Parallel and Distributed Databases

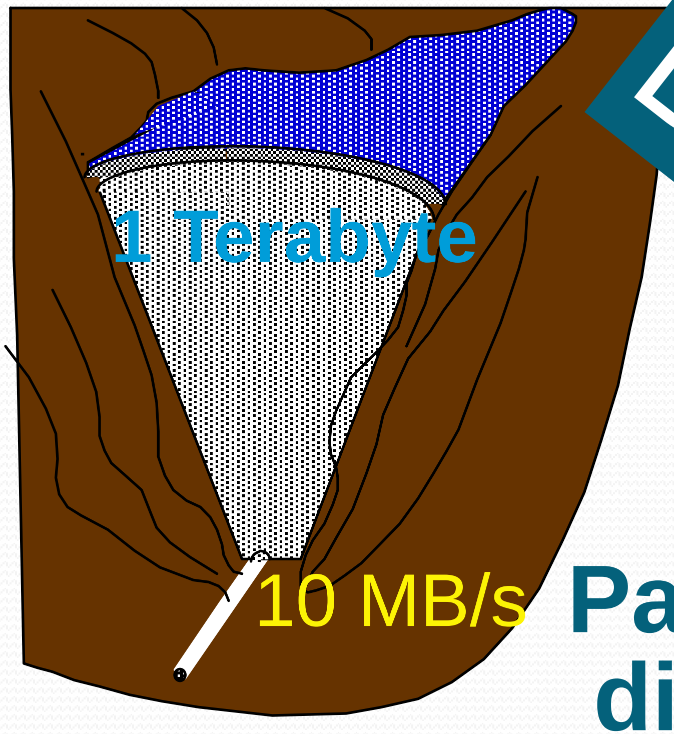
ΠΜΣ "Ερευνητικές Κατευθύνσεις στην Πληροφορική"

Επεξεργασία και Ανάλυση Δεδομένων

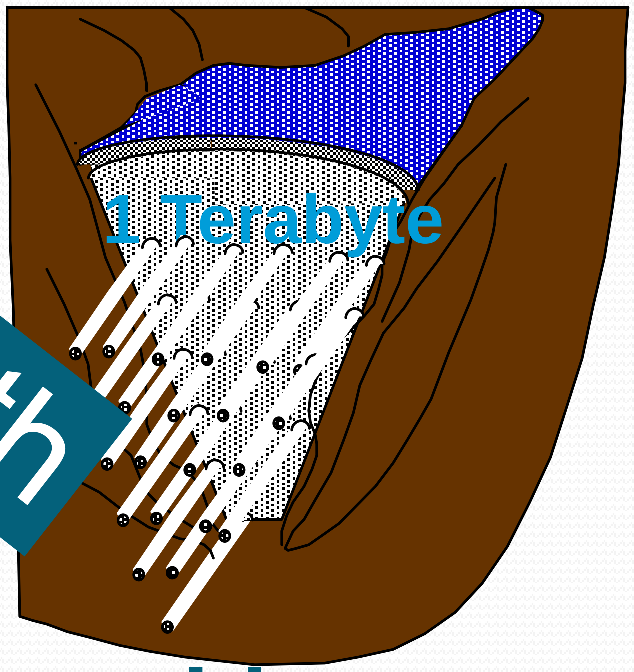
SPRING SEMESTER 2020

Why Parallel Access To Data?

At 10 MB/s
1.2 days to scan



1,000 x parallel
1.5 minute to scan.

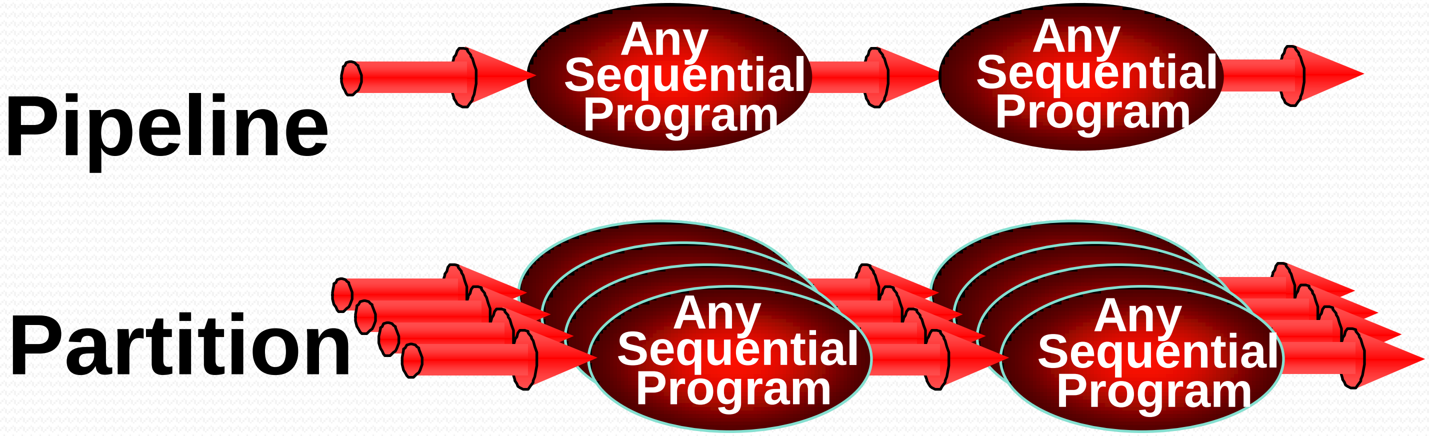


Bandwidth

Parallelism:
divide a big problem
into many smaller ones
to be solved in parallel.

Parallel DBMS: Introduction

- Parallelism is natural to DBMS processing
  - *Pipeline parallelism:* many machines each doing one step in a multi-step process.
  - *Partition parallelism:* many machines doing the same thing to different pieces of data.
  - **Both are natural in DBMS!**



Pipeline

Any Sequential Program

Any Sequential Program

Partition

Any Sequential Program

Any Sequential Program

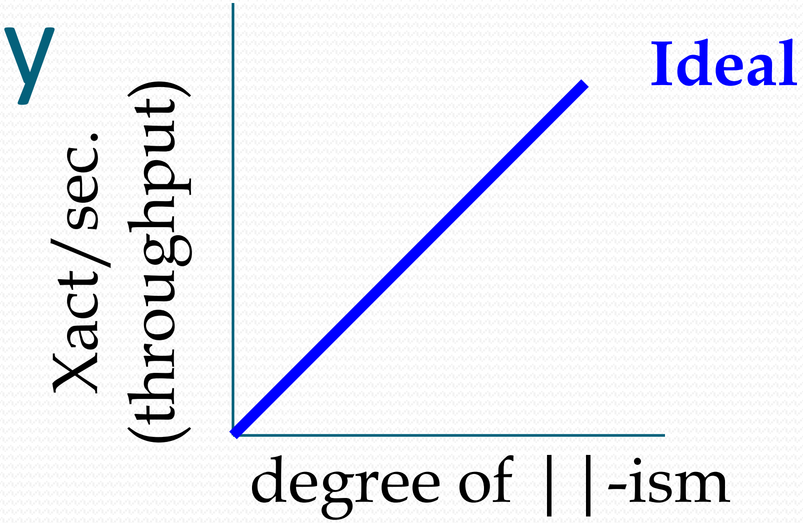
outputs split N ways, inputs merge M ways

DBMS: **The** || Success Story

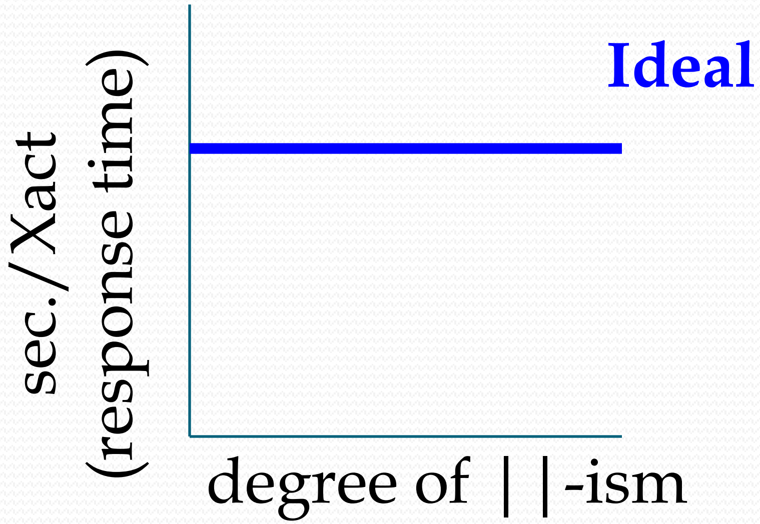
- DBMSs are the most successful application of parallelism.
  - Teradata, Tandem, Thinking Machines
  - Every major DBMS vendor has some || server
- Reasons for success:
  - Bulk-processing (= partition ||-ism).
  - Natural pipelining.
  - Inexpensive hardware can do the trick
  - Users/app-programmers don’t need to think in ||

Some || Terminology

- Speed-Up
  - More resources means proportionally less time for given amount of data.



- Scale-Up
  - If resources increased in proportion to increase in data size, time is constant.

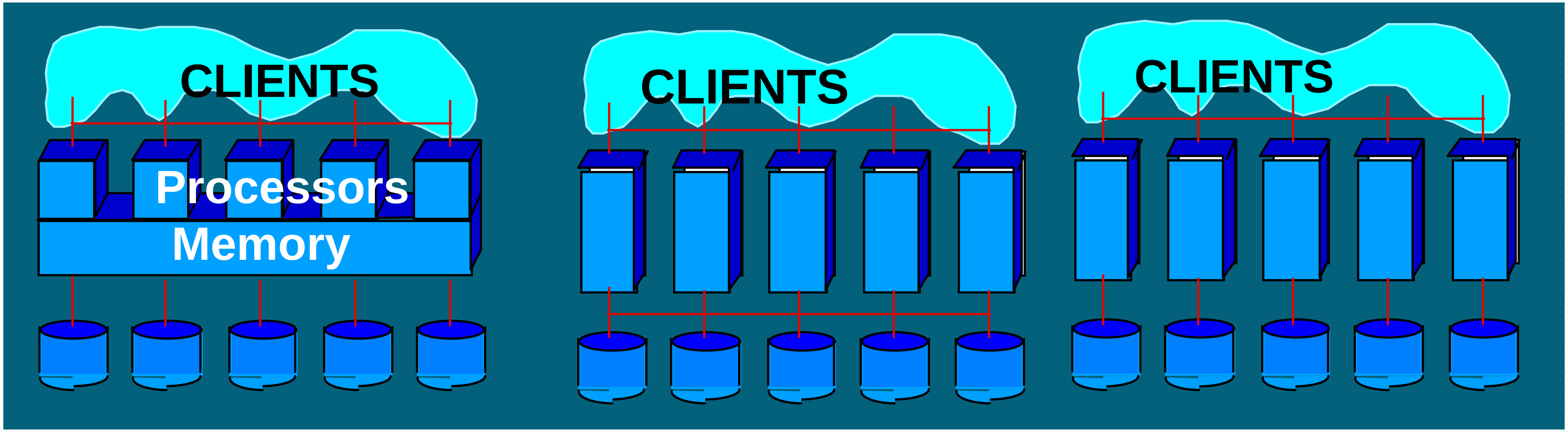


Architecture Issue: Shared What?

