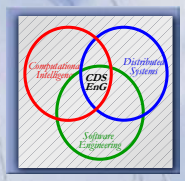
ES595a - Advanced Topics in Software & Systems Design

Cooperative Distributed Systems Engineering: Technologies &Applications



W02: Basic Concepts

Western

Outlines

- Distributed Computing Systems
  - Design Concepts
  - References

Distributed Systems: Principles and Paradigms,

Andrew S. Tanenbaum and Maarten van Steen, Prentice-Hall.

Principles of Computation Machines: Virtual & Physical

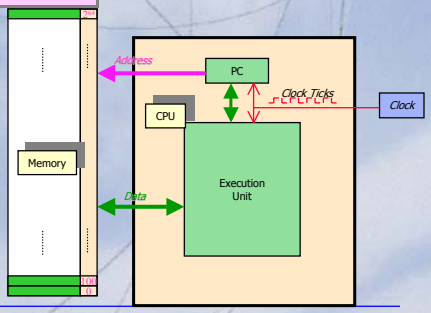
Programming language

1. Syntax
2. Semantics
3. Data Types
4. Variables
   - Binding
   - Type Checking
   - Referencing Environment (Scoping)
5. Subprogramming

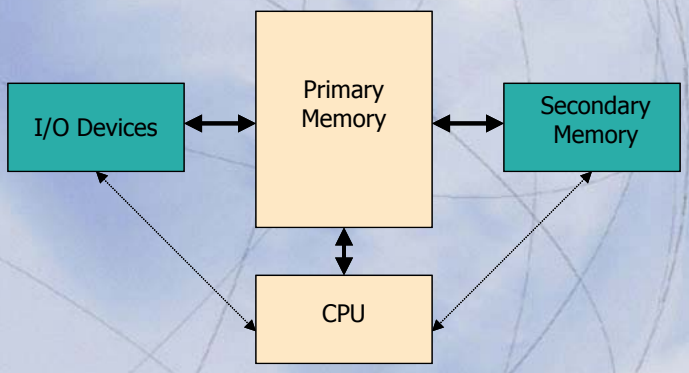
A Modeling Tool

Abstraction & Control Tool

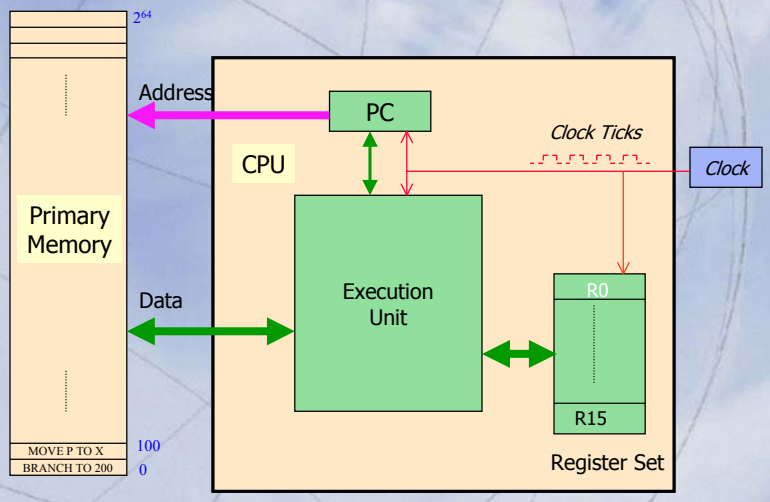
von Neumann Machine



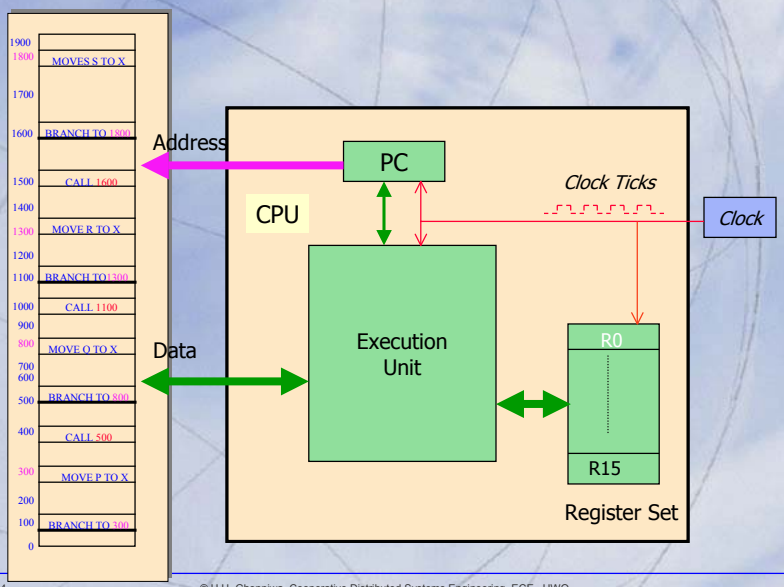
The Computation Machine: von Neumann Model



Inside the CPU: Generic



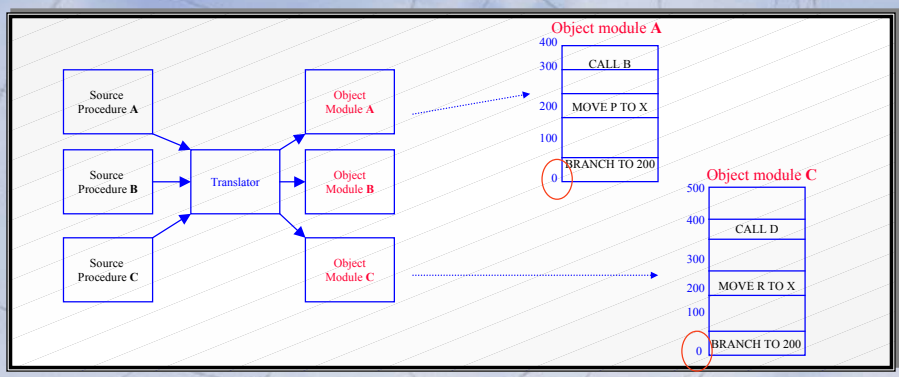
Example



Modularization :

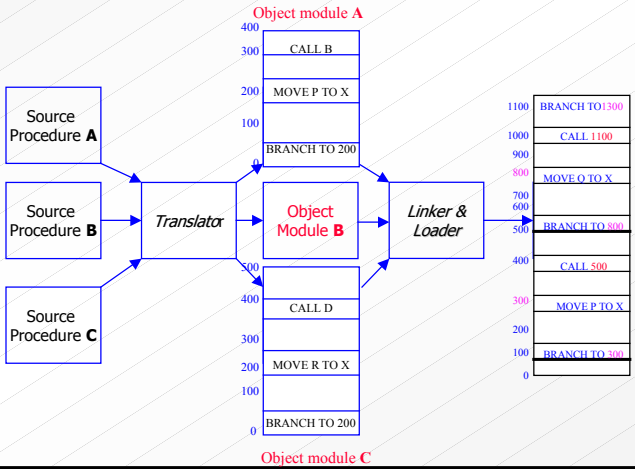
- At the start of pass one of the process,

  the instruction location counter is set to 0.
  - This assumes that the object module will be located at (virtual) address 0 during execution



Linker & Loader

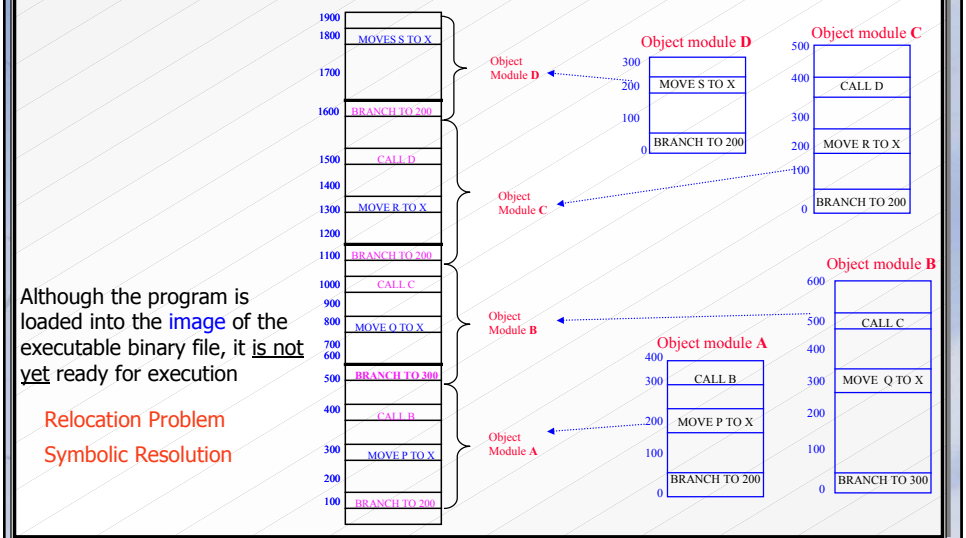
- The linker's function
  - To collects modules translated separately and glue them together to run as a unit called an executable binary program



