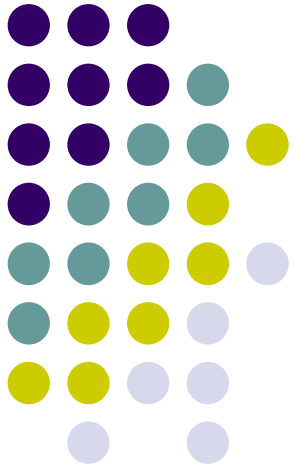
FINAL TOUCH

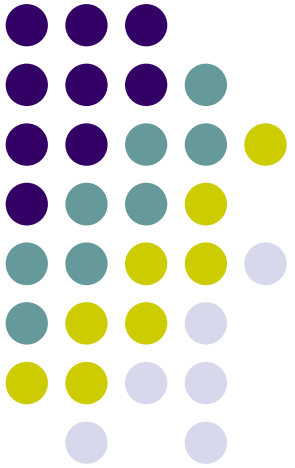
Chapter 15: More Topics in Operating Systems

- Distributed computing
- Virtualization



Distributed Computing

- Background
- Client/server computing
- Middleware
- Remote procedure calls
- Clusters



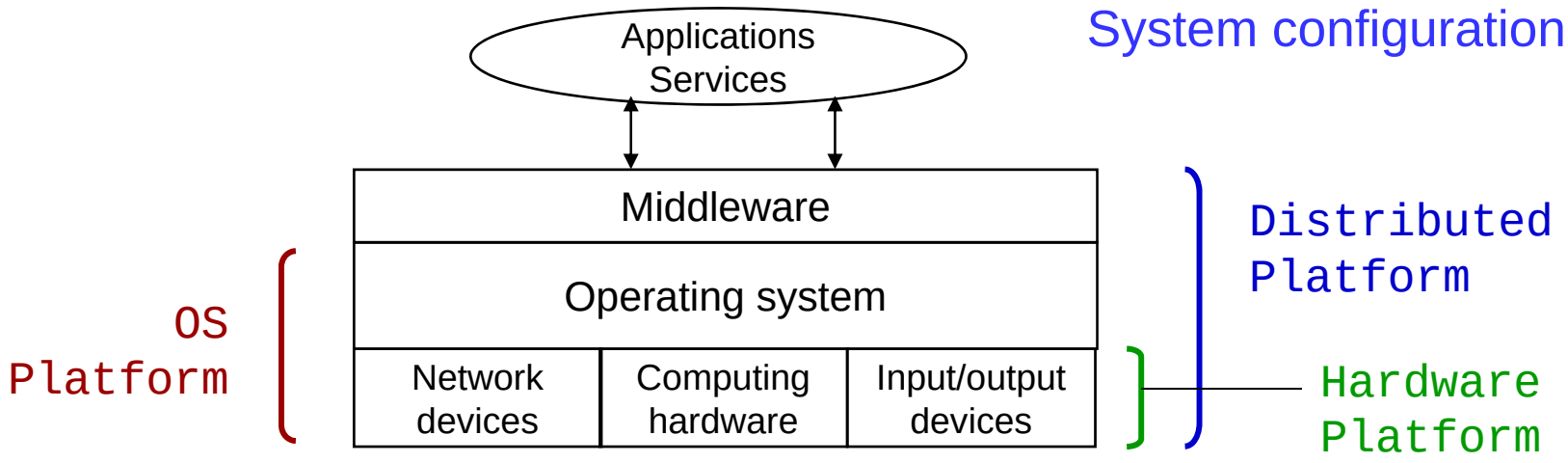
Note: These lecture materials are based on the lecture notes prepared by William Stallings for his book titled *Operating Systems: Internals and Design Principles*, 7e.

Distributed Computing - Background

- Definition (present)
  - a system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages
- Brief history
  - Workstations and powerful small computers
    - 1973: The first workstation, Alto, from Xerox PARC
    - early 1980s: IBM PC(1981), Apple MacIntosh(1984)
    - 1990s, 2000s: cluster-based Internet servers
  - Local area network (LAN)
    - 1973-1975: Robert Metcalfe and his colleagues developed Ethernet at Xerox PARC
  - Distributed applications
    - 1980s: LAN-based two-tier applications
    - 1990s, 2000s: Internet/Web-based multi-tier applications
  - Internet and WWW
    - 1989: "Information Management: A Proposal" written by Tim Berners-Lee at CERN.
    - Jan. 2014: > 1.01 billion Internet hosts (https://www.isc.org/services/survey/)

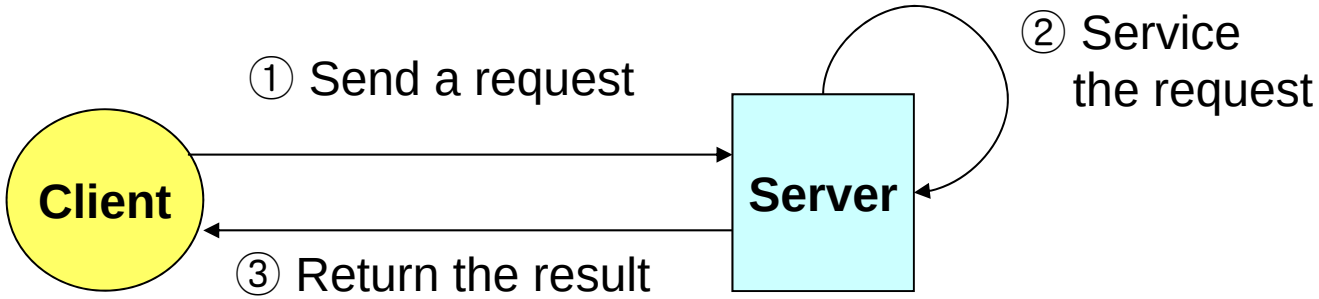
Software Architecture

- OS platform
  - hardware and underlying operating system
- Middleware
  - software layer that masks heterogeneity and provides a programming interface
  - main functions: implements interaction mechanisms across the networked computers
- Applications/services
  - provided by one or more servers



System Architecture

- Client-server model
  - basic model for distributed systems



- Services involving multiple servers
  - e.g. sets of data that are distributed across many servers
- Peer-to-peer (P2P) model
  - peer processes perform a distributed activity or computation without any distinction between clients and servers
  - coordinated at application level
  - the nodes involved act as both client and server

Client/Server Computing

- ***Client*** machines are generally single-user workstations providing a user-friendly interface to the end user
  - client-based stations generally present the type of graphical interface that is most comfortable to users, including the use of windows and a mouse
  - Microsoft Windows and Macintosh OS provide examples of such interfaces
  - client-based applications are tailored for ease of use and include such familiar tools as the spreadsheet

- Each ***server*** provides a set of shared services to the clients
  - most common type of server currently is the database server, usually controlling a relational database
  - enables many clients to share access to the same database
  - enables the use of a high-performance computer system to manage the database

- The third essential ingredient of the client/server environment is the ***network***

