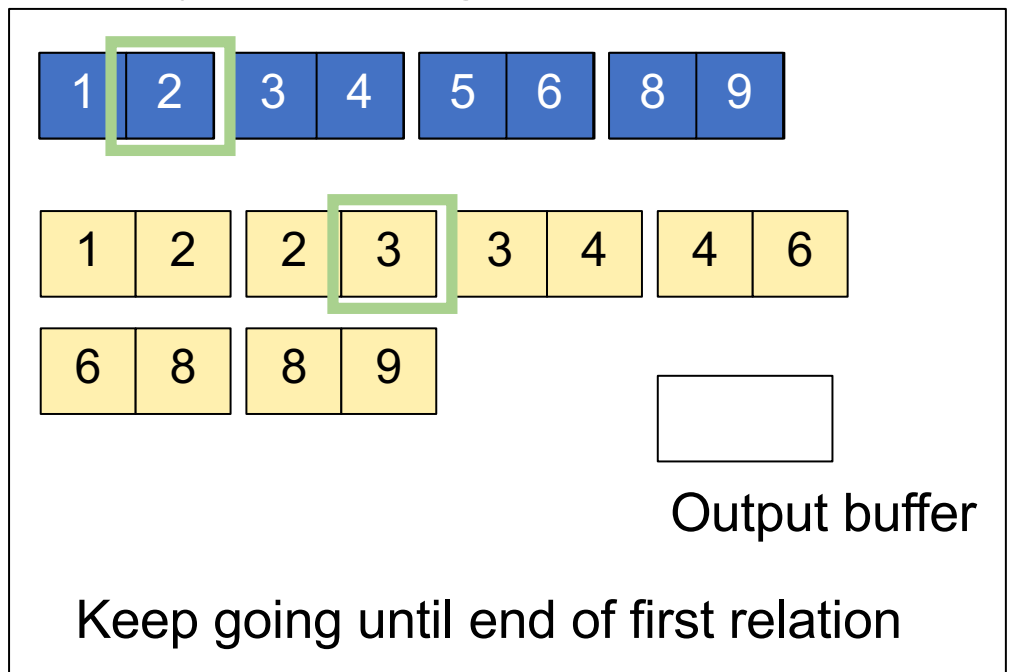
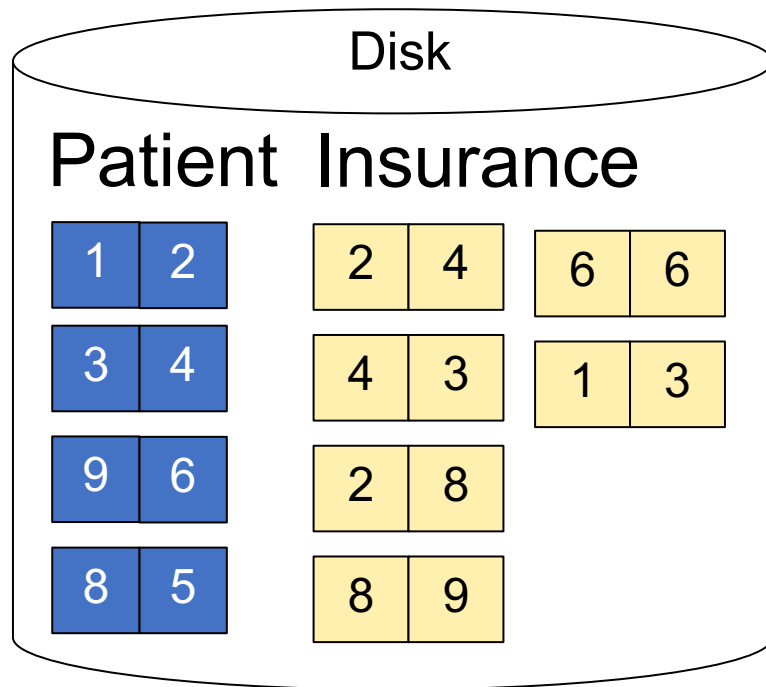
Memory M = 21 pages



Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory M = 21 pages



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

Note: we ignore I/O cost for index pages

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 :

Note: we ignore I/O cost for index pages

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 :

Note: we ignore I/O cost for index pages

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:**

$$\begin{aligned} B(R) &= 2000 \\ T(R) &= 100,000 \\ V(R, a) &= 20 \end{aligned}$$

$$\text{cost of } \sigma_{a=v}(R) = ?$$

- **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:**

$$\begin{aligned} B(R) &= 2000 \\ T(R) &= 100,000 \\ V(R, a) &= 20 \end{aligned}$$

$$\text{cost of } \sigma_{a=v}(R) = ?$$

- **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:**

$$\begin{aligned} B(R) &= 2000 \\ T(R) &= 100,000 \\ V(R, a) &= 20 \end{aligned}$$

$$\text{cost of } \sigma_{a=v}(R) = ?$$

- **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:**

$$\begin{aligned} B(R) &= 2000 \\ T(R) &= 100,000 \\ V(R, a) &= 20 \end{aligned}$$

$$\text{cost of } \sigma_{a=v}(R) = ?$$

- **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 Based Selection

- **Example:**

$$\begin{aligned} B(R) &= 2000 \\ T(R) &= 100,000 \\ V(R, a) &= 20 \end{aligned}$$

$$\text{cost of } \sigma_{a=v}(R) = ?$$

- **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
- Previous nested loop join: cost
 - $B(R) + T(R) * B(S)$
- **Index Nested Loop Join 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

▪ **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

- Fastest algorithm seen so far is one-pass hash join
What if data does not fit in memory?
- Need to process it in multiple passes
- Two key techniques
 - **Sorting**
 - **Hashing**

Basic Terminology

- A run in a sequence is an increasing subsequence
- What are the runs?

2, 4, 99, 103, 88, 77, 3, 79, 100, 2, 50

Basic Terminology

- A run in a sequence is an increasing subsequence
- What are the runs?

