



Chapter 17: Database System Architectures

Database System Concepts, 6th Ed.

- Chapter 1: Introduction
- **Part 1: Relational databases**
 - Chapter 2: Introduction to the Relational Model
 - Chapter 3: Introduction to SQL
 - Chapter 4: Intermediate SQL
 - Chapter 5: Advanced SQL
 - Chapter 6: Formal Relational Query Languages
- **Part 2: Database Design**
 - Chapter 7: Database Design: The E-R Approach
 - Chapter 8: Relational Database Design
 - Chapter 9: Application Design
- **Part 3: Data storage and querying**
 - Chapter 10: Storage and File Structure
 - Chapter 11: Indexing and Hashing
 - Chapter 12: Query Processing
 - Chapter 13: Query Optimization
- **Part 4: Transaction management**
 - Chapter 14: Transactions
 - Chapter 15: Concurrency control
 - Chapter 16: Recovery System
- **Part 5: System Architecture**
 - [Chapter 17: Database System Architectures](#)
 - Chapter 18: Parallel Databases
 - Chapter 19: Distributed Databases
- **Part 6: Data Warehousing, Mining, and IR**
 - Chapter 20: Data Mining
 - Chapter 21: Information Retrieval
- **Part 7: Specialty Databases**
 - Chapter 22: Object-Based Databases
 - Chapter 23: XML
- **Part 8: Advanced Topics**
 - Chapter 24: Advanced Application Development
 - Chapter 25: Advanced Data Types
 - Chapter 26: Advanced Transaction Processing
- **Part 9: Case studies**
 - Chapter 27: PostgreSQL
 - Chapter 28: Oracle
 - Chapter 29: IBM DB2 Universal Database
 - Chapter 30: Microsoft SQL Server
- **Online Appendices**
 - Appendix A: Detailed University Schema
 - Appendix B: Advanced Relational Database Model
 - Appendix C: Other Relational Query Languages
 - Appendix D: Network Model
 - Appendix E: Hierarchical Model



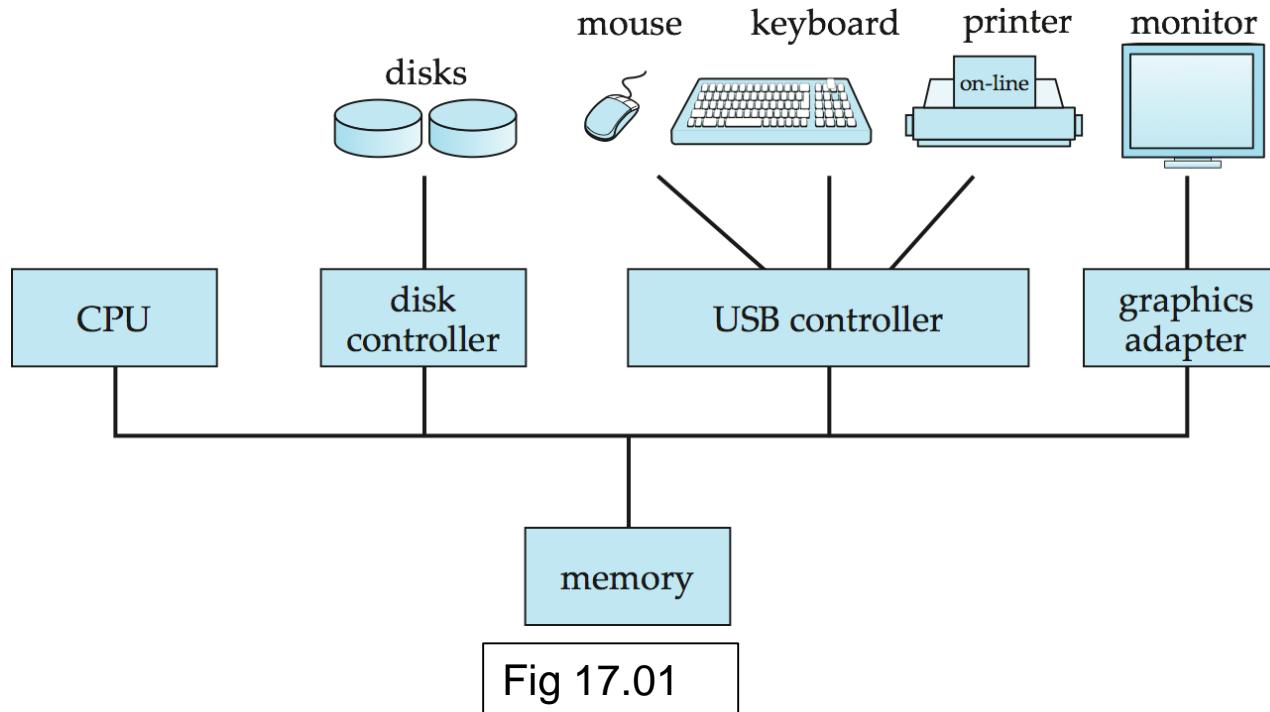
Chapter 17: Database System Architectures

- 17.1 Centralized and Client-Server Systems
- 17.2 Server System Architectures
- 17.3 Parallel Systems
- 17.4 Distributed Systems
- 17.5 Network Types



Centralized Systems

- Run on a single computer system and do not interact with other computer systems.
- **General-purpose computer system:** one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- **Single-user system (e.g., personal computer or workstation):** desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
- **Multi-user system:** more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system via terminals. Often called server systems.

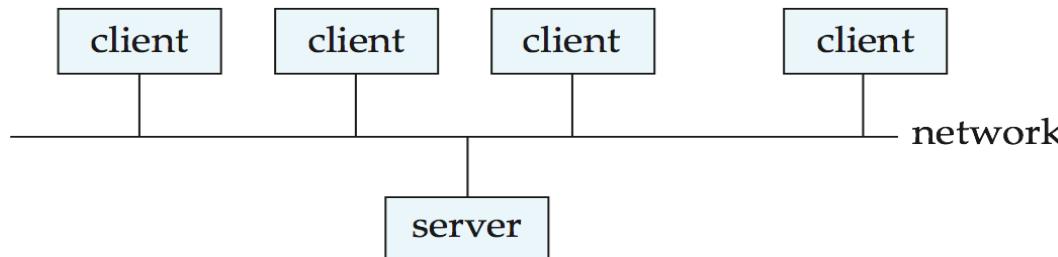




Client-Server Systems [1/2]

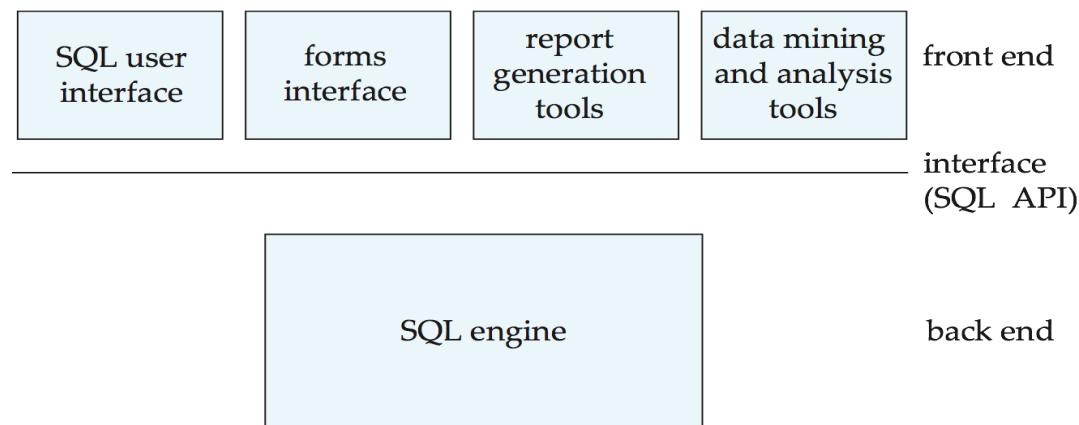
- Server systems satisfy requests generated at m client systems, whose general structure is shown below:

Fig 17.02



- Database functionality can be divided into:
 - **Back-end**: manages access structures, query evaluation & optimization, CC and recovery.
 - **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical UI facilities.
- The interface between the front-end and the back-end is through **SQL** or through **an application program interface (ex. CLI)**

Fig 17.03





Client-Server Systems [2/2]

- Client-Server System의 시작동기: Down Sizing
- Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
 - better functionality for the cost
 - flexibility in locating resources and expanding facilities
 - better user interfaces
 - easier maintenance
- Server systems can be broadly categorized into two kinds:
 - **transaction servers** which are widely used in relational database systems
 - **data servers**, used in object-oriented database systems

