THE UNIVERSITY *of* EDINBURGH

Advanced Database Systems

Spring 2026

Q&A Session 2

If you require this document in an alternative format, such as large print or a coloured background, please contact milos.nikolic@ed.ac.uk

About This Session

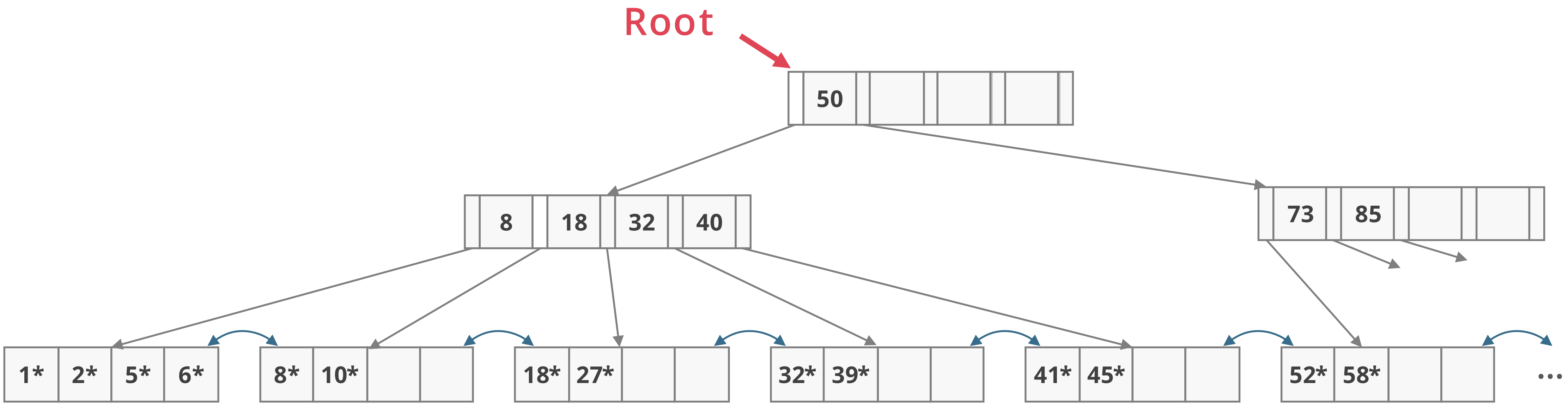
- Practice Worksheet 2 is now available on Learn
  - We will work through some questions during this session
- This slide deck includes animations of discussed algorithms
  - External sorting
  - Join algorithms
- **Main Purpose:** To reinforce and strengthen your understanding

Question 1

B+ Tree: Search for 39

Find key = 39

- Find split on each node
- Follow pointer to next node

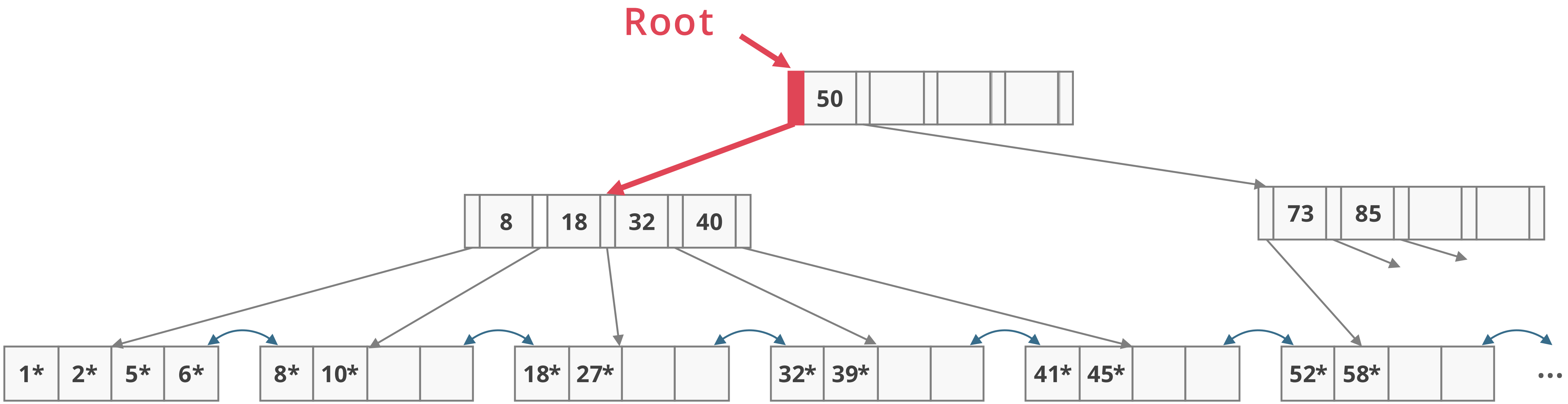


B+ Tree: Search for 39

Find key = 39

- Find split on each node
- Follow pointer to next node

Use binary search on each page

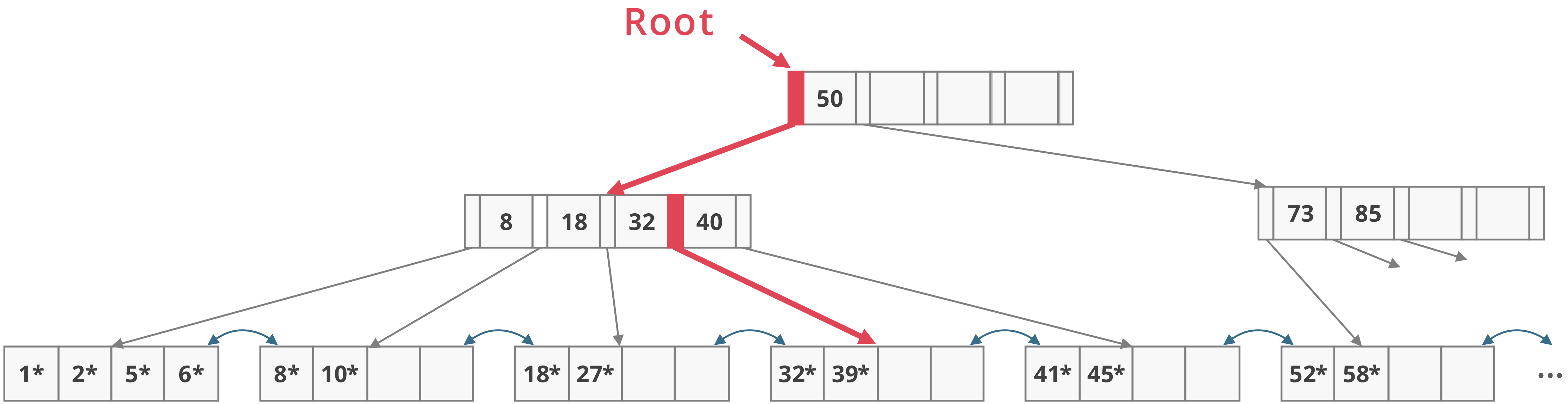


B+ Tree: Search for 39

Find key = 39

- Find split on each node
- Follow pointer to next node

Use binary search on each page

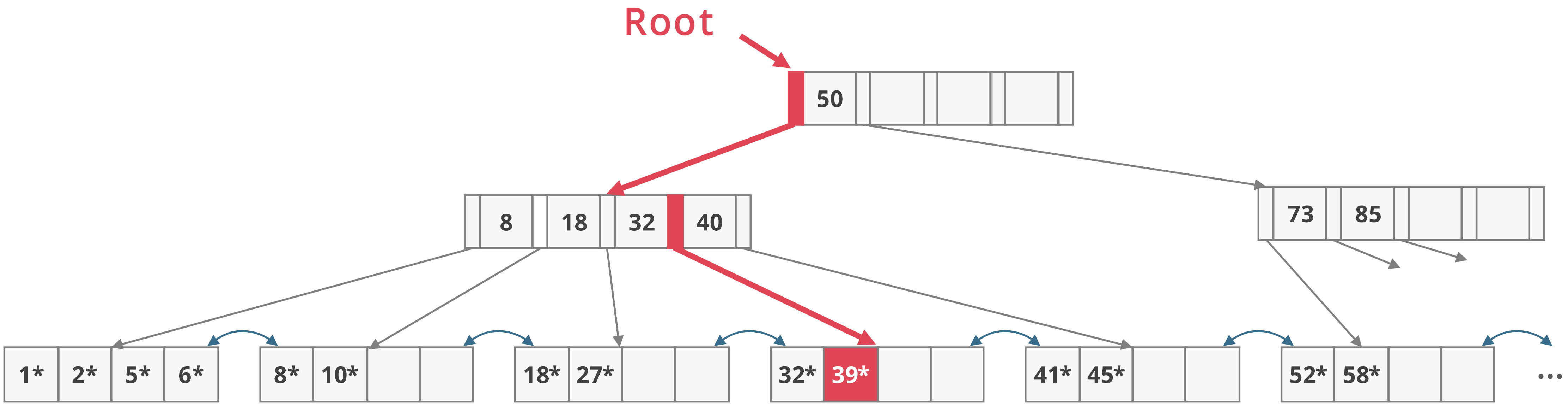


B+ Tree: Search for 39

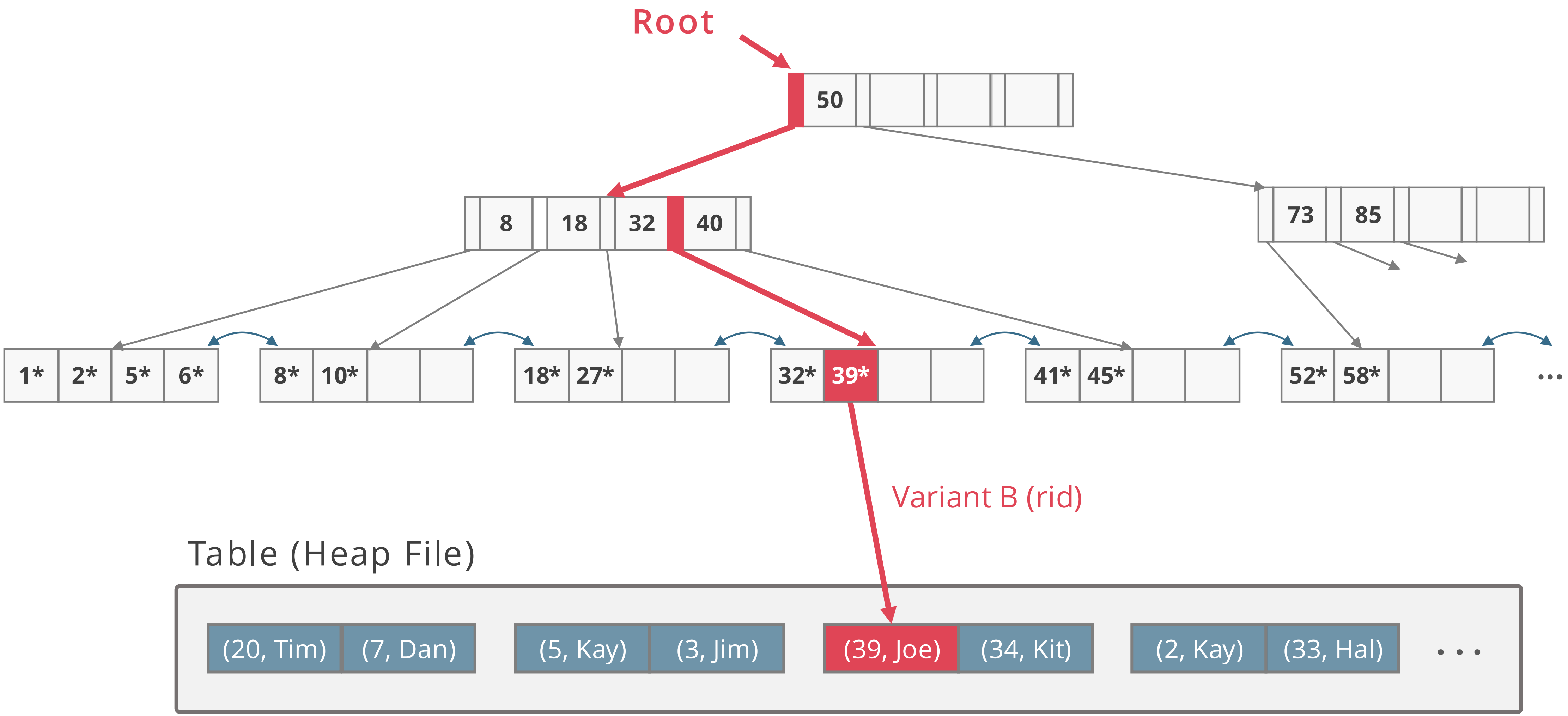
Find key = 39

- Find split on each node
- Follow pointer to next node

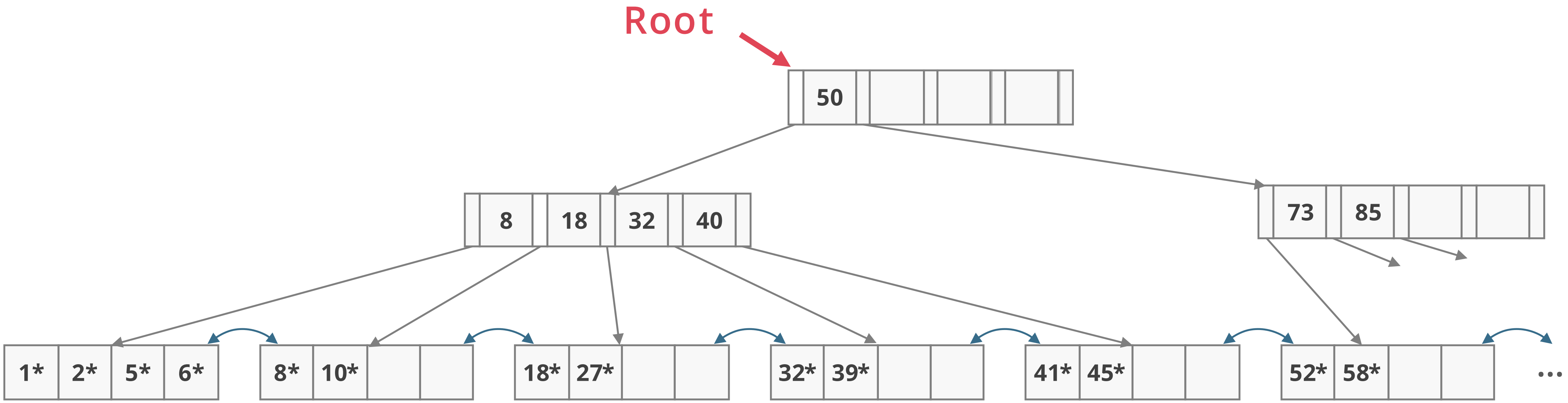
Use binary search on each page



B+ Tree: Search for 39



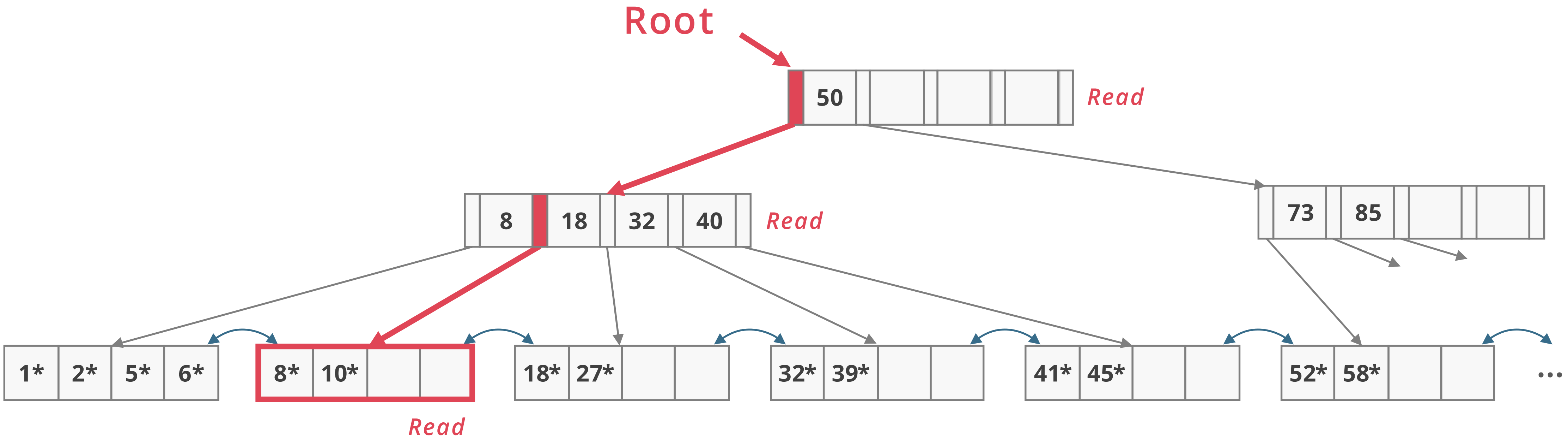
B+ Tree: Insert Entry 9*



I/O Total (so far): 0

B+ Tree: Insert Entry 9*

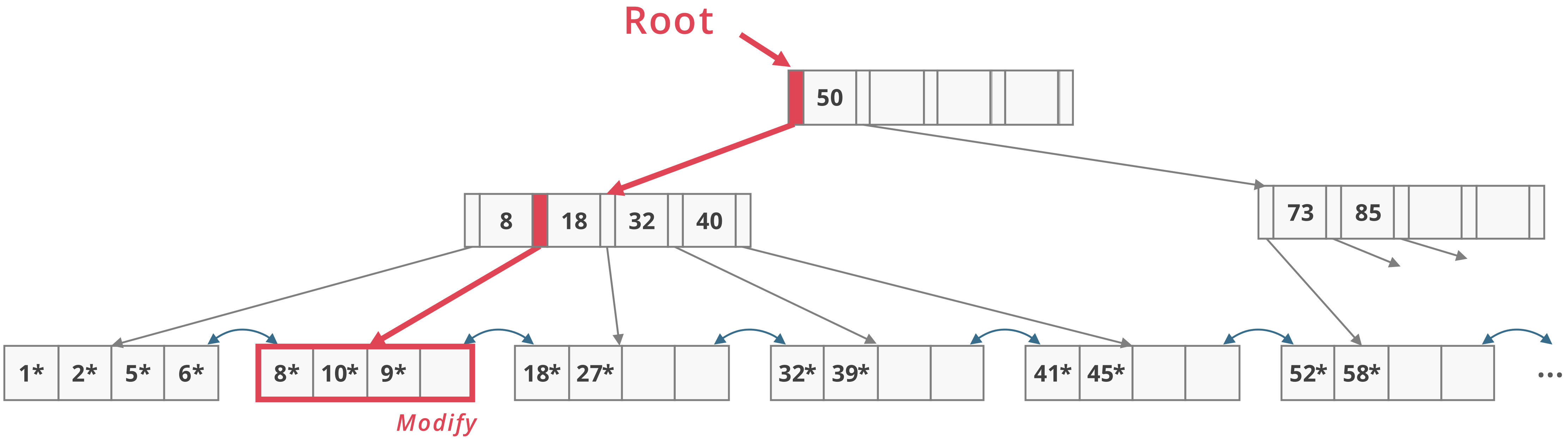
Find the correct leaf node



I/O Total (so far): 3

B+ Tree: Insert Entry 9*

If there is room in leaf, just add the entry

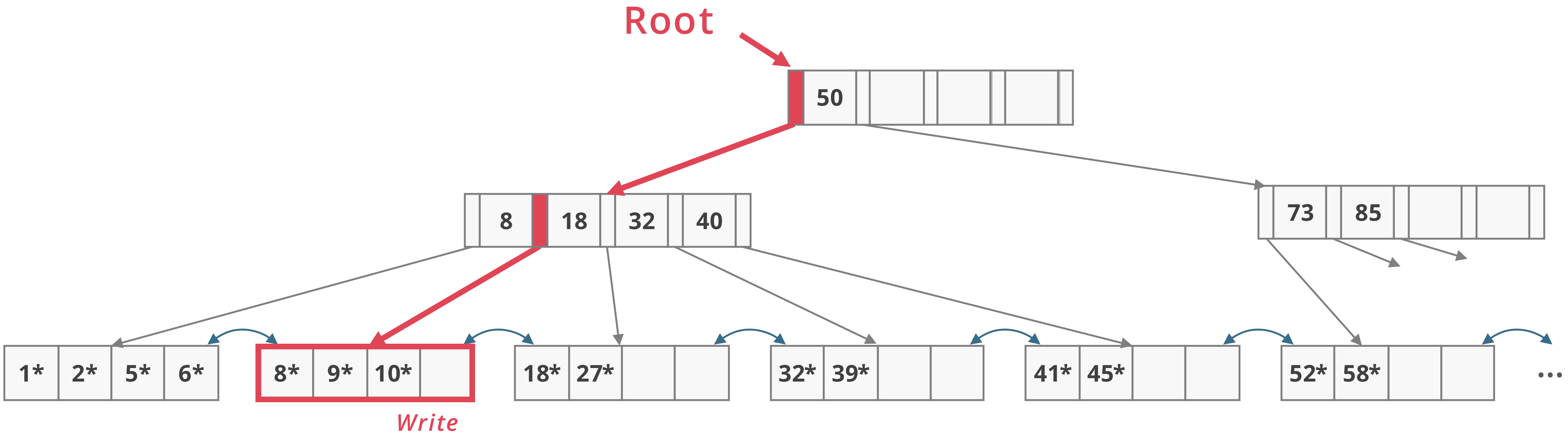


I/O Total (so far): 3

B+ Tree: Insert Entry 9*

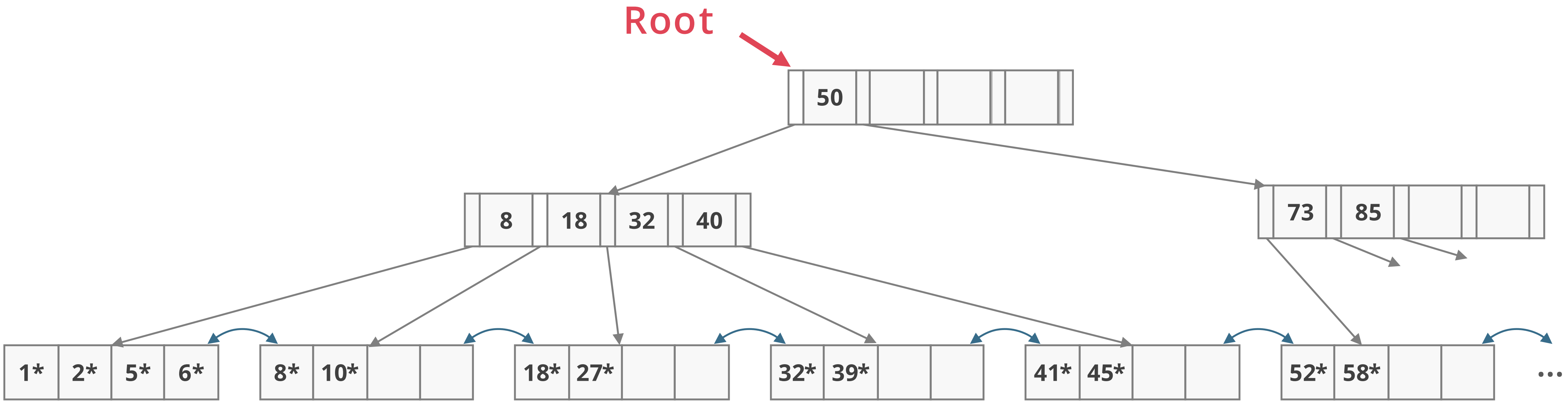
If there is room in leaf, just add the entry

... and keep the leaf sorted



I/O Total (so far): 4

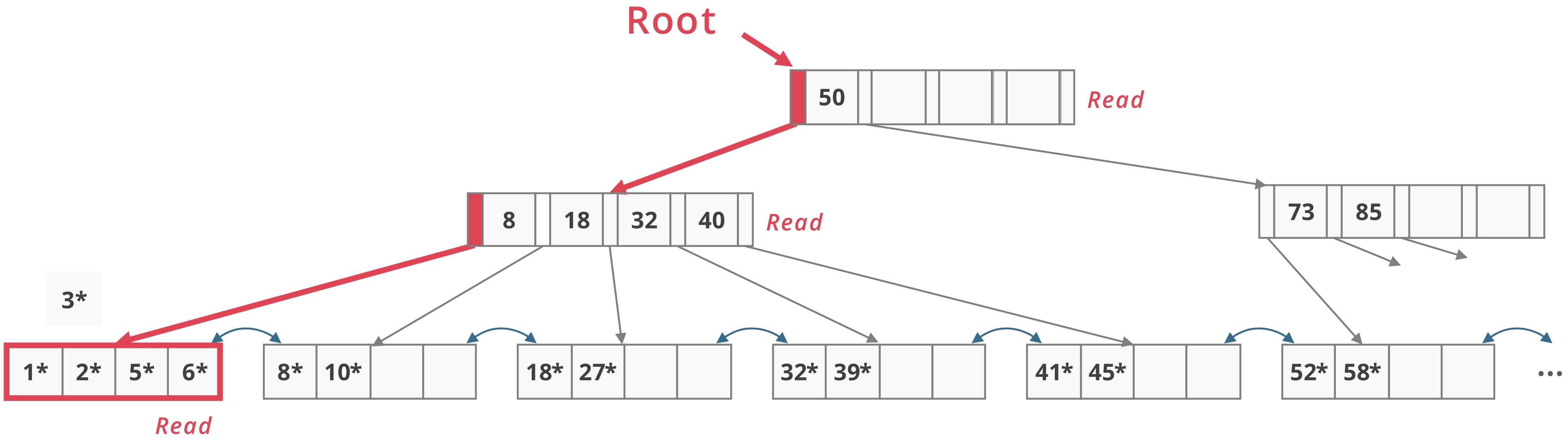
B+ Tree: Insert Entry 3*



I/O Total (so far): 0

B+ Tree: Insert Entry 3*

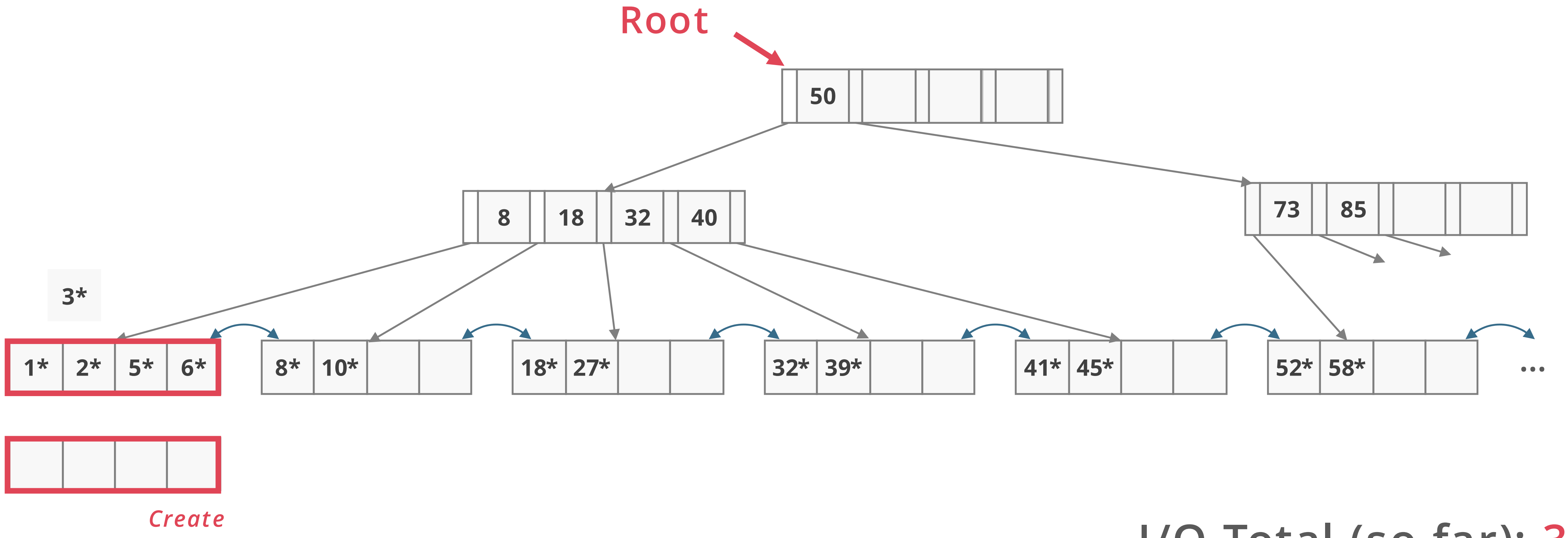
Find the correct leaf node



I/O Total (so far): 3

B+ Tree: Insert Entry 3*

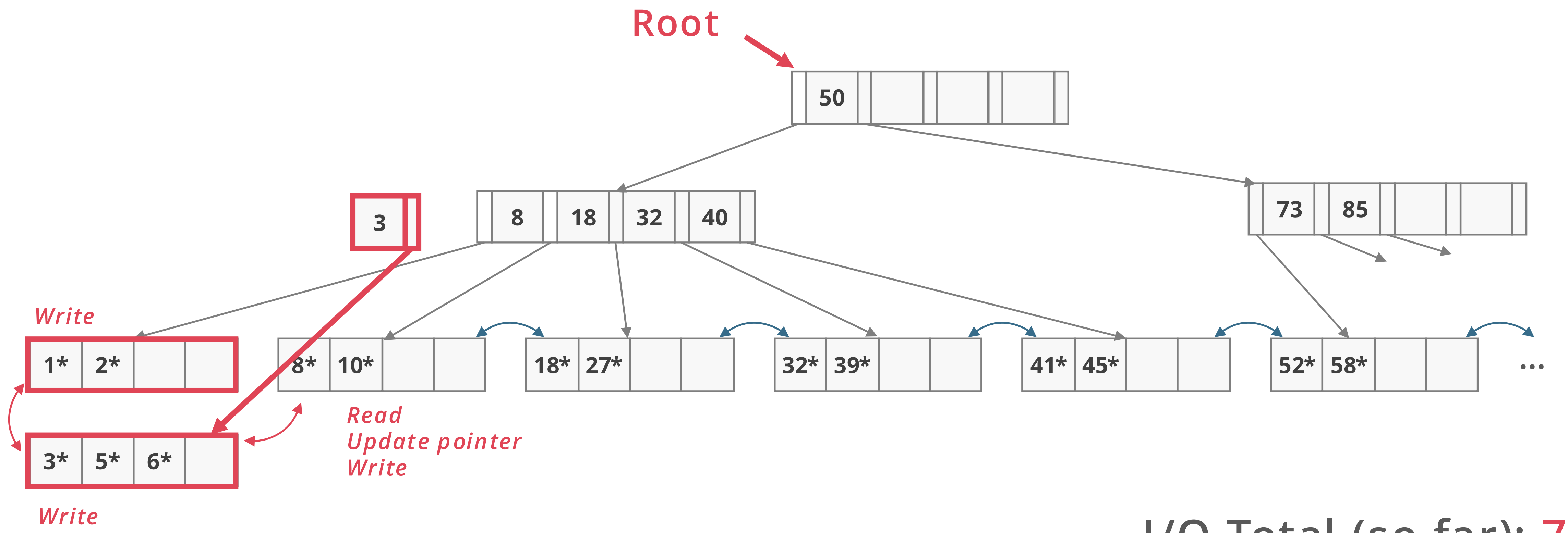
Split leaf if not enough room: into two leaves with d and d + 1 entries



I/O Total (so far): 3

B+ Tree: Insert Entry 3*

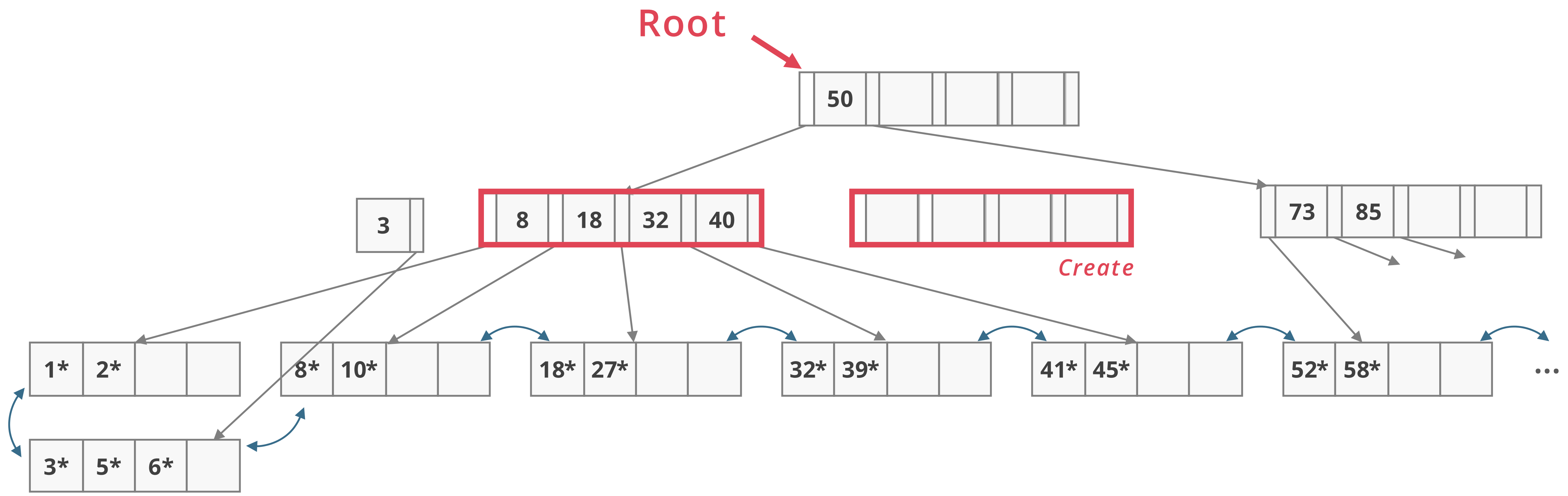
Copy up the middle key to inner node (since leaf nodes have data)



I/O Total (so far): 7

B+ Tree: Insert Entry 3*

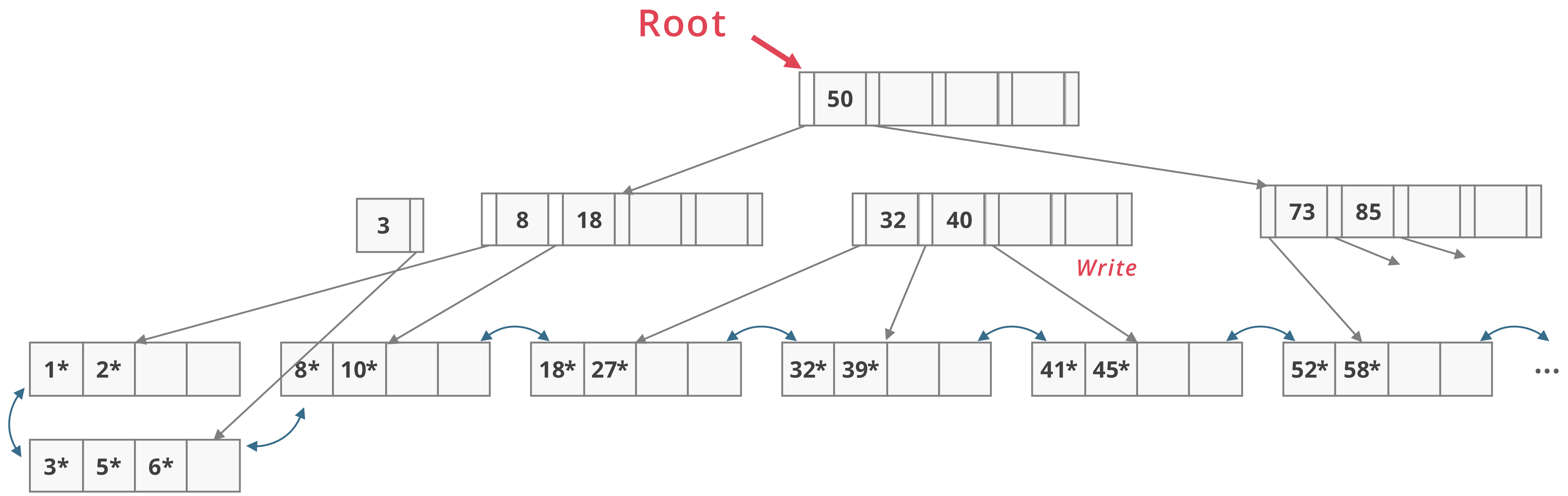
If inner node is full, split the inner node into two and **push** the middle key up



I/O Total (so far): 7

B+ Tree: Insert Entry 3*

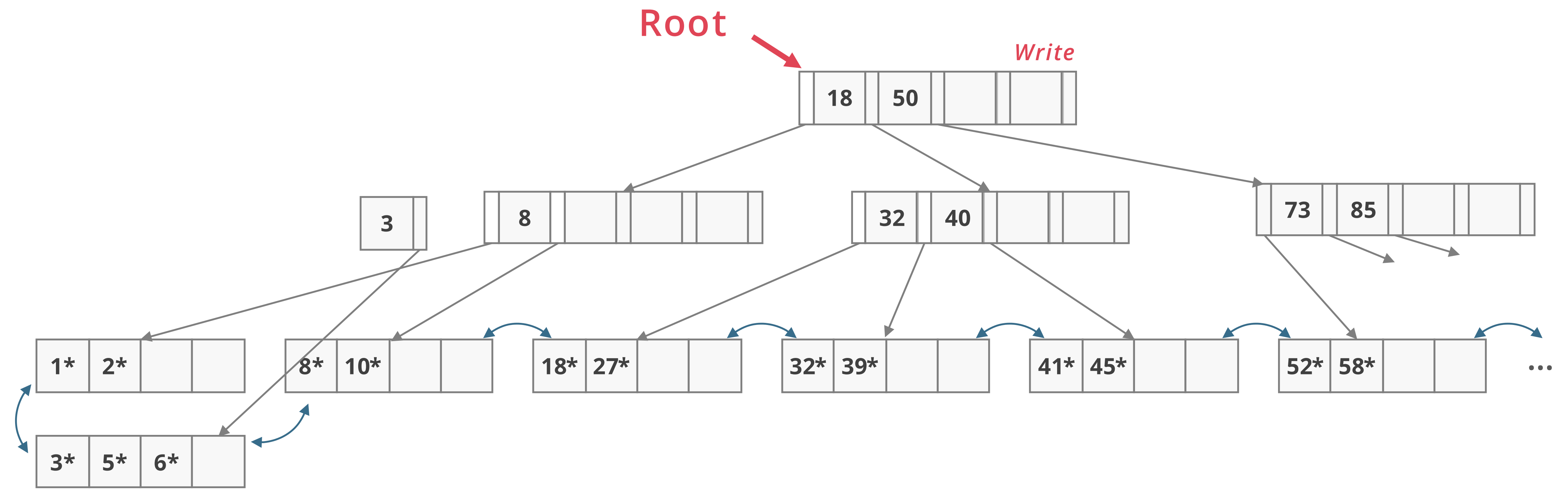
If inner node is full, split the inner node into two and **push** the middle key up



I/O Total (so far): 8

B+ Tree: Insert Entry 3*

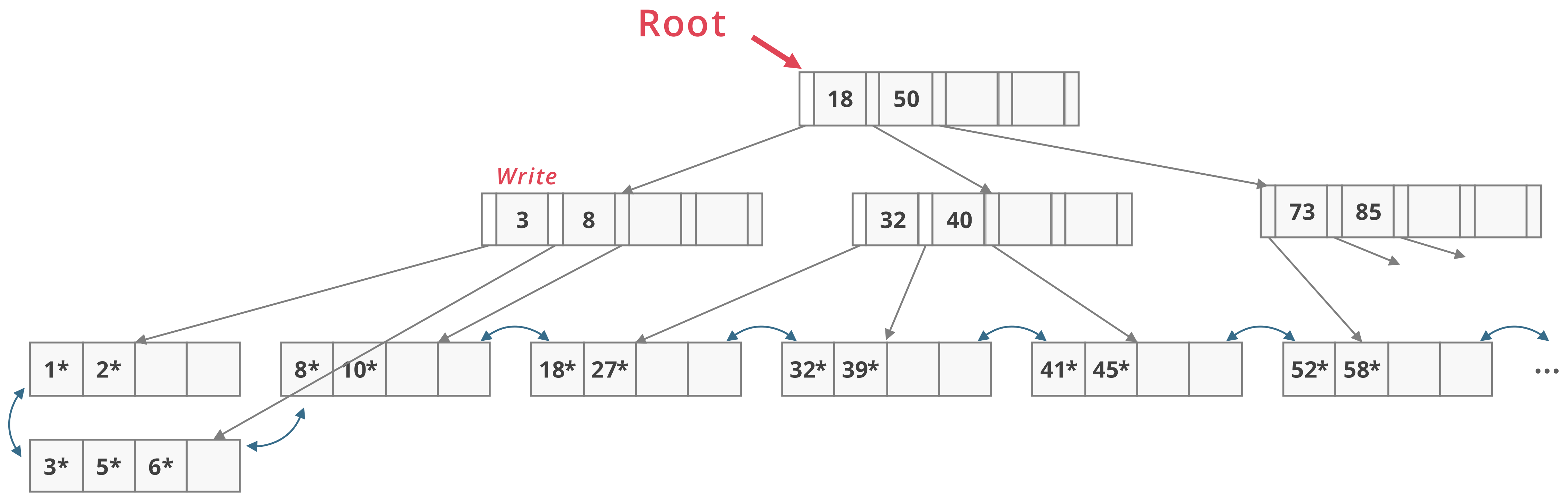
If inner node is full, split the inner node into two and **push** the middle key up



