CSC553 Advanced Database Concepts

Tanu Malik

School of Computing

DePaul University

Last time

- Index structures
- Hash-based indexes
- B+ trees

Types of Operator Algorithms

- One-pass algorithms
  - Reading data from disk only once.
  - One argument to fit in memory except select and project operators
- Two-pass algorithms
  - Data too large to fit in main memory
  - Reading data a first time from disk, processing it is some way, then reading again from disk.
- Index-based algorithms
  - Use indexes to reduce the amount of data fetched.

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)

Cost Convention

- Cost = the cost of reading operands from disk
- Cost of writing the final result to disk is not included; need to count it separately when applicable

- **Assumption:** Arguments to operator are on disk but result is in main memory.
  - If final answer, then result is written to disk and the cost of doing so depends on the size of the answer and not how it was computed.

Join Algorithms

- **Hash join : B(R) + B(S)**
- Nested loop join
- Sort-merge join

Hash Join T1 ⋈ T2

T1

1	‘Bob’	‘Seattle’
2	‘Ela’	‘Everett’

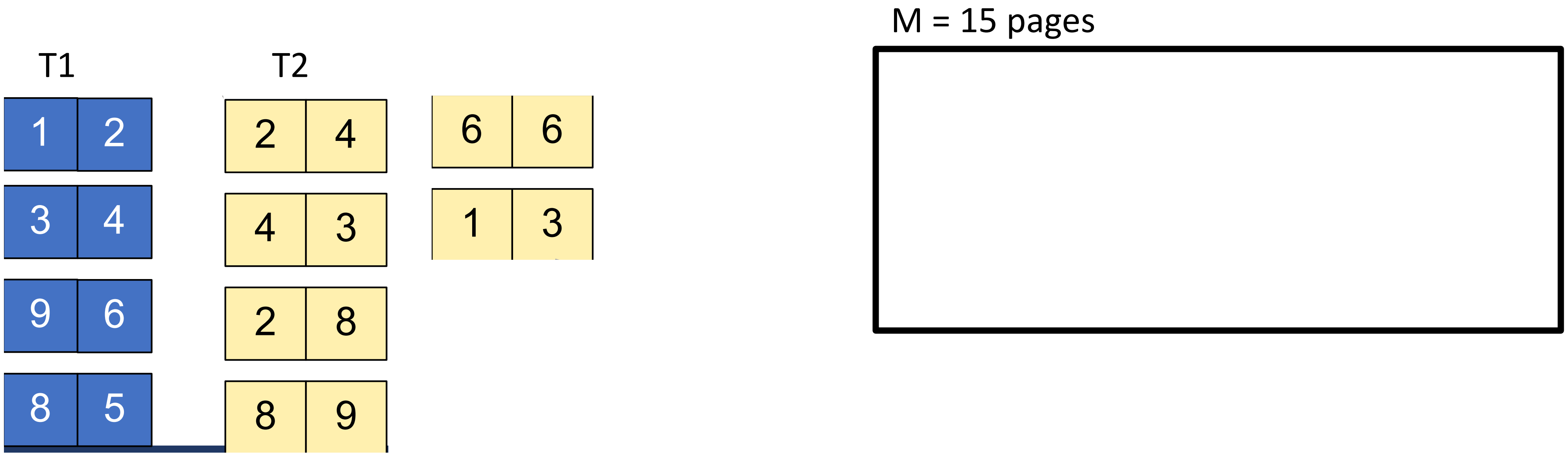
3	‘Jill’	‘Kent’
4	‘Joe’	‘Seattle’

T2

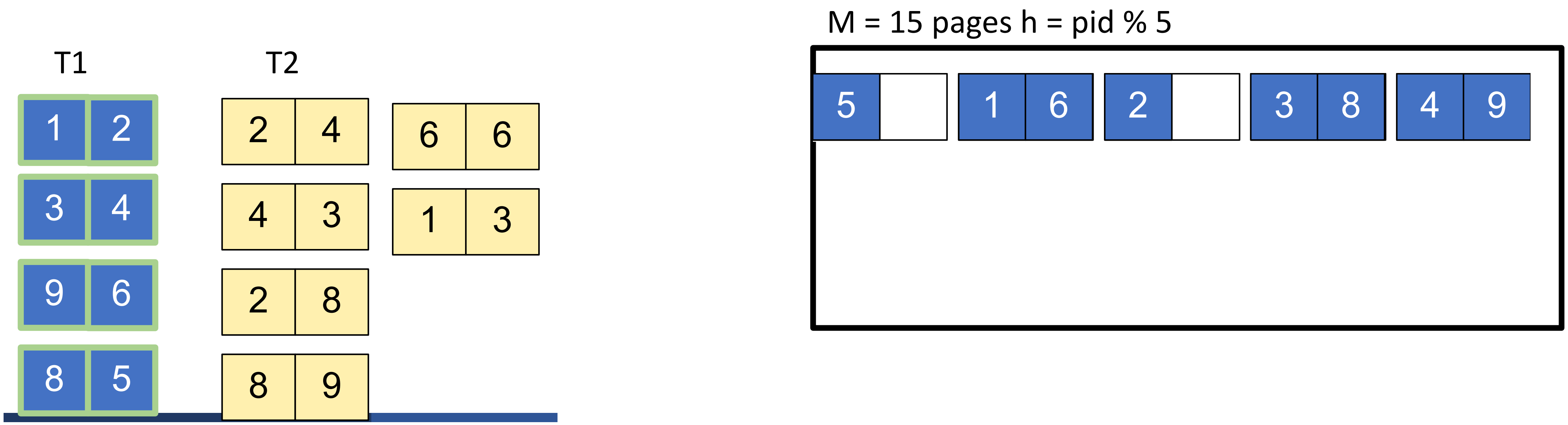
2	‘Blue’	123
4	‘Prem’	432

4	‘Prem’	343
5	‘GrpH’	554

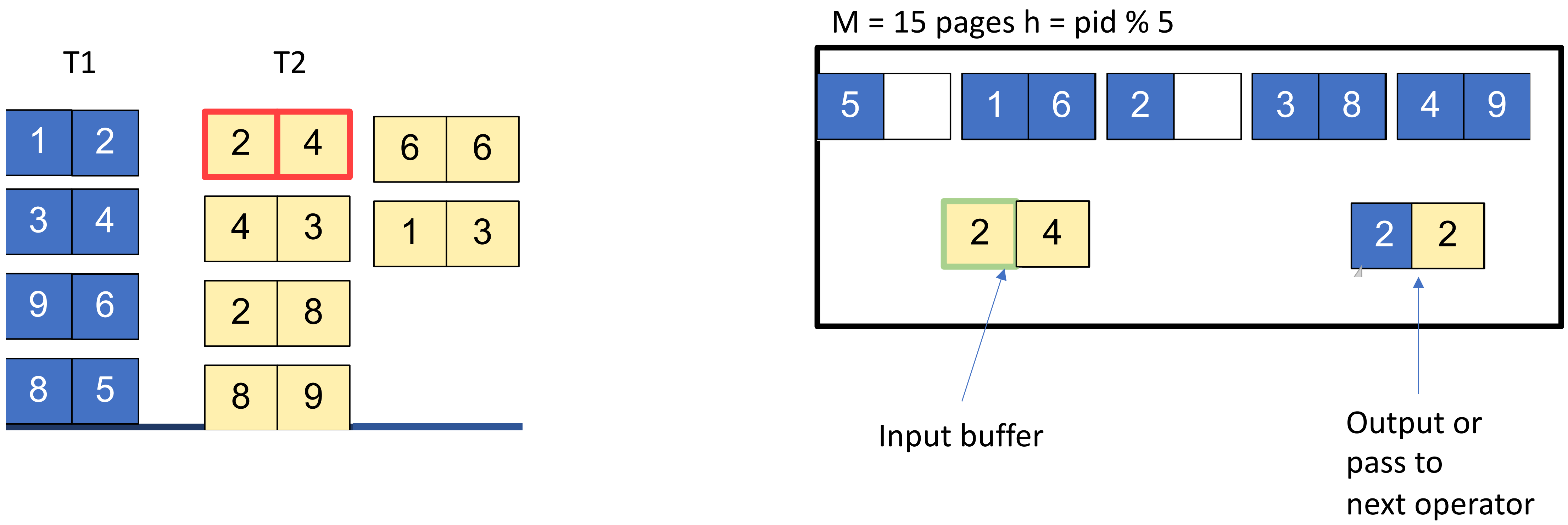
T1 ⋈ T2



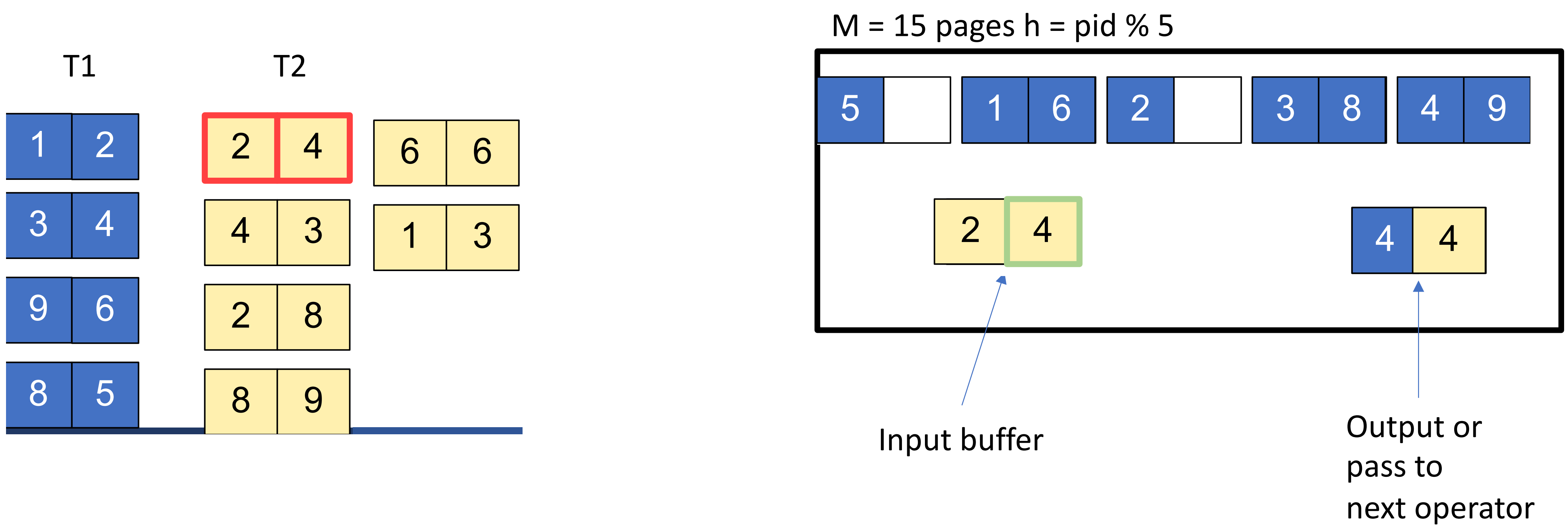
Scan T1 (open())



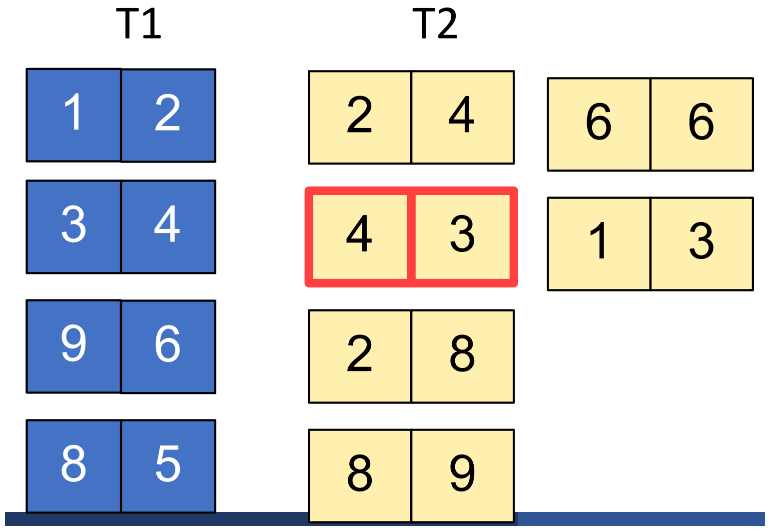
Scan T2 and probe into hash table (next())

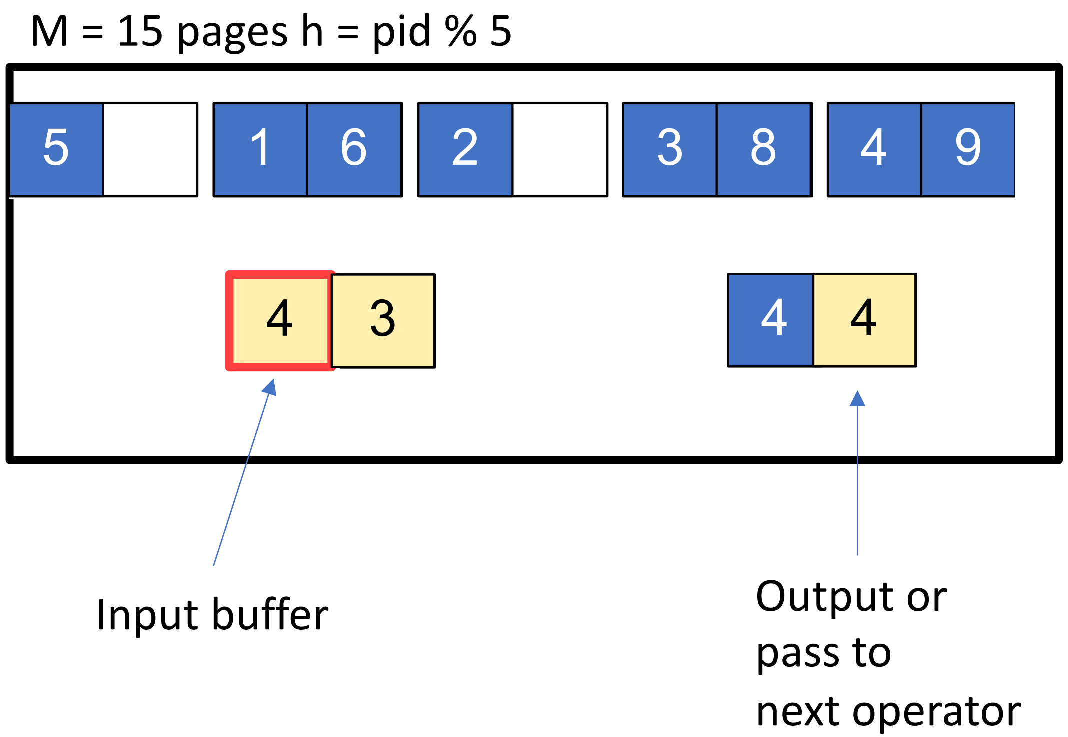


Scan T2 and probe into hash table (next())



Scan T2 and probe into hash table (next())





Join Algorithms

- Hash join : B(R) + B(S)
- **Nested loop join: B(R) + B(S) *T(R); B(R)+ B(S)B(R)**
- Sort-merge join

Nested Loop Join

- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

```
for each tuple t1 in R do
    for each tuple t2 in S do
        if t1 and t2 join then output (t1,t2)
```

