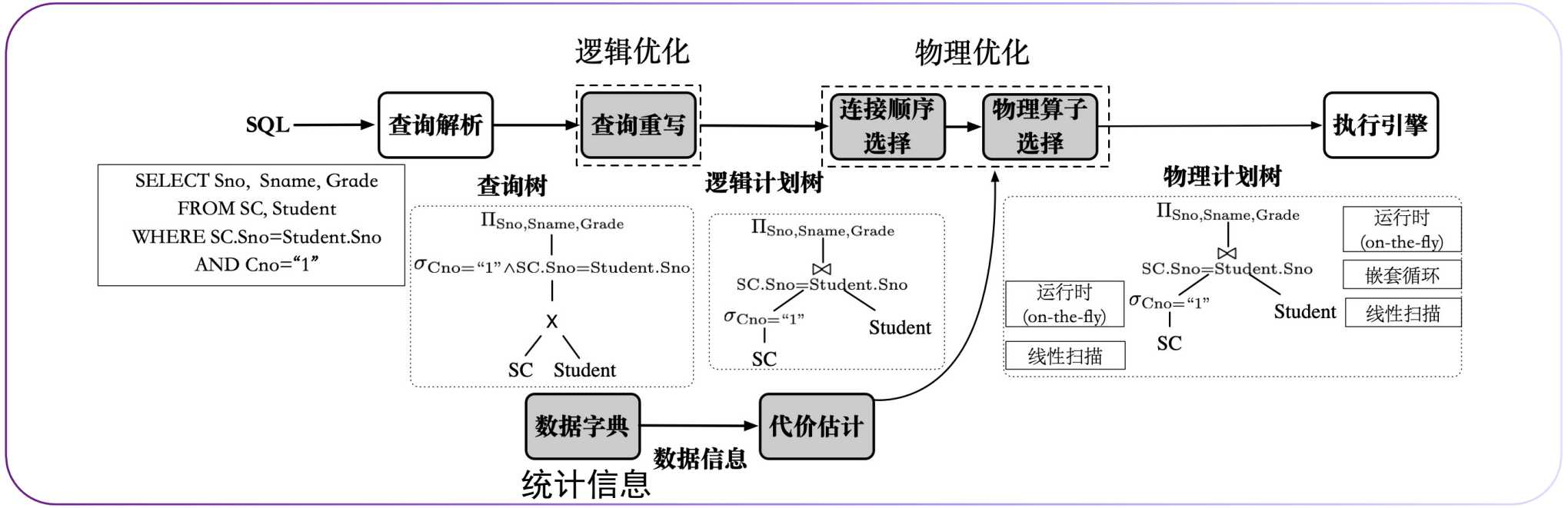
第11章 查询优化

清华大学 计算机系 数据库组 李国良

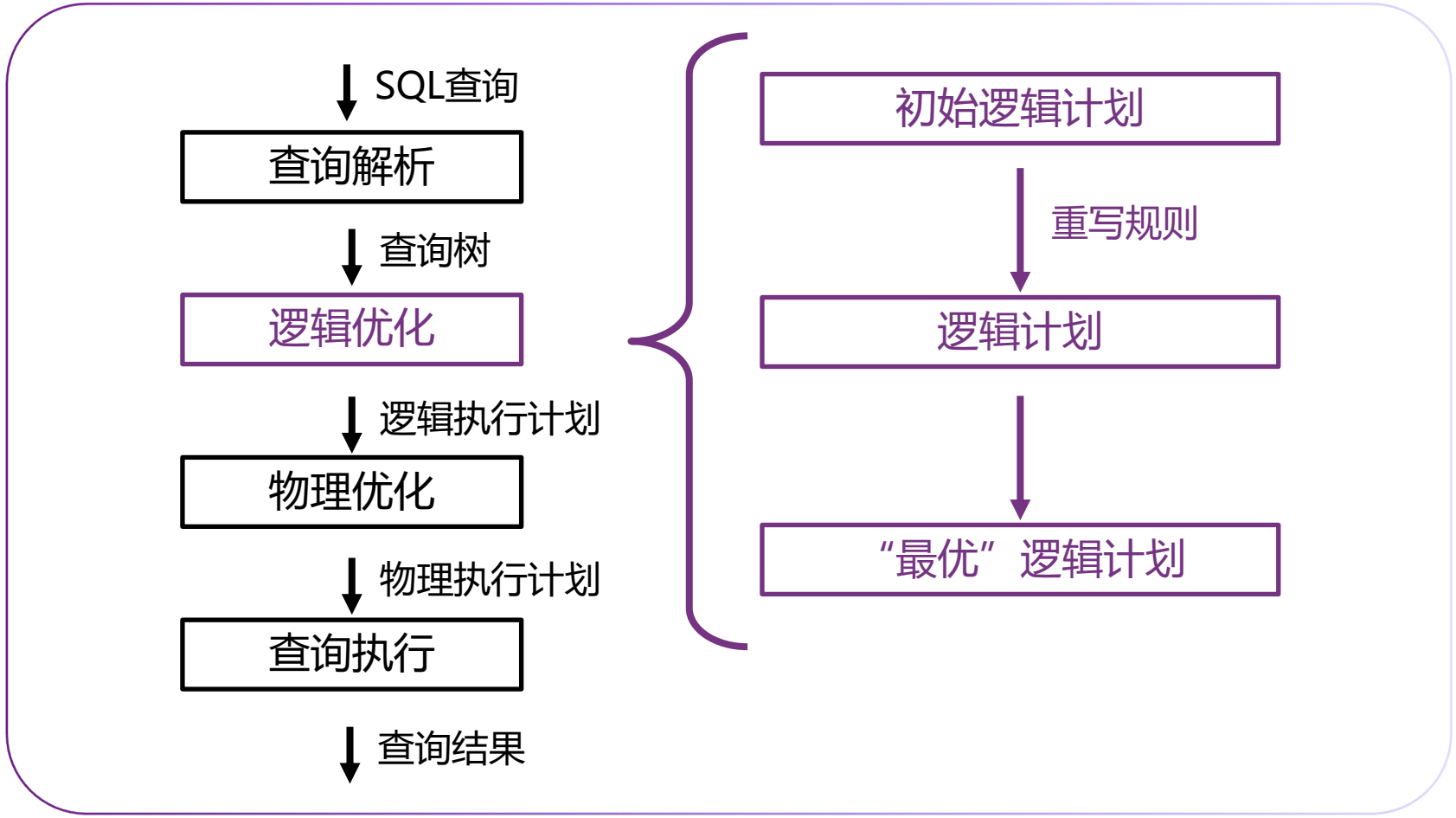
liguoliang@tsinghua.edu.cn
http://dbgroup.cs.tsinghua.edu.cn/ligl

查询优化流程

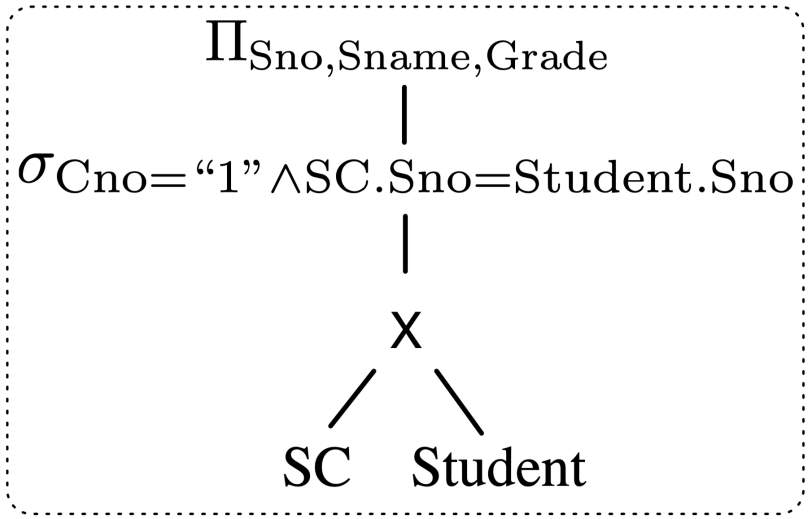
- **学生表** Student(Sno ,Sname, Sgender, Sage, Sdept)
- **学生选课表** SC(Sno, Cno, Grade)
- **查询**：获取选修了1号课程的学生学号、姓名及成绩

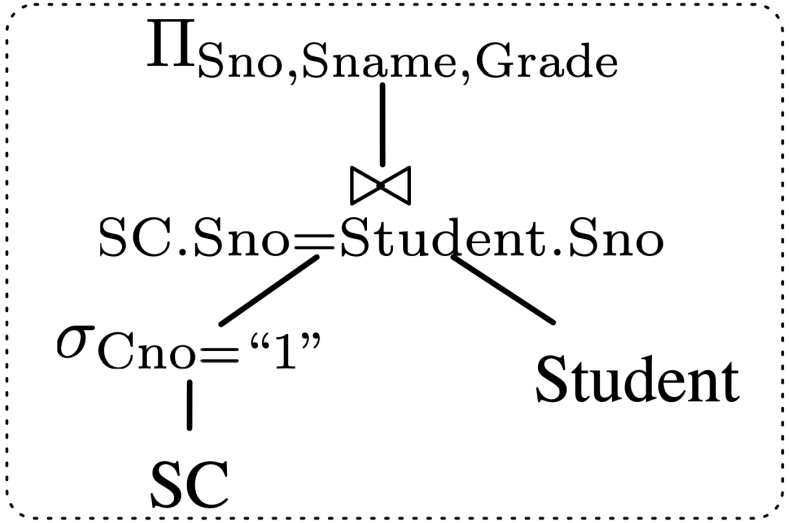


1、逻辑优化（查询重写）


SQL查询
查询解析
查询树
逻辑优化
逻辑执行计划
物理优化
物理执行计划
查询执行
查询结果
初始逻辑计划
重写规则
逻辑计划
“最优”逻辑计划

1、逻辑优化（查询重写）

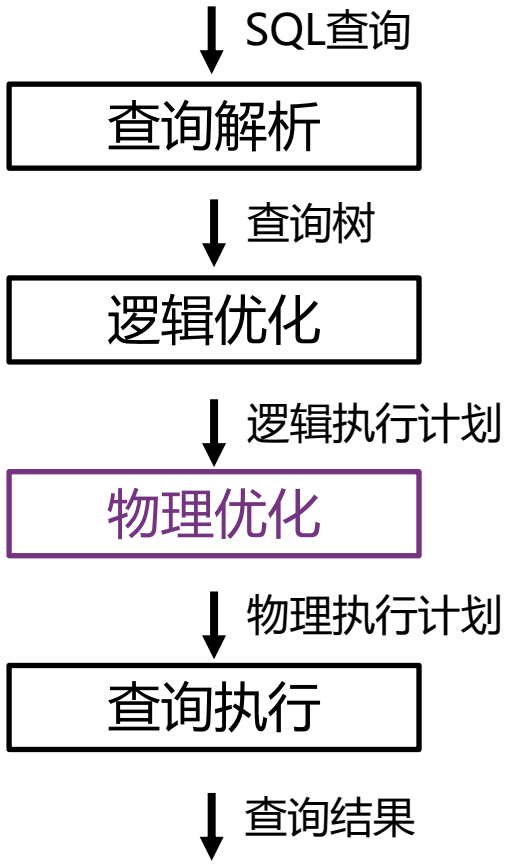




$$\Pi_{\text{Sno,Sname,Grade}}(\sigma_{\text{Cno}=\text{“1”}\wedge\text{SC.Sno}=\text{Student.Sno}}(\text{SC}\times\text{Student}))$$

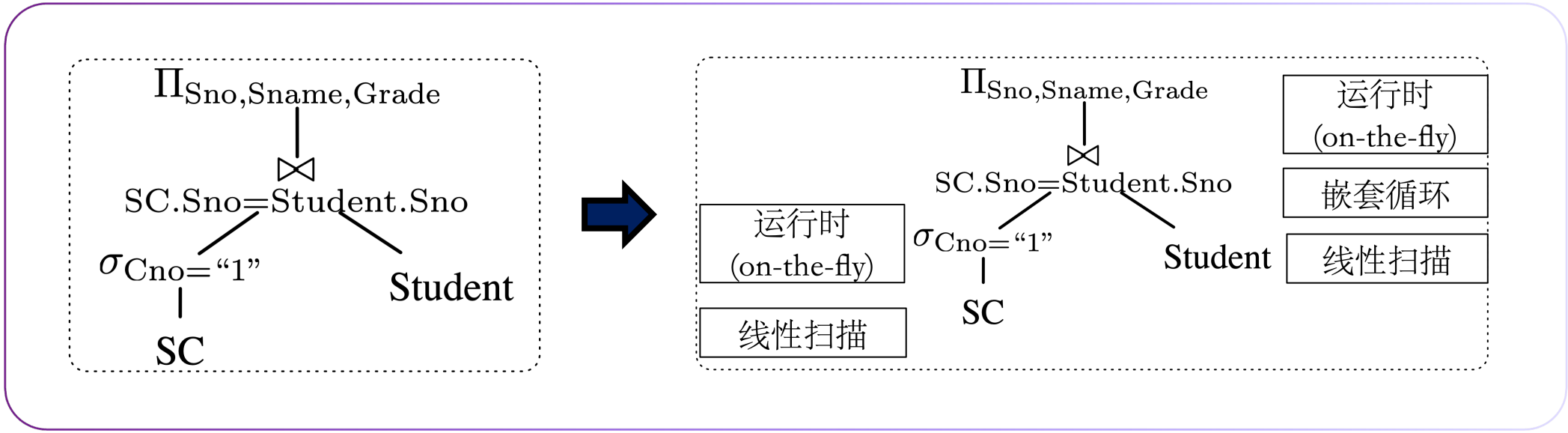
$$\Rightarrow \Pi_{\text{Sno,Sname,Grade}}((\sigma_{\text{Cno}=\text{“1”}}\text{SC})\bowtie\text{Student})$$

2、物理优化



- 枚举候选物理计划
- 计算每个计划的代价
- 选择代价最小的计划

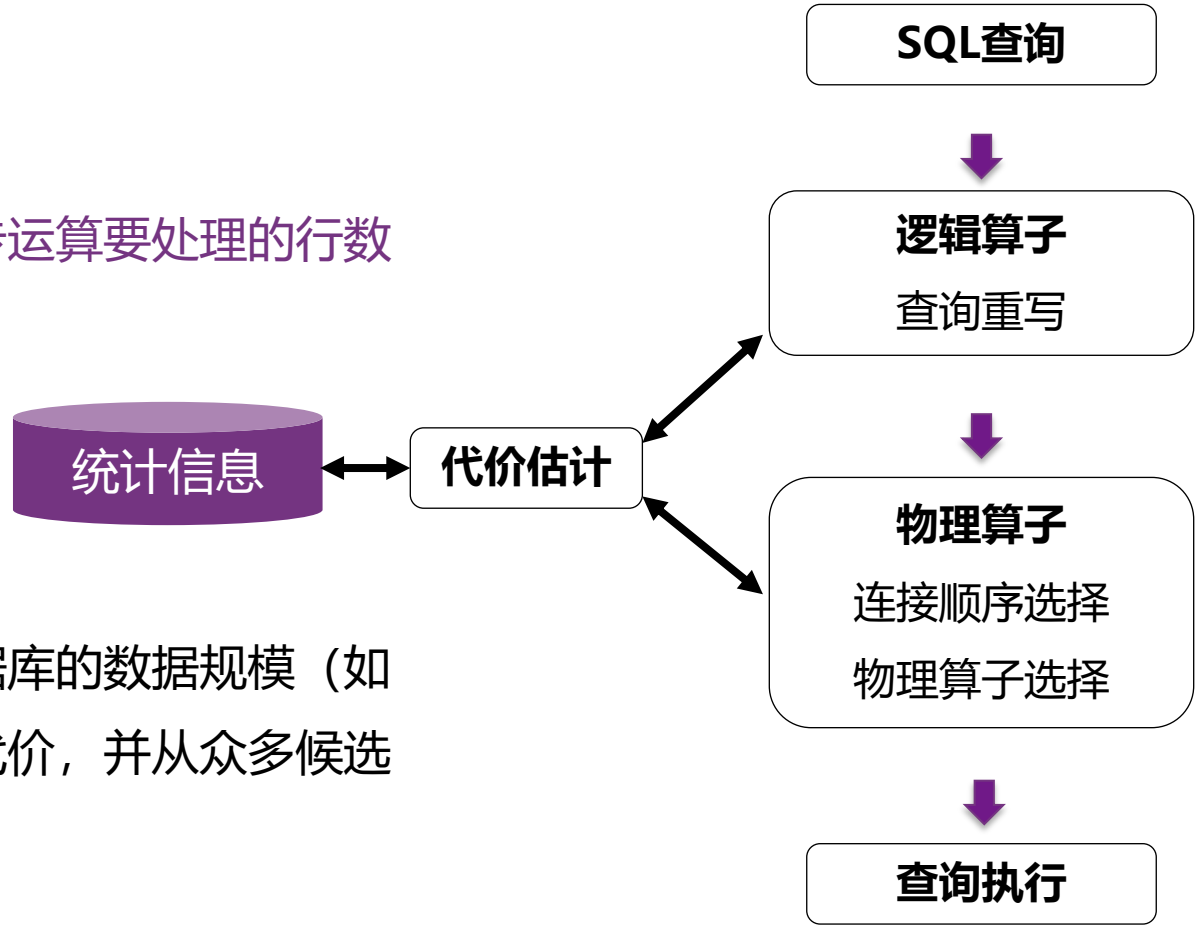
2、物理优化


Π Sno,Sname,Grade
⋈
SC.Sno=Student.Sno
σ Cno="1"
Student
SC
Π Sno,Sname,Grade
⋈
SC.Sno=Student.Sno
运行时
(on-the-fly)
σ Cno="1"
Student
线性扫描
SC
运行时
(on-the-fly)
嵌套循环
线性扫描

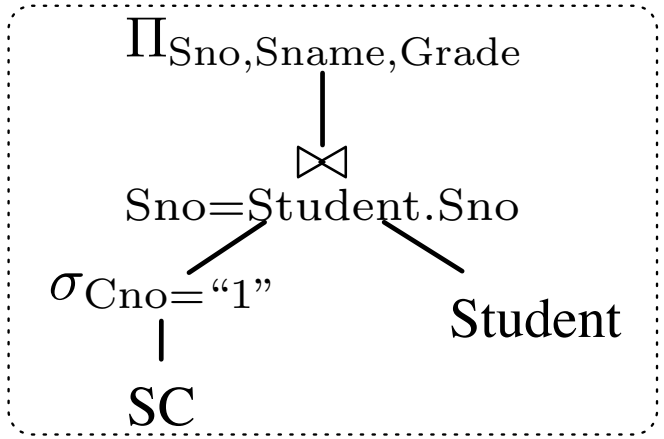
2、物理优化

- **逻辑优化：**
  - 找出比原始计划更高效的等价计划
  - 通过调整关系代数运算顺序来最小化每步运算要处理的行数

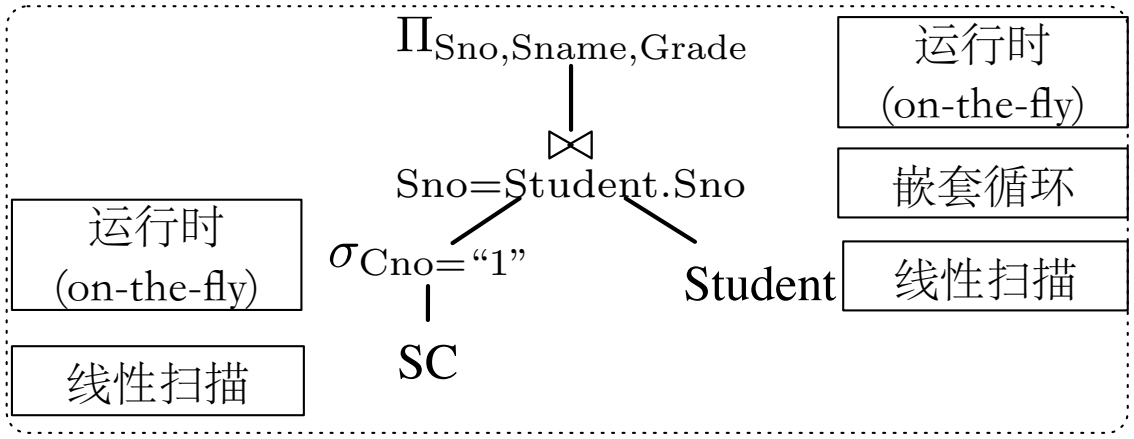
- **物理优化：**
  - 找出代价最小的算法来执行计划
  - 基于物理参数（如缓存大小等）估计数据库的数据规模（如数据块大小等），从而估计计划执行的代价，并从众多候选计划中选择代价最小的计划



2、物理优化



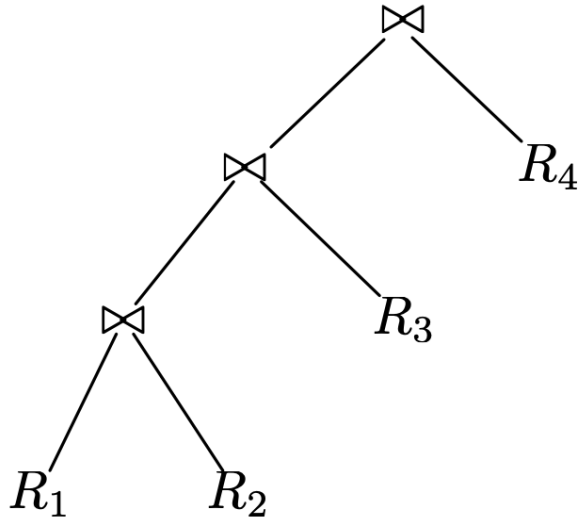
最优逻辑计划



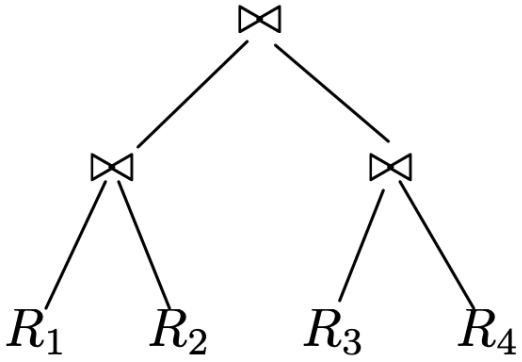
物理计划

2、物理优化

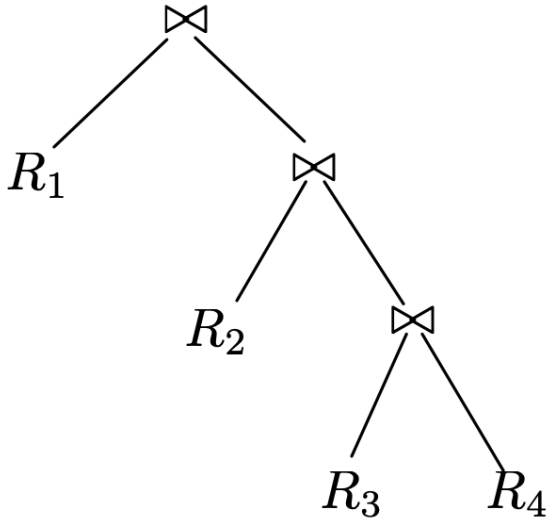
- 不同连接顺序和物理算子代价不同，选择代价最小的连接顺序和物理算子
  - ① **基数估计**
  - ② **代价估计**
  - ③ **连接顺序选择（NP问题）**
  - ④ **物理算子选择**



(a)



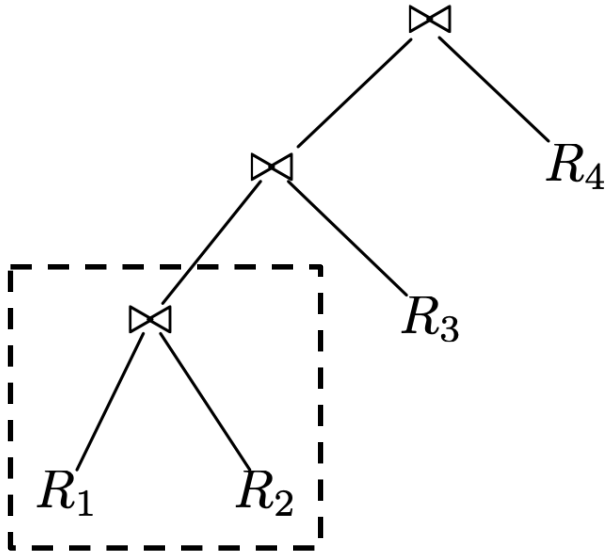
(b)

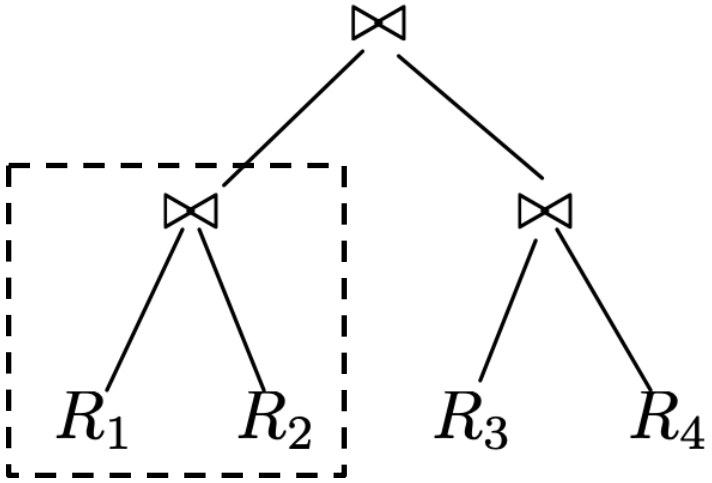


(c)

3、物化视图

- 将频繁的子查询物化为视图
- 利用视图来提升查询速度
  - 物化视图生成
  - 利用物化视图查询优化
  - 物化视图更新





目录

查询重写

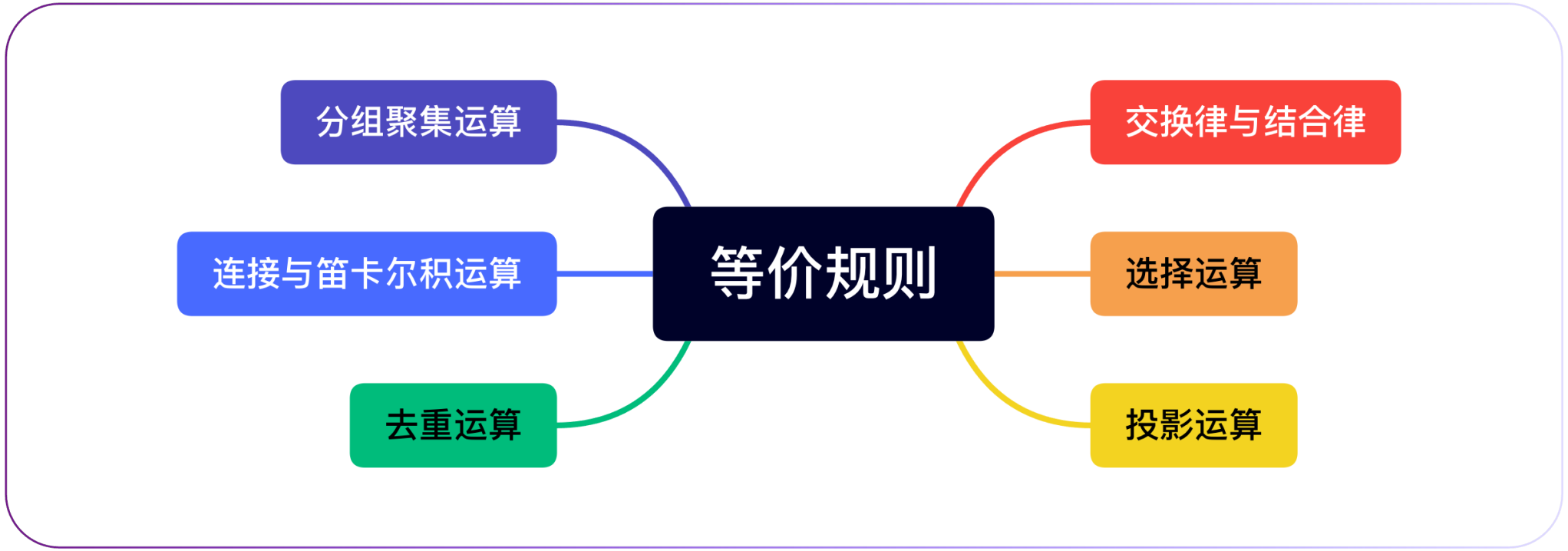
- **查询重写(query rewrite)**：按照一系列关系代数表达式的**等价规则**，对查询的关系代数表达式进行**等价转换**，从而提高查询执行效率
- **关系表达式的等价（equivalent）**：对于两个关系代数表达式，如果使用相同的关系对表达式中的关系进行替换，总能得到**完全相同的结果**，则称这两个关系表达式**等价**

例如：

- $R_1 \bowtie R_2 = R_2 \bowtie R_1$
- $(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$

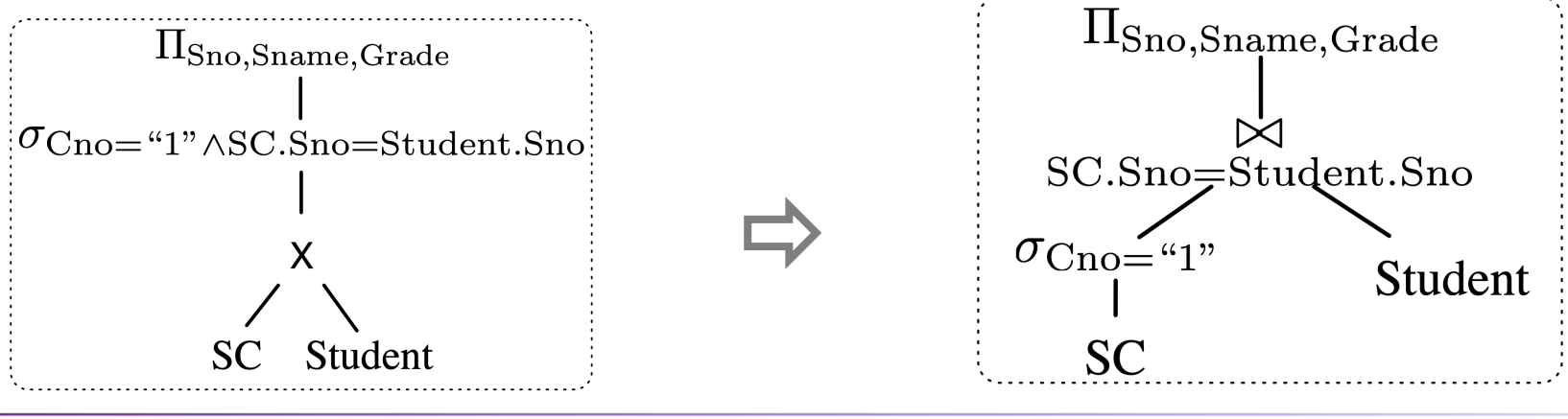
查询重写

- **等价规则(equivalence rule)**：指出哪些不同形式的关系表达式是等价的
- **核心思想**：通过调整关系代数运算的顺序，来**最小化每步运算要处理的行数**，从而提高运算效率。

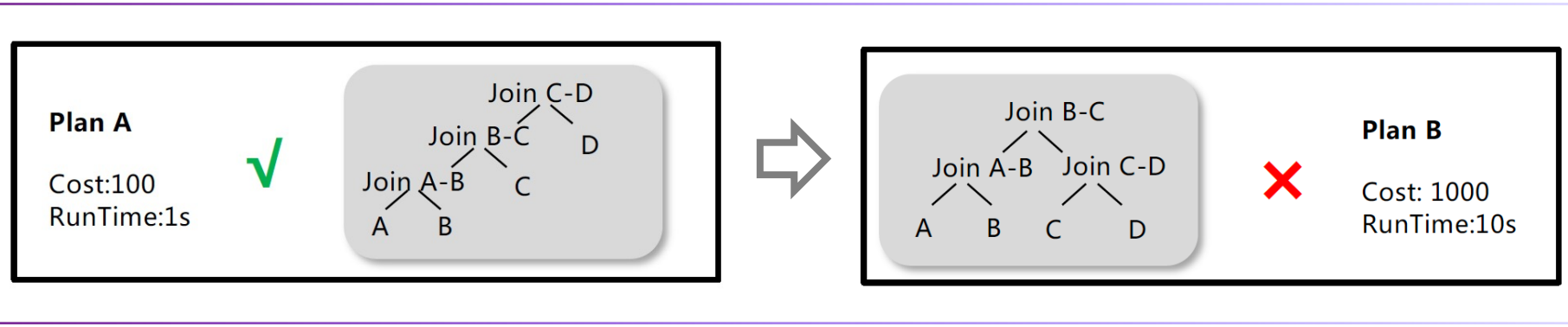


查询重写

1.基于规则



2.基于代价：按照代价重写



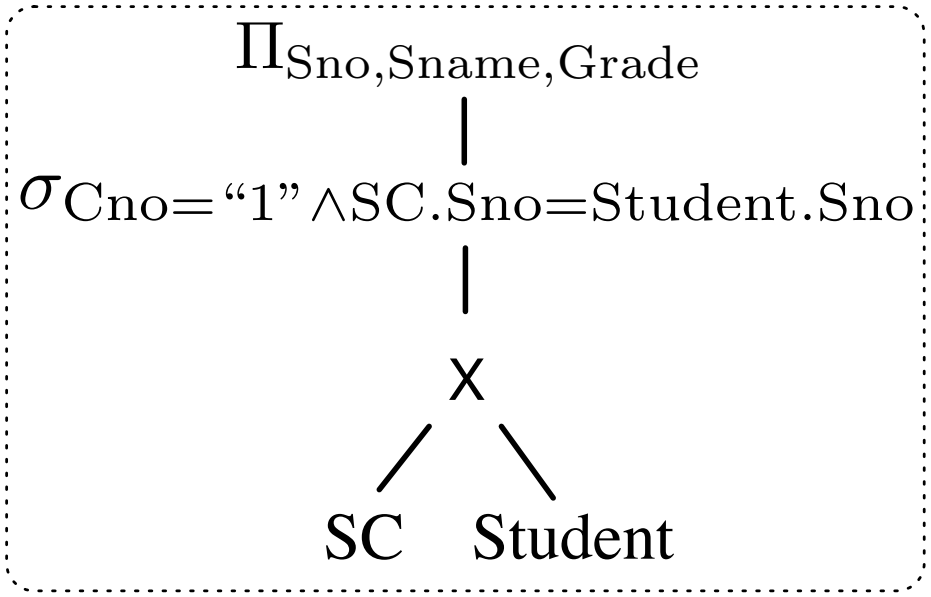
重写规则概览

- **下推谓词:** $\sigma_p(R_1 \times R_2) = \sigma_p(R_1) \times R_2$
- **尽早做投影:** $\Pi(R_1 \times R_2) = \Pi(R_1) \times \Pi(R_2)$
- **尽可能避免笛卡尔积:** $\sigma_p(R_1 \times R_2) = R_1 \bowtie_p R_2$
- **常数折叠/常数传播: A=B and A > 5 → B > 5**
- **去除非必要谓词: max(distinct A) → max(A)**
- **使用左深连接树**
- **将子查询转换为连接**
- **……**

查询重写对于提高查询执行效率至关重要!