Linking & Loading: Tasks

- To run the program,
  - The loader brings the object modules into main memory to form the image of the executable binary program
    - If there is not enough memory to form the image, a disk file can be used
  - The objective is to
    - make an exact image of the executable program's virtual address space inside the linker
    - position all the object modules at their correct locations

Example



Although the program is loaded into the image of the executable binary file, it is not yet ready for execution

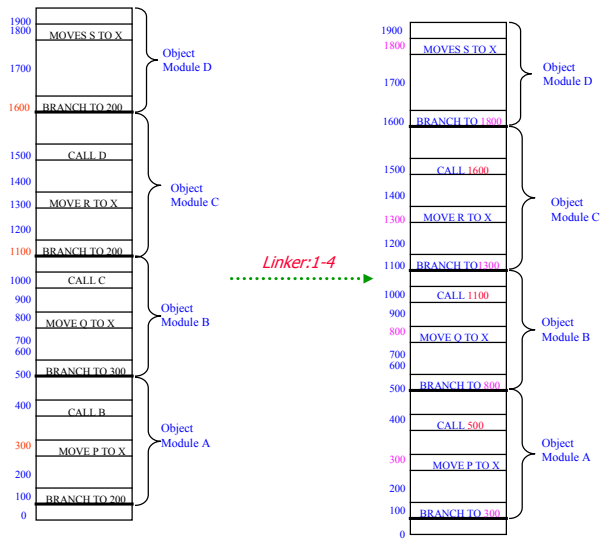
Relocation Problem

Symbolic Resolution

Linking: Steps *(rough)*

- Linkers can be used to solve both
  - relocation and symbolic resolution problems
- The linker merges the separate address spaces of the object modules into a single linear address space in the following steps:
  1. constructs a table of all the object modules and their lengths
  2. Based on this table, it assigns a starting address to each object module
  3. finds all the instructions that reference memory (relocation problem)
     a. adds to each a relocation constant equal to the starting address of its module
  4. finds all the instructions that reference other modules (resolution problem)
     a. inserts the address of these procedures in place.

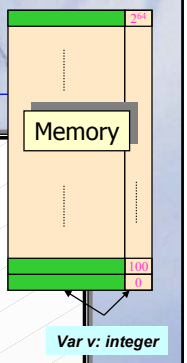
Module	Length	Starting Address
A	400	100
B	600	500
C	500	1100
D	300	1600



Address space of the program after the linker has performed 1- 4 steps.

Address, Value and Type

- Address
  - the address of the memory at which the variable is associated
- Value
  - the contents of the location with which the variable is associated
- Type
  - determines the
    - range of values of variables and
    - the set of operations that are defined for values of that type;
    - in the case of floating point, type also determines the precision



Storage Bindings

- Two issues
  - Allocation
    - getting a cell from some pool of available cells
  - De-allocation
    - putting a cell back into the pool
- The lifetime of a variable
  - Categories
    - Static: Objects with absolute address throughout the program's execution
    - Stack: Subprograms calls and returns
    - Heap: Allocation and de allocation at arbitrary times
      - Explicit heap-dynamic
      - Implicit heap-dynamic

Semantic Integration:
Type Declaration & Checking

- Declaration
  - Explicit declaration

    a program statement used for declaring the types of variables

  - Implicit declaration

    a default mechanism for specifying types of variables
    - the first appearance of the variable in the program
- Checking
  - Static checking is done at compile time.
  - Dynamic checking is done at runtime

Referencing Environment (Scoping)

- The referencing environment of a statement is

  the collection of all names that are visible in the statement

  - In a static scoped language
    - local variables and all visible variables in all of the enclosing scopes
  - In a dynamic scoped language
    - local variables and all visible variables in all active subprograms
      - A subprogram is active if its execution has begun but has not yet terminated
- Scope and lifetime
  - sometimes closely related, but are different concepts!

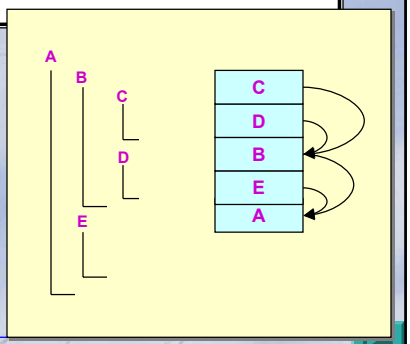
Dynamic Scope

Based on calling sequences of program units, not their textual layout
temporal versus spatial

References to variables are connected to declaration by searching back through the chain of subprogram calls that forced execution to this point

Advantage: convenience

Disadvantage: poor readability



Static vs. Dynamic Scope

Example:

```
MAIN
 - declaration of x
 SUB1
  - declaration of x
  ...
  call SUB2
  ...
 SUB2
  ...
  - reference to x
  ...
  ...
  call SUB1
  ...
```

MAIN calls SUB1

SUB1 calls SUB2

SUB2 uses x

Static scoping –

reference to x is to MAIN's x

Dynamic scoping –

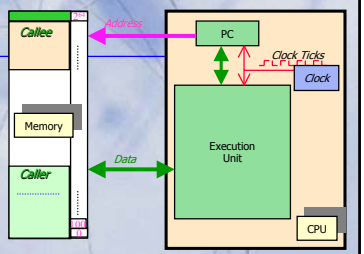
reference to x is to SUB1's x

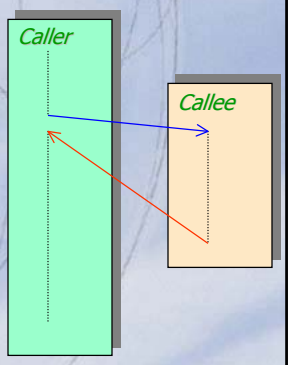
Fundamentals of Subprograms

- Transfer of control mechanism
  - caller is suspended
    - the **control is transferred** to the subprogram
  - The subprogram terminates
    - the **control returns** to the caller
- Parameter Passing

Transfer of Control Mechanism

- Semantic of the Call
  - Save the execution configuration of the current program unit
  - Carry out the parameter passing process
  - Pass the needed information (*return address*, ..) to the callee
  - Transfer control to the callee
- Semantic of the Return
  - Configuration of the caller is re-established
  - Parameters are moved, copied, etc.
    - according to the method of passing
    - If stack dynamic some cleaning is needed
  - Control is transferred back to the caller

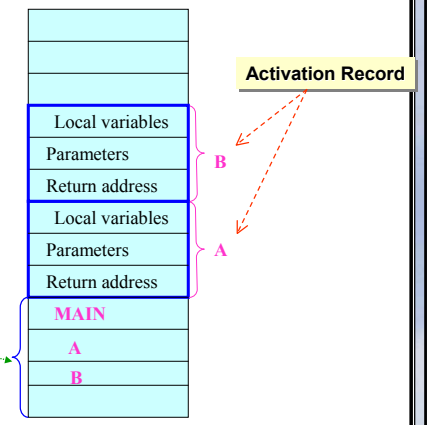




Conventional Procedure Call:
Memory Allocation

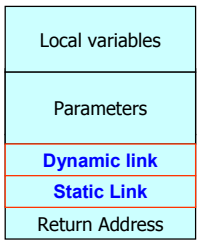
Static allocated

Assume MAIN calls A and B



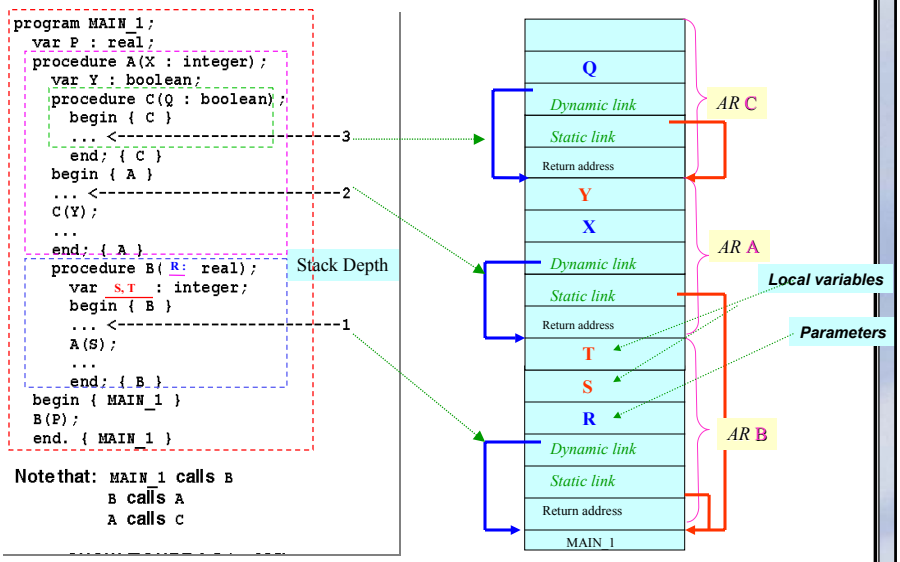
Conventional Procedure Call :
Memory Allocation