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

Block refinement

```
for each block of tuples r in R do
    for each block 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)

Keep smaller relation between R and S as the outer one

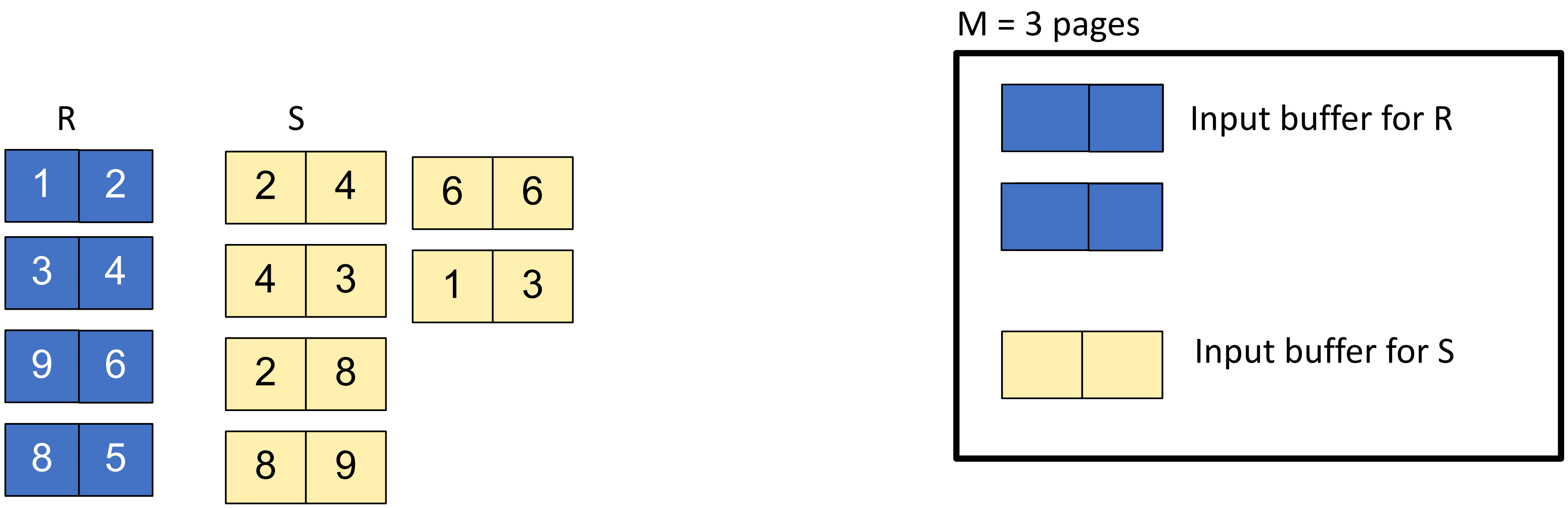
Group/Chunk-Block refinement

- for each group of M-1 blocks r in R do
  - for each block 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)

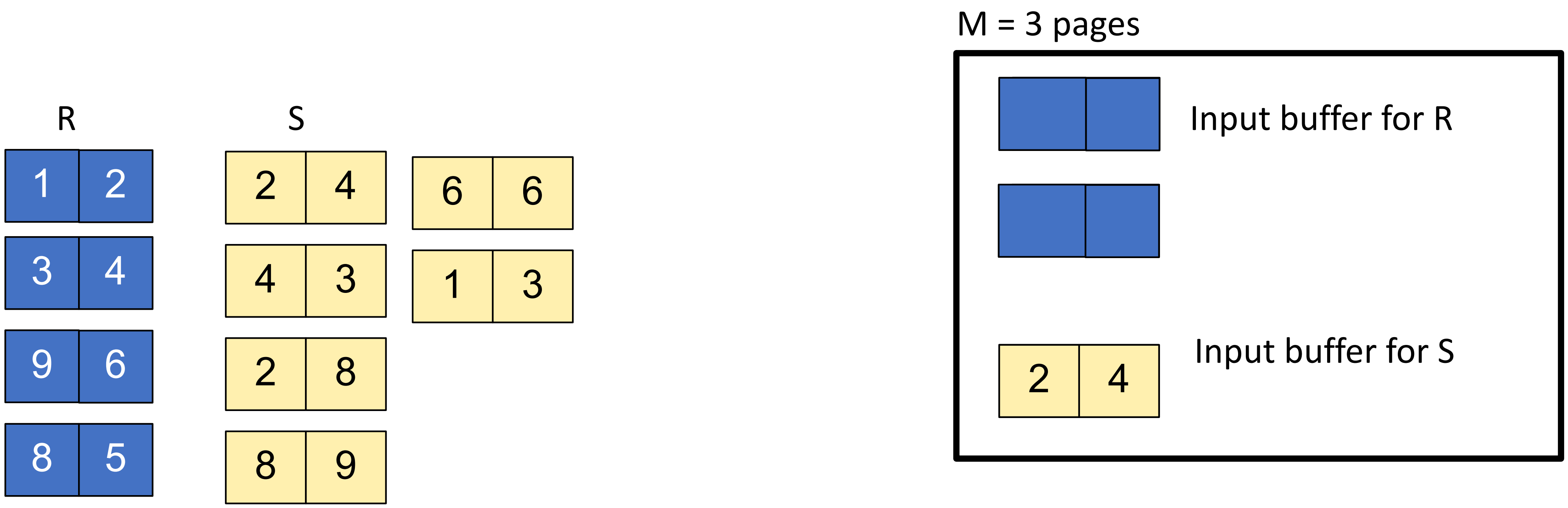
Cost: $B(R) + B(R)B(S)/(M-1)$

Key idea: Remember that the fewer times we read in S, the better.
Utilize the buffer more by reading several pages of R more.

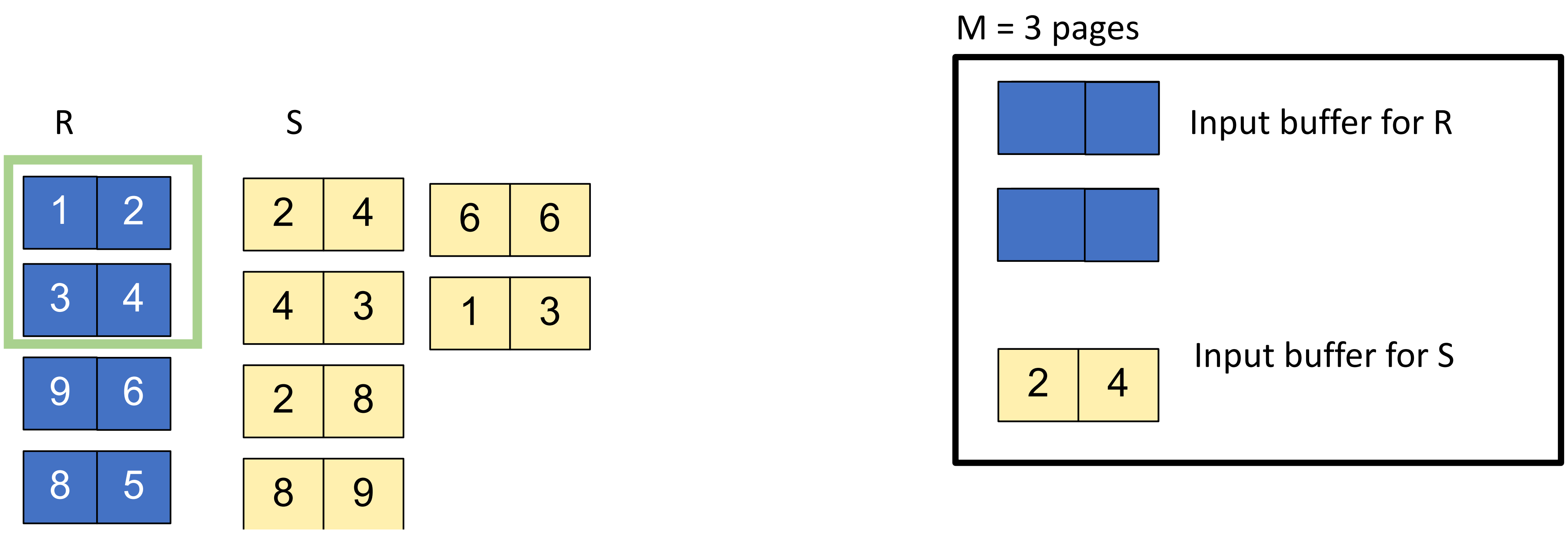
Group-based NLJ



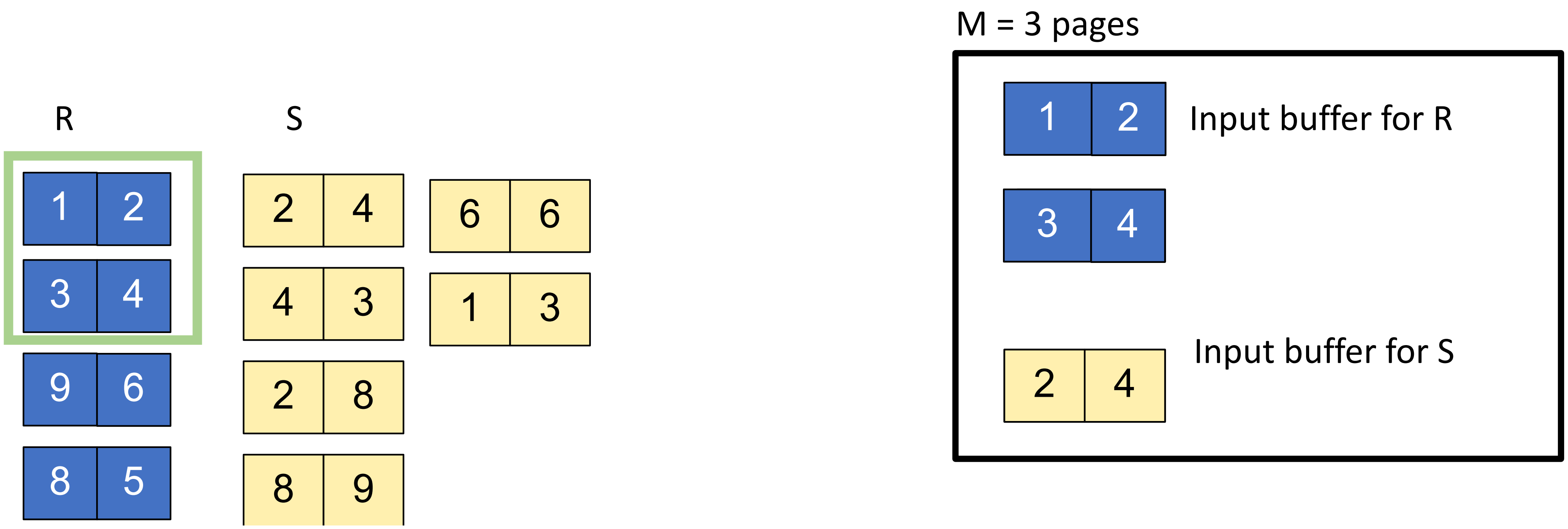
Block-based NLJ



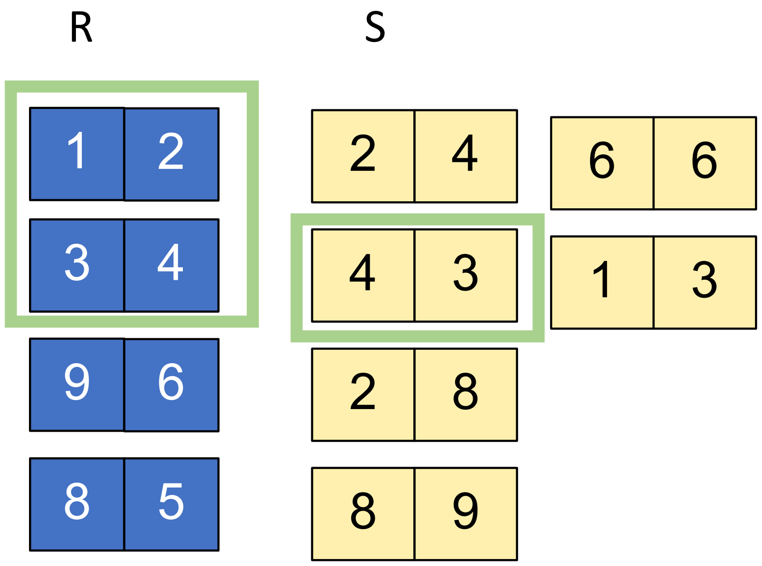
Block-based NLJ

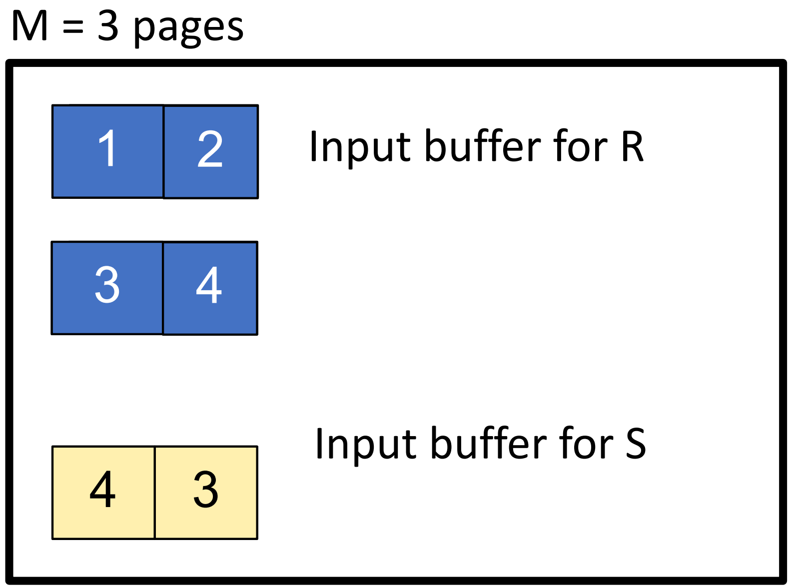


Block-based NLJ

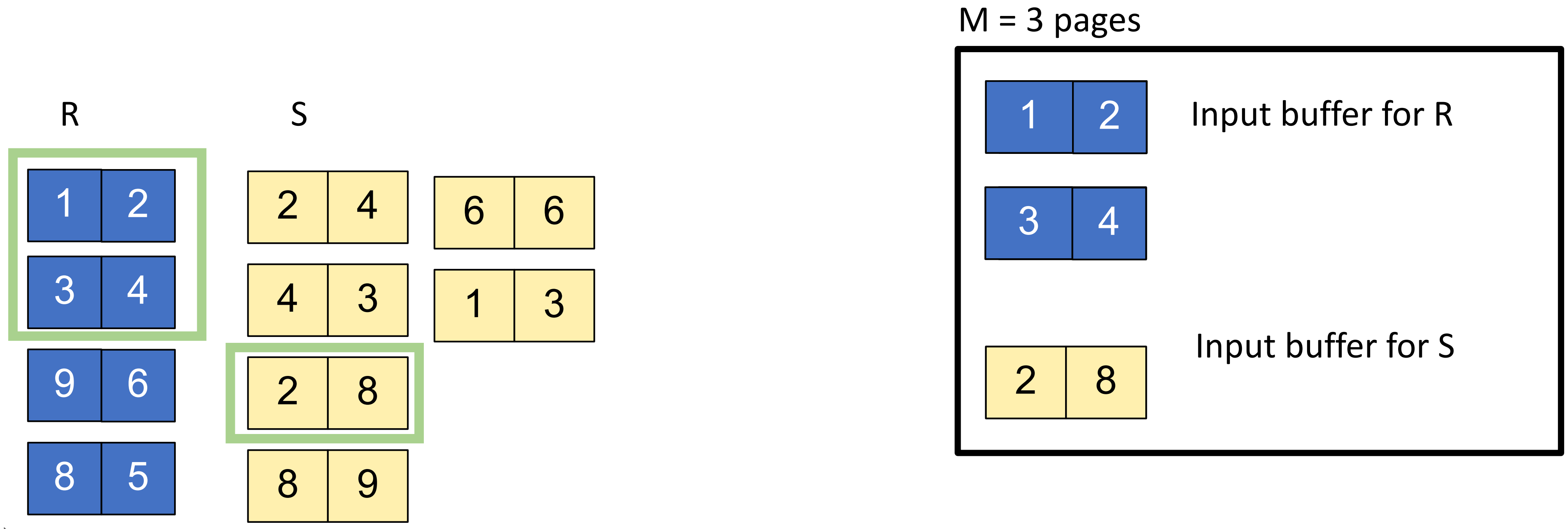


Block-based NLJ



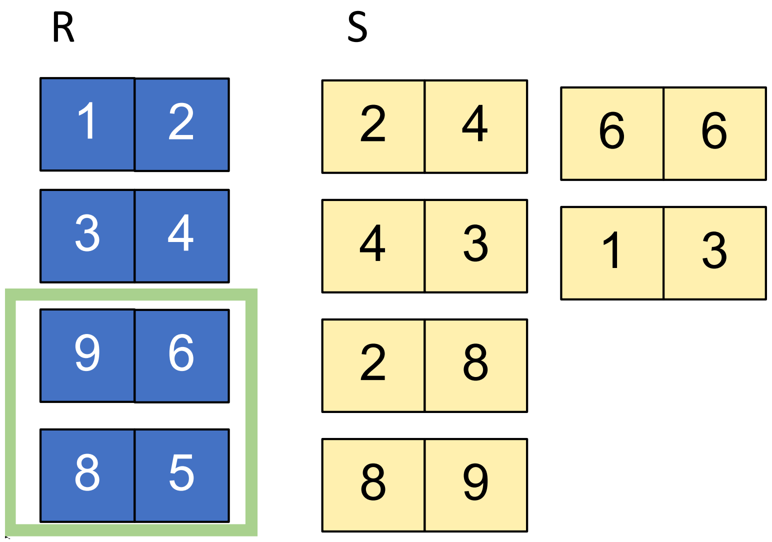


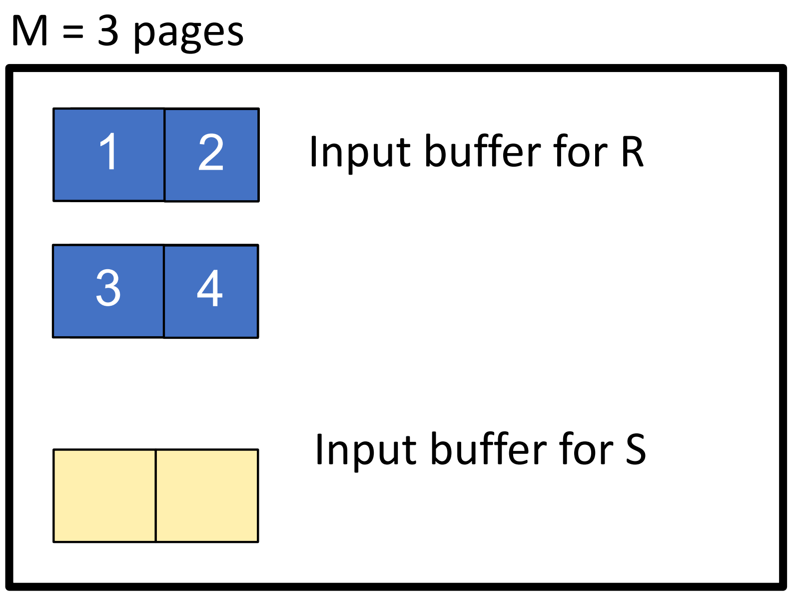
Block-based NLJ



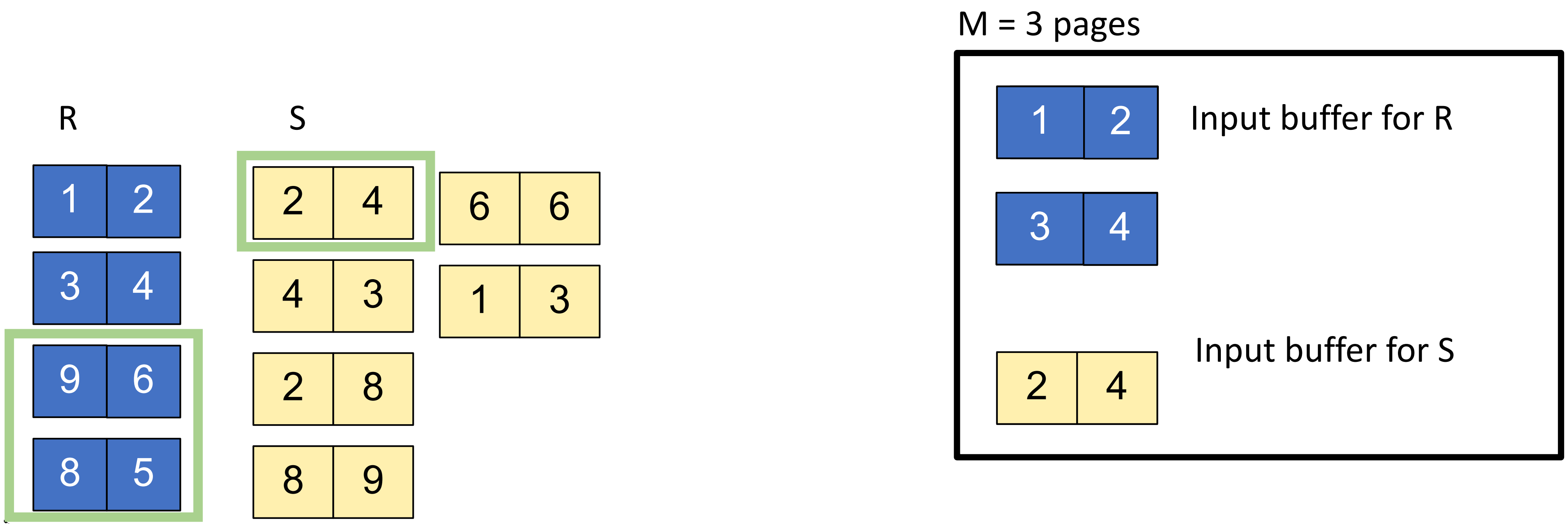
And so on till one scan of S is done.

Block-based NLJ





Block-based NLJ



Any further improvements?

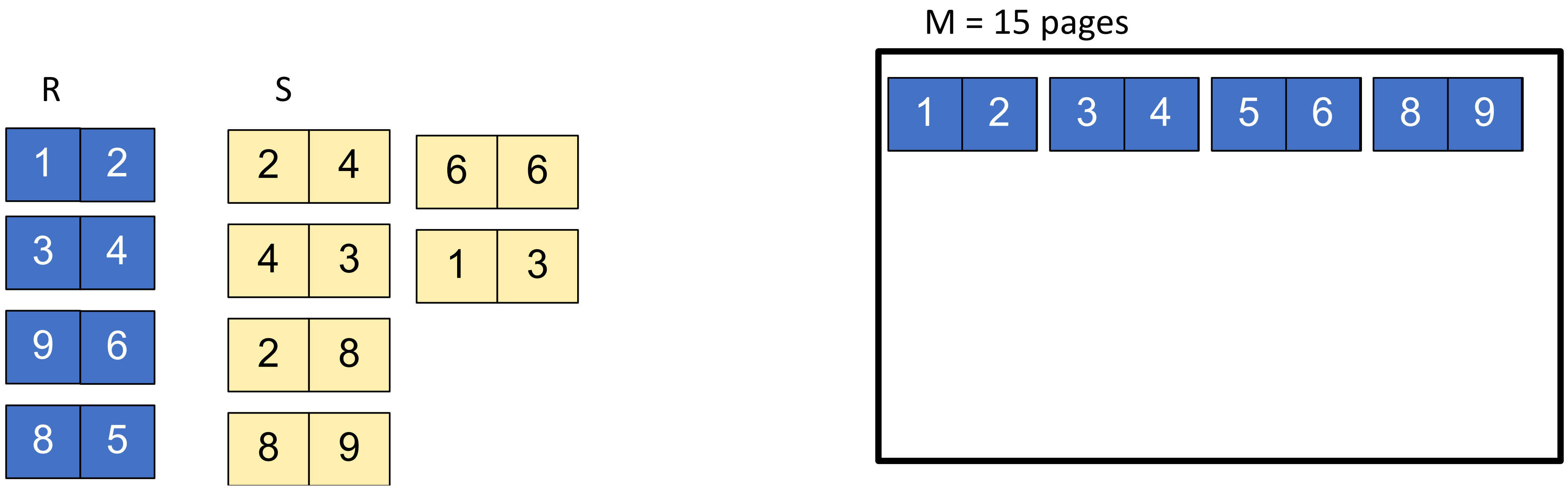
- Index-based NLJ
- An index on T2 that is on the appropriate field (i.e. the field we are joining on), it can be very fast to look up matches.

- Cost: The I/O cost is $B(R) + T(R) * $(cost to look up matching records in S).

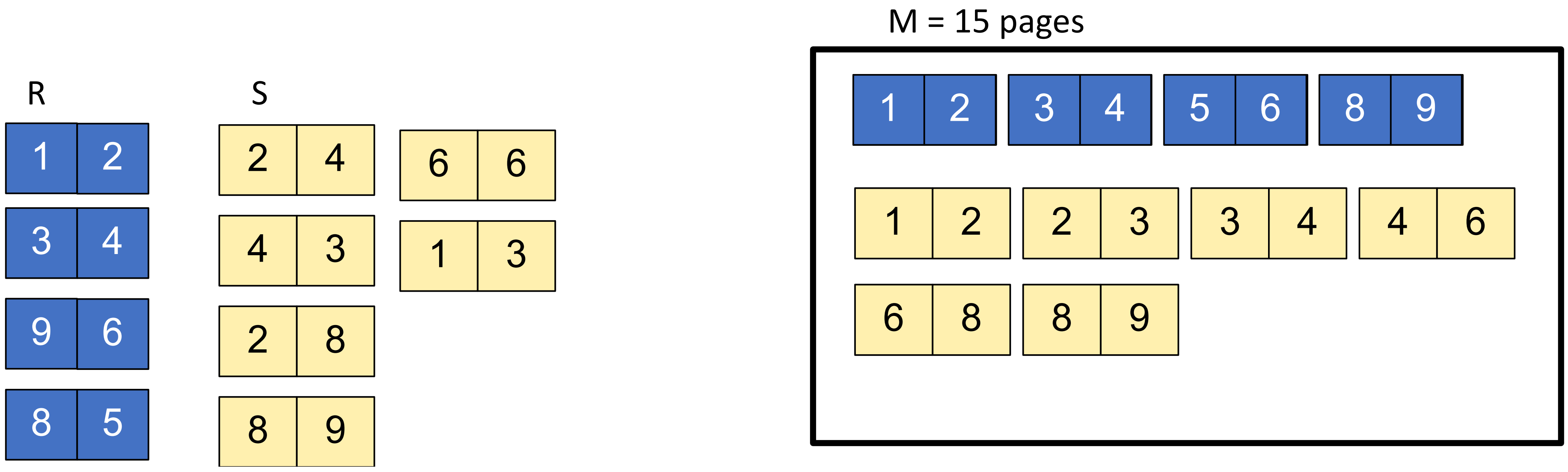
Sort-merge Join

- Sort-merge join: R ⋈ S
  - Scan R and sort in main memory
  - Scan S and sort in main memory
  - Merge R and S
- Cost: sort(R) + sort(S) + B(R) + B(S)
- One pass algorithm when B(S) + B(R) <= M
- Typically, this is NOT a one pass algorithm.

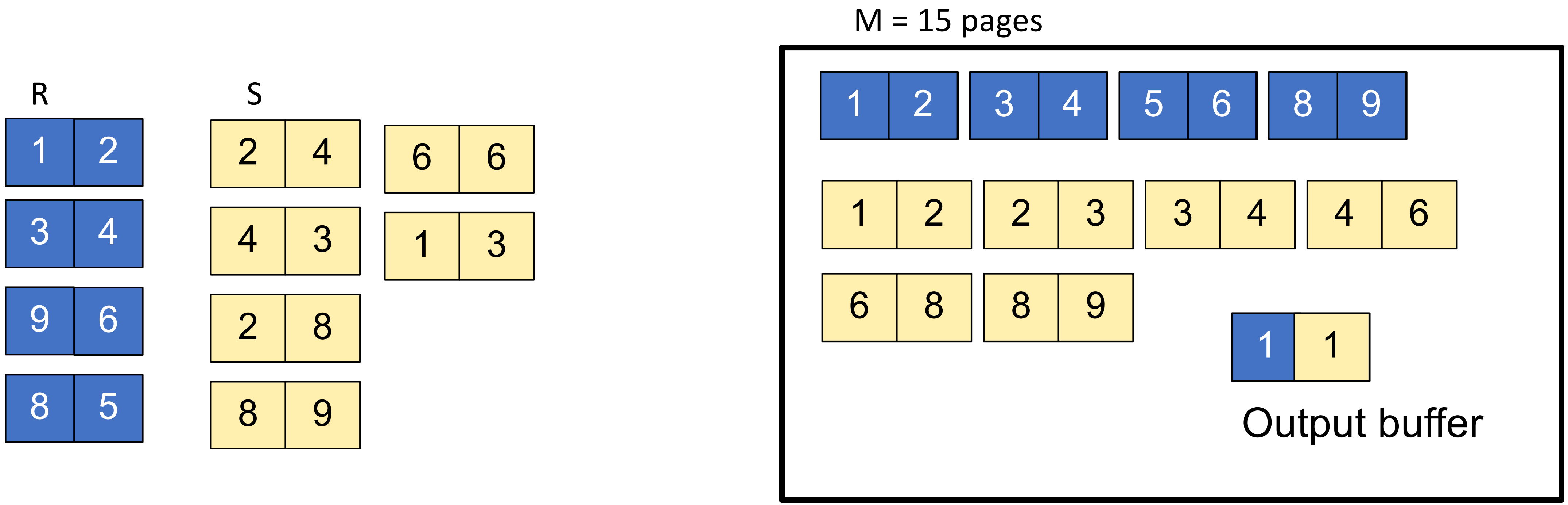
Scan T1 and sort in memory



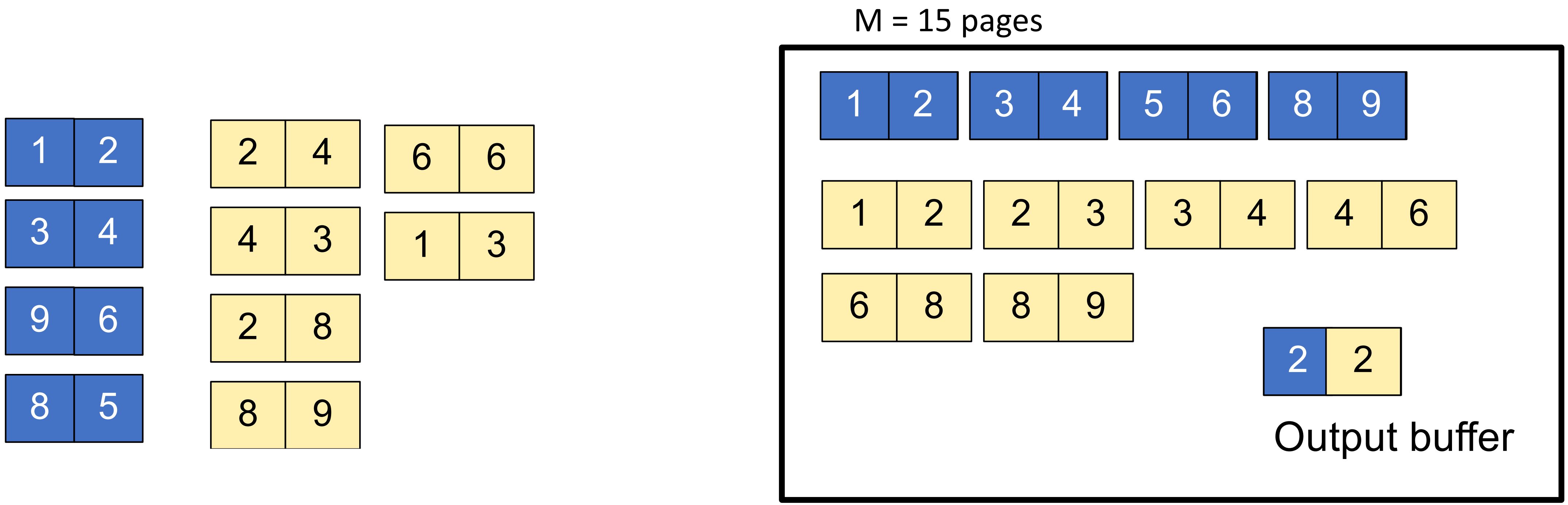
Scan T2 and sort in memory



Merge T1 and T2



Merge T1 and T2



Keep merging till on relation ends

Algorithm

