

Database Clusters

Daniel Barbará

What is a cluster?

- Loosely defined is a “small parallel system”
- Is it an SMP? (Symmetric Multiprocessor)
No: there is a difference on how you program clusters

Why clusters?

- The standard litany:
 - Performance: claim that one can get more using a bunch of machines
 - Availability: potentially re-distribute work among computers.
 - Price/performance: the aggregate retains the price/performance of its individual members.
 - Incremental growth: enhance adding more machines.
 - Scaling: (an overused buzzword)
- Enhanced litany:
 - Scavenging: unused cycles are free!

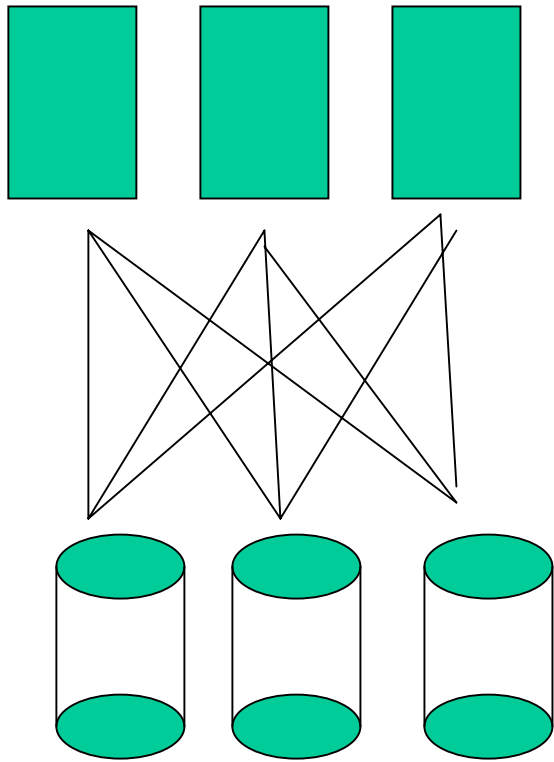
Enabling technologies

- Very high-performance microprocessors:
> 1GHz machines.
- Standard high-speed communication: e.g.,
ATM, SCI.
- Standard tools for distributed computing:
TCP/IP, Jini, CORBA.

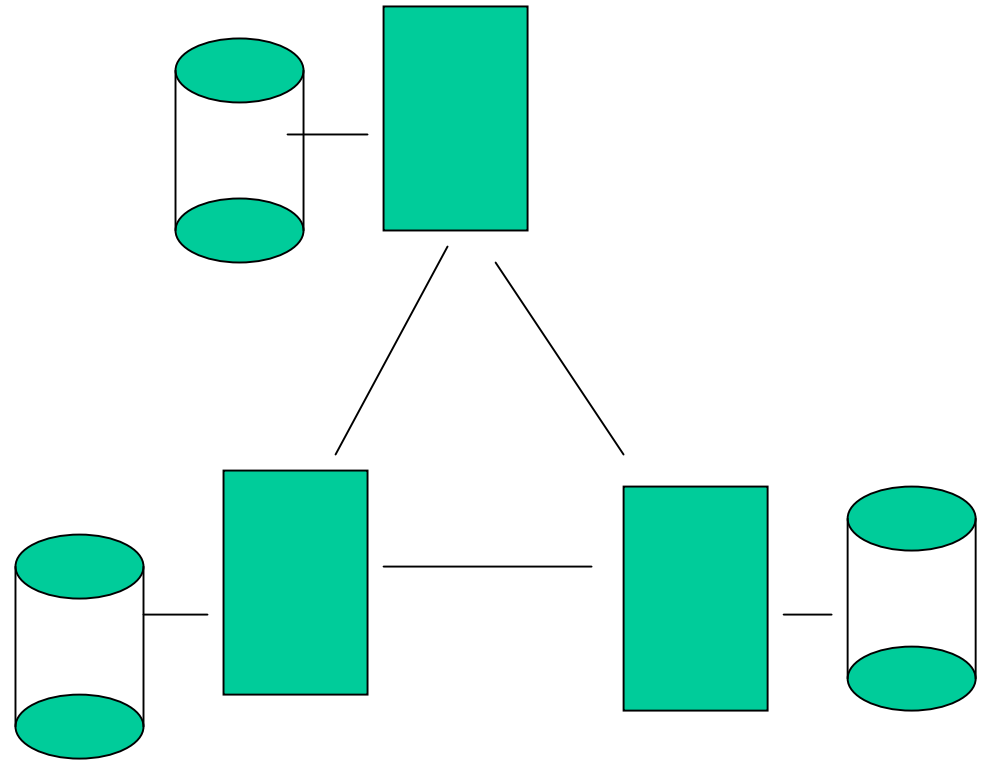
Distinctions from Distributed Systems

- Internal anonymity:
 - nodes of a DS retain their individual identities
 - Elements of a cluster are usually viewed as anonymous.
- Peer relationship:
 - Modern DS use an underlying communication layer that is peer-to-peer. (Although at higher levels, they may be organized into client-server paradigms.)
 - Cluster nodes are often specialized: particularly the use of file-serving cluster nodes.

Cluster Peer File Systems



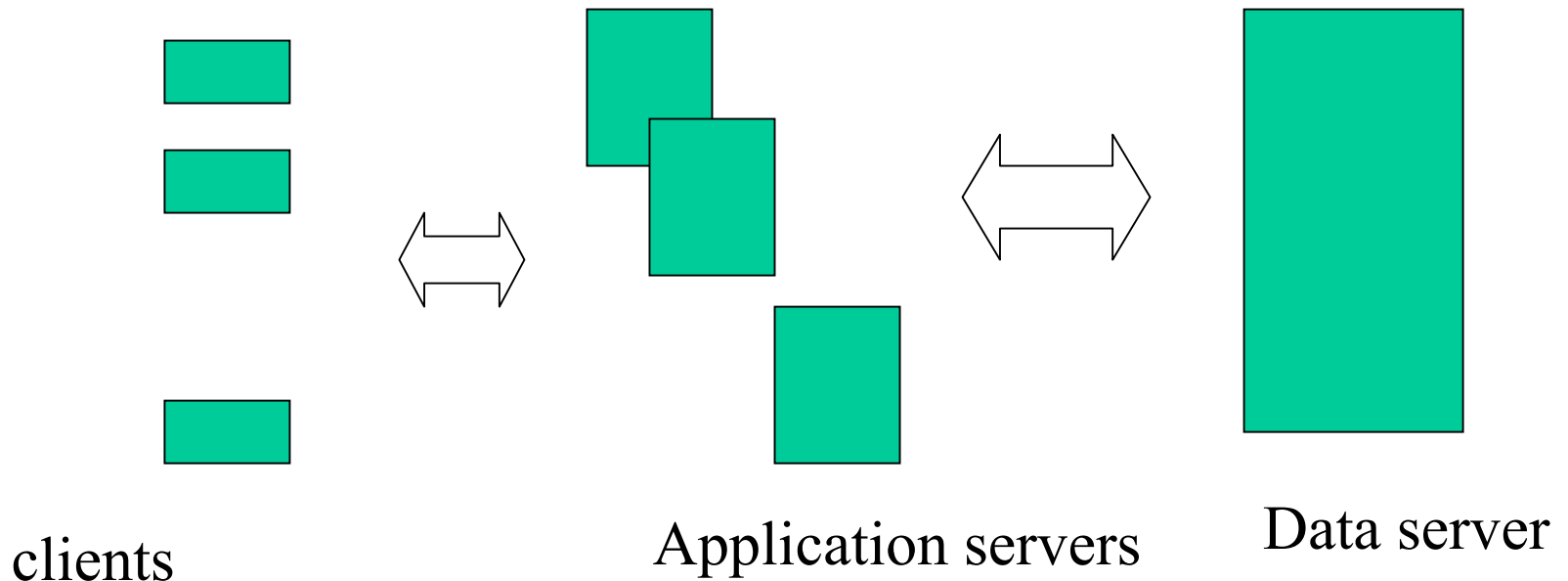
Shared Disks



Shared nothing

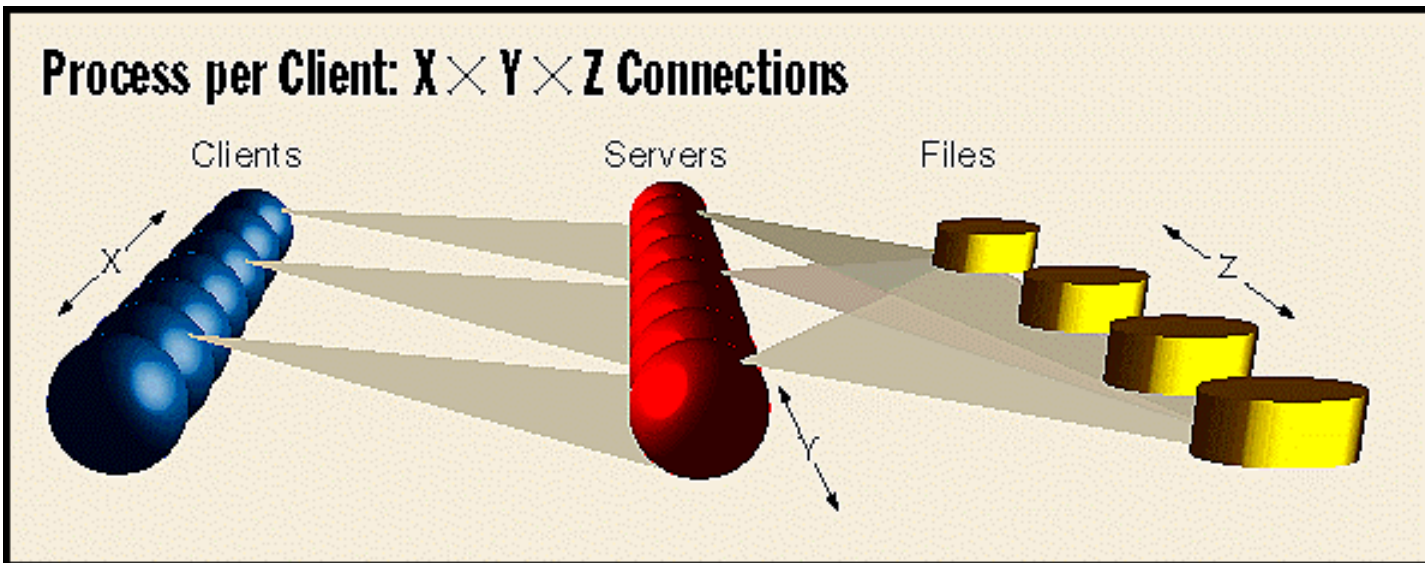
Clusters and three-tiered database applications

- Applications are getting too big for the clients.
- The alternative of performing the application on the “back-end” database engine is problematic: capacity.
- Solution: three tiers:



Scalability

- Simple client/server system: Many clients issue requests, one server responds
- One server process per client (one breaks, the rest are unaffected), but
 - Percentage problem: 10 clients get 10% CPU, 100 get 1%...
 - Polynomial explosion: X clients opening Y applications each with Z open files make for XY processes and XYZ connections.. A crashed operating system...

