

CSE 444: Database Internals

Lecture 8 Operator Algorithms (part 2)

Announcements

- Lab 2 / part 1 due on Wednesday
 - We will not run any tests – So bugs are OK
- Homework 2 due on Friday
- Paper review for master's due on Friday

Outline

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

Index Based Selection

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

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

Index Based Selection

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

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

What is the cost in each case?

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

Index Based Selection

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

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

What is the cost in each case?

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

Index Based Selection

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

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

What is the cost in each case?

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

Note: we ignore I/O cost for index pages

Index Based Selection

- Example:

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

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

- Table scan:
- Index based selection:

Index Based Selection

- Example:

$B(R) = 2000$
 $T(R) = 100,000$
 $V(R, a) = 20$

cost of $\sigma_{a=v}(R) = ?$
- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:

Index Based Selection

- Example:

$B(R) = 2000$
 $T(R) = 100,000$
 $V(R, a) = 20$

cost of $\sigma_{a=v}(R) = ?$
- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:
 - If index is clustered:
 - If index is unclustered:

Index Based Selection

- Example:

$B(R) = 2000$
 $T(R) = 100,000$
 $V(R, a) = 20$

cost of $\sigma_{a=v}(R) = ?$
- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:
 - If index is clustered: $B(R)/V(R,a) = 100$ I/Os
 - If index is unclustered:

Index Based Selection

- Example:

$B(R) = 2000$
 $T(R) = 100,000$
 $V(R, a) = 20$

cost of $\sigma_{a=v}(R) = ?$
- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:
 - If index is clustered: $B(R)/V(R,a) = 100$ I/Os
 - If index is unclustered: $T(R)/V(R,a) = 5,000$ I/Os

Index Based Selection

- Example:

$B(R) = 2000$
 $T(R) = 100,000$
 $V(R, a) = 20$

cost of $\sigma_{a=v}(R) = ?$
- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:
 - If index is clustered: $B(R)/V(R,a) = 100$ I/Os
 - If index is unclustered: $T(R)/V(R,a) = 5,000$ I/Os

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

Index Nested Loop Join

$R \bowtie S$

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S
- **Cost:**
 - If index on S is clustered: $B(R) + T(R)B(S)/V(S,a)$
 - If index on S is unclustered: $B(R) + T(R)T(S)/V(S,a)$

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

- 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

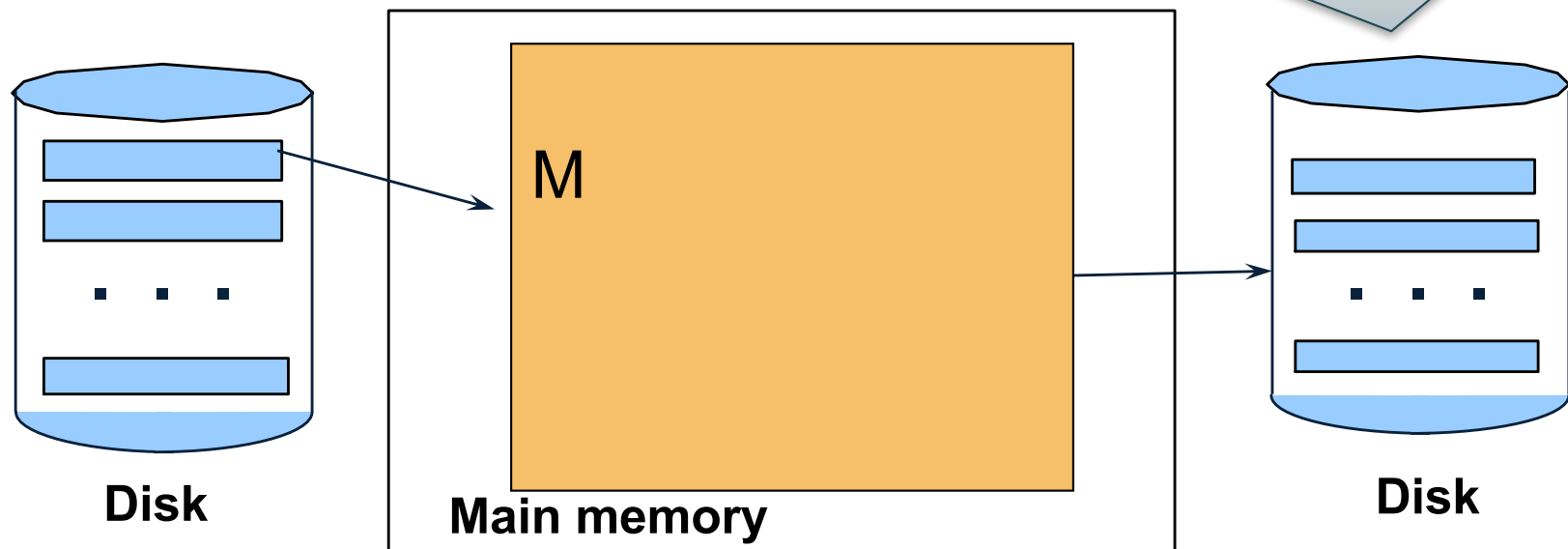
External Merge-Sort: Step 1

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

External Merge-Sort: Step 1

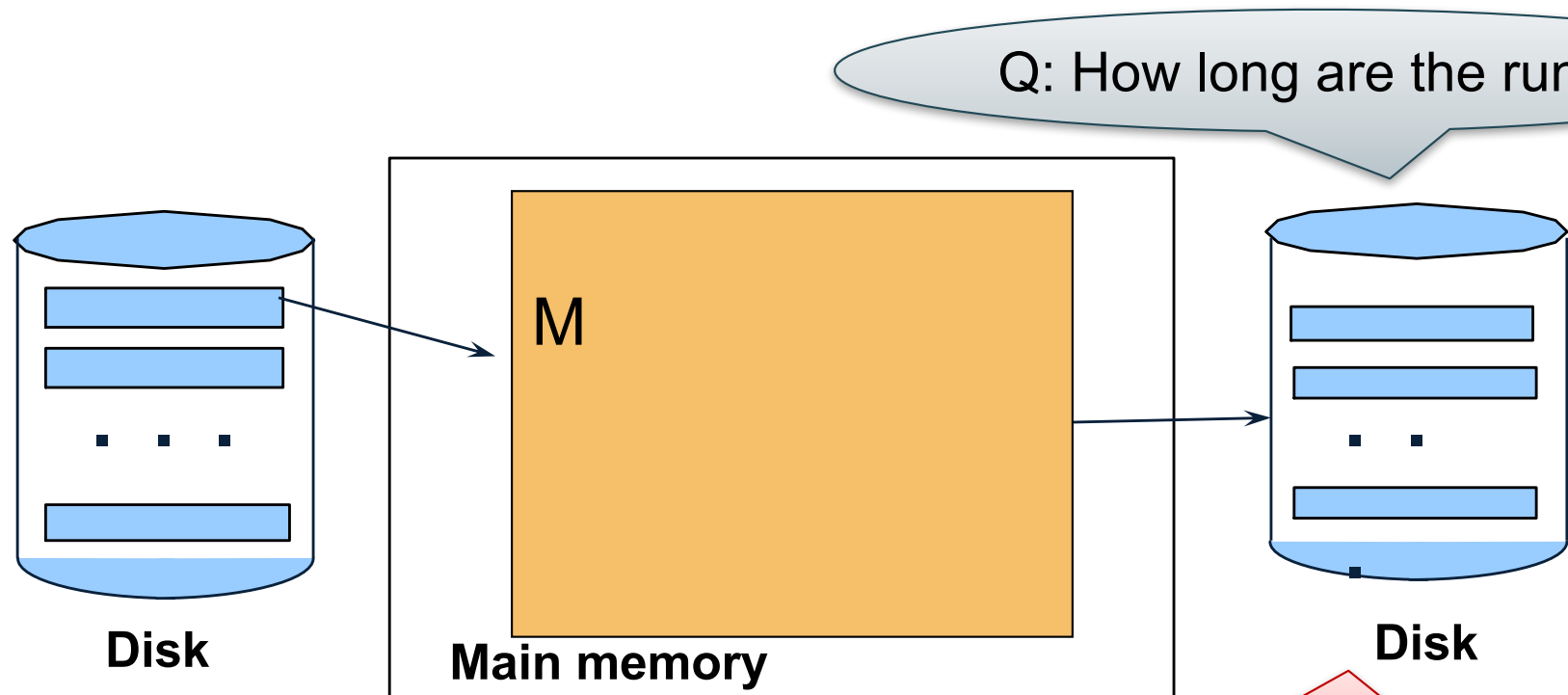
Phase one: load M blocks in memory, sort, sent to disk, repeat

Q: How long are the runs?



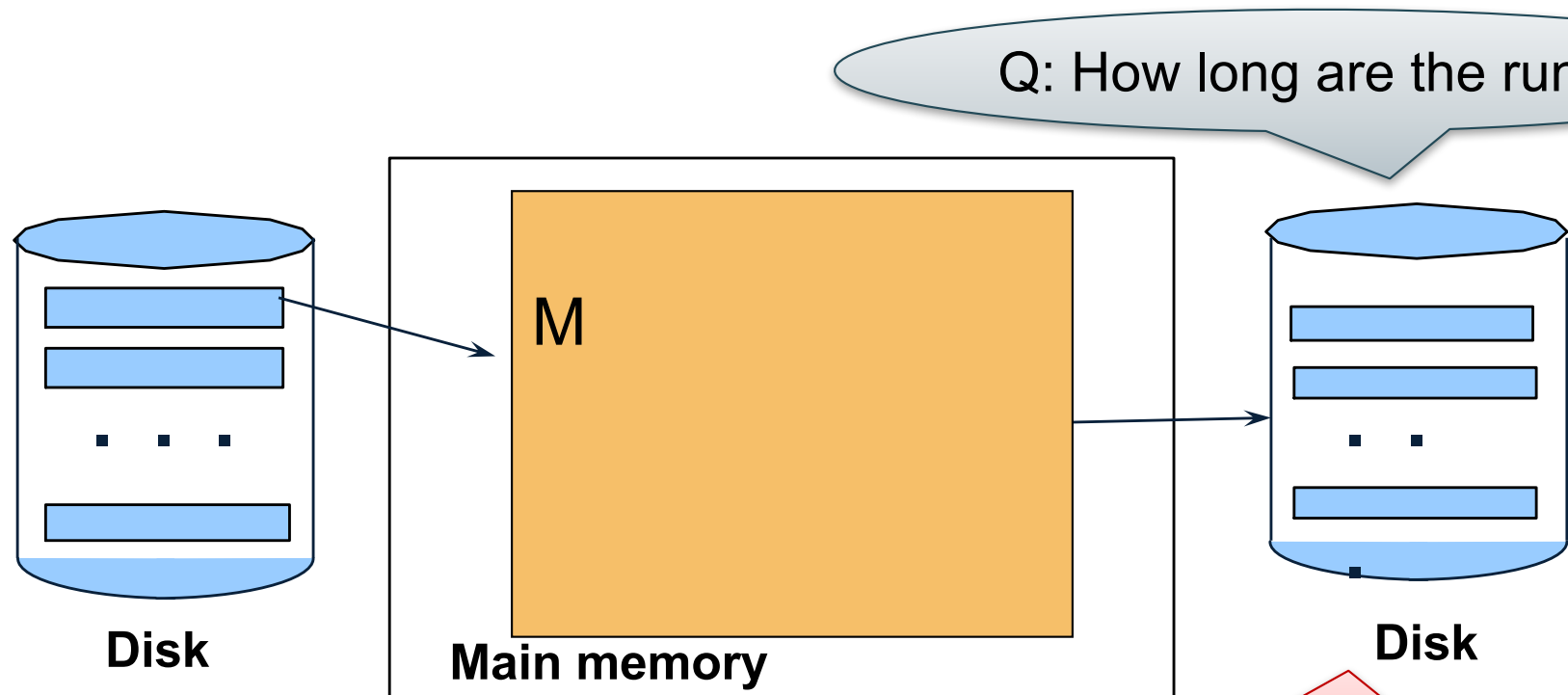
External Merge-Sort: Step 1

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



External Merge-Sort: Step 1

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

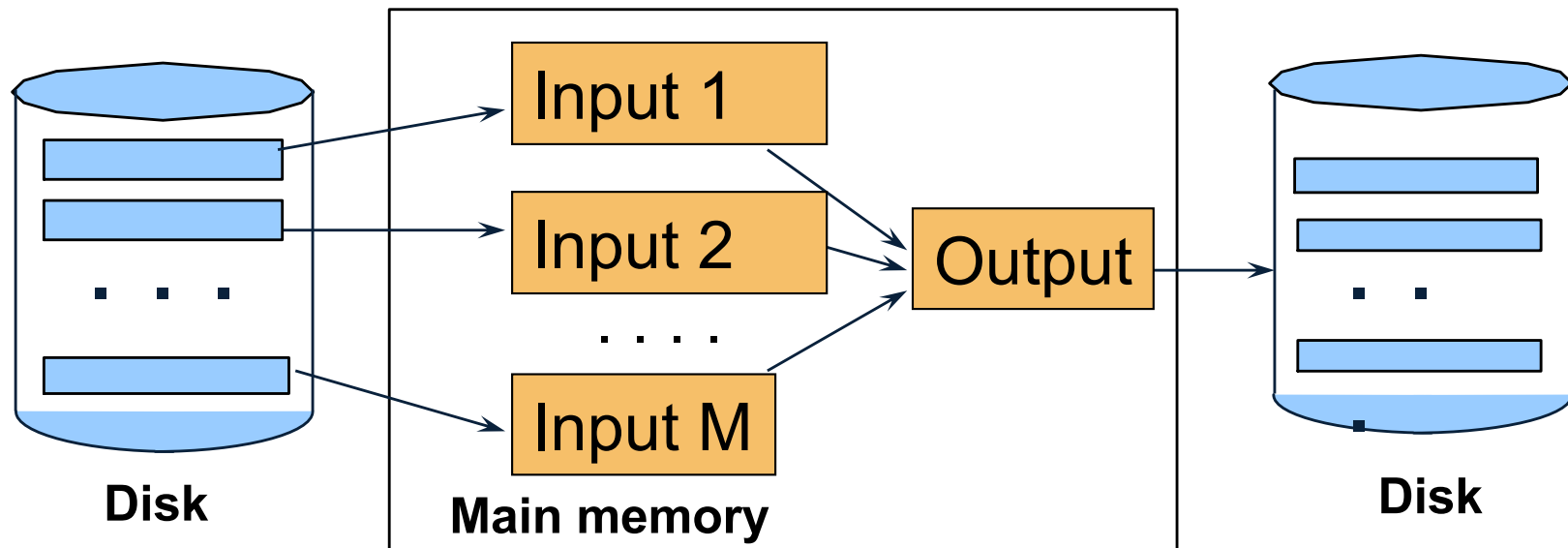


Can increase to length $2M$ using “replacement selection”

