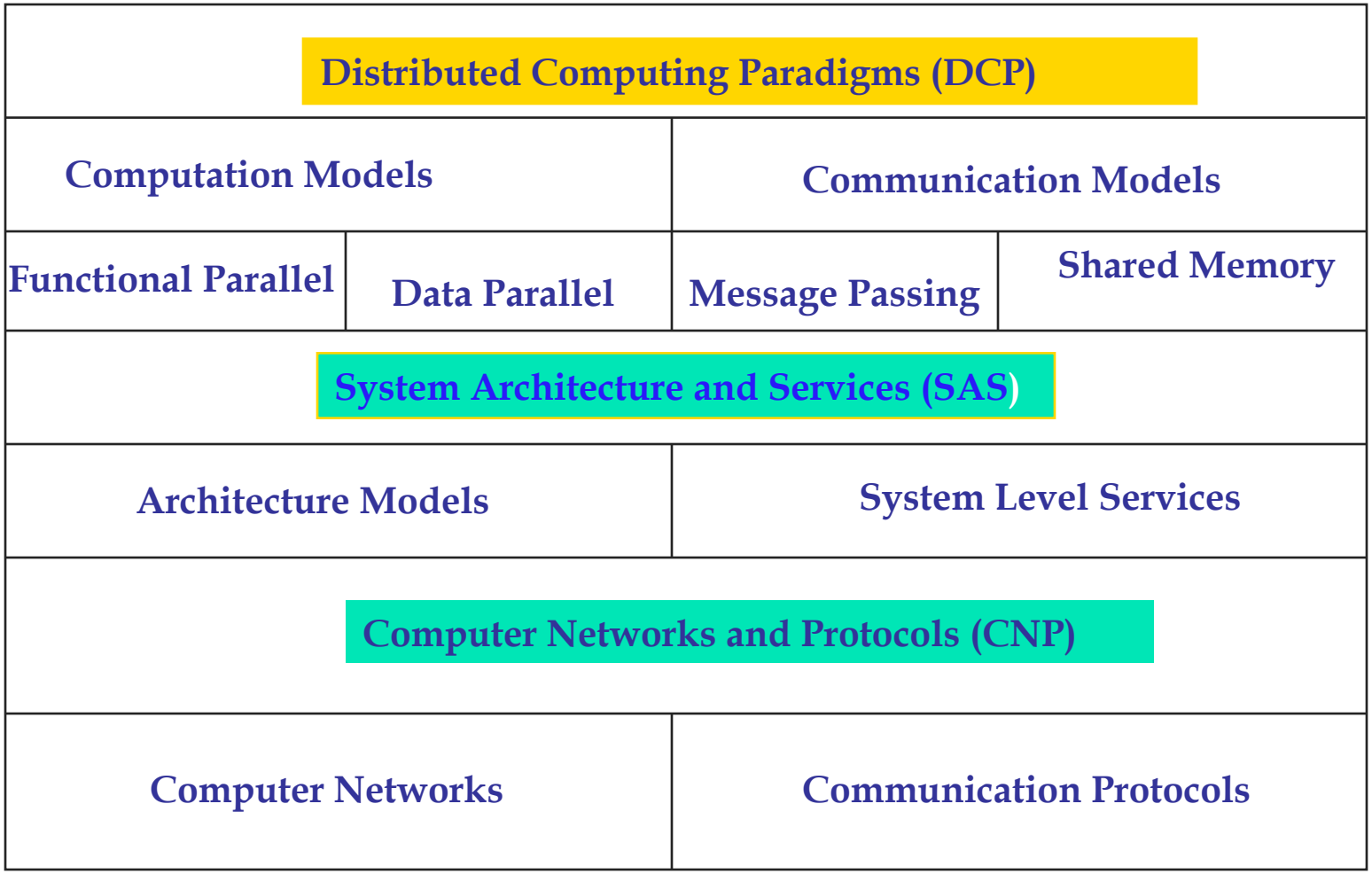
ECE 677: *Distributed Computing Systems*

Distributed Operating Systems: Design Issues

Case Studies

Distributed Systems Design Framework (Cont)



Outline

- Operating System Services and Designs
- Distributed System Design Issues
- Case Studies

Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - User interface - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - I/O operations - A running program may require I/O, which may involve a file or an I/O device.
  - File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont):
  - Communications – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the OS)
  - Error detection – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - Some (such as CPU cycles,mainmemory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code.
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources
  - **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
    - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

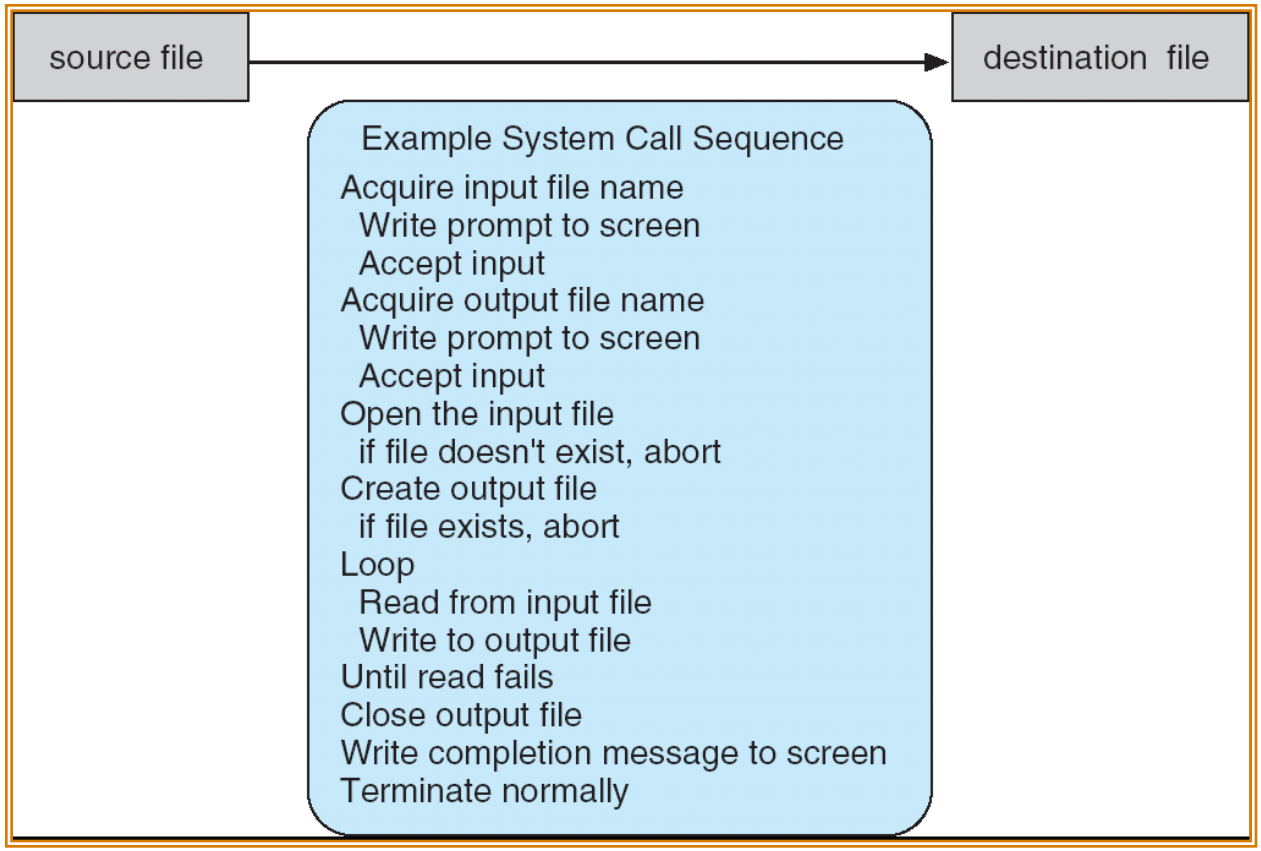
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

(Note that the system-call names used throughout this text are generic)

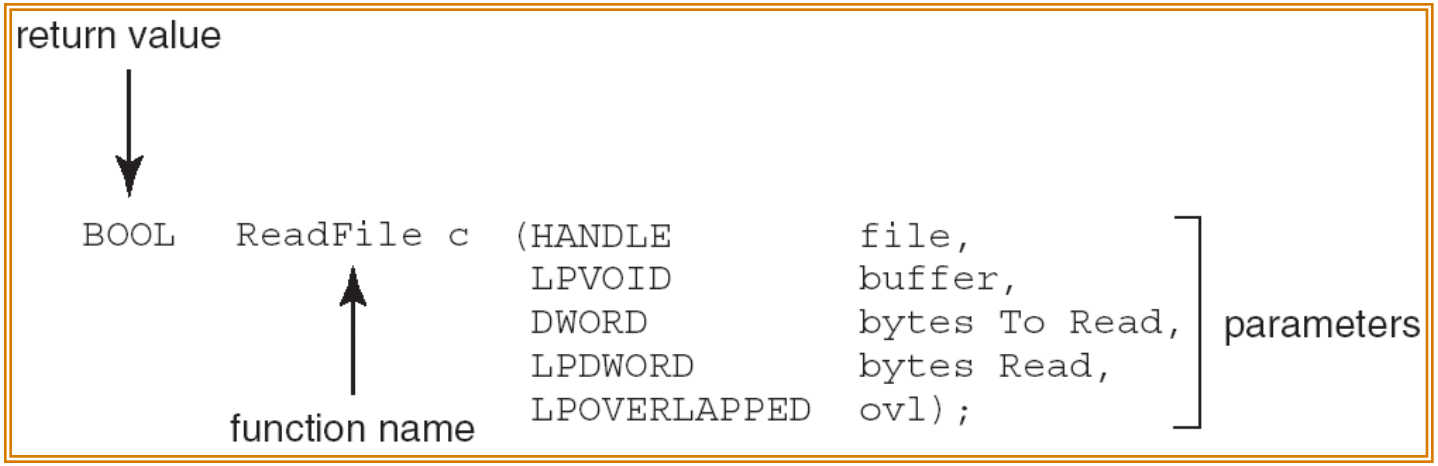
Example of System Calls

- System call sequence to copy the contents of one file to another file



Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



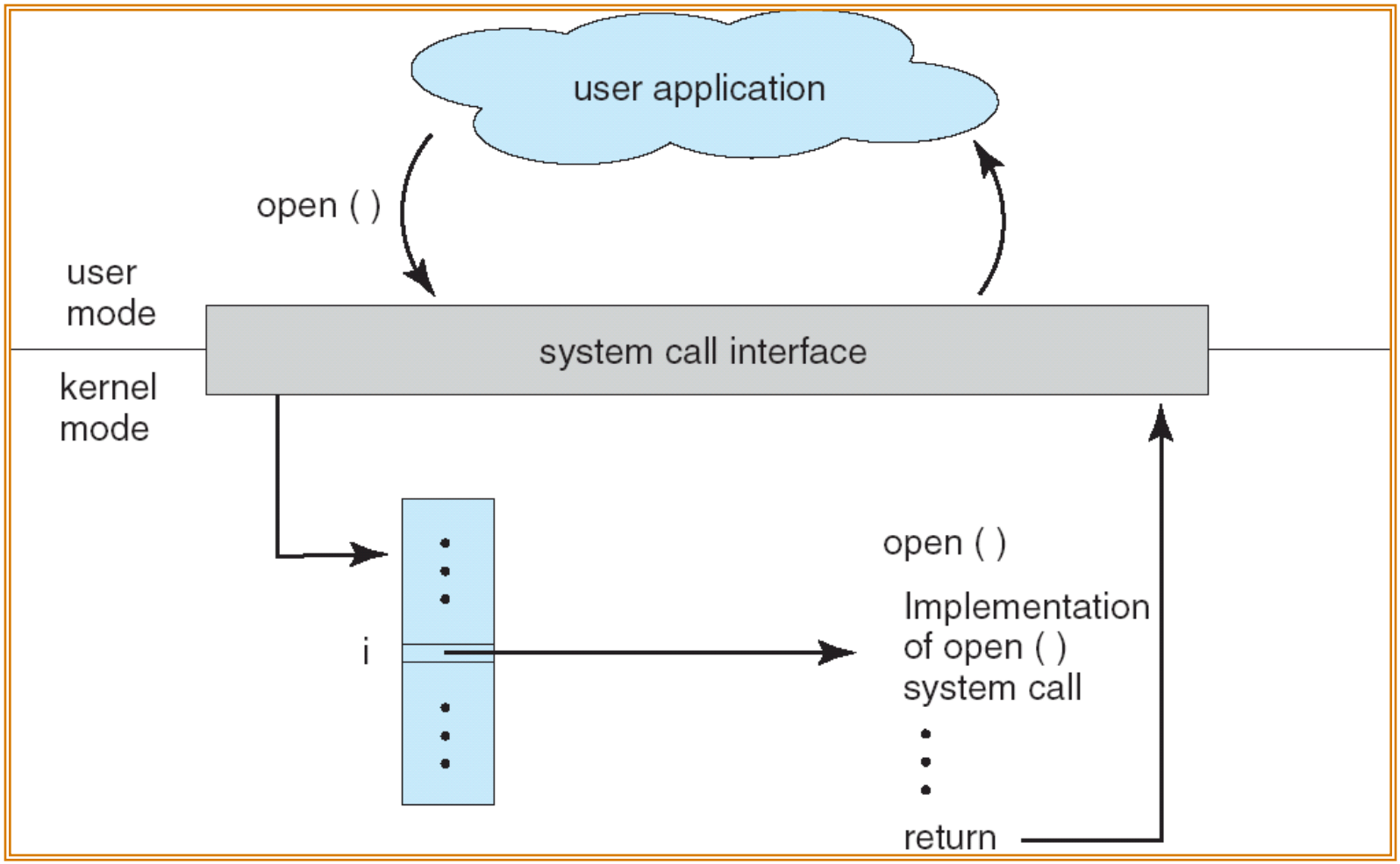
A description of the parameters passed to ReadFile()

- HANDLE file—the file to be read
- LPVOID buffer—a buffer where the data will be read into and written from
- DWORD bytesToRead—the number of bytes to be read into the buffer
- LPDWORD bytesRead—the number of bytes read during the last read
- LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

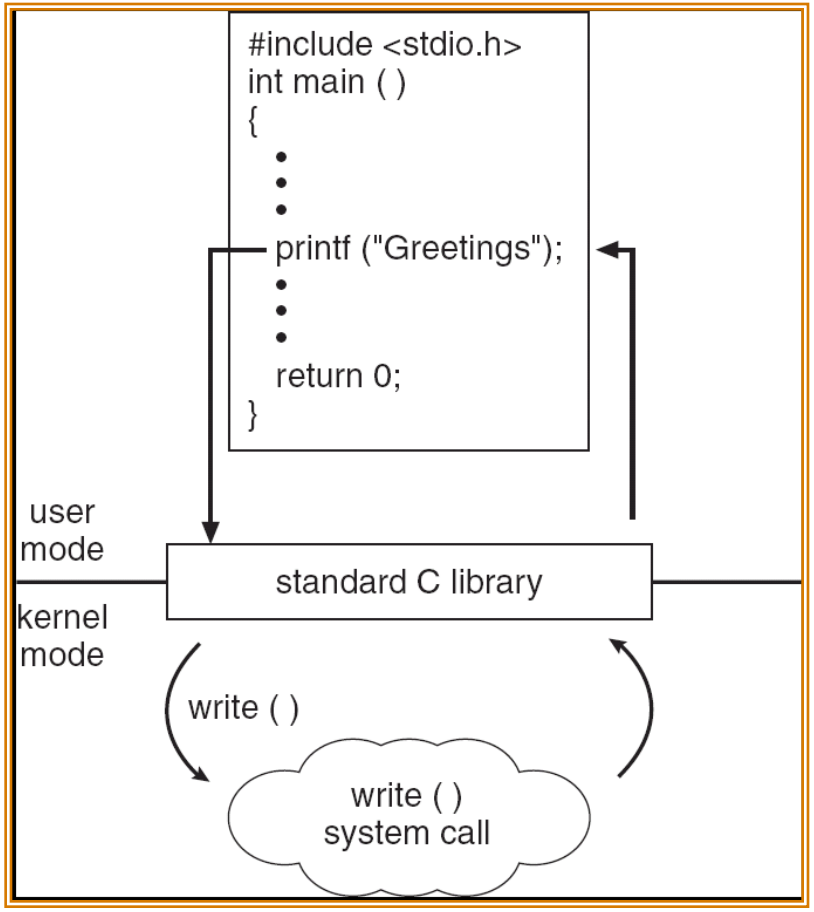
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



Standard C Library Example

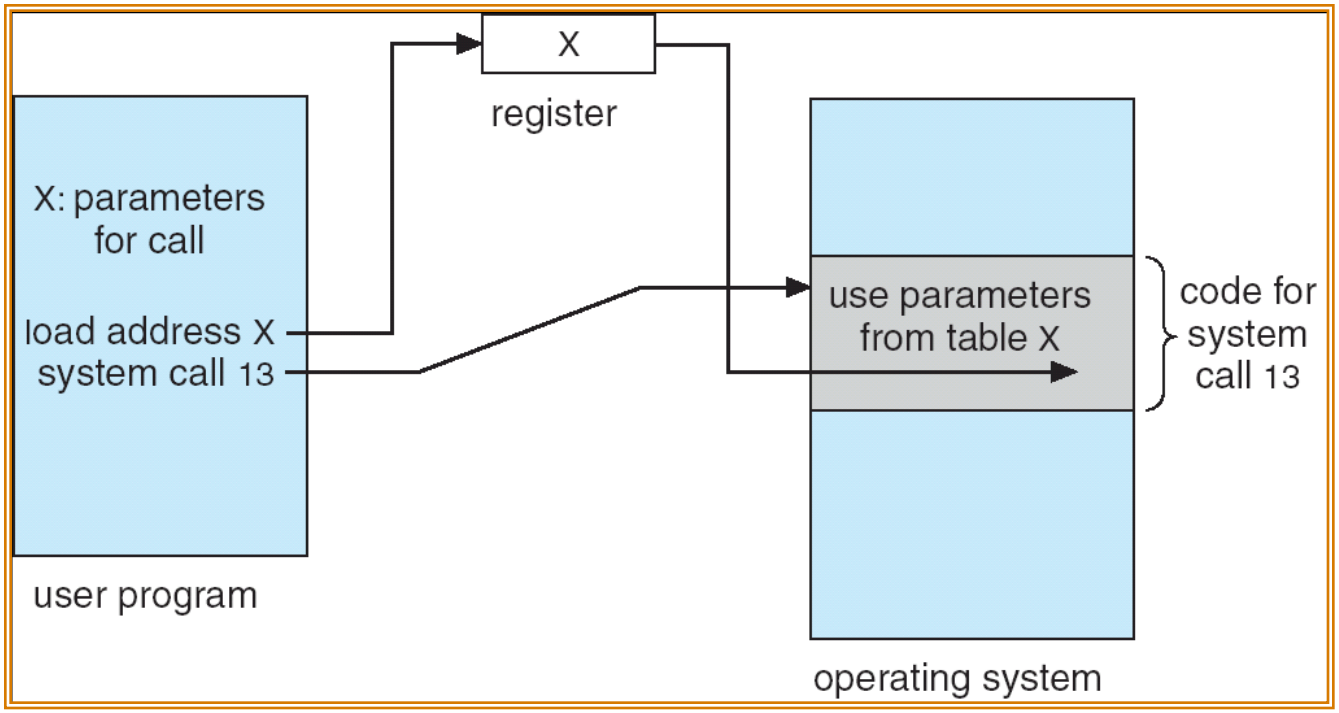
- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

