# Welcome to CS 186, Section 7!

**TA:** Bryan Munar
**OH:** Mondays 11-12pm and Thursdays 2:30-3:30pm (651 Soda)
**DISC:** Tuesdays 11-12am (136 Barrows) and Wednesdays 10-11am (130 Wheeler)

# Announcements and Such

- Mid-semester survey! Please fill it out!! (plz plz plz)

- Homework 4 out soon! (or it's already out LOL)

- Midterm 2 on November 9th

# Discussion 7: Query Optimization

# Overview:

1. Query Optimization
2. Worksheet walkthrough

(A majority of the slides are from Michelle and lecture!)

# Query Optimization

IMPORTANT

# Context and Problem

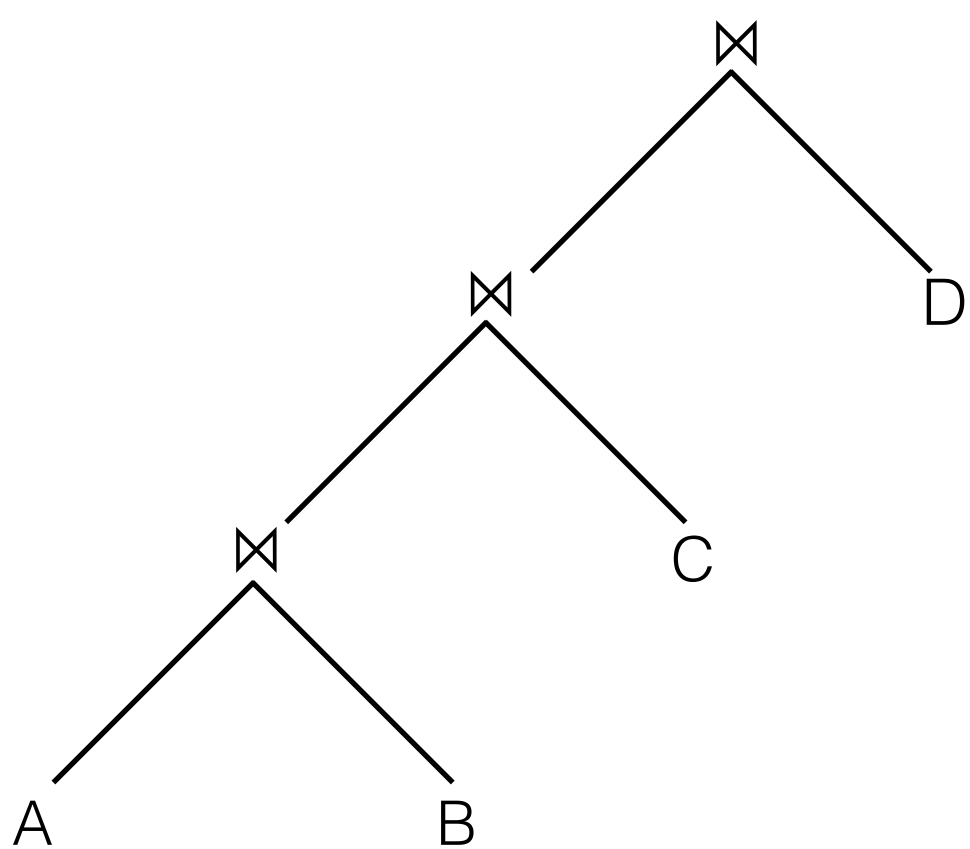
- We’re given a SQL query where we have to do more than 2 or 3 table joins (e.g. multiple equijoins)
- But in the end we really only want a small subset of that result, like A.name = “Bryan”
- This is a specific case, but I’m sure there are more types of queries out there that can be really optimized
- Is there anyway we can optimize this?

