第10讲 查询处理（上）

教学内容

1. 查询处理概述
2. 查询代价的度量
3. 查询计划的执行策略:
4. 具体查询算法
   （1）选择操作算法
   （2）连接操作算法
   （3）排序算法
   （4）其他操作算法

1. 查询处理概述

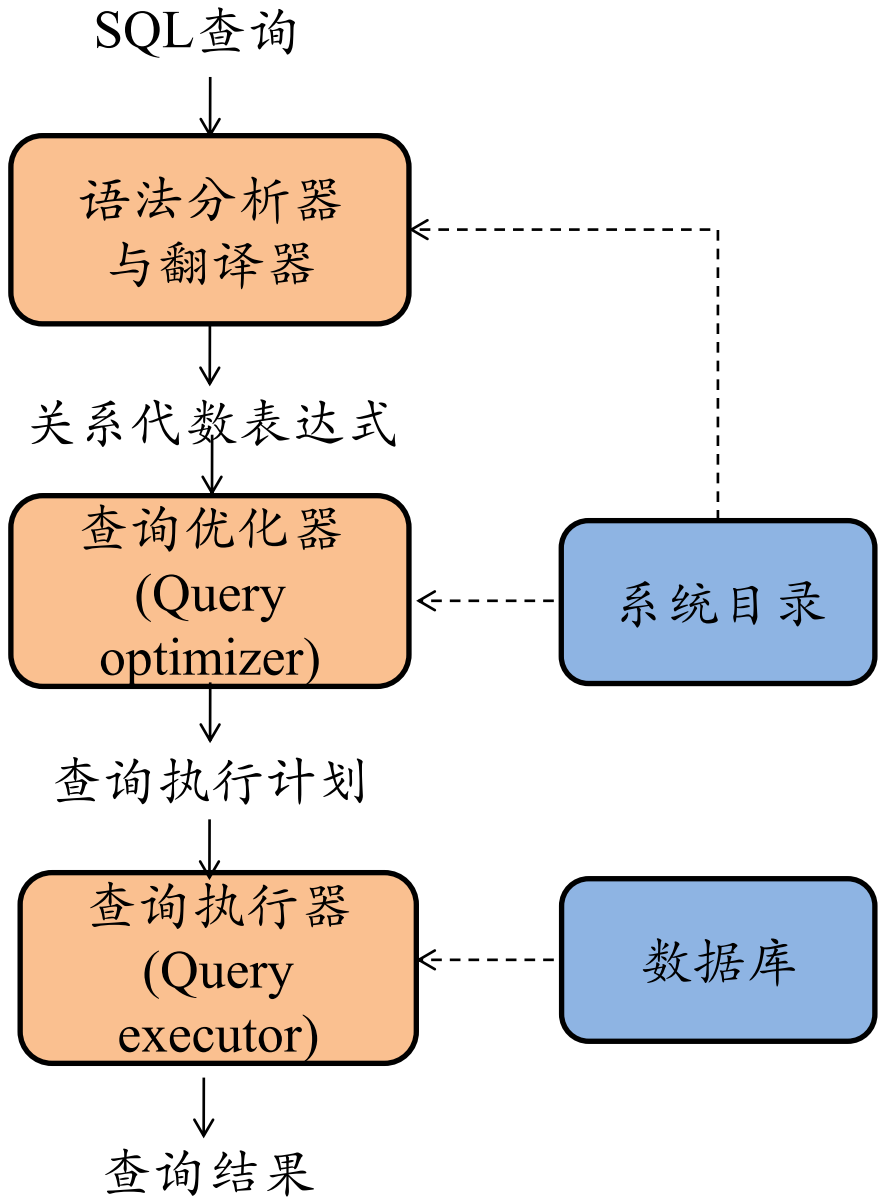
（1）查询处理的基本过程

查询处理 (Query Processing) 是指从数据库中提取数据时涉及的一系列活动。

这些活动包括：将SQL查询语句翻译为能在文件系统的物理层上使用的表达式，为优化查询而进行各种转换，以及查询的实际执行。

基本过程包括：

1. 语法分析与翻译
2. 优化
3. 执行



（2）语法分析器与翻译器

语法分析器与翻译器将一个SQL查询转换成关系代数表达式

Example (语法分析与翻译)

关系： Teacher (ID, Name, Dno, salary)， Dept(Dno, Dname)

SQL查询：

select salary, Dname **from** Teacher **Natural Join** Dept **where** salary < 75000;

关系代数表达式：

$$\Pi_{salary,Dname}(\sigma_{salary<75000}(Teacher \bowtie Dept))$$

原语操作是带有“如何执行”注释的关系代数操作

查询执行计划是用于执行一个查询的原语操作序列

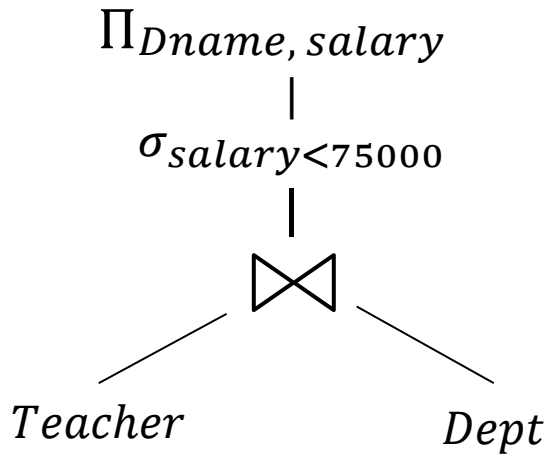
1. 查询处理概述

（3）查询优化器

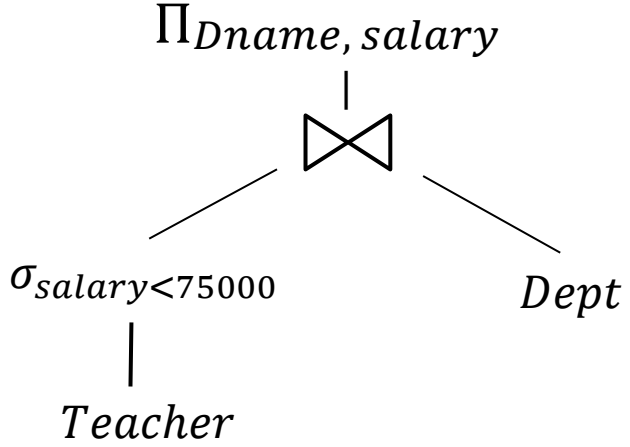
- 查询优化器将一个关系代数表达式转换为一个执行效率最高的等价关系代数表达式，并最终转换为一个查询执行计划 (Query Execution Plan)
- 查询计划通常可表达为树形结构，每个节点为一个基本操作

Example (查询优化)

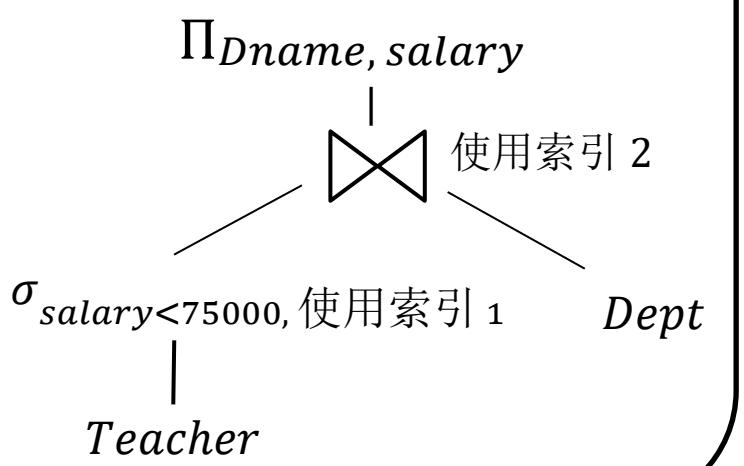
初始关系代数表达式



等价代数关系表达式



查询执行计划



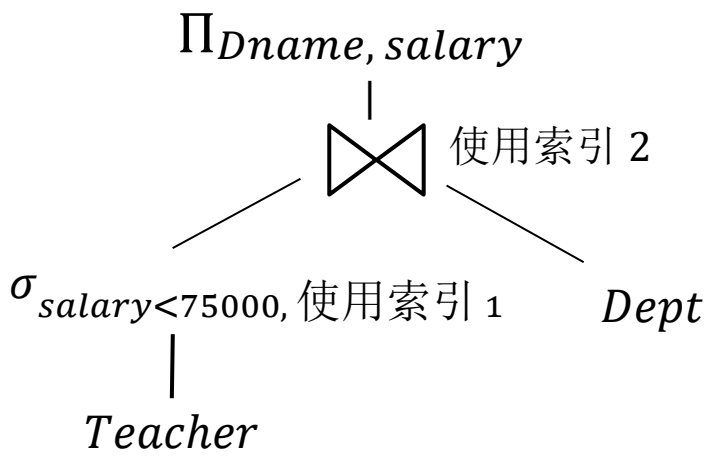
（4）查询执行器

查询执行器 (Query Executor) 按照查询优化器给出的查询执行计划执行查询

查询处理：Query Evaluation/Query Execution

Example (查询执行)

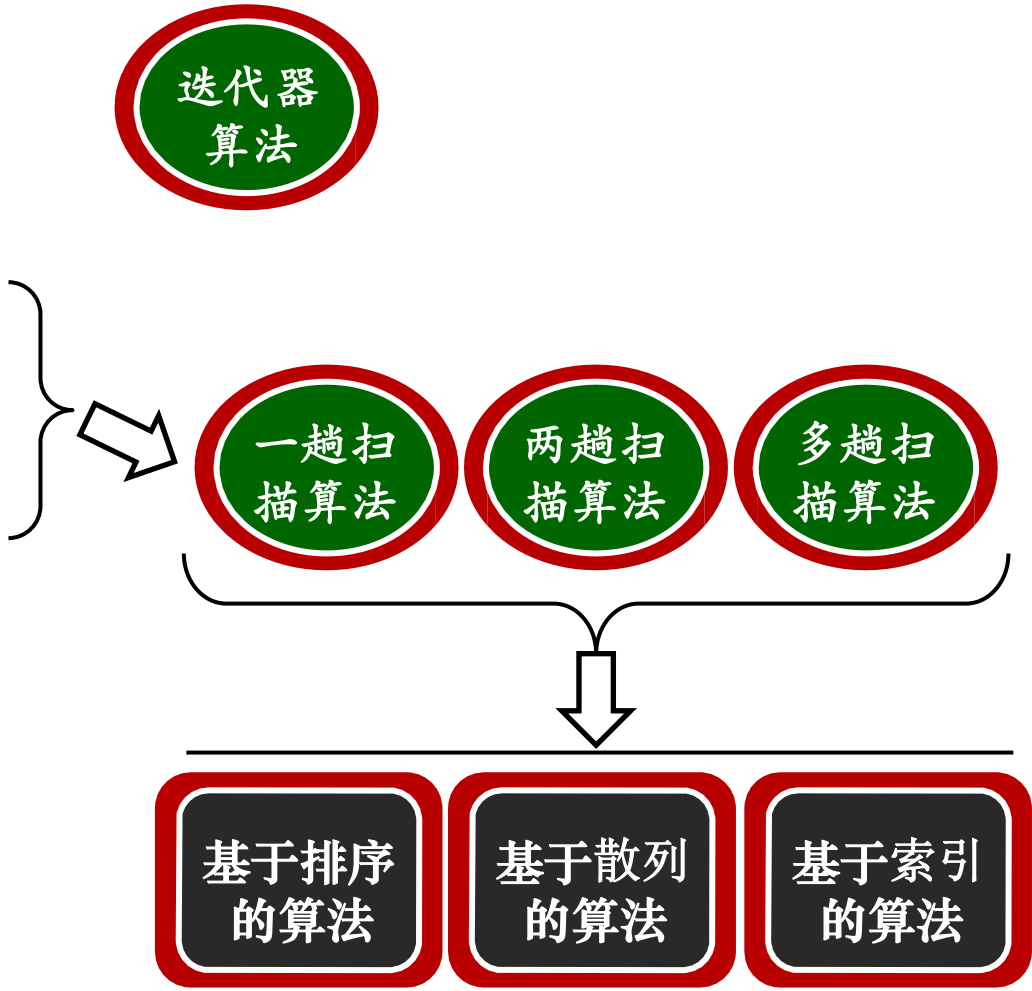
查询执行计划



（5）查询实现算法总览

数据库的三大类操作

- 一次单一元组的一元操作
  - ✓ $\sigma_F(R), \pi_\alpha(R)$——**selection, projection**
- 整个关系的一元操作
  - ✓ $\delta(R), \gamma(R), \tau(R)$——**distinct, group by, sorting**
- 整个关系上的二元操作
  - ✓ 集合上的操作：$\cup_S, \cap_S, -_S$
  - ✓ 包上的操作：$\cup_B, \cap_B, -_B$
  - ✓ 积、连接：**product, join**



教学内容

1. 查询处理概述
2. 查询代价的度量
3. 查询计划的执行策略
4. 选择操作的实现算法(σ)
5. 连接操作的实现算法($\bowtie$)

2. 查询代价的度量

查询处理的代价可通过该查询对各种资源的使用情况进行度量，包括磁盘存储、执行一个查询所用CPU时间，再并行/分布式数据库系统中的通信代价等

磁盘上存取数据的代价通常是最主要的代价，因为磁盘存取比内存操作速度慢

DBMS如何衡量物理查询计划的优劣

- **I/O访问次数(最重要)：多少个数据块的读写操作。假设DBMS数据块=磁盘数据块**
- CPU的占用时间
- 内存使用代价(与缓冲区数目与大小的匹配)
- 中间结果存储代价
- 计算量(如搜索记录、合并记录、排序记录、字段值的计算等)
- 网络通信量等

2. 查询代价的度量

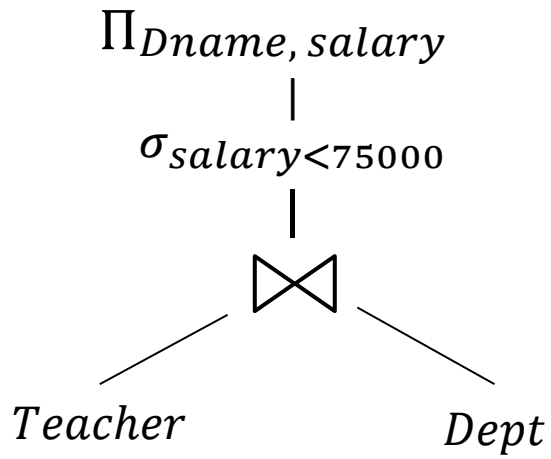
如何进行查询优化?

