DISTRIBUTED AND PARALLELL DATABASE SYSTEMS

Tore Risch

Uppsala Database Laboratory

Department of Information Technology

Uppsala University

Sweden

http://user.it.uu.se/~torer

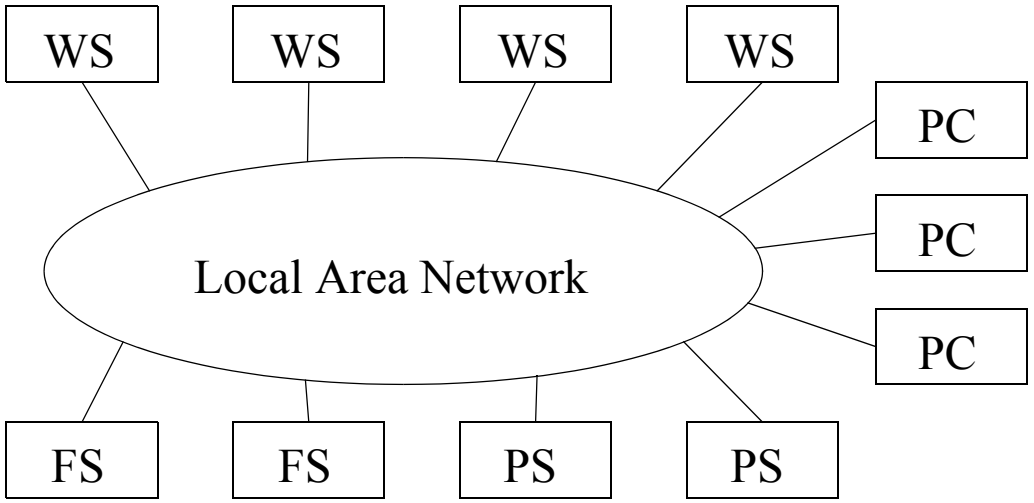
Background

What is a Distributed System?

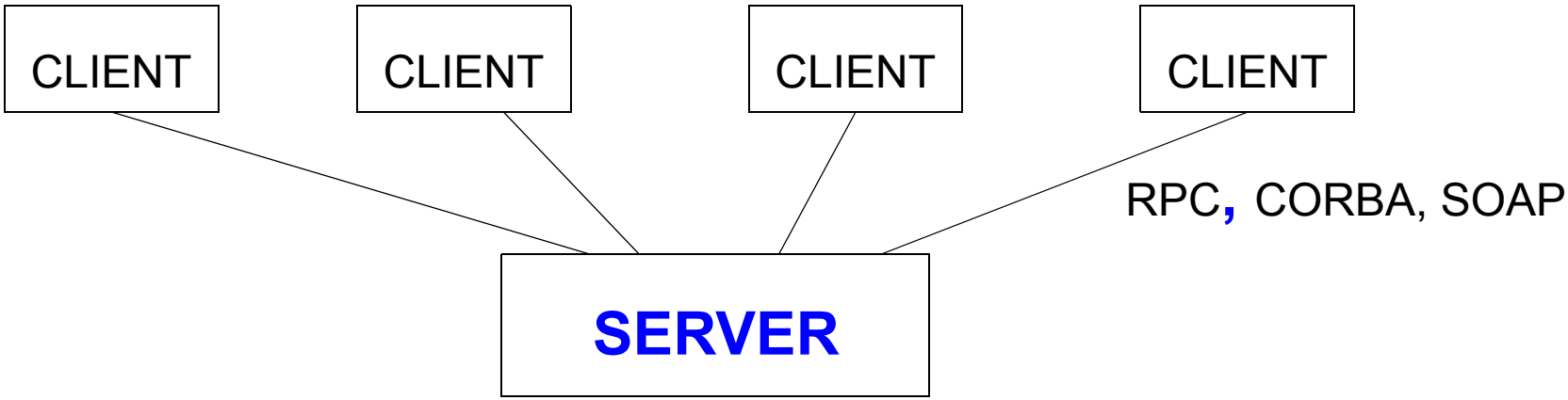
A Distributed System is a number of autonomous computers communicating over a network with software for integrated tasks.

