ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΕΠΕΡΩΤΗΣΕΩΝ

Γιάννης Κωτίδης

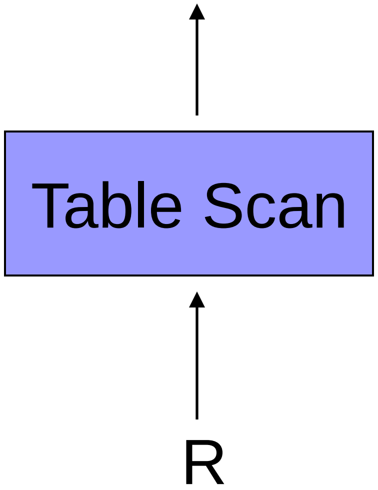
Σύνθετο παράδειγμα

- ΣΔΒΔ με 2 αλγόριθμους σύζευξης
  - NLJ
  - Sort-Merge join
- Δύο access methods
  - Table Scan
  - Index Seek (we assume B+tree indexes that are dense & in memory)
- Κόστος: αριθμός I/O

Υπενθύμιση

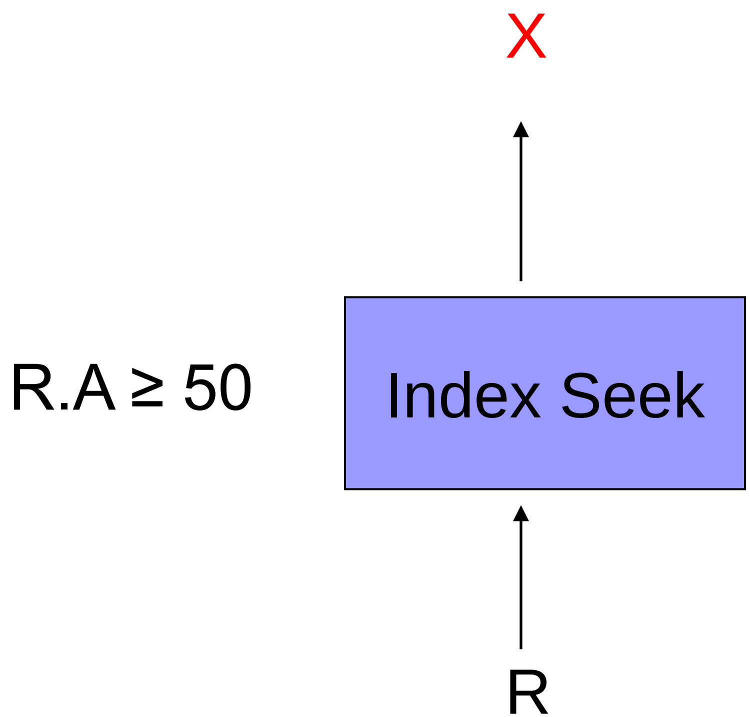
- T(R) : # εγγραφών της R
- B(R) : # σελίδων της R

Κόστος: Table Scan



Cost: B(R)

Κόστος Index Seek
(ευρετήριο συστάδων-clustered index)

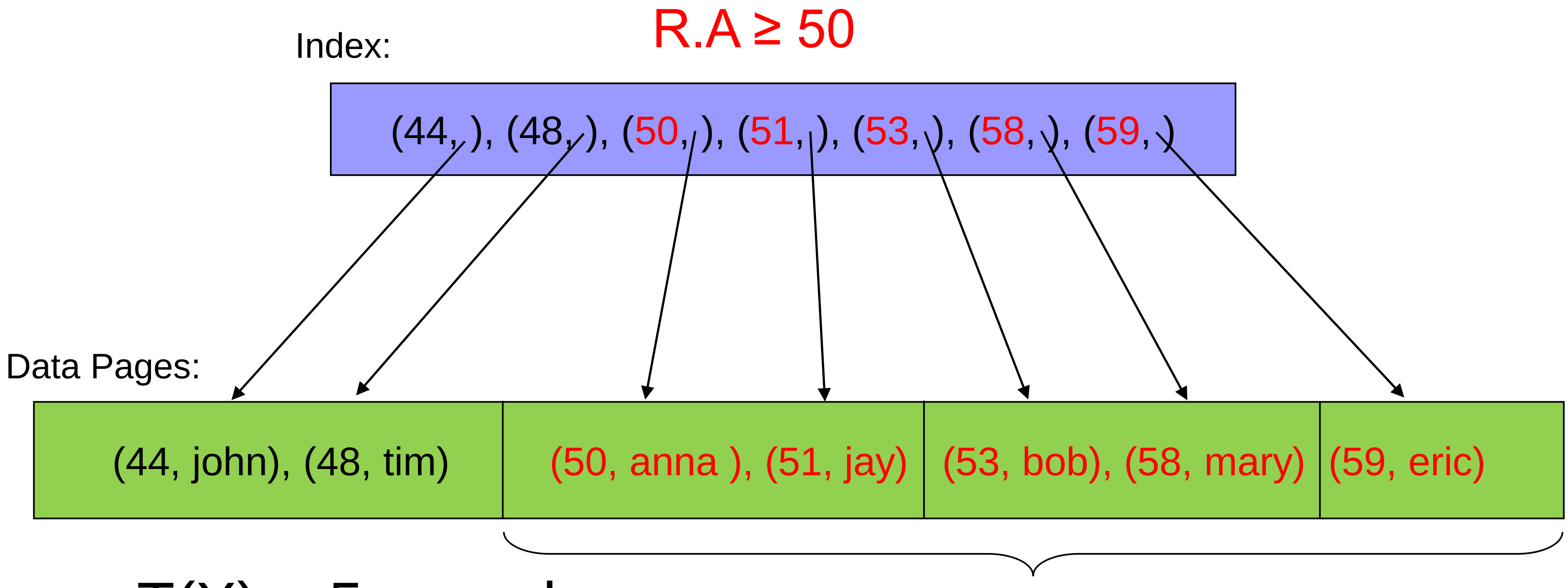


Cost: B(X)

(best case scenario)

Index Seek (Conceptual)

(ευρετήριο συστάδων-clustered index)

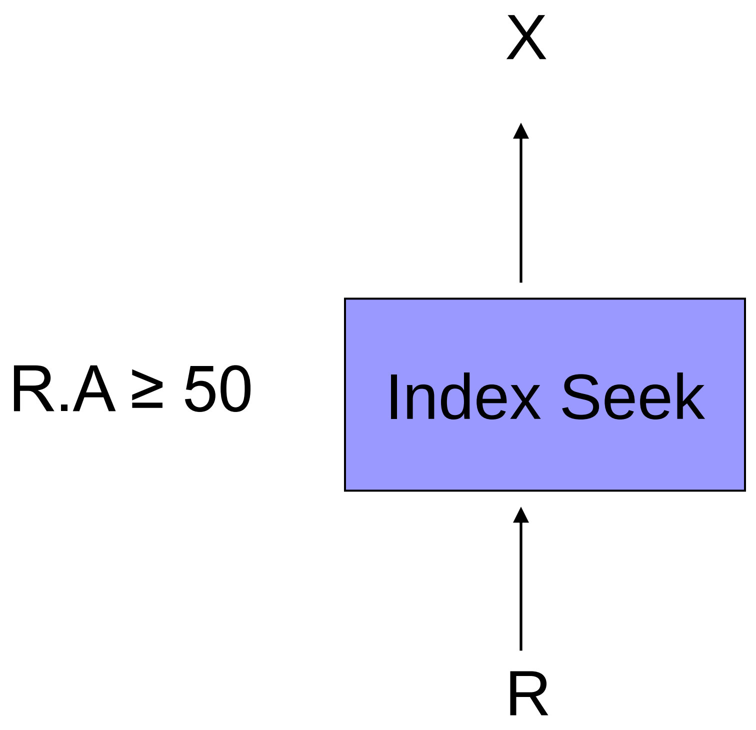


Cost(X) = 3 pages

T(X) = 5 records

B(X) = 3 pages

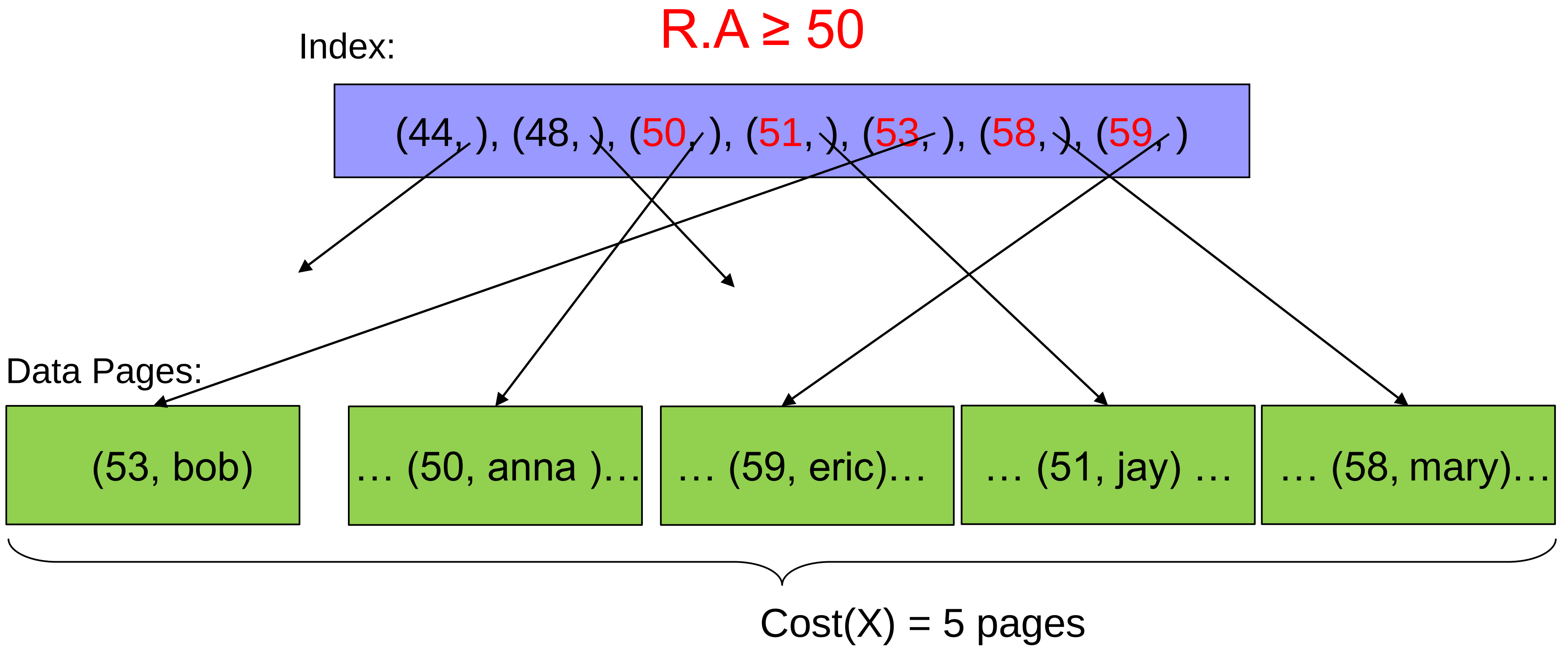
Κόστος Index Seek (non-clustered index)



Cost: T(X)

(worst case)

Index Seek (Conceptual)
(non-clustered index)



T(X) = 5 records

Παράδειγμα

- B(R) = 1000, T(R)=5000
- V(R,A) = 500

```
select * from R where R.A=761;
```

- Cost(no-index) = ?
- Cost(non-clustering index on R.A) = ?
- Cost(clustering index on R.A) = ?
- Size of the result in pages = ?

