UNIT-3
DISTRIBUTED SYSTEMS

PