



THE UNIVERSITY
of EDINBURGH

Advanced Database Systems

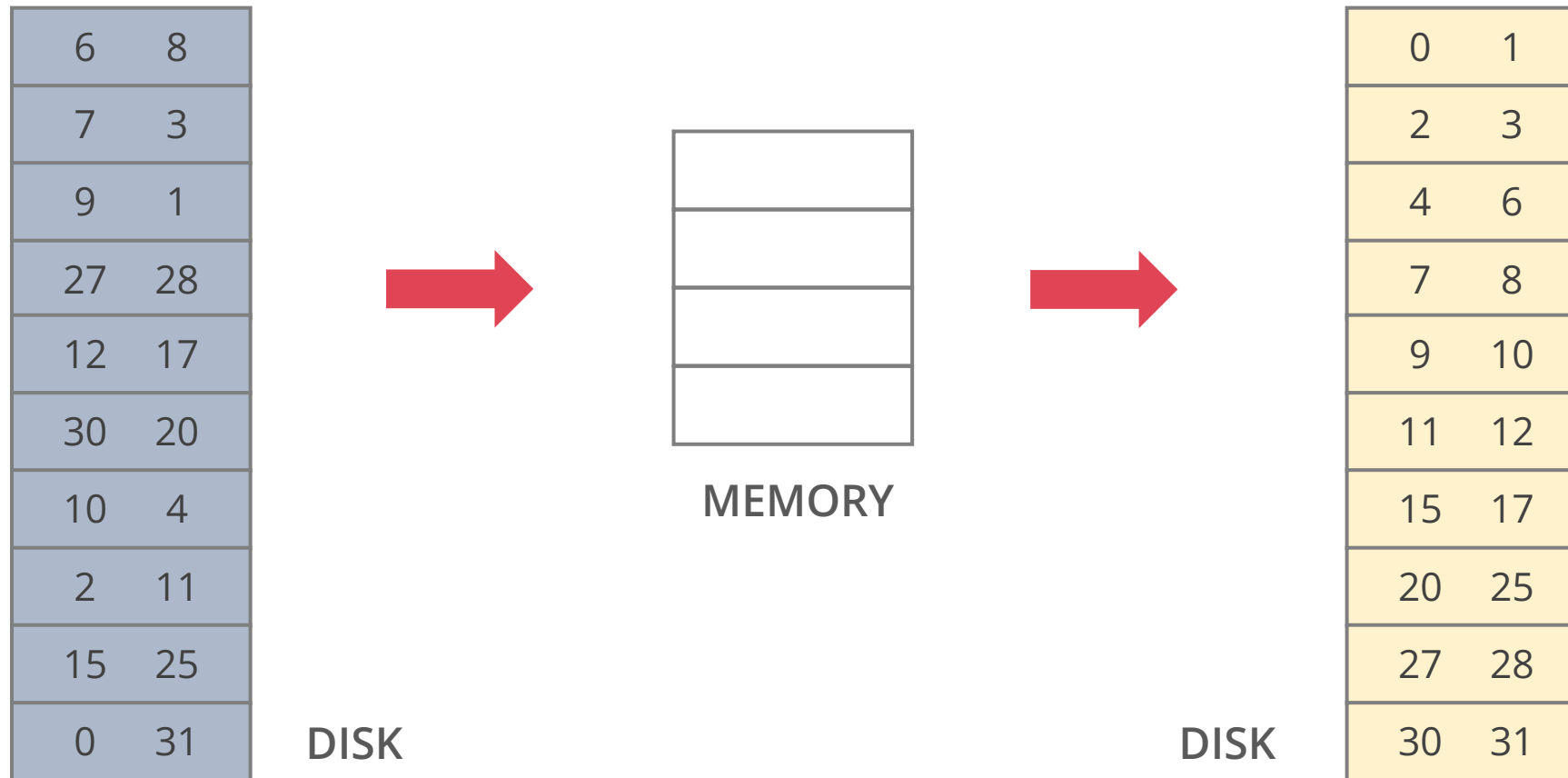
Spring 2024

Tutorial 3

EXTERNAL SORTING

GENERAL EXTERNAL MERGE SORT

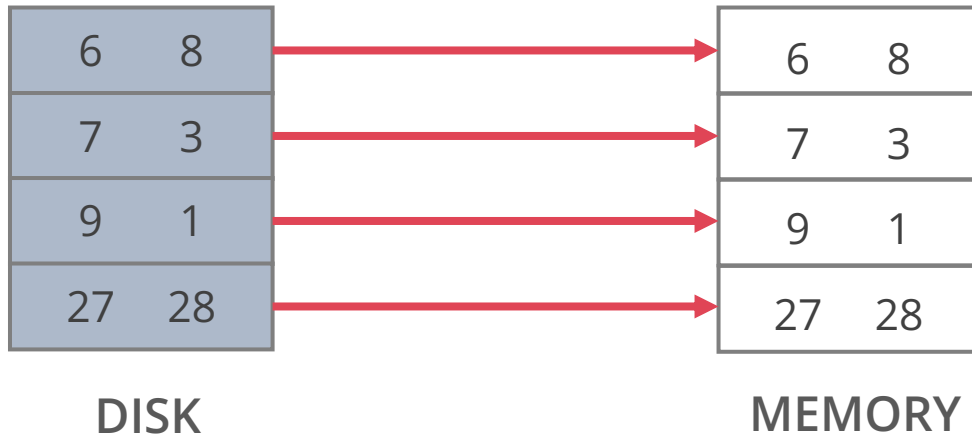
Goal: sort records on 10 data pages using 4 buffer pages



GENERAL EXTERNAL MERGE SORT

Pass 0, Run 1

Read 4 pages into memory



I/O Total (so far): 4

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 1

In-memory sort

6	8
7	3
9	1
27	28

DISK

1	3
6	7
8	9
27	28

MEMORY

I/O Total (so far): 4

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 1

Write 4 sorted pages to disk

6	8
7	3
9	1
27	28

DISK

1	3
6	7
8	9
27	28

MEMORY



1 sorted run of 4 pages

1	3	6	7	8	9	27	28
---	---	---	---	---	---	----	----

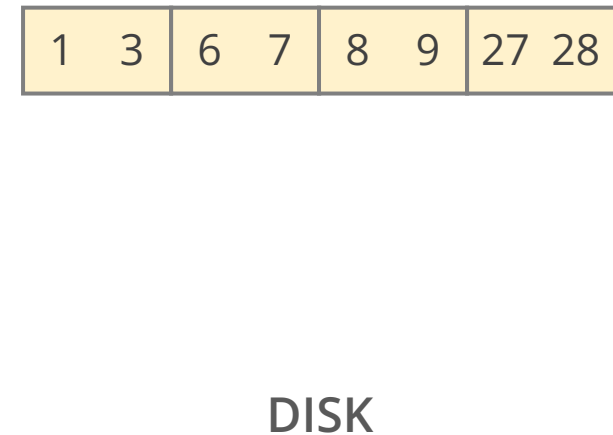
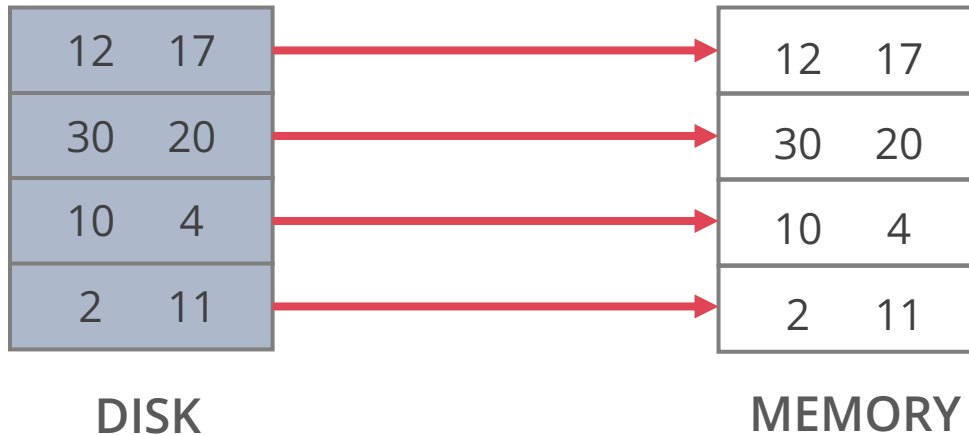
DISK

I/O Total (so far): 8

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 2

Read 4 pages into memory



I/O Total (so far): 12

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 2

In-memory sort

12	17
30	20
10	4
2	11

DISK

2	4
10	11
12	17
20	30

MEMORY

1	3	6	7	8	9	27	28
---	---	---	---	---	---	----	----

DISK

I/O Total (so far): 12

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 2

Write 4 sorted pages to disk

12	17
30	20
10	4
2	11

DISK

2	4
10	11
12	17
20	30

MEMORY



1	3	6	7	8	9	27	28
2	4	10	11	12	17	20	30

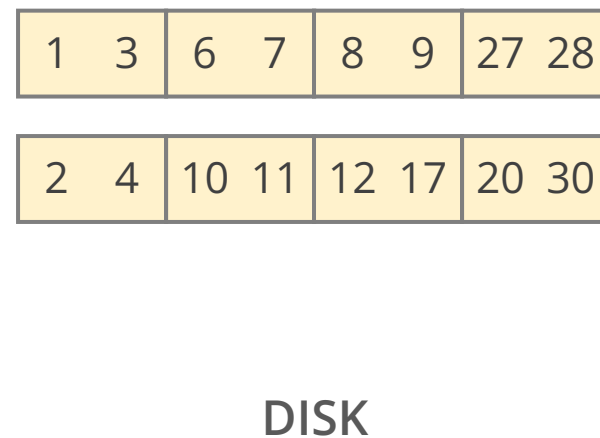
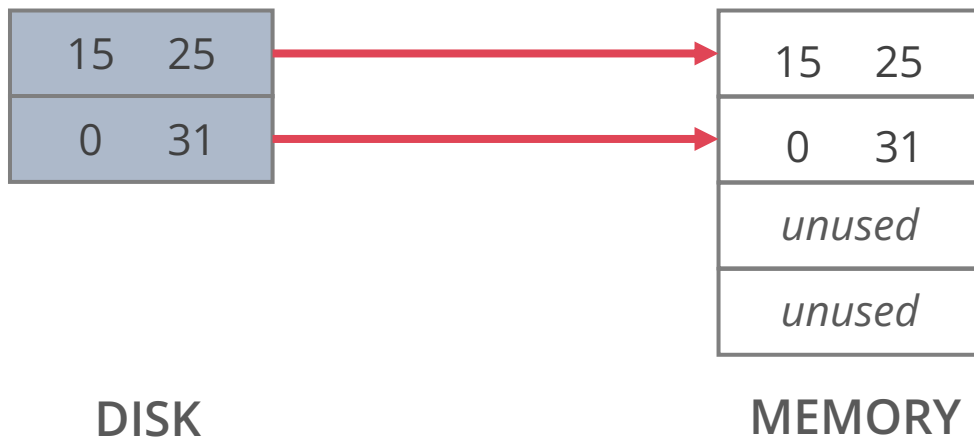
DISK

I/O Total (so far): 16

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 3

Read 2 pages into memory



I/O Total (so far): **18**

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 3

In-memory sort

15	25
0	31

DISK

0	15
25	31
<i>unused</i>	
<i>unused</i>	

MEMORY

1	3	6	7	8	9	27	28
2	4	10	11	12	17	20	30

DISK

I/O Total (so far): **18**

GENERAL EXTERNAL MERGE SORT

Pass 0, Run 3

Write 2 sorted pages to disk

15	25
0	31

DISK

0	15
25	31
<i>unused</i>	
<i>unused</i>	

MEMORY



1	3	6	7	8	9	27	28
2	4	10	11	12	17	20	30
0	15	25	31				

DISK

I/O Total (so far): 20

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	1	3	6	7	8	9	27	28
Run 2	2	4	10	11	12	17	20	30
Run 3	0	15	25	31				

DISK

<i>input</i>
<i>input</i>
<i>input</i>
<i>output</i>

MEMORY

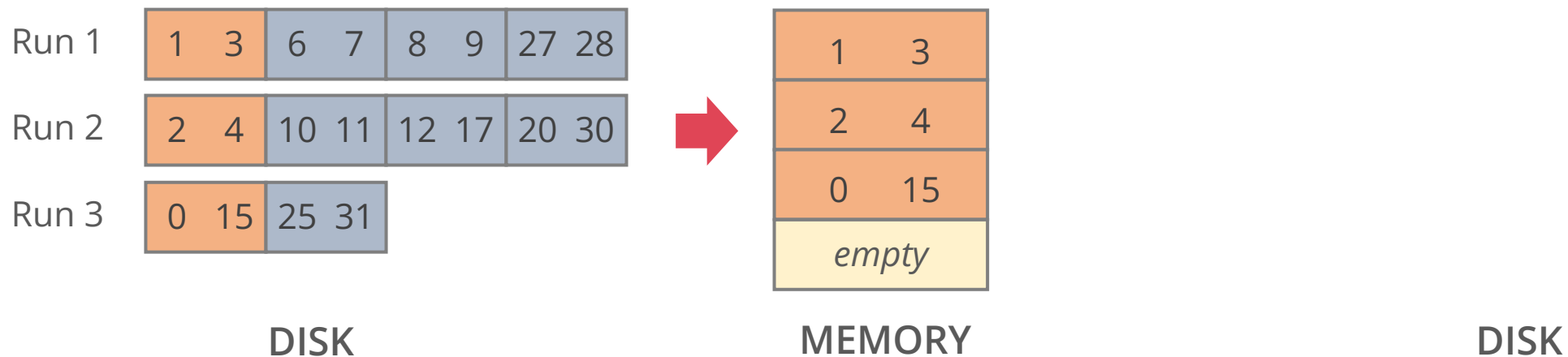
DISK

Reserve ***B-1*** input buffers and ***1*** output buffer. Load one page from each run at a time. Store sorted results in output buffer. Write to disk when output buffer is full

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



I/O Total (so far): 23

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6	7	8	9	27	28
Run 2	10	11	12	17	20	30
Run 3	25	31				

DISK

1	3
2	4
0	15
empty	

MEMORY

Output buffer

DISK

I/O Total (so far): 23

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6	7	8	9	27	28
Run 2	10	11	12	17	20	30
Run 3	25	31				

DISK

1	3
2	4
15	
0	

MEMORY

DISK

I/O Total (so far): 23

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6 7	8 9	27 28
Run 2	10 11	12 17	20 30
Run 3	25 31		

DISK

1	3
2	4
15	
0	

MEMORY

DISK

I/O Total (so far): 23

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6	7	8	9	27	28
Run 2	10	11	12	17	20	30
Run 3	25	31				

DISK

	3
2	4
	15
0	1

MEMORY

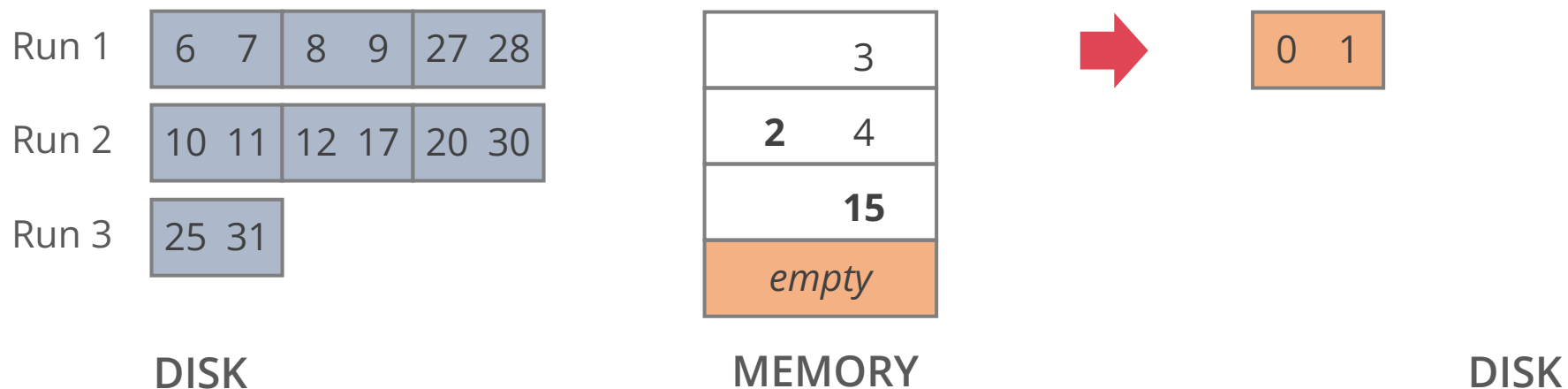
DISK

I/O Total (so far): 23

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



I/O Total (so far): **24**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6 7	8 9	27 28
Run 2	10 11	12 17	20 30
Run 3	25 31		

DISK

3
2 4
15
<i>empty</i>

MEMORY

0 1

DISK

I/O Total (so far): 24

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6 7	8 9	27 28
Run 2	10 11	12 17	20 30
Run 3	25 31		

DISK

3
4
15
2

MEMORY

0 1

DISK

I/O Total (so far): 24

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6 7	8 9	27 28
Run 2	10 11	12 17	20 30
Run 3	25 31		

DISK

3
4
15
2

MEMORY

0 1

DISK

I/O Total (so far): 24

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	6 7	8 9	27 28
Run 2	10 11	12 17	20 30
Run 3	25 31		

DISK

<i>empty</i>
4
15
2 3

MEMORY

0 1

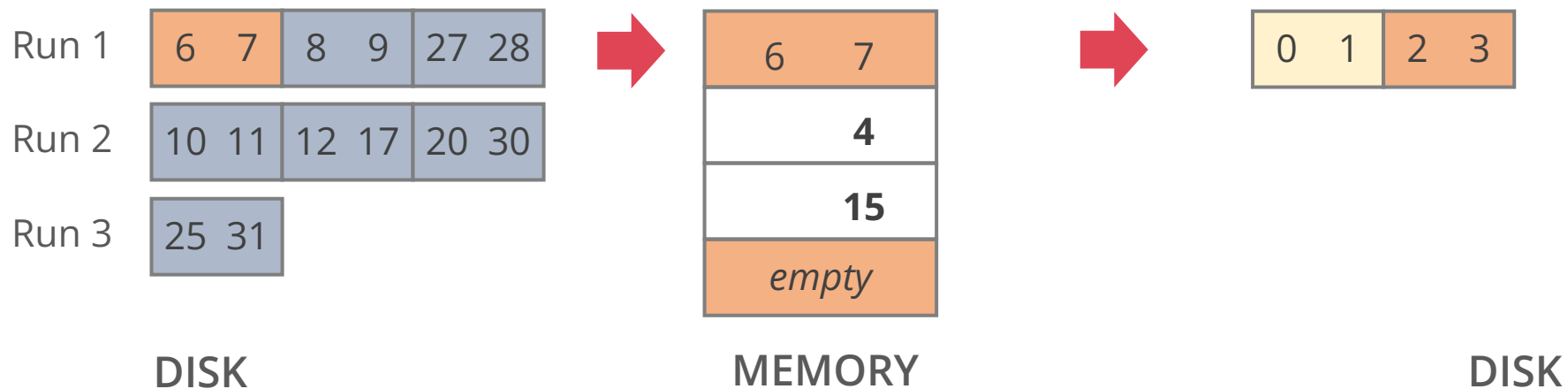
DISK

I/O Total (so far): **24**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



I/O Total (so far): **26**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1	8 9		27 28	
Run 2	10 11	12 17	20 30	
Run 3	25 31			