Examples of Distributed Systems:

- SUN's Network File System (NFS), distributed file system

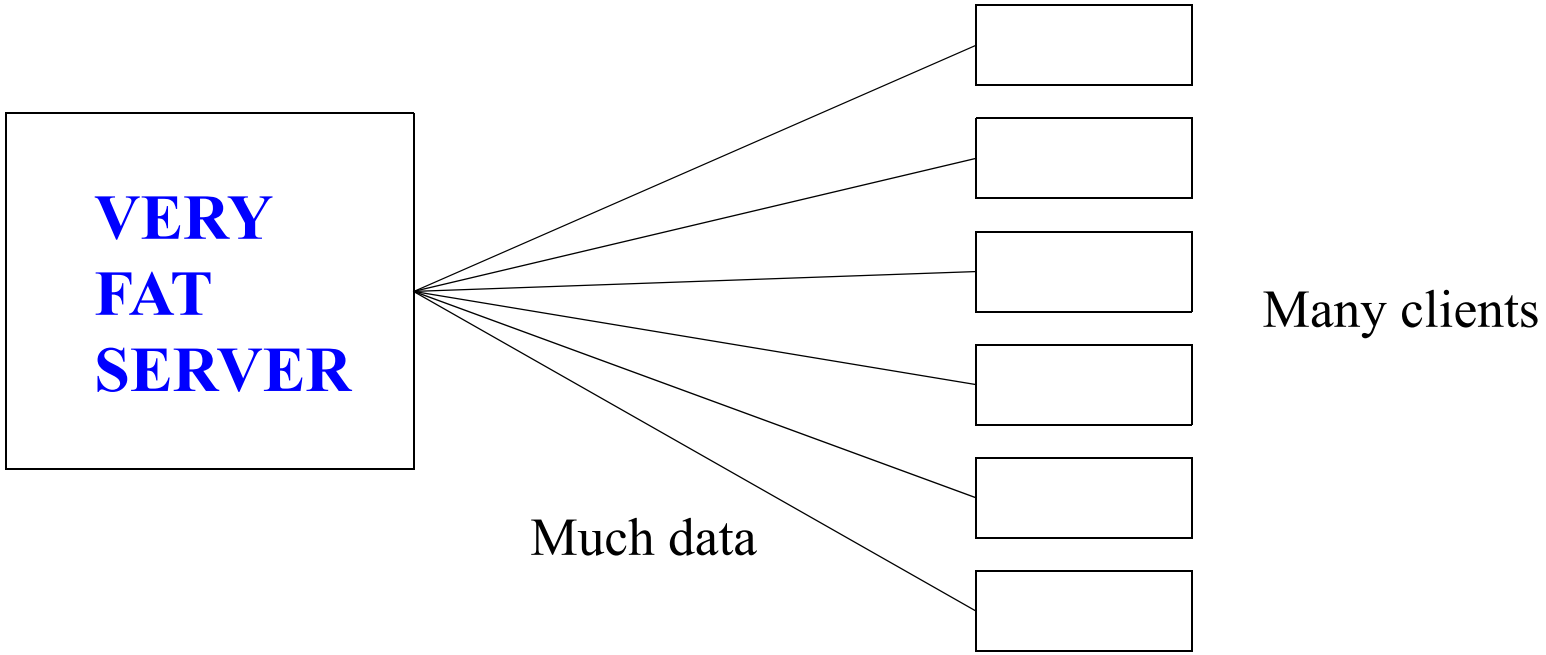


Client-Server Architecture



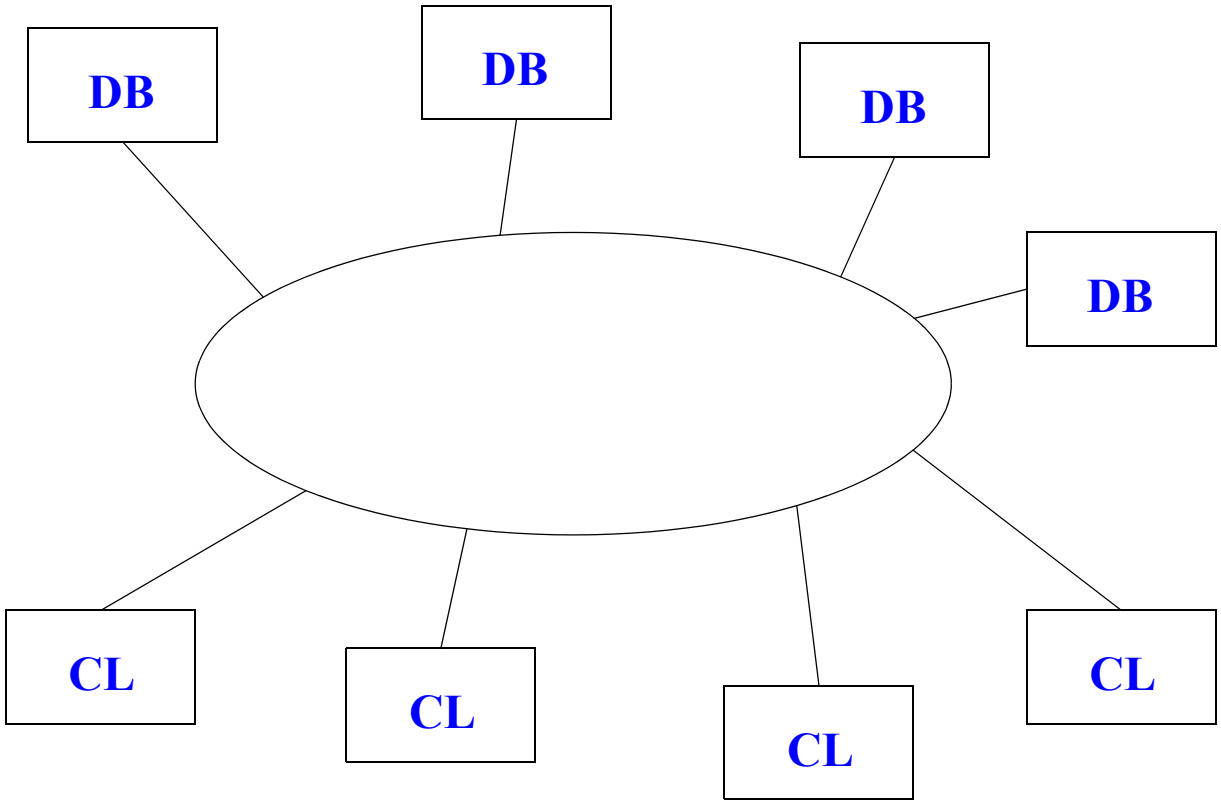
- Each client sends request (RPCs) to server
- Server waits for requests from clients
- FAT Server:
  Databases, File servers, Heavy computations
- Thin Client:
  Graphics, User interactions
- FAT Client:
  CAD System, Numerical computations

Database Communication



- Stream based client-server interfaces
- DBMS specific interfaces
- Compiler integrated interfaces (embedded SQL)
- ODBC: SQL-based standardized subroutine call library (MicroSoft)
- JDBC: ODBC for Java (not MicroSoft)

Distributed Databases



- Database *transparently* seen from application ONE database, usual SQL interface.
- *Manual* partitioning or *fragmentation* and *repliction* of data tables:
  Distributed database design
- DDBMS *automatically* optimizes queries and updates to distributed database.

What is a Distributed Database?

"A distributed database (DDB) is a collection of multiple, *logically interrelated* databases *distributed over a computer network*. A DDBMS is the software system that permits the management of DDBs and makes the distribution transparent to the user."

