Database System Internals

External Memory Algorithms (part 3)

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

Announcements

- Lab 2 (part 1) due next Wednesday, 4/25

- HW2 due following Monday, 4/28

Index Based Selection

- **Example:**

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

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:
  - 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

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 ⋈ S

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

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 of nested loop join
  - B(R) + T(R)*B(S)
- Cost of Index Nested Loop Join:
  - If index on S is clustered:
  - If index on S is unclustered:

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 of nested loop join
  - B(R) + T(R)*B(S)
- Cost of Index Nested Loop Join:
  - If index on S is clustered:
  - If index on S is unclustered: B(R) + T(R)T(S)/V(S,a)

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 of nested loop join
  - B(R) + T(R)*B(S)

- Cost of Index Nested Loop Join:
  - 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

- Hash-join, merge-join assumed data <= memory

- Next: algorithm when the data >> main memory
  Called _external memory_ algorithm

- Merge-join
- Partitioned hash-join

Questions

- **What is the "best" algorithm for sorting an array of n elements in main memory?**

- What is its runtime?

- What is the best algorithm for sorting a large file of n items on disc?

- What is its runtime?

Questions

- **What is the "best" algorithm for sorting an array of n elements in main memory?**
  - Quicksort
- **What is its runtime?**

- What is the best algorithm for sorting a large file of n items on disc?

- What is its runtime?

Questions

- What is the "best" algorithm for sorting an array of n elements in main memory?
  - Quicksort
- What is its runtime?
  - $O(n \log n)$

- What is the best algorithm for sorting a large file of n items on disc?

- What is its runtime?

Questions

- What is the "best" algorithm for sorting an array of n elements in main memory?
  - Quicksort

- What is its runtime?
  - $O(n \log n)$

- What is the best algorithm for sorting a large file of n items on disc?
  - Multi-way Merge sort

- What is its runtime?
  - $O(n \log n)$ CPU time; $O(B \log_M B)$ disk I/O's

Questions

- What is the "best" algorithm for sorting an array of n elements in main memory?
  - Quicksort

- What is its runtime?
  - $O(n \log n)$

- What is the best algorithm for sorting a large file of n items on disc?
  - Multi-way Merge sort

  > Main memory merge-sort: 2-way
  > External memory merge-sort: multi-way

- What is its runtime?
  - $O(n \log n)$ CPU time; $O(B \log_M B)$ disk I/O's

Questions

- What is the "best" algorithm for sorting an array of n elements in main memory?
    - Quicksort

- What is its runtime?
    - $O(n \log n)$

- What is the best algorithm for sorting a large file of n items on disc?
    - Multi-way Merge sort

> Main memory merge-sort: 2-way
> External memory merge-sort: multi-way

- What is its runtime?
    - $O(n \log n)$ CPU time; $O(B \log_M B)$ disk I/O's

Merge-Join is based on the multi-way merge-sort (next)

Merge-Sort: 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

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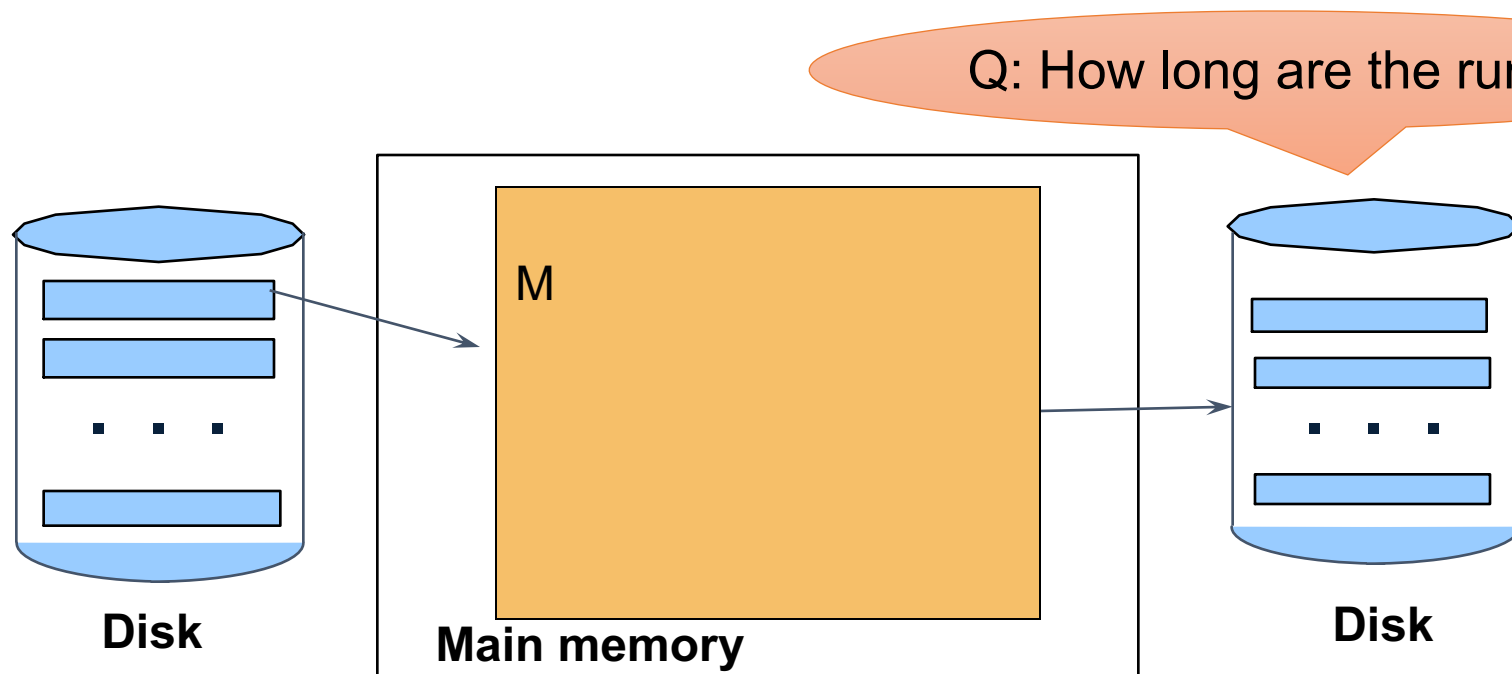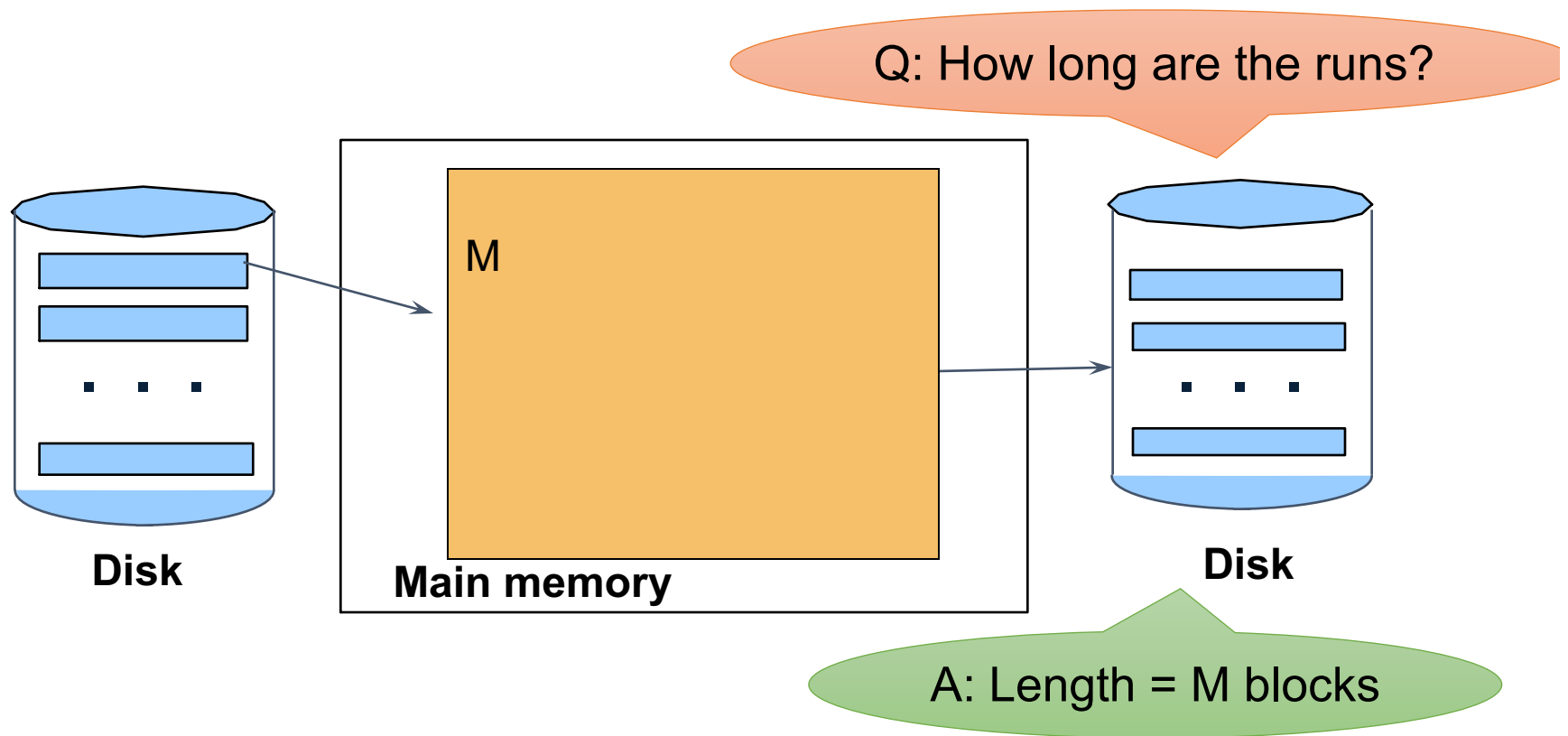
Phase one: load M blocks in memory, sort, send to disk, repeat

Q: How long are the runs?

M

Disk

Main memory

Disk

External Merge-Sort: Step 1

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

Q: How long are the runs?

M

Disk

Main memory

Disk

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) ≈ M^2
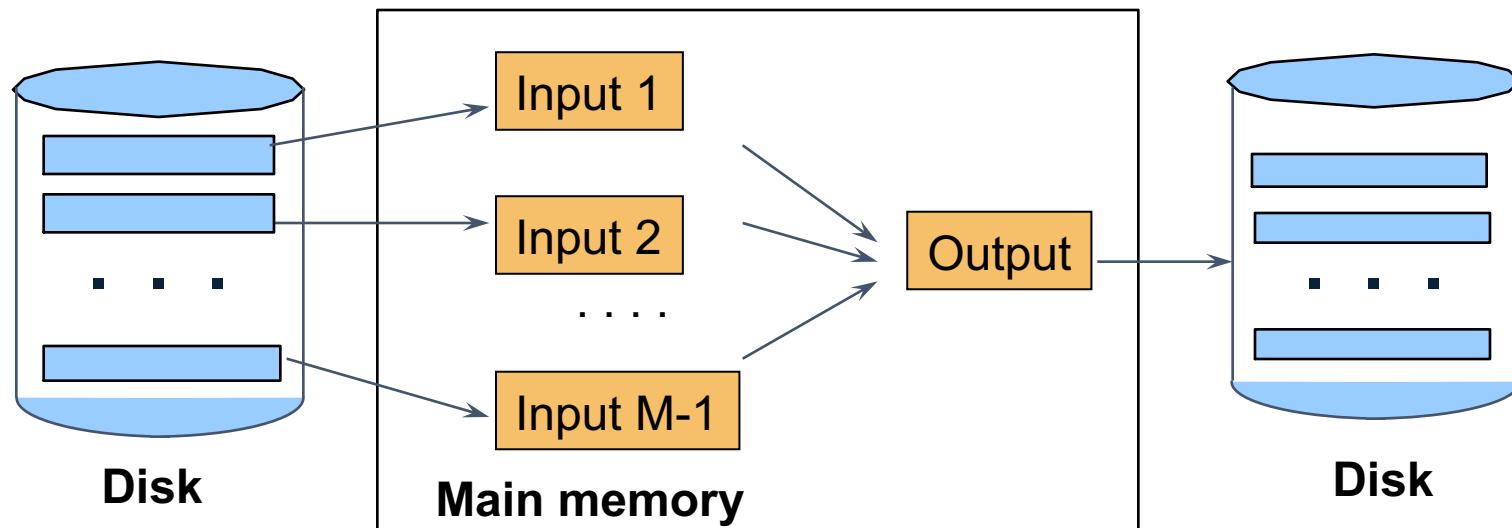
Example

- 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) ≈ M^2
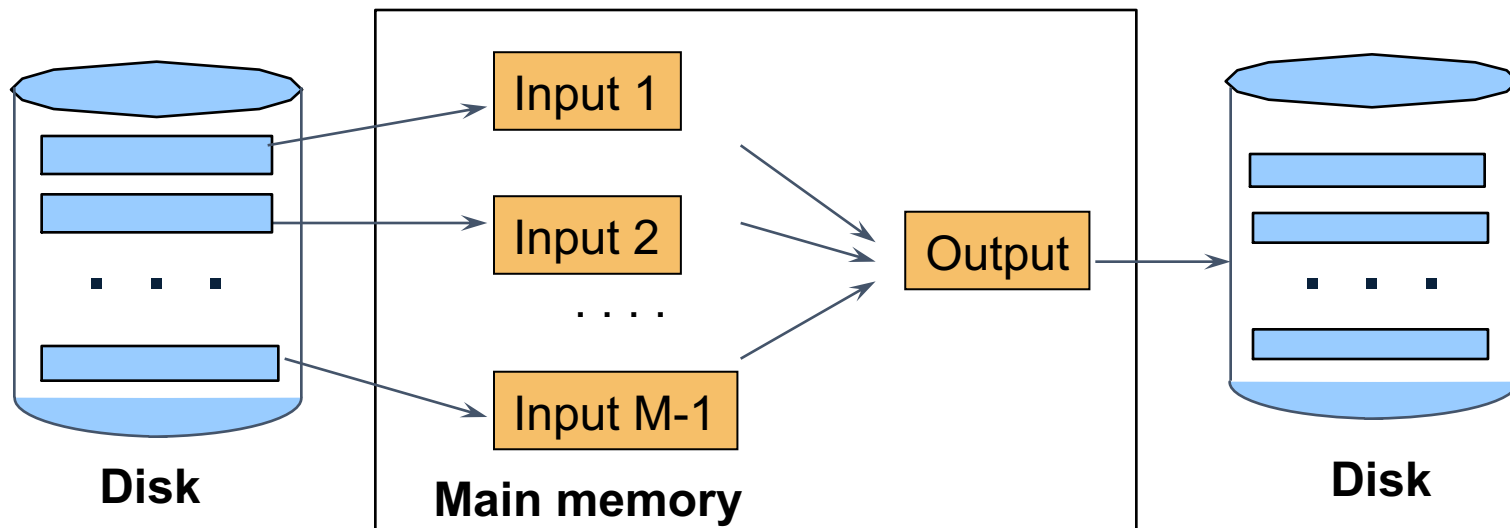
If approx. B <= M^2 then we are done

Cost of External Merge Sort

In theory:

- Number of I/O's: $O(B(R) * \log_M B(R))$

In practice:

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

Discussion

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

Discussion

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

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

Discussion

- What does $B(R) <= M^2$ 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 ⋈ S

- How?....