I/O Total (so far): 9

B+ Tree: Insert Entry 3*

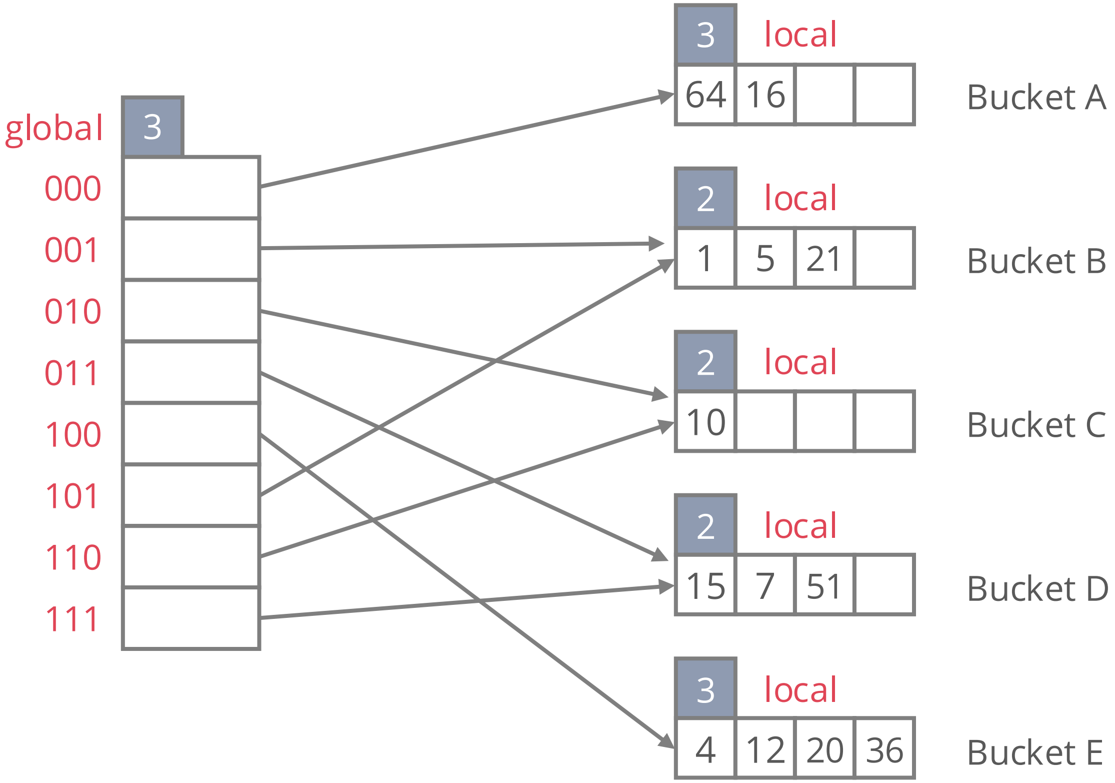
If inner node is full, split the inner node into two and **push** the middle key up



I/O Total (so far): 10

QUESTION 2

Extendible Hashing



global 3

000

001

010

011

100

101

110

111

3 local

64 16

Bucket A

2 local

1 5 21

Bucket B

2 local

10

Bucket C

2 local

15 7 51

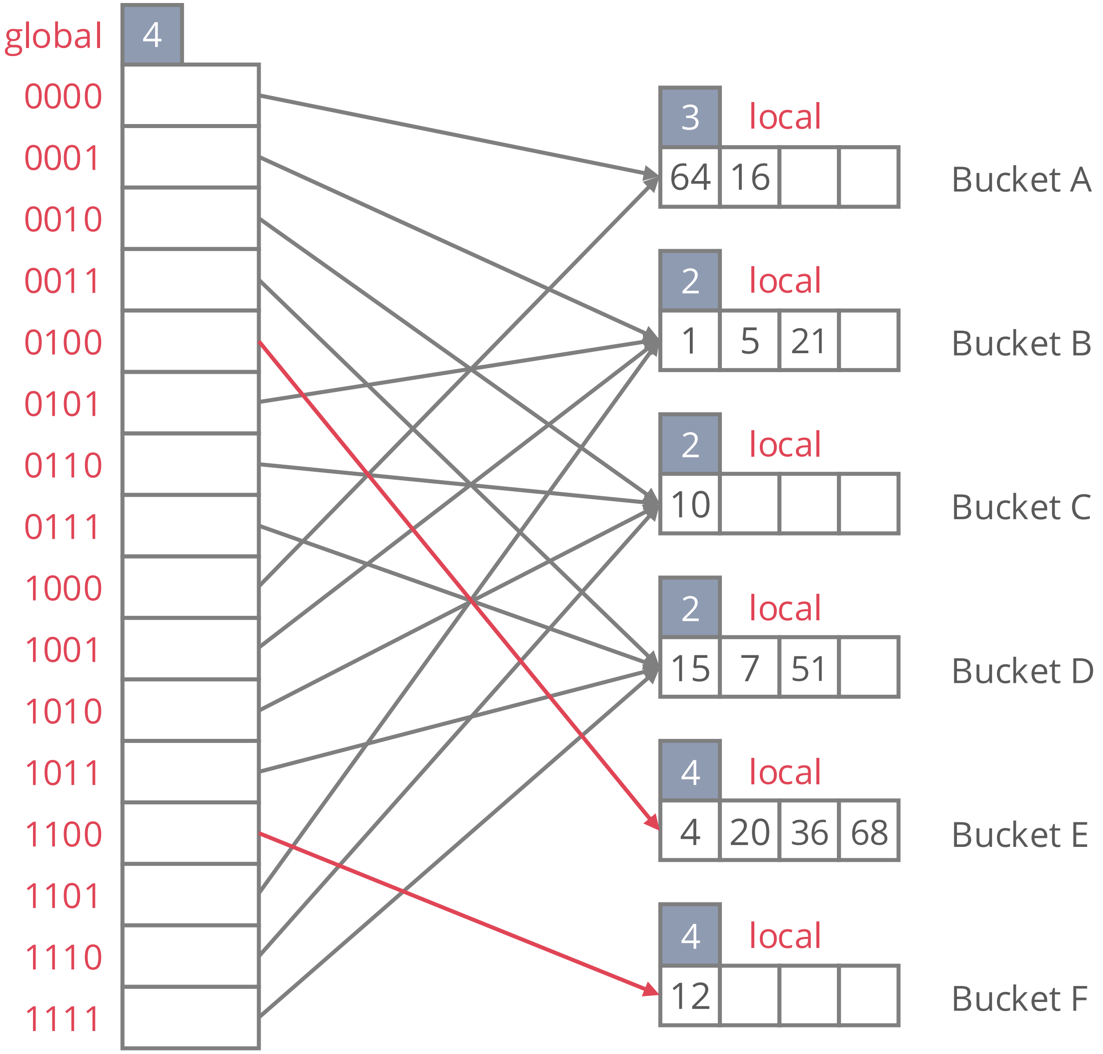
Bucket D

3 local

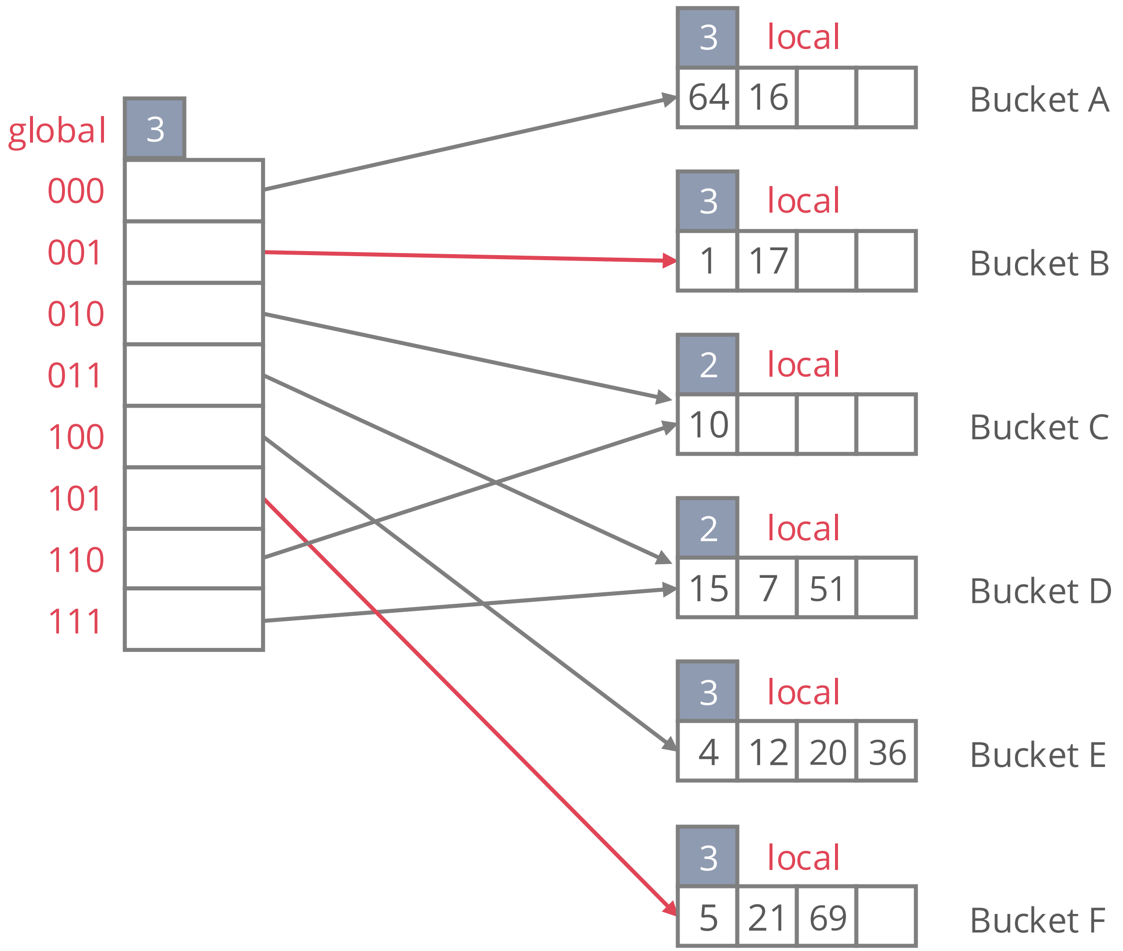
4 12 20 36

Bucket E

INSERT ENTRY WITH HASH VALUE 68



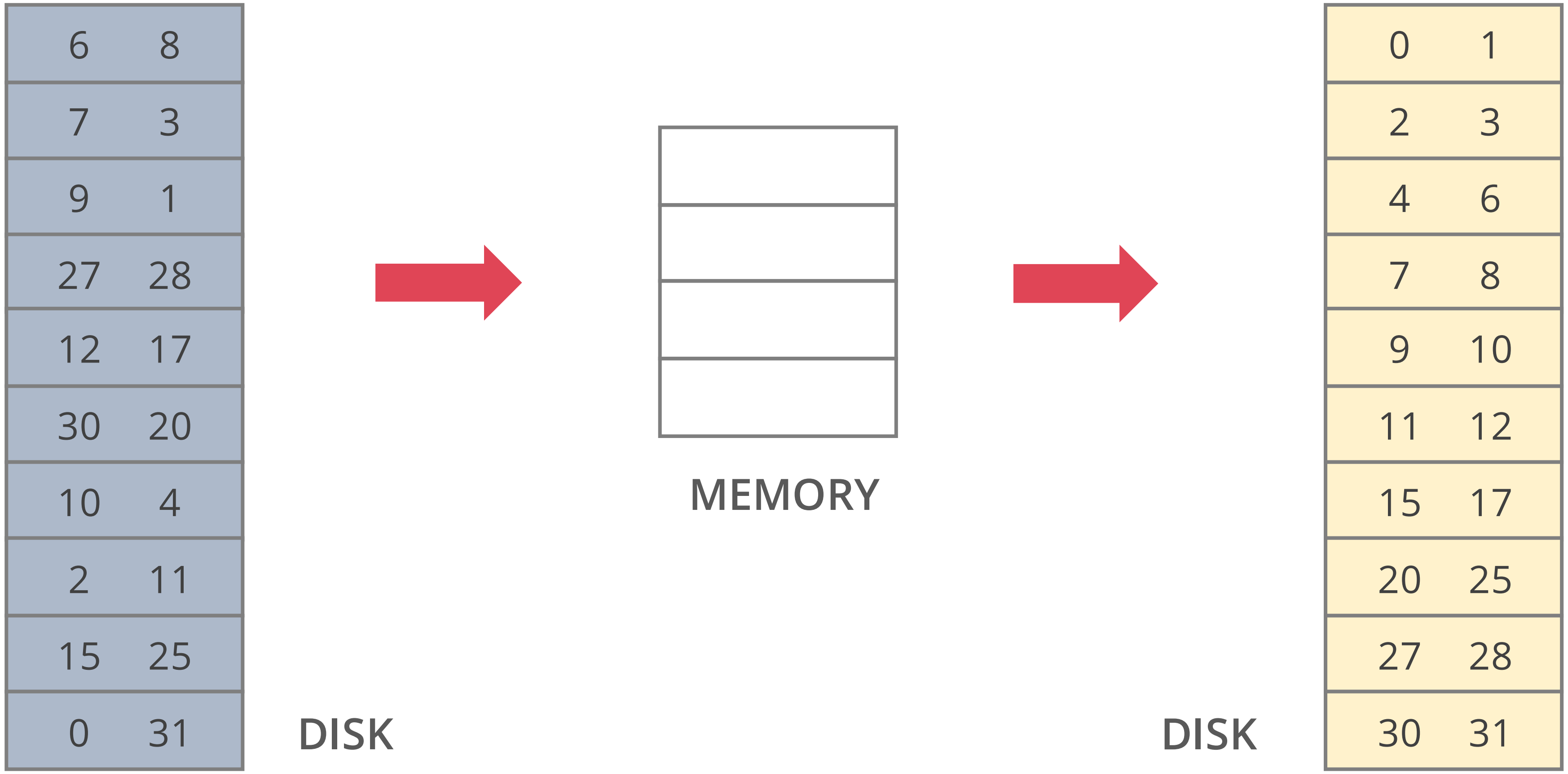
Insert Entries with Hash Values 17 & 69



External Sorting

General External Merge Sort

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



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

DISK

MEMORY

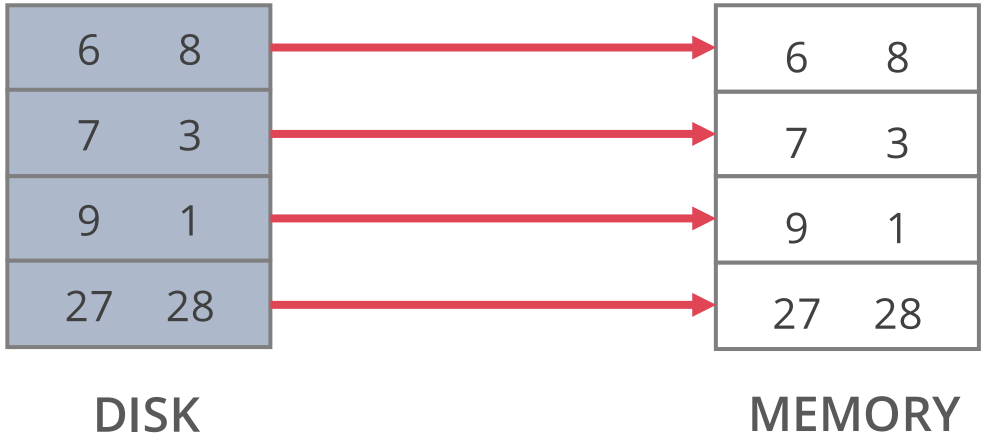
DISK

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

General External Merge Sort

Pass 0, Run 1

Read 4 pages into memory



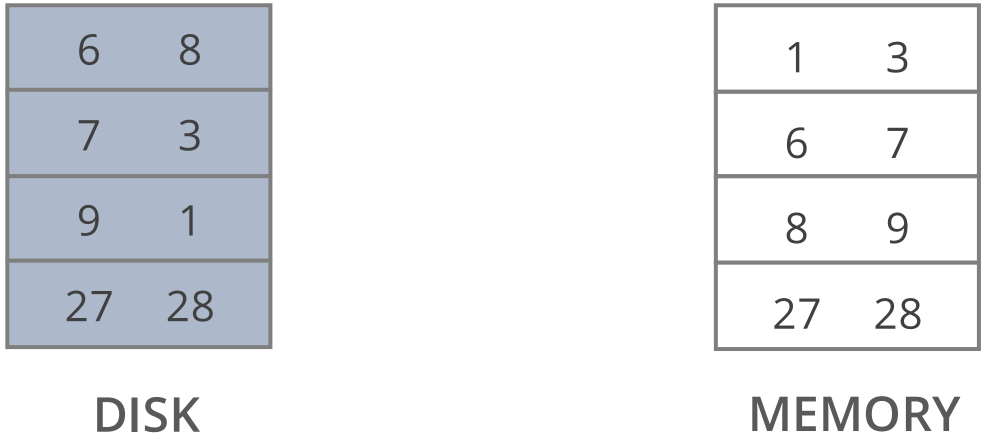
DISK		MEMORY	
6	8	6	8
7	3	7	3
9	1	9	1
27	28	27	28

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

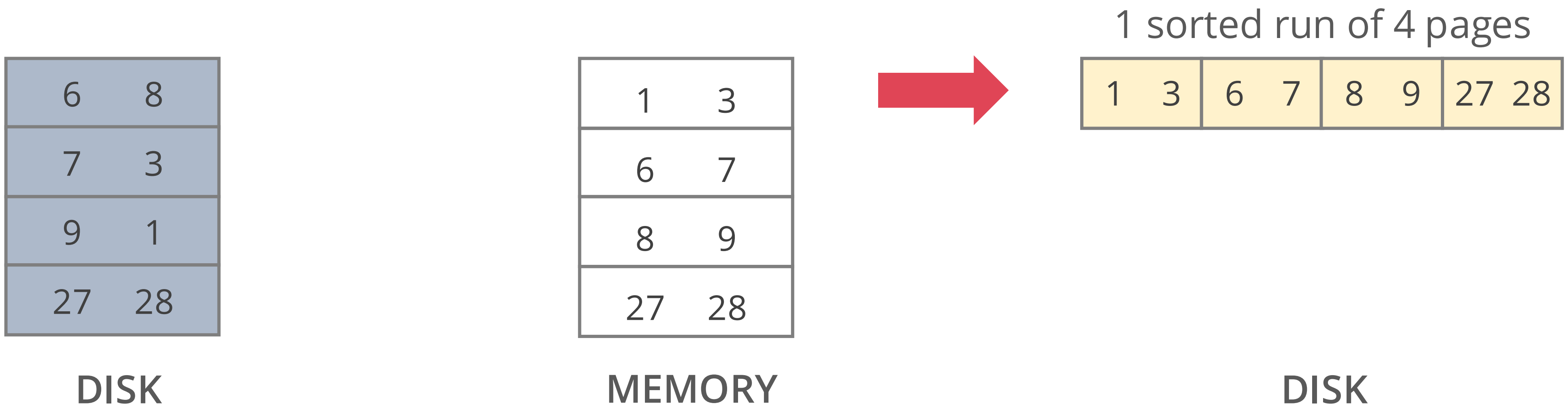
MEMORY

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

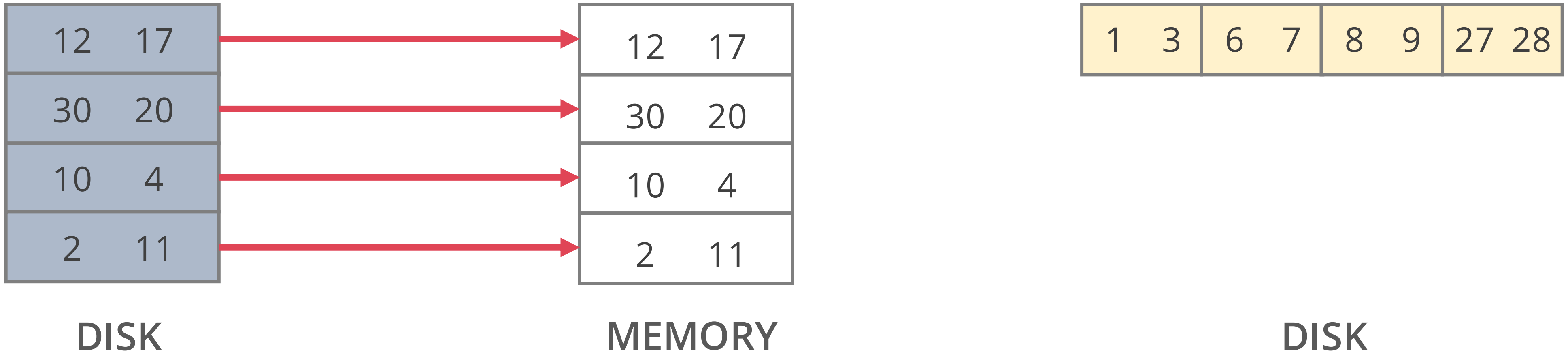
DISK

I/O Total (so far): 8

General External Merge Sort

Pass 0, Run 2

Read 4 pages into memory

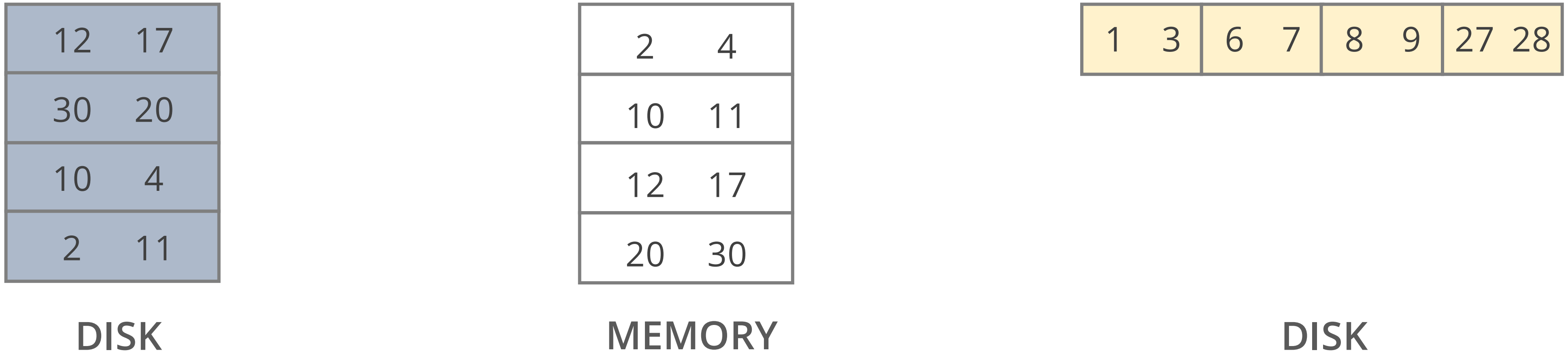


I/O Total (so far): 12

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

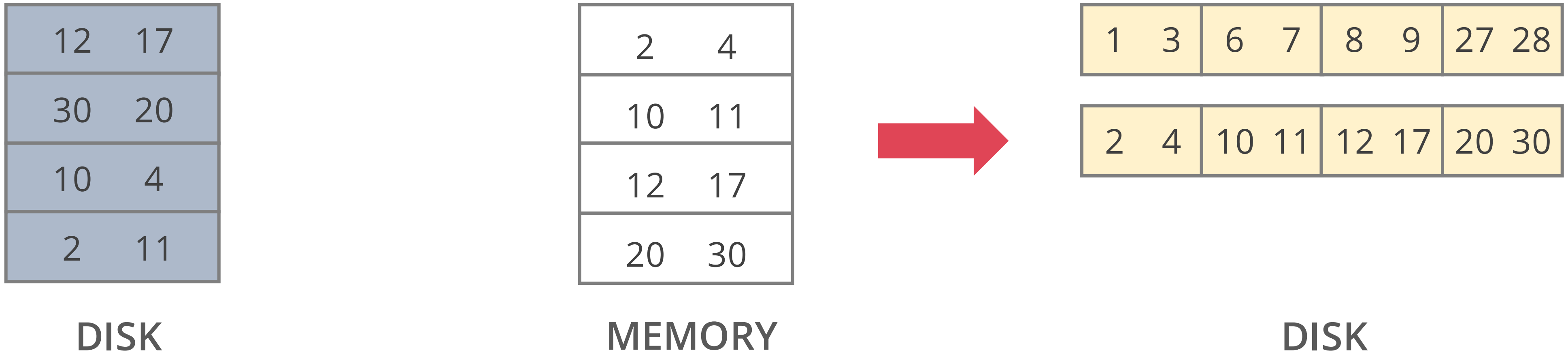
DISK

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

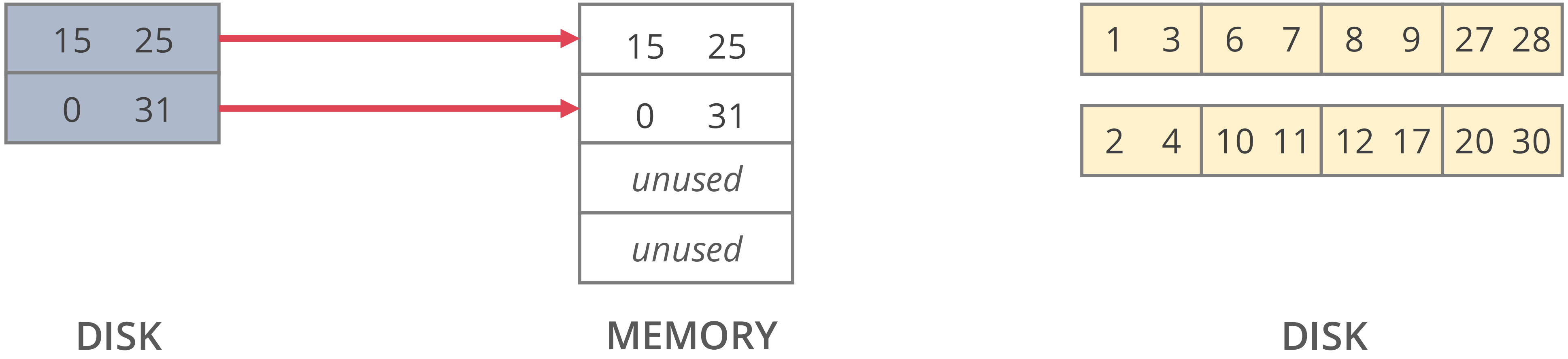
DISK

I/O Total (so far): 16

General External Merge Sort

Pass 0, Run 3

Read 2 pages into memory



15 25

0 31

15 25

0 31

unused

unused

1 3 | 6 7 | 8 9 | 27 28

2 4 | 10 11 | 12 17 | 20 30

DISK

MEMORY

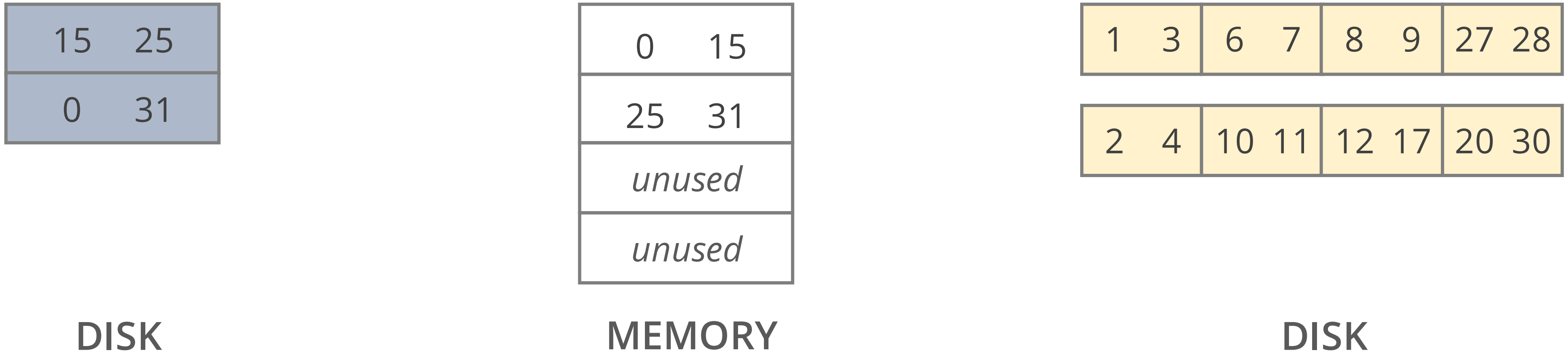
DISK

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

unused

unused

MEMORY

1 3 | 6 7 | 8 9 | 27 28

2 4 | 10 11 | 12 17 | 20 30

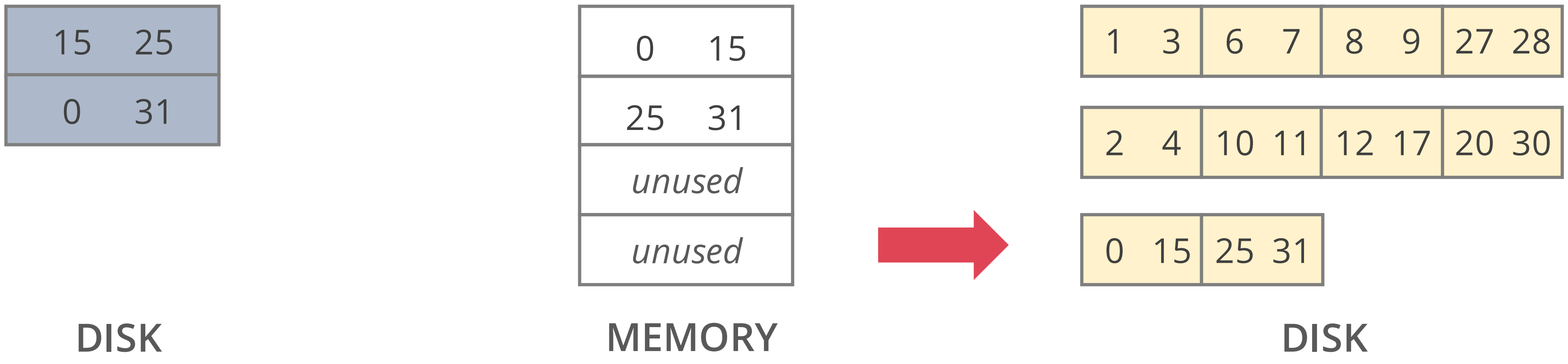
DISK

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

unused

unused

MEMORY

1 3 | 6 7 | 8 9 | 27 28

2 4 | 10 11 | 12 17 | 20 30

0 15 | 25 31

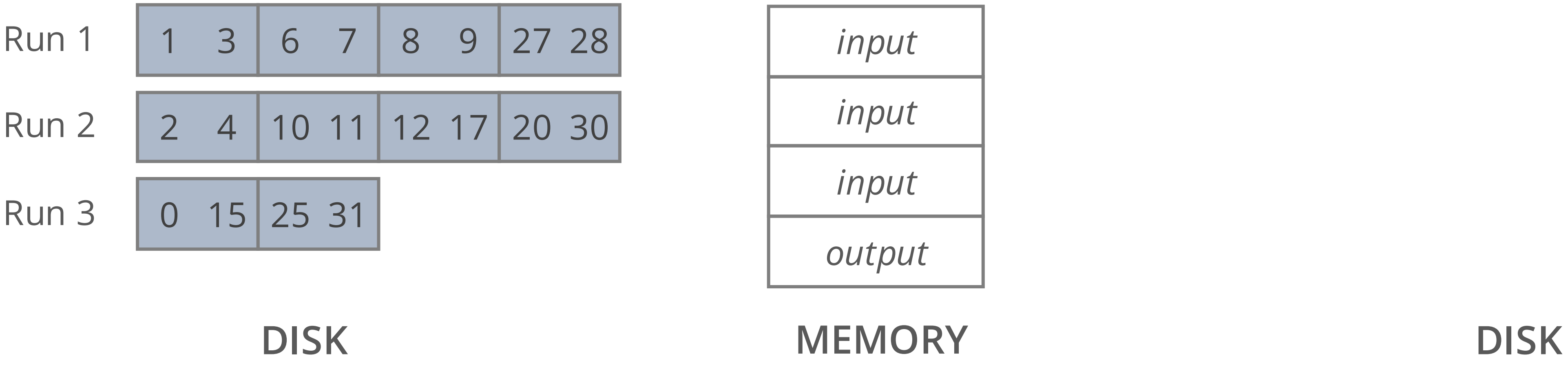
DISK

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		

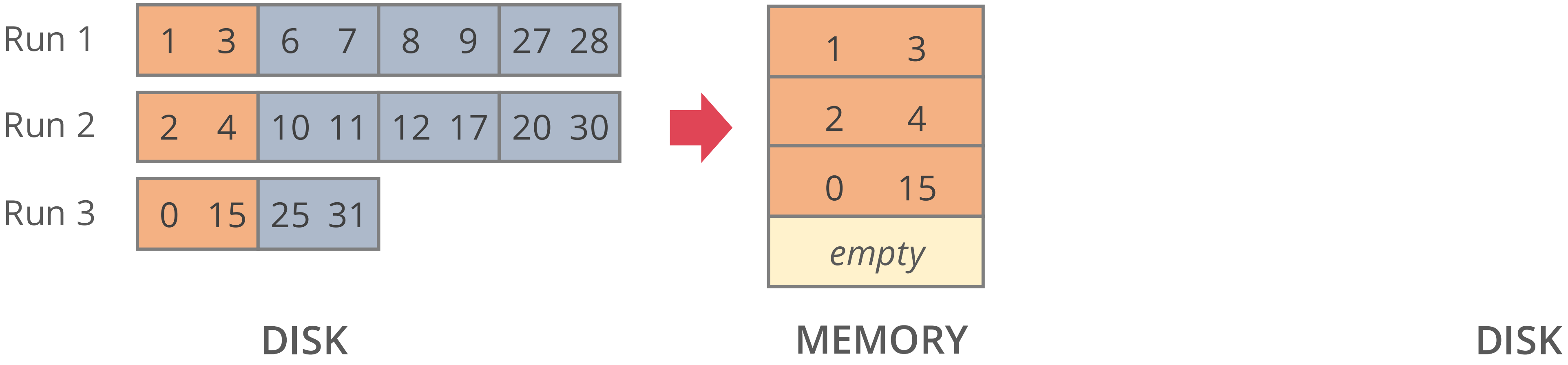
Reserve ***B-1*** input buffers and ***1*** output buffer. Load one page from each run at a time.

