

# 数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

# 第九章 关系查询处理及查询优化

*Principles of Database Systems*

# 第九章 关系查询处理及查询优化

## 9.1 关系数据库系统的查询处理

## 9.2 关系数据库系统的查询优化

## 9.3 代数优化

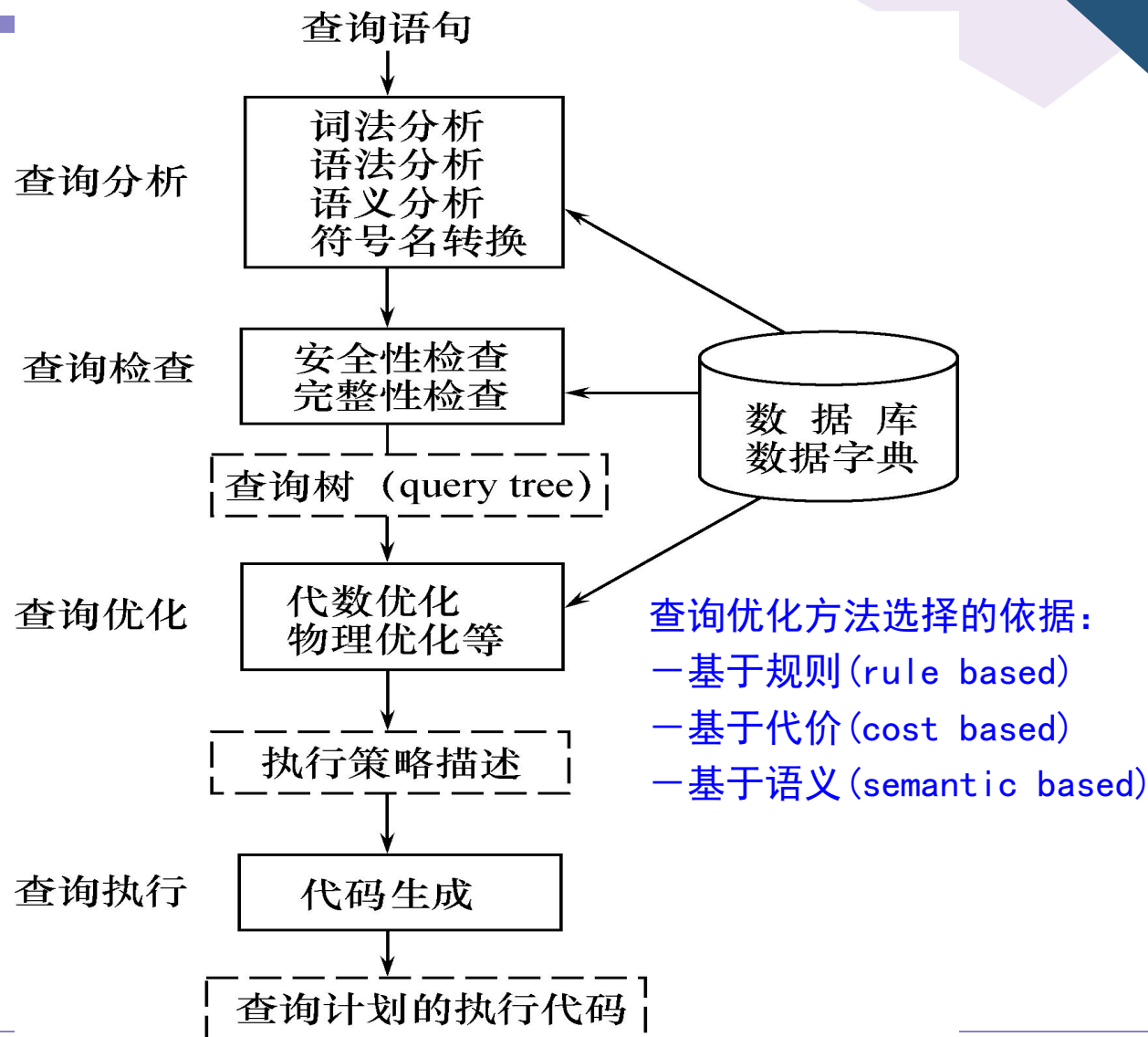
## 9.4 物理优化

## 9.5 小结

# 9.1 关系数据库系统的查询处理

## 9.1.1 查询处理步骤

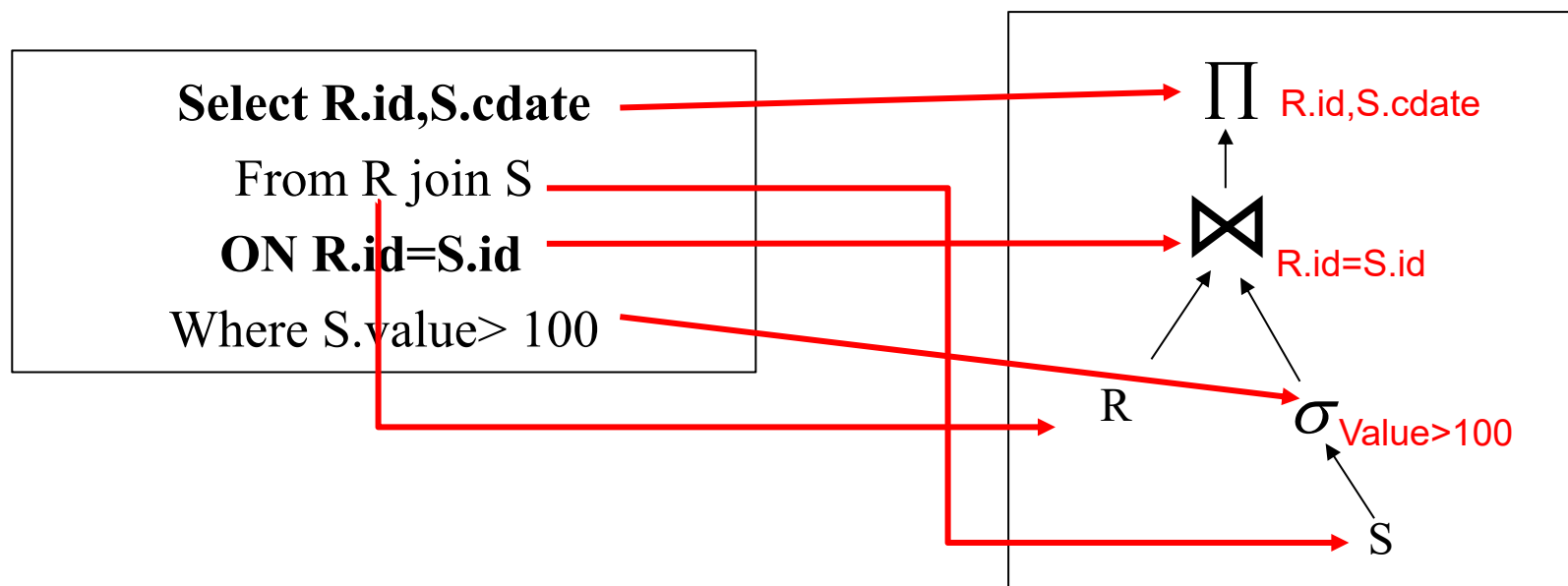
1. 查询分析
2. 查询检查
3. 查询优化
4. 查询执行



# 关系数据库系统的查询处理流程

## □ 查询计划:

- 操作算子以树的形式进行组织
- 数据流从叶子结点流向根节点
- 根节点的输出是查询的结果



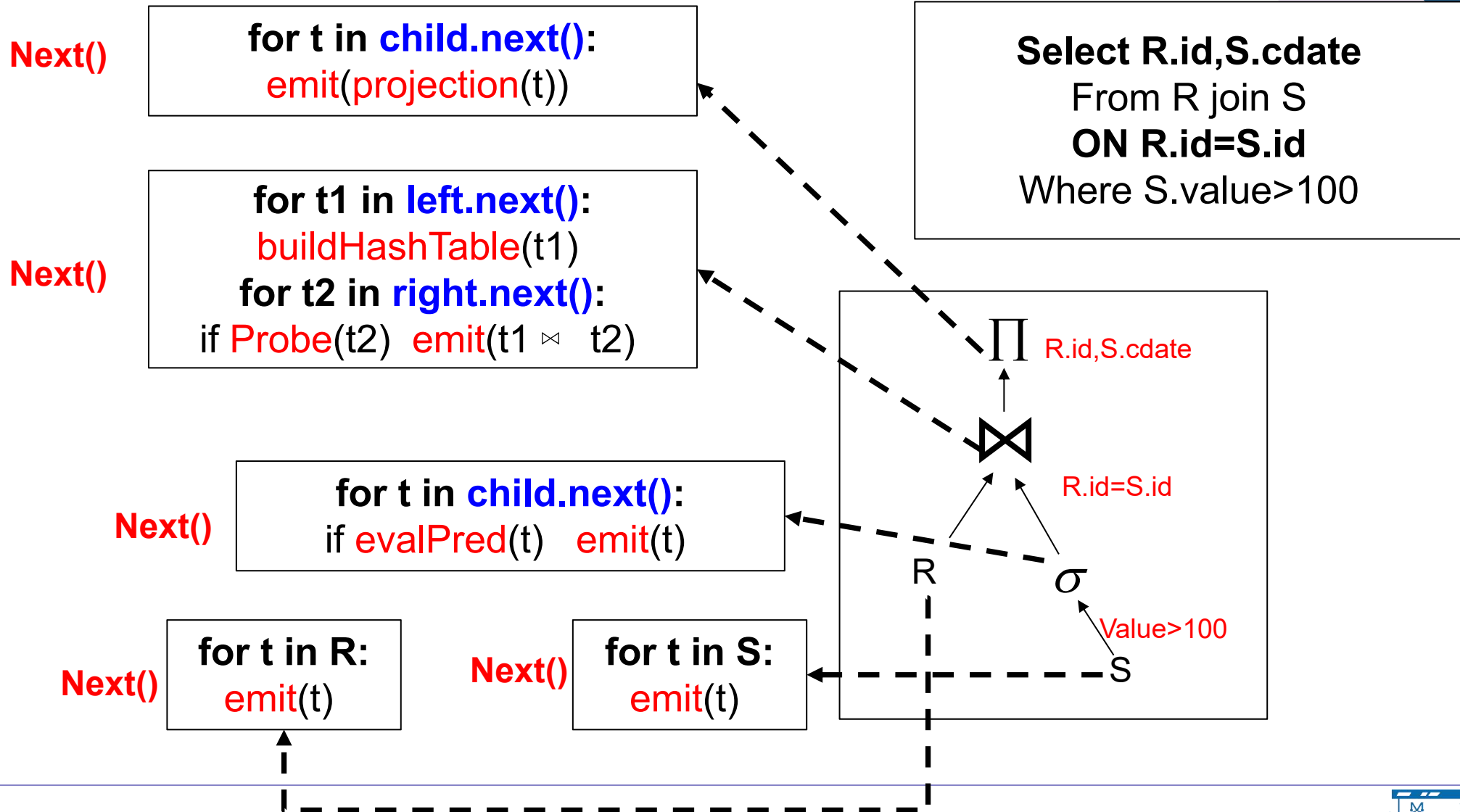
## 9.1.2 查询处理模型

- 处理模型定义系统如何执行一个查询计划，不同的任务负载对应不同的权衡结果。
