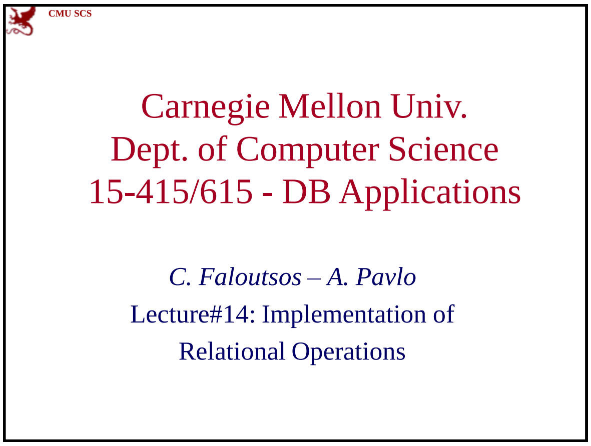
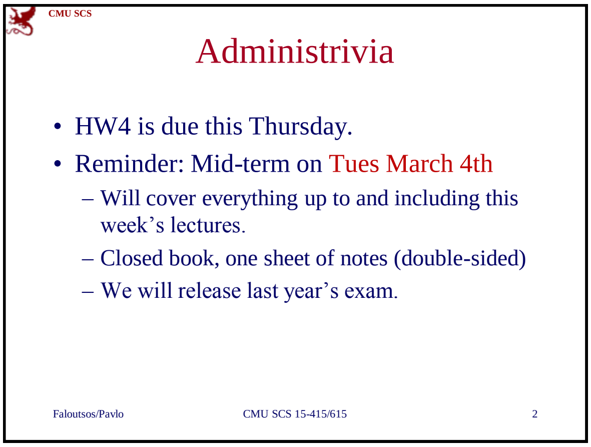


Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

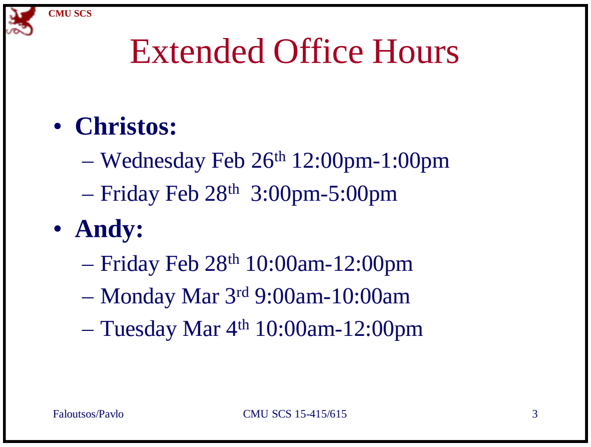
C. Faloutsos – A. Pavlo

Lecture#14: Implementation of Relational Operations



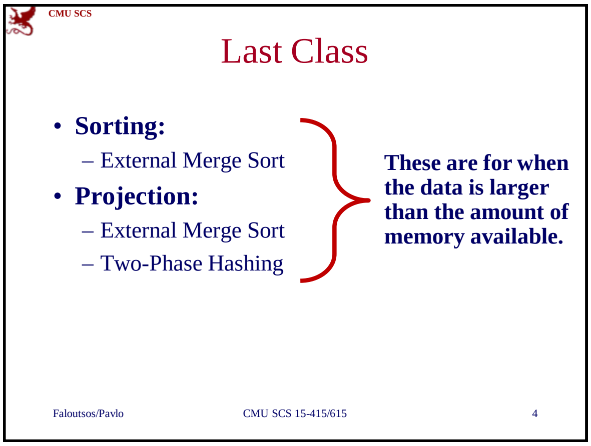
Administrivia

- HW4 is due this Thursday.
- Reminder: Mid-term on Tues March 4th
  - Will cover everything up to and including this week’s lectures.
  - Closed book, one sheet of notes (double-sided)
  - We will release last year’s exam.



Extended Office Hours

- **Christos:**
  - Wednesday Feb 26th 12:00pm-1:00pm
  - Friday Feb 28th 3:00pm-5:00pm
- **Andy:**
  - Friday Feb 28th 10:00am-12:00pm
  - Monday Mar 3rd 9:00am-10:00am
  - Tuesday Mar 4th 10:00am-12:00pm



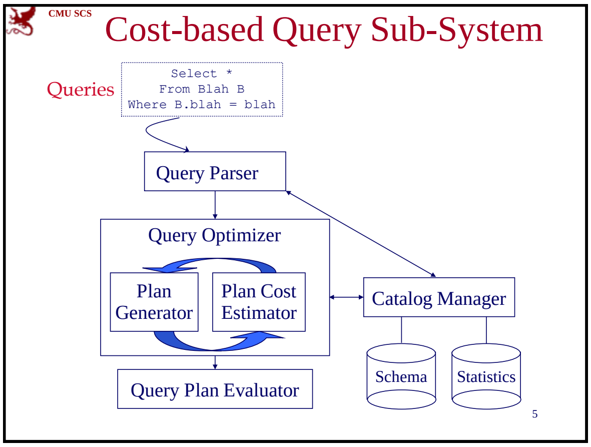
CMU SCS

Last Class

- **Sorting:**
  - External Merge Sort
- **Projection:**
  - External Merge Sort
  - Two-Phase Hashing

These are for when the data is larger than the amount of memory available.

Faloutsos/Pavlo CMU SCS 15-415/615 4



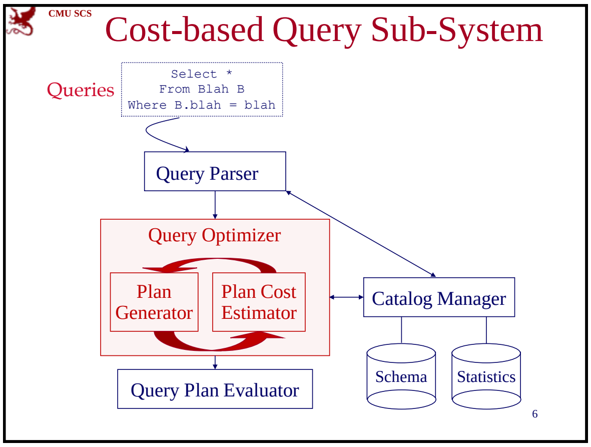
CMU SCS

Cost-based Query Sub-System

Queries

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

5



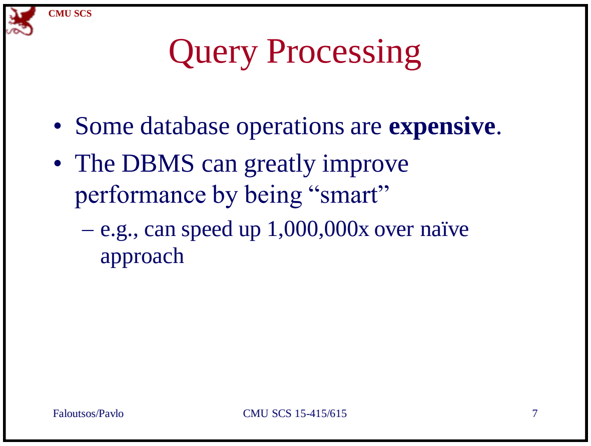
CMU SCS

Cost-based Query Sub-System

Queries

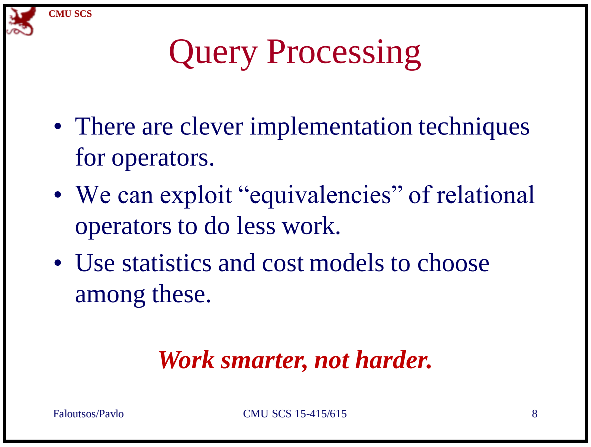
```
Select *
From Blah B
Where B.blah = blah
```

6



Query Processing

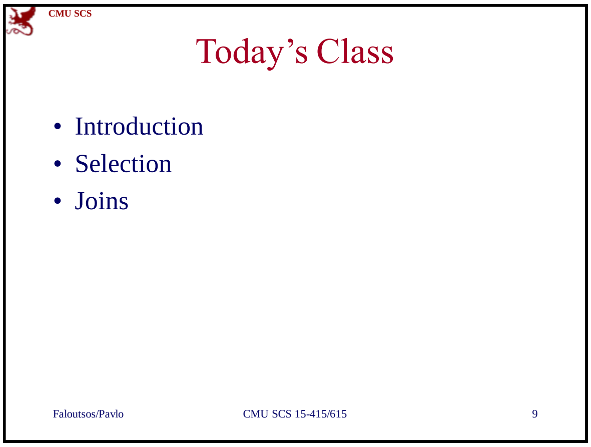
- Some database operations are **expensive**.
- The DBMS can greatly improve performance by being “smart”
  - e.g., can speed up 1,000,000x over naïve approach



Query Processing

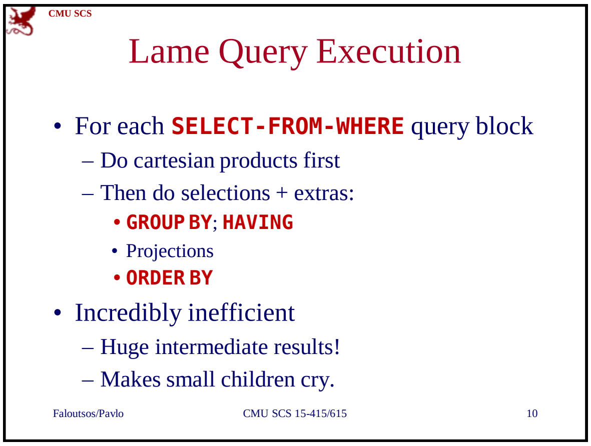
- There are clever implementation techniques for operators.
- We can exploit “equivalencies” of relational operators to do less work.
- Use statistics and cost models to choose among these.

Work smarter, not harder.



Today’s Class

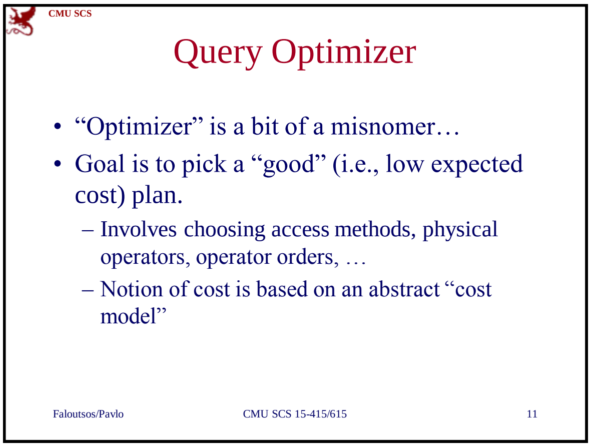
- Introduction
- Selection
- Joins



CMU SCS

Lame Query Execution

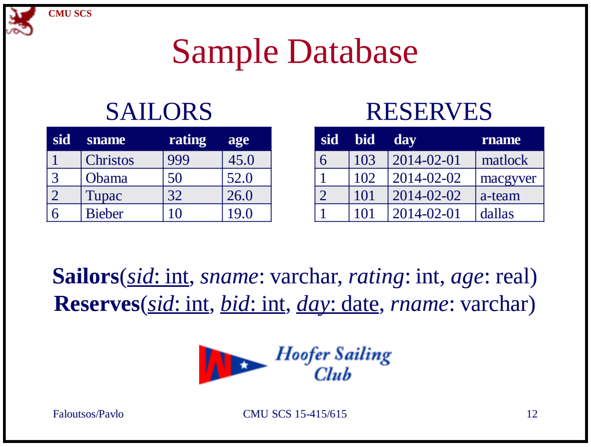
- For each **SELECT-FROM-WHERE** query block
  - Do cartesian products first
  - Then do selections + extras:
    - **GROUP BY**; **HAVING**
    - Projections
    - **ORDER BY**
- Incredibly inefficient
  - Huge intermediate results!
  - Makes small children cry.