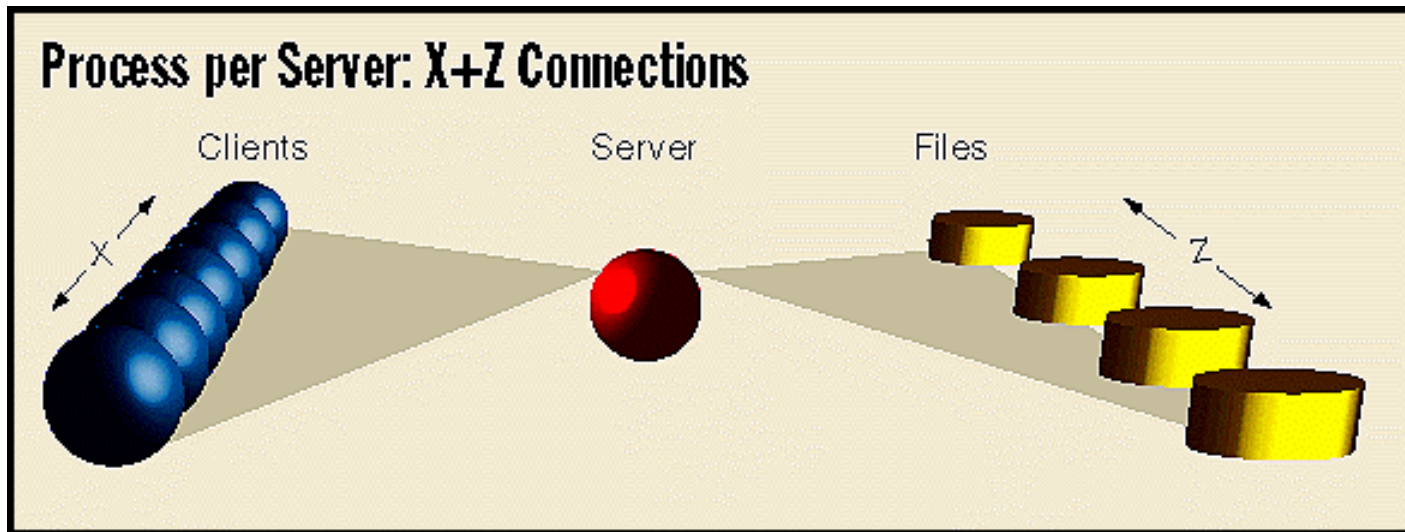


From Jim Gray and Jeri Edward's Byte article "Scale up with TP monitors"

<http://www.byte.com/art/9504/sec11/art3.htm>

A solution

- The obvious solution was to go from a server process/client to a server process/service. One operating system process with a private thread library and a private file system. Ex: IBM's original CICS – 100s of clients – Novell's NetWare, Sybase's SQL Server.
- Only X client connections to the server; Y applications at the server collectively open only YZ files. Better numbers.



From Jim Gray and Jeri Edward's Byte article "Scale up with TP monitors"

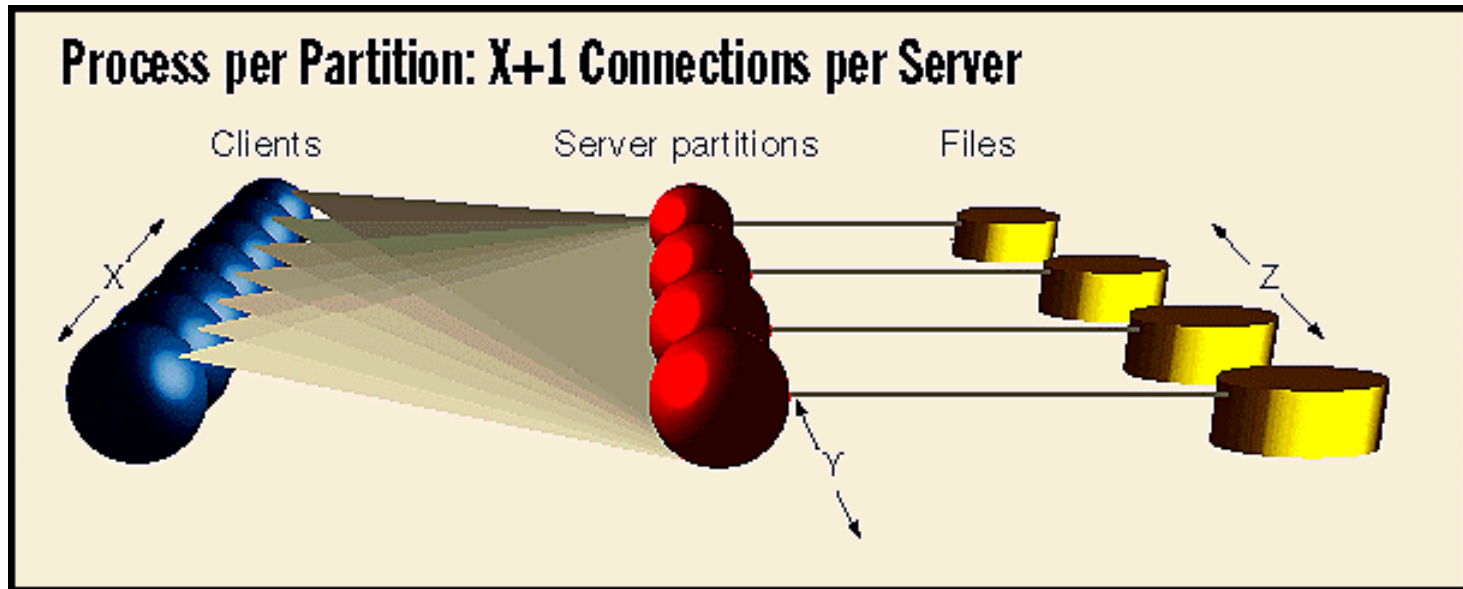
<http://www.byte.com/art/9504/sec11/art3.htm>

Problems with the server/service solution

- It does not scale to SMMP (symmetric multiprocessors).
- Collapses all applications into a single address space...(each becomes a sitting duck for any bug...)

Next step

- Process-per-application-server partition: specialize a process or processor to service a particular application. Scale the system by adding servers per application or partition the application data. Ex: New IBM's CICS, NetWare, Sybase and ORACLE applications.



From Jim Gray and Jeri Edward's Byte article "Scale up with TP monitors"

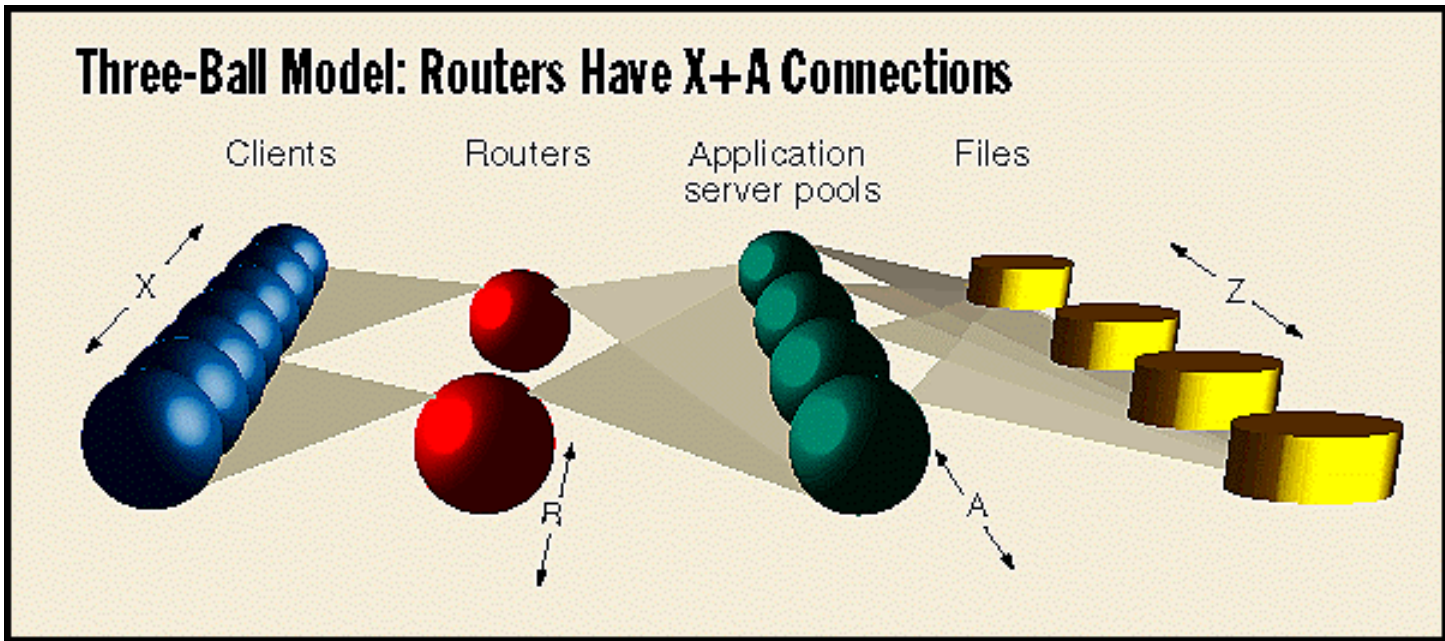
<http://www.byte.com/art/9504/sec11/art3.htm>

Problem with process-per-application

- We are back to square one: polynomial explosion. Clients must connect to each application-server partition, log on to it and maintain a connection.
- ALL THESE SOLUTIONS HAVE TWO KINDS OF PROCESSES: Clients and Servers (TWO BALLS)

ROUTERS

- Three-ball model: Clients, Routers, and Servers.
- Client connects to a Router, and the router brokers clients requests to servers (X+A connections)
- Ex: IBM's IMS – first three-ball system, with a single router– DEC's ACMS and RTR, A.T.&T. Tuxedo and Topend, Transarc's Encina, IBM's CICS reimplemented in UNIX as a three-ball system on top of Encina's toolkit.
- ALL TP monitors in the market are three-ball systems



From Jim Gray and Jeri Edward's Byte article "Scale up with TP monitors"