# Solution: Query Optimization

- What is the best way to run a query?
- Change order and methods of operators for:
  - Faster queries, better resource utilization
  - Smaller # of total I/Os

# Plan Space

- Based on relational equivalences
- Only consider left-deep join trees
  - Includes all join orders and join methods



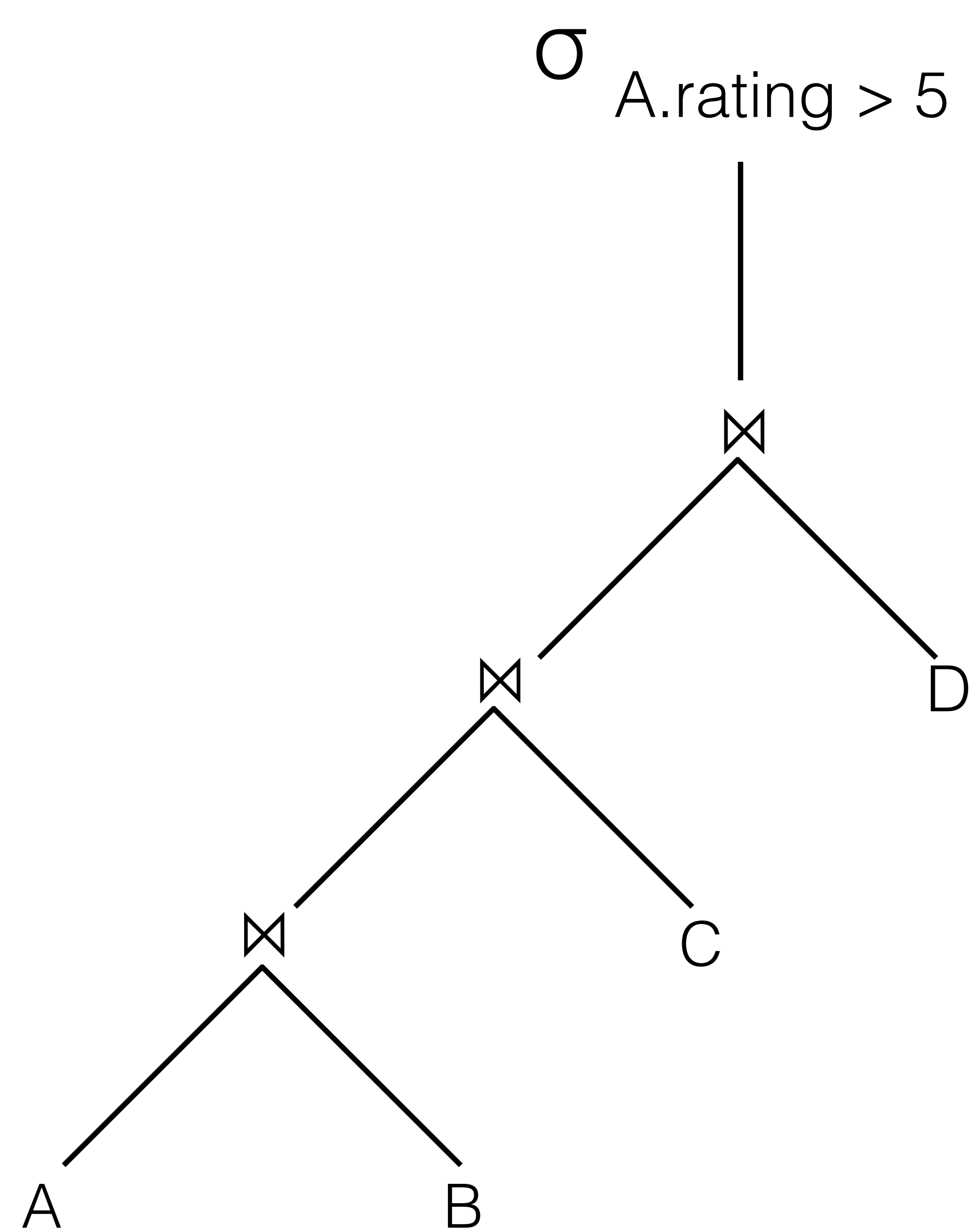
# Determinants of Plan Cost

- Access method of base tables
  - Scan, index, range vs. lookup, clustered vs. unclustered
- Join ordering
  - Do we want to keep rereading a big table over and over again?
- Join method
  - Sort-merge? Hash? BNL?