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

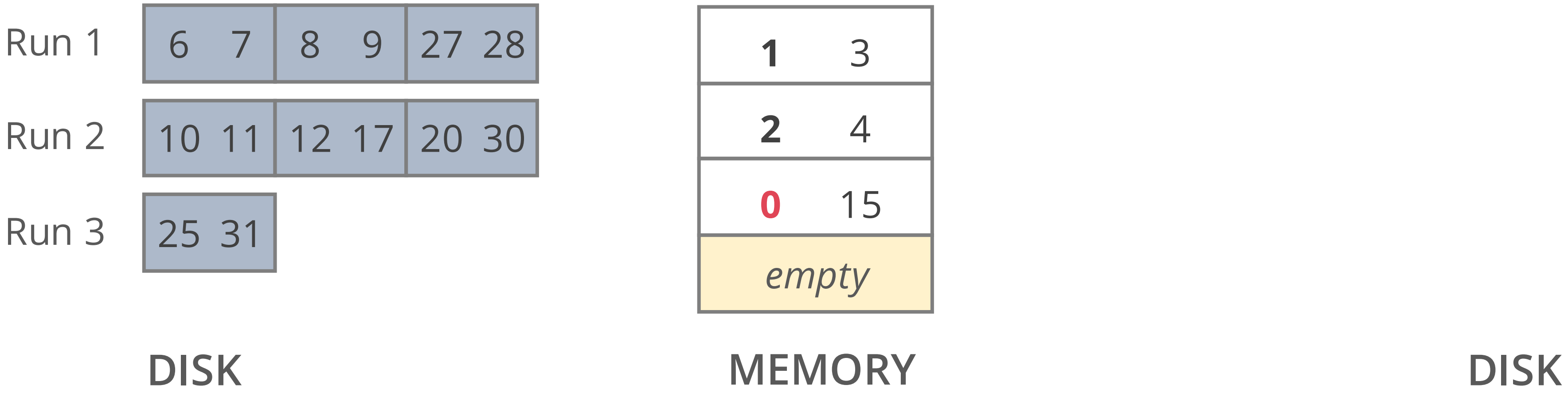


I/O Total (so far): 23

General External Merge Sort

Pass 1

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

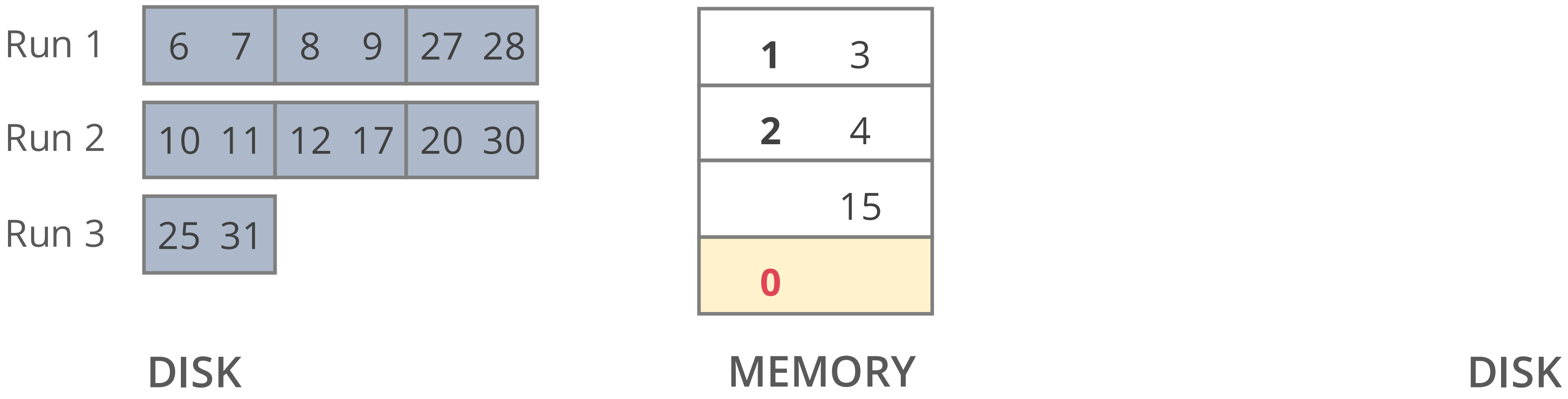


I/O Total (so far): 23

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		

1 3
2 4
15
0

DISK

MEMORY

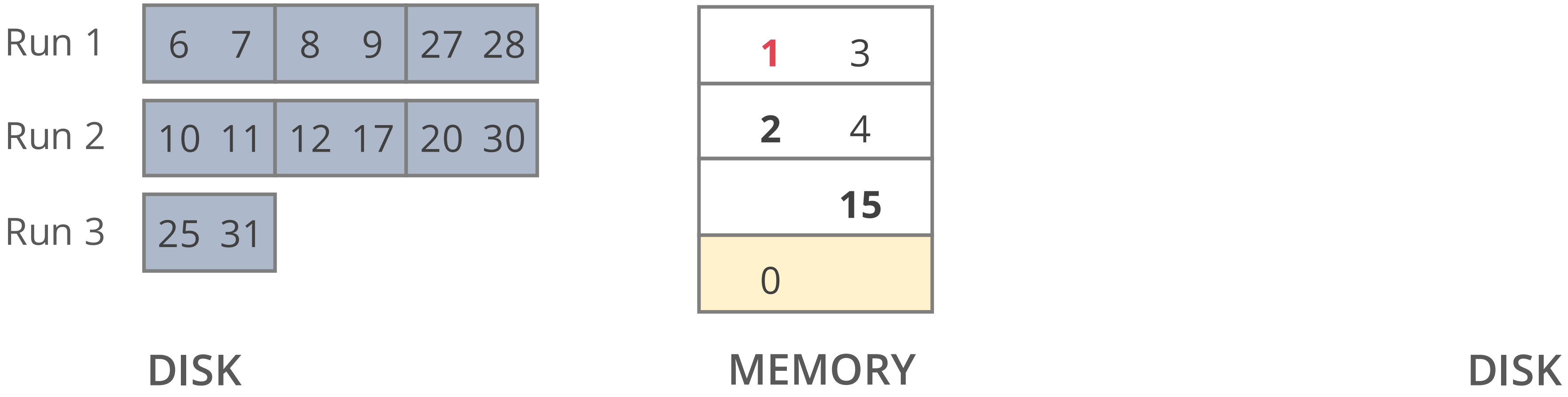
DISK

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

1 3

2 4

15

0

DISK

MEMORY

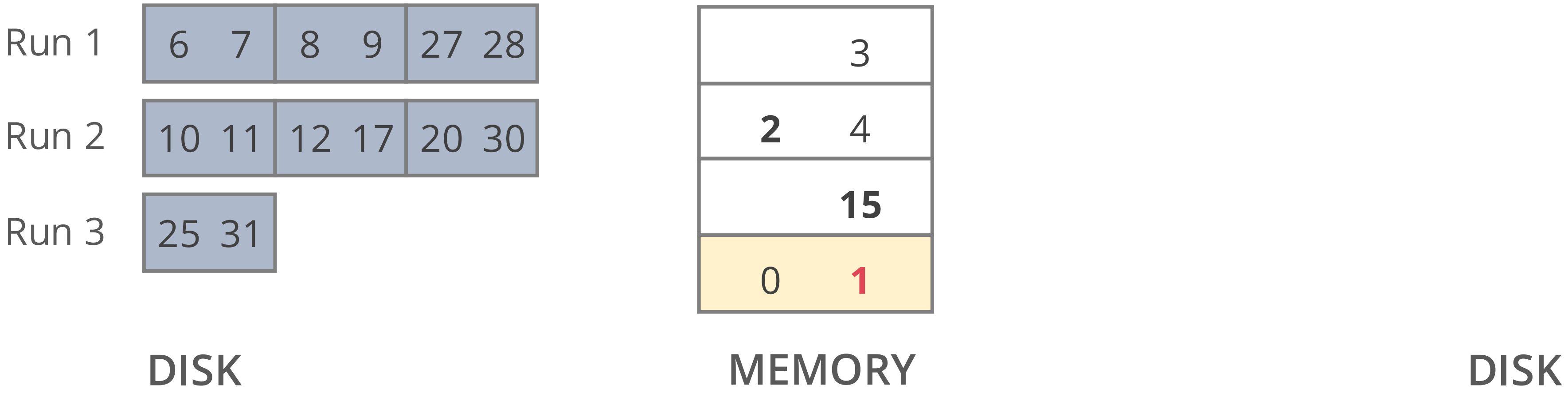
DISK

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

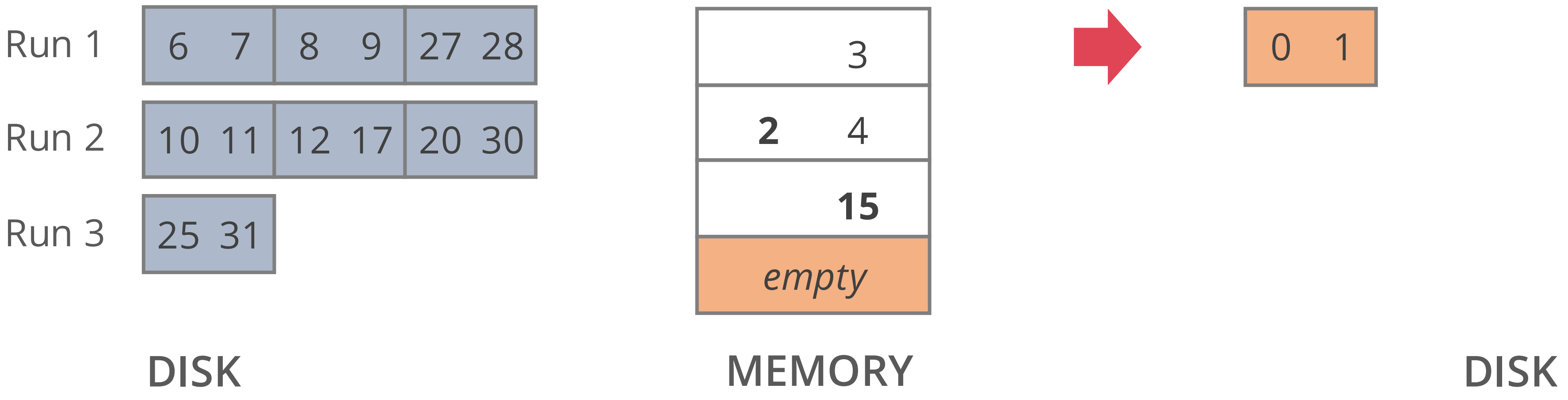
DISK

I/O Total (so far): 23

General External Merge Sort

Pass 1

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



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

empty

MEMORY

0 1

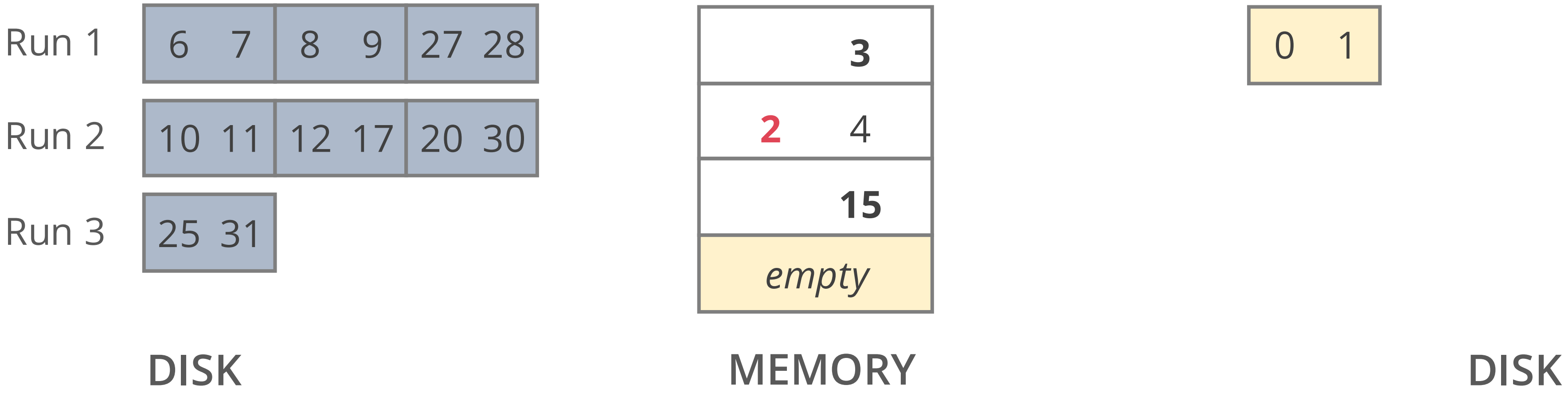
DISK

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

2 4

15

empty

MEMORY

0 1

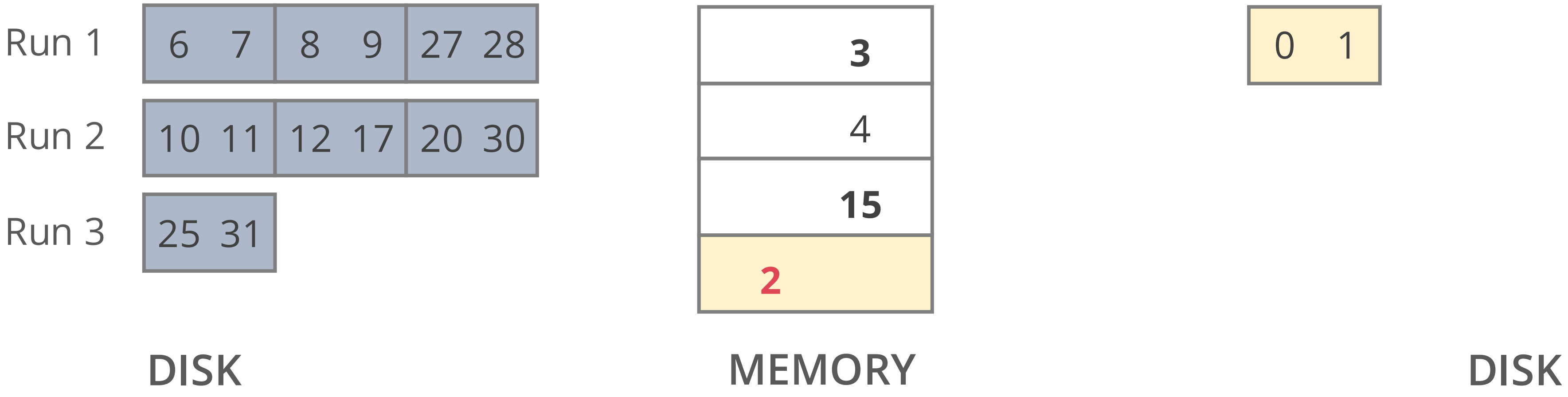
DISK

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

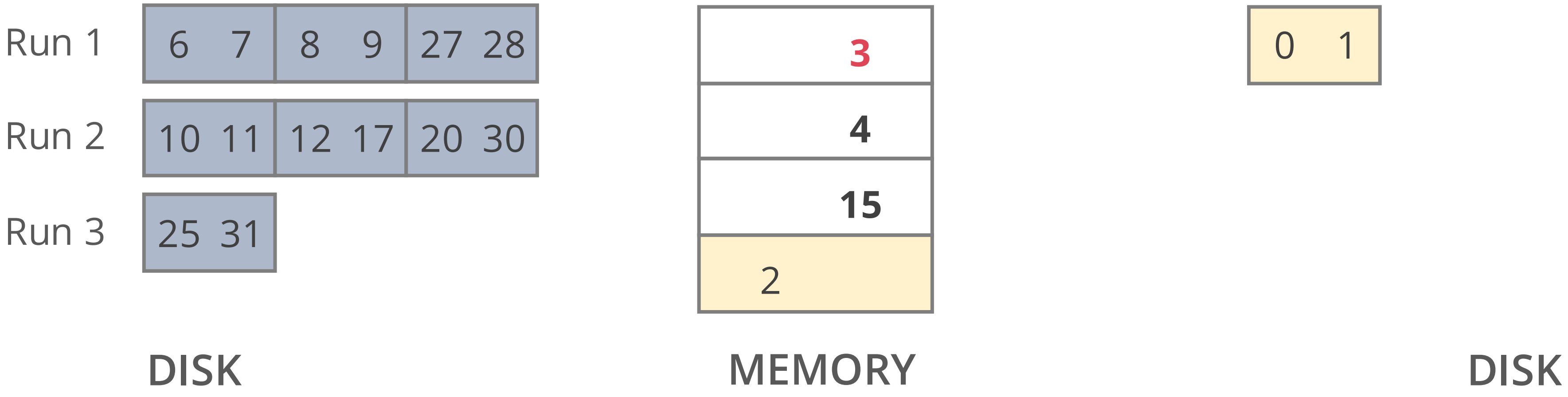
DISK

I/O Total (so far): 24

General External Merge Sort

Pass 1

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

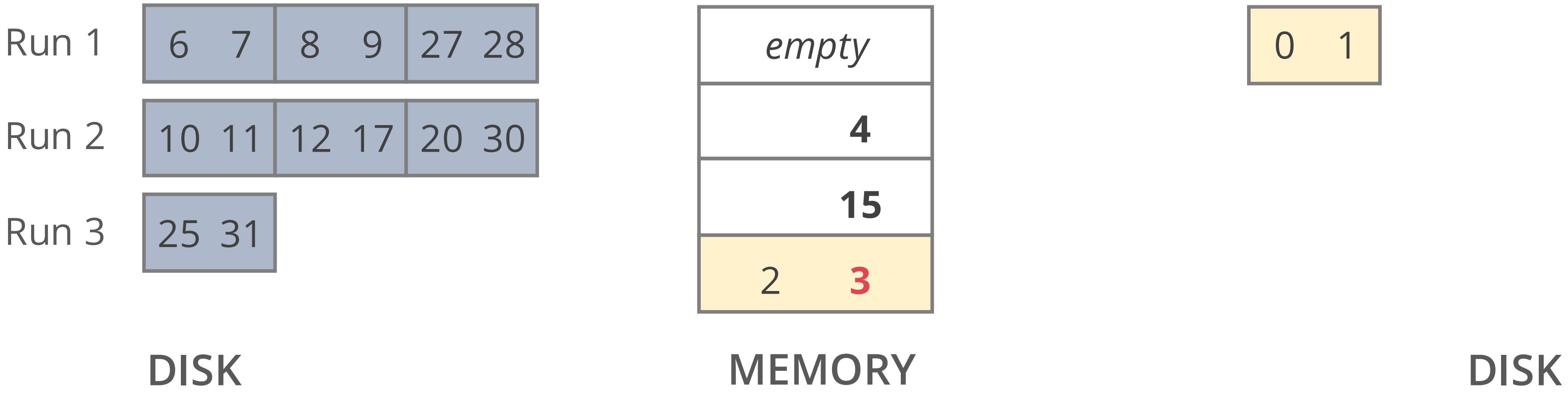


I/O Total (so far): 24

General External Merge Sort

Pass 1

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

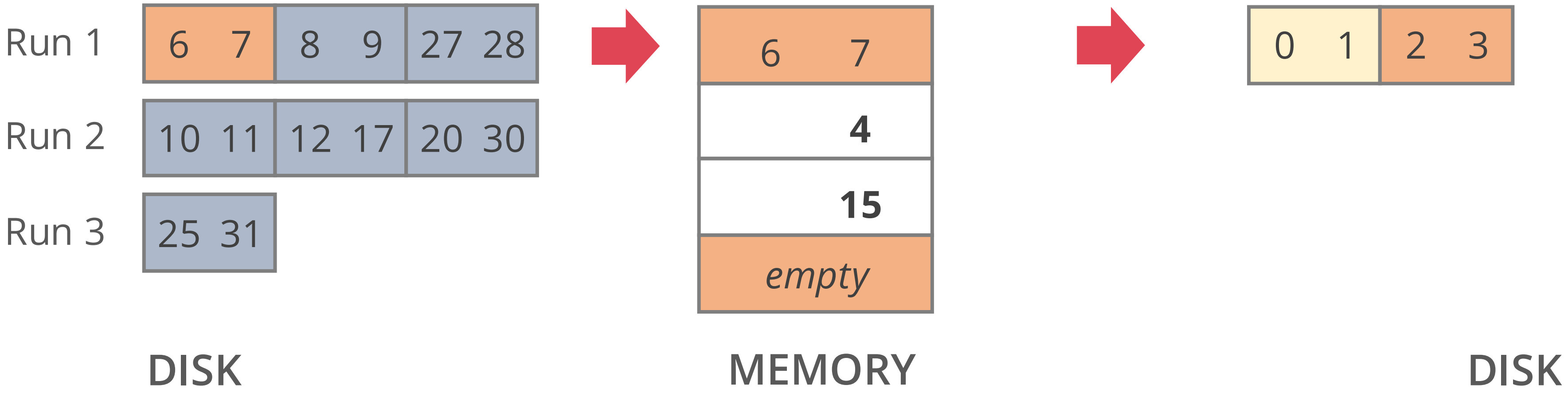


I/O Total (so far): 24

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

6 7
4
15
empty

MEMORY

0 1 | 2 3

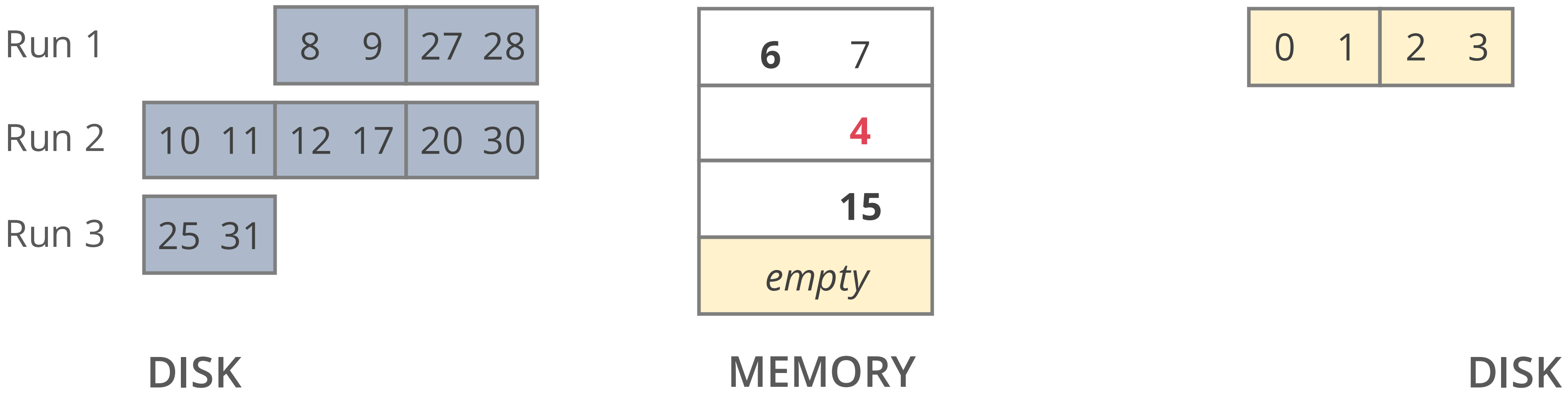
DISK

I/O Total (so far): 26

GENERAL EXTERNAL MERGE SORT

Pass 1

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



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

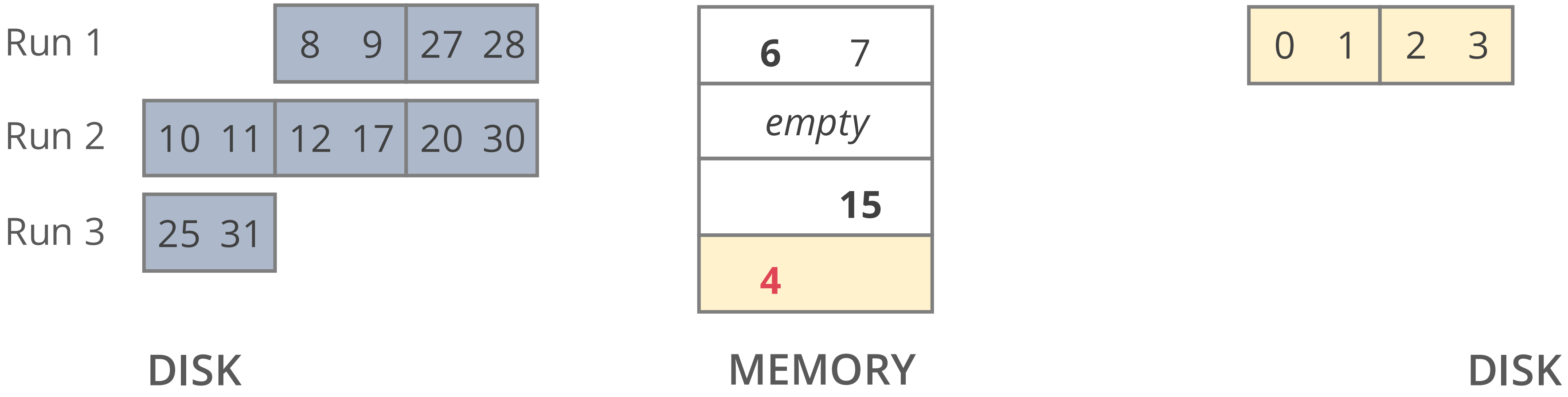
DISK

I/O Total (so far): 26

GENERAL EXTERNAL MERGE SORT

Pass 1

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

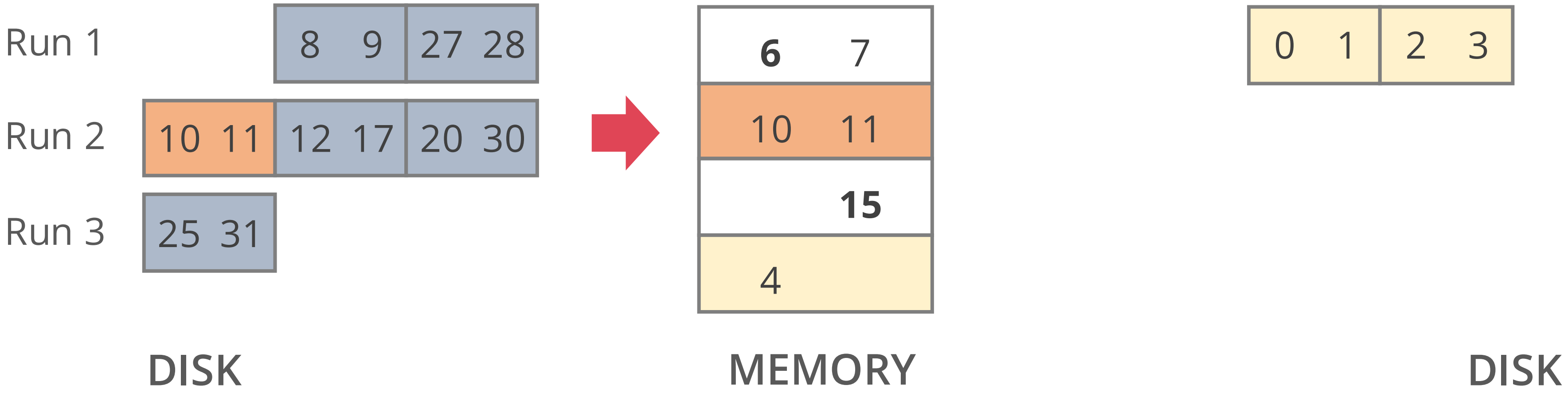


I/O Total (so far): 26

General External Merge Sort

Pass 1

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

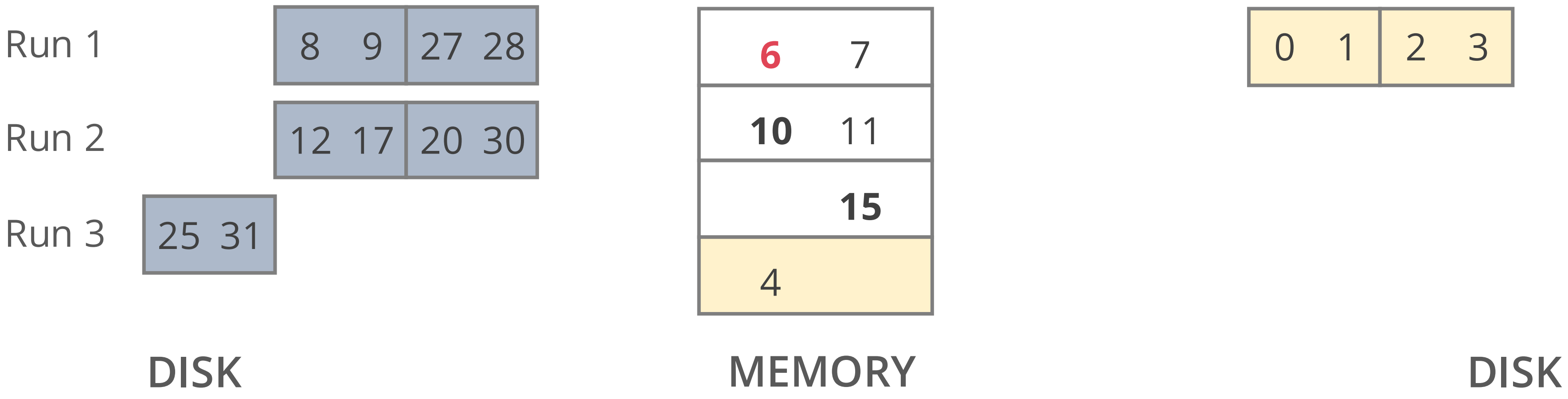


I/O Total (so far): 27

GENERAL EXTERNAL MERGE SORT

Pass 1

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



Run 1 | 8 9 | 27 28

Run 2 | 12 17 | 20 30

Run 3 | 25 31

DISK

MEMORY

6 7

10 11

15

4

DISK

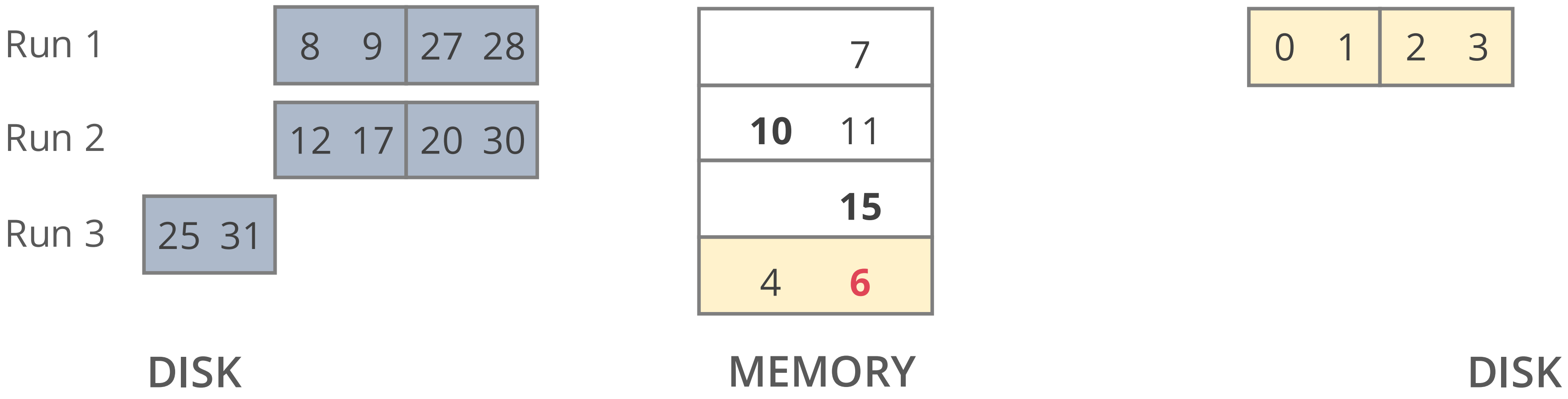
0 1 | 2 3

I/O Total (so far): 27

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 | 12 17 | 20 30

Run 3 | 25 31

DISK

7

10 11

15

4 **6**

MEMORY

0 1 | 2 3

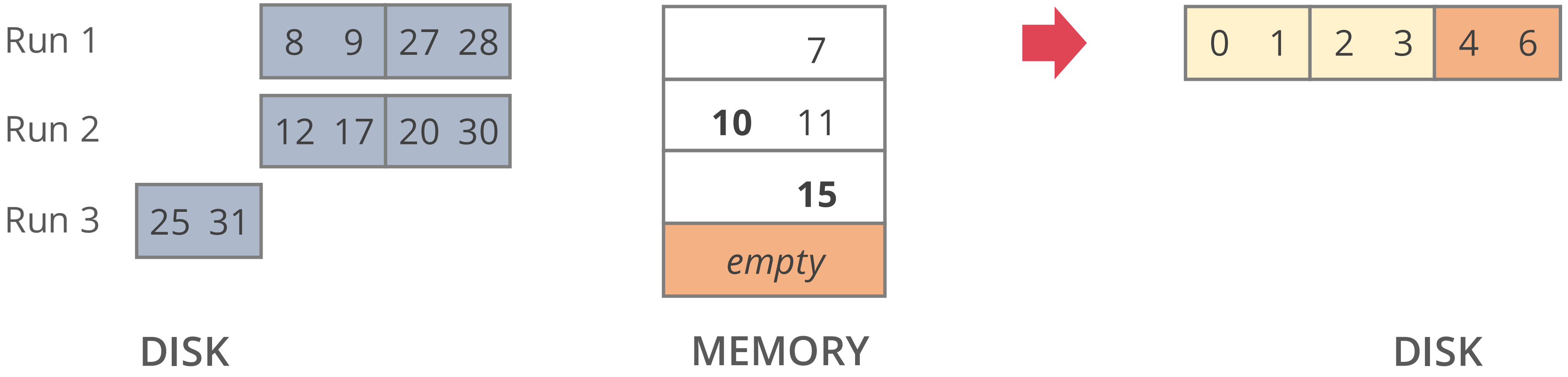
DISK

I/O Total (so far): 27

General External Merge Sort

Pass 1

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



Run 1 8 9 | 27 28

Run 2 12 17 | 20 30

Run 3 25 31

DISK

7

10 11

15

empty

MEMORY

0 1 | 2 3 | 4 6

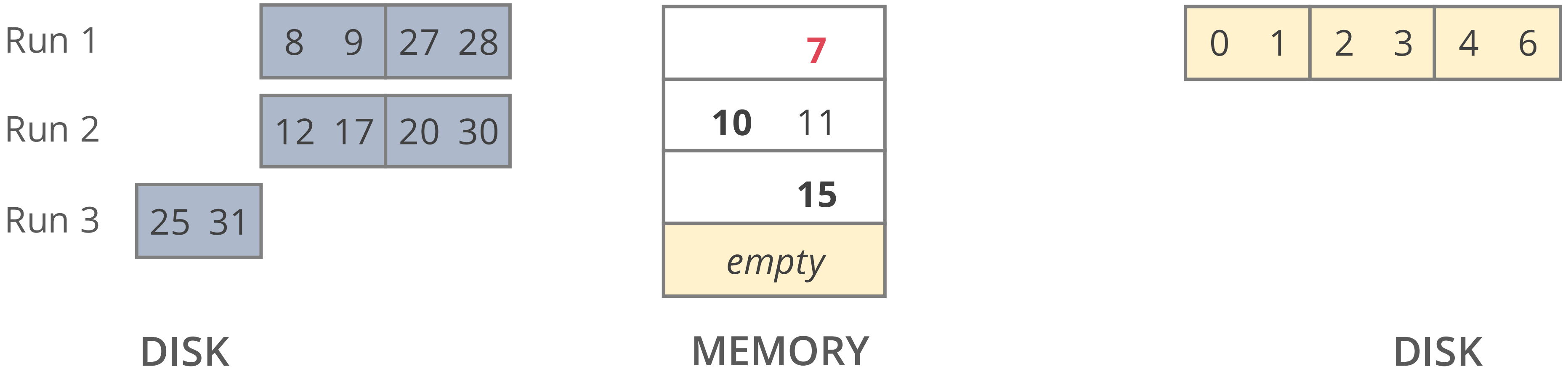
DISK

I/O Total (so far): 28

General External Merge Sort

Pass 1

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

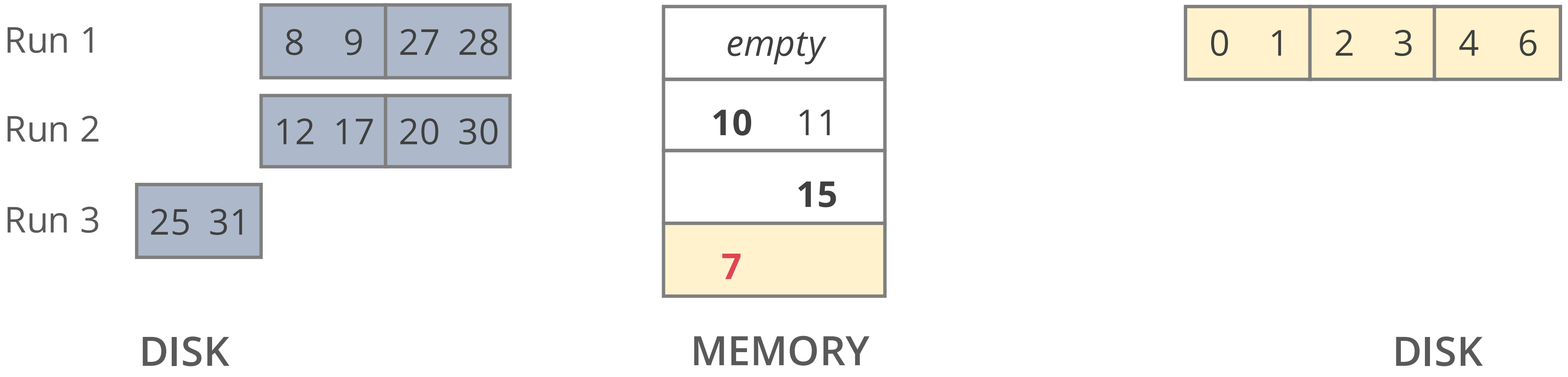


I/O Total (so far): 28

General External Merge Sort

Pass 1

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

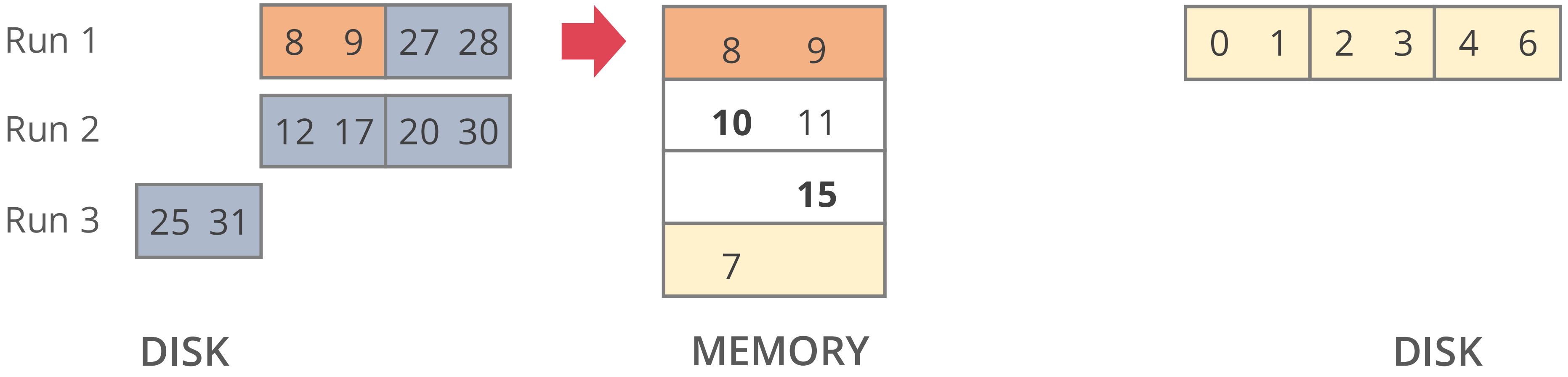


I/O Total (so far): 28

General External Merge Sort

Pass 1

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



Run 1

8 9 | 27 28

Run 2

12 17 | 20 30

Run 3

25 31

DISK

8 9

10 11

15

7

MEMORY

0 1 | 2 3 | 4 6

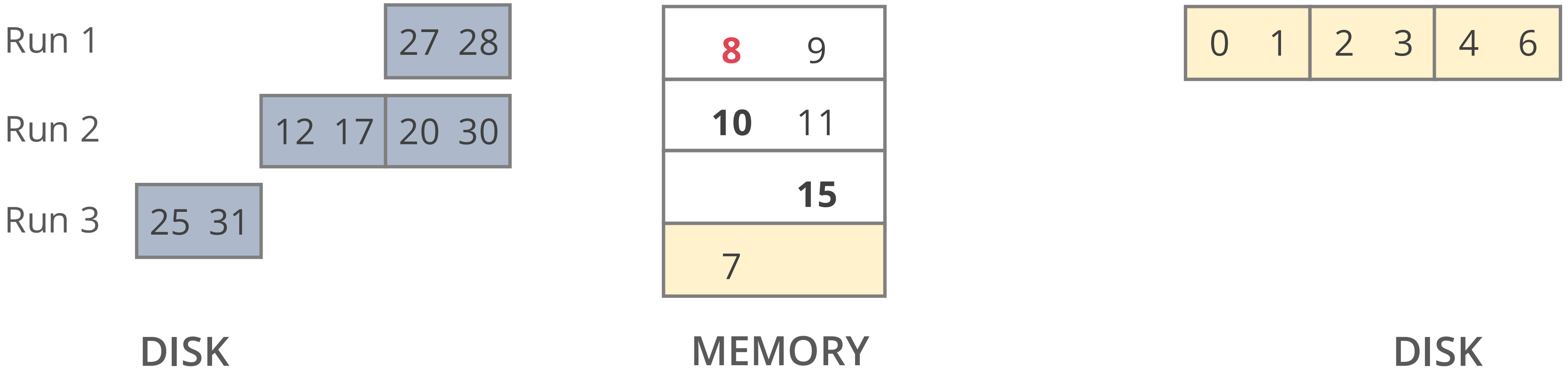
DISK

I/O Total (so far): 29

General External Merge Sort

Pass 1

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

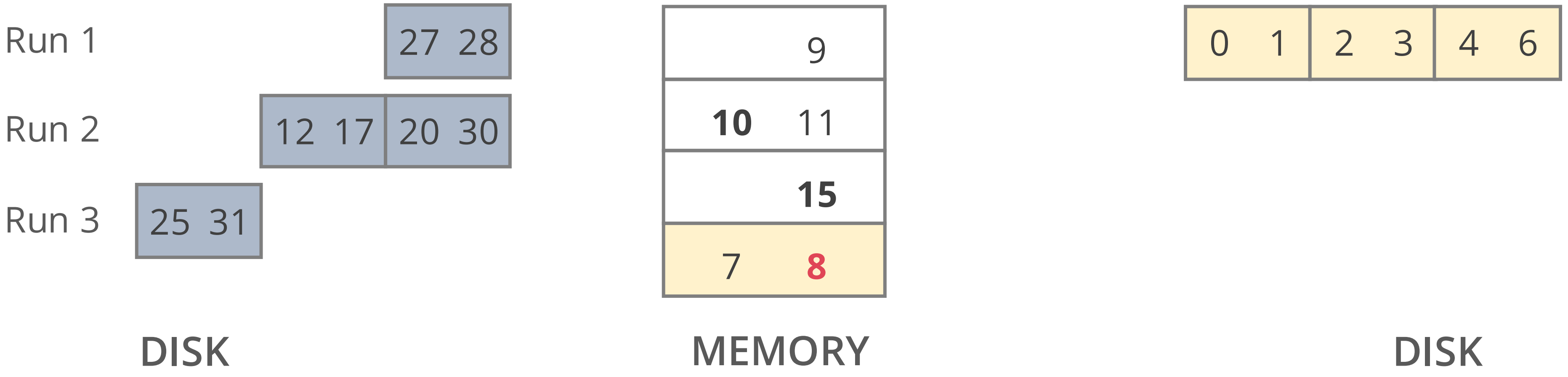


I/O Total (so far): 29

General External Merge Sort

Pass 1

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

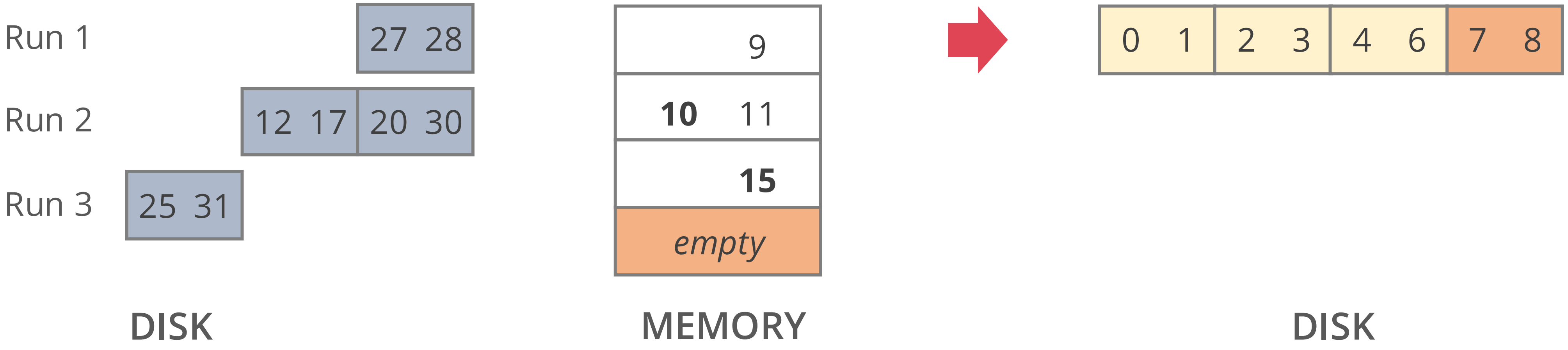