一个查询重写实例（1）

```
SELECT Sno,  Sname, Grade
FROM SC, Student
WHERE SC.Sno=Student.Sno
AND Cno="1"
```

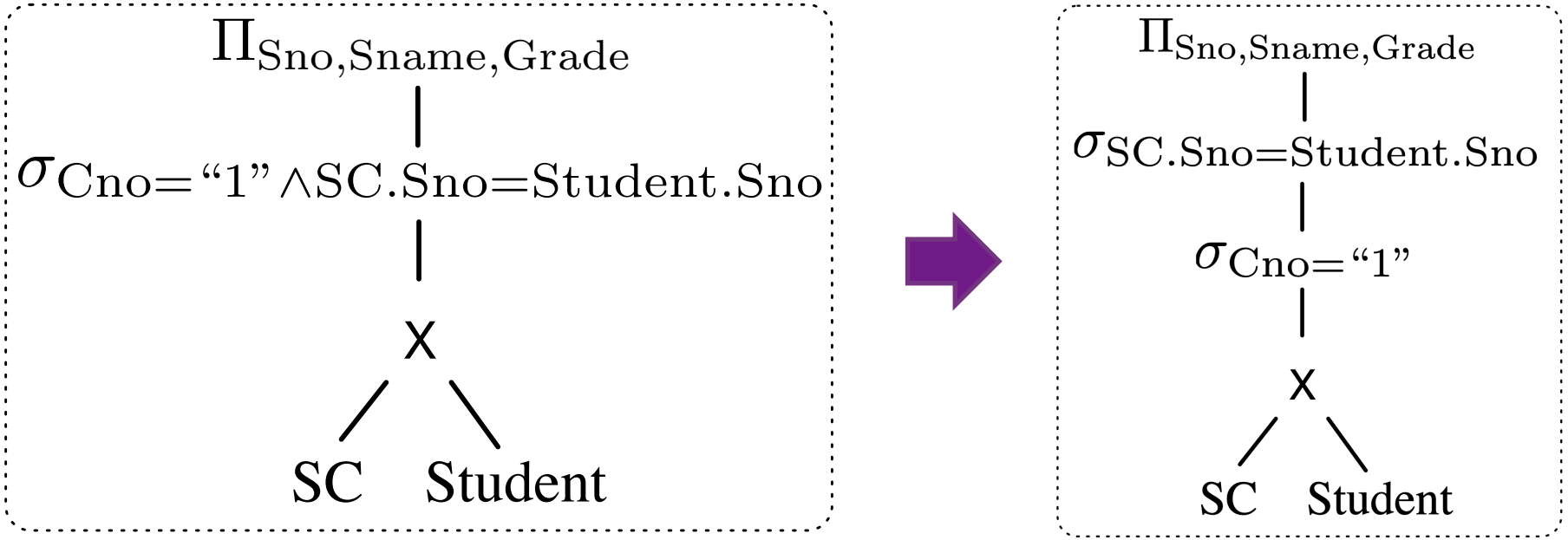


最初的逻辑计划

关系代数： $\Pi_{Sno,Sname,Grade}(\sigma_{Cno="1" \wedge SC.Sno=Student.Sno}(SC \times Student))$

一个查询重写实例（2）

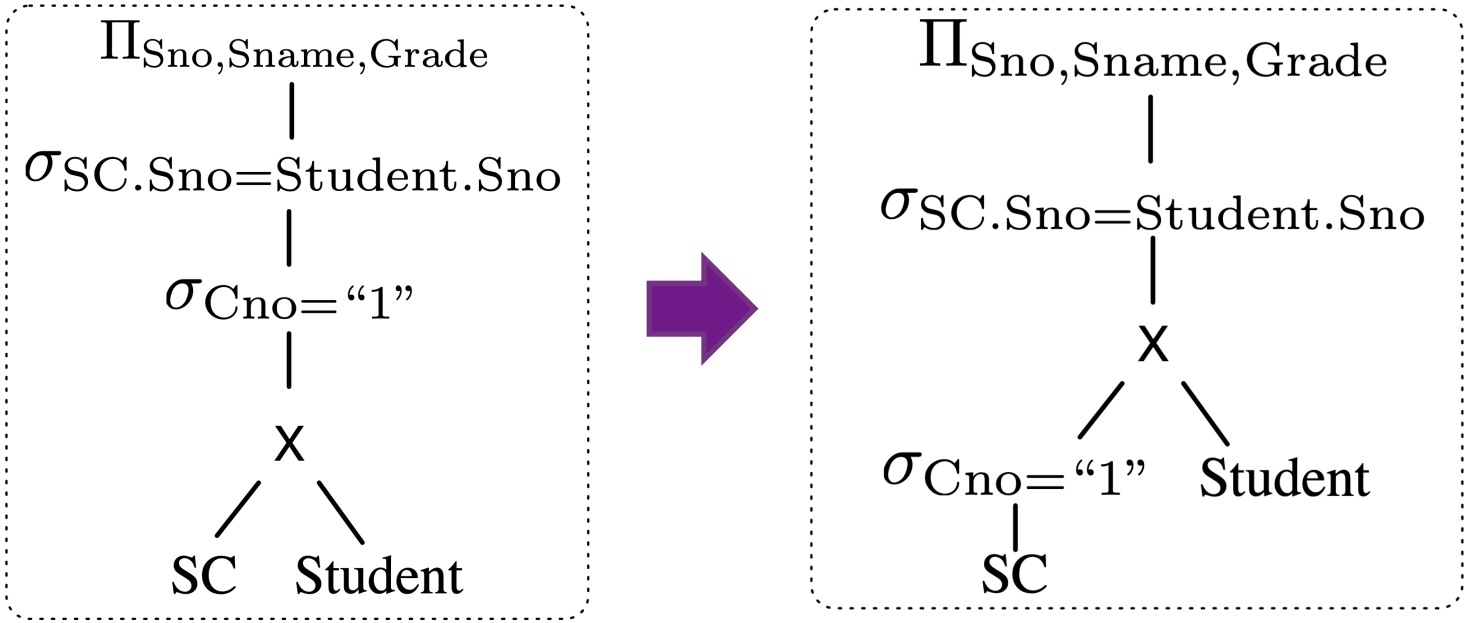
使用重写规则：$\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_1}\left(\sigma_{p_2}(R)\right)$



关系代数： $\Pi_{\text{Sno,Sname,Grade}}(\sigma_{\text{SC.Sno=Student.Sno}}(\sigma_{\text{Cno="1"}}(\text{SC} \times \text{Student})))$

一个查询重写实例（3）

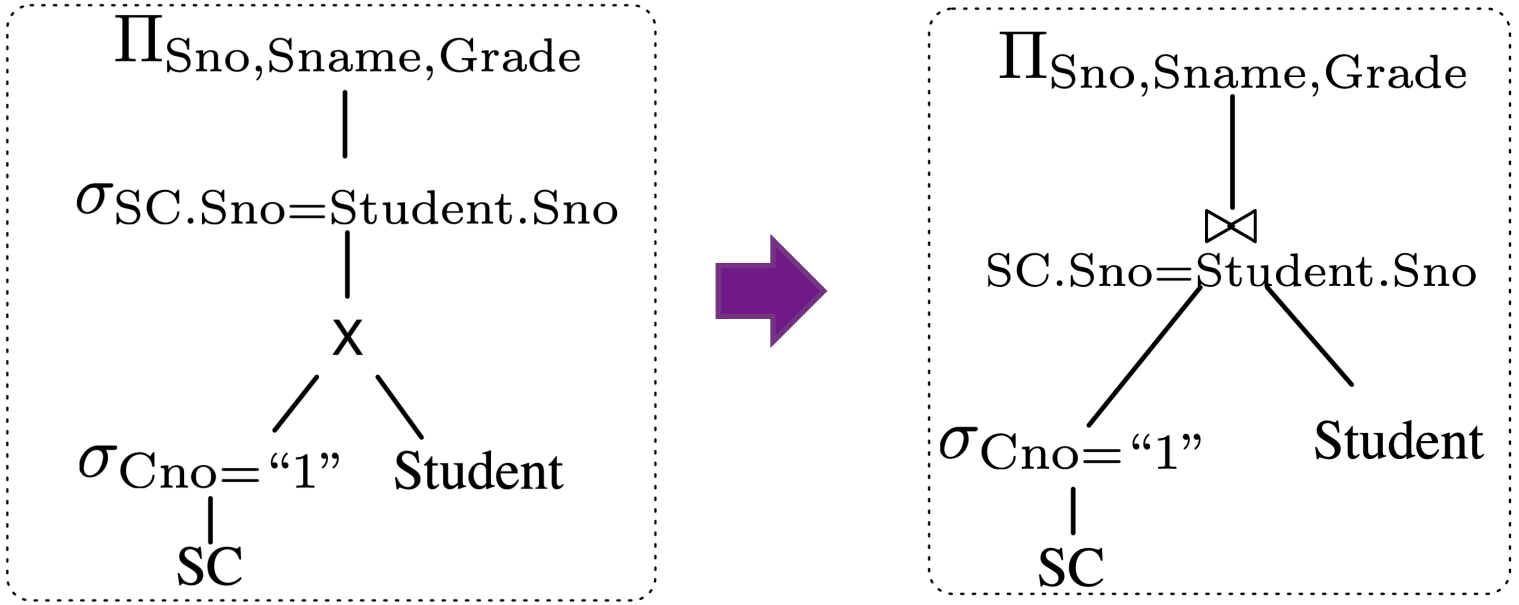
使用重写规则：若R_1包含p涉及的所有属性，$\sigma_p(R_1 \times R_2) = \sigma_p(R_1) \times R_2$



关系代数： $\Pi_{\text{Sno,Sname,Grade}}(\sigma_{\text{SC.Sno=Student.Sno}}(\sigma_{\text{Cno="1"}}(\text{SC}) \times \text{Student}))$

一个查询重写实例（4）

使用重写规则：$\sigma_p(R_1 \times R_2) = R_1 \bowtie_p R_2$



关系代数： $\Pi_{\text{Sno,Sname,Grade}}(\sigma_{\text{Cno}=\text{“1”}}(\text{SC}) \bowtie \text{Student})$

查询重写规则

序号	等价转换规则
1	$R_1 \times R_2 = R_2 \times R_1$
2	$(R_1 \times R_2) \times R_3 = R_1 \times (R_2 \times R_3)$
3	$R_1 \bowtie R_2 = R_2 \bowtie R_1$
4	$(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$
5	$R_1 \cup R_2 = R_2 \cup R_1$
6	$(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$
7	$R_1 \cap R_2 = R_2 \cap R_1$
8	$(R_1 \cap R_2) \cap R_3 = R_1 \cap (R_2 \cap R_3)$
9	$\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_1}\left(\sigma_{p_2}(R)\right)$

查询重写规则

序号	等价转换规则
10	$\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_2}\left(\sigma_{p_1}(R)\right)$
11	当R为集合时，$\sigma_{p_1 \vee p_2}(R) = \sigma_{p_1}(R) \cup \sigma_{p_2}(R)$
12	$\sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2)$
13	$\sigma_p(R_1 - R_2) = \sigma_p(R_1) - R_2$
14	$\sigma_p(R_1 - R_2) = \sigma_p(R_1) - \sigma_p(R_2)$
15	若关系R_1包含选择条件p涉及的所有属性，$\sigma_p(R_1 \times R_2) = \sigma_p(R_1) \times R_2$
16	若关系R_1包含选择条件p涉及的所有属性，$\sigma_p(R_1 \bowtie R_2) = \sigma_p(R_1) \bowtie R_2$
17	若关系R_1包含选择条件p涉及的所有属性，$\sigma_p(R_1 \cap R_2) = \sigma_p(R_1) \cap R_2$
18	若$S_1 \subseteq S_2$且都是关系R的属性，$\Pi_{S_1}\left(\Pi_{S_2}(R)\right) = \Pi_{S_1}(R)$

查询重写规则

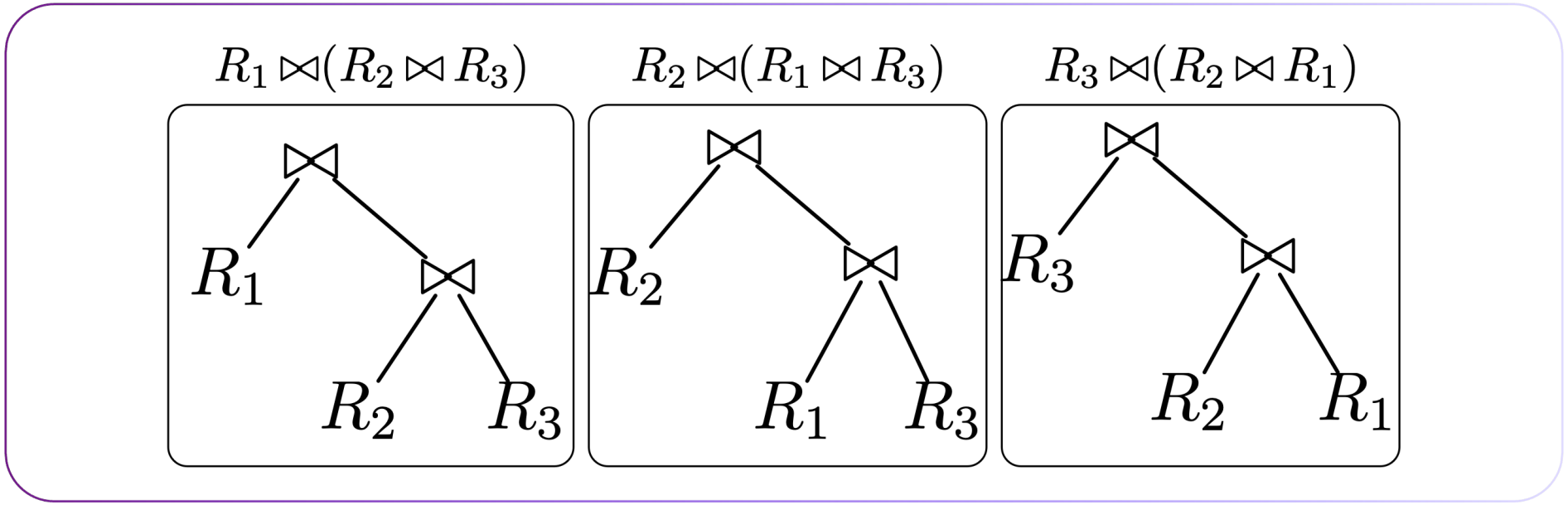
序号	等价转换规则
19	若选择条件p只涉及投影S中的属性，$\sigma_p(\Pi_S(R)) = \Pi_S(\sigma_p(R))$
20	$\Pi_{S_1 \cup S_2}(R_1 \times R_2) = \Pi_{S_1}(R_1) \times \Pi_{S_2}(R_2)$, 其中投影属性集合$S_1$和$S_2$分别是$R_1$和$R_2$中的属性
21	$\Pi_S(R_1 \cup R_2) = \Pi_S(R_1) \cup \Pi_S(R_2)$
22	$\sigma_p(R_1 \times R_2) = R_1 \bowtie_p R_2$
23	等值连接$R_1 \bowtie R_2 = \Pi_S(\sigma_p(R_1 \times R_2))$，其中$S$为$R_1, R_2$属性的并集
24	对于没有重复元素的关系R，$\delta(R) = R$
25	$\delta(R_1 \times R_2) = \delta(R_1) \times \delta(R_2)$
26	$\delta(R_1 \bowtie R_2) = \delta(R_1) \bowtie \delta(R_2)$

查询重写规则

序号	等价转换规则
27	$\delta(R_1 \bowtie_p R_2) = \delta(R_1) \bowtie_p \delta(R_2)$
28	$\delta(\sigma_p(R)) = \sigma_p(\delta(R))$
29	$\delta(R_1 \cap R_2) = \delta(R_1) \cap \delta(R_2) = R_1 \cap \delta(R_2) = \delta(R_1) \cap R_2$
30	当聚集函数C是取最小值或取最大值时，$\delta(\mathcal{G}_C(R)) = \mathcal{G}_C(R)$
31	$\mathcal{G}_C(R) = \mathcal{G}_C(\Pi_S(R))$，其中$C \subseteq S$

交换律与结合律

- **交换律：** $R_1 \bowtie R_2 = R_2 \bowtie R_1$
- **结合律：** $(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$
- **也适用于笛卡尔积、并运算、交运算**



交换律与结合律

对于**笛卡尔积**运算：

- $R_1 \times R_2 = R_2 \times R_1$
- $(R_1 \times R_2) \times R_3 = R_1 \times (R_2 \times R_3)$

对于**并**运算：

- $R_1 \cup R_2 = R_2 \cup R_1$
- $(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$

对于**连接**运算：

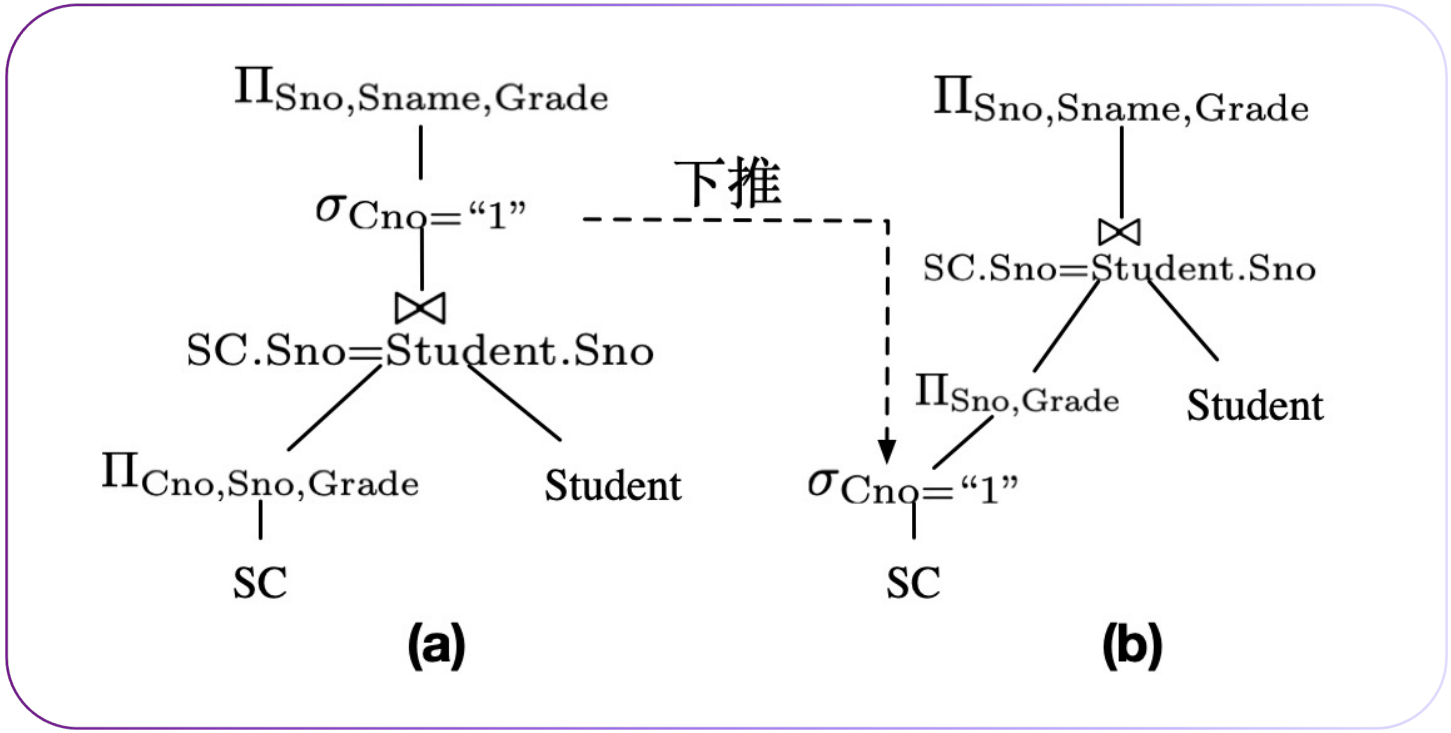
- $R_1 \bowtie R_2 = R_2 \bowtie R_1$
- $(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$

对于**交**运算：

- $R_1 \cap R_2 = R_2 \cap R_1$
- $(R_1 \cap R_2) \cap R_3 = R_1 \cap (R_2 \cap R_3)$

选择下推

- 将选择运算尽可能**下推**，一般可以对中间关系的大小进行缩减，从而减少后续磁盘读取以及运算次数，由此达到加速查询执行的目的。



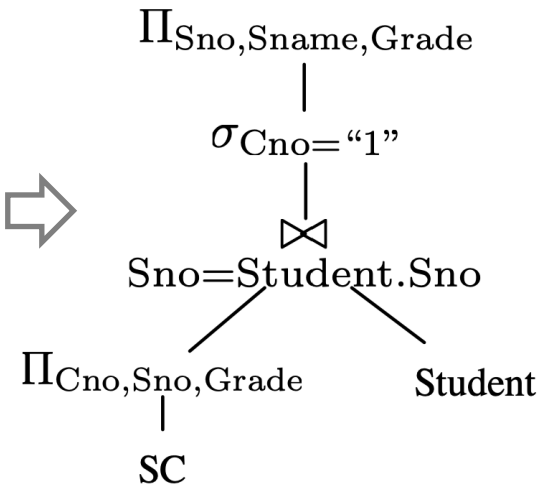
选择下推示例（1）

查询：获取选修了1号课程的学生学号、姓名及成绩

- **学生表 Student(Sno ,Sname, Sgender, Sage, Sdept)**
- **学生选课表 SC(Sno, Cno, Grade)**

```
SELECT Sno,  Sname, Grade
FROM SC, Student
WHERE Sno=Student.Sno
AND Cno="1"
```

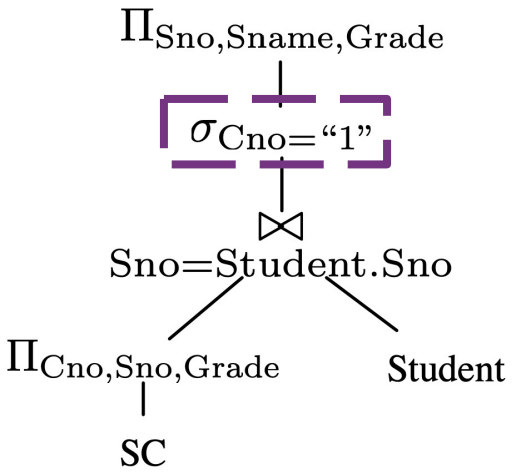
$$\Pi_{Sno,\ Sname,\ Grade}\left(\sigma_{Cno="1"}\ (SC \bowtie Student)\right)$$



SQL **关系代数表达式** **逻辑计划树**

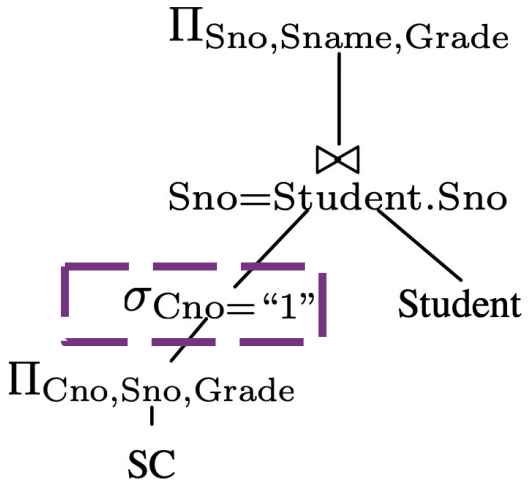
选择下推示例（2）

查询：获取选修了1号课程的学生学号、姓名及成绩



利用等价规则：

$$\sigma_p(R_1 \bowtie R_2) = \sigma_p(R_1) \bowtie R_2$$

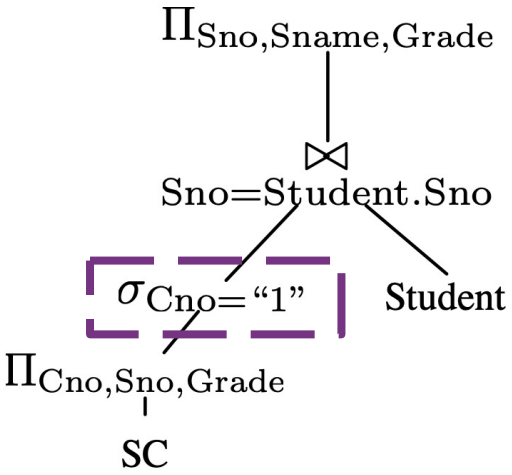


$$\Pi_{Sno,\,Sname,\,Grade}(\sigma_{Cno="1"}\ (SC \bowtie Student))$$

$$\Pi_{Sno,\,Sname,\,Grade}(\sigma_{Cno="1"}\ (\Pi_{Cno,Sno,Grade}SC) \bowtie Student)$$

选择下推示例（3）

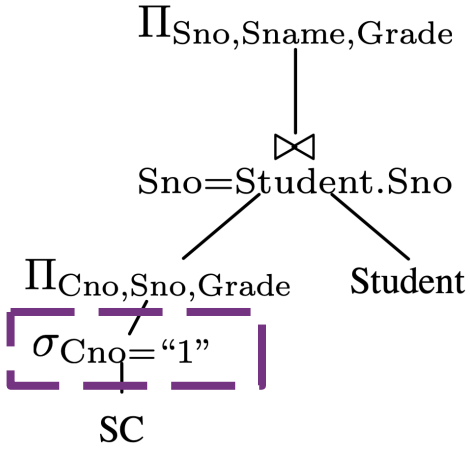
查询：获取选修了1号课程的学生学号、姓名及成绩



利用等价规则：

$\sigma_p(\Pi_S(R)) = \Pi_S(\sigma_p(R))$,

选择条件p只涉及投影S中的属性



$\Pi_{Sno,\,Sname,\,Grade}(\sigma_{Cno="1"}(\Pi_{Cno,Sno,Grade}SC) \bowtie Student)$

$\Pi_{Sno,\,Sname,\,Grade}((\Pi_{Cno,Sno,Grade}(\sigma_{Cno="1"}\,SC)) \bowtie Student)$

选择运算的等价规则

- $\sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2)$
- $\sigma_p(R_1 - R_2) = \sigma_p(R_1) - R_2$
- $\sigma_p(R_1 - R_2) = \sigma_p(R_1) - \sigma_p(R_2)$

若关系R_1包含选择条件p涉及的所有属性：

- $\sigma_p(R_1 \times R_2) = \sigma_p(R_1) \times R_2$
- $\sigma_p(R_1 \bowtie R_2) = \sigma_p(R_1) \bowtie R_2$
- $\sigma_p(R_1 \cap R_2) = \sigma_p(R_1) \cap R_2$

- $\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_1}\left(\sigma_{p_2}(R)\right)$
- $\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_2}\left(\sigma_{p_1}(R)\right)$
- $\sigma_p\left(\Pi_S(R)\right) = \Pi_S(\sigma_p(R))$，*其中选择*条件$p$只涉及投影$S$中的属性

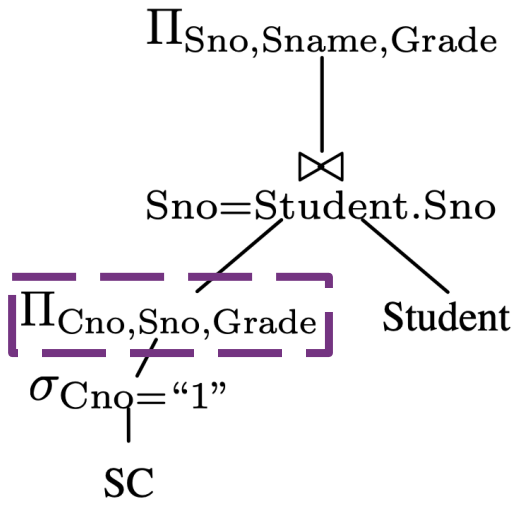
  当R无重复元组时：
- $\sigma_{p_1 \vee p_2}(R) = \sigma_{p_1}(R) \cup \sigma_{p_2}(R)$