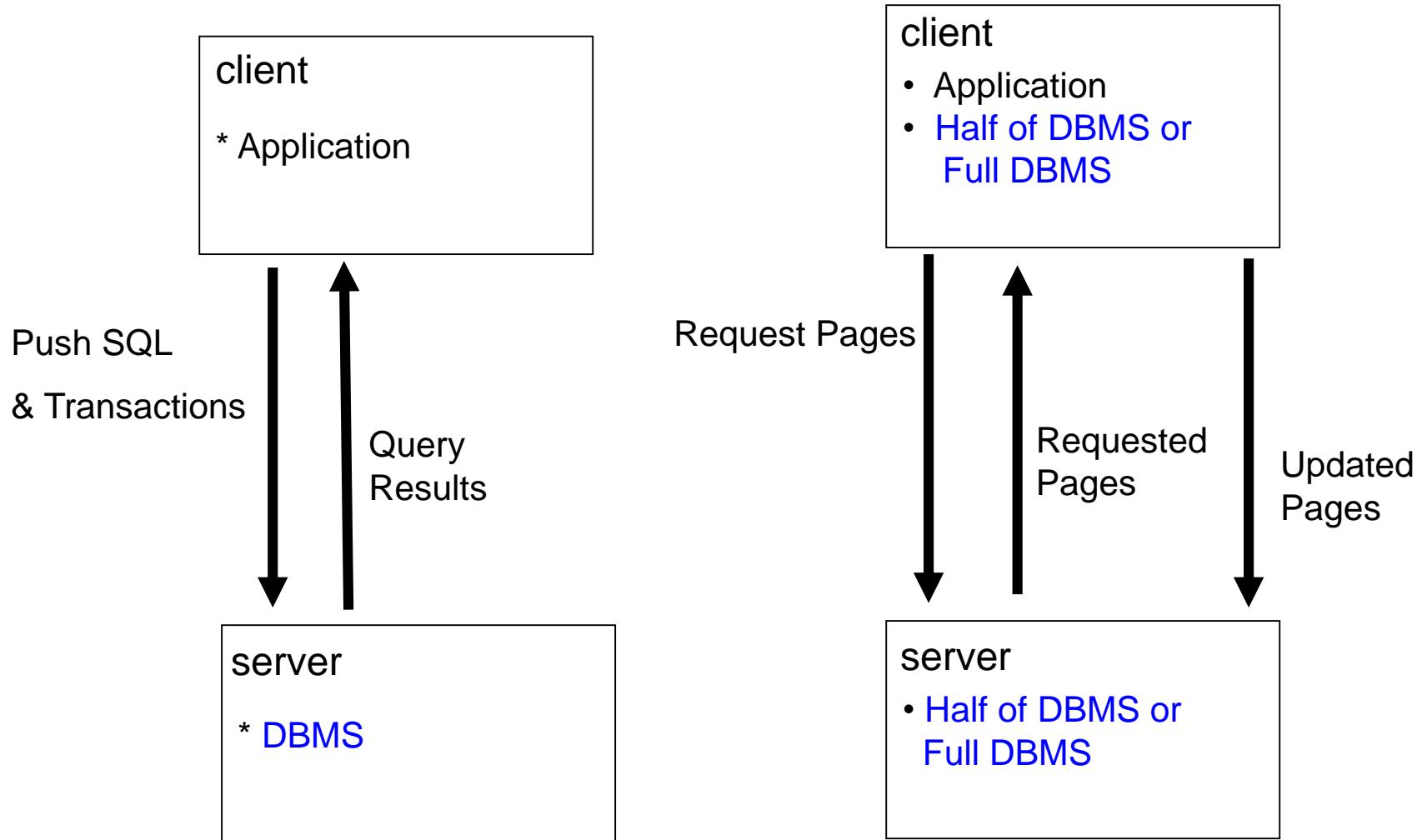


Chapter 17: Database System Architectures

- 17.1 Centralized and Client-Server Systems
- 17.2 Server System Architectures
- 17.3 Parallel Systems
- 17.4 Distributed Systems
- 17.5 Network Types



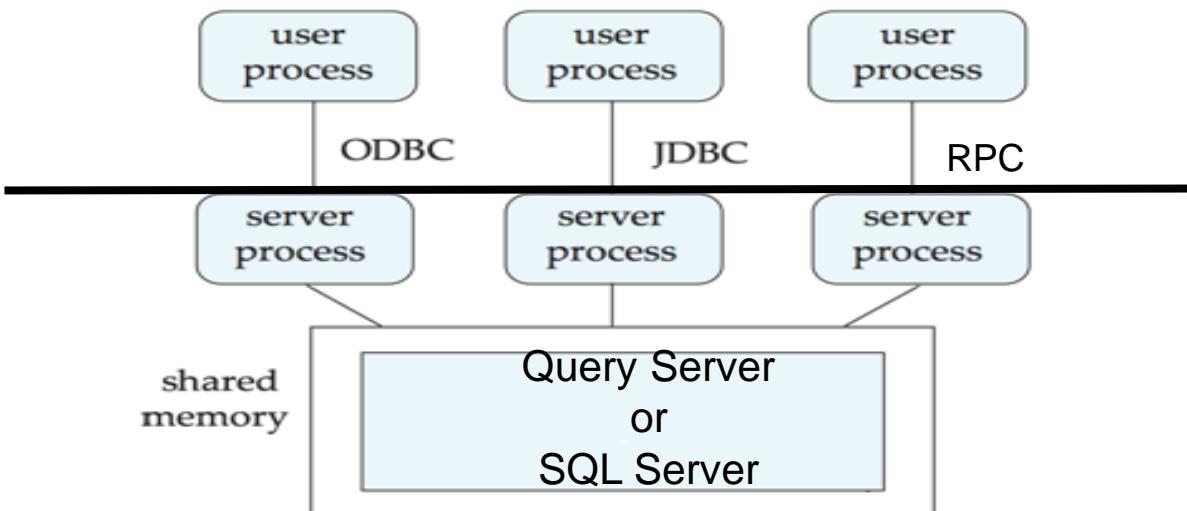
Transaction Server vs Data Server





Transaction Servers

- Also called **query server** systems or **SQL server** systems
 - clients send requests to the server system where the transactions are executed, and results are shipped back to the client
- Requests specified in SQL, and communicated to the server through a *remote procedure call (RPC)* mechanism
 - Transactional RPC allows many RPC calls to collectively form a transaction
 - *Open Database Connectivity (ODBC)* is a C API standard from Microsoft for connecting to a server, sending SQL requests, and receiving results
 - **JDBC standard** similar to ODBC, for Java





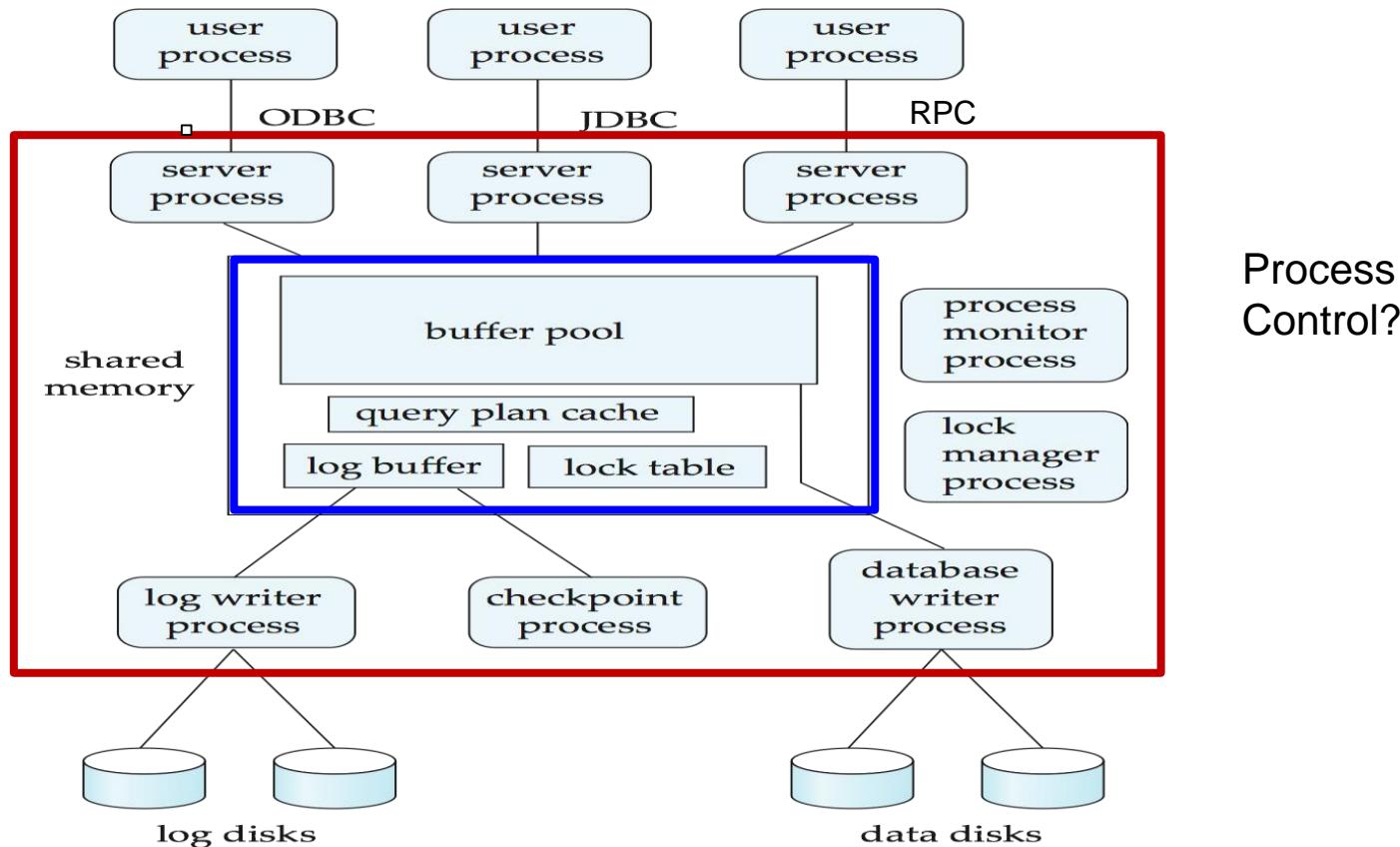
Transaction Server Processes [1/3]

- A typical transaction server consists of **multiple processes** accessing data in shared memory
- **Server processes**
 - Receive user queries (transactions), execute them & send results back
 - Typically **multiple multithreaded** server processes allowing a single process to execute several user queries concurrently
- **Lock manager process:** More on this later
- **Database writer process:** Output modified buffer blocks to disks continually
- **Log writer process**
 - Server processes simply add log records to log record buffer
 - Log writer process outputs log records to stable storage.
- **Checkpoint process:** Performs periodic checkpoints
- **Process monitor process**
 - Monitors other processes, and takes recovery actions if any of the other processes fail
 - ▶ E.g. aborting any transactions being executed by a server process and restarting it



Transaction Server Processes [2/3]

Fig 17.04



Process
Control?