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 $B \leq M^2$ then we are done

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, **?**

Cost of External Merge Sort

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

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 $R \bowtie S$

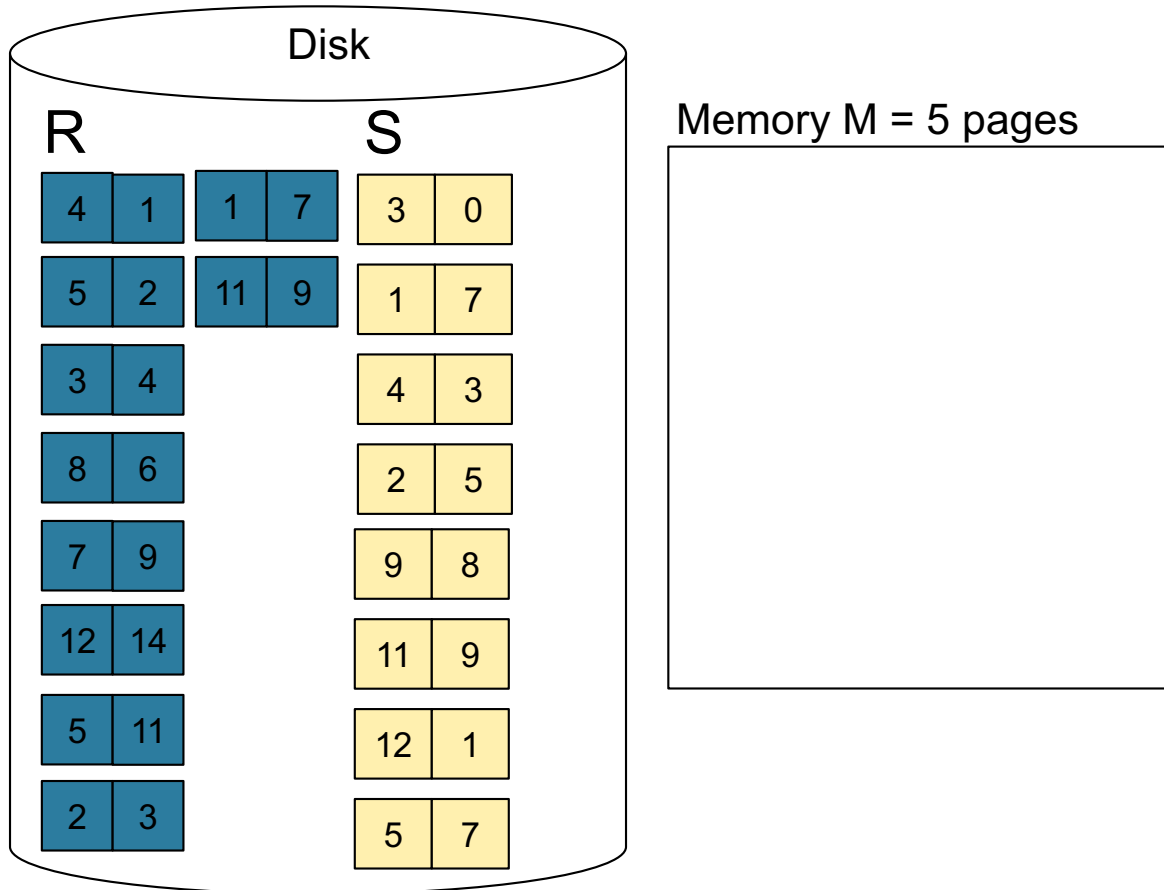
- Step 1a: generate initial runs for R
- Step 1b: generate initial runs for S
- Step 2: merge and join
 - Either merge first and then join
 - Or merge & join at the same time

Merge-Join Example

Setup: Want to join R and S

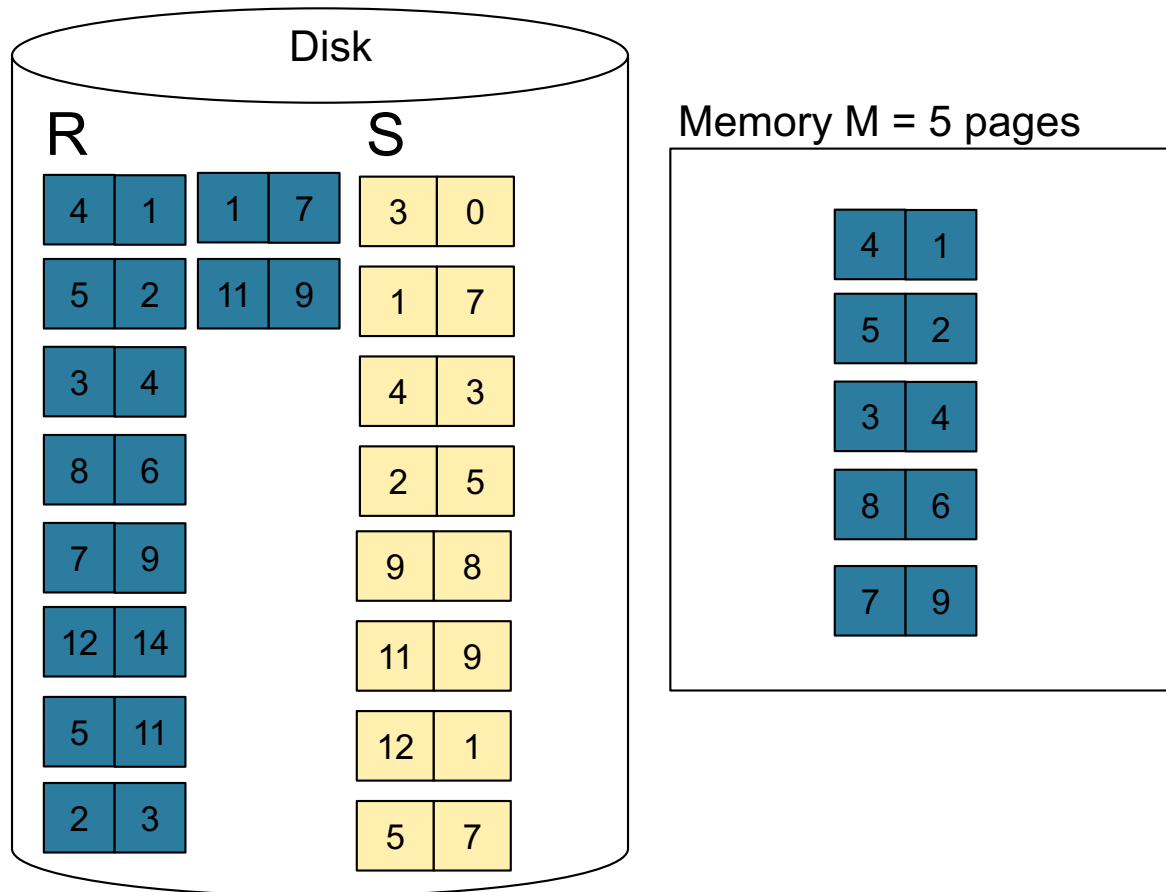
- Relation R has 10 pages with 2 tuples per page
- Relation S has 8 pages with 2 tuples per page

