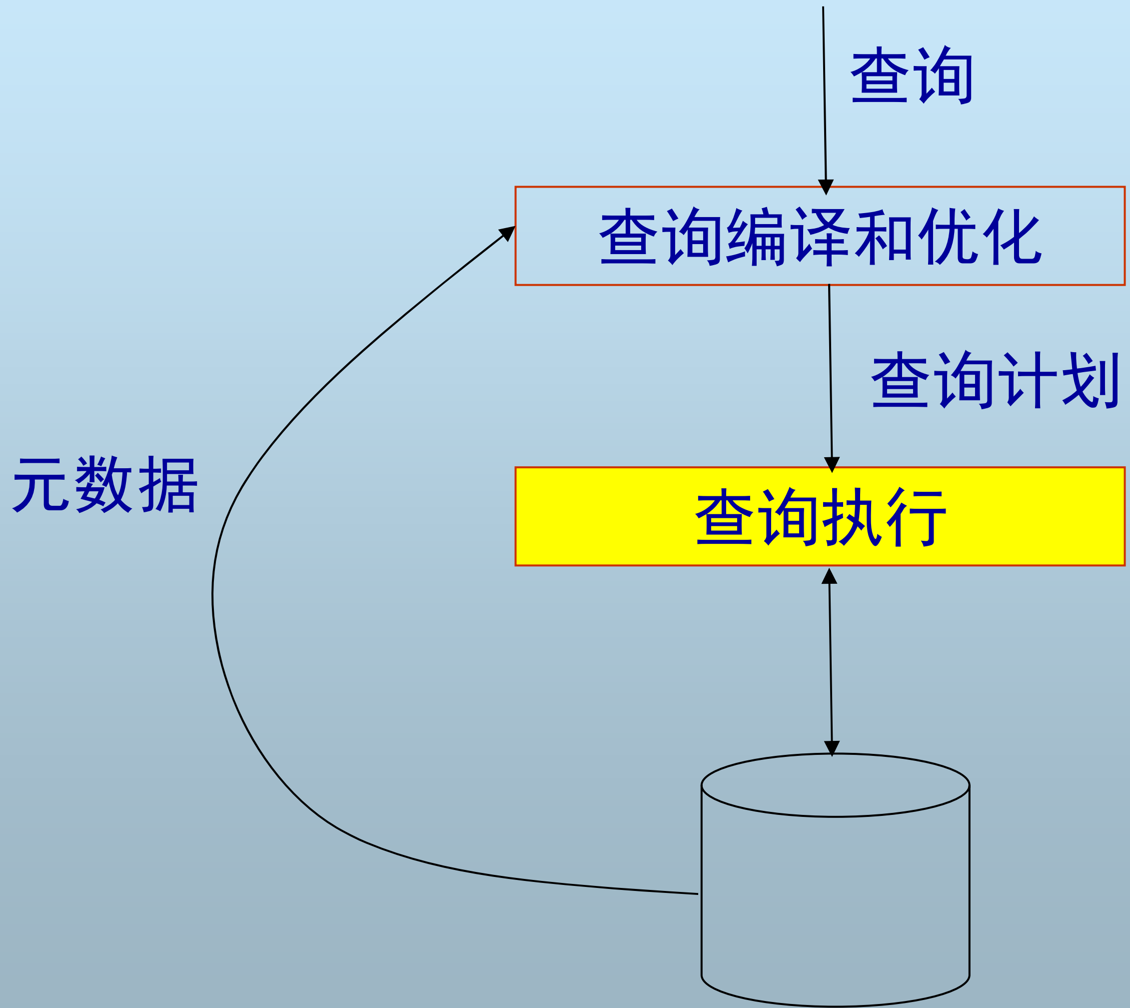
Query Execution

Chp.15 in textbook

金培权（jpq@ustc.edu.cn）

查询处理概述



主要内容

- 物理查询计划操作符
- 连接操作的实现算法 大多数的优化是针对连接操作的
  - 嵌套循环连接
  - 归并连接
  - 索引连接
  - 散列连接
- 连接算法的I/O代价估计

物理查询计划就是物理操作符的执行序列

一、物理查询计划操作符

- 逻辑操作符的物理操作符
  - 逻辑操作符的特定实现
- 其它物理操作符
  - 表扫描：TableScan
  - 排序扫描：SortScan
  - 索引扫描：IndexScan

一、物理查询计划操作符

- 物理操作符的执行算法
  - 一趟算法
  - 两趟算法
  - 多趟算法

  $\Big\}$ 按数据的读取方式

  - 基于排序的算法
  - 基于散列的算法
  - 基于索引的算法

  $\Big\}$ 按所基于的底层算法

二、连接操作(Join)的实现算法

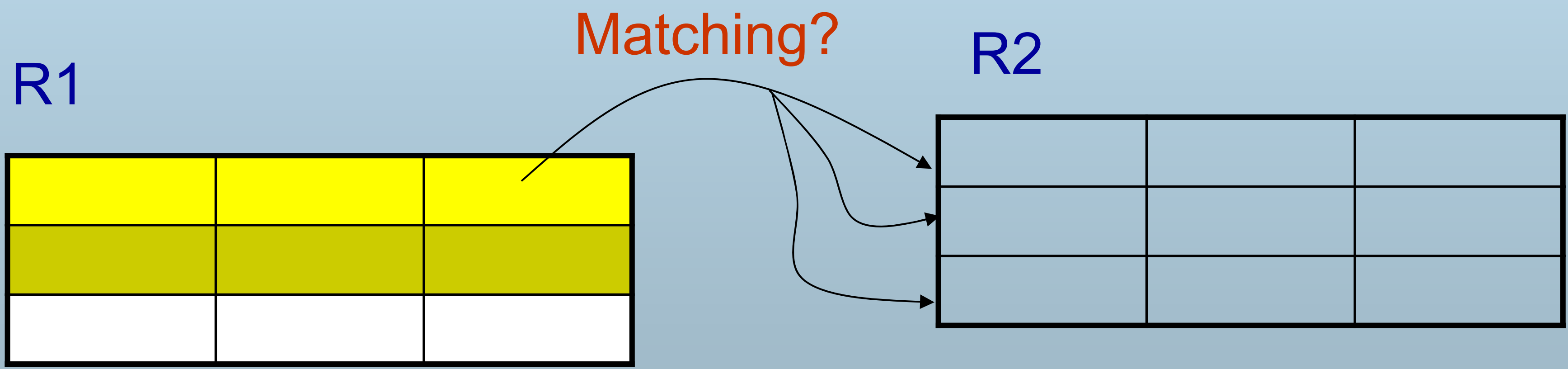
- R1(A,C)$\bowtie$R2(C,D)
  - 嵌套循环连接 (Nested loops join or Iteration join)
  - 归并连接 (Merge join)
  - 索引连接 (Join with index)
  - 散列连接 (Hash join)