Merge-Join

Join R ⋈ 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
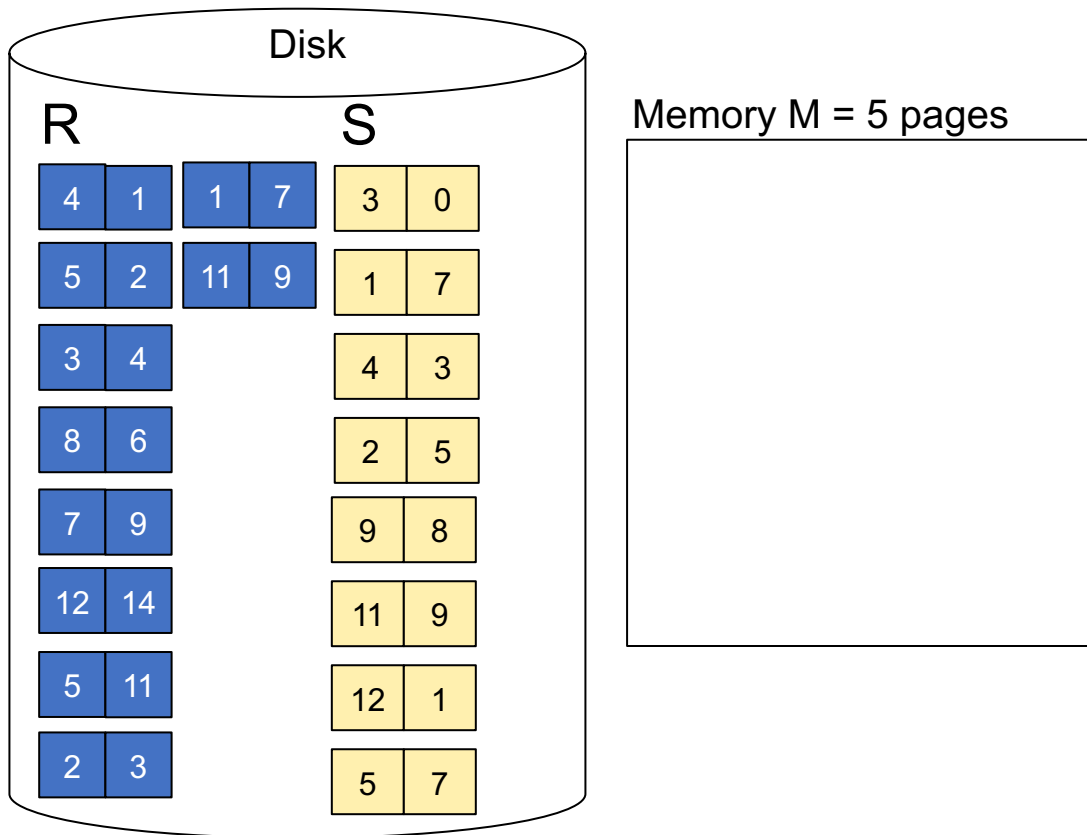
Merge-Join Example
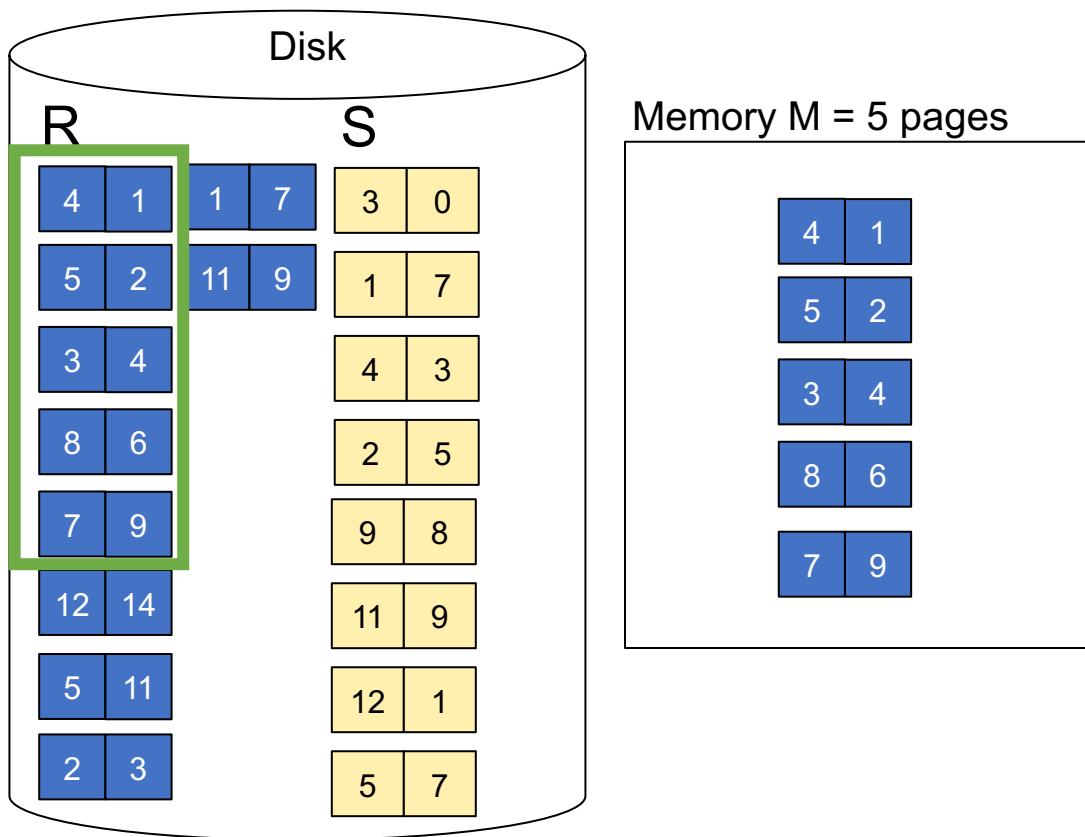
Step 1: Read M pages of R and sort in memory

Merge-Join Example
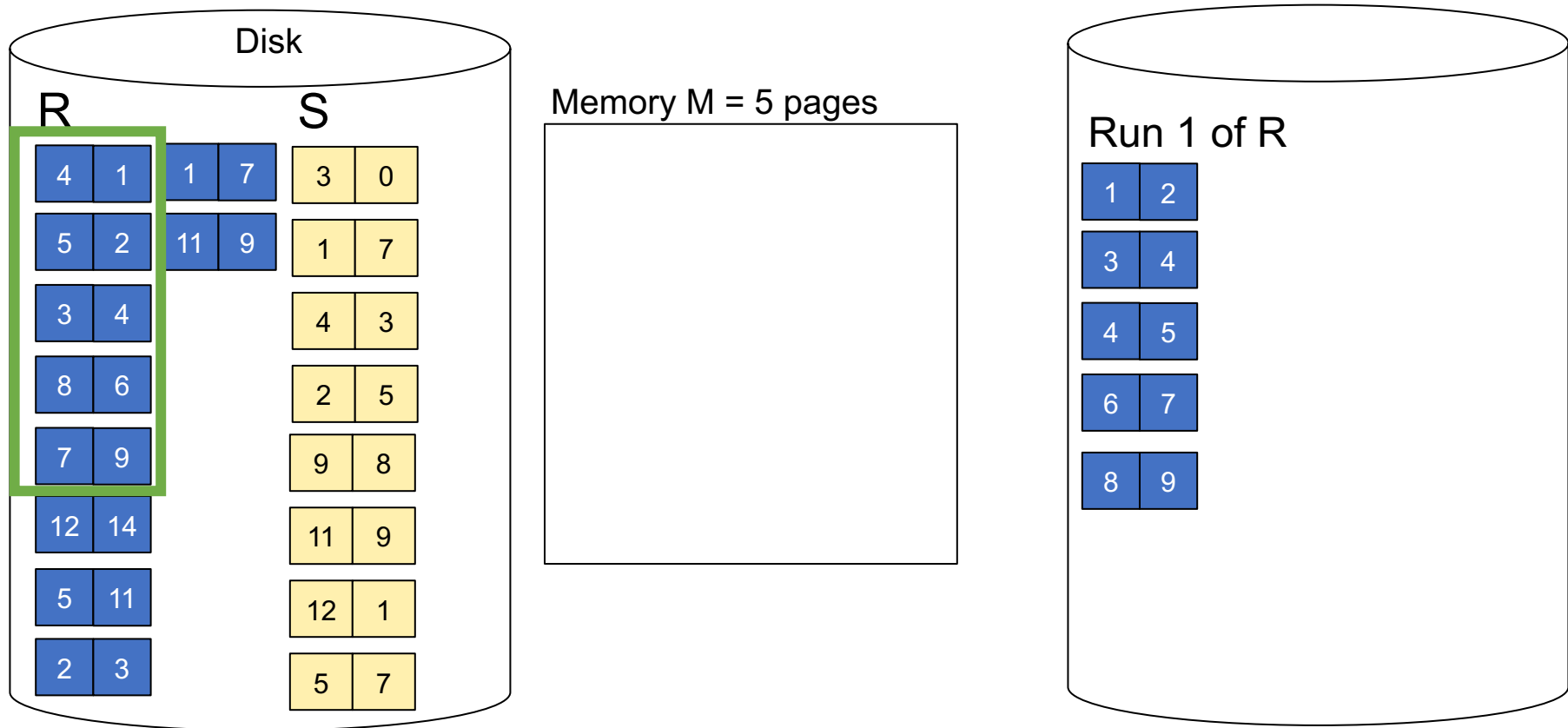
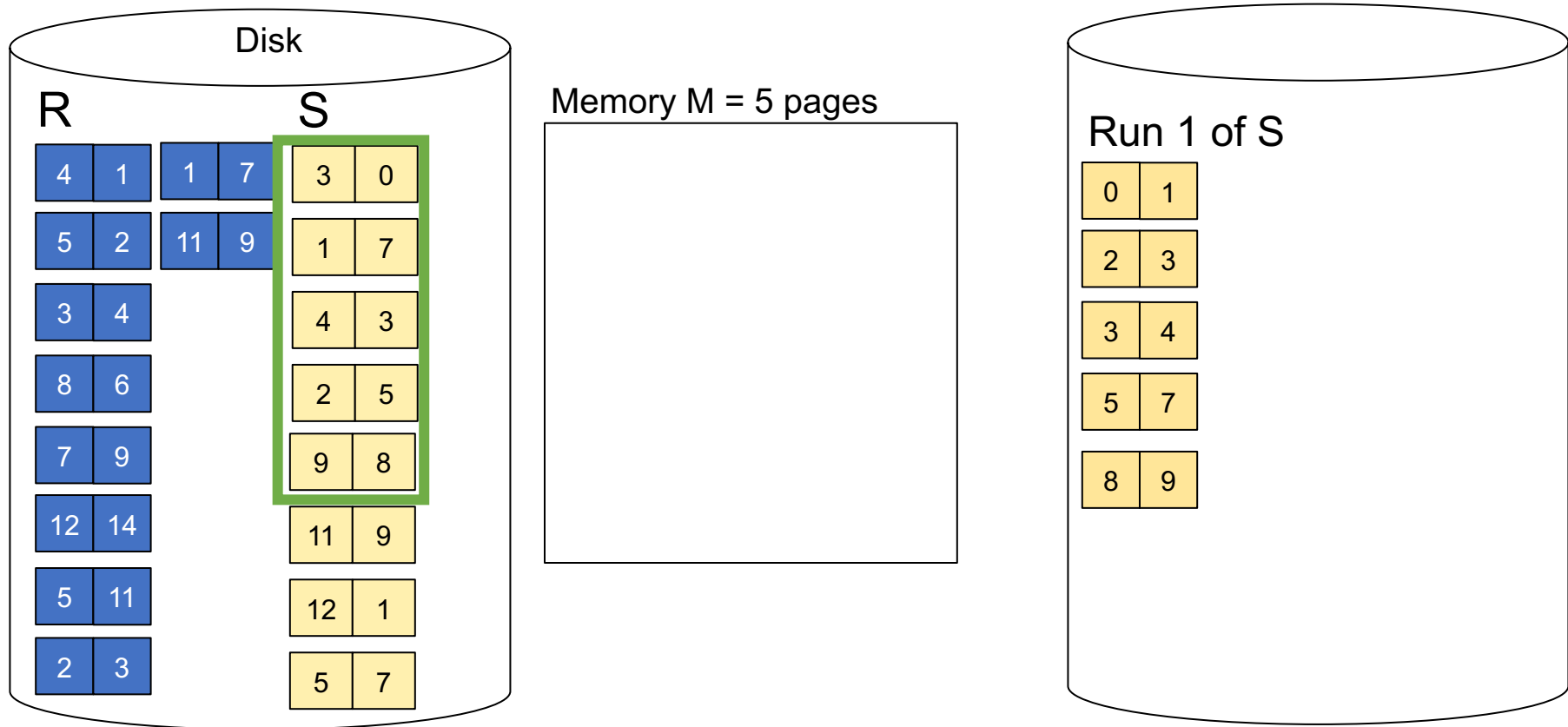**Step 1:** Read M pages of R and sort in memory, then write to disk

