Tutorial 10 Query Processing

CSCI3170 Tutorial

Table of Content

- Query Evaluation
  - Join
  - Selection
  - Projection
- Query Optimization

Schema

- Supplier(sid, sname, location)
    - 500 pages, 80 tuples/page

- Supplier_Part(sid, pid, quantity)
    - 1000 pages, 120 tuples/page

Example

```
Select *
From Supplier S, Supplier_Part SP
Where S.sid = SP.sid
```

Join Operation

- Nested Loops Join
  - A tuple at a time
  - A page at a time
- Block Nested Loops Join
- Index Nested Loops Join
- Sort-Merge Join

A tuple at a time

```
for each tuples s in S  do (S is called outer relation)
    for each tuple sp in SP do (SP is called inner relation)
        if (s.sid = sp.sid) then
            add <s, sp> to result set
```

Cost:

- Scan S: 500 I/Os
- For each tuple of S, SP is scanned once: 1000 I/Os
- Total = 500 + **500 * 80 * 1000** = 40,000,500
- Switch S and SP, the total is 1000 + 1000 * 120 * 500 = 60,001,000

A page at a time

Improve the join by joining a page of tuples at a time

```
for each page p of S
    for each page q of SP
        output all $s \in p$ and $sp \in q$ such that $s.sid = sp.sid$
```

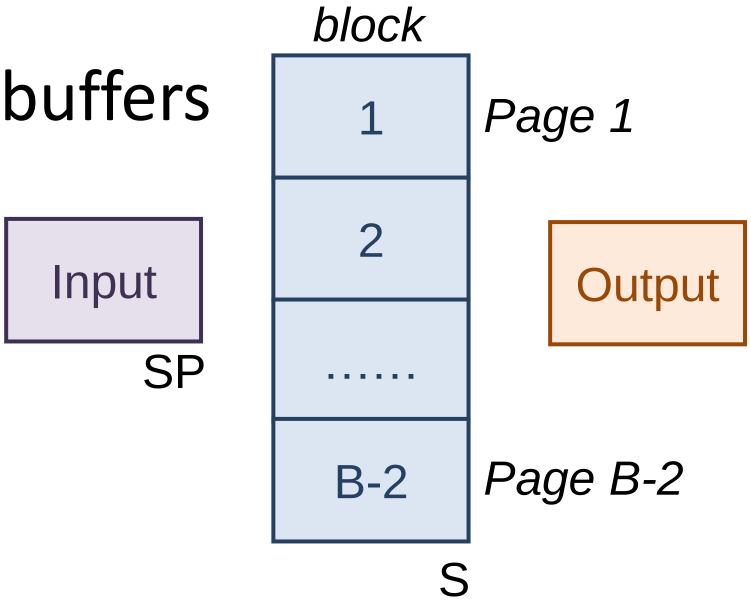
Cost

- Scan S: 500 I/Os
- For each page of S, SP is scanned once: 1000 I/Os
- Total: 500 + 500x1000 = 500,500

