ICS 421 Spring 2010
Query Evaluation (ii)

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa

What do these queries have in common ?

```
SELECT  S.sname
FROM  Sailors S
WHERE S.rating>5
ORDER BY S.age
```

```
SELECT  DISTINCT S.sname
FROM  Sailors S
```

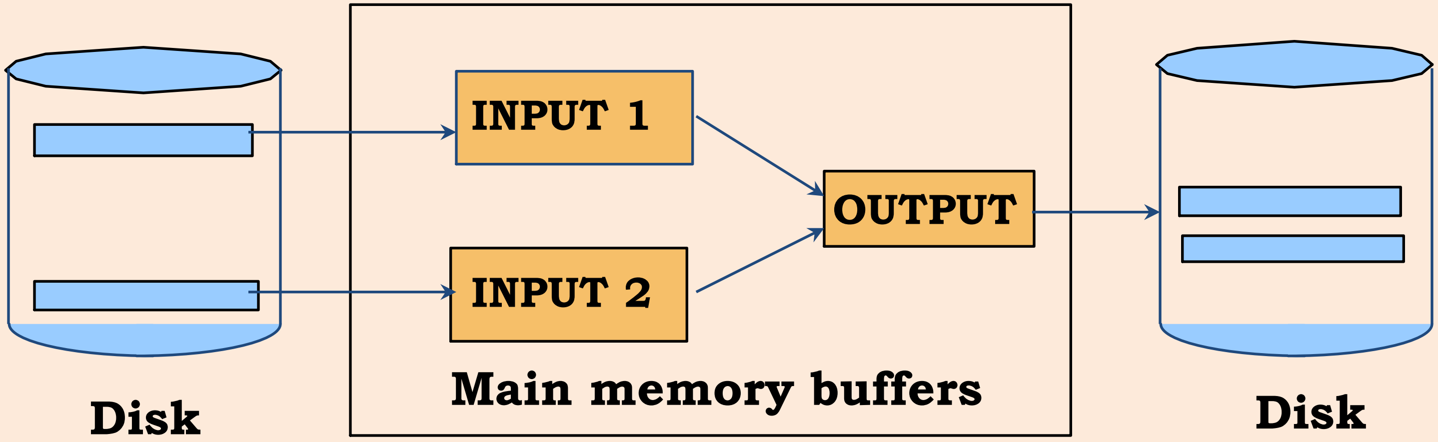
```
SELECT S.age, AVG(S.rating)
FROM  Sailors S
GROUP BY S.age
```

