Sorting

- Commonly required for many operations
  - Duplicate elimination, group by's, sort-merge join
  - Queries may have ASC or DSC in the query
- One option:
  - Read the lowest level of B+-tree
    - May be enough in many cases
  - But if relation not sorted, too many random accesses
- If relation small enough…
  - Read in memory, use quicksort (qsort() in C)
- What if relation too large to fit in memory ?
  - External sort-merge

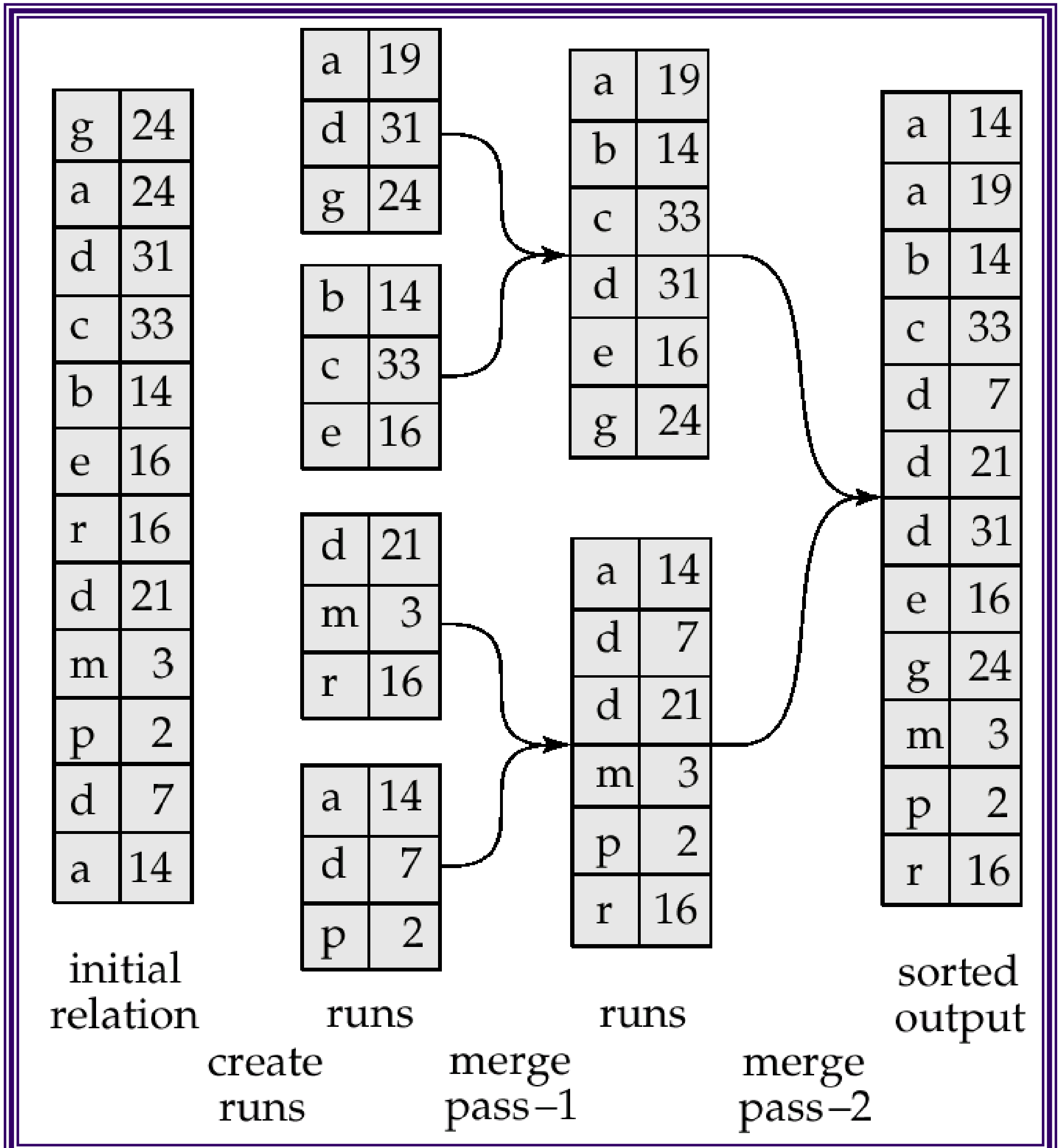
External sort-merge

- Divide and Conquer !!
- Let *M* denote the memory size (in blocks)
- Phase 1:
  - Read first M blocks of relation, sort, and write it to disk
  - Read the next M blocks, sort, and write to disk …
  - Say we have to do this "N" times
  - Result: *N* sorted runs of size *M* blocks each
- Phase 2:
  - Merge the *N* runs (*N-way merge)*
  - Can do it in one shot if $N < M$

External sort-merge

- Phase 1:
  - Create *sorted runs of size M* each
  - Result: N sorted runs of size M blocks each
- Phase 2:
  - Merge the N runs (*N-way merge)*
  - Can do it in one shot if $N < M$
- What if $N > M$?
  - Do it recursively

Example: External Sorting Using Sort-Merge (N > M)



M = 3
N = 4

How big does DB have to be for N>M? Assume size of memory is 4MB, size of block is 4KB, tuple = 100 bytes

- Phase 1:
  - Create *sorted runs of size M* each
  - Result: *N* sorted runs of size *M* blocks each

- Phase 2:
  - Merge the *N* runs (*N-way merge)*
  - Can do it in one shot if *N* < *M*

A. 4GB (40M tuples)

B. 400MB (4M tuples)

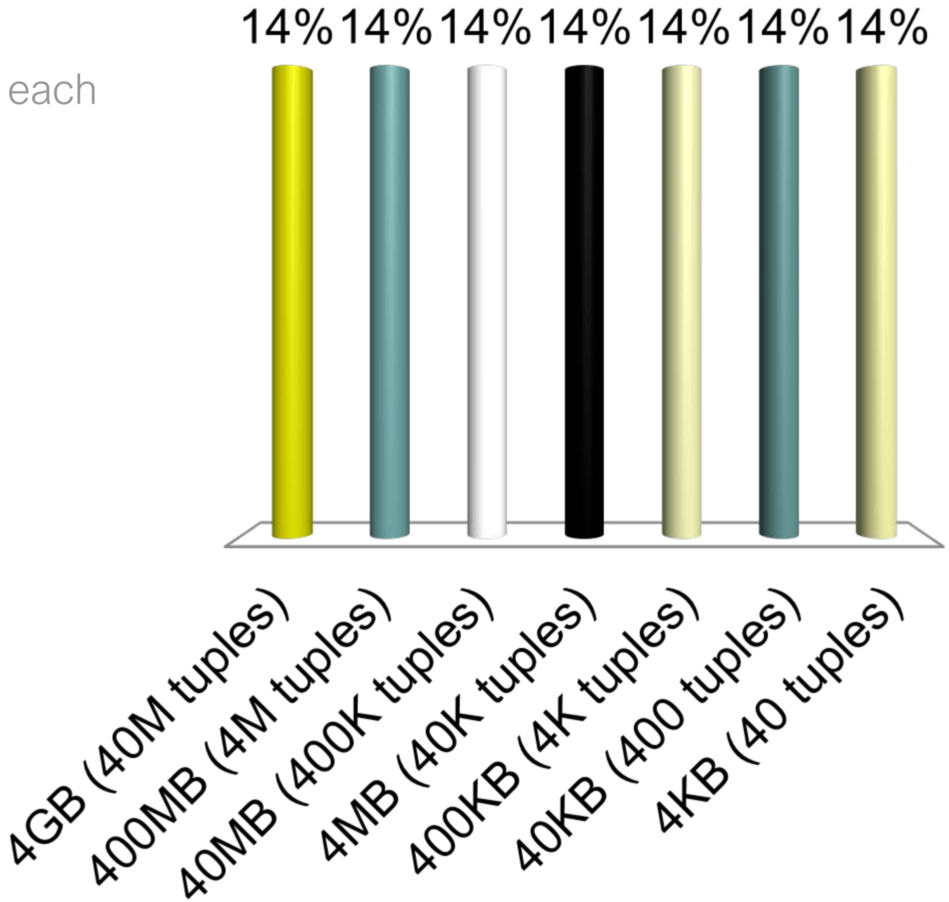
C. 40MB (400K tuples)

D. 4MB (40K tuples)

E. 400KB (4K tuples)

F. 40KB (400 tuples)

G. 4KB (40 tuples)



External sort-merge

- Phase 1:
  - Create *sorted runs of size M* each
  - Result: *N* sorted runs of size *M* blocks each

- Phase 2:
  - Merge the *N* runs (*N-way merge)*
  - Can do it in one shot if $N < M$

- What if $N > M$?
  - Do it recursively
  - Not expected to happen
  - If M = 1000, can compare 1000 runs
    - (4KB blocks): can sort: 1000 runs, each of 1000 blocks, each of 4k bytes = 4GB of data

External Merge Sort (Cont.)

- Cost analysis (according to book):
  - Total number of merge passes required: $\lceil \log_{M-1}(b_r/M) \rceil$.
  - Disk for initial run creation as well as in each pass is $2b_r$
    - for final pass, we don't count write cost
      - output may be *pipelined* (sent via memory to parent operation)

Thus total number of disk transfers for external sorting:

$$b_r (2 \lceil \log_{M-1}(b_r / M) \rceil + 1)$$

Seeks:

$$2 \lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2 \lceil \log_{M-1}(b_r / M) \rceil - 1)$$

b_b is #blocks read at a time, and how many output blocks needed

What is I/O cost of Phase 1? (Assume table is B blocks, memory is M blocks)

- Phase 1:
  - Create *sorted runs of size M* each
  - Result: *N* sorted runs of size *M* blocks each
- Phase 2:
  - Merge the *N* runs (*N-way merge)*
  - Can do it in one shot if *N* < *M*

A. 1 seek + B IOs

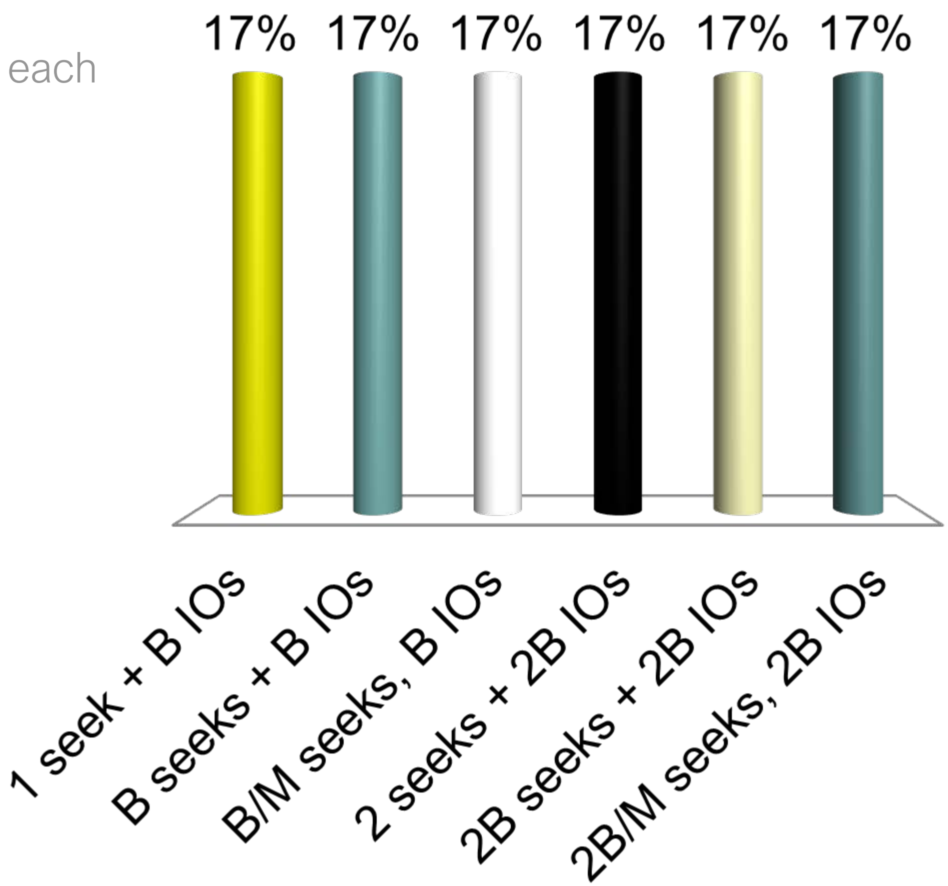
B. B seeks + B IOs

C. B/M seeks, B IOs

D. 2 seeks + 2B IOs

E. 2B seeks + 2B IOs

F. 2B/M seeks, 2B IOs



What is I/O cost of Phase 2? (Assume table is B blocks, memory is M blocks, 1 block at a time per run in memory, not writing out output to disk, N < M)

- Phase 1:
  - Create *sorted runs of size M* each
  - Result: *N* sorted runs of size *M* blocks each
- Phase 2:
  - Merge the *N* runs (*N-way merge)*
  - Can do it in one shot if *N* < *M*

A. 1 seek + B seq IOs

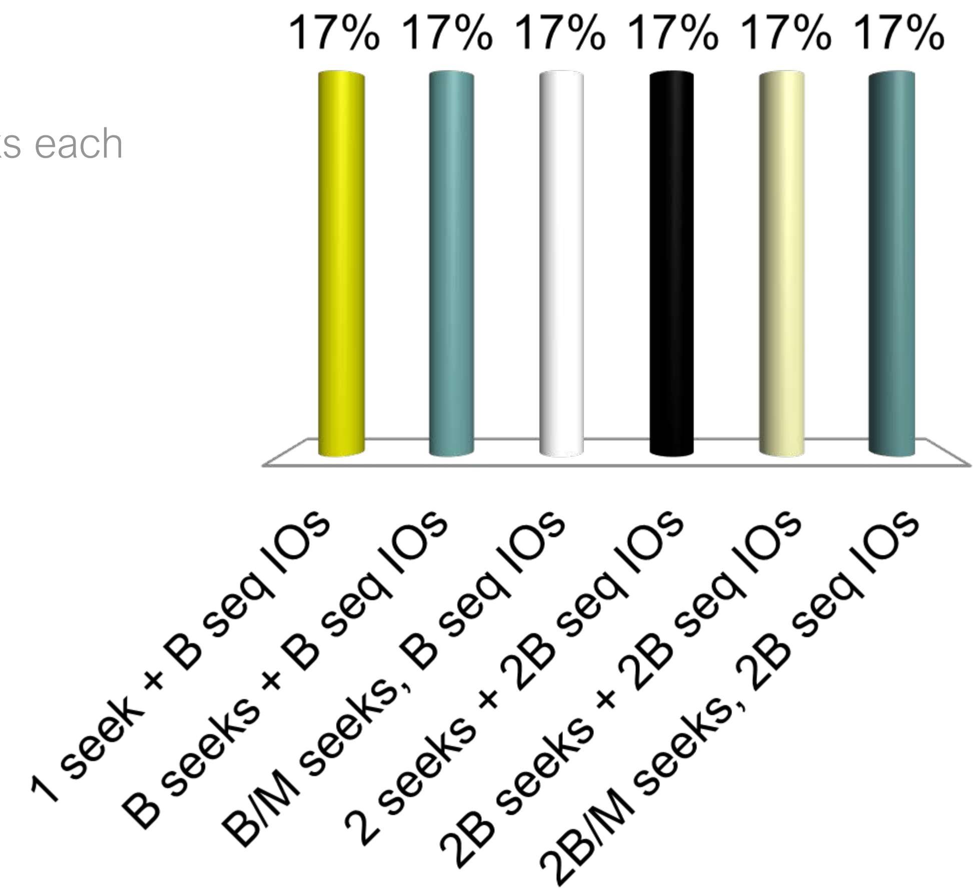
B. B seeks + B seq IOs

C. B/M seeks, B seq IOs

D. 2 seeks + 2B seq IOs

E. 2B seeks + 2B seq IOs

F. 2B/M seeks, 2B seq IOs



External Merge Sort (Cont.)