Shared Memory (SMP)

Shared Disk

Shared Nothing (network)



Easy to program
Expensive to build
Difficult to scaleup

Sequent, SGI, Sun

VMScluster, Sysplex

Hard to program
Cheap to build
Easy to scaleup

Tandem, Teradata, SP2

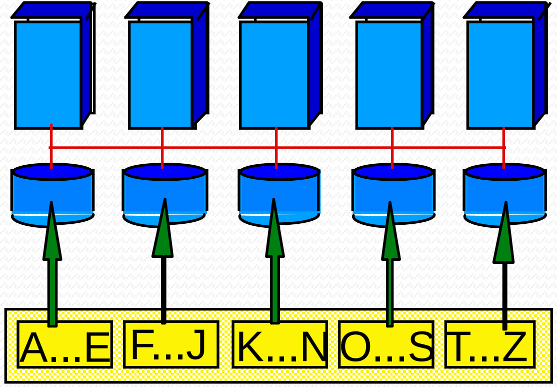
Different Types of DBMS ||-ism

- **Intra-operator parallelism**
  - get all machines working to compute a given operation (scan, sort, join)
- Inter-operator parallelism
  - each operator may run concurrently on a different site (exploits pipelining)
- Inter-query parallelism
  - different queries run on different sites
- We'll focus on intra-operator ||-ism

Automatic Data Partitioning

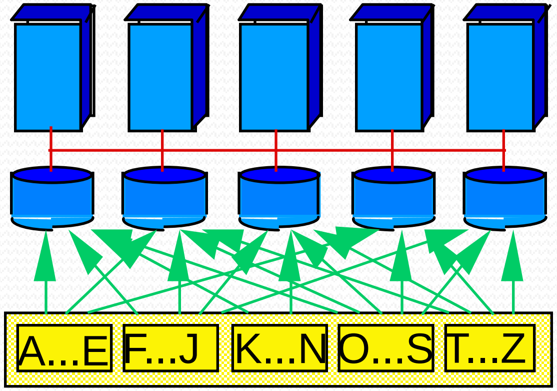
Partitioning a table:

Range



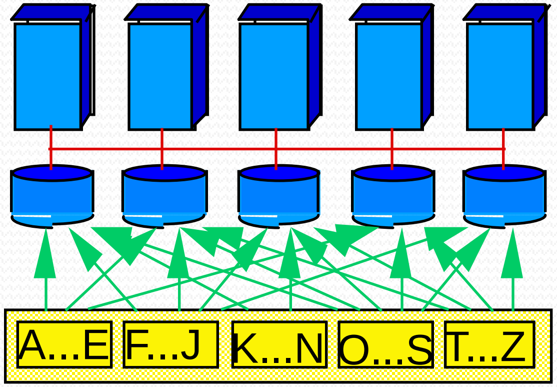
Good for equijoins, range queries group-by

Hash



Good for equijoins

Round Robin



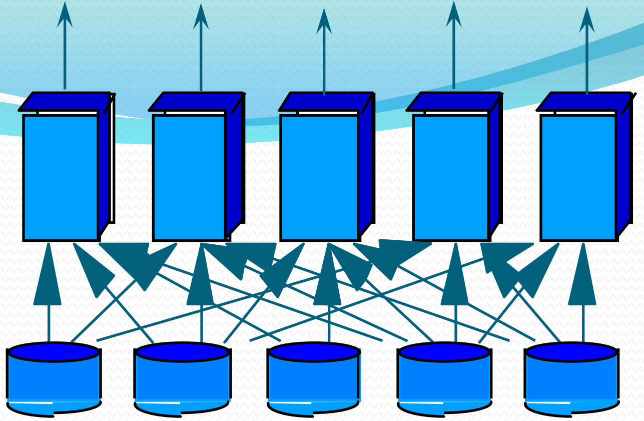
Good to spread load

Shared disk and memory less sensitive to partitioning,
Shared nothing benefits from "good" partitioning

Parallel Scans

- Scan in parallel, and merge.
- Selection may not require all sites for range or hash partitioning.
- Indexes can be built at each partition.

Parallel Sorting



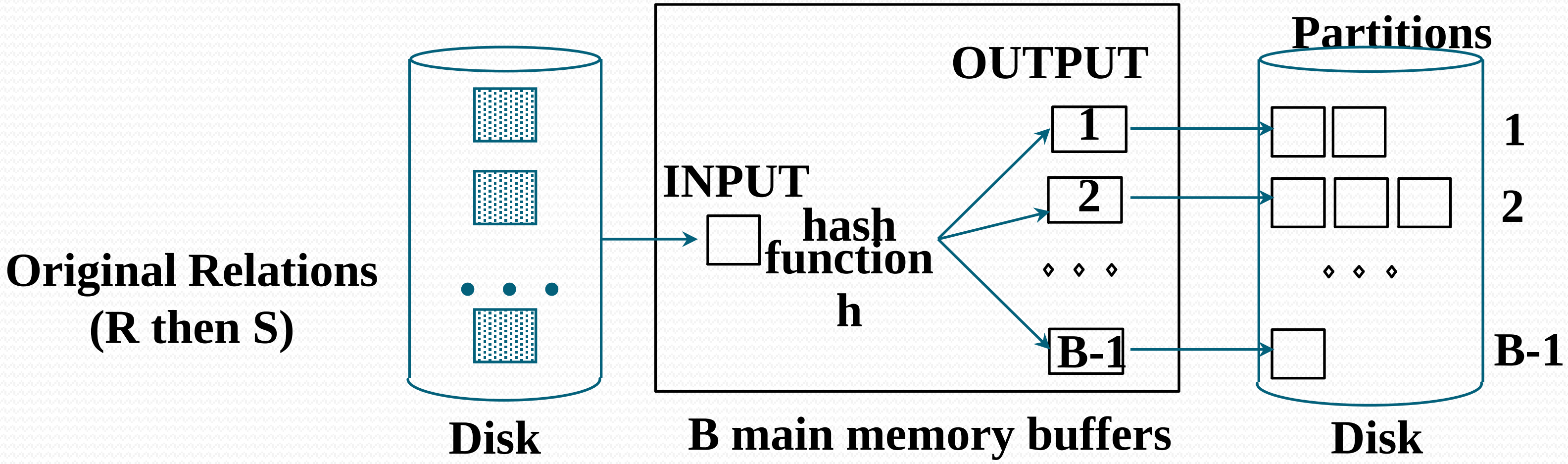
- Idea:
  - Scan in parallel, and range-partition as you go.
  - As tuples come in, begin "local" sorting on each
  - Resulting data is sorted, and range-partitioned.
  - Problem: ***skew!***
  - Solution: "sample" the data at start to determine partition points.

Parallel Joins

- Nested loop:
  - Each outer tuple must be compared with each inner tuple that might join.
  - Easy for range partitioning on join cols, hard otherwise!
- Sort-Merge (or plain Merge-Join):
  - Sorting gives range-partitioning.
  - Merging partitioned tables is local.

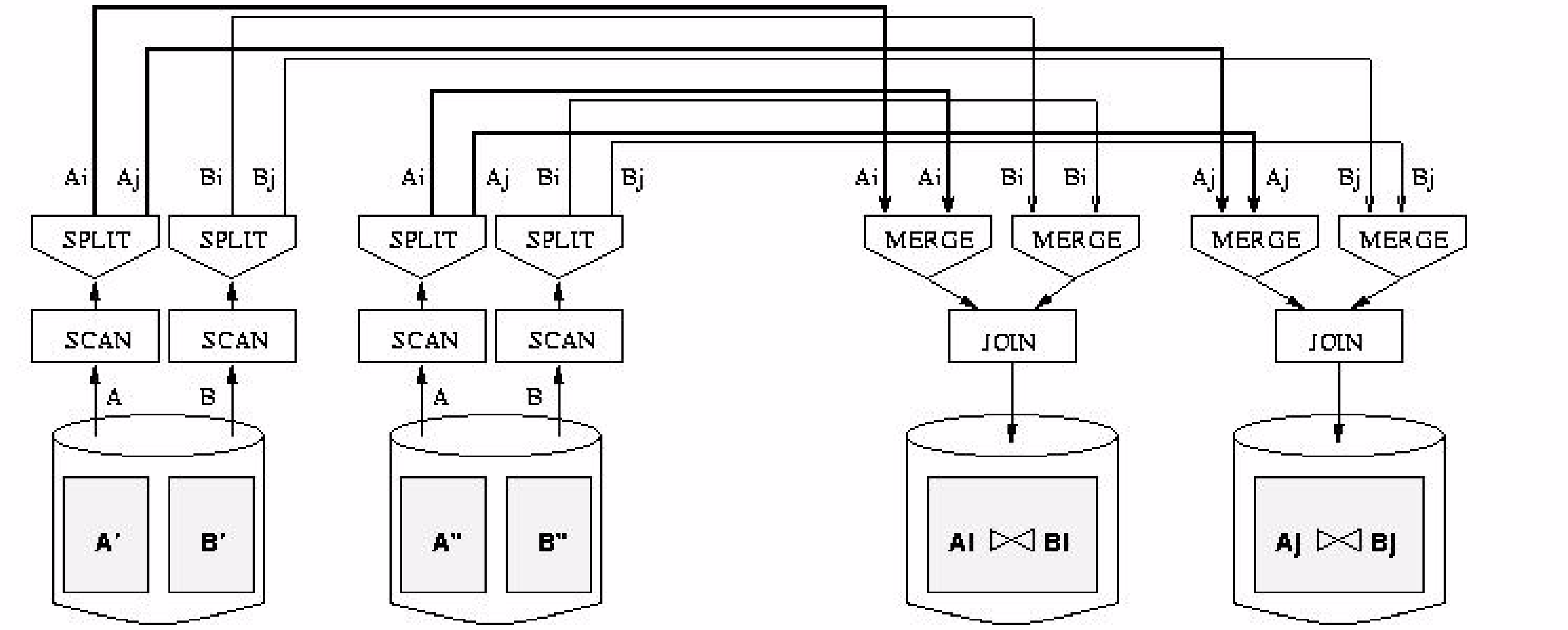
Parallel Hash Join

Phase 1



- In first phase, partitions get distributed to different sites:
  - A good hash function *automatically* distributes work evenly!
- Do second phase at each site.
- Almost always the winner for equi-join.

