CS150: Database & Datamining
Lecture 14: Relational Algebra & Query Optimization

ShanghaiTech-SIST

Spring 2019

Acknowledgement: Slides are adopted from the Berkeley course CS186 by Joey Gonzalez and Joe Hellerstein, Stanford CS145 by Peter Bailis.

Today's Lecture

1. The Relational Model & Relational Algebra

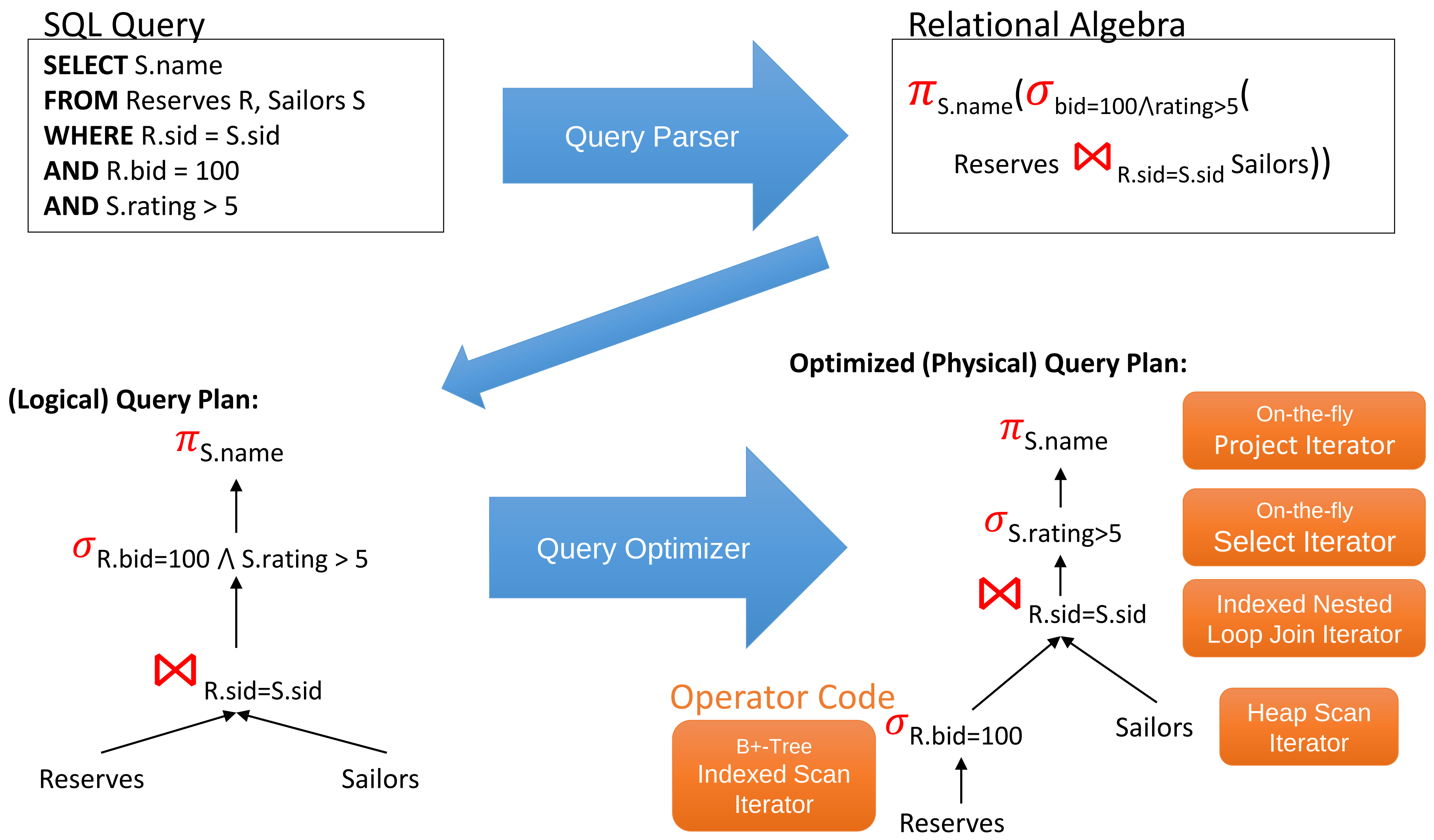
2. Query Optimization:

    - Logical Optimization

    - Physical Optimization (next time)

1. The Relational Model & Relational Algebra

Big Picture Overview



Big Picture Overview

SQL Query

SELECT S.name
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
AND R.bid = 100
AND S.rating > 5

Relational Algebra

$$\pi_{S.name}(\sigma_{bid=100 \wedge rating>5}(\text{Reserves} \bowtie_{R.sid=S.sid} \text{Sailors}))$$

Why?

(Logical) Query Plan:

Optimized (Physical) Query Plan:

SQL

A **declarative** expression of the query result

Relational Algebra

Operational description of a computation.

Reserves

Sailors

Systems optimize and execute relational algebra query plan.

Motivation

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

Database maps internally into this *procedural language.*

The Relational Model: Schemata

- Relational Schema:

Students(sid: *string*, name: *string*, gpa: *float*)

Relation name

String, float, int, etc. are the **domains** of the attributes

Attributes

The Relational Model: Data

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of attributes is the **arity** of the relation

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of tuples is the <u>**cardinality**</u> of the relation

A <u>**tuple**</u> or <u>**row**</u> (or *record)* is a single entry in the table having the attributes specified by the schema

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

Recall: In practice DBMSs relax the set requirement, and use multisets.

A **relational instance** is a ***set*** of tuples all conforming to the same *schema*

To Reiterate

- A *relational schema* describes the data that is contained in a *relational instance*

Let $R(f_1{:}Dom_1,\ldots,f_m{:}Dom_m)$ be a *relational schema* then, an *instance* of R is a subset of $Dom_1 \times Dom_2 \times \ldots \times Dom_n$

In this way, a *relational schema* R is a **total function from attribute *names* to types**

One More Time

- A ***relational schema*** describes the data that is contained in a ***relational instance***

A relation R of arity *t* is a function:
$R : Dom_1 \times \ldots \times Dom_t \rightarrow \{0,1\}$

I.e. returns whether or not a tuple of matching types is a member of it

Then, the schema is simply the *signature* of the function

Note here that order matters, attribute name doesn't…
We'll (mostly) work with the other model (last slide) in which **attribute name matters, order doesn't!**

A relational database

- A ***relational database schema*** is a set of relational schemata, one for each relation

- A ***relational database instance*** is a set of relational instances, one for each relation

Two conventions:

1. We call relational database instances as simply ***databases***
2. We assume all instances are valid, i.e., satisfy the *domain constraints*

Remember the CMS

- *Relation DB Schema*
  - Students(sid: *string*, name: *string*, gpa: *float*)
  - Courses(cid: *string*, cname: *string*, credits: *int*)
  - Enrolled(sid: *string,* cid*: string,* grade*: string*)

Note that the schemas impose effective domain / type constraints, i.e. Gpa can't be "Apple"

Relation Instances

Sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

cid	cname	credits
564	564-2	4
308	417	2

Courses

sid	cid	Grade
123	564	A

Enrolled

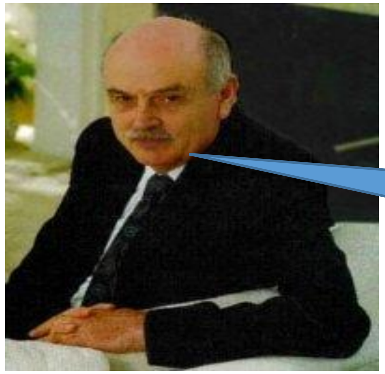
2nd Part of the Model: Querying

```
SELECT S.name
FROM Students S
WHERE S.gpa > 3.5;
```

We don't tell the system *how* or *where* to get the data- **just what we want, i.e., Querying is *declarative***

"Find names of all students with GPA > 3.5"

To make this happen, we need to translate the *declarative* query into a series of operators… we'll see this next!



Actually, I showed how to do this translation for a much richer language!

Formal Relational QL’s

- **Relational Calculus:** (Basis for SQL)
  - Describe the result of computation
  - Based on first order logic
  - Tuple Relational Calculus **(TRC)**
    - $\{S \mid S \in \text{Sailors}\ \exists R \in \text{Reserves}$
      $(R.sid = S.sid \wedge R.bid = 103)\}$

Are these equivalent?

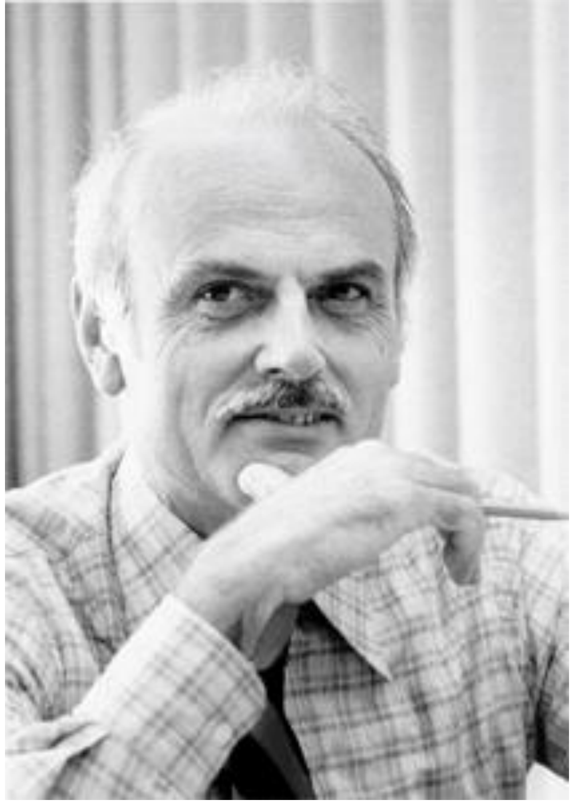
Can we go from one to the other?