I/O Total (so far): 29

General External Merge Sort

Pass 1

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

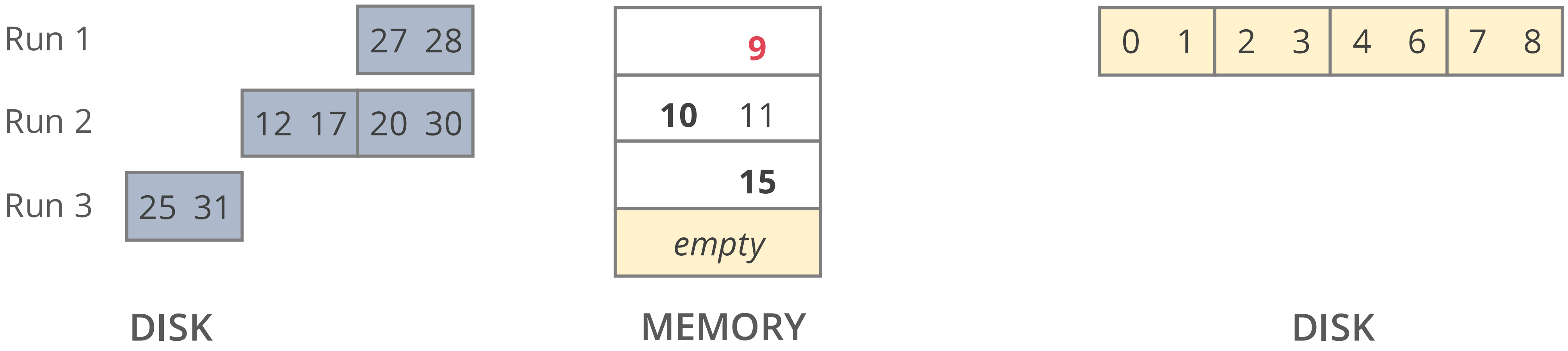


I/O Total (so far): 30

GENERAL EXTERNAL MERGE SORT

Pass 1

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



Run 1 — 27 28

Run 2 — 12 17 | 20 30

Run 3 — 25 31

DISK

9

10 11

15

empty

MEMORY

0 1 | 2 3 | 4 6 | 7 8

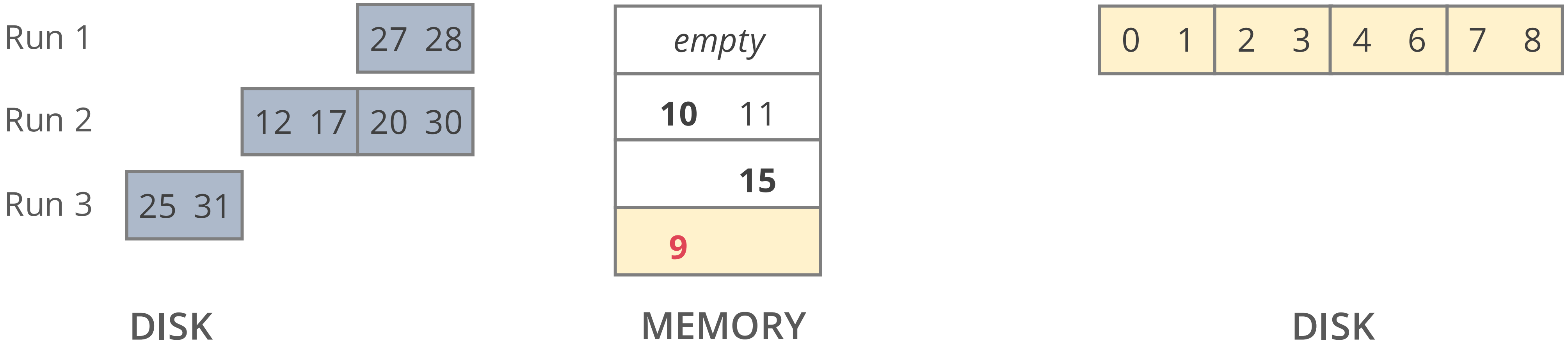
DISK

I/O Total (so far): 30

General External Merge Sort

Pass 1

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



Run 1 | 27 28

Run 2 | 12 17 | 20 30

Run 3 | 25 31

DISK

empty

10 11

15

9

MEMORY

0 1 | 2 3 | 4 6 | 7 8

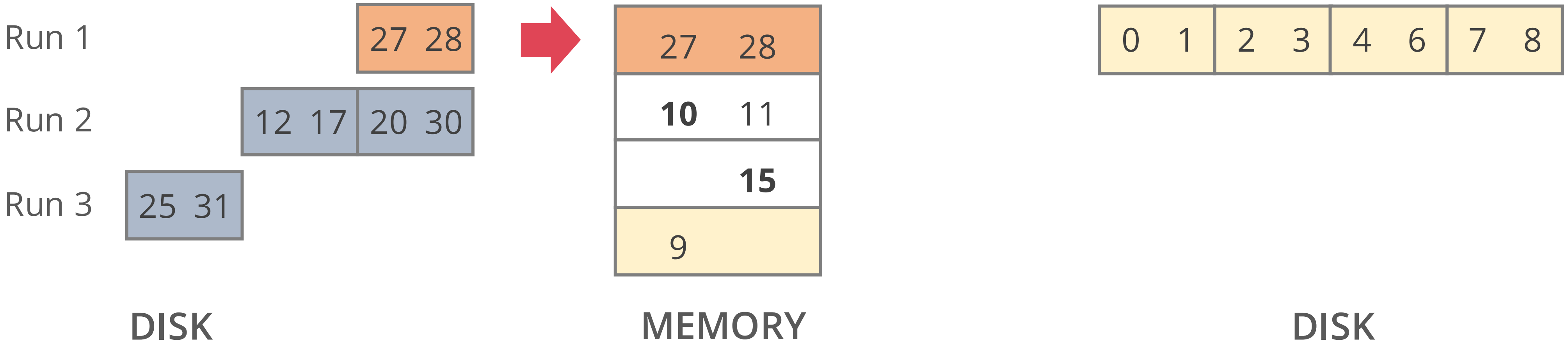
DISK

I/O Total (so far): 30

General External Merge Sort

Pass 1

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

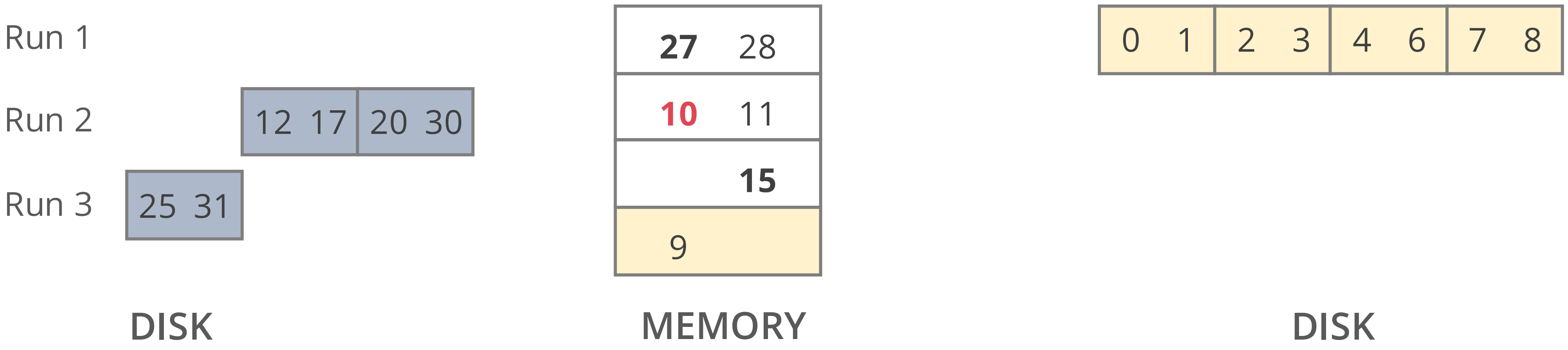


I/O Total (so far): 31

General External Merge Sort

Pass 1

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



Run 1

Run 2 | 12 17 | 20 30

Run 3 | 25 31

DISK

27 28

10 11

15

9

MEMORY

0 1 | 2 3 | 4 6 | 7 8

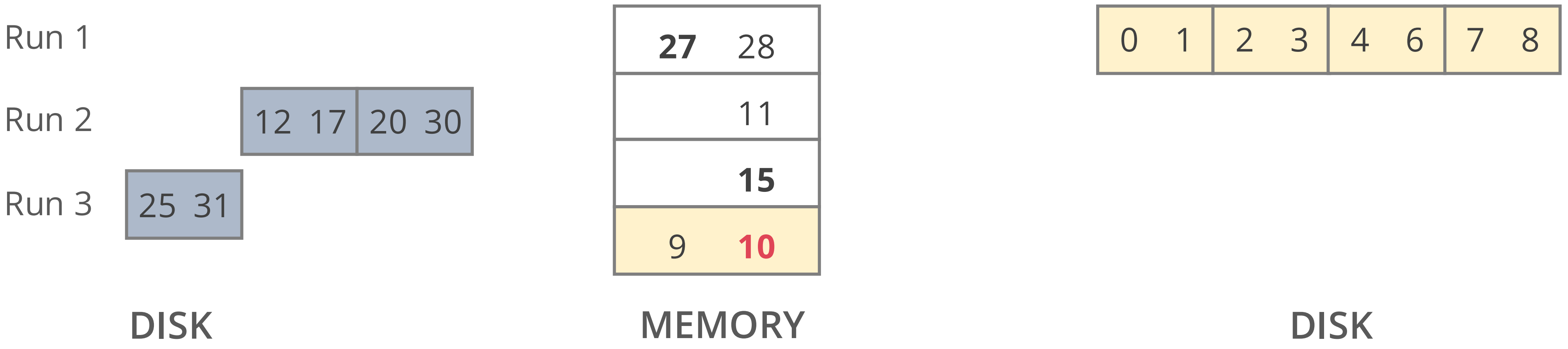
DISK

I/O Total (so far): 31

General External Merge Sort

Pass 1

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

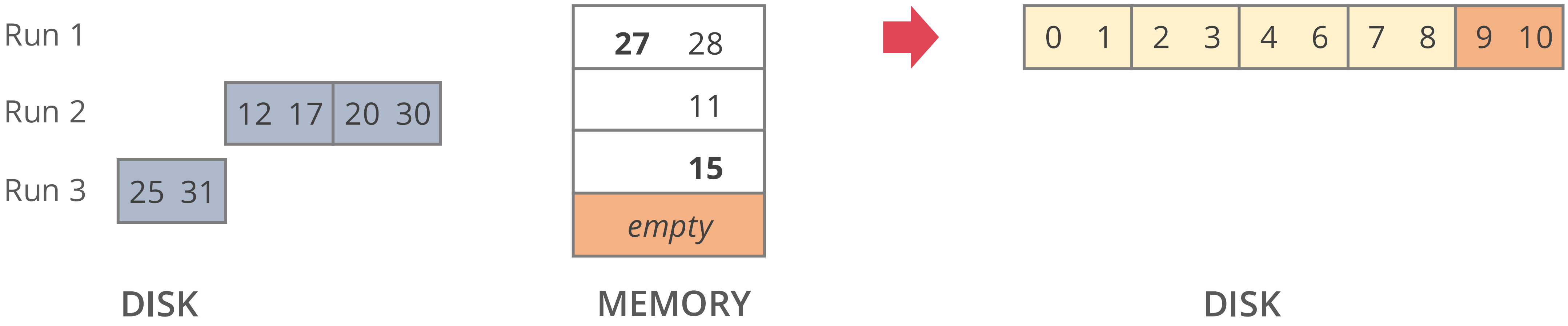


I/O Total (so far): 31

General External Merge Sort

Pass 1

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



Run 1

Run 2: 12 17 | 20 30

Run 3: 25 31

DISK

27 28

11

15

empty

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

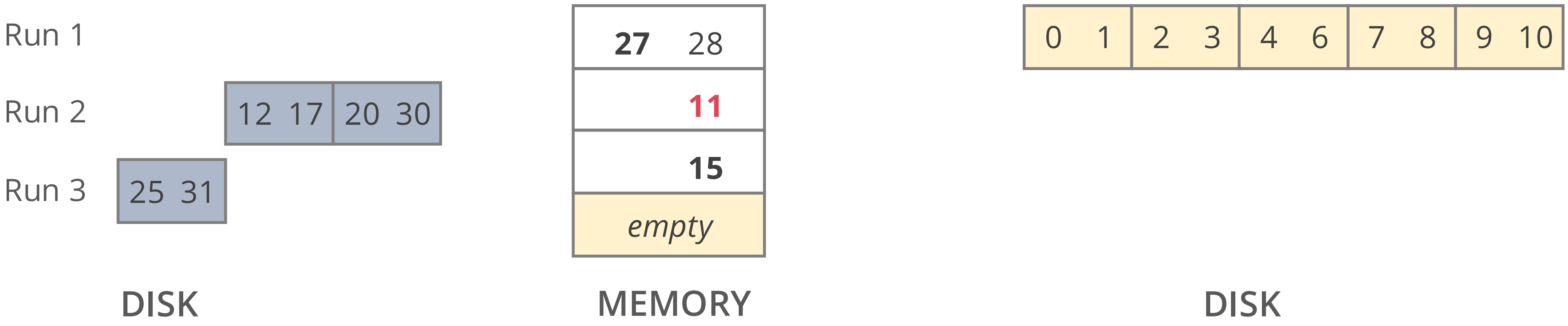
DISK

I/O Total (so far): 32

General External Merge Sort

Pass 1

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

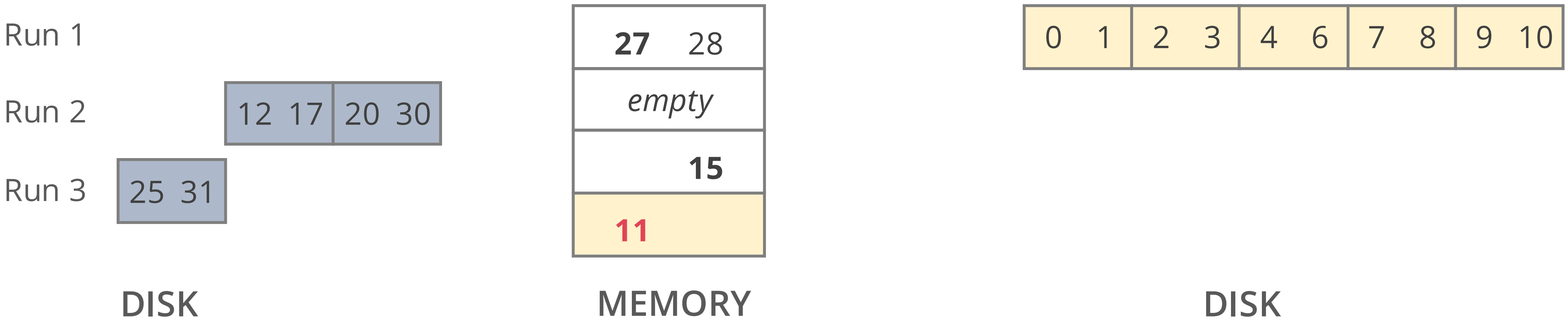


I/O Total (so far): 32

General External Merge Sort

Pass 1

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

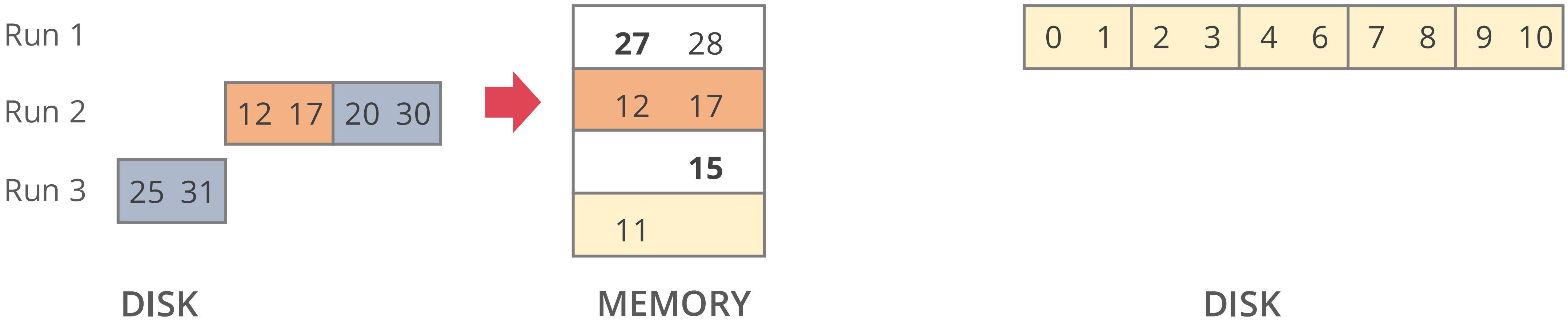


I/O Total (so far): 32

GENERAL EXTERNAL MERGE SORT

Pass 1

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

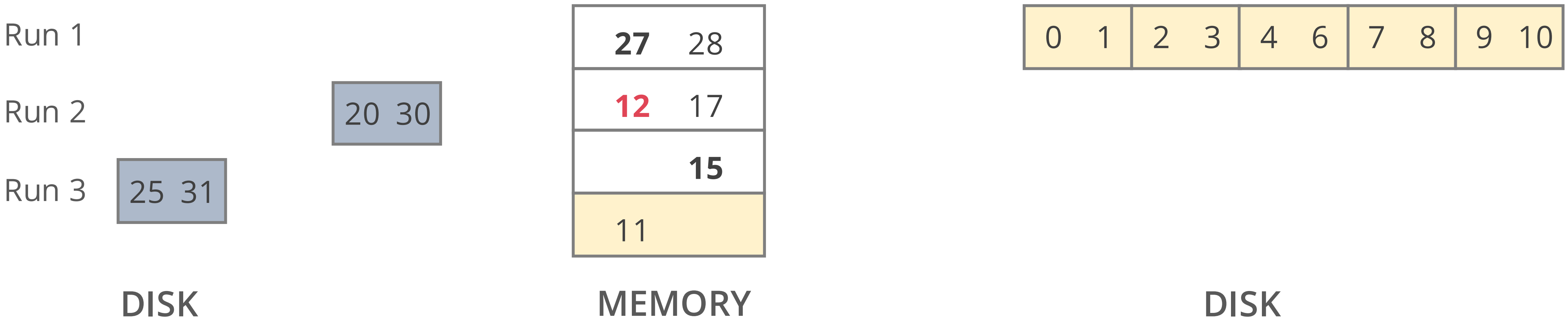


I/O Total (so far): 33

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

20 30

25 31

DISK

27 28

12 17

15

11

MEMORY

0 1 2 3 4 6 7 8 9 10

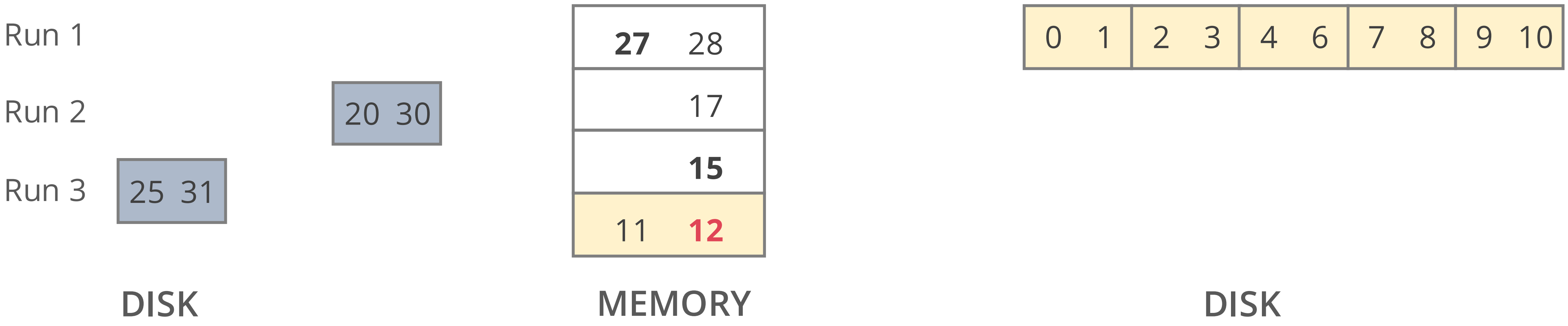
DISK

I/O Total (so far): 33

General External Merge Sort

Pass 1

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

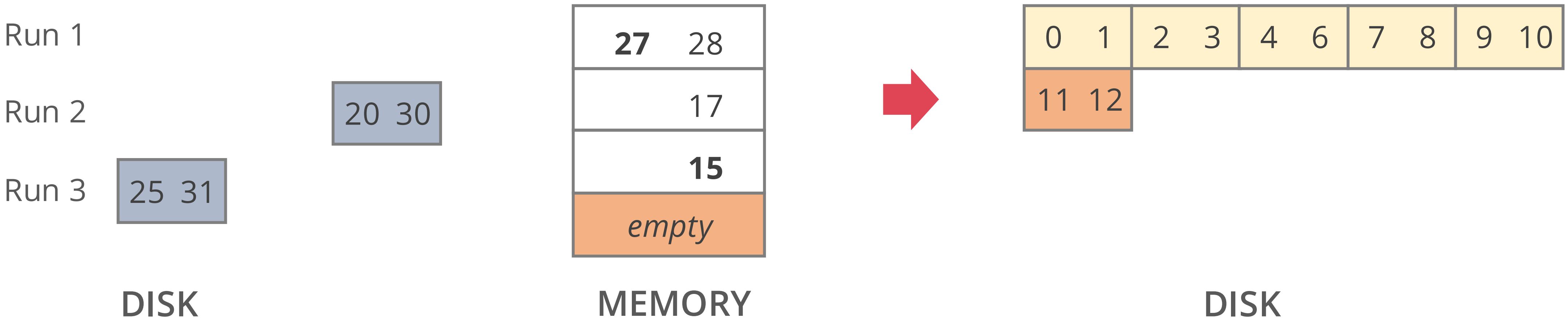


I/O Total (so far): 33

General External Merge Sort

Pass 1

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

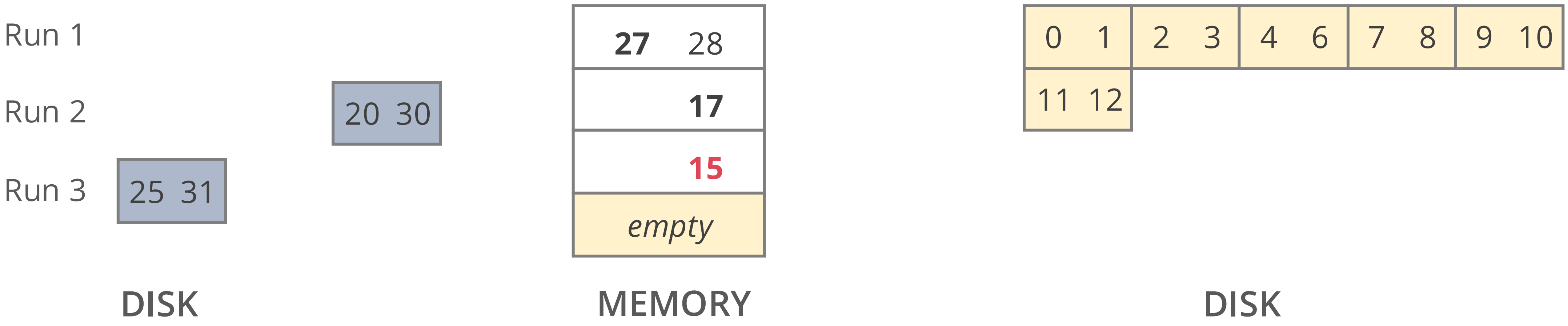


I/O Total (so far): 34

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

25 31

20 30

DISK

27 28

17

15

empty

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12

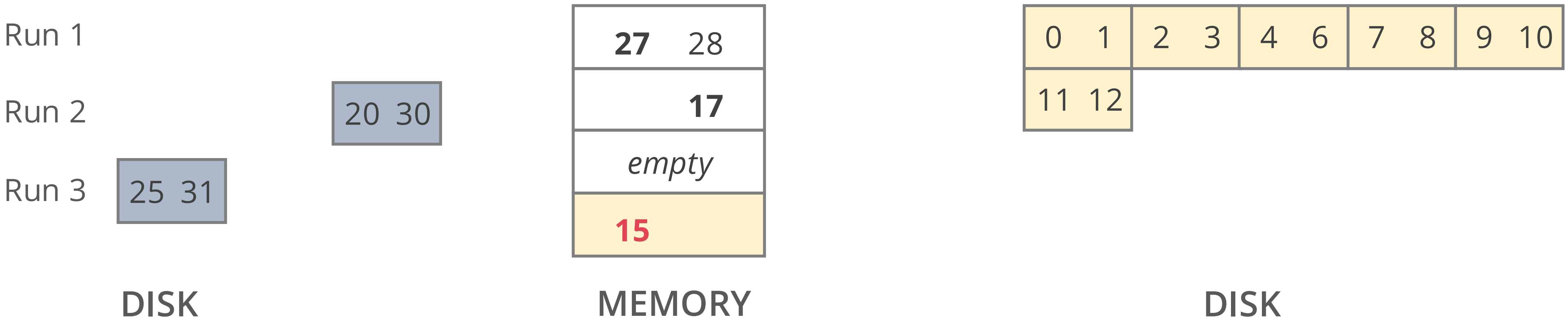
DISK

I/O Total (so far): 34

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

25 31

20 30

DISK

27 28

17

empty

15

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12

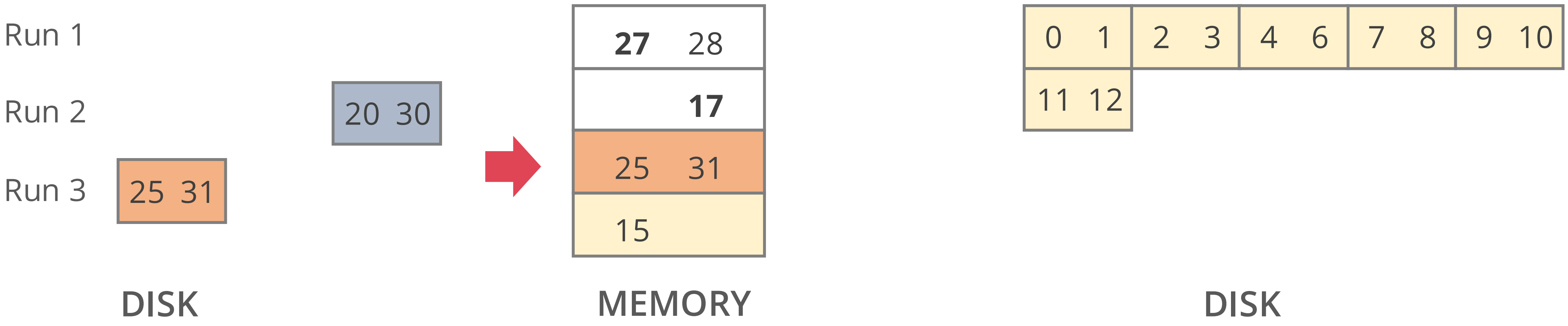
DISK

I/O Total (so far): 34

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

25 31

20 30

DISK

27 28

17

25 31

15

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12

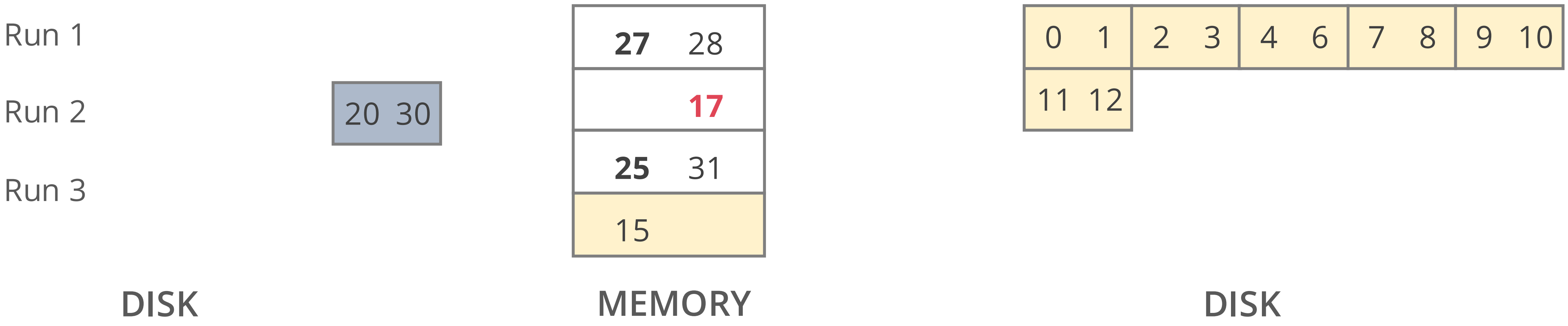
DISK

I/O Total (so far): 35

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

20 30

DISK

27 28

17

25 31

15

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12

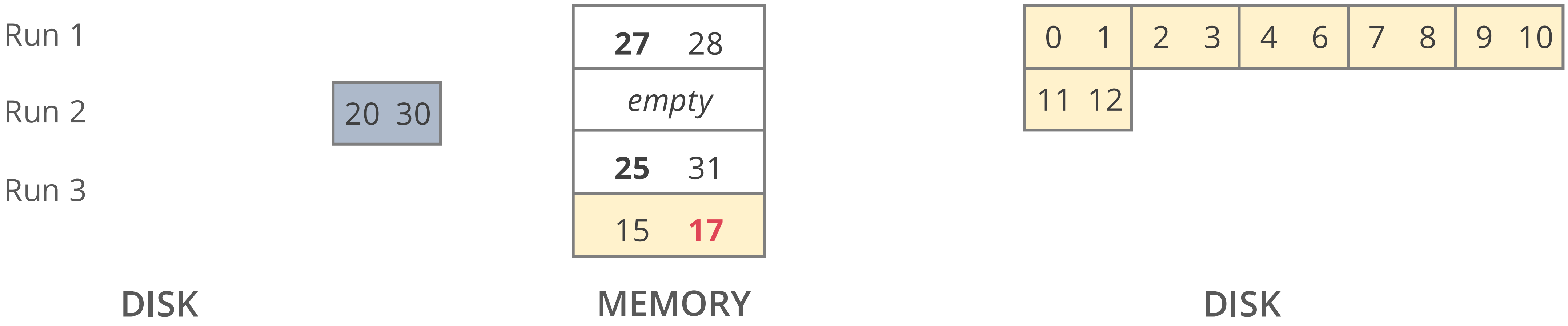
DISK

I/O Total (so far): 35

GENERAL EXTERNAL MERGE SORT

Pass 1

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



Run 1

Run 2

Run 3

20 30

27 28

empty

25 31

15 **17**

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12

DISK

MEMORY

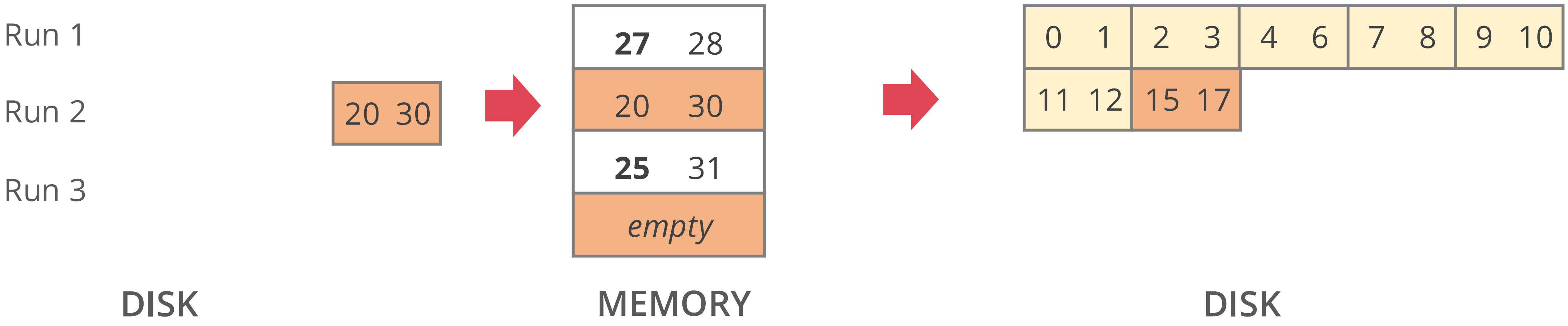
DISK

I/O Total (so far): 35