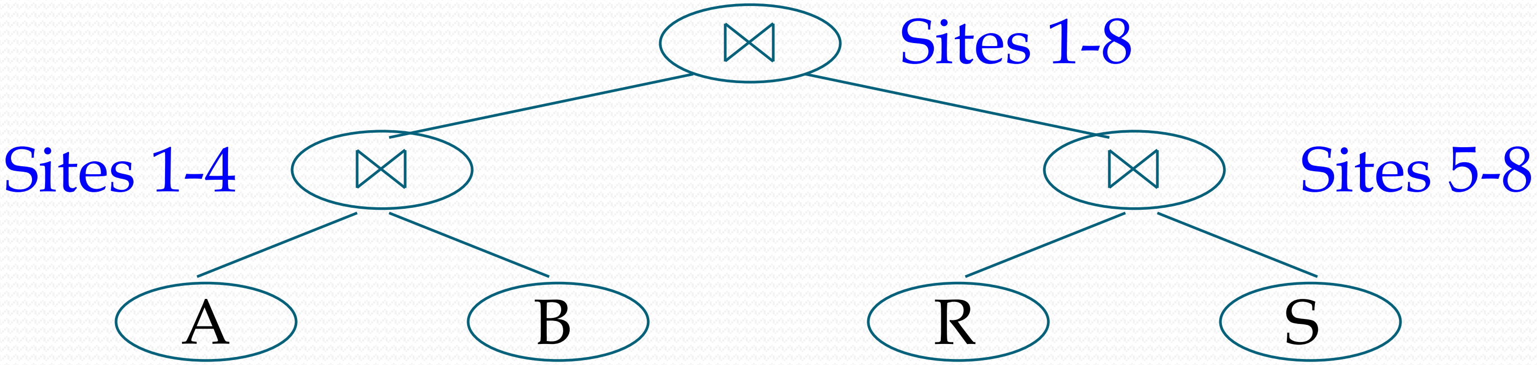
Dataflow Network for || Join (hash join)



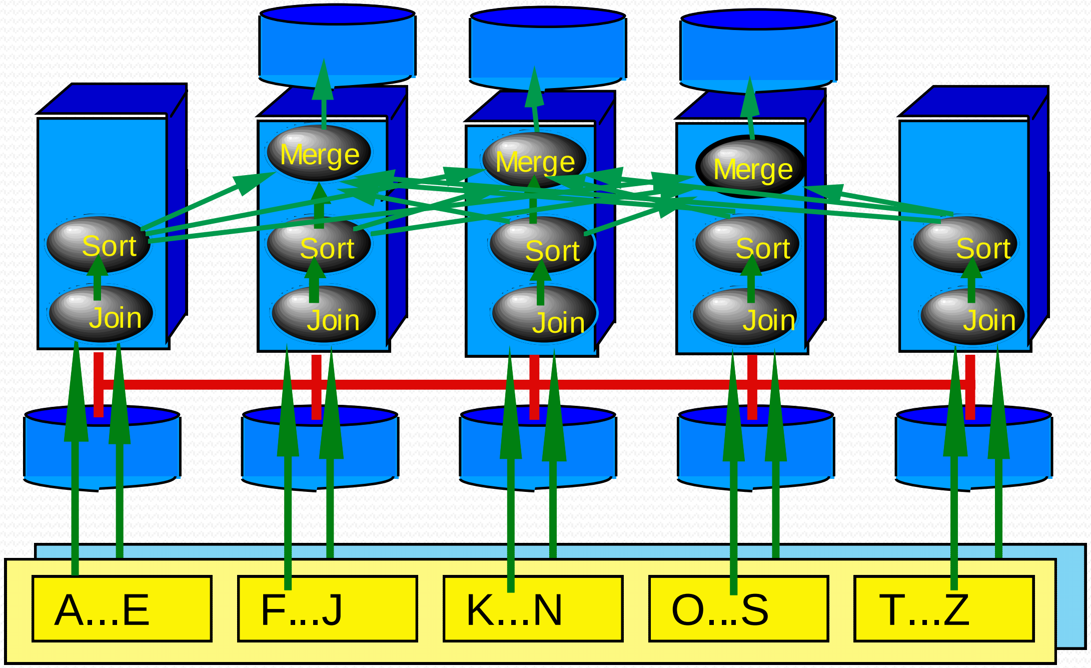
- Good use of split/merge makes it easier to build parallel versions of sequential join code.

Complex Parallel Query Plans

- Complex Queries: Inter-Operator parallelism
  - Pipelining between operators:
    - note that sort and phase 1 of hash-join block the pipeline!!
  - Bushy Trees



N×M-way Parallelism



N inputs, M outputs, no bottlenecks.

Partitioned Data
Partitioned and Pipelined Data Flows

Observations

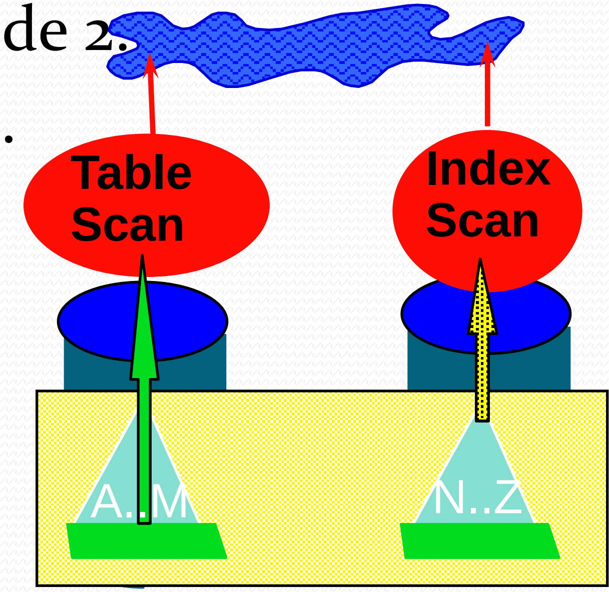
- It is relatively easy to build a fast parallel query executor
- It is hard to write a robust and world-class parallel query optimizer.
  - There are many tricks.
  - One quickly hits the complexity barrier.
  - Still open research!

Parallel Query Optimization

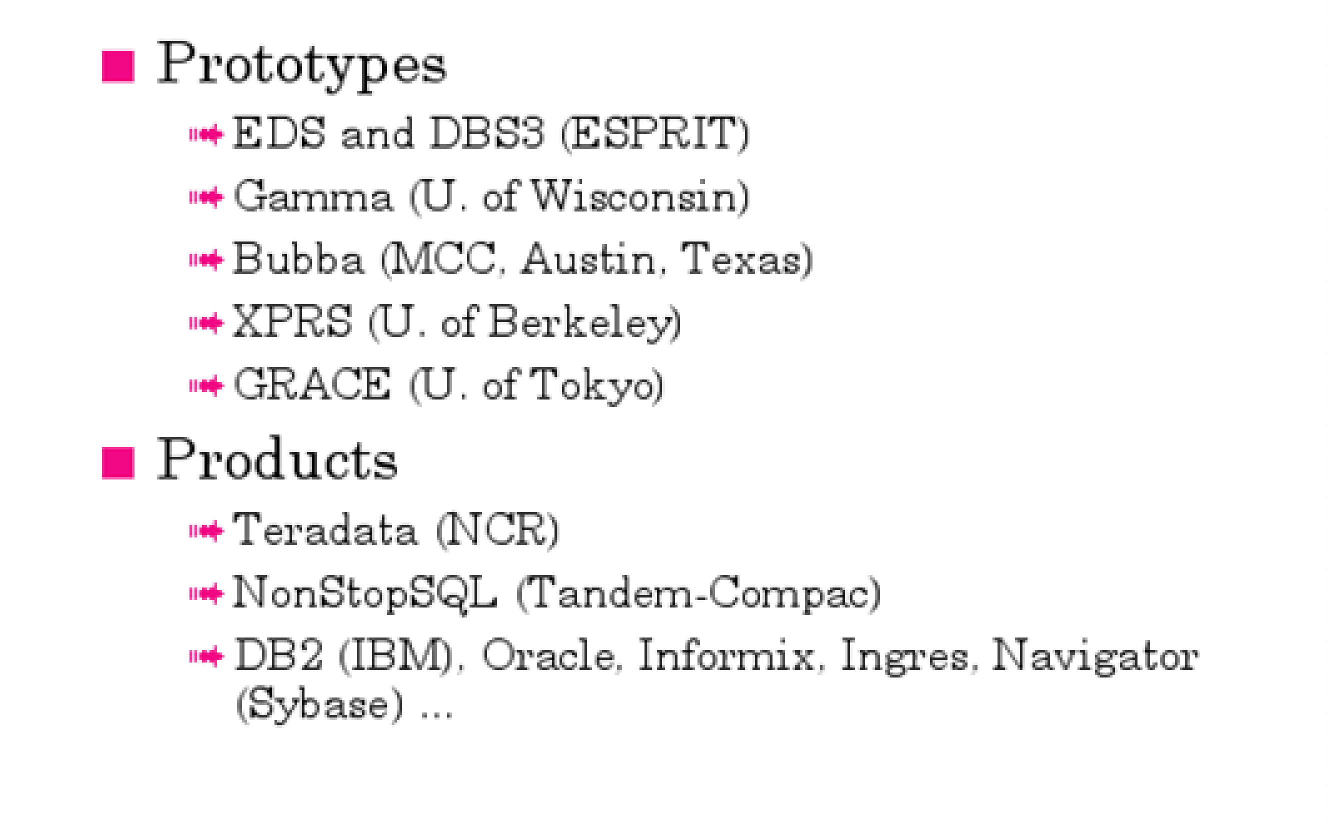
- Common approach: 2 phases
  - Pick best sequential plan (System R algorithm)
  - Pick degree of parallelism based on current system parameters.
- “Bind” operators to processors
  - Use query tree.

