Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#15: Query Optimization

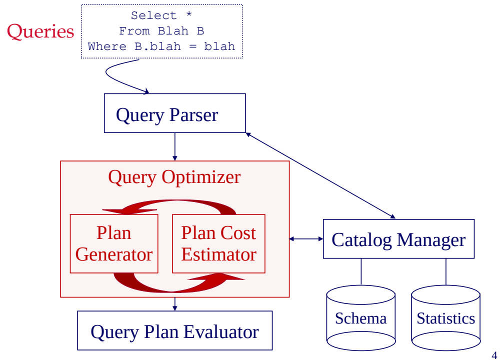
Last Class

- Set Operations
- Aggregate Operations
- Explain

Today's Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration
- Nested Sub-queries

Cost-based Query Sub-System



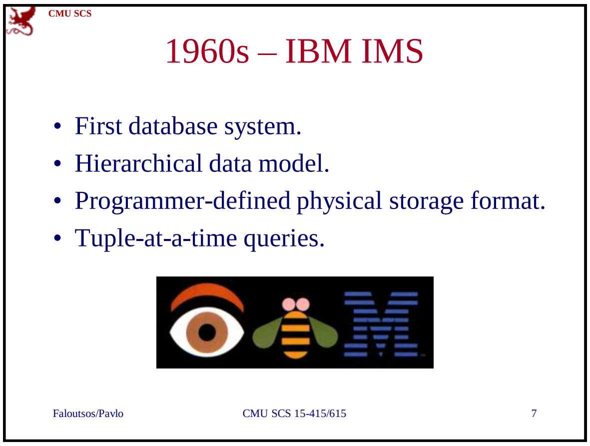
Queries

```
Select *
From Blah B
Where B.blah = blah
```

Query Optimization

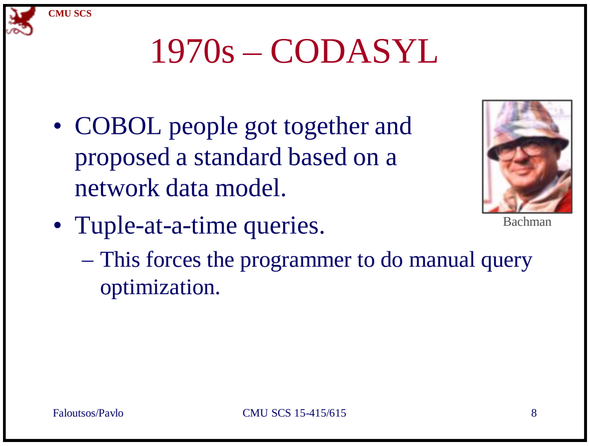
- Remember that SQL is declarative.
  - User tells the DBMS *what* answer they want, not *how* to get the answer.
- There can be a big difference in performance based on plan is used:
  - See last week: 5.7 days vs. 45 seconds

Quick DB History Lesson



1960s – IBM IMS

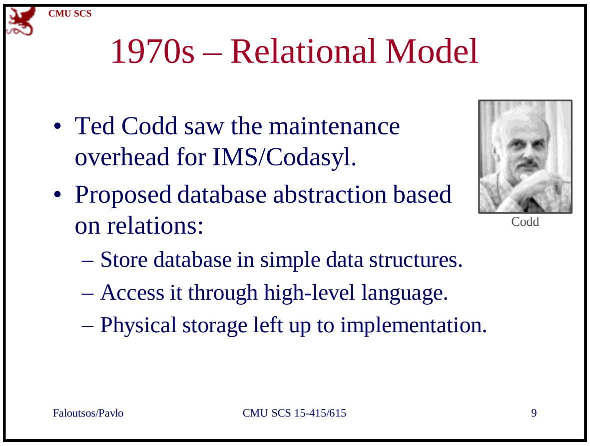
- First database system.
- Hierarchical data model.
- Programmer-defined physical storage format.
- Tuple-at-a-time queries.



1970s – CODASYL

- COBOL people got together and proposed a standard based on a network data model.
- Tuple-at-a-time queries.
  - – This forces the programmer to do manual query optimization.

Bachman



1970s – Relational Model

- Ted Codd saw the maintenance overhead for IMS/Codasyl.
- Proposed database abstraction based on relations:
  - – Store database in simple data structures.
  - – Access it through high-level language.
  - – Physical storage left up to implementation.

Codd

IBM System R

- Skunkworks project at IBM Research in San Jose to implement Codd's ideas.
- Had to figure out all of the things that we are discussing in this course themselves.
- IBM never commercialized System R.

IBM System R

- First implementation of a query optimizer.
- People argued that the DBMS could never choose a query plan better than what a human could write.
- A lot of the concepts from System R's optimizer are still used today.

Sample Database

SAILORS

sid	sname	rating	age
1	Christos	999	45.0
3	Obama	50	52.0
2	Tupac	32	26.0
6	Bieber	10	19.0

RESERVES

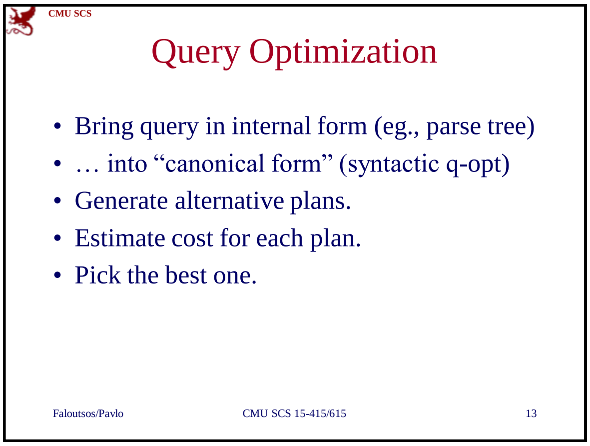
sid	bid	day	rname
6	103	2014-02-01	matlock
1	102	2014-02-02	macgyver
2	101	2014-02-02	a-team
1	101	2014-02-01	dallas

BOATS

bid	bname	color
101	The GZA	red
102	The RZA	white
103	Raekwon	green
104	O.D.B.	brown

Sailors(*sid*: int, *sname*: varchar, *rating*: int, *age*: real)
Reserves(*sid*: int, *bid*: int, *day*: date, *rname*: varchar)
Boats(*bid*: int, *bname*: varchar, *color*: varchar)

Hoofer Sailing Club

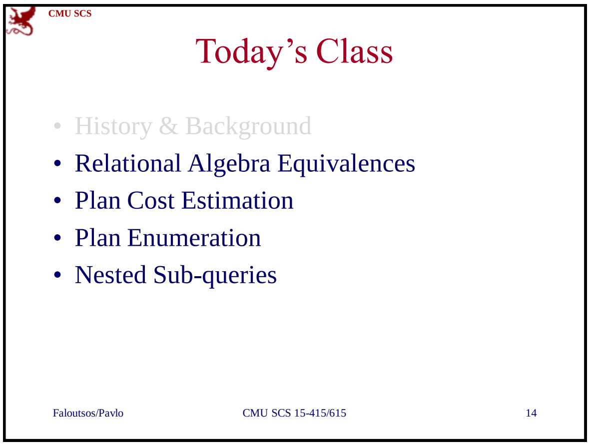


CMU SCS

Query Optimization

- Bring query in internal form (eg., parse tree)
- … into "canonical form" (syntactic q-opt)
- Generate alternative plans.
- Estimate cost for each plan.
- Pick the best one.