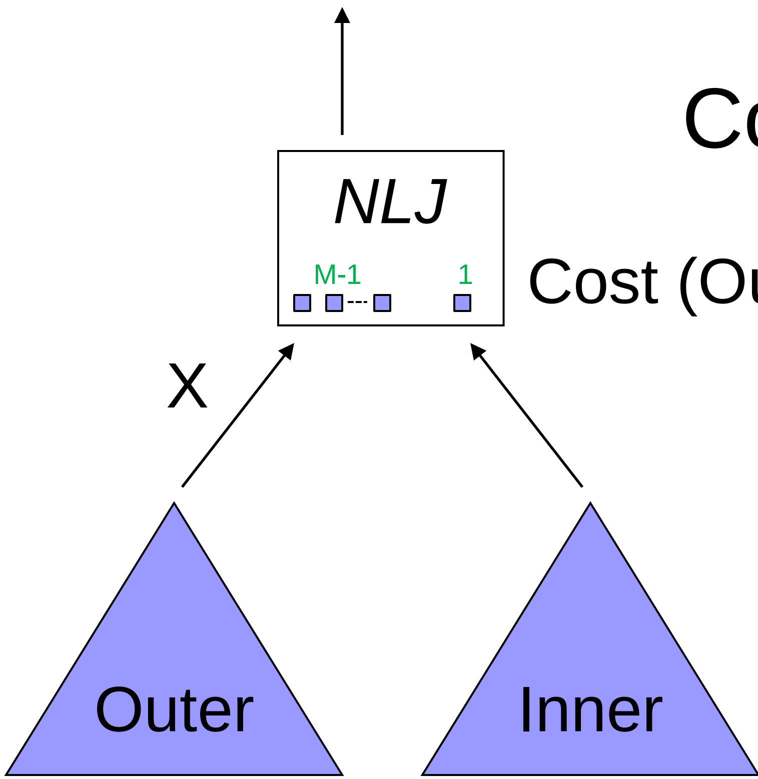
Παράδειγμα

- B(R) = 1000, T(R)=5000
- V(R,A) = 500

  select * from R where R.A=761;

- Cost(no-index) = 1000
- Cost(non-clustering index) = 10
- Cost(clustering index) = 2
- Size of the result in pages = 2

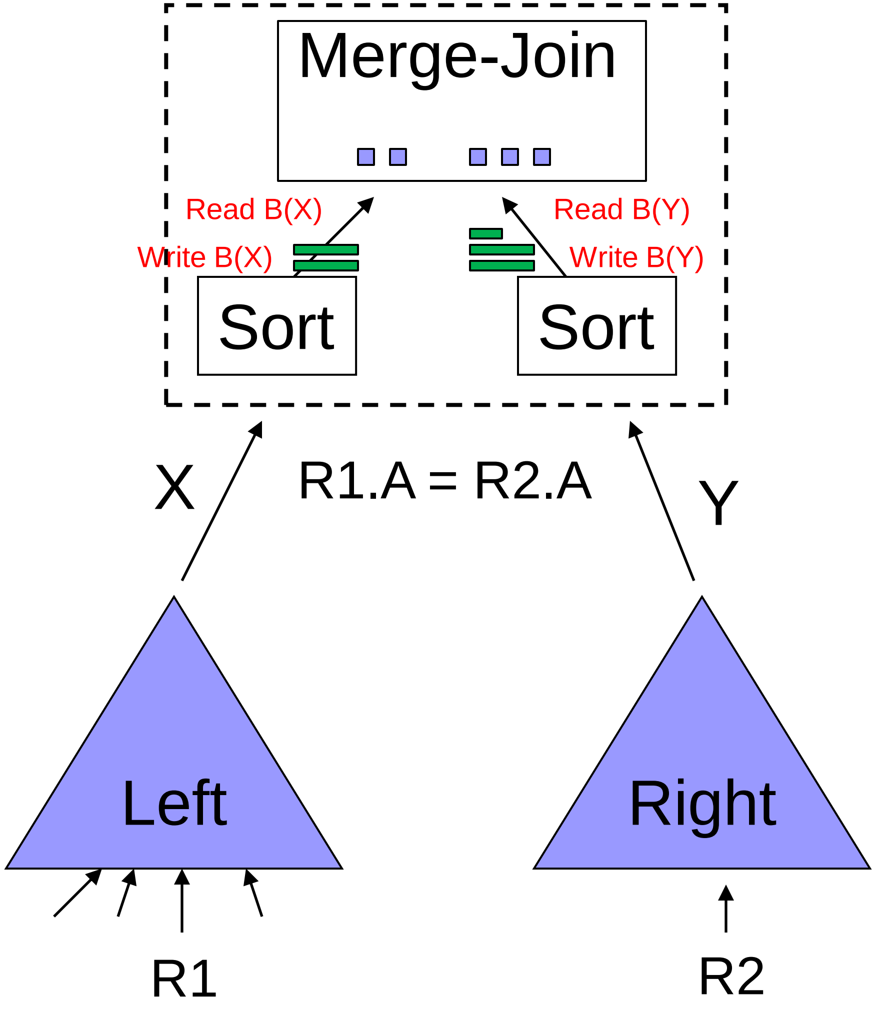
Κόστος NLJ



Cost for **entire** plan:

$$\text{Cost (Outer)} + \lceil B(X)/(M-1) \rceil \times \text{Cost (Inner)}$$

Κόστος Sort-Merge Join

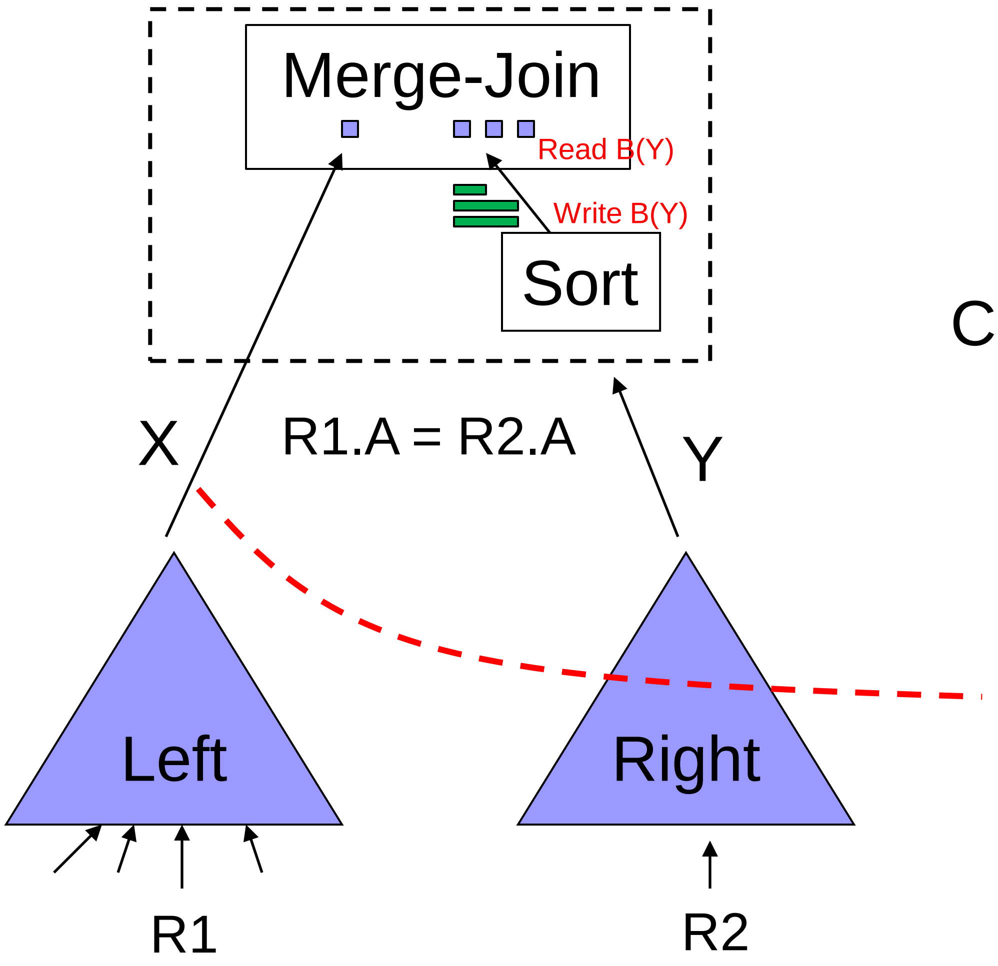


Cost for **entire** plan:

Cost (Left) + Cost (Right) + 2*(B(X) + B(Y))

Εφόσον B(X)+B(Y) ≤ M^2

Κόστος Sort-Merge Join

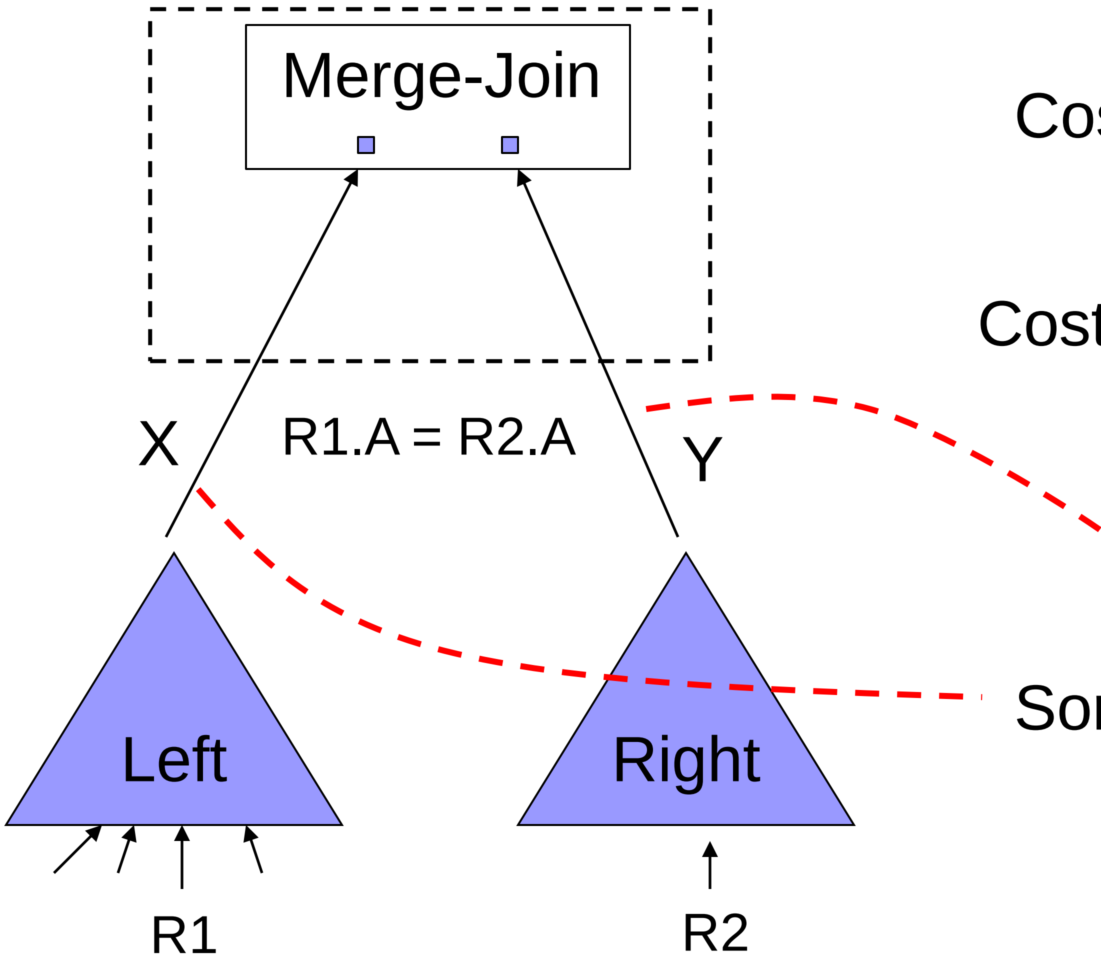


Cost for **entire** plan:

Cost (Left) + Cost (Right) + 2*B(Y)

Sorted on R1.A

Κόστος Sort-Merge Join



Cost for **entire** plan:

Cost (Left) + Cost (Right)

Sort-Merge Join

Το κόστος αλλάζει όταν η μία ή και οι δύο σχέσεις εισόδου είναι ταξινομημένες

Παράδειγμα

- Σχέσεις R(a,b) και S(b,c,d).
  - T(R)=3000, B(R)=30 σελίδες
  - T(S)=100, B(S)=50 σελίδες
  - M=11 σελίδες
- Επερώτηση:

  SELECT a,d FROM R, S WHERE R.b=S.b
- Κόστος SMJ, NLJ?