<http://www.byte.com/art/9504/sec11/art3.htm>

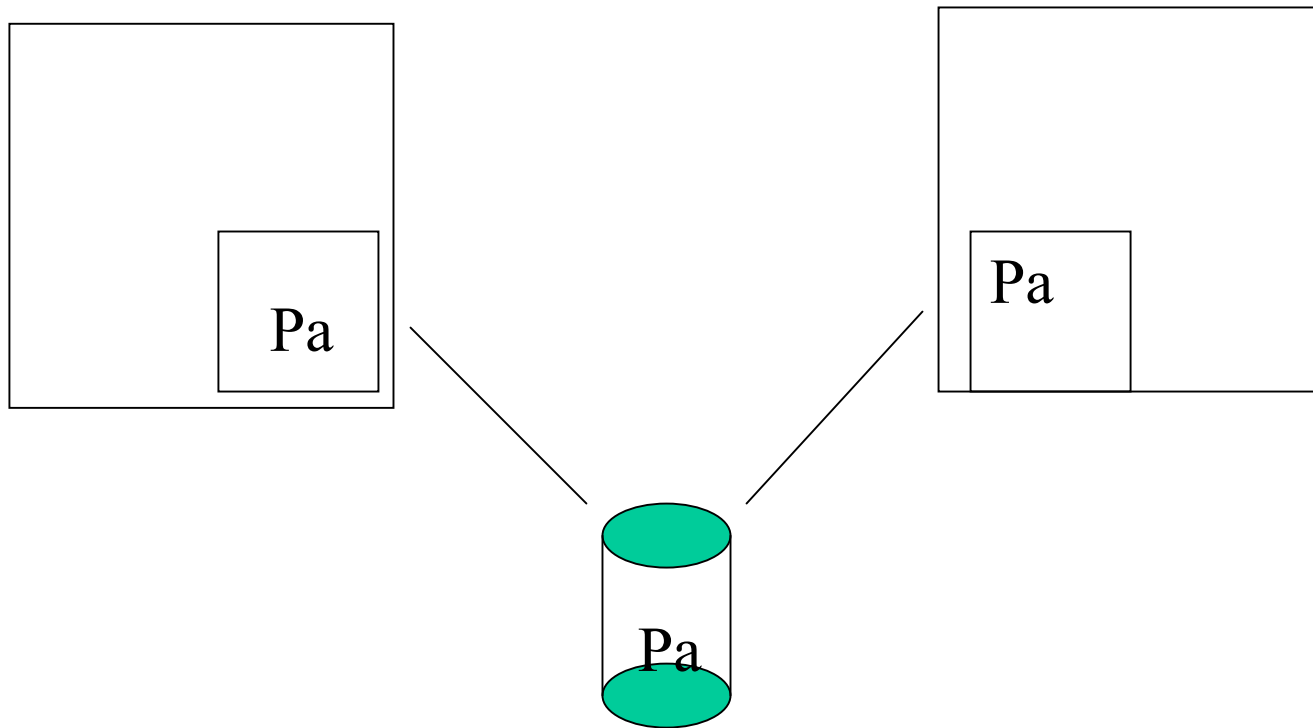
Other comparisons

Characteristic	High parallelism	Cluster	Distributed
Number of nodes	Thousands	Tens or less	Thousands to millions
Performance	Turnaround	Throughput and turnaround	Response time
Node individualization	None	None	Yes
Internode communication	proprietary	varies	standards
Node size	smaller	larger	larger
Inter-node security	Nonexistent	Unnecessary if unexposed	Required
Node OS	Homogeneous	Often homogeneous	heterogeneous

Coherency and Concurrency Control

- Consider “shared disks” systems:
- Each node has its individual main memory and local page cache.
- Problems: concurrency control and cache coherency. Both stem from the fact that data pages can be dynamically replicated in more than one server cache to exploit access locality. Hence, synchronizing reads and writes require distributed lock management and cache invalidation.

Cache coherency problem



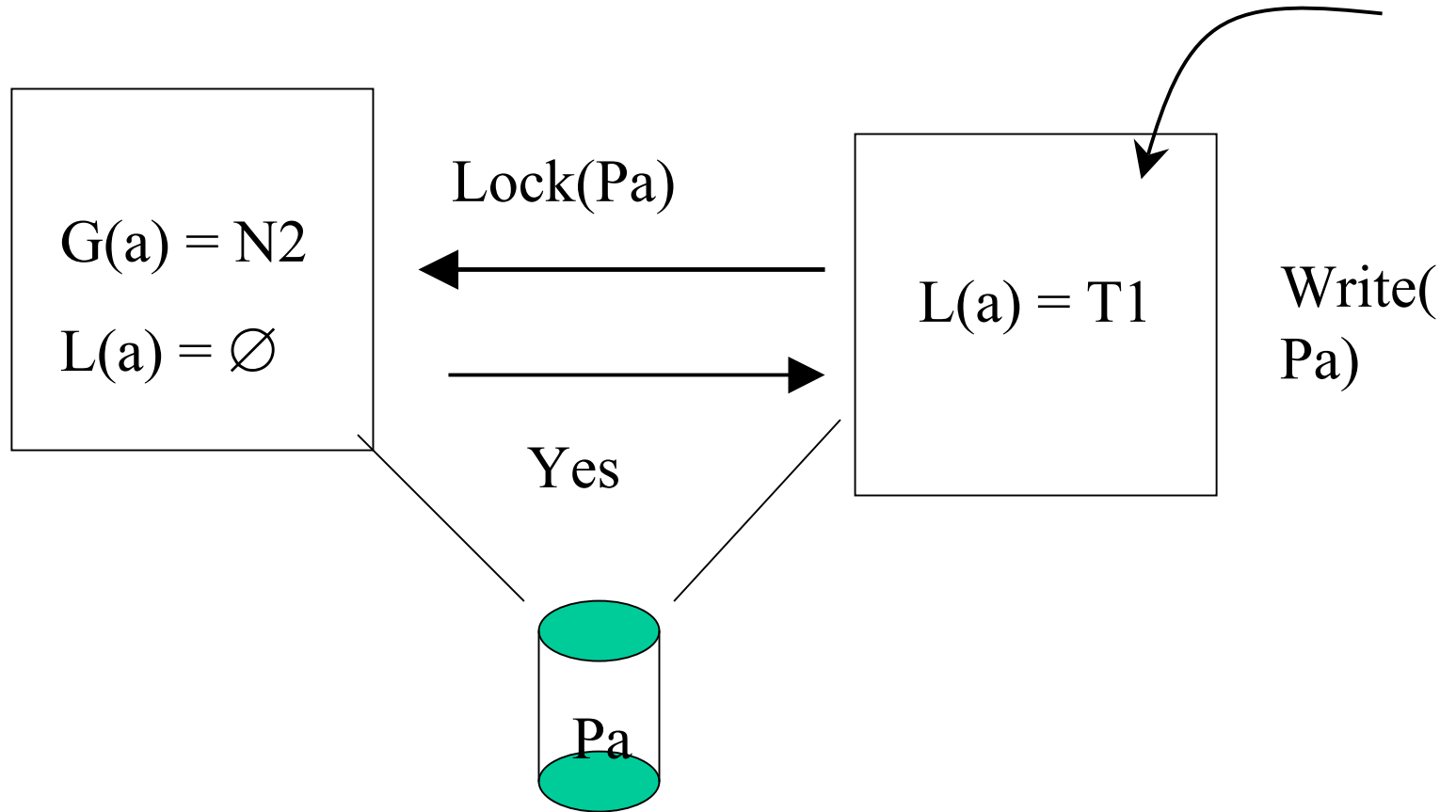
Concurrency control

- Assumption: each transaction, once routed to an specific server in the cluster is executed on this server.
- Each server has a global and a local lock manager. Data items are assigned to global lock managers in static manner.
- Global lock managers know if an item is locked and by which server and mode . Local lock managers know which transaction holds locks.

Concurrency control

- When a transaction requests a lock (or releases one) it addresses its local lock manager, which contacts the global lock manager.
- Messages to the global manager can be saved by authorizing local managers to grant locks: writes to one local manager, reads to many.

Concurrency Control



Cache coherency

- Servers keep frequently and recently accessed pages in their local caches.
- Hence, multiple copies of the page may reside in several caches across a cluster.
- Need to ensure that:
 - Multiple caches can hold up-to-date versions of a page simultaneously as long as the page is only read.
 - Once a page has been modified in one of the caches, this cache is the only one allowed to hold a copy.

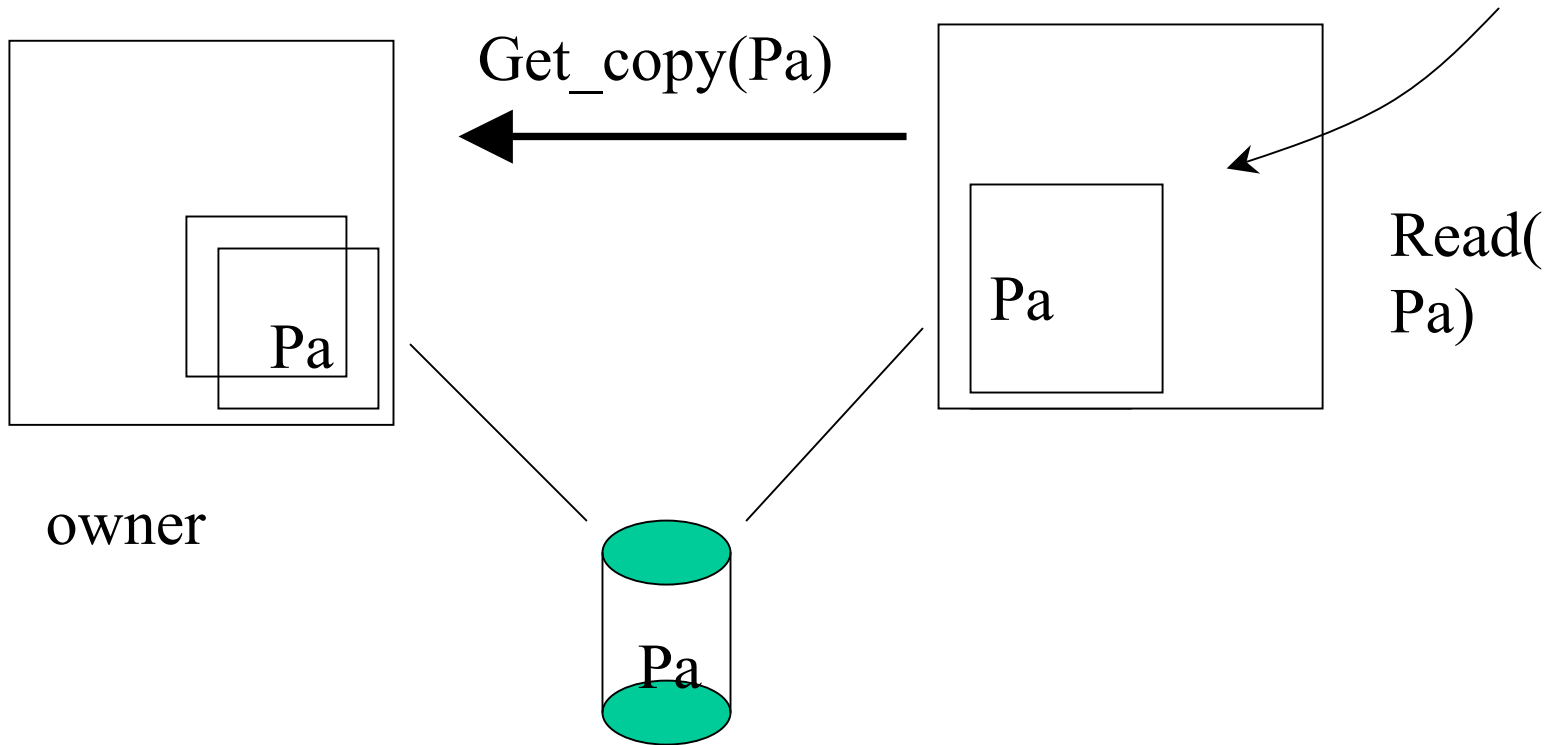
Cache coherency

- A protocol maintaining these invariants is **owner-based coherency control**.
- Initially, each page is statically assigned to one of the servers in the cluster; this is called the **home** of the page.
- A server holding an up-to-date and ready-to-access copy of a page is an **owner** of the page.