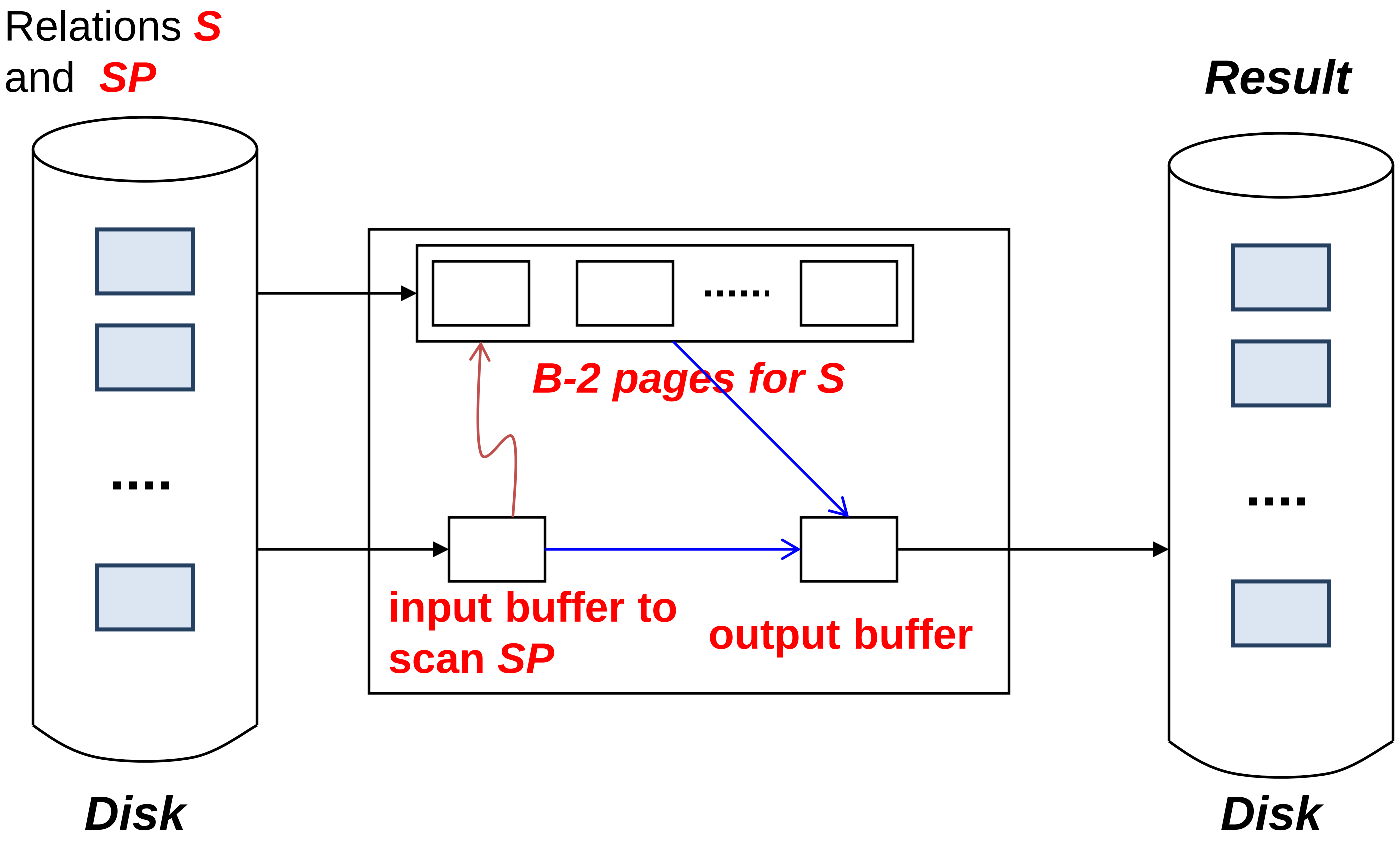
Block Nested Loops Join

```
For each block P for S
    For each page q for SP
        For each s ∈ P and sp ∈ q such that s.sid = sp.sid
            add <s, sp> to the result.
```

- Suppose we assign enough buffers to hold B pages. (B > 2)
  - 1 for input buffer (for SP)
  - 1 for output buffer
  - B-2 for outer relation (for S)



Block Nested Loops Join (Cont.)



Block Nested Loops Join (Cont.)

- Cost (Assume B = 52. So S is divided into 10 blocks.)
  - Scan S: 500 I/Os
  - For each block of S, scan SP once: 1000 I/Os
  - Total: 500 + 10 × 1000 = 10,500
  - Switch S and SP, the cost is: 1000 + 20 × 500 = 11,000
- Observation:
  - Choice of outer and inner relation will affect the cost.

