查询计划优化与执行

徐辰
华东师范大学
数据科学与工程学院
cxu@dase.ecnu.edu.cn

关系数据库的查询处理过程

- SQL → Plans → Best Plan → Results
- SQL　　　　　　　　Plans

```
SELECT Distinct C.name,C.type,I.ino
FROM Customer C, Invoice I
WHERE I.amount>10000 AND
C.country="Sweden" AND
C.cno=I.cno
ORDER BY amount DESC
```

$\Pi_{name,type,ino}(\sigma_{amount>10000 \wedge country = 'Sweden'}(Customer \bowtie Invoice))$

$\Pi_{name,type,ino}(\sigma_{amount>10000}(Invoice) \bowtie \sigma_{country = 'Sweden'}(Customer))$

$\Pi_{name,type,ino}(\sigma_{amount>10000}(Invoice \bowtie \sigma_{country = 'Sweden'}(Customer)))$

$\Pi_{name,type,ino}(\sigma_{country = 'Sweden'}(\sigma_{amount>10000}(Invoice) \bowtie Customer))$

关系数据库的查询处理过程

- SQL → Plans：Interpretation
- Plans → Best Plan：Query Optimization
- Best Plan → Results：Query Evaluation

查询优化

- 一个SQL查询对应若干查询计划：
  - 操作的执行顺序可以不同；
  - 每个操作的执行算法可以不同；（扫描 vs 使用索引 ......）
  - 随着查询复杂度的增加，查询计划的数量呈指数增长。
- 查询优化：找到尽可能优的查询计划。

查询优化

- 基于规则的优化：
  - 如果有聚簇索引，则使用索引；
  - 先做Selection，再做Join；
  - 如果参与Join的其中一个表较小，则使用Hash Join ……
- 基于代价的优化：
  - 估算每个候选查询的I/O代价；
  - 选择 I/O代价最小的查询