The Sort Operator

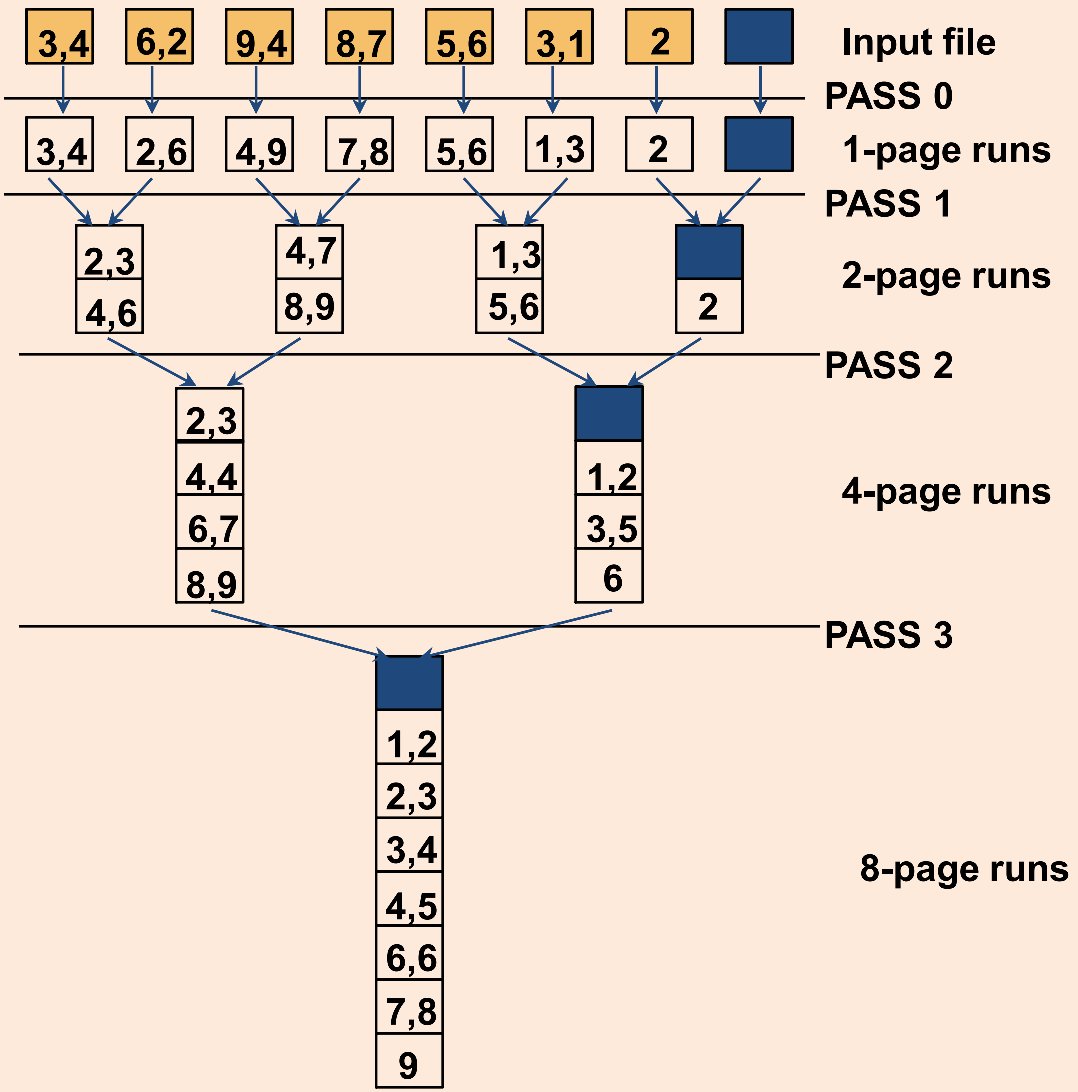
- Sorting is a classic problem in computer science!
- Data requested in sorted order
  - e.g., find students in increasing *gpa* order
- Sorting is first step in *bulk loading* B+ tree index.
- Sorting useful for eliminating *duplicate copies* in a collection of records (Why?)
- *Sort-merge* join algorithm involves sorting.
- Problem: sort 100Gb of data with 1Gb of RAM.
  - why not virtual memory?

Two-Way External Merge Sort

- Pass 0:
  - Read a page, sort it in memory, write it to disk
  - Only one buffer page is needed
- Pass 1, 2, 3, 4 …:
  - Read two (sorted) pages, merge them to fill output page, flush output page when full.
  - 2 input pages and 1 output page are needed