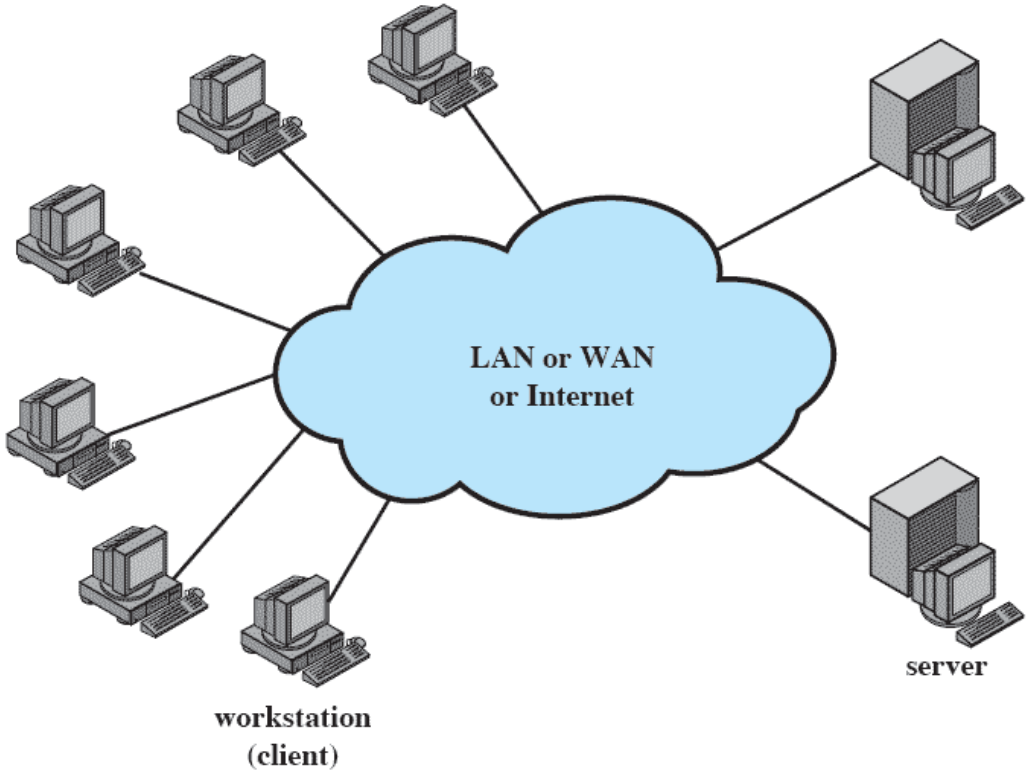


Figure 16.1 Generic Client/Server Environment

Generic Client/Server Architecture

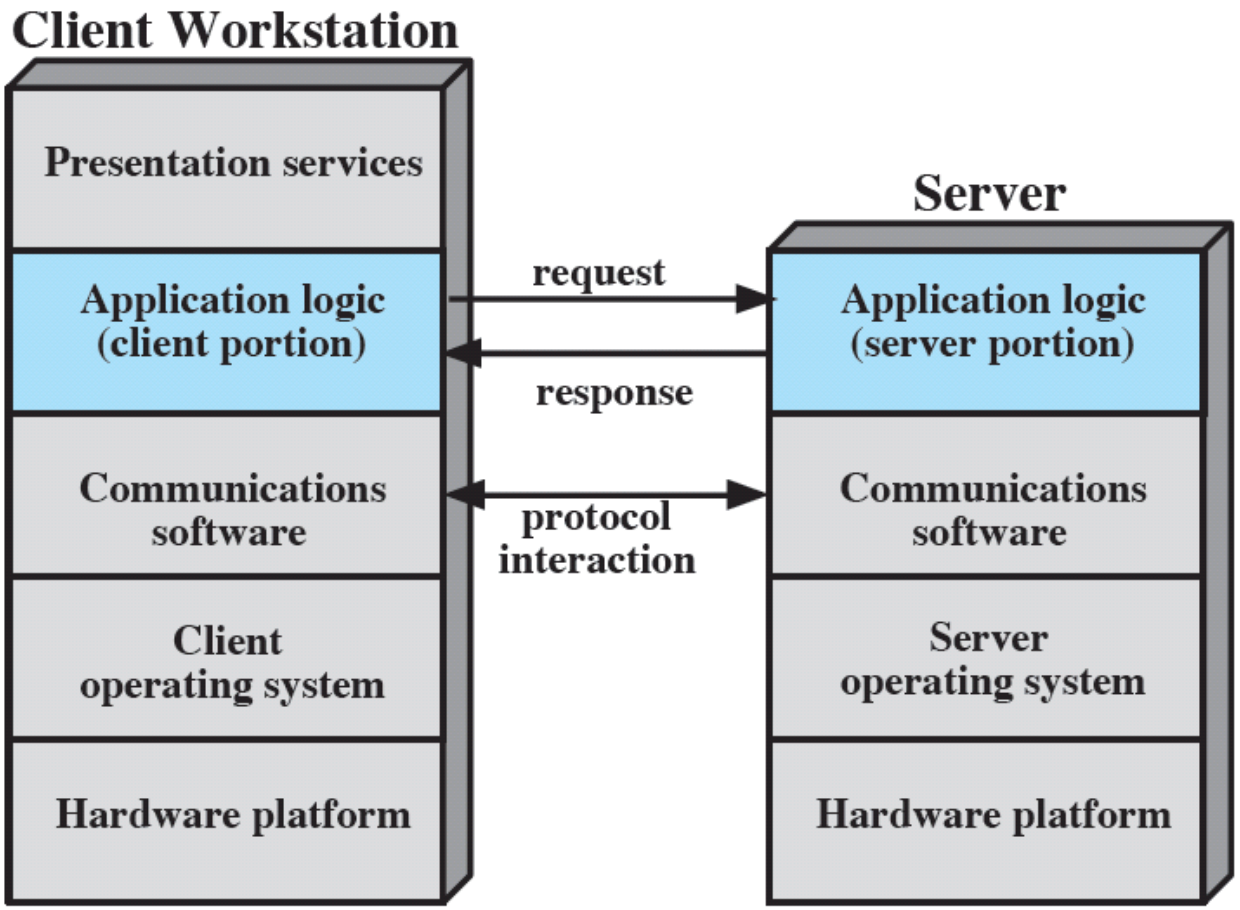


Figure 16.2 Generic Client/Server Architecture

Middleware

- To achieve the true benefits of the client/server approach developers must have a set of tools that provide a uniform means and style of access to system resources across all platforms
- This would enable programmers to build applications that look and feel the same
- Enable programmers to use the same method to access data regardless of the location of that data
- The way to meet this requirement is by the use of standard programming interfaces and protocols that sit between the application (above) and communications software and operating system (below)

