

CSE 444: Database Internals

Lecture 8 Operator Algorithms (part 2)

Announcements

- Lab 1 due tonight at 11pm
- Make sure your code builds on attu!
 - Log in to attu
`git clone <your directory>`
`cd <your directory>`
`ant test-report`
 - Errors mean that you may have committed extra files to GitLab, or other IDE specific problems

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)
```

What is the Cost?

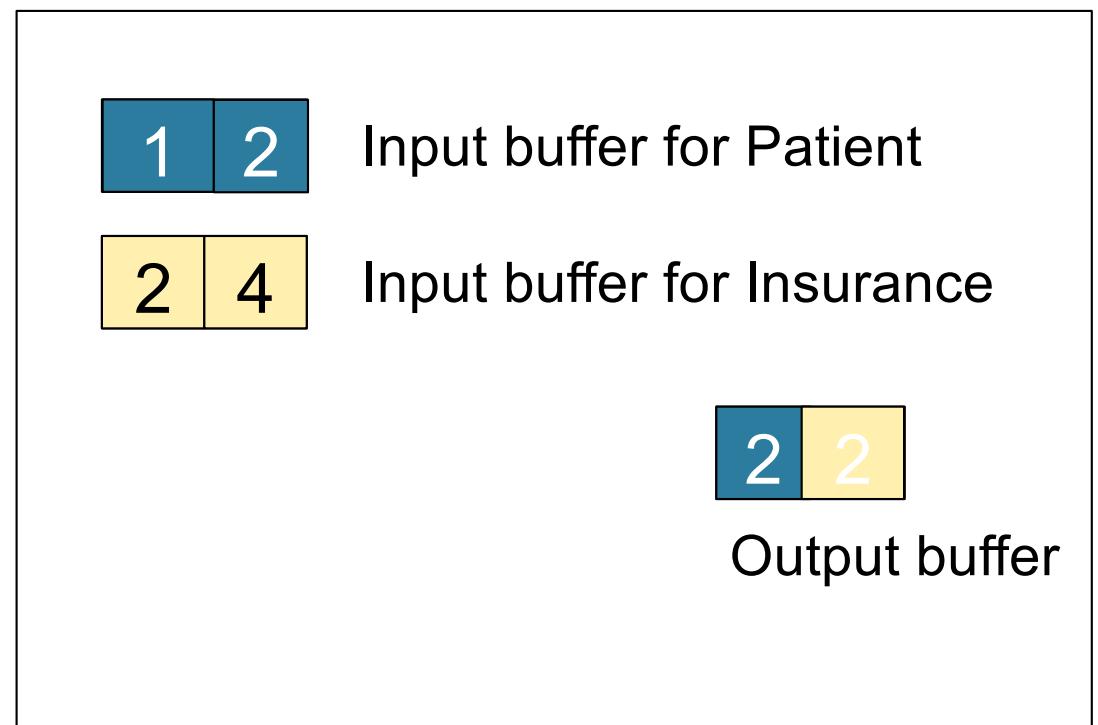
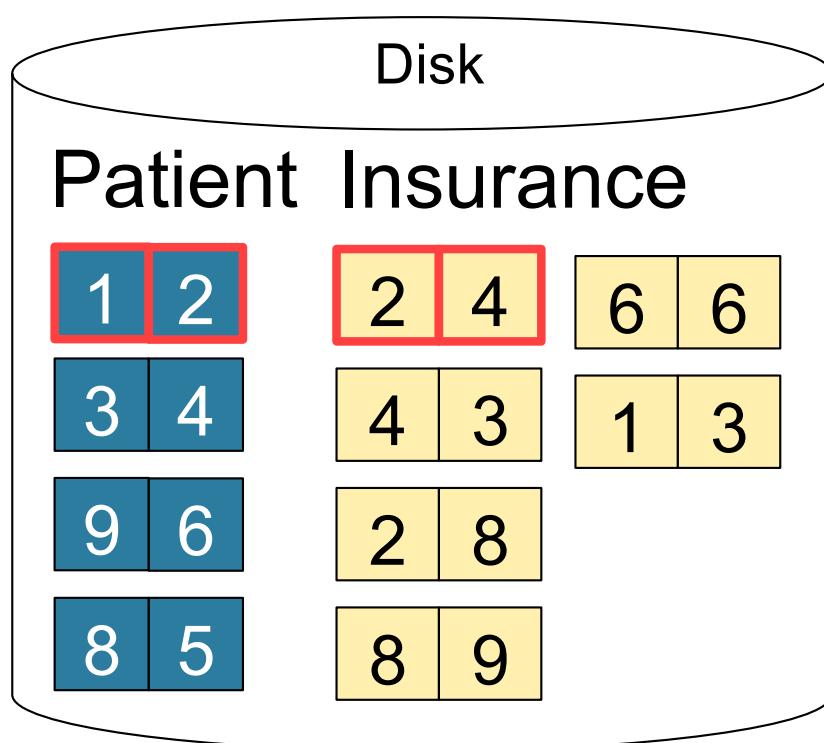
Page-at-a-time Refinement