投影下推

- 与选择运算类似，投影运算也可以进行下推，从而减少中间结果列的数量，降低中间结果的数据规模。
- 比起优化选择运算，优化投影运算所能提升的效率会小些。
- 这是因为选择运算的过滤作用通常会使得中间结果的行数大幅度下降，而投影运算减少的是元组的列数（即属性数量），对中间关系的规模影响有限。

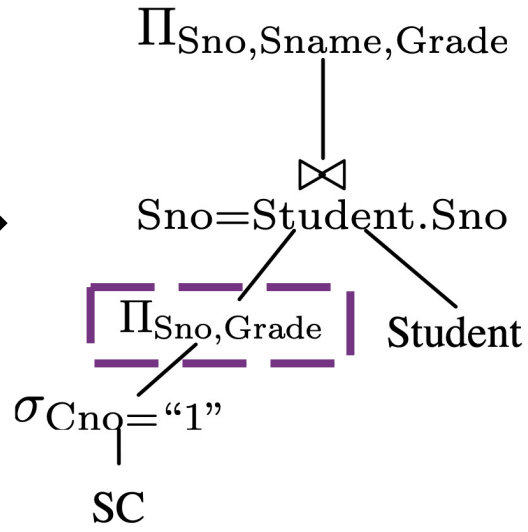
投影下推示例

查询：获取选修了1号课程的学生学号、姓名及成绩



利用等价规则：

$\Pi_{S_1}\left(\Pi_{S_2}(R)\right)=\Pi_{S_1}(R), S_1 \subseteq S_2$且都是关系$R$的属性



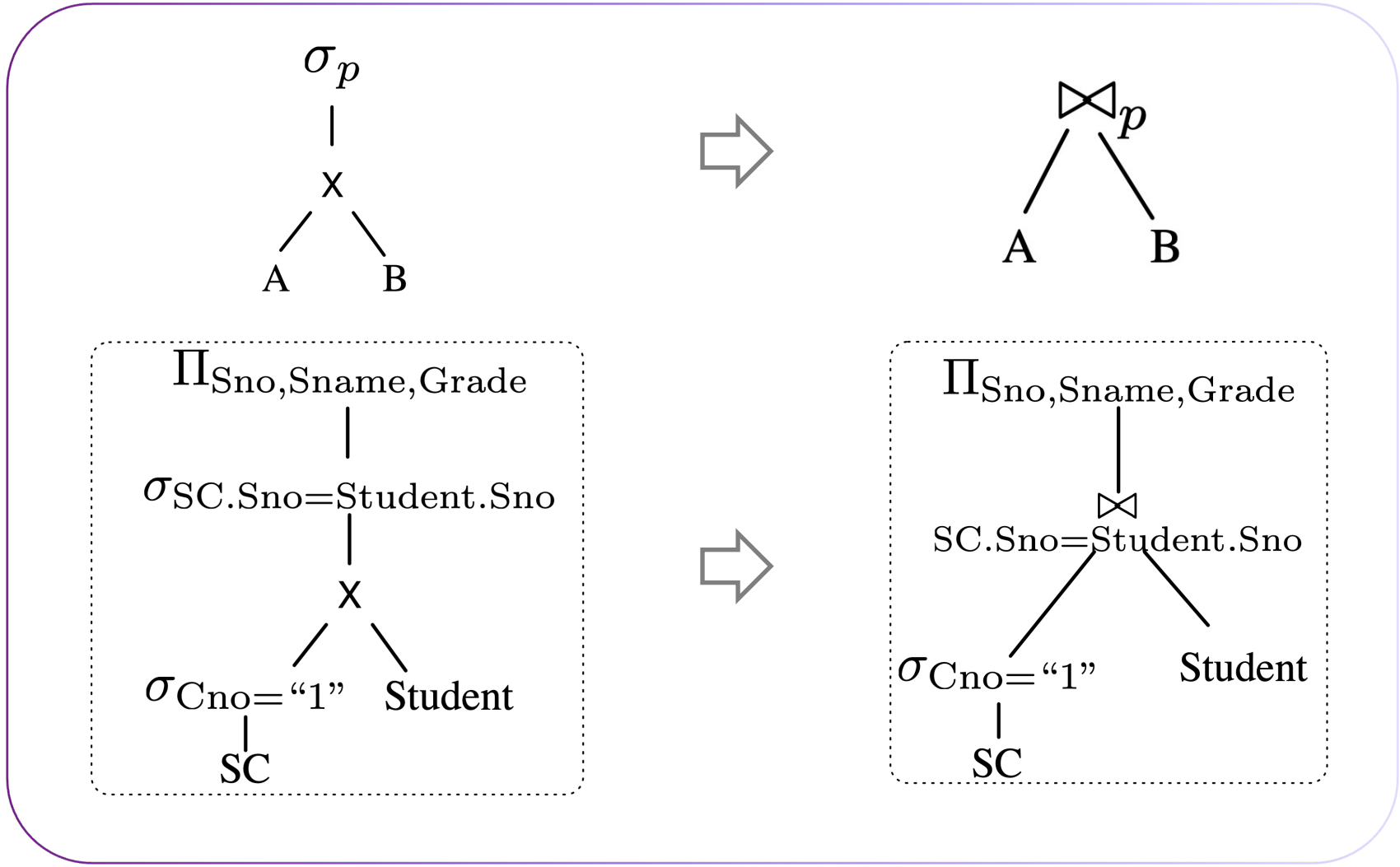
$\Pi_{\text{Sno, Sname, Grade}}\left(\left(\Pi_{\text{Cno,Sno,Grade}}(\sigma_{\text{Cno="1"}}\ \text{SC})\right) \bowtie \text{Student}\right)$

$\Pi_{\text{Sno, Sname, Grade}}\left(\left(\Pi_{\text{Sno,Grade}}(\sigma_{\text{Cno="1"}}\ \text{SC})\right) \bowtie \text{Student}\right)$

投影运算的等价规则

- $\Pi_{S_1}\left(\Pi_{S_2}(R)\right) = \Pi_{S_1}(R), S_1 \subseteq S_2$且都是关系R 的属性
- $\sigma_p\left(\Pi_S(R)\right) = \Pi_S(\sigma_p(R))$, 其中选择条件p只涉及投影S中的属性
- $\Pi_{S_1 \cup S_2}(R_1 \times R_2) = \Pi_{S_1}(R_1) \times \Pi_{S_2}(R_2)$, 其中投影属性集合$S_1$和$S_2$分别是$R_1$和$R_2$的属性
- $\Pi_S(R_1 \cup R_2) = \Pi_S(R_1) \cup \Pi_S(R_2)$, 其中投影属性集合S是关系$R_1$和$R_2$的属性

笛卡尔积转连接示例


σ_p
X
A
B
p
A
B
$\Pi_{Sno,Sname,Grade}$
$\sigma_{SC.Sno=Student.Sno}$
X
$\sigma_{Cno=“1”}$
Student
SC
$\Pi_{Sno,Sname,Grade}$
SC.Sno=Student.Sno
$\sigma_{Cno=“1”}$
Student
SC

连接与笛卡尔积运算的等价规则

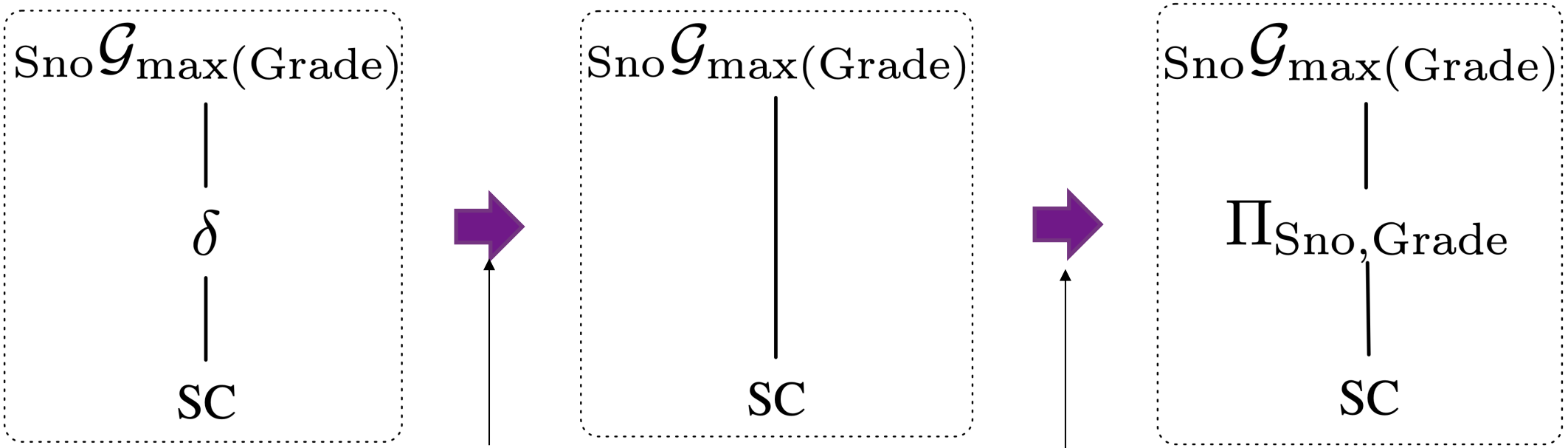
- 条件连接：条件连接是先对关系R_1、R_2做笛卡尔积，再按条件p做选择运算，因此可以得到：

$$\sigma_p(R_1 \times R_2) = R_1 \bowtie_p R_2$$

- 等值连接：等值连接是特殊的条件连接。等值连接中的条件p要求关系R_1, R_2中的连接属性按照等值比较作为选择条件。
- 自然连接：是一种特殊的等值连接。要求等值连接属性必须是同名属性，并在连接结果中去掉重复的属性。因此可以得到：

$$R_1 \bowtie R_2 = \Pi_S(\sigma_p(R_1 \times R_2))$$

分组聚集与去重的重写示例



去重运算的等价规则

- 按照定义，对于没有重复元组的关系R:

$$\delta(R) = R$$

- 其他下推去重算子δ的等价规则有:

$$\delta(R_1 \times R_2) = \delta(R_1) \times \delta(R_2)$$

$$\delta(R_1 \bowtie R_2) = \delta(R_1) \bowtie \delta(R_2)$$

$$\delta(R_1 \bowtie_p R_2) = \delta(R_1) \bowtie_p \delta(R_2)$$

$$\delta(\sigma_p(R)) = \sigma_p(\delta(R))$$

$$\delta(R_1 \cap R_2) = \delta(R_1) \cap \delta(R_2) = R_1 \cap \delta(R_2) = \delta(R_1) \cap R_2$$

分组聚集运算的等价规则

- 根据定义$\mathcal{G}$最终会产生一个无重复元组的关系，利用去重运算的规则：

$$\delta(\mathcal{G}_C(R)) = \mathcal{G}_C(R)$$

- 当聚集函数是取最小值或取最大值时，$\mathcal{G}$的运算结果与输入关系是否去过重无关：

$$\mathcal{G}_C(R) = \mathcal{G}_C(\delta(R))$$

- 可以作满足投影属性集合S包含$\mathcal{G}$所需要的所有属性，也就是满足C ⊆ S的投影：

$$\mathcal{G}_C(R) = \mathcal{G}_C(\Pi_S(R))$$

重写顺序

- 重写规则数量很多，重写时逐个枚举会产生巨大的搜索空间
- 一般按照特定的规则枚举顺序进行枚举，并在枚举一定数量后就停止搜索
  - 按照重写规则的重要性
- 通过优先利用那些大概率能提升执行效率的等价规则，可以降低搜索空间
- 重写规则顺序会影响重写质量

重写顺序

- **对规则的枚举顺序一般为：**

① 分解复合的选择谓词，如$\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_1}\left(\sigma_{p_2}(R)\right)$

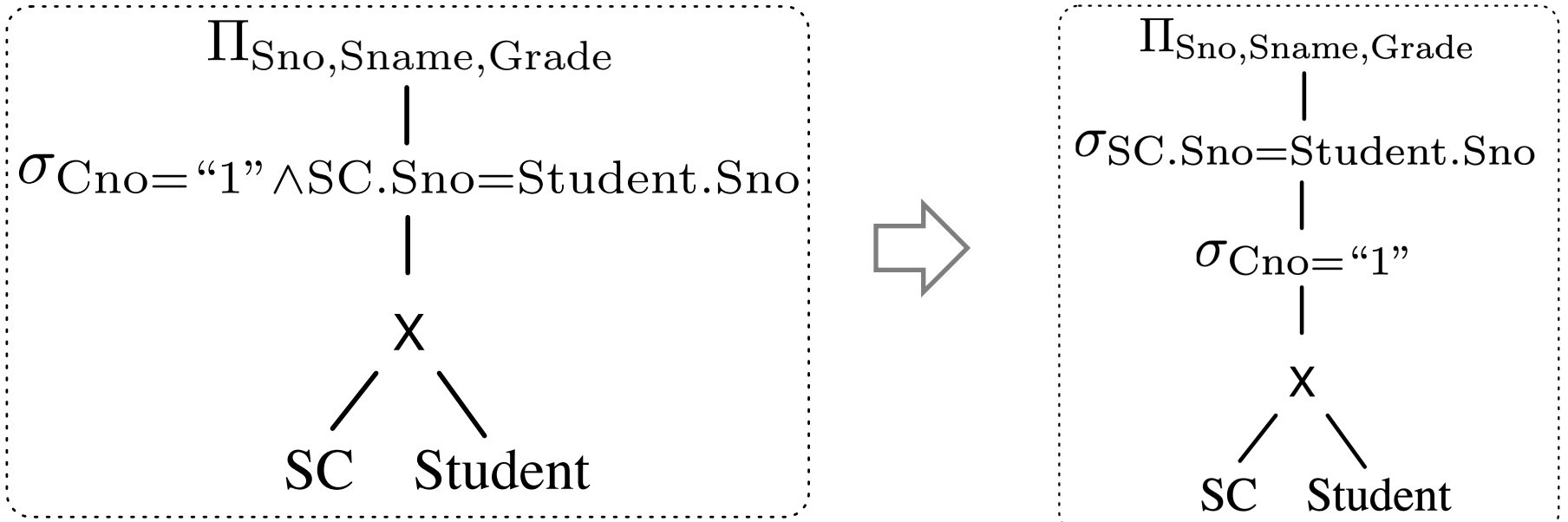
② 添加推理的谓词，例如 A=B，A>5 → B > 5

③ 下推选择运算，如$\sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2)$

④ 将笛卡尔积转换为连接运算，如$\sigma_p(R_1 \times R_2) = R_1 \bowtie_p R_2$

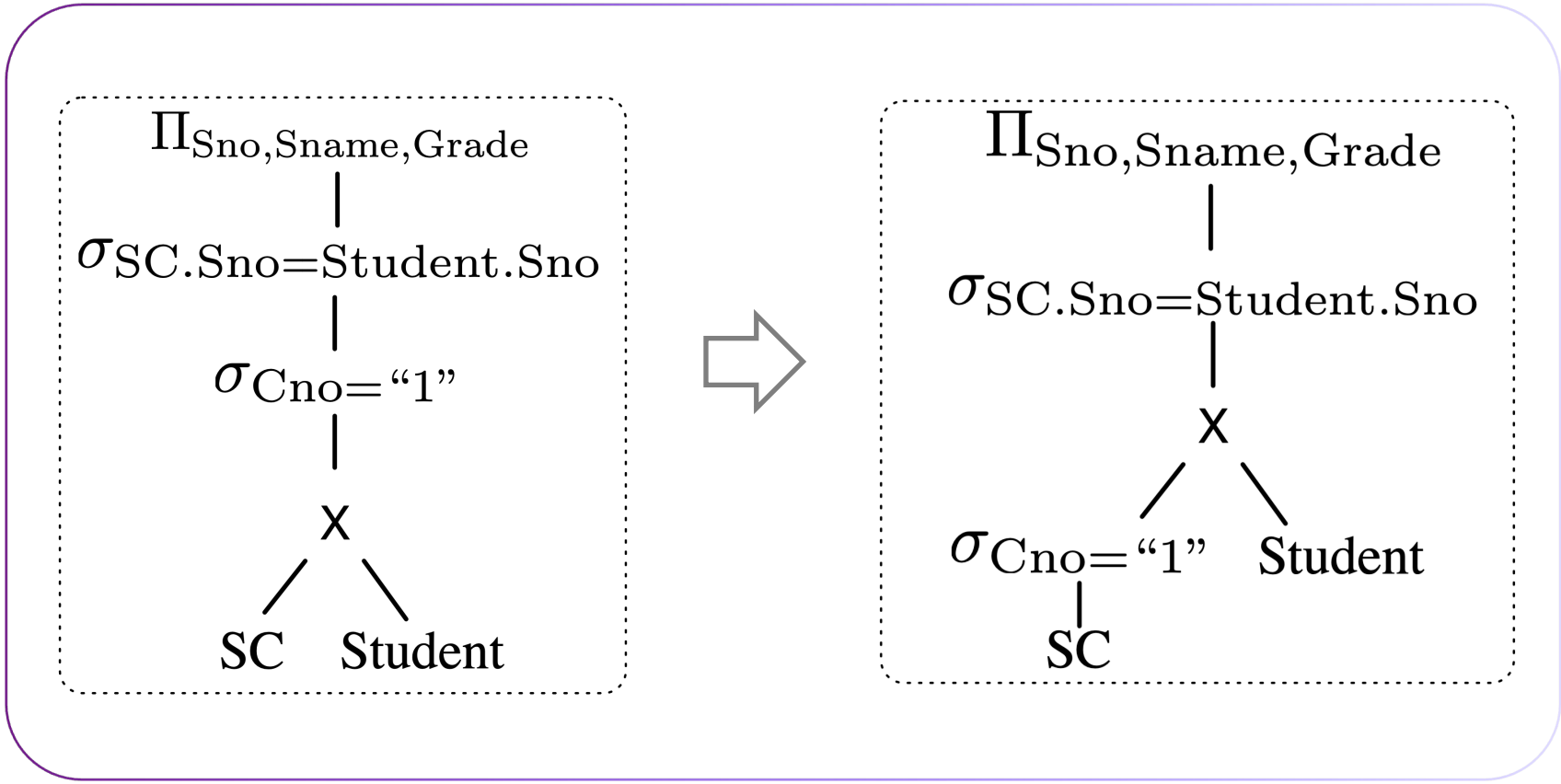
⑤ 引入并下推投影运算，如$\Pi_{S_1}\left(\Pi_{S_2}(R)\right) = \Pi_{S_1}(R), S_1 \subseteq S_2$且都是关系R的属性

步骤1 分解复合的选择谓词

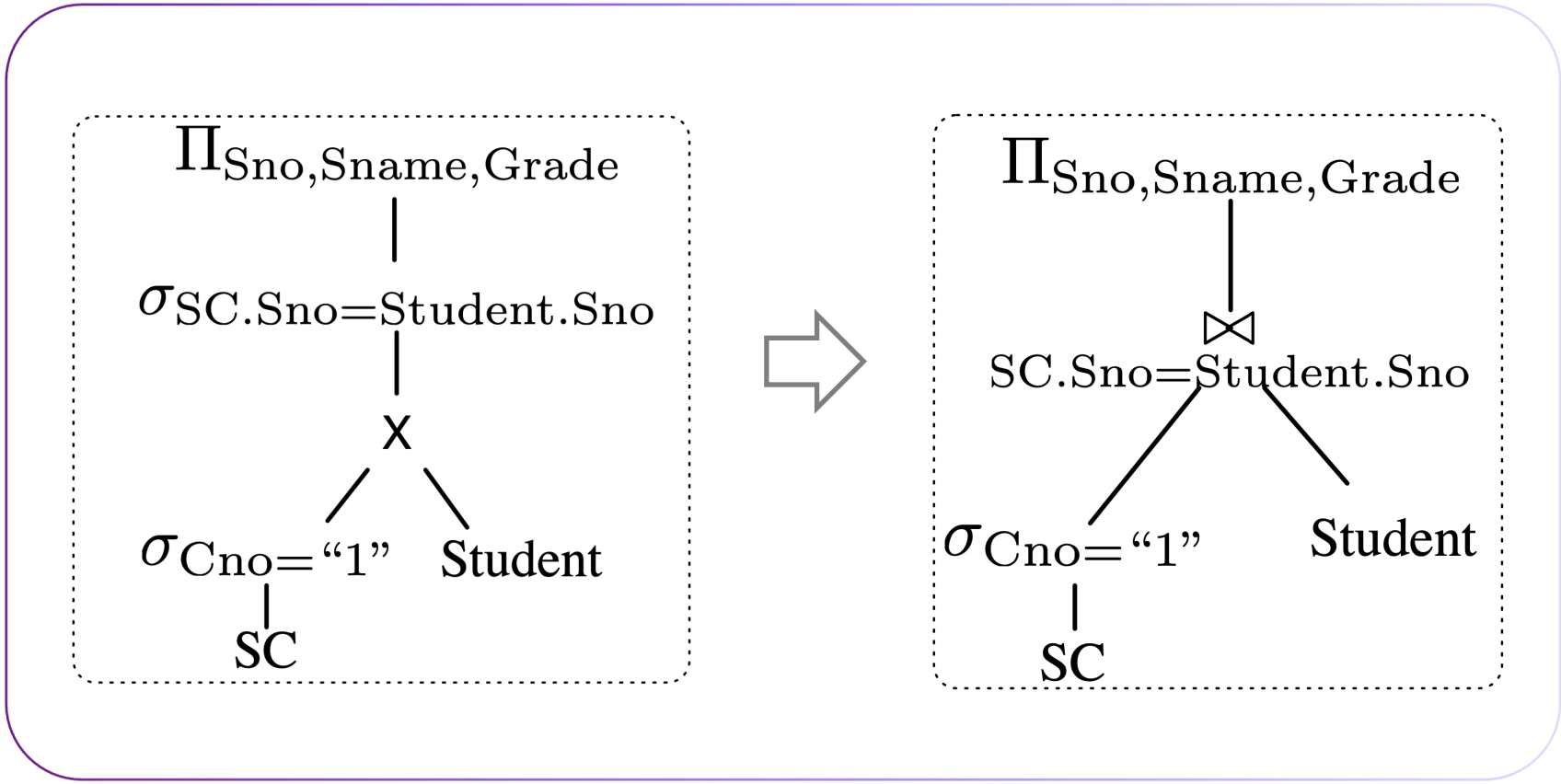


最初的逻辑计划

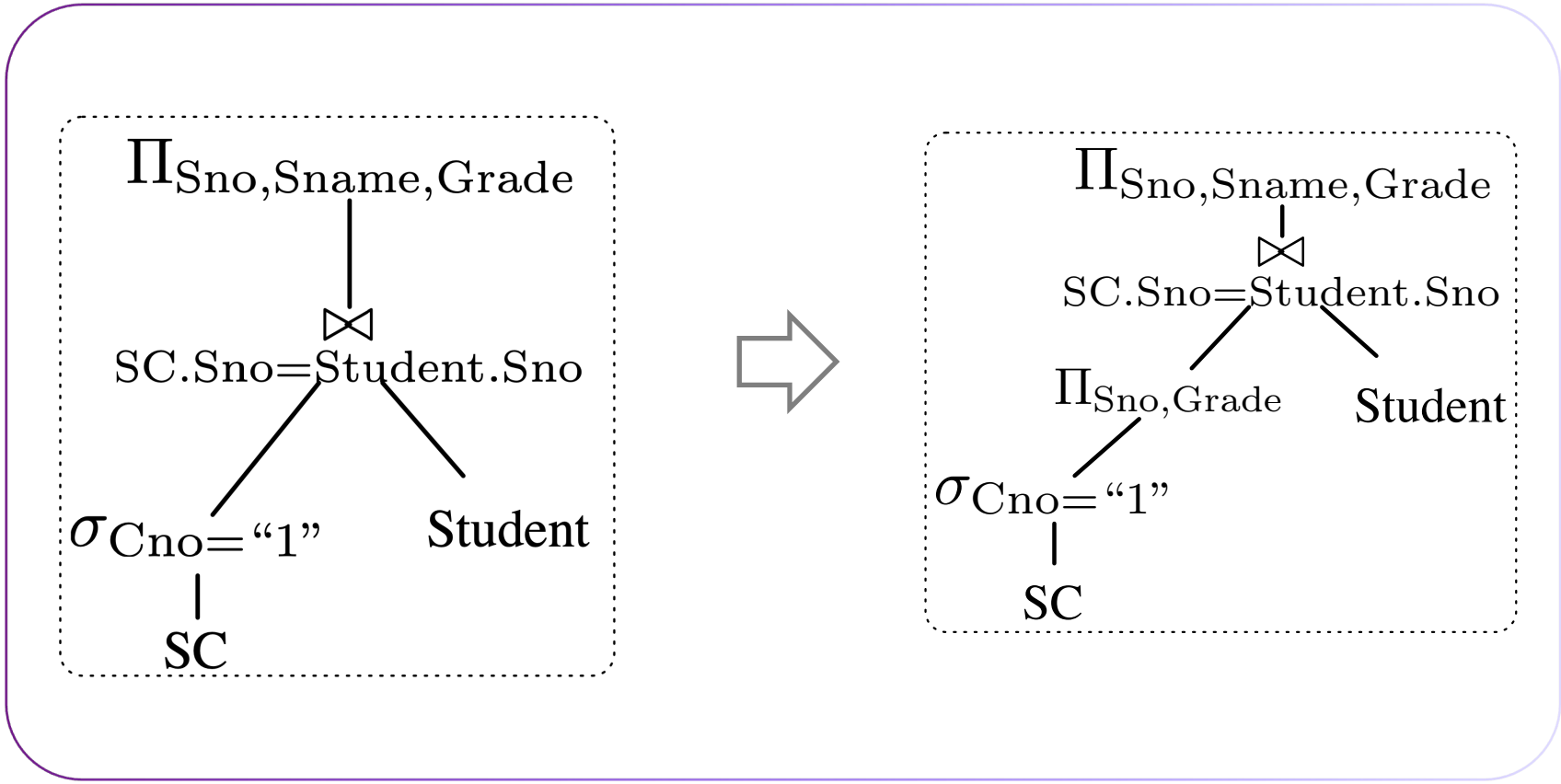
步骤2 下推选择运算


$\Pi_{Sno,Sname,Grade}$
$\sigma_{SC.Sno=Student.Sno}$
$\sigma_{Cno=“1”}$
X
SC
Student
$\Pi_{Sno,Sname,Grade}$
$\sigma_{SC.Sno=Student.Sno}$
X
$\sigma_{Cno=“1”}$
Student
SC

步骤3 将笛卡尔积转换为连接



步骤4 引入以及下推投影运算



其他重写策略：子查询转为连接

查询选修了1号课程的学生学号、姓名及所在系

```
SELECT Sno, Sname, Sdept
FROM Student
WHERE Sno IN
   (SELECT Sno
   FROM SC
   WHERE Cno="1");
```

```
SELECT Student.Sno,Sname,Sdept
FROM Student, SC
WHERE Student.Sno=SC.Sno
     AND Cno="1";
```

其他重写策略：常数传播

查询学号小于100的学生学号、姓名及成绩

```
SELECT SC.Sno, Student.Sname, SC.Grade
FROM Student, SC
WHERE Student.Sno=SC.Sno AND Student.
Sno<100;
```

```
SELECT SC.Sno, Student.Sname, SC.Grade
FROM Student, SC
WHERE Student.Sno=SC.Sno AND Student.
Sno<100
AND SC.Sno<100;
```

其他重写策略：去除非必要谓词

查询1号课程的最高分

```
SELECT MAX(DISTINCT Grade)
FROM SC
WHERE Cno="1";
```

```
SELECT MAX(Grade)
FROM SC
WHERE Cno="1";
```

查询重写总结

- 关系代数是查询重写中的重要抽象
- 查询重写的规则质量很重要
- 查询重写规则使用顺序也很重要
- 许多等价规则被用于搜索“好”的计划
- 重写规则也不是越多越好，可能影响重写时间
- 重写规则的匹配方法也很重要

目录

代价估计

- **查询重写过后，还需确定：**
  - 符合交换律和结合律的运算的执行顺序
  - 逻辑计划中每个算子在物理执行时采用的算法
  - 逻辑计划中未体现出的物理运算，例如排序和扫描
  - 在运算符之间数据传递的方式
- **利用代价模型（cost model）比较不同执行计划的优劣**
- **代价估计模型需考虑：**
  - 统计信息：数据分布、行数、Distinct值个数
  - 统计信息收集时机、方法：不能影响在线业务；采样收集
  - 基数估计→代价估计：基数估计来估算中间结果个数，从而用于代价估计

基数估计

- 代价估计的核心是**基数估计（cardinality estimation）**，对于谓词p组成的查询：
- **基数**（cardinality）：查询结果的元组数。表示为 $\mathrm{card}(p)$。
- **选择度**（selectivity）：符合p元组占所有元组的比例,表示为$\mathrm{sel}(p)$。

Sno（学号）	Sname（姓名）	Sgender（性别）	Sage（年龄）	Sdept（所在系）
2021310721	李博	男	17	CS
2021310722	赵宇	男	19	CS
2021310723	张敏	女	18	CS
2021310724	王勇	男	18	MA
2021310725	刘佳	女	17	MA

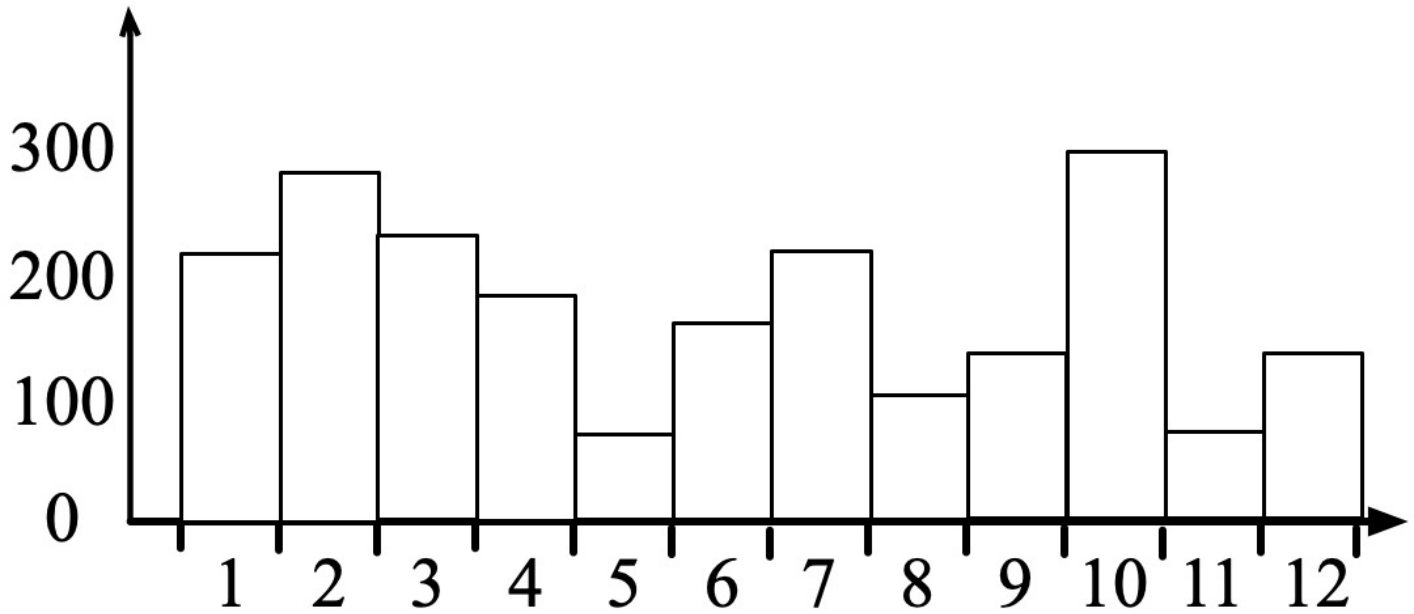
- 查询“计算机系所有学生的信息”，$\sigma_{\mathrm{Sdept}=\text{“CS”}}(\mathrm{Student})$。
- 5条记录中共有3条符合查询谓词限制。因此基数为3，选择度$\frac{3}{5}$。