What's Wrong With That?

- Best serial plan != Best || plan! Why?
- Trivial counter-example:
  - Table partitioned with local secondary index at two nodes
  - Range query: all of node 1 and 1% of node 2.
  - Node 1 should do a scan of its partition.
  - Node 2 should use secondary index.



Examples of Parallel Databases



- Prototypes
  - EDS and DBS3 (ESPRIT)
  - Gamma (U. of Wisconsin)
  - Bubba (MCC, Austin, Texas)
  - XPRS (U. of Berkeley)
  - GRACE (U. of Tokyo)
- Products
  - Teradata (NCR)
  - NonStopSQL (Tandem-Compac)
  - DB2 (IBM), Oracle, Informix, Ingres, Navigator (Sybase) ...

|| DBMS Summary

- Hardest part of the equation: optimization.
  - 2-phase optimization simplest, but can be ineffective.
  - More complex schemes still at the research stage.
- We haven’t said anything about Xacts, logging.
  - Easy in shared-memory architecture.
  - Takes some care in shared-nothing.

- References :
- Database Management System , 2nd Edition, Raghu Ramakrishnan and Johannes Gehrke
- http://www.research.microsoft.com/research/BARC/Gray/PDB95.ppt

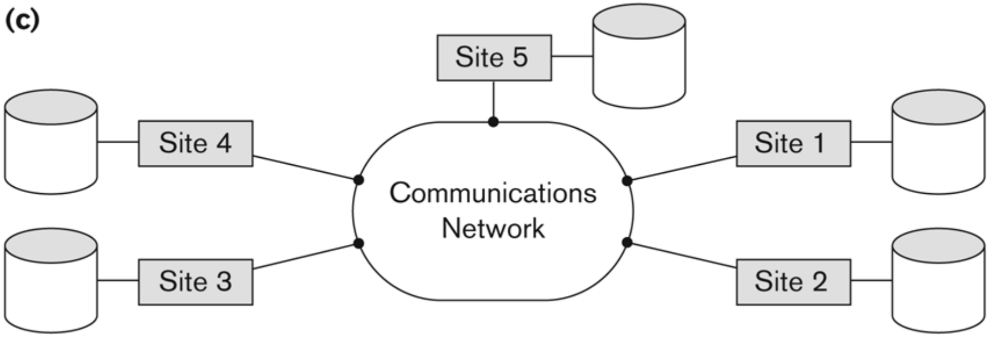
Distributed Database Concepts

- A transaction can be executed by multiple networked computers in a unified manner.
- A distributed database (DDB) can be defined as
  - A **distributed database (DDB)** is a collection of multiple logically related database distributed over a computer network, and a distributed database management system as a software system that manages a distributed database while making the distribution transparent to the user.

Distributed Database System

- Advantages
  - Management of distributed data with different **levels of transparency**:
    - This refers to the physical placement of data (files, relations, etc.) which is not known to the user (distribution transparency).

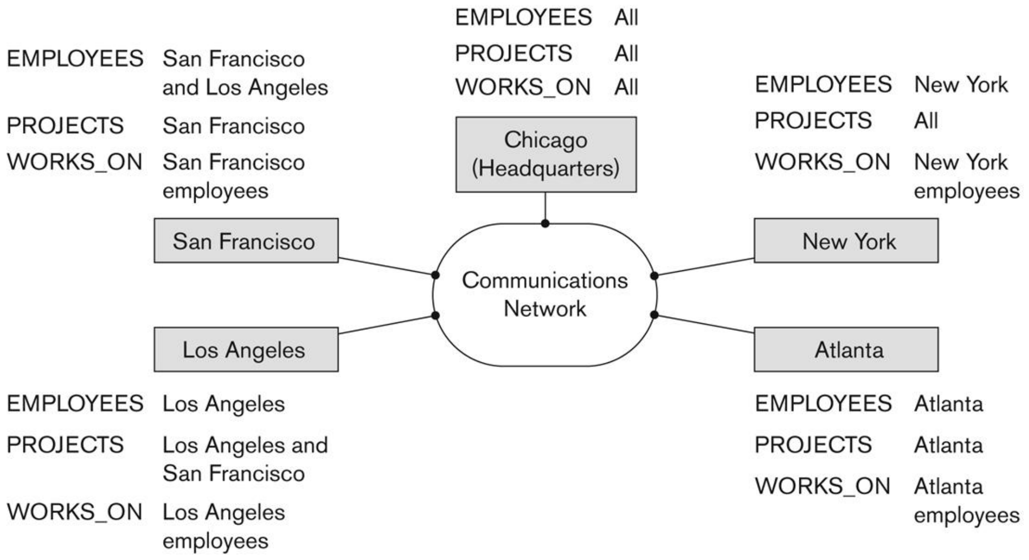
Figure 25.1
Some different database system architectures. (a) Shared nothing architecture. (b) A networked architecture with a centralized database at one of the sites. (c) A truly distributed database architecture.



Distributed Database System

- Advantages (transparency, contd.)
  - The EMPLOYEE, PROJECT, and WORKS_ON tables may be fragmented horizontally and stored with possible replication as shown below.

Figure 25.2
Data distribution and replication among distributed databases.



Distributed Database System

- Advantages (transparency, contd.)
  - **Distribution and Network transparency**:
    - Users do not have to worry about operational details of the network.
      - There is Location transparency, which refers to freedom of issuing command from any location without affecting its working.
      - Then there is Naming transparency, which allows access to any names object (files, relations, etc.) from any location.

Distributed Database System

- Advantages (transparency, contd.)
  - **Replication transparency**:
    - It allows to store copies of a data at multiple sites as shown in the above diagram.
    - This is done to minimize access time to the required data.
  - **Fragmentation transparency**:
    - Allows to fragment a relation horizontally (create a subset of tuples of a relation) or vertically (create a subset of columns of a relation).

Distributed Database System

- Other Advantages
  - **Increased reliability and availability**:
    - Reliability refers to system live time, that is, system is running efficiently most of the time. Availability is the probability that the system is continuously available (usable or accessible) during a time interval.
    - A distributed database system has multiple nodes (computers) and if one fails then others are available to do the job.

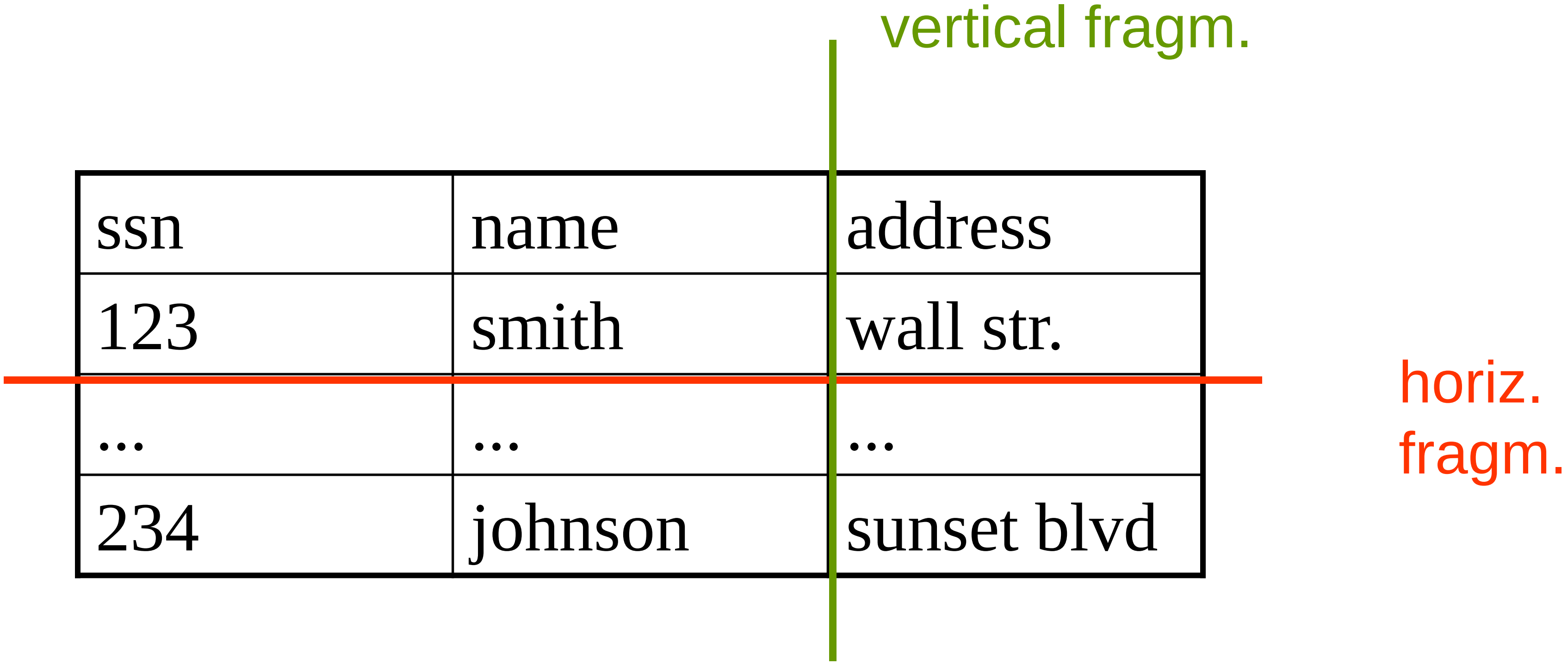
Distributed Database System

- Other Advantages (contd.)
  - **Improved performance**:
    - A distributed DBMS fragments the database to keep data closer to where it is needed most.
    - This reduces data management (access and modification) time significantly.
  - **Easier expansion (scalability)**:
    - Allows new nodes (computers) to be added anytime without chaining the entire configuration.

Data Fragmentation, Replication and Allocation

- **Data Fragmentation**
  - Split a relation into logically related and correct parts. A relation can be fragmented in two ways:
    - **Horizontal Fragmentation**
    - **Vertical Fragmentation**

Design of Distributed DBMSs



ssn	name	address
123	smith	wall str.
...	...	...
234	johnson	sunset blvd

vertical fragm.

horiz.
fragm.