```
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)
```

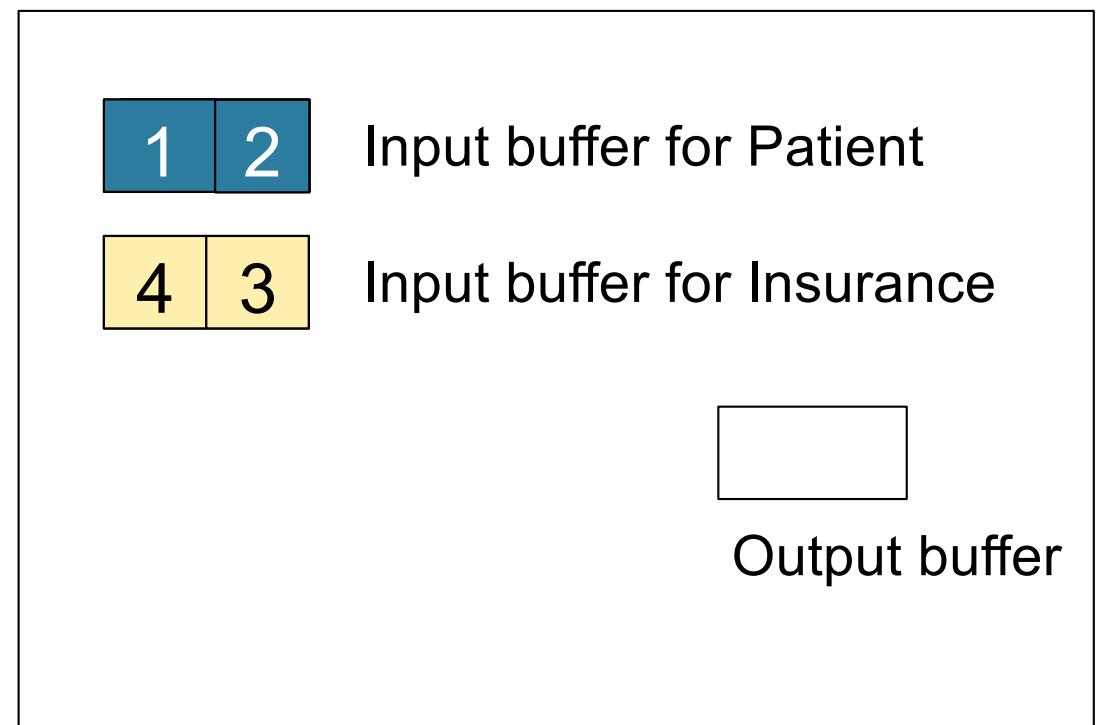
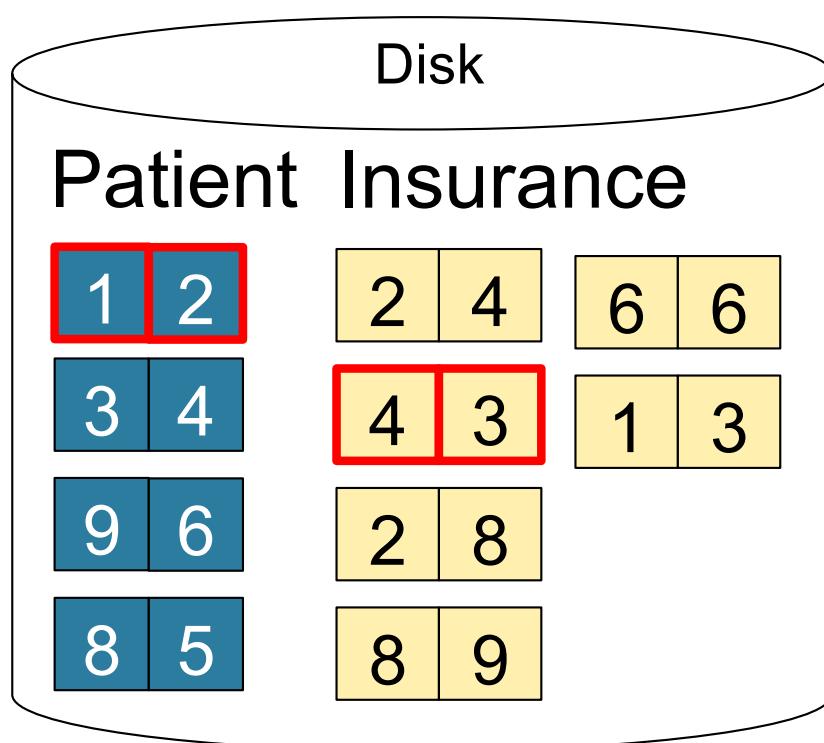
- Cost: $B(R) + B(R)B(S)$

What is the Cost?

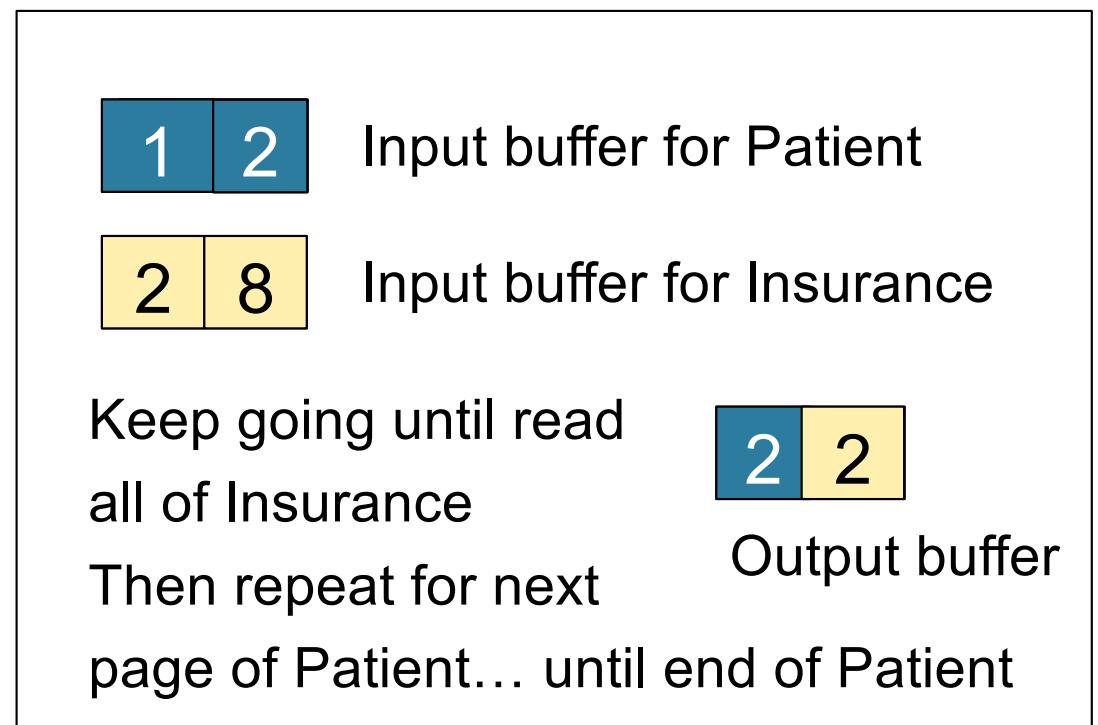
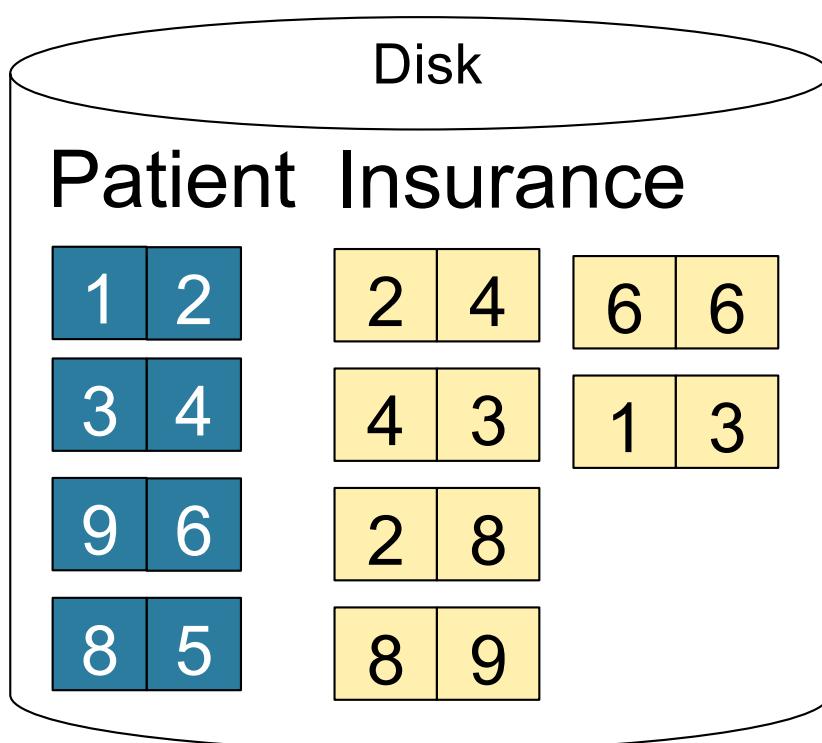
Page-at-a-time Refinement



Page-at-a-time Refinement



Page-at-a-time Refinement



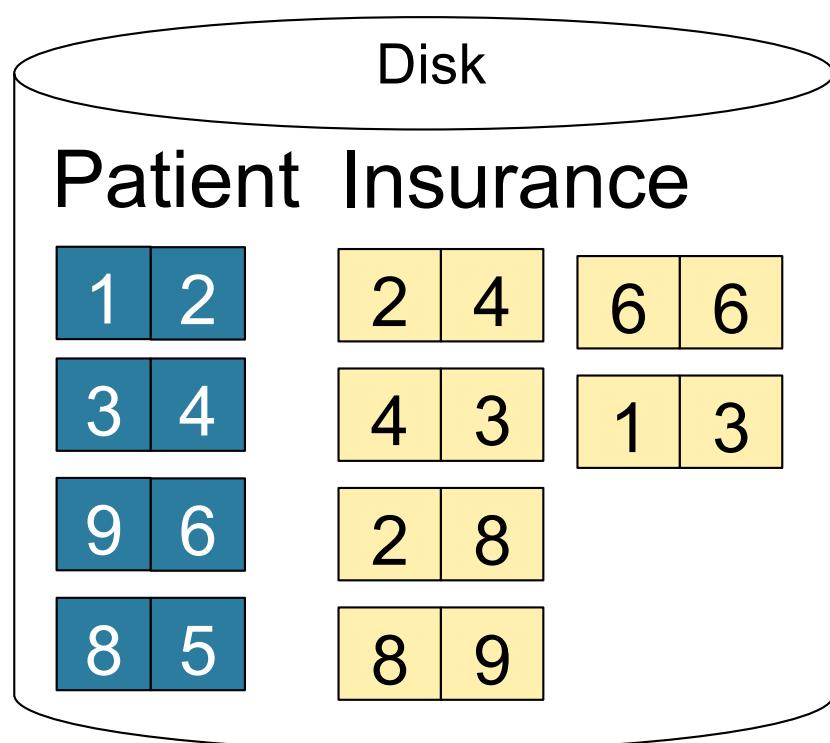
Cost: $B(R) + B(R)B(S)$

Block-Nested-Loop Refinement

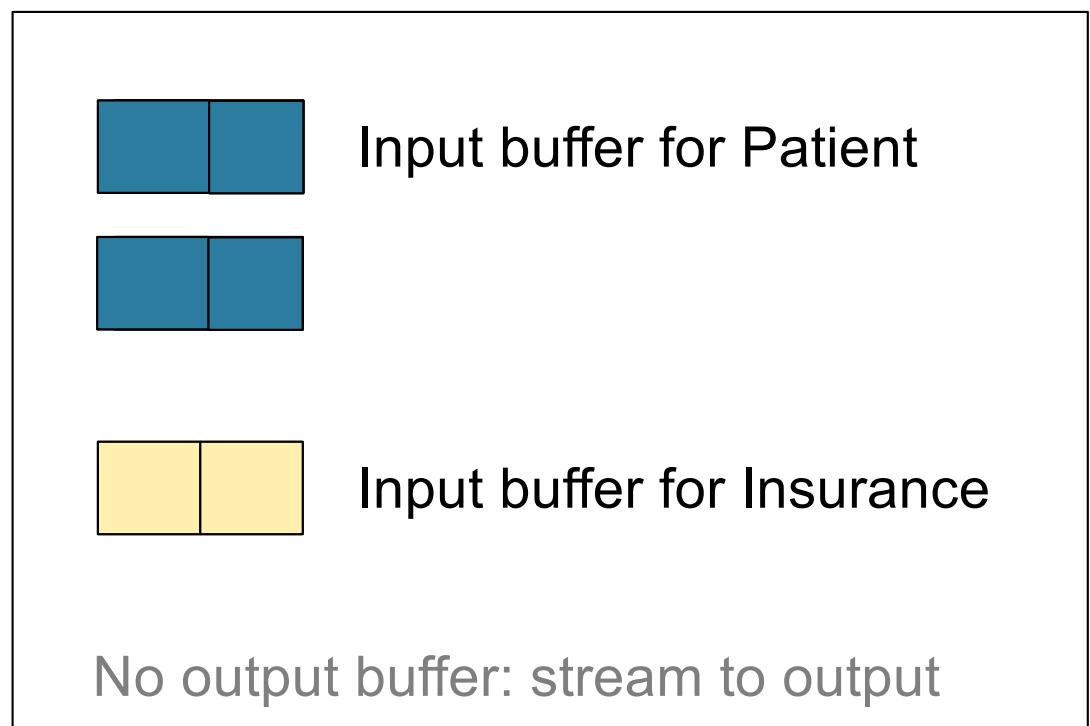
```
for each group of M-1 pages r in R do  
  for each page of tuples s in S do  
    for all pairs of tuples t1 in r, t2 in s  
      if t1 and t2 join then output (t1,t2)
```

What is the Cost?

Block Memory Refinement

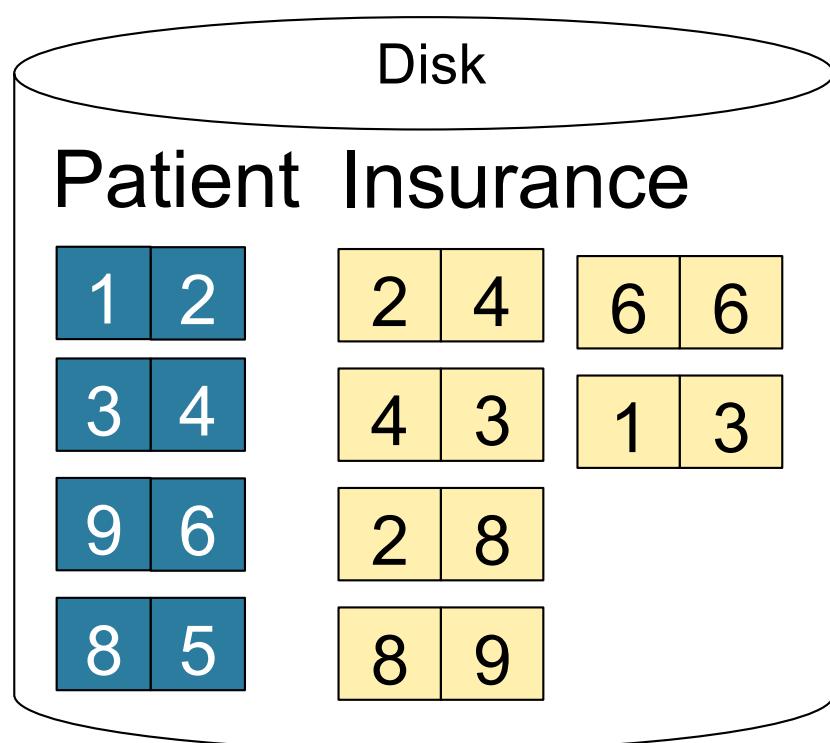


M= 3

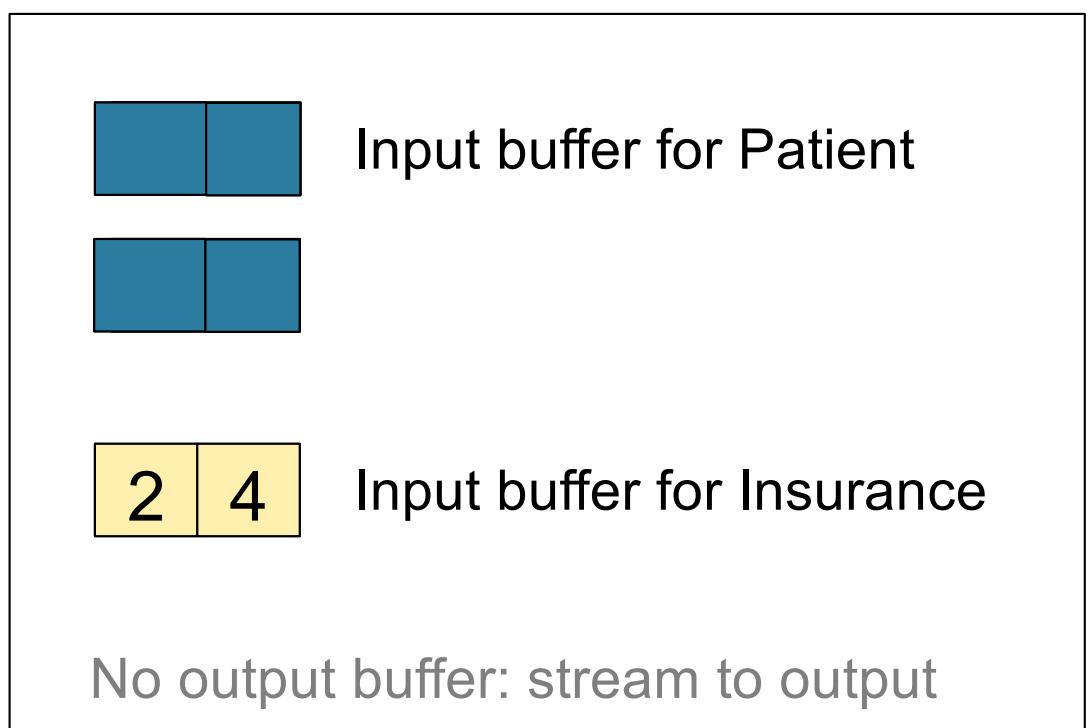


Cost:

Block Memory Refinement



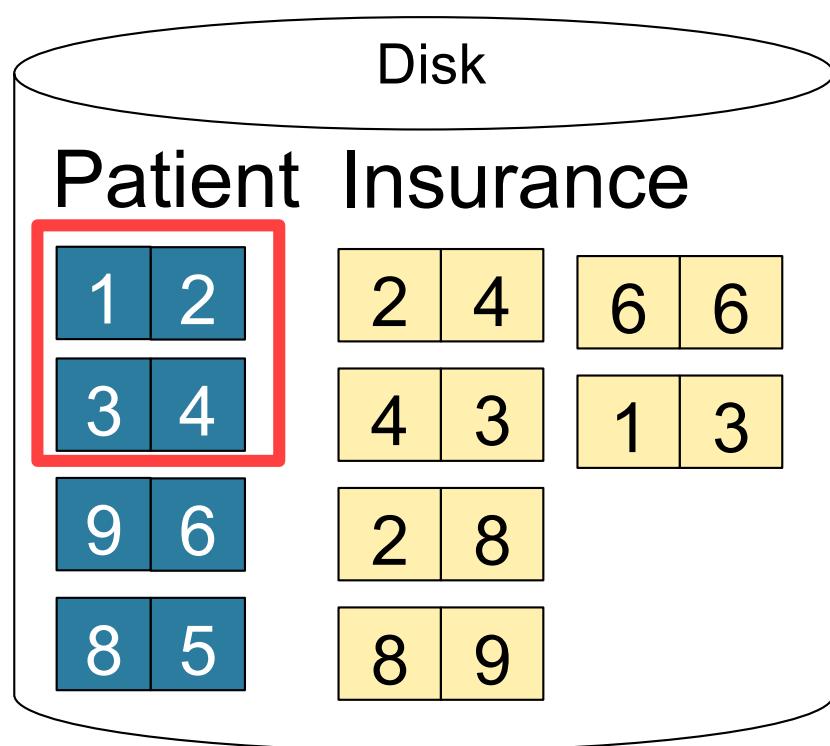
M= 3



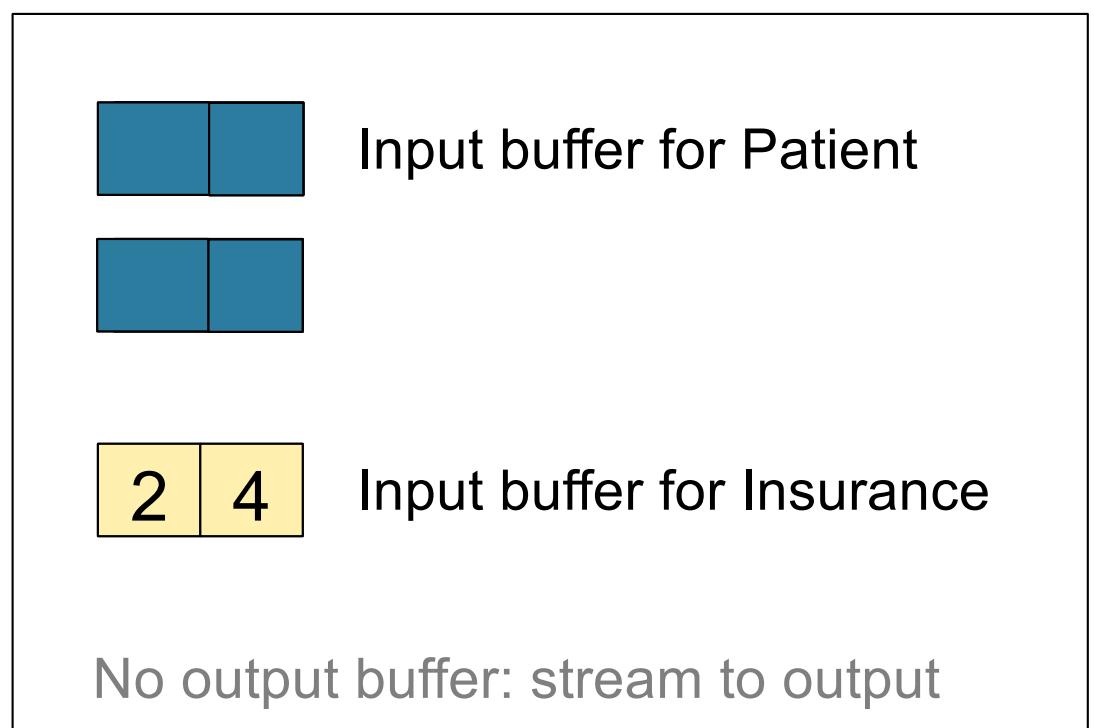
Cost:

10

Block Memory Refinement

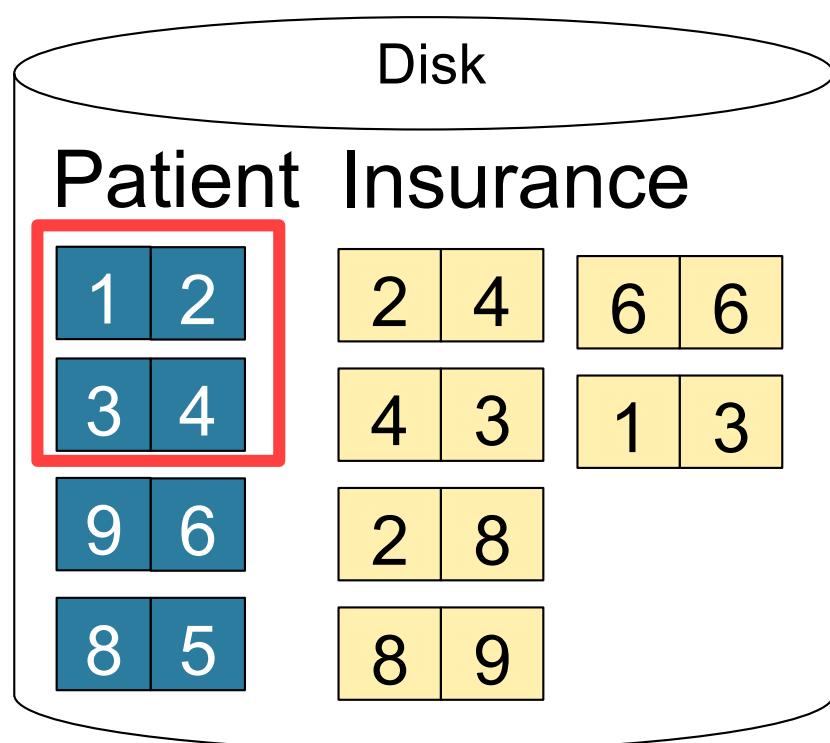


M= 3

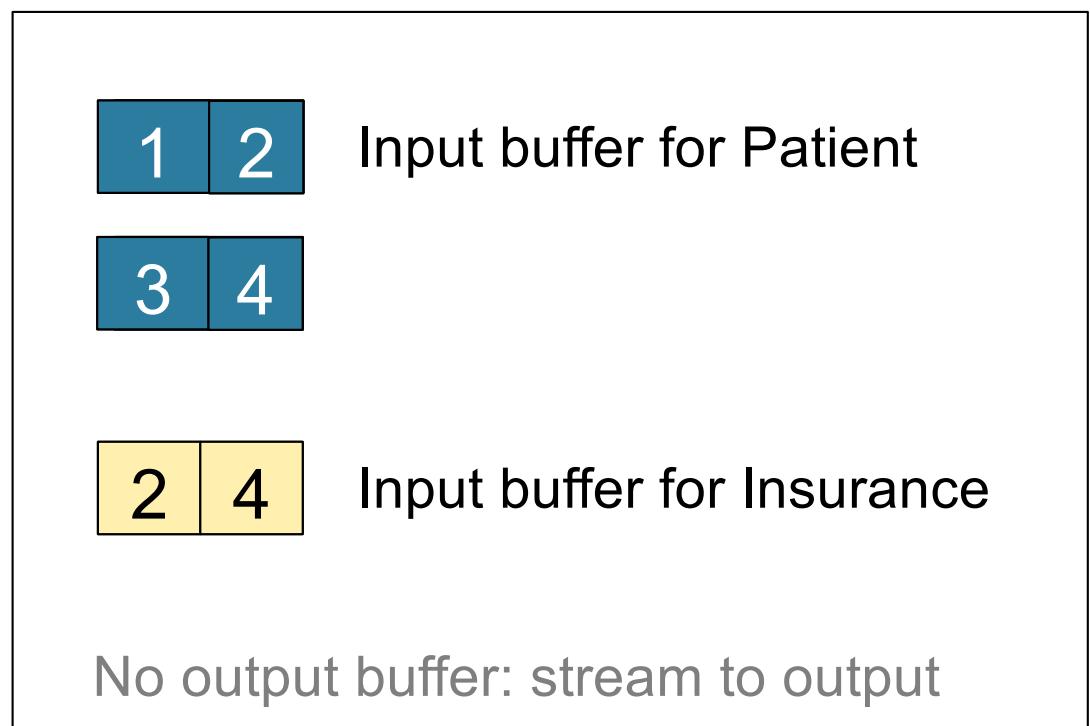


Cost:

Block Memory Refinement

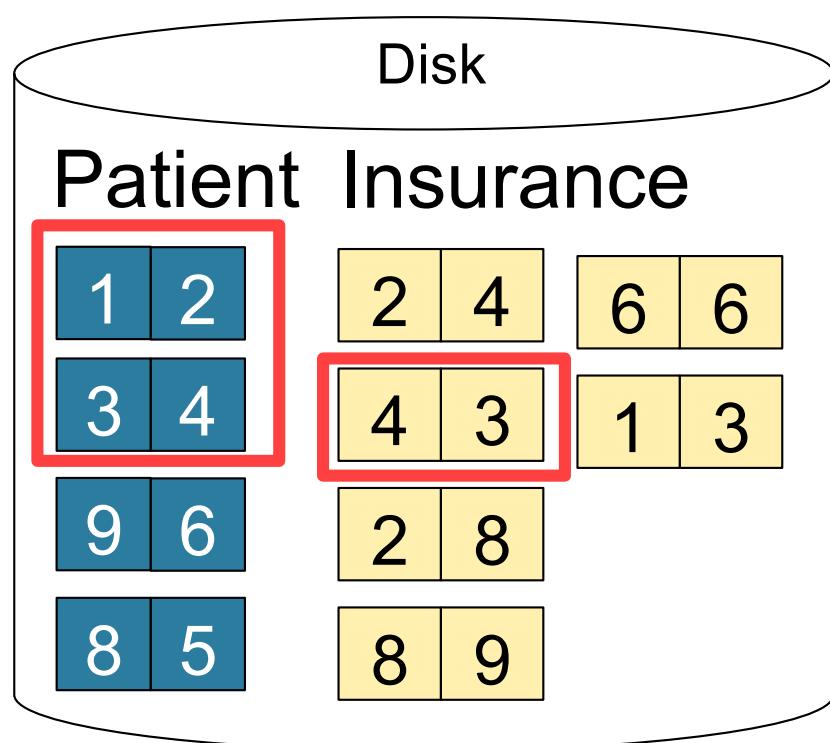


M= 3

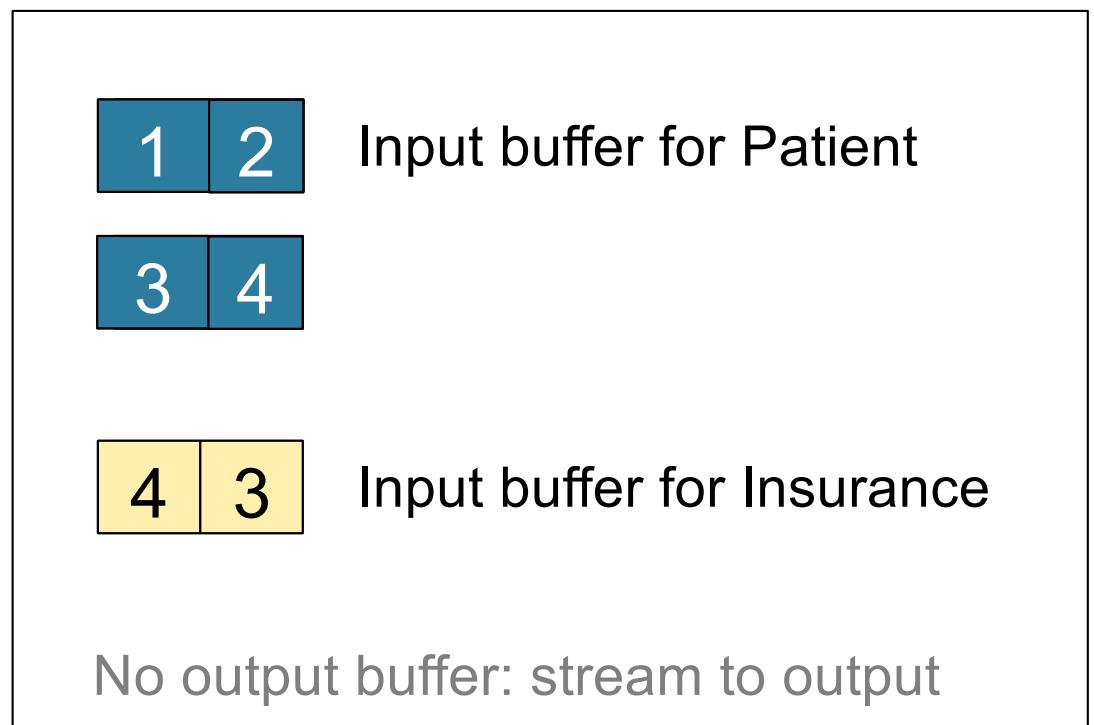


Cost:

Block Memory Refinement

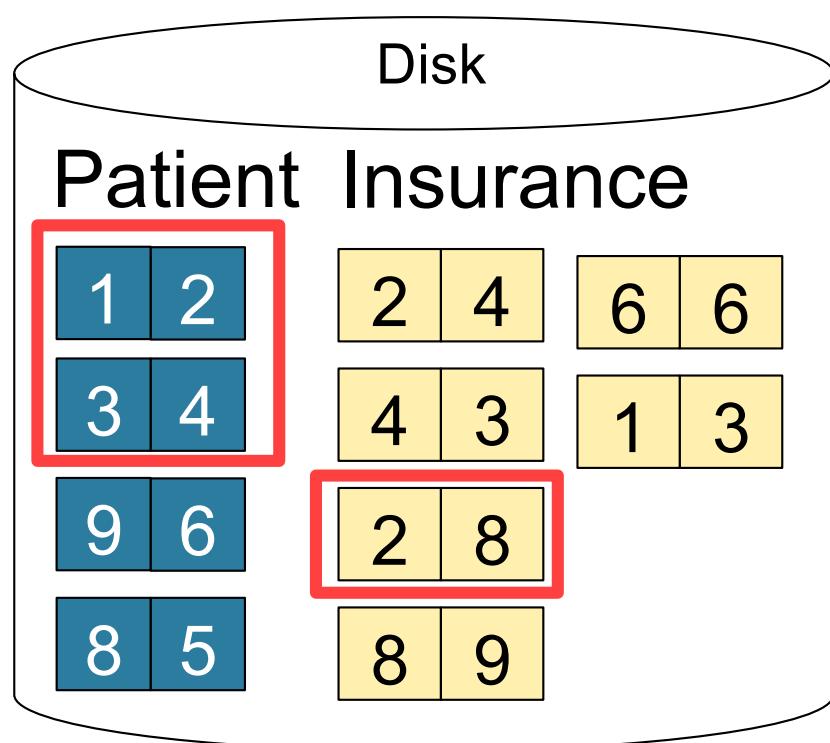


M= 3

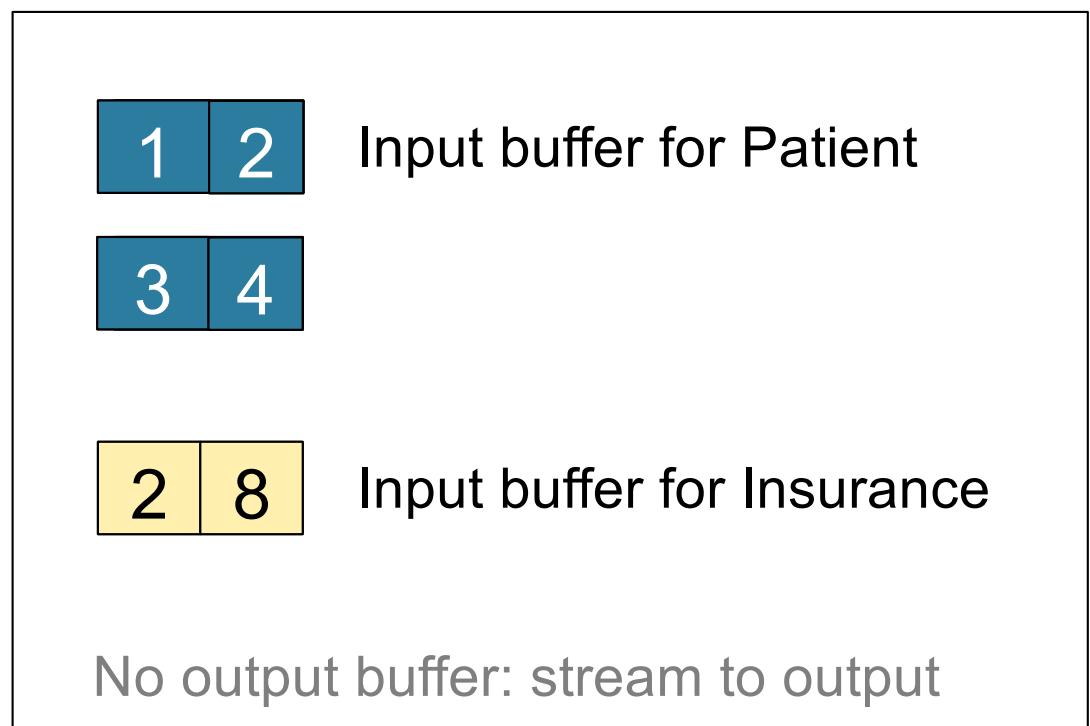


Cost:

Block Memory Refinement

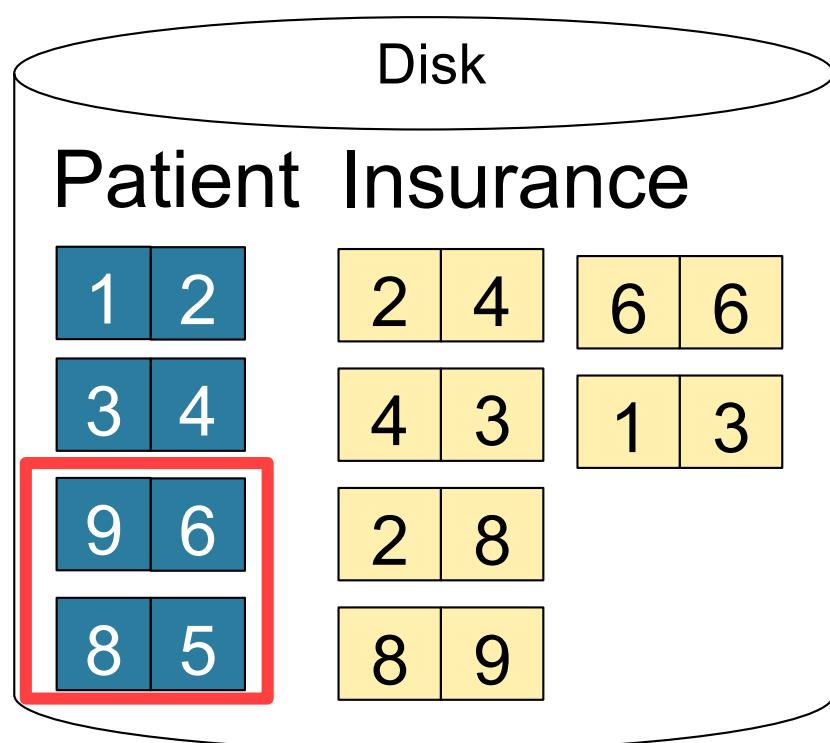


M= 3

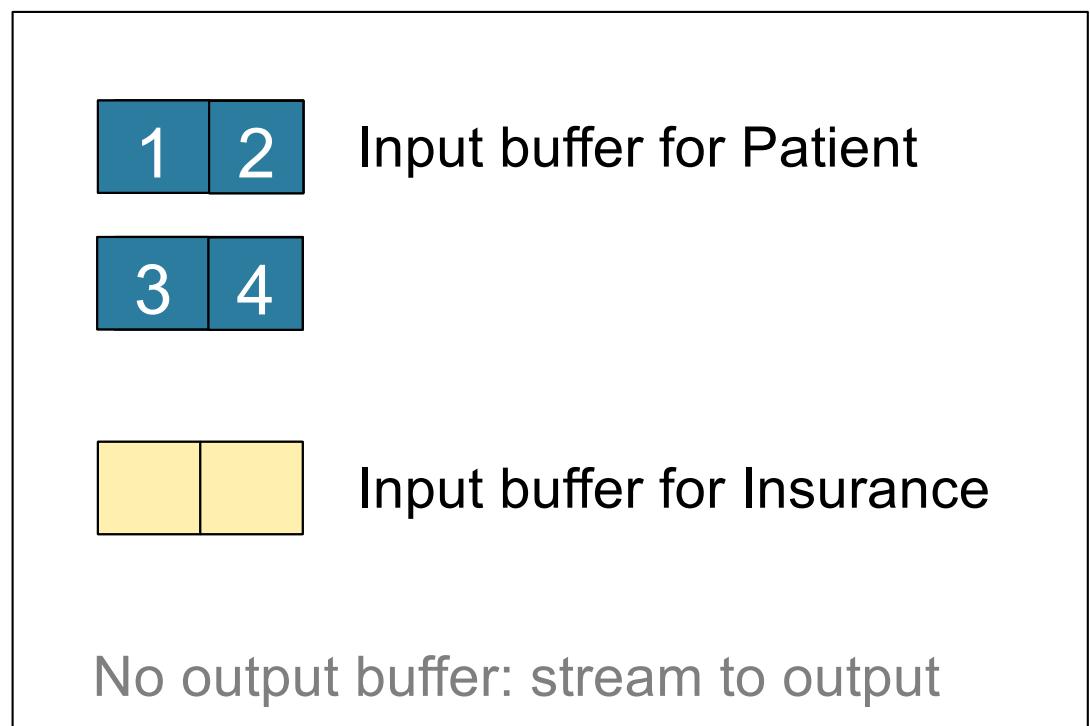


Cost:

Block Memory Refinement



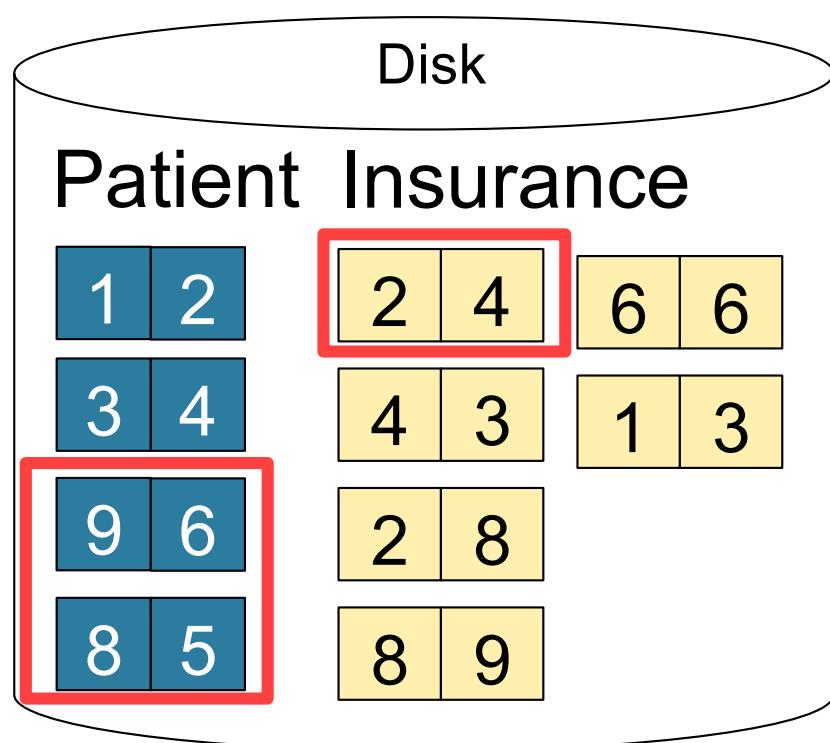
M= 3



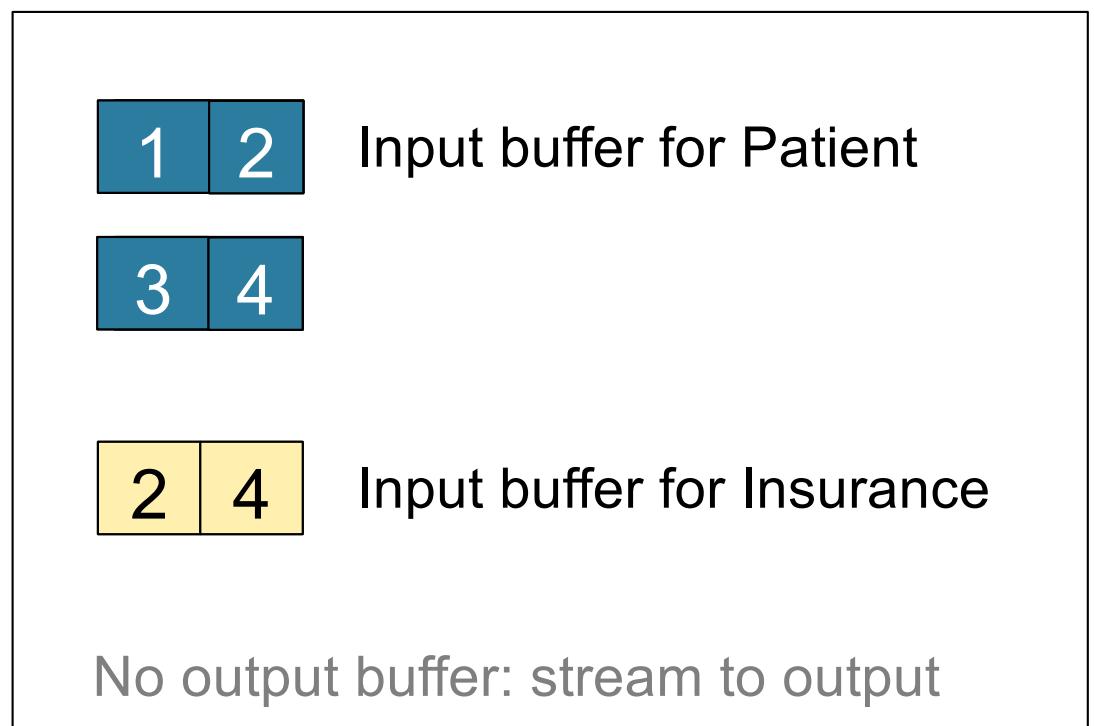
Cost:

15

Block Memory Refinement



M= 3



Cost:

Block-Nested-Loop 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)
```

What is the Cost?

