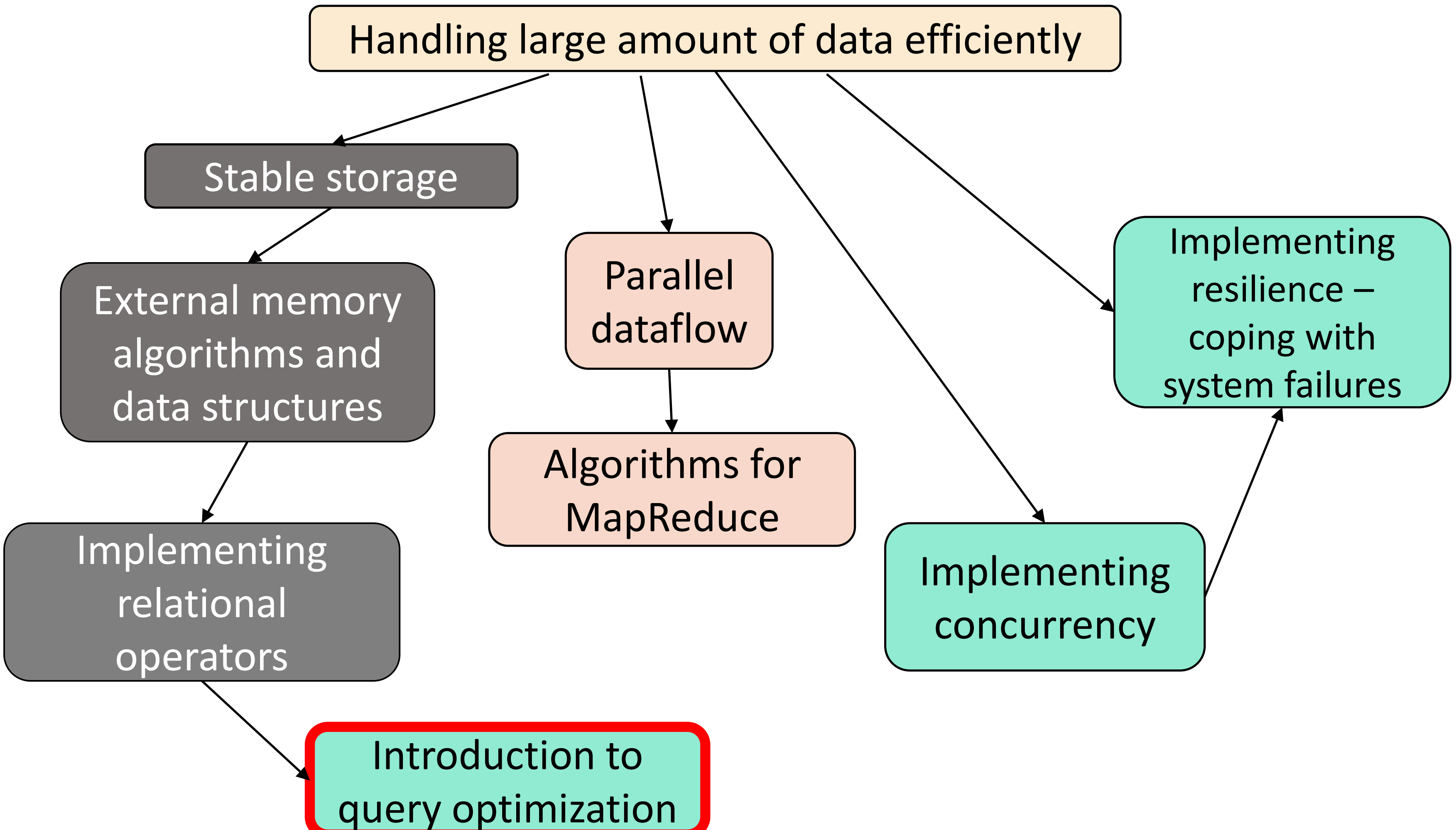
Roadmap


Handling large amount of data efficiently
Stable storage
External memory algorithms and data structures
Implementing relational operators
Introduction to query optimization
Parallel dataflow
Algorithms for MapReduce
Implementing concurrency
Implementing resilience – coping with system failures

Lecture 03.03

Query optimization

Part I. Logical query optimization

By Marina Barsky
Winter 2017, University of Toronto

Reminder:
Relational Algebra Operators

Core operators:

- Selection σ
- Projection π
- Cartesian product x
- Union $\cup$
- Difference –
- Renaming ρ

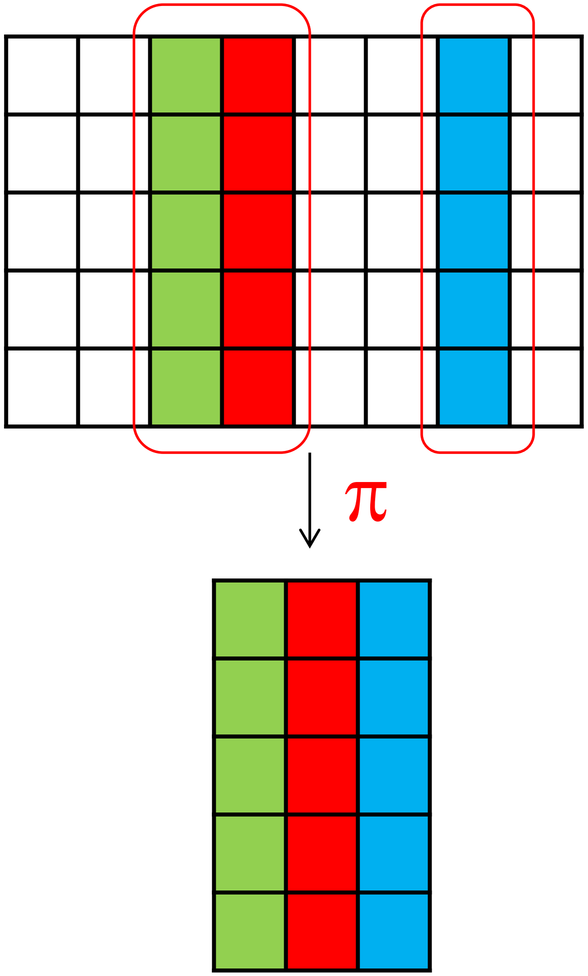
Derived operators:

- Join $\bowtie$
- Intersection $\cap$

Core RA operators

Slice operations: Projection

Produces from relation **R** a new relation that has only the A_1, …, A_n columns of **R**.

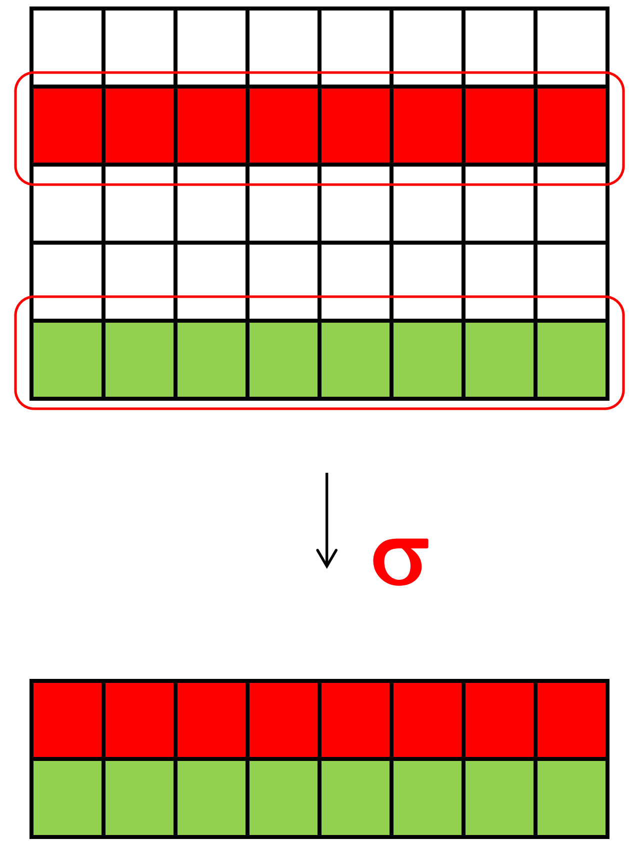


$S=\pi_{\text{attribute list}}(R)$

Slice operations: Selection

Produces a new relation with those tuples of **R** which satisfy condition **C**.