Query Optimization

--> Generating and comparing plans

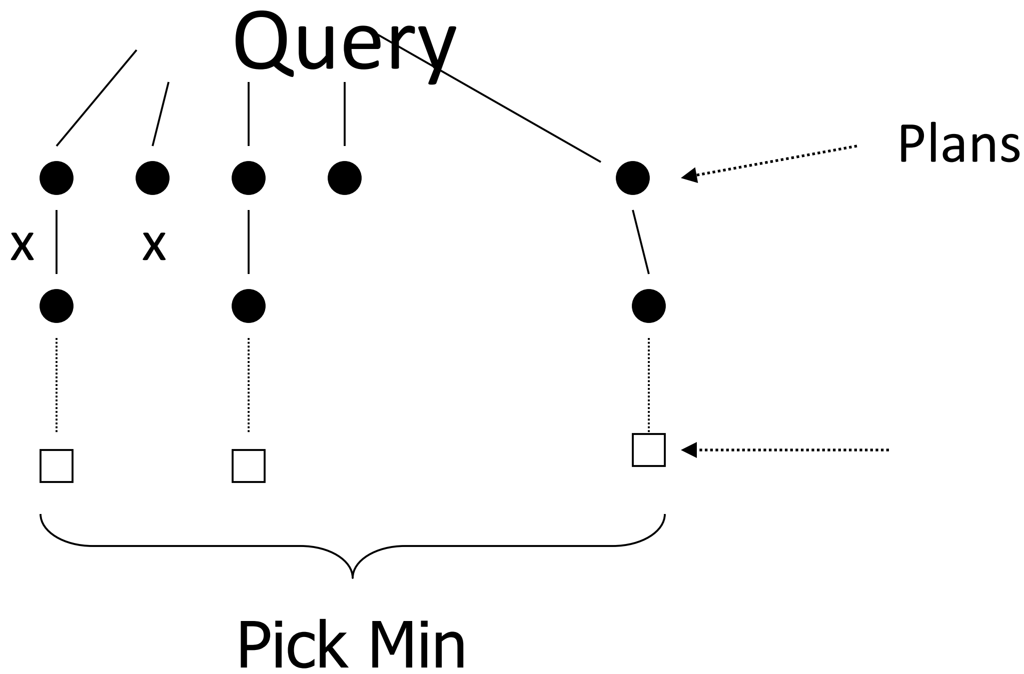
Generate

Pruning

Estimate Cost

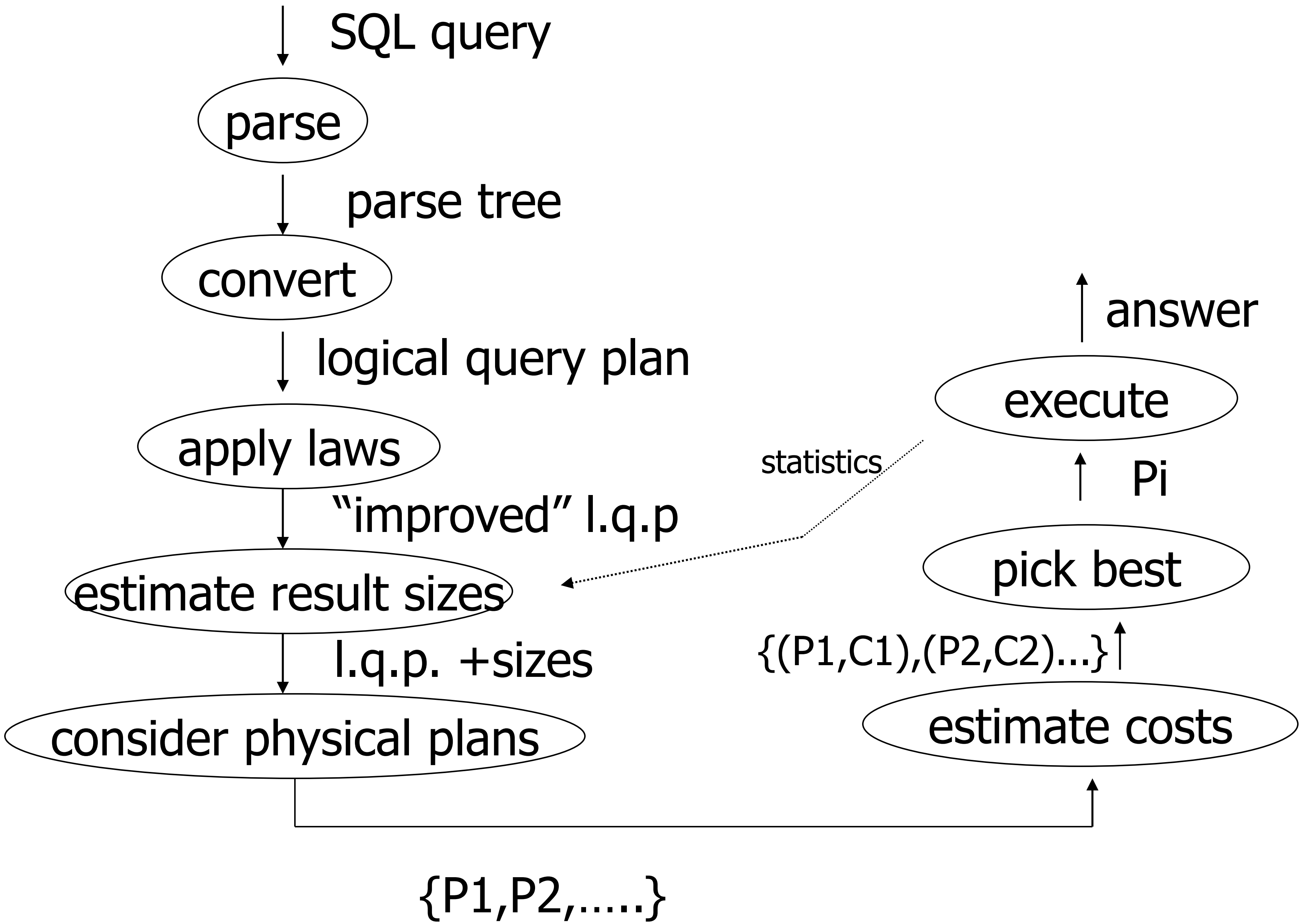
Cost

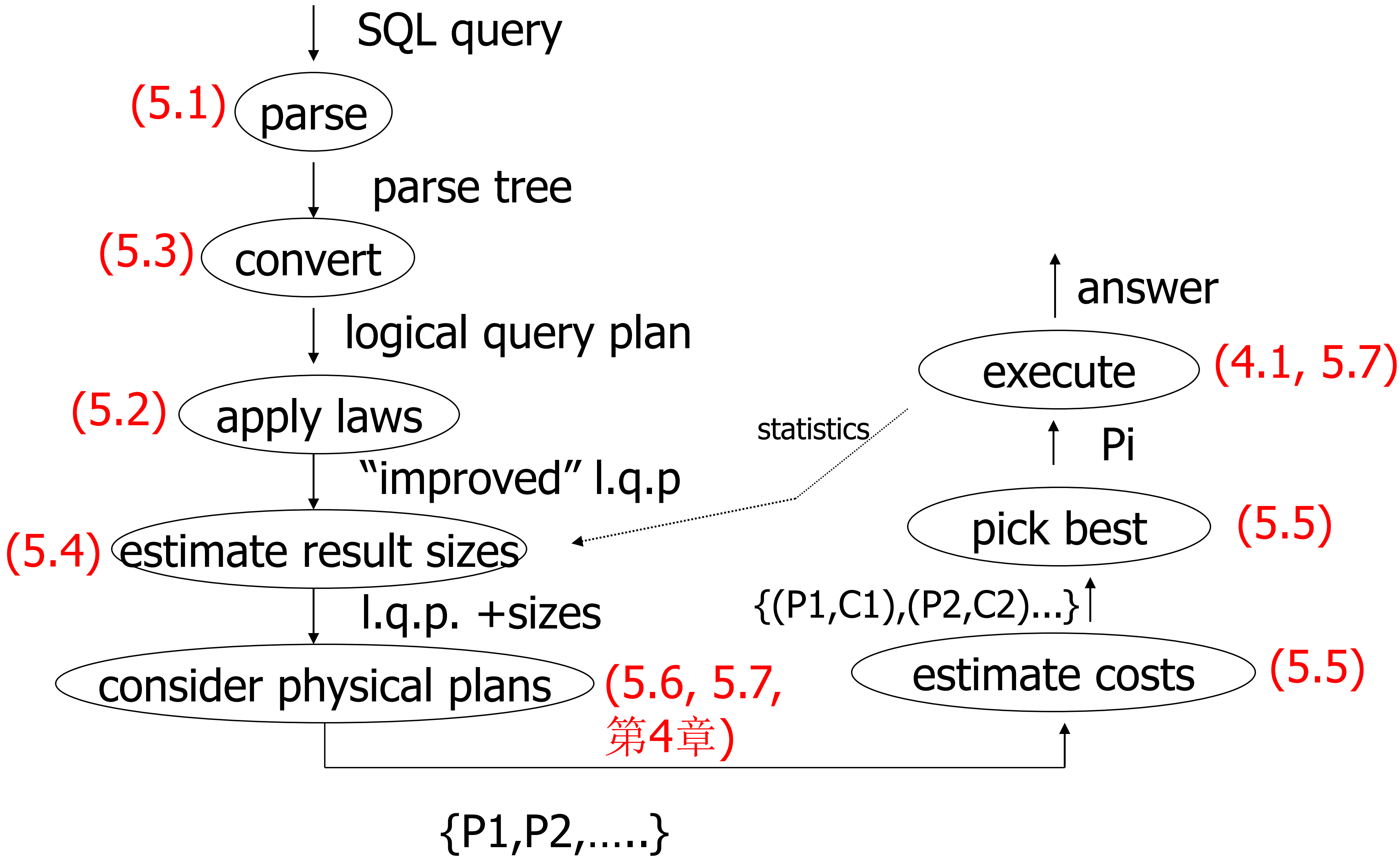
Select



Query Optimization -

- Relational algebra level (逻辑计划优化)
- Detailed query plan level (物理计划优化)
  - Estimate Costs
    - without indexes
    - with indexes
  - Generate and compare plans


SQL query
parse
parse tree
convert
logical query plan
apply laws
“improved” l.q.p
estimate result sizes
l.q.p. +sizes
consider physical plans
{P1,P2,…..}
estimate costs
{(P1,C1),(P2,C2)...}
pick best
Pi
execute
answer
statistics

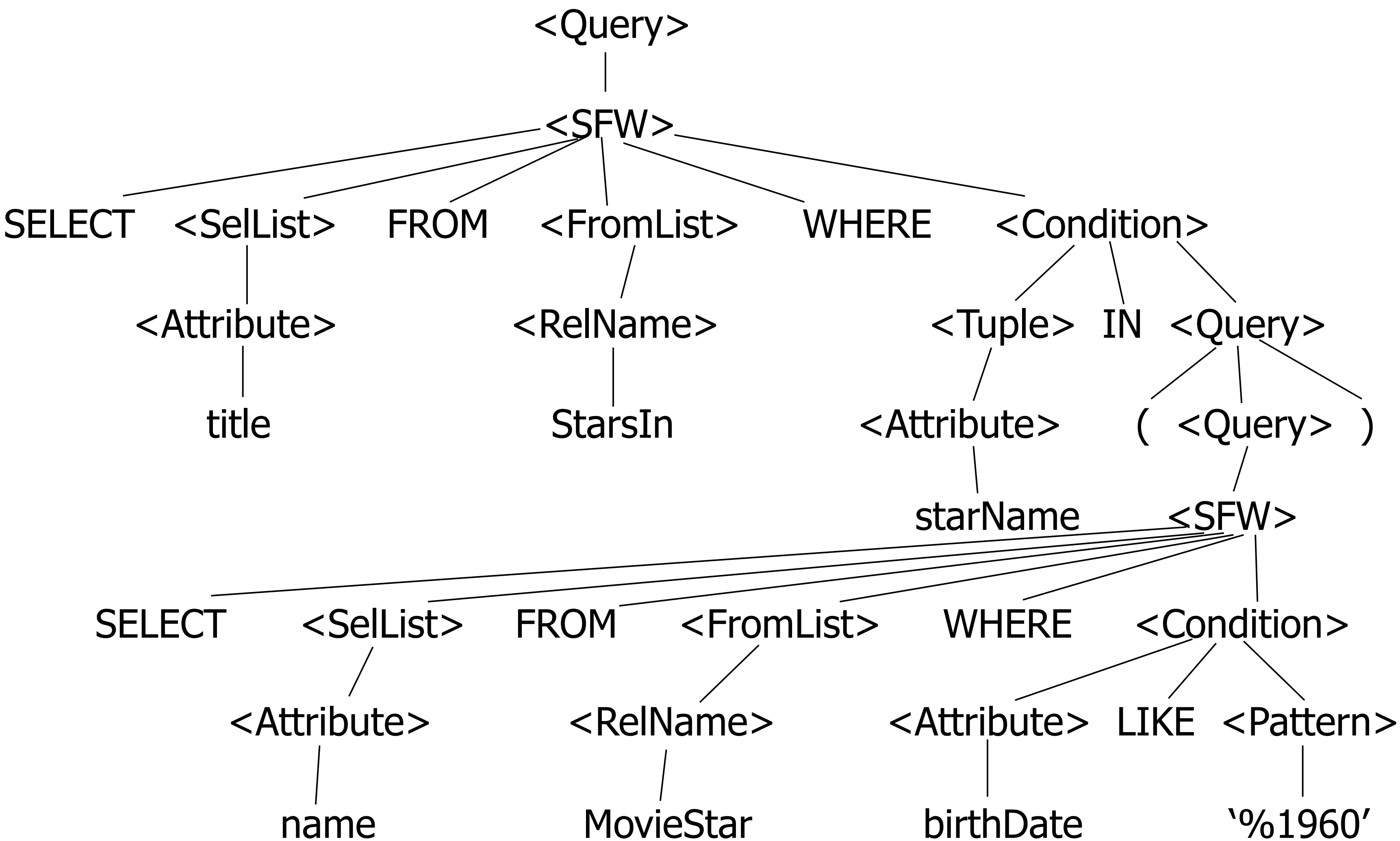

SQL query
(5.1)
parse
parse tree
(5.3)
convert
logical query plan
(5.2)
apply laws
"improved" l.q.p
(5.4)
estimate result sizes
l.q.p. +sizes
consider physical plans
(5.6, 5.7, 第4章)
{P1,P2,.....}
estimate costs
(5.5)
{(P1,C1),(P2,C2)...}
pick best
(5.5)
Pi
execute
(4.1, 5.7)
answer
statistics

Example: SQL query

```
SELECT title
FROM StarsIn
WHERE starName IN (
        SELECT name
        FROM MovieStar
        WHERE birthdate LIKE ‘%1960’
);
```

(Find the movies with stars born in 1960)

