# *Otimizador de Consultas*

## Banco de Dados II

Cap. 16 – Complete Book

Cap. 14 – Barquinho

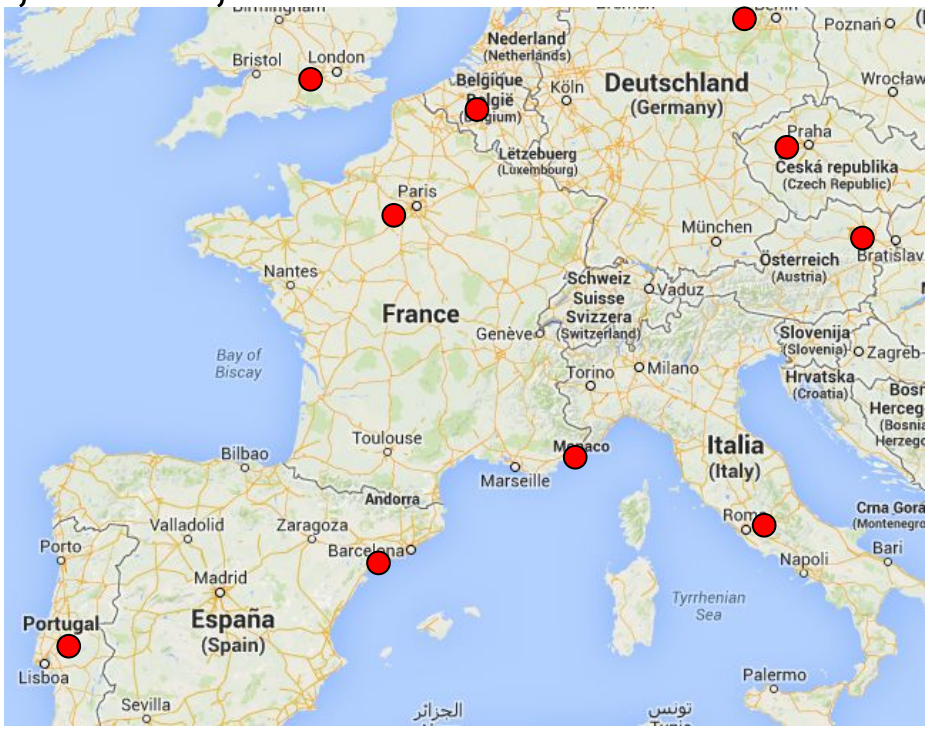
Cap. 12 – Vaquinha

Cap. 19 - Elmasri

Denio Duarte

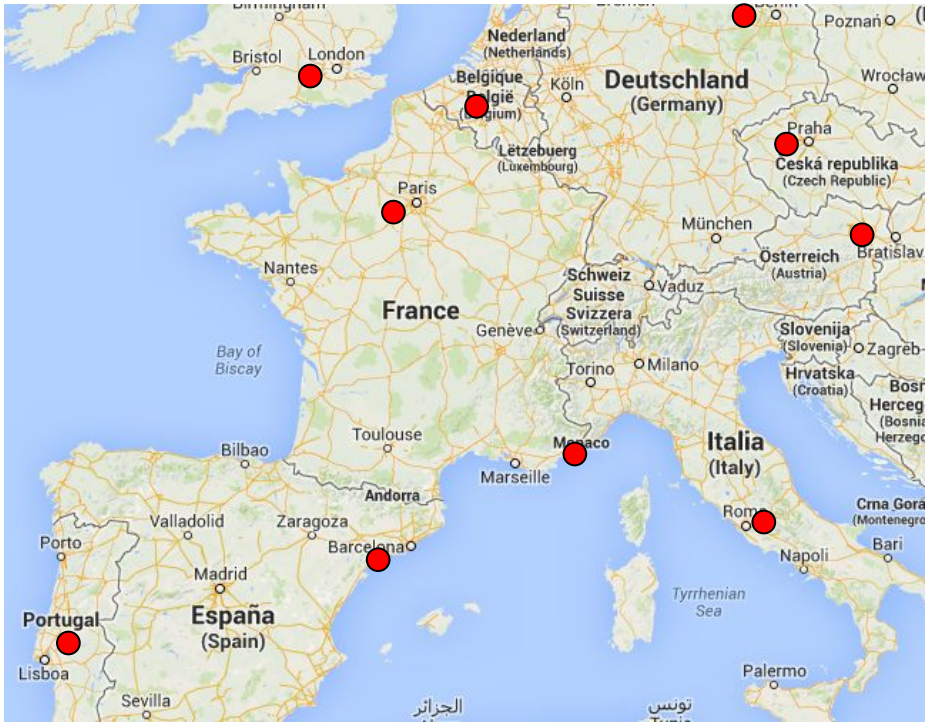
# *Introdução*

- Imagine-se na Europa com 10 cidades para visitar.
  - Paris, Barcelona, Berlim, Praga, Lisboa, Londres, Bruxelas, Viena, Roma e Nice.



# *Introdução*

- Imagine-se na Europa com 10 cidades para visitar.



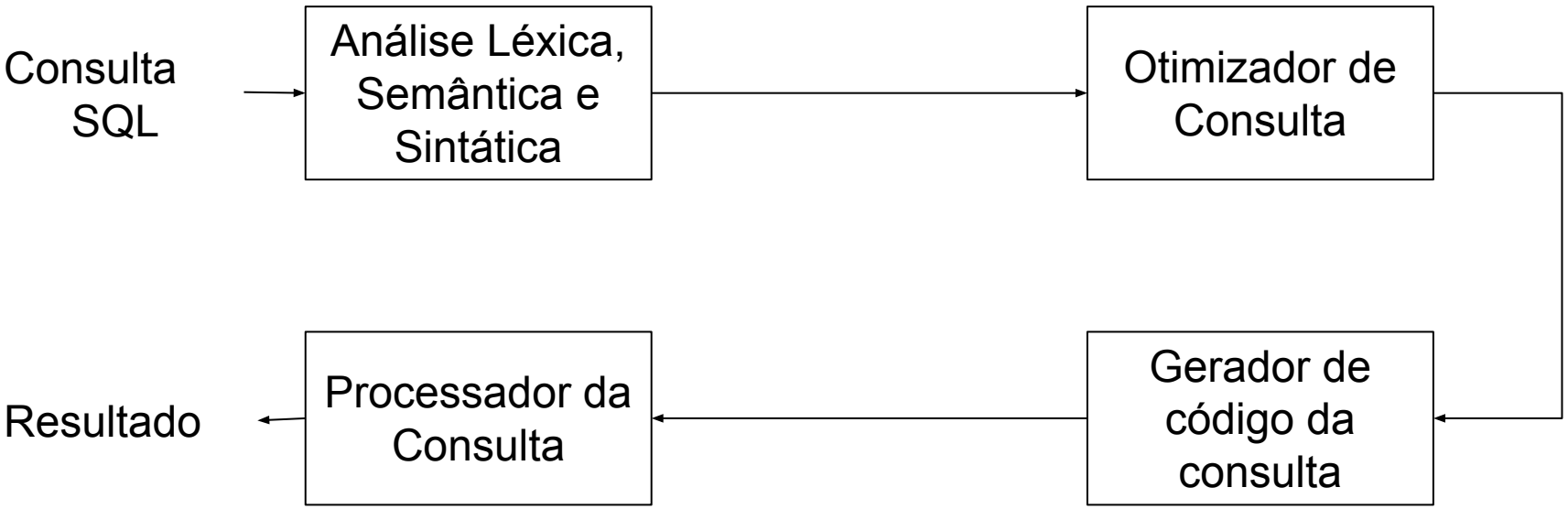

- Qual seria o melhor Roteiro?
- Otimizar
  - Tempo
  - Custo

# *Introdução*

- O trabalho do otimizador de consulta
  - Dada uma consulta
    - Encontrar a melhor forma de dar a resposta
    - Melhor forma = menor tempo
  - Problema:
    - A otimização exata de uma consulta é um problema intratável da computação – O(n!) - NP hard
    - Algoritmos devem se basear em heurísticas
    - Estratégias para otimizar o tempo de resposta de uma consulta

# *Introdução*

- Os módulos responsáveis pelo processamento de consulta




O processador utiliza ainda: gerenciador de índice/arquivo/registro, gerenciador de buffer e gerenciador de armazenamento

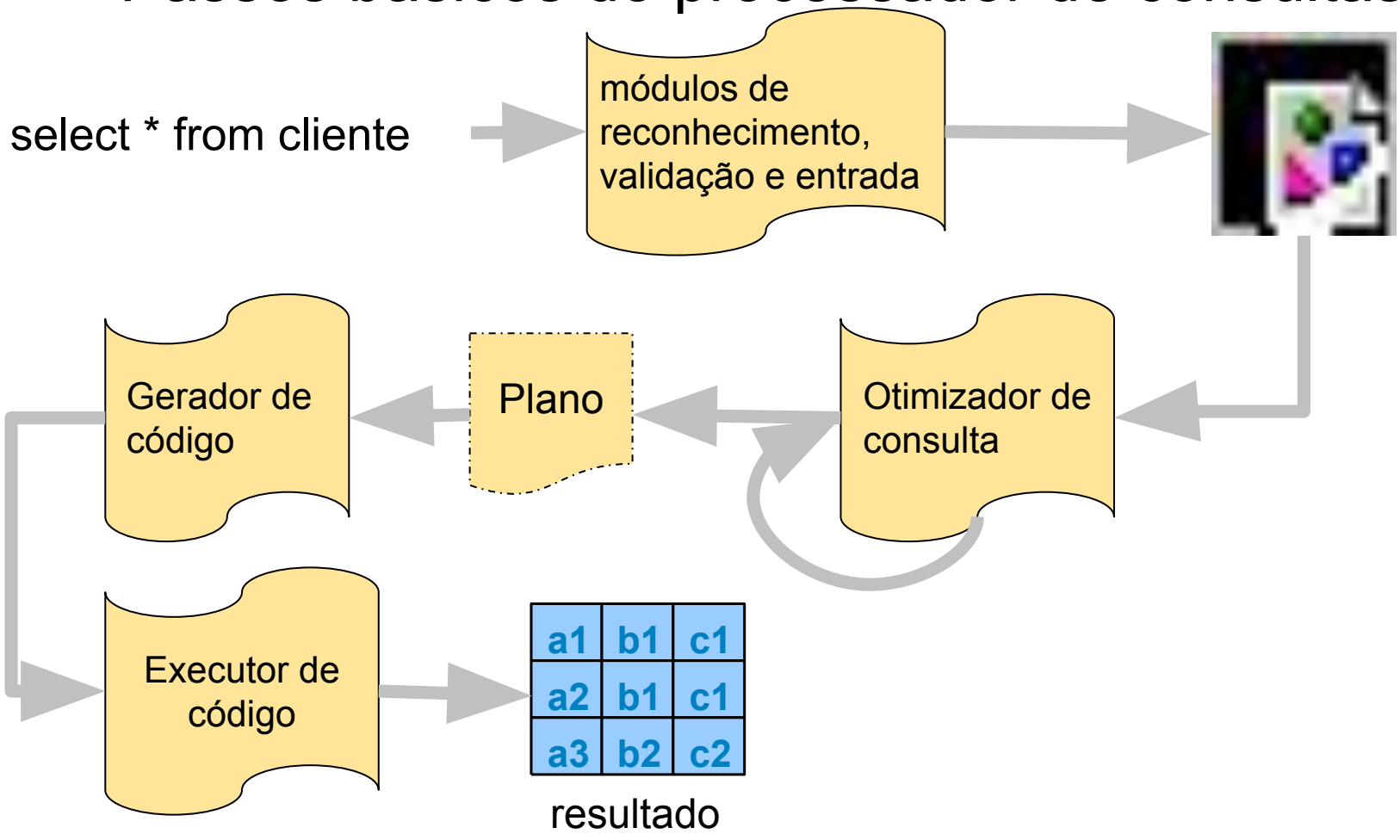
# *Processador de Consultas*

# *Processador de Consulta*

- Algumas operações no banco de dados são caras
- Armas para melhorar o desempenho
  - Implementações inteligentes para operadores explorando as equivalências entre operadores relacionais
  - Utilização de estatística e modelos de custos para escolher as melhores equivalências entre os operadores relacionais

# *Processador de Consulta*

- Passos básicos do processador de consultas



# *Processador de Consulta*

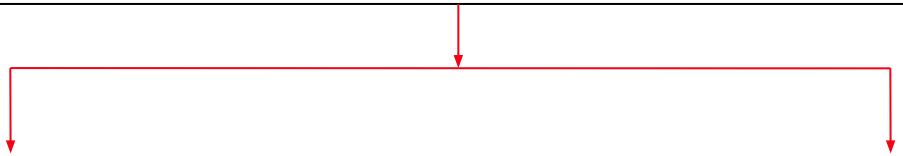
- Transformar consultas SQL em uma estrutura similar à Álgebra Relacional
  - Consulta é transformada em álgebra relacional
  - Consultas são decompostas em blocos, formando unidades básicas que podem ser transformadas em operadores algébricos e otimizadas
  - Blocos de consultas contém expressões simples SELECT-FROM-WHERE, cláusulas GROUP-BY e HAVING
  - Consultas aninhadas são identificadas como blocos separados