根据各查询计划的代价计算来选择最优查询计划。

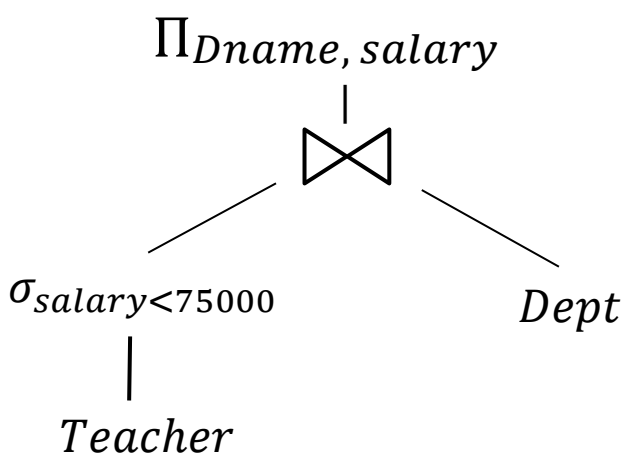
在下一讲中具体讨论。

Example (查询优化)

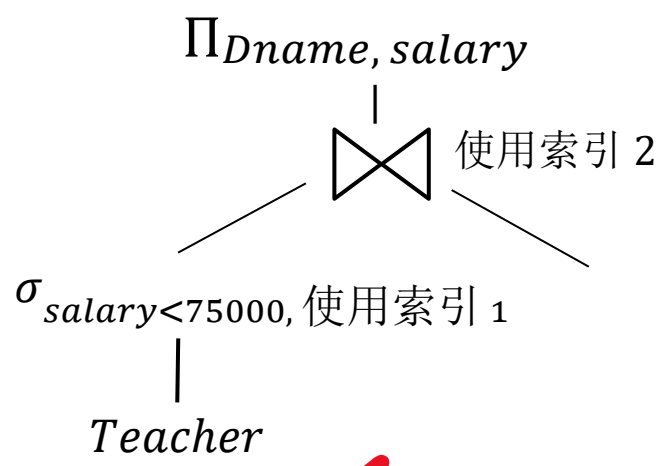
执行计划1



执行计划2



执行计划3



教学内容

1. 查询处理概述
2. 查询代价的度量
3. 查询计划的执行策略
4. 选择操作的实现算法(σ)
5. 连接操作的实现算法($\bowtie$)

3. 查询计划的执行方案: 查询实现的两种策略

查询实现的不同策略

- 物化计算策略
- 流水线计算策略（又称迭代器模型）

3. 查询计划的执行方案

策略一：物化执行（Materialization）

物化执行：运算的每个中间结果被创建(物化)，然后用于下一层的运算

- 自底向上执行查询计划中的操作
- 每个中间结果写入临时关系文件，作为后续操作的输入

$$\Pi_{S\#,Sname}(\sigma_{C\#="001"\wedge Student.S\#=SC.S\#}(Student\times SC))$$

物化执行的缺点

1. 增加了查询执行的代价
   - 执行完一个操作后，临时关系须写入文件
   - 执行后续操作时，临时文件被再次读入缓冲区
2. 获得结果的时间延迟大

$$Temp1 \leftarrow Student \bowtie SC$$

$$\Downarrow$$

$$Temp2 \leftarrow \sigma_{C\#="001"}(Temp1)$$

$$\Downarrow$$

$$结果关系 \leftarrow \pi_{S\#,Sname}(Temp2)$$

扫描三遍

3. 查询计划的执行方案

物化执行

物化执行：$\prod_{Sname}(\sigma_{Sdept='Math'}Student)$, Student(<u>SID</u>, Sname, Sdept,)

SID	Sname	Sdept
1	张三	Math
2	李四	Math
3	王五	Music
4	赵六	English

$\sigma_{Sdept='Math'}$

SID	Sname	Sdept
1	张三	Math
2	李四	Math

$\prod_{Sname}$

Sname
张三
李四

每个中间结果被创建，然后用于下一层的运算

3. 查询计划的执行方案

策略二：流水线执行（Pipelining）

流水线执行：将查询计划中若干操作组成流水线，一个操作的结果不进行存储，而是直接传给流水线中下一个操作

- 避免产生临时关系及读写临时关系文件的I/O开销
- 用户能够尽早得到查询结果

几乎所有DBMS都使用流水线执行查询计划

符合选择条件的记录直接送给投影操作

Example (流水线执行)

Π_{salary} - - - - - - - - - - - - - - - - - - → 输出关系

| 流水线

$\sigma_{salary>75000;使用索引1}$ ~~临时关系~~

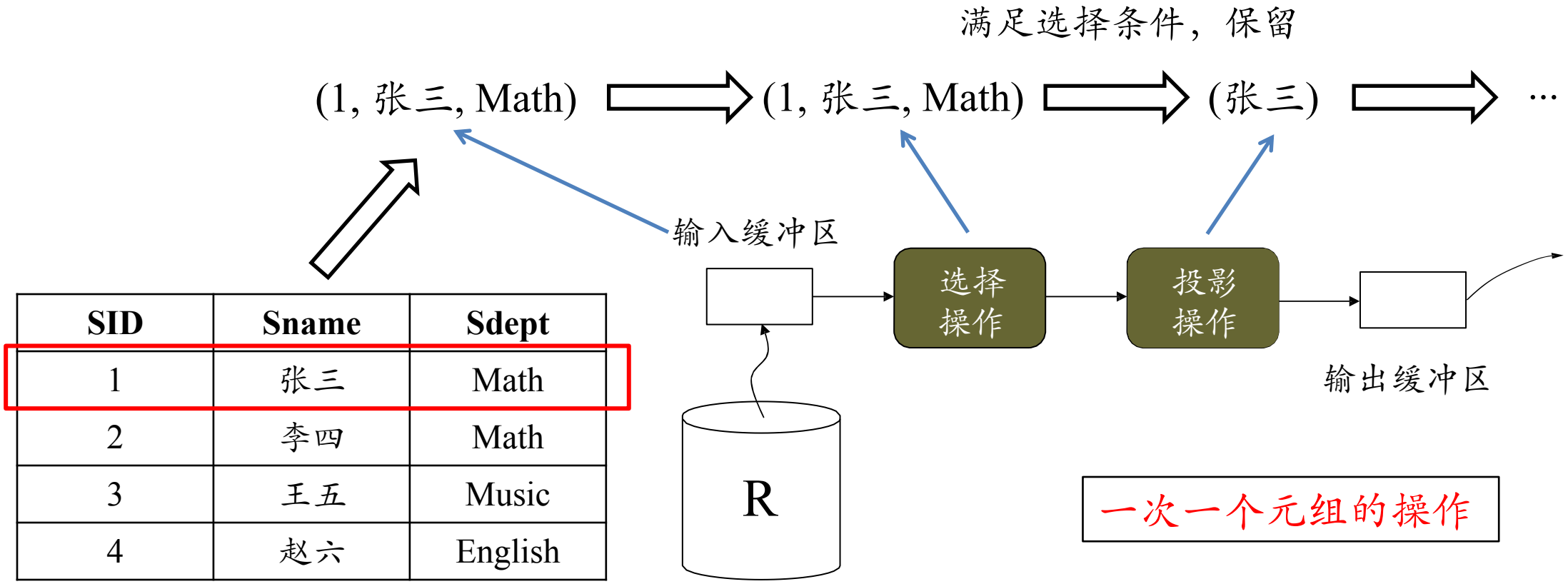
|

Teacher ← - - - - - - - - - - - - - - - - 输入关系

3. 查询计划的执行方案

流水线执行

流水线执行：$\prod_{Sname}(\sigma_{Sdept='Math'}Student)$, Student(SID, Sname, Sdept,)

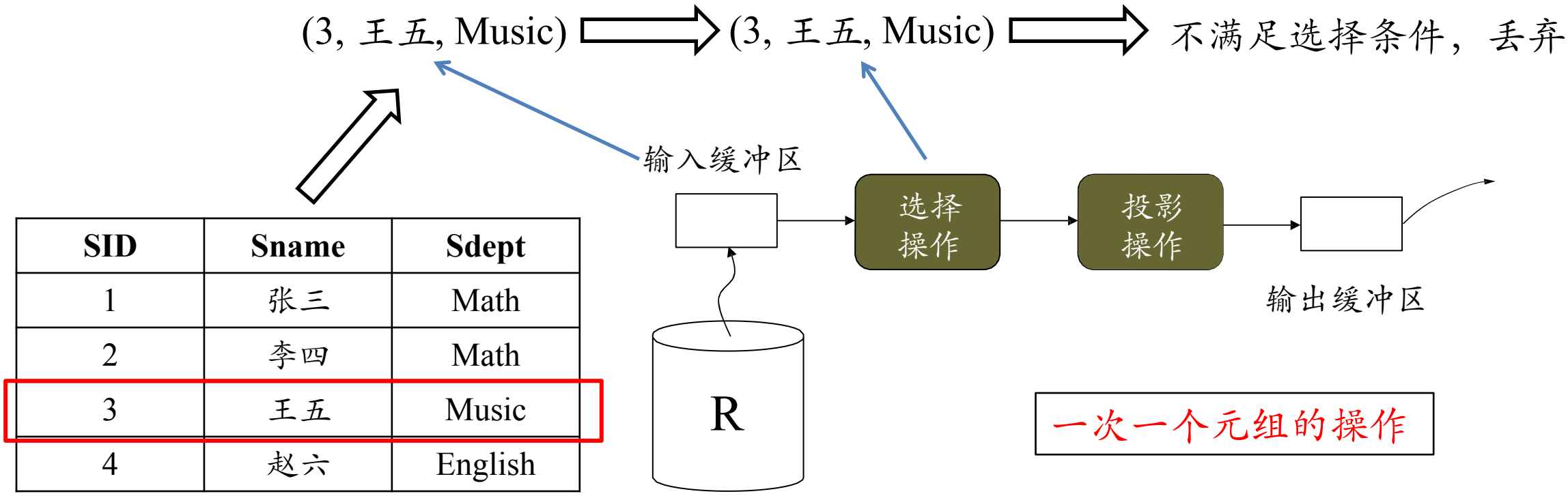


SID	Sname	Sdept
1	张三	Math
2	李四	Math
3	王五	Music
4	赵六	English

3. 查询计划的执行方案

流水线执行

流水线执行：$\prod_{Sname}(\sigma_{Sdept='Math'}Student)$, Student(SID, Sname, Sdept,)



SID	Sname	Sdept
1	张三	Math
2	李四	Math
3	王五	Music
4	赵六	English

3. 查询计划的执行方案

流水线执行（Pipelining）

流水线执行：两种模式

- Pull (拉动) 模式：每个操作向它的前序操作“要”结果来完成自己的计算
- Push(推动)模式：每个操作主动向它的后续操作“推送”自己的结果
- Pull模式为自顶向下执行；Push模式为自底向上执行。

流水线策略中操作的通用实现方法：**将每个基本操作都实现为一个“迭代器”**

- **全表扫描**
- **选择操作**
- **投影操作**
- **连接操作**
- **聚集函数计算**

迭代器：以元组/记录为单位读取前序操作的输出；三个基本操作：Open, GetNext, Close，和一个元组指针iterator.

