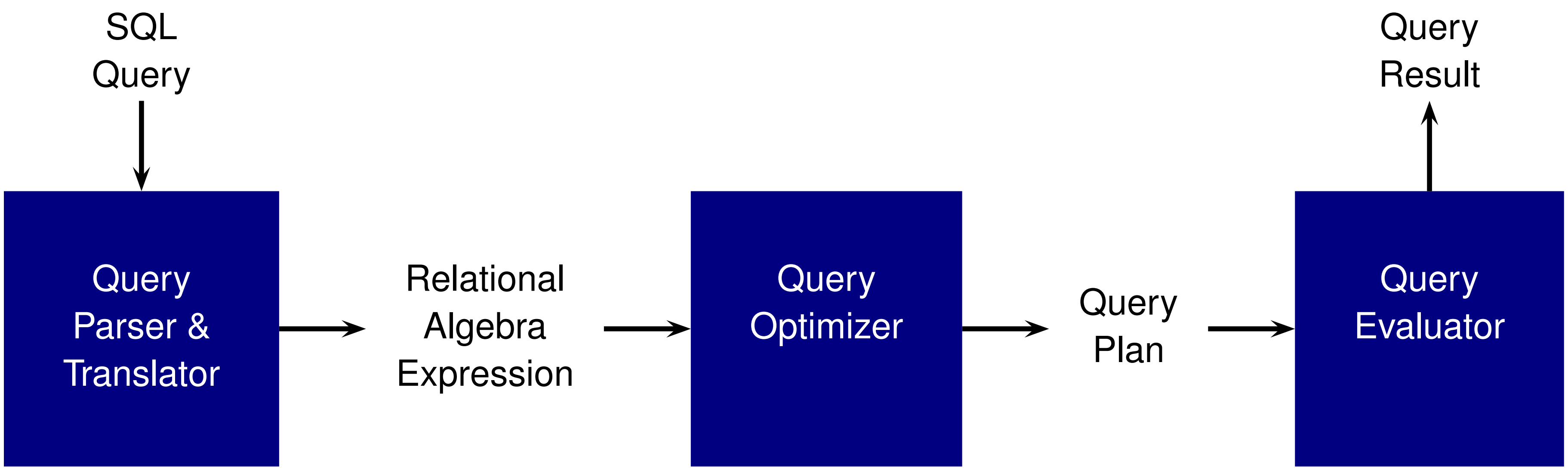
CS3223 Lecture 4
Query Evaluation: Sorting & Selection

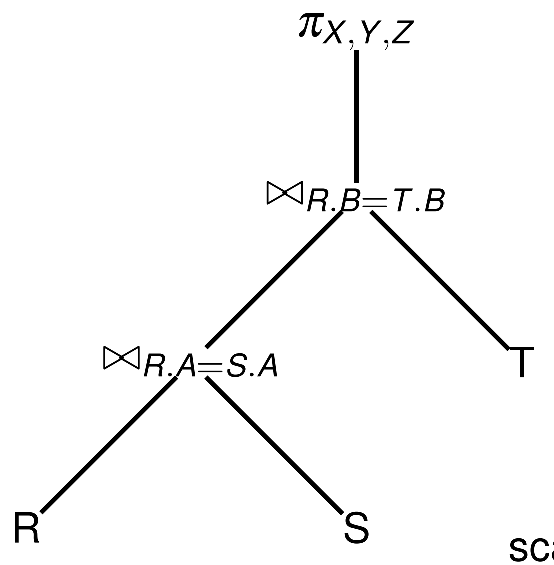
Query Processing



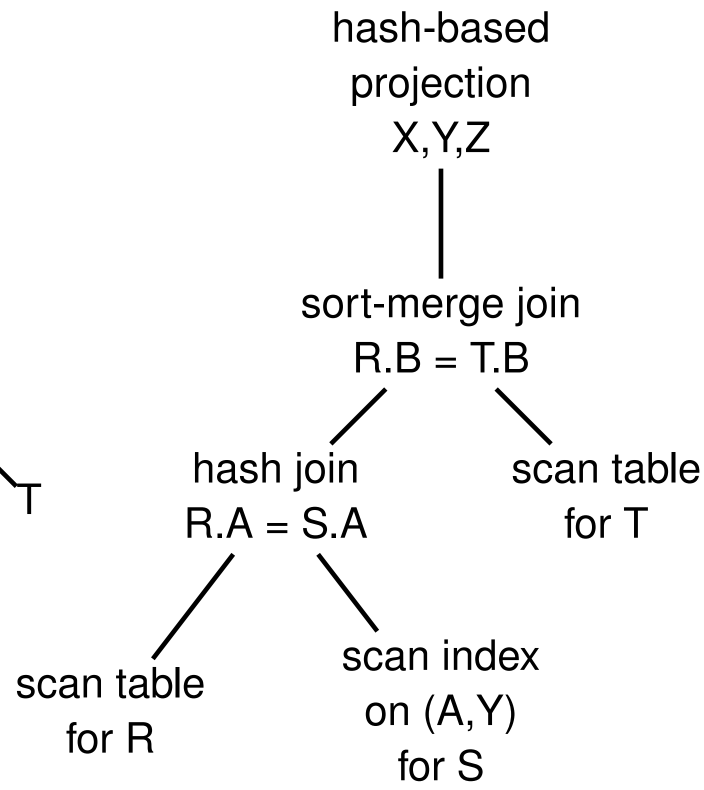
Query Plans

```
select R.X, S.Y, T.Z
from   R, S, T
where  R.A = S.A
and    R.B = T.B
```

SQL query *Q*



Logical plan for *Q*



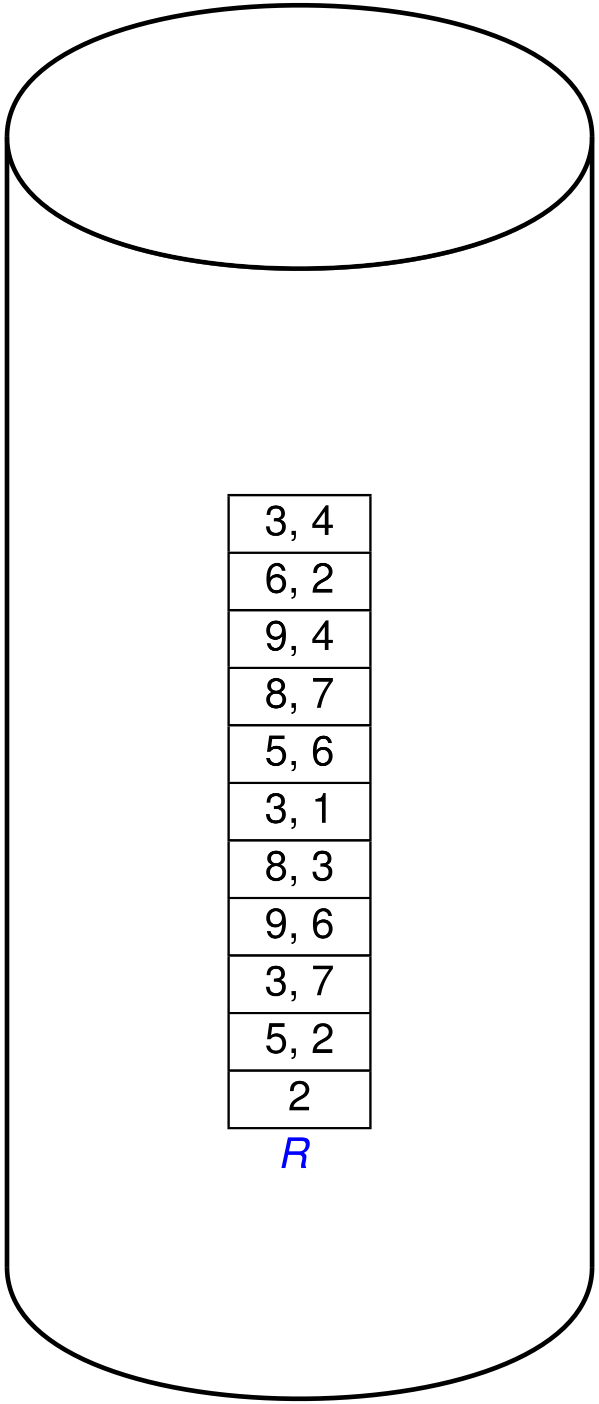
Physical plan for *Q*

Sorting in Database Systems

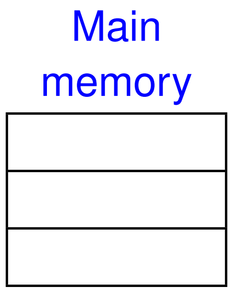
- Producing a sorted table of results

  `select` *
  `from` student
  `order by` age

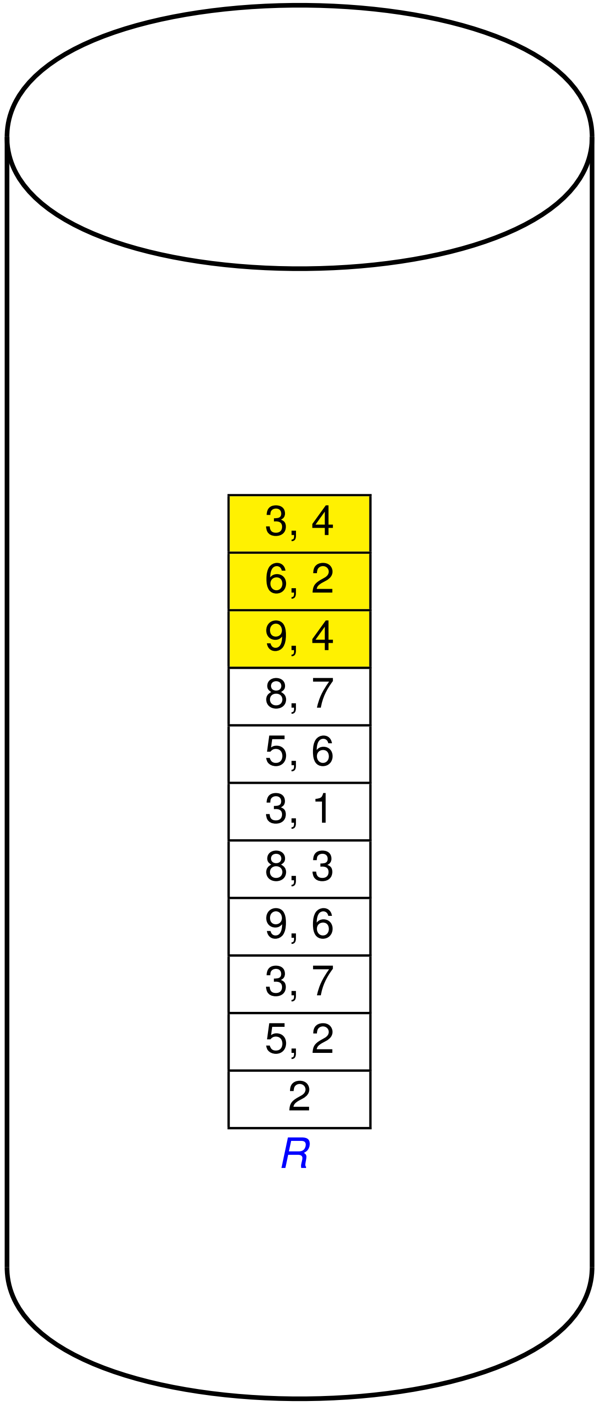
- Bulk loading a B^+-tree index
- Implementation of other relational algebra operators
  - Examples: projection, join

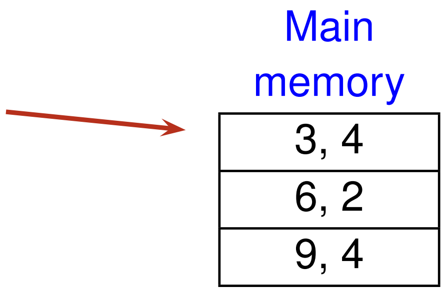


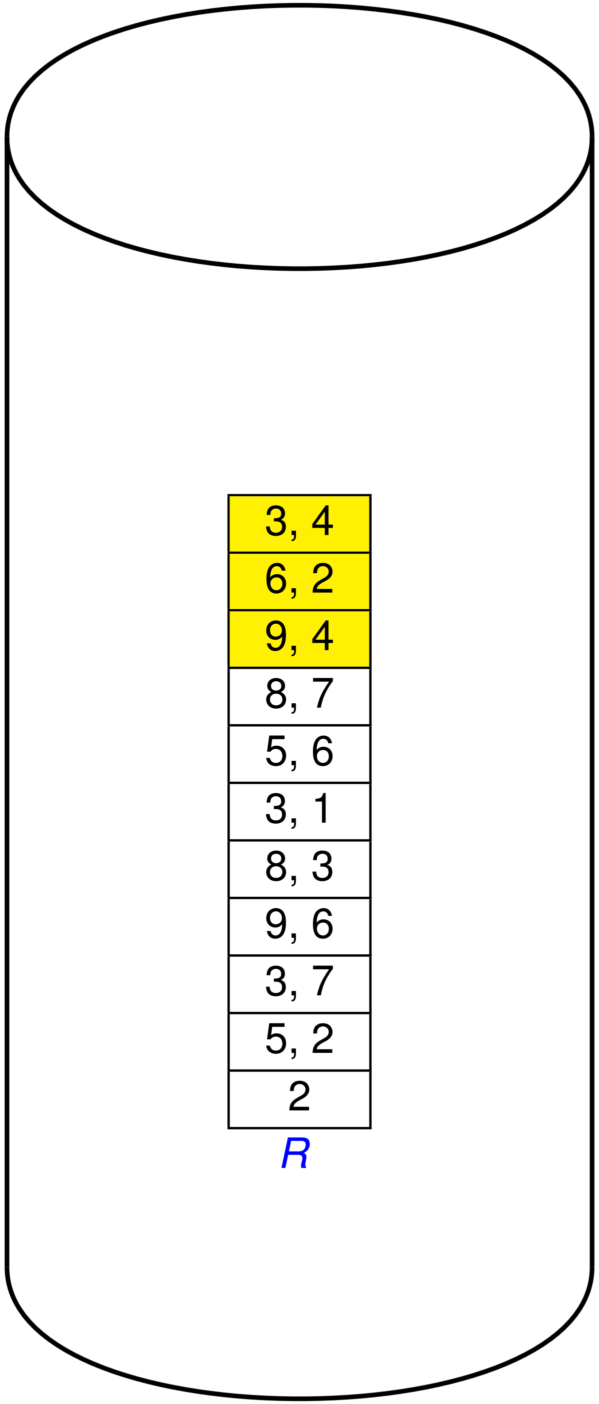
How to sort 11-page data R using 3 memory pages?



Read the first 3 data pages into allocated memory





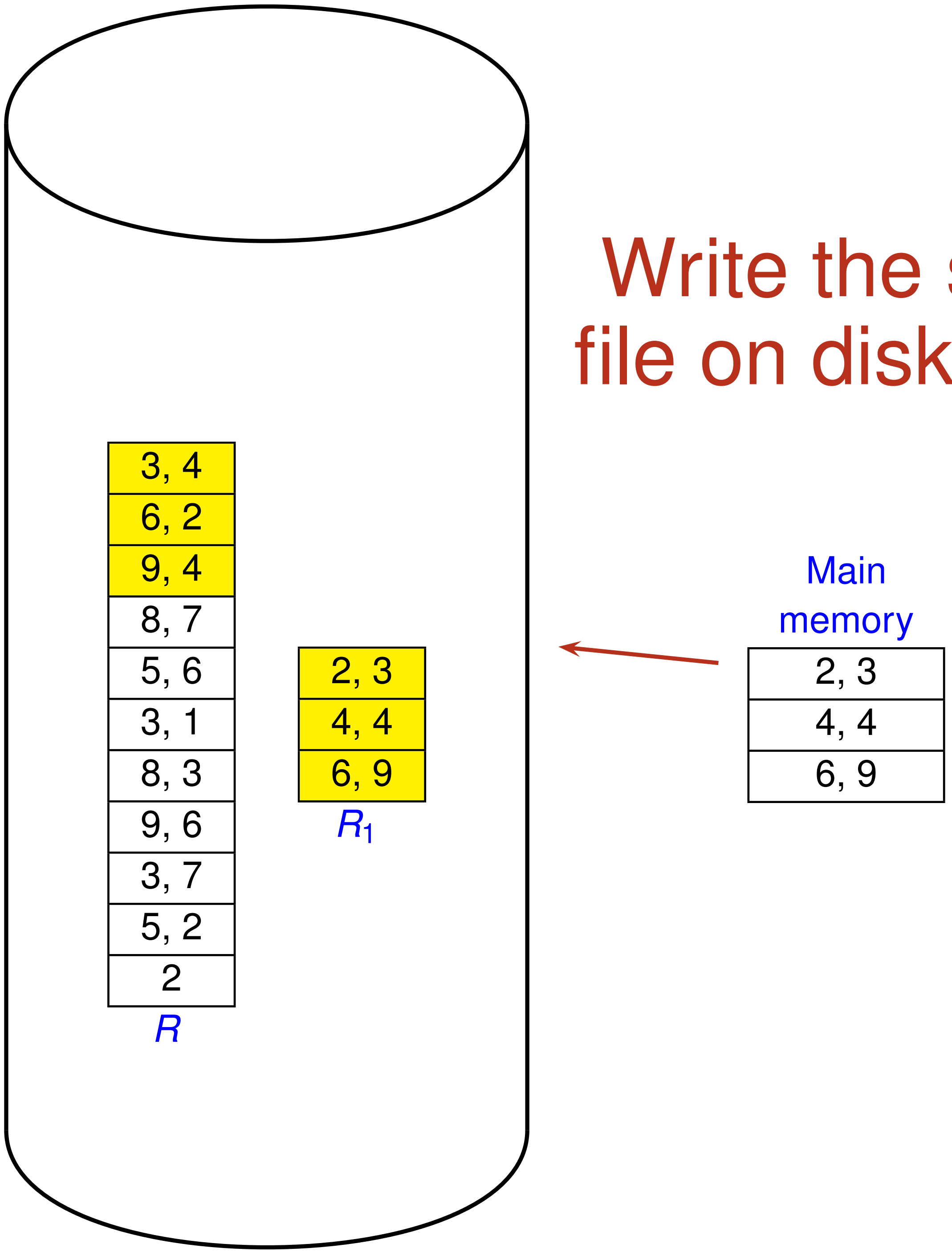


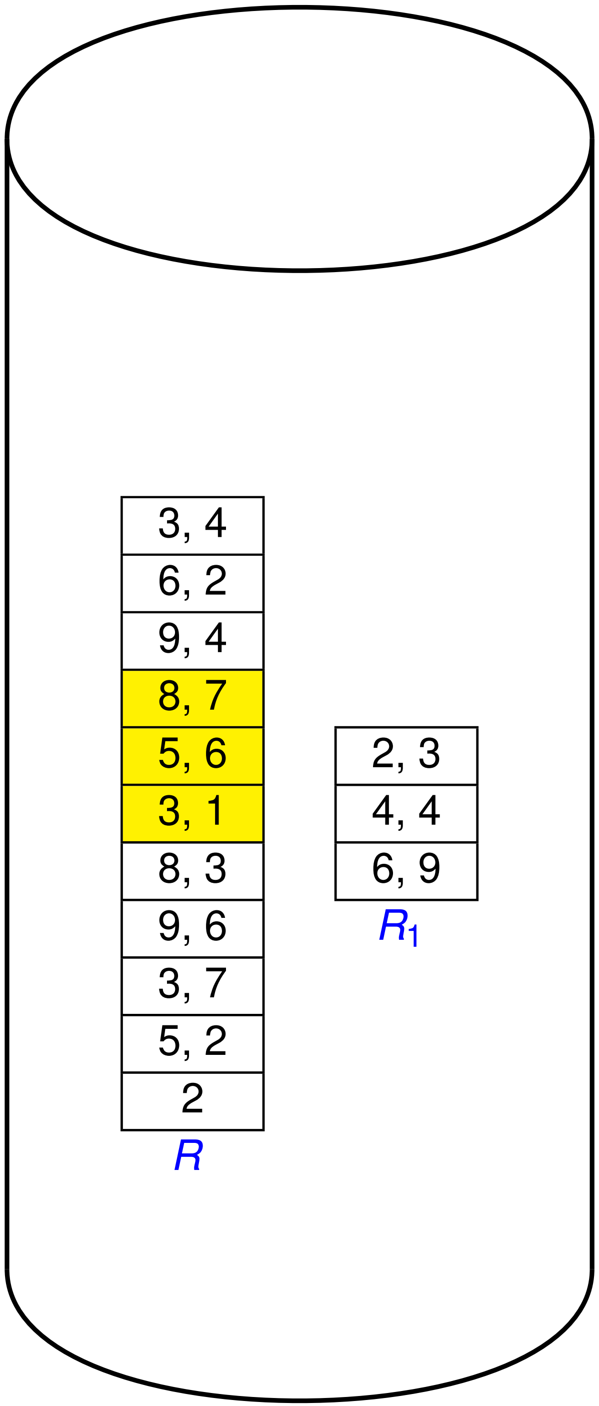
Sort the data records in main memory

Main memory

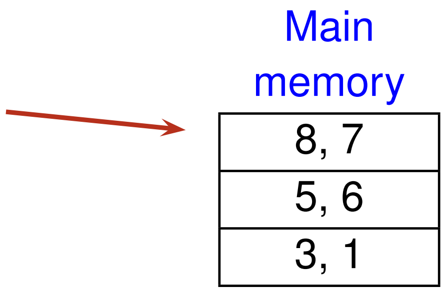
2, 3
4, 4
6, 9

Write the sorted data records to a file on disk (known as a sorted run)