- Cost analysis (in reality):
  - Assume just two phases total (N < M)
  - B blocks of data, M blocks in memory → phase 1 creates B/M runs
  - Phase 1 needs to read and write every block → 2B IOs
    - Also two seeks for each run --- one to read in and one to write out
      - So if B/M runs, there are 2* B/M seeks
  - Phase 2 needs to read every block once
    - Each of those reads is a new seek
      - So total of B seeks and B IOs
  - Total cost adds phase 1 and phase 2 together
    - So total cost is B+2B/M seeks, 3B IOs
    - Note that B is the same thing as b_r from the book
  - Compare with book formula (!):
    - $2\lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2 \lceil \log_{M-1}(b_r / M) \rceil - 1)$ seeks + $b_r (2 \lceil \log_{M-1}(b_r / M) \rceil + 1)$ IOs

Join

- *select * from R, S where R.a = S.a*
  - Called an "*equi-join*"
- *select * from R, S where |R.a – S.a | < 0.5*
  - *Not an "equi-join"*

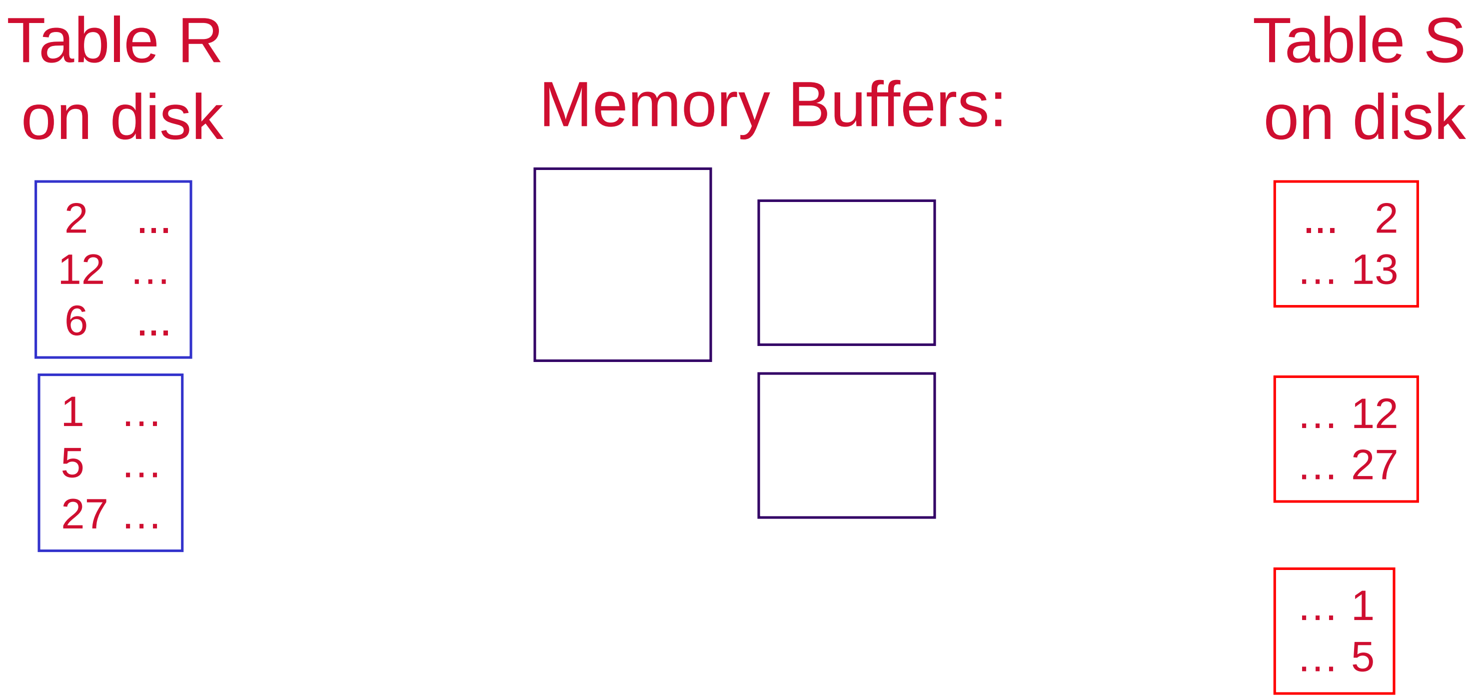
- Option 1: Nested-loops

  for each tuple r in R

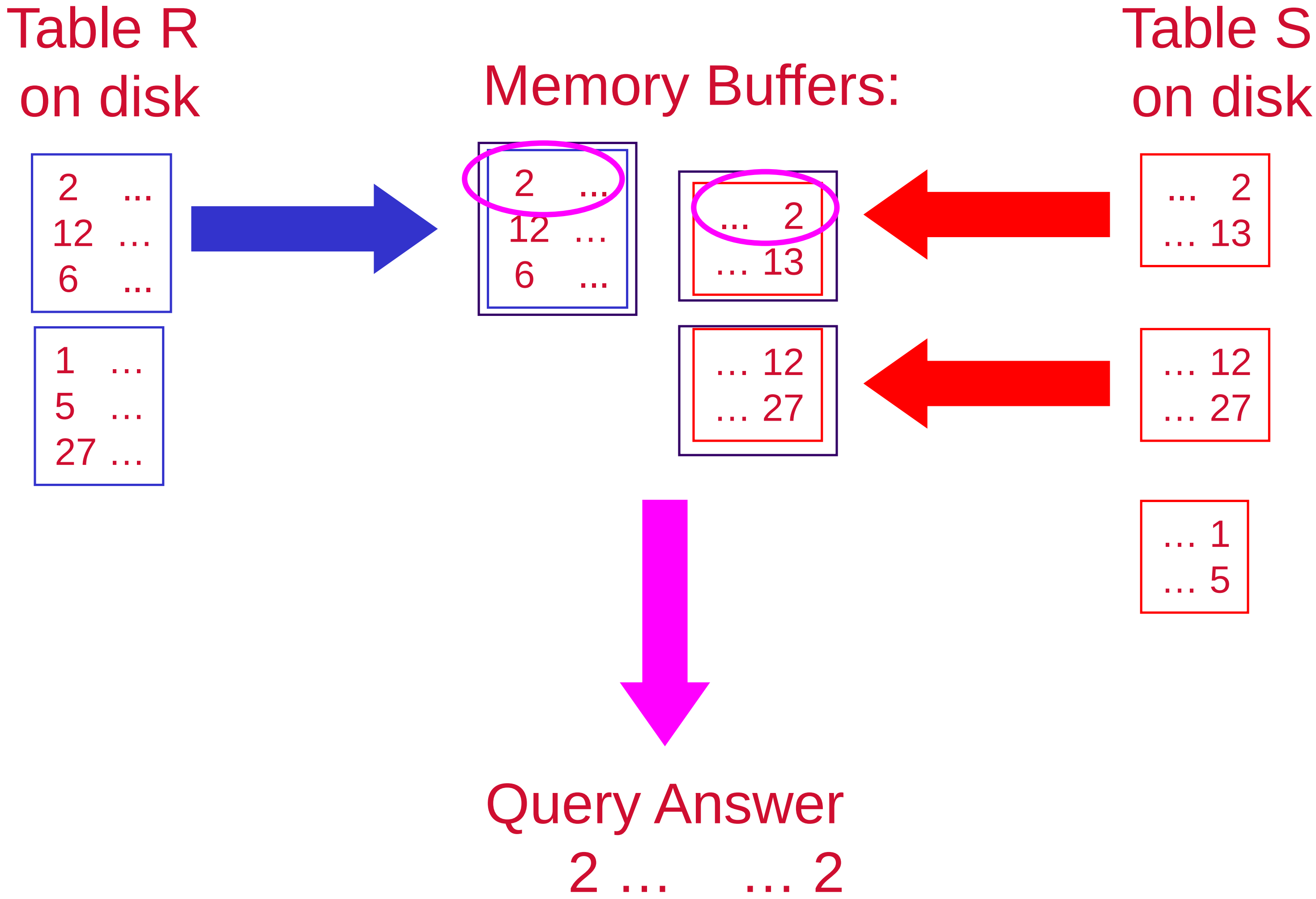
      for each tuple s in S

          check if r.a = s.a (or whether |r.a – s.a| < 0.5)
- Can be used for any join condition
  - As opposed to some algorithms we will see later
- R called *outer relation*
- S called *inner relation*