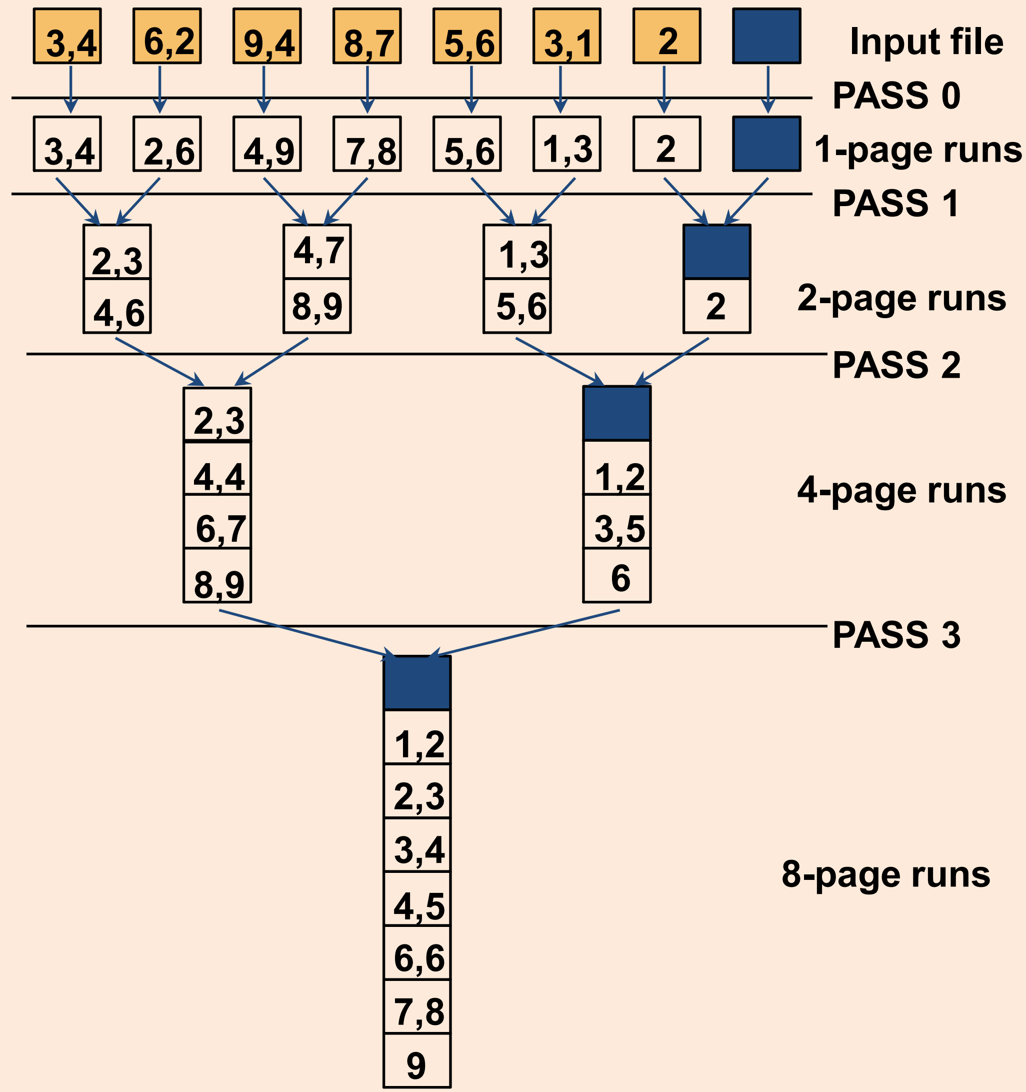


Two-Way Merge Sort: Example



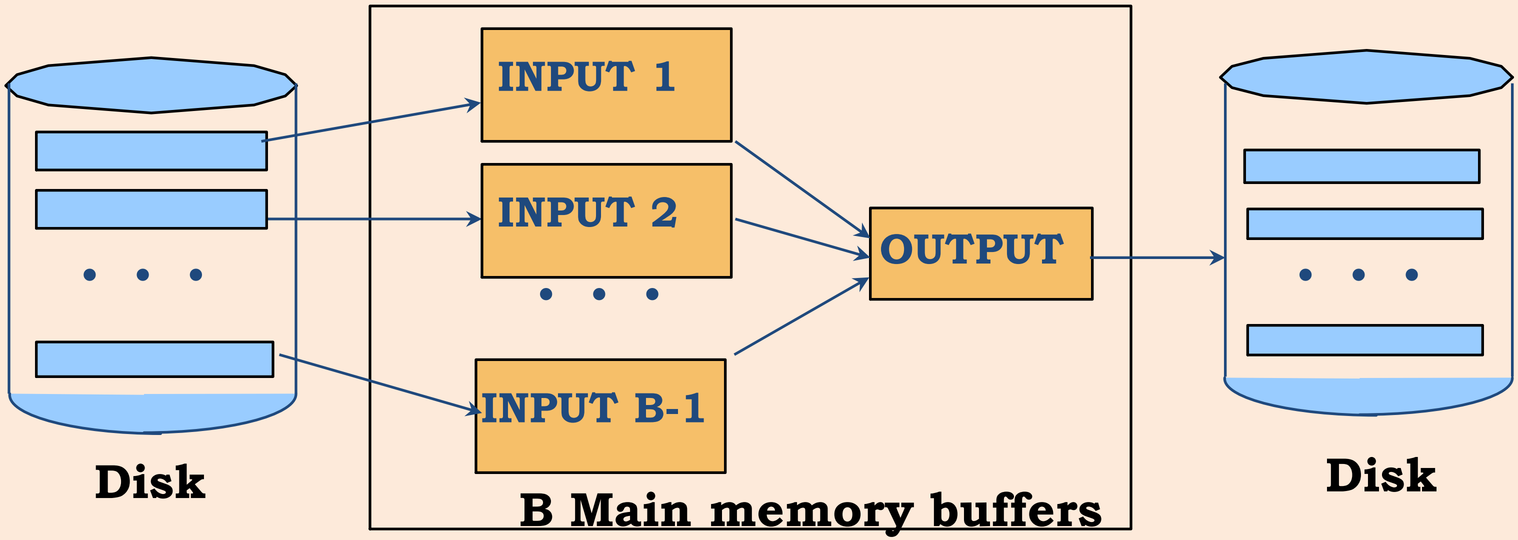
Two-Way Merge Sort: Analysis

- Input file has N pages
- Each pass reads N pages and writes N pages.
- The number of passes = $\lceil \log_2 N \rceil + 1$
- So total cost is = $2N(\lceil \log_2 N \rceil + 1)$
- *<u>Idea:</u>* ***Divide and conquer:*** sort subfiles and merge