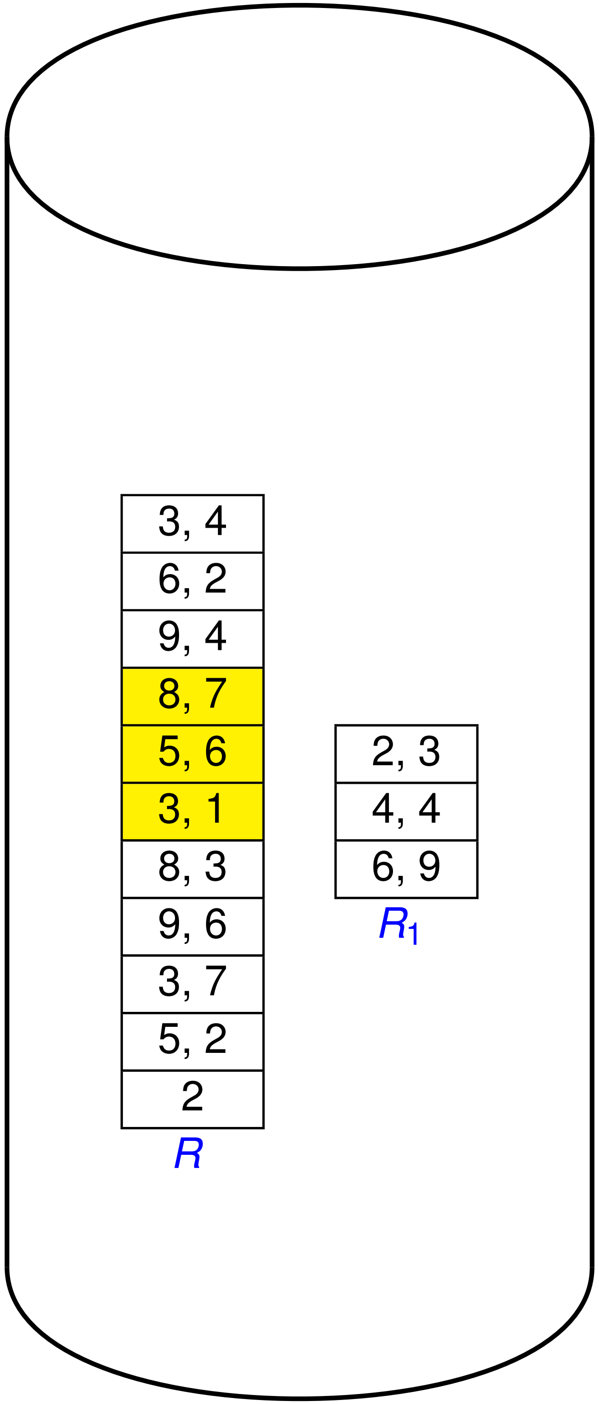


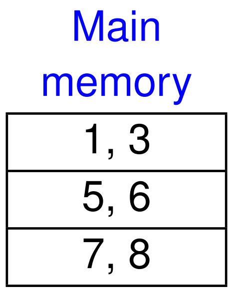


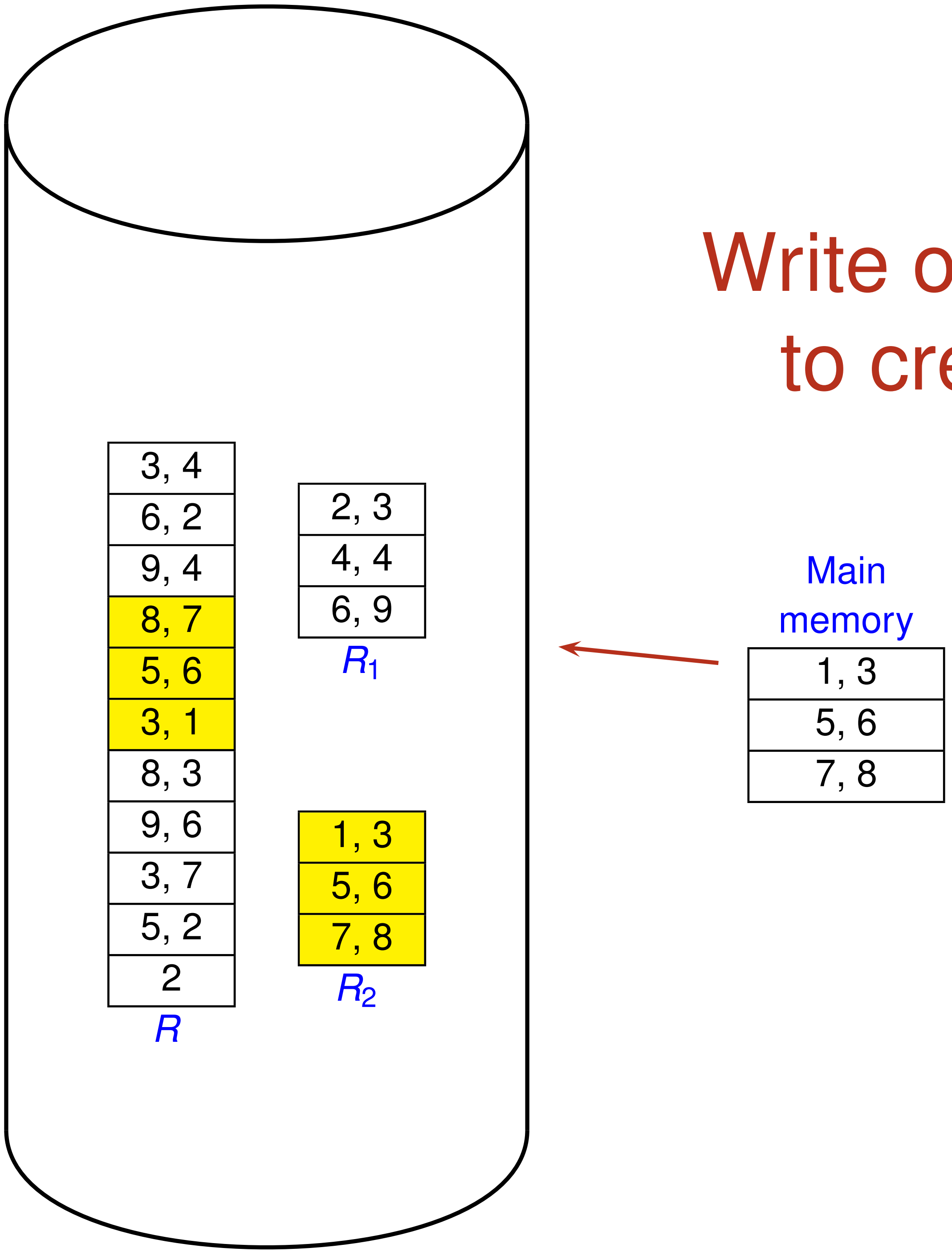
Read the next 3 data pages into allocated memory



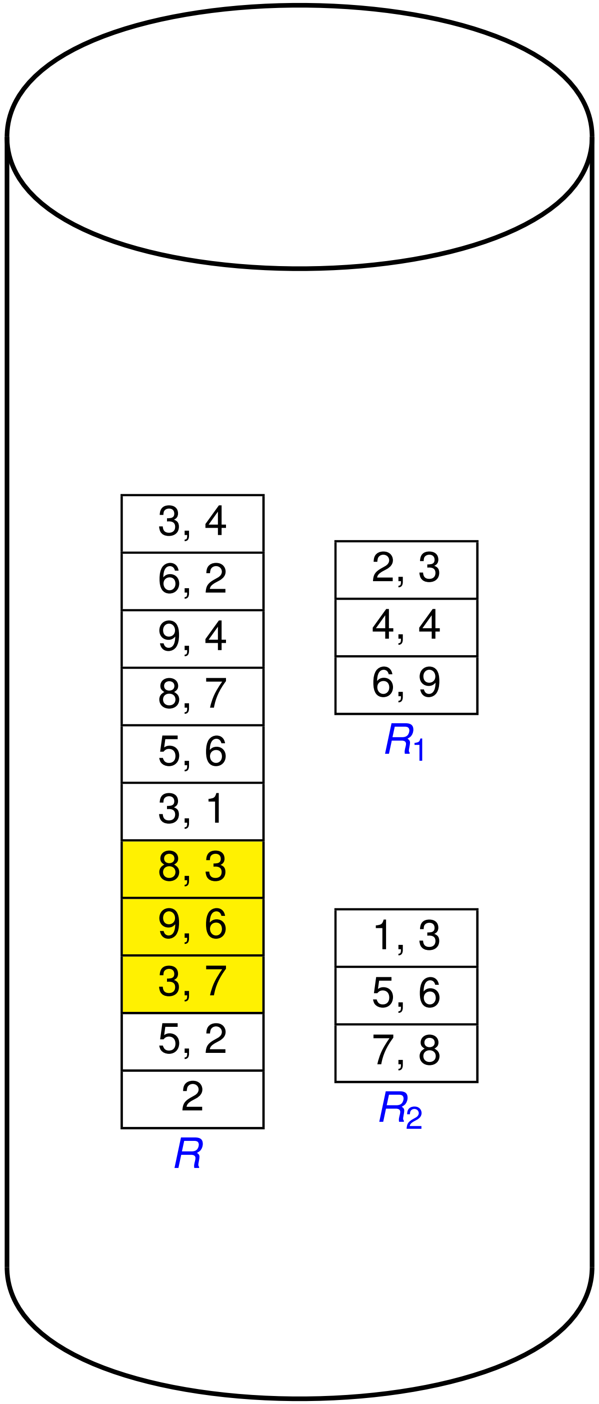
Sort the data records in main memory



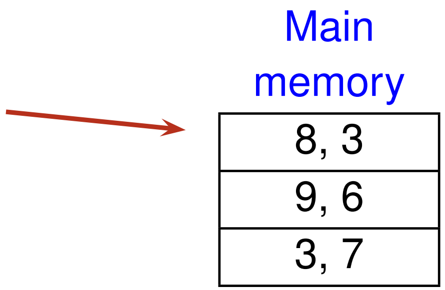




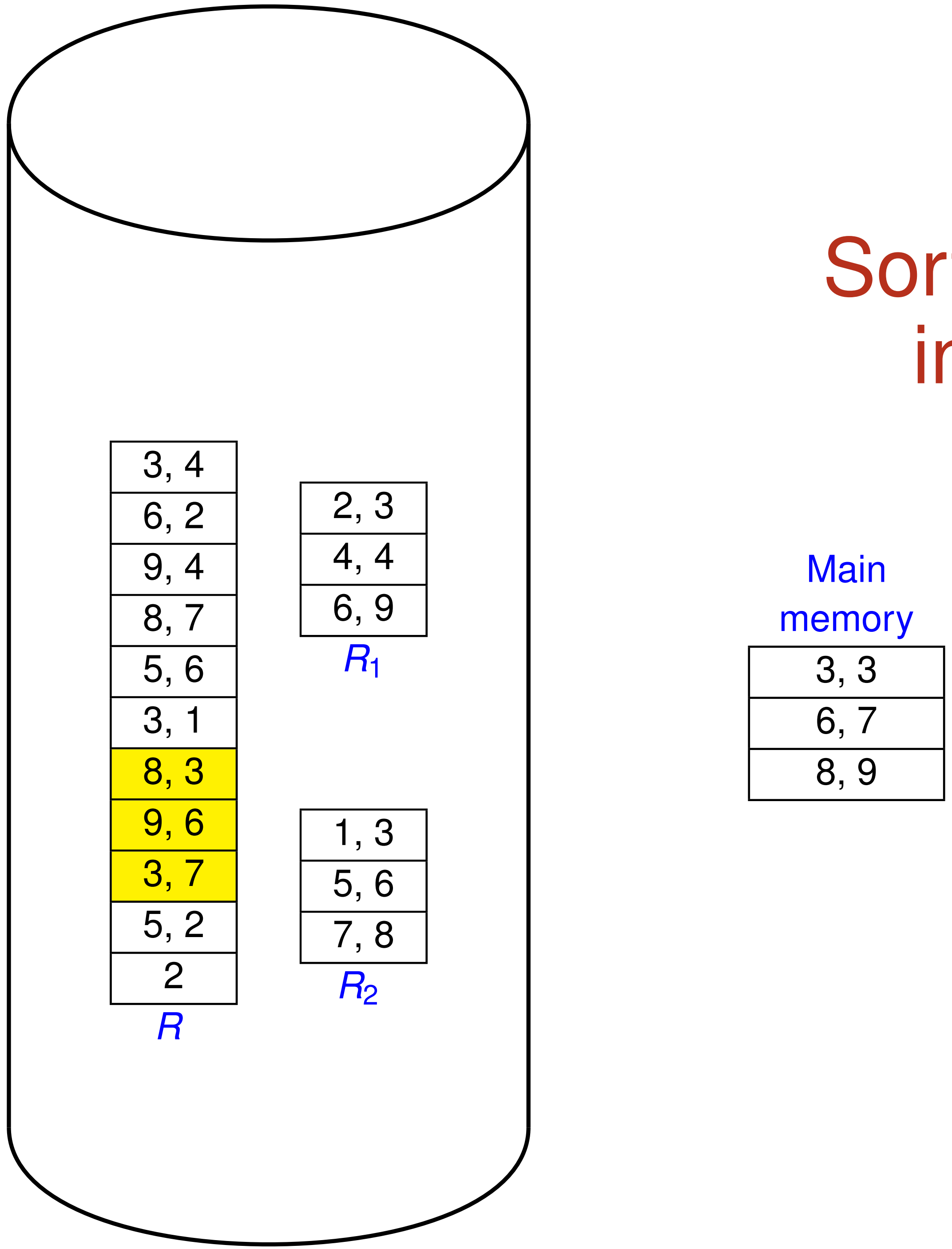
Write out the sorted records to create 2^{nd} sorted run