Example: Parse Tree



Example: Generating Relational Algebra

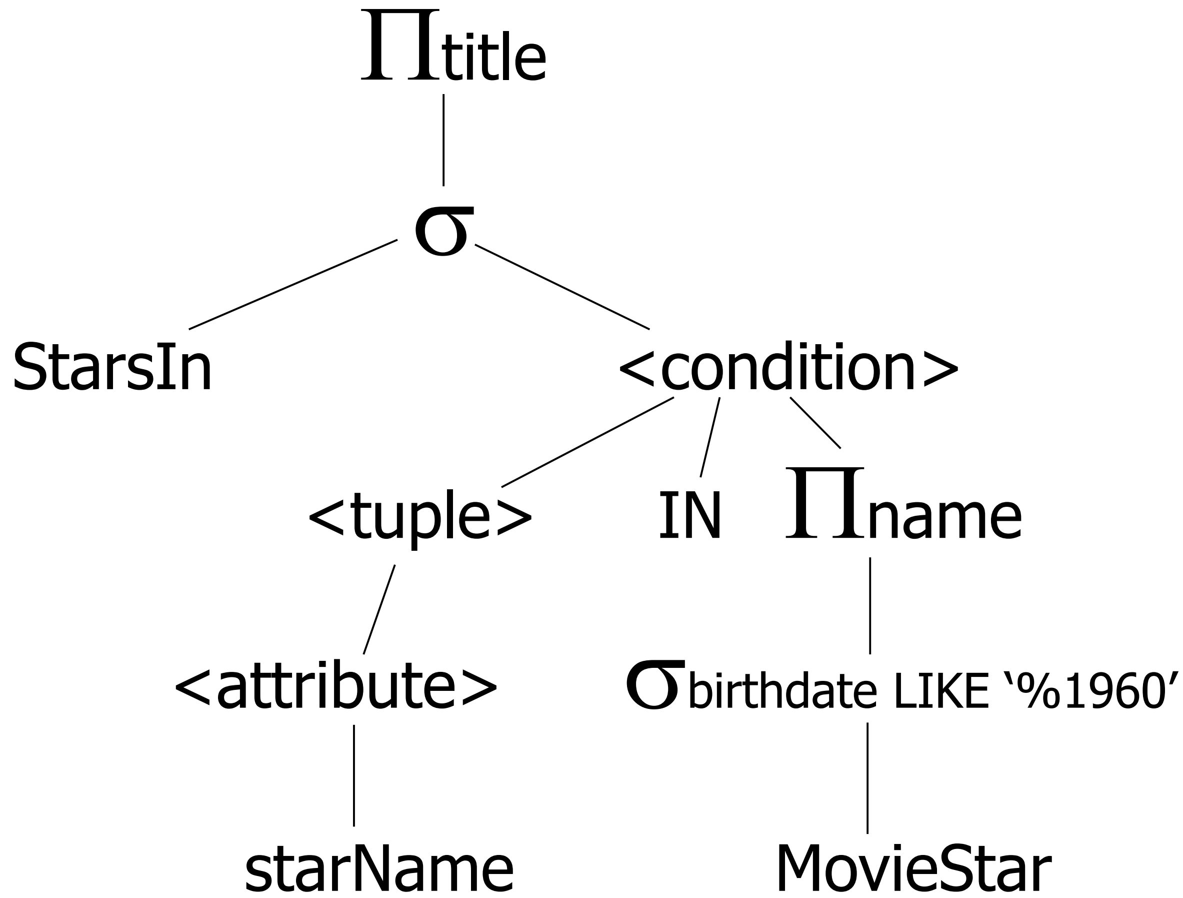


Fig. 7.15: An expression using a two-argument σ, midway between a parse tree and relational algebra

Example: Logical Query Plan

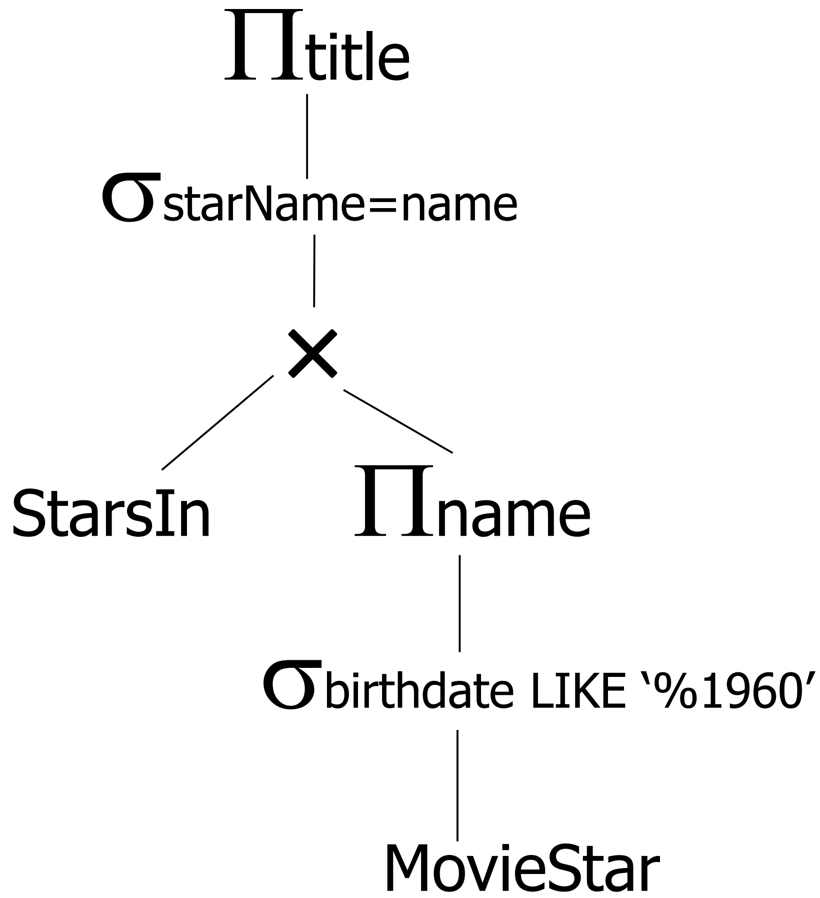


Fig. 7.18: Applying the rule for IN conditions

Example: Improved Logical Query Plan

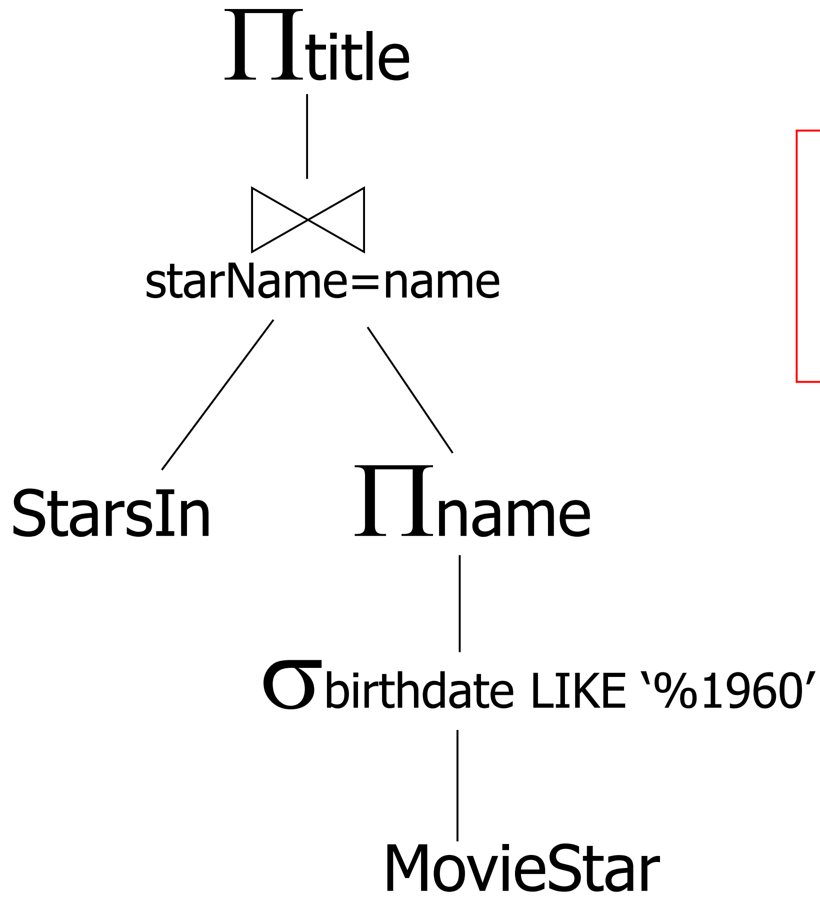
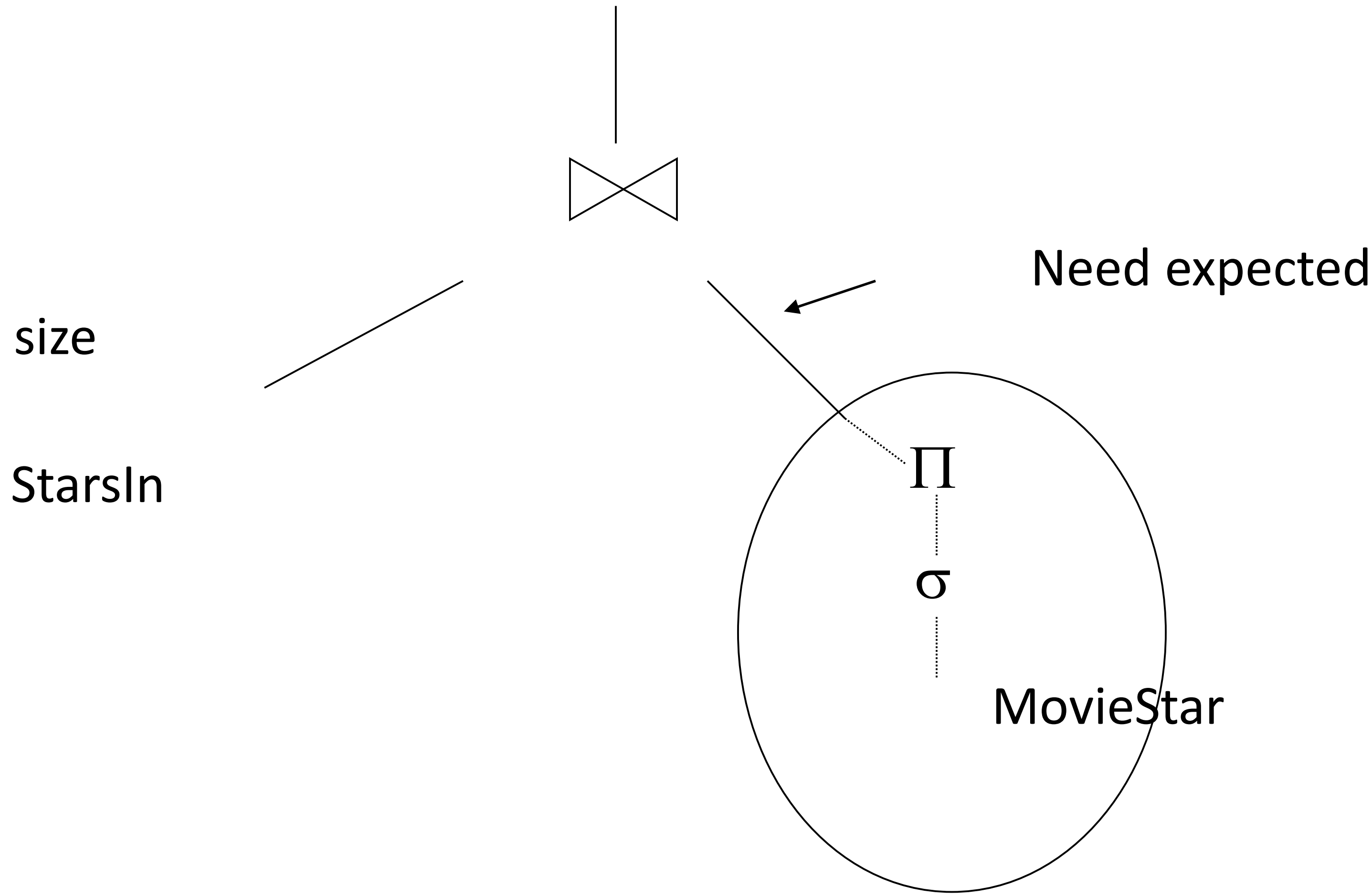