- Shared memory contains shared data
 - Buffer pool // Lock table // Log buffer // Cached query plans
 - All database processes can access shared memory
- To ensure that no two processes are accessing the same data structure at the same time, databases systems implement mutual exclusion
 - OS semaphores or Atomic instructions such as test-and-set



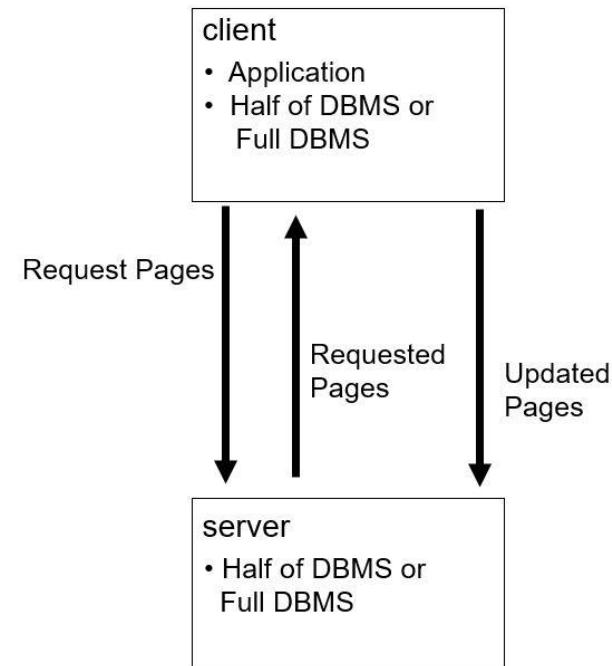
Transaction Server Processes [3/3]

- Lock Manager의 역할: Lock의 허가, Deadlock의 해결
- Sending requests to lock manager process creates **severe overhead** of interprocess communication for lock request/grant (Lock manager에 message passing는 큰부담)
- Instead **each DB server process operates directly on the lock table** (Section 15.1)
 - Mutual exclusion ensured on the lock table using semaphores, or more commonly, atomic instructions
 - ▶ If a lock can be obtained, the lock table is updated directly in shared memory
 - ▶ If a lock cannot be immediately obtained, a lock request is noted in the lock table and the process (or thread) then waits for lock to be granted
 - ▶ When a lock is released, releasing process updates lock table to record release of lock, as well as grant of lock to waiting requests (if any)
 - Process/Thread waiting for lock may either:
 - ▶ Continually scan lock table to check for lock grant, or
 - ▶ Use OS semaphore mechanism to wait on a semaphore
 - Semaphore identifier is recorded in the lock table
 - When a lock is granted, the releasing process signals the semaphore to tell the waiting process/thread to proceed
- Lock manager process still used for deadlock detection



Data Servers [1/3]

- Used in high-speed LANs, in cases where
 - The client machines are comparable in processing power to the server machine
 - The tasks to be executed in the client machines are **compute intensive**
- **Data** are shipped to **client machines where processing is performed**, and then shipped **results** back to the server machine.
- This architecture requires **full back-end functionality at the clients**
- Used in **many object-oriented database systems**
- Issues:
 - Page-Shipping versus Item-Shipping
 - Locking
 - Data Caching
 - Lock Caching





Data Servers [2/3]

■ Page-Shipping versus Item-Shipping

- Smaller unit of shipping \Rightarrow more messages
- Worth **prefetching** related items along with requested item
- Page shipping can be thought of as a form of prefetching

■ Locking

- **Overhead** of requesting and getting locks from server is high due to message delays
 - ▶ With item shipping, locks are granted on **requested and prefetched items**
 - ▶ With page shipping, locks are granted on **whole page**
- **Locks on a prefetched item** can be **called back** by the server, and returned by client transaction if the prefetched item has not been used
- **Locks on the page** can be **deescalated** to locks on items in the page when there are lock conflicts
 - ▶ Locks on unused items can then be returned to server



Data Servers

[3/3]

■ Data Caching

- Data can be cached at client even in between transactions
- But need to check that data is up-to-date before it is used (**cache coherency**)
- Check can be done when requesting lock on data item

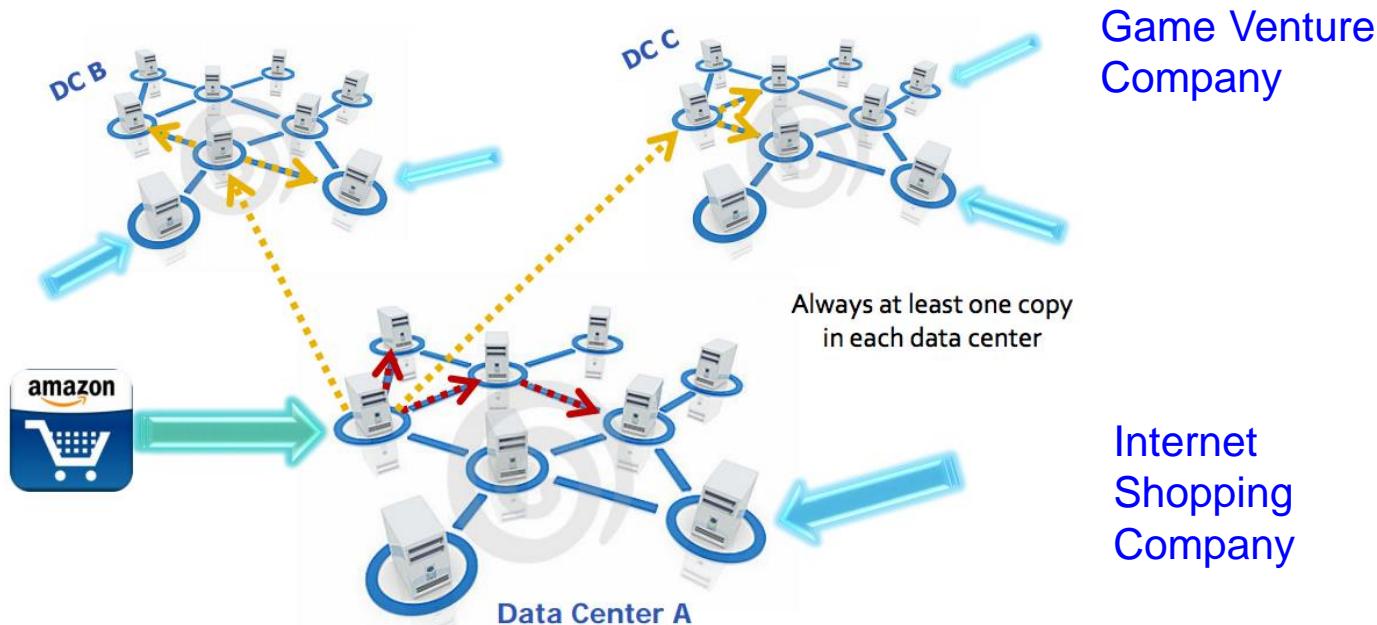
■ Lock Caching

- Locks can be retained by client system even in between transactions
- Transactions can acquire cached locks locally, without contacting server
 - ▶ Reduce communication overhead
- Server **calls back** locks from clients when it receives conflicting lock request
 - ▶ Client returns lock once no local transaction is using it
- Similar to deescalation, but across transactions



Cloud-based Servers

- Third party cloud vendors provide a collection of machines
- Not real machines, but software simulated machines (called virtual machines)
 - Add/Release machines as needed
 - Software define architecture (SDA)
 - Software defined storage (SDS)
- Various degree of cloud services
 - Providing Just raw storage and virtual machines
 - Providing varying degree of control over data placement and replication
 - Providing even DBMS service





Chapter 17: Database System Architectures

- 17.1 Centralized and Client-Server Systems
- 17.2 Server System Architectures
- 17.3 Parallel Systems
- 17.4 Distributed Systems
- 17.5 Network Types



Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network
 - A **coarse-grain parallel** machine consists of a small number of powerful processors
 - A **fine grain parallel** or **massively parallel** machine utilizes thousands of smaller processors
- Two main performance measures:
 - **throughput** --- the number of tasks that can be completed in a given time interval
 - **response time** --- the amount of time it takes to complete a single task from the time it is submitted



Speed-Up and Scale-Up

- **Speedup:** a fixed-sized problem executing on a small system is given to a N -times larger system

$\text{speedup} = \text{small system elapsed time} / \text{large system elapsed time}$

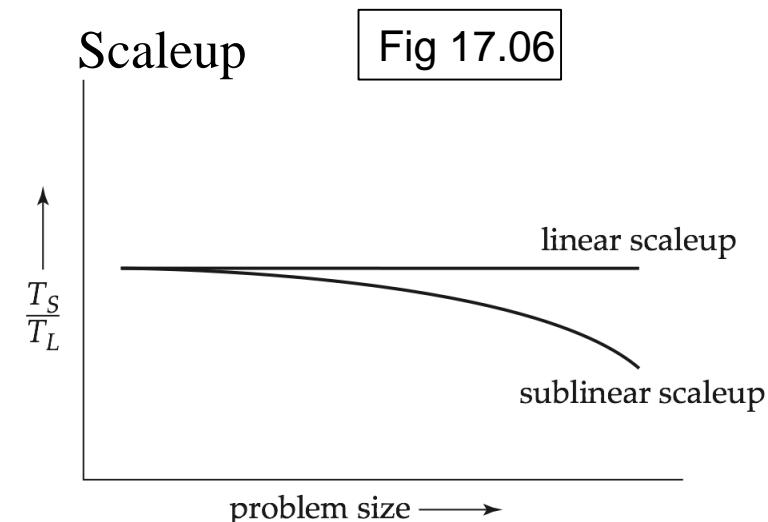
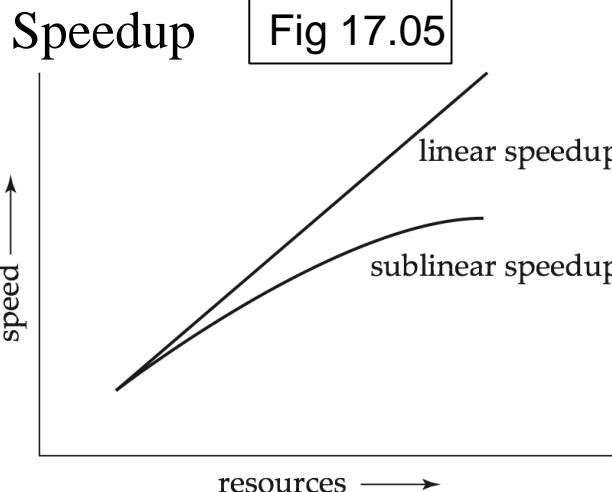
- Speedup is **linear** if equation equals N

- **Scaleup:** increase the size of both the problem and the system

- N -times larger system used to perform N -times larger job

$\text{scaleup} = \text{small system small problem elapsed time } (T_S) / \text{big system big problem elapsed time } (T_L)$

- Scale up is **linear** if equation equals 1





Batch and Transaction Scaleup

■ Batch scaleup:

- A single large job
 - ▶ typical of most database queries and scientific simulation
- Use an N -times larger computer on N -times larger problem

■ Transaction scaleup:

- Numerous small queries submitted by independent users to a shared database
 - ▶ typical transaction processing and timesharing systems
- N -times as many users submitting requests (hence, N -times as many requests) to an N -times larger database, on an N -times larger computer
- Well-suited to parallel execution



Factors Limiting Speedup and Scaleup

Speedup and scaleup of parallel processing are often **sublinear** due to:

■ **Startup costs**

- Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high

■ **Interference**

- Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other, thus spending time waiting on other processes, rather than performing useful work

■ **Skew**

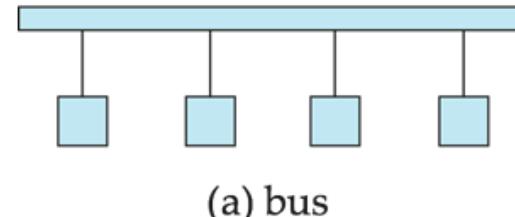
- Increasing the degree of parallelism increases the variance in service times of parallelly executing tasks
- Overall execution time determined by **slowest** of parallelly executing tasks