1、嵌套循环连接

For each r ∈ R1 Do

For each s ∈ R2 do

If r.C = s.C Then output r, s pair



2、归并连接

(1) if R1 and R2 not sorted, sort them

(2) $i \leftarrow 1$; $j \leftarrow 1$;

While $(i \le T(R1)) \wedge (j \le T(R2))$ do {

if R1[i].C = R2[j].C then OutputTuples

else if R1[i].C > R2[j].C then $j \leftarrow j+1$

else if R1[i].C < R2[j].C then $i \leftarrow i+1$

}

2、归并连接

```
Procedure OutputTuples
  While (R1[ i ].C = R2[ j ].C) ∧ (i ≤ T(R1)) do {
      jj ← j;
      while (R1[ i ].C = R2[ jj ].C) ∧ (jj ≤ T(R2)) do {
          output pair R1[ i ], R2[ jj ];
          jj ← jj+1;
      }
      i ← i+1;
  }
```

2、归并连接

Example

i	R1[i].C	R2[j].C	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7

3、索引连接

For each r ∈ R1 do {

  X ← index (R2, C, r.C)

  For each s ∈ X do

    Output r,s pair

}

Assume R2.C index

T(R) / V(R, C) + k

Note: X ← index(rel, attr, value)

then X = set of rel tuples with attr = value

4、散列连接

- C上的散列函数 h, range 0 → k
- Buckets for R1: G0, G1, ... Gk
- Buckets for R2: H0, H1, ... Hk

Algorithm

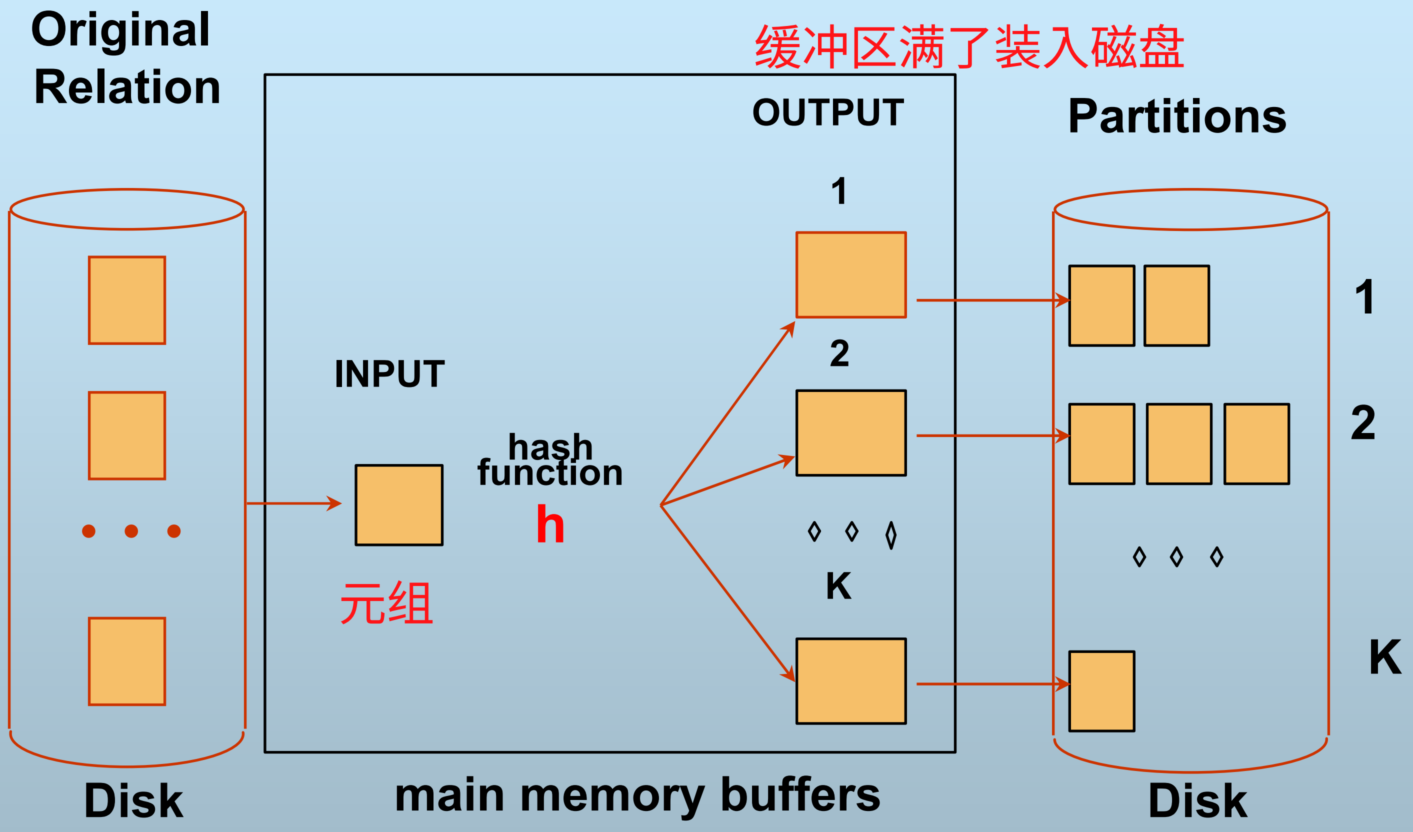
(1) Hash R1 tuples into G buckets

(2) Hash R2 tuples into H buckets

(3) For i = 0 to k do

match tuples in Gi, Hi buckets

4、散列连接


Original
Relation
缓冲区满了装入磁盘
OUTPUT
Partitions
1
INPUT
2
hash
function
h
元组
K
1
2
K
Disk
main memory buffers
Disk

4、散列连接

Simple example　　hash: even/odd

R1	R2
2	5
4	4
3	12
5	3
8	13
9	8
	11
	14

Buckets

	R1	R2
Even	2 4 8	4 12 8 14
Odd:	3 5 9	5 3 13 11

T(R) / k

三、连接算法的代价分析

- 影响连接算法代价(I/O)的因素
  - 关系的元组是否在磁盘块中连续存放？(contiguous?)
  - 关系是否按连接属性有序？ (ordered?)
  - 连接属性上是否存在索引？ (indexed?)