Block-Nested-Loop 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?

Sort-Merge Join

Sort-merge join: $R \bowtie S$

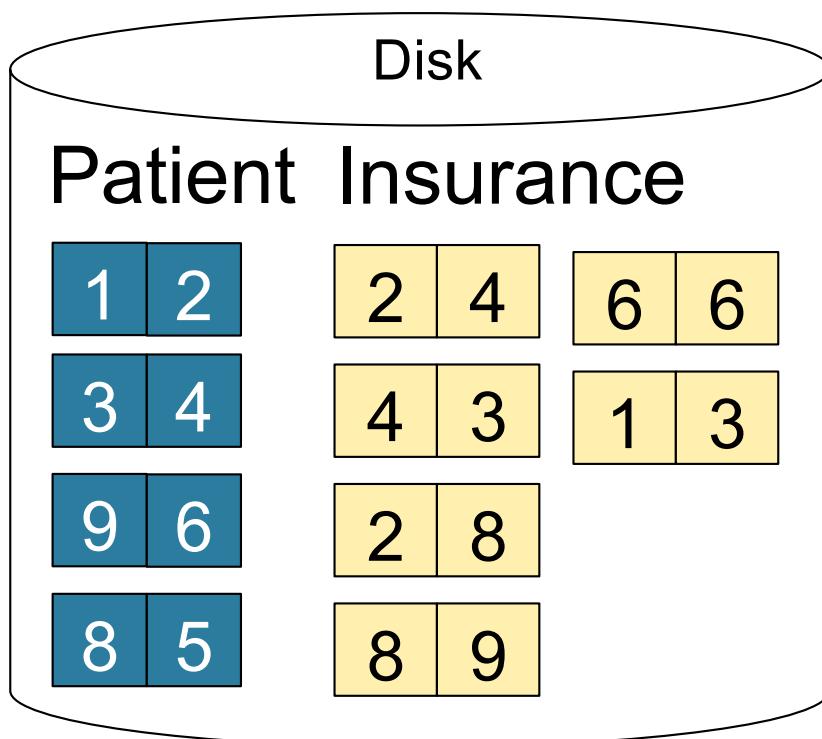
- Scan R and sort in main memory
 - Scan S and sort in main memory
 - Merge R and S
-
- Cost: $B(R) + B(S)$
 - One pass algorithm when $B(S) + B(R) \leq M$
 - Typically, this is NOT a one pass algorithm

Sort-Merge Join Example

Step 1: Scan Patient and **sort** in memory

Memory M = 21 pages

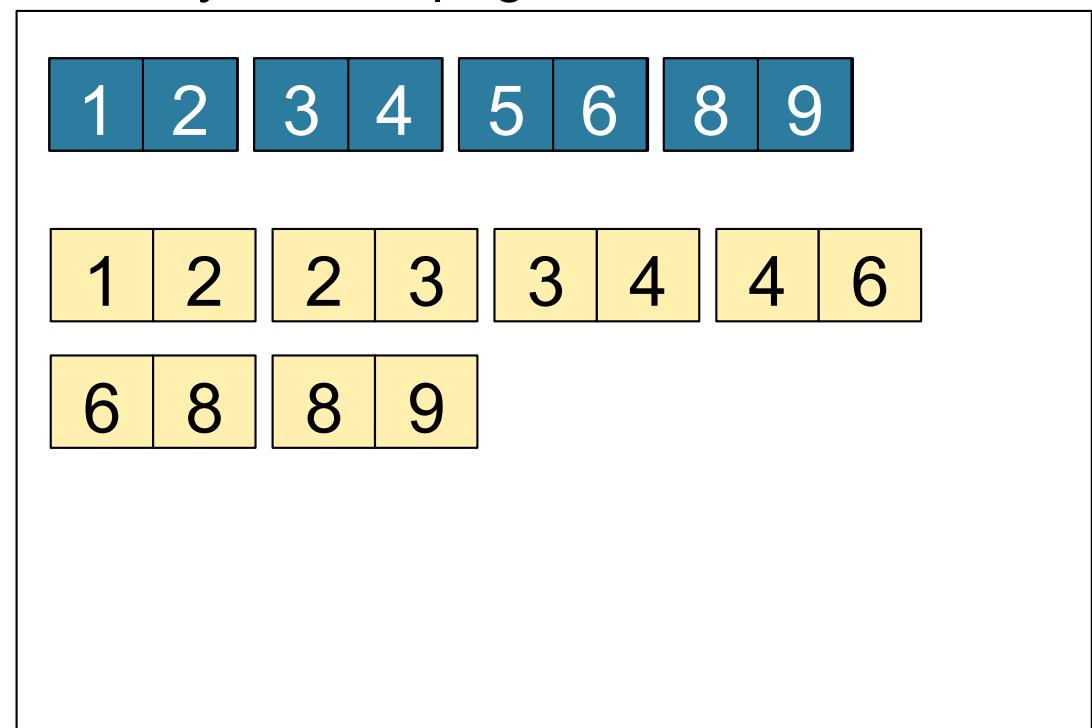
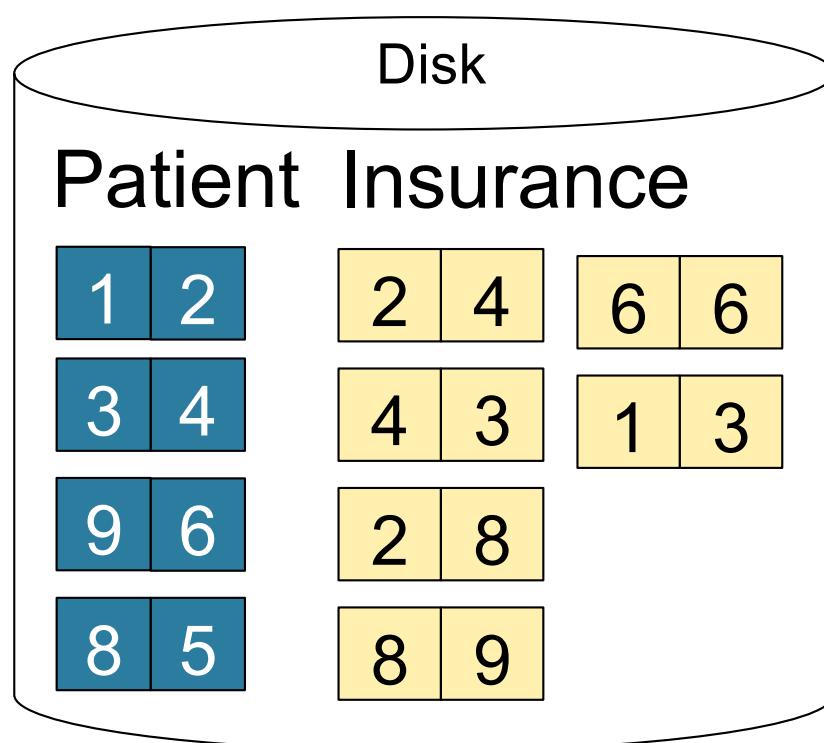
1	2	3	4	5	6	8	9
---	---	---	---	---	---	---	---



Sort-Merge Join Example

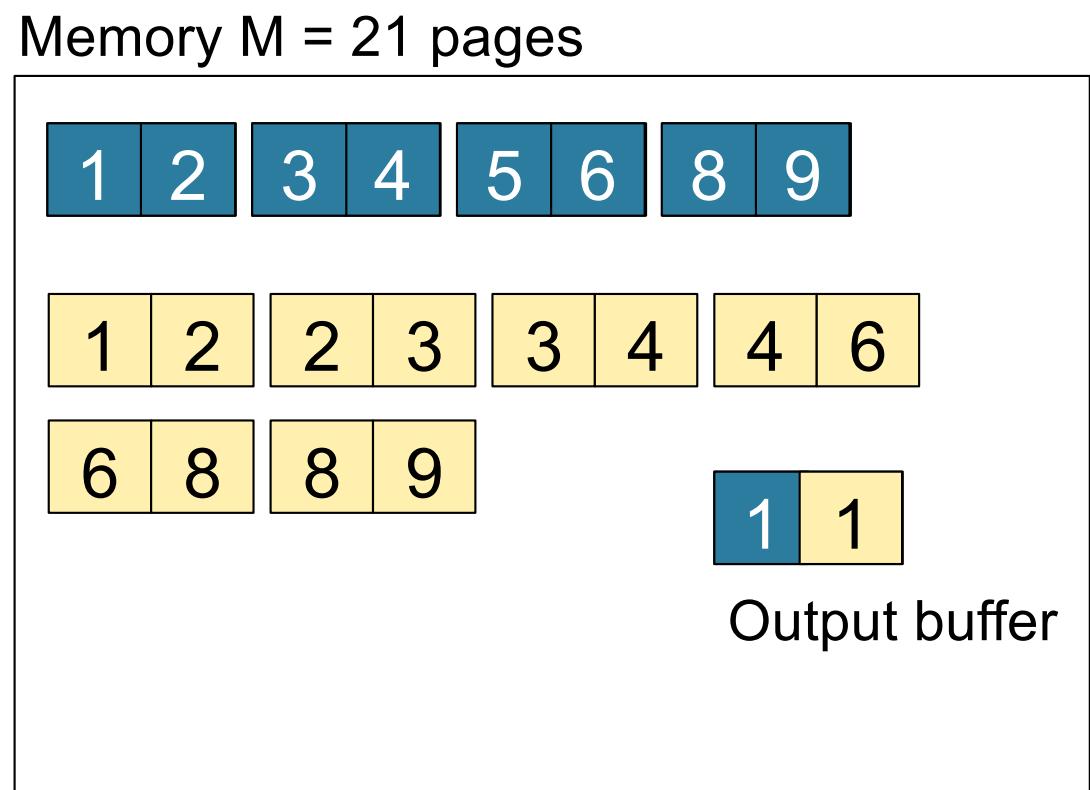
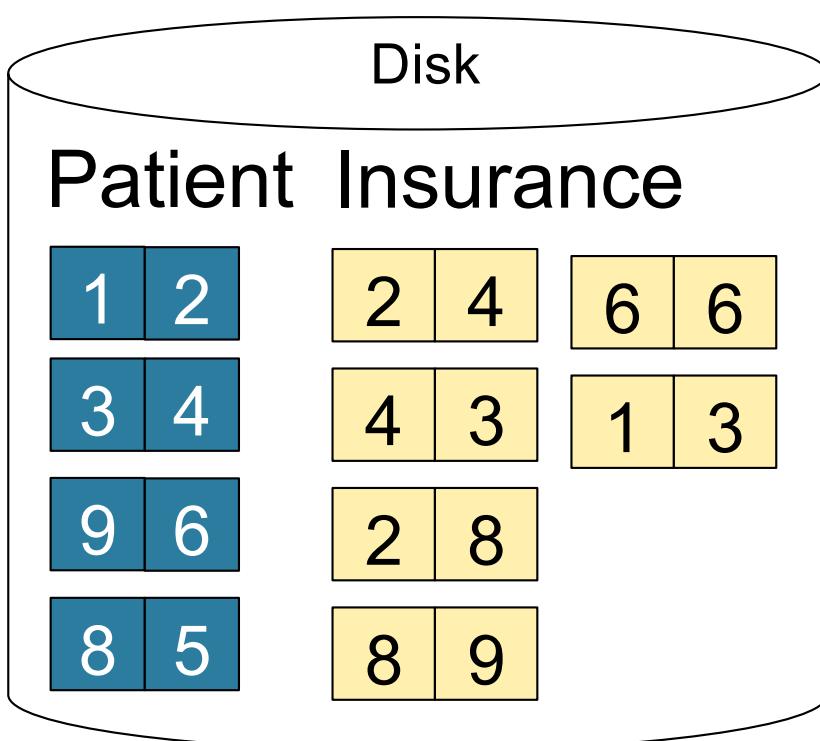
Step 2: Scan Insurance and **sort** in memory

Memory M = 21 pages



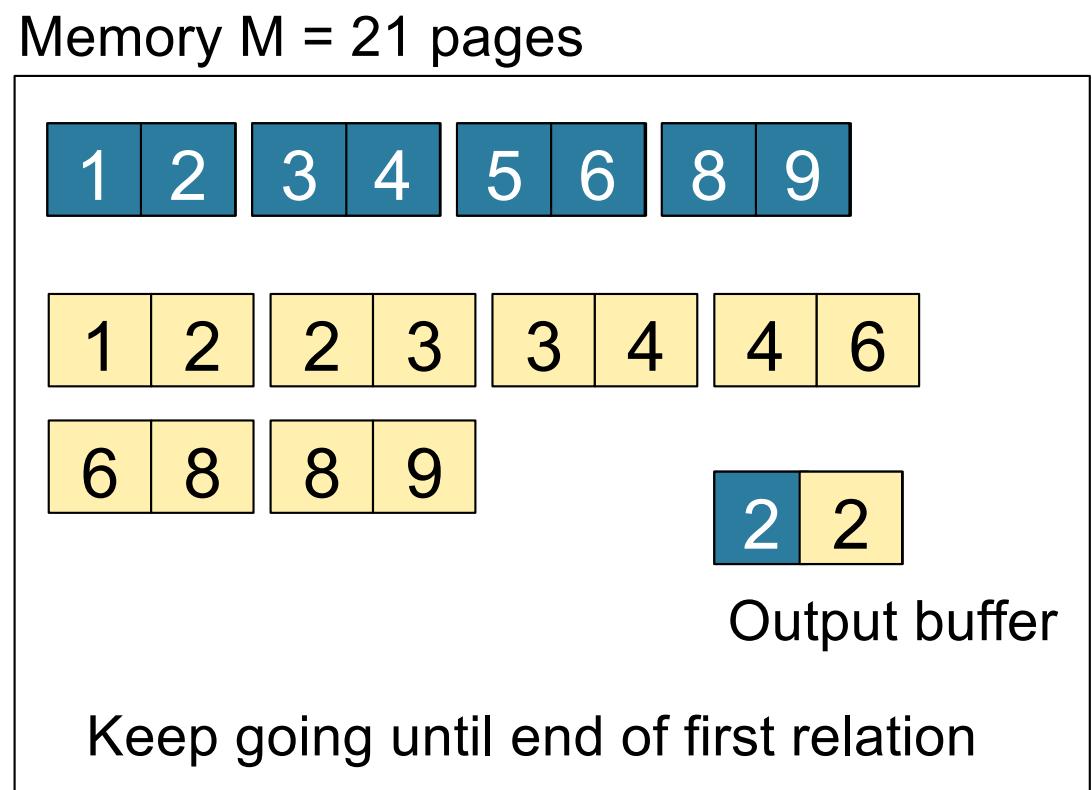
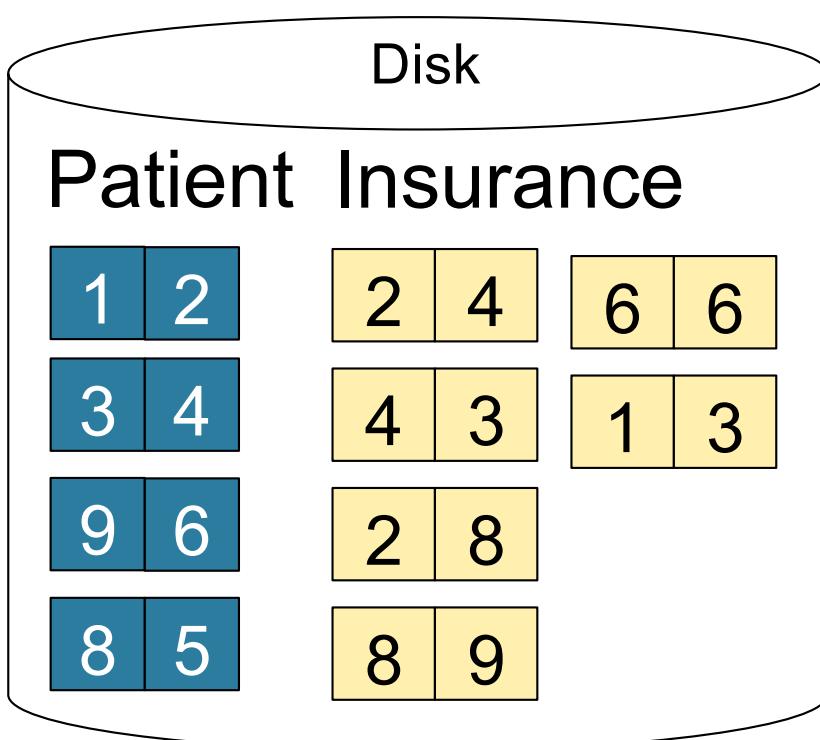
Sort-Merge Join Example

Step 3: **Merge** Patient and Insurance



Sort-Merge Join Example

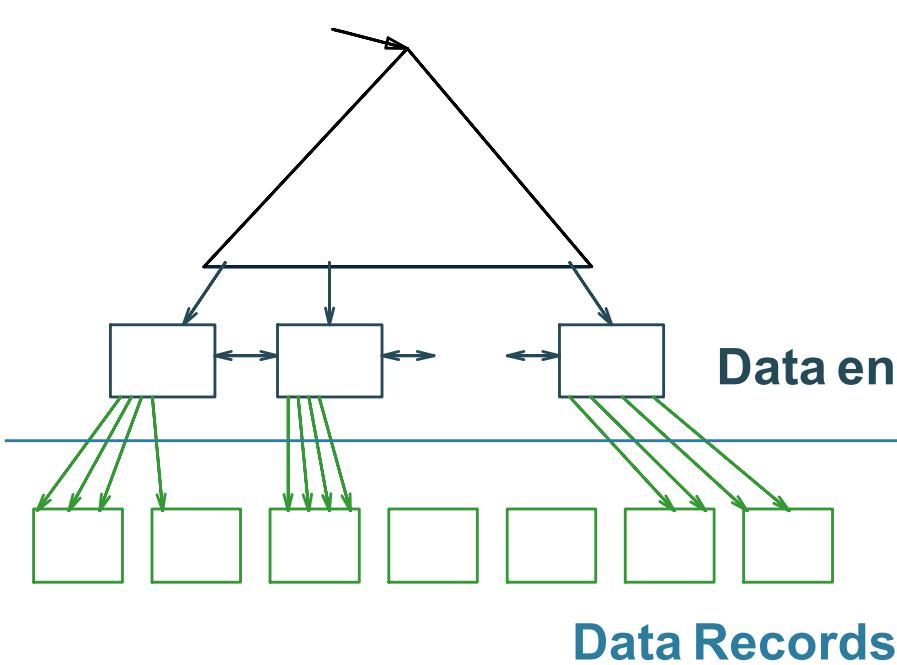
Step 3: **Merge** Patient and Insurance



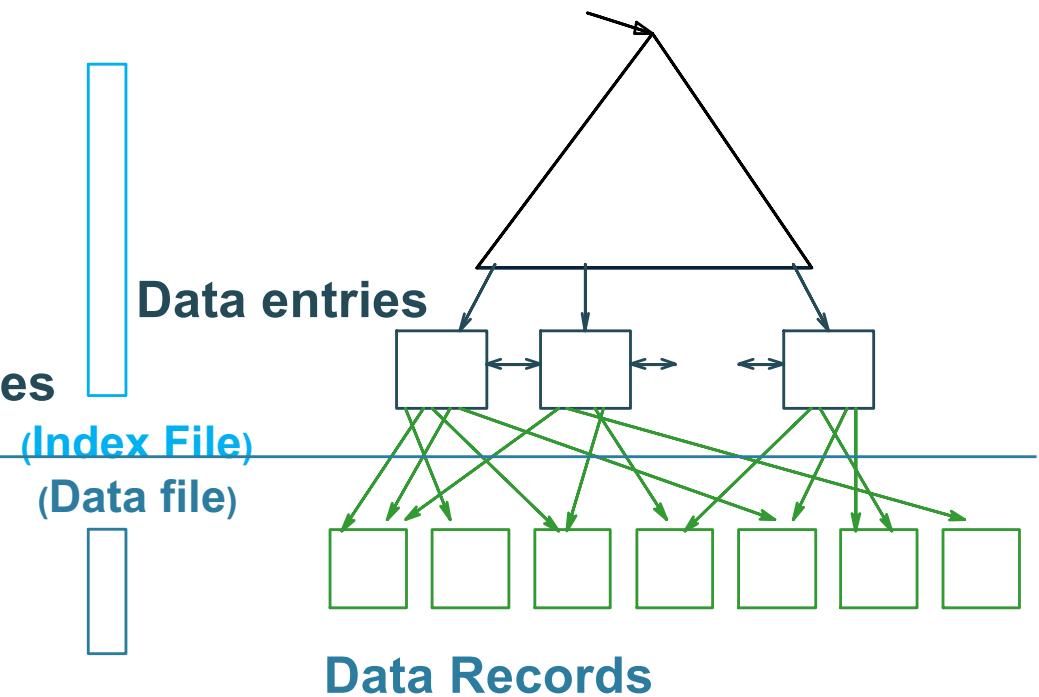
Outline

- **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)