- We begin at the start of R and S and advance one or the other until we get to a match
  - If $r_i < s_j$, advance R; else if $r_i > s_j$, advance S – the idea is to advance the lesser of the two until we get a match
- Let's say pair (r_i, s_j) is match. Mark this spot in S as marked(S) and check each subsequent record in S (s_j ,s_{j+1},s_{j+2}, etc) until we find something that is not a match (i.e. read in all records in S that match to r_i).
- Go to the next record in R and go back to the marked spot in S and begin again at step 1 (except instead of beginning at the start of R and the start of S, do it at the indices we just indicated)

Example

R

sid	sname
22	dustin
28	yuppy
31	lubber
31	lubber2
44	guppy
57	rusty

S

sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

sid	sname	bid
28	yuppy	103
28	yuppy	104

28 <31; advance S

R

sid	sname
22	dustin
28	yuppy
31	lubber
31	lubber2
44	guppy
57	rusty

S

sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

sid	sname	bid
28	yuppy	103
28	yuppy	104
31	lubber	101

Mark 31 (black arrow);
Output match

R

sid	sname
22	dustin
28	yuppy
31	lubber
31	lubber2
44	guppy
57	rusty

S

sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

sid	sname	bid
28	yuppy	103
28	yuppy	104
31	lubber	101
31	lubber	102

Advance S
Output match

R S

sid	sname
22	dustin
28	yuppy
31	lubber
31	lubber2
44	guppy
57	rusty

sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

sid	sname	bid
28	yuppy	103
28	yuppy	104
31	lubber	101
31	lubber	102

Advance S
Mismatch
Reset S
Advance R
Another match

R S

sid	sname
22	dustin
28	yuppy
31	lubber
31	lubber2
44	guppy
57	rusty

sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

sid	sname	bid
28	yuppy	103
28	yuppy	104
31	lubber	101
31	lubber	102
31	lubber2	101

R S

sid	sname
22	dustin
28	yuppy
31	lubber
31	lubber2
44	guppy
57	rusty

sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

sid	sname	bid
28	yuppy	103
28	yuppy	104
31	lubber	101
31	lubber	102
31	lubber2	101
31	lubber2	102

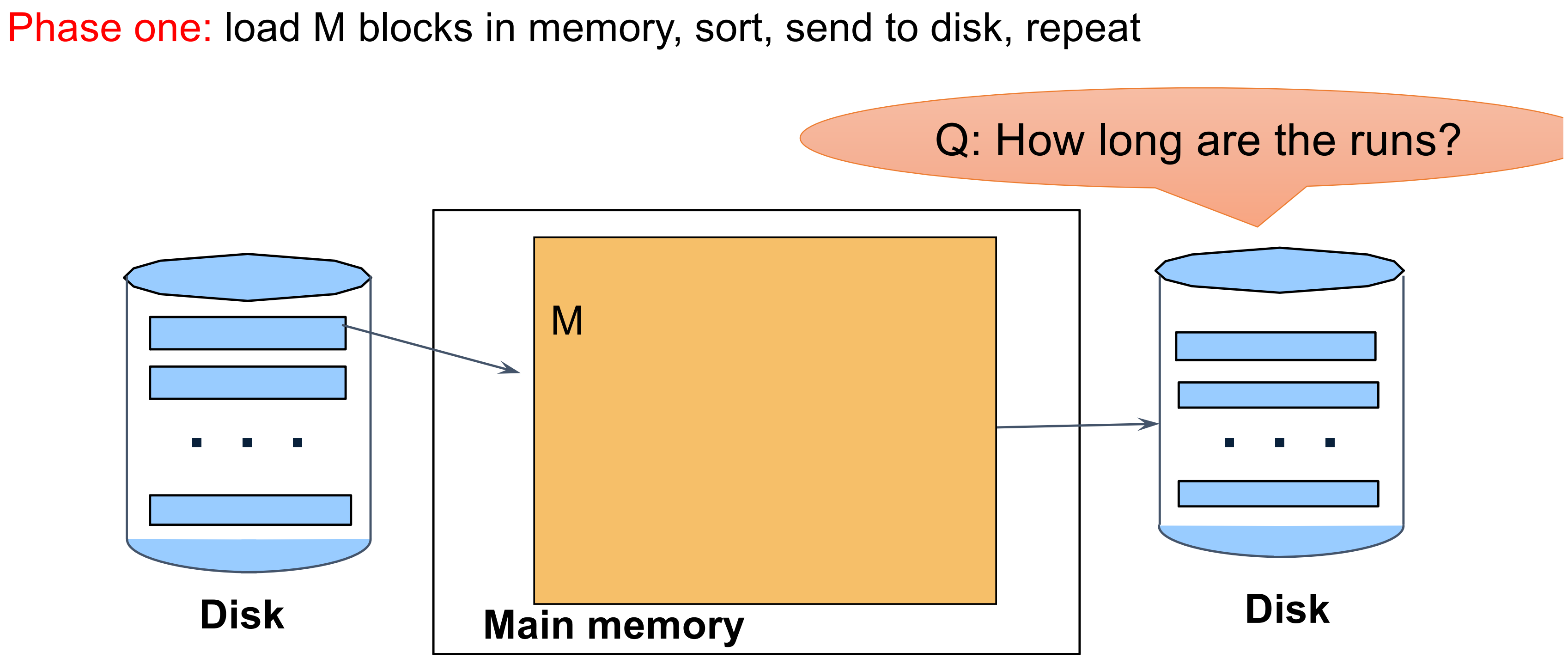
Advance S
Output Match