Interconnection Network Architectures

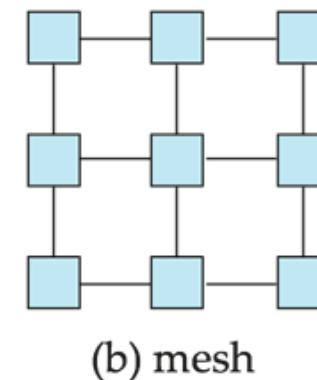
■ Bus

- System components send data on and receive data from a single bus
- Does not scale well with increasing parallelism



■ Mesh

- Components are arranged as nodes in a grid, and each component is connected to all adjacent components
- Communication links grow with growing number of components, and so scales better
- But may require $2\sqrt{n}$ hops to send message to a node (or \sqrt{n} with wraparound connections at edge of grid)



■ Hypercube

- Components are numbered in binary
- Components are connected to one another if their binary representations differ in exactly one bit
- N components are connected to $\log(n)$ other components and can reach each other via at most $\log(n)$ links; reduces communication delays

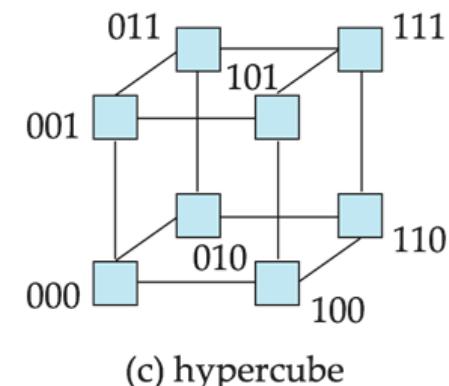
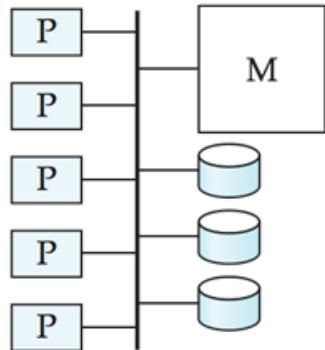


Fig 17.07

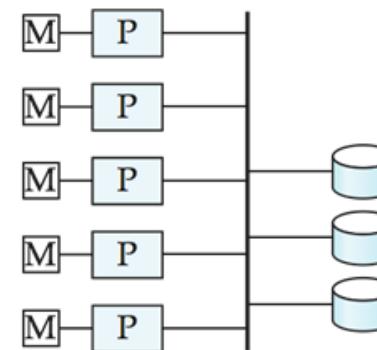


Parallel Database Architectures

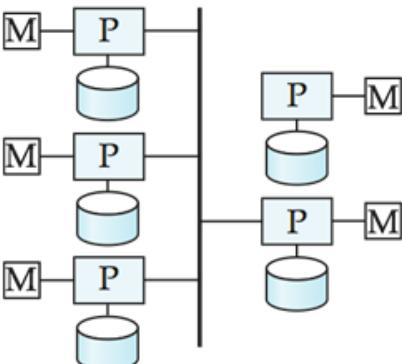
- **Shared memory** -- processors share a common memory
- **Shared disk** -- processors share a common disk, sometimes called **clusters**
- **Shared nothing** -- processors share **neither** a common memory **nor** common disk
- **Hierarchical** -- hybrid of the above architectures



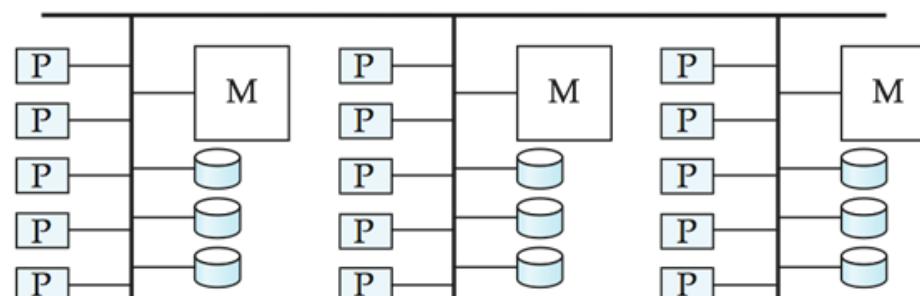
(a) shared memory



(b) shared disk



(c) shared nothing



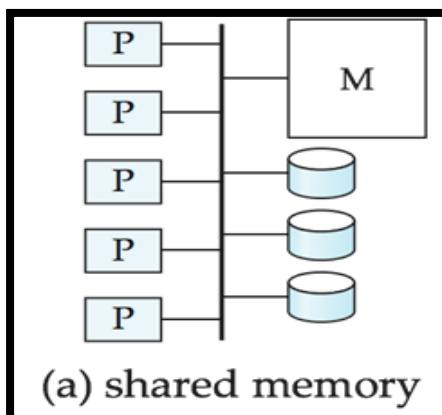
(d) hierarchical

Fig 17.08



Shared-Memory Parallel DB

- Processors and disks have access to a **common memory**, typically via a bus or through an interconnection network
 - **SMP (Symmetric Multi Processing)** 방식
 - Each processor has its own cache → **cache coherence problem**
 - Widely used for **lower degrees of parallelism** (4 to 8)
- **Advantage**
 - Extremely efficient communication between processors
 - Data in shared memory can be accessed by any processor without having to move it using software
- **Downside**
 - Architecture is **not scalable beyond 32 or 64 processors** since the bus or the interconnection network becomes a bottleneck



Multi-processor Server Systems based on Intel® Xeon® processors

MP SuperServer® Solution

TU 8-Way: SYS-7088B-TR4FT

- Supports 8 Intel® Xeon® E7-8800 v4/v3 product families (up to 24 cores per CPU)
- Up to 24TB DDR4 in 192 DIMM slots
- Up to 15 PCI-E 3.0 slots, U.2 NVMe support option available
- Up to 12x 2.5" hot-swap SAS3/SATA3 HDD/SSD drive bays
- 4x 10GbE-T LAN ports (via SIO)
- 2 SATA3 SATA DOM + 2 SATA2 M.2 for internal storage device connection via C602J onboard, Onboard PMI 2.0
- 1600W Hot-swap (N+1) redundant Titanium Level power supplies

8-Way 4-Way

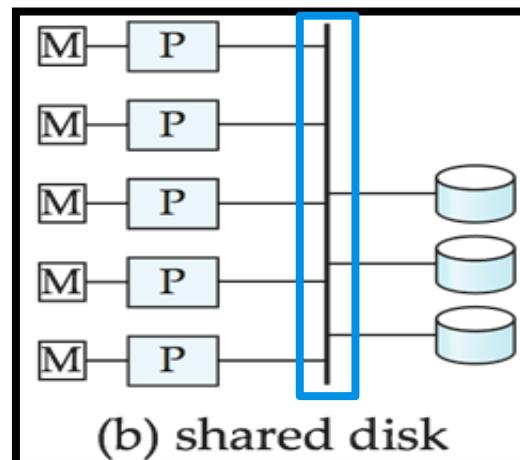
CPU1 CPU8
8 CPU Modules: CPU1 to CPU8

For more detail >



Shared-Disk Parallel DB

- All processors with private memories can directly access **all disks** via an interconnection network
 - The memory bus is not a bottleneck
 - Architecture provides a degree of **fault-tolerance** — if a processor fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors
 - Mainframe 기반 parallel processor
 - IBM Sysplex with IBM DB2 and IMS
 - DEC clusters (now part of Compaq) running Oracle RDBMS
- **Downside:** bottleneck now occurs at **interconnection** to the disk subsystem
- Shared-disk systems can scale to **a somewhat larger number of processors**, but communication between processors is slower

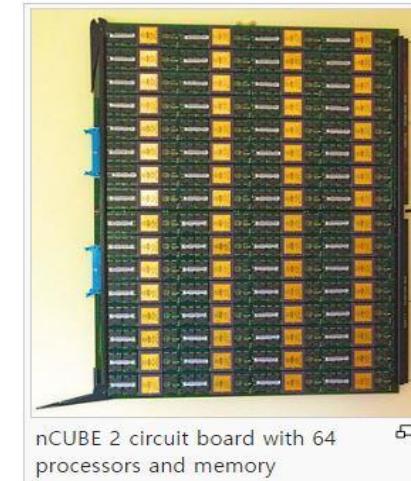
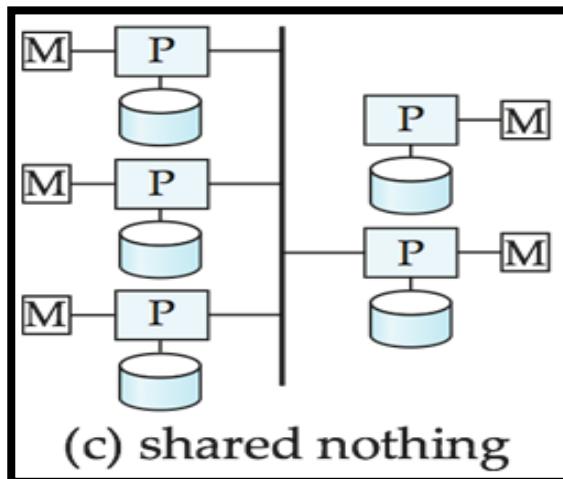


(b) shared disk



Shared-Nothing Parallel DB

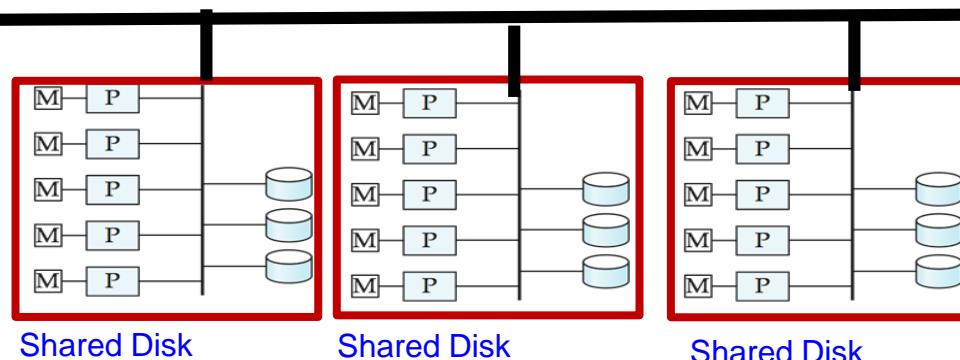
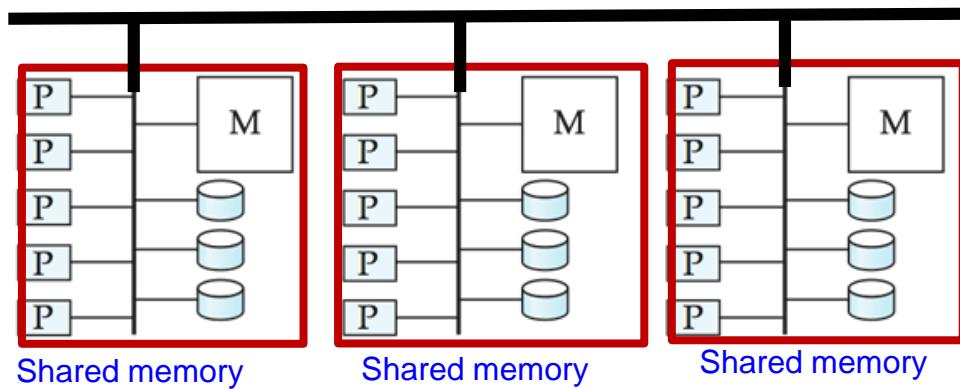
- Node consists of a processor, memory, and one or more disks
 - Processors at one node communicate with another processor at another node using an interconnection network
 - A node functions as **the server for the data on the disk the node owns**
- Examples: Teradata, Tandem, Oracle n-CUBE
- Data in local disks (and local memory) cannot be accessed through interconnection network, thereby minimizing the interference of resource sharing
- Shared-nothing multiprocessors can be scaled up to **thousands of processors without interference**
 - MPP (Massively Parallel Processing) 방식
- Main drawback: cost of communication and non-local disk access
 - Sending data involves software interaction at both ends