B+ Trees



CLUSTERED



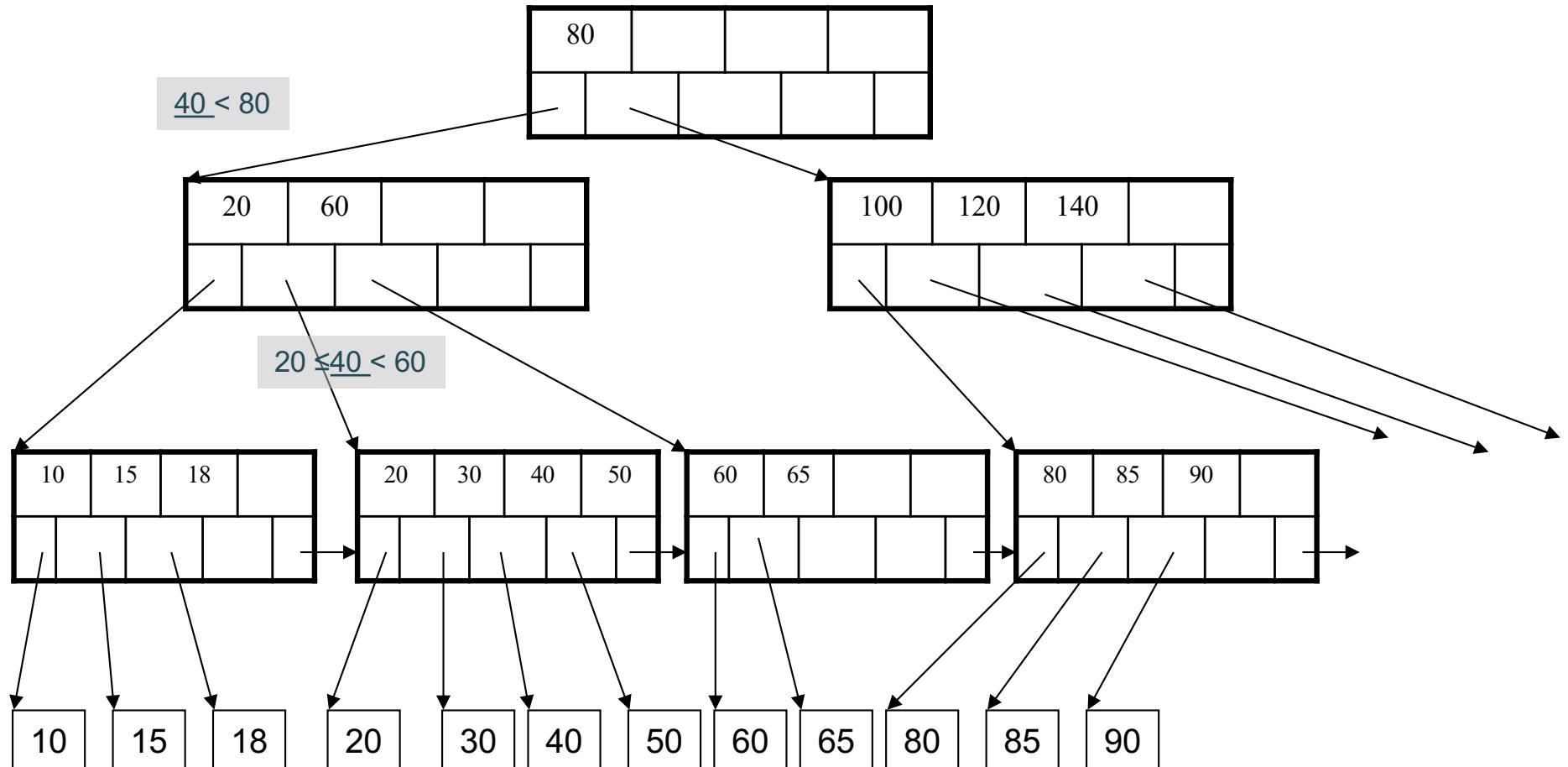
UNCLUSTERED

Note: can also store data records directly as data entries

B+ Tree Example

$d = 2$

Find the key 40



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 :

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:

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

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

- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:

Index Based Selection

- Example:
 - $B(R) = 2000$
 - $T(R) = 100,000$
 - $V(R, a) = 20$
- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:
 - If index is clustered:
 - If index is unclustered:

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

Index Based Selection

- Example:

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

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

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:

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

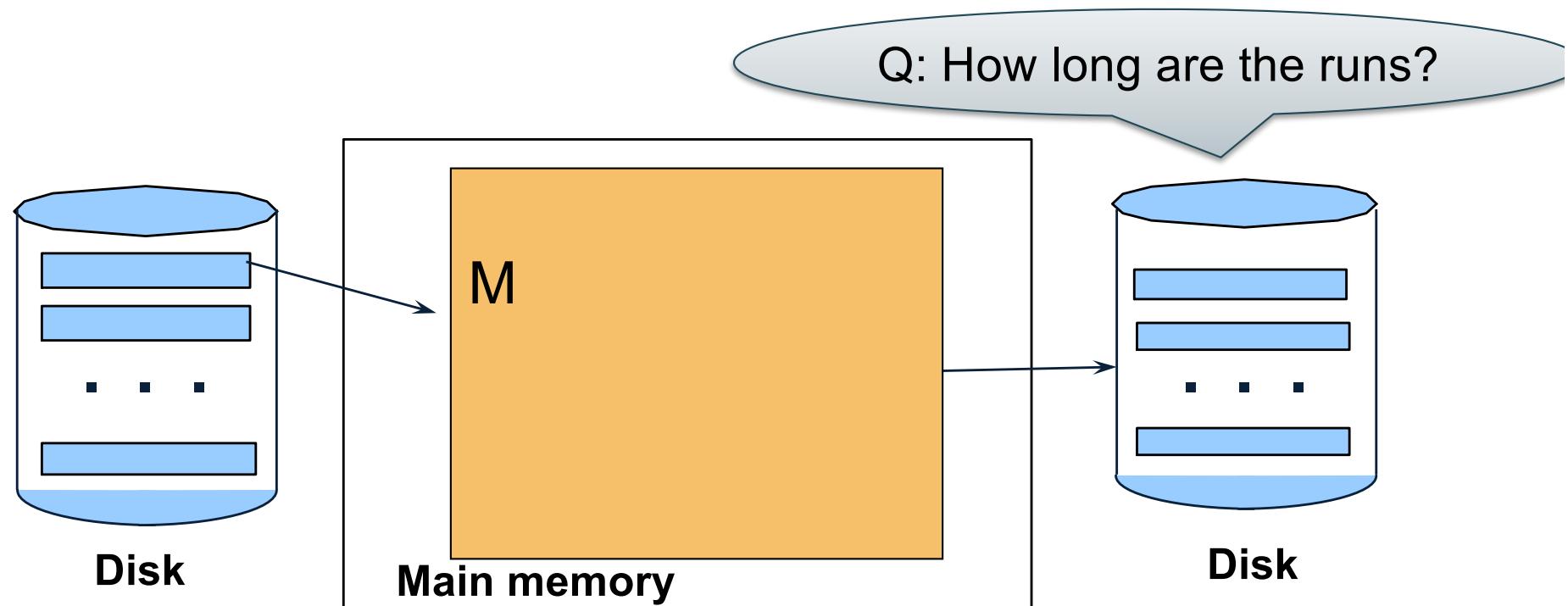
2, 4, 99, 103, | 88, | 77, | 3, 79, 100, | 2, 50

External Merge-Sort: Step 1

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

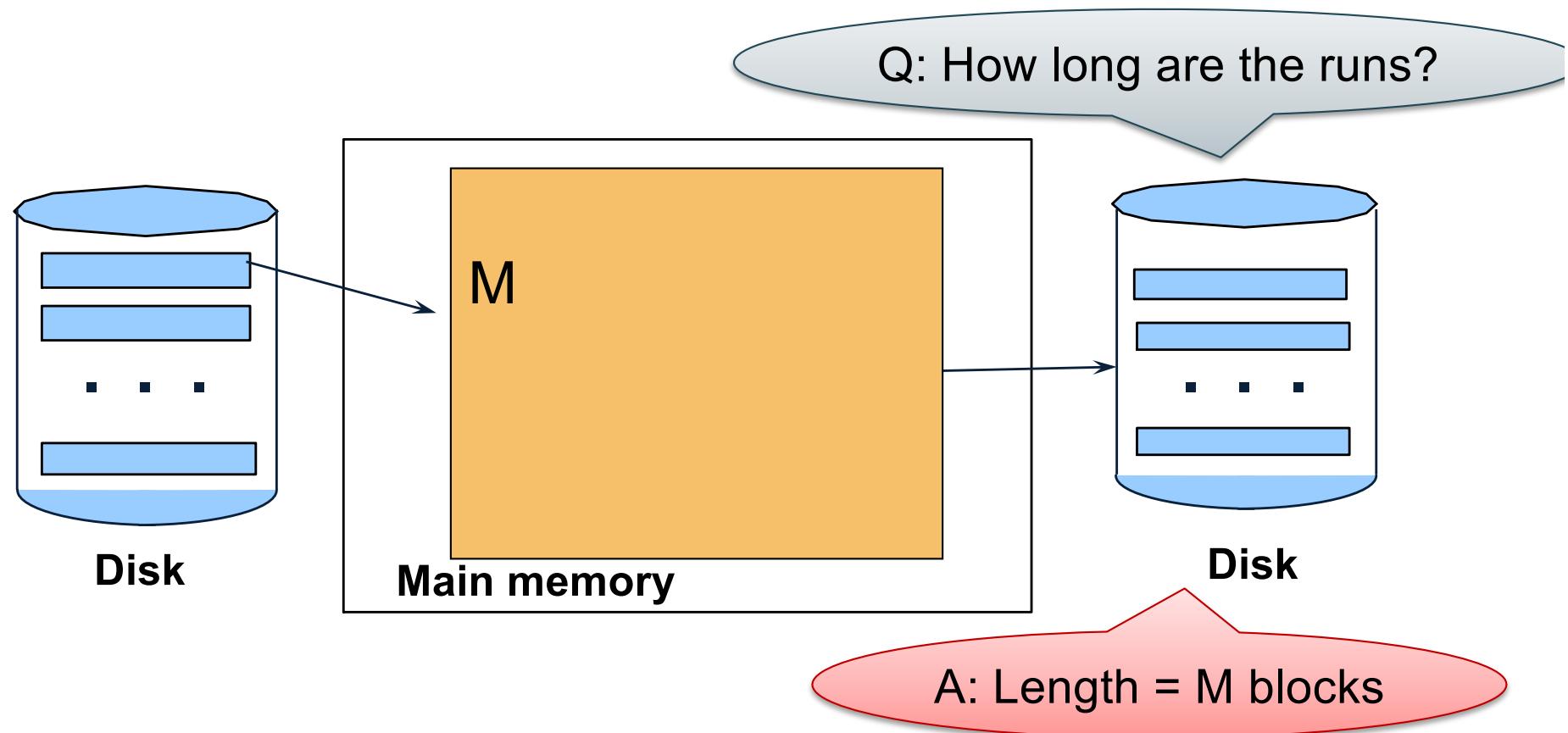
External Merge-Sort: Step 1

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



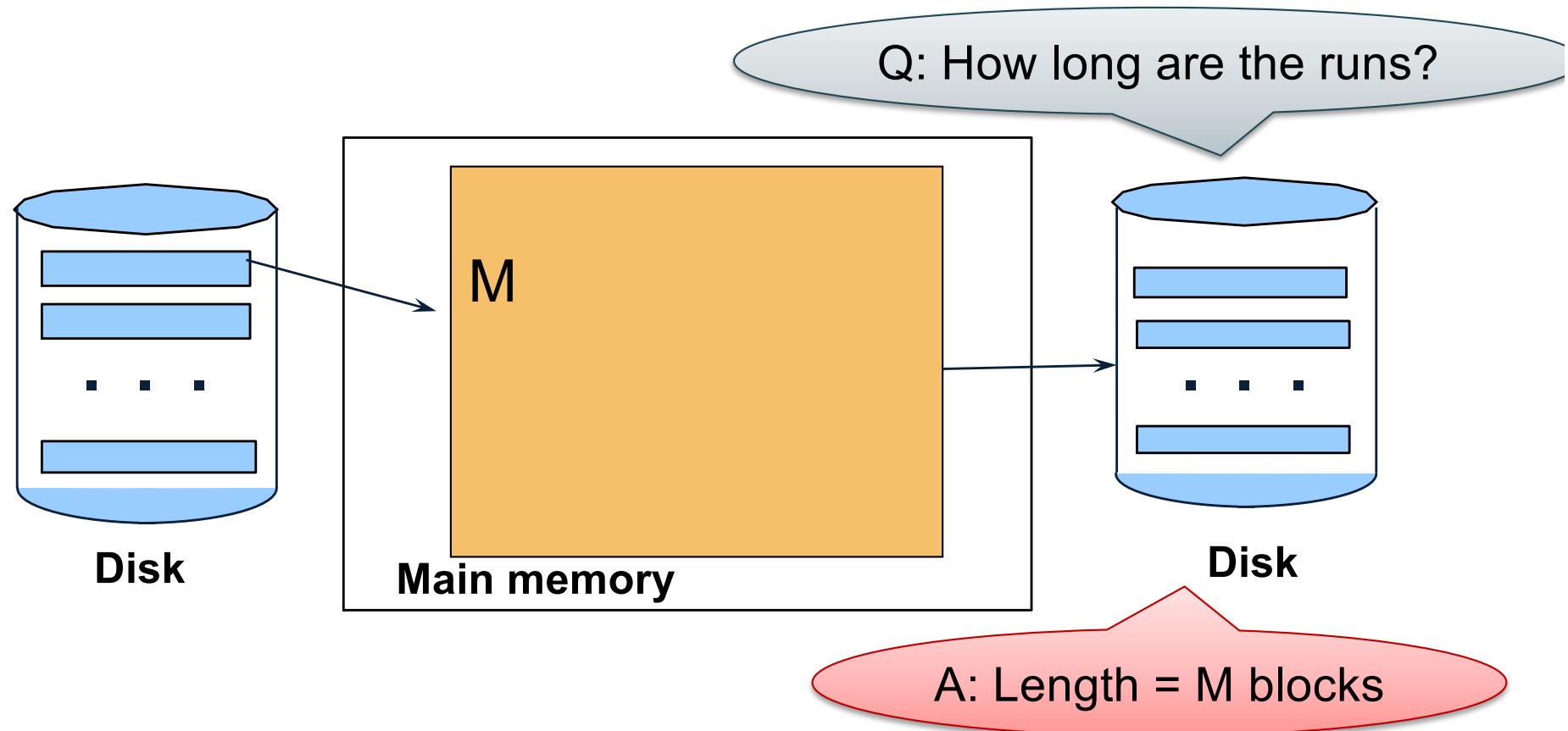
External Merge-Sort: Step 1

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

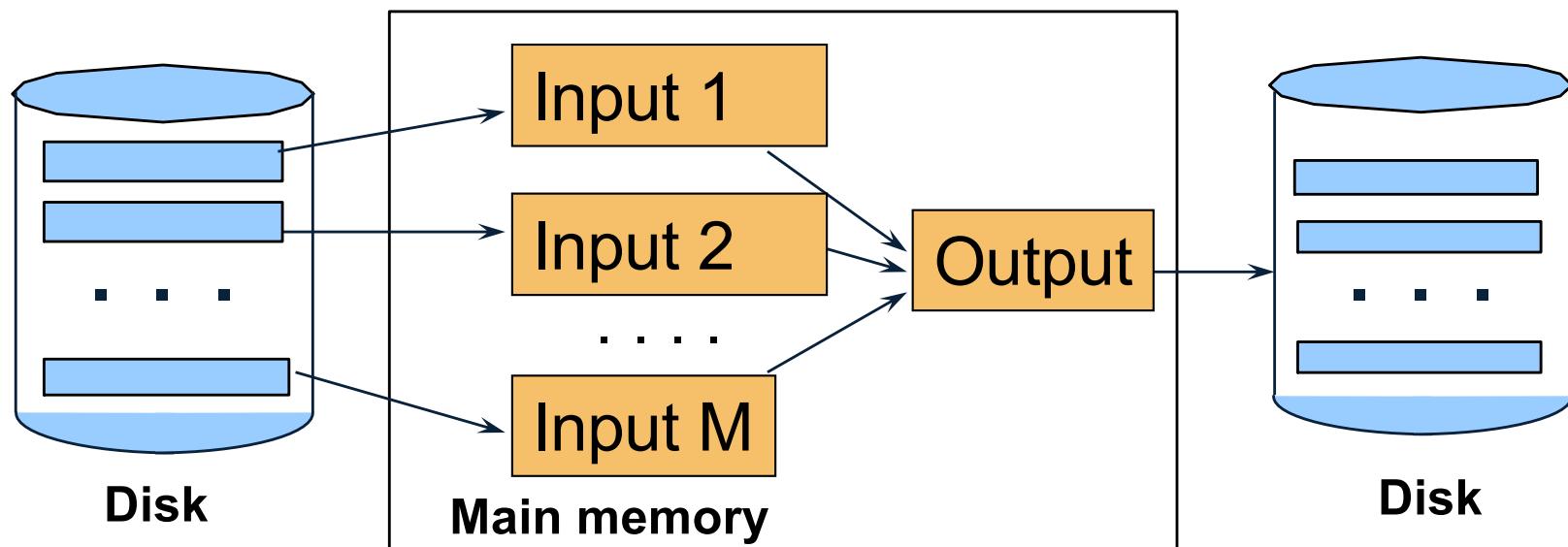


Sidenote: Can increase to length $2M$ using “replacement selection”
(details in book)

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 ($M - 1 \approx M^2$)



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

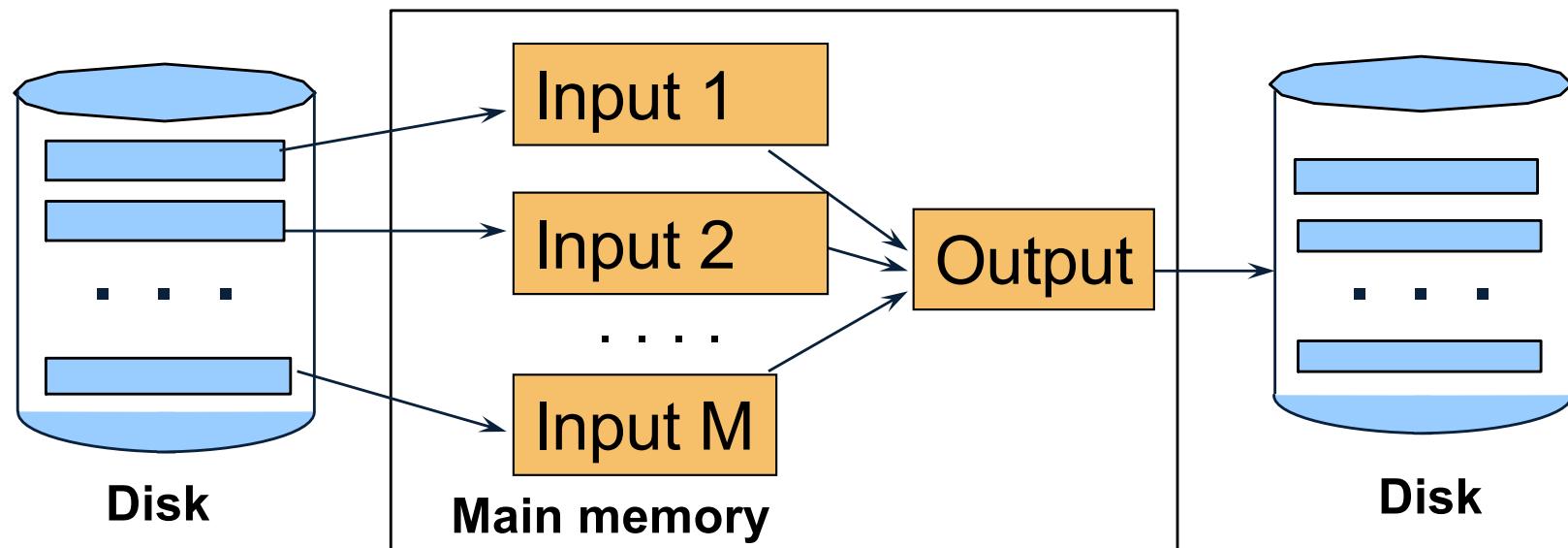
Output:

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 $R \bowtie S$

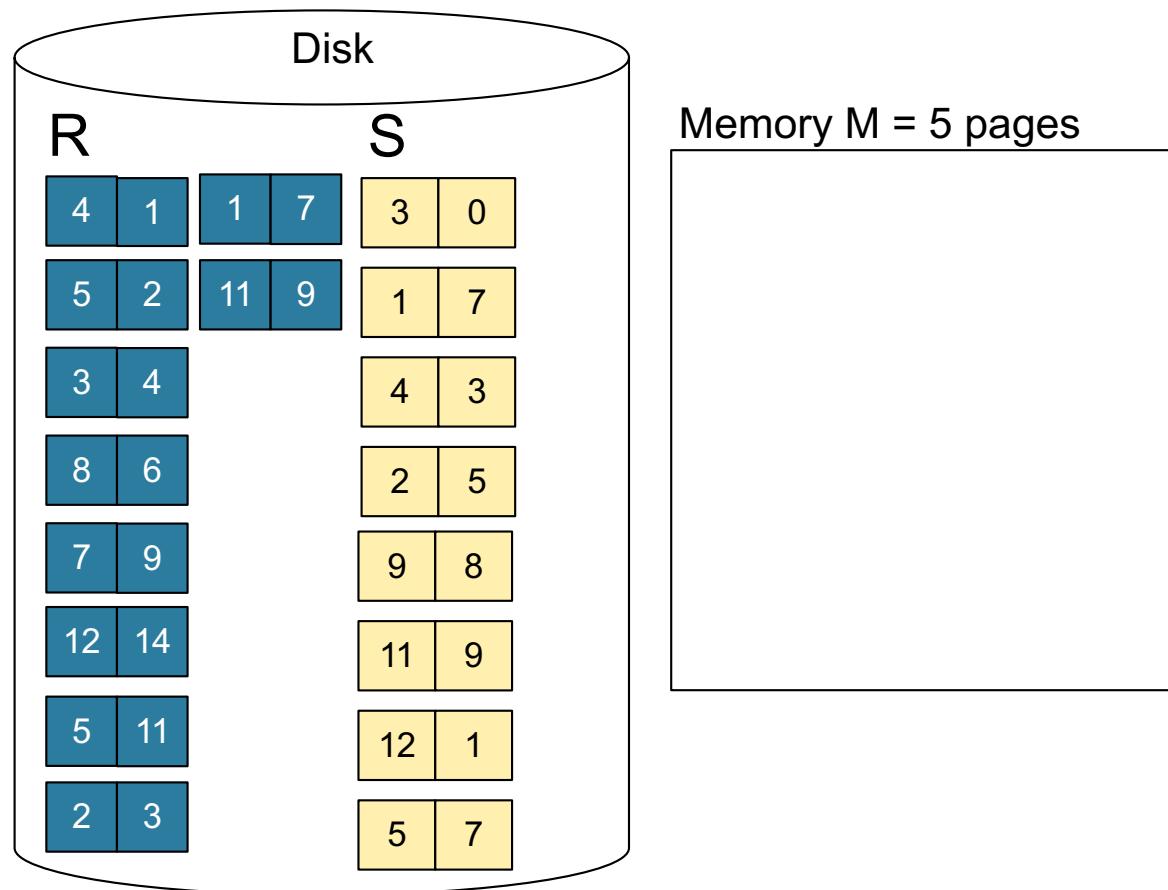
- Step 1a: generate initial runs for R
- Step 1b: generate initial runs for S
- Step 2: merge and join
 - Either merge first and then join
 - Or merge & join at the same time