CMU SCS

Query Optimizer

- "Optimizer" is a bit of a misnomer…
- Goal is to pick a "good" (i.e., low expected cost) plan.
  - Involves choosing access methods, physical operators, operator orders, …
  - Notion of cost is based on an abstract "cost model"



CMU SCS

Sample Database

SAILORS

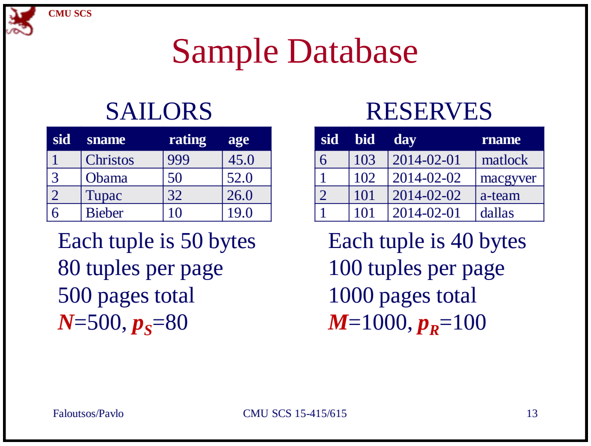
sid	sname	rating	age
1	Christos	999	45.0
3	Obama	50	52.0
2	Tupac	32	26.0
6	Bieber	10	19.0

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

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

Hoofer Sailing Club



CMU SCS

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

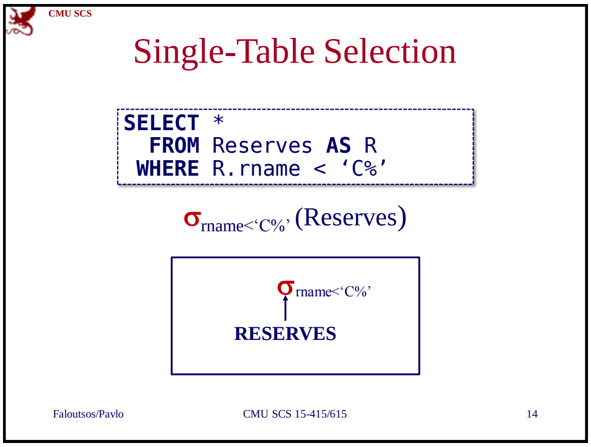
Each tuple is 50 bytes
80 tuples per page
500 pages total
N=500, p_S=80

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

Each tuple is 40 bytes
100 tuples per page
1000 pages total
M=1000, p_R=100

Faloutsos/Pavlo CMU SCS 15-415/615 13



CMU SCS

Single-Table Selection

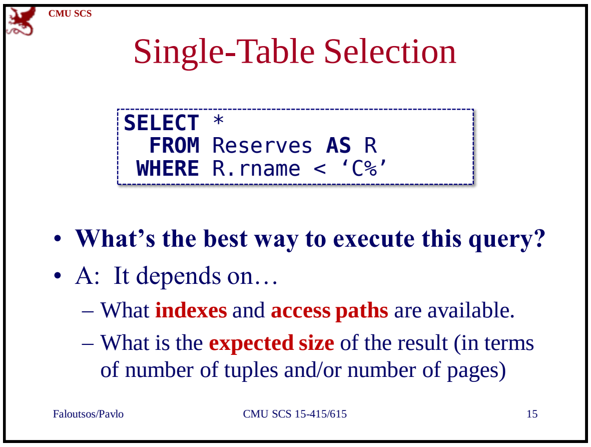
```
SELECT *
  FROM Reserves AS R
 WHERE R.rname < 'C%'
```

$\sigma_{rname<'C\%'}$ (Reserves)

$\sigma_{rname<'C\%'}$

RESERVES

Faloutsos/Pavlo CMU SCS 15-415/615 14



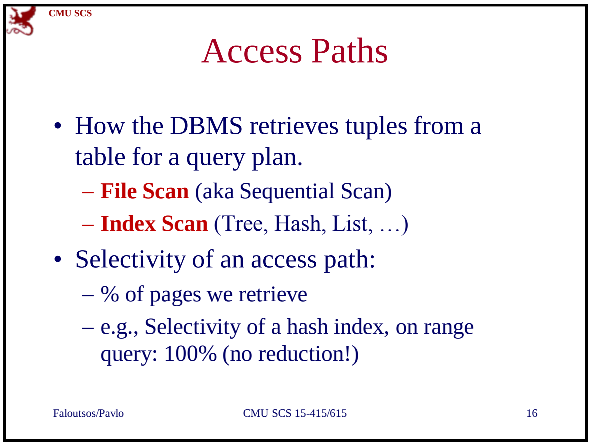
CMU SCS

Single-Table Selection

```
SELECT *
  FROM Reserves AS R
 WHERE R.rname < 'C%'
```

- **What's the best way to execute this query?**
- A: It depends on…
  – What **indexes** and **access paths** are available.
  – What is the **expected size** of the result (in terms of number of tuples and/or number of pages)

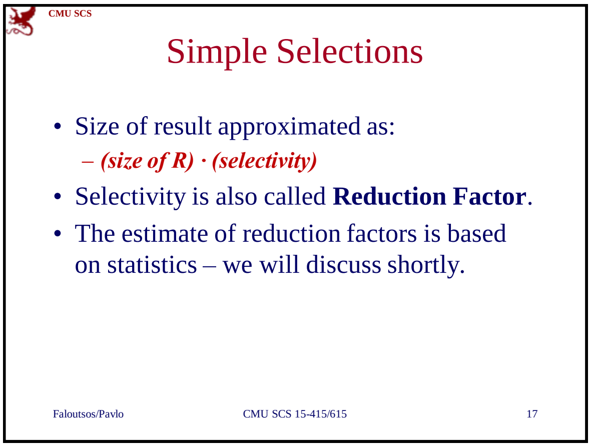
Faloutsos/Pavlo CMU SCS 15-415/615 15



CMU SCS

Access Paths

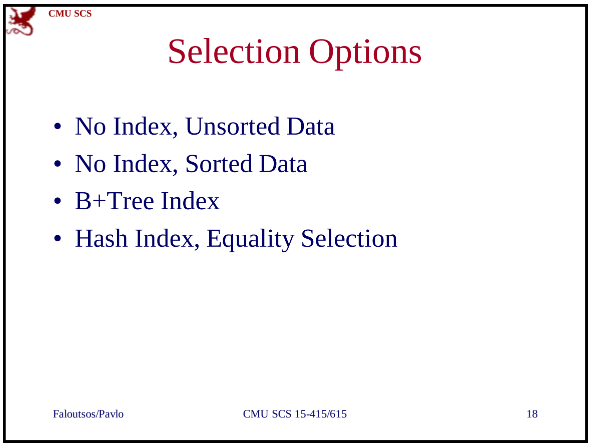
- How the DBMS retrieves tuples from a table for a query plan.
  - **File Scan** (aka Sequential Scan)
  - **Index Scan** (Tree, Hash, List, …)
- Selectivity of an access path:
  - % of pages we retrieve
  - e.g., Selectivity of a hash index, on range query: 100% (no reduction!)



CMU SCS

Simple Selections

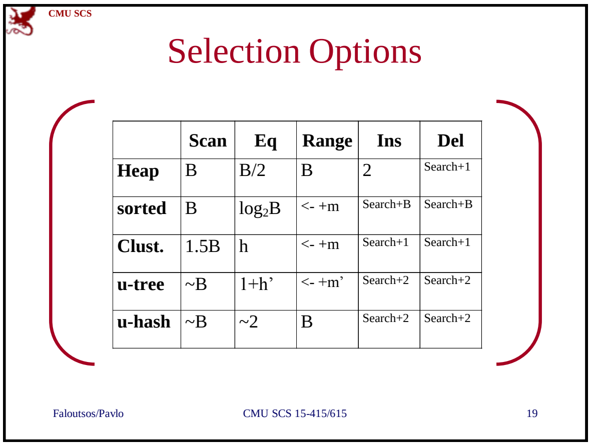
- Size of result approximated as:
  - *(size of R) · (selectivity)*
- Selectivity is also called **Reduction Factor**.
- The estimate of reduction factors is based on statistics – we will discuss shortly.



CMU SCS

Selection Options

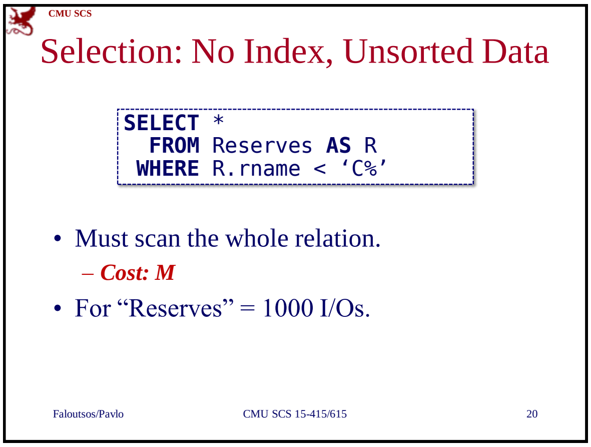
- No Index, Unsorted Data
- No Index, Sorted Data
- B+Tree Index
- Hash Index, Equality Selection



CMU SCS

Selection Options

	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

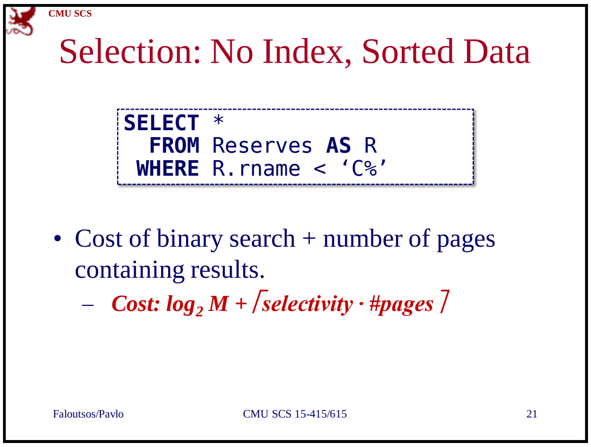


CMU SCS

Selection: No Index, Unsorted Data

```
SELECT *
  FROM Reserves AS R
 WHERE R.rname < ‘C%’
```

- Must scan the whole relation.
  - *Cost: M*
- For “Reserves” = 1000 I/Os.

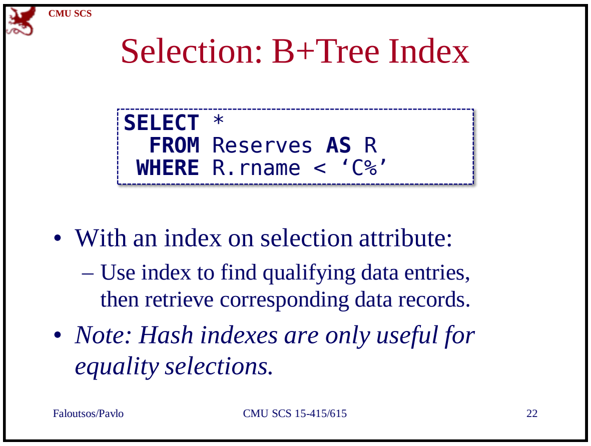


CMU SCS

Selection: No Index, Sorted Data

```
SELECT *
  FROM Reserves AS R
 WHERE R.rname < ‘C%’
```

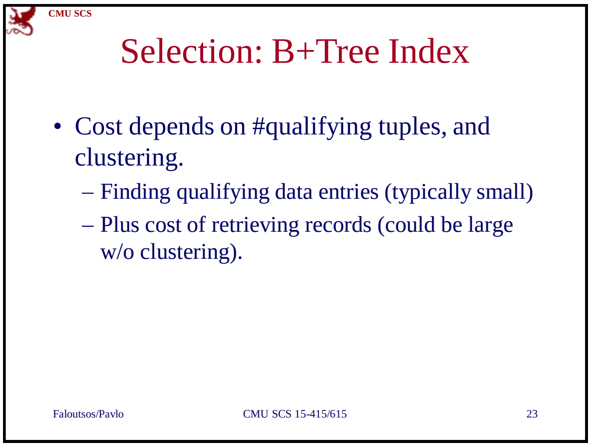
- Cost of binary search + number of pages containing results.
  - *Cost:* $\log_2 M + \lceil selectivity \cdot \#pages \rceil$



Selection: B+Tree Index

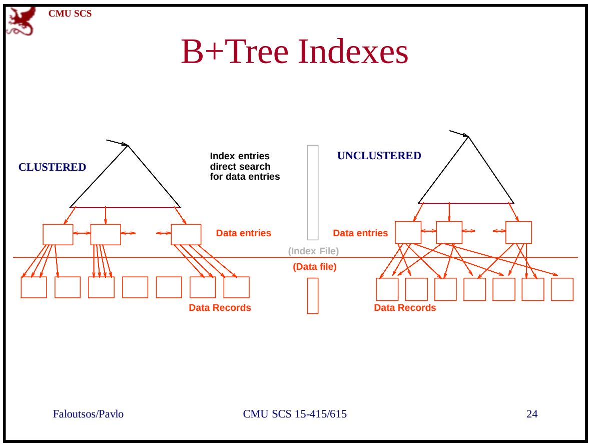
```
SELECT *
  FROM Reserves AS R
 WHERE R.rname < 'C%'
```

- With an index on selection attribute:
  - Use index to find qualifying data entries, then retrieve corresponding data records.
- *Note: Hash indexes are only useful for equality selections.*

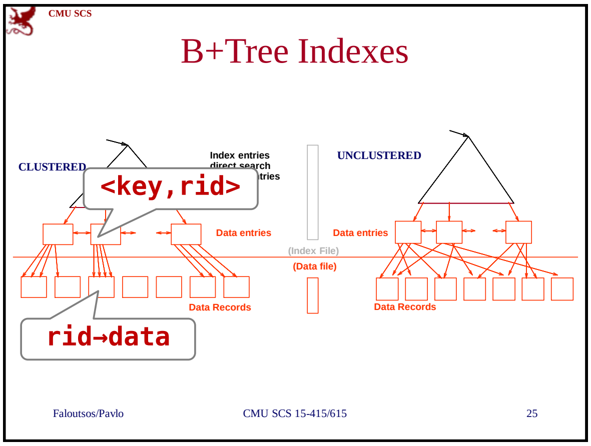


Selection: B+Tree Index

- Cost depends on #qualifying tuples, and clustering.
  - Finding qualifying data entries (typically small)
  - Plus cost of retrieving records (could be large w/o clustering).