    ---- Choose the **smaller one** as the outer relation
  - The buffer size will affect the cost

    ---- The bigger is the buffer, the fewer is the I/O cost

    ---- Trade off between space and time

Index Nested Loops Join

- Assume we have a hash index on sid of S then

```
For each sp $\in$ SP do
    For each s $\in$S where s.sid = sp.sid (use index)
        add <s, sp> to the result
```

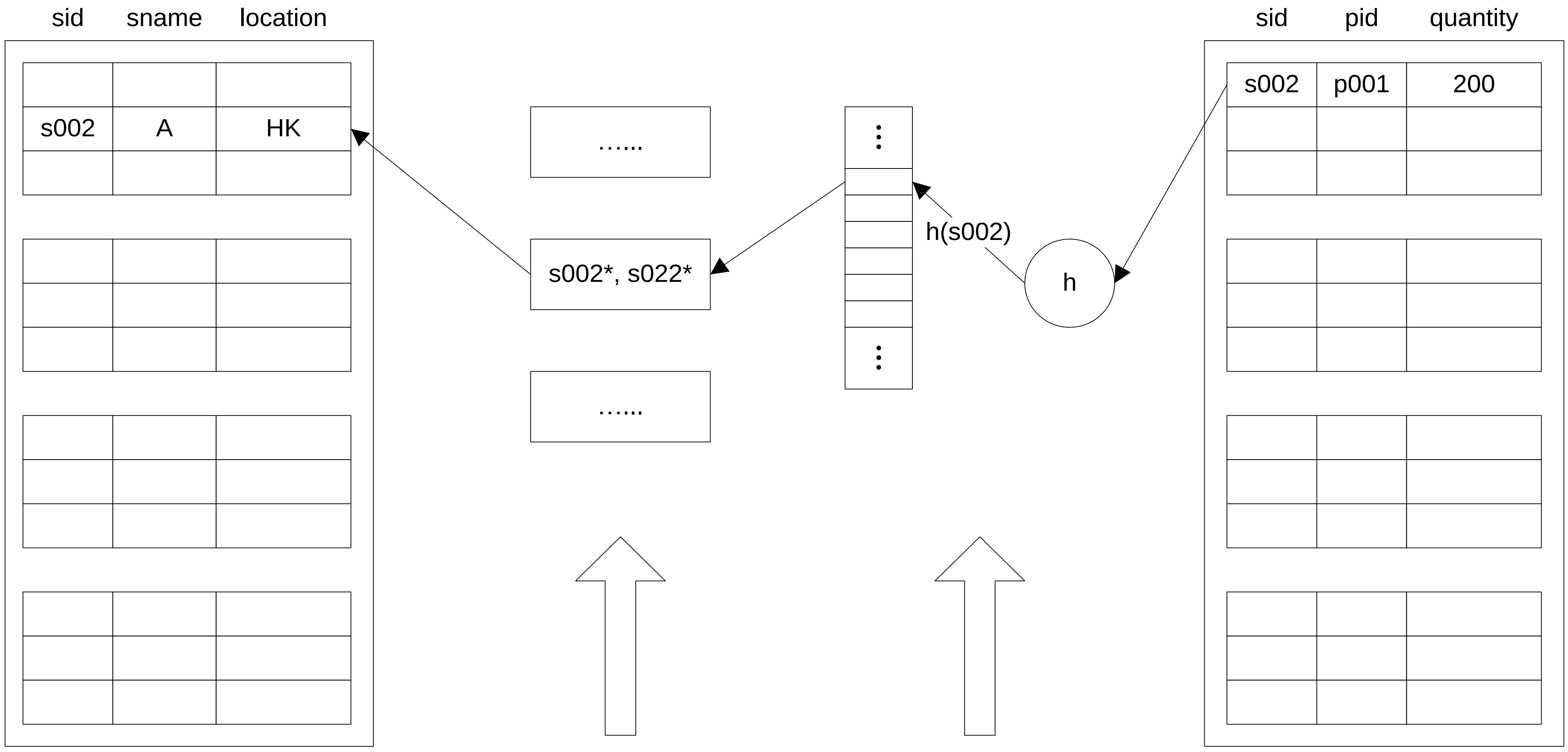
- Note that for each tuple in S, we use index to find match tuple in SP.