Data Fragmentation, Replication and Allocation

- **Horizontal fragmentation**
  - It is a horizontal subset of a relation which contain those of tuples which satisfy selection conditions.
  - Consider the Employee relation with selection condition (DNO = 5). All tuples that satisfy this condition will create a subset which will be a horizontal fragment of Employee relation.
  - A selection condition may be composed of several conditions connected by AND or OR.
  - Derived horizontal fragmentation: It is the partitioning of a primary relation to other secondary relations which are related with Foreign keys.

Data Fragmentation, Replication and Allocation

- **Vertical fragmentation**
  - It is a subset of a relation which is created by a subset of columns. Thus a vertical fragment of a relation will contain values of selected columns. There is no selection condition used in vertical fragmentation.
  - Consider the Employee relation. A vertical fragment of can be created by keeping the values of Name, Bdate, Sex, and Address.
  - Because there is no condition for creating a vertical fragment, each fragment must include the primary key attribute of the parent relation Employee. This way, all vertical fragments of a relation are connected.

Data Fragmentation, Replication and Allocation

- **Representation**
  - **Mixed (Hybrid) fragmentation**
    - A combination of Vertical fragmentation and Horizontal fragmentation.
    - This is achieved by SELECT-PROJECT operations which is represented by $\Pi_{Li}(\boldsymbol{\sigma}_{Ci}(R))$.
    - If C = True (Select all tuples) and L ≠ ATTRS(R), we get a vertical fragment, and if C ≠ True and L ≠ ATTRS(R), we get a mixed fragment.
    - If C = True and L = ATTRS(R), then R can be considered a fragment.

Data Fragmentation, Replication and Allocation

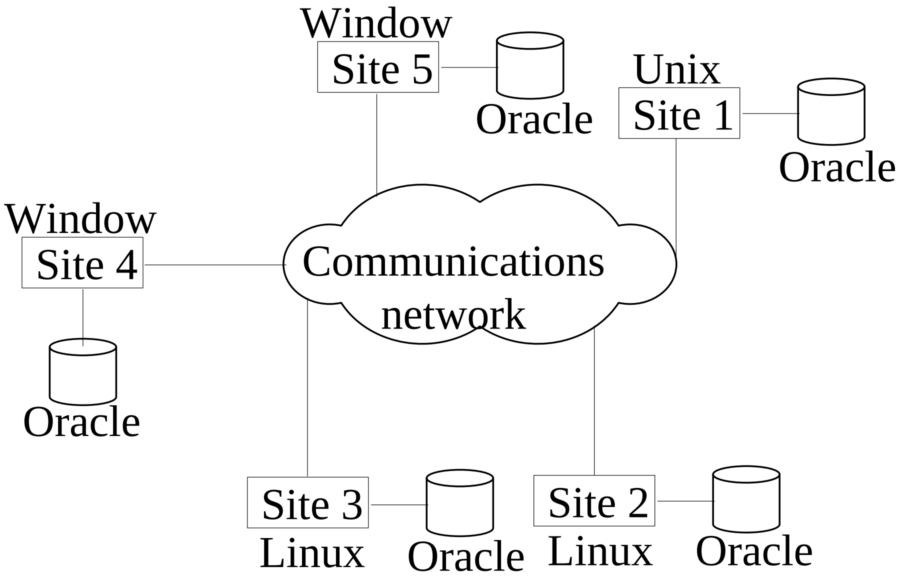
- **Fragmentation schema**
  - A definition of a set of fragments (horizontal or vertical or horizontal and vertical) that includes all attributes and tuples in the database that satisfies the condition that the whole database can be reconstructed from the fragments by applying some sequence of UNION (or OUTER JOIN and UNION operations).
- **Allocation schema**
  - It describes the distribution of fragments to sites of distributed databases. It can be fully or partially replicated or can be partitioned.

Data Fragmentation, Replication and Allocation

- **Data Replication**
  - Database is replicated to all sites.
  - In full replication, the entire database is replicated and in partial replication some selected part is replicated to some of the sites.
  - Data replication is achieved through a replication schema.
- **Data Distribution (Data Allocation)**
  - This is relevant only in the case of partial replication or partition.
  - The selected portion of the database is distributed to the database sites.

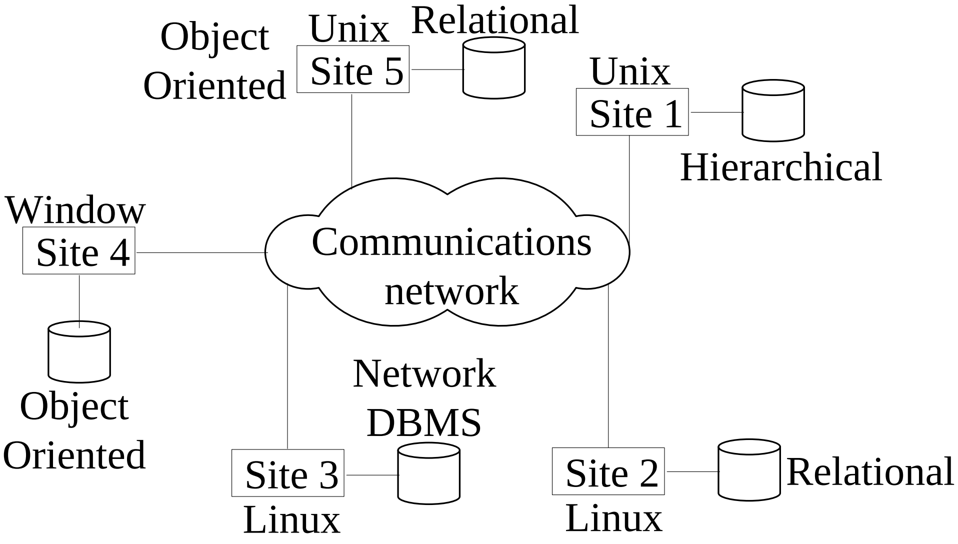
Types of Distributed Database Systems

- Homogeneous
  - All sites of the database system have identical setup, i.e., same database system software.
  - The underlying operating system may be different.
    - For example, all sites run Oracle or DB2, or Sybase or some other database system.
  - The underlying operating systems can be a mixture of Linux, Window, Unix, etc.



Types of Distributed Database Systems

- Heterogeneous
  - Federated: Each site may run a different database system but the data access is managed through a single conceptual schema.
    - This implies that the degree of local autonomy is minimum. Each site must adhere to a centralized access policy. There may be a global schema.
  - Multi-database: There is no global schema. For data access, a schema is constructed dynamically as needed by the application software.



Types of Distributed Database Systems

- Federated Database Management Systems Issues
  - Differences in data models:
    - Relational, Objected oriented, hierarchical, network, etc.
  - Differences in constraints:
    - Each site may have their own data accessing and processing constraints.
  - Differences in query language:
    - Some site may use SQL, some may use SQL-89, some may use SQL-92, and so on.

Query Processing in Distributed Databases

- Issues
  - Cost of transferring data (files and results) over the network.
    - This cost is usually high, so some optimization is necessary.
    - Example relations: Employee at site 1 and Department at Site 2
      - Employee at site 1. 10,000 rows. Row size = 100 bytes. Table size = 10^6 bytes.

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Superssn	Dno

      - Department at Site 2. 100 rows. Row size = 35 bytes. Table size = 3,500 bytes.

Dname	Dnumber	Mgrssn	Mgrstartdate

    - Q: For each employee, retrieve employee name and department name Where the employee works.
    - Q: $\Pi_{\text{Fname,Lname,Dname}}$ (Employee $\bowtie_{\text{Dno = Dnumber}}$ Department)

Query Processing in Distributed Databases

- Result
  - The result of this query will have 10,000 tuples, assuming that every employee is related to a department.
  - Suppose each result tuple is 40 bytes long. The query is submitted at site 3 and the result is sent to this site.
  - Problem: Employee and Department relations are not present at site 3.

Query Processing in Distributed Databases

- Strategies:
  1. Transfer Employee and Department to site 3.
     - Total transfer bytes = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3.
     - Query result size = 40 * 10,000 = 400,000 bytes. Total transfer size = 400,000 + 1,000,000 = 1,400,000 bytes.
  3. Transfer Department relation to site 1, execute the join at site 1, and send the result to site 3.
     - Total bytes transferred = 400,000 + 3500 = 403,500 bytes.
- Optimization criteria: minimizing data transfer.

Query Processing in Distributed Databases

- Consider the query
  - Q’: For each department, retrieve the department name and the name of the department manager
- Relational Algebra expression:
  - $\Pi_{Fname,Lname,Dname}$ (Employee $\bowtie_{Mgrssn = SSN}$ Department)

Query Processing in Distributed Databases

- The result of this query will have 100 tuples, assuming that every department has a manager, the execution strategies are:
  1. Transfer Employee and Department to the result site and perform the join at site 3.
     - Total bytes transferred = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3. Query result size = 40 * 100 = 4000 bytes.
     - Total transfer size = 4000 + 1,000,000 = 1,004,000 bytes.
  3. Transfer Department relation to site 1, execute join at site 1 and send the result to site 3.
     - Total transfer size = 4000 + 3500 = 7500 bytes.

Query Processing in Distributed Databases

- Now suppose the result site is 2. Possible strategies :
  1. Transfer Employee relation to site 2, execute the query and present the result to the user at site 2.
     - Total transfer size = 1,000,000 bytes for both queries Q and Q'.
  2. Transfer Department relation to site 1, execute join at site 1 and send the result back to site 2.
     - Total transfer size for Q = 400,000 + 3500 = 403,500 bytes and for Q' = 4000 + 3500 = 7500 bytes.