B+Tree Indexes



CMU SCS

B+Tree Indexes

CLUSTERED

Index entries
direct search
tries

<key,rid>

Data entries

Data Records

rid→data

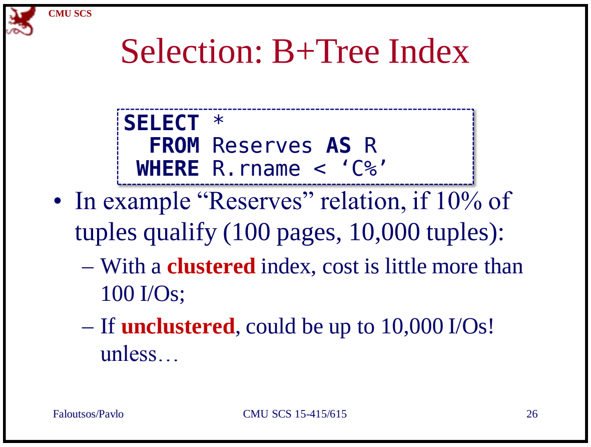
(Index File)
(Data file)

UNCLUSTERED

Data entries

Data Records

Faloutsos/Pavlo CMU SCS 15-415/615 25



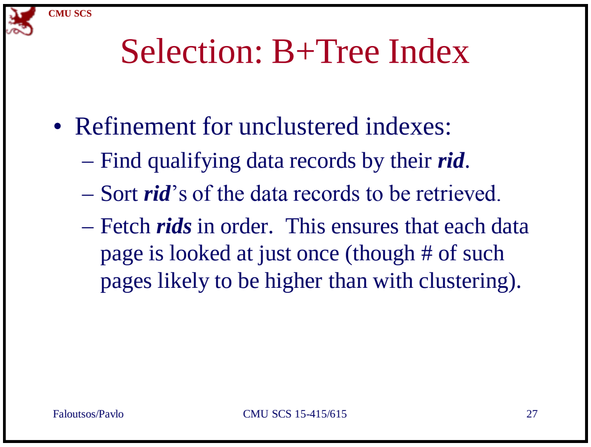
CMU SCS

Selection: B+Tree Index

```
SELECT *
  FROM Reserves AS R
 WHERE R.rname < ‘C%’
```

- In example “Reserves” relation, if 10% of tuples qualify (100 pages, 10,000 tuples):
  - With a **clustered** index, cost is little more than 100 I/Os;
  - If **unclustered**, could be up to 10,000 I/Os! unless…

Faloutsos/Pavlo CMU SCS 15-415/615 26

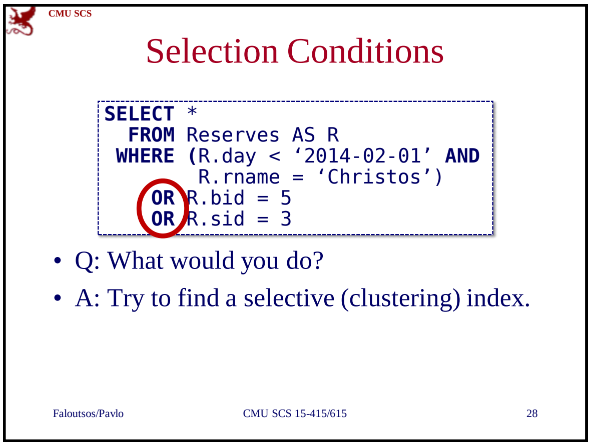


CMU SCS

Selection: B+Tree Index

- Refinement for unclustered indexes:
  - Find qualifying data records by their ***rid***.
  - Sort ***rid***’s of the data records to be retrieved.
  - Fetch ***rids*** in order. This ensures that each data page is looked at just once (though # of such pages likely to be higher than with clustering).

Faloutsos/Pavlo CMU SCS 15-415/615 27

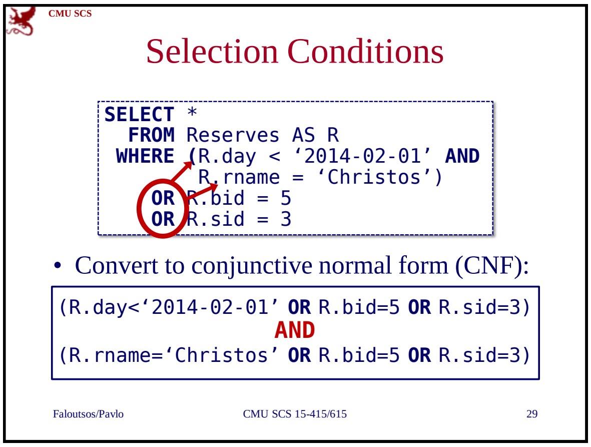


CMU SCS

Selection Conditions

```
SELECT *
  FROM Reserves AS R
 WHERE (R.day < ‘2014-02-01’ AND
        R.rname = ‘Christos’)
    OR R.bid = 5
    OR R.sid = 3
```

- Q: What would you do?
- A: Try to find a selective (clustering) index.



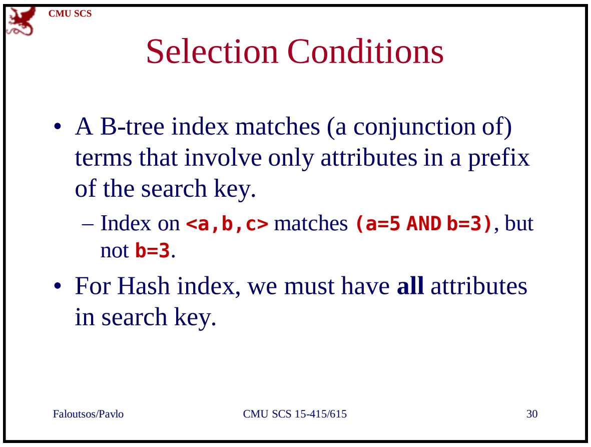
CMU SCS

Selection Conditions

```
SELECT *
  FROM Reserves AS R
 WHERE (R.day < ‘2014-02-01’ AND
        R.rname = ‘Christos’)
    OR R.bid = 5
    OR R.sid = 3
```

- Convert to conjunctive normal form (CNF):

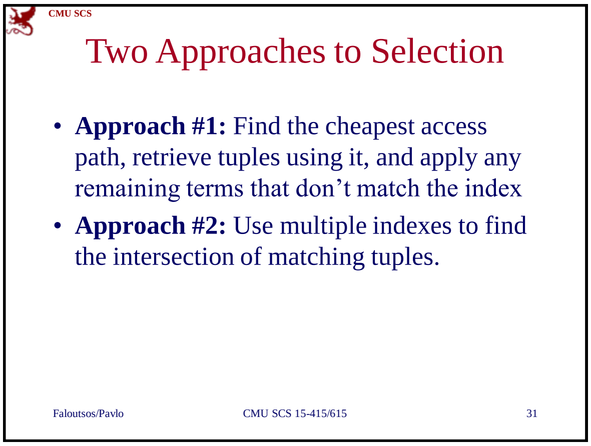
```
(R.day<‘2014-02-01’ OR R.bid=5 OR R.sid=3)
                    AND
(R.rname=‘Christos’ OR R.bid=5 OR R.sid=3)
```



CMU SCS

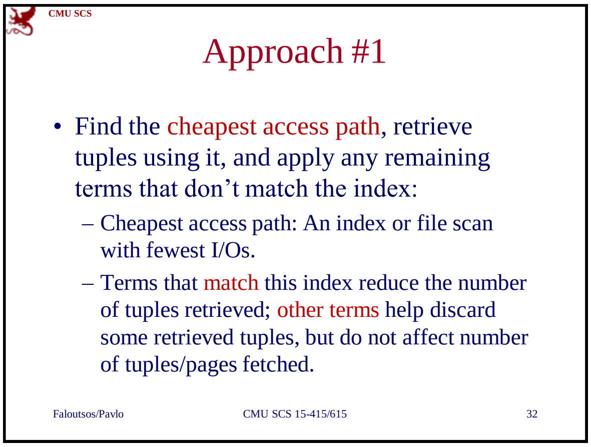
Selection Conditions

- A B-tree index matches (a conjunction of) terms that involve only attributes in a prefix of the search key.
  - Index on `<a,b,c>` matches `(a=5 AND b=3)`, but not `b=3`.
- For Hash index, we must have **all** attributes in search key.



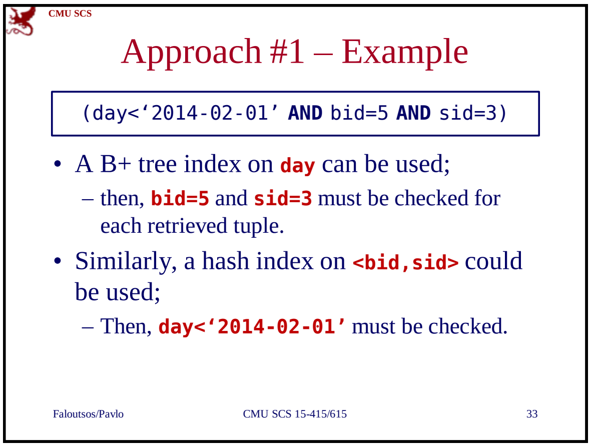
Two Approaches to Selection

- **Approach #1:** Find the cheapest access path, retrieve tuples using it, and apply any remaining terms that don't match the index
- **Approach #2:** Use multiple indexes to find the intersection of matching tuples.



Approach #1

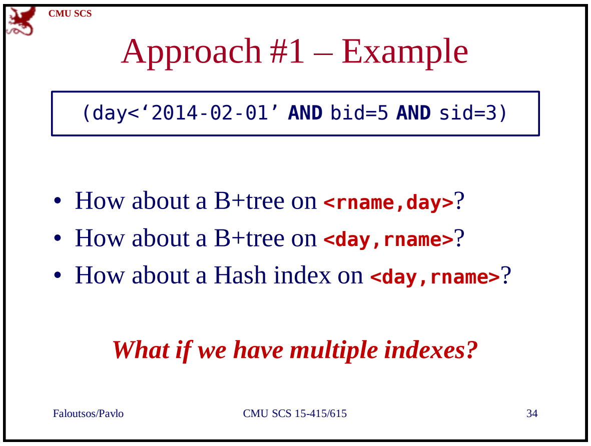
- Find the cheapest access path, retrieve tuples using it, and apply any remaining terms that don't match the index:
  - Cheapest access path: An index or file scan with fewest I/Os.
  - Terms that match this index reduce the number of tuples retrieved; other terms help discard some retrieved tuples, but do not affect number of tuples/pages fetched.



Approach #1 – Example

```
(day<'2014-02-01' AND bid=5 AND sid=3)
```

- A B+ tree index on `day` can be used;
  - then, `bid=5` and `sid=3` must be checked for each retrieved tuple.
- Similarly, a hash index on `<bid,sid>` could be used;
  - Then, `day<'2014-02-01'` must be checked.