- Stack-dynamic allocated
  - The activation record format is static
    - but its size may be dynamic
  - The **static link** points to the bottom of the activation record instance of an activation **of the static parent** (used for access to non-local variables)
  - The **dynamic link** points to the instance of the activation record of **the caller**
  - An activation record instance (AR) is **dynamically created** when a subprogram is called



activation record

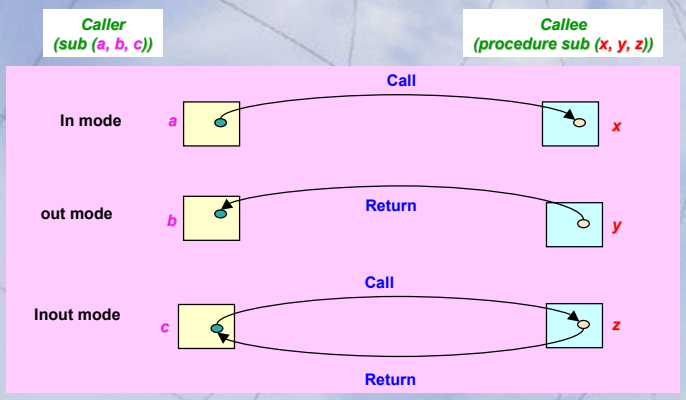
Static link - where is declared
Dynamic link - Top of AR of caller



Parameter Passing

- *Callee*: int Ag (int i, real a) …
  - *i, a:* ***formal parameters***
- *Caller*: Ag (1, x)
  - *1, x:* ***actual parameters***
    - During the execution
      - each formal parameter is "somehow" connected to the corresponding actual parameter
        - **parameter passing** method (or mode) determines the nature of this connection

Modes of passing parameters



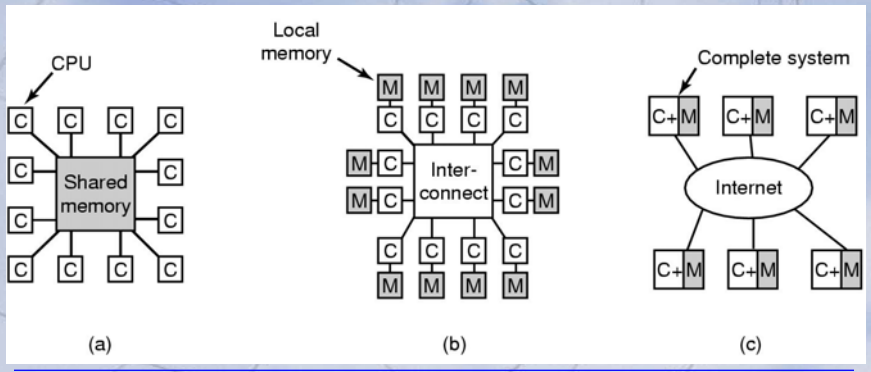
Parameter Passing Methods

- Pass by value
- Pass by reference
- Pass by result
- Pass by value result
- Pass by name

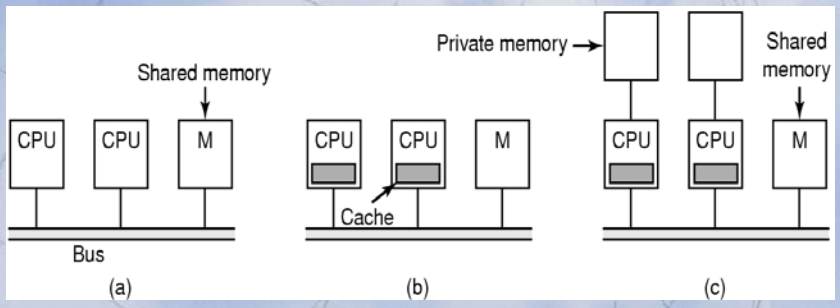
Multiprocessor Systems

Definition:
A computer system in which two or more CPUs share full access to a common RAM

- Continuous need for faster computers
  - shared memory model
  - message passing multiprocessor
  - wide area distributed system

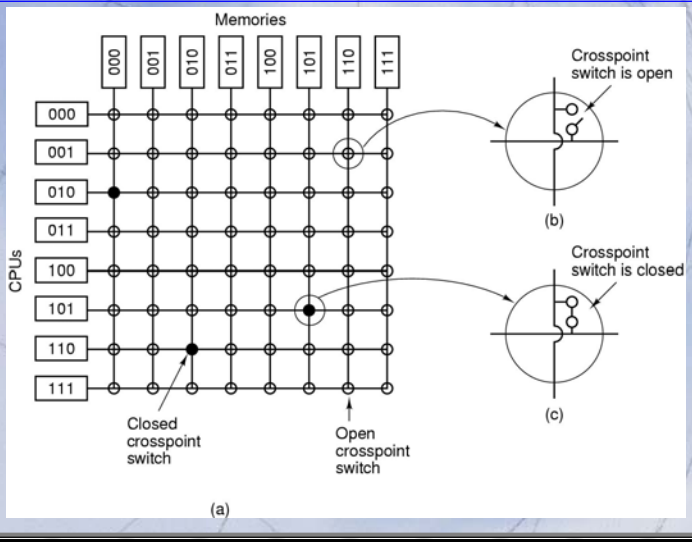


Multiprocessor



Bus-based multiprocessors

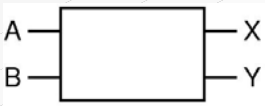
Multiprocessor



- Multiprocessor using a crossbar switch

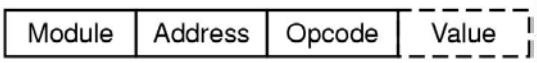
Multiprocessor

- multiprocessors using multistage switching networks can be built from 2x2 switches



(a)

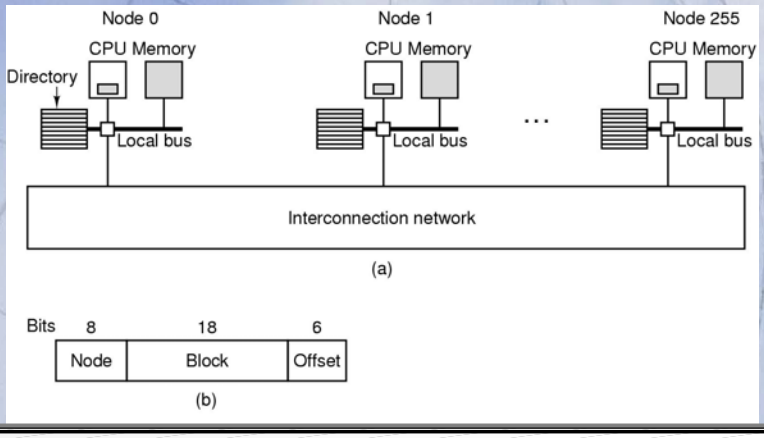
2x2 switch



(b)

Message format

Multiprocessor



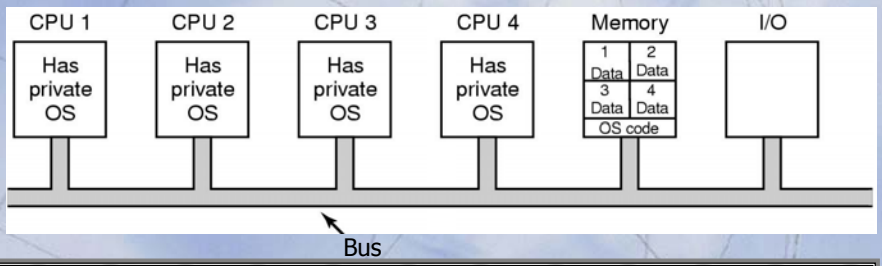
(a) 256-node directory based multiprocessor
(b) Fields of 32-bit memory address

Multiprocessor

Multiprocessor Characteristics

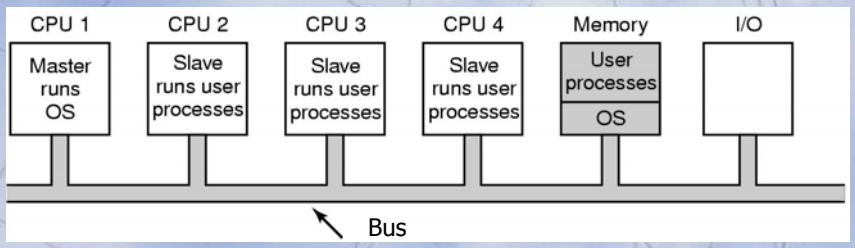
1. Single address space visible to all CPUs
2. Access to remote memory via commands
   - LOAD
   - STORE
3. Access to remote memory slower than to local

Multiprocessor OS Types



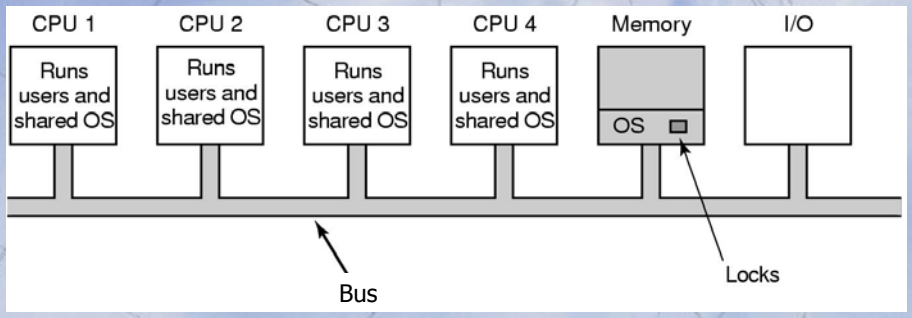
Each CPU has its own operating system

Multiprocessor OS *(Cont.)*



Master-Slave multiprocessors

Multiprocessor OS (Cont.)