所有关系操作可继承此迭代器，构造不同的Open()，GetNext()，Close()函数

教学内容

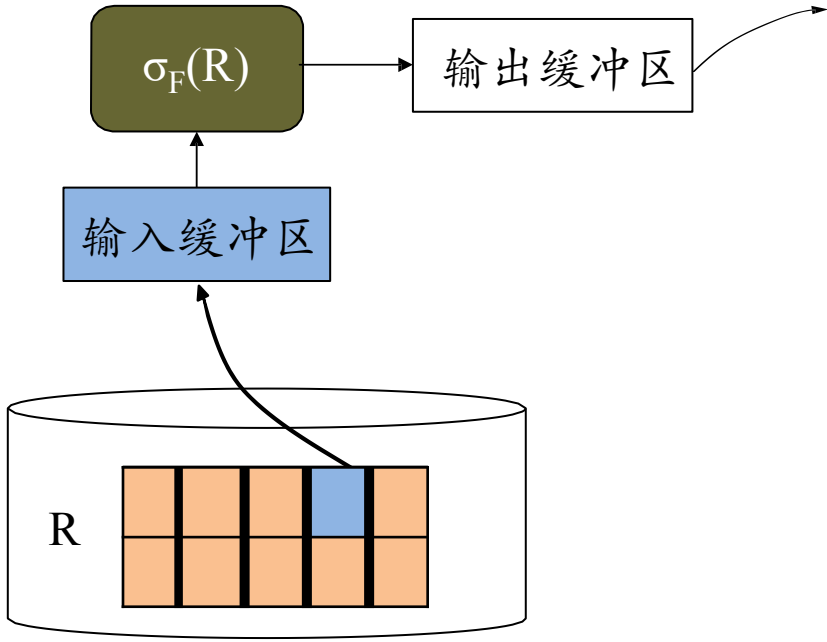
1. 查询处理概述
2. 查询代价的度量
3. 查询计划的执行策略
4. 选择操作的实现算法(σ)
5. 连接操作的实现算法($\bowtie$)

4. 选择操作的实现算法

（1）选择算法：线性搜索

$\sigma_F(R)$——selection （这里只介绍单属性等值查询方法，即 WHERE A = X）

- 选择算法是一次一个元组的操作，可以针对表中每个元组进行流水化处理。
- 如果不使用索引，则需要对输入关系进行全表扫描（即线性搜索）
- 仅需要一个磁盘块大小的输入缓冲区即可。
- 任何查询条件、索引有无、文件是否排序情况都可以使用



- 计算I/O开销(假设主文件未排序)：B_R
- 若A为候选键：$\lceil B_R/2 \rceil$（平均开销，最好1，最坏B_R）

B_R：关系R所占的磁盘块/页面数目

4. 选择操作的实现算法

(1) 基于索引的选择算法：单属性等值查询

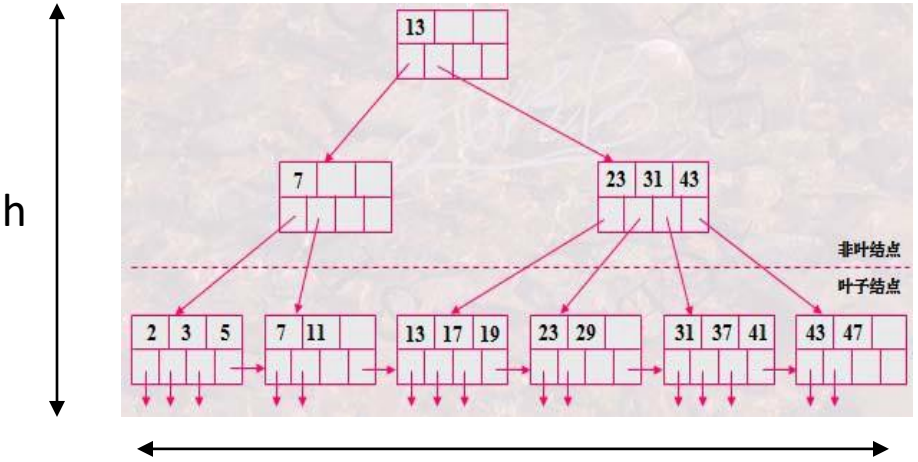
I/O代价分析

1. 若索引是聚簇索引：I/O代价 ≈ $h + \lceil B_R/V(R, K) \rceil$

- 结果元组连续存储于文件中。
- 属性K有V(R, K)个不同的值，每个值所占页数平均为 $\lceil B_R/V(R, K) \rceil$（不精确的估计）。
- 当搜索键为候选键时，I/O 代价简化为 $h+1$

2. 若索引是非聚簇索引：I/O代价 ≈ $h + \lceil T_R/V(R, K) \rceil$

- 平均有$T_R/V(R, K)$个满足查询的元组（不精确的估计）
- 每条记录都需要通过索引查找，文件里的记录是没排序的，一次只能找到一条记录
- 最坏情况下，读取每个元组都需要重新读取一个数据块（为什么会出现这种情况？）



T_R：关系R的元组数目

B_R：关系R的磁盘块数目

V(R, K)：关系R中属性K不同值的个数

h: B+树的高度

h的估计：$h \leq \log_{\lceil n/2 \rceil} N$

n为每个节点最大指针数

N为索引值的数量

4. 选择操作的实现算法

（1）基于索引的选择算法

索引应用分析示例

假设B_R=1000，T_R=20000，即R有20000个元组存放到1000个块中。a是R的一个属性，在a上有一个索引，并且考虑$\sigma_{a=0}(R)$ 操作：

- 如果R是聚簇的，且不使用索引， 平均查询代价＝500个I/O （最好1，最坏1000）
- 如果R不是聚簇的，且不使用索引， 查询代价＝1000个I/O（最好=最坏=1000）
- 如果V(R, a)=100且索引是聚簇的， 查询代价= h + 1000/100=(h+10)个I/O
- 如果V(R, a)=100且索引是非聚簇的， 查询代价= h + 20000/100=(h+200)个 I/O
- 如果V(R, a)=20000，即a是候选键，查询代价= h+20000/20000=h+1个 I/O，不管是否是聚簇的

V(R, a)：a属性在R中出现的不同值的个数

教学内容

1. 查询处理概述
2. 查询代价的度量
3. 查询计划的执行策略
4. 选择操作的实现算法(σ)
5. 连接操作的实现算法($\bowtie$)

5. 连接操作的实现方法

连接操作(Join)的实现方法

- 嵌套循环连接(Nested-Loop Join)
- 块嵌套循环连接(Block Nested-Loop Join)
- 索引连接(Indexed Join)
- 排序归并连接(Sort-Merge Join) (后面讲)
- 散列连接(Hash Join) （课上不讲，可看书）

3. 连接操作的实现算法

（1）嵌套循环连接(Nested-Loop Join)：基本逻辑

无需索引支持，适用于所有类型连接操作

$R \bowtie S$

R.A θ S.H

R	
A	B
a	1
b	2

S	
H	C
1	x
1	y
3	z

$R \bowtie_{B=H} S$

A	B	H	C
a	1	1	x
a	1	1	y

T_R：关系R的元组数目；

T_S：关系S的元组数目；

```
For  each record r in R;        //R称为外层关系(Outer Relation)
    For each record s in S;     //S称为内层关系(Inner Relation)
        if r.A θ s.H then
        { 串接 r 和 s 并存入结果关系； }
    Next j
Next I
```

复杂度: $O(T_R * T_S)$

3. 连接操作的实现算法

（2）关系物理存储相关参数

物理算法需要考虑：关系存储在磁盘上，磁盘是以磁盘块为操作单位，首先要被装载进内存 (I/O操作)，再进行元组的处理

R	
A	B
a	1
b	2

S	
H	C
1	x
1	y
3	z

T_R：关系R的元组数目

B_R：关系R的磁盘块数目

M：主存缓冲区的页数 (主存每页容量等于一个磁盘块的容量)

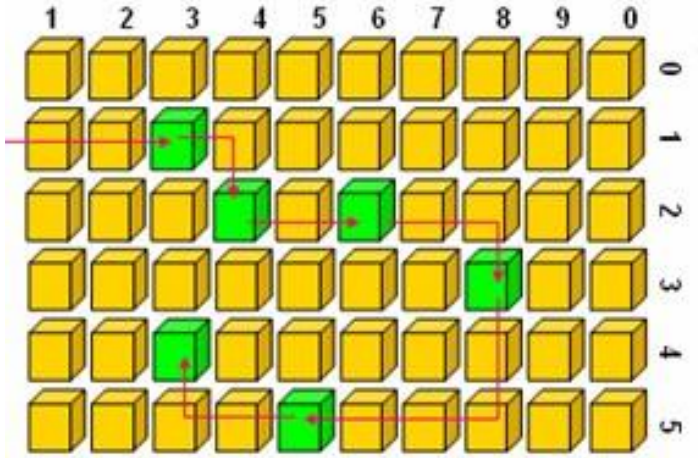
I_R：关系R的每个元组的字节数

b：每个磁盘块的字节数

R和S的连接操作结果所占用的磁盘块数最坏情况为：

$B_{R\times S} = T_R T_S (I_R + I_S)/b$ （笛卡尔积）

需要一个输出缓冲页，不断写回结果



磁盘块为I/O基本单位

3. 连接操作的实现算法

（3）Nested-Loop Join全主存实现（最好情况）

两个关系都可以完全装入主存缓冲区！

For i = 1 to B_R //注：有可能一次性读入连续的多块
    read i-th block of R;
Next I
For j = 1 to B_S //注：同上
    read j-th block of S;
Next j
For p = 1 to T_R
    read p-th record r of R;
    For q = 1 to T_S
        read q-th record s of S;
        if r.A θ s.H then
        {串接 r 和 s; 存入结果关系；}
    Next q
Next p

$R \bowtie S$

R.A θ S.H

算法概述：

将两个关系R和S均先全部读入内存，在内存中完成两个关系的连接操作

算法复杂性：

I/O次数估计为$B_R + B_S$ (暂忽略结果关系保存的I/O次数)

应用条件：

算法假定$M > B_R + B_S$，即内存能同时装下两个关系

3. 连接操作的实现算法

（4） Nested-Loop Join半主存实现算法（只有一个关系能完全装入内存）

$R \bowtie S$
$R.A\ \theta\ S.H$

For i = 1 to B_R //注：有可能一次性读入连续的多块
    read i-th block of R;
Next i
For j = 1 to B_S //注：一次读入一块
    read j-th block of S;
    For p = 1 to T_R
        read p-th record r of R;
        For q = 1 to b/I_S
            read q-th record s of S;
            if r.A θ s.H then { 串接 r 和 s；存入结果关系；}
        Next q Next p
Next j

算法概述：
先将关系R全部读入内存中，利用剩下的内存逐步读取关系S进行连接操作

算法复杂性：
I/O次数估计为$B_R + B_S$（暂忽略结果关系保存的I/O次数）

应用条件：
算法假定$B_S >= B_R$，$B_R < M$，即内存能完全装下其中一个关系

3. 连接操作的实现算法

（5） Nested-Loop Join 最坏情况: 两个关系都无法完全装入内存
采用 Block Nested-Loop： 块嵌套循环连接

内、外关系每次循环各读取一个数据块，在内存中匹配元组

磁盘I/O操作

内存操作

```
For  i = 1 to B_R
     read i-th block of R;
     For j = 1 to B_S
          read j-th block of S;
          For  p = 1 to  b/I_R
               read p-th record r of R;
               For q = 1 to b/I_S
                    read q-th record s of S;
                    if r.A θ s.H then
                    { 串接 r 和 s; 存入结果关系; }
               Next q
          Next p
     Next j
Next i
```

算法复杂性：I/O 次数估计为 $B_R + B_R \times B_S$ (暂忽略保存结果关系的 I/O 次数)

应用条件：仅需要三个内存页即可应用，一页装入R，一页装入S，一页输出

- 应选择较小的关系作为外层关系:$B_R =< B_S$

3. 连接操作的实现算法

$R \bowtie S$

$R.A\ \theta\ S.H$

（6） Block Nested-Loop 的改进：若缓冲区有M个可用页面(M>3)

```
For  i = 1 to B_R/(M-2) //注：一次读入M-2块
    read i-th subset of R into M_R;
    For j = 1 to B_S  //注：一次读入一块
        read j-th block of S into M_S;
        For  p = 1 to (M-2)b/I_R
            read p-th record r of R;
            For q = 1 to b/I_S
                read q-th record s of S;
                if r.A θ s.H then
                { 串接 r 和 s; 存入结果关系; }
            Next q
        Next p
        Next j
Next i
```

充分利用内存

算法概述：

令M_R为M-2块容量的R的主存缓冲区，M_S为1块容量的S的主存缓冲区，还有1块作为输出缓冲区。

每次读入M-2块的R关系，而不是1块。

每次仍然读入1块S关系。

应选择较小的关系作为外层关系:B_R=< B_S

算法复杂性：

I/O次数估计为$B_S \times (B_R/(M-2))+B_R$ (暂忽略结果关系保存的I/O次数)

应用条件：

算法假定B_S >= M，B_R >= M。

3. 连接操作的实现算法

$R \bowtie S$

R.A = S.H

（7）索引嵌套循环连接算法(Index Nested-Loop Join)

如果…

- 内层关系S的连接属性H上建有索引（如B+Tree）
- 且连接为等值或者自然连接

则可以利用索引帮助查找S中匹配的元组，即Index Nested-Loop Join

For i = 1 to B_R

read i-th block of R; \\无内层关系读取，直接用索引搜索

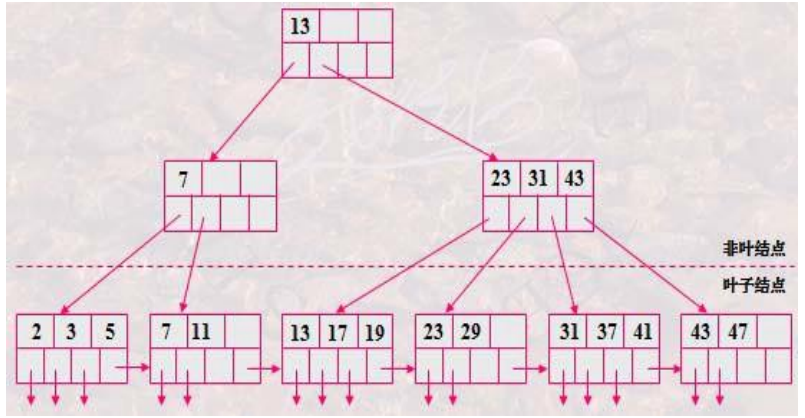
For p = 1 to b/I_R

read p-th record r of R;

Search for record s of S with s.H = r.A using index on H

\\此步骤等于进行一个Selection操作，前面讲过

串接 r 和每一个返回的 s；存入结果关系;