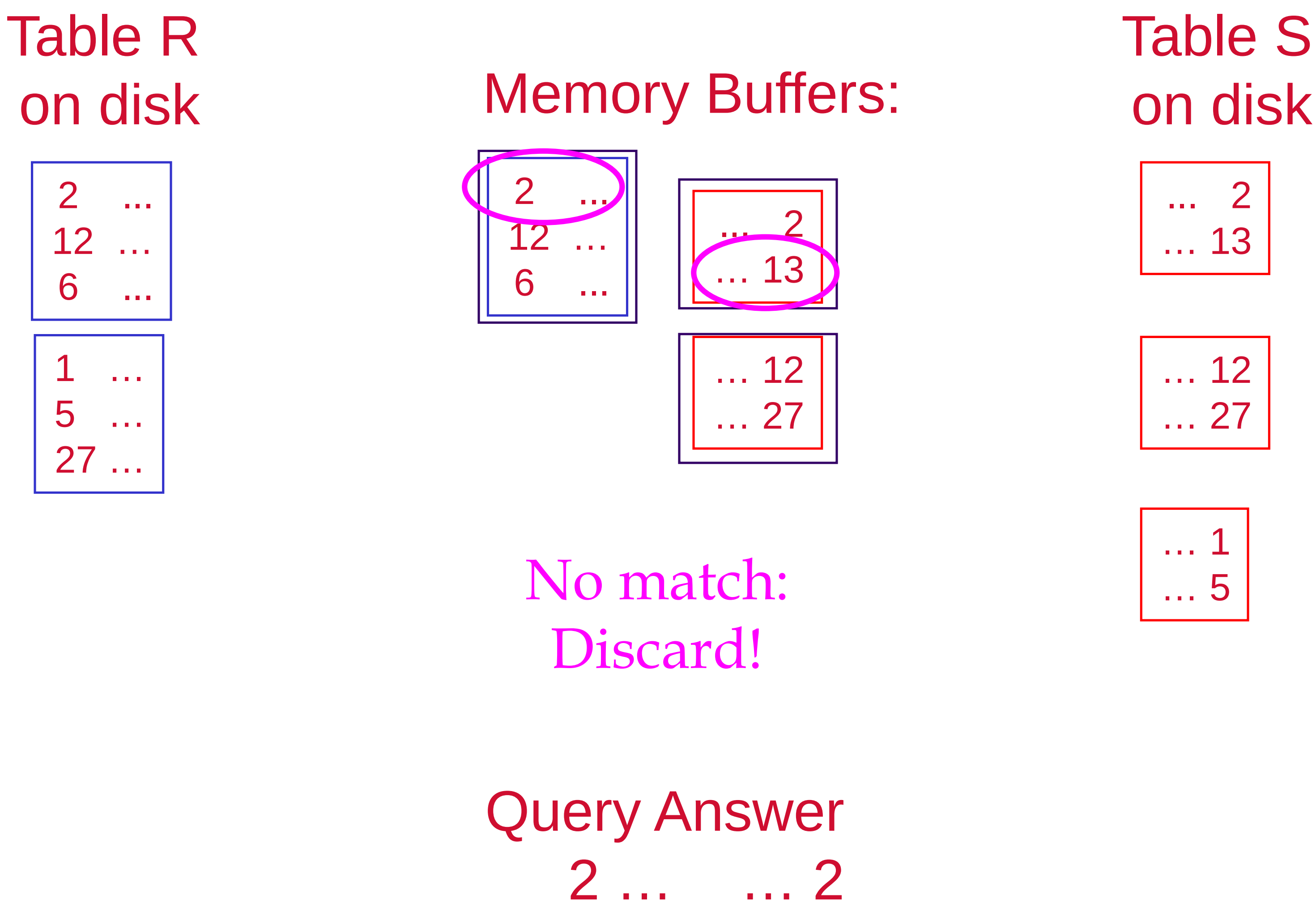
Nested Loops Join



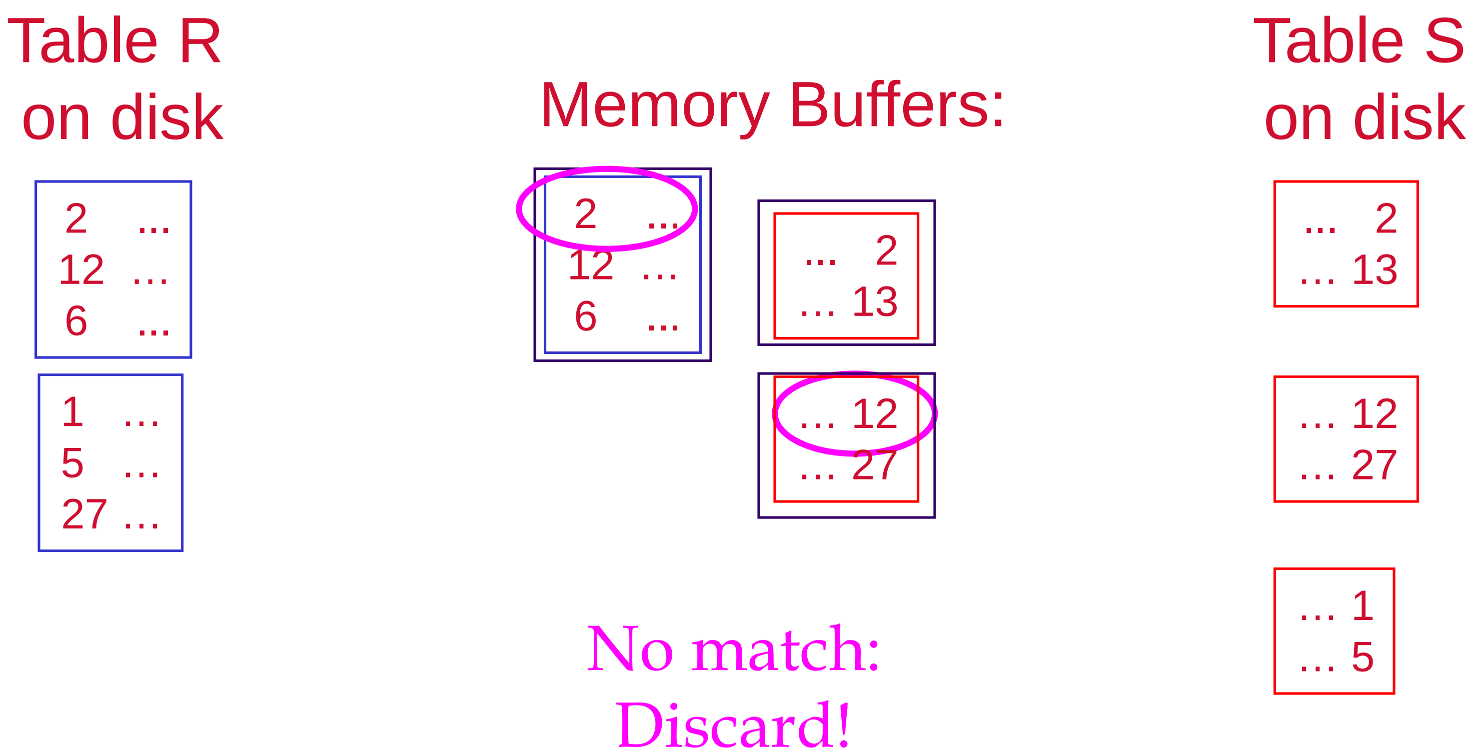
Nested Loops Join



Nested Loops Join



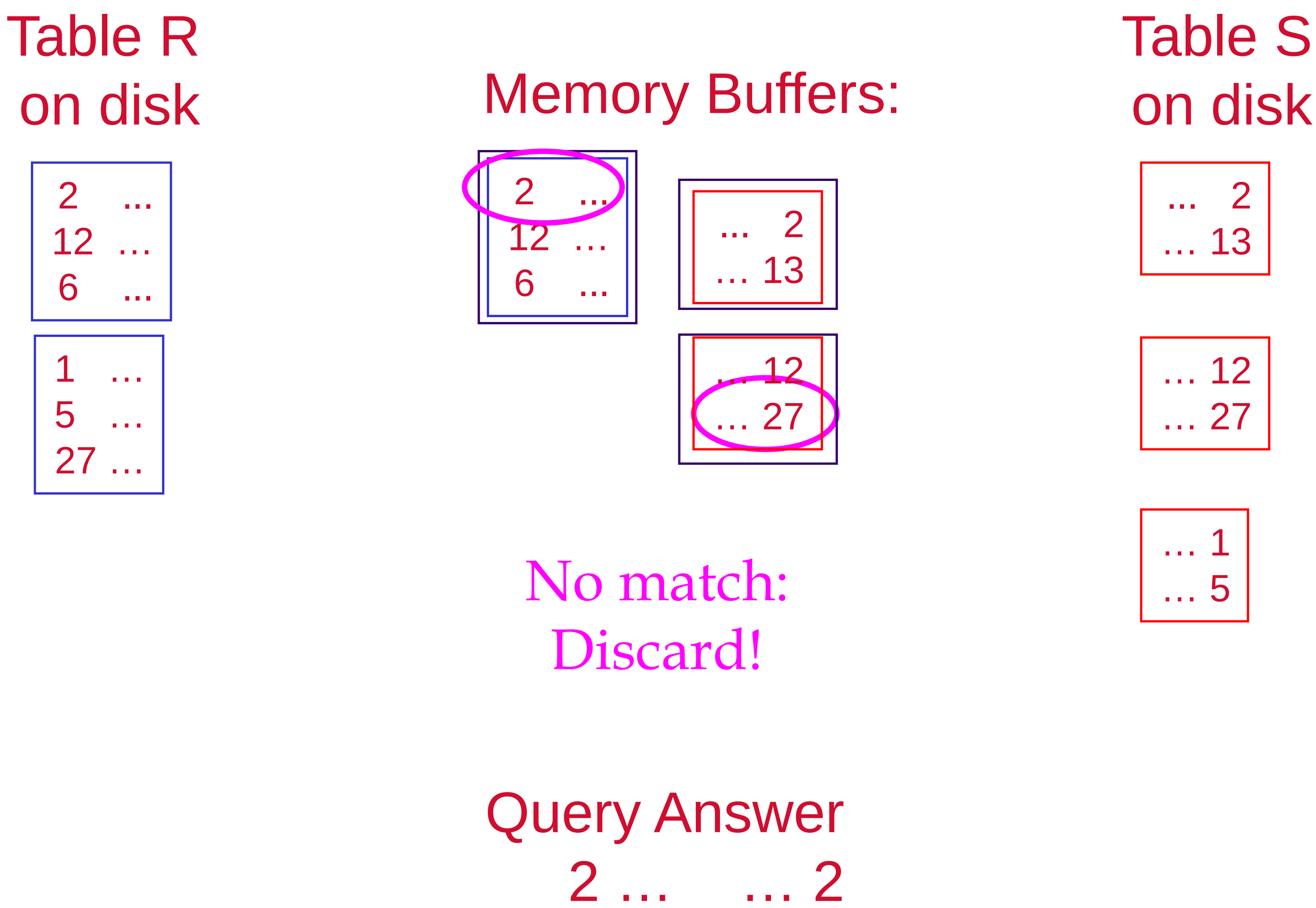
Nested Loops Join



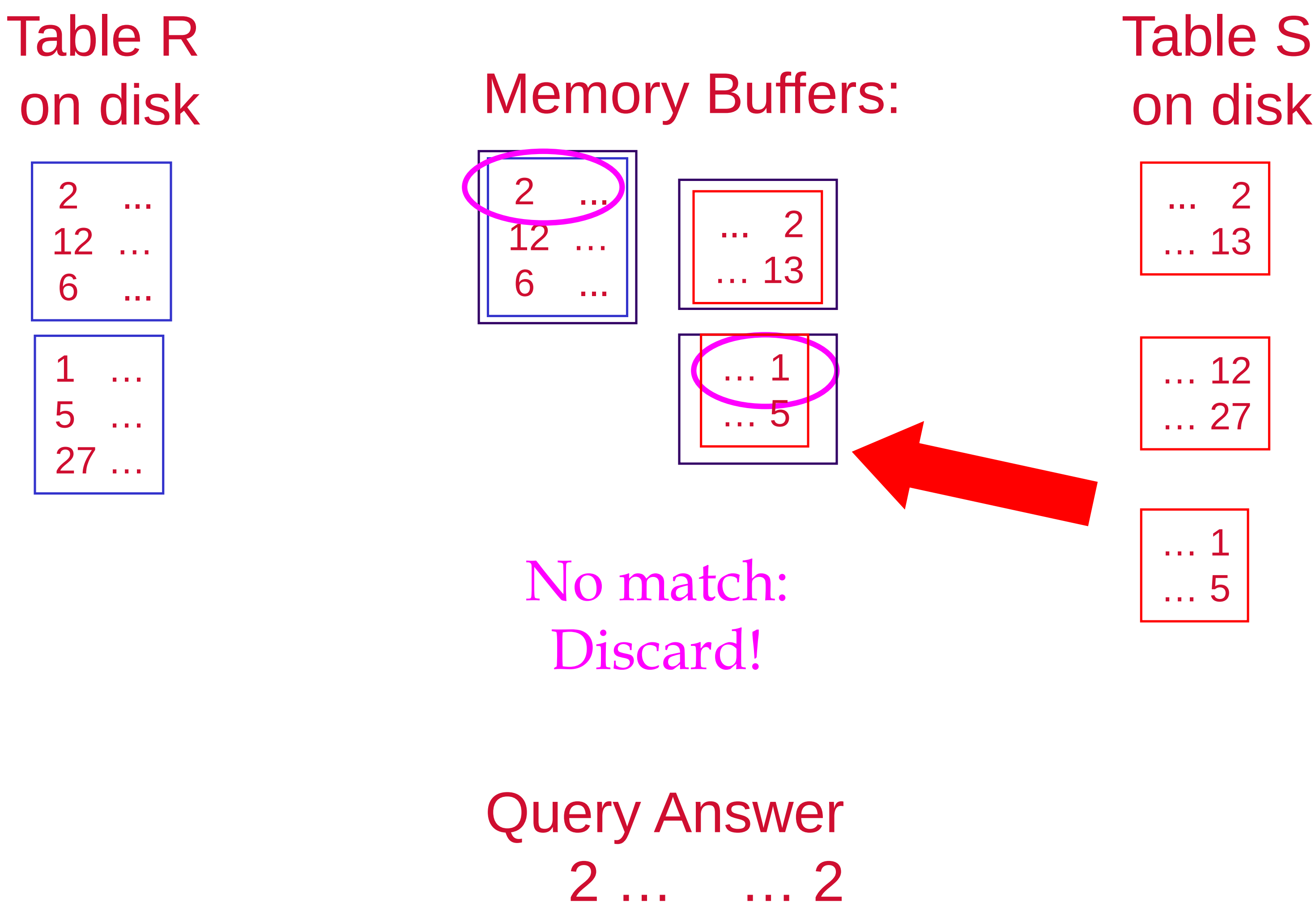
Query Answer

2 2

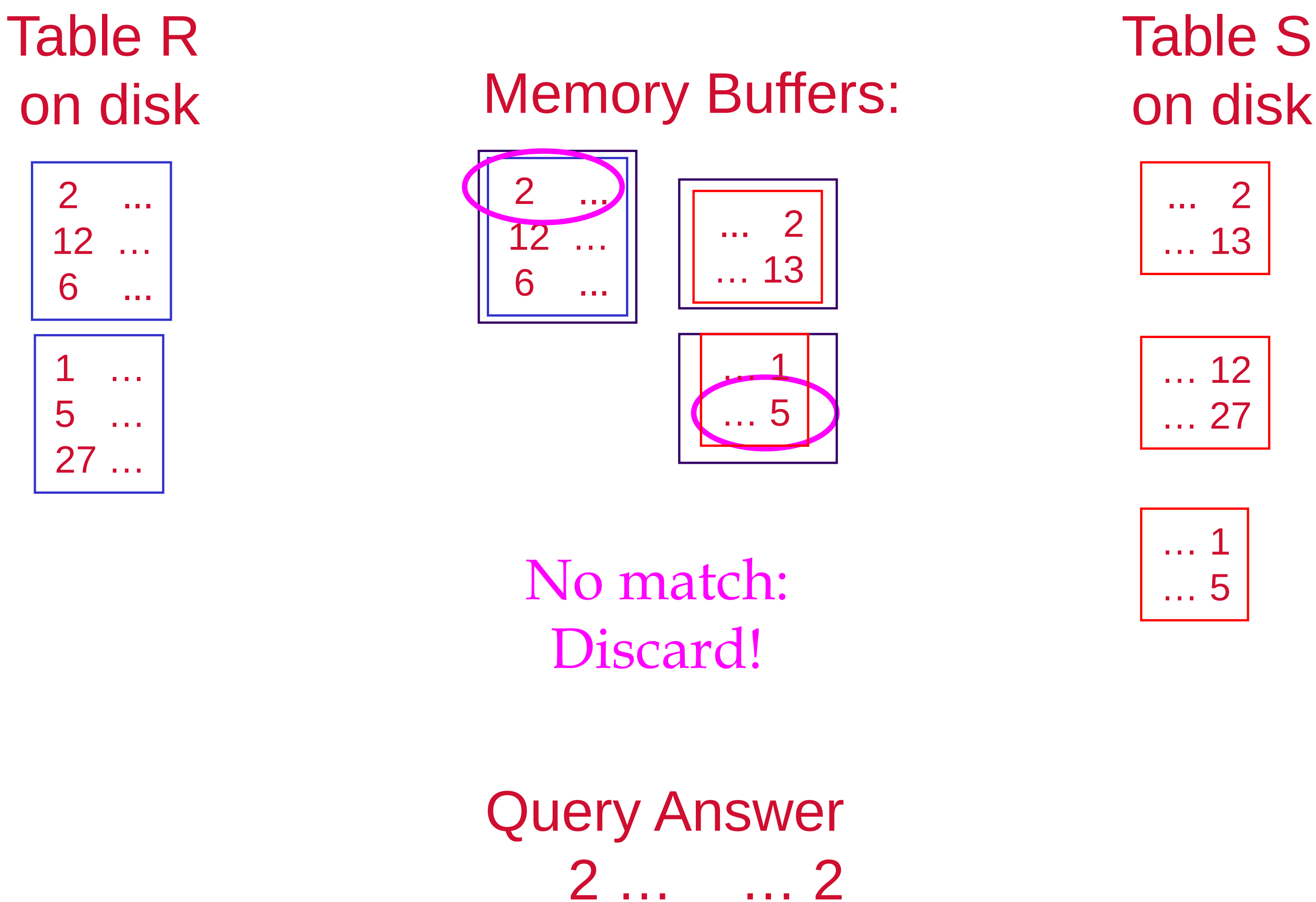
Nested Loops Join



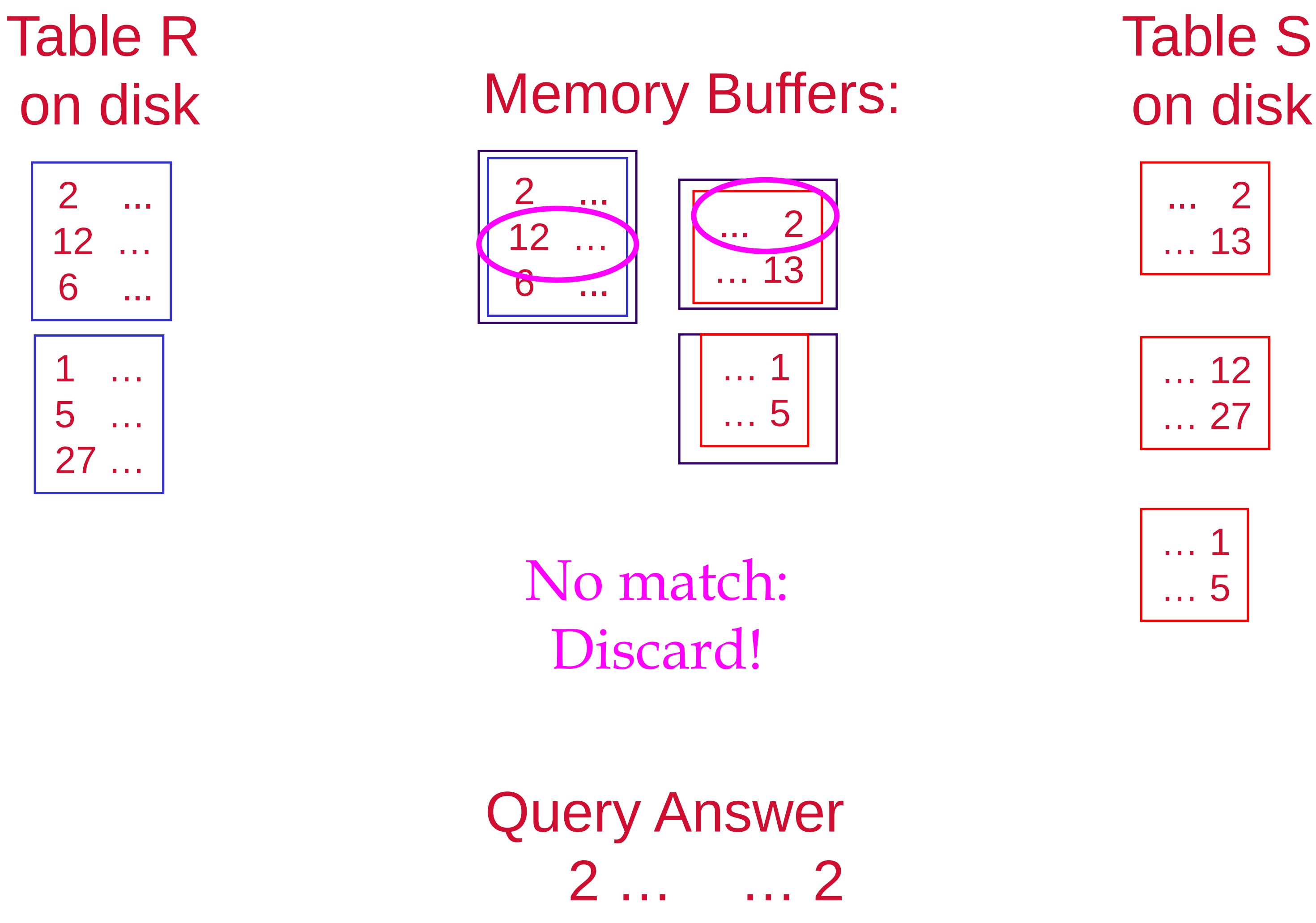
Nested Loops Join



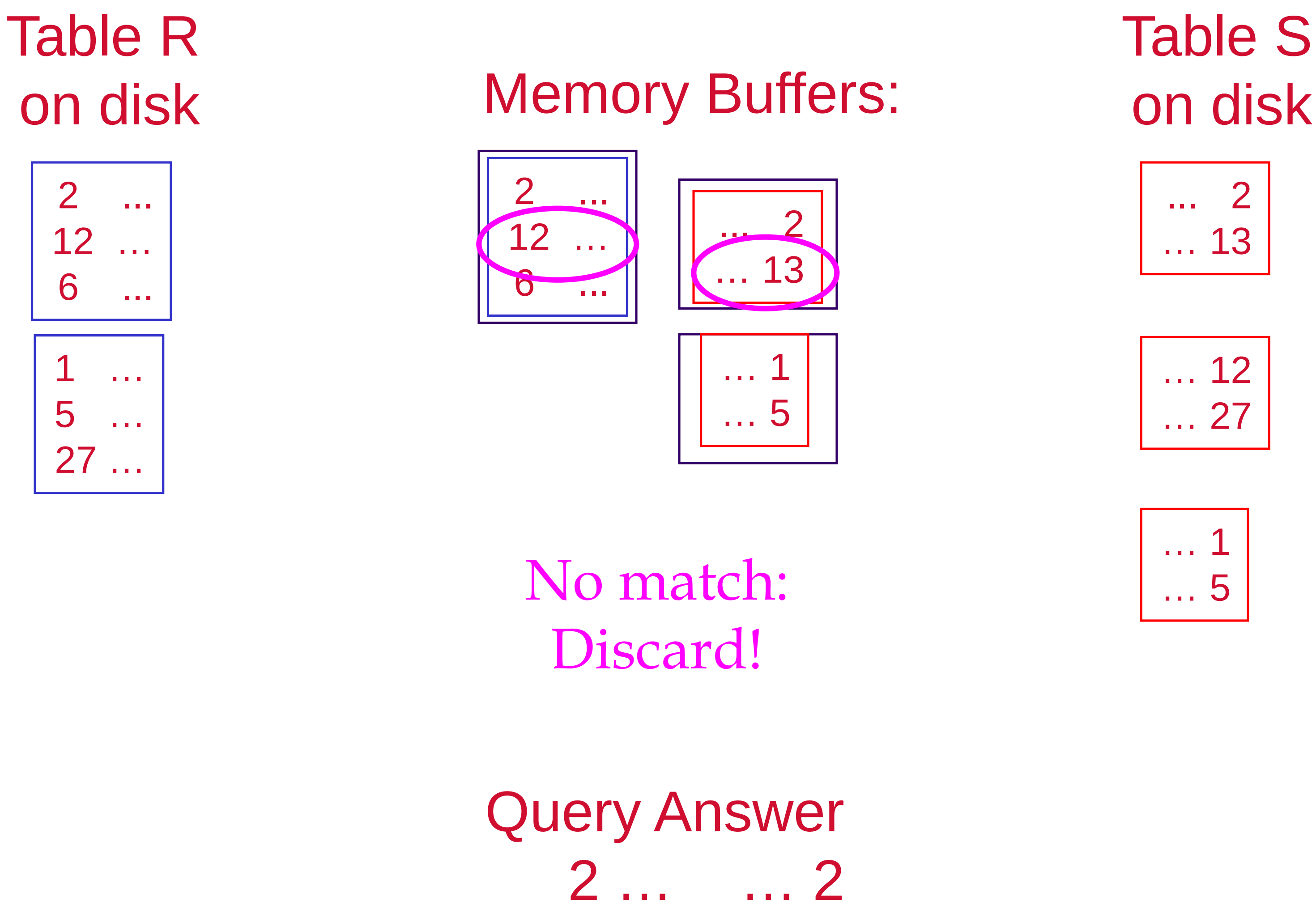
Nested Loops Join



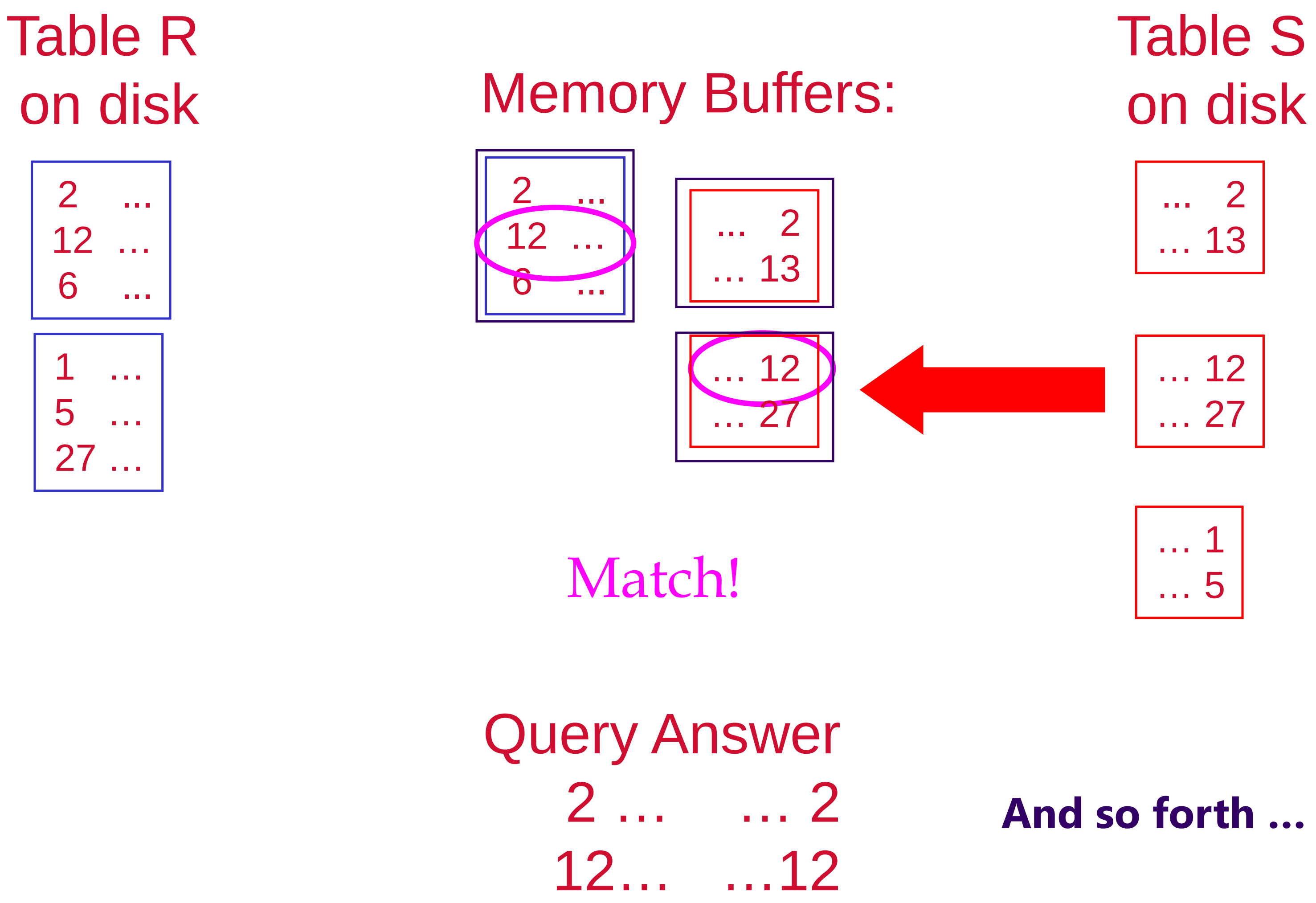
Nested Loops Join



Nested Loops Join



Nested Loops Join



How many blocks of S are read into memory from disk in the previous example?

$M = 4$, $b_r = 2$, $b_s = 3$, n_r, n_s *are both 6*

A. 2

B. 3

C. 6

D. 7

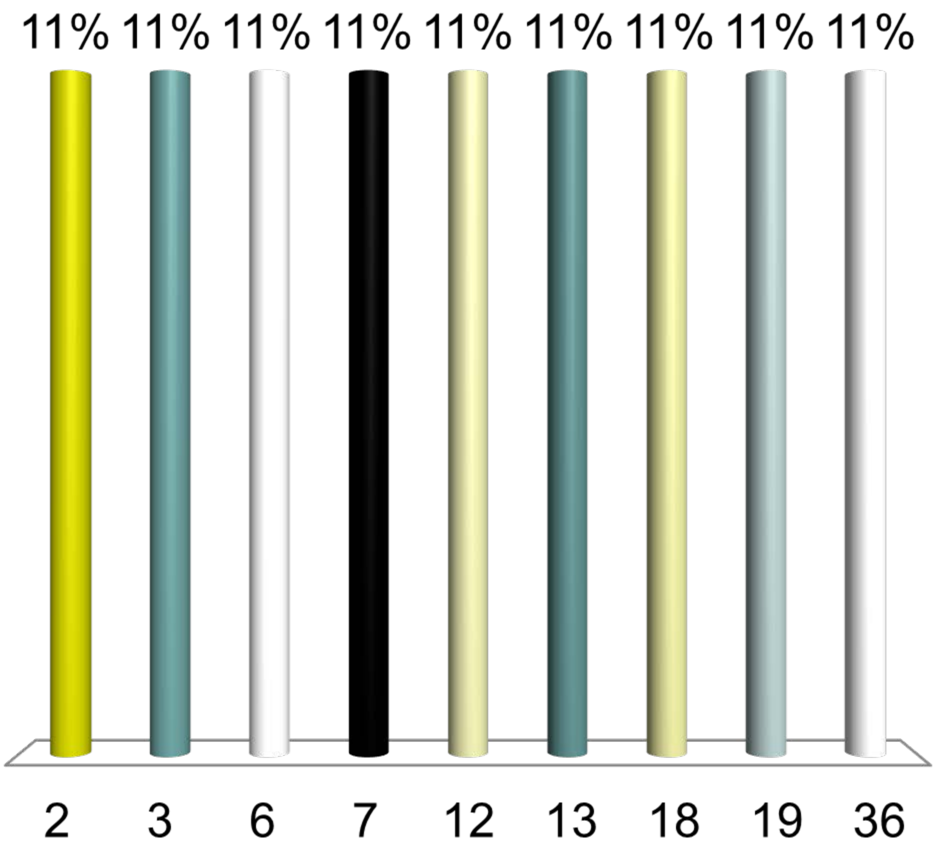
E. 12

F. 13

G. 18

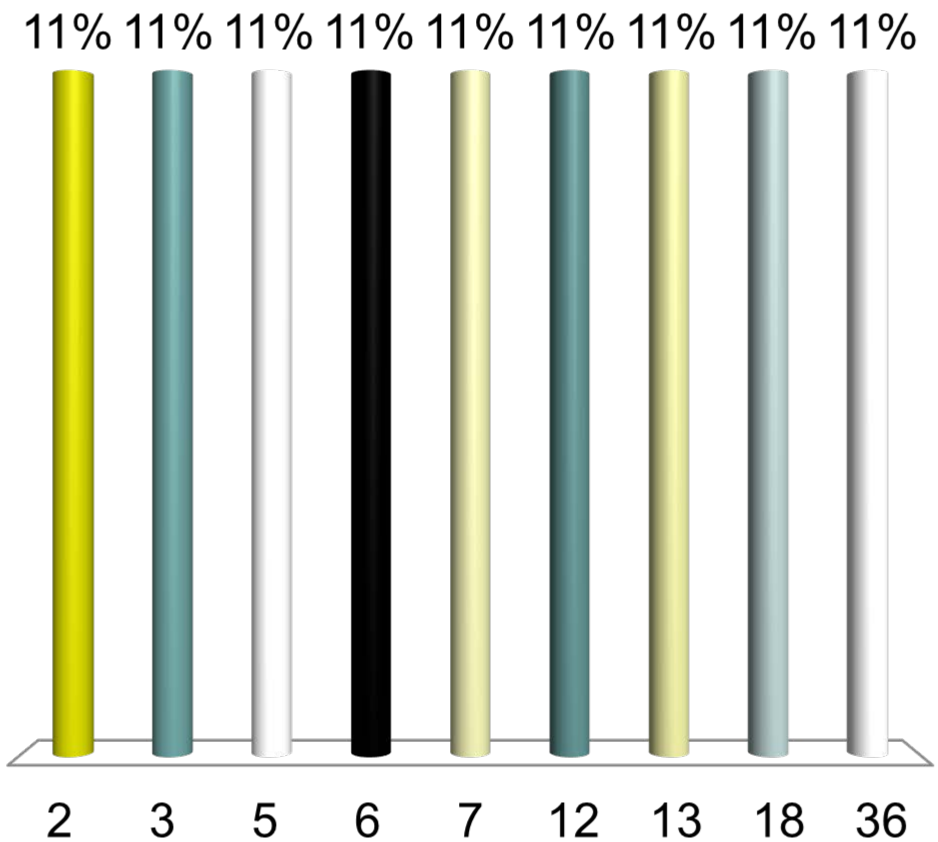
H. 19

I. 36



How many blocks of R are read into memory from disk in the previous example? $M = 3$ or 4, $b_r = 2$, $b_s = 3$, n_r, n_s *are both 6*

A. 2
B. 3
C. 5
D. 6
E. 7
F. 12
G. 13
H. 18
I. 36



How many seeks in the previous example?

M = 3 or 4, $b_r = 2$, $b_s = 3$, n_r, n_s *are both 6*

A. 2

B. 3

C. 5

D. 6

E. 7

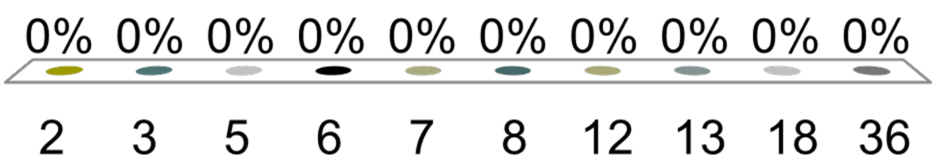
F. 8

G. 12

H. 13

I. 18

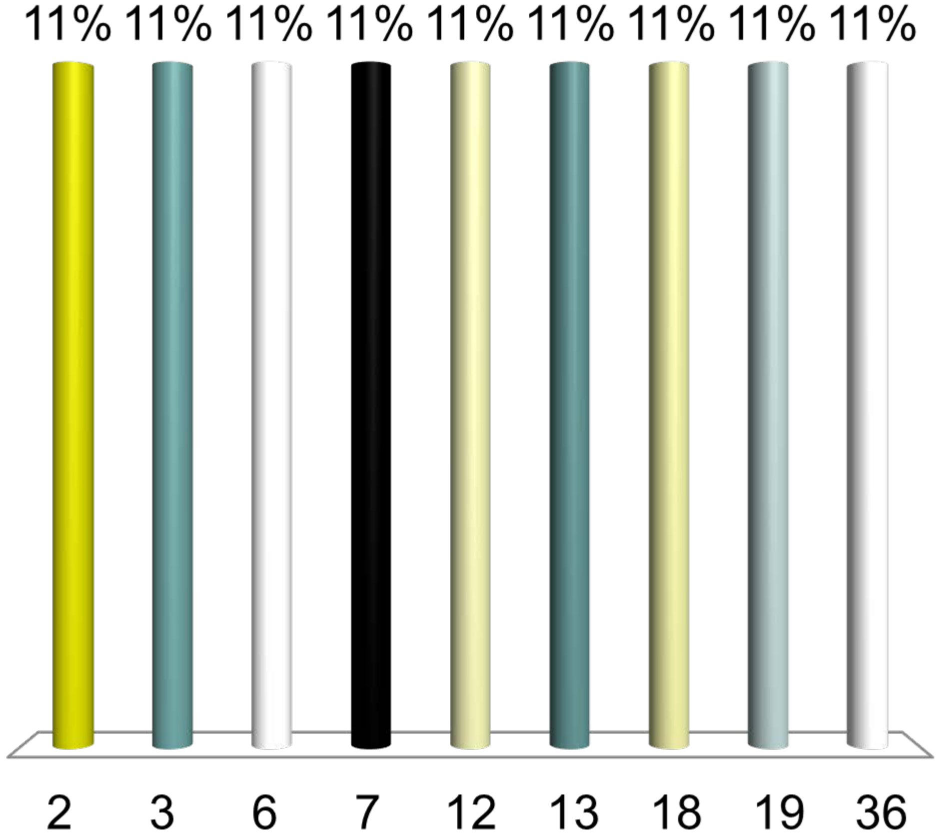
J. 36



How many blocks of S are read into memory from disk in the previous example if we had one less block of memory?

M = 3, b_r = 2, b_s = 3, n_r, n_s are both 6