Semi Joins

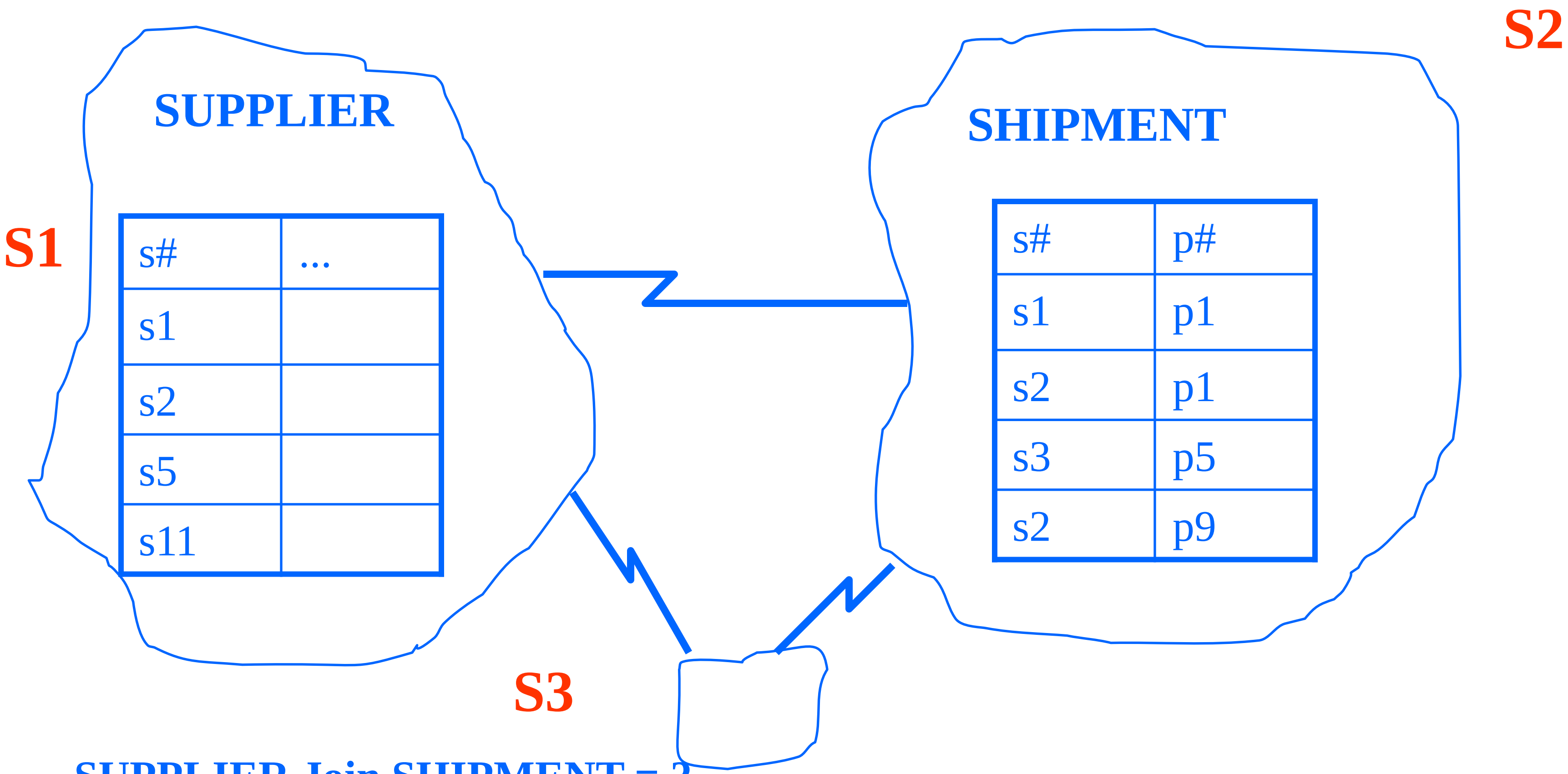
- A join where the result only contains columns from one of the joined tables
- Very useful in distributed databases, so that we don’t transmit a lot of data over the network
- Can dramatically speed up certain classes of queries

Query Processing in Distributed Databases

- Semijoin:
  - Objective is to reduce the number of tuples in a relation before transferring it to another site.
- Example execution of Q or Q':

1. Project the join attributes of Department at site 2, and transfer them to site 1. For Q, 4 * 100 = 400 bytes are transferred and for Q', 9 * 100 = 900 bytes are transferred.
2. Join the transferred file with the Employee relation at site 1, and transfer the required attributes from the resulting file to site 2. For Q, 34 * 10,000 = 340,000 bytes are transferred and for Q', 39 * 100 = 3900 bytes are transferred.
3. Execute the query by joining the transferred file with Department and present the result to the user at site 2.

Distr. Q-opt – semijoins



s#	...
s1	
s2	
s5	
s11	

s#	p#
s1	p1
s2	p1
s3	p5
s2	p9

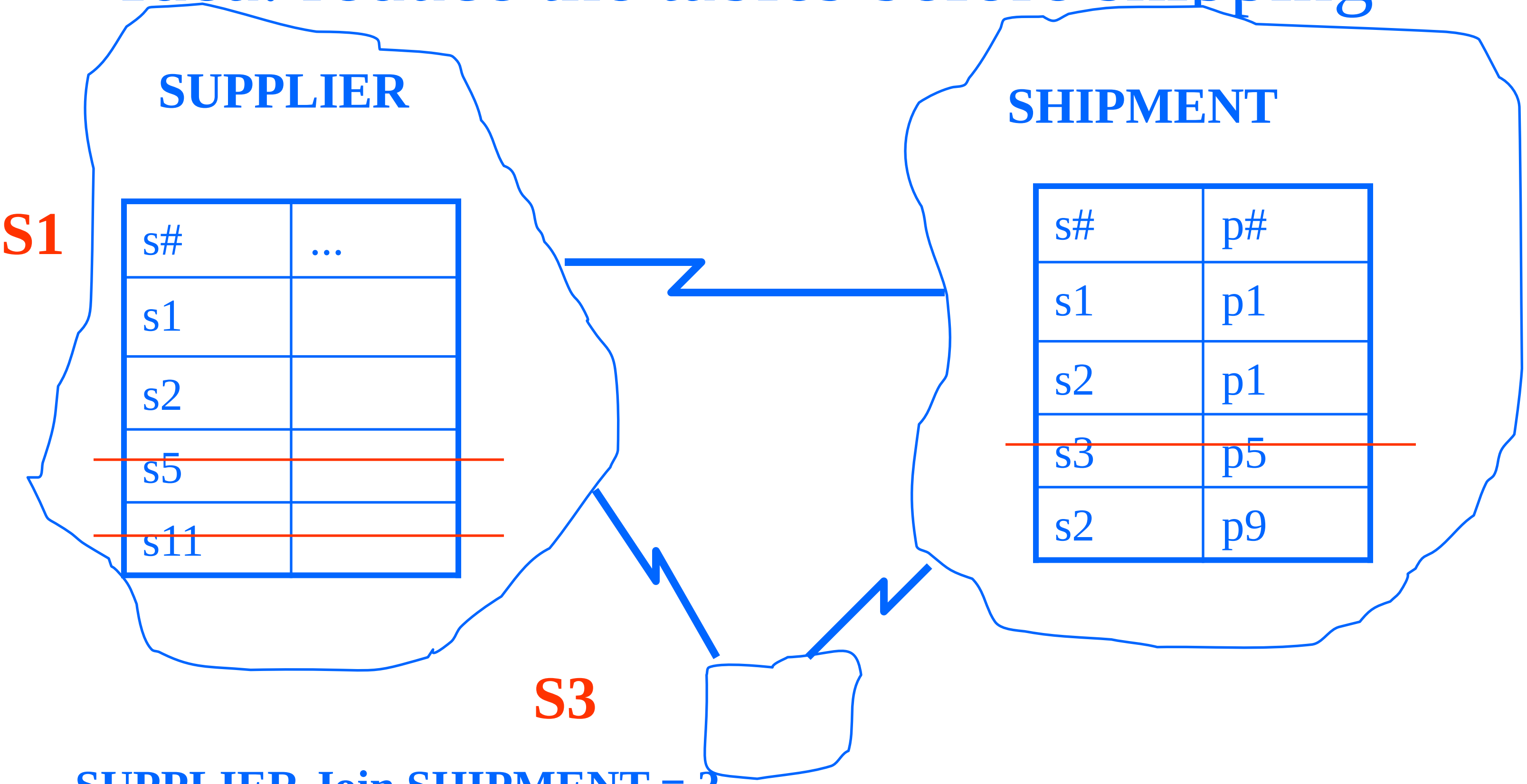
SUPPLIER Join SHIPMENT = ?

semijoins

- choice of plans?
- plan #1: ship SUP -> S2; join; ship -> S3
- plan #2: ship SHIP->S3; ship SUP->S3; join
- ...
- others?

Semijoins

- Idea: reduce the tables before shipping



S1

SUPPLIER

s#	...
s1	
s2	
~~s5~~	~~ ~~
~~s11~~	~~ ~~

SHIPMENT

s#	p#
s1	p1
s2	p1
~~s3~~	~~p5~~
s2	p9

S3

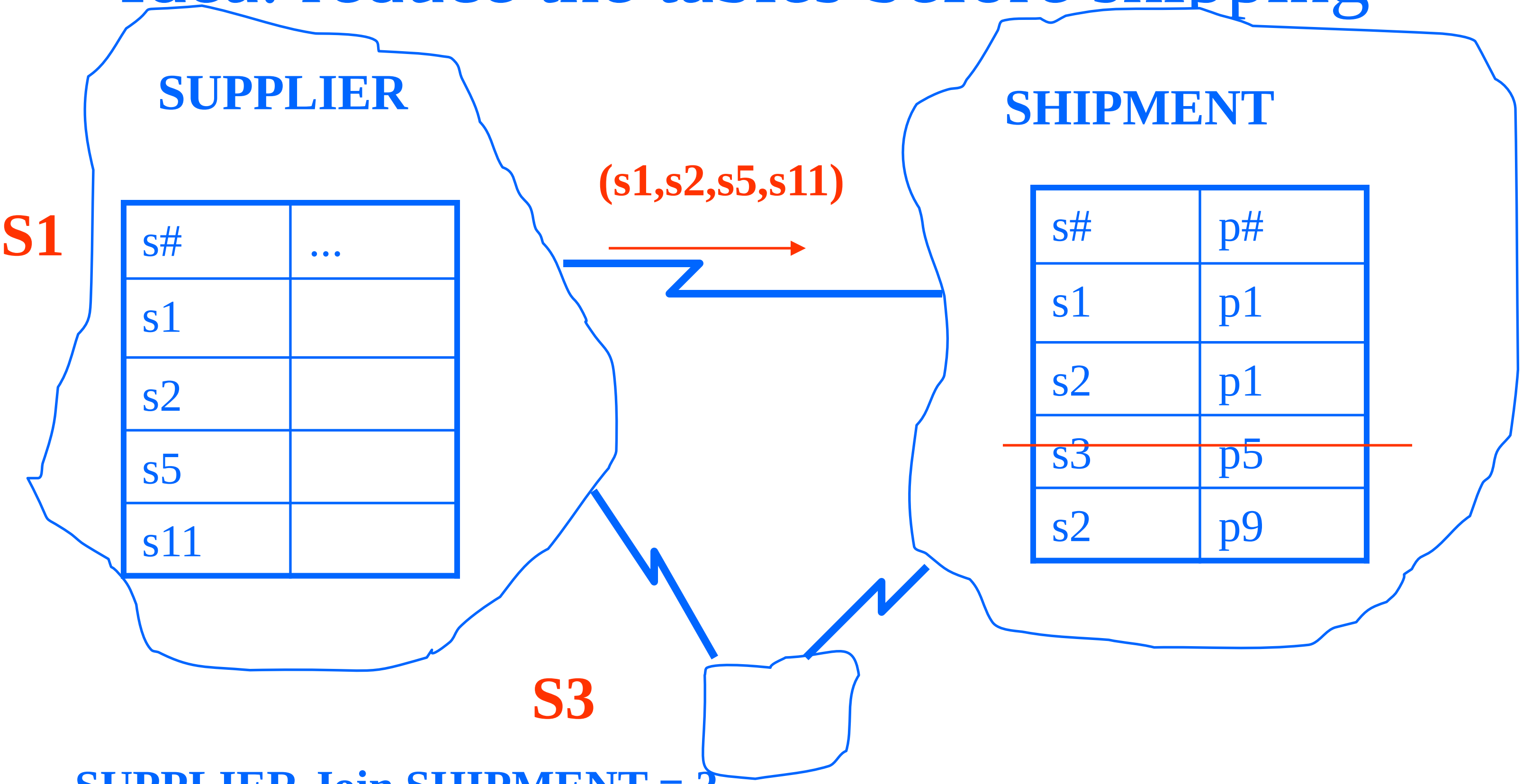
SUPPLIER Join SHIPMENT = ?