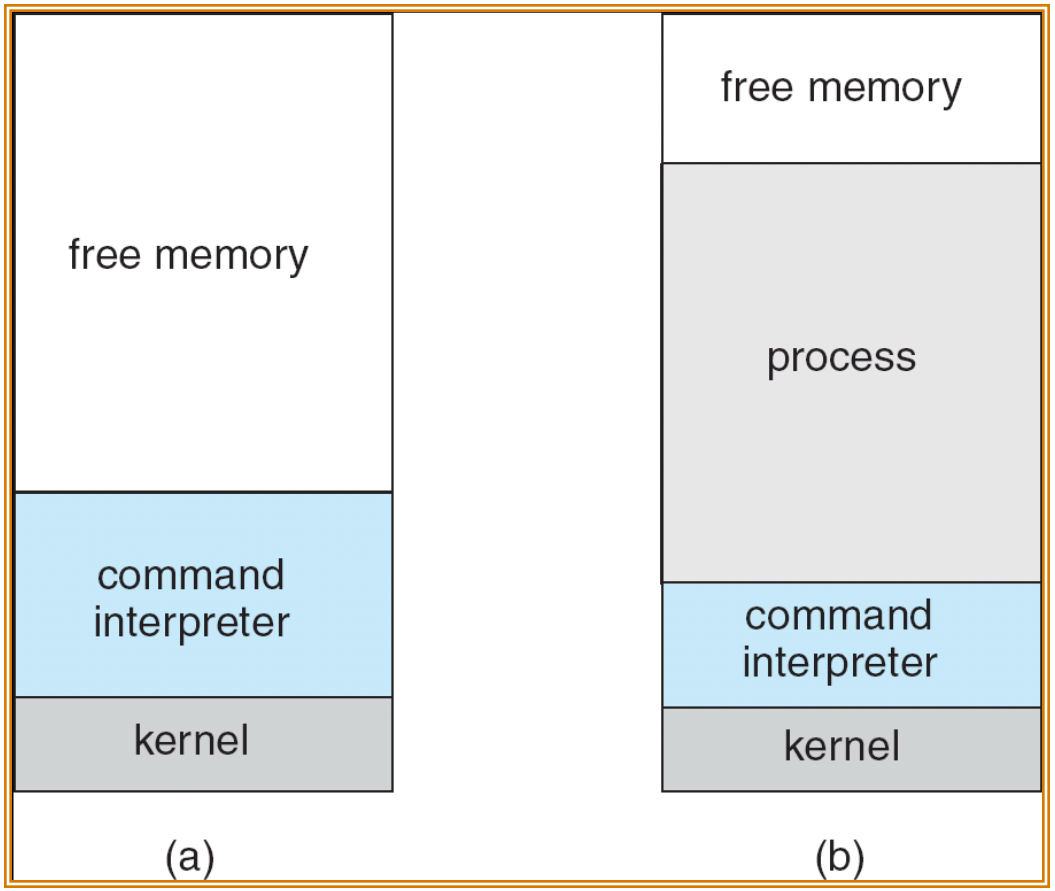
Parameter Passing via Table



Types of System Calls

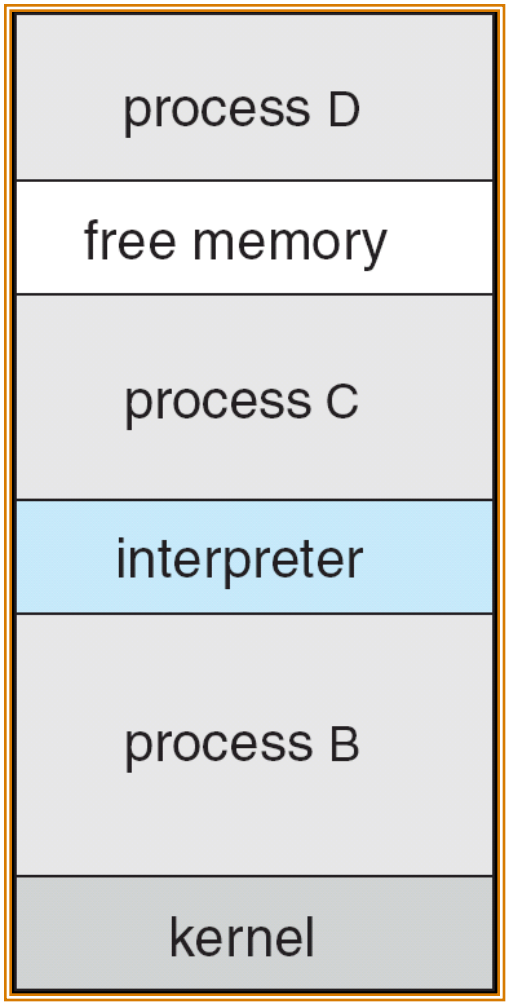
- Process control
- File management
- Device management
- Information maintenance
- Communications

MS-DOS execution



(a) At system startup (b) running a program

FreeBSD Running Multiple Programs



System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

System Programs

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Programs (cont'd)

- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Operating System Design and Implementation

- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont.)

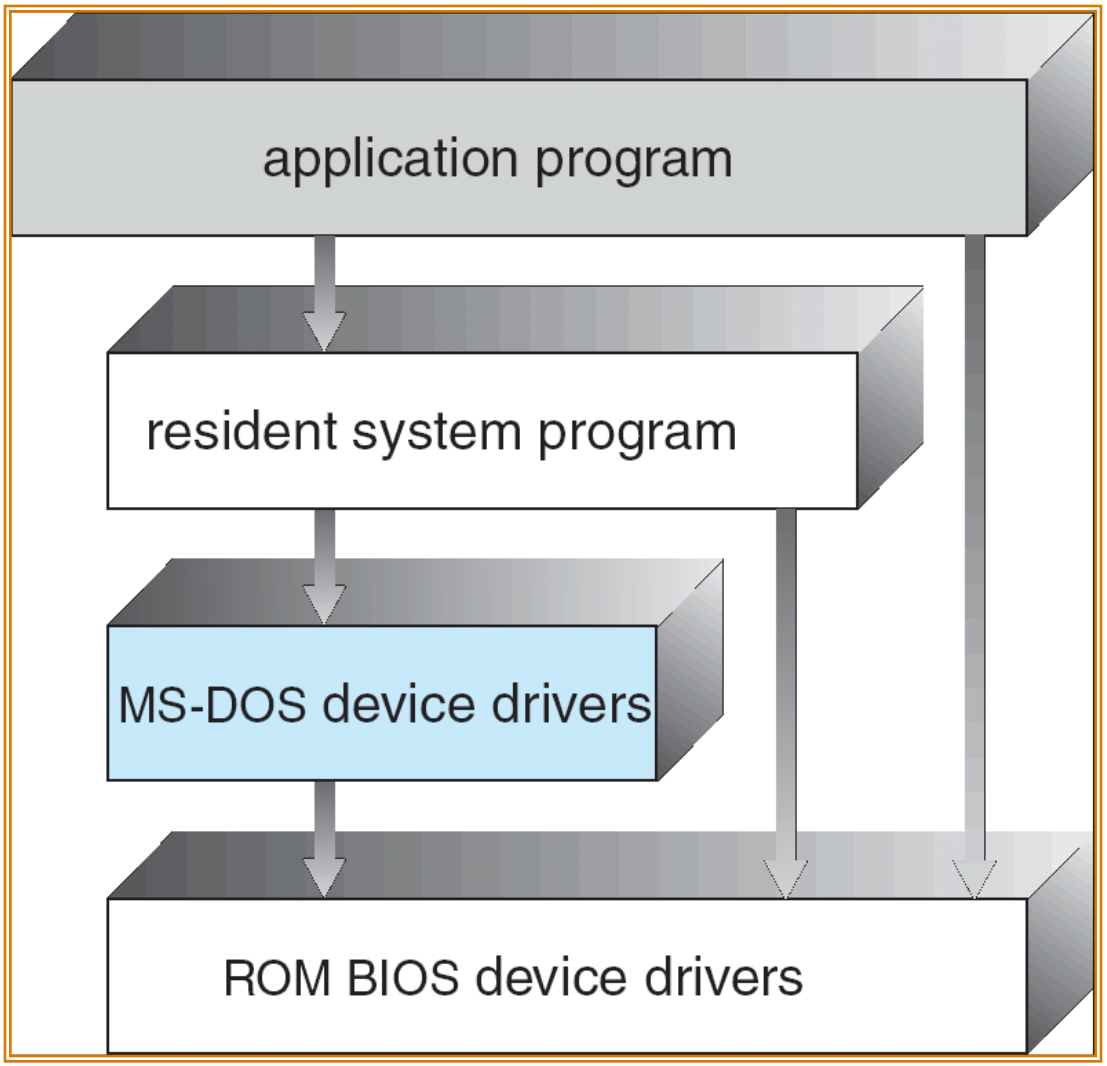
- Important principle to separate

  Policy: What will be done?
  Mechanism: How to do it?
- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

Simple Structure

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

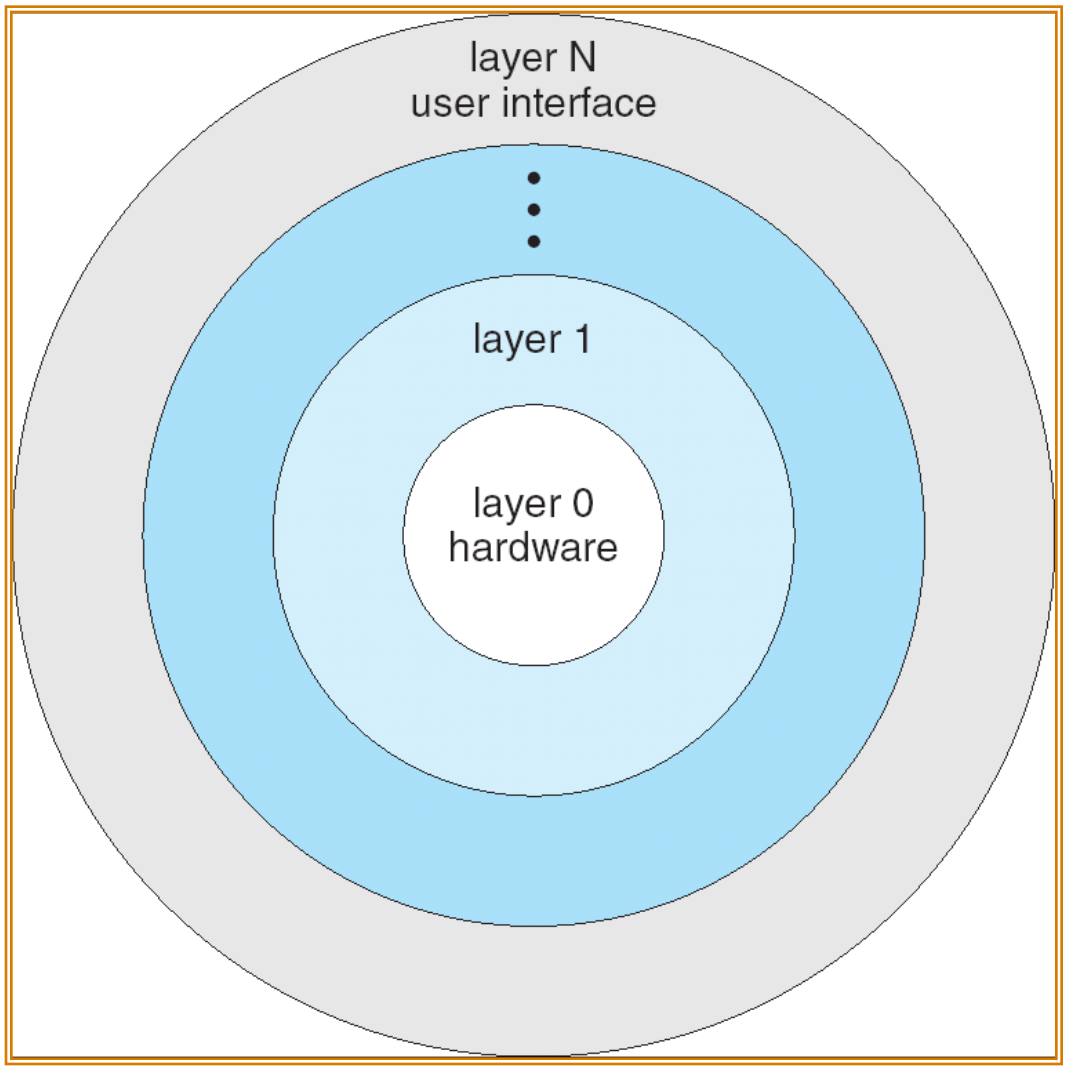
MS-DOS Layer Structure



Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

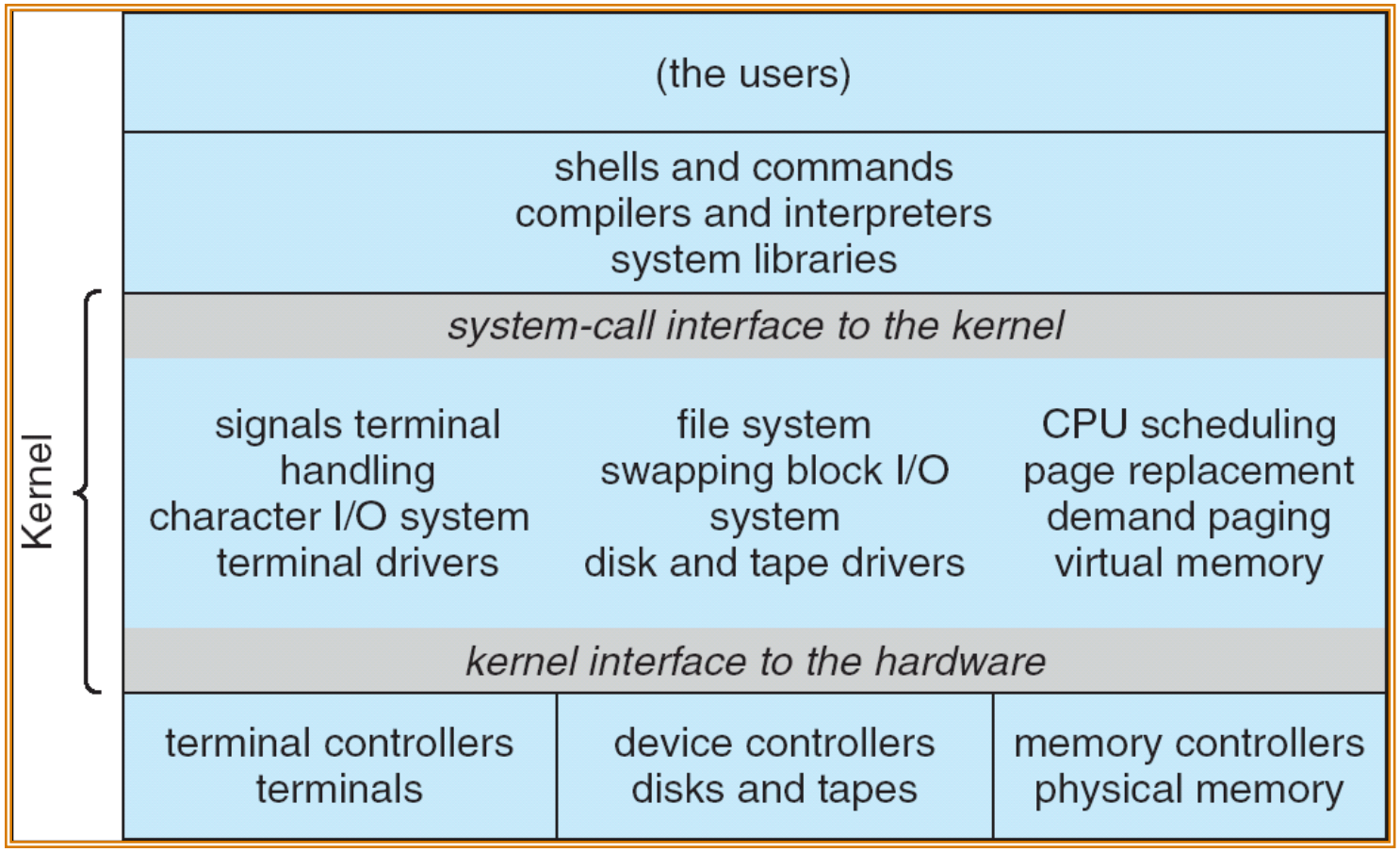
Layered Operating System



UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

UNIX System Structure



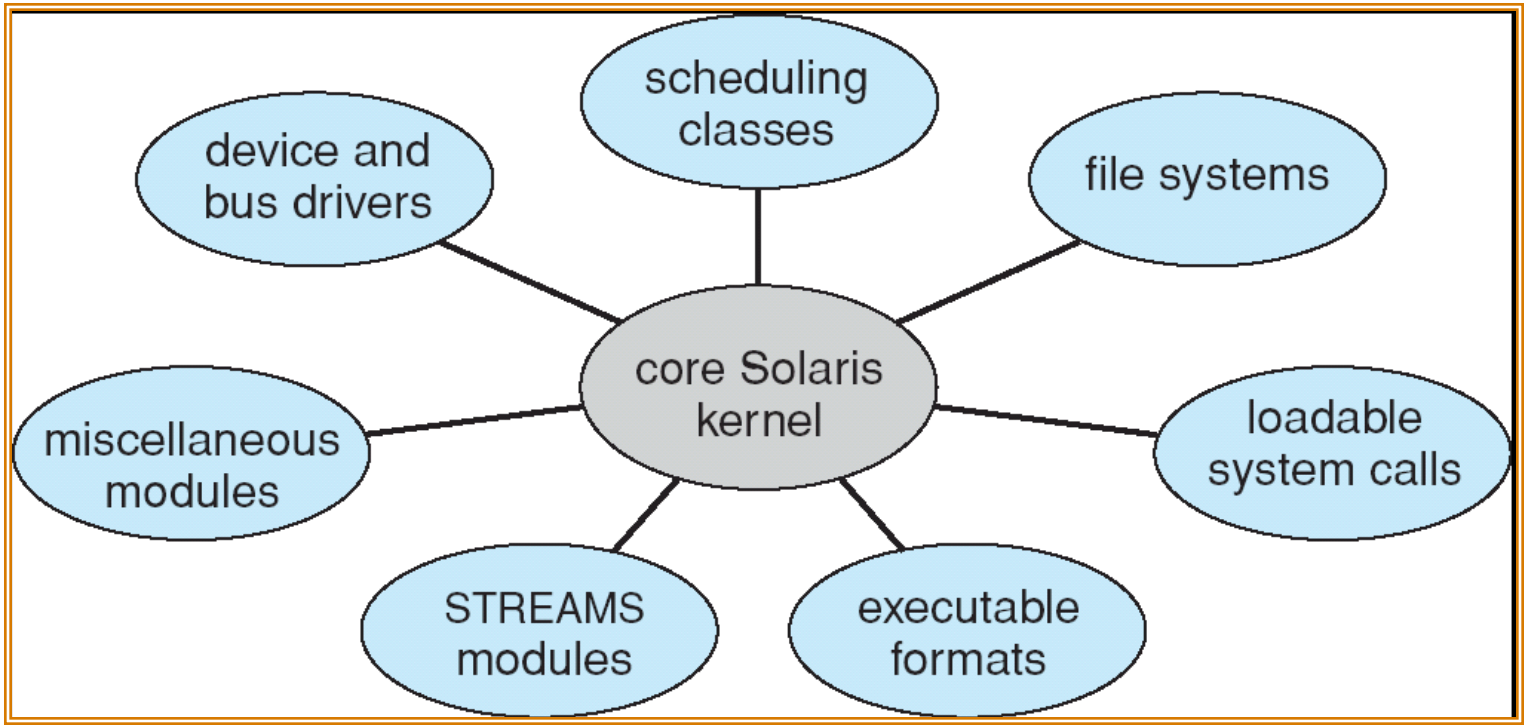
Microkernel System Structure

- Moves as much from the kernel into "*user*" space
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