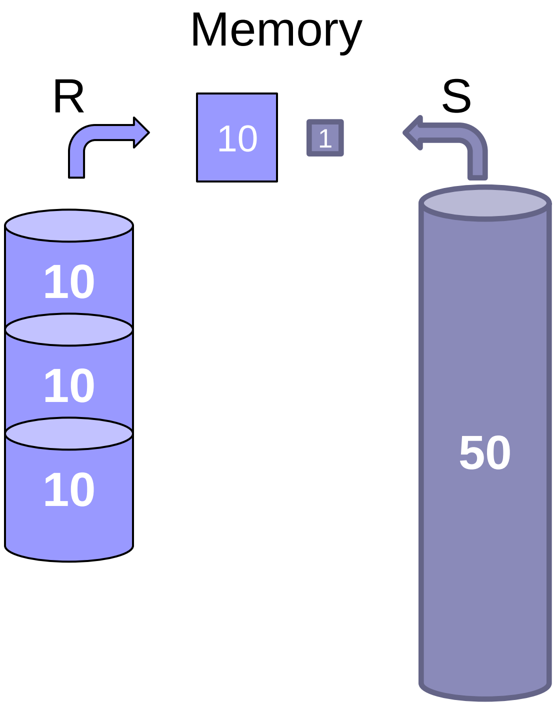
Ανάλυση SMJ

SELECT a,d FROM R, S WHERE R.b=S.b

- Παρατηρώ:
  - Κάθε μία σχέση είναι μεγαλύτερη από τη διαθέσιμη μνήμη (M=11)
  - B(R)+B(S) = 30+50 = 80 < 11^2
- Επομένως θα τρέξει η αποδοτική έκδοση του αλγορίθμου σε 2 περάσματα
- Κόστος SMJ= 3*(B(R)+B(S))=240 σελίδες

Ανάλυση NLJ

- NLJ: εξωτερική η R (γιατί?)
  - M=11
  - B(R)=30, B(S)=50
  - Πόσες φορές θα διαβάσω την εσωτερική σχέση S?
    - ⌈B(R)/(M-1)⌉=30/10=3
  - Κόστος NLJ = 30+3*50=180



Ίδιο πρόβλημα για M=8

- Σχέσεις R(a,b) και S(b,c,d).
  - T(R)=3000, B(R)=30 σελίδες
  - T(S)=100, B(S)=50 σελίδες
  - M=8 σελίδες
- Επερώτηση:

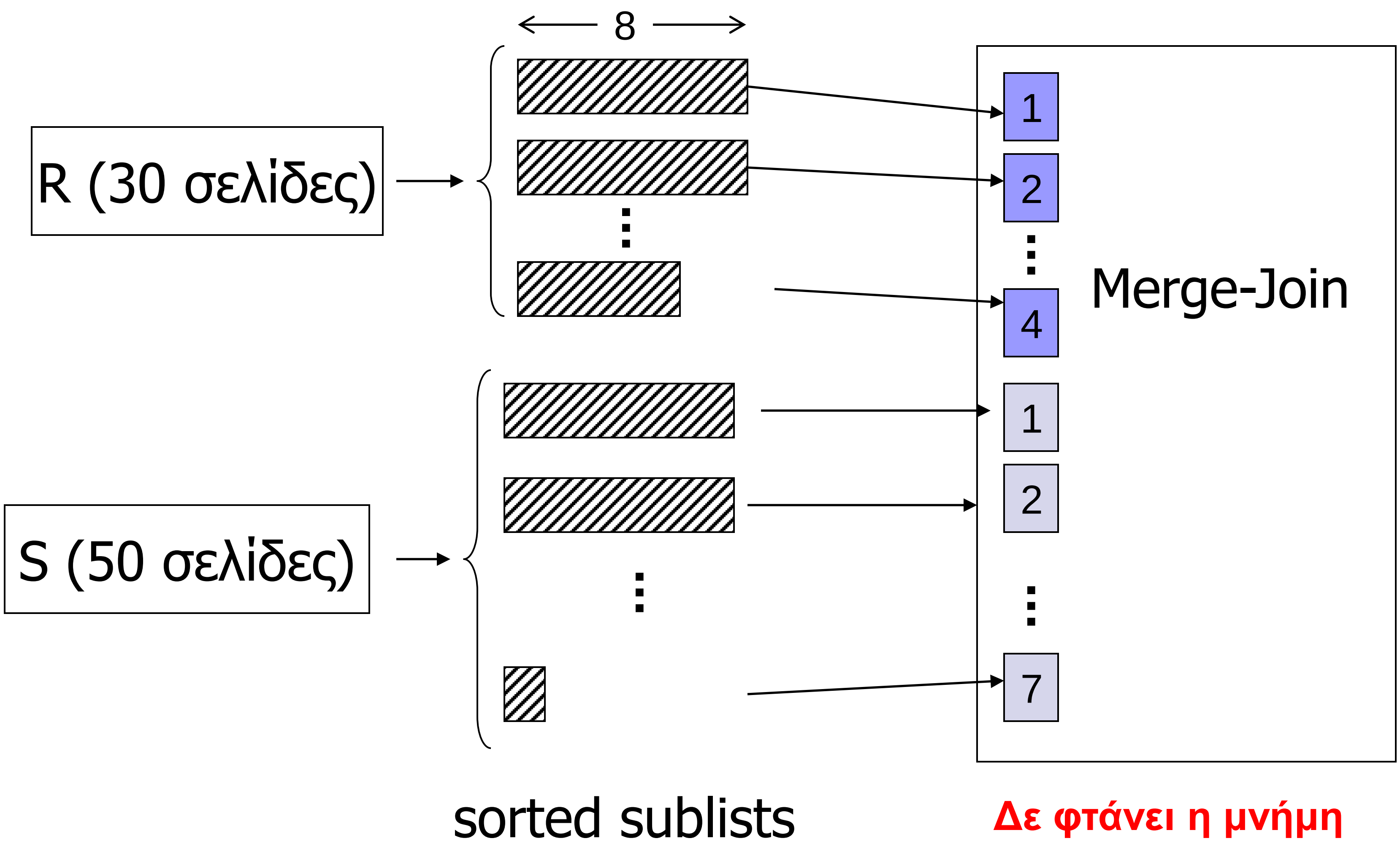
  SELECT a,d FROM R, S WHERE R.b=S.b
- Κόστος SMJ, NLJ?

Ανάλυση SMJ (M=8)

SELECT a,d FROM R, S WHERE R.b=S.b

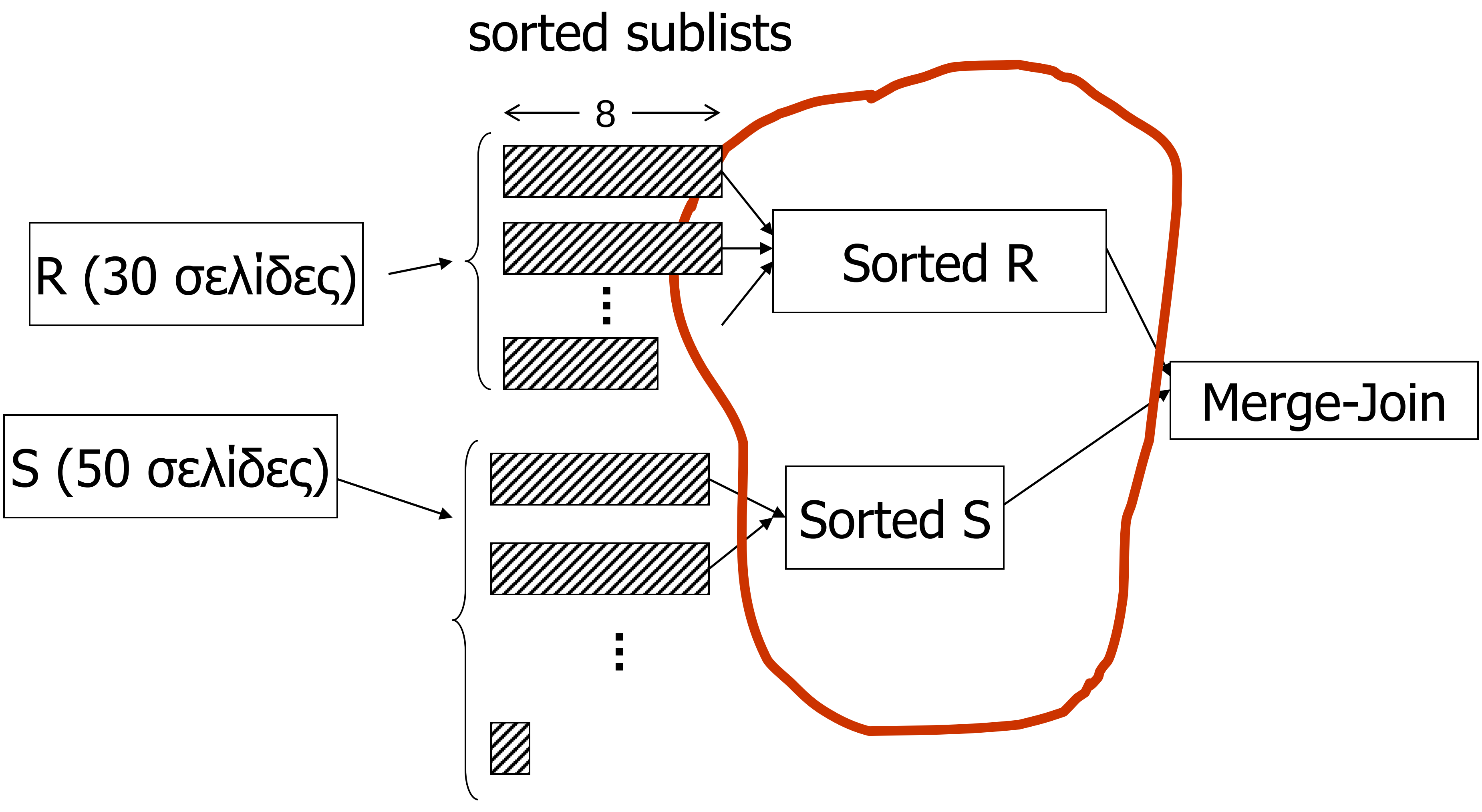
- B(R) > M , B(S) > M
  - Άρα τουλάχιστον 2 περάσματα
- B(R)+B(S) = 80 > M^2 = 64
  - Άρα δε μπορεί να εφαρμοστεί η αποδοτική έκδοση του αλγορίθμου
- Όμως B(R)=30 < 64, B(S)=50 < 64
  - Άρα μπορεί να εκτελεστεί η μη-αποδοτική έκδοση σε δύο περάσματα

Αποδοτική έκδοση του αλγορίθμου Sort-Merge Join (M=8)?



Δε φτάνει η μνήμη (θέλω 4+7=11 input buffers)

Εναλλακτική εκτέλεση SMJ



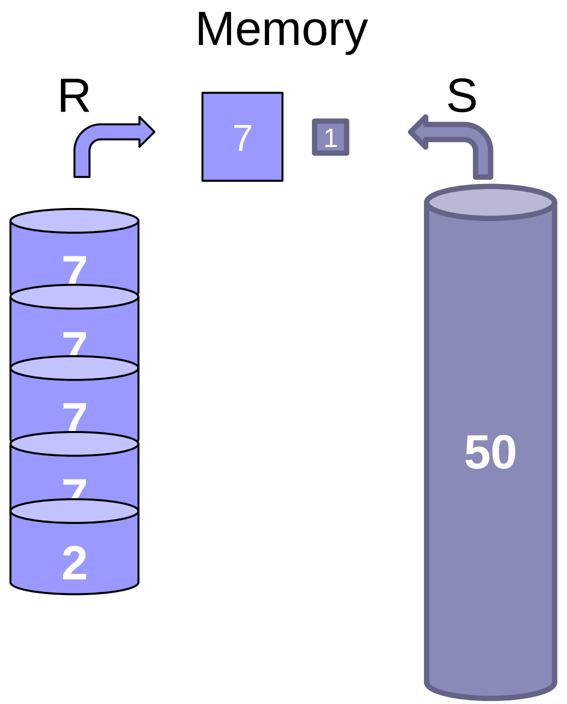
Για M=8 θα δημιουργήσουμε τις ταξινομημένες R, S στο δίσκο και μετά θα γίνουν merge-join

Ανάλυση SMJ για M=8

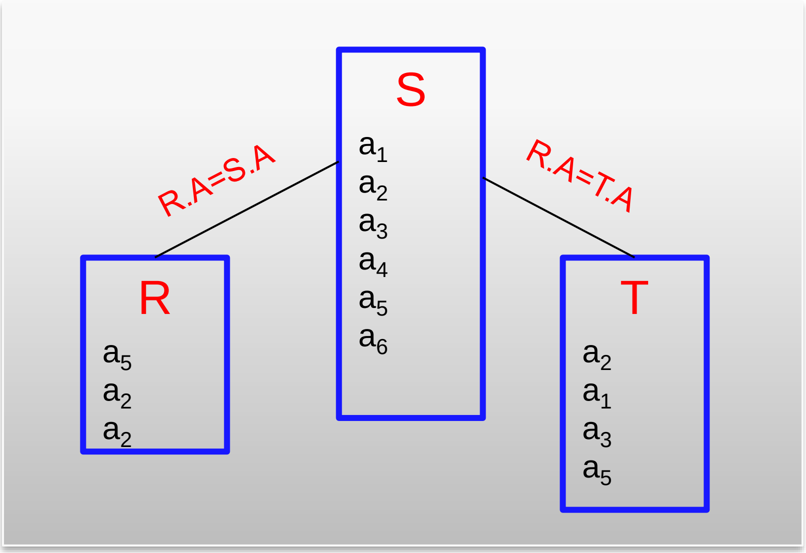
- B(R) = 30 > 8, B(S) = 30 > 8
- B(R) = 30 < 64, B(S) = 50 < 64
- Κόστος SMJ = 5*(30+50)=400

Ανάλυση NLJ

- NLJ: εξωτερική η R (γιατί?)
  - M=8
  - $\lceil B(R)/(M-1) \rceil = \lceil 30/7 \rceil = 5$
  - Κόστος NLJ = 30+5*50=280



Άσκηση 2

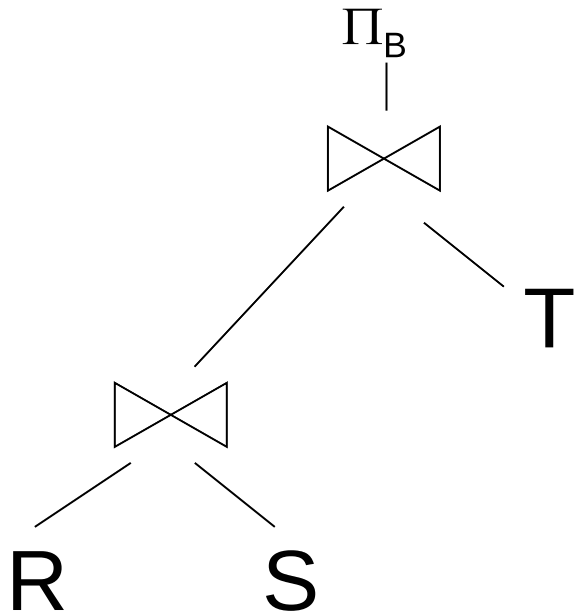


- Consider the following SQL query

select R.B
from R,S,T
where R.A=S.A and S.A=T.A

Schema: R(A,B), S(A,C), T(A,D)

Δίνεται το παρακάτω λογικό πλάνο



select R.B
from R,S,T
where R.A=S.A and S.A=T.A

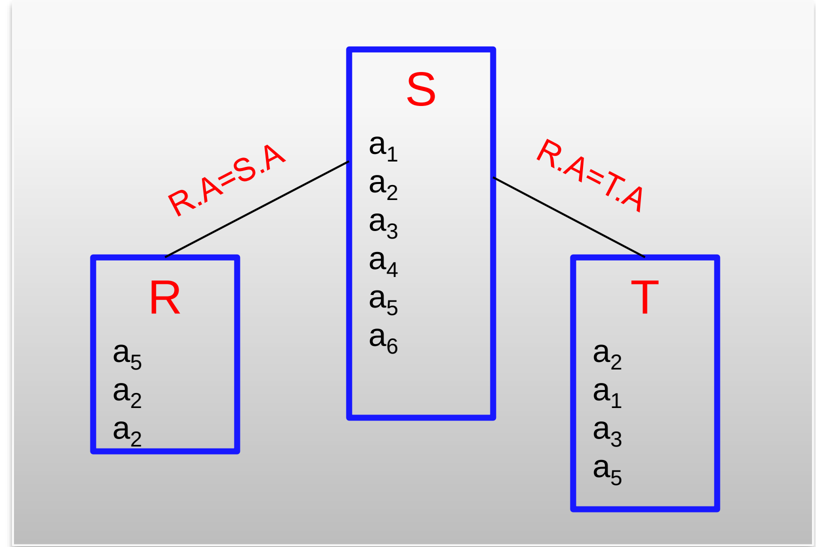
Στατιστικά

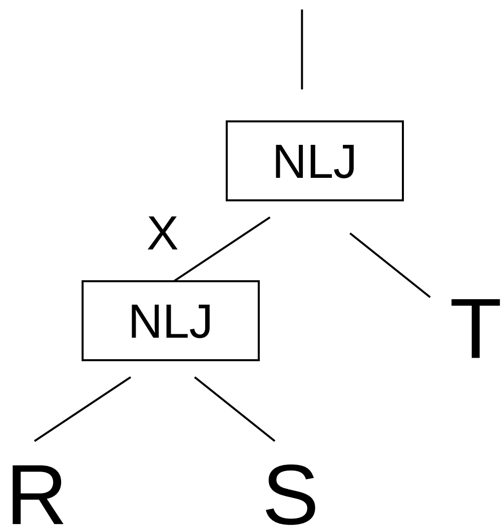
- B(R) = 10, B(S) = 20, B(T) = 16
- B(R JOIN S) = 19
- M = 14
- Υποθέστε ότι η διαθέσιμη μνήμη ισομοιράζεται ανάμεσα στους φυσικούς τελεστές

Physical Plan P1

<u>Υπόθεση</u>: η μνήμη ισομοιράζεται ανάμεσα στα 2 NLJ στιγμιότυπα: M’=7 σελίδες το κάθε ένα

$Cost(P1)=Cost(X)+\lceil B(X)/(M'-1) \rceil * B(T)$





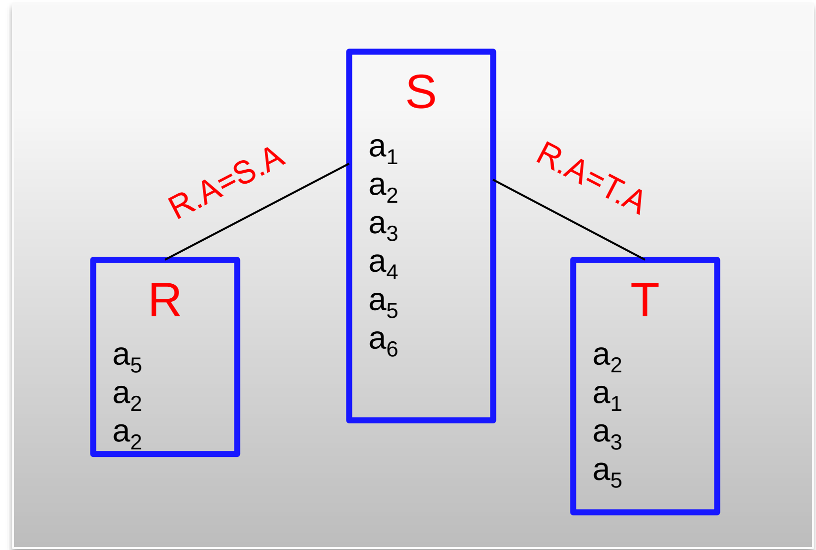
$Cost(X)=B(R)+\lceil B(R)/(M'-1) \rceil * B(S)$
$=10+\lceil 10/6 \rceil * 20$
$=10+2*20=50$

Επομένως:

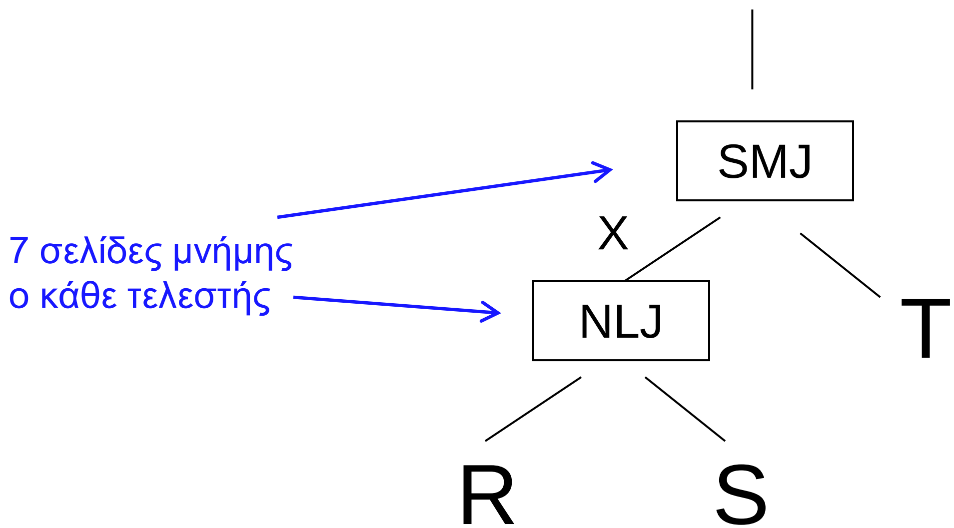
$Cost(P1)=Cost(X)+\lceil B(X)/(M'-1) \rceil * B(T)$
$=50+\lceil 19/6 \rceil * 16$
$= 50+4*16$
$= 114$ I/Os

B(R)=10, B(S)=20, B(T)=16, B(X)=19

Physical Plan P2



Cost(P2)=Cost(X)+2*B(X)+3*B(T)



B(R)=10, B(S)=20, B(T)=16, B(X)=19

Cost(X)=50 (προηγούμενη διαφάνεια)

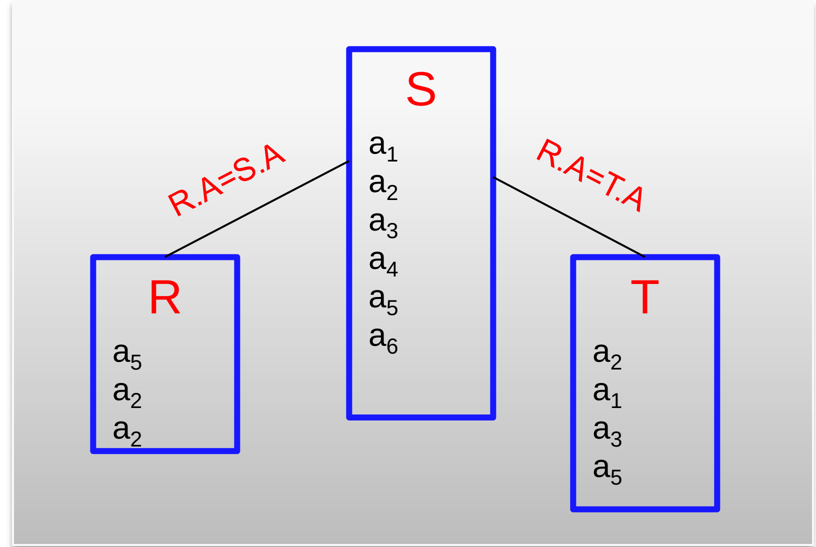
Επομένως:
Cost(P2)=50+2*19+3*16=136 I/Os

Προσοχή: υποθέσαμε ότι τρέχει η αποδοτική έκδοση του SMJ σε δύο περάσματα διότι:

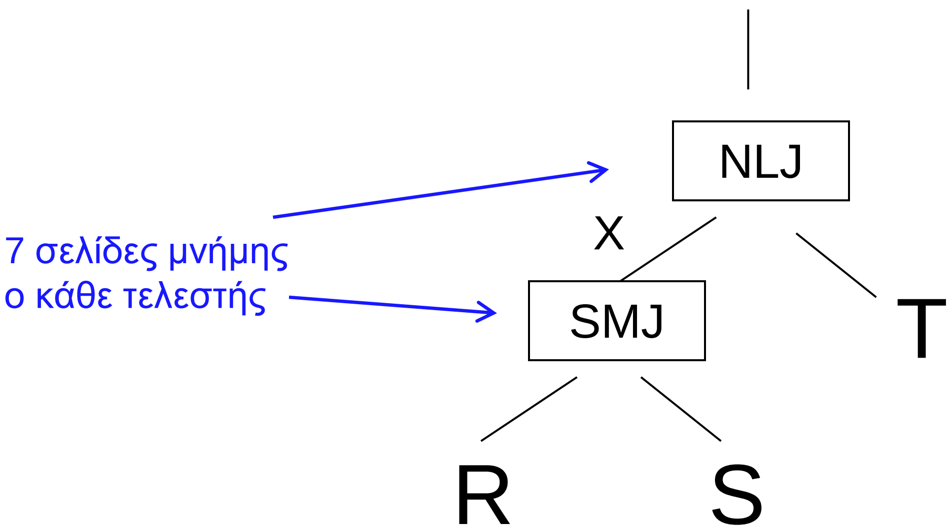
B(X) > 7, B(T) > 7, και
$B(X)+B(T) = 35 < 7^2$

(ισοδύναμα το X θα δημιουργήσει 3 sublists (7+7+5) και το T άλλα 3 (7+7+2), δηλαδή 6 συνολικά τα οποία μπορούν αν γίνουν merge μαζί στο δεύτερο βήμα)

Physical Plan P3



$Cost(P3)=Cost(X)+\lceil B(X)/(M'-1)\rceil * B(T)$



B(R)=10, B(S)=20, B(T)=16, B(X)=19

Cost(X)=3*(B(R)+B(S))=3*30=90

- Αποδοτική έκδοση SMJ διότι B(R)=10>7, B(S)=20>7 και $10+20<7^2$
- Ισοδύναμα το R θα δημιουργήσει 2 sublists (7+3) και το S 3 (7+7+6), δηλαδή 6 συνολικά τα οποία μπορούν αν γίνουν merge μαζί στο δεύτερο βήμα M'=7

Επομένως:

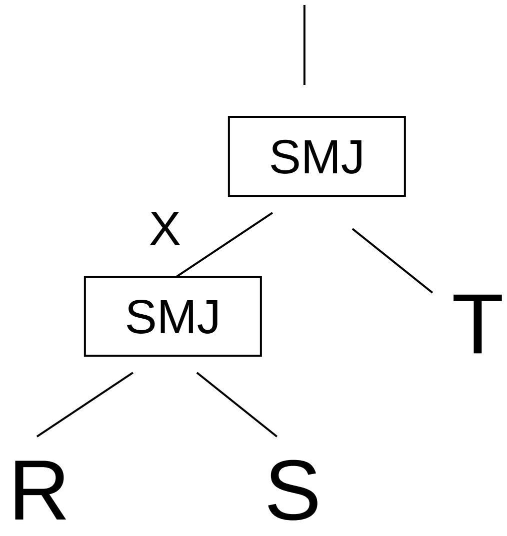
$Cost(P3) = Cost(X)+\lceil B(X)/(M'-1)\rceil * B(T)$

$= 90+\lceil 19/6 \rceil * 16$

$= 90+4*16 = 154$ I/Os

Physical Plan P4

Cost(P3)=Cost(X)+3*B(T) *ΠΡΟΣΟΧΗ!*



Cost(X)=3*(10+20)=90
(βλ. προηγούμενη διαφάνεια)

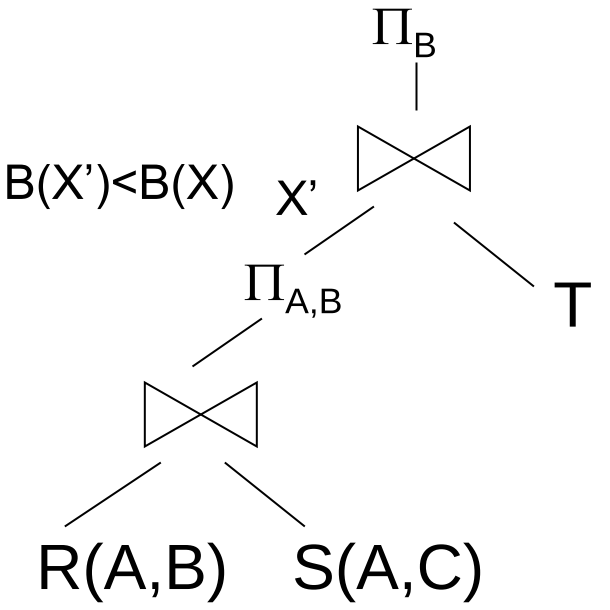
Επομένως:
Cost(P3)=90+3*16=138 I/Os
διότι 7 < B(T)=16 < 7^2
(η T θα ταξινομηθεί τμηματικά σε 3 sublists (7+7+2) θα οποία θα γίνουν merge-join με την ήδη ταξινομημένη X)

B(R)=10, B(S)=20, B(T)=16, B(X)=19

There are more plans…

- Plans that use indexes (INLJ)
- Change the order of the joins!!!
  - S JOIN R JOIN T
  - T JOIN R JOIN S
  - …

Καλύτερο λογικό πλάνο;



select R.B
from R,S,T
where R.A=S.A and S.A=T.A

Query Optimization Problem

Pick the best plan from the space of physical plans

Cost-Based Optimization

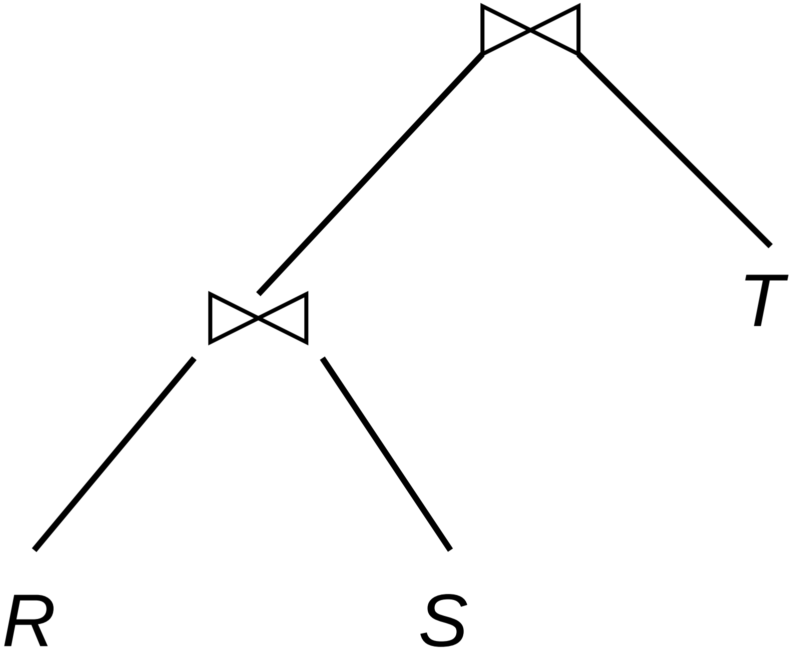
- Prune the space of plans using heuristics
- Estimate cost for remaining plans
- Pick the plan with least cost

Focus on queries with joins

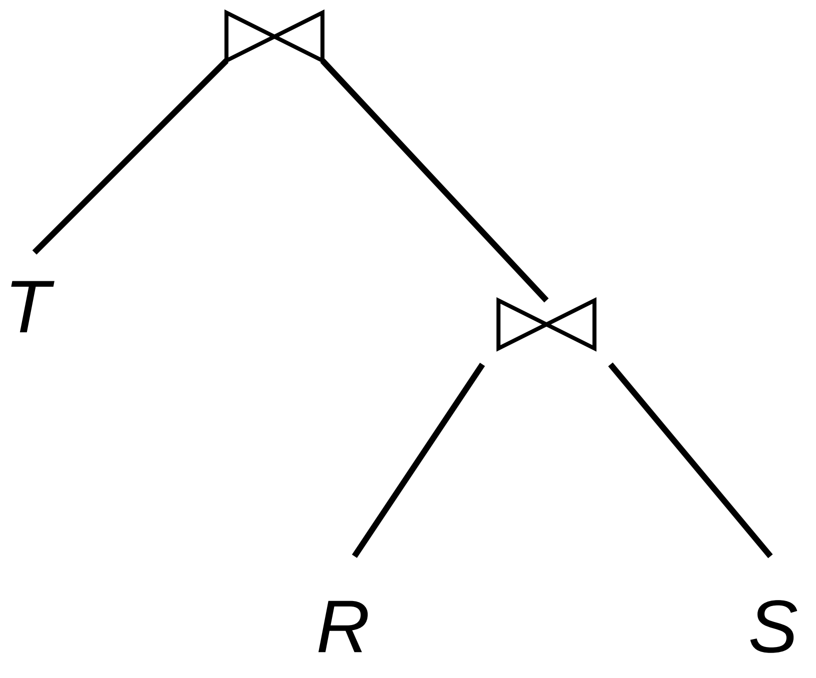
Heuristics for pruning plan space

- Predicates as early as possible
- Avoid plans with cross products
- Only left-deep join trees

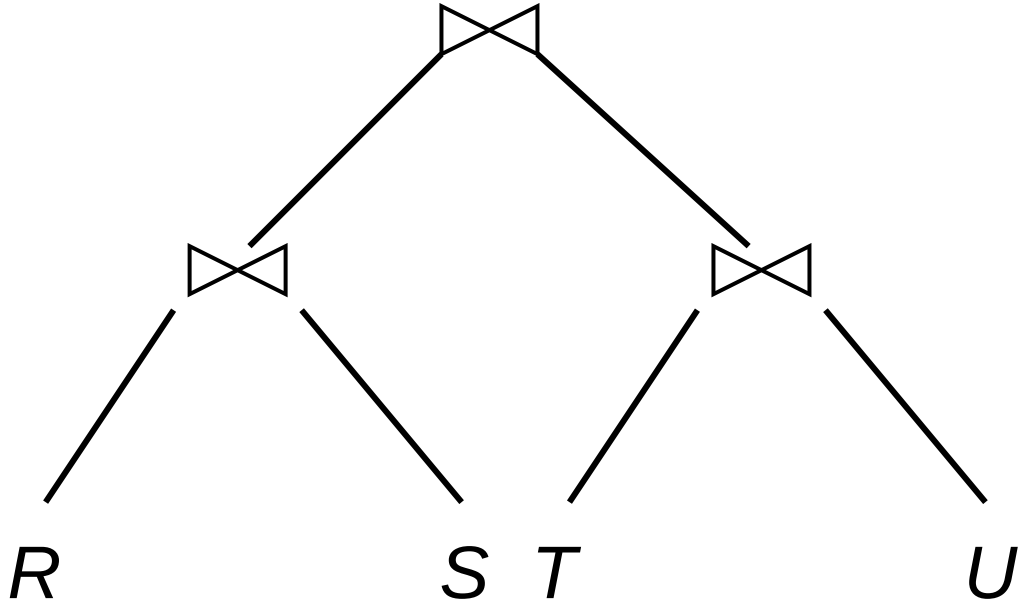
Left-deep Tree



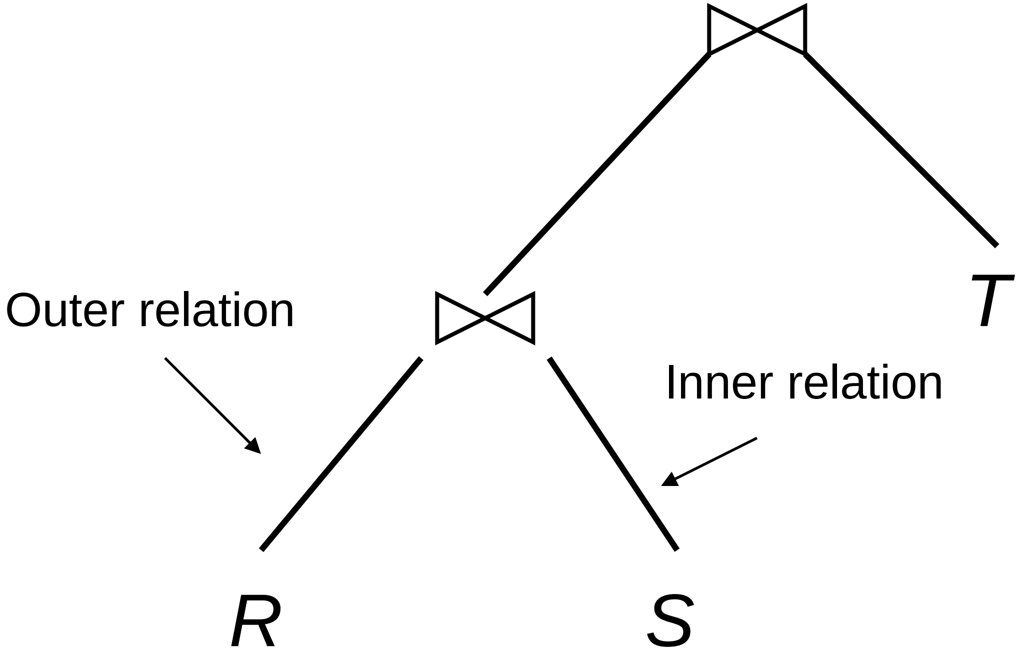
Right-deep Tree



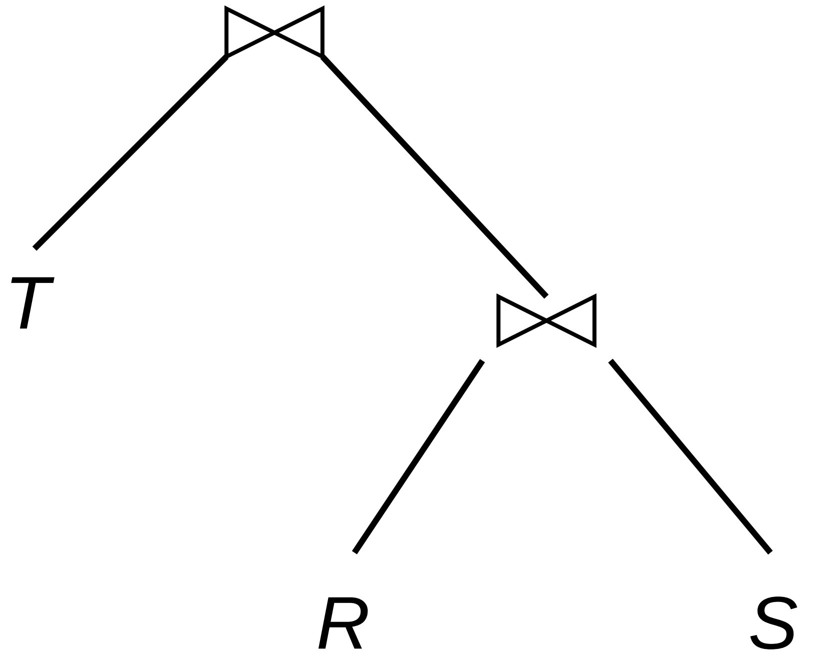
Bushy Tree



Consider left-deep tree & our join algorithms



And now?