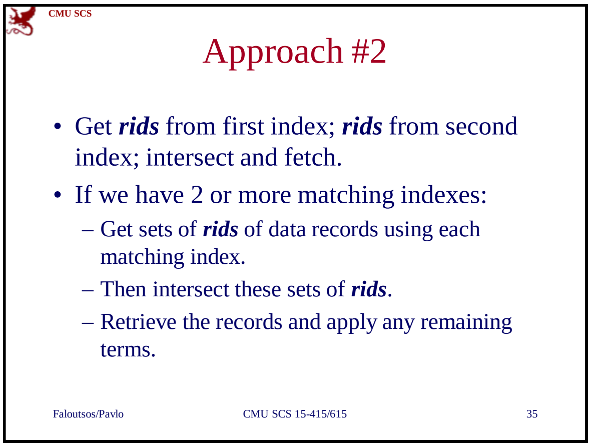
CMU SCS

Approach #1 – Example

```
(day<'2014-02-01' AND bid=5 AND sid=3)
```

- How about a B+tree on `<rname,day>`?
- How about a B+tree on `<day,rname>`?
- How about a Hash index on `<day,rname>`?

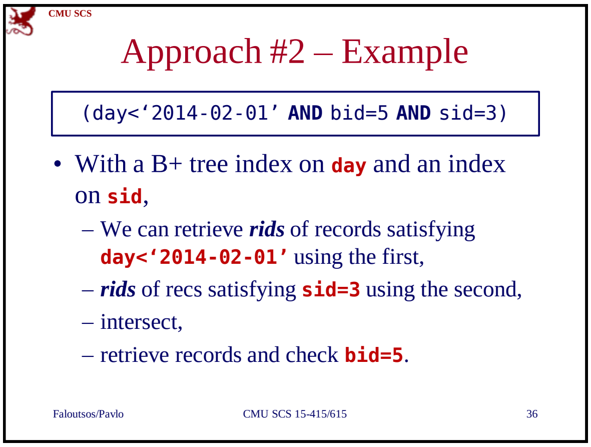
What if we have multiple indexes?



CMU SCS

Approach #2

- Get ***rids*** from first index; ***rids*** from second index; intersect and fetch.
- If we have 2 or more matching indexes:
  - Get sets of ***rids*** of data records using each matching index.
  - Then intersect these sets of ***rids***.
  - Retrieve the records and apply any remaining terms.

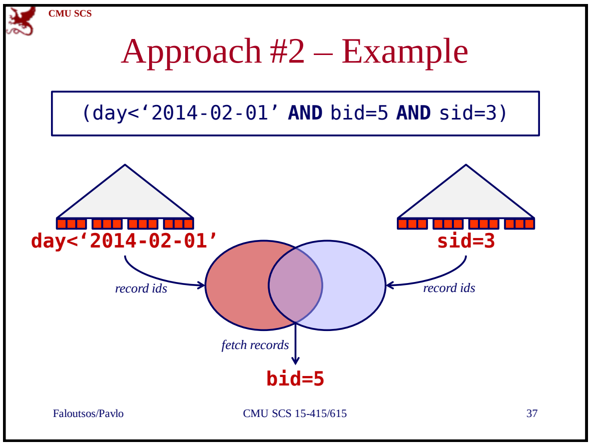


CMU SCS

Approach #2 – Example

```
(day<'2014-02-01' AND bid=5 AND sid=3)
```

- With a B+ tree index on `day` and an index on `sid`,
  - We can retrieve ***rids*** of records satisfying `day<'2014-02-01'` using the first,
  - ***rids*** of recs satisfying `sid=3` using the second,
  - intersect,
  - retrieve records and check `bid=5`.



CMU SCS

Approach #2 – Example

`(day<‘2014-02-01’ AND bid=5 AND sid=3)`

day<‘2014-02-01’

sid=3

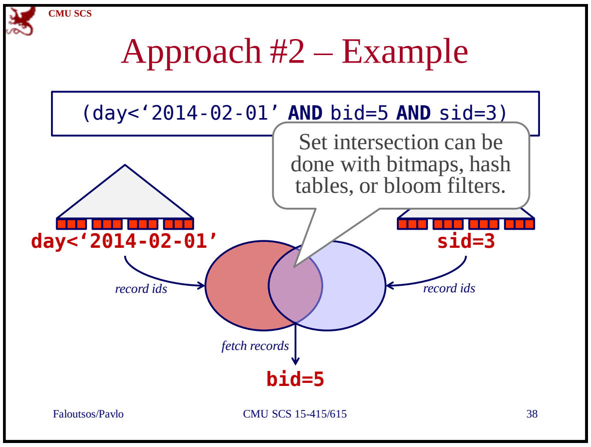
record ids

record ids

fetch records

bid=5

Faloutsos/Pavlo CMU SCS 15-415/615 37



CMU SCS

Approach #2 – Example

`(day<‘2014-02-01’ AND bid=5 AND sid=3)`

Set intersection can be done with bitmaps, hash tables, or bloom filters.

day<‘2014-02-01’

sid=3

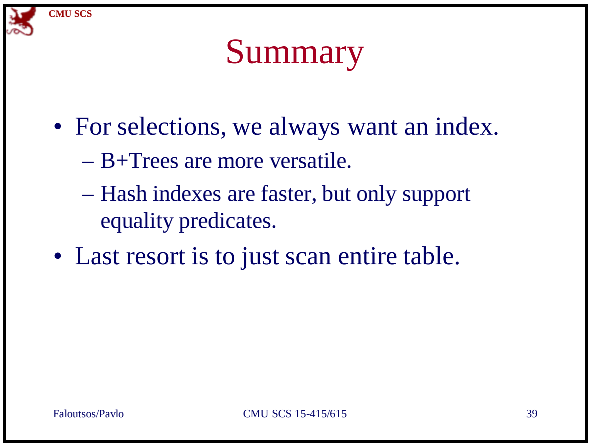
record ids

record ids

fetch records

bid=5

Faloutsos/Pavlo CMU SCS 15-415/615 38

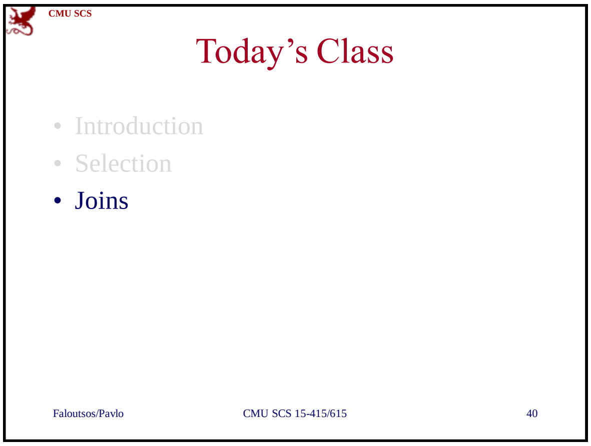


CMU SCS

Summary

- For selections, we always want an index.
  - B+Trees are more versatile.
  - Hash indexes are faster, but only support equality predicates.
- Last resort is to just scan entire table.

Faloutsos/Pavlo CMU SCS 15-415/615 39

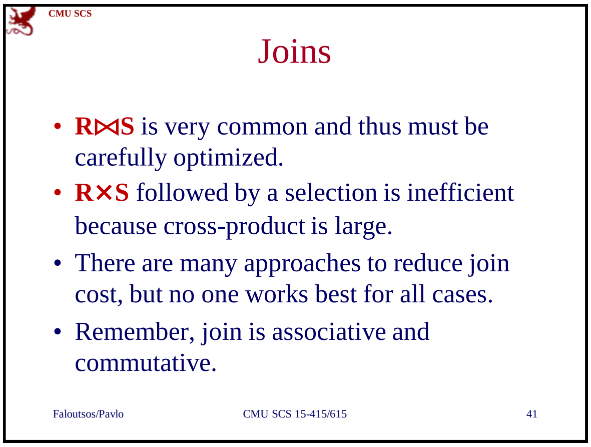


CMU SCS

Today's Class

- Introduction
- Selection
- Joins

Faloutsos/Pavlo CMU SCS 15-415/615 40

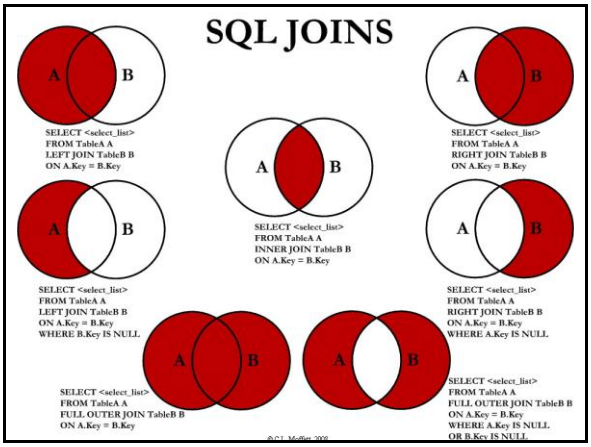


CMU SCS

Joins

- **R⋈S** is very common and thus must be carefully optimized.
- **R×S** followed by a selection is inefficient because cross-product is large.
- There are many approaches to reduce join cost, but no one works best for all cases.
- Remember, join is associative and commutative.

Faloutsos/Pavlo CMU SCS 15-415/615 41



SQL JOINS

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

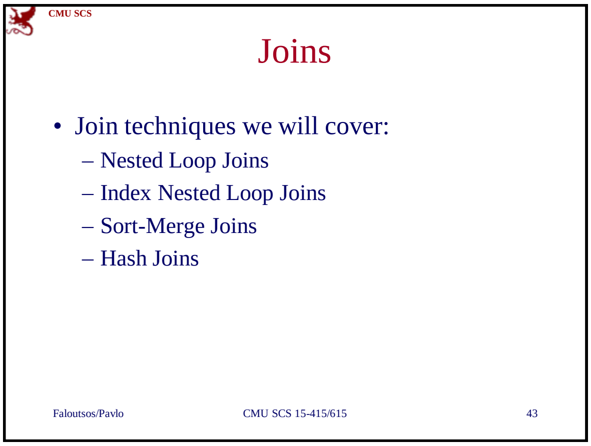
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

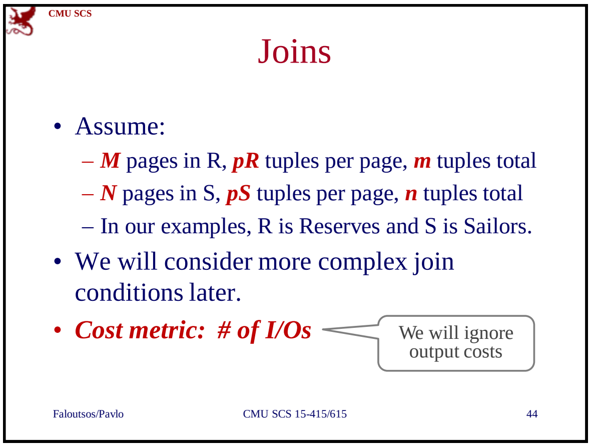
© C.L. Moffatt 2008



CMU SCS

Joins

- Join techniques we will cover:
  - Nested Loop Joins
  - Index Nested Loop Joins
  - Sort-Merge Joins
  - Hash Joins

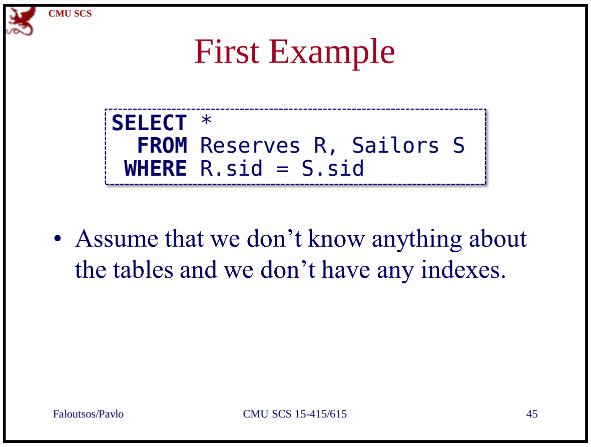


CMU SCS

Joins

- Assume:
  - ***M*** pages in R, ***pR*** tuples per page, ***m*** tuples total
  - ***N*** pages in S, ***pS*** tuples per page, ***n*** tuples total
  - In our examples, R is Reserves and S is Sailors.
- We will consider more complex join conditions later.
- ***Cost metric: # of I/Os***

We will ignore output costs

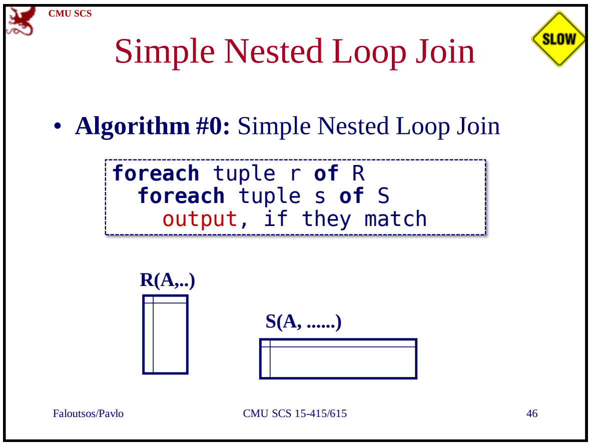


CMU SCS

First Example

```
SELECT *
  FROM Reserves R, Sailors S
 WHERE R.sid = S.sid
```

- Assume that we don’t know anything about the tables and we don’t have any indexes.