Solaris Modular Approach



Virtual Machines

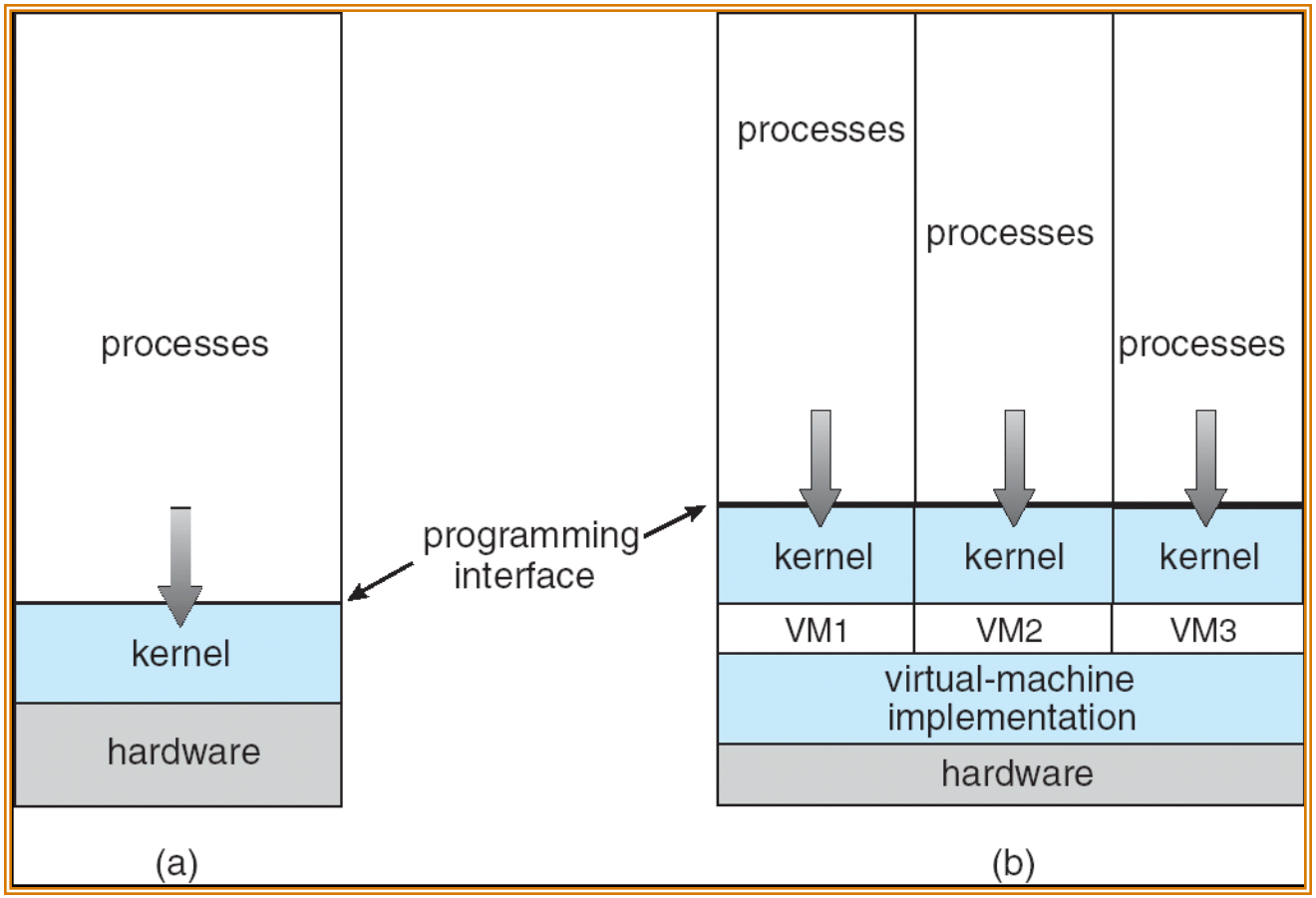
- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Spooling and a file system can provide virtual card readers and virtual line printers
  - A normal user time-sharing terminal serves as the virtual machine operator's console

Virtual Machines (Cont.)

Non-virtual Machine

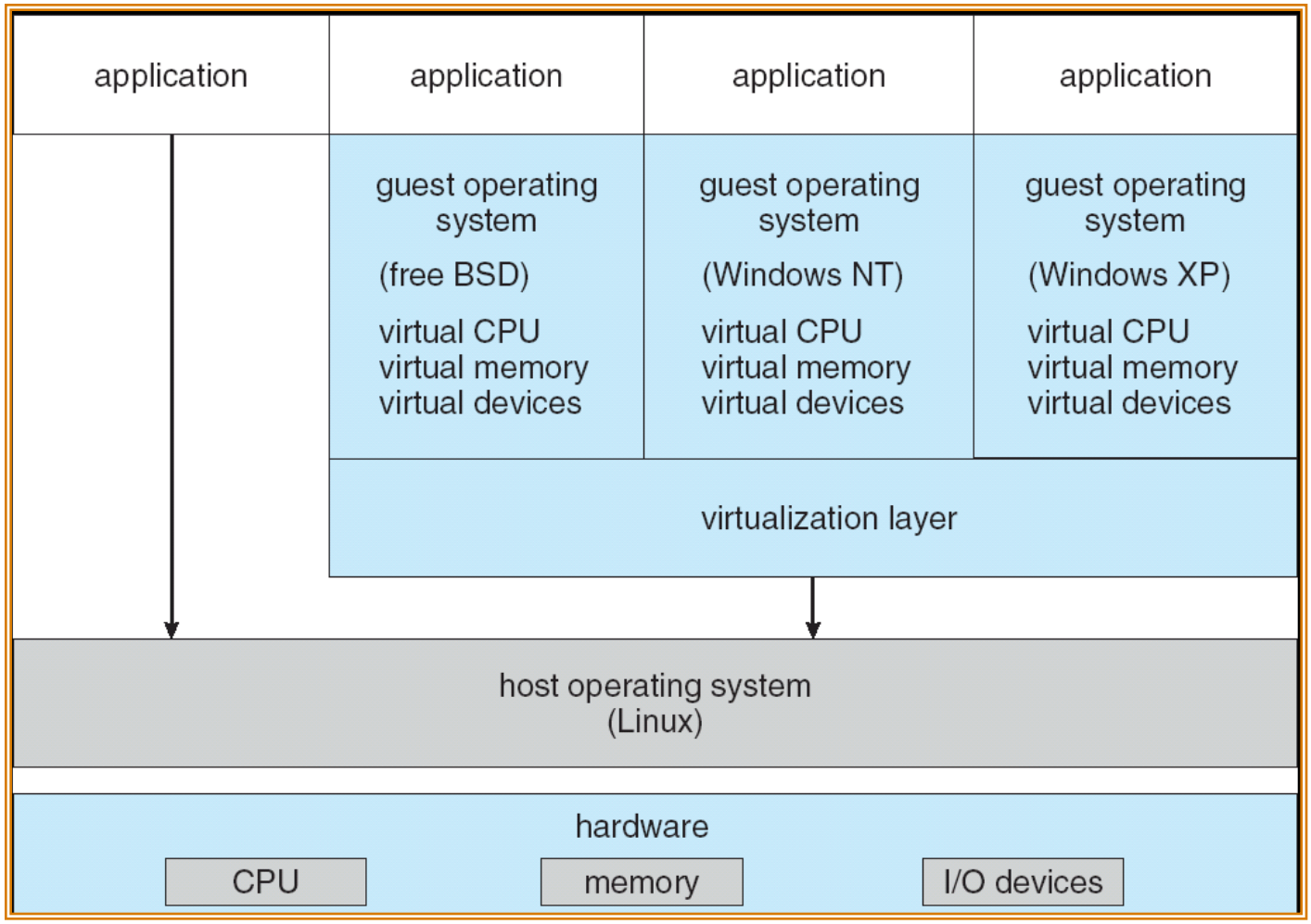


(a) Nonvirtual machine (b) virtual machine

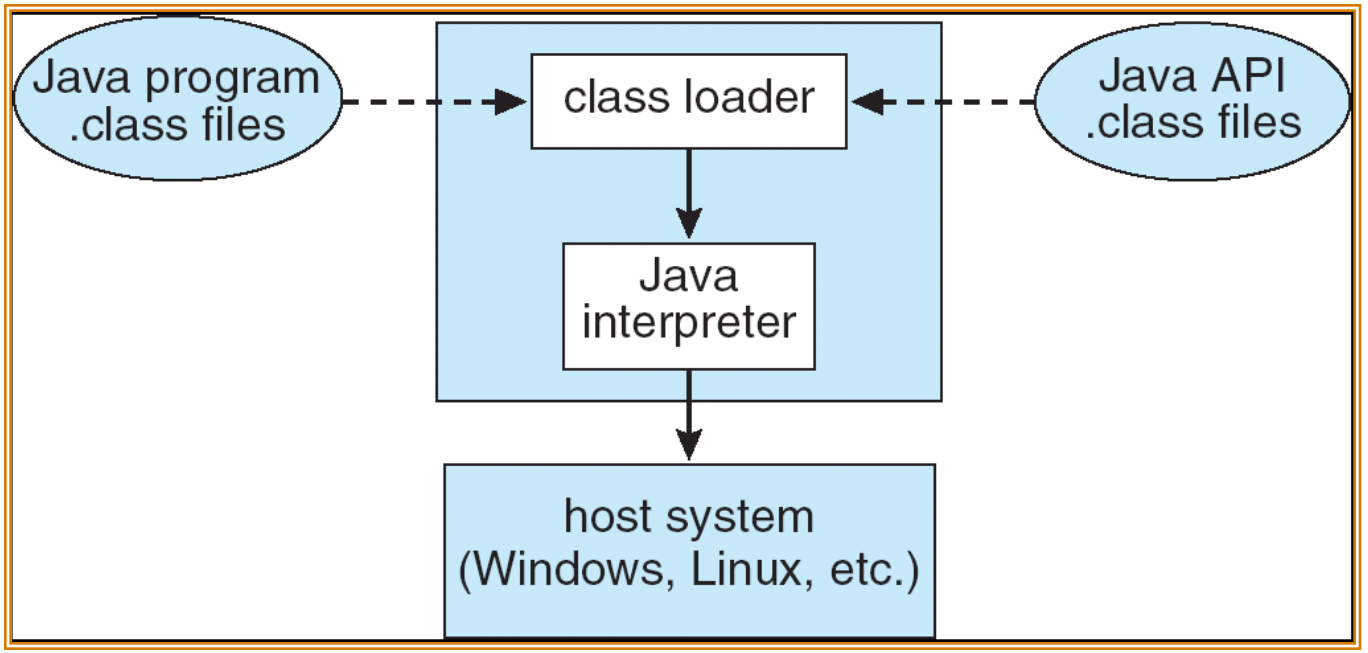
Virtual Machines (Cont.)

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

VMware Architecture



The Java Virtual Machine

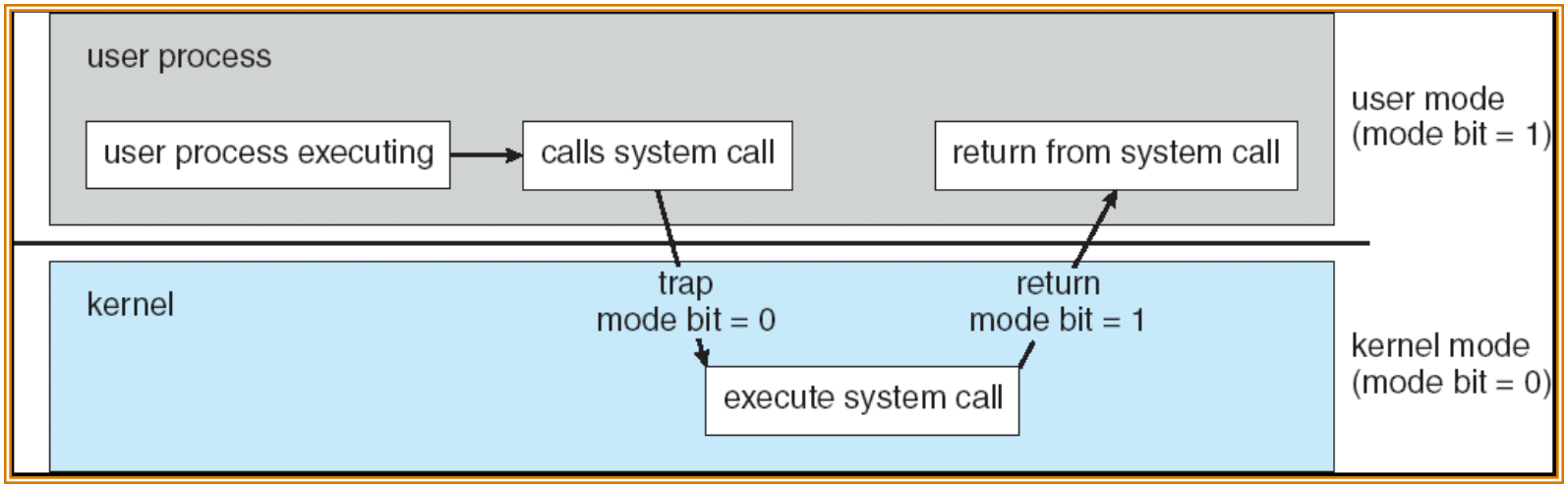


Operating-System Operations

- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long" period of time.
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (g**roup ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights

Distributed Operating Systems

1. Design Issues

   - structure, IPC, Resource Management, Security, Naming,

3. Case Studies

   - LOCUS, Amoeba, V System, Mach, x-Kernel
   - Advanced Systems: Somberero, 2K, et al.

