Методы и системы обработки
больших данных

9. Реализация исполнителя запросов

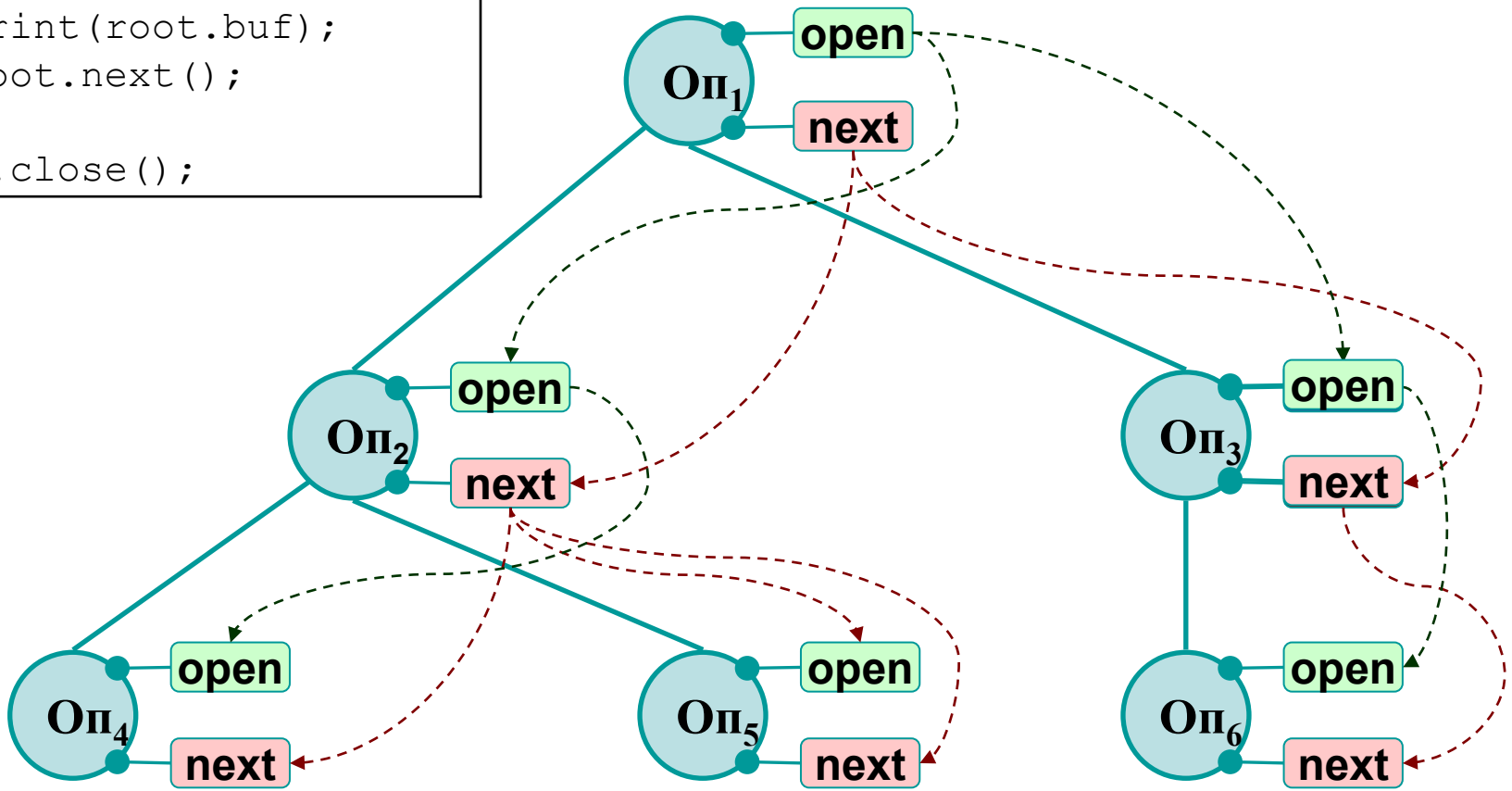
Итераторы

Физическая операция реализуется в виде итератора, представляющего собой группу из трех методов *open*, *next*, *close*, которые позволяют потребителю результата физической операции получать *покортежно* (по одному кортежу за один раз).

1. ***open***() – Выделяет память для внутренних структур данных итератора; инициализирует структуры данных, необходимые для работы итератора и вызывает метод *open*() для дочерних узлов. При необходимости может вызывать методы *next* и *close* для дочерних узлов.
2. ***next***() – помещает очередной кортеж результата в выходной буфер своего узла; модифицирует внутренние структуры данных, подготавливая выдачу следующего кортежа; вызывает метод *next* (возможно многократно) для дочерних улов. При необходимости может вызывать методы *open* и *close* для дочерних узлов. Если множество кортежей исчерпано, помещает в выходной буфер специальный кортеж ***EOF***, который невозможно спутать ни с каким другим.
3. ***close***() – освобождает память; вызывает *close*() для дочерних узлов.

Итераторная модель

```
root.open();
root.next();
while(root.buf!=EOF){
   print(root.buf);
   root.next();
};
root.close();
```



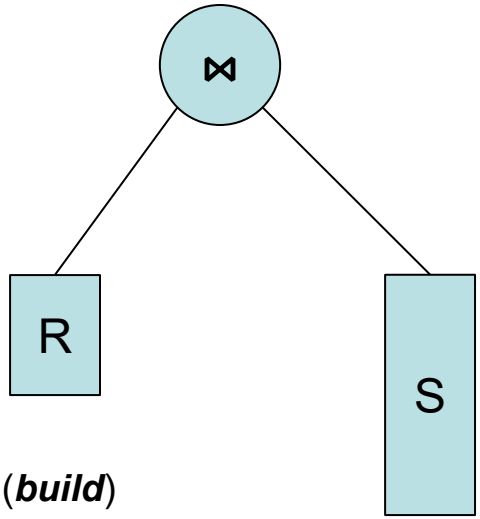
Реализация итератора для операции *scan*, считывающей кортежи отношения R

```
void open(R){
   b = первый блок отношения R;
   t = первый кортеж блока b;
};

void next(R){
   if(t == NULL){
      b = следующий блок отношения R;
      if (b == NULL){
         node.buf = EOF;
         return;
      };
      t = первый кортеж блока b;
   };
   node.buf = t;
   t = следующий кортеж блока b;
   return;
};

void close(R){};
```

Методы и системы обработки
больших данных

Алгоритмы соединений

Порядок операндов соединения



Левое отношение:
меньшее ⇔ ***опорное*** (***build***)

Правое отношение:
большее ⇔ ***тестируемое*** (***probe***)

NLMJ (Nested Loops Memory Join)

Реализация соединения вложенными циклами в оперативной памяти

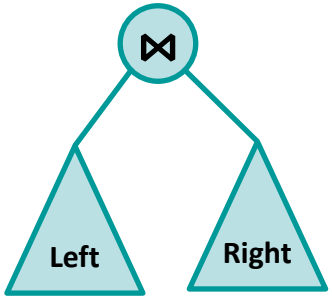
- $R \underset{(A)}{\bowtie} S$
- R – опорное отношение целиком помещается в оперативную память
- S – тестируемое отношение
- Структура внутренних данных:
  - M – массив в оперативной памяти (для R)
  - m – указатель на текущий элемент в M

Схема алгоритма NLMJ

1. Загрузить в память опорное (меньшее) отношение
2. Во внешнем цикле однократно сканируется тестируемое (большее) отношение
   - Во внутреннем цикле многократно сканируется образ опорного отношения в оперативной памяти

for $r \in R$ **do** $M \leftarrow r$; // Поместить кортеж в массив
for $s \in S$ **do**
    for $r \in M$ **do**
        if $r.A = s.A$ **then**
            print $(r \bowtie s)$; // Соединить кортежи

NLMJ: схема реализации метода open



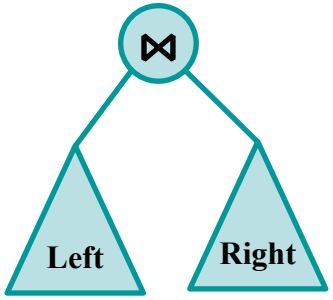

```
void open(node){
    выделить память для M;
    node.leftSon.open();
    m = 0; /* Номер текущего элемента в массиве M */
    do{/* Заполняем массив M кортежами опорного (левого) отношения */
      node.leftSon.next();
      M[m] = node.leftSon.buf;
      m++;
    } while (node.leftSon.buf!=EOF);
    m = 0;
    node.rightSon.open();
    node.rightSon.next();
};
```

NLMJ: схема реализации метода next

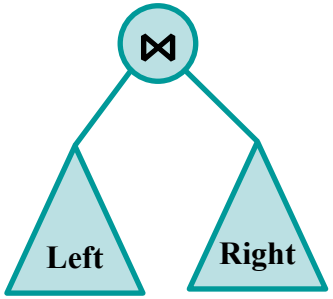
```
void next(node){
    while(node.rightSon.buf!=EOF){// Сканируем тестируемое (правое) отношение
        while(M[m]!=EOF){  // Сканируем опорное (левое) отношение
            if(MATCH_A(M[m],node.rightSon.buf)){
                node.buf=JOIN_A(M[m],node.rightSon.buf);
                m++;
                return;
            };
            m++;
        };
        m=0;
        node.rightSon.next();
    };
    node.buf=EOF;
};
```

Проверяет на совпадение значений атрибута A у двух кортежей

Выполняет естественное соединение двух кортежей по атрибуту A



NLMJ: схема реализации метода close




```
void close(node){
    освободить память для M;
    node.leftSon.close();
    node.rightSon.close();
};
```

Оценка эффективности алгоритма

- Считаем только дисковые операции
- Более эффективным является тот алгоритм, который выполняет меньшее число дисковых операций

Оценка эффективности *NLMJ*

$R(A,B) \bowtie S(C,A)$ $\mathbf{T}(R)=150$ $\mathbf{T}(S)=3000$

Атрибут	Тип	Семантика	Длина (байт)
A	int	Целое	16
B	int	Целое	16
C	string	Строка	176

Объект	Длина (байт)
Заголовок записи	8
Длина блока	3024
Заголовок блока	24

	R	***S***
Длина записи	2·16 + 8 = **40**	16 + 176 + 8 = 200
Записей в блоке	(3024-24)/40 = **75**	(3024-24)/200 = 15
Количество блоков	150/75 = **2**	3000/15 = **200**

Число блоков ⇨

Количество чтений с диска:	**B**(R) + **B**(S) = 2+200 = **202**

NLDJ (Nested Loops Disk Join)

Соединения вложенными циклами на диске

- Во внешнем цикле однократно сканируется тестируемое (большее) отношение
- Во внутреннем цикле многократно сканируется опорное (меньшее) отношение
- Алгоритм применим к отношениям любых размеров
- Один из самых неэффективных алгоритмов

Реализация NLDJ

$$R(A, B) \bowtie S(C, A)$$

- R – опорное (меньшее) отношение
- S – тестируемое (большее) отношение

FOR $s \in S$ **DO**
  FOR $r \in R$ **DO**
    IF $r.A = s.A$ **THEN** output $(r \bowtie s)$;