2, 4, 99, 103, | 88, | 77, | 3, 79, 100, | 2, 50

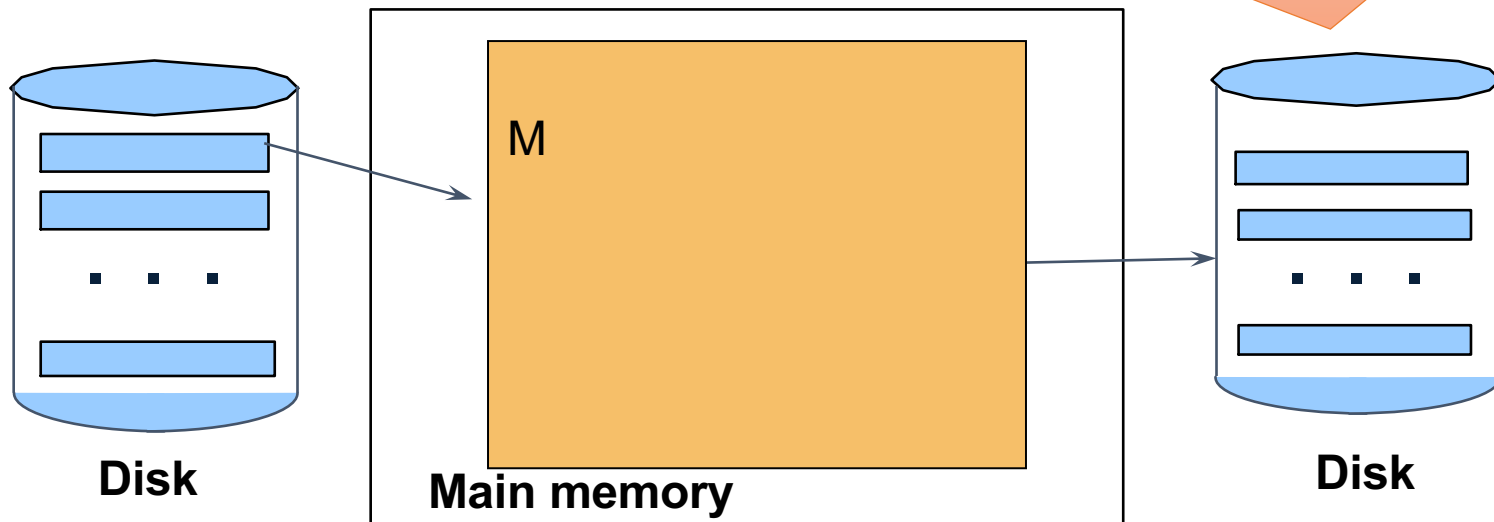
External Merge-Sort: Step 1

Phase one: load M blocks in memory, sort, send to disk, repeat

External Merge-Sort: Step 1

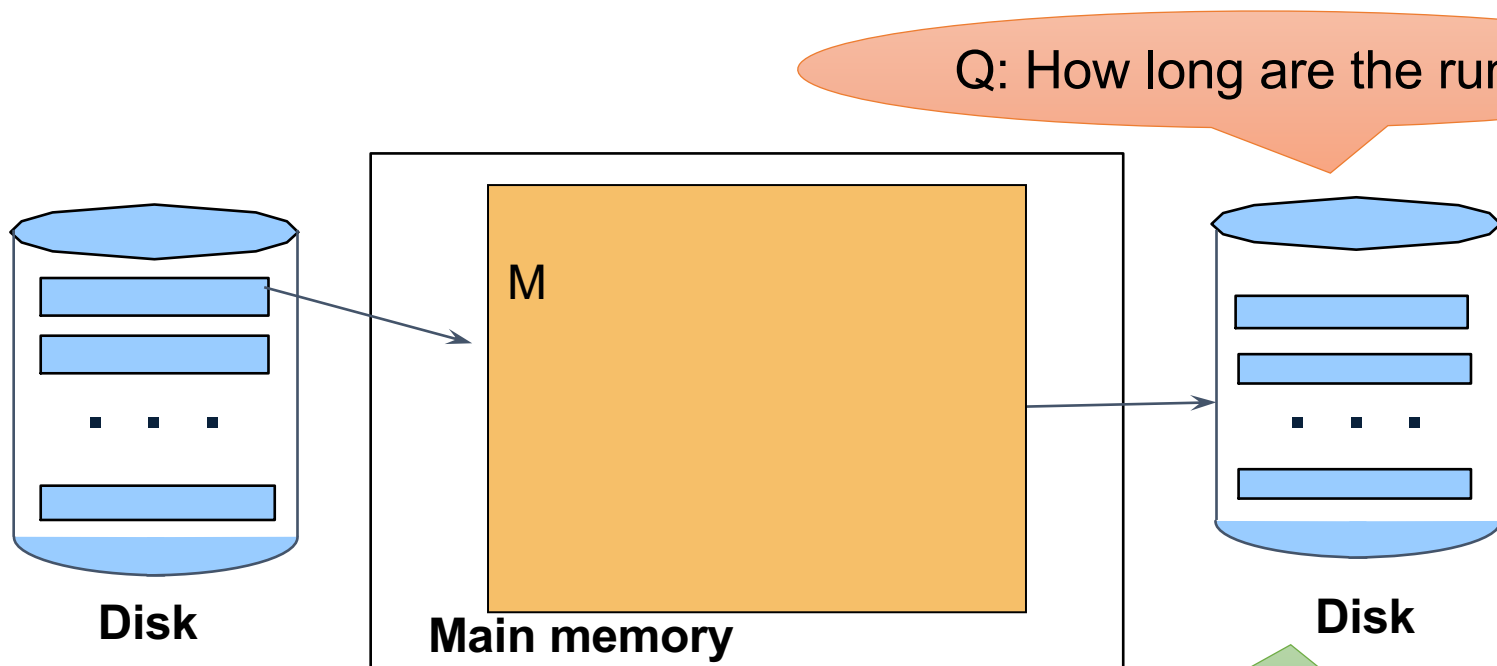
Phase one: load M blocks in memory, sort, send to disk, repeat

Q: How long are the runs?



External Merge-Sort: Step 1

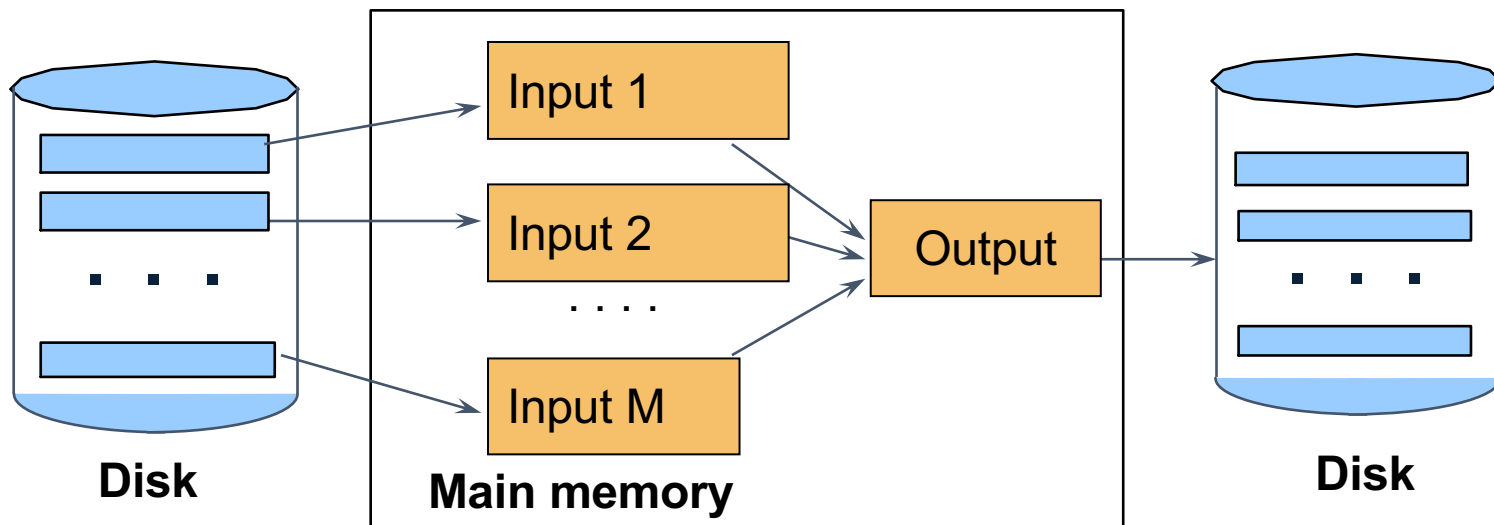
Phase one: load M blocks in memory, sort, send to disk, repeat