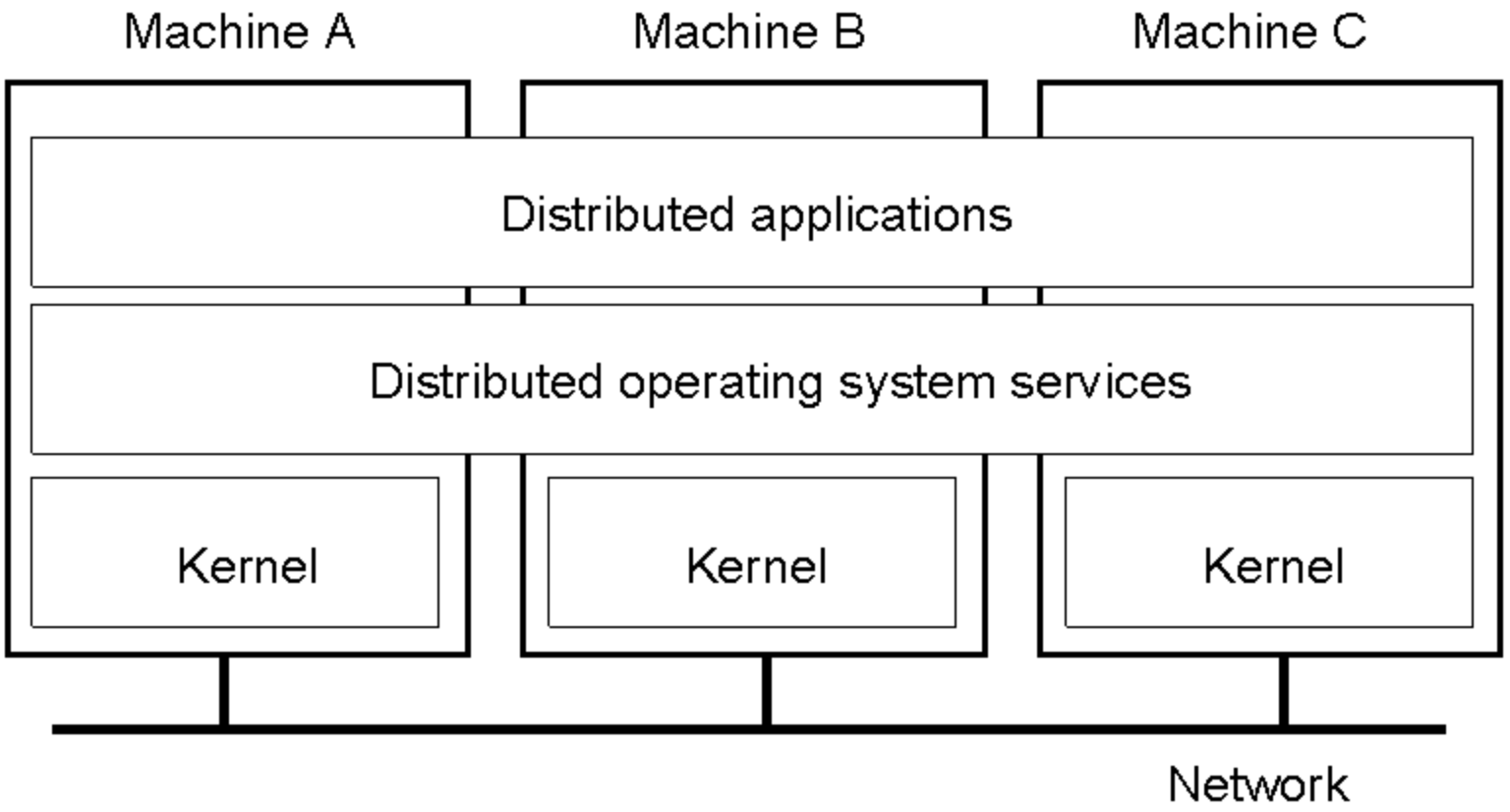
Types of Distributed OSs

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

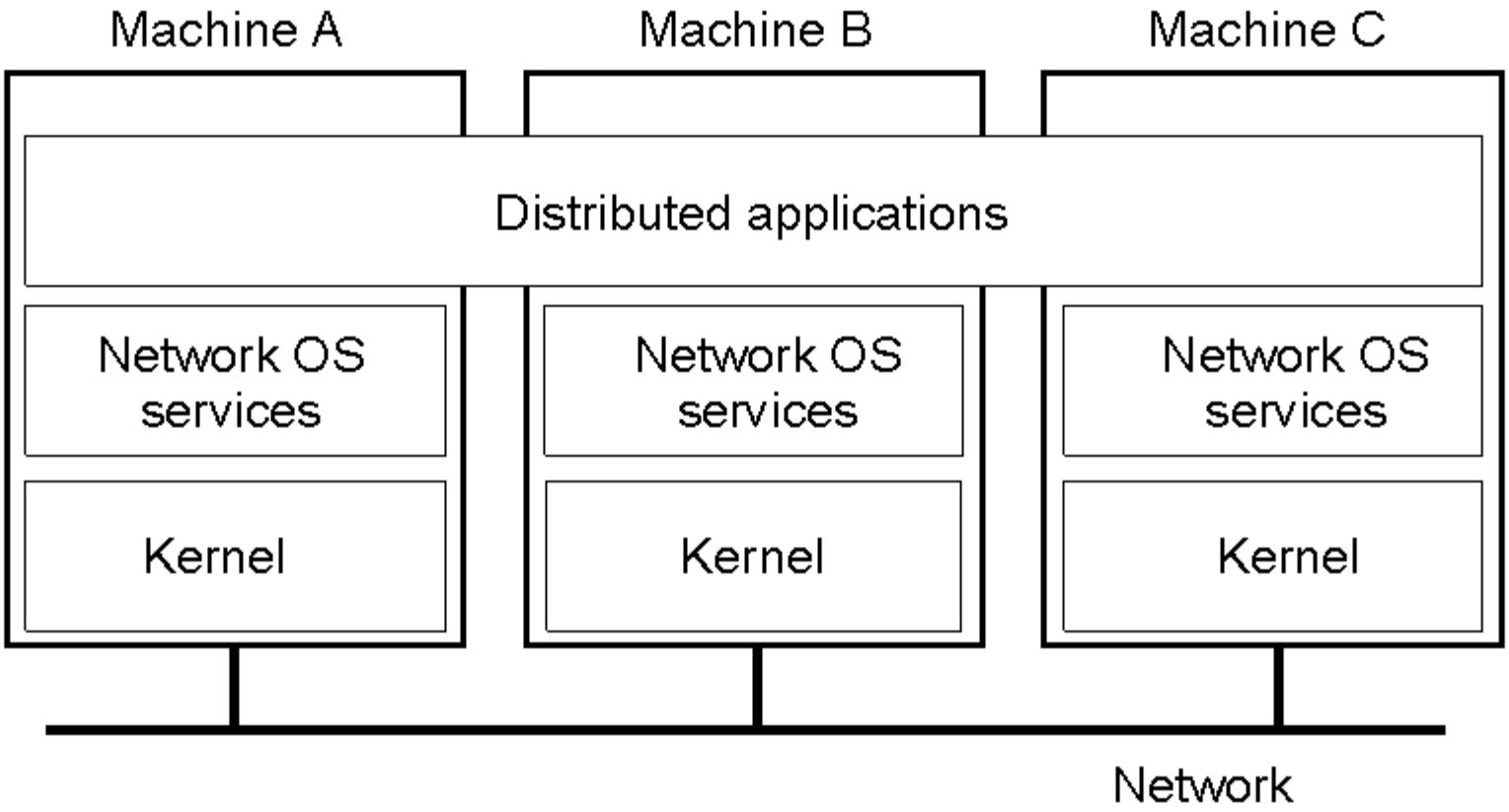
Multiprocessor Operating Systems

- Like a uniprocessor operating system
- Manages multiple CPUs transparently to the user
- Each processor has its own hardware cache
  - Maintain consistency of cached data

Multicomputer Operating Systems

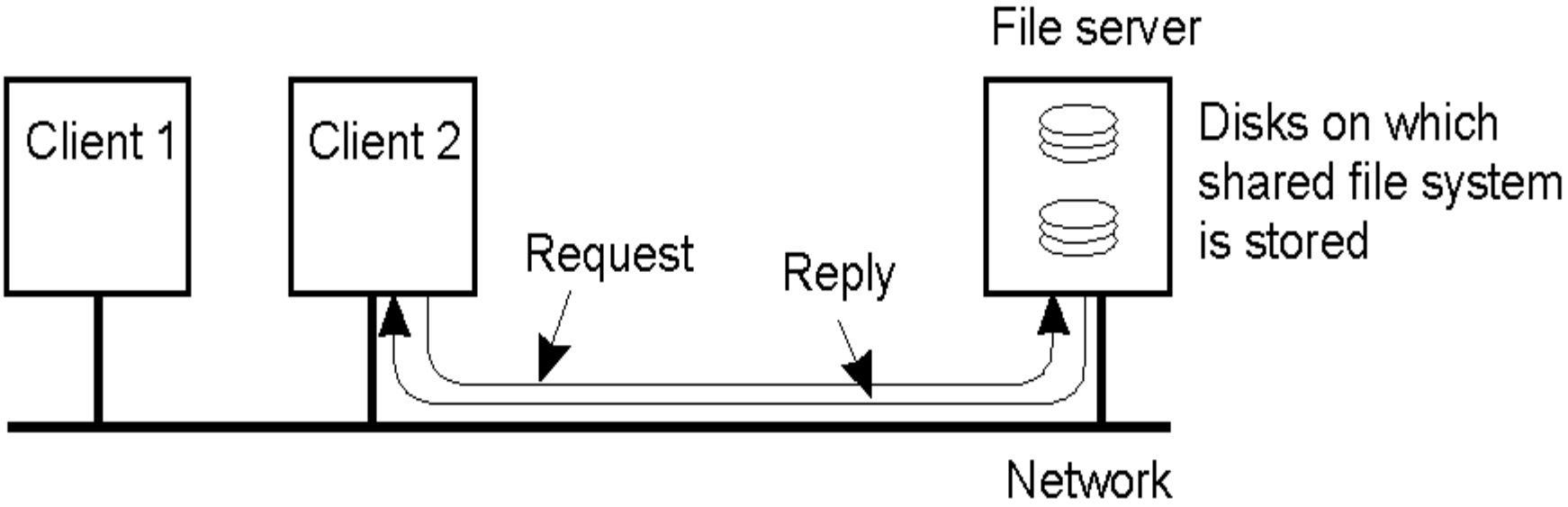


Network Operating System



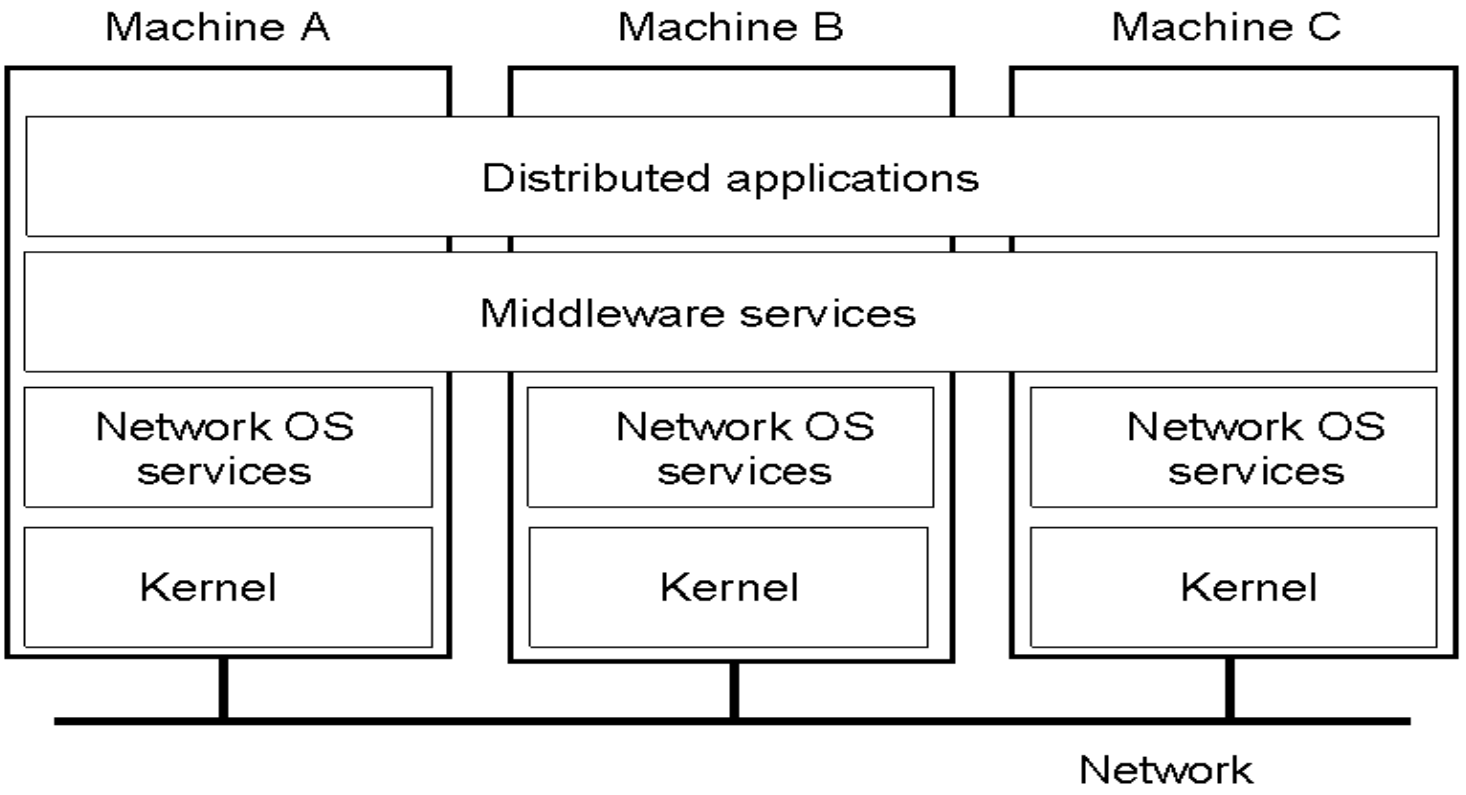
Network Operating System

- Employs a client-server model
  - Minimal OS kernel
  - Additional functionality as user processes



Middleware-based Systems

- General structure of a distributed system as middleware.



Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Distributed Operating Systems Implementations

1) Monlithic Kernel : All operating system services implemented in one big monolithic kernel

2) Micro Kernel : The operating system services are structured as a collection of independent processes

3) Object Oriented Operating System :

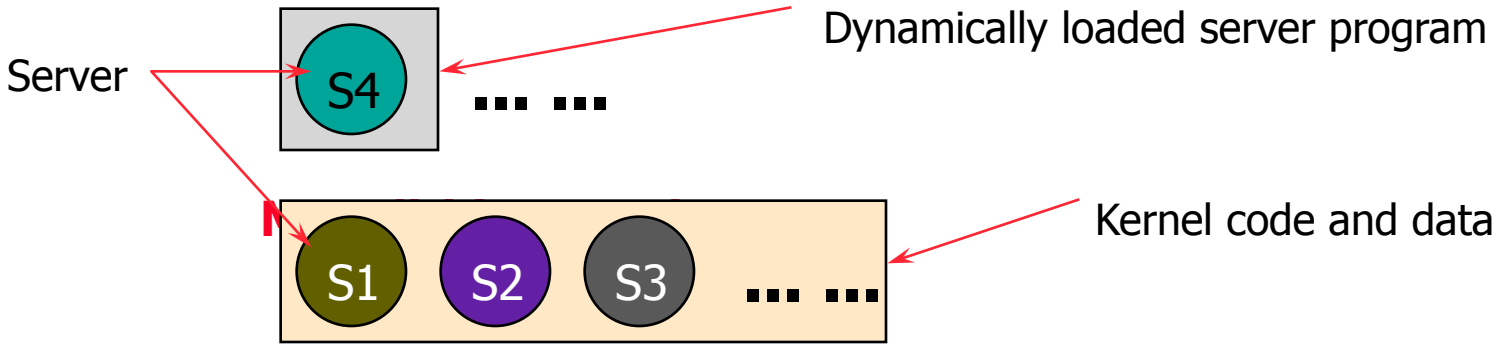
- Previous techniques realize OS services as a set of processes
- Here the OS services are implemented as objects (data structures and operations)

Distributed Operating System Implementation

1. Monolithic Approach

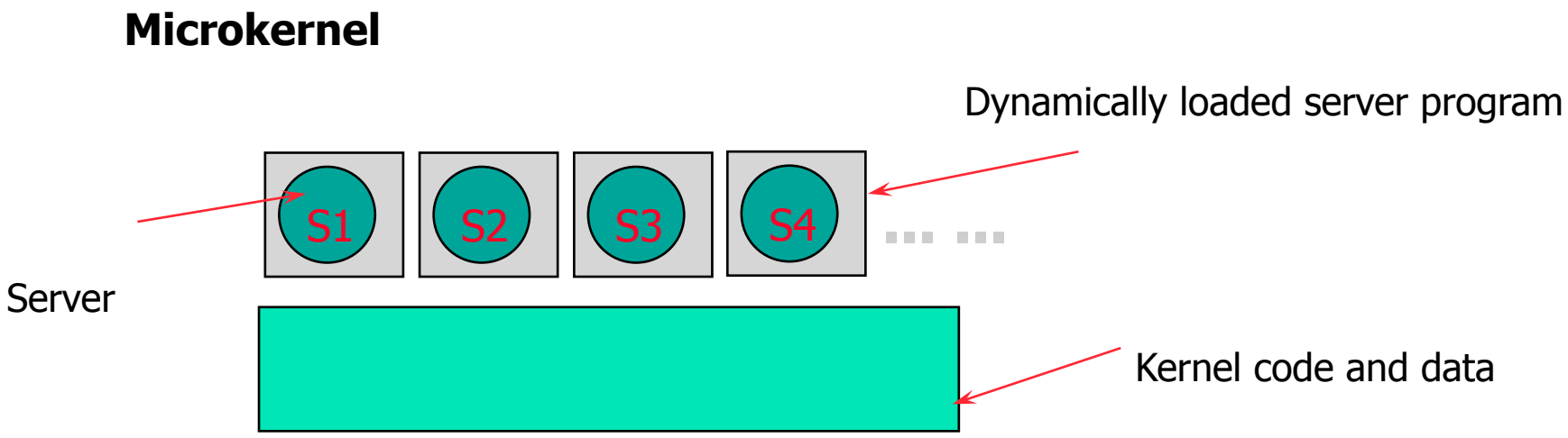
- UNIX, Sprite
- Every system functions included in the DOS
- It is very complex and large (1Mbytes)

- Intractable
- Efficient process execution



Microkernel

- Only the basic services: process management, memory management, interprocess communication
- Other services are provided at user levels and dynamically loaded to relevant computers
- It is open and modular architecture
- Small kernel makes it easy to debug and be more efficient



Microkernel System Structure

- Moves as much from the kernel into "*user*" space.
- Communication takes place between user modules using message passing.
- Benefits:

  - easier to extend a microkernel

  - easier to port the operating system to new architectures

  - more reliable (less code is running in kernel mode)

  - more secure

DoS Design Issues

- Communication Primitives
  - Message Passing, RPC, DSM, etc
  - Support for error handling in case of communication failure

- Naming
  - Name-location mapping can be centralized, partitioned or replicated
  - Name space can be hierarchical or flat
  - Typically, there is a name server to provide the mapping.

Design Issues Contd...

- Resource management
  - It is impossible to gather information about utilization or availability of resources coherently.
  - Hence these have to be calculated approximately using heuristic methods.
- Processor allocation
  - Load balancing
  - Hierarchical organization of processors.
  - If a processor can’t handle a request, ask the parent for help.
  - Issues: Crashing of a higher level processor will result in isolation of processors attached to it.

Design Issues (contd.)

- Process scheduling
  - Communication dependency has to be considered.
- Fault tolerance
  - The design should consider distribution of control and data.
- Services provided
  - Typical services include name, directory, file, time, etc.

Design Issues: Inter-process Communications

- Inter-Process Communication Types:
  - Message Passing
  - Remote Procedure Calls
  - Shared Memory

Inter-Process Communication: Message Passing

- There are three fundamental design issues that must be addressed in designing an IPC scheme
  - blocking or non-blocking,
  - reliable or unreliable,
  - buffered or un-buffered primitives

Inter-Process Communication: Message Passing

- minimize the number of communication primitives
- adopt request/response communication model
  - only three communication primitives (e.g., Ameoba).
  - The first primitive is used to send a request to a server and wait for the reply from the server
  - The second primitive is to receive the Client requests
  - The third primitive is to send the server reply message to the client after the request has been processed.

Inter-Process Communication: Remote Procedure Calls

- It has been widely accepted and used.
- using procedures to transfer control from one procedure to another is simple and well understood in high-level language programs running on a single computer.
- This mechanism has been extended to distributed operating systems
- RPC Design Issues
  - how to pass parameters from the calling procedure to the called procedure.
  - We could pass parameters by using either value or reference techniques.
    - Passing parameters by value is easy
    - passing parameters by reference is more complicated because we do need to have a unique and system-wide pointers
    - Passing parameters get even more complicated if different types of computers are used

Naming Concepts

- Name
  - What you call something
- Address
  - Where it is located
- Route
  - How one gets to it

- **But it is not that clear anymore, it depends on perspective.**
  - **A name from one perspective may be an address**
  - **from another Perspective means layer of abstraction**

What is http://www.ece.arizona.edu/~hpdc ?