Faloutsos/Pavlo CMU SCS 15-415/615 13

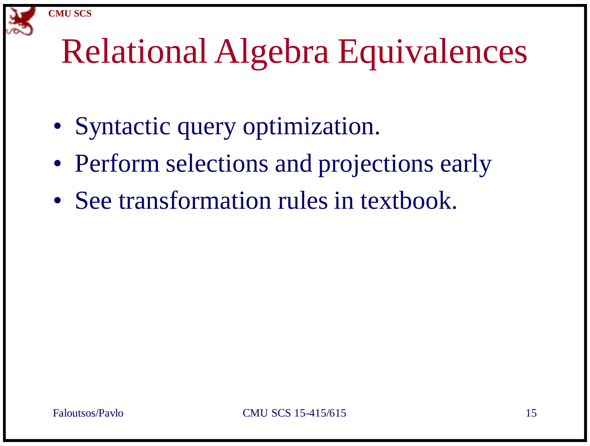


CMU SCS

Today's Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration
- Nested Sub-queries

Faloutsos/Pavlo CMU SCS 15-415/615 14

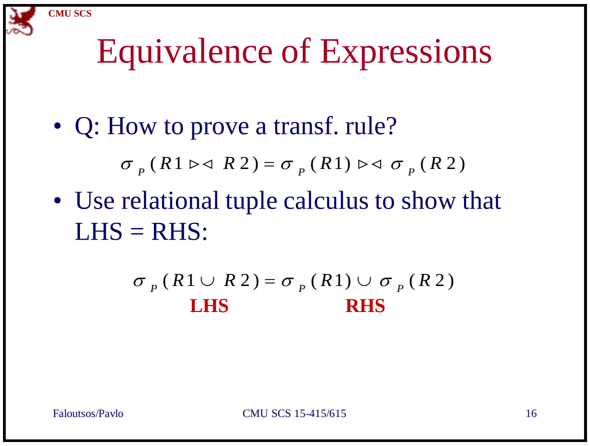


CMU SCS

Relational Algebra Equivalences

- Syntactic query optimization.
- Perform selections and projections early
- See transformation rules in textbook.

Faloutsos/Pavlo CMU SCS 15-415/615 15



CMU SCS

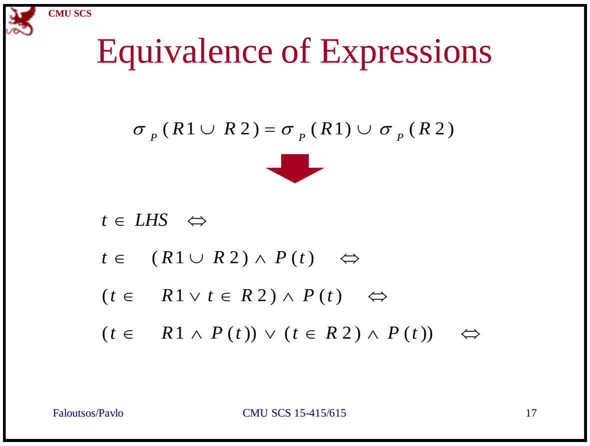
Equivalence of Expressions

- Q: How to prove a transf. rule?

$$\sigma_P(R1 \bowtie R2) = \sigma_P(R1) \bowtie \sigma_P(R2)$$

- Use relational tuple calculus to show that LHS = RHS:

$$\underbrace{\sigma_P(R1 \cup R2)}_{\textbf{LHS}} = \underbrace{\sigma_P(R1) \cup \sigma_P(R2)}_{\textbf{RHS}}$$



CMU SCS

Equivalence of Expressions

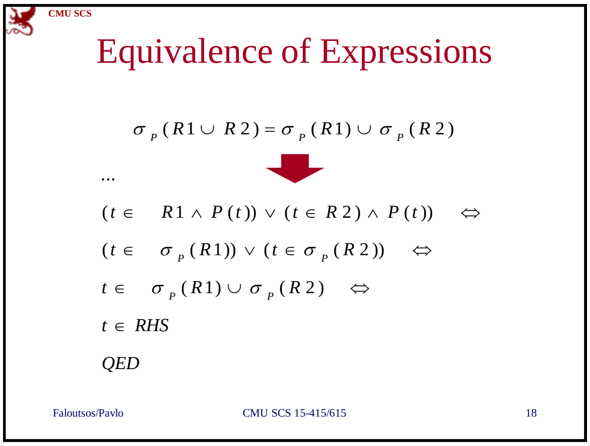
$$\sigma_P(R1 \cup R2) = \sigma_P(R1) \cup \sigma_P(R2)$$

$t \in LHS \quad \Leftrightarrow$

$t \in (R1 \cup R2) \wedge P(t) \quad \Leftrightarrow$

$(t \in R1 \vee t \in R2) \wedge P(t) \quad \Leftrightarrow$

$(t \in R1 \wedge P(t)) \vee (t \in R2) \wedge P(t)) \quad \Leftrightarrow$



CMU SCS

Equivalence of Expressions

$$\sigma_P(R1 \cup R2) = \sigma_P(R1) \cup \sigma_P(R2)$$

…

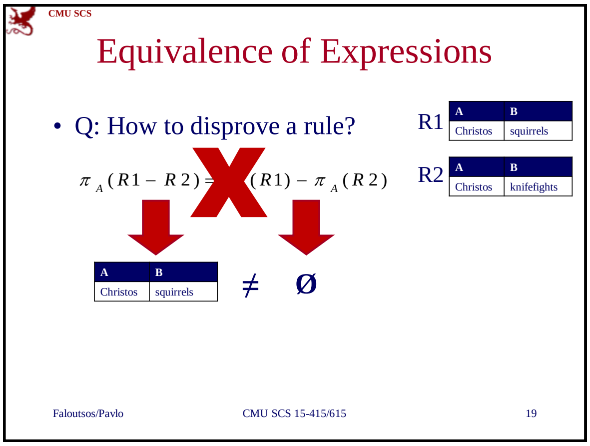
$(t \in R1 \wedge P(t)) \vee (t \in R2) \wedge P(t)) \quad \Leftrightarrow$

$(t \in \sigma_P(R1)) \vee (t \in \sigma_P(R2)) \quad \Leftrightarrow$

$t \in \sigma_P(R1) \cup \sigma_P(R2) \quad \Leftrightarrow$

$t \in RHS$

QED



CMU SCS

Equivalence of Expressions

- Q: How to disprove a rule?

$$\pi_A(R1 - R2) \cancel{=} \pi_A(R1) - \pi_A(R2)$$

R1

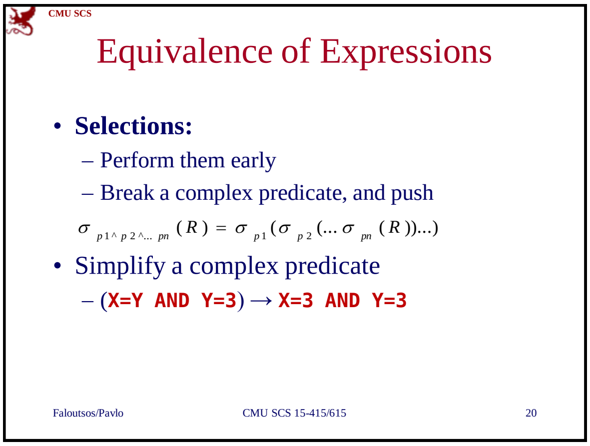
A	B
Christos	squirrels

R2

A	B
Christos	knifefights

A	B
Christos	squirrels

$\neq$ Ø



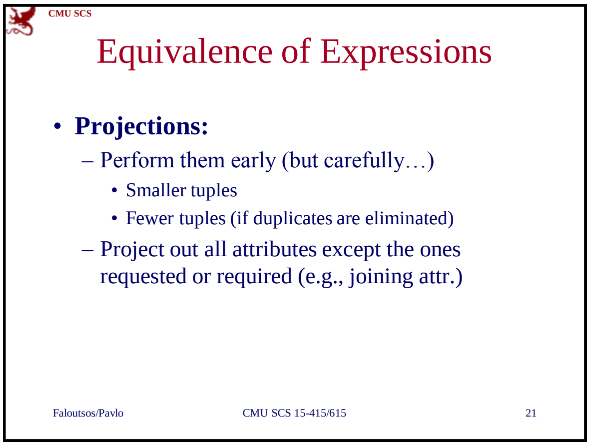
CMU SCS

Equivalence of Expressions

- **Selections:**
  - Perform them early
  - Break a complex predicate, and push

$$\sigma_{p1 \wedge p2 \wedge \ldots pn}(R) = \sigma_{p1}(\sigma_{p2}(\ldots \sigma_{pn}(R))\ldots)$$

- Simplify a complex predicate
  - (`X=Y AND Y=3`) → `X=3 AND Y=3`



CMU SCS

Equivalence of Expressions

- **Projections:**
  - Perform them early (but carefully…)
    - Smaller tuples
    - Fewer tuples (if duplicates are eliminated)
  - Project out all attributes except the ones requested or required (e.g., joining attr.)