DISK

6	7
	4
	15
	empty

MEMORY

0	1	2	3
---	---	---	---

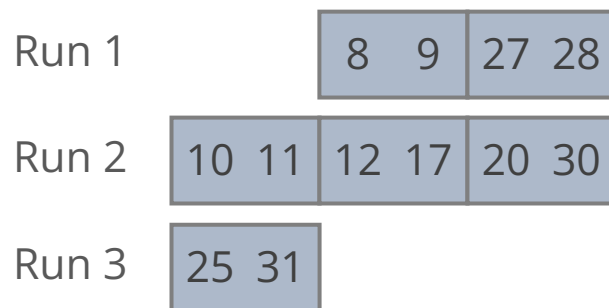
DISK

I/O Total (so far): 26

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



DISK



MEMORY



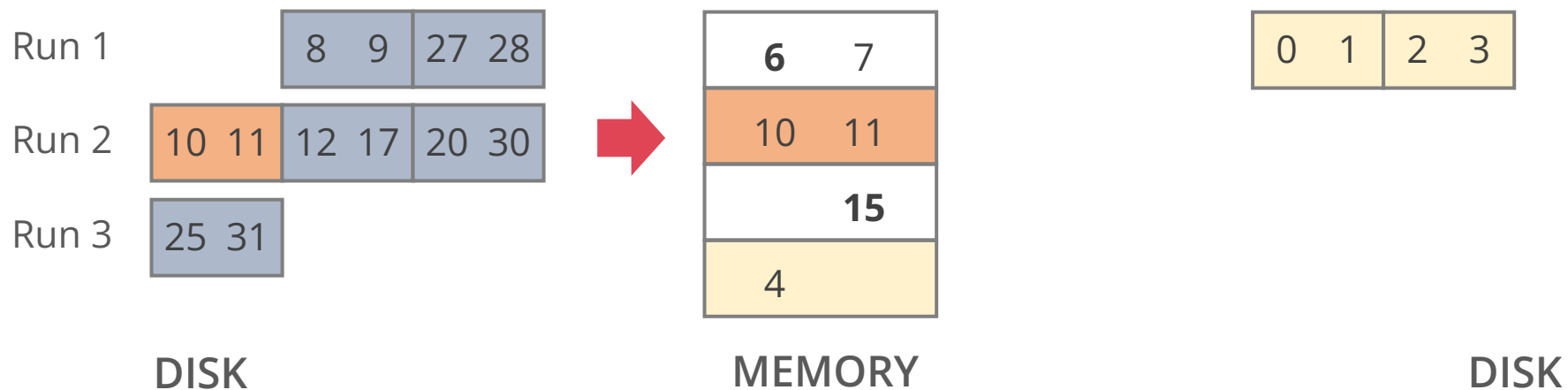
DISK

I/O Total (so far): 26

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

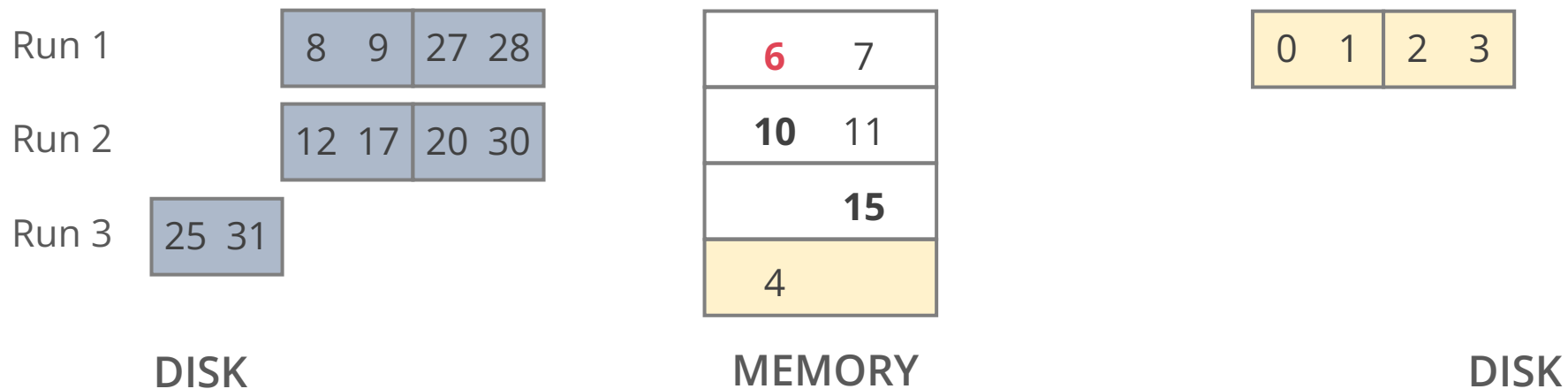


I/O Total (so far): **27**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

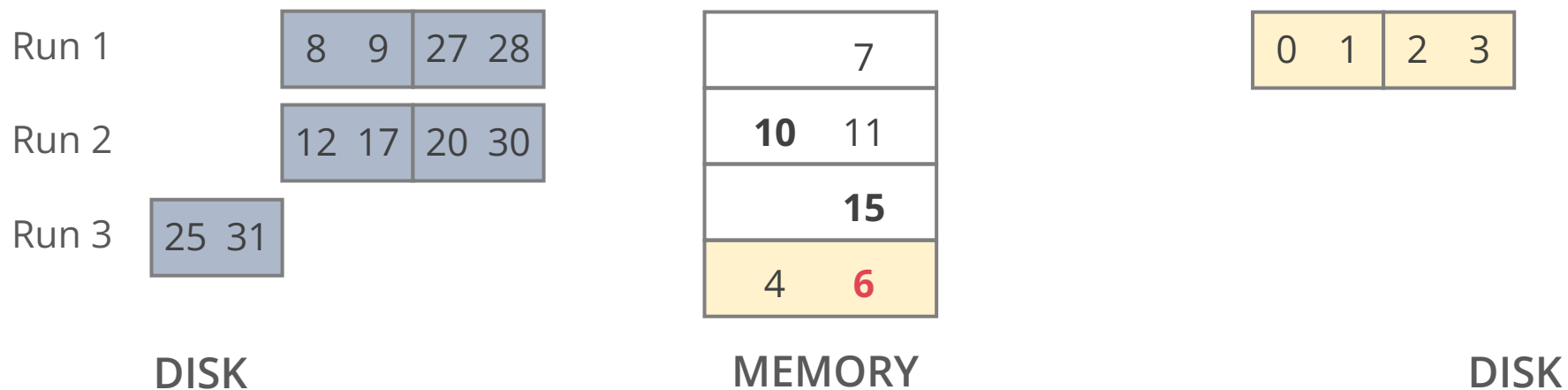


I/O Total (so far): **27**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

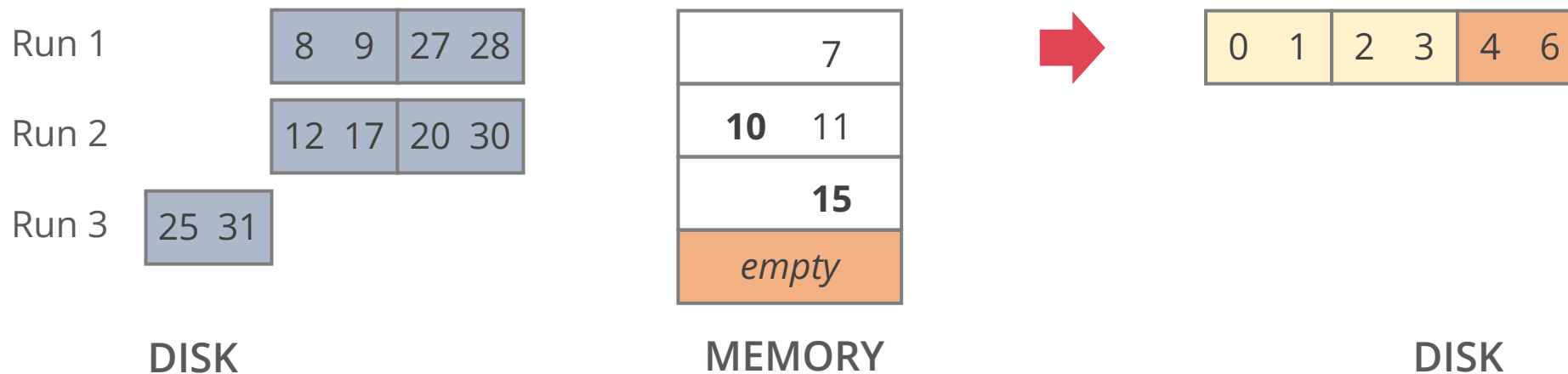


I/O Total (so far): **27**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

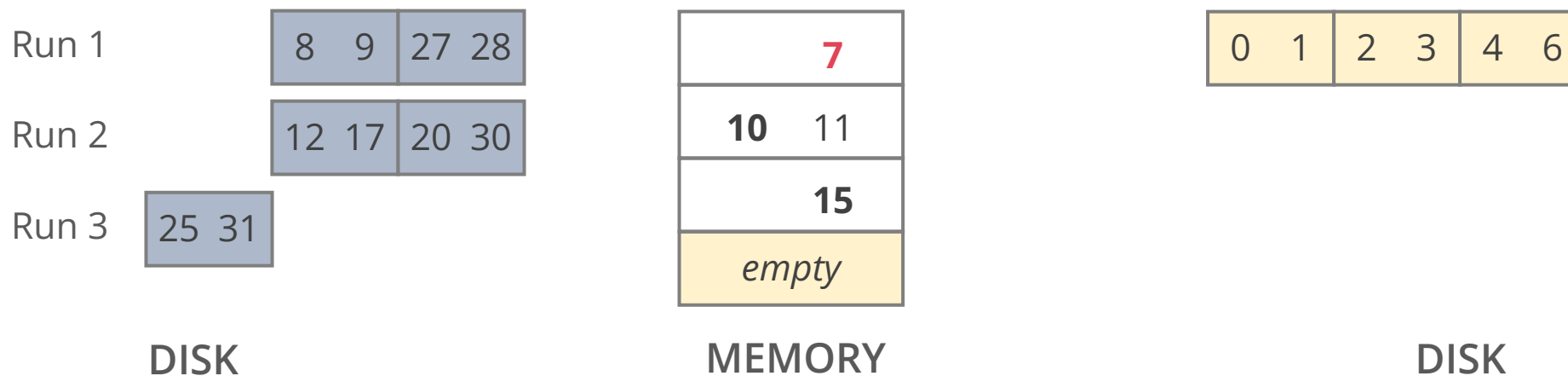


I/O Total (so far): **28**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

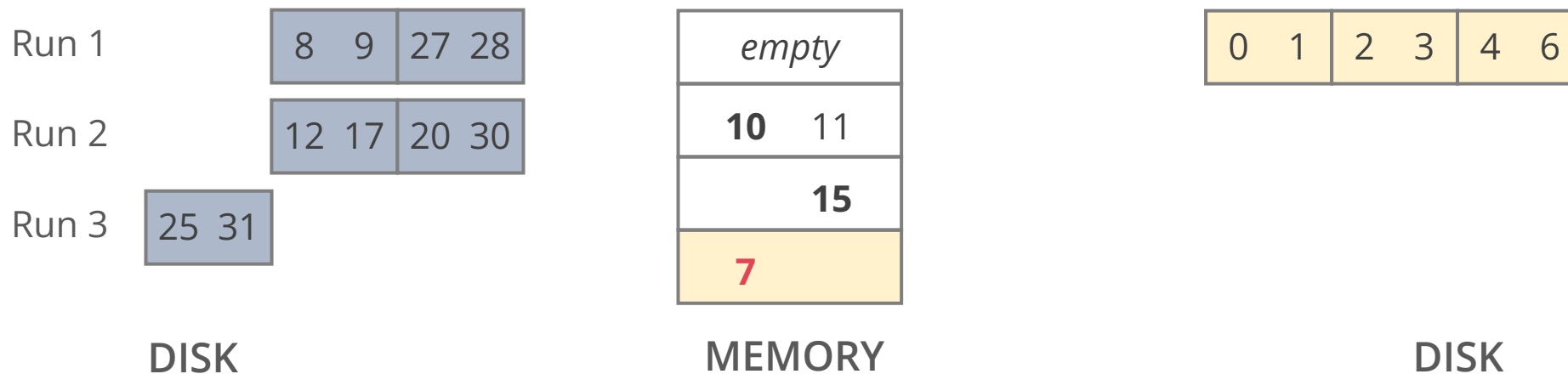


I/O Total (so far): 28

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

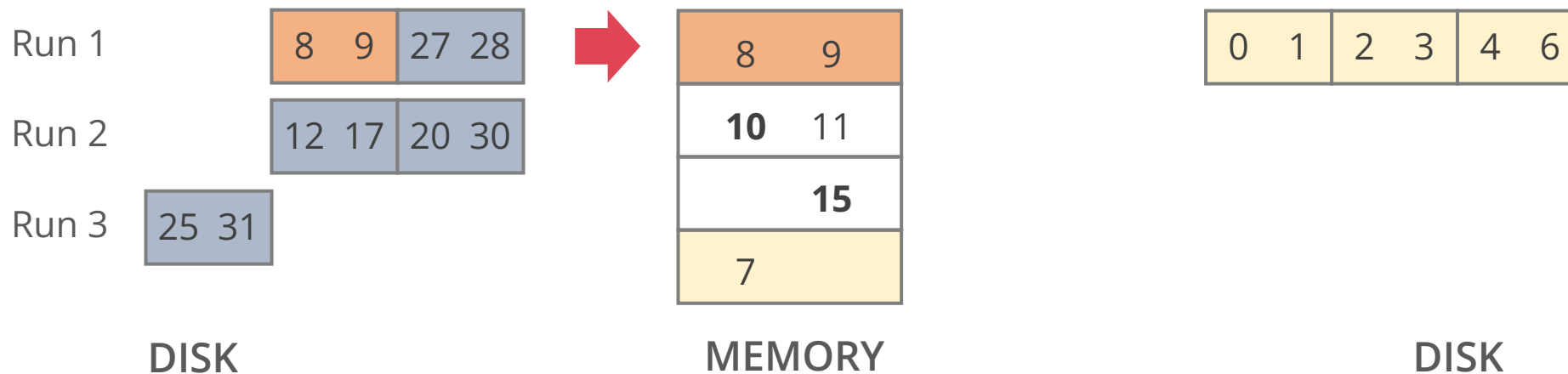


I/O Total (so far): 28

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

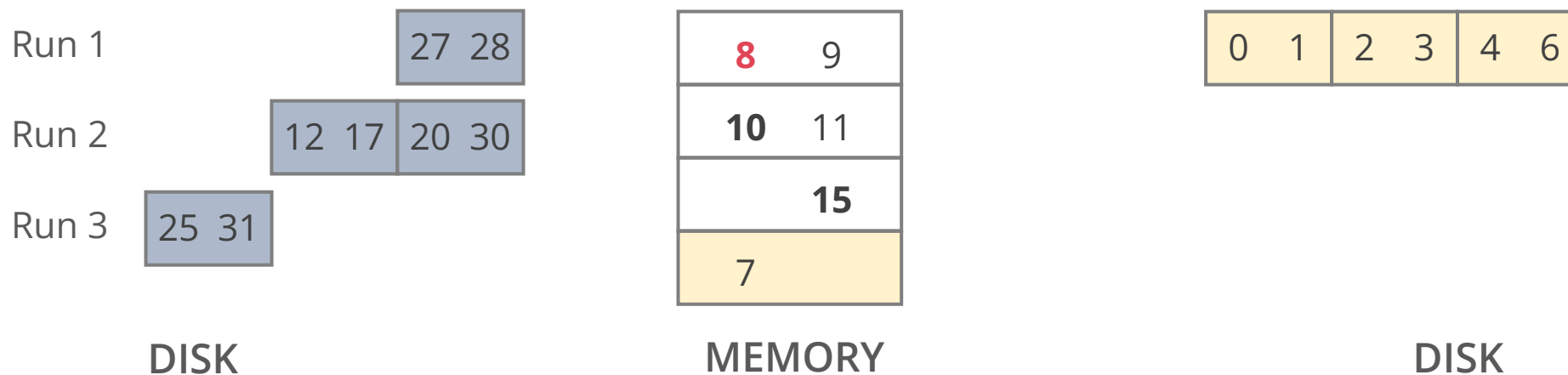


I/O Total (so far): 29

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

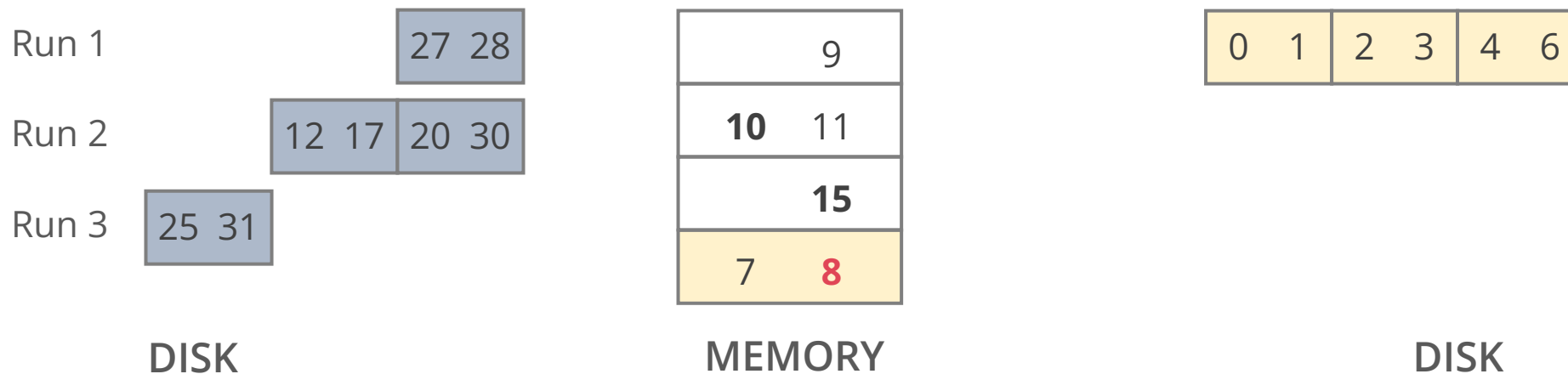


I/O Total (so far): 29

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

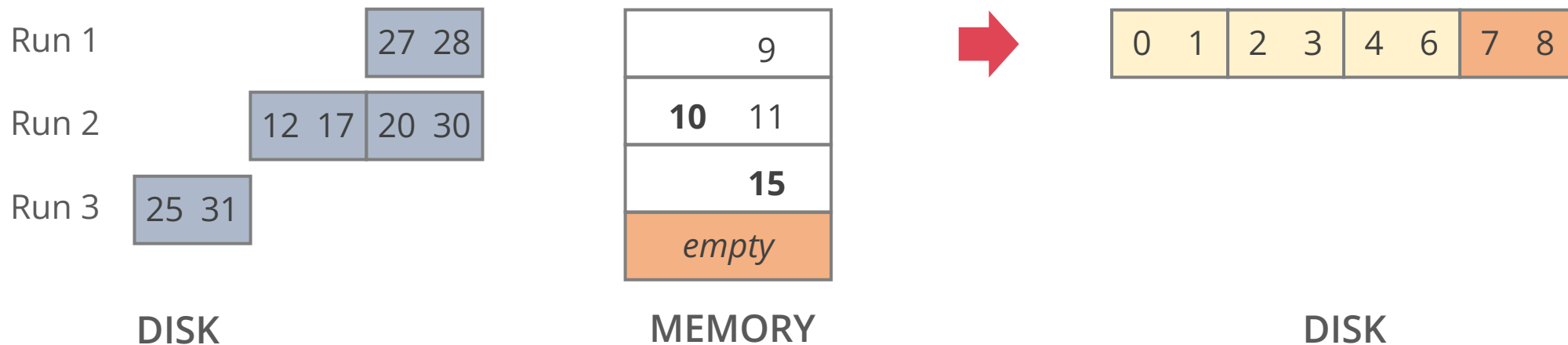


I/O Total (so far): 29

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

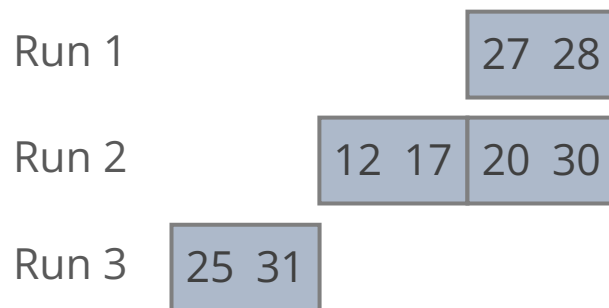


I/O Total (so far): **30**

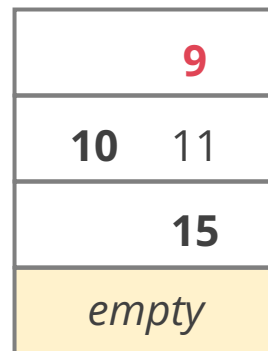
GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



DISK



MEMORY



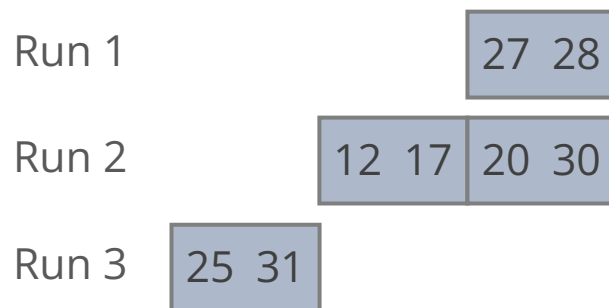
DISK

I/O Total (so far): 30

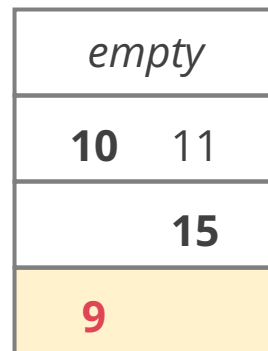
GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