Merge-Join Example

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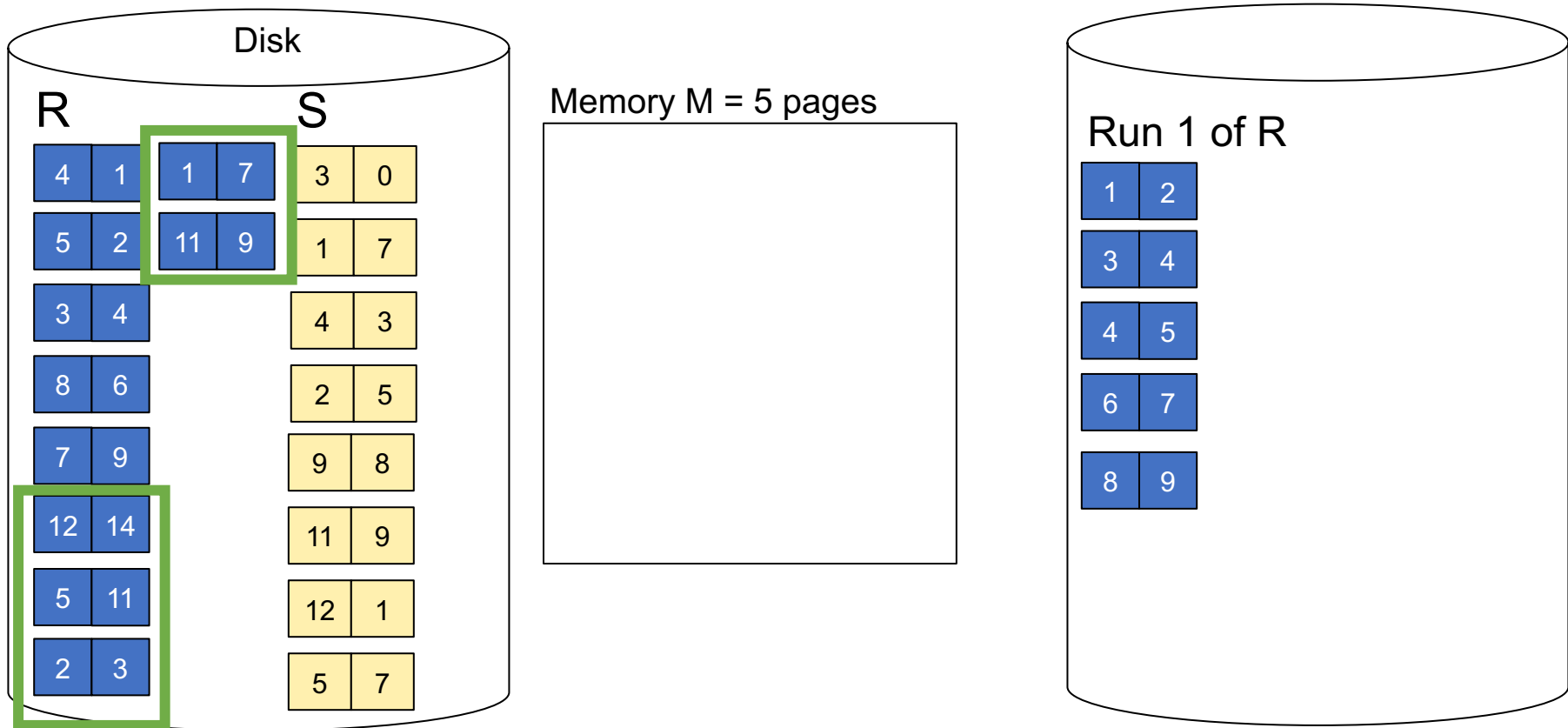
Merge-Join Example

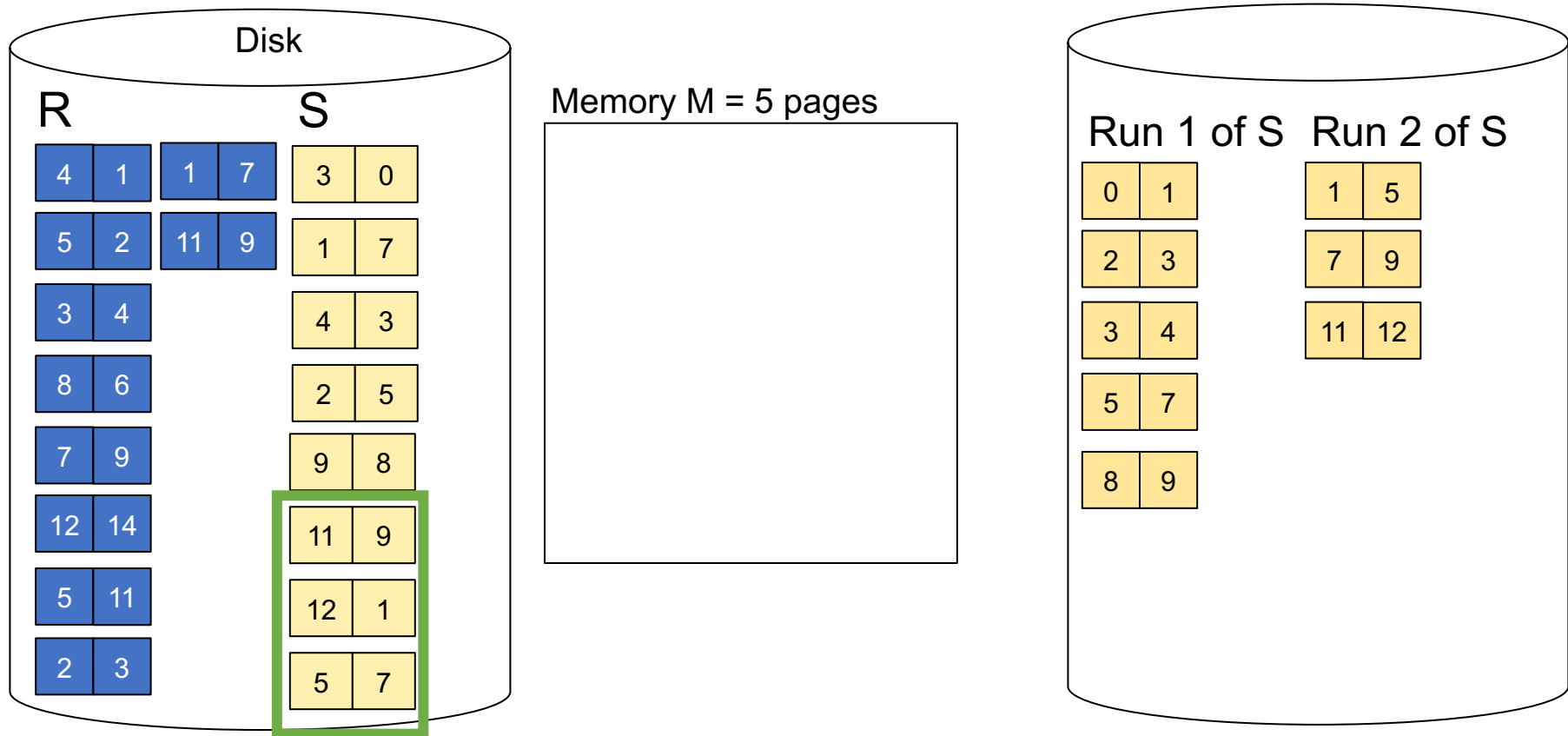**Step 1:** Repeat for next M pages until all R is processed

Merge-Join Example

Step 1: Do the same with S
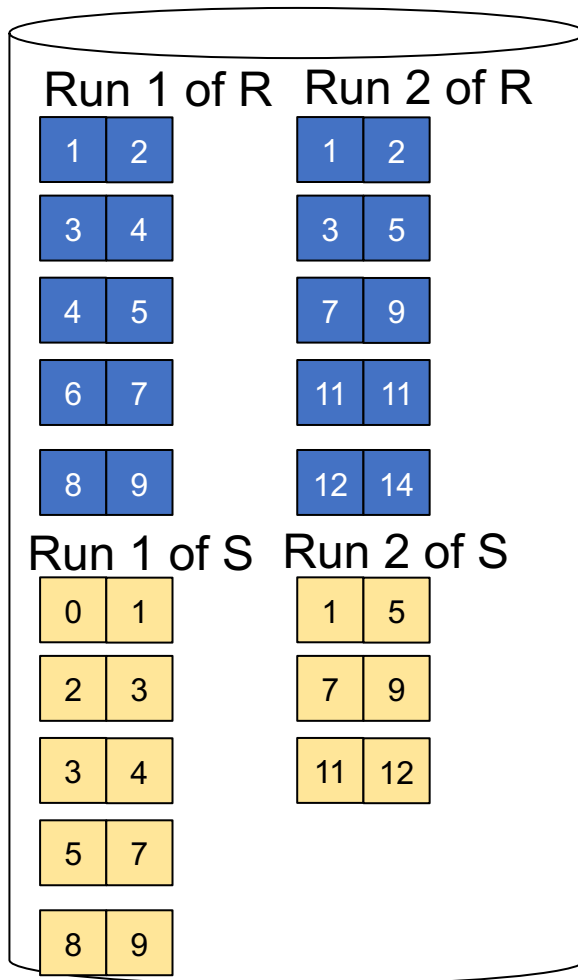
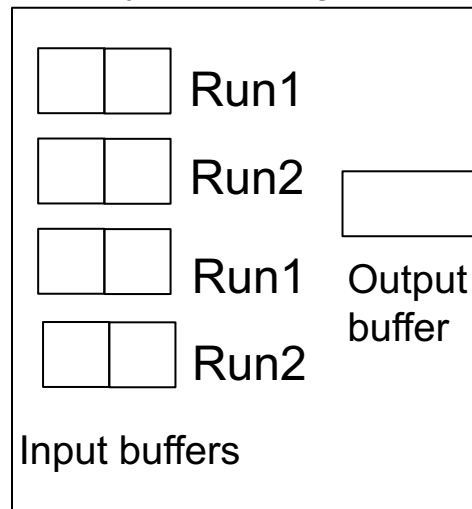# Merge-Join Example

Step 1: Do the same with S

Merge-Join Example

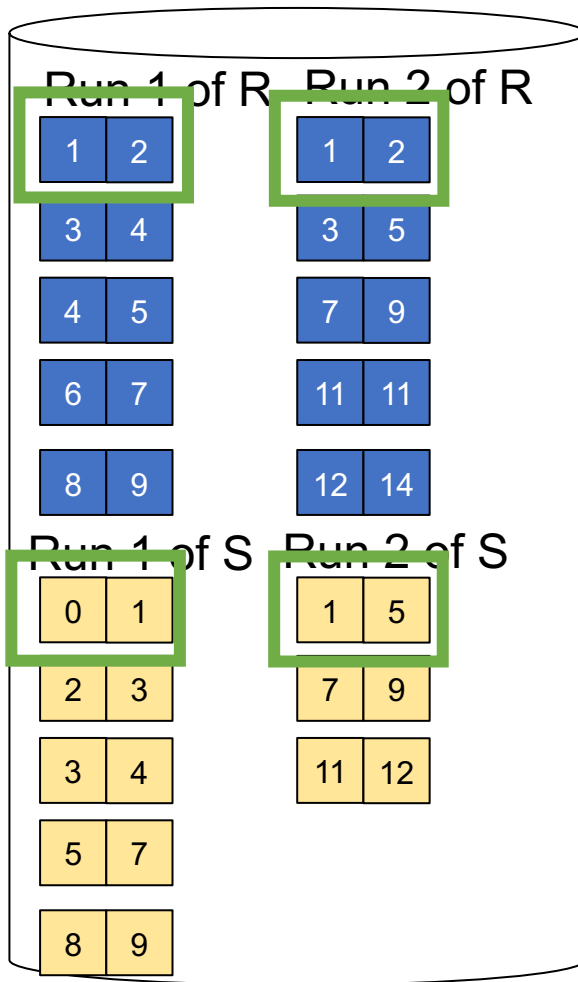**Step 2:** Join while merging sorted runs

Total cost: 3B(R) + 3B(S)
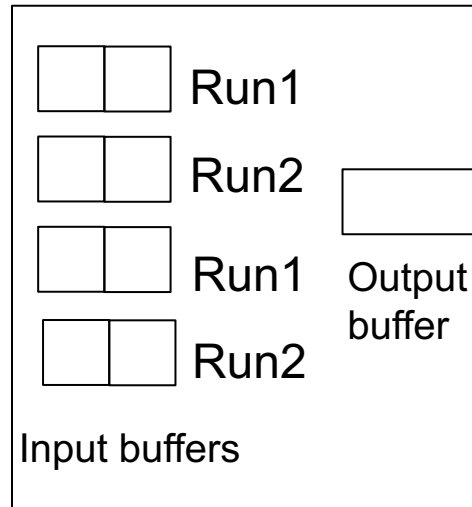
Step 2: Join while merging
Output tuples

