# CSE 444: Database Internals

Lecture 23
Distributed Query Processing
and Optimization

# Readings

- Main textbook: Sections 20.3 and 20.4

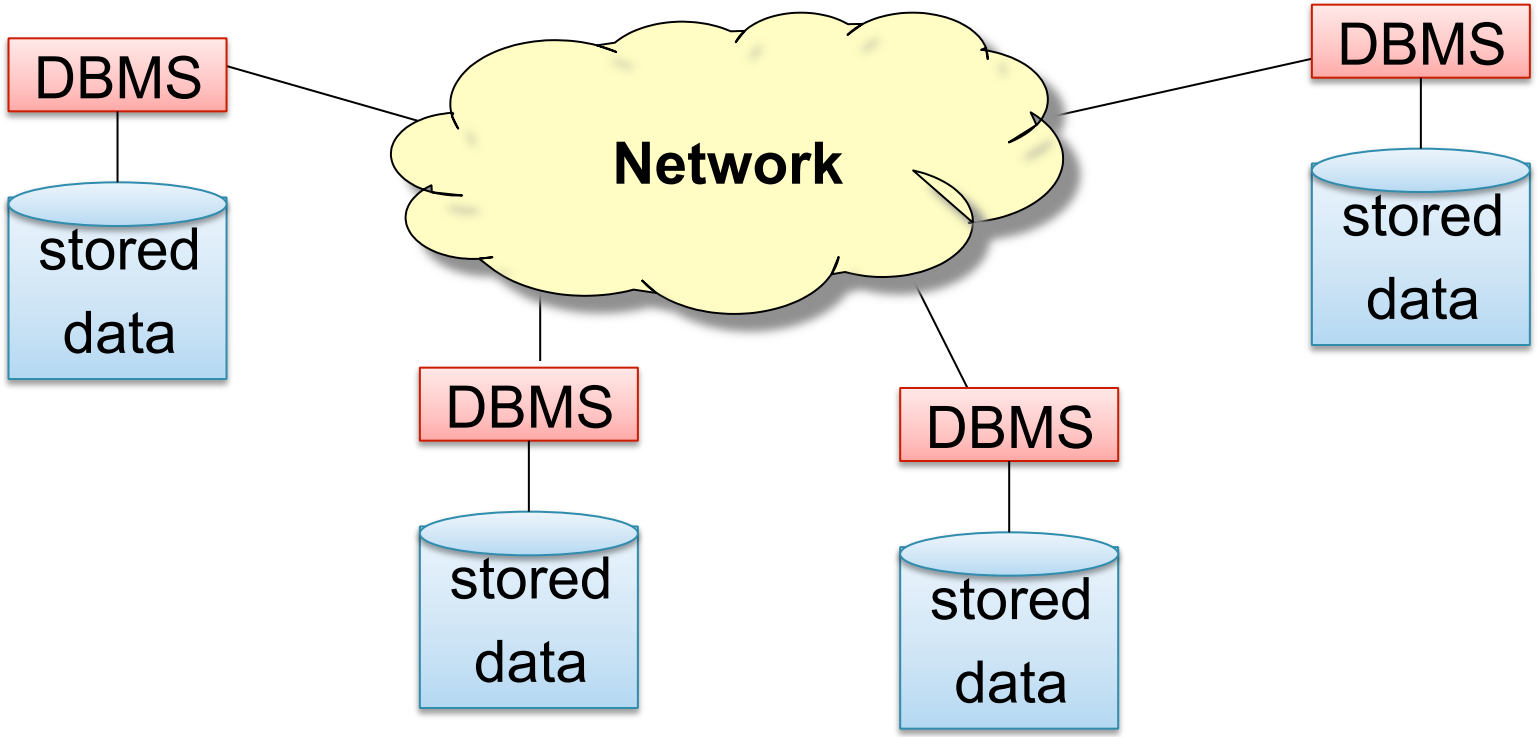
- Other textbook:
  Database management systems.
  Ramakrishnan and Gehrke.
  Third Ed. Chapter 22

# Outline

- Distributed DBMS motivation
- Distributed query optimization/processing
- Distributed DBMS limitations and challenges