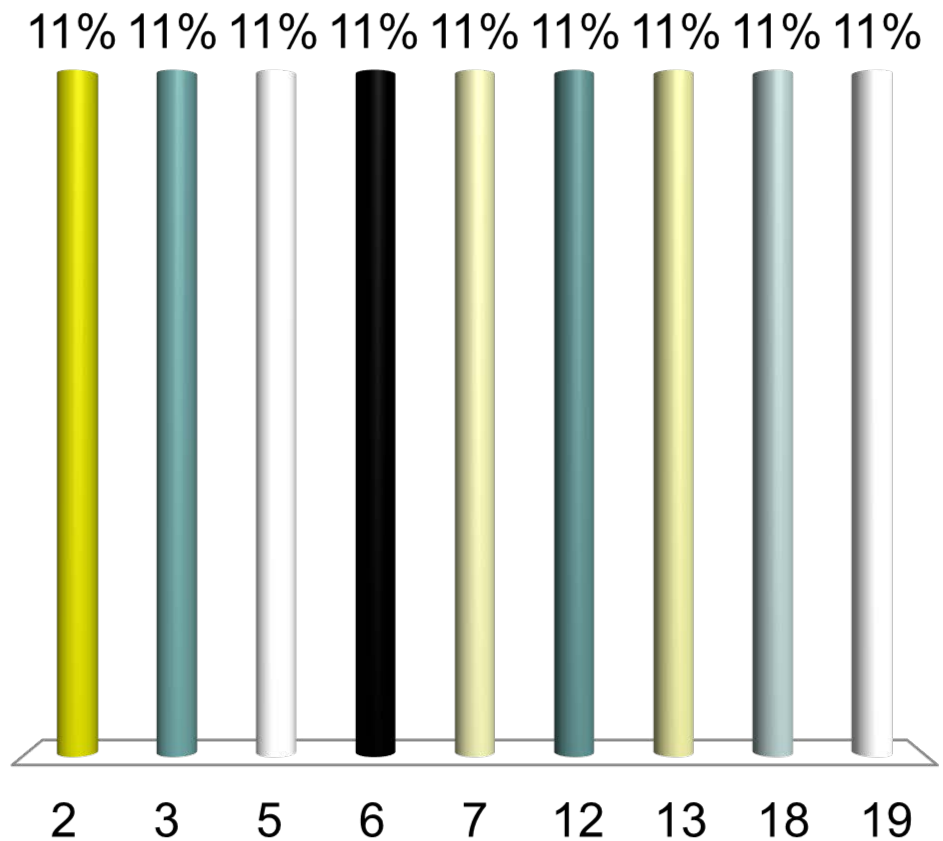
A. 2
B. 3
C. 6
D. 7
E. 12
F. 13
G. 18
H. 19
I. 36



How many blocks of S are read into memory from disk in the previous example if we had one more block of memory?

$M = 5$, $b_r = 2$, $b_s = 3$, n_r, n_s *are both 6*

A. 2
B. 3
C. 5
D. 6
E. 7
F. 12
G. 13
H. 18
I. 19



Nested-loops Join

not using indexes

- Cost ? Depends on the actual values of parameters, especially memory
- b_r, b_s → *Number of blocks of R and S*
- n_r, n_s → *Number of tuples of R and S*
- Case 1: Minimum memory required = 3 blocks
  - One to hold the current *R* block, one for current S block, one for the result being produced
  - Blocks transferred:
    - Must scan *R* tuples once: b_r
    - For each *R* tuple, must scan *S*: $n_r * b_s$
  - Seeks ?
    - $n_r + b_r$

Nested-loops Join

- Case 1: Minimum memory required = 3 blocks
  - Blocks transferred: $n_r * b_s + b_r$
  - Seeks: $n_r + b_r$
- Example:
  - Number of records -- R: $n_r = 10{,}000$, S: $n_s = 5000$
  - Number of blocks -- R: $b_r = 400$, S: $b_s = 100$
- Then for R "outer relation":
  - blocks transferred: $n_r * b_s + b_r$ = 10000 * 100 + 400 = 1,000,400
  - seeks: 10400
  - time: 1000400 t_T + 10400 t_S = 1000400(.1ms) + 10400(4ms) = 1020.8 sec
- What if S outer relation?
  - 5000 * 400 + 100 = 2,000,100 block transfers,
  - 5100 seeks
  - = 2000100 t_T + 5100 t_S = 2041.7sec

Order matters!