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

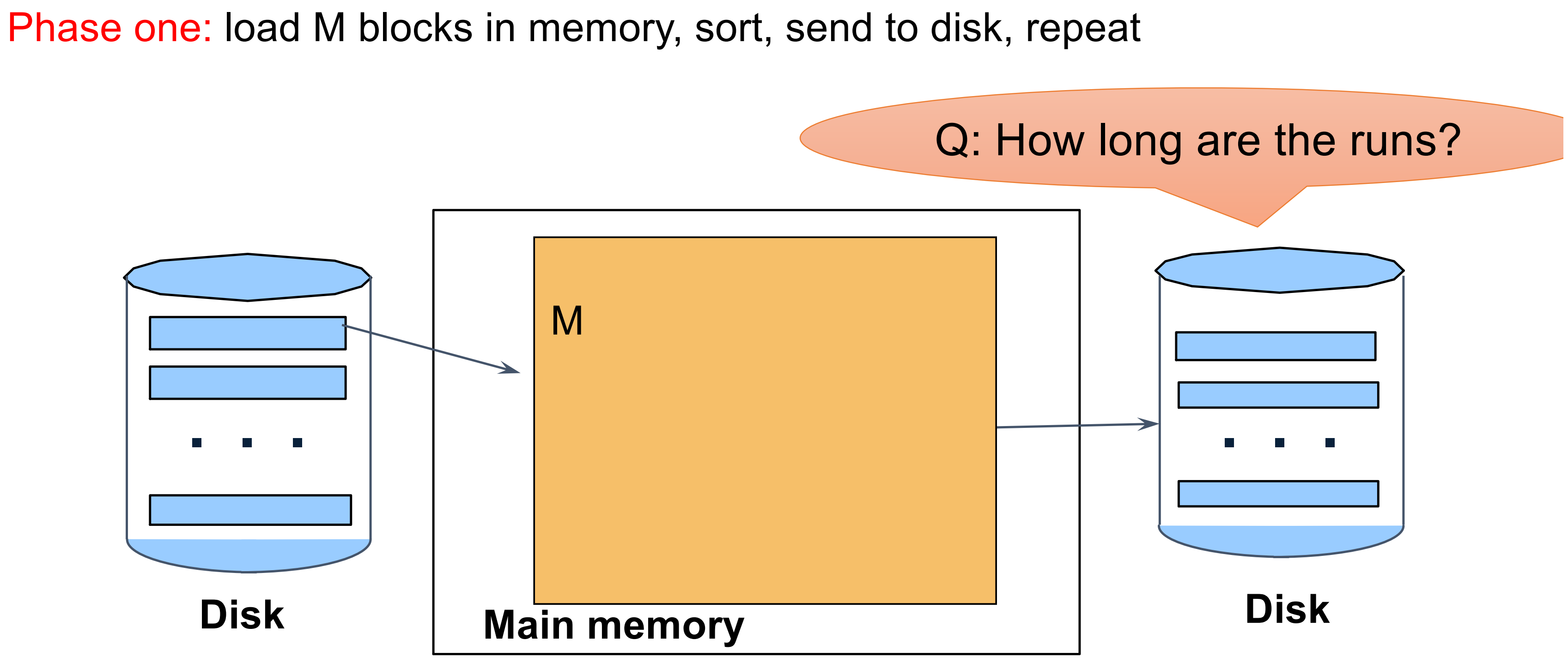
External Sort Merge

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



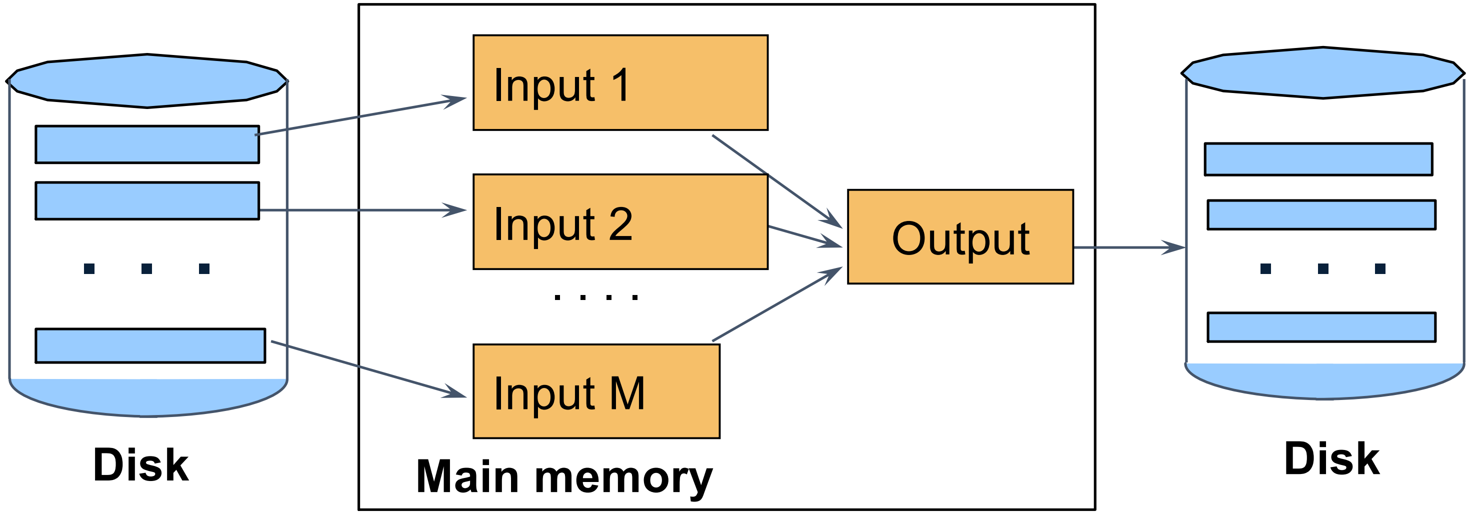
External Sort Merge---M block long runs

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



External Sort Merge

- **Phase two:** merge M runs into a bigger run
- In effect Merge M – 1 runs into a new run and 1 for output buffer.



External Sort Merge

- 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 Sort-Merge: M-way Merge

- Use M blocks of memory to buffer (sorted) input runs. Reserve 1 block to buffer output
- Repeat until done
  - Select next record from one of the buffer pages
  - Write that record to output buffer (if the output buffer is full, write the page to disk)
  - Delete the processed record from the buffer
  - If buffer is empty, read the next block (in that run)

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

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

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

Short video to watch

- https://www.youtube.com/watch?v=1dtIutGlSsQ

Example

- Sort table T = 1960 pages with 8 available buffers

- Questions
  - How many sorted runs will be produced after each pass?
  - How many pages will be in each sorted run for each pass?
  - How many I/Os does the entire sorting operation take?

Example

- Sort table T = 1960 pages with 8 available buffers

- Questions
  - How many sorted runs will be produced after each pass?

  1st pass = 1960/8 = 245 sorted runs of 8 pages each

  Subsequent passes 7 pages each.

  2nd pass = 245/7 = 35 sorted runs of 8*7 = 56 pages

  3rd pass = 35/7 = 5 sorted runs of 56*7 = 392 pages

  4th pass = can merge all remaining sorted runs since less than 7 sorted runs

  Produces one sorted run of 1960 pages.

Example

- Sort table T = 1960 pages with 8 available buffers

- Questions
  - How many sorted runs will be produced after each pass?
    - 245, 35, 5, 1