1、嵌套循环连接代价分析

- **Case1： not contiguous**
- **Case2： contiguous**

1、嵌套循环连接代价分析

Example 1：**not contiguous**

设 **T(R1) = 10,000　　T(R2) = 5,000**

S(R1) = S(R2) =1/10 block --元组大小

MEM=101 blocks

元组在块中不连续，每读取1个元组需要读取1个块，1次IO

<u>Cost:</u> For each R1 tuple:

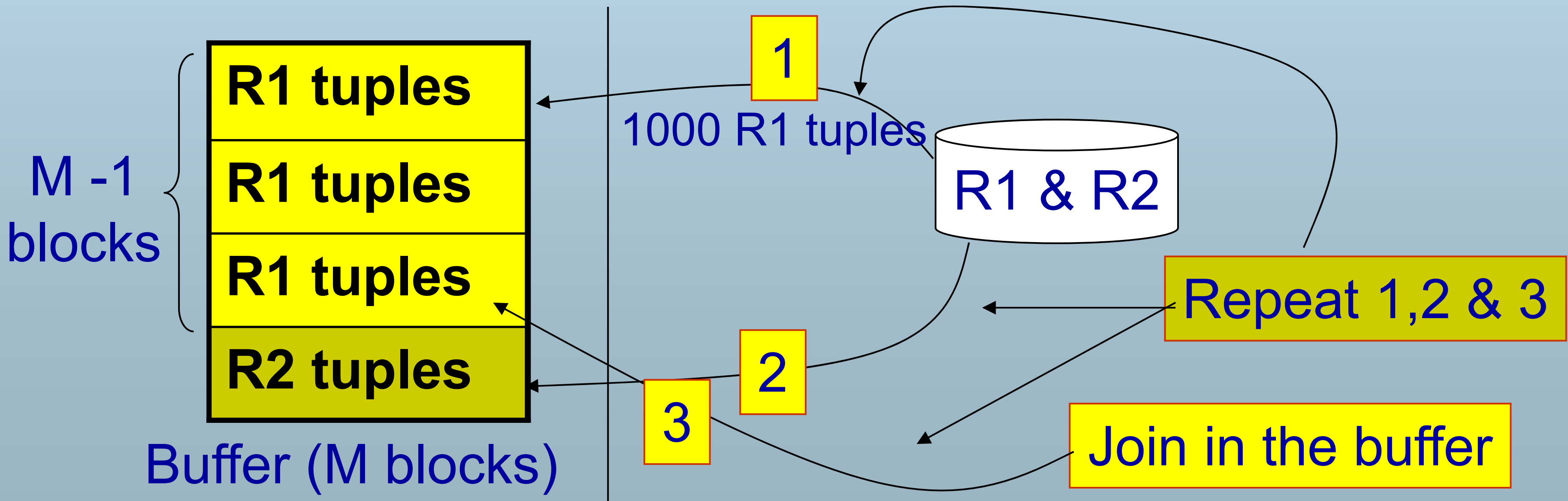
[Read tuple + Read R2]

Total =10,000 [1+5000]=50,010,000 IOs

1、嵌套循环连接代价分析

改进的执行策略

- (1) Read 100 blocks of R1 1个buffer块可装10个元组，100个buffer块可装1000个元组，需要1000次IO
- (2) Read all of R2 (using 1 block) + join
- (3) Repeat until done



1、嵌套循环连接代价分析

改进的执行策略

Cost: **For each loop:**

Read R1：1000 IOs (1000 tuples)

Read R2：5000 IOs (5000 tuples)

Total：6000 IOs

$$\text{Total} = \frac{10{,}000}{1{,}000} \times 6000 = 60{,}000 \text{ IOs}$$

Better than previous one!

1、嵌套循环连接代价分析

- **Can we further improve it?**

Reverse Join order! Since R1 $\bowtie$ R2 $\Leftrightarrow$ R2 $\bowtie$ R1

(1) Read 100 blocks of R2

(2) Read all of R1 (using 1 block) + join

(3) Repeat until done

$$\text{Total} = \frac{5000}{1000} \times (1000 + 10{,}000) = 5 \times 11{,}000 = 55{,}000 \text{ IOs}$$

much better!

1、嵌套循环连接代价分析

Example 2：**contiguous**

R2 $\bowtie$ R1

<u>Cost</u>

For each loop:

Read R2:	100 IOs
Read R1:	1000 IOs
	1,100

Total= 5 loops x 1,100 = 5,500 IOs

Where are we?

- 物理查询计划操作符
- 连接操作的实现算法
- 连接算法的I/O代价估计
  - 嵌套循环连接代价分析
  - 归并连接代价分析
  - 索引连接代价分析
  - 散列连接代价分析