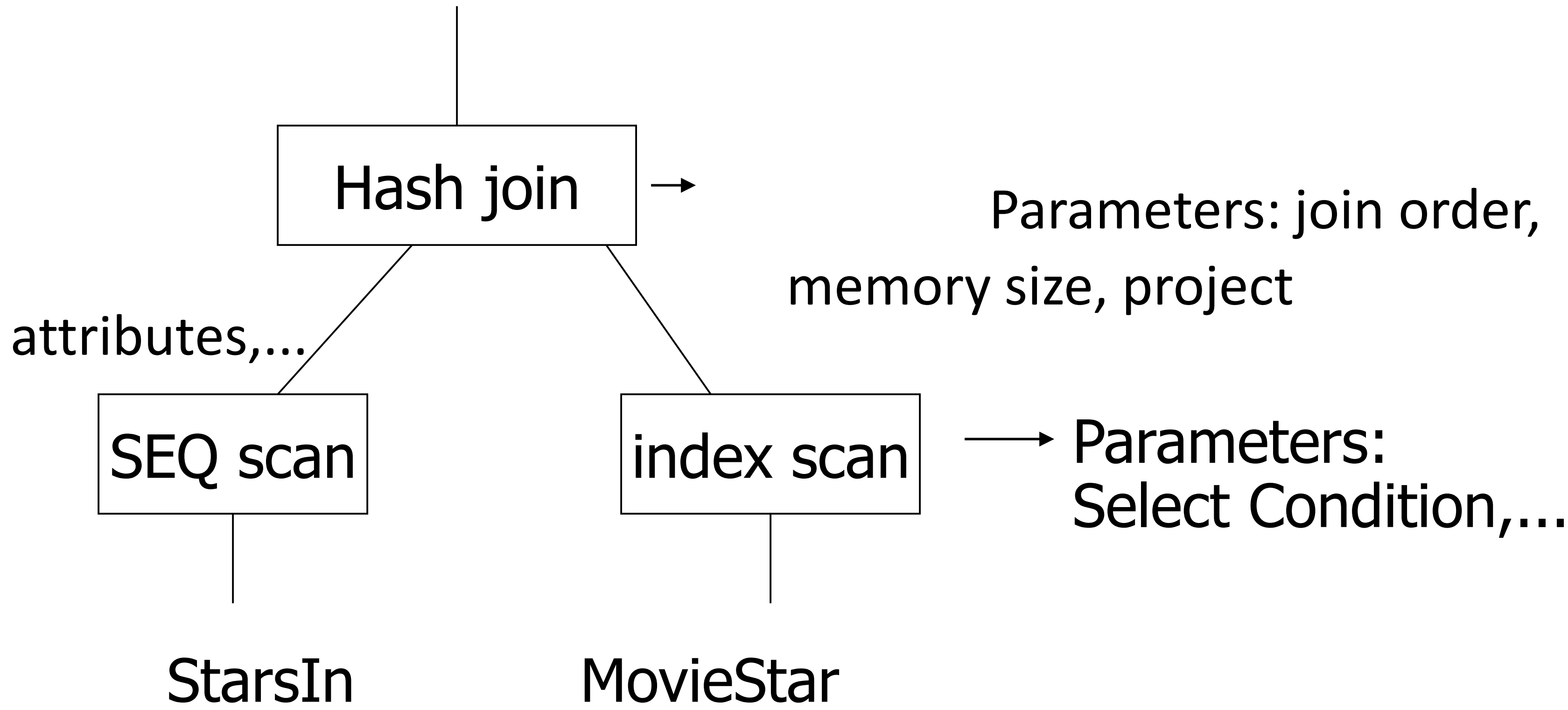
Fig. 7.20: An improvement on fig. 7.18.

Question:
Push project to
StarsIn?

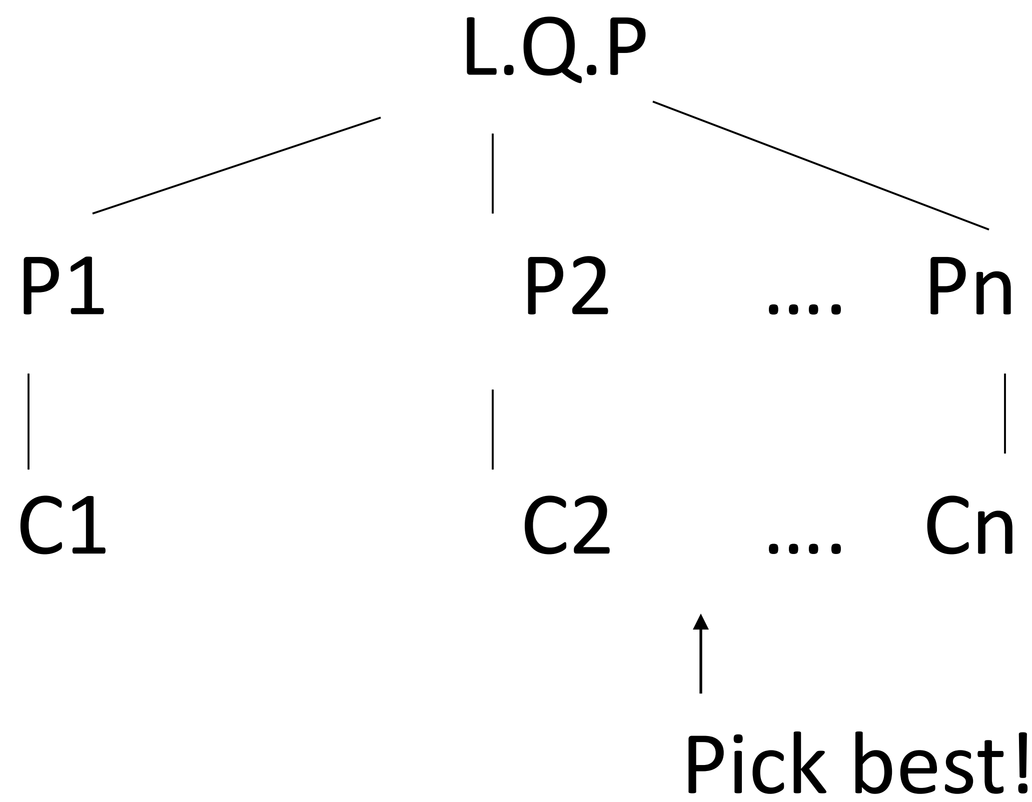
Example: Estimate Result Sizes



Example: One Physical Plan



Example: Estimate costs



查询变换/查询改写

- $\Pi_{name,type,ino}(\sigma_{amount>10000 \wedge country = 'Sweden'}(Customer \bowtie Invoice))$
- $\Pi_{name,type,ino}(\sigma_{amount>10000}(Invoice) \bowtie \sigma_{country = 'Sweden'}(Customer))$
- $\Pi_{name,type,ino}(\sigma_{amount>10000}(Invoice \bowtie \sigma_{country = 'Sweden'}(Customer)))$
- $\Pi_{name,type,ino}(\sigma_{country = 'Sweden'}(\sigma_{amount>10000}(Invoice) \bowtie Customer))$

估算查询的I/O代价

- $\Pi_{name,type,ino,amount}(\sigma_{amount>10000 \wedge country = 'Sweden'}(Customer \bowtie Invoice))$
- $Customer \bowtie Invoice$的代价跟Customer和Invoice的大小有关系。
- $\sigma_{amount>10000 \wedge country = 'Sweden'}(Customer \bowtie Invoice)$ 的代价跟 $Customer \bowtie Invoice$ 的大小有关系。
- $Customer \bowtie Invoice$的大小跟cno的值在Customer和Invoice上的分布有关系。