Role of Middleware in C/S Architecture

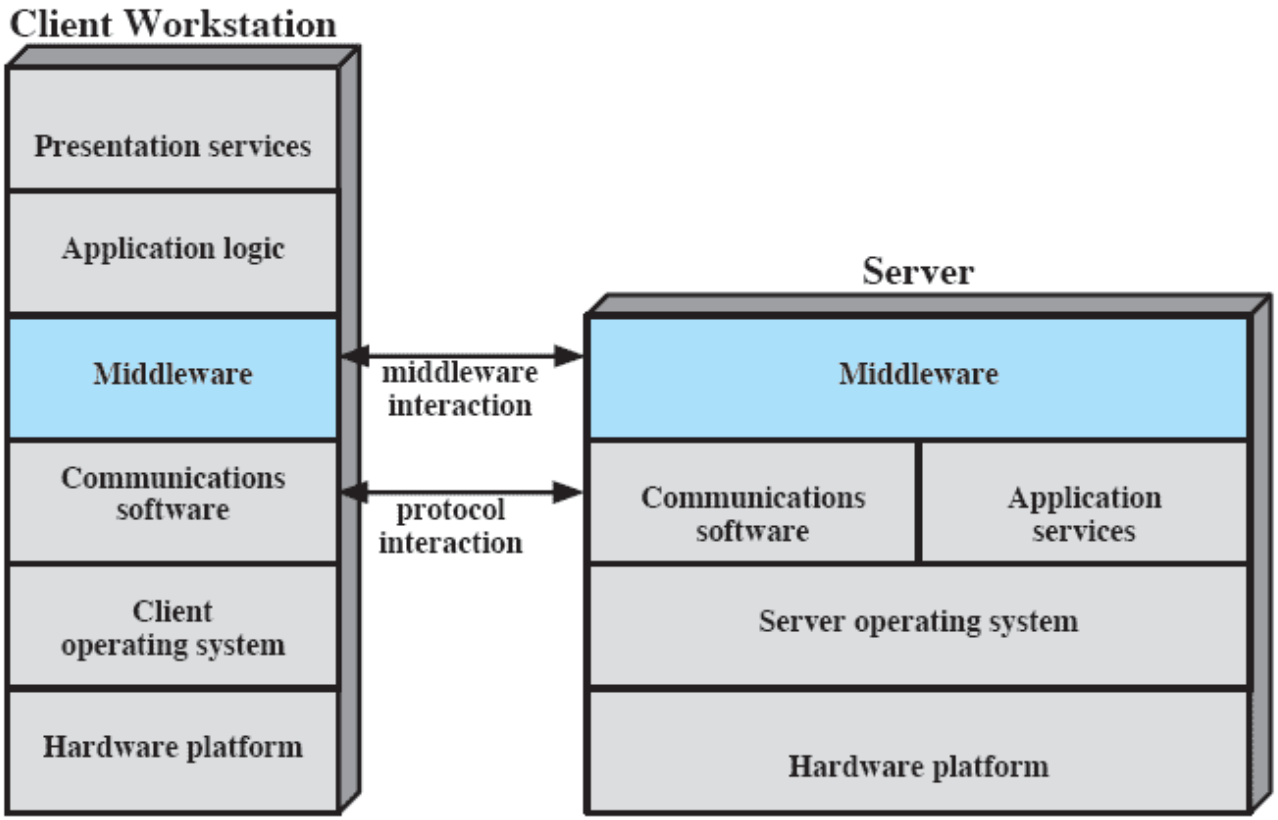


Figure 16.8 The Role of Middleware in Client/Server Architecture

Logical View of Middleware

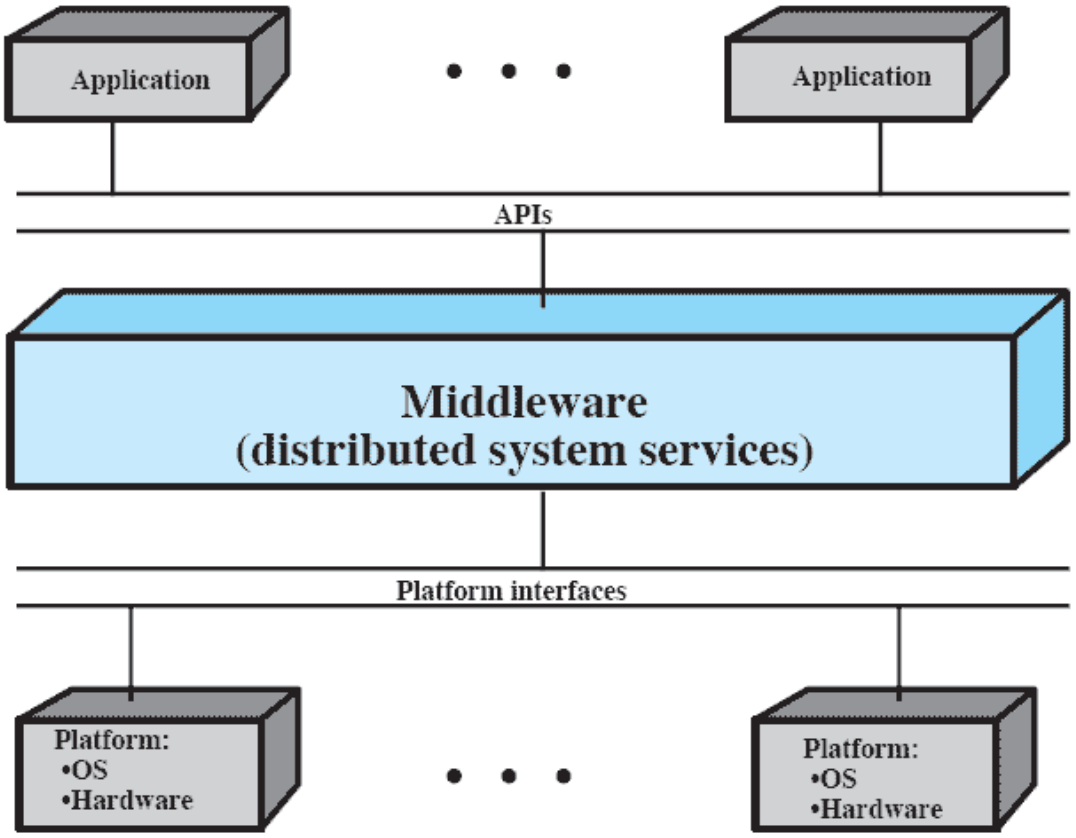


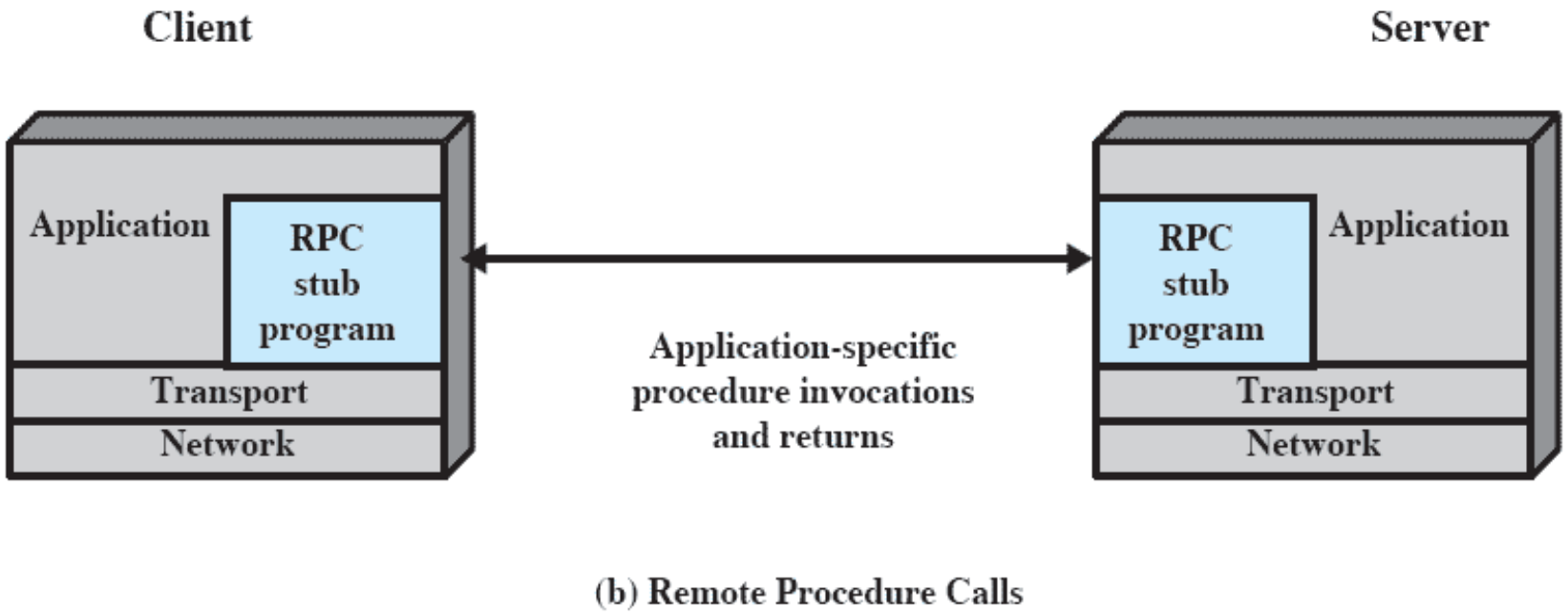
Figure 16.9 Logical View of Middleware

Remote Procedure Calls (RPC)

- Allow programs on different machines to interact using simple procedure call/return semantics
- Used for access to remote services
- Widely accepted and common method for encapsulating communication in a distributed system

- Standardized RPC protocols - Benefits
  - The communication code for an application can be generated automatically
  - Client and server modules can be moved among computers and operating systems with little modification and recoding

Remote Procedure Call Architecture



(b) Remote Procedure Calls

Figure 16.12 Middleware Mechanisms

Remote Procedure Call Mechanism

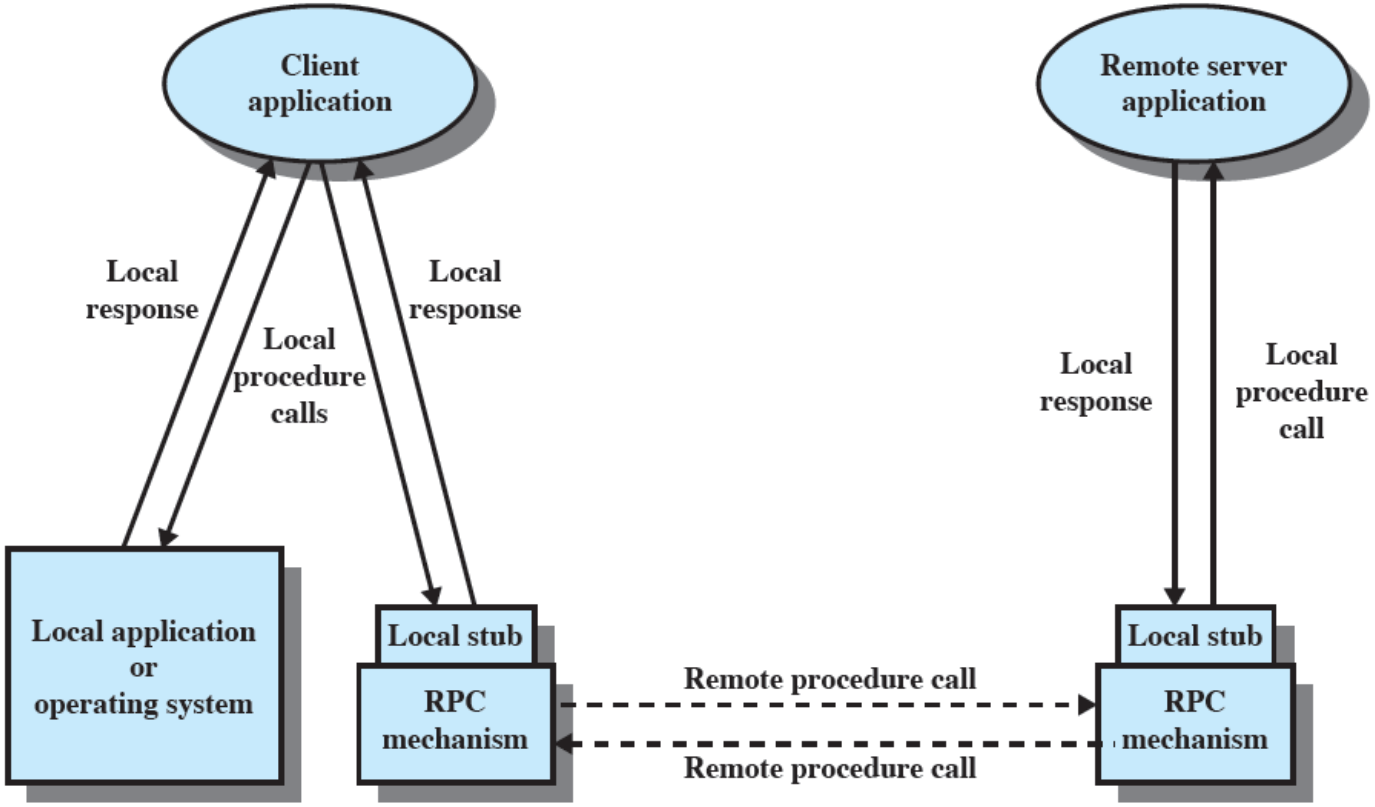
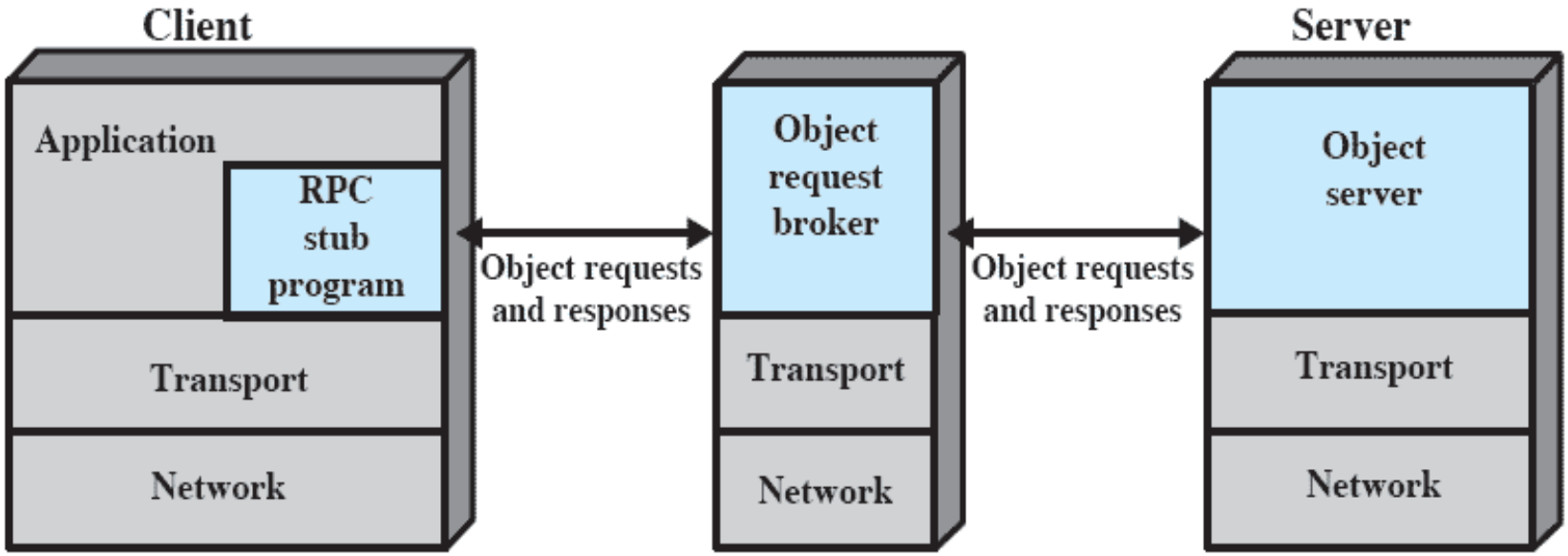