Equivalence of Expressions

- **Joins:**
  - Commutative, associative

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

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

- Q: How many different orderings are there for an *n*-way join?

Equivalence of Expressions

- **Joins:** How many different orderings are there for an n-way join?
- A: Catalan number ~ 4^n
  - Exhaustive enumeration: too slow.

Query Optimization

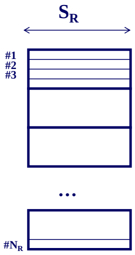
- Bring query in internal form (eg., parse tree)
- … into “canonical form” (syntactic q-opt)
- Generate alternative plans.
- ➡ Estimate cost for each plan.
- Pick the best one.

Cost Estimation

- How long will a query take?
  - **CPU**: Small cost; tough to estimate.
  - **Disk**: # of block transfers.
- How many tuples will qualify?
- What statistics do we need to keep?

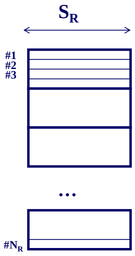
Cost Estimation – Statistics

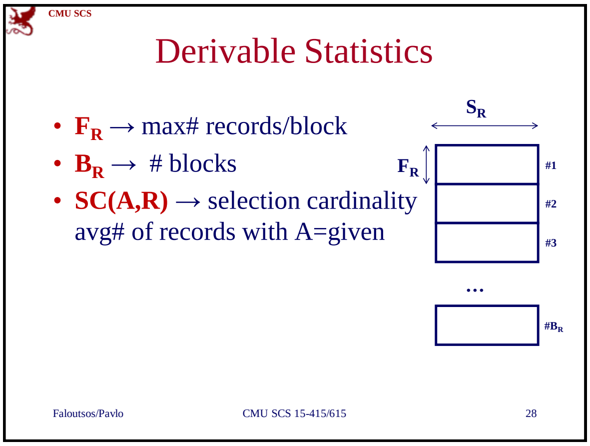
- For each relation **R** we keep:
  - **N_R** → # tuples;
  - **S_R** → size of tuple in bytes



Cost Estimation – Statistics

- For each relation **R** we keep:
  - **N_R** → # tuples;
  - **S_R** → size of tuple in bytes
  - **V(A,R)** → # of distinct values of attribute ‘A’
  - And histograms…





Derivable Statistics

- **F_R** → max# records/block
- **B_R** → # blocks
- **SC(A,R)** → selection cardinality avg# of records with A=given

Derivable Statistics

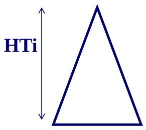
- **F_R** → max# records/block
  Blocking Factor → **B/S_R**, where B is the block size in bytes.
- **B_R** → # blocks → **N_R/F_R**

Derivable Statistics

- **SC(A,R)** → Selection Cardinality
  avg# of records with A=given
  → **N_R / V(A,R)**
- Note that this assumes data uniformity
  - 10,000 students, 10 colleges – how many students in SCS?

Additional Statistics

- For index **i**:
  - **F_i** → average fanout (~50-100)
  - **HT_i** → # levels of index **i** (~2-3) ~ log(#entries)/log(**F_i**)
  - **LB_i** # → blocks at leaf level



Statistics

- Where do we store them?
- How often do we update them?

Selection Statistics

- We saw simple predicates (‘`name=Christos`’)
- How about more complex predicates, like
  - ‘`salary > 10K`’
  - ‘`age=30 AND jobCode=“Gangstarr”` ’
- What is their selectivity?

Selections – Complex Predicates

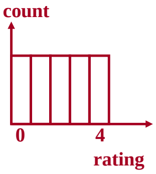
- Selectivity **sel(P)** of predicate **P**:

  == fraction of tuples that qualify

  sel(P) = **SC(P) / N_R**

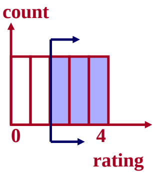
Selections – Complex Predicates

- Assume that **V(rating, SAILORS)** has 5 distinct values (i.e., 0 to 4).
- simple predicate **P**: A=constant
  - – **sel(A=constant)** = **1/V(A,R)**
  - – eg., **sel(rating=‘2’)** = 1/5
- What if **V(A,R)** is unknown??



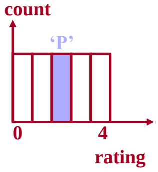
Selections – Complex Predicates

- Range Query: **sel(**rating >= ‘2’**)**
- **sel**(A>a) = $(A_{max} - a) / (A_{max} - A_{min})$



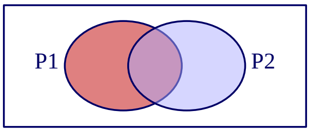
Selections – Complex Predicates

- Negation: **sel(**rating != ‘2’**)**
  - **sel(not** P**)** = **1** – **sel(**P**)**
- Observation: selectivity ≈ probability



Selections – Complex Predicates

- **Conjunction:**
  - **sel(**rating = ‘2’ **and** name LIKE ‘C%’**)**
  - **sel(**P1 ∧ P2**)** = **sel(**P1**)** · **sel(**P2**)**
  - INDEPENDENCE ASSUMPTION

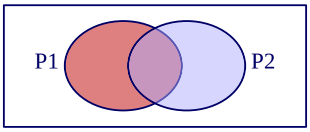


Selections – Complex Predicates

- **Disjunction**:
  - **sel(**rating = ‘2’ **or** name LIKE ‘C%’**)**
  - **sel(**P1 ∨ P2**)**
    = **sel(**P1**)** + **sel(**P2**)** – **sel(**P1 ∨ P2**)**
    = **sel(**P1**)** + **sel(**P2**)** – **sel(**P1**)** · **sel(**P2**)**
  - INDEPENDENCE ASSUMPTION, again

Selections – Complex Predicates

- **Disjunction, in general**:
  - **sel(**P1 **or** P2 **or** … Pn) =
  - 1 - (1- **sel(**P1)) · (1 - **sel(**P2)) · … (1 - **sel(**Pn))



Selections – Summary

- sel(A=constant) → 1/V(A,r)
- sel(A>a) → (A_{max} – a) / (A_{max} – A_{min})
- sel(**not** P) → 1 – sel(P)
- sel(P1 **and** P2) → sel(P1) · sel(P2)
- sel(P1 **or** P2) → sel(P1) + sel(P2) – sel(P1) · sel(P2)
- sel(P1 **or** ... **or** Pn) = 1 - (1-sel(P1)) · ... · (1-sel(Pn))

Joins

- Q: Given a join of **R** and **S**, what is the range of possible results sizes in #of tuples?
  - Hint: what if $\mathbf{R_{cols}} \cap \mathbf{S_{cols}}$ = **Ø**?
  - $\mathbf{R_{cols}} \cap \mathbf{S_{cols}}$ is a key for **R** and a foreign key in **S**?

Joins

- Q: Given a join of **R** and **S**, v range of possible result sizes in of tuples?
  - Hint: what if $\mathbf{R_{cols}} \cap \mathbf{S_{cols}} = \emptyset$?
  - $\mathbf{R_{cols}} \cap \mathbf{S_{cols}}$ is a key for **R** and a foreign key in **S**?