  - 迭代模型 (Iterator Model)
  - 物化模型 (Materialization Model)
  - 向量 / 批量模型 ( Vectorized / Batch Model )

# 1. 迭代模型

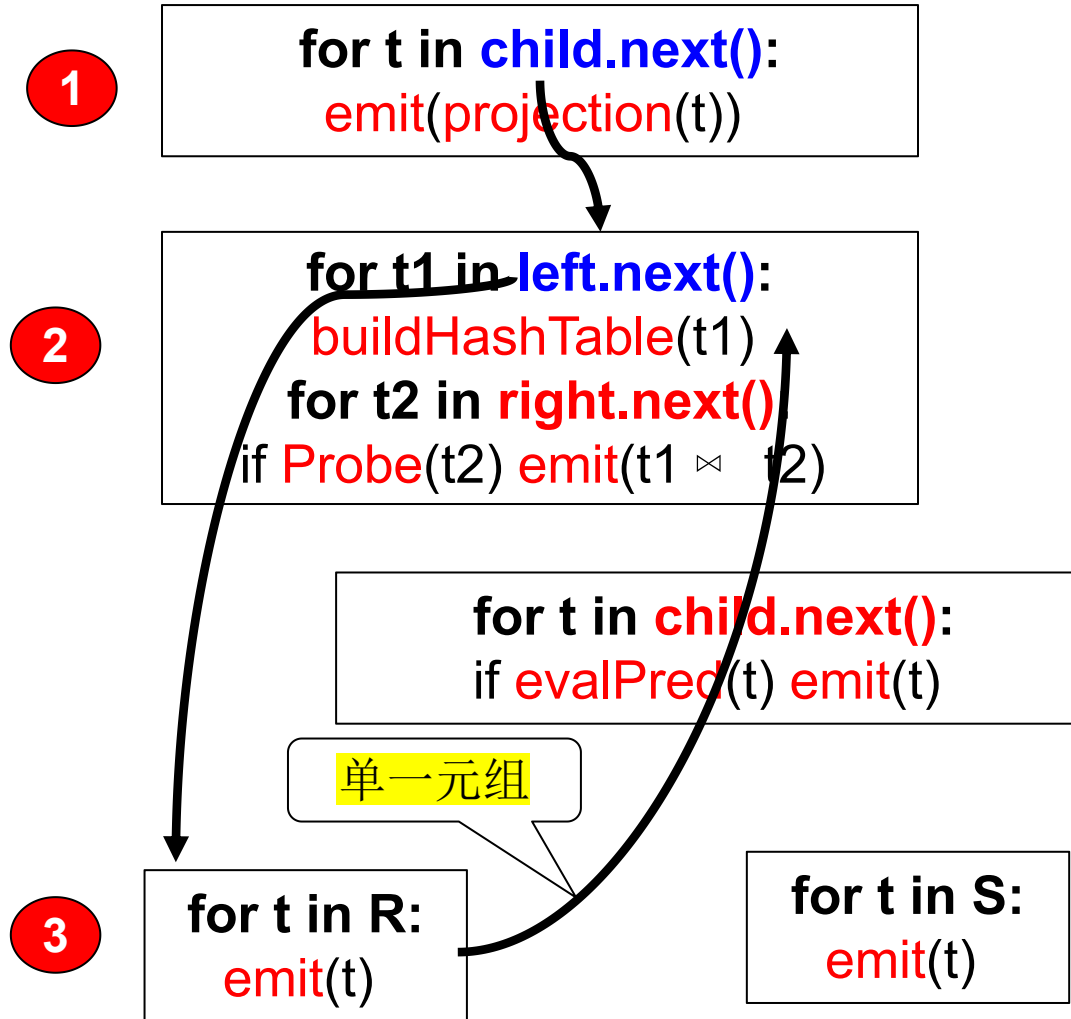
- **迭代模型**(Iteration Model): 每个查询计划的算子执行一个Next函数。
  - 也称作**火山模型** ( Volcano Model ) / **流水线模型** ( Pipeline Model ) 。
  - 每次调用Next得到一个**元组**, 或者一个空值标记 (null marker) 。
  - 该计算模型将关系代数中每一种操作抽象为一个 Operator, 将整个 SQL 构建成一个 Operator 树, 查询树**自顶向下**的调用next()接口, 数据则自底向上的被**拉取**处理。
  - 算子通过**循环调用**它的子节点的Next函数用于获取元组进行, 当处理完成后再通过Next函数获取下一个元处理组进行处理, 直到得到一个NULL Marker
  - 获取磁盘数据代价太大, 需要它**在内存中做足够多的操作**。