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

20 30

DISK

27 28

20 30

25 31

empty

MEMORY

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

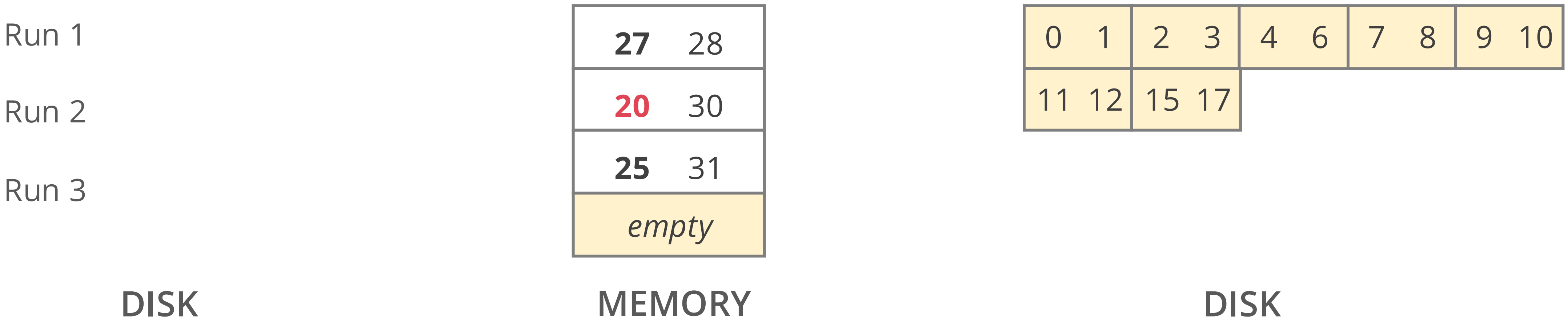
DISK

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

20 30

25 31

empty

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12 | 15 17

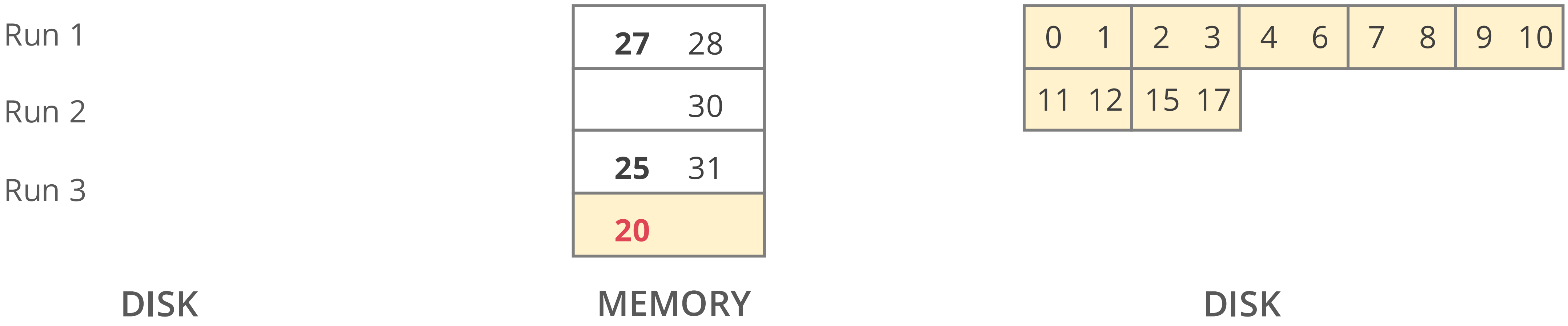
DISK

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

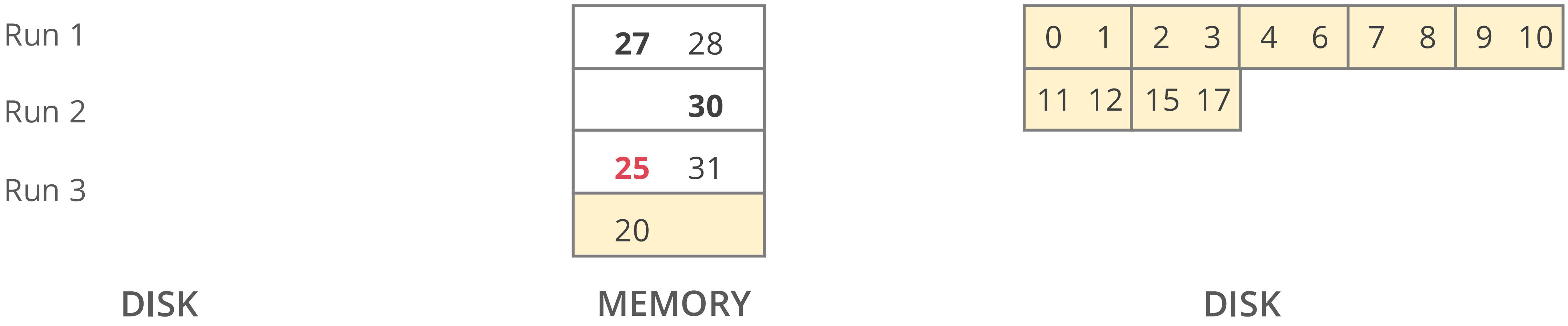
DISK

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

27 28

30

25 31

20

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12 | 15 17

DISK

MEMORY

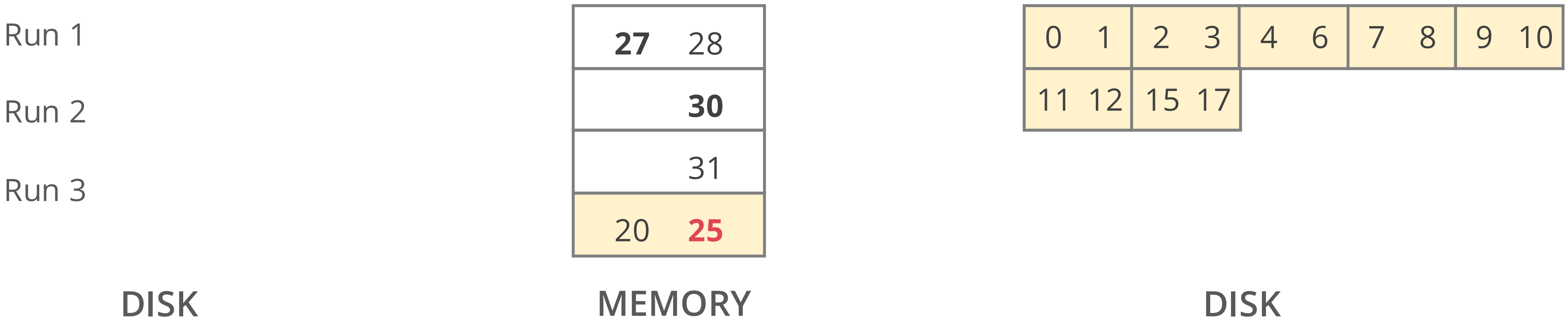
DISK

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

31

20 **25**

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12 | 15 17

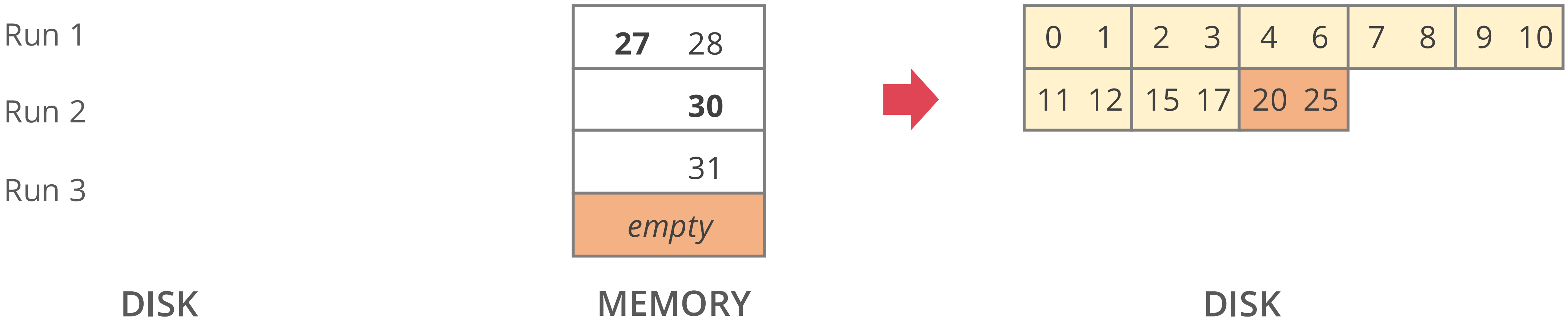
DISK

I/O Total (so far): 37

General External Merge Sort

Pass 1

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

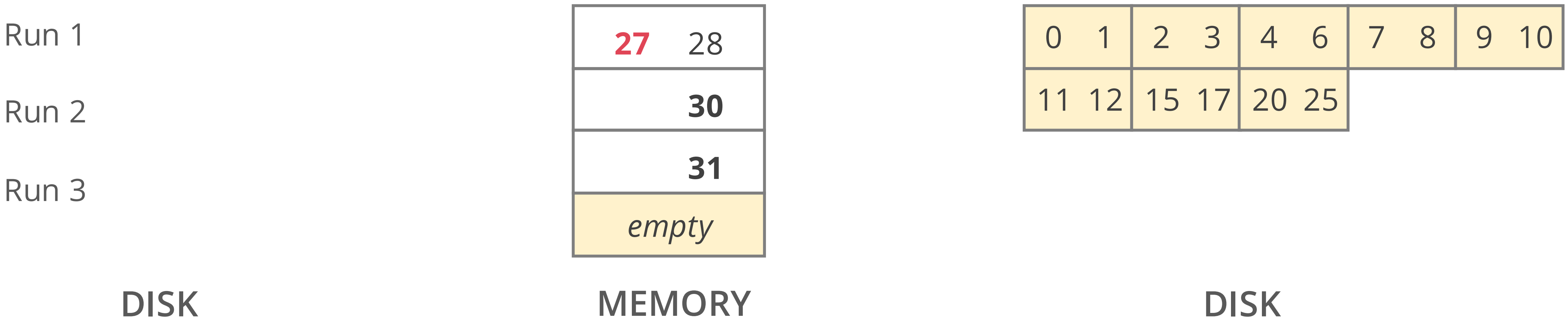


I/O Total (so far): 38

GENERAL EXTERNAL MERGE SORT

Pass 1

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



27 28

30

31

empty

0 1 | 2 3 | 4 6 | 7 8 | 9 10

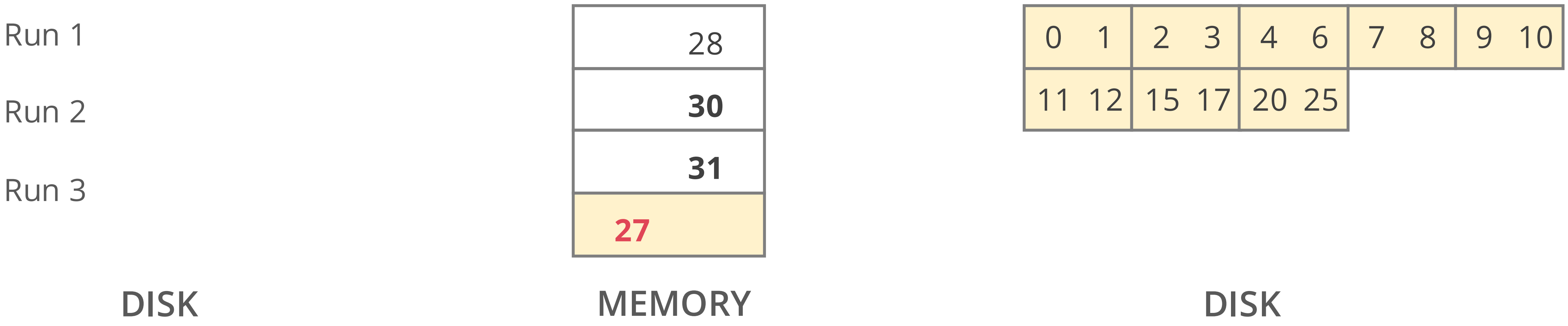
11 12 | 15 17 | 20 25

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

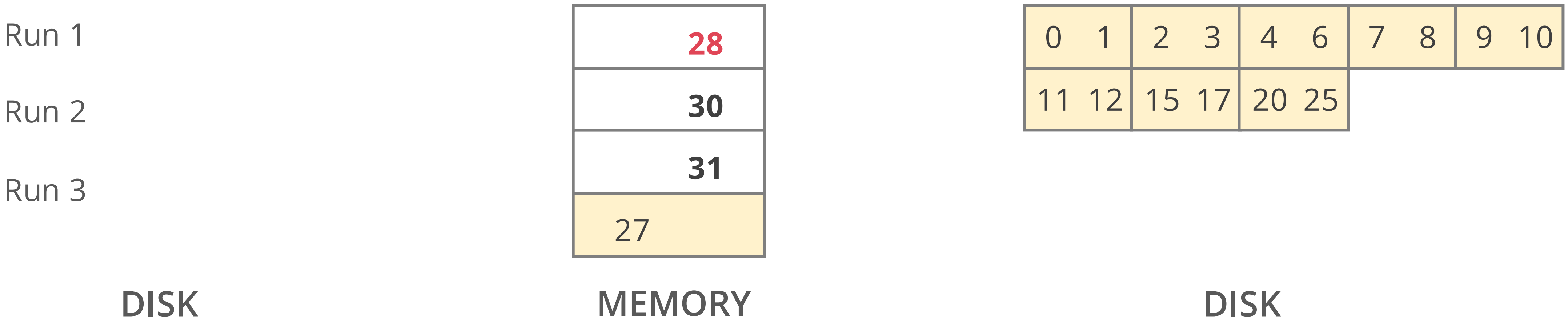
DISK

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

28

30

31

27

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12 | 15 17 | 20 25

DISK

MEMORY

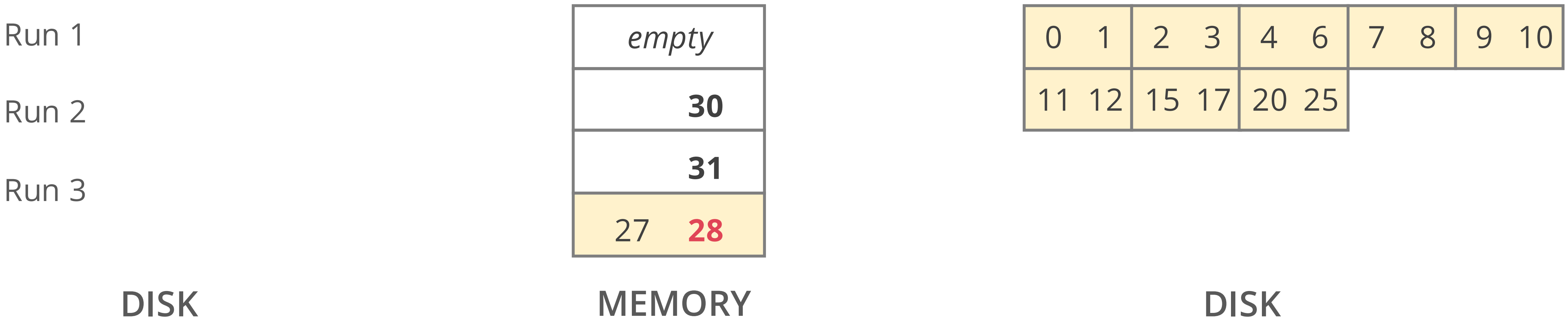
DISK

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

empty
30
31
27 **28**

MEMORY

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

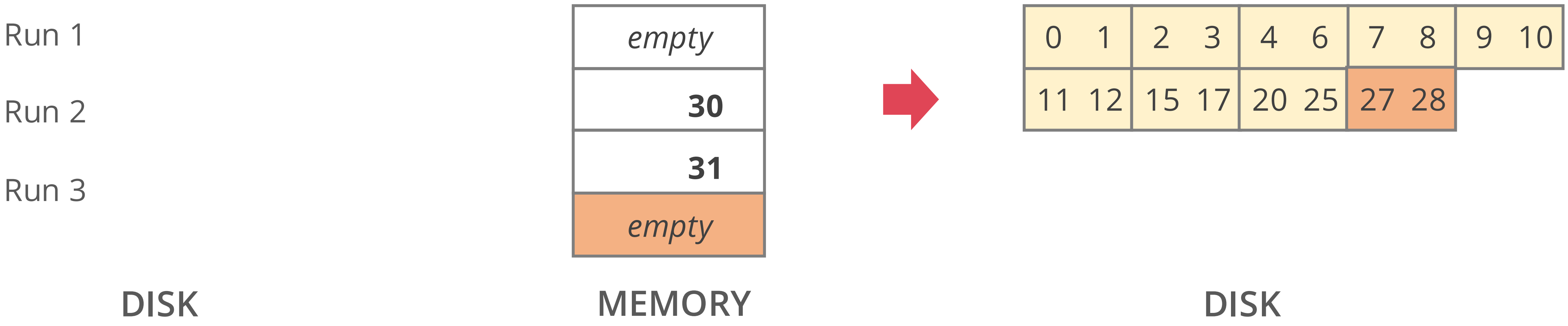
DISK

I/O Total (so far): 38

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

DISK

MEMORY
empty
30
31
empty

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

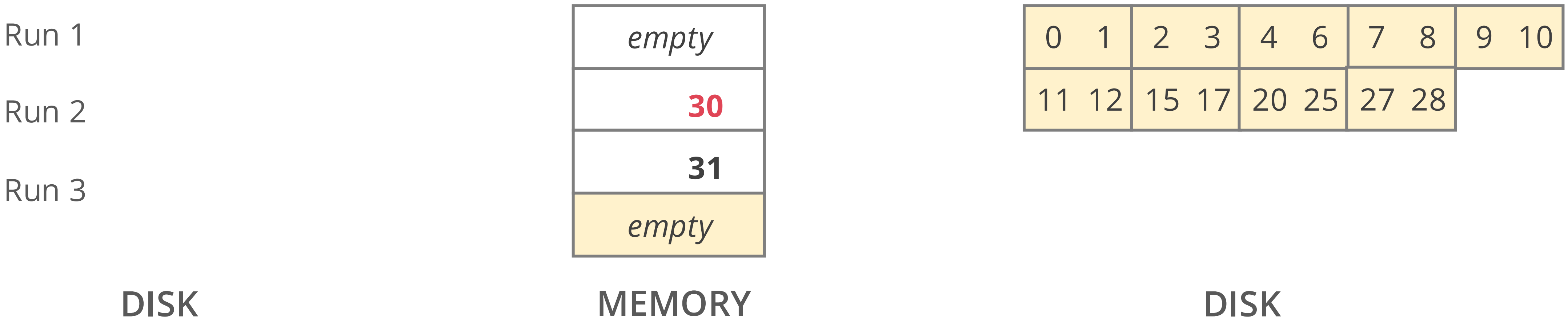
DISK

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

30

31

empty

MEMORY

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

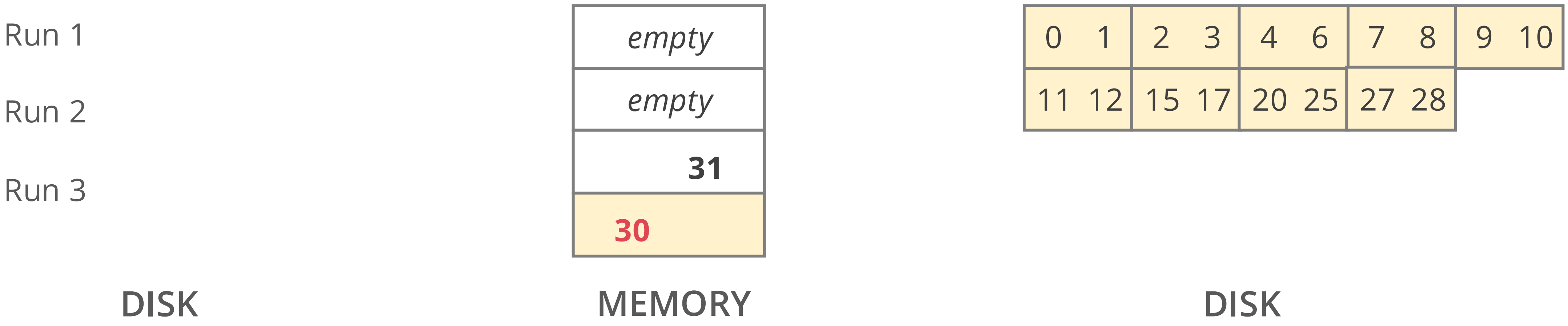
DISK

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	

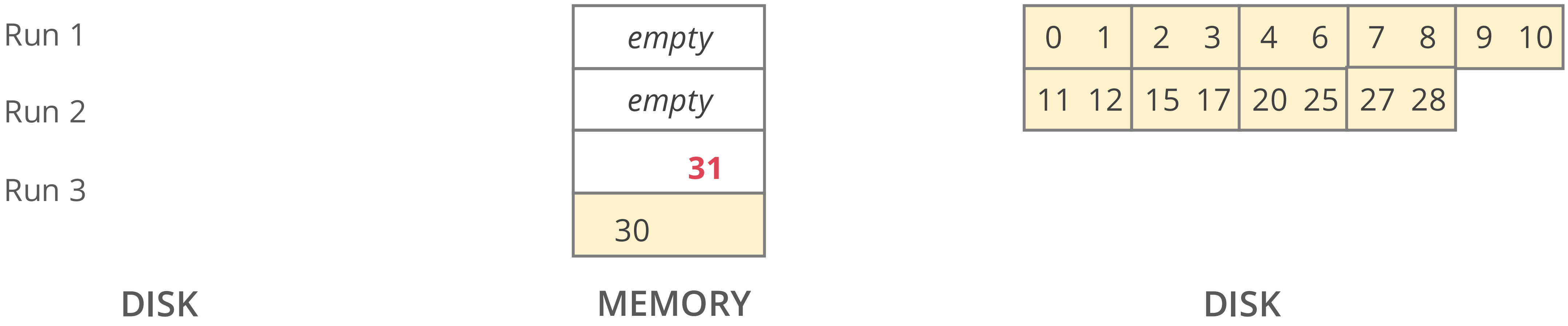
DISK

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

MEMORY
empty
empty
31
30

MEMORY

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

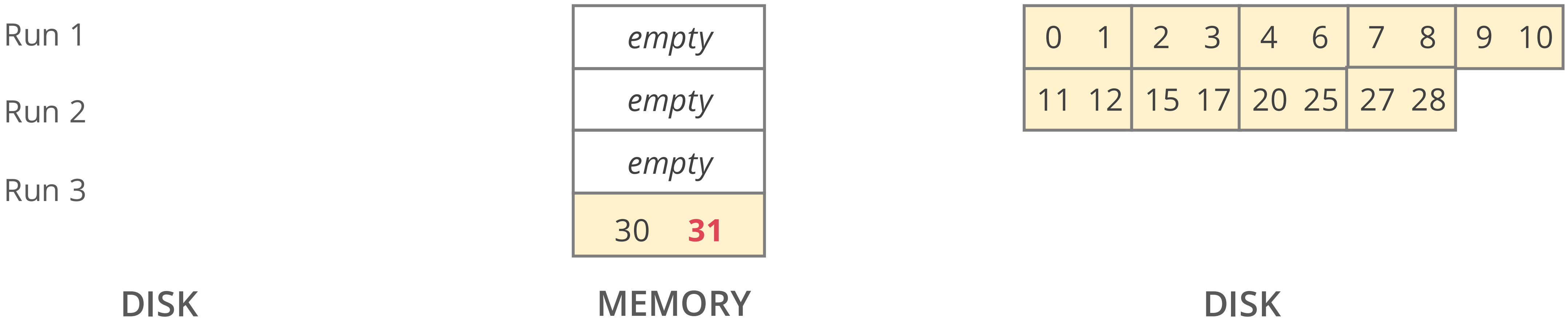
DISK

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

empty

empty

empty

30 31

MEMORY

0 1 | 2 3 | 4 6 | 7 8 | 9 10

11 12 | 15 17 | 20 25 | 27 28

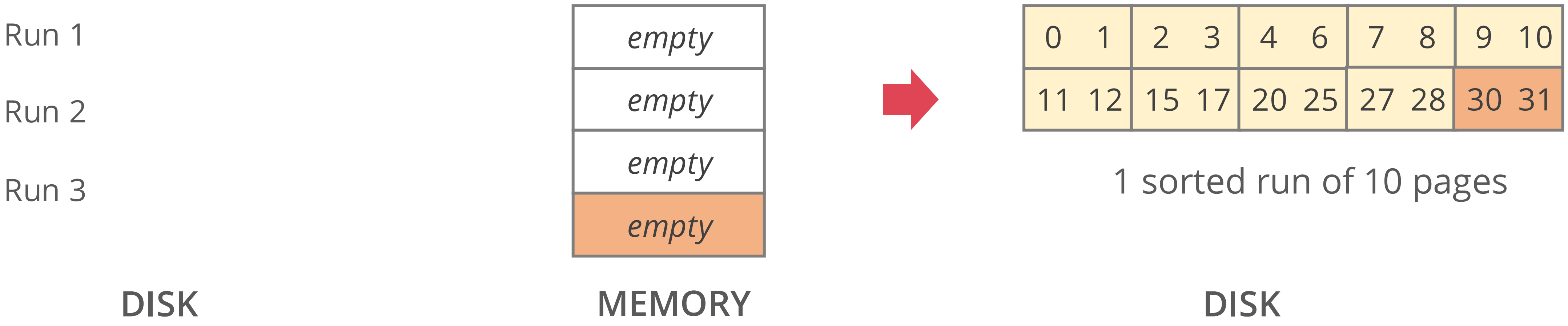
DISK

I/O Total (so far): 39

General External Merge Sort

Pass 1

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



Run 1

Run 2

Run 3

DISK

empty

empty

empty

empty

MEMORY

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

DISK

I/O Total: 40

Sanity Check

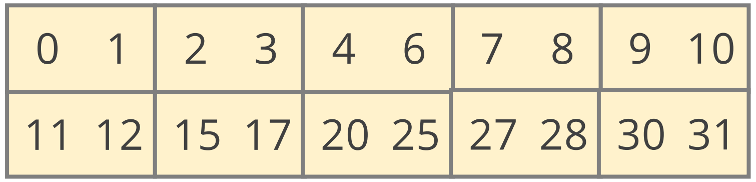
$N = 10, B = 4$

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

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

$= 20 * (1 + 1)$

$=$ 40 I/Os ✓



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

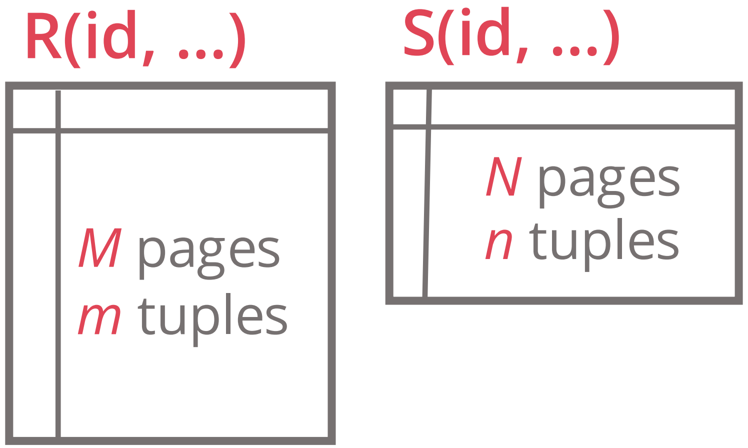
I/O Total: 40

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!

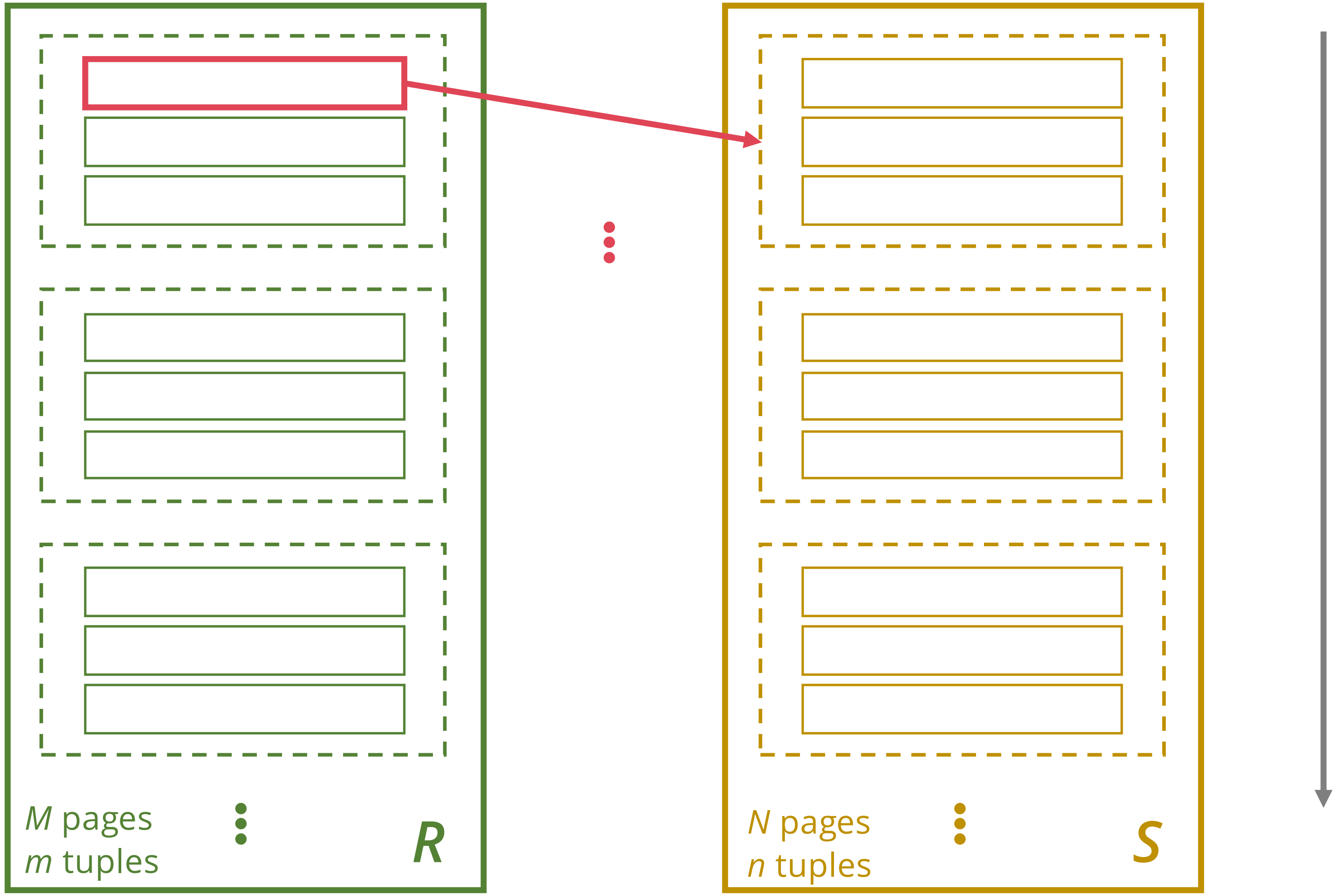
Simple Nested Loops Join (SNLJ)

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)

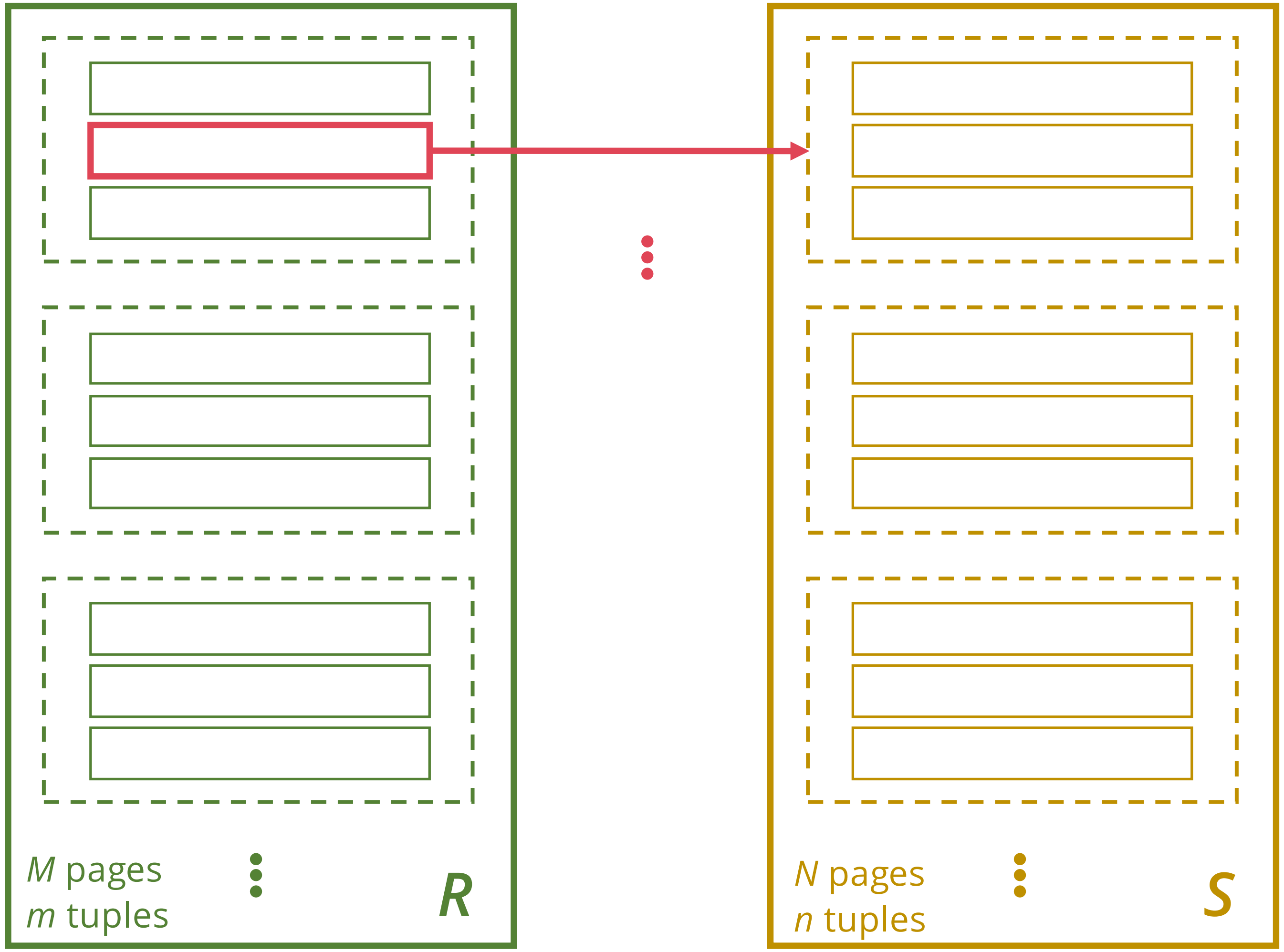


First iteration of outer loop…

Compare with all tuples in ***S***

… and add matches to result

Simple Nested Loops Join (SNLJ)

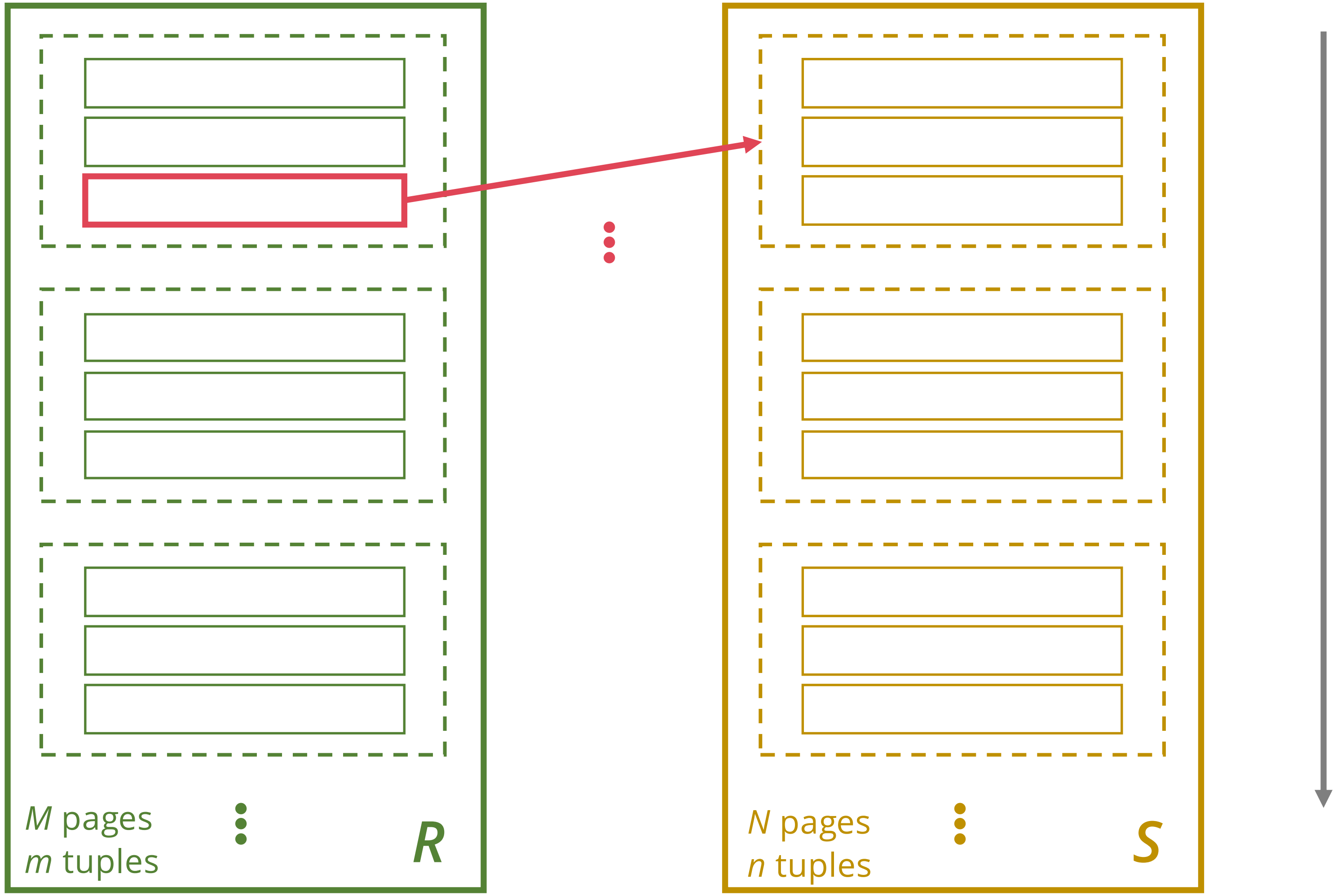


Second iteration of outer loop…

Compare with all tuples in ***S***

… and add matches to result

Simple Nested Loops Join (SNLJ)