Hierarchical Parallel DB

- Combines characteristics of shared-memory, shared-disk, and shared-nothing
 - Top level is a **shared-nothing architecture** – nodes connected by an interconnection network, and do not share disks or memory with each other
 - Each node could be a **shared-memory system** with a few processors
 - Alternatively, each node could be a **shared-disk system**, and each of the systems sharing a set of disks could be a shared-memory system



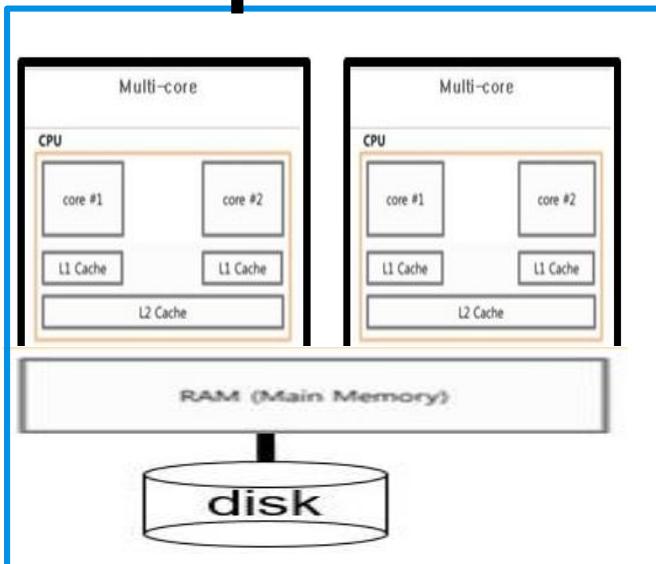


대세! PC Cluster and Server Cluster

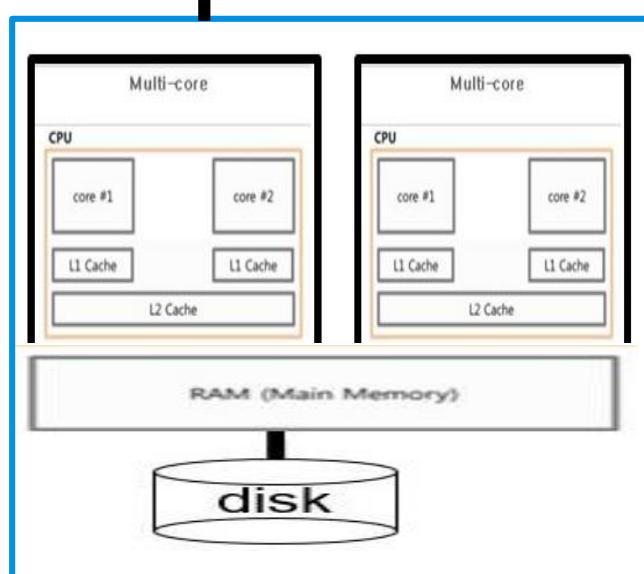
■ Multiprocessors vs. Multicores

- Multiprocessors have private cache hierarchy (on chip)
- Multicores have shared L2 (on chip)

- Total 12 node cluster
- Each node
 - Intel Core i5-2400 SandyBridge (3.1GHz, quad-core)
 - 4GB memory / 4TB HDD



Node

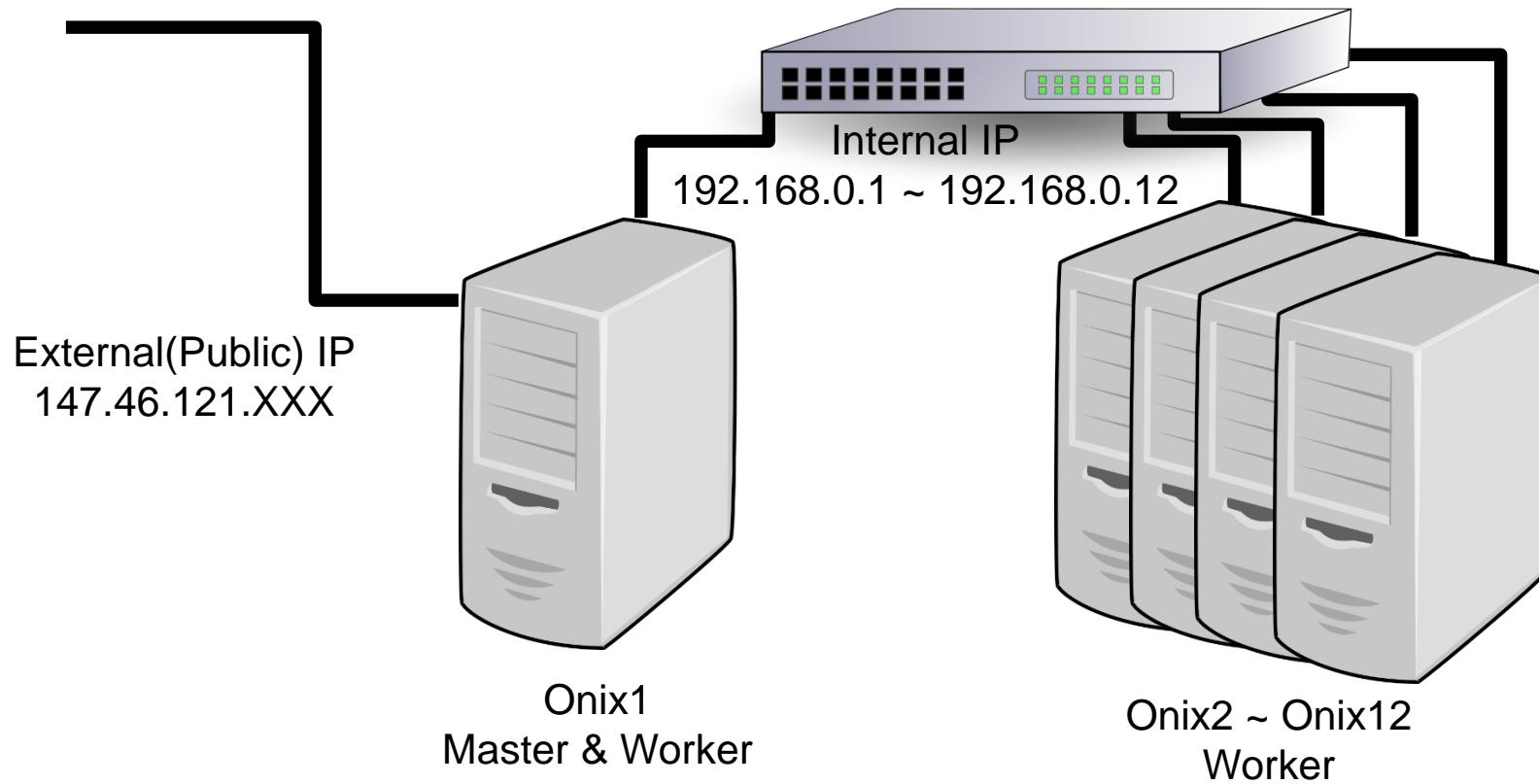


Node



Sample Cluster Environment

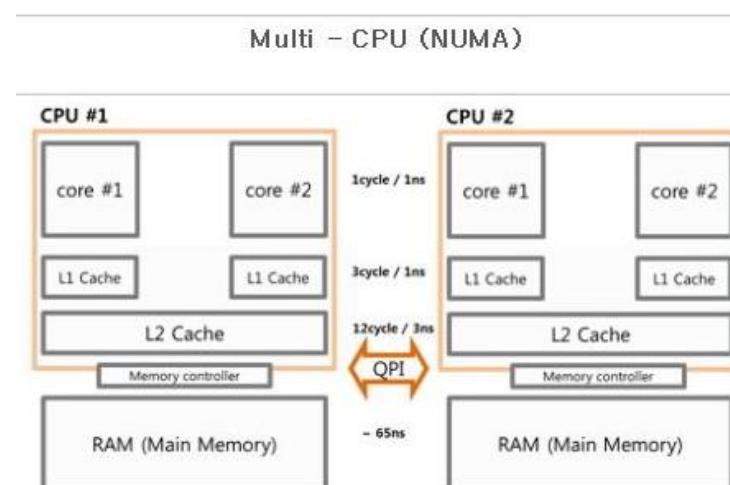
- 12 Clusters
 - 1 Master & 12 Workers
 - Master node works as a worker node (slave node)