DISK



MEMORY



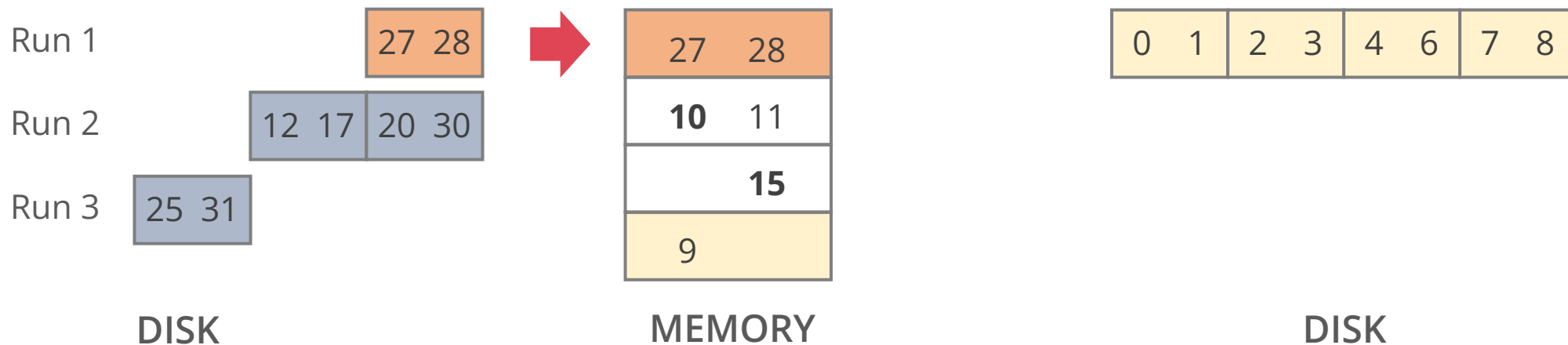
DISK

I/O Total (so far): **30**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

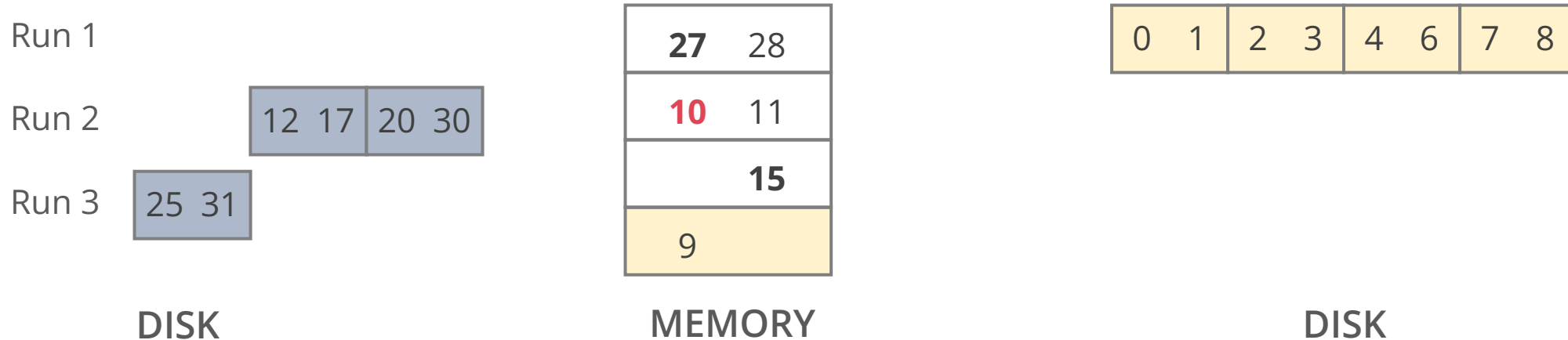


I/O Total (so far): **31**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

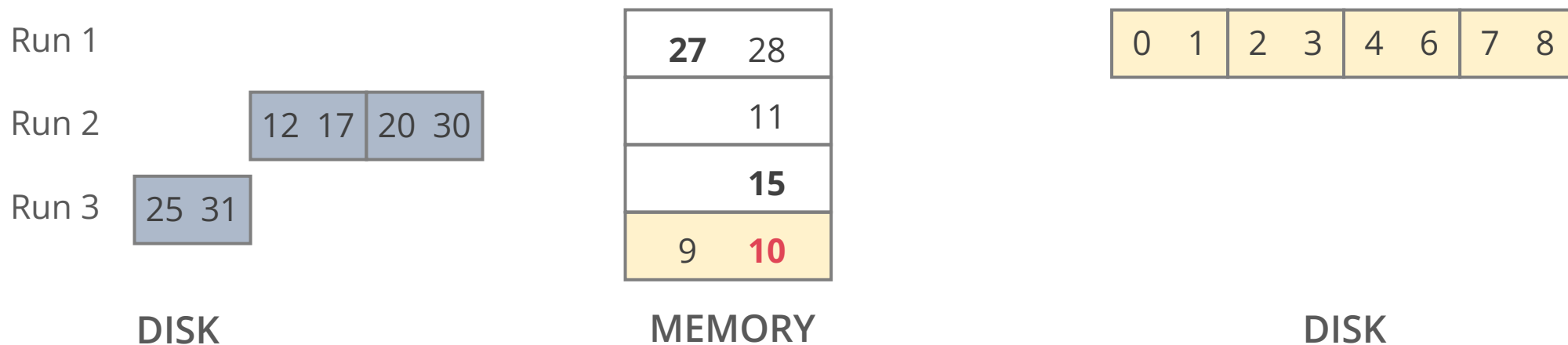


I/O Total (so far): 31

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

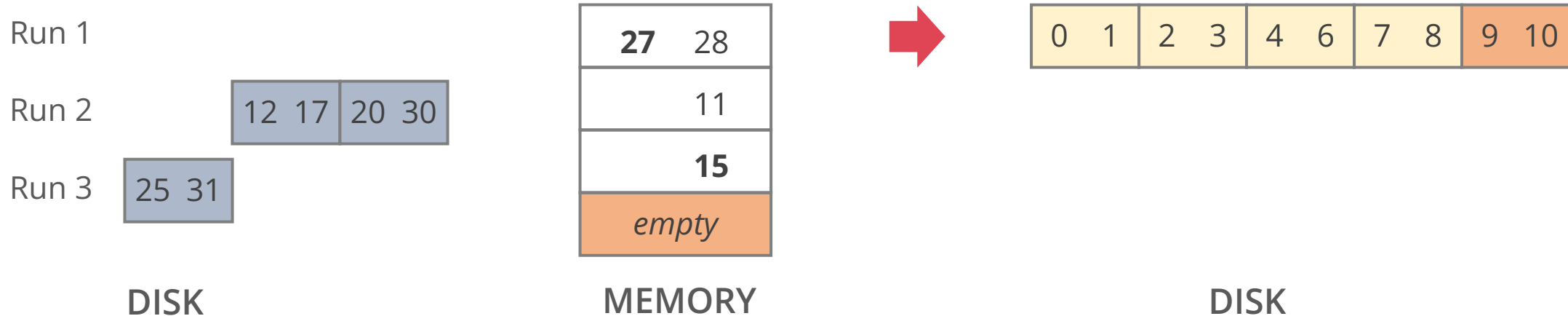


I/O Total (so far): **31**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

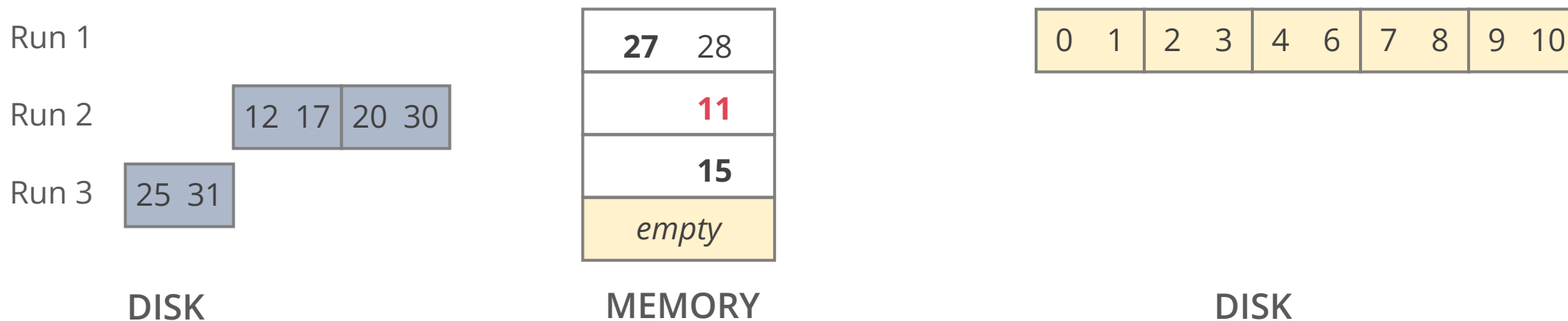


I/O Total (so far): 32

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

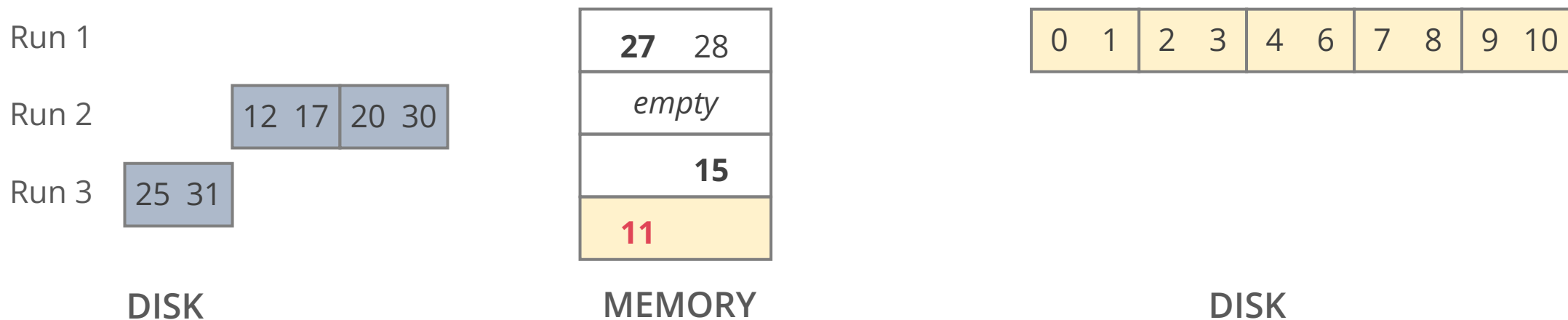


I/O Total (so far): 32

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

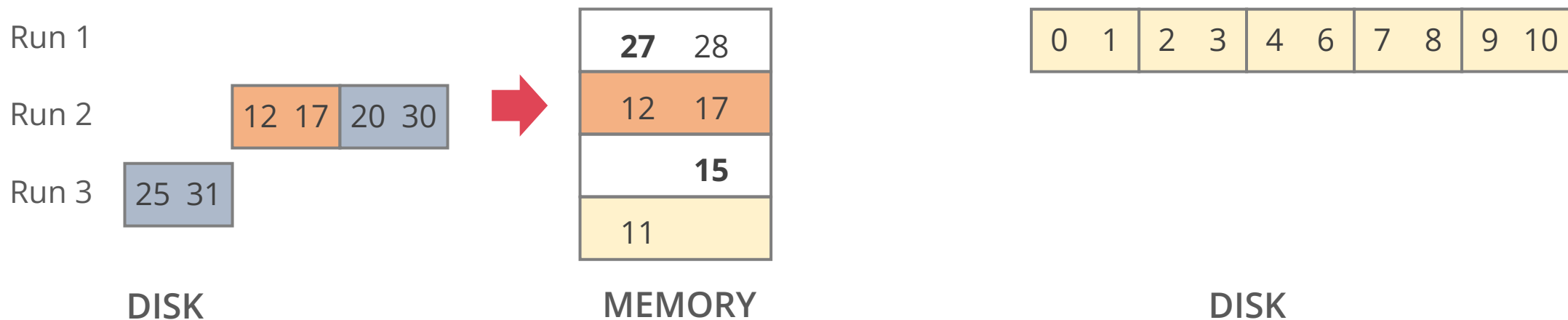


I/O Total (so far): 32

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

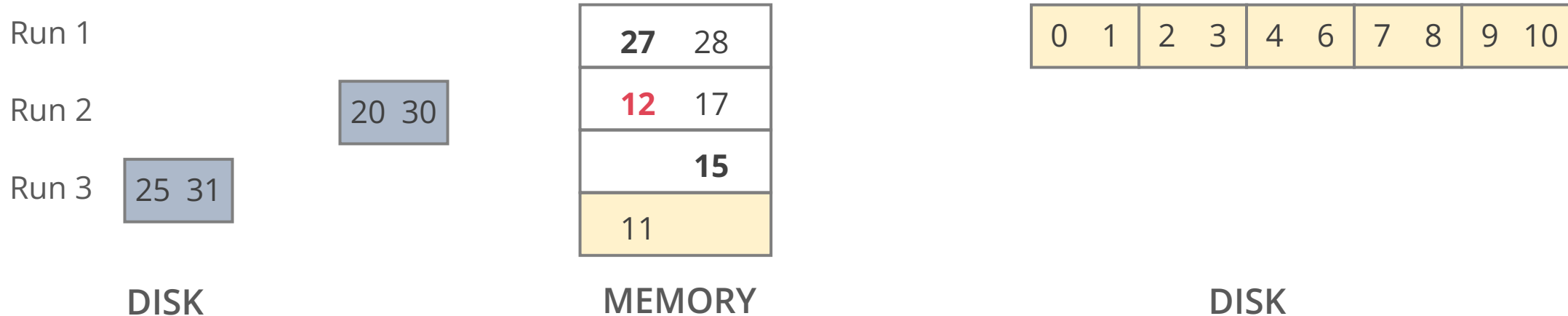


I/O Total (so far): **33**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

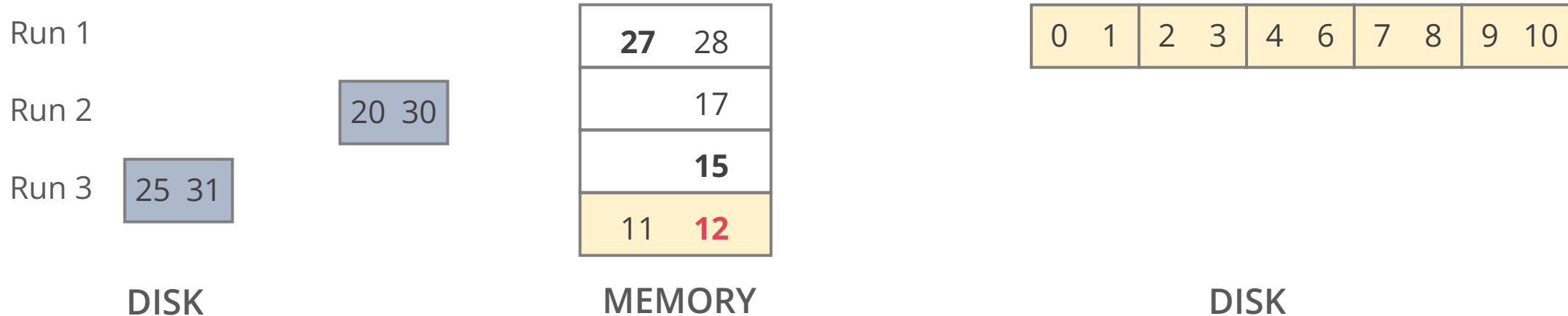


I/O Total (so far): 33

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

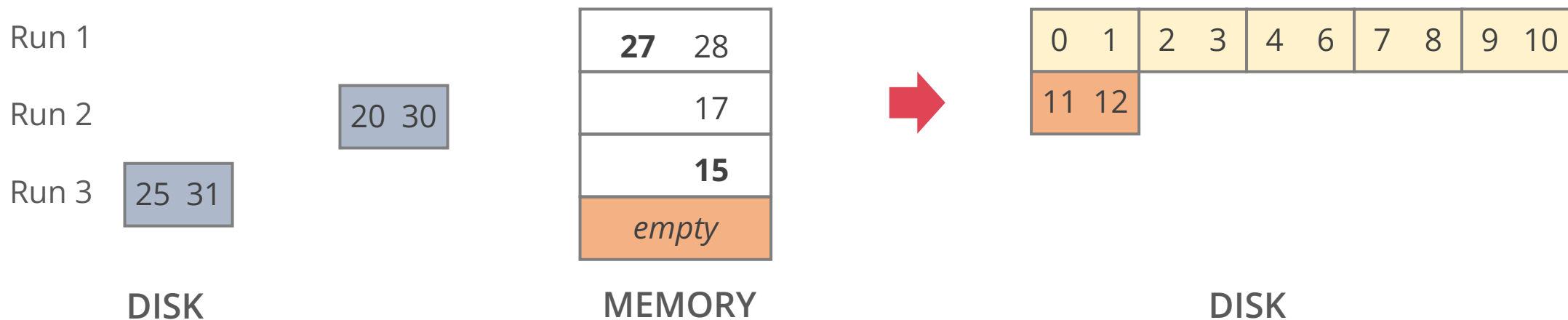


I/O Total (so far): **33**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

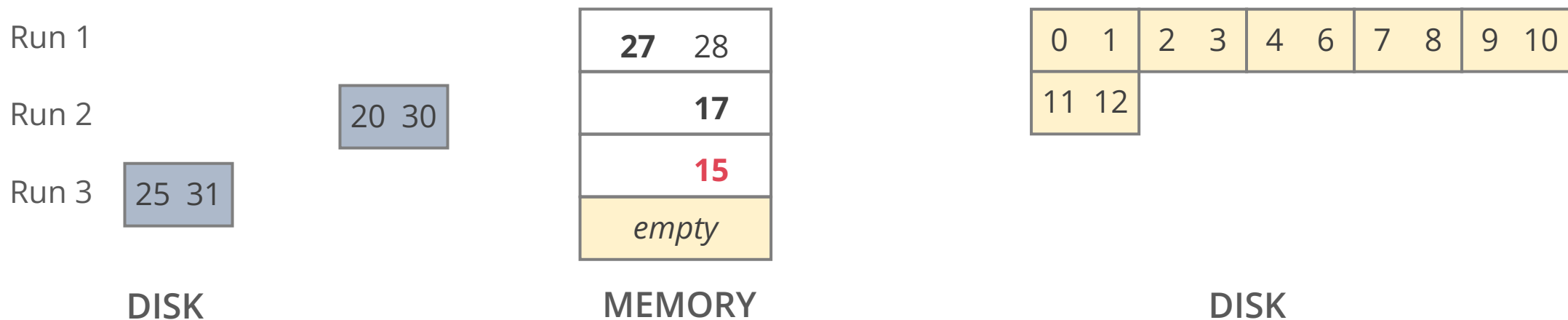


I/O Total (so far): 34

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

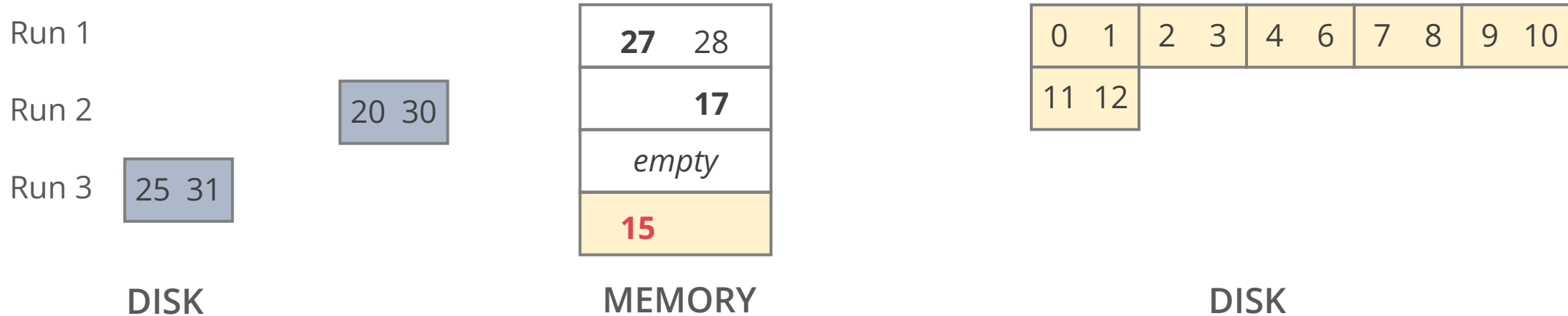


I/O Total (so far): 34

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

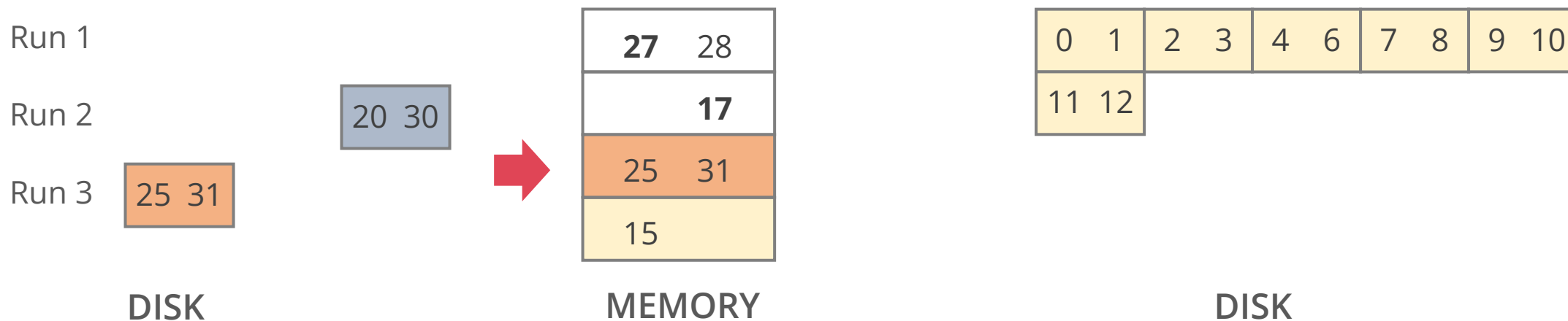


I/O Total (so far): 34

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

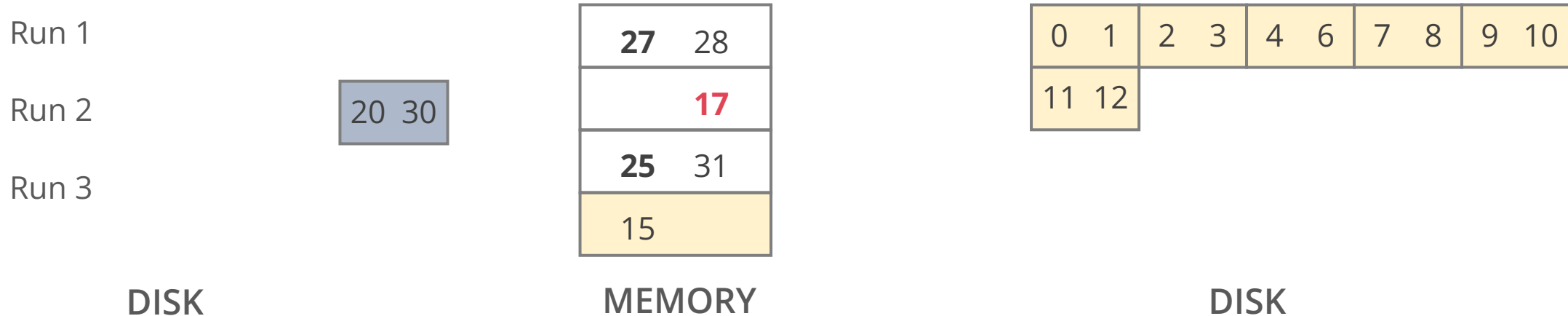


I/O Total (so far): **35**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

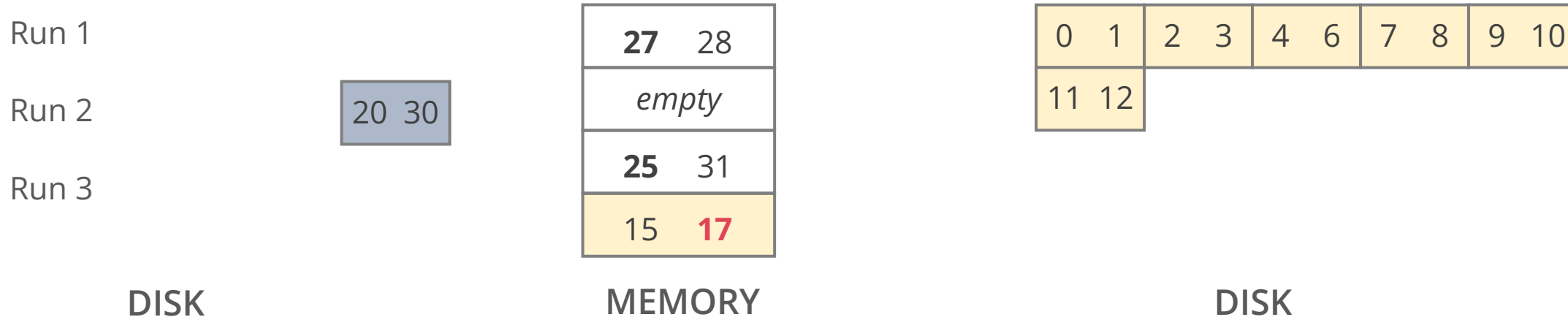


I/O Total (so far): 35

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

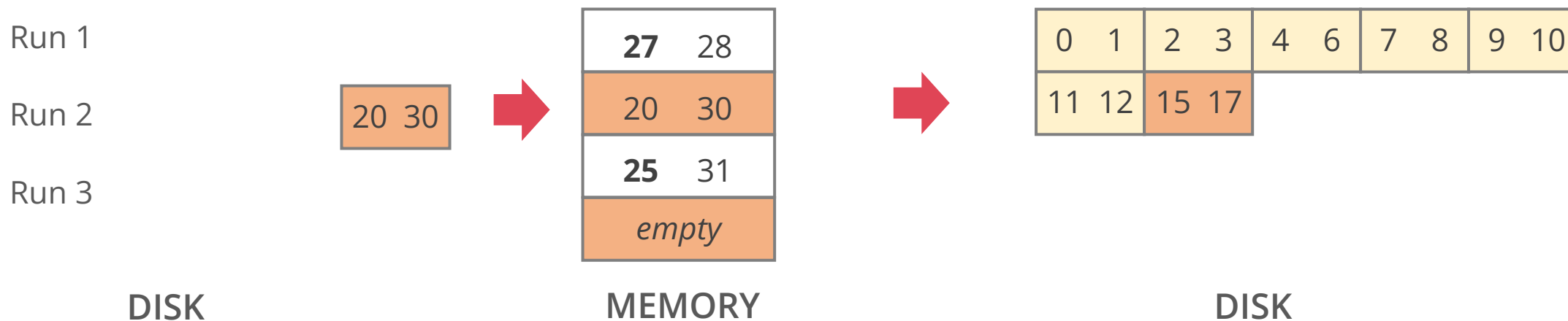


I/O Total (so far): 35

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



I/O Total (so far): **37**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

27	28
20	30
25	31
empty	

Run 2

Run 3

DISK

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17						

DISK

I/O Total (so far): **37**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

27	28
	30
25	31
20	

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17						

DISK

I/O Total (so far): **37**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

MEMORY

DISK

27	28
	30
25	31
20	

0	1	2	3	4	6	7	8	9	10
11	12	15	17						

I/O Total (so far): **37**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

27	28
	30
	31
20	25

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17						

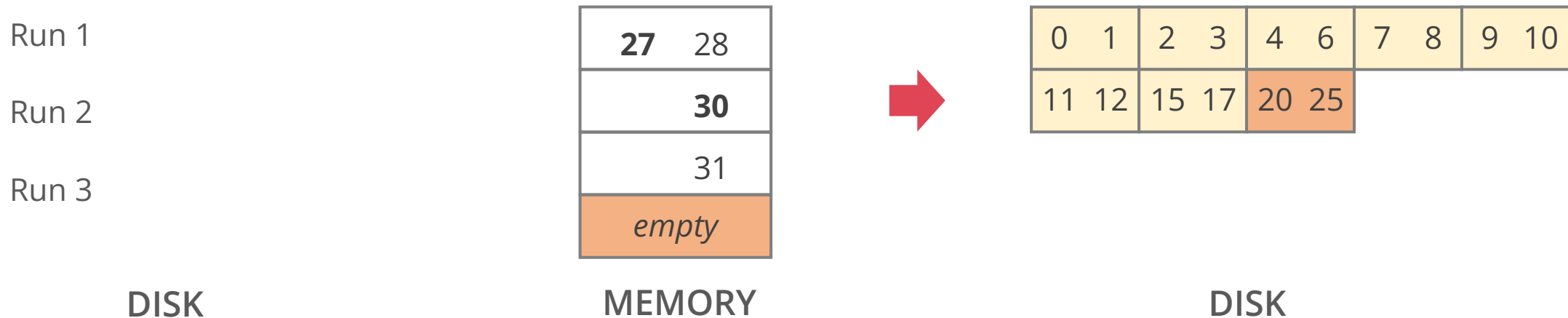
DISK

I/O Total (so far): **37**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



I/O Total (so far): **38**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

27	28
30	
31	
<i>empty</i>	

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25				

DISK

I/O Total (so far): **38**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

28
30
31
27

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25				

DISK

I/O Total (so far): **38**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

28
30
31
27

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25				

DISK

I/O Total (so far): 38

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

<i>empty</i>
30
31
27 28

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25				

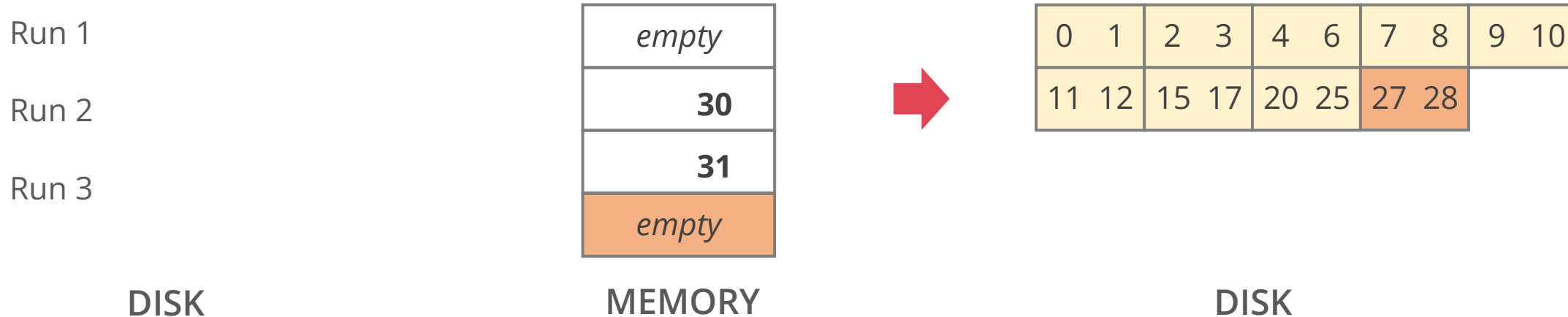
DISK

I/O Total (so far): **38**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



I/O Total (so far): 39

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

<i>empty</i>
30
31
<i>empty</i>

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25	27	28		

DISK

I/O Total (so far): **39**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

<i>empty</i>
<i>empty</i>
31
30

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25	27	28		

DISK

I/O Total (so far): **39**

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

empty
empty
31
30

MEMORY

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25	27	28		

DISK

I/O Total (so far): 39

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk

Run 1

Run 2

Run 3

DISK

<i>empty</i>
<i>empty</i>
<i>empty</i>
30 31

MEMORY

0 1	2 3	4 6	7 8	9 10
11 12	15 17	20 25	27 28	

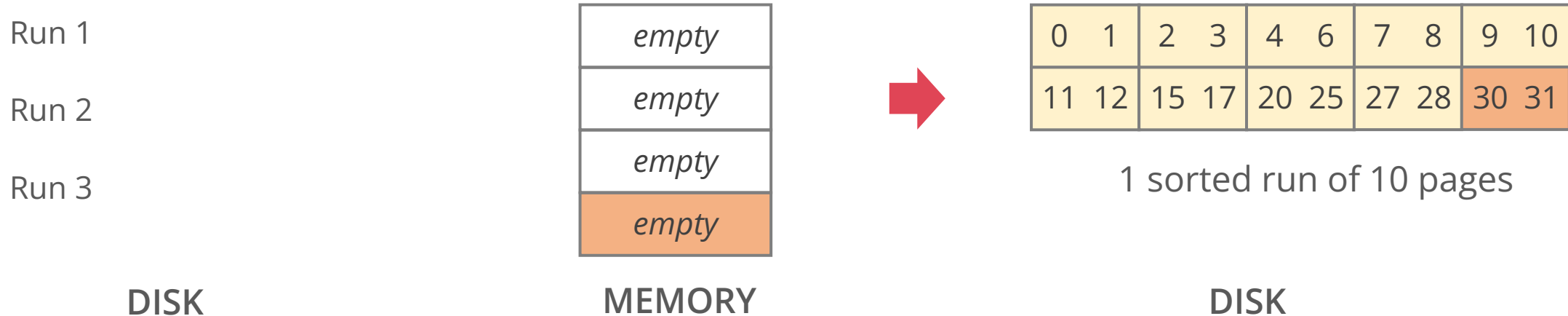
DISK

I/O Total (so far): 39

GENERAL EXTERNAL MERGE SORT

Pass 1

Read 3 sorted runs into memory, write 1 sorted run to disk



I/O Total: 40

SANITY CHECK

$$N = 10, B = 4$$

$$\text{Cost} = 2N * (1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil)$$

$$= 2 * 10 * (1 + \lceil \log_3(2.5) \rceil)$$

$$= 20 * (1 + 1)$$

$$= 40 \text{ I/Os} \quad \checkmark$$

0	1	2	3	4	6	7	8	9	10
11	12	15	17	20	25	27	28	30	31

1 sorted run of 10 pages

I/O Total: **40**

JOINS

JOINS

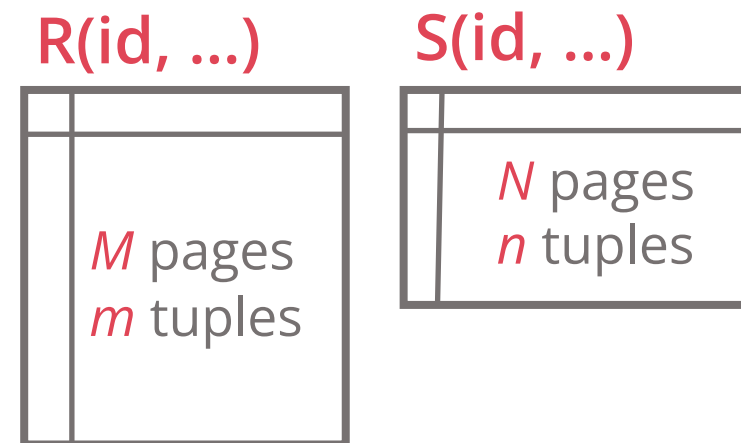
Bit of notation:

M = number of pages in R

m = number of records in R (the **cardinality** of R)

N = number of pages in S

n = number of records in S



We typically exclude the final write's I/O cost

Don't add the cost of writing the joined output to disk

We might decide to stream it to the next operator instead of materializing results!

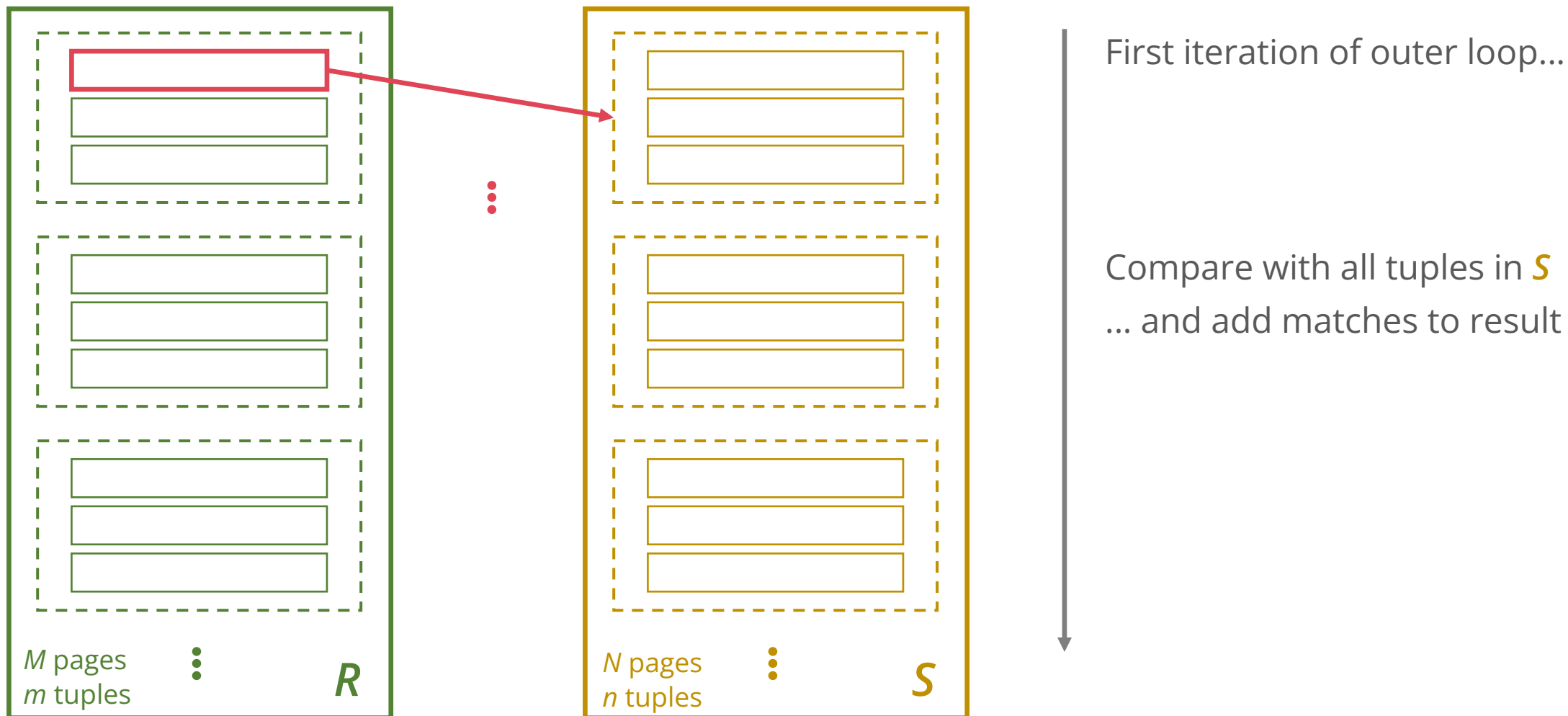
SIMPLE NESTED LOOPS JOIN (SNLJ)

Direct translation of the definition of join into code

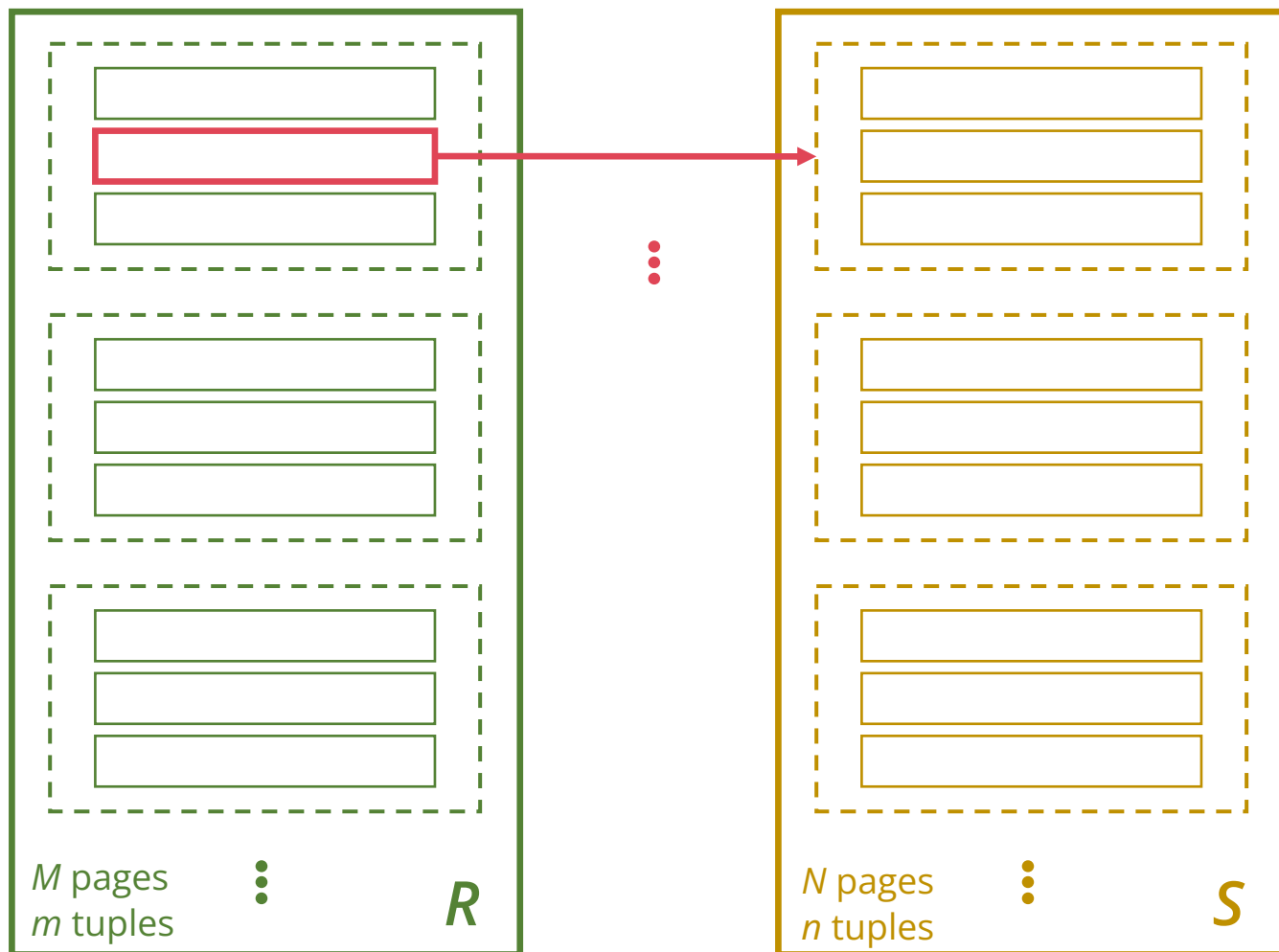
To perform $R \bowtie_{\theta} S$, take each tuple in R and scan through S to find the matching S-tuples!

```
foreach tuple  $r \in R$ :  
  foreach tuple  $s \in S$ :  
    if  $\theta(r, s)$ : output  $r$  joined with  $s$ 
```

SIMPLE NESTED LOOPS JOIN (SNLJ)



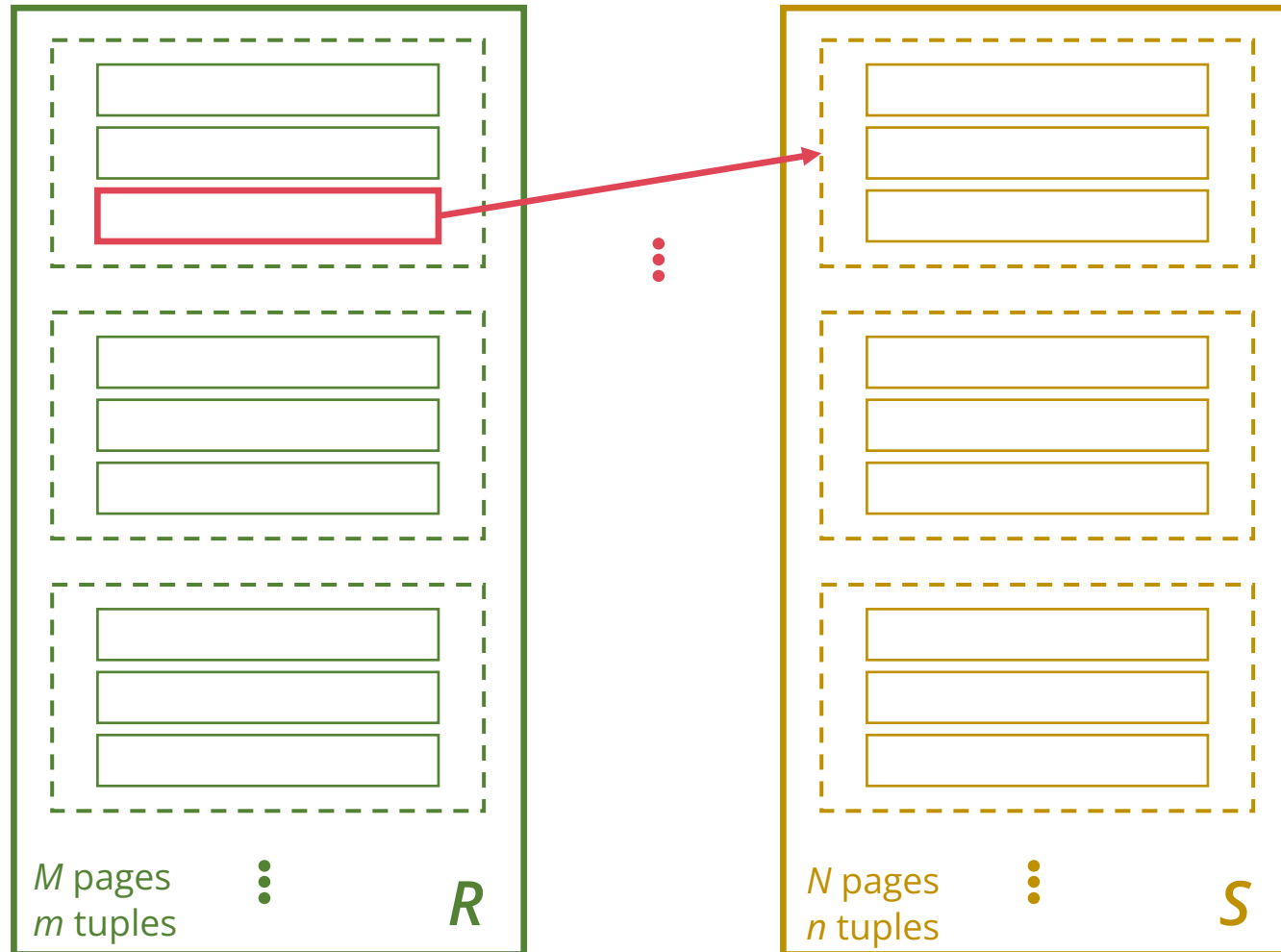
SIMPLE NESTED LOOPS JOIN (SNLJ)



Second iteration of outer loop...

Compare with all tuples in S
... and add matches to result

SIMPLE NESTED LOOPS JOIN (SNLJ)



$$\text{Cost} = M + m \cdot N$$

Flipping the order of R and S
can change the I/O cost

Which relation to use as outer?

PAGE NESTED LOOPS JOIN (PNLJ)

Can we do better?

We scan *S* for every tuple in *R*,

... but we had to load an entire page of *R* into memory to get that tuple!

Instead of finding the tuples in *S* that match a tuple in *R*,

... do the check for all tuples in a page in *R* at once

PAGE NESTED LOOPS JOIN (PNLJ)

SNLJ

```
foreach tuple r ∈ R:  
  foreach tuple s ∈ S:  
    if  $\theta(\mathbf{r}, \mathbf{s})$ : output r joined with s
```

PAGE NESTED LOOPS JOIN (PNLJ)

SNLJ (but with page fetches written out explicitly)

```
foreach page  $P_R \in R$ :  
  foreach tuple  $r \in P_R$ :  
    foreach page  $P_S \in S$ :  
      foreach tuple  $s \in P_S$ :  
        if  $\theta(r, s)$ : output  $r$  joined with  $s$ 
```

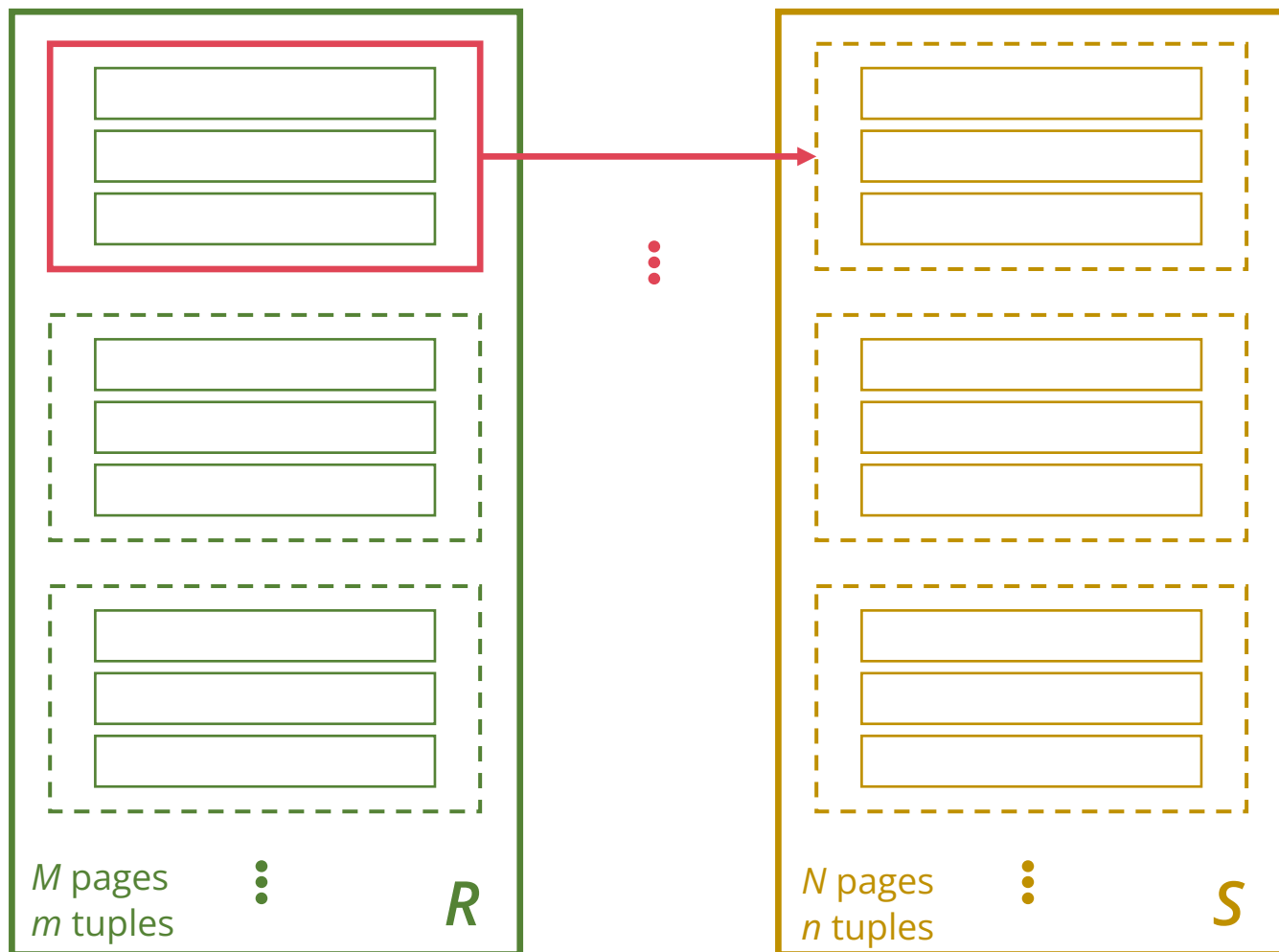
PAGE NESTED LOOPS JOIN (PNLJ)

PNLJ

```
foreach page  $P_R \in R$ :  
  foreach page  $P_S \in S$ :  
    foreach tuple  $r \in P_R$ :  
      foreach tuple  $s \in P_S$ :  
        if  $\theta(r, s)$ : output  $r$  joined with  $s$ 
```

flipped loops

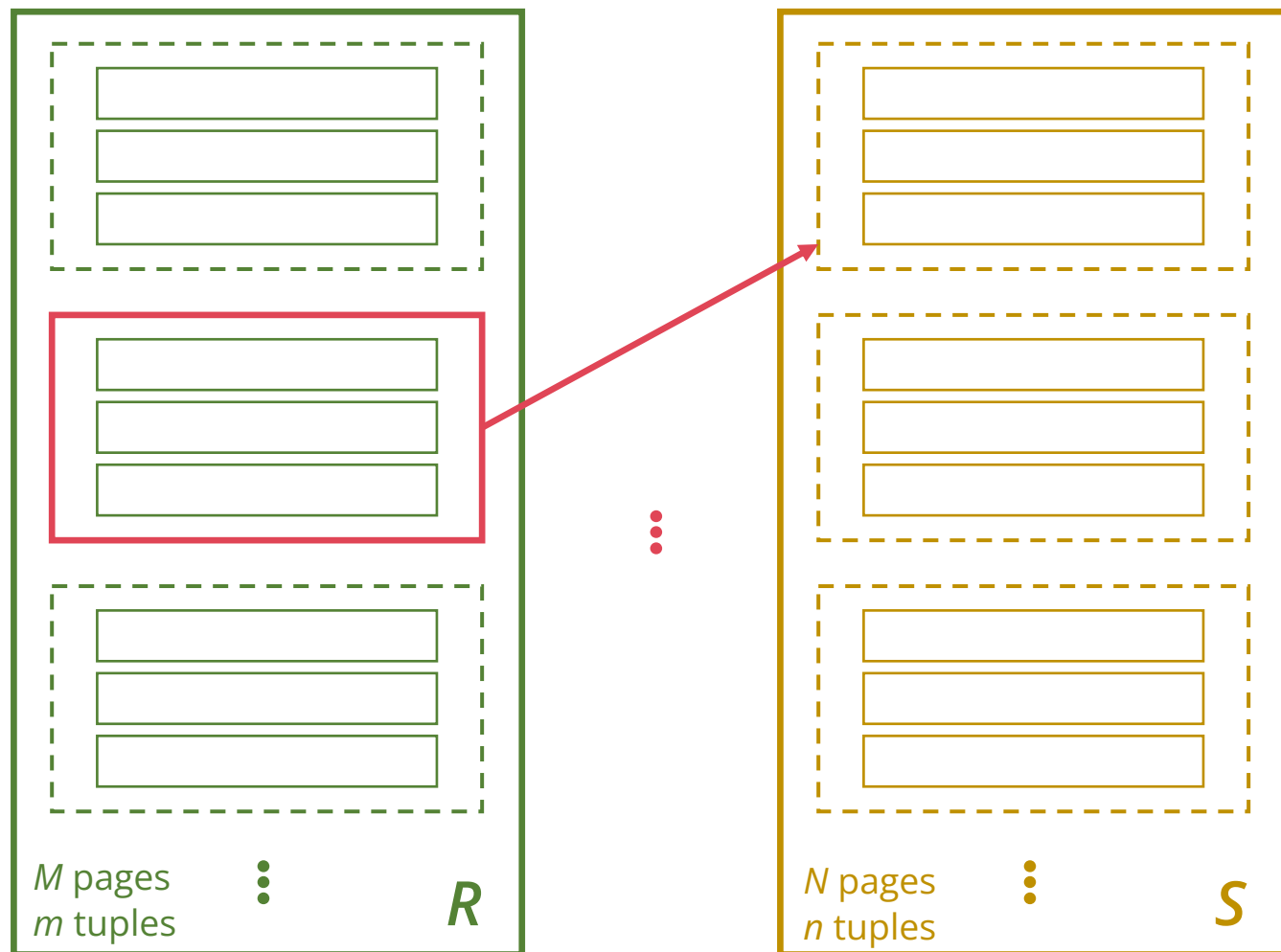
PAGE NESTED LOOPS JOIN (PNLJ)



First iteration of outer loop...

Compare with all tuples in S
... and add matches to result

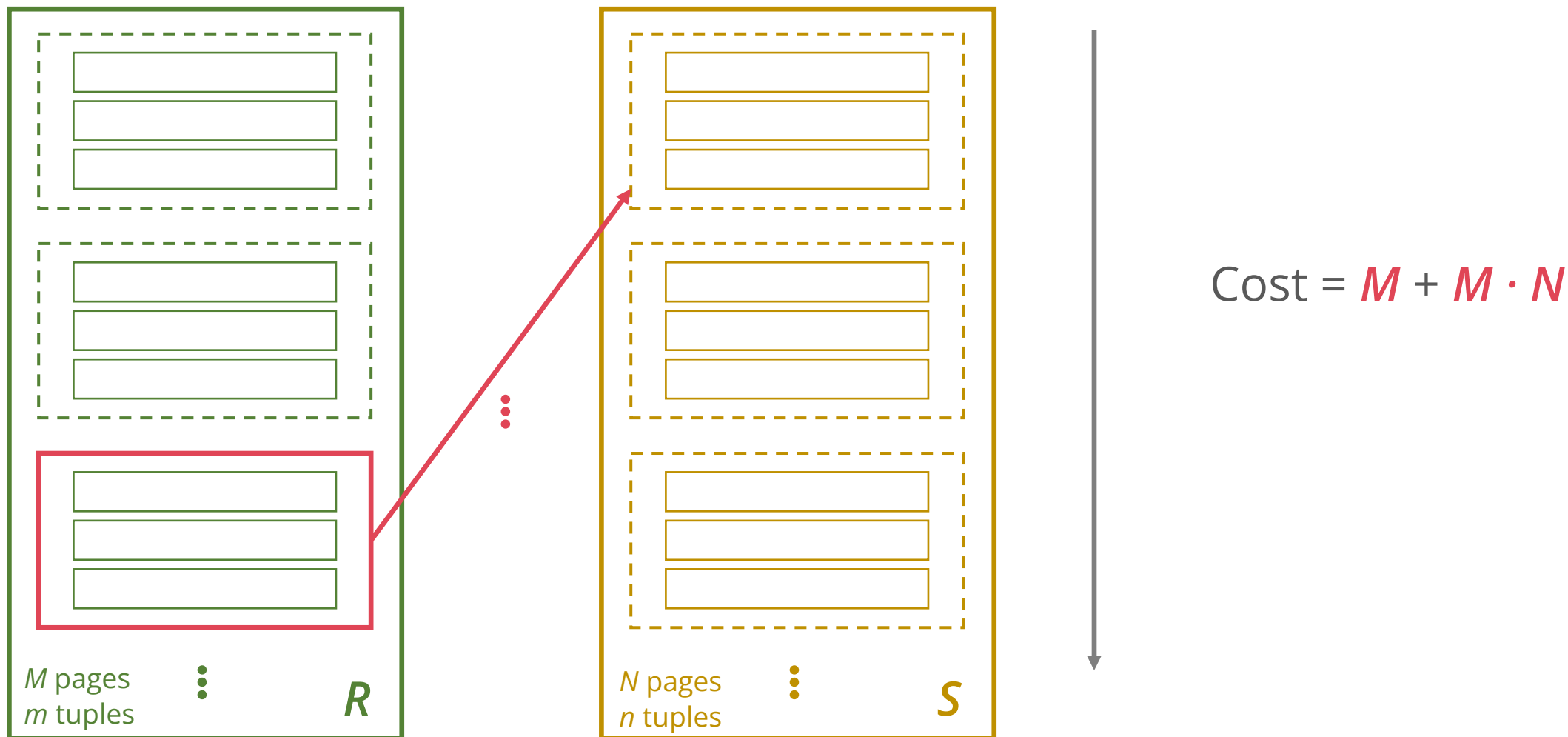
PAGE NESTED LOOPS JOIN (PNLJ)



Second iteration of outer loop...

Compare with all tuples in S
... and add matches to result

PAGE NESTED LOOPS JOIN (PNLJ)



BLOCK NESTED LOOPS JOIN (BNLJ)

Can we do even better?

We only use three page of memory for PNLJ

... (one buffer for R , one buffer for S , one output buffer),

... but we usually have more memory!

Instead of fetching one page of R at a time,

.. why not fetch as many pages of R as we can fit ($B - 2$ pages)!