2、归并连接代价分析

- 沿用前面的例子

  T(R1) = 10,000 T(R2) = 5,000

  S(R1) = S(R2) =1/10 block --元组大小

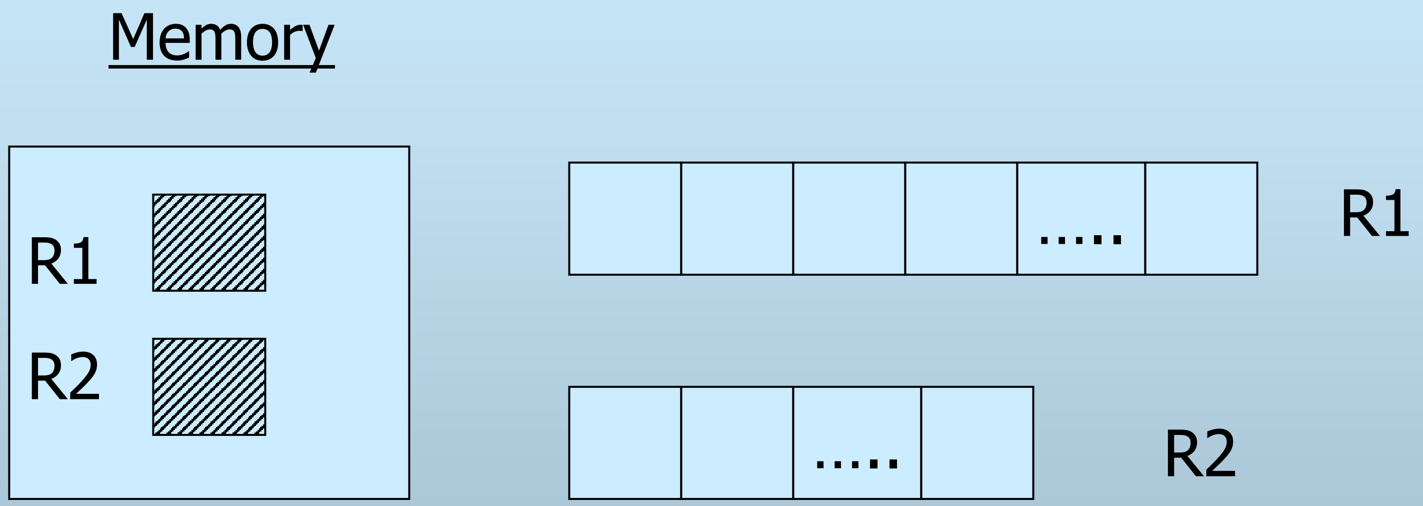
  MEM=101 blocks

2、归并连接代价分析

- **Still need to consider**
  - **Contiguous?**
  - **Ordered?**

2、归并连接代价分析

Example 3：**contiguous and ordered**



<u>Total cost</u>: Read R1 cost + read R2 cost

= 1000 + 500 = 1,500 IOs

2、归并连接代价分析

Example 4：**contiguous but not ordered**

- **Need to sort R1 and R2 first**

2、归并连接代价分析

Example 4：**contiguous but not ordered**

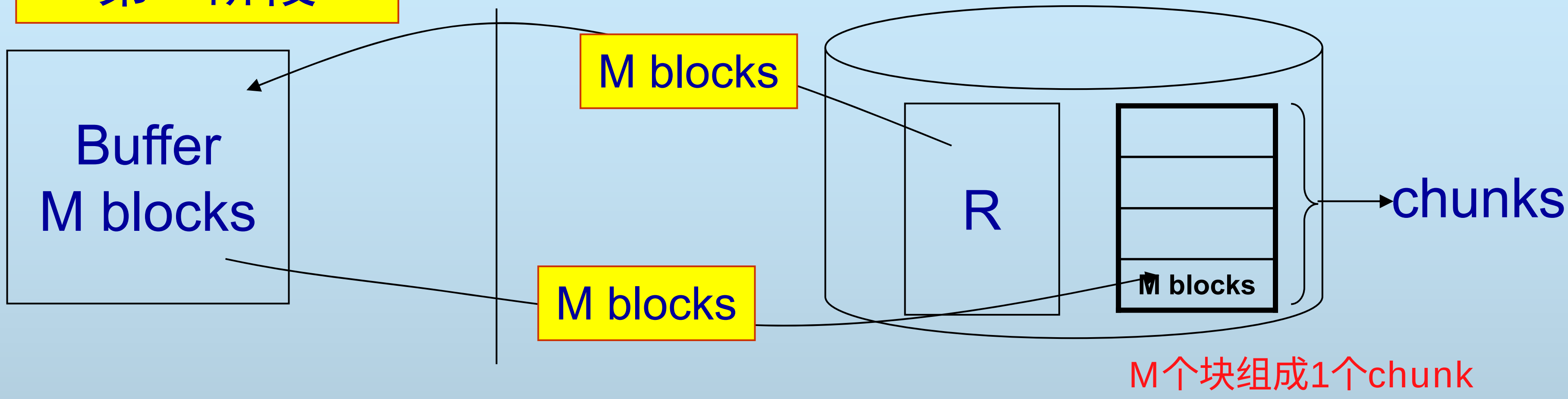
一种排序方法：两阶段多路归并排序

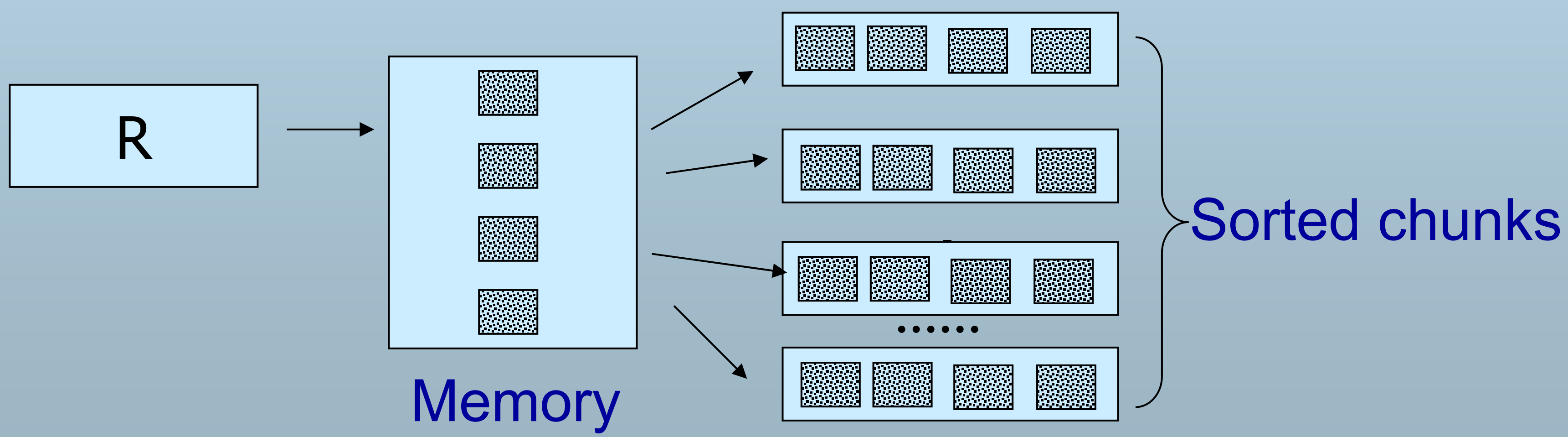
(i) For each 100 blocks of R:
- Read into memory
- Sort in memory
- Write to disk as a chunk