# *Processador de Consulta*

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > (SELECT    MAX (SALARY)
                    FROM      EMPLOYEE
                    WHERE     DNO = 5);
```

# *Processador de Consulta*

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > (SELECT    MAX (SALARY)
                    FROM      EMPLOYEE
                    WHERE     DNO = 5);
```



```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > C
```

```
SELECT    MAX (SALARY)
FROM      EMPLOYEE
WHERE     DNO = 5
```

# Processador de Consulta

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > (SELECT   MAX (SALARY)
                   FROM     EMPLOYEE
                   WHERE    DNO = 5);
```

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > C
```

```
SELECT   MAX (SALARY)
FROM     EMPLOYEE
WHERE    DNO = 5
```

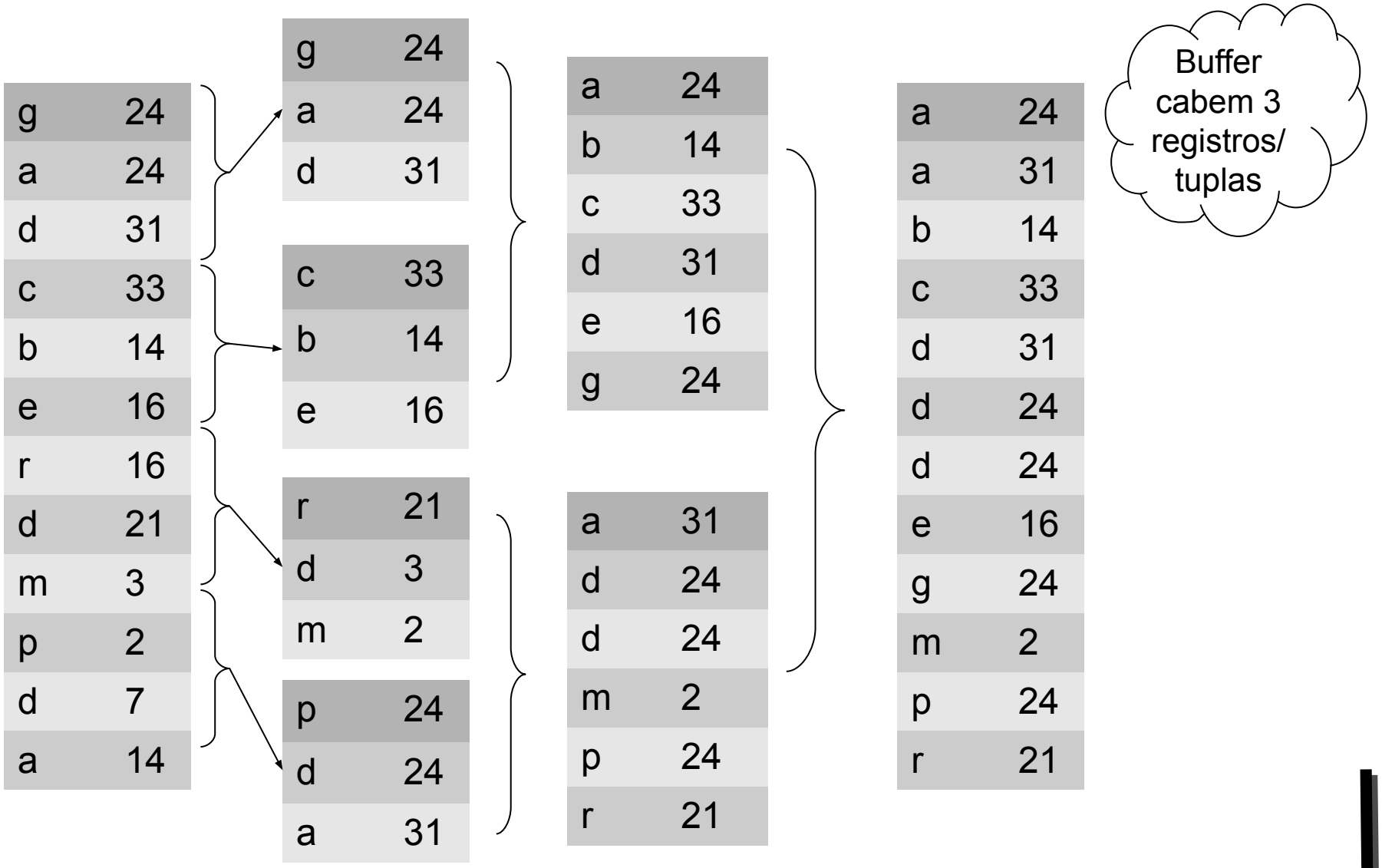
$\pi_{LNAME,\ FNAME}\ (\sigma_{SALARY>C}(EMPLOYEE))$

$\mathcal{G}_{Max(SALARY)}\ (\sigma_{DNO=5}\ (EMPLOYEE))$

# *Processador de Consulta*

- Ordenação Externa
  - Método de ordenação utilizado com os dados a serem ordenados não cabem na memória principal
  - SGBD utilizam para implementar:
    - ORDER BY
    - JOIN, UNION, INTERSECTION
    - DISTINCT
  - Geralmente utiliza a estratégia sort-merge
    - Divide-se arquivo em pequenos pedações que caibam na memória (runs)
    - Fusione os arquivos até obter um só arquivo ordenado

# *Processador de Consulta*



# *Processador de Consulta*

- Ordenação Externa
  - Os SGBD realizam ordenação externa constantemente, assim os algoritmos são eficientes
  - ORDER BY e DISTINCT são “pesados” para o processador de consultas

# *Processador de Consulta*

- Algoritmos para executar operações relacionais
  - O SGBD deve ter um ou mais algoritmos que implementa cada operador (Select, Join) e, alguns casos, combinação dos mesmos
  - Cada algoritmo pode ser aplicado a casos particulares (dependendo se existe índice ou não, entre outros)
  - O otimizador pode considerar, no processo de otimização, apenas algoritmos que se aplicam para o caso em particular

# *Processador de Consulta*

- Algoritmos para seleção
  - Várias operações para seleções simples
    - S1: Busca linear (força bruta): recupera linha a linha testando o predicado da seleção (abordagem cara)
      - Custo #p/2 no caso médio ou #p, caso contrário
        - #p número de páginas ocupadas pelas tuplas
    - S2: Busca em árvore: utilizada se arquivo ordenado pelo atributo da seleção
      - $\sigma_{cpf=5463}(Empregado)$ - CPF atributo chave
      - Custo $\log_k n$ - **k** aridade da B+ (**n** é #p)
    - S3: Utilizando índice primário (ou hash) (igualdade)
      - Recupera no máximo 1 linha/registro
      - Custo: mesmo de S2 (hash: 1 página – sem colisão)

# *Processador de Consulta*

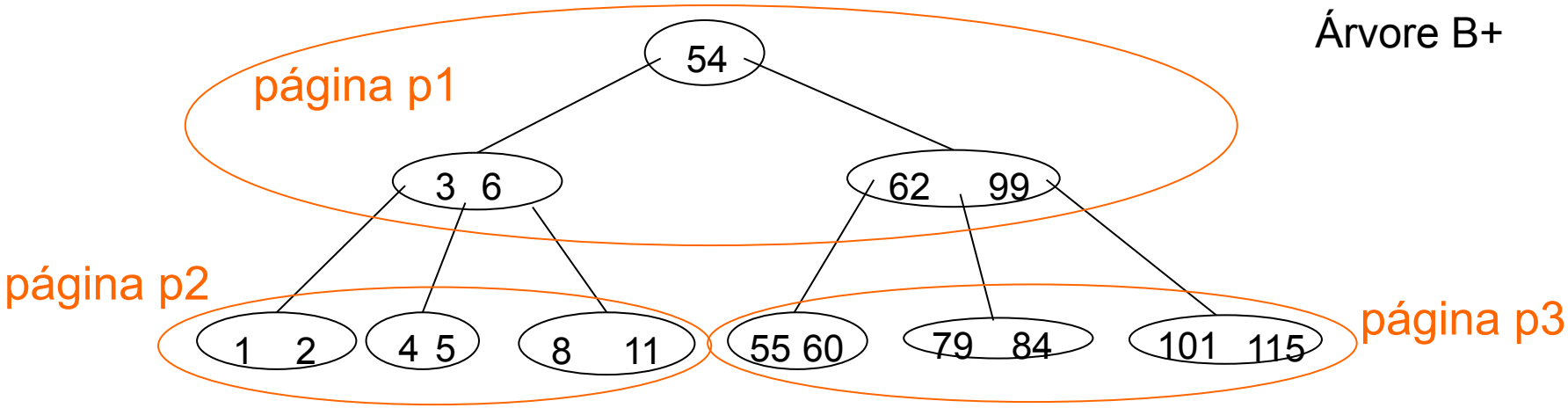
- S4: Utilizando índice primário para recuperar múltiplas linhas (<,>,<= ou >=)
  - $\sigma_{ndepto > 10}(Depto)$
  - Utiliza o índice para encontrar o departamento 10, depois recupera os departamentos seguintes
  - Se menor, recupera os departamentos anteriores
  - Utilizado para consultas por faixa (com operador **and**)

# *Processador de Consulta*

- S5: Utilizando índices clusterizados
  - Comparação de igualdade em um atributo não chave com índice clusterizado
  - $\sigma_{ndepto=5}(Empregado)$
  - Utiliza o índice para encontrar todos as linhas que satisfaçam o predicado
  - Se menor, recupera os departamentos anteriores
  - Utilizado para consultas por faixa (com operador **and**)

# *Processador de Consulta*

- S6: Utilizando índices B+ em igualdades
  - Recupera 1 tupla se for chave ou várias caso contrário
  - Pode ser utilizado para recuperar faixa de valores
  - Custo: altura da árvore (se couber em uma página= 1), mais uma página para igualdade, para faixa o número de blocos que compõem a faixa de valores



# *Processador de Consulta*

- Consultas utilizando predicados compostos, ligados por **and** (predicados conjuntivos)
- S7: consulta conjuntiva utilizando um índice individual
  - Utiliza primeiro a condição do atributo com índice (métodos S2 à S6) para recuperar um grupo de linhas
  - Verifica quais linhas recuperadas satisfazem o restante das condições

# *Processador de Consulta*

- S8: consulta conjuntiva utilizando um índice composto
  - Se as condições conjuntivas do predicado envolvem índices compostos (ou hashing), torna-se igual à S3
  - Exemplo, tabela *DeptoLotado* com índice (ndepto,cpf), e consulta $\sigma_{ndepto=5 \wedge cpf=53456}(DeptoLotado)$

# *Processador de Consulta*

- S9: seleção conjuntiva envolvendo intersecção de linhas
  - Se índices secundários estão disponíveis para as condições no predicado, cada índice recupera uma coleção de linhas
  - A intersecção das linhas é o resultado da consulta
  - Se existirem condições em índices, cada linha recuperada é testada com as condições remanescentes

# *Processador de Consulta*

- Algoritmos para implementar junção (Join)
  - Dois tipos
    - Two-way join (junção dupla)
    - Multiway join (junção múltipla)
  - Exemplo utilizado $R \bowtie_{A=B} S$
  - J1: junção de laço-aninhado (nested-loop join) – força bruta
    - Para cada tupla ***t*** em ***R*** (laço externo), recupera todas as linhas ***t'*** em ***S*** testando se ***t[A]=t'[B]***
    - Esquema de implementação

```
for r in R
   for s in S
      if r.A=s.B then show (r,s)
```

# *Processador de Consulta*

- Algoritmos para implementar junção (Join)
  - $R \bowtie_{A=B} S$
  - J2: junção com laço simples (single-loop join)
    - Utiliza alguma estrutura para recuperar tuplas que atendem a junção
    - Se um índice (ou hash) existe para um dos atributos (***B*** de ***S***, por exemplo), recupera uma tupla ***t*** de ***R*** uma vez (laço simples) e para cada ***t[A]*** acessa diretamente a tupla ***t'[S]*** através do índice/hash
    - Esquema de implementação (supondo índice em *S.B*)

```
for r in R
    rid=find r.A in S(B)    // rid physical record id
    if (rid is not null) then
        tS=get(rid,S)
        show (r,tS)
```

# *Processador de Consulta*

- Algoritmos para implementar junção (Join)
  - $R \bowtie_{A=B} S$
  - J3: sort-merge join
    - Se as tuplas de ***R*** e ***S*** estão ordenadas fisicamente por ***A*** e ***B***, a junção pode ser implementada de forma mais eficiente
    - Ambas as tabelas são varridas concorrentemente na ordem dos atributos de junção, combinando as tuplas onde ***t[A]=t'[B]***
    - Se as tabelas não estão ordenadas, são ordenados utilizando a ordenação externa, em seguida as tabelas são copiadas para o buffer e as tuplas varridas de maneira direta, sem looping

# *Processador de Consulta*

- Algoritmos para implementar junção (Join)
  - $R \bowtie_{A=B} S$
  - J4: hash-join
    - O atributo ***B*** *possui índice hashing*.
    - Primeiro, varre ***R*** e para cada tupla **t[A]** de **R** em seguida acessa a tupla **t'[B]** de **S** através do índice hashing em **B**.
    - Esquema de implementação para ambos **J3** e **J4**

```
Saux=merge(S,S.B) // ou Hash(S,S.B)
for r in R
    rid=find R.A in Saux
    if (rid is not null && exists r.A in Saux) then
        tS= get(rid,S)
        show (r,tS)
```

# *Processador de Consulta*

- Algoritmos para implementar junção (Join)
  - SGBD comerciais:
    - Sybase ASE possui  single-loop join e sort-merge join.
    - Oracle possui  page-oriented nested loop join, sort-merge join, e variação de hybrid hash join.
    - IBM DB2 possui single-loop join, sort-merge, e hybrid hash-join.
    - Microsoft SQL Server possui  single-loop join, sort-merge, hash join, e técnicas chamadas equipe de hash teams.
    - PostgreSQL implementa merge-join, hybrid hash-join e bitmap-join.