A: Length = M blocks

Phase two: merge M runs into a bigger run

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



Example

- Merging three runs to produce a longer run:

0, 14, 33, 88, 92, 192, 322

2, 4, 7, 43, 78, 103, 523

1, 6, 9, 12, 33, 52, 88, 320

Output:

0

Example

- Merging three runs to produce a longer run:

0, **14**, 33, 88, 92, 192, 322

2, 4, 7, 43, 78, 103, 523

1, 6, 9, 12, 33, 52, 88, 320

Output:

0, **?**

Example

- Merging three runs to produce a longer run:

0, **14**, 33, 88, 92, 192, 322

2, 4, 7, 43, 78, 103, 523

1, **6**, 9, 12, 33, 52, 88, 320

Output:

0, 1, **?**

Example

- Merging three runs to produce a longer run:

0, **14**, **33**, **88**, **92**, **192**, **322**

2, 4, 7, **43**, **78**, **103**, **523**

1, 6, **9**, **12**, **33**, **52**, **88**, **320**

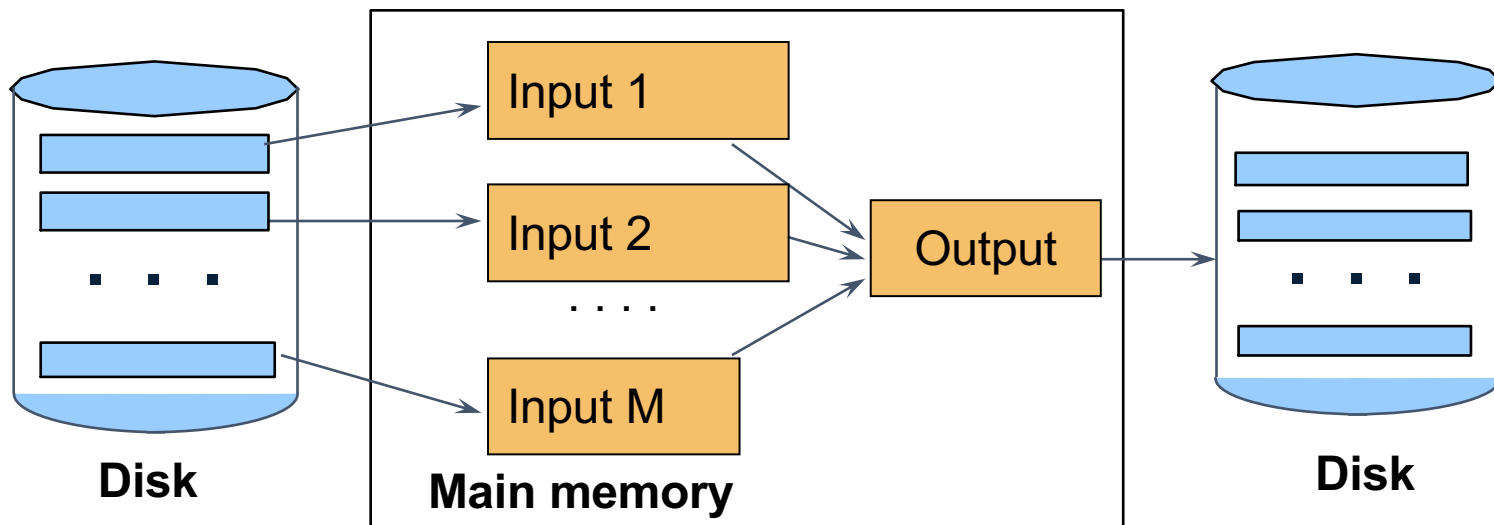
Output:

0, **1**, **2**, **4**, **6**, **7**, **?**

External Merge-Sort: Step 2

Phase two: merge M runs into a bigger run

- Merge $M - 1$ runs into a new run
- Result: runs of length $M(M - 1) \approx M^2$



If approx. $B \leq M^2$ then we are done

Cost of External Merge Sort

- Assumption: $B(R) \leq M^2$
- Read+write+read = $3B(R)$

Discussion

- What does $B(R) \leq M^2$ mean?
- How large can R be?

Discussion

- What does $B(R) \leq M^2$ mean?
- How large can R be?
- Example:
 - Page size = 32KB
 - Memory size 32GB: $M = 10^6$ -pages

Discussion

- What does $B(R) \leq M^2$ mean?
- How large can R be?

- Example:
 - Page size = 32KB
 - Memory size 32GB: $M = 10^6$ pages

- R can be as large as 10^{12} pages
 - 32×10^{15} Bytes = 32 PB

Merge-Join

Join $R \bowtie S$

- How?.....

Merge-Join

Join $R \bowtie S$

- Step 1a: generate initial runs for R
- Step 1b: generate initial runs for S
- Step 2: merge and join
 - Either merge first and then join
 - Or merge & join at the same time