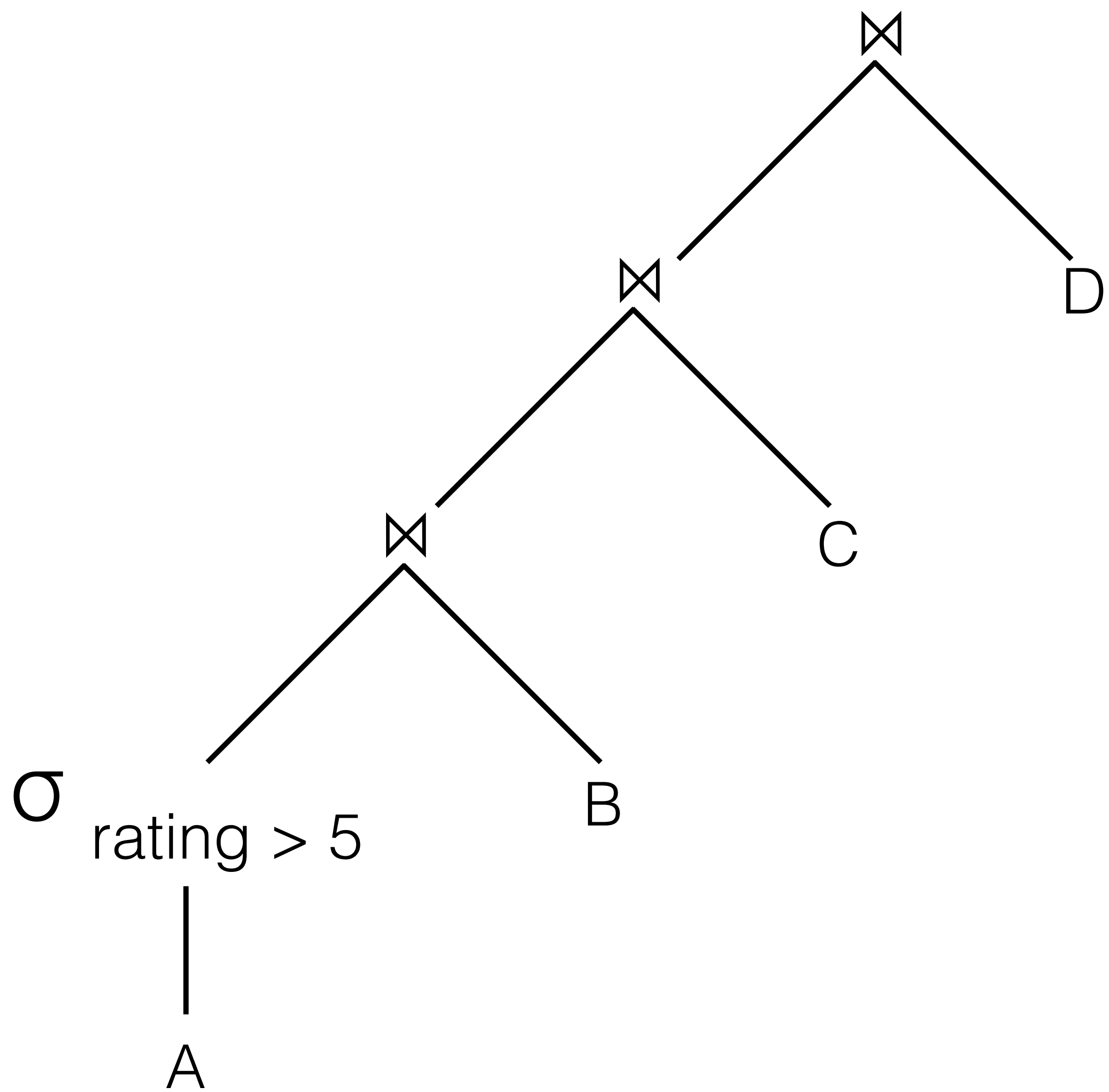
# Cost Estimation

- Estimate cost of each operation in plan tree
- Must estimate size of result for each operation using info from system catalog
- For System R, cost: #I/Os + CPU-factor * #tuples
- **On-the-fly:** The result of one operator is pipelined to another operator without creating a temporary table to hold intermediate result (e.g. push select on the fly)