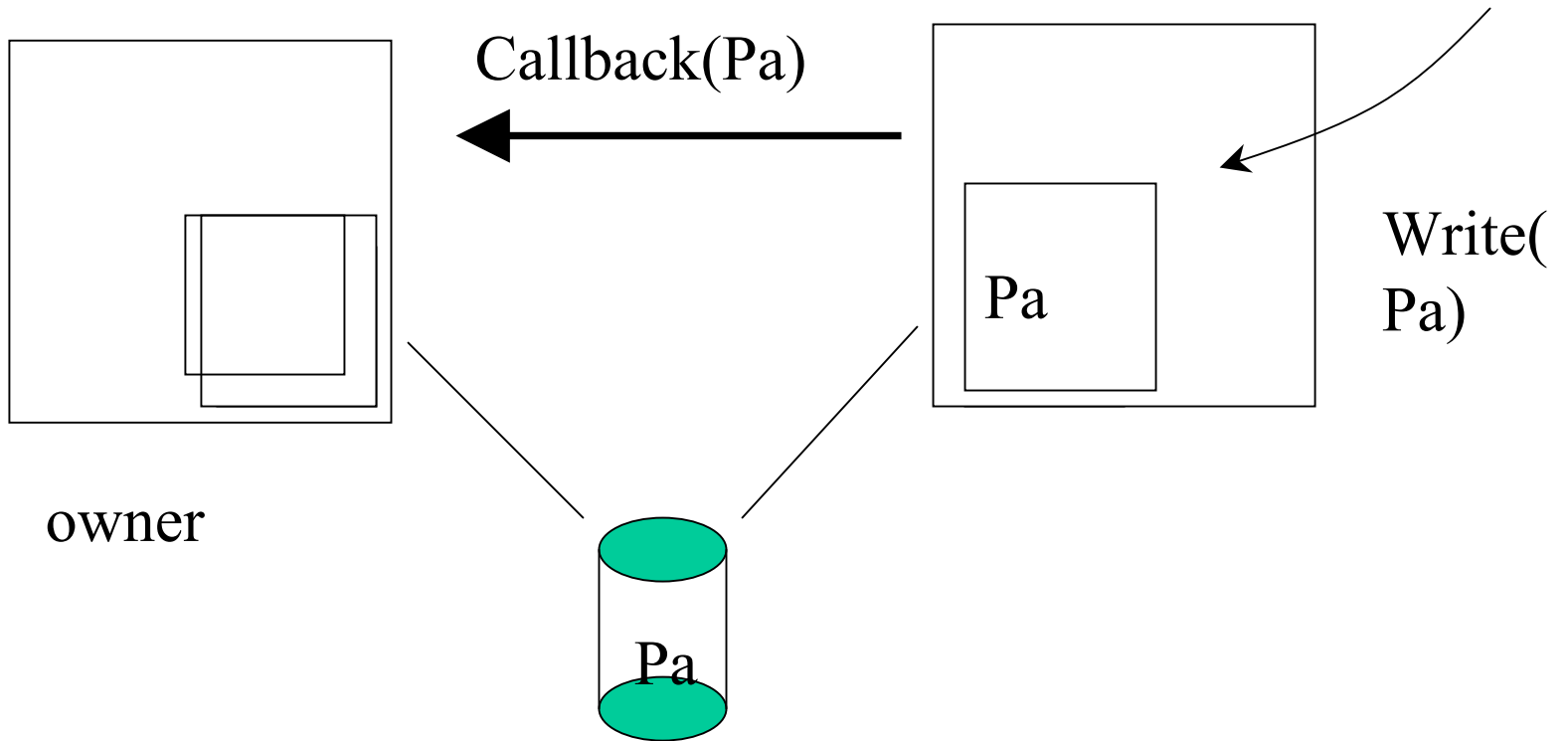
Cache coherency

- When a server requests a page for read access and is not yet an owner, it must obtain a copy from one of the owners and add itself to the list (maintained in the home.)
- When a server requests a page for write access, it has to identify all owners of the page and issue callback requests to all; by this their copies are invalidated and the server becomes the solely owner of the page.

Owner-based protocol



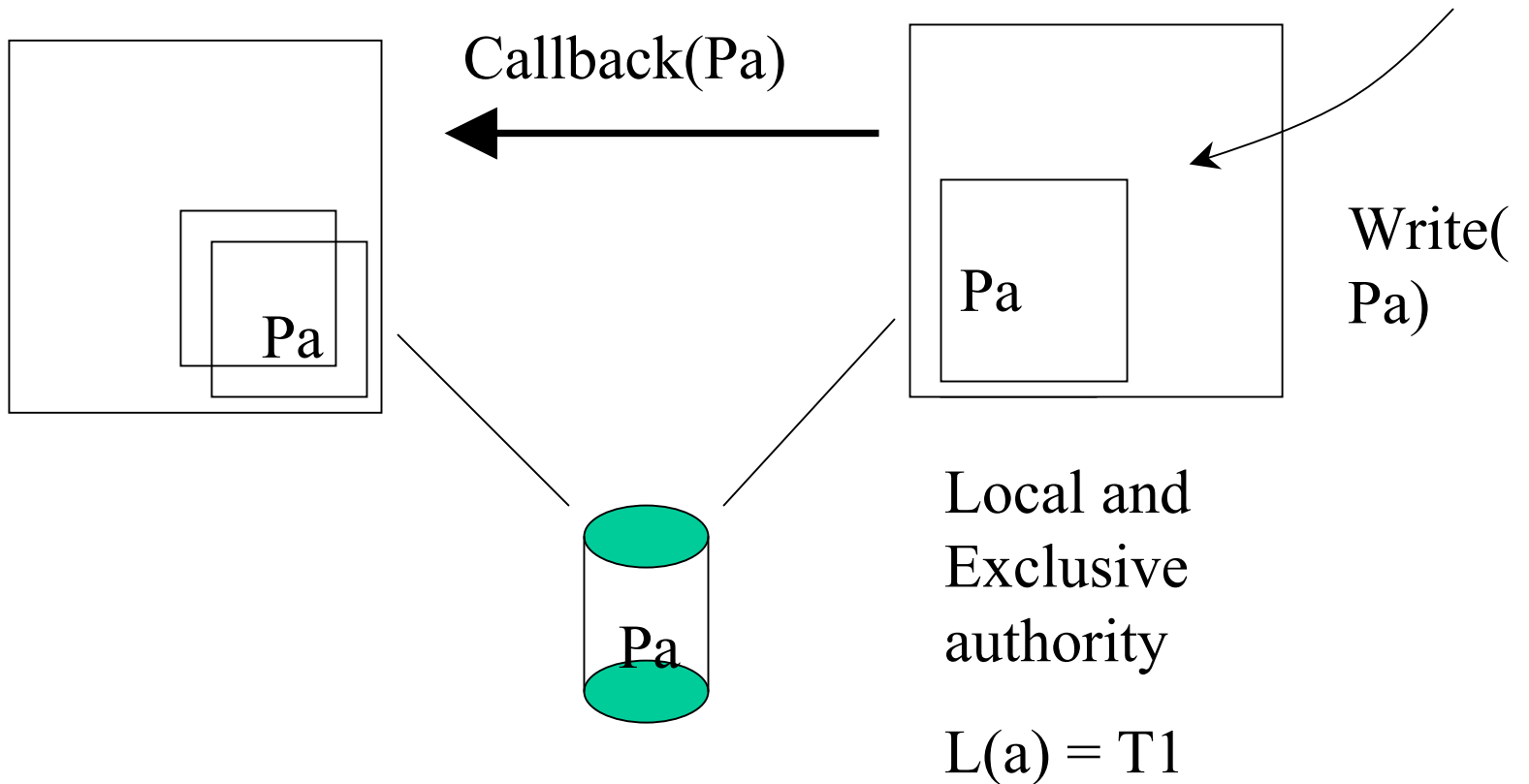
Owner-based protocol



Combining concurrency and coherency control

- Callback locking:
- Every owner of a page holds a read lock authority for the page and can acquire and release read locks locally.
- An exclusive owner of the page holds write lock authority.
- Callback revokes the local lock authority.

Owner-based protocol



Shared-nothing systems

- Pages are brought to the memory of the server from another server that happens to have that page in its cache (its owner). **Data shipping.**
- Locking then becomes simpler.

Recovery

- Ideally, each server writes its own local log.
- Each server writes log entries only locally to its private log.
- In the general case, where redo-relevant log entries for the same page may be dispersed across multiple private logs, we require some form of handshake among the log managers to insure that the log sequence numbers are properly coordinated.

Recovery

- All logs are accessible by all servers.
- The server in charge of the recovery for the failed server needs to read and merge the logs.
- Reading and merging multiple log files can be avoided if a dirty page is always flushed to the stable database when it is transferred to another server.

The following is taken
from a talk by Mark Wood
(Microsoft)
on SQL clusters



SQL Server and NT Cluster Manager Availability Demo

Wolfpack
NT Clusters



Microsoft Server Program Developers
Conference
November 4-6 1996



Windows NT Server and Microsoft SQL Server

A Fault-Tolerant Server Platform

- **Protection against application faults**
 - Preemptive multitasking, protected memory
 - *Instrumented*: monitoring, logging, alerting
 - SQL Enterprise Manager automates operations
 - Distributed Transactions recover from failures
- **Protection against data loss**
 - NTFS Journalled & recoverable + RAID 2, 5
- **Protection against power loss**
 - Uninterruptible Power Supply (UPS) support
- **Protection against logon or name server failure**
 - Replicated Directory & Backup logon & DNS servers





Windows NT Server Clustering

High Availability On Standard Hardware

Standard API for clusters on many platforms

No special hardware required.

Resource Group is unit of failover

Typical resources:

shared disk, printer, ...

IP address

Service (SQL, File, Print, ...)