- **关键：估算中间结果的大小。**

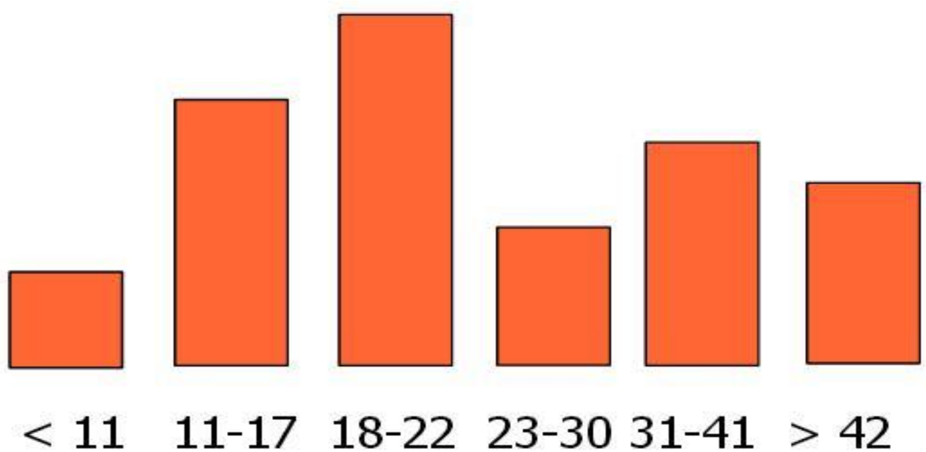
估算连接的中间结果

- $T(R \bowtie S) = T(R)T(S)/\max(\underbrace{V(R,Y)}_{\text{Cardinality}}, V(S,Y))$

- Cardinality Estimation: 基数估计

估算选择的中间结果

- 简单估算：在R中抽取样本R′。计算$\sigma_c(R')$的结果A。那么$\sigma_c(R)$的结果数量为|A|x|R|/|R′|。
- 更精确的估算：直方图（Histogram）

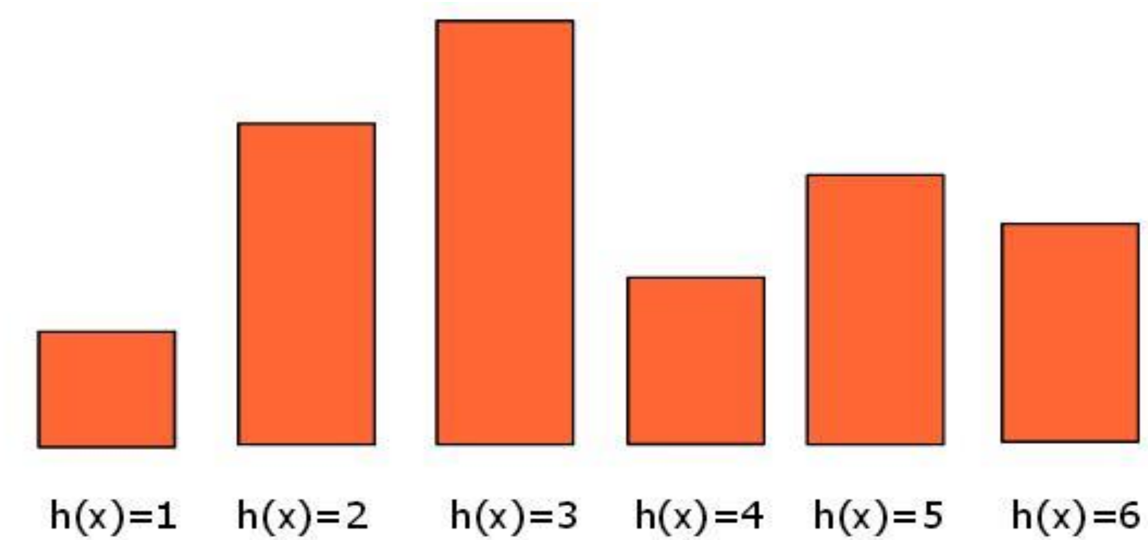


估算连接的中间结果

- 在R和S中抽取样本R′和S′，计算$R' \bowtie S'$的结果A。那么$R \bowtie S$的结果数量为$|A| \times |R| \times |S| / |R'| \times |S'|$。

- 将哈希函数H(x)使用到R和S的链接属性上，选择出哈希值相同的元组R′和S′，计算$R' \bowtie S'$的结果 A。那么$R \bowtie S$的结果数量为$|A| \times |R| / |R'|$。

哈希直方图

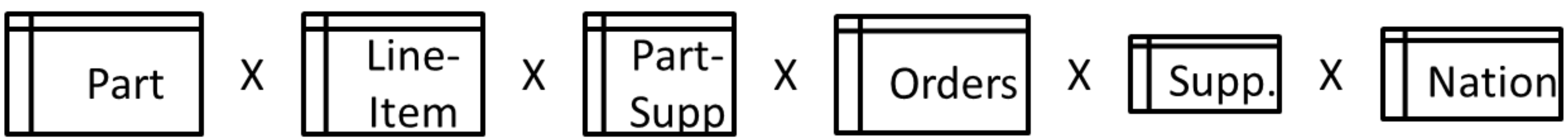


选择执行方式（Physical）

- Sequential Scan vs Index Scan ?
- Join: Nest Loop vs Sort Merge vs Hash vs Index ?

连接顺序问题

- How many different join orders exist in total for n tables?



连接顺序问题

- How many different join orders exist in total for n tables?

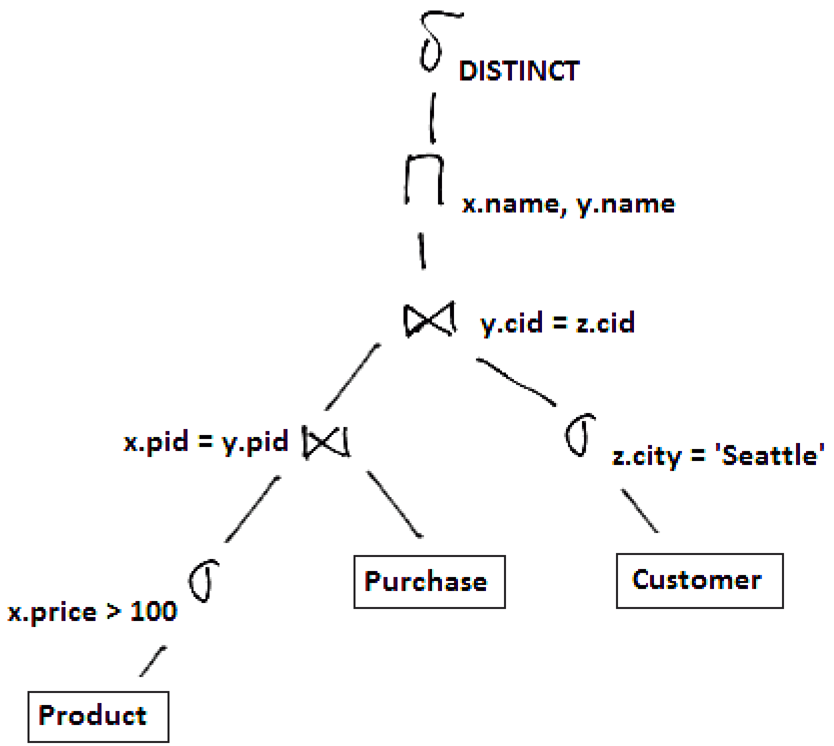
Part X Line-Item X Part-Supp X Orders X Supp. X Nation

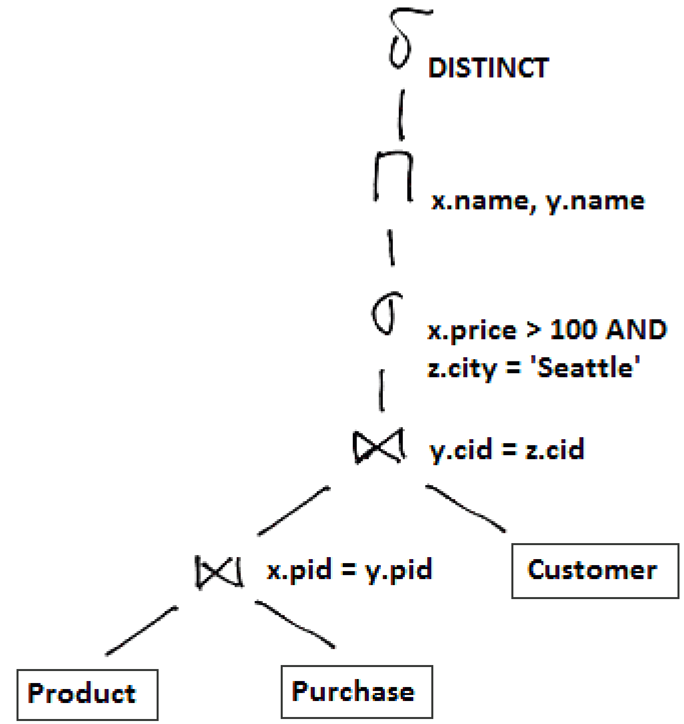
- Idea:
  - Join orders can be expressed as binary trees.
    - ➔ Each possible tree is a possible join order.
  - Number of possible binary trees with n leaves: Catalan number C_n.
    - ➔ $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\cdot n!}$
  - We have C_{n-1} possible trees **for each possible permutation of tables.**
    - ➔ There are n! possible permutations.

- ➔ In total: $n! \cdot \frac{(2\cdot(n-1))!}{(n-1)!\cdot n!} = \frac{(2n-2)!}{(n-1)!}$ orders.