# 1. 迭代模型

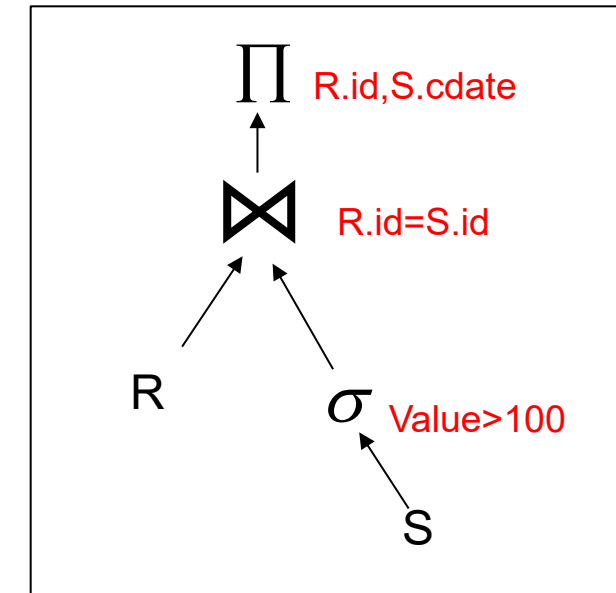




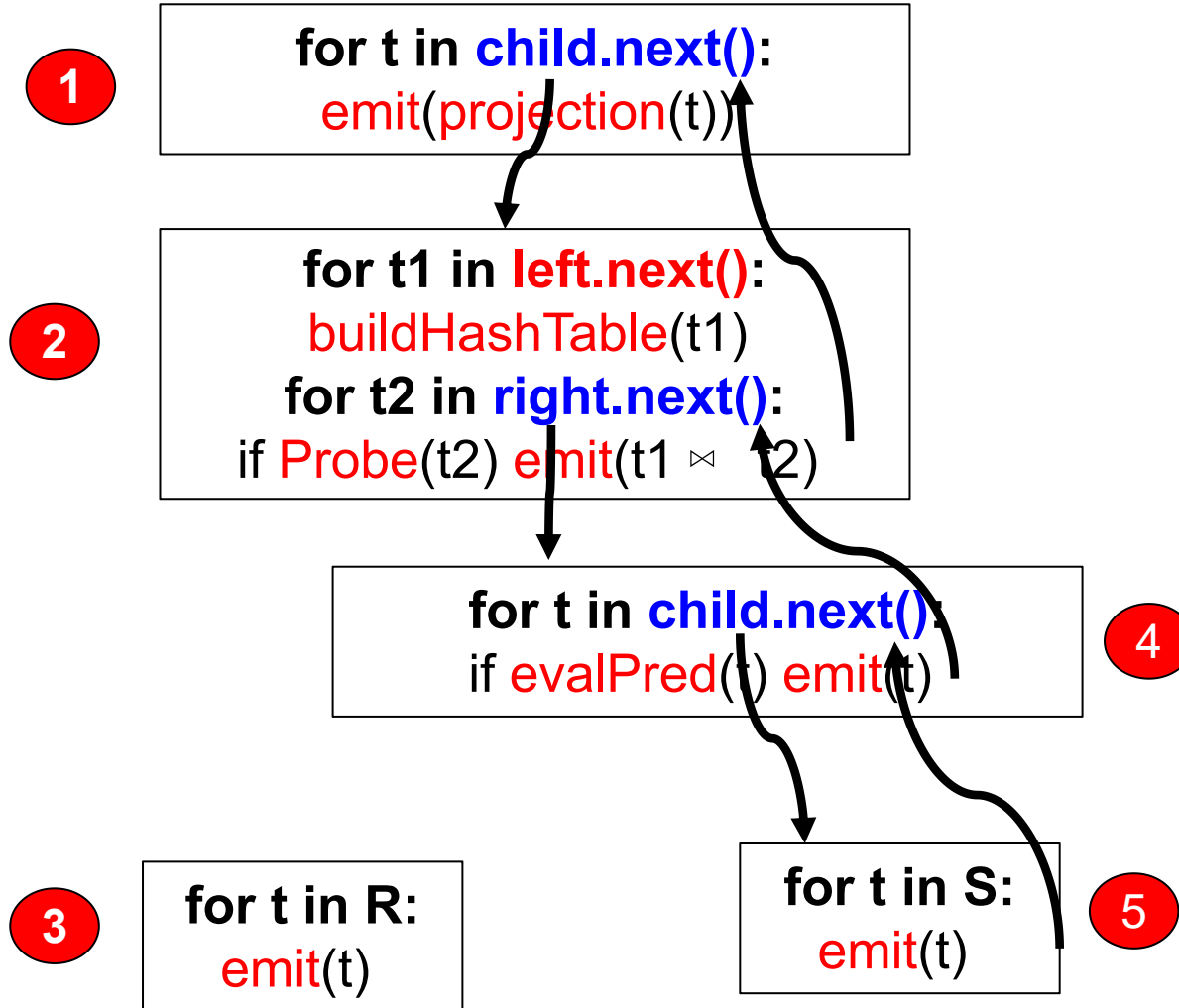
# 1. 迭代模型



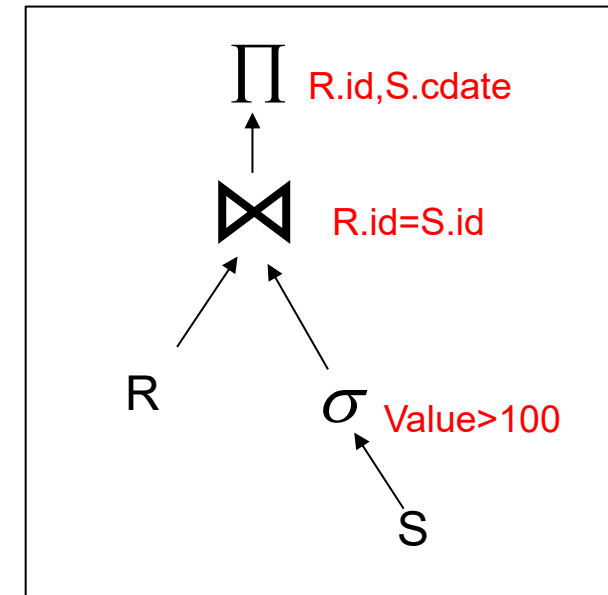
Select R.id,S.cdate  
From R join S  
ON R.id=S.id  
Where S.value>100



# 1. 迭代模型



Select R.id,S.cdate  
From R join S  
ON R.id=S.id  
Where S.value>100



# 1. 迭代模型

## □ 迭代模型特点:

- 大部分DBMS都采用该模型，允许元组流水线。
- 流水线带来的好处是：
  - (1) 消除读和写临时关系的代价，从而减少查询计算代价。
  - (2) 流水线产生查询结果，边生成边输出给用户，提高响应时间。很容易的控制输出，如limit操作。
- 有些算子会阻塞数据的流动，直到其子节点发送所有的元组，例如join操作，子查询操作，排序操作。