统计数据

统计信息	说明
$T(R)$	关系R的元组数目
$B(R)$	关系R的所有元组需要的页面数
$L(R)$	关系R一个元组平均所需的字节数
$f(R)$	一个页面能存储的关系R的元组数
$\min(R,A)$	关系R在属性A上取值的最小值
$\max(R,A)$	关系R在属性A上取值的最大值
$V(R,A)$	关系R在属性A上不同值个数。当A扩展为属性集合$\mathcal{A}$时，$V(R,\mathcal{A})$表示关系R在属性集$\mathcal{A}$上投影后不同的元组数，即$\Pi_{\mathcal{A}}(R)$元组数
$\mathrm{MCV}(R,A)$	关系R在属性A上取值最频繁的topk（most common value）

直方图

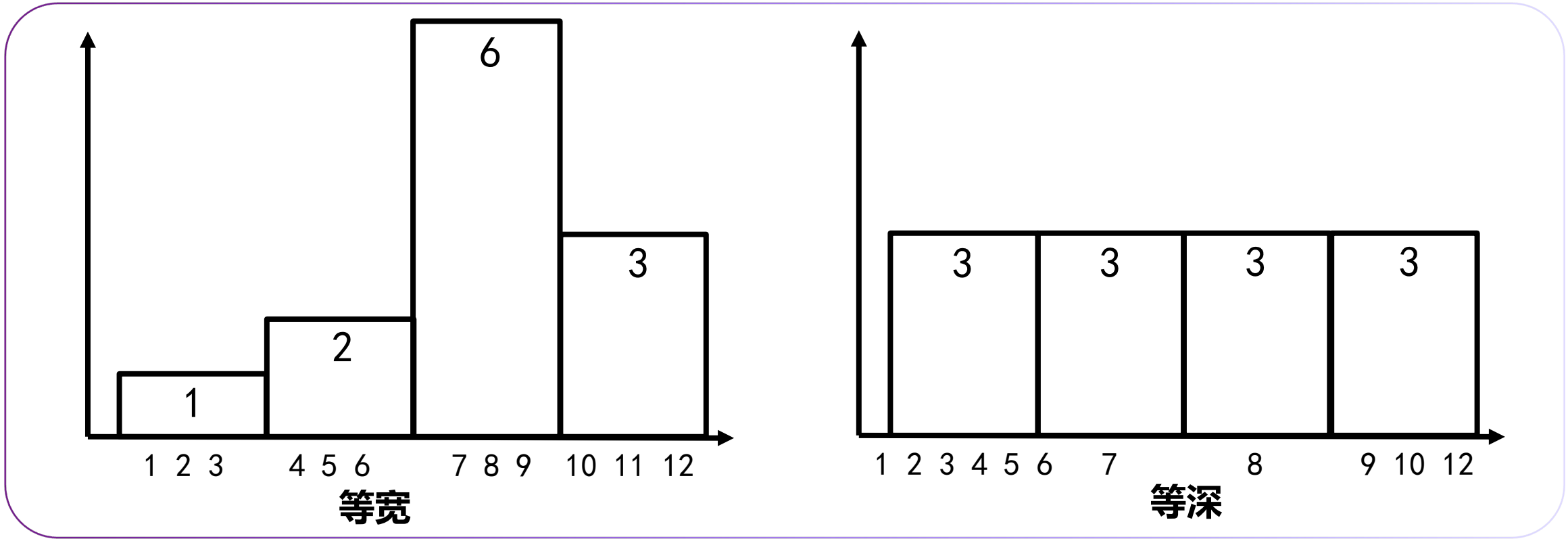
- 直方图（histogram）：一组区间（桶）以及这些区间中取值出现的频率



示例，对月份属性建立直方图

直方图分类

- 等深直方图（equal-depth histogram）：按属性不同值出现频率调整区间宽度
- 等宽直方图（equal-width histogram）：将属性按照相同间距划分区间
- 数据: 1、4、6、7、7、7、8、8、8、9、10、12



统计数据假设

利用统计数据进行估计依赖着以下假设：

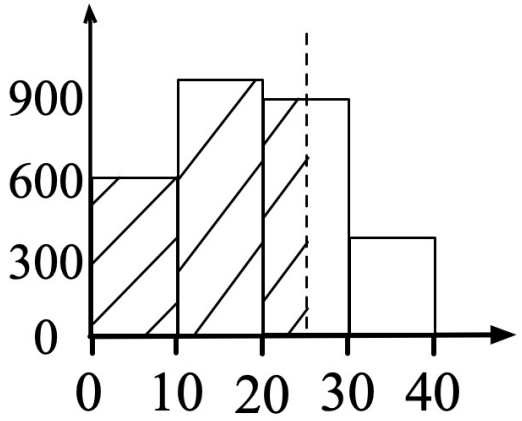
- **均匀分布假设（uniform distribution assumption）：**
  - 认为属性内部取值是均匀分布的
- **属性独立假设（attribute independence assumption）：**
  - 认为不同属性的取值之间相互独立

选择运算的基数估计

- **等值谓词，$\sigma_{A=x}(R)$：$T(R)/V(R,A)$**
- **范围谓词，$\sigma_{A\leq x}(R)$：**
  - 若$x < \min(R,A)$，以0作为估计
  - 若$x \geq \max(R,A)$，以$T(R)$作为估计
  - 若x落在$[\min(R,A), \max(R,A)]$，以$T(R)\cdot\frac{x-\min(R,A)}{\max(R,A)-\min(R,A)}$作为估计
  - 对于$<, >, \geq$等其他的运算符以此类推

选择运算的基数估计

- **使用直方图进行估计**
  - $T(R) \cdot \text{freq}(A \leq x)$
- **举例： 利用直方图估计范围谓词$A \leq 25$ 的结果基数**



$$\text{freq}(A \leq 25) = (T(1 \leq A \leq 10) + T(10 < A \leq 20) + \cdot 0.5 \cdot T(20 < A \leq 30))/T(R)$$

选择运算的基数估计

- **取反**
  - 取反的一般形式为：$\sigma_{\neg p}(R)$
  - 以$\mathrm{card}(\sigma_{\neg p}) = T(R) - \mathrm{card}(\sigma_p)$作为取反的基数估计
- **合取**
  - 合取的一般形式为：$\sigma_{p_1 \wedge \cdots \wedge p_k}(R)$，表示同时符合谓词$\sigma_{p_i}$，$1 \le i \le k$

$$\mathrm{card}(\sigma_{p_1 \wedge \cdots \wedge p_k}) = T(R) \cdot \mathrm{sel}(\sigma_{p_1 \wedge \cdots \wedge p_k})$$

$$= T(R) \cdot \frac{\mathrm{card}(\sigma_{p_1}) \times \cdots \times \mathrm{card}(\sigma_{p_k})}{T(R)^k}$$

选择运算的基数估计

➢ 析取

- 析取的一般形式为：$\sigma_{p_1 \vee \cdots \vee p_k}(R)$，表示至少符合谓词$\sigma_{p_i}, 1 \le i \le k$中的一个

$$\begin{aligned}
&\mathrm{sel}\left(\sigma_{p_1 \vee \cdots \vee p_k}\right) = 1 - \prod_i (1 - \mathrm{sel}_i) \\
&= 1 - (1 - \mathrm{sel}_1) \times \cdots \times (1 - \mathrm{sel}_k) \\
&= 1 - \frac{\left(T(R) - \mathrm{card}\left(\sigma_{p_1}\right)\right) \times \cdots \times \left(T(R) - \mathrm{card}\left(\sigma_{p_k}\right)\right)}{T(R)^k}
\end{aligned}$$

$$\begin{aligned}
&\mathrm{card}\left(\sigma_{p_1 \vee \cdots \vee p_k}\right) = T(R) \cdot \mathrm{sel}\left(\sigma_{p_1 \vee \cdots \vee p_k}\right) \\
&= T(R) \cdot \left(1 - \frac{\left(T(R) - \mathrm{card}\left(\sigma_{p_1}\right)\right) \times \cdots \times \left(T(R) - \mathrm{card}\left(\sigma_{p_k}\right)\right)}{T(R)^k}\right)
\end{aligned}$$

选择运算的基数估计

- **对中间结果在属性A上的不同取值个数$V(\sigma_p(R), A)$ 的估计**
  - 如果属性A在选择条件p中出现，利用均匀分布假设，可以估计

$$V\left(\sigma_p(R), A\right) = \mathrm{sel}(\sigma_p) \cdot V(R, A)$$

  - 如果属性A没有在选择条件p中出现，利用属性独立假设，属性A的取值与选择条件无关，可以估计

$$V\left(\sigma_p(R), A\right) = \min\left(V(R, A), T\left(\sigma_p(R)\right)\right)$$

选择运算的基数估计示例

- **考虑有三个属性的关系$\mathbf{R(A,B,C)}$**
  - 其中$T(R)=10000$，$V(R,A)=5$，$V(R,B)$=10，$V(R,C)=20$
  - 假设关系R在属性A,B,C上的取值分别为$\{1,2,\cdots 5\}$，$\{1,2,\cdots 10\}$，$\{1,2,\cdots 20\}$，且完全符合**属性独立假设**和**均匀分布假设**
- **对于关系代数表达式$\sigma_{\mathbf{A=5\wedge B\leq 5}}(\mathbf{R})$可以进行如下估计**
- **首先，对两个选择谓词的基数估计为，**
  - $\mathrm{card}(\sigma_{A=5})=\frac{T(R)}{V(R,A)}=2000$
  - $\mathrm{card}(\sigma_{B\leq 5})=T(R)\times\frac{1}{2}=5000$

选择运算的基数估计示例

- **利用上文介绍的合取估计方法**
  - $\mathrm{card}(\sigma_{A=5\wedge B\le 5}) = T(R)\cdot\frac{card(\sigma_{A=5})\cdot card(\sigma_{B\le 5})}{\mathrm{T}(R)^2}$
    - $= 10000\times\frac{2000\times 5000}{10000\times 10000} = 1000$
- **用R'表示结果关系$\sigma_{\mathbf{A=5\wedge B\le 5}}(\boldsymbol{R})$，可以得到**
  - $V(R',A) = \mathrm{sel}(\sigma_{A=5})\cdot V(R,A) = \frac{1}{5}\times 5 = 1$
  - $V(R',B) = \mathrm{sel}(\sigma_{B\le 5})\cdot V(R,B) = \frac{1}{2}\times 10 = 5$
  - $V(R',C) = \min\big(V(R,C), T(\sigma_{A=5\wedge B\le 5})\big) = \min(20,1000) = 20$

连接运算的基数估计

➢ **情况1: 两个关系在单属性上连接**

- 考虑两个关系$R_1(A,B)$和$R_2(A,C)$。
- 属性A为这两个关系进行自然连接的属性，B和C分别是关系R_1和R_2的非连接属性。
- 要进行结果基数估计的自然连接是$R_1(A,B) \bowtie R_2(A,C)$
- 使用A_1, A_2来分别表示关系R_1和R_2中属性A的取值集合

连接运算的基数估计

- **难点：**
  - 连接的基数深受$|A_1 \cap A_2|$的影响
  - 优化器并不清楚A_1和A_2的关联情况
- **例如：**
  - 当$A_1 \cap A_2 = \emptyset$时，$T(R_1 \bowtie R_2) = 0$
  - 当R_1和R_2在A上都只取同一个值时，$T(R_1 \bowtie R_2) = T(R_1) \times T(R_2)$
  - 当属性A是R_1的主键，是R_2的外键，也就是主键-外键连接时，$T(R_1 \bowtie R_2) = T(R_2)$

连接运算的基数估计

- **A_1和A_2的关联情况在查询执行时才能确定**
- **为了在查询执行前估计，优化器作出以下简化假设：**

  - **假设1**：对于关系R_1, R_2的连接属性A，若$V(R_2, A) \leq V(R_1, A)$，则假设$A_2 \subseteq A_1$
  - **假设2**：对于不是关系R_1, R_2中共有且不参与连接的属性W，不妨设W是关系R_2中的属性，则有$V(R_1 \bowtie R_2, W) = V(R_2, W)$。
    - 认为不参与连接的属性的取值个数不会因为连接而减少。

在属性A是R_1的主键，并且是R_2的外键的**主键-外键**连接的情况下成立

连接运算的基数估计

- **基于以上假设，可以对$R_1 \bowtie R_2$的基数估计作下述推导：**

- 根据**对称性**，不妨设$V(R_2, A) \leq V(R_1, A)$
- 根据**假设1**，$A_2 \subseteq A_1$，因此A_2的任何一个取值都能在A_1中找到匹配
- 根据**均匀分布假设**，关系R_1中属性A的每一个取值出现的频率均为$1/V(R_1, A)$
- 因此对于关系R_2中的$T(R_2)$个元组，每一个都能在R_1中找到$T(R_1)/V(R_1, A)$个匹配
- 所以连接结果的大小估计值为$\frac{T(R_1) \times T(R_2)}{V(R_1, A)}$

连接运算的基数估计

- 根据**对称性**，当$V(R_2, A) > V(R_1, A)$时同样可以进行上述分析

- 综上所述，对连接运算的基数估计为：

- $T\big(R_1(A,B) \bowtie R_2(A,C)\big) = \frac{T(R_1) \times T(R_2)}{\max(V(R_1,A), V(R_2,A))}$

- 并且根据**假设1**还可以得到：

- $V(R_1 \bowtie R_2, A) = \min\big(V(R_1, A), V(R_2, A)\big)$

连接运算的基数估计

- **对于关系R_1, R_2中公有的属性A**
  - $V(R_1 \bowtie R_2, \mathrm{A}) = \min\big(V(R_1, A), V(R_2, A)\big)$
- **对于其它并非关系R_1, R_2中公有，因而不参与连接的属性W，不妨设W是关系R_2中的属性**
  - $V(R_1 \bowtie R_2, W) = V(R_2, W)$
- **对连接结果的基数估计**
  - $T\big(R_1(A,B) \bowtie R_2(A,C)\big) = \frac{T(R_1)\times T(R_2)}{\max(V(R_1,A),V(R_2,A))}$

连接运算的基数估计示例

- 三个关系R_1, R_2, R_3在属性A上进行自然连接，估计连接结果的基数

关系	$T(R_i)$	$V(R_i, A)$
R_1	100	10
R_2	500	20
R_3	1000	50

- 对$(R_1 \bowtie R_2) \bowtie R_3$和$R_1 \bowtie (R_2 \bowtie R_3)$两种连接顺序分别进行估计

连接运算的基数估计示例

- **对$(R_1 \bowtie R_2) \bowtie R_3$ 的连接顺序进行估计**
  - $T(R_1 \bowtie R_2) = \frac{T(R_1) \times T(R_2)}{max(V(R_1,A),V(R_2,A))} = \frac{100 \times 500}{20} = 2500$
  - 用R'表示$R_1 \bowtie R_2$，$V(R',A) = min\big(V(R_1,A),V(R_2,A)\big) = 10$
  - $T(R' \bowtie R_3) = \frac{T(R') \times T(R_3)}{max(V(R',A),V(R_3,A))} = \frac{2500 \times 1000}{50} = 50000$
- **因此对结果基数的估计为50000**

连接运算的基数估计示例

- **对$R_1 \bowtie (R_2 \bowtie R_3)$的连接顺序进行估计**
  - $T(R_2 \bowtie R_3) = \frac{T(R_2) \times T(R_3)}{\max(V(R_2,A),V(R_3,A))} = \frac{500 \times 1000}{50} = 10000$
  - 用R''表示$R_2 \bowtie R_3$ ， $V(R'',A) = min\big(V(R_2,A),V(R_3,A)\big) = 20$
  - $T(R_1 \bowtie R'') = \frac{T(R_1) \times T(R'')}{\max(V(R_1,A),V(R'',A))} = \frac{100 \times 10000}{20} = 50000$
- **因此对结果基数的估计同样为50000**

连接运算的基数估计

➢ 情况2: 两个关系在多个属性上连接

- 考虑两个关系$R_1(A,B)$和$R_2(A,C)$。
- 属性A为这两个关系进行自然连接的属性集合，B和C分别是关系R_1和R_2的非连接属性。
- 要进行结果基数估计的自然连接是$R_1(A,B) \bowtie R_2(A,C)$。
- 基数估计结果为$T\big(R_1(A,B) \bowtie R_2(A,C)\big) = \frac{T(R_1)\times T(R_2)}{\prod_{a\in A}\max(V(R_1,a),V(R_2,a))}$

连接运算的基数估计示例（2）

- 两个关系R_1, R_2在属性A, B上进行自然连接，估计连接结果基数

关系	$T(R_i)$	$V(R_i, A)$	$V(R_i, B)$
R_1	100	10	5
R_2	500	20	10

- $T(R_1 \bowtie R_2) = \frac{T(R_1) \times T(R_2)}{\max(V(R_1,A),V(R_2,A)) \times \max(V(R_1,A),V(R_2,A))} = \frac{100 \times 500}{20 \times 10} = 250$

连接运算的基数估计

➢ 情况3: 多个关系在多个属性上连接

- 考虑m个关系$R_1, R_2, \cdots, R_m$，其自然连接相当于在$R_1, R_2, \cdots, R_m$的笛卡尔积上加入重名属性取值相等的限制。
- 对于任意一个自然连接属性A，假设其涉及的关系集合为Q，则笛卡尔积结果中的任意一个元组符合该连接限制的概率为

$$P(A) = \begin{cases} \dfrac{1}{V(R,A)}, \text{其中} R \in Q, & \text{若}|Q| = 1 \\ \dfrac{\min_{R \in Q}\{V(R,A)\}}{\prod_{R \in Q} V(R,A)}, & \text{若}|Q| \neq 1 \end{cases}$$

- 对结果的基数估计$T(R_1 \bowtie R_2 \cdots \bowtie R_m) = \prod_{1 \le i \le m} T(R_i) \prod_{A \in S} P(A)$
- 其中S表示涉及的连接属性集合

连接运算的基数估计示例（3）

- 三个关系R_1, R_2, R_3在属性A, B上进行自然连接，$S = \{A, B\}$，表格中用横线表示某个关系不存在该属性，估计连接结果基数

关系	$T(R_i)$	$V(R_i, A)$	$V(R_i, B)$
R_1	100	10	2
R_2	500	-	4
R_3	1000	50	5

- 对于属性A的限制，$P(A) = \frac{\min_{R\in\{R_1,R_3\}}\{V(R,A)\}}{\prod_{R\in\{R_1,R_3\}} V(R,A)} = \frac{10}{10\times 50} = \frac{1}{50}$
- 对于属性B的限制，$P(B) = \frac{\min_{R\in\{R_1,R_2,R_3\}}\{V(R,B)\}}{\prod_{R\in\{R_1,R_2,R_3\}} V(R,B)} = \frac{2}{2\times 4\times 5} = \frac{1}{20}$
- $T(R_1 \bowtie R_2 \bowtie R_3) = \prod_{1\le i\le 3} T(R_i) \prod_{A\in S} P(A) = 100\times 500\times 1000\times \frac{1}{50}\times \frac{1}{20} = 50000$

其他算子的基数估计

- **投影运算**
  - 对于投影$\Pi_A(R)$，由于投影过程中会去除重复元组，因此可以估计为：

$$T\left(\Pi_A(R)\right) = V(\Pi_A(R), A) = V(R, A)$$

- **并运算**
  - 对于多集的并，由于多集可以包含重复元组，故$T(R_1 \cup R_2) = T(R_1) + T(R_2)$。
  - 对于集合的并，一般以$\frac{max(T(R_1),T(R_2))+T(R_1)+T(R_2)}{2}$作为估计。
- **交运算**
  - 对于两关系的交，一般以$\frac{\min(T(R_1),T(R_2))}{2}$作为估计

其他算子的基数估计

- **差运算**
  - 对于两关系的差，一般以$\frac{2\cdot T(R_1)-T(R_2)}{2}$作为估计
- **去重运算**
  - 对于属性集合为S的关系R，一般取$\min(\frac{T(R)}{2}, \prod_{A\in S} V(R,A))$作为估计
- **分组聚集运算**
  - 对于关系R，以及对其进行的分组操作 ${}_G\mathcal{G}_C(R)$，其中G是分组属性集合，一般以$\min(\frac{T(R)}{2}, \prod_{A\in G} V(R,A))$作为结果的大小估计

基于数据画像的基数估计

- **数据画像（data sketch）：通过建立能够较为准确近似数据信息的概率性的数据结构来替代直方图，在一些应用场景中可以进一步提高基数估计的准确度**
- **估计数据出现的频率**
  - Count-Min算法
- **估计数据中不同元素个数（Distinct）**
  - HyperLogLog算法

Count-Min算法

- **考虑一个简单情况**
- **给定一列数据，建立一个哈希函数H，以及一个w项的数据画像**
  - M[i]被初始化为0，0≤i≤w-1
  - 对该列数据的每一个取值v
  - M[H[v]%w]++
- **给定值x，用M[H[x]%w]来估计基数**

由于哈希冲突，常常**高估基数**

Count-Min算法

- 考虑使用d个哈希函数来减少哈希冲突的出现
- 建立有d行w列的矩阵M，初始化为0；对应于d个哈希函数
- 对每个值 v，以及每个哈希函数 h_i：
  - $M[i][h_i(v)] = M[i][h_i(v)] + 1;\ h_i(v)$ in [0,w)
- 给定值x，x出现的频率f(x)可以被估计为：
  - $f(x) = \min_{i \text{ in } [0,d-1]} M[i][h_i(v)]$

	0	1	2	3	4	…	w-1
$h_0(v)$	1	0	0	2	0	0	1
$h_1(v)$	1	0	0	1	0	0	0
…	0	0	0	0	2	0	0
$H_{d-1}(v)$	3	0	0	1	0	0	1

Count-Min算法

- **d=4 个哈希函数， w=7 列**
- **给定值{2,3,2,4,3,2,5}:**
  - $h_0(v)=v\%w$; $h_1(v)=v^2\%w$; $h_2(v)=(2v+1)\%w$; $h_3(v)=(3v^2+1)\%w$;
- **加入 2**

	0	1	2	3	4	5	6
$h_0(v)$	0	0	1	0	0	0	0
$h_1(v)$	0	0	0	0	1	0	0
$h_2(v)$	0	0	0	0	0	1	0
$h_3(v)$	0	0	0	0	0	0	1

Count-Min算法

- d=4 个哈希函数， w=7 列
- 给定值{2,3,2,4,3,2,5}:
  - $h_0(v)=v\%w$; $h_1(v)=v^2\%w$; $h_2(v)=(2v+1)\%w$; $h_3(v)=(3v^2+1)\%w$;
- 加入 2，3

	0	1	2	3	4	5	6
$h_0(v)$	0	0	1	1	0	0	0
$h_1(v)$	0	0	1	0	1	0	0
$h_2(v)$	1	0	0	0	0	1	0
$h_3(v)$	1	0	0	0	0	0	1

Count-Min算法

- d=4 个哈希函数， w=7 列
- 给定值{2,3,2,4,3,2,5}:
  - $h_0(v)=v\%w$; $h_1(v)=v^2\%w$; $h_2(v)=(2v+1)\%w$; $h_3(v)=(3v^2+1)\%w$;
- 加入 2，3

	0	1	2	3	4	5	6
$h_0(v)$	0	0	1	1	0	0	0
$h_1(v)$	0	0	1	0	1	0	0
$h_2(v)$	1	0	0	0	0	1	0
$h_3(v)$	1	0	0	0	0	0	1

Count-Min算法

- **d=4 个哈希函数， w=7 列**
- **给定值{2,3,2,4,3,2,5}:**
  - $h_0(v)=v\%w$; $h_1(v)=v^2\%w$; $h_2(v)=(2v+1)\%w$; $h_3(v)=(3v^2+1)\%w$;
- **加入 2，3，2**

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	0	0	0
$h_1(v)$	0	0	1	0	2	0	0
$h_2(v)$	1	0	0	0	0	2	0
$h_3(v)$	1	0	0	0	0	0	2

Count-Min算法

- d=4 个哈希函数， w=7 列
- 给定值{2,3,2,4,3,2,5}:
  - $h_0(v)=v\%w$; $h_1(v)=v^2\%w$; $h_2(v)=(2v+1)\%w$; $h_3(v)=(3v^2+1)\%w$;
- 加入 2，3，2，4

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	1	0	0
$h_1(v)$	0	0	1+1	0	2	0	0
$h_2(v)$	1	0	1	0	0	2	0
$h_3(v)$	1+1	0	0	0	0	0	2

Count-Min算法

- d=4 个哈希函数， w=7 列
- 给定值{2,3,2,4,3,2,5}:
  - $h_0(v)=v\%w$; $h_1(v)=v^2\%w$; $h_2(v)=(2v+1)\%w$; $h_3(v)=(3v^2+1)\%w$;
- 加入 2，3，2，4，5

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	1	1	0
$h_1(v)$	0	0	1+1	0	2+1	0	0
$h_2(v)$	1	0	1	1	0	2	0
$h_3(v)$	1+1	0	0	0	0	0	2+1

count(2)=2; count(3)=1; count(4)=1; count(5)=1

HyperLogLog算法

- **估计某列不同取值个数**
  - 线性扫描进行统计：过于**低效**
  - 哈希：**空间占用过大**

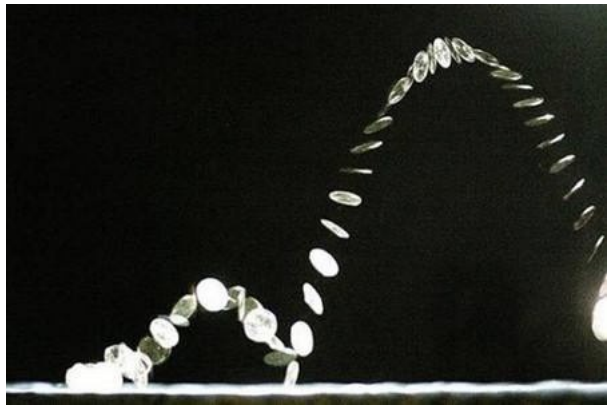
    …

- **HyperLogLog**
  - 只需要**12KB**的内存，就可以计算最高达2^{64}个不同元素的基数
  - 广泛应用在内存数据库Redis、大数据运算框架Spark、Flink中

HyperLogLog算法

- **核心思想：假设我们有一个完美的哈希函数h，以[1,r]的整数位输入，输出一个[0,n]之间整数形式的哈希值**
  - “完美”指：[1,r]的输入对应的哈希值为[0,n]的均匀分布
- **二进制形式下，哈希值后缀零个数的期望概率为：**
  - 1个连续后缀0的概率：1/2
  - 2个连续后缀0的概率：1/4
  - 3个连续后缀0的概率：1/8
  - 4个连续后缀0的概率：1/16

    …
  - w<log n 个连续后缀0的概率 $1/2^w$



HyperLogLog算法

- 用$f(x)$表示x在二进制下后缀中第一个1出现的位置
- 对于经过哈希后均匀分布的随机数集S，考虑S中所有数字的二进制表示，定义$k = max_{a \in S}(f(a))$
- 对于$max_{a \in S}(f(a)) = k$的数组S，其基数的期望为2^k
  - 所有数据放在一起，估计值误差大

数据	哈希	二进制
a	9	1001
a	9	1001
a	9	1001
b	4	0100
b	4	0100
c	13	1101

$k = 3$

估计基数

$2^k = 2^3 = 8$

□ 实际基数：3

□ 估计基数：8

计算**全局的k，误差很大**！

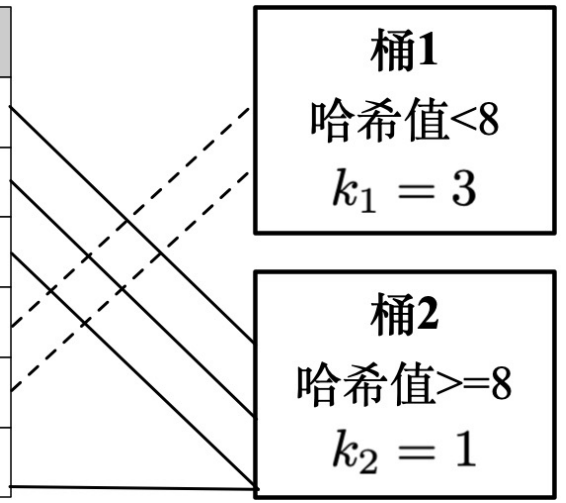
HyperLogLog算法

- **改进：将数据按照哈希值放入m个（默认16384个）桶中**
  - 数据划分为S_1、S_2、S_3、…、S_m
  - 每个桶分别计算桶内数据的k值，得到k_1、k_2、k_3、…、k_m
  - 计算这些k的调和平均值$k' = \frac{m}{\frac{1}{k_1}+\frac{1}{k_2}+\cdots+\frac{1}{k_m}}$
  - 将k'作为估计基数时采用的更加准确的参考数据
  - 使用$2^{k'}$作为对基数的估计

HyperLogLog算法示例（2）

- 实际基数：3
- 估计基数：2.83≈ 3

数据	哈希	二进制
a	9	1001
a	9	1001
a	9	1001
b	4	0100
b	4	0100
c	13	1101



按照第一位分桶

$$k' = \frac{2}{\frac{1}{3} + \frac{1}{1}} = \frac{3}{2}$$

估计基数

$$2^{k'} = 2^{\frac{3}{2}} \approx 2.83 \approx 3$$

分桶计算k，有效提高准确度

HyperLogLog复杂度分析

- **分桶个数m，最大基数n**
- **空间复杂度：$O(mlog(log(n))$**
  - 建立了m个桶
  - n个不同元素哈希后最长后缀零的长度为$O(\log(n)$
  - 二进制存储，$O(\log(n)$种不同的后缀零长度，只需要$O(\log(\log(n))$个位（bit）就能存储
- **时间复杂度：$O(m)$**
  - 查询时只需要整合m个桶的信息
  - 桶的个数m常被视为一个固定的常数，因此也可以认为时间复杂度为$O(1)$的

基于采样的基数估计

- 使用所有数据来计算统计信息的代价较高，而且受到统计信息不断动态改变的影响
- 数据库一般会选择在负载较低或发生了大量修改操作的时候才更新统计信息，以减少对正常业务的影响
- 然而在关系表很大时，统计信息的更新仍然会给正常业务带来显著影响
- 数据库在进行统计信息的计算和维护前，一般会使用采样方法维护一个样本池来代表其存储的全量元组
- 优化器针对这个样本池来收集和维护统计信息，并在数据库发生更新时对应更新该样本池
- 收集统计信息时机：业务闲、计划不准

基于采样的基数估计

- **蓄水池采样是一种随机采样算法，在不知道集合大小n的情况下，随机抽取k个样本。**

- **采样过程只需遍历一次集合中的所有元素，并且保证每个元素的被选取概率相同。采样的原理如下：**
  - 维护一个大小为k的蓄水池，集合中前k个元素用于初始化蓄水池
  - 对于后续的第i(i>k)个元素，都以$\frac{k}{i}$的概率替换蓄水池中的元素，且蓄水池中每个元素被替换的概率均为$\frac{1}{k}$