Figure 16.14 Remote Procedure Call Mechanism

Object-Oriented Mechanisms

- Clients and servers ship messages back and forth between objects
- A client that needs a service sends a request to an object broker
- The broker calls the appropriate object and passes along any relevant data
- The remote object services the request and replies to the broker, which returns the response to the client
- The success of the object-oriented approach depends on standardization of the object mechanism
- Examples include Microsoft's COM and CORBA

Object Request Broker



(c) Object request broker

Figure 16.12 Middleware Mechanisms

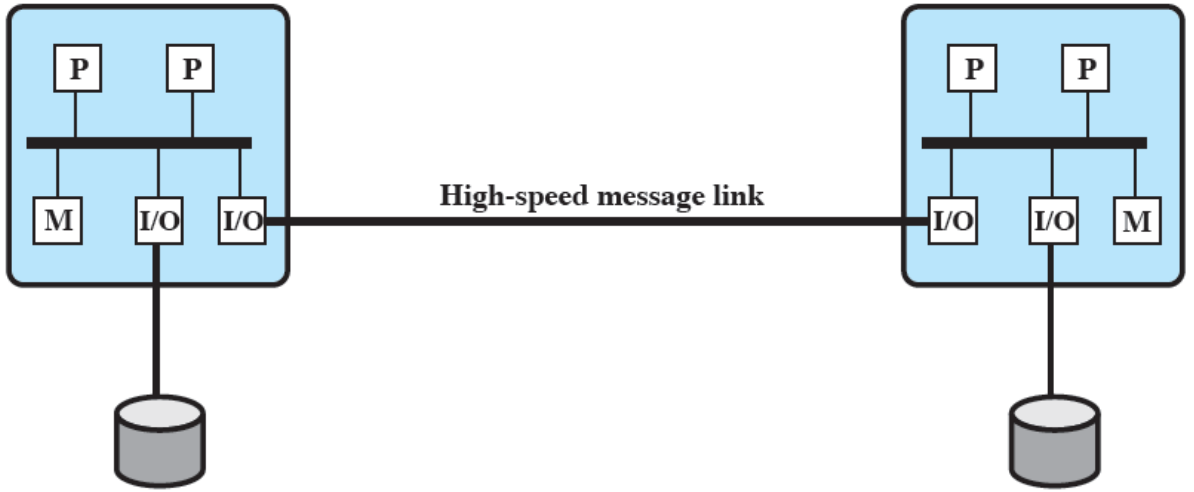
Clusters

- Alternative to symmetric multiprocessing (SMP) as an approach to providing high performance and high availability
- Group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine
- ***Whole computer*** means a system that can run on its own, apart from the cluster
- Each computer in a cluster is referred to as a ***node***

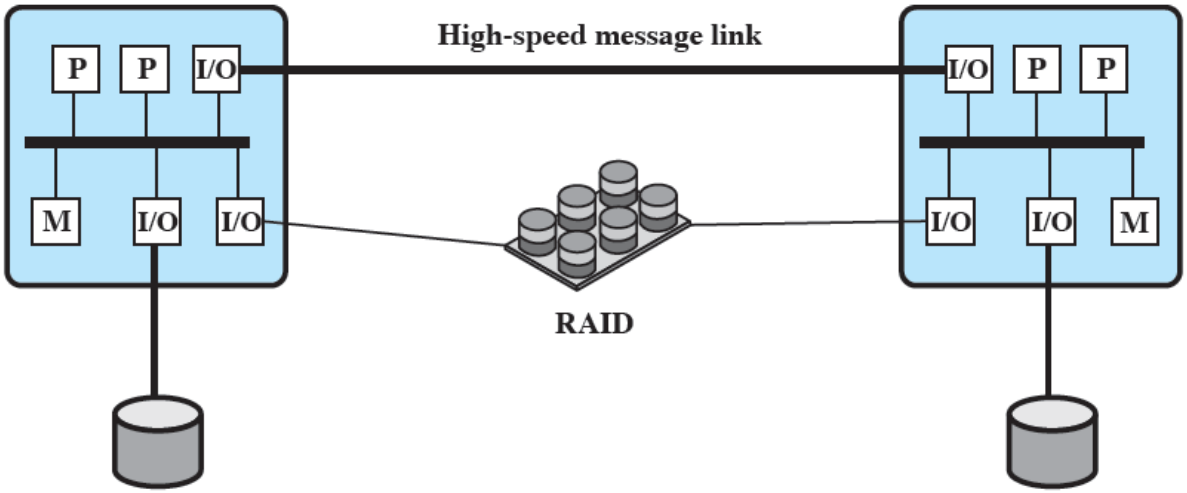
Benefits of Clusters

Absolute scalability	Incremental scalability	High availability	Superior price/perf.
It is possible to create large clusters that far surpass the power of even the largest stand-alone machines • A cluster can have dozens or even hundreds of machines, each of which is a multiprocessor	Configured in such a way that it is possible to add new systems to the cluster in small increments	Failure of one node is not critical to system	By using commodity building blocks, it is possible to put together a cluster at a much lower cost than a single large machine