Values shown are values of join attribute for each given tuple



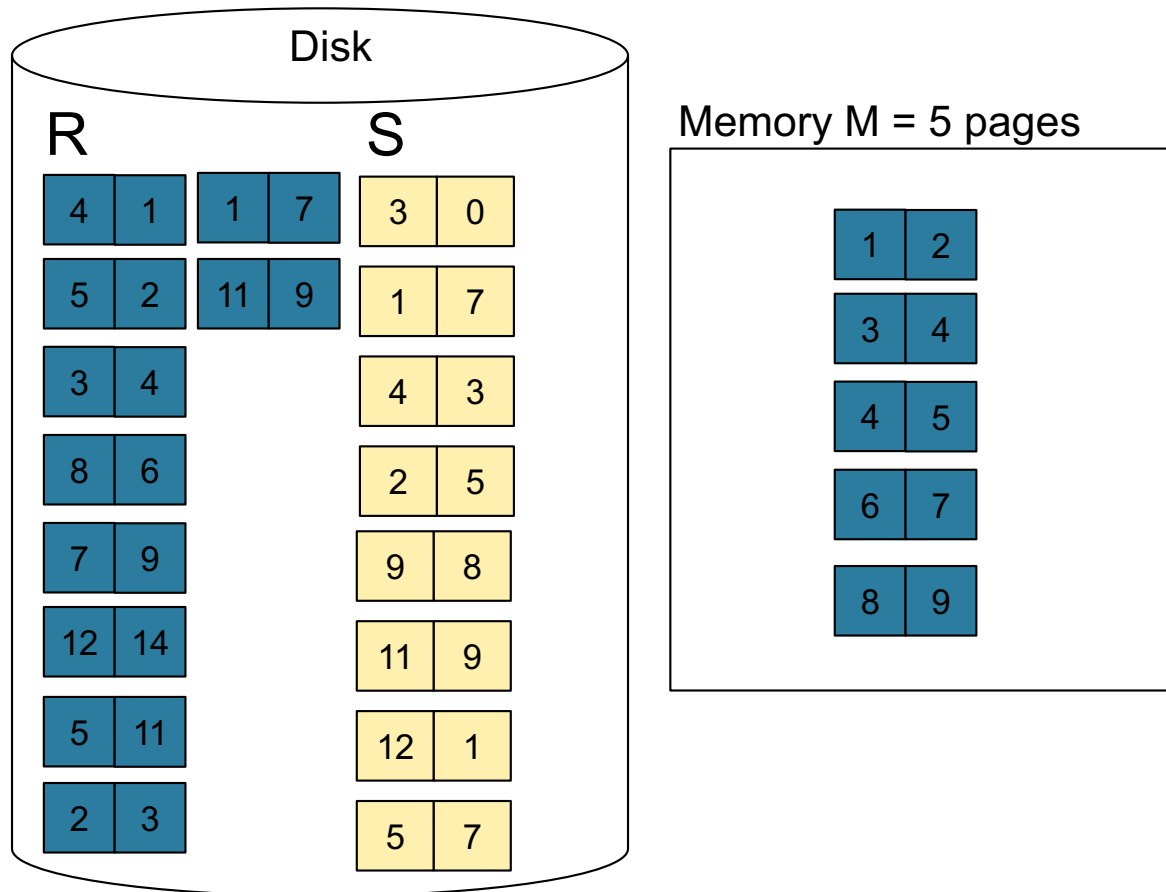
Merge-Join Example

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



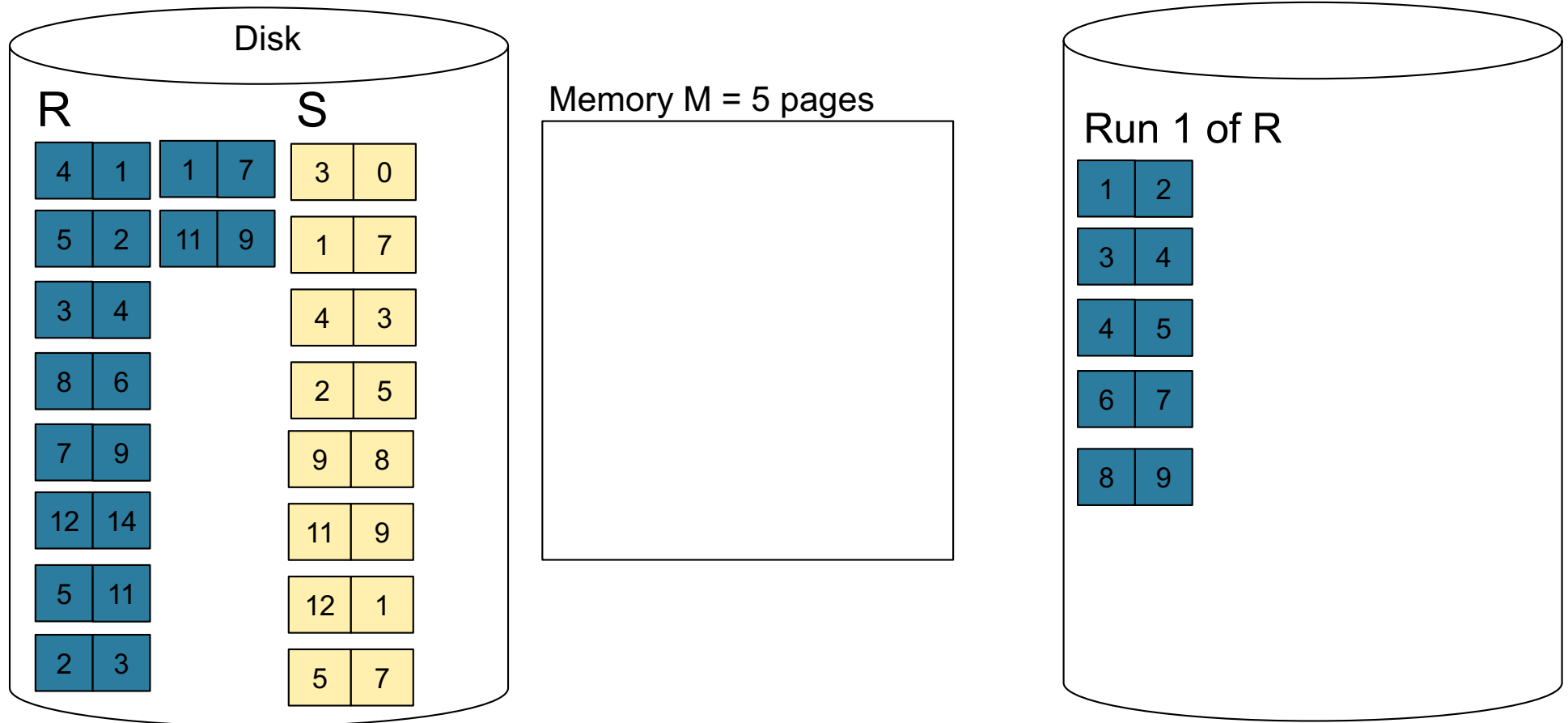
Merge-Join Example

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



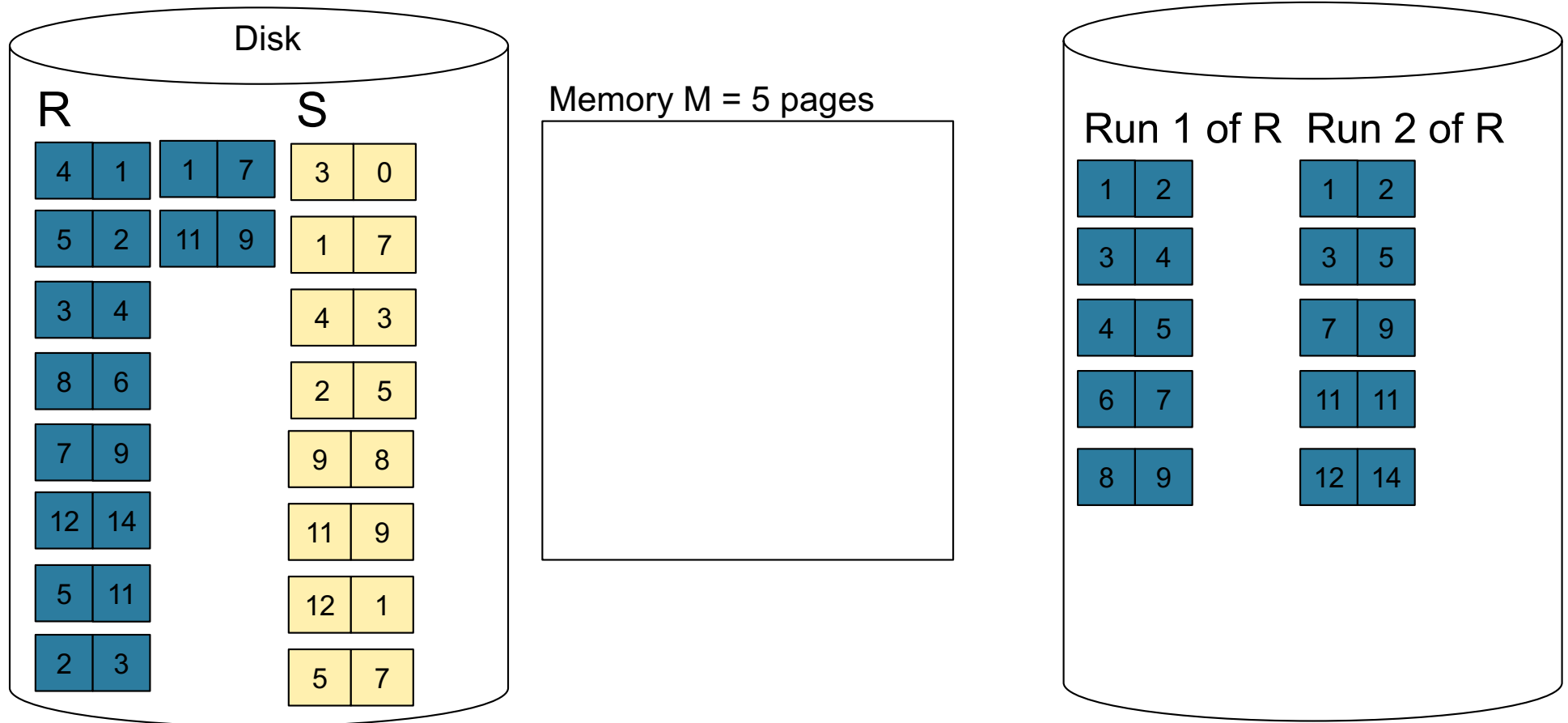
Merge-Join Example

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



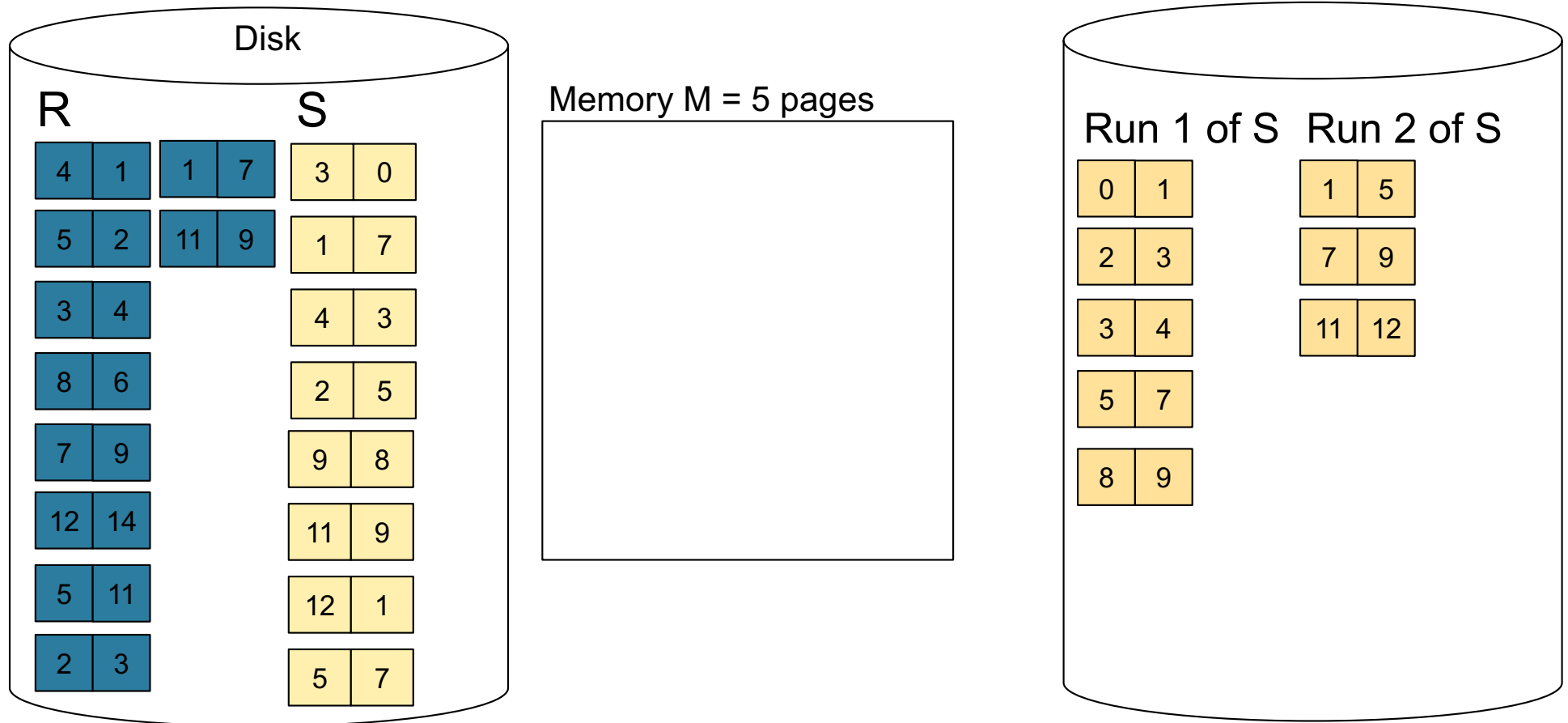
Merge-Join Example

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



Merge-Join Example

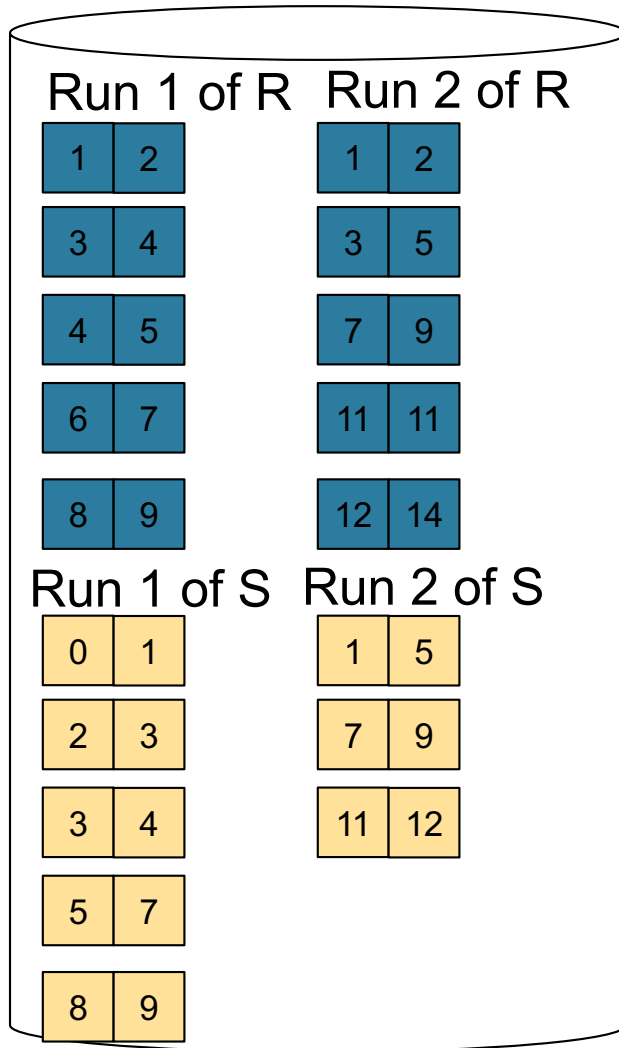
Step 1: Do the same with S



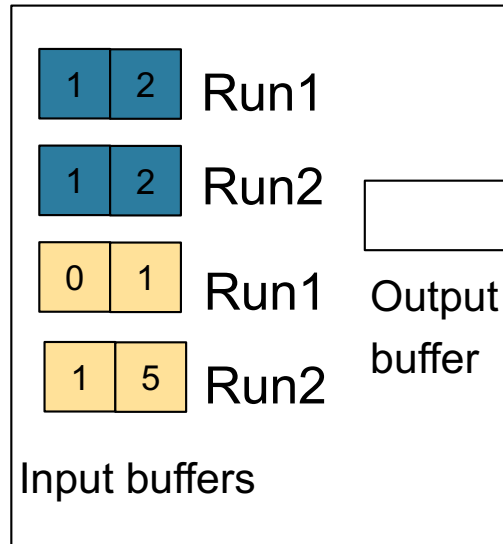
Merge-Join Example

Step 4: Join while merging sorted runs

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



Memory M = 5 pages



Step 4: Join while merging

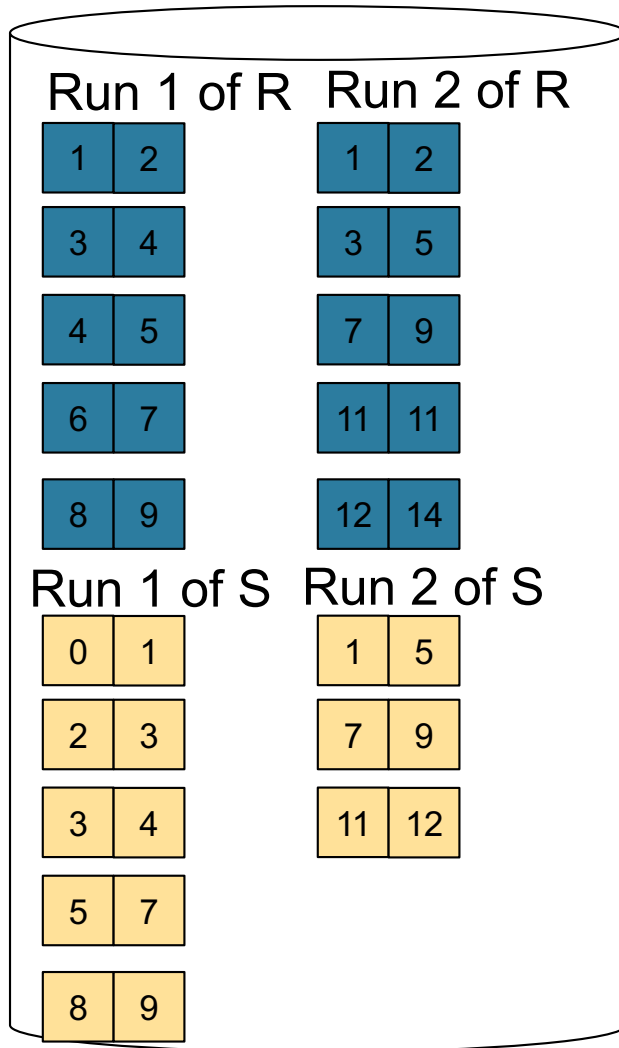
Output tuples

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

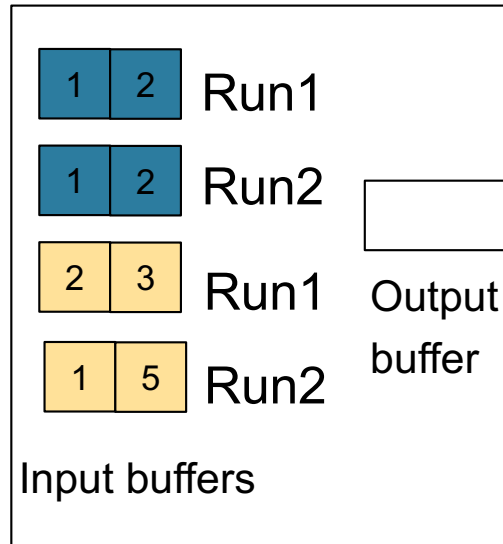
Merge-Join Example

Step 4: Join while merging sorted runs

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



Memory M = 5 pages



Step 4: Join while merging

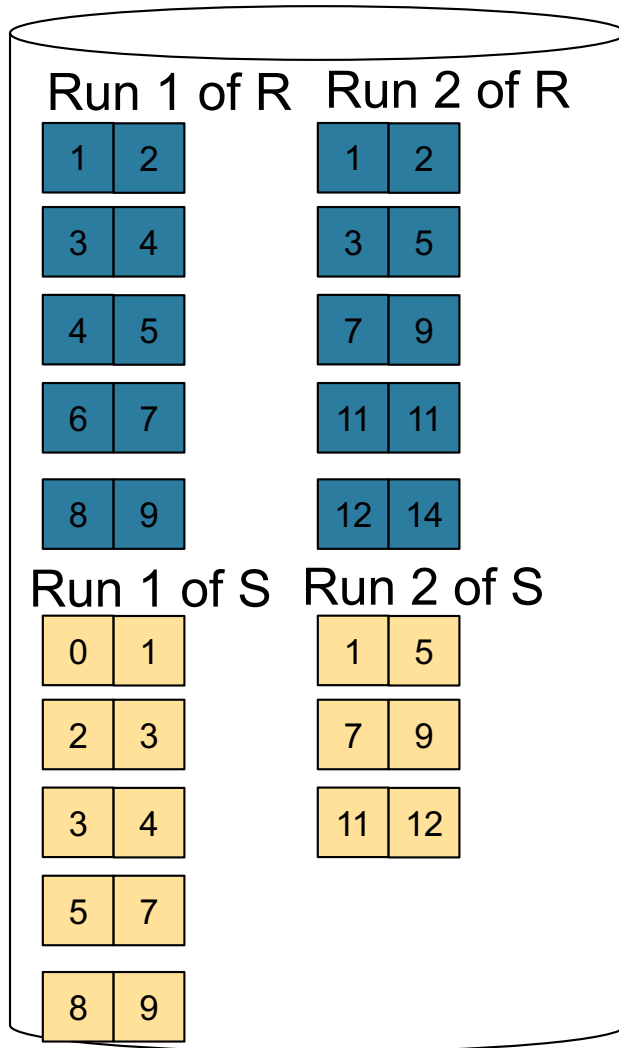
Output tuples

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

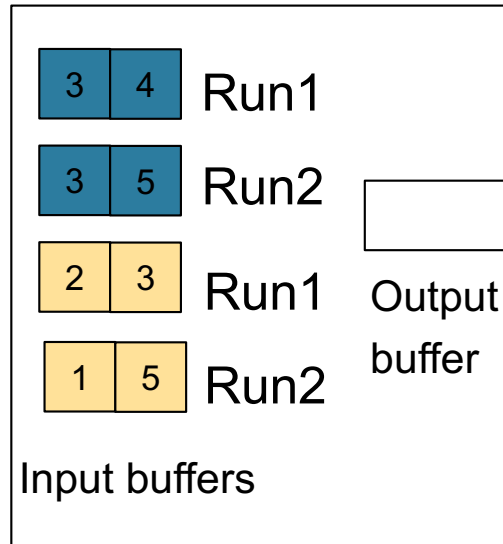
Merge-Join Example