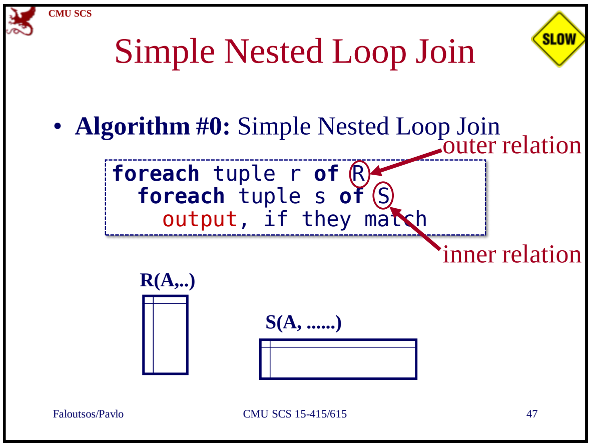


CMU SCS

Simple Nested Loop Join

- **Algorithm #0:** Simple Nested Loop Join

```
foreach tuple r of R
  foreach tuple s of S
    output, if they match
```



CMU SCS

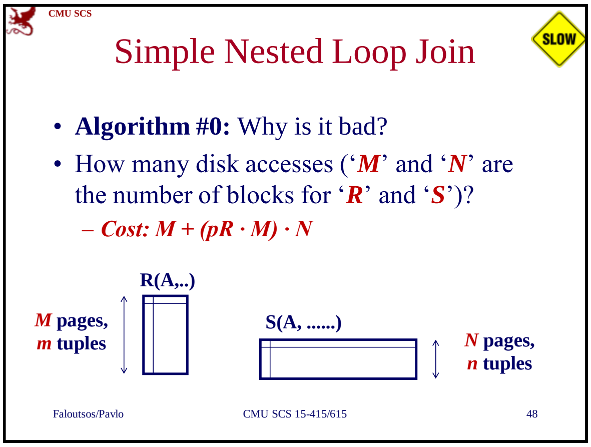
Simple Nested Loop Join

- **Algorithm #0:** Simple Nested Loop Join

```
foreach tuple r of R
  foreach tuple s of S
    output, if they match
```

outer relation

inner relation

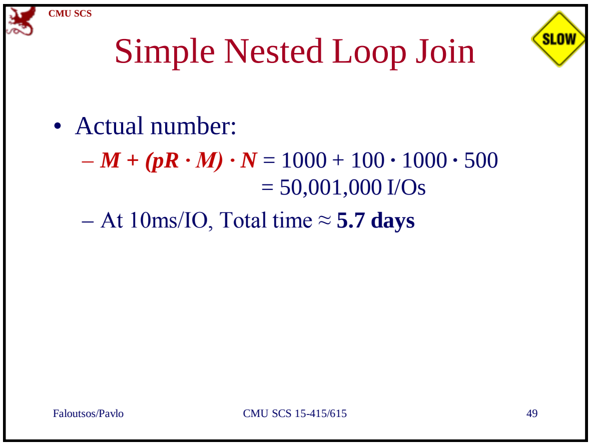


CMU SCS

Simple Nested Loop Join

- **Algorithm #0:** Why is it bad?
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: M + (pR · M) · N***

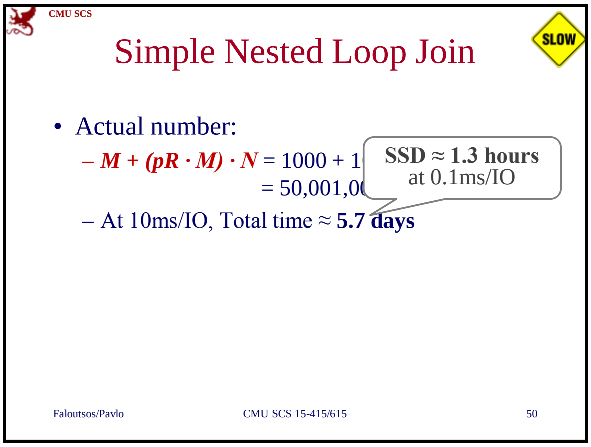
SLOW
R(A,..)
S(A, ...)
M pages,
m tuples
N pages,
n tuples



CMU SCS

Simple Nested Loop Join

- Actual number:
  - $M + (pR \cdot M) \cdot N = 1000 + 100 \cdot 1000 \cdot 500$
    $= 50{,}001{,}000$ I/Os
  - At 10ms/IO, Total time ≈ **5.7 days**

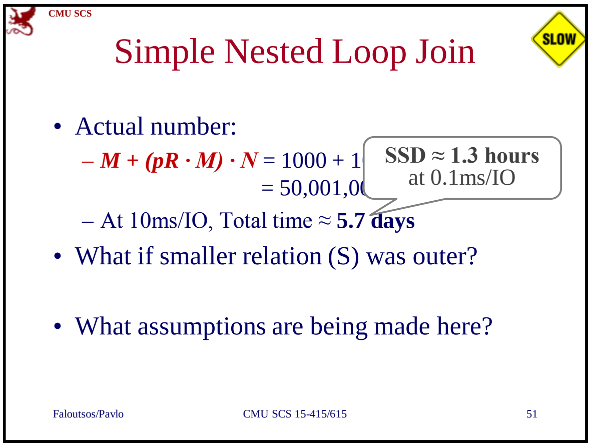


CMU SCS

Simple Nested Loop Join

- Actual number:
  - $M + (pR \cdot M) \cdot N = 1000 + 1$
    $= 50{,}001{,}0$
  - At 10ms/IO, Total time ≈ **5.7 days**

SSD ≈ 1.3 hours at 0.1ms/IO

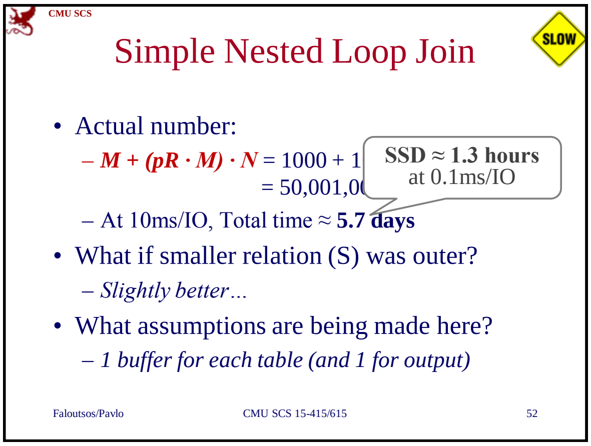


CMU SCS

Simple Nested Loop Join

- Actual number:
  - $M + (pR \cdot M) \cdot N = 1000 + 1$
    $= 50{,}001{,}0$
  - At 10ms/IO, Total time ≈ **5.7 days**
- What if smaller relation (S) was outer?
- What assumptions are being made here?

SSD ≈ 1.3 hours at 0.1ms/IO



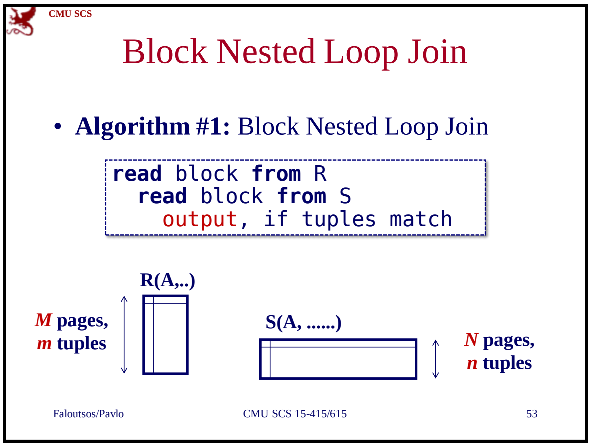
CMU SCS

Simple Nested Loop Join

SLOW

- Actual number:
  - $M + (pR \cdot M) \cdot N$ = 1000 + 1
    = 50,001,0
  - At 10ms/IO, Total time ≈ **5.7 days**
- What if smaller relation (S) was outer?
  - *Slightly better…*
- What assumptions are being made here?
  - *1 buffer for each table (and 1 for output)*

SSD ≈ **1.3 hours** at 0.1ms/IO



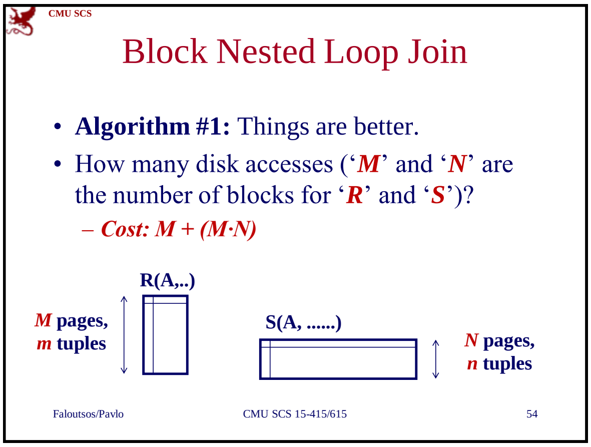
CMU SCS

Block Nested Loop Join

- **Algorithm #1:** Block Nested Loop Join

```
read block from R
  read block from S
    output, if tuples match
```

R(A,..)
M pages,
m tuples
S(A, ...)
N pages,
n tuples

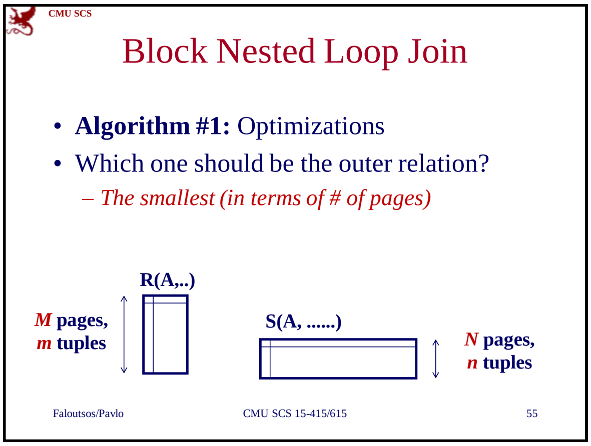


CMU SCS

Block Nested Loop Join

- **Algorithm #1:** Things are better.
- How many disk accesses ('***M***' and '***N***' are the number of blocks for '***R***' and '***S***')?
  - ***Cost: M + (M·N)***

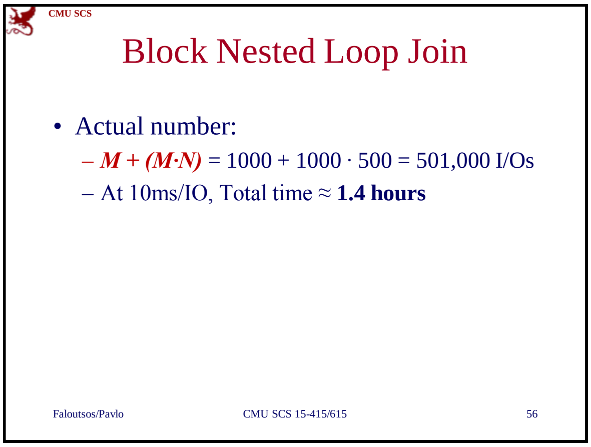
R(A,..)
M pages,
m tuples
S(A, ...)
N pages,
n tuples