$$S=\sigma_{condition}(R)$$

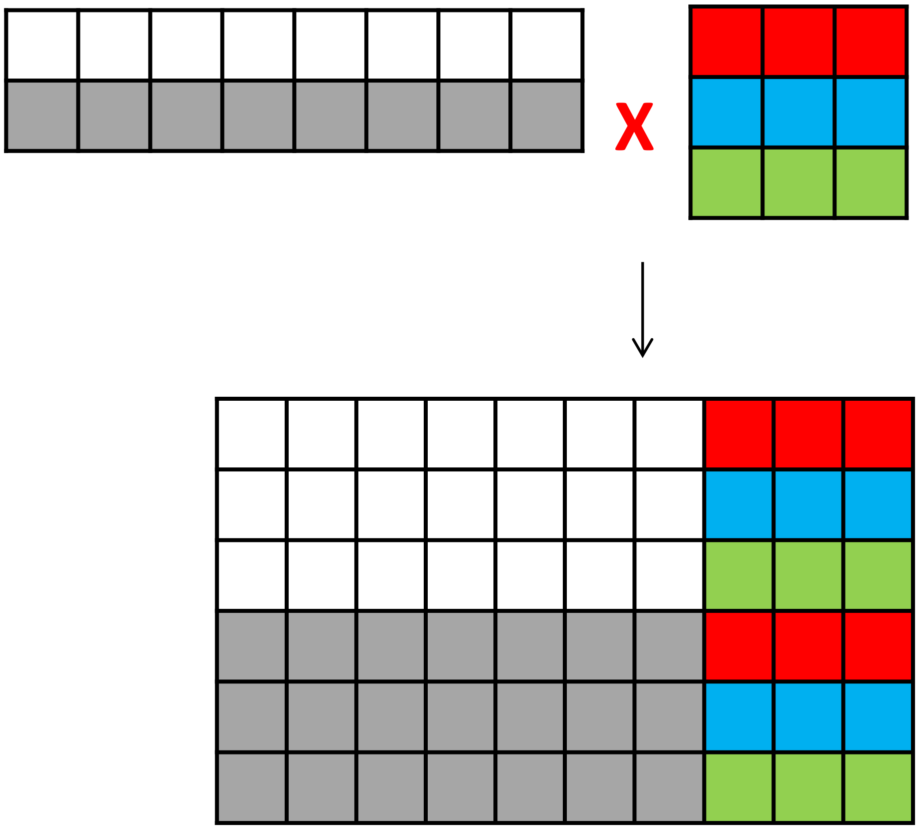


Join operation: Cartesian product

1. Set of tuples ***rs*** that are formed by choosing the first part (***r***) to be any tuple of **R** and the second part (***s***) to be any tuple of **S**.

2. Schema for the resulting relation is the union of schemas for **R** and **S**.

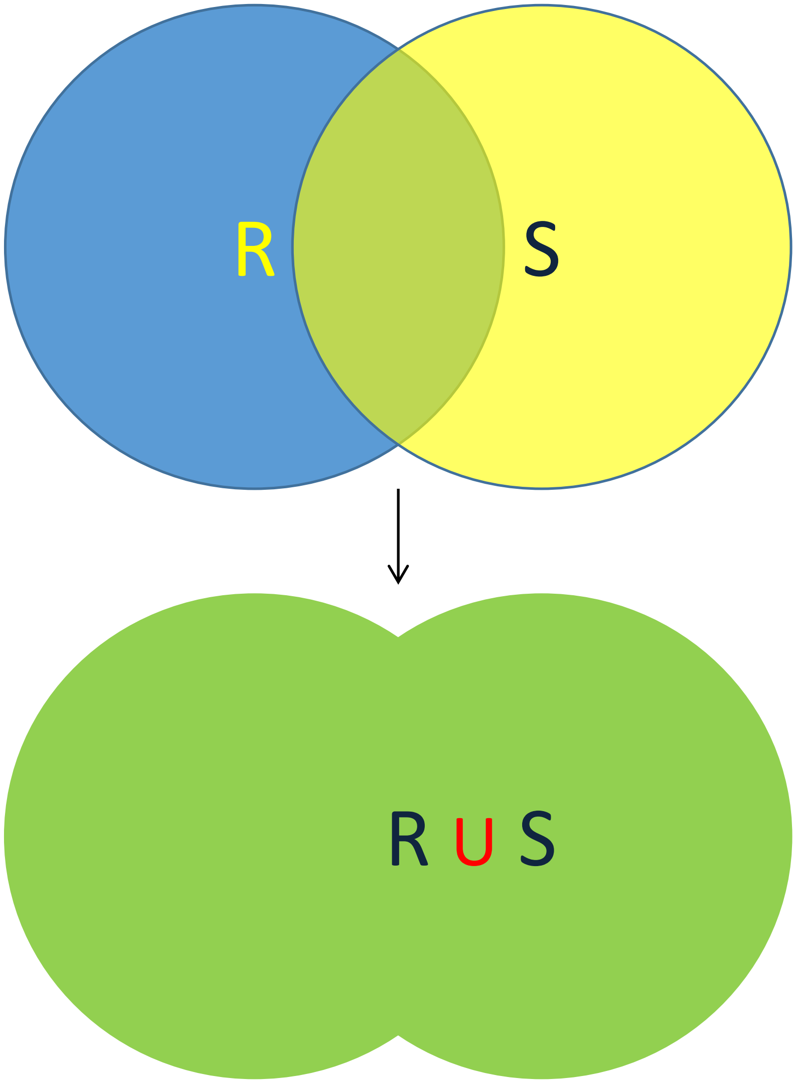
3. If **R** and **S** happen to have some attributes in common, then prefix those attributes by the relation name.



T=R x S

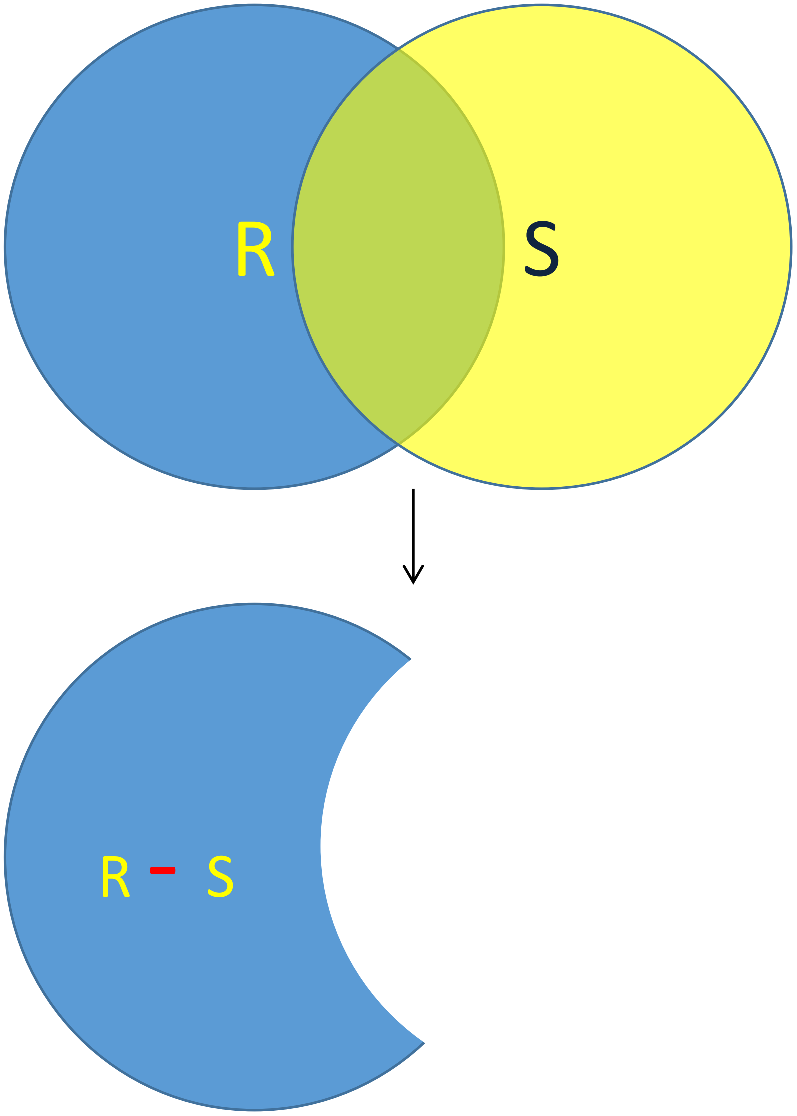
Union

T=R ∪ S



Difference

R – S



Renaming Operator

$$\rho_{S(A1,A2,\ldots,An)}\ (R)$$

1. Resulting relation has exactly the same tuples as **R**, but the name of the relation is **S**.
2. Moreover, the attributes of the resulting relation **S** are named A_1, A_2, …, A_n, in order from the left.