$\mathbf{N_R \cdot N_S}$

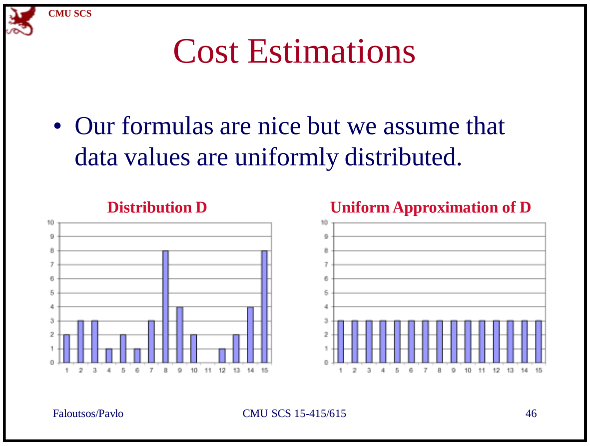
$\leq \mathbf{N_S}$

Result Size Estimation for Joins

- General case: $\mathbf{R_{cols}} \cap \mathbf{S_{cols}}$ = {A} where A is not a key for either table.
- *Hint: for a given tuple of R, how many tuples of S will it match?*

Result Size Estimation for Joins

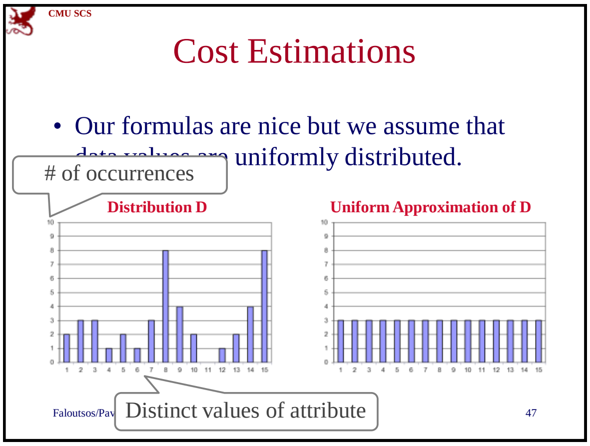
- General case: $\mathbf{R_{cols}} \cap \mathbf{S_{cols}}$ = {A} where A is not a key for either table.
  - Match each **R**-tuple with **S**-tuples: estSize ≈ $\mathbf{N_R \cdot N_S / V(A,S)}$
  - Symmetrically, for **S**: estSize ≈ $\mathbf{N_R \cdot N_S / V(A,R)}$
- Overall:
  - estSize ≈ $\mathbf{N_R \cdot N_S / max(\{V(A,S), V(A,R)\})}$



CMU SCS

Cost Estimations

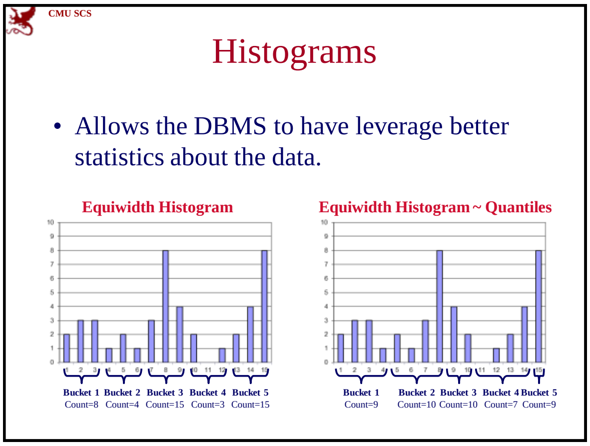
- Our formulas are nice but we assume that data values are uniformly distributed.



CMU SCS

Cost Estimations

- Our formulas are nice but we assume that data values are uniformly distributed.



CMU SCS

Histograms

- Allows the DBMS to have leverage better statistics about the data.

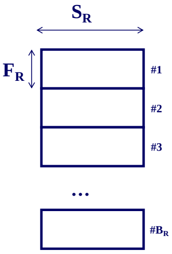
Query Optimization

- Bring query in internal form (eg., parse tree)
- … into “canonical form” (syntactic q-opt)
- ➡ Generate alternative plans.
  - Single relation.
  - Multiple relations.
- Estimate cost for each plan.
- Pick the best one.