What are the things we name

- Users
  - To direct, and to identify
- Hosts (computers)
  - High level and low level
- Services
  - Service and instance
- Files and other “objects”
  - Content and repository
- Groups
  - Of any of the above

How we name things

- Host-Based Naming
  - Host-name is required part of object name
- Global Naming
  - Must look-up name in global database to find address
  - Name transparency
- User/Object Centered Naming
  - Namespace is centered around user or object
- Attribute-Based Naming
  - Object identified by unique characteristics
  - Related to resource discovery / search / indexes

Naming Service

- It is a mapping between two domains
  - in a centralized system, we have one directory to perform the mapping
- Naming Servers
  - **Centralized Name Server:**
    - In this approach, we have a single name server that accepts names in one domain and simply maps them to another name normally understood by the system
    - (e.g., In UNIX environment, this represents the mapping of an ASCII file name into its I-node number).
    - A server or a process needs first to register its name in the database to publicly advertise the availability of the service offered by that server or process.

Naming Service- Cont.

- **Hierarchical Name Server:**
  - divide the system into several logical domains
  - each logical domain maintains its own mapping table.
  - similar mapping in telephone network in which a country code followed by an area code followed by an exchange code all precede the actual user phone number.
  - an object can be located or found based on which domain or subdomain it resides in
  - the name server of its domain or subdomain will perform the mapping and locate the requested object or process.
- Use a set of pairs that will point to either the physical location of the object or the next name that might contain the physical location of the object.

Naming Service- Cont.

- **Distributed Name Server:**
  - allow each computer or resource to implement the name service function.
  - each machine on the network will be responsible for managing its own names.
  - Whenever a given object is to be found, the machine requesting the name will broadcast the name on the network, if its not known in its mapping table.
  - Each machine on the network will then search its local map to determine if there is a match.
  - If a match exists, then a reply is sent to the requesting machine.
  - If no match is found, then no reply is made.

Resource/Process Management

- Resource management is concerned with making both local and remote resources available to a user in an efficient and transparent manner
  - the location of the resources is hidden from the user
  - remote and local resources are accessed in the same manner.
- In a centralized computing system, the scheduler has full knowledge of the processes running in the system
- In a distributed system, there is no centralized table and even if it exists, it is very difficult to keep that database up-to-date.
- Lack of accurate global system status and load, make process scheduling and resource management is a challenging task
  - Allocate processors to processes
  - Schedule/balance loads
  - Migrate processes

Resource Management- Processor Allocation

- In the case of a processor pool model,
  - When processes are submitted there is some knowledge of the process type;
  - whether CPU intensive, I/O intensive, its memory or resource requirements, etc.
  - The resource manager then determines the number of processors to be allocated to the process
- one can organize the system processors into a logical hierarchy: master and workers
  - A single master may oversee multiple workers and keep track of their status
  - when a job is submitted, the master decides how the process may be split up among the workers.
  - When several masters are used, one can designate a node or a committee of nodes the responsibility of managing all the masters in the system
  - When a manager fails, one of the worker processors should be designated to carry the responsibilities of the failed manager.

Resource Management - Scheduling

- Scheduling is to maximize the utilization of the system resources and improves performance
  - Schedule to improve reliability/availability
  - Schedule to improve security, etc.
- Scheduling is even more important in distributed applications
- the scheduler needs to allocate the communicating tasks simultaneously on different processors in order to minimize the communication delays

Process Scheduling

	1	2
T	***T1***	***X***
2T	***X***	***T2***
3T	***X***	***X***
4T	***T1***	***T2***
5T	***X***	***X***

Resource Management - Load balancing and scheduling

- It aims at improving the performance by moving data, computation, or processes
- Load balancing has a more stringent requirement than scheduling
- Load balancing/scheduling techniques can be achieved by either migrating data, computation or processes.
- Process migration incurs high overhead

File Service

- There are three file system functions:
  - disk service, flat file service, and directory service.
- The file system characteristics depend on how these functions are implemented
  - One extreme approach is to implement all the functions as one program running on one computer.
  - The other extreme is to implement each function independently so we can easily support different types of disk and file systems.
    - this approach is inefficient because these modules will communicate with each other using the inter-process communication services.

File Service- Cont.

- A common file system is typically provided by file servers
  - lower system costs, facilitate the sharing of data among users, simplify the management and administration of the file system
- file service can be distributed across several servers to provide a distributed file service (DFS)
  - DFS should look to its users as a conventional file system
  - The multiplicity of data and the geographic dispersion of data should be transparent to the users.
  - DFS introduces new problems that the distributed operating system must address such as concurrent access, transparency and availability of the file service

Fault Tolerance

- Distributed systems are potentially more fault tolerant than a non-distributed system
- Fault tolerance enables a computing system to continue its operations successfully in-spite some failures in system components (hardware or software resources)
- Fault intolerance system crashes when some failure occurs
- There are two approaches to achieve fault tolerance:
  - Redundancy
  - Atomic transaction

Fault Tolerance - Redundancy

- Redundancy is the most widely used technique
  - detecting faults once they occur,
  - locating and isolating the faulty components, and then recover from the fault(s)
  - fault detection and recovery are the most important and difficult to achieve
- Types of failures which may occur in the system are:
  - hardware errors are of a Boolean nature
  - Intermittent faults
  - Software errors can be broken down into
    - Specification errors
    - Programming errors refer to the case where the program fails its specification
  - Byzantine Faults – malicious faults

Fault Tolerance - Atomic Transactions

- They refer to operations with a guarantee that
  - operations will either be performed successfully until completion or
  - non of them is executed and the system is restored to its initial state.
- To achieve atomic operations, the system relies
  - careful read and write operations,
  - stable storage, and
  - stable processor

Fault Tolerance - Atomic Transactions

- **Careful Disk Operations:**
  - CAREFUL_WRITE operation
    - the data block is written to the disk
    - a READ service is called to immediately read back the written block to make sure it was not written to a bad spot on the disk.
    - If found persistent discrepancy after predetermined number of times, the disk spot is declared bad.
    - Even, after writing correctly to the disk, the written spot can go bad.
  - This is normally detected by checking the parity check field of any read block during the CAREFUL_READ operation.

Fault Tolerance - Atomic Transactions

- **Stable Storage:**
  - attempt to mirror all data on more than one disk
  - minimize the amount of data lost in the event of a disk failure.
- Writing to the stable storage abstraction
  - first attempts a CAREFUL_WRITE to the primary disk
  - If the operation completes without error, a CAREFUL_WRITE is attempted on the secondary disk
  - If, on corresponding disk blocks the blocks are the same and GOOD, nothing further needs to be done with these blocks.
  - On the other hand, if one is BAD and the other is GOOD,
  - the BAD block is replaced by the data from the GOOD block on the other disk.
  - If the disk blocks are both GOOD but the data is not identical, the data from the primary disk is written over the data from the secondary disk.

Fault Tolerance - Atomic Transactions

- **Stable Processor:**
  - processes may checkpoint themselves to stable storage periodically
  - in the event of the processor running the process crashes,
  - the process may restore its last ckeckpointed state by reading the stable storage
- An atomic transaction can be implemented as:
  - When a process wishes to make changes to a shared database, the changes are recorded in stable storage as an intention list.
  - When all of the changes have been made, the process issues a commit request.
  - The intention list is then written to memory.
  - By using this method, all the intents of process are in stable storage and at any time during disk update,
    - a crash can be recovered from by simply examining the contents of any outstanding intention list in memory each time a processor is brought up.

Security and Protection

- Security and protection are necessary to avoid unintentional or malicious attempts to harm the integrity of the distributed system.
- Security is implemented using techniques that are based on the following principles:
  - What you have
  - What you know
  - What you are

Security Goals

- Confidentiality
  - No one not authorize can see or access data
  - Authentication
    - Determining identity of principal
- Integrity
  - Authenticity of document
  - That it hasn’t changes
- Availability
  - Whenever you need to access your data, you can
  - DoS attacks affect availability of services

Security Issues:

- Two security issues must be dealt with:
  - authentication and authorization.
- **Authentication:** Authentication is making sure that an entity is what it claims to be.
  - Password protection is sufficient in general systems
  - high security systems might resort to physical identification or voice identification, cross examination, or user profiling to authenticate a user.
- **Authorization:** granting a process to do something based on the privileges it has.
- Privileges of a user over an entity may be expressed either as
  - Access Control Lists (ACLs)
  - Capabilities.

Security Policy

- Access Matrix

Subject	*OBJ1*	*OBJ2*
bcn	RW	R
gost-group	RW	-
obraczka	R	RW
tyao	R	R
Csci555	R	-

  - implemented as:
    - Capabilities or
    - Access Control list

Access Control Lists

- Advantages
  - Easy to see who has access
  - Easy to change/revoke access
- Disadvantages
  - Time consuming to check access
- Extensions to ease management
  - Groups
  - EACLs

Extended Access Control Lists

- Conditional authorization
  - Implemented as restrictions on ACL entries and embedded as restrictions in authentication and authorization credentials

Principal	*Rights*	*Conditions*
bcn	RW	HW-Authentication Retain Old Items
gost-group	RW	TIME: 9AM-5PM
authorization server	R	Delegated-Access
*	R	Load Limit 8 Use: Non-Commercial
*	R	Payment: $Price

Capabilities

- Advantages
  - Easy and efficient to check access
  - Easily propagated
- Disadvantages
  - Hard to protect capabilities
  - Hard to revoke
- Hybrid approach
  - EACL's/proxies

Protecting capabilities

- Stored in Hidden Place
  - Only protected calls manipulate
- Limitations ?
  - Works in centralized systems
- Distributed Systems
  - Tokens with random or special coding
  - Possibly protect through encryption

Distributed Operating Systems: Case Studies

1. Design Issues

- Network Operating System v.s. Distributed Operating Systems
- Design Issues

2. Case Studies

- LOCUS
- Amoeba
- V System
- Mach
- x-Kernel

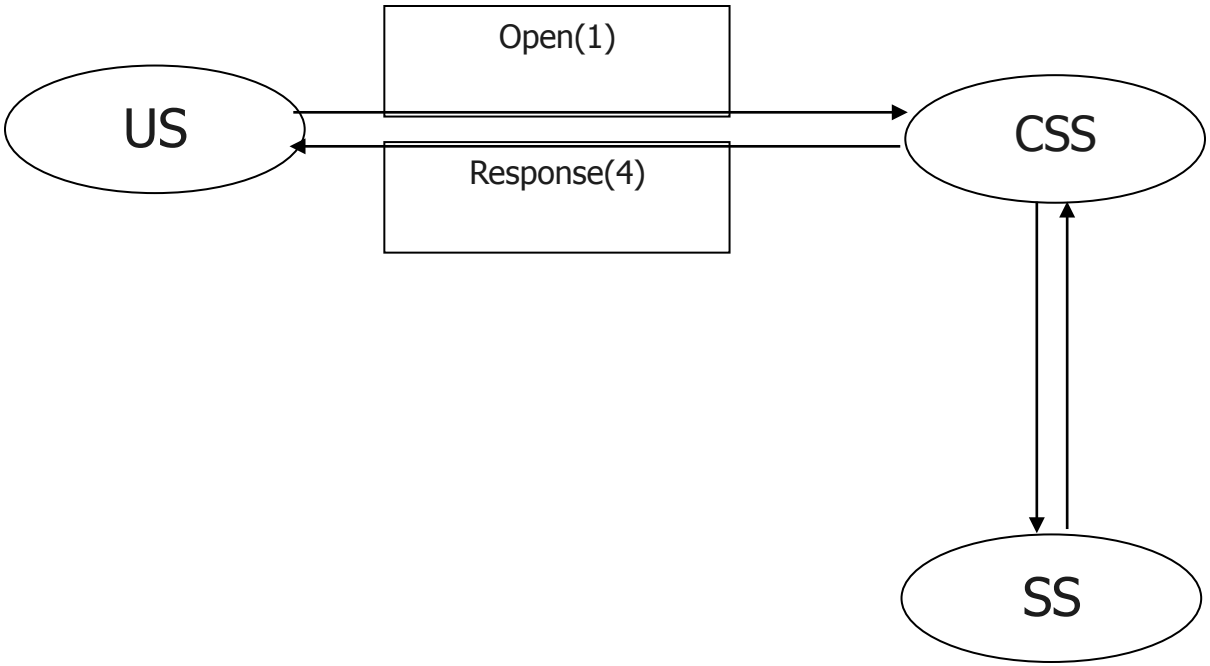
Locus : Main Goal & Features

- Main Goal
  - Distributed and reliable Unix - like operating system
  - High degree of location transparency, concurrency, replication and failure
- Features
  - Automatic replication of data under user control
  - User might be needed to produce a consist system after a network partitioning caused by computer or network failures

Locus : Descriptions

- It is compatible with Unix operating system (BSD and system V)
- It support DEC Vax/750, DEC PDP 11/45 and IBM PC connected by Ethernet or token ring
- Full Unix semantics and emulated Single tree structure
- File replication is made using multiple physical containers for each logical file group
- Atomic commit using shadow page used for redundancy management

Logical File Access



Amoeba : Main Goal & Advantage

- Main Goal
  - Develop a capability - based, object - based distributed operation system
  - make a large collection of machines (larger than the number of users) behave as a centralized time - sharing system
- Advantage
  - Transparency and high performance
  - Main disadvantages are inability to run existing binary Unix programs and the lack of virtual memory

Amoeba : Description

- It is a microkernel based operating system.
- The microkernel runs on each machine and handles communication, I/O, low-level memory, and process management
- Other O.S. services are provided using servers running in user mode
- Most computing is done on the processor pool :

  a collection of CPUs that are dynamically allocated to jobs and then released
- Specialized servers (File servers and directory servers) handle certain system functions
- Each object has a global unique name, expressed through its capability
- The port number is generated randomly using 48 bits and stored in directories managed by a directory service
- To locate the service corresponding to a port, the client's kernel broadcasts a "local" packet

Amoeba : Description (cont)

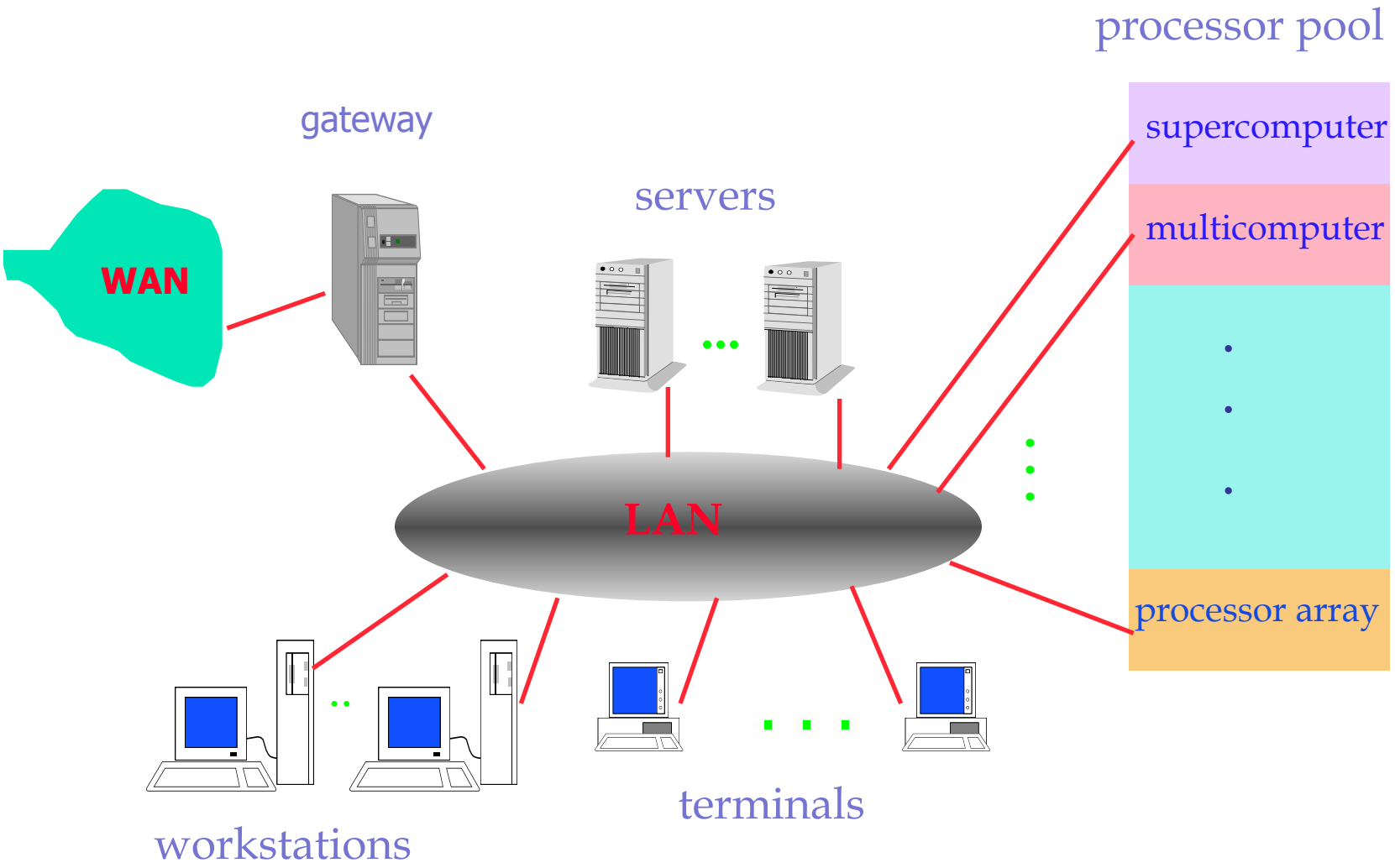
- Bullet server supports immutable files that are stored contiguously on the disk. It achieves high throughput (800 Kbytes/sec)
- Directory server maps ASCI strings onto capabilities
- Run server handles requests to run a process and assigns it to the pool processor with the lightest load
- Object server handles replication in order to support fault tolerance. Processes that need fault tolerance need to register that request with the boot server
- Other servers include TCP/IP server, X-window server, I/O server, etc

Amoeba : Hardware Architecture

has four components: workstations, processor pool, specialized servers and gateways

- Processor pool provides most of the computing power
  - Processors can be allocated dynamically to user applications
  - the number of processors exceeds the number of users by an order of magnitude
- Gateways are used to access other Amoeba systems over a wide area networks
- Centralizing the computing power allows incremental growth, fault tolerance and ability to obtain a large amount of computing power temporarily