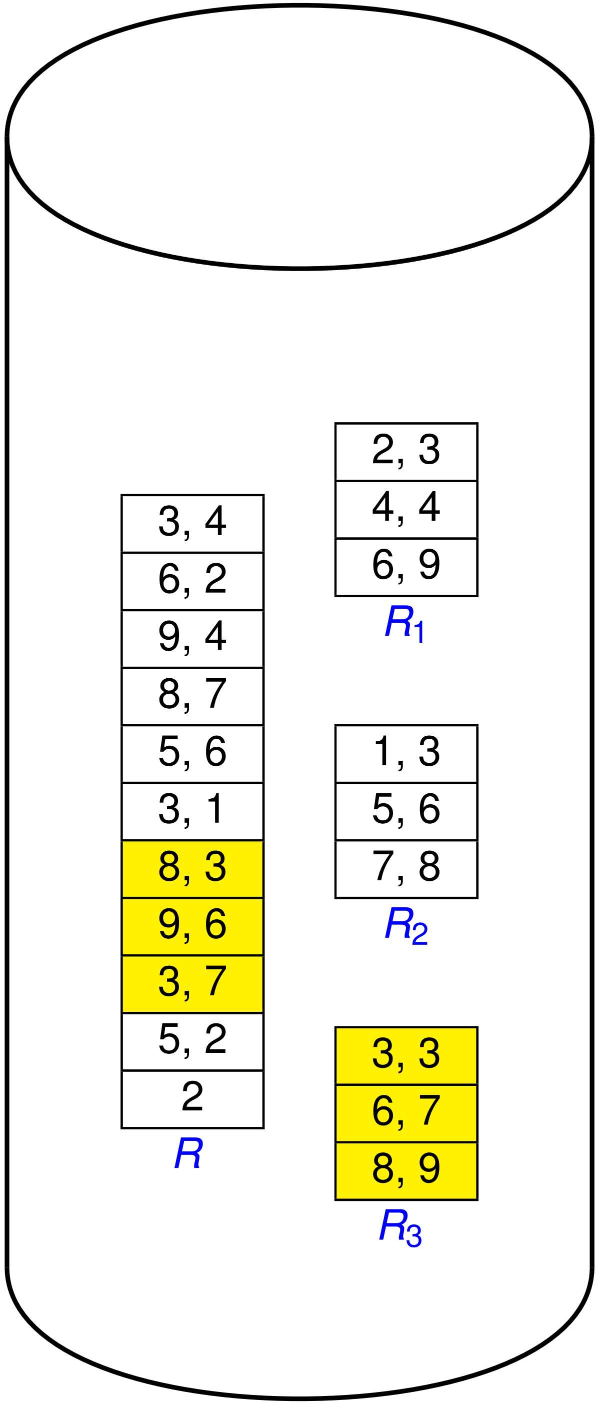


Read the next 3 data pages into allocated memory

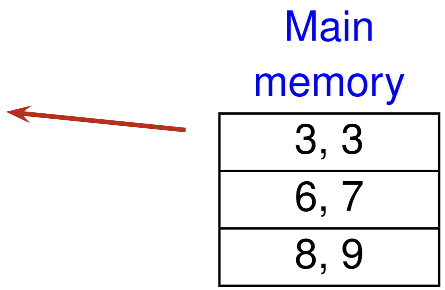


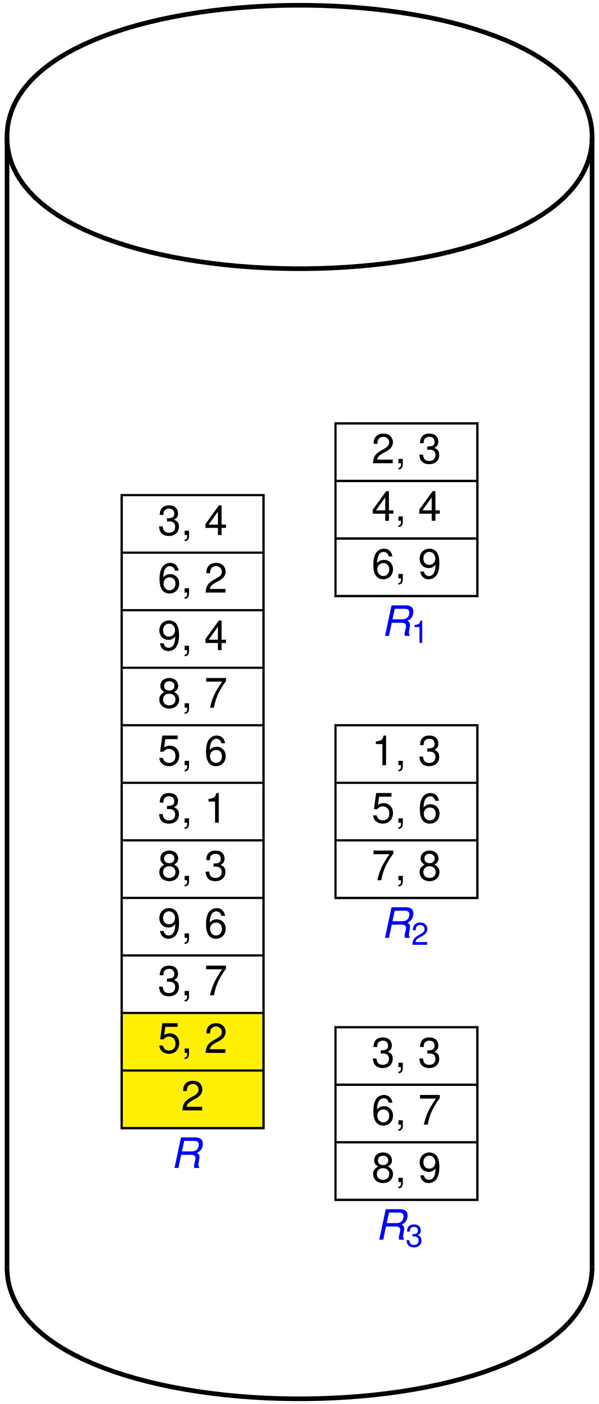
Sort the data records in main memory



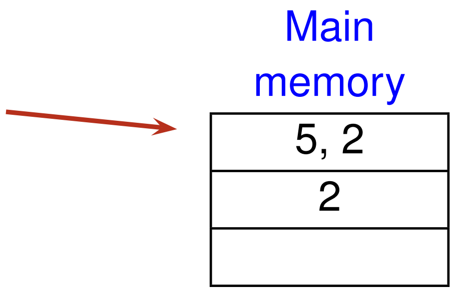


Write out the sorted records to create 3^{rd} sorted run

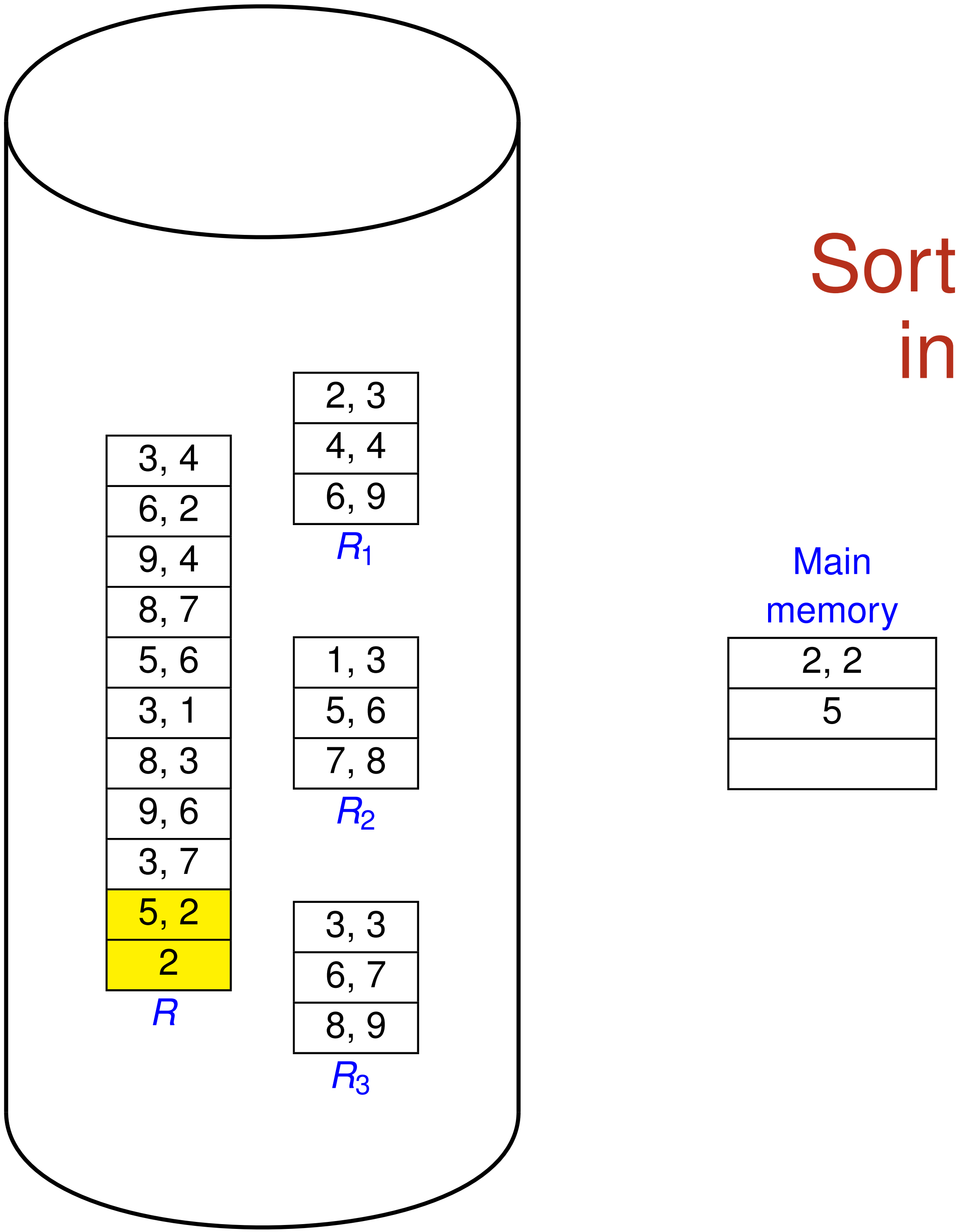


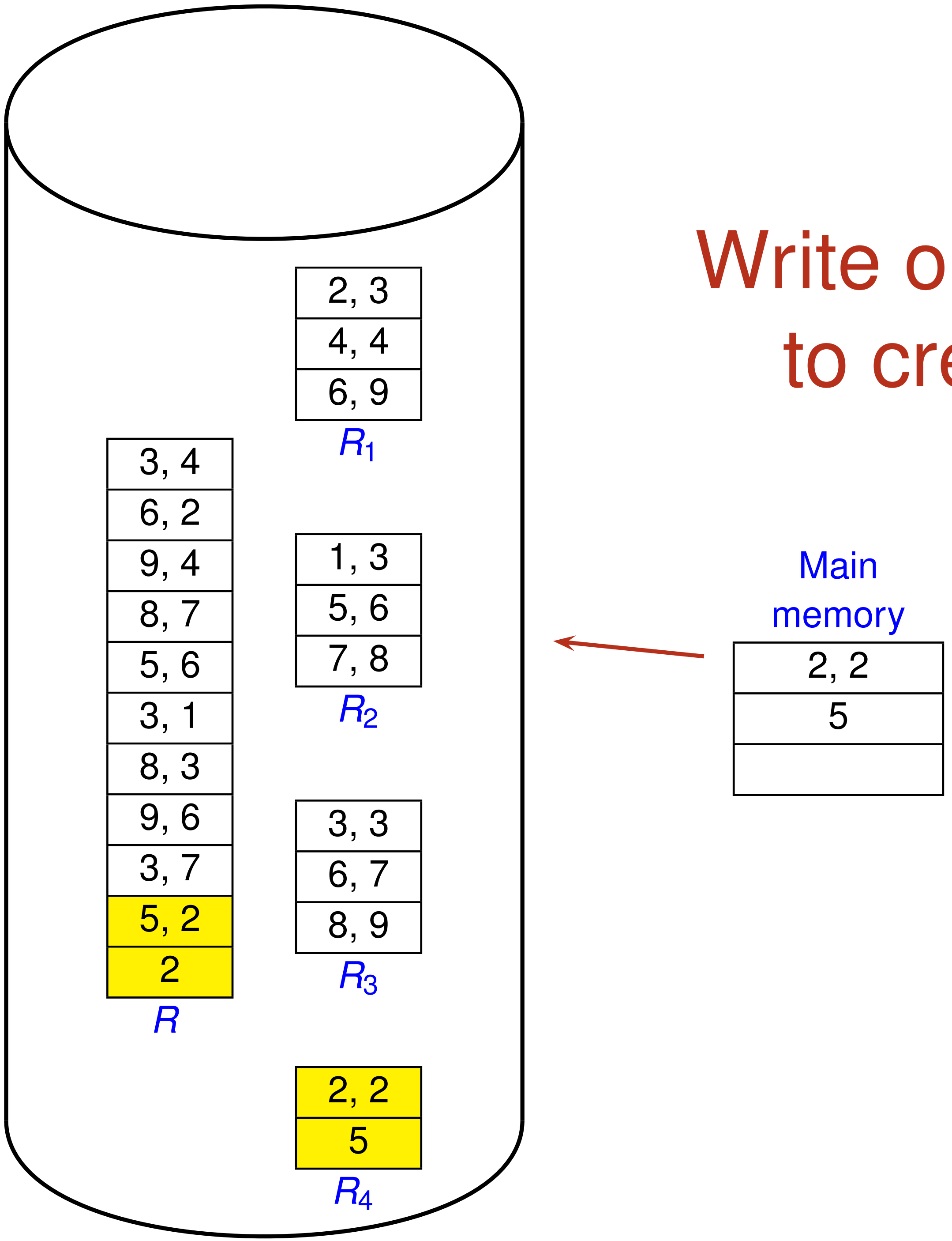


Read the remaining data pages into allocated memory

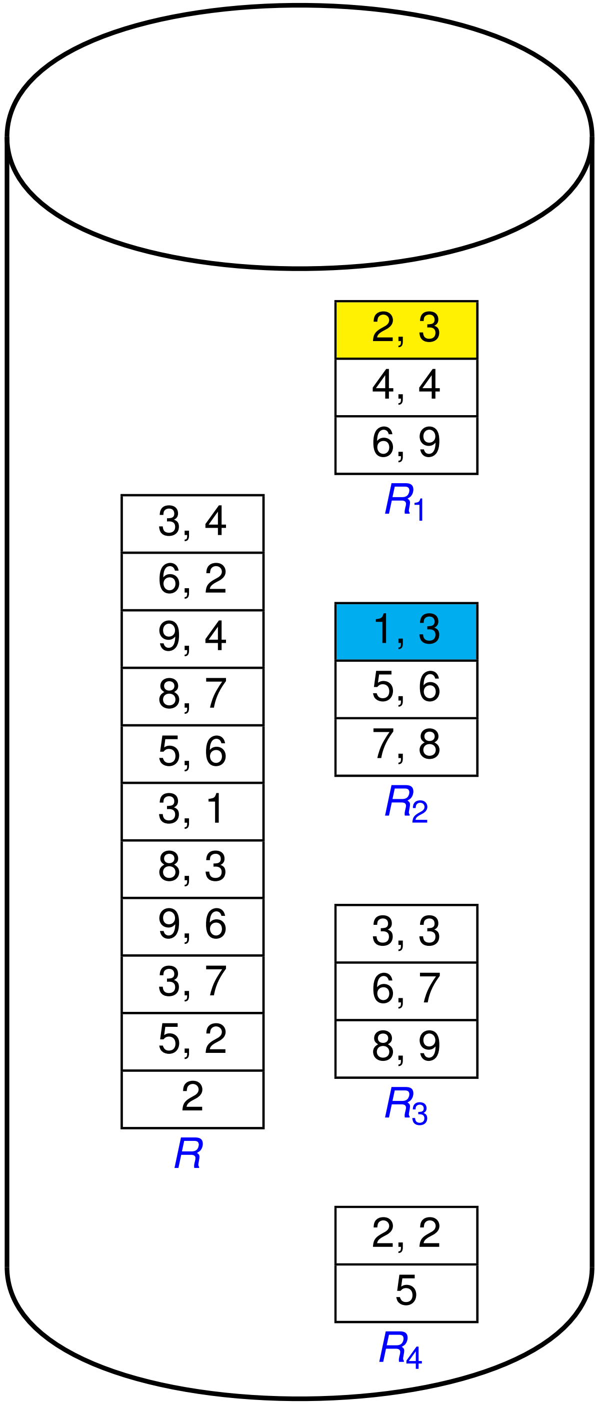


Sort the data records in main memory





Write out the sorted records to create 4^{th} sorted run

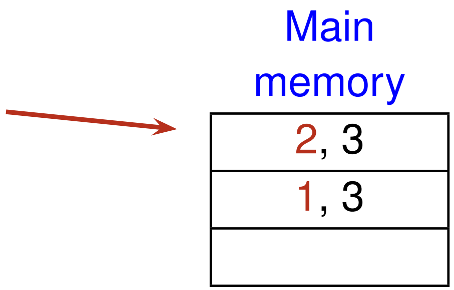


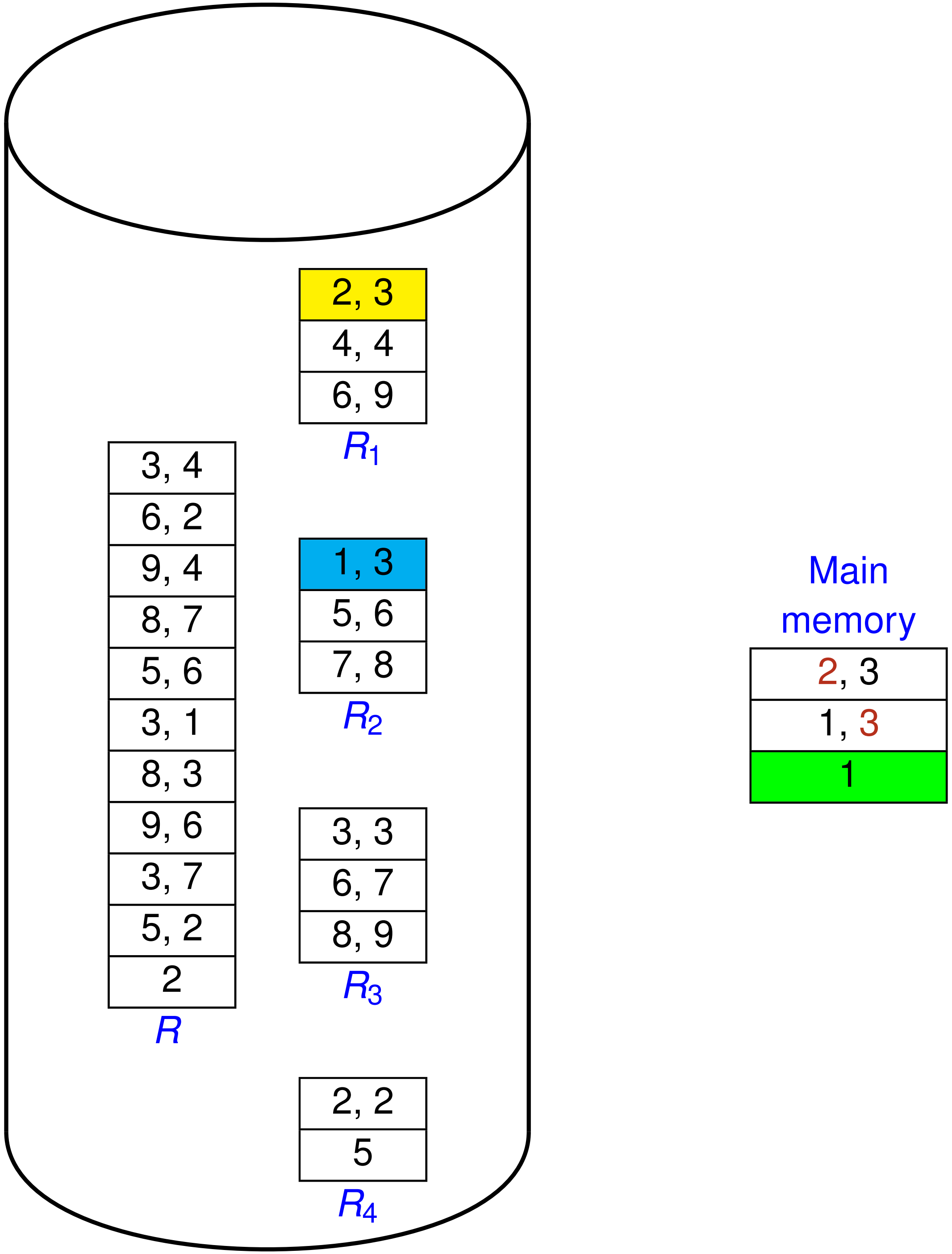
Merge R_1 & R_2 to create a longer sorted run

Merge one page of R_1 & R_2 at a time

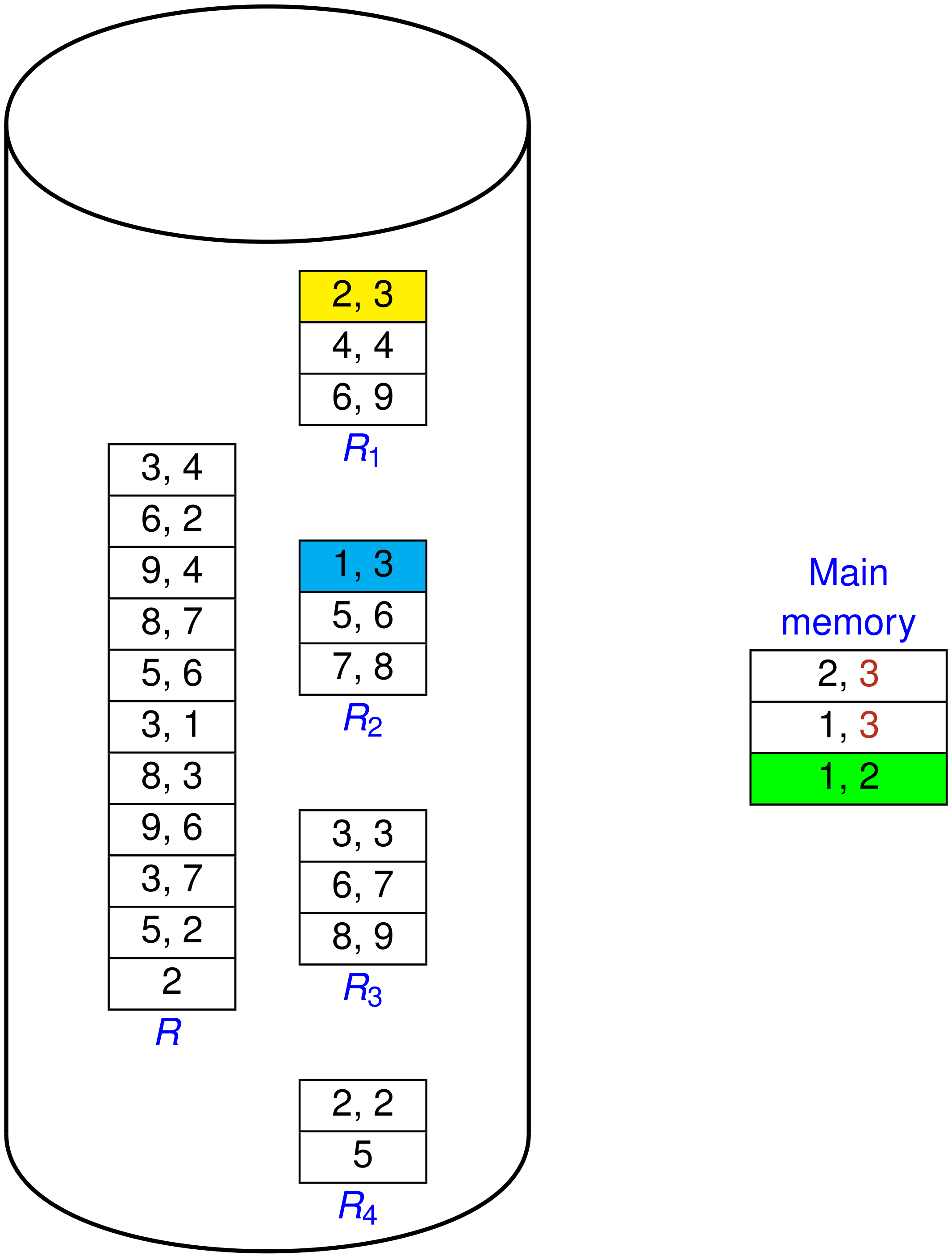
Use two memory pages for input records

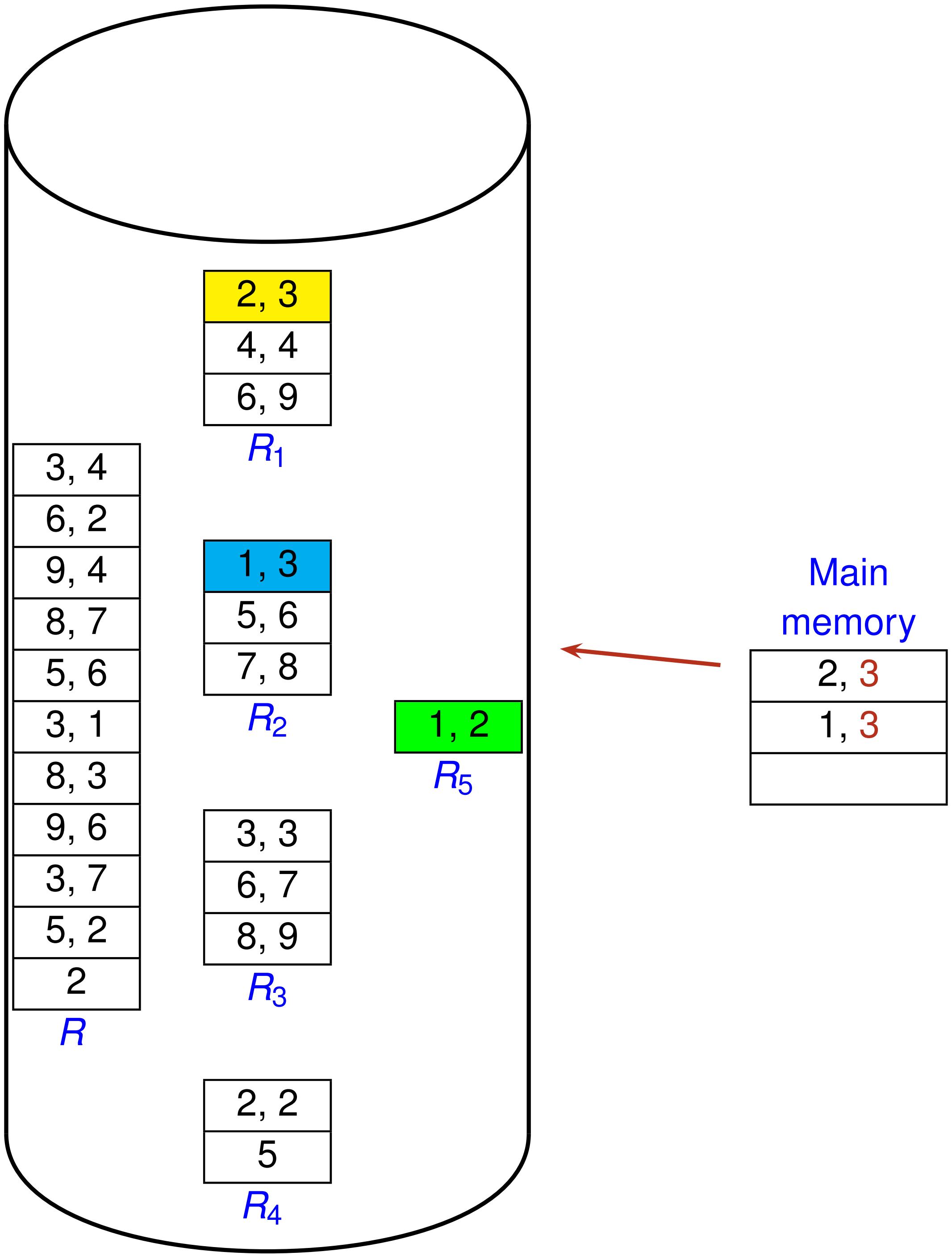
Use one memory page for output records



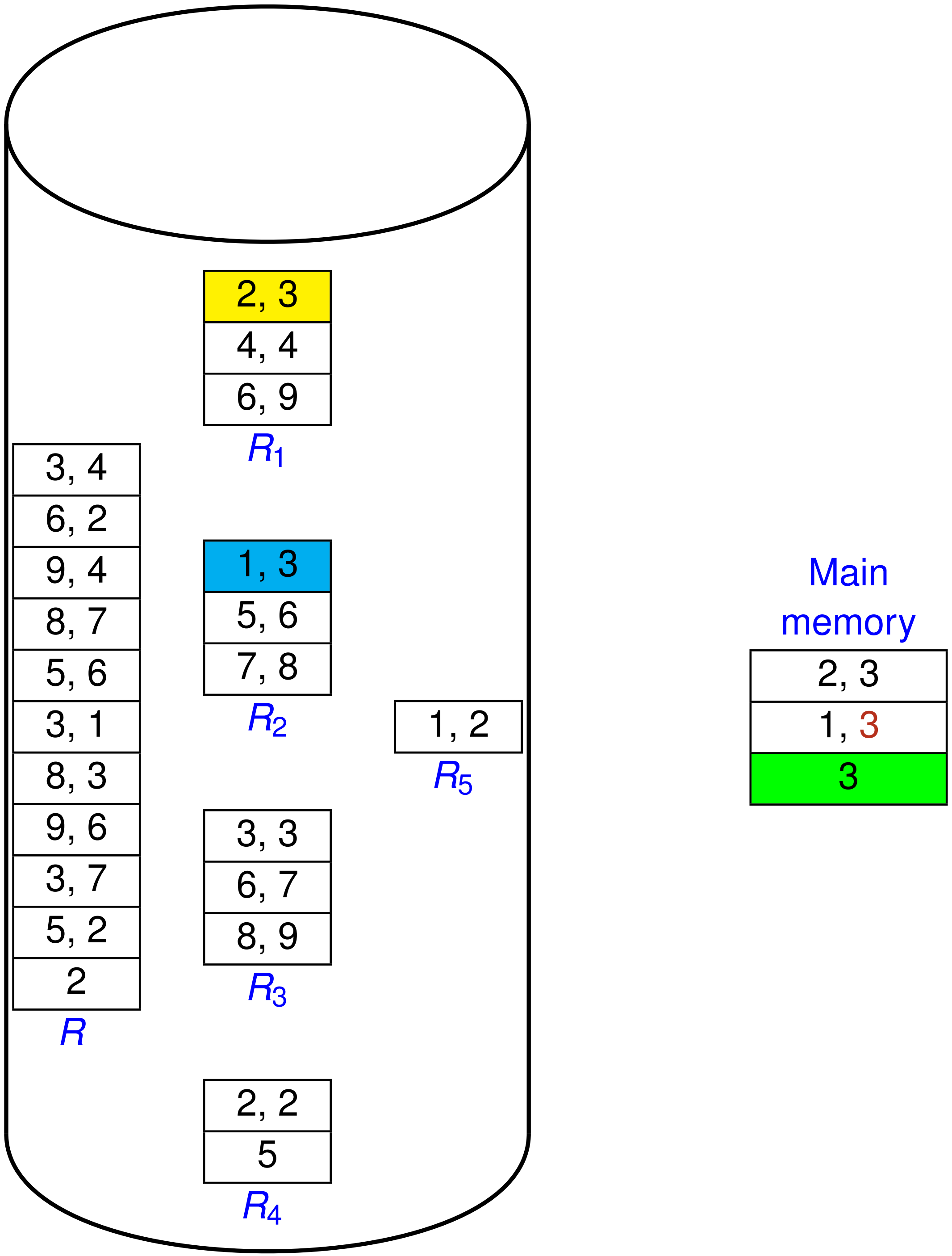


3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
2, 3
4, 4
6, 9
R_1
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
Main memory
2, 3
1, 3
1