- **Relational Algebra:**
  - Algebra on sets
  - Operational description of transformations

Codd’s Theorem

- Established equivalence in expressivity between :
  - **Relational Calculus***
  - **Relational Algebra**

- Why an import result?
  - Connects **declarative** representation of queries with **operational description**
  - **Constructive:** we can compile SQL into relational algebra



Edgar F. “Ted” Codd
(1923 - 2003)
Turing Award 1981

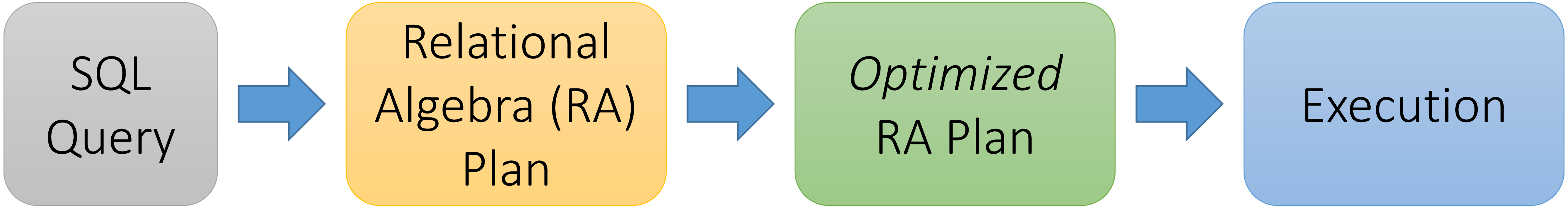
*Domain Independent Relational Calculus

Imaged obtained from https://en.wikipedia.org/wiki/Edgar_F._Codd

Relational Algebra

RDBMS Architecture

How does a SQL engine work ?



Declarative query (from user)

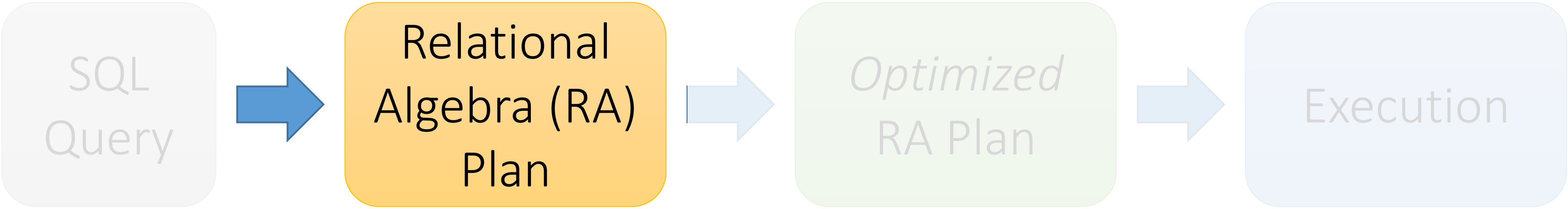
Translate to relational algebra expresson

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

RDBMS Architecture

How does a SQL engine work ?



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

Relational Algebra (RA)

- <u>Five **basic** operators:</u>
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: -

We'll look at these first!

- <u>Derived or auxiliary operators:</u>
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ
  - Division

And also at one example of a derived operator (natural join) and a special operator (renaming)

Keep in mind: RA operates on sets!

- RDBMSs use *multisets*, however in relational algebra formalism we will consider **sets!**

- Also: we will consider the ***named perspective***, where every attribute must have a unique name
  - →attribute order does not matter…

Now on to the basic RA operators…

1. Selection (σ)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary > 40000}}$ (Employee)
  - $\sigma_{\text{name = “Smith”}}$ (Employee)
- The condition c can be =, <, ≤, >, ≥, <>

Students(sid,sname,gpa)

SQL:

```
SELECT *
FROM Students
WHERE gpa > 3.5;
```

RA:

$$\sigma_{gpa > 3.5}(Students)$$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{Salary > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

2. Projection (Π)

Students(sid,sname,gpa)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}$ (Employee)
  - Output schema: Answer(SSN, Name)

SQL:

```
SELECT DISTINCT
  sname,
  gpa
FROM Students;
```

RA:

$$\Pi_{sname,gpa}(Students)$$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{\text{Name,Salary}}$ (Employee)

Name	Salary
John	200000
John	600000

Note that RA Operators are Compositional!

Students(sid,sname,gpa)

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

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

$$\Updownarrow$$

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

How do we represent this query in RA?

Are these logically equivalent?

3. Cross-Product (×)

- Each tuple in R1 with each tuple in R2
- Notation: R1 × R2
- Example:
  - Employee × Dependents
- Rare in practice; mainly used to express joins

Students(sid,sname,gpa)
People(ssn,pname,address)

SQL:

```
SELECT *
FROM Students, People;
```

RA:

$$Students \times People$$

Another example:

People

ssn	pname	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse

$\times$

Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

$$Students \times People$$

ssn	pname	address	sid	sname	gpa
1234545	John	216 Rosse	001	John	3.4
5423341	Bob	217 Rosse	001	John	3.4
1234545	John	216 Rosse	002	Bob	1.3
5423341	Bob	216 Rosse	002	Bob	1.3

Renaming (ρ)

- Changes the schema, not the instance
- A ‘special’ operator- neither basic nor derived
- Notation: $\rho_{B1,\ldots,Bn}$ (R)

- **Note: this is shorthand for the proper form (since names, not order matters!):**
  - $\rho_{A1 \rightarrow B1,\ldots,An \rightarrow Bn}$ (R)

Students(sid,sname,gpa)

SQL:

```
SELECT
  sid AS studId,
  sname AS name,
  gpa AS gradePtAvg
FROM Students;
```

RA:

$$\rho_{studId,name,gradePtAvg}(Students)$$

We care about this operator *because* we are working in a *named perspective*

Another example:

Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

$$\rho_{studId,name,gradePtAvg}(Students)$$

Students

studId	name	gradePtAvg
001	John	3.4
002	Bob	1.3

Natural Join (⋈)

- Notation: $R_1 \bowtie R_2$
- Joins R_1 and R_2 on *equality of all shared attributes*
  - If R_1 has attribute set A, and R_2 has attribute set B, and they share attributes $A \cap B = C$, can also be written: $R_1 \bowtie_C R_2$
- Our first example of a *derived* RA operator:
  - Meaning: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \to D}(R_1) \times R_2))$
  - Where:
    - The rename $\rho_{C \to D}$ renames the shared attributes in one of the relations
    - The selection $\sigma_{C=D}$ checks equality of the shared attributes
    - The projection $\Pi_{A \cup B}$ eliminates the duplicate common attributes

Students(sid,name,gpa)
People(ssn,name,address)

SQL:

```
SELECT DISTINCT
  ssid, S.name, gpa,
  ssn, address
FROM
  Students S,
  People P
WHERE S.name = P.name;
```

RA:

$$Students \bowtie People$$

Another example:

Students S

sid	S.name	gpa
001	John	3.4
002	Bob	1.3

$\bowtie$

People P

ssn	P.name	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse

$$Students \bowtie People$$

sid	S.name	gpa	ssn	address
001	John	3.4	1234545	216 Rosse
002	Bob	1.3	5423341	216 Rosse

Natural Join

- Given schemas R(A, B, C, D), S(A, C, E), what is the schema of $R \bowtie S$?
- Given R(A, B, C), S(D, E), what is $R \bowtie S$?
- Given R(A, B), S(A, B), what is $R \bowtie S$?

Example: Converting SFW Query -> RA

Students(sid,sname,gpa)
People(ssn,sname,address)

```
SELECT DISTINCT
  gpa,
  address
FROM Students S,
     People P
WHERE gpa > 3.5 AND
   sname = pname;
```

$$\Pi_{gpa,address}(\sigma_{gpa>3.5}(S \bowtie P))$$

How do we represent this query in RA?

Logical Equivalece of RA Plans

- Given relations R(A,B) and S(B,C):
  - Here, projection & selection commute:
    - $\sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R))$
  - What about here?
    - $\sigma_{A=5}(\Pi_B(R))\ ?= \Pi_B(\sigma_{A=5}(R))$

We’ll look at this in more depth later in the lecture…

RDBMS Architecture

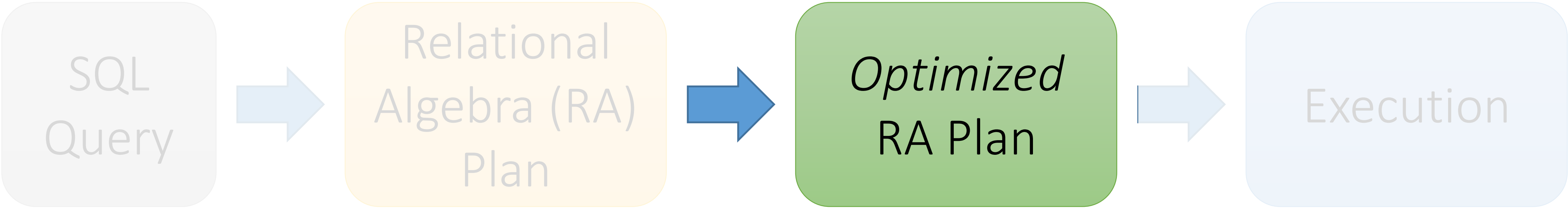
How does a SQL engine work ?

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

We saw how we can transform declarative SQL queries into precise, compositional RA plans

RDBMS Architecture

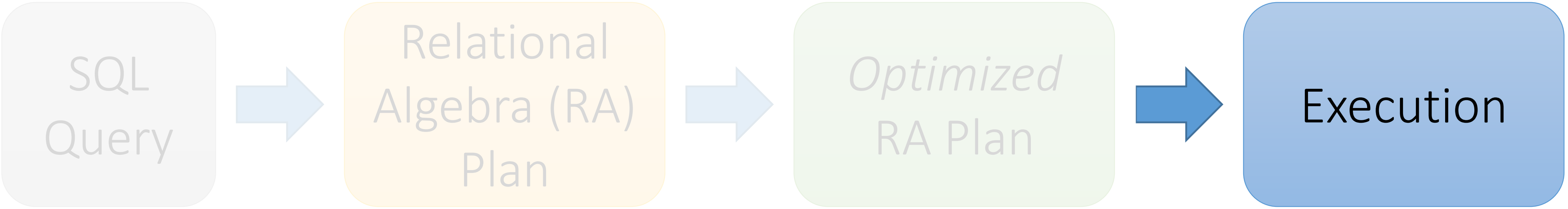
How does a SQL engine work ?