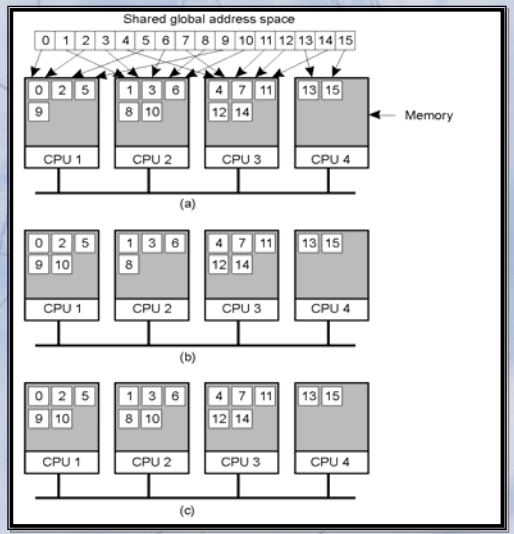
- Symmetric Multiprocessors
  - SMP multiprocessor model

Distributed Shared Memory Systems

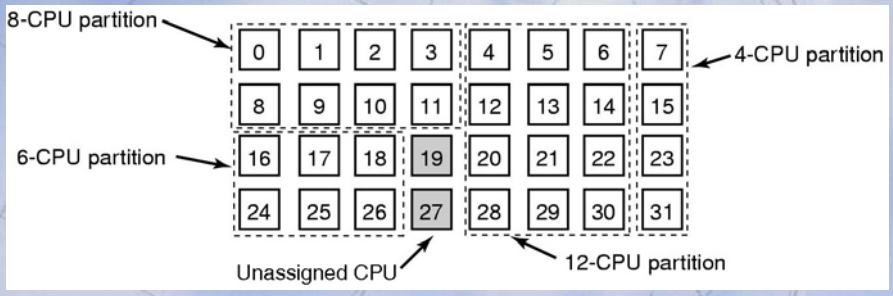
a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10

c) Situation if page 10 is read only and replication is used

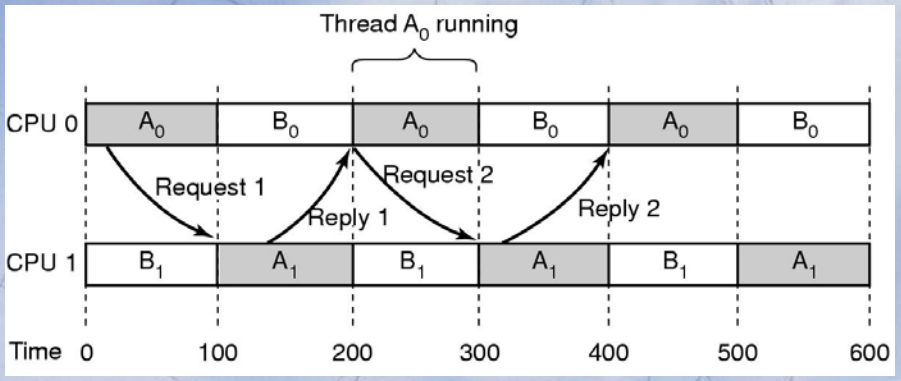


Multiprocessor Scheduling



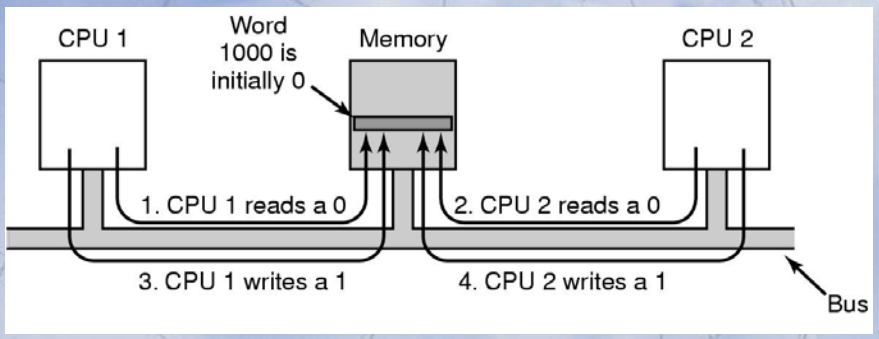
- Space sharing
  - multiple threads at same time across multiple CPUs

Multiprocessor Scheduling



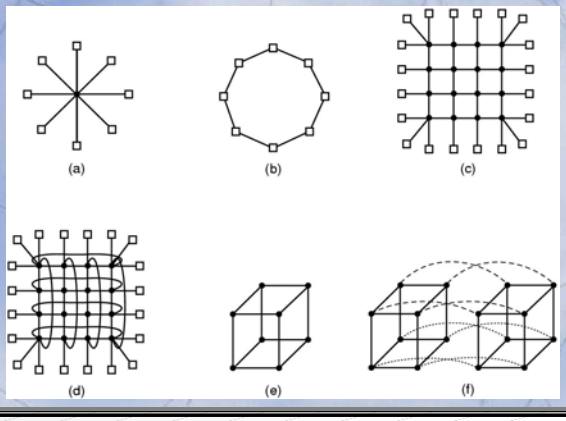
- Problem with communication between two threads
  - both belong to process A
  - both running out of phase

Multiprocessor Synchronization



TSL instruction can fail if bus already locked

Multicomputer

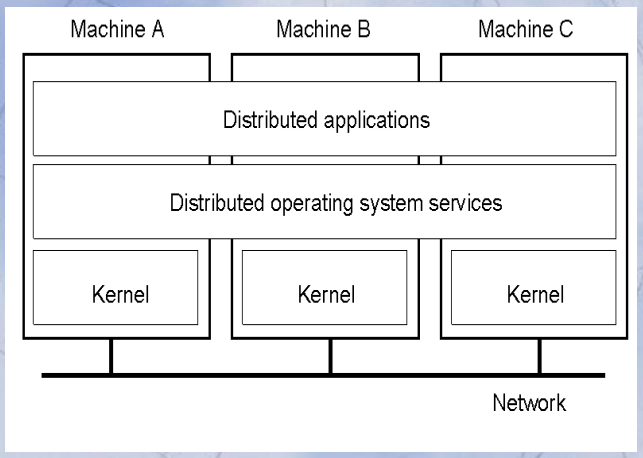


- Interconnection topologies
  - (a) single switch
  - (b) ring
  - (c) grid
  - (d) double torus
  - (e) cube
  - (f) hypercube

Software Concepts

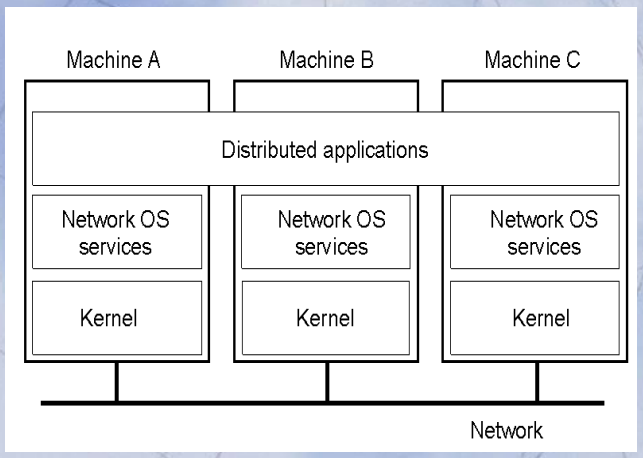
System	Description	Main Goal
DOS (Distributed Operating Systems)	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS (Network Operating Systems)	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer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

Multicomputer Operating Systems



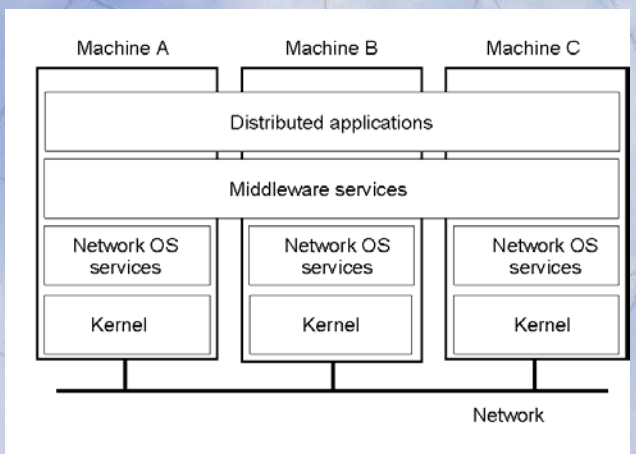
- General structure of a multicomputer operating system

Network Operating System



- General structure of a network operating system.