# Distributed DBMS

- Important: many forms and definitions
- Our definition: shared nothing infrastructure
  - Multiple machines connected with a network
  - Machines can be far away from each other



# Reasons for a Distributed DBMS

- Scalability (ex: Amazon, eBay, Google)
  - Many small servers cheaper than one big server
  - Need to scale incrementally
  - This is the focus of ***parallel*** DBMSs
- Inherent distribution
  - Large organizations have data at multiple locations (different offices) -> original motivation
  - Different types of data in different DBMSs
  - Web-based and Internet-based applications
  - This is the focus of ***distributed*** DBMSs

# Distributed vs Parallel DBMS

- Parallel DBMS
  - Goal: improve performance through parallelization
  - Data distribution is governed solely by performance
  - Typically all machines are on the same cluster
- Distributed DBMS
  - Goal: manage data stored across several sites
  - Each site is administered autonomously
  - Each site capable of running independently of others

# Distributed Databases: Principles

- Distributed data independence:
  - Users should be able to write queries without specifying where the relations, or fragments of relations are stored
  - Generalizes physical data independence

- Distributed transaction atomicity:
  - Users should be able to write transactions that access and update data at several sites, as if they were stored on at a single site

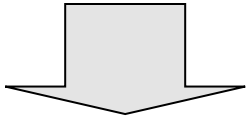
# Distributed DBMS Architectures

- Client-server
  - Client talks to different DBMSs separately
  - Back-end DBMSs do not know about each other
- Collaborating server
  - DBMS servers are inter-connected with each other
  - Servers know how to send parts of queries to others
- Middleware system
  - Only one DBMS server needs to know about all others and how to split queries among them

# Vertical Fragmentation

**Students**

| StudentID | Name | Address | GPA | Status |
|---|---|---|---|---|
| 234234 | Mary | Houston | 3.5 | Scholarship |
| 345345 | Sue | Seattle | 2.9 | Fellowship |
| 345343 | Joan | Seattle | 3.2 | Veteran |
| 234234 | Ann | Portland | 3.9 | None |



**Registrar**

| StudentID | Name | Address | GPA |
|---|---|---|---|
| 234234 | Mary | Houston | 3.5 |
| 345345 | Sue | Seattle | 2.9 |
| … | … | … | … |

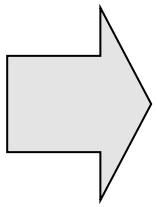
**Bursar**

| SID | Status |
|---|---|
| 234234 | Scholarship |
| 345345 | Fellowship |
| | |

# Horizontal Fragmentation

**Customers**

| SSN | Name | City | Country |
|---|---|---|---|
| 234234 | Mary | Houston | USA |
| 345345 | Sue | Seattle | USA |
| 345343 | Joan | Seattle | USA |
| 234234 | Ann | Portland | USA |
| -- | Frank | Calgary | Canada |
| -- | Jean | Montreal | Canada |



**CustomersInHouston**

| SSN | Name | City | Country |
|---|---|---|---|
| 234234 | Mary | Houston | USA |

**CustomersInSeattle**

| SSN | Name | City | Country |
|---|---|---|---|
| 345345 | Sue | Seattle | USA |
| 345343 | Joan | Seattle | USA |

**CustomersInCanada**

| SSN | Name | City | Country |
|---|---|---|---|
| -- | Frank | Calgary | Canada |
| -- | Jean | Montreal | Canada |

# Goals of a Distributed DBMS

- Shield users from distribution details
- Distribution transparency
  - Naming transparency
  - Location transparency
  - Fragmentation transparency
  - Performance transparency
    - Distributed query optimizer ensures similar performance no matter where query is submitted
  - Schema change and transaction transparency
- Replication transparency

# Distributed DBMS Features

- 70's and 80's, three main prototypes:
  - SDD-1, distributed INGRES, and R*
- Main components of a distributed DBMS
  - Defining data placement and fragmentation
  - Distributed catalog
  - Distributed query optimization (today)
  - Distributed transactions (next lecture)
  - Managing replicas (lecture after that)