Query with renaming: example

T (node1, node2)

A → B
B → A
B → C
A → C
C → B

```
SELECT R.node1, R. node2
FROM T as R, T as S
WHERE R. node1 = S. node2
AND R. node2 = S. node1
```

- Find all reciprocally connected nodes in a directed graph
- By renaming T we created two identical relations R and S, and we now extract all tuples where for each pair X → Y in R there is a pair Y → X in S

$$\pi_{R.node1,\ R.node2}\ \sigma_{R.node1=S.node2\ AND\ R.node2\ =\ S.node1}(\rho_{R}\ (T)\ x\ \rho_{S}\ (T))$$

Core operators – sufficient to express any query in relational model

- Relational model due to Edgar "Ted" Codd, a mathematician at IBM in 1970
  - A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* **13** (6): 377–387

- He proved that any query can be expressed using these core operators: σ, π, x, U, –, ρ

The Relational model is **precise**, **implementable**, and we can operate on it (query/update, etc.)

Relational algebra: closure

Students(sid,sname,gpa)

```
SELECT DISTINCT
  sname,
  gpa
FROM Students
WHERE gpa > 3.5;
```

How do we represent this query in RA?

Note that any RA Operator returns relation, so we can compose complex queries from known operators

$$\pi_{sname,gpa}(\sigma_{gpa>3.5}(Students))$$

$$\Updownarrow$$

$$\sigma_{gpa>3.5}(\pi_{sname,gpa}(Students))$$

Are these logically equivalent?

RA has Limitations !

- Cannot compute “transitive closure”

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!!
  - Need to write C program, use a graph engine, or PL-SQL…

Derived RA operators

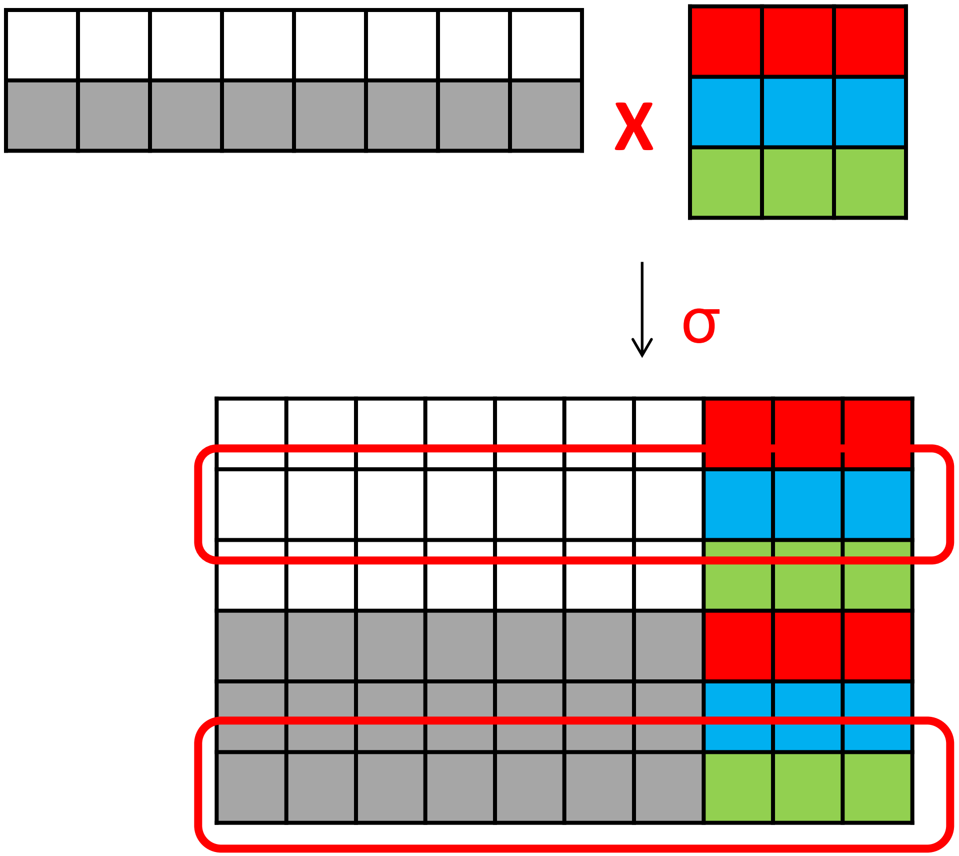
Join operation: Theta-join

1. The result of this operation is constructed as follows:

    a) Take the Cartesian product of **R** and **S**.

    b) Select from the product only those tuples that satisfy the condition **C**.

2. Schema for the result is the union of the schema of **R** and **S,** with **"R"** or **"S"** prefix as necessary.



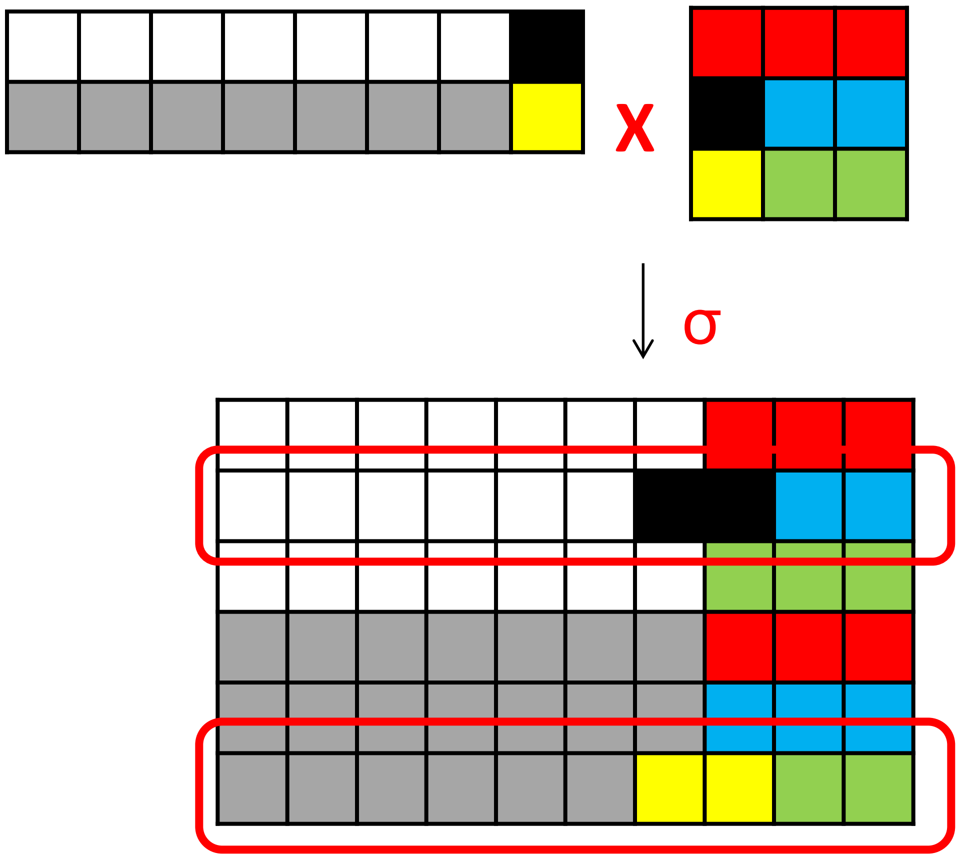
$T = R \bowtie_{condition} S$

Shortcut for

$T = \sigma_{condition} (R \times S)$

Join operation: Equijoin

1.Equijoin is a subset of theta-joins where the join condition is equality



T= R ⋈$_{R.A\ =\ S.B}$ S

Shortcut for

T=σ $_{R.A\ =\ S.B}$ (R x S)

Natural Join

Special case of equijoin when attributes we want to use in join have the same name in both tables