BLOCK NESTED LOOPS JOIN (PNLJ)

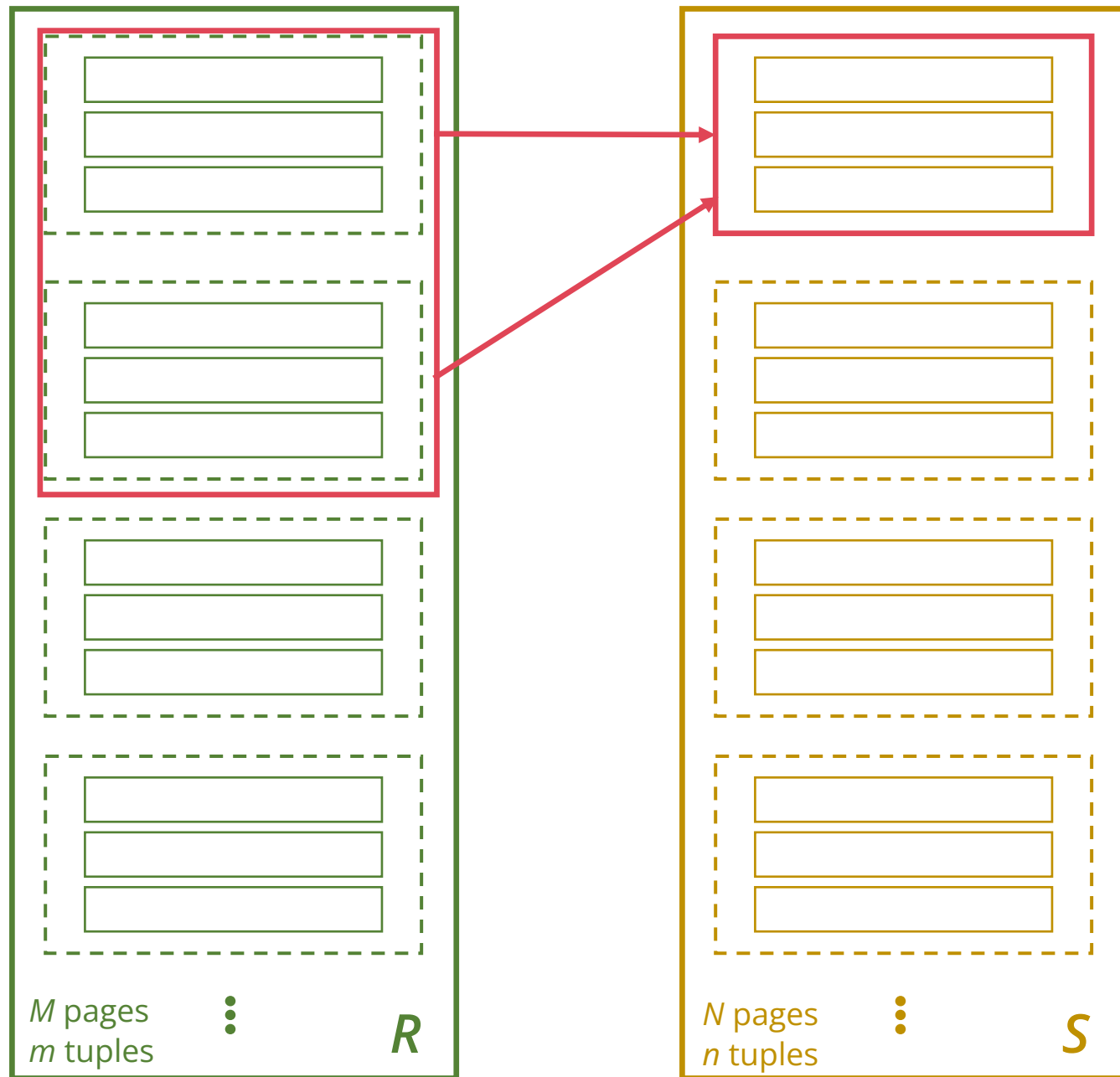
PNLJ

```
foreach page  $P_R \in R$ :  
  foreach page  $P_S \in S$ :  
    foreach tuple  $r \in P_R$ :  
      foreach tuple  $s \in P_S$ :  
        if  $\theta(r, s)$ : output  $r$  joined with  $s$ 
```

BLOCK NESTED LOOPS JOIN (PNLJ)

BNLJ

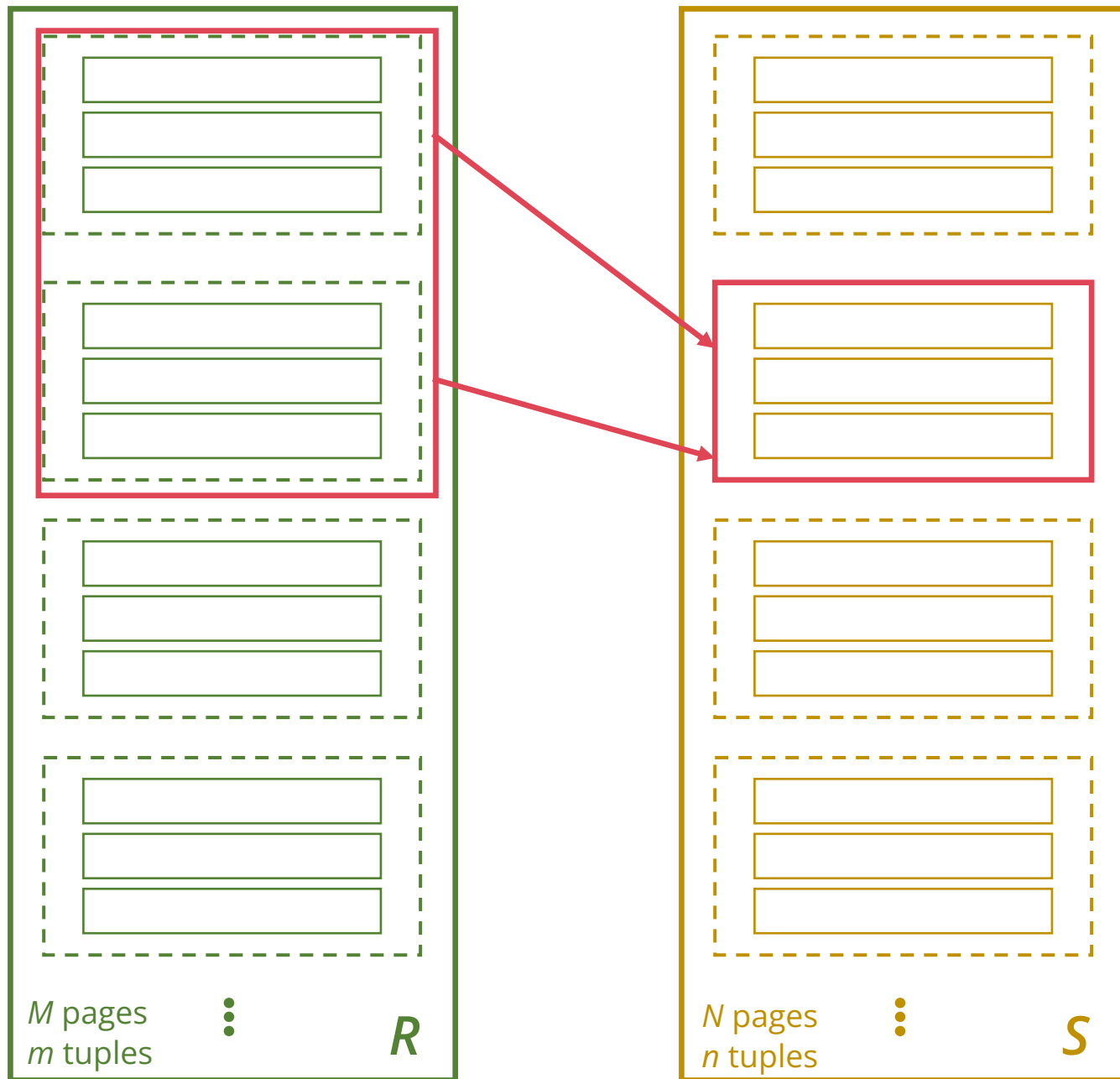
```
foreach block  $C_R$  of  $B-2$  pages  $\in R$ :  
  foreach page  $P_S \in S$ :  
    foreach tuple  $r \in C_R$ :  
      foreach tuple  $s \in P_S$ :  
        if  $\theta(r, s)$ : output  $r$  joined with  $s$ 
```



$$B = 4$$

First iteration of outer loop...

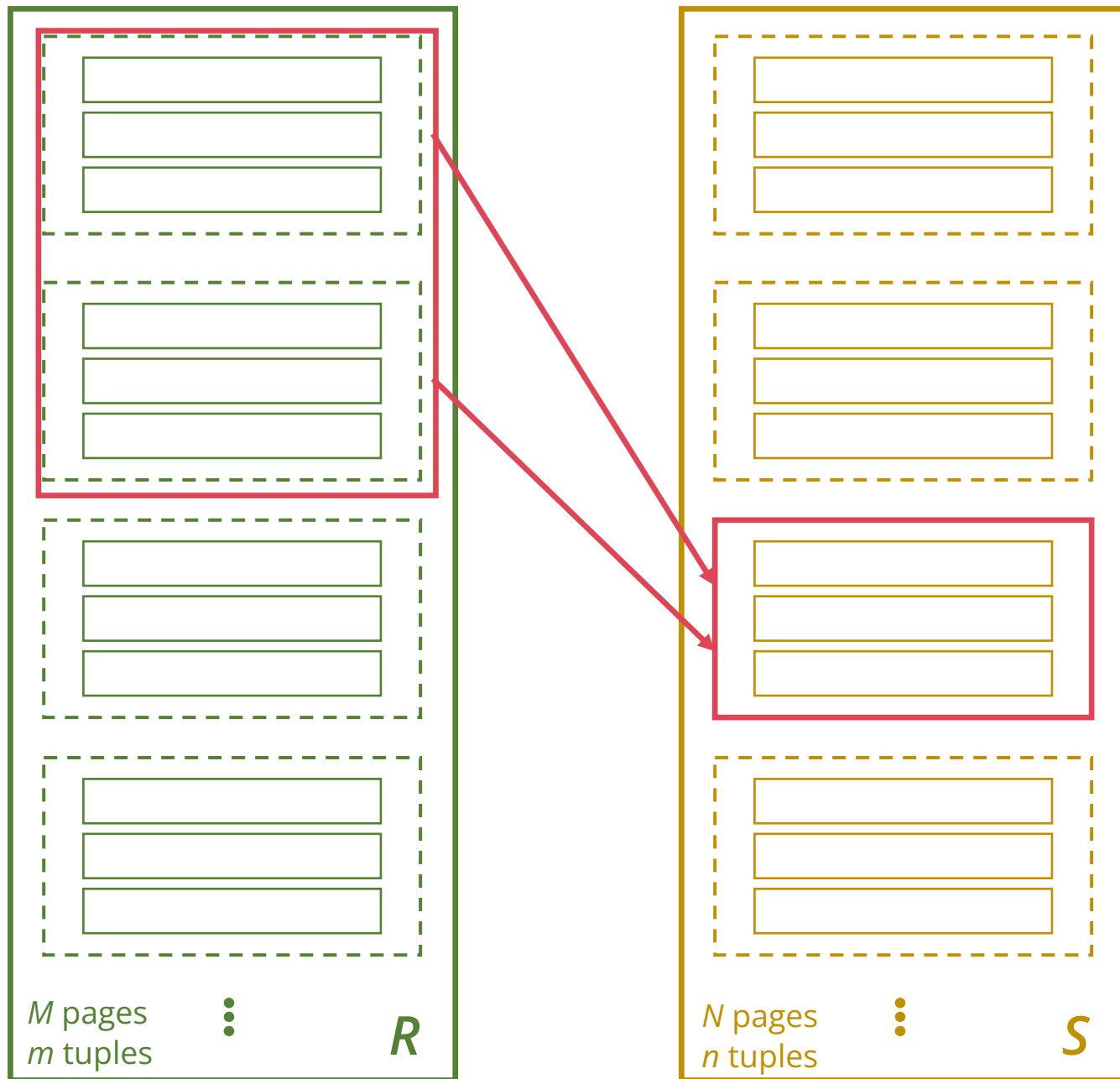
Compare with all tuples in S
... and add matches to result



$$B = 4$$

First iteration of outer loop...

Compare with all tuples in S
... and add matches to result

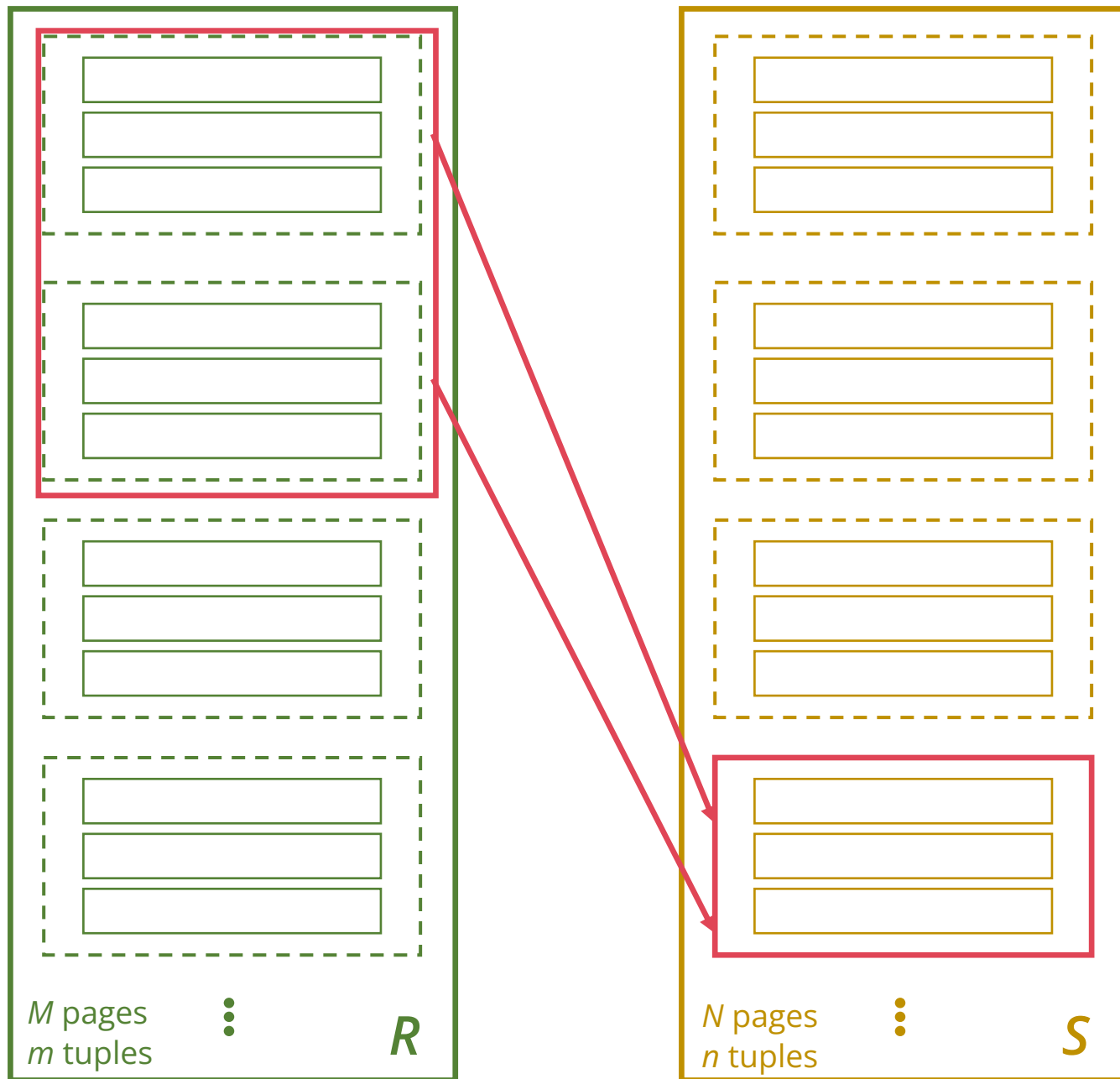


$$B = 4$$

First iteration of outer loop...

Compare with all tuples in S
... and add matches to result

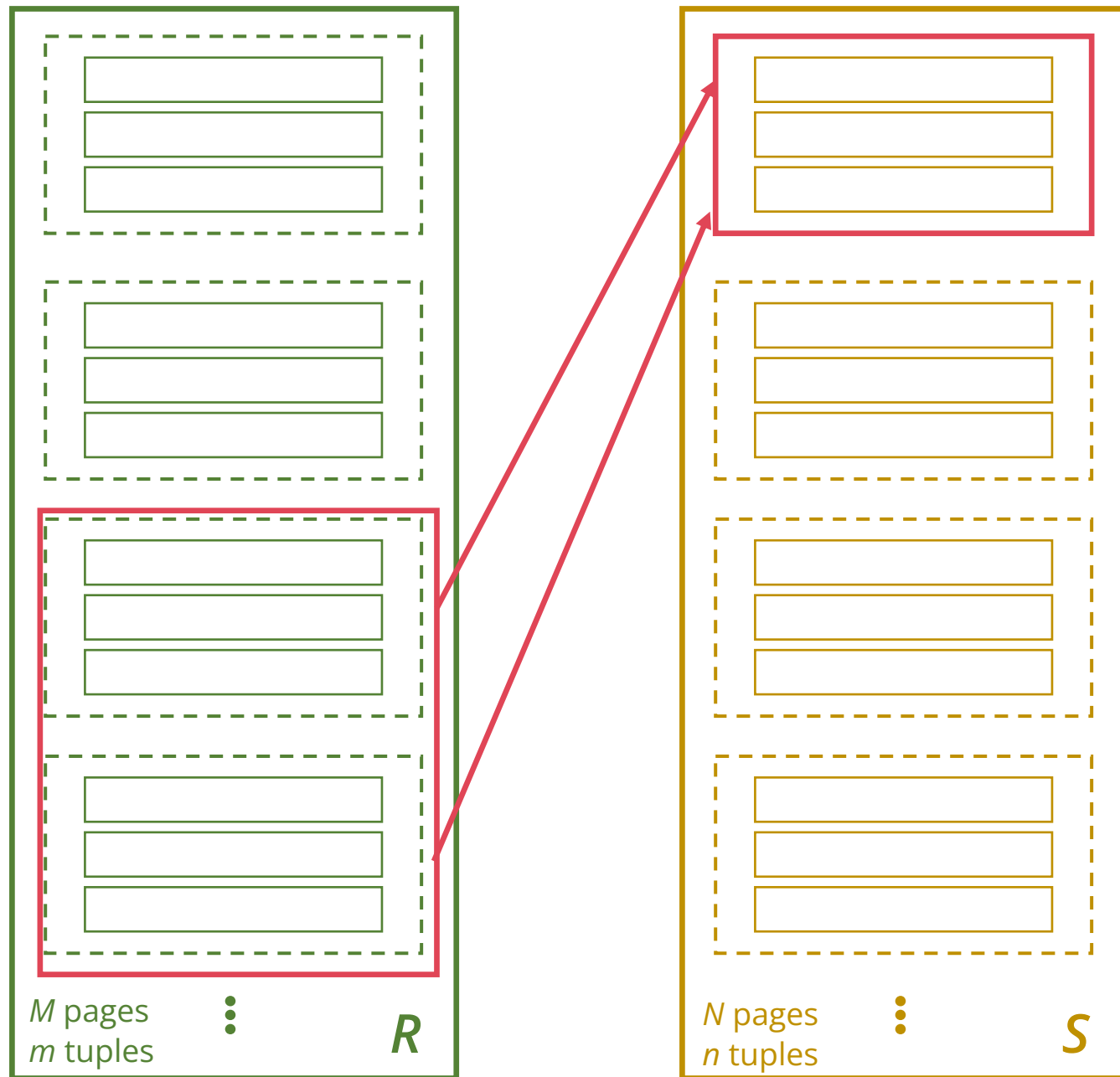




$$B = 4$$

First iteration of outer loop...

Compare with all tuples in S
... and add matches to result

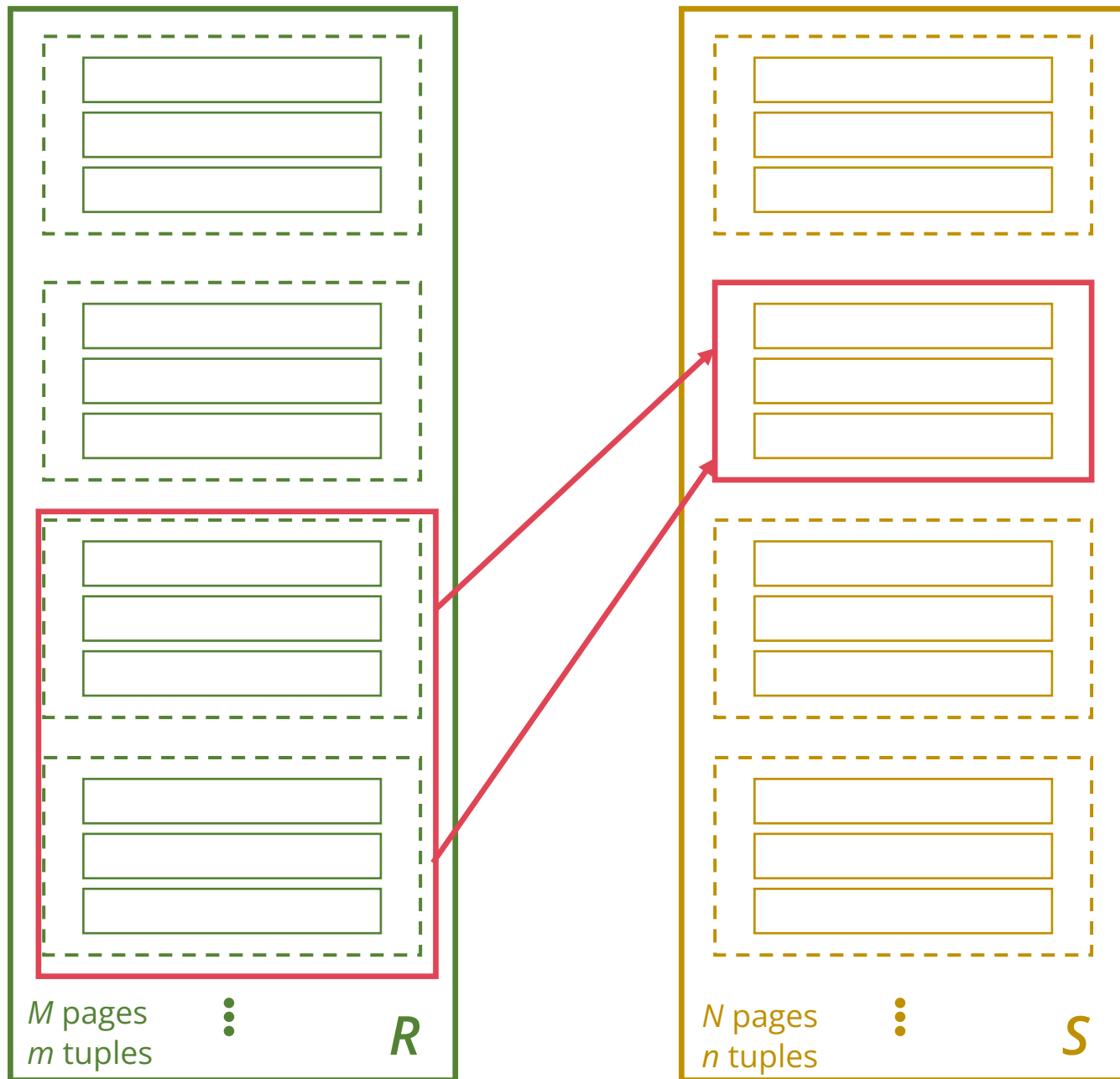


$$B = 4$$

Second iteration of outer loop...

Compare with all tuples in S
... and add matches to result

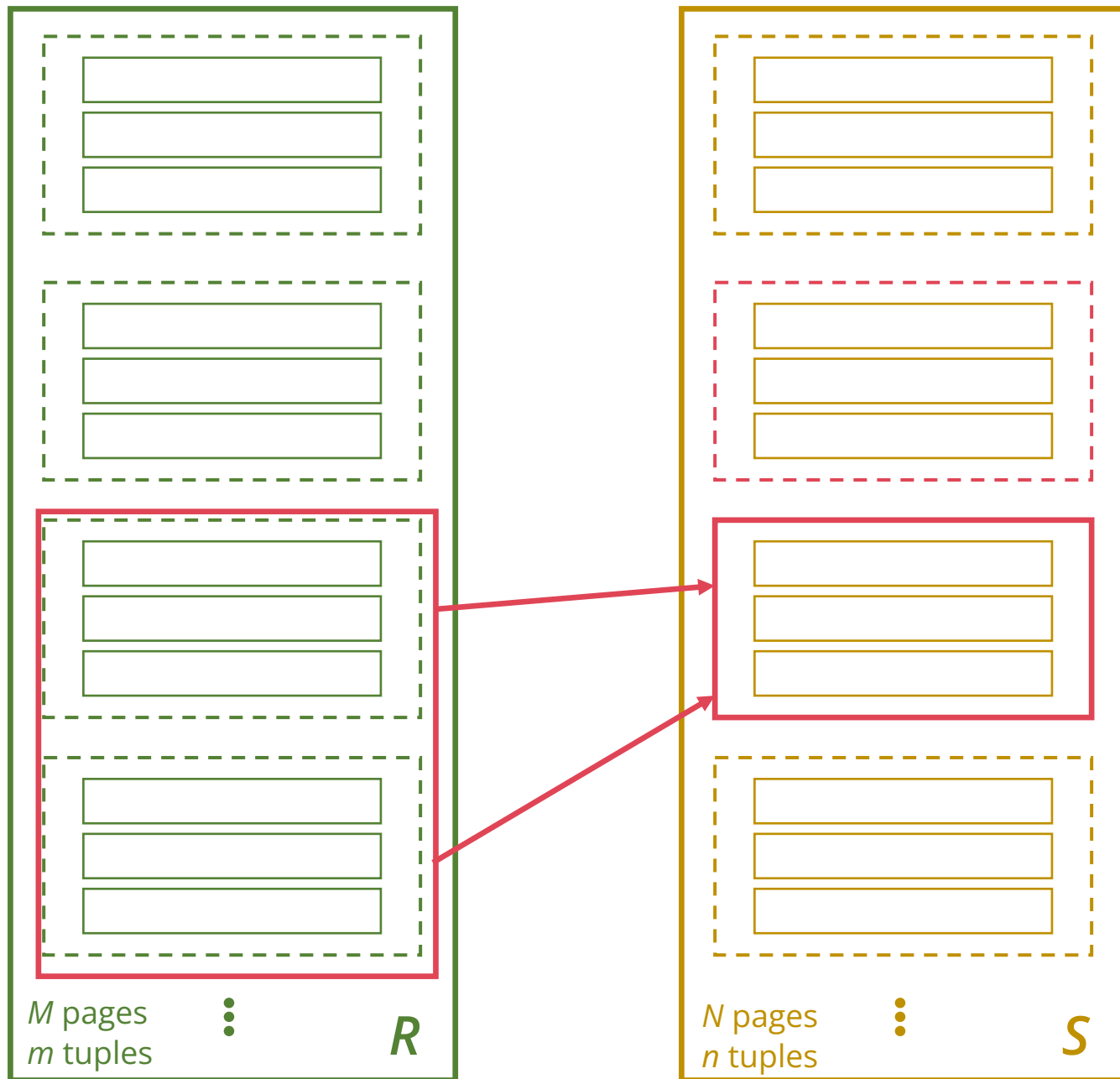
$$B = 4$$



Second iteration of outer loop...

Compare with all tuples in S
... and add matches to result

$$B = 4$$



Second iteration of outer loop...

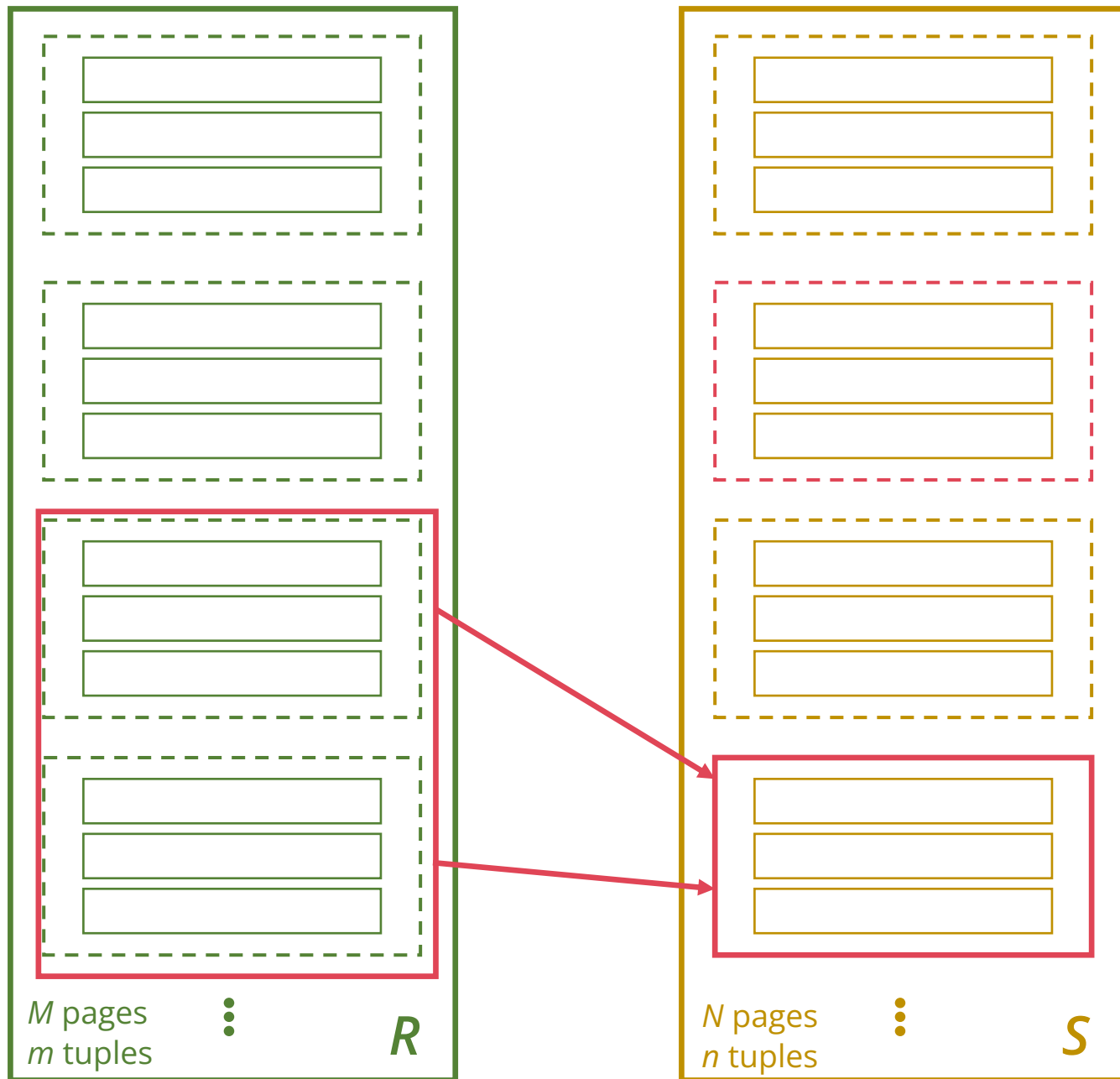
Compare with all tuples in S
... and add matches to result

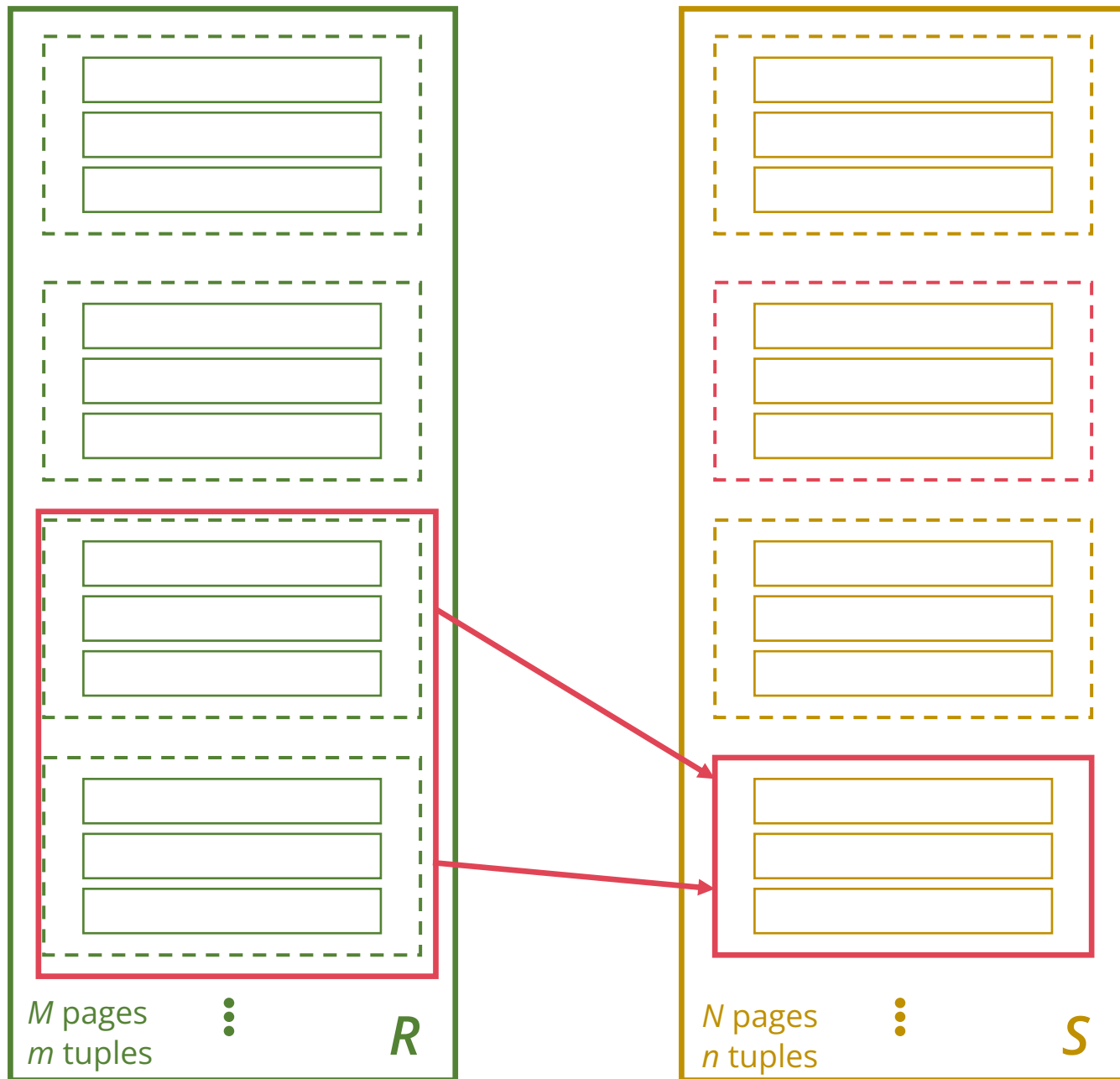


$$B = 4$$

Second iteration of outer loop...