# Outline

- Distributed DBMS motivation
- Distributed query optimization/processing
- Distributed DBMS limitations and challenges

# Review: Query Evaluation

Steps of query evaluation

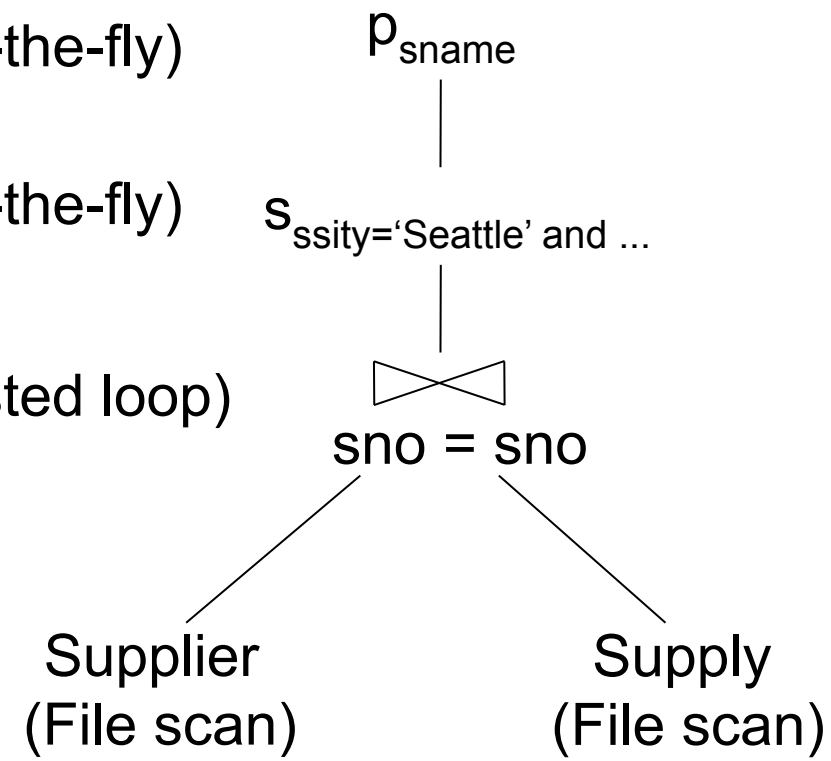
Query Parser

Query Rewrite

Optimizer

Executor

Query plan



# Review: Query Optimization

- Enumerate alternative plans
  - Many possible equivalent trees: e.g., join order
  - Many implementations for each operator
- Compute estimated cost of each plan
  - Compute number of I/Os and CPU utilization
  - Based on statistics
- Choose plan with lowest cost

# Distributed Query Optimization

- Search space is larger
  - Must select sites for joining relations
  - Must select method for shipping inner relation: whole or matches
- Minimize resource utilization
  - I/O, CPU, & ***communication costs***
  - Example cost function used in R*
  - $W_{CPU}\,Nb_{inst} + W_{I/O}\,Nb_{I/O} + W_{msg}\,Nb_{msg} + W_{byte}Nb_{bytes}$
- Could also try to minimize response time
  - Least cost plan != Fastest plan

# Inner Table Transfer Strategy

- Ship whole
  - Read inner relation at its home site (using index or not)
  - Apply any single-table predicates
  - Project inner relation to remove attributes not needed
  - Ship results to site of outer relation and store in temp file
    - Note: we lose any indexes on the inner relation
- Fetch matches
  - For each tuple of outer, project tuple on join column
  - Send value to site of inner relation
  - Find matching tuples from inner relation
  - Ship projected, matching tuples back