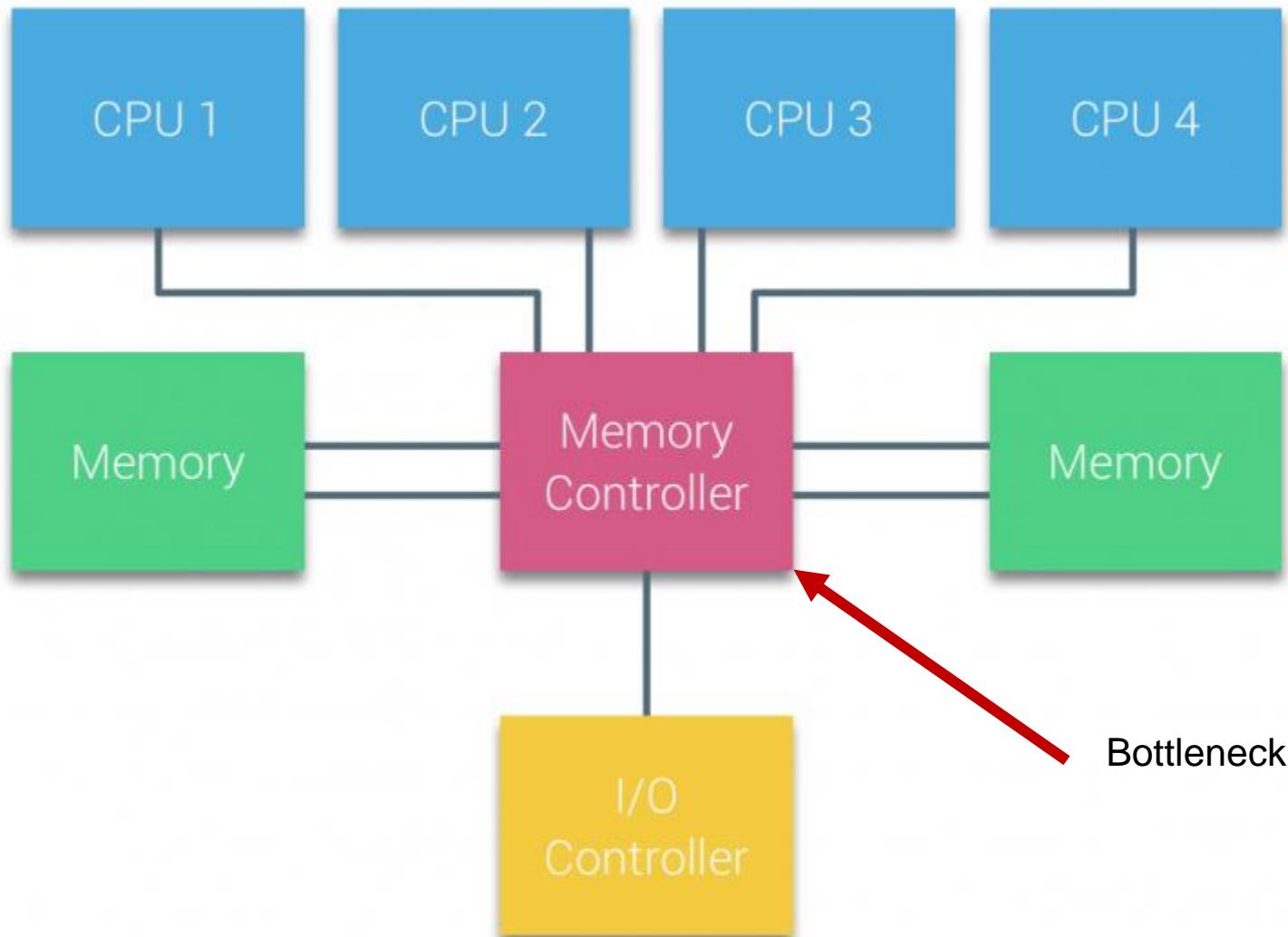


Difficult Parallel Programming!

- Shared Memory Parallel Architecture의 문제점 → Memory bottleneck
- Shared Nothing Parallel Architecture의 문제점 → Difficult Parallel Programming
- Parallel Programming with **distributed virtual-memory on NUMA Capable Architecture**
 - **Distributed Virtual Memory**
 - **Virtual memory-mapping HW + Special SW**
 - **To view multiple disjoint memories as a single virtual memory**
 - **Non-uniform memory architecture (NUMA)**
 - 몇 개의 core들이 중간단계의 L2 cache를 공유하므로 **Memory access**를 대폭줄일수 있다 → 성능향상, 시스템확장성증대
- **Hadoop**
- **Spark**



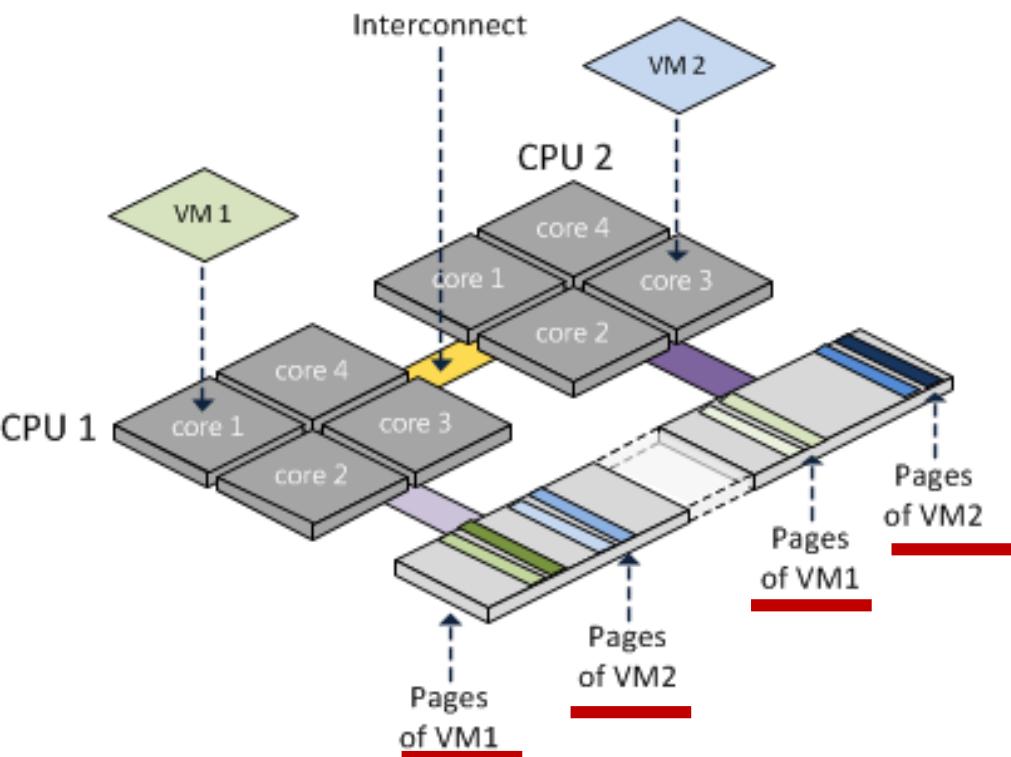
Typical UMA Structure (Uniform Memory Access)



<http://frankdenneman.nl/2016/07/07/numa-deep-dive-part-1-uma-numa/>

UMA

- Virtual machine usage

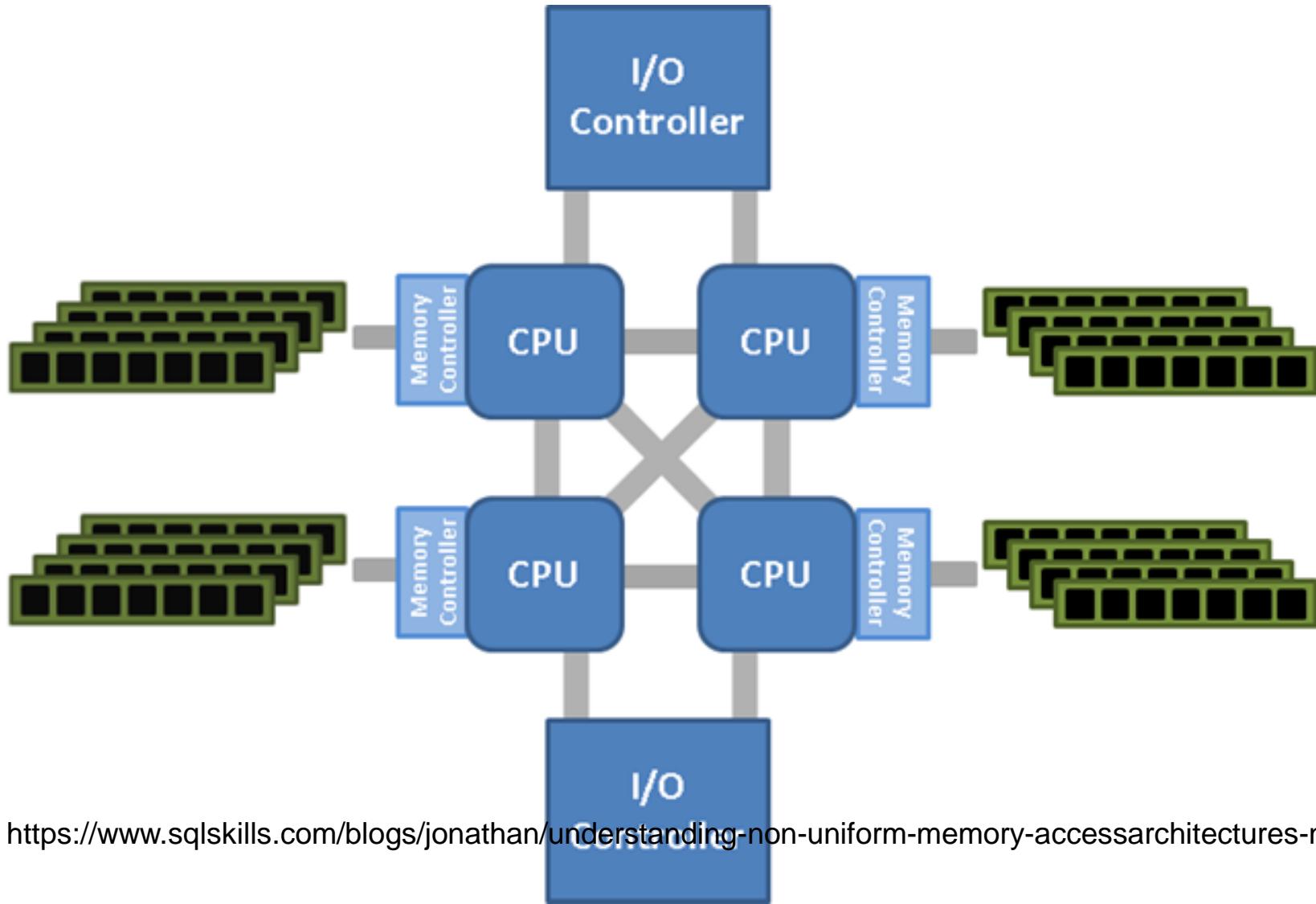


2 VM on UMA Structure

Vm1과 VM2가 각각 다른 CPU에 존재하나 메모리는 하나를 사용하므로 두 VM이 동시에 메모리를 접근하려면 한쪽이 기다려야 함

NUMA Structure (Non Uniform Memory Access)