Selinger Algorithm

- *Dynamic Programming* based
  - General algorithmic paradigm
  - Exploits "principle of optimality"

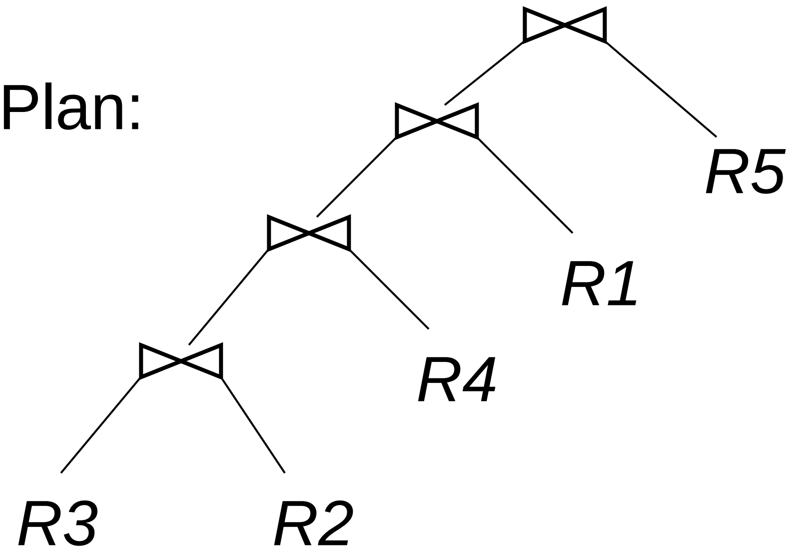
Principle of Optimality

Optimal for “whole” made up from optimal for “parts”

Principle of Optimality

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$

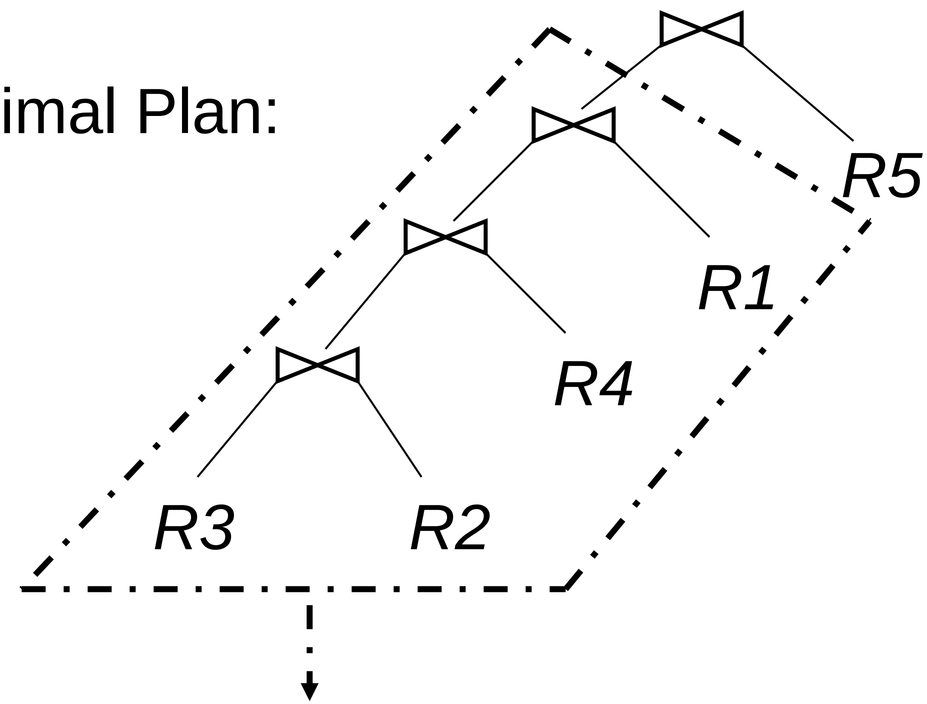
Optimal Plan:



Principle of Optimality

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$

Optimal Plan:

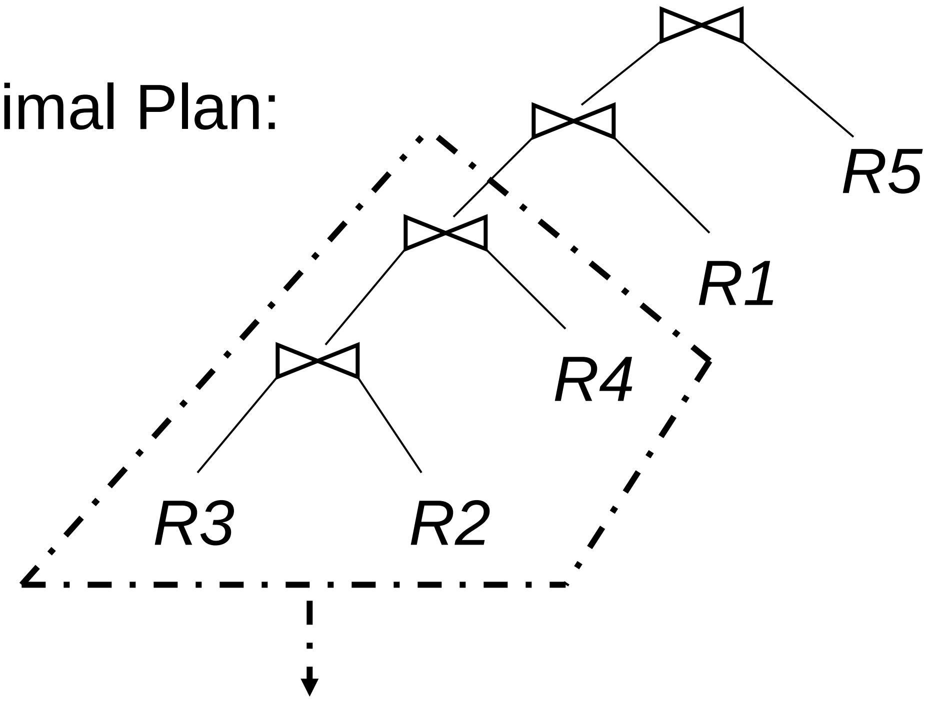


Optimal plan for joining *R3, R2, R4, R1*

Principle of Optimality

Query: *R1* ⋈ *R2* ⋈ *R3* ⋈ *R4* ⋈ *R5*

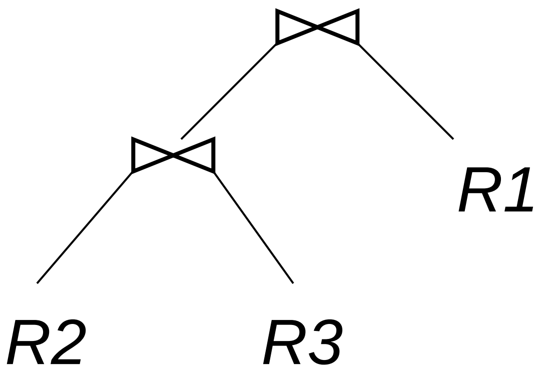
Optimal Plan:



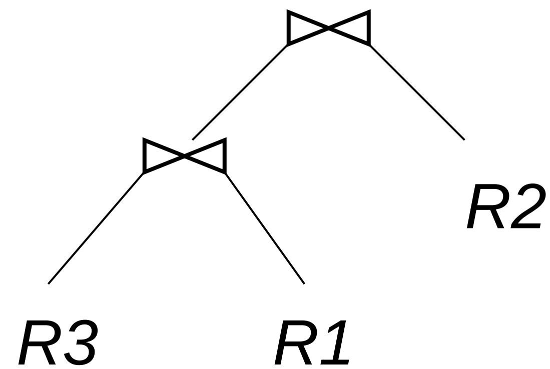
Optimal plan for joining *R3, R2, R4*

Exploiting Principle of Optimality

Query: $R1 \bowtie R2 \bowtie \ldots \bowtie Rn$

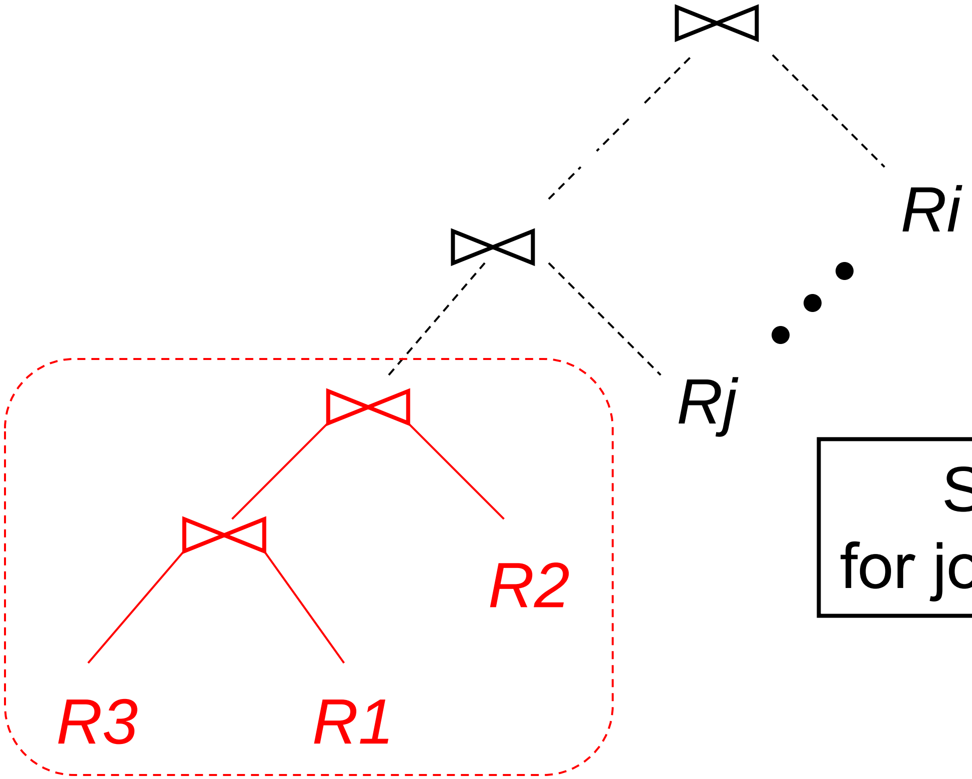


Optimal
for joining *R1, R2, R3*



Sub-Optimal
for joining *R1, R2, R3*

Exploiting Principle of Optimality



Sub-Optimal
for joining R1,…,Rn

Remove (n-3)! plans from consideration

Notation

OPT ({ *R1, R2, R3* }):

Cost of optimal plan to join *R1,R2,R3*

COST({X,Y})=Cost of joining X and Y

Selinger Algorithm:

OPT ({ R1, R2, R3 }):

(R1 $\bowtie$ R2) $\bowtie$ R3

$$\text{Min} \begin{cases} \text{OPT (\{ R1, R2 \})} + \text{COST (\{\{ R1,R2 \},R3\})} \\ \text{OPT (\{ R1, R3 \})} + \text{COST (\{\{ R1,R3 \},R2\})} \\ \text{OPT (\{ R2, R3 \})} + \text{COST (\{\{ R2,R3\},R1\})} \end{cases}$$

Selinger Algorithm:

OPT ({ R1, R2, R3 }):

$(R1 \bowtie R2) \bowtie R3$

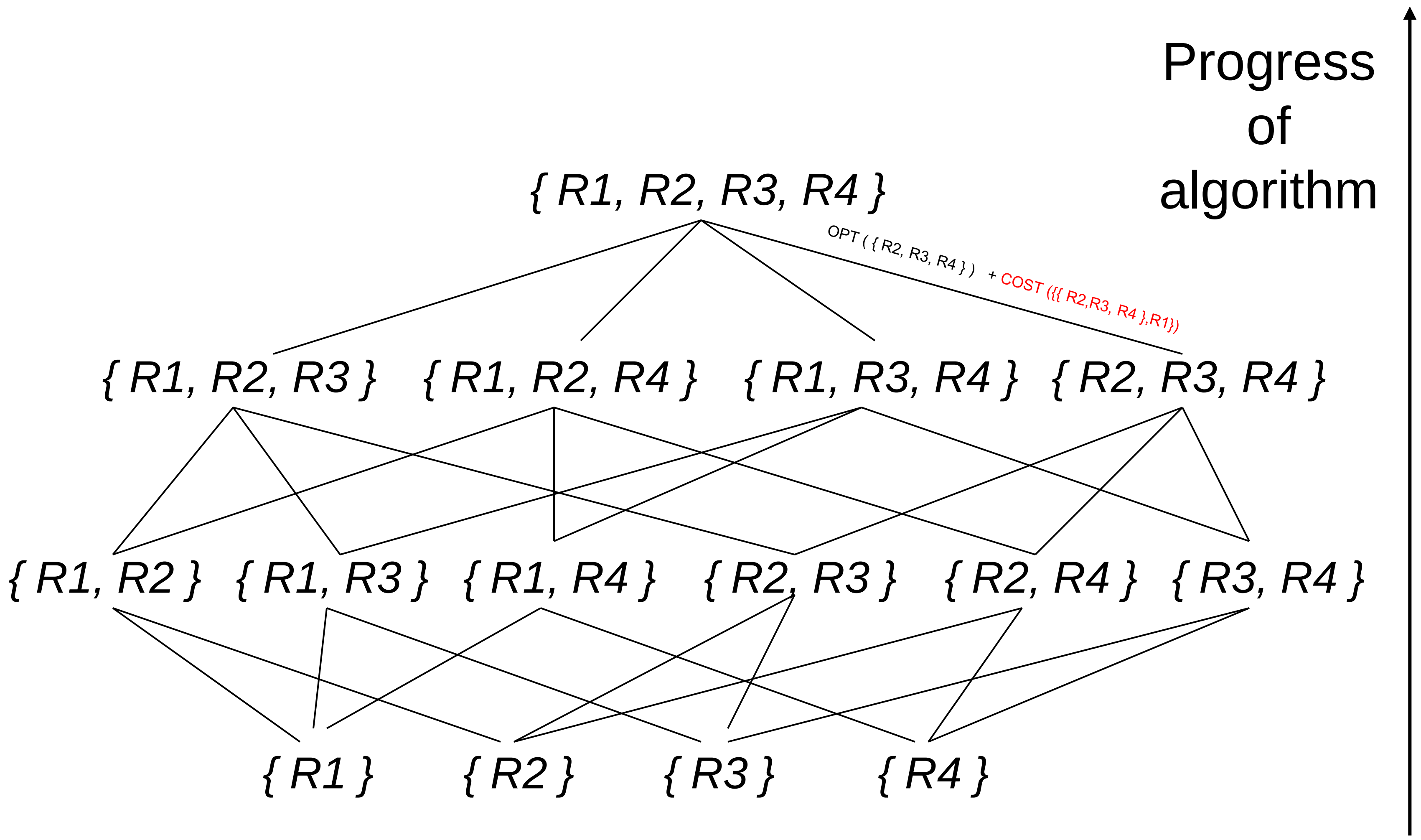
Min

- OPT ({ R1, R2 }) + COST ({{ R1,R2 },R3})
- OPT ({ R1, R3 }) + COST ({{ R1,R3 },R2})
- OPT ({ R2, R3 }) + COST ({{ R2,R3},R1})