蓄水池采样证明

- **对于第i项（i<=k）**
  - 根据算法定义，前**k**步中被选中的概率为1
  - 第**k+1**步中，被替换的概率为：
    - 选中第k+1项的概率 * 选中i来替换的概率，即为 $\frac{k}{\mathrm{k+1}} * \frac{1}{\mathrm{k}} = \frac{1}{k+1}$
  - 因此，第**k+1**步中，没有被替换的概率为：$1 - \frac{1}{k+1} = \frac{k}{k+1}$
  - 同理可得，在第**k+2**步中，没有被替换的概率为：$\frac{k}{k+2}$
  - 前**n**步没有被替换的概率：

= 前**k**步没有被替换的概率 * 第**k+1**步没有被替换的概率 * 第**k+2**步...

$$= 1 * \frac{k}{k+1} * \frac{k+1}{k+2} * \cdots * \frac{n-1}{n}$$

$$= \frac{k}{n}$$

蓄水池采样证明

- **对于第i项（i>k)**
  - 根据算法定义，在第**i**步中被选中的概率为$\frac{k}{i}$
  - 按照先前的分析，在第**j**步（j>i）中没有被替换的概率为：$\frac{j-1}{j}$

所以，第i项在前**n**步中没有被替换的概率

= 在第**i**步中被选中的概率 * 在第**i+1**步没有被替换的概率 *在第**i+2**步……

$$= \frac{k}{i} * \frac{i}{i+1} * \cdots * \frac{n-1}{n}$$

$$= \frac{k}{n}$$

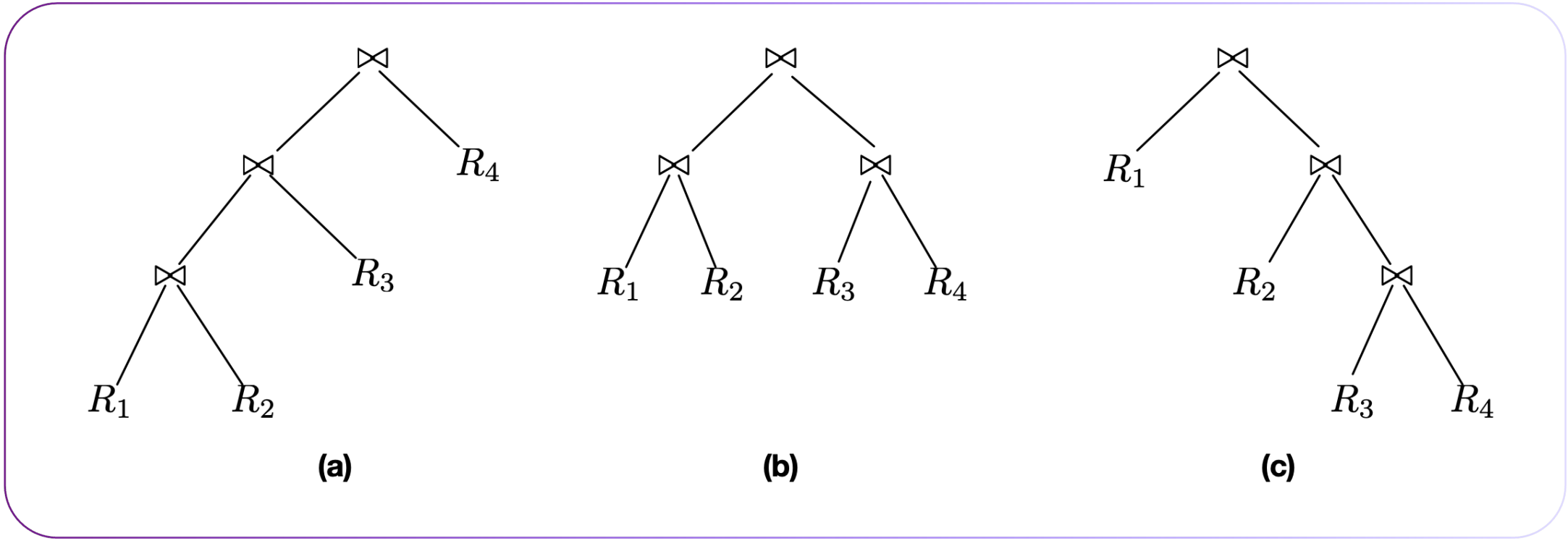
任意一项被采样的概率均为$\frac{k}{n}$，是等概率随机采样

目录

连接树

- **连接树（join tree）：表示n个关系连接时采用的连接顺序**
  - 二叉树
  - 内部节点都是连接运算
  - n个叶子节点与n个关系一一对应
- **连接树分类：**
  - **左深连接树**：内部节点的右子节点都是叶子节点的连接树
  - **右深连接树**：内部节点的左子节点都是叶子节点的连接树
  - **浓密树**：不是左深树和右深树的连接树

连接树示例



左深连接树

$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$

浓密树

$((R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4))$

右深连接树

$R_1 \bowtie (R_2 \bowtie (R_3 \bowtie R_4))$

连接树数量规模分析

- **n个关系构成的结构不同的连接树个数为：**
  - $C_n = \frac{1}{n}\binom{2n-2}{n-1} = \frac{(2n-2)!}{n!(n-1)!}$
  - 恰好为著名的**卡特兰数**（Catalan number）

- **对于每个连接树结构，n个关系的n!个全排列中每个都对应一个连接树**
  - 不同连接树个数为：$\mathrm{NTree}(n) = n!\,C_n = \frac{(2n-2)!}{(n-1)!}$
  - 规模迅速增加，$\mathrm{NTree}(10) \approx 10^{10}$

- **只考虑部分连接树——左深连接树 n!**

左深连接树

- **相较于所有连接树，左深树数量很小 n!：**
  - 对于n个关系连接的情况，只有1个确定的左深树结构
  - 例如，当$n = 10$时，左深树只占所有连接树数量的$\frac{1}{4862}$
  - 左深3628800 vs 稠密17643225600

- **左深连接树每个右子节点均为基本关系，利于连接运算高效执行**
  - 例如，嵌套循环连接可以利用基本关系已经建立的索引，执行更加高效
  - 避免将连接中一些过大的中间结果物化到磁盘上

连接顺序选择算法

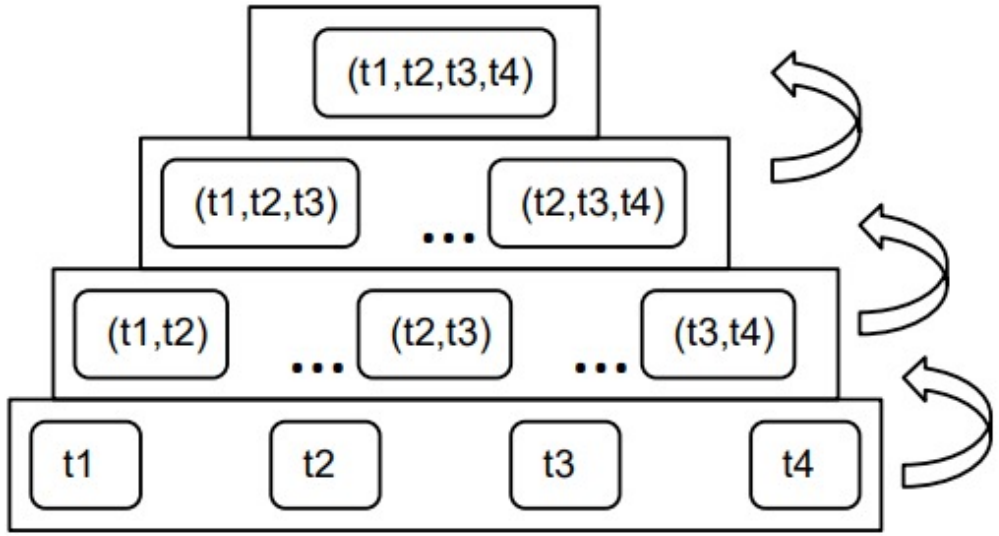
① **动态规划算法**

② 贪心算法

③ 遗传算法

动态规划算法

- **尽管只考虑左深树，枚举过程仍然花销很大：**
  - 枚举过程中存在大量的**重复子结构**
  - 考虑枚举R_1, R_2, R_3, R_4的连接顺序
    - $((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$
    - $((R_1 \bowtie R_2) \bowtie R_4) \bowtie R_3$
    - $R_1 \bowtie R_2$的估计结果可以复用
- **动态规划算法**
  - **核心思想：**将计算过的结果记忆化存储
  - 当遇到重复的相同子问题时，可以**直接利用计算过的结果**，无需重新计算。
  - 由此实现**减少计算量**



动态规划算法

- **当枚举到集合S时，按如下方式计算集合S的信息（稠密树）：**
  - $C(S)$：连接的最小代价
    - $|S| > 1$:
      - 枚举集合S的子集S_l，计算$S_r = S - S_l$，$1 \le |S_l| \le |S| - 1$
      - 状态转移方程：
        $C(S) = \min_{S_l \subset S \,\&\, S_l \neq \emptyset}(C(S_l) + C(S_r) + Cost(S_l, S_r))$
    - $|S| = 1$:
      - $C(S) = T(S)$
  - $Cost(S_l, S_r) \approx T(S)$
    - 按照连接运算的基数估计方法进行代价估算
  - $\mathrm{Seq}(S)$：连接的顺序
    - 确定了使$C(S)$最小的S_l, S_r后可以得到最优连接顺序
    - $\mathrm{Seq}(S) = \mathrm{Seq}(S_l) \bowtie \mathrm{Seq}(S_r)$

左深树：枚举每个表A
$C(S) = \min(C(A) + C(S - A) + Cost(S - A, A))$

动态规划算法示例

- **考虑为四个关系R_1, R_2, R_3, R_4进行连接顺序选择：**
  - 每个关系的大小均为1000

$$R_1 \overset{A}{\text{——}} R_2 \overset{B}{\text{——}} R_3 \overset{C}{\text{——}} R_4$$

集合大小为1的动态规划信息表

关系集合S	$T(S)$	$V(S,A)$	$V(S,B)$	$V(S,C)$	$C(S)$	$\text{Seq}(S)$
$\{R_1\}$	1000	50	-	-	1000	R_1
$\{R_2\}$	1000	40	100	-	1000	R_2
$\{R_3\}$	1000	-	20	200	1000	R_3
$\{R_4\}$	1000	-	-	250	1000	R_4

动态规划算法示例

- 考虑两个关系连接后的情况：

$$R_1 \overset{A}{—} R_2 \overset{B}{—} R_3 \overset{C}{—} R_4$$

集合大小为2的动态规划信息表

关系集合S	$T(S)$	$V(S,A)$	$V(S,B)$	$V(S,C)$	$C(S)$	$\mathrm{Seq}(S)$
$\{R_1,R_2\}$	20000	40	100	-	22000	$R_1 \bowtie R_2$
$\{R_2,R_3\}$	10000	40	20	200	12000	$R_2 \bowtie R_3$
$\{R_3,R_4\}$	4000	-	20	200	6000	$R_3 \bowtie R_4$

动态规划算法示例

- 考虑三个关系连接后的情况：

$$R_1 \overset{A}{—} R_2 \overset{B}{—} R_3 \overset{C}{—} R_4$$

集合大小为3的动态规划信息表

关系集合S	$T(S)$	$V(S,A)$	$V(S,B)$	$V(S,C)$	$C(S)$	$\mathrm{Seq}(S)$
$\{R_1,R_2,R_3\}$	200000	40	20	200	213000	$(R_2 \bowtie R_3) \bowtie R_1$
$\{R_2,R_3,R_4\}$	40000	40	20	200	47000	$(R_3 \bowtie R_4) \bowtie R_2$

动态规划算法示例

- 考虑所有四个关系时，$S = \{R_1, R_2, R_3, R_4\}$
- 需要在以下选项中选择
  - $\text{Seq}(\{R_1, R_2, R_3\}) \bowtie \text{Seq}(\{R_4\})$,
  - $\text{Seq}(\{R_2, R_3, R_4\}) \bowtie \text{Seq}(\{R_1\})$,
  - $\text{Seq}(\{R_1, R_2\}) \bowtie \text{Seq}(\{R_3, R_4\})$
- 对于$\text{Seq}(\{R_1, R_2, R_3\}) \bowtie Seq(\{R_4\})$，总代价：

$$C(\{R_1, R_2, R_3\}) + C(\{R_4\}) + Cost(\{R_1, R_2, R_3\}, \{R_4\}) = 1014000$$

动态规划算法示例

- **对于$Seq(\{R_2, R_3, R_4\}) \bowtie Seq(\{R_1\})$，总代价**

$$\mathbf{C}(\{R_2, R_3, R_4\}) + \mathbf{C}(\{R_1\}) + Cost(\{R_2, R_3, R_4\}, \{\mathbf{R_1}\}) = \mathbf{848000}$$

- **对于$Seq(\{R_1, R_2\}) \bowtie Seq(\{R_3, R_4\})$，总代价**

$$C(\{R_1, R_2\}) + C(\{R_3, R_4\}) + Cost(\{R_1, R_2\}, \{R_3, R_4\}) = \mathbf{828000}$$

- **最终确定使用代价最低的连接顺序：**
  - $Seq(\{R_1, R_2\}) \bowtie Seq(\{R_3, R_4\}) = (R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$

动态规划算法复杂度

- **空间复杂度（左深树）**
  - 由于非空集合数量是$2^n - 1$，而维护的信息大小是常数
  - 总的空间复杂度 $O(2^n)$
- **时间复杂度（左深树）**

$$O\left(\sum_{i=1}^{n} C_n^i * i\right) = O\left(\sum_{i=1}^{n} \frac{n!}{i!\,(n-i)!} * i\right) = O\left(n \sum_{i=1}^{n} \frac{(n-1)!}{(i-1)!\,((n-1)-(i-1))!}\right) = O\left(n * \sum_{i=1}^{n} C_{n-1}^{i-1}\right)$$

$$= O(n * 2^{n-1}) = O(n * 2^n)$$

- **稠密树时间复杂度：**

$$O\left(\sum_{i=1}^{n} C_n^i * (2^i - 2)\right) = O\left(\sum_{i=1}^{n} C_n^i * 2^i\right) = O(3^n)$$

连接顺序选择算法

① 动态规划算法

② 贪心算法

③ 遗传算法

贪心算法

- **核心思想：每次选择当前状态下增加的代价最小的关系连接**
- **算法执行过程：**
  - （1）选择一个大小$T(R)$最小的关系作为当前关系R_{cur}；
  - （2）从余下没有被选择过的关系中选择与R_{cur}连接后结果最小的R_{next}，将R_{cur}更新为$R_{\text{cur}} \bowtie R_{\text{next}}$；
  - （3）不断重复步骤（2），直到所有关系都被选择过。
- **时间复杂度：**
  - 单次复杂度为$O(n)$的决策（计算与R_{cur}连接后的大小）
  - 重复n轮，因此使用贪心算法只有$O(n^2)$的复杂度

贪心算法示例

关系集合S	$T(S)$	$V(S,A)$	$V(S,B)$	$V(S,C)$
$\{R_1\}$	1000	50	-	-
$\{R_2\}$	1000	40	100	-
$\{R_3\}$	1000	-	20	200
$\{R_4\}$	1000	-	-	250

$$R_1 \overset{A}{\text{——}} R_2 \overset{B}{\text{——}} R_3 \overset{C}{\text{——}} R_4$$

- **第一步：所有关系大小都相等，但R_2和R_3在进行连接时有2种选择，选择R_2作为当前关系R_{cur}。**

贪心算法示例

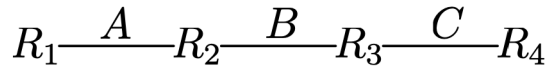
关系集合S	$T(S)$	$V(S,A)$	$V(S,B)$	$V(S,C)$
$\{R_1,R_2\}$	20000	40	100	-
$\{R_2,R_3\}$	10000	40	20	200
$\{R_3,R_4\}$	4000	-	20	200

$$R_1 \overset{A}{—} R_2 \overset{B}{—} R_3 \overset{C}{—} R_4$$

- **第二步：比较$T(R_1 \bowtie R_2)$和$T(R_2 \bowtie R_3)$，由于$T(R_2 \bowtie R_3)$更小，选择R_3进行连接，更新R_{cur}为$R_2 \bowtie R_3$。**

贪心算法示例

关系集合S	$T(S)$	$V(S,A)$	$V(S,B)$	$V(S,C)$
$\{R_1,R_2,R_3\}$	200000	40	20	200
$\{R_2,R_3,R_4\}$	40000	40	20	200



- **第三步：比较$T((R_2 \bowtie R_3) \bowtie R_1)$和$T((R_2 \bowtie R_3) \bowtie R_4)$，由于后者更小，选择$R_4$进行下一步连接，更新$R_{cur}$为$(R_2 \bowtie R_3) \bowtie R_4$。**
- **第四步：仅剩R_1，因此最终确定了连接顺序为$((R_2 \bowtie R_3) \bowtie R_4) \bowtie R_1$。**
- **核心思想：每次选择当前状态下增加的代价最小的关系连接**
- **算法执行过程：**
  - （1）选择一个大小$N(R)$最小的关系作为当前关系R_{cur}；
  - （2）从余下没有被选择过的关系中选择与R_{cur}连接后结果最小的R_{next}，将R_{cur}更新为$R_{cur} \bowtie R_{next}$；
  - （3）不断重复步骤（2），直到所有关系都被选择过。
- **时间复杂度：**
  - 单次复杂度为$O(n)$的决策（计算与R_{cur}连接后的大小）
  - 重复n轮，因此使用贪心算法只有$O(n^2)$的复杂度

连接顺序选择算法

① 动态规划算法

② 贪心算法

③ 遗传算法

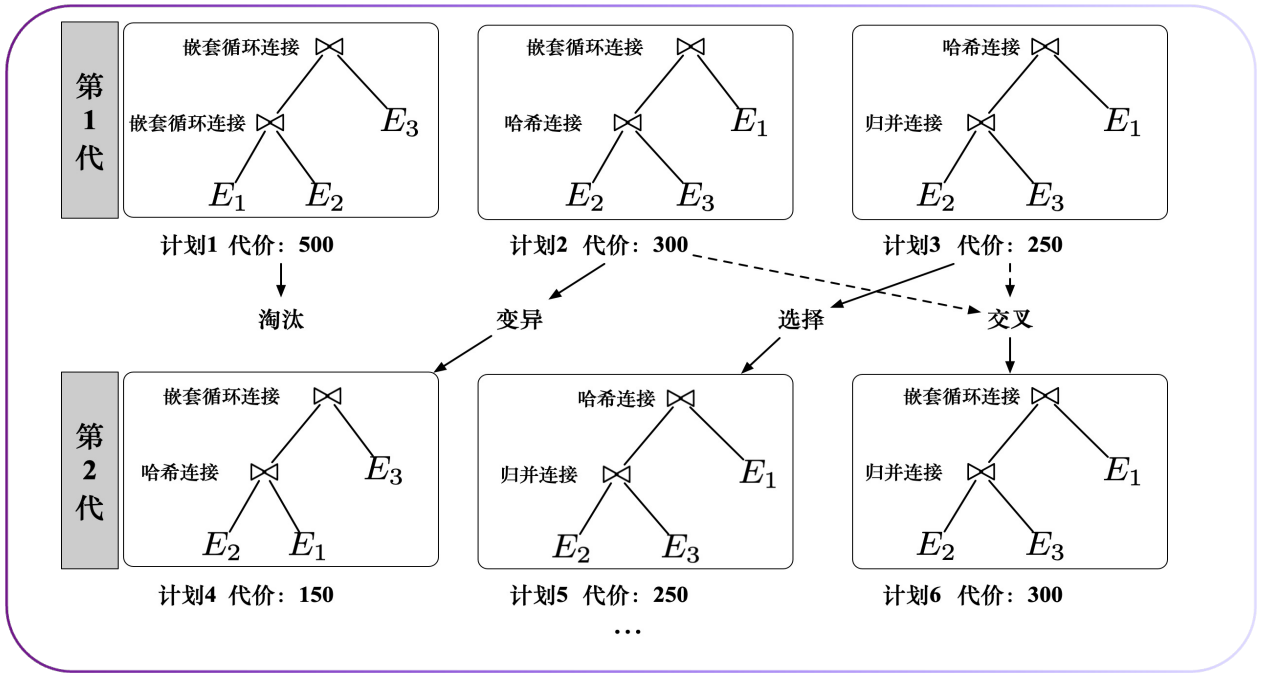
遗传算法

- 一种启发式的优化方法
- 核心思想：“自然选择”，淘汰“适应程度”低的“个体”
- 个体：不同的连接顺序
- 适应程度：连接顺序的代价
- 优点：不受关系数的限制。易于跳出局部最优。空间开销较小
- 缺点：可解释性较差。作为随机化算法，优化结果具有不确定性。

遗传算法

- 交叉：对连接顺序进行组合替换。也就是由已有的两个连接顺序合并为一个新的连接顺序，其中每个连接顺序只有一部分被保留在合并后的结果中。
- 变异：对某个顺序自身进行调整（例如替换物理算子），得到一个新的连接顺序。
- 选择：代价较低的顺序，直接遗传到下一代。

遗传算法示例



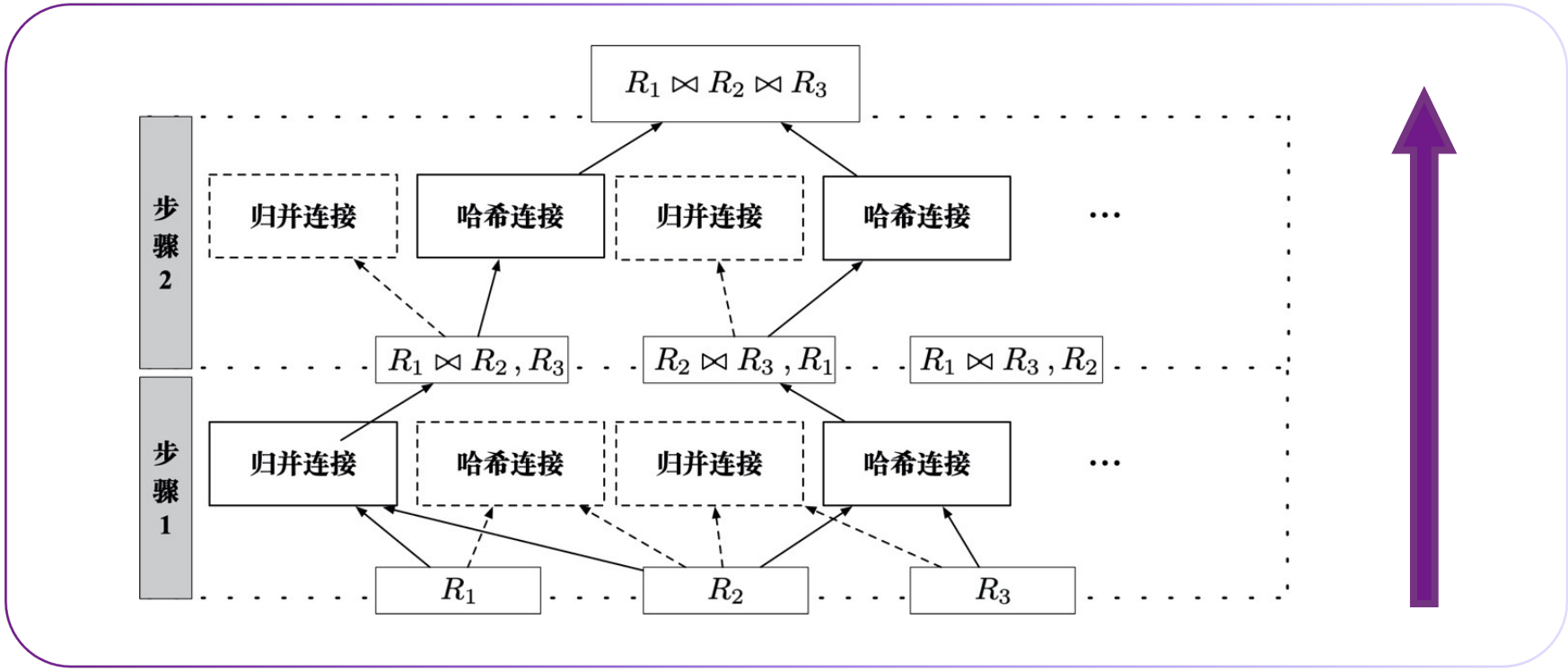
- 初始化：随机选择一批计划（例如1000）
- 第一轮：计划1代价最大，被淘汰
- 计划2、计划3 变异，选择与交叉，得到第二轮候选计划
- 不断迭代以上过程，结束条件：代价不再降低；达到轮数

目录

动态规划

- 添加每个算子的物理算子代价，动态规划中添加物理算子代价
- **最初从空的计划树开始，在第一步中先确定对哪两个关系进行连接**

三个关系$R_1(A,B), R_2(A,C), R_3(A,D)$，某查询要得到$R_1 \bowtie R_2 \bowtie R_3$的结果



动态规划：Selinger风格的优化

- **对于每个子表达式不只保留代价最小的计划，也会保留一些当前代价较高，但具有一些可能利于后续优化的性质的计划。**
- **有趣顺序（interesting order）：Selinger风格的优化器考虑的性质是查询计划的输出结果在某列的有序性。**
  - 查询可能存在对有序的要求
  - 有序性在查询树的更高层节点可以被利用到，则可能会大幅度降低后续运算的代价，使得查询计划的全局代价更小
- **原始动态规划过程没有考虑有趣顺序，需要单独记录符合有趣顺序的计划**
  - **动态规划算法中单独保留带有顺序方法的计划即可**