Cluster Configurations



(a) Standby server with no shared disk



(b) Shared disk

Figure 16.15 Cluster Configurations

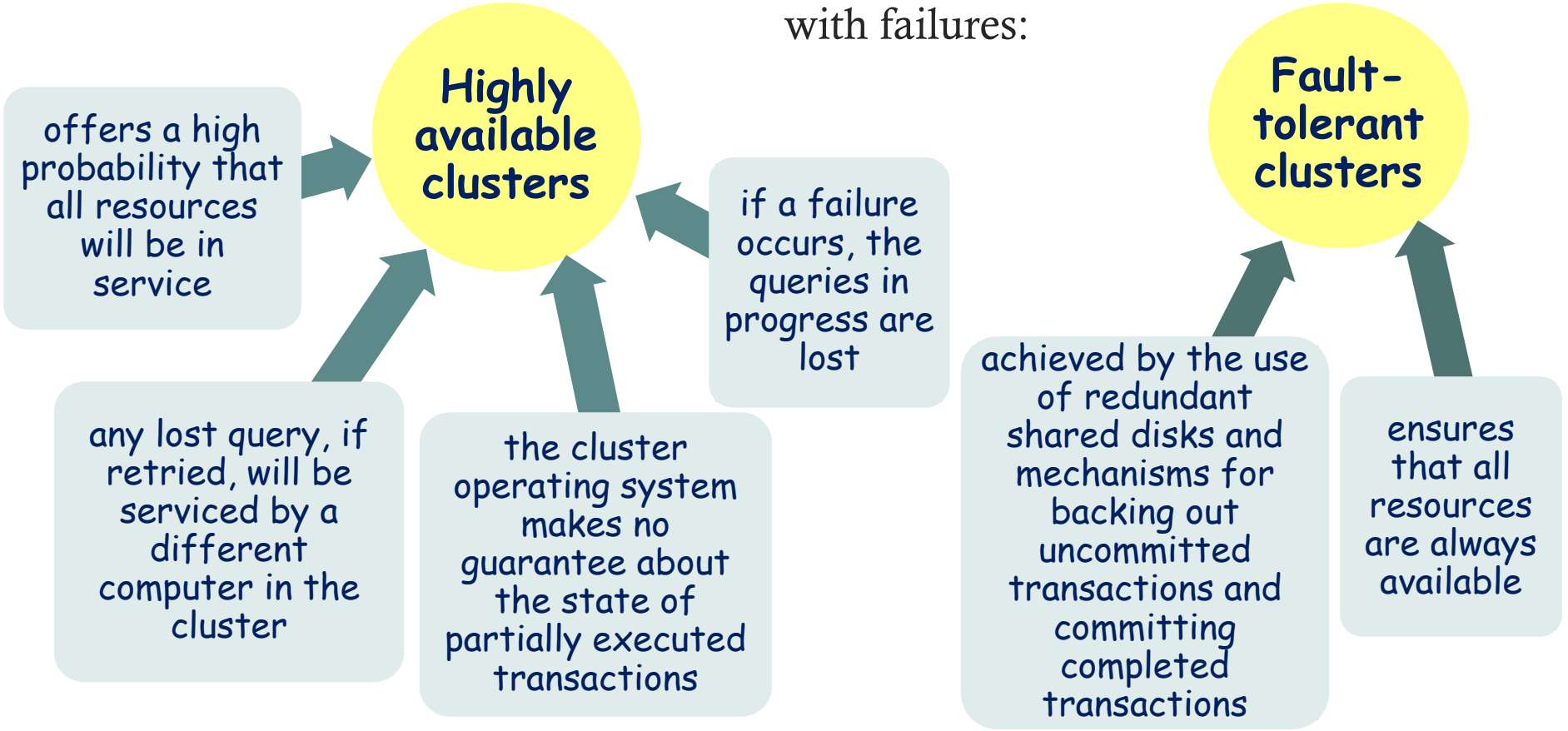
Clustering Methods: Benefits and Limitations

Table 16.2 Clustering Methods: Benefits and Limitations

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

Operating System Design Issues - *Failure Management*

- Two approaches can be taken to deal with failures:



Load Balancing

- A cluster requires an effective capability for balancing the load among available computers
- This includes the requirement that the cluster be incrementally scalable
- When a new computer is added to the cluster, the load-balancing facility should automatically include this computer in scheduling applications
- Middleware must recognize that services can appear on different members of the cluster and may migrate from one member to another

Parallelizing Computation

Parallelizing compiler

- Determines, at compile time, which parts of an application can be executed in parallel
- Performance depends on the nature of the problem and how well the compiler is designed

Parallelized application

- The programmer writes the application from the outset to run on a cluster and uses message passing to move data, as required, between cluster nodes
- This places a high burden on the programmer but may be the best approach for exploiting clusters for some applications

Cluster Computer Architecture

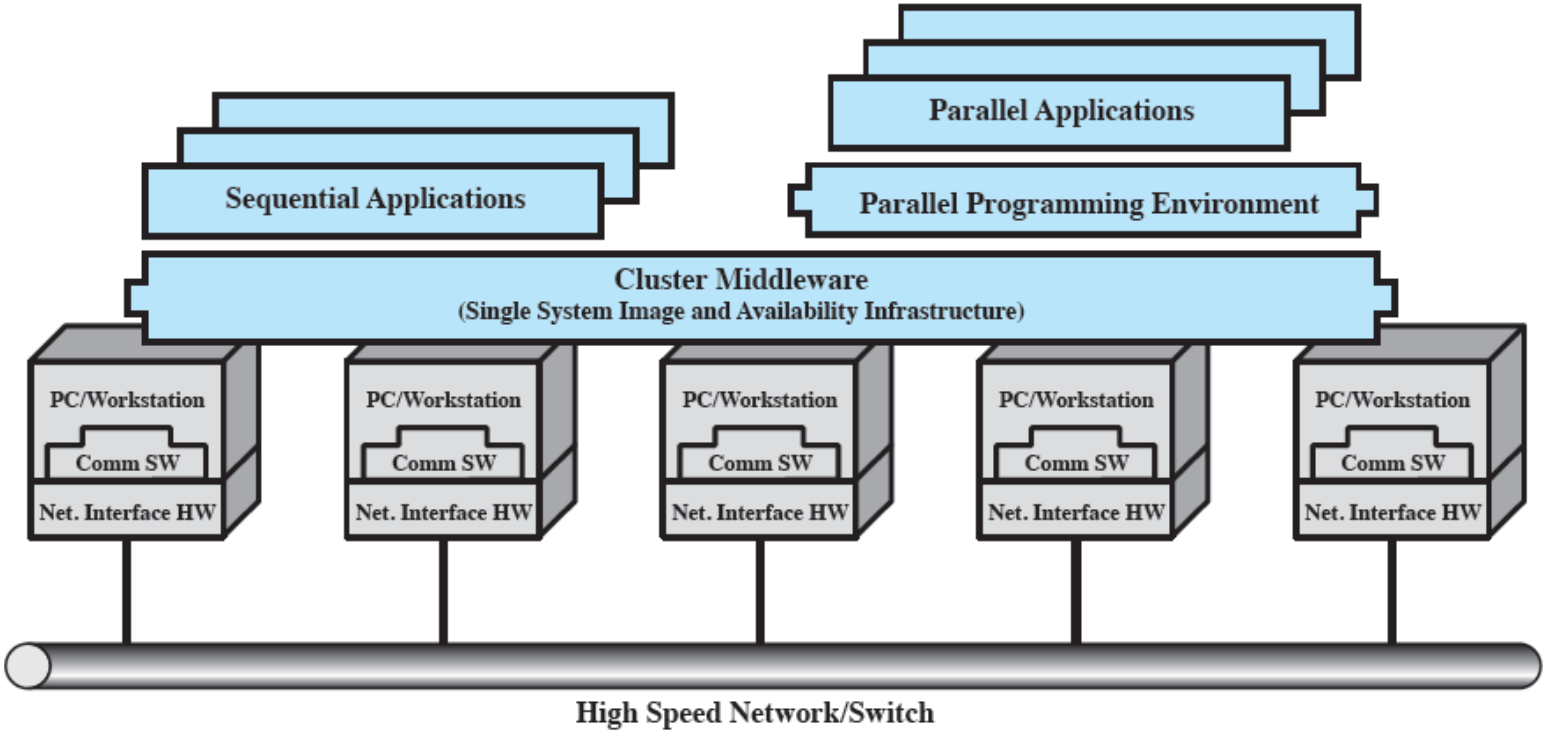


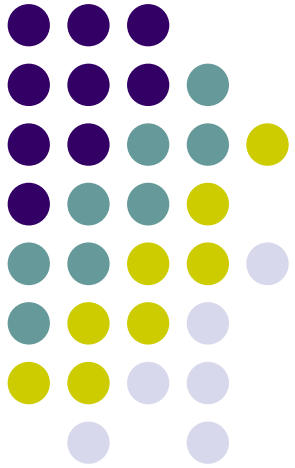
Figure 16.16 Cluster Computer Architecture [BUYY99a]

Clusters Compared to SMP

- Both clusters and SMP provide a configuration with multiple processors to support high-demand applications
- Both solutions are commercially available
- SMP has been around longer
- SMP is easier to manage and configure
- SMP takes up less physical space and draws less power than a comparable cluster
- SMP products are well established and stable
- Clusters are better for incremental and absolute scalability
- Clusters are superior in terms of availability

Virtualization

- Overview
- Implementation of VMMs
- Benefits and features
- Building blocks
- Example: VMware