Cost

Obtained from experiments
(for overflow pages)

- Scan SP: 1000 I/Os
- For each tuple in SP, an average of 1.2 I/O to get to the bucket page containing the matching S data entry, retrieve the S tuple for 1 I/O (Note: sid is the primary key of Supplier relation)
- Total: $1000 + 120 \times 1000 \times (1+1.2) = 265{,}000$ I/Os

Index Nested Loops Join (2)



sid	sname	location
s002	A	HK

sid	pid	quantity
s002	p001	200

……..

s002*, s022*

……..

h(s002)

h

Supplier

Buckets

Directory

Supplier_Part

Index Nested Loops Join (3)

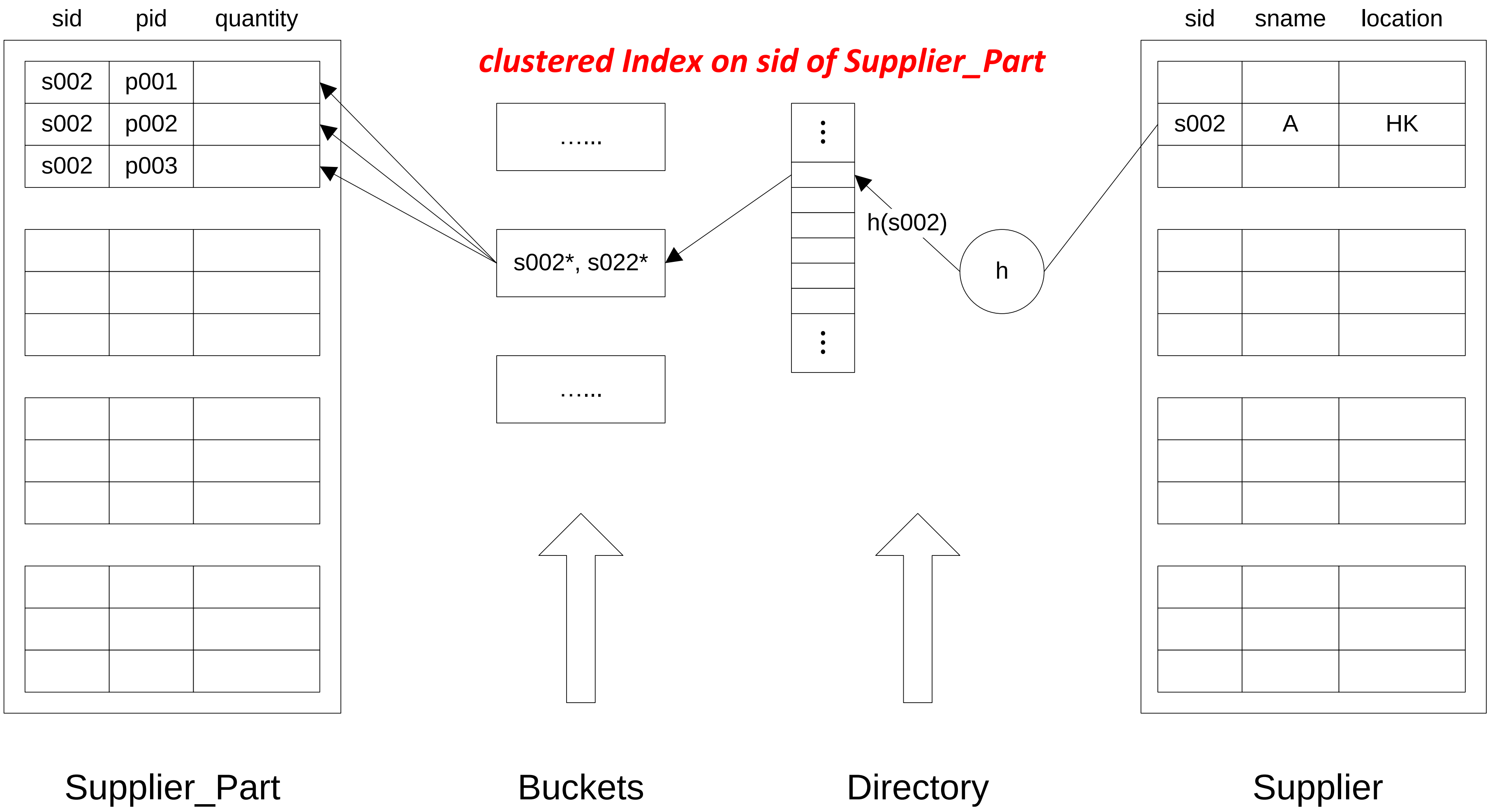
- Assume a hash index on sid of SP

```
foreach $s \in S$ do
    foreach $sp \in SP$ where $s.sid = sp.sid$ (use index)
        add <s, sp> to the result
```