Run 1 of R   Run 2 of R

1	2
3	4
4	5
6	7
8	9

1	2
3	5
7	9
11	11
12	14

Run 1 of S   Run 2 of S

0	1
2	3
3	4
5	7
8	9

1	5
7	9
11	12

Memory M = 5 pages

Run1
Run2
Run1
Run2

Output buffer

Input buffers

Merge-Join Example

Step 2: Join while merging sorted runs

Total cost: 3B(R) + 3B(S)
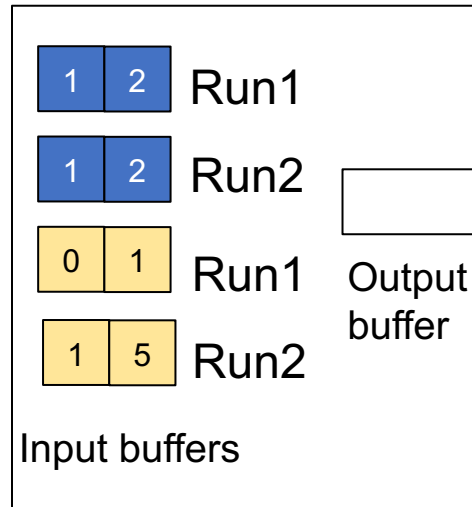
Step 2: Join while merging
Output tuples

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7
8	9

Memory M = 5 pages

Run1
Run2
Run1   Output
          buffer
Run2

Input buffers

Merge-Join Example

Step 2: Join while merging sorted runs

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7
8	9

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

Step 2: Join while merging
Output tuples

Memory M = 5 pages

| 1 | 2 | Run1
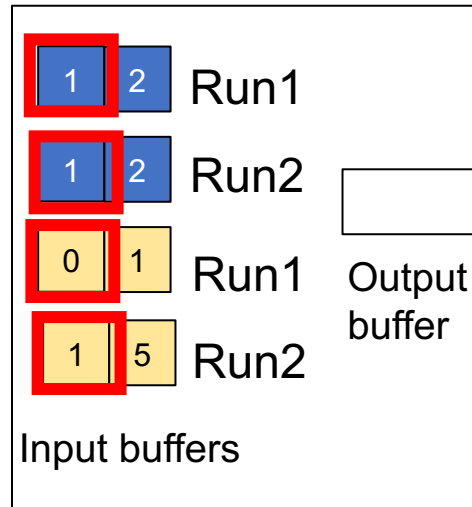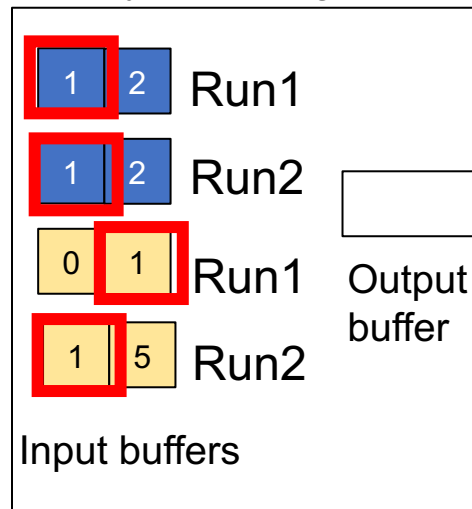| 1 | 2 | Run2   Output buffer
| 0 | 1 | Run1
| 1 | 5 | Run2

Input buffers

Merge-Join Example

Step 2: Join while merging sorted runs

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

Step 2: Join while merging
Output tuples

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7
8	9

Memory M = 5 pages

| 1 | 2 | Run1
| 1 | 2 | Run2
| 0 | 1 | Run1
| 1 | 5 | Run2

Output buffer

Input buffers

Merge-Join Example

Step 2: Join while merging sorted runs

Total cost: 3B(R) + 3B(S)
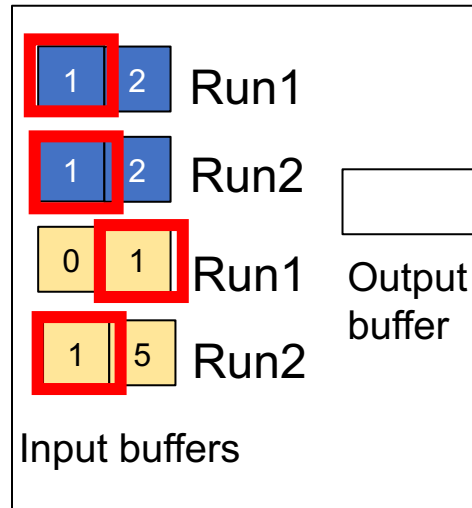
Step 2: Join while merging
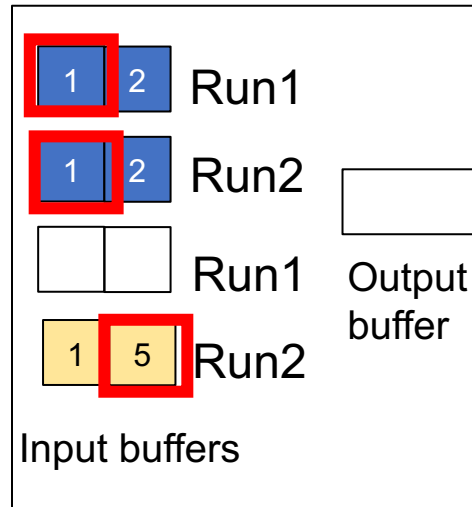Output tuples

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7			
8	9			

Memory M = 5 pages

| 1 | 2 | Run1
| 1 | 2 | Run2
| 0 | 1 | Run1   Output buffer
| 1 | 5 | Run2

Input buffers

Merge-Join Example

Step 2: Join while merging sorted runs

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

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7
8	9

Memory M = 5 pages

1	2	Run1
1	2	Run2
0	1	Run1
1	5	Run2

Output buffer

Input buffers

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)

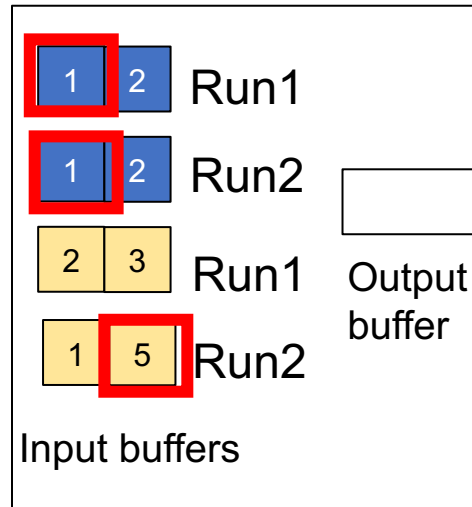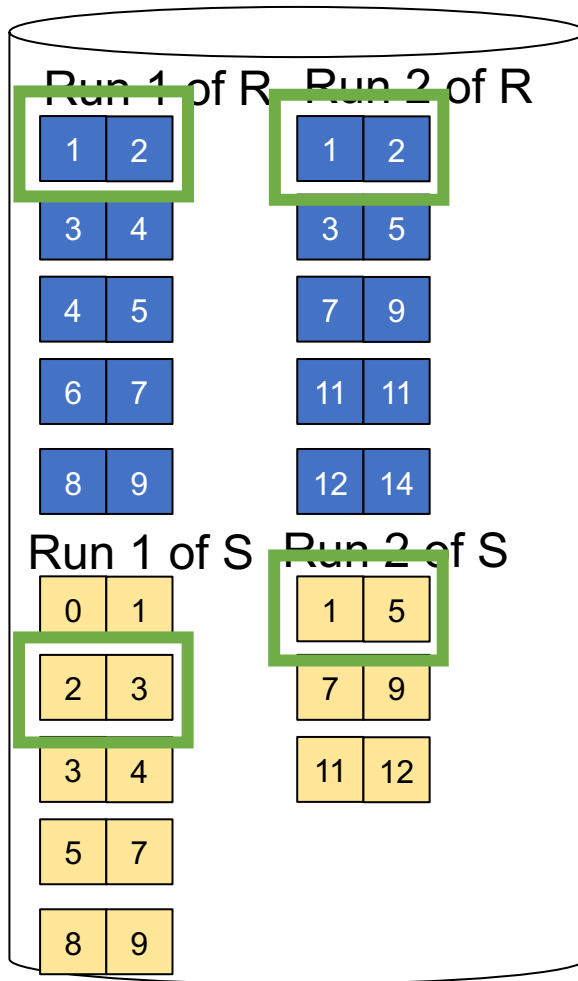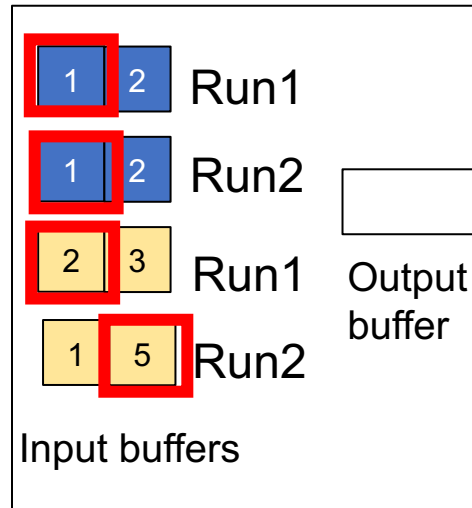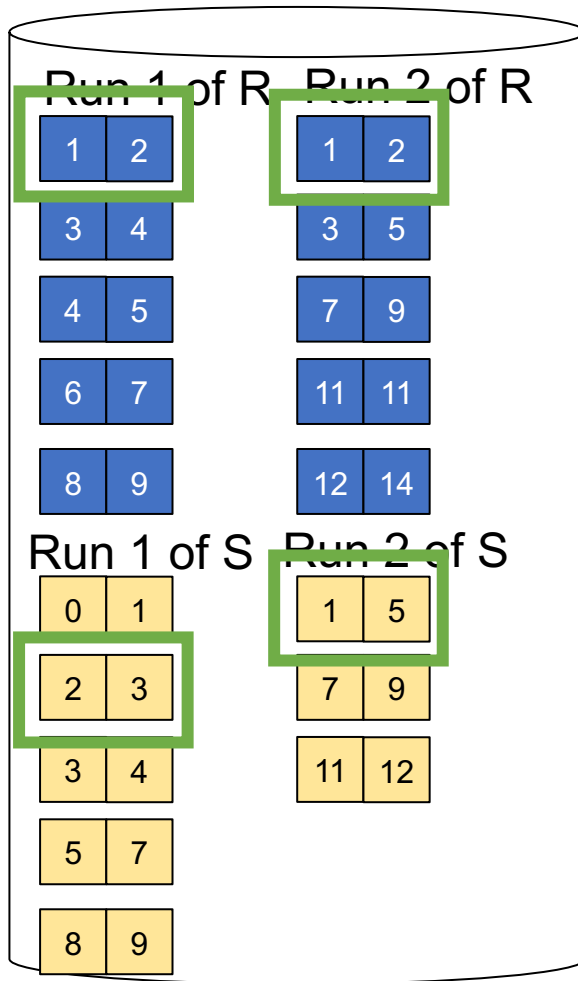**Step 2:** Join while merging
Output tuples
(1,1)
(1,1)
(1,1)
(1,1)

Merge-Join Example

Step 2: Join while merging sorted runs

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

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7
8	9

Memory M = 5 pages

| 1 | 2 | Run1
| 1 | 2 | Run2
| 2 | 3 | Run1
| 1 | 5 | Run2

Output buffer

Input buffers

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)

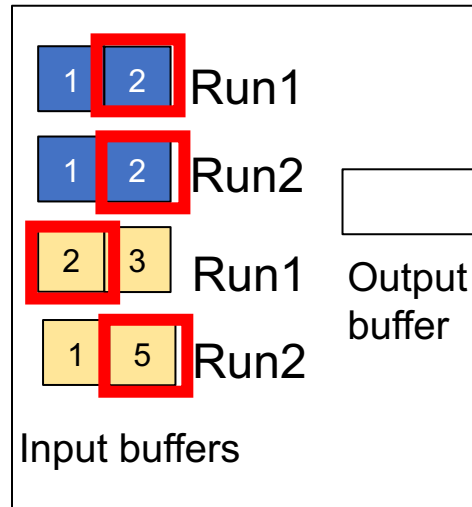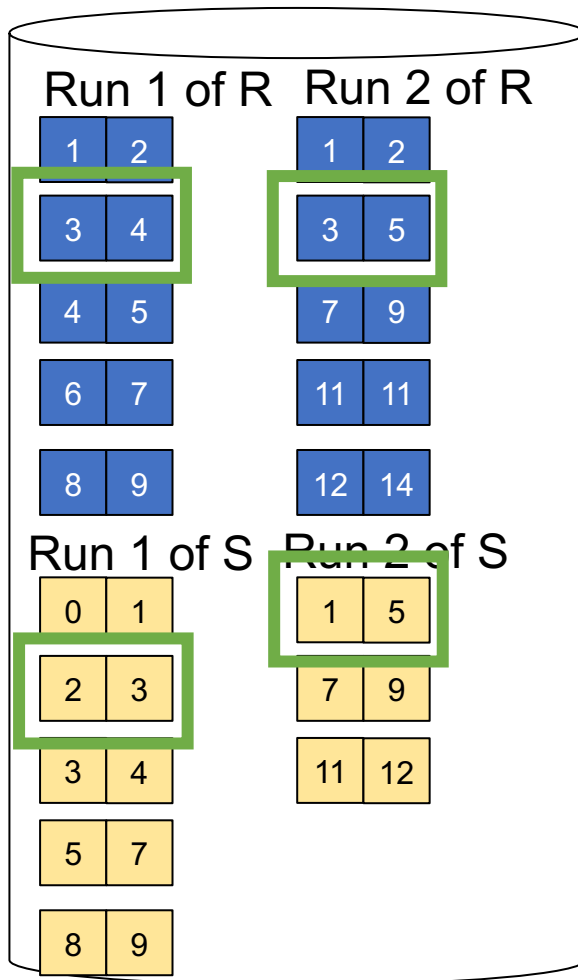**Step 2:** Join while merging
Output tuples
(1,1)
(1,1)
(1,1)
(1,1)