B+树有序索引

算法复杂性：

I/O 次数估计为（最坏情况）

$B_R+T_R \times C$

C为S上一次索引等值查询的开销

(暂忽略结果关系保存的I/O次数)

3. 连接操作的实现算法

总结

- ✓ Nested-Loop 全主存算法

  要求内存能够完全装载两个关系；算法复杂性：$B_R + B_S$ ①

- ✓ Nested-Loop 半主存算法

  要求内存能够完全装载一个关系；算法复杂性：$B_R + B_S$ ②

- ✓ Block Nested-Loop 块嵌套循环连接

  3块内存即可； 算法复杂性：$B_R + B_R \times B_S$ ④

- ✓ Block Nested-Loop改进算法

  >3块内存即可；算法复杂性：$B_R(B_S/(M-2)) + B_S$ ③

I/O Cost排序从低到高

- ✓ Index Nested-Loop：索引嵌套循环连接

  3块内存即可；算法复杂性：$B_R + T_R \times C$ ？？

第10讲 查询处理(下)

教学内容

1. 查询处理概述
2. 查询代价的度量
3. 查询计划的执行方法
4. 选择操作的实现算法（一趟算法）
5. 连接操作的实现算法（一趟算法）

6. 外存排序算法 （两趟/多趟算法），

7. 归并排序连接

8. 其他操作

1. 为什么需要两趟算法

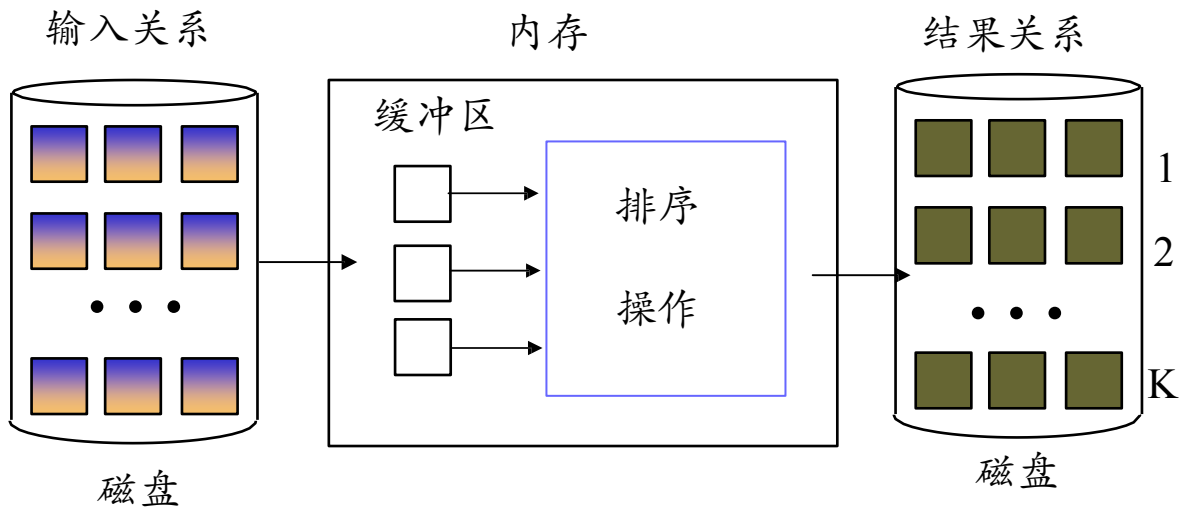
（1）内排序和外排序

数据排序在数据库系统中有重要作用

- SQL查询可能会指明对查询结果进行排序
- 当输入关系已排序时，关系运算中的一些运算（如连接运算）能够得到高效实现

内排序：待排序的数据可一次性地装入内存，即排序者可以完整地看到和操纵所有数据。内存中数据的排序算法：插入排序算法、选择排序算法、冒泡排序算法等

外排序：待排序的数据不能一次性装入内存，即排序者不能一次完整 地看到和操纵所有数据，需要将数据分批装入内存分批处理的排序问题 。如内存--2GB；待排序数据--7GB, 10GB, 100GB，需要使用硬盘等外部存储设备进行排序

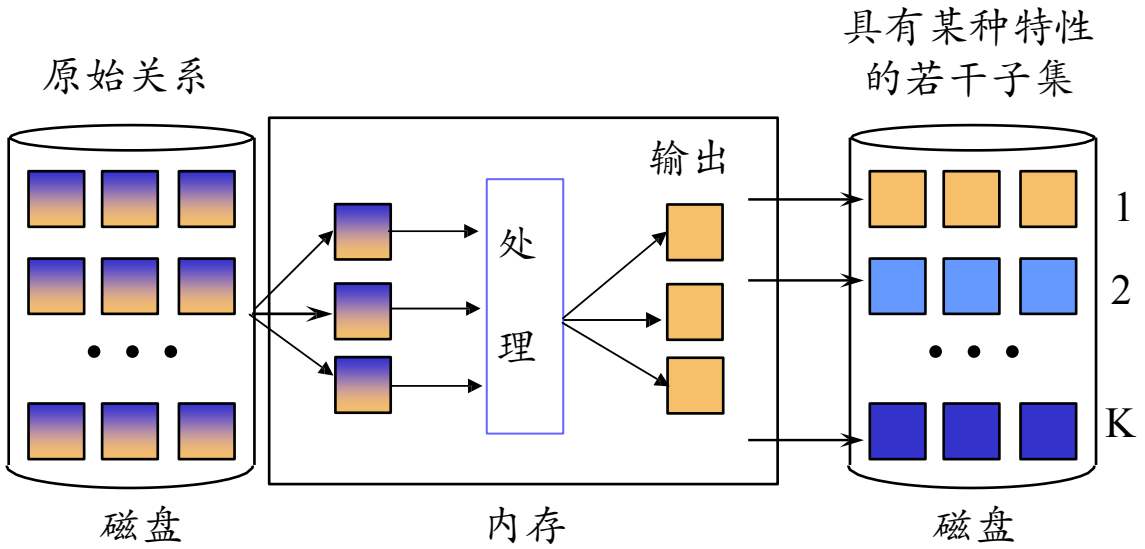


1. 为什么需要两趟算法

（2）两趟算法的基本思想

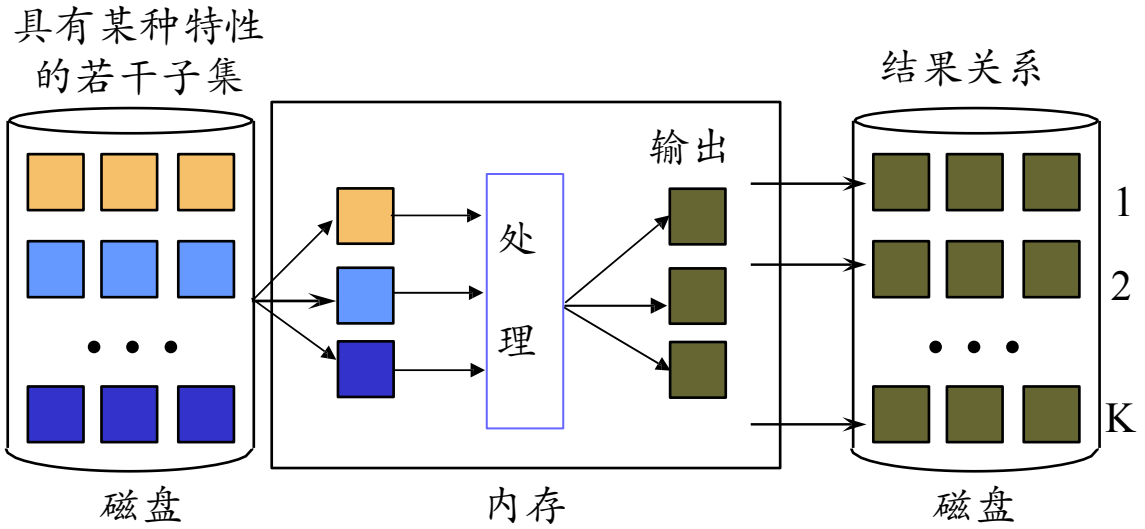
第一趟

划分子集，并使子 集具有某种特性， 如有序或相同散列 值等



第二趟

处理全局性内容的 操作，形成结果关 系。如多子集间的 归并排序，相同散 列值子集的操作等



2. 两阶段多路归并算法

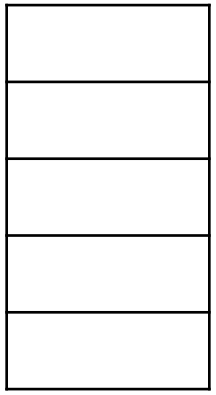
（1）外排序问题分析

外排序问题分析（仅介绍思想，忽略一些细节），假设:

- 可用内存大小：共M=5块, 每块可装载f_R=5个元组
- 待排序关系R：T_R=100个元组, 每块5个元组，共占用B_R=20块

问题：B_R块的数据怎样利用M块的内存进行排序?

内存缓冲区块M=5



待排序关系R， B_R=20

199 301 201 331 188	99 102 101 222 188	11 86 60 80 75	70 82 58 43 05	28 15 90 08 32
312 197 187 911 409	512 97 187 111 109	03 45 35 30 27	06 20 08 10 09	18 08 24 04 55
172 666 887 468 777	172 144 887 268 103	80 31 72 10 52	50 42 38 34 78	16 19 29 13 62
99 198 199 200 723	96 98 99 100 421	42 70 68 09 62	60 45 70 38 35	25 20 15 64 08

2. 两阶段多路归并算法

两阶段多路归并排序算法TPMMS

（3）外排序问题TPMMS算法

基本排序策略

B_R块数据可划分为N个子集合，使每个子集合的块数小于等于内存可用块数，即

$$B_R/N \leq M$$

每个子集合都可装入内存并采用内排序算法排好序并重新写回磁盘

问题转化为：N个已排序子集合的数据怎样利用内存进行总排序?

子集合1	199 301 201 331 188	99 102 101 222 188	11 86 60 80 75	70 82 58 43 05	28 15 90 08 32
子集合2	312 197 187 911 409	512 97 187 111 109	03 45 35 30 27	06 20 08 10 09	18 08 24 04 55
子集合3	172 666 887 468 777	172 144 887 268 103	80 31 72 10 52	50 42 38 34 78	16 19 29 13 62
子集合4	99 198 199 200 723	96 98 99 100 421	42 70 68 09 62	60 45 70 38 35	25 20 15 64 08