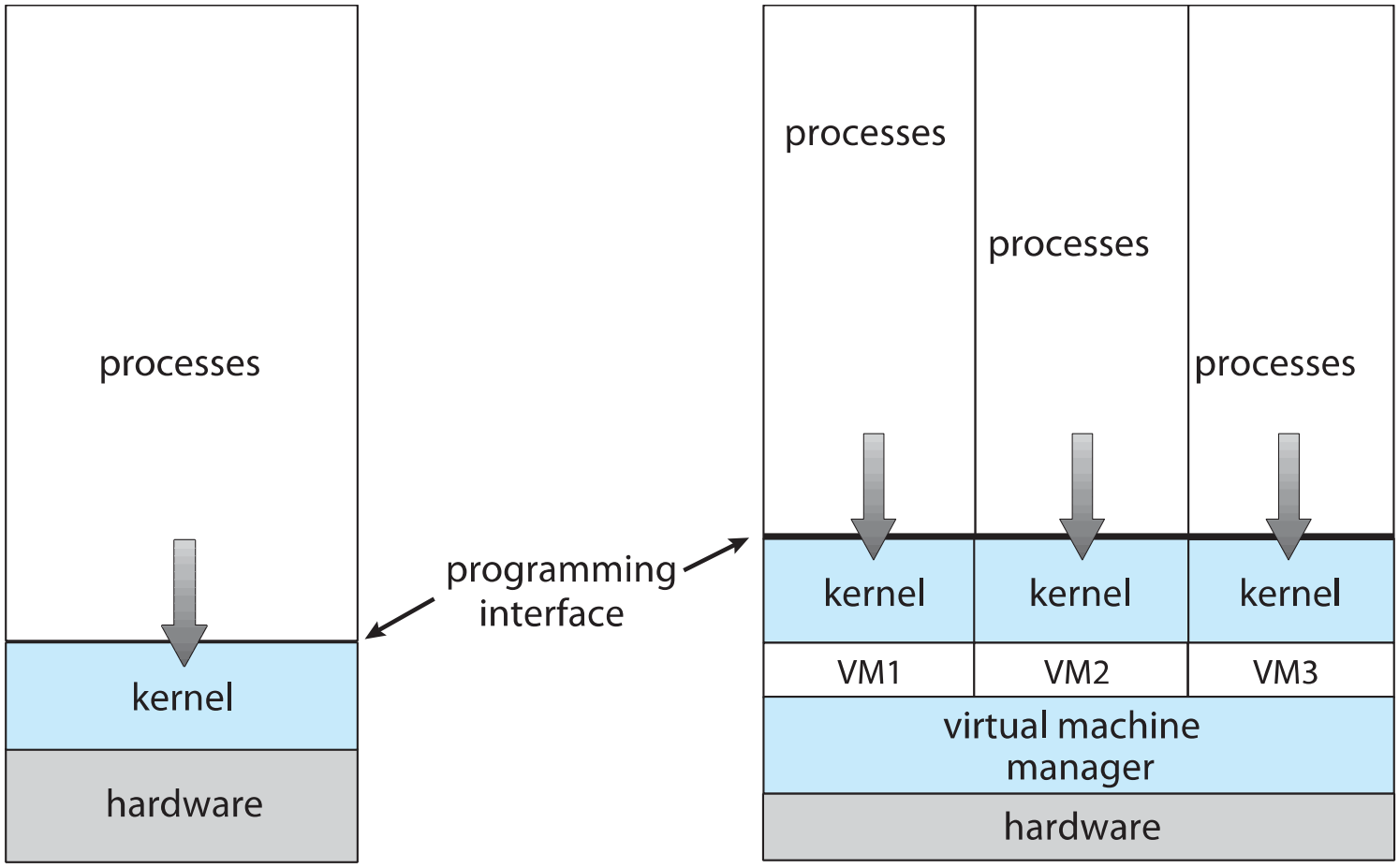


Note: These lecture materials are based on the lecture notes from *Operating System Concepts*, 9e (Wiley), and http://courses.cs.vt.edu/~cs5204/fall09-kafura/ (Dr. D. Kafura)

Overview

- Fundamental idea – abstract hardware of a single computer into several different execution environments
  - Similar to layered approach
  - But layer creates virtual system (**virtual machine**, or VM) on which operating systems or applications can run
- Several components
  - **Host** – underlying hardware system
  - **Virtual machine manager** (VMM) or **hypervisor** – creates and runs virtual machines by providing interface that is identical to the host
    - (Except in the case of paravirtualization)
  - **Guest** – process provided with virtual copy of the host
    - Usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

System Models



Non-virtual machine

Virtual machine

History

- First appeared in IBM mainframes in 1972
- Allowed multiple users to share a batch-oriented system
- Formal definition of virtualization helped move it beyond IBM
  - A VMM provides an environment for programs that is essentially identical to the original machine
  - Programs running within that environment show only minor performance decreases
  - The VMM is in complete control of system resources
- In late 1990s Intel CPUs fast enough for researchers to try virtualizing on general purpose PCs
  - **Xen** and **VMware** created technologies, still used today
  - Virtualization has expanded to many OSs, CPUs, VMMs

Implementation of VMMs

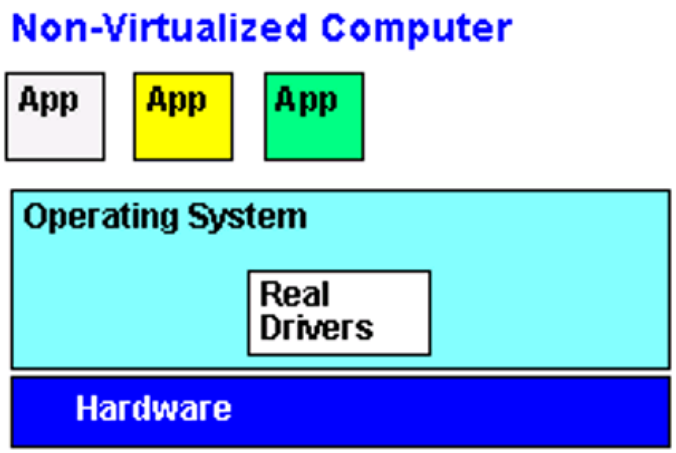
- Full virtualization varies greatly, with options including:
  - **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
    - IBM LPARs and Oracle LDOMs are examples
  - **Type 1 hypervisors** (1/2) - Operating-system-like software built to provide virtualization
    - Including VMware ESX, Joyent SmartOS, and Citrix XenServer
  - **Type 1 hypervisors** (2/2) - Also includes general-purpose operating systems that provide standard functions as well as VMM functions
    - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
  - **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
    - Including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

Implementation of VMMs (Cont.)

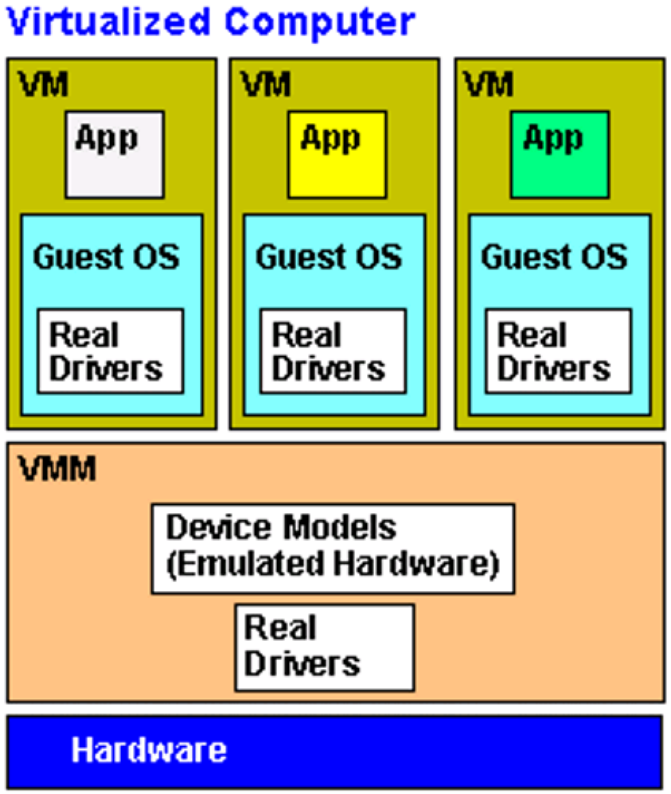
- Other variations include:
  - **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
  - **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
    - Used by Oracle Java and Microsoft.Net
  - **Emulators** – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU
  - **Application containment** - Not virtualization at all but rather provides virtualization-like features by segregating applications from the operating system, making them more secure, manageable
    - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs
- Much variation due to breadth, depth and importance of virtualization in modern computing