查询执行流水线
Query Execution Pipeline

查询树


DISTINCT
x.name, y.name
y.cid = z.cid
x.pid = y.pid
z.city = 'Seattle'
x.price > 100
Purchase
Customer
Product


DISTINCT
x.name, y.name
x.price > 100 AND
z.city = 'Seattle'
y.cid = z.cid
x.pid = y.pid
Customer
Product
Purchase

批处理 vs 流水线

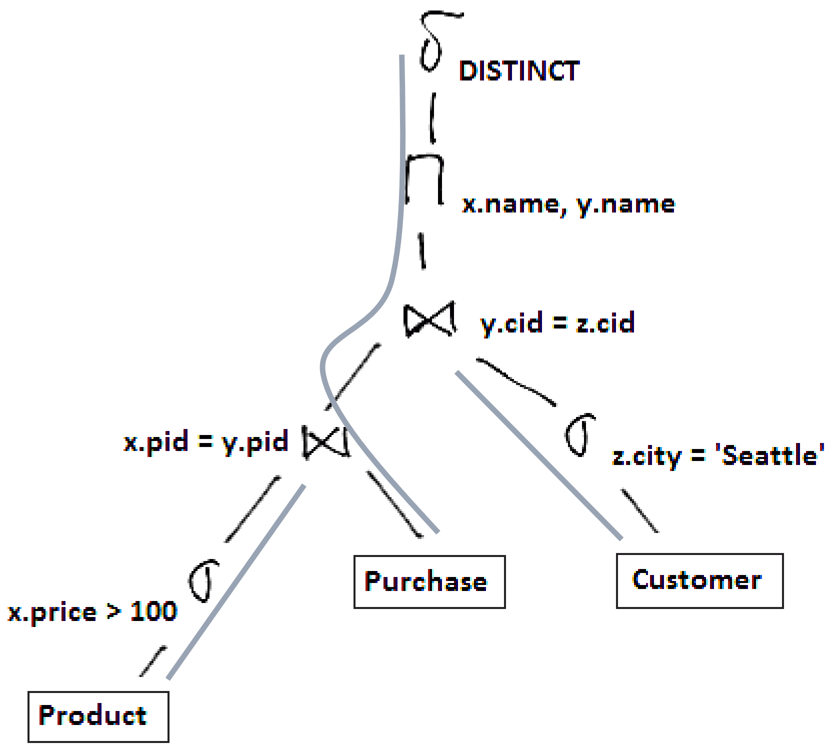
π payment * 2

σ age < 25

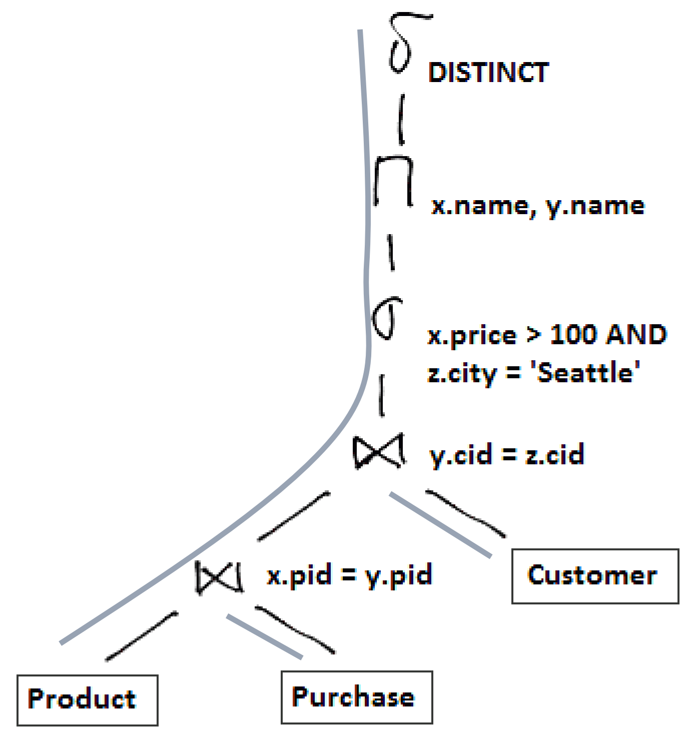
R employees

- 依次执行每个操作：将一个操作在所有数据上执行完之后再执行下一个操作。
- 在逐个数据上执行所有的操作。

查询流水线



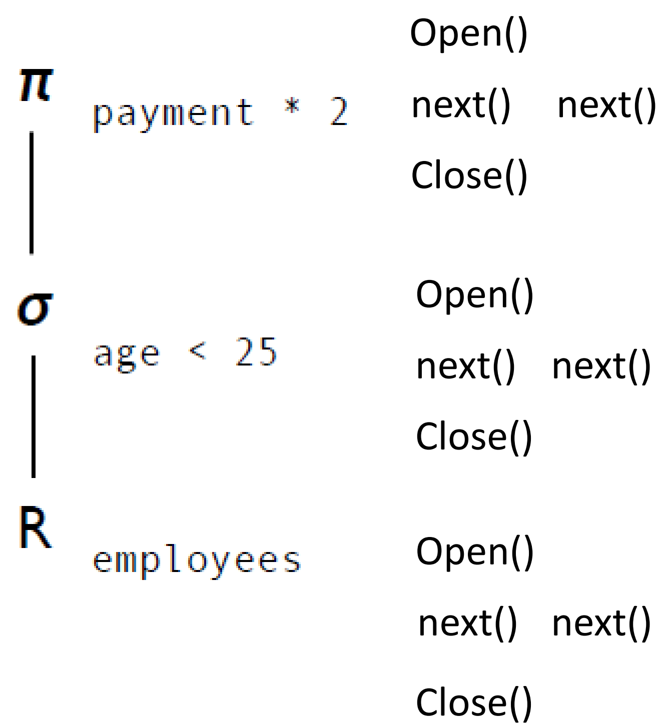
DISTINCT
x.name, y.name
y.cid = z.cid
x.pid = y.pid
z.city = 'Seattle'
x.price > 100
Purchase
Customer
Product



DISTINCT
x.name, y.name
x.price > 100 AND
z.city = 'Seattle'
y.cid = z.cid
x.pid = y.pid
Customer
Product
Purchase

Volcano Model

- 每个数据库操作都使用共同的接口：
  - Open()：准备取第一条数据。
  - Next()：取下一条数据。
  - Close()：完成操作。
- 自上而下执行整个流水线



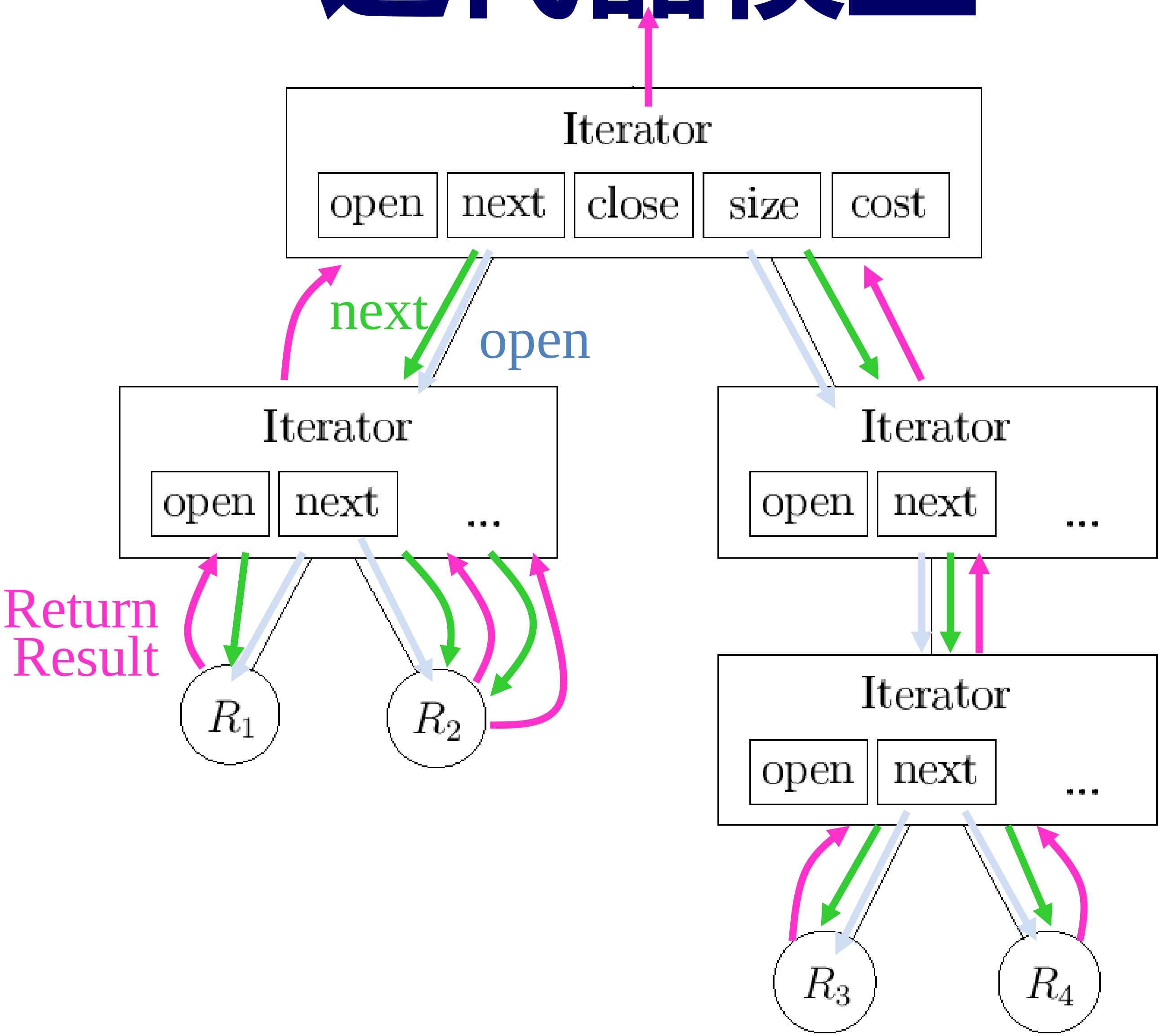
Volcano Model on Join

```
join.open() {
  lhs.open();
  l = lhs.next();
  rhs.open();
}
```

```
join.close() {
  lhs.close();
  rhs.close();
}
```

```
join.next() {
  do {
    if (l == EOF) return EOF;
    r = rhs.next();
    if (r == EOF) {
      l = lhs.next();
      rhs.close();
      rhs.open();
      continue;
    }
  }
  while (¬ Θ(l,r));
  return <l,r>;
}
```