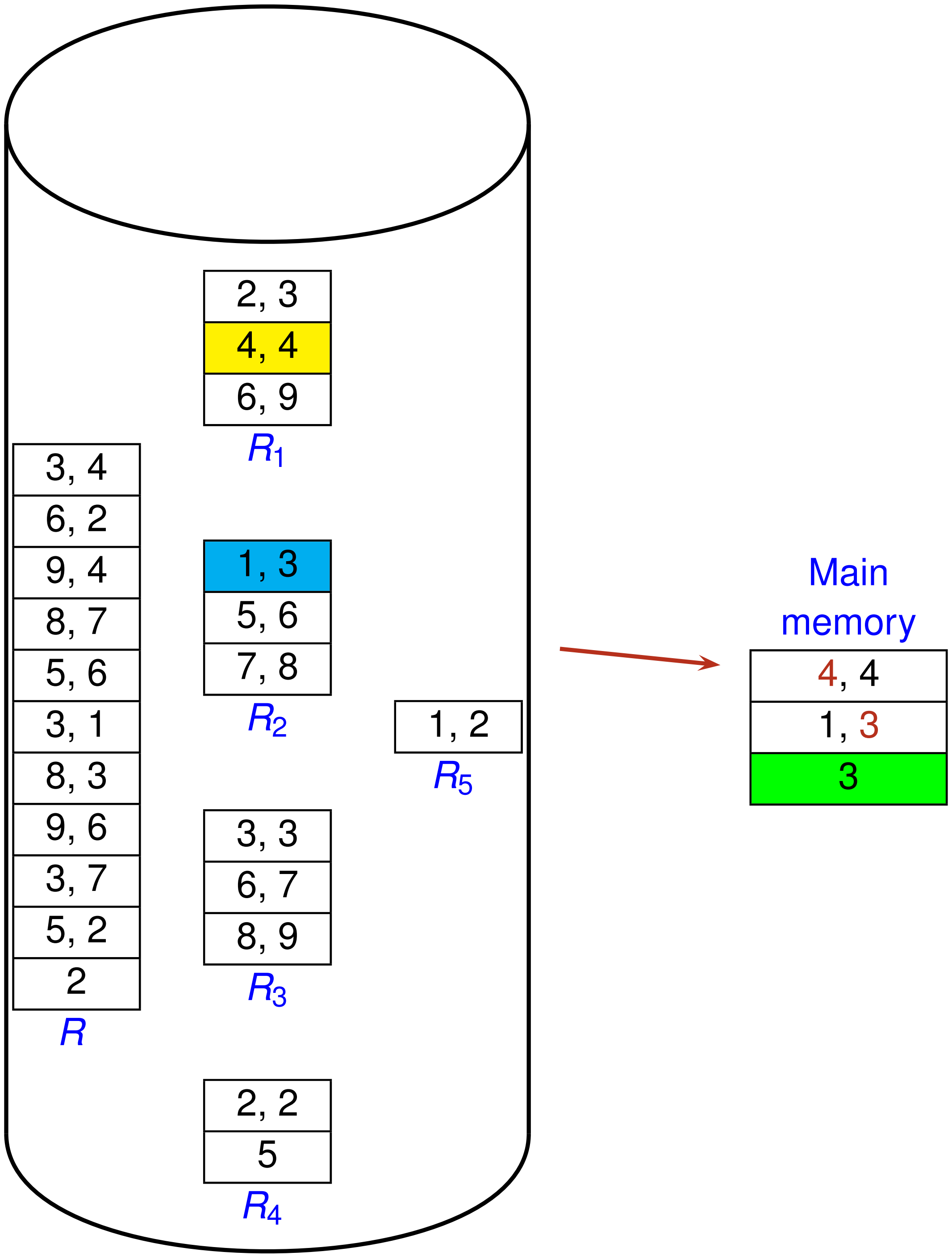

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
Main memory
2, 3
1, 3
1, 2

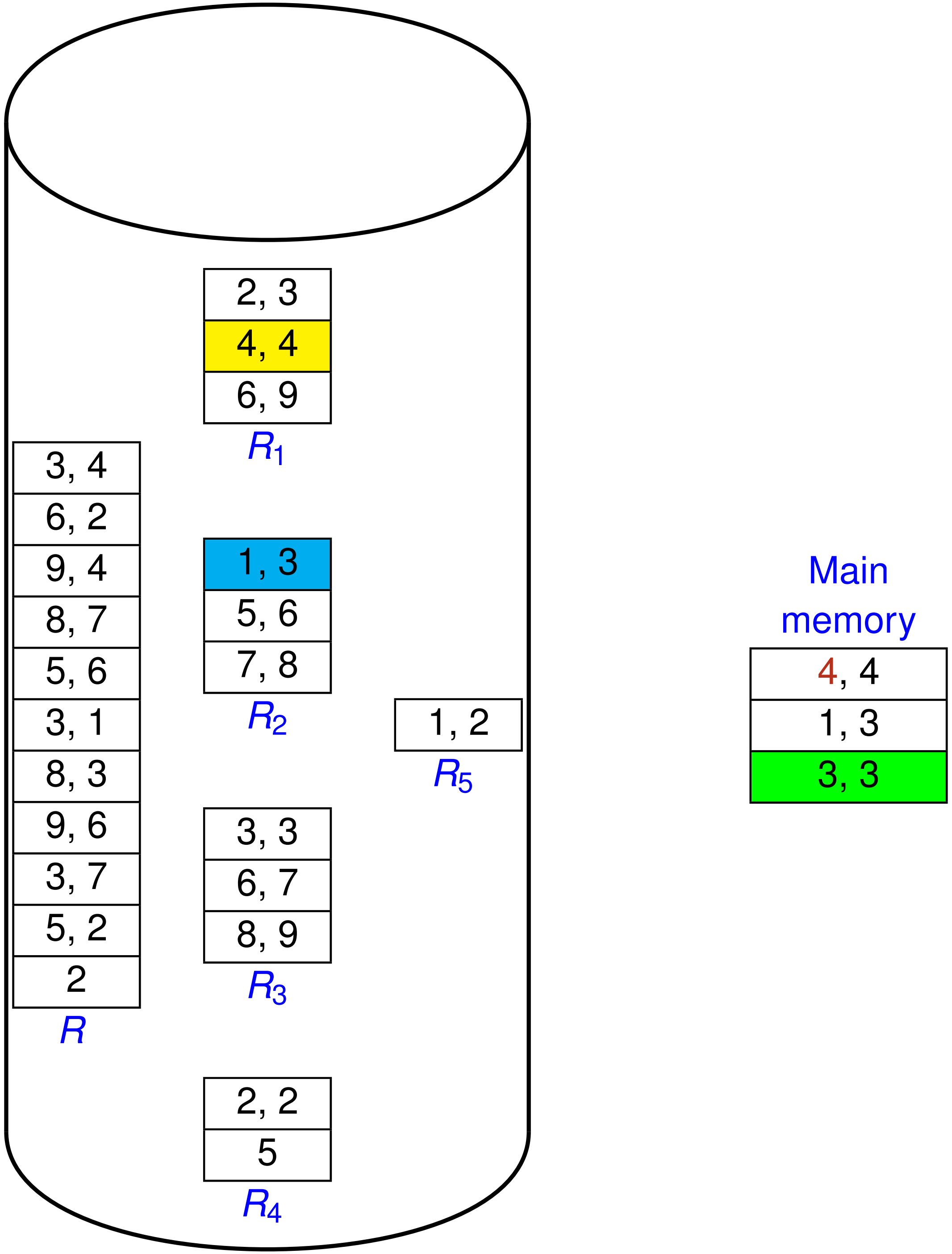


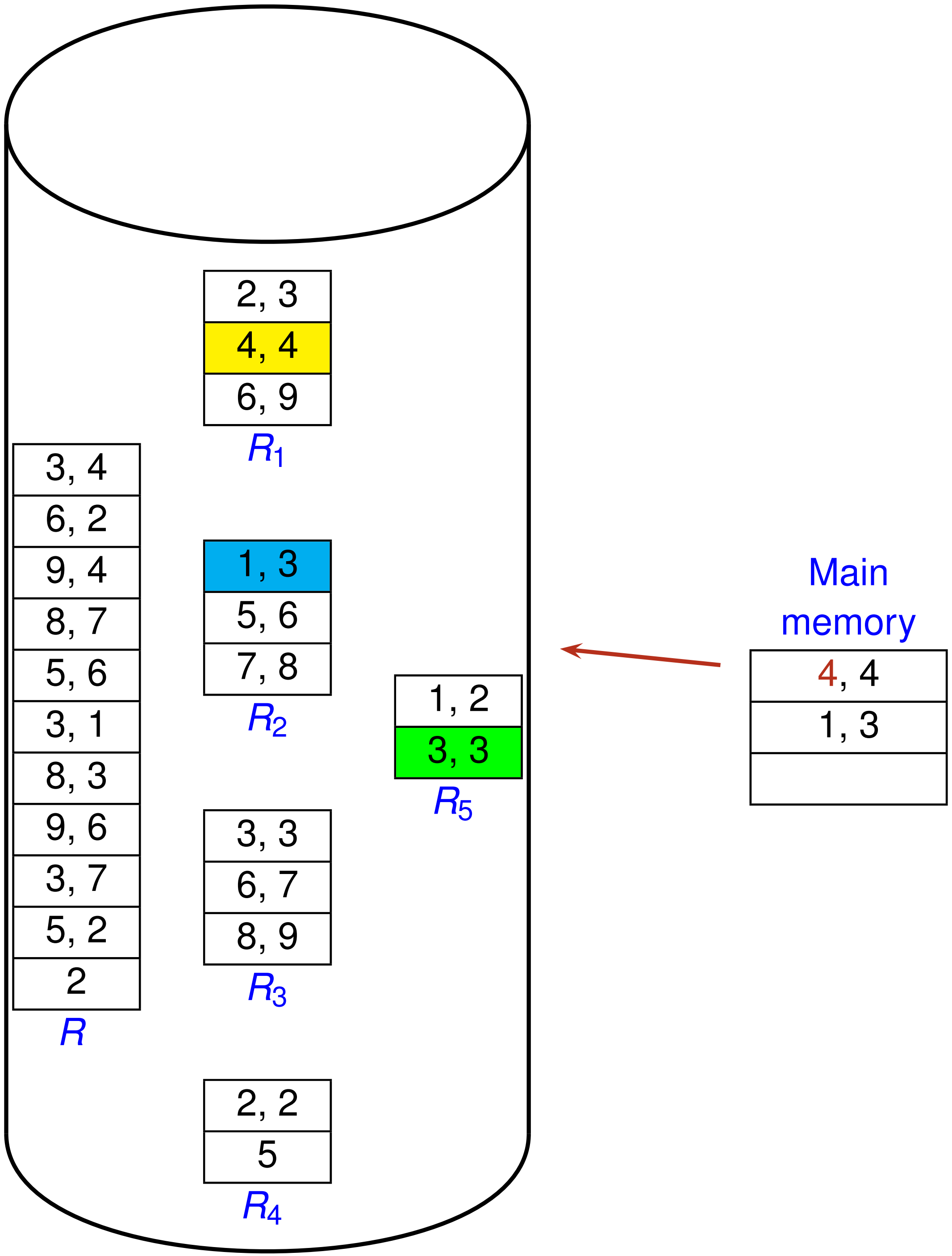
2, 3
4, 4
6, 9
R1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R2
1, 2
R5
3, 3
6, 7
8, 9
R3
2, 2
5
R4
Main memory
2, 3
1, 3

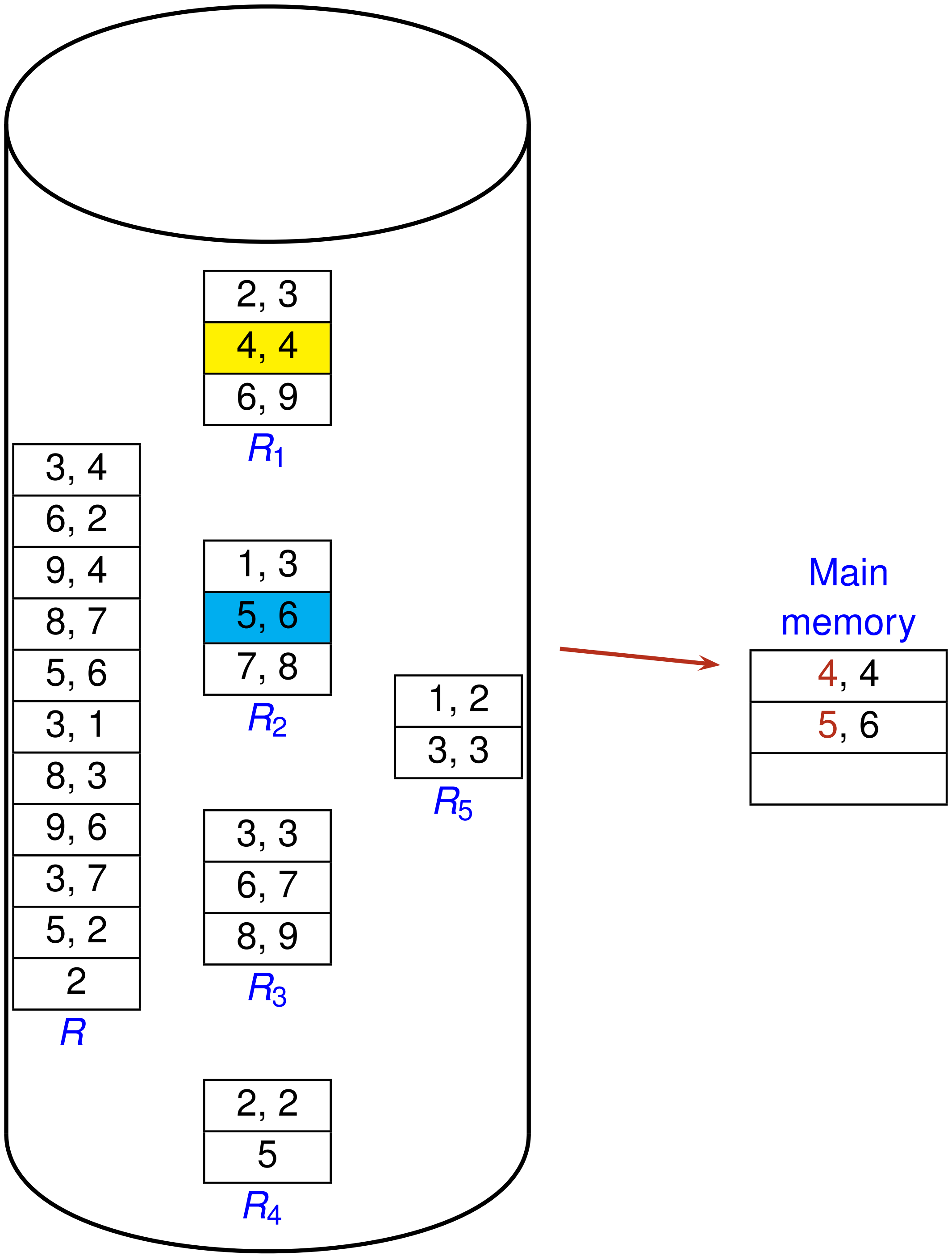


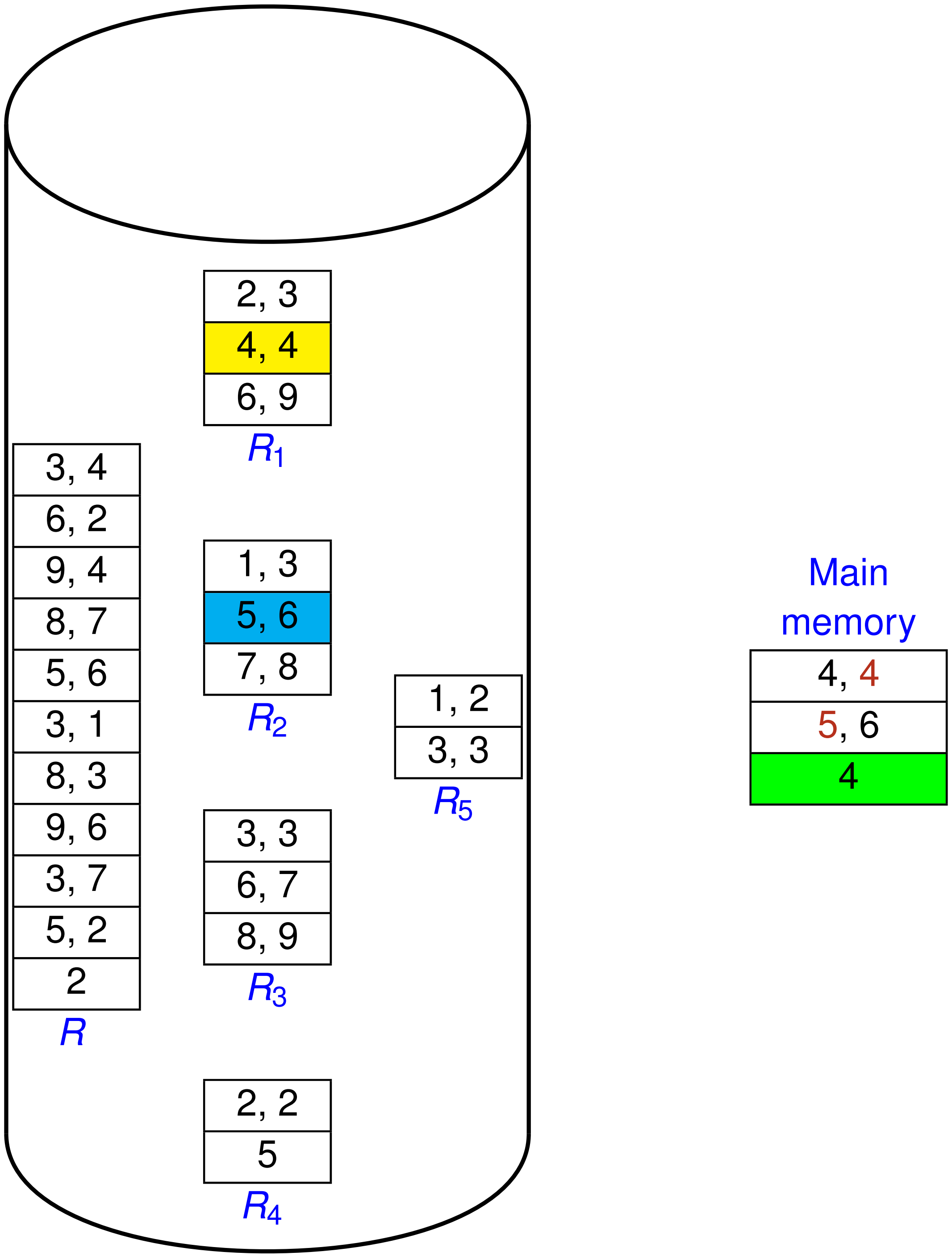
2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
1, 2
R_5
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
Main memory
2, 3
1, 3
3


2, 3
4, 4
6, 9
R1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R2
1, 2
R5
3, 3
6, 7
8, 9
R3
2, 2
5
R4
Main memory
4, 4
1, 3
3

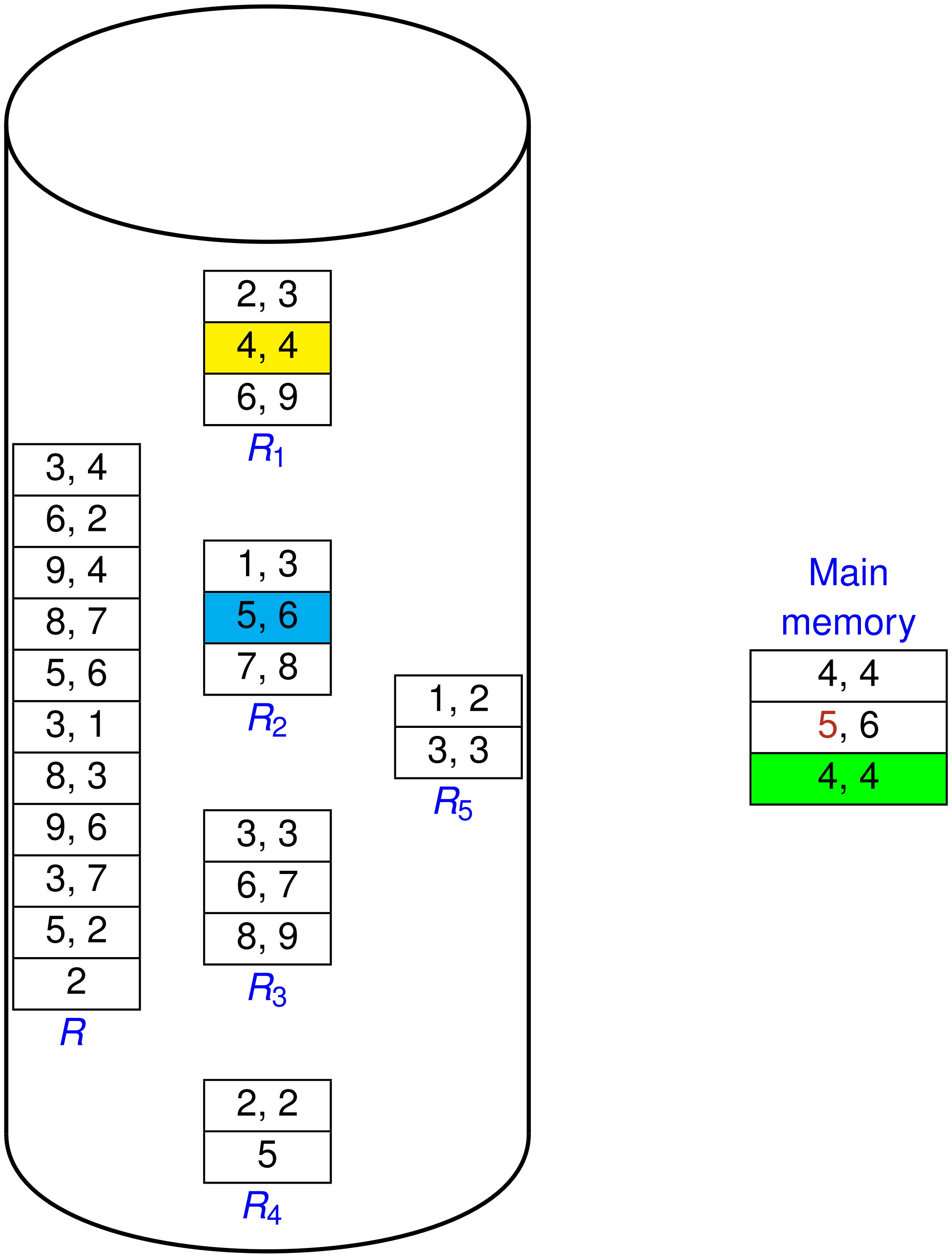

2, 3
4, 4
6, 9
R1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R2
1, 2
R5
3, 3
6, 7
8, 9
R3
2, 2
5
R4
Main
memory
4, 4
1, 3
3, 3

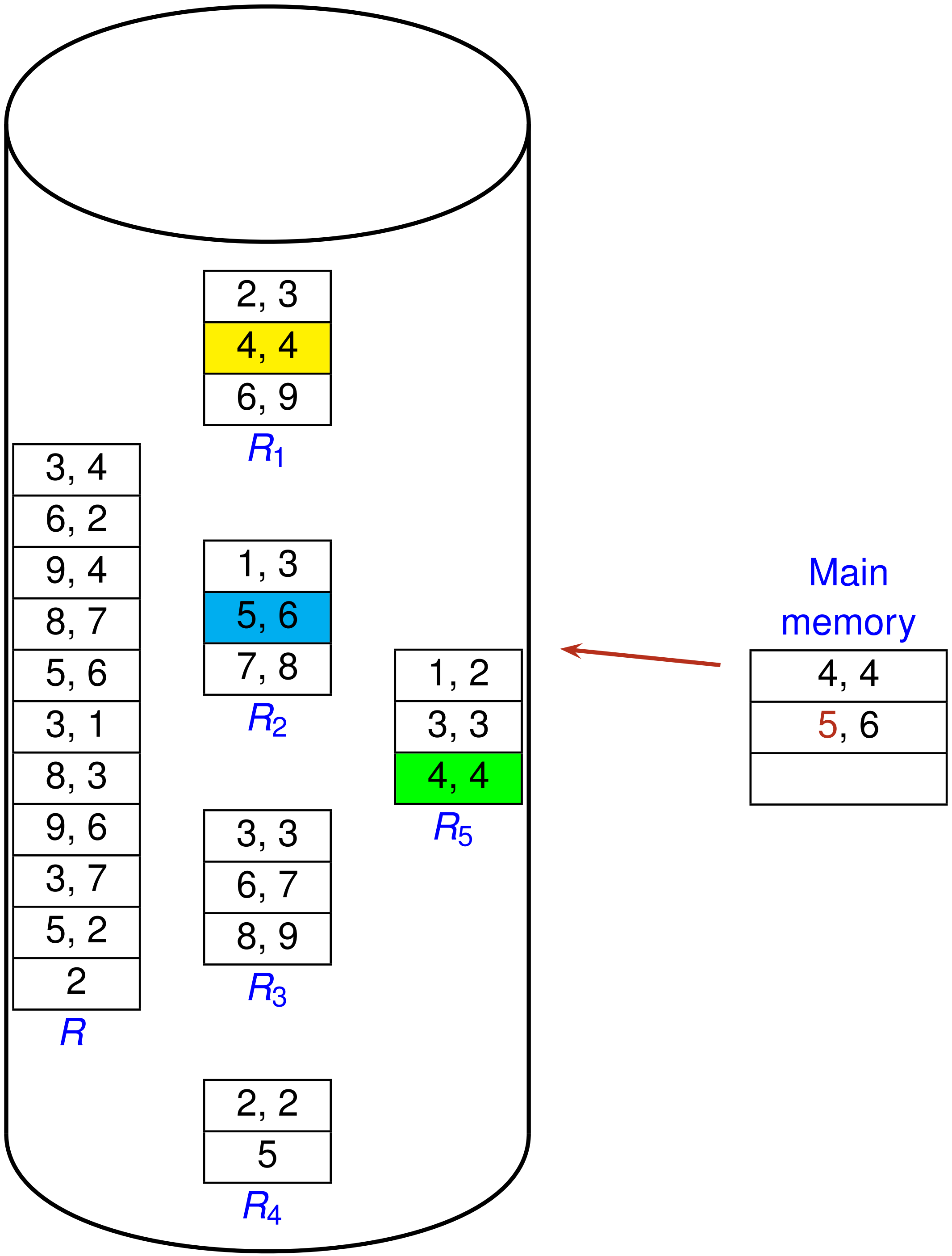

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
1, 2
3, 3
R_5
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
Main memory
4, 4
1, 3

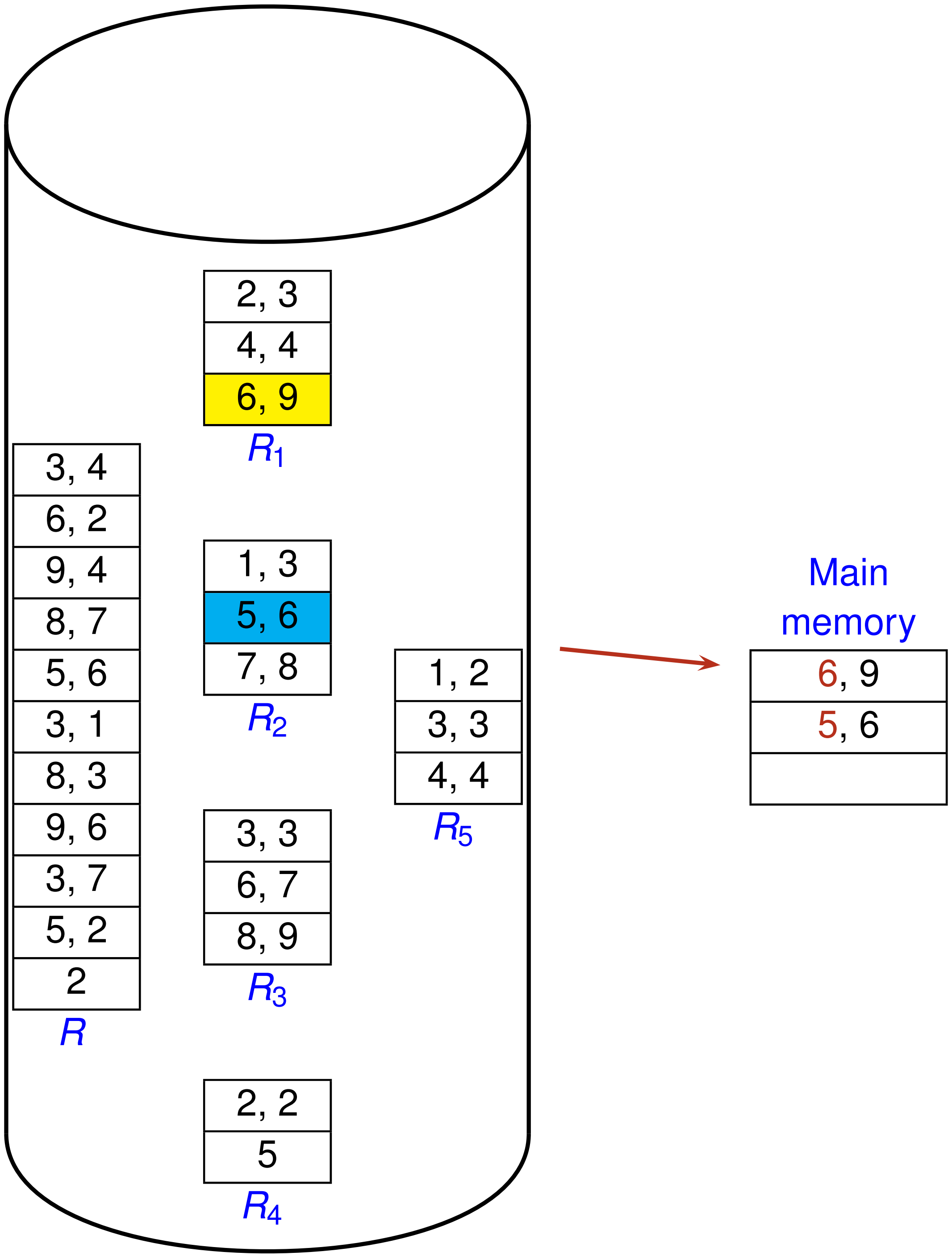

2, 3
4, 4
6, 9
R1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R2
1, 2
3, 3
R5
3, 3
6, 7
8, 9
R3
2, 2
5
R4
Main memory
4, 4
5, 6