K-Way External Merge Sort

- What if we have more memory ?
- Sort a file with *N* pages using *B* buffer pages:
  - Pass 0:
    - read in B pages, sort all B pages in memory, write to disk as 1 run, repeat until all N pages are sorted -- outputs $\lceil N/B \rceil$ sorted runs
  - Pass 1,2,…:
    - Use B-1 buffer pages as input and perform (B-1)-way merge to fill 1 output buffer page.



K-Way Merge Sort: Analysis

- B=5 buffer pages, N=108 pages
  - Pass 0: $\lceil 108/5 \rceil$=22 sorted runs of 5 pages each
  - Pass 1: $\lceil 22/4 \rceil$= 6 sorted runs of 20 pages each
  - Pass 2: $\lceil 6/4 \rceil$= 2 sorted runs of 80 & 28 pages
  - Pass 3: 1 sorted file of 108 pages
- Number of passes = $\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1$
- Each pass still reads N pages and writes N pages
- Total number of I/O's = 2N * ($\lceil \log_{B-1} \lceil N/B \rceil \rceil$+1)

Selection Operator

- Index vs Table Scan
- Multiple Indexes
  - Eg. Use index(age) & index(rating) for “age>20 AND rating>9”
  - Intersect RID sets using bloom filters
  - Eg. Use index(age) & index(rating) for “age>20 OR rating>9”
  - Union RID sets

Projection Operator

- Two steps:
  1. Remove unwanted columns
  2. Eliminate duplicates
- How do we do step 2 ?
  - External merge sort
  - Scan sorted data to remove duplicates
- Optimization: combine the 2 steps into merge sort:
  - Remove unwanted columns in Pass 0.
  - Subsequent passes can remove duplicates whenever they are encountered.

```
SELECT DISTINCT S.sname
FROM Sailors S
```

Join Algorithms

- Cost model
  - Single DBMS server: I/Os in number of pages
  - Distributed DBMS: network I/Os + local disk I/Os
  - t_d : time to read/write one page to local disk
  - t_s: time to ship one page over the network to another node
- Single server:
  - Nested Loop Join
  - Index Nested Loop Join
  - Sort Merge Join
  - Hash Join
- Distributed:
  - Semi-Join
  - Bloom Join

Nested Loop Join

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

$\bowtie$

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

```
For each data page P_S1 of S1
  For each tuple s in P_S1
    For each data page P_R1 of R1
      For each tuple r in P_R1
        if (s.sid==r.sid)
        then output s,r
```

- Worst case number of local disk reads

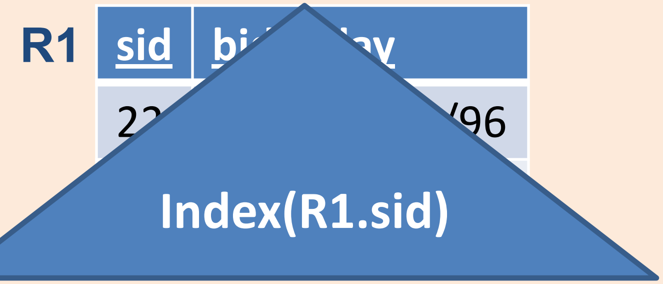
$$= \text{Npages}(S1) + |S1| * \text{Npages}(R1)$$

Index Nested Loop Join

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

$\bowtie$




```
For each data page P_S1 of S1
   For each tuple s in P_S1
      if (s.sid ∈ Index(R1.sid))
      then fetch r & output <s,r>
```

- Worst case number of local disk reads with tree index

  $= \text{Npages}(S1) + |S1| * (1 + \log_F \text{Npages}(\text{RIDS}(R1)))$

- Worst case number of local disk reads with hash index

  $= \text{Npages}(S1) + |S1| * 2$

Sort Merge Join

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

R1

sid	bid	day
19	100	8/8/99
22	101	10/10/96
22	99	10/12/95
58	103	11/12/96

```
1.  Sort S1 on SID
2.  Sort R1 on SID
3.  Compute join on SID using Merging algorithm
```