API to define

resource groups,

dependencies,

resources,

GUI administrative interface

Defined by a gang-of-60 (everybody who is anybody).



2-Node Cluster in beta test now.

Available 97H1

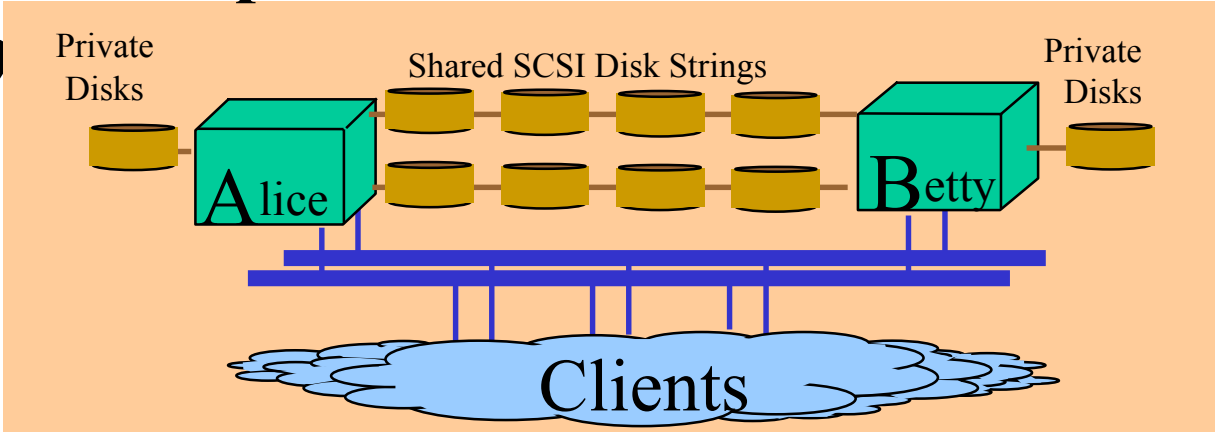
>2 node is next

SQL Server and Oracle

Demo on it today

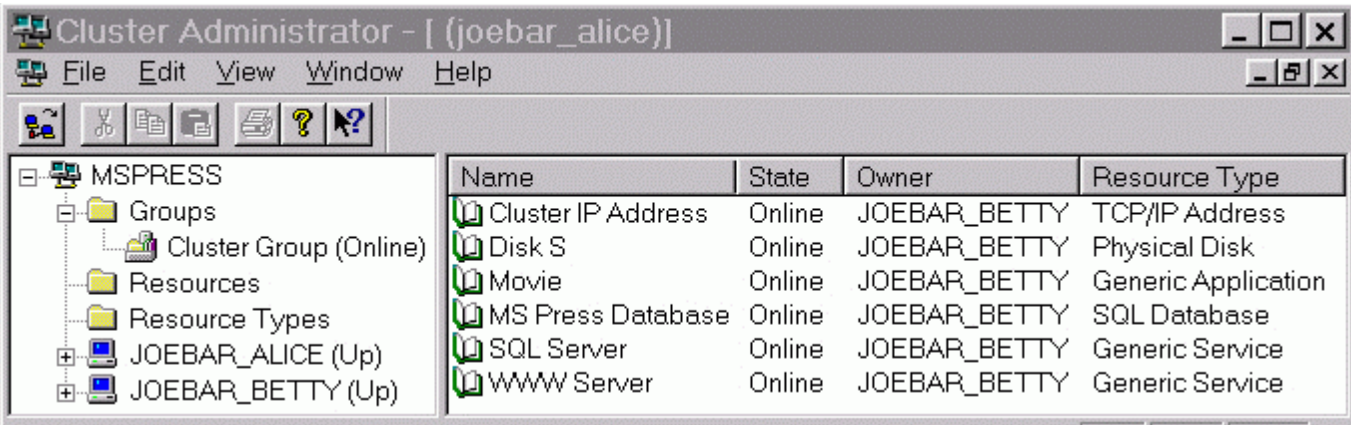
- Two node file and print failover

Wolfp



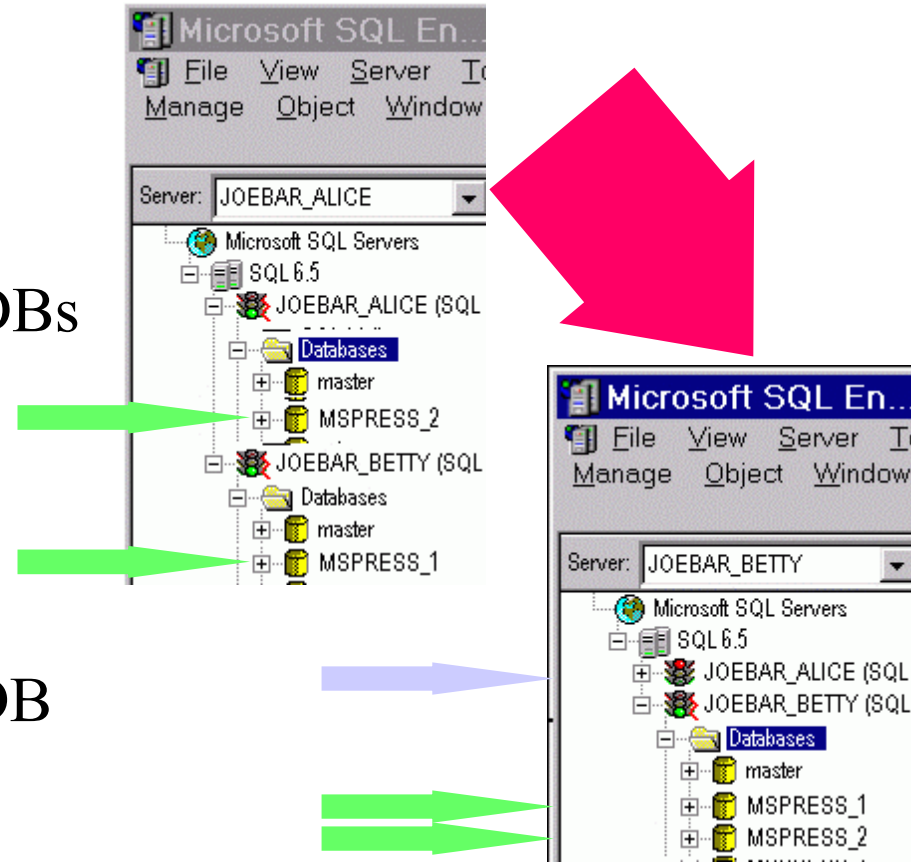
Q1

- GUI admin interface



SQL Server 6.5 Failover

- Failover unit is Database
- Symmetric:
 - each server can serve 1/2 the DBs
- When one server fails,
 - other takes over shared disks
 - recovers database
 - starts offering service to the DB



- Client failover via reconnect

IP impersonation or

ODBC or DBlib reconnect in SQL Server 6.5

Demo Configuration

Server "SQL_Alice"

SMP Pentium® Pro Processors
Windows NT Server with Wolfpack
Microsoft Internet Information Server
Microsoft SQL Server

Server "SQL_Betty"

SMP Pentium® Pro Processors
Windows NT Server with Wolfpack
Microsoft Internet Information Server
Microsoft SQL Server

Interconnect

standard Ethernet

SCSI Disk Cabinet

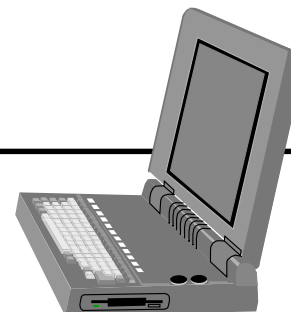
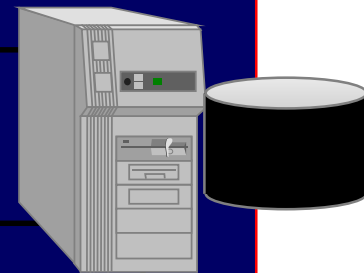
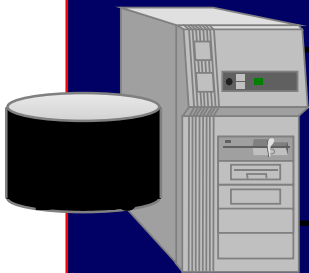
Windows NT Server Cluster

Administrator

Windows NT Workstation
Cluster Admin
SQL Enterprise Mgr

Client

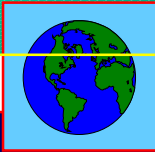
Windows NT Workstation
Internet Explorer
MS Press OLTP app



Demo Administration

Server "SQL_Alice"

Runs SQL Trace
Runs Globe



```
select * from emp  
insert into T values ....  
commit work..
```

Server "SQL_Betty"

Run SQL Trace

```
select * from emp  
insert into T values ....  
commit work..
```

SCSI Disk Cabinet

Windows NT Server Cluster

Cluster Admin Console

- Windows GUI
- Shows cluster resource status
- Replicates status to all servers
- Define apps & related resources
- Define resource dependencies
- Orchestrates recovery order