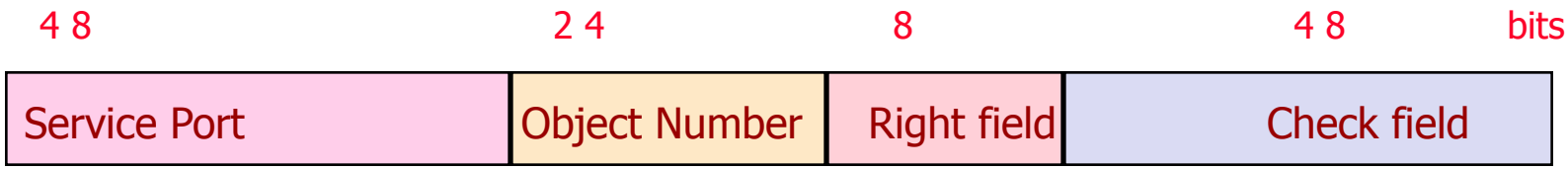
Amoeba : Hardware Architecture



Amoeba : Software Architecture

- Object-based system using clients and servers
- Each object is identified and protected by a capability
- It identifies the set of operations that the holder can perform on the object
- Guessing an object capability is infeasible because of using cryptographic protection
- Capability are kept secret by embedding them in a huge address space
- Given a capability, the system can easily find a server process that manages the corresponding object
- Kernel manages memory, supports processes with multiple threads and handle interprocess communication
- All other services are provided by user-level processes

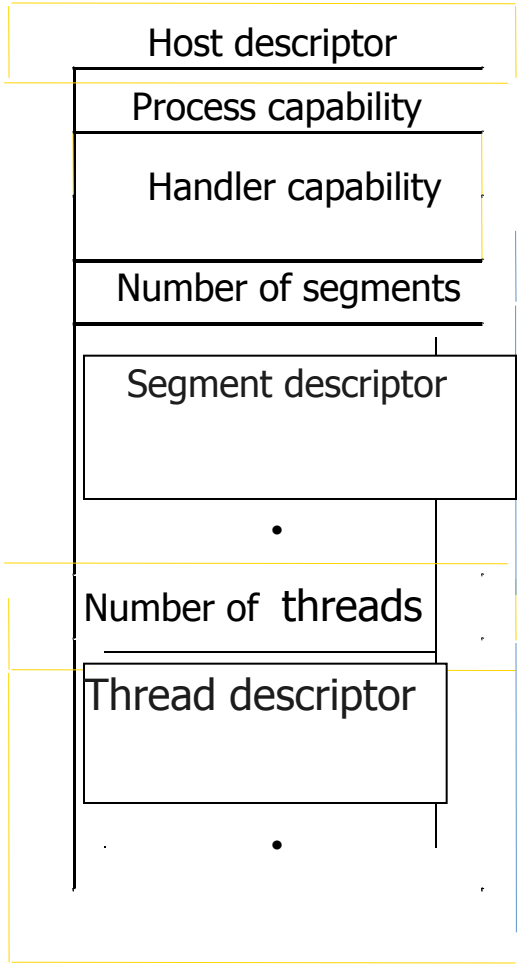
Amoeba : Software Architecture



Structure of a capability.

- The service port identifies the service that manages the object.
- The object number specifies the object.
- The rights field determines which operations are permitted.
- The check field cryptographic protects from users tampering with the other field

Process capability



Amoeba process descriptor

Amoeba : Communications

- It is based on a client thread (light-weight processes) performing operations on objects
- A client sends a request message to a server that manages the object
- A server thread, accepts the message, carries out the request, and sends the reply
- Multiple server processes could manage a collection of objects
  - increase performance and fault tolerance

Amoeba: Remote Procedure Call

- The kernel provides three basic operations:
- do_operation : used by client to get work done, send a message to server and block until reply comes back
- get_request : used by servers to announce their willingness to accept messages addressed to specific ports
- send_reply: sends results to clients
- A more user-oriented interface has been built on top of these three primitives

Amoeba: Remote Procedure Call

- The Amoeba Interface language compiler generates code to marshal or un-marshal the parameters into and out of message buffers and then call the transport mechanism

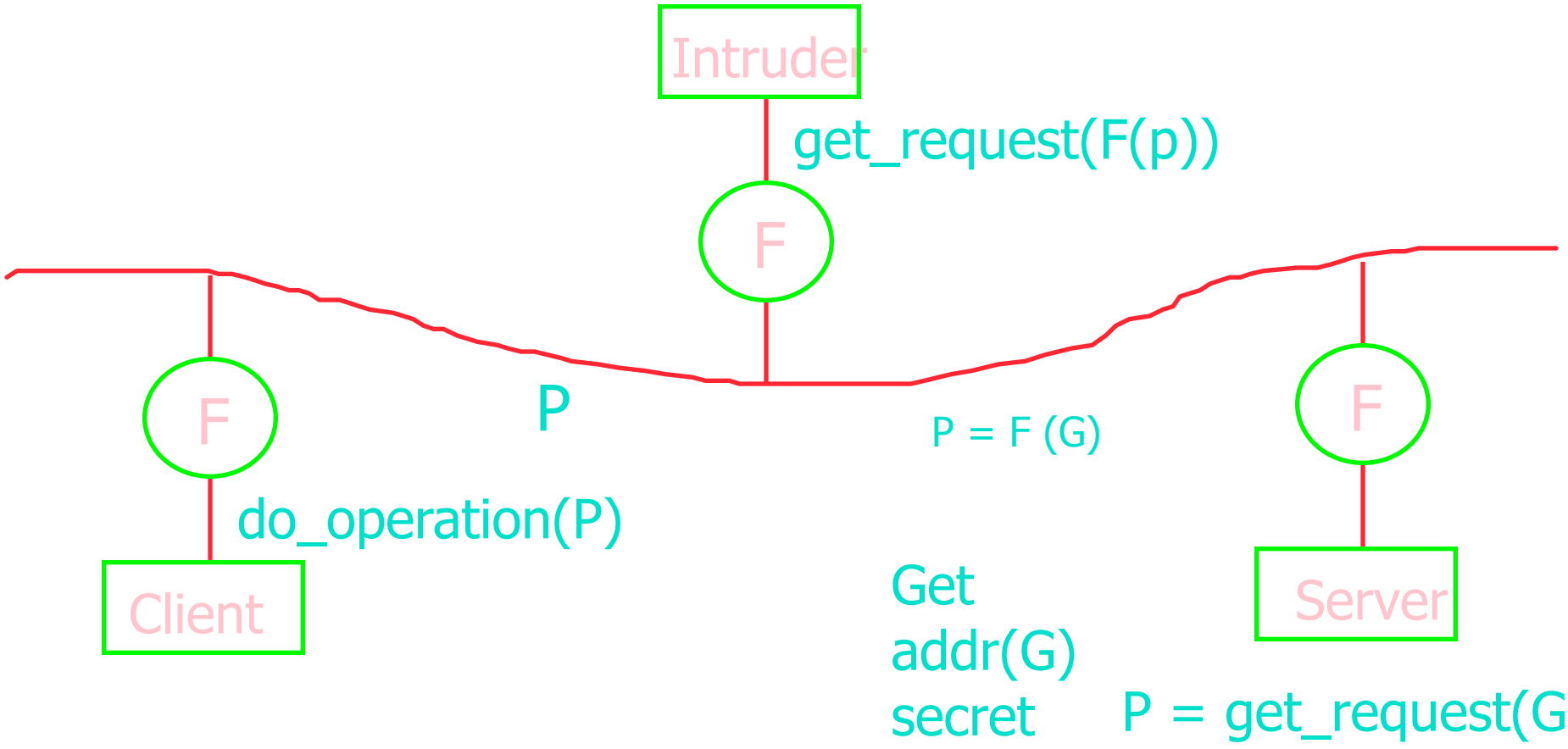
Amoeba : Locating Objects

- do_operation : If the kernel knows the server address, it can locate the object.
- If the kernel doesn't know the server address, it uses broadcast to get the address
- server perform : get_request

Amoeba : Secure Communications

- Knowing the port is taken by the system as evidence that the sender has a right to communicate with the service
- Amoeba has two level of protection:
  - ports for protecting access to servers
  - capabilities for protecting access to individual objects
- How do we secure communications?
  - Introduce a one-way filter that can be implemented in hardware or software

Amoeba : Secure Communications



Client, server, intruders and F-boxes

Amoeba : File Service

- Bullet Service: it stores all files contiguously both on disk and in the bullet server's memory
- No disk access is needed to fetch the i-node and one disk access is needed to access the whole file
- Supports three operations: read file, create file, and delete file => files are immutable
- Several advantages
  - fast retrieval
  - simplify administration
  - remove inconsistency patterns
- File can be transferred in one (single) RPC if less than 30K

Amoeba : Directory Service

- Bullet server has no high level memory services
- to access a file, a person must provide the relevant capability
- directory service maps an ASCII string into a capability

Object Name	Capability	Owner	Group	Other
	cap1	11111	11000	10000
games_dir	cap2	11111	10000	10000
paper.1	cap3	11111	00000	00000
prog.c	cap4	11111	11100	10000

A directory with three user classes

Mach Distributed Operating System

- An earlier roots go back to Rochester Intelligent gateway (RIG) in 1975 (a message passing OS)
- second generation was in 1979
- Mach third generation was in 1984
  - should be compatible with Unix (1986)
- Open Software Foundation (OSF) choose MACH 2.5 to be its first version of OSF/1.
  - a base to build other OS
  - support large sparse address space
  - transparent access to network resources
  - exploit parallelism in both system and applications
  - portable to a larger collection of machines

Mach : Main Goal & Advantages

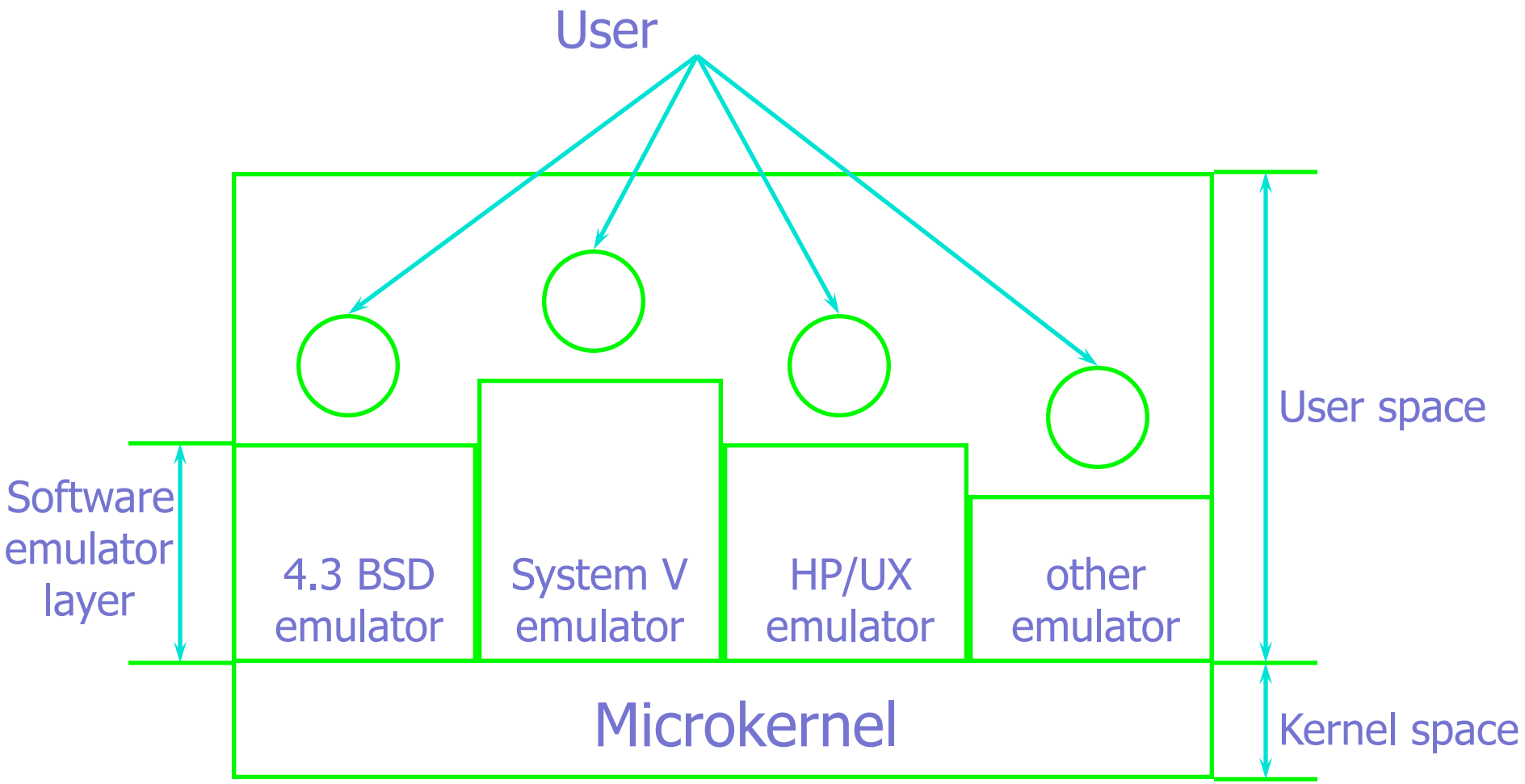
- Main Goal
  - Integrate distributed and multiprocessor functionality
  - Full binary compatibility with Unix BSD
- Advantages
  - Good environment for distributed and parallel applications because of efficient support to :
    * shared memory
    * message passing for uniprocessor and multiprocessors
    * smooth transition from Unix environment to MACH

System Model

- Mach is a micro-kernel based operating system.
- It has been designed to provide a base for building new operating systems and emulating existing ones (e.g., UNIX, MS-Windows, etc.)
- Mach is based on the concepts of processes, threads, ports, and messages.
- The MACH kernel provides the basic critical resource management tasks:
  - memory management,
  - scheduling,
  - device management,
  - inter-process communication.
- The emulated operating systems (e.g., UNIX systems) run on top of the kernel as servers or applications that provide users or applications the operating system environment in a transparent manner.

Mach (cont)

The abstract model for Unix emulation using Mach



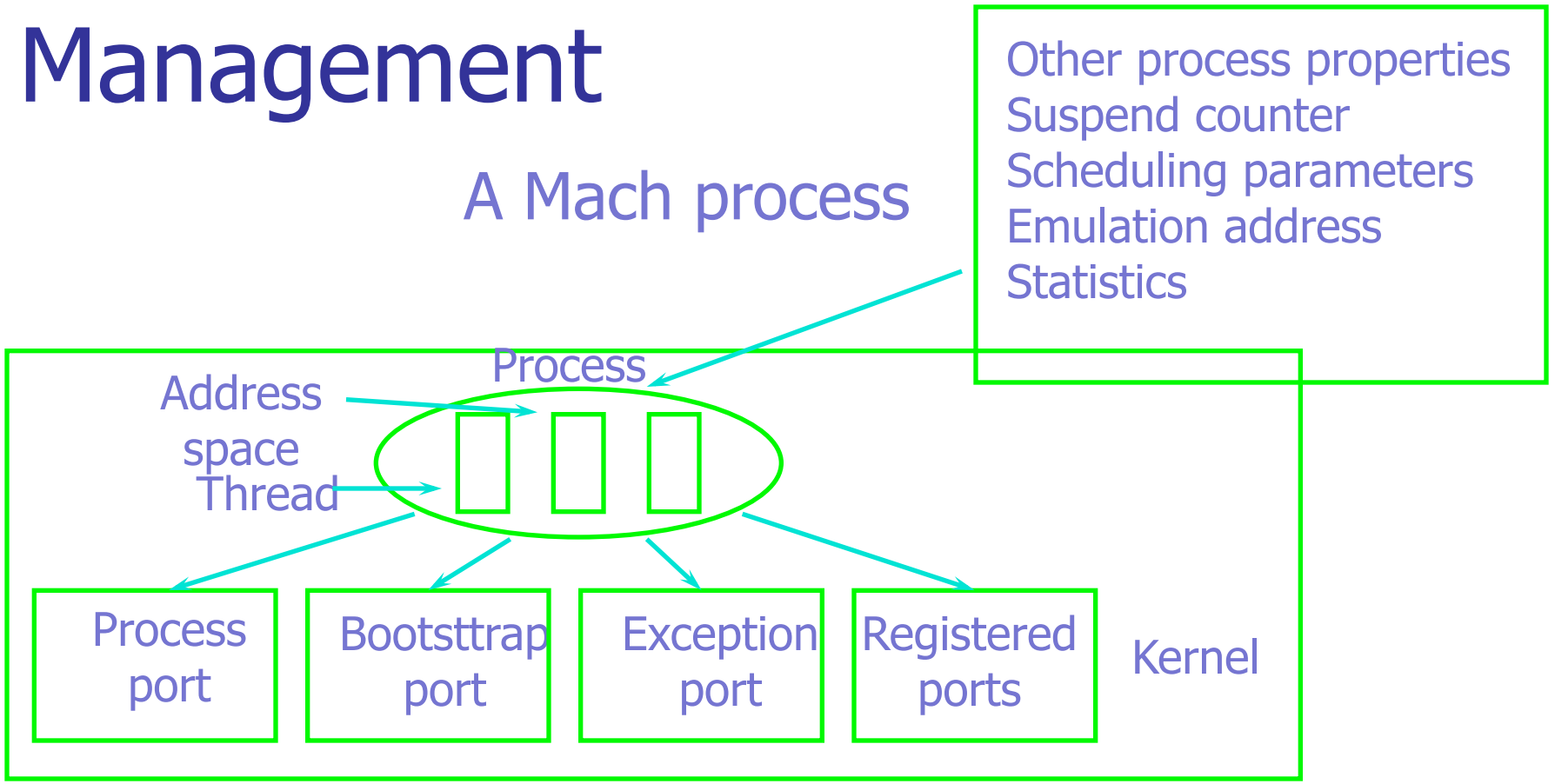
Mach - Resource Management

- **Process Management:** Mach split the traditional UNIX abstraction of a process into a process and threads.
- A process in Mach consists primarily of an address space and a collection of threads that execute in that address space
- In Mach, processes are passive and are used for collecting all the resources related to a group of cooperating threads into convenient containers.
- The active entities in Mach are the threads that execute instructions and manipulate their registers and address spaces.
- Each thread belongs to exactly one process. A process cannot do anything unless it has one or more threads.
- A thread contains the processor state, and the contents of a machine's registers.
- All threads within a process share the virtual memory address space and communications privileges associated with their process.

Mach - Process Management

- The UNIX abstraction of a process is simulated in Mach by combining a process and a single thread.
- Mach allows multiprocessor threads to execute in parallel on separate processors.
- Mach threads are managed by the kernel, that is, they are heavyweight threads rather than lightweight threads (pure user space threads).
- Thread creation and destruction are performed by the kernel and involve updating kernel data structures.
- The primary data structure used by the Mach scheduler is the run queue
- A hint is maintained to indicate the probable location of the highest priority thread. Each run queue also contains a mutual exclusion lock and a count of threads currently enqueued.