  - How many pages will be in each sorted run for each pass?
    - 8, 56, 392, 1960
  - How many I/Os does the entire sorting operation take?
    - Easch pass takes 2*N I/Os = 4*2*1960 = 15,680

Approximation to Cost

- Approximately: Read+ write+ read = 3B(R) in each pass (without storing final output to disk)
  - B(R) to read B blocks
  - B (R) to write sorted sublists
  - Again read all sorted sublists.

How large a table can be sorted?

- Observation 1: For external sort-merge to work there must not be more than M-1 runs.
- Observation 2: Each run is M blocks long.

- Suppose R fits in B blocks, then M* (M-1) >= B
- If approx. B <= M^2 then we are done

Size of R

- Assumption: $B(R) <= M^2$
  - How large can R be?
  - Suppose blocks are 64K = 2^{16} bytes and main memory is 1GB = 2^{30} bytes
  - M = ?
  - M^2 = ?
  - Size of Relation = ?

Using Ext. Sort Merge in Join

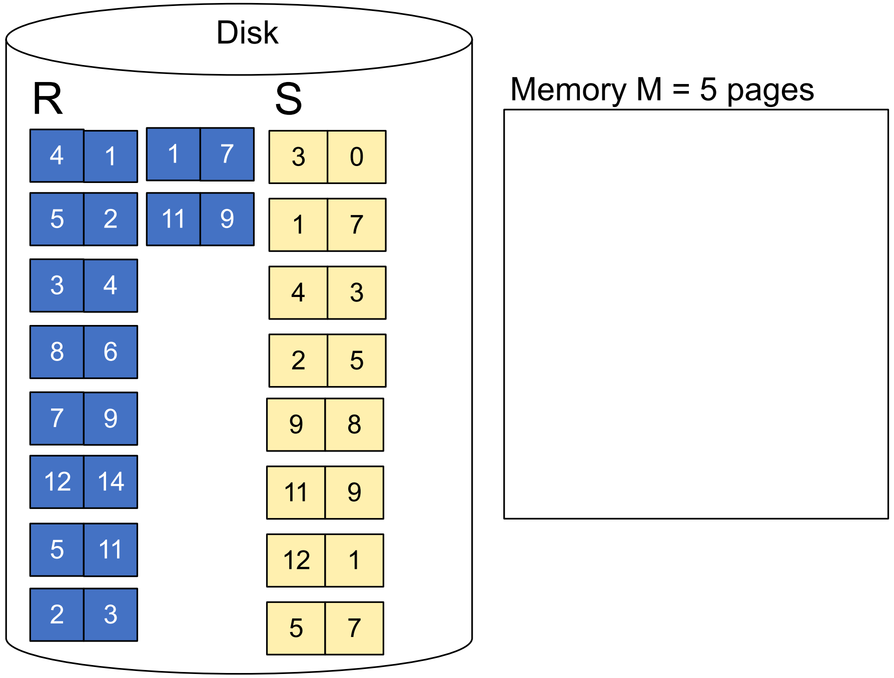
- How?

Using Ext. Sort Merge in Join

- Step 1a: generate initial runs for R (X,Y)
- Step 1b: generate initial runs for S (Y,Z)
- Step 2: merge and join
  - Either merge first and then join
  - Or merge & join at the same time
- Repeat
- Find the least value y of Y that is currently in front of R and S.
- If y does not appear at the front of other relation, then remove the tuple with y
- Else, identify all the tuples from both relations having sort key y. If necessary, read blocks and R and S until no further blocks. Maximum buffers available = M.
- Output all tuples.

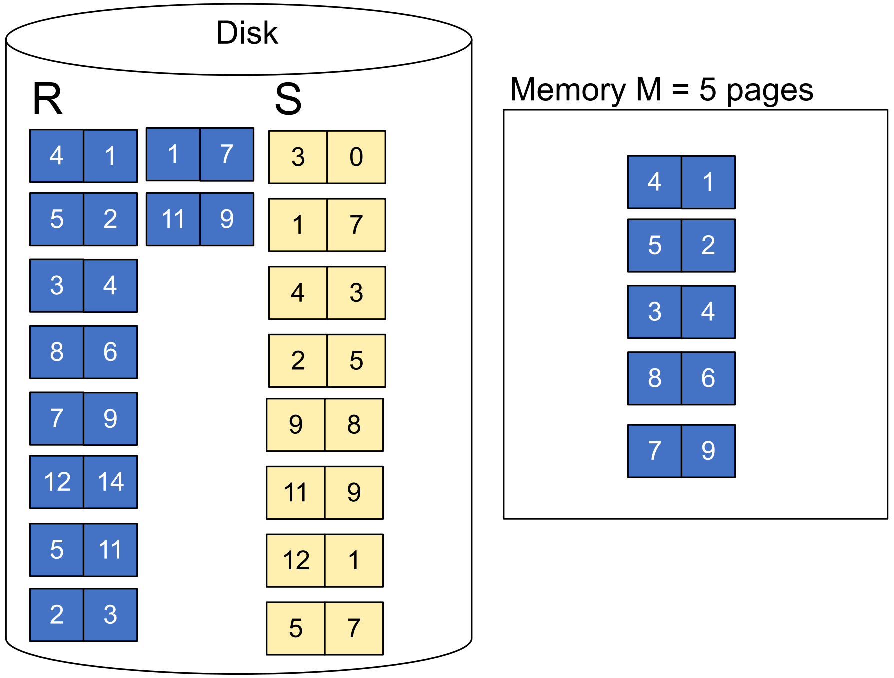
Example

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



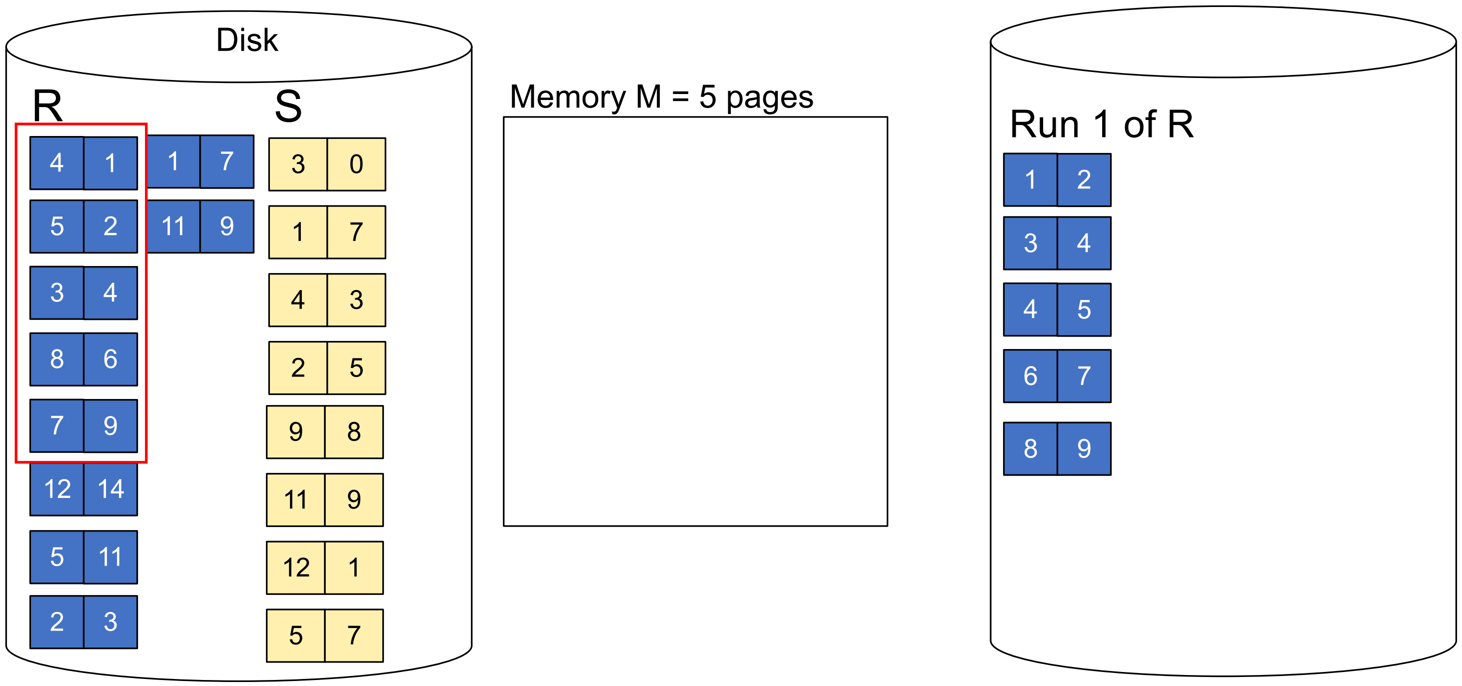
Example

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



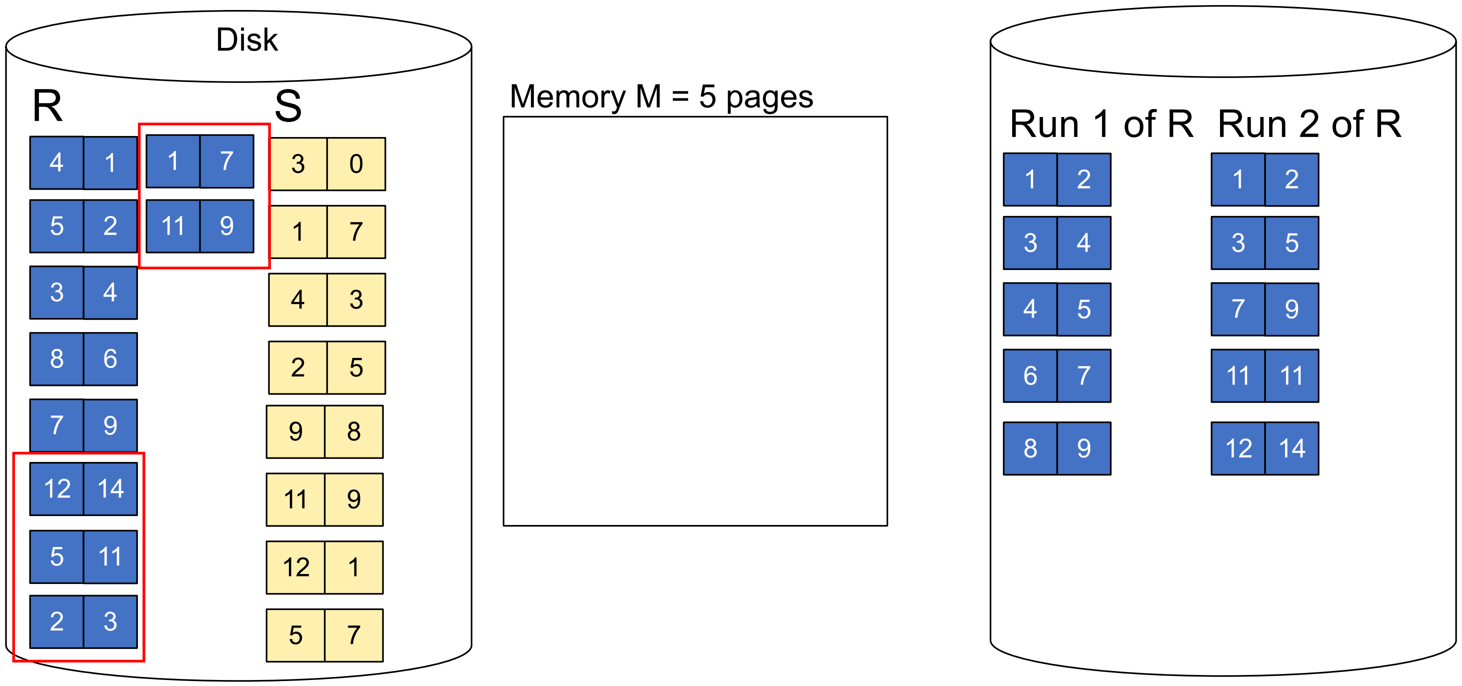
Example

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



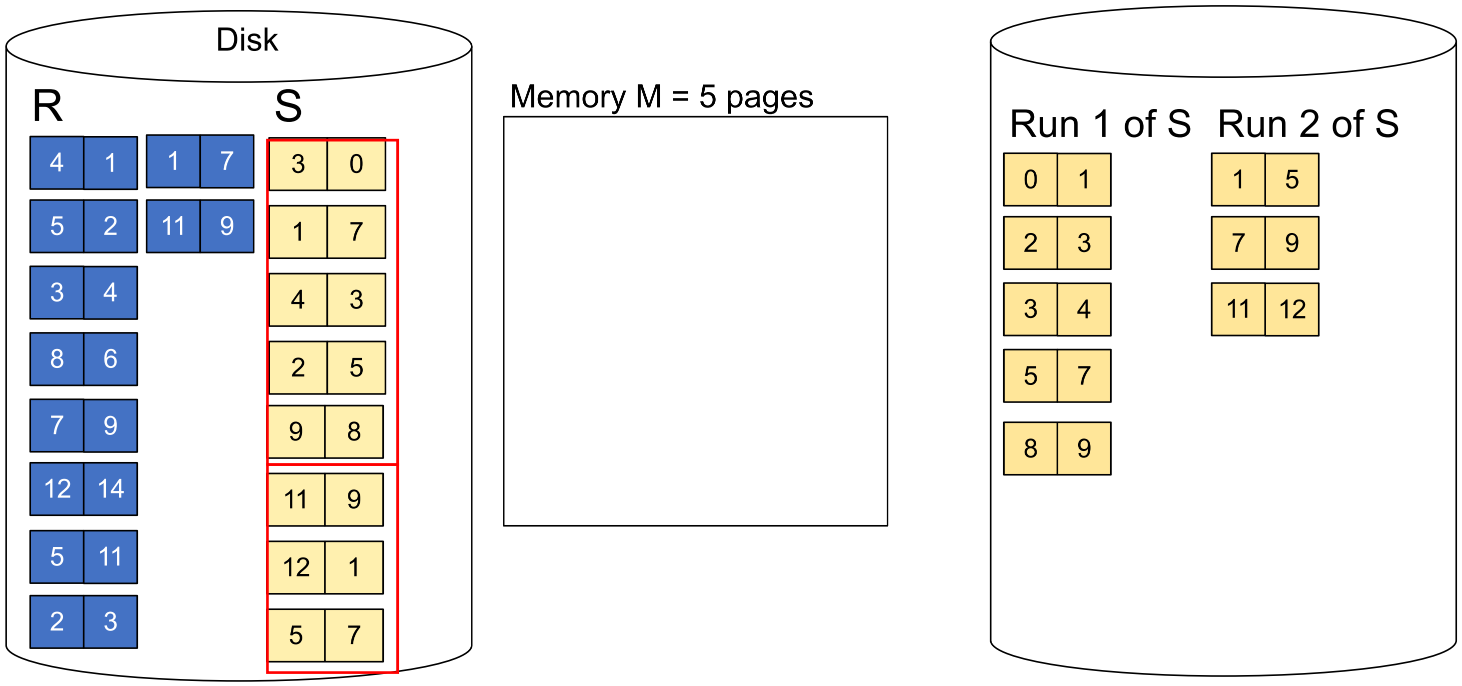
Example

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



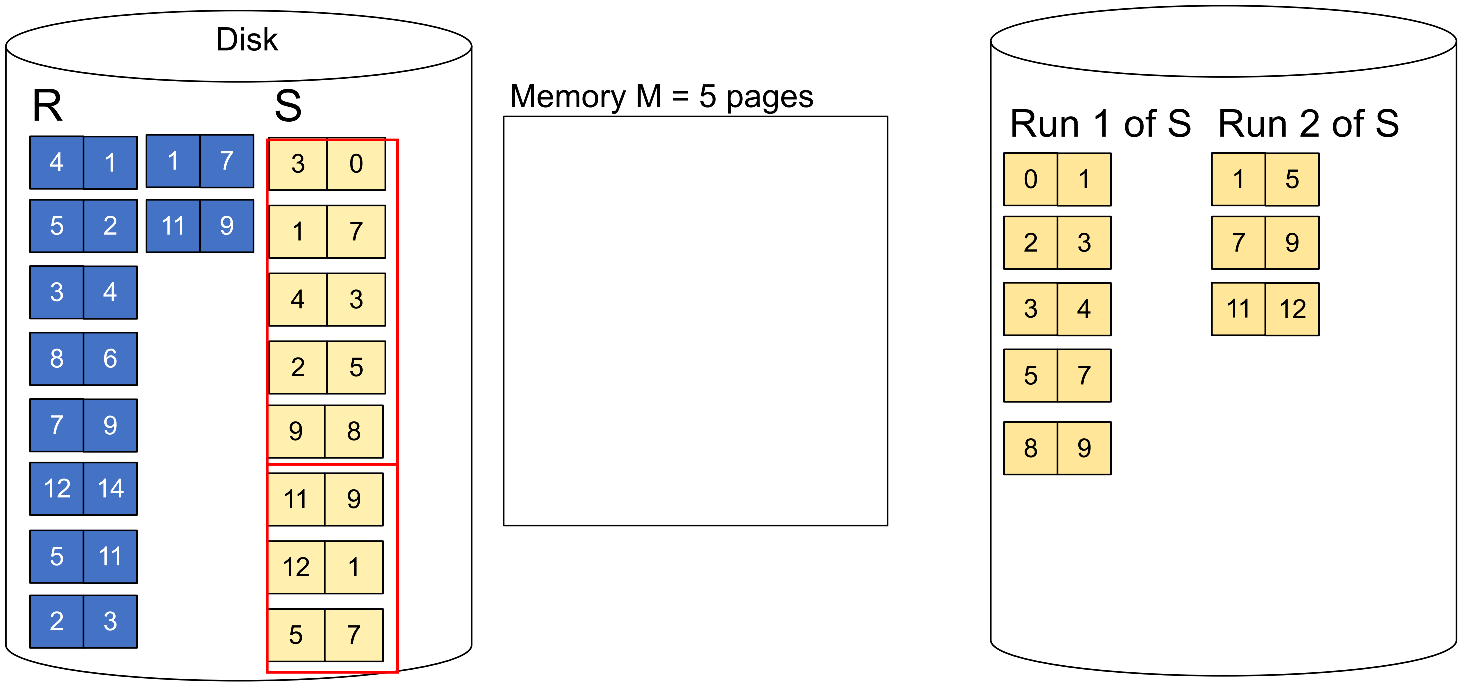
Example

Step 1: Do the same with S