Compare with all tuples in S
... and add matches to result





$$B = 4$$

Cost

$$= M + (\# \text{ blocks in } R) \cdot N$$

$$= M + \lceil M / \text{chunk size} \rceil \cdot N$$

$$= M + \lceil M / B - 2 \rceil \cdot N$$

INDEX NESTED LOOPS JOIN (INLJ)

A join is essentially

```
foreach tuple r ∈ R:  
  foreach tuple s ∈ S that satisfies  $\theta(\mathbf{r}, \mathbf{s})$ :  
    output r joined with s
```


INDEX NESTED LOOPS JOIN (INLJ)

An index on **S** allows us to do the inner loop efficiently!

```
foreach tuple r ∈ R:  
  foreach tuple s ∈ S that satisfies  $\theta(\mathbf{r}, \mathbf{s})$ :  
    (found using the index)  
    output r joined with s
```

INDEX NESTED LOOPS JOIN (INLJ)

Cost = $M + m * \text{cost to find matching } S \text{ tuples}$

M from scanning through R

Cost to find matching S tuples via **tree index**

Variant **A**: cost to traverse root to leaf + read all the leaves with matching tuples

Variants **B** or **C**: cost of retrieving RIDs (similar to Variant **A**) + cost to fetch actual records

1 I/O per page if clustered, 1 I/O per matching tuple if not

INDEX NESTED LOOPS JOIN (INLJ)

Cost = M + m * cost to find matching S tuples

M from scanning through R

Cost to find matching S tuples via hash index

1-2 I/Os to reach the target bucket

Then scan pages in that bucket

May stop early if search key values are unique and we found a match!

INDEX NESTED LOOPS JOIN (INLJ)

Cost = $M + m * \text{cost to find matching } S \text{ tuples}$

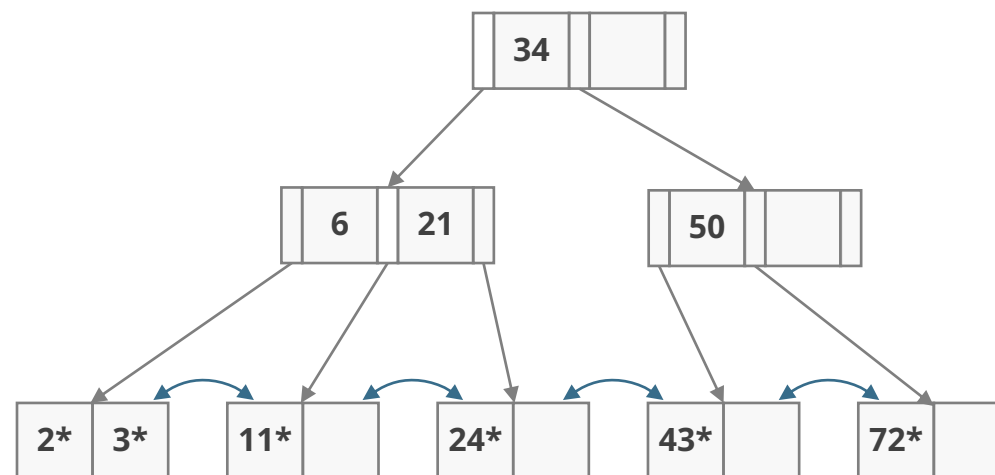
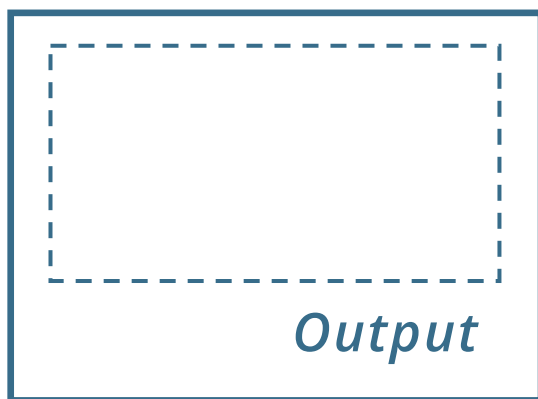
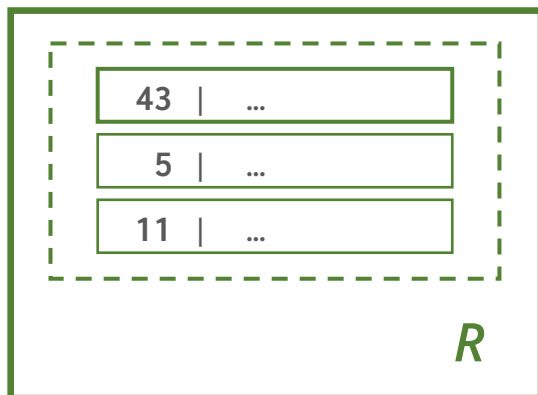
M from scanning through R

If we have **no index**

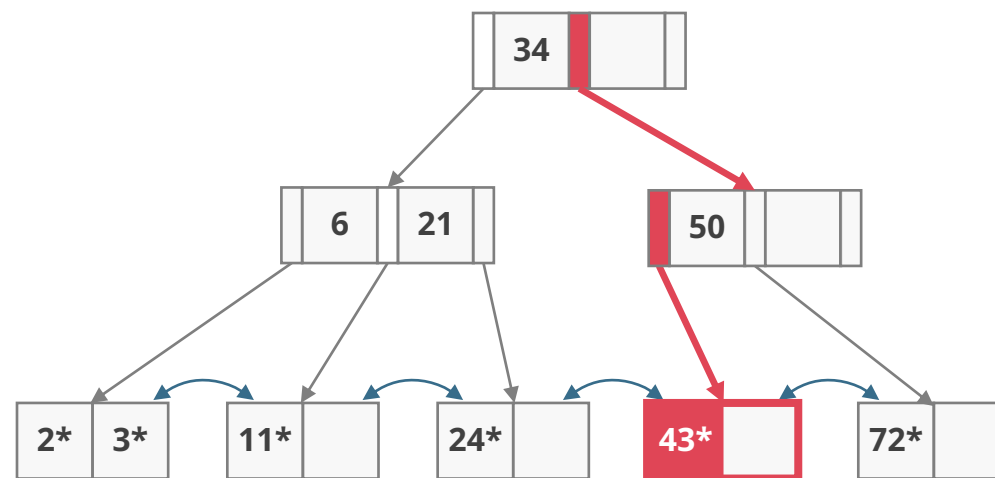
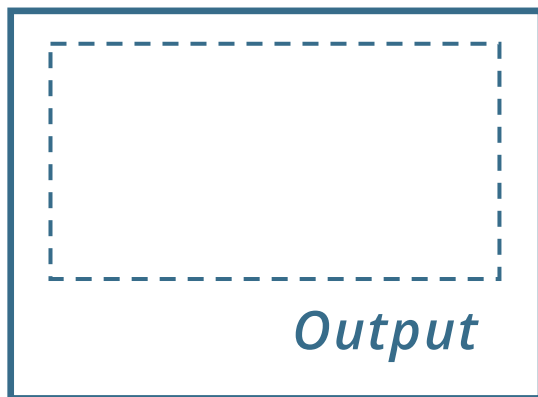
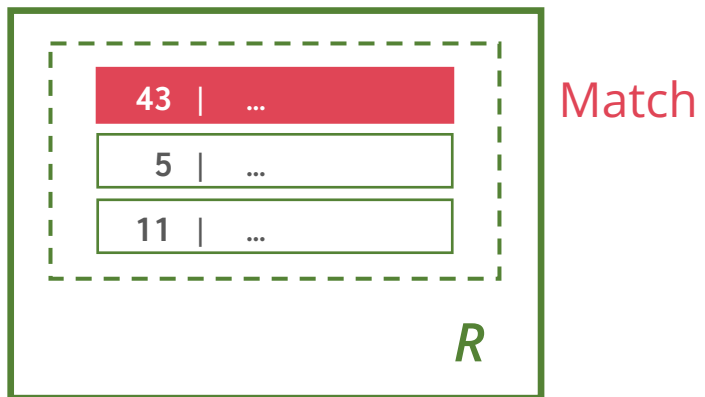
Then the only way to search for matching S tuples is by scanning all of $S \Rightarrow \text{SNLJ}$

Cost to find matching S tuples is then N , giving us the formula for SNLJ cost

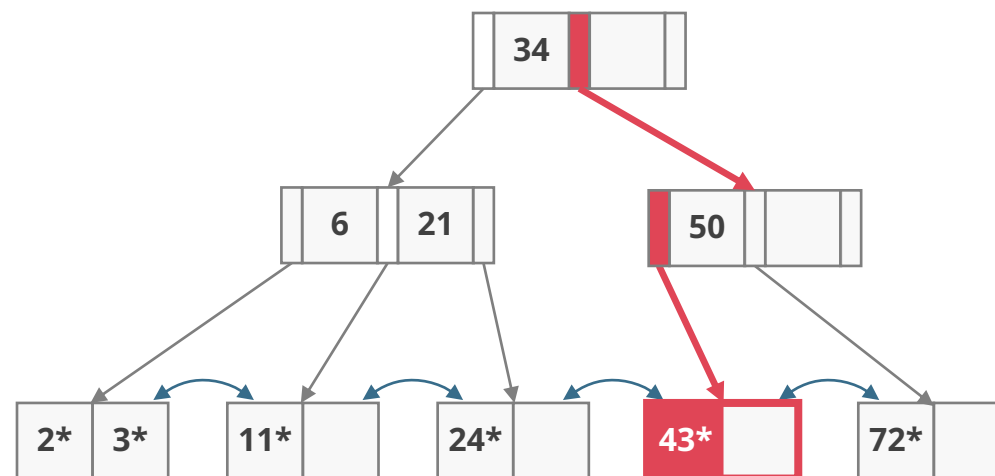
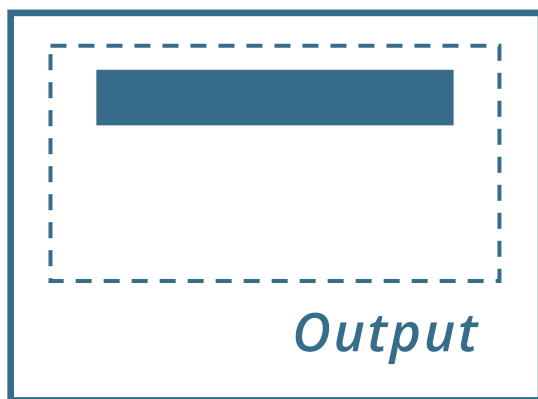
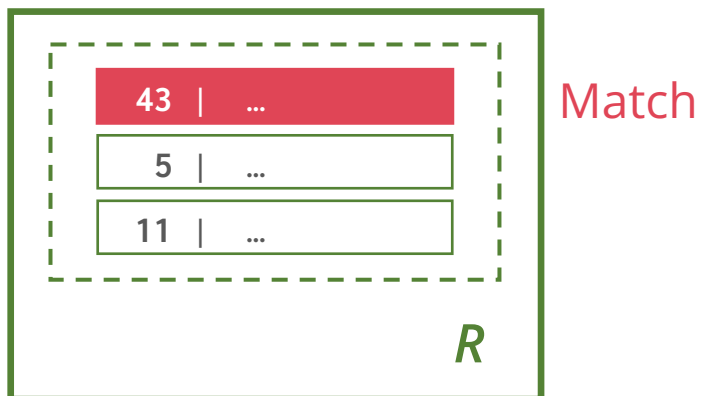
INDEX NESTED LOOPS JOIN (INLJ)



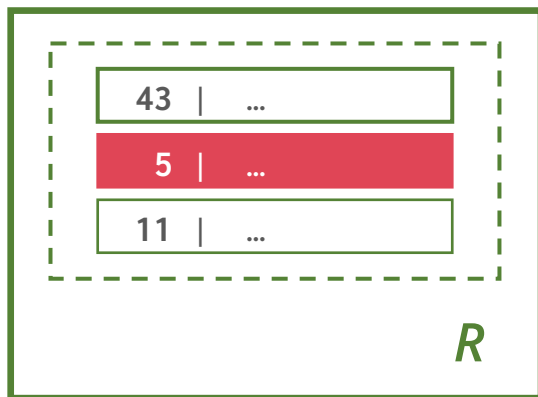
INDEX NESTED LOOPS JOIN (INLJ)



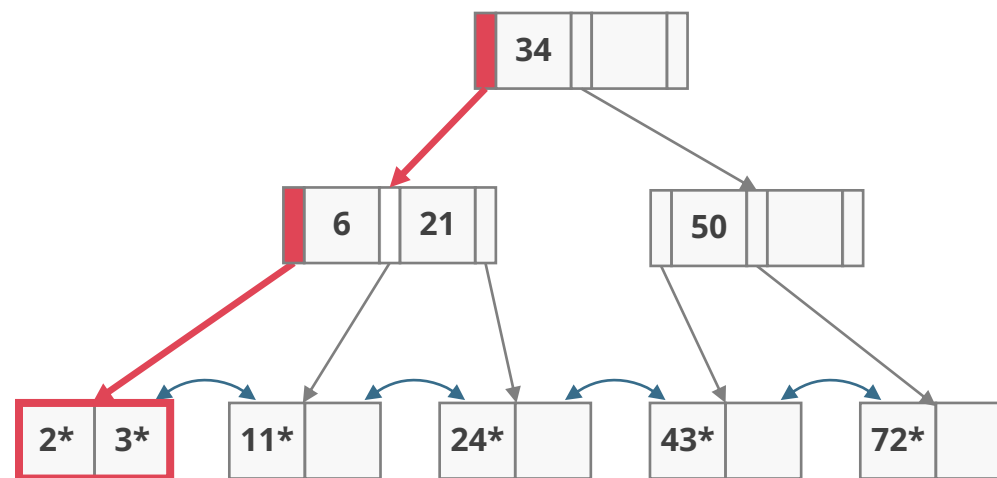
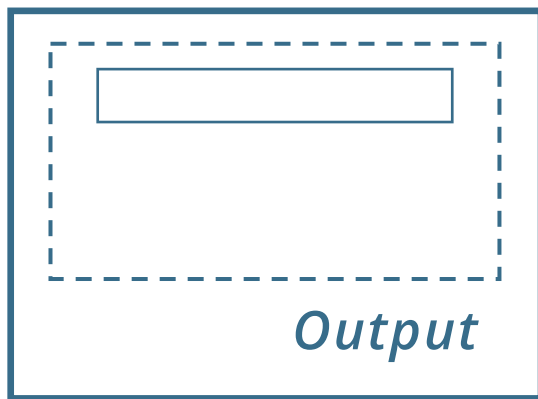
INDEX NESTED LOOPS JOIN (INLJ)



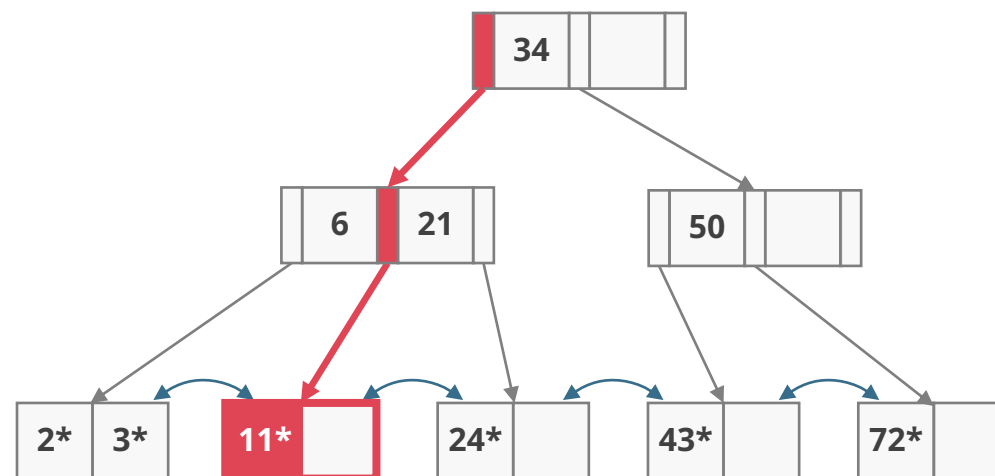
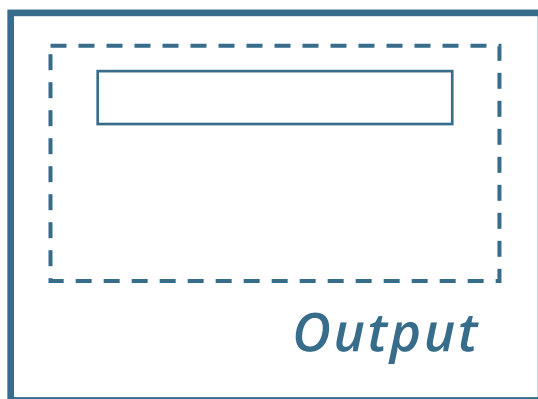
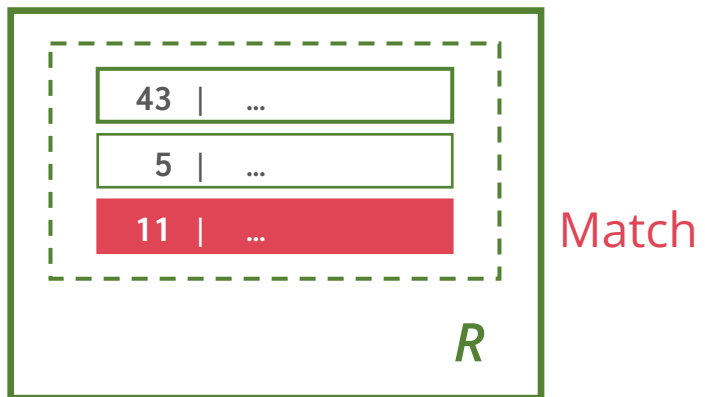
INDEX NESTED LOOPS JOIN (INLJ)



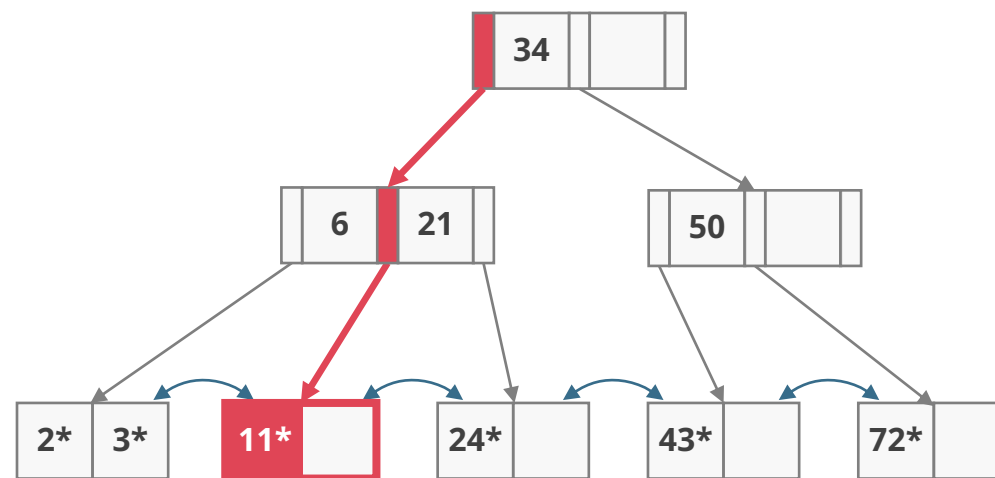
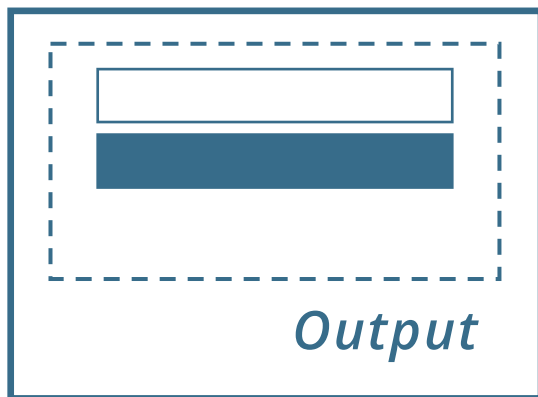
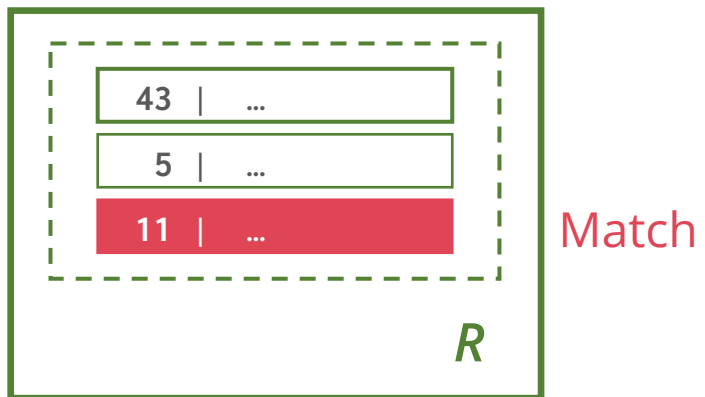
Not a match



INDEX NESTED LOOPS JOIN (INLJ)



INDEX NESTED LOOPS JOIN (INLJ)



SORT MERGE JOIN (SMJ)

What if we process the data a bit before we join things together?

For example, sort both relations first! Then we can join them efficiently

In some cases, we might even have one of the relations already sorted on the right key, and then we don't even have to spend time sorting it!

SORT MERGE JOIN (SMJ)

First step: **sort** both ***R*** and ***S*** (with external sorting)

Second step: **merge** matching tuples from ***R*** and ***S*** together

We do this efficiently by moving iterators over sorted ***R*** and sorted ***S*** in lockstep:
move the iterator with the smaller key

We know that this key is smaller than all remaining key values in the other relation,
so we're completely done joining that tuple!

SORT MERGE JOIN (SMJ)

First step: **sort** both **R** and **S** (with external sorting)

Second step: **merge** matching tuples from **R** and **S** together

Need a bit more care than this:

we might have multiple tuples in **R** matching with multiple tuples in **S**

Mark the first matching tuple in **S**

Match tuples with the first matching tuple in **R**,

Then **reset** the iterator to the mark

... so we can go through the tuples in **S** again for the second matching tuple in **R**

SORT MERGE JOIN (SMJ)

```
while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta



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

SORT MERGE JOIN (SMJ)

```
while not done:
```

```
    while (r < s): advance r
```

```
    while (r > s): advance s
```

```
    mark s    // same start of "block"
```

```
    while (r == s):
```

```
        // outer loop over r
```

```
        while (r == s):
```

```
            // inner loop over s
```

```
            output r joined with s
```

```
            advance s
```

```
        reset s to mark
```

```
        advance r
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta




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


SORT MERGE JOIN (SMJ)

```
while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

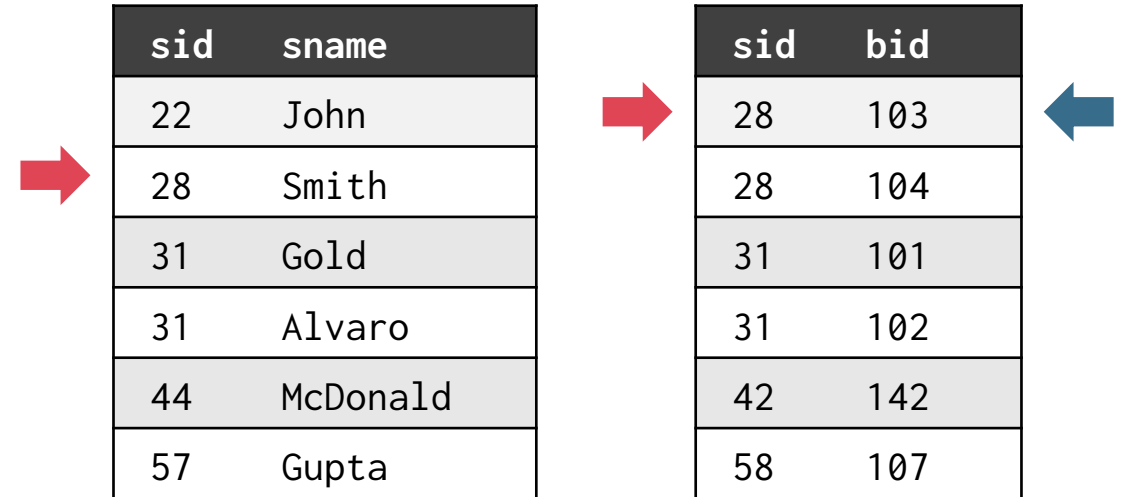


sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

SORT MERGE JOIN (SMJ)

```
while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
```

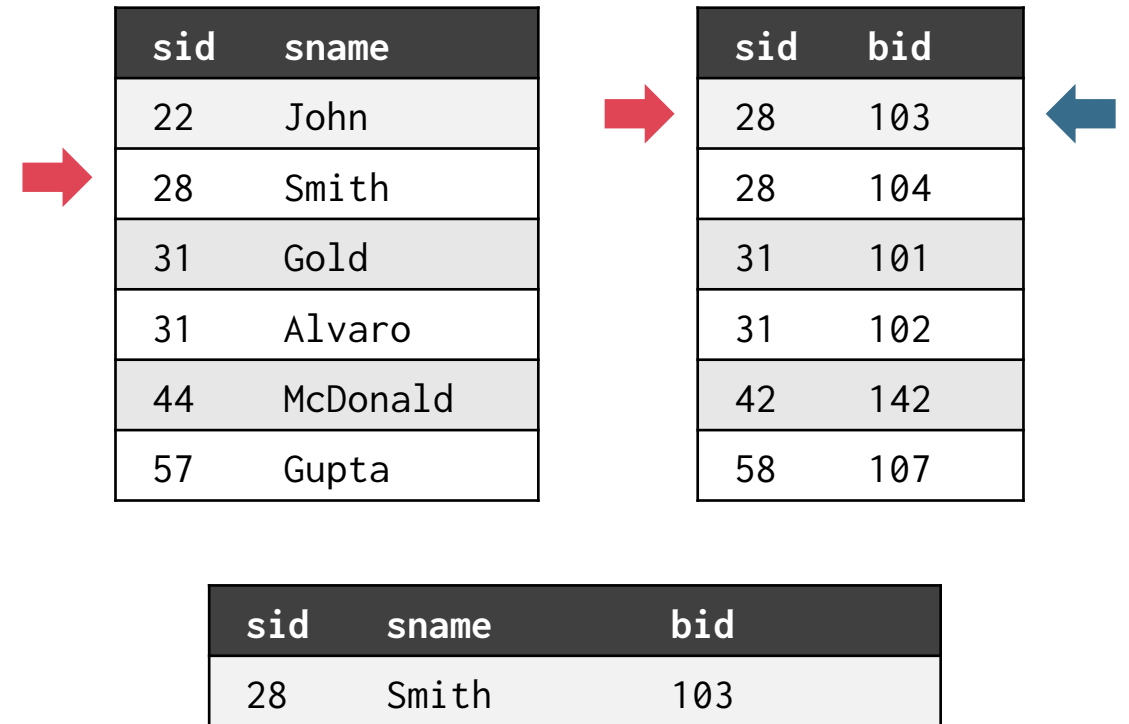


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

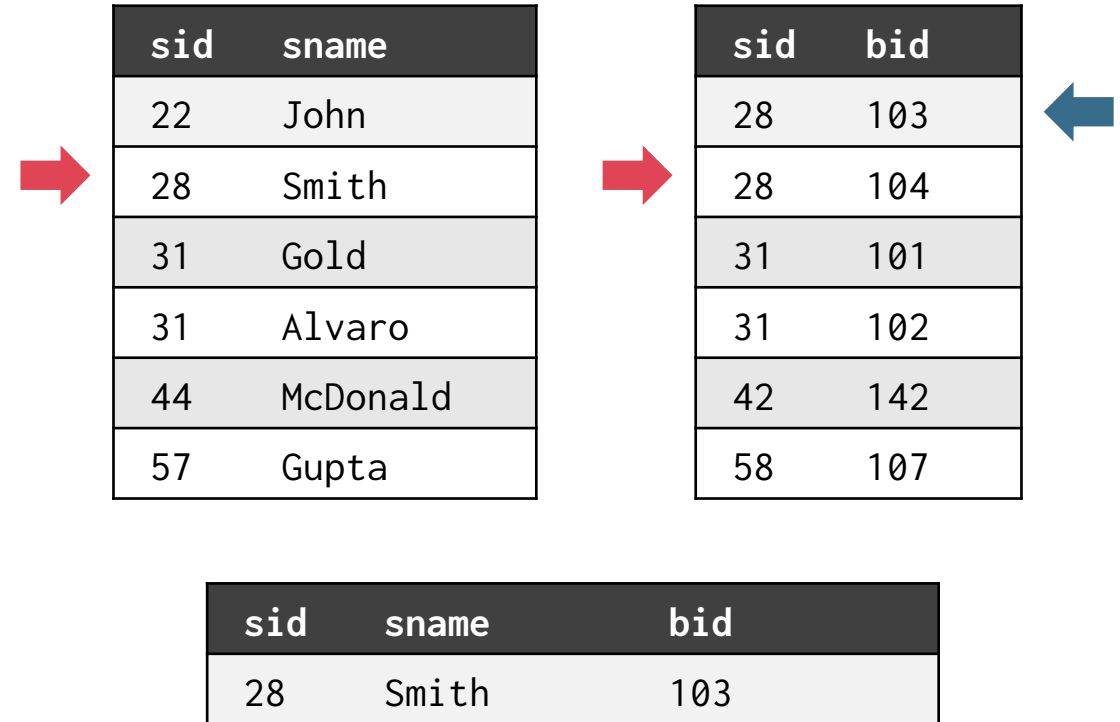


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

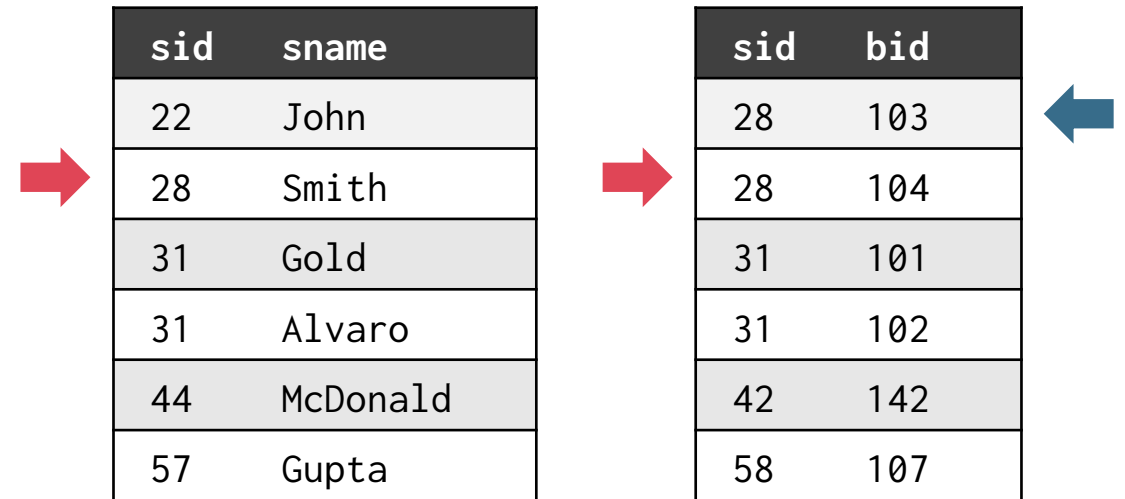


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



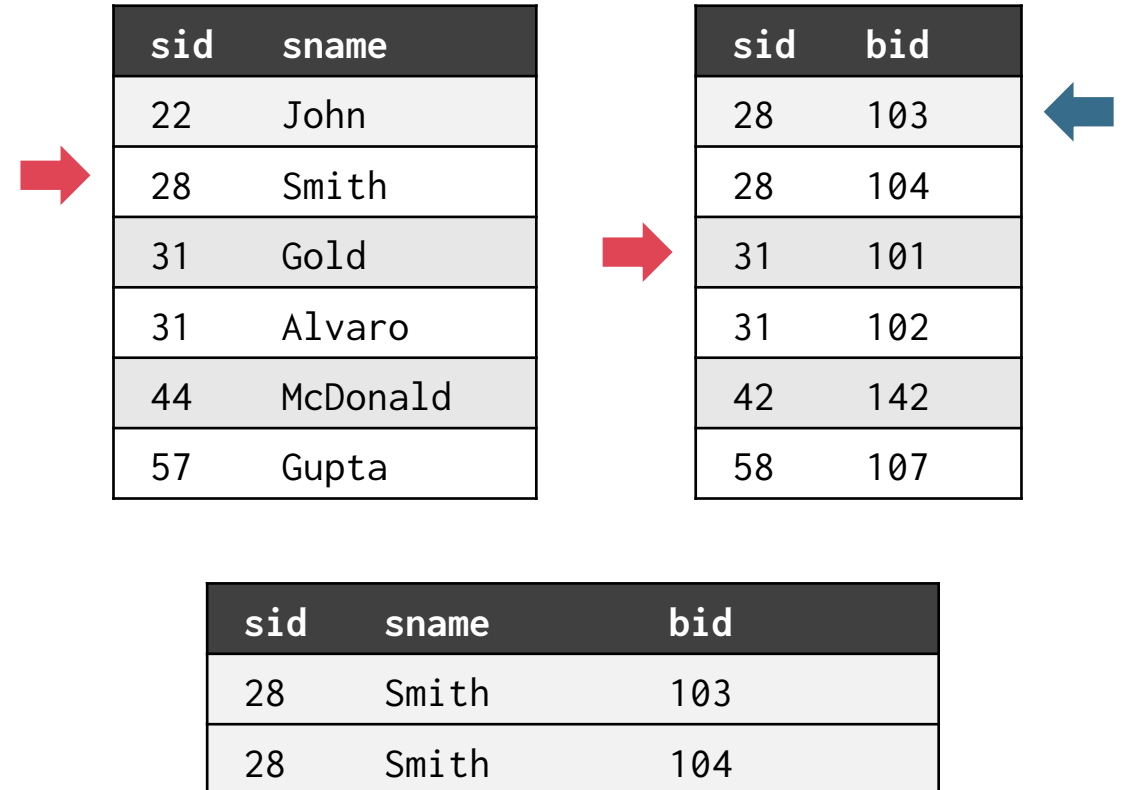
sid	sname	bid
28	Smith	103
28	Smith	104

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

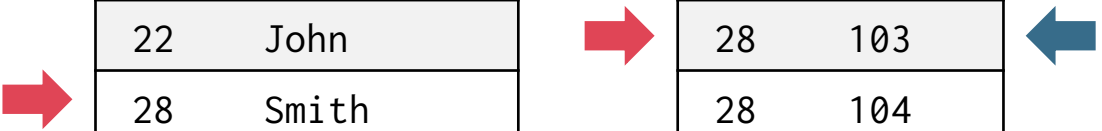


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

sid	sname	bid
28	Smith	103
28	Smith	104

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



SORT MERGE JOIN (SMJ)