Plan Generation

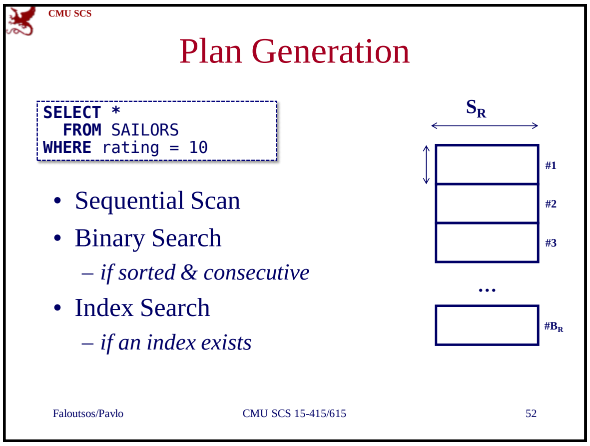
```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- What are our plan options?



Reminder

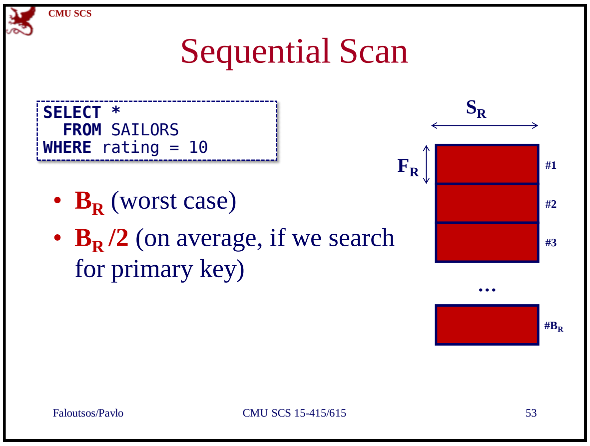
	Scan	Eq	Range	Ins	Del
Heap	B	B/2	B	2	Search+1
sorted	B	$\log_2 B$	<- +m	Search+B	Search+B
Clust.	1.5B	h	<- +m	Search+1	Search+1
u-tree	~B	1+h’	<- +m’	Search+2	Search+2
u-hash	~B	~2	B	Search+2	Search+2



Plan Generation

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

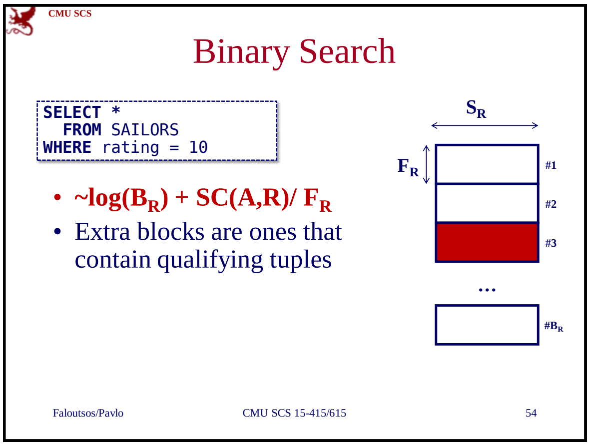
- Sequential Scan
- Binary Search
  - *if sorted & consecutive*
- Index Search
  - *if an index exists*



Sequential Scan

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- B_R (worst case)
- $B_R/2$ (on average, if we search for primary key)



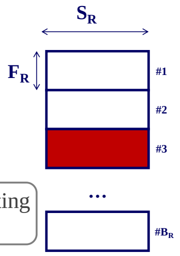
Binary Search

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- $\sim\log(B_R) + SC(A,R)/ F_R$
- Extra blocks are ones that contain qualifying tuples

Binary Search

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

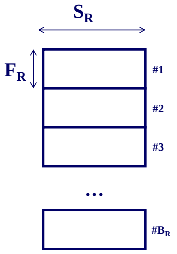


- **$\sim\log(B_R) + SC(A,R)/ F_R$**
- Extra blocks a ones that contain qualif tuples

We showed that estimating this is non-trivial.

Index Search

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```



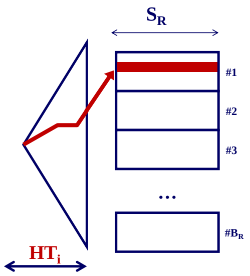
- Index Search:
  - levels of index + blocks w/ qual. tuples

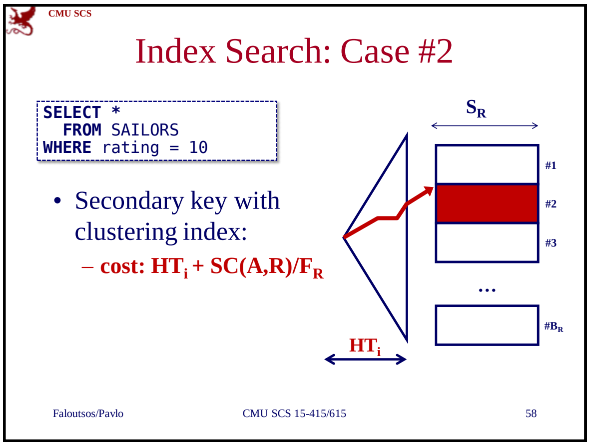
Case#1: Primary Key
Case#2: Secondary key – clustering index
Case#3: Secondary key – non-clust. index

Index Search: Case #1

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- Primary Key
  - **cost: $HT_i + 1$**

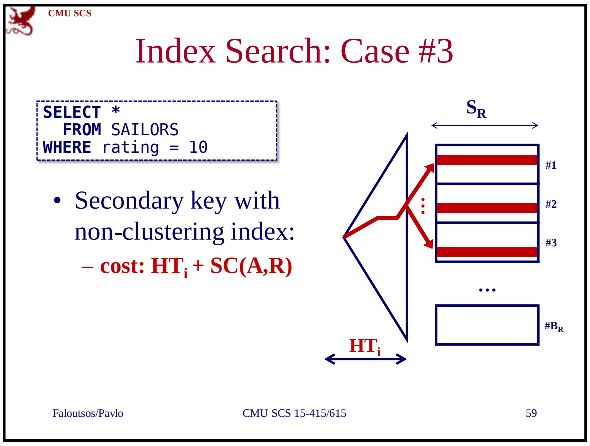




Index Search: Case #2

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- Secondary key with clustering index:
  - **cost: $HT_i + SC(A,R)/F_R$**



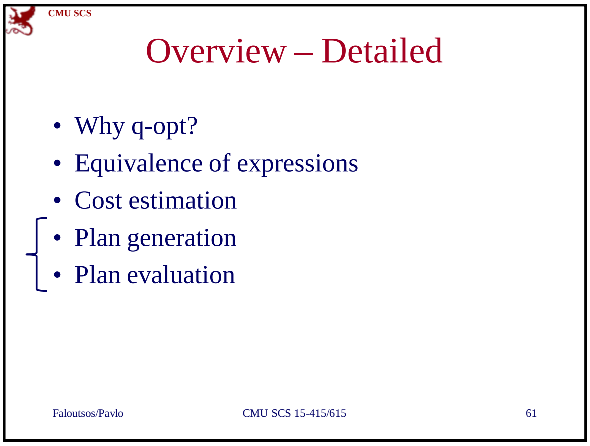
Index Search: Case #3

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- Secondary key with non-clustering index:
  - **cost: $HT_i + SC(A,R)$**

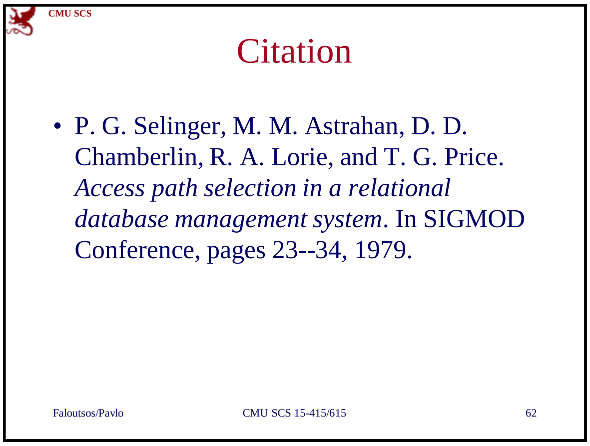
Single Relation Plans

- With no index: scan (dup-elim; sort)
- With index:
  - Single index access path
  - Multiple index access path
  - Sorted index access path
  - Index-only access path



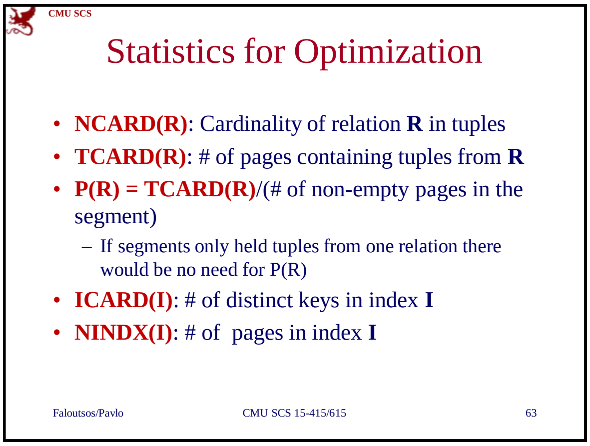
Overview – Detailed

- Why q-opt?
- Equivalence of expressions
- Cost estimation
- Plan generation
- Plan evaluation



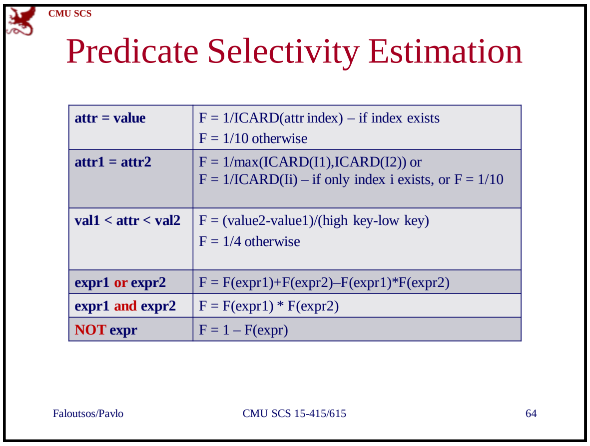
Citation

- P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. *Access path selection in a relational database management system*. In SIGMOD Conference, pages 23--34, 1979.



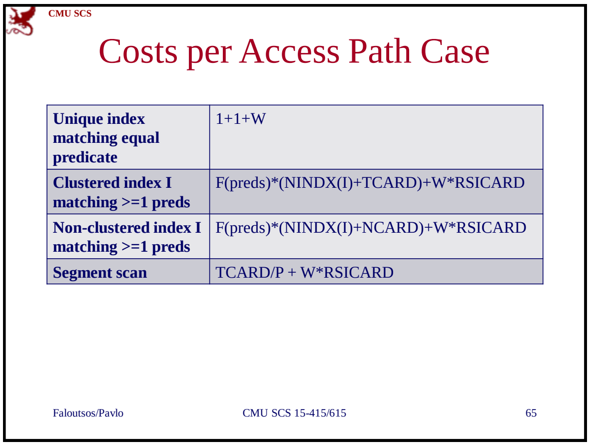
Statistics for Optimization

- **NCARD(R)**: Cardinality of relation **R** in tuples
- **TCARD(R)**: # of pages containing tuples from **R**
- **P(R) = TCARD(R)**/(# of non-empty pages in the segment)
  - If segments only held tuples from one relation there would be no need for P(R)
- **ICARD(I)**: # of distinct keys in index **I**
- **NINDX(I)**: # of pages in index **I**



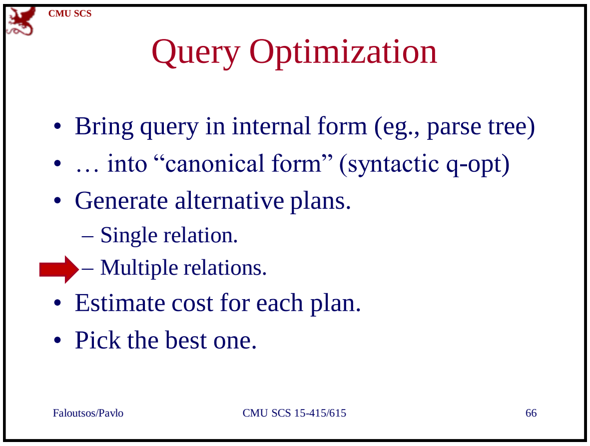
Predicate Selectivity Estimation

attr = value	F = 1/ICARD(attr index) – if index exists F = 1/10 otherwise
attr1 = attr2	F = 1/max(ICARD(I1),ICARD(I2)) or F = 1/ICARD(Ii) – if only index i exists, or F = 1/10
val1 < attr < val2	F = (value2-value1)/(high key-low key) F = 1/4 otherwise
expr1 or expr2	F = F(expr1)+F(expr2)–F(expr1)*F(expr2)
expr1 and expr2	F = F(expr1) * F(expr2)
NOT expr	F = 1 – F(expr)