CMU SCS

Block Nested Loop Join

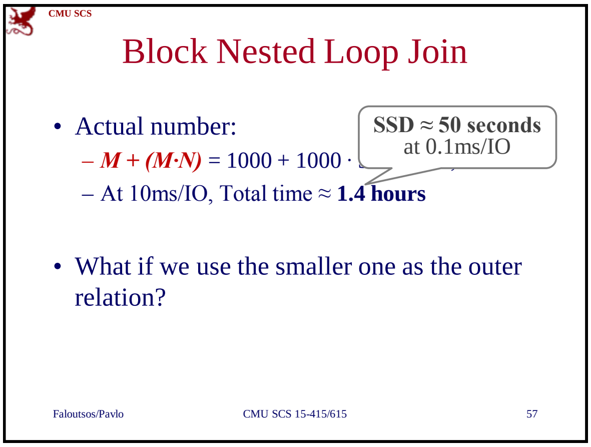
- **Algorithm #1:** Optimizations
- Which one should be the outer relation?
  - *The smallest (in terms of # of pages)*



CMU SCS

Block Nested Loop Join

- Actual number:
  - ***M* + (*M*·*N*)** = 1000 + 1000 · 500 = 501,000 I/Os
  - At 10ms/IO, Total time ≈ **1.4 hours**



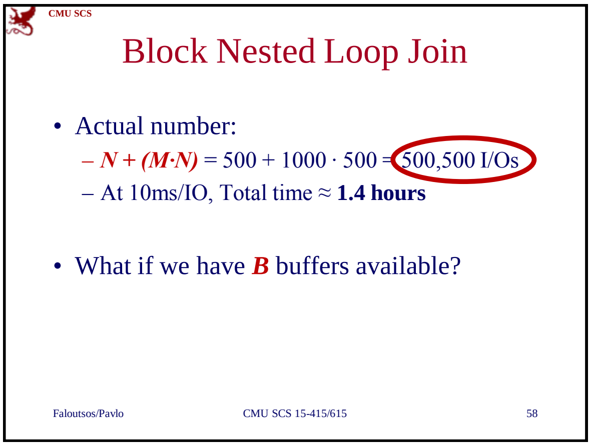
CMU SCS

Block Nested Loop Join

- Actual number:
  - ***M* + (*M*·*N*)** = 1000 + 1000 ·
  - At 10ms/IO, Total time ≈ **1.4 hours**

SSD ≈ 50 seconds at 0.1ms/IO

- What if we use the smaller one as the outer relation?

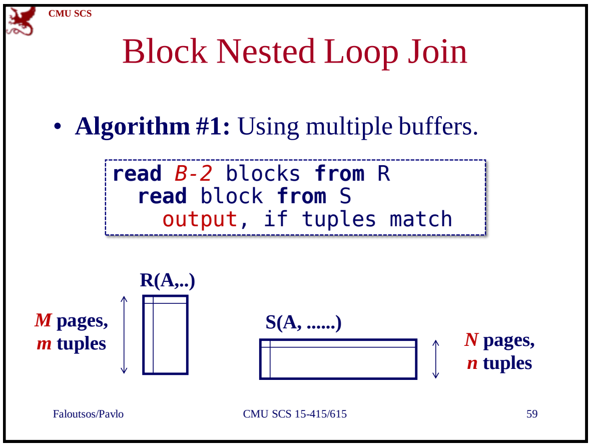


CMU SCS

Block Nested Loop Join

- Actual number:
  - $N + (M \cdot N) = 500 + 1000 \cdot 500 = \boxed{500{,}500 \text{ I/Os}}$
  - At 10ms/IO, Total time ≈ **1.4 hours**
- What if we have ***B*** buffers available?

Faloutsos/Pavlo CMU SCS 15-415/615 58



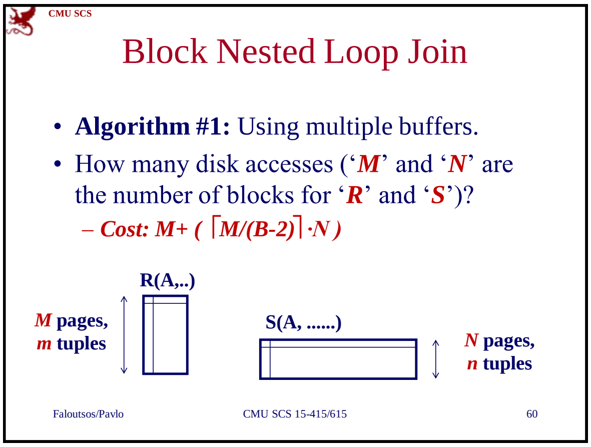
CMU SCS

Block Nested Loop Join

- **Algorithm #1:** Using multiple buffers.

```
read B-2 blocks from R
  read block from S
    output, if tuples match
```

Faloutsos/Pavlo CMU SCS 15-415/615 59

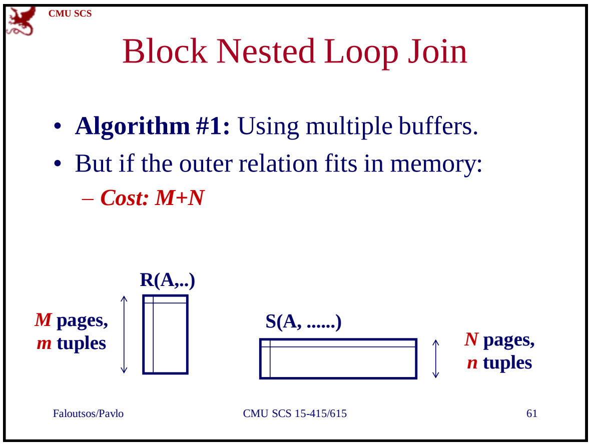


CMU SCS

Block Nested Loop Join

- **Algorithm #1:** Using multiple buffers.
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost:*** $M + (\lceil M/(B-2) \rceil \cdot N)$

Faloutsos/Pavlo CMU SCS 15-415/615 60



CMU SCS

Block Nested Loop Join

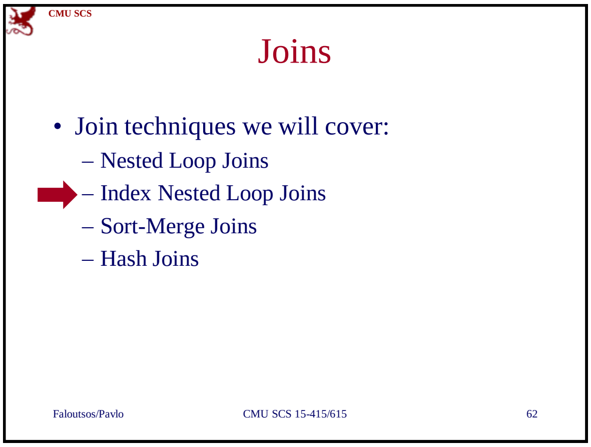
- **Algorithm #1:** Using multiple buffers.
- But if the outer relation fits in memory:
  - *Cost: M+N*

R(A,..)

M pages, *m* tuples

S(A, ...)

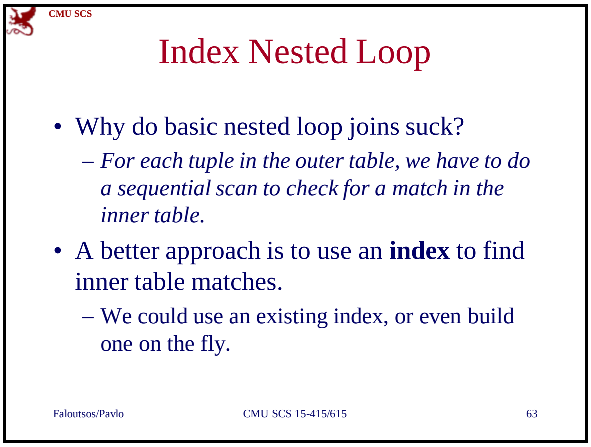
N pages, *n* tuples



CMU SCS

Joins

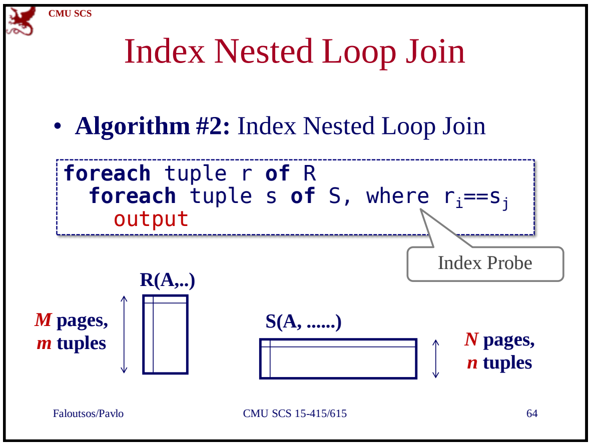
- Join techniques we will cover:
  - Nested Loop Joins
  - ➡ Index Nested Loop Joins
  - Sort-Merge Joins
  - Hash Joins



CMU SCS

Index Nested Loop

- Why do basic nested loop joins suck?
  - *For each tuple in the outer table, we have to do a sequential scan to check for a match in the inner table.*
- A better approach is to use an **index** to find inner table matches.
  - We could use an existing index, or even build one on the fly.



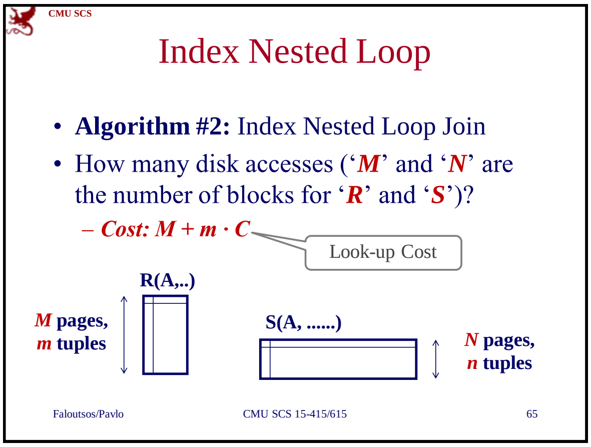
CMU SCS

Index Nested Loop Join

- **Algorithm #2:** Index Nested Loop Join

```
foreach tuple r of R
  foreach tuple s of S, where r_i==s_j
    output
```

Index Probe

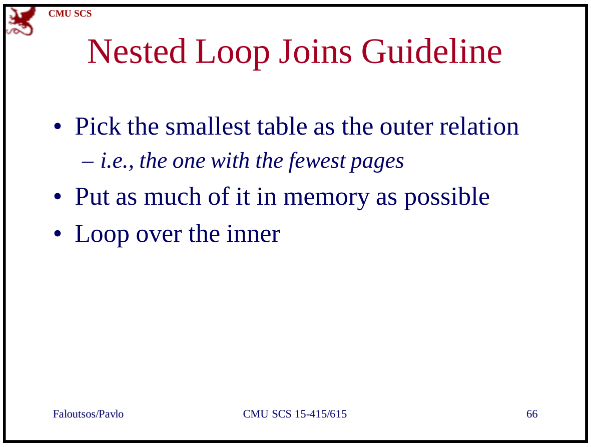


CMU SCS

Index Nested Loop

- **Algorithm #2:** Index Nested Loop Join
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost:*** $M + m \cdot C$

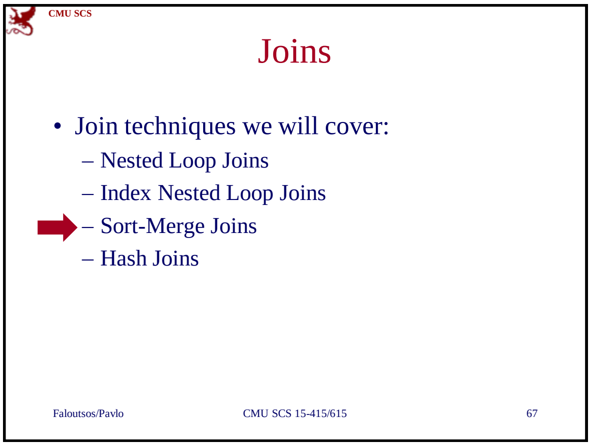
Look-up Cost



CMU SCS

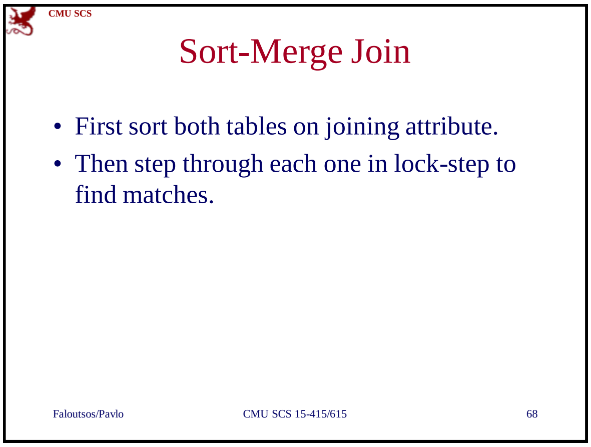
Nested Loop Joins Guideline

- Pick the smallest table as the outer relation
  - *i.e., the one with the fewest pages*
- Put as much of it in memory as possible
- Loop over the inner