Merge-Join Example

Setup: Want to join R and S

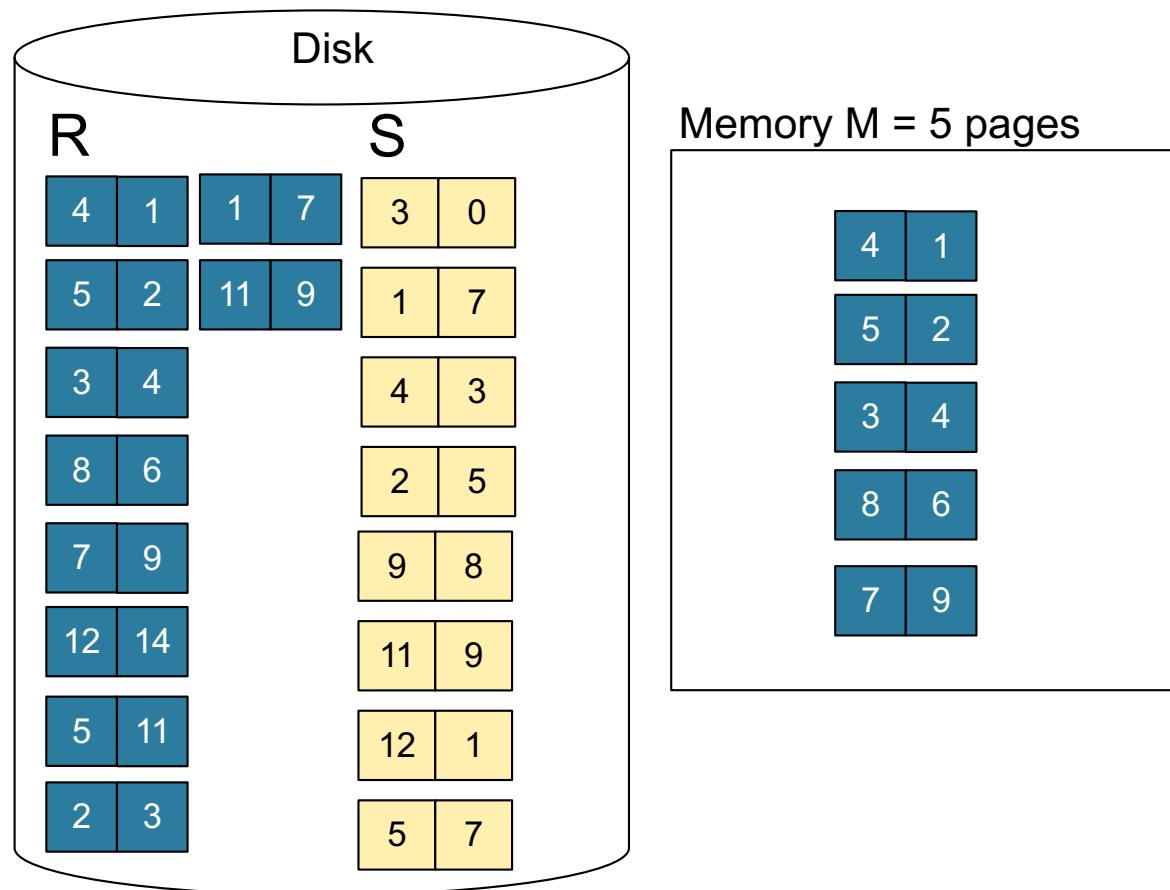
- Relation R has 10 pages with 2 tuples per page
- Relation S has 8 pages with 2 tuples per page

Values shown are values of join attribute for each given tuple



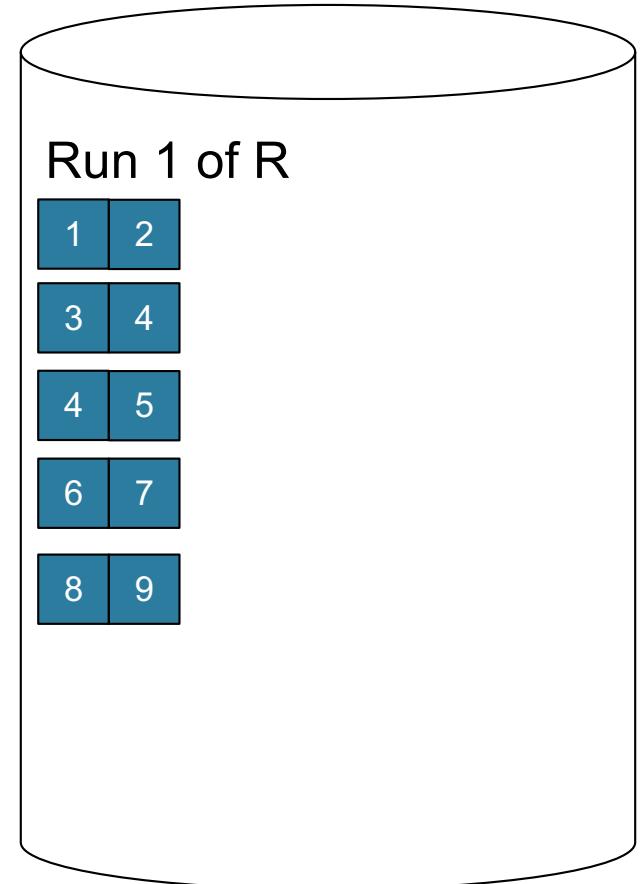
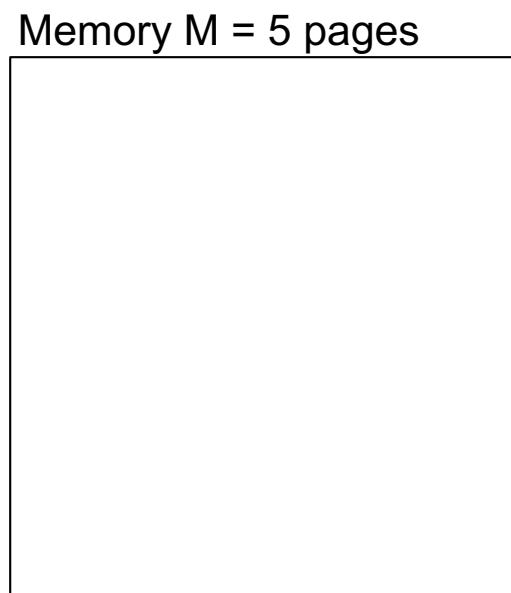
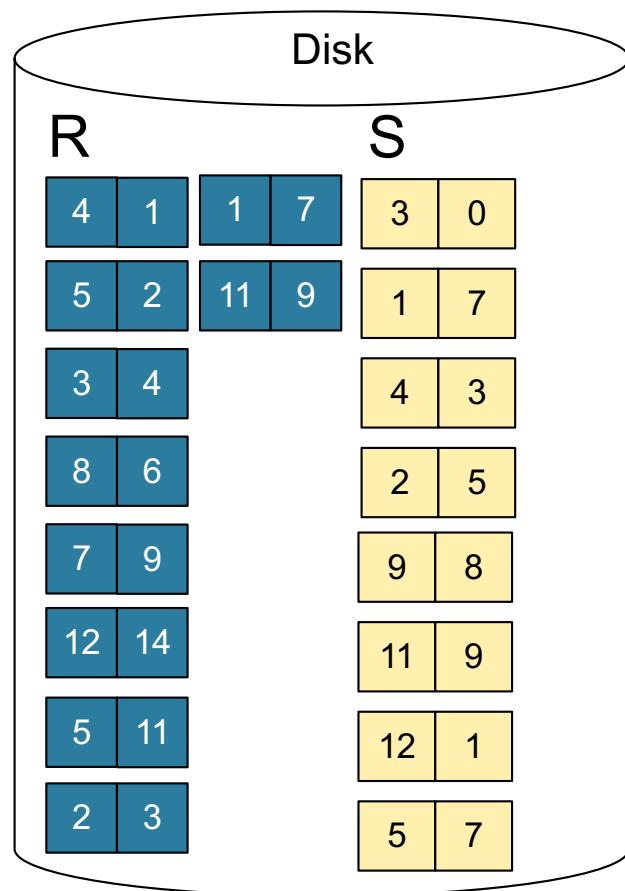
Merge-Join Example

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



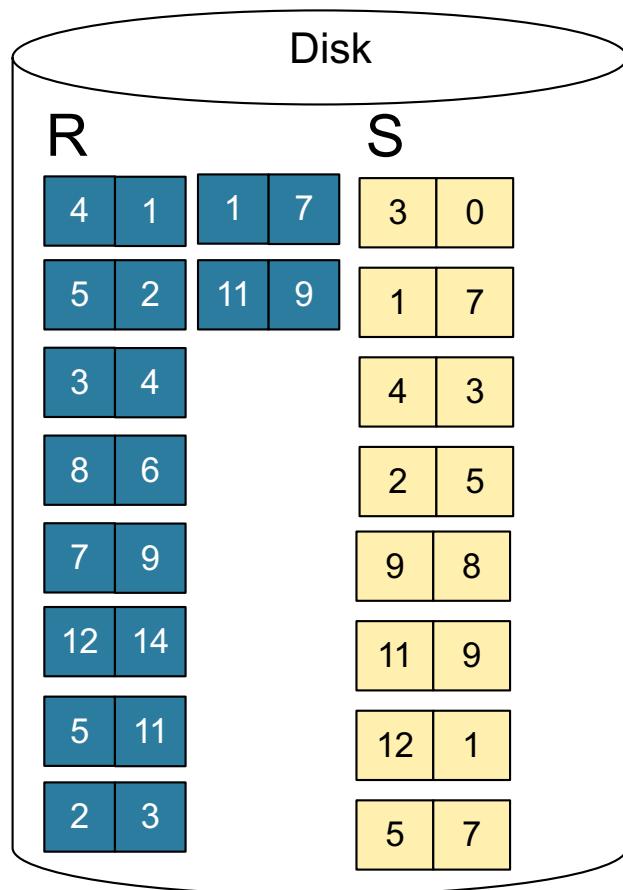
Merge-Join Example

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

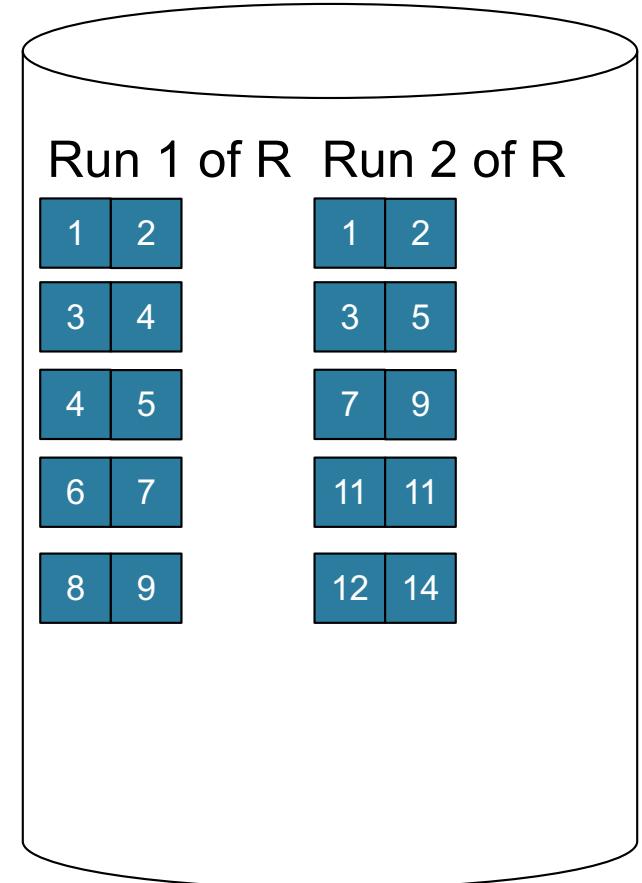
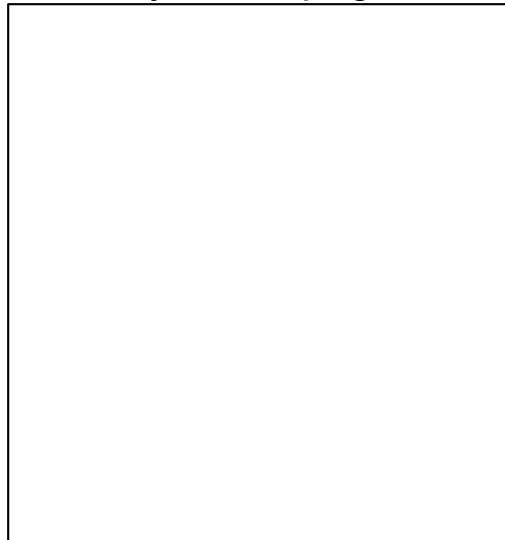


Merge-Join Example

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

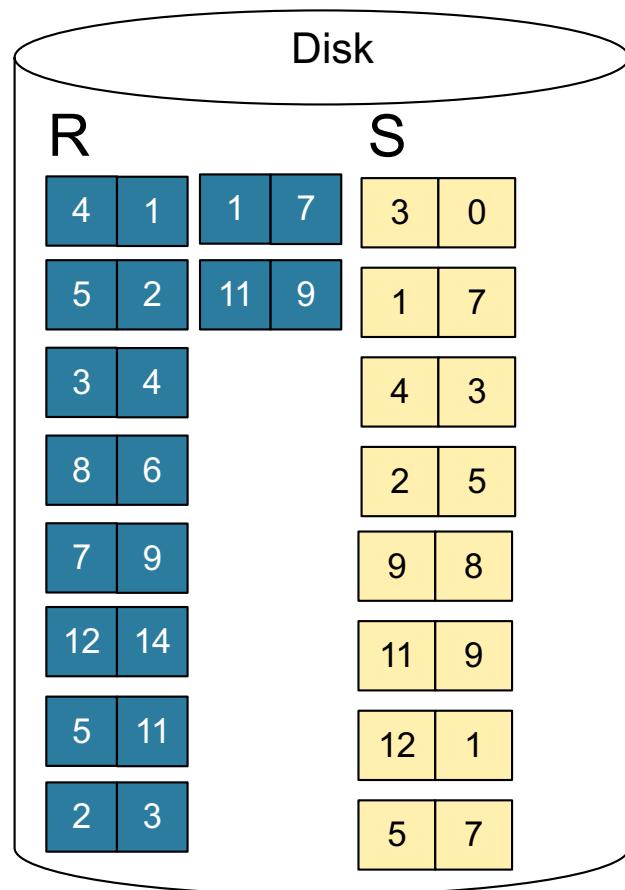


Memory M = 5 pages

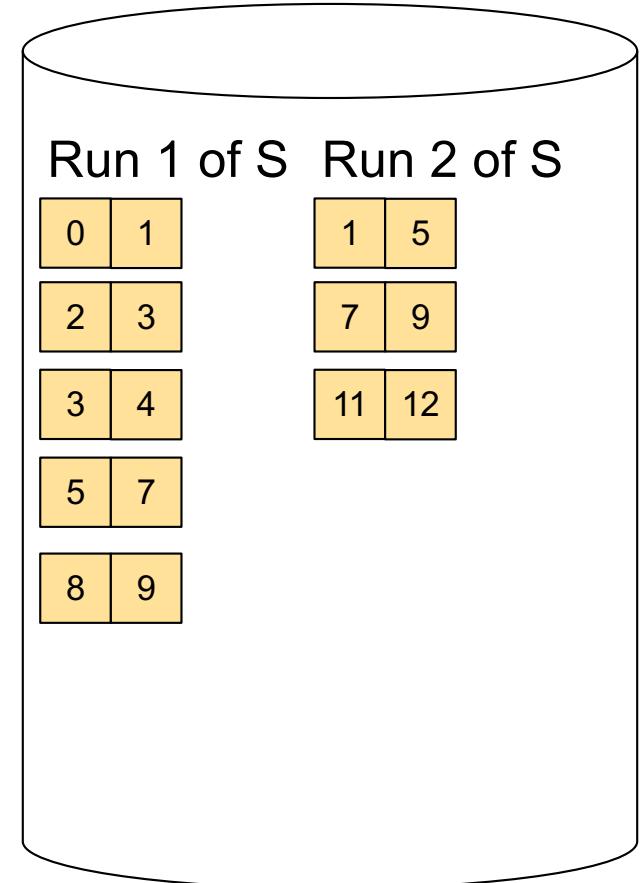
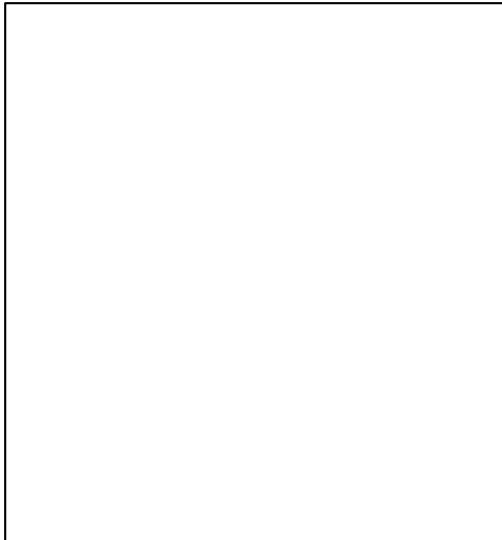