- Note that for each tuple in S, we use index to find match tuple in SP.

Index Nested Loops Join (4)

clustered Index on sid of Supplier_Part



sid	pid	quantity
s002	p001	
s002	p002	
s002	p003	

sid	sname	location
s002	A	HK

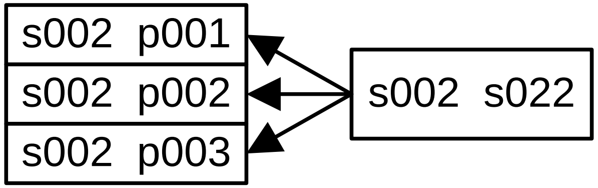
Index Nested Loops Join (5)

- Scan S: 500 I/Os
- For each tuple in S, an average of 1.2 I/O to get to the bucket page containing the matching SP data entry
- Estimation: 40000 suppliers supply 120000 parts, so each supplier supplies 3 parts on average.
  - Clustered Index (3 parts are in same page):

    total = $500 + 40000 \times (1.2 + 1) = 88{,}500$
  - Unclustered Index (3 parts are in 3 different pages in the worst case):

    total = $500 + 40000 \times (1.2 + 3) = 168{,}500$

Clustered Index



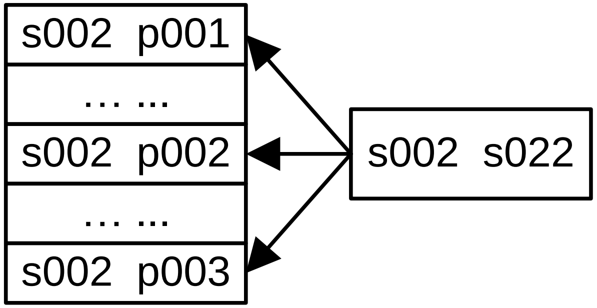
s002 p001

s002 p002

s002 p003

s002 s022

Unclustered Index



s002 p001

… ...

s002 p002

… ...

s002 p003

s002 s022

Sort-Merge Join

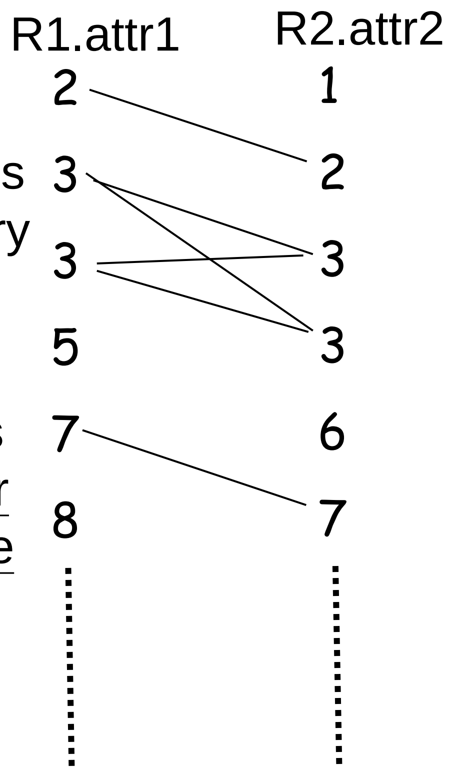
- $S \bowtie_{i=j} SP$
- Sort Supplier and Supplier_Part in ascending order on the sid, then scan them to do a merge
- Scan S until s.sid >= sp.sid
- Scan SP until sp.sid >= s.sid
- Until s.sid = sp.sid. At this point, all S tuples with same value in S_i (current S group) and all SP tuples with same value in SP_j (current SP group) match. Output <s, sp> for all pairs of such tuples.
- Resume scanning S and SP

Sort-Merge Join (Cont.)