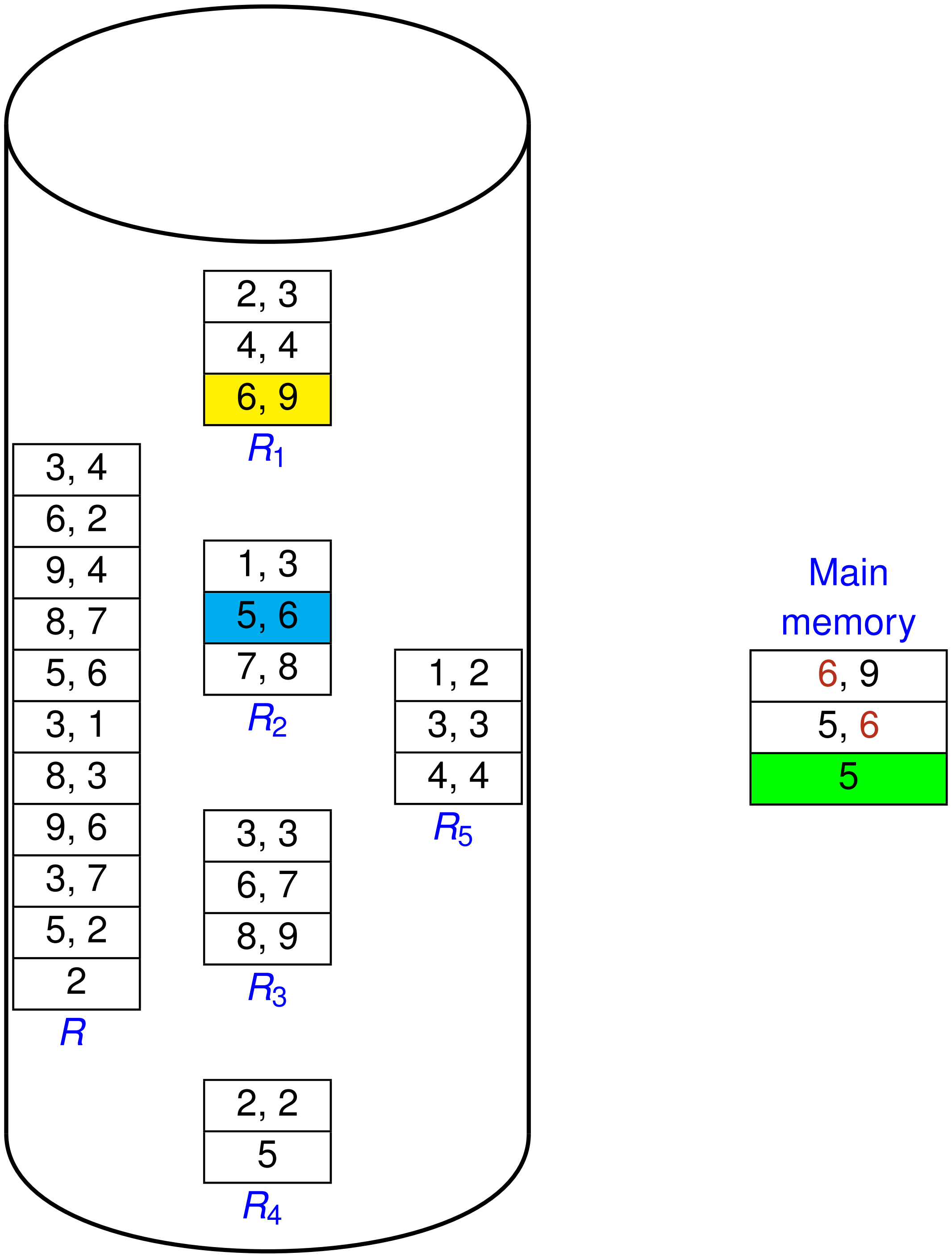


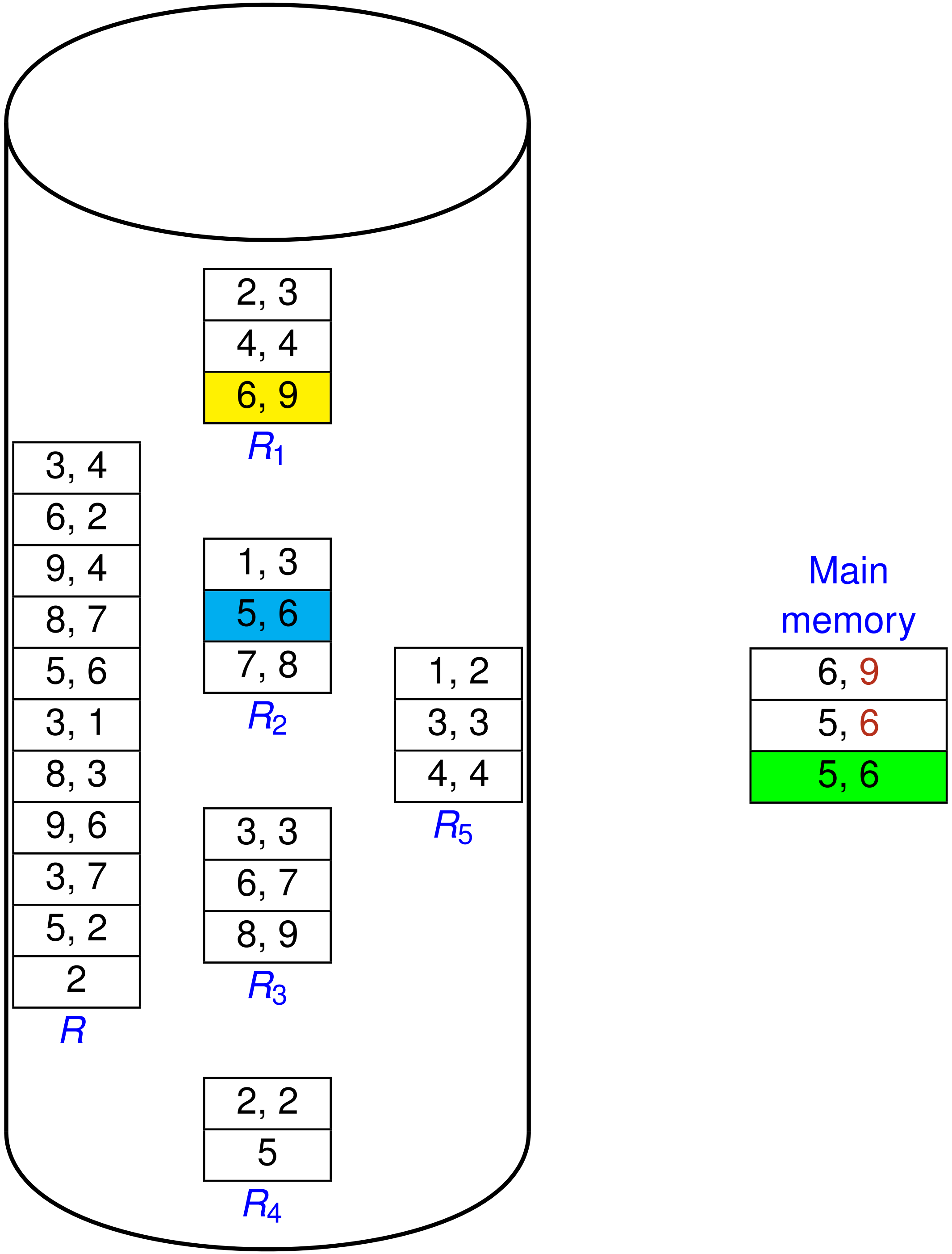
2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
1, 2
3, 3
R_5
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
Main memory
4, 4
5, 6
4

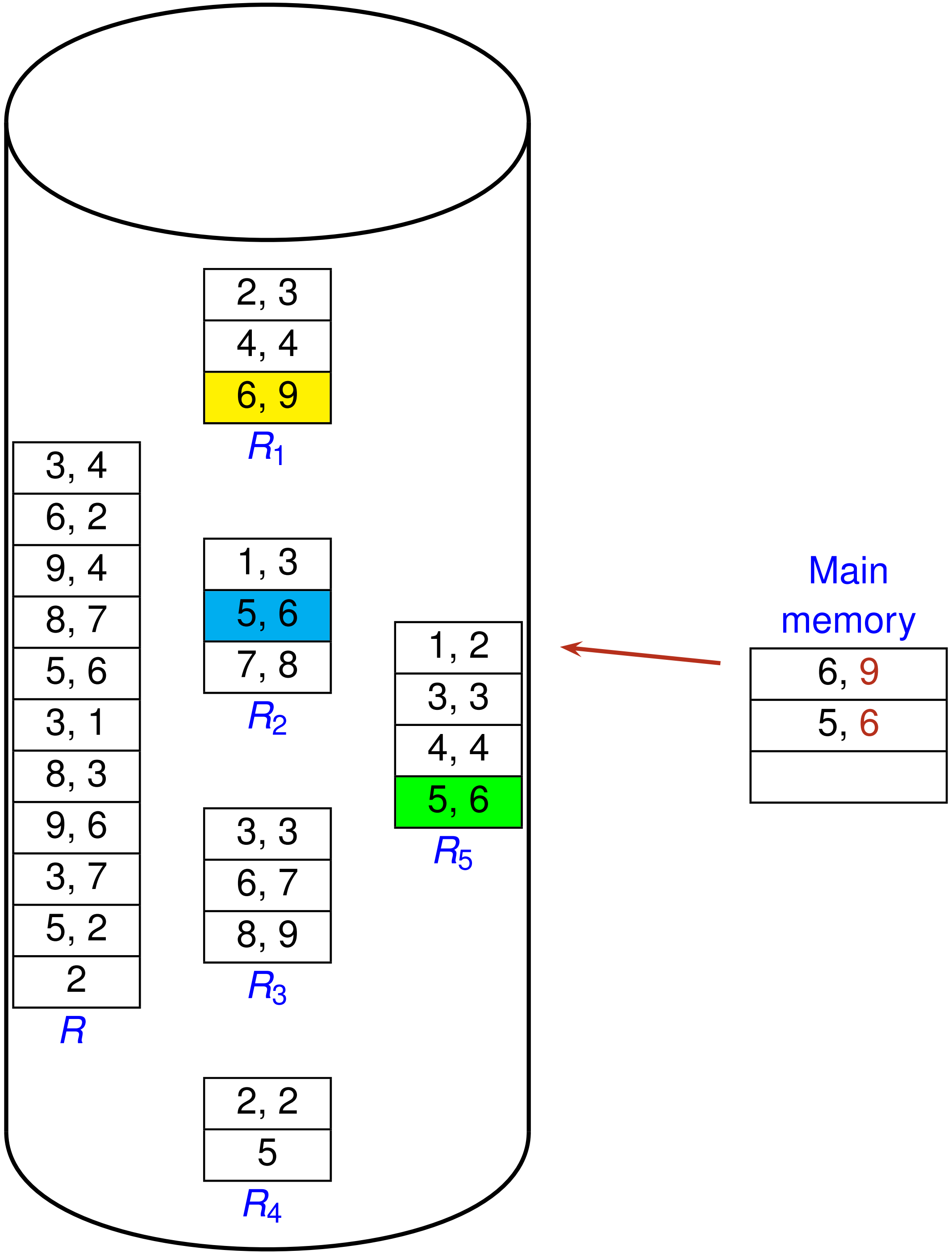

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
1, 2
3, 3
R_5
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
Main memory
4, 4
5, 6
4, 4

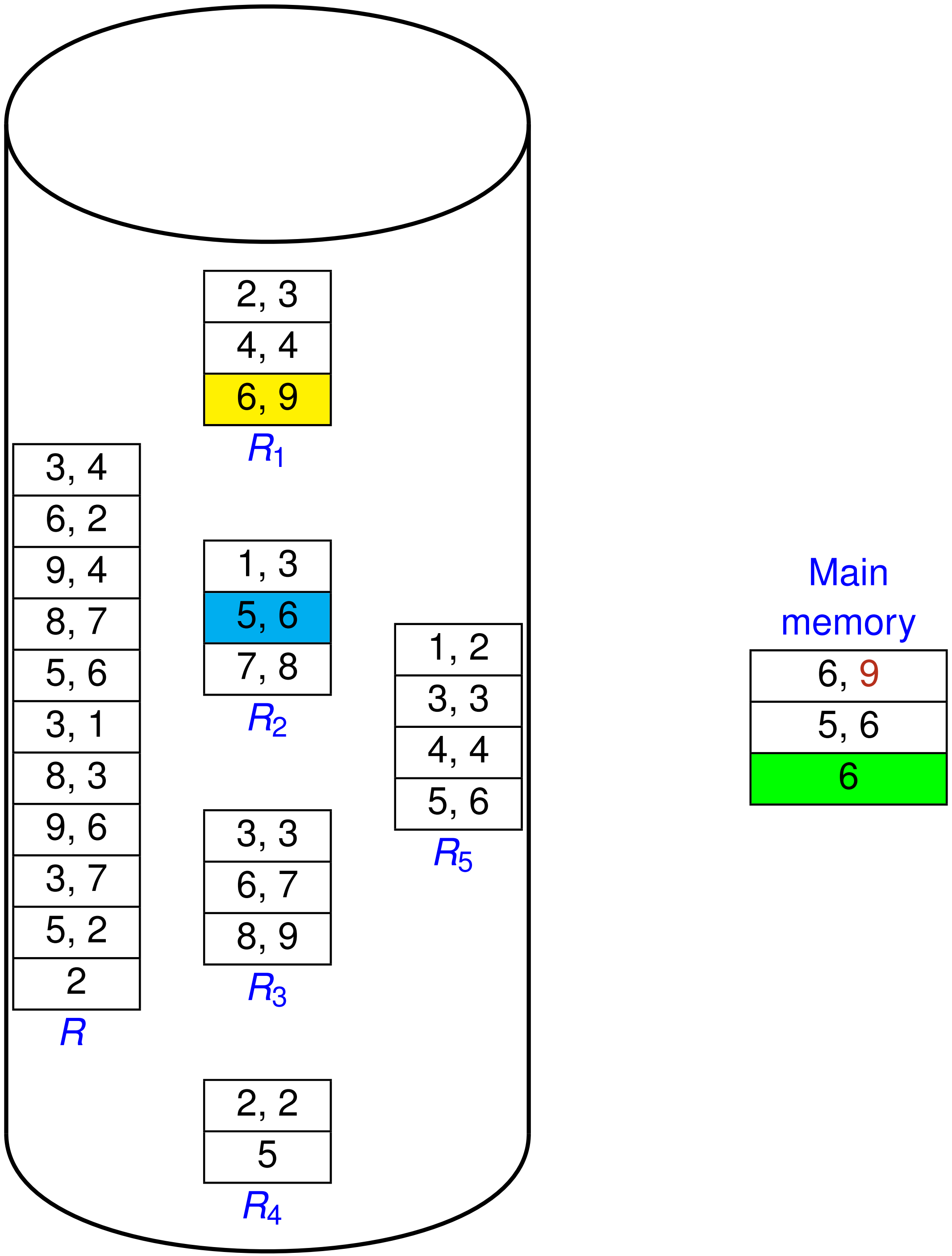

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
1, 2
3, 3
4, 4
R_5
Main memory
4, 4
5, 6

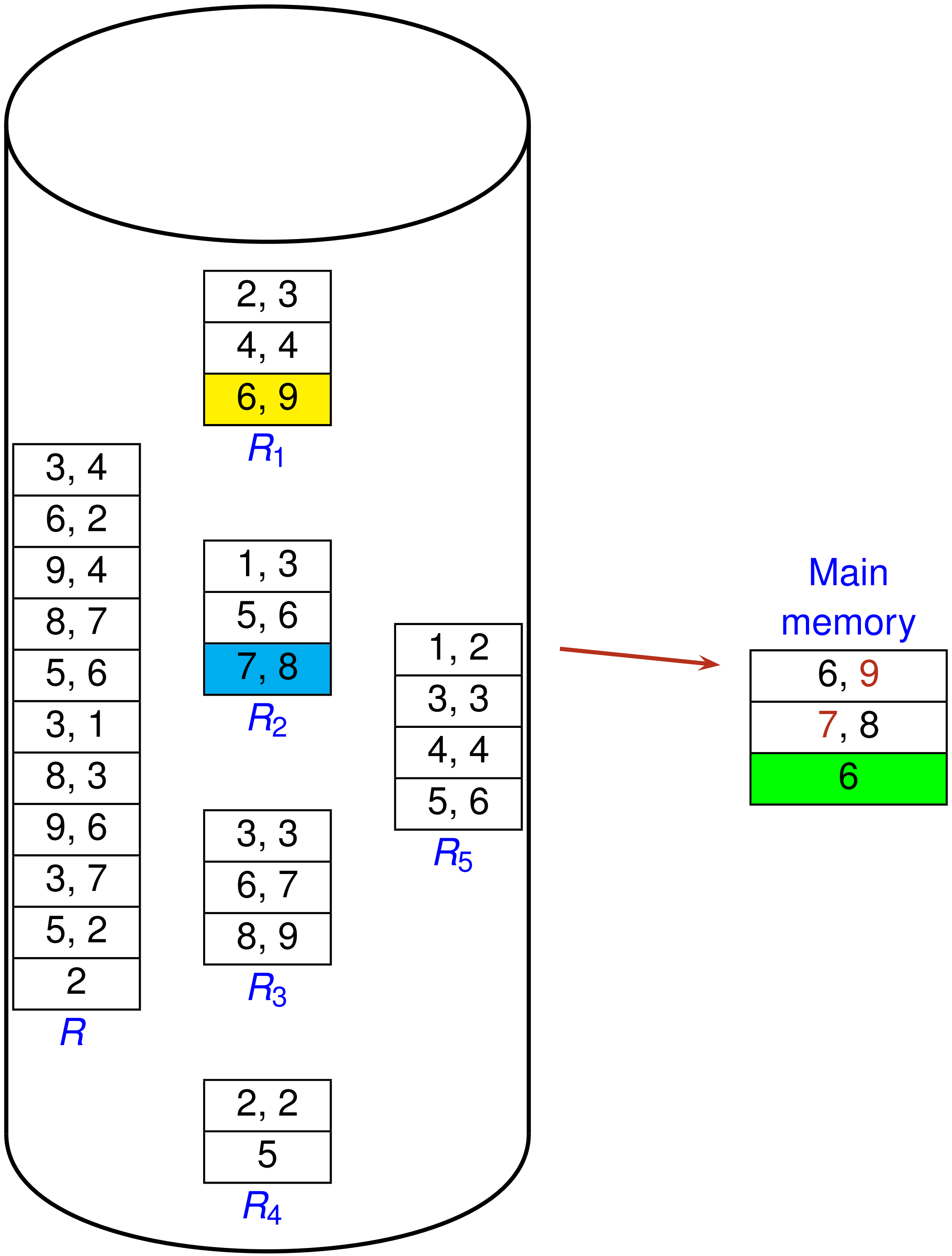

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
1, 2
3, 3
4, 4
R_5
Main memory
6, 9
5, 6

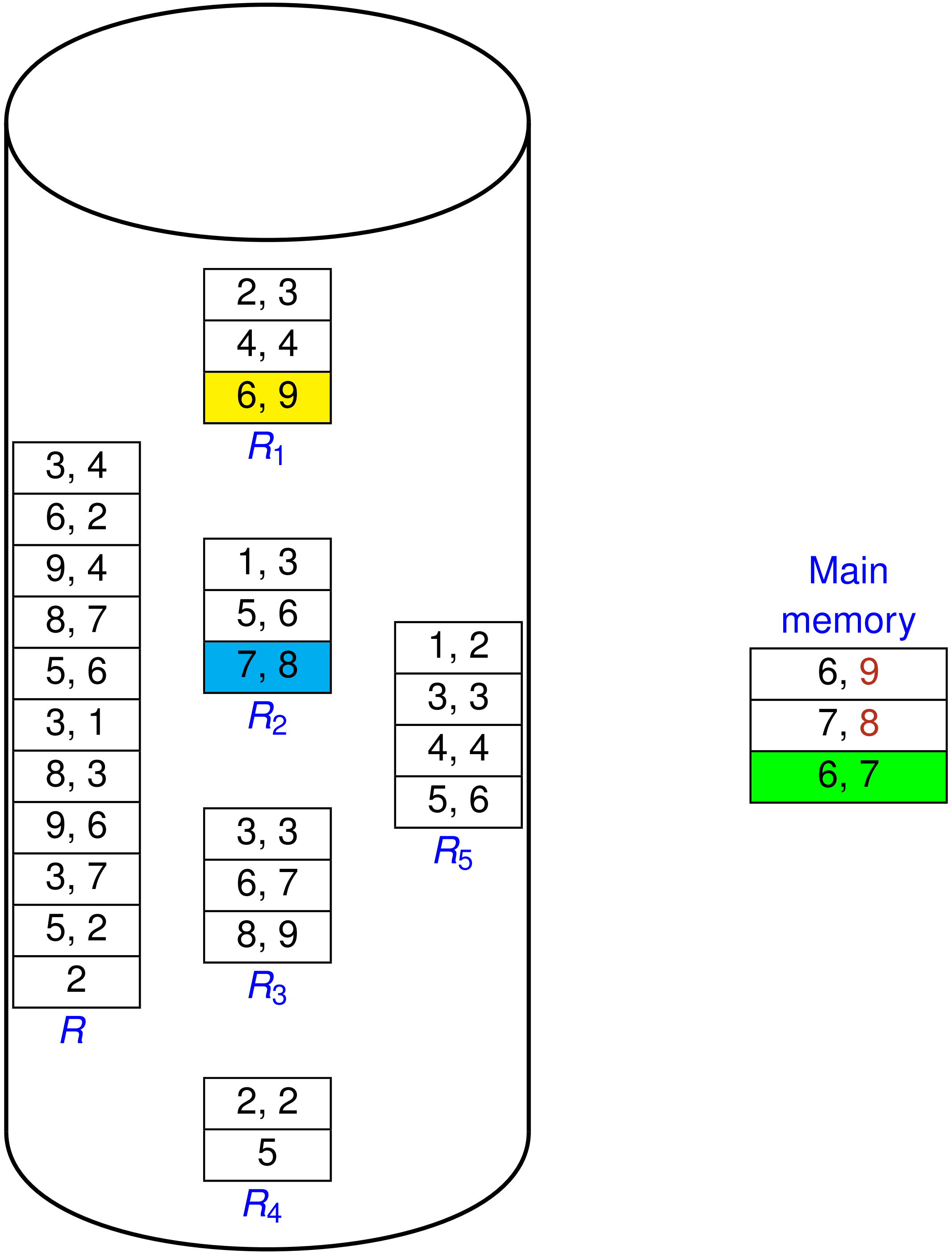

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
1, 2
3, 3
4, 4
R_5
Main memory
6, 9
5, 6
5