目录

优化器系统

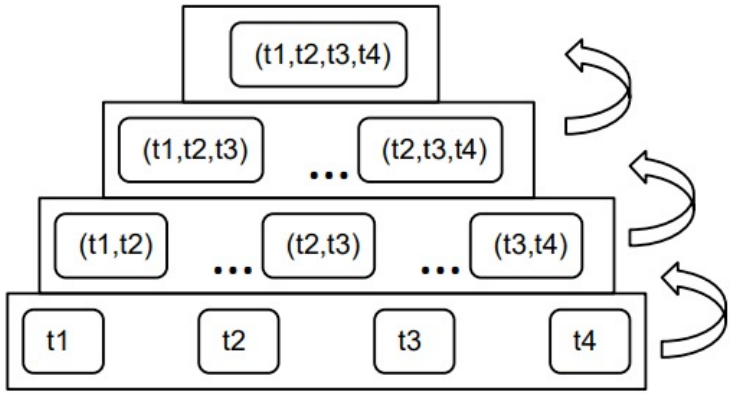
① **优化器系统的计划生成**

② 物化与流水线

自底向上 vs 自顶向下

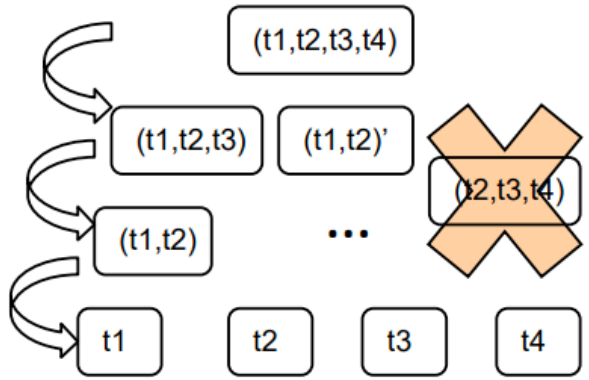
自底向上（例如R系统）

- 直观，易于实现
- 难以使用剪枝策略
- 无法提前停止



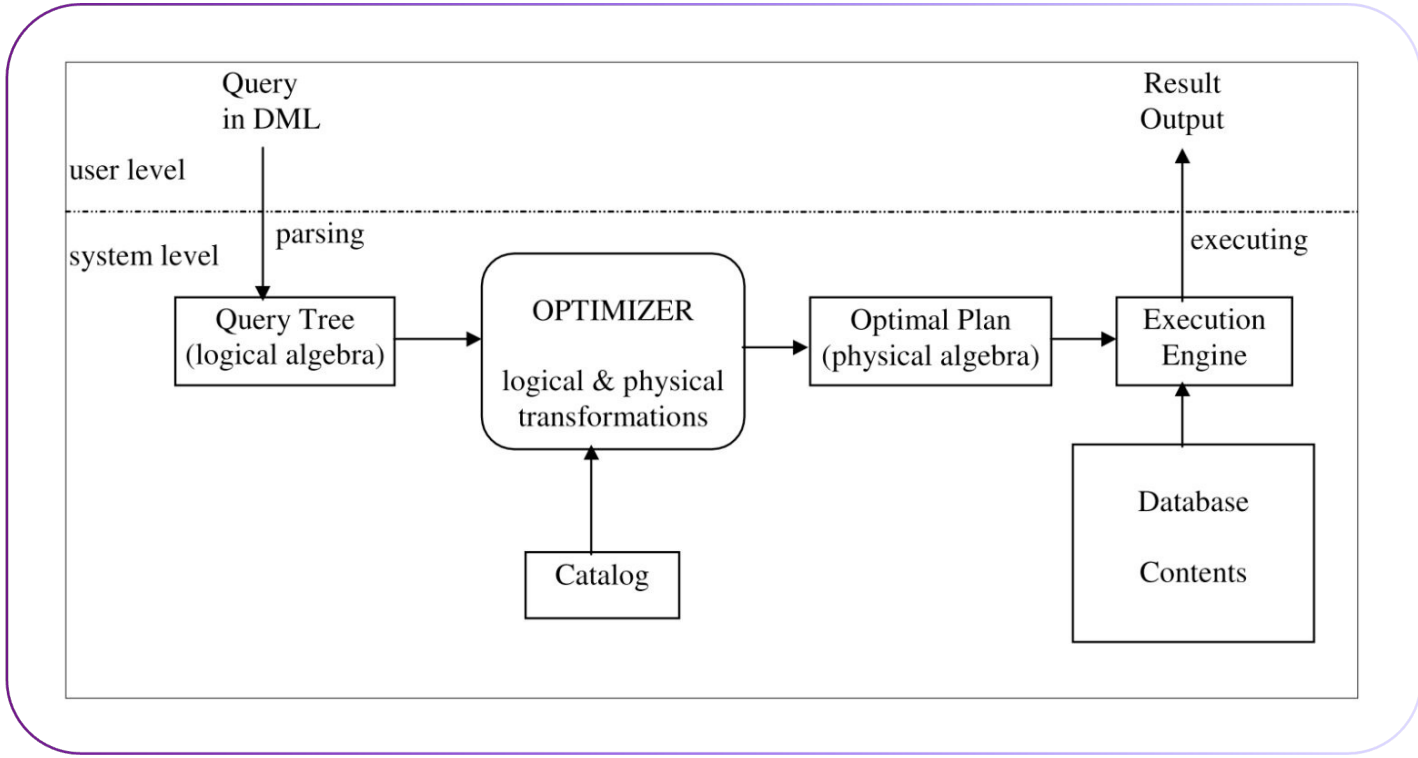
自顶向下（例如Cascades）

- 实现比较复杂
- 易于添加新规则
- 可以进行大量剪枝

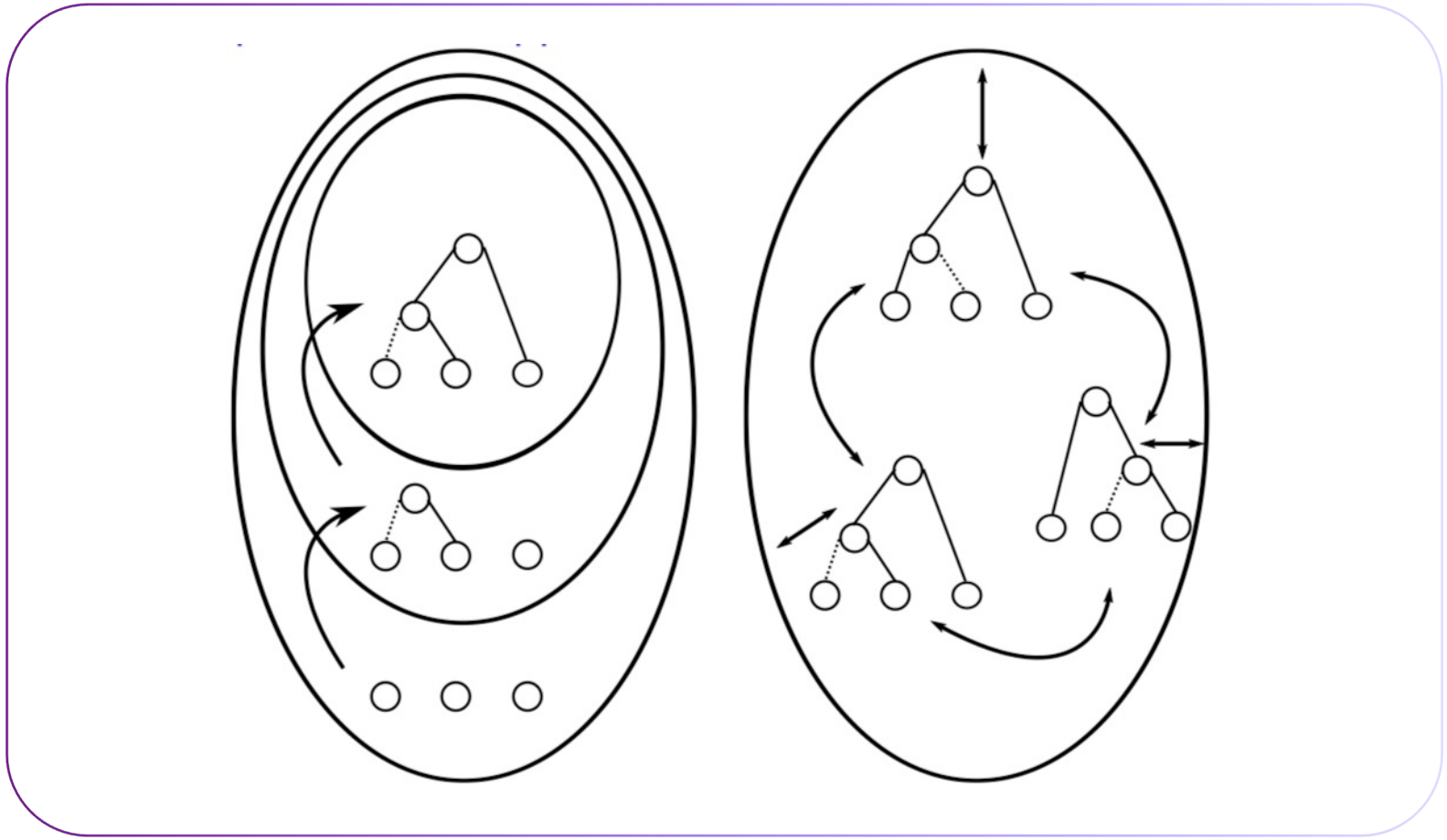


自底向上 vs 自顶向下

- 自底向上R系统：枚举计划并选择代价最低的计划
- 自顶向下Cascades：优化的过程中不断剪枝



自底向上 vs 自顶向下

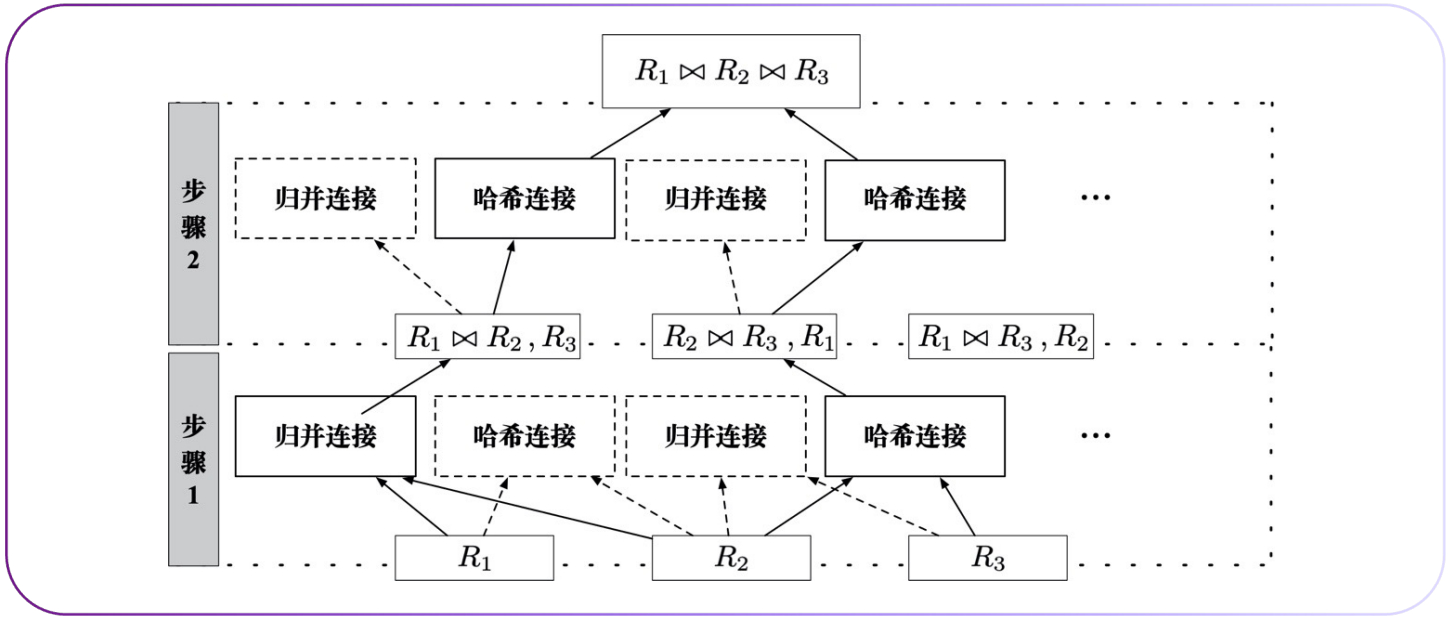


自底向上

- 动态规划是一种典型的“自底向上”的方法
- “自底向上”指：由空白的物理计划开始，从叶子节点出发，考虑每个子表达式，对可能的转换方法搜索，结合代价估计，每一步中选取代价最小的物理运算，最终确定整个计划树。
- 动态规划算法中，每个子表达式仅代价最小的计划会被保留
  - 然而物理计划转换问题并不满足最优子结构的性质
  - 原始的动态规划算法并不能保证全局的最优（interesting order ）

自底向上示例

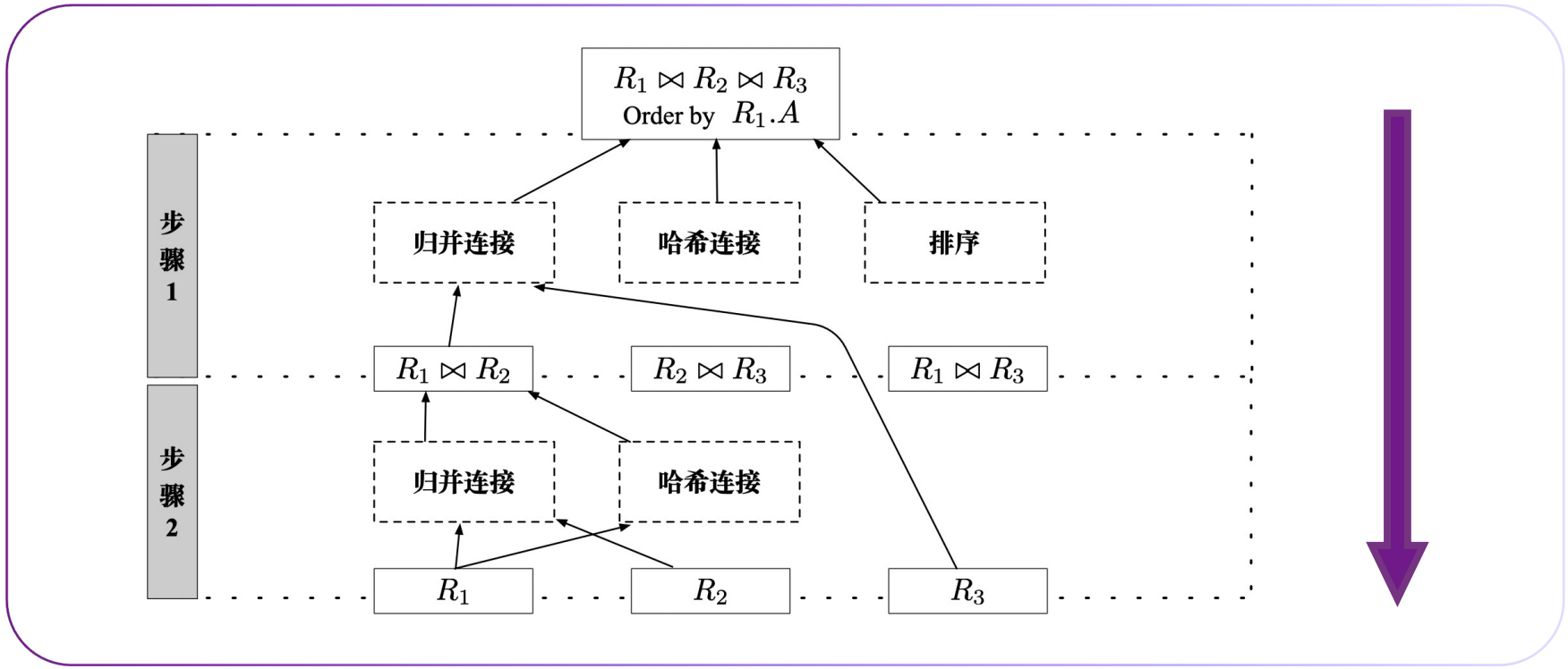
- 确定计划的过程中，通过比较估计代价的大小，对每个连接运算选择代价最小的物理算法。



- 优点：容易实现，不需要穷尽搜索就能找到较为合理的执行计划
- 缺点：仍然没有摆脱启发式转换，优化时间长，添加规则繁琐。

自顶向下

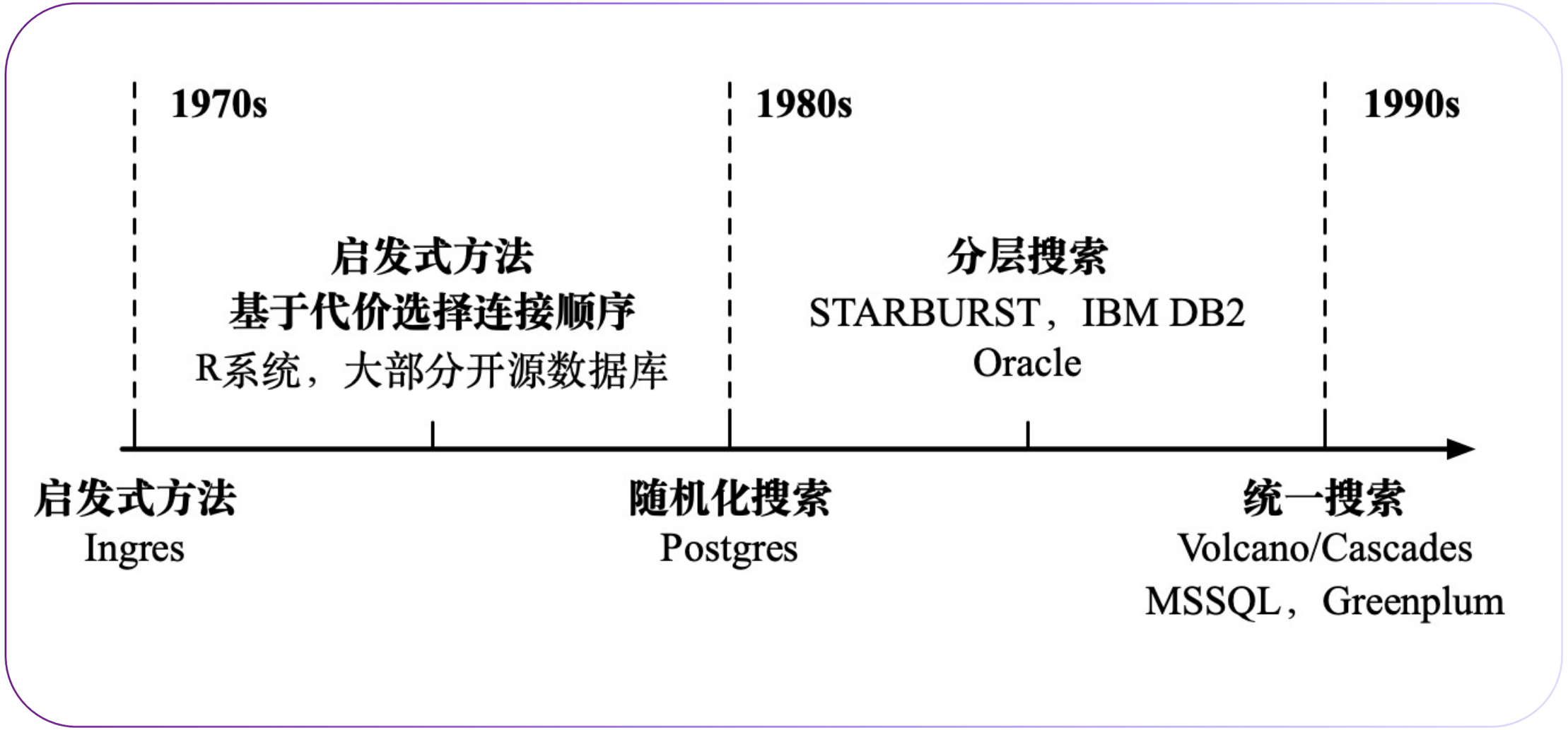
- 通用的基于关系代数等价转换和代价模型的查询优化器
  - 易于添加新的算子和等价转换规则
  - 重点关注数据的物理特性
  - 自顶向下，并利用分支界定剪枝



自顶向下

- 自顶向下：以查询的最终输出为起点，由计划树的根节点向下进行搜索，不断添加为得到最终输出需要的逻辑算子和物理算子，最终确定要采用的计划树。
- 整个过程中可以用分支界定（branch and bound）进行剪枝
- 优点：
  - 相较于两阶段的方法，统一搜索会产生更多的转换，优化效果可能更好
  - 可以提前中止
  - 添加规则简单
- 缺点：空间开销更大、优化慢、添加规则繁琐

优化器系统


1970s
1980s
1990s
启发式方法
基于代价选择连接顺序
R系统，大部分开源数据库
分层搜索
STARBURST，IBM DB2
Oracle
启发式方法
Ingres
随机化搜索
Postgres
统一搜索
Volcano/Cascades
MSSQL，Greenplum

① 启发式方法

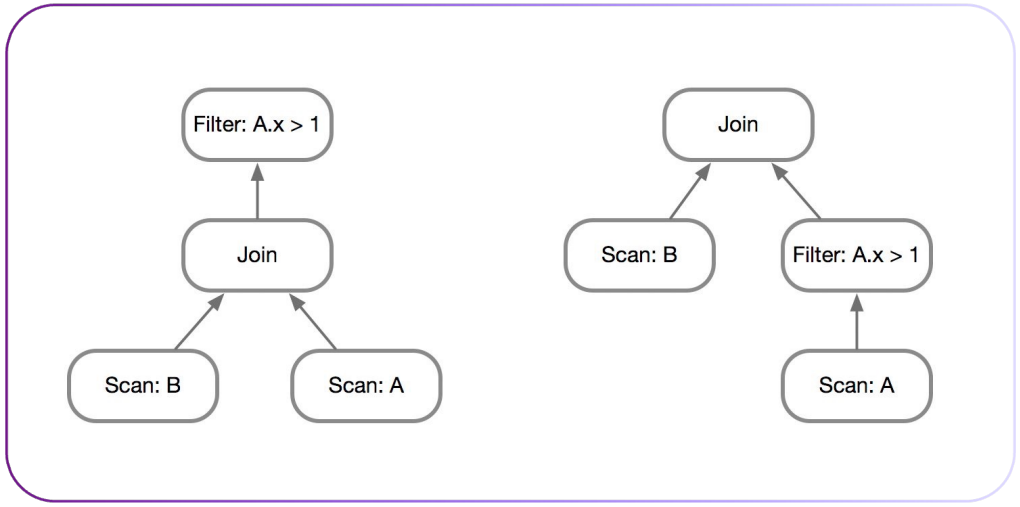
- 通过在优化器代码中静态定义规则来实现由逻辑计划到物理计划的转换
- 常用规则：
  - 优先执行选择度更高的选择运算
  - 在连接运算前执行所有的选择运算
  - 当选择条件为等值时，如果关系在该属性上有索引，则优先使用索引扫描
- 例子：INGRES, Oracle(1990s以前)
- 优点：
  - 易于实现与调试；优化速度快
- 缺点：
  - 完全依赖于预定义的规则；复杂查询优化效果不好，没有考虑算子之间的相互关系

② 启发式方法+代价选择连接顺序

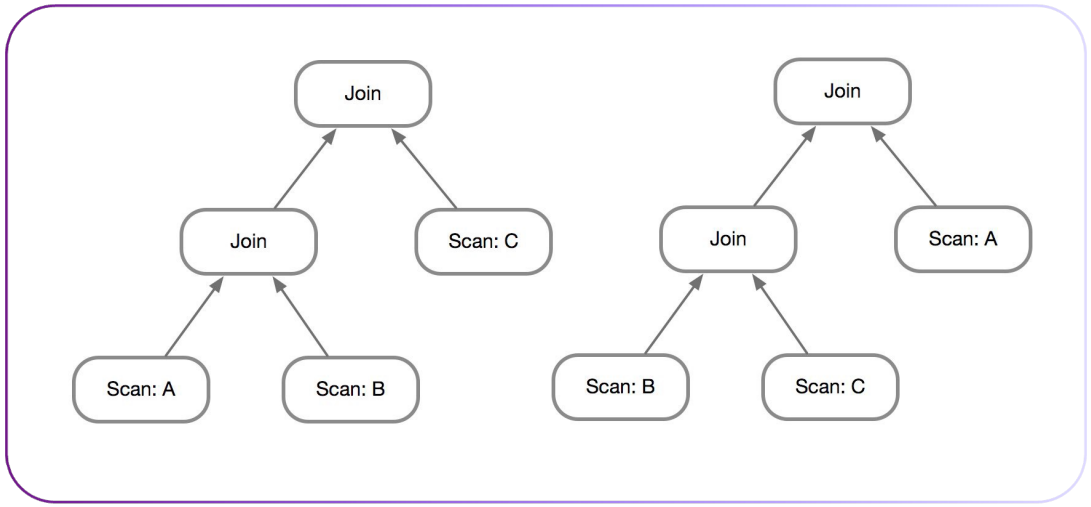
- 首先，利用静态规则进行初步优化
- 然后，利用动态规划算法选择多表连接时的连接顺序
  - 首个基于代价的查询优化器
  - 首次采用自底向上的搜索策略
- 例子：R系统，早期的IBM DB2，大多数开源数据库
- R系统的Selinger风格的优化器是现代优化器的设计基础
- 后来的Volcano，Cascades等方法均为在此基础上的改进

② 启发式方法+代价选择连接顺序

- 首个自底向上的优化器：R系统（IBM）
  - 逻辑：基于规则的等价转换；使用静态规则初始化逻辑算子
  - 物理：自底向上地基于代价模型寻找最优连接顺序；对于每个逻辑算子，枚举实现该运算符的物理算子，得到物理算子的所有组合
- 然后迭代地构建左深连接树，以最小化执行计划所需的估计代价。



基于规则

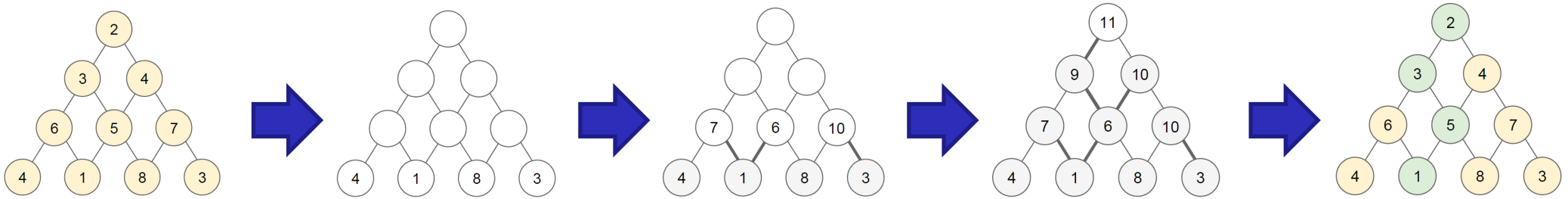


基于代价选择连接顺序

② 启发式方法+代价选择连接顺序

- 首个自底向上的优化器：R系统（IBM）
  - 逻辑优化和物理优化被严格区分
  - 与优化规则耦合紧密。加入新规则需要同时加入其与其他规则的关系。
  - 扩展能力差：
    - 难以添加新的规则
    - 自底向上的动态规划缺少全局优化视角

自底向上地基于代价模型寻找最优计划



② 启发式方法+代价选择连接顺序

➢ **优点：**

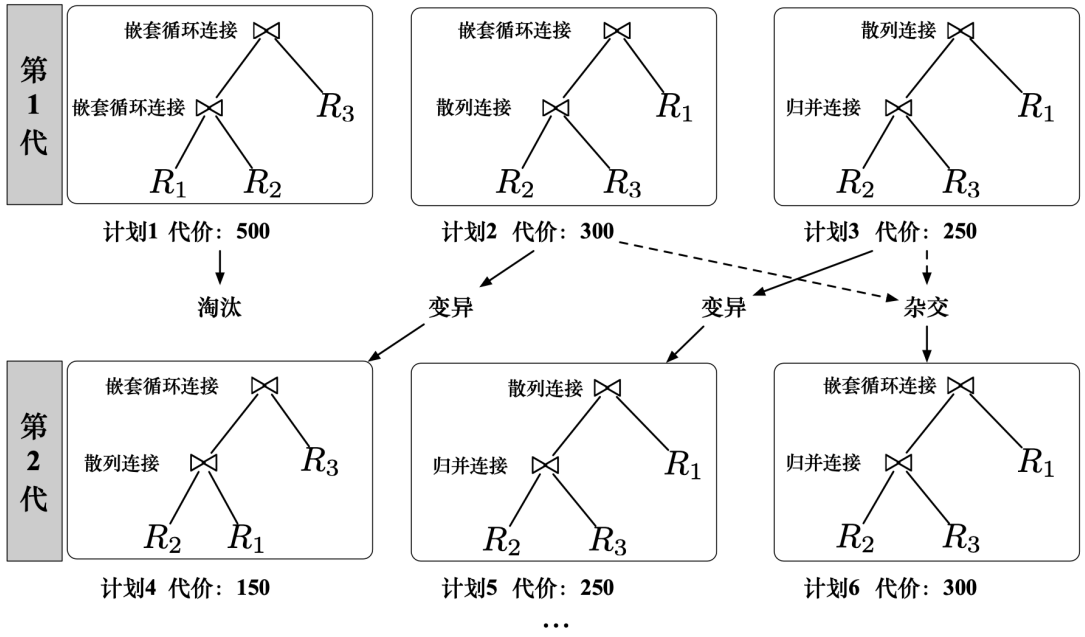
- 通常无需进行穷尽的搜索就可以找到一个比较好的计划
- 简单容易实现

➢ **缺点：**

- 添加规则繁琐
- 优化时间慢
- 左深连接树不总是最优
- 在代价模型中需要考虑到数据的物理属性（例如有序性）

③ 随机化搜索

- 20世纪80年代，学术界也提出了随机化的搜索策略，利用随机化策略来跳出搜索空间中的局部最优
- 例子： Postgres中的遗传算法。仍在Postgres中进行应用，但一般只用于参与连接的关系数在13个以上，搜索空间过大的情况

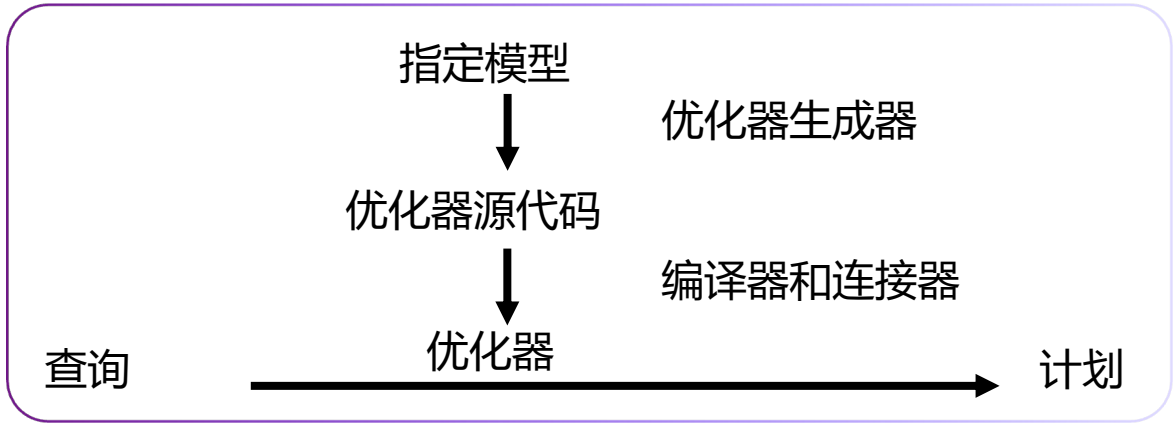


③ 随机化搜索

- **优点：**
  - 可以快速优化各种复杂查询
  - 能够跳出局部最优
  - 内存占用较小
  - 计划质量好于启发式，优化速度好于动态规划
- **缺点：**
  - 计划质量无法保证，无法在所有情形下都得到好的优化效果
  - 不可解释，难以确定选择某个计划的原因

优化器生成器：④分层+⑤统一搜索

- 用过程式语言编写变换规则既困难又容易产生错误：
  - 在不进行大量测试的情况下难以验证规则的正确性
  - 为每个逻辑算子确定物理算子与查询的深层语义信息脱钩
- 更好的方式是
  - 声明式DSL（domain-specific language）来编写转换规则，让优化器在计划过程中强制使用
- 使数据库开发人员只需编写用于优化查询的声明式规则，由优化器充当编译器的角色来自动地确定等价规则，并在优化过程中执行这些规则。



优化器生成器：④分层+⑤统一搜索

- 允许数据库的实现人员编写声明式的规则优化查询
  - 将搜索策略与数据模型分离
  - 将等价变换规则以及逻辑算子与物理规则和物理算子分离
- 利用优化器生成器，规则可以与优化器的搜索策略相独立，例子: Cascades
- 类型1:分层搜索
  - 分多个阶段（逻辑与物理）进行计划搜索
- 类型2:统一搜索
  - 同时进行逻辑和物理计划搜索

④ 分层搜索

- 首先使用等价转换规则重写逻辑计划
  - 应用规则前，检查变换是否允许使用
  - 在这一阶段，完全不考虑代价模型
- 之后进行基于代价的搜索，将逻辑计划转换为物理计划
- 分层搜索在20世纪80年代被提出，是IBM原型系统STARBURST中采用的方法，仍在Oracle和DB2中使用，基于启发式方法+基于代价选择连接顺序方法的改进。
- 不同之处在于，分层搜索利用了优化器生成器，在数据库内维护的不再是一个规则列表，而是一个规则引擎。在优化时，规则引擎以最初的逻辑计划为输入，会自动生成出需要使用的规则。

④ 分层搜索：Starburst优化器

- R系统更好的实现
- 使用声明式规则（优化器生成器）
- 阶段1:查询重写
  - 计算一个SQL块级的的查询关系演算表示
- 阶段2:物理优化
  - 执行与R系统类似的动态规划
- 例子：最新版本的IBM DB2

④ 分层搜索

- **优点：**
  - 实现简单，实践中验证了其高效性
- **缺点：**
  - 难以确定转换规则执行顺序
  - 由于在转换时完全不考虑代价的估计，一些转换的好坏在选择前难以进行评估
  - 维护转换规则比较困难

⑤ 统一搜索

- 统一搜索在20世纪90年代被提出，是Cascades采用的优化方法，仍在MSSQL和Greenplum数据库中被使用。
- 在统一搜索策略下，逻辑到逻辑的转换与逻辑到物理的转换都被统一成同一阶段进行的转换。
  - 整个查询计划的确定都在同一个阶段利用搜索完成。
  - 产生了许多转换，因此会大量使用了记忆化，以减少重复的计算。
- 一般也被称为“自顶向下”的方法

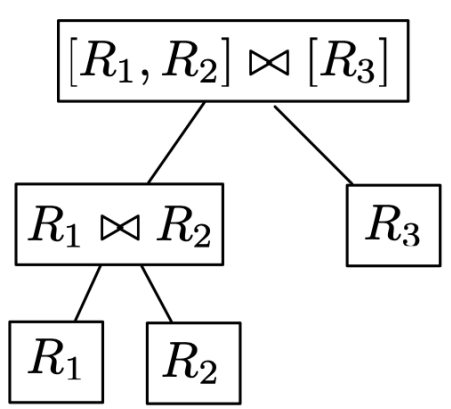
⑤ 统一搜索

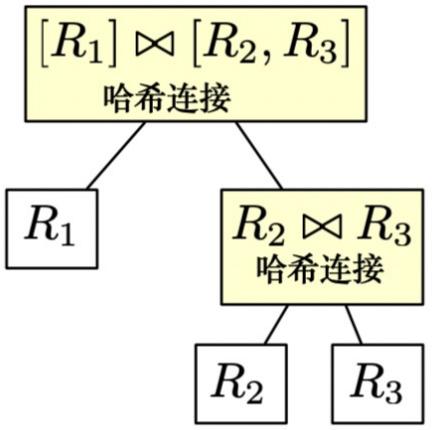
- **优点：**
  - 使用声明式规则生成等价转换，利于添加新规则
  - 规则扩展性好
  - 利用记忆化减少冗余的计算
  - 可以提前剪枝，提升表数较多时的计划生成速度
- **缺点：**
  - 所有的等价转换都会被使用，生成出所有可能的逻辑计划
  - 逻辑规则较多时，优化代价高

⑤ 统一搜索：Cascades优化器

- 自顶向下迭代使用规则进行优化，整个搜索空间会形成一个Operator Tree的森林。
- 将一组逻辑上等价的逻辑表达式以及它们对应的所有物理表达式定义为一个组
  - 逻辑2逻辑Implementation：逻辑算子可以转换成物理算子；例如Join转换成NestLoop或者HashJoin等
  - 逻辑2物理Transformation：逻辑算子可以做等价变换；例如交换Inner Join的两个子节点，即可枚举Join顺序
- 探索更多的表达式，按优先级对转换规则进行排序，并在搜索过程中，动态地调整排序

组		逻辑多重表达式	物理多重表达式
	$[R_1, R_2, R_3]$	$[R_1, R_2] \bowtie [R_3]$	$[R_1, R_2] \bowtie [R_3]$ 归并连接
		$[R_2, R_3] \bowtie [R_1]$	$[R_1, R_2] \bowtie [R_3]$ 哈希连接
		$[R_1, R_3] \bowtie [R_2]$	$[R_1, R_2] \bowtie [R_3]$ 嵌套循环
		$[R_1] \bowtie [R_2, R_3]$	$[R_2, R_3] \bowtie [R_1]$ 归并连接
		...	...