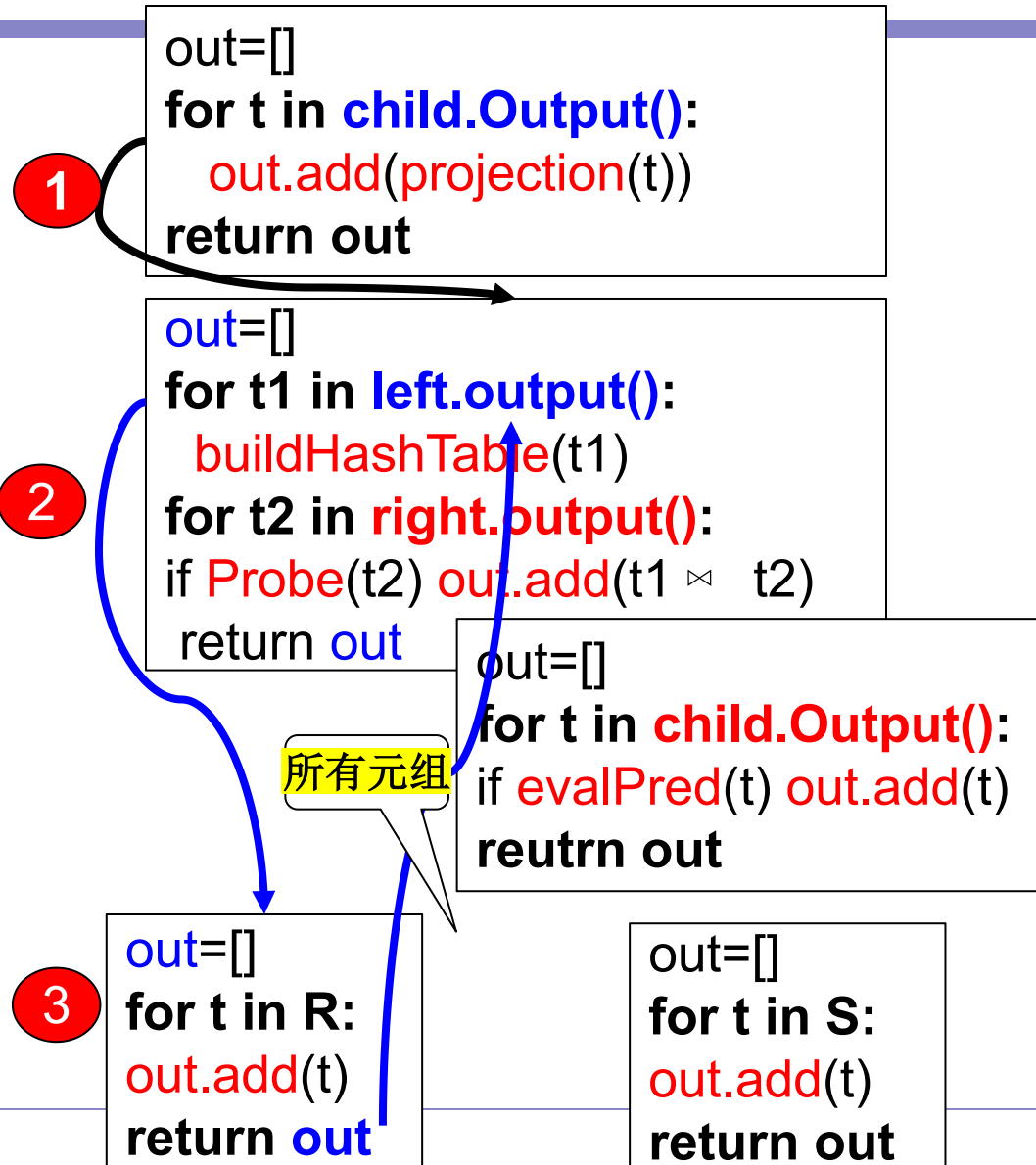
大多数关系型数据库都是使用迭代模型的，如 *SQLite*、*MongoDB*、*Impala*、*DB2*、*SQLServer*、*Greenplum*、*PostgreSQL*、*Oracle*、*MySQL* 等。

## 2. 物化模型

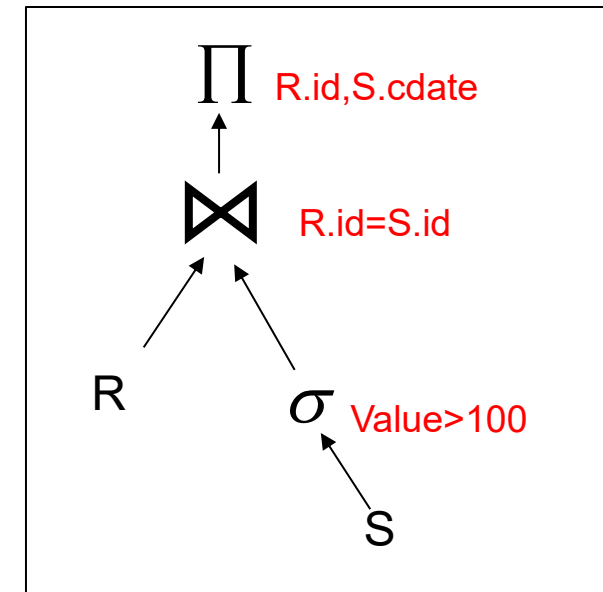
### □ 物化模型 (Maturation Model) :

- 算子一次性获取它所有的输入，当处理完成后再返回给它的父节点。
- 算子能“物化”其输出作为一整个结果。
- 为了避免下层算子返回的数据量过大，上层算子可以尽可能提供多的过滤条件。DBMS会将一些参数传递到操作符中防止处理过多的数据（如 limit）。
- 输出可以是整个元组（NSM存储模式），或属性子集（DSM存储模式）。

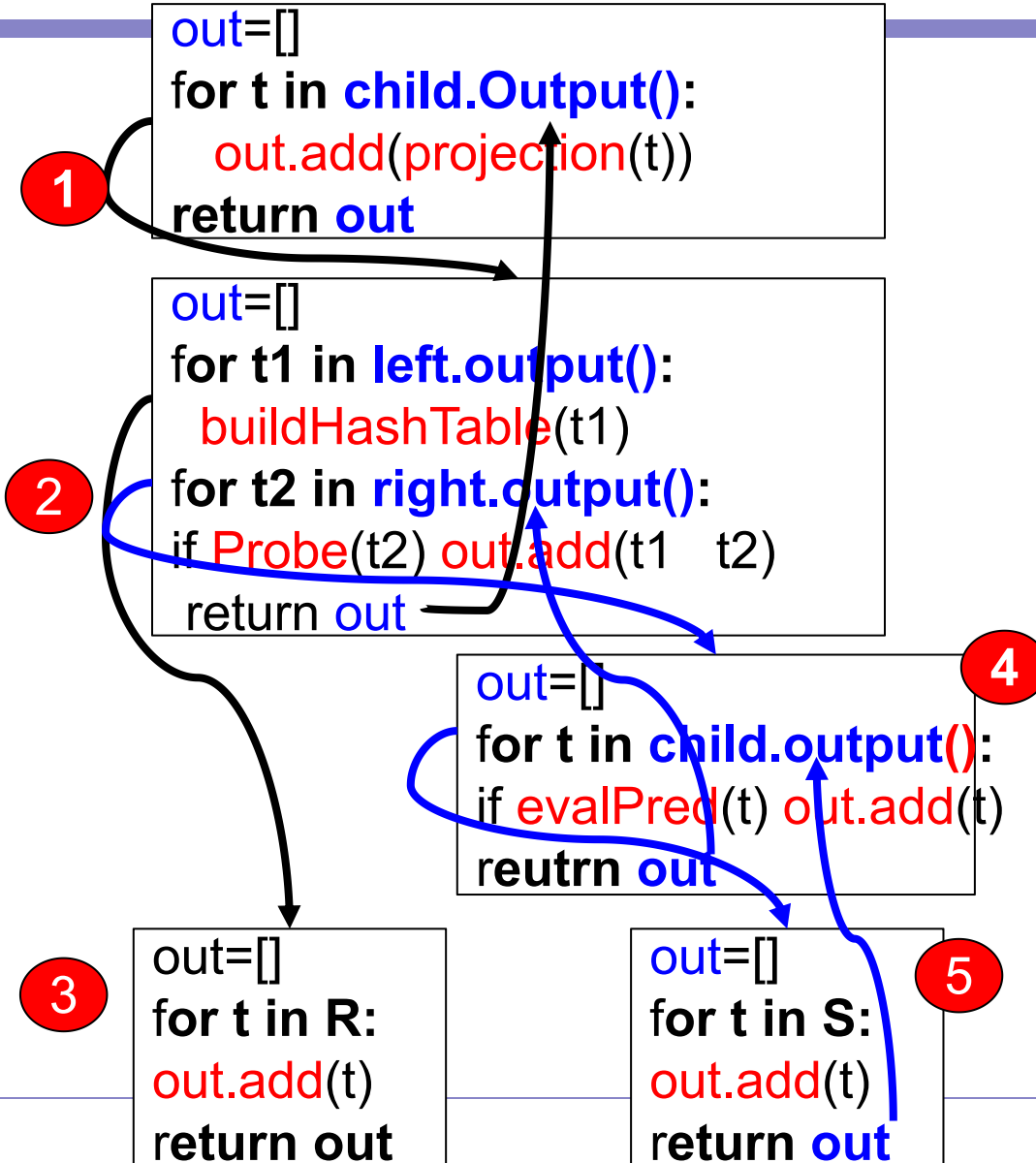
## 2. 物化模型



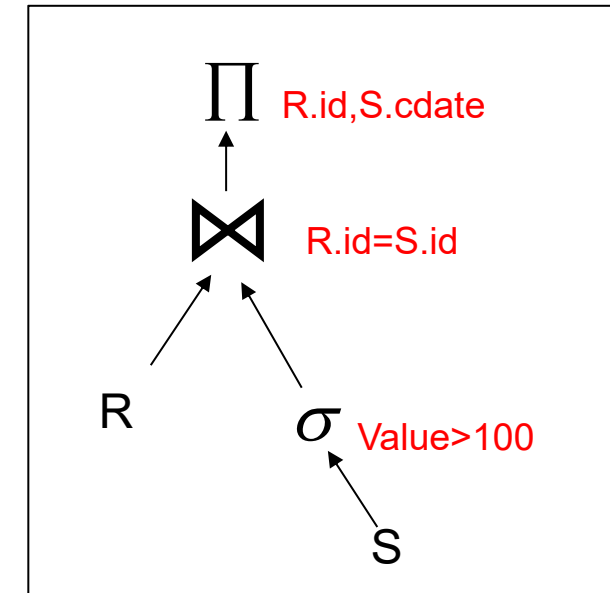
Select R.id,S.cdate  
From R join S  
ON R.id=S.id  
Where S.value>100