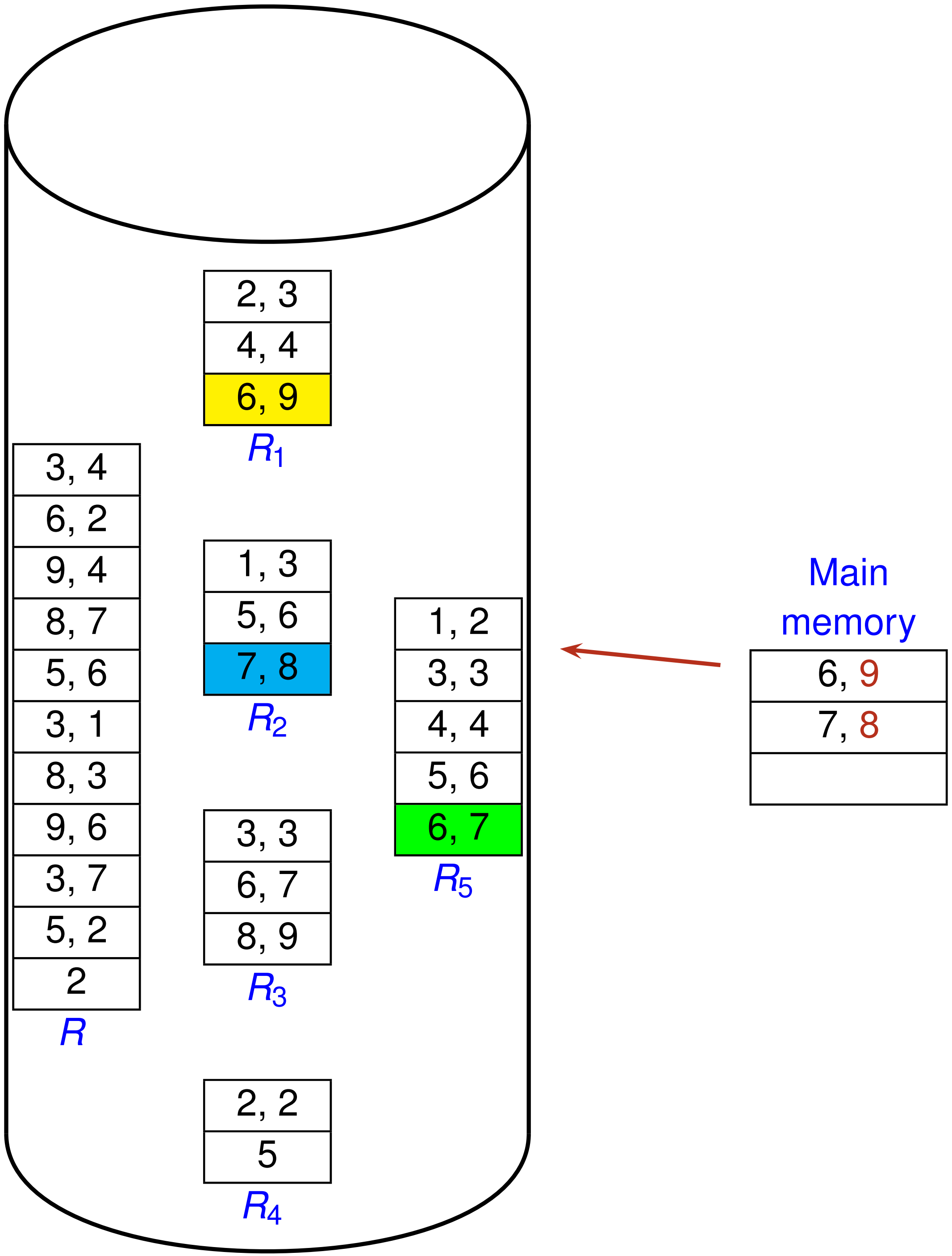

3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
2, 3
4, 4
6, 9
R1
1, 3
5, 6
7, 8
R2
3, 3
6, 7
8, 9
R3
2, 2
5
R4
1, 2
3, 3
4, 4
R5
Main memory
6, 9
5, 6
5, 6

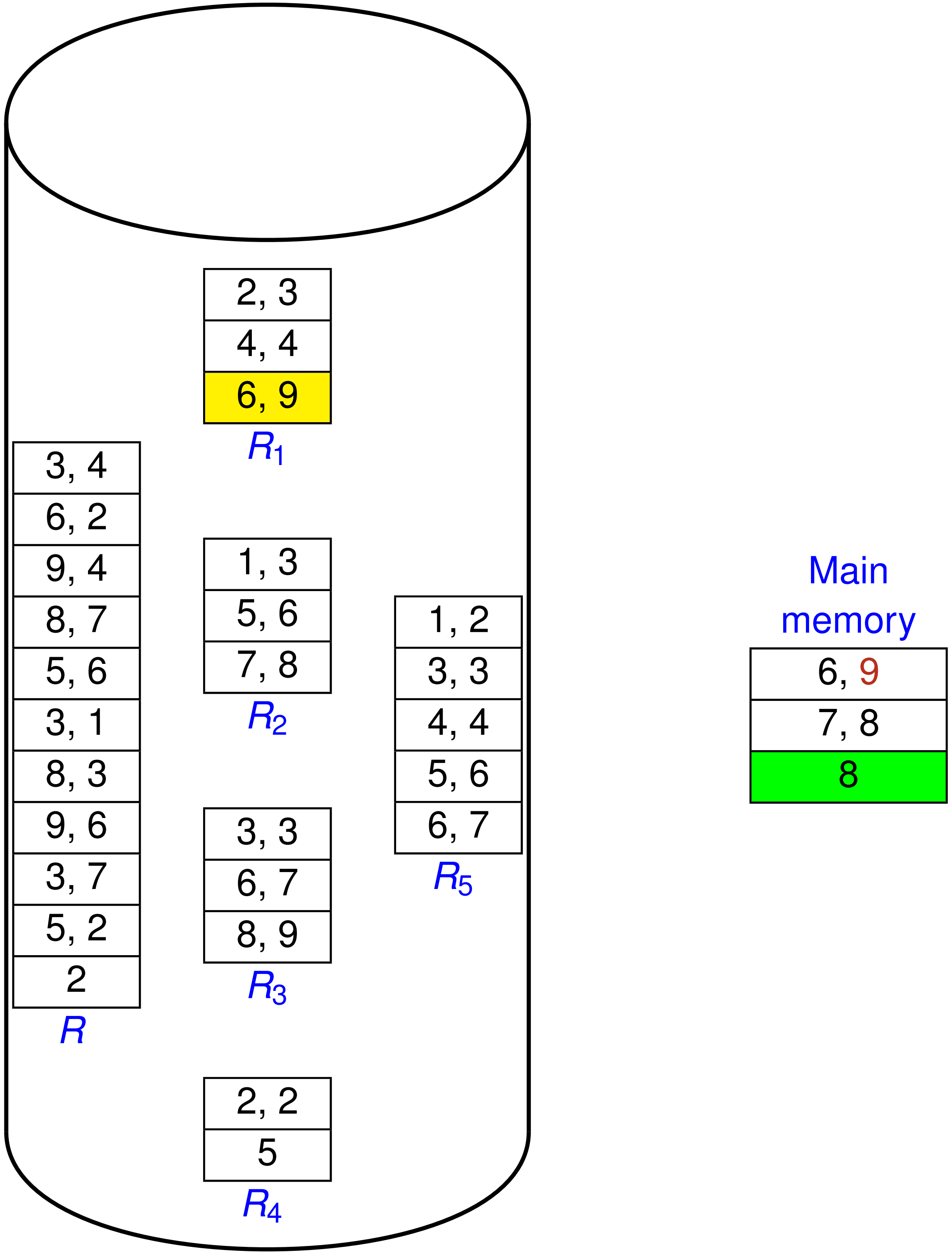

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
1, 2
3, 3
4, 4
5, 6
R_5
Main memory
6, 9
5, 6

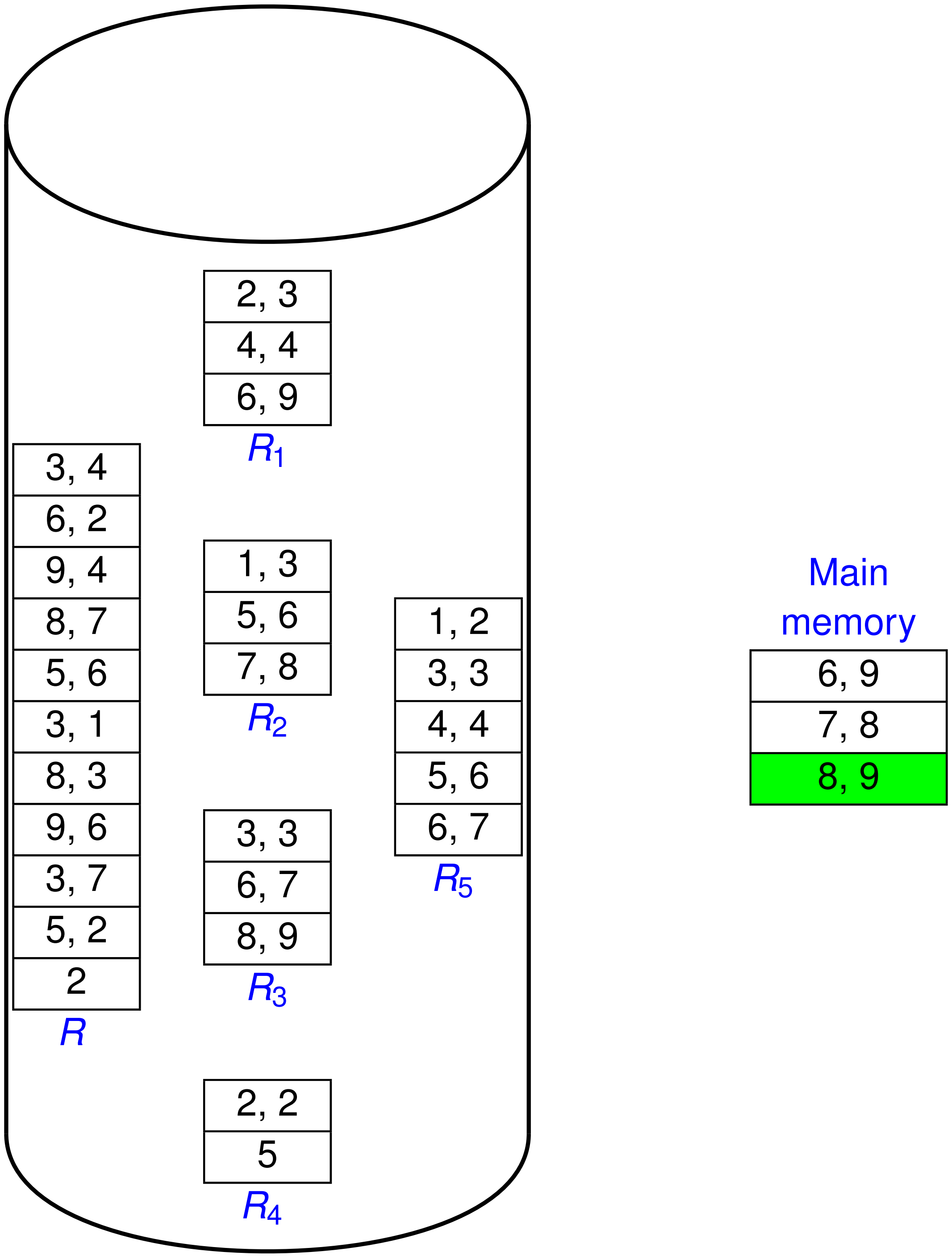

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
1, 2
3, 3
4, 4
5, 6
R_5
Main memory
6, 9
5, 6
6

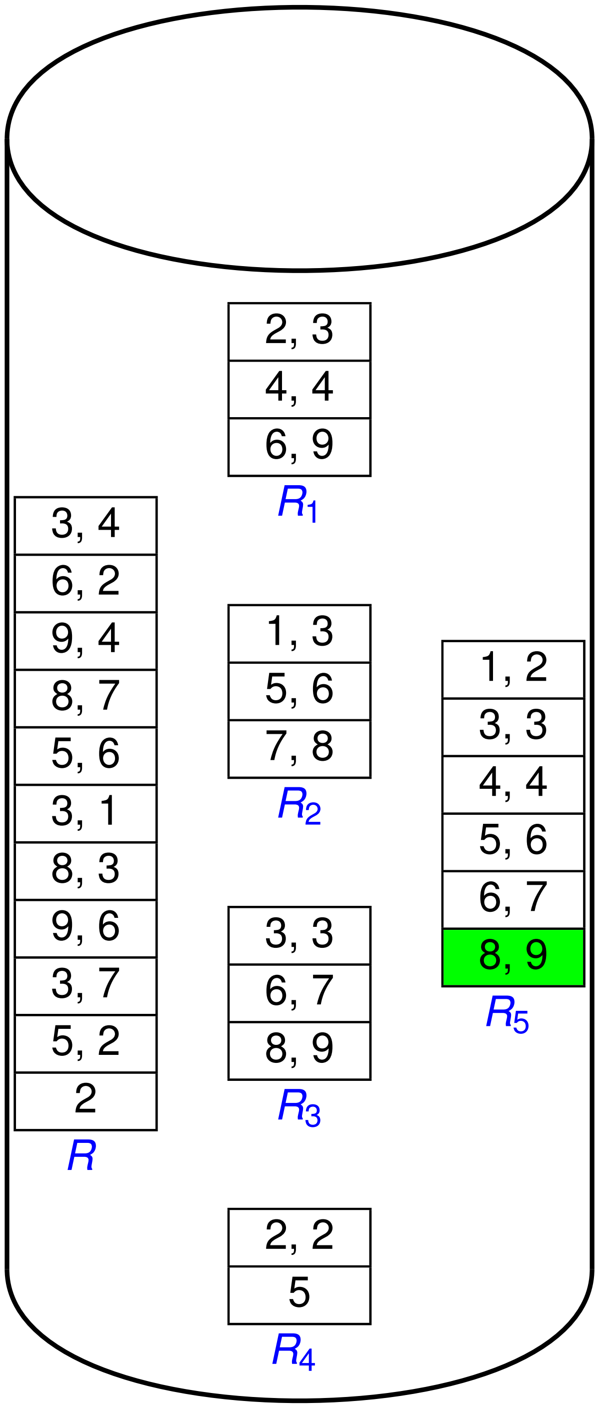

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
1, 2
3, 3
4, 4
5, 6
R_5
Main memory
6, 9
7, 8
6


2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
1, 2
3, 3
4, 4
5, 6
R_5
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
Main memory
6, 9
7, 8
6, 7

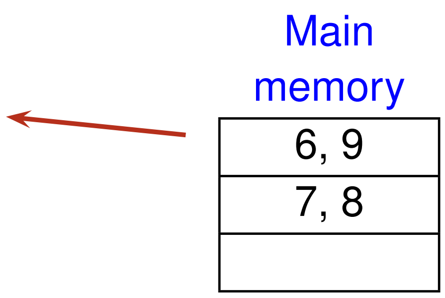

2, 3
4, 4
6, 9
R_1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R_2
3, 3
6, 7
8, 9
R_3
2, 2
5
R_4
1, 2
3, 3
4, 4
5, 6
6, 7
R_5
Main memory
6, 9
7, 8


2, 3
4, 4
6, 9
R1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R2
3, 3
6, 7
8, 9
R3
2, 2
5
R4
1, 2
3, 3
4, 4
5, 6
6, 7
R5
Main memory
6, 9
7, 8
8

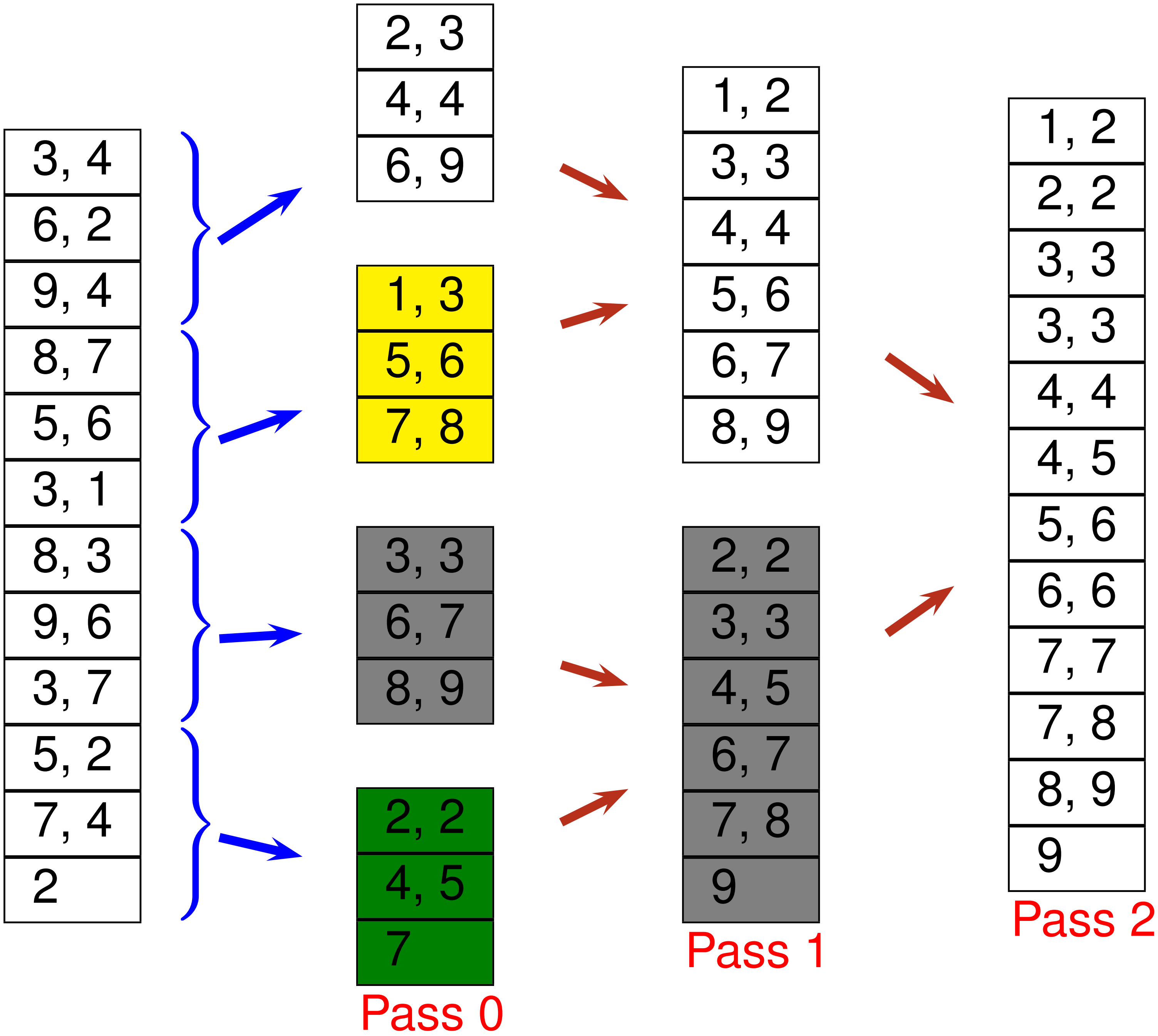

2, 3
4, 4
6, 9
R1
3, 4
6, 2
9, 4
8, 7
5, 6
3, 1
8, 3
9, 6
3, 7
5, 2
2
R
1, 3
5, 6
7, 8
R2
3, 3
6, 7
8, 9
R3
2, 2
5
R4
1, 2
3, 3
4, 4
5, 6
6, 7
R5
Main memory
6, 9
7, 8
8, 9



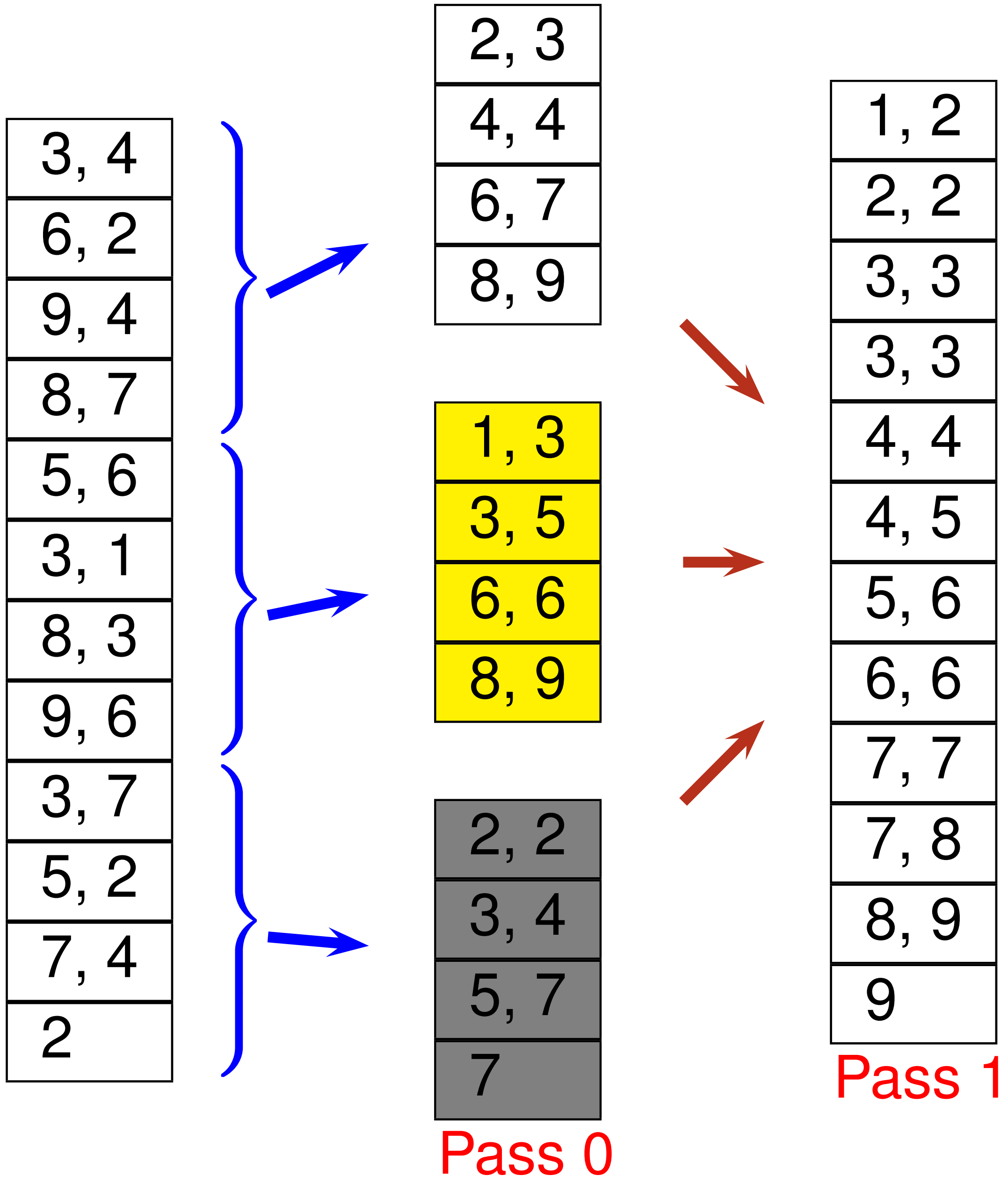
R_1 & R_2 have been merged into R_5
Next, merge R_3 & R_4 to create R_6
Finally, merge R_5 & R_6 to create final sorted run



External Merge Sort with 3 Memory Pages



External Merge Sort with 4 Memory Pages



External Merge Sort

- File size of N pages
- Uses B number of buffer pages
- Pass 0: Creation of sorted runs
  - Read in and sort B pages at a time
  - Number of sorted runs created = $\lceil N/B \rceil$
  - Size of each sorted run = B pages (except possibly for last run)
- Pass i, $i \geq 1$: Merging of sorted runs
  - Use $B-1$ buffer pages for input & one buffer page for output
  - Performs (B-1)-way merge
- Analysis:
  - N_0 = number of sorted runs created in pass 0 = $\lceil N/B \rceil$
  - Total number of passes = $\lceil \log_{B-1}(N_0) \rceil + 1$
  - Total number of I/O = $2N(\lceil \log_{B-1}(N_0) \rceil + 1)$
    - Each pass reads N pages & writes N pages