Αναδρομικός υπολογισμός — Υπολογίζεται αναλυτικά

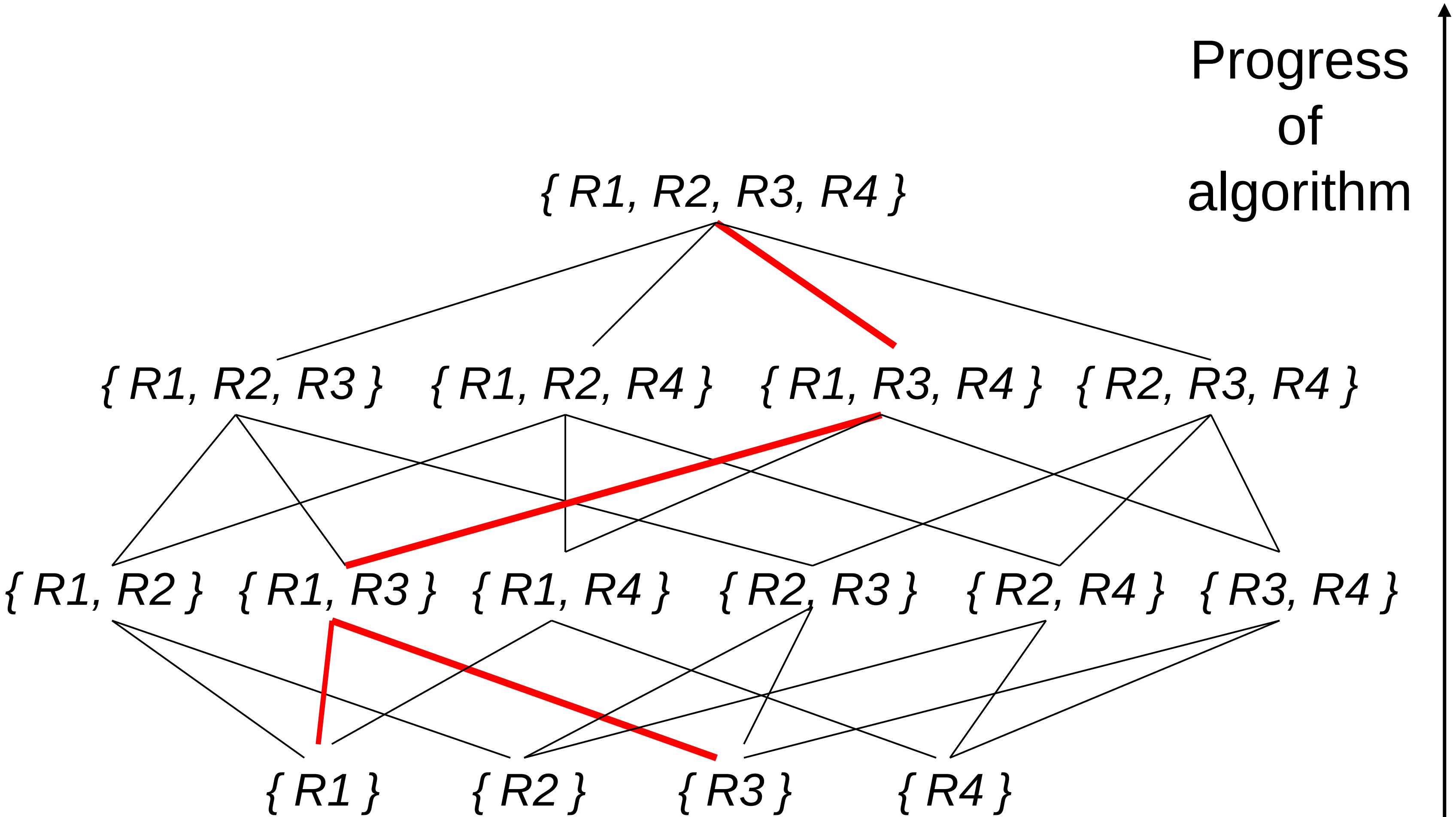
Selinger Algorithm:

Query: *R1* ⋈ *R2* ⋈ *R3* ⋈ *R4*



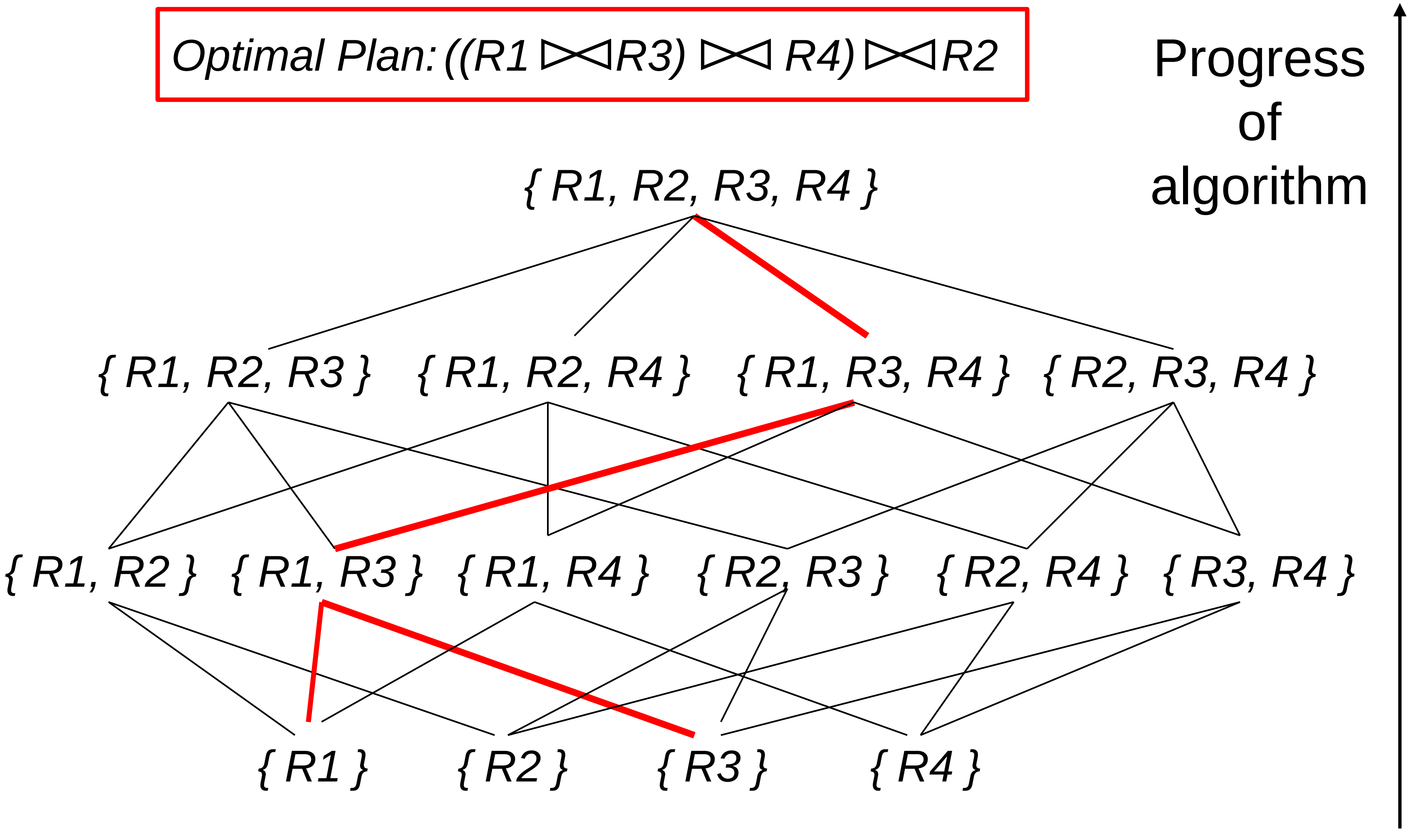
Selinger Algorithm:

Query: *R1* ⋈ *R2* ⋈ *R3* ⋈ *R4*



Selinger Algorithm:

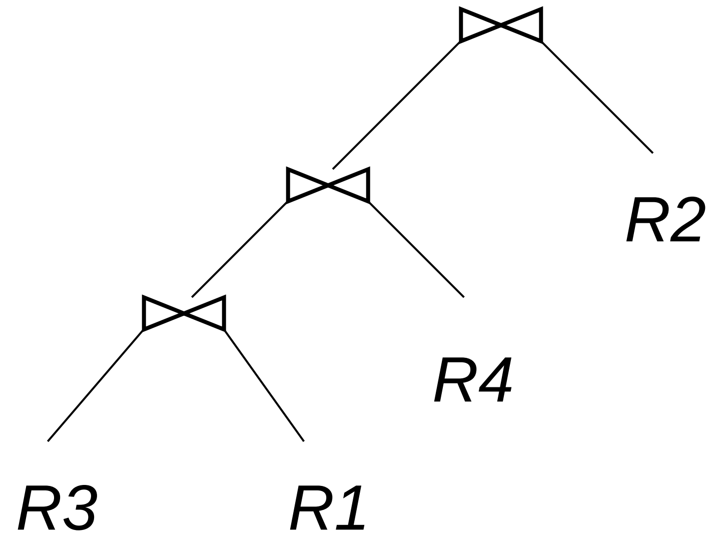
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

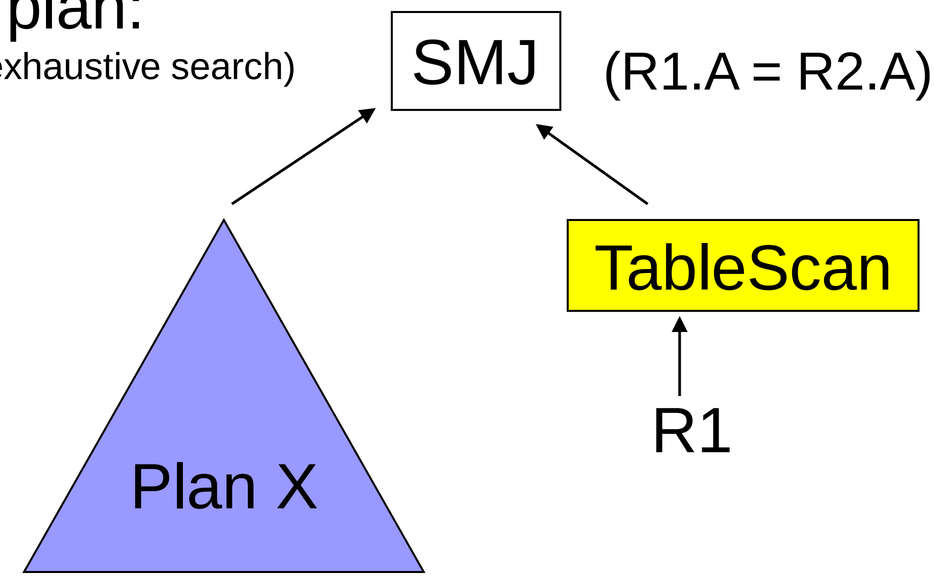
Optimal plan:



Principle of Optimality?