R ⋈ S

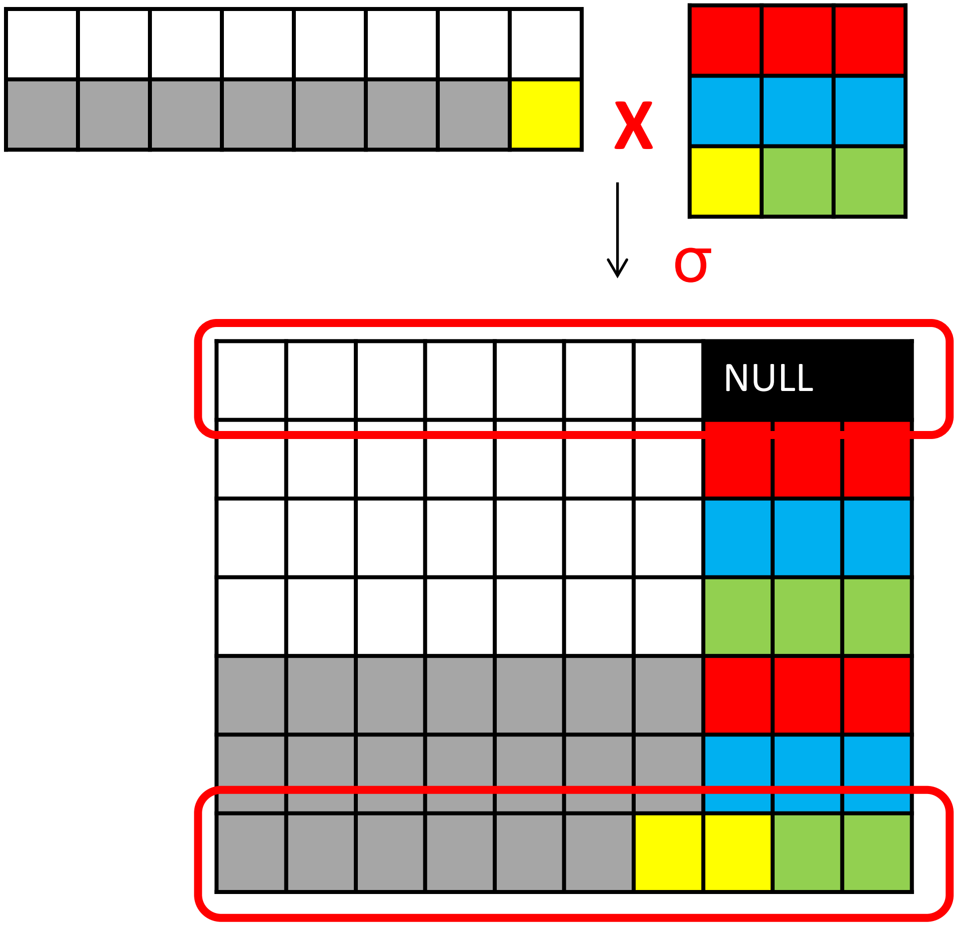
Let $\mathbf{A_1, A_2, \ldots, A_n}$ be the attributes in both the schema of **R** and the schema of **S**.

Then a tuple ***r*** from **R** and a tuple ***s*** from **S** are successfully paired if and only if ***r*** and ***s*** agree on each of the attributes $\mathbf{A_1, A_2, \ldots, A_n}$.

Outer join

1. For each tuple in R, include all tuples in S which satisfy join condition, but include also tuples of R that do not have matches in S

2. For this case, pair tuples of R with NULL



Left outer join

$$T = R ⟕_{condition} S$$

Outer join: example

Anonymous patient P

age	zip	disease
54	99999	heart
20	44444	flue
33	66666	lung

Anonymous occupation O

age	zip	job
54	99999	lawyer
20	44444	cashier

T= P ⟕ O

age	zip	disease	job
54	99999	heart	lawyer
20	44444	flue	cashier
33	66666	lung	NULL

Quick question

If I have a relation R with 100 records and a relation S with exactly 1 record, how many records will be in the result of R LEFT OUTER JOIN S?

A. At least 100, but could be more
B. Could be any number between 0 and 100 inclusive
C. 0
D. 1
E. 100

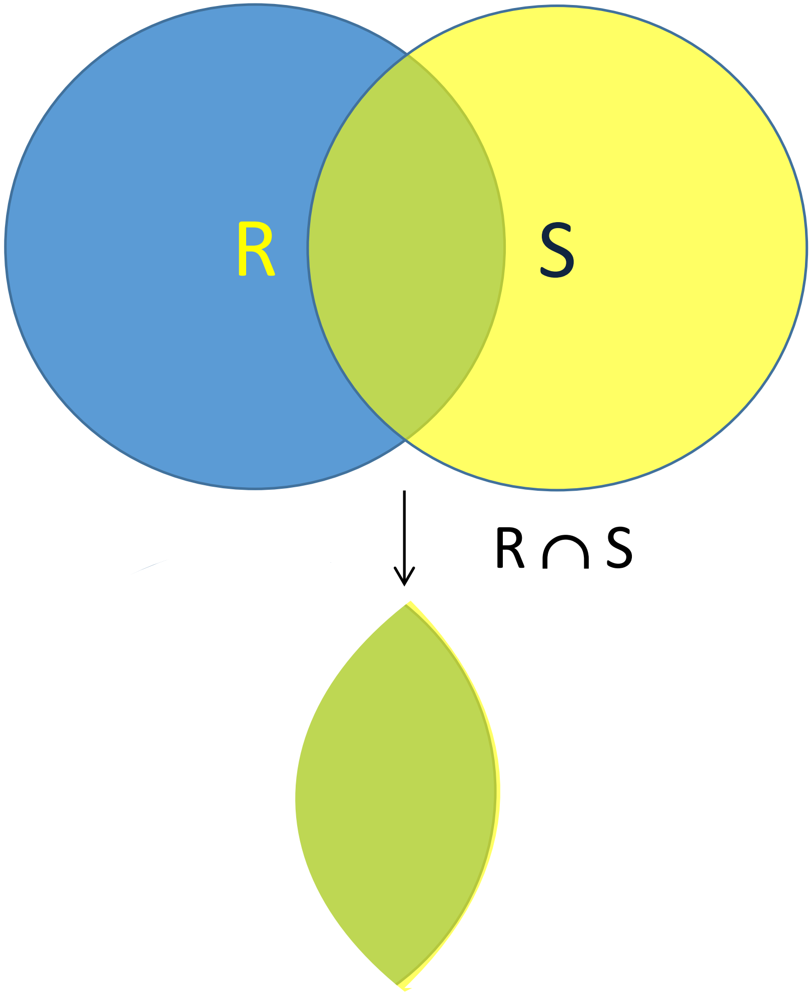
Quick question

If I have a relation R with 100 records and a relation S with exactly 1 record, how many records will be in the result of R LEFT OUTER JOIN S?

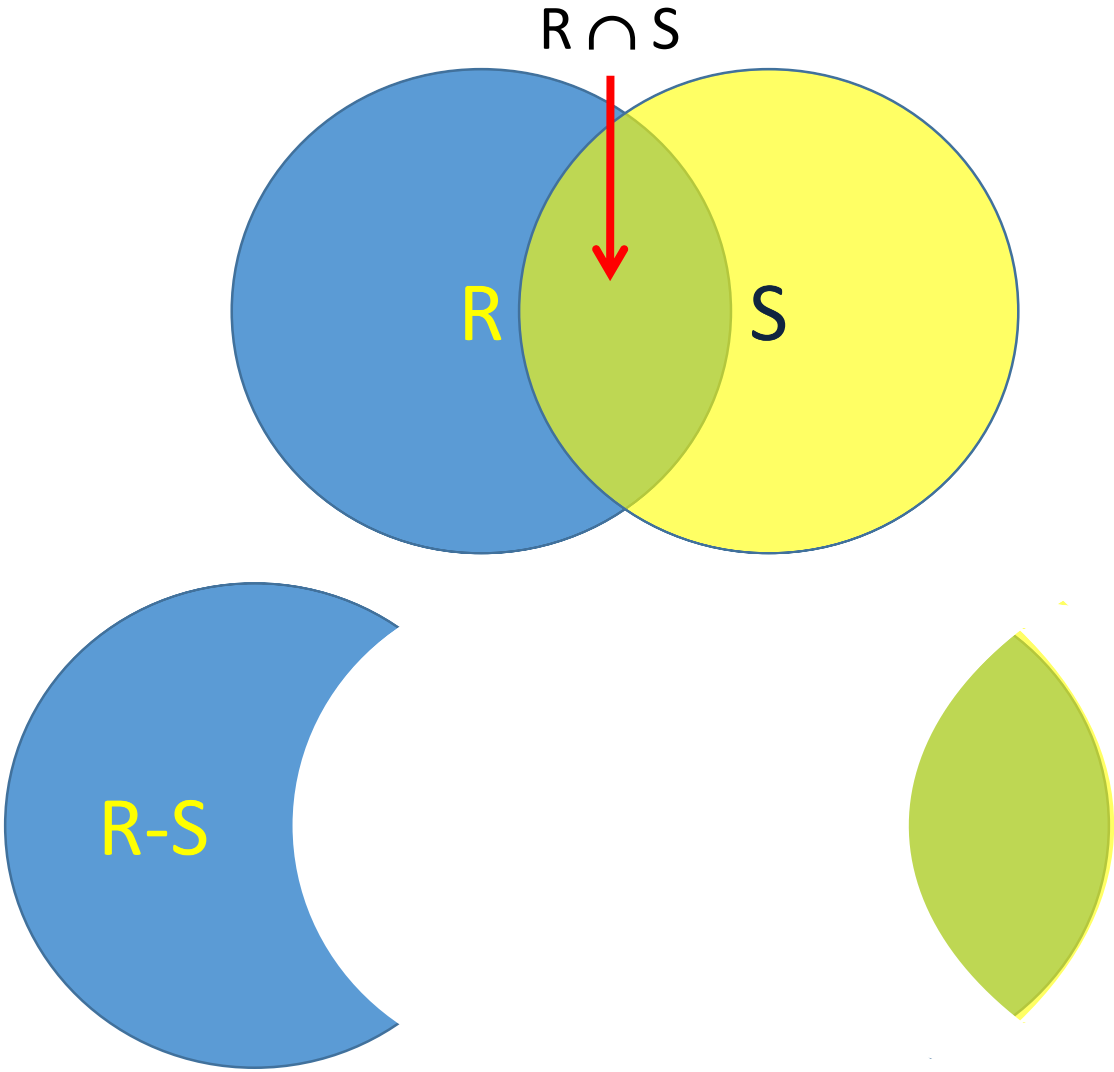
A. At least 100, but could be more
B. Could be any number between 0 and 100 inclusive
C. 0
D. 1
E. 100