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 θ(r,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 $s$
```

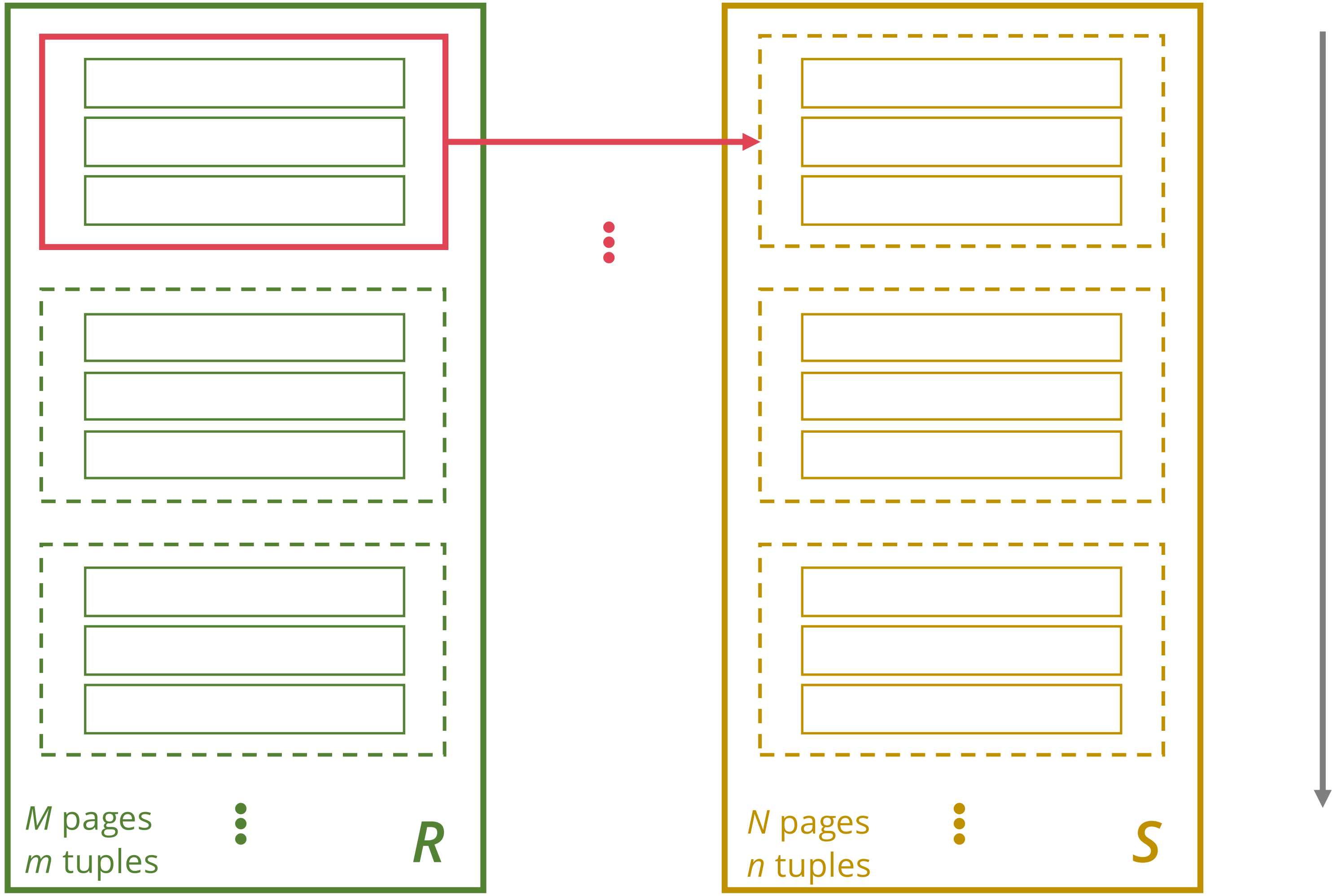
Page Nested Loops Join (PNLJ)

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

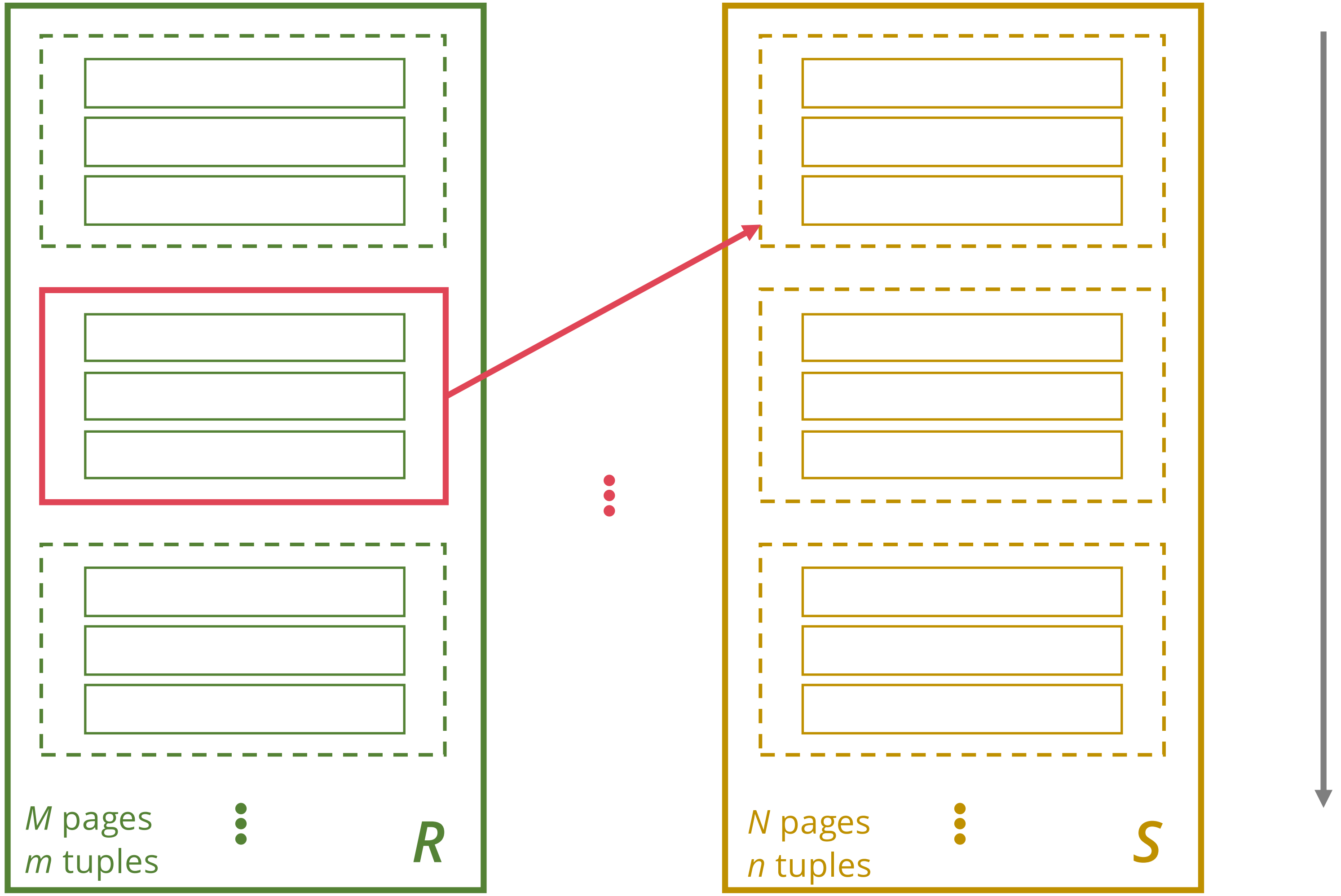
PAGE NESTED LOOPS JOIN (PNLJ)



First iteration of outer loop…

Compare with all tuples in ***S***
… and add matches to result

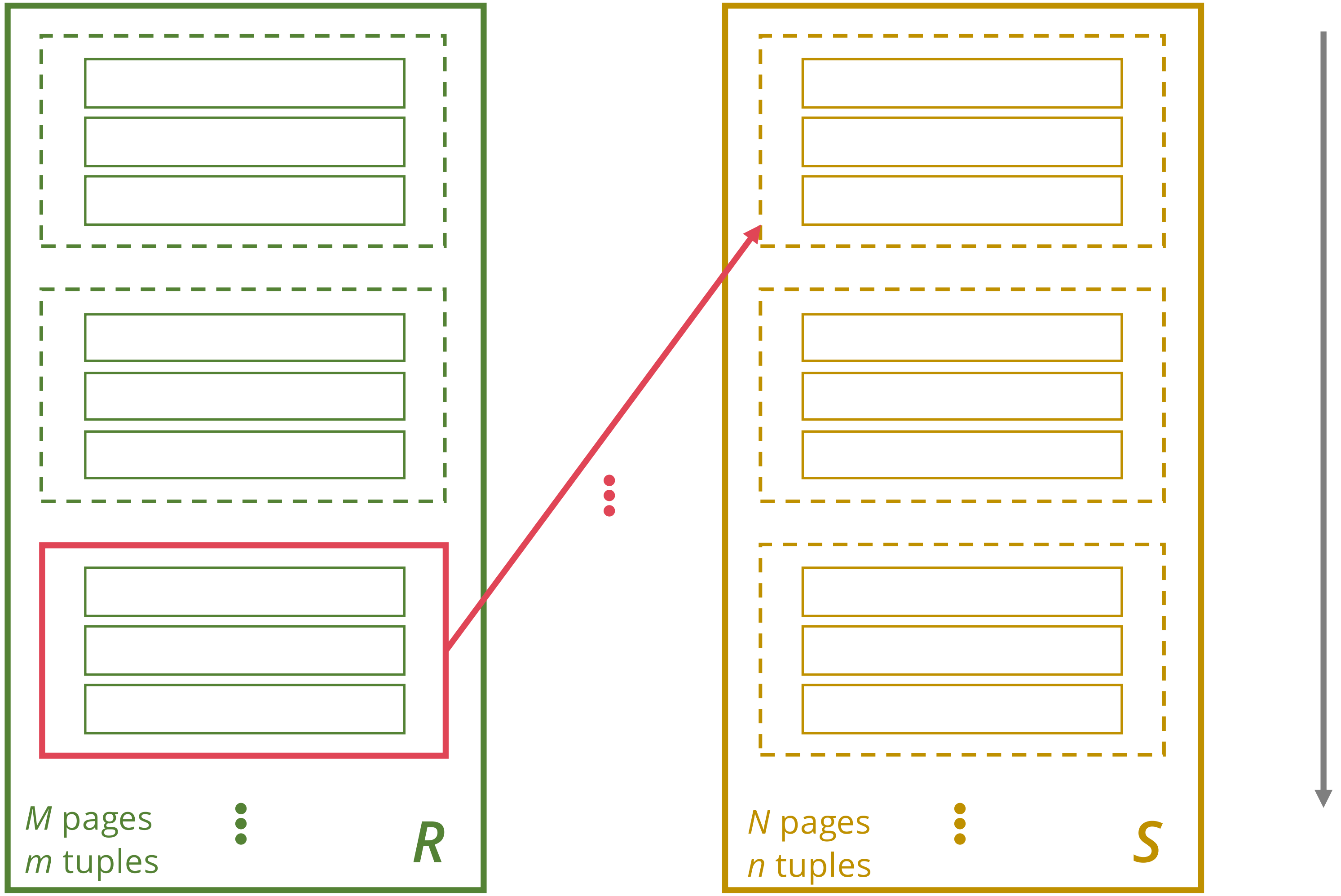
Page Nested Loops Join (PNLJ)



Second iteration of outer loop…

Compare with all tuples in ***S***
… and add matches to result

Page Nested Loops Join (PNLJ)



Cost = *M* + *M* · *N*

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

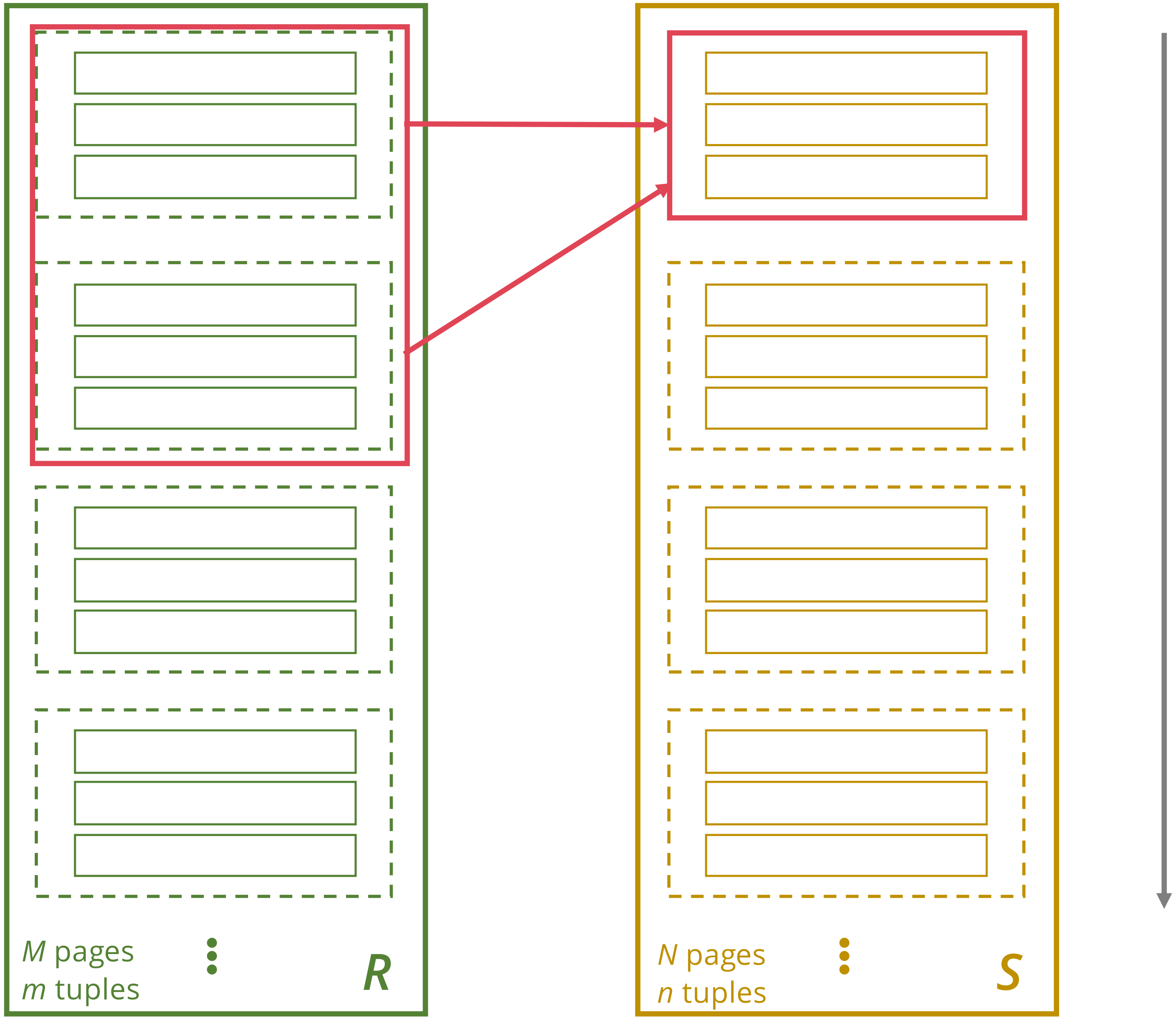
```
foreach page $P_R \in R$:
  foreach page $P_S \in S$:
    foreach tuple $r \in P_R$:
      foreach tuple $s \in P_S$:
        if $\theta(r,s)$: output $r$ joined with $s$
```

Block Nested Loops Join (PNLJ)

BNLJ

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

B = 4

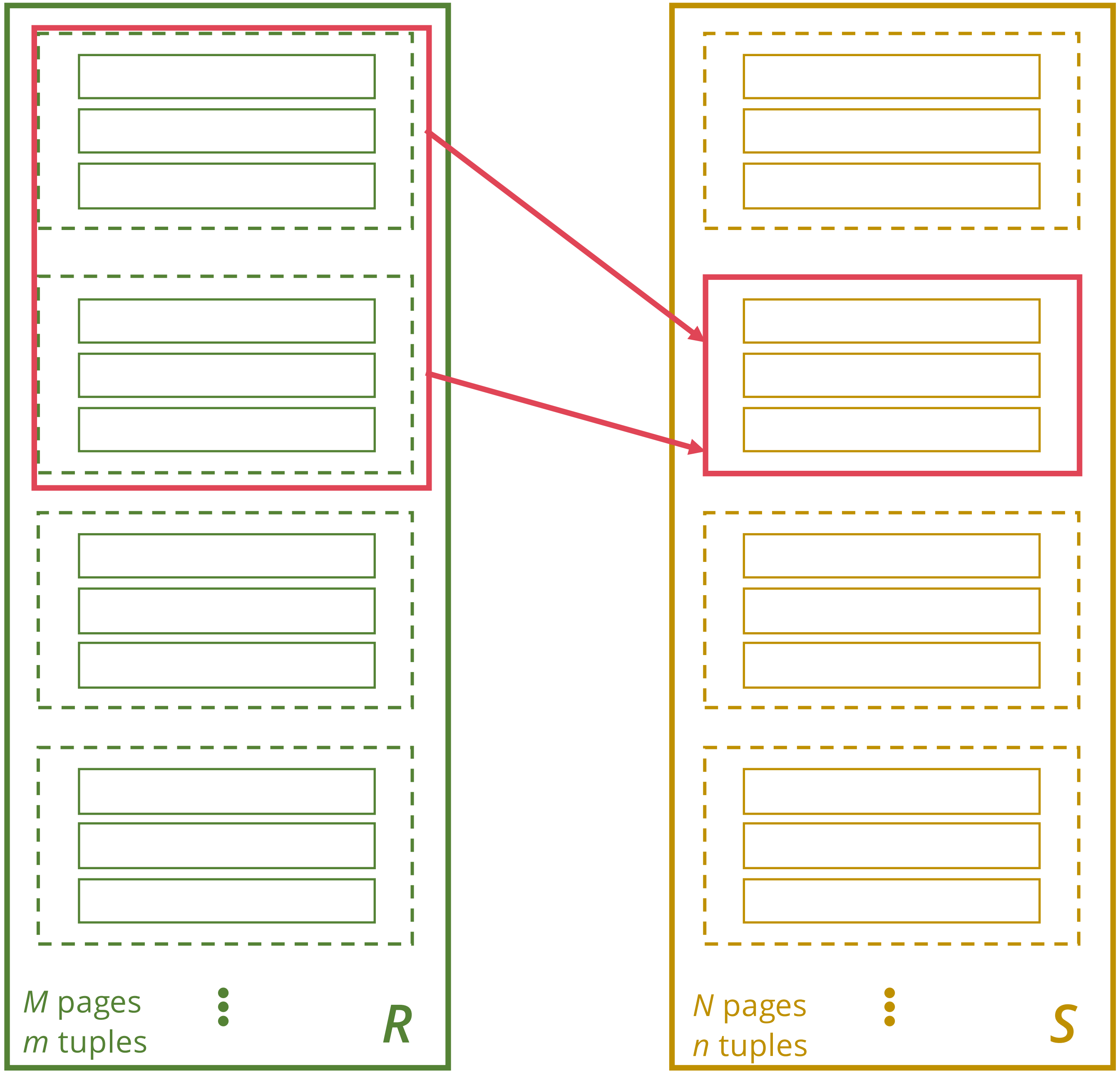


First iteration of outer loop…

Compare with all tuples in *S*

… and add matches to result

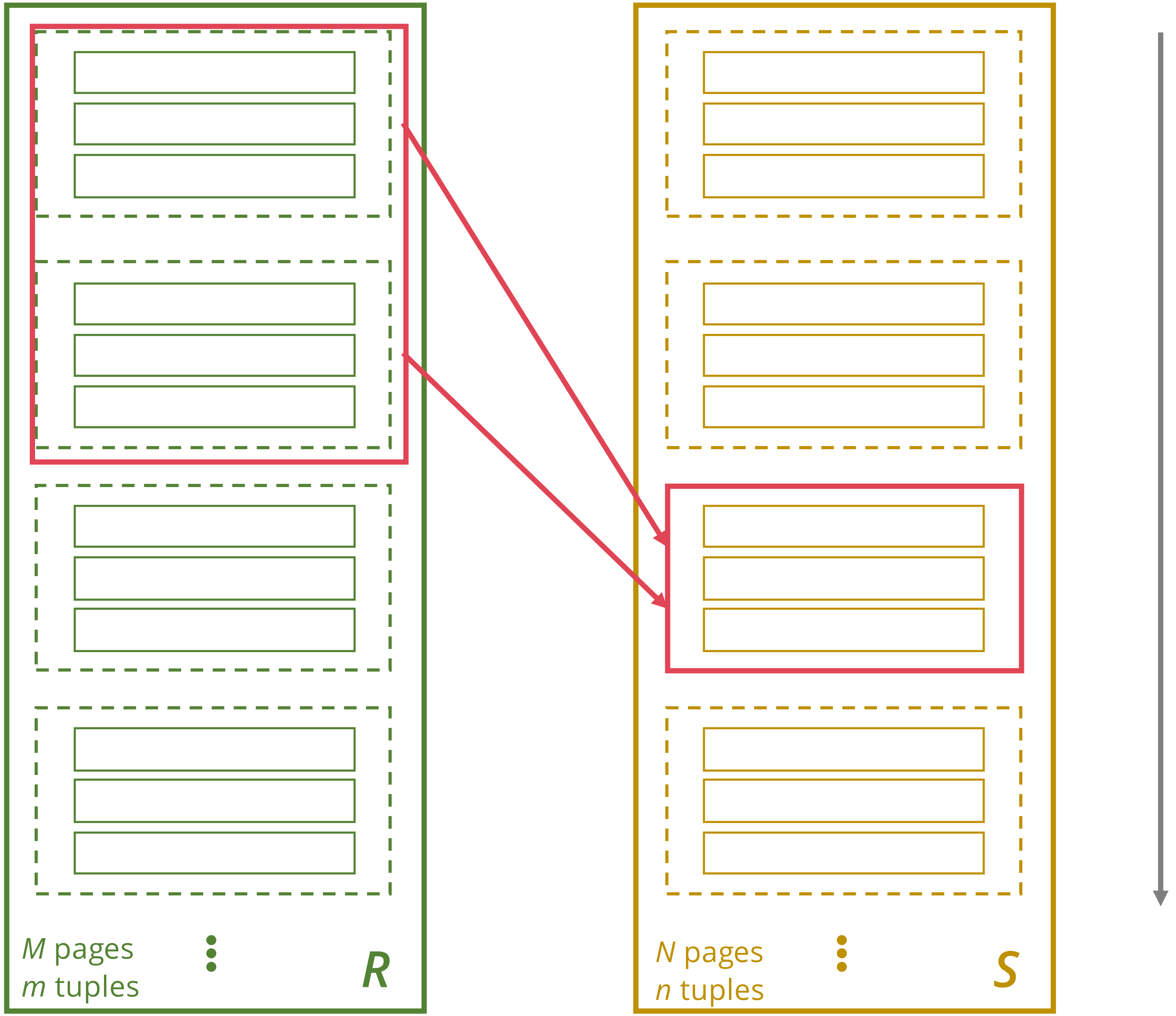
B = 4



First iteration of outer loop…

Compare with all tuples in ***S***
… and add matches to result

B = 4

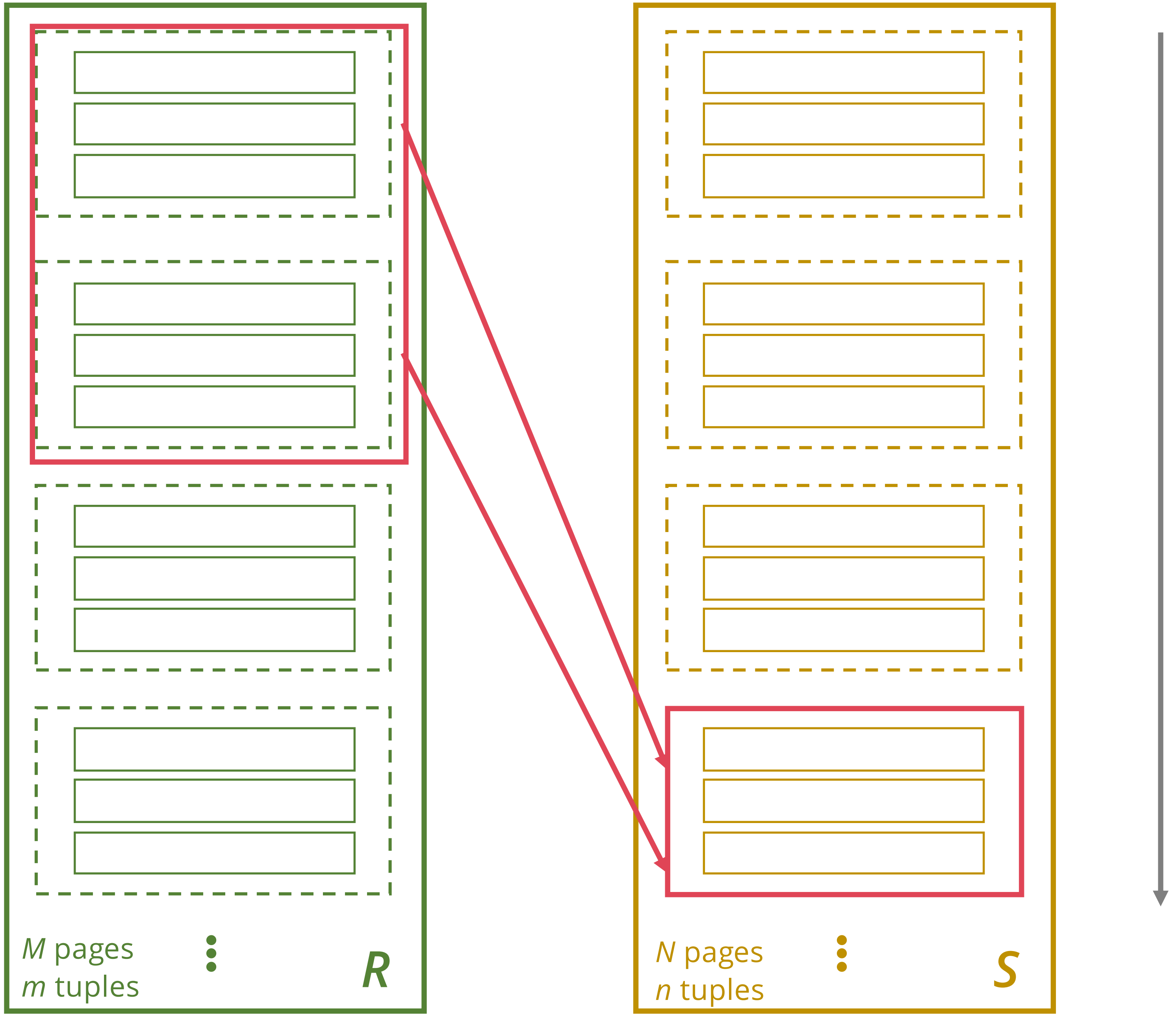


First iteration of outer loop…

Compare with all tuples in ***S***

… and add matches to result

B = 4

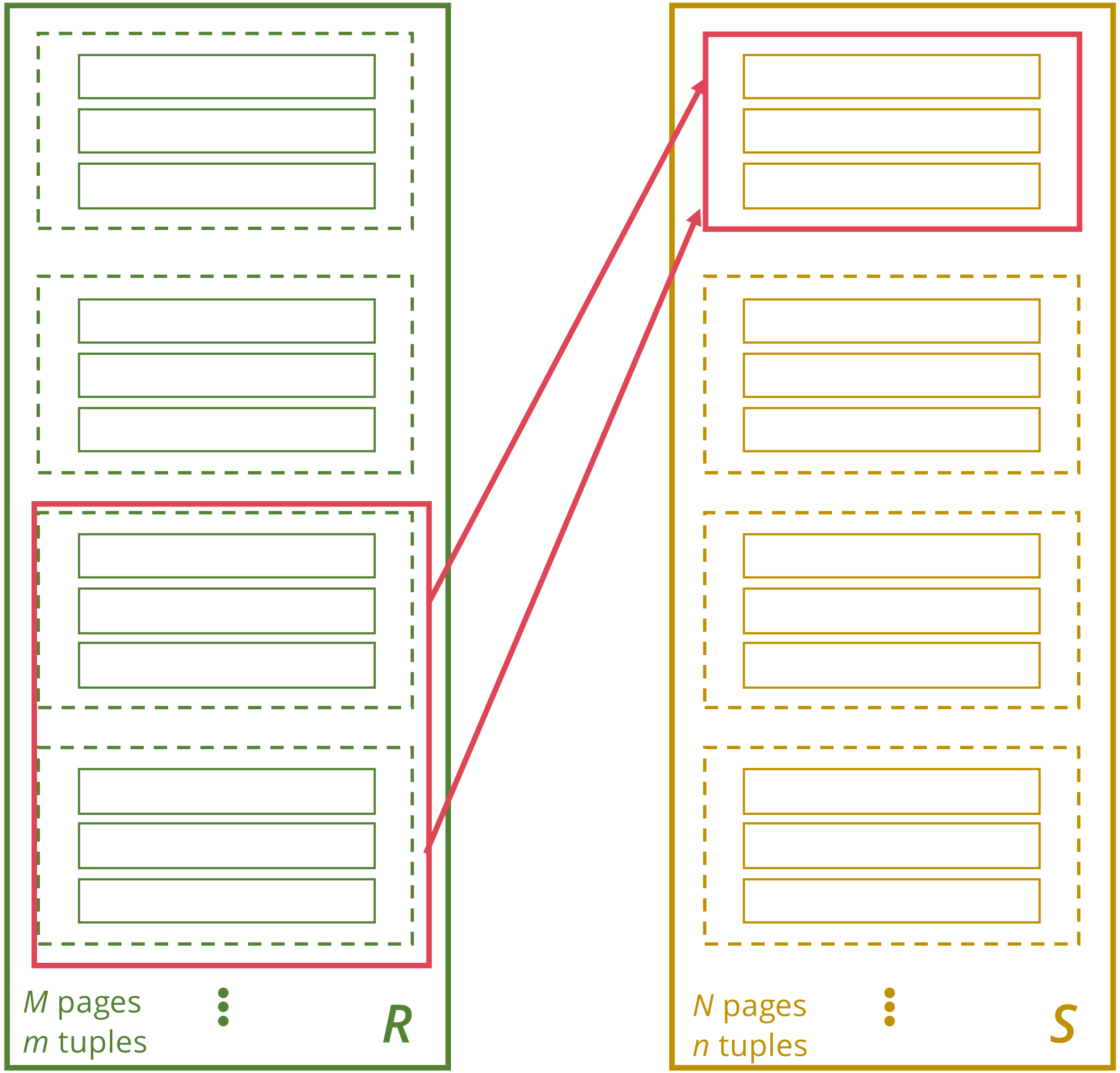


First iteration of outer loop…

Compare with all tuples in *S*

… and add matches to result

B = *4*

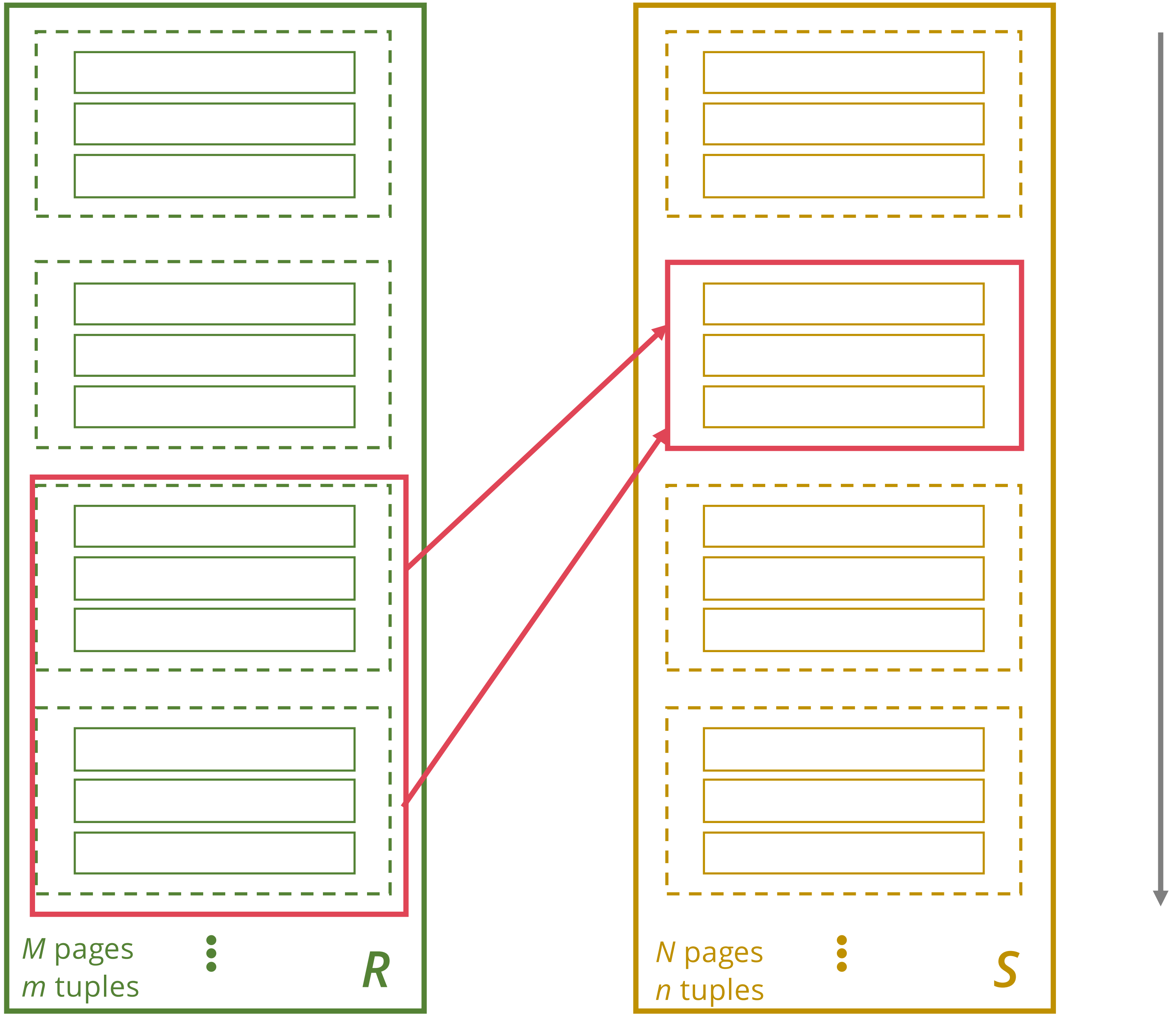


Second iteration of outer loop…

Compare with all tuples in *S*

… and add matches to result

B = *4*

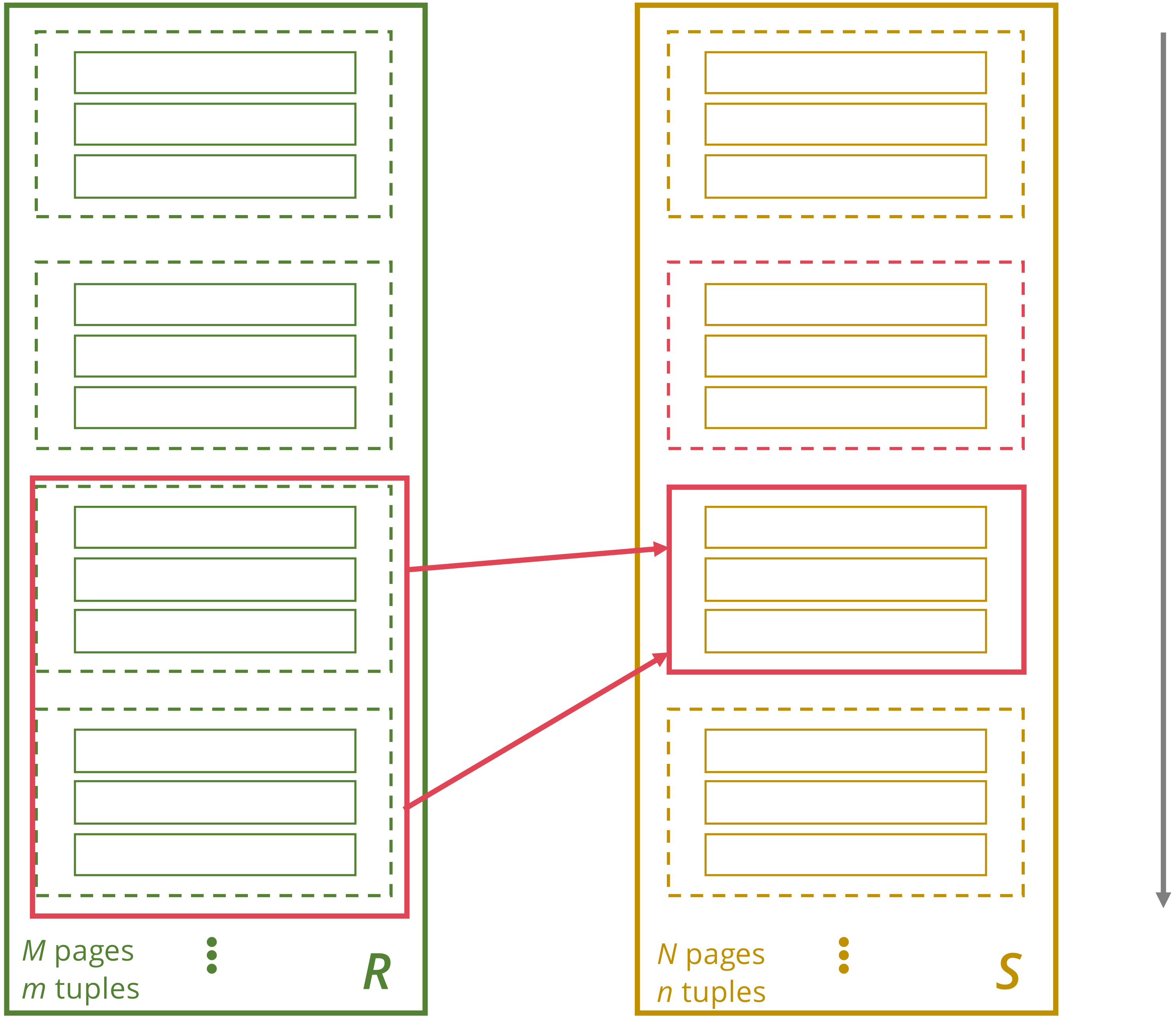


Second iteration of outer loop...

Compare with all tuples in *S*

... and add matches to result

$B = 4$

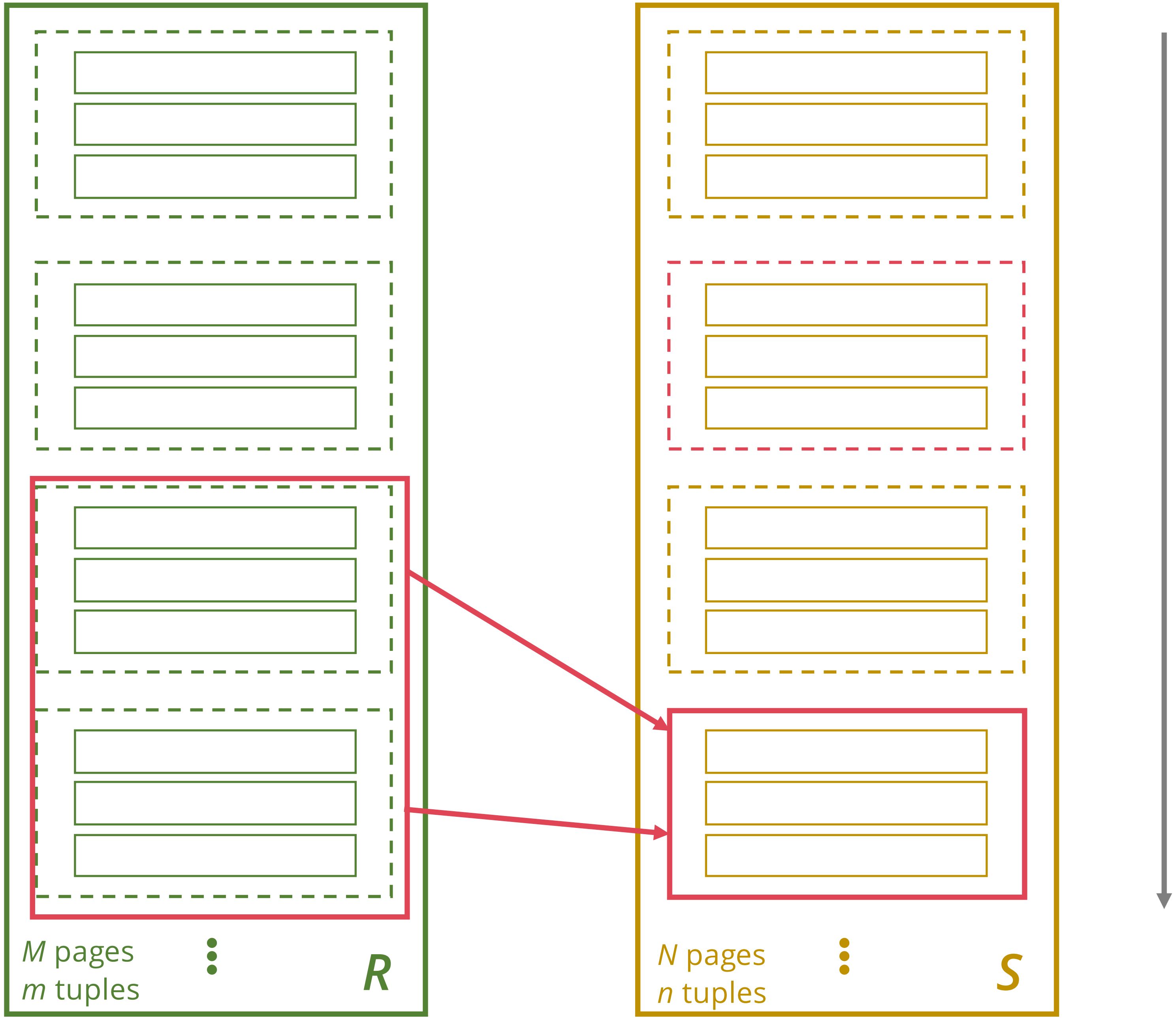


Second iteration of outer loop…

Compare with all tuples in *S*

… and add matches to result

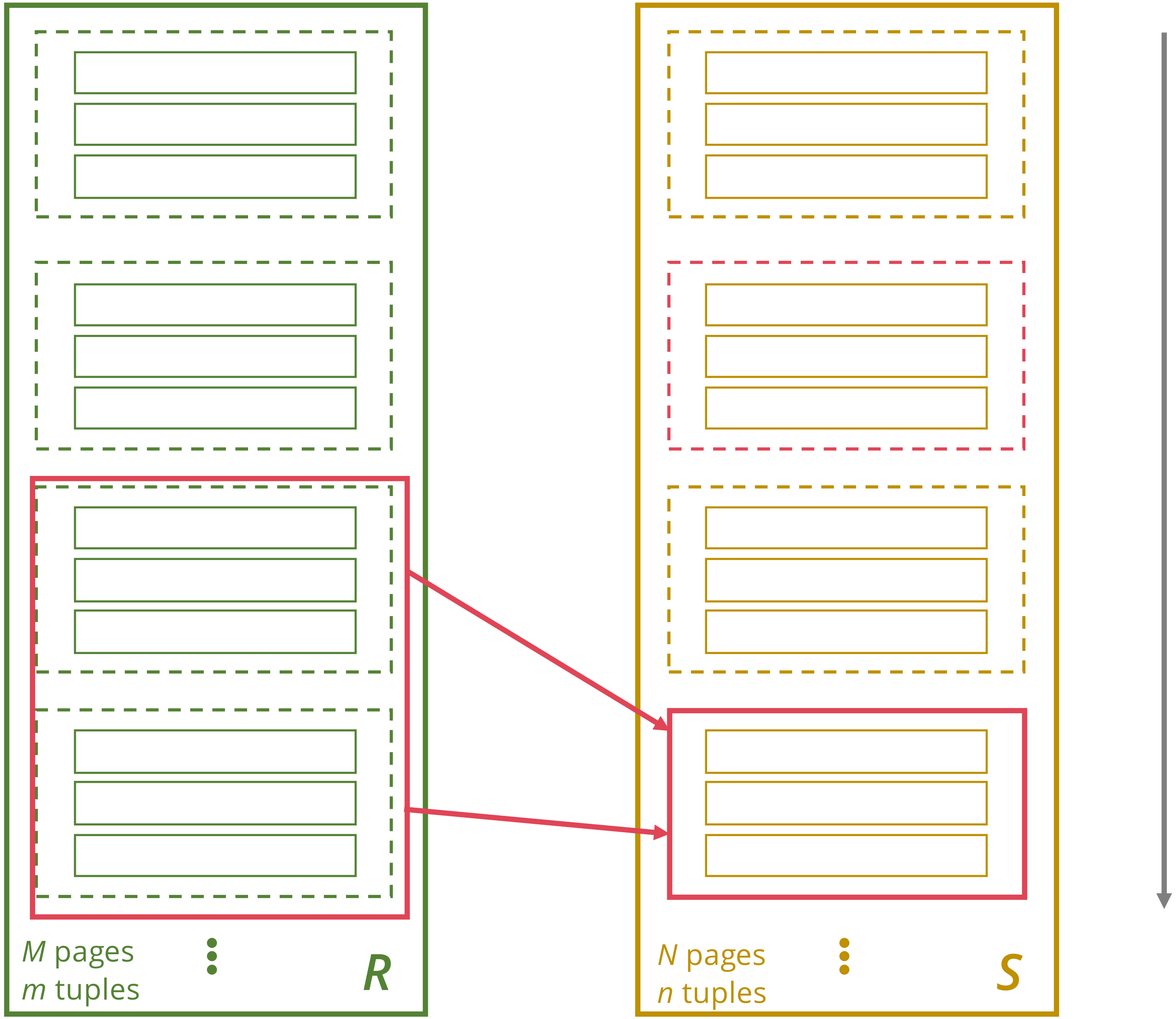
B = *4*



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

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 **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)

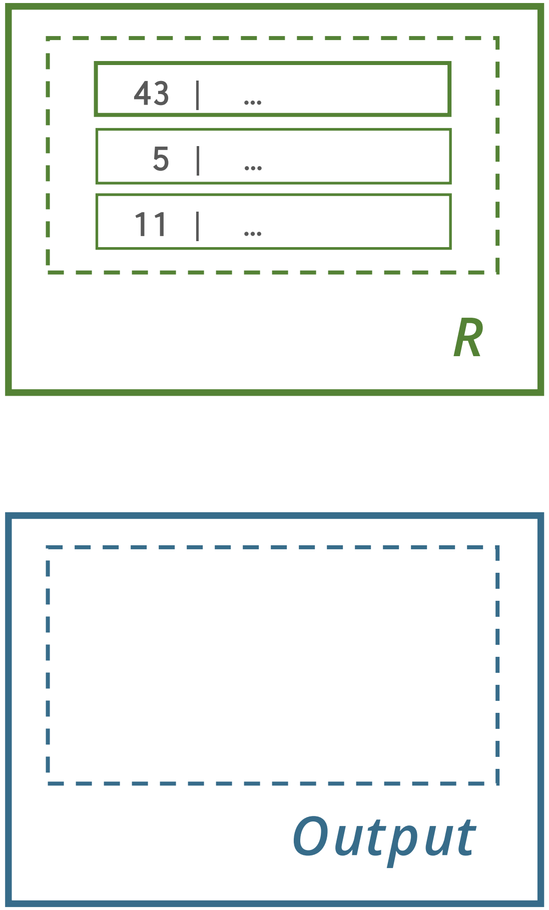
Cost = M + m * cost to find matching S tuples

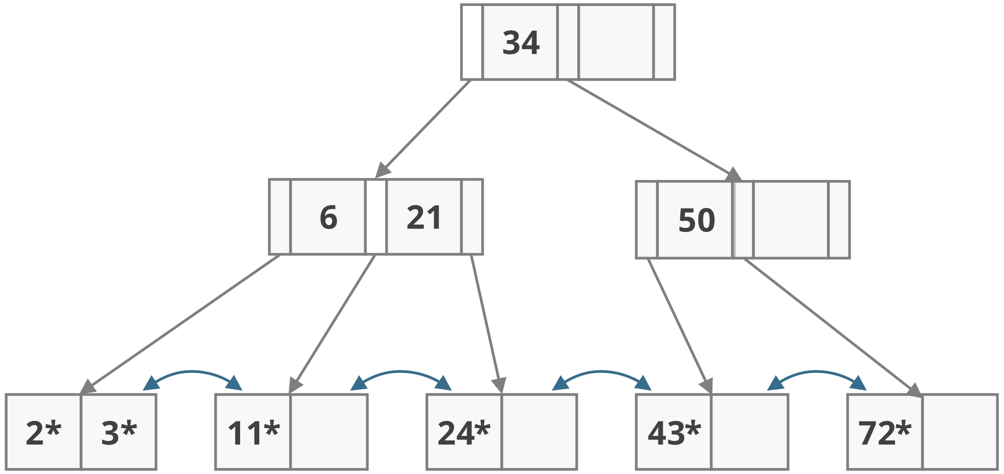
- M from scanning through R

If we have **no index**

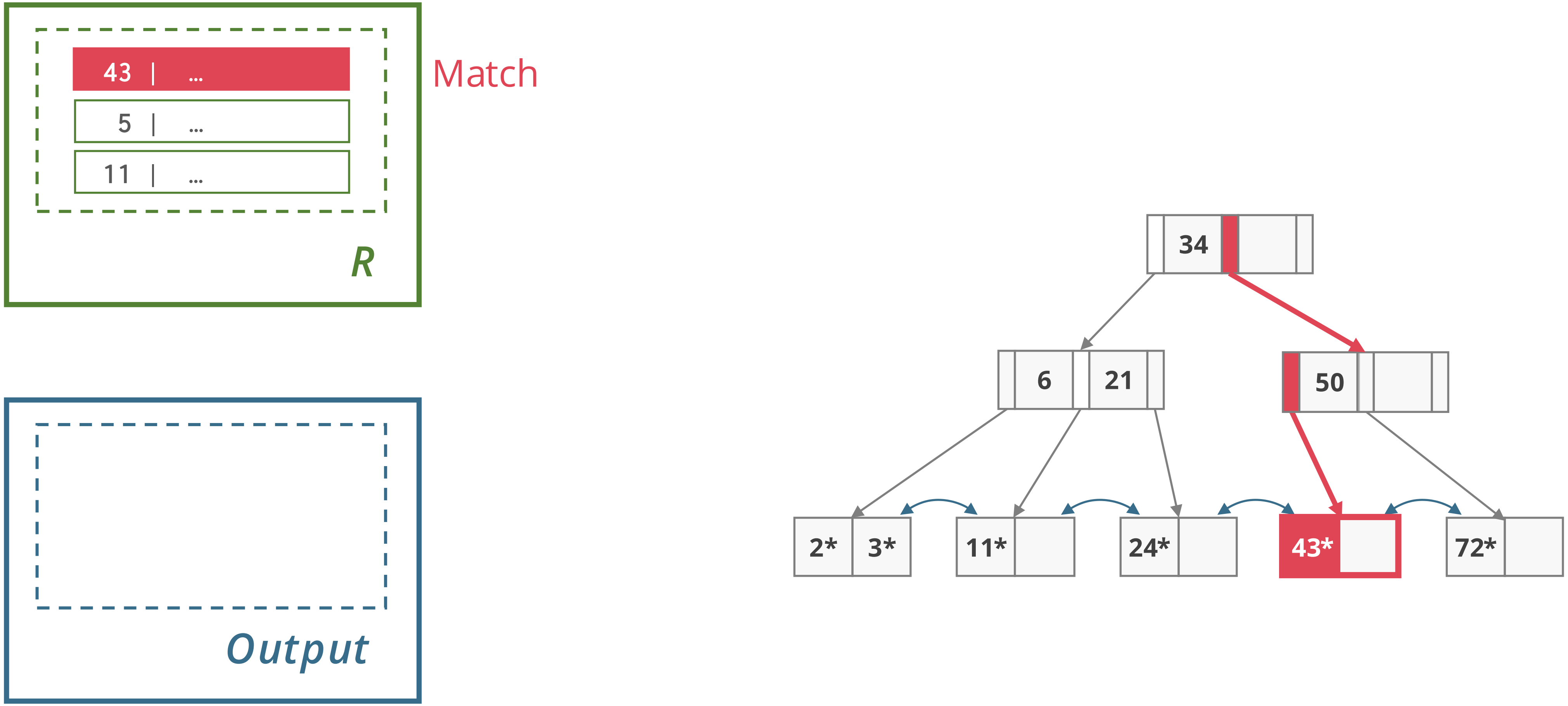
- Then the only way to search for matching S tuples is by scanning all of S ⇒ SNLJ
- Cost to find matching S tuples is then N, giving us the formula for SNLJ cost

Index Nested Loops Join (INLJ)





Index Nested Loops Join (INLJ)



43 | …

5 | …

11 | …

Match

R

Output

34

6 21

50

2* 3*

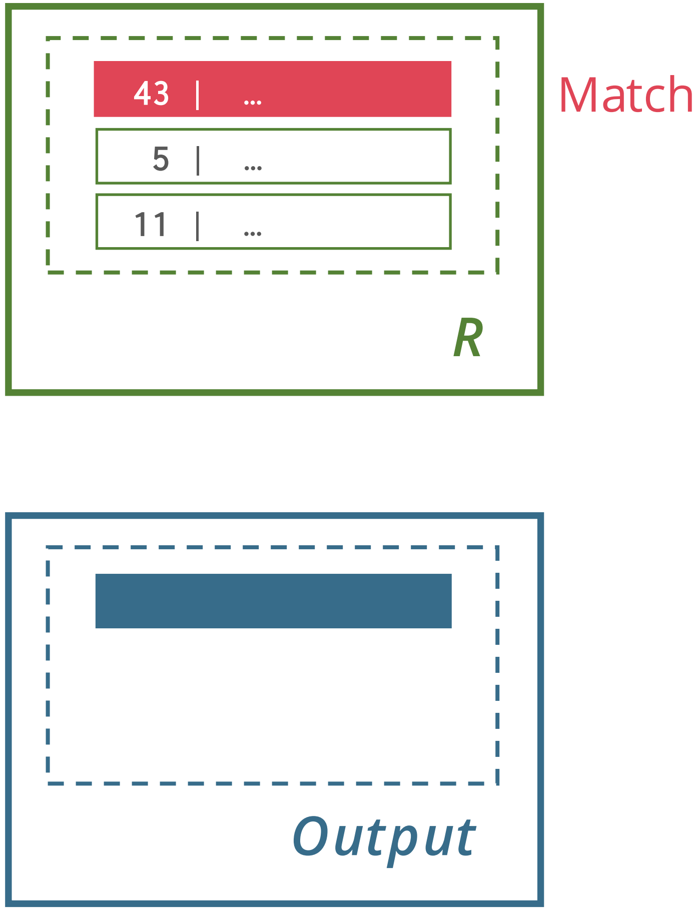
11*

24*

43*

72*

Index Nested Loops Join (INLJ)



43 | …

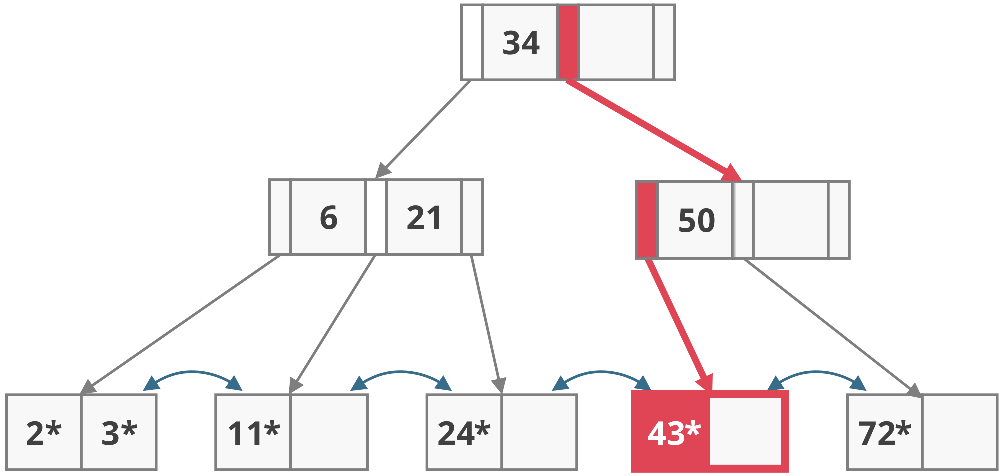
5 | …

11 | …

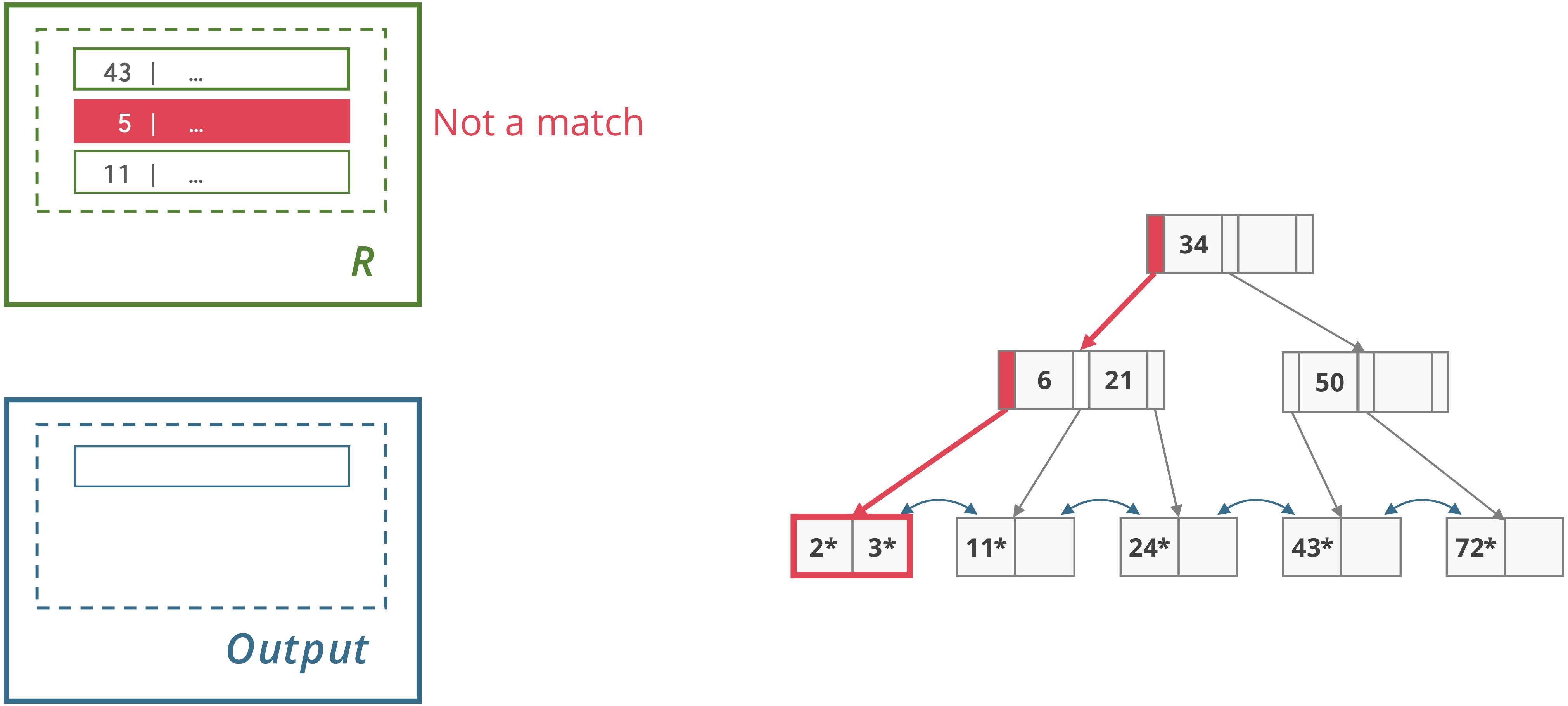
Match

R

Output

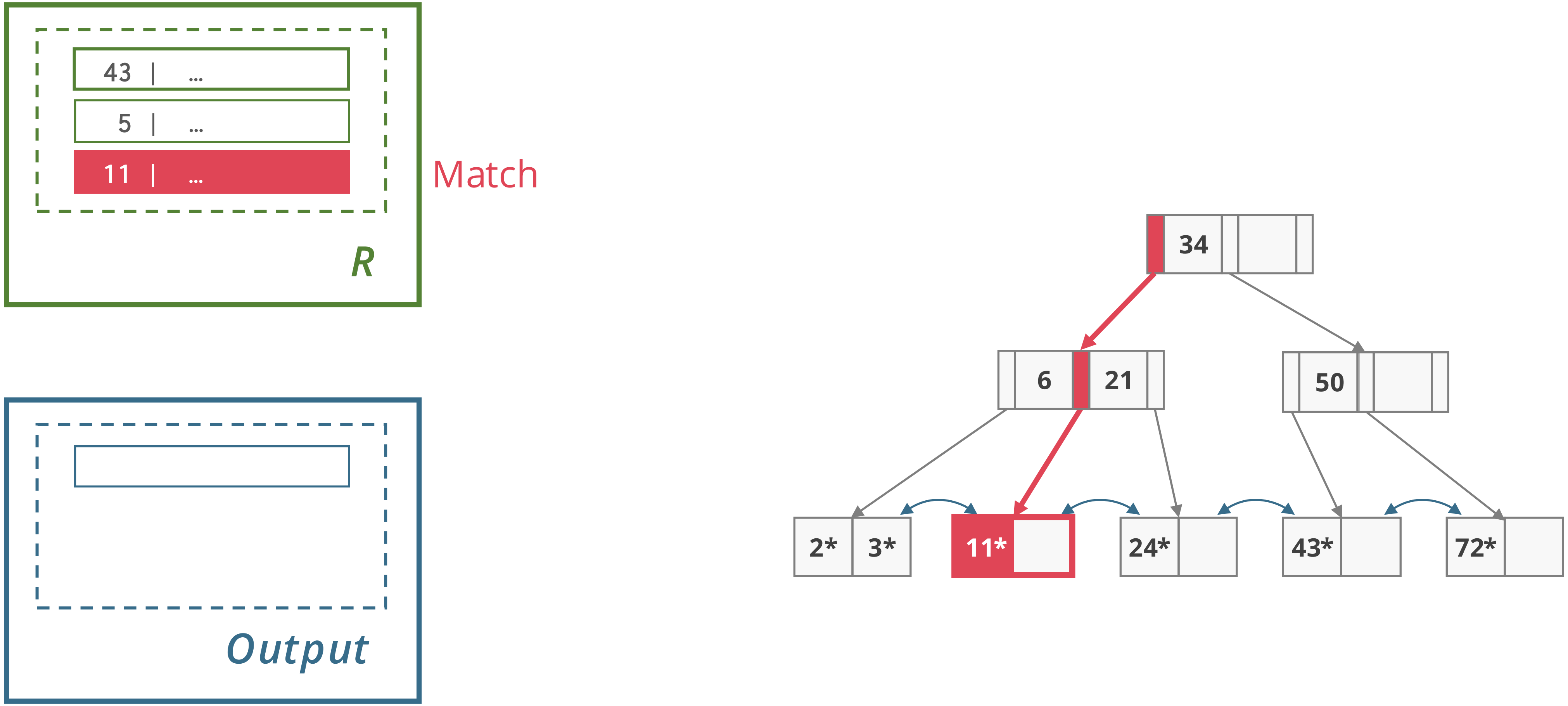


Index Nested Loops Join (INLJ)

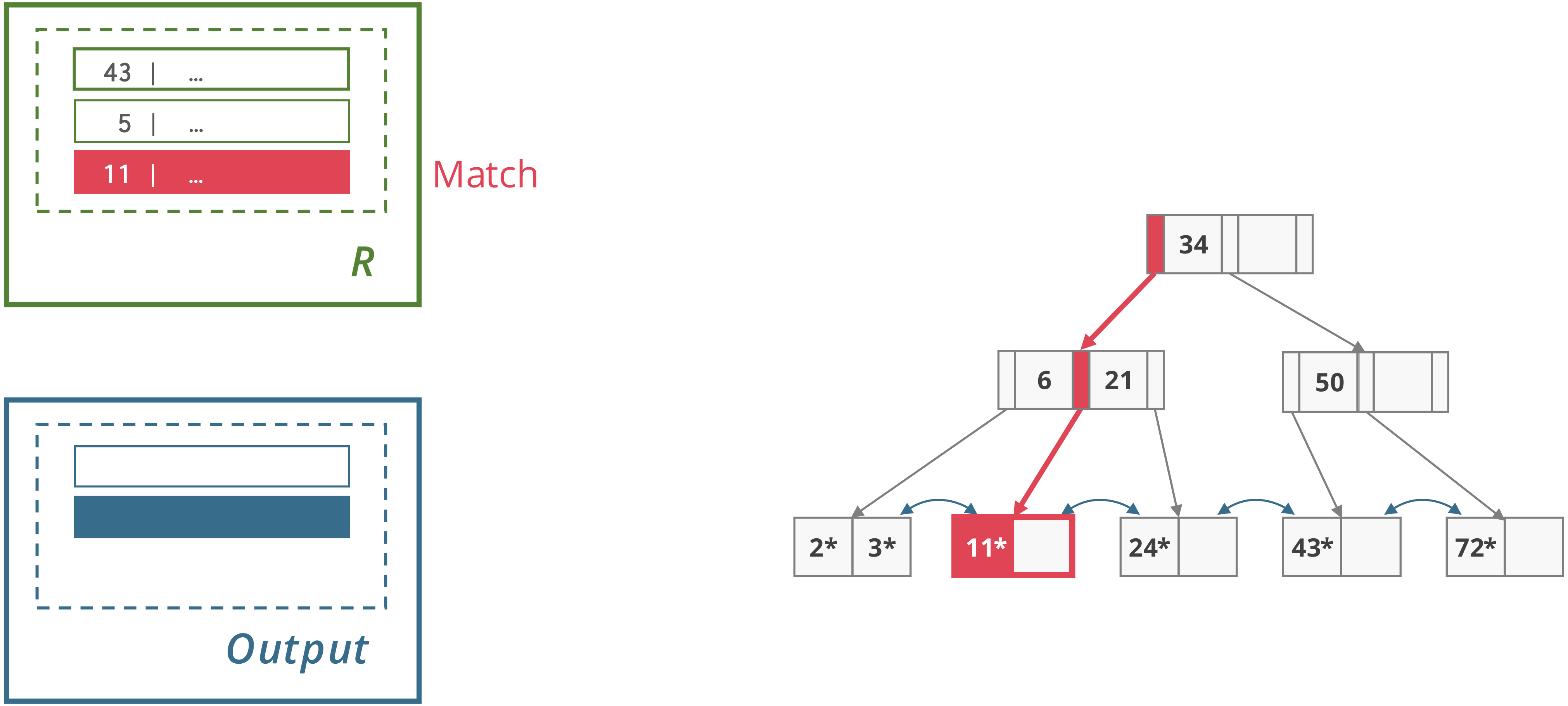


Not a match

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
```