- If join attributes are relatively unique, the number of disk pages

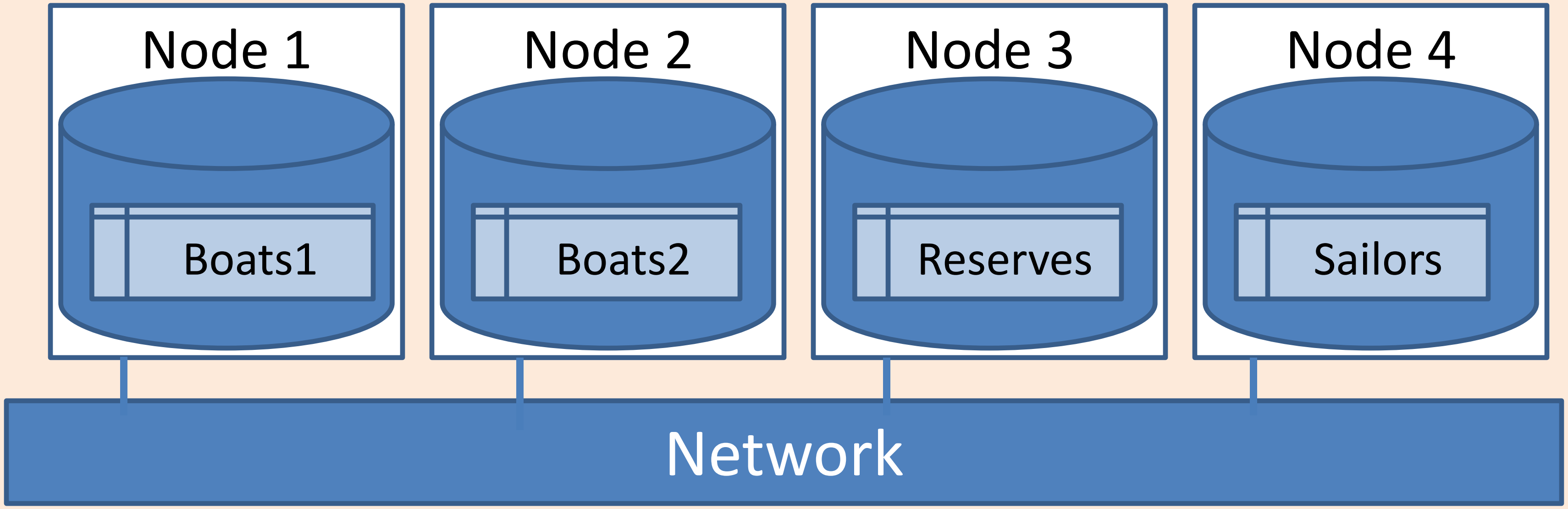
  = Npages(S1) log Npages(S1)

  + Npages(R1) log Npages(R1)

  + Npages(S1) + Npages(R1)

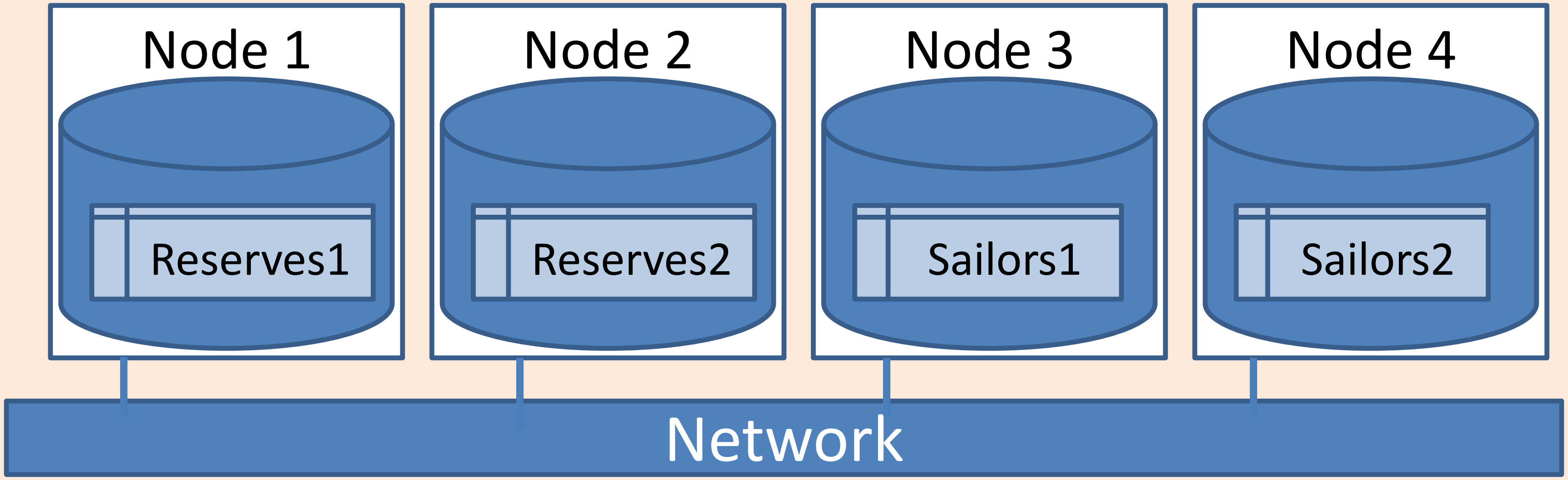
- If the number of duplicates in the join attributes is large, the number of disk pages approaches that of nested loop join.