- Тестируемое отношение S считывается с диска один раз во внешнем цикле
- Опорное отношение R считывается с диска $T(S)$ раз во внутреннем цикле

NLDJ: схема реализации метода open

```
void* open(node){
    node.leftSon.open();
    node.rightSon.open();
    node.leftSon.next();
    node.rightSon.next();
};
```

NLDJ: схема реализации метода next

```
void next(node){
   while(node.rightSon.buf!=EOF){// Сканируем тестируемое (правое) отношение
      while(node.leftSon.buf!=EOF){// Сканируем опорное (левое) отношение
         if(MATCH_A(node.leftSon.buf,node.rightSon.buf)){
            node.buf=JOIN_A(node.leftSon.buf,node.rightSon.buf);
            node.leftSon.next();
            return;
         };
         node.leftSon.next();
      };
      node.leftSon.close();
      node.leftSon.open();
      node.leftSon.next();
      node.rightSon.next();
   };
   node.buf=EOF;
};
```

Проверяет на совпадение значений атрибута A у двух кортежей

Выполняет естественное соединение двух кортежей по атрибуту A

Оценка эффективности *NLDJ*

$R(A,B) \bowtie S(C,A) \quad T(R) = 150 \quad T(S) = 3000$

Атрибут	Тип	Семантика	Длина (байт)
A	int	Целое	16
B	int	Целое	16
C	string	Строка	176

Объект	Длина (байт)
Заголовок записи	8
Длина блока	3024
Заголовок блока	24

	R	***S***
Длина записи	2·16 + 8 = **40**	16 + 176 + 8 = **200**
Записей в блоке	(3024-24)/40 = **75**	(3024-24)/200 = **15**
Количество блоков	150/75 = **2**	3000/15 = **200**

Число блоков ⇨

	R сканируется во внутреннем цикле	*R* сканируется во внешнем цикле
Кол-во чтений с диска	**T**(*S*)·**B**(*R*)+**B**(*S*) = 3000·2+200 = **6 200**	**T**(*R*)·**B**(*S*)+**B**(*R*) = 150·200+2 = **30 002**

HJ (Hash Join)

Соединение хешированием

$$Q = R(A, B) \bowtie S(C, A)$$

Задается хеш-функция h: $D_A \rightarrow \{1, \dots, k\}$

- $R_i = \{r | r \in R \wedge h(r.A) = i\}$ – i-тый сегмент отношения R
- $S_i = \{s | s \in S \wedge h(s.A) = i\}$ – i-тый сегмент отношения S
- k выбирается так, что R_i целиком помещается в оперативную память $\forall i = 1, \dots, k$

I проход:

1. Разбить R на $R_1 \dots R_k$
2. Разбить S на $S_1 \dots S_k$

II проход:

1. **for** i **from** 1 **to** k **do** $Q_i = NLMJ(R_i, Si)$
2. $Q = \bigcup_{i=1}^{k} Q_i$

R

A*	B	C
1	20	100
2	40	300
3	20	100
4	10	300
5	10	200
6	30	100

S

D*	A#	E
1	3	0.2
2	1	0.5
3	1	0.5
4	1	0.6
5	6	0.4
6	3	0.9
7	5	0.1
8	3	0.1
9	5	0.5
10	1	0.7
11	2	0.4
12	2	0.3

$h(r.A) = (r.A \bmod 3) + 1$

$h(a) = (a \bmod 3) + 1$

$h(s.A) = (s.A \bmod 3) + 1$

R_1

A*	B	C
3	20	100
6	30	100

R_2

A*	B	C
1	20	100
4	10	300

R_3

A*	B	C
2	40	300
5	10	200

S_1

D*	A#	E
1	3	0.2
5	6	0.4
6	3	0.9
8	3	0.1

S_2

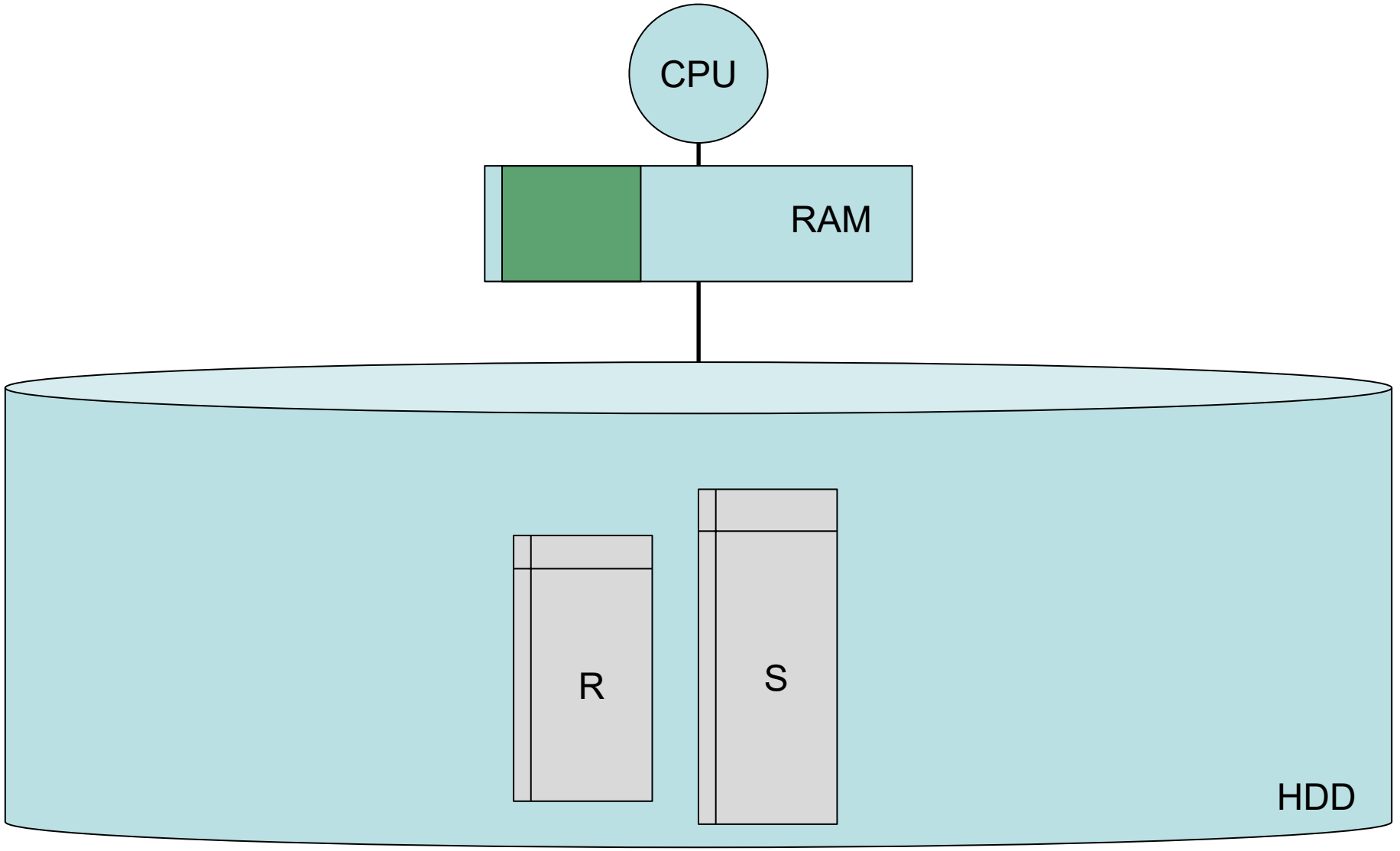
D*	A#	E
2	1	0.5
3	1	0.5
4	1	0.6
10	1	0.7

S_3

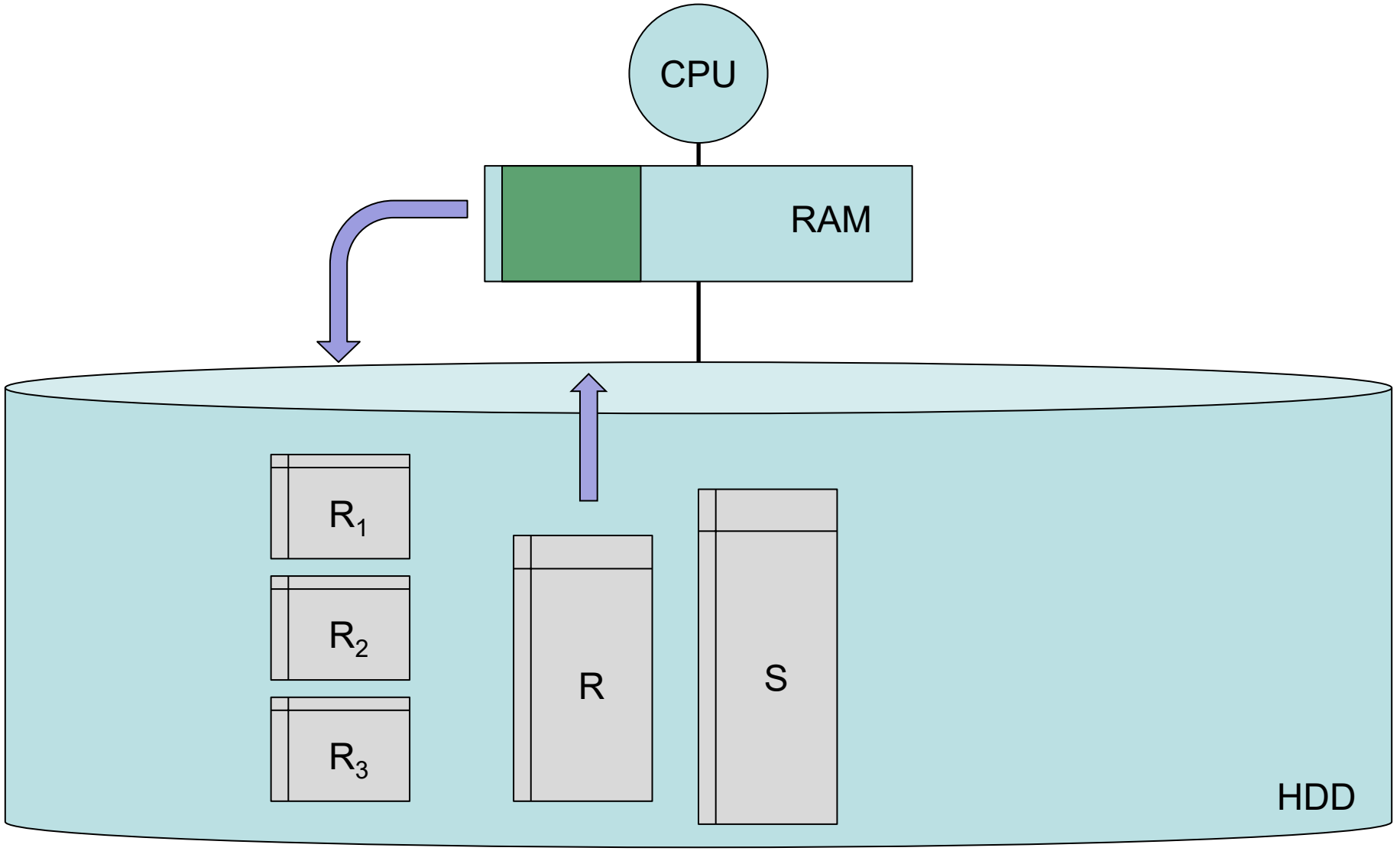
D*	A#	E
7	5	0.1
11	2	0.4
12	2	0.3
9	5	0.5