Merge-Join Example

Step 1: Do the same with S

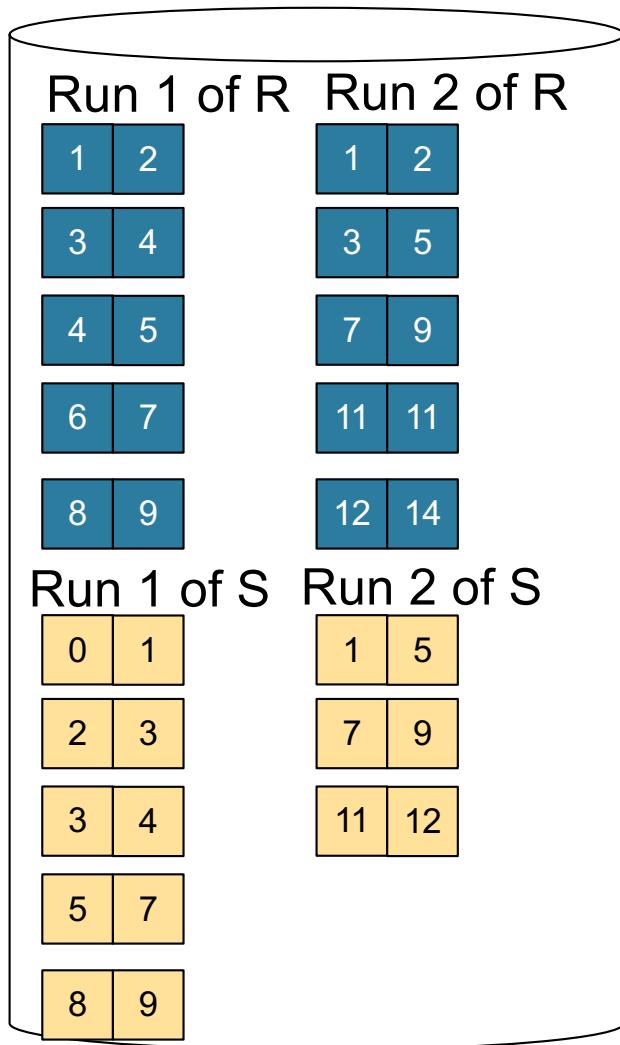


Memory M = 5 pages



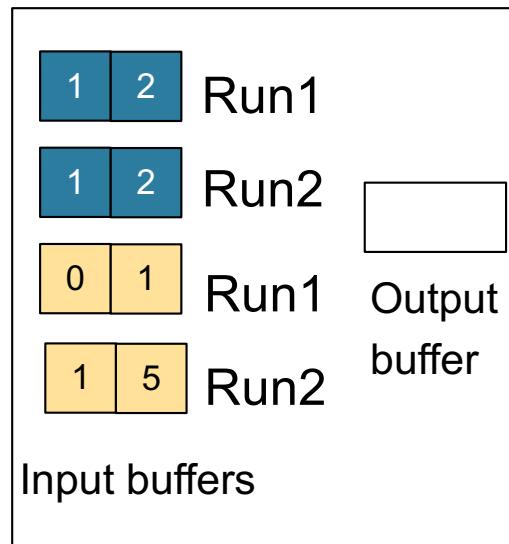
Merge-Join Example

Step 2: Join while merging sorted runs



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

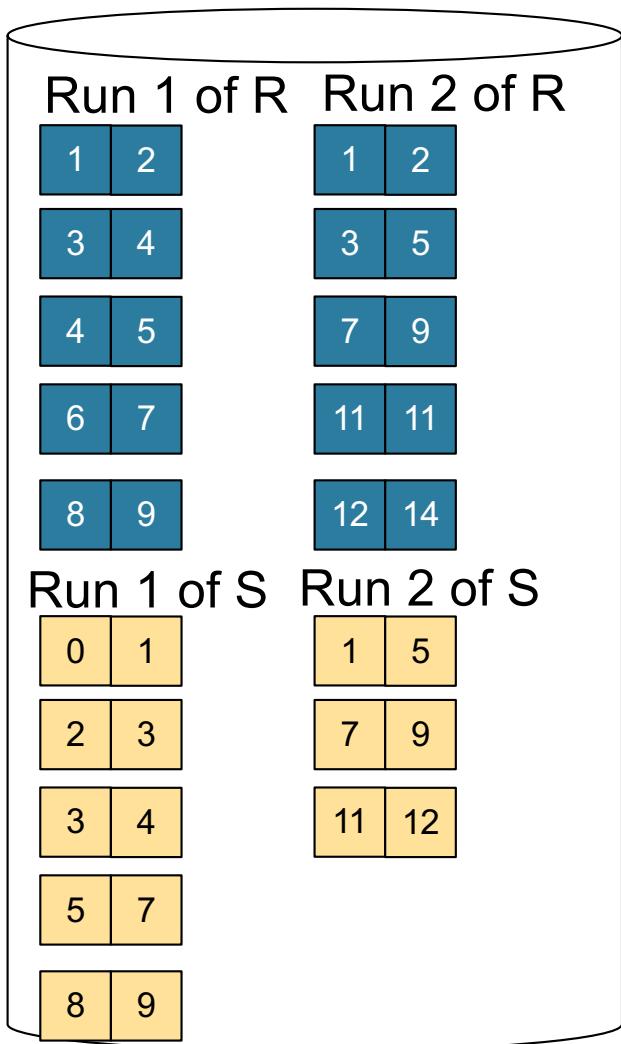
Memory M = 5 pages



Step 2: Join while merging
Output tuples

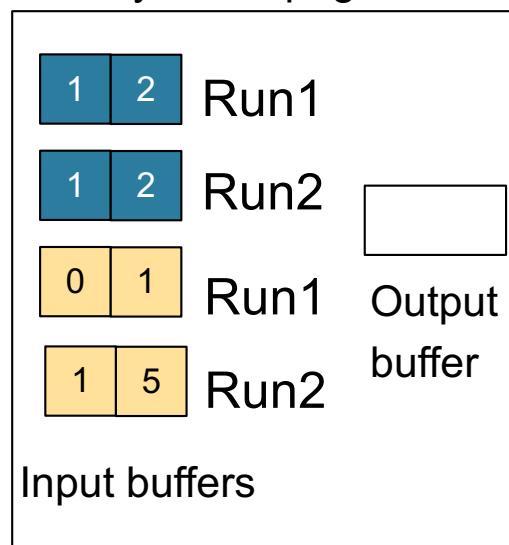
Merge-Join Example

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

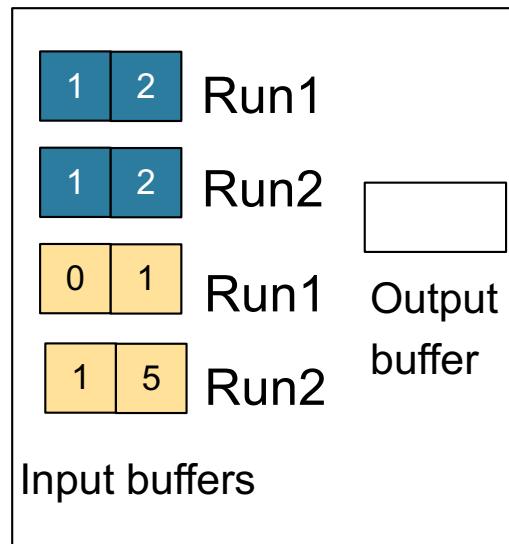
Step 2: Join while merging sorted runs

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

Memory M = 5 pages



Step 2: Join while merging

Output tuples

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

Merge-Join Example

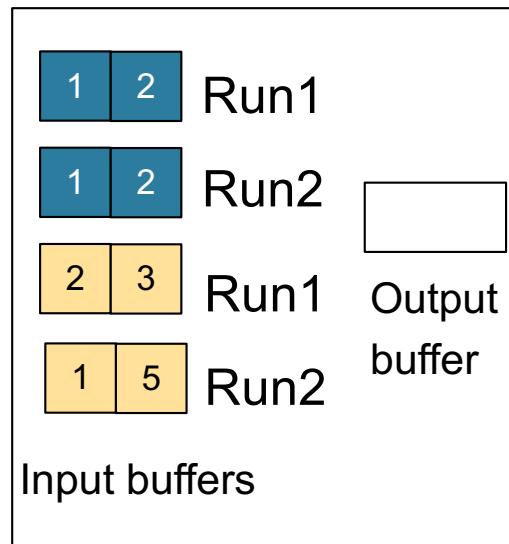
Step 2: Join while merging sorted runs

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

Memory M = 5 pages



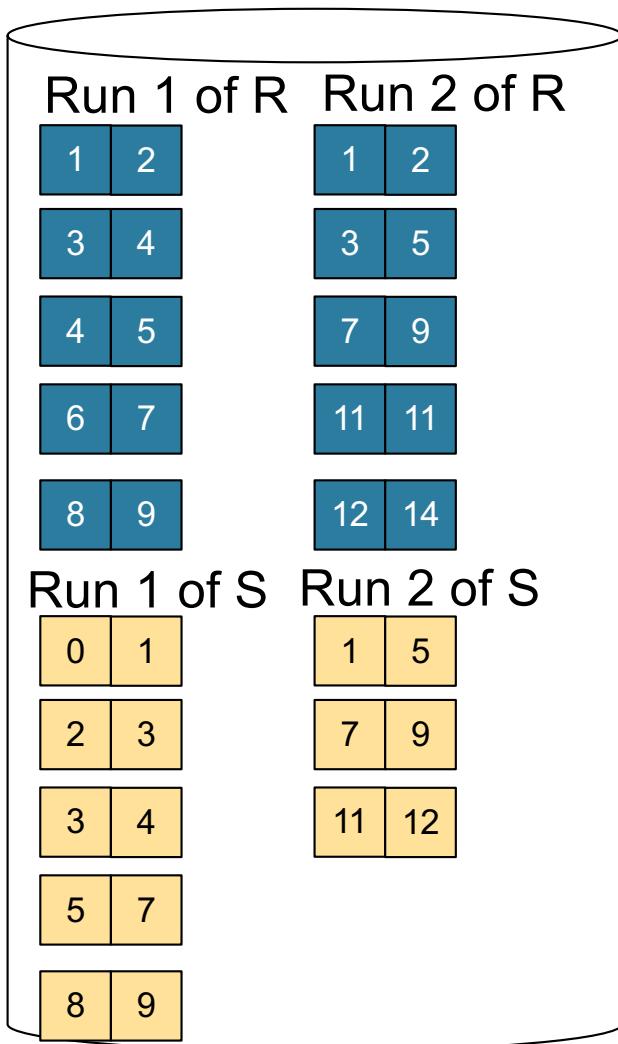
Step 2: Join while merging

Output tuples

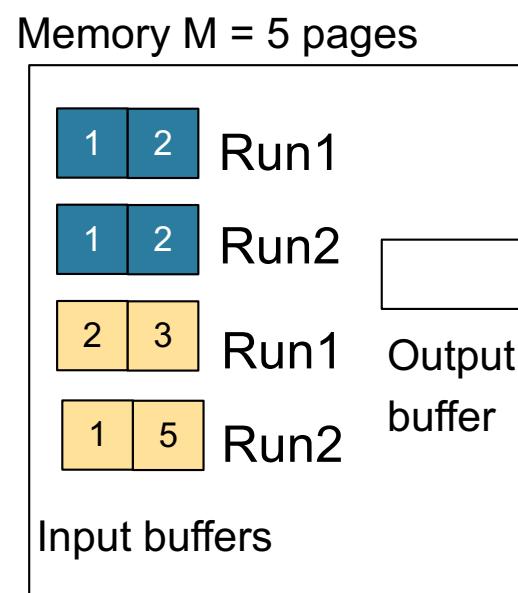
- (1,1)
- (1,1)
- (1,1)
- (1,1)

Merge-Join Example

Step 2: Join while merging sorted runs



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



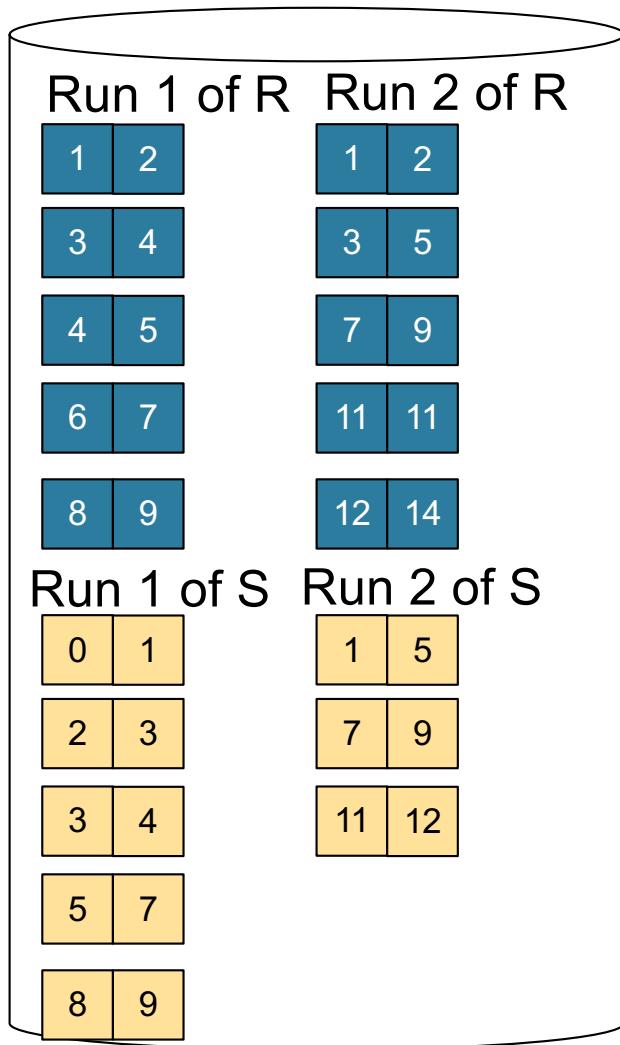
Step 2: Join while merging

Output tuples

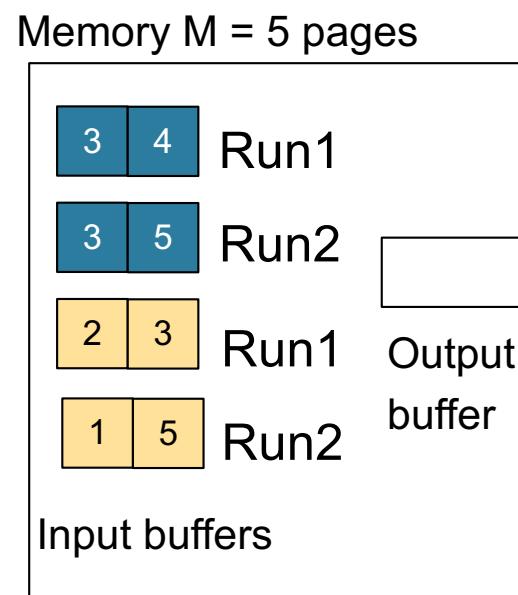
- (1,1)
- (1,1)
- (1,1)
- (1,1)
- (2,2)
- (2,2)

Merge-Join Example

Step 2: Join while merging sorted runs



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



Step 2: Join while merging

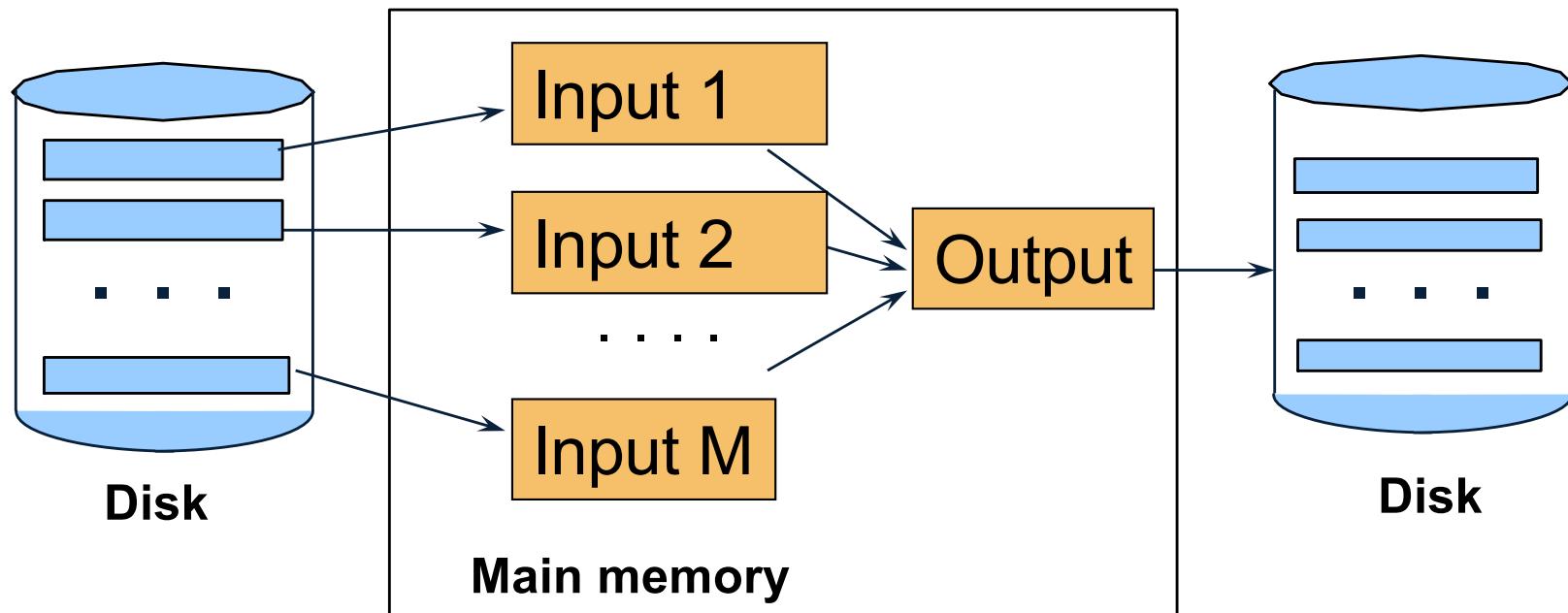
Output tuples

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

Announcements

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

Merge-Join



$$M_1 = B(R)/M \text{ runs for } R$$

$$M_2 = B(S)/M \text{ runs for } S$$

Merge-join $M_1 + M_2$ runs;

need $M_1 + M_2 \leq M$ to process all runs

$$\text{i.e. } B(R) + B(S) \leq M^2$$

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
 $B(R_i) = B(R)/k, \text{ for all } i$

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
 $B(R_i) = B(R)/k, \text{ for all } i$
- Goal: each R_i should fit in main memory:
 $B(R_i) \leq M$

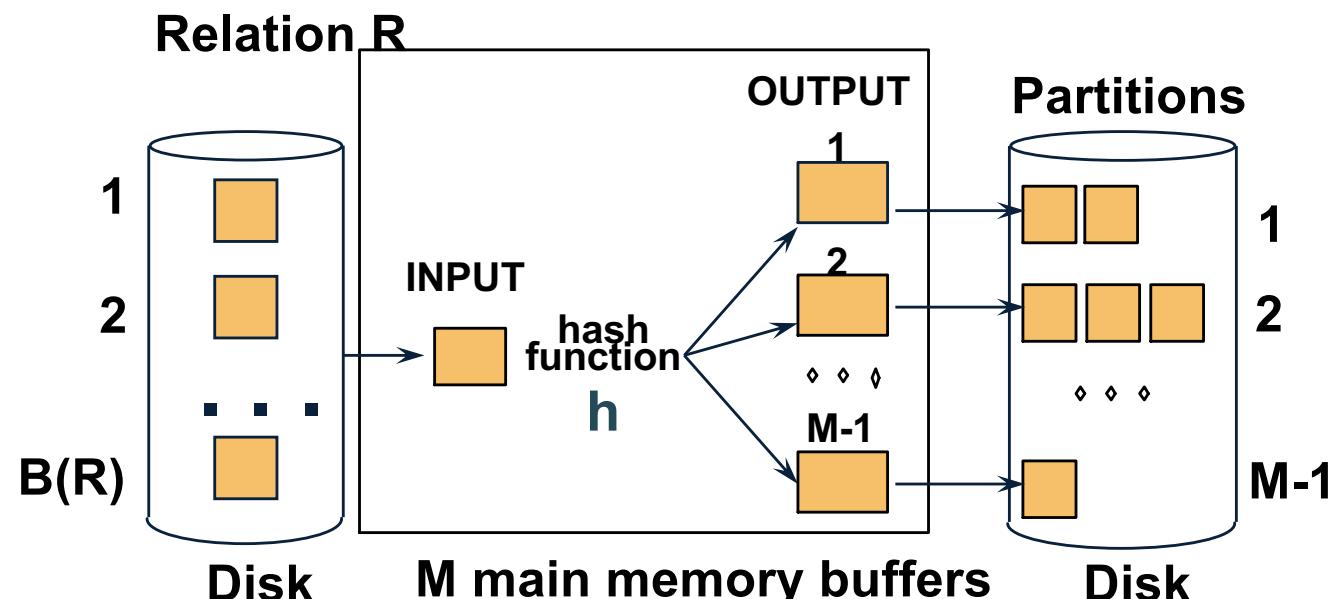
Partitioned Hash Algorithms

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

How do we choose k?

Partitioned Hash Algorithms

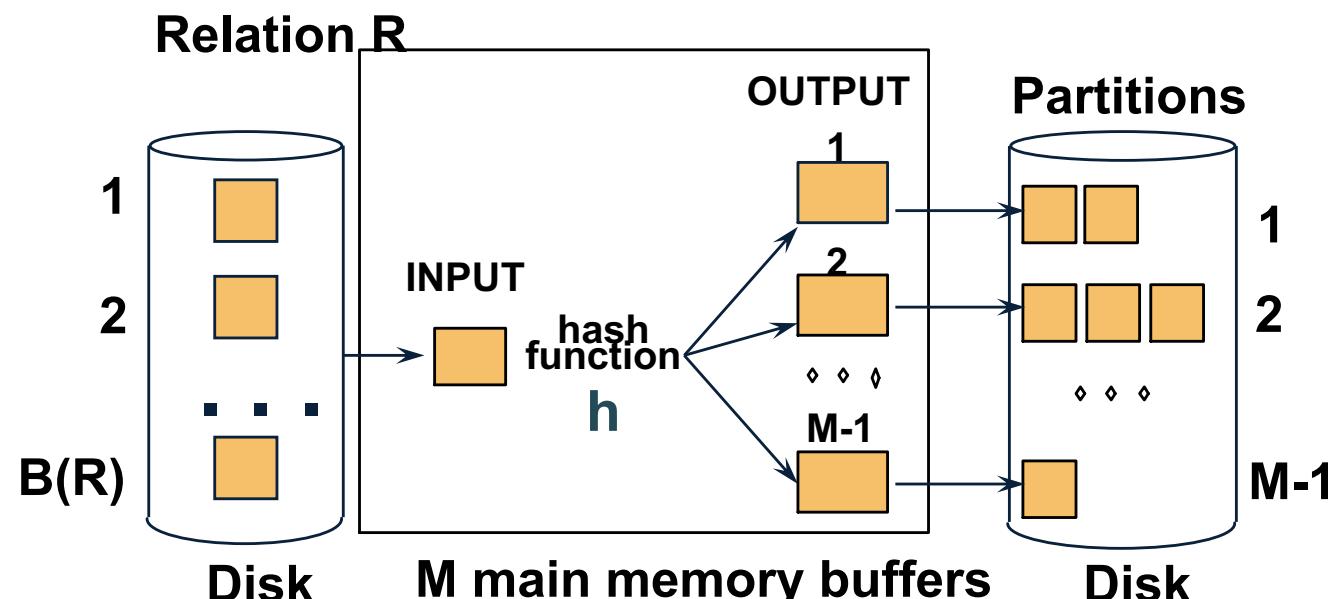
- We choose $k = M-1$ Each bucket has size approx. $B(R)/(M-1) \approx B(R)/M$



Assumption: $B(R)/M \leq M$, i.e. $B(R) \leq M^2$

Partitioned Hash 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

$R \bowtie S$

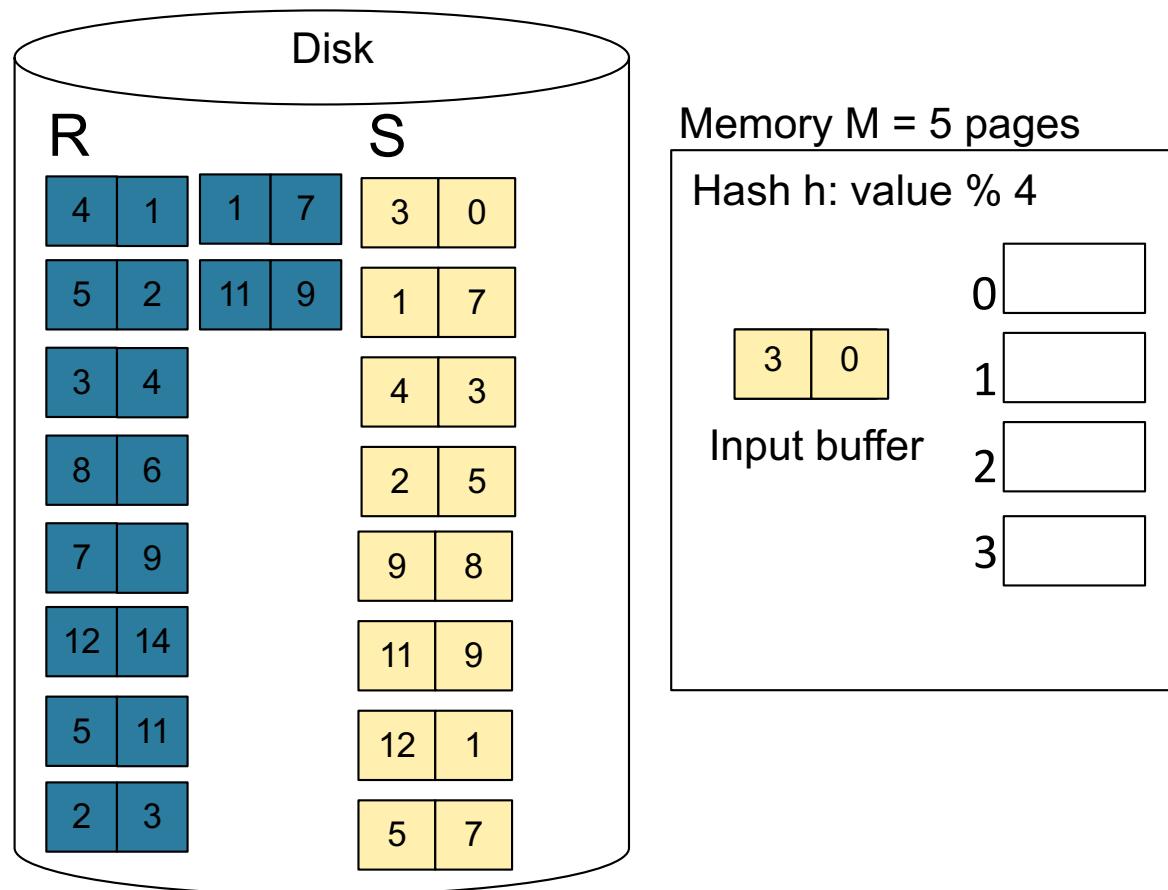
- Step 1:
 - Hash S into M-1 buckets
 - Send all buckets to disk
- Step 2
 - Hash R into M-1 buckets
 - Send all buckets to disk
- Step 3
 - Join every pair of buckets