```
while not done:
```

```
  while (r < s): advance r
```

```
  while (r > s): advance s
```

```
  mark s    // same start of "block"
```

```
  while (r == s):
```

```
    // outer loop over r
```

```
    while (r == s):
```

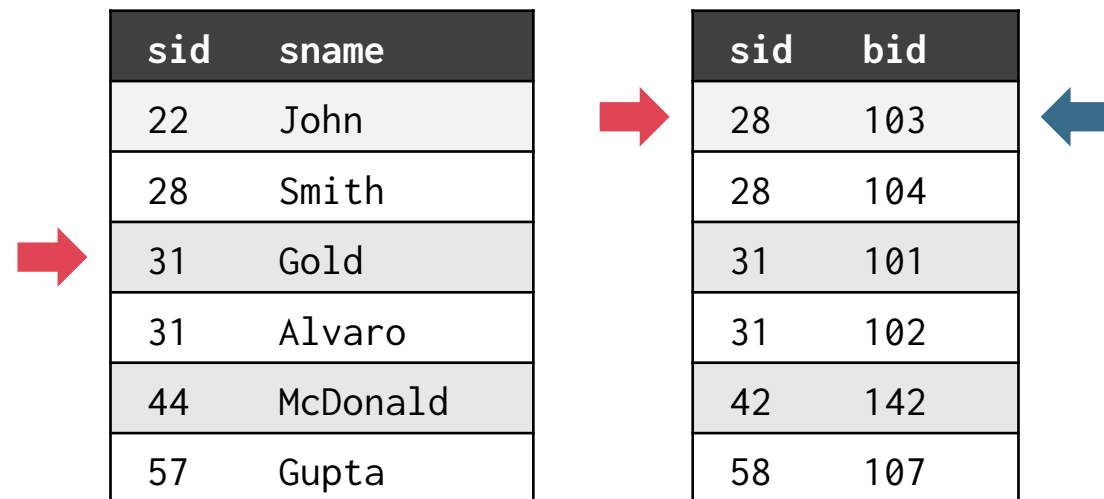
```
      // inner loop over s
```

```
      output r joined with s
```

```
      advance s
```

```
    reset s to mark
```

```
    advance r
```



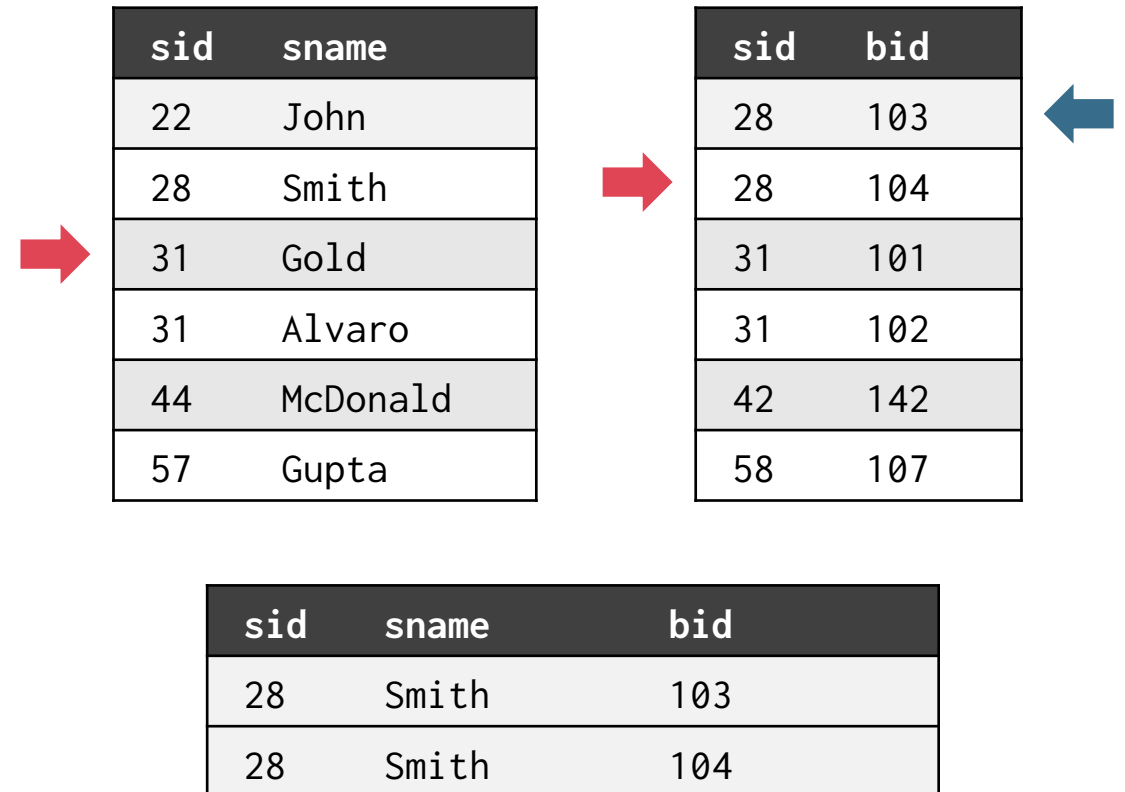
sid	sname	bid
28	Smith	103
28	Smith	104

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

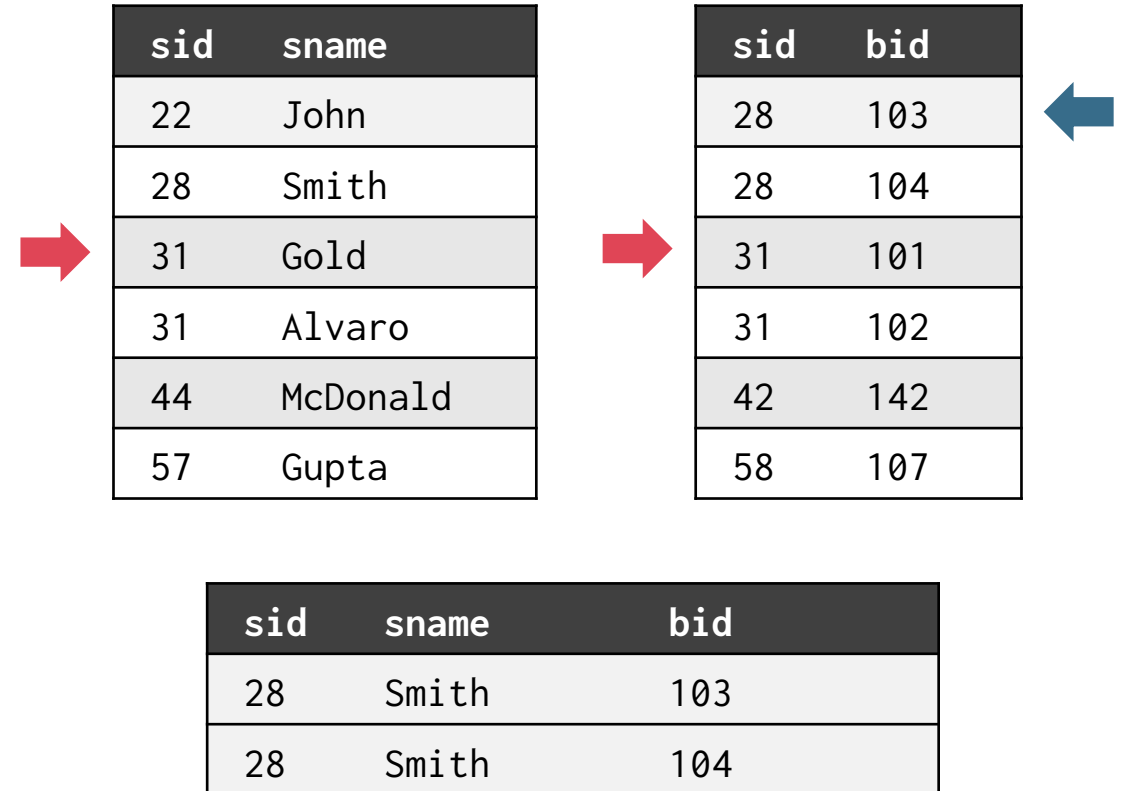


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

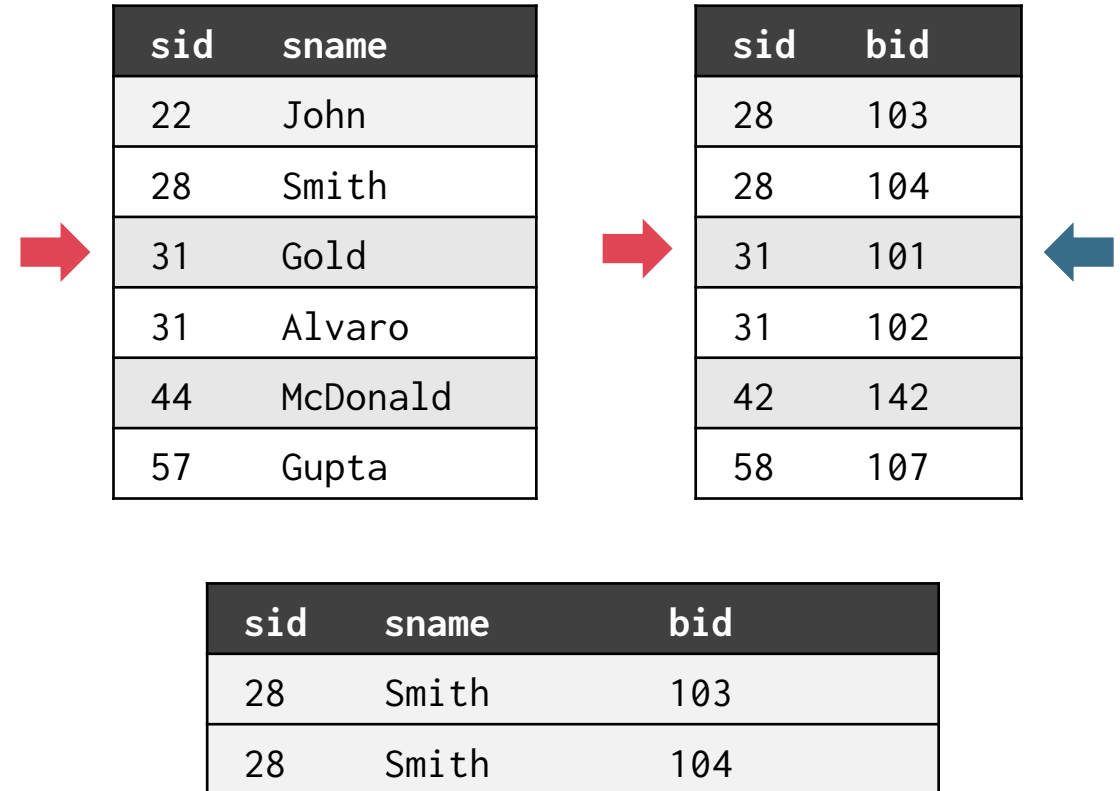


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

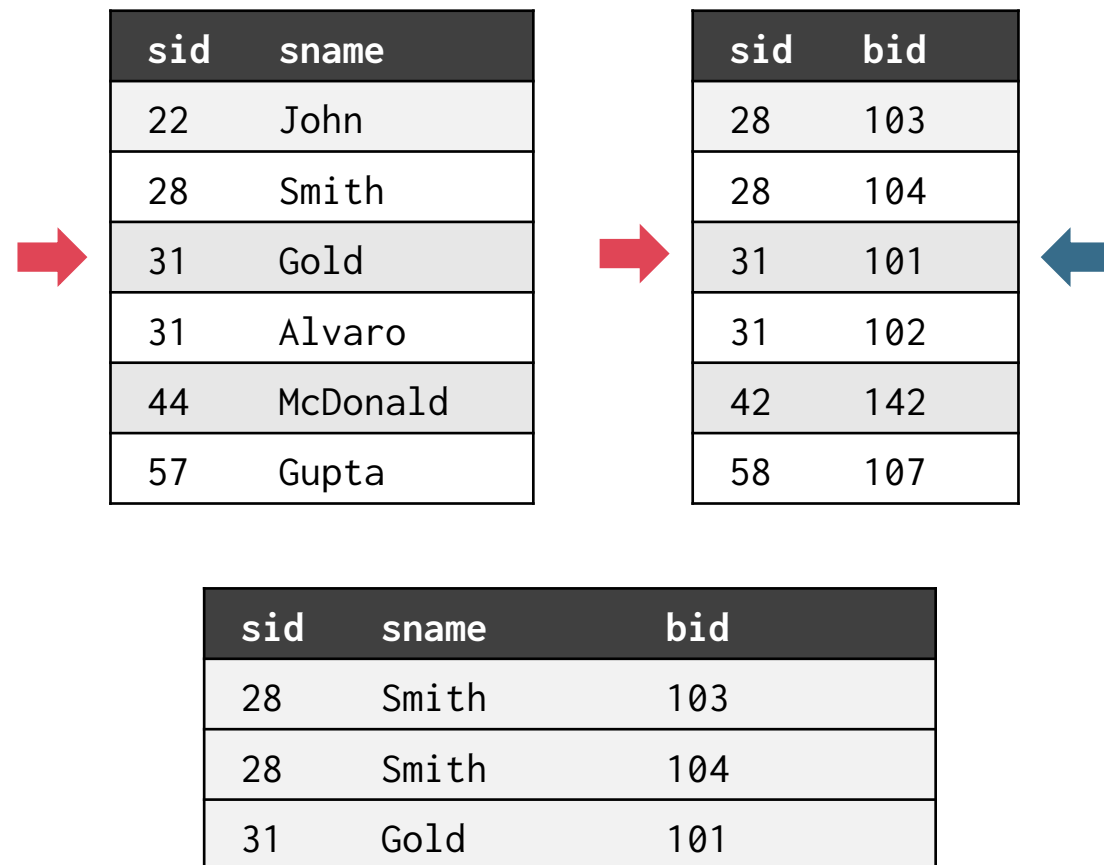


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

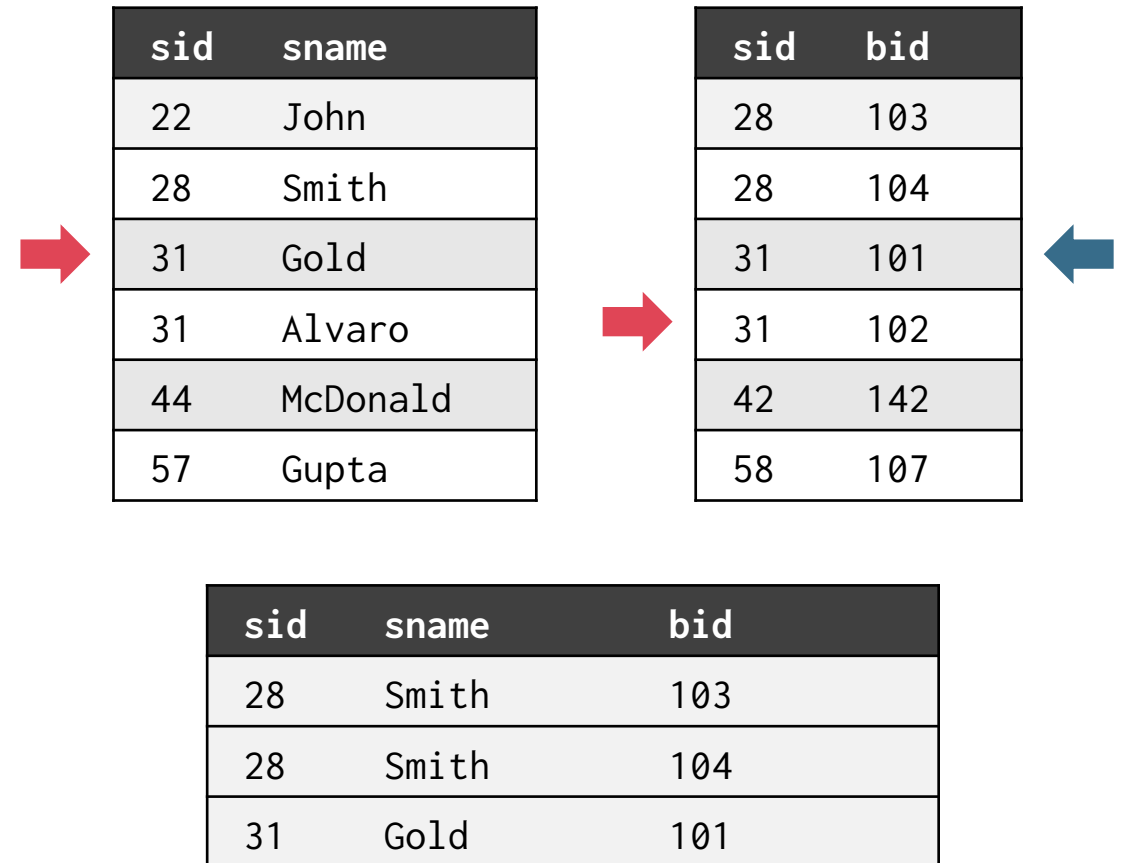


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

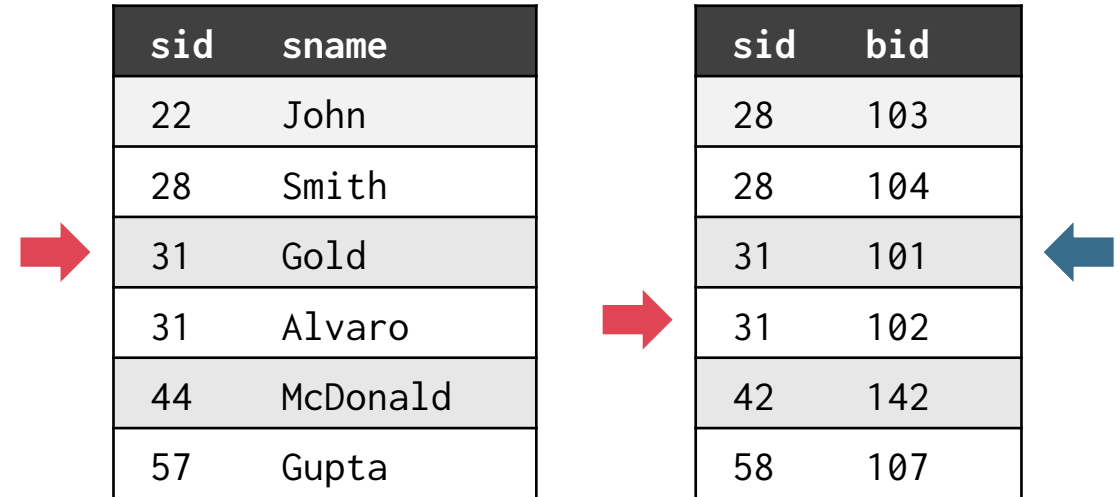


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



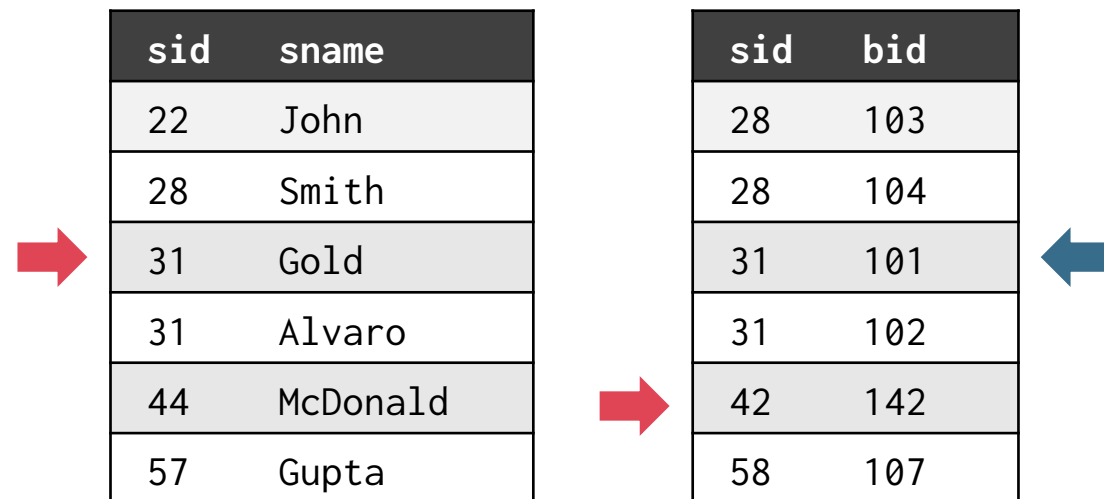
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



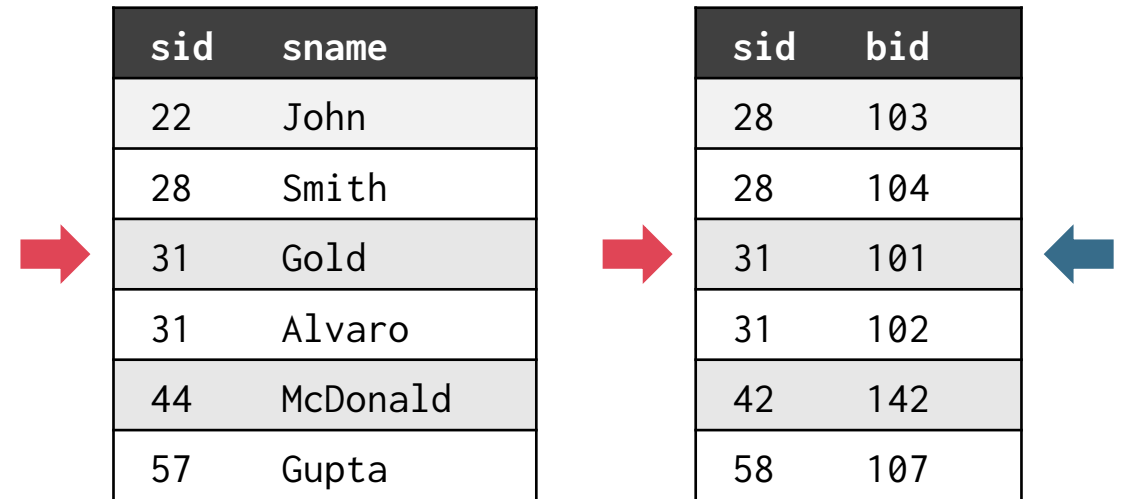
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



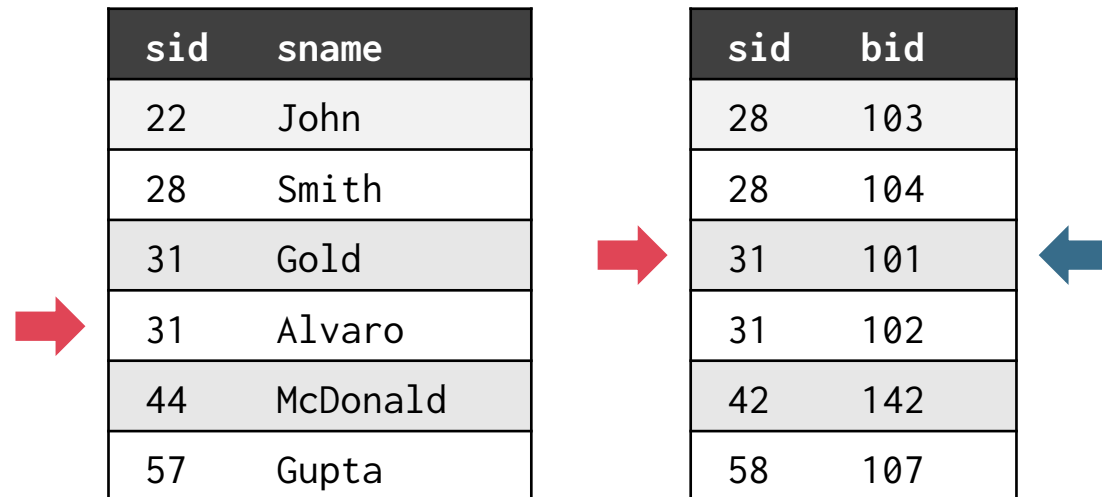
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

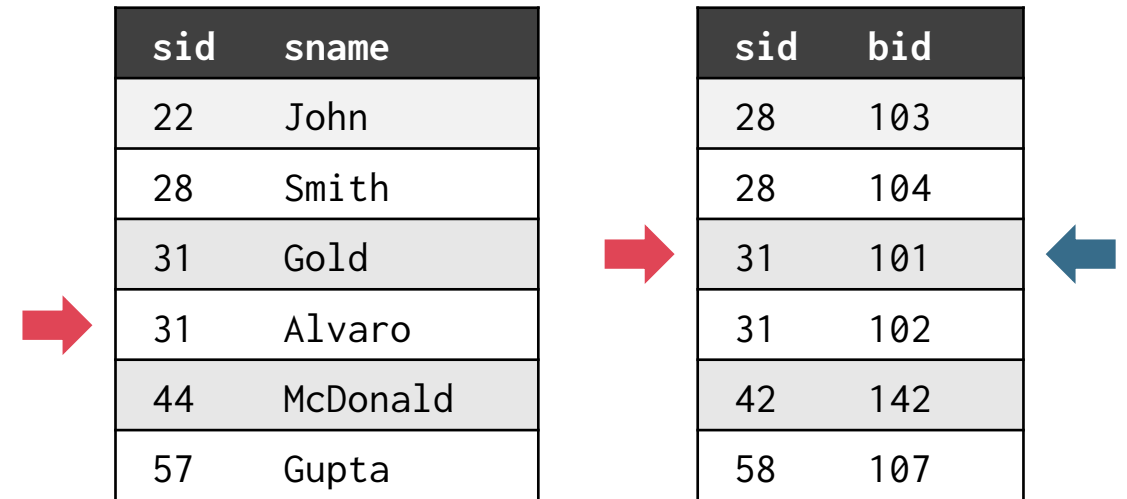
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

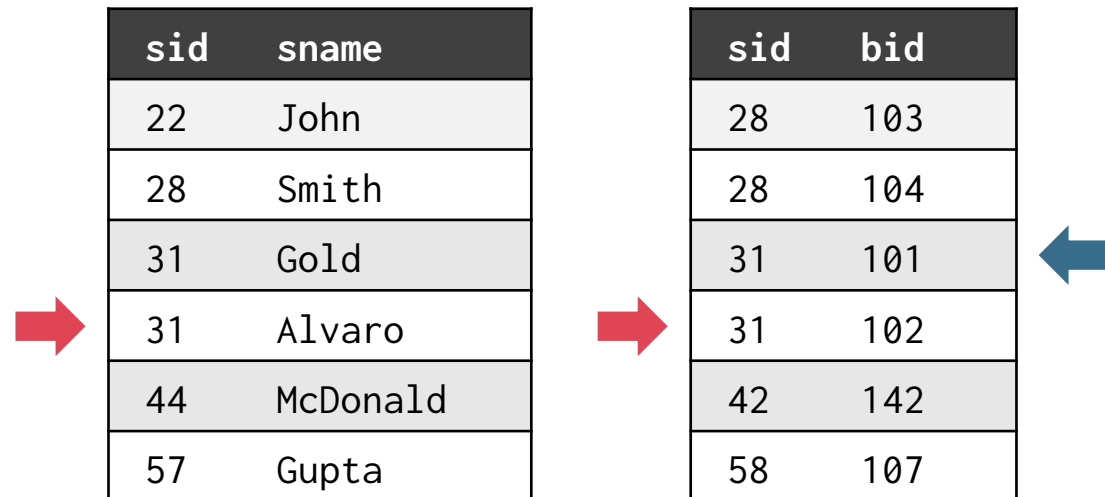


SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

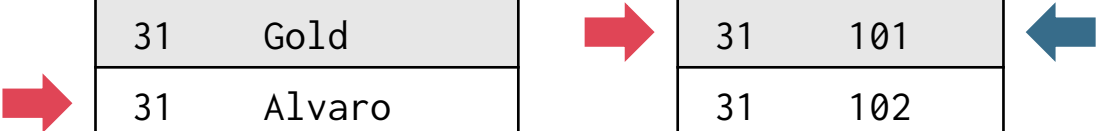
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

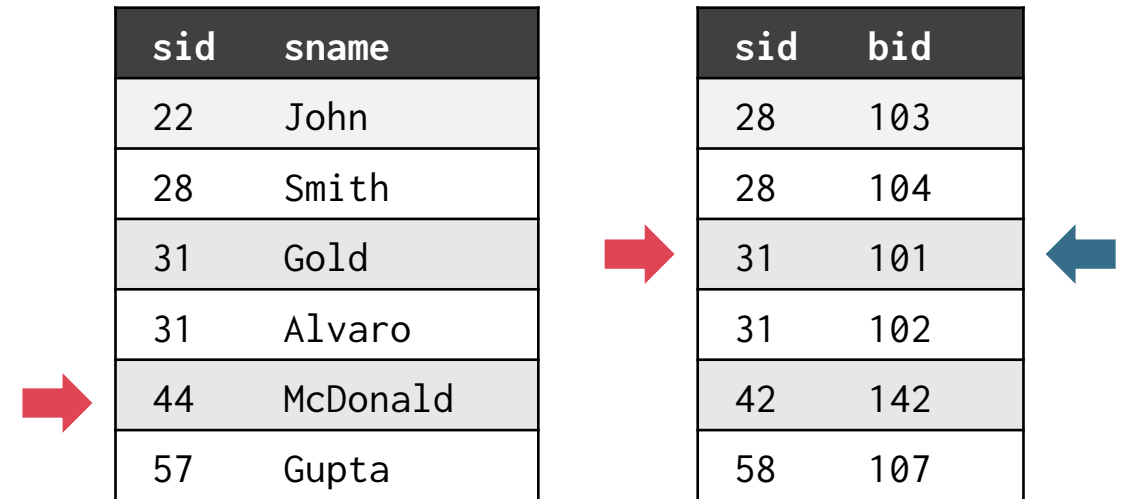
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```
while not done:
```

```
    while (r < s): advance r
```

```
    while (r > s): advance s
```

```
    mark s    // same start of "block"
```

```
    while (r == s):
```

```
        // outer loop over r
```

```
        while (r == s):
```

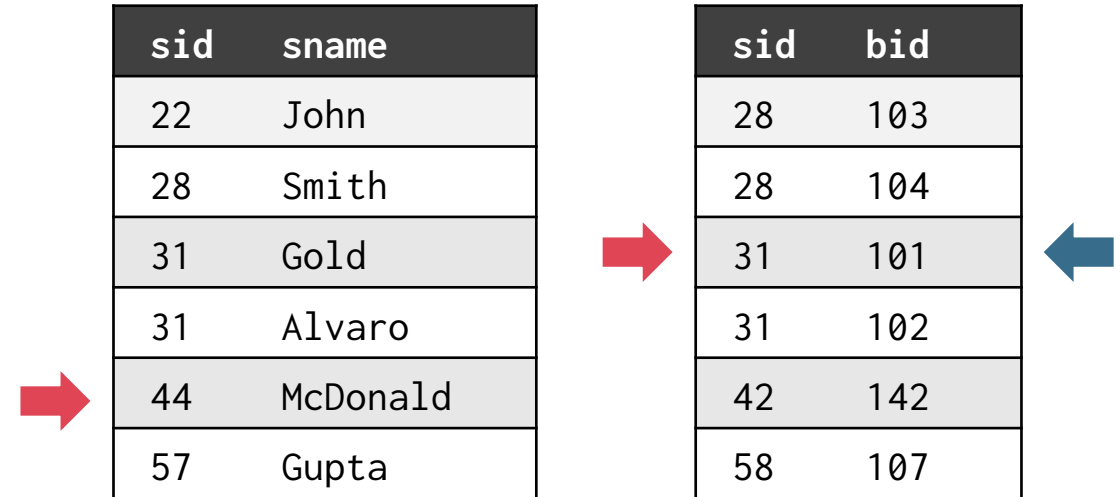
```
            // inner loop over s
```

```
            output r joined with s
```

```
            advance s
```

```
        reset s to mark
```

```
        advance r
```