Сегменты

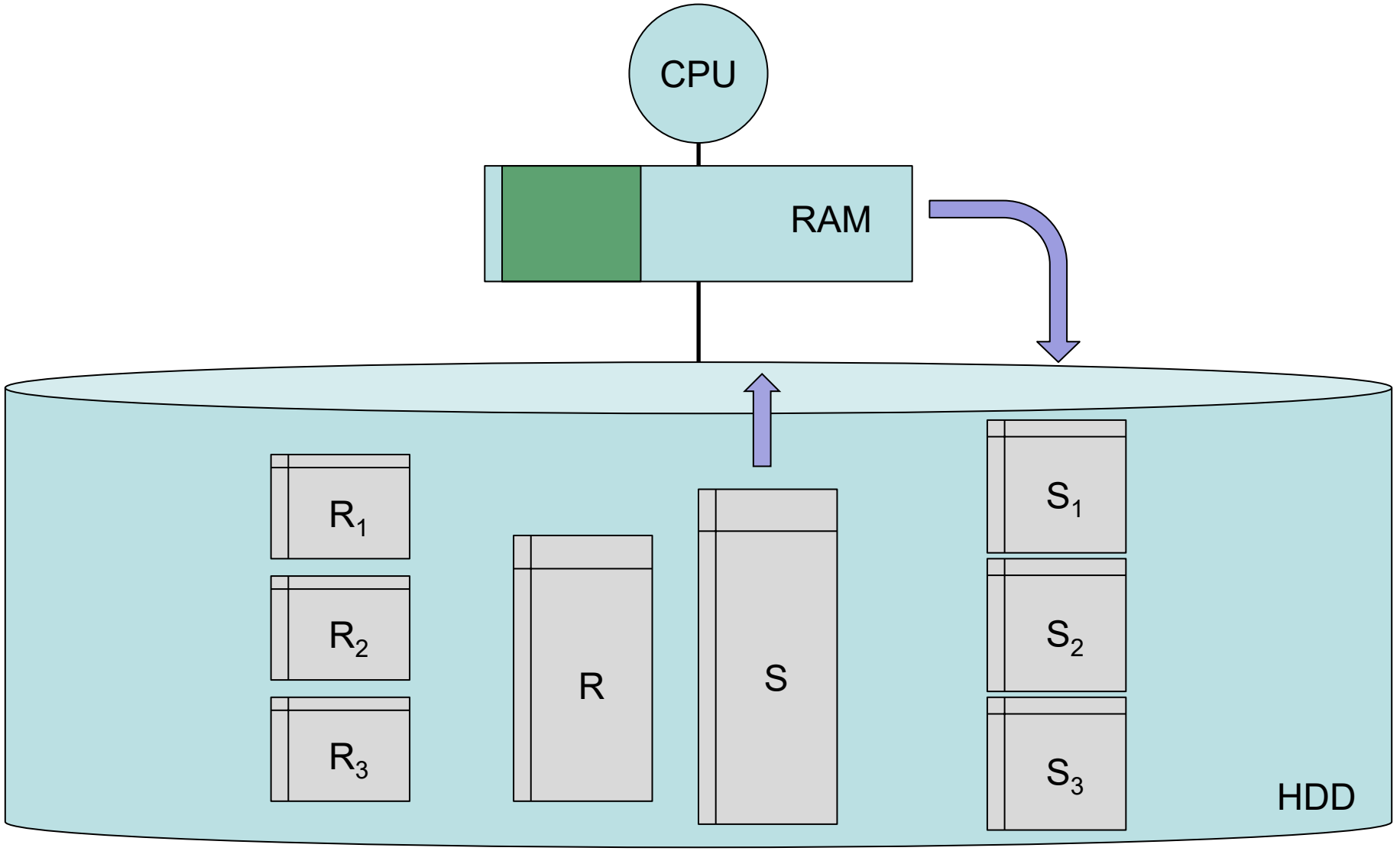
HJ. 1-й проход



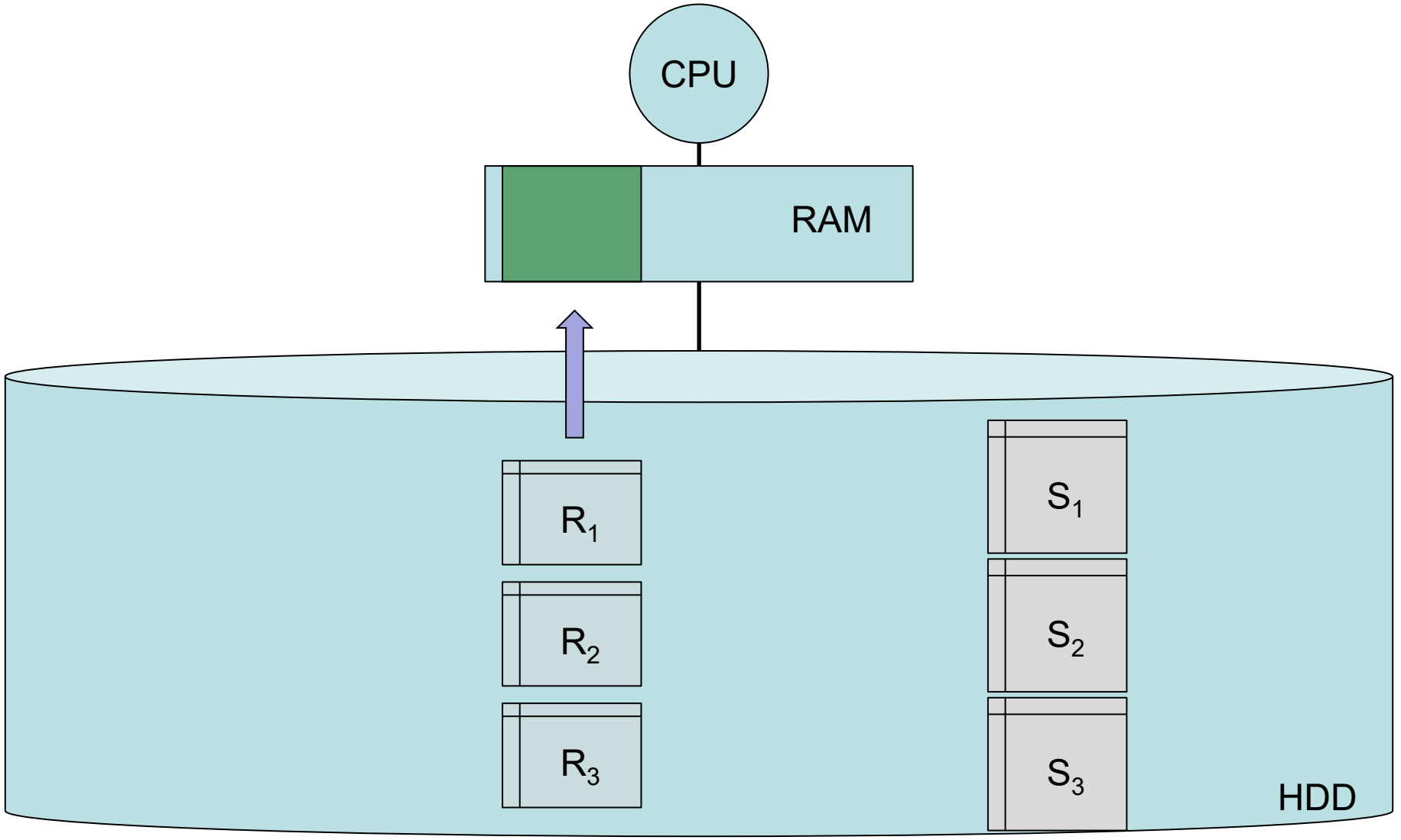
HJ. 1-й проход



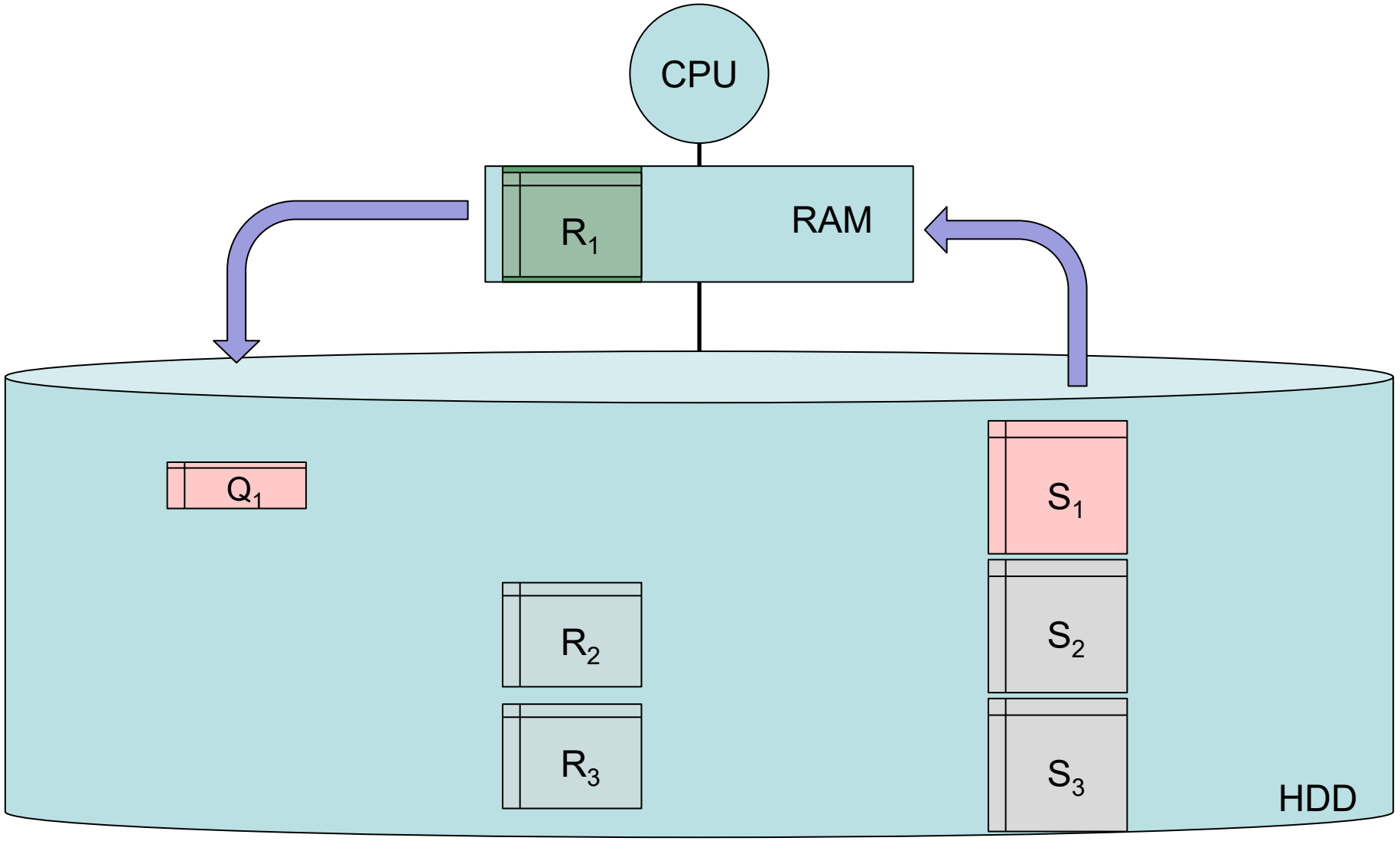
HJ. 1-й проход



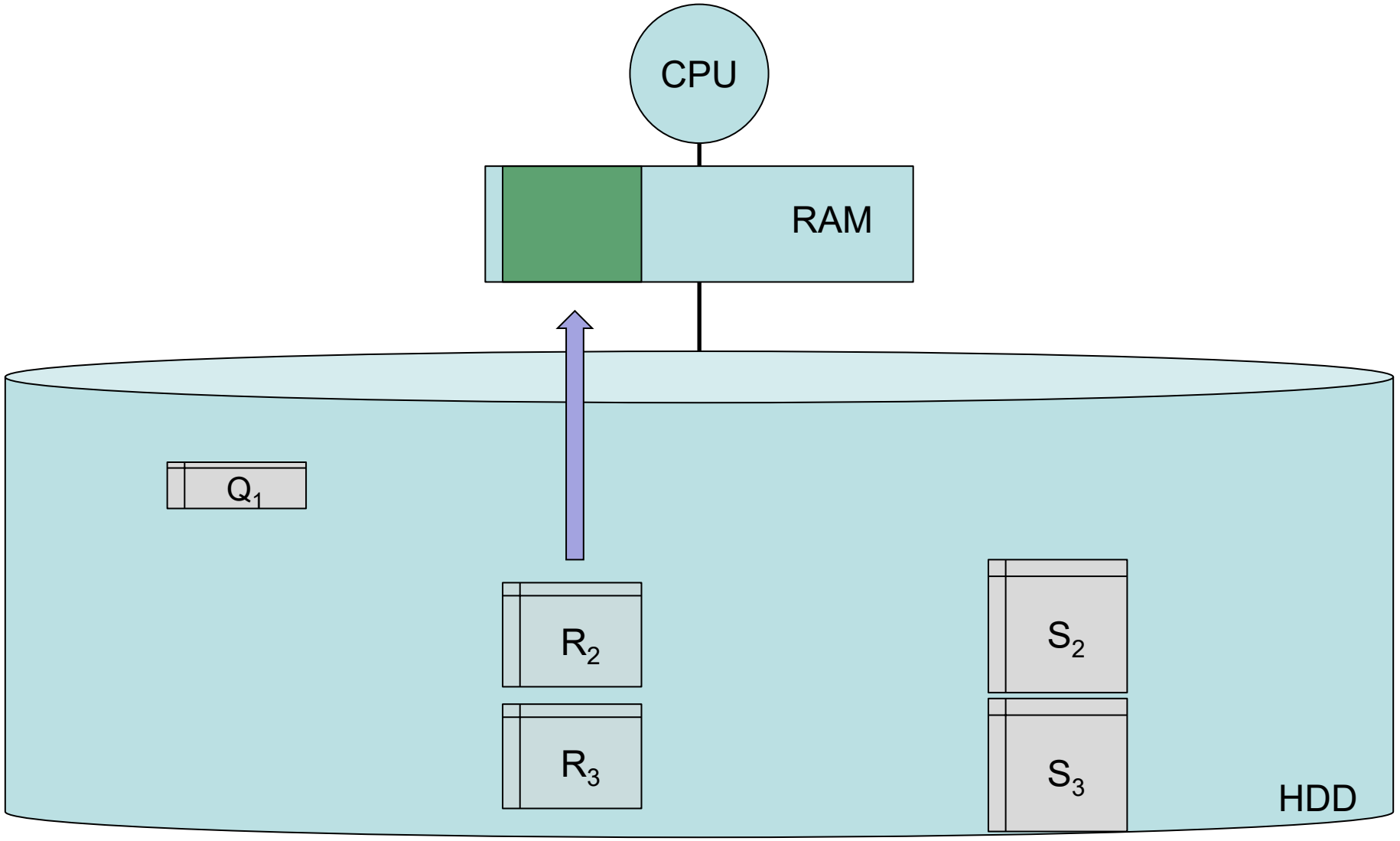
HJ. 2-й проход



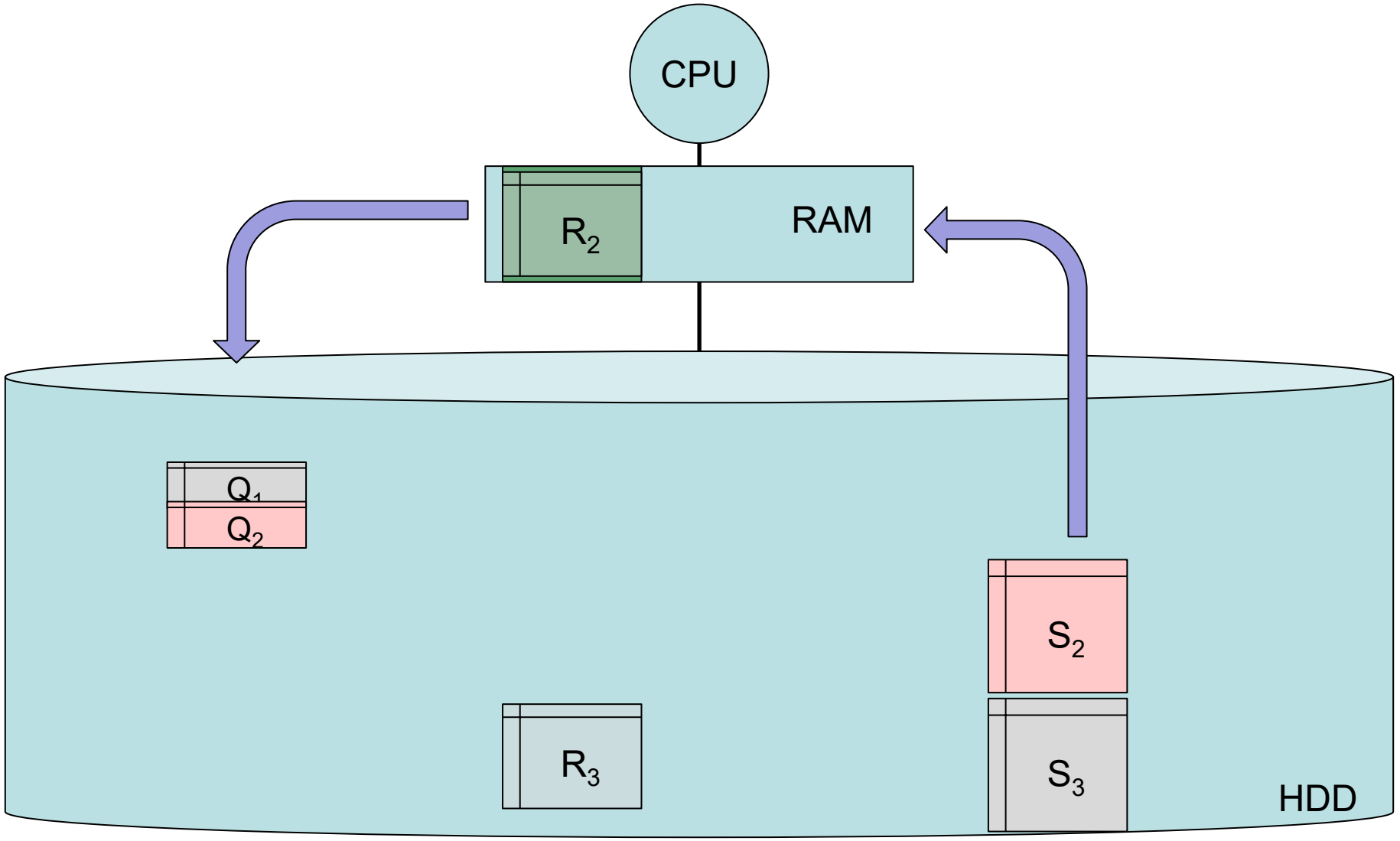
HJ. 2-й проход



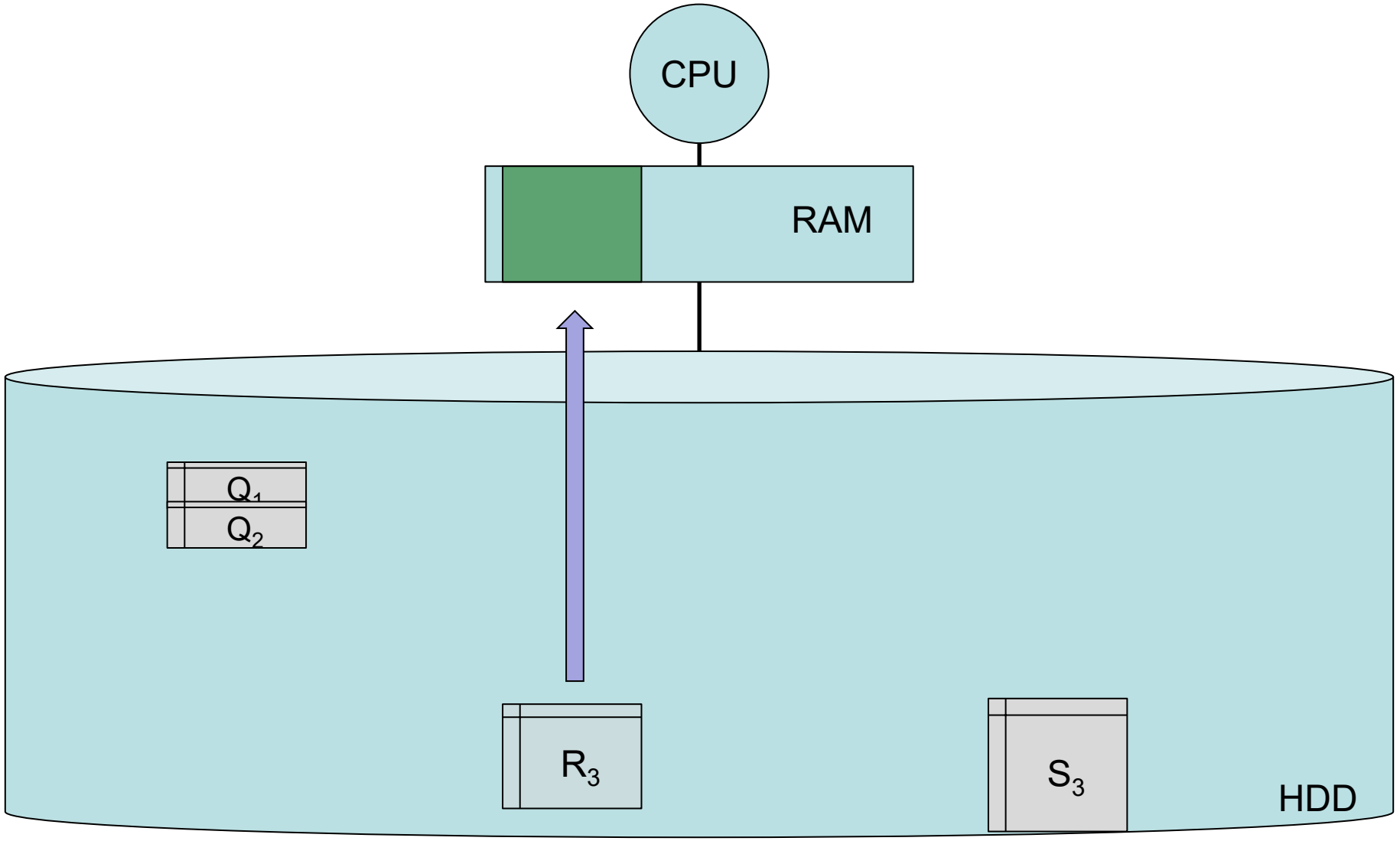
HJ. 2-й проход



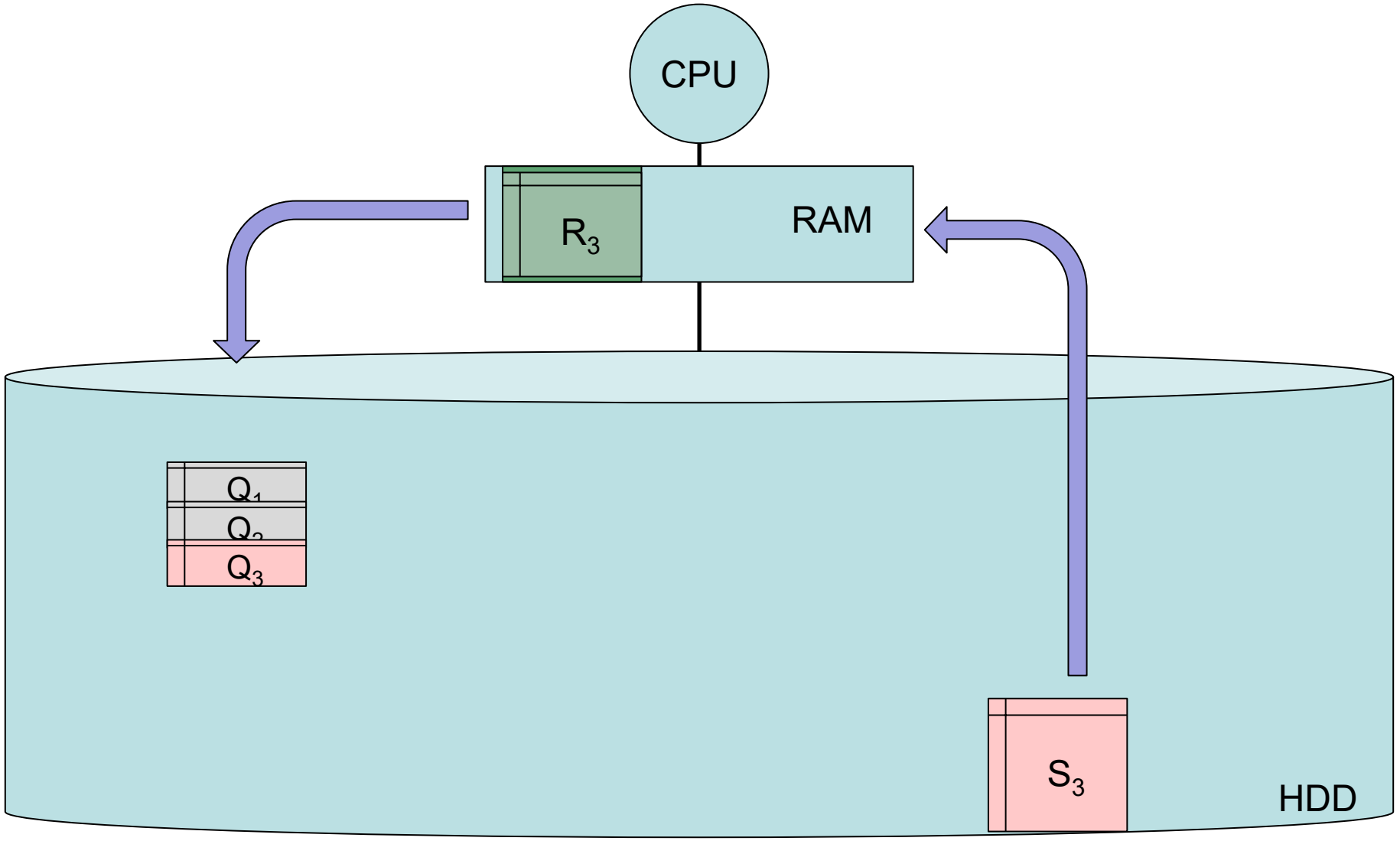
HJ. 2-й проход



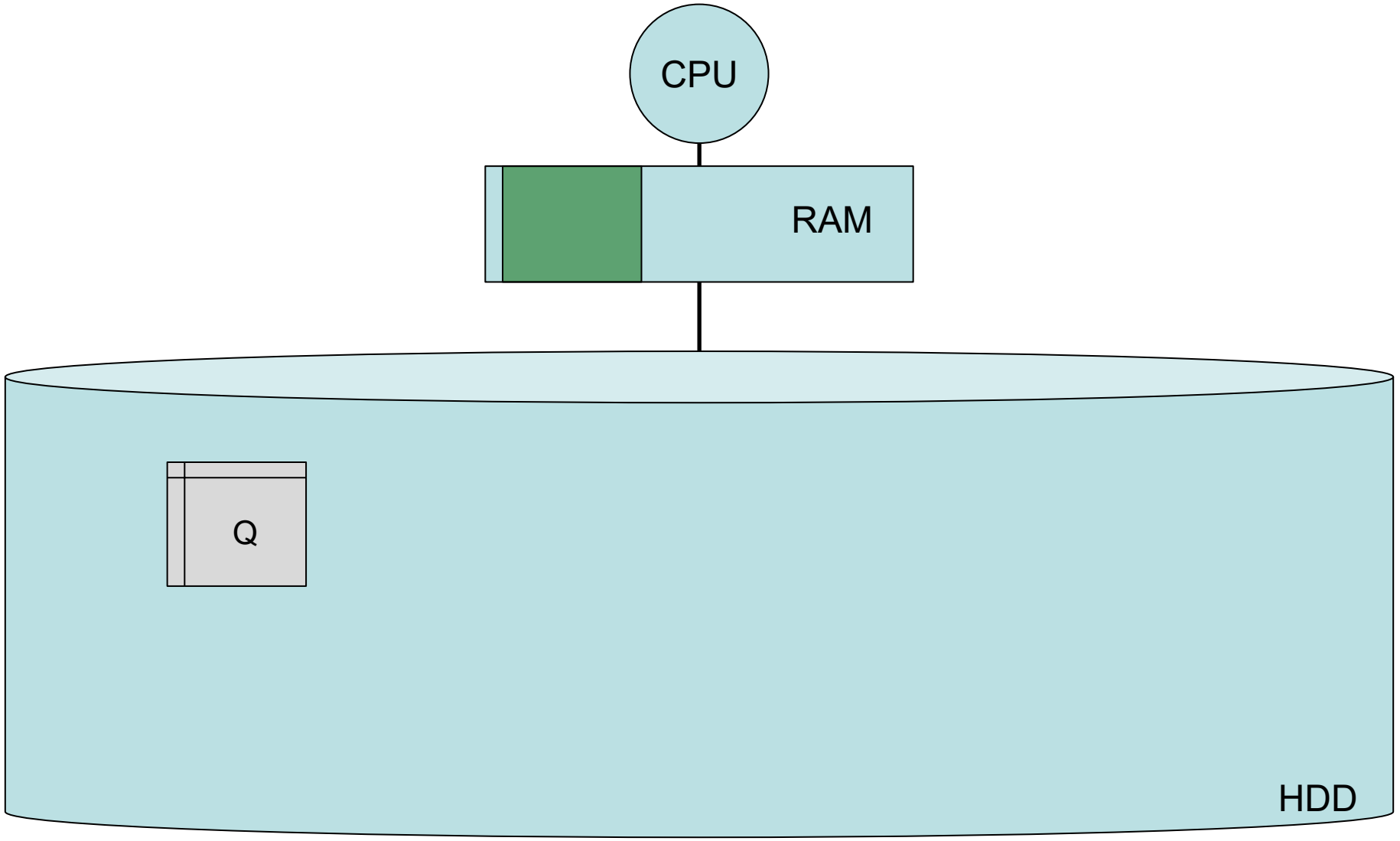