Case 1:

Both join attributes are not the primary key

R1 is scanned once, R2 group is scanned once per matching R1 tuple

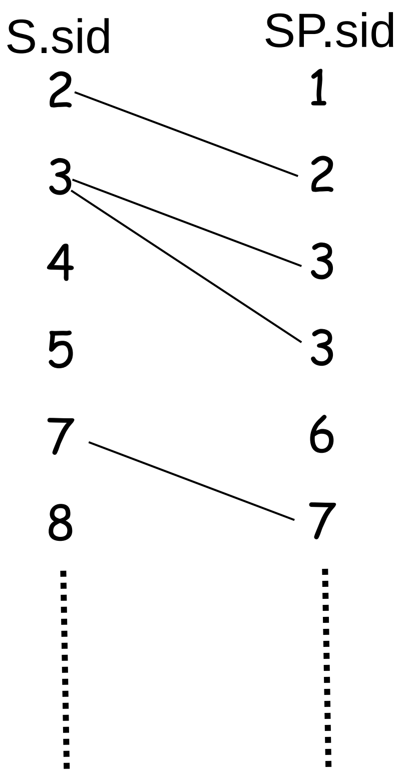


Case 2:

One of the join attribute is the primary key

Both S, SP are scanned once

Cost of join is the sum of the size of S and SP, plus the cost of sorting these two relations



Query Optimization

- Motivation
  - Ideal: find the best plan
  - Practical: avoid the worst plan
- Optimization steps
- Query Evaluation Plan
- An example

Optimization Steps

- A query is essentially treated as a σ-Π-$\bowtie$ algebra expression
- Optimizing such a relational algebra expression involves two basic steps:
  - Enumerate alternative plans for expression evaluation.
  - Estimate the cost of each plan and choose the plan with the lowest cost.

Query Evaluation Plan

- An extended algebra tree with annotations
- Each node indicates the relational operator and the implementation method for the relational operator.
- Each edge points to where the input comes from

Example

```
Select S.sname
From Supplier S, Supplier_Part SP
Where  S.sid = SP.sid and
          SP.pid > ‘p800’ and
          S.location = ‘HK’
```

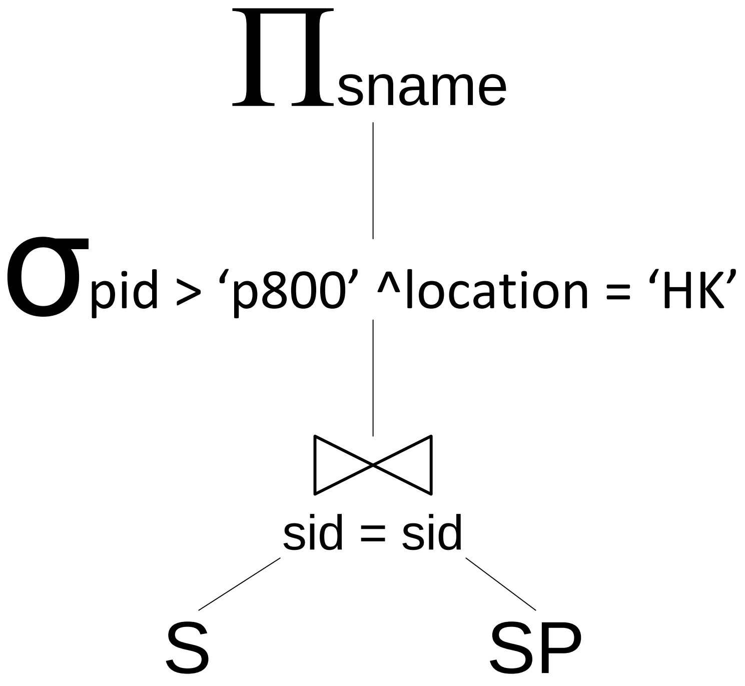
- Supplier:
  - 500 pages, 80 tuples/page
  - 50 possible locations, uniformly distributed
- Supplier_Part:
  - 1000 pages, 120 tuples/page
  - Max part id is p1000, min part id is p1, uniformly distributed