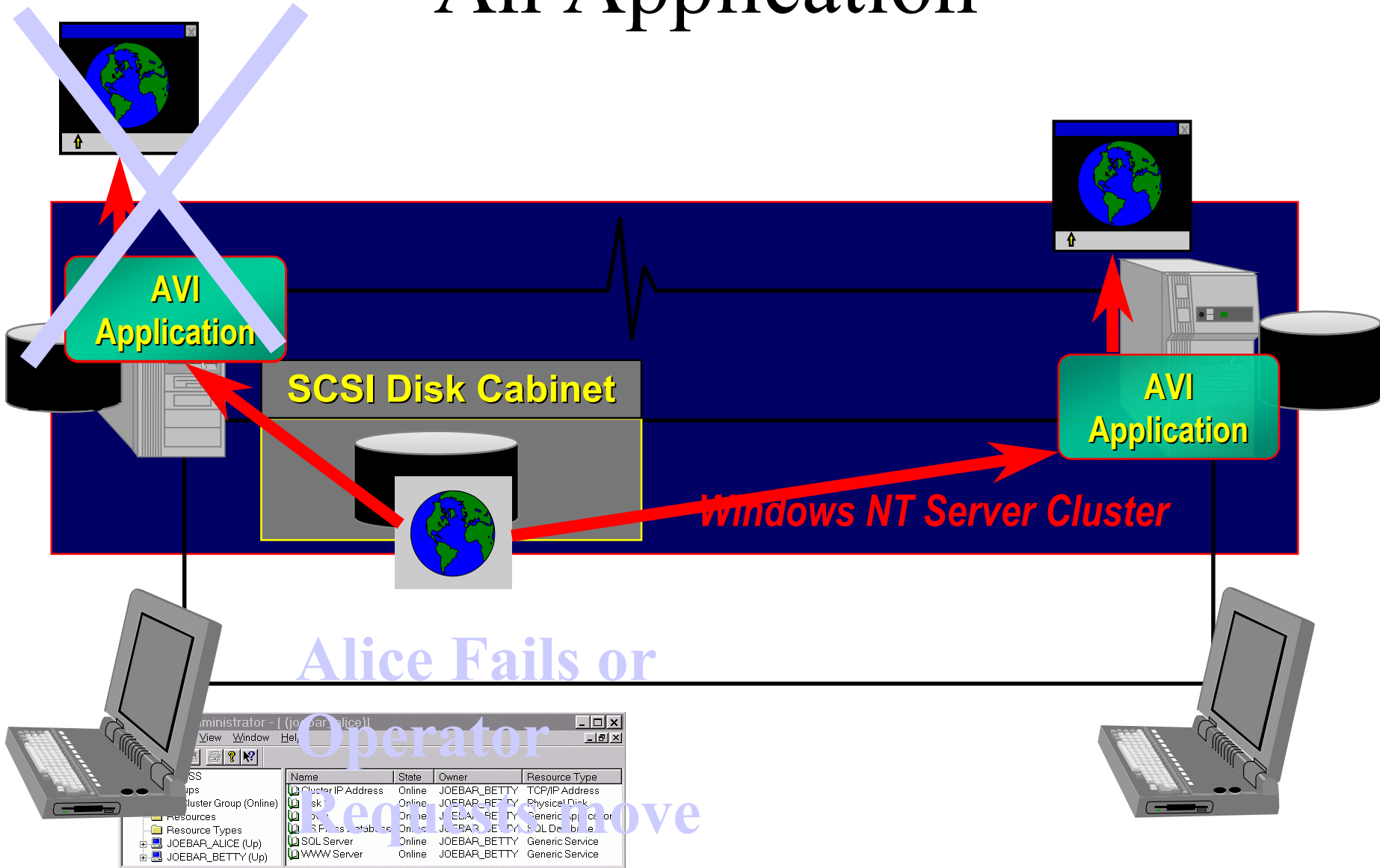
SQL Enterprise Mgr

- Windows GUI
- Shows server status
- Manages many servers
- Start, stop manage DBs

Client



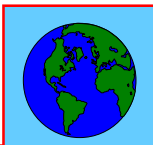
Demo Moving or Failing Over An Application



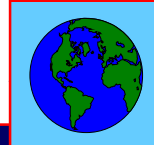
2: Request Failover to Betty

No SQL Activity

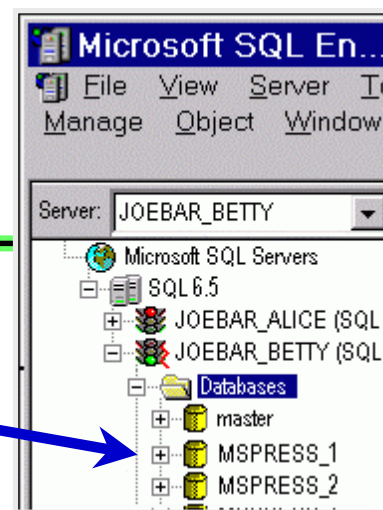
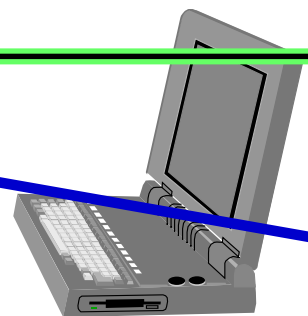
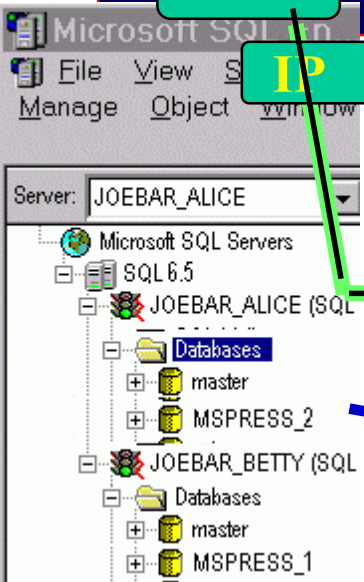
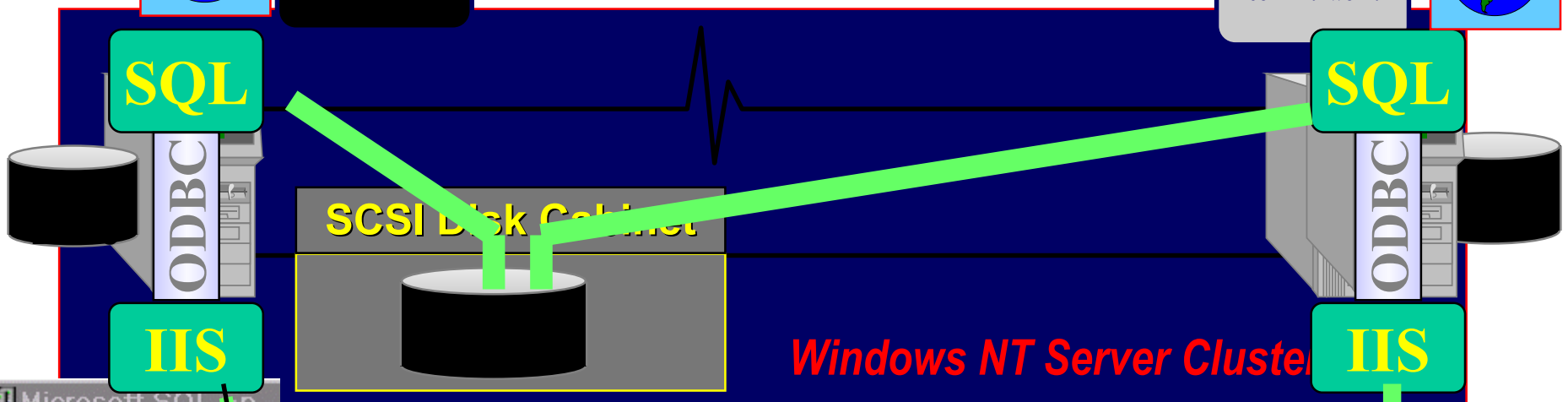
SQL Activity



select * from emp
insert into T values
commit work.



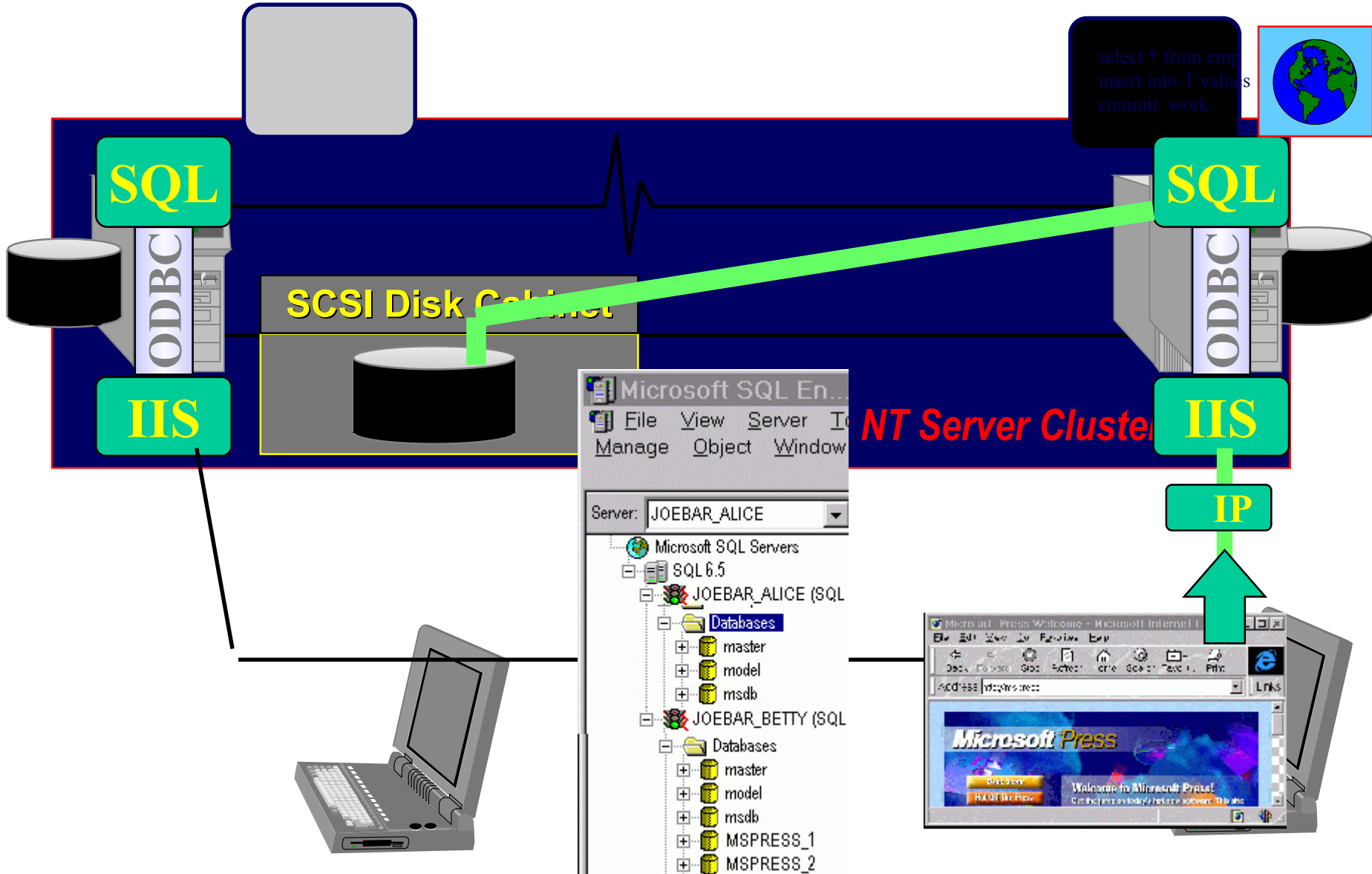
select * from emp
insert into T values
commit work.