## 2. 物化模型



Select R.id,S.cdate  
From R join S  
ON R.id=S.id  
Where S.value>100



## 2. 物化模型

### □ 物化模型特点:

- 适合OLTP, 一次处理少量数据;
- 相对火山模型, 较少的函数调用, 能减少不必要的执行、调度成本。
- 不适合OLAP, AP查询可能产生较大的中间结果。

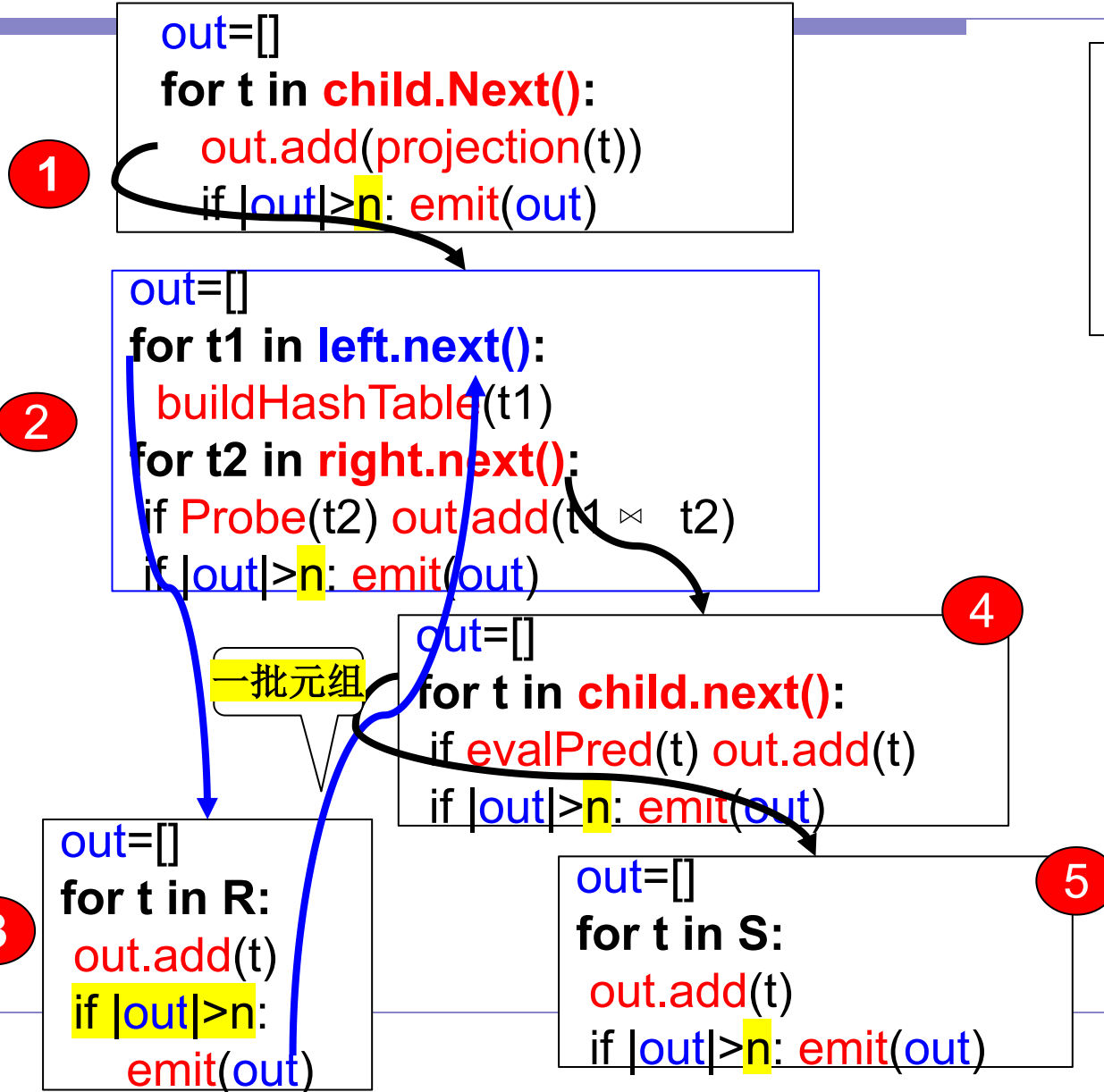
### 3. 向量/批处理模型

#### □ 向量/批处理模型：

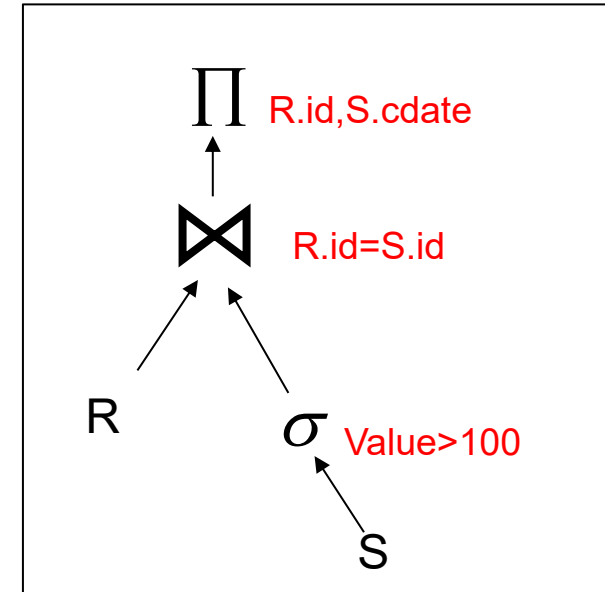
- 是前两种模型的折中
- 执行框架同火山模型
- 不同之处是每次调用Next函数，返回的是一批（batch）元组而不是一个元组；操作符内部的循环每次也是一批一批元组地处理。
- Batch的大小可以预先指定（比如硬件、查询的性质）。



# 3. 向量模型



Select R.id,S.cdate  
From R join S  
ON R.id=S.id  
Where S.value>100



# 3. 向量模型

- 向量模型特点：
  - 介于火山模型和物化模型之间；
  - 适用于OLAP；
  - 允许用户使用向量化的SIMD指令处理批量元组。

*Presto、snowflake、SQLServer、Amazon Redshift等数据库支持这种处理模式。*

# 查询树处理的方向

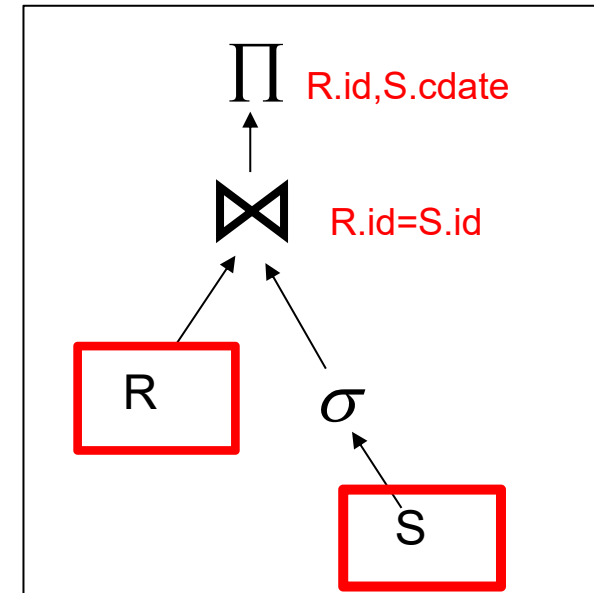
- **自顶而下**：从根节点开始不断的向子节点**拉**取数据，元组通过函数调用传递。
- **自底而上**：从叶子点开始获取数据不断的向父节点**推**送数据。