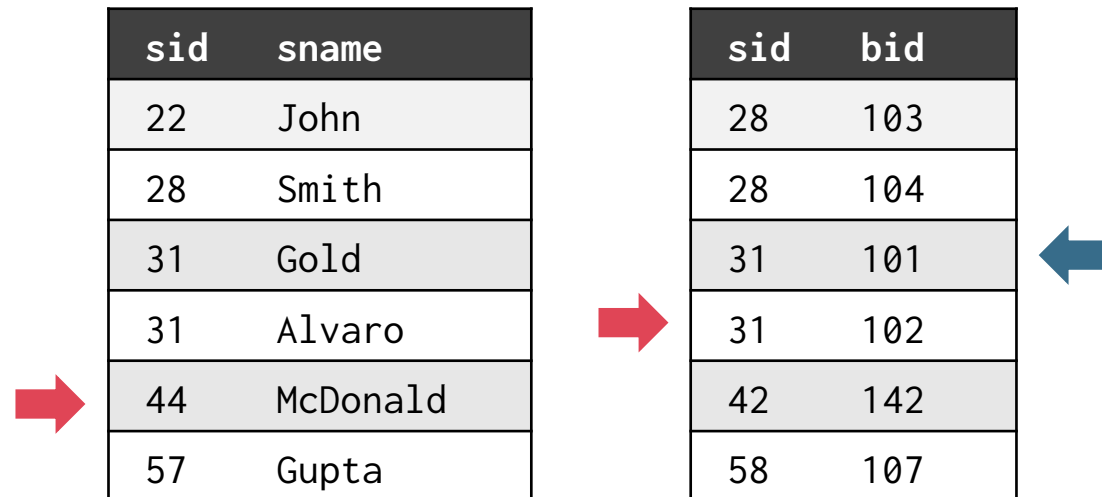
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```

sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

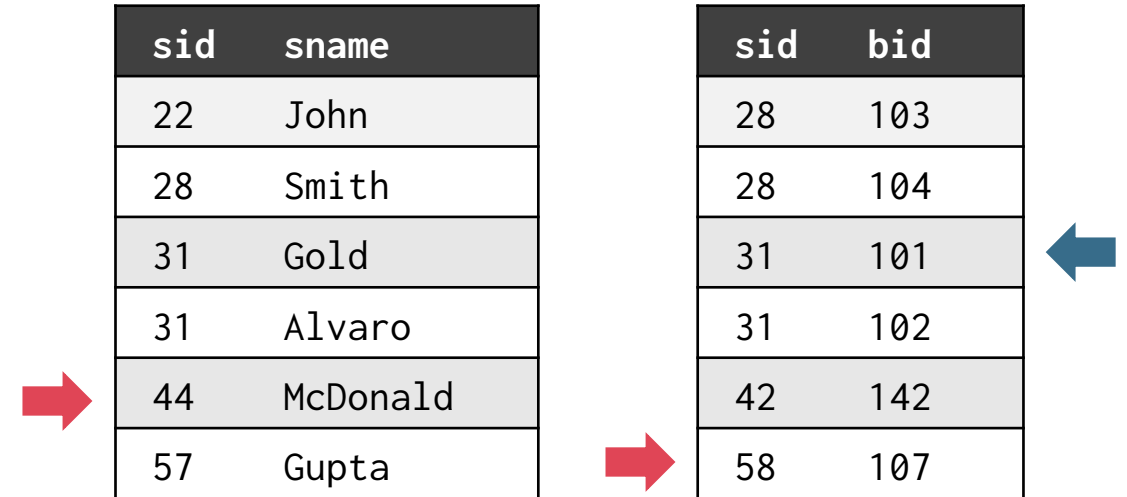
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s    // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

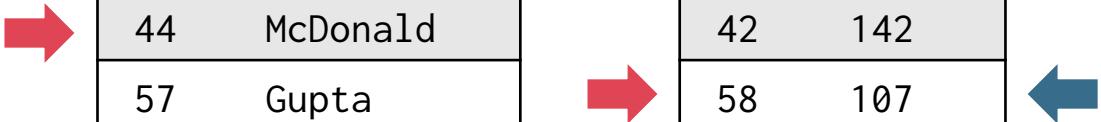
sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```

while not done:
    while (r < s): advance r
    while (r > s): advance s

    mark s // same start of "block"
    while (r == s):
        // outer loop over r
        while (r == s):
            // inner loop over s
            output r joined with s
            advance s
        reset s to mark
        advance r
  
```



sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```
while not done:
```

```
  while (r < s): advance r
```

```
  while (r > s): advance s
```

```
  mark s    // same start of "block"
```

```
  while (r == s):
```

```
    // outer loop over r
```

```
    while (r == s):
```

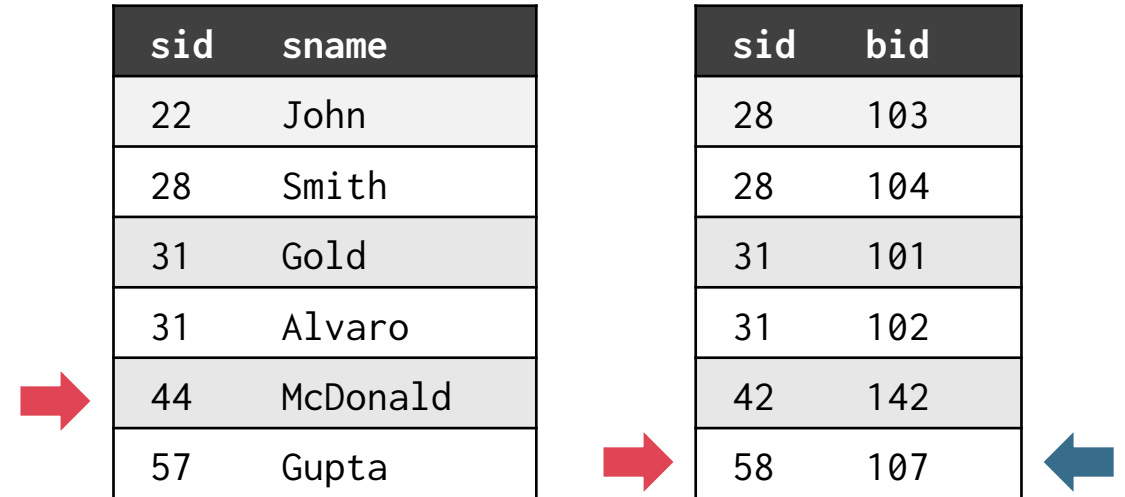
```
      // inner loop over s
```

```
      output r joined with s
```

```
      advance s
```

```
    reset s to mark
```

```
    advance r
```



sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

```
while not done:
```

```
  while (r < s): advance r
```

```
  while (r > s): advance s
```

```
  mark s    // same start of "block"
```

```
  while (r == s):
```

```
    // outer loop over r
```

```
    while (r == s):
```

```
      // inner loop over s
```

```
      output r joined with s
```

```
      advance s
```

```
    reset s to mark
```

```
    advance r
```

sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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



sid	sname	bid
28	Smith	103
28	Smith	104
31	Gold	101
31	Gold	102
31	Alvaro	101
31	Alvaro	102

SORT MERGE JOIN (SMJ)

Total cost:

Cost of sorting R

Cost of sorting S

Cost of merging: $M + N$

Only one pass (if we assume there aren't a lot of duplicates)

SORT MERGE JOIN (SMJ)

We can be sometimes smarter about SMJ

Observation:

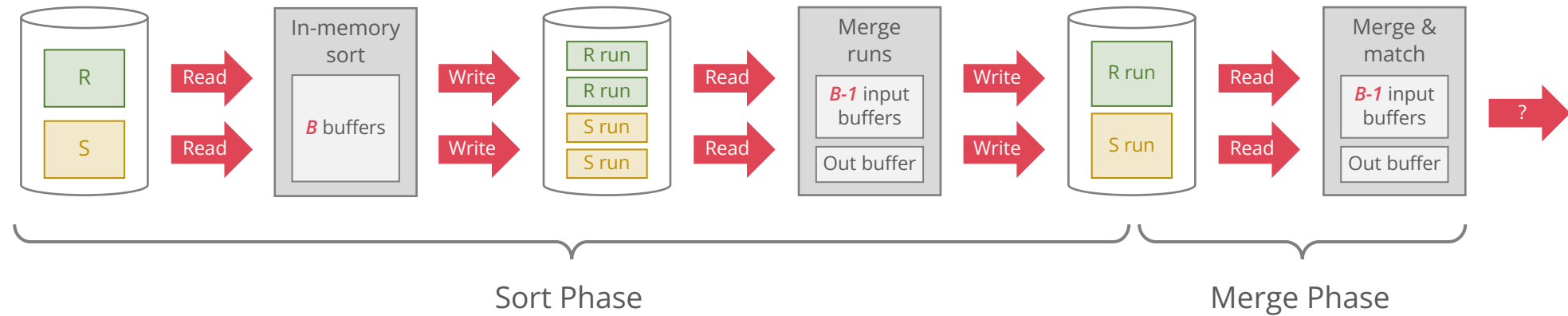
We make **one pass** through each sorted relation (assuming no duplicate values in ***R***)

⇒ We do **not need** the sorted relations to be materialized!

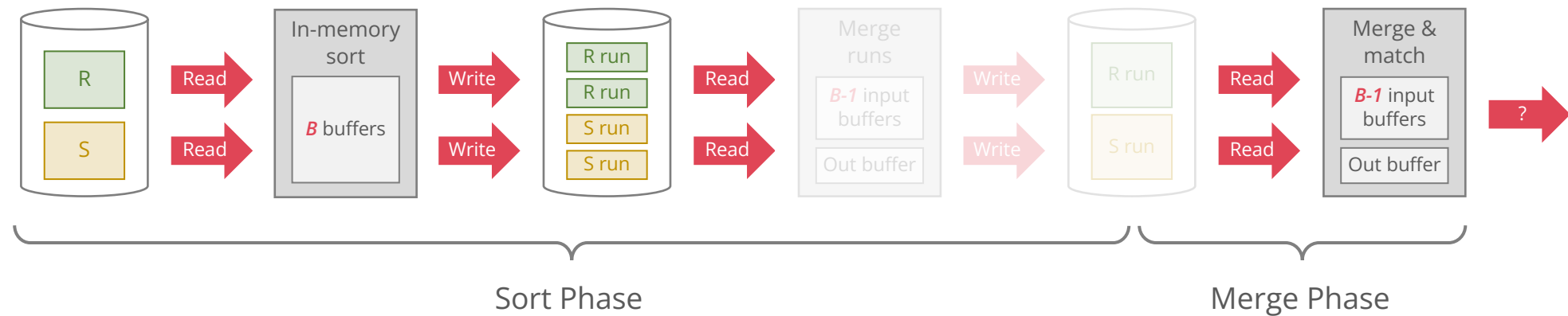
Optimisation:

In the final merge pass of sorting both relations, instead of writing the sorted relations to disk, we can stream them into the second part of SMJ!

SORT MERGE JOIN (SMJ)



SORT MERGE JOIN (SMJ)



We have to be able to fit the input buffers of the last merge pass of sorting **R** and sorting **S** in memory, as well as have one output buffer for joined tuples

Need: $(\# \text{ runs in last merge pass for } R) + (\# \text{ runs in last merge pass for } S) \leq B - 1$

Reduces I/O cost by $2 \cdot (M + N)!$

GRACE HASH JOIN

Similar idea as SMJ, but let's build some hash tables instead

Two passes: **partition** the data, then **build** a hash table and **probe** it

Partition ***R*** and ***S*** into ***B - 1*** partitions (like in external hashing) using same hash function

All the tuples in ***R*** matching a tuple in ***S*** must be in the same partition

⇒ We can consider each partition independently

GRACE HASH JOIN

Similar idea as SMJ, but let's build some hash tables instead

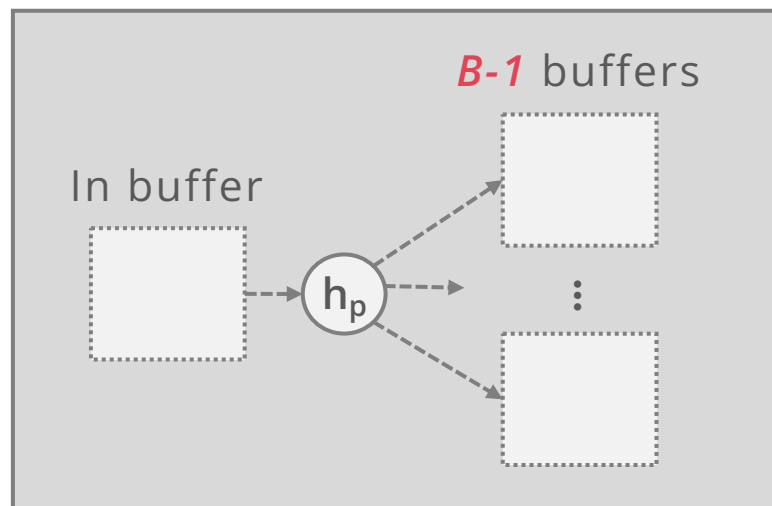
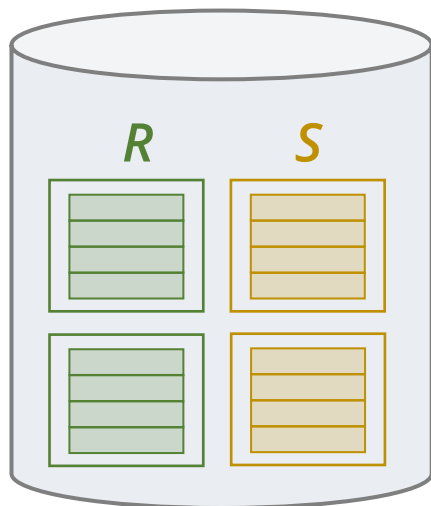
Two passes: **partition** the data, then **build** a hash table and **probe** it

Then, build an in-memory hash table for a partition of **R**

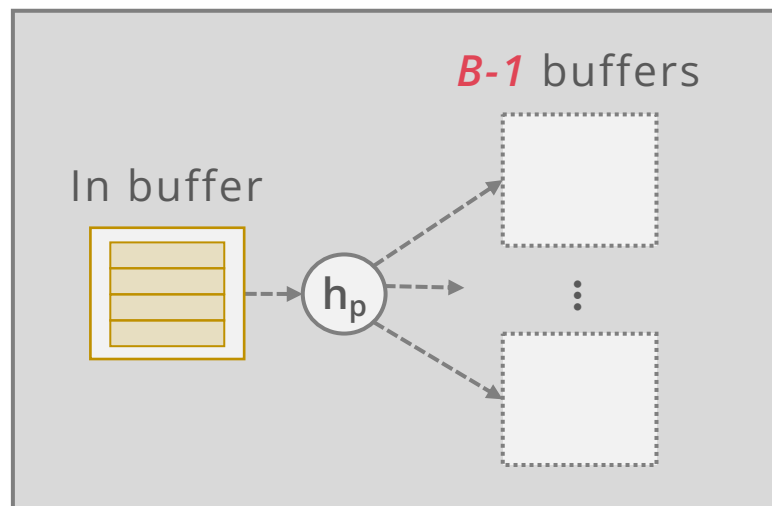
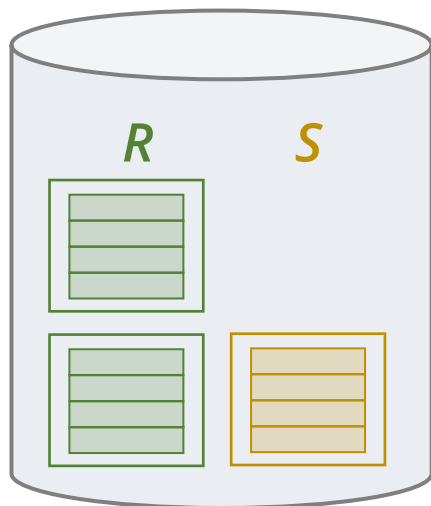
We can use this in-memory hash table to find all the tuples in **R** that match a tuple in **S**

Stream in tuples of **S**, probe the hash table, output matching tuples

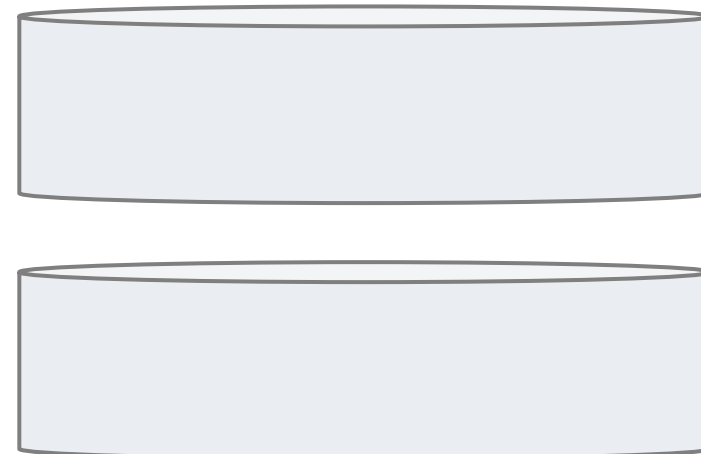
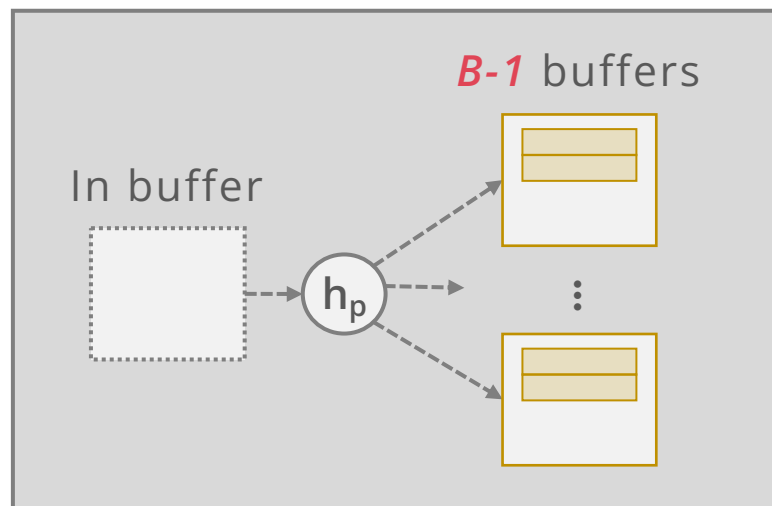
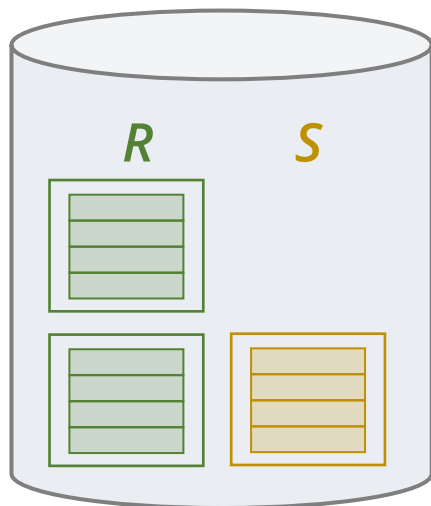
GRACE HASH JOIN: PARTITION



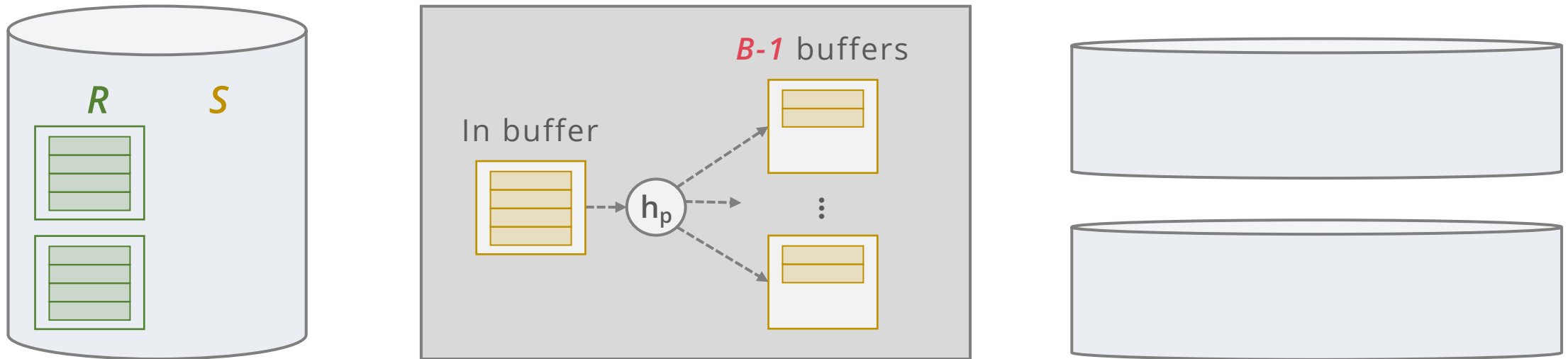
GRACE HASH JOIN: PARTITION



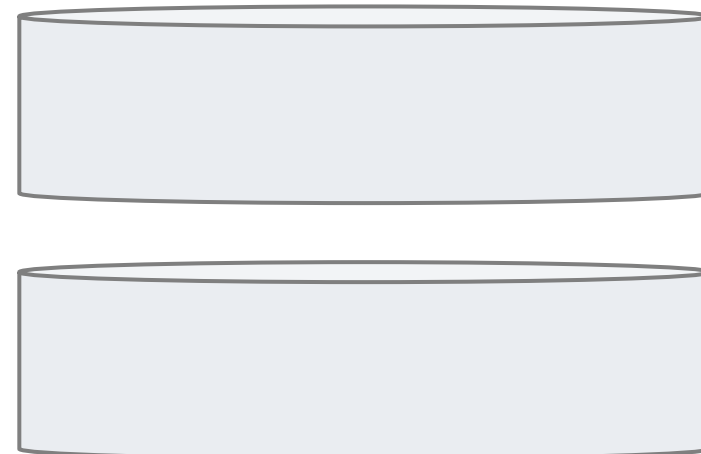
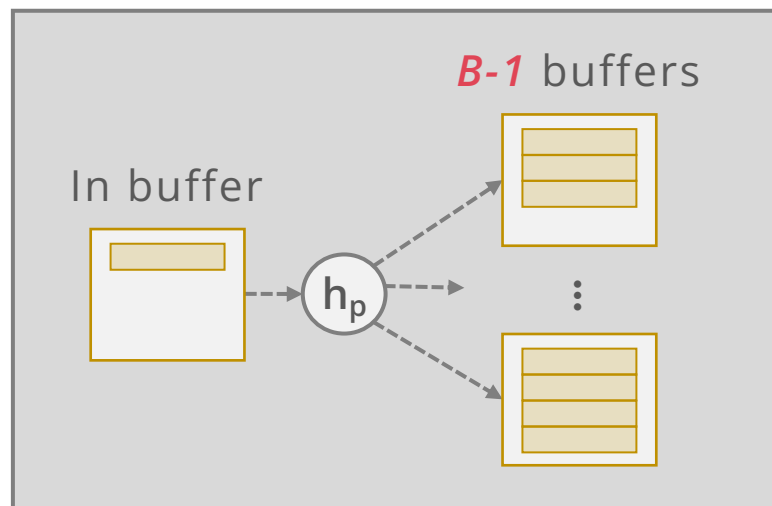
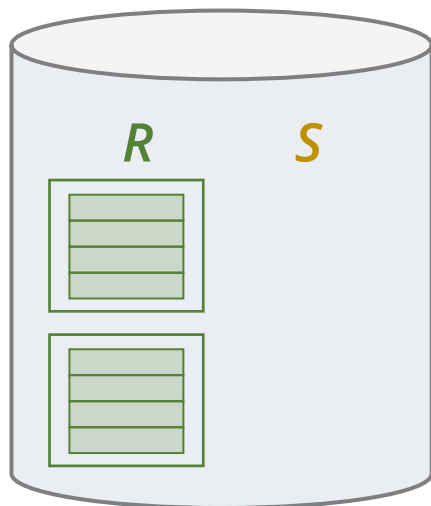
GRACE HASH JOIN: PARTITION



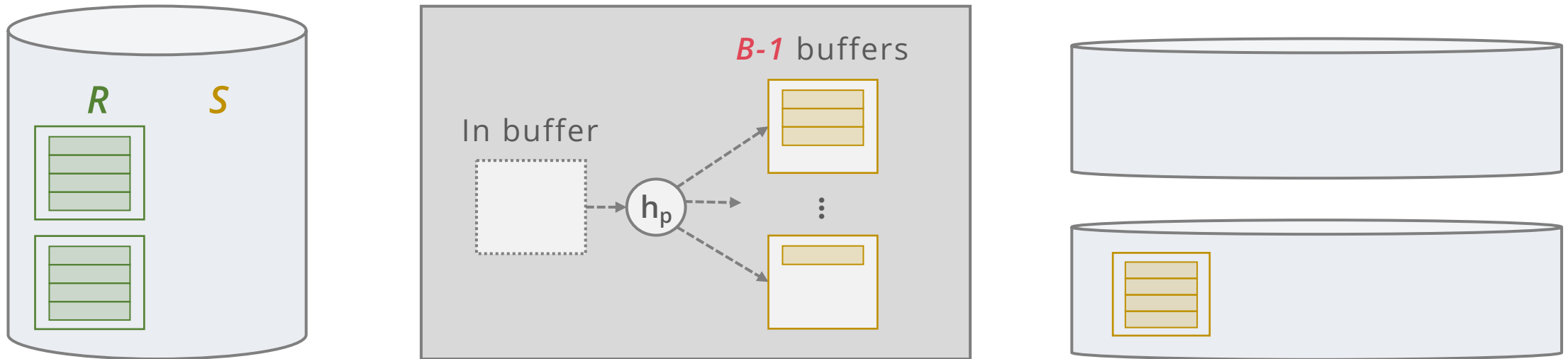
GRACE HASH JOIN: PARTITION



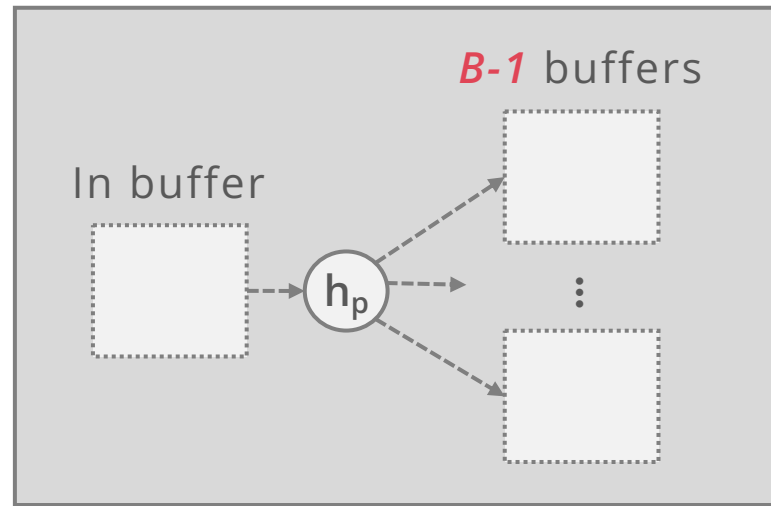
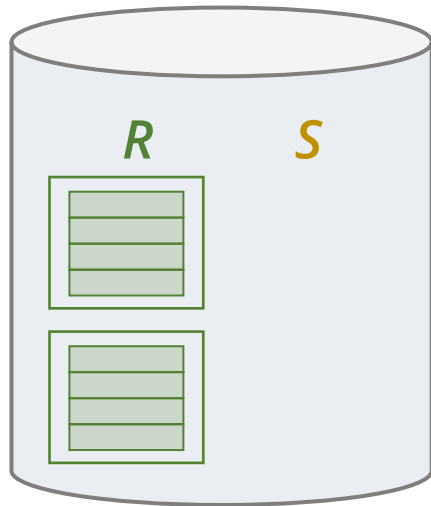
GRACE HASH JOIN: PARTITION



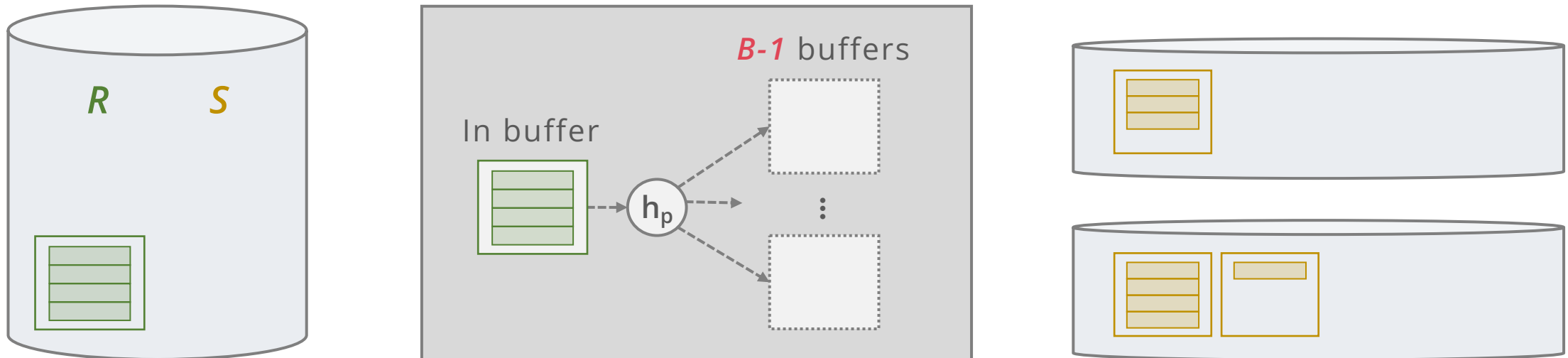
GRACE HASH JOIN: PARTITION



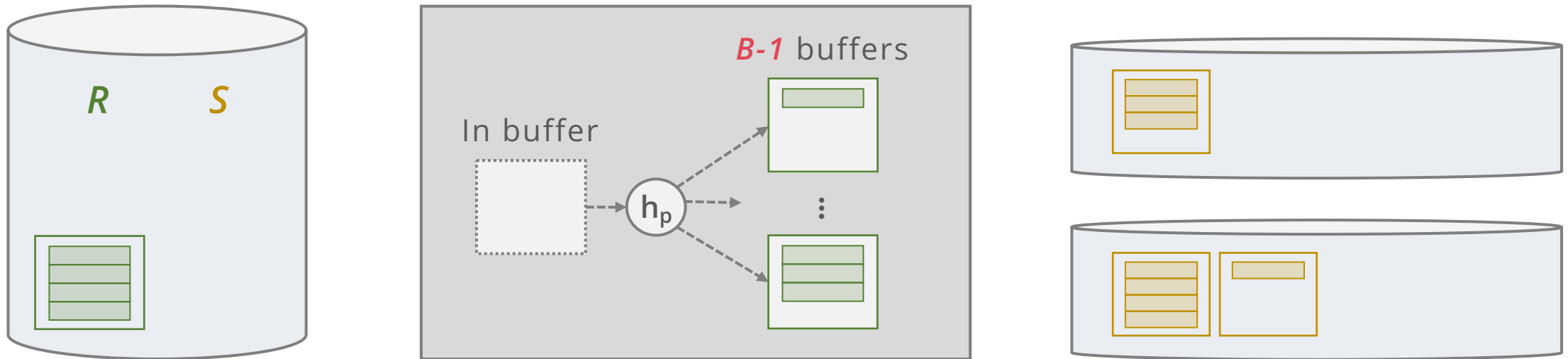
GRACE HASH JOIN: PARTITION



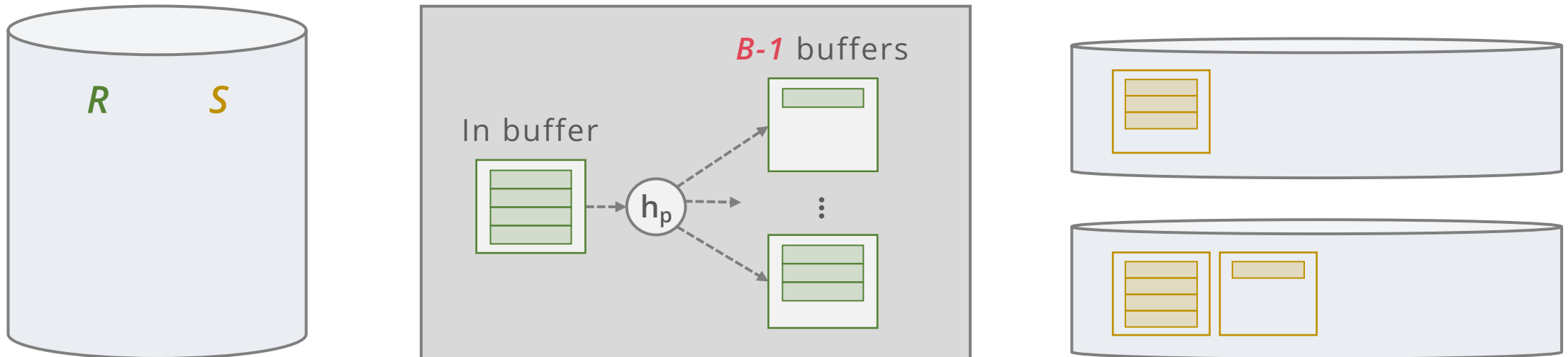
GRACE HASH JOIN: PARTITION



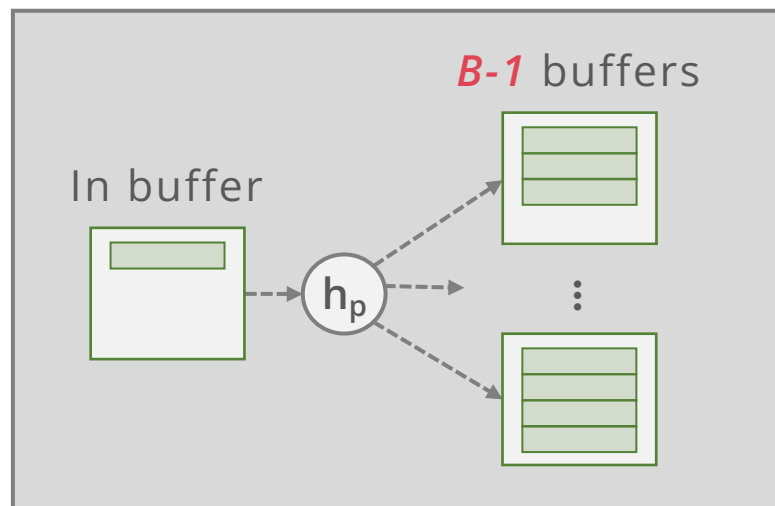
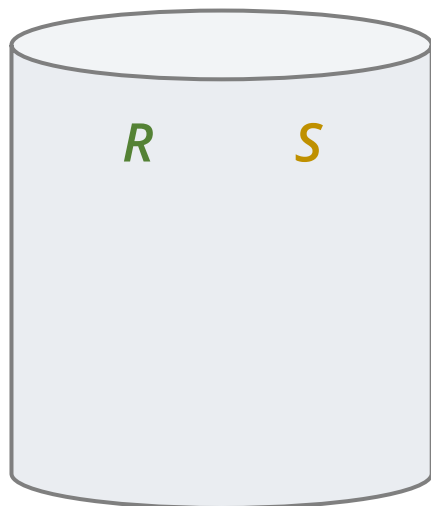
GRACE HASH JOIN: PARTITION