# *Processador de Consulta*

- Algoritmos para implementar projeção
  - Seleciona apenas os atributos da lista a partir dos registros solicitados
  - Estratégias
    - Remover os atributos não solicitados
    - Eliminar as duplicatas (caso houver DISTINCT)

# *Processador de Consulta*

- Algoritmos para implementar projeção
  - Exemplo:
```
Select DISTINCT nome, cidade
From empregado
```
  - Projeção baseada em ordenamento
    - Varre tabela empregado e produz um conjunto de tuplas que contenha apenas os atributos desejados
    - Ordena o conjunto de tuplas combinando todos os atributos da lista
    - Varra o resultado e descarte as tuplas repetidas

# *Processador de Consulta*

- Algoritmos para implementar projeção
  - Exemplo:
```
Select DISTINCT nome, cidade
From empregado
```
  - Projeção baseada em hashing
    - Hashing é utilizado para eliminar duplicatas
    - Cada tupla é hashed (atributos da lista) e armazenados no buffer
    - Se der colisão e valores repetidos, este não é armazenado

# *Processador de Consulta*

- Algoritmos para implementar projeção
  - Exemplo:
```
Select DISTINCT nome, cidade
From empregado
```
  - Projeção baseada em índice
    - Se existir um índice com todos os atributos, utilize o índice sem acessar a tabela
  - Pode-se utilizar o merge-sort para implementar a projeção

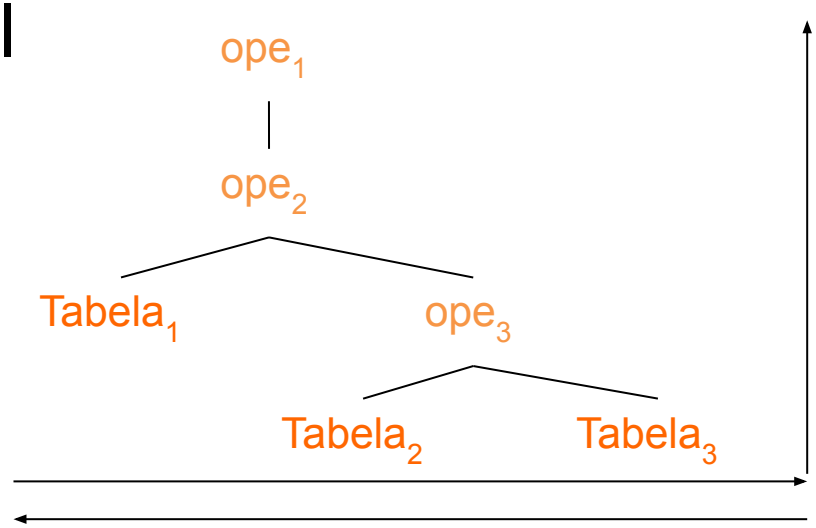
# *Processador de Consulta*

- Operações combinadas
  - Utiliza cada operação de forma independente e combina os resultados (materializa os resultados)
  - Executa as operações independentemente e passa o resultado para o próximo nível (pipeline)

# Planos de Consulta

# *Plano de Consulta*

- Estratégia que o banco utiliza para executar uma consulta
- Normalmente, vários planos são proposto e um deles é escolhido
- O plano é um pseudocódigo em forma de árvore e álgebra relacional



# *Plano de Consulta*

- Cenário
  - Velejador(vid,vname, level, age)
    - Tupla com 50b
    - Uma página armazena 80 tuplas
    - Necessário 500 páginas
  - Reserva(vid, bid, day, desc)
    - Tupla com 40b
    - Uma página armazena 100 tuplas
    - Necessário 1.000 páginas

# *Plano de Consulta*

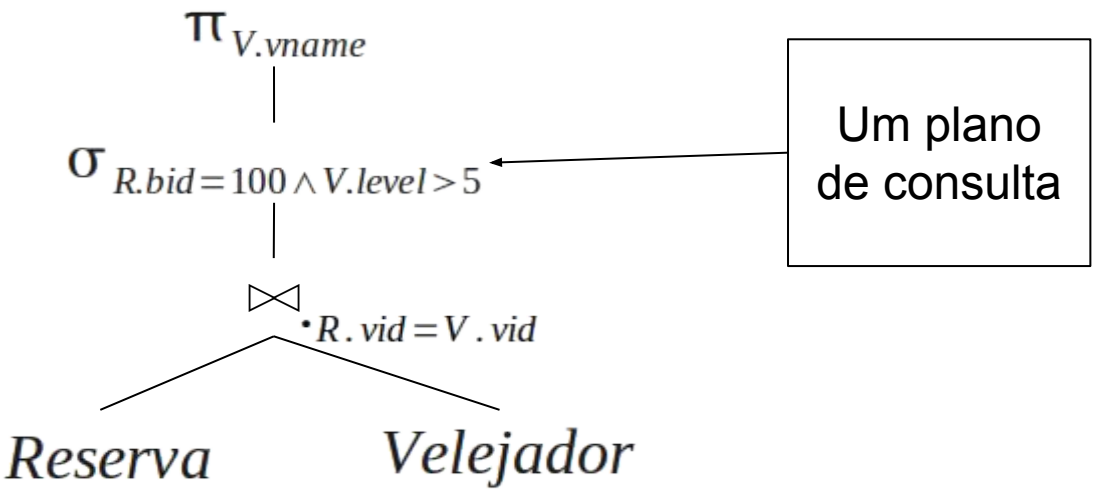
- Cenário

```
Select V.vname,
From Reserva R   natural join Velejador V
Where R.bid=100 and V.level>5
```

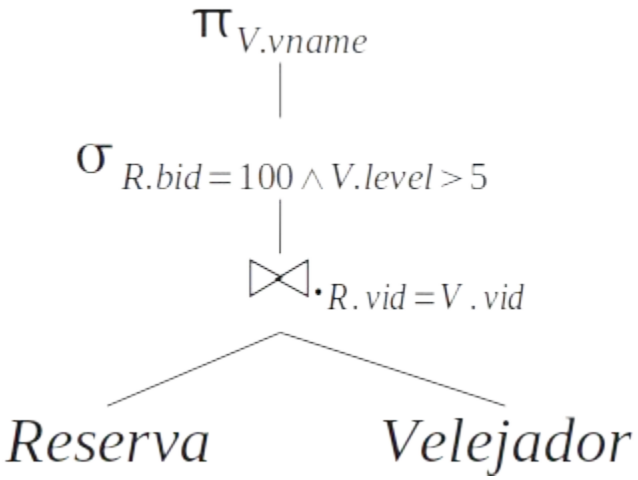
# *Plano de Consulta*

- Cenário

```
Select V.vname,
From Reserva R natural join Velejador V
Where R.bid=100 and V.level>5
```

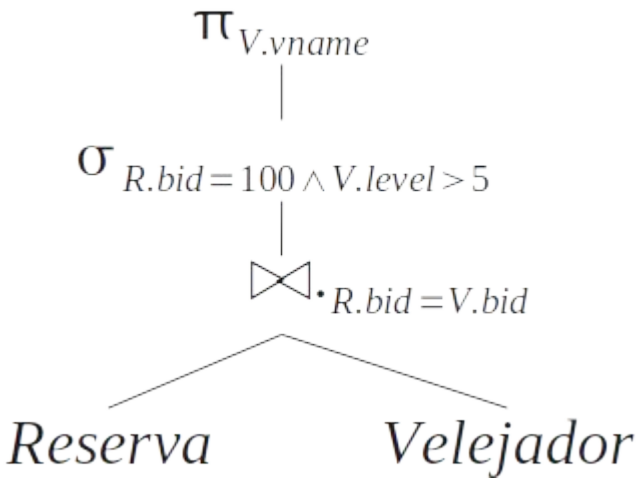


# Plano de Consulta

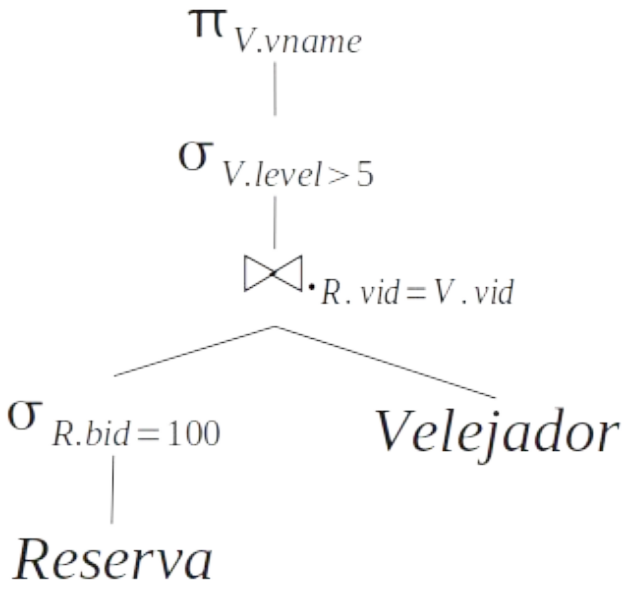



É o melhor?

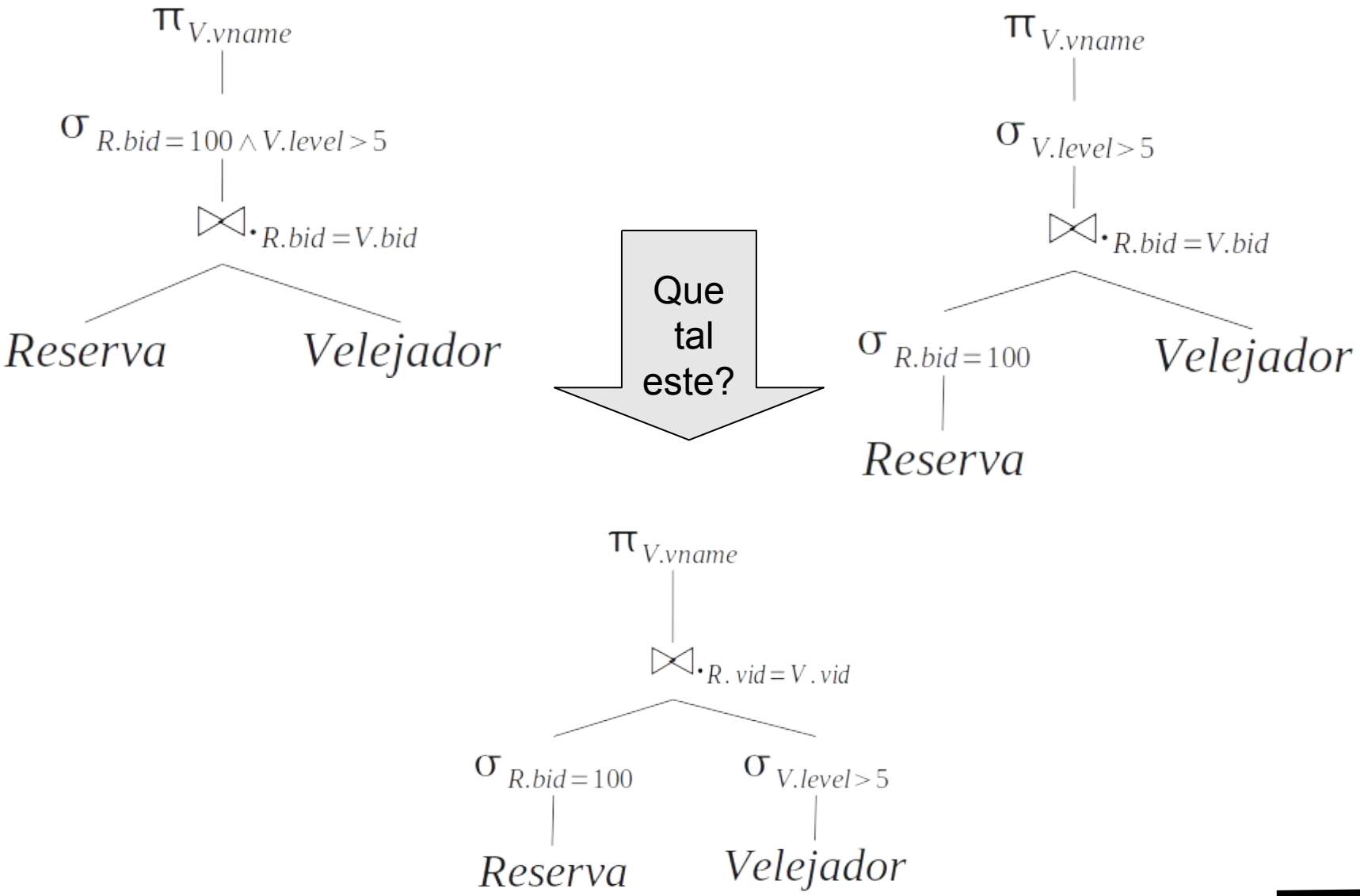
# Plano de Consulta




Ou este?