HJ. 2-й проход



HJ. 2-й проход



HJ. 2-й проход



Оценка эффективности *HJ*

$R(A,B) \bowtie S(C,A)$ $\mathbf{T}(R)=150$ $\mathbf{T}(S)=3000$

Атрибут	Тип	Семантика	Длина (байт)
A	int	Целое	16
B	int	Целое	16
C	string	Строка	176

B

Объект	Длина (байт)
Заголовок записи	8
Длина блока	3024
Заголовок блока	24

Число блоков ⇨

	R	***S***
Длина записи	$2 \cdot 16 + 8 =$ **40**	$16 + 176 + 8 =$ **200**
Записей в блоке	$(3024-24)/40 =$ **75**	$(3024-24)/200 =$ **15**
Количество блоков	$150/75 =$ **2**	$3000/15 =$ **200**

Количество обменов с диском:	$3(\mathbf{B}(R) + \mathbf{B}(S)) = 3(2+200) =$ **606**

HHJ (Hybrid Hash Join)

Гибридное соединение с хешированием

I проход:

for $r \in R$ **do**

Если $h(r.A) = 1$ поместить r в RAM

Если $h(r.A) = 2$ записать r в R_2

…

Если $h(r.A) = k$ записать r в R_k

for $s \in S$ **do**

Если $h(s.A) = 1$ найти в RAM все r такие, что $s.A = r.A$;