Mach : Process Management



Selected process management calls in Mach :
Create, Terminate, Suspend, Resume, Priority, Assign, Info, Threads.

Mach - Processor Allocation

- processor allocation approach must be flexible and portable
  - 1) allocating processors to applications should support different languages with different programming modes;
  - 2) allocation approach should be easily portable to different parallel and distributed architectures;
  - 3) it should accommodate different allocation policies;
  - 4) it should offer applications complete control over which threads execute on which processors, but it should not force implementation on applications not wanting this degree of control.

Mach - Processor Allocation

- Responsibility for the allocation and use of dedicated processors is divided among
  - the application, server, and kernel.
- The application controls the assignment of processes and threads to processor sets.
- The server controls the assignment of processors to processor sets.
- The kernel does whatever the application and server asks it to do.
- In this scheme, the physical processors allocated to the processor sets of an application can be chosen to match the application requirements.
- Assigning threads to processor sets gives the application complete control over which threads run on which processors.
- isolating scheduling policy in a server, simplifies changes for different hardware architectures and site-specific usage policies.

Mach : Memory Management

- It has three parts:
  - pmap module: sets the MMU registers and hardware page tables
  - machine-independent kernel code: to process page faults, managing address maps, and replacing pages
  - external manager (user space): management of backup store, which pages in main memory, where they are kept on disks
- Distributed Memory in Mach
  - single, linear virtual address space
  - shared pages are managed by one or more special memory managers
  - one Distributed Shared Memory (DSM) server
  - DSM handles all references to shared pages
  - readable pages are replicated at all nodes
  - writable pages will have only one copy

Communication in Mach

- It can handle asynchronous message passing, RPC, byte streams, and other forms
- The basis of all communications is a protected mailbox called ``port''

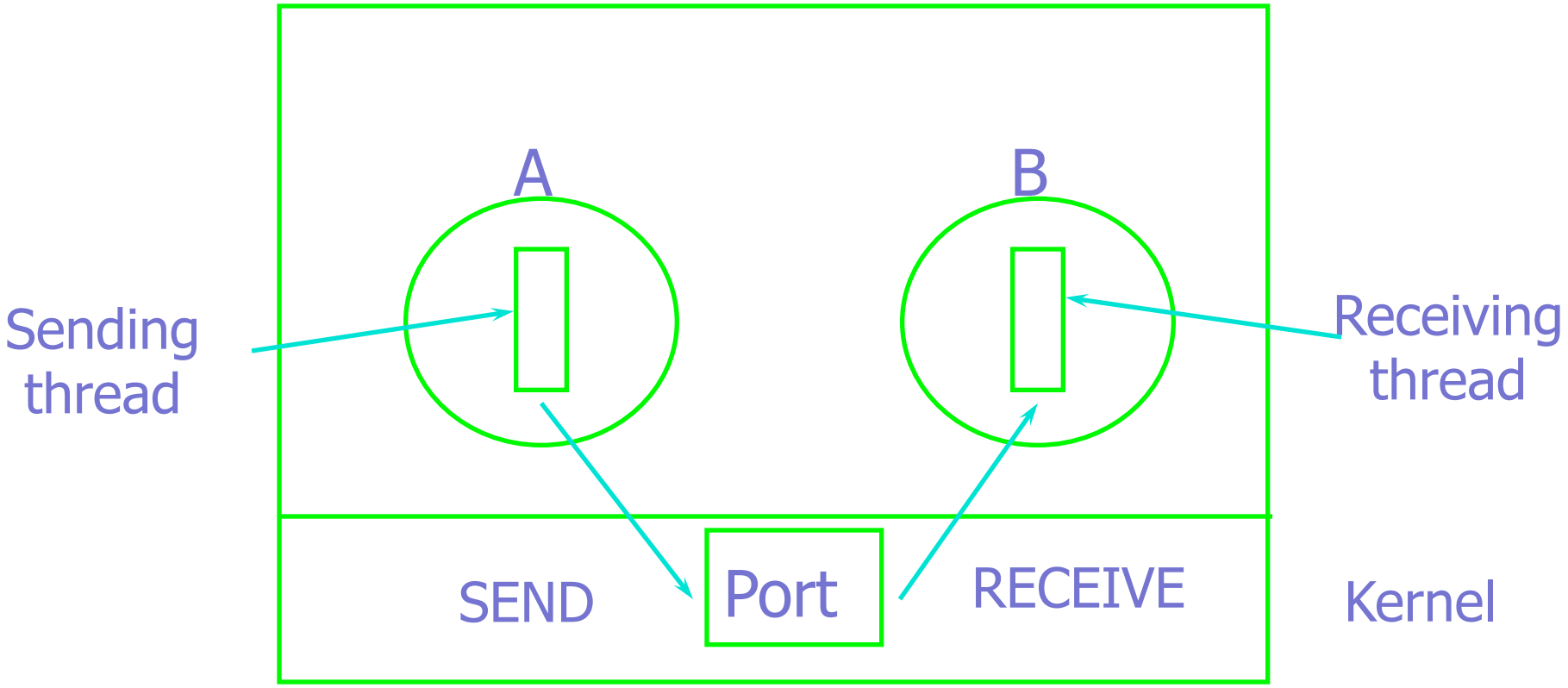
Selected port management calls in Mach :
Allocate, Destroy, Deallocate, Extract_right, Insert_right, Move_member, Set_limit

MACH : Descriptions (cont)

- Every access to a port is protected by the use of capabilities
- Network servers provide all comm. over the network. Each host will have one network server
- Each network server maintains a mapping between network ports and corresponding local ports
- MACH provides an interface specification language (MIG) to specify interfaces between clients and servers and then generate remote procedure call stubs
- It is possible to share memory between multiple threads (the entities of control) and tasks (the entities of resource allocation) in a highly machine independent manner

MACH : Descriptions

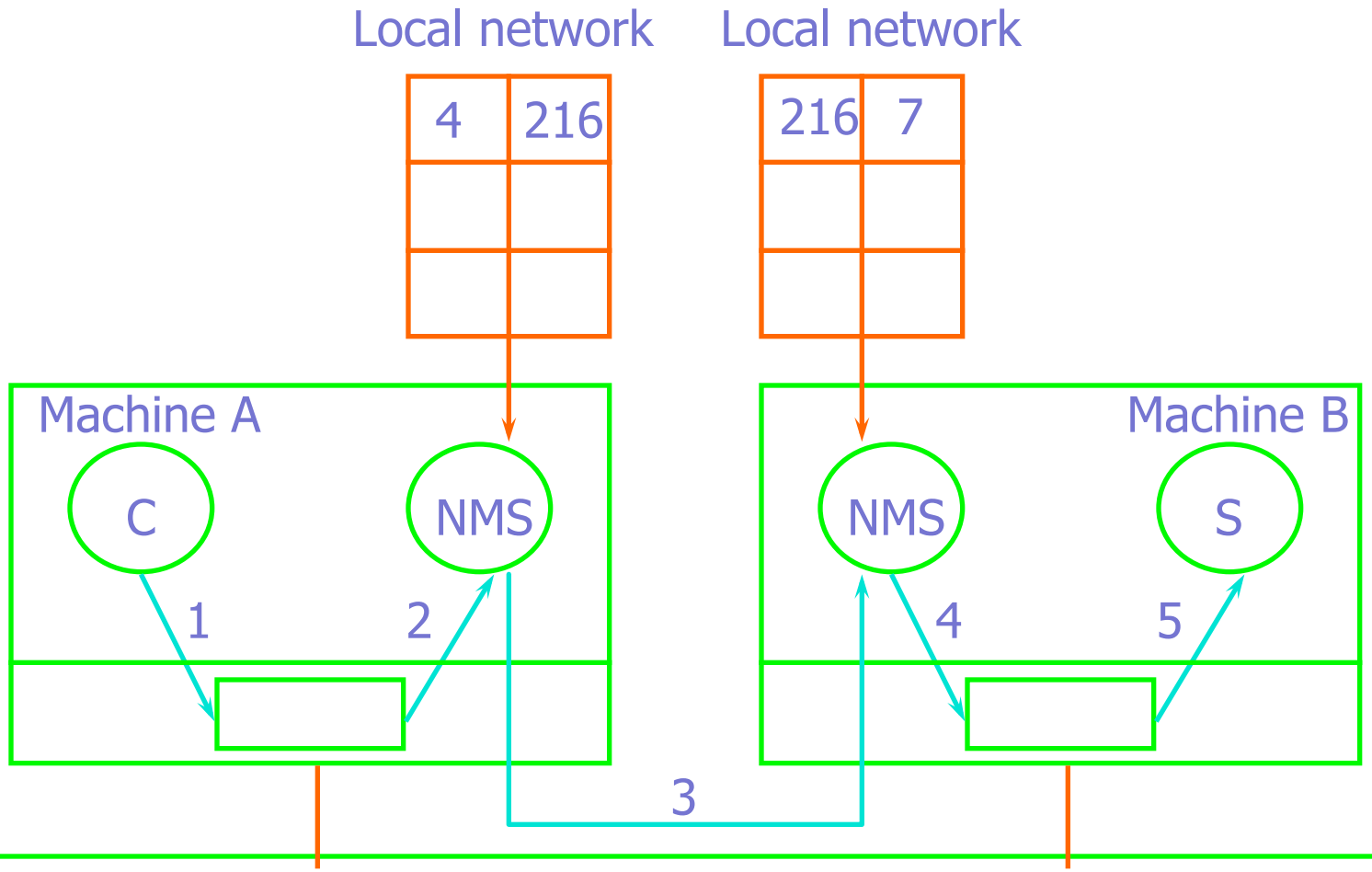
- MACH runs on many platforms
- Interprocess Communication (IPC) : It uses ports as an abstraction to which messages can be sent asynchronously



The Network Management Server

- Communication over the network is handled by net. message servers (NMS)
- NMS is a multithread process that performs a variety of functions: interfacing with local threads, forward message over network, translating data types, network-wide name lookup service and authentication services

The Network Management Server



Inter-machine communication in Mach proceeds in five steps

CASE STUDIES:The X-Kernel

Reference: *A platform for accessing Internet resources*, IEEE Computer, May 1990

- An experimental operating system to allow uniform access to resources throughout a nationwide internet
- Focus is on wide-area networks rather than LANs
  - a workstation needs to access several different file systems
  - more protocols lead to more accessible resources
- X-kernel supports a library of protocols and it accesses different resources with different protocol combinations
- On top of X-kernel, the user-level systems provides an integrated and uniform interface to resources

X-Kernel - Goals and Objectives

- X-Kernel can be viewed as a toolkit that provides all the tools and building blocks
  - to build and experiment with distributed operating systems configured to meet certain class of applications.
- The main advantages of x-Kernel are
  - x-Kernel configurability
  - its ability to provide an efficient environment to experiment with new operating systems as well as different communications protocols.

X - Kernel - System Model

- Kernel is a micro-kernel-based operating system configurable to support experimentation in inter-process communication and distributed programming
- The motivation of this approach is two fold:
  - 1) no communication paradigm is appropriate for all applications; and
  - 2) use the X-Kernel framework to obtain realistic performance measures.
- The x-Kernel supports
  - memory management
  - lightweight processes
  - development of different communications protocols.

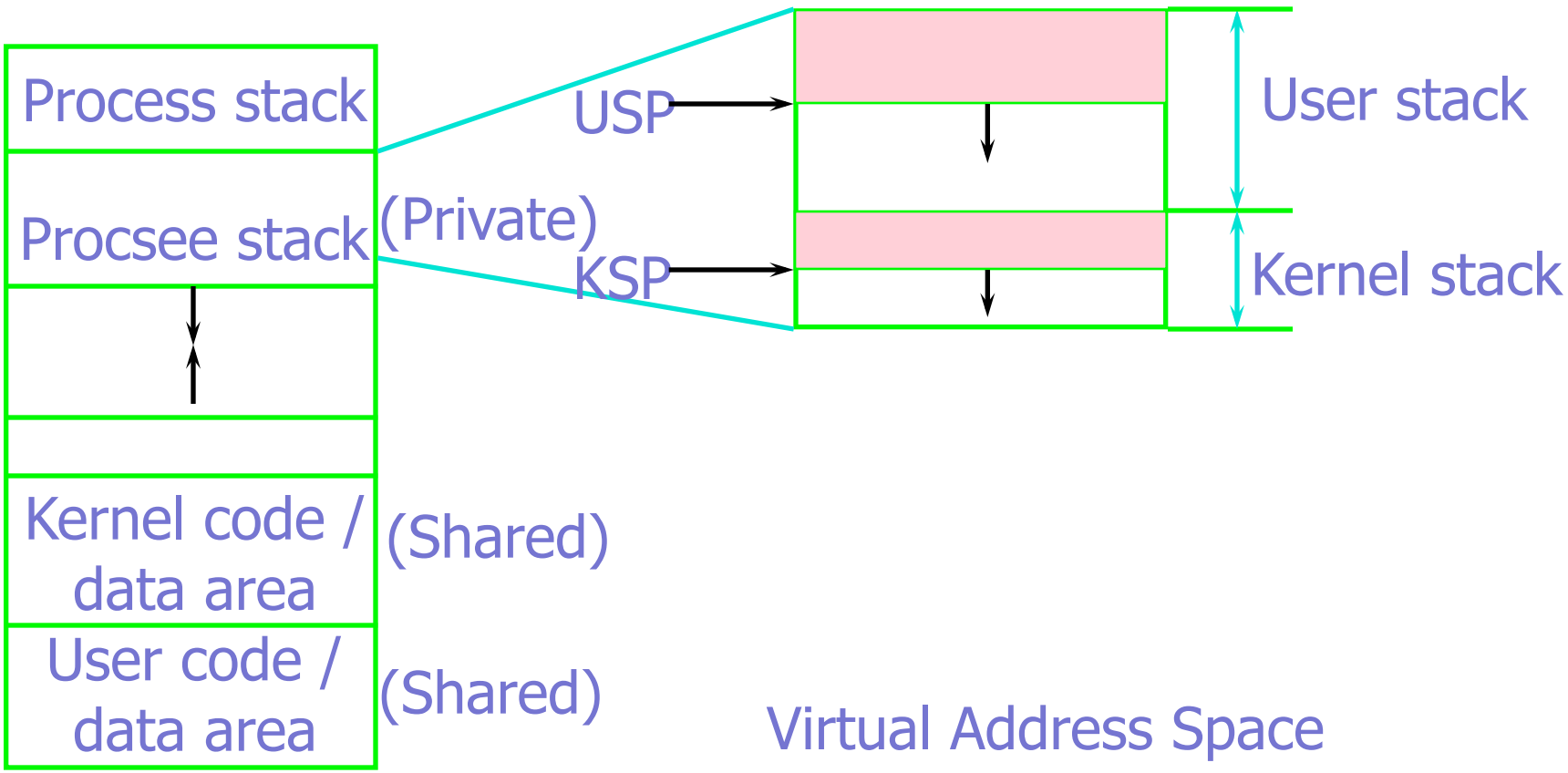
X - Kernel - Resource Management

- An important aspect of any distributed system involves its ability to pass control and data efficiently between the kernel and user programs.
- In x-Kernel, the transfer between user and kernel space has been made efficient by having the kernel execute in the same address space of the user data

The X-Kernel : Process and Memory

- an important aspect of a workstation OS involves its ability to pass control and data efficiently between the kernel and user programs
  - user data is accessible because kernel process executes in same address space
- kernel process -> user process
  - sets up user stack
  - pushes arguments
  - use user-stack
  - access only user data
- kernel -> user (245 usec), user -> kernel 20 usec on SUN 3/75

The X-Kernel : Process and Memory



X - Kernel - Inter-Process Communication

- There are three communication objects:
  - protocols, sessions, and messages.
- A different protocol object is used for each protocol type
- The session object is the protocol's objects interpreter and it contains the data that represents the protocol state.
- The message objects are transmitted by the protocol and session objects.
- To communicate between processes in different address spaces or different machines the protocol objects are used and messages are sent.
- Processes in the same address space can synchronize using kernel semaphores.

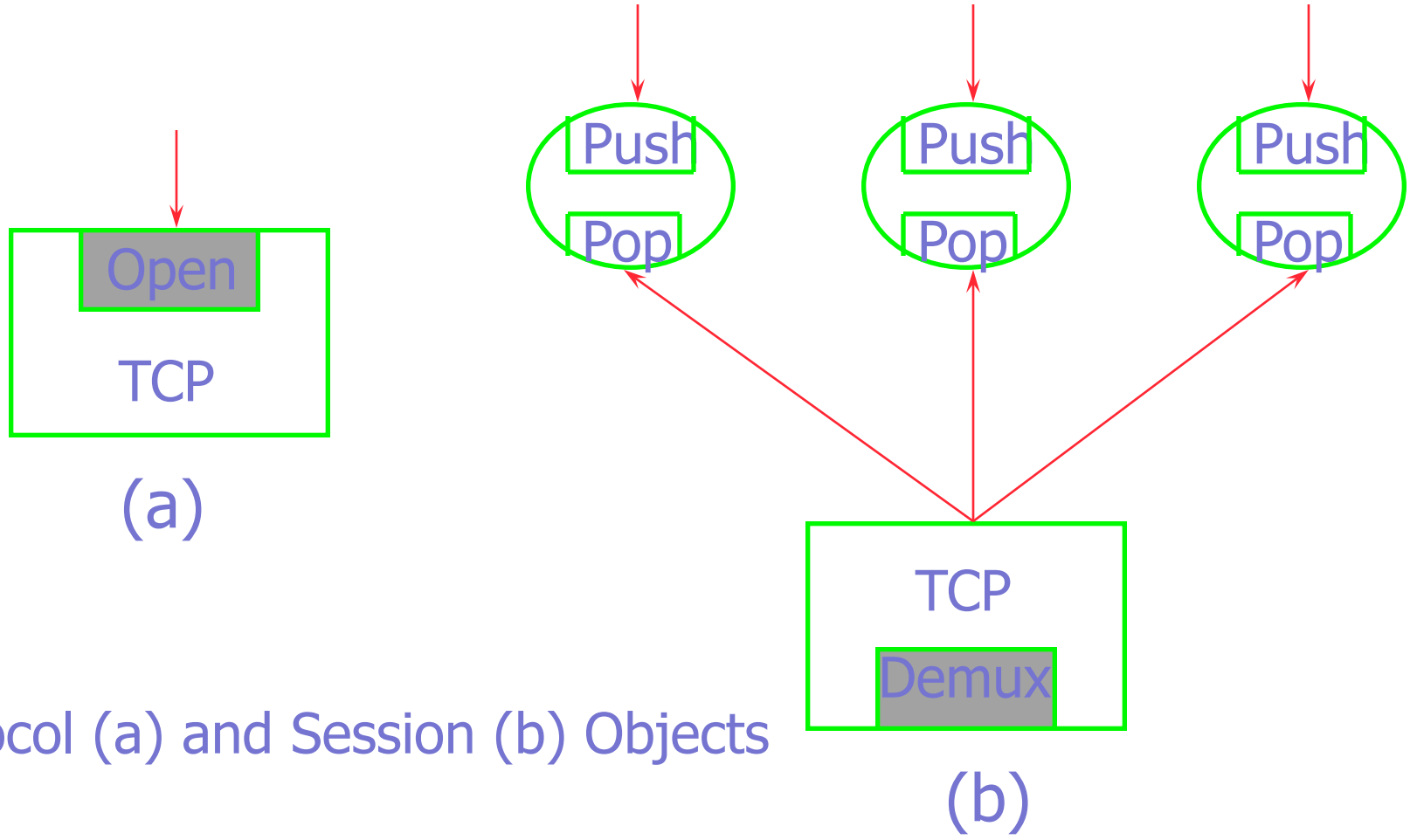
X - Kernel - Inter-Process Communiation

- There are several routines which are used to provide the wide variety of protocols:.
  - buffer manager, map manager, and event manager,
- The buffer manager routines uses the heap to allocate message buffers
- The map manager maps one identifier from a message header to capabilities used by the kernel objects.
- The event manager allows a protocol to provide a procedure call which is a timed event
- A protocol object can create a session object and demultiplex messages before sent to session object.
- There are three operations that the protocol object uses which are
  - open, open_enable, and open_done

The X-Kernel : Communication Management

- The open creates a session object caused by a user process,
- the open_enable and open_done are invoked by a message from the network.
- The protocol object is also a demux function operation and can send any message from the network to one of the session created by the protocol object
- Session have two operations push and pop.
- Push is used by a higher session to send a message to a lower session.
- The pop operation is used by the demux operation of a protocol object to send a message to its session.
- As a message is passed between sessions, information is added to the header and sometimes the message can be split up into several messages.
- If a message goes from a device to the user level, it can be put into a larger message.