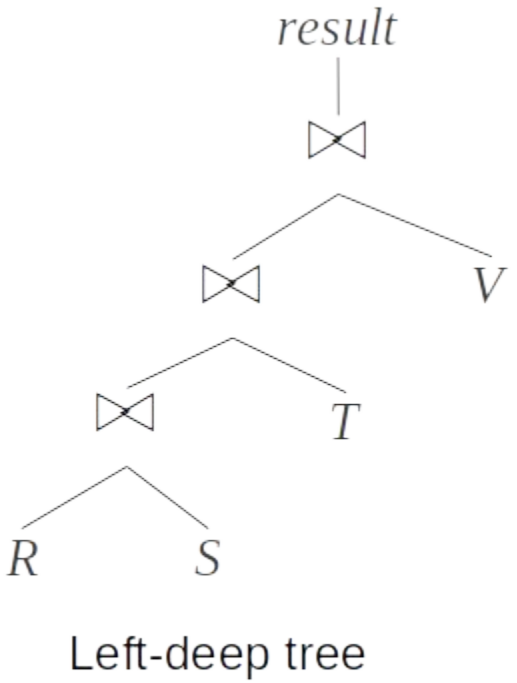
# Plano de Consulta



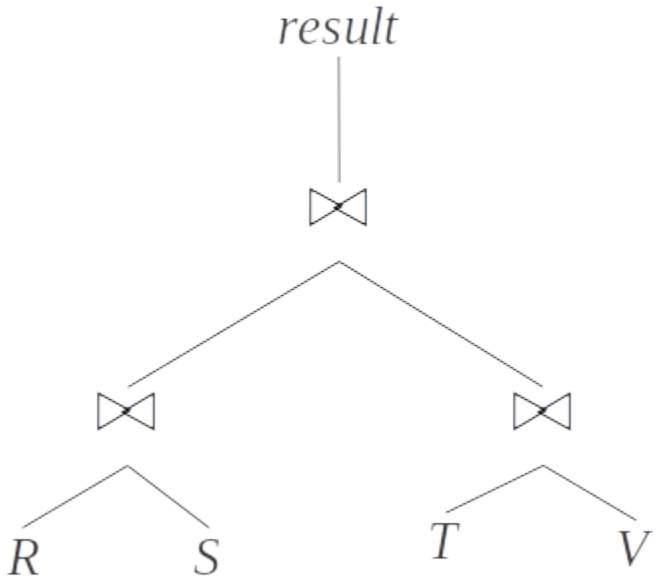
# *Plano de Consulta*

- Dada a consusta:

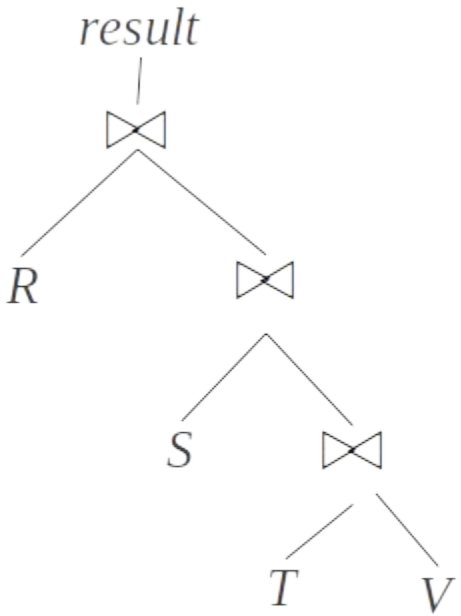
```
select *
from R natural join S
natural join T
natural join V;
```




Left-deep tree




Bushy-deep tree

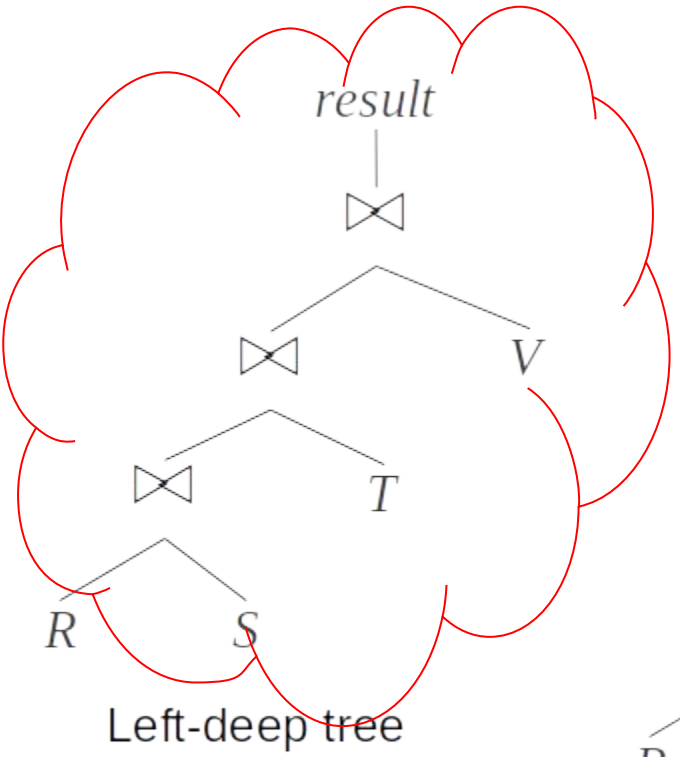



Right-deep tree

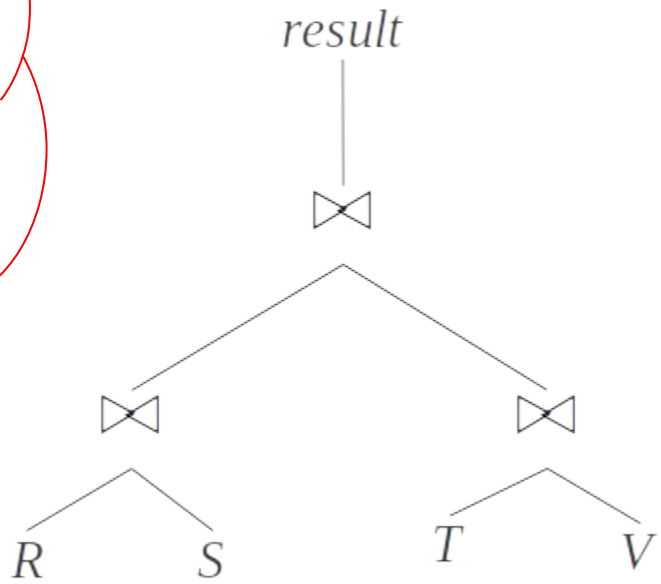
# *Plano de Consulta*

- Dada a consusta:

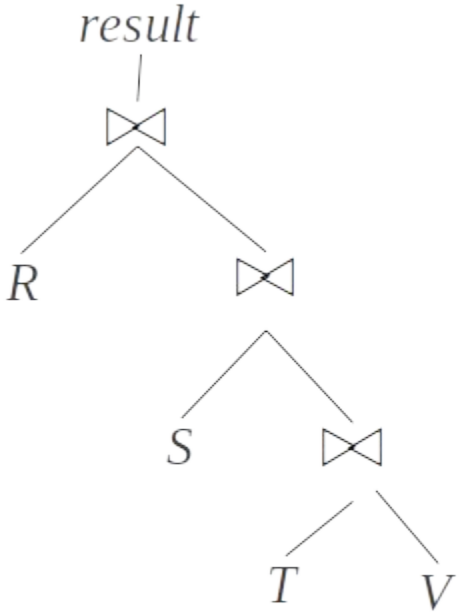
```
select *
from R natural join S
natural join T
natural join V;
```




Left-deep tree



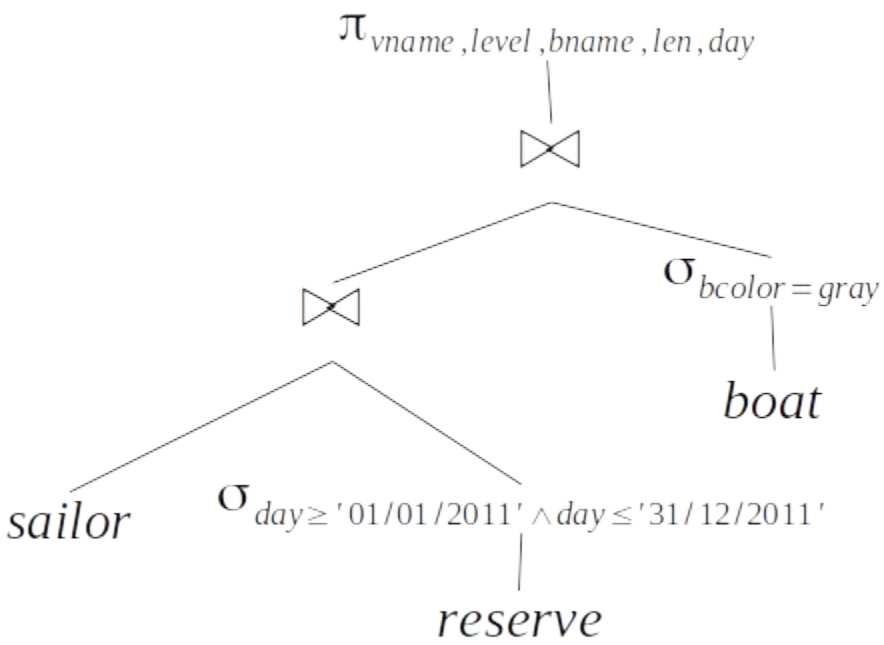

Bushy-deep tree




Right-deep tree

# Plano de Consulta

```
select vname, level, bname, len, day
from sailor natural join reserve
            natural join boat
where day between '01/01/2011' and '31/12/2011'
and   bcolor='gray';
```

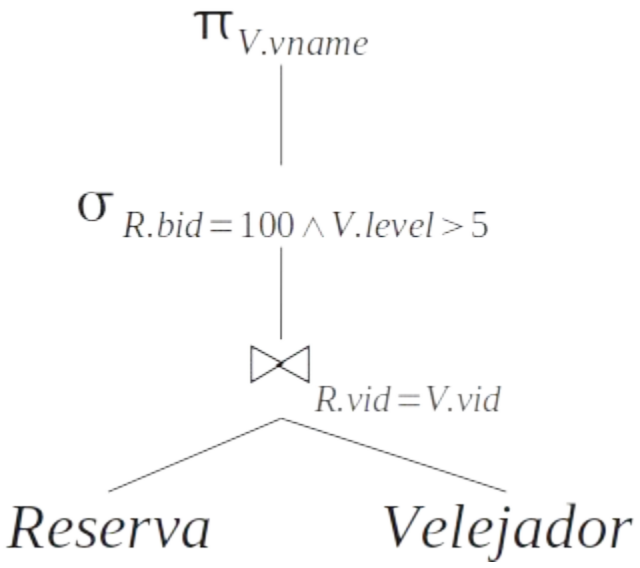


# *Plano de Consulta*

- Estratégias para processar consulta
  - Qual tabela processar primeiro
    - Mais ou menos volumosa
  - Utilizar índice
  - Ordenar tabela
  - Tratamento junção
  - Melhor decomposição
  - Quantos planos propor
  - Como escolher o melhor plano
  - Algoritmo de tratamento dos operadores