Example (cont.)

- Relation algebra of that query:

$\Pi_{sname}(\sigma_{pid > \text{‘p800’} \wedge location = \text{‘HK’}} (S \bowtie_{sid=sid} SP))$

- This algebra can be shown as a tree:



Π_{sname}

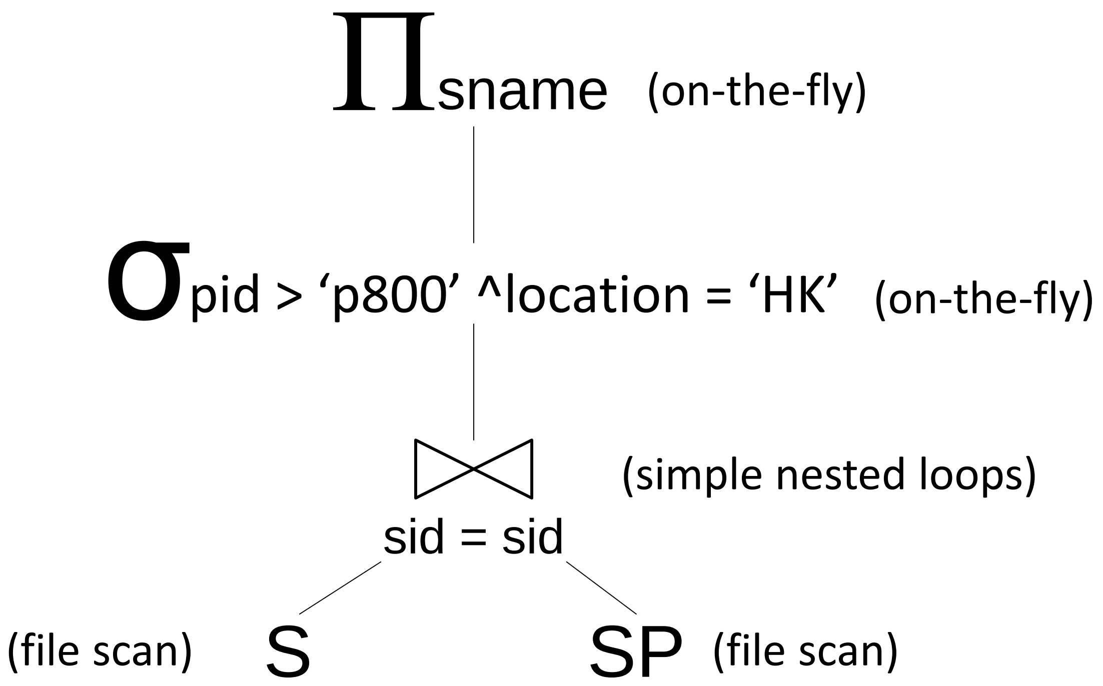
$\sigma_{pid > \text{‘p800’} \wedge location = \text{‘HK’}}$

$\bowtie_{sid = sid}$

S

SP

Full evaluation plan



Π_{sname} (on-the-fly)

$\sigma_{pid > ‘p800’ \wedge location = ‘HK’}$ (on-the-fly)

$\bowtie_{sid = sid}$ (simple nested loops)