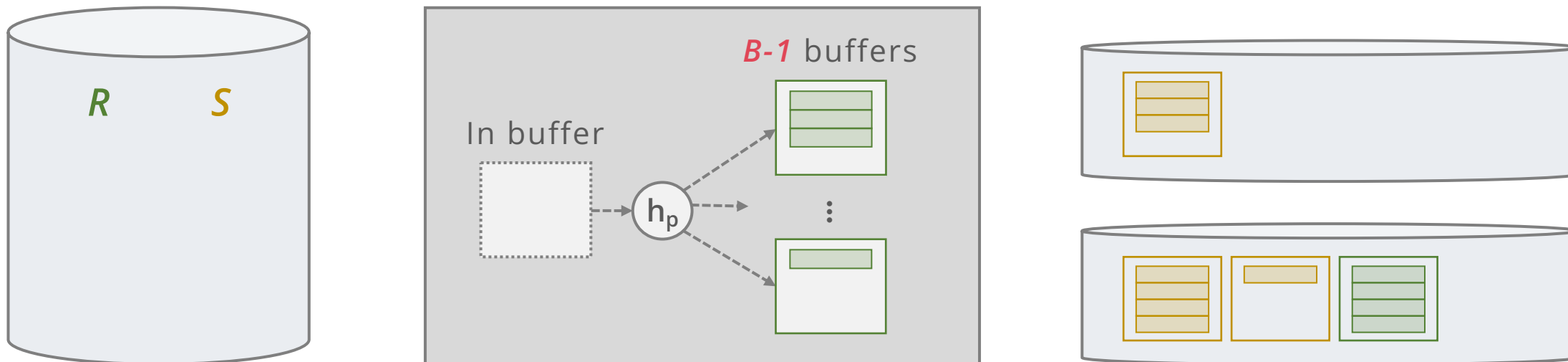
GRACE HASH JOIN: PARTITION



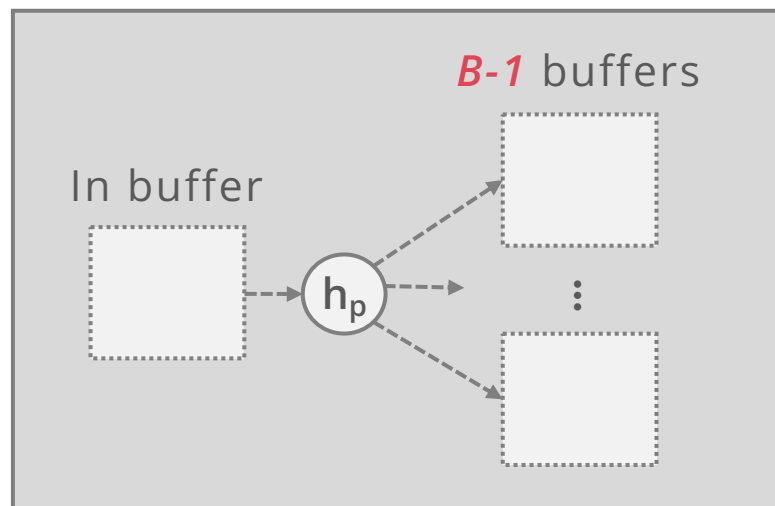
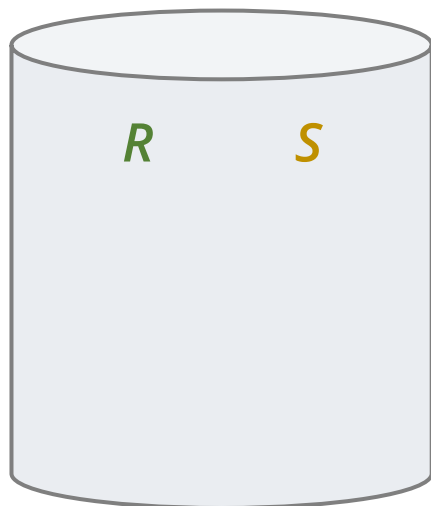
GRACE HASH JOIN: PARTITION



GRACE HASH JOIN: PARTITION



GRACE HASH JOIN: PARTITION



GRACE HASH JOIN

We need partitions of R (but not S !) to fit in $B - 2$ pages

- 1 page reserved for streaming S partition

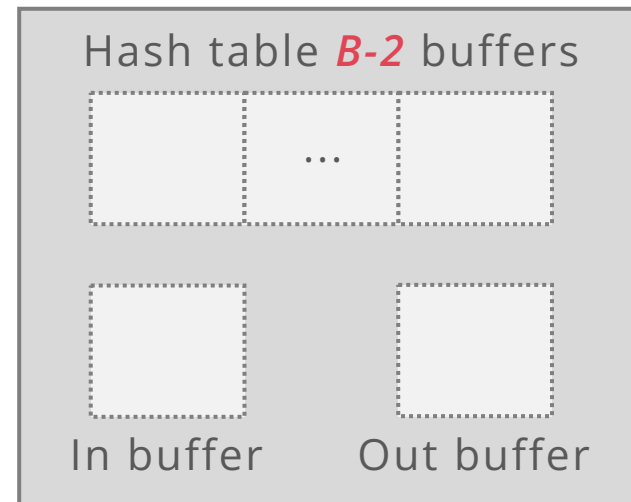
- 1 page reserved for streaming output

What if partitions of R are too big?

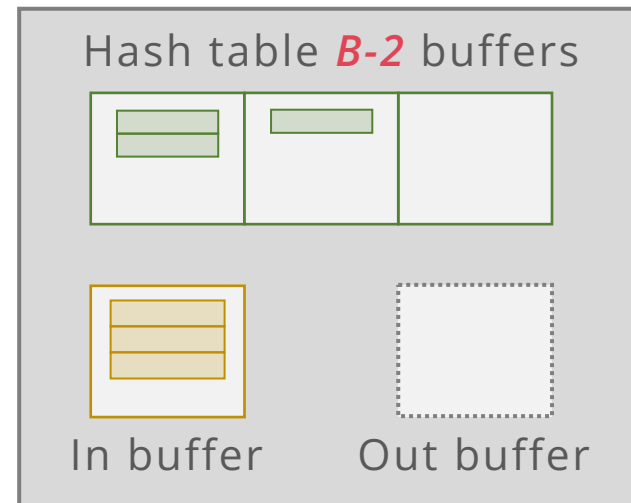
- If S is smaller, do $S \bowtie_{\theta} R$ instead

- Recursively partition! Make sure that for any partition of R you recursively partition, the matching S partition is also recursively partitioned!

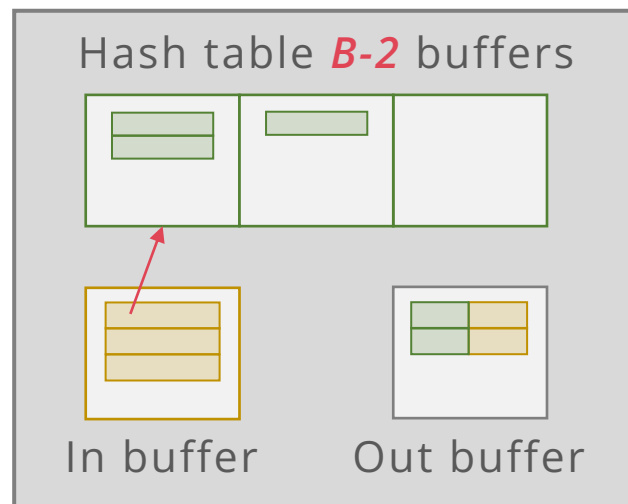
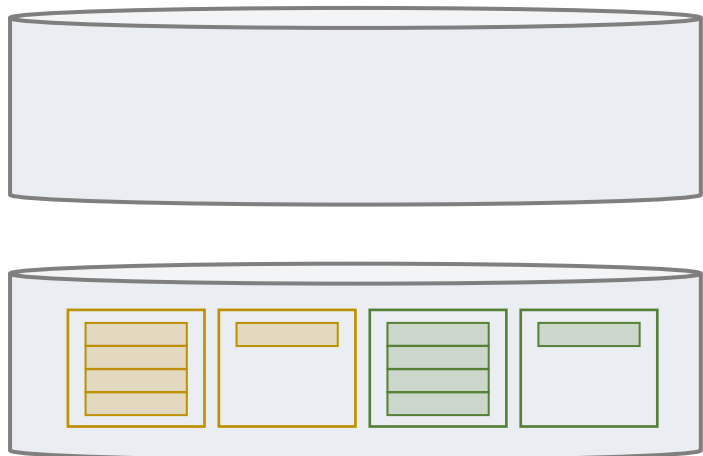
GRACE HASH JOIN: BUILD & PROBE



GRACE HASH JOIN: BUILD & PROBE

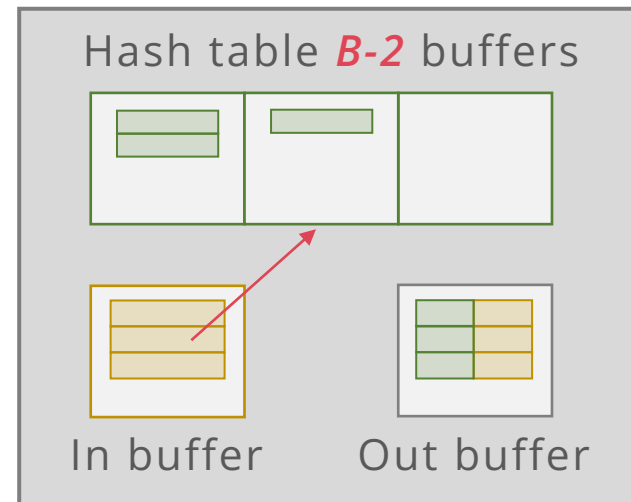


GRACE HASH JOIN: BUILD & **P**ROBE



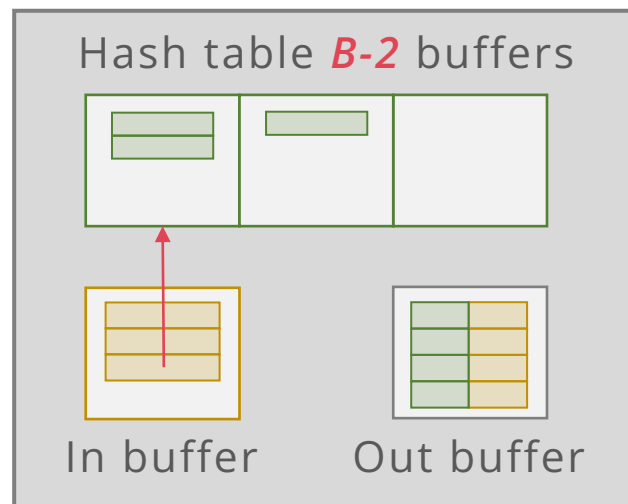
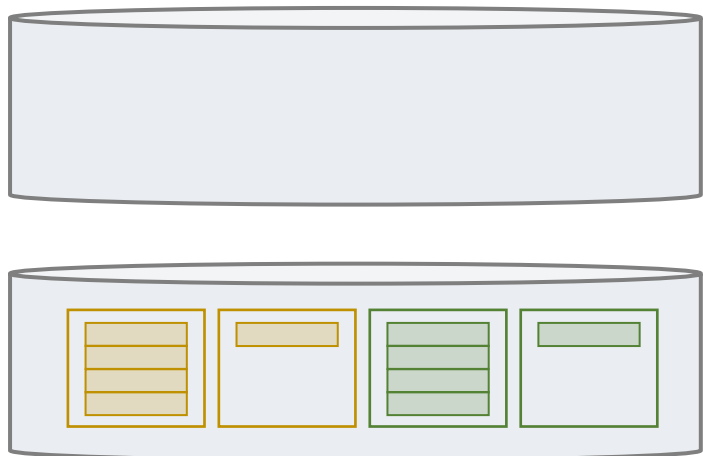
2 matches

GRACE HASH JOIN: BUILD & **P**ROBE



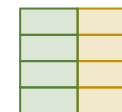
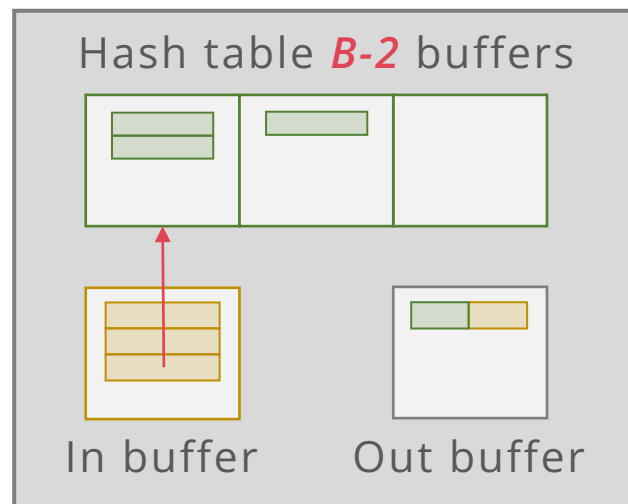
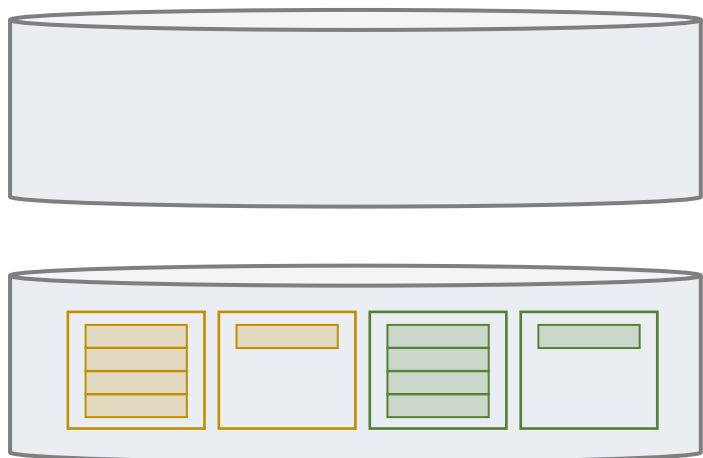
1 match

GRACE HASH JOIN: BUILD & **P**ROBE



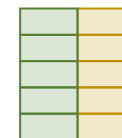
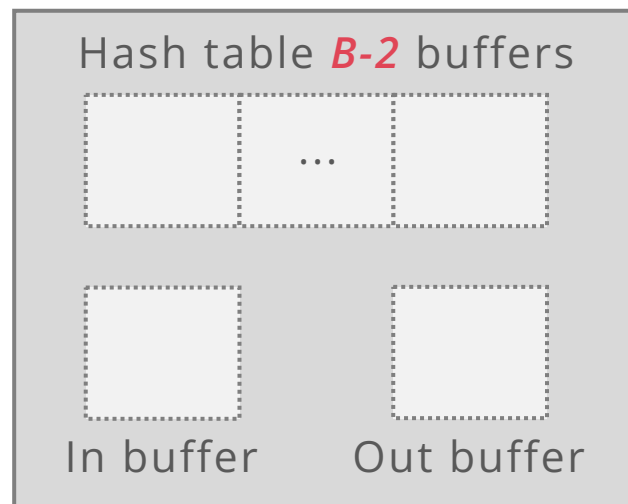
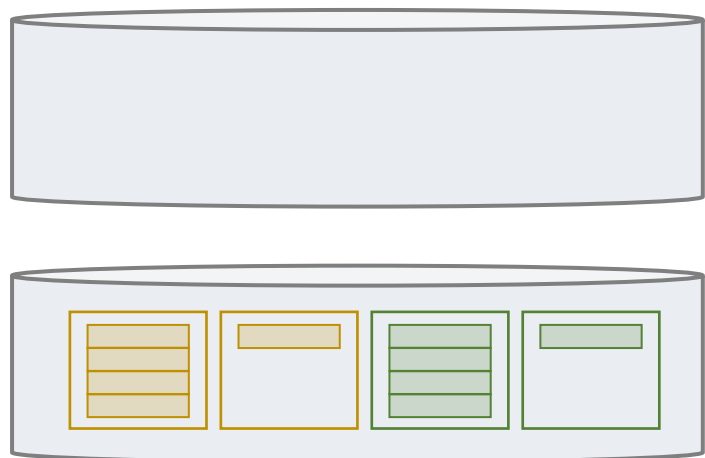
2 matches

GRACE HASH JOIN: BUILD & **P**ROBE



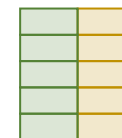
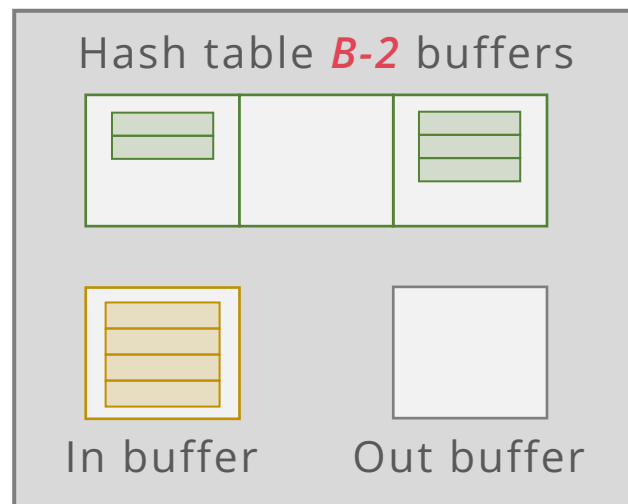
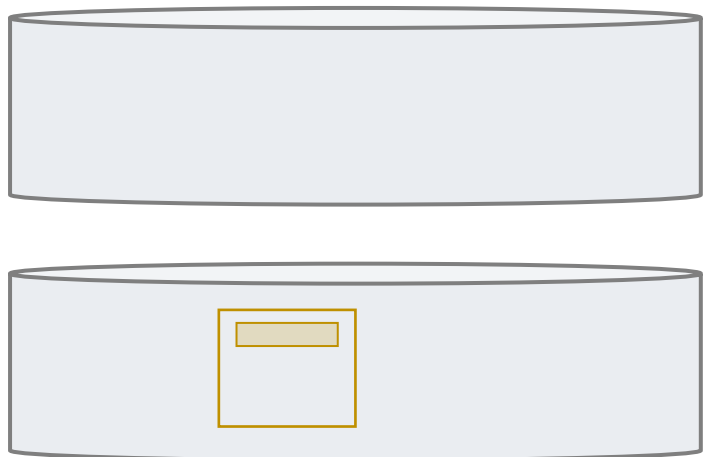
2 matches

GRACE HASH JOIN: BUILD & PROBE

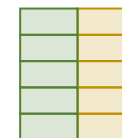
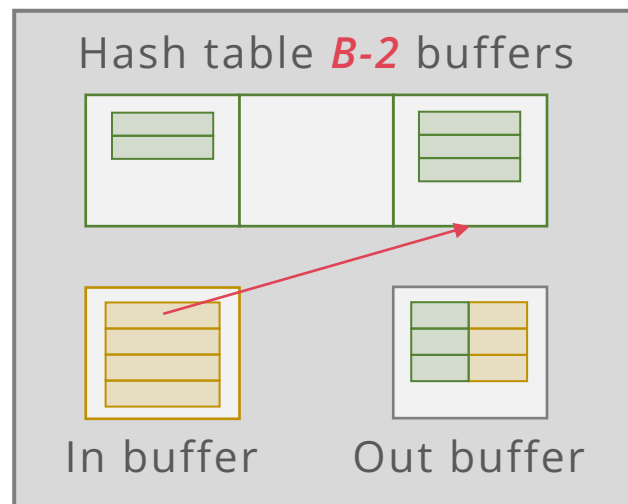
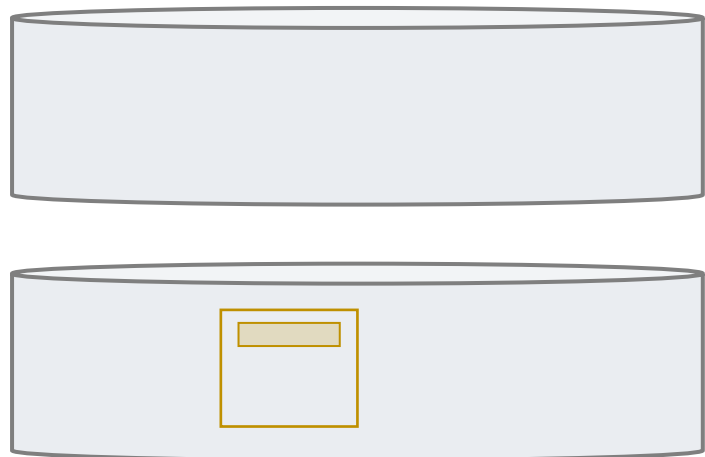


Flushing output buffer not strictly necessary, can reuse for next pair of partitions

GRACE HASH JOIN: BUILD & PROBE

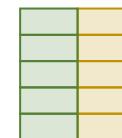
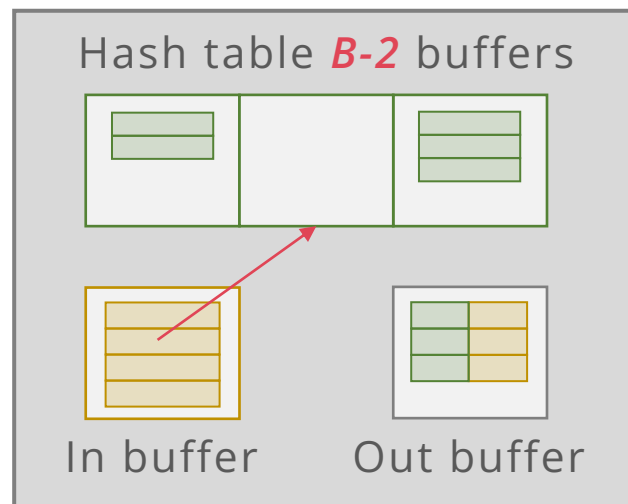
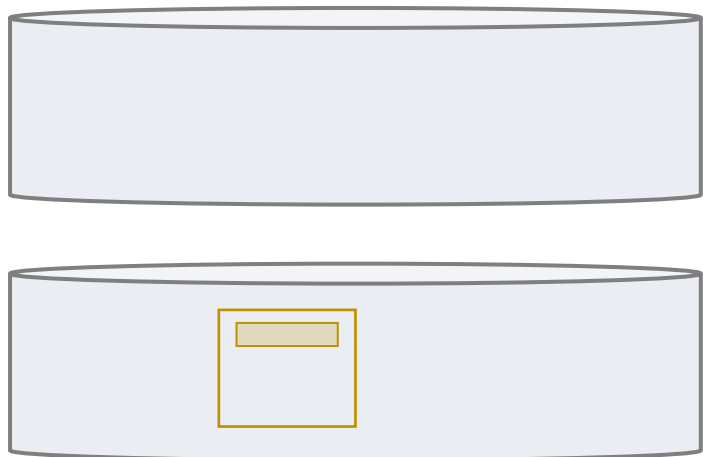


GRACE HASH JOIN: BUILD & **P**ROBE



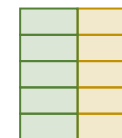
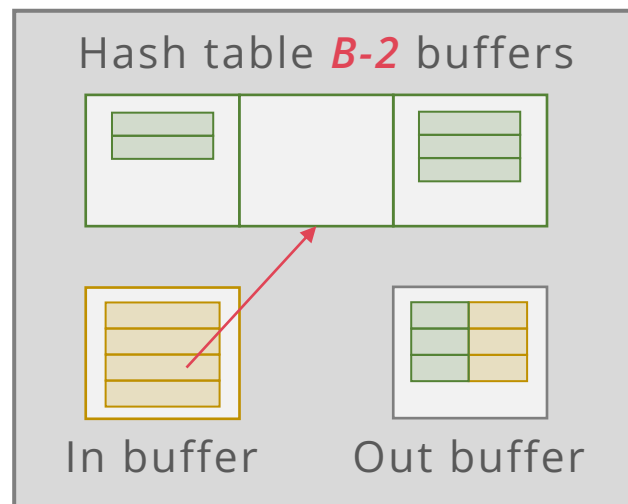
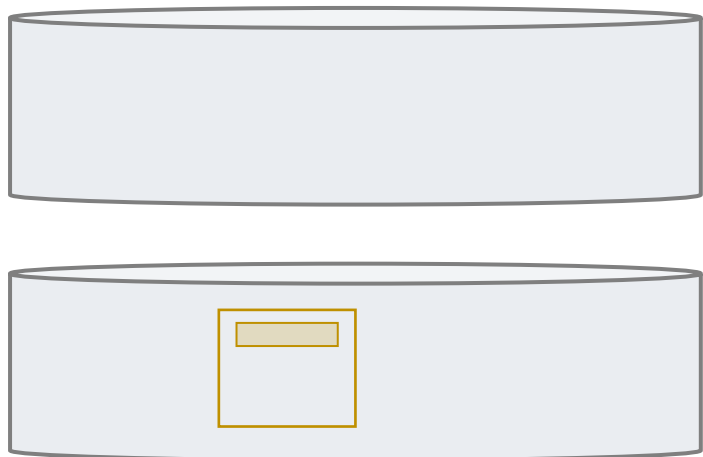
3 matches

GRACE HASH JOIN: BUILD & **P**ROBE



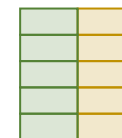
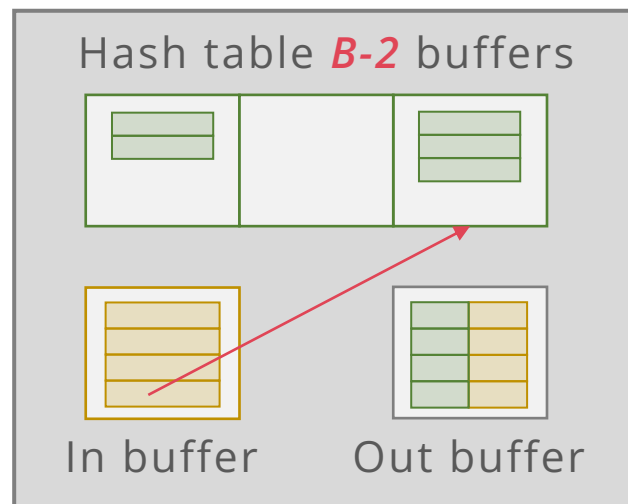
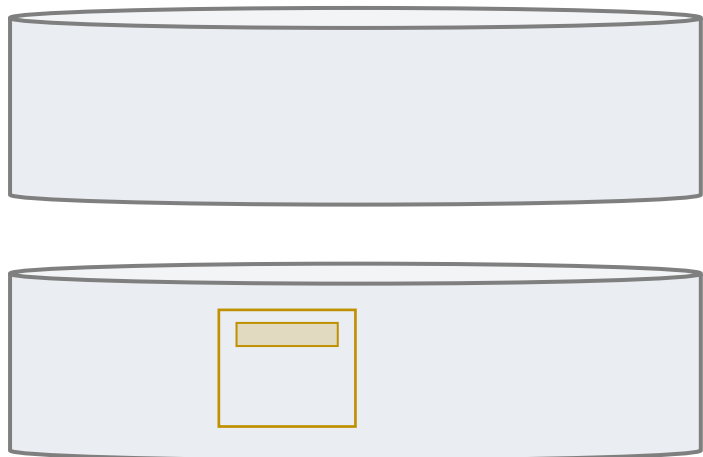
No match

GRACE HASH JOIN: BUILD & **P**ROBE



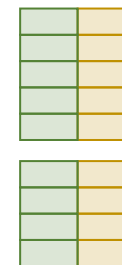
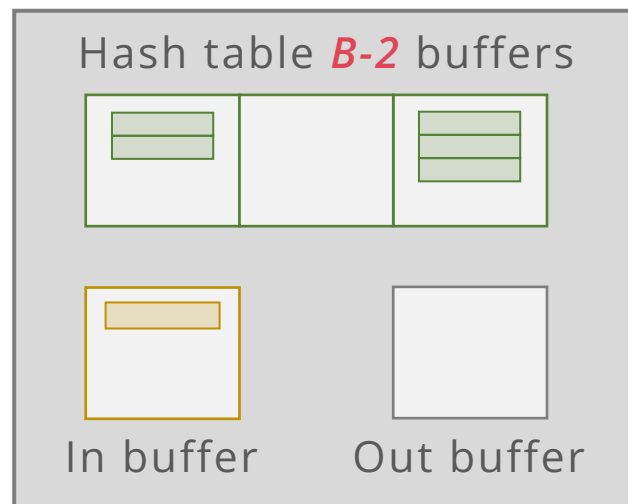
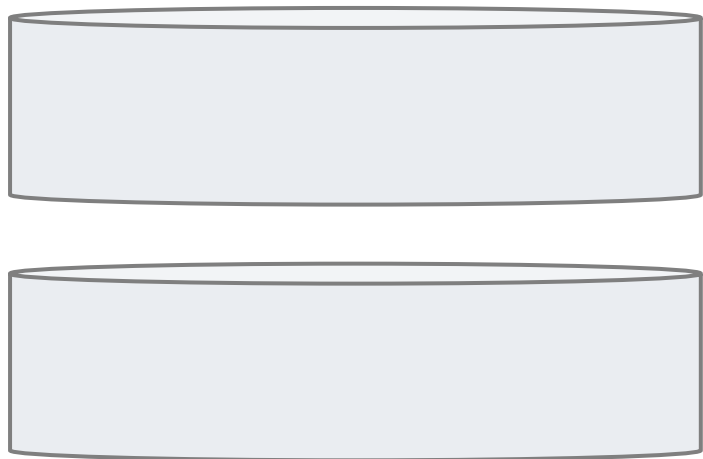
No match

GRACE HASH JOIN: BUILD & **P**ROBE

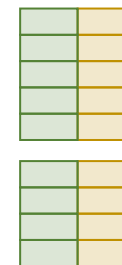
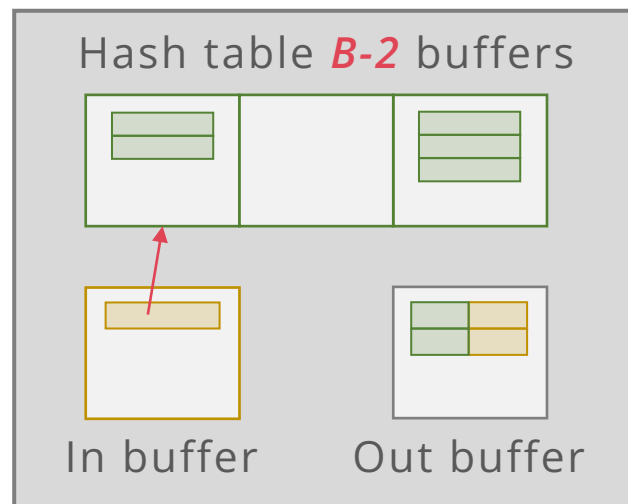
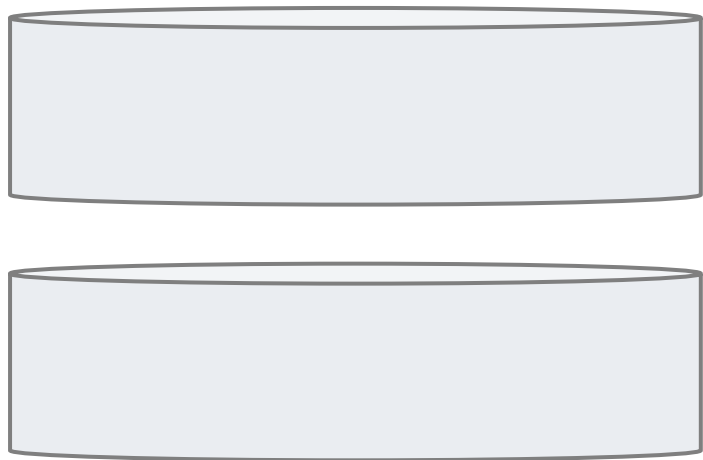


1 match

GRACE HASH JOIN: BUILD & **P**ROBE



GRACE HASH JOIN: BUILD & **P**ROBE



2 matches

GRACE HASH JOIN: BUILD & PROBE