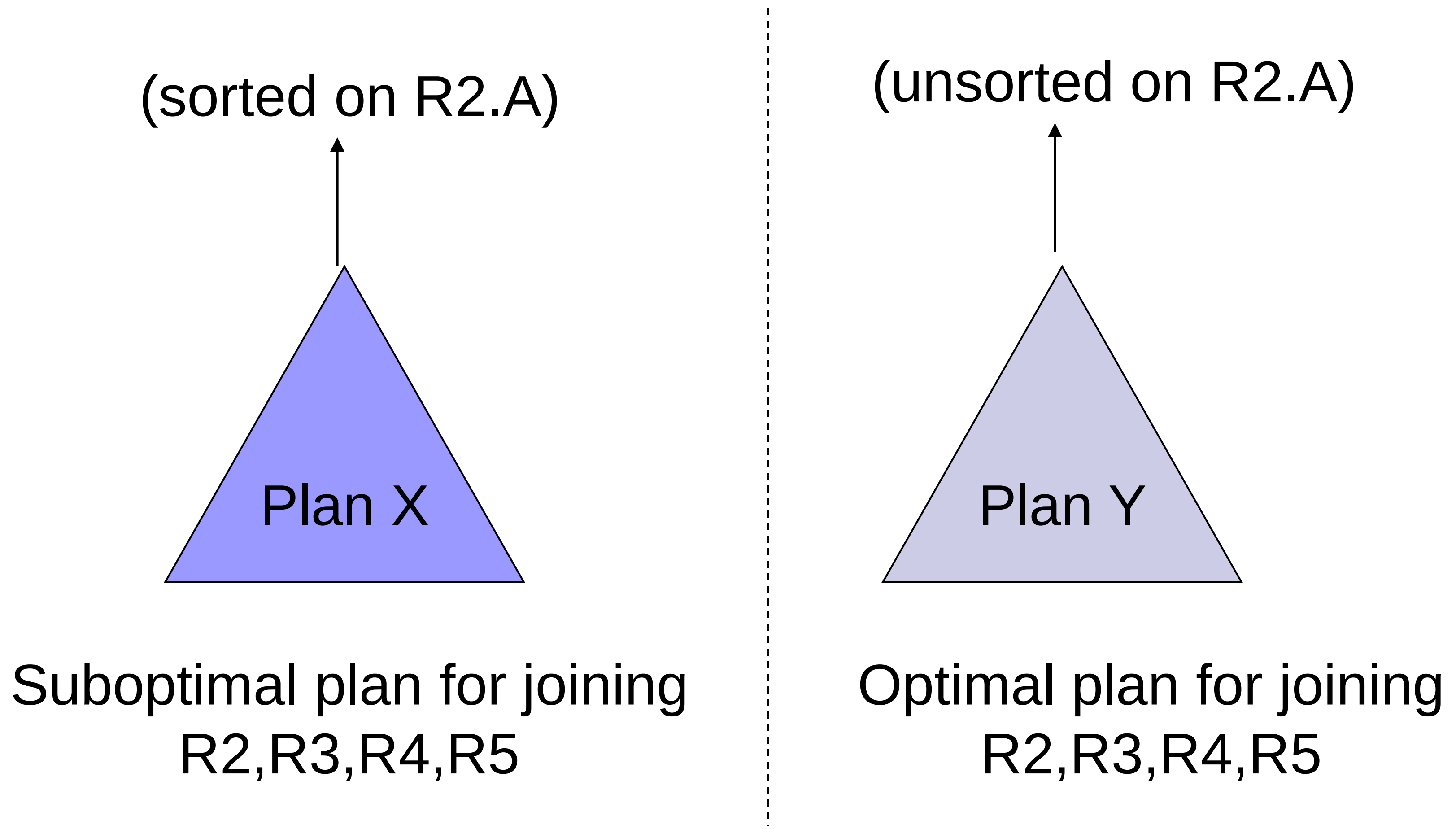
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$

Optimal plan:
(found using exhaustive search)



Is Plan X the optimal plan for joining R2,R3,R4,R5?

Violation of Principle of Optimality



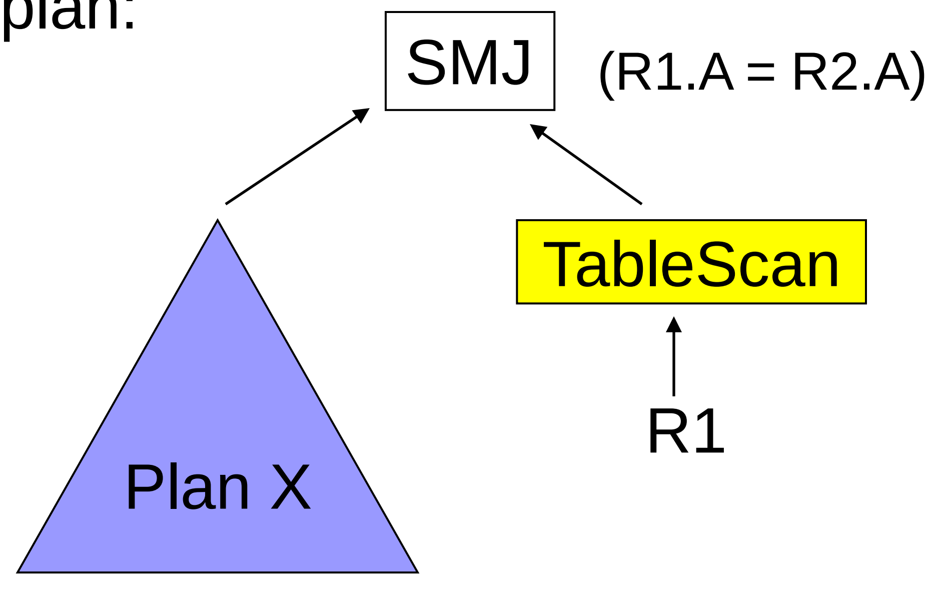
Consider

- Cost(X) = 100
- B(R1)=100
- Cost(Y) = B(Y) = 90 < Cost(X)

Case 1: Use Plan X

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$

Optimal plan:

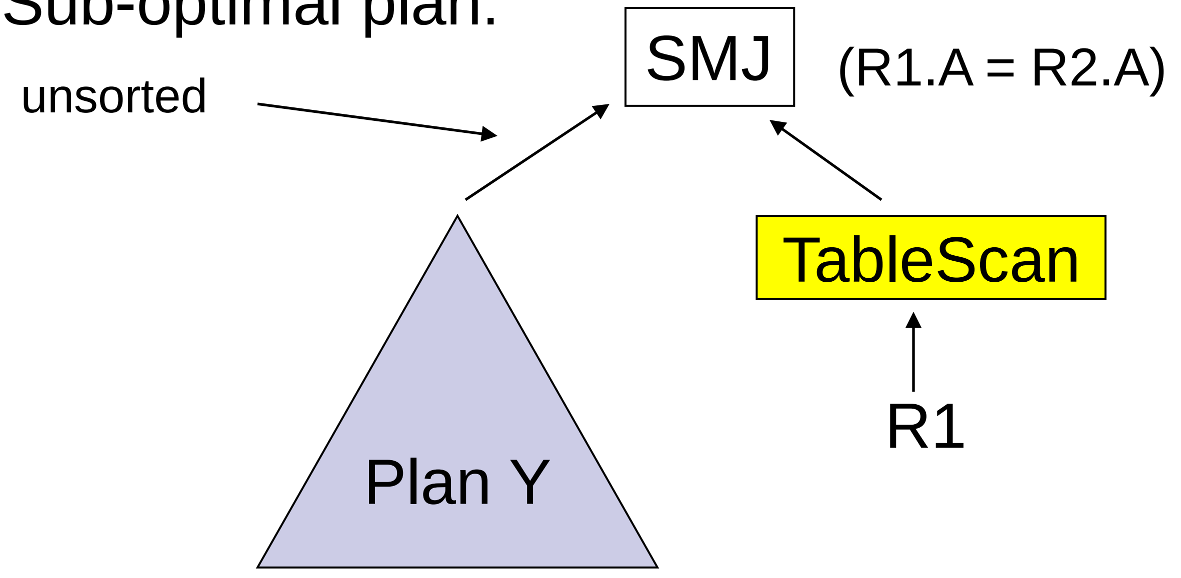


Total Cost = Cost(X)+Cost(R1)+2*B(R1)=100+100+200=400

Case 2: Use optimal plan Y

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$

Sub-optimal plan:

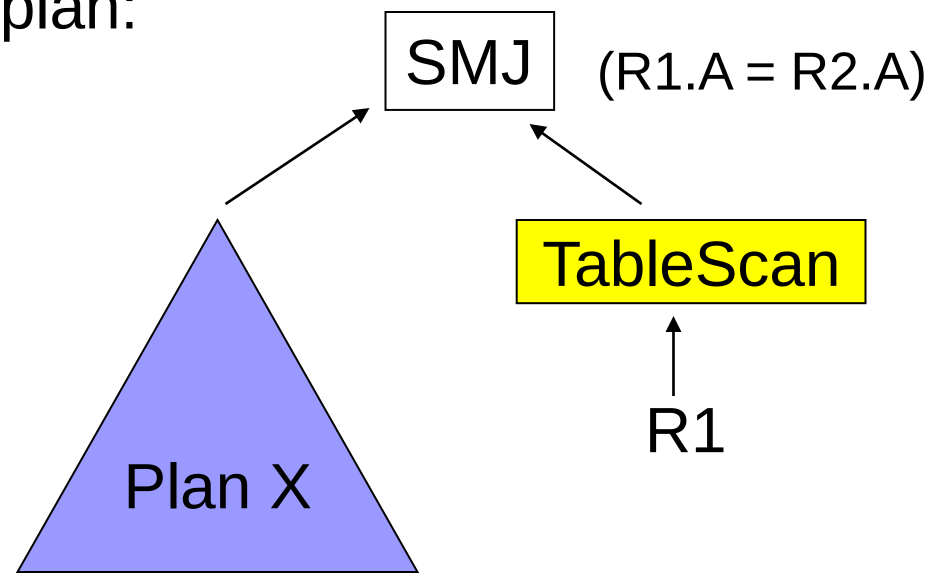


Total Cost = Cost(Y)+Cost(R1)+2*(**B(Y)**+B(R1))=
= 90+100+2*190=570

Principle of Optimality?

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$

Optimal plan:



Can we assert anything about plan X?

Weaker Principle of Optimality

If plan X produces output sorted on R2.A then plan X is the **optimal plan** for joining R2,R3,R4,R5 that produces output sorted on R2.A

If plan X produces output unsorted on R2.A then plan X is the **optimal plan** for joining R2, R3, R4, R5

Note

- This is a problem of us not stating the Dynamic Programming problem correctly
  - Answer depends of how individual parts have been solved
- Solution?
  - Make sorted-ness part of the "state"

Interesting Order

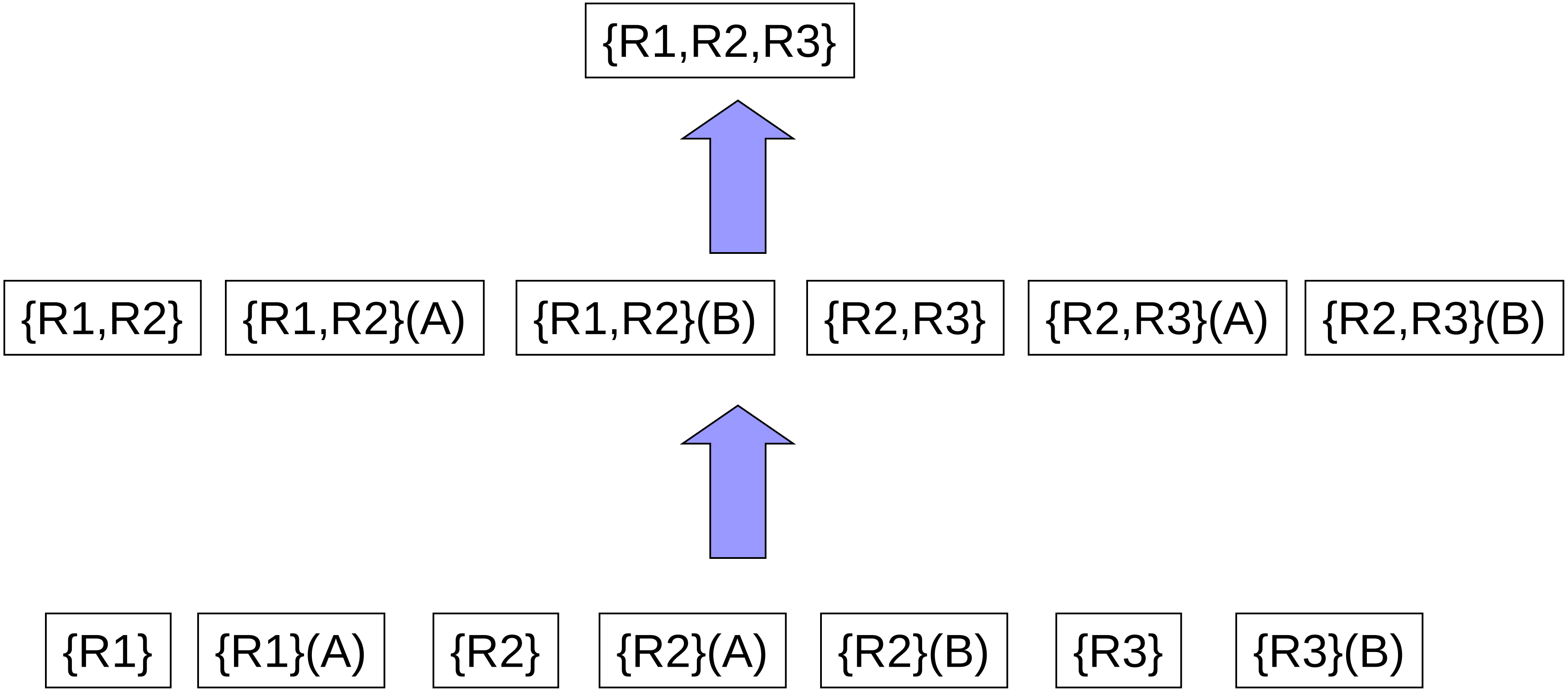
- An attribute is an **interesting order** if:
  - Participates in a join predicate
  - Occurs in the Group By clause
  - Occurs in the Order By clause

Interesting Order: Example

Select *
From R1(A,B), R2(A,B), R3(B,C)
Where R1.A = R2.A and R2.B = R3.B

Interesting Orders: R1.A, R2.A, R2.B, R3.B

Modified Selinger Algorithm



Notation

{R1,R2} (C)

Optimal way of joining R1, R2 so that output is sorted on attribute R2.C

Modified Selinger Algorithm

```
Select   *
From     R1(A,B), R2(A,B), R3(B,C)
Where    R1.A = R2.A and R2.B = R3.B
```