Middleware-based Distributed Systems



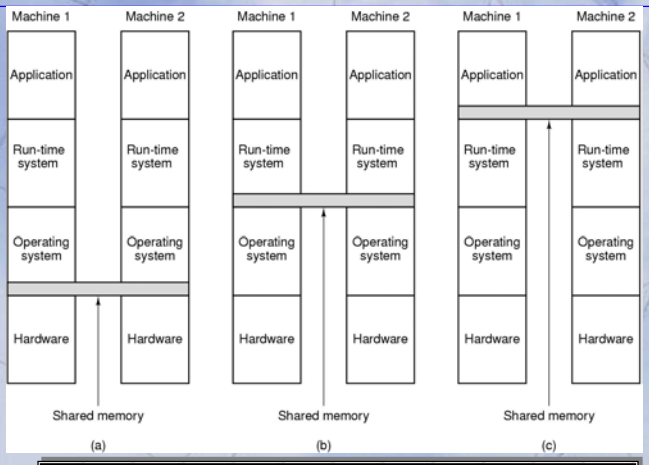
- General structure of a distributed system as middleware.

Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several users
Concurrency	Hide that a resource may be shared by several users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

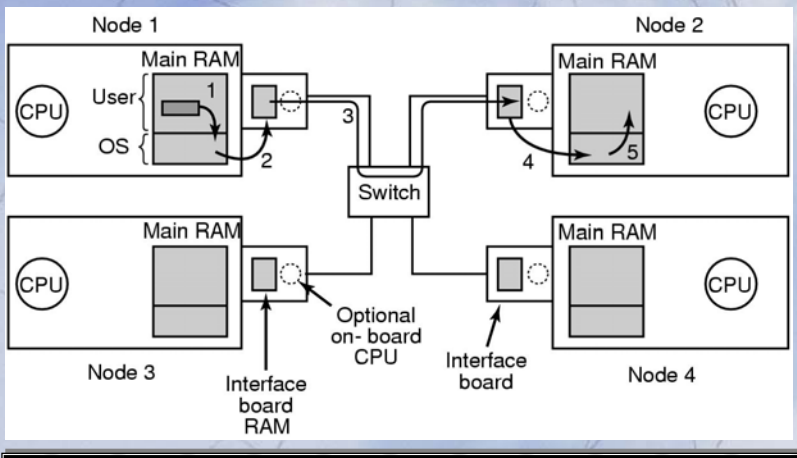
Different forms of transparency in a distributed system.

Distributed Shared Memory



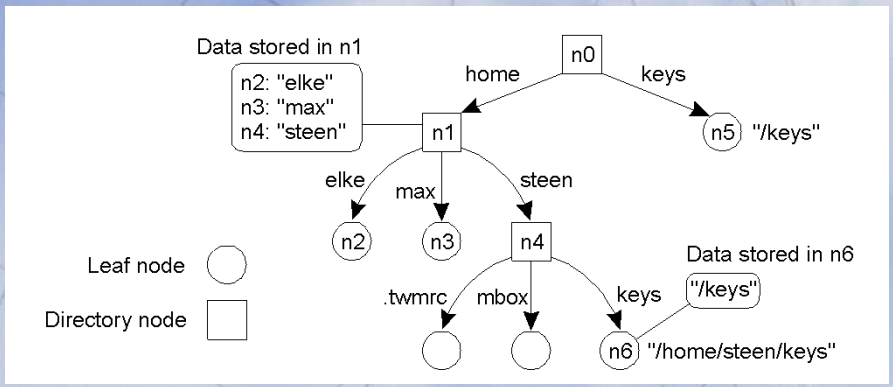
- Note layers where it can be implemented
  - hardware
  - operating system
  - user-level software

Multicomputer



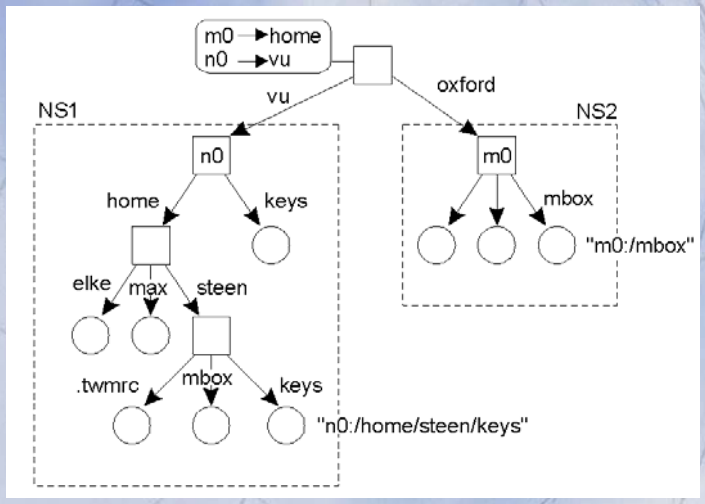
Network interface boards in a multicomputer

Name Spaces
Linking and Mounting



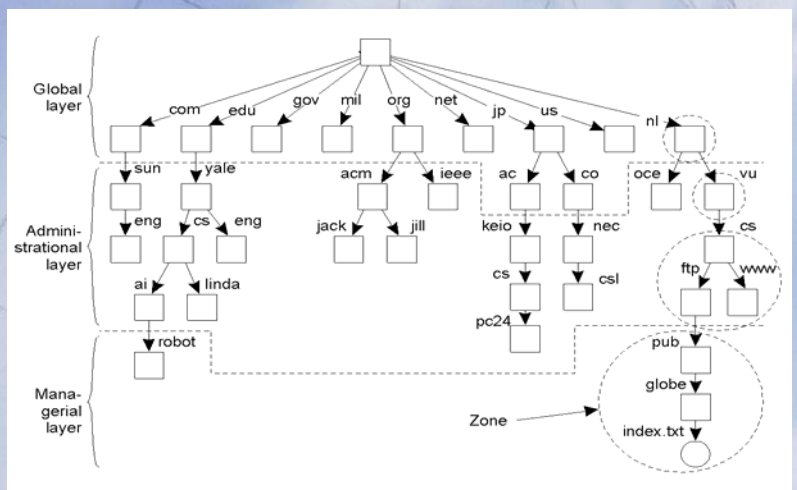
- A general naming graph with a single root node.
- The concept of a symbolic link explained in a naming graph.

Linking and Mounting



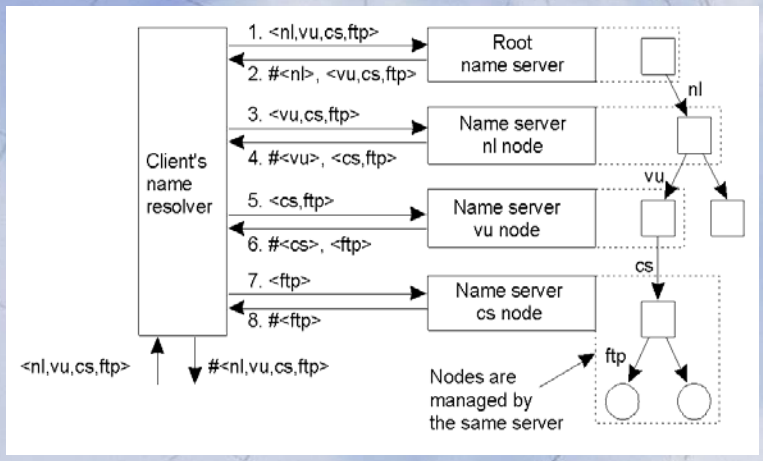
- Organization of the DEC Global Name Service

Name Space Distribution



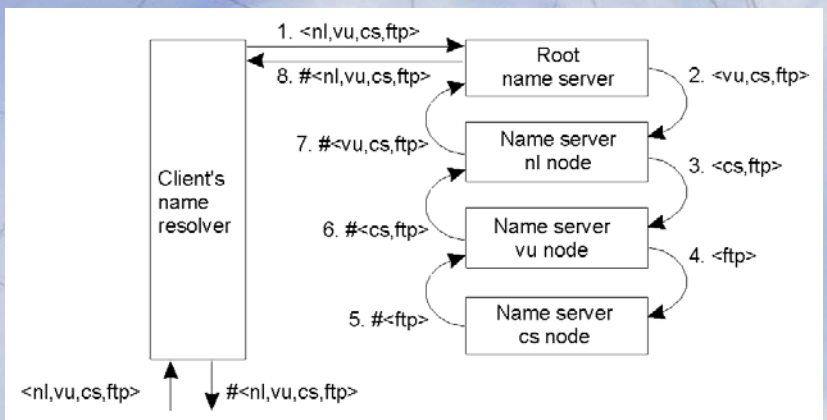
- An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Implementation of Name Resolution



- The principle of iterative name resolution.

Implementation of Name Resolution



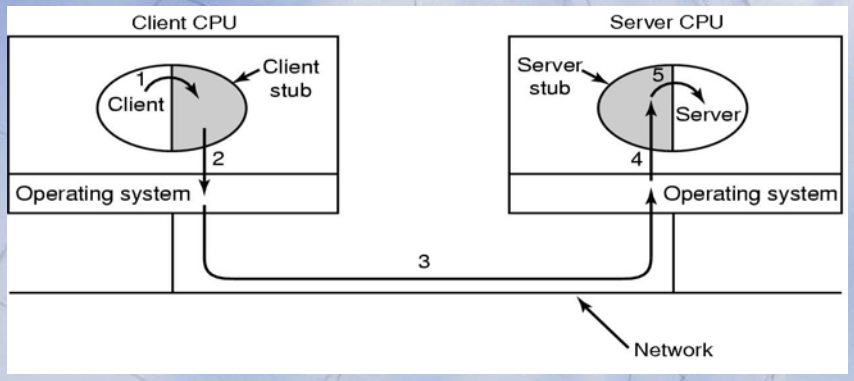
- The principle of recursive name resolution.