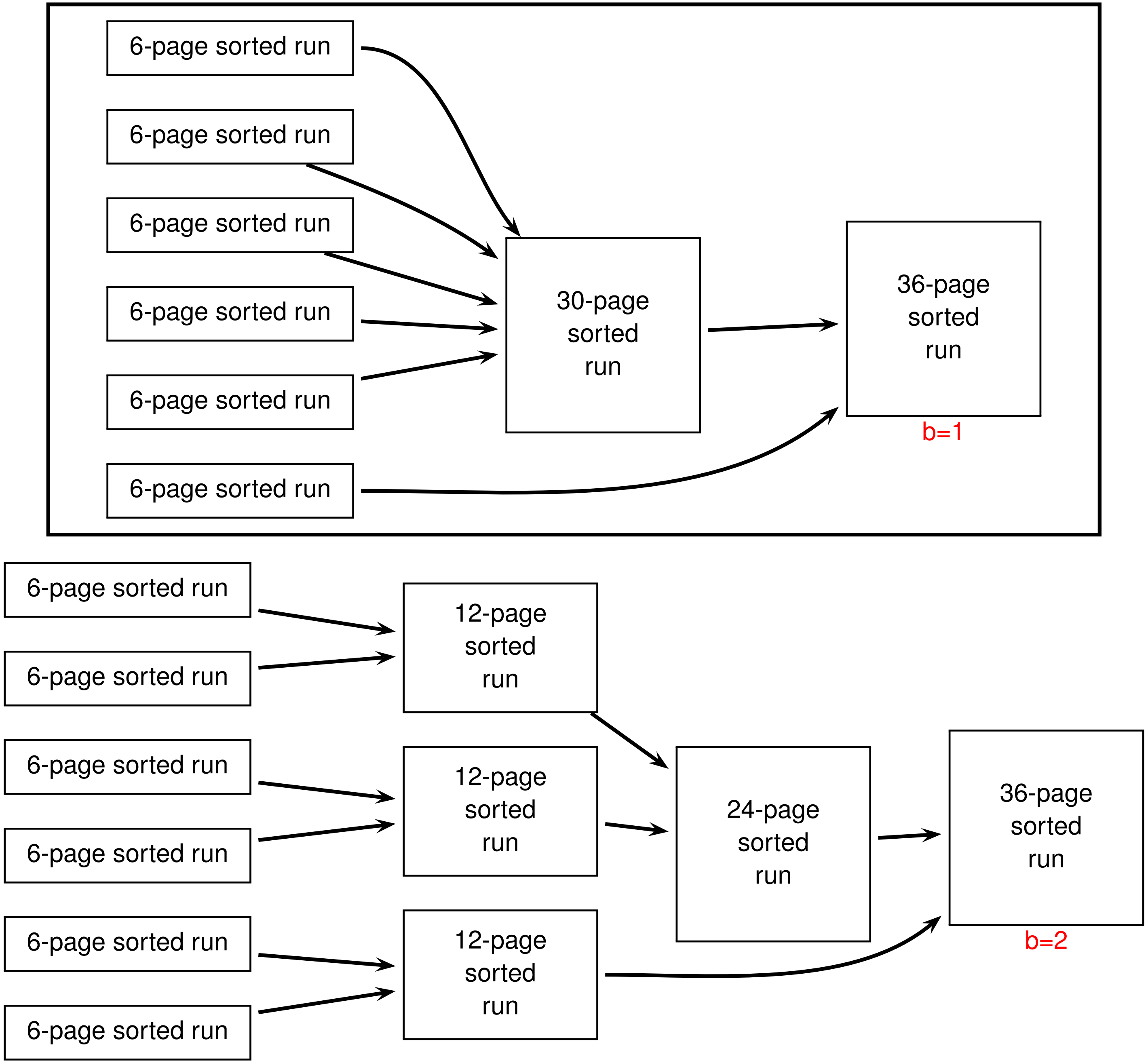
Number of Passes of External Merge Sort, $\lceil \log_{B-1}(\lceil N/B \rceil) \rceil + 1$

Num. of data pages, N	Num. of buffer pages, B					
	3	5	9	17	129	257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Optimization with Blocked I/O

- Read and write in units of buffer blocks of *b* pages
- Given an allocation of B buffer pages for sorting,
  - Allocate one block (*b* pages) for output
  - Remainder space can accommodate $\lfloor \frac{B-b}{b} \rfloor$ blocks for input
  - Thus, can merge at most $\lfloor \frac{B-b}{b} \rfloor$ sorted runs in each merge pass
- Analysis:
  - N = number of pages in file to be sorted
  - B = number of available buffer pages
  - b = number of pages of each buffer block
  - N_0 = number of initial sorted runs = $\lceil N/B \rceil$
  - F = number of runs that can be merged at each merge pass $= \lfloor \frac{B}{b} \rfloor - 1$
  - Number of passes = $\lceil \log_F(N_0) \rceil + 1$

Blocked I/O: Sorting 36-page data with B=6



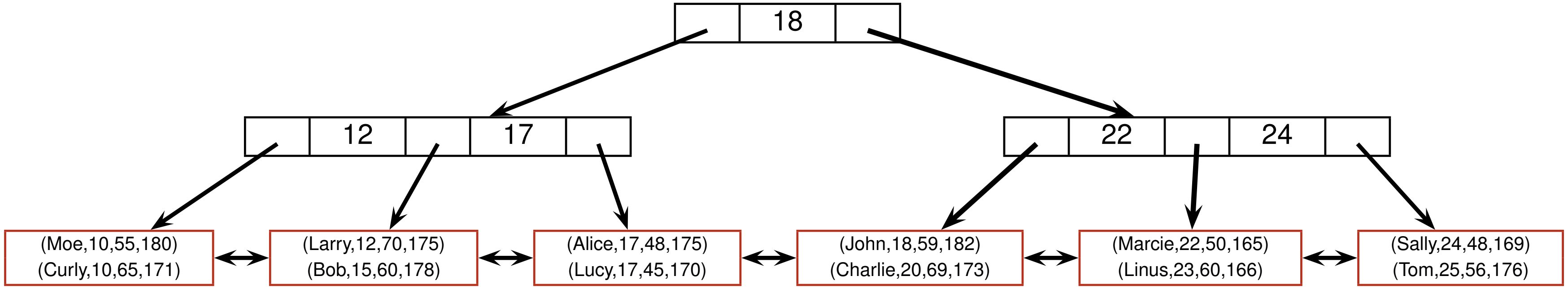
Sorting using B^+-trees

- When table to be sorted has a B^+-tree index on sorting attributes
- Example: Sort Student table on year using B^+-tree index on (year,major)
- How:
  1. Format 1: Sequentially scan leaf pages of B^+-tree
  2. Format 2 or 3:
     - Sequentially scan leaf pages of B^+-tree
     - For each leaf page visited, retrieve data records using RIDs

Clustered vs Unclustered Index

- An index is a clustered index if the order of its data entries is the same as or ‘close to’ the order of the data records; otherwise, it is an unclustered index
- An index using Format 1 for its data entries is a clustered index
- There is at most one clustered index for each relation

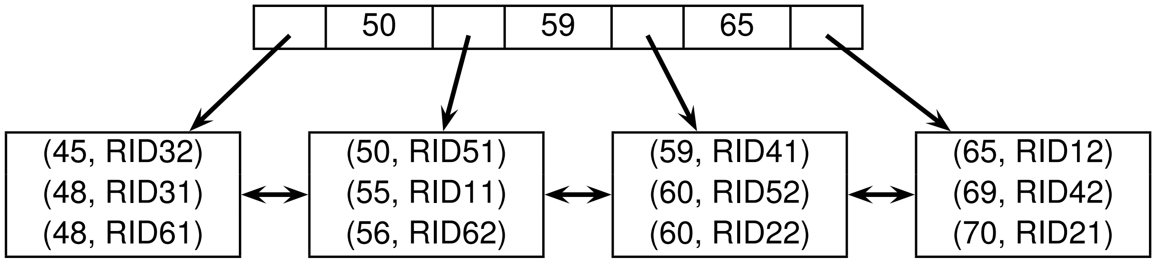
Clustered vs Unclustered Index: Example



Clustered index on R.age

Relation R

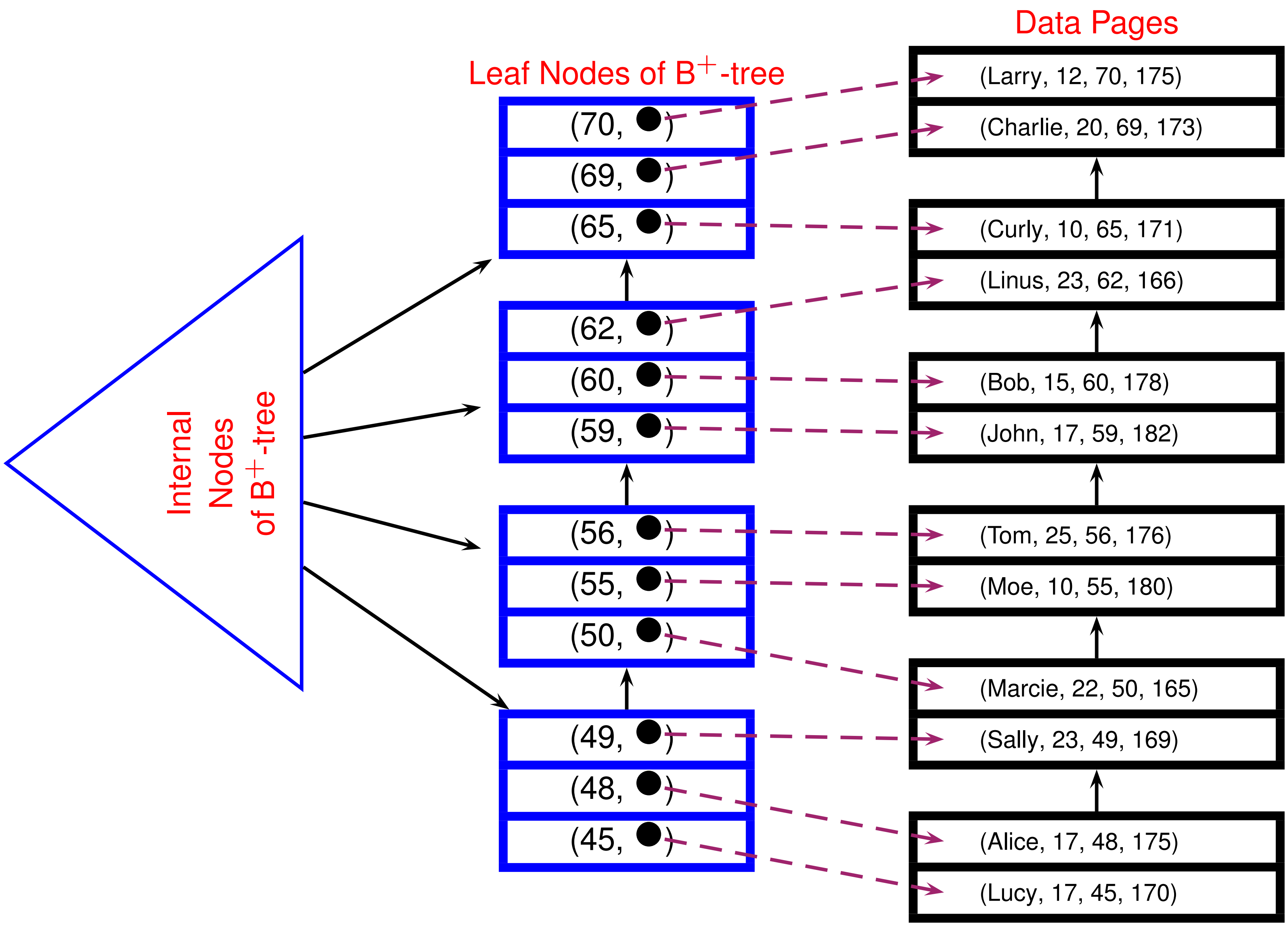
name	age	weight	height
Moe	10	55	180
Curly	10	65	171
Larry	12	70	175
Bob	15	60	178
Alice	17	48	175
Lucy	17	45	170
John	18	59	182
Charlie	20	69	173
Marcie	22	50	165
Linus	23	60	166
Sally	24	48	169
Tom	25	56	176



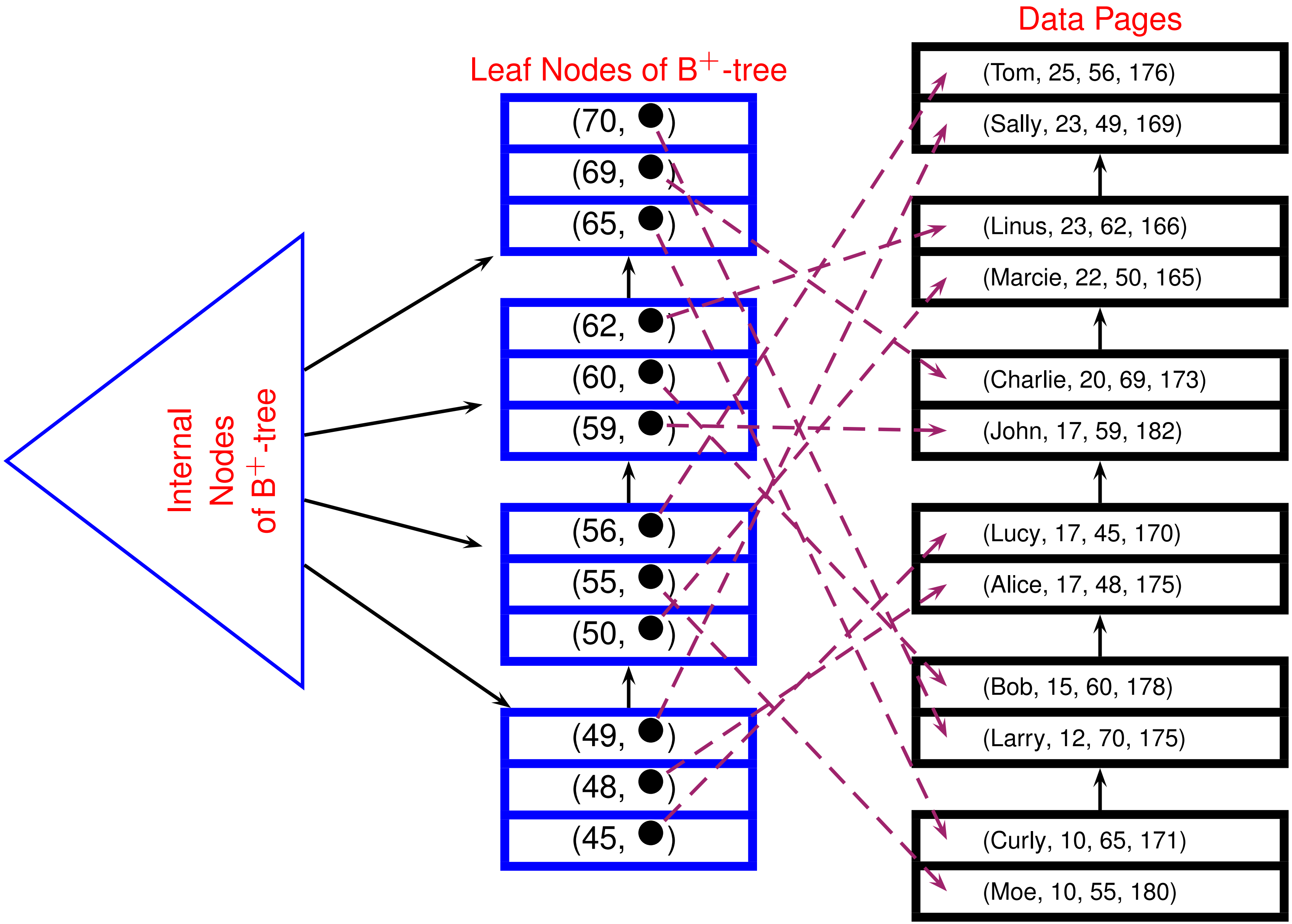
Unclustered index on R.weight

(RIDij = slot j on data page i)

Clustered B$^+$-tree on R.weight



Unclustered B^+-tree on R.weight



Selection: $\sigma_p(R)$

- $\sigma_p(R)$ selects rows from relation R that satisfy selection predicate p
- **Example:**

Relation R

name	age	weight	height
Moe	10	55	180
Curly	10	65	171
Larry	12	70	175
Bob	15	60	178
Alice	17	48	175
Lucy	17	45	170
John	18	59	182
Charlie	20	69	173
Marcie	22	50	165
Linus	23	60	166
Sally	24	48	169
Tom	25	56	176

$\sigma_{(\textbf{weight}>\textbf{64})\wedge(\textbf{height}>\textbf{170})}(\textbf{R})$

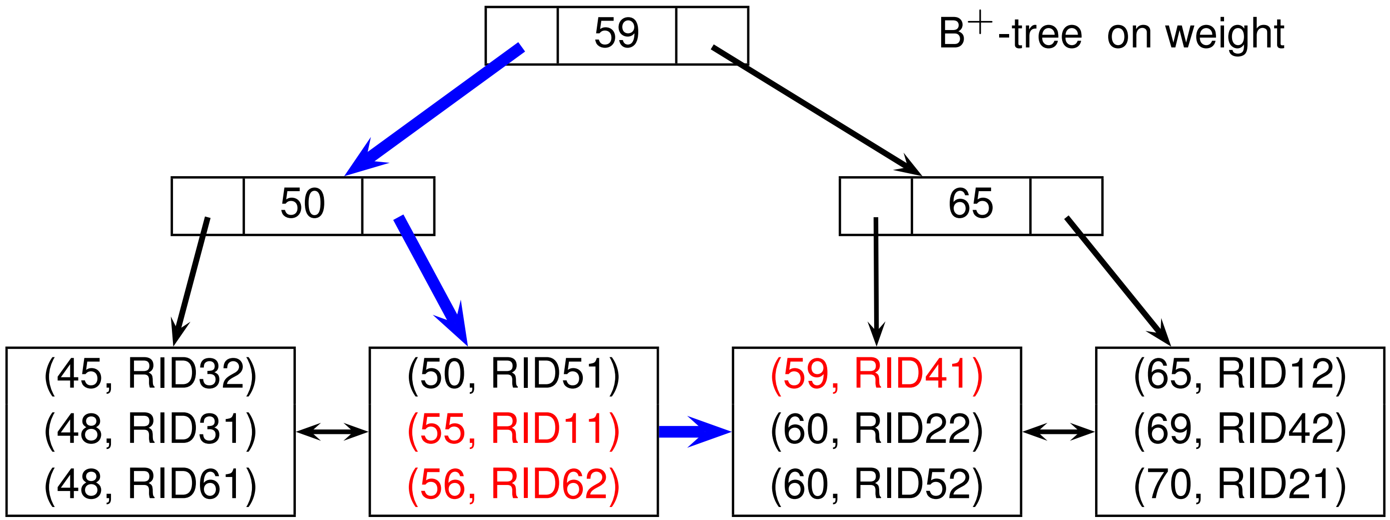
name	age	weight	height
Curly	10	65	171
Larry	12	70	175
Charlie	20	69	173

Access Path

- Access path refers to a way of accessing data records/entries
  - Table scan = scan all data pages
  - Index scan = scan index pages
  - Index intersection = combine results from multiple index scans (e.g., intersect, union)
- Index scan/intersection can be followed by RID lookups to retrieve data records

Selectivity of Access Path

- Selectivity of an access path = number of index & data pages retrieved to access data records/entries
- The most selective access path (ie with smallest selectivity) retrieves the fewest pages



Q_1: **select** weight **from** R **where** weight **between** 51 **and** 59

Q_2: **select** * **from** R **where** weight **between** 51 **and** 59

Covering Index

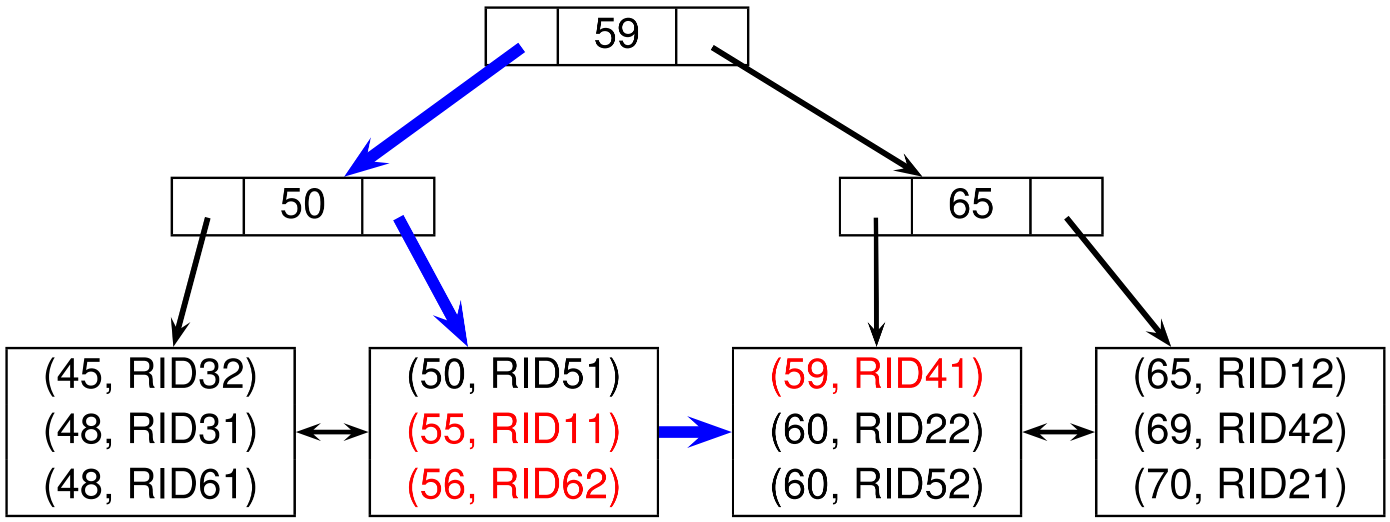
- An index *I* is a covering index for a query *Q* if all the attributes referenced in *Q* are part of the key of *I*
  - *Q* can be evaluated using *I* without any RID lookup
  - Such an evaluation plan is known as index-only plan
- **Example**:
  - Consider query *Q*:

    select name **from** Dept **where** deptno = 25

  - An index on (deptno, name) is a covering index for *Q*
  - An index on (deptno) is not a covering index for *Q*