для каждой такой пары построить кортеж-соединение

Если $h(s.A) = 2$ записать r в S_2

…

Если $h(s.A) = k$ записать r в S_k

II проход:

for i **from** 2 **to** k **do**

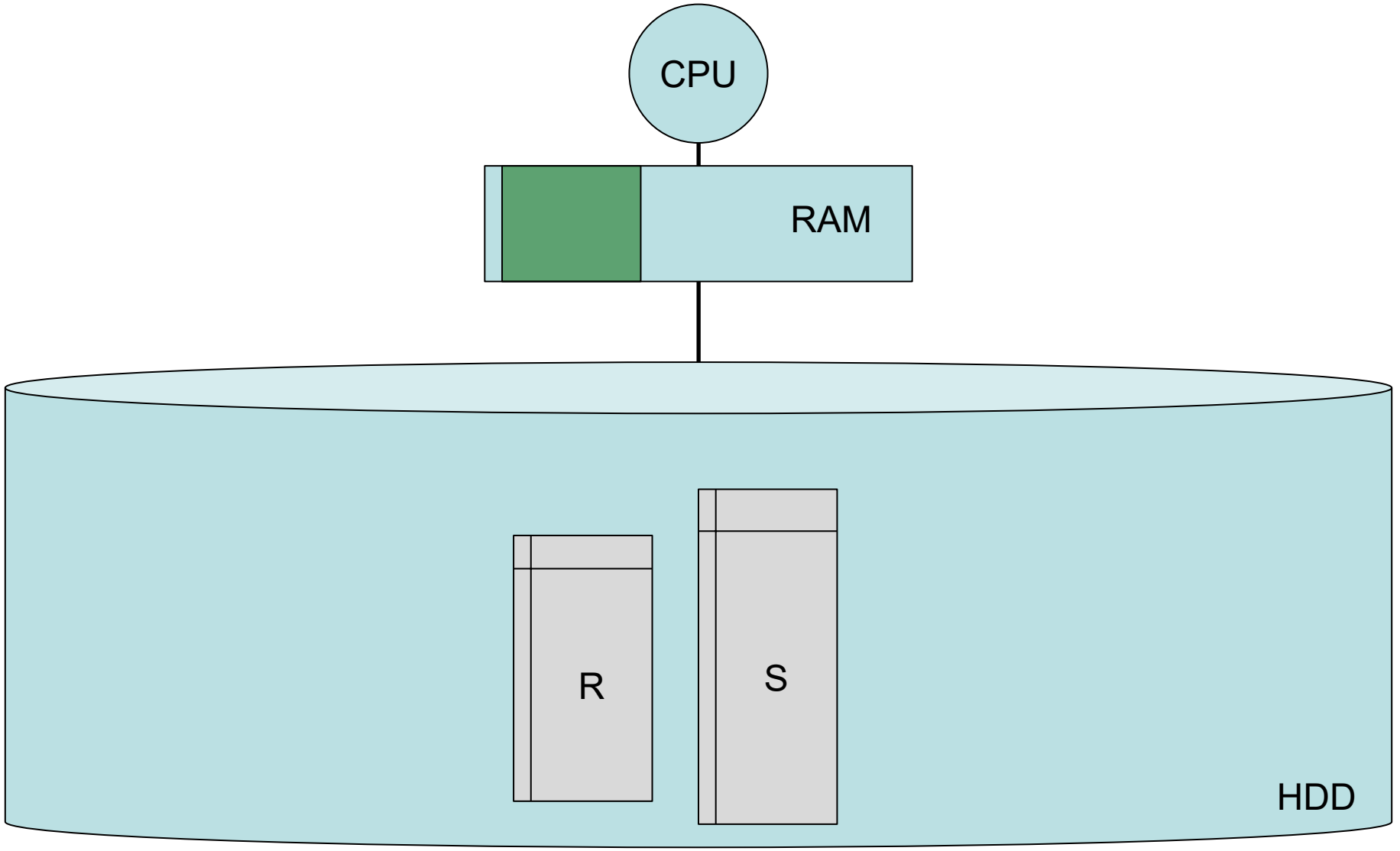
Загрузить R_i в RAM

for $s \in \boldsymbol{S_i}$ **do**

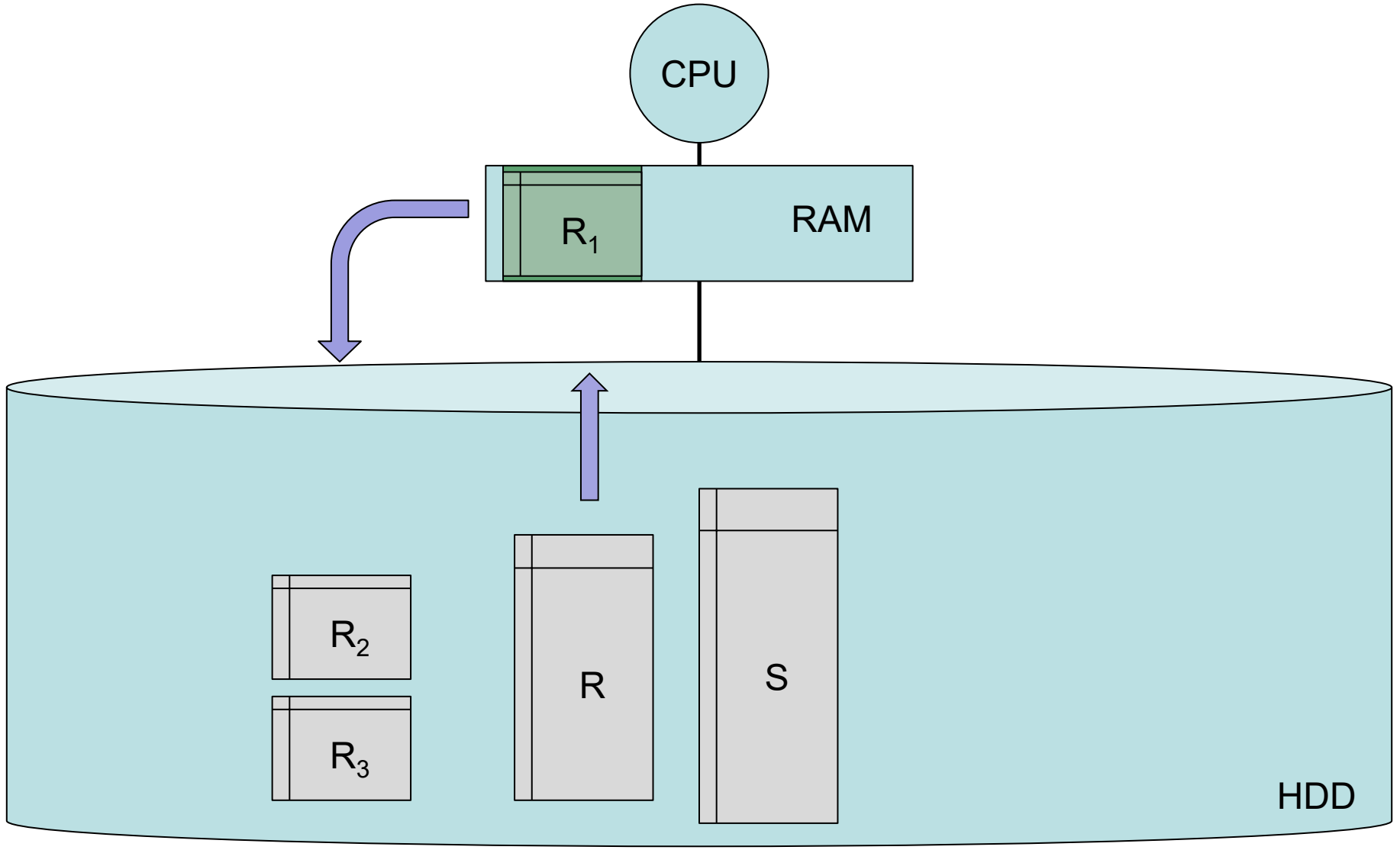
Найти в RAM все r такие, что $s.A = r.A$;