Distributed Joins



- Consider:
  - Reserves join Sailors
- Depends on:
  - Which node get the query
  - Whether tables are fragmented/partitioned or not
- Node 1 gets query
  - Perform join at Node 3 (or 4) ship results to Node 1 ?
  - Ship tables to Node 1 ?
- Node 3 gets query
  - Fetch sailors in loop ?
  - Cache sailors locally ?

Distributed Joins over Fragments



R join S

$= \sigma_{R.sid=S.sid} (R \times S)$

$= \sigma_{R.sid=S.sid} ((R1 \cup R2) \times (S1 \cup S2))$

$= \sigma_{R.sid=S.sid} ((R1 \times S1) \cup (R1 \times S2) \cup (R2 \times S1) \cup (R2 \times S2))$

$= \sigma_{R.sid=S.sid} (R1 \times S1) \cup \sigma_{R.sid=S.sid} (R1 \times S2) \cup \sigma_{R.sid=S.sid} (R2 \times S1) \cup \sigma_{R.sid=S.sid} (R2 \times S2)$

$= (R1 \text{ join } S1) \cup (R1 \text{ join } S2) \cup (R2 \text{ join } S1) \cup (R2 \text{ join } S2)$

This equivalence applies to splitting a relation into pages in a single server DBMS system too!

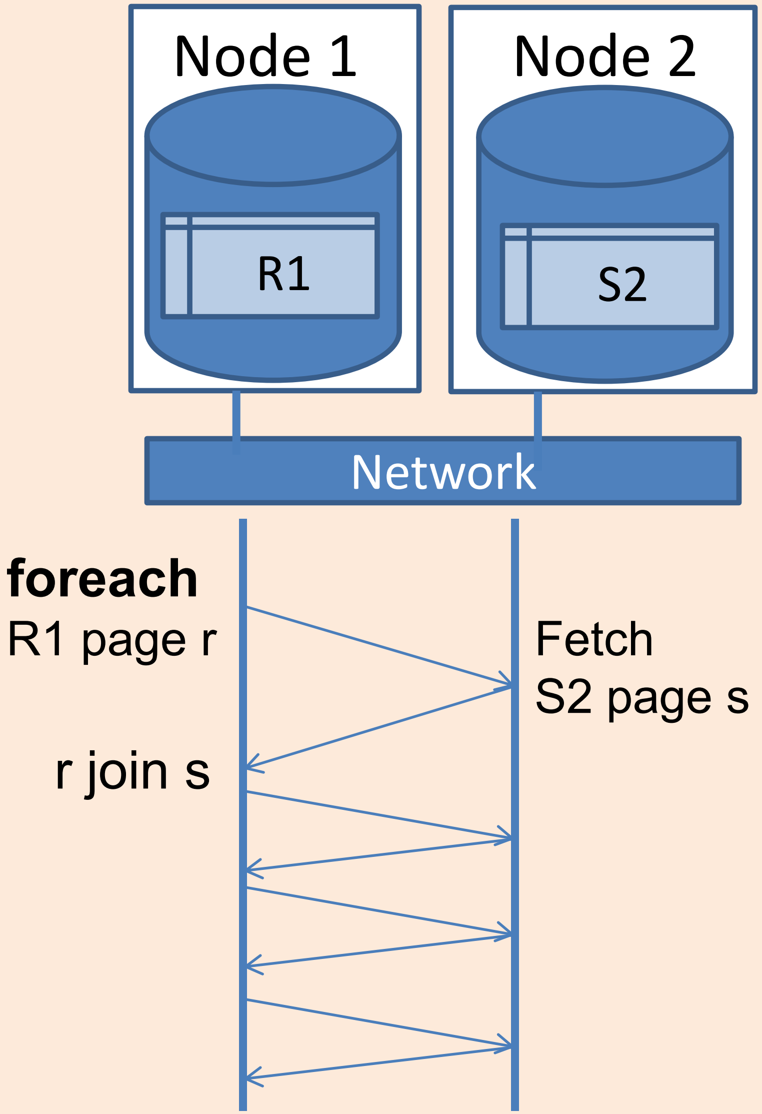
Equivalent to a union of joins over each pair of fragments