- A DDB is NOT:
  - A collection of files (need structure and DB manager)
  - A client-server interface to a database

    Data on one node, clients on other nodes in network

    (Almost) every centralized DBMS has client-server interface

Example of DDB

- Multi-national company with distributed departments
- Distributed data management:
  - Each location keep local records of local employees
  - R & D keeps track of what is going on at its facility.
  - Manufacturing plants keep data related to their engineering operations and access to R & D
  - Manufacturing keeps track of local inventory. Access to warehouse also possible.
  - Warehouses keeps local inventory. Manufacturing can access inventory levels.
  - Headquarters keep marketing and sales records per region. Share with other headquarters and can access inventory data at plants and warehouses.

Advantages with DDBs

- *Local autonomy* for DDB nodes
  - Local control
  - Local policies
- Improved performance
  - Avoid data shipping
- Improved Reliability
  - Crashes less severe (if application not dependent on non-local data)
- Expandability
  - Easy to add new nodes (not always linear scale-up because of central directory)
- Sharability
  - Uniform interface and sharing through DDBMS

Replication and fragmentation

- Data replication
  Same data on several nodes
  - For reliability and read performance
  - Not necessary to replicate all tables
    Fully replicated vs *partially replicated*
    Full replication often not realistic!
  - Updates must be propagated to each replica!
  - Special procedures after failures to restore consistency
  - More problematic transaction synchronization!
  - Asynchronous propagation often OK
- Data Fragmentation (= data partitioning)
  Tables transparently split over several nodes
  - For access performance
  - Good when nodes far apart

Problems with DDBs

- Complexity

  Database administration may be complex (e.g. design, recovery)
- Distribution of administrative Control
- Security
- Networking a known problem
- Distributed schema management

  Schema is accessed whenever SQL query issued!

  - Global directory => Central Database becomes *hot spot*

  - Local directories => Data replication

  => Since schema is not updated often but need to be accessed very often it is normally *fully replicated* by the DDBMS.
- Synchronous distributed concurrency control decreases update performance
- Reliability of DDBMS

  - Maintain consistency of replicas

  - Bring up (fragmented) database at failed sites

Transparency

- Replication Transparency
  - User unaware of data replicas
  - Automatic replica propagation at update
  - Asyncrounous replica propagation may suffice
  - Special problems when nodes are down
- Fragmentation Transparency
  - E.g. a logical relation is horizontally fragmented into local physical tables
  - Translation from *global queries* to *fragmented queries*
- Network Transparency
  - Protect user from operational details of network
  - Hide existence of network
  - No machine names in database table references

    Location transparency

    Naming transparency

Distributed database design

- Where to put data and applications?
- Partitioned data:

  Split data into distributed partitions
- Replicated data:

  Several sites have data copies
- Goal: Minimize combined cost of storing data, communication, transactions.
- NP complete optimization problem.

- Distributed Query Processing
  - Automatically done by distributed query processor of DDBMS
  - Analyze query --> distributed execution plan
  - Factors:

    Data replication

    Data fragmentation

  - Communication costs

Distributed Database Design (Fragmentation)

- *Correctness* of Fragmentation

1. *Completeness*

   R has fragments R_1, R_2,...,R_n

   Should be possible to find every tuple of relation in some fragment(s)

   Lossless decomposition of fragmented relation.

2. *Reconstructability*

   Should be possible to reconstruct original relation with some relational operator, ∇:

   $R = \nabla R_i, \forall R_i \in F_R$

3. *Distinctness*

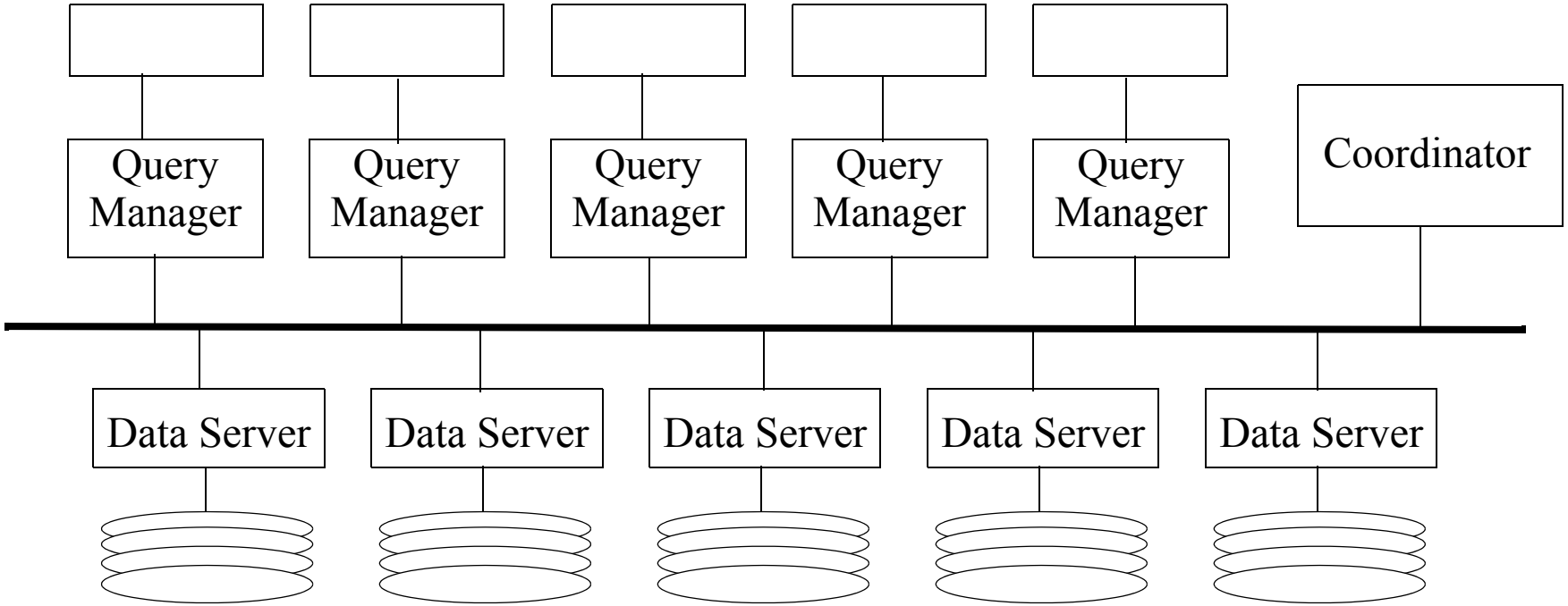
   Data items (tuples or columns) d of R should only occur in exactly one fragment

   $d \in R_i \Rightarrow d \notin R_j$, $i \neq j$

Horizontal: d are rows

- Vertical: d are columns

Parallel databases



Parallel Databases

Parallel data servers

- Use parallel processing in cluster of computer nodes for data servers
- *Automatic* fragmentation and replication of data over the different nodes by the Parallel DBMS (PDBMS)

  To achieve maximal throughput and availablility
- Utilization of modern hardware
  - Multiple independent hardware components interconnected through fast communication medium (e.g. a bus)
  - Modern multi-processor architectures natural to support interquery, intraquery, and intraoperation parallelism
  - Connect disk per processor unit

Distributed Algorithms, SDDS

Distributed data storage algorithms should be designed so that no hot spot nodes are created and so that they scale well.