3: Betty Delivering Service

No SQL Activity

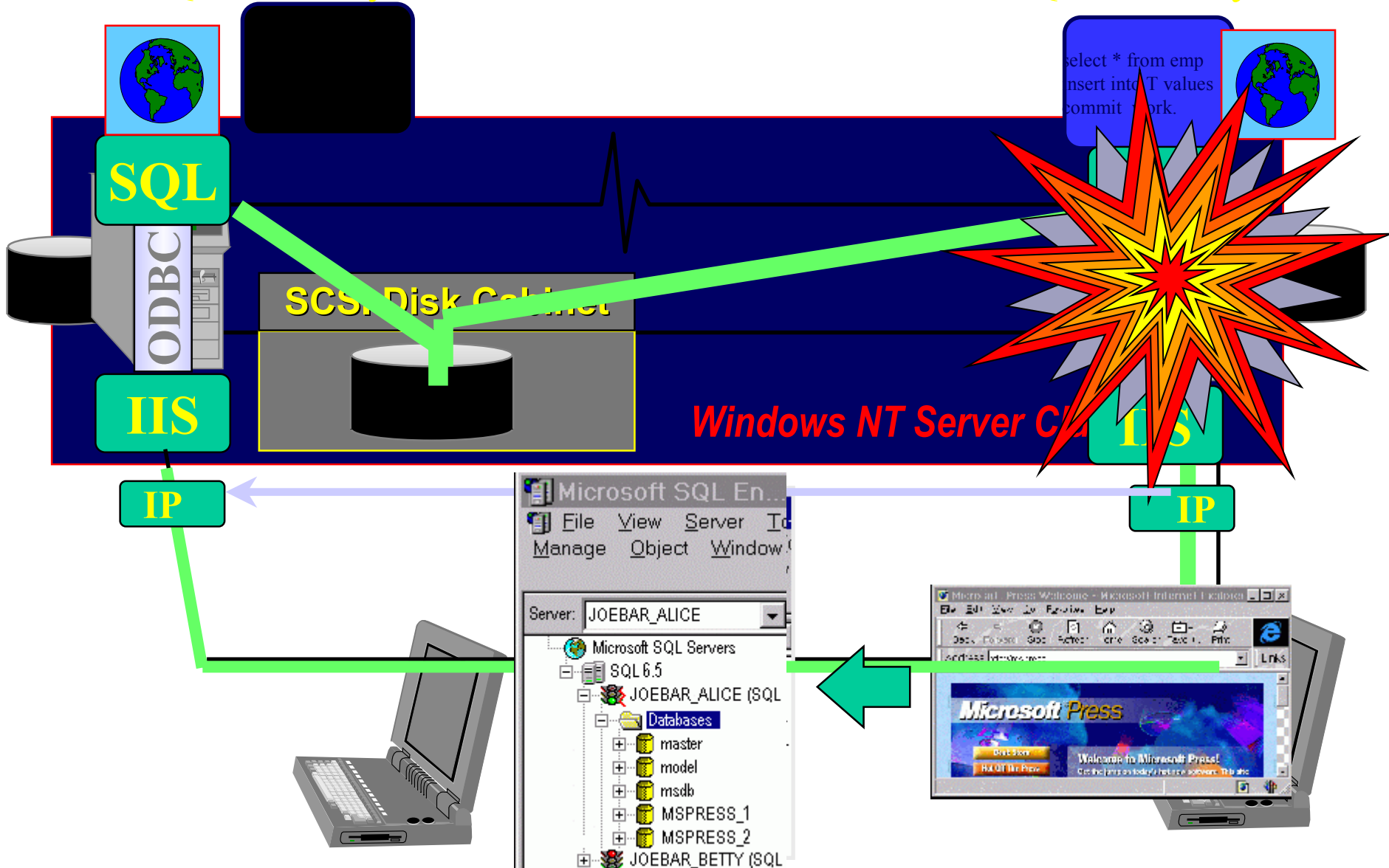
SQL Activity



4: Power Fail Betty, Alice Takeover

No SQL Activity

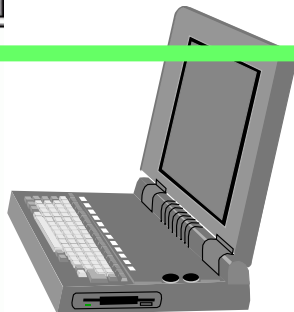
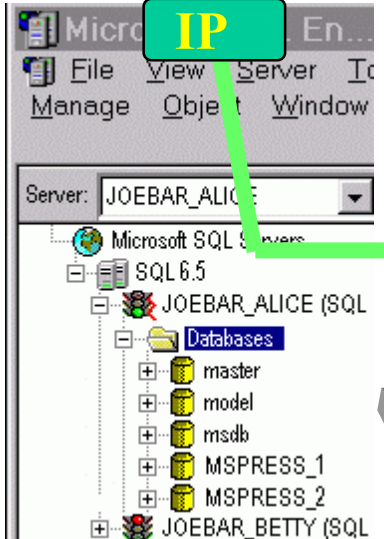
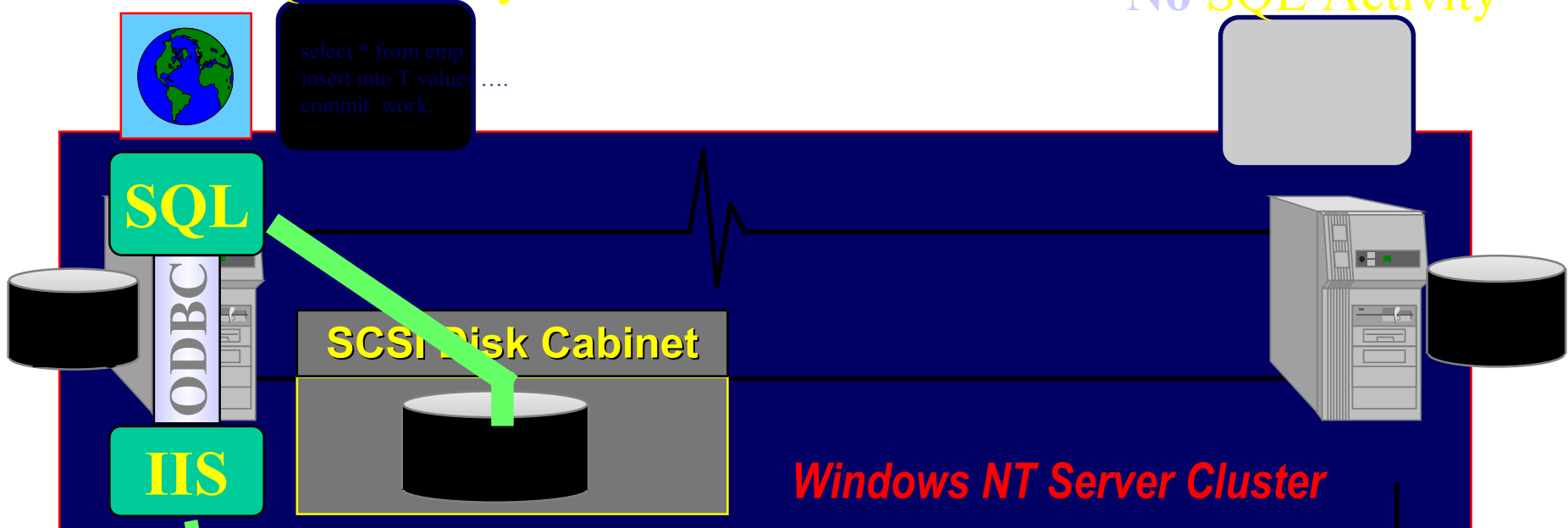
SQL Activity



5: Alice Delivering Service

SQL Activity

No SQL Activity



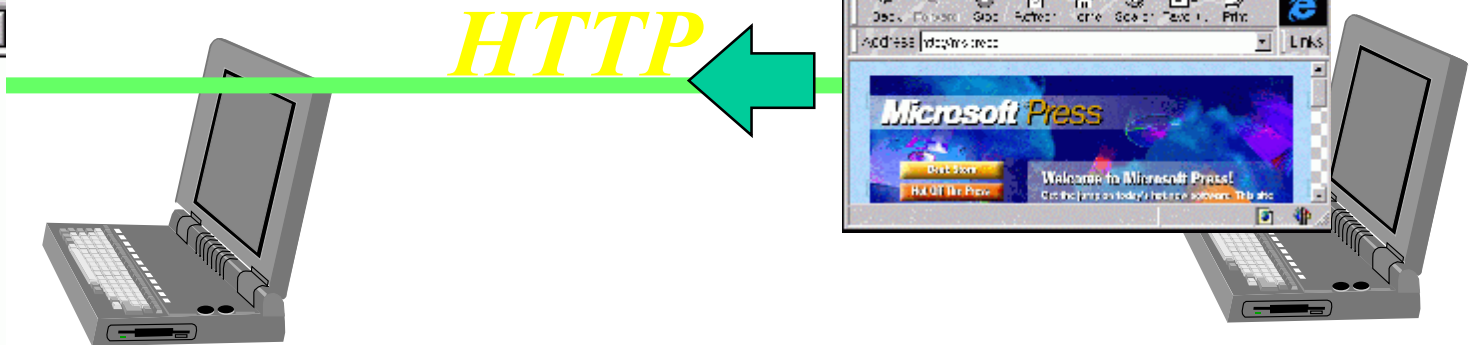
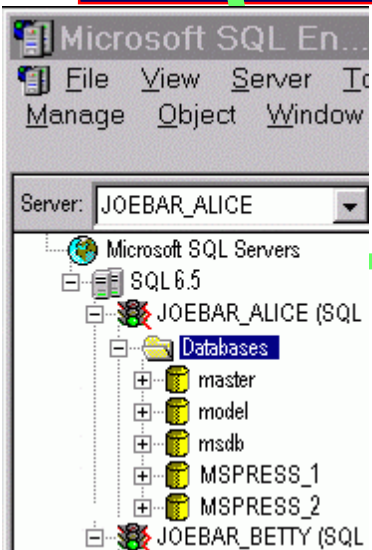
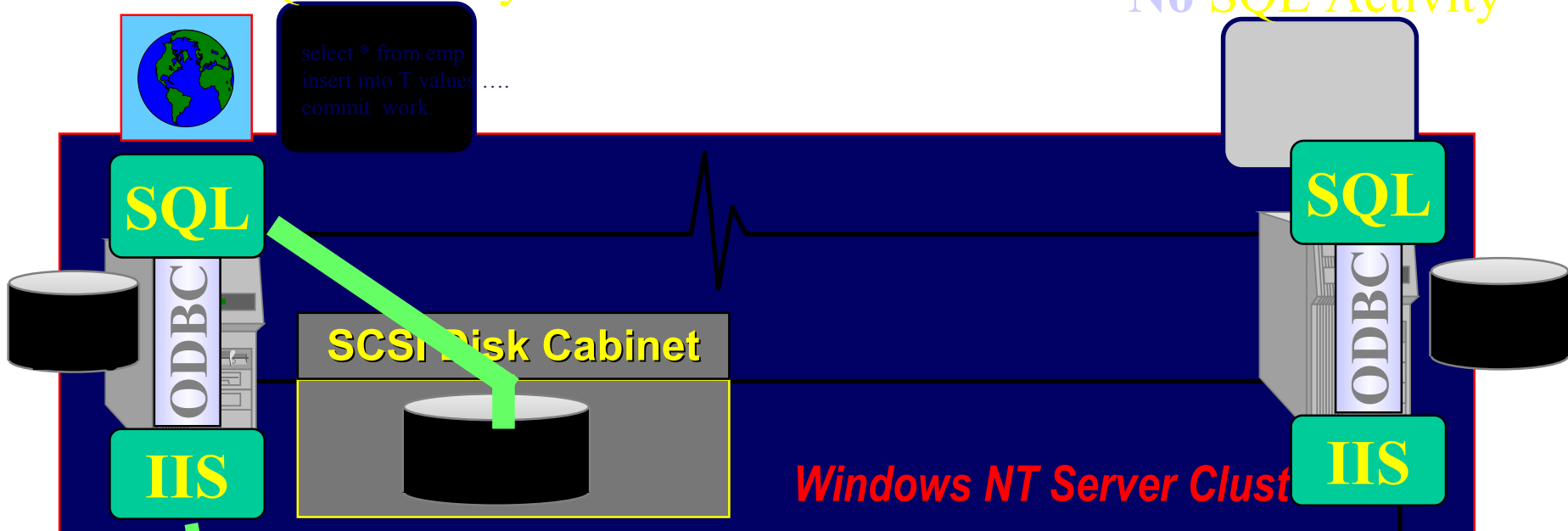
HTTP