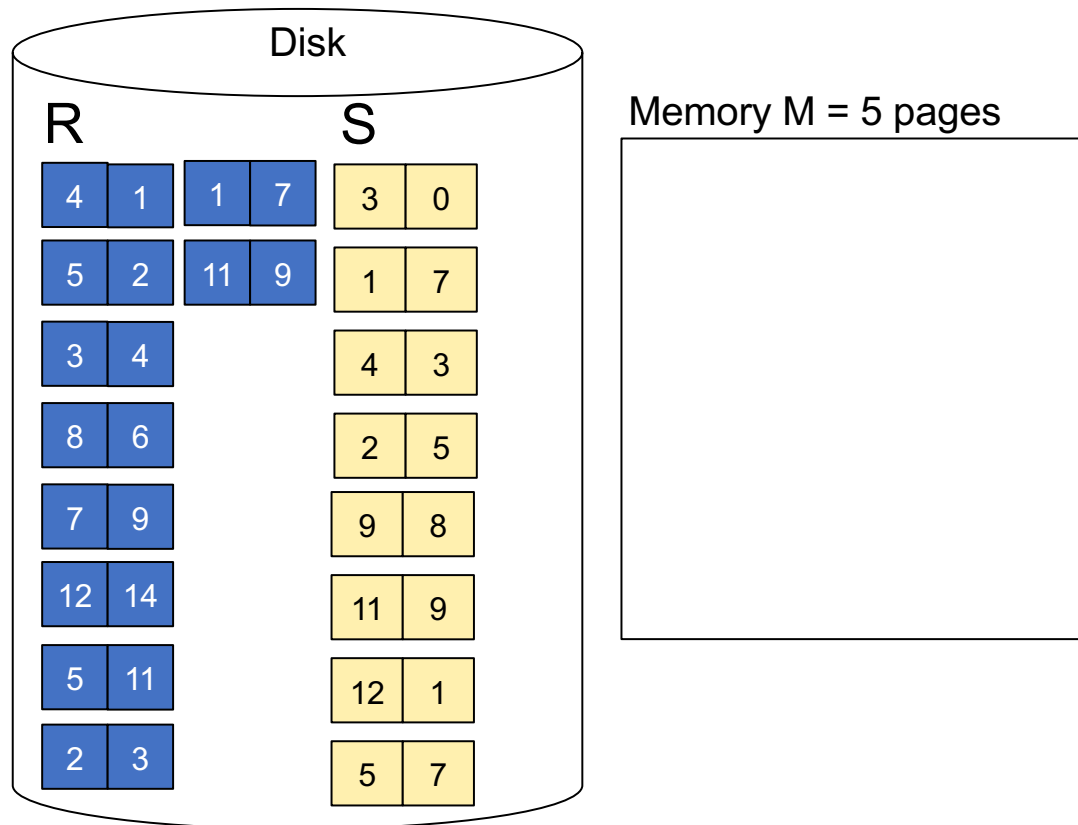
Merge-Join Example

Setup: Want to join R and S

Relation R has 10 pages with 2 tuples per page

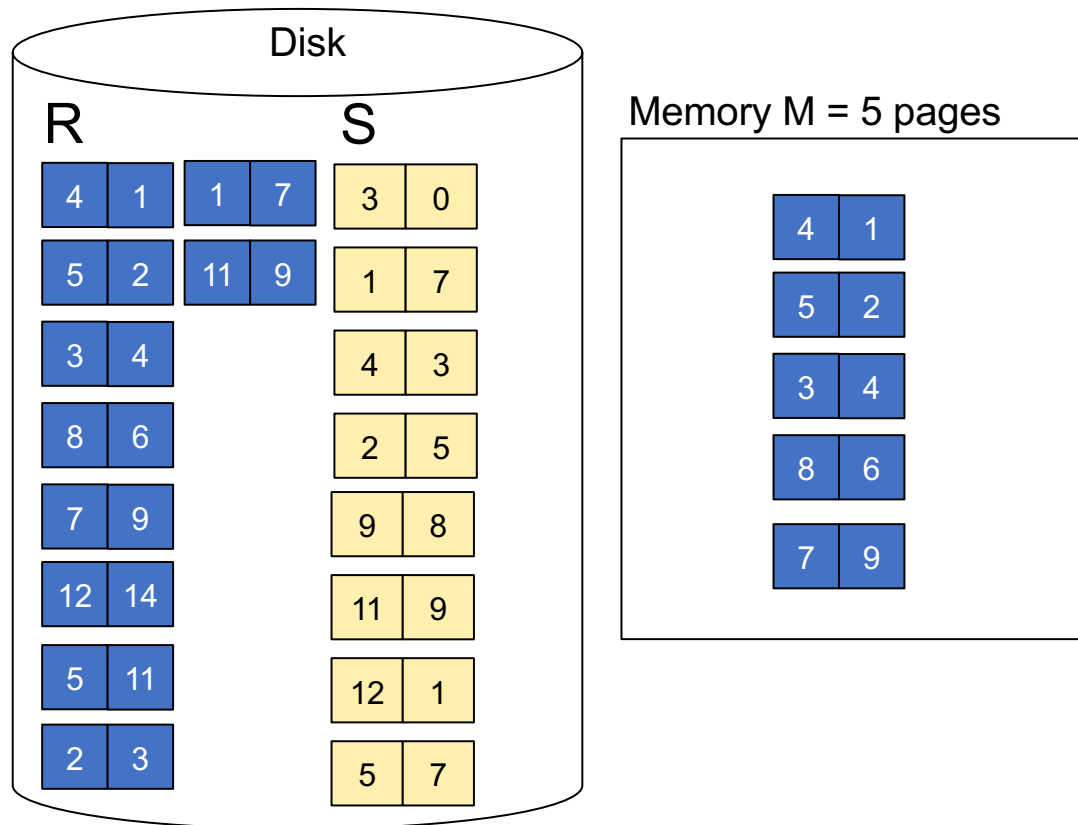
Relation S has 8 pages with 2 tuples per page