## 9.1.3 数据存取

- 数据存取方法(access method):  
访问存储在表中数据的方法。  
它并没有在关系代数中被定义。

```
Select R.id,S.cdate  
From R join S  
ON R.id=S.id  
Where S.value>100
```

- 三种基本方法:
  - 顺序扫描
  - 索引扫描
  - 多索引扫描(位图扫描)



# 1. 顺序扫描

## □ 顺序扫描 (Sequential Scan) :

- 对于表的每一个Page
- 加载到BufferPool中;
- 对BufferPool的Page中符合条件的tuple依次进行处理。
- DBMS内部维持一个游标指向上次访问的page/slot。

```
For Page in Table.Pages:  
  For t in Page.tuples:  
    If evalPred(t):  
      Do Something
```

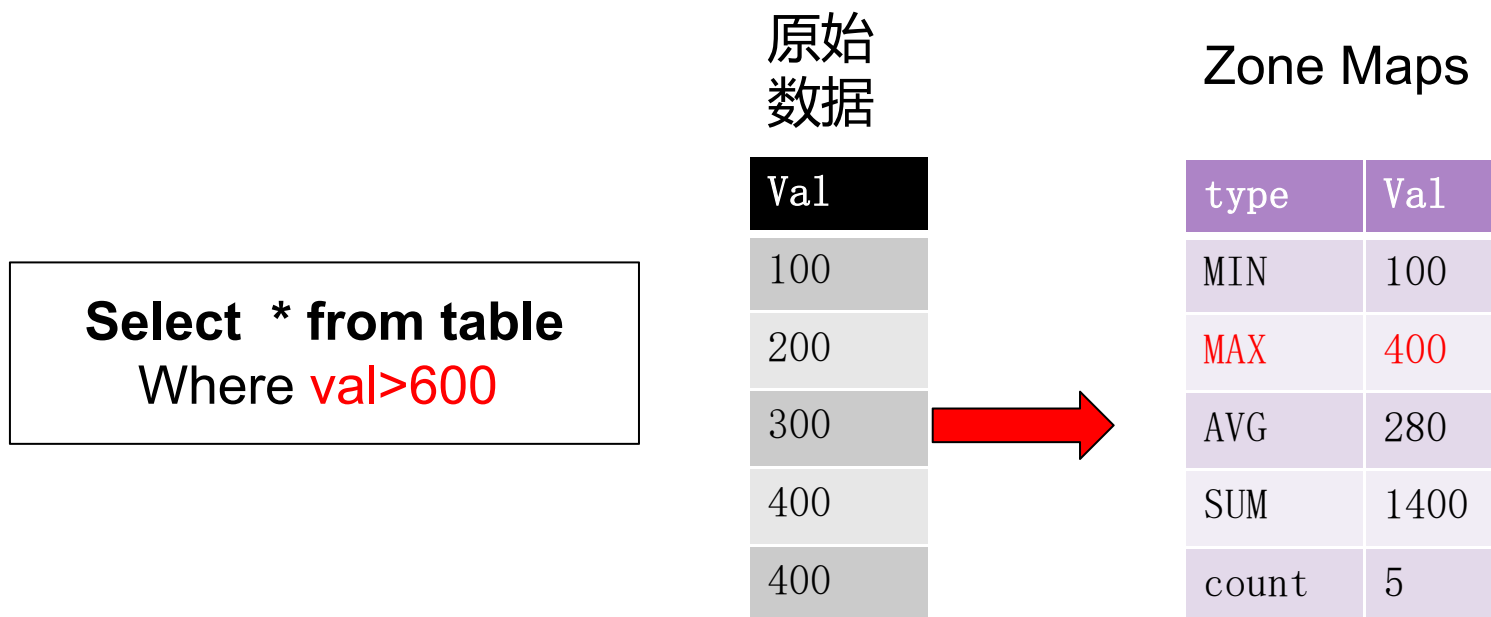
## □ 顺序扫描的优化:

- Prefetching, buffer pool bypass, Parallization, Heap Clustering
- Zone map
- Late Materialization (迟物化)

# 1. 顺序扫描

## □ Zone Maps

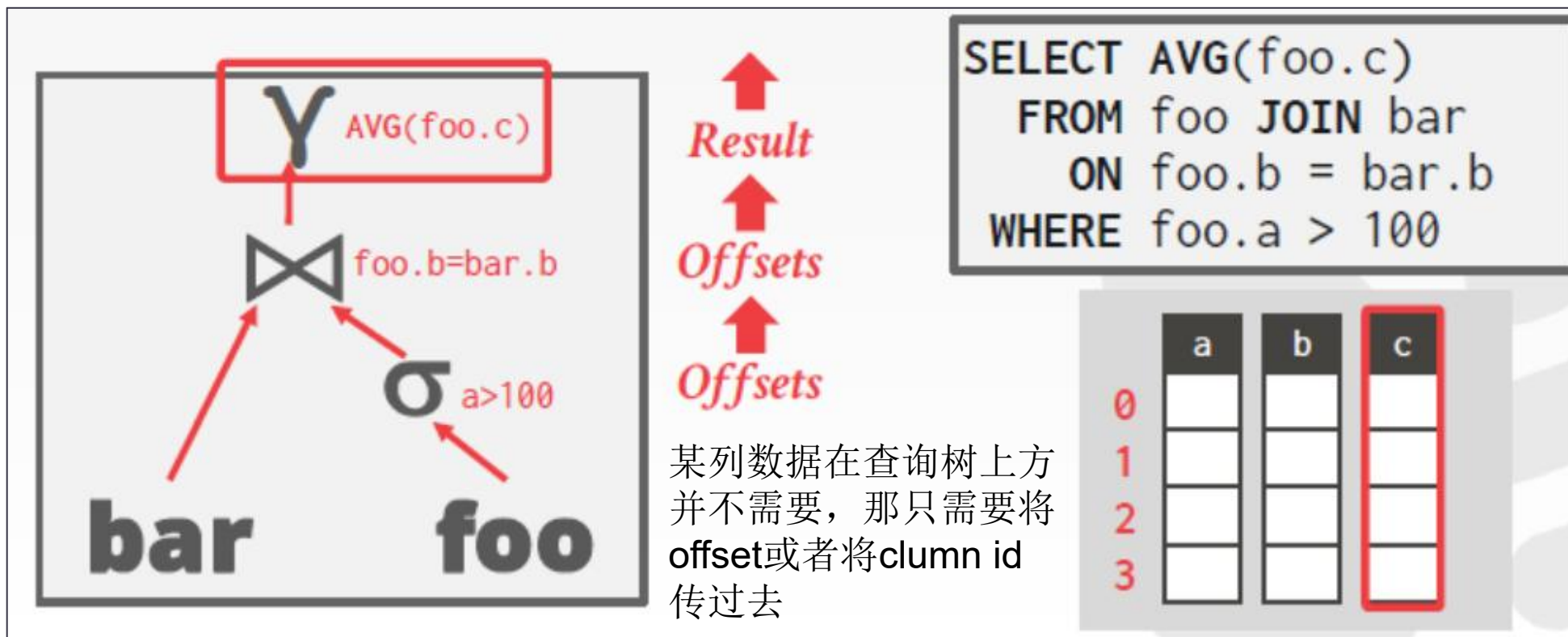
- 预先计算一个页中的属性统计值（最大值，最小值，平均值），并存入 zone map 中；
- 在顺序扫描中根据 zone map 来决定是否读取该页。



# 1. 顺序扫描

## □ Late Materialization

延迟缝合元组，直到查询计划的上层才缝合。



## 2. 索引扫描

- **索引扫描**：通过索引来访问查询所需要的数据页。
- 例：假设students表包含100个元组和两个索引。

数据的分布情况  
决定了索引效果

- 索引 1: age
- 索引 2: dept
- 通常只用一个索引

```
Select * from students
Where age<30
And dept= 'CS'
And Country='US'
```