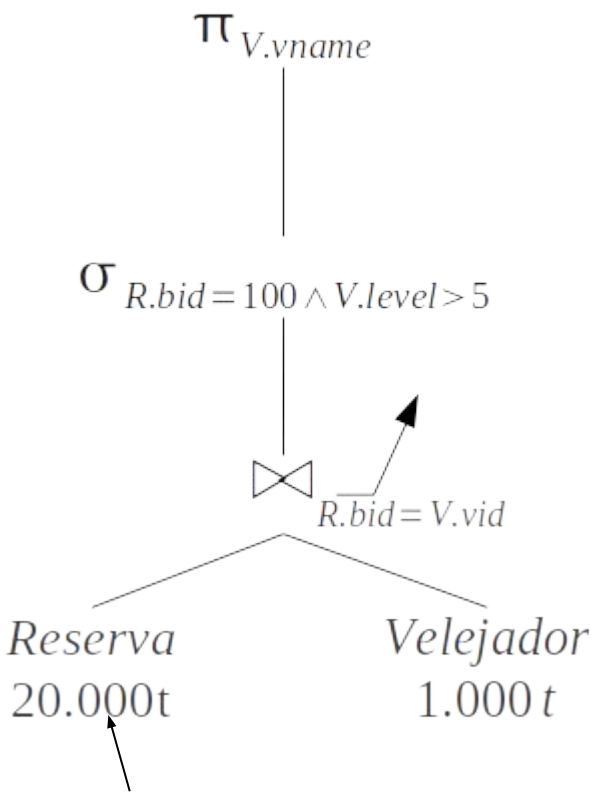
# *Plano de Consulta*

- Exemplo (tabelas com mais tuplas)
  - Reserva: 20.000 tuplas / 1.000 páginas
  - Velejador: 1.000 tuplas / 200 páginas



# *Plano de Consulta*

- Exemplo

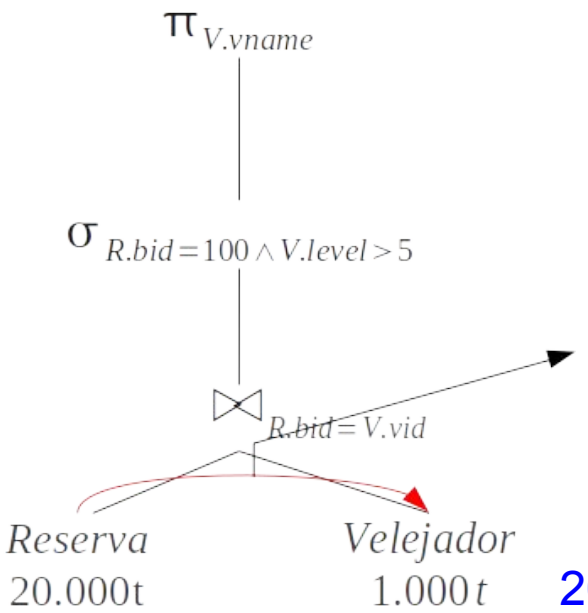



#IO: 1000

varre as tabelas: 1000 I/O

# *Plano de Consulta*

- Exemplo




nested-loop join

20000 x 200 = 4.000.000

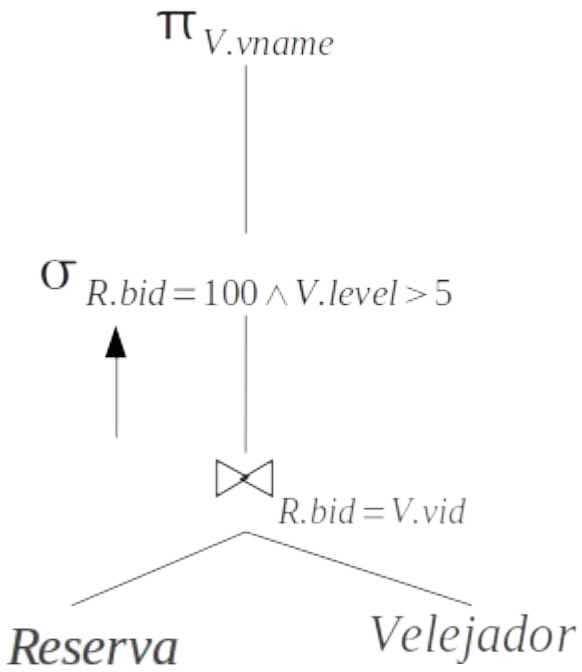
#IO: 1.000
\+ 4.000.000

varre as tabelas: 1 t → percorre 1.000t (pior caso) - 200b

# *Plano de Consulta*

- Exemplo

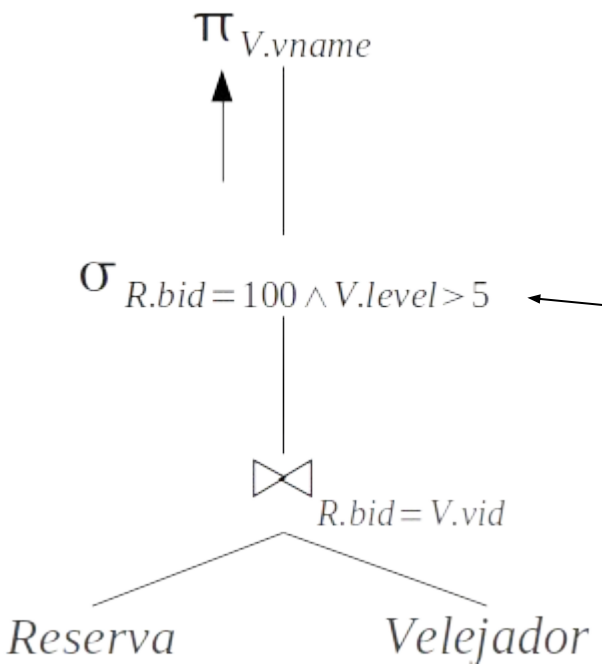
envia resposta
como tabela
temporária **Temp** terá no
pior caso:
20.000t → 1.050 páginas




```
#IO:     1.000
+    4.000.000
+        1.050
```

# *Plano de Consulta*

- Exemplo



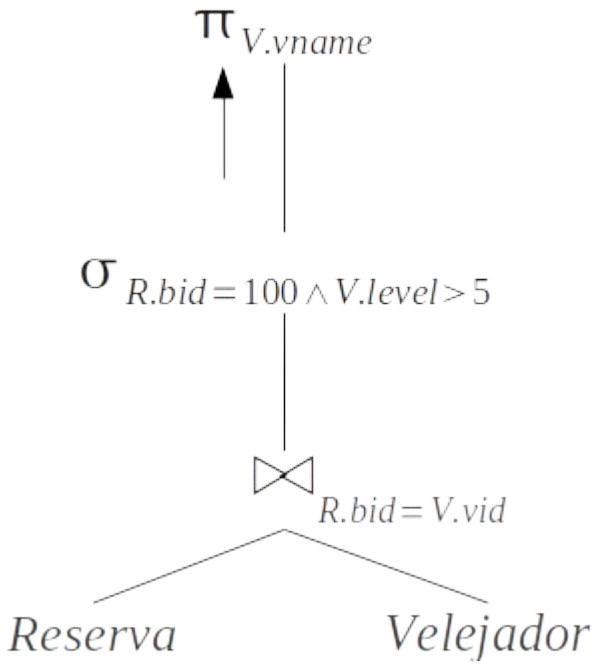

varre selecionado
Tuplas de **`Temp`**
1050 IO

```
#IO:     1.000
+    4.000.000
+        1.050
+        1.050
```

# *Plano de Consulta*

- Exemplo

  envia resposta
  como tabela
  temporária **Temp2**
  100t → 2p




```
#IO:     1.000
+    4.000.000
+        1.050
+        1.050
+            2
T    4.003.102
```

# *Plano de Consulta*

- Estimativa do custo
  - Dicionário de Dados (Catálogo)
    - Várias estatísticas
  - Outros fatores
    - Tamanho da página
    - Algoritmos de gerenciamento de buffer

# *Plano de Consulta*

- Equivalências na álgebra relacional
  - Seleções
    - $\sigma_{c_1 \wedge c_2 \wedge \cdots \wedge c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\cdots(\sigma_{c_n}(R)\cdots)))$
    - $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$
  - Projeção
    - $\pi_{a_1}(R) \equiv \pi_{a_1}(\pi_{a_2}(\cdots(\pi_{a_n}(R)\cdots)))$
  - Produtos cartesianos e junções
    - $R \times S \equiv S \times R$
    - $R \bowtie S \equiv S \bowtie R$
    - $R \times (S \times T) \equiv (R \times S) \times T$
    - $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$

# *Plano de Consulta*

- Equivalências na álgebra relacional
  - Seleções, projeções e junções
    - $\pi_a(\sigma_c(R)) \equiv \sigma_c(\pi_a(R))$
    - $R \bowtie_c S$
    - $\sigma_c(R \times S) \equiv \sigma_c(R) \times S$
    - $\sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S$

# *Plano de Consulta*

- Heurística
  - Algumas regras que diminuem o custo da consulta
    - Quando possível empurra-se as projeções para a parte baixo da árvore
      - Diminui o tamanho da resposta
      - Relação R(a, b, c) com 20.000 tuplas
      - Cada tupla 190 bytes (header = 24 bytes, a = 8 bytes, b = 8 bytes, c = 150 bytes)
      - Página 1024 b sendo header = 24 bytes
      - 1 página = 5 tuplas (5 * 190 = 950 b – sobram 5b)
      - Para 20.000 tuplas: 4.000 página
      - Projeção eliminando o atributo **c**:
        - Tupla 40 b, cabem 25 em um bloco
        - Necessários 800 blocos (fator de redução 5)

# *Plano de Consulta*

- Heurística
  - Algumas regras que diminuem o custo da consulta
    - Quando possível deixa-se as seleções próximas as tabelas que são aplicadas
      - Diminui o número de tuplas para serem processadas mais adiante
    - Aplicar os joins por último (quando possível)

# *Plano de Consulta*

- Exercícios

**Q1**

```
SELECT b.date, s.region, s.location
FROM brweather b natural join state s
WHERE b.temperature > 30
AND b.time > '1500' AND b.conditions = 'Sunny';
```

**Q2**

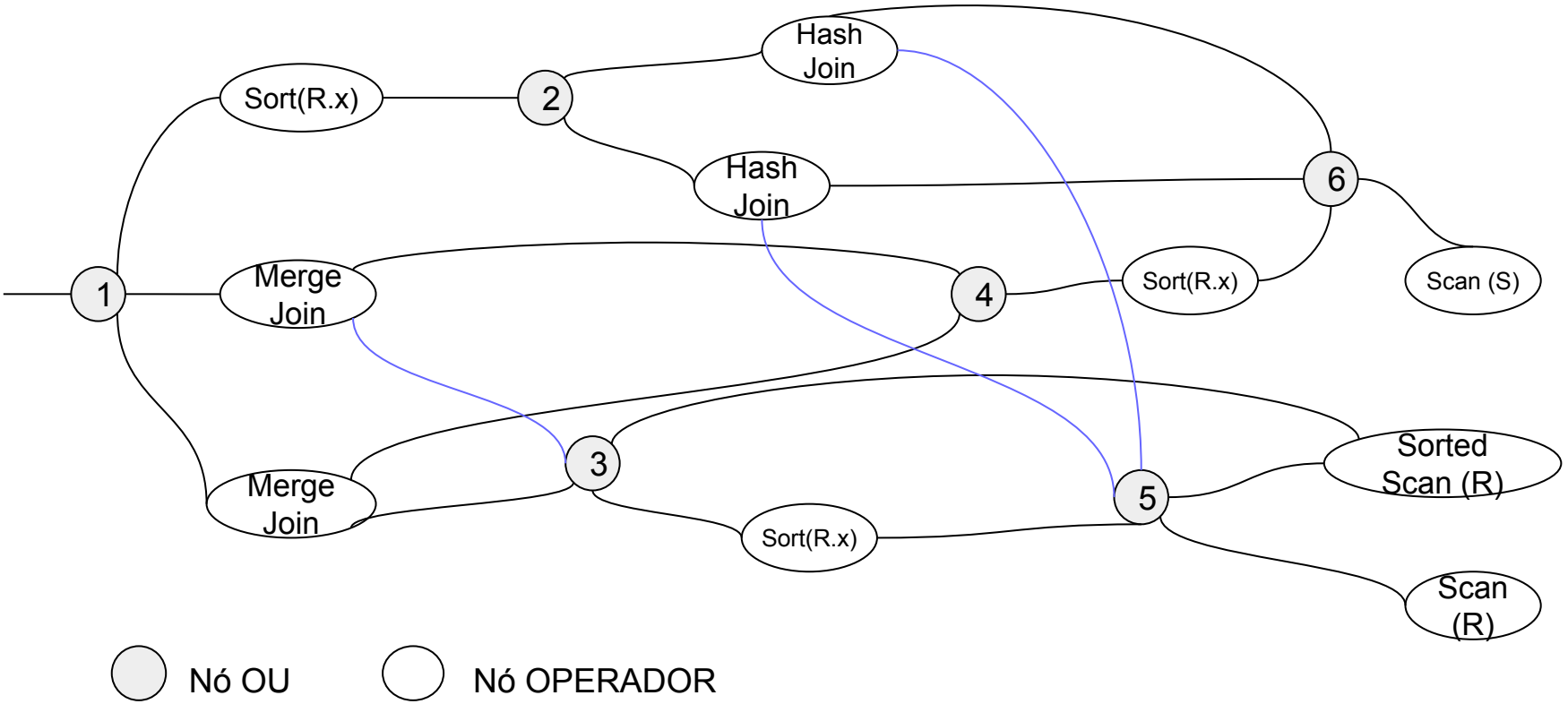
```
SELECT DISTINCT s.region
FROM brweather b natural join state s
WHERE s.location = 'Semi-Árido';
```