Distributed Nested Loop

- Consider performing R1 join S2 on Node 1
- Page-oriented nested loop join:

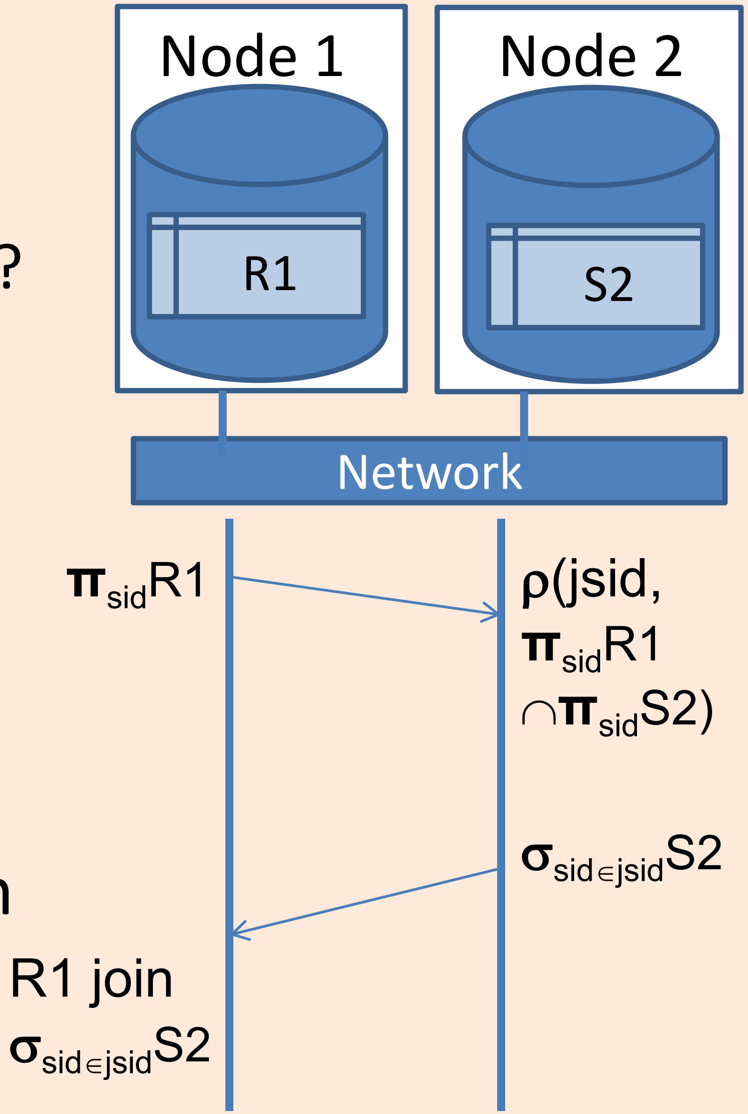
```
For each page r of R1
   Fetch r from local disk
   For each page s of S2
    Fetch s if s∉cache
    Output r join s
```

- Cost = Npages(R1)* t_d + Npages(R1)*Npages(S2)*(t_d + t_s)
- If cache can hold entire S2, cost is Npages(R1)* t_d + Npages(S2)* (t_d + t_s)