Semijoins

- How to do the reduction, cheaply?
- Eg., reduce ‘SHIPMENT’:

Semijoins

- Idea: reduce the tables before shipping



SUPPLIER

s#	…
s1	
s2	
s5	
s11	

SHIPMENT

s#	p#
s1	p1
s2	p1
~~s3~~	~~p5~~
s2	p9

SUPPLIER Join SHIPMENT = ?

Semijoins

- Formally:
- SHIPMENT’ = SHIPMENT $\triangleright\!<$ SUPPLIER
- express semijoin w/ rel. algebra

$$R' = R \triangleright\!< S$$

$$= \pi_R(R \triangleright\!\triangleleft S)$$

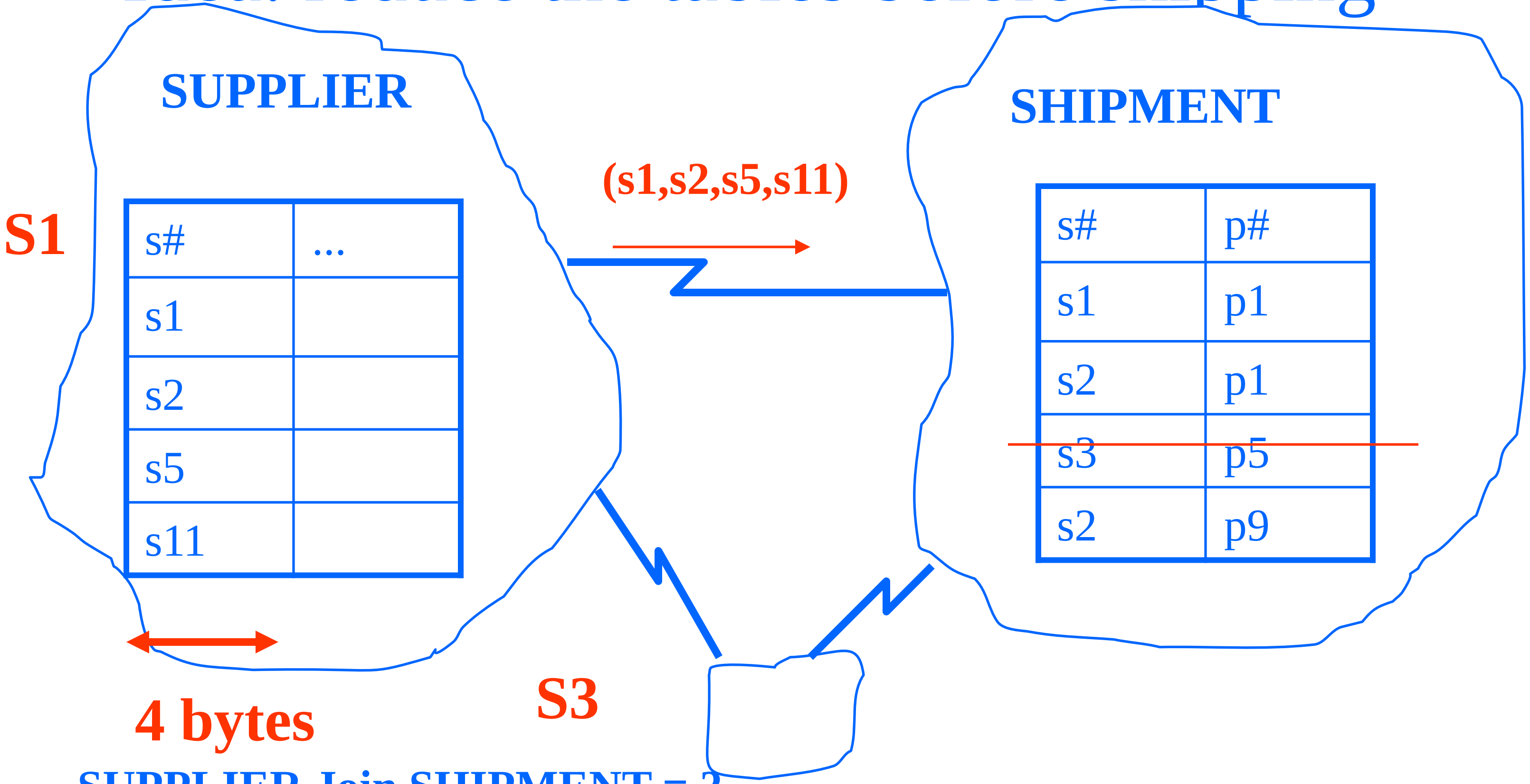
Semijoins – eg:

- suppose each attr. is 4 bytes
- Q: transmission cost (#bytes) for semijoin

  SHIPMENT’ = SHIPMENT semijoin SUPPLIER

Semijoins

- Idea: reduce the tables before shipping



S1

SUPPLIER

s#	...
s1	
s2	
s5	
s11	

4 bytes

(s1,s2,s5,s11)

SHIPMENT

s#	p#
s1	p1
s2	p1
~~s3~~	~~p5~~
s2	p9

S3

SUPPLIER Join SHIPMENT = ?

Semijoins – eg:

- suppose each attr. is 4 bytes
- Q: transmission cost (#bytes) for semijoin

  SHIPMENT’ = SHIPMENT semijoin SUPPLIER
- A: 4*4 bytes

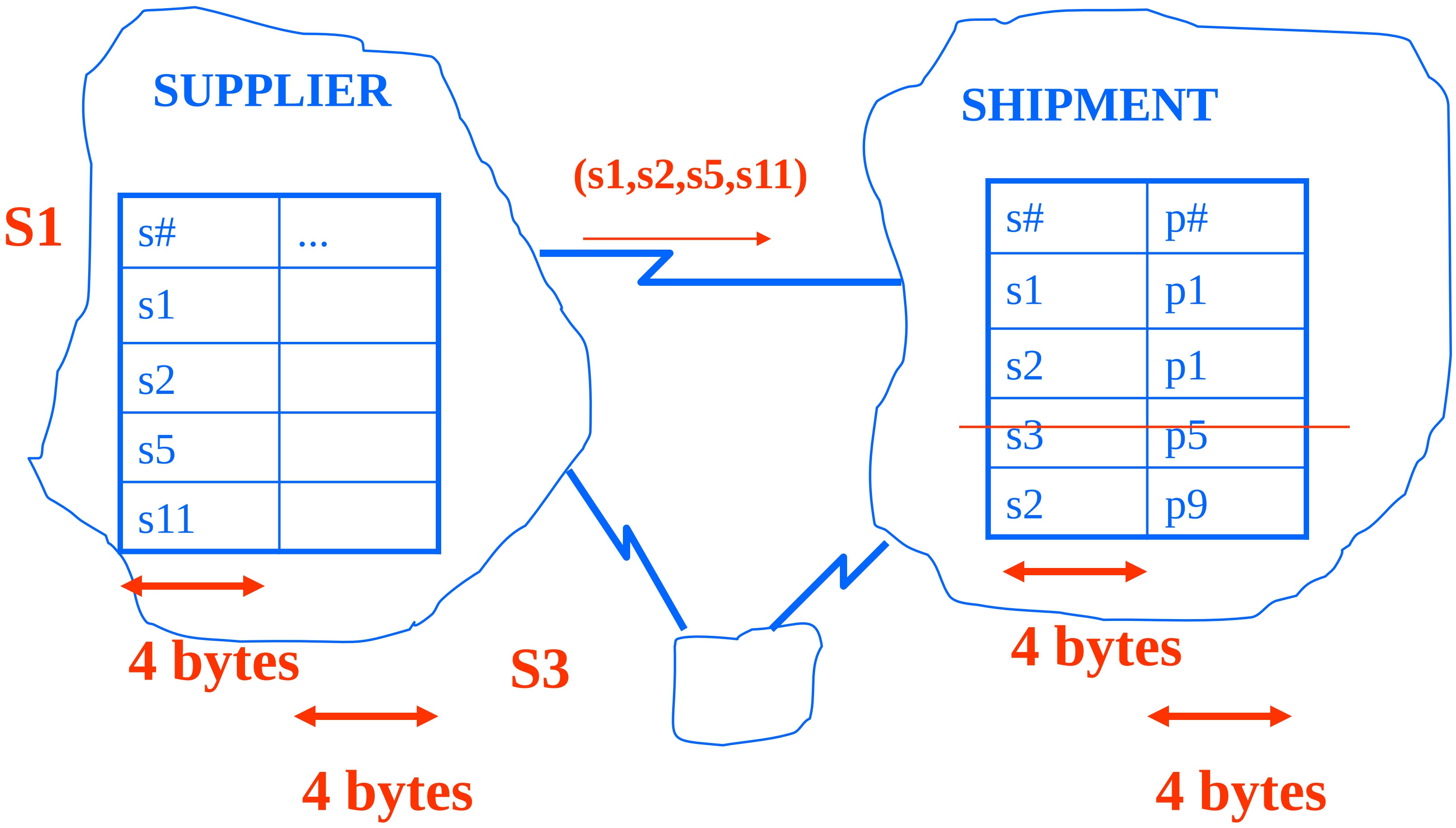
Semijoins – eg:

- suppose each attr. is 4 bytes
- Q1: give a plan, with semijoin(s)
- Q2: estimate its cost (#bytes shipped)

Semijoins – eg:

- Q1:
  - reduce SHIPMENT to SHIPMENT’
  - SHIPMENT’ -> S3
  - SUPPLIER -> S3
  - do join @ S3
- Q2: cost?