We'll look at how to then optimize these plans later in this lecture

RDBMS Architecture

How is the RA “plan” executed?



We already know how to execute all the basic operators!

RA Plan Execution

- Natural Join / Join:
  - We saw how to use **memory & IO cost considerations to pick the correct algorithm to execute a join with (BNLJ, SMJ, HJ…)!**
- Selection:
  - We saw how to use **indexes to aid selection**
  - Can always fall back on scan / binary search as well
- Projection:
  - The main operation here is finding *distinct* values of the project tuples; we briefly discussed how to do this with e.g. **hashing** or **sorting**

We already know how to execute all the basic operators!

Boats(<u>bid</u>,bname,color)
Sailors(<u>sid</u>, sname, rating, age)
Reserves(<u>sid</u>, <u>bid</u>, <u>day</u>)

Exercise:

Find names of sailors who've reserved boat #103

- Solution 1:

$$\pi_{\text{sname}}(\sigma_{\text{bid=103}}(\text{Sailors} \bowtie \text{Reserves}))$$

- Solution 2:

$$\pi_{\text{sname}}(\text{Sailors} \bowtie \sigma_{\text{bid=103}}(\text{Reserves}))$$

Exercise:

Boats(bid,bname,color)
Sailors(sid, sname, rating, age)
Res(sid, bid, day)

Find names of sailors who've reserved a red boat

- Solution 1:

$$\pi_{\text{sname}}(\sigma_{\text{color='red'}}(\text{Boats}) \bowtie \text{Res} \bowtie \text{Sailors})$$

- More "efficient" Solution 2:

$$\pi_{\text{sname}}(\ \pi_{\text{sid}}(\ \pi_{\text{bid}}(\sigma_{\text{color='red'}}(\text{Boats})) \bowtie \text{Res}) \bowtie \text{Sailors})$$

In general many possible equivalent expressions: **algebra**…

Relational Algebra Rules

- **Operator Precedence:**
  - Unary operators before binary operators
- **Selections:**
  - $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ (*cascade*)
  - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (*commute*)
- **Projections:**
  - $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{a1, \ldots, an-1}(R))\ldots)$ (*cascade*)
- **Cartesian Product**
  - $R \times (S \times T) \equiv (R \times S) \times T$ (associative)
  - $R \times S \equiv S \times R$ (commutative)
  - *Applies for joins as well but* be careful with join predicates …

Adv. Relational Algebra

Relational Algebra (RA)

- Five **basic** operators:
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: -

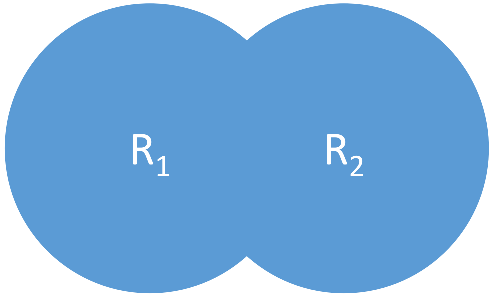
We'll look at these

- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ
  - Division

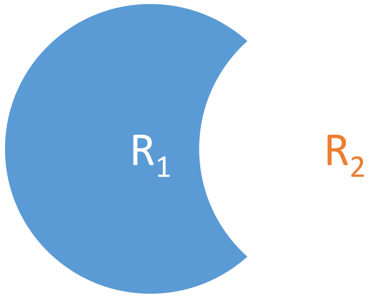
And also at some of these derived operators

1. Union (∪) and 2. Difference (–)

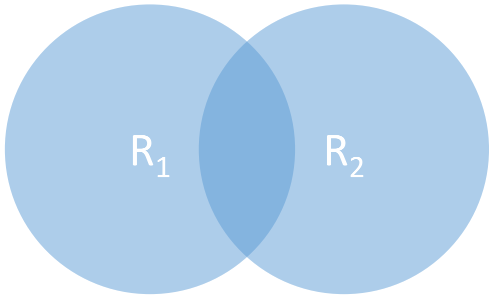
- R1 ∪ R2
- Example:
  - ActiveEmployees ∪ RetiredEmployees



- R1 – R2
- Example:
  - AllEmployees -- RetiredEmployees



What about Intersection (∩) ?



- It is a derived operator
- $R1 \cap R2 = R1 - (R1 - R2)$
- Also expressed as a join!
- Example
  - UnionizedEmployees ∩ RetiredEmployees

Theta Join ($\bowtie_\theta$)

- A join that involves a predicate
- R1 $\bowtie_\theta$ R2 $=$ σ_θ (R1 $\times$ R2)
- Here θ can be any condition

Note that natural join is a theta join + a projection.

Students(sid,sname,gpa)
People(ssn,pname,address)

SQL:

```
SELECT *
FROM
  Students,People
WHERE θ;
```

RA:

$$Students \bowtie_\theta People$$

Equi-join ($\bowtie_{A=B}$)

- A theta join where θ is an equality
- $R1 \bowtie_{A=B} R2 = \sigma_{A=B}(R1 \times R2)$
- Example:
  - Employee $\bowtie_{SSN=SSN}$ Dependents

Most common join in practice!

Students(sid,sname,gpa)
People(ssn,pname,address)

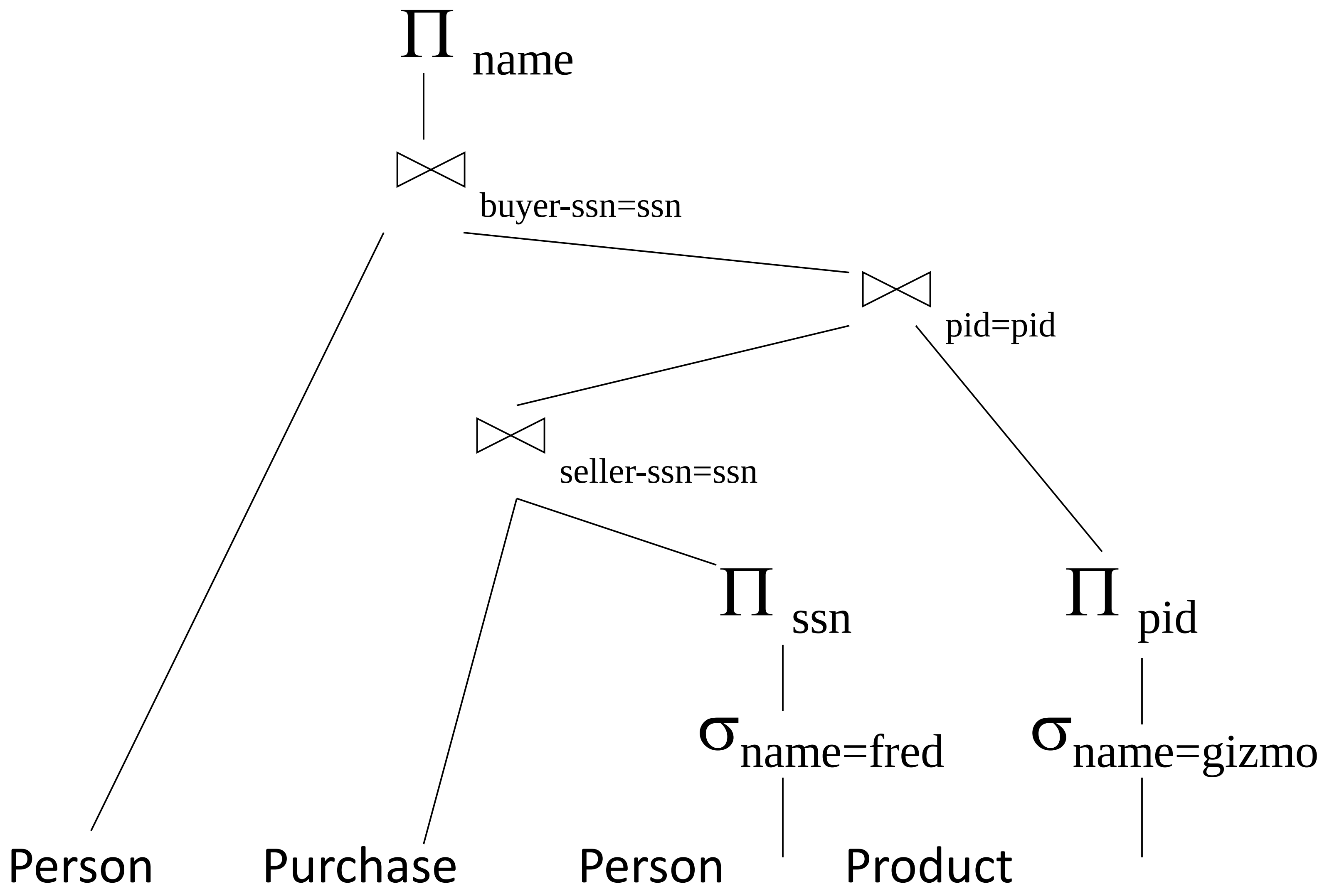
SQL:

```
SELECT *
FROM
  Students S,
  People P
WHERE sname = pname;
```

RA:

$$S \bowtie_{sname=pname} P$$

RA Expressions Can Get Complex!



Multisets

Recall that SQL uses Multisets

$\boldsymbol{\lambda}(\boldsymbol{X})$= "Count of tuple in X"
(Items not listed have implicit count 0)

Multiset X

Tuple
(1, a)
(1, a)
(1, b)
(2, c)
(2, c)
(2, c)
(1, d)
(1, d)

Equivalent Representations of a <u>Multiset</u>

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

Note: In a set all counts are {0,1}.

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 engines work on multisets, not sets!