Intersection

T=R ∩ S



Why intersection is not a “core” operation?



R - S (are in R but not in S)

R - (R-S)

Why intersection is not a “core” operation?

$R \cap S$ shortcut to
$R - (R - S)$

Can be derived using core operations

Set vs. bag (multi-set) semantics

- Sets: {a,b,c}, {a,d,e,f}, …
- Bags: {a,a,b,c}, {b,b,b,b,b}, …

- Relational algebra has two semantics:
  - Set semantics = standard relational algebra
  - Bag semantics = extended Relational Algebra

- Rule of thumb:
  - Every paper will assume set semantics
  - Every implementation will assume bag semantics

Operations on multisets

All RA operations need to be defined carefully on bags

- $\sigma_C(R)$: preserve the number of occurrences
- $\pi_A(R)$: no duplicate elimination
- Cross-product, join: no duplicate elimination

This is important- relational DBMSs work on multisets, not sets!

Extended operators on bags

- Duplicate elimination δ
- Sorting τ
- Grouping and aggregation γ

RDBMS query evaluation

How does a RDBMS answer your query?

SQL Query → Relational Algebra (RA) Plan → *Optimized* RA Plan → Execution

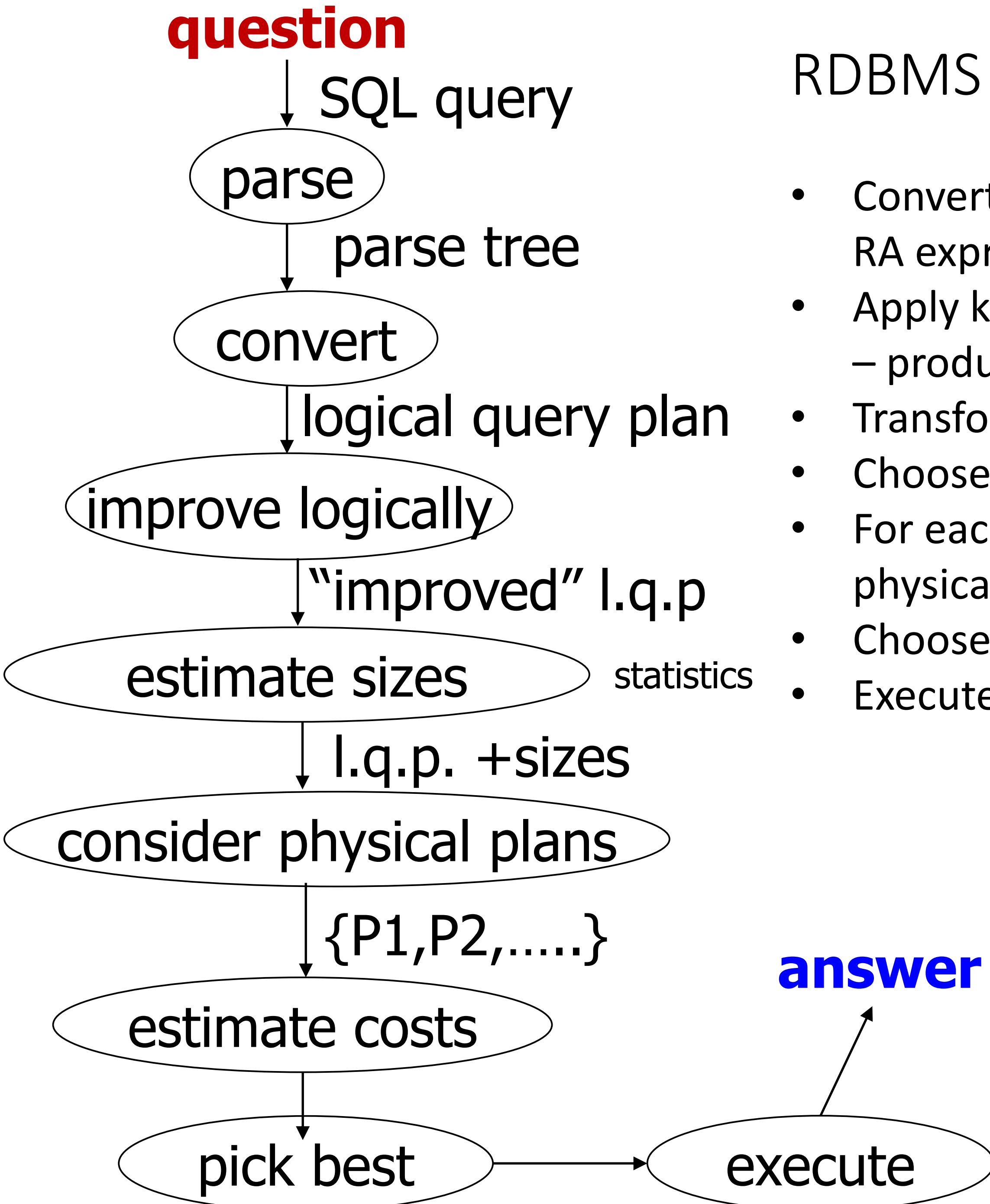
Declarative query (user declares what results are needed)

Translate to relational algebra expression

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

RDBMS query optimizer: steps



- Convert parsed SQL into corresponding RA expression
- Apply known algebraic transformations – produce improved logical query plan
- Transform based on estimated cost
- Choose one min-cost logical expression
- For each step, consider alternative physical implementations
- Choose physical plan with min I/Os
- Execute

RDBMS query evaluation

How does a RDBMS answer your query?

SQL Query → Relational Algebra (RA) Plan → *Optimized* RA Plan → Execution

Declarative query (user declares what results are needed)

Translate to relational algebra expression

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

That we just learned how to do!