6: Reboot Betty, now can takeover

SQL Activity

No SQL Activity



What We Demonstrated

- Quick (5-second) NT failover
- Quick (10-second) SQL Server failover
 - longer if SQL has more work to redo
- Using SQL Server and NT Clusters
 - GUI tools to configure and manage
 - Central operations console
- Using commodity qualified SCSI parts
- Symmetric design: each can serve 1/2 DB
- No single point of failure



One billion transactions per day

- MS wanted to demonstrate that a three-tier application using Microsoft Transaction Server, DCOM, and SQL Server can scale to serve huge online applications. To do this they picked the DebitCredit scaleable transaction application as workload

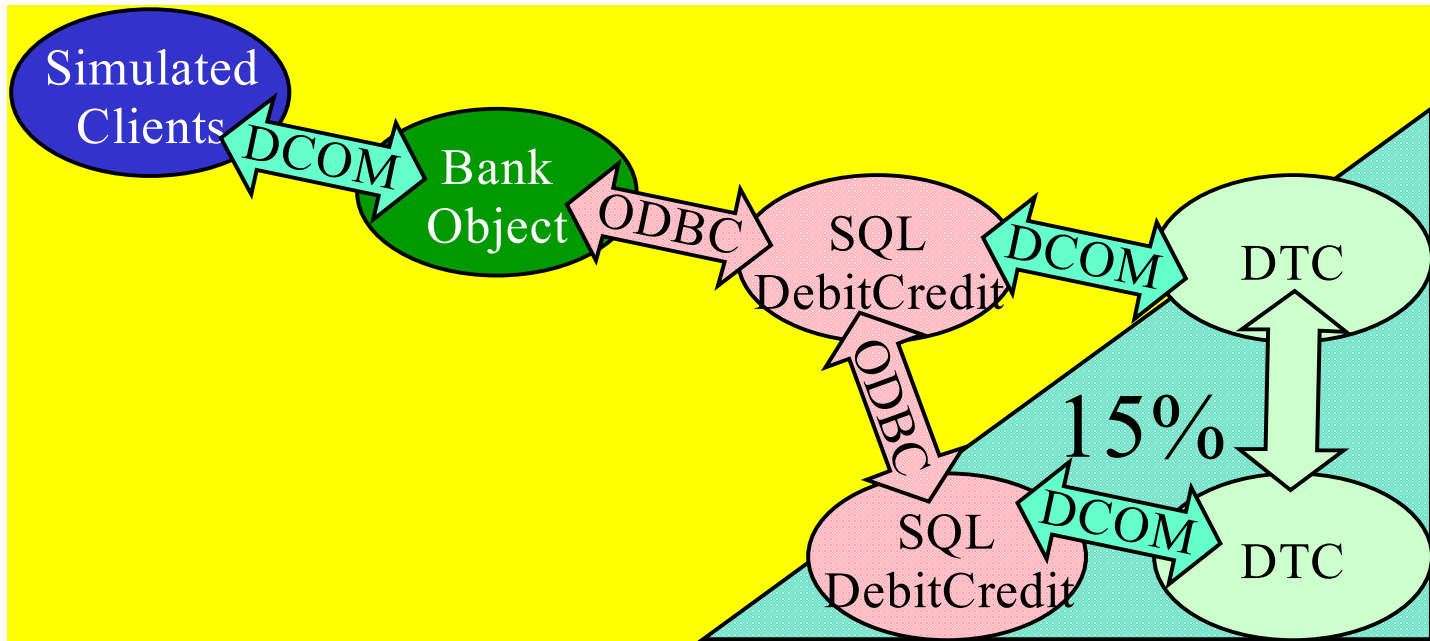
DEBIT-CREDIT

- The application postulates a memo-posting banking application with many branches and many customers. The application is a debit or credit to some account. The application also updates the teller cash drawer, branch balance, and does a memo-post to a general ledger. A thirty-day history of this ledger is kept online.

DEBIT-CREDIT

- Each of the tellers at a branch collectively generate one transaction per second. Each branch has ten tellers and 100,000 customer accounts. The system grows by adding more branches, tellers, and accounts. A thousand-transactions-per-second system has 1,000 branches, 10,000 tellers, and 100 million accounts. There is one additional requirement, 15% of all transactions are "remote". That is, customers do debits and credits at their home branch 85% of the time, but 15% of the time a customer goes to a non-home branch and does a debit or credit to his account.

Flows among the processes



The flows among the processes in a DebitCredit transaction. 85% of the time, the transaction involves a single SQL Server. 15% off the time the transaction involves a second SQL Server and one or two Distributed Transaction Coordinators.

Database size

- The database is spread among twenty servers running Microsoft SQL Server 6.5. Each server is sized for 800 bank branches. So each server manages eighty million bank accounts. The sizes of the tables are 38 Billion records for a total of 2.3 TB

Application

- In summary, this is a classic 3-tier application. The 1,100 threads simulate a network of 160,000 terminals. The threads make DCOM DebitCredit calls on the bank object. The DCOM calls are translated to ODBC calls that are executed by Microsoft SQL server. Most of the transactions involve only a single node, but about 30% of the work involves multi-node transactions.

Hardware

- The hardware consists of 20 workflow nodes (the middle tier) driving 20 SQL Server nodes. Five additional nodes coordinate distributed transactions involving two or more server nodes. In all, the cluster has 140 processors executing about 14 billion instructions per second on this workload.

Assessment

- The system runs approximately 1.2 B transactions per day. That is nearly a million transactions a minute and 14 thousand transactions a second. During the day-long run, each of the 20 SQL Servers in the system is checkpointing every five minutes

The Terra-Server (MS)

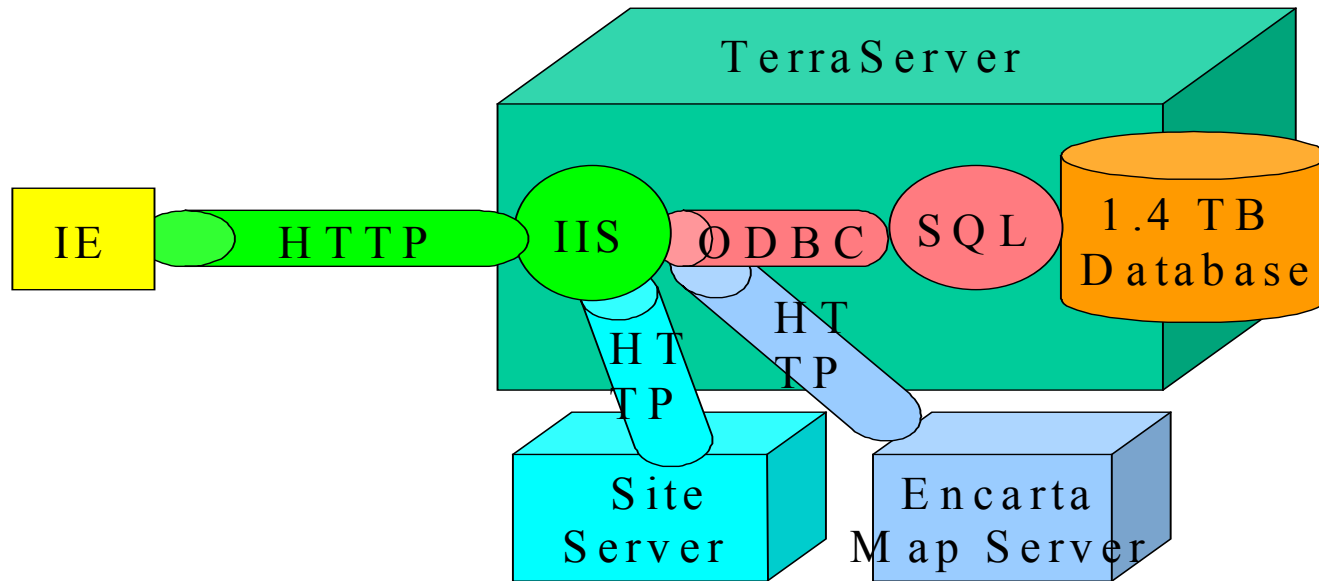
<http://terraservert.homeadvisor.msn.com/default.asp>



Terra-Server is a multi-media database that stores both classical text and numeric data, and also stores multi-media data

Server fills four larger cabinets: one for the Digital Alpha processors, and three cabinets for the 324 disks

Hardware and software



SQL database

The SQL database has several components:

- **Gazetteer:** The *Encarta World Atlas Gazetteer* has over a million entries describing most places on earth. All these records are stored and indexed by Microsoft SQL Server. Stored procedures to look up these names and produce an HTML page describing the top 10 "hits", with hot-links to the images if they are in the Terra-Server.

SQL Components

- **Map Control:** The *Encarta World Atlas* group implemented a map control for us and for other Microsoft groups (e.g., Expedia™ and Sidewalk™). This server program, given the corners of a rectangle and an altitude generates the view of the earth inside that rectangle. It generates a GIF image that is downloaded to the Java-based applet in the client browser. The map server runs on a Digital Prioris computer next to the Terra-Server.

components

- **The Reference Frame:** Terra Server uses the latitude and longitude coordinate system, potentially with centimeter accuracy, to geolocate objects. All the themes are represented in this coordinate system. After several false starts, Terra-Server settled on a simple latitude-longitude-tiling scheme. When indexing, it use the Z-transform (interleave latitude and longitude bits) to give a single clustering key for each spot on earth. All the spatial indices are based on this scheme.

components

- **Terra Server uses Sphinx:** Terra-Server uses the next version of Microsoft SQL Server, code-named Sphinx. This version supports larger page sizes, has better support for multi-media, supports parallelism within queries, parallel load, backup and restore utilities, and supports much larger databases. Terra-Server has been a good alpha test for Sphinx.

Other commercial DB clusters

- DB2 for LINUX
- Oracle9i on Linux