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 modern SQL…

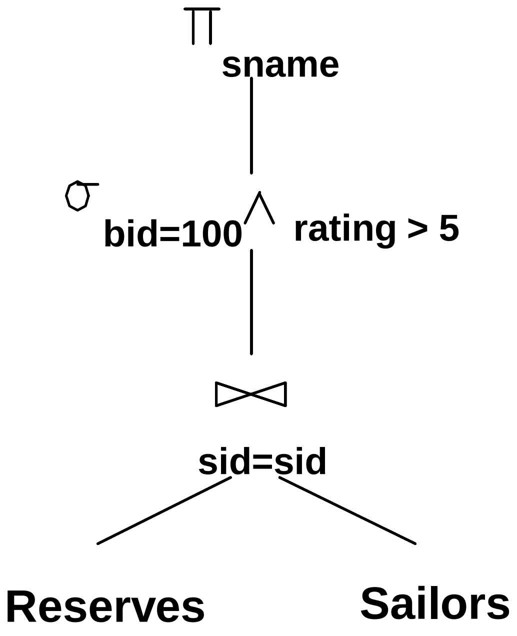
2. Query Optimization

Query Optimization Overview

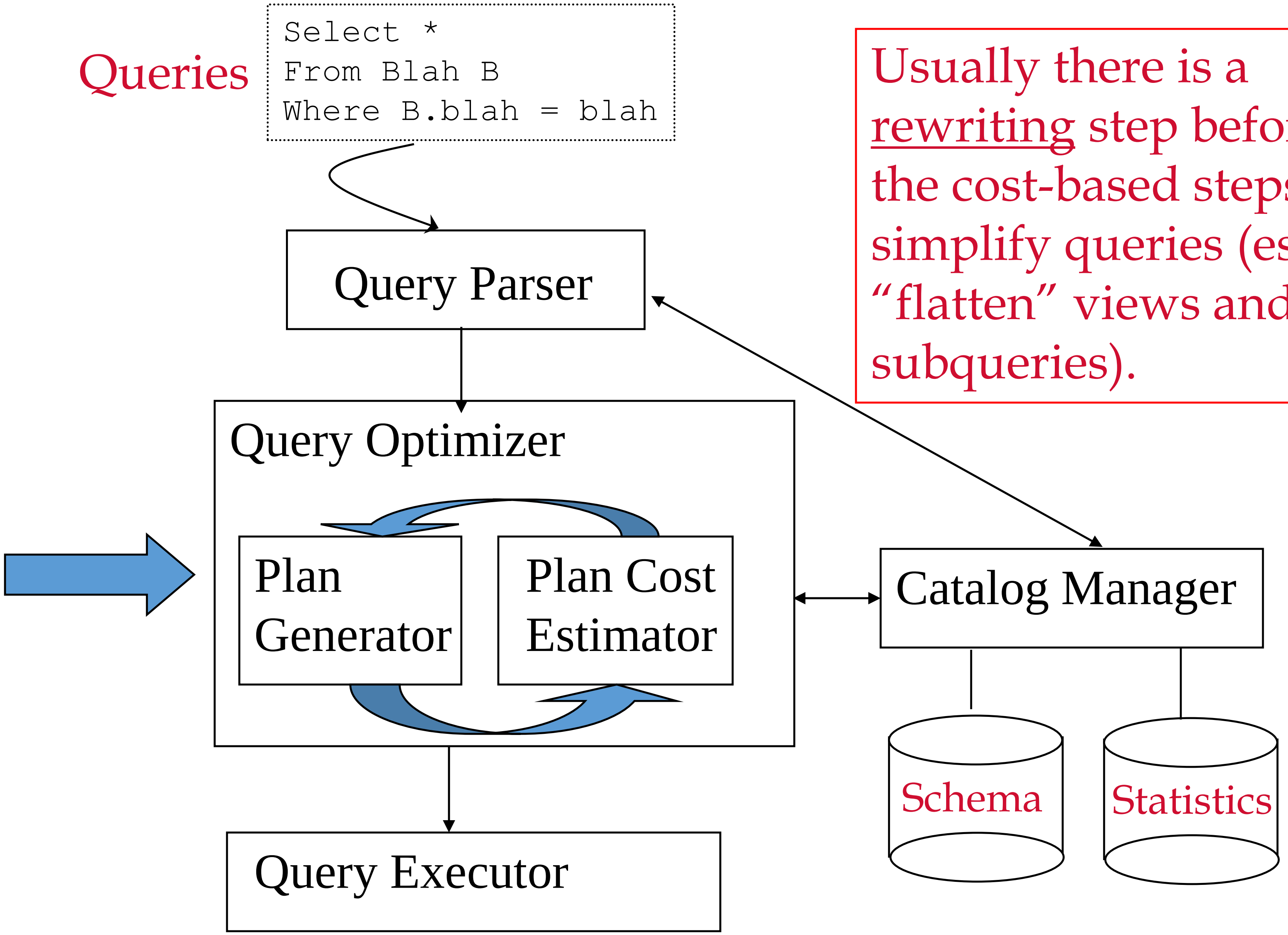
- Query can be converted to relational algebra
- Rel. Algebra converts to tree
- Each operator has implementation choices
- Operators can also be applied in different orders!

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

$$\pi_{(sname)}\sigma_{(bid=100 \wedge rating > 5)}(Reserves \bowtie Sailors)$$

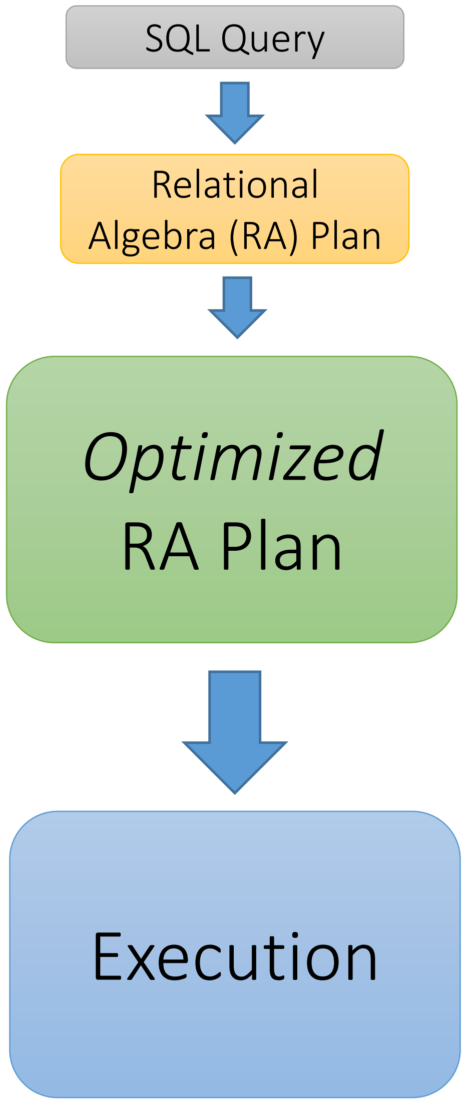


Cost-based Query Sub-System



Usually there is a <u>rewriting</u> step before the cost-based steps to simplify queries (esp. to "flatten" views and subqueries).

Logical vs. Physical Optimization



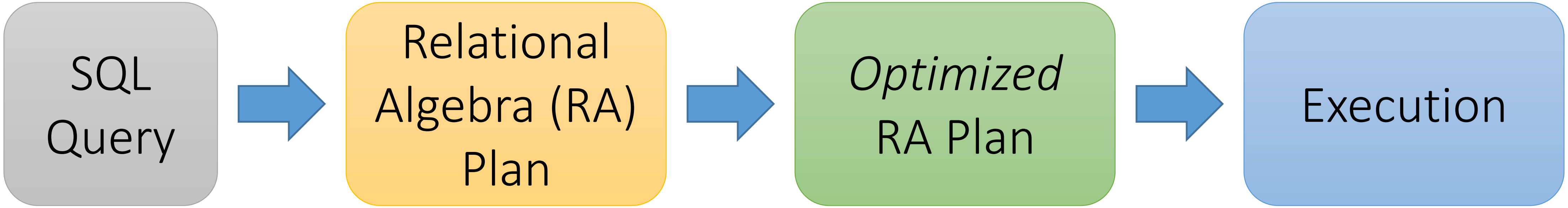
- **<u>Logical optimization:</u>**
  - Find equivalent plans that are more efficient
  - *Intuition: Minimize # of tuples at each step by changing the order of RA operators*

- **<u>Physical optimization:</u>**
  - Find algorithm with lowest IO cost to execute our plan
  - *Intuition: Calculate based on physical parameters (buffer size, etc.) and estimates of data size (histograms)*

A. Logical Optimization

RDBMS Architecture

How does a SQL engine work ?



Declarative query (from user)

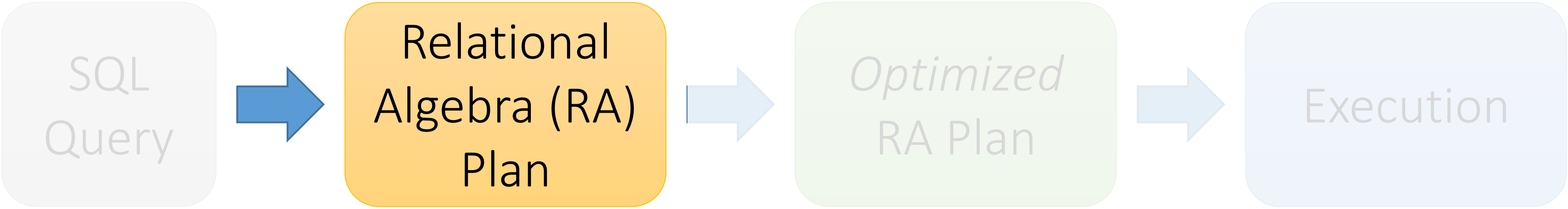
Translate to relational algebra expresson

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

RDBMS Architecture

How does a SQL engine work ?



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

Recall: Relational Algebra (RA)

- Five **basic** operators:
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: -

We'll look at these first!

- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ
  - Division

And also at one example of a derived operator (natural join) and a special operator (renaming)

Recall: Converting SFW Query -> RA

Students(sid,sname,gpa)
People(ssn,sname,address)

```
SELECT DISTINCT
  gpa,
  address
FROM Students S,
     People P
WHERE gpa > 3.5 AND
   sname = pname;
```

$$\Pi_{gpa,address}(\sigma_{gpa>3.5}(S \bowtie P))$$

How do we represent this query in RA?