Costs per Access Path Case

Unique index matching equal predicate	1+1+W
Clustered index I matching >=1 preds	F(preds)*(NINDX(I)+TCARD)+W*RSICARD
Non-clustered index I matching >=1 preds	F(preds)*(NINDX(I)+NCARD)+W*RSICARD
Segment scan	TCARD/P + W*RSICARD



Query Optimization

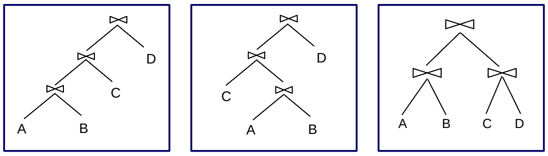
- Bring query in internal form (eg., parse tree)
- … into “canonical form” (syntactic q-opt)
- Generate alternative plans.
  - Single relation.
  - ➡ Multiple relations.
- Estimate cost for each plan.
- Pick the best one.

Queries over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly
  - We need to restrict search space.
- **Fundamental decision in System R:** only left-deep join trees are considered.

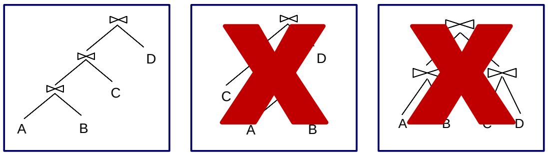
Queries over Multiple Relations

- **Fundamental decision in System R:** only left-deep join trees are considered.



Queries over Multiple Relations

- **Fundamental decision in System R:** only left-deep join trees are considered.



Queries over Multiple Relations

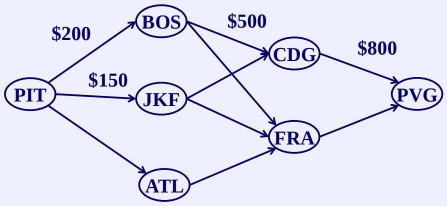
- **Fundamental decision in System R:** only left-deep join trees are considered.
  - Allows for fully pipelined plans where intermediate results not written to temp files.
  - Not all left-deep trees are fully pipelined (e.g., SM join).

Queries over Multiple Relations

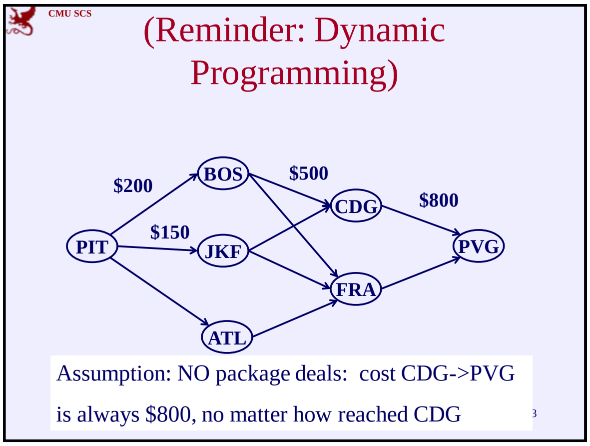
- Enumerate the orderings (= left deep tree)
- Enumerate the plans for each operator
- Enumerate the access paths for each table

- Use **dynamic programming** to save cost estimations.

(Reminder: Dynamic Programming)

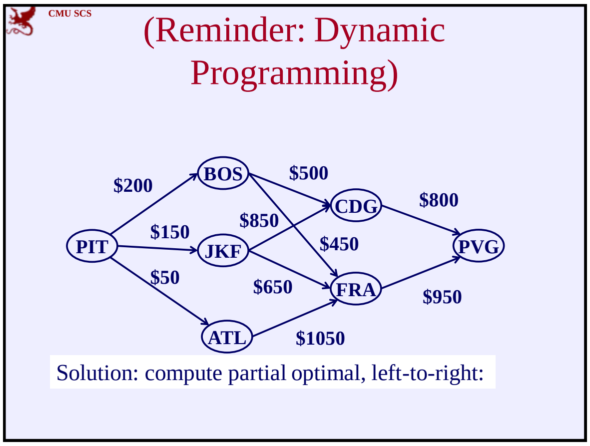


Cheapest flight PIT -> PVG?



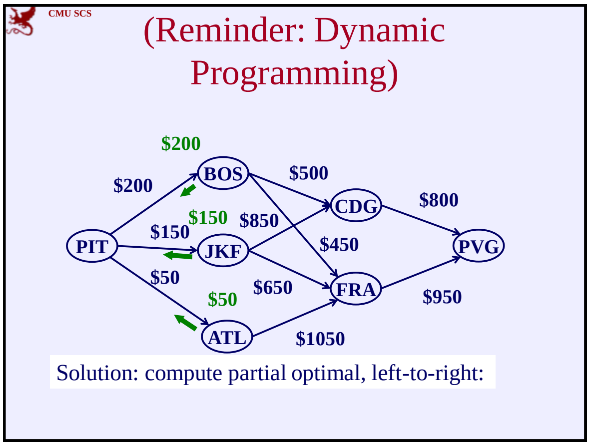
(Reminder: Dynamic Programming)

Assumption: NO package deals: cost CDG->PVG is always $800, no matter how reached CDG



(Reminder: Dynamic Programming)

Solution: compute partial optimal, left-to-right:

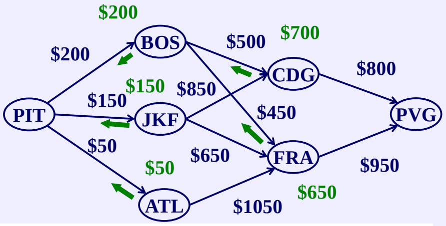


(Reminder: Dynamic Programming)

Solution: compute partial optimal, left-to-right:

CMU SCS

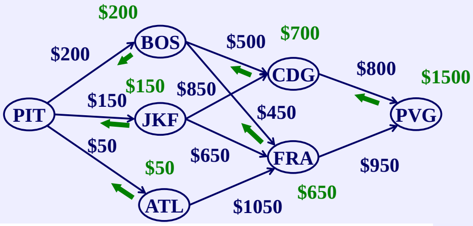
(Reminder: Dynamic Programming)



Solution: compute partial optimal, left-to-right:

CMU SCS

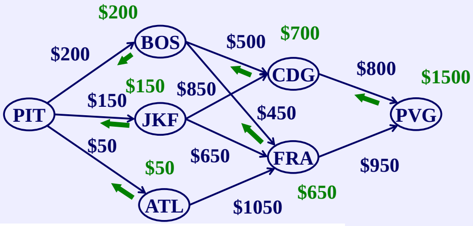
(Reminder: Dynamic Programming)



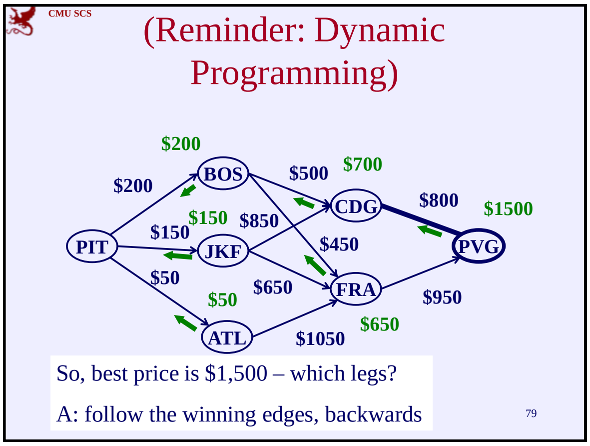
Solution: compute partial optimal, left-to-right:

CMU SCS

(Reminder: Dynamic Programming)



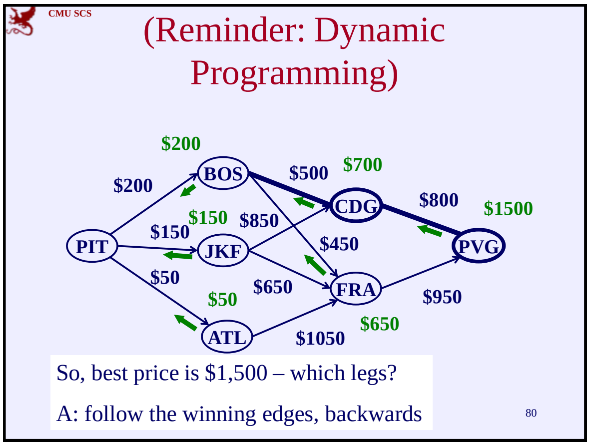
So, best price is $1,500 – which legs?


CMU SCS

(Reminder: Dynamic Programming)

So, best price is $1,500 – which legs?

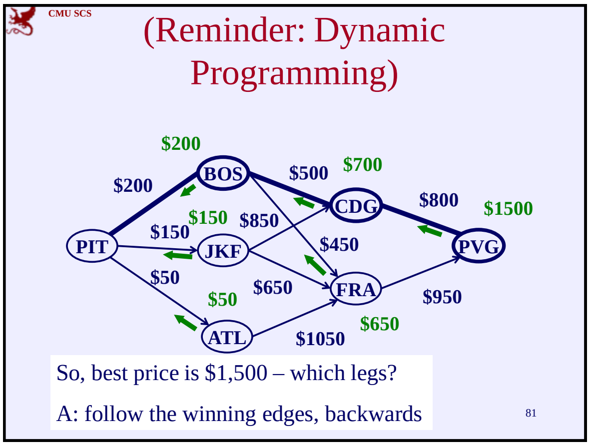
A: follow the winning edges, backwards


CMU SCS

(Reminder: Dynamic Programming)

So, best price is $1,500 – which legs?

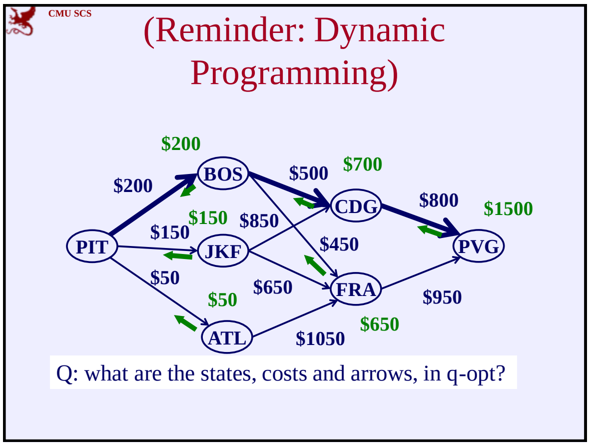
A: follow the winning edges, backwards


CMU SCS

(Reminder: Dynamic Programming)

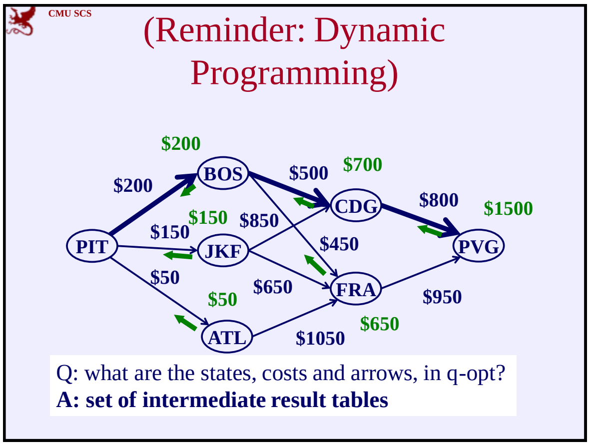
So, best price is $1,500 – which legs?

A: follow the winning edges, backwards



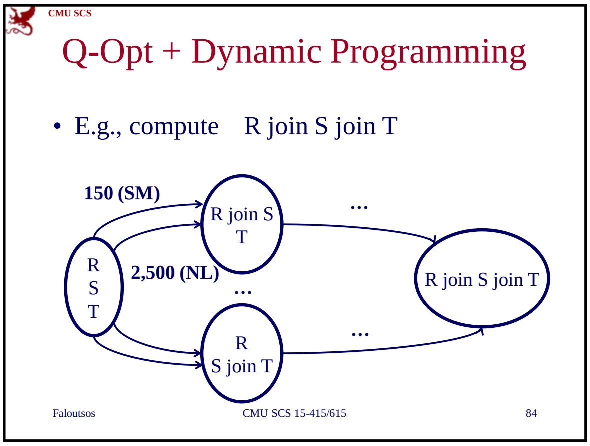
(Reminder: Dynamic Programming)

Q: what are the states, costs and arrows, in q-opt?



(Reminder: Dynamic Programming)

Q: what are the states, costs and arrows, in q-opt?
A: set of intermediate result tables



Q-Opt + Dynamic Programming

- E.g., compute R join S join T

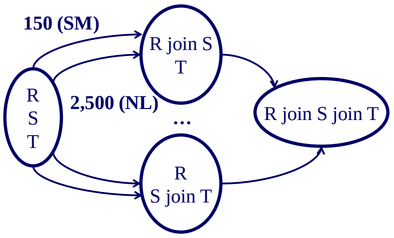
Q-Opt + Dynamic Programming

- Details: how to record the fact that, say **R** is sorted on **R.a**? or that the user requires sorted output?
- Consider the following query:

```
SELECT *
  FROM R, S, T
 WHERE R.a = S.a AND S.b = T.b
 ORDER BY R.a
```

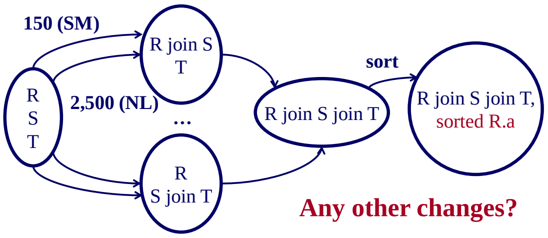
Q-Opt + Dynamic Programming

- E.g., compute R join S join T order by R.a

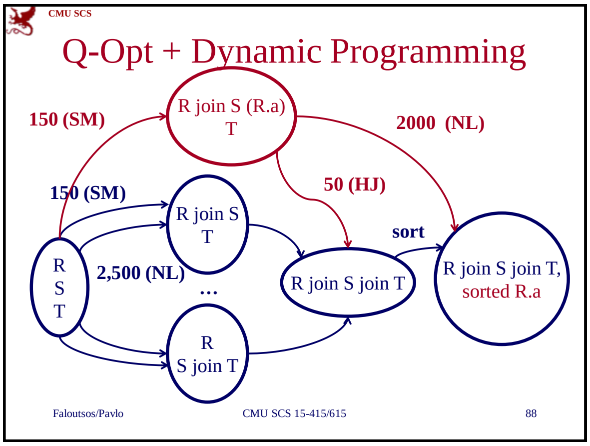


Q-Opt + Dynamic Programming

- E.g., compute R join S join T order by R.a



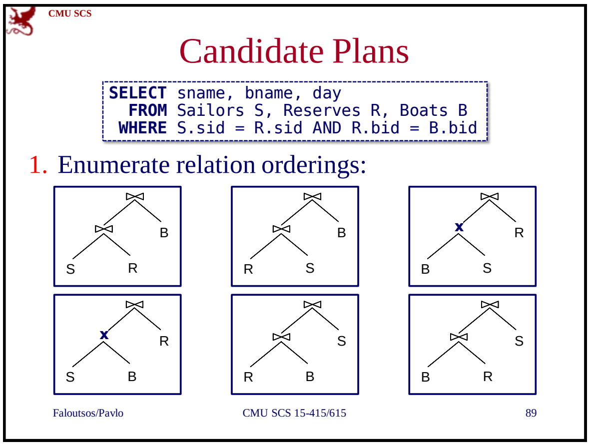
Any other changes?



CMU SCS

Q-Opt + Dynamic Programming

Faloutsos/Pavlo CMU SCS 15-415/615 88



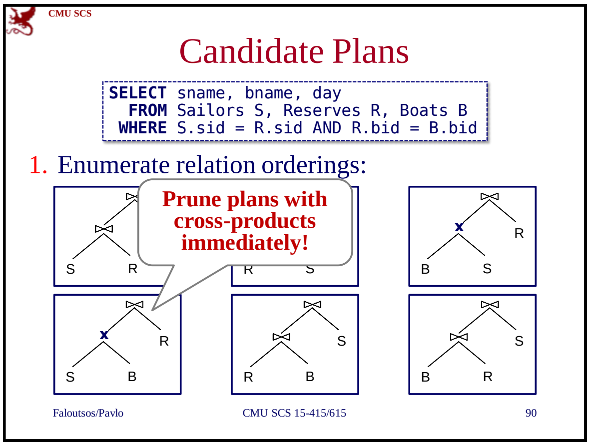
CMU SCS

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

1. Enumerate relation orderings:

Faloutsos/Pavlo CMU SCS 15-415/615 89



CMU SCS

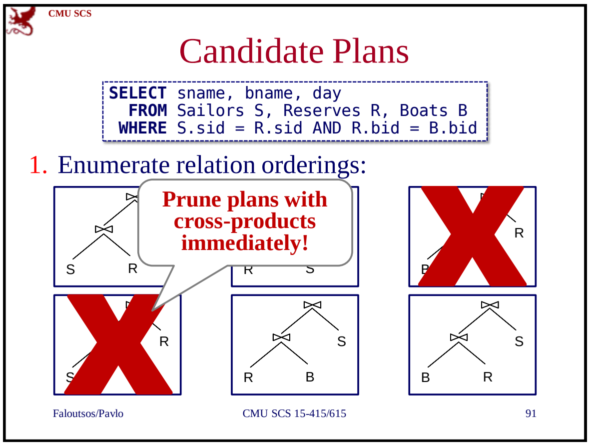
Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

1. Enumerate relation orderings:

Prune plans with cross-products immediately!

Faloutsos/Pavlo CMU SCS 15-415/615 90



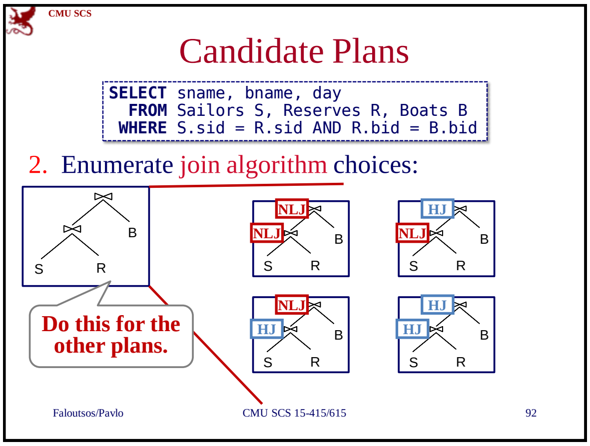
CMU SCS

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

1. Enumerate relation orderings:

Prune plans with cross-products immediately!



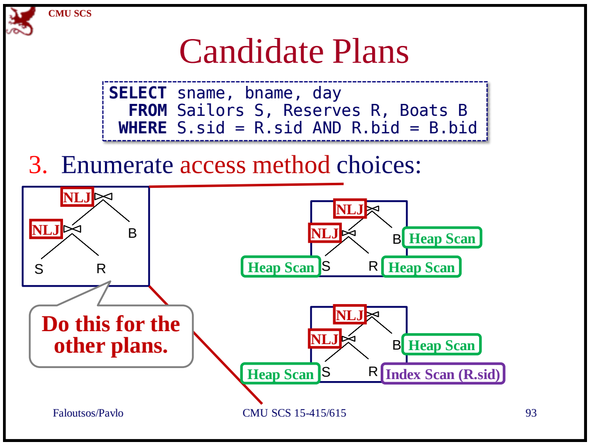
CMU SCS

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

2. Enumerate join algorithm choices:

Do this for the other plans.



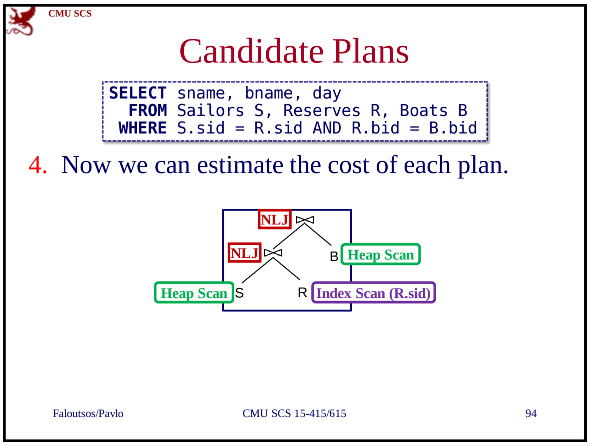
CMU SCS

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

3. Enumerate access method choices:

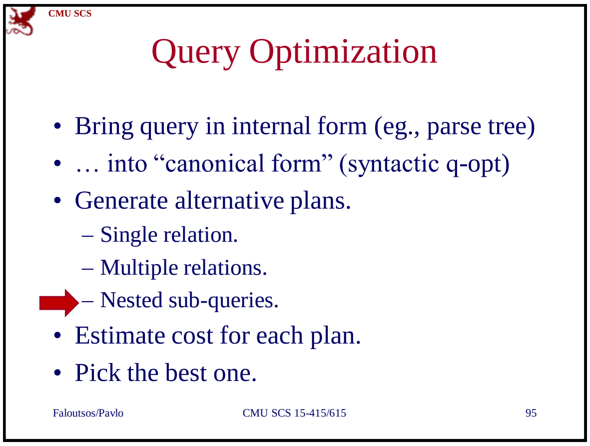
Do this for the other plans.



Candidate Plans

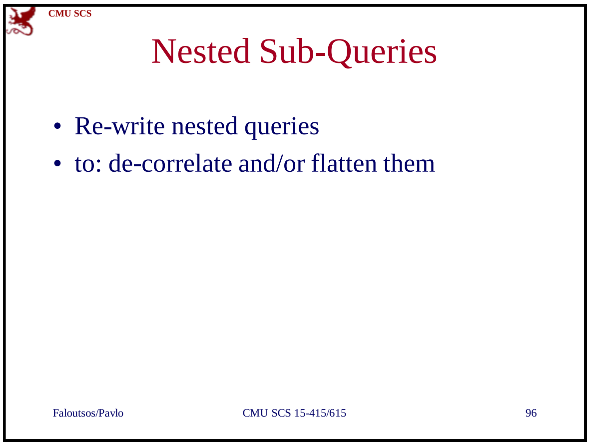
```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