⑤ Cascades优化器: 表达式

SELECT * FROM R_1, R_2,R_3
WHERE $R_1.id = R_2.id$ AND $R_1.id = R_3.id$;

- **逻辑表达式示例：** $(R_1 \bowtie R_2) \bowtie R_3$

- **物理表达式示例** $(R_1 \bowtie R_2) \bowtie R_3$

  归并连接 嵌套循环

⑤ Cascades优化器: 组

➢ **将一组逻辑上等价的逻辑表达式以及它们对应的所有物理表达式定义为一个组**

- 一个表达式的所有逻辑等价表达式
- 为组内逻辑表达式选择可选的物理算子能够得到的物理表达式

组		逻辑表达式	物理表达式
	$[R_1, R_2, R_3]$	$(R_1 \bowtie R_2) \bowtie R_3$	$(R_1 \bowtie R_2) \bowtie R_3$ 归并连接 嵌套循环
		$(R_2 \bowtie R_3) \bowtie R_1$	$(R_2 \bowtie R_3) \bowtie R_1$ 哈希连接 归并连接
		$(R_1 \bowtie R_3) \bowtie R_2$	$(R_1 \bowtie R_3) \bowtie R_2$ 嵌套循环 哈希连接
		$R_1 \bowtie (R_2 \bowtie R_3)$	$R_1 \bowtie (R_2 \bowtie R_3)$ 哈希连接 归并连接
		...	...

⑤ Cascades优化器: 多重表达式

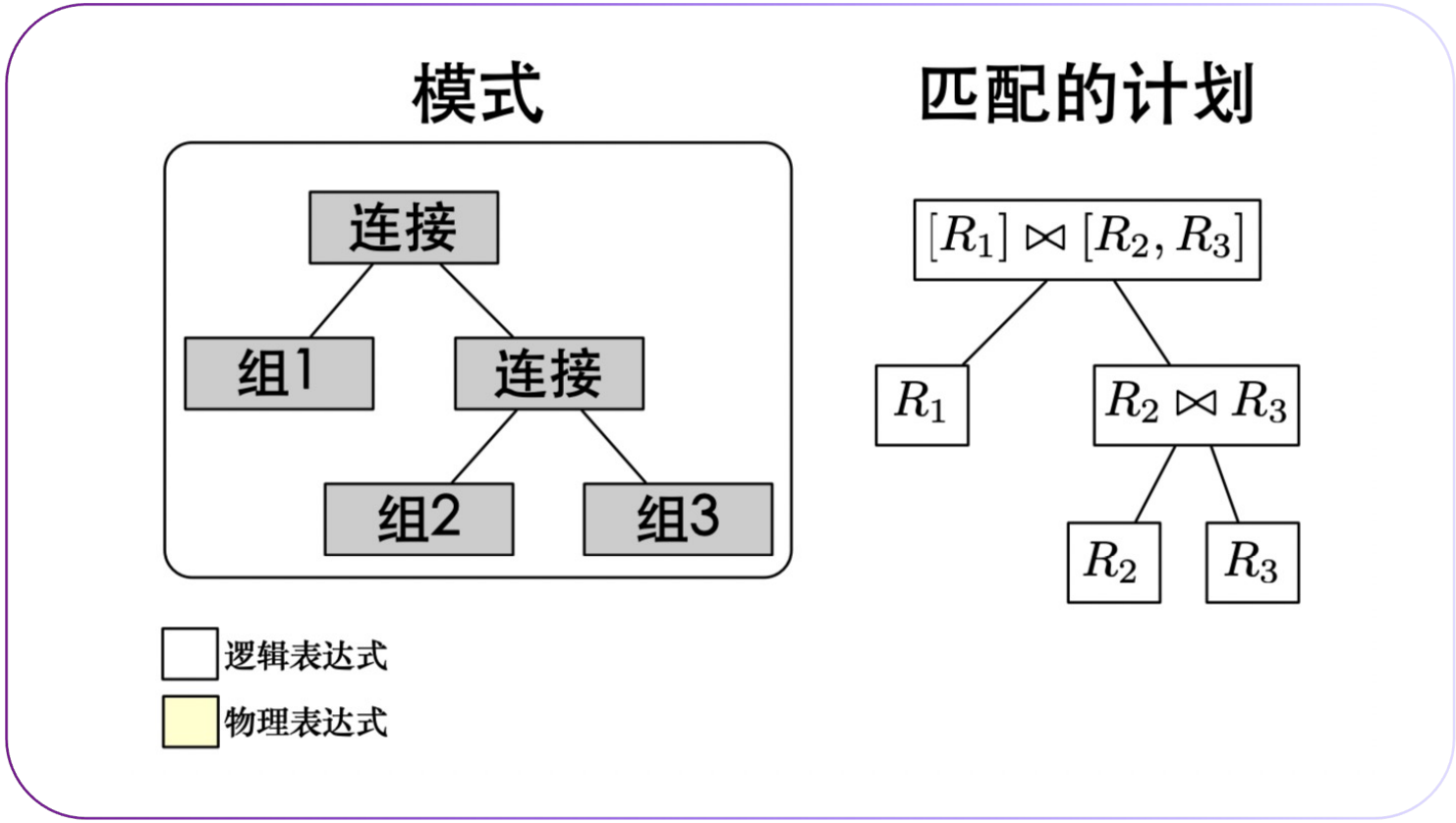
- 优化器不将一个组中所有可能的表达式实例化，而是隐式地将一个组中重复的表达式表示为一个多重表达式
  - 由此减少了变换个数，从而节省了空间开销，避免了重复的代价估计
  - 可以决定对多重表达式的搜索顺序

组		逻辑多重表达式	物理多重表达式
	$[R_1, R_2, R_3]$	$[R_1, R_2] \bowtie [R_3]$	$[R_1, R_2] \bowtie [R_3]$ 归并连接
		$[R_2, R_3] \bowtie [R_1]$	$[R_1, R_2] \bowtie [R_3]$ 哈希连接
		$[R_1, R_3] \bowtie [R_2]$	$[R_1, R_2] \bowtie [R_3]$ 嵌套循环
		$[R_1] \bowtie [R_2, R_3]$	$[R_2, R_3] \bowtie [R_1]$ 归并连接
		...	...

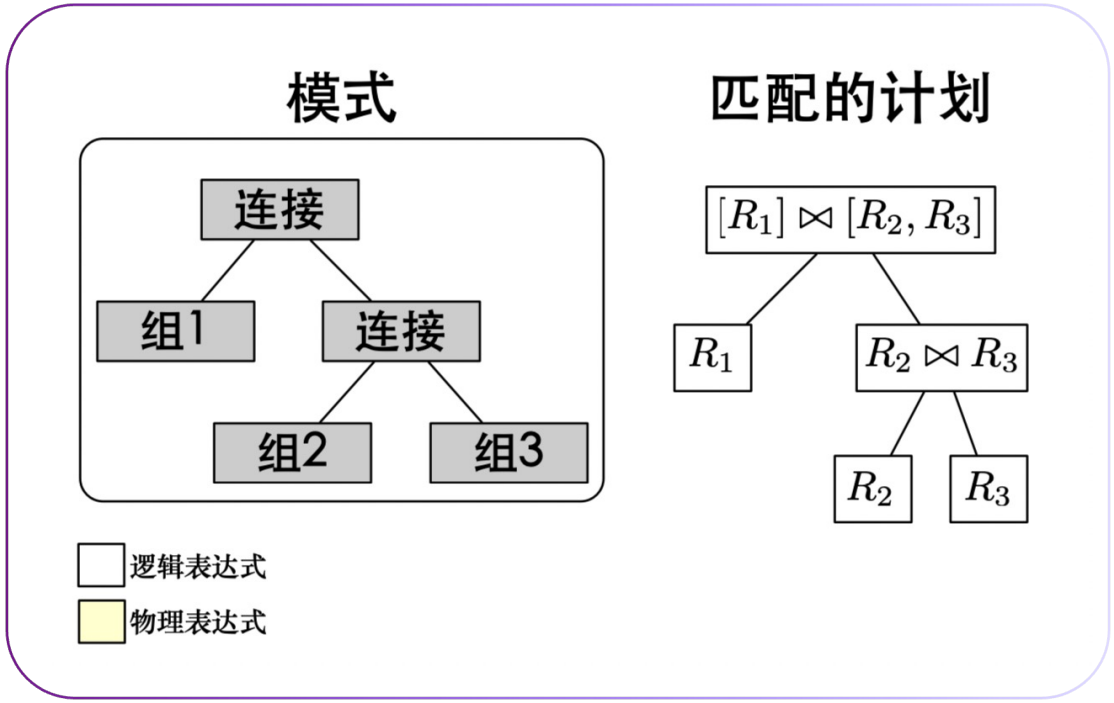
⑤ Cascades优化器: 规则

- 规则是将一个表达式转换为逻辑等价的表达式
  - 转换规则：逻辑到逻辑
  - 实现规则：逻辑到物理
- 每个规则都可以表示为一对属性：
  - 模式Pattern：定义可以应用规则的逻辑表达式的结构
  - 替换Substitute：定义应用规则产生的结果结构（转换后的）
  - Binding: (A join B) join C = A join (B join C) → (S join SC) join C = S join (SC join C)
- 规则的好坏：Promise

⑤ Cascades优化器: 规则

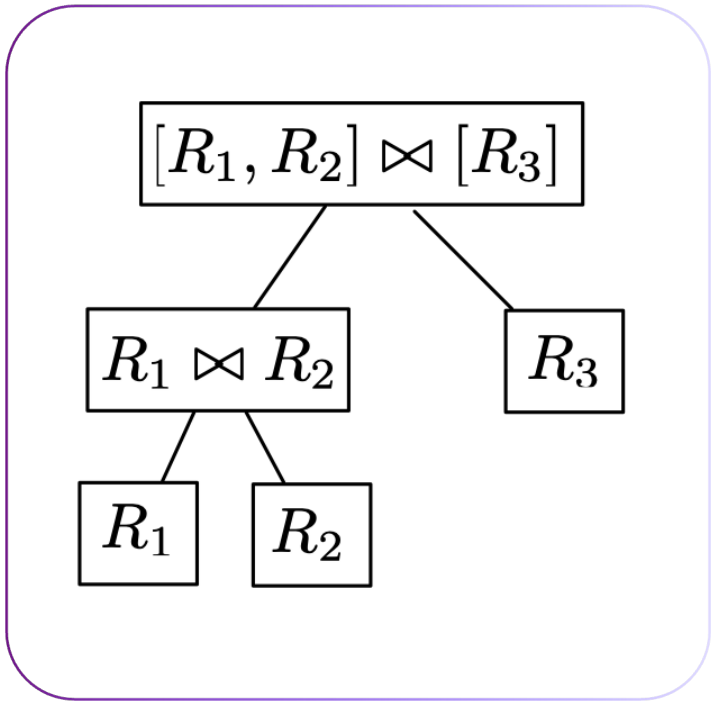


⑤ Cascades优化器: 规则

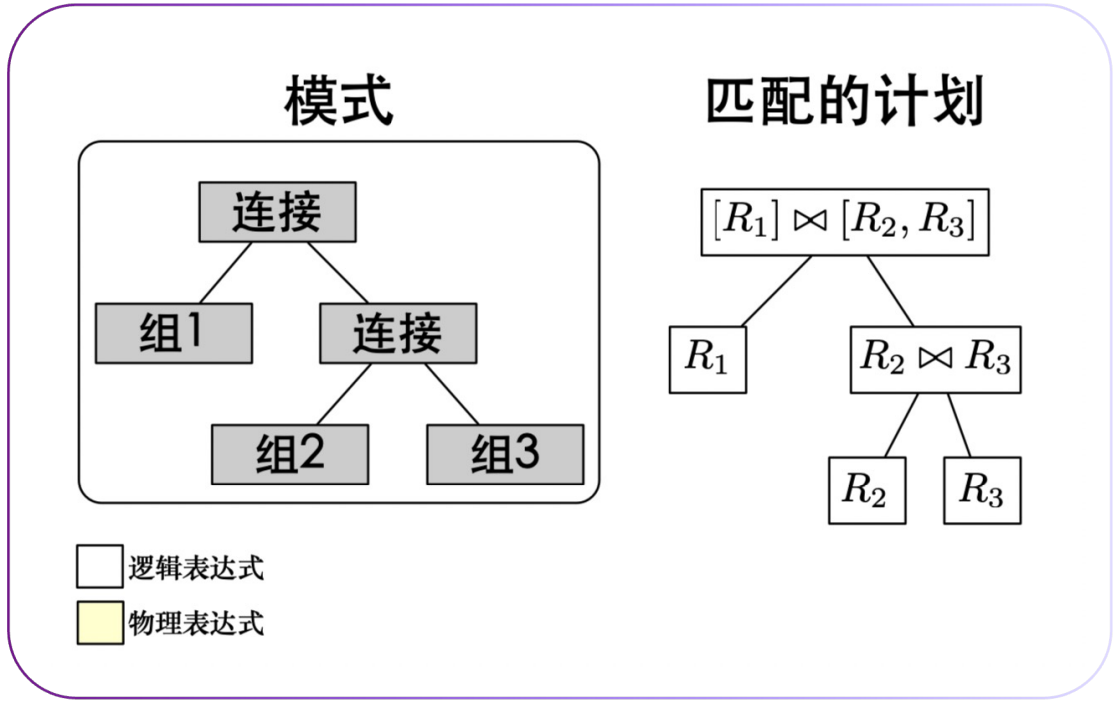


转换规则

交换连接顺序

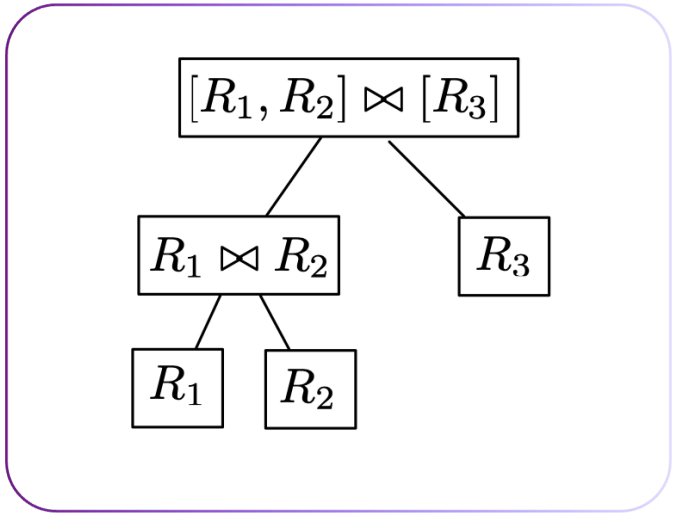


⑤ Cascades优化器: 规则



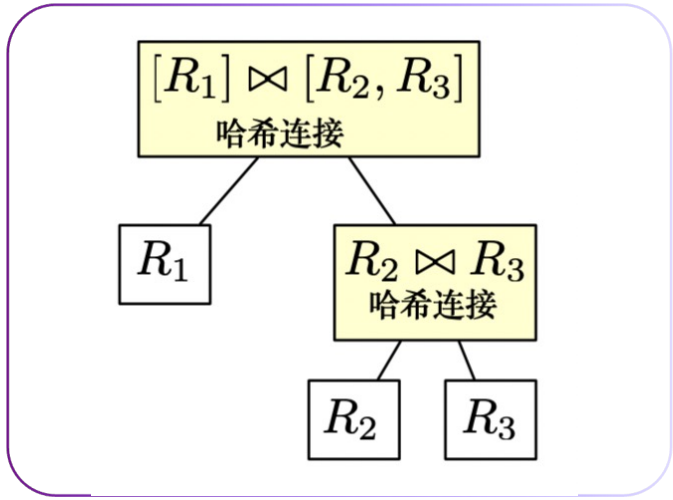
转换规则

交换连接顺序



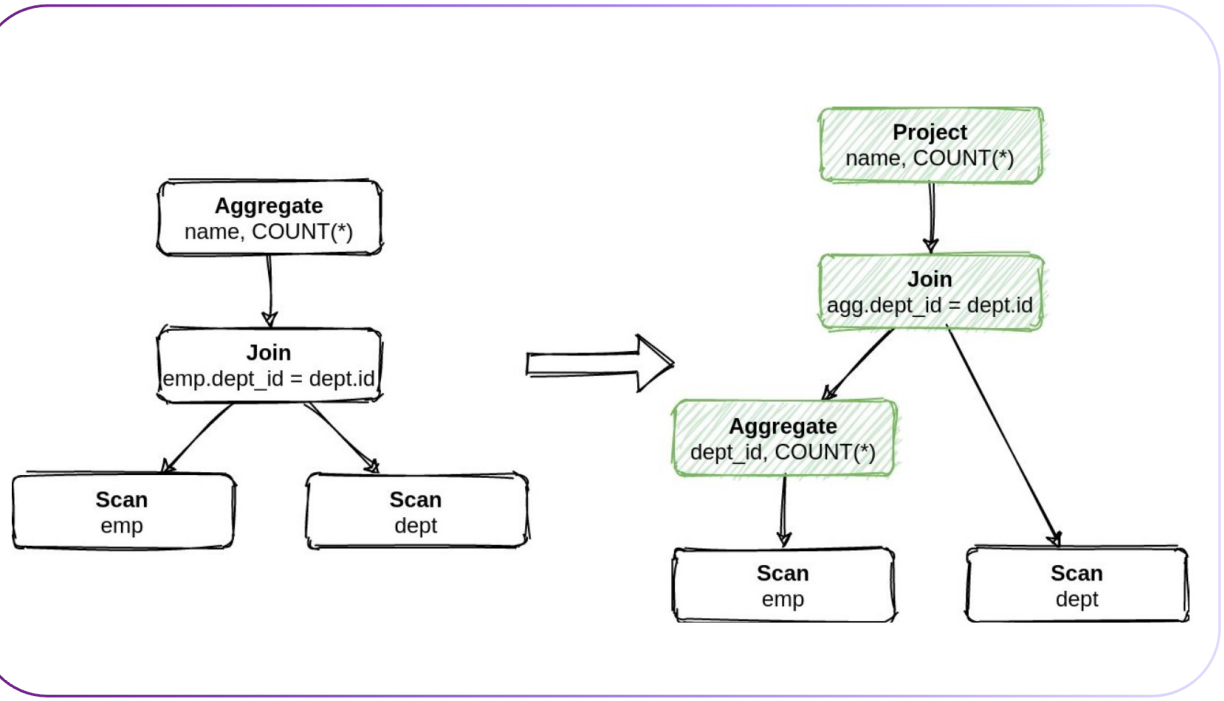
实现规则

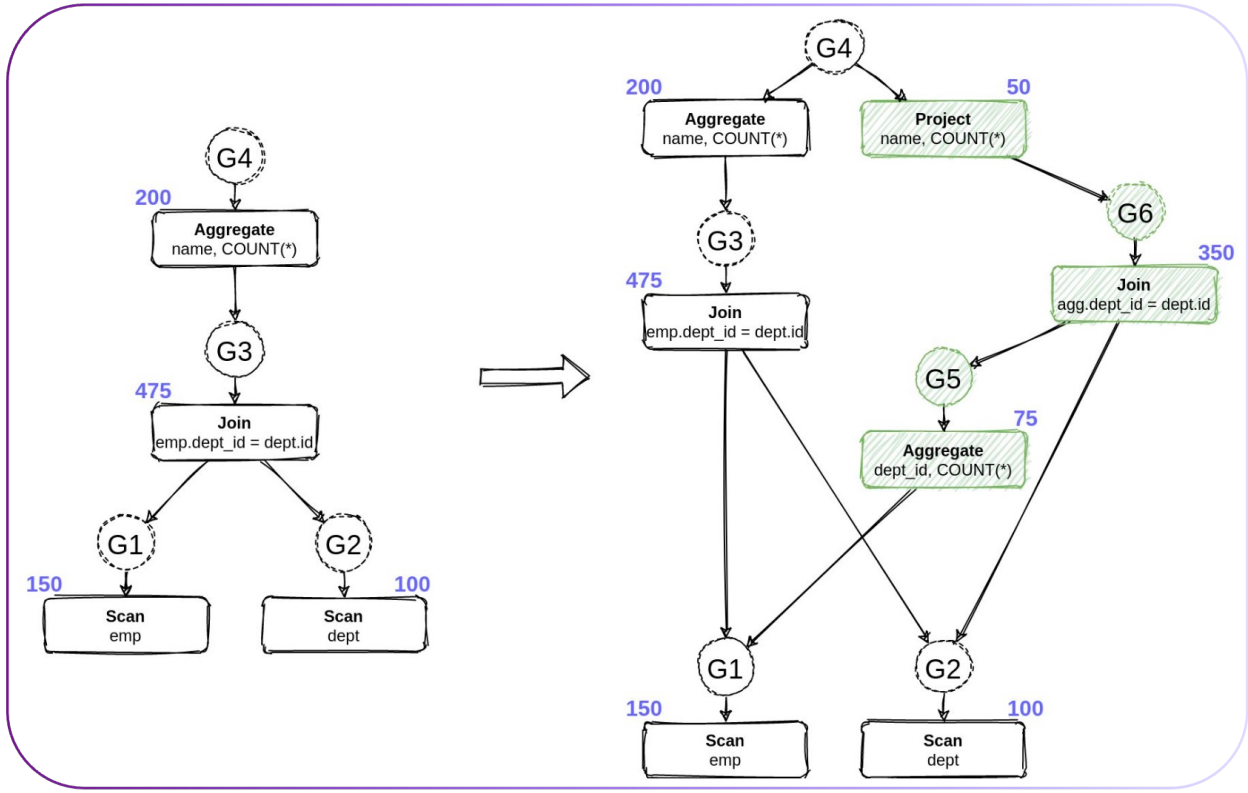
连接->哈希连接



⑤ Cascades优化器: 规则

- 基于规则的方法
- 基于代价的方法
- 基于AI的方法





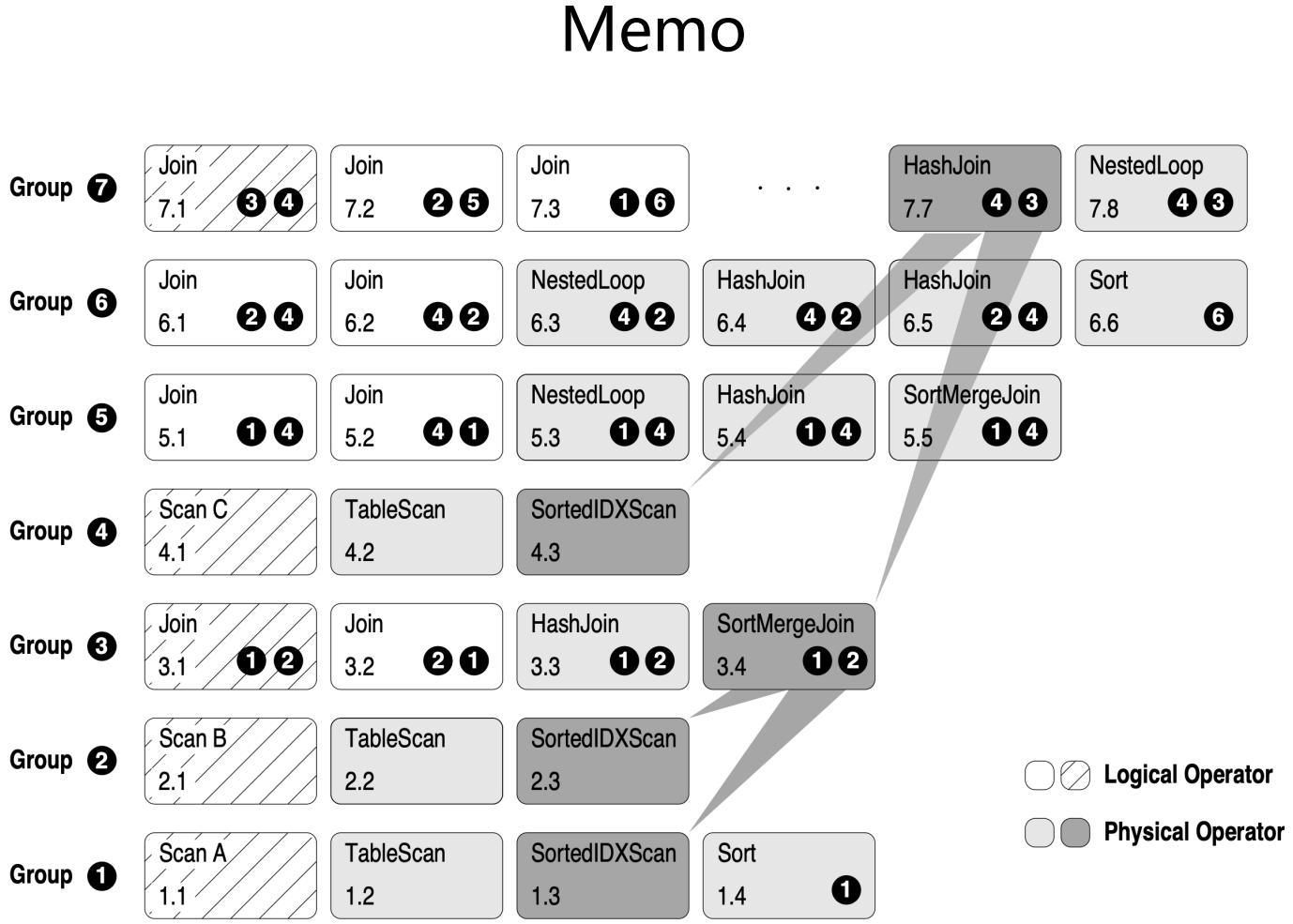
⑤ Cascades优化器: Memo表

- 在一个哈希表中存储所有以前搜索过的候选方案
- 等价的计划成组地存储在一起
- 使用记忆化，避免重复计算
- 搜索中动态维护Memo表
- 最终$[R_1, R_2, R_3]$的最优表达式即为所求

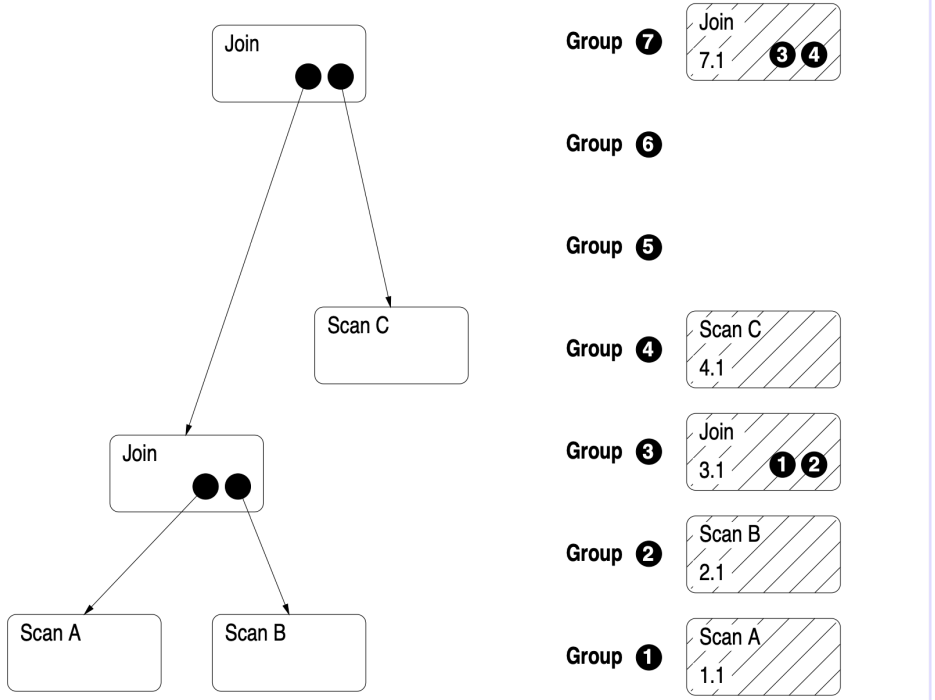
	最优表达式	代价
$[R_1, R_2, R_3]$	$[R_1] \bowtie [R_2, R_3]$ 哈希连接	40+(30+80)=150
$[R_1, R_2]$	$[R_1] \bowtie [R_2]$ 归并连接	200+(30+20)=250
$[R_1, R_3]$	$[R_1] \bowtie [R_3]$ 哈希连接	80+(30+10)=120
$[R_2, R_3]$	$[R_2] \bowtie [R_3]$ 哈希连接	50+(20+10)=80
$[R_1]$	线性扫描 R_1	30
$[R_2]$	线性扫描 R_2	20
$[R_3]$	线性扫描 R_3	10

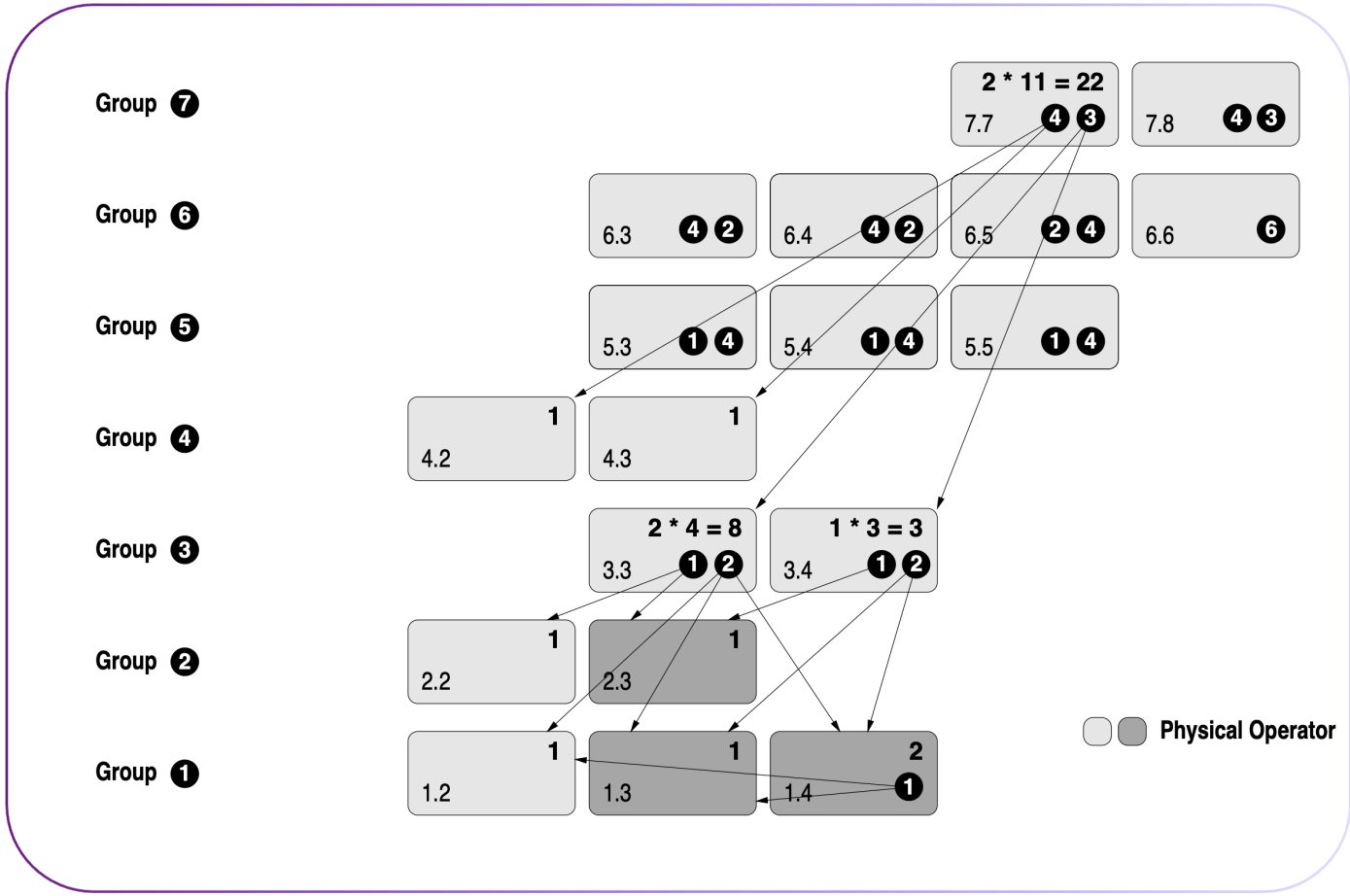
⑤ Cascades优化器：Memo

- Memo:
  - Vector<Group>
- Group:
  - Map<Property, GroupExpression>
  - Property：属性如Join、Selection
  - GroupExpression：子Group
- GroupExpression
  - Vector<GroupID> children
  - 记录子表达式
- 通过动态规划DP获得一个完整计划