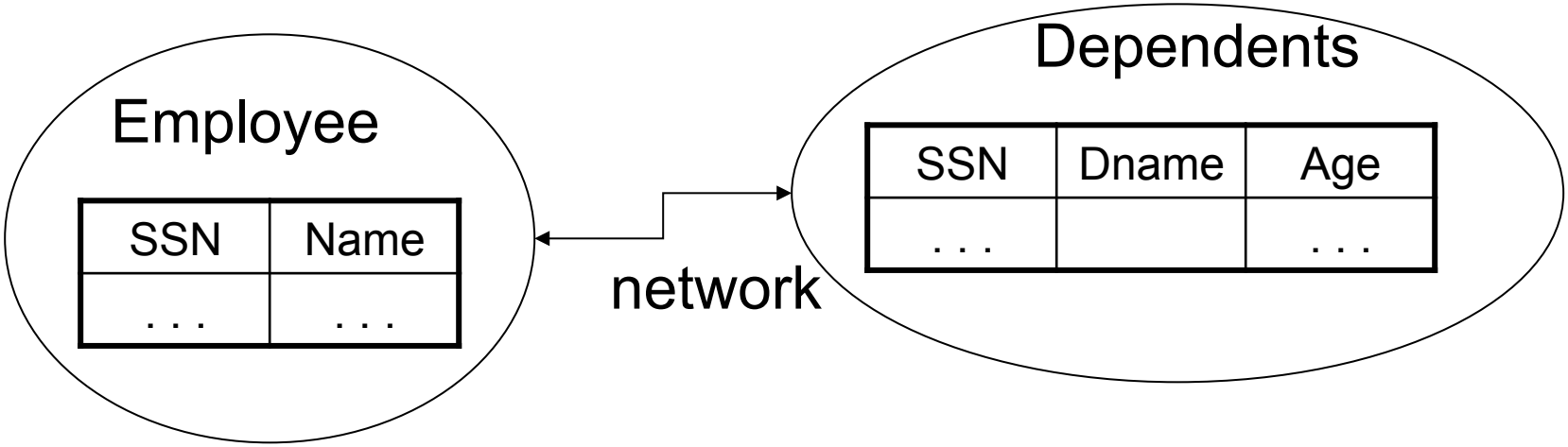
# Additional Join Strategies

- Dynamically-created temporary index on inner
  - Ship inner relation, store in temp table, index temp
- Semijoin
  - Project outer relation on join column (eliminate dups)
  - Ship projected column to site with inner relation
  - Compute natural join and ship matching tuples back
  - Join the two relations
- Bloomjoin
  - Same idea as semijoin, but use Bloom filter instead of sending all values in the join column
  - Bloom filter creates some false positives through collisions

# Semijoin

- $R \ltimes S = \Pi_{A1,\ldots,An} (R \ltimes S)$
- Where $A_1, \ldots, A_n$ are the attributes in R
- Example:
  - Employee $\ltimes$ Dependents