# Push Select



# Push Select



# Selectivity

- Selectivity represents a predicate's impact on reducing result size
  - |output| / |input|
  - Tuples that contain rating 0 to 100:
    - $\sigma_{\text{rating} > 0}$ has large selectivity
    - $\sigma_{\text{rating} > 99}$ has smaller selectivity
- If missing info to estimate selectivity, assume 1/10!
- Selectivity also known as reduction factor (RF)

# Selectivity

- Predicate col=value
  - Selectivity = 1/NKeys(col)
- Predicate col1=col2
  - Selectivity = 1/MAX(NKeys(col1), NKeys(col2))
- Predicate col>value
  - Selectivity= (High(col)-value)/(High(col)-Low(col) + 1)
- Assumes that values and uniformly distributed and independent!
- Result Cardinality: Max # tuples * product of all selectivities

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
zipcode)
Owner(ssn, license, name, gender, street, city,
zipcode)
```

- For the query: "SELECT * FROM Accident A, Car C WHERE A.license = C.license AND A.damage_amount > X;" For what types of values of X would selection push-down significantly improve the cost of the query (Car is the inner table of the join)?"

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
zipcode)
Owner(ssn, license, name, gender, street, city,
zipcode)
```

- For the query: "SELECT * FROM Accident A, Car C WHERE A.license = C.license AND A.damage_amount > X;" For what types of values of X would selection push-down significantly improve the cost of the query (Car is the inner table of the join)?"

- Selection push-down will help with very large values of X, since that would be more selective, and thus result in fewer resulting tuples for the rest of the plan.

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
zipcode)
Owner(ssn, license, name, gender, street, city,
zipcode)
```

- For the query: "SELECT O.name FROM Car C, Owner O WHERE C.license = O.license AND C.company = 'Volvo';" What is the expected cardinality of the Car relation after the initial selections are applied (before the join)?

NTuples(Car) = 1000 ; NPages(Car) = 100
NTuples(Accident) = 500 ; NPages(Accident) = 20
NTuples(Owner) = 800 ; NPages(Owner) = 50
NDistinct(Car.company) = 50;

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
zipcode)
Owner(ssn, license, name, gender, street, city,
zipcode)
```

- For the query: "SELECT O.name FROM Car C, Owner O WHERE C.license = O.license AND C.company = 'Volvo';" What is the expected cardinality of the Car relation after the initial selections are applied (before the join)?

- You can only push down the Car.company = 'Volvo' selection predicate. NDistinct(Car.company) = 50, so we can estimate Selectivity(Car.company) = 1/50. Cardinality(Car.company = 'Volvo') = Selectivity(Car.company) * NTuples(Car) = 1000 / 50 = 20

# Another Example

## Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

As seen in previous lectures…

- **Reserves:**
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats
- **Sailors:**
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings

- Assume we have 5 pages in our buffer pool!