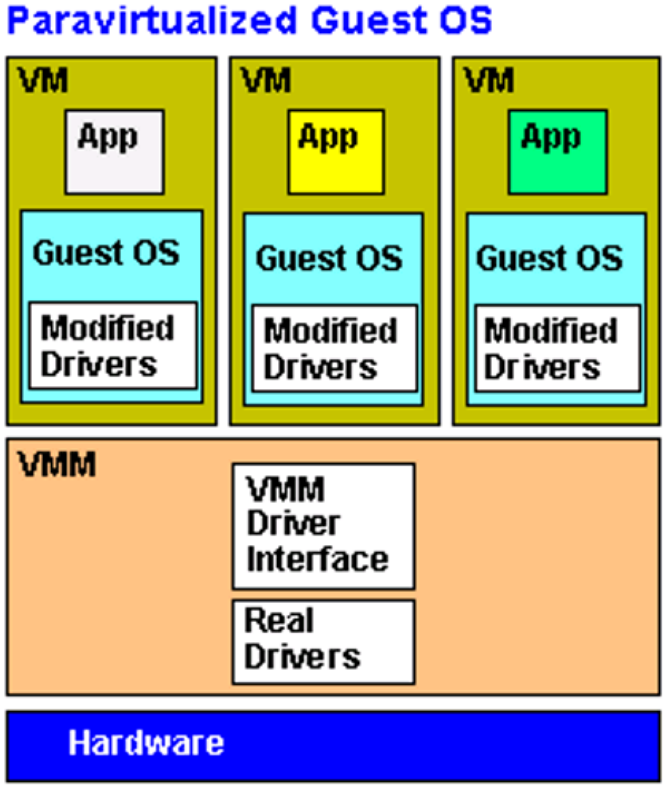
Full Virtualization vs. Paravirtualization

- Various software-based solutions
- Typical implementation techniques for virtualization
  - Full virtualization
    - Direct execution - Guest OS doesn't need modification
    - The VMM presents a "device model" to the guest OS, which emulates the hardware.
    - Hypervisors: Type 0, Type 1, Type 2
  - Paravirtualization
    - Guest OS is modified to work in cooperation with the VMM to optimize performance
      - Including the Xen virtual machine monitor with paravirtualized Linux, OpenBSD, FreeBSD and OpenSolaris as guest OS
    - Calls to the hardware from the guest OS are replaced with calls to the virtual machine monitor (VMM).



Illustrations of virtualization with emphasis on device drivers





http://www.pcmag.com/encyclopedia/term/59102/paravirtualization

Benefits and Features

- Host system protected from VMs, VMs protected from each other
  - I.e. A virus less likely to spread
  - Sharing is provided though via shared file system volume, network communication
- Freeze (or suspend) a running VM
  - Then can freeze, copy and move/clone somewhere else, and then resume
  - Snapshot of a given state, able to restore back to that state
    - Some VMMs allow multiple snapshots per VM
  - **Clone** by creating copy and running both original and copy
- Great for OS research, better system development efficiency
- Run multiple, different OSs on a single machine
  - Application developers: rapid porting and testing of programs in varying environments
  - Data centers: system **consolidation** running multiple VMs on one system

Benefits and Features (cont.)

- **Templating**
  - VM templates are preconfigured virtual machines images and configurations
  - System managers create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
  - E. g., good for the deployment of extra VMs into the cloud service, taking minimal time
- **Live migration**
  - Move a running VM from one host to another!
  - No interruption of user access
- All those features taken together -> **cloud computing**
  - Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops

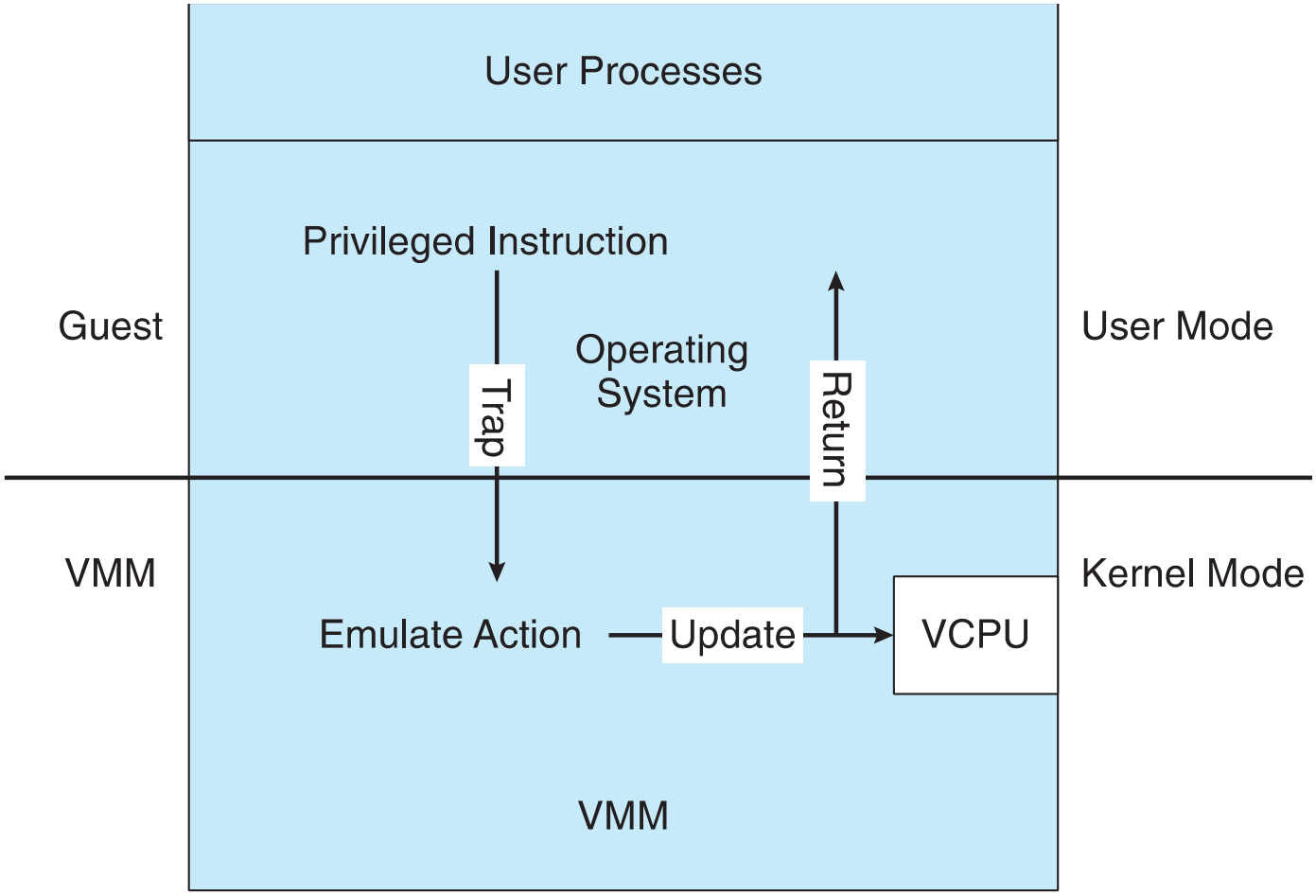
Building Blocks for VM Implementation

- Virtual CPU (VCPU)
  - Generally difficult to provide an exact duplicate of underlying machine
    - Especially if only dual-mode operation available on CPU
    - But getting easier as CPU support for VMM improves
  - Most VMMs implement virtual CPU (VCPU) to represent state of CPU per guest
    - When guest context switched onto CPU by VMM, information from VCPU loaded and stored
- Several techniques employed
  - Trap-and-emulate → next slide
  - Binary translation
    - If impossible to emulate an instruction, translate into new set of instructions that perform equivalent task
  - Nested page table
    - VMM maintains per-guest "nested page tables" for address translation

Building Block – *Trap and Emulate*

- Dual mode (user mode, kernel mode) CPU
  - Guest runs in user mode; host kernel runs in kernel mode
    - Not safe to let guest kernel run in kernel mode, too
  - So VM needs two modes for guest – virtual user mode and virtual kernel mode
    - Both of which run in real user mode
  - Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode
- Switching from virtual user mode to virtual kernel mode
  - Attempting a privileged instruction in user mode causes an error -> trap
  - VMM gains control, analyzes error, executes operation as attempted by guest
  - Returns control to guest in user mode
  - Known as **trap-and-emulate**
  - Most virtualization products use this at least in part
- User mode code in guest runs at same speed as if not a guest
  - But kernel mode or privilege mode code runs slower due to trap-and-emulate

Trap-and-Emulate (Cont.)