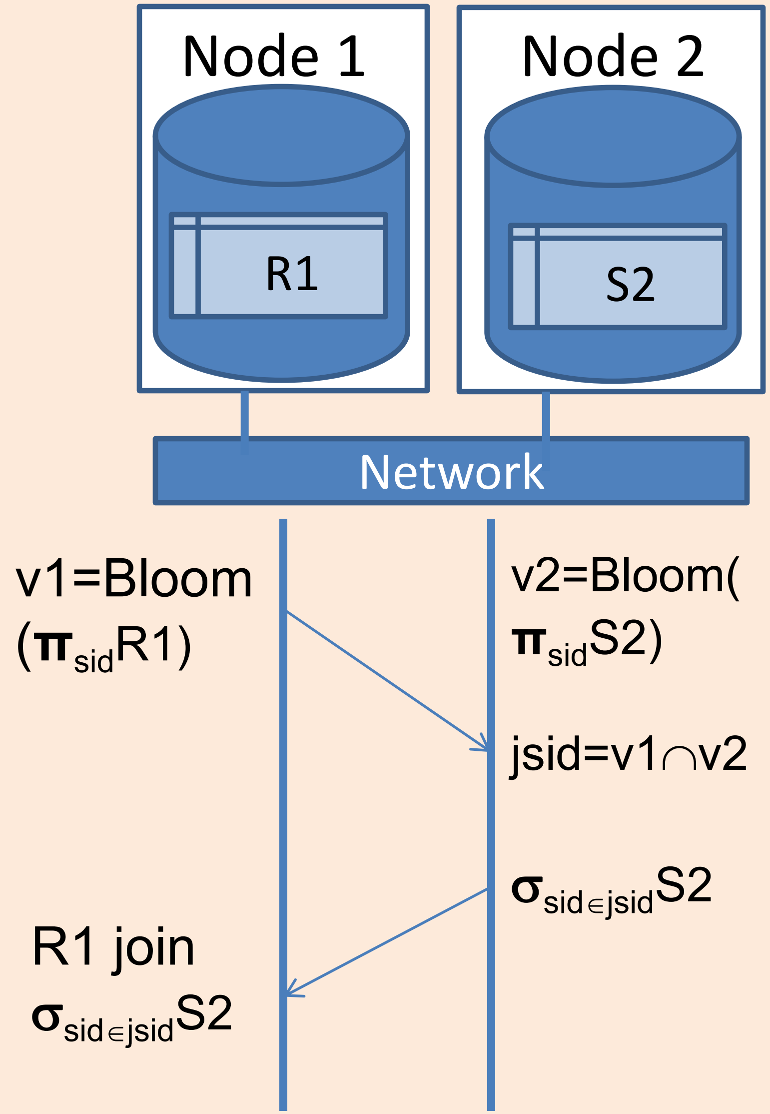
Semijoins

- Consider performing R1 join S2 on Node 1
- S2 needs to be shipped to R1
- Does every tuple in S2 join with R1 ?
- Semijoin:
  - Don't ship all of S2
  - Ship only those S2 rows that will join with R1
  - Assumes that the join causes a reduction in S2!
- Cost = Npages(R1)*t_d + Npages(π_{sid}R1)*t_s + Cost($\cap$) + Npages($\sigma_{sid \in jsid}$S2)*t_s + Cost(R1 join $\sigma_{sid \in jsid}$S2)



Bloomjoins

- Consider performing R1 join S2 on Node 1
- Can we do better than semijoin ?
- Bloomjoin:
  - Don't ship all of ($\boldsymbol{\pi}_{sid}$R1)
  - Node 1: Ship a "bloom filter" (like a signature) of ($\boldsymbol{\pi}_{sid}$R1)
    - Hash each sid
    - Set the bit for hash value in a bit vector
    - Send the bit vector v1
  - Node 2:
    - Hash each ($\boldsymbol{\pi}_{sid}$S2) to bit vector v2
    - Computer (v1 $\cap$ v2)
    - Send rows of S2 in the intersection
- False positives



Hash Join

R equijoin S on sid

1. Partition R into k partitions using hash function h1(R.sid)
2. Partition S into k partitions using hash function h1(S.sid)
3. Foreach partition i
   1. Build inmemory hash table H(R[i]) for R[i] using h2(R.sid)
   2. Foreach row in S[i]
      1. Probe H(R[i])
      2. Output join tuples <r,s>

- Works only on equi-joins
- Total I/Os = 2*NPages(R) + 2*NPages(S) + NPages(R) + NPages(S) = 3 * [Npages(R) + Npages(S)]
- Can be applied in a distributed DBMS with hash partitions on the join attribute!