Example

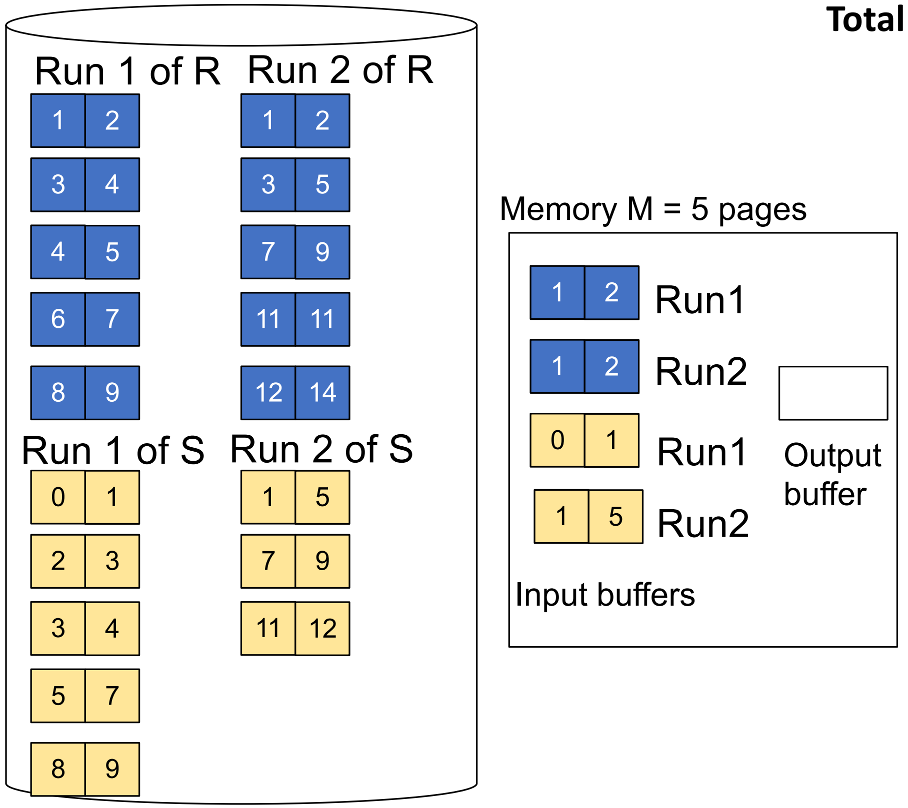
Step 1: Do the same with S



Example

Step 2: Join while merging sorted runs

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

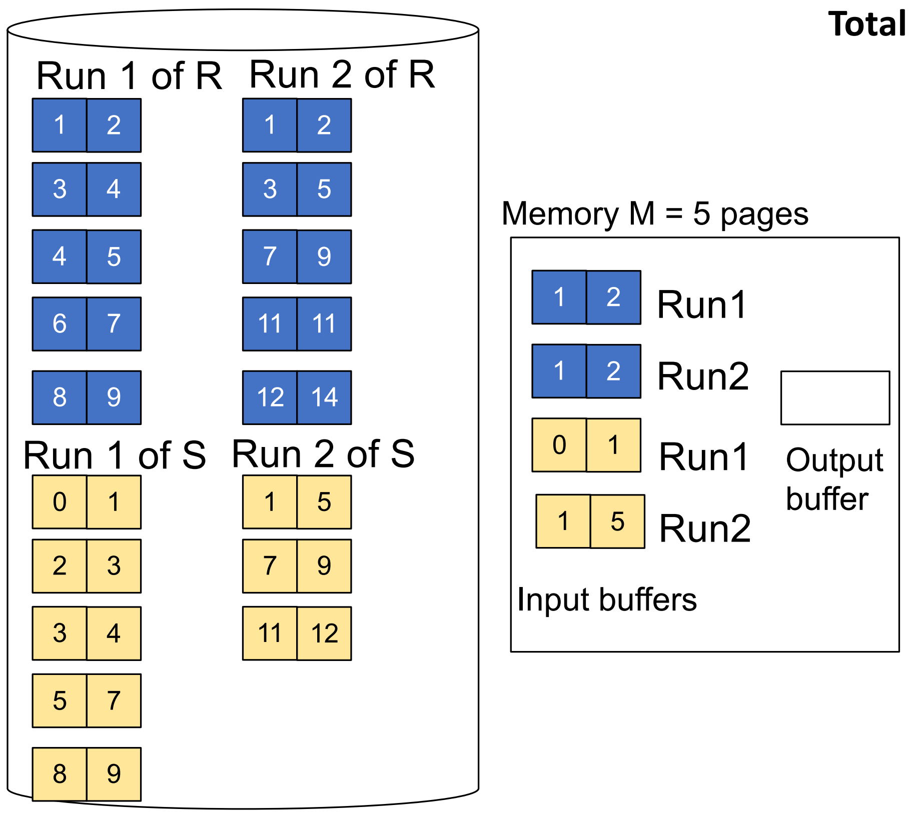


Step 2: Join while merging
Output tuples

Example

Step 2: Join while merging sorted runs

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



Step 2: Join while merging

Output tuples

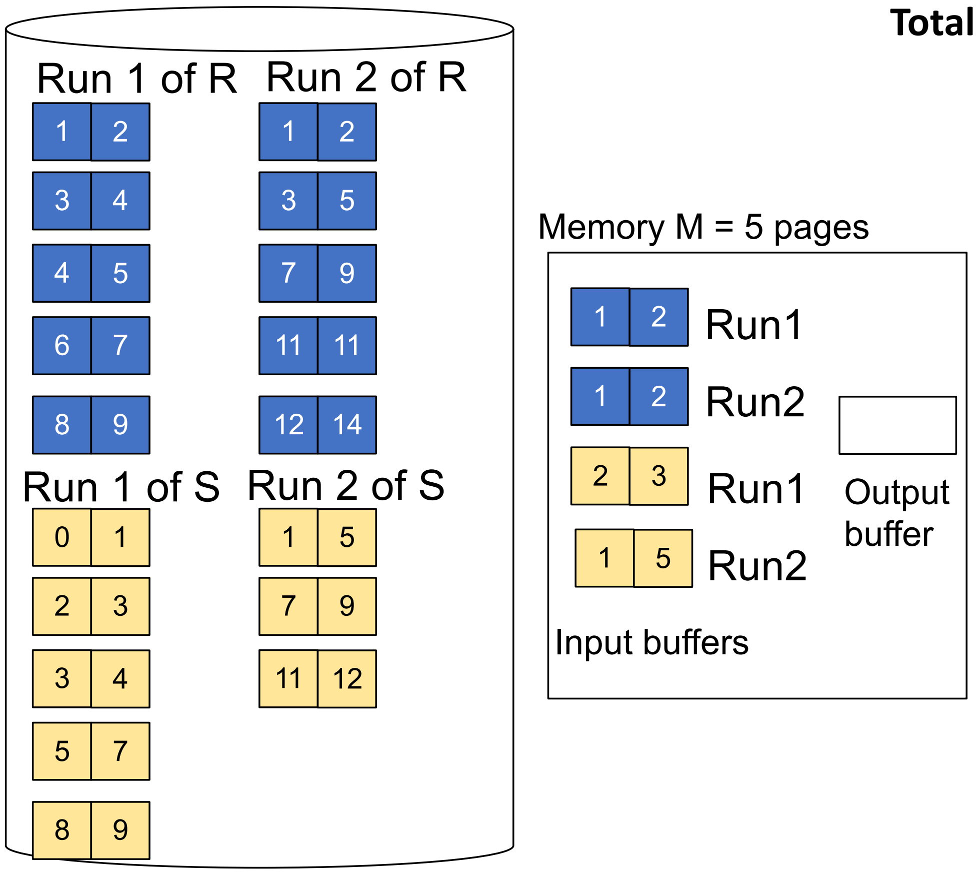
(1,1)

(1,1)

(1,1)

(1,1)

Example



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

Step 2: Join while merging

Output tuples

(1,1)

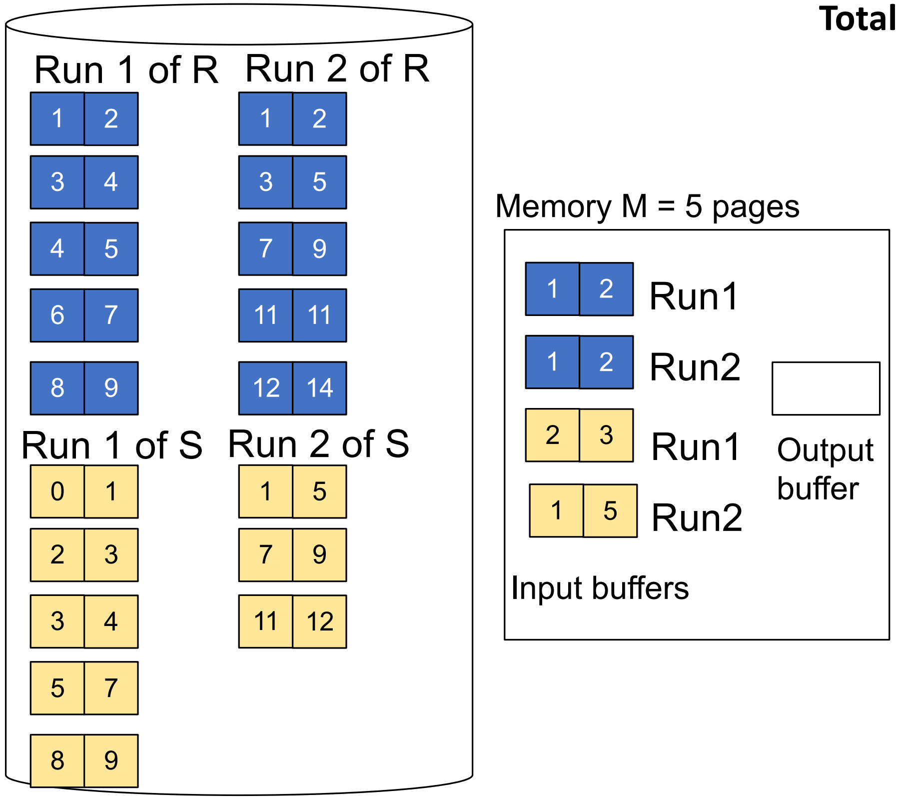
(1,1)

(1,1)

(1,1)

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)

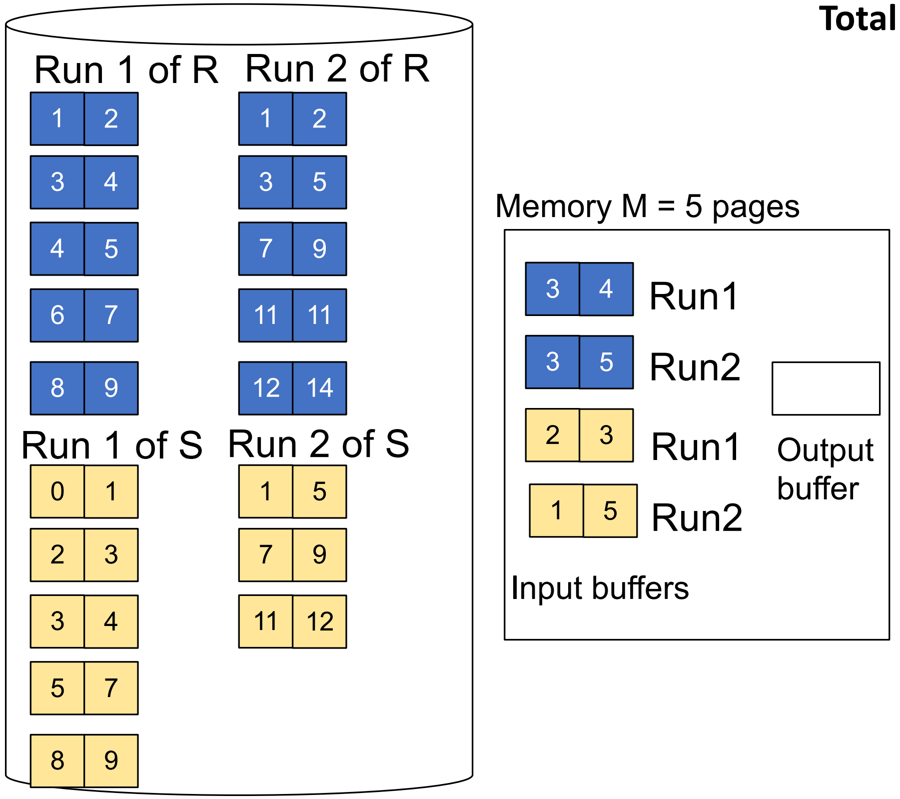
(1,1)

(2,2)

(2,2)

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)

…

Cost

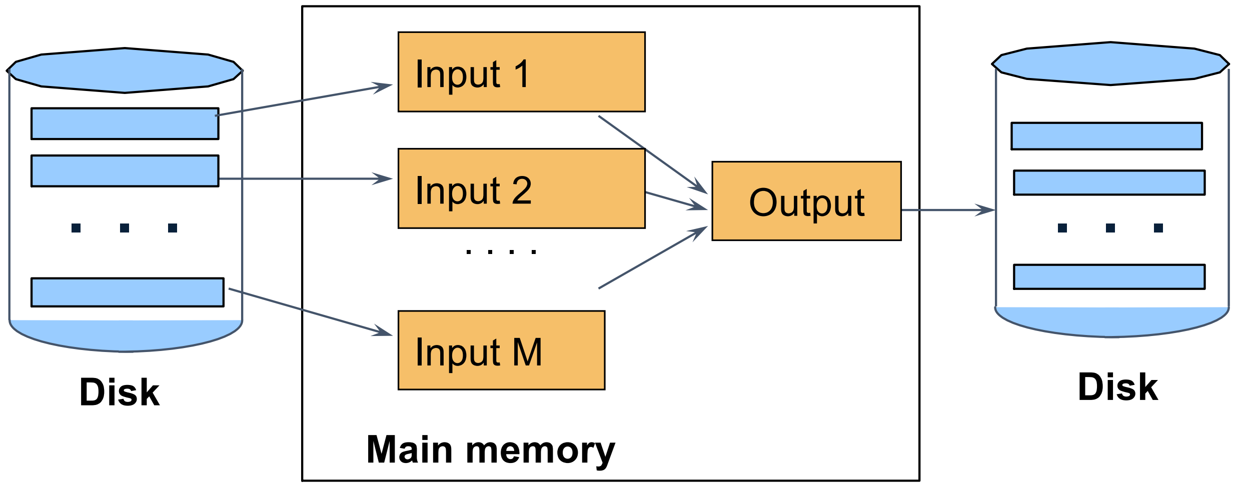
- Sort and write S to disk: 4(B(S))
- Sort and write R to disk: 4(B(R))
- Read and merge = B(R) + B(S)
- Total = 5(B(R) + B(S))
- With B(R) <= M^2 and B(S) <= M^2

Example

- R = 1000 blocks
- S = 500 blocks
- M = 101 buffers

- Note some more savings can be accrued by combining the second phase of sorting with the join itself.

Example



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. $B(R) + B(S) <= M^2$

Main memory and disk I/O required for sort based algorithms.

Operators	Approximate M required	Disk I/O	Section
τ, γ, δ	$\sqrt{B}$	$3B$	15.4.1, 15.4.2, 15.4.3
$\cup$, $\cap$, $-$	$\sqrt{B(R)+B(S)}$	$3\big(B(R)+B(S)\big)$	15.4.4, 15.4.5
$\bowtie$	$\sqrt{\max\big(B(R), B(S)\big)}$	$5\big(B(R)+B(S)\big)$	15.4.6
$\bowtie$	$\sqrt{B(R)+B(S)}$	$3\big(B(R)+B(S)\big)$	15.4.8

Two-Pass Algorithms

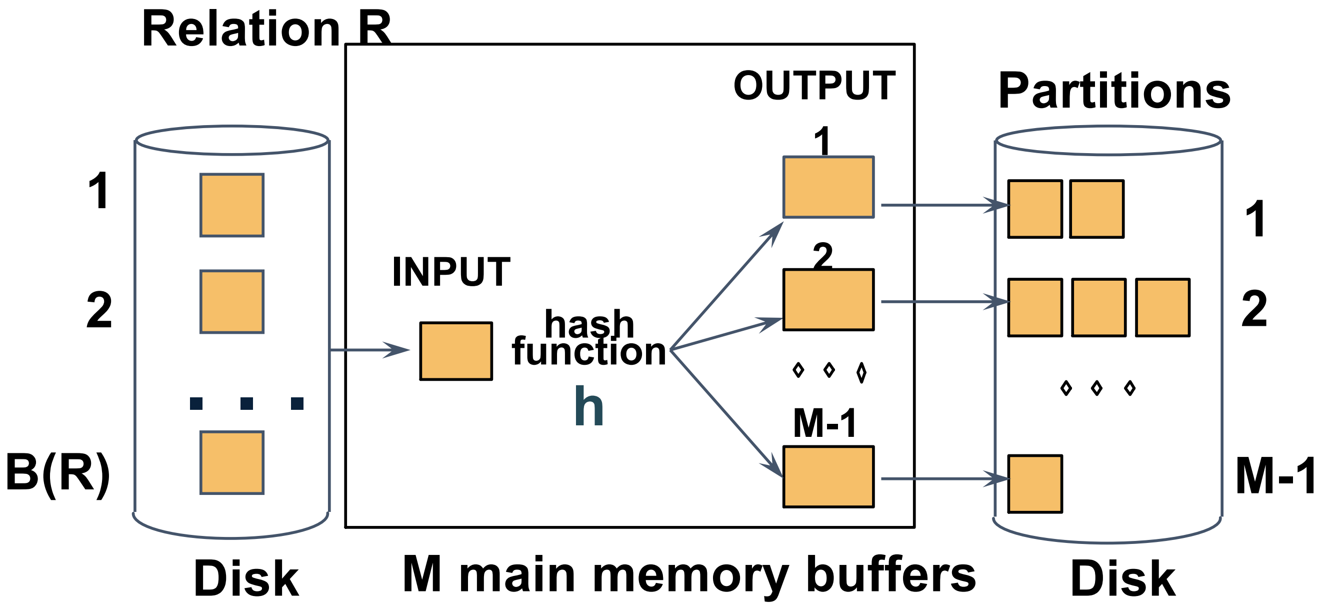
- What if data does not fit in memory?
- Need to process it in multiple passes
- Two key techniques
  - Sorting
  - **Hashing**

Partitioned Hash

- Partition R it into k buckets: R_1, R_2, R_3, …, R_k
- Assuming $B(R_1)=B(R_2)=…= B(R_k)$, 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?