(file scan) S

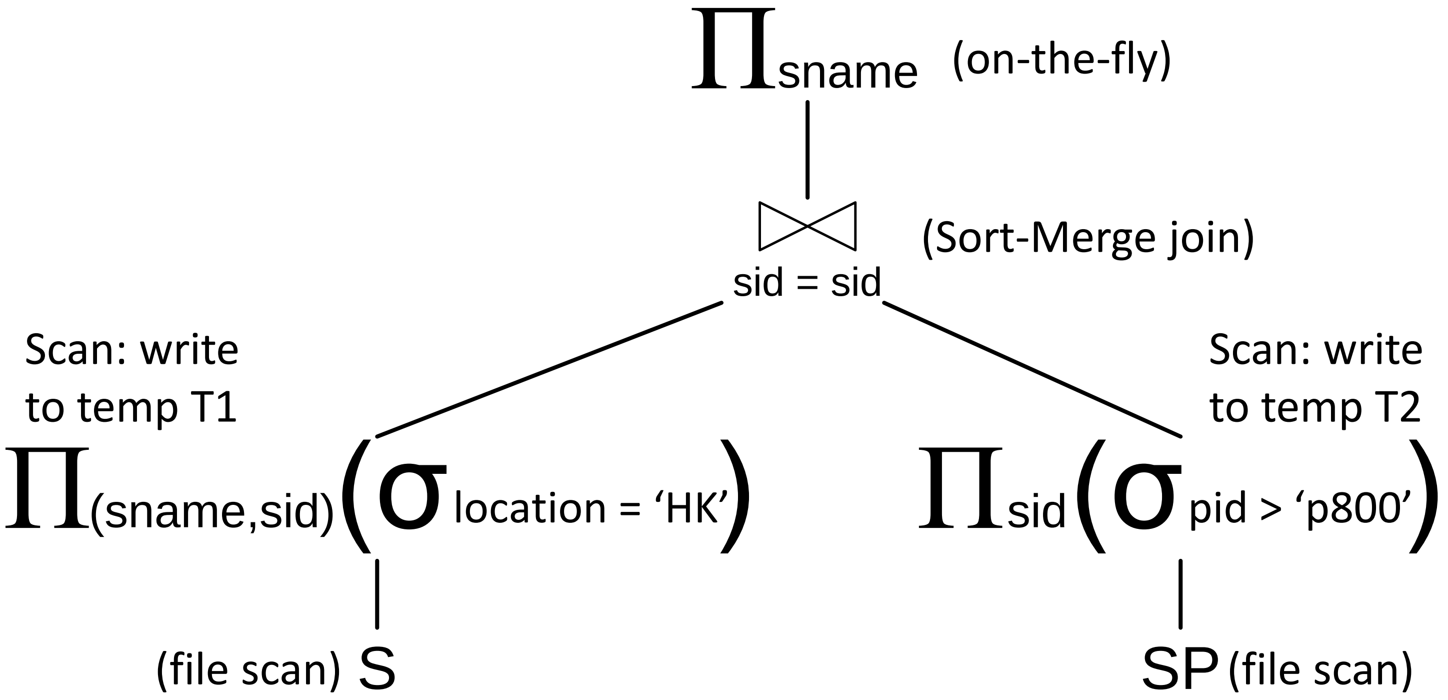
SP (file scan)

Cost: 500 + 500 × 1000 = 500500

Push Selection and projection

- May greatly reduce the data involved in a join
- In the example, if we do the selection and projection first, then the join will involve small portion of data with sid, sname attributes only.
- May reduce the cost sometimes

Example



Π_{sname} (on-the-fly)

$\bowtie_{sid = sid}$ (Sort-Merge join)

Scan: write to temp T1

$\Pi_{(sname,sid)}(\sigma_{location = \text{'HK'}})$

Scan: write to temp T2

$\Pi_{sid}(\sigma_{pid > \text{'p800'}})$

(file scan) S

SP (file scan)

Cost:
Scan S: 500 I/Os; write T1: 10 I/Os.
Scan SP: 1000 I/Os; write T2: 200 I/Os.
Sort-merge join of T1 and T2: 10 + 200 + I/Os for sorting T1 and T2.