(ii) Read all chunks + merge + write out

2、归并连接代价分析

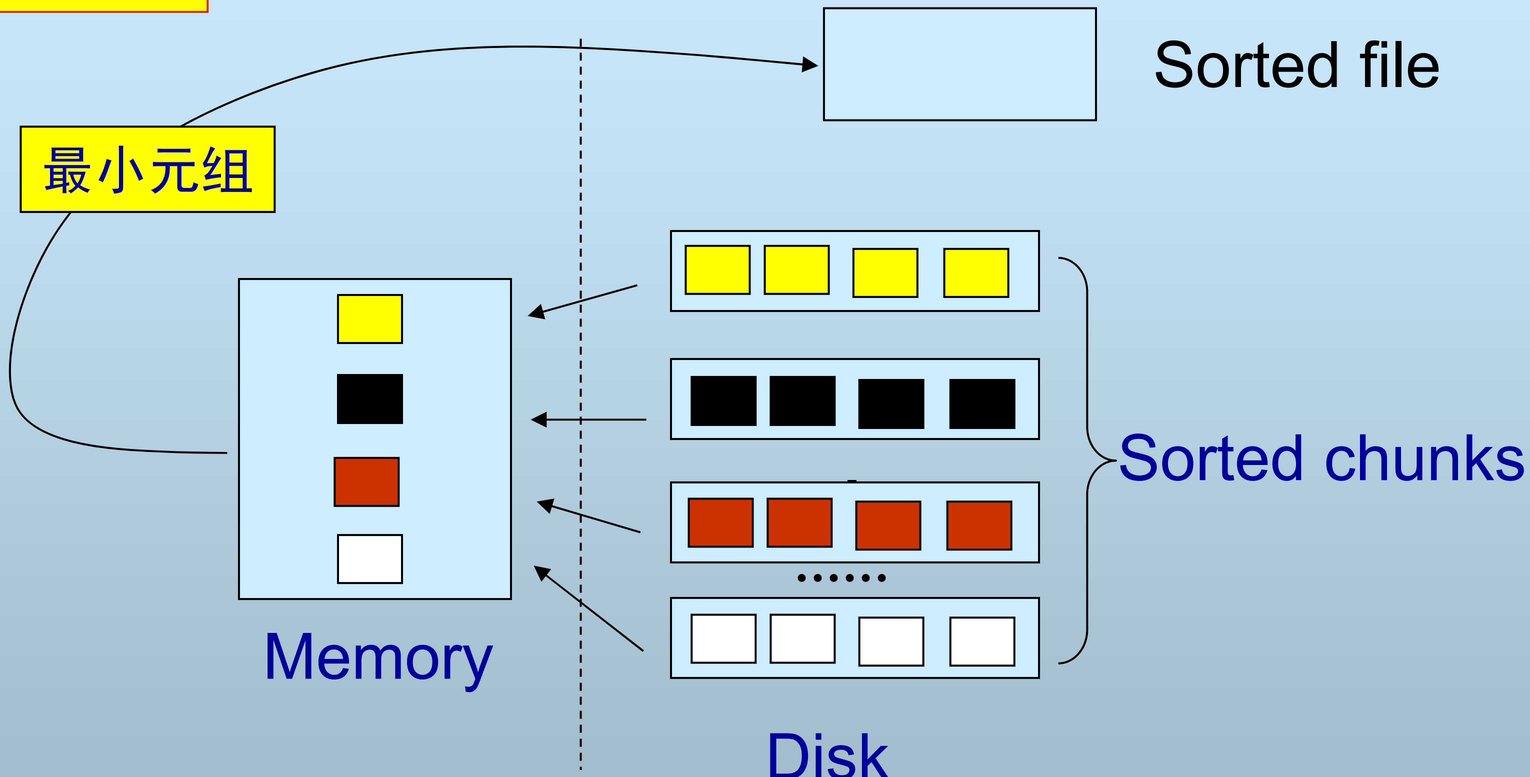
第一阶段





2、归并连接代价分析

第二阶段



从chunk中选第一个(最小块)载入buffer，再从buffer的块中归并连接

2、归并连接代价分析

Cost: Sort

Each tuple is read, written (first phase)
read, written (second phase)
So each tuple costs 4 IOs.

Sort cost R1: 4 x 1,000 = 4,000
Sort cost R2: 4 x 500 = 2,000
Total: 6,000 IOs

2、归并连接代价分析

Example 4：**contiguous but not ordered**

Cost: Merge join

Sort cost: 6,000
Join cost: 1,500
Total: 7,500 IOs = 5* 1,500 // 每个元组5次IO

But nested loop join only costs 5,500
So merge join does not pay off.

2、归并连接代价分析

Example 4：**contiguous but not ordered**

But if R1 = 10,000 blocks
R2 = 5,000 blocks

Iterate: $\frac{5000}{100} \times (100+10{,}000) = 50 \times 10{,}100$
$= 505{,}000$ IOs

Merge join: $5*(10{,}000+5{,}000) = 75{,}000$ IOs

Merge Join (with sort) WINS!

2、归并连接代价分析

Example 4：**contiguous but not ordered**

- **Cost : <u>Nested loop join</u> vs. <u>Merge Join</u>**
  - **Nested loop join**

$$Cost = \frac{B(R2)}{M-1}(M-1+B(R1)) = B(R2) + \frac{B(R1)B(R2)}{M-1}$$

  - **Merge Join**

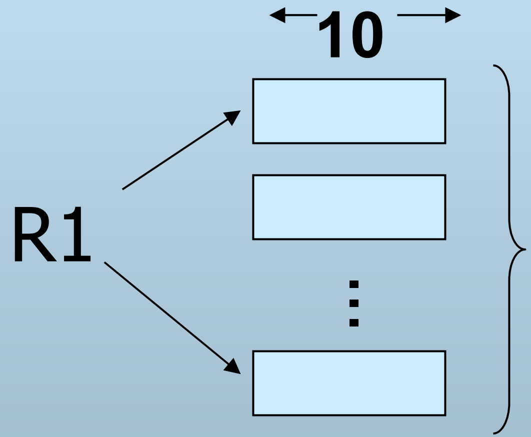
$$Cost = 5(B(R1)+B(R2))$$

嵌套循环连接是固有的二次算法，而归并连接是一次算法，当关系较小时，嵌套循环连接可能优于归并连接，但当关系较大时，归并连接更优。

2、归并连接代价分析

- 两阶段多路归并排序对Memory的要求

E.g: B(R1)=1000 and M=10



100 chunks ⇒ to merge, need
100 memory blocks!