Values shown are values of join attribute for each given tuple



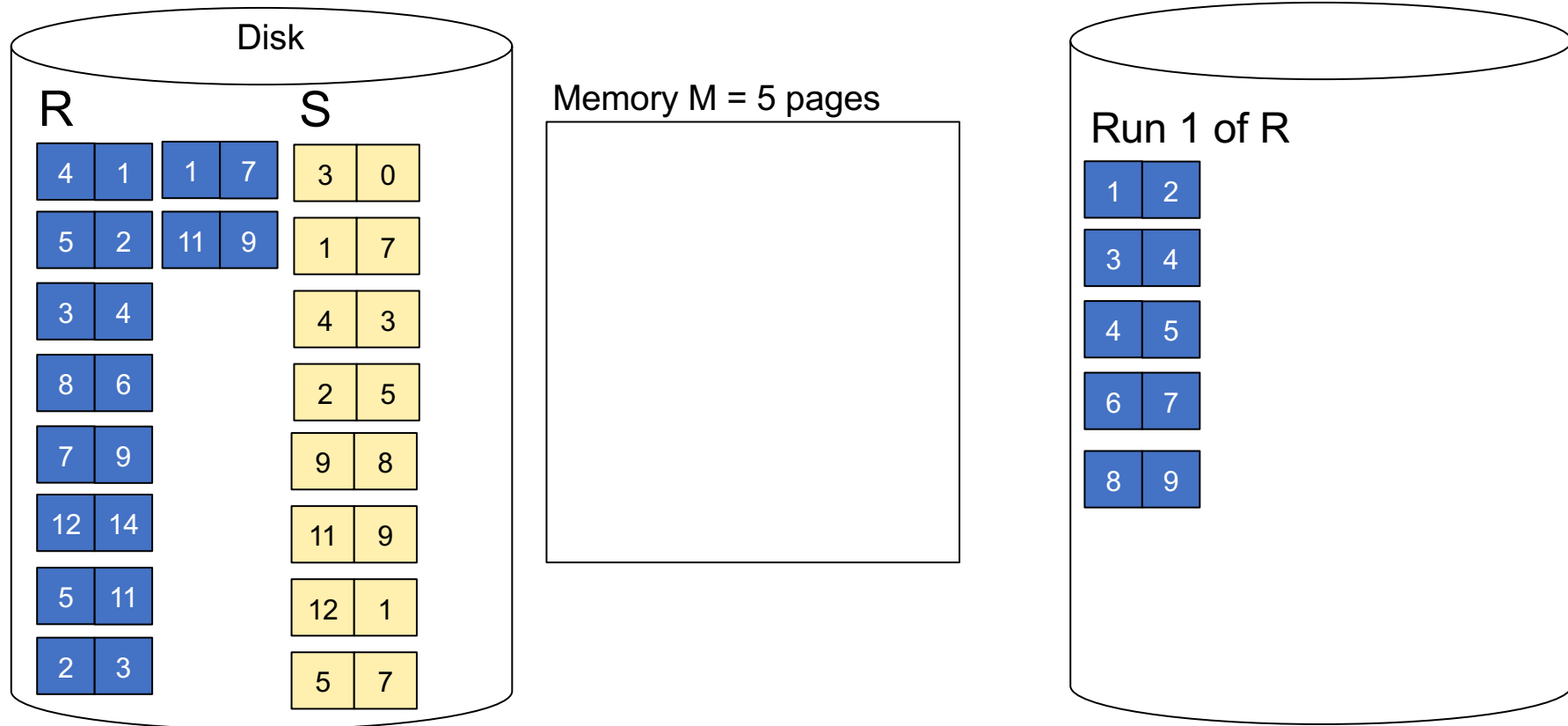
Merge-Join Example

Step 1: Read M pages of R and sort in memory



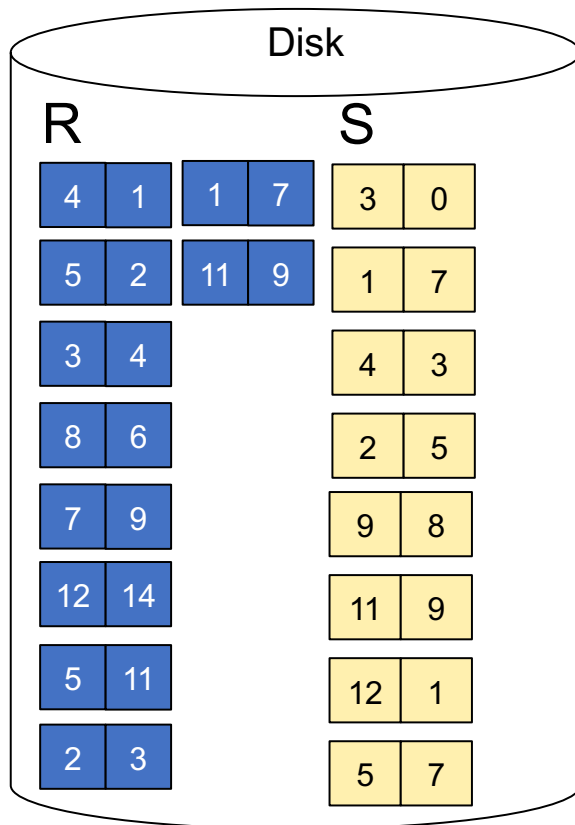
Merge-Join Example

Step 1: Read M pages of R and sort in memory, then write to disk

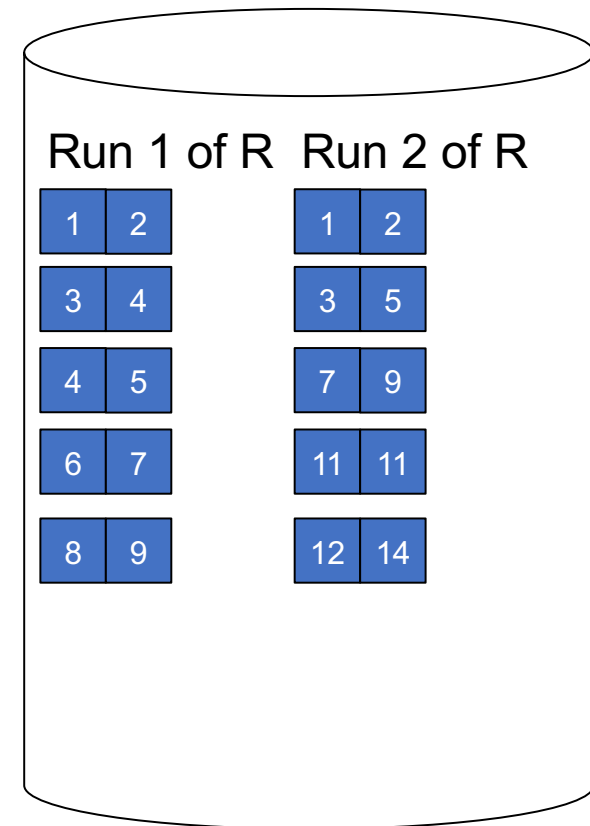
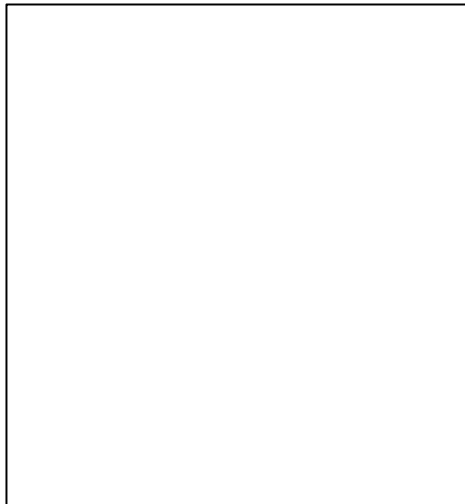