The X-Kernel : Communication Management

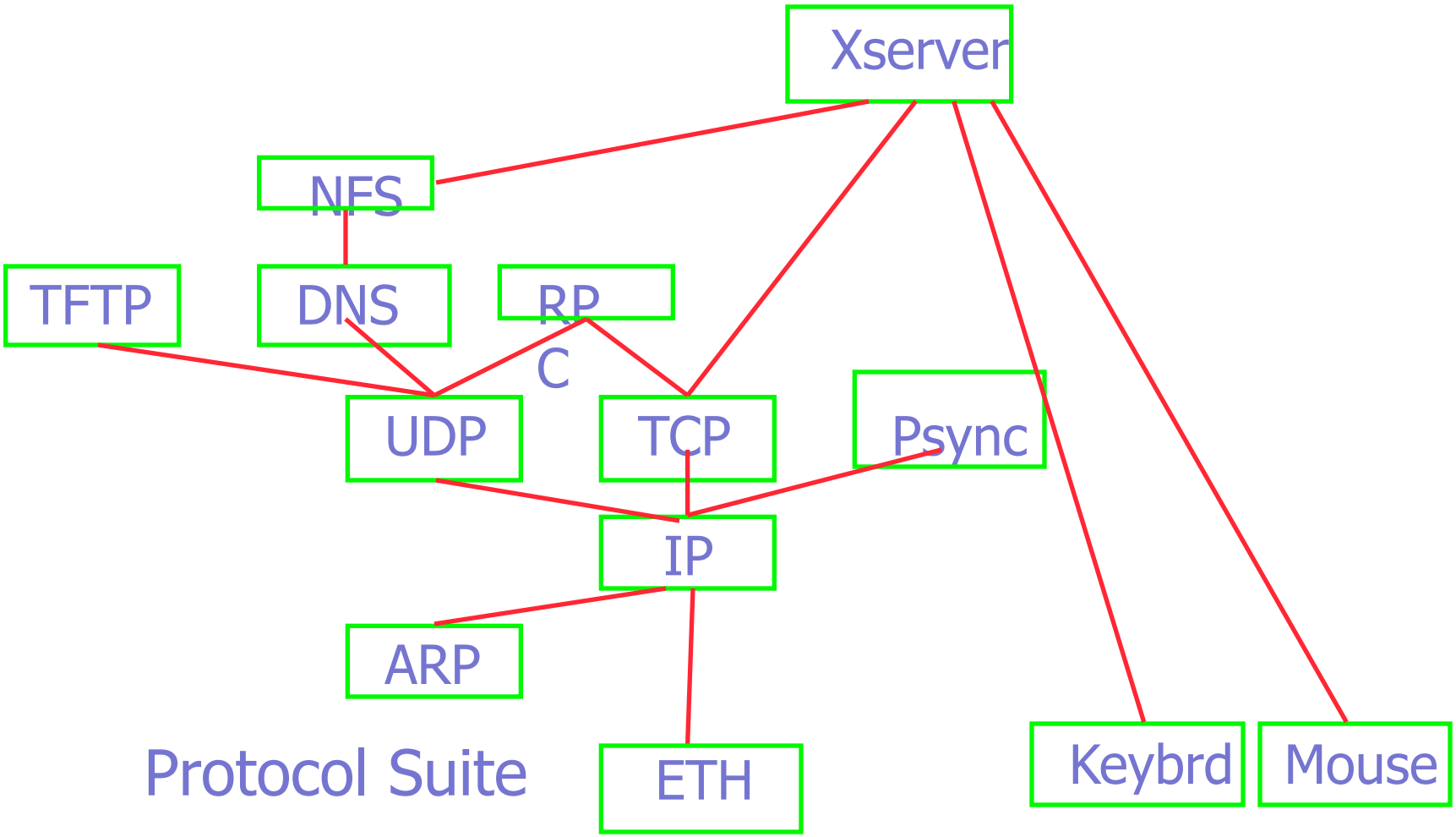


Protocol (a) and Session (b) Objects

The X-Kernel : Communication Management

- In X-kernel, each protocol is encapsulated into a single protocol object and a collection of session objects
- TCP protocol is directly accessible by user programs.
- For a user to access one protocol, it makes itself a protocol and then access that protocol
- In X -kernel, the responsibility for processing the packet passes from one protocol to another by having the first protocol invoke an operation exported by the second.
  - no need to process switch between protocols
- X-kernel has a library of support routines that provides efficient solutions to programming tasks common to all protocols (message manager and map manager and event manager).
  - Massage manger: add headers, strip headers, fragment, etc.
  - Map manager: to switch between lower level protocols to appropriate session objects
  - Event manger: provides an alarm (clock) for protocols

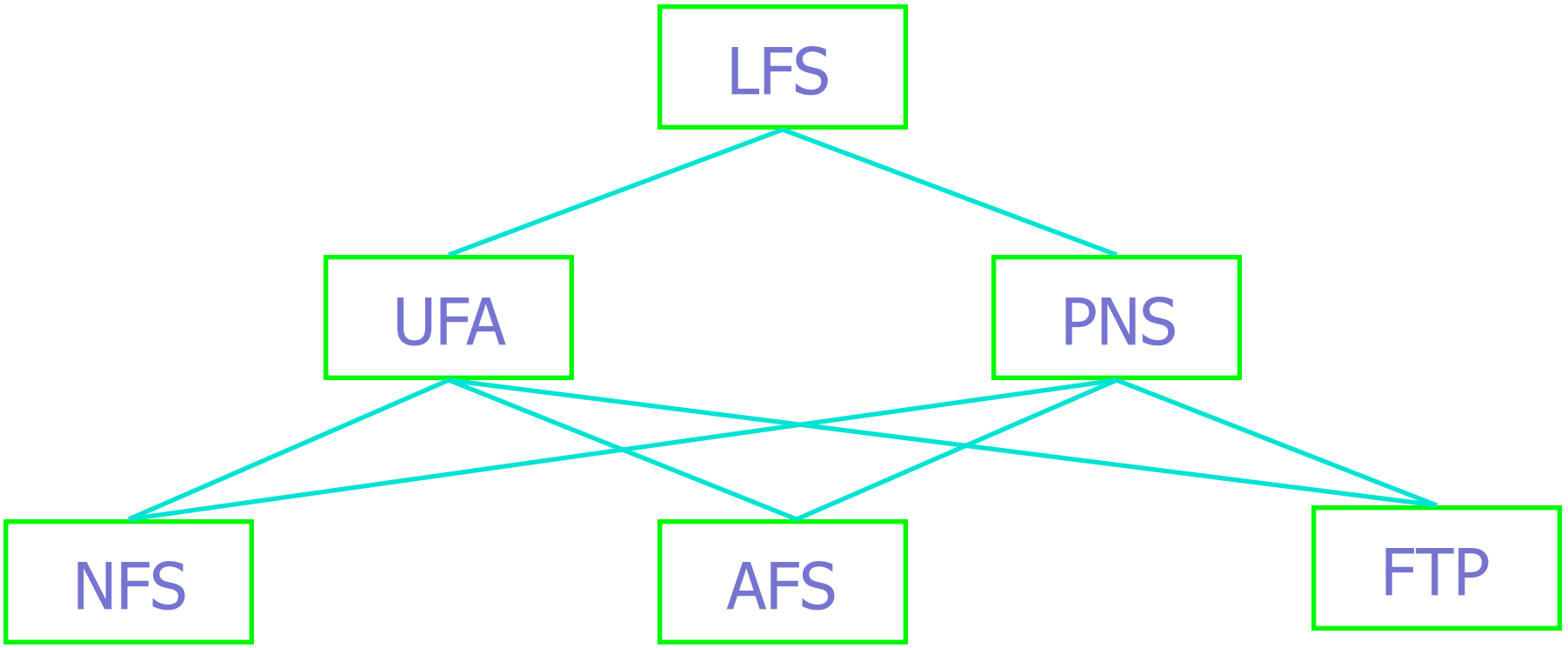
The X-Kernel : Communication Management



X - Kernel File System

- The x-Kernel file system allows its users to access any x-Kernel resource regardless of the user location.
- It provides a uniform interface by implementing a logical file system that can contain several different physical file systems.
- the logical file system concept allows the file system to be tailored to user`s requirements instead of being tied to a machine architecture.

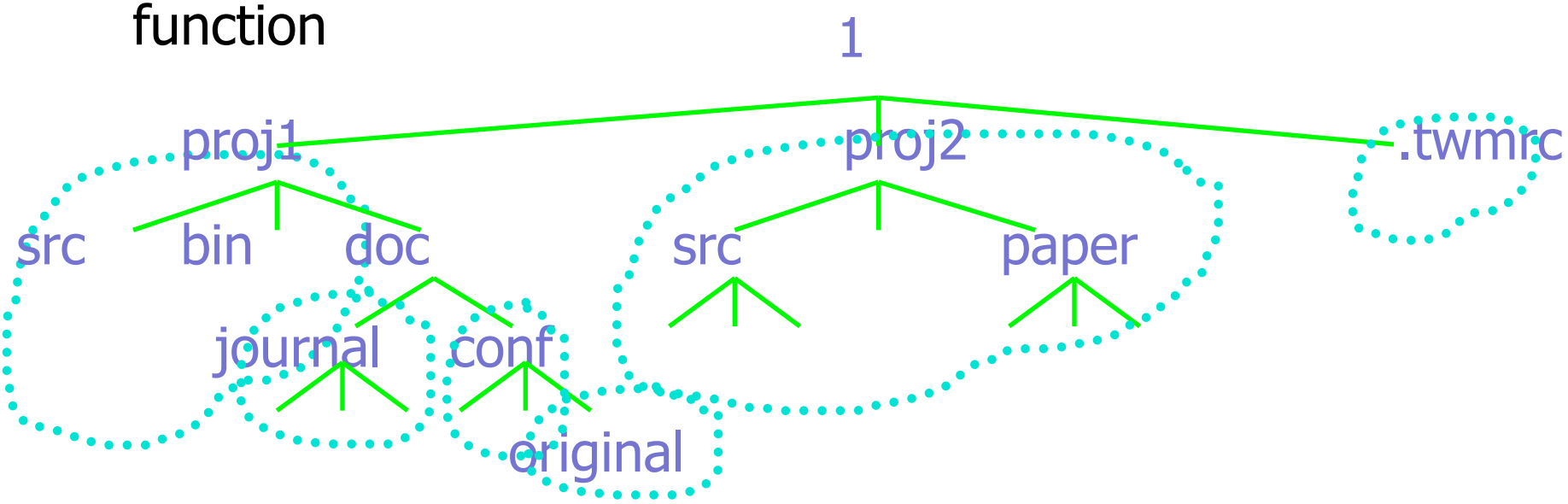
The X-Kernel : Logical File System



file protocol confguration

The X-Kernel : Logical File System

- It is defined by a per user-basis and it behaves as Unix file system
- Private Name Space (PNS) protocol implements the directory functions
- Uniform File Access (UFA) protocol implements the storage function



The X-Kernel : Performance

- X-kernel supports efficient implementation of a wide variety of protocols
- all times are user-to-user except for Sprite RPC (kernel to kernel)
- Unix protocol performance is limited by socket abstraction

Protocol	X-kernel (msec)	Other System (msec)
UDP	2.0	5.4 (Unix) 7.5 (Sprite)
TCP	3.3	6.1 (Unix) 11.0 Mach
Sun RPC	4.0	9.2 (Unix)
Spite RPC	1.7	2.6 (Sprite)

Latency of various protocol

DISTRIBUTED OPERATING SYSTEM FEATURES

Name	Transparency				
	location	access	replication	concurrency	failure
Amoeba	*	*		*	*
LOCUS	*	*	*	*	*
MACH	*	*			*
SPRITE	*	*		*	
V	*	*			

Name	Heterogeneity	
	OS	CPU
Amoeba	(UNIX)	VAXs, SUN 3, (386s, SPARC)
LOCUS	UNIX 4.x BSD, V	DEC VAX/750, PDP-11/45, IBM PC
MACH	UNIX 4.3 BSD	VAXs, SUN 3, NS32032, MacII, I386
SPRITE	UNIX 4.3 BSD	SUNs, DEC&SPARCstations, Sequent Symmetr
V	UNIX 4.x BSD	SUN 2/3, VAXstationII

DISTRIBUTED OPERATING SYSTEMS FEATURES

Name	Changes made		Communication				Connection		
	new kernel	new kernel	standard protocol	specialized protocol	shared memory	RPC based	VC	datagram	pipes/ stream
Amoeba	*			Amoeba		*			
LOCUS	*			*			*		
MACH	*			*	*	*		*	
SPRITE	*			*		*			
V	*								
VMTP			*	*					

Name	Semantics			Naming			Security		
	may be	at most once	exactly once	object - oriented	hier- archical	en- cryption	special HW	capa- bilities	mutual auth.
Amoeba		*		*		*		*	*
LOCUS		*	*		*				
MACH		*			*				*
SPRITE		*			*				
V		*			*				

DISTRIBUTED OPERATING SYSTEMS FEATURES

Name	Availability				Failures				object/	
	synchro-nization	TA	nested TA	repli-cation	recovery client crash	recovery server crash	stable storage	orphan detection		Process Mobility
Amoeba	*					*	*	*		
LOCUS	*	*	*	*	*	*				*
MACH					+	+				*
SPRITE	*			+		+				*
V	*	(*)		(+)						*

Advances in D.O.S- Case Studies

- SOMBRERO
- 2K
- Network Hardware IS the O.S.
- Virtually Owned Computers-an interesting sidenote

SOMBRERO

- Arizona State University, 2006
- Very Large Single Address Space D.O.S.
- Systemwide set of virtual addresses
- Unique address for EVERY datum in memory
  - 64 bit Virtual Address Space
  - Could create a 4GB object once a second for 136 years!

SOMBRERO

- Single Address Space reduces (eliminates) overhead
  - Interprocess Communications
  - File Systems
  - Multiple Virtual Address Spaces
- All activities are distributed and shared by DEFAULT

SOMBRERO (cont'd)

- Virtual Addresses
  - Permanently and uniquely bound to all objects
  - Spans all levels of storage
  - Manipulated directly by the CPU- no address translation
  - All physical storage devices are viewed as caches for the contents of virtual objects

SOMBRERO (cont'd)

- Transparent Interprocess Communication
- Unified Resource Management
  - threads may travel with no changes to address space
- Security by restricting OBJECT access

2K: A Component-Based Network-Centric Operating System For the next Millennium

- University of Illinois: Urbana-Champaign
- Integrated Environment to support:
  - Dynamic instantiation
  - Heterogeneous computing systems
  - Distributed resource management

2K (cont'd)

- Operates as Configurable Middleware
  - does not rely on TCP/IP
  - dynamically configurable reflective ORB
  - adaptable Microkernel
- "What you need is what you get" (WYNIWYG)
  - only those objects needed by application are loaded
- Object based resource management

2K (cont'd)

- All elements represented as CORBA objects
- Each object has a network-wide identity
- Upon dynamic configuration, objects that constitute a service are assembled
- QOS: after negotiating a connection, applications have access to system's dynamic state

2K (cont'd)

- Each Node executes a Local Resource Manager (LRM)
- Uses symbolic naming and CORBA compliant naming
- Access to objects is restricted to controlled CORBA interfaces

Network Hardware IS the O.S.

- University of Madrid, 1997
- Adaptable and Flexible D.O.S.
- Minimal Adaptable Distributed Microkernel
- "...build distributed-microkernel based Operating Systems instead of microkernel based distributed systems."

Network Hardware IS the O.S.

- Entire network is considered exported and multiplexed hardware instead of isolated entities
- OFF: microkernel and its abstractions are distributed and adaptable
  - "Normal" microkernels multiplex local resources only
  - OFF microkernels multiplexe local and REMOTE resources

Network Hardware IS the O.S.

- Only abstraction is the “shuttle”- program counter and stack pointer
- shuttle can be migrated between processors
- Communication handled by “portals”- a distributed interrupt line that behaves like active messaging

Network Hardware IS the O.S.

- Portals do not use buffering
- User determines Synchronous, Asynchronous, or RPC communication
- Not a Single Address Space BUT physical addresses MAY refer to remote memory locations
- Uses a distributed software TLB

Virtually Owned Computers

- University of Texas at Austin
- Each user owns an imaginary computer- a virtual computer
- The virtual computer is only a "description" and may not correspond to any real hardware components
- The virtual computer consists of a CPU and a scheduling algorithm

Virtually Owned Computers

- Each user is promised a given quality of service
- Services received are independent of the execution location
- Quality of Service is measured using response time