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



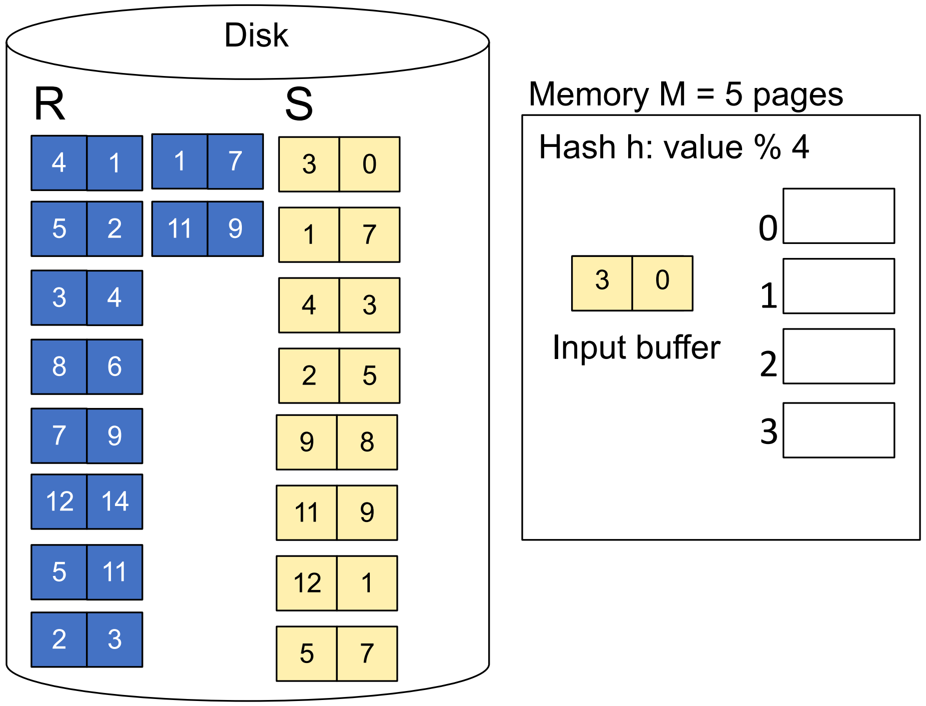
Assumption: $B(R)/M \leq M$, i.e. $B(R) \leq M^2$

Partitioned Hash Join--Algorithm

- 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

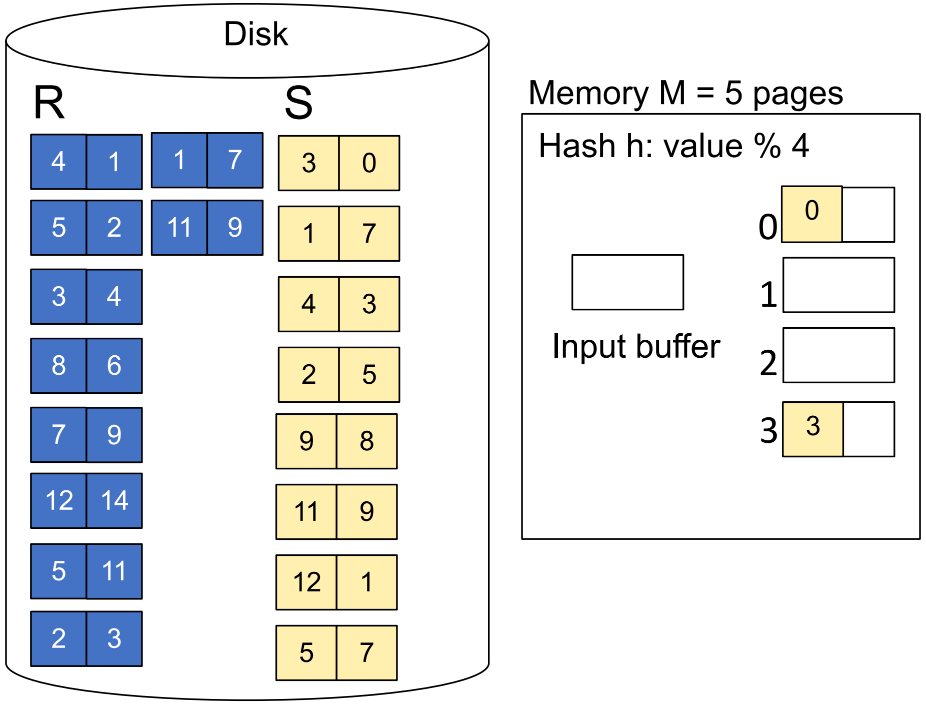
Example

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



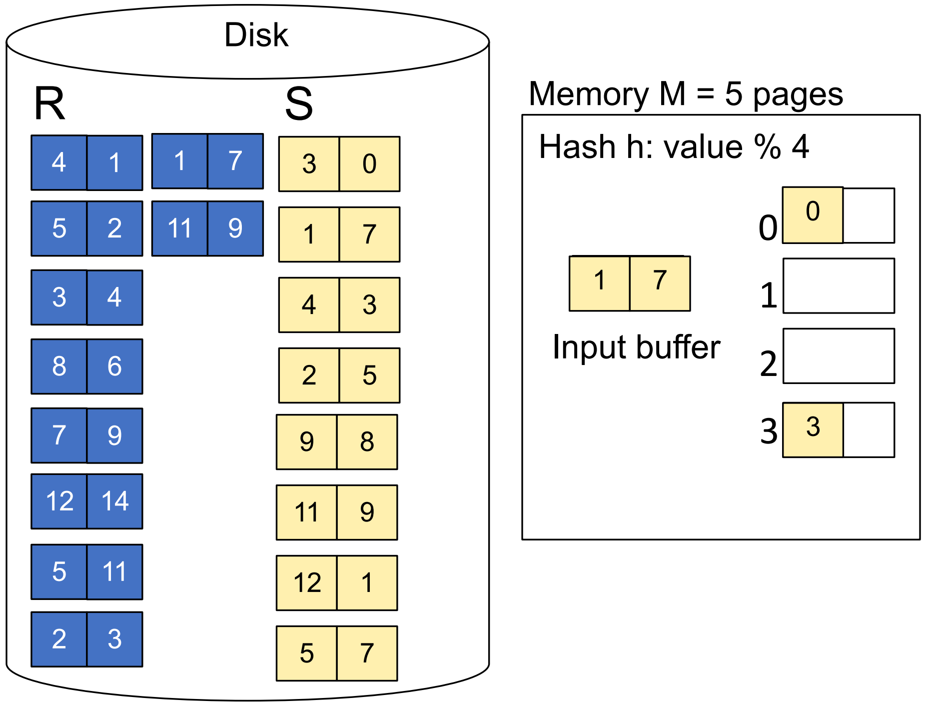
Example

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



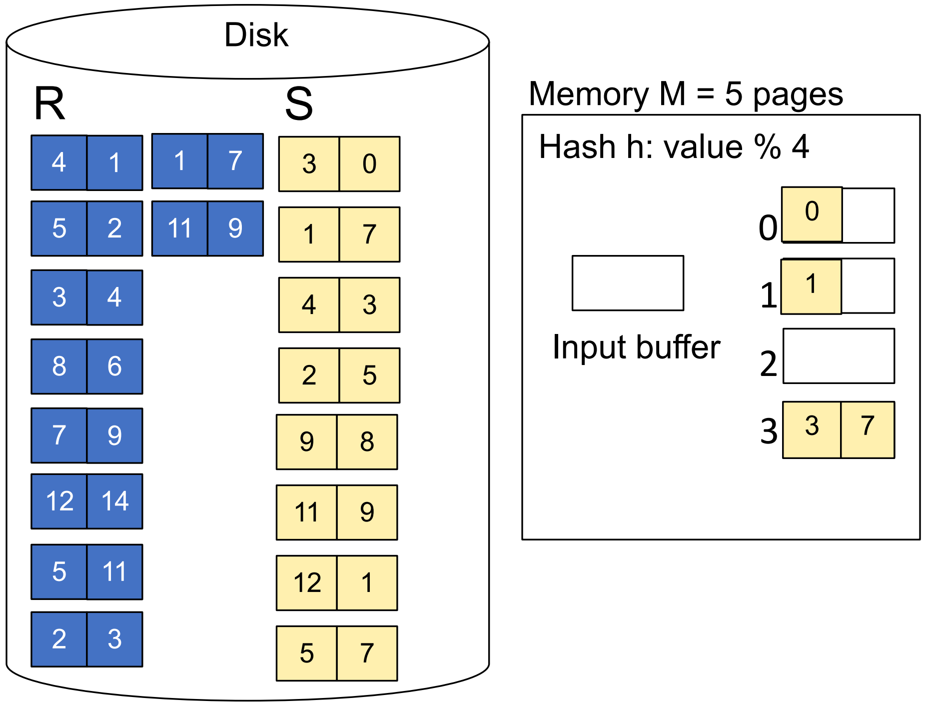
Example

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



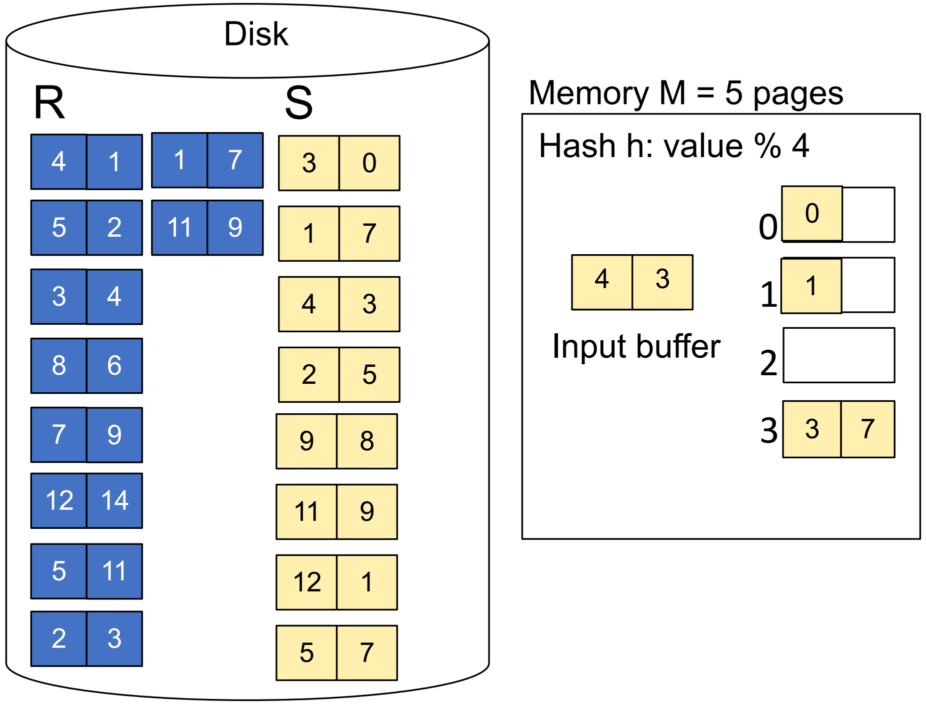
Example

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



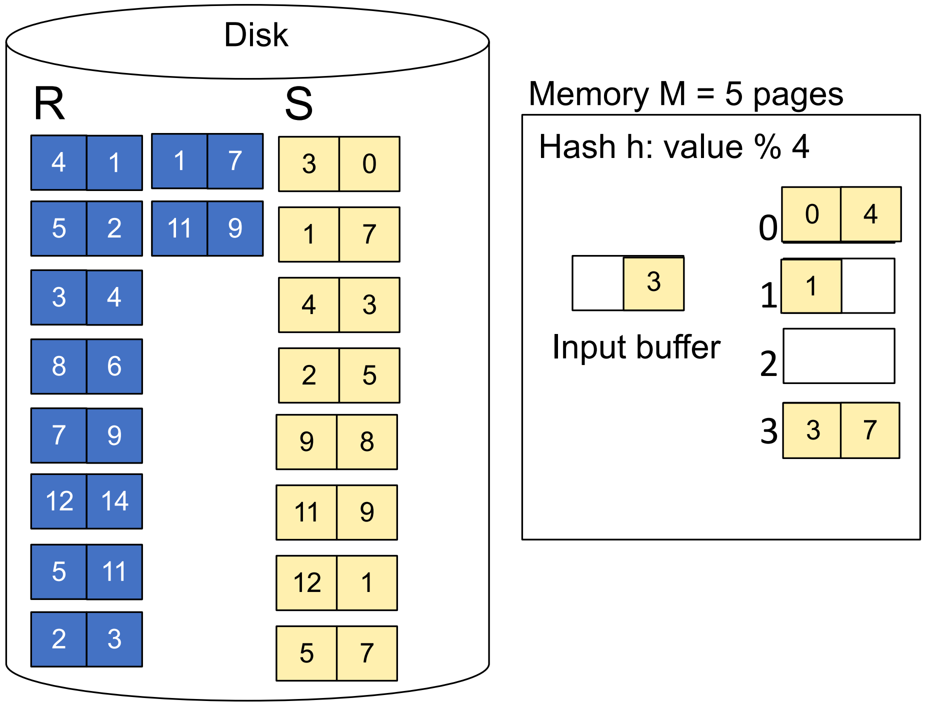
Example

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



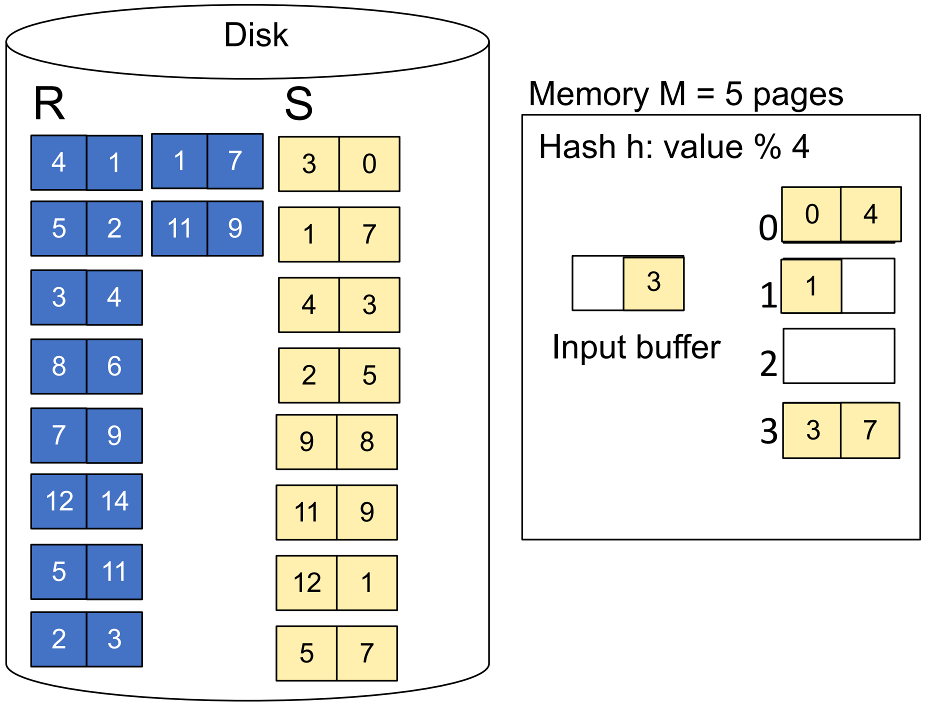
Example

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



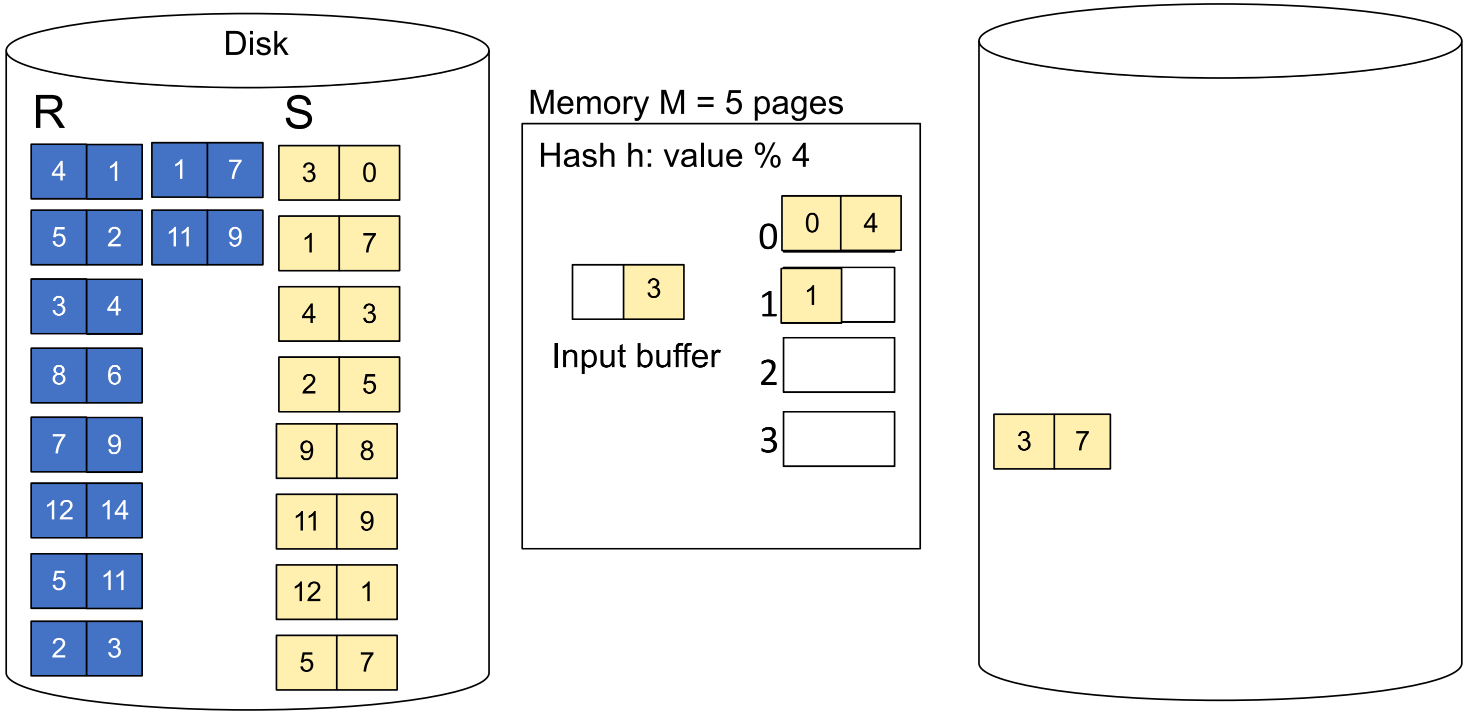
Example

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



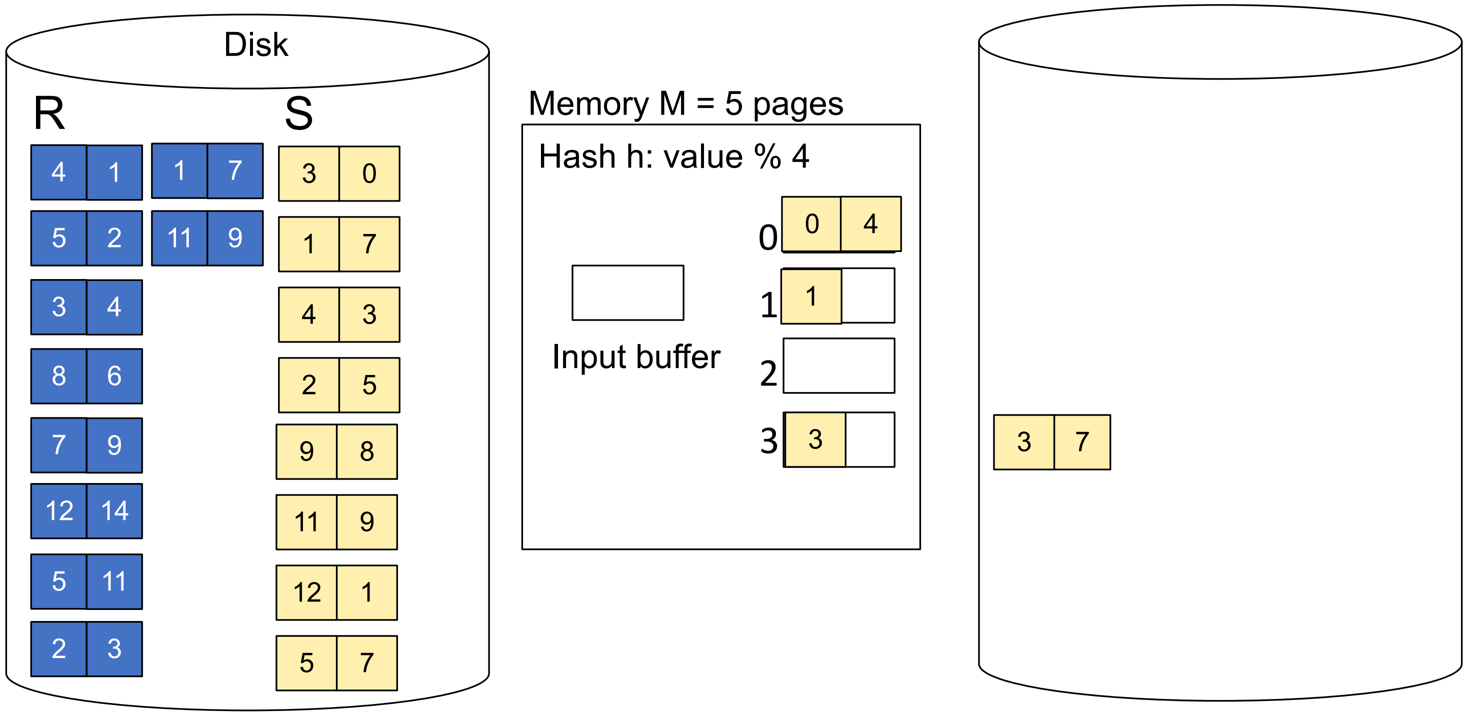
Example

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



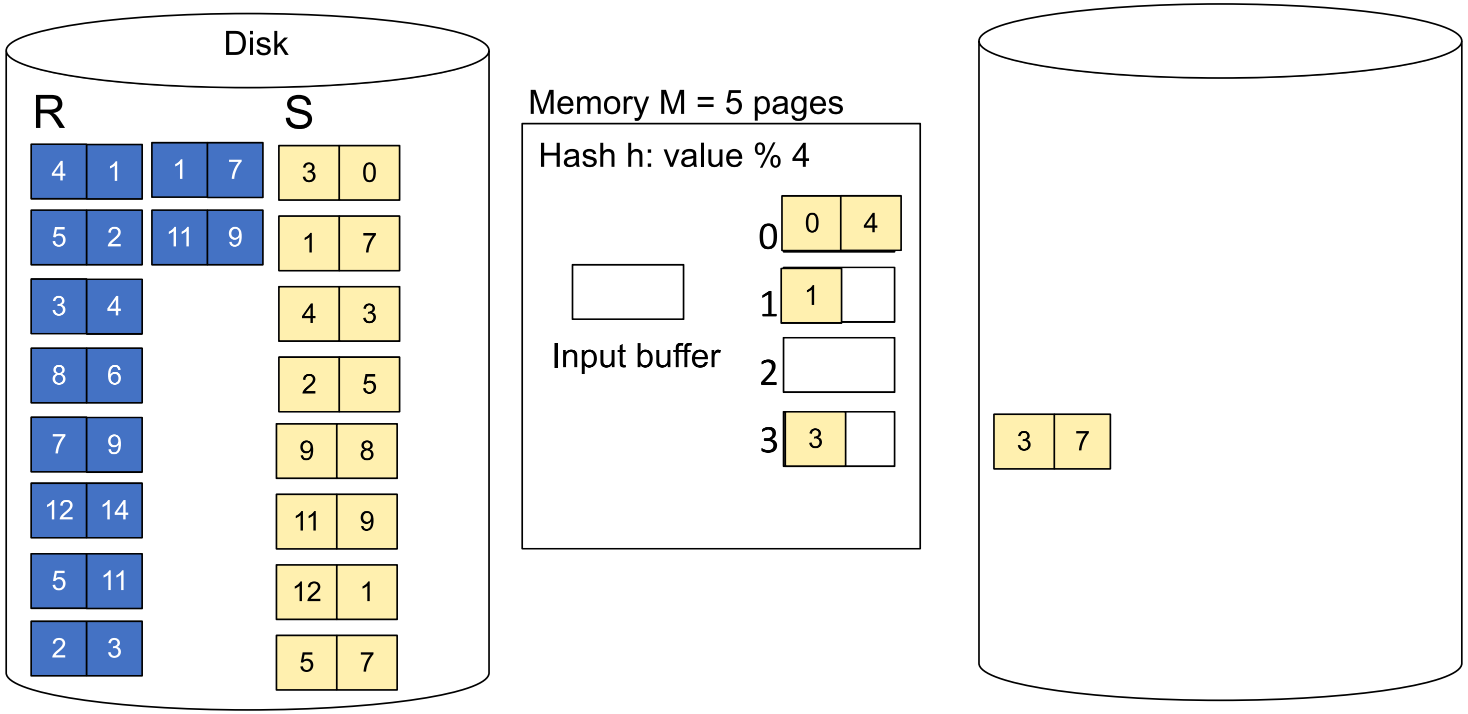
Example

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



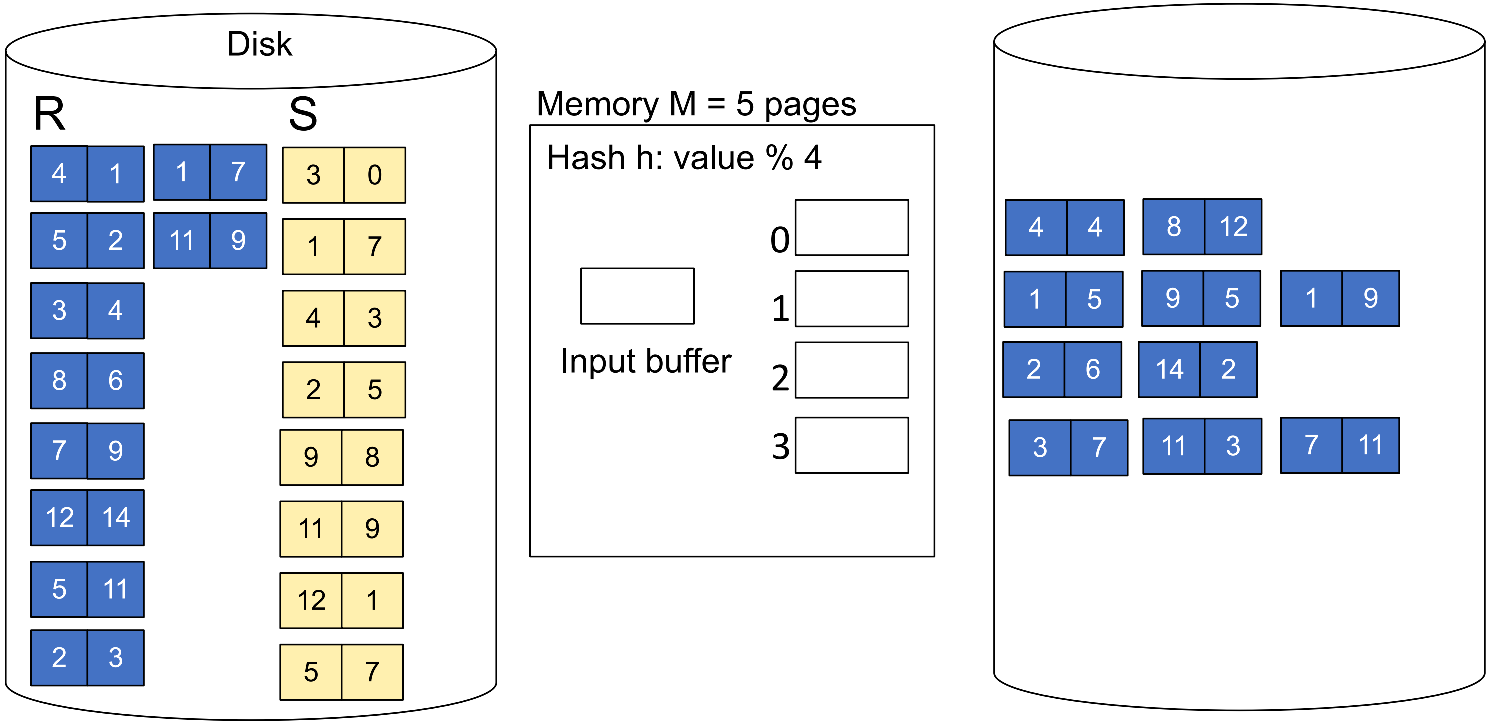
Example

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



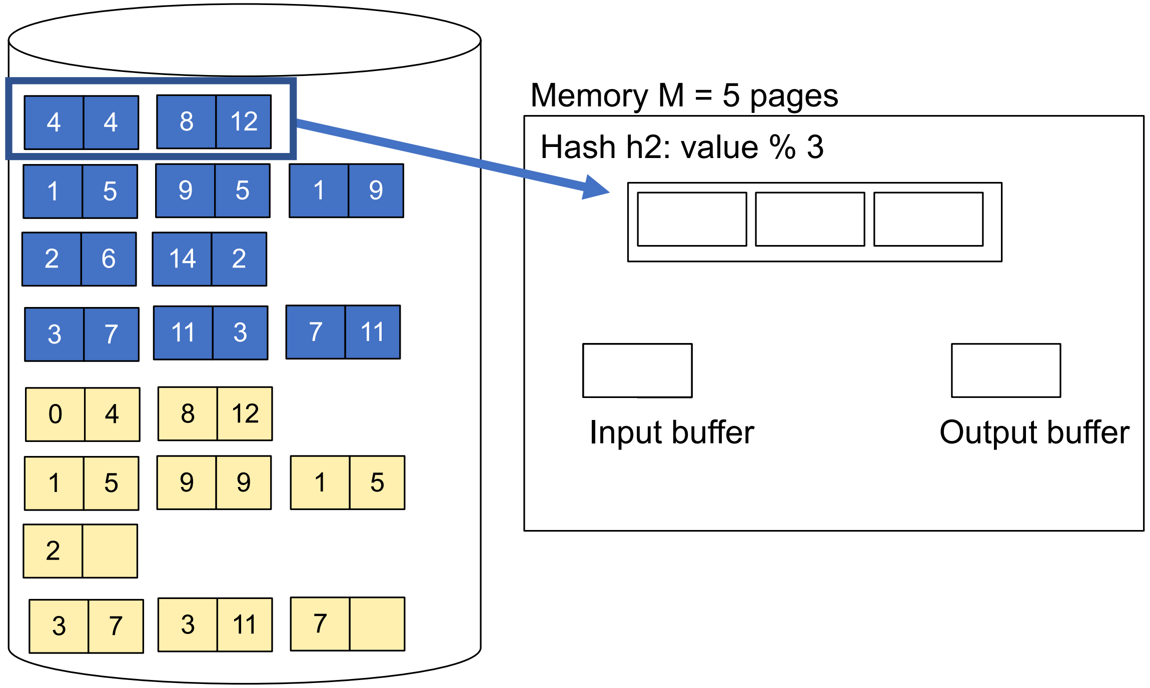
Example

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



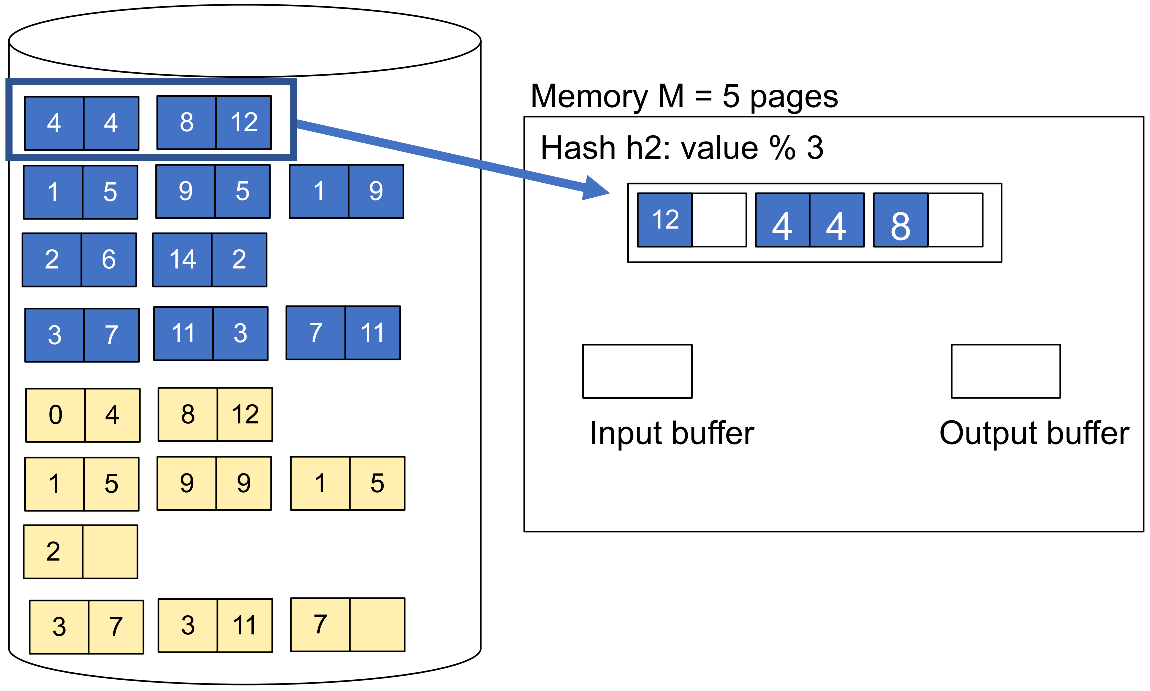
Example

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



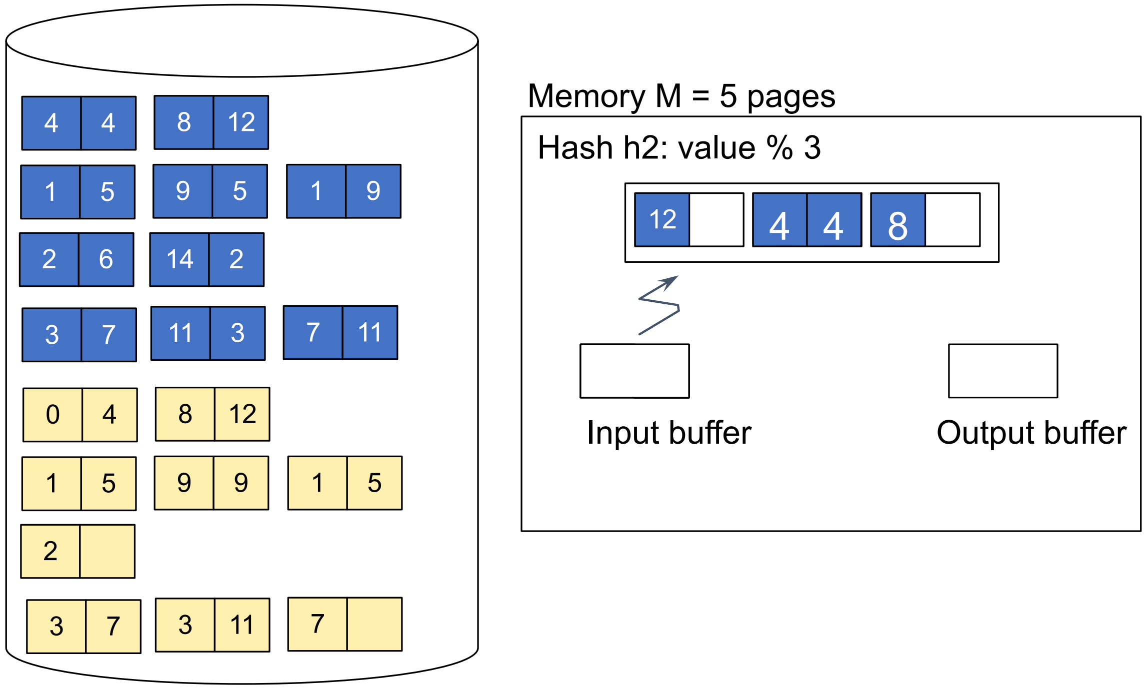
Example

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



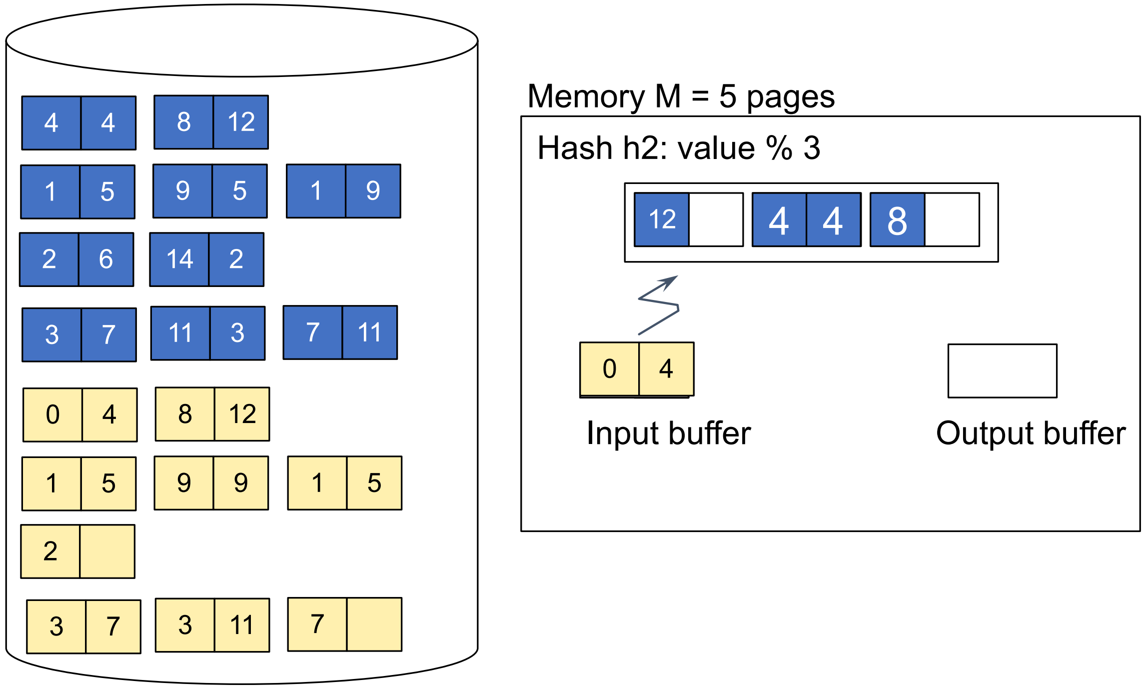
Example

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



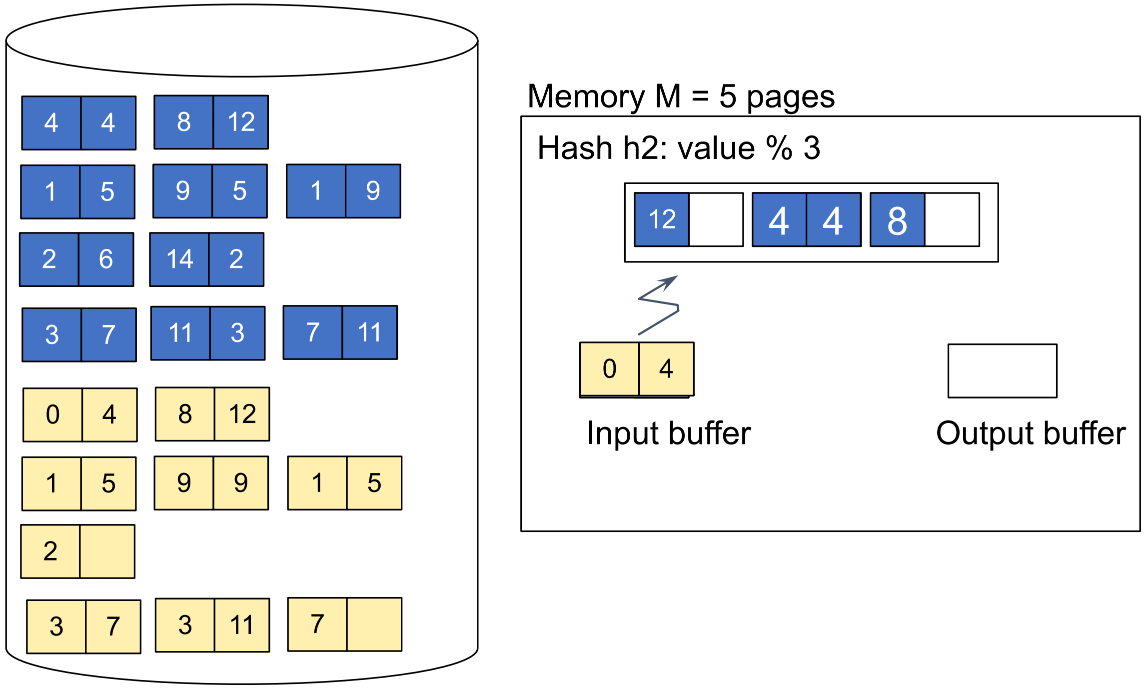
Example

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)



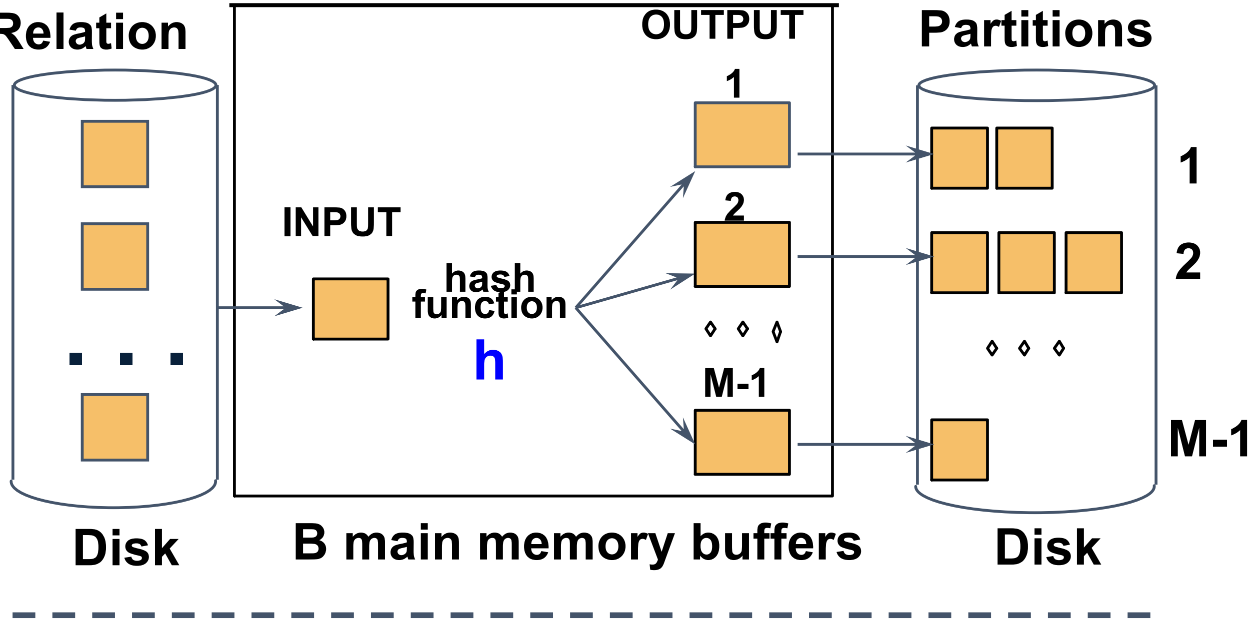
Example

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)

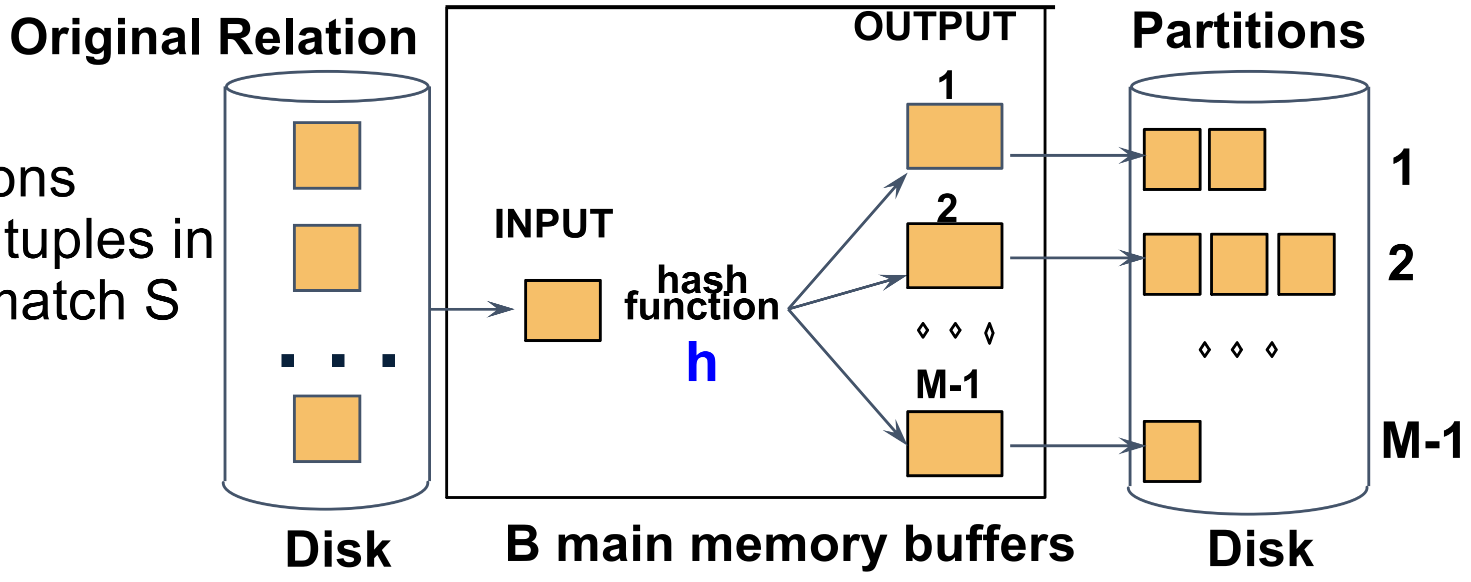