场景1:

假设有99个学生年龄小于30，但是只有2个学生是CS系的，那么应该选择哪个属性索引呢？

场景2:

假设有99个学生是CS系的，但是只有2个学生年龄小于30，那么应该选择哪个属性索引呢？



## 2. 索引扫描

### □ 多索引扫描

- 对每个索引进行扫描，获取那些指向满足单个条件的元组ID（或元组指针）；
  - 根据谓词求取这些元组ID集合的并集或交集；
  - 最后对于根据剩余的谓词（没有索引）过滤这些元组。
- 
- Postgres 中Bitmap Scan

## 2. 索引扫描

### □ 多索引扫描举例：

(假设在age和dept上有索引)

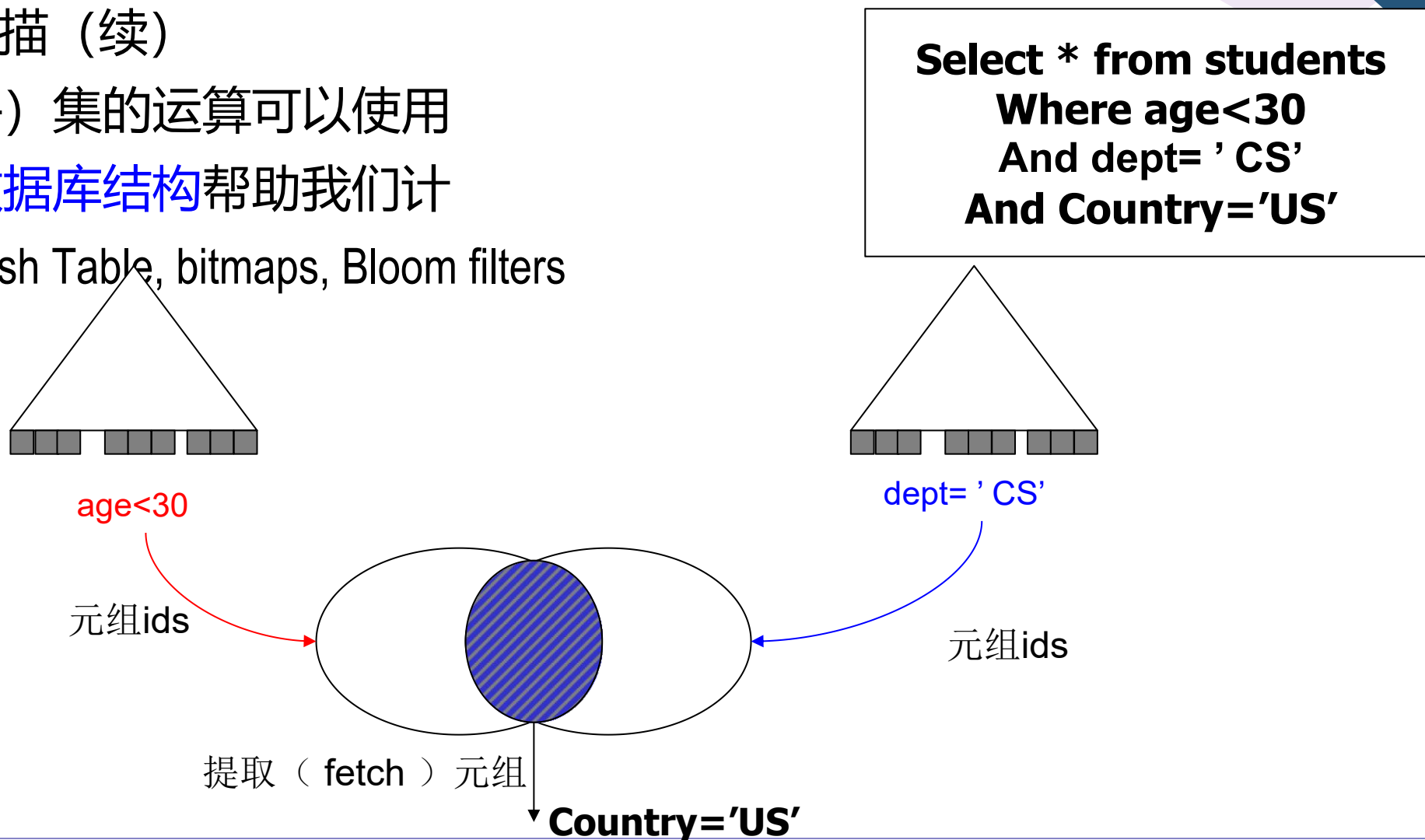
- ① 利用age的索引求age<30的元组指针集和 (或元组ID集和) ；
- ② 利用dept的索引求dept= ' CS'的元组指针 (或元组ID) ；
- ③ 求两个集合的交集；
- ④ 根据交集检索数据并查看元组是否满足谓词Country='US'。

```
Select * from students
Where age<30
And dept= ' CS'
And Country='US'
```

## 2. 索引扫描

### □ 多索引扫描（续）

- 交（并）集的运算可以使用
- 辅助数据库结构帮助我们计算如Hash Table, bitmaps, Bloom filters



# 实现查询操作的算法示例

## 1.选择操作的实现

[例1] `Select * from student where <条件表达式> ;`

考虑<条件表达式>的几种情况：

C1: 无条件;

C2: `Sno = '200215121';`

C3: `Sage>20;`

C4: `Sdept = 'CS' AND Sage>20;`

**选择操作典型实现方法：**

❖ **全表扫描**（适合小表，不适合大表）；

❖ **索引（或散列）扫描**（适合选择条件中的属性上有索引）

# 实现查询操作的算法示例

## 2.连接操作的实现

[例2] Select \* from student , SC WHERE Student.Sno=SC.Sno ;

### 连接操作典型实现方法:

1. 嵌套循环方法(nested loop)
2. 排序-合并方法(sort-merge join 或merge join)
3. 索引连接(index join)方法
4. Hash Join方法

外关系 (outer relation) 和 内关系 (inner relation)

左

m个元组, 占M页

右

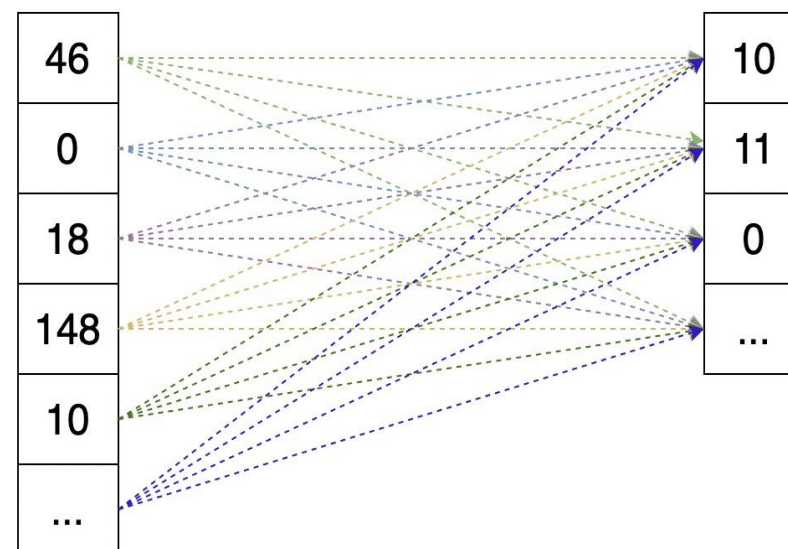
n个元组, 占N页

# 连接操作典型实现方法

## 1. 嵌套循环方法(nested loop)

- 对外层循环(Student)的每一个元组(s), 检索内层循环(SC)中的每一个元组(sc);
- 检查这两个元组在连接属性(sno)上是否相等;
- 如果满足连接条件, 则串接后作为结果输出, 直到外层循环表中的元组处理完为止。

- 从算法上看,  $O(m*n)$ ;
- 从磁盘IO看, 从磁盘读  $m+m*n$ ;
- 若外关系在内存中, 只需要读  $m+n$  行
- 若外关系太大, 采用**块嵌套循环连接方式**,  $M+M*N$