Step 4: Join while merging sorted runs

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



Memory M = 5 pages

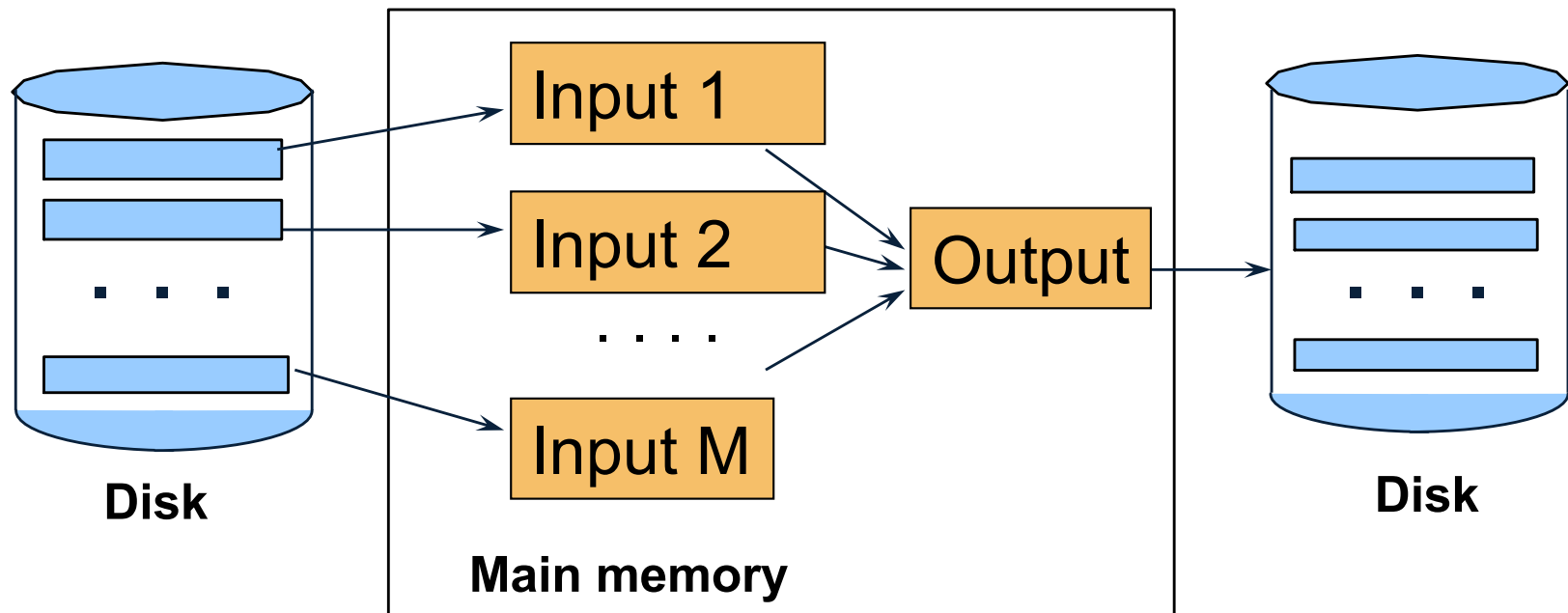


Step 4: Join while merging

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 $M_1 + M_2 \leq M$

Partitioned Hash Algorithms

- Partition R it into k buckets:
 $R_1, R_2, R_3, \dots, R_k$

Partitioned Hash Algorithms

- Partition R it into k buckets:
 $R_1, R_2, R_3, \dots, R_k$
- Assuming $B(R_1)=B(R_2)=\dots=B(R_k)$, we have
 $B(R_i) = B(R)/k$, for all i

Partitioned Hash Algorithms

- Partition R it into k buckets:
 $R_1, R_2, R_3, \dots, R_k$
- Assuming $B(R_1)=B(R_2)=\dots=B(R_k)$, we have
 $B(R_i) = B(R)/k$, for all i
- Goal: each R_i should fit in main memory:
 $B(R_i) \leq M$

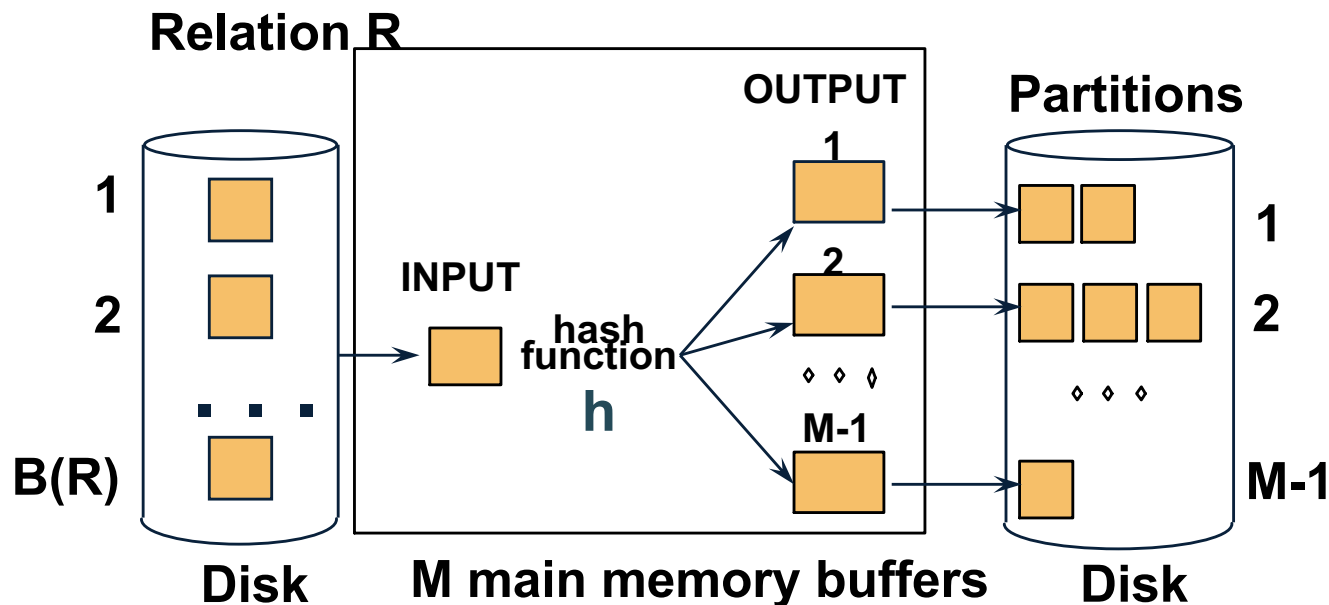
Partitioned Hash Algorithms

- Partition R it into k buckets:
 $R_1, R_2, R_3, \dots, R_k$
- Assuming $B(R_1)=B(R_2)=\dots=B(R_k)$, 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. $B(R) \leq M^2$

Grace-Join

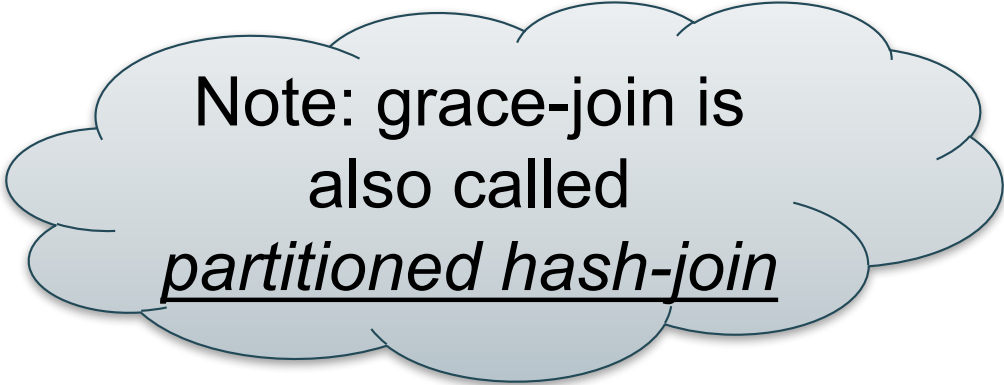
$R \bowtie S$

Note: grace-join is
also called
partitioned hash-join

Grace-Join

$R \bowtie S$

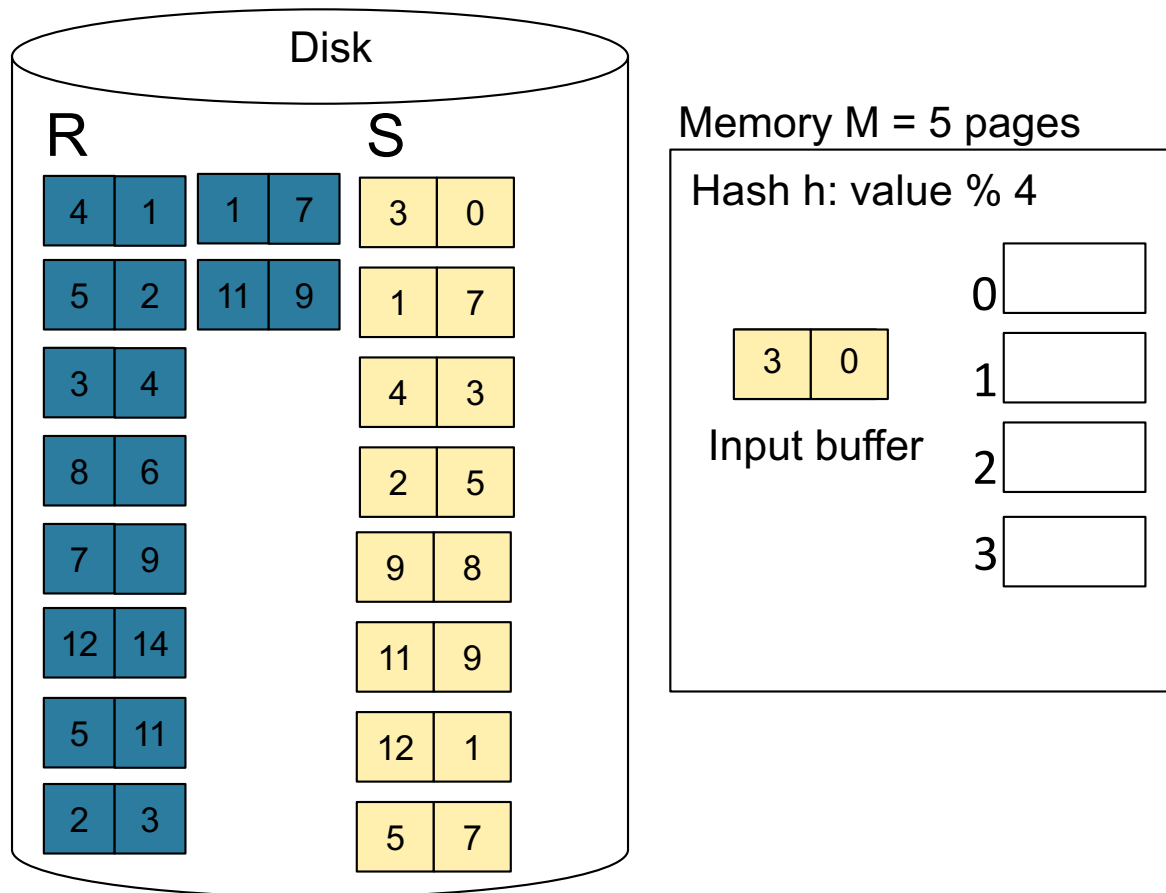
- Step 1:
 - Hash S into M buckets
 - Send all buckets to disk
- Step 2
 - Hash R into M buckets
 - Send all buckets to disk
- Step 3
 - Join every pair of buckets