для каждой такой пары построить кортеж-соединение

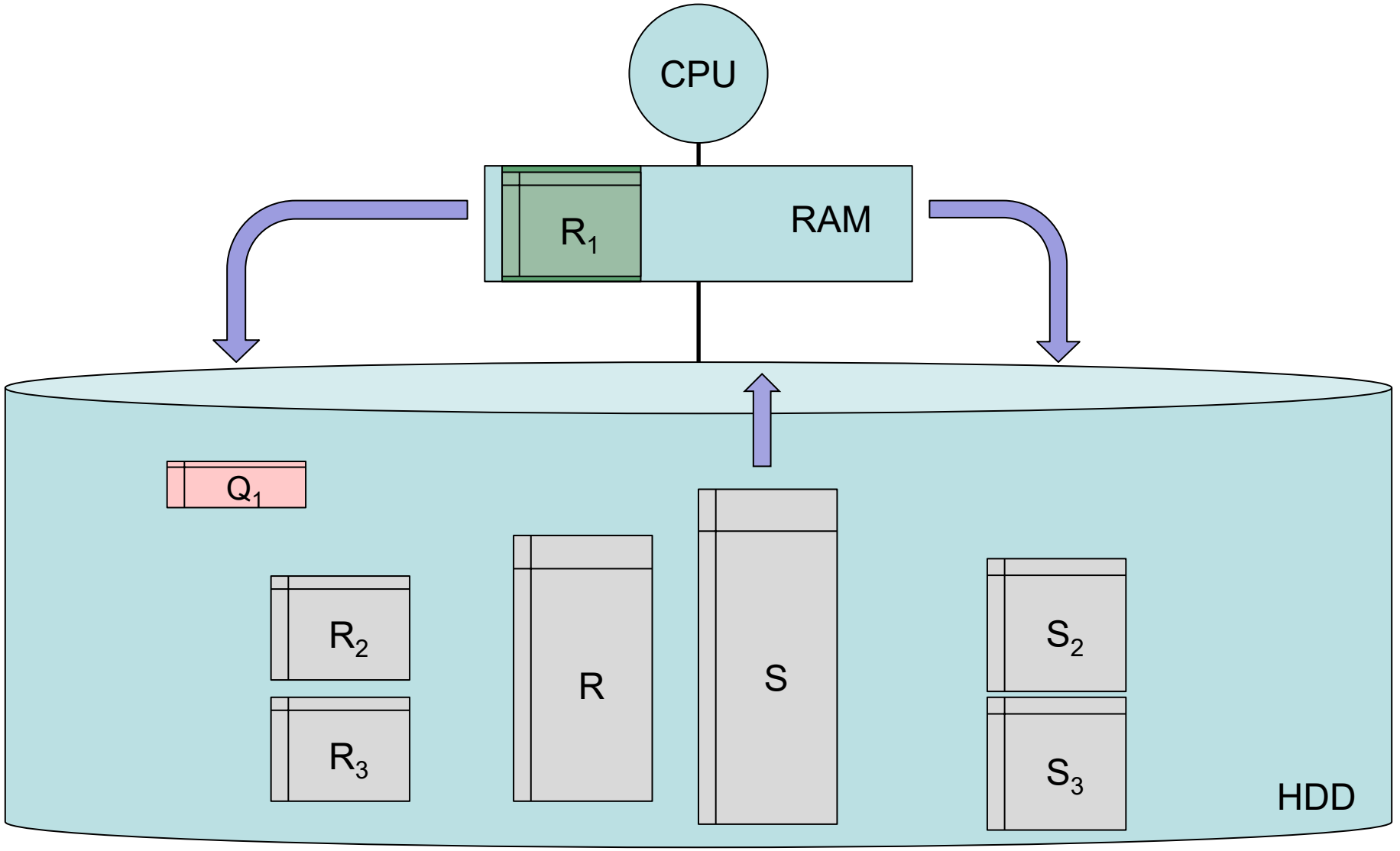
HJ. 1-й проход



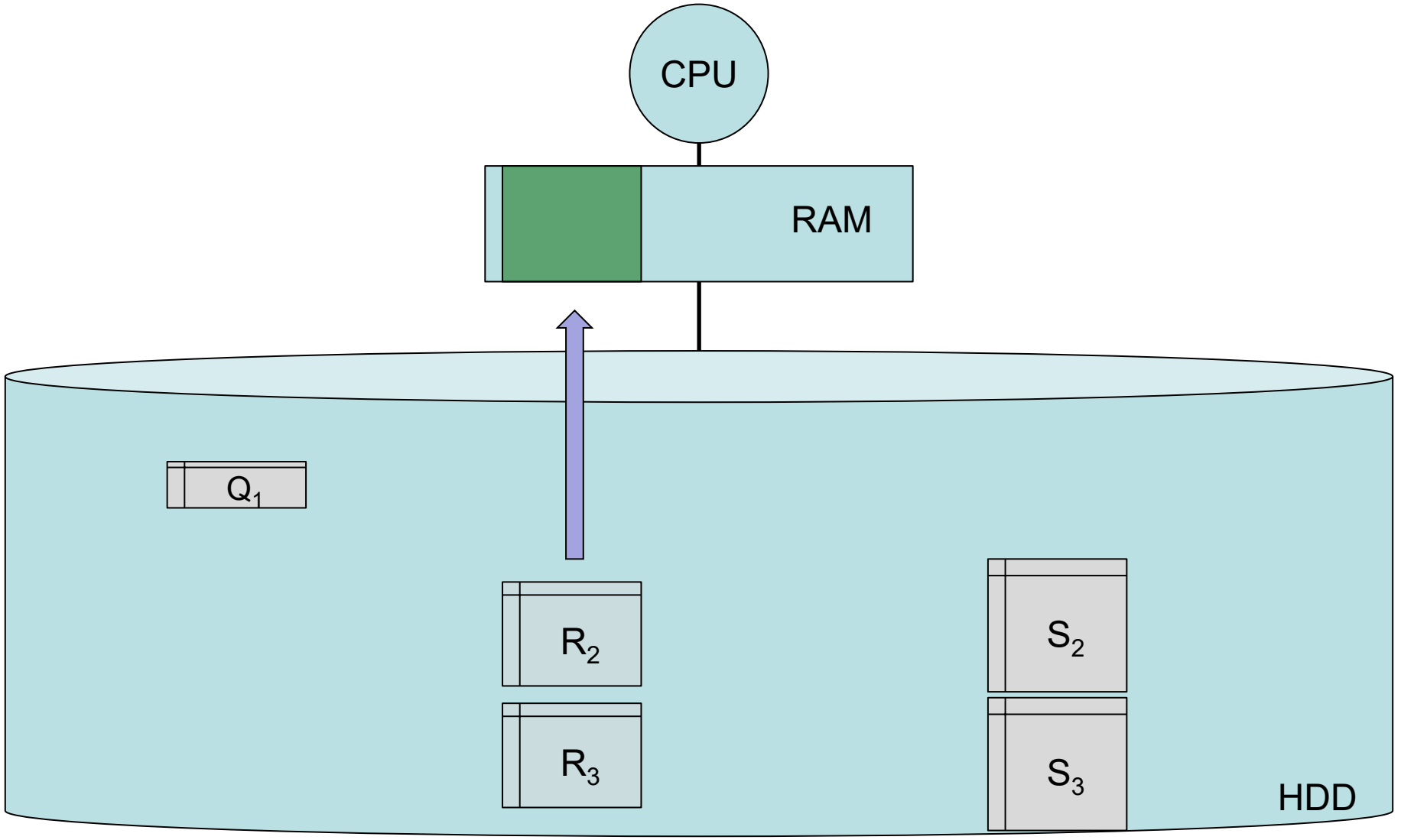
HJ. 1-й проход



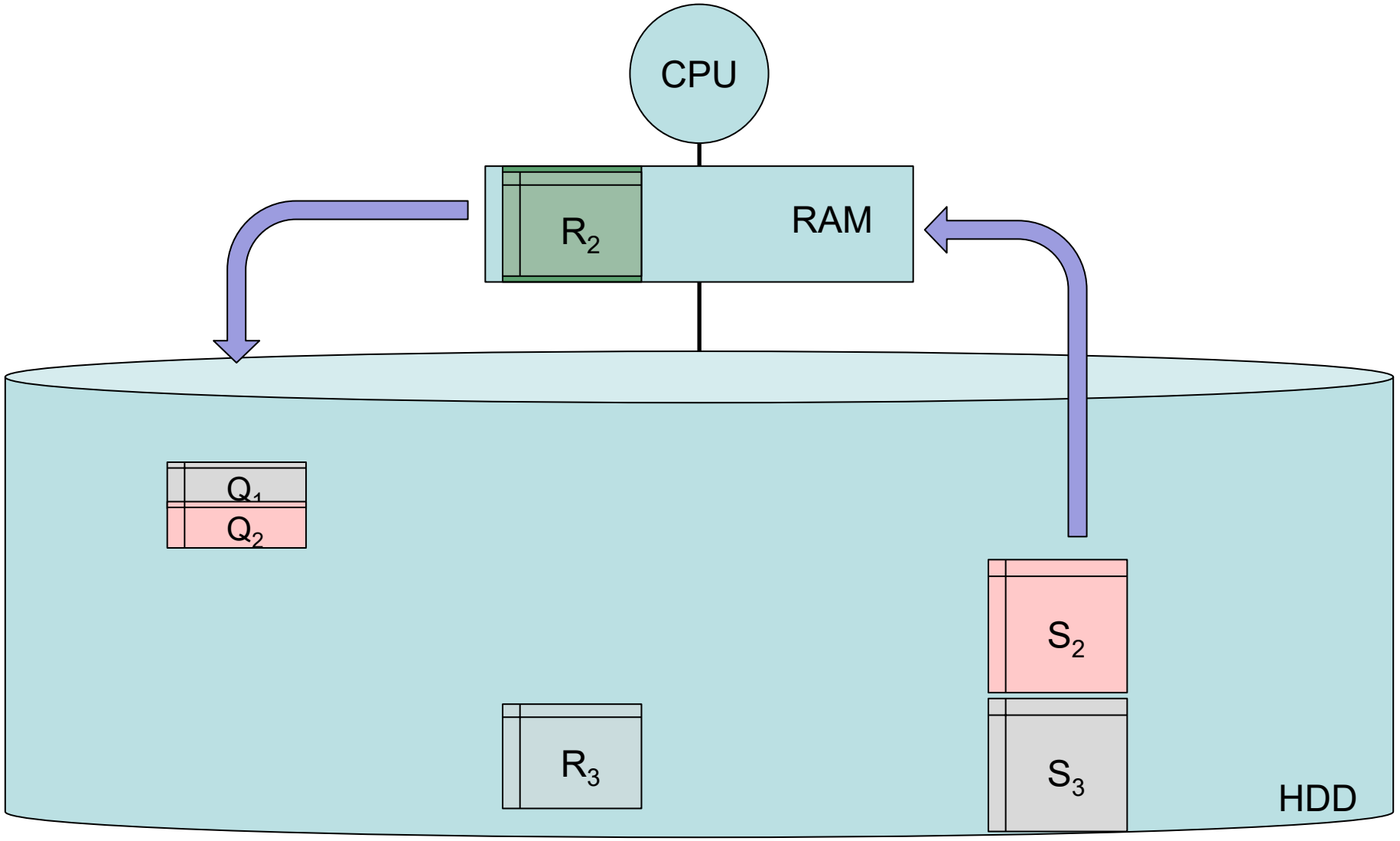
HJ. 1-й проход



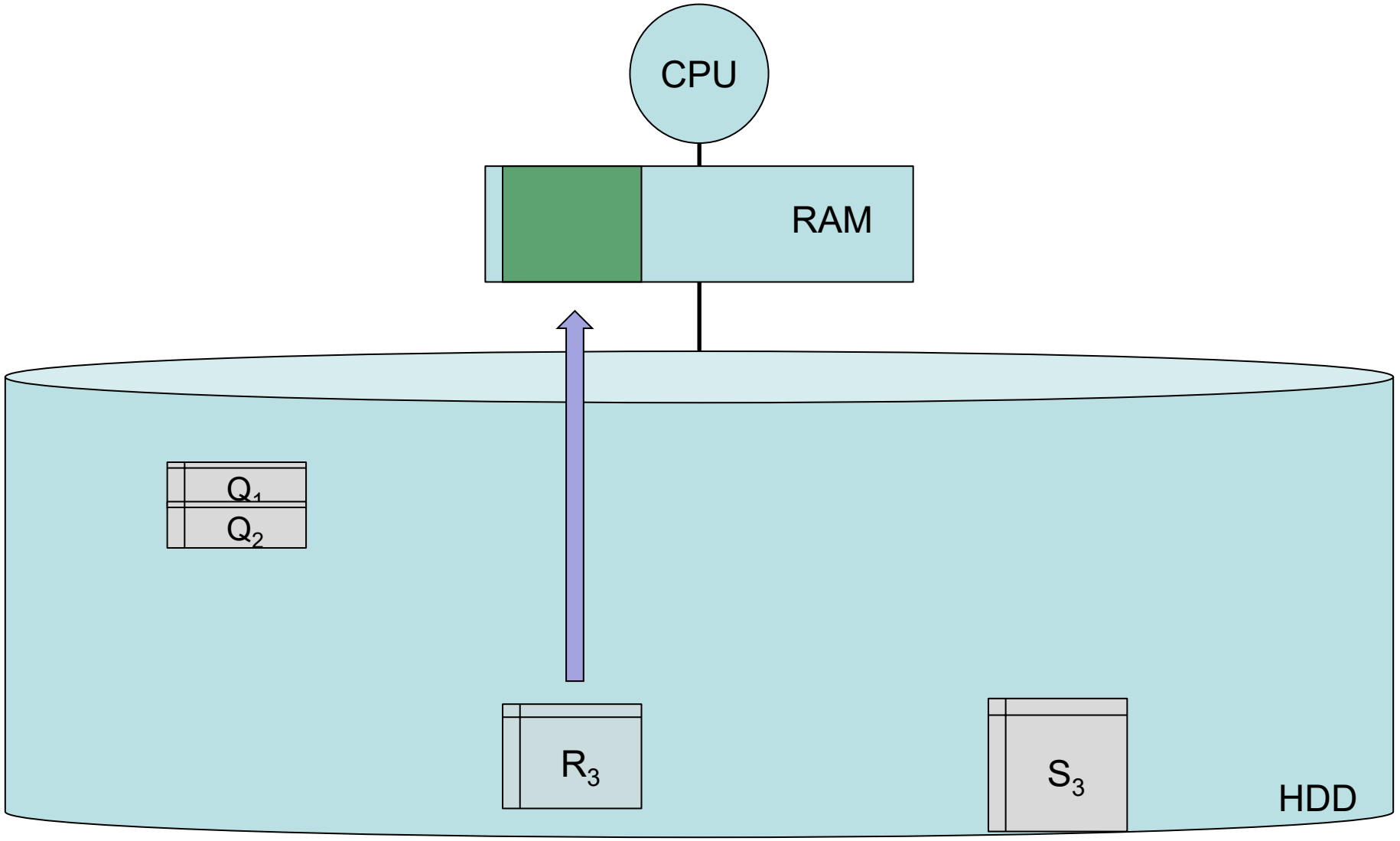
HJ. 2-й проход



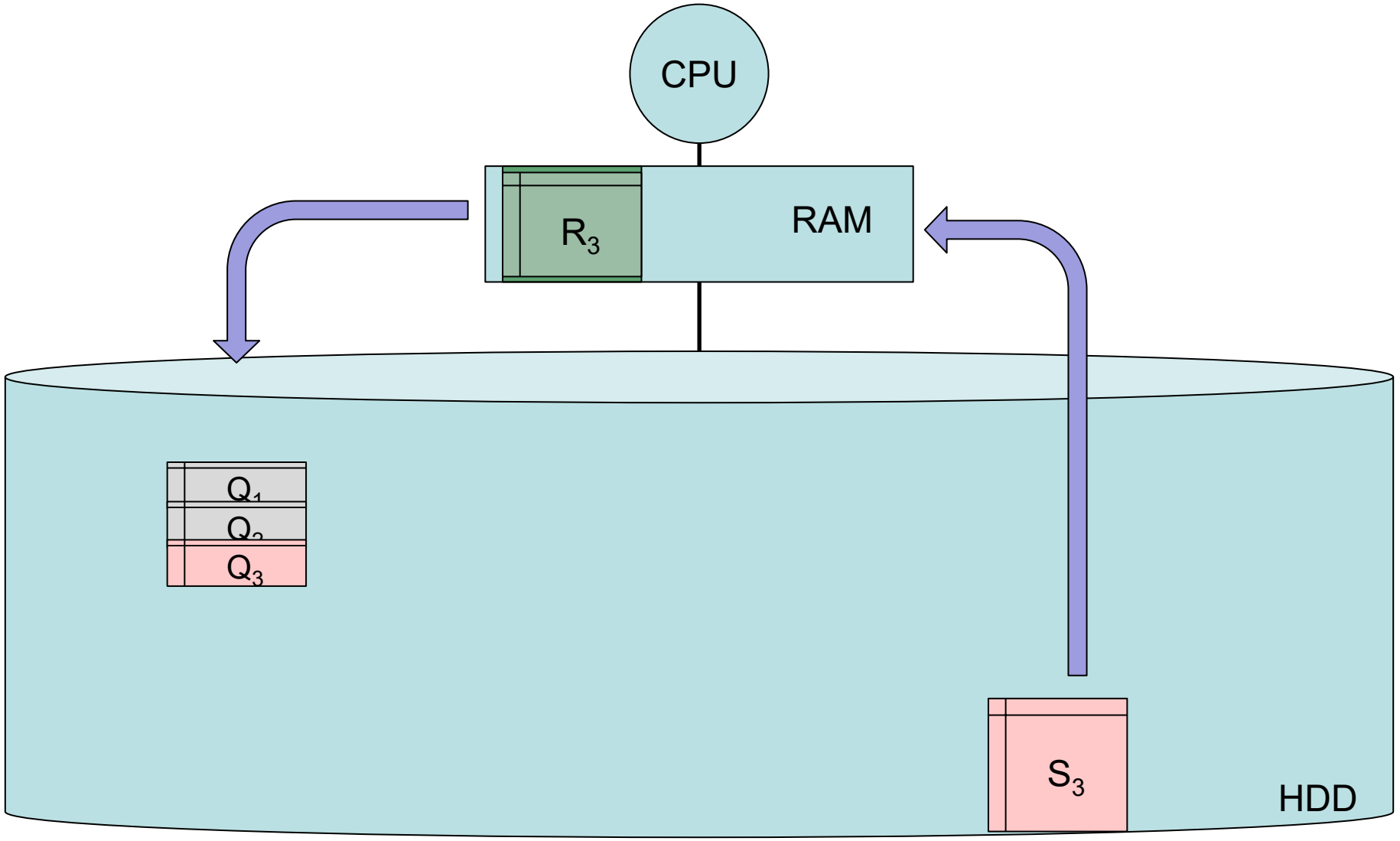
HJ. 2-й проход



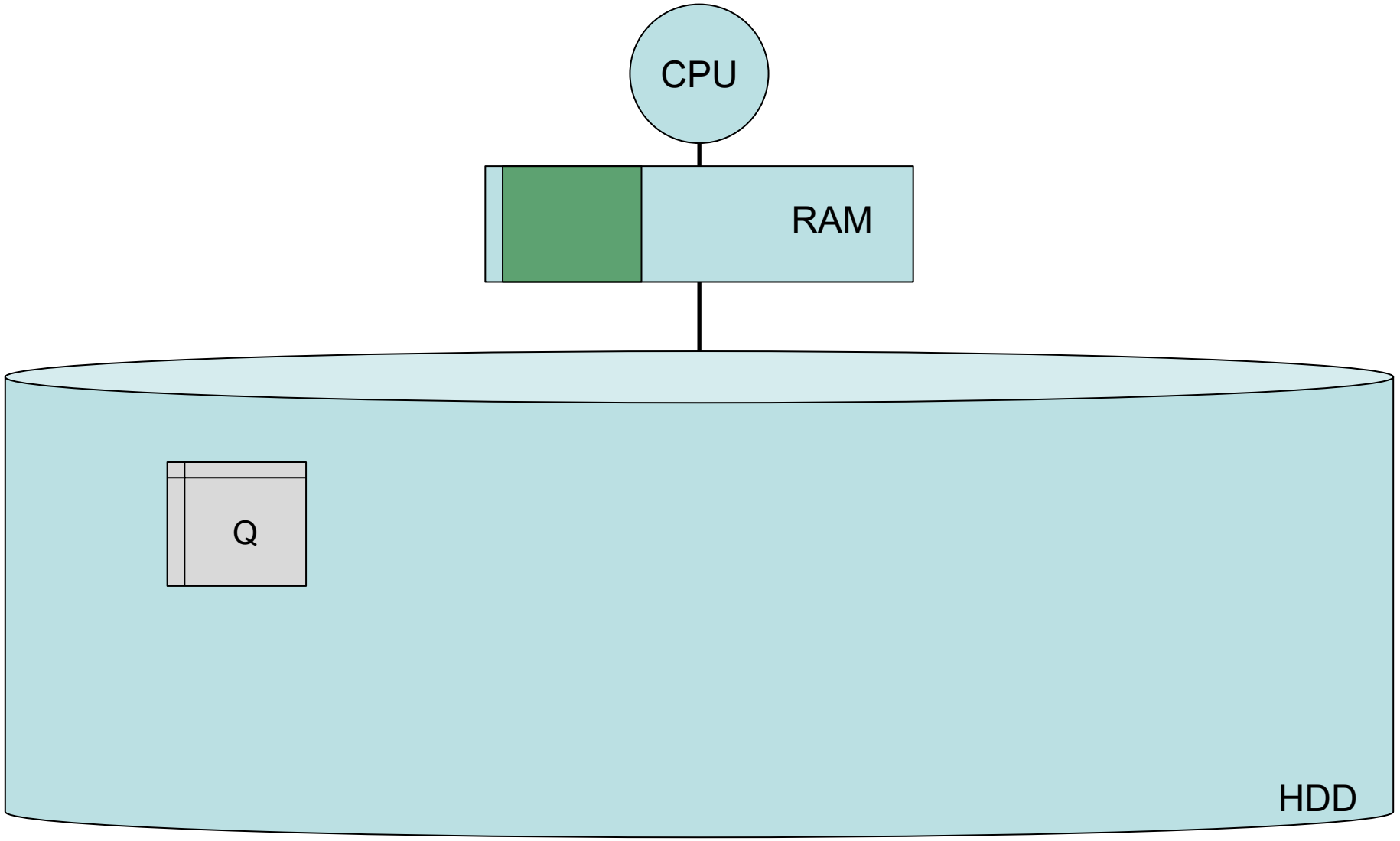
HJ. 2-й проход



HJ. 2-й проход



HJ. 2-й проход



Оценка эффективности *HHJ*

$R(A,B) \bowtie S(C,A)$ $\quad \boldsymbol{T}(R) = 150$ $\quad \boldsymbol{T}(S) = 3000$

Атрибут	Тип	Семантика	Длина (байт)
A	int	Целое	16
B	int	Целое	16
C	string	Строка	176

Объект	Длина (байт)
Заголовок записи	8
Длина блока	3024
Заголовок блока	24

	R	***S***
Длина записи	2·16 + 8 = **40**	16 + 176 + 8 = **200**
Записей в блоке	(3024-24)/40 = **75**	(3024-24)/200 = **15**
Количество блоков	150/75 = **2**	3000/15 = **200**

Число блоков ⇨

Для случая, когда буфер для R имеет размер в 1 блок:

Кол-во обменов с диском: $3\boldsymbol{B}(R)-2+3\boldsymbol{B}(S)-2\lfloor \boldsymbol{B}(S)/\boldsymbol{B}(R) \rfloor = 3\cdot 2-2+3\cdot 200-2\cdot\lfloor 200/2 \rfloor = 404$

MJPK (Merge Join by Primary Key)

Соединение слиянием по первичному ключу