Merge-Join Example

Step 1: Repeat for next M pages until all R is processed

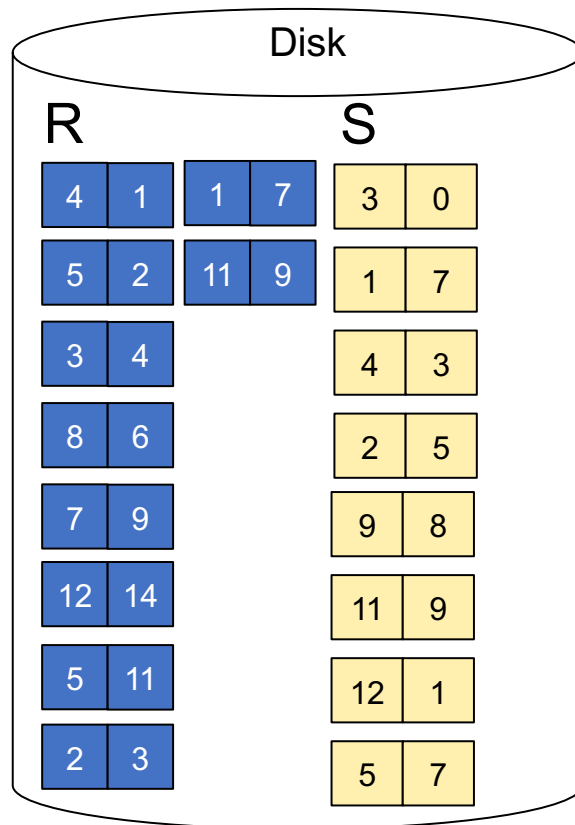


Memory M = 5 pages

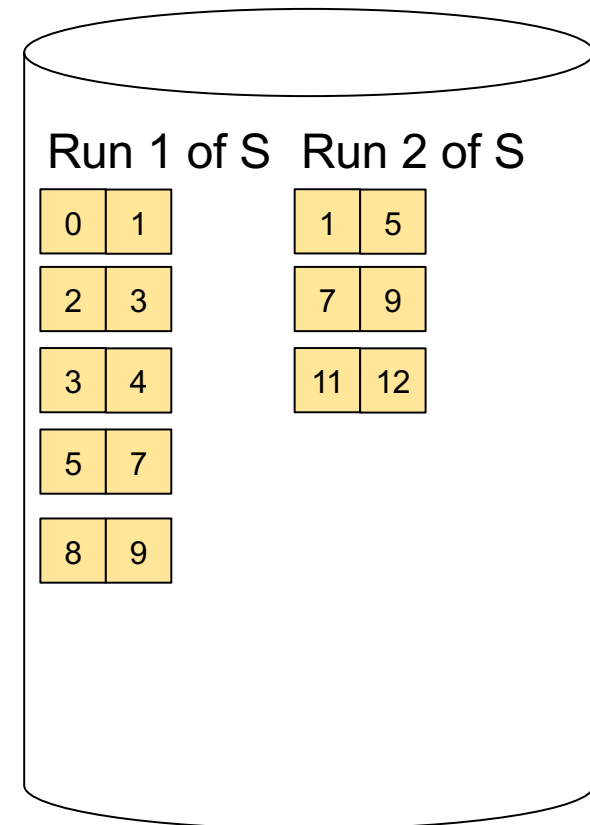


Merge-Join Example

Step 1: Do the same with S

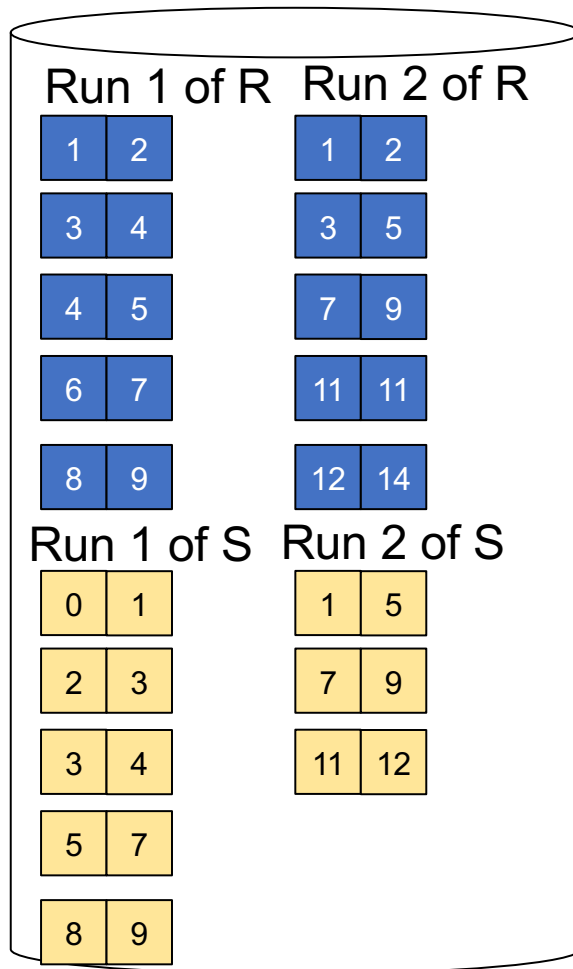


Memory M = 5 pages



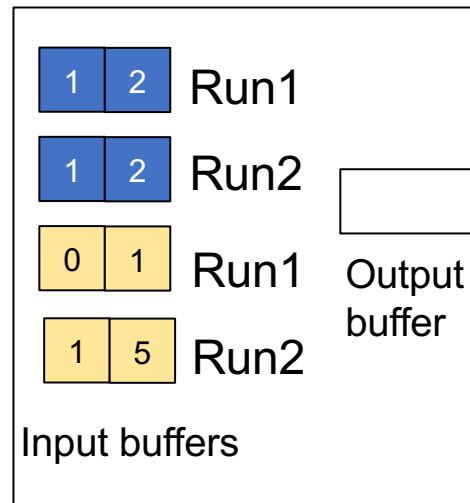
Merge-Join Example

Step 2: Join while merging sorted runs



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

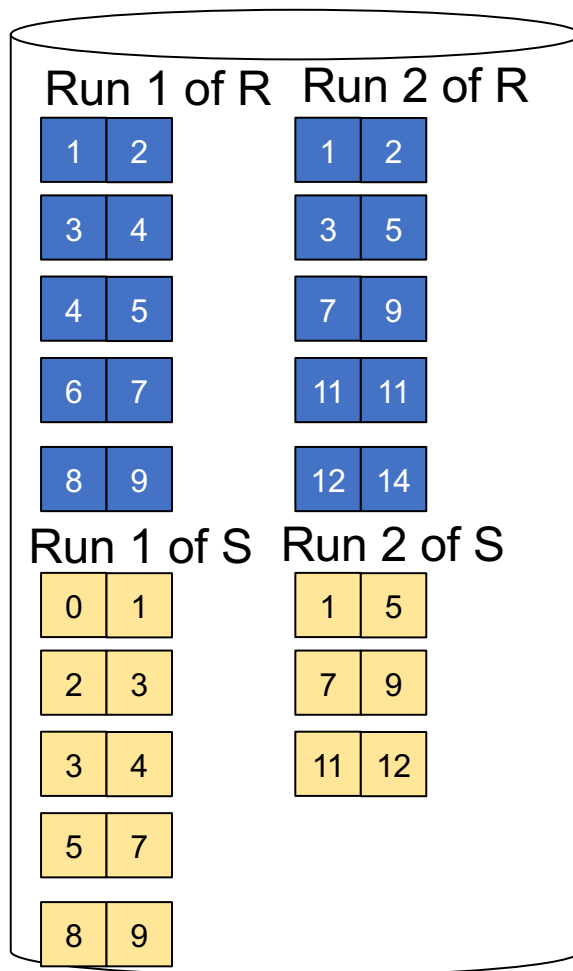
Memory $M = 5$ pages



Step 2: Join while merging
Output tuples

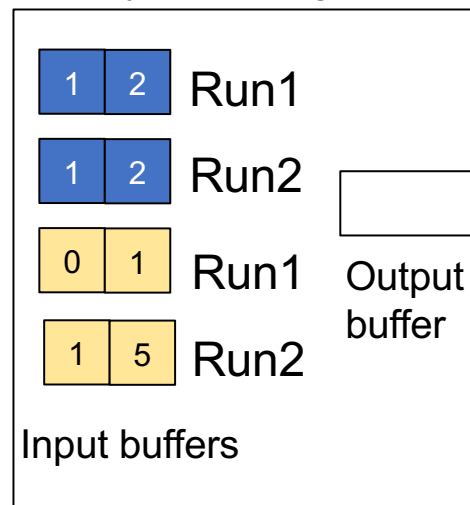
Merge-Join Example

Step 2: Join while merging sorted runs



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