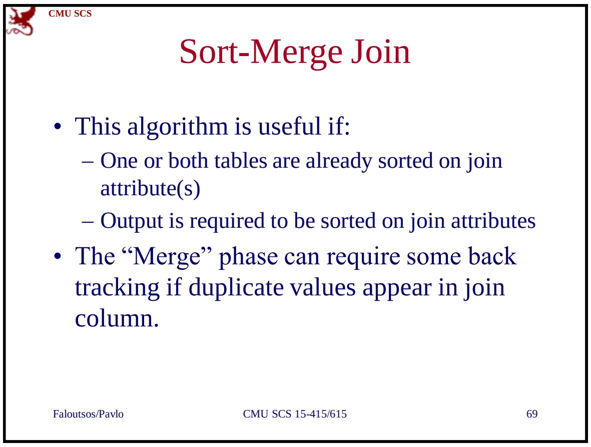
Joins

- Join techniques we will cover:
  - Nested Loop Joins
  - Index Nested Loop Joins
  - ➡ Sort-Merge Joins
  - Hash Joins



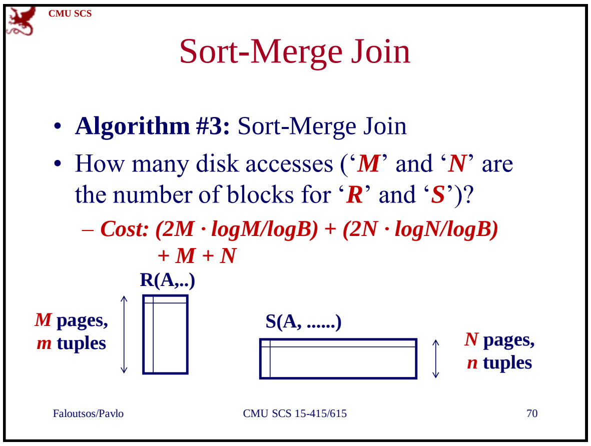
Sort-Merge Join

- First sort both tables on joining attribute.
- Then step through each one in lock-step to find matches.



Sort-Merge Join

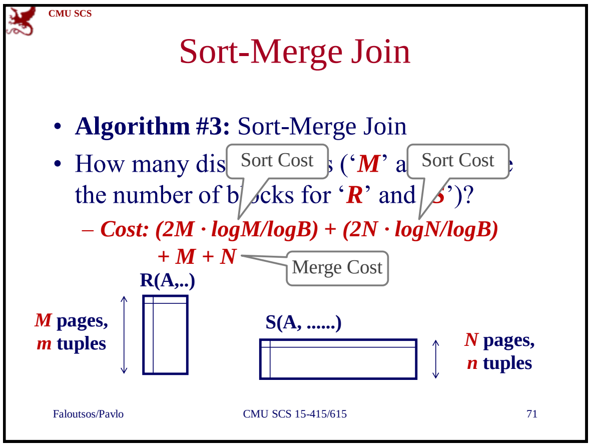
- This algorithm is useful if:
  - One or both tables are already sorted on join attribute(s)
  - Output is required to be sorted on join attributes
- The “Merge” phase can require some back tracking if duplicate values appear in join column.



CMU SCS

Sort-Merge Join

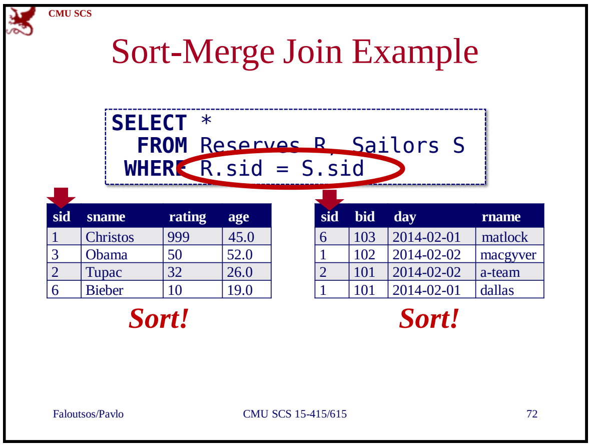
- **Algorithm #3:** Sort-Merge Join
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: (2M · logM/logB) + (2N · logN/logB) + M + N***



CMU SCS

Sort-Merge Join

- **Algorithm #3:** Sort-Merge Join
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: (2M · logM/logB) + (2N · logN/logB) + M + N***



CMU SCS

Sort-Merge Join Example

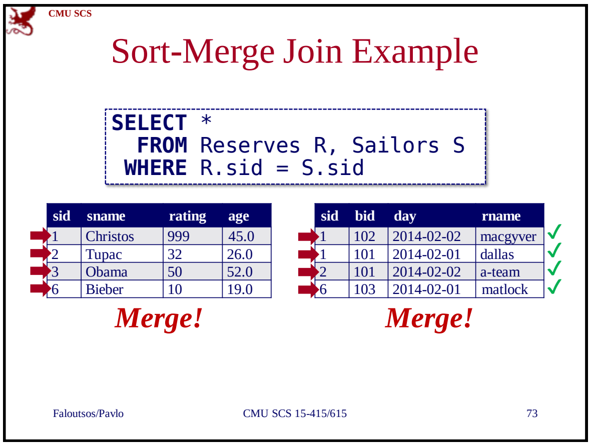
```
SELECT *
  FROM Reserves R, Sailors S
 WHERE R.sid = S.sid
```

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

Sort!

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

Sort!



CMU SCS

Sort-Merge Join Example

```
SELECT *
  FROM Reserves R, Sailors S
 WHERE R.sid = S.sid
```

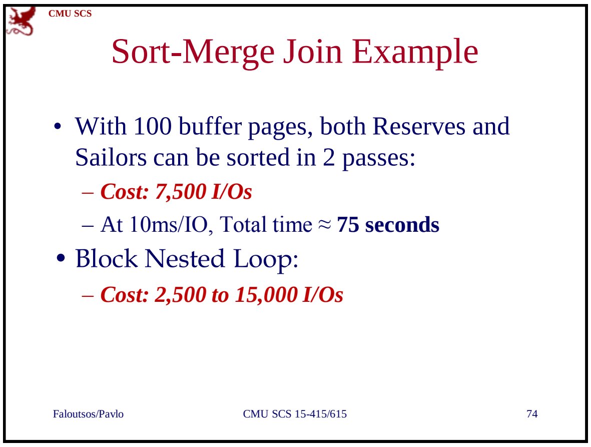
sid	sname	rating	age
1	Christos	999	45.0
2	Tupac	32	26.0
3	Obama	50	52.0
6	Bieber	10	19.0

Merge!

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

Merge!

Faloutsos/Pavlo CMU SCS 15-415/615 73

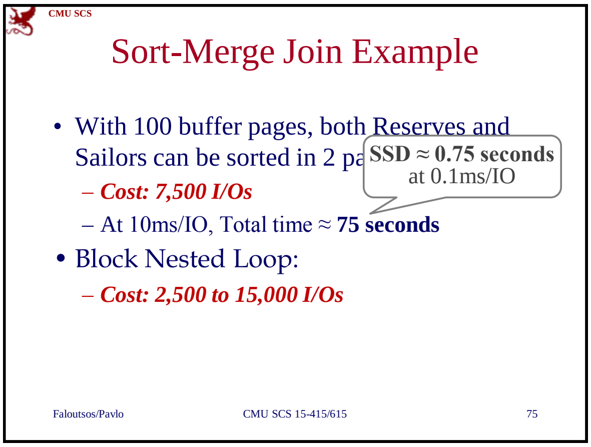


CMU SCS

Sort-Merge Join Example

- With 100 buffer pages, both Reserves and Sailors can be sorted in 2 passes:
  - ***Cost: 7,500 I/Os***
  - At 10ms/IO, Total time ≈ **75 seconds**
- Block Nested Loop:
  - ***Cost: 2,500 to 15,000 I/Os***

Faloutsos/Pavlo CMU SCS 15-415/615 74



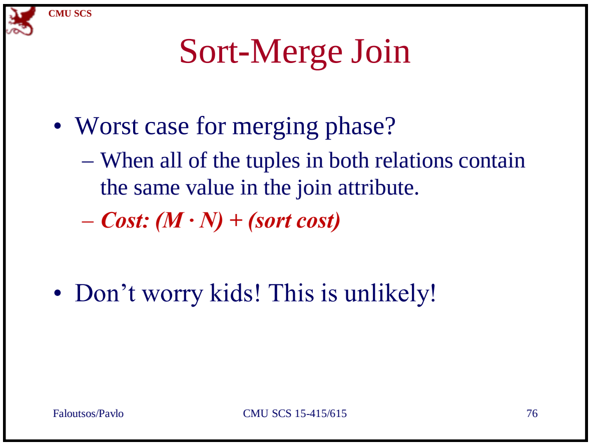
CMU SCS

Sort-Merge Join Example

- With 100 buffer pages, both Reserves and Sailors can be sorted in 2 passes:
  - ***Cost: 7,500 I/Os***
  - At 10ms/IO, Total time ≈ **75 seconds**
- Block Nested Loop:
  - ***Cost: 2,500 to 15,000 I/Os***

SSD ≈ 0.75 seconds
at 0.1ms/IO

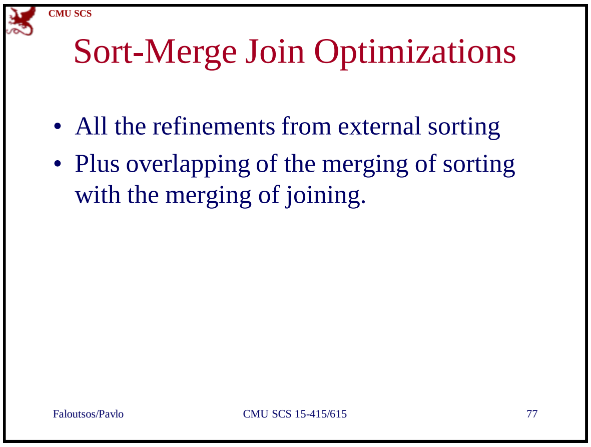
Faloutsos/Pavlo CMU SCS 15-415/615 75



CMU SCS

Sort-Merge Join

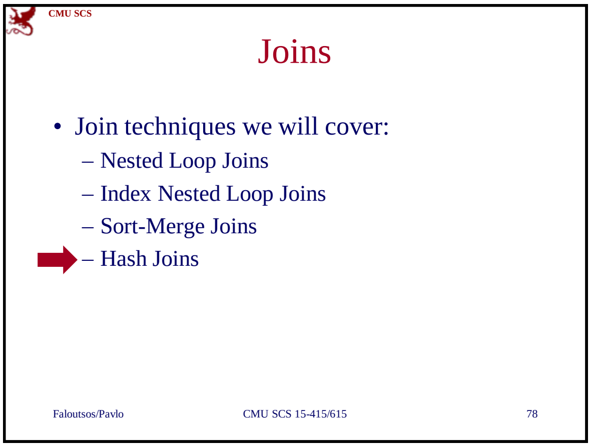
- Worst case for merging phase?
  - When all of the tuples in both relations contain the same value in the join attribute.
  - ***Cost: $(M \cdot N)$ + (sort cost)***
- Don’t worry kids! This is unlikely!



CMU SCS

Sort-Merge Join Optimizations

- All the refinements from external sorting
- Plus overlapping of the merging of sorting with the merging of joining.



CMU SCS

Joins

- Join techniques we will cover:
  - Nested Loop Joins
  - Index Nested Loop Joins
  - Sort-Merge Joins
  - ➡ Hash Joins

CMU SCS

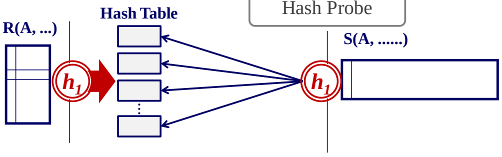
In-Memory Hash Join

- **Algorithm #4:** In-Memory

```
build hash table H for R
foreach tuple s of S
   output, if h(s_j)∈ H
```

This assumes H fits in memory!