Note: grace-join is
also called
partitioned hash-join

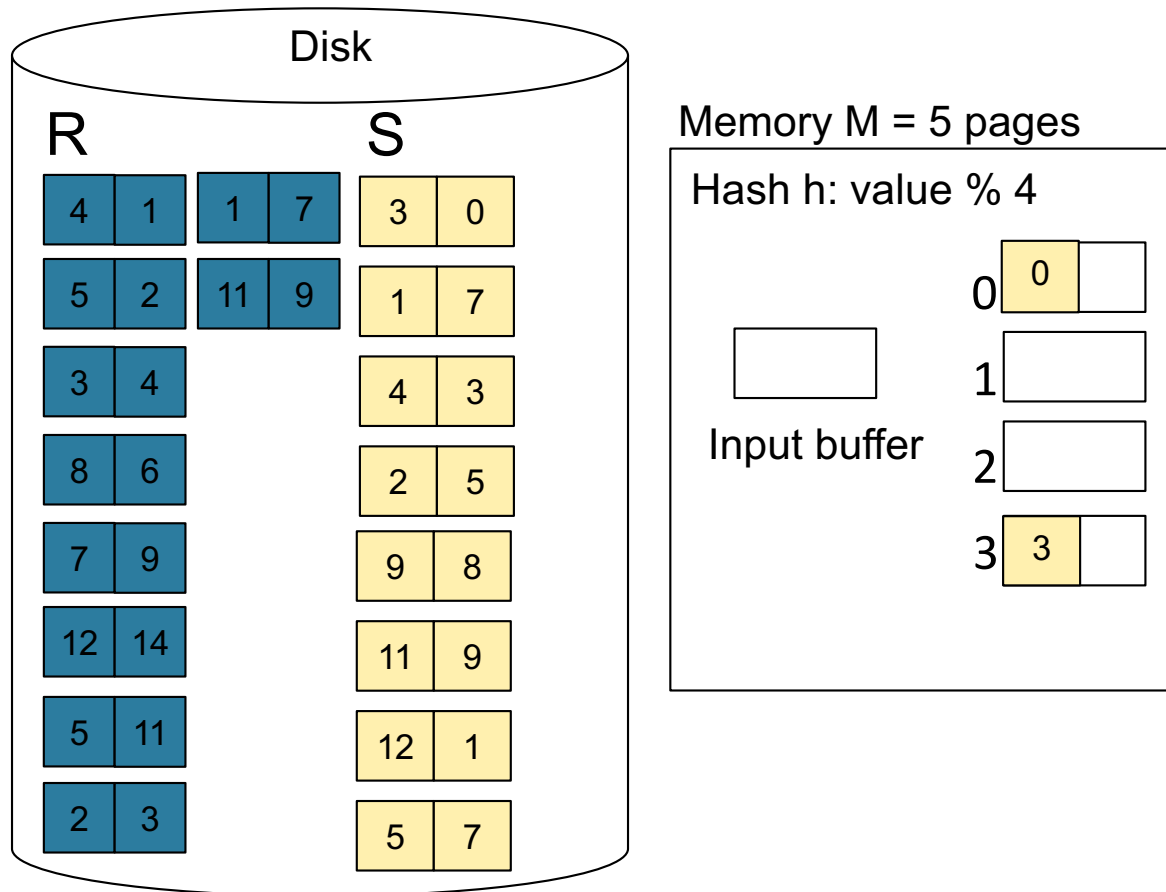
Partitioned Hash-Join Example

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



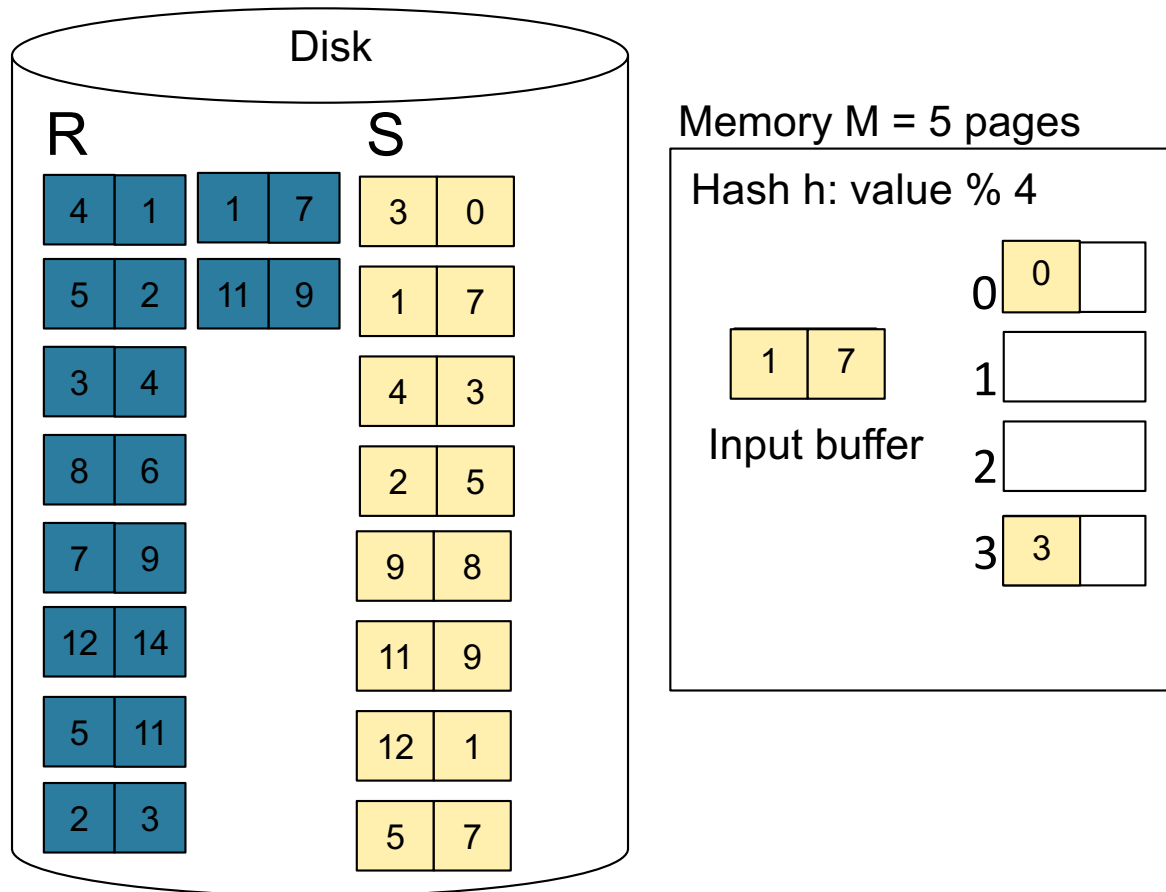
Partitioned Hash-Join Example

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



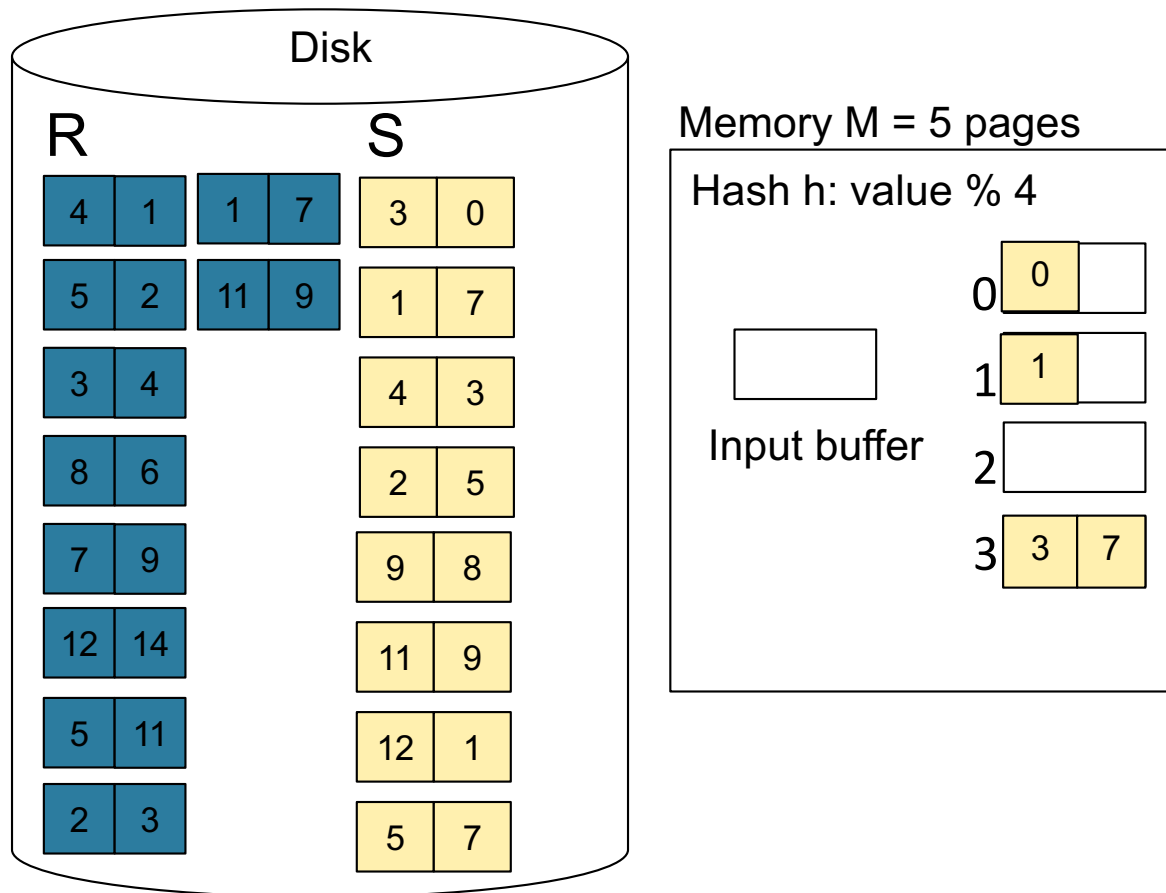
Partitioned Hash-Join Example

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



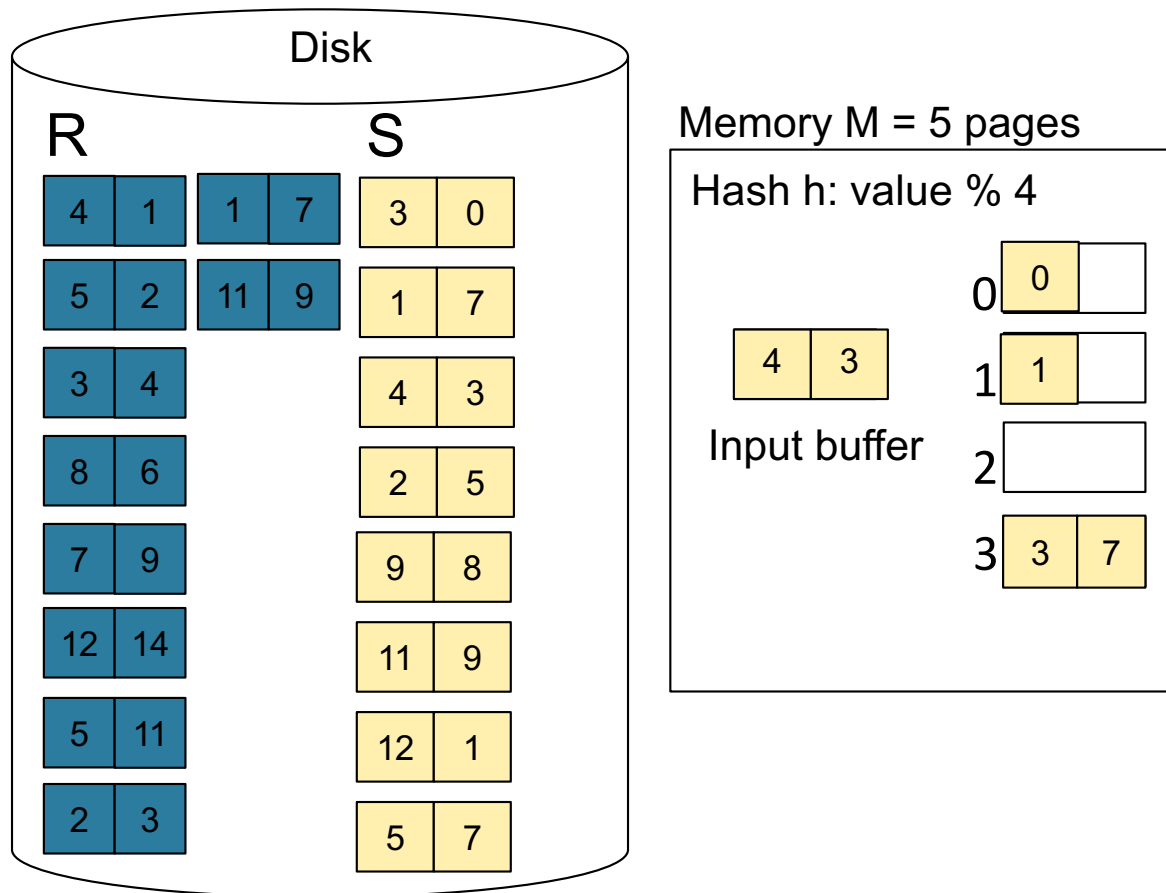
Partitioned Hash-Join Example