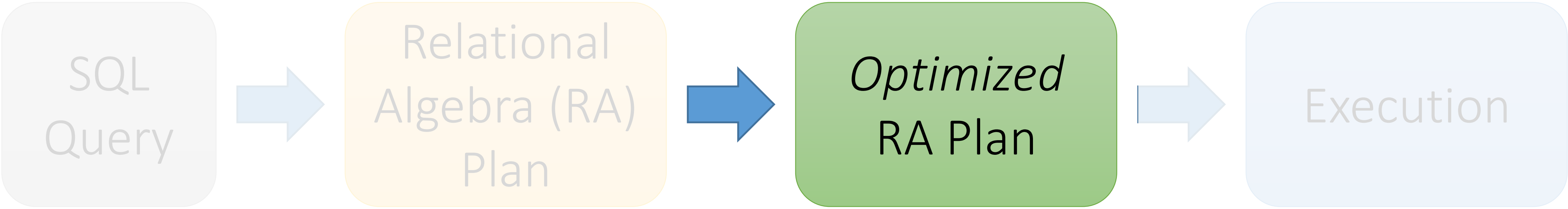
Recall: Logical Equivalece of RA Plans

- Given relations R(A,B) and S(B,C):
  - Here, projection & selection commute:
    - $\sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R))$
  - What about here?
    - $\sigma_{A=5}(\Pi_B(R))\ ?= \Pi_B(\sigma_{A=5}(R))$

We'll look at this in more depth later in the lecture…

RDBMS Architecture

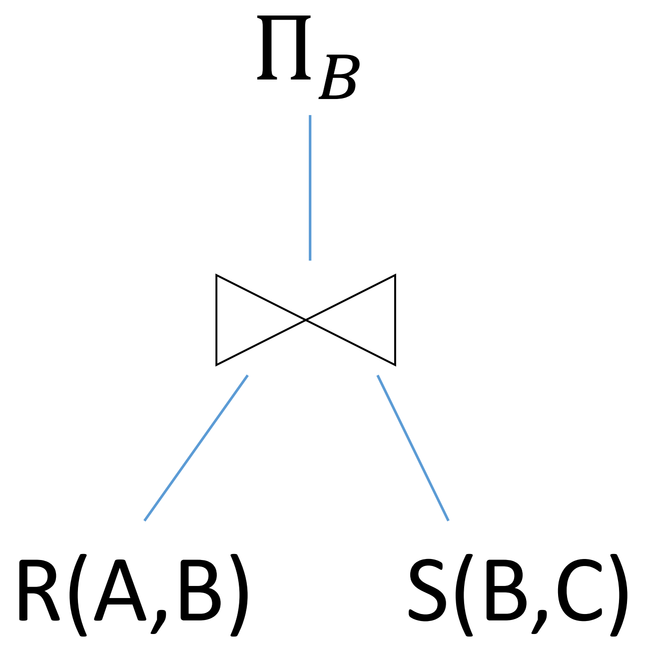
How does a SQL engine work ?



We'll look at how to then optimize these plans now

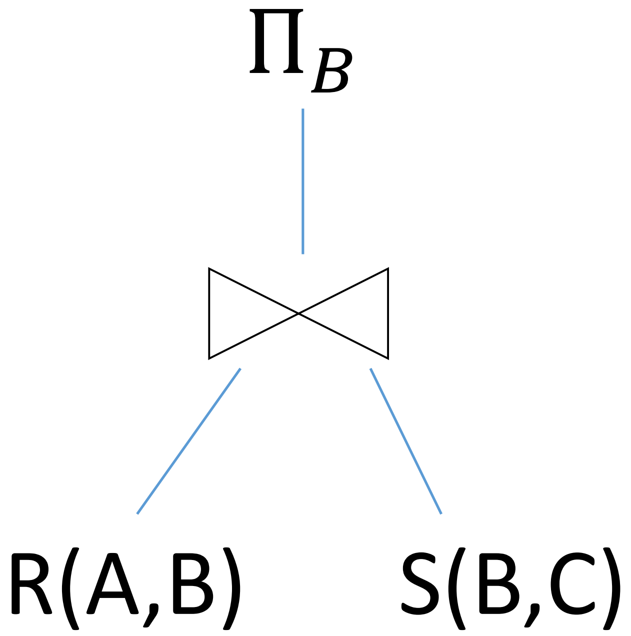
Note: We can visualize the plan as a tree

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



Bottom-up tree traversal = order of operation execution!

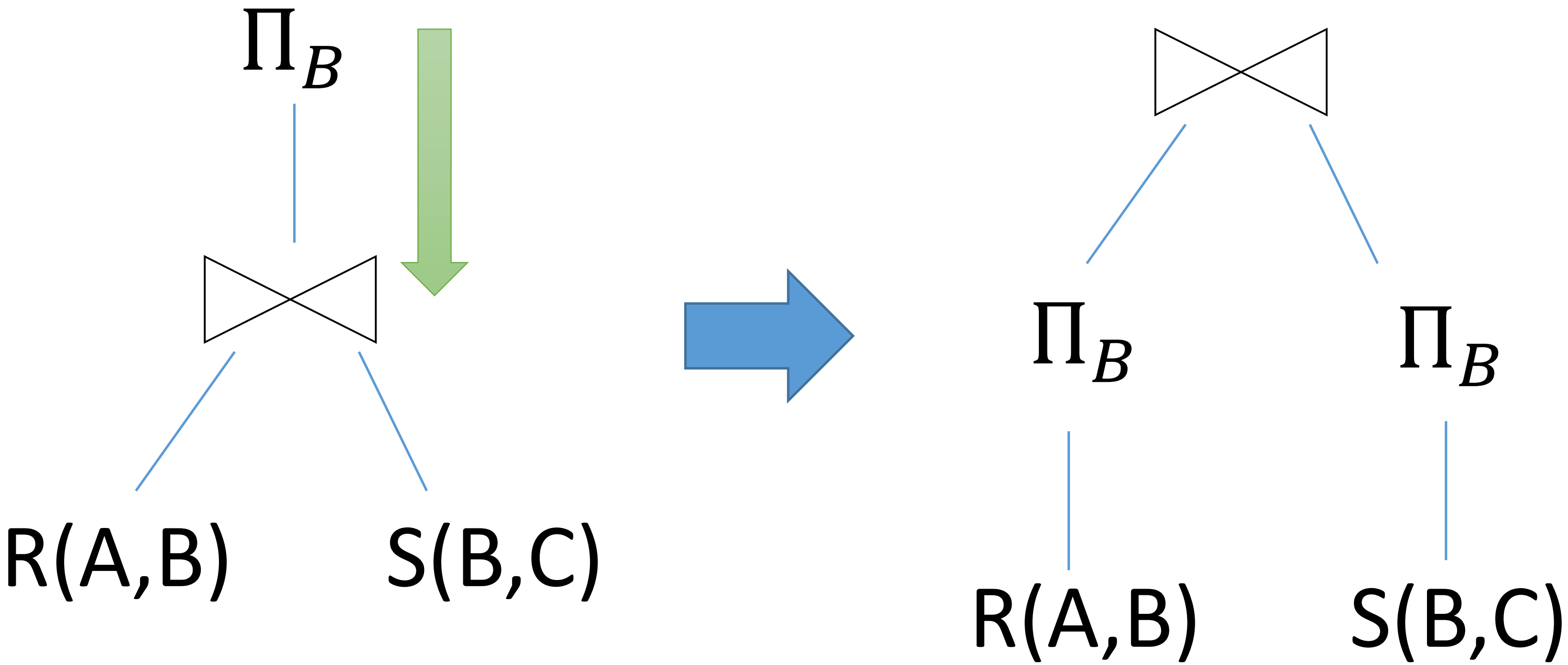
A simple plan



What SQL query does this correspond to?

Are there any logically equivalent RA expressions?

“Pushing down” projection



Why might we prefer this plan?

Takeaways

- This process is called logical optimization
- Many equivalent plans used to search for “good plans”
- Relational algebra is an important abstraction.

RA commutators

- The basic commutators:
  - Push **projection** through **(1) selection**, **(2) join**
  - Push **selection** through **(3) selection, (4) projection, (5) join**
  - *Also:* Joins can be re-ordered!

- Note that this is not an exhaustive set of operations
  - This covers *local re-writes; global re-writes possible but much harder*

This simple set of tools allows us to greatly improve the execution time of queries by optimizing RA plans!

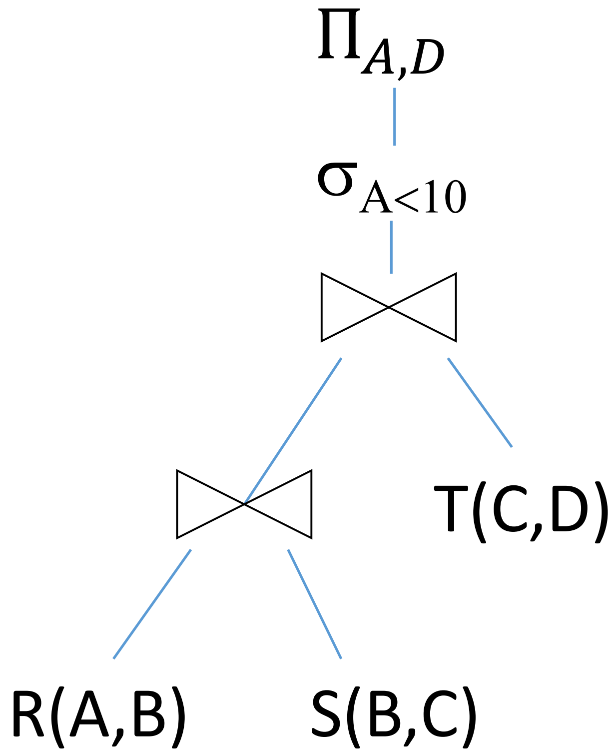
Optimizing the SFW RA Plan

Translating to RA

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

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

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



Logical Optimization

- Heuristically, we want selections and projections to occur as early as possible in the plan
  - Terminology: “push down **selections**” and “pushing down **projections.”**

- **Intuition:** We will have fewer tuples in a plan.
  - Could fail if the selection condition is very expensive (say runs some image processing algorithm).
  - Projection could be a waste of effort, but more rarely.