$$R(A^*, B) \bowtie S(C, A^\#)$$

- R и S – ***отсортированы*** в порядке возрастания A
- A – первичный ключ в R и внешний ключ в S

MJPK : схема реализации метода open

```
void* open(node){
    node.leftSon.open();
    node.rightSon.open();
    node.leftSon.next();
    node.rightSon.next();
};
```

MJPK: схема реализации метода next

```
void* next(node){
   while(node.leftSon.buf!=EOF AND node.rightSon.buf!=EOF){
      if(MATCH_A(node.leftSon.buf,node.rightSon.buf)){
            node.buf=JOIN_A(node.leftSon.buf,node.rightSon.buf);
            node.rightSon.next();
            return;
      };
      node.leftSon.next();
   };
   node.buf=EOF;
};
```

Оценка эффективности *MJPK*

$R(A,B) \bowtie S(C,A)$ $\mathbf{T}(R)=150$ $\mathbf{T}(S)=3000$

Атрибут	Тип	Семантика	Длина (байт)
A	int	Целое	16
B	int	Целое	16
C	string	Строка	176

Объект	Длина (байт)
Заголовок записи	8
Длина блока	3024
Заголовок блока	24

	R	***S***
Длина записи	2·16 + 8 = **40**	16 + 176 + 8 = **200**
Записей в блоке	(3024-24)/40 = **75**	(3024-24)/200 = **15**
Количество блоков	150/75 = **2**	3000/15 = **200**

Число блоков ⇨

Количество чтений с диска:	**B**(*R*) + **B**(*S*) = 2+200 = **202**

Конец лекции 9