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



Partitioned Hash-Join Example

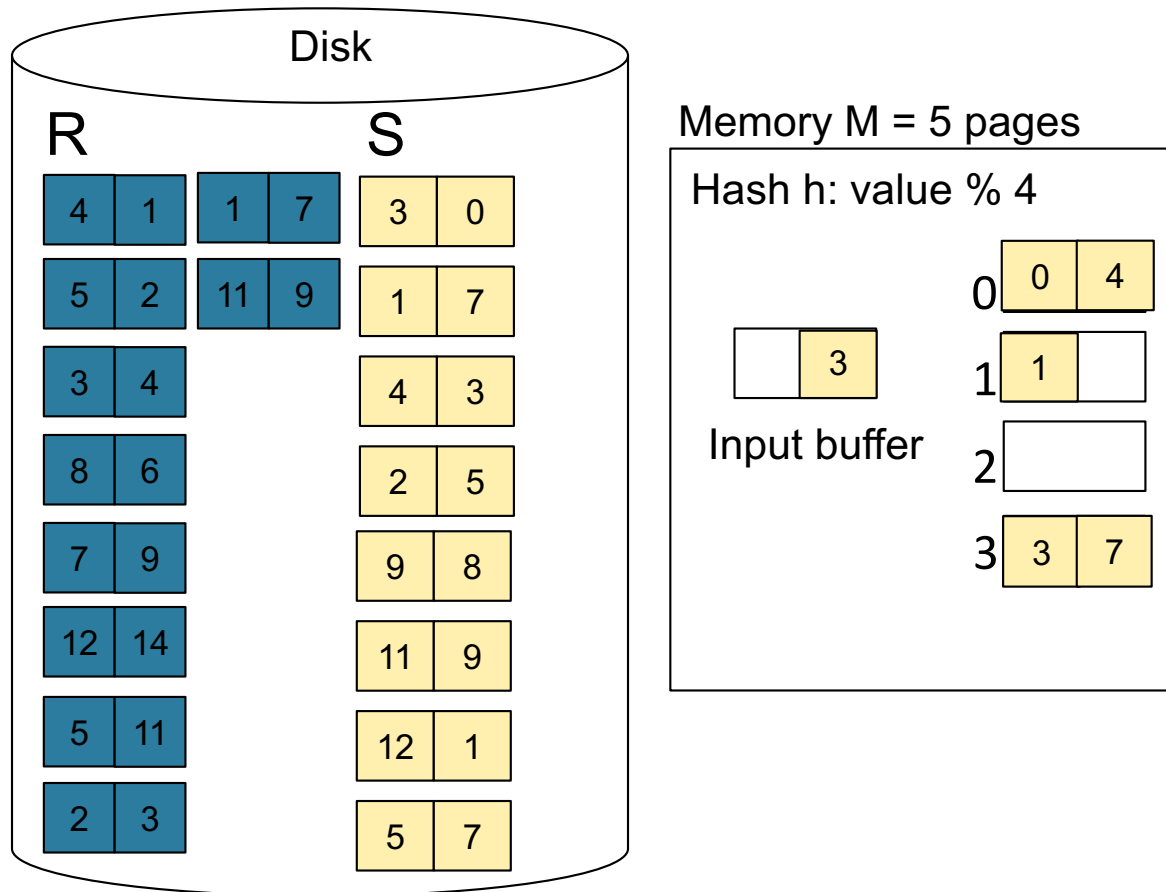
Step 1: Read relation S one page at a time and hash into the 4 buckets



Partitioned Hash-Join Example

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

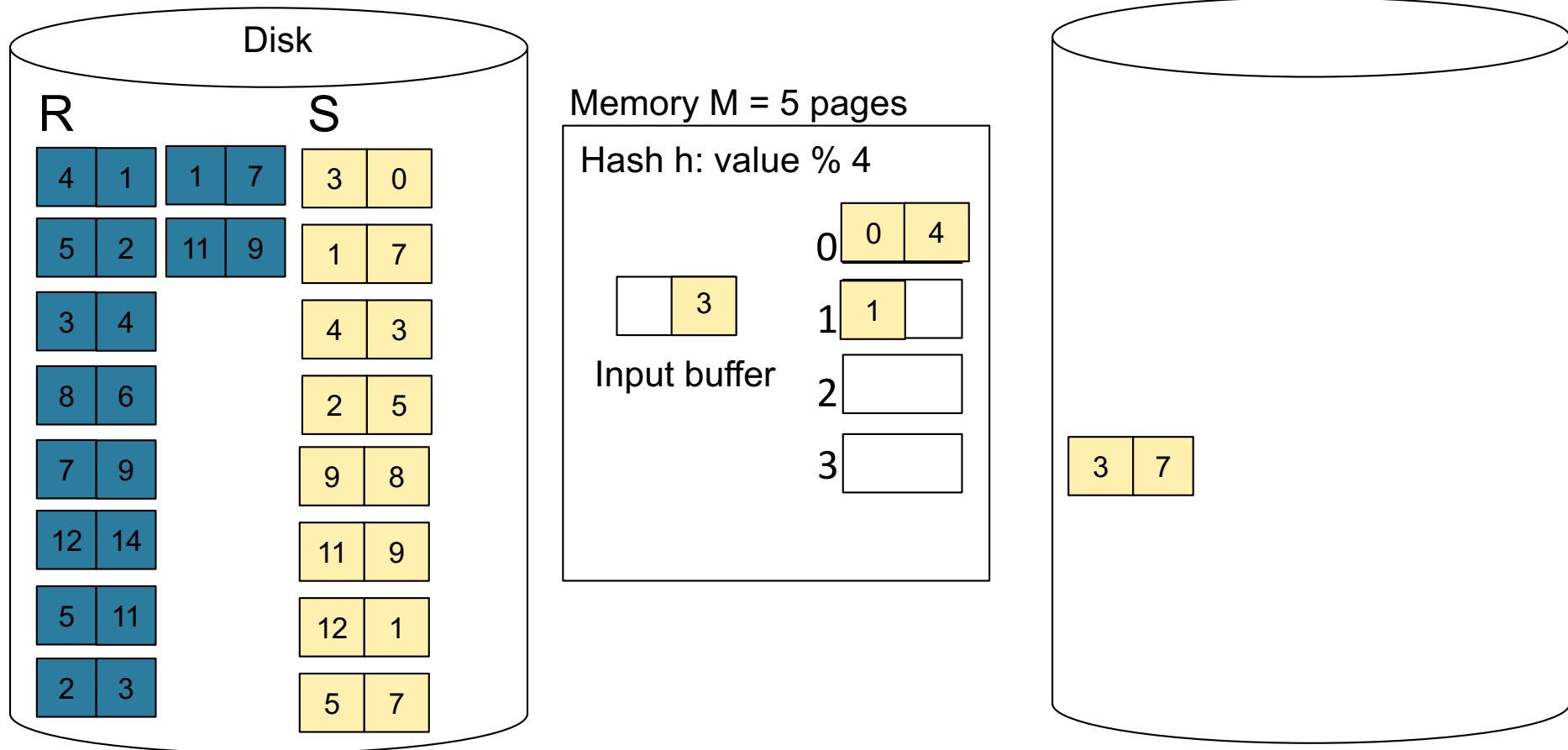
When a bucket fills up, flush it to disk



Partitioned Hash-Join Example

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

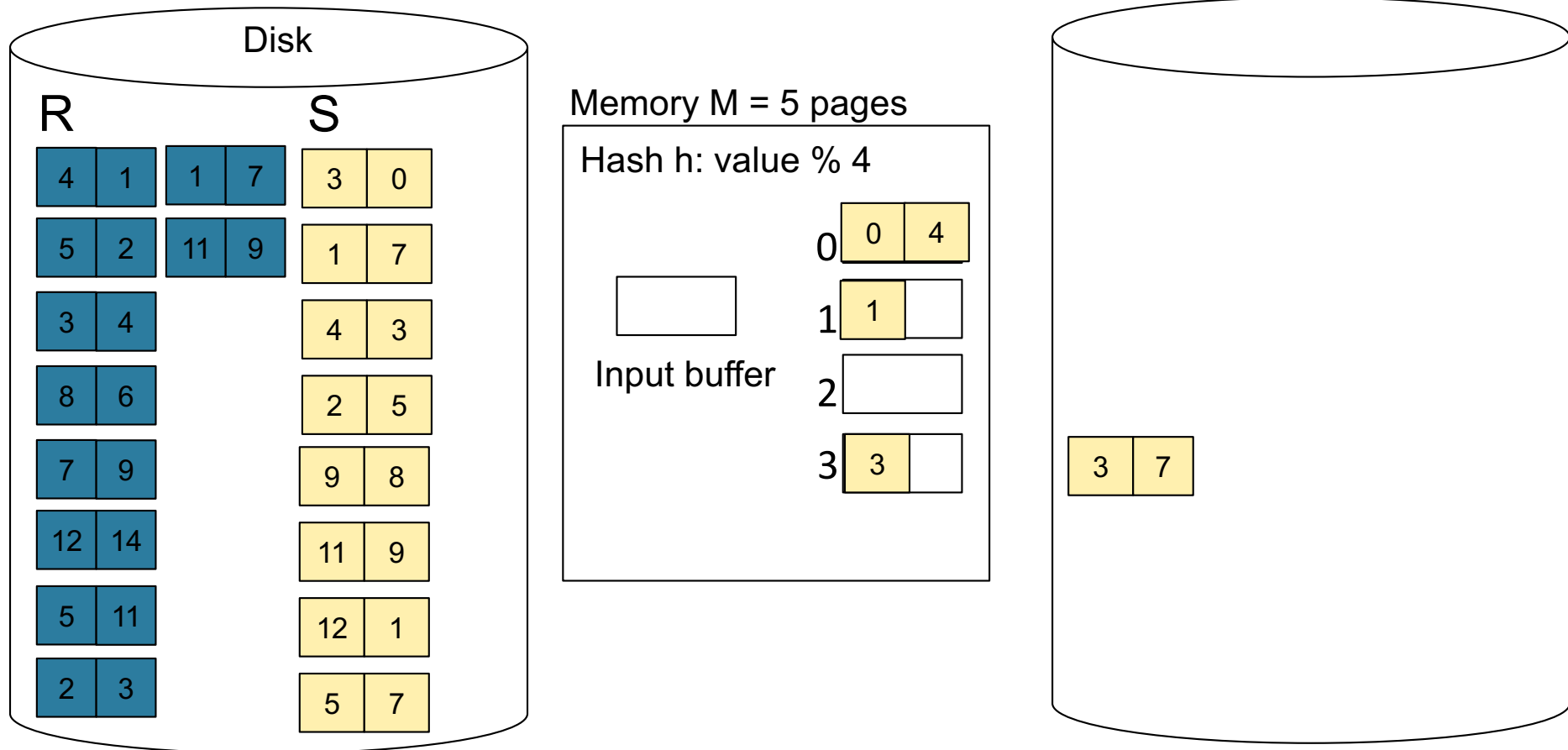
When a bucket fills up, flush it to disk