Nested-loops Join

- Case 2: *S* fits in memory
  - Blocks transferred: $b_s + b_r$
  - Seeks: *2*
- Example:
  - Number of records -- *R:* $n_r = 10{,}000$, *S:* $n_s = 5000$
  - Number of blocks -- *R:* $b_r = 400$, *S:* $b_s = 100$
- Then:
  - blocks transferred: 400 + 100 = 500
  - seeks: 2
  - 58 ms

Orders of magnitude difference

Block Nested-loops Join

- Simple modification to "nested-loops join" (block at a time)

for each block B_r in R

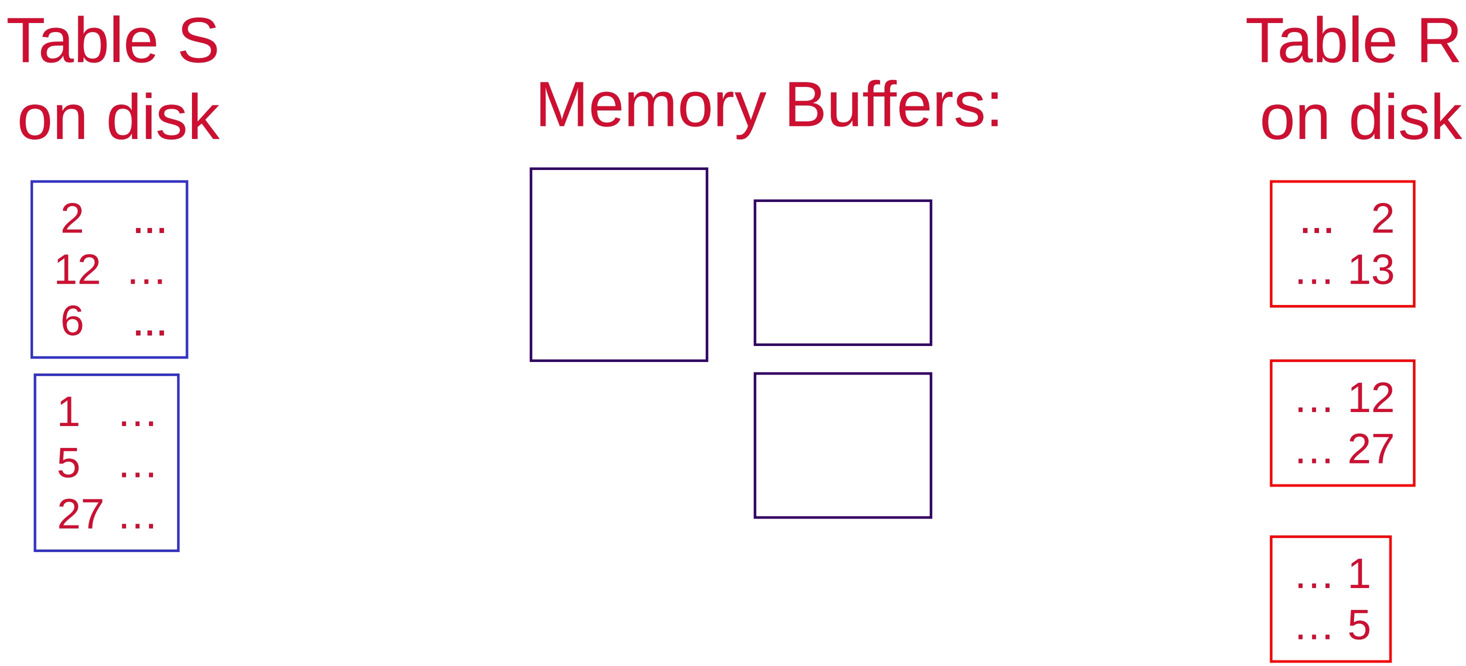
for each block B_s in S

for each tuple s in B_s

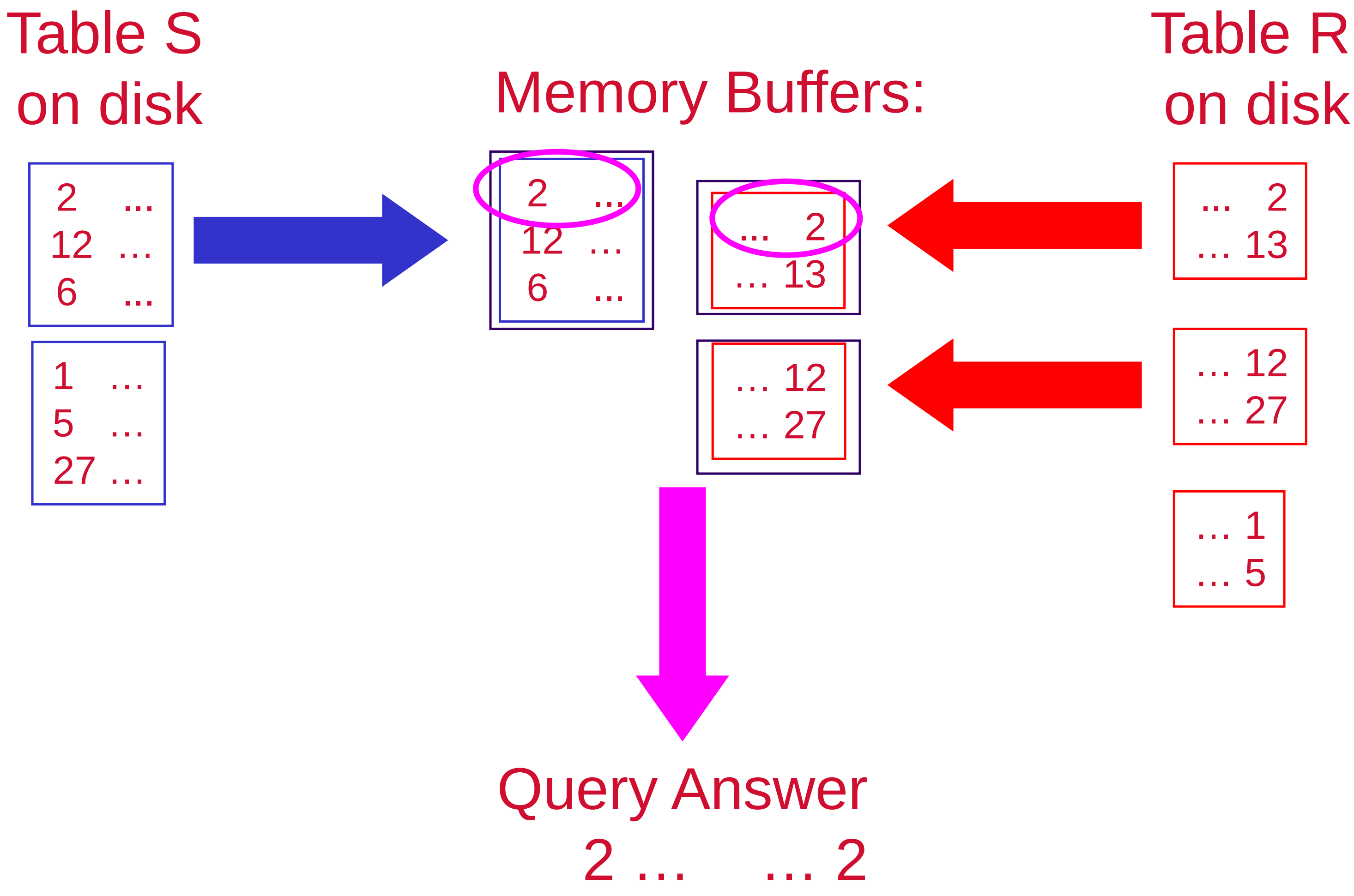
for each tuple r in B_r

check if r.a = s.a (or whether |r.a – s.a| < 0.5)

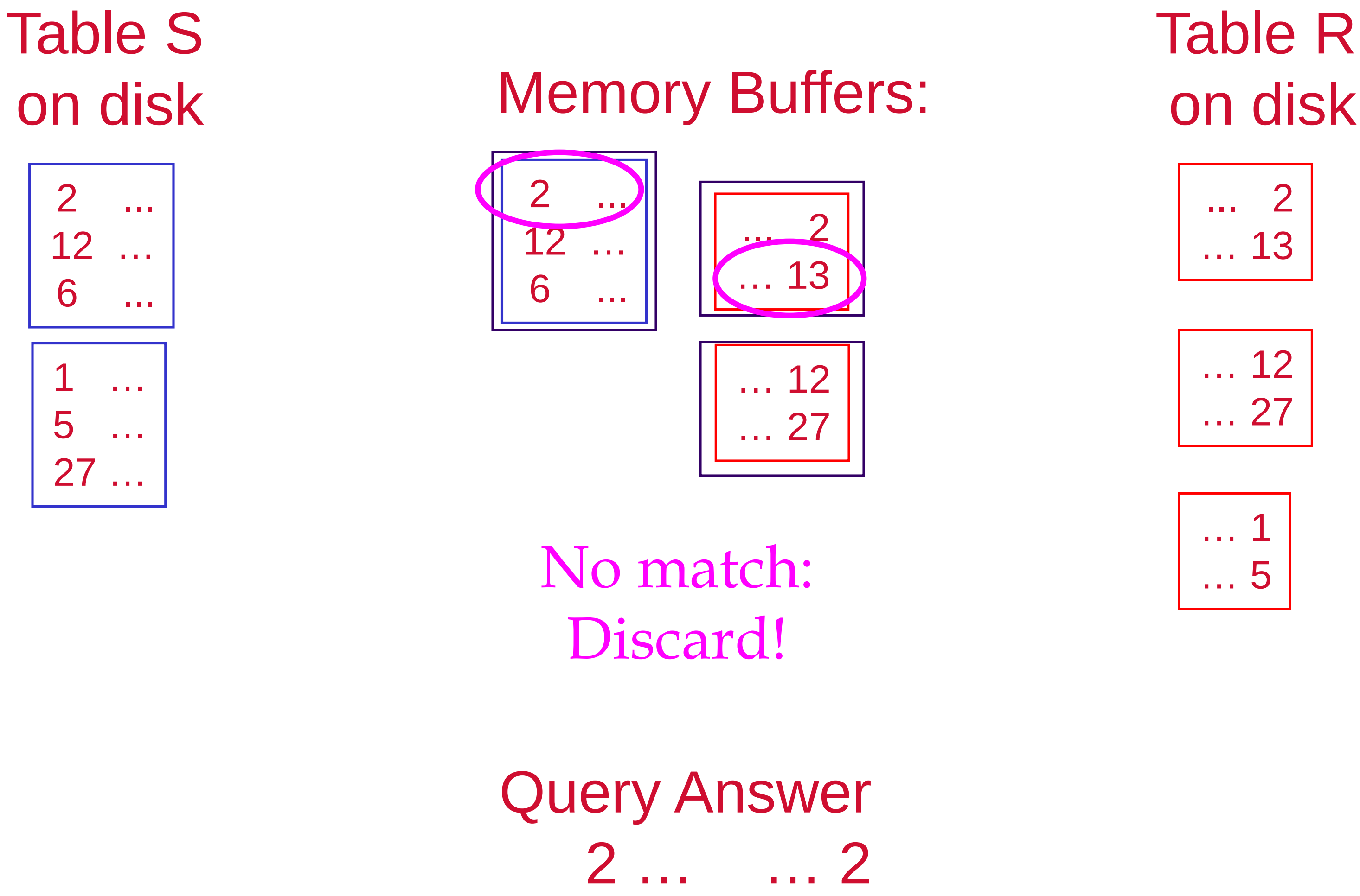
Block Nested Loops Join



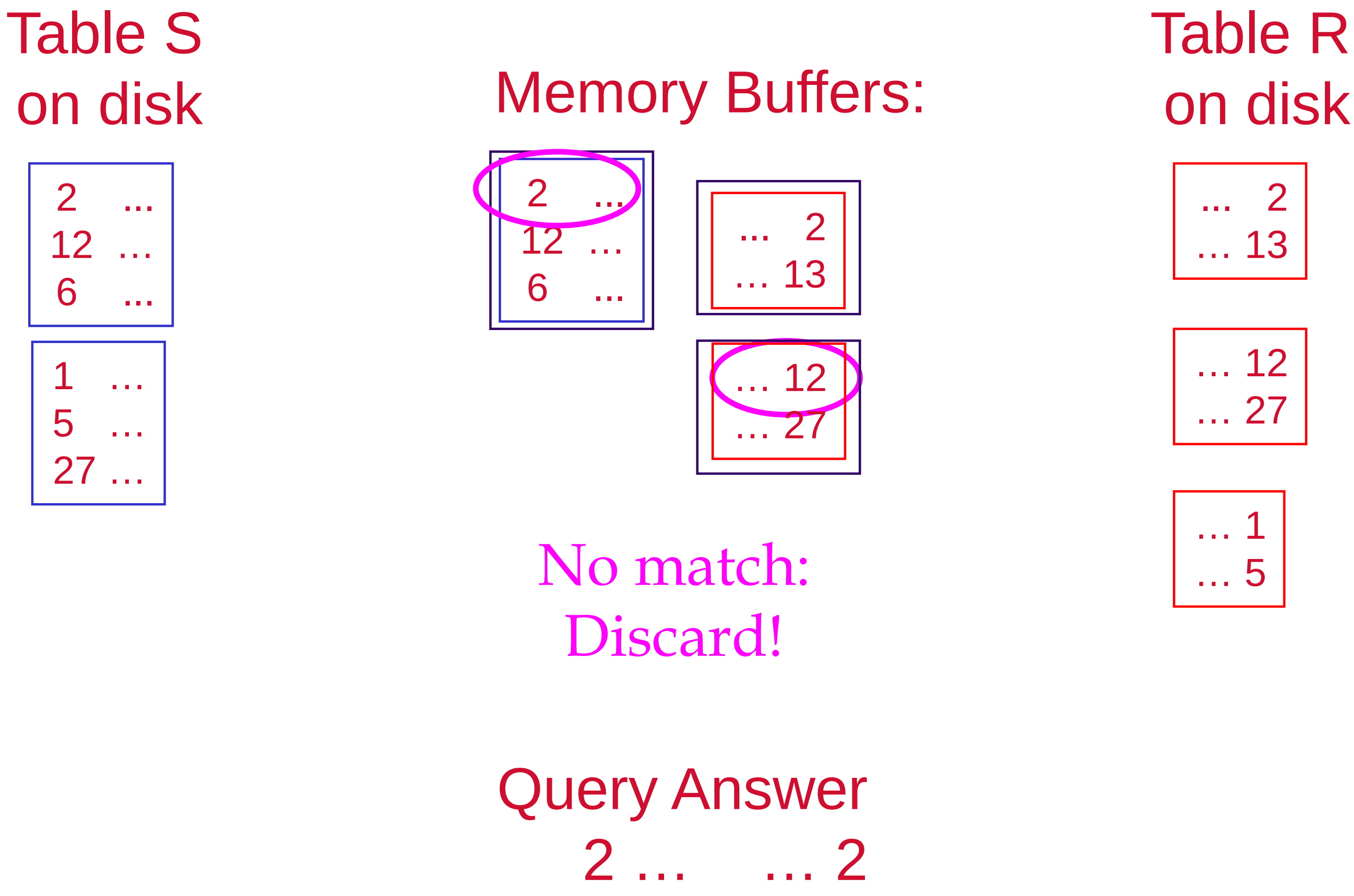
Block Nested Loops Join



Block Nested Loops Join



Block Nested Loops Join

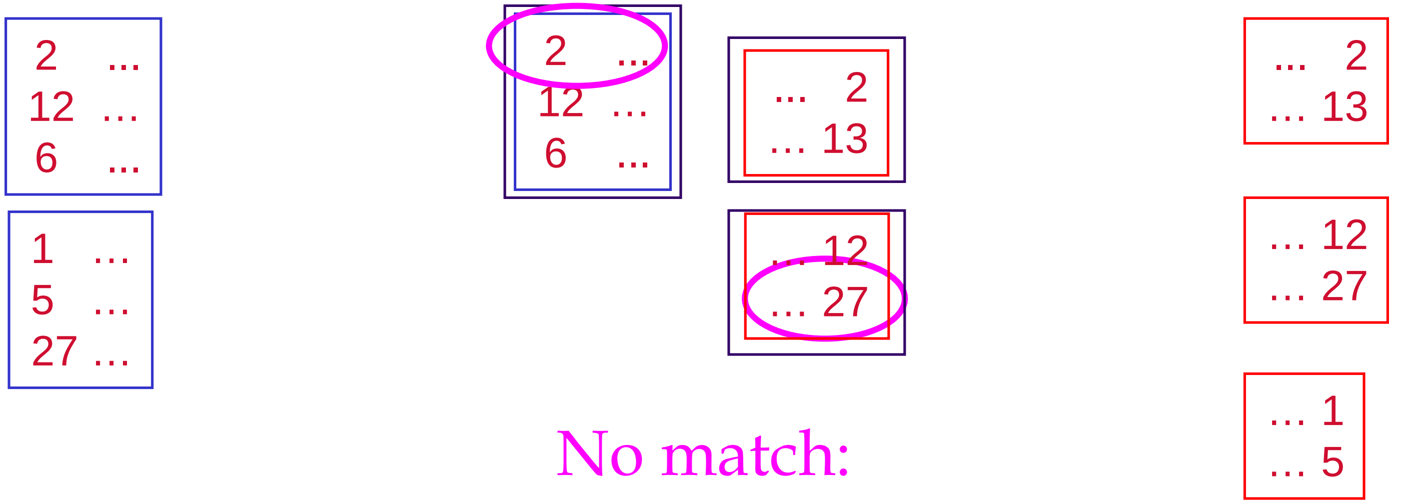


Block Nested Loops Join

Table S on disk

Memory Buffers:

Table R on disk

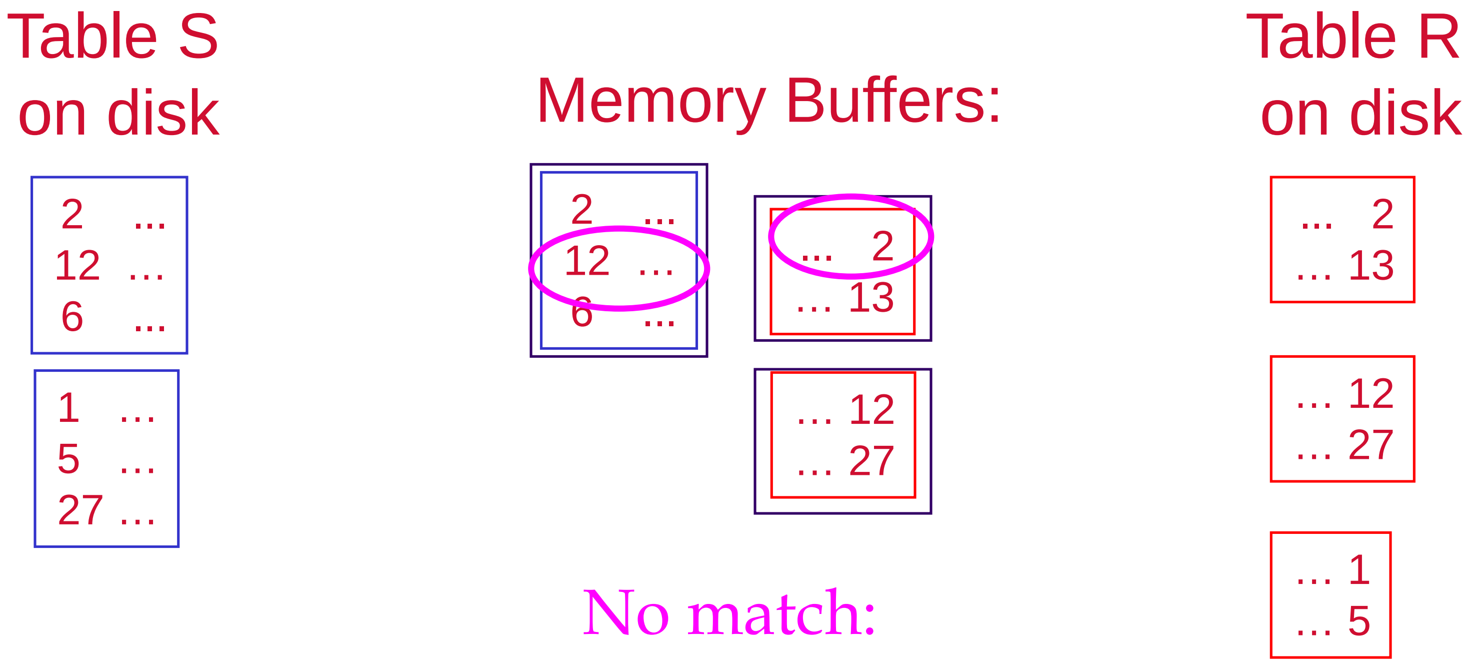


No match:
Discard!

Query Answer

2 2

Block Nested Loops Join

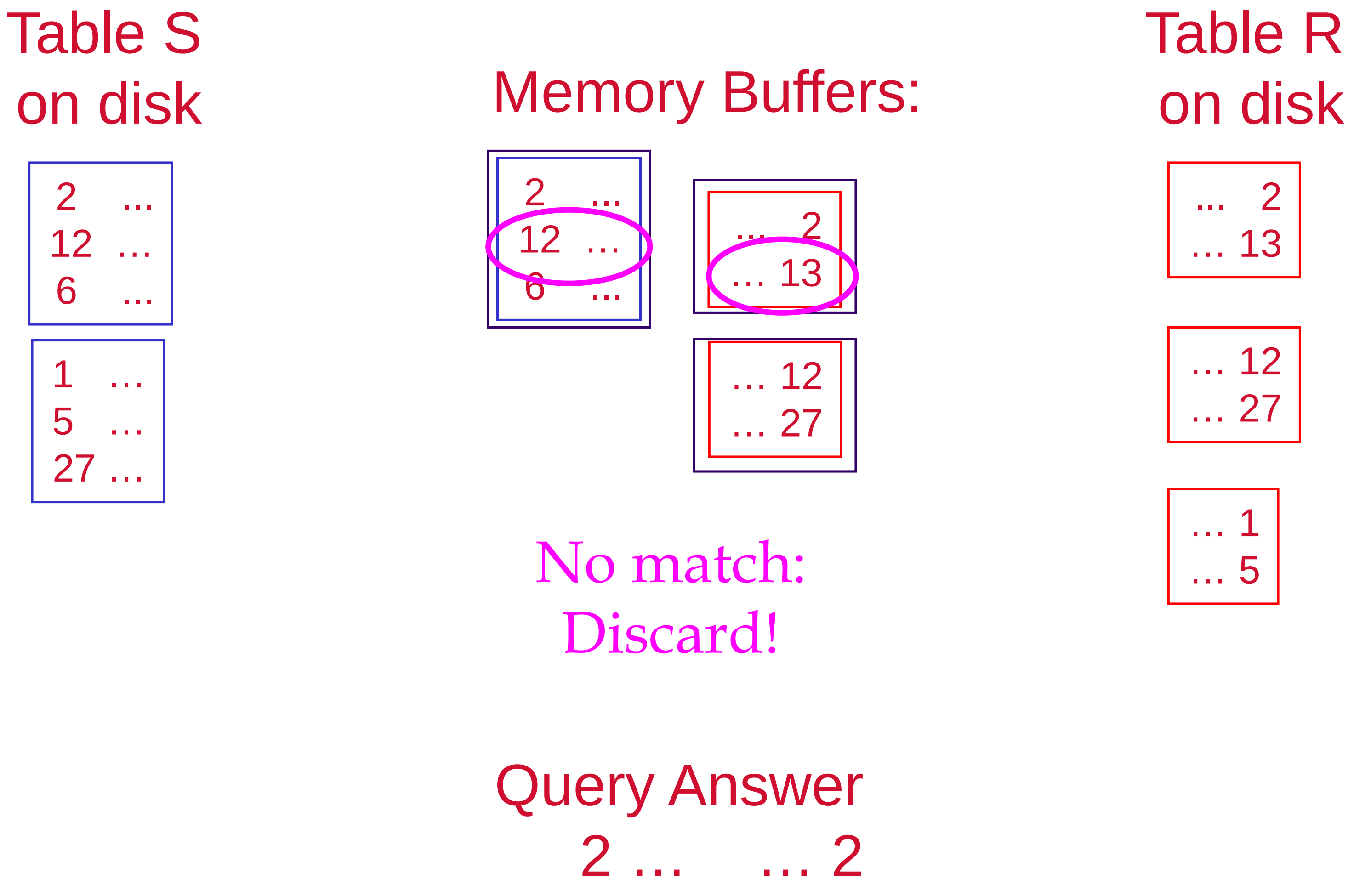


No match:
Discard!