Problem:

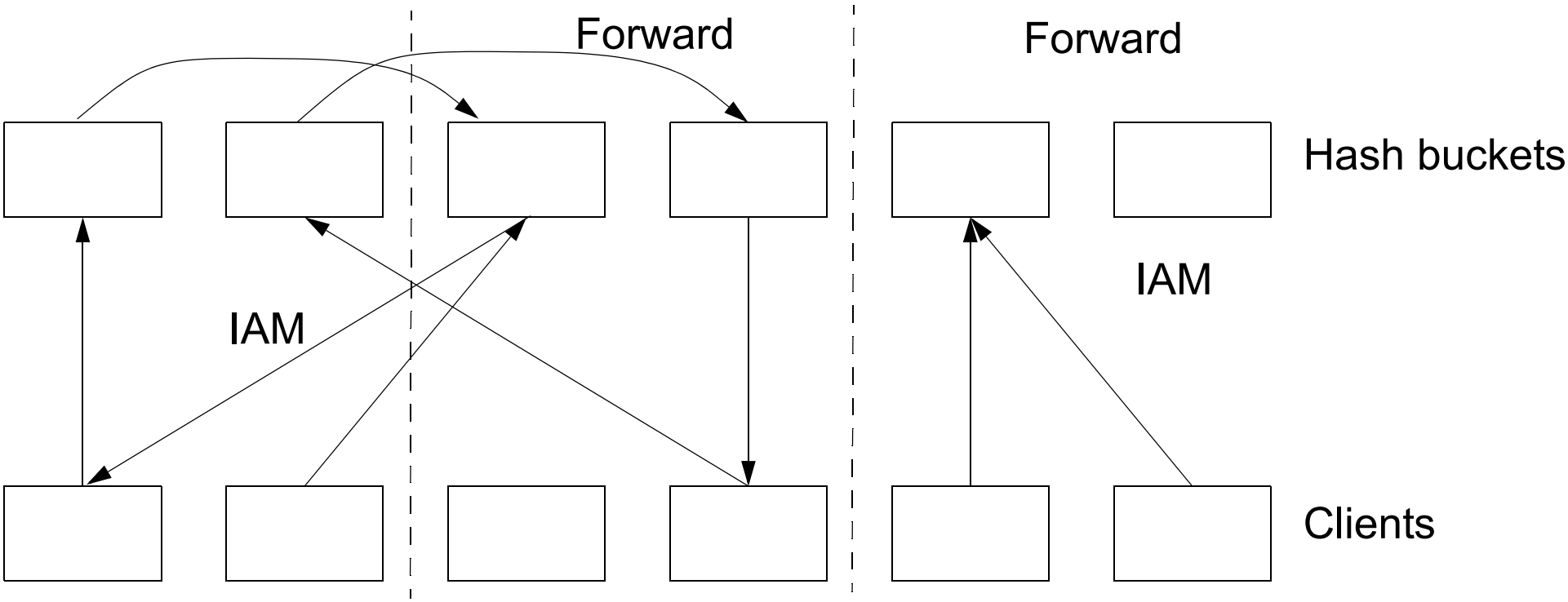
- How to store large storage structures on distributed nodes without any central hot spot.
- How to design storage structure that gracefully grows and shrinks over nodes in network as database evolves.

Solution:

SDDS, Scalable Distributed Data Structures

- Most well known SDDS: LH* by Witold Litwin (Chord very similar)
- LH* is a scalable and distributed hash table.
- Can be extended with high-availability: LH*g, LH*m