2、归并连接代价分析

- 两阶段多路归并排序对Memory的要求

Say M=k, B(R)=x

\# chunks = (x/k) size of chunk = k

\# chunks 不能大于可用的Buffer block数

so... $(x/k) \leq k$

or $k^2 \geq x$ or $k \geq \sqrt{x}$

Buffer block数的平方必须大于等于排序关系R的块数B(R)

2、归并连接代价分析

- 在前面的例子中

  R1 is 1000 blocks, k $\geq$ 31.62
  R2 is 500 blocks, k $\geq$ 22.36

- 至少需有32个Buffer blocks才能执行归并连接

2、归并连接代价分析

- 归并连接算法改进 (for contiguous but not ordered)
  - 将第二阶段的排序和 join 合并进行

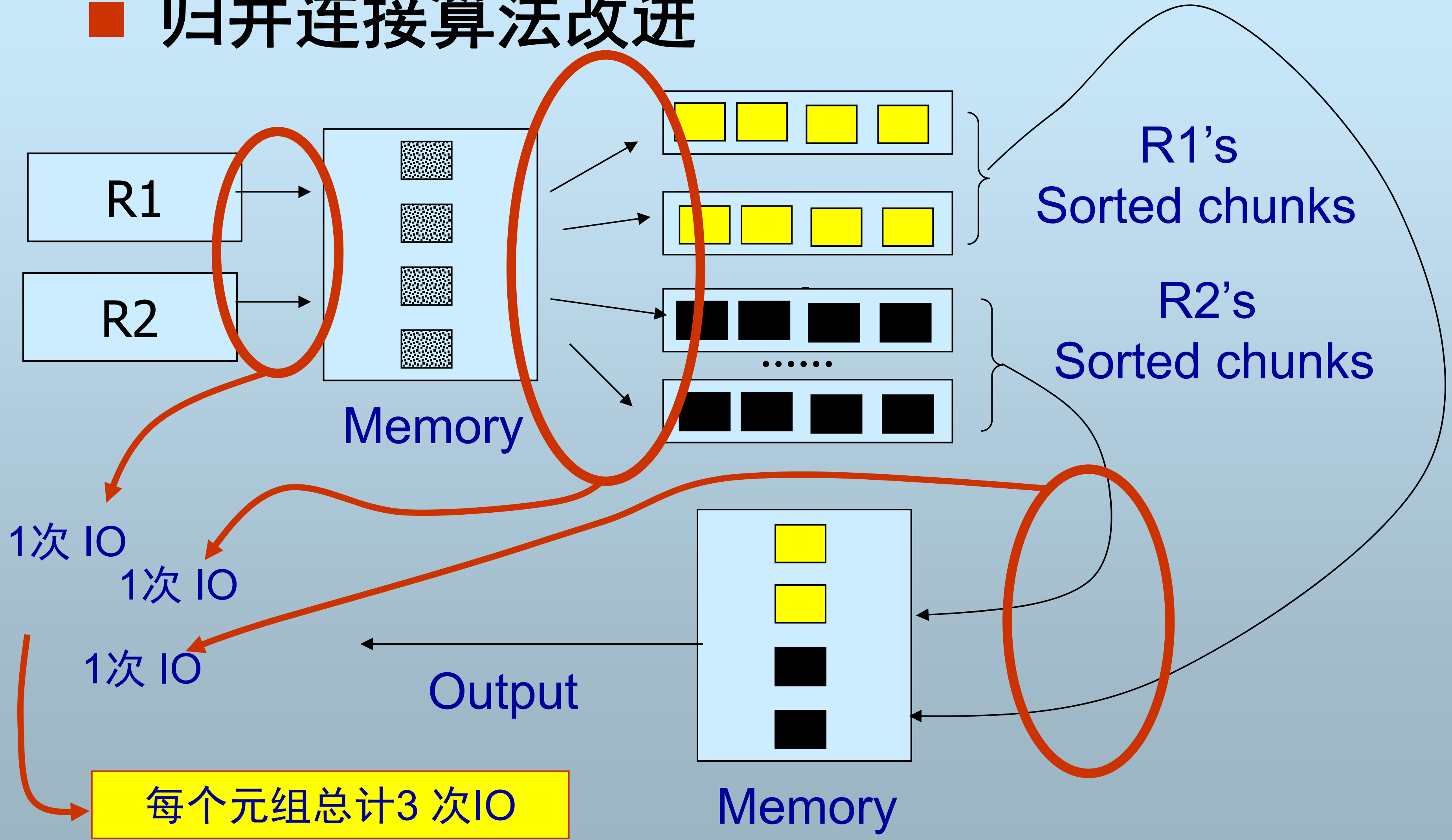
(1) Read R1 and R2 into sorted chunks (each has M blocks)

(2) Read first blocks of both R1's chunks and R2's into buffer

(3) Join in the memory

2、归并连接代价分析

- 归并连接算法改进



2、归并连接代价分析

- 归并连接算法改进

- Cost＝3（B(R1)+B(R2)）
  ＝3×1,500＝4,500

What are required？

- R1’s #chunks + R2’s #chunks ≤ M

Where we are?

- 物理查询计划操作符
- 连接操作的实现算法
- 连接算法的I/O代价估计
  - 嵌套循环连接代价分析
  - 归并连接代价分析
  - 索引连接代价分析
  - 散列连接代价分析

3、索引连接算法代价分析

- $R1(A,C) \bowtie R2(C,D)$
  - Assume R1.C index exists
  - Assume R1.C index fits in memory
  - Assume R2 contiguous, unordered

有索引的作为内循环，无索引的作为外循环

3、索引连接算法代价分析

Algorithm

```
for each R2 tuple:
        - probe index on R1.C     (1)
        - if match, read R1 tuple (2)
```

Cost

T(R1)=10,000, T(R2) = 5,000

(0) Read R2 tuples => 500 IOs
(1) Probe index =>No IOs

(2) Read matching R1 tuples => ?

3、索引连接算法代价分析

Matching tuples 选中率 p 估计

1. **若R1.C是主键, R2.C是外键,则 每个R2 tuple在R1中,选中率 p =1**
2. **若V(R1,C)=5,000, T(R1) = 10,000, 则 每个R2 tuple在R1中的选中率 p = T(R1)/V(R1,C)=2**

3、索引连接算法代价分析

Index join 总代价估计 **Cost= B(R2) + T(R2) * p**

1. **Cost = 500 + 5000*1 = 5,500**
2. **Cost = 500 + 5000*2 = 10,500**

3、索引连接算法代价分析

- **如果R1.C上的Index不能全部放在内存?**
  - **Suppose R1.C index is 200 blocks**

(1)把第二级索引块(假设只有1块)和另外98块第一级索引块放在Memory中
(2)Cost to probe index
=(0 IOs)*(98/200)+(1 IOs)*(102/200)
≈ 0.5 IOs

3、索引连接算法代价分析

- **如果R1.C上的Index不能全部放在内存?**
- **Cost = B(R2) + T(R2) *(Probe index cost + read tuples)**
  1. **Cost = 500 + 5000 * (0.5 +1) = 8,000**
  2. **Cost = 500 + 5000 * (0.5 + 2) = 13,000**

Where are we?