Example: two SQL queries – the same execution plan

$R \bowtie_{R.A = S.B} S \Leftrightarrow \sigma_{R.A = S.B} (R \times S)$

Join

Cross-product with selection

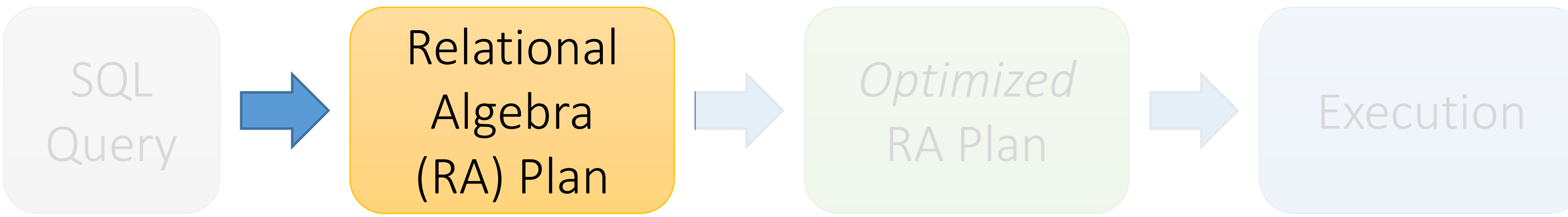
```
SELECT *
FROM R JOIN S
ON R.A = S.B
```

```
SELECT *
FROM R, S
WHERE R.A = S.B
```

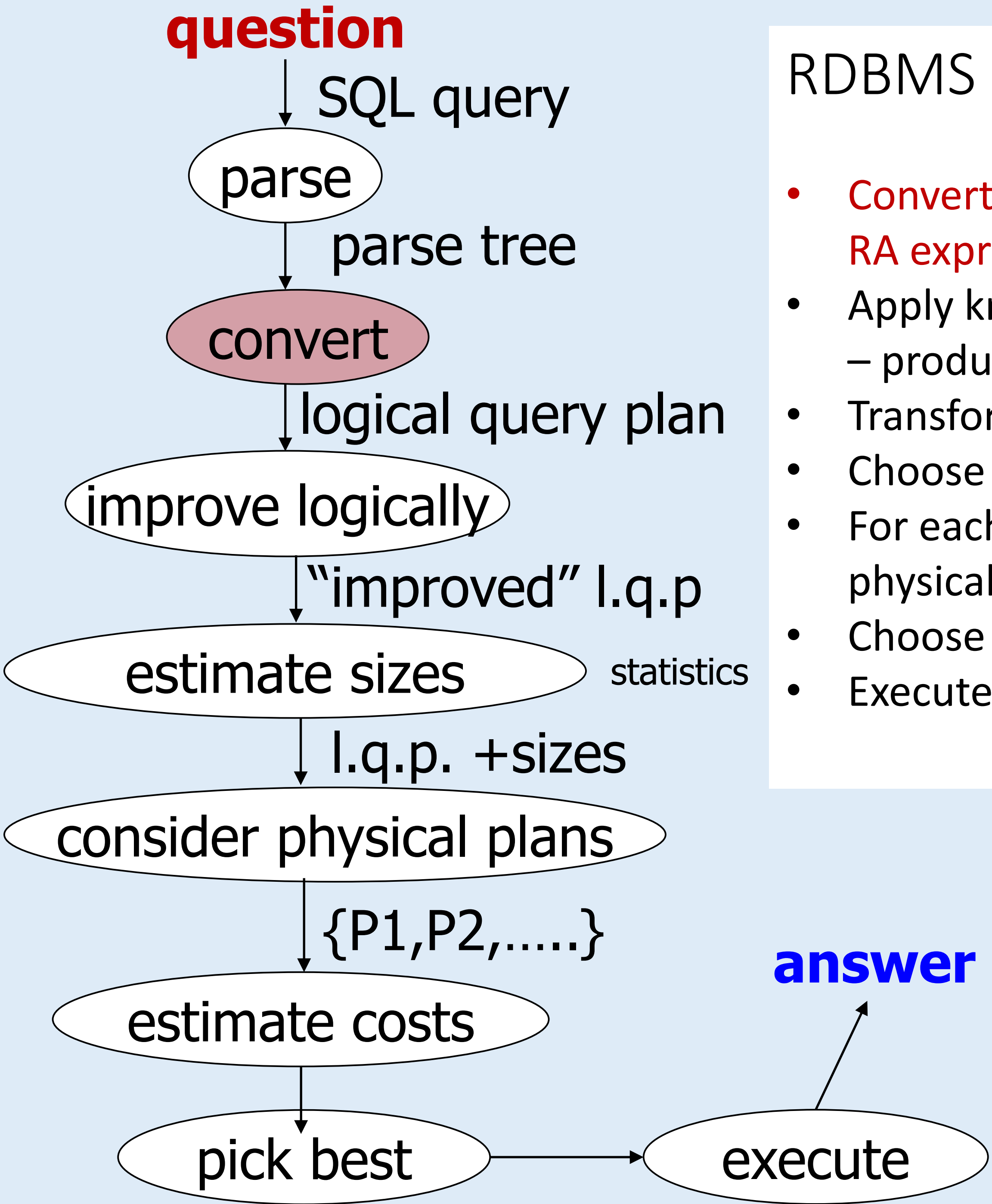
- Two ways to request the same results
- The optimizer does not care about the syntax of SQL query: it is going to work on the algebraic representation anyway
- Because the algebraic expressions are equivalent, the optimizer will have the same final plan for both queries

RDBMS query evaluation

How does a RDBMS answer your query?



Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!



RDBMS query optimizer: steps

- Convert parsed SQL into corresponding RA expression
- Apply known algebraic transformations – produce improved logical query plan
- Transform based on estimated cost
- Choose one min-cost logical expression
- For each step, consider alternative physical implementations
- Choose physical plan with min I/Os
- Execute

Translating general SQL queries (SELECT-FROM-WHERE) into RA

- What is the general form of an SFW query in RA?

- Given a general SFW SQL query:

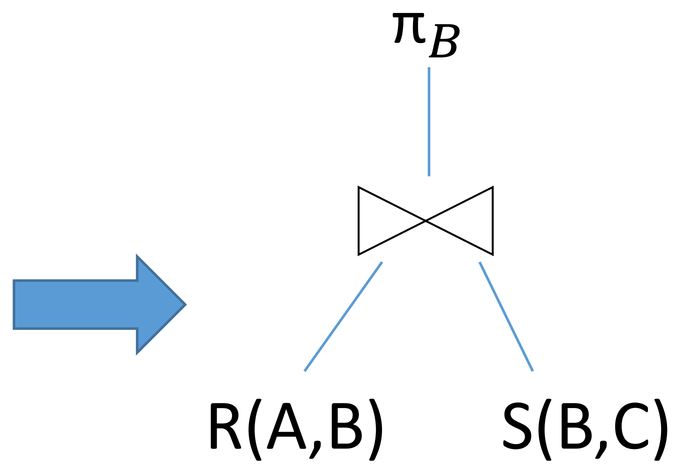
```
SELECT A_1, ..., A_n
FROM R_1, ..., R_m
WHERE c_1, ..., c_k;
```

- We can express this in relational algebra as follows:

$\pi_{A1,\ldots,An}(\sigma_{c1} \cdots \sigma_{ck}(R_1 x \ldots x R_m))$

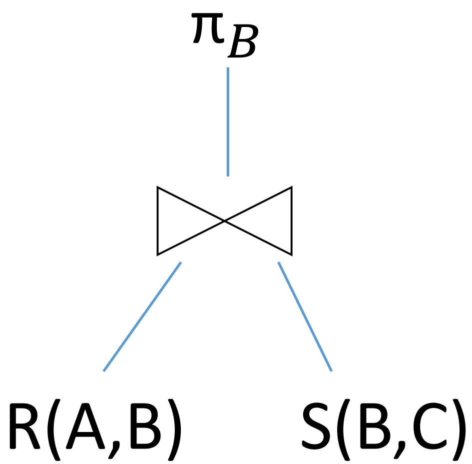
We can visualize the RA expression as a tree

$$\pi_B(R(A,B) \bowtie S(B,C))$$



Bottom-up tree traversal = order of operation execution!

From RA to SQL



What SQL query does this correspond to?

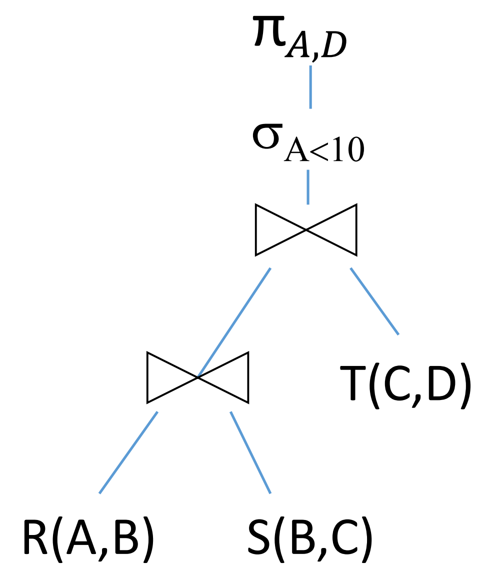
Are there any logically equivalent RA expressions?