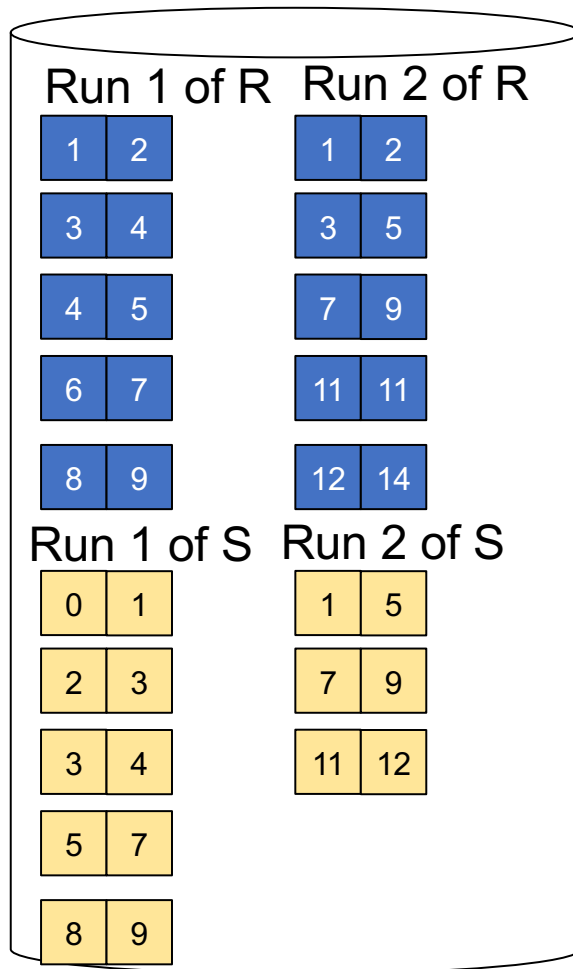
Memory $M = 5$ pages



Step 2: Join while merging
Output tuples

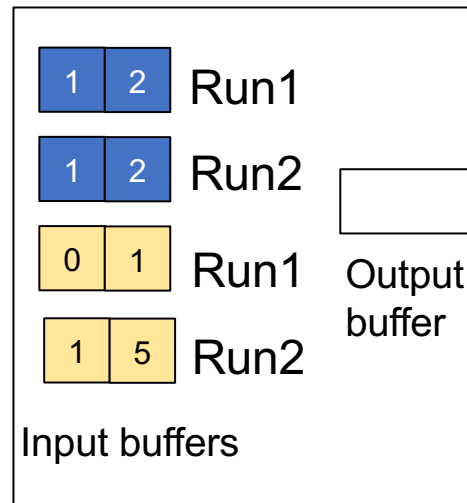
Merge-Join Example

Step 2: Join while merging sorted runs



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

Memory $M = 5$ pages

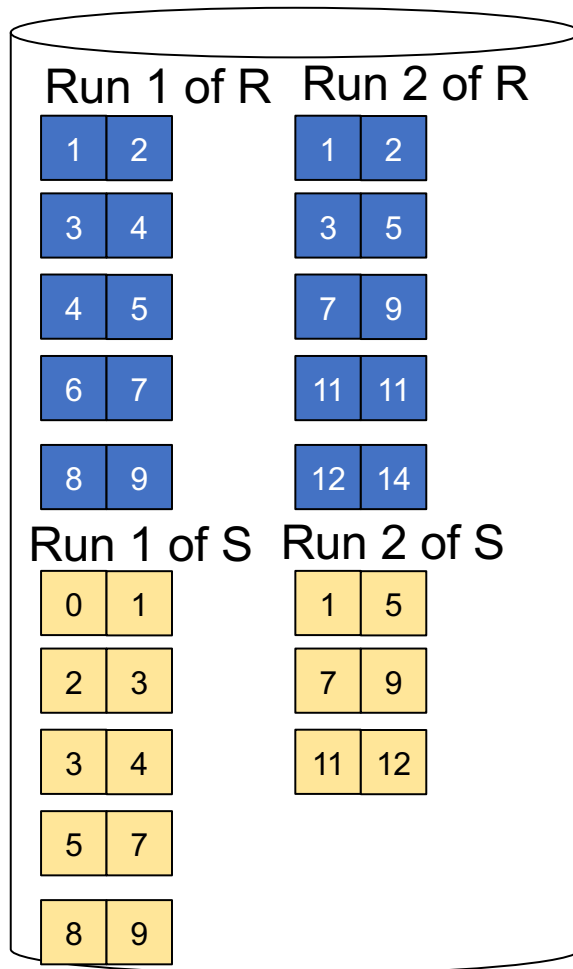


Step 2: Join while merging
Output tuples

- (1,1)
- (1,1)
- (1,1)
- (1,1)

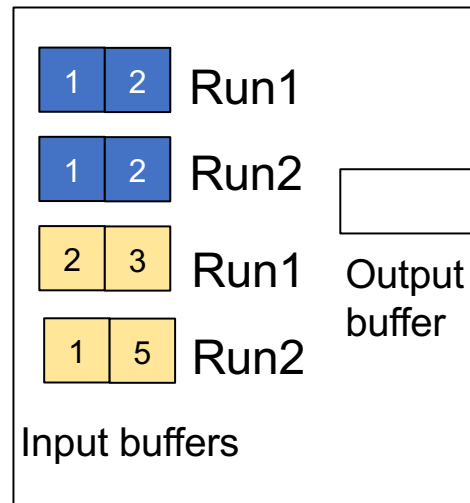
Merge-Join Example

Step 2: Join while merging sorted runs



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

Memory $M = 5$ pages

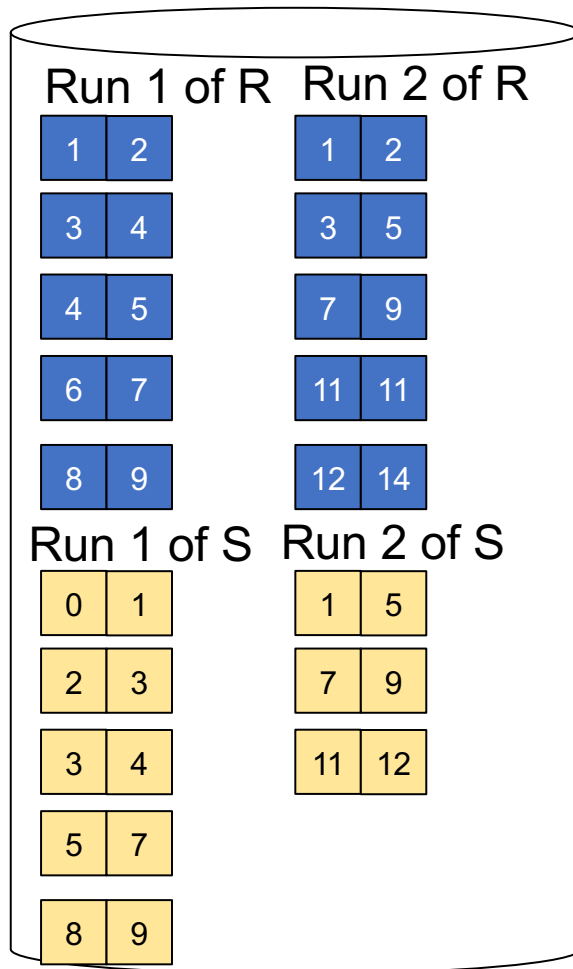


Step 2: Join while merging
Output tuples

- (1,1)
- (1,1)
- (1,1)
- (1,1)