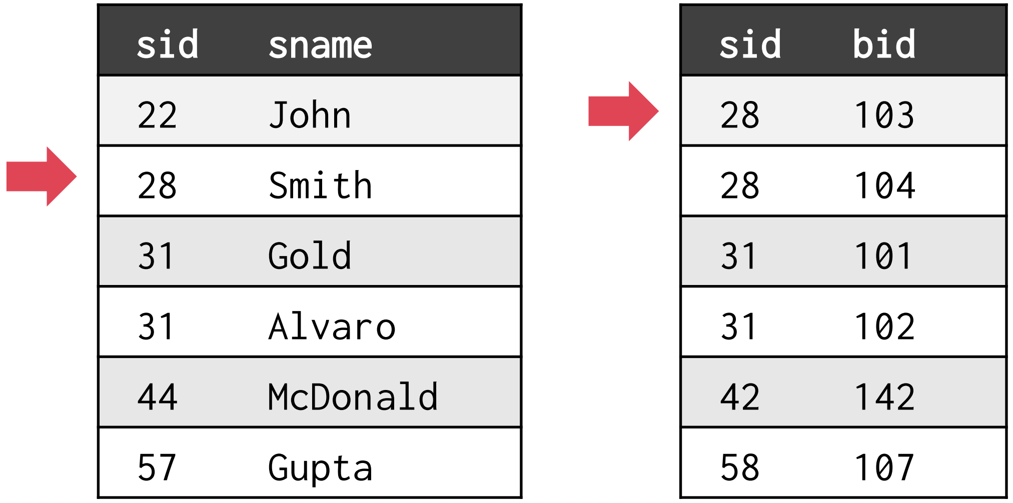
sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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
```

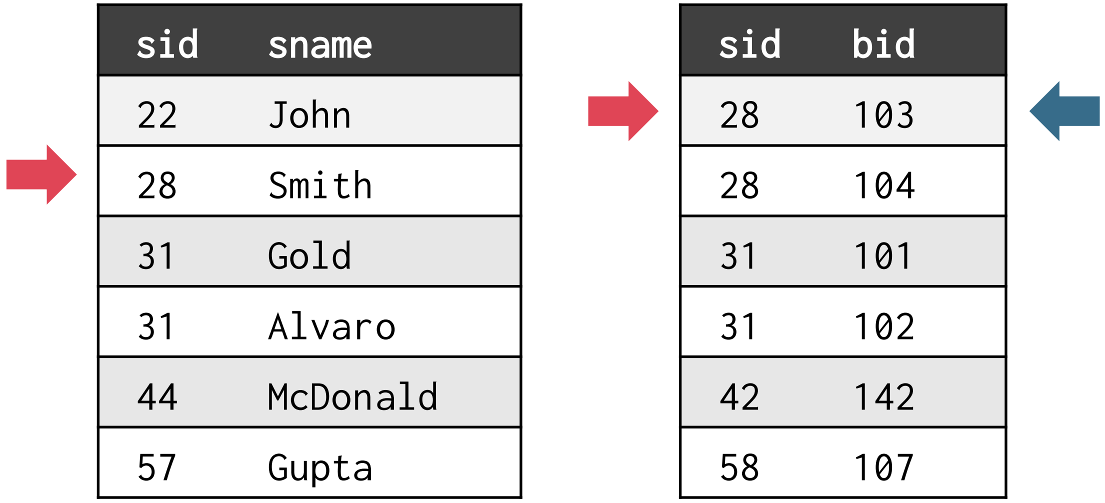
sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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
```



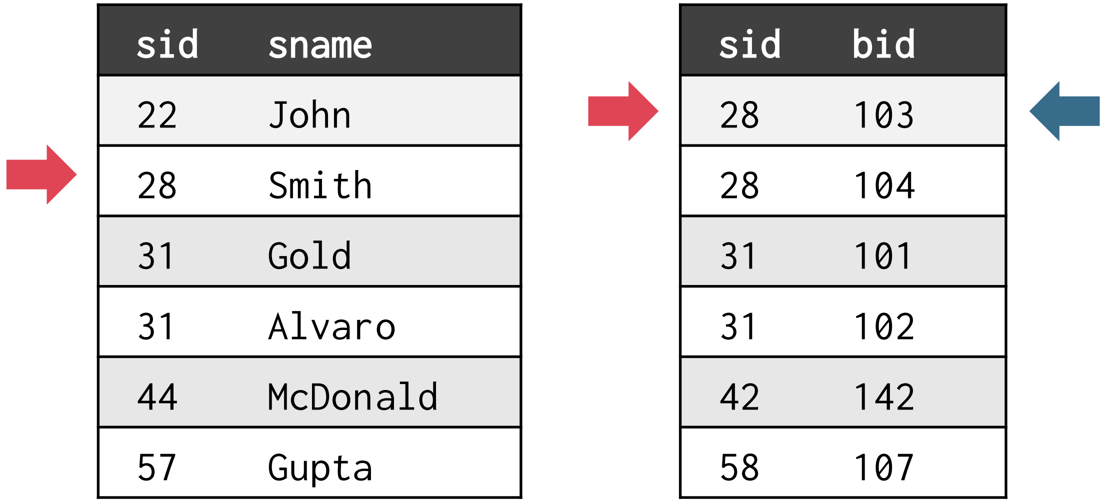
sid	sname
22	John
28	Smith
31	Gold
31	Alvaro
44	McDonald
57	Gupta

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

sid	sname	bid
28	Smith	103

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

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
```

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
```

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
```

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
```

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
```

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
```

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

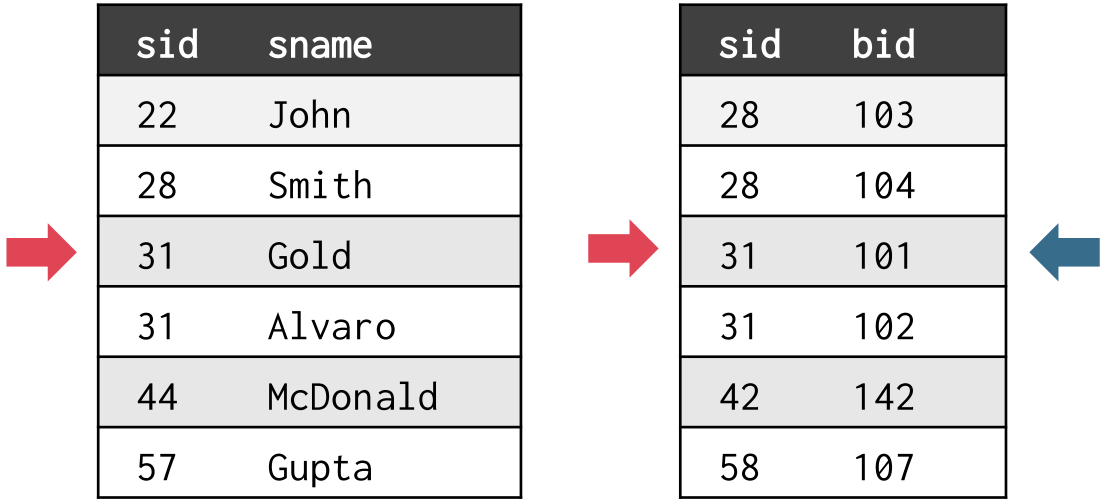
sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

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
```



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
```

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

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

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

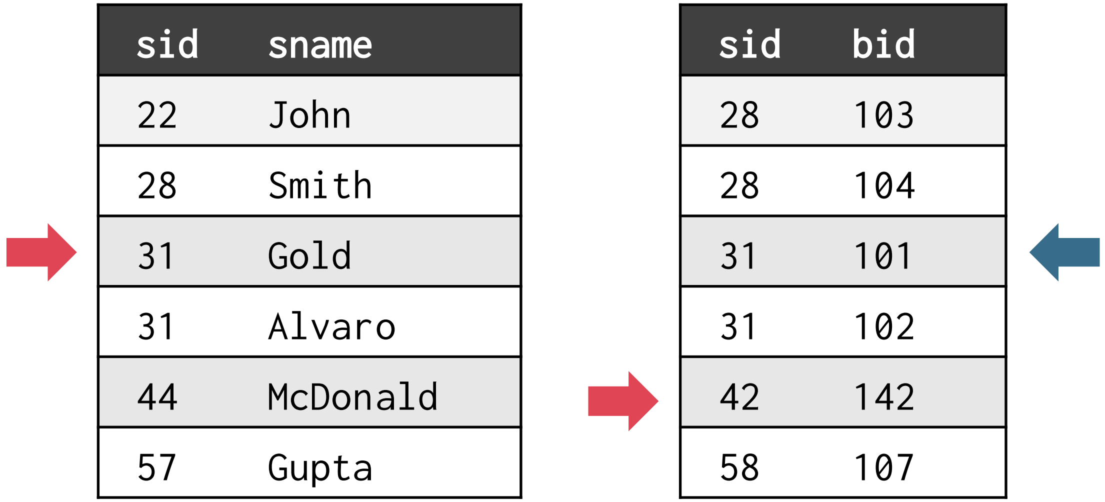
sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

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

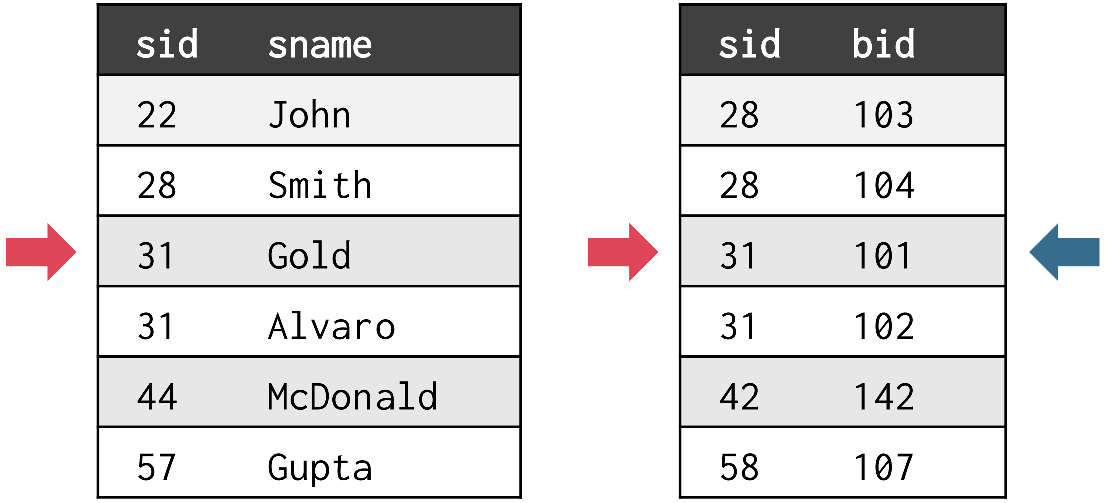
sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

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

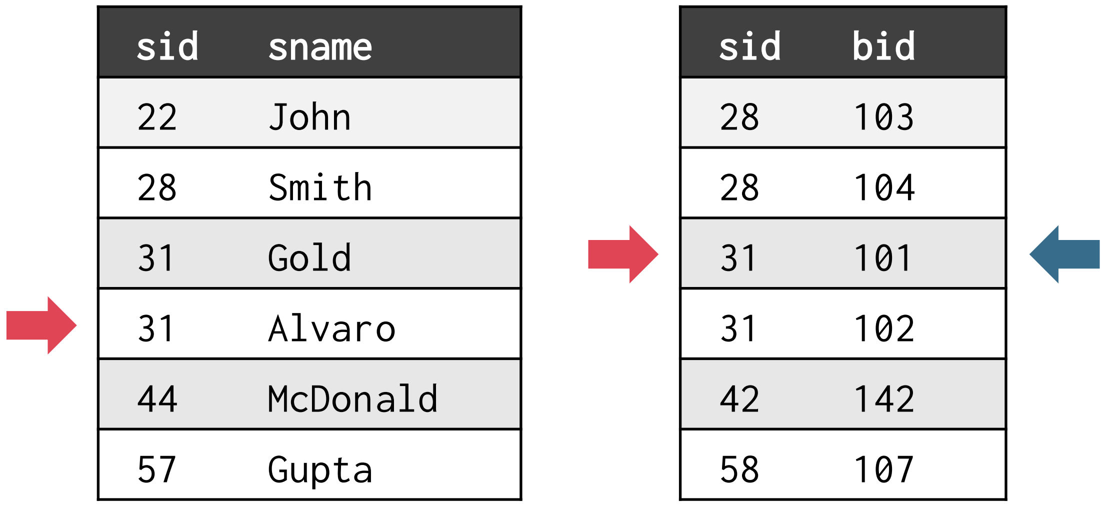
sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

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

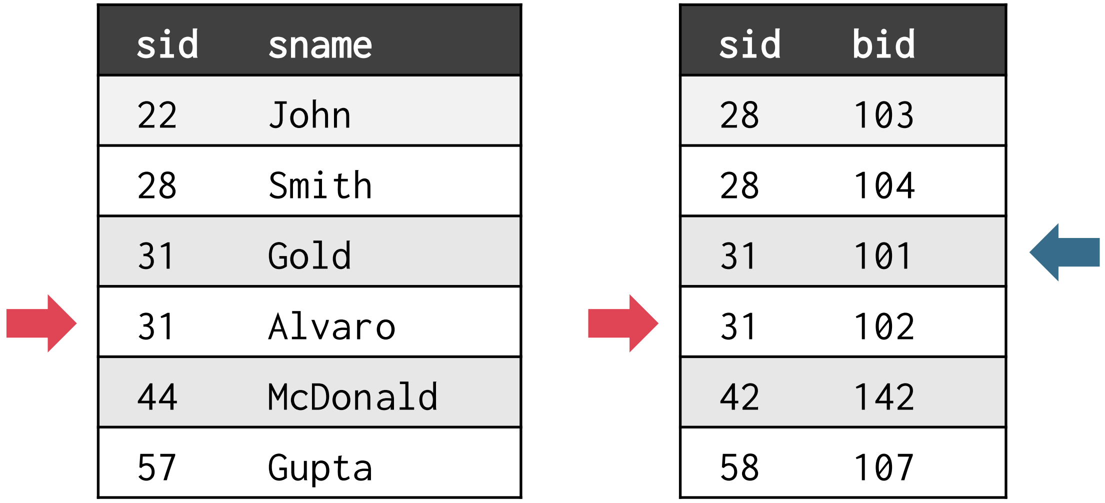
sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

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

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

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
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

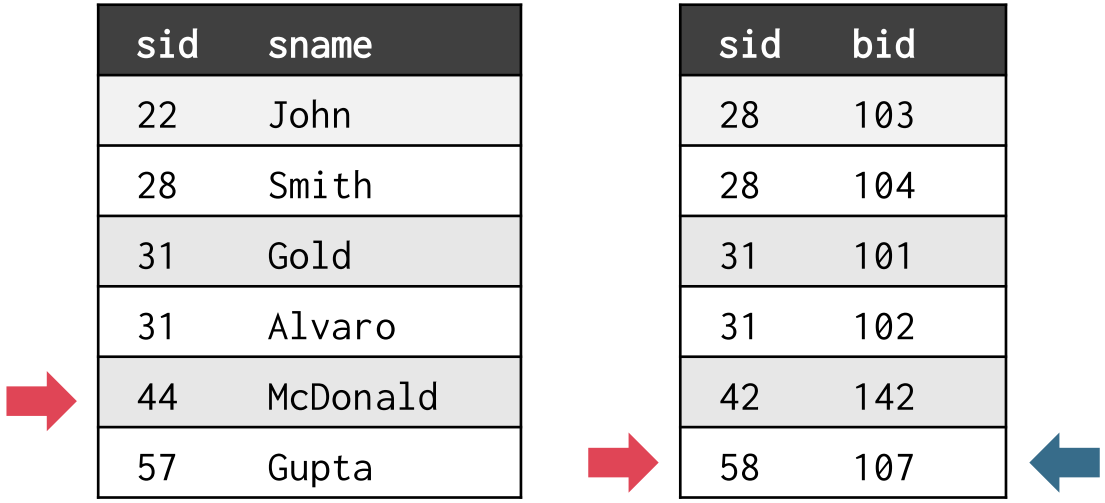
sid	bid
28	103
28	104
31	101
31	102
42	142
58	107

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
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)

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:

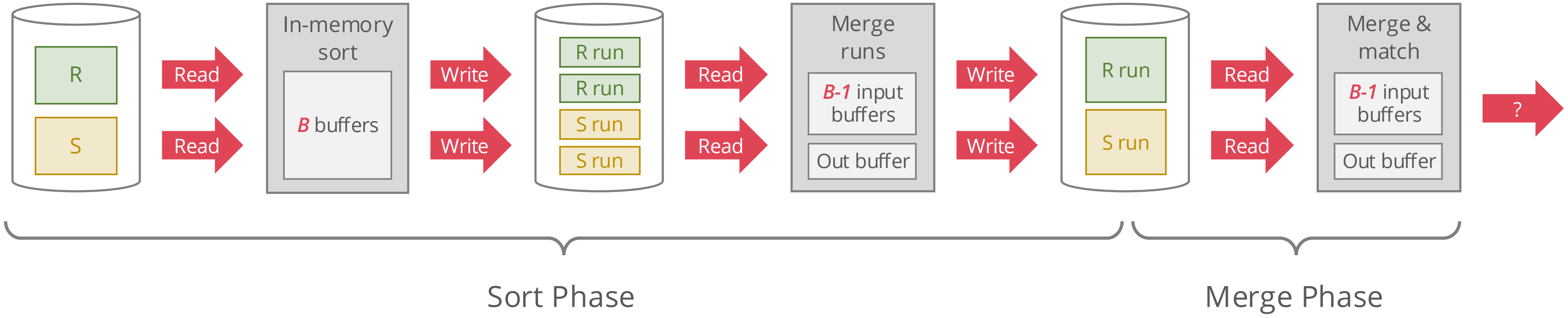
We make **one pass** through each sorted relation (assuming no duplicate values in ***R***)

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

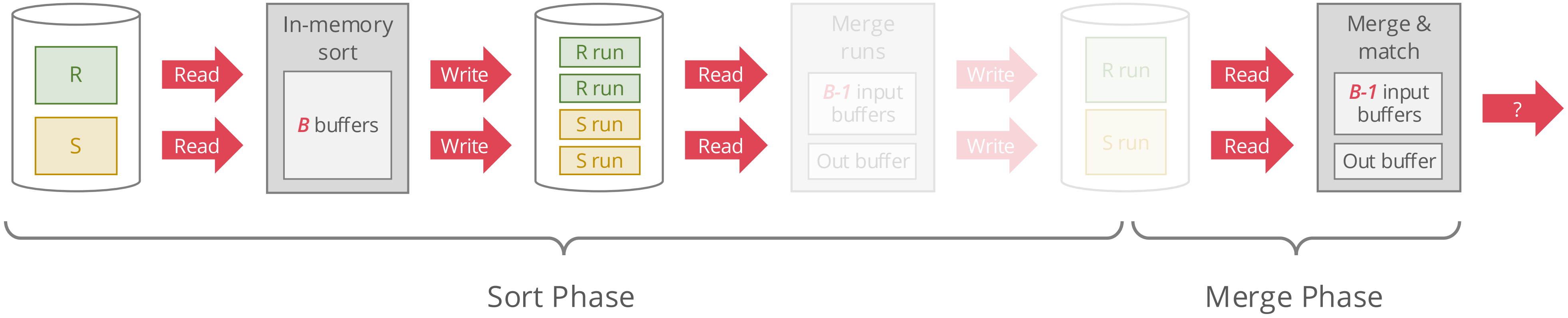
Optimisation:

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: (# runs in last merge pass for ***R***) + (# runs in last merge pass for ***S***) ≤ ***B – 1***

Reduces I/O cost by ***2 · (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

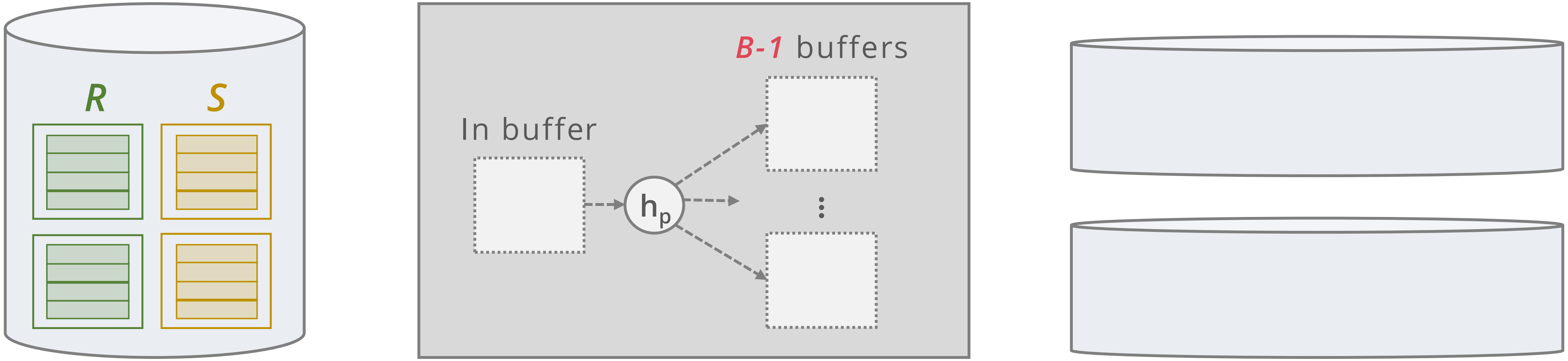
Grace Hash Join

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

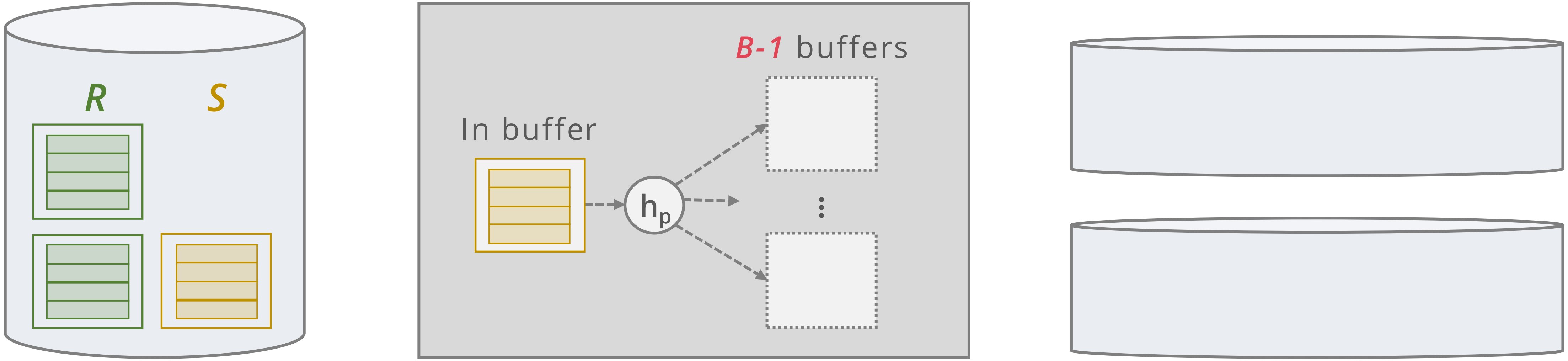
Two passes: **partition** the data, then **build** a hash table and **probe** it