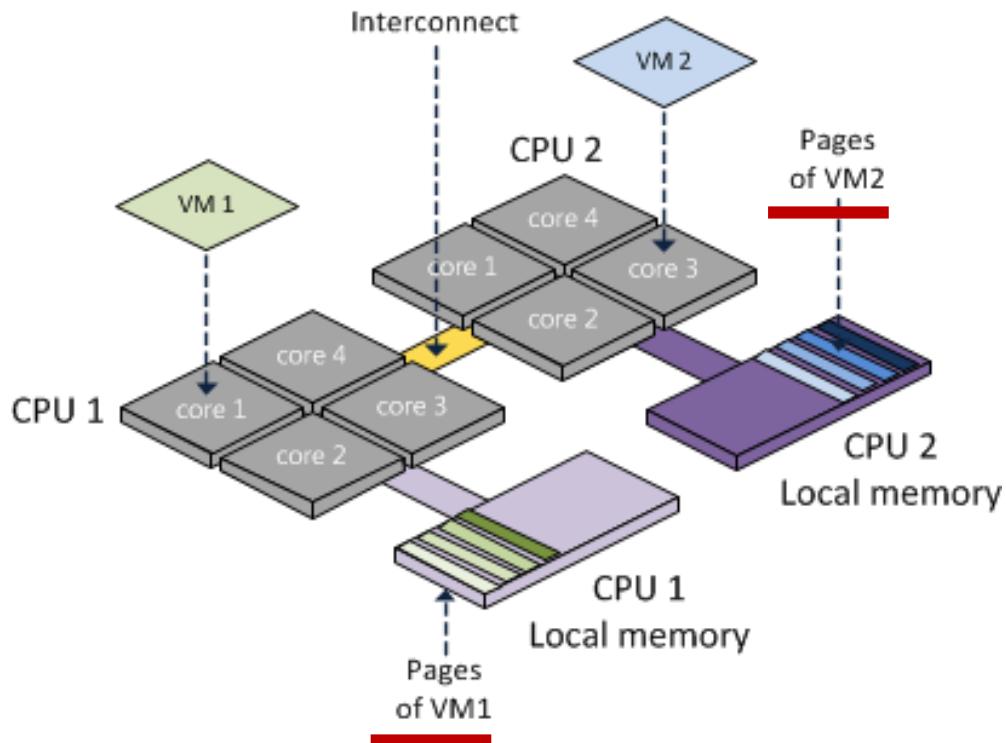
- Each CPU reserves its' own Memory & Controller



<https://www.sqlskills.com/blogs/jonathan/understanding-non-uniform-memory-accessarchitectures-numa/>

NUMA

- Virtual machine usage



2 VM on NUMA Structure

<http://frankdenneman.nl/2015/02/27/memory-deep-dive-numa-data-locality/>

Numa C code

- Check if NUMA is available
- Bind thread to specific core
- Not always NUMA is better
 - If remote memory access is often

```
#define _GNU_SOURCE
#include <numa.h>
#include <sched.h>
#include <stdio.h>
#include <pthread.h>

int main(int argc, const char **argv)
{
    int num_cpus = numa_num_task_cpus();
    printf("num cpus: %d\n", num_cpus);

    printf("numa available: %d\n", numa_available());
    numa_set_localalloc();

    struct bitmask *bm = numa_bitmask_alloc(num_cpus);
    for (int i=0; i<=numa_max_node(); ++i)
    {
        numa_node_to_cpus(i, bm);
        printf("numa node %d ", i);
        print_bitmask(bm);
        printf(" - %g GiB\n", numa_node_size(i, 0) / (1024.*1024*1024.));
    }
    numa_bitmask_free(bm);

#pragma omp parallel
{
    assert(omp_get_num_threads() == num_cpus);
    int tid = omp_get_thread_num();

    pin_to_core(tid);
    if(tid == 0)
        x = (char *) numa_alloc_local(array_size);

}
```



NUMA Coding

- UMA coding과 NUMA coding이 특별히 달라지기 보다는 thread를 어느 core에 binding하느냐에 따라 해당 thread는 release 되기 전까지 계속 동일한 core에서 수행
- NUMA를 disable시키는 것이 좋은 경우 (즉 UMA방식이 더 좋은 경우)
- 예를 들어 cpu0에서 main thread를 실행시켜 3개의 thread를 추가로 생성해 각각 cpu 1, cpu 2, cpu 3에 bind했다면, 이 때 3개의 생성된 thread가 main thread의 메모리를 참조할 경우 계속해서 cpu0의 메모리를 참조하게 되어 오버헤드가 발생
- 결론적으로 CPU Intense작업을 주로 할 경우에는 NUMA가 성능을 더 낼 수 있으나 Application에 따라 memory intensive할 경우 UMA가 더 좋을 수도 있습니다.



Chapter 17: Database System Architectures

- 17.1 Centralized and Client-Server Systems
- 17.2 Server System Architectures
- 17.3 Parallel Systems
- 17.4 Distributed Systems
- 17.5 Network Types



Distributed Systems

- Data spread over multiple machines (also referred to as **sites** or **nodes**)
- Network interconnects the machines
- Data shared by users on multiple machines

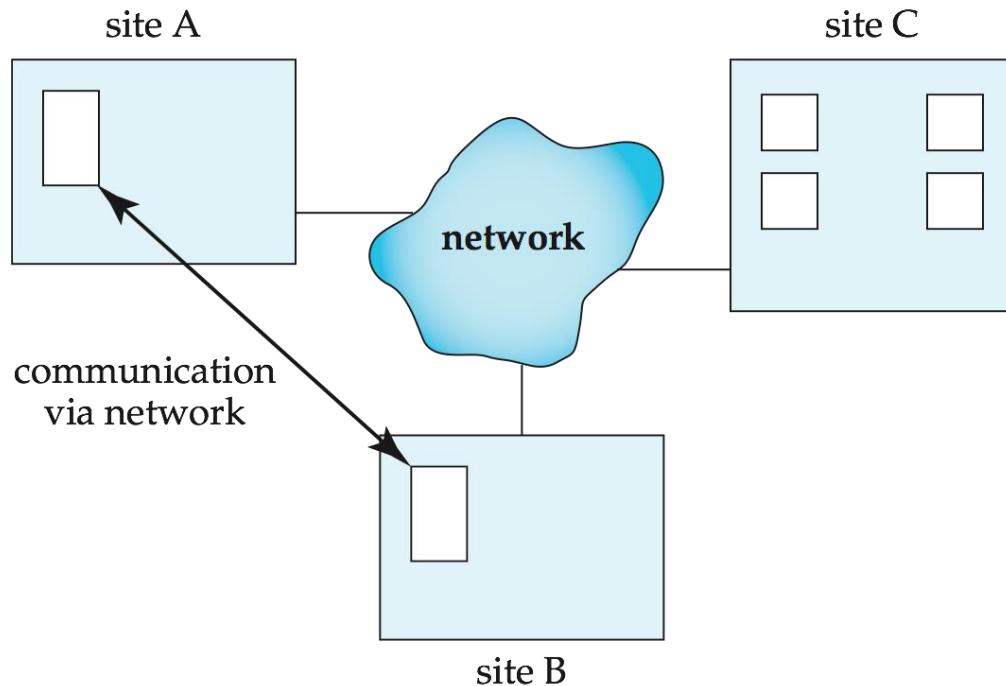


Fig 17.09



Distributed Databases

- Homogeneous distributed databases
 - Same software/schema on all sites, data may be partitioned among sites
 - Goal: provide a view of a single database, hiding details of distribution
- Heterogeneous distributed databases
 - Different software/schema on different sites
 - Goal: integrate existing databases to provide useful functionality
- Differentiate between *local* and *global* transactions
 - A local transaction accesses data in the *single* site at which the transaction was initiated
 - A global transaction either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites



Trade-offs in Distributed Systems

■ Advantages

- Sharing data
 - ▶ Users at one site able to access the data residing at some other sites
- Autonomy
 - ▶ Each site is able to retain a degree of control over data stored locally
- Higher system availability through redundancy
 - ▶ Data can be replicated at remote sites, and system can function even if a site fails
 - ▶ Primary site vs Back-up site

■ Disadvantages: added complexity for proper coordination among sites

- Software development cost
- Greater potential for bugs
- Increased processing overhead



Implementation Issues for Distributed Databases

- Atomicity needed even for transactions that update data at multiple sites
- The two-phase commit protocol (2PC) used to ensure atomicity (Section 19.4.1)
 - Basic idea: each site executes transaction till just before commit, and then leaves final decision to a coordinator
 - Each site must follow decision of coordinator: even if there is a failure while waiting for coordinator's decision
 - 2PC is not always appropriate
 - ▶ Other transaction models (persistent messaging, workflows), may be used
- Distributed concurrency control (and deadlock detection) required
- Distributed query processing: reducing communication overhead
- Replication of data items required for improving data availability
- Details of above in Chapter 19



Chapter 17: Database System Architectures

- 17.1 Centralized and Client-Server Systems
- 17.2 Server System Architectures
- 17.3 Parallel Systems
- 17.4 Distributed Systems
- 17.5 Network Types



Local-Area Network

- Local-area networks (LANs) – composed of processors that are distributed over small geographical areas, such as a single building or a few adjacent buildings