Semijoins



S1

SUPPLIER

s#	...
s1	
s2	
s5	
s11	

4 bytes

4 bytes

(s1,s2,s5,s11)

S3

SHIPMENT

s#	p#
s1	p1
s2	p1
~~s3~~	~~p5~~
s2	p9

4 bytes

4 bytes

Semijoins – eg:

- A2:
  - 4*4 bytes - reduce SHIPMENT to SHIPMENT’
  - 3*8 bytes - SHIPMENT’ -> S3
  - 4*8 bytes - SUPPLIER -> S3
  - 0 bytes - do join @ S3

---

72 bytes TOTAL

Other plans?

P2:

- reduce SHIPMENT to SHIPMENT’
- reduce SUPPLIER to SUPPLIER’
- SHIPMENT’ -> S3
- SUPPLIER’ -> S3

Other plans?

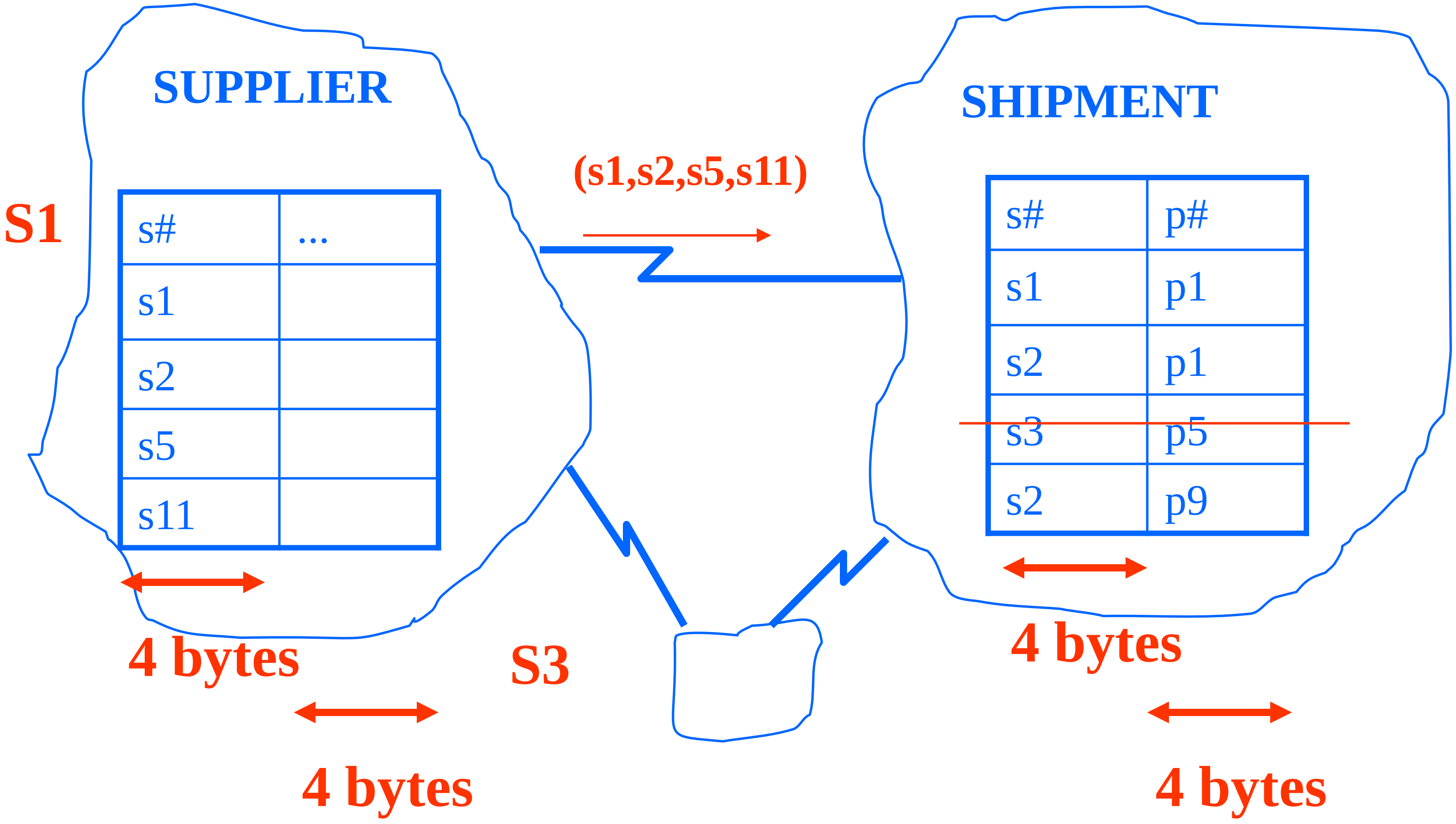
P3:

- reduce SUPPLIER to SUPPLIER’
- SUPPLIER’ -> S2
- do join @ S2
- ship results -> S3

A brilliant idea: ‘Bloom-joins’

- how to ship the projection, say, of SUPPLIER.s#, even cheaper?
- A: Bloom-filter [Lohman+] =
  - quick&dirty membership testing

Semijoins



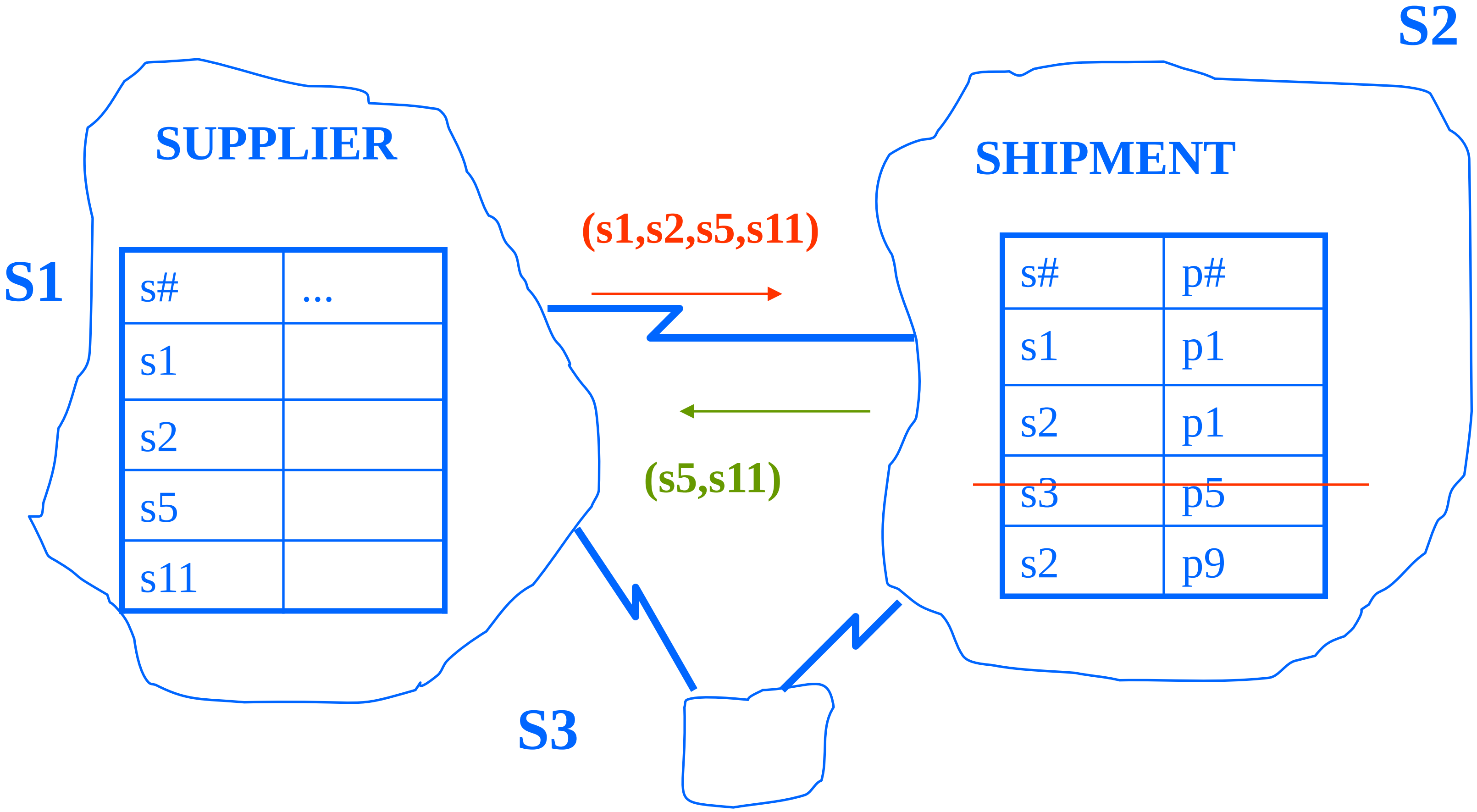
s#	...
s1	
s2	
s5	
s11	

s#	p#
s1	p1
s2	p1
~~s3~~	~~p5~~
s2	p9

Another brilliant idea: two-way semijoins

- reduce both relations with one more exchange: [Kang, ’86]
- ship back the list of keys that didn’t match
- CAN NOT LOSE! (why?)
- further improvement:
  - or the list of ones that matched – whatever is shorter!

Two-way Semijoins



SUPPLIER

s#	…
s1	
s2	
s5	
s11	

SHIPMENT

s#	p#
s1	p1
s2	p1
~~s3~~	~~p5~~
s2	p9