- 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

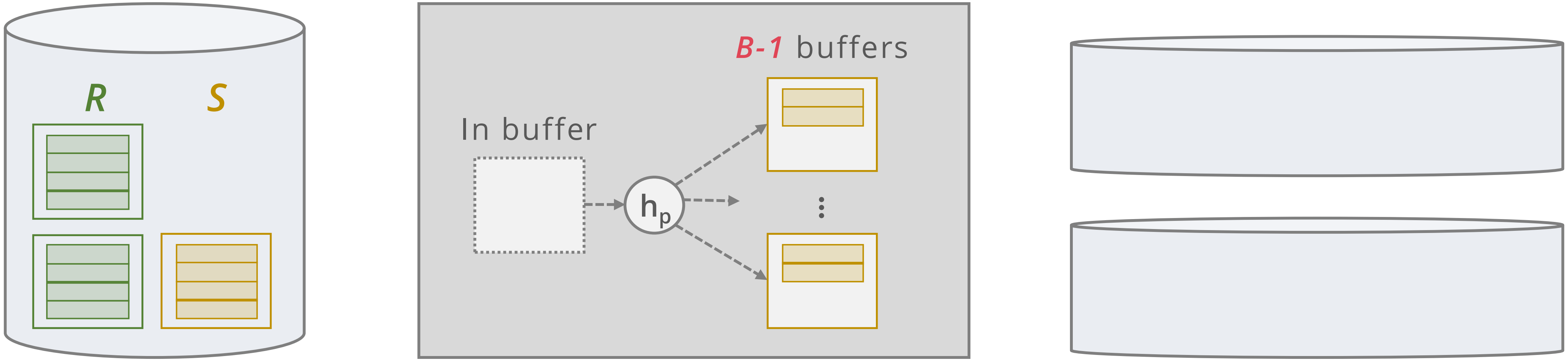
Grace Hash Join: Partition



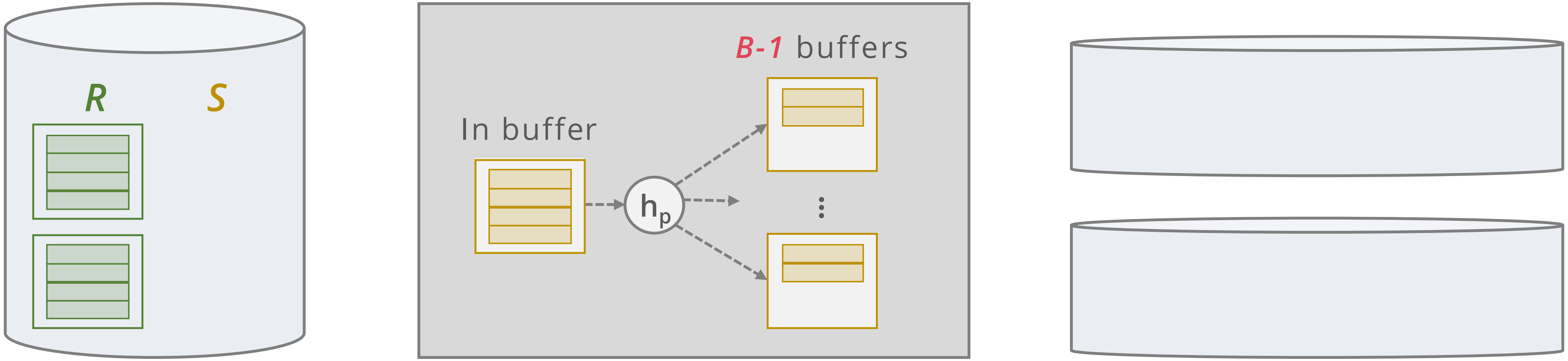
Grace Hash Join: Partition



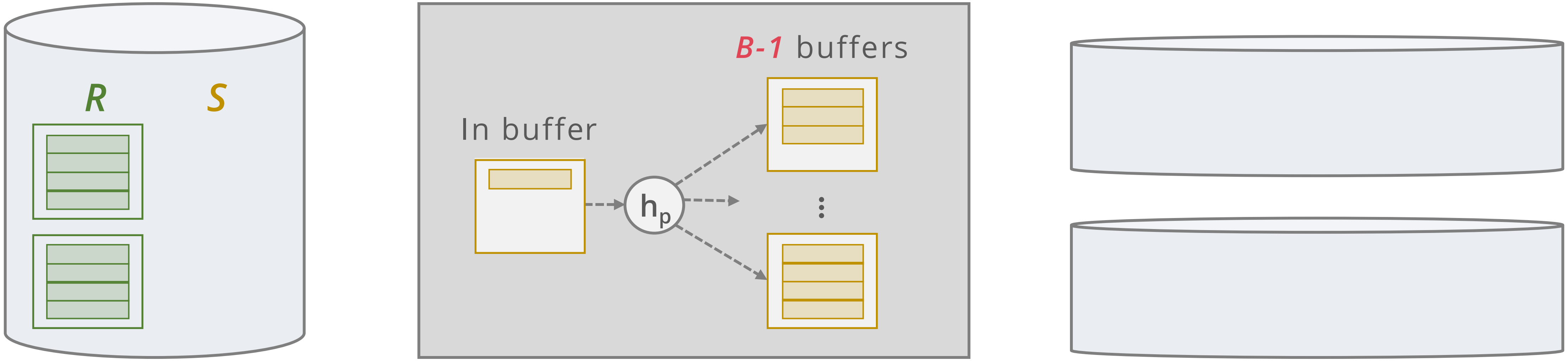
Grace Hash Join: Partition



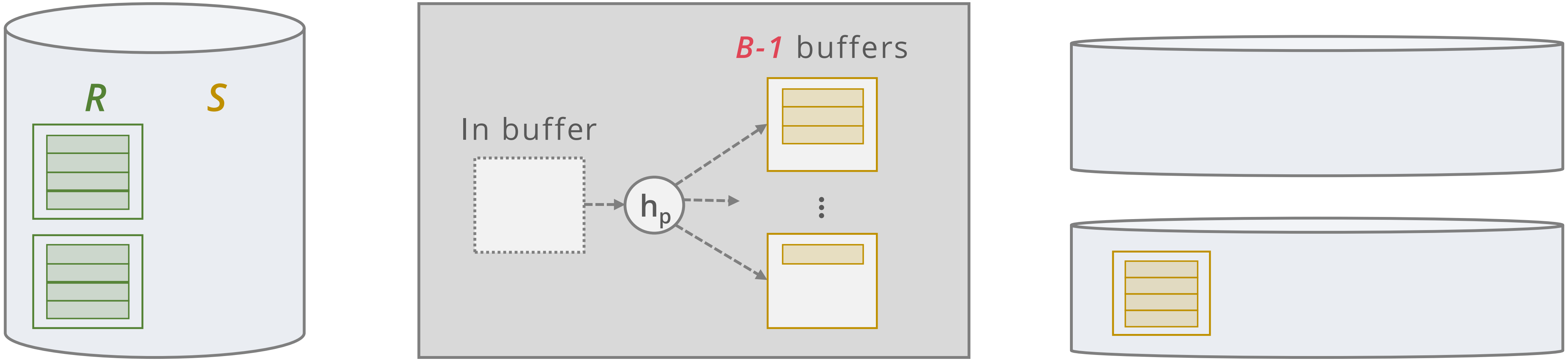
Grace Hash Join: Partition



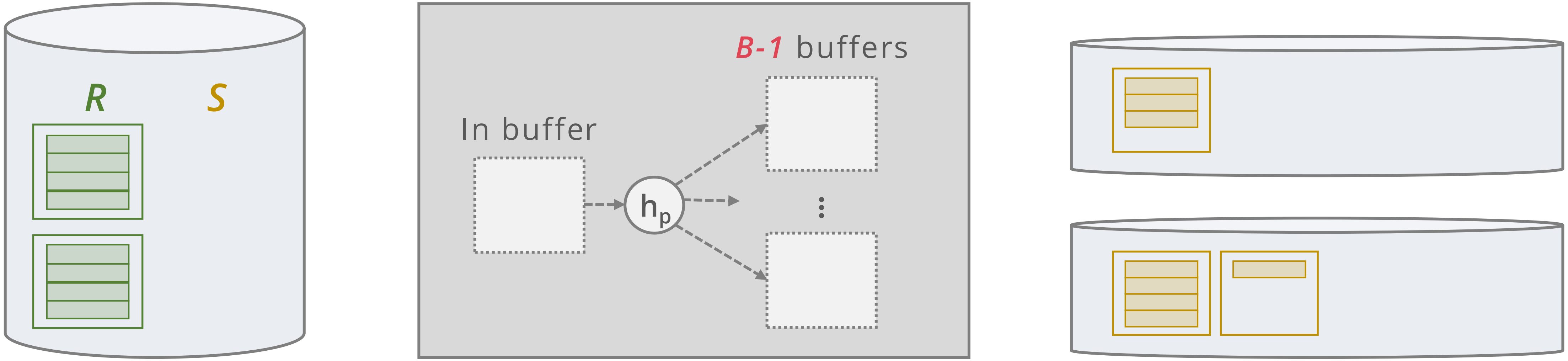
Grace Hash Join: Partition



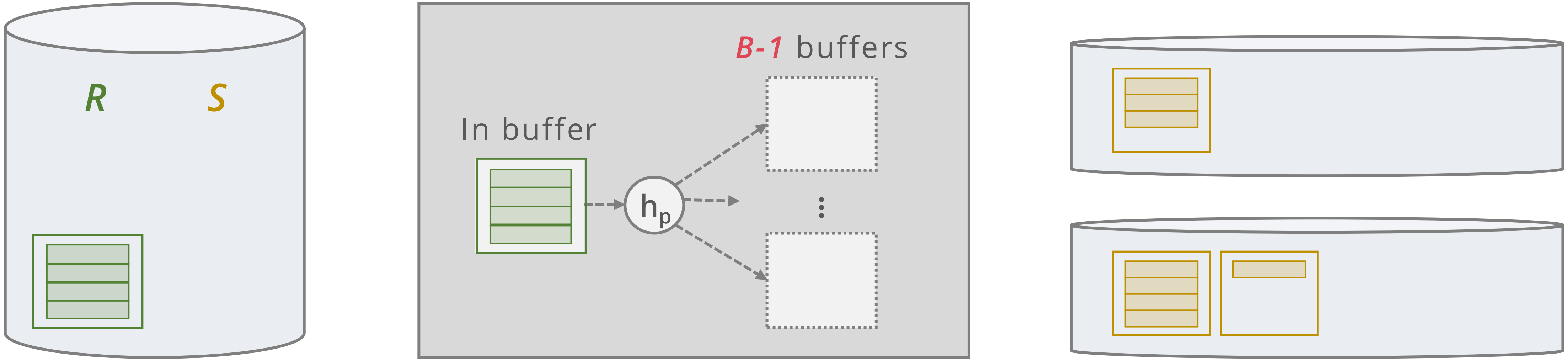
GRACE HASH JOIN: PARTITION



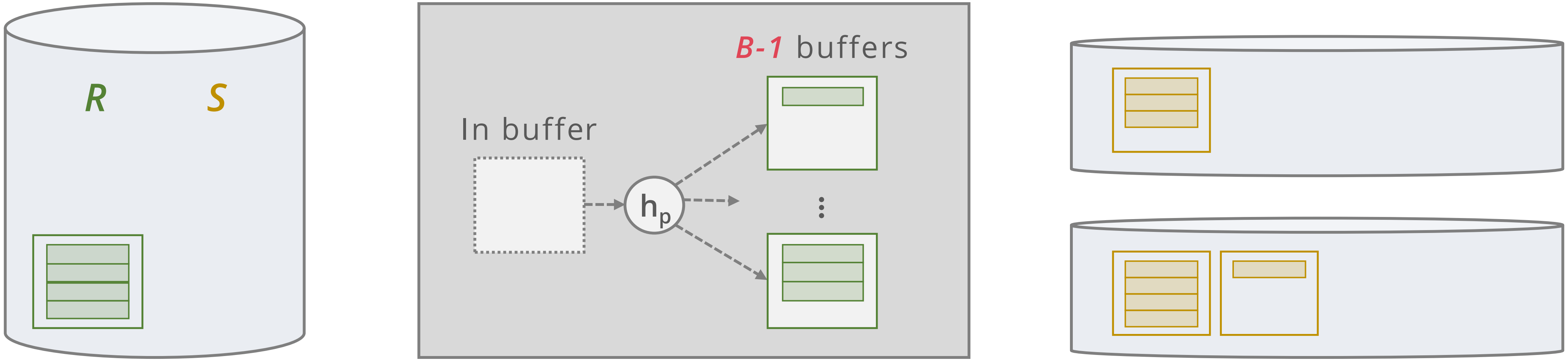
Grace Hash Join: Partition



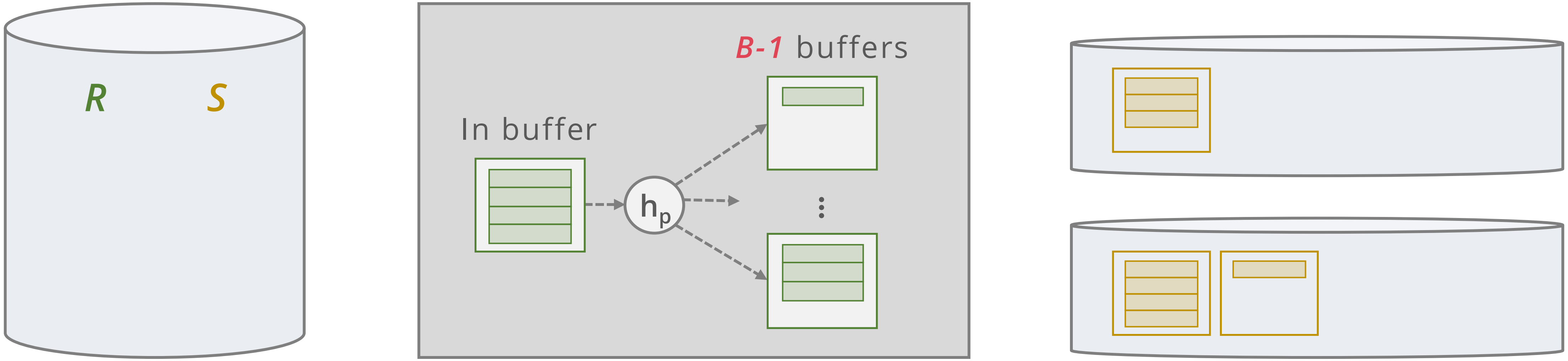
Grace Hash Join: Partition



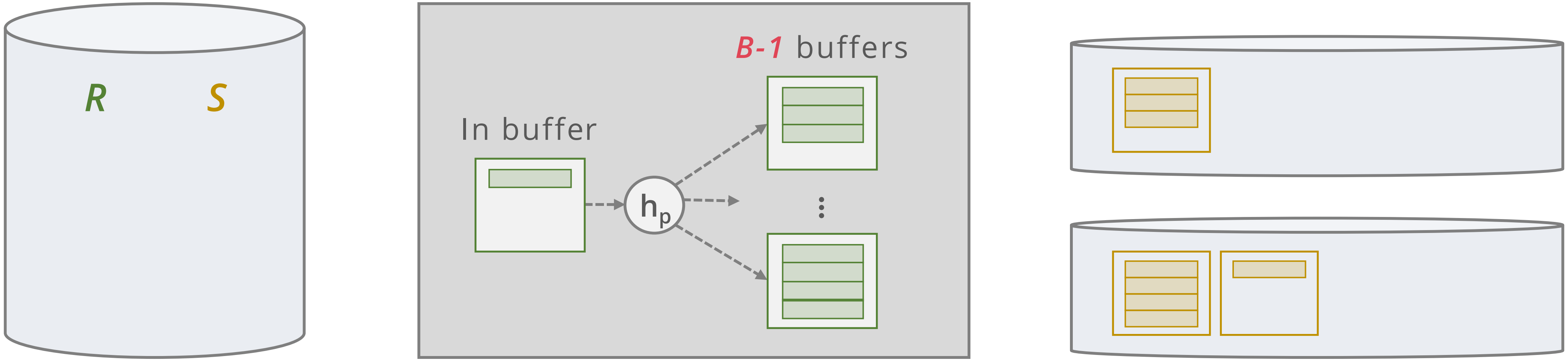
Grace Hash Join: Partition



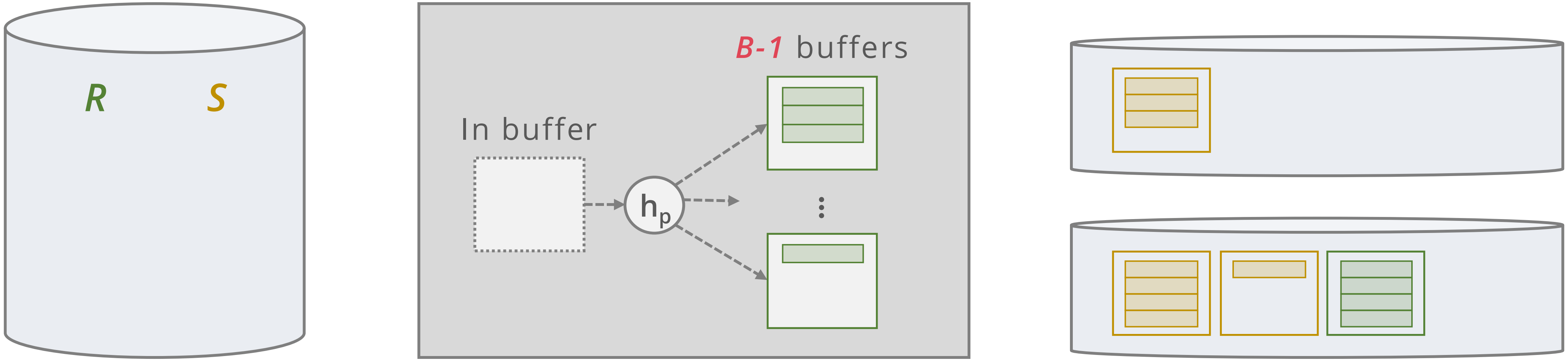
Grace Hash Join: Partition



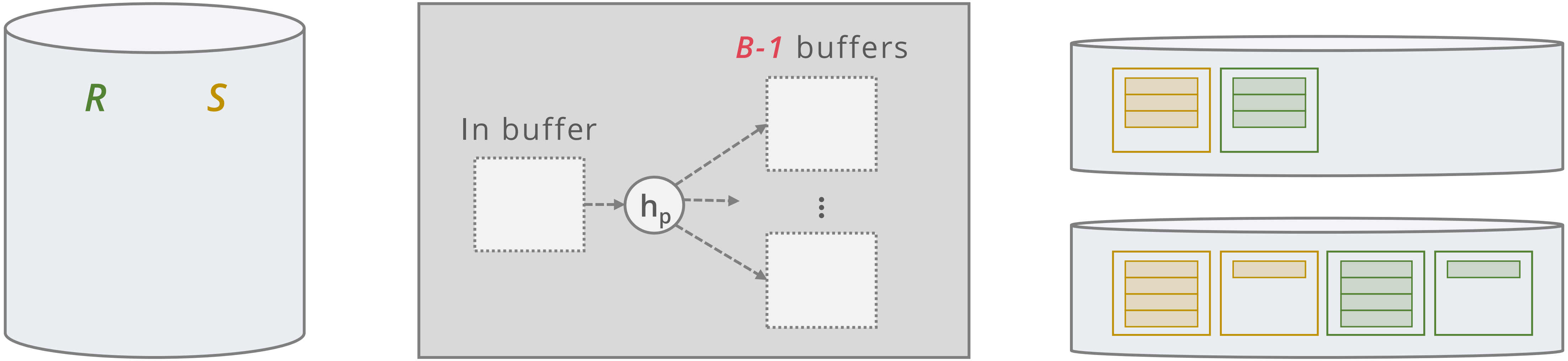
GRACE HASH JOIN: PARTITION



GRACE HASH JOIN: PARTITION



Grace Hash Join: Partition



Grace Hash Join

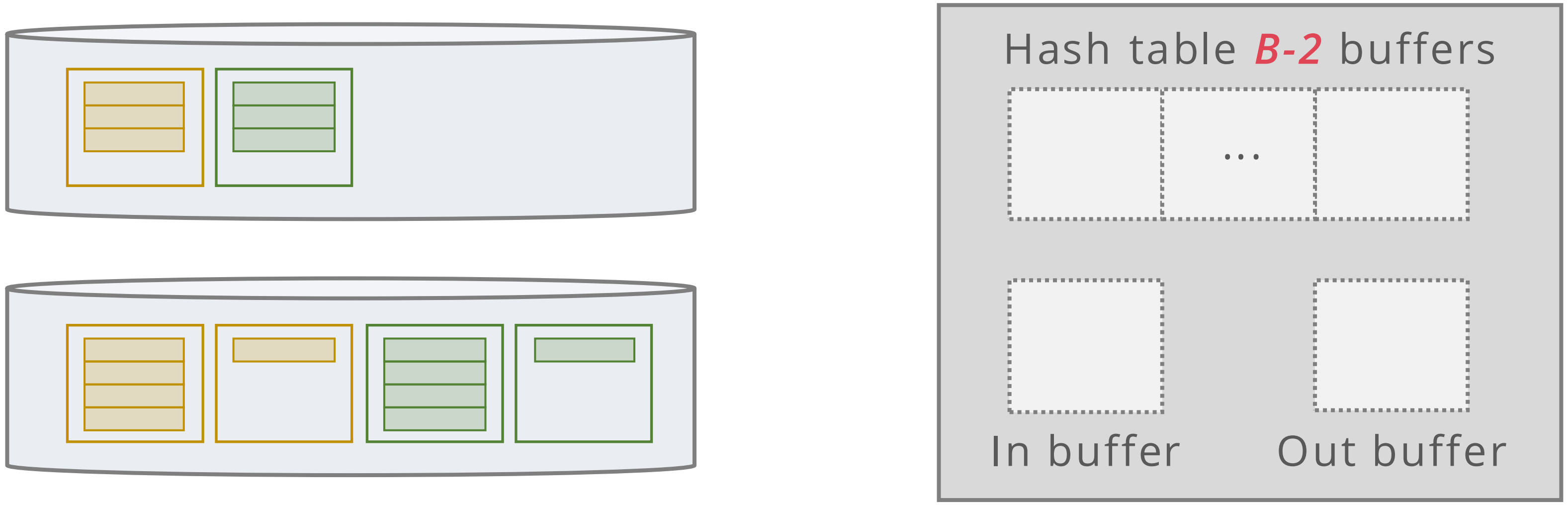
We need partitions of ***R*** (but not ***S***!) to fit in ***B* - 2** pages

- 1 page reserved for streaming ***S*** partition
- 1 page reserved for streaming output

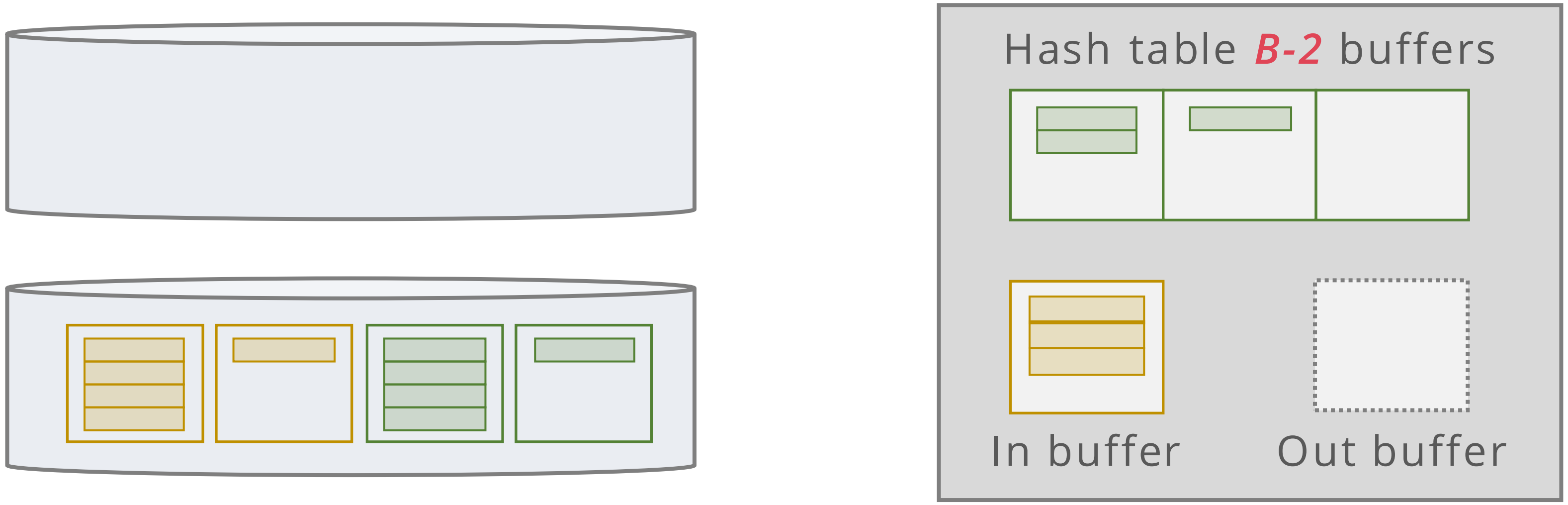
What if partitions of ***R*** are too big?

- 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!

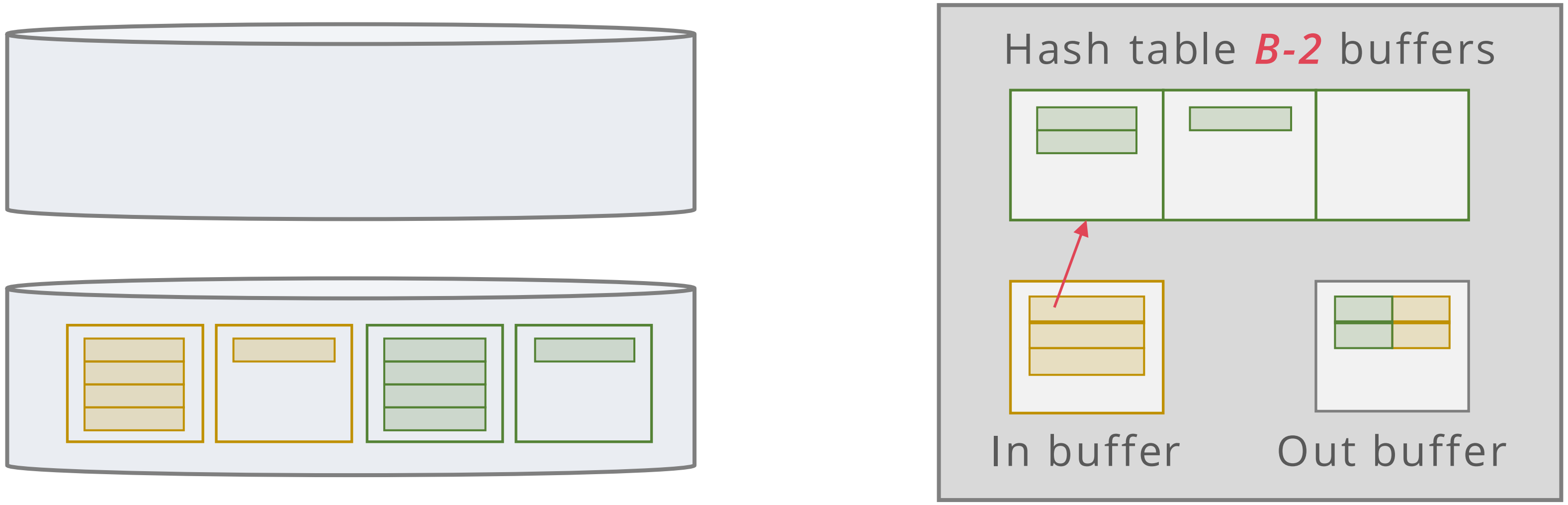
Grace Hash Join: Build & Probe



Grace Hash Join: Build & Probe

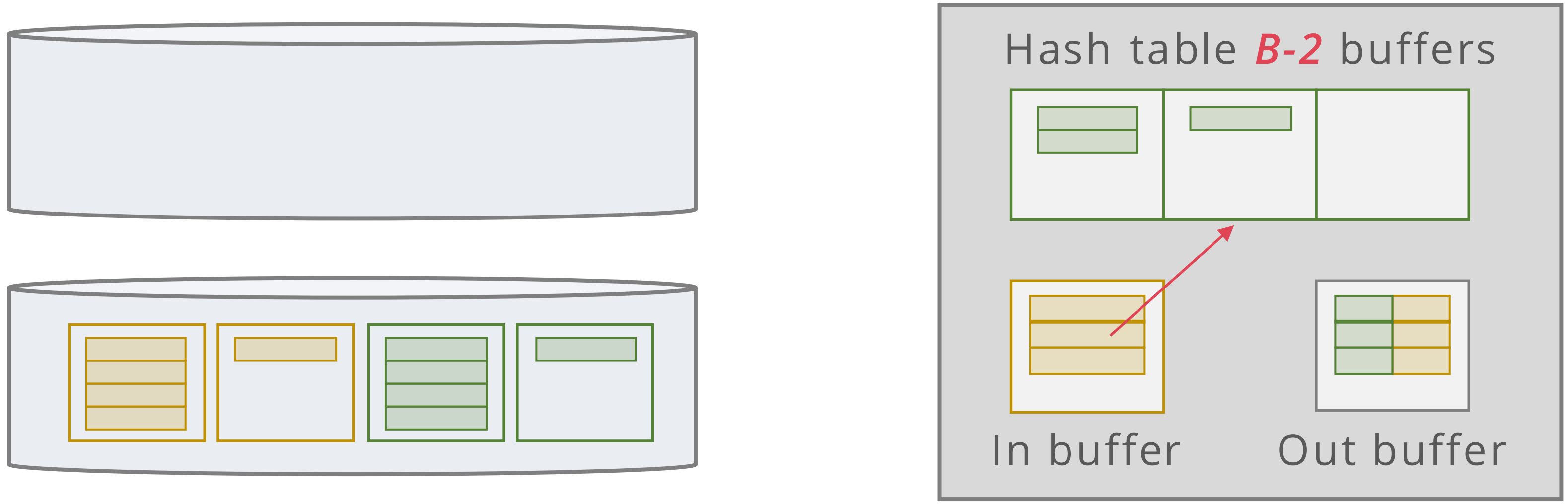


GRACE HASH JOIN: BUILD & PROBE



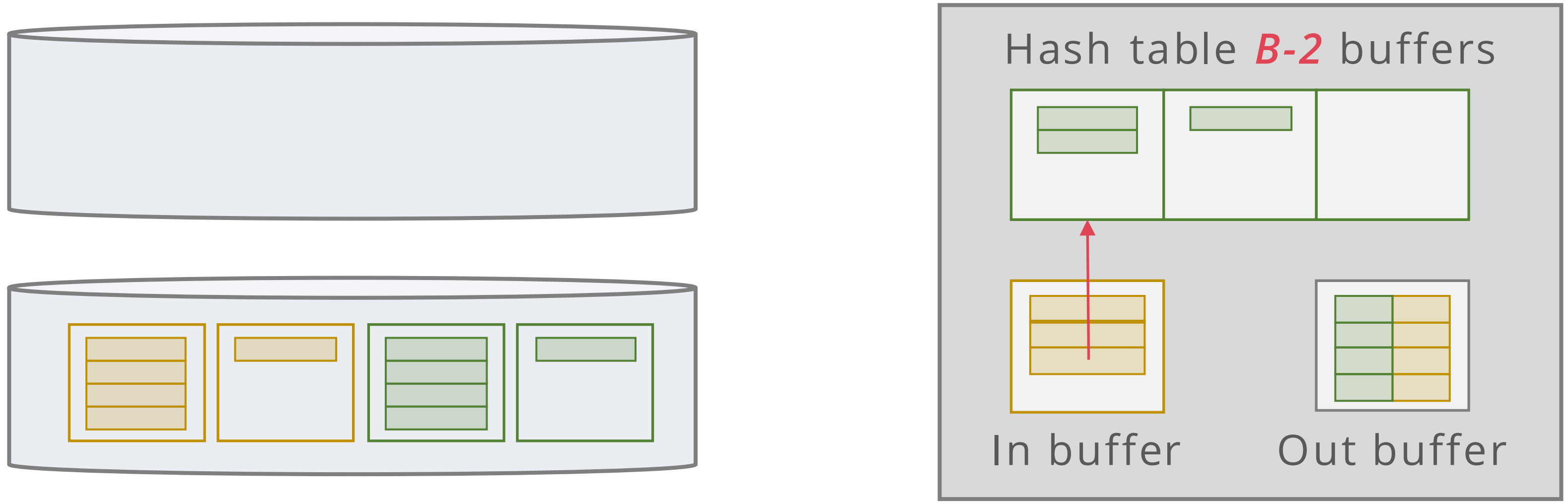
2 matches

GRACE HASH JOIN: BUILD & PROBE



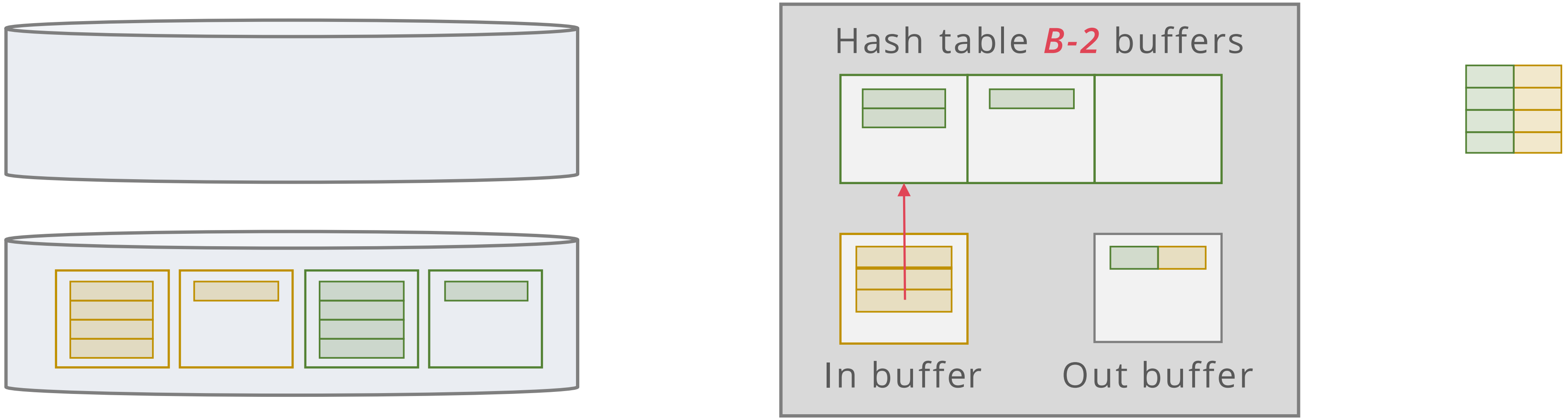
1 match

Grace Hash Join: Build & Probe



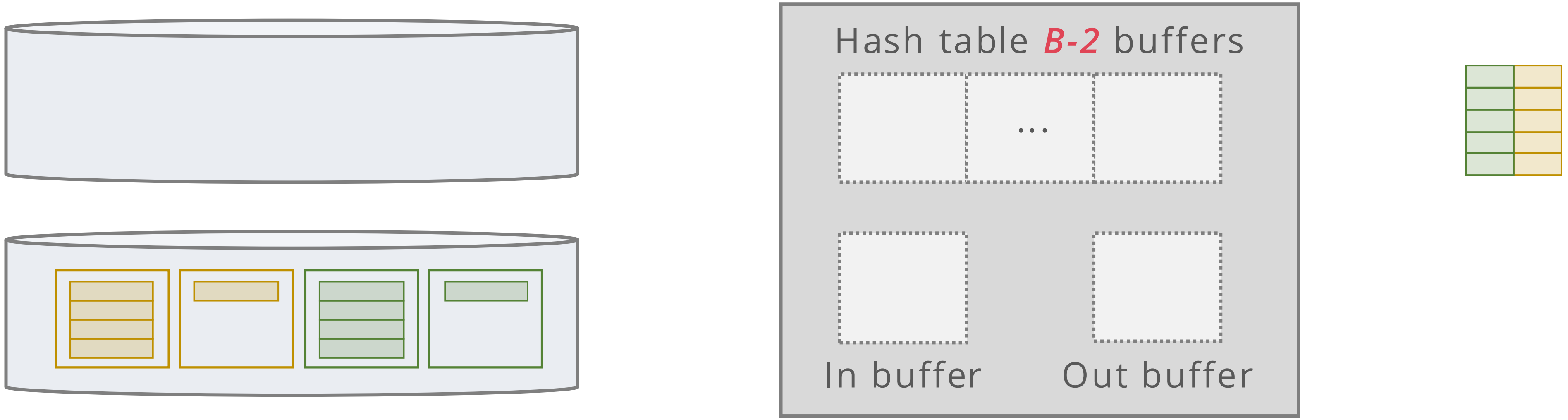
2 matches

GRACE HASH JOIN: BUILD & PROBE



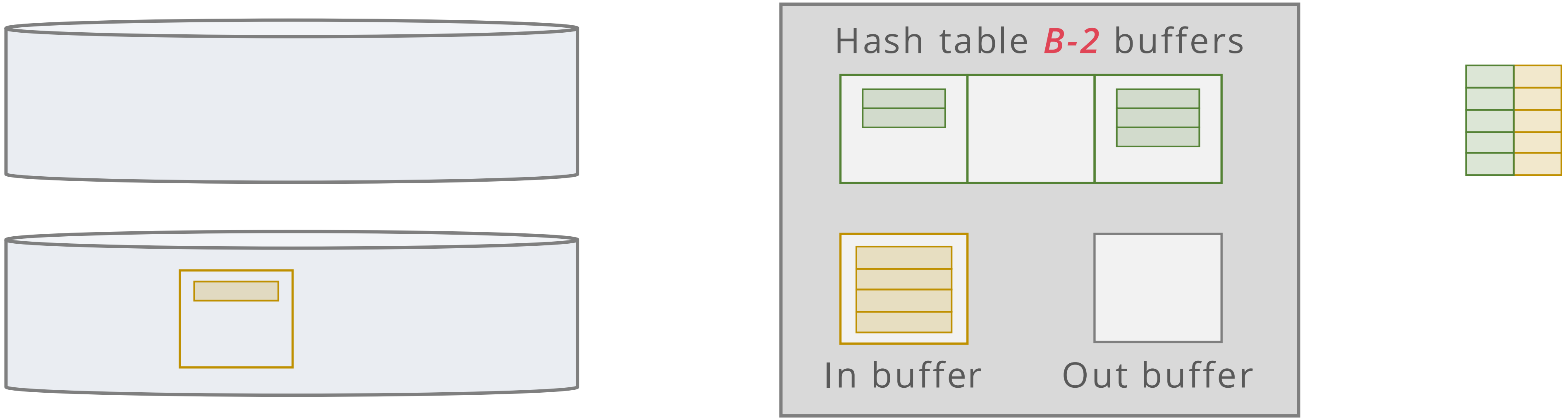
2 matches

Grace Hash Join: Build & Probe

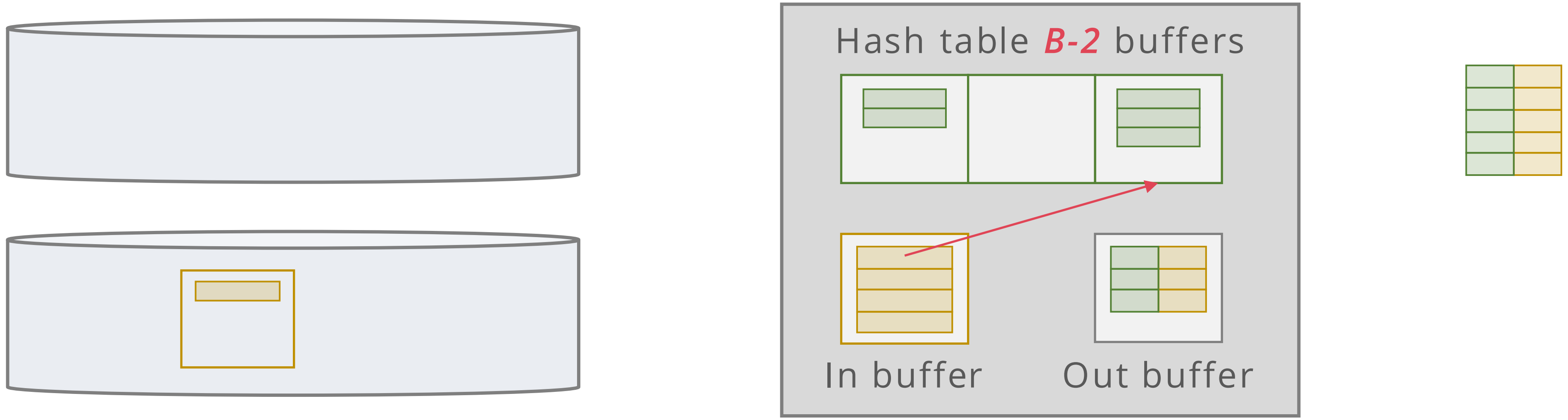


Flushing output buffer not strictly necessary, can reuse for next pair of partitions

GRACE HASH JOIN: BUILD & PROBE

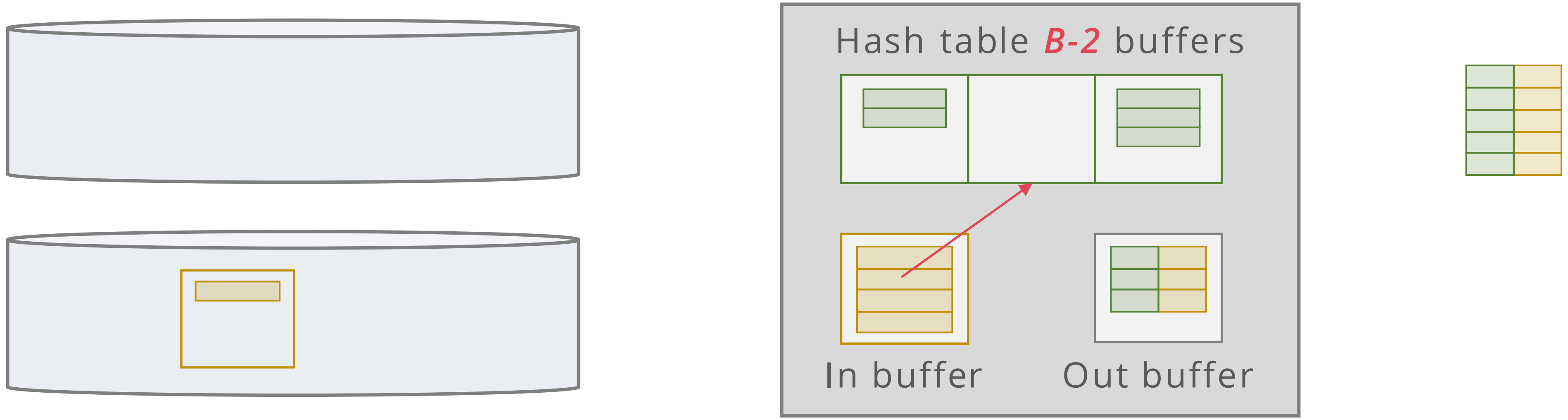


Grace Hash Join: Build & Probe



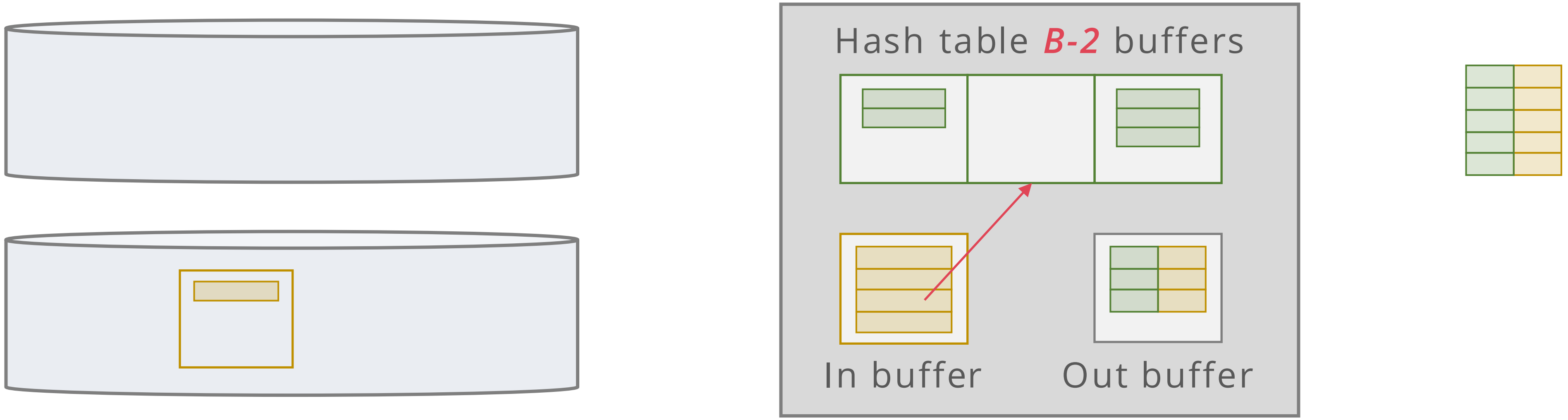
3 matches

GRACE HASH JOIN: BUILD & PROBE



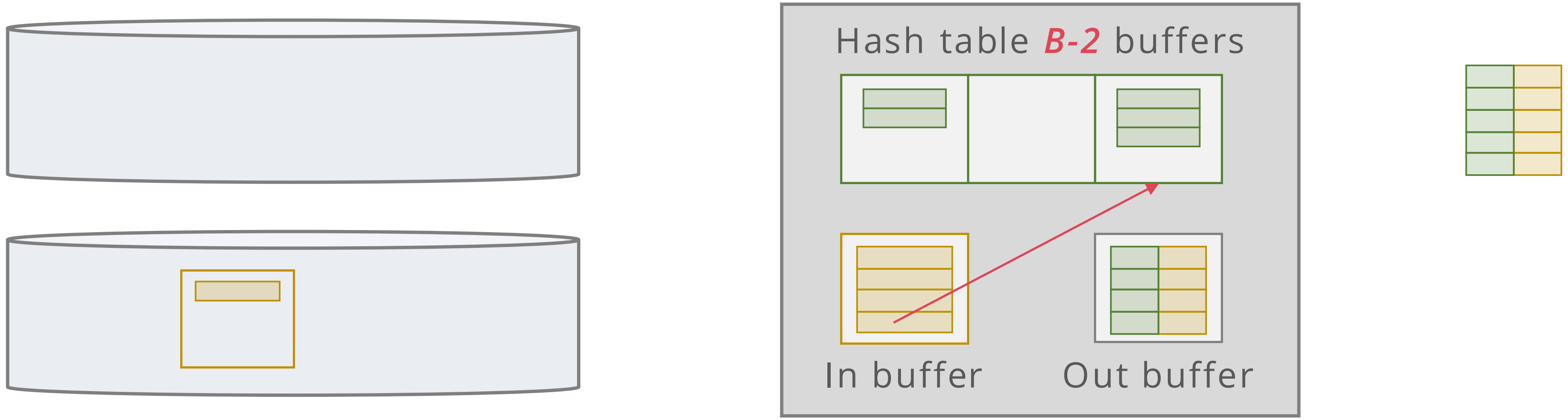
No match

GRACE HASH JOIN: BUILD & PROBE



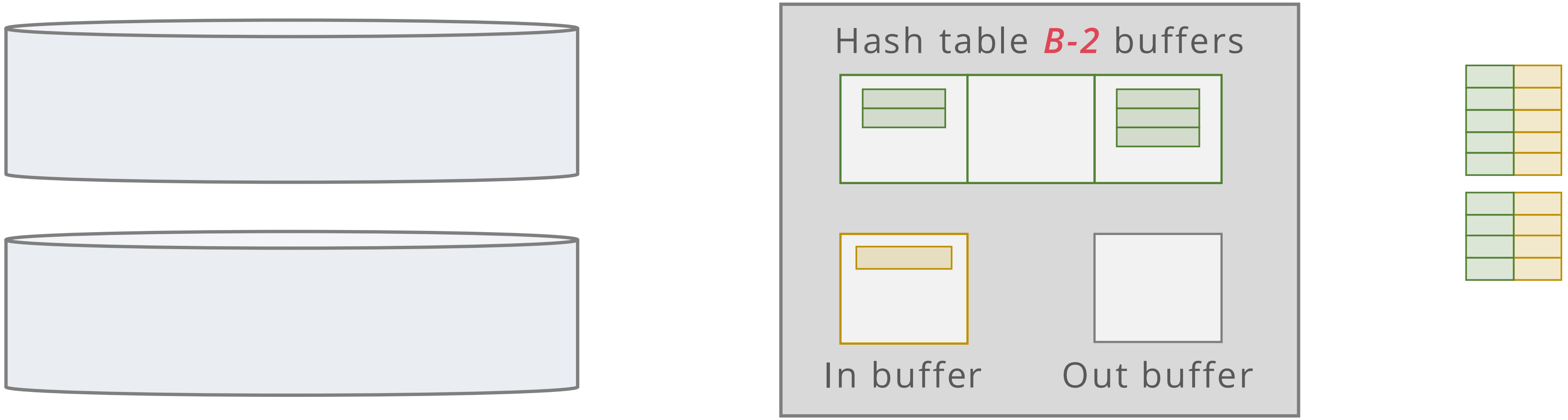
No match

Grace Hash Join: Build & Probe

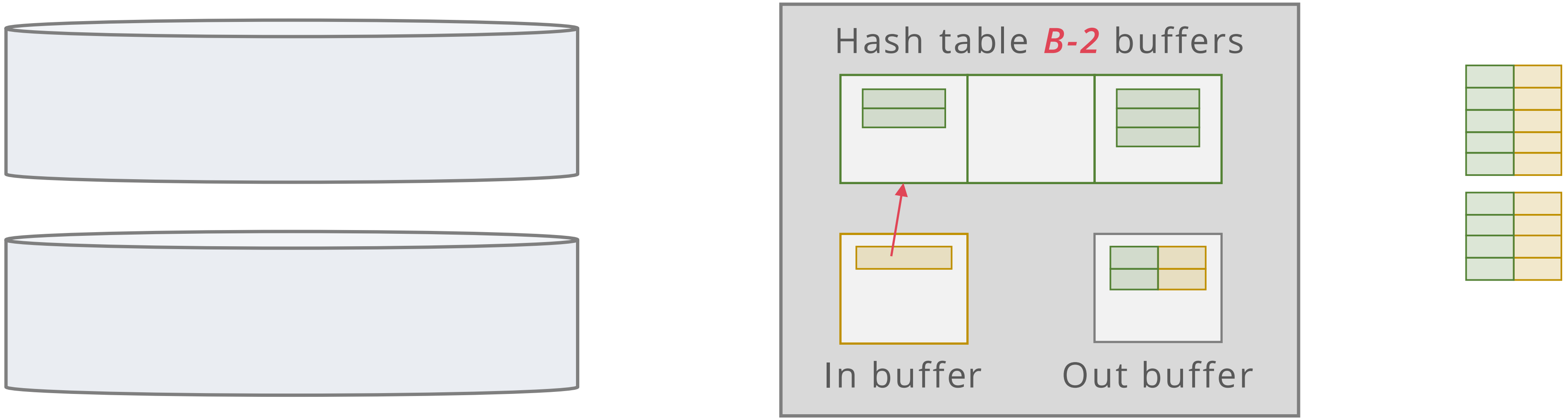


1 match

Grace Hash Join: Build & Probe

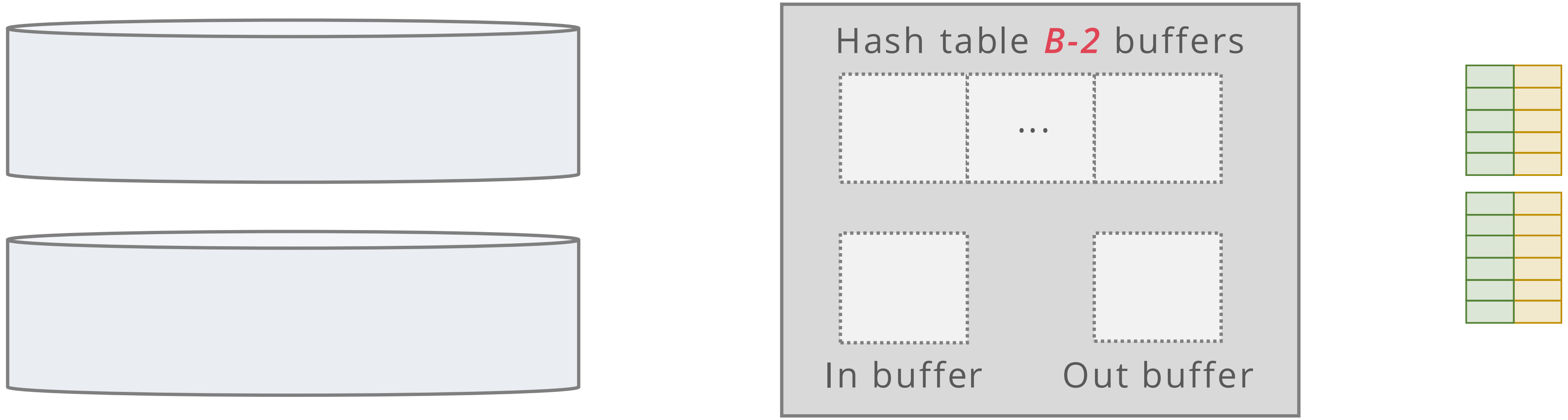


GRACE HASH JOIN: BUILD & PROBE



2 matches

Grace Hash Join: Build & Probe


Hash table *B-2* buffers
...
In buffer
Out buffer