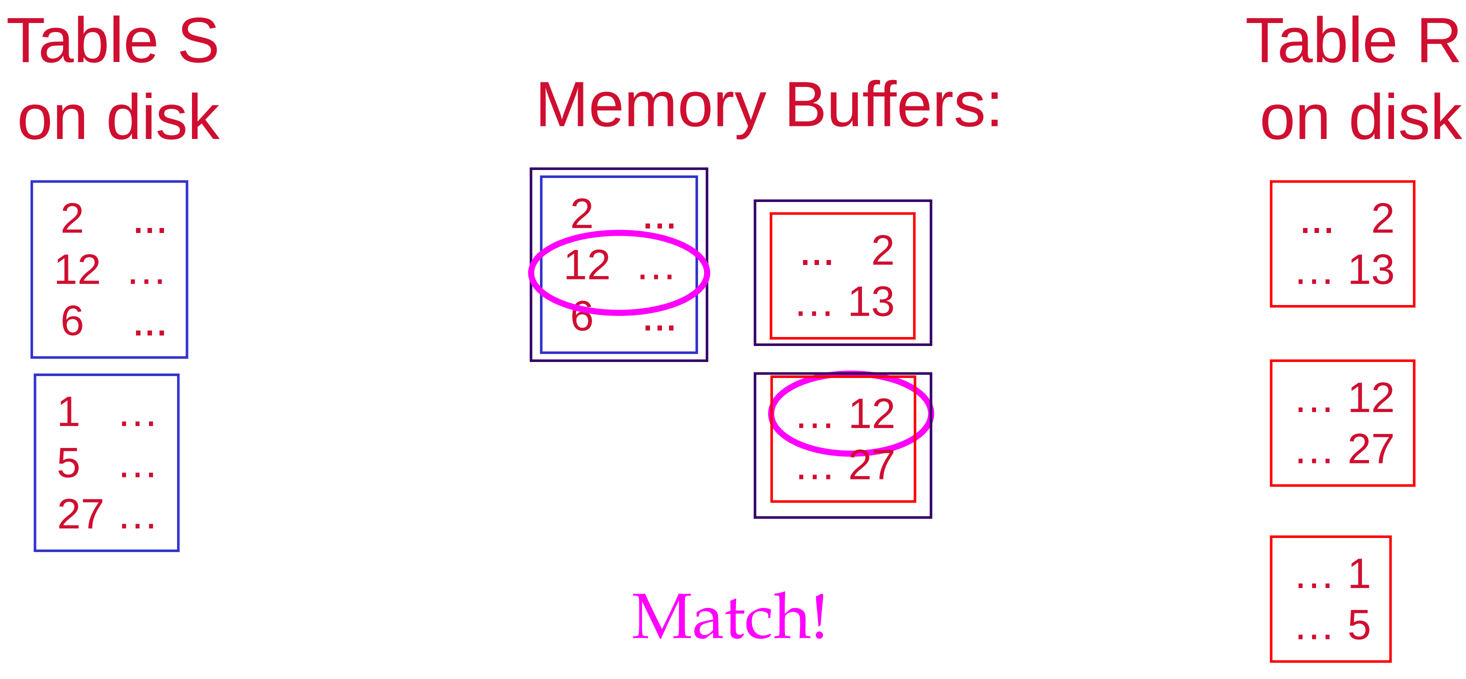
Query Answer

2 2

Block Nested Loops Join



Block Nested Loops Join

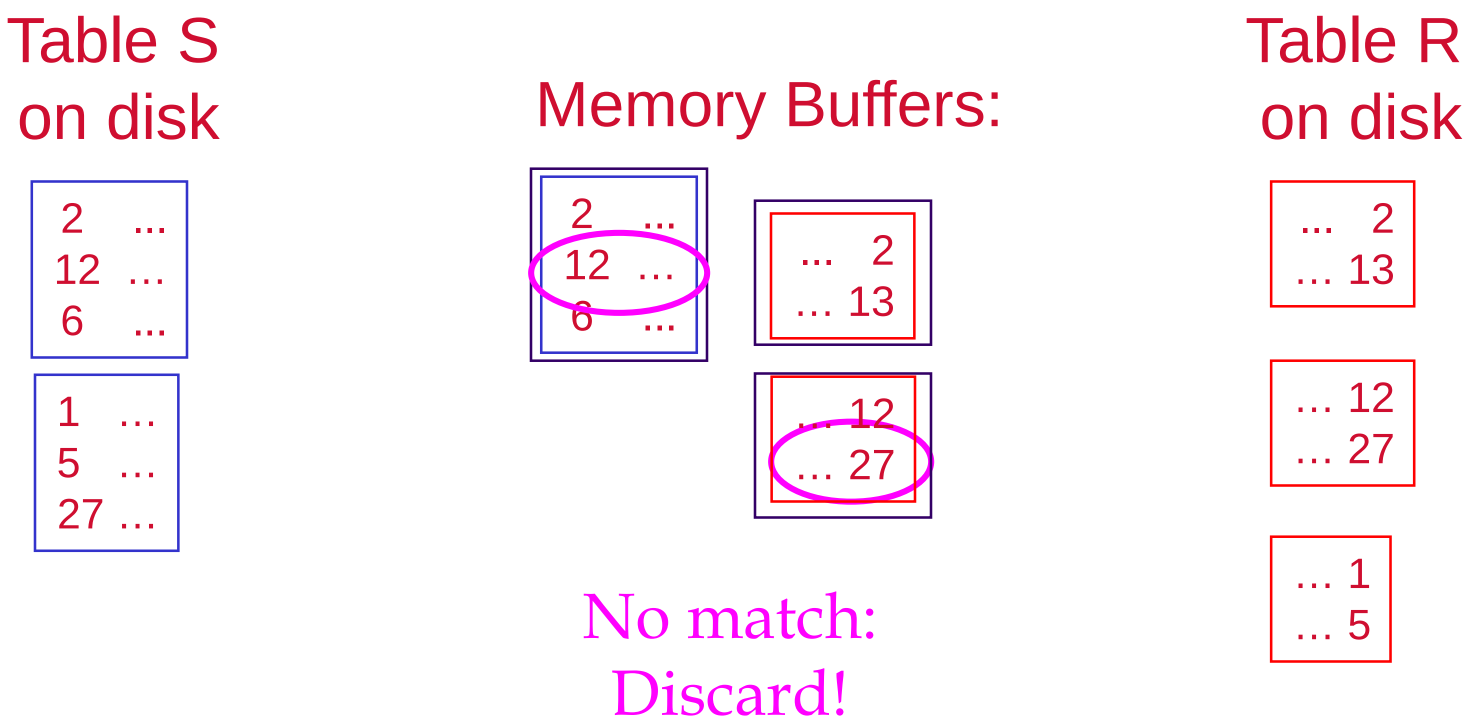


Query Answer

2 2

12... ...12

Block Nested Loops Join

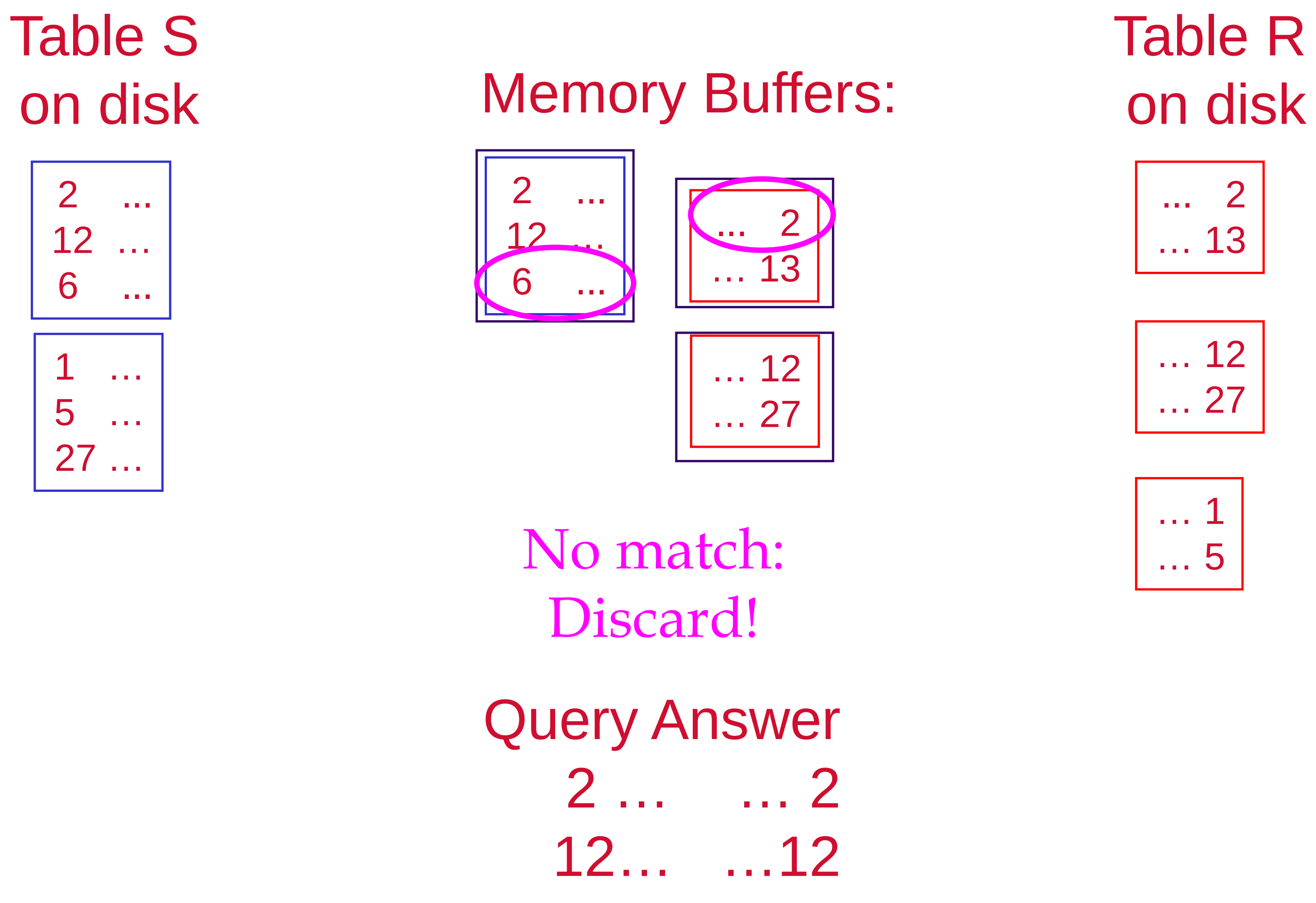


Query Answer

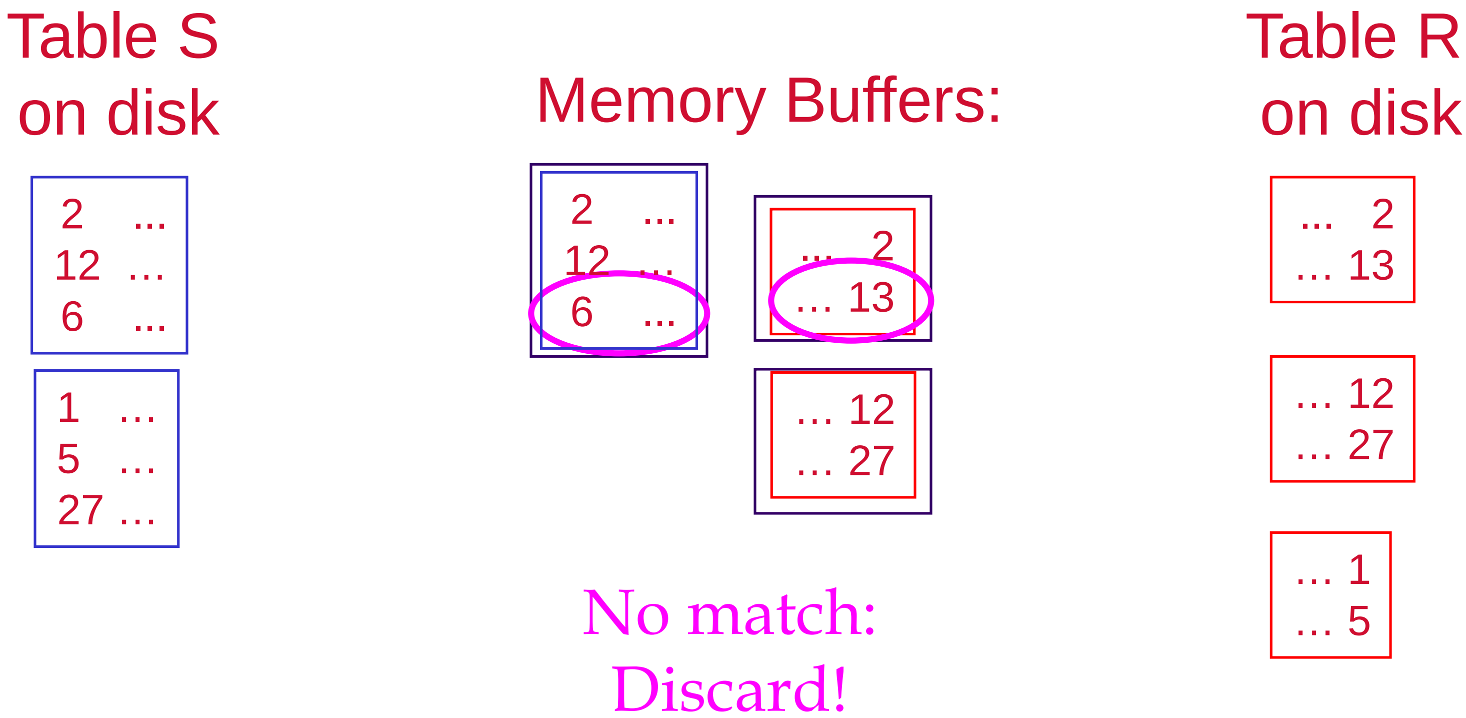
2 … … 2

12… …12

Block Nested Loops Join



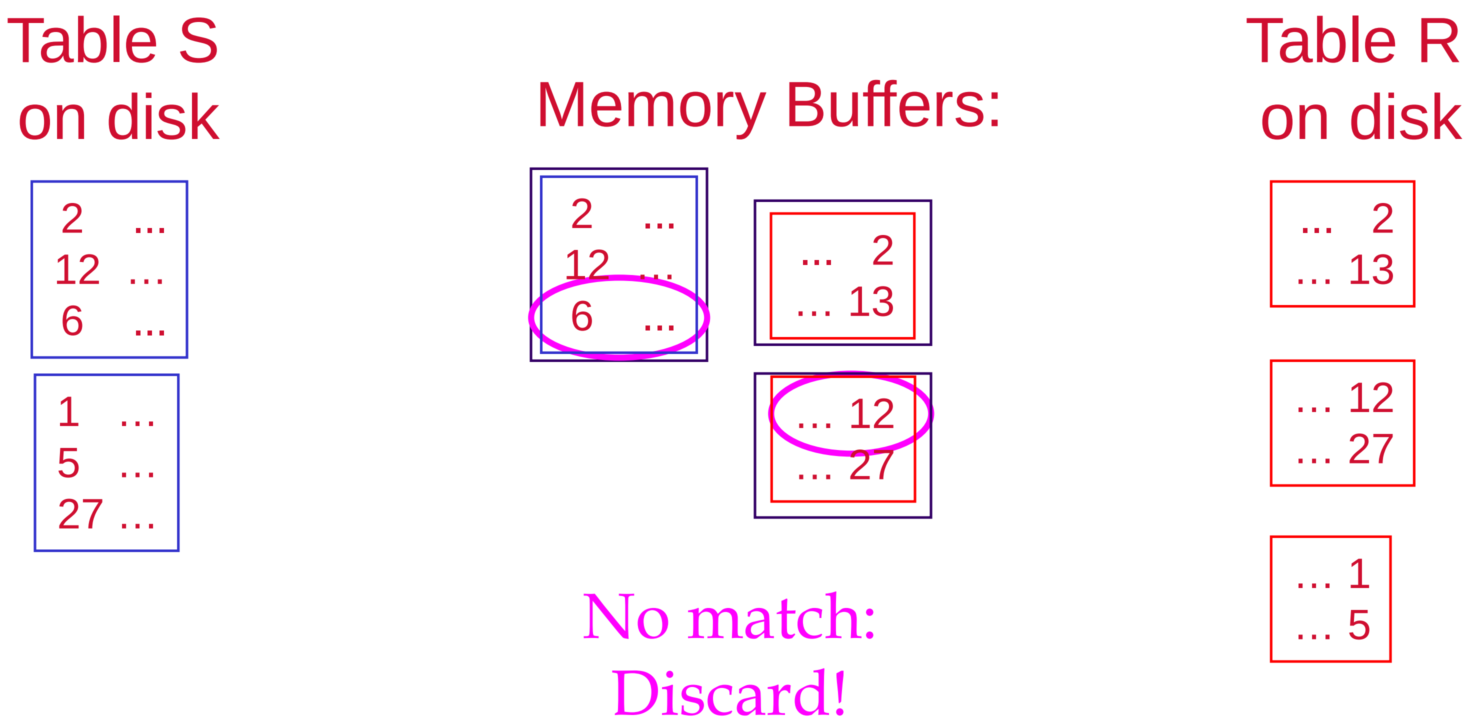
Block Nested Loops Join



Query Answer

2 ...	... 2
12...	...12

Block Nested Loops Join

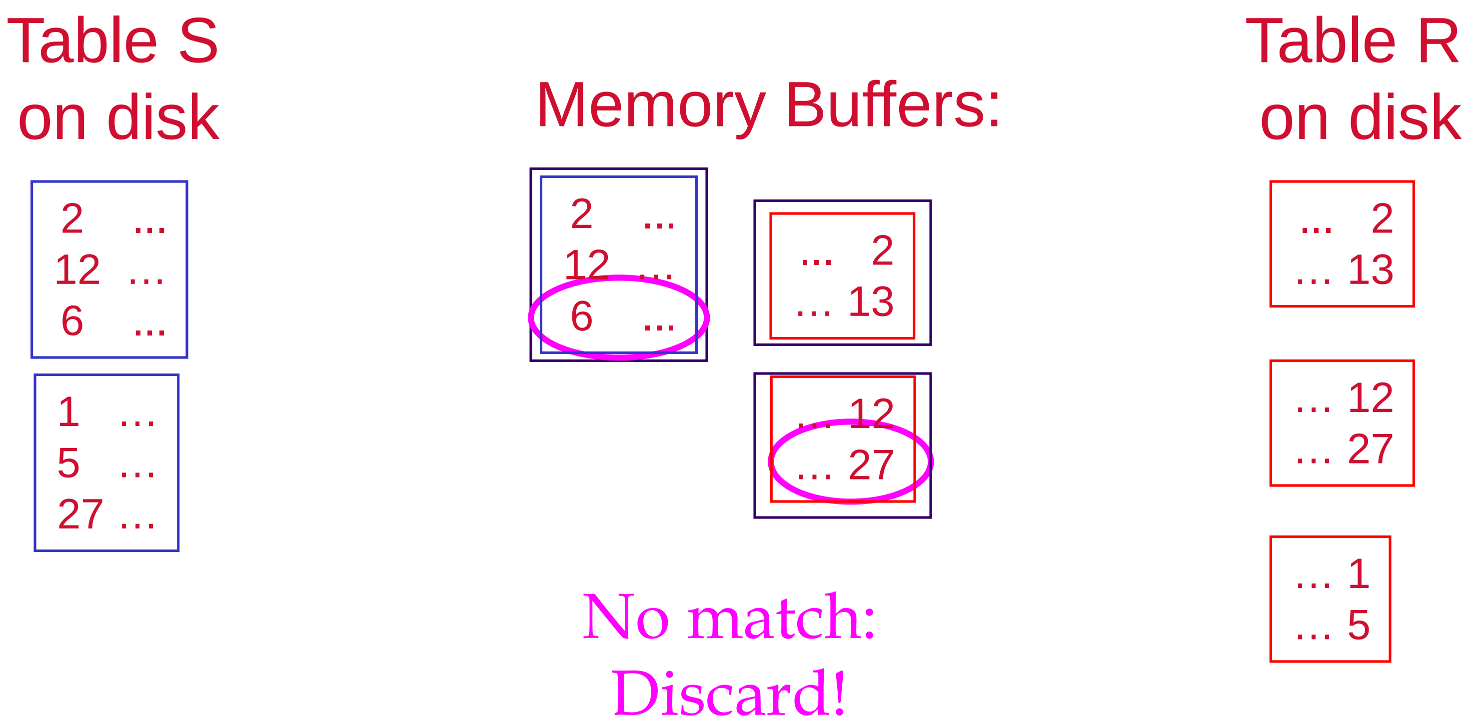


Query Answer

2 2

12... ...12

Block Nested Loops Join

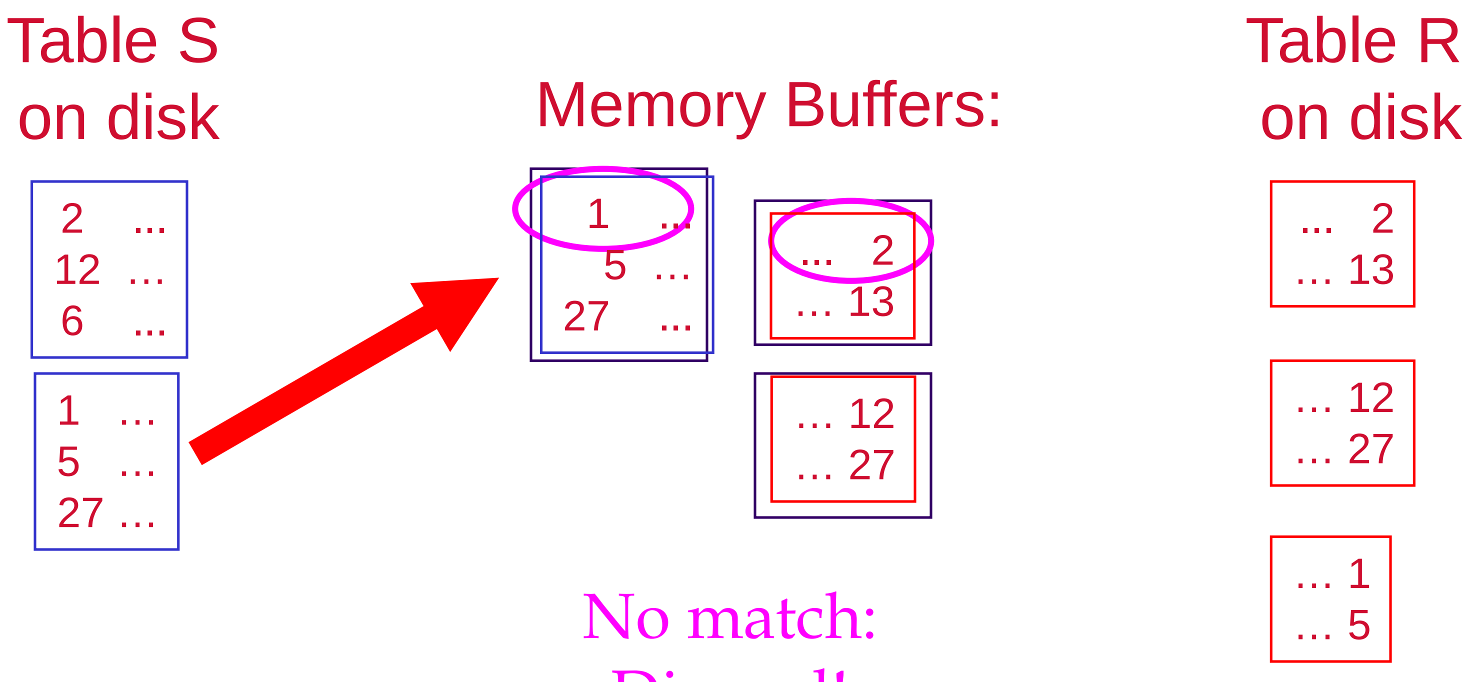


Query Answer

2 … … 2

12… …12

Block Nested Loops Join

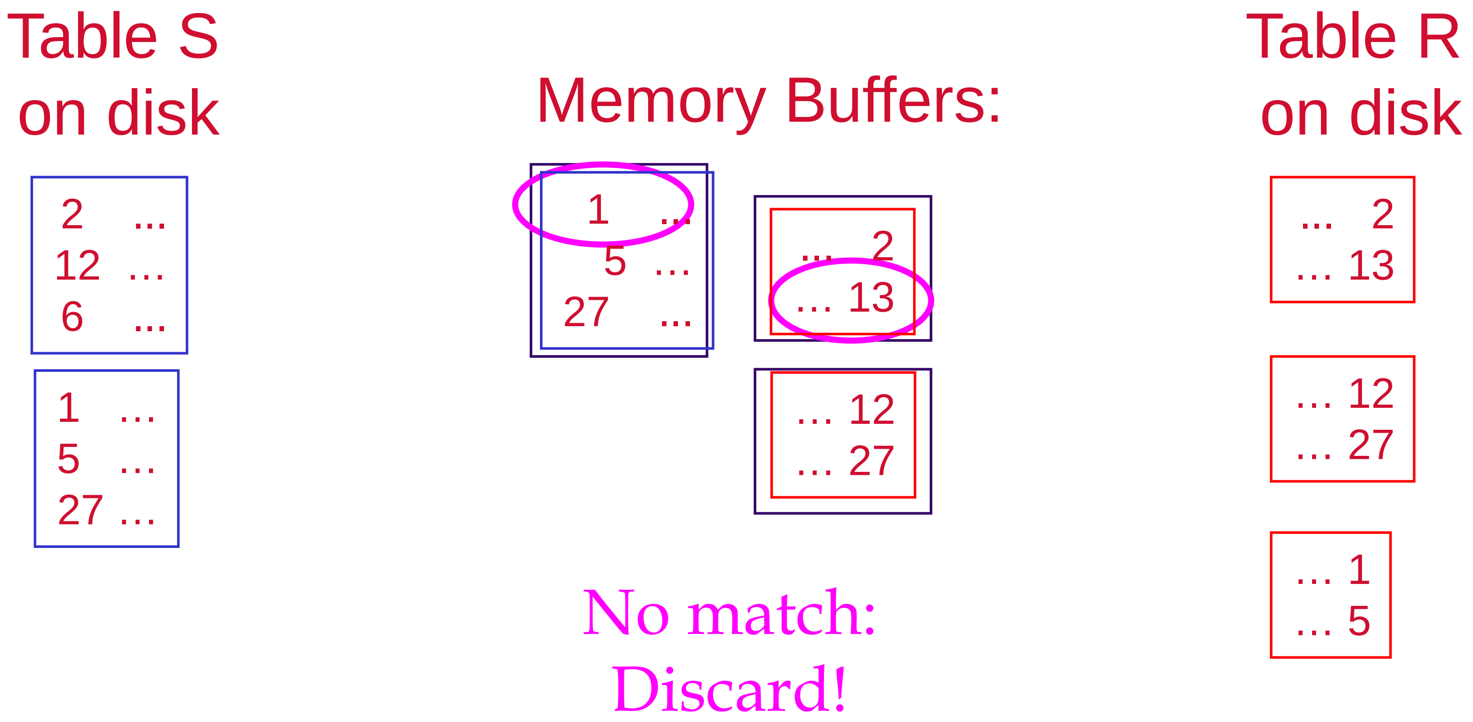


Query Answer

2 2

12... ...12

Block Nested Loops Join



Query Answer

2 … … 2

12… …12

Block Nested Loops Join

Table S on disk

2 ...
12 ...
6 ...