- 物理查询计划操作符
- 连接操作的实现算法
- 连接算法的I/O代价估计
  - 嵌套循环连接代价分析
  - 归并连接代价分析
  - 索引连接代价分析
  - 散列连接代价分析

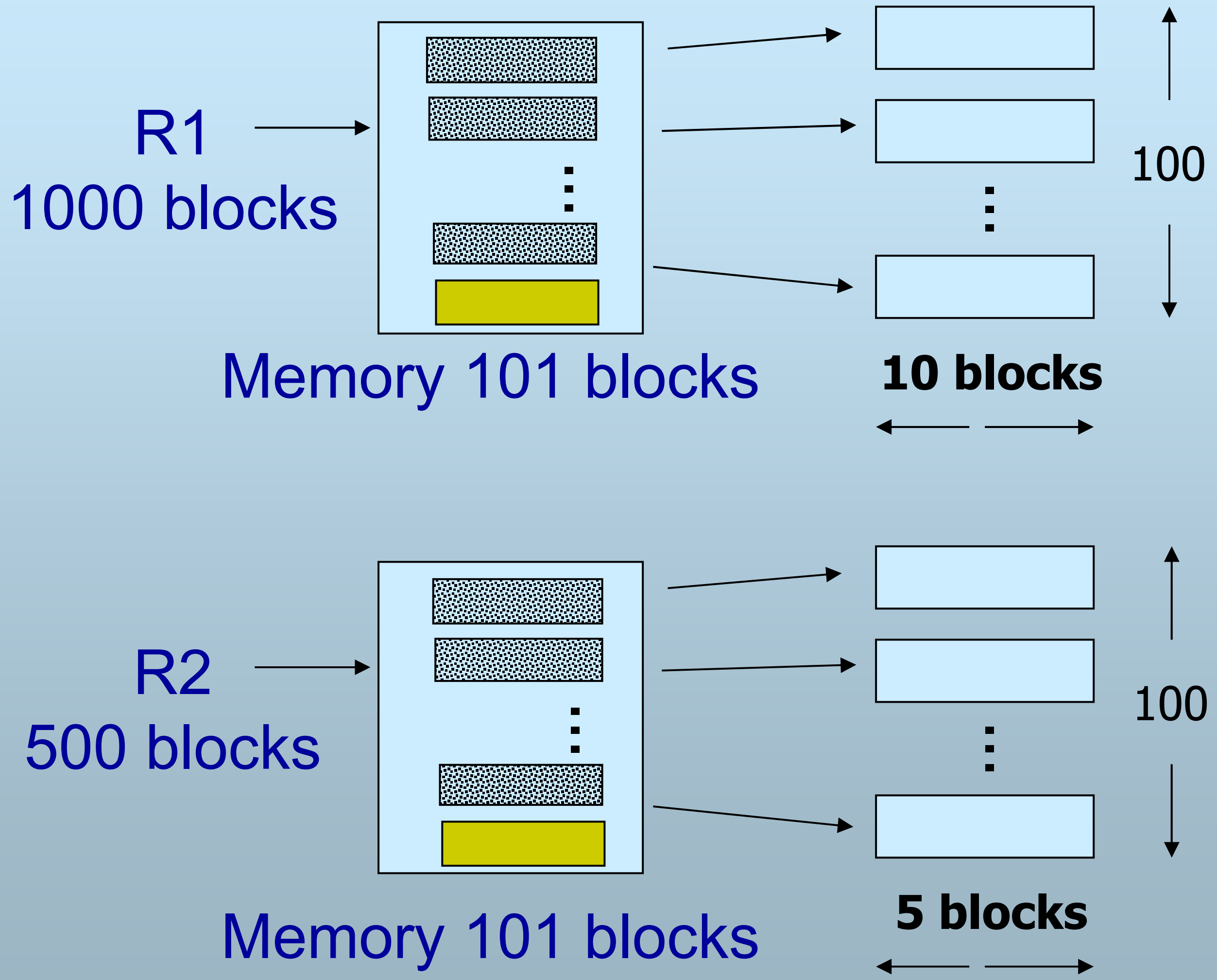
4、散列连接算法代价分析

- Say R1, R2 contiguous but not ordered
- Say 100 hash buckets

(1)Read R1, Hash, Write into buckets
(2)Read R2, Hash, Write into buckets
(3)Repeat
　① Read one bucket of R2 (say B(R2)≤ B(R1))
　② Read corresponding R1 bucket
　③ Join in the memory

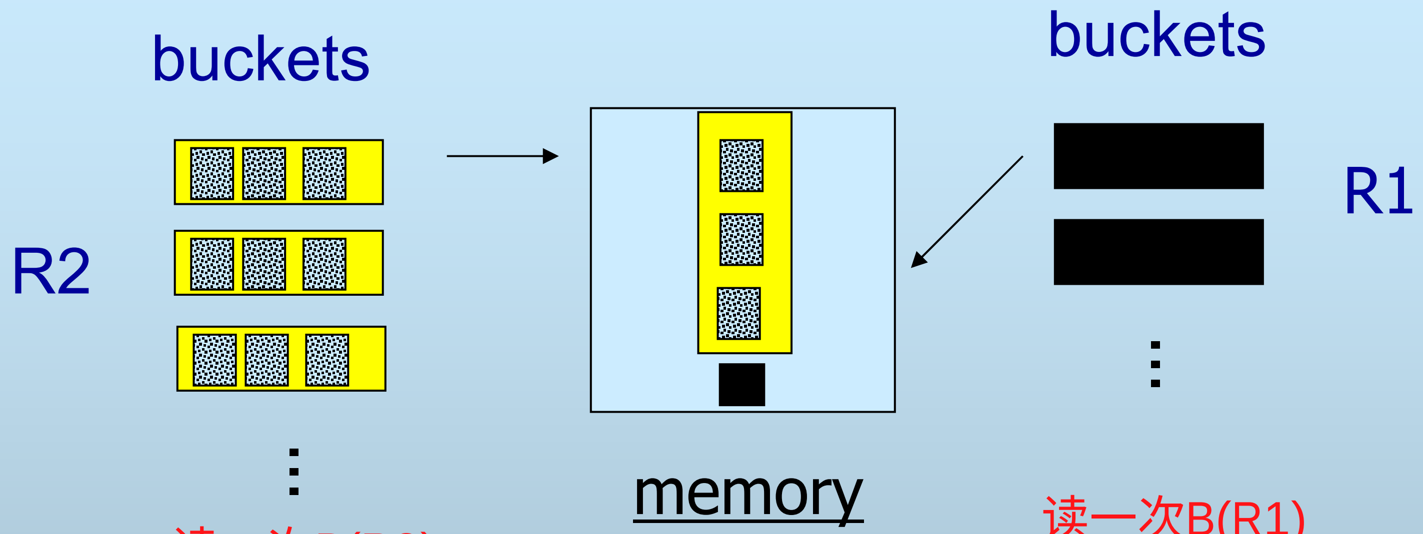
Note：一块一块地读入R1 bucket中的块，并Join。但这不影响IO代价