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

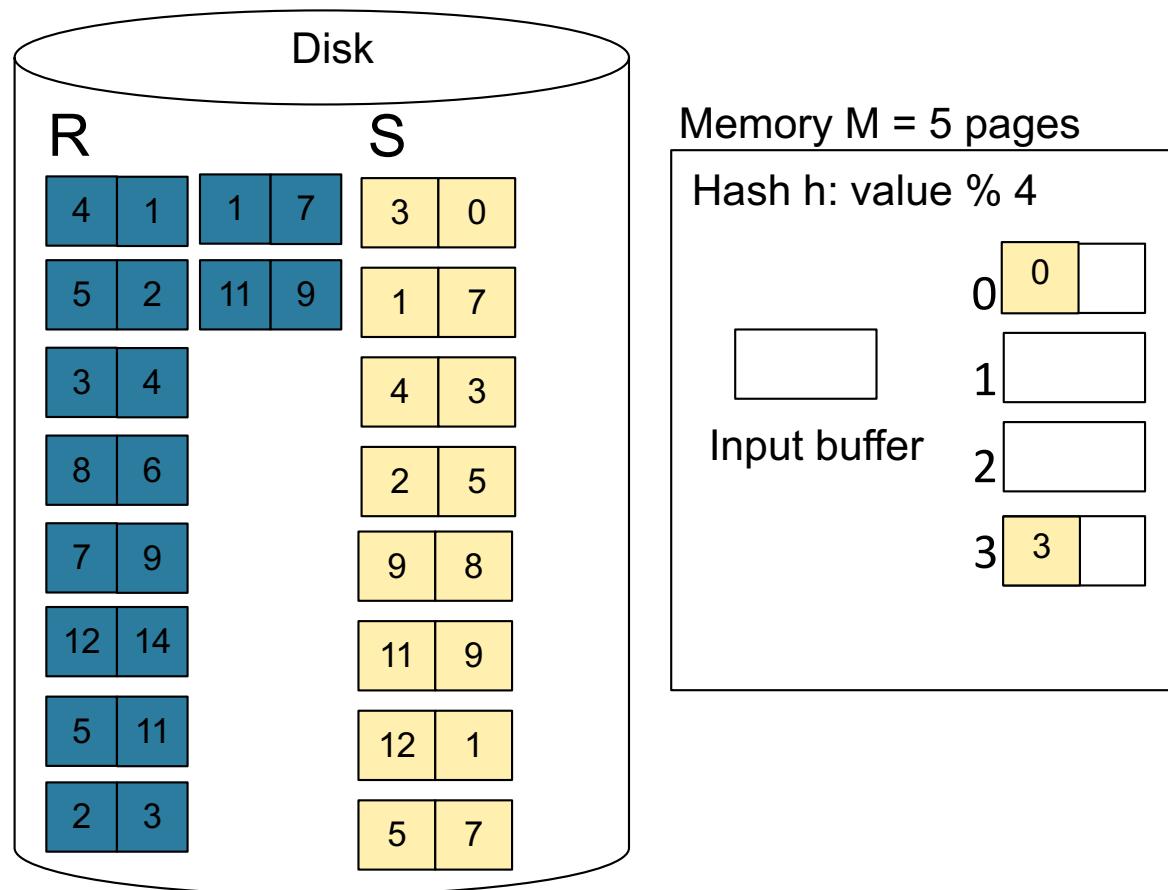
Partitioned Hash-Join Example

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



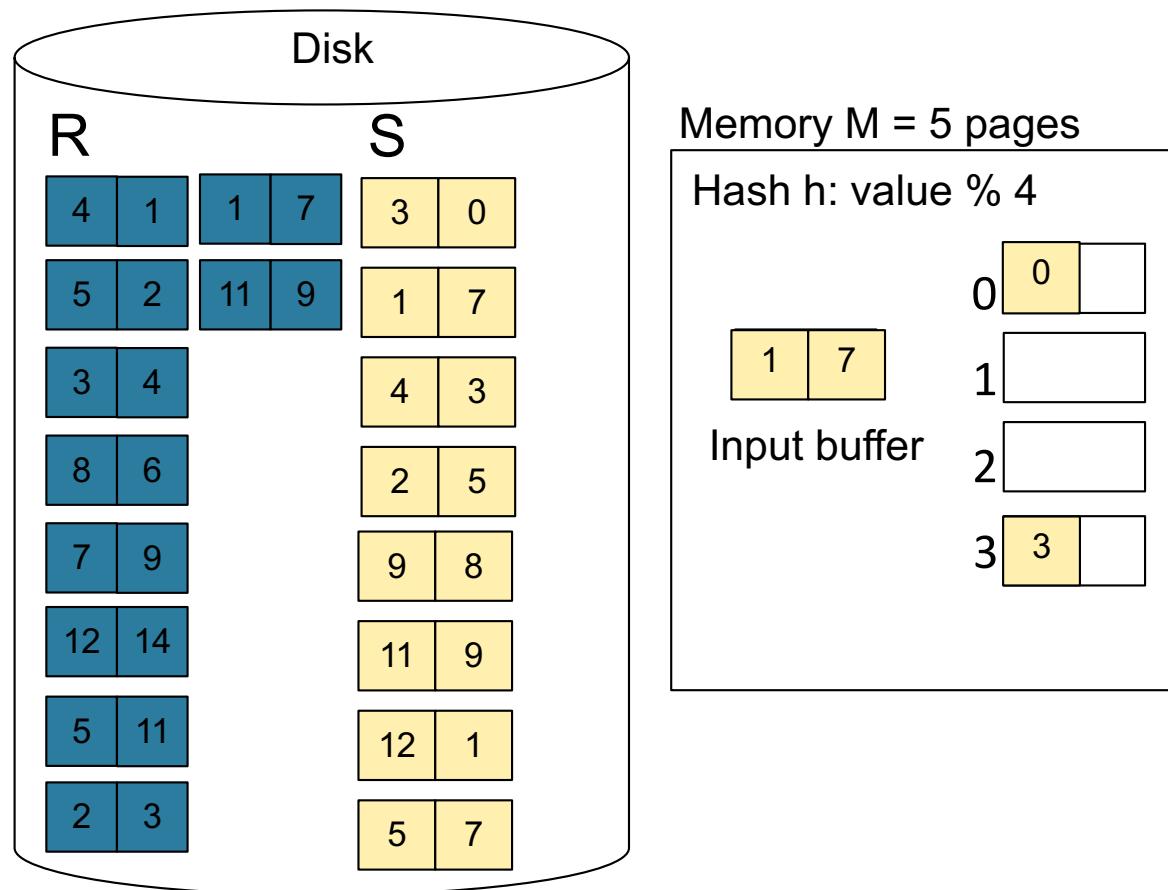
Partitioned Hash-Join Example

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



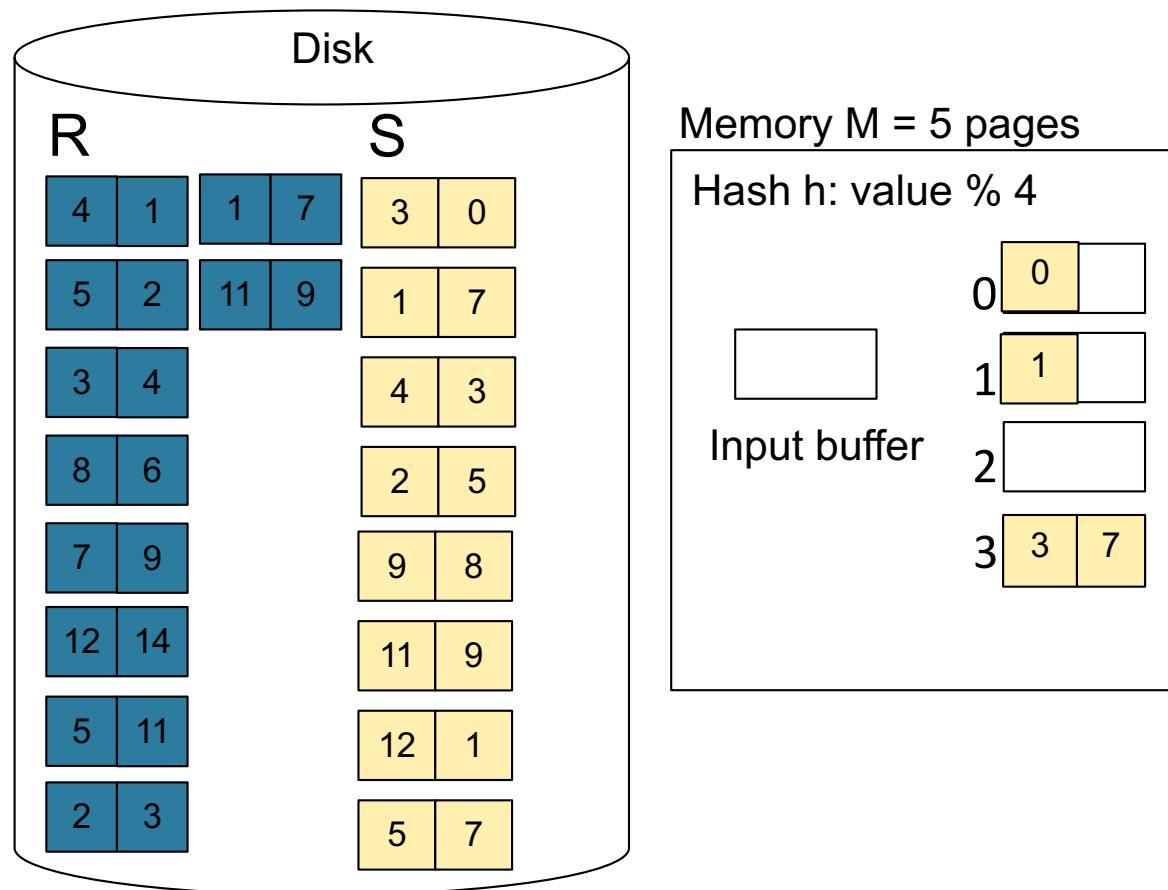
Partitioned Hash-Join Example

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



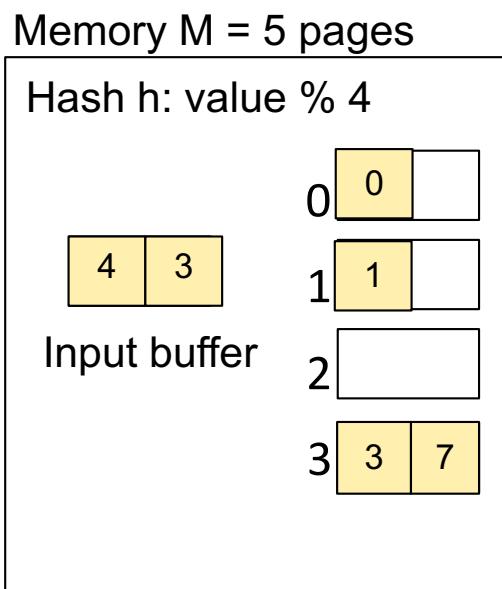
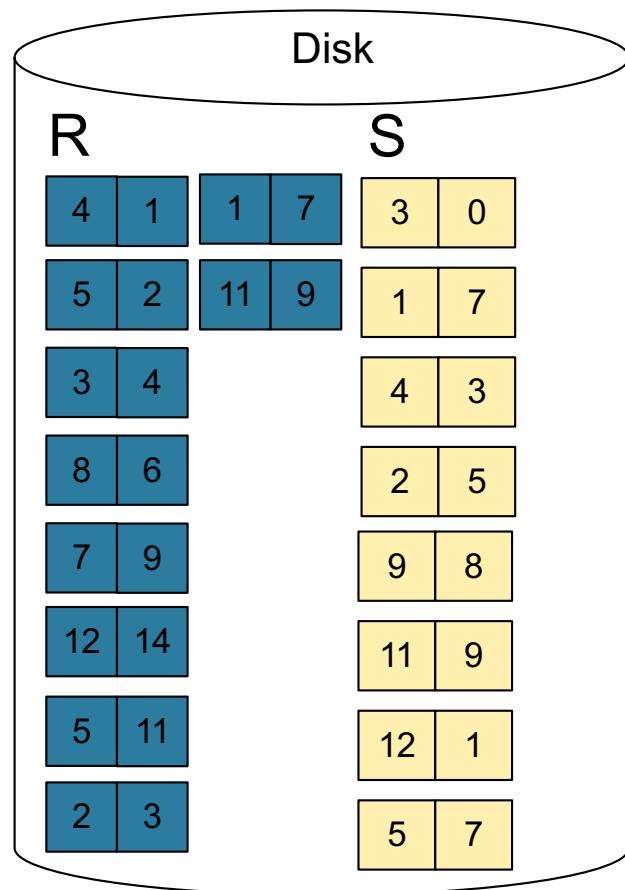
Partitioned Hash-Join Example

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



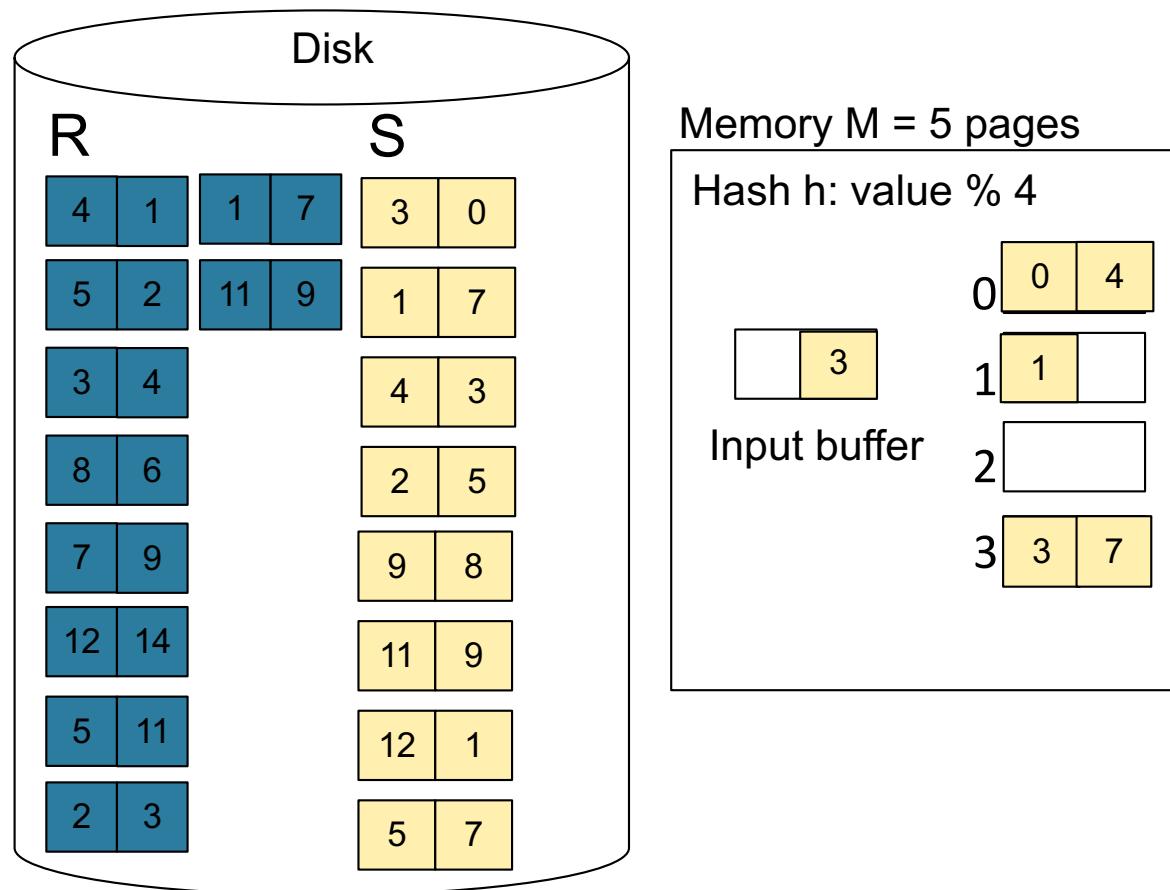
Partitioned Hash-Join Example

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



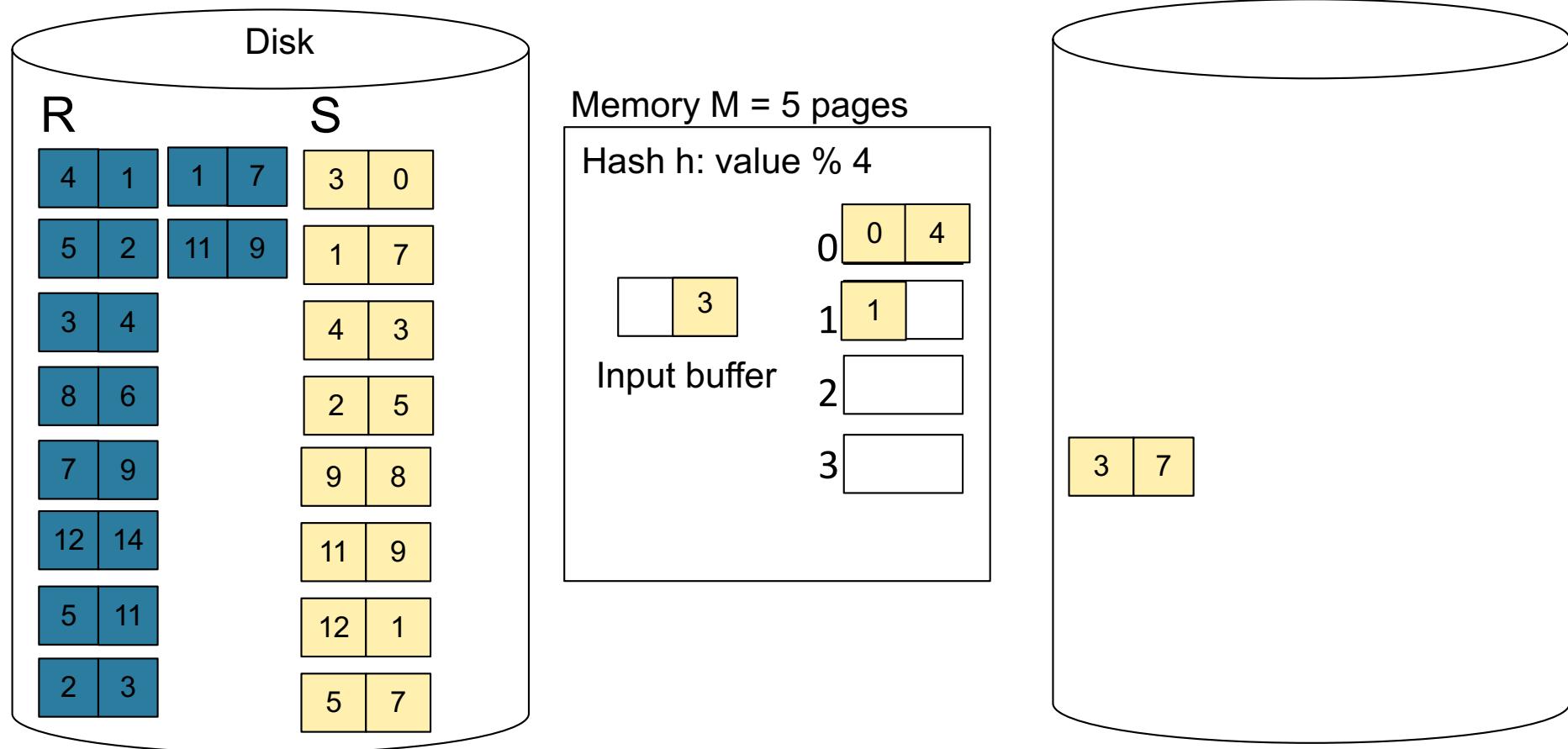
Partitioned Hash-Join Example

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

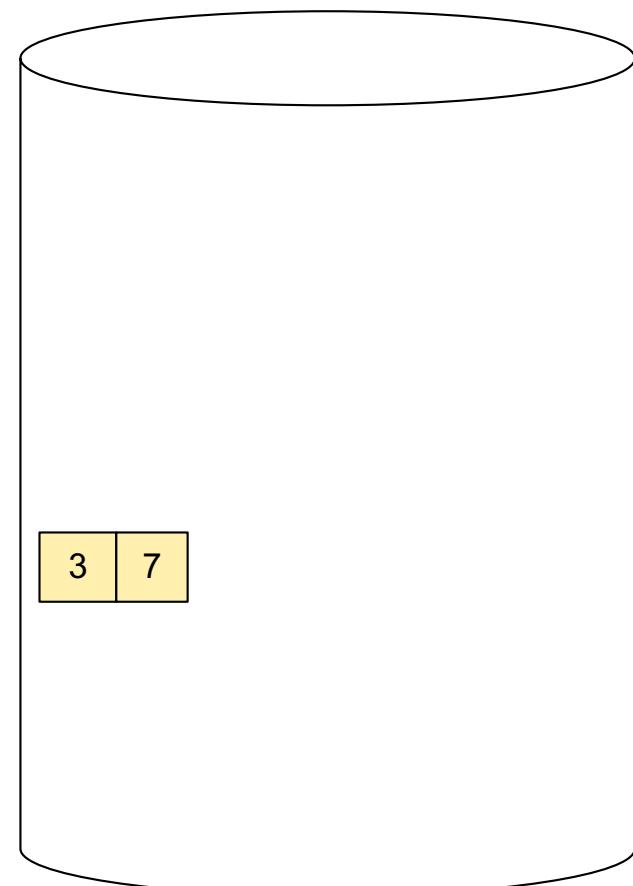
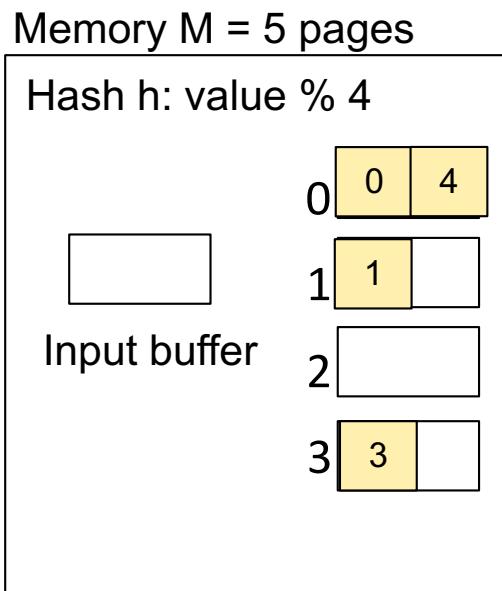
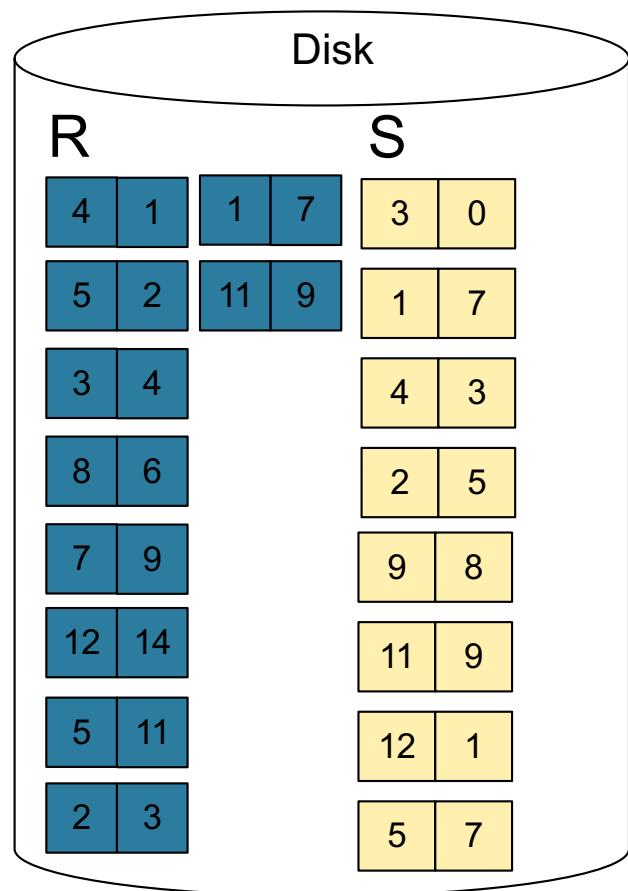
Step 1: Read relation S one page at a time and hash into the 4 buckets
When a bucket fills up, flush it to disk



Partitioned Hash-Join Example

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

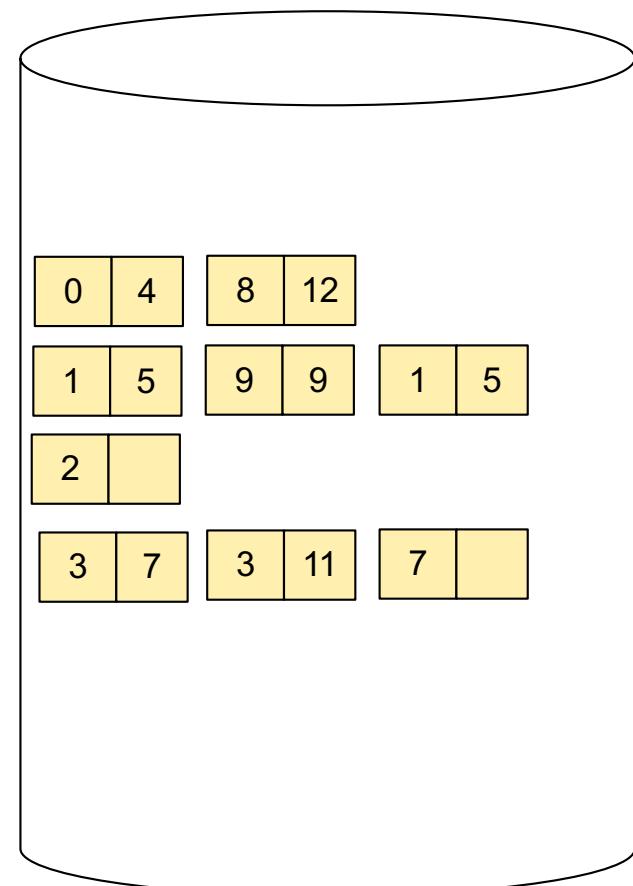
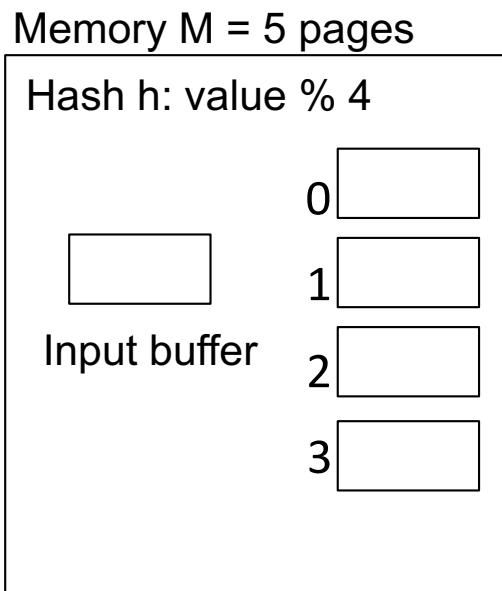
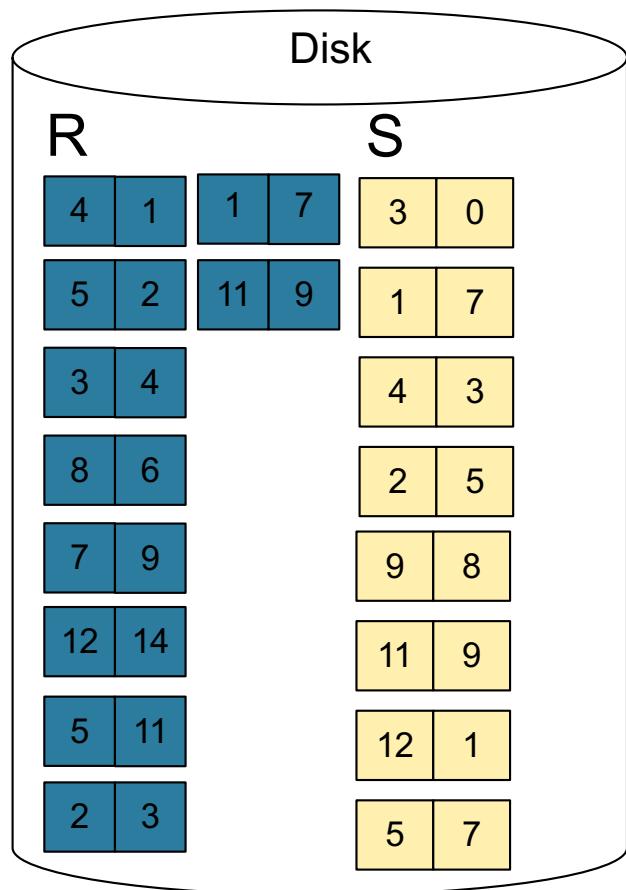
When a bucket fills up, flush it to disk