Implementation of Name Resolution

Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
eng	<ftp>	#<ftp>	--	--	#<ftp>
uwo	<eng,ftp>	#<eng>	<ftp>	#<ftp>	#<eng> #<eng, ftp>
ca	<uwo,eng,ftp>	#<uwo>	<eng,ftp>	#<eng> #<eng,ftp>	#<uwo> #< uwo,eng> #< uwo,eng,ftp>
root	**<ca, uwo,eng,ftp>**	#<ca>	<uwo,eng,ftp>	#<uwo > #<uwo,eng> #<uwo,eng,ftp>	#<ca> #<ca,uwo > #<ca,uwo,eng> #<ca,uwo,eng,ftp>

- Recursive name resolution of ***<ca, uwo,eng, ftp>.***
- Name servers cache intermediate results for subsequent lookups.

Remote Procedure Call



- Steps in making a remote procedure call

Steps of a Remote Procedure Call

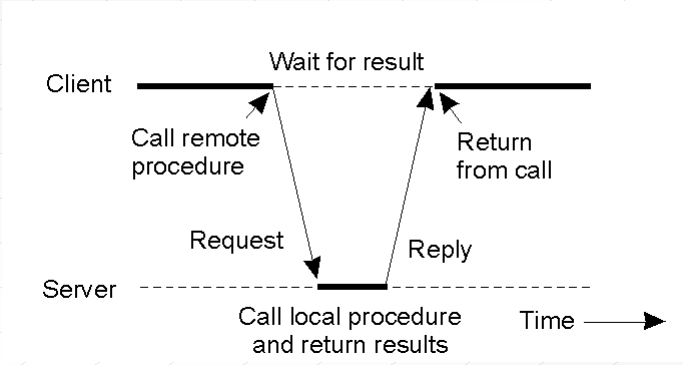
1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS

4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS

9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

Transfer of Control
Client Server Arch.

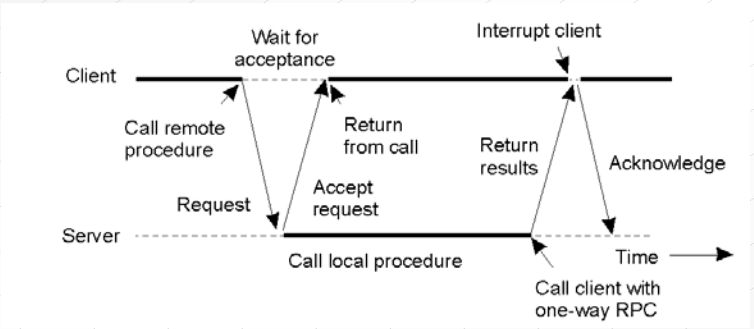
- Principle of RPC between a client and server program.



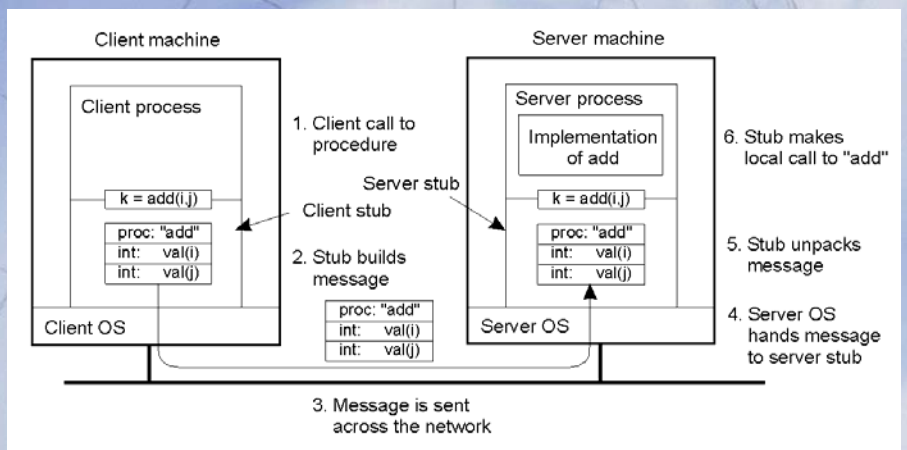
Transfer of Control

Asynchronous RPC

- A client and server interacting through two asynchronous RPCs

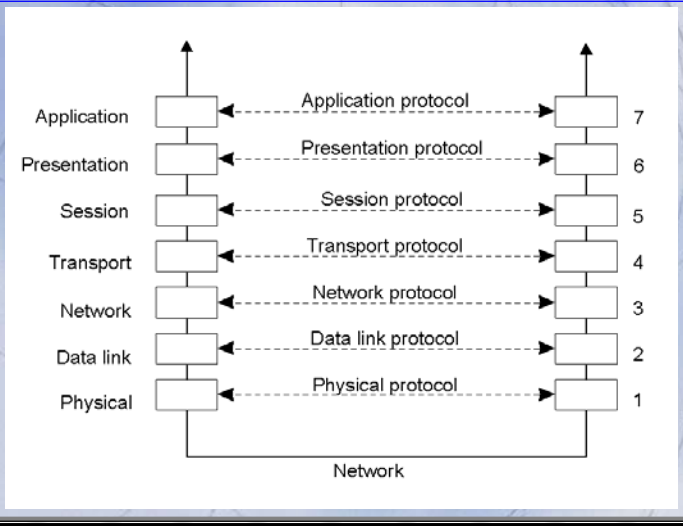


Passing Value Parameters



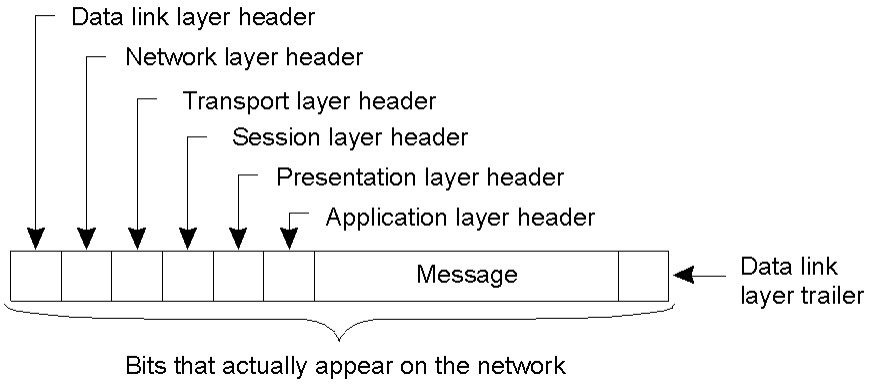
- Steps involved in doing remote computation through RPC

Parameter Passing
Communication: Layered Protocols



- Layers, interfaces, and protocols in the OSI model.

Layered Protocols



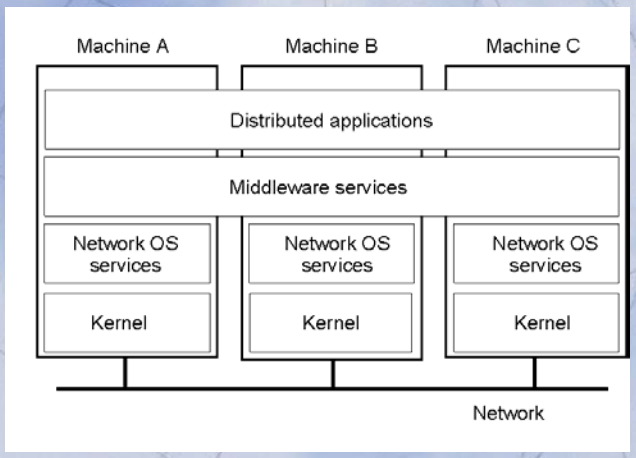
- A typical message as it appears on the network.