⑤ Cascades优化器：Memo

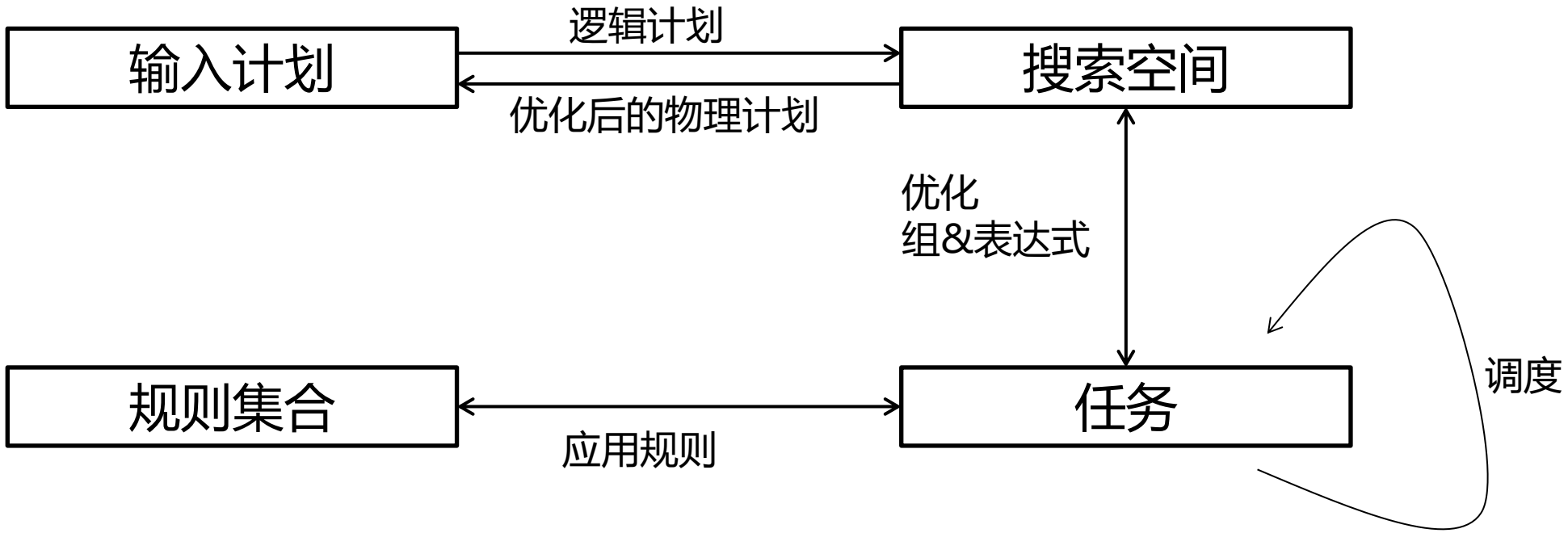




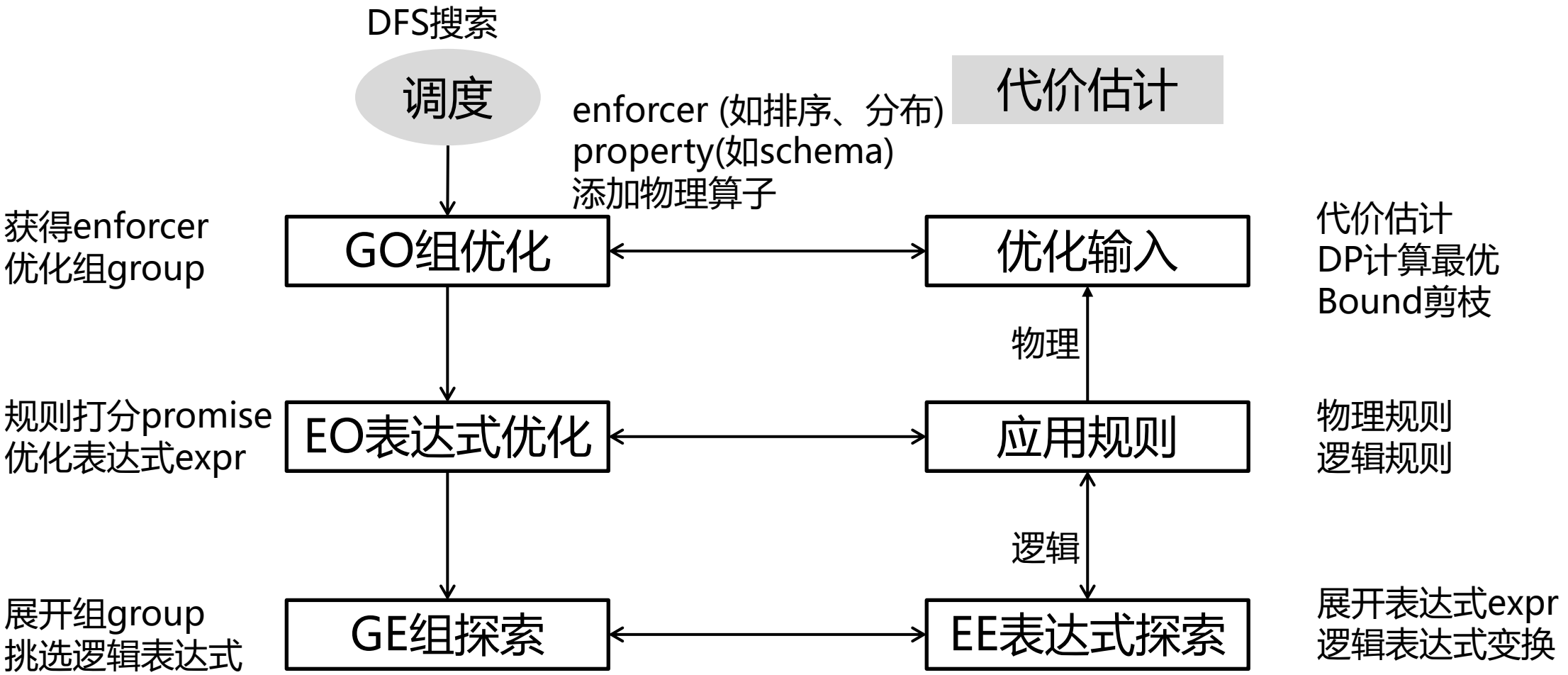
⑤ Cascades优化器：需要解决的问题

- 调度流程
  - 探索Group和Expression
  - 应用规则
- 计算可行的物理计划
  - 动态规划（区别于自底向上：利用Memo里Group和Expression来计算）
- 剪枝
  - 保留当前代价最低的物理计划Winner，并记录其代价upperbound
  - 估算Group内的计划的lowerbound
  - 剪掉lowerbound>upperbound的group

⑤ Cascades优化器：流程

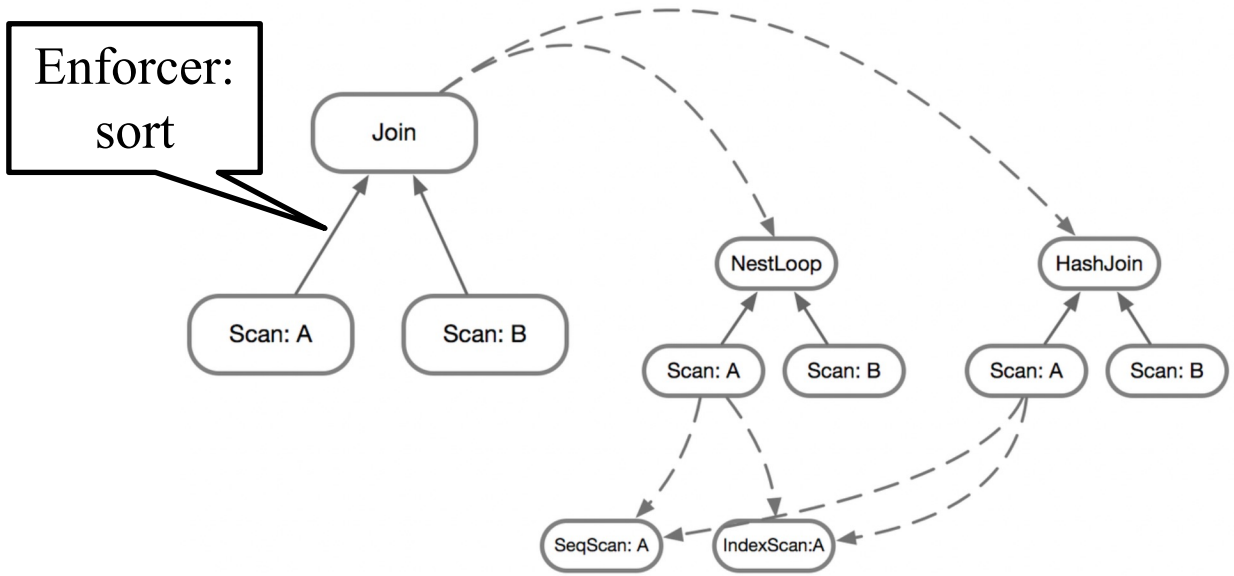


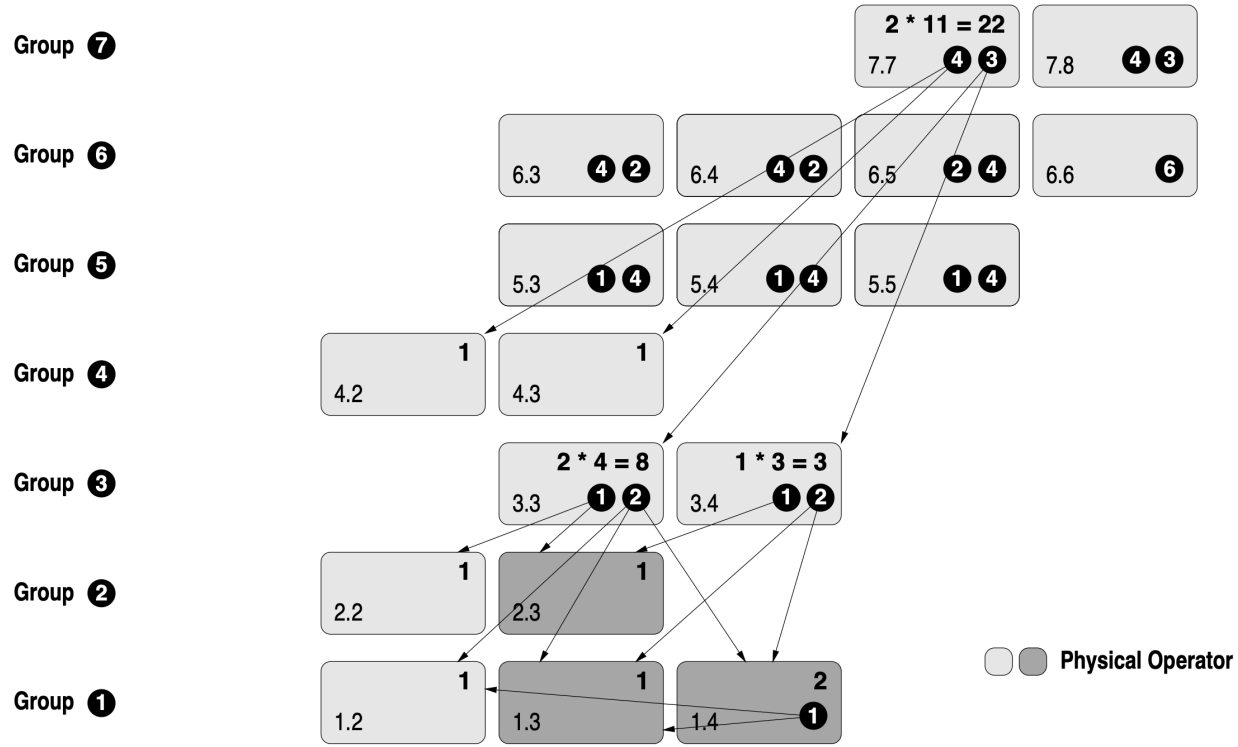
⑤ Cascades优化器：流程



⑤ Cascades优化器: 最优原则

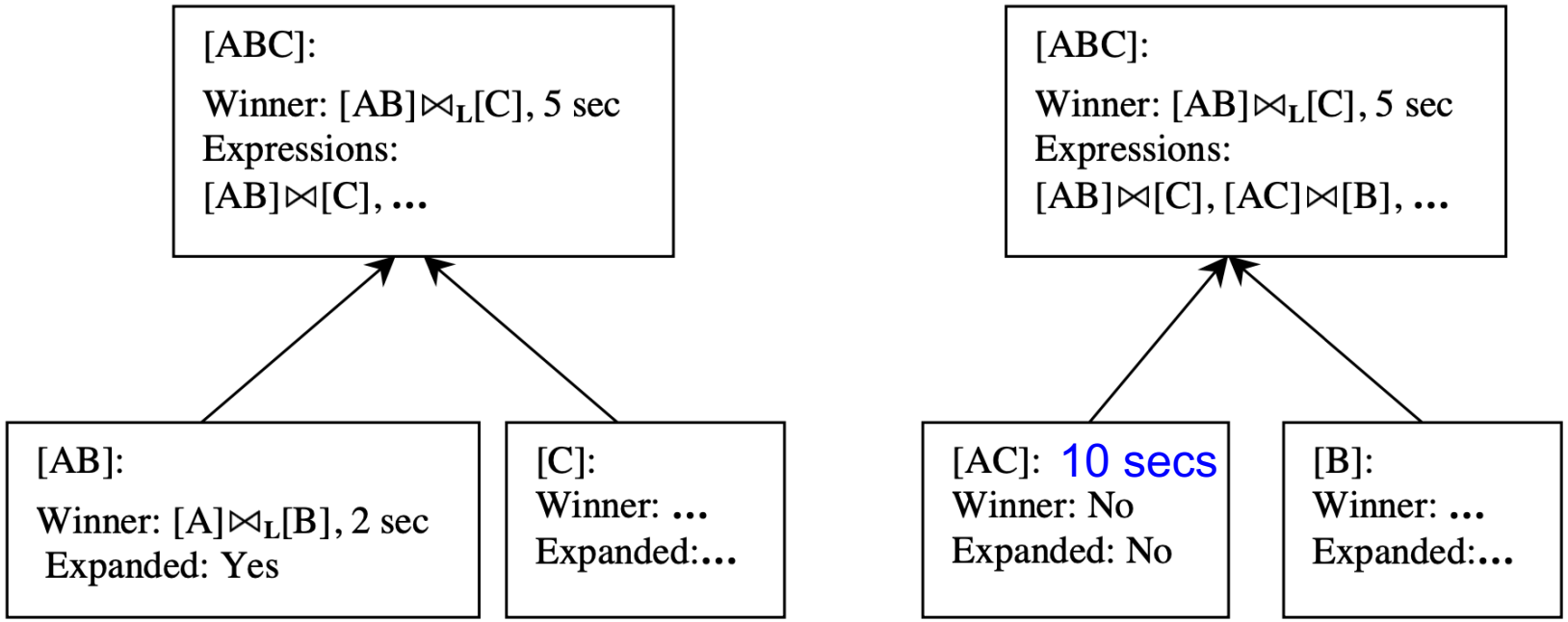
- Cascades中认为最优计划的每个子计划也是最优的
- 将搜索空间限制在一个较小的范围中，通过DP计算最优
  - 不需要搜索含有非最优子计划的候选计划
  - 复用已经计算的子树
  - 效率更高





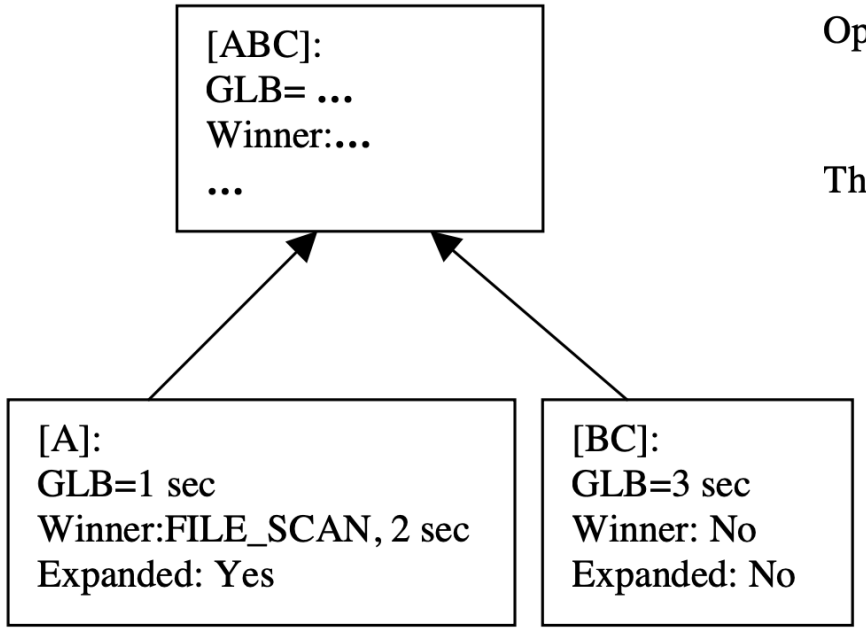
⑤ Cascades优化器: 剪枝

- 首先获得可行的物理计划，并得到一个上界upperbound
- 如果一个子计划lowerbound超过upperbound，则可以剪枝



⑤ Cascades优化器: 剪枝

- 计划lowerbound计算方法
  - 以下三个求和
    - 当前表达式计算代价
    - 子计划代价
    - 其他的全局组最小代价GLB

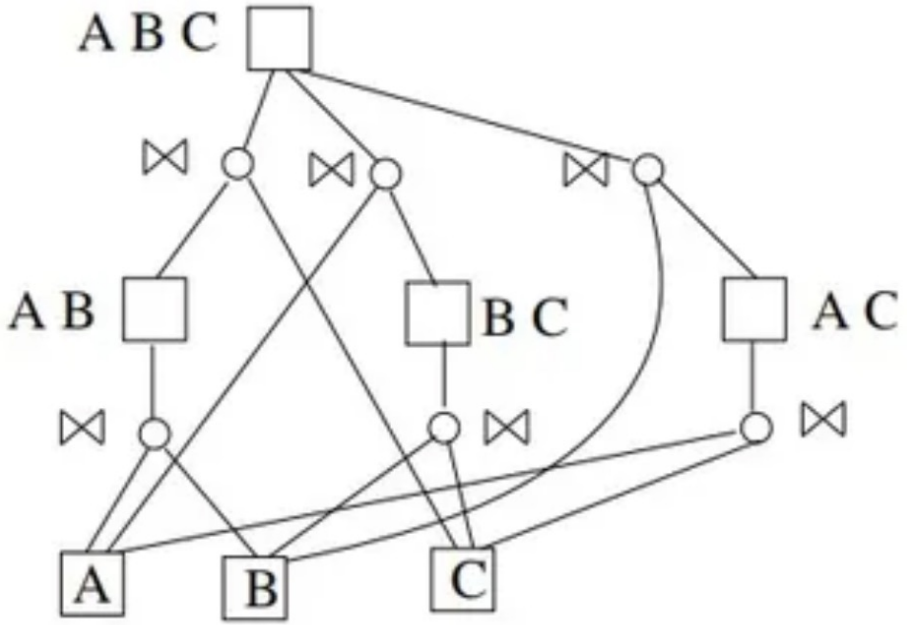


Optimizing $[A] \bowtie_L [BC]$:
Context: UpperBound = 5 sec
Cost of $\bowtie_L$ = 1 sec
Then:
Pruning happen for [BC] since:
Winner[A]+GLB[BC]+Cost[$\bowtie_L$] =
2 + 3 +1 = 6
greater than UpperBound = 5

[BC] will not be expanded.

⑤ Cascades优化器: 枚举连接顺序

- 目的：避免连接顺序重复和子Group不连通性
- 传统方法：通过哈希表避免重复
- 优化方法：无向图划分
  - DPCCP(Bottom-Up)
  - DPHyp(Bottom-Up)
  - MinCutBranch(Top-Down)



⑤ Cascades优化器: 搜索终止条件

- **方案1:时钟时间**
  - 在优化器搜索指定时间后停止
- **方案2:代价阈值**
  - 当优化器找到一个代价低于某个阈值的计划时停止
- **方案3:穷尽转换**
  - 当没有更多的方法来转换目标计划时停止。通常是按组进行

⑤ Cascades优化器：实现

➢类别1:独立优化器

- Wisconsin OPT++ (1990s)
- Portland State Columbia (1990s)
- Pivotal Orca (2010s)
- Apache Calcite (2010s)

➢类别2:集成式

- Microsoft SQL Server (1990s)
- Tandem NonStop SQL (1990s)
- Clustrix (2000s)
- CMU Peloton (2010s – RIP)

优化器系统

① 优化器系统的计划生成

② 物化与流水线

物化与流水线

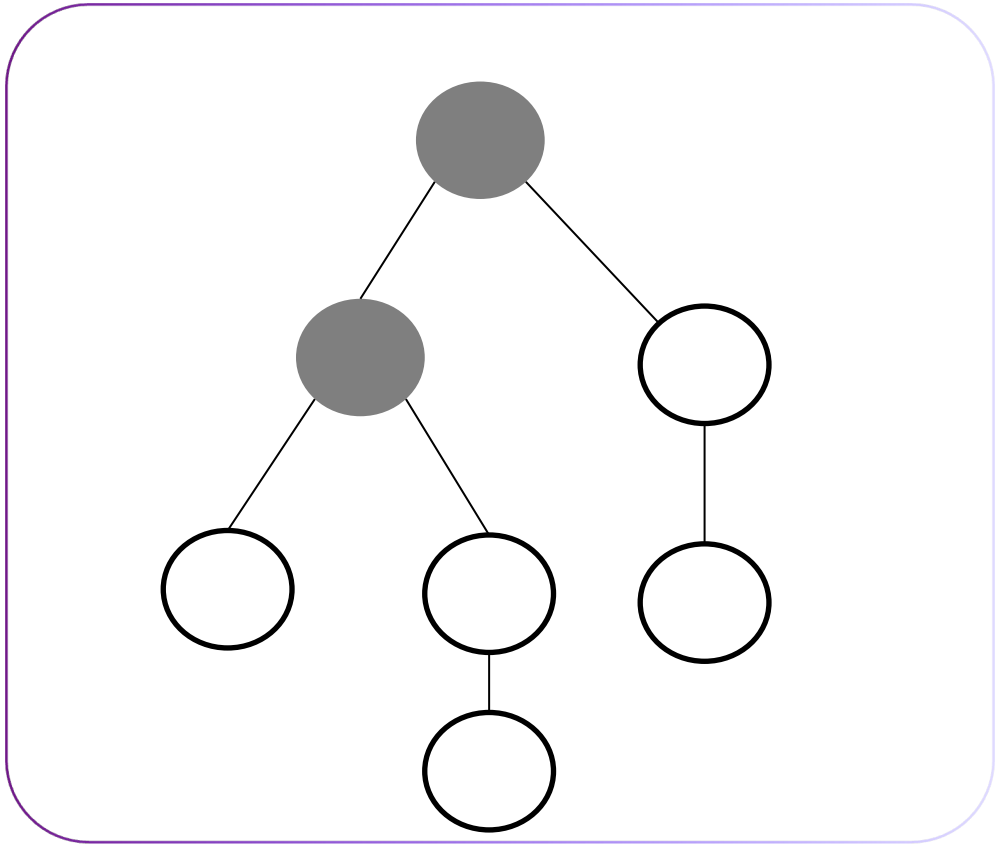
- 物化：将每个运算的结果存储为磁盘上的临时关系，会带来大量磁盘读写的开销
- 流水线：得到一个操作的部分结果时就会将数据传递到下一个操作中，避免了临时文件的读写
- 使用流水线执行能够减少磁盘的I/O，因而在内存足够的情况下流水线方法一般总是更好的选择。

流水线方法的限制

- 并不是任何操作都可以使用流水线执行
- 能够流水线执行的运算需要像选择、投影运算一样，在收到部分输入元组后就可以开始输出运算产生的结果元组
- 存在阻塞的算子不能流水线执行：
  - 如排序运算在所有输入元组输入完成前处于阻塞的阶段，在这时无法输出任何结果，因此不能采用流水线执行
  - 一些运算本身不会阻碍流水线，但物理执行算法可能会阻碍
    - 哈希连接需要先对内外关系建立哈希表，在没有收到完整的输入元组前无法完成，因此无法流水线执行。
    - 而如果要进行连接的两个关系都是有序的，归并连接就是可以流水线执行的

物化与流水线示例

- 灰色节点：物化执行
- 白色节点：流水线执行
- 对于这些子树内部的其他运算符，数据库则会采用流水线的方式近似于同步地执行



目录

物化视图

- 视图：一个经过命名存储在数据库中的查询，又被称为“虚关系”
  - 因为定义视图的查询的结果并没有存储在数据库中，每次调用视图都需要按照其定义重新执行查询
- 物化视图：将定义视图的结果存储到磁盘上
  - 在遇到使用物化视图的查询时可以直接调用存储的结果，以此避免重复查询而节省大量的查询执行时间
  - 空间换时间

物化视图

- 假设该SQL是个频繁执行的查询，则可以将该查询按如下SQL语句声明为一个物化视图存储在磁盘上：

```
SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno;
```

```
CREATE MATERIALIZED VIEW T
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
AS SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno;
```

- 立即执行定义该物化视图的查询
- 仅在该物化视图需要被刷新时才增量地更新物化视图

物化视图的维护方式

- 核心问题：当定义物化视图的基表数据发生改变时，如何保证物化视图的数据能够高效地被更新
- 当基本关系发生变化时，需要对物化视图进行更新，从而使物化视图与其派生自的基本关系保持一致
- 按照何时更新可分为：
  - 立即视图维护
  - 延迟视图维护
- 视图维护方式可分为：
  - 重新建立视图
  - 增量视图维护

增量维护

- **选择运算：**
  - 物化视图$M = \sigma_p(R)$
  - 插入T：$M' = M \cup \sigma_p(T)$
  - 删除T：$M' = M - \sigma_C(T)$

增量维护

➢ 投影运算：

- 物化视图$M = \Pi_S(R)$
- 需要为R中每个元组在属性集S上取值的出现次数$Count$进行记录
- 插入：
  - $Count'(\Pi_S(T)) = Count(\Pi_S(T)) + 1$
  - 如果原计数次数为0，$M' = M \cup \Pi_S(T)$
- 删除：
  - $Count'(\Pi_S(T)) = Count\big(\Pi_S(T)\big) - 1$
  - 如果新计数次数为0，$M' = M - \Pi_S(T)$

增量维护

➢ 连接运算：

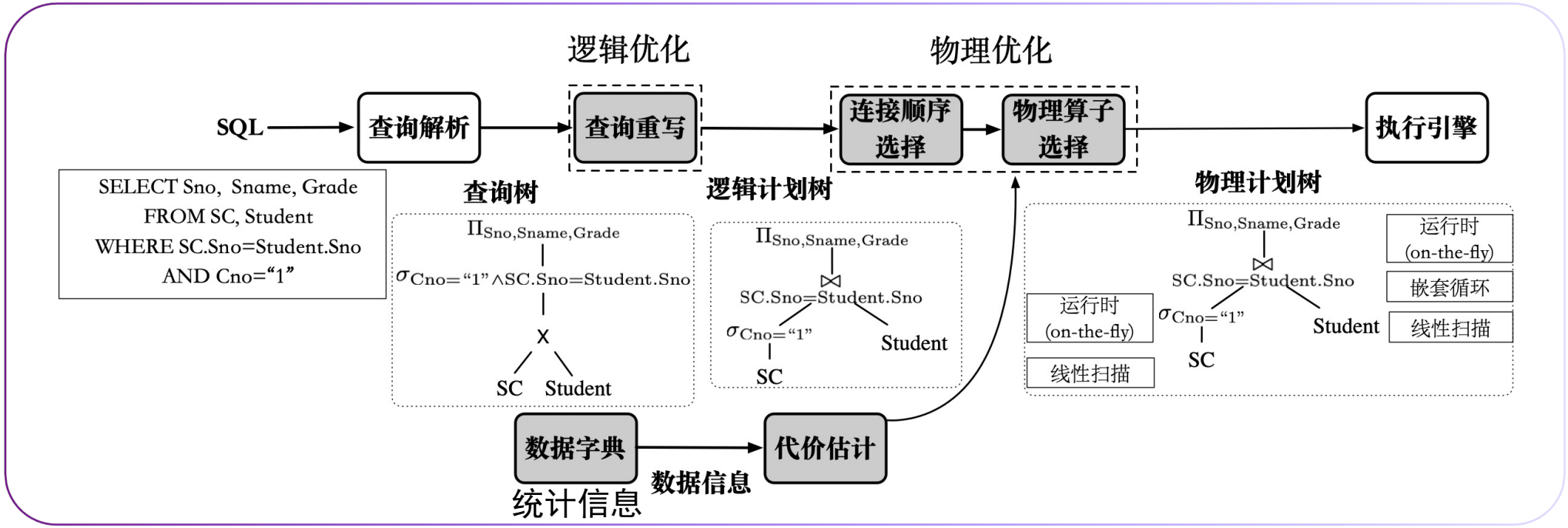
- 物化视图$M = R_1 \bowtie R_2$，表示关系R_1、R_2进行连接运算的结果
- 根据对称性，只需考虑对R_1的插入或删除
- 当R_1插入一个元组T时：
  - $M' = M \cup (T \bowtie R_2)$
- 当从R_1中删除一个元组T时：
  - $M' = M - (T \bowtie R_2)$

增量维护

- **去重运算：**
  - 物化视图$M = \delta(R)$，该视图表示对关系R进行去重的结果
  - 去重可以视作是**投影属性集S 是关系R 中所有属性**的**特殊投影**
  - 可以按照与投影运算相同的方法处理

查询优化总结

- 学生表 Student(Sno ,Sname, Sgender, Sage, Sdept)
- 学生选课表 SC(Sno, Cno, Grade)
- 查询：获取选修了1号课程的学生学号、姓名及成绩



查询优化总结

- 逻辑优化
  - 查询重写
- 物理优化
  - 基数估计
  - 代价估计
  - 连接顺序选择
  - 物理算子选择
- 优化器模型
  - 自顶向下、自下向上
- 物化与流水线
- 物化视图