Partitioned Hash-Join Example

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

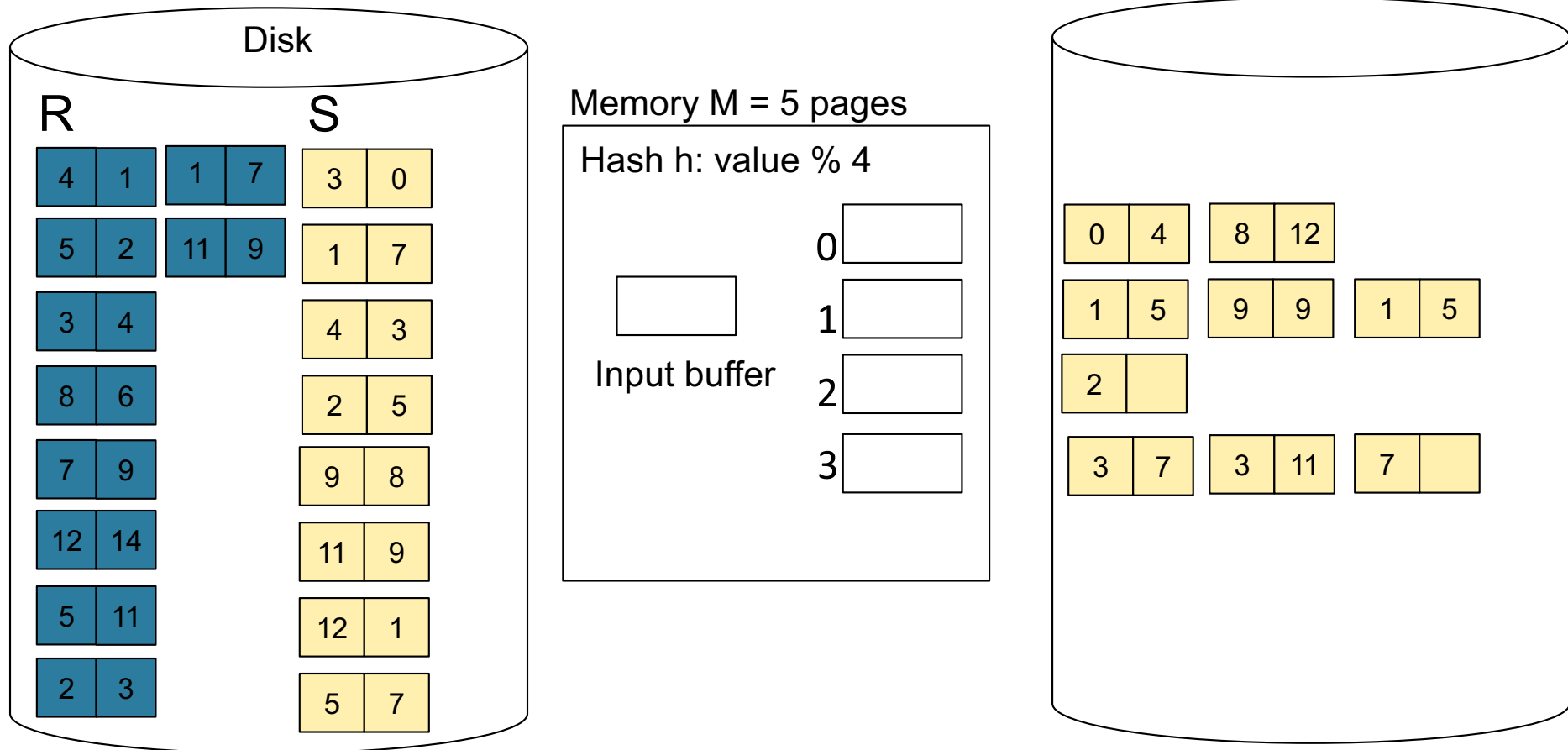
When a bucket fills up, flush it to disk



Partitioned Hash-Join Example

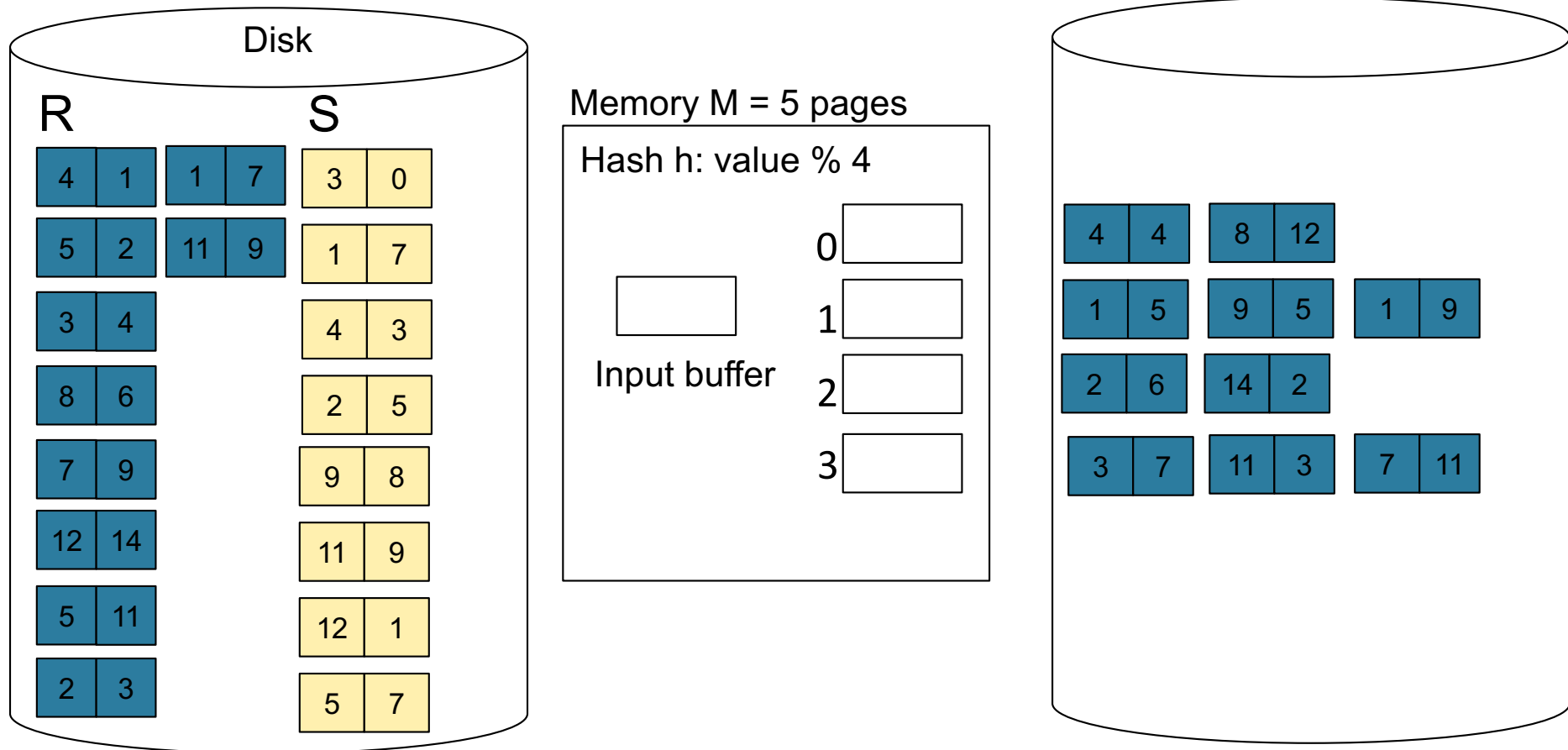
Step 1: Read relation S one page at a time and hash into the 4 buckets