Implementation Issues
RPC

Implementation Issues

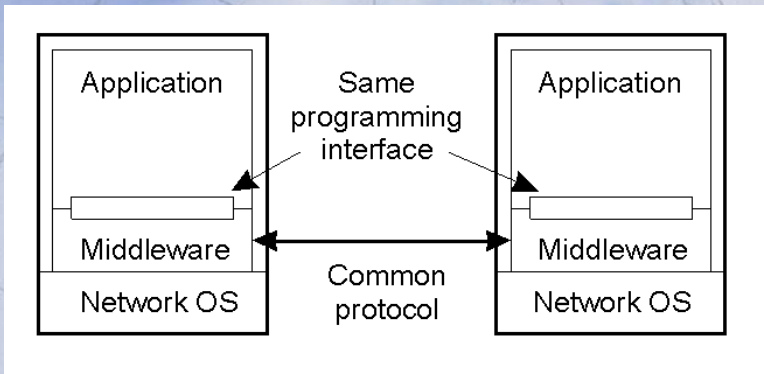
- cannot pass pointers
  - call by reference becomes copy-restore (but might fail)
- weakly typed languages
  - client stub cannot determine size
- cannot use global variables
  - may get moved to remote machine

Middleware-based Distributed Systems



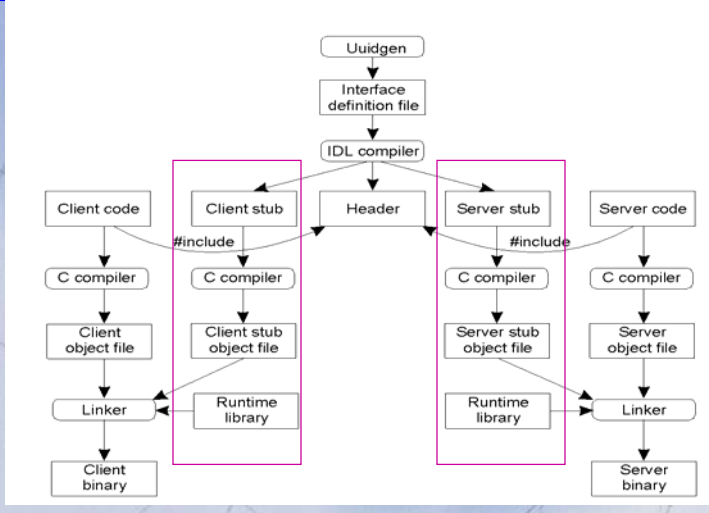
- General structure of a distributed system as middleware.

Middleware and Openness



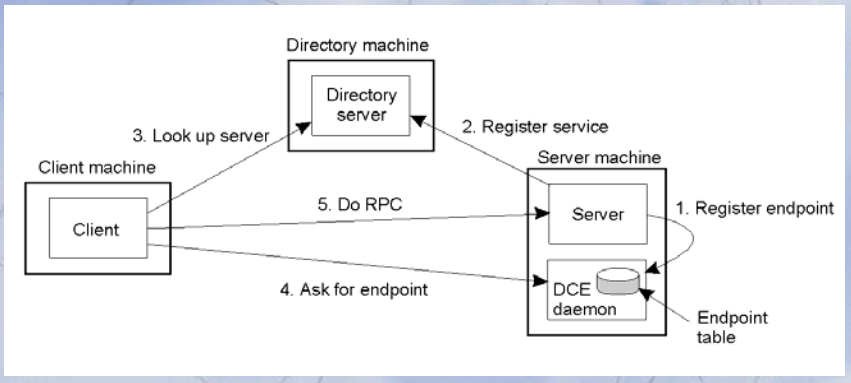
- In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.

Linking: Client and Server



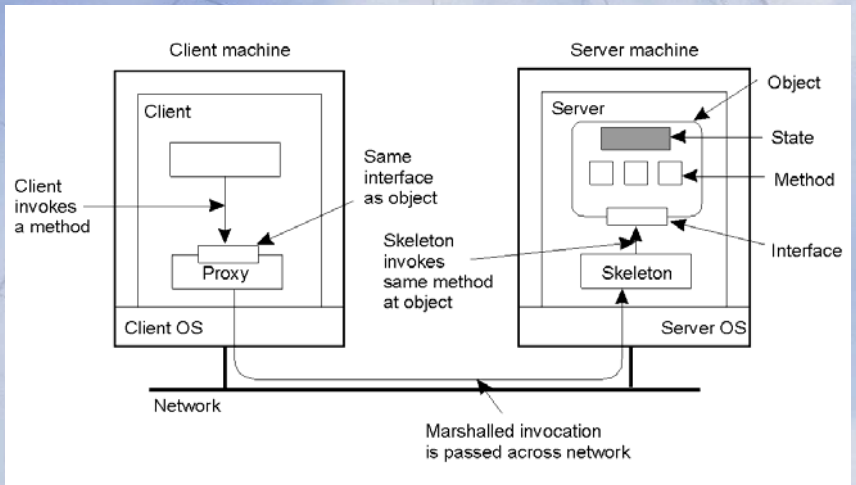
- The steps in writing a client and a server in DCE RPC.

Binding a Client to a Server



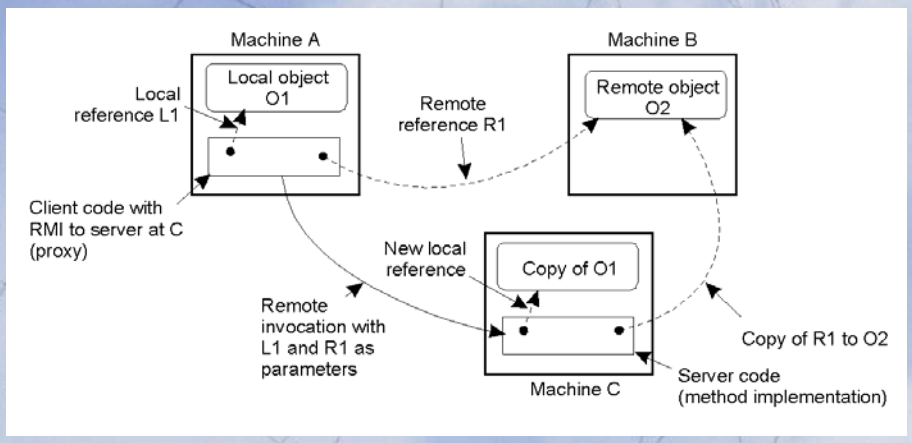
- Client-to-server binding in DCE.

Distributed Objects



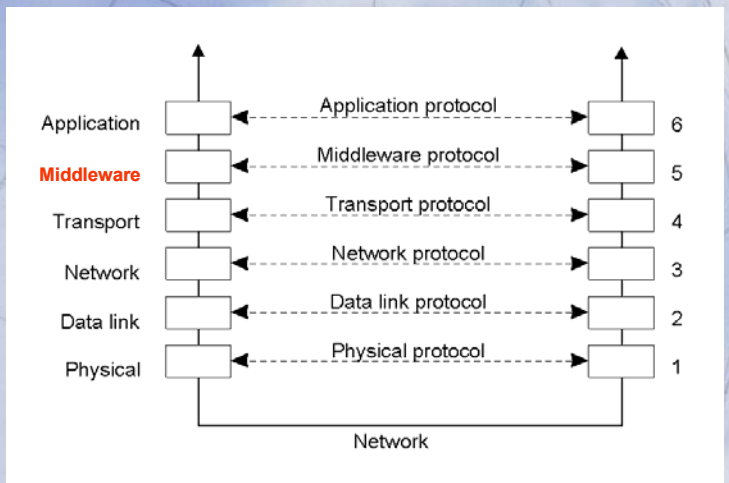
- Common organization of a remote object with client-side proxy.

Parameter Passing



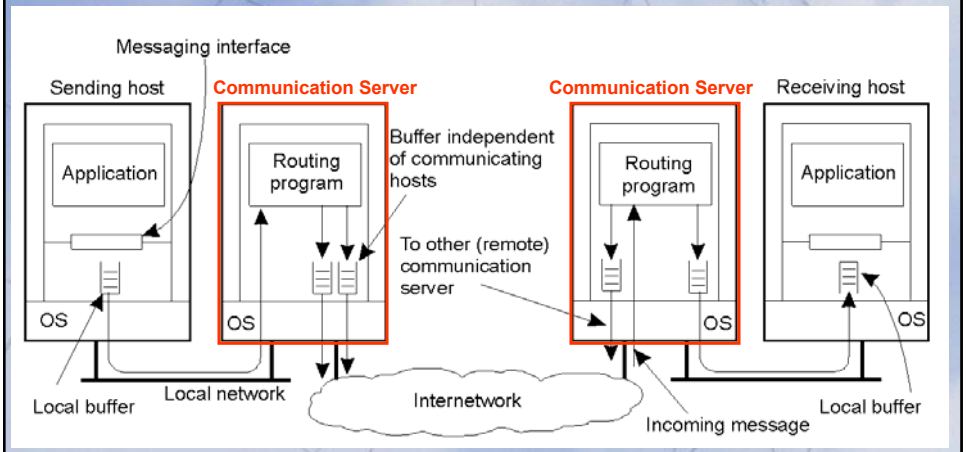
- The situation when passing an object by reference or by value.

Parameter Passing
Communication: Layered Protocols



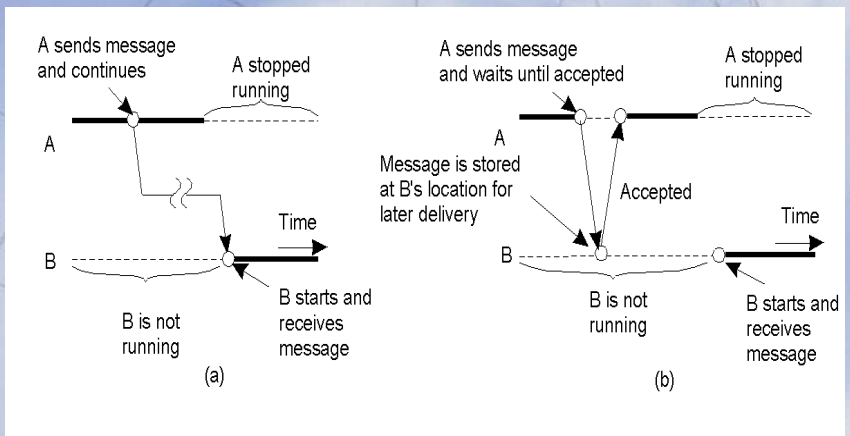
- An adapted reference model for networked communication.