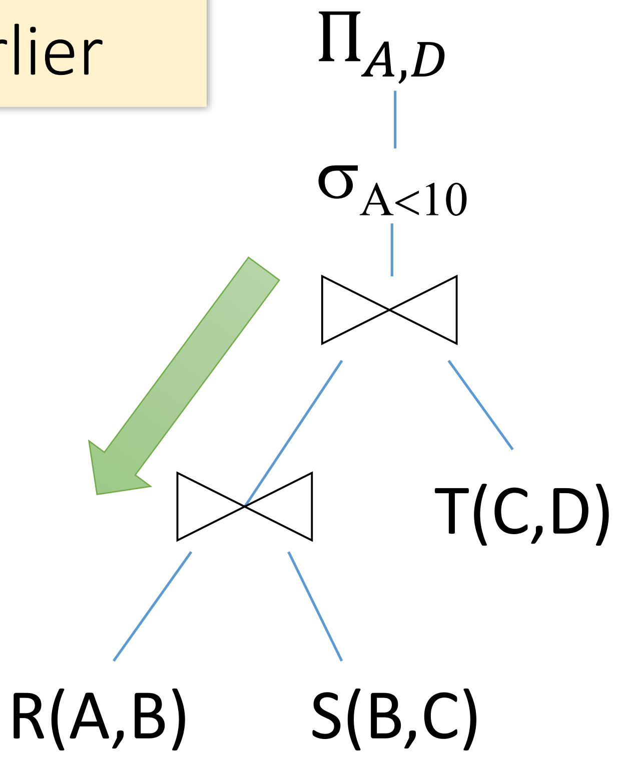
Optimizing RA Plan

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

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

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

Push down selection on A so it occurs earlier



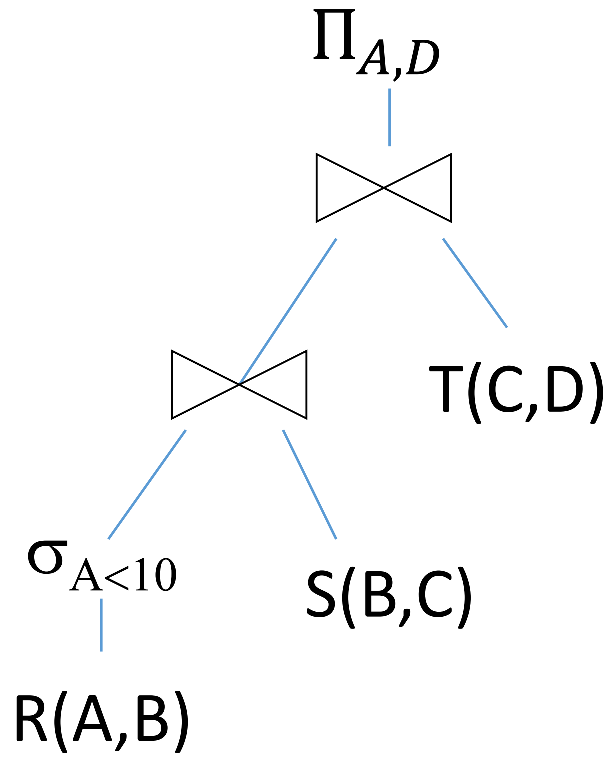
Optimizing RA Plan

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

```
SELECT R.A,S.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 selection on A so it occurs earlier



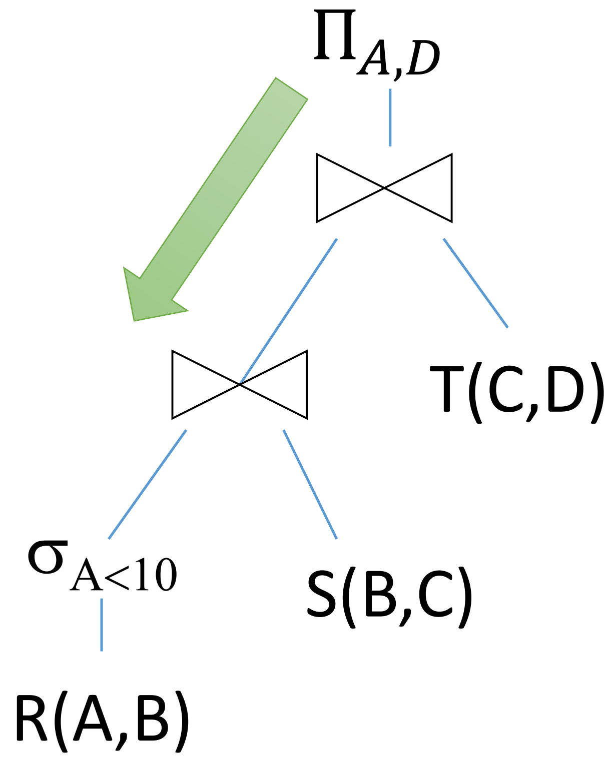
Optimizing RA Plan

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

```
SELECT R.A,S.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 projection so it occurs earlier



Optimizing RA Plan

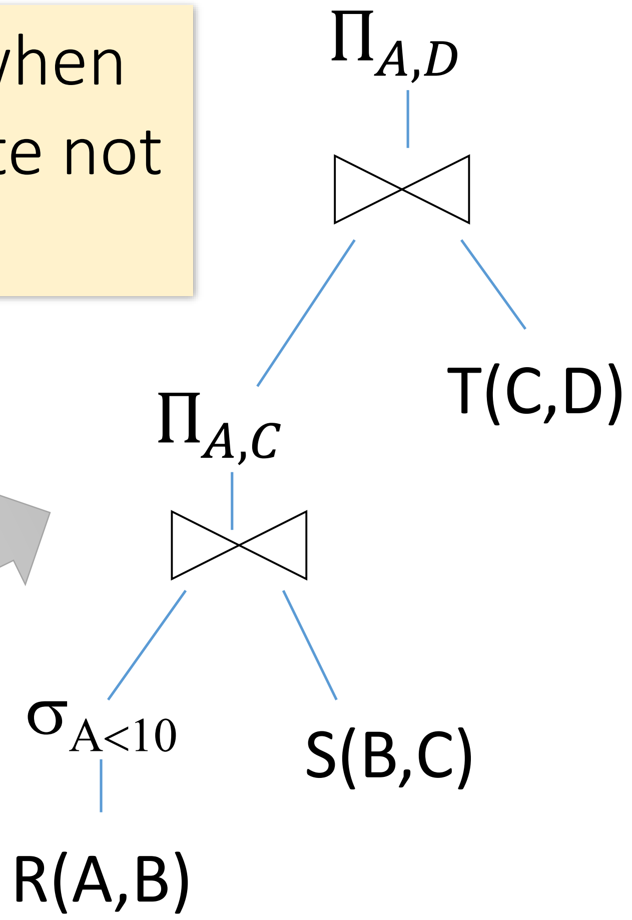
R(A,B) S(B,C) T(C,D)

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

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

We eliminate B earlier!

In general, when is an attribute not needed…?



Summary: R. A. Equivalences

- Allow us to choose different join orders and to "push" selections and projections ahead of joins.
- Selections:
  - $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ (*cascade*)
  - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (*commute*)
- Projections:
  - $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{a1, \ldots, an-1}(R))\ldots)$ (*cascade*)
- Cartesian Product
  - $R \times (S \times T) \equiv (R \times S) \times T$ (associative)
  - $R \times S \equiv S \times R$ (commutative)
  - *This means we can do joins in any order.*
    - But…beware of cartesian product!
    - $(R \bowtie S) \bowtie T \equiv (R \times T) \bowtie S$

More R. A. Equivalences

- Eager projection
  - Can cascade and "push" (some) projections thru selection
  - Can cascade and "push" (some) projections below one side of a join
  - Rule of thumb: can project anything not needed "downstream"
- $\pi_{a1}(\sigma_{c1}(R)) \equiv \pi_{a1}(\sigma_{c1}(\pi_{a1,c1} (R)))$
- Selection on a cross-product is equivalent to a join.
  - If selection is comparing attributes from each side
  - E.g. $\sigma_{R.a=S.b}(R \times S) \equiv R \bowtie_{R.a=S.b} S$
- A selection only on attributes of R commutes with $R \bowtie S$.
  - i.e., $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie S$
  - but only if the selection doesn't refer to S!

B. Physical Optimization

Index Selection

Input:

- Schema of the database
- **Workload description:** set of (query template, frequency) pairs

Goal: Select a set of indexes that minimize execution time of the workload.

- Cost / benefit balance: Each additional index may help with some queries, but requires updating

This is an optimization problem!

Example

Workload description:

```
SELECT pname
FROM Product
WHERE year = ? AND category = ?
```

Frequency 10,000,000

```
SELECT pname,
FROM Product
WHERE year = ? AND Category = ?
AND manufacturer = ?
```

Frequency 10,000,000

Which indexes might we choose?

Example

Workload description:

```
SELECT pname
FROM Product
WHERE year = ? AND category =?
```

Frequency 10,000,000

```
SELECT pname
FROM Product
WHERE year = ? AND Category =?
AND manufacturer = ?
```

Frequency 100

Now which indexes might we choose? Worth keeping an index with manufacturer in its search key around?

Simple Heuristic

- Can be framed as standard optimization problem: Estimate how cost changes when we add index.
  - We can ask the optimizer!
- Search over all possible space is too expensive, optimization surface is really nasty.
  - Real DBs may have 1000s of tables!
- Techniques to exploit *structure* of the space.
  - In SQLServer Autoadmin.

NP-hard problem, but can be solved!

Estimating index cost?

- Note that to frame as optimization problem, we first need an estimate of the ***cost*** of an index lookup

- Need to be able to estimate the costs of different indexes / index types…

We will see this mainly depends on getting estimates of result set size!

Ex: Clustered vs. Unclustered

Cost to do a range query for M entries over N-page file (P per page):

- Clustered:
  - To traverse: $\text{Log}_f(1.5N)$
  - To scan: 1 random IO + $\left\lceil \frac{M-1}{P} \right\rceil$ sequential IO
- Unclustered:
  - To traverse: $\text{Log}_f(1.5N)$
  - To scan: ~ M random IO

Suppose we are using a B+ Tree index with:

- Fanout f
- Fill factor 2/3

Plugging in some numbers

To simplify:
- Random IO = ~10ms
- Sequential IO = free

- Clustered:
  - To traverse: $Log_F(1.5N)$
  - **To scan: 1 random IO +** $\left\lceil \frac{M-1}{P} \right\rceil$ **sequential IO**

~ 1 random IO = 10ms

- Unclustered:
  - To traverse: $Log_F(1.5N)$
  - **To scan: ~ M random IO**

~ *M* random IO = M*10ms

- If M = 1, then there is no difference!
- If M = 100,000 records, then difference is ~10min. Vs. 10ms!