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7
8	9

Memory M = 5 pages

1	2	Run1
1	2	Run2
2	3	Run1
1	5	Run2

Output buffer

Input buffers

Merge-Join Example

Step 2: Join while merging sorted runs

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

Step 2: Join while merging
Output tuples
(1,1)
(1,1)
(1,1)
(1,1)
(2,2)
(2,2)

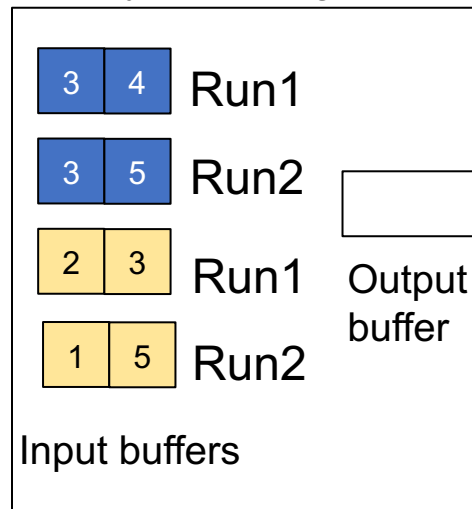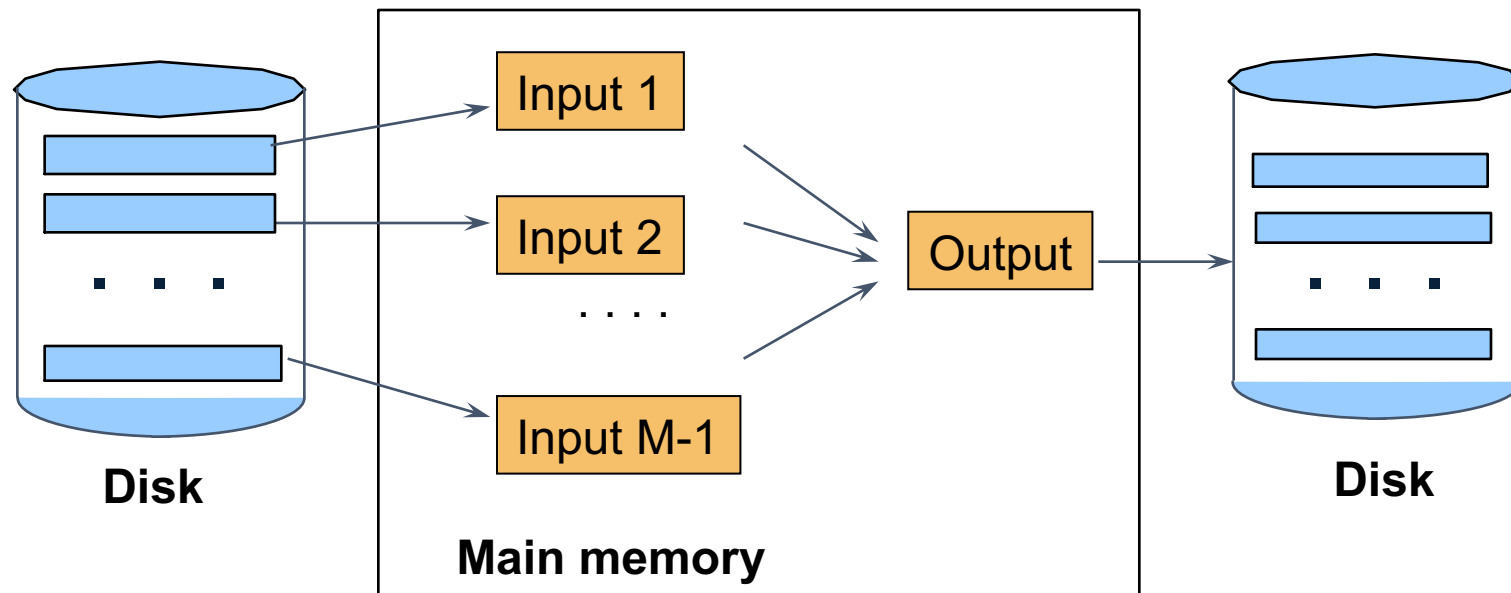# Merge-Join Example

Step 2: Join while merging sorted runs

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

Run 1 of R   Run 2 of R

1	2		1	2
3	4		3	5
4	5		7	9
6	7		11	11
8	9		12	14

Run 1 of S   Run 2 of S

0	1		1	5
2	3		7	9
3	4		11	12
5	7
8	9

Memory M = 5 pages

| 3 | 4 | Run1
| 3 | 5 | Run2   Output buffer
| 2 | 3 | Run1   Output buffer
| 1 | 5 | Run2

Input buffers

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 <= M to process all runs
    i.e. B(R) + B(S) <= M^2

Summary of External Join Algorithms

- Block Nested Loop: $B(S) + B(R)*B(S)/(M-1)$

- Index Join:
  - Clustered: $B(R) + T(R)B(S)/V(S,a)$
  - Unclustered: $B(R) + T(R)T(S)/V(S,a)$

- Merge Join: $3B(R)+3B(S)$
  - $B(R)+B(S) <= M^2$

- Partitioned Hash Join: (coming up next)

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
    $R_1, R_2, R_3, \ldots, R_k$

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
  $R_1, R_2, R_3, \ldots, R_k$

- Assuming $B(R_1) = B(R_2) = \ldots = B(R_k)$, we have
  $B(R_i) = B(R)/k$, for all i

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
$$R_1, R_2, R_3, \ldots, R_k$$

- Assuming $B(R_1)=B(R_2)=\ldots= B(R_k)$, we have
$$B(R_i) = B(R)/k, \quad \text{for all } i$$

- Goal: each R_i should fit in main memory:
$$B(R_i) \leq M$$

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
  $R_1, R_2, R_3, \ldots, R_k$

- Assuming $B(R_1)=B(R_2)=\ldots= B(R_k)$, we have
  $B(R_i) = B(R)/k$,   for all i

- Goal: each R_i should fit in main memory:
  $B(R_i) \leq M$

  How do we choose k?

Partitioned Hash Algorithms

- We choose k = M-1 Each bucket has size approx. $B(R)/(M-1) \approx B(R)/M$
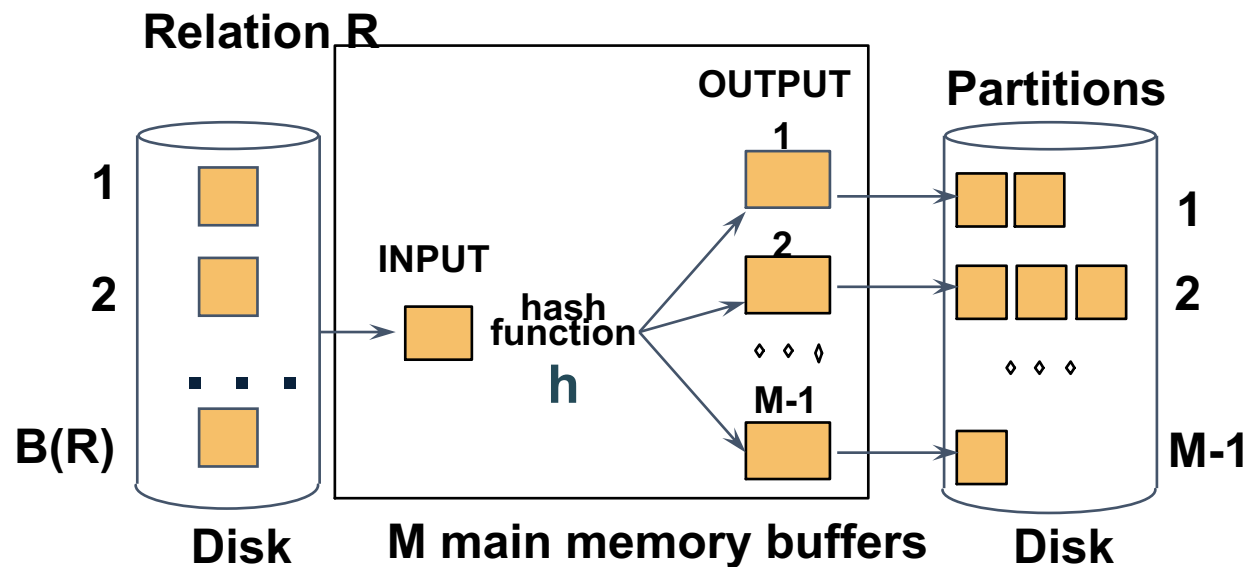
Assumption: $B(R)/M \leq M$, i.e. $B(R) \leq M^2$

Partitioned Hash Join (Grace-Join)

R ⋈ 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

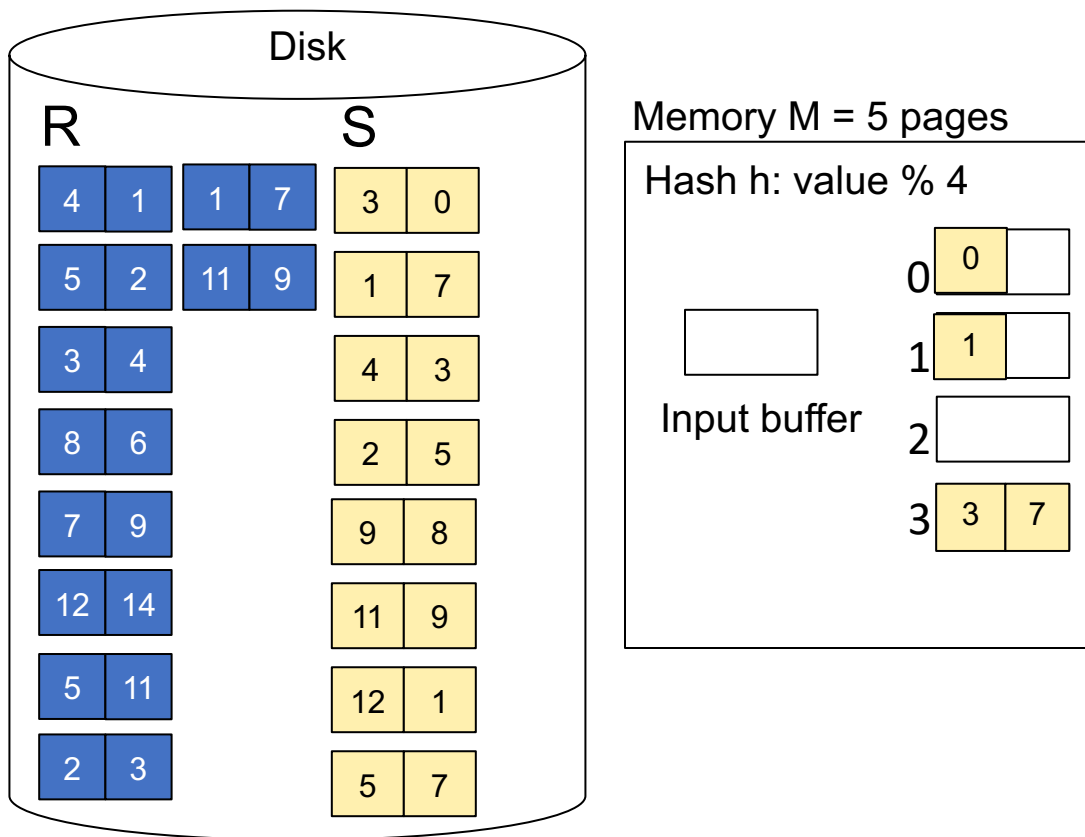Note: partitioned hash-join is sometimes called *grace-join*

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into M-1 (=4 buckets)