B_R = 20
N = 4
M = 5

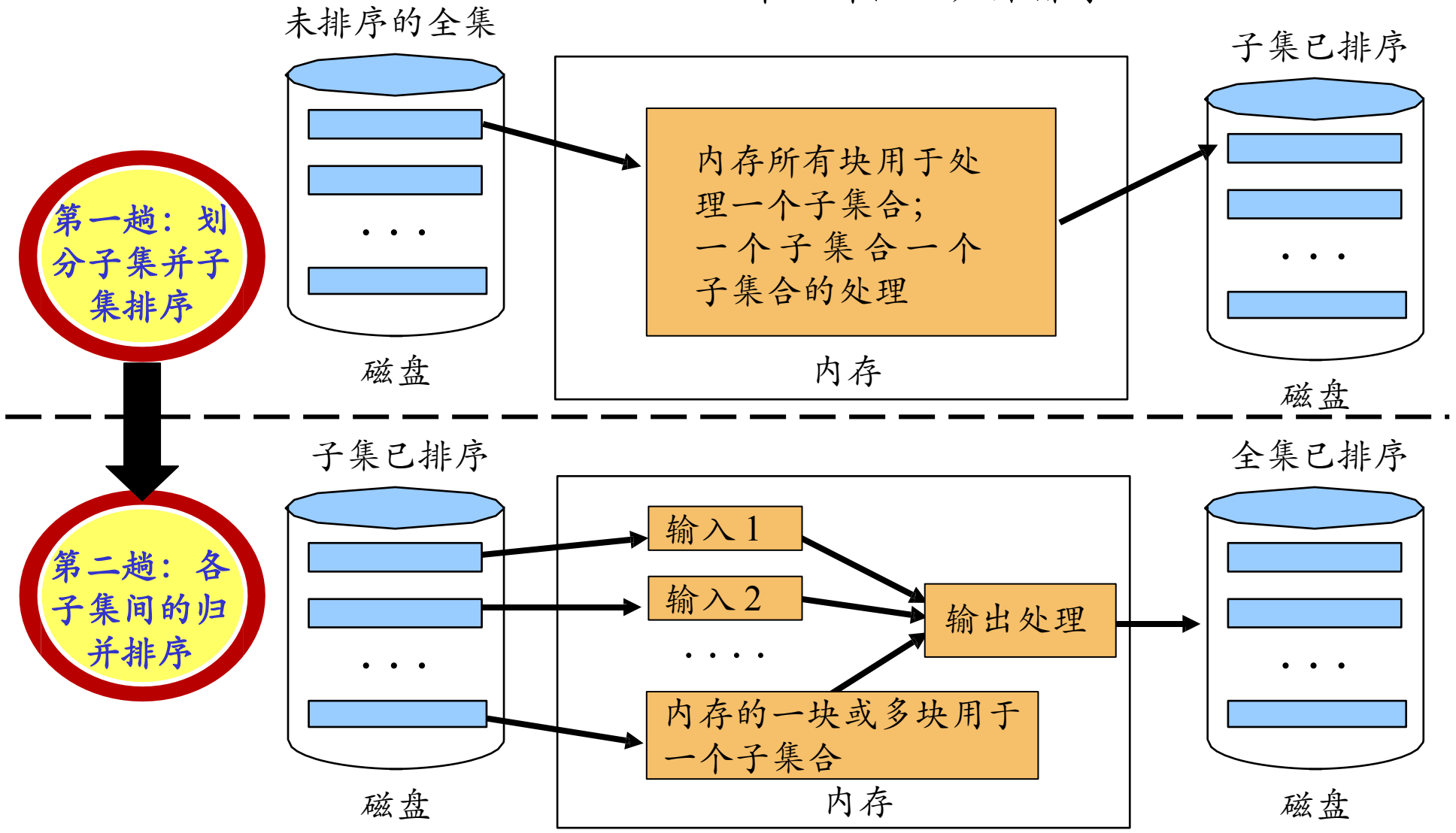
20块分为4个子集合，每组5块，共25个记录

2. 两阶段多路归并算法

（2）实现步骤

两阶段多路归并排序TPMMS

- 第一阶段：创建归并段
- 第二阶段：归并排序



2. 两阶段多路归并算法

（3）创建归并段

将关系R划分为 $\lceil B_R/M \rceil$ 个归并段

创建归并段

for R的每M块 do

将这M块读如内存的M页

对这M页中的元组按排序键进行排序，形成归并段

将该归并段写入文件

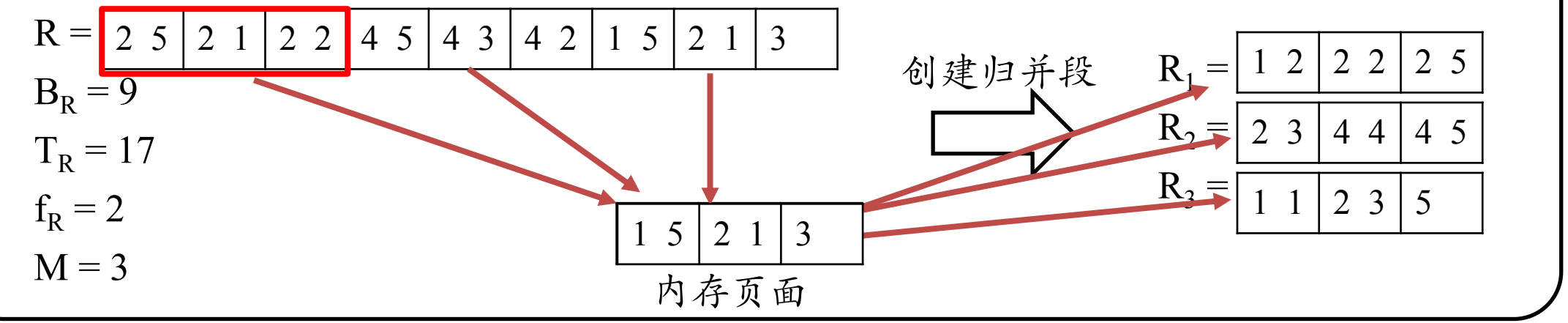
M：可用内存页数

B_R：关系R占用的磁盘块数

T_R：关系R包含的元组数

f_R：每个块最多存储的元组数量

Example（外存多路归并排序）



2. 两阶段多路归并算法

（4）多路归并

将$\lceil B_R/M \rceil$个归并段中的元组进行归并

多路归并

将每个归并段的第一块读入缓冲池中的一页

repeat

找出所有输入内存也中的最小的排序键值v

将所有排序键值等于v的元组写入输出缓冲区

任意输入缓冲页中的元组若归并完毕，则读入其归并段的下一页

until 所有归并段都已归并完毕

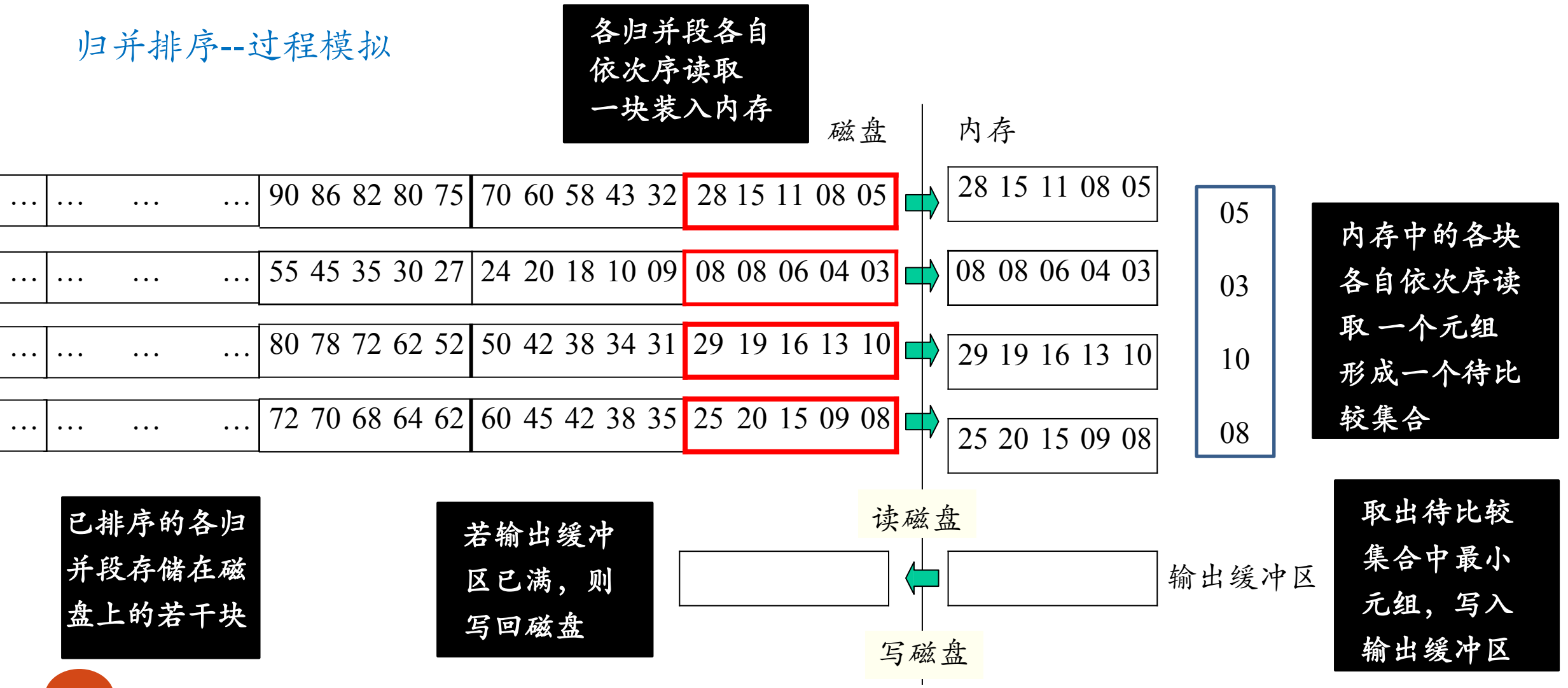
2. 两阶段多路归并算法

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$

（4） 多路归并

归并排序--过程模拟



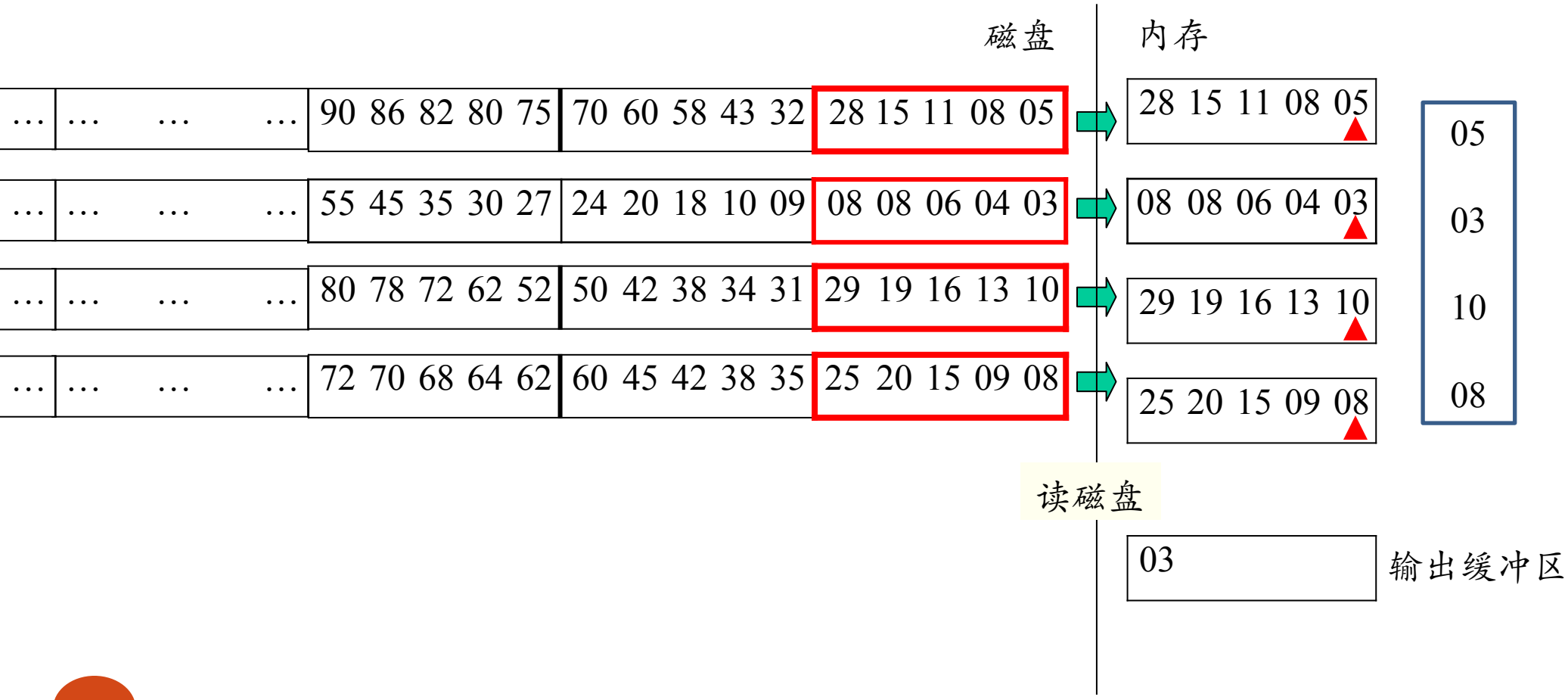
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



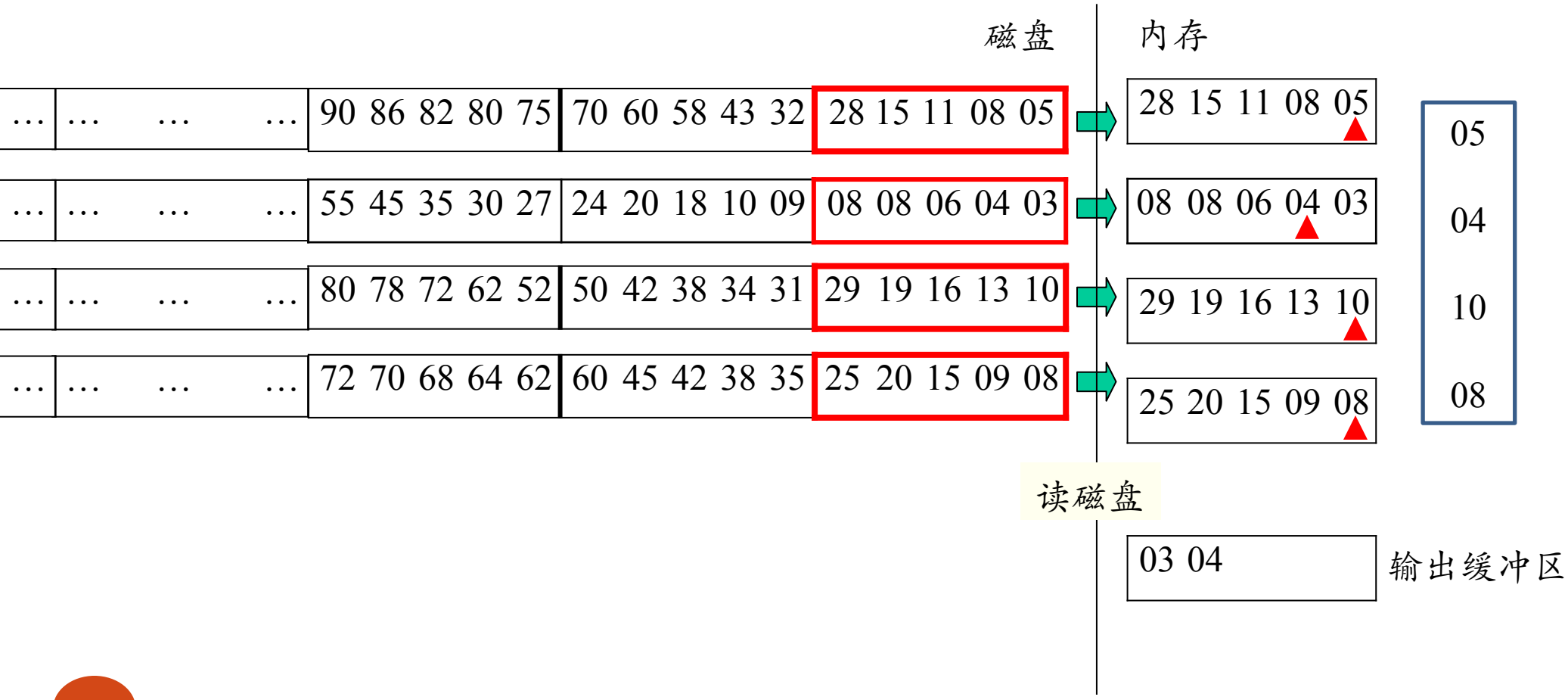
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