***Trap-and-Emulate* Virtualization Implementation**

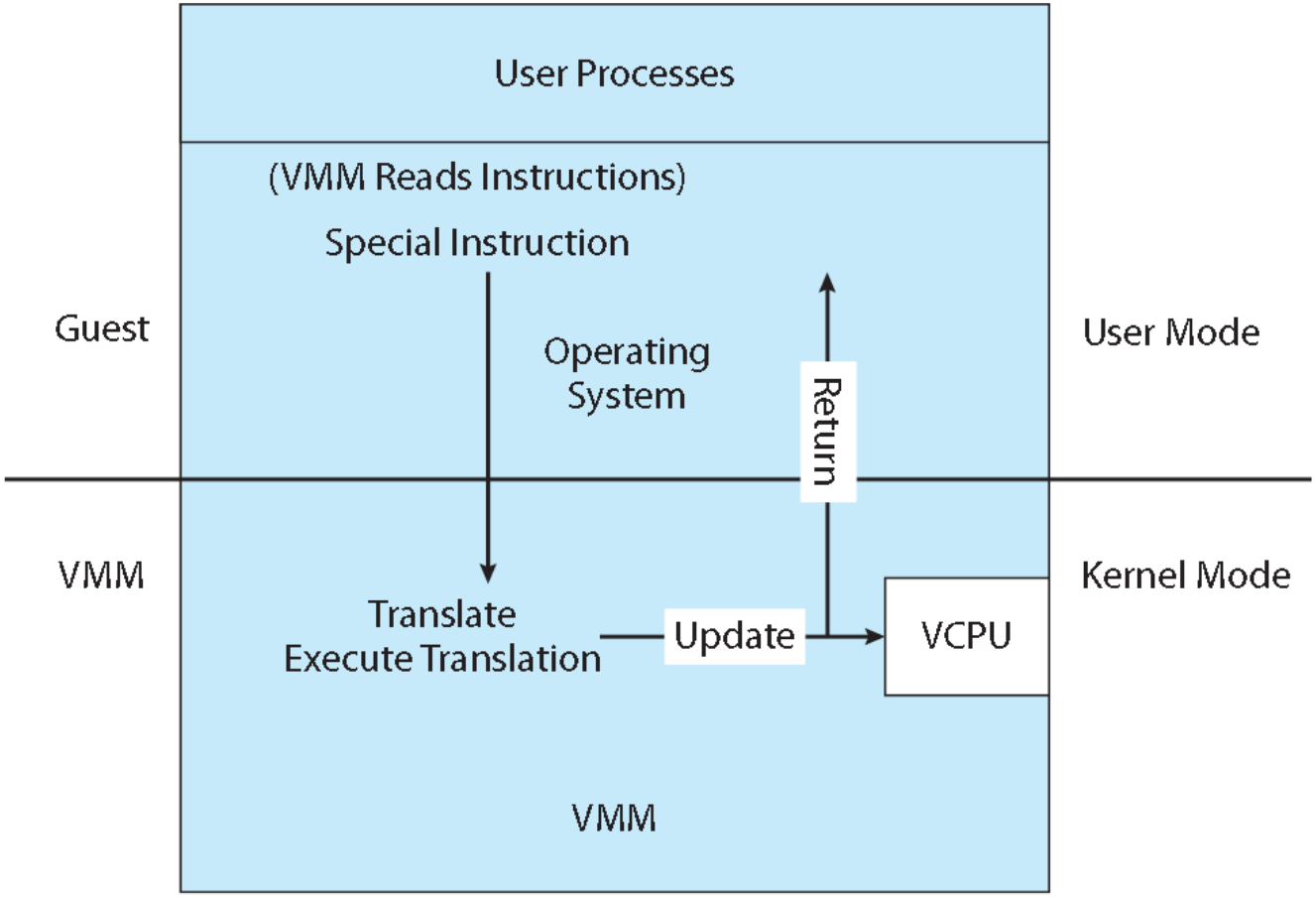
Building Block – Binary Translation

- Some CPUs don't have clean separation between privileged and nonprivileged instructions
  - Earlier Intel x86 CPUs are among them
    - Backward compatibility means difficult to improve
  - Consider Intel x86 `popf` instruction
    - Loads CPU flags register from contents of the stack
    - If CPU in privileged mode -> all flags replaced
    - If CPU in user mode -> on some flags replaced (No trap is generated)
  - Other similar problem instructions we will call ***special instructions***
    - Caused trap-and-emulate method considered impossible until 1998
- Binary translation solves the problem
  - Basics are simple, but implementation very complex
  - If guest VCPU is in user mode, guest can run instructions natively
  - If guest VCPU in kernel mode (guest believes it is in kernel mode)
    - VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
    - Non-special-instructions run natively
    - Special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)

Binary Translation (cont.)

- Implemented by translation of code within VMM
- Code reads native instructions dynamically from guest, on demand, generates native binary code that executes in place of original code
- Performance of this method would be poor without optimizations
  - Products like VMware use caching
    - Translate once, and when guest executes code containing special instruction cached translation used instead of translating again
    - Testing showed booting Windows XP as guest caused 950,000 translations, at 3 microseconds each, or 3 second (5 %) slowdown over native

Binary Translation Virtualization Implementation

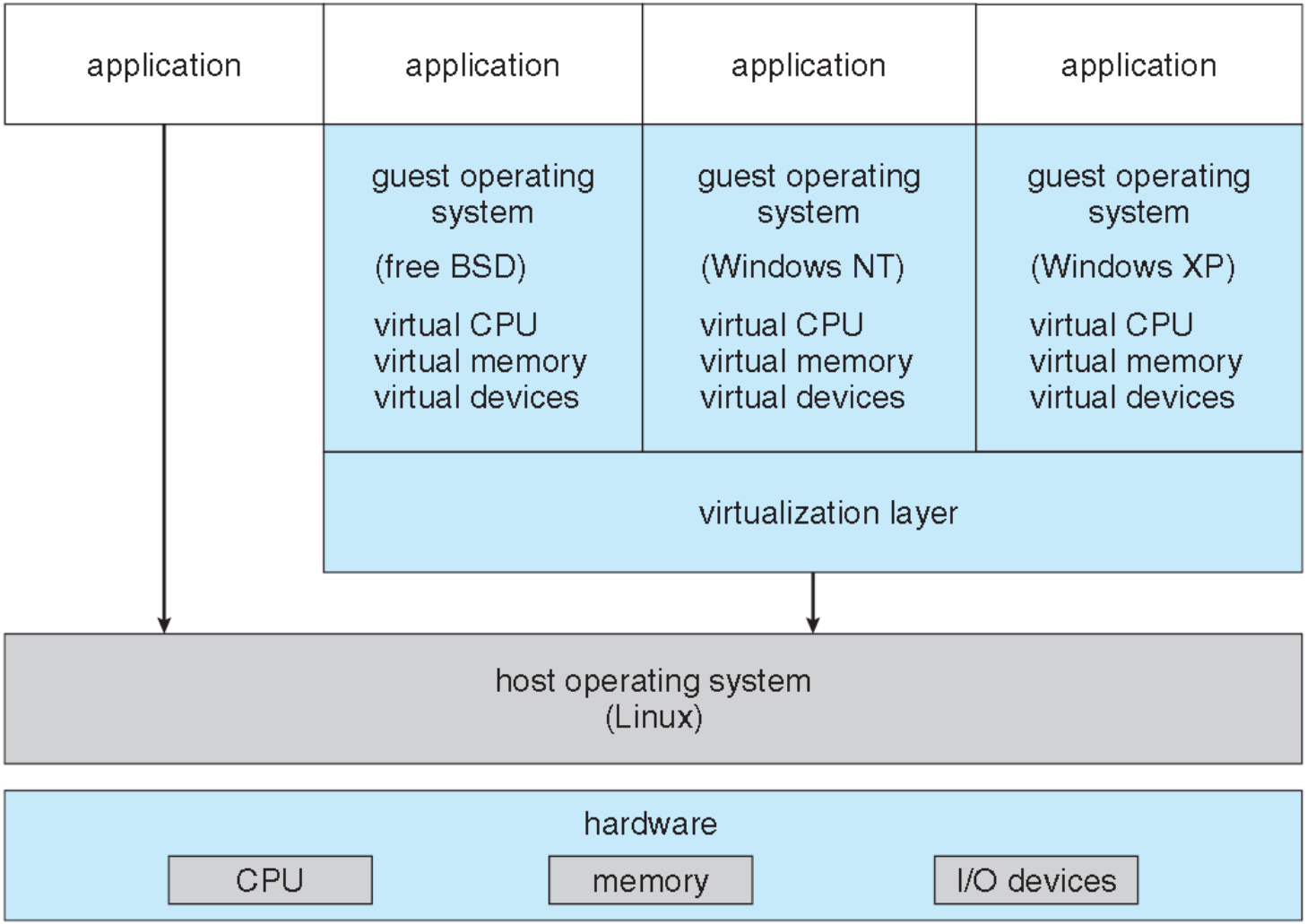


Examples - VMware

- VMware Workstation runs on x86, provides VMM for guests
- Runs as application on other native, installed host operating system -> Type 2
- Lots of guests possible, including Windows, Linux, etc. all runnable concurrently (as resources allow)
- Virtualization layer abstracts underlying HW, providing guest with is own virtual CPUs, memory, disk drives, network interfaces, etc.
- Physical disks can be provided to guests, or virtual physical disks (just files within host file system)

VMware Workstation Architecture

- Case of full virtualization



Summary

- 분산 컴퓨팅
  - 발전과정
  - 미들웨어
  - 클라이언트-서버 컴퓨팅 모델
  - 원격 프로시저 호출(RPC)
  - 클러스터
- 가상화
  - 가상기계
  - 가상기계 매니저/모니터 (VMM)
  - 이점 및 특징
  - 구현기법
  - 구현을 위한 빌딩 블록
  - 예: VMware

- Distributed computing
  - history
  - middleware
  - client-server computing model
  - remote procedure call (RPC)
  - clusters
- Virtualization
  - virtual machine
  - VM manager/monitor
  - Benefits and features
  - Implementation techniques
  - Building blocks for implementation
  - Example: VMware