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets

Partitioned Hash-Join Example

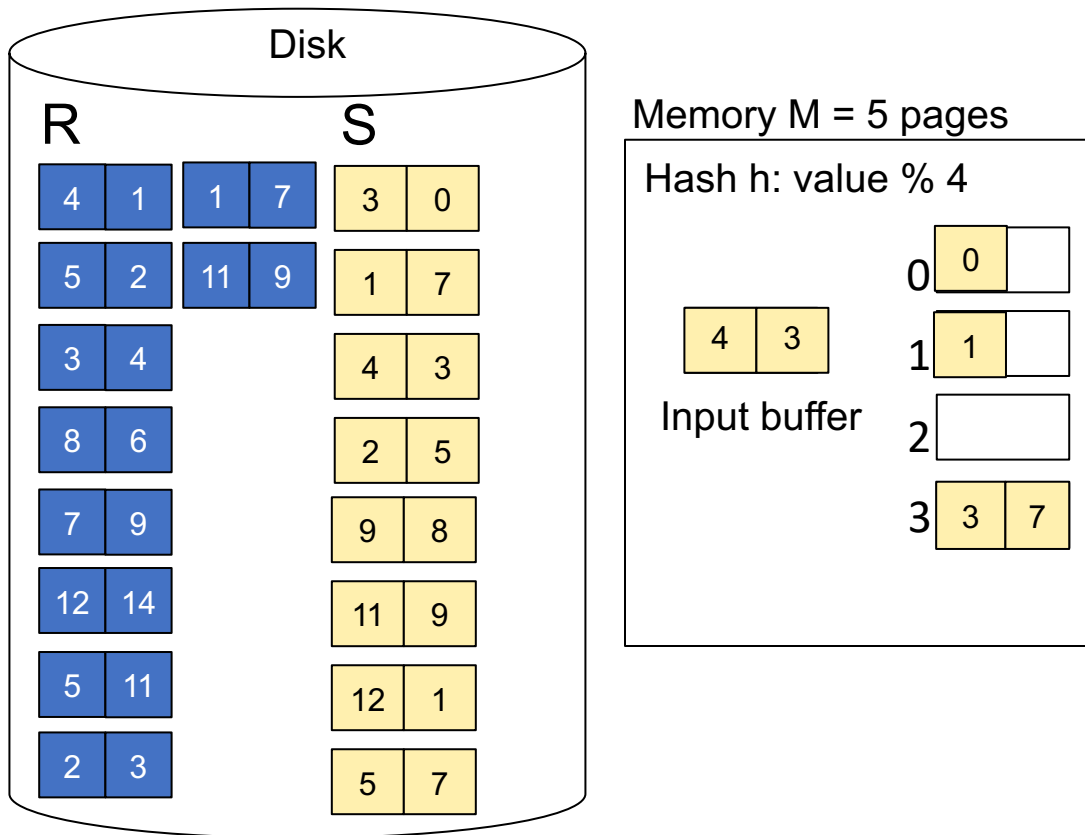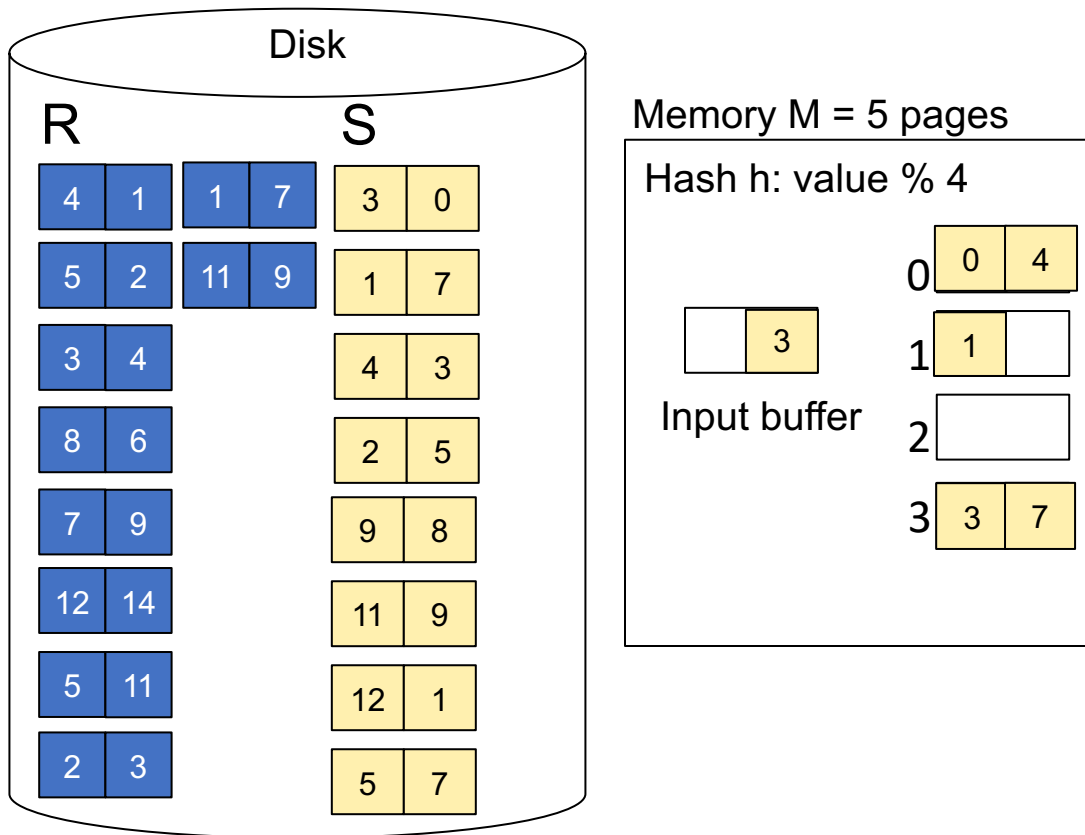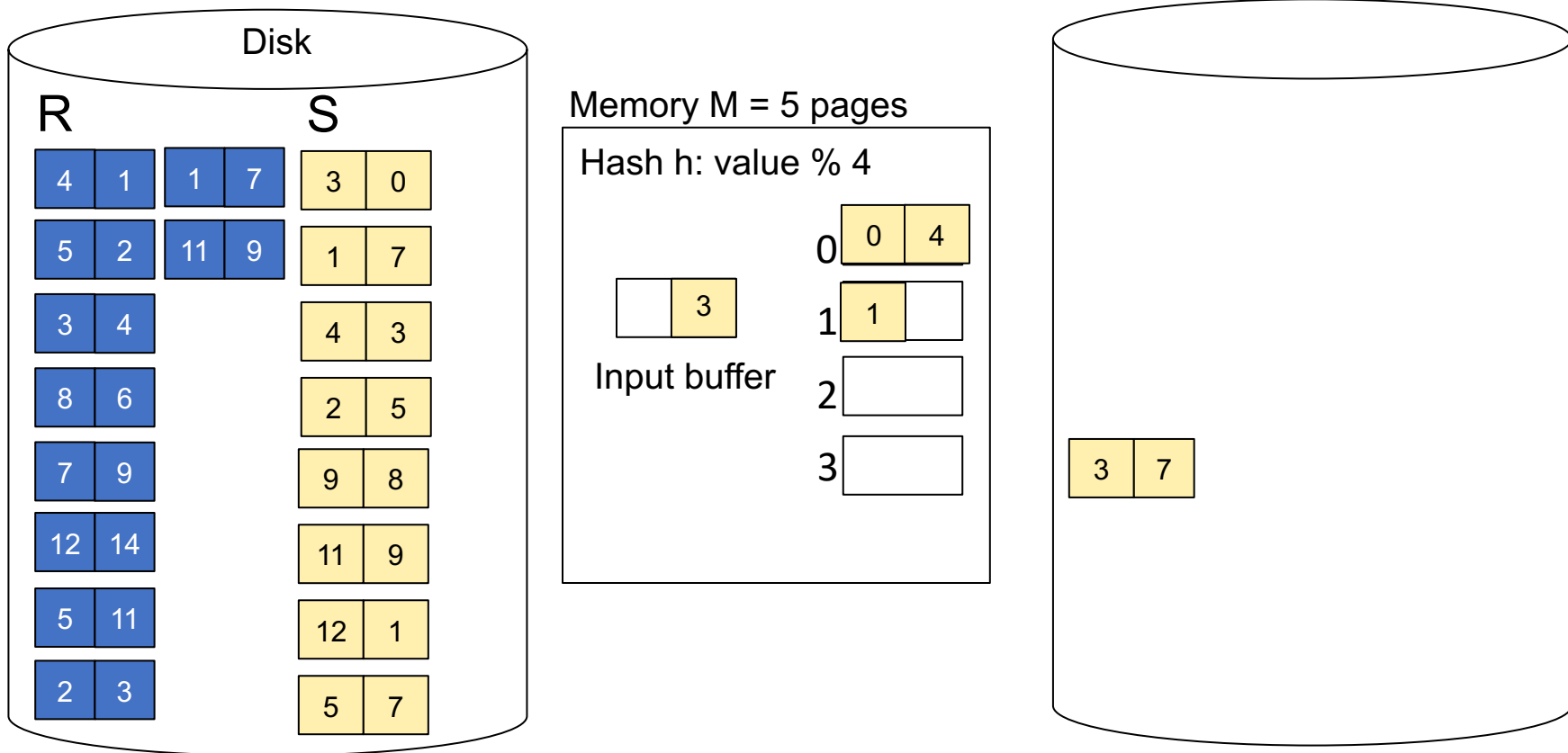**Step 1:** Read relation S one page at a time and hash into the 4 buckets

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets

Disk

R   S

4	1
5	2
3	4
8	6
7	9
12	14
5	11
2	3

| 1 | 7 |
| 11 | 9 |

3	0
1	7
4	3
2	5
9	8
11	9
12	1
5	7

Memory M = 5 pages

Hash h: value % 4

Input buffer

0 | 0 |
1 | 1 |
2 |
3 | 3 | 7 |

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
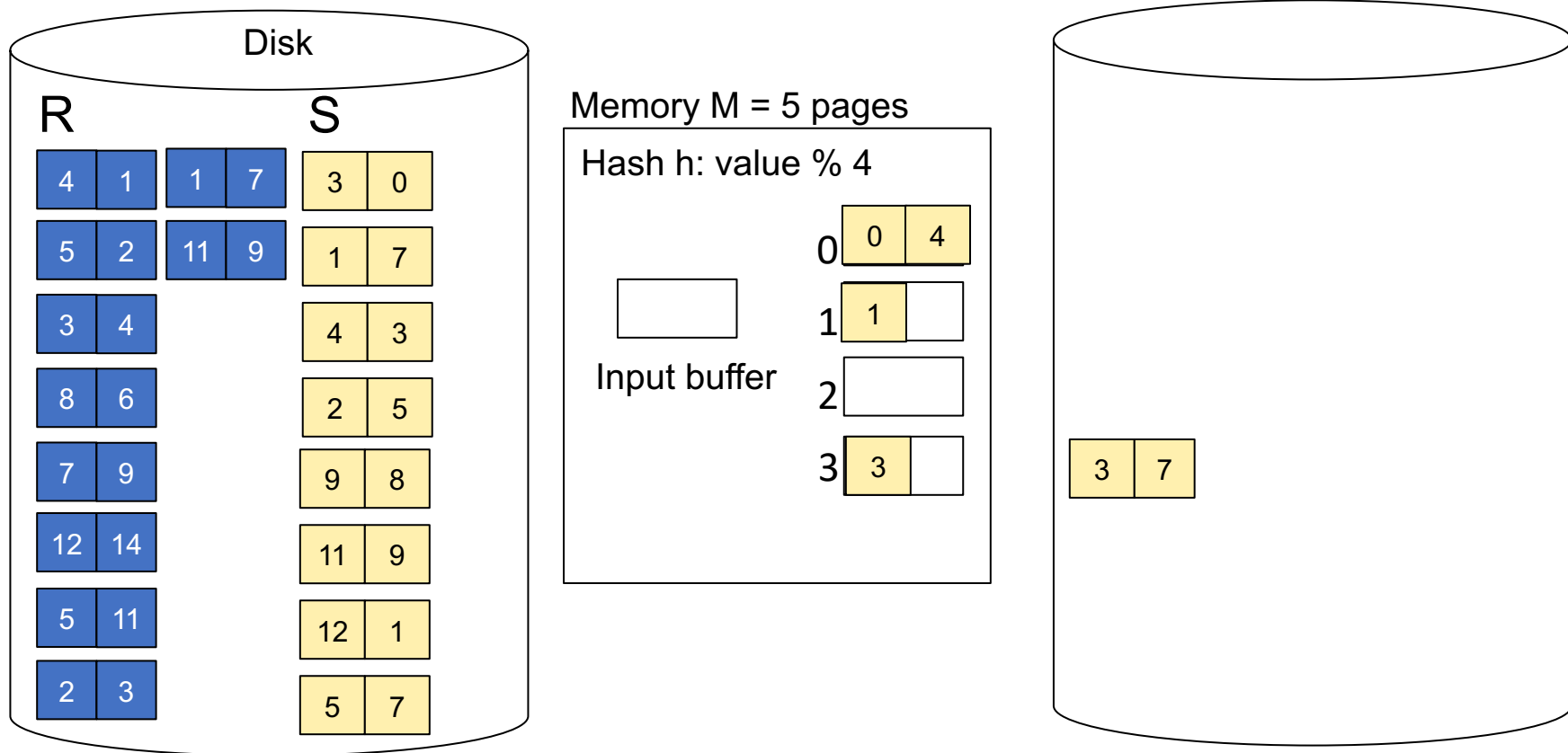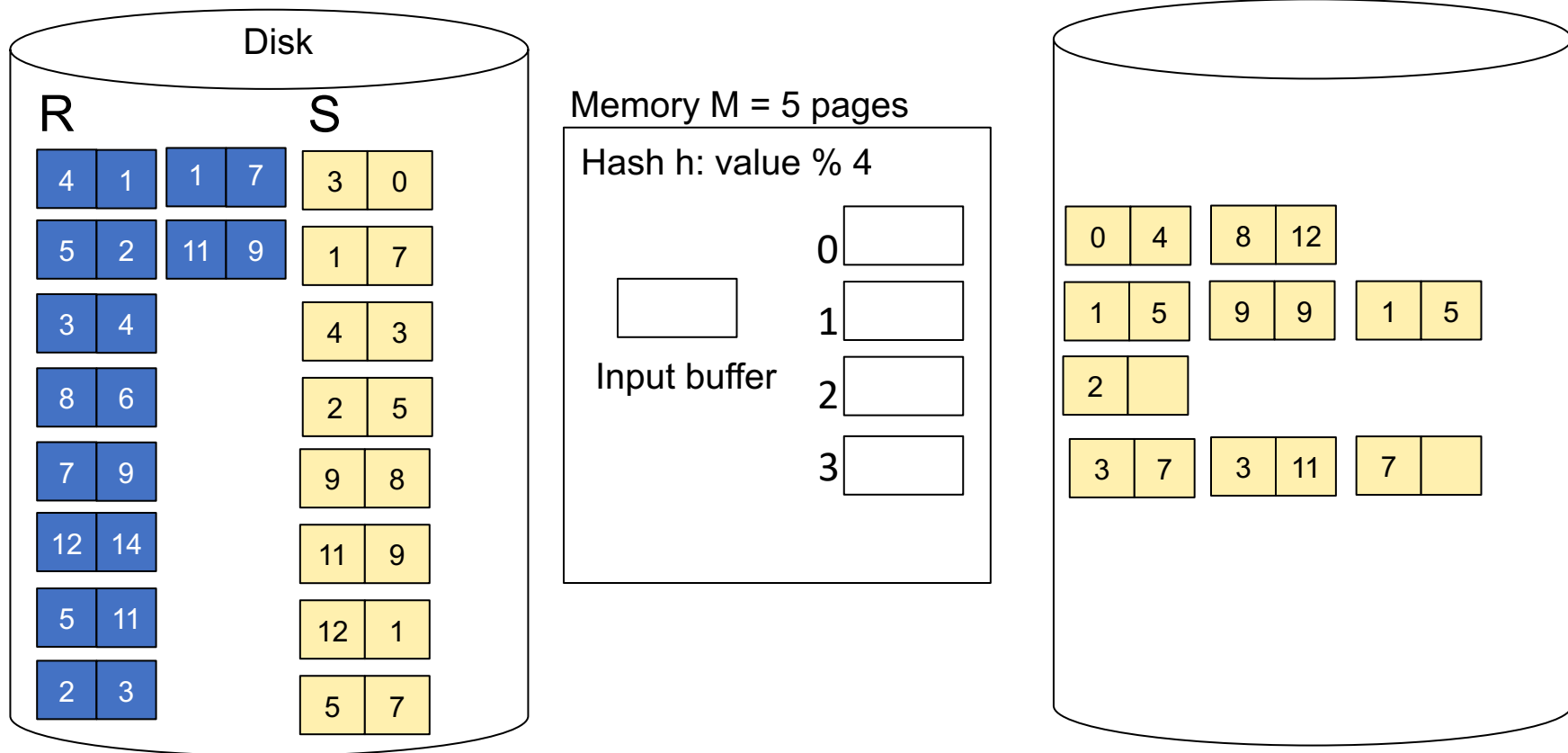When a bucket fills up, flush it to disk