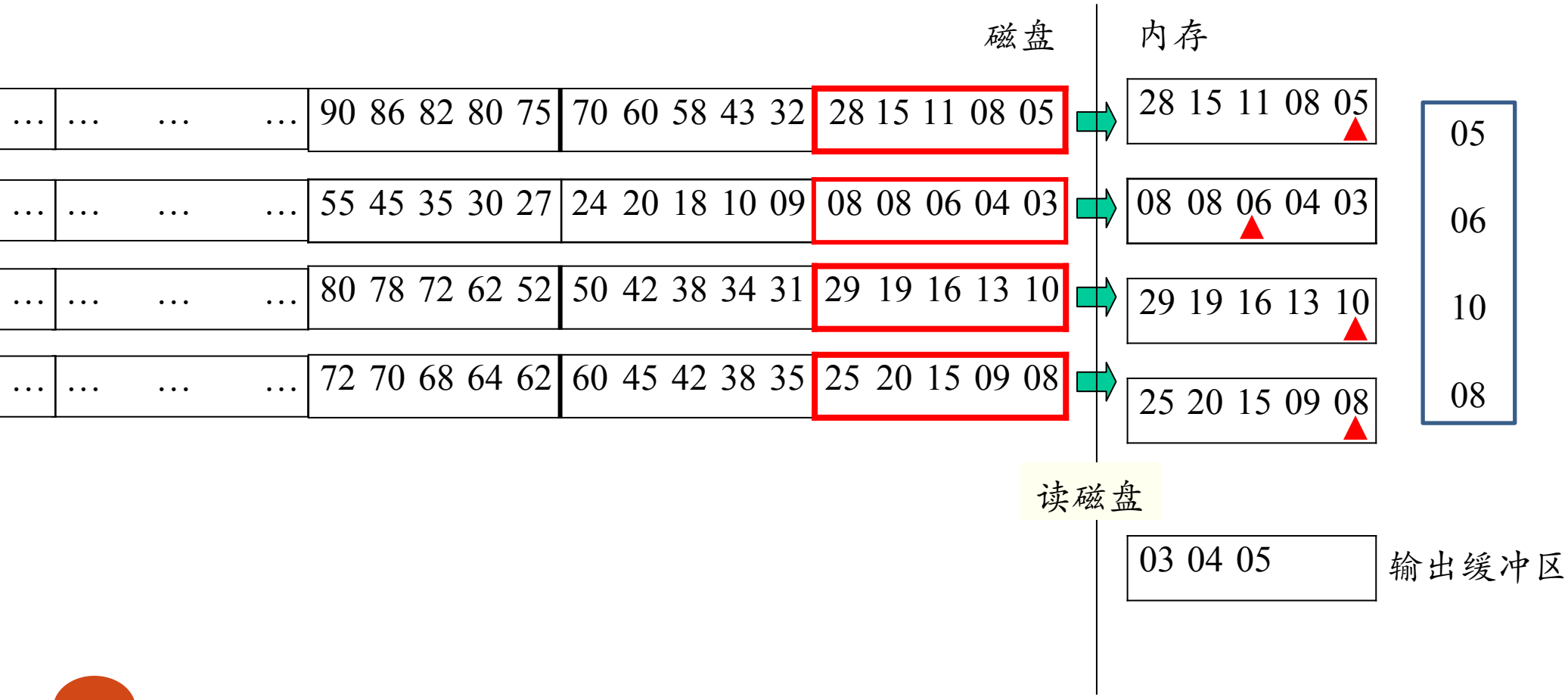
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



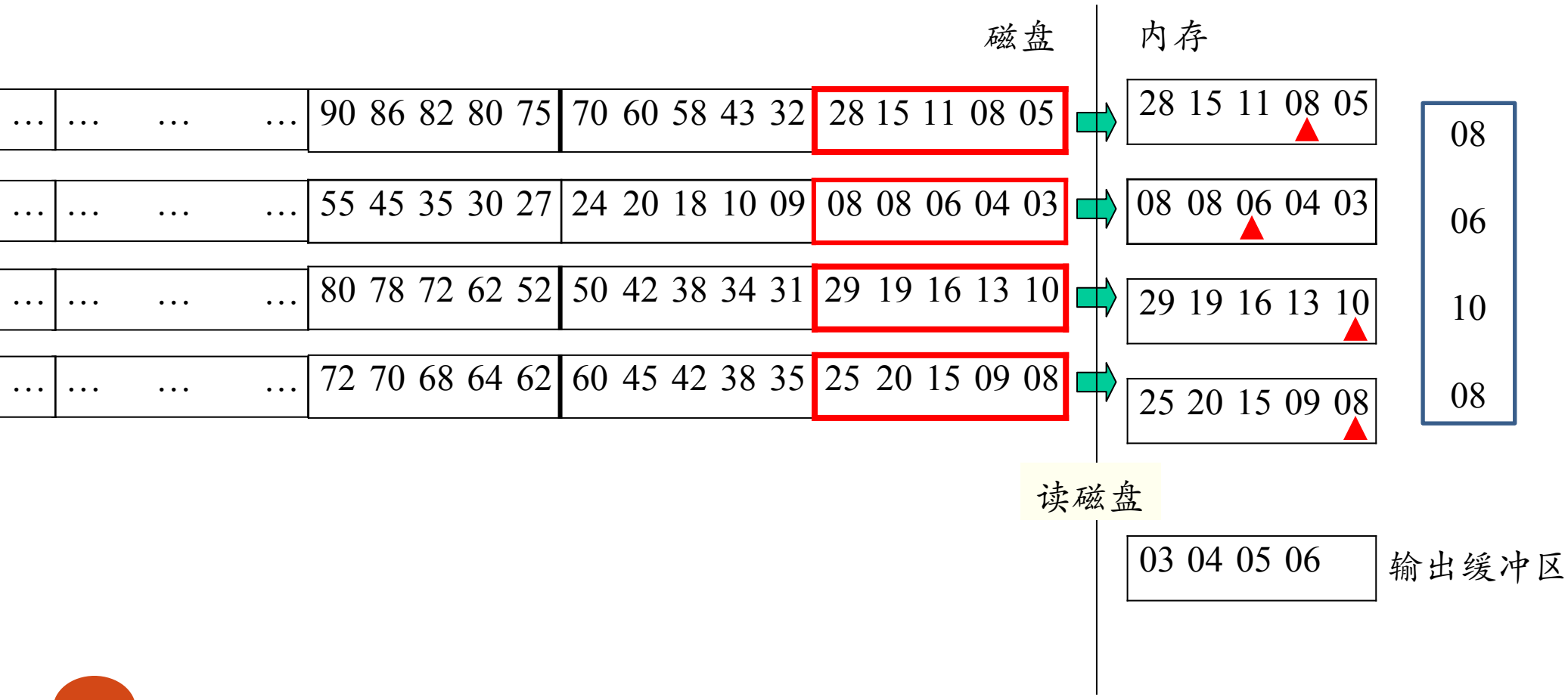
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



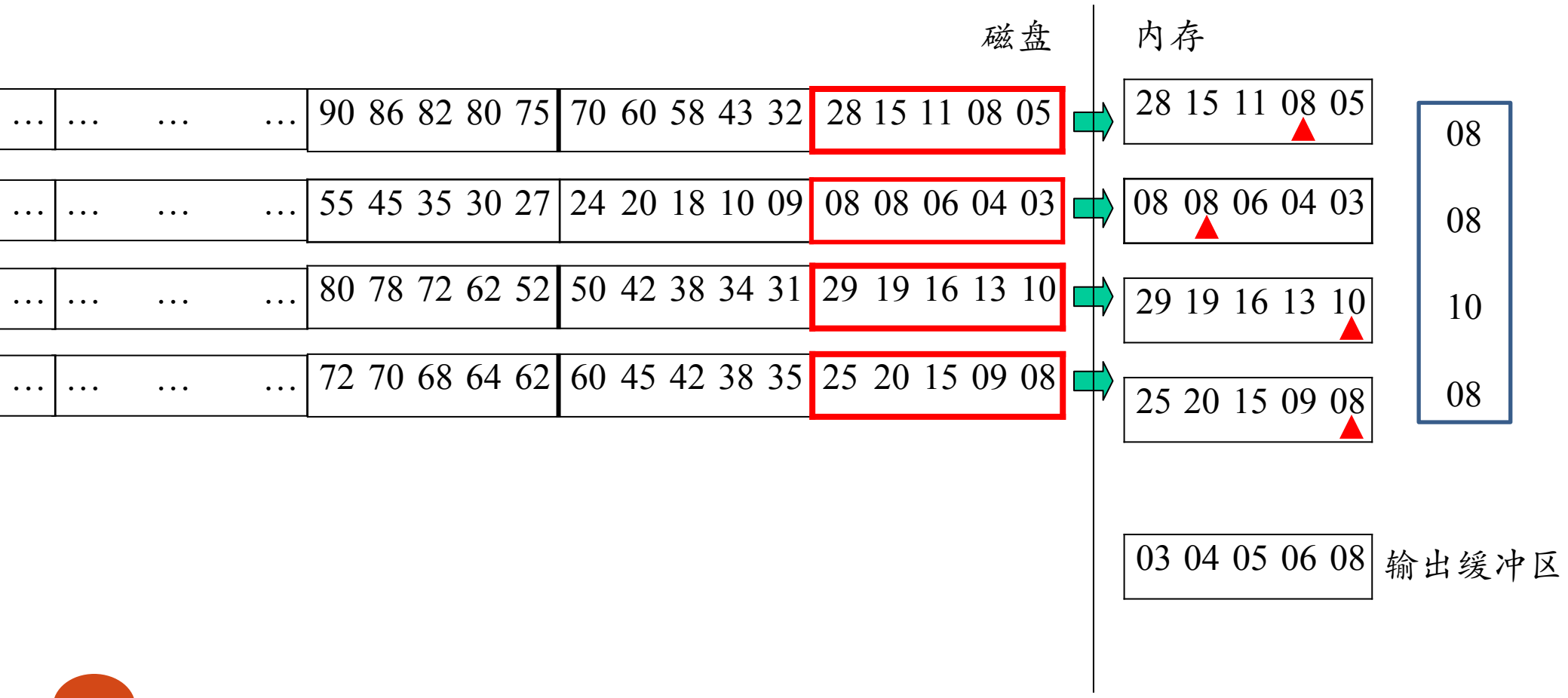
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



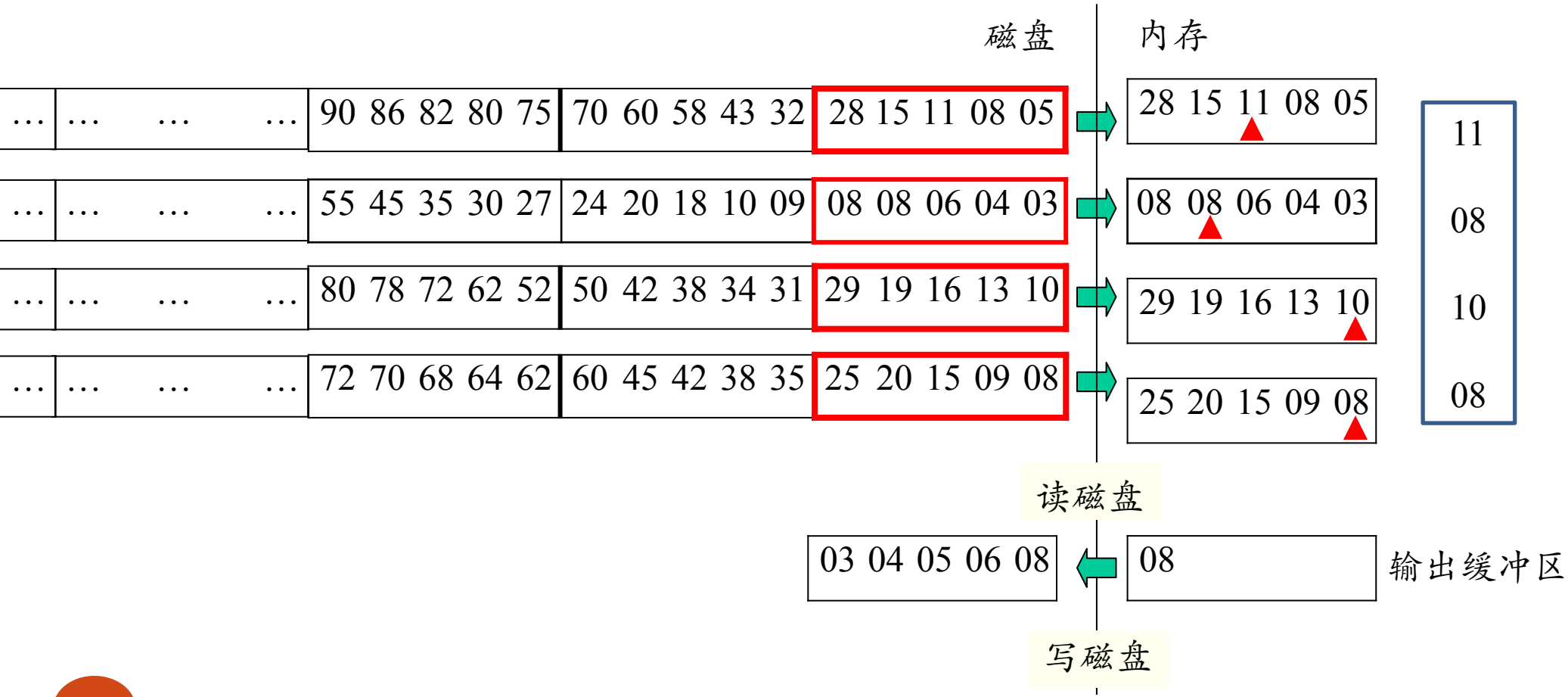
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