1 ...
5 ...
27 ...

Memory Buffers:

1 ...
5 ...
27 ...

... 2
... 13

... 12
... 27

Table R on disk

... 2
... 13

... 12
... 27

... 1
... 5

No match: Discard!

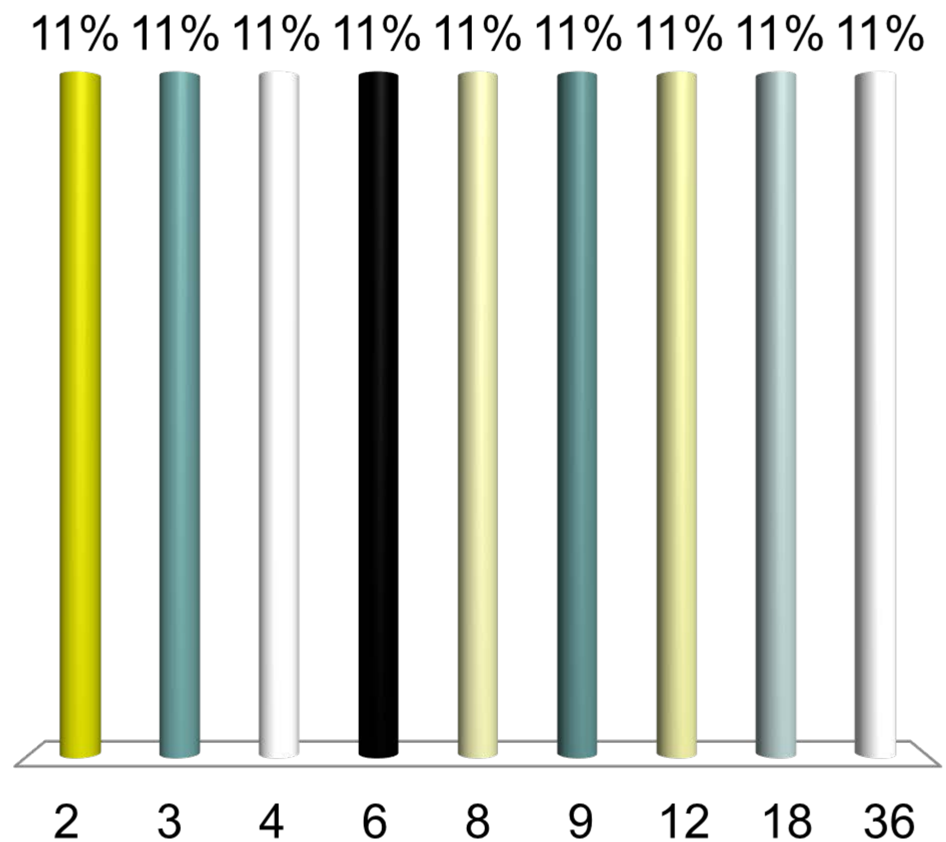
Query Answer

2 2
12... ...12

And so forth ...

How many blocks of R were accessed from disk in previous example by the end of the join?
M = 4, $b_r = 3$, $b_s = 2$, n_r, n_s *are both 6*

A. 2
B. 3
C. 4
D. 6
E. 8
F. 9
G. 12
H. 18
I. 36



How many blocks of S were accessed from disk in previous example by the end of the join?

M = 4, $b_r = 3$, $b_s = 2$, n_r, n_s *are both 6*

A. 2

B. 3

C. 4

D. 6

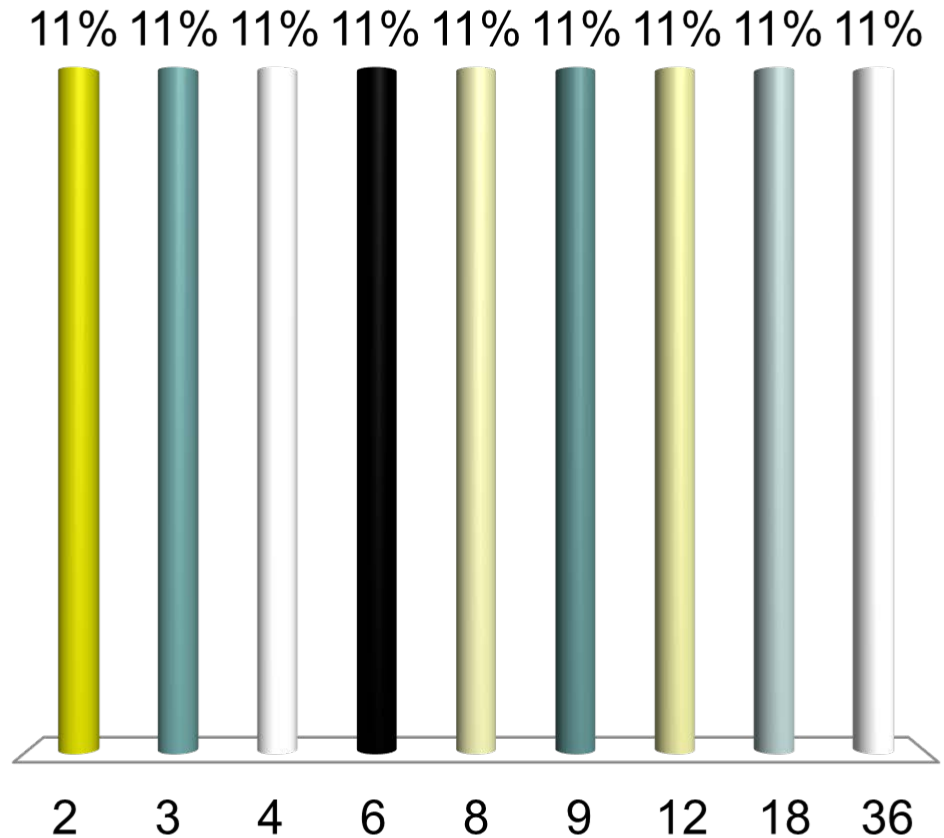
E. 8

F. 9

G. 12

H. 18

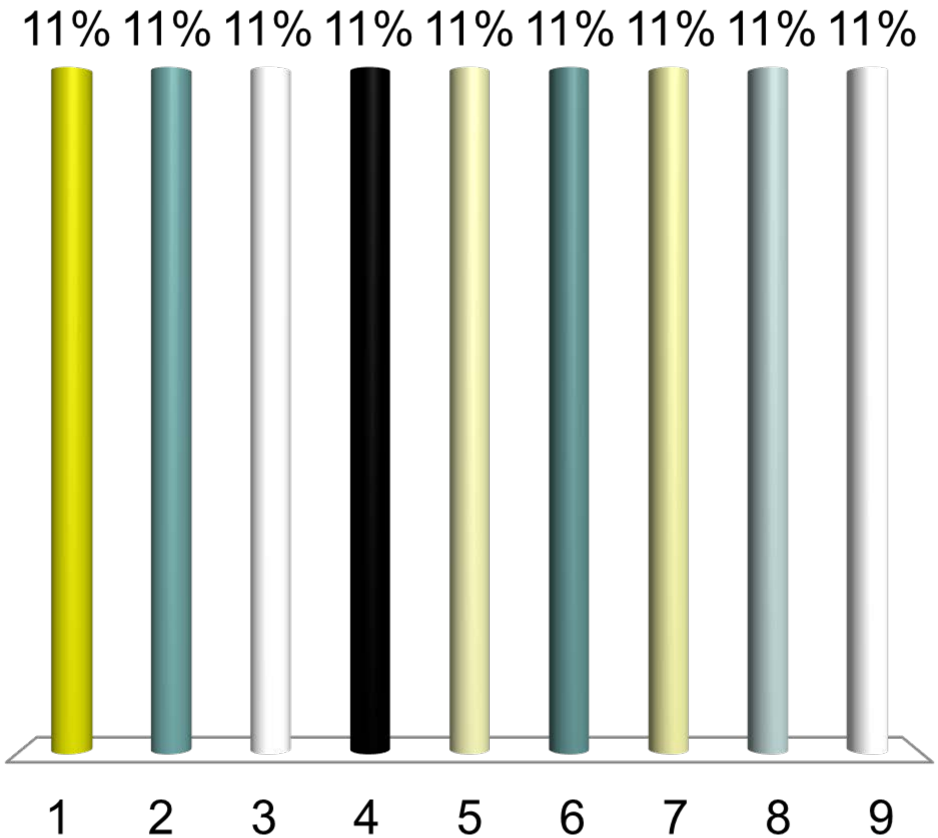
I. 36



How many total seeks (of R or S) in previous example by the end of the join?

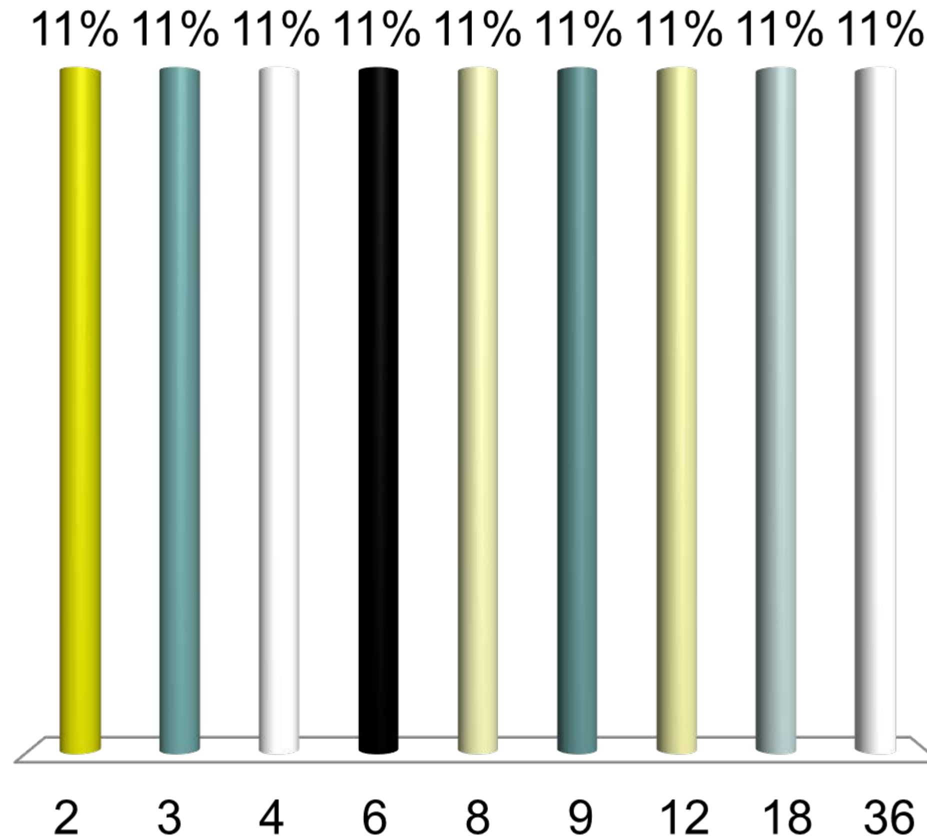
M = 4, $b_r = 3$, $b_s = 2$, n_r, n_s *are both 6*

A. 1
B. 2
C. 3
D. 4
E. 5
F. 6
G. 7
H. 8
I. 9



How many blocks of S would be accessed from disk in previous example if M was 3 instead of 4? M = 3, $b_r = 3$, $b_s = 2$, n_r, n_s *are both 6*

A. 2
B. 3
C. 4
D. 6
E. 8
F. 9
G. 12
H. 18
I. 36



Block Nested-loops Join

- Case 1: Minimum memory required = 3 blocks
  - Blocks transferred: $b_r * b_s + b_r$
  - Seeks: $2 * b_r$
- Case 2: S fits in memory
  - Blocks transferred: $b_s + b_r$
  - Seeks: *2*
- Case 3: Situation between case 1 and case 2 (e.g, the example on the previous slides)
  - Continued on next slide …

Block Nested-loops Join

- Case 3: (e.g., S > 50 blocks, M is 50 blocks)

  for each group of (M – 2) blocks in R

  for each block B_s in S

  for each tuple s in B_s

  for each tuple r in the group of (M – 2) blocks

  check if r.a = s.a (or whether |r.a – s.a| < 0.5)

- Why is this good ?
  - We only have to read *S* a total of $b_r / (M - 2)$ times (instead of b_r times)
  - Blocks transferred: $b_s * b_r / (M - 2) + b_r$
  - Seeks: $2 * b_r / (M - 2)$

What should be R and S?

- Nested loops join
  - Particular values of b_s, b_r, n_s, *and* n_r *matter*
  - *Often want the smaller relation to be inner (S)*
- *Block nested loops join*
  - *Want the smaller relation to be outer (R)*

Index Nested-loops Join

- *select * from R, S where R.a = S.a*
  - "equi-join"
- Nested-loops

  for each tuple r in R

  for each tuple s in S

  check if r.a = s.a (or whether |r.a – s.a| < 0.5)
- Suppose there is an index on *S.a*
- *Why not use the index instead of the inner loop ?*

  for each tuple r in R

  use the index to find S tuples with S.a = r.a

Index Nested-loops Join

- *select * from R, S where R.a = S.a*
  - Called an "*equi-join*"
- *Why not use the index instead of the inner loop ?*

  for each tuple r in R

  use the index to find S tuples with S.a = r.a

- Cost of the join:
  - $b_r\,(t_T + t_S) + n_r * c$
  - *c == the cost of index access*
    - *Computed using the formulas discussed earlier*

Index Nested-loops Join

- W/ indexes for both *R*, *S*, use one w/ fewer tuples as outer.
- Recall example:
  - Number of records -- *R*: $n_r = 10{,}000$, *S*: $n_s = 5000$
  - Number of blocks -- *R*: $b_r = 400$, *S*: $b_s = 100$

$\boxed{b_r + n_r * c}$

- Assume B⁺-tree for *R*, avg fanout of 20, implies height *R* is 4
  - Cost is 100 + 5000 * (4 + 1) = 25,100, each w/ seek and transfer
- Assume B⁺-tree is on S: height = 3
  - Cost is 400 + 10000 * (3+1) = 40,400, each w/ seek and transfer

Index Nested-loops Join

- Restricted applicability
  - An appropriate index must exist
  - What about $|R.a - S.a| < 5$?
- Great for queries with joins and selections

  *SELECT **

  FROM accounts, customers

  WHERE accounts.customer-SSN = customers.customer-SSN AND

  accounts.acct-number = "A-101"

  - Use *accounts* as outer, use select to prune reads of customers