**Q3**

```
SELECT b.date, b.time, s.region, s.location, b.temperature
FROM brweather b natural join state s
WHERE  b.date >= '20090801'
AND b.date <= '20090831'
AND b.temperature = (SELECT MAX(temperature)  FROM brweather
                                    WHERE date >= '20090801'
                                    AND date <= '20090831'
                                    AND windspeed > 10);
```

# *Espaço de Busca*

- Opções que o otimizador deve considerar para responder a consulta
  - `select * from R Join S on R.x=S.y Order by R.x`



# *Dicionário de Dados*

- Grupos de tabelas que armazenam dados sobre os dados (metadados)
  - Tabelas do banco
    - Nome lógico, organização
    - Atributos (nome, tipo, tamanho)
    - Chaves e Restrições
  - Índices
    - Estrutura (B+, hash...)
    - Atributos envolvidos
  - Visões
    - Nome
    - Construção

# *Dicionário de Dados*

- Grupos de tabelas que armazenam dados sobre os dados (metadados)
  - Estatísticas
    - Cardinalidade da tabela: *NTuplas(T)*
    - Tamanho ocupado em páginas/blocos : *NPages(T)*
    - Cardinalidade de índices: *NKeys($I_T$)*
    - Tamanho do índice: *INPages($I_T$)* – para B+ só as folhas
    - Altura do índice: número de nós de uma B+ (*IHeight($I_T$)*)
    - Faixa de valores dos índices: menor *ILow($I_T$)* e maior *IHigh($I_T$)*.

# *Dicionário de Dados*

- Exemplo:
  - Tabela *Velejador(vid,vname, level, age)*
    - NTuplas(Velejador)=100.000
    - NPages(Velejador)=1.000
    - $I_{vid}$ => índice B+ sobre *vid*
    - NKeys($I_{vid}$)=100.000 (chave não possui duplicatas)
    - INPages($I_{vid}$)=200 (o índice é menor que a tabela)
    - IHeight($I_{vid}$)=10
    - ILow($I_{vid}$)=1
    - IHigh($I_{vid}$)=100000

# *Dicionário de Dados*

- Exemplo:
  - Tabela *Velejador(vid,vname, level, age)*
    - $I_{level}$ => índice B+ sobre *level*
    - NKeys($I_{level}$)=10
    - INPages($I_{level}$)=200
    - IHeight($I_{level}$)=3
    - ILow($I_{level}$)=1
    - IHigh($I_{lelvel}$)=10

# *Caminhos de Acesso*

- Maneira que o processador de consultas fará o acesso à tabela
  - Índice (quando o predicado poder utilizar índices)
    - Hash
      - Apenas para igualdade
    - Árvore B+
      - Permite ser utilizada para prefixo do índice (caso índice com mais de um atributo)
  - Varredura (Scan)
    - Quando não for possível ou não valha a pena aplicar índice
  - Ordenação em tempo real
    - Caso a ordenação otimize a seleção de tuplas

# *Caminhos de Acesso*

- Formas de acesso
  - Suponha hash `H(desc,bid,vid)` para Reserva
    - Utilizado em predicados como `desc='Verao' and bid=5 and vid=3`
    - Não para `desc='Verao' and bid=5`
  - Caso índice B+ (desc,bid,vid)
    - Podem ser utilizados >, <, >=, <= e =
    - Permite uso de prefixo
      - Predicado `desc='Verao' and bid=5` pode ser utilizado

# *Caminhos de Acesso*

- Formas de acesso
  - Índice (bid,vid) para Reserva (hash ou árvore)
    - Pode ser utilizado em como `desc='Verao' and bid=5 and vid=3`
      - Utiliza `bid=5 and vid=3` para reduzir o espaço de busca
      - No resultado aplica `desc='Verao'`
    - Múltiplos índices podem ser utilizados

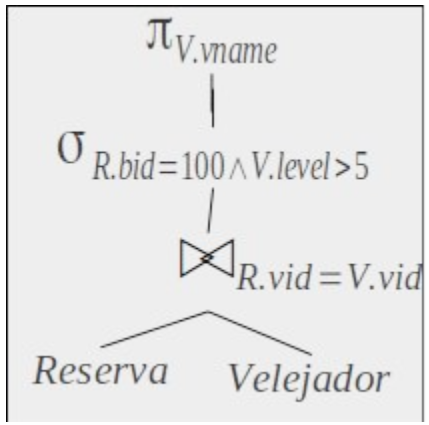
# *Caminhos de Acesso*

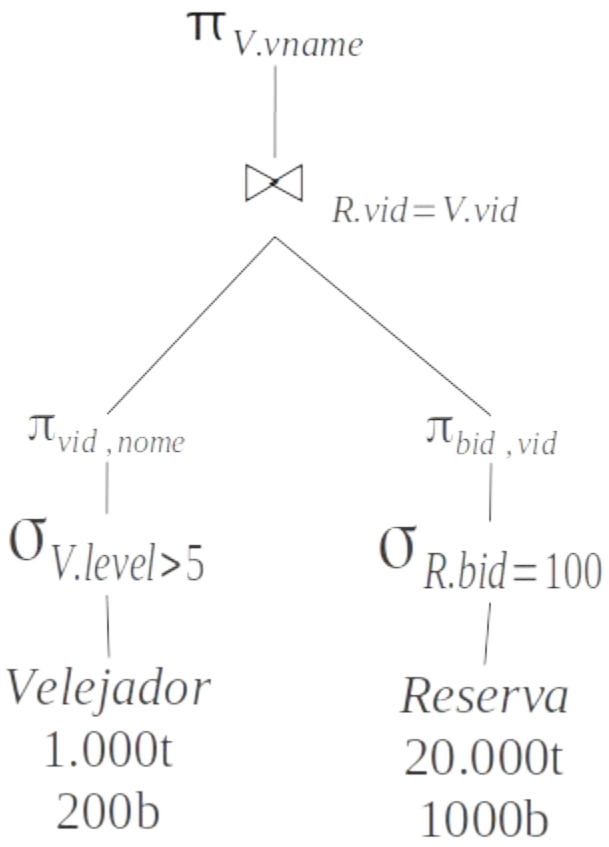
- Formas de acesso
  - Índices podem ser utilizados apenas em seleções conjuntivas (ligadas por **and**)
  - Condições disjuntivas (ligadas por **or**) são mais difíceis de tratar e otimizar

# *Seletividade de Caminhos*

- O processador de consultas deve decidir qual caminho de acesso escolher para uma consulta
  - Se uma tabela tem um índice:
    - Utilizar o índice para acessar os dados
    - Varrer a tabela diretamente
    - Utilizar apenas o índice
  - O caminho mais seletivo é aquele que retorna o menor número de páginas

# Revisitando o Exemplo




Índice B+ reserva(bid)
6 páginas

# *Revisitando o Exemplo*

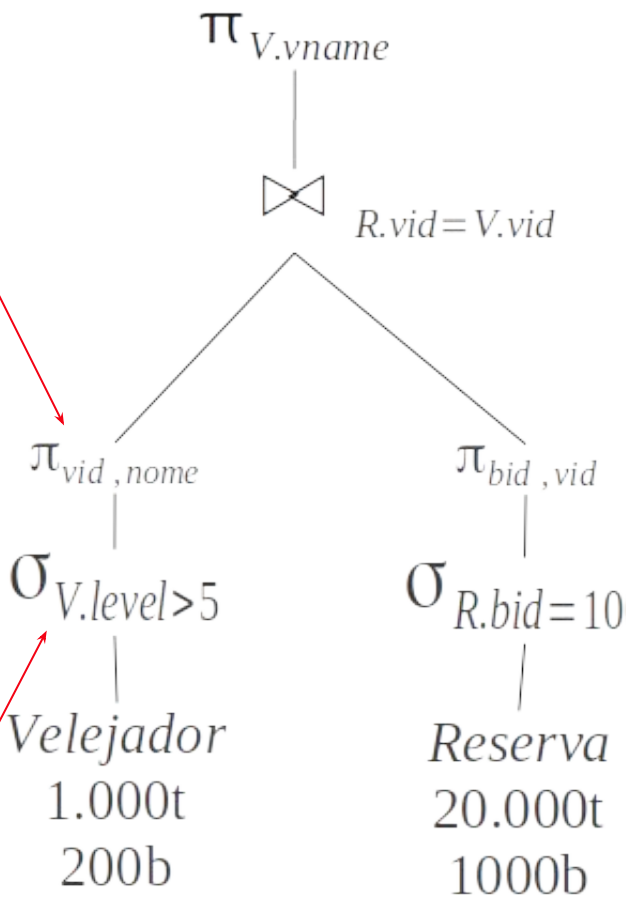
Reduz tamanho tupla de Velejador

Cria **temp1** com 50t e ocupa 1 página + 1 I/O

Resolve a seleção e a projeção em pipe

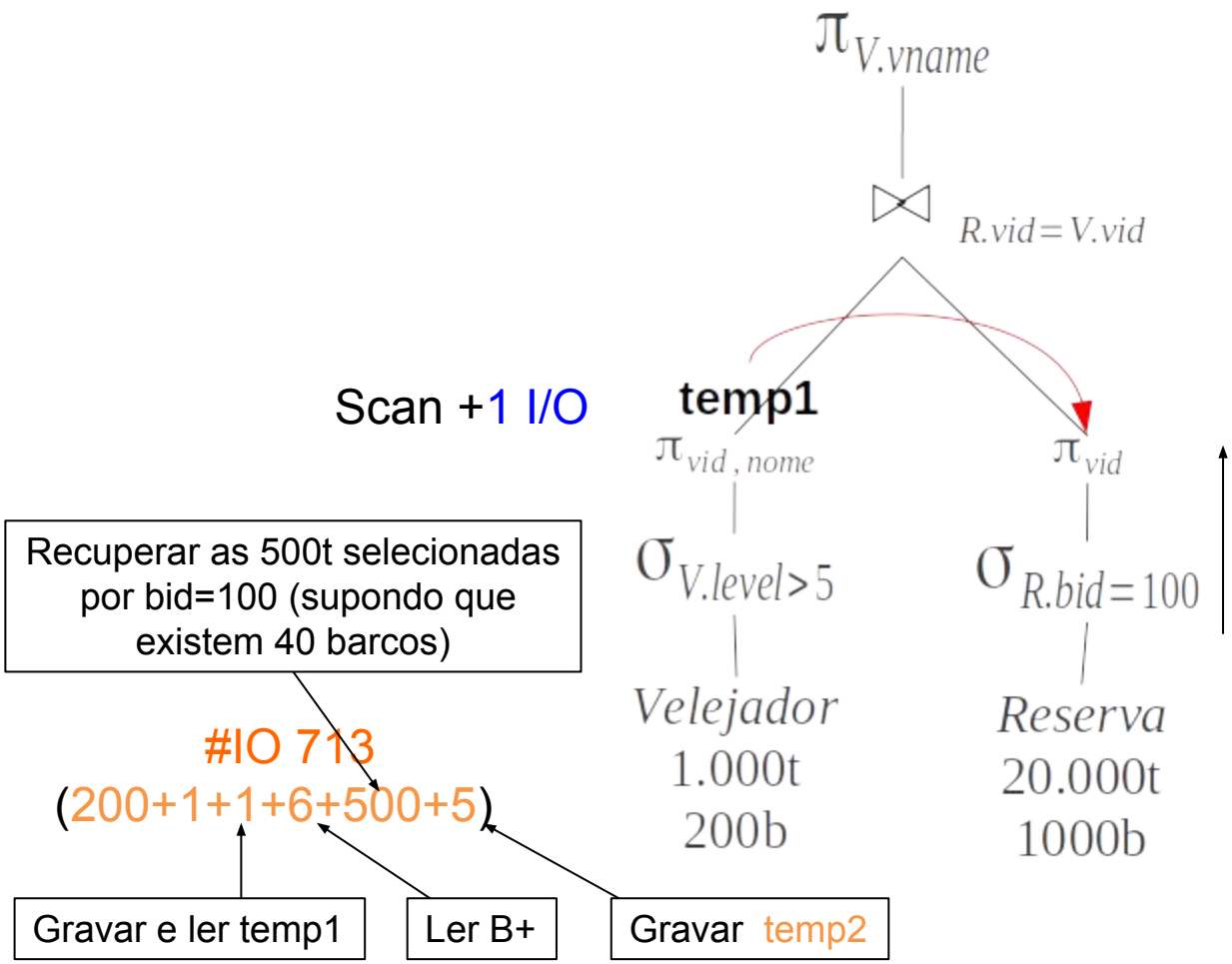
#IO 200 (carga Velejador)
+ 1 (armazena temp1)