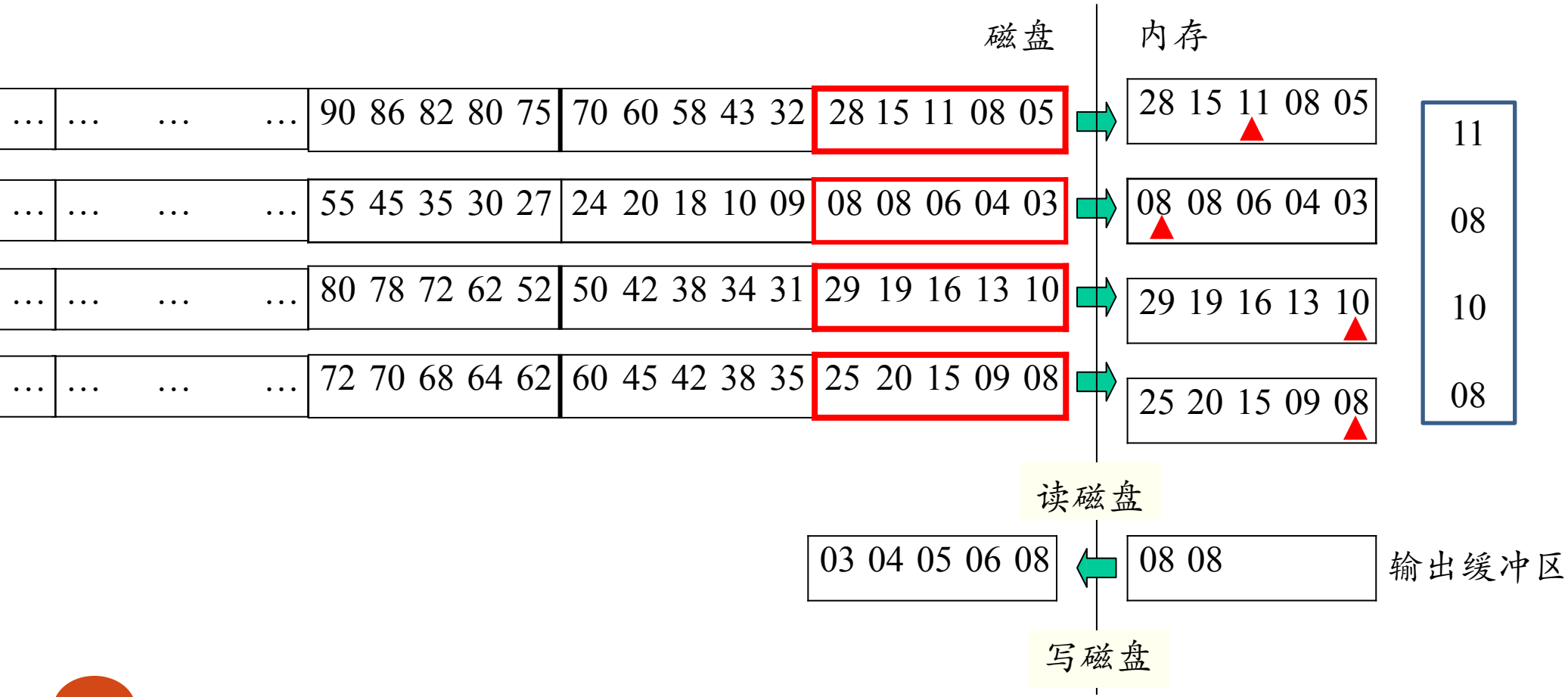
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



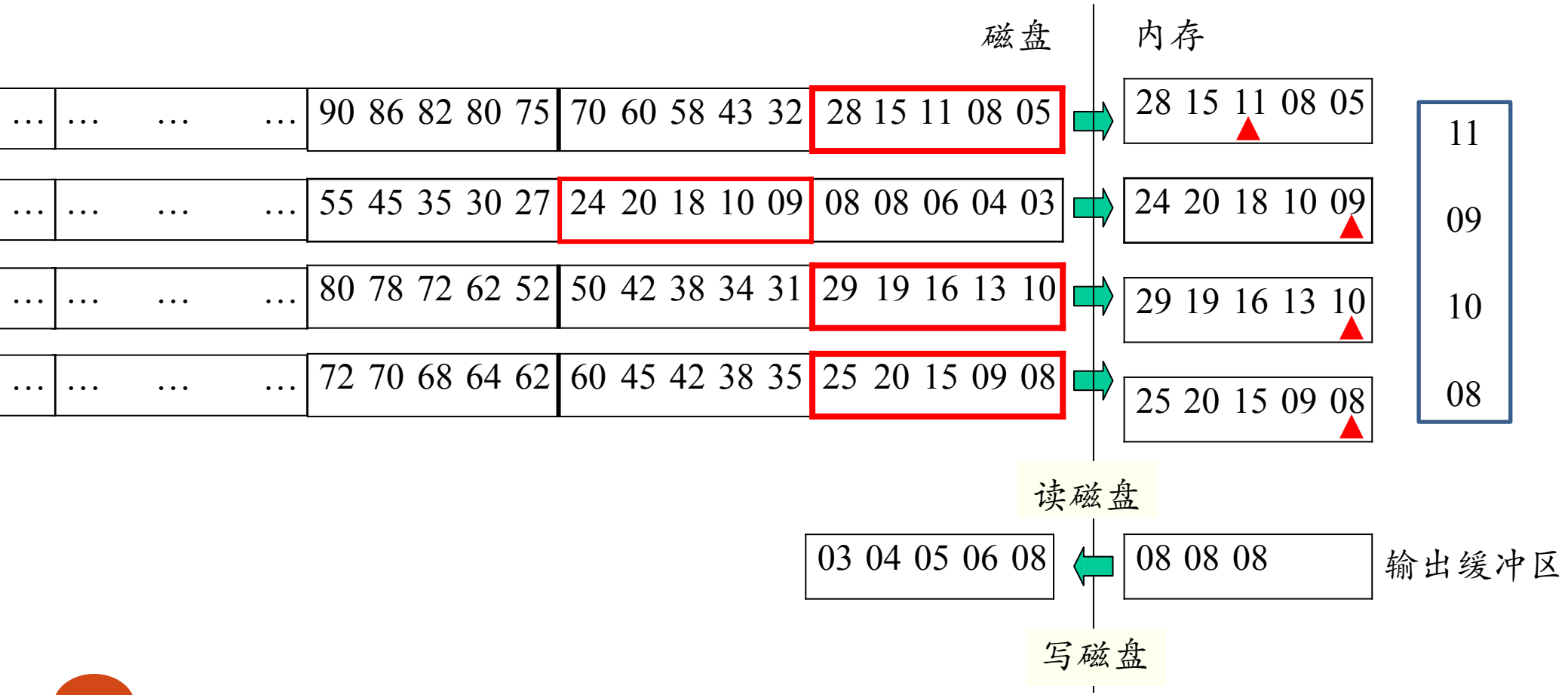
2. 两阶段多路归并算法

（4） 多路归并

归并排序--过程模拟

可用内存页数：M = 5

每个块最多存储的元组数量$f_R = 5$



2. 两阶段多路归并算法

（5）算法分析

I/O代价：$3B_R$

- 在创建归并段时，R的每块读1次，合计B_R次I/O
- 将每个归并段写入文件，合计B_R次I/O
- 在归并阶段，每个归并段扫描1次，合计B_R次I/O
- 若考虑最终结果的写回，则I/O代价为$4B_R$

可用内存页数要求：$B_R \leq M \times (M-1)$

- 子集合数＜M，即最多M个归并段
- 子集合块数≤M，即每个归并段不超过M页

2. 两阶段多路归并算法

(6) 多阶段多路归并算法

更大规模数据集的排序问题—多趟/多阶段

- 可用内存大小: 共M = 3块
- 待排序关系R: 共占用$B_R = 30$块

基本策略

（1） 30块的数据集→10个子集合，每个子集合3块，排序并存储

（2）10个已排序子集合分成5个组：每个组2个子集合，分别进行二路归并，则可得到5个排好序的集合

（3）5个集合再分成3个组：每个组2个子集，剩余一个单独1组，分别进行二路归并，可得3个排好序的集合；再分组，再归并得到2个排好序的集合；再归并便可完成最终的排序

总开销= $B_R \times (2\lceil \log_{M-1}(B_R/M) \rceil + 1)$

思考题

假如内存共有8块，问如何排序有70块的数据集呢？设计具体算法，并指出磁盘读写次数是多少。最少需要几个阶段？最少需要多少次磁盘读写操作？

教学内容

1. 为什么需要两趟算法
2. 两阶段多路归并排序算法
3. 归并排序连接算法
4. 其他运算

3. 归并排序连接算法

$R \bowtie S$

R.Y = S.Y

（1）归并排序连接（Merge-Join）

算法

// 创建归并段

将R划分为$\lceil B_R/M \rceil$个归并段，并按R.Y排序

将S划分为$\lceil B_S/M \rceil$个归并段，并按S.Y排序

// 归并

读入R和S的每个归并段的第1页

repeat

　　找出输入缓冲区中元组Y属性的最小值y

　　for R中满足R.Y = y的元组r do

　　　　for S中满足S.Y = y的元组s do

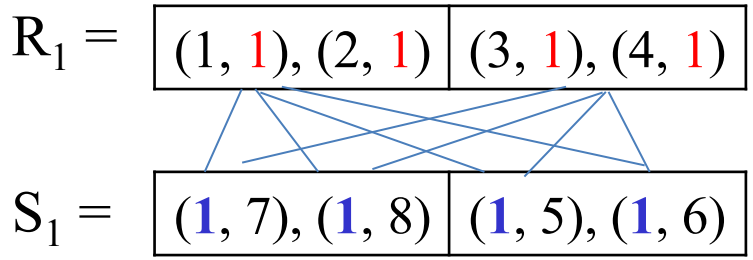
　　　　　　连接r和s，并将结果写入输出缓冲区

　　任意输入缓冲页中的元组若归并完毕，读入其归并段的下一页

until R或S的所有归并段都已归并完毕

补充说明：该算法是一个简化版，它假设对每一个y值，所有R.Y=y的元组和所有S.Y=y的元组，至少一方只占一个数据块，比如Y在某一方是候选键时。否则算法不正确。

如果S和R中的具有相同y值的元组均超过一个数据块，则需要更多的内存并修改算法（如下例）。



3. 归并排序连接算法

$R \bowtie S$

$R.Y = S.Y$

（1）归并排序连接（Merge-Join）

算法

将R按R.Y进行归并排序

将S按S.Y进行归并排序

读入排序后R和S各自的第1页

repeat

找到当前缓冲区内未处理元组中Y列的最小值y

将含有R.Y=y的R元组对应页面全部读入内存缓冲区

for R中满足R.Y = y的元组r do

for S中满足S.Y = y的元组s do

连接r和s，并将结果写入输出缓冲区

任意输入缓冲页中的S页，若元组全部处理完毕则读入S的下一页将其替换

until R或S的所有页面都已处理完毕

补充：一般的归并排序连接算法，前提假设对任何y，保证所有R. Y=y或所有S. Y=y的元组可以全部放入内存缓冲区。

3. 归并排序连接算法

$R \bowtie S$

R.Y = S.Y

（2）运行示例

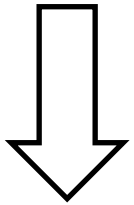
Example（外存多路归并排序）

R(X, Y) =	(1, **1**), (5, **4**)	(3, **2**), (3, **1**)	(6, **3**), (2, **1**)	(4, **2**), (8, **5**)	(4, **1**), (3, **4**)
S(Y, Z) =	(**2**, 6), (**1**, 7)	(**1**, 8), (**5**, 9)	(**5**, 3), (**2**, 5)	(**3**, 1), (**2**, 7)	(**3**, 7), (**4**, 9)

$f_R = 2$

M = 5

二路归并



创建归并段

R_1 =	(1, 1), (2, 1)	(3, 1), (4, 1)	(3, 2), (4, 2)	(6, 3), (3, 4)	(5, 4) , (8, 5)
S_1 =	(**1**, 7), (**1**, 8)	(**2**, 5), (**2**, 6)	(**2**, 7), (**3**, 1)	(**3**, 7), (**4**, 9)	(**5**, 3), (**5**, 9)