Partitioned Hash-Join Example

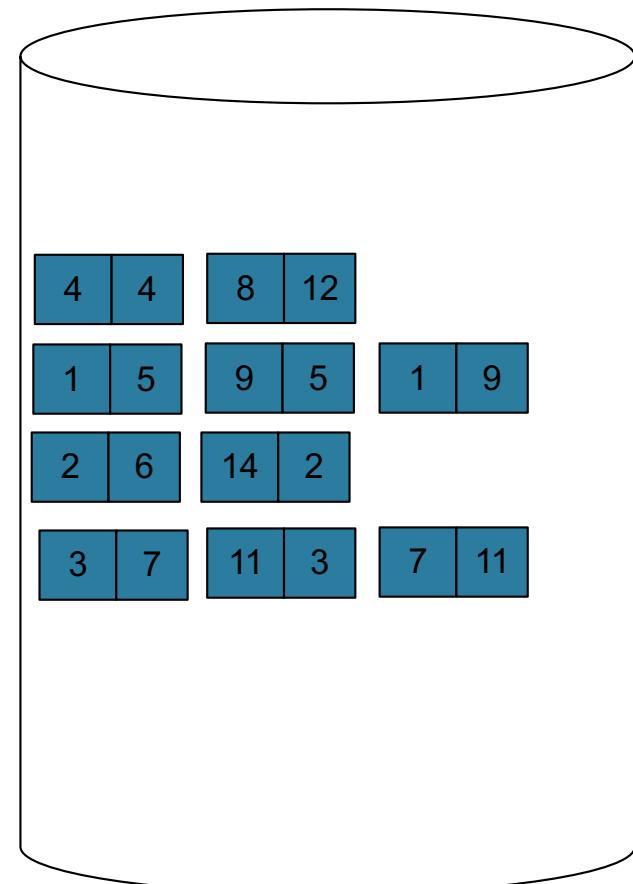
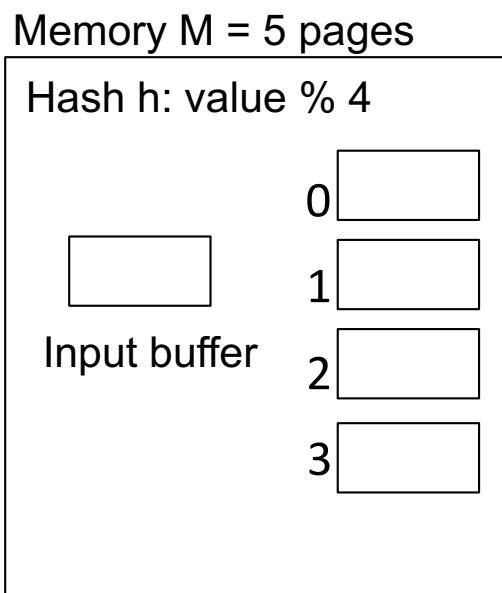
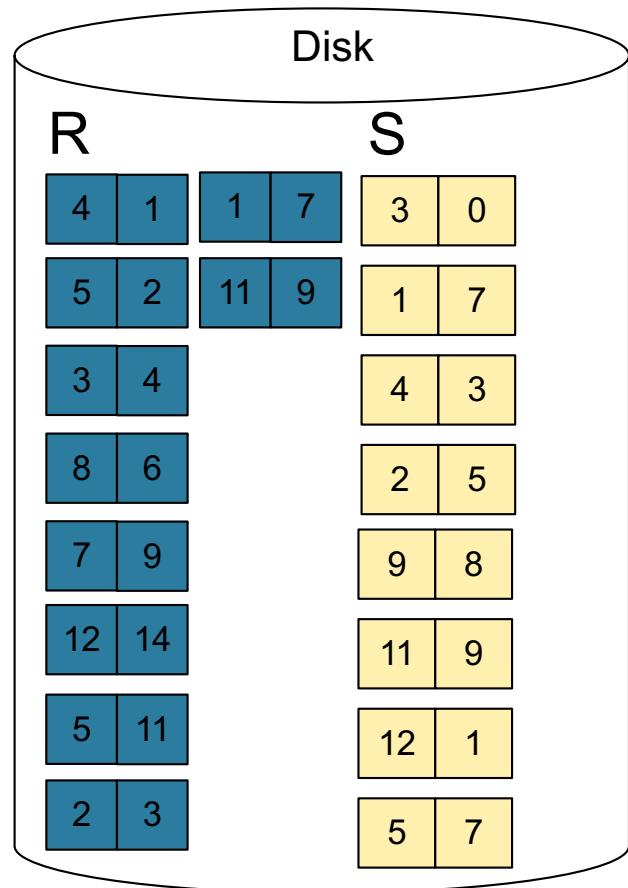
Step 1: Read relation S one page at a time and hash into the 4 buckets

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



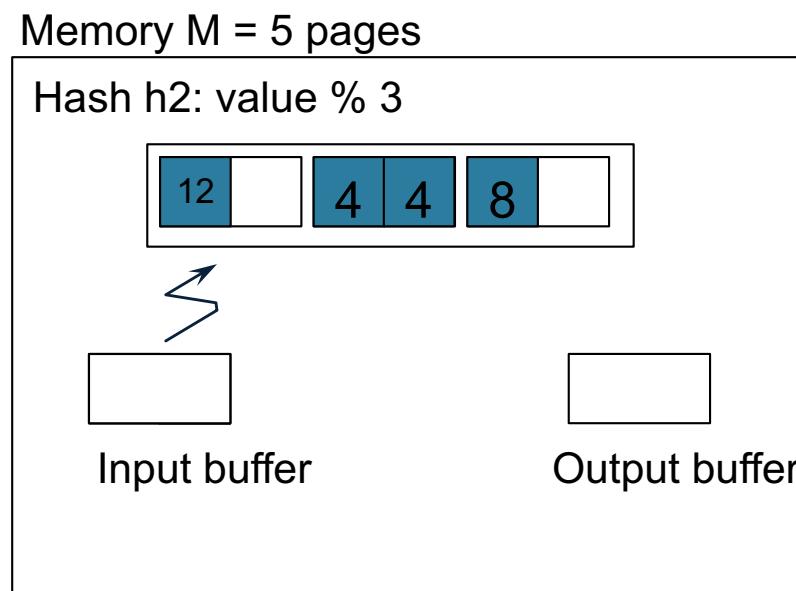
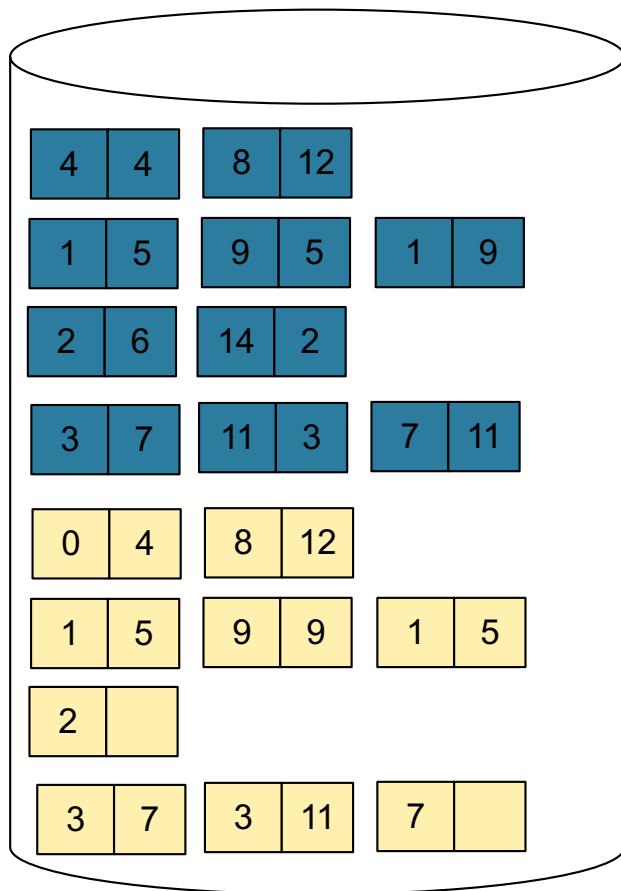
Partitioned Hash-Join Example

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

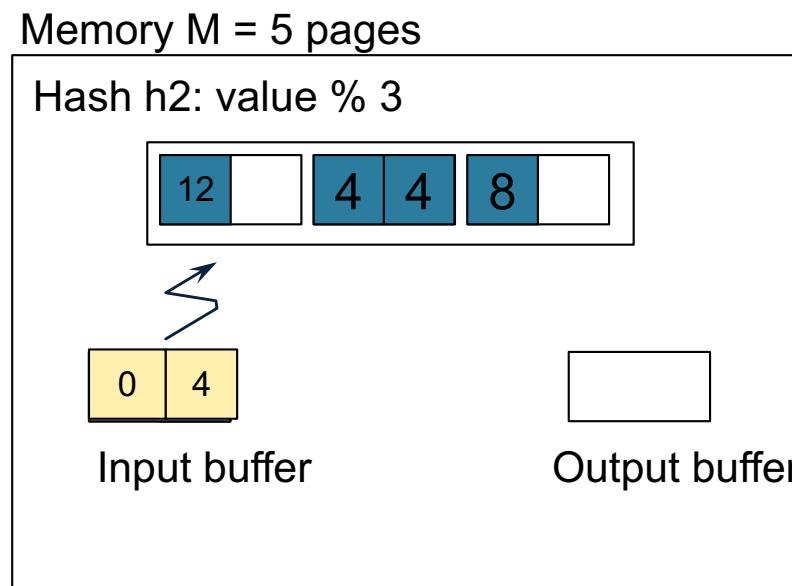
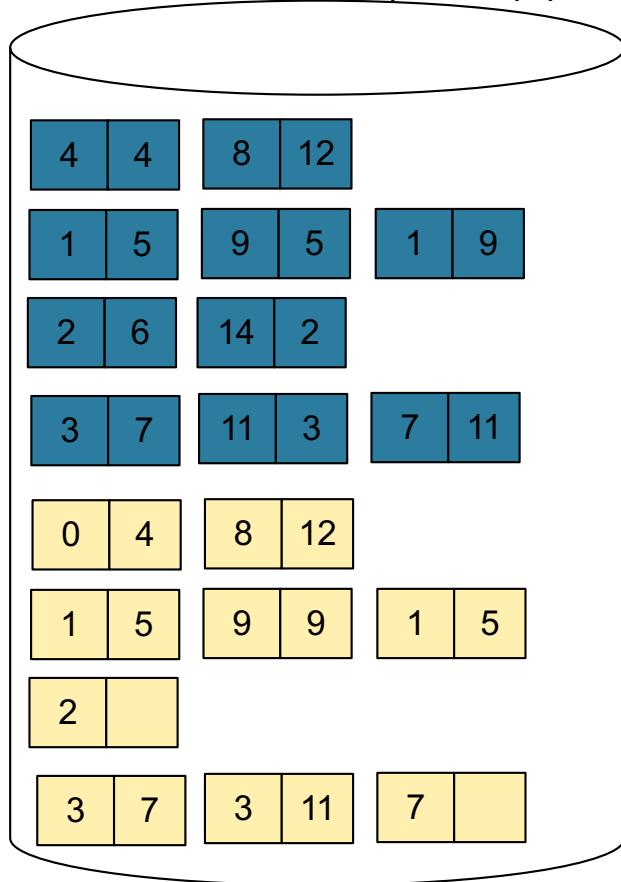


Partitioned Hash-Join Example

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

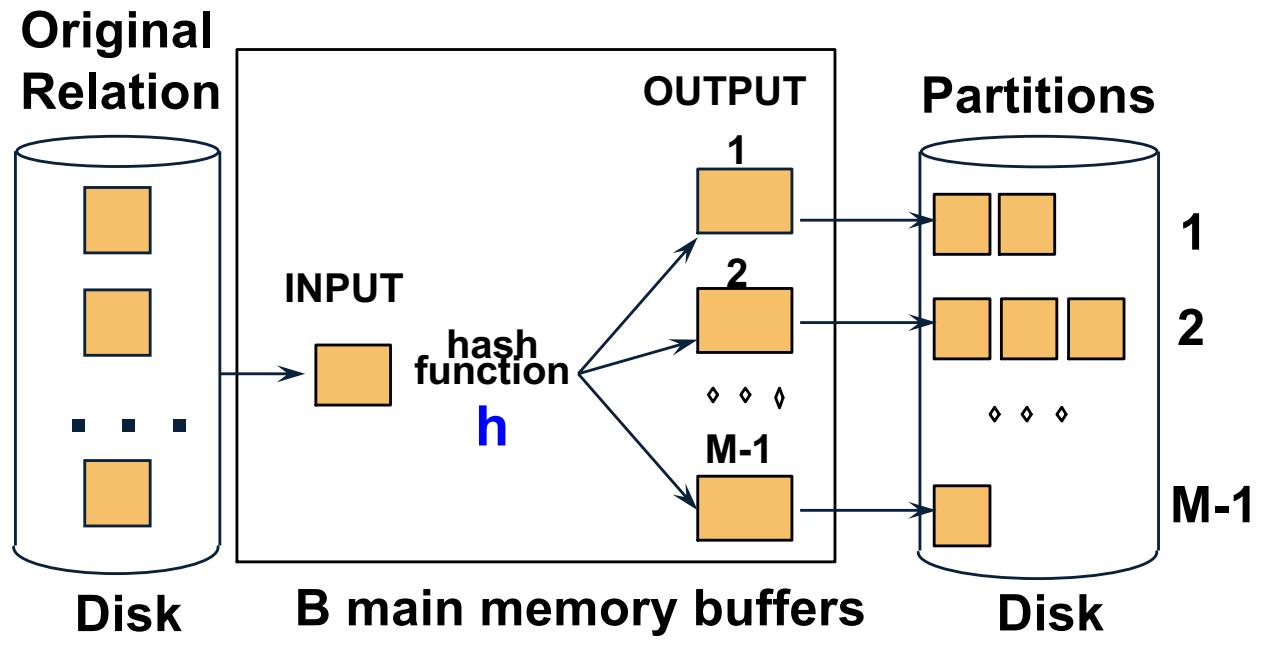
Step 5: Repeat for all the buckets

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



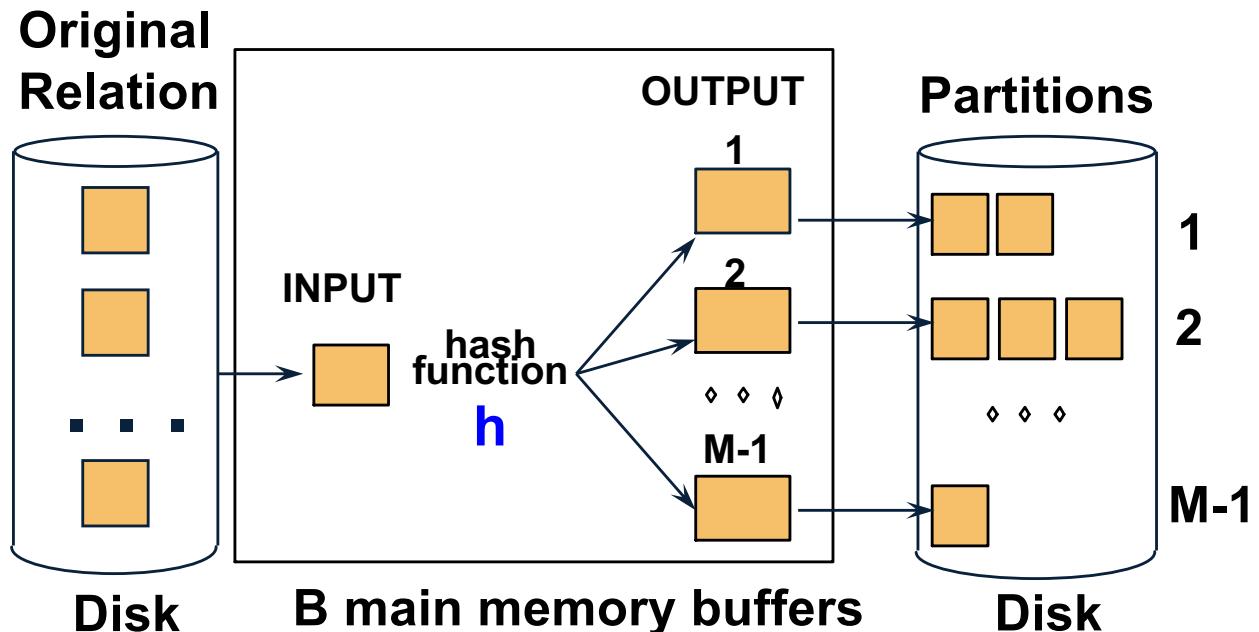
Grace-Join

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

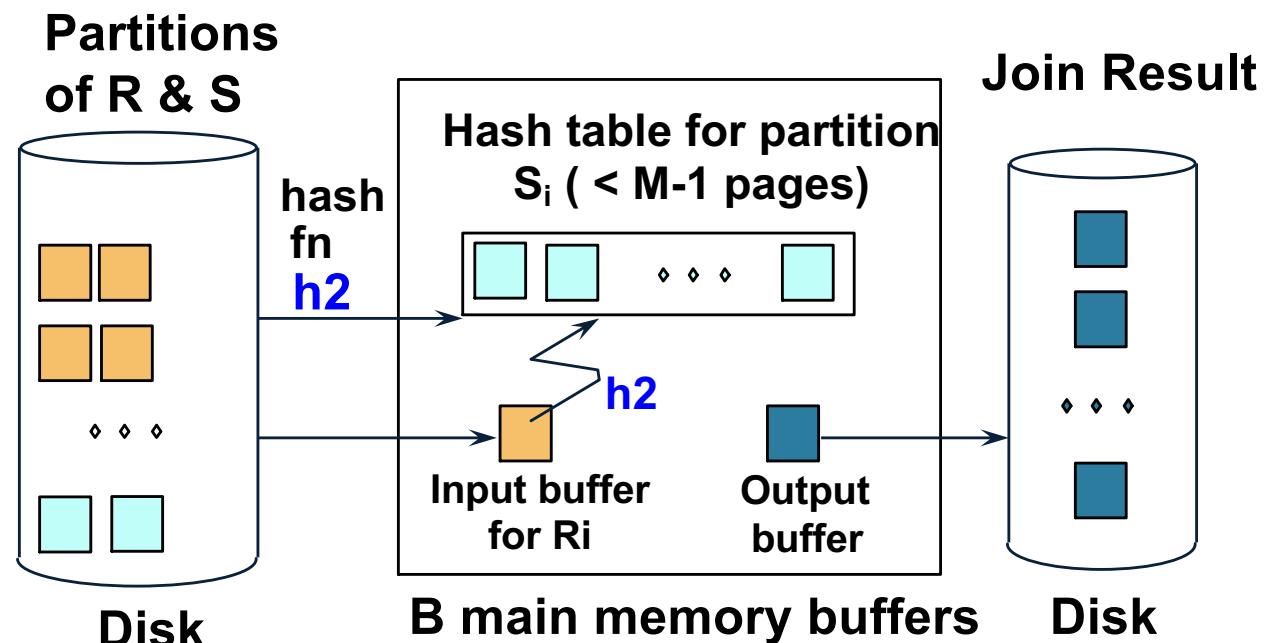


Grace-Join

- 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 $h_2 (< \neq h)$. Scan matching partition of S, search for matches.



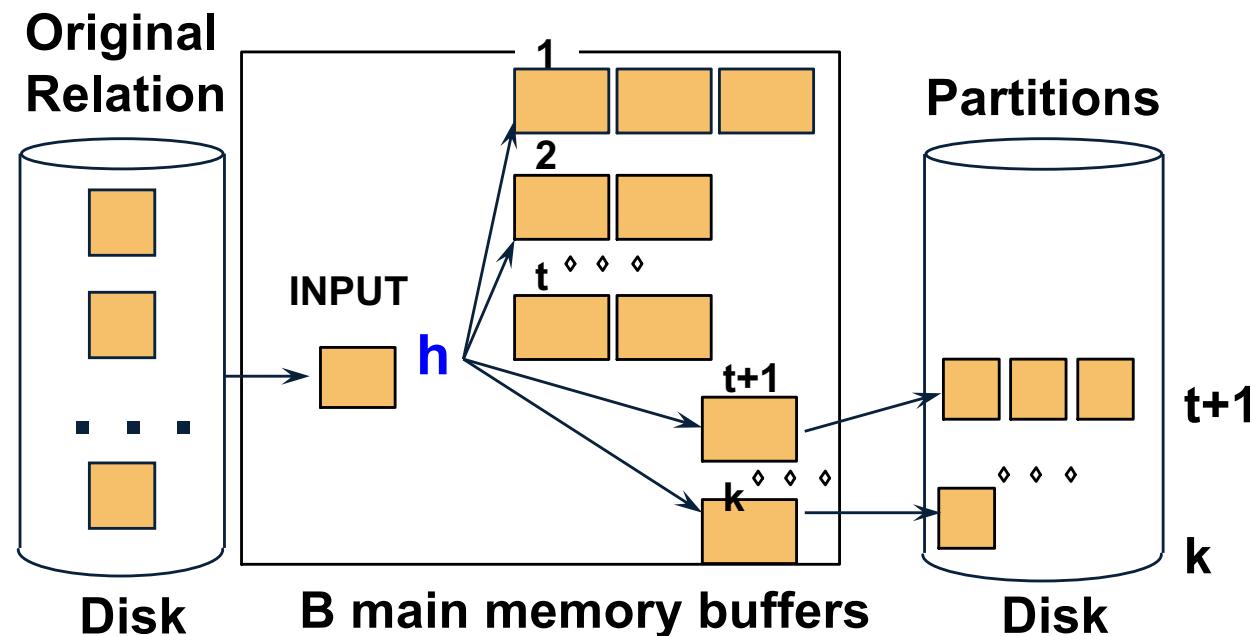
Grace Join

- 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.

$$\frac{t}{k} * B(S) \leq M$$

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

$$\frac{t}{k} * B(S) \leq M$$

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

$$\frac{t}{k} * B(S) \leq M$$

- Together:

$$\frac{t}{k} * B(S) + k - t \leq M$$

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

$$\frac{t}{k} * B(S) \leq M$$

- Together:

$$\frac{t}{k} * B(S) + k - t \leq M$$

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

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

$$\frac{t}{k} * B(S) \leq M$$

- Together:

$$\frac{t}{k} * B(S) + k - t \leq M$$

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

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

$$k \leq M$$

- Choose t/k large but s.t.

First t buckets in memory

$$\frac{t}{k} * B(S) \leq M$$

- Together:

$$\frac{t}{k} * B(S) + k-t \leq M$$

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

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 $2 t/k$ 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: $B(S) + B(R)^*B(S)/(M-1)$
- Index Join: $B(R) + T(R)B(S)/V(S,a)$
(unclustered)
- 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?