Reduz Velejador para 50 t

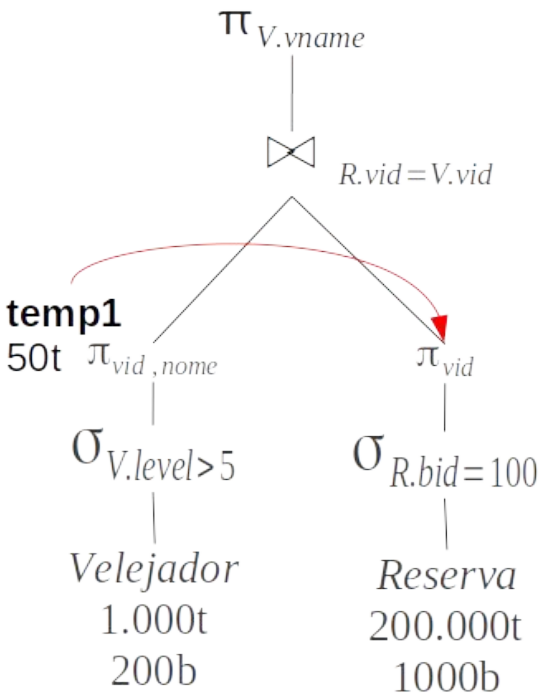



Índice B+ reserva(bid)
6 páginas

# Revisitando o Exemplo



# *Revisitando o Exemplo*




nested loop join

250 I/0 (50 tuplas acessando 5 páginas)

**temp2**

500t

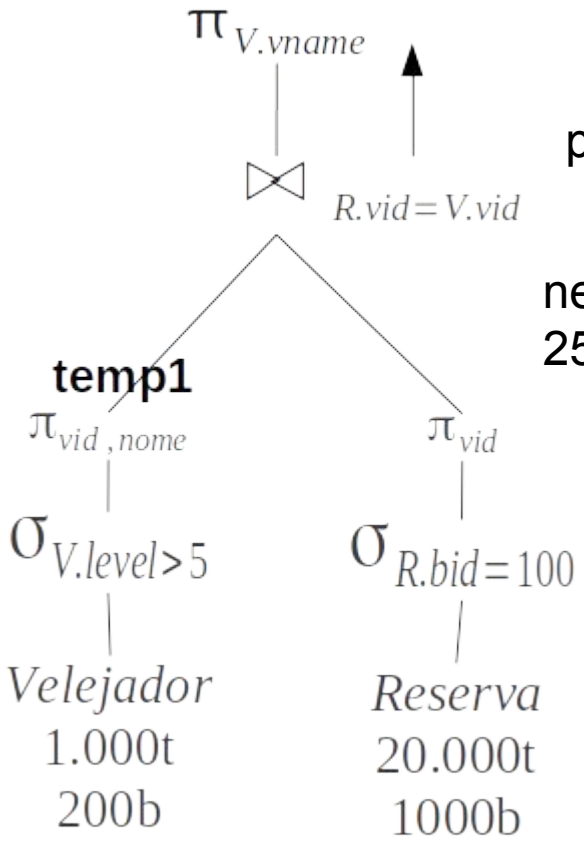
5b – 5 I/O

Índice B+ clusterizado reserva(bid) 6 páginas

#IO 713
\+ 250

# *Revisitando o Exemplo*

**Final #IO 963**




pipeline

Sem I/O pois as tuplas do resultado são mostradas quando selecionadas (pipeline)

nested loop join
250 I/0

**temp2**
500t
5b – 5 I/O

Índice B+ clusterizado
reserva(bid) 6 páginas

Anterior:
#IO 4.002.104

# *Fases de um Plano (deep-left)*

- Join com n tabelas na junção:
  - Passo 1: encontre o melhor plano para 1-relação para cada relação
  - Passo 2: encontre a melhor forma de fazer a junção para  o plano 1-relação com outra relação (planos para 2-relações)
  - Passo 3: encontre a melhor forma de fazer a junção para o plano (n-1)-relação para a n-ésima relação.
- Para cada sub-conjunto de relações, guarde:
  - O plano mais barato para um scan completo, e
  - O plano mais barato considerando índices ou ordenação externas

# *Fases de um Plano (deep-left)*

- Passo 1 (plano 1-relação)
  - Identifique os termos de seleção no WHERE que utilizam apenas os atributos de R (execute o acesso a R antes da junção)
  - Identifique os atributos de R que não aparecem nem no SELECT nem no WHERE (faça uma projeção sem esses atributos antes da junção)
  - Mantenha o plano mais barato para recuperar todas as tuplas (scan na tabela)
  - Mantenha o plano mais barato com tuplas na ordem da chave de procura (índice B+, hashing, ...)

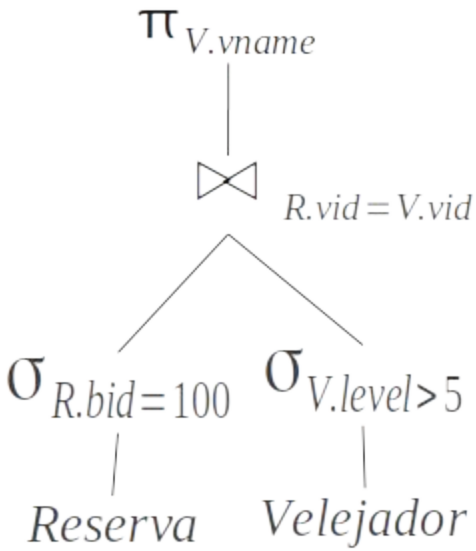
# *Fases de um Plano (deep-left)*

- Passo 2 (plano 2-relações)
  - Para cada plano 1-relação, faça o plano 2-relações considerando qual relação é a da esquerda
  - Verifique o WHERE
    - Seleções envolvendo apenas a relação interna são aplicados antes da junção
    - Seleções que definem a junção
    - Seleções que envolvam atributos de outras relação são aplicados após a junção
  - Aplique apenas as seleções que envolvam as tabelas no plano 2-relações
  - Dependendo do algoritmo de junção escolhido, a tabela resultante pode ser materializada

# *Fases de um Plano (deep-left)*

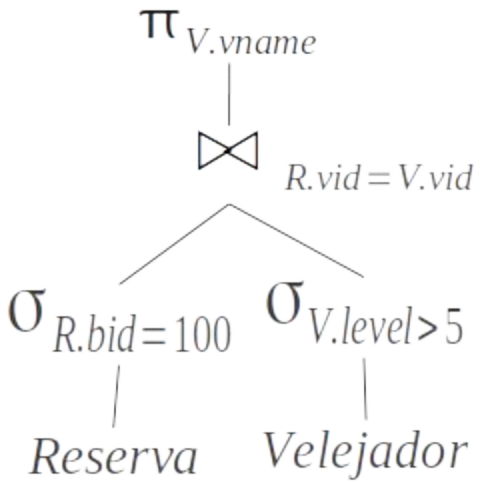
- Próximos passos
  - Planos n-relações são combinados apenas com outras relações que tenham atributos de junção
    - Evitar produto cartesiano
  - ORDER BY, GROUP BY e agregações são feitas como passo final utilizando ou “uma ordem interessante” ou “um operador de ordenação”

# *Fases de um Plano (deep-left)*



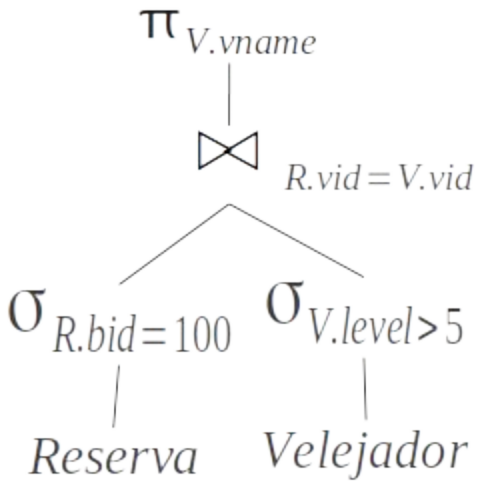

- Exemplos dos passos
  - Velejador B+ em level e Hash em vid
  - Reserva B+ em bid
- Passo 1:
  - Velejador
    - Escolhas: B+ para `level > 5` ou scan na tabela
    - (se a seleção espera retornar várias tuplas ou índice não clusterizado, talvez scan seja mais barato)
    - Mantem o plano com B+
  - Reserva
    - B+ em bid para `bid=100` (+barato)
    - Scan na tabela

# *Fases de um Plano (deep-left)*




- Exemplos dos passos
  - Velejador B+ em level e Hash em vid
  - Reserva B+ em bid
- Passo 2:
  - Cada plano do passo 1 é para a relação ser a da esquerda
    - Reserva à esquerda: índice Hash pode ser utilizado para recuperar as tuplas de velejador
      - Satisfaz r.vid=valor de vid da tupla da relação da direita
    - Velejador à esquerda: utilizar um algoritmo de merge-sort outra estratégia
    - Escolha: `reserva` ⋈ `velejador`

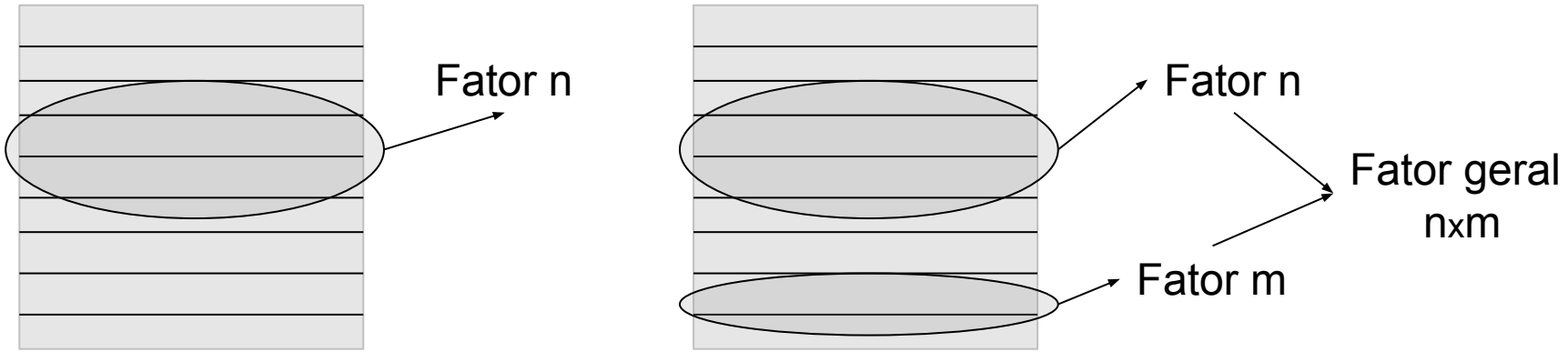
# *Fases de um Plano (deep-left)*




- Exemplos dos passos
  - Velejador B+ em level e Hash em vid
  - Reserva B+ em bid
- Resultado
  - Os predicados de seleção são executados com B+ em level e B+ em bid
  - A junção é feita lendo-se primeiro reserva e, utilizando Hash em vid, buscando as tuplas de velejador (*hash loop join*)

# *Seletividade de Caminhos*

- Fator de redução
  - Filtro que retorna apenas uma fração das tuplas de uma determinada tabela
  - Para várias condições, a fração de tuplas que satisfaçam todas as condições é aproximadamente a multiplicação de seus fatores de redução.