If only we had good estimates of M…

Histograms & IO Cost Estimation

IO Cost Estimation via Histograms

- For **index selection**:
  - What is the cost of an index lookup?

- Also for **deciding which algorithm to use**:
  - Ex: To execute $\mathrm{R} \bowtie S$, which join algorithm should DBMS use?

  - **What if we want to compute $\boldsymbol{\sigma_{A>10}(\mathrm{R}) \bowtie \sigma_{B=1}(S)}$?**

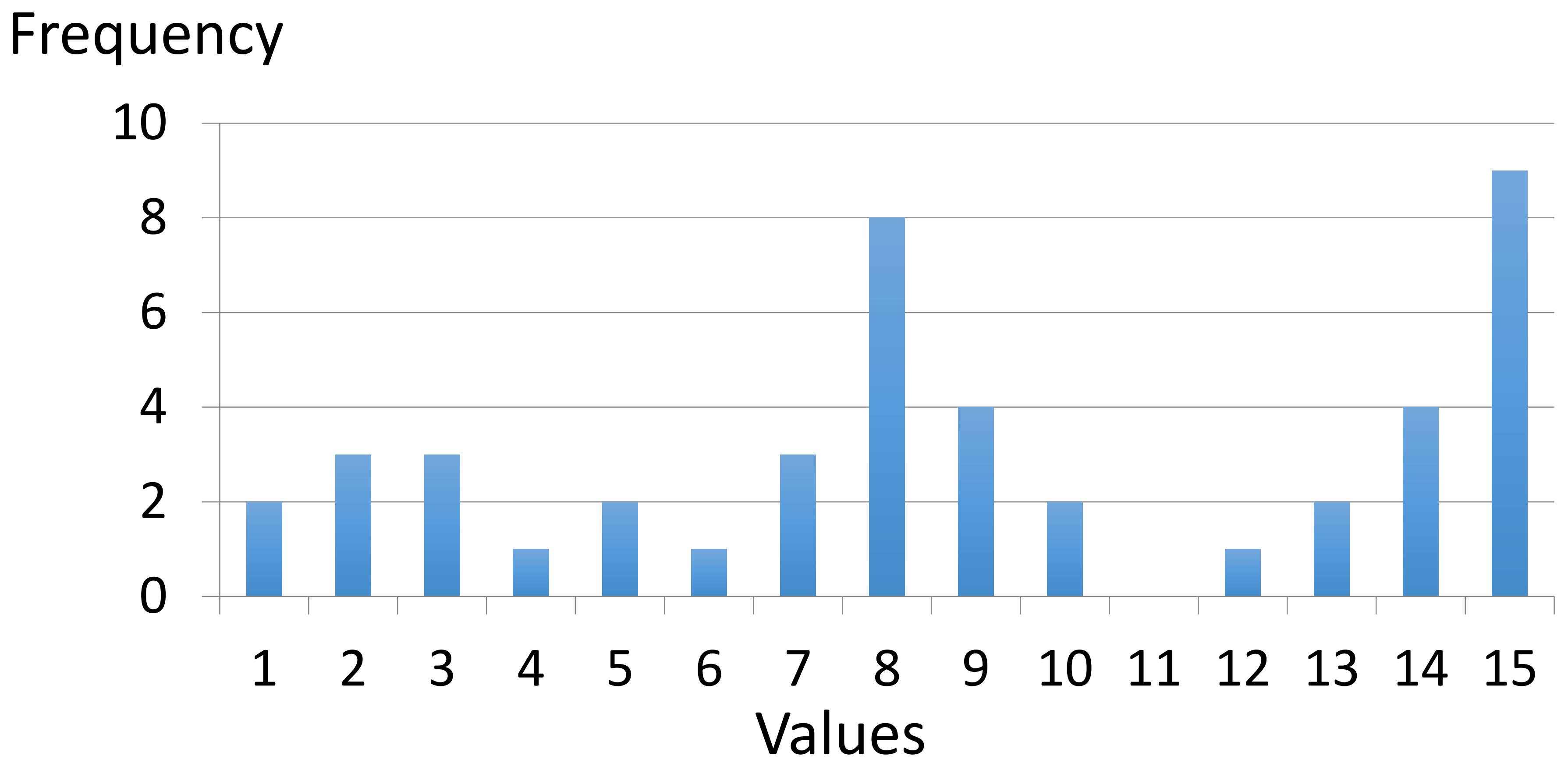
- In general, we will need some way to ***estimate intermediate result set sizes***

Histograms provide a way to efficiently store estimates of these quantities

Histograms

- A histogram is a set of value ranges ("buckets") and the frequencies of values in those buckets occurring
- How to choose the buckets?
    - Equiwidth & Equidepth
- Turns out high-frequency values are **very** important

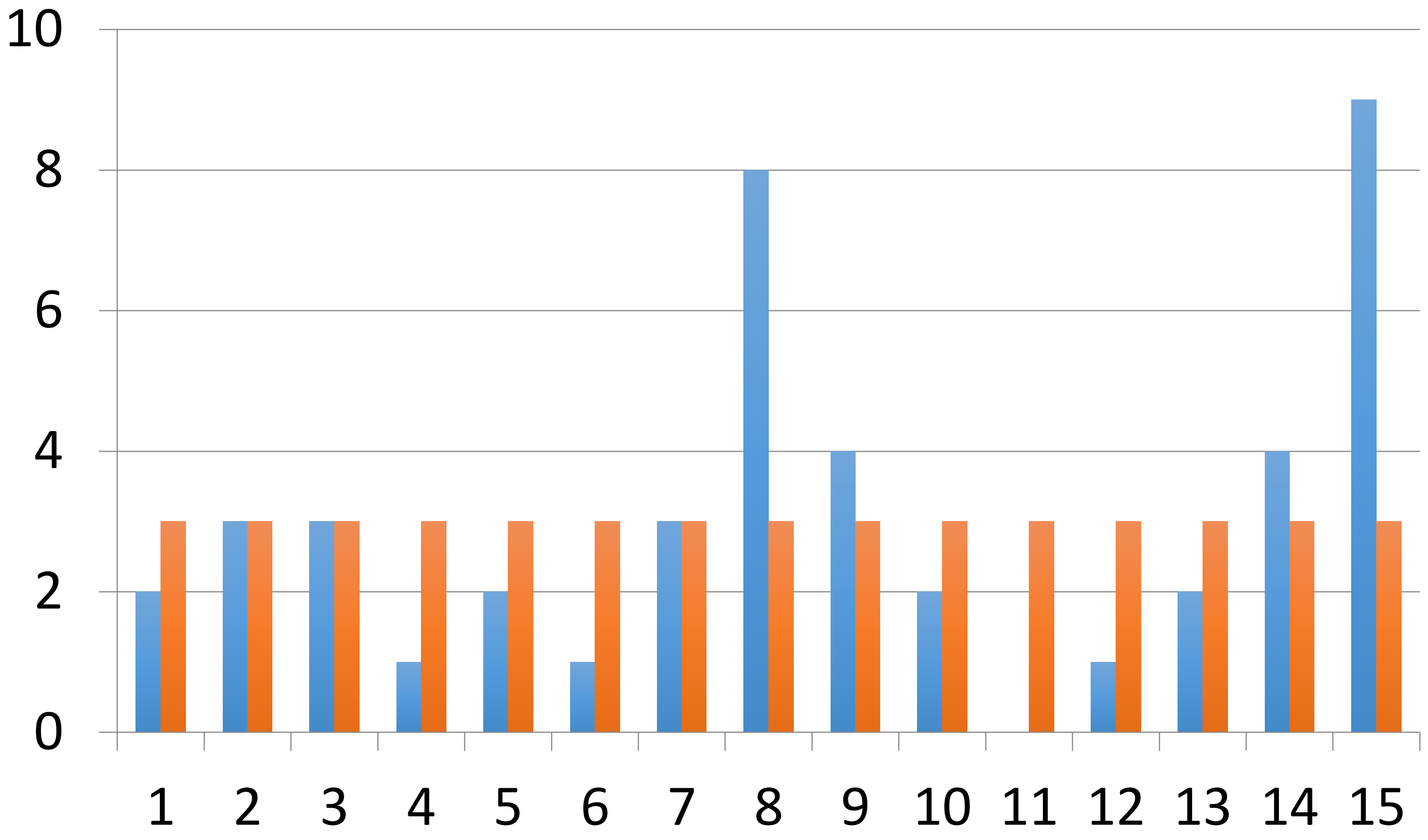
Example



How do we compute how many values between 8 and 10?
(Yes, it's obvious)

Problem: counts take up too much space!

Full vs. Uniform Counts



How much space do the full counts (bucket_size=1) take?