4. Now we can estimate the cost of each plan.



Query Optimization

- Bring query in internal form (eg., parse tree)
- … into “canonical form” (syntactic q-opt)
- Generate alternative plans.
  - Single relation.
  - Multiple relations.
  - Nested sub-queries.
- Estimate cost for each plan.
- Pick the best one.



Nested Sub-Queries

- Re-write nested queries
- to: de-correlate and/or flatten them

Nested Sub-Queries

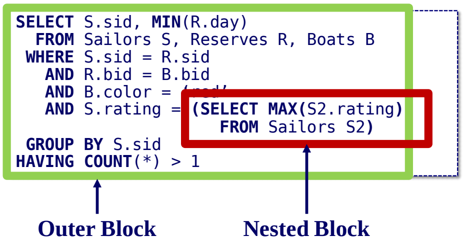
```
SELECT S.sid, MIN(R.day)
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = (SELECT MAX(S2.rating)
                     FROM Sailors S2)
 GROUP BY S.sid
HAVING COUNT(*) > 1
```

For each sailor with the highest rating (over all sailors) and at least two reservations for red boats, find the sailor id and the earliest date on which the sailor has a reservation for a red boat.

Decomposing Queries into Blocks

- The optimizer breaks up queries into blocks and then concentrates on one block at a time.

Decomposing Queries into Blocks



```
SELECT S.sid, MIN(R.day)
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = (SELECT MAX(S2.rating)
                     FROM Sailors S2)
 GROUP BY S.sid
HAVING COUNT(*) > 1
```

Outer Block **Nested Block**

Decomposing Queries into Blocks

- The optimizer breaks up queries into blocks and then concentrates on one block at a time.
- Split *n*-way joins into 2-way joins, then individually optimize.

Query Optimizer Overview

- System R:
  - Break query in query blocks
  - Simple queries (ie., no joins): look at stats
  - n-way joins: left-deep join trees; ie., only one intermediate result at a time
    - *Pros: smaller search space; pipelining*
    - *Cons: may miss optimal*
  - 2-way joins: NL and sort-merge

Conclusions

- Ideas to remember:
  - Syntactic q-opt – do selections early
  - Selectivity estimations (uniformity, indep.; histograms; join selectivity)
  - Hash join (nested loops; sort-merge)
  - Left-deep joins
  - Dynamic programming