At the end, we get relation S back on disk split into 4 buckets



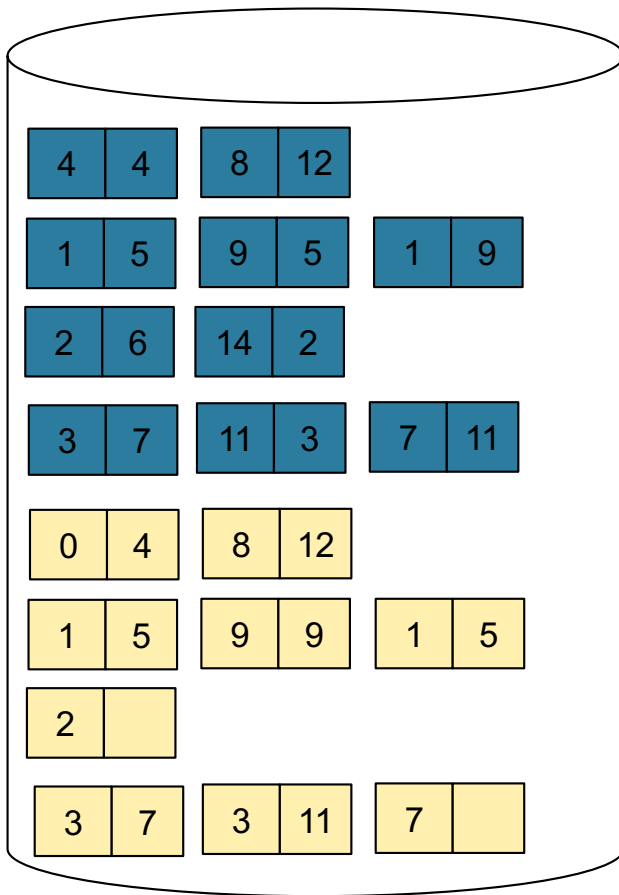
Partitioned Hash-Join Example

Step 2: Read relation R one page at a time and hash into same 4 buckets



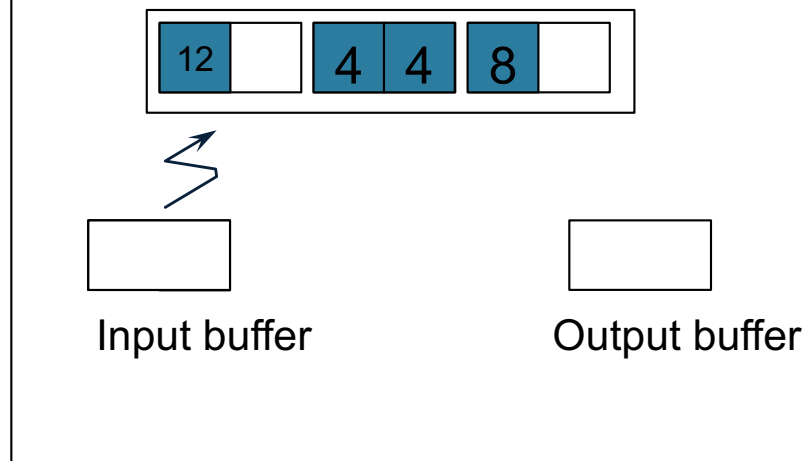
Partitioned Hash-Join Example

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



Memory $M = 5$ pages

Hash h_2 : value % 3

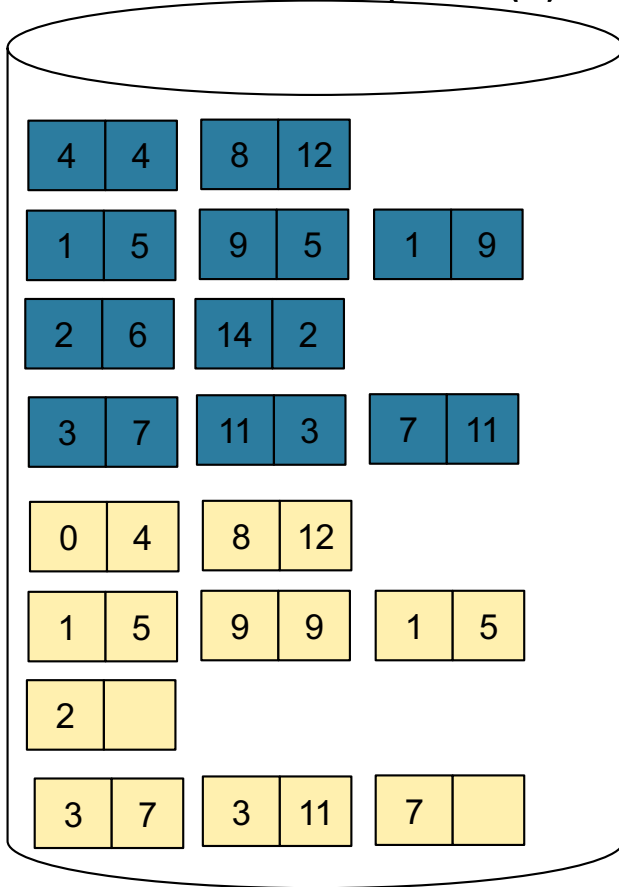


Partitioned Hash-Join Example

Step 4: Scan matching partition of S and probe the hash table

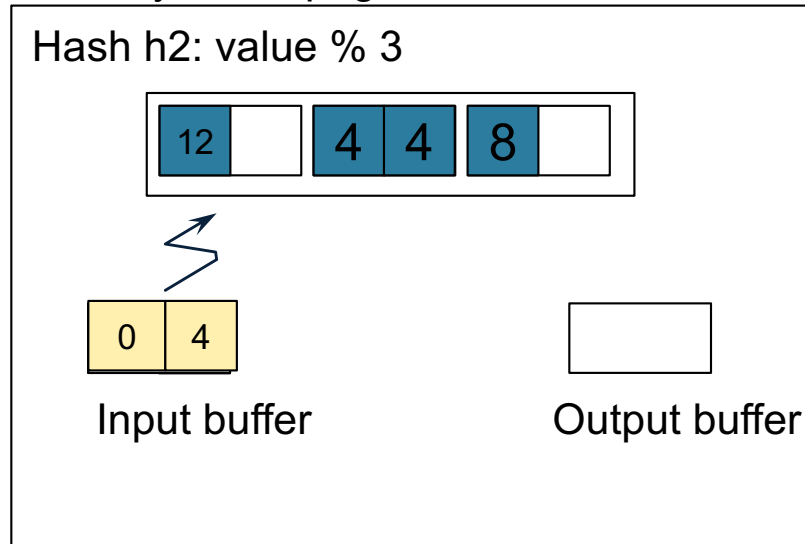
Step 5: Repeat for all the buckets

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



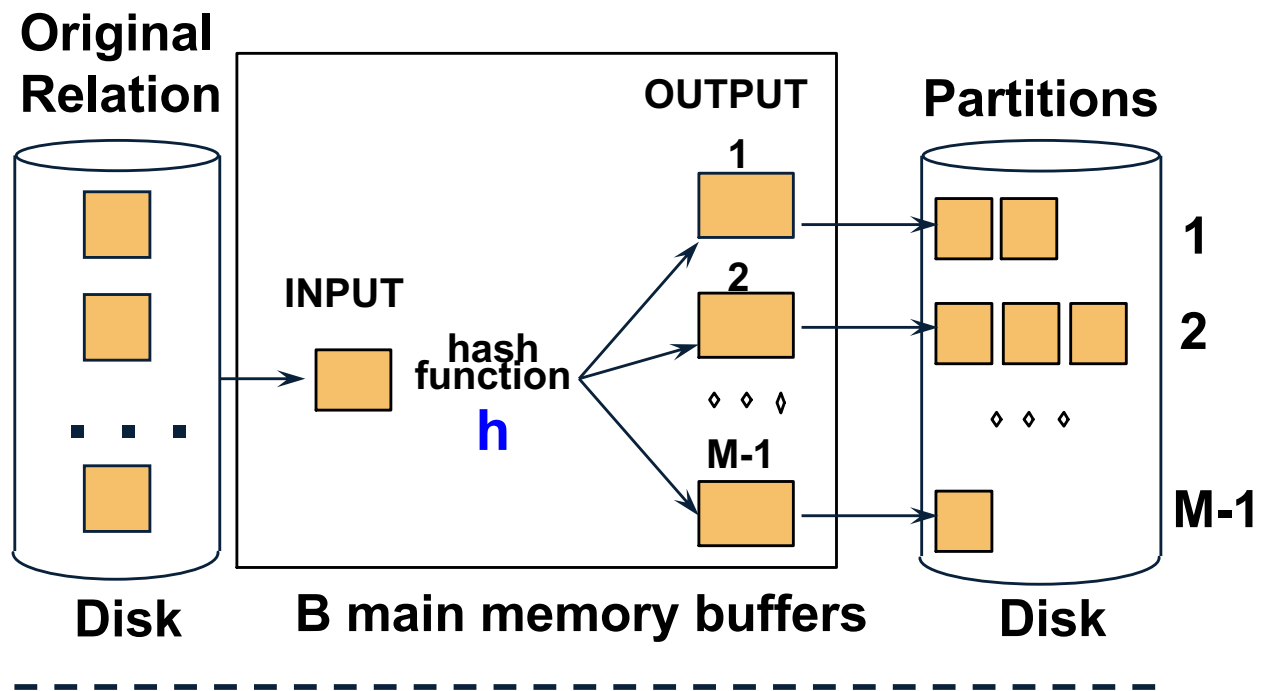
Memory $M = 5$ pages

Hash h_2 : value % 3



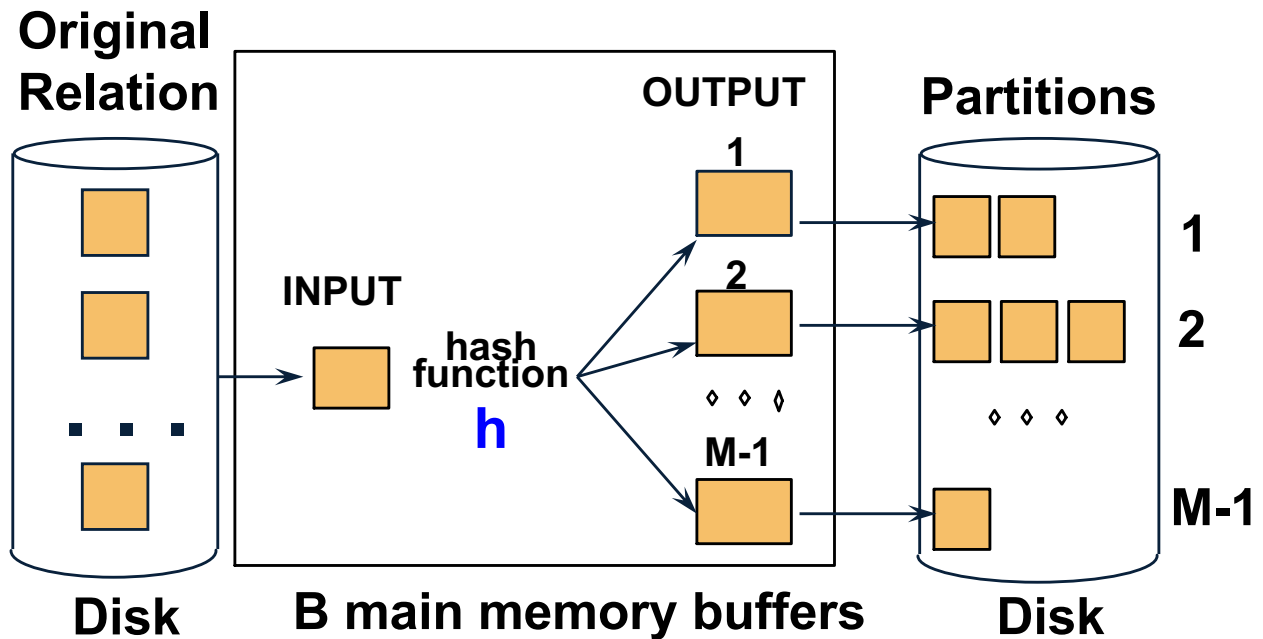
Grace-Join

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

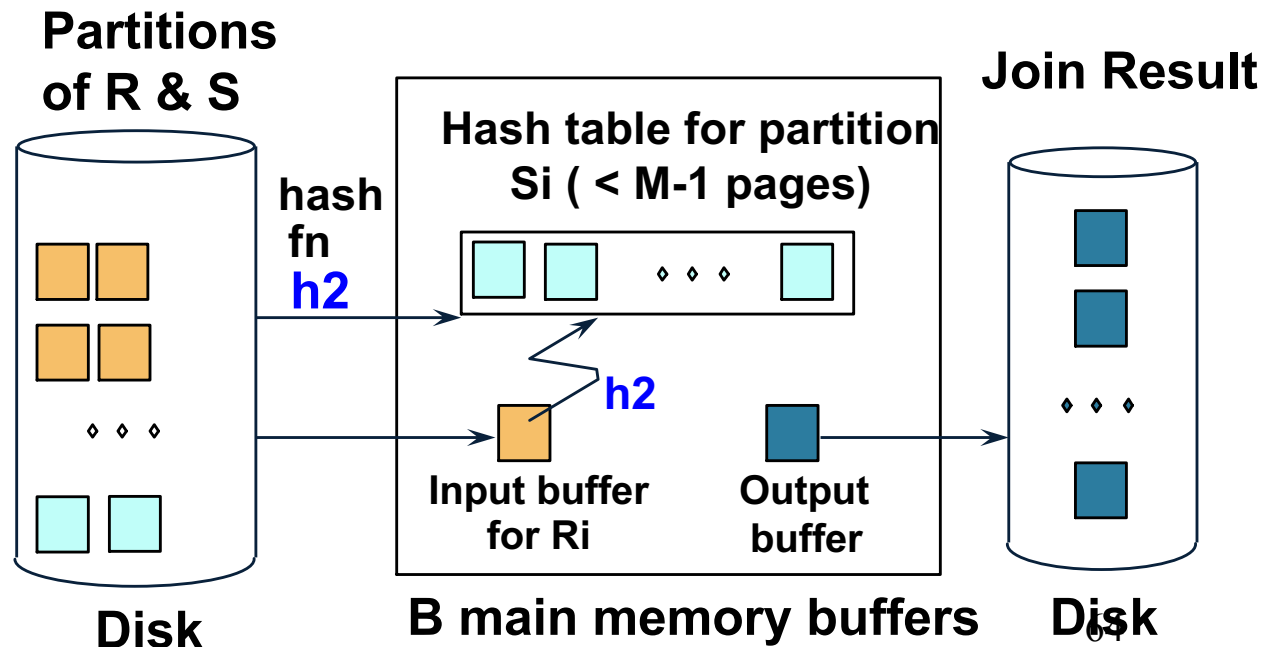


Grace-Join

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



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



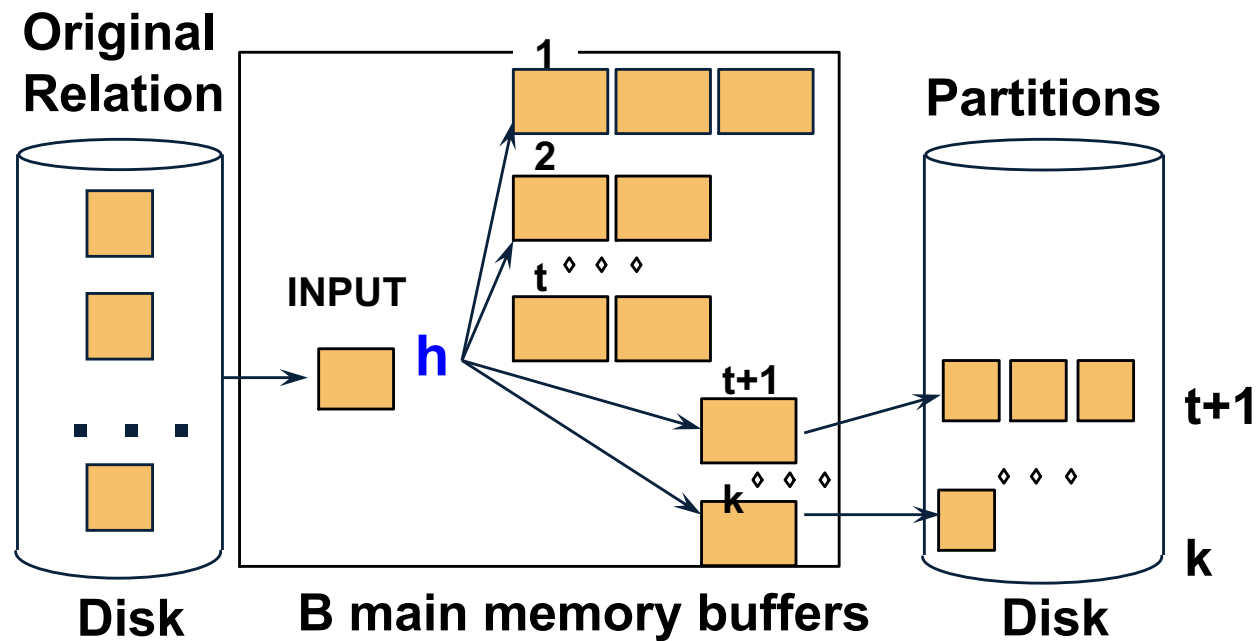
Grace Join

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

Hybrid Hash Join Algorithm

- Partition S into k buckets
 - t buckets S_1, \dots, S_t stay in memory
 - $k-t$ buckets S_{t+1}, \dots, 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:
 $(R_{t+1}, S_{t+1}), (R_{t+2}, S_{t+2}), \dots, (R_k, S_k)$

Hybrid Hash Join Algorithm



Hybrid Join Algorithm

- How to choose k and t ?

Hybrid Join Algorithm

- How to choose k and t ?
 - Choose k large but s.t. $k \leq M$

Hybrid Join Algorithm

- How to choose k and t ?
 - Choose k large but s.t.

One block/bucket in memory

$$k \leq M$$

Hybrid Join Algorithm

- How to choose k and t ?

- Choose k large but s.t.

One block/bucket in memory

$$k \leq M$$

- Choose t/k large but s.t.

$$t/k * B(S) \leq M$$

Hybrid Join Algorithm

- How to choose k and t ?
 - Choose k large but s.t.
 - Choose t/k large but s.t.

One block/bucket in memory

$$k \leq M$$

First t buckets in memory

$$t/k * B(S) \leq M$$

Hybrid Join Algorithm

- How to choose k and t ?

- Choose k large but s.t.

One block/bucket in memory

$$k \leq M$$

- Choose t/k large but s.t.

First t buckets in memory

$$t/k * B(S) \leq M$$

- Together:

$$t/k * B(S) + k - t \leq M$$

Hybrid Join Algorithm

- How to choose k and t ?

- Choose k large but s.t.

One block/bucket in memory

$$k \leq M$$

- Choose t/k large but s.t.

First t buckets in memory

$$t/k * B(S) \leq M$$

- Together:

$$t/k * B(S) + k - t \leq M$$

- Assuming $t/k * B(S) \gg k - t$: $t/k = M/B(S)$

Hybrid Join Algorithm

- How to choose k and t ?

- Choose k large but s.t.

One block/bucket in memory

$$k \leq M$$

- Choose t/k large but s.t.

First t buckets in memory

$$t/k * B(S) \leq M$$

- Together:

$$t/k * B(S) + k - t \leq M$$

- Assuming $t/k * B(S) \gg k - t$: $t/k = M/B(S)$

Total size of first t buckets

Hybrid Join Algorithm

- How to choose k and t ?

- Choose k large but s.t.

One block/bucket in memory

$$k \leq M$$

First t buckets in memory

- Choose t/k large but s.t.

$$t/k * B(S) \leq M$$

- Together:

$$t/k * B(S) + k - t \leq M$$

- Assuming $t/k * B(S) \gg k - t$: $t/k = M/B(S)$

Total size of first t buckets

Number of remaining buckets

Hybrid Join Algorithm

Even better: adjust t dynamically

- Start with $t = k$: all buckets are in main memory
- Read blocks from S , insert tuples into buckets
- When out of memory:
 - Send one bucket to disk
 - $t := t-1$
- Worst case:
 - All buckets are sent to disk ($t=0$)
 - Hybrid join becomes grace join

Hybrid Join Algorithm

Cost of Hybrid Join:

- Grace join: $3B(R) + 3B(S)$
- Hybrid join:
 - Saves 2 I/Os for t/k fraction of buckets
 - Saves $2t/k(B(R) + B(S))$ I/Os
 - Cost:
 $(3-2t/k)(B(R) + B(S)) = (3-2M/B(S))(B(R) + B(S))$

Hybrid Join Algorithm

- What is the advantage of the hybrid algorithm?

Hybrid Join Algorithm

- What is the advantage of the hybrid algorithm?

It degrades gracefully when S larger than M :

- When $B(S) \leq M$
 - Main memory hash-join has cost $B(R) + B(S)$
- When $B(S) > M$
 - Grace-join has cost $3B(R) + 3B(S)$
 - Hybrid join has cost $(3-2t/k)(B(R) + B(S))$

Summary of External Join Algorithms

- Block Nested Loop: $B(S) + B(R) \cdot B(S) / M$
- Index Join: $B(R) + T(R)B(S) / V(S, a)$
- Partitioned Hash: $3B(R) + 3B(S)$;
 - $\min(B(R), B(S)) \leq M^2$
- Merge Join: $3B(R) + 3B(S)$
 - $B(R) + B(S) \leq M^2$

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?