Hash Probe



CMU SCS

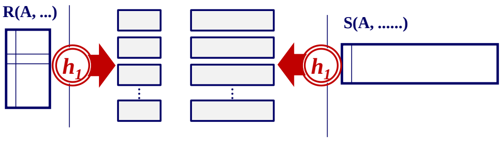
Grace Hash Join

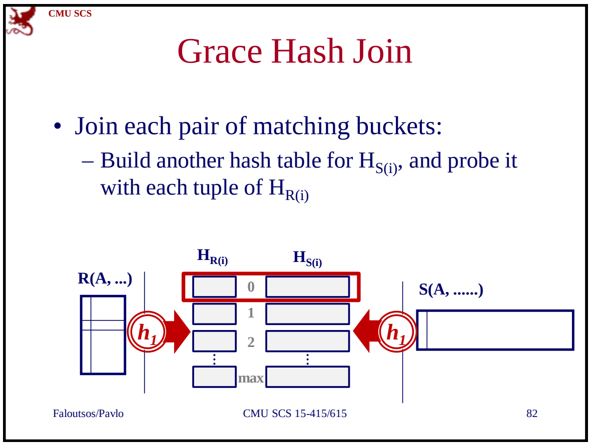
- Hash join when tables don’t fit in memory.
  - **Partition Phase:** Hash both tables on the join attribute into partitions.
  - **Probing Phase:** Compares tuples in corresponding partitions for each table.
- Named after the GRACE database machine.

CMU SCS

Grace Hash Join

- Hash R into (0, 1, ..., ‘max’) buckets
- Hash S into buckets (same hash function)

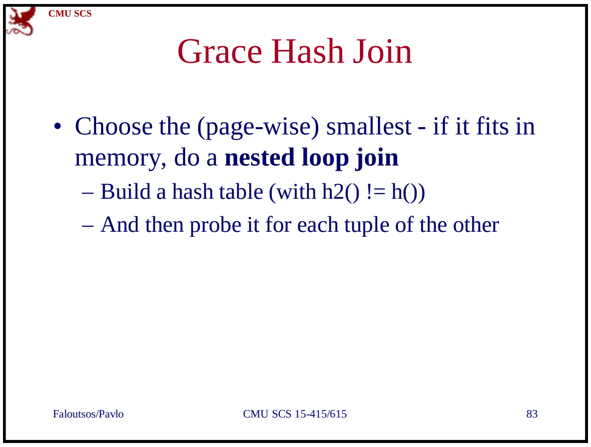




CMU SCS

Grace Hash Join

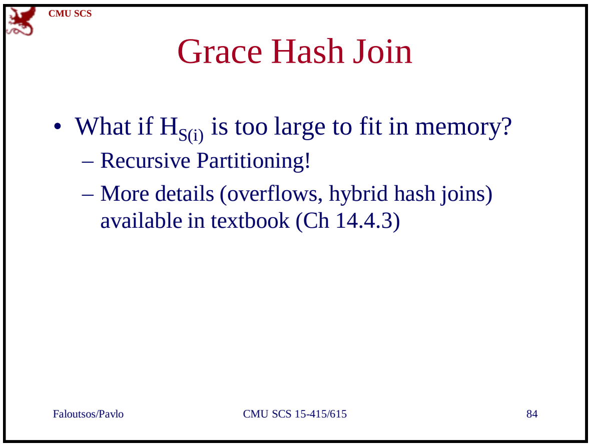
- Join each pair of matching buckets:
  - Build another hash table for $H_{S(i)}$, and probe it with each tuple of $H_{R(i)}$



CMU SCS

Grace Hash Join

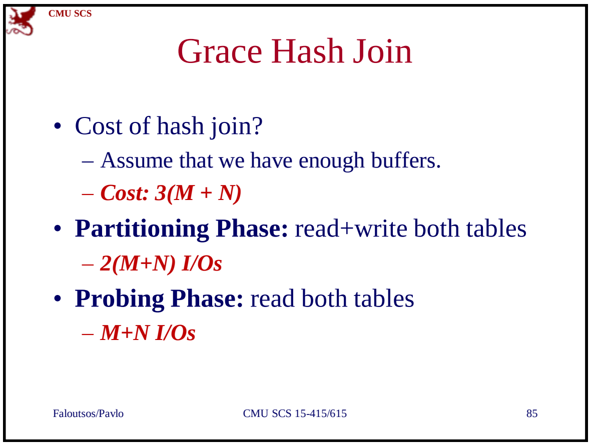
- Choose the (page-wise) smallest - if it fits in memory, do a **nested loop join**
  - Build a hash table (with h2() != h())
  - And then probe it for each tuple of the other



CMU SCS

Grace Hash Join

- What if $H_{S(i)}$ is too large to fit in memory?
  - Recursive Partitioning!
  - More details (overflows, hybrid hash joins) available in textbook (Ch 14.4.3)

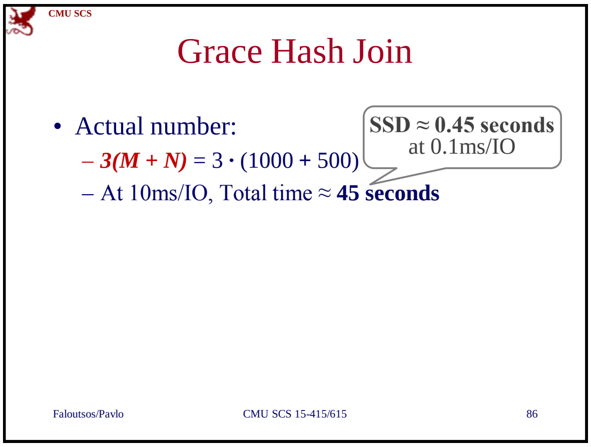


CMU SCS

Grace Hash Join

- Cost of hash join?
  - Assume that we have enough buffers.
  - ***Cost: 3(M + N)***
- **Partitioning Phase:** read+write both tables
  - ***2(M+N) I/Os***
- **Probing Phase:** read both tables
  - ***M+N I/Os***

Faloutsos/Pavlo CMU SCS 15-415/615 85



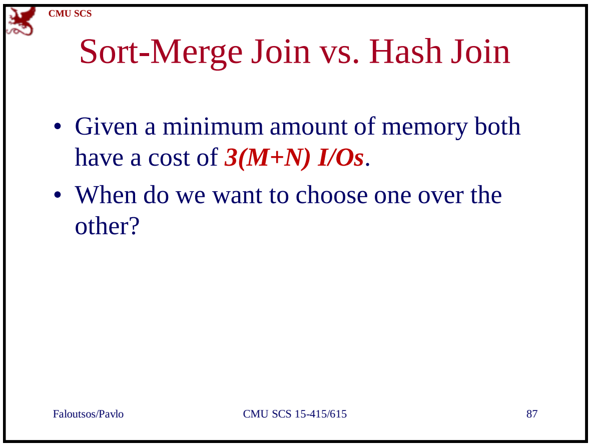
CMU SCS

Grace Hash Join

- Actual number:
  - ***3(M + N)*** = 3 · (1000 + 500)
  - At 10ms/IO, Total time ≈ **45 seconds**

SSD ≈ 0.45 seconds
at 0.1ms/IO

Faloutsos/Pavlo CMU SCS 15-415/615 86

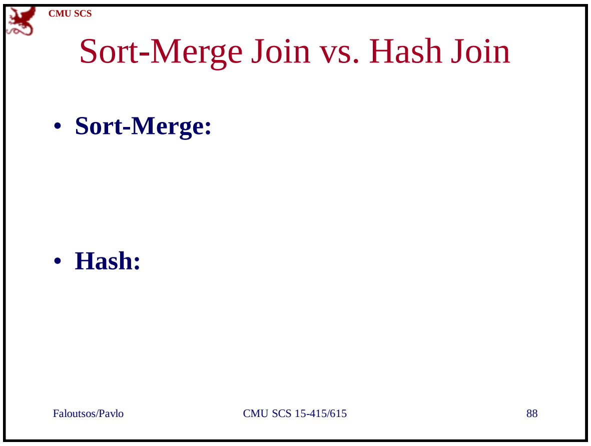


CMU SCS

Sort-Merge Join vs. Hash Join

- Given a minimum amount of memory both have a cost of ***3(M+N) I/Os***.
- When do we want to choose one over the other?

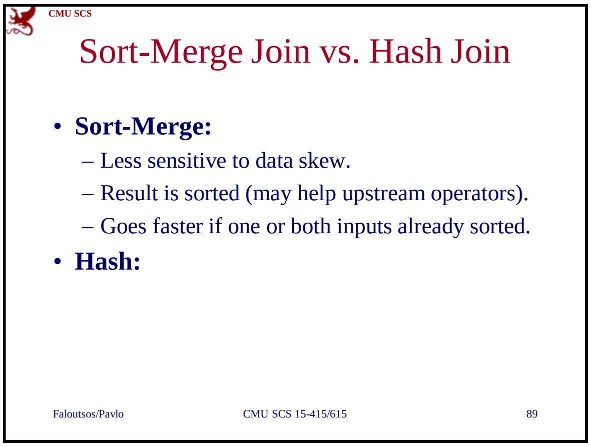
Faloutsos/Pavlo CMU SCS 15-415/615 87



CMU SCS

Sort-Merge Join vs. Hash Join

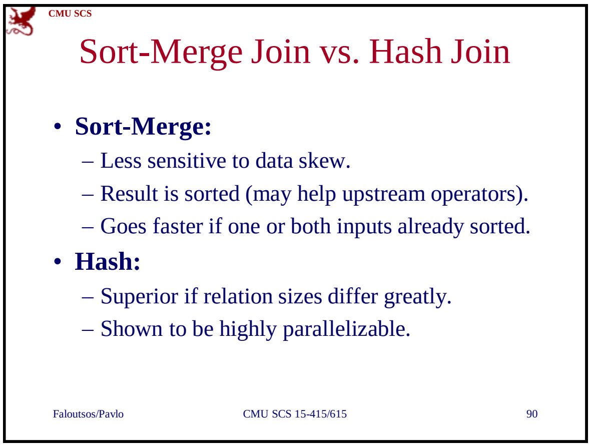
- **Sort-Merge:**
- **Hash:**



CMU SCS

Sort-Merge Join vs. Hash Join

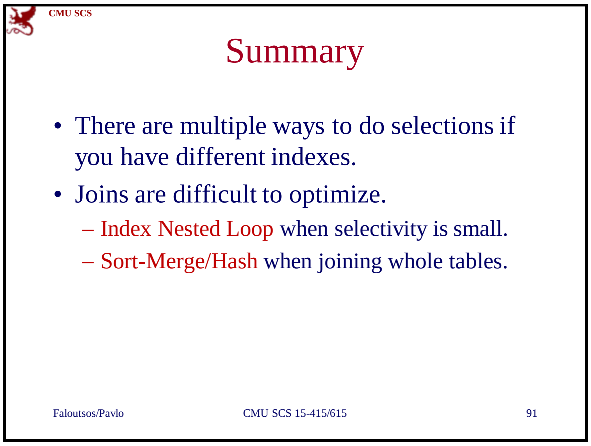
- **Sort-Merge:**
  - Less sensitive to data skew.
  - Result is sorted (may help upstream operators).
  - Goes faster if one or both inputs already sorted.
- **Hash:**



CMU SCS

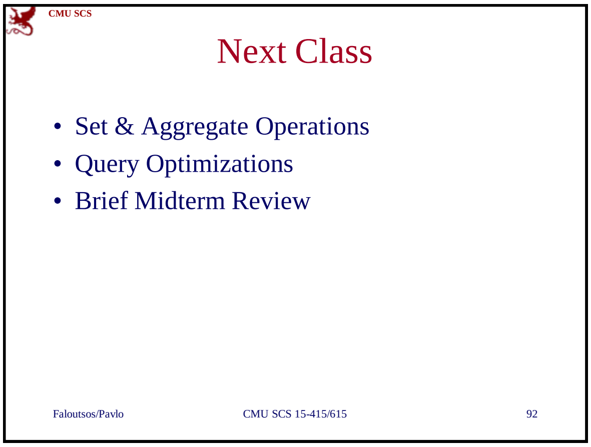
Sort-Merge Join vs. Hash Join

- **Sort-Merge:**
  - Less sensitive to data skew.
  - Result is sorted (may help upstream operators).
  - Goes faster if one or both inputs already sorted.
- **Hash:**
  - Superior if relation sizes differ greatly.
  - Shown to be highly parallelizable.



Summary

- There are multiple ways to do selections if you have different indexes.
- Joins are difficult to optimize.
  - Index Nested Loop when selectivity is small.
  - Sort-Merge/Hash when joining whole tables.



Next Class

- Set & Aggregate Operations
- Query Optimizations
- Brief Midterm Review