From SQL to RA: example 1

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

```
SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

$$\pi_{A,D}(\sigma_{A<10}(T \bowtie (R \bowtie S)))$$



From SQL to RA: example 2

S (*product, city, price*)

```
SELECT city, count (*)
FROM S
GROUP BY city
HAVING sum(price)>100
```

$\pi_{city,c}$

$\sigma_{p>100}$

$\gamma_{\ city,\ sum(\text{price})\ \rightarrow\ \text{p},\ count(*)\rightarrow c}$

S(*product,city,price*)

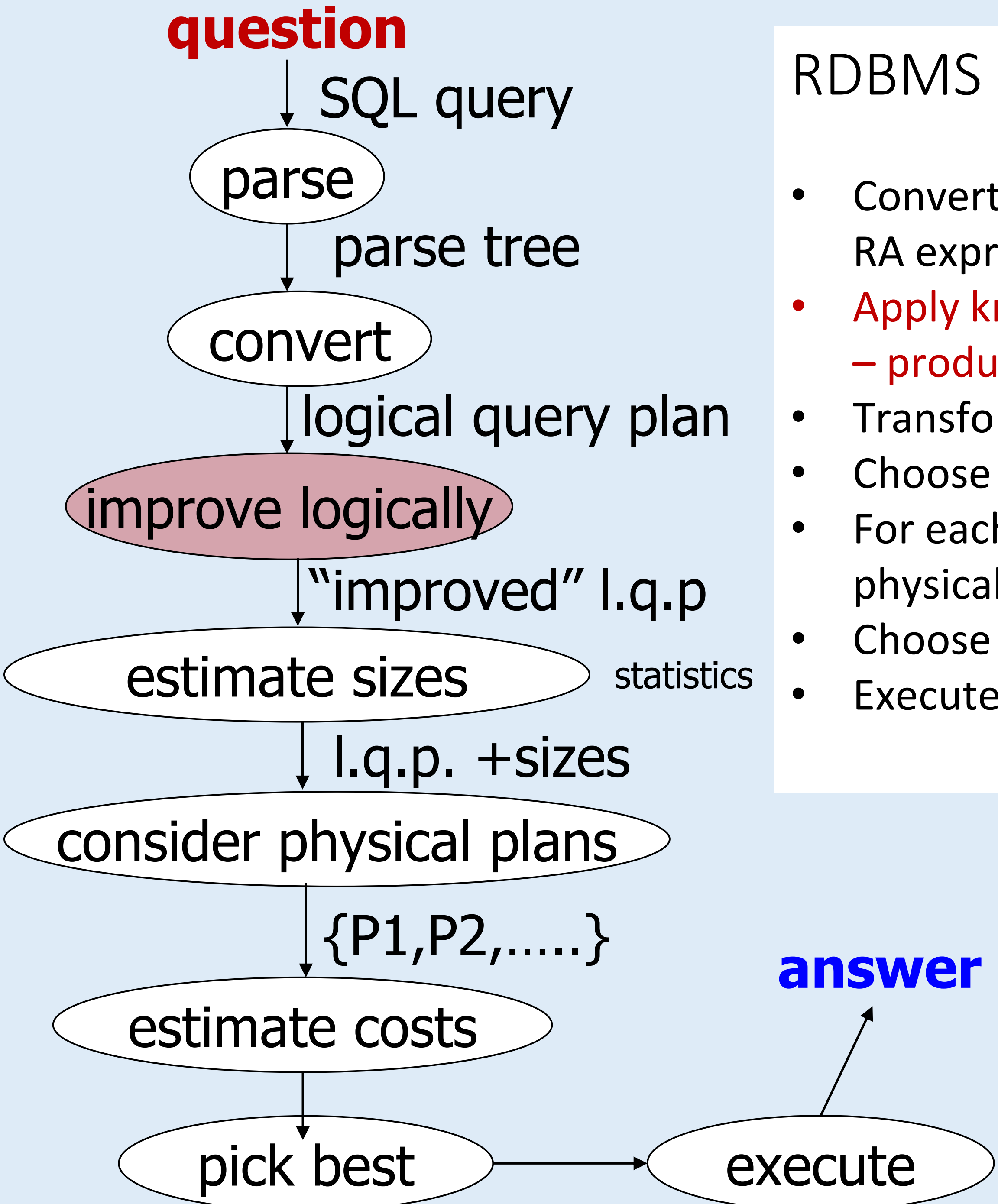
$$\pi_{city,\ c}\left(\sigma_{\ p>100}\left(\gamma_{\ city,\ sum(\text{price})\ \rightarrow\ \text{p},\ count(*)\rightarrow c}(S)\right)\right)$$

RDBMS query evaluation

How does a RDBMS answer your query?

SQL Query → Relational Algebra (RA) Plan → *Optimized* RA Plan → Execution

We transform the original RA expression into equivalent expressions using algebraic laws



RDBMS query optimizer: steps

- Convert parsed SQL into corresponding RA expression
- Apply known algebraic transformations – produce improved logical query plan
- Transform based on estimated cost
- Choose one min-cost logical expression
- For each step, consider alternative physical implementations
- Choose physical plan with min I/Os
- Execute

RA laws involving selection (σ)

Selection for a **single** relation

- Splitting law:

$$\sigma_{C \wedge D}(R) = \sigma_C(\sigma_D(R))$$

- Commutative law: order is flexible

$$\sigma_C(\sigma_D(R)) = \sigma_D(\sigma_C(R))$$

RA laws involving selection (σ)

Binary selection (on 2 relations)

$$\sigma_C(R \times S) = R \bowtie_C S$$

$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$$

$$\sigma_C(R \bowtie_D S) = \sigma_C(R) \bowtie_D S$$

For the binary operators, we **push the selection to R** only if all attributes in the condition *C* are in R.

Pushing selections: example

Consider ***R*(*A*,*B*)** and ***S*(*B*,*C*)** and the expression below:

$$\sigma_{A=1 \cap B<C}(R \bowtie S)$$

1. Splitting **AND** $\sigma_{A=1}(\sigma_{B<C}(R \bowtie S))$
2. Push σ to S $\sigma_{A=1}(R \bowtie \sigma_{B<C}(S))$
3. Push σ to R $\sigma_{A=1}(R) \bowtie \sigma_{B<C}(S)$

Laws for (bag) projection

A simple law: Project out attributes that are not needed later.

- i.e. keep only the output attr. and any join attribute.

$$\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$$

$$\pi_L(R \bowtie_C S) = \pi_L(\pi_M(R) \bowtie_C \pi_N(S))$$

$$\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$$

$$\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$$

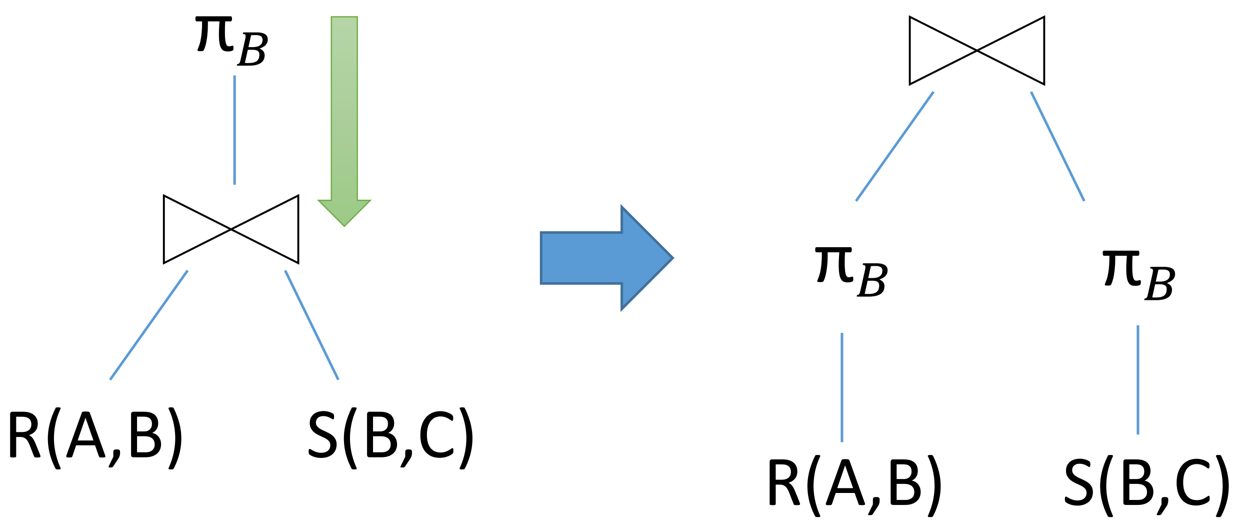
Pushing projection: example

Schema $R(a,b,c)$, $S(c,d,e)$

$$\pi_{a+e\rightarrow x}(R \rhd\lhd S) \equiv \pi_{a+e\rightarrow x}(\pi_{a,c}(R) \rhd\lhd \pi_{c,e}(S))$$

$$\pi_{a+b\rightarrow x,d+e\rightarrow y}(R \rhd\lhd S) \equiv \pi_{x,y}(\pi_{a+b\rightarrow x,c}(R) \rhd\lhd \pi_{d+e\rightarrow y,c}(S))$$

Why to push projections?



Why might we prefer this plan?

Commutative and associative laws for joins

- Commutative and associative laws for joins:

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

Above laws are applicable for both **sets** and **bags**

Quick question

- Given relation R(A,B):
  - Here, projection & selection commute:
    - $\sigma_{A=5}(\pi_A(R)) \leftrightarrow \pi_A(\sigma_{A=5}(R))$
  - What about here?
    - $\pi_B\left(\sigma_{A=5}(R)\right) \leftrightarrow \sigma_{A=5}(\pi_B(R))$?

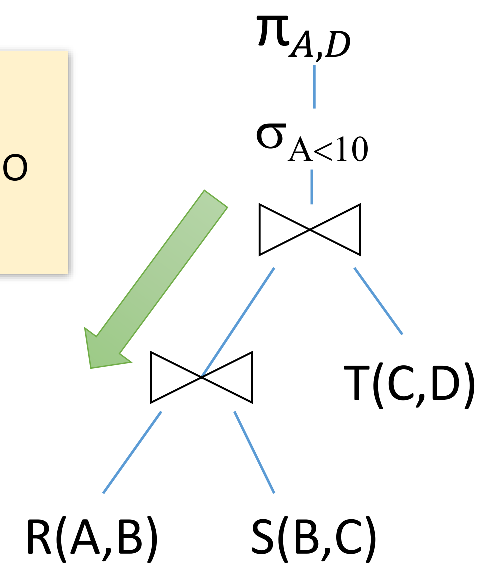
Logical optimization: example

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