迭代器模型

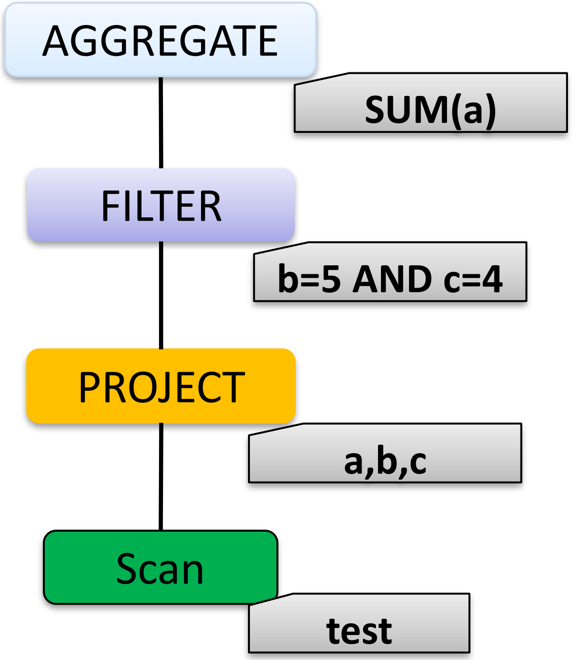

Iterator
open
next
close
size
cost
next
open
Iterator
open
next
...
Iterator
open
next
...
Return
Result
Iterator
open
next
...
R_1
R_2
R_3
R_4

查询执行模型

- **数据库SQL查询**

  SELECT sum(a) FROM test WHERE b=5 AND c>4;

- **查询计划:**

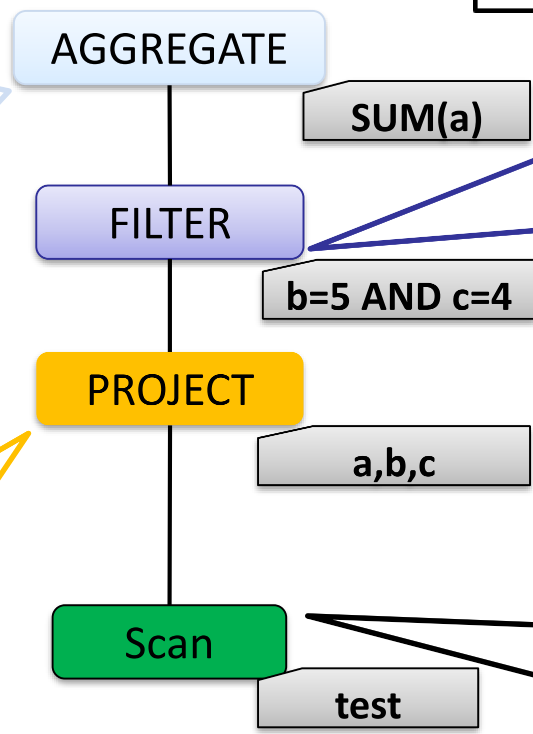


The iterator model

- tuple-at-a-time model
  - 自顶向下函数调用
  - 子算子向父算子返回**一条**元组

```
Op root = new Sum(
  new Filter(
    new Project(
      new Scan("Test"),
      new Colmap("a","b","c")
    ),
    new Predicate("b=5 AND c=4")
  ),
  "a"
);
```

```
class Sum : public Op {
 Tuple next(void) {
  T res = 0;
  while (in = child.next()) {
    res += in->get(column);
  }
  return new Tuple(res);
 }
};
```




```
class Filter : public Op {
 Tuple next(void) {
  while (in=child.next()) {
    if (pred(in)) return in;
  }
  return NULL;
 }
};
```

```
class Project : public Op {
 Tuple next(void) {
   T in = child.next();
   if (in)
     return in->proj(colmap);
   else return NULL;
 }
};
```

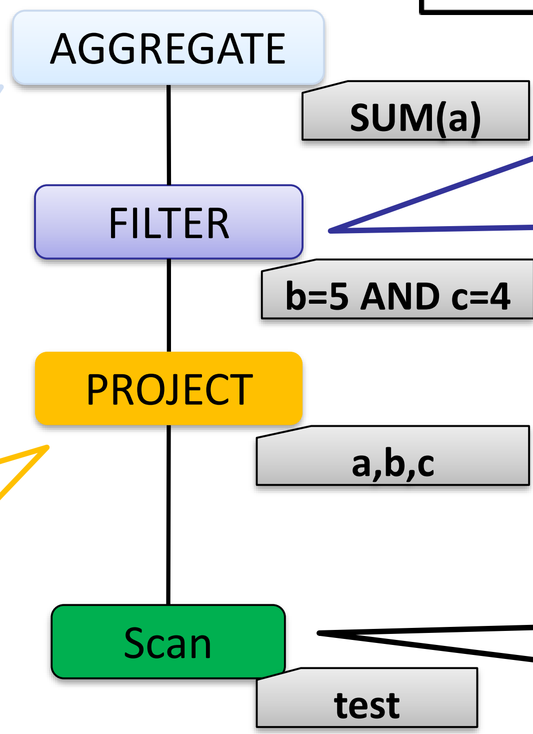
```
class Scan : public Op {
 Tuple next(void) {
   // Return next tuple from
   // specified table.
 }
};
```

The bulk-processing model

- operator/table-at-a-time model
  - 自下而上推送数据
  - 子算子向父算子返回**所有**元组

```
Table x_1, x_2, x_3, x_4;
x_1 = scan("test");
x_2 = project(x_1,
    new Colmap("a","b","c"));
x_3 = filter(x_2,
    new Predicate("b=5 & c=4"));
x_4 = sum(x_3);
```

```
Table sum(Table in, ..) {
 T res = 0;
 for (Tuple t : in) {
  res += t->get(column);
 }
 Table out = new Table(..);
 return out.insert(res);
}
```




```
Table filter(Table in, ..) {
 Table out = new Table(..);
 for (Tuple t : in) {
   if (pred(t))
    out.insert(t);
 }
 return out;
}
```

```
Table project(Table in, ..) {
 Table out = new Table(..);
 for (Tuple t : in) {
  out.insert(
   t->project(colmap));
 }
 return out;
}
```

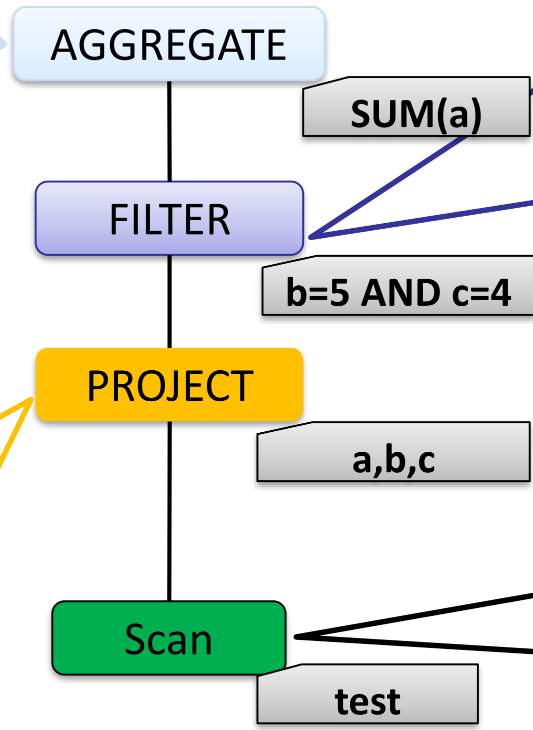
```
Table scan(string name) {
 return new Table(name);
}
```

Vectorized model

- vector-at-a-time model
  - 自顶向下函数调用
  - 子算子向父算子返回一组元组

```
Op root = new Sum(
  new Filter(
    new Project(
      new Scan("Test"),
      new Colmap("a","b","c")
    ),
    new Predicate("b=5 AND c=4")
  ),
  "a"
);
```

```
class Sum : public Op {
 Tuple next(void) {
  T res = 0;
  while (in = child.next()) {
    res += in->get(column);
    for (Tuple t : in) {
      res += t->get(column);
    }
  }
  return new Tuple(res);
 }
};
```