Partitioned Hash Join

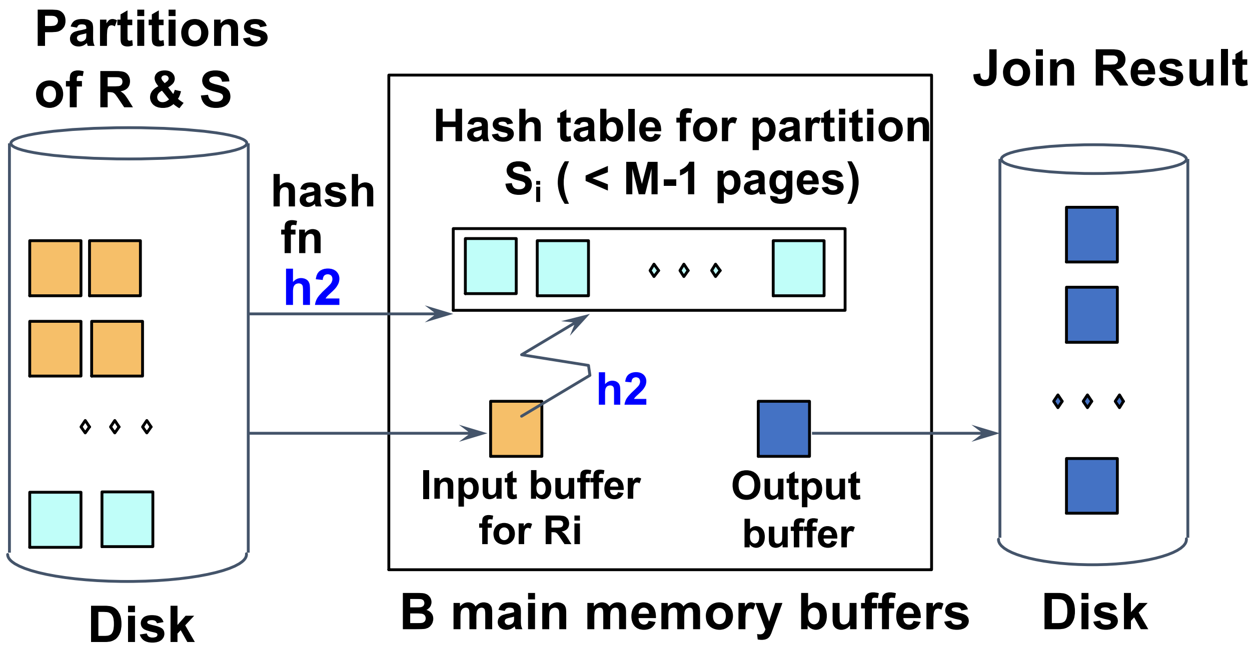
- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i.



- 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 (<> h!)**. Scan matching partition of S, search for matches.



Cost

- Cost: 3B(R) + 3B(S)
- Assumption: min(B(R), B(S)) <= M^2
- Minimum because 1-pass require the smaller operand to be less than M-1 and the larger one can always be streamed in

Summary of Join algorithms

- 1-pass
  - Block Nested Loop: B(S) + B(R)*B(S)/(M-1)
- 2-pass
  - Partitioned Hash: 3B(R)+3B(S);
    - min(B(R),B(S)) <= M^2
  - Merge Join: 3B(R)+3B(S)
    - B(R)+B(S) <= M^2

Hash Vs Sort

- Hash-based algorithms have a size requirement that depends on the smaller of the two arguments rather than the sum of two arguments.
- Sort-based algorithms produce result in sorted order---save some more if results to be piped to other operators.
- Hash-based algorithms depend on buckets being equal in size.

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)
- Note: we ignore I/O cost for index pages

Index-based selection; cost of $\sigma_{a=v}(R)$

- Example:
  - B(R) = 2000
  - T(R) = 100,000
  - V(R, a) = 20
- Table scan:
- Index based selection:

Index-based selection; cost of $\sigma_{a=v}(R)$

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

Index-based selection

- Example:
  - B(R) = 2000
  - T(R) = 100000
  - V(R, a) = 20
- Table scan: B(R) = 2000 I/Os
- Index-based selection:
  - If index is clustered: 2000/20 = 100
  - If index is unclustered: 100000/20 = 5000

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

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:
- 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)

Summary of Join algorithms

- Block Nested Loop: B(S) + B(R)*B(S)/(M-1)
- Partitioned Hash: 3B(R)+3B(S);
  - min(B(R),B(S)) <= M^2
- Merge Join: 3B(R)+3B(S)
  - B(R)+B(S) <= M^2
- Index Join: B(R) + T(R)B(S)/V(S,a)
  - (unclustered)

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