```
SELECT R.A, T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

$$\pi_{A,D}\left(\sigma_{A<10}\left(T \bowtie (R \bowtie S)\right)\right)$$

Push down selection on A so it occurs earlier



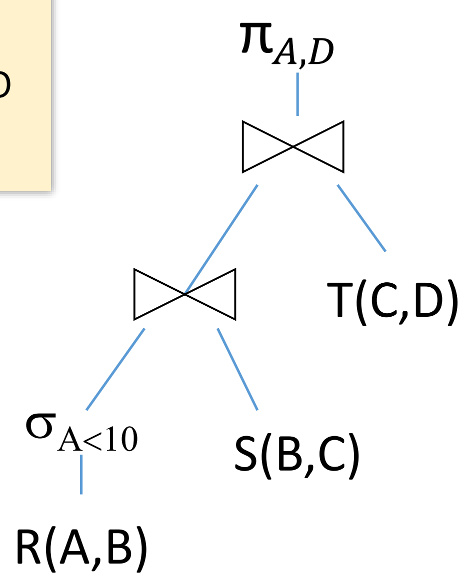
Logical optimization: example

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

```
SELECT R.A, T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

$$\pi_{A,D}\left(T \bowtie \left(\sigma_{A<10}(R) \bowtie S\right)\right)$$

Push down selection on A so it occurs earlier



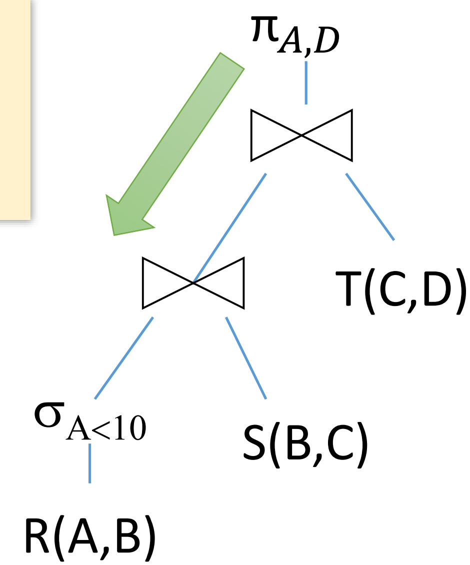
Logical optimization: example

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

SELECT R.A, T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;

$$\pi_{A,D}(T \bowtie (\sigma_{A<10}(R) \bowtie S))$$

Push down projection so it occurs earlier



Logical optimization: example

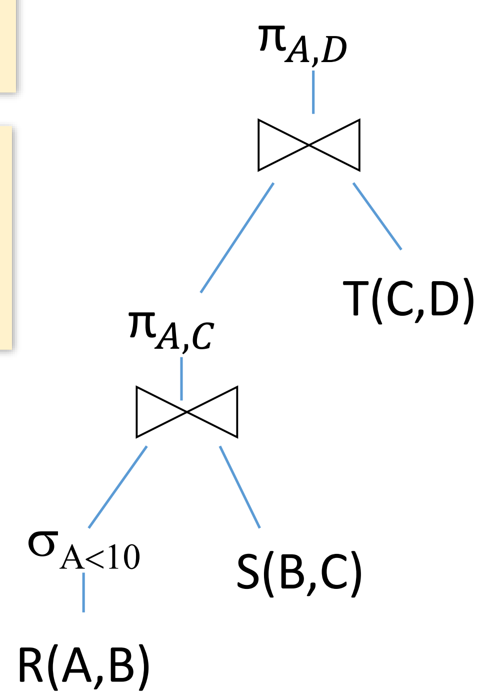
R(A,B) S(B,C) T(C,D)

```
SELECT R.A, T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

We eliminate B earlier!

In general, when is an attribute not needed…?

$$\pi_{A,D}\left(T \bowtie \pi_{A,C}(\sigma_{A<10}(R) \bowtie S)\right)$$



Improving logical query plans using algebraic laws: summary

1. Push σ as far down as possible

2. Do splitting of complex conditions in σ in order to push σ even further

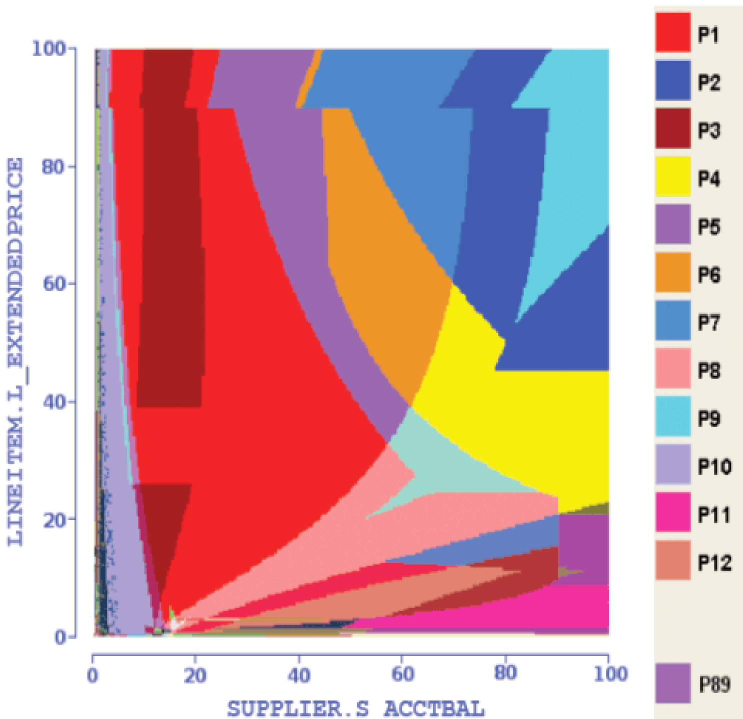
3. Push π as far down as possible, introduce new early π (but take care for exceptions)

4. Combine σ with $\times$ to produce Θ-joins or equijoins

5. Choose an order for joins

Topic by itself

Why still so many different plans selected for the same query? Depends on sizes of intermediate outputs



Picasso Database Query Optimizer Visualizer: [link]()

SELECT extendedprice
FROM lineitem, supplier
WHERE lineitem.sID = supplier.sID
AND **extendedprice**: *varies*
AND supplier.**accountbalance**: *varies*

- Same query executed with different selection cardinality – covering from 0 to 100% of all values – results in completely different plans

- Here: 89 plans, each in different color

Coming next:
cost-based transformations