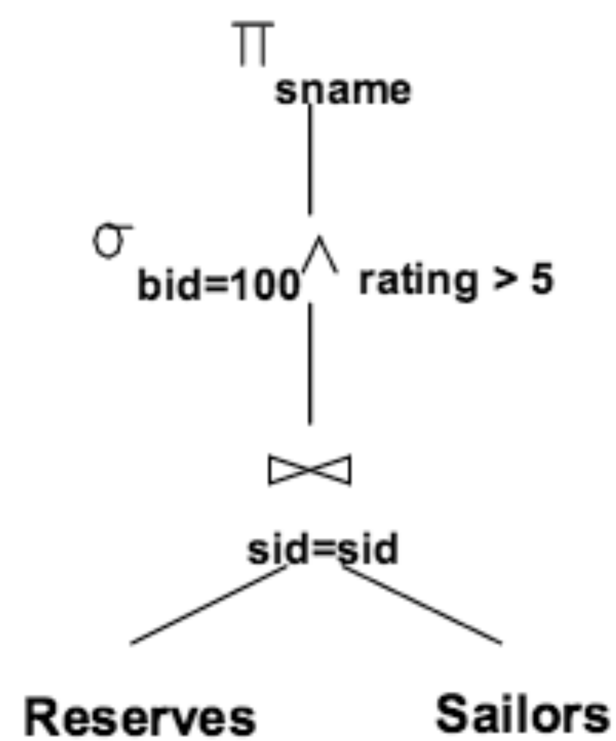
# Another Example

## Query Optimization Overview

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

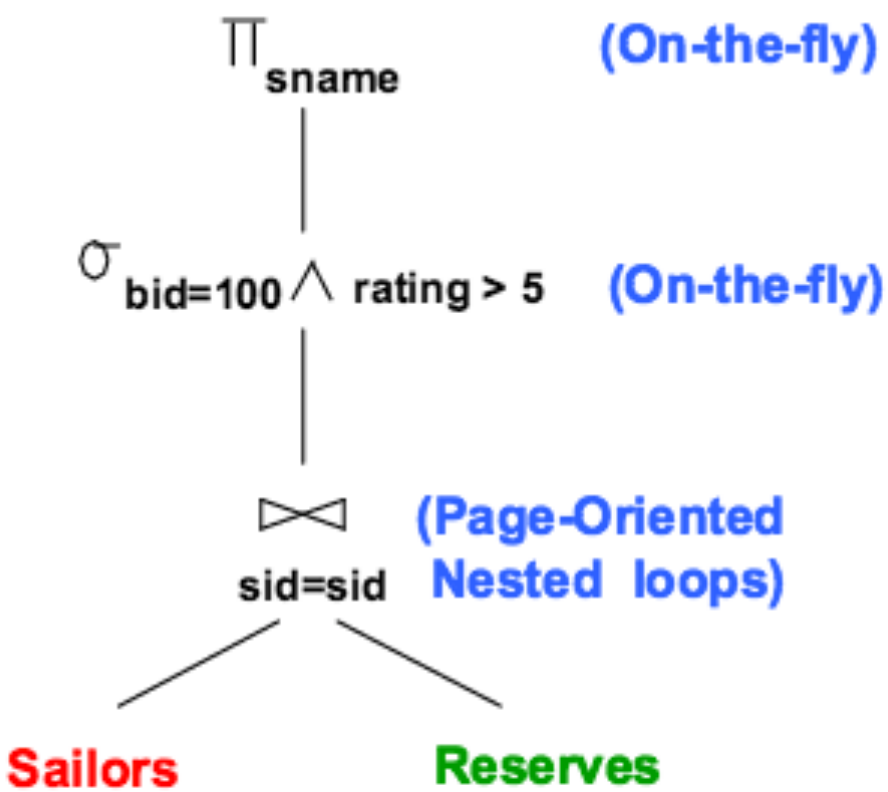
```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

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

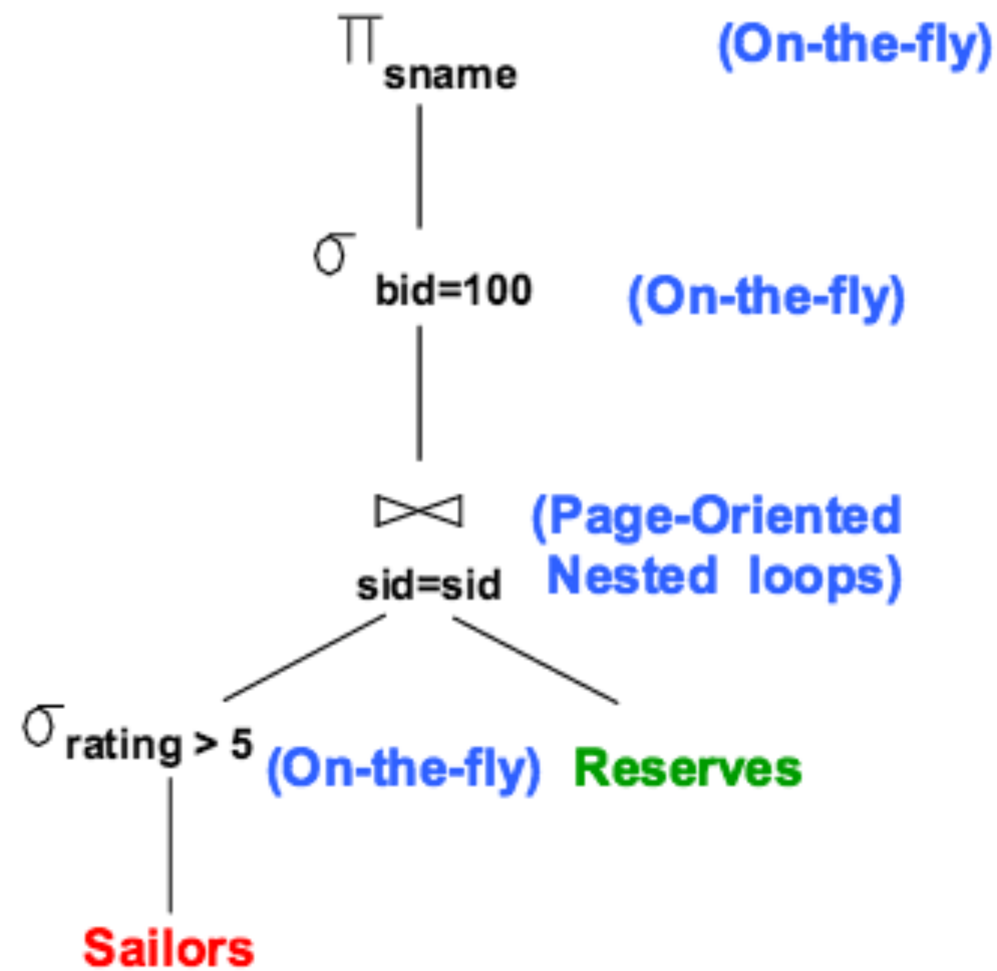


# Another Example

## Alternative Plans – Push Selects (No Indexes)

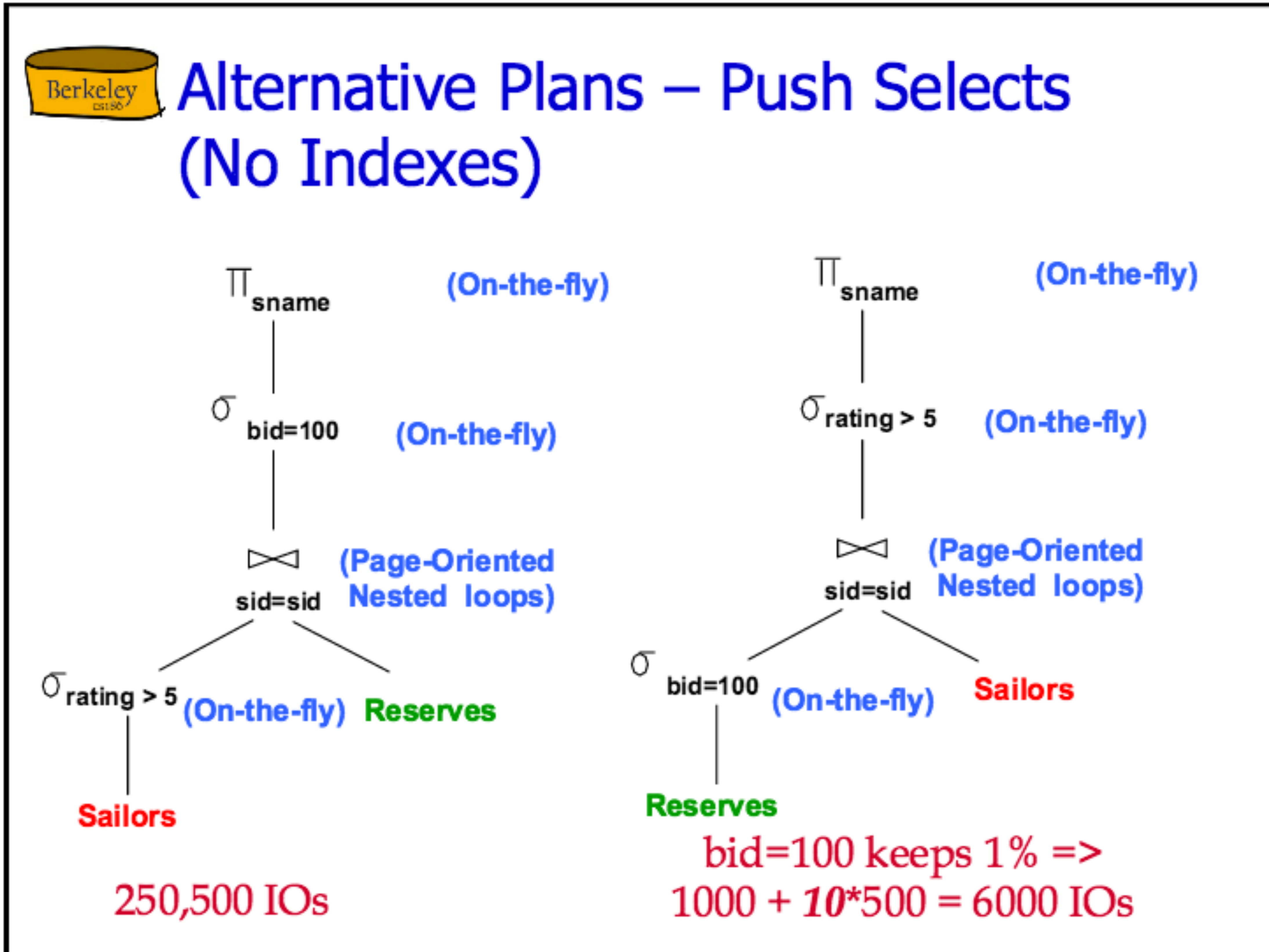



500,500 IOs




rating > 5 throws out half =>
250,500 IOs

# Another Example



# Aside: Histograms

- Represent distribution of one or more attributes
- Better estimate of selectivity than assuming uniformity
  - Can be Gaussian (Normal), exponential, etc.
- Think of as lossy compression, more buckets, more accurate

# System R (Selinger) Optimizer

1. Plan Space
   - only consider left-deep join trees
   - avoid cartesian products
   - push selects and joins
   - consider interesting orders
2. Cost estimation
   - use cost formulas and size estimations
   - Selectivity (reduction factor) estimation = |output| / |input|
3. Search Algorithm
   - dynamic programming, consider using AI

# Interesting Orders

- Operator returns an “interesting order” if its result is in order of:
  - some *ORDER BY* attribute
    - means we don’t have to sort later!
  - some *GROUP BY* attribute
    - means we can use a nice scan method for our group-by later!
  - some Join attribute of other joins
    - Means we can use sort-merge far cheaper!

# Search Algorithm

- Find the best 1-table access method.
- Given the best 1-table method as the outer, find the best 2-table.
- …
- Given the best (N-1)-table method as the outer, find the best N-table.
- ** Instead of “strictly the best” we return the best for each interesting order of the tuples.
- ** Do cross products last!

# Example