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
When a bucket fills up, flush it to disk

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
When a bucket fills up, flush it to disk

Disk

R   S

4	1		1	7		3	0
5	2		11	9		1	7
3	4		4	3
8	6		2	5
7	9		9	8
12	14		11	9
5	11		12	1
2	3		5	7

Memory M = 5 pages

Hash h: value % 4

Input buffer

0 | 0 | 4 |
1 | 1 | |
2 | | |
3 | 3 | |

| 3 | 7 |

Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
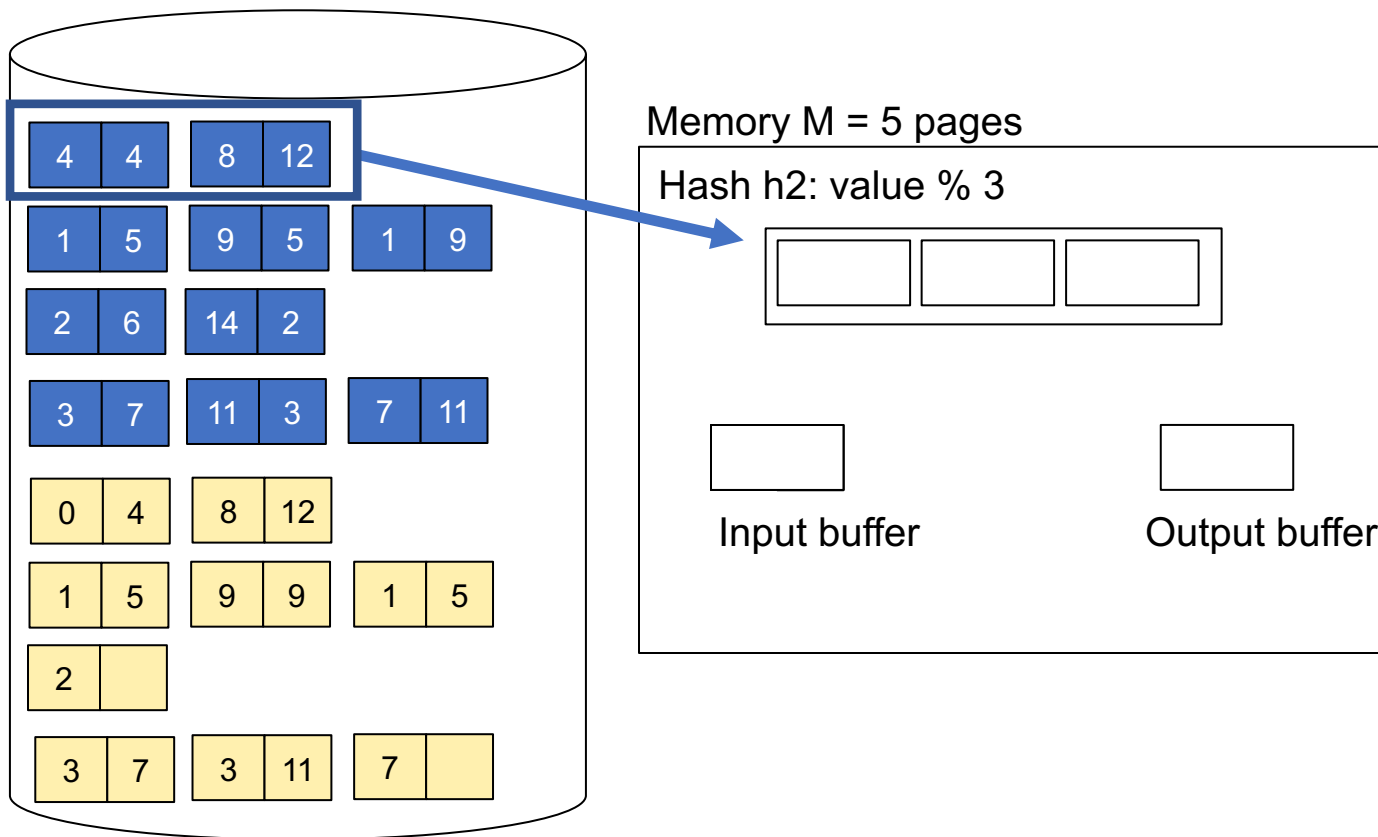At the end, we get relation S back on disk split into 4 buckets

Disk

R   S

4	1		1	7		3	0
5	2		11	9		1	7
3	4					4	3
8	6					2	5
7	9					9	8
12	14					11	9
5	11					12	1
2	3					5	7

Memory M = 5 pages

Hash h: value % 4

Input buffer

0
1
2
3

0	4		8	12
1	5		9	9		1	5
2	
3	7		3	11		7	

Partitioned Hash-Join Example

Step 2: Read relation R one page at a time and hash into same 4 buckets
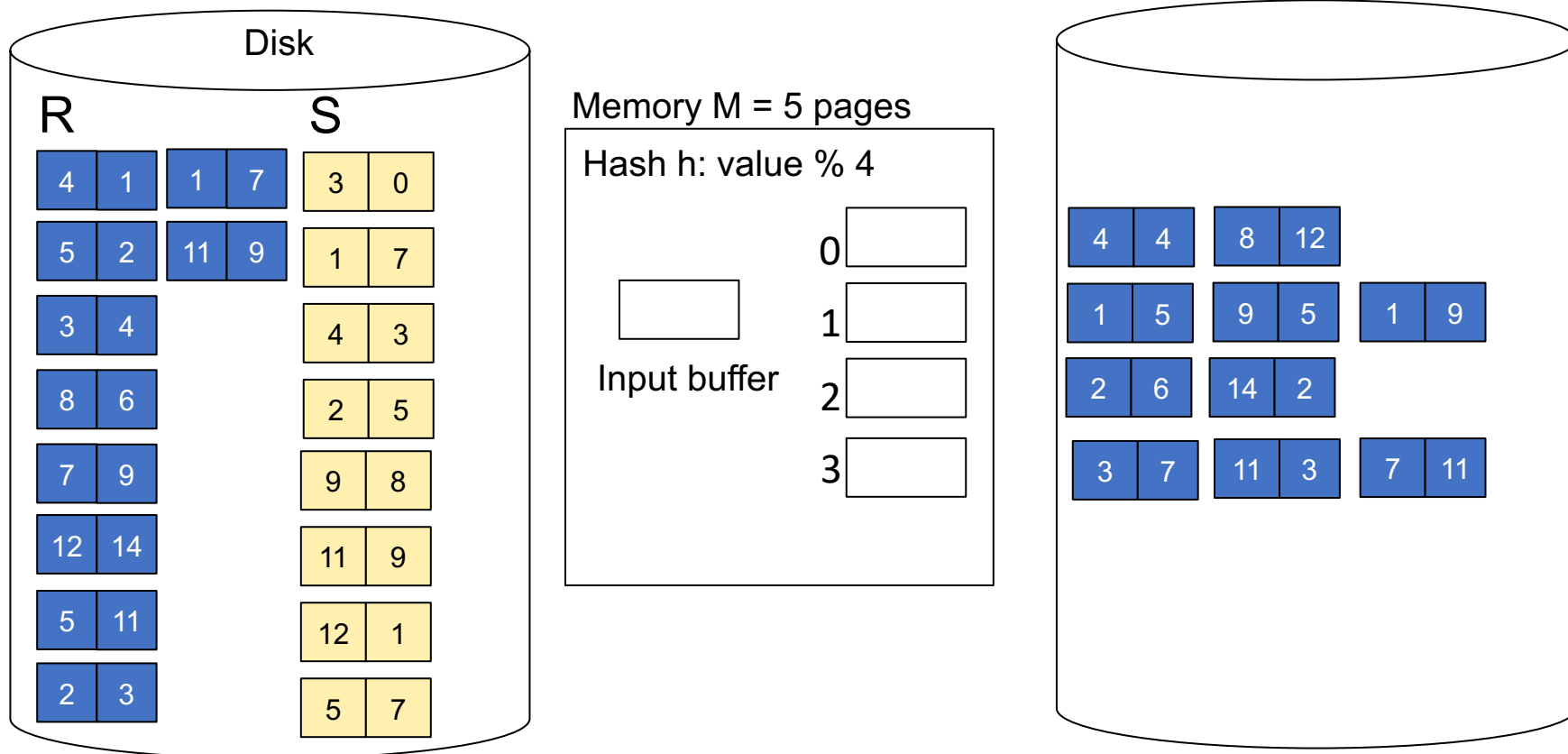
Partitioned Hash-Join Example

Step 3: Read one partition of R and create hash table in memory using a ***different*** hash function
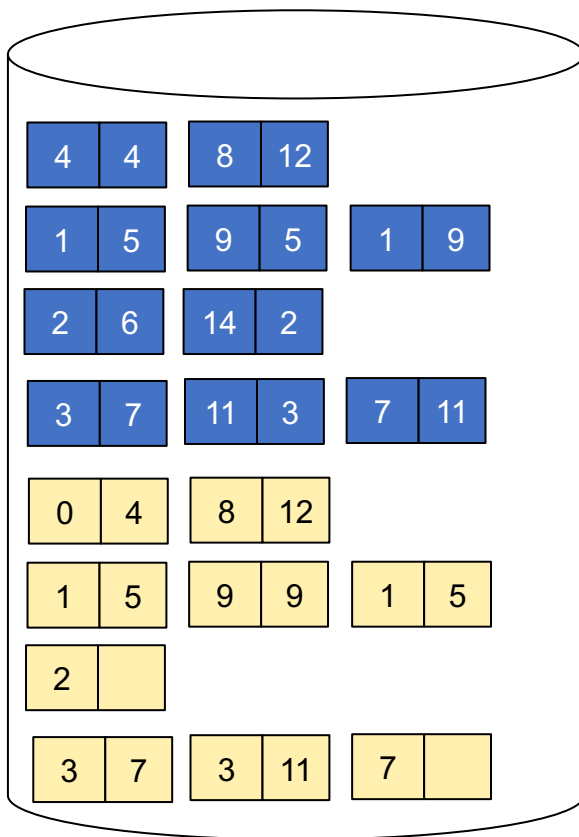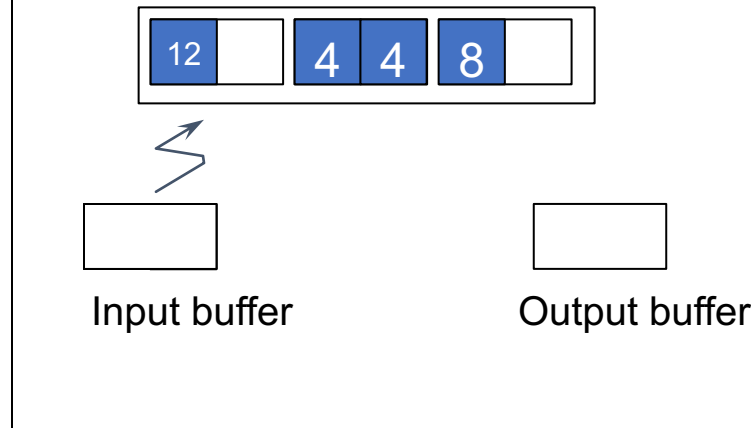
Partitioned Hash-Join Example

Step 3: Read one partition of R and create hash table in memory using a *different* hash function
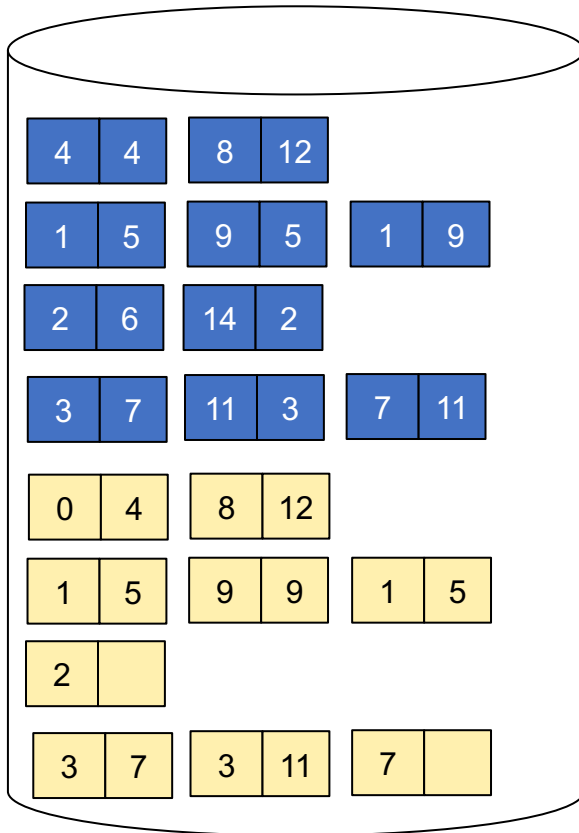
Memory M = 5 pages