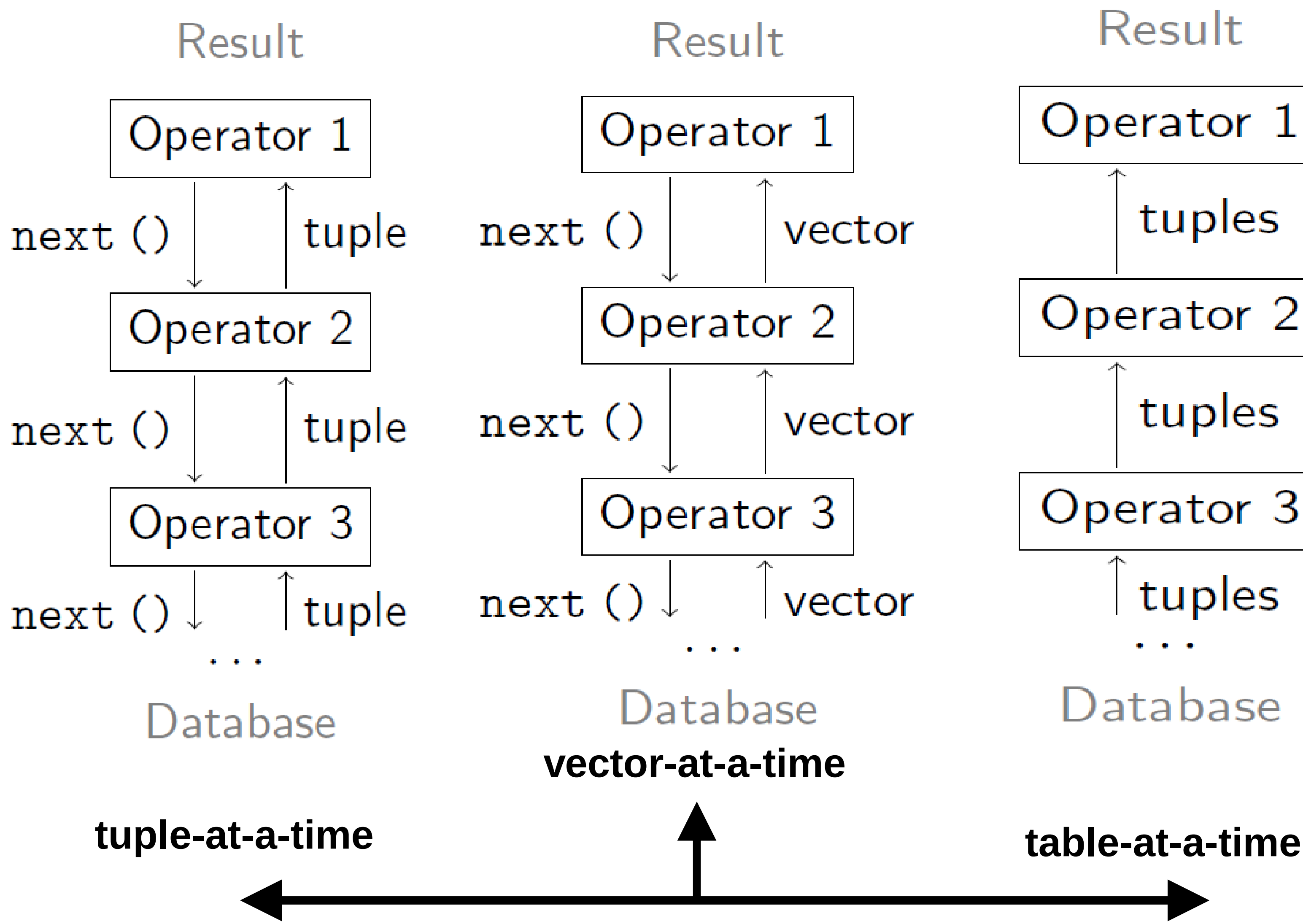



```
class Filter : public Op {
 Vector next(void) {
   Vector out = new Vector(..);
   while (in=child.next()) {
     for (Tuple t : in) {
       if (pred(t))
       out.insert(t);
     }
     return out
   }
   return NULL;
 }
};
```

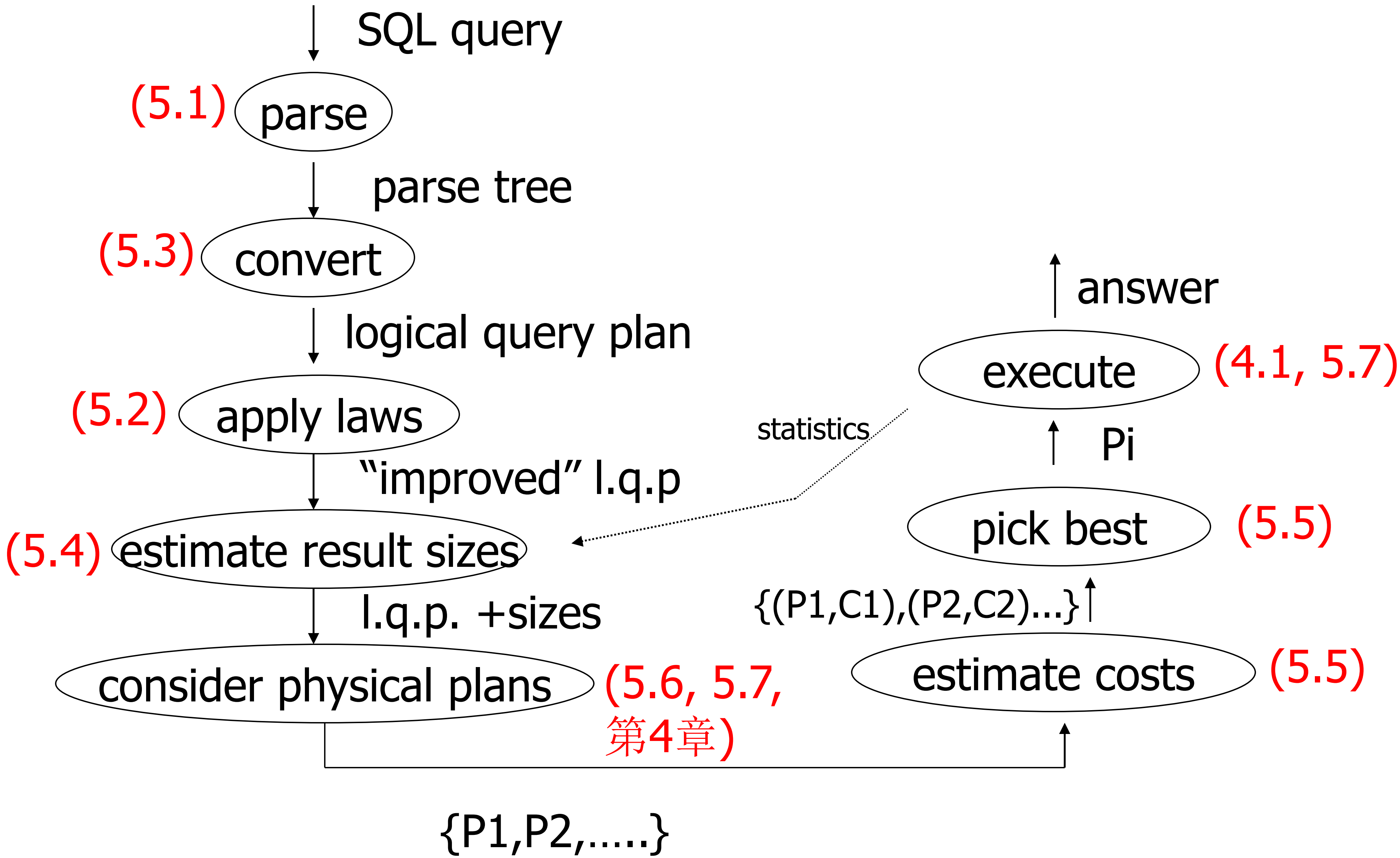
```
class Project : public Op {
 Vector next(void) {
   Vector in = child.next();
   Vector out = new Vector(..);
   for (Tuple t : in) {
     out.insert(
     t->project(colmap));
   }
   return out;
 }
};
```

```
class Scan : public Op {
 Vector next(void) {
   // Return next a few tuples
   // from specified table.
 }
};
```

三种物理实现方式


Result
Operator 1
next ()
tuple
Operator 2
next ()
tuple
Operator 3
next ()
tuple
...
Database
Result
Operator 1
next ()
vector
Operator 2
next ()
vector
Operator 3
next ()
vector
...
Database
Result
Operator 1
tuples
Operator 2
tuples
Operator 3
tuples
...
Database
vector-at-a-time
tuple-at-a-time
table-at-a-time

思考题：列存数据库中的查询计划如何执行？


SQL query
(5.1)
parse
parse tree
(5.3)
convert
logical query plan
(5.2)
apply laws
“improved” l.q.p
(5.4)
estimate result sizes
l.q.p. +sizes
consider physical plans
(5.6, 5.7,
第4章)
{P1,P2,…..}
statistics
answer
execute
(4.1, 5.7)
Pi
pick best
(5.5)
{(P1,C1),(P2,C2)...}
estimate costs
(5.5)

Credits

- Volker Markl
- Xuan Zhou