Persistence and Synchronicity in Communication



- General organization of a communication system in which hosts are connected through a network

Transfer of Control:
Persistence and Synchronicity in Communication



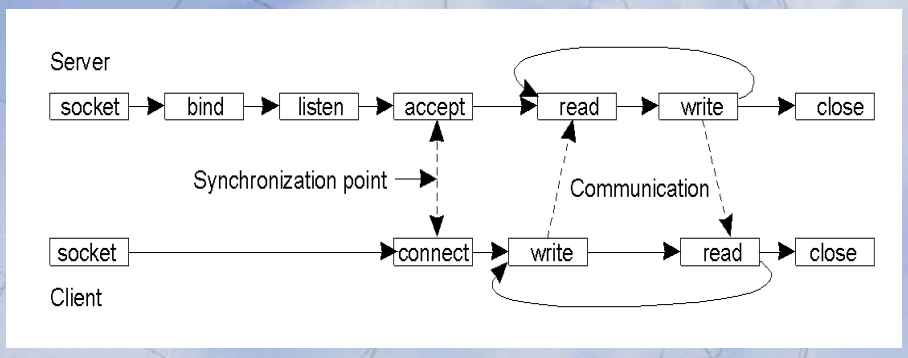
a) Persistent asynchronous communication
b) Persistent synchronous communication

Berkeley Sockets

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

- Socket primitives for TCP/IP.

Berkeley Sockets *(Cont.)*



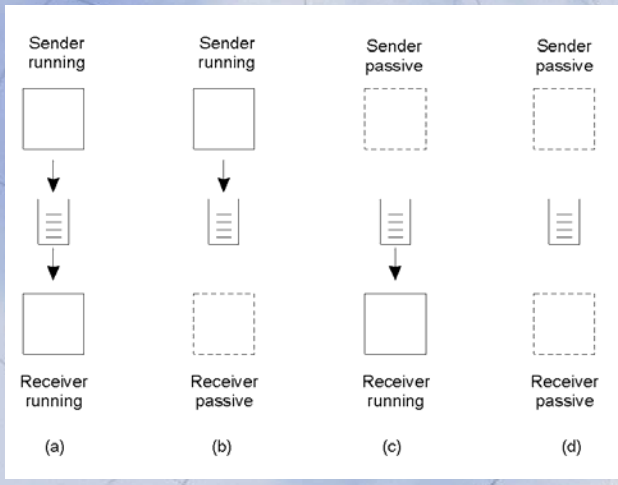
- Connection-oriented communication pattern using sockets.

The Message-Passing Interface (MPI)

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there are none
MPI_irecv	Check if there is an incoming message, but do not block

- Some of the most intuitive message-passing primitives of MPI.

Message-Queuing Model



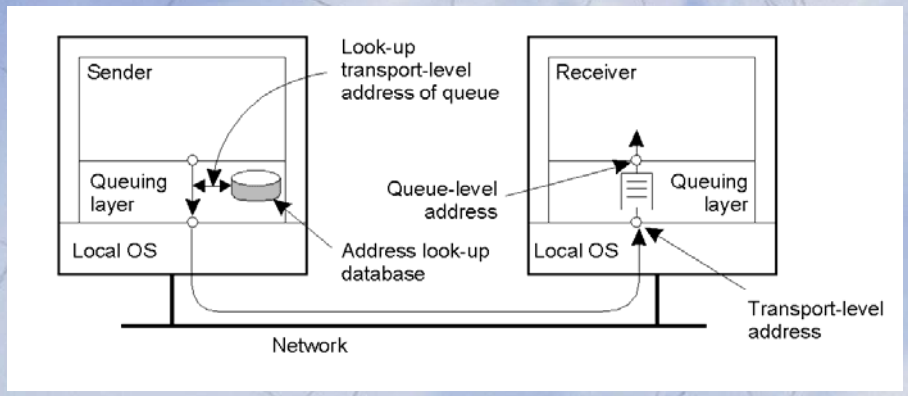
- Four combinations for loosely-coupled communications using queues.

Message-Queuing Model *(Cont.)*

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into the specified queue.

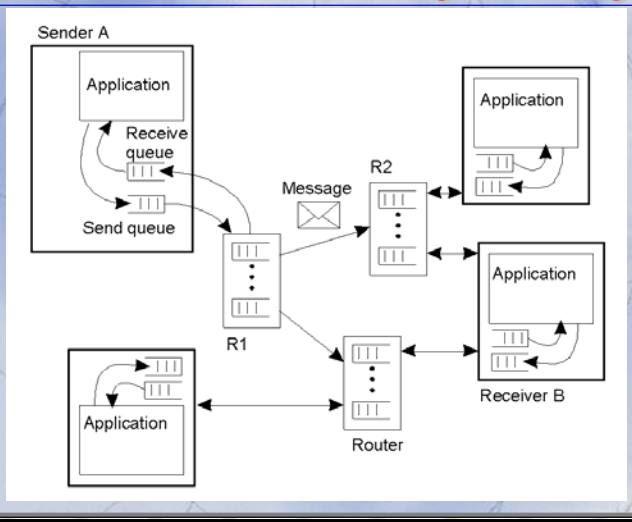
- Basic interface to a queue in a message-queuing system.

General Architecture of a Message-Queuing System



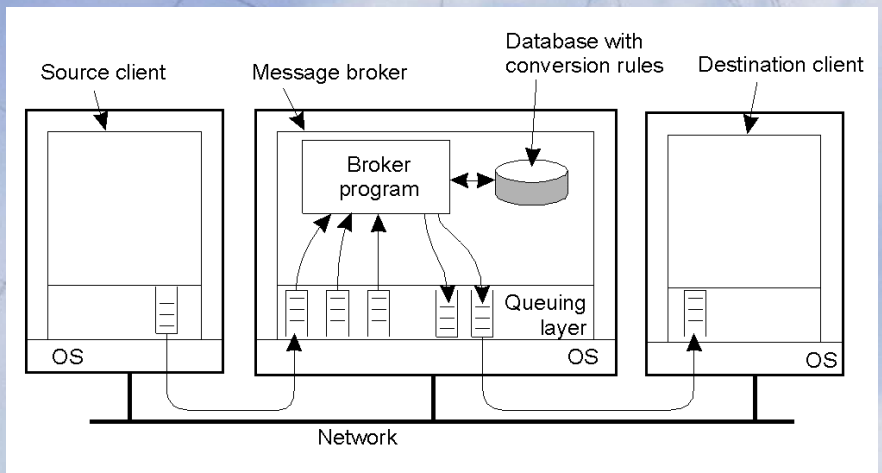
- The relationship between queue-level addressing and network-level addressing.

General Architecture of a Message-Queuing System



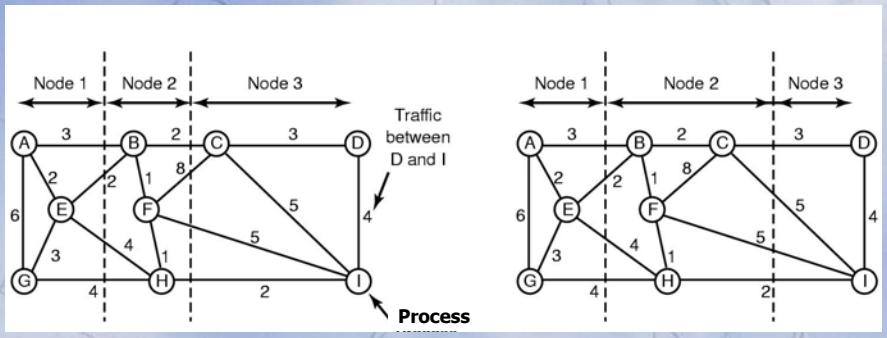
- The general organization of a message-queuing system with routers.

Message Brokers



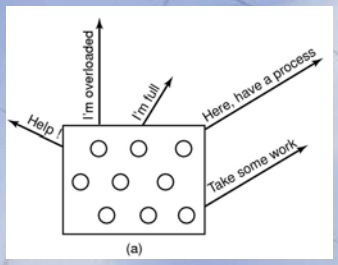
- The general organization of a message broker in a message-queuing system.

Multicomputer Scheduling
Load Balancing

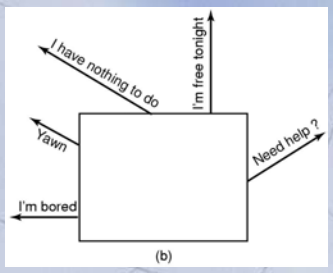


- Graph-theoretic deterministic algorithm

Multicomputer Scheduling
Load Balancing



(a) Sender-initiated distributed heuristic algorithm
- overloaded sender



(b) Receiver-initiated distributed heuristic algorithm
- under loaded receiver

Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multi-Proc.	Multi-Comp.		
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