Hash h2: value % 3

Input buffer

Output buffer

Partitioned Hash-Join Example

Step 3: Read one partition of R and create hash table in memory using a *different* hash function

Memory M = 5 pages

Hash h2: value % 3

Input buffer

Output buffer
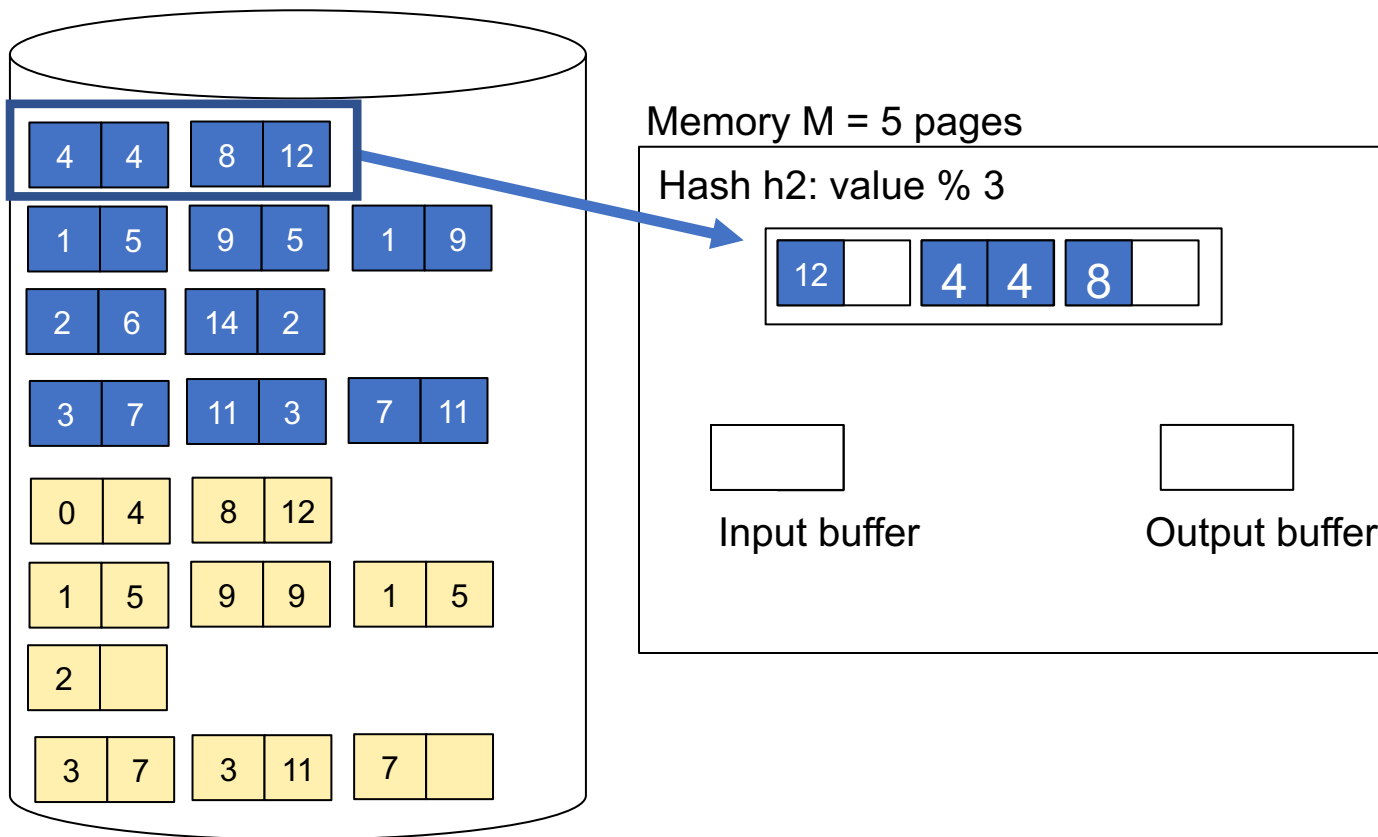
Partitioned Hash-Join Example

Step 4: Scan matching partition of S and probe the hash table
Step 5: Repeat for all the buckets
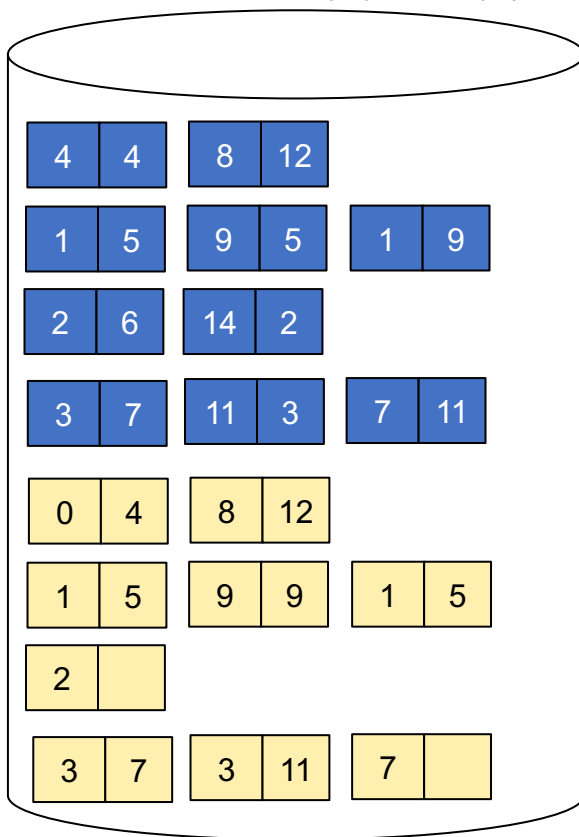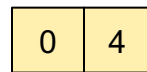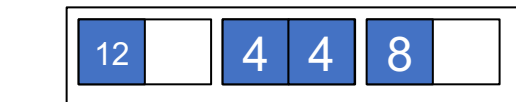**Total cost**: 3B(R) + 3B(S)

Memory M = 5 pages

Hash h2: value % 3

Input buffer   Output buffer

Partitioned Hash-Join Example

Step 4: Scan matching partition of S and probe the hash table
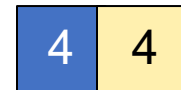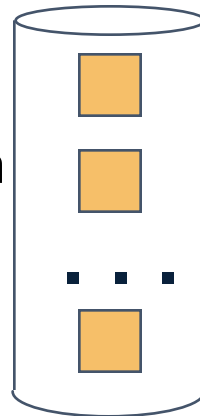**Step 5**: Repeat for all the buckets
Total cost: 3B(R) + 3B(S)

Partitioned Hash-Join

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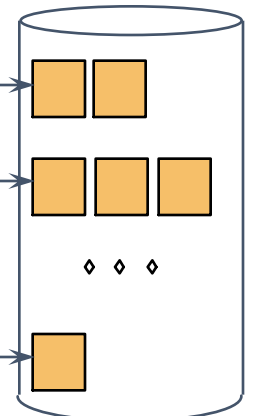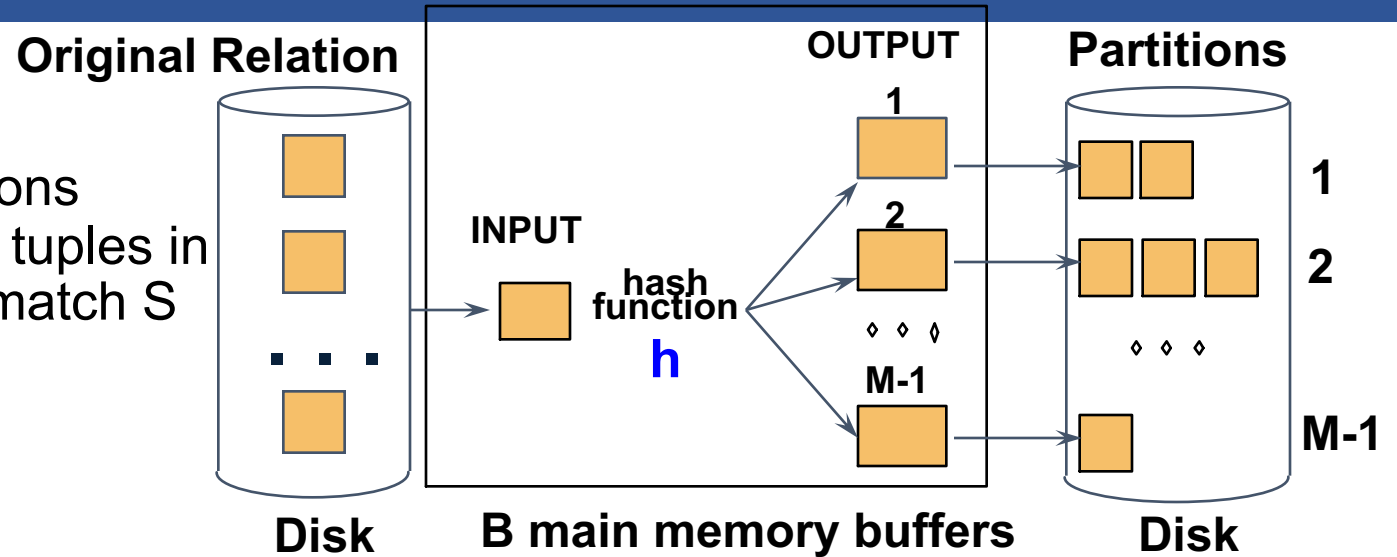
Original Relation

Disk

INPUT

hash function
h

OUTPUT
1
2
M-1

B main memory buffers

Partitions
1
2
M-1

Disk

Partitioned Hash-Join

Original Relation

OUTPUT

Partitions

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

INPUT

hash function
h

1

2

⋄ ⋄ ⋄

M-1

1

2

⋄ ⋄ ⋄

M-1

Disk

B main memory buffers

Disk

Partitions of R & S

Join Result

Hash table for partition S_i (< M-1 pages)

- ❖ Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for matches.

hash fn h2

⋄ ⋄ ⋄

h2

⋄ ⋄ ⋄

Input buffer for Ri

Output buffer

Disk

B main memory buffers
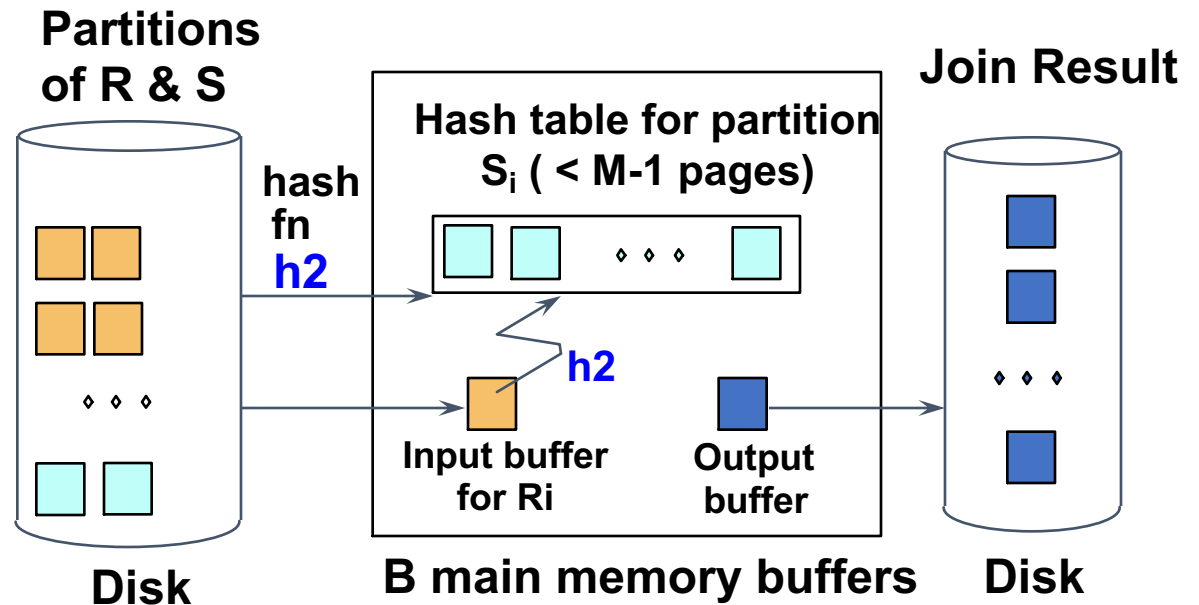
Disk

Partitioned Hash-Join

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

Hybrid Hash Join Algorithm (see book)

- **Partition S into k buckets**

  t buckets S_1, …, S_t stay in memory

  k-t buckets S_{t+1}, …, S_k to disk

- **Partition R into k buckets**

  - First t buckets join immediately with S
  - Rest k-t buckets go to disk

- **Finally, join k-t pairs of buckets:**

  (R_{t+1}, S_{t+1}), (R_{t+2}, S_{t+2}), …, (R_k, S_k)

Summary of External Join Algorithms

- **Block Nested Loop:** $B(S) + B(R)*B(S)/(M-1)$

- **Index Join:**
  - Clustered: $B(R) + T(R)B(S)/V(S,a)$
  - Unclustered: $B(R) + T(R)T(S)/V(S,a)$

- **Merge Join:** $3B(R)+3B(S)$
  - $B(R)+B(S) <= M^2$

- **Partitioned Hash Join:** $3B(R)+3B(S)$
  - $min(B(R), B(S)) <= M^2$