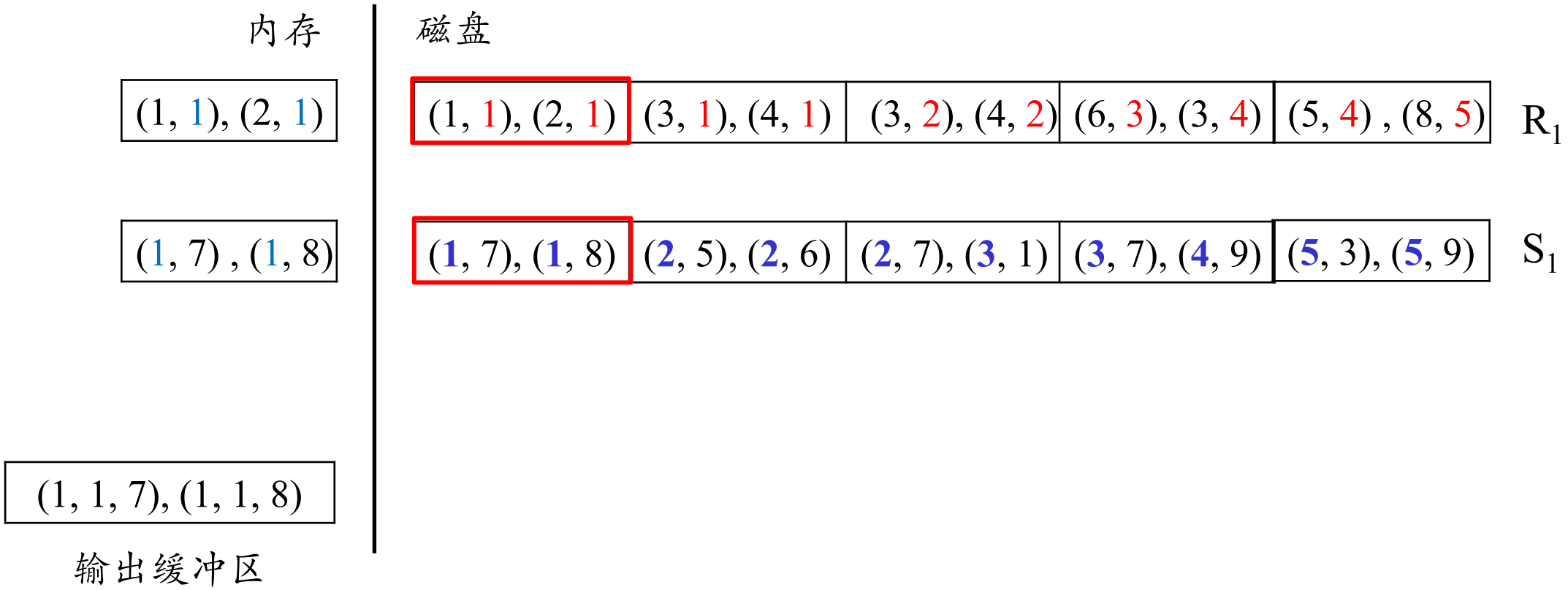
3. 归并排序连接算法

$R \bowtie S$ R(X, Y)

R.Y = S.Y S(Y, Z)

（2）运行示例

Example（外存多路归并排序）（续）

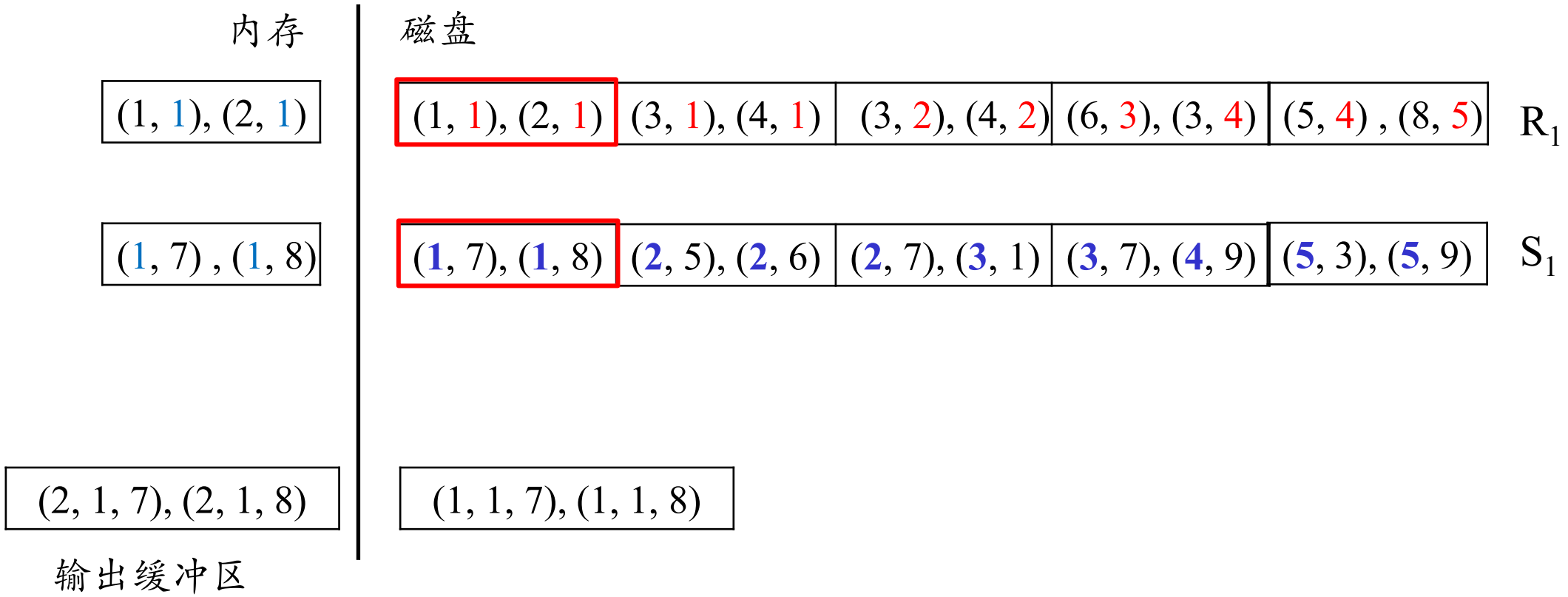


3. 归并排序连接算法

$R \bowtie S$ R.Y = S.Y　　R(X, Y)　S(Y, Z)

（2）运行示例

Example（外存多路归并排序）（续）



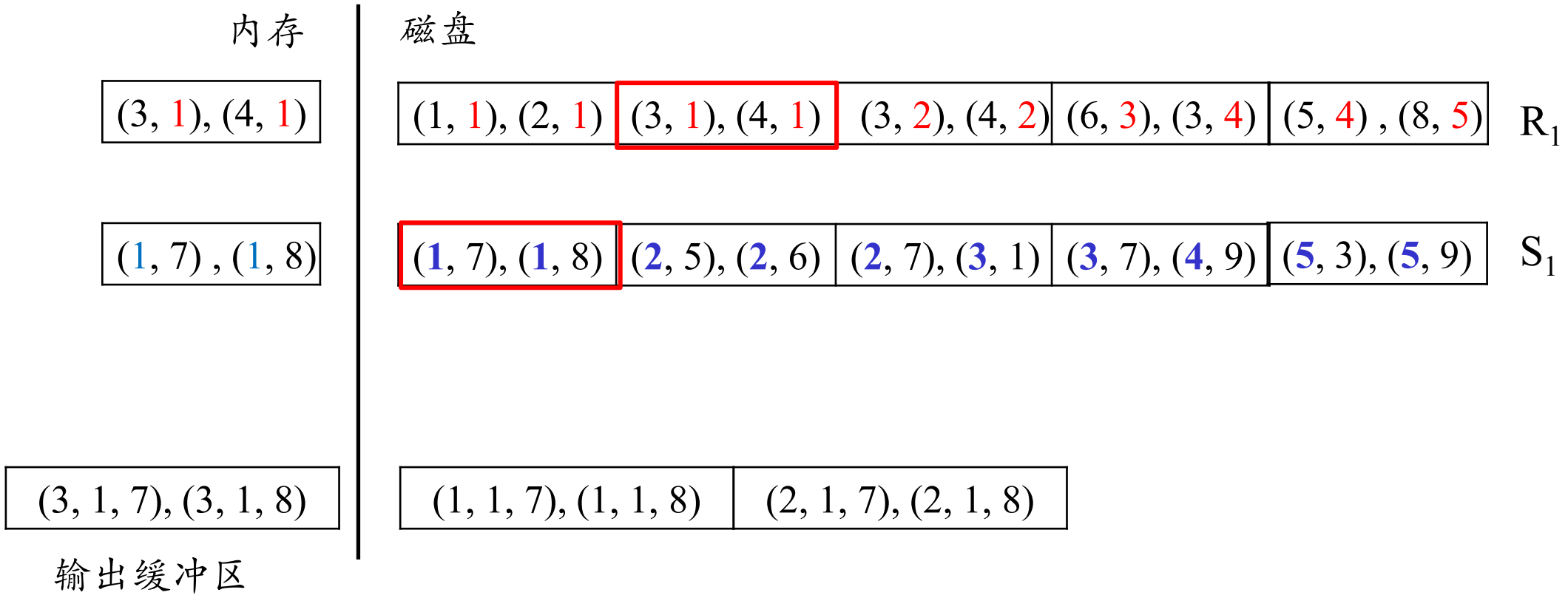
3. 归并排序连接算法

（2）运行示例

$R \bowtie S$　　R(X, Y)

R.Y = S.Y　　S(Y, Z)

Example（外存多路归并排序）（续）



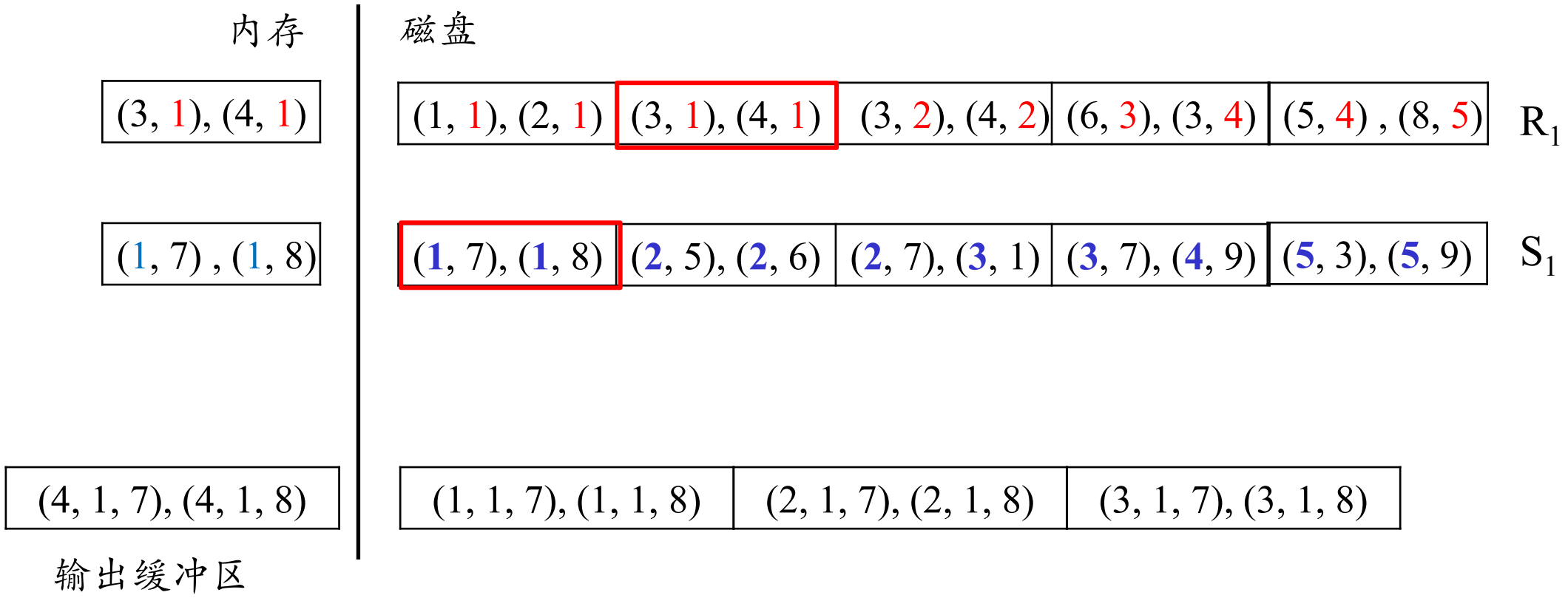
3. 归并排序连接算法

（2）运行示例

$R \bowtie S$　　R(X, Y)

R.Y = S.Y　　S(Y, Z)

Example（外存多路归并排序）（续）



3. 归并排序连接算法

（3）算法分析

I/O代价：$3B_R+3B_S$ （不考虑结果写回的代价）

- 在对R创建归并段时，R的每块读1次，合计B_R次I/O
- 将R的归并段写入文件，合计B_R次I/O
- 在对S创建归并段时，S的每块读1次，合计B_S次I/O
- 将S的归并段写入文件，合计B_S次I/O
- 在归并阶段，对R和S的每个归并段扫描1次，合计B_R+B_S次I/O

可用内存页数要求：$B_R+B_S \leq M*(M-1)$

- 基于简化版算法，假设任意时刻含有R.Y=y或S.Y=y的元组不超过一个数据块。
- 如果具有重复值的元组超过一个数据块，则需要更大的内存空间。

教学内容

1. 为什么需要两趟算法
2. 两阶段多路归并排序算法
3. 归并排序连接算法
4. 其他运算

4. 其他运算实现算法

去除重复(DISTINCT)

- 排序方法：排序时等价元组相互邻近，删除其他副本只留一个元组副本即可。对于外部归并排序，可在创建归并段时去除重复元组，减少块的传输次数，剩余的可在归并时去除
- 散列方法
- 由于去除重复代价较大，SQL查询语句要去用户显式指明需要去除重复

投影

- 首先对每个元组作投影，所得结果关系可能有重复元组，然后去除重复记录。若投影列表中属性含有关系的码，则结果中不会有重复元组

集合运算

- 并、交、差：首先对两个关系进行排序，然后对每个已排序的关系扫描一次。并运算中，只保留一个相同元组；交运算中，只保留同时出现的元组；差运算中，保留一个关系中不属于另一个关系的元组