```
Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND B.color = "red"
GROUP BY S.sid
```

Sailors:

- Hash, B+ on sid

Reserves:

- Clustered B+ tree on bid
- B+ on sid

Boats

- B+ on color

# Example

Find the best 1-table access method for each relation (only consider the predicates dealing with one table).

```
Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND B.color = "red"
GROUP BY S.sid
```

Sailors:

- Hash, B+ on sid

Reserves:

- Clustered B+ tree on bid
- B+ on sid

Boats

- B+ on color

# Example

Find the best 1-table access method for each relation.

- Sailors, Reserves: File Scan
  - (B+ tree on Reserves.bid as interesting order)
  - (B+ tree on Sailors.sid as interesting order)
- Boats: B+ tree on color

```
Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND B.color = “red”
GROUP BY S.sid
```

Sailors:

Hash, B+ on sid

Reserves:

Clustered B+ tree on bid

B+ on sid

Boats

B+ on color

# Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

# Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- File Scan Reserves (outer) with Boats (inner)
- File Scan Reserves (outer) with Sailors (inner)

# Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- Reserves Btree on bid (outer) with Boats (inner)
- Reserves Btree on bid (outer) with Sailors (inner)

# Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- File Scan Sailors (outer) with Boats (inner)
- File Scan Sailors (outer) with Reserves (inner)

# Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- B+ tree Sailors (outer) with Boats (inner)
- B+ tree Sailors (outer) with Reserves (inner)

# Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- Boats Btree on color with Sailors (inner)
- Boats Btree on color with Reserves (inner)

# Example

Given the best 1-table method as the outer, find the best 2-table.

- File Scan Reserves (outer) with Boats (inner)
- File Scan Reserves (outer) with Sailors (inner)
- Reserves Btree on bid (outer) with Boats (inner)
- Reserves Btree on bid (outer) with Sailors (inner)
- File Scan Sailors (outer) with Boats (inner)
- File Scan Sailors (outer) with Reserves (inner)
- B+ tree Sailors (outer) with Boats (inner)
- B+ tree Sailors (outer) with Reserves (inner)
- Boats Btree on color with Sailors (inner)
- Boats Btree on color with Reserves (inner)

- Retain cheapest plan for each (pair of relations, order)

Remember that we're **estimating** the optimality our proposed optimization! This doesn't mean that the values we compute are exact!

# Worksheet

# What are the best single-table plans?

You should be calculating the number of I/Os it takes per access method you're given, corresponding to single table predicates in the query (like we did in the previous slides)

# What are the best single-table plans?

- Kitties: B+ tree on cuteness
  - **File scan = 100**
    - **see note in discussion answers for food4thought!**

- Puppies: B+ tree on yappiness
  - File scan = 50
  - **B+ tree = (5+200)*1/10 = about 20**

- Humans: File Scan
  - **File scan = 1000**
  - B+ tree = (20 + 50,000)*1,200/50,000)

# List the pairs of tables the optimizer will consider for 2-way joins

# List the pairs of tables the optimizer will consider for 2-way joins

- Kitties[file scan] ⋈ Puppies
- Kitties[file scan] ⋈ Humans
- Puppies[unclustered B+] ⋈ Kitties
- Puppies[unclustered B+] ⋈ Humans
- Humans[file scan] ⋈ Kitties
- Humans[file scan] ⋈ Puppies

# Which plans will be avoided?

- Kitties[file scan] $\bowtie$ Puppies
- Kitties[file scan] $\bowtie$ Humans
- Puppies[unclustered B+] $\bowtie$ Kitties
- Puppies[unclustered B+] $\bowtie$ Humans
- Humans[file scan] $\bowtie$ Kitties
- Humans[file scan] $\bowtie$ Puppies

# Which plans will be avoided?

- Kitties[file scan] ⋈ Puppies
- ~~Kitties[file scan] x Humans~~
- Puppies[unclustered B+] ⋈ Kitties
- Puppies[unclustered B+] ⋈ Humans
- ~~Humans[file scan] x Kitties~~
- Humans[file scan] ⋈ Puppies

Humans and kitties don't have a join predicate!

What would be the IO cost of doing index nested loops join using Puppies as the outer, with the optimal single table selection methods?

# What would be the IO cost of doing index nested loops join using Puppies as the outer, with the optimal single table selection methods?

- Index scan: (5+200)*1/10 = ~21 I/Os to select puppies
- # selected puppies = 200 * 1/10 = 20
- 20*(5+400)*1/10 = 810 I/Os
- 21 + 810 = 831 I/Os

#tuples in puppies * index lookup on K.cuteness = P.yappiness

What would be the IO cost of doing index nested loops join using Kitties as the outer, with the optimal single table selection methods?

# What would be the IO cost of doing index nested loops join using Kitties as the outer, with the optimal single table selection methods?

- (5+400)*1/10 = 41 I/Os to select kitties
- Clustered lookup for (owner = val and yappiness = 7)
  - (15+50)*(1/10)*(1/10) = 1 I/O — just an estimate!
- 41 + 40*1 = 81 I/Os