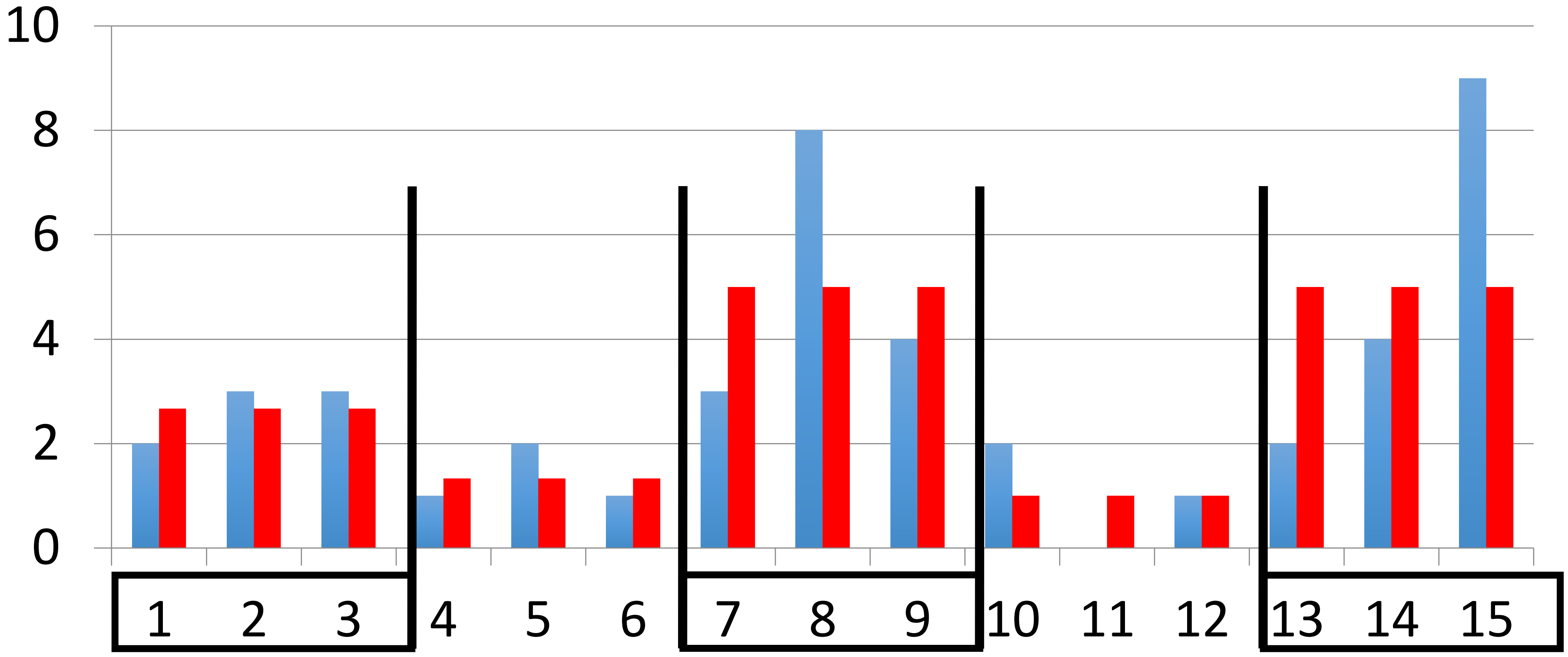
How much space do the uniform counts (bucket_size=ALL) take?

Fundamental Tradeoffs

- Want high resolution (like the full counts)
- Want low space (like uniform)
- Histograms are a compromise!

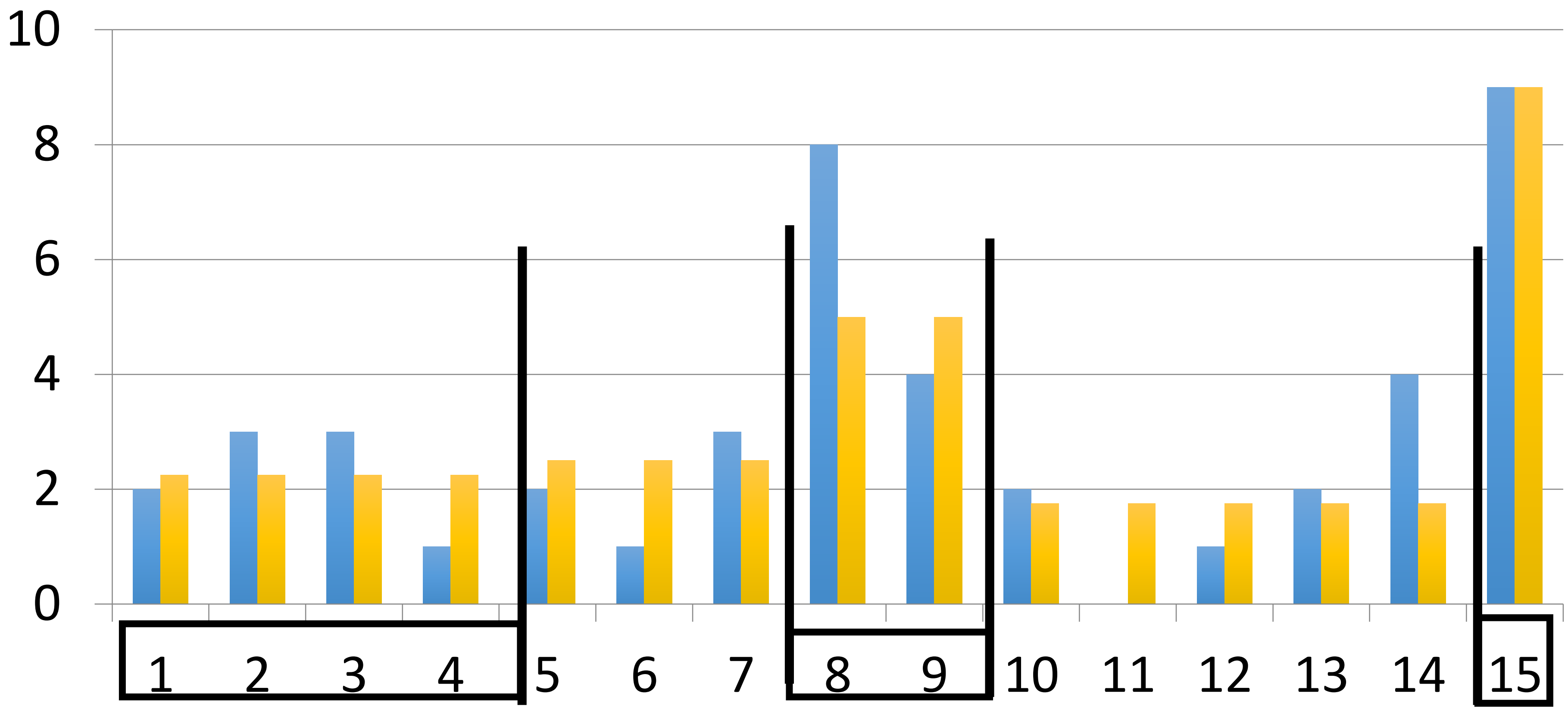
So how do we compute the “bucket” sizes?

Equi-width



All buckets roughly the same width

Equidepth



All buckets contain roughly the same number of items (total frequency)

Histograms

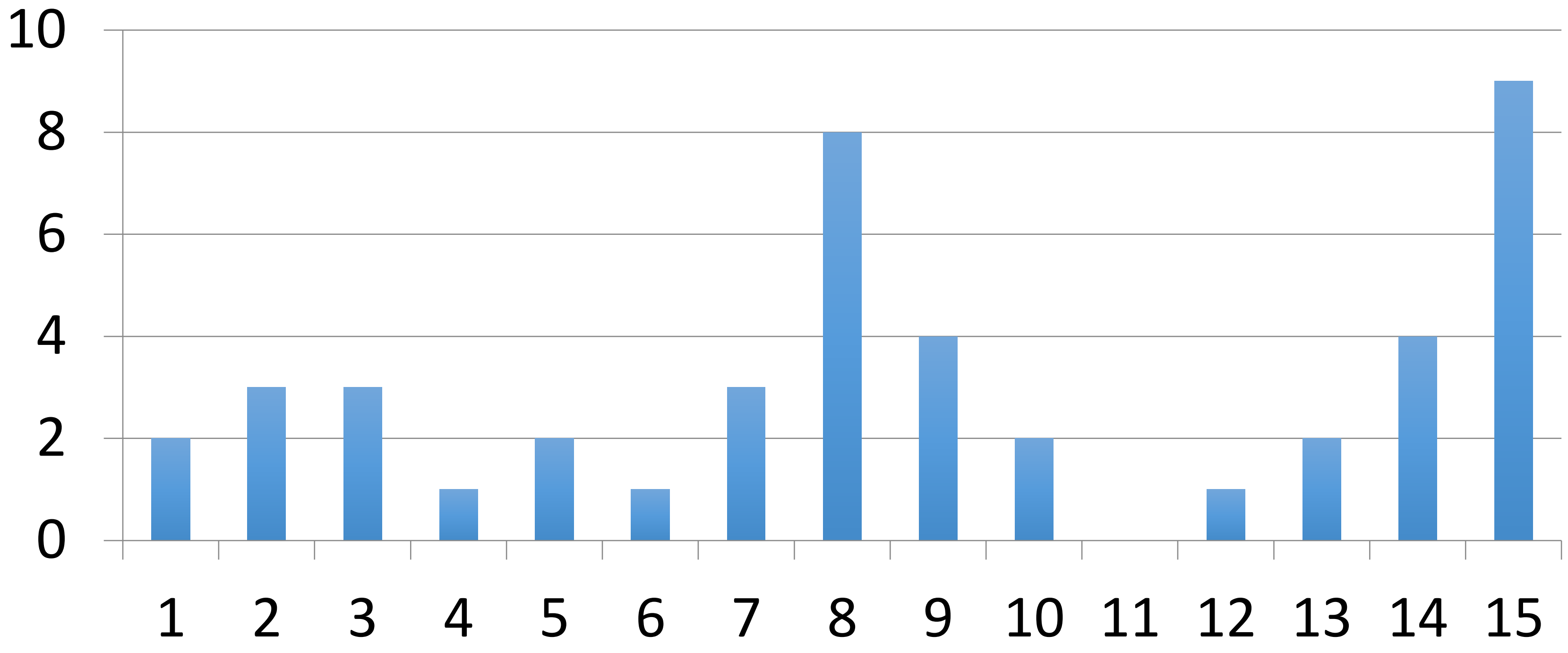
- Simple, intuitive and popular
- Parameters: # of buckets and type
- Can extend to many attributes (multidimensional)

Maintaining Histograms

- Histograms require that we update them!
  - Typically, you must run/schedule a command to update statistics on the database
  - Out of date histograms can be terrible!

- There is research work on self-tuning histograms and the use of query feedback
  - Oracle 11g

Nasty example



1. we insert many tuples with value > 16
2. we do **not** update the histogram
3. we ask for values > 20?

Compressed Histograms

- One popular approach:
  1. Store the most frequent values and their counts explicitly
  2. Keep an equiwidth or equidepth one for the rest of the values

People continue to try all manner of fanciness here
wavelets, graphical models, entropy models,…