# *Seletividade de Caminhos*

- Fator de redução
  - coluna=valor
    - Se coluna tem índice $\frac{1}{NKeys(Indice_{coluna})}$
    - Senão padrão $\frac{1}{10}$
  - $coluna_1 = coluna_2$
    - Ambos com índice $\frac{1}{MAX(NKeys(Indice_{coluna_1}), NKeys(Indice_{coluna_2}))}$
    - Se apenas uma das colunas possuir índice, utiliza-se a fórmula de coluna=valor
    - Se nenhuma possuir utiliza-se o padrão.

# *Seletividade de Caminhos*

- Fator de redução
  - coluna > valor
    - Se coluna tem índice $\frac{High(coluna) - valor}{High(coluna) - Low(coluna)}$
    - Senão utiliza um valor <0,5 (arbitrário)
  - coluna in (lista valores)
    - Utiliza-se a estratégia coluna=coluna, multiplica-se os fatores

# *Seletividade de Caminhos*

- Exemplo
  - Índice hashing H (desc,bid,vid)
  - Predicado `desc='Verao' and bid=5 and vid=3`
  - NKeys(H)=150
  - NPages(Reserva)=100
  - Fórmula
    - $NPages(Reserva) \times \frac{1}{NKeys(H)}$
  - O fator de redução é 0,667.
  - Utilizando o índice para Reserva com, por exemplo, 100.000 tuplas, teríamos 66.700 descartadas com o índice

# *Seletividade de Caminhos*

- Exemplo
  - Se tem-se um Índice (bid,vid) com a condição `bid=5 and vid=3`
  - Se é conhecido o número de valores distintos de `bid`, pode-se estimar o fator de redução para esta coluna.
    - Por exemplo, se existem 30 valores distintos, a redução seria 1/30.
  - Fazendo o mesmos para `vid`, tem-se o fator de redução de `vid`.
  - Multiplicando os dois temos o fator de redução total
  - Pode-se estimar o fator de redução de uma condição como day > 10/02/2011. Assumindo que os valores estão uniformemente distribuídos

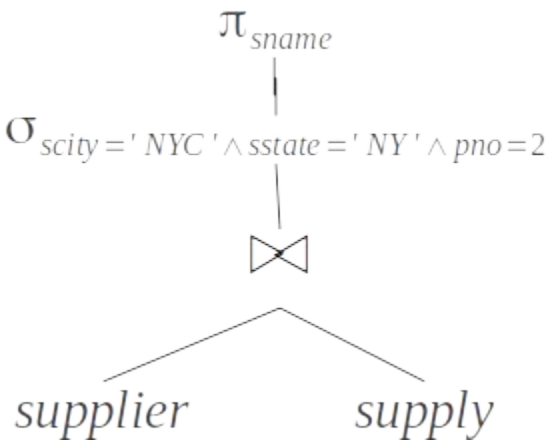
# *Seletividade de Caminhos*

- Exemplo
  - Pode-se estimar o fator de redução de uma condição como day > 10/02/2011. Assumindo que os valores estão uniformemente distribuídos:

$$\frac{High(day) - valor}{High(day) - Low(day)}$$

# *Planos*

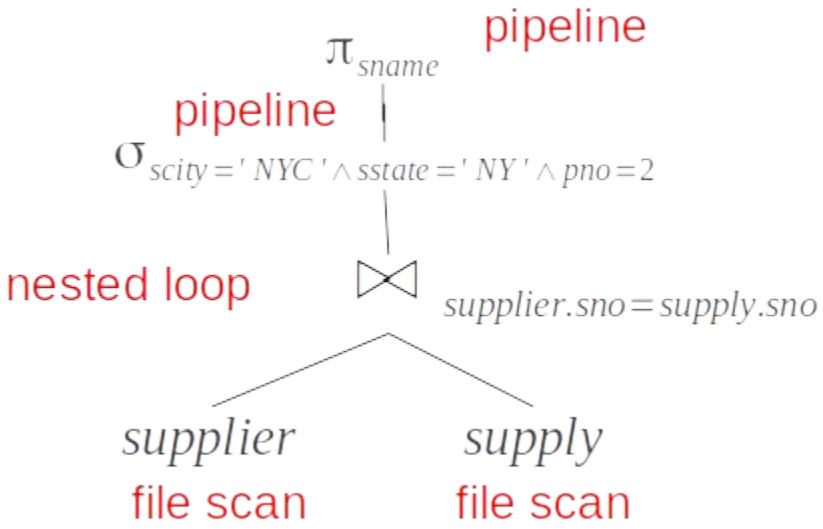
- Para cada plano lógico pode existir vários planos físicos



# *Planos*

- Para cada plano lógico pode existir vários planos físicos

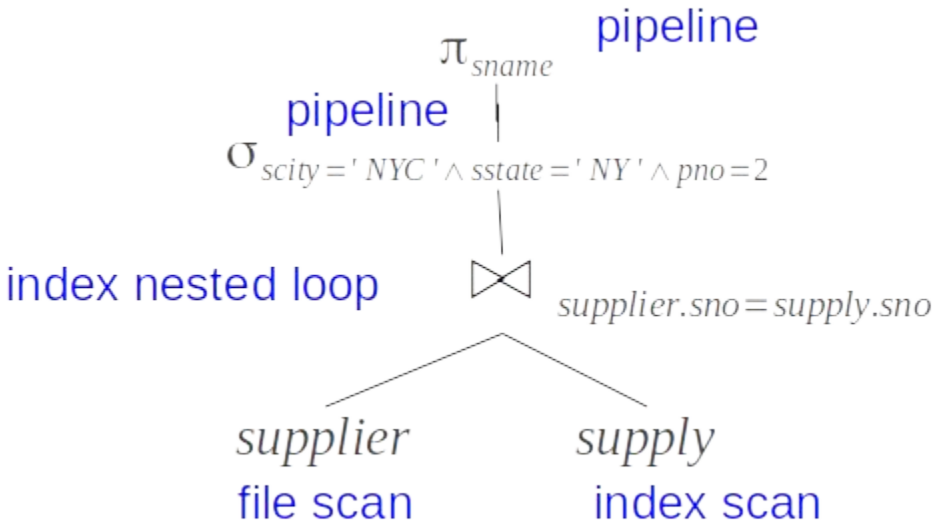
**Plano Físico 1**



# *Planos*

- Para cada plano lógico pode existir vários planos físicos

**Plano Físico 2**



# *Planos*

- Para cada plano lógico pode existir vários planos físicos
  - Quanto menos nós em  plano lógico mais rápida será a execução.

# *O Que Faz Um Plano Ser Bom*

- O planejador escolhe um plano (entre vários) baseado no seu custo estimado
- Assume: o I/O domina o custo de uma consulta (assim, seleciona o plano que requer menos I/O)
  - I/O randômico é mais caro que I/O sequencial nos hardwares modernos
- O I/O é estimado tentando predizer o tamanho dos resultados intermediários, usando as estatísticas do SGBD
  - Isso é uma ciência imperfeita (para não dizer algo pior)
- Distinguir entre custo de saída (I/O para a primeira tupla) e custo total

# *Princípios Gerais de Otimização*

- O custo de um nó é em função de sua entrada: o número de linhas produzidas pelos nós filhos e a distribuição de seus valores:
  - Reorganizar os nós pode mudar todo o custo
  - Uma escolha pobre perto das folhas pode levar a um desastre
  - Aplicar os predicados mais cedo para reduzir o número de tuplas em resultados intermediários
- Mantenha em mente a ordenação: uma entrada ordenada pode ter planos mais baratos
- Planejar os joins de forma eficiente é fundamental

# *Algoritmo Planejador (PG)*

- Conceitualmente, três fases:
  - Enumerar todos os planos disponíveis
  - Avaliar o custo de cada plano
  - Escolher o mais barato
- Naturalmente, executar essas fases não seria muito eficiente
- O algoritmo do projeto Sistema R é normalmente utilizado (algoritmo PD criado pela IBM em 1970s)
- Ideia básica: encontre bons planos para uma consulta simplificada com n joins
  - Encontre bons planos para n+1 joins, junte os planos (Repita)

# *Algoritmo do Sistema R*

- Considera cada relação de base, busca sequencial e busca por índices disponíveis, aplicando os predicados que aparecem as relação de base
  - O plano sem ordem mais barato
  - O plano para cada ordem de classificação
- Enquanto planos candidatos têm menos joins que o necessário, adicione cada plano com uma relação que ainda não está no plano. Guarde:
  - Os planos sem ordem mais baratos para cada conjunto distinto de relação
  - Os planos ordenados mais baratos para cada conjunto distinto de relação

# *Algoritmo do Sistema R*

- Agrupamentos e ordenações são feitos em etapas finais
- Considere planos left, right ou bushy trees (left é preferível)
- O número de planos considerados explode em relação ao crescimento do número de joins.
  - Para consultas com mais de 11 joins, o PostgreSQL utiliza um algoritmo genético (GEQO – Genetic Query Optimization)
    - Busca não exaustiva e não determinística para possíveis ordenações de joins utilizando left-deep tree.

# *Planejamento Outer Joins*

- São como os inner joins, exceto por incluírem tuplas que não têm correspondência no resultado
- Operadores de inner join são associativos e comutativos
  - Operadores de outer joins não são nem associativos nem comutativos (não é possível reordená-los)
  - Menos opções para os planos com outer join
- As vezes podem ser convertidos para inner join:
  - `select * from a left  join b on b.x=<condição>`
- No PostreSQL pode-se forçar a ordem de inner join setando o parâmetro join_collapse_limit para 1

# *Planejamento sub queries*

- Três tipos: IN-cláusula, FROM-lista e expressão
- Sempre “puxar pra cima” as subqueries IN para tornarem um tipo especial de join com o pai
- Tentar “puxar pra cima” as subqueries FROM para tornarem um tipo especial de join com o pai
  - Pode ser feito se as subqueries são simples: sem order by, having ou group by
  - Caso contrário, avaliar a subquery através de um nó plano separado – semelhante a um scan

# *Planejamento sub queries*

- FROM-lista:

```
select * from filme f, (select * from diretor d
                        where d.city='NYC') d
where f.codd=d.codd;
--- otimizador pode converter
select * from filme f, diretor d
where f.codd=d.codd and d.city='NYC';
```

- A subquery foi puxada permitindo ao planejador todas as abordagens para otimizar join
- Integrando os qualificadores da subquery na consulta pai pode fazer com o otimizador otimize melhor a consulta pai

# *Planejamento sub queries*

- Expressões
  - Produz planos aninhados através de chamadas recursivas do planejador
  - Subqueries sem relação com o pai necessitam ser executadas apenas um vez (melhor para o plano)

```
select * from filme f
where f.codd=(select codd from diretor
                where lname='Spielberg');
--- A subquery é independente
$var= select codd from diretor where lname='Spielberg';
select * from filme f where f.codd=$var;
```

- Se a subquery utiliza valores do pai, a mesma deva ser avaliada durante a execução do pai

# *Planejamento operadores de conjunto*

- O planejamento é primitivo (trivial)
- Gera planos para as subqueries filhos, então adicione um nó para concatenar os conjuntos resultantes
- Alguns operadores exigem mais trabalhos
  - Union: ordena e remove duplicatas
  - Except, Intersect: ordena e remove duplicatas, produz o conjunto resultante através de um scan linear
- Não são consideradas nenhuma outra alternativa, assim os planos são simples

# *Passos Otimizador Consultas*

| Step | Entrada | Componente | Saída |
|---|---|---|---|
| 1 | String da consulta | Parser | Query Tree |
| 2 | Query Tree | Checker | Valid Query Tree |
| 3 | Valid Query Tree | View Expander | Valid Query Tree wo/ Views |
| 4 | Valid Query Tree wo/ Views | Gerador de plano lógico | Plano lógico |
| 5 | Plano lógico | Rewriter (heurística) | Plano lógico otimizado |
| 6 | Plano lógico | Gerador de plano físico (custo) | Plano físico |
| 7 | Plano físico | Gerador de código | Código executável |
| 8 | Código executável | Executor | Resposta |

# *Melhoras Potenciais*

- Difíceis
  - Estatísticas do banco para correlação entre colunas
  - Funções de otimização
  - Reescrever o GEQO
- Quase impossíveis:
  - Recolher as estatística online
  - O executor receber online o feedback do otimizador
  - Processamento paralelo em um servidor (uma consulta em múltiplos processadores concorrentes)
  - Distribuição do processamento da consulta (via rede)

# *Conclusões*

- Consultas são as operações mais caras do SGBD
- Encontrar a melhor forma de executar uma consulta é um problema intratável em computação
- O dicionário de dados tem um papel importante no processamento de consultas
- Índices são essenciais
- O SGBD tenta resolver os problemas de otimização de forma automática