4、散列连接算法代价分析



Note：
划分为M－1个桶，每一块对应一个桶，最后一块用于读入R1的一块，计算其中每个元组的h，并将元组复制到相应的块中。

4、散列连接算法代价分析



读一次B(R2)

读一次B(R1)

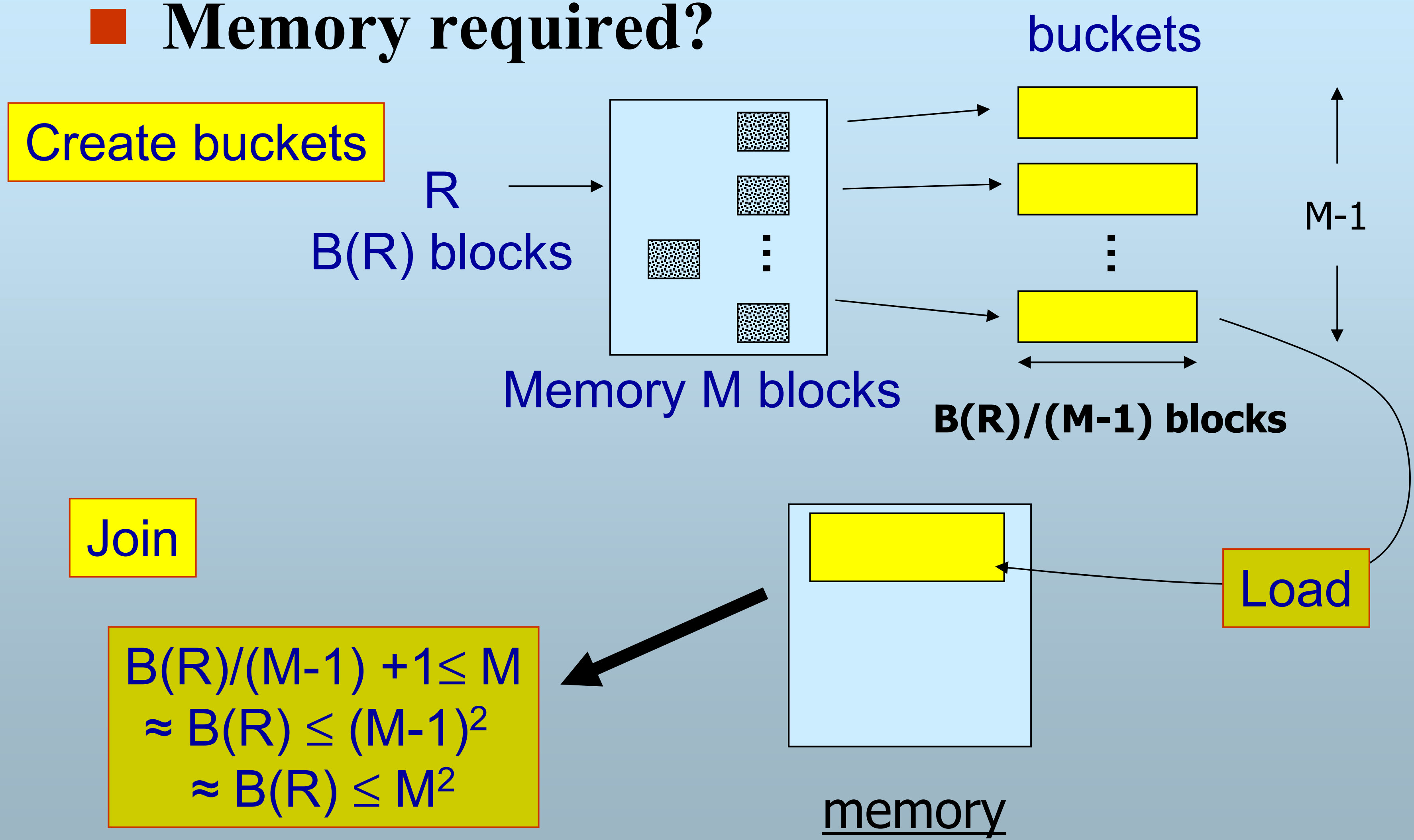
4、散列连接算法代价分析

- Cost: For each block
  - Create buckets
    - R1: Read + Write
    - R2: Read + Write
  - Join
    - R1: Read
    - R2: Read

Total : 3 * (B(R1) + B(R2)) = 4,500

4、散列连接算法代价分析

- **Memory required?**



$$B(R)/(M-1) + 1 \leq M$$
$$\approx B(R) \leq (M-1)^2$$
$$\approx B(R) \leq M^2$$

4、散列连接算法代价分析

- **Memory required?**
  - **For R1 $\bowtie$ R2**
  - **$Min(B(R1), B(R2)) \leq M^2$**

局限性：只能处理等值连接，需要设计好的Hash函数，将元组均匀分布到各个桶中

5、连接算法总结

算法 [1]	Cost	M
Nested Loop Join	B(R2)+B(R1)B(R2)/M	≥ 2
Merge Join	5(B(R1)+B(R2))	$\sqrt{B(R1)}$
Merge Join (improved)	3(B(R1)+B(R2))	$\sqrt{B(R1)+B(R2)}$
Index Join	B(R2)+T(R1)T(R2)/V(R1,C)	LB(R1.C) [2]
Hash Join	3(B(R1)+B(R2))	$\sqrt{B(R2)}$

1: suppose B(R2) ≤ B(R1)
2: suppose index fits in memory

5、连接算法总结

- Nested loop **ok** for "small" relations (relative to memory size)
- For equi-join, where relations not sorted and no indexes exist, hash join usually best
- Sort + merge join good for non-equi-join (e.g., R1.C > R2.C)
- If relations already sorted, use merge join
- If index exists, it could be useful

  (depends on expected result size)

本章小结

- 物理查询计划操作符
- 连接操作的实现算法
  - 嵌套循环连接
  - 归并连接
  - 索引连接
  - 散列连接
- 连接算法的I/O代价估计
- Other operators？ -- see textbook