B^+-tree : Index Scan

select weight
from Student
where weight **between** 55 **and** 59

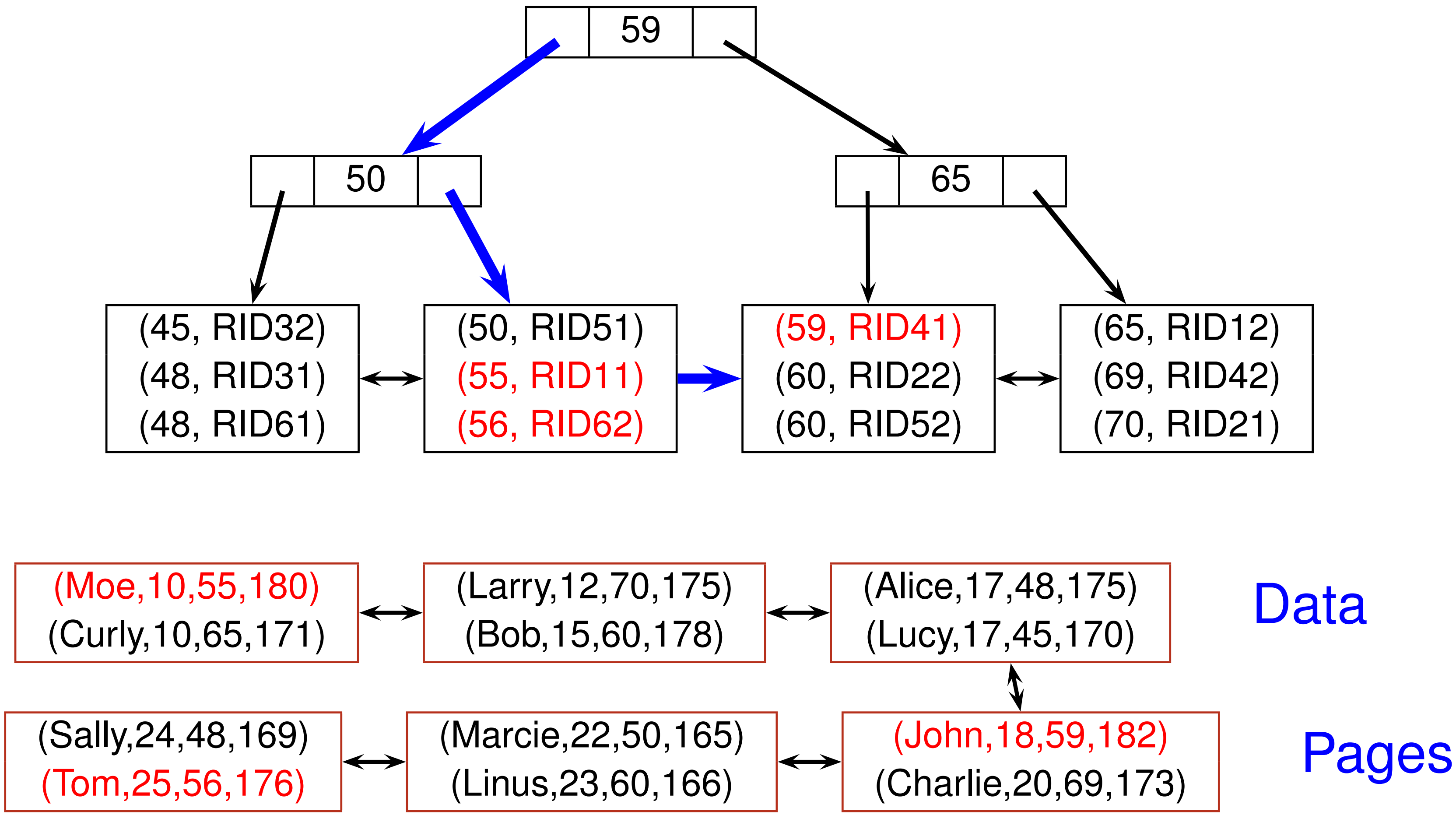


B^+-tree index on weight

B^+-tree : Index Scan + RID Lookups

select name
from Student
where weight **between** 55 **and** 59

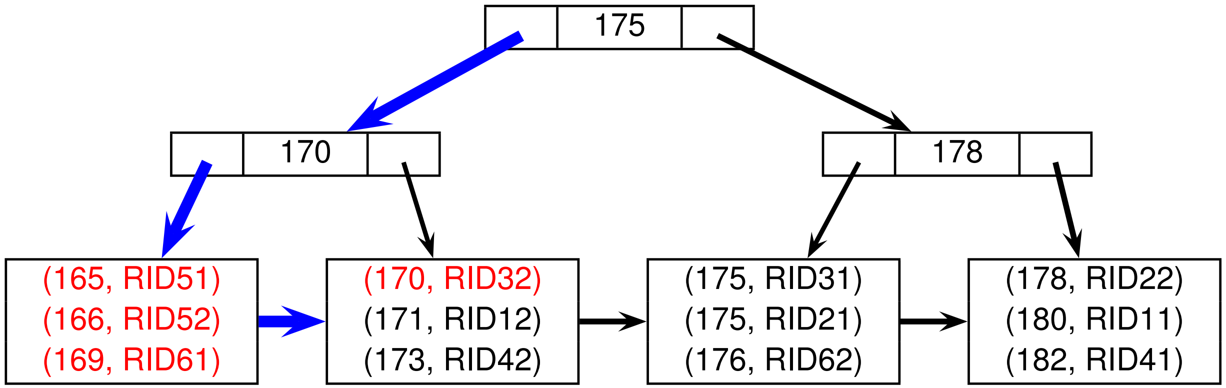
select weight
from Student
where weight **between** 55 **and** 59
and age $\geq$ 20



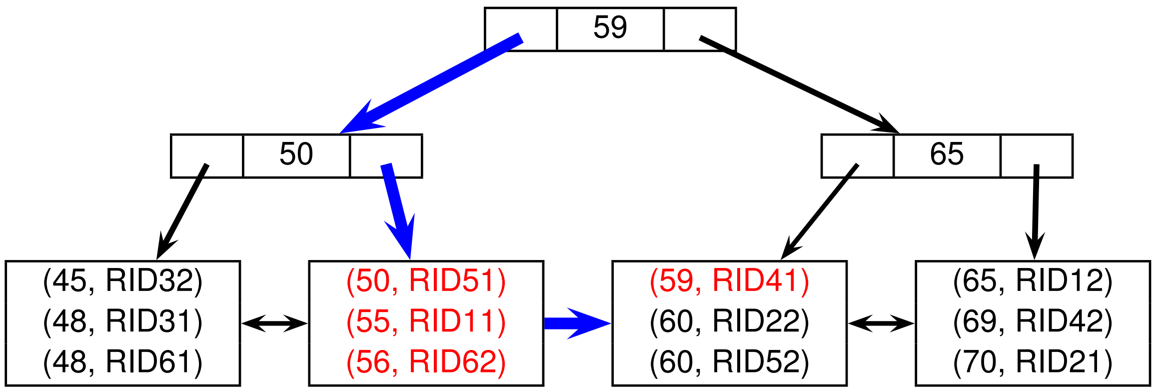
B^+-tree : Index Intersection

select height, weight **from** Student
where height **between** 164 **and** 170
and weight **between** 50 **and** 59

I_1 = (height)



I_2 = (weight)



CNF Predicates

- A term is of the form $R.A$ op c or $R.A_i$ op $R.A_j$
- A conjunct consists of one or more terms connected by $\vee$
- A conjunct that contains $\vee$ is said to be disjunctive (or contains a disjunction)
- A conjunctive normal form (CNF) predicate consists of one or more conjuncts connected by $\wedge$

$$\overbrace{(\underbrace{\text{rating} \geq 8}_{\text{term/conjunct}} \vee \underbrace{\text{director} = \text{“Coen”}}_{\text{term/conjunct}})}^{\text{disjunctive conjunct}} \wedge \underbrace{(\text{year} > 2003)}_{\text{term/conjunct}} \wedge \underbrace{(\text{language} = \text{“English”})}_{\text{term/conjunct}}$$

B$^+$-tree : Matching predicates

- B$^+$-tree index $I = (K_1, K_2, \cdots, K_n)$
- Non-disjunctive CNF predicate p
- I matches p if p is of the form:

$$\underbrace{(K_1 = c_1) \wedge \cdots \wedge (K_{i-1} = c_{i-1})}_{\text{zero or more equality predicates}} \wedge (K_i \ op_i \ c_i), \ i \in [1, n]$$

where

1. $(K_1, \cdots, K_i)$ is a prefix of the key of I, and
2. there is at most one non-equality comparison operator which must be on the last attribute of the prefix (i.e., K_i)

- Note: this definition is stronger than R&G's definition

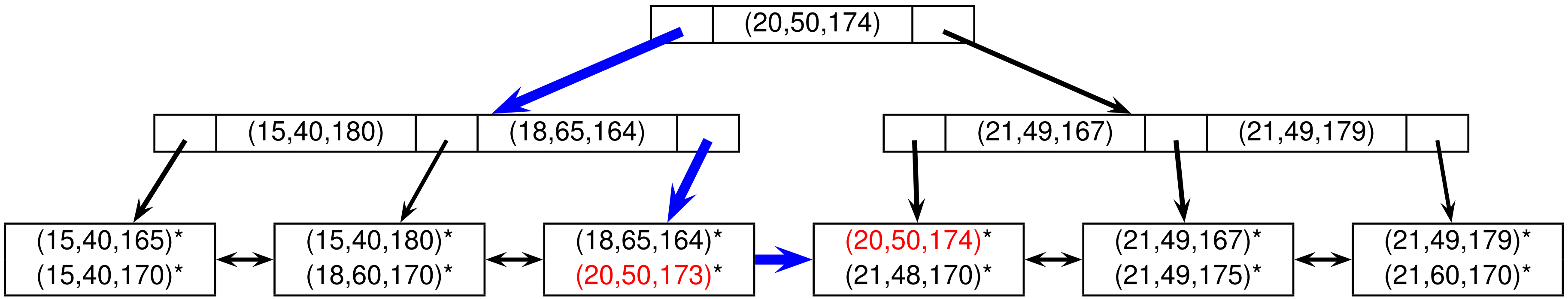
B^+-tree : Matching predicates (cont.)

Example: Which predicates does $I = (age, weight, height)$ match?

1. age $\geq$ 20
2. weight = 80
3. (age = 20) $\wedge$ (weight = 70)
4. (age = 20) $\wedge$ (weight $<$ 70)
5. (age $>$ 20) $\wedge$ (weight $=$ 70)
6. (age $=$ 20) $\wedge$ (height $=$ 170)
7. (height $>$ 180) $\wedge$ (weight $=$ 65) $\wedge$ (age = 20)
8. (age = 20) $\wedge$ (weight $\leq$ 65) $\wedge$ (height $=$ 180)

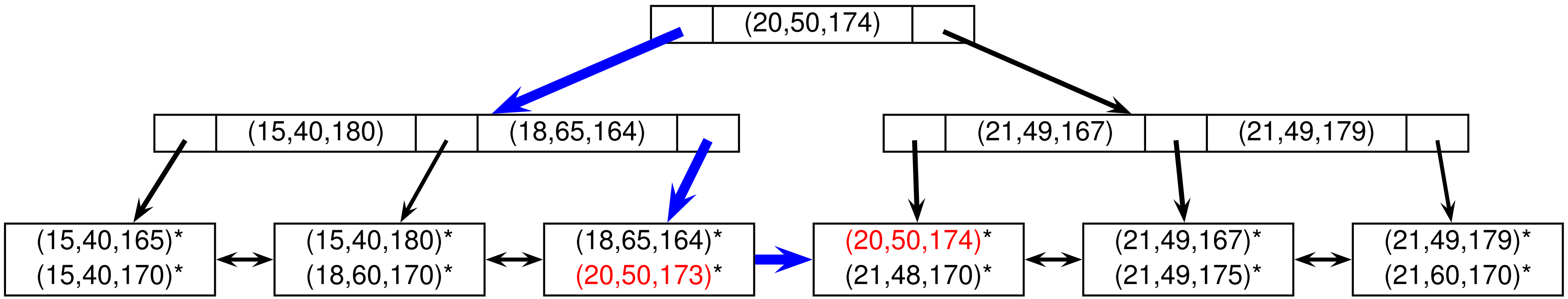
Example: B^+-tree on (age,weight,height)

p_1: (age = 20)

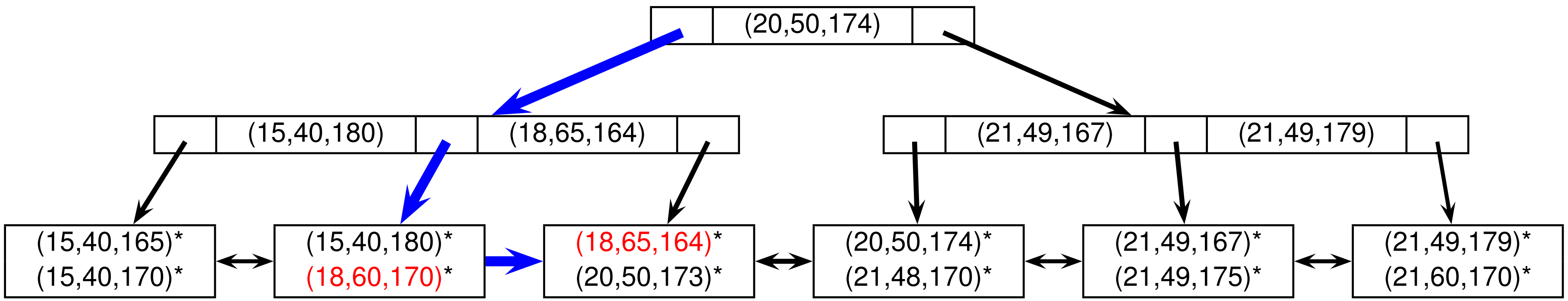


Example: B^+-tree on (age,weight,height)

p_1: (age = 20)

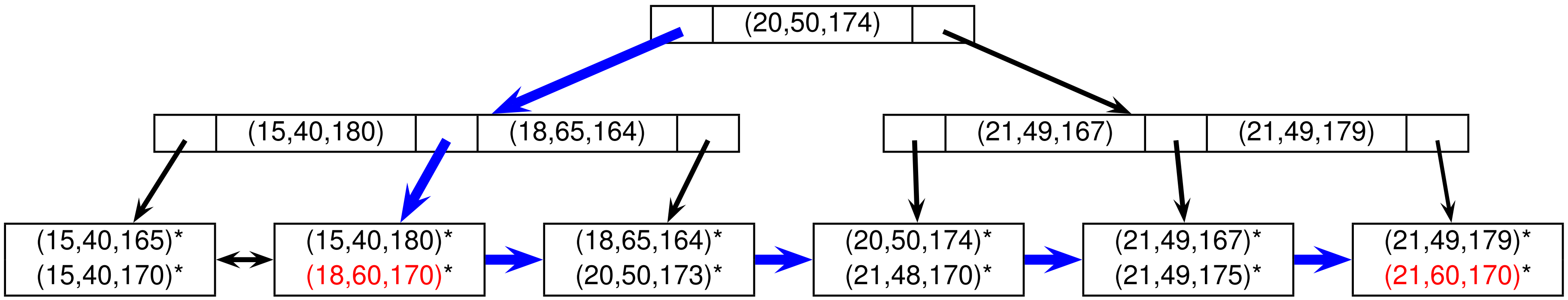


p_2: (age = 18) $\wedge$ (weight $\geq$ 50)



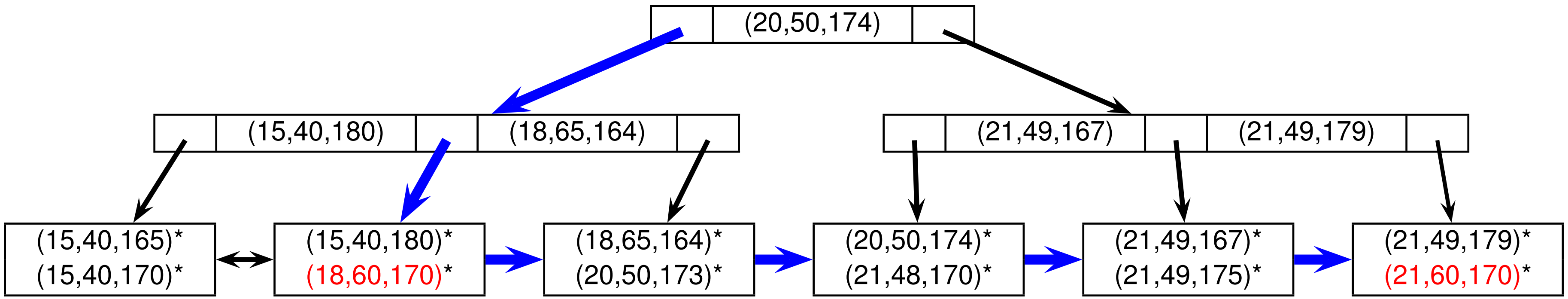
Example: B$^+$-tree on (age,weight,height)

p_3: (age $\geq$ 18) $\wedge$ (weight $=$ 60)

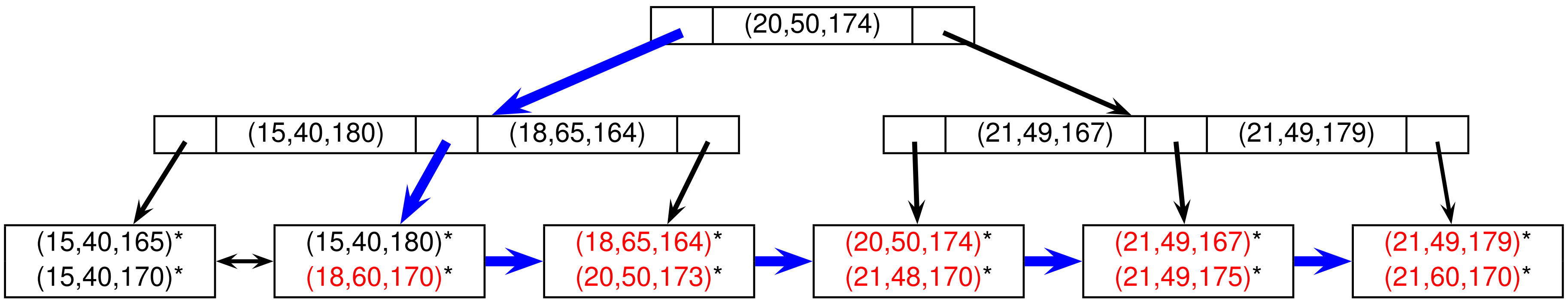


Example: B^+-tree on (age,weight,height)

p_3: (age $\geq$ 18) $\wedge$ (weight $=$ 60)



p_4: (age $\geq$ 18)



Hash Index: Matching predicates

- Hash index $I = (K_1, K_2, \cdots, K_n)$
- Non-disjunctive CNF predicate p
- I matches p if p is of the form:

$$(K_1 = c_1) \wedge (K_2 = c_2) \wedge \cdots \wedge (K_n = c_n)$$

Hash Index: Matching predicates (cont.)

Example: Which predicates does $I = (age, weight, height)$ match?

1. age $\geq$ 20
2. weight = 80
3. (age = 20) $\wedge$ (weight = 70)
4. (age = 20) $\wedge$ (weight $<$ 70)
5. (age $>$ 20) $\wedge$ (weight $=$ 70)
6. (age $=$ 20) $\wedge$ (height $=$ 170)
7. (height $=$ 180) $\wedge$ (weight $=$ 65) $\wedge$ (age = 20)
8. (age = 20) $\wedge$ (weight $\leq$ 65) $\wedge$ (height $=$ 180)

Primary Conjuncts

- In general, an index *I* matches only a subset of the conjuncts in a selection predicate *p*
- The subset of conjuncts in *p* that I matches are called primary conjuncts

Example 1:

- B^+-tree index *I* = (age,weight,height)
- Predicate p = (age $\geq$ 18) $\wedge$ (age $\leq$ 20) $\wedge$ (weight = 65) $\wedge$ (level = 3)
- Primary conjuncts: (age $\geq$ 18) $\wedge$ (age $\leq$ 20)
- Non-primary conjuncts: (weight = 65) $\wedge$ (level = 3)

Primary Conjuncts (cont.)

Example 2:

- Hash index I = (age,level)
- Predicate p = (age $= 22$) $\wedge$ (height $= 170$) $\wedge$ (level = 3)
- Primary conjuncts: (age $= 22$) $\wedge$ (level $= 3$)
- Non-primary conjunct: height = 170

Covered Conjuncts

- The subset of conjuncts in p that are covered by I are called covered conjuncts
- Each attribute in covered conjuncts appears in the key of I
- Primary conjuncts $\subseteq$ Covered conjuncts

Example 1:

- B^{+}-tree index I = (age,weight,height)
- Predicate p = (age $\geq$ 18) $\wedge$ (age $\leq$ 20) $\wedge$ (height = 180) $\wedge$ (level = 3)
- Covered conjuncts: (age $\geq$ 18) $\wedge$ (age $\leq$ 20) $\wedge$ (height = 180)

Notation

Notation	Meaning
r	relational algebra expression
$\|\|r\|\|$	number of tuples in output of r
$\|r\|$	number of pages in output of r
b_d	number of data records that can fit on a page
b_i	number of data entries that can fit on a page
F	average fanout of B^+-tree index (i.e., number of pointers to child nodes)
h	height of B^+-tree index (i.e., number of levels of internal nodes)
	$h = \lceil \log_F(\lceil \frac{\|\|R\|\|}{b_i} \rceil) \rceil$ if format-2 index on table R
B	number of available buffer pages

Cost of B$^+$-tree index evaluation of p

Let p' = primary conjuncts of p, p_c = covered conjuncts of p

1. Navigate internal nodes to locate first leaf page

$$Cost_{internal} = \begin{cases} \lceil \log_F(\lceil \frac{||R||}{b_d} \rceil) \rceil & \text{if } I \text{ is a format-1 index,} \\ \lceil \log_F(\lceil \frac{||R||}{b_i} \rceil) \rceil & \text{otherwise.} \end{cases}$$

2. Scan leaf pages to access all qualifying data entries

$$Cost_{leaf} = \begin{cases} \lceil \frac{||\sigma_{p'}(R)||}{b_d} \rceil & \text{if } I \text{ is a format-1 index,} \\ \lceil \frac{||\sigma_{p'}(R)||}{b_i} \rceil & \text{otherwise.} \end{cases}$$

Cost of B$^+$-tree index evaluation of p (cont.)

3. Retrieve qualified data records via RID lookups

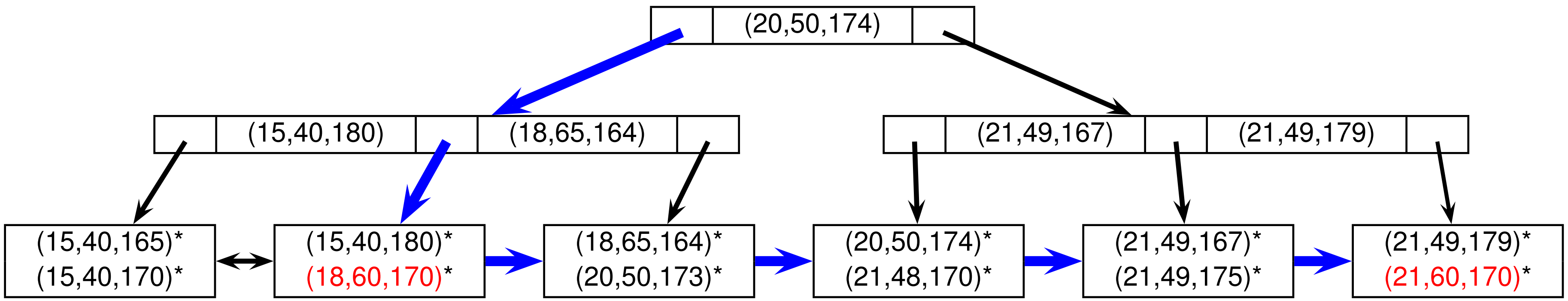
$$Cost_{rid} = \begin{cases} 0 & \text{if I is covering or format-1 index,} \\ ||\sigma_{p_c}(R)|| & \text{otherwise.} \end{cases}$$

Cost of RID lookups could be reduced by first sorting the RIDs

$$\lceil \frac{||\sigma_{p_c}(R)||}{b_d} \rceil \leq \text{ Cost}_{rid} \leq \min\{||\sigma_{p_c}(R)||,\ |R|\}$$

Example

- B^+-tree index I = (age,weight,height), Format 2
- Query: **select** * **from** R **where** p
  - p: (age $\geq$ 18) $\wedge$ (weight = 60) $\wedge$ (level = 3)
  - p' = (age $\geq$ 18)
  - p_c = (age $\geq$ 18) $\wedge$ (weight = 60)
- $||R|| = 12, \ ||\sigma_{p'}(R)|| = 9, \ ||\sigma_{p_c}(R)|| = 2$
- $b_d = b_i = 2$, Height of I = 2
- Evaluation cost of p using I = 2 + $\lceil \frac{9}{2} \rceil$ + 2 = 9



Cost of hash index evaluation of p

- Let p' = primary conjuncts of p
- For format-1 index
  - Cost to retrieve data records: at least $\lceil \frac{||\sigma_{p'}(R)||}{b_d} \rceil$
- For format-2 index
  - Cost to retrieve data entries: at least $\lceil \frac{||\sigma_{p'}(R)||}{b_i} \rceil$
  - $$\begin{array}{l}\text{Cost to retrieve} \\ \text{data records}\end{array} = \begin{cases} 0 & \text{if I is a covering index,} \\ ||\sigma_{p'}(R)|| & \text{otherwise.} \end{cases}$$

Evaluating Non-Disjunctive Conjuncts

- Consider the query $\sigma_p(R)$, where

  $$p = (\text{age} = 21) \wedge (\text{weight} \geq 70) \wedge (\text{height} = 180)$$

- Suppose the available unclustered indexes on R are
  - a hash index H_{age} on (age), and
  - a B$^+$-tree index T_{weight} on (weight)
- What are the possible strategies to evaluate p?

Evaluating Disjunctive Conjuncts

- Suppose the available unclustered indexes are
  - a hash index H_{age} on (age), and
  - a B^+-tree index T_{weight} on (weight)
- What are the possible strategies to evaluate the following predicates?
- $p_1 = (\text{age} = 21) \vee (\text{height} = 180)$
- $p_2 = ((\text{age} = 21) \vee (\text{height} = 180)) \wedge (\text{weight} \geq 70)$
- $p_3 = (\text{age} = 21) \vee (\text{weight} \geq 70)$