LH*



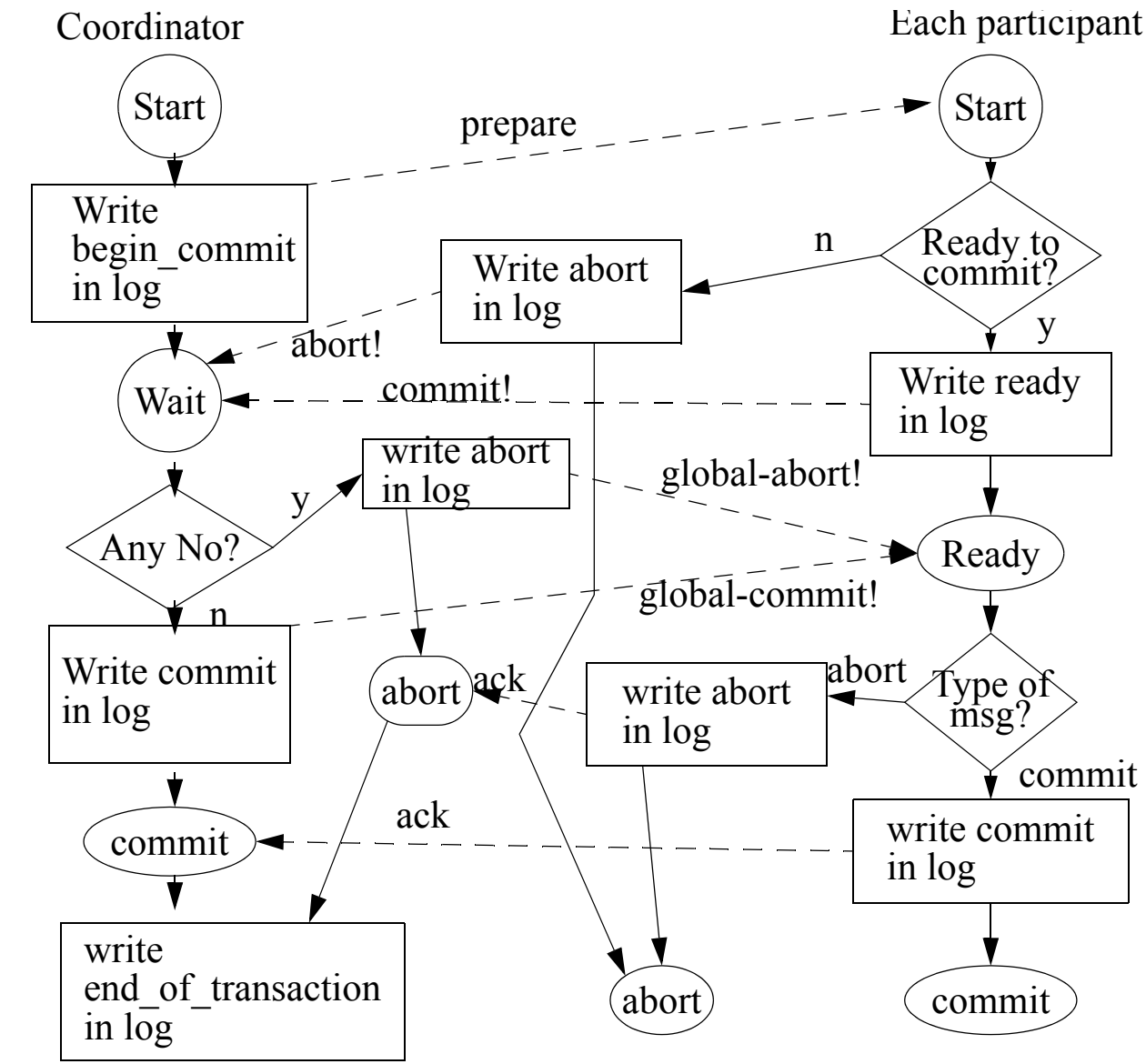
- No directory (no hot spot)
- Each client has approximate *image* of hash buckets
- Addressing error => max 2 forwardings + *Image Adjustment Messages*

Table growth => bucket splits => dynamic extension of # of hash buckets

DDBMS Reliability

Two-Phase Commit Protocol (2PC)

- The most well known protocol to ensure atomic commitment of distributed transactions.
- Extend local site commit protocols by ‘contract’ (synchronization) that they all agree on committing (or aborting) in a distributed transaction.


Coordinator
Each participant
Start
Start
prepare
Write
begin_commit
in log
Ready to
commit?
n
Write abort
in log
abort!
y
Wait
commit!
Write ready
in log
write abort
in log
global-abort!
y
Any No?
Ready
global-commit!
n
Write commit
in log
abort
ack
write abort
in log
abort
Type of
msg?
commit
ack
commit
write commit
in log
write
end_of_transaction
in log
abort
commit