# Semijoins in Distributed Databases



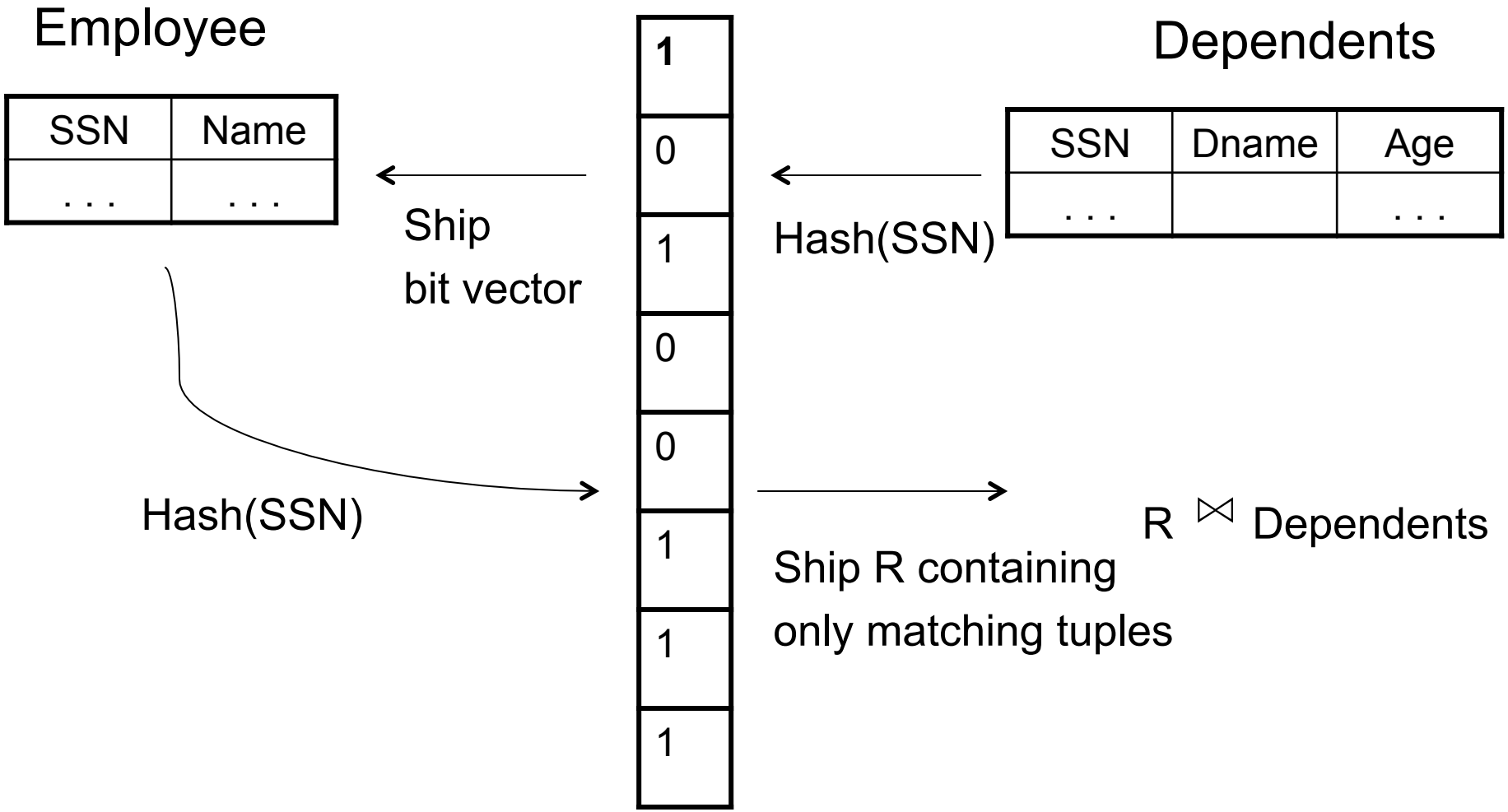

$$\text{Employee} \bowtie_{ssn=ssn} (\sigma_{age>71} (\text{Dependents}))$$

$$R = \text{Employee} \ltimes T$$

$$T = \Pi_{SSN} (\sigma_{age>71} (\text{Dependents}))$$

$$\text{Answer} = R \bowtie \text{Dependents}$$

# Bloomjoin


Employee
SSN
Name
. . .
. . .
Ship
bit vector
Hash(SSN)
1
0
1
0
0
1
1
1
Dependents
SSN
Dname
Age
. . .
. . .
Hash(SSN)
Hash(SSN)
Ship R containing
only matching tuples
R ⋈ Dependents

# Distributed vs Local Joins

Why can distributed joins be faster than local ones?

- More resources are available to the join
  - Ex: Distributed query can use twice the buffer pool (useful when accessing relations through unclustered indexes)
- Different parts of the join can proceed in parallel
  - Ex: Join tuples from page 1 while shipping page 2
  - Ex: Can sort the two relations in parallel

# Outline

- Distributed DBMS motivation
- Distributed query optimization/processing
- Distributed DBMS limitations and challenges

# Distributed DBMS Limitations

- Top-down
  - Global, a priori data placement
  - Global query optimization, one query at a time; no notion of load balance
  - Distributed transactions, tight coupling
- Assumes full cooperation of all sites
- Assumes uniform sites
- Assumes short-duration operations
- Limited scalability

# Distributed DBMS Challenges

In some distributed databases

- Autonomy: different administrative domains
  - Cannot always assume full cooperation
  - Do not want distributed transactions
- Heterogeneity
  - Different capabilities at different locations
  - Different data types, different semantics -> data integration problem
- Large-scale
  - Internet-scale query processor