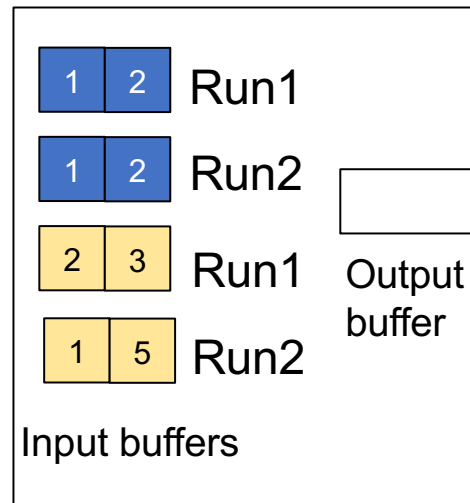
Merge-Join Example

Step 2: Join while merging sorted runs



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

Memory $M = 5$ pages

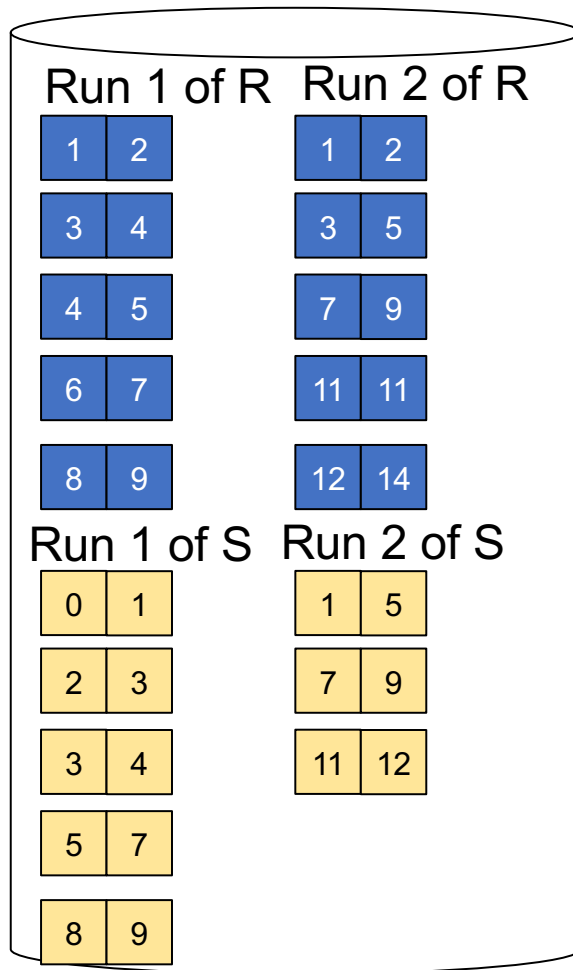


Step 2: Join while merging
Output tuples

- (1,1)
- (1,1)
- (1,1)
- (1,1)
- (2,2)
- (2,2)

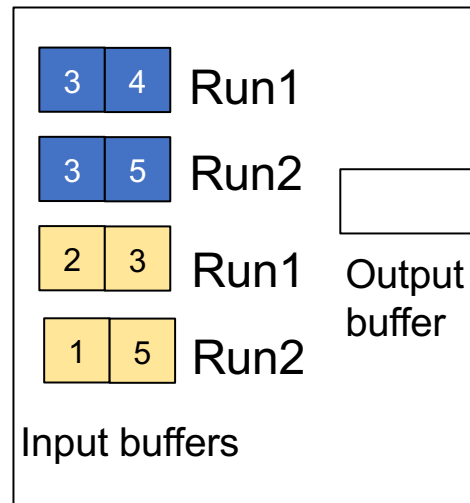
Merge-Join Example

Step 2: Join while merging sorted runs



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

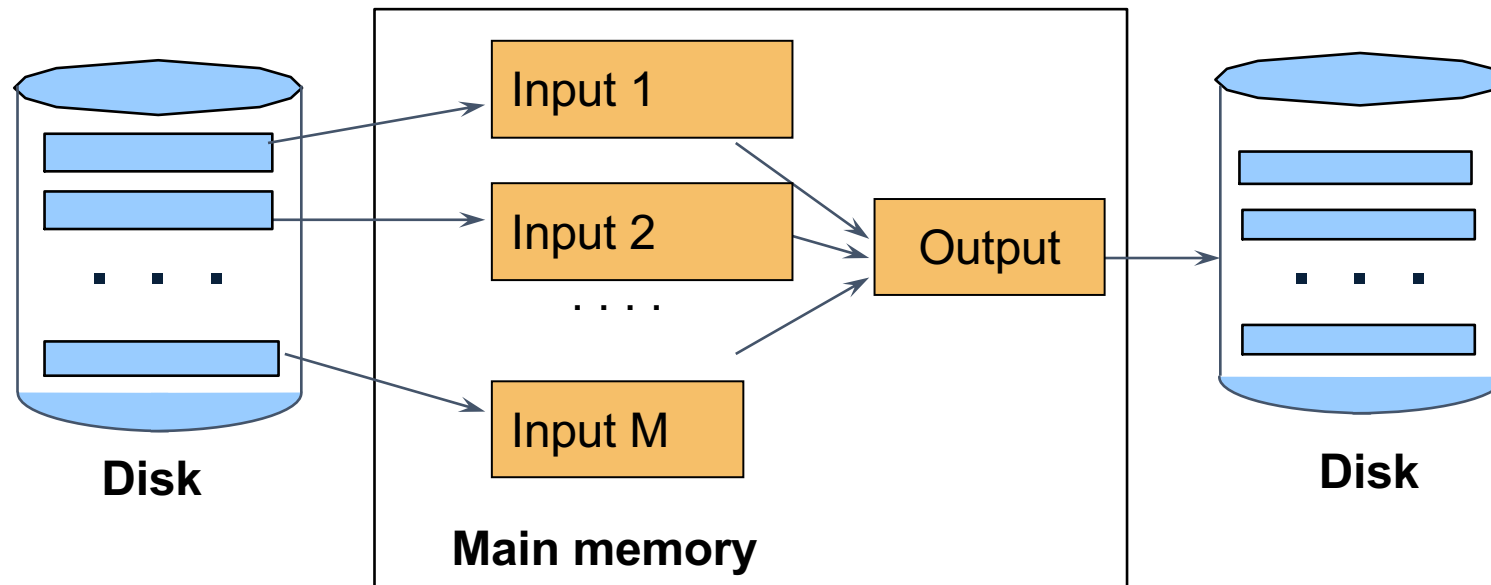
Memory $M = 5$ pages



Step 2: Join while merging
Output tuples

- (1,1)
- (1,1)
- (1,1)
- (1,1)
- (2,2)
- (2,2)
- (3,3)
- (3,3)
- ...

Merge-Join



$M_1 = B(R)/M$ runs for R
 $M_2 = B(S)/M$ runs for S
Merge-join $M_1 + M_2$ runs;
need $M_1 + M_2 \leq M$ to process all runs
i.e. $B(R) + B(S) \leq M^2$

$B(R)/M \leq M$