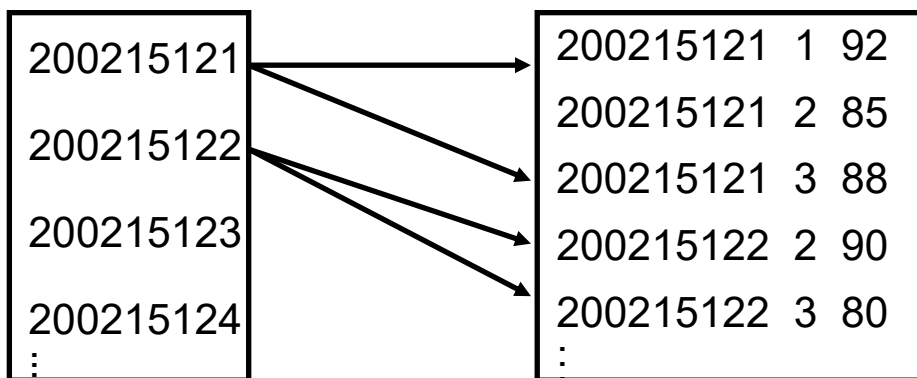
外关系

内关系

# 连接操作典型实现方法

## 2. 排序-合并方法(sort-merge join 或merge join)

- 适合连接的诸表已经排好序的情况。
- 步骤：
  - 如果连接的表没有排好序，先对Student表和SC表按连接属性Sno排序；
  - 取Student表中第一个Sno，依次扫描SC表中具有相同Sno的元组；
  - .....
  - 重复上述步骤直到Student 表扫描完。



Student表和SC表都只要扫描一遍

若左右表均有序,  $O(m+n)$

若还需要排序, 排序成本:

$O(m \cdot \text{Log}(m) + n \cdot \text{Log}(n))$

# 连接操作典型实现方法

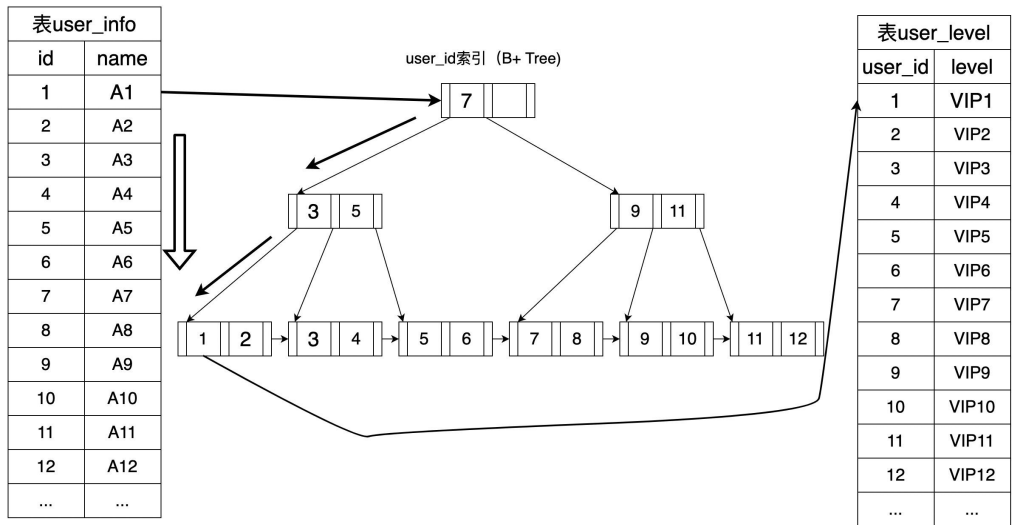
## 3. 索引连接(index join)方法

### ■ 步骤:

- ① 在**内**关系 (SC表) 上**建立连接属性Sno的索引**(如果原来没有);
  - ② 对外关系 (S) 中每个元组, 由Sno值通过SC的索引查找相应的SC元组;
  - ③ 把这些SC元组和Student元组连接起来;
- 循环执行②③, 直到Student表中的元组处理完为止。

假设索引查找带来的I/O开销为C,  
则总的I/O开销为:  $M + (m \times C)$

如果两个关系上均有索引时, 一般把元组较少的关系作外关系时效果较好。





# 连接操作典型实现方法

## □ 4. Hash Join方法

把连接属性作为hash码，用同一个hash函数把R和S中的元组散列到同一个hash文件中。

### ■ 步骤：

□ 划分阶段(partitioning phase)

□ 试探阶段(probing phase):

也称为连接阶段(join phase)

设内关系被划分成  $X$  个散列桶，  
创建散列表的成本  $(m) +$  散列函数的计算开销  $*n + (m/X) * n$

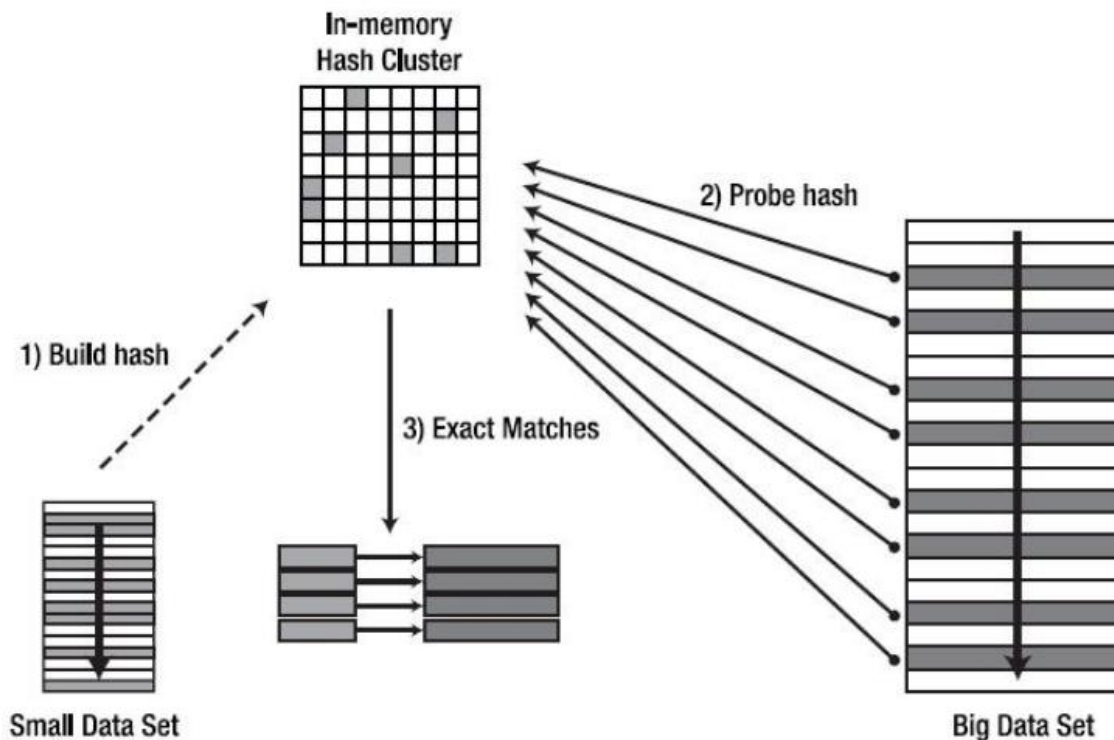


Figure 12-1. Optimal hash join

# 连接算法的选择

- (1) **空闲内存**：内存不够则无法使用内存中的散列连接。
- (2) **两个数据集的大小**：  
若一个大表连接一个很小的表，则嵌套循环连接就比散列连接快；  
若两个表都非常大，则嵌套循环连接的CPU成本就很高。
- (3) **是否有索引**：单一有，选索引连接；  
若连接属性上有两个B+树索引的话，合并连接是好选择。
- (4) 关系**是否已经排序**：有则选用合并连接。
- (5) **结果是否需要排序**：即使参与连接的是无序数据集，也可考虑使用成本较高的合并连接。
- (6) **连接的类型**：是等值连接？还是内连接？外连接？笛卡尔积？或者自连接？有些连接算法在某些情况下是不适用的。
- (7) **数据的分布**：若连接条件的数据是倾斜的，散列连接不是好选择。
- (8) **多表连接**：连接顺序的选择很重要。