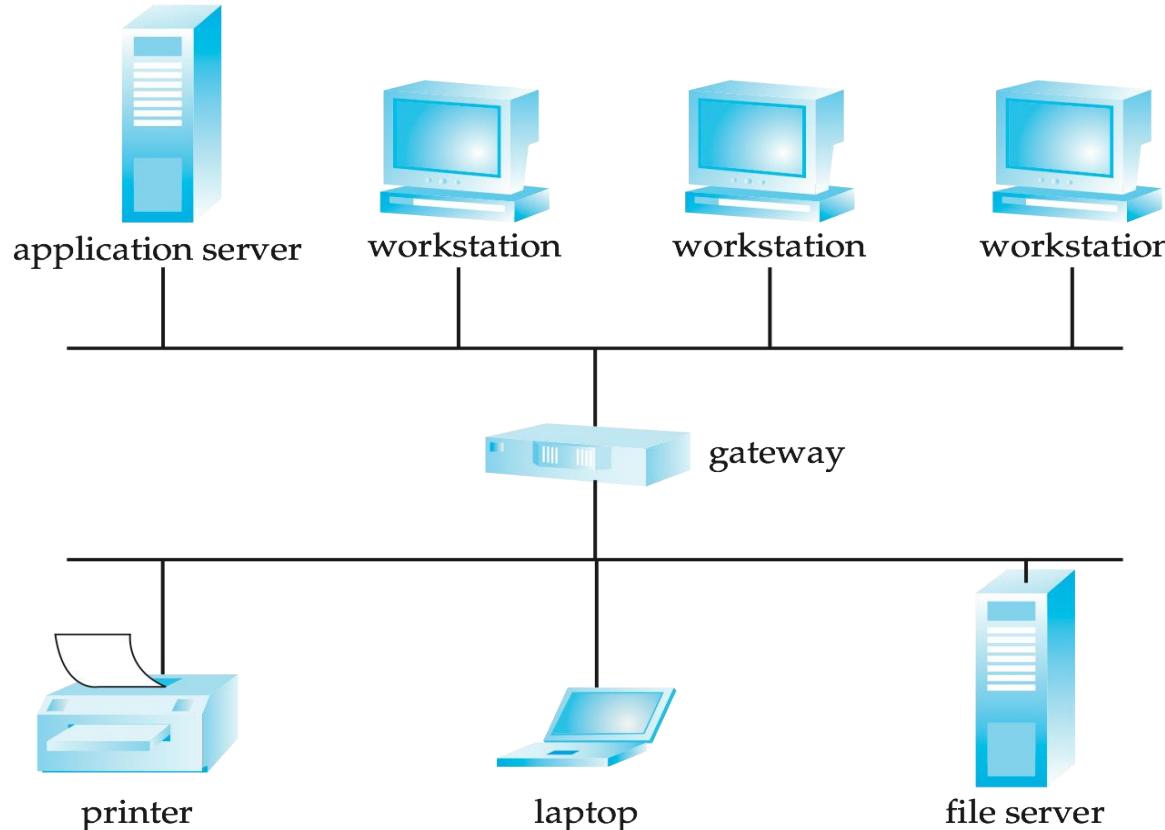
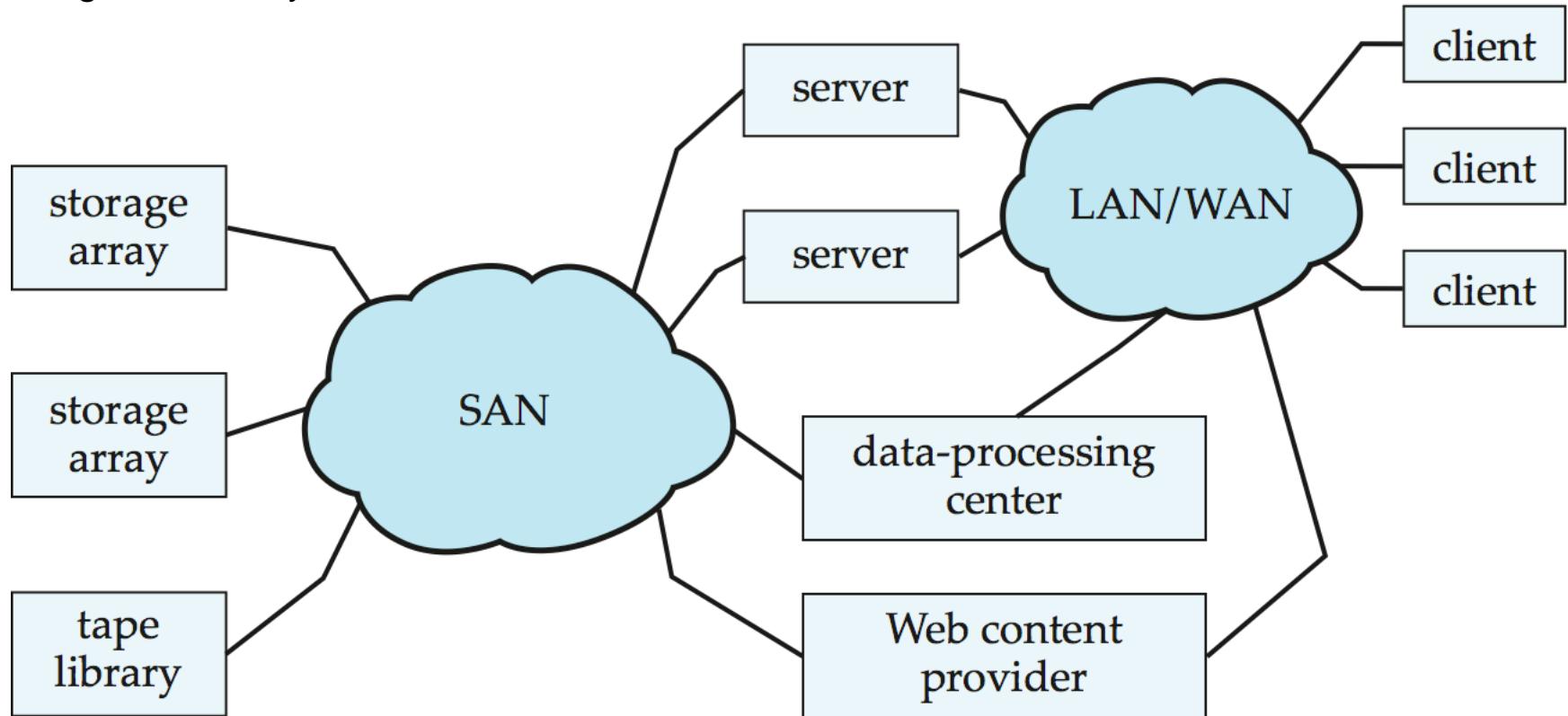


Fig 17.10



Storage-Area Network

- To build large-scale shared-disk systems
 - Scalability by adding more computer
 - High availability



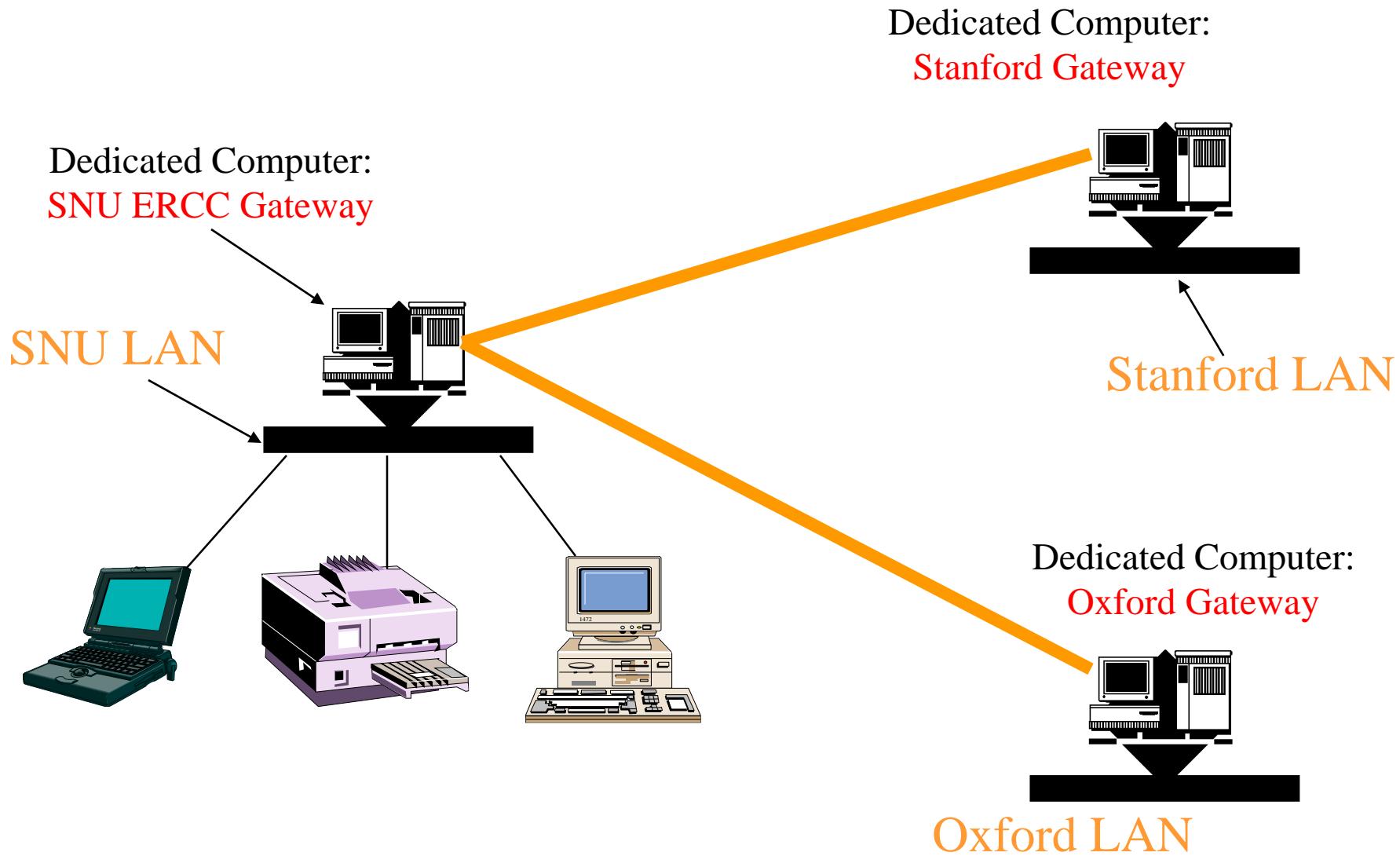


Wide Area Networks

- **Wide-area networks (WANs)** – composed of processors distributed over a large geographical area
 - *Discontinuous Connection WANs*, such as those based on periodic dial-up (e.g. using UUCP), that are connected only for part of the time
 - *Continuous Connection WANs*, such as the Internet, where hosts are connected to the network at all times
- **WANs with continuous connection** are needed for distributed database systems
- **WANs with discontinuous connection**
 - **Mobile wireless connection**
 - Earlier groupware applications such as Lotus Notes can work on this
 - Data is replicated, so updates are propagated to replicas periodically
 - No global locking is possible, and copies of data may be independently updated
 - Non-serializable executions can thus result
 - ▶ Conflicting updates may have to be detected, and resolved in an application dependent manner



WAN (Wide Area Network): Example





So What?

- Client-Server System 특징
 - Transactional Server, Data Server, Cloud Server의 특징
 - Parallel HW의 성질, Distributed Computing 환경의 성질
-
- Parallel HW의 성질의 이용해서 DBMS 내부에서 어떻게 이용할것인가?
 - Chapter 18
 - Distributed Computing 환경을 고려하여 DBMS는 무엇을 지원해야 하나?
 - Chapter 19
-
- Bonus Project (Optional Team Project: 3%)
 - NUMA를 직관적으로 설명하는 PPT
 - 기술현황, 제품현황, Application 개발 현황
 - NUMA위에서 Programming할때의 핵심 issue
 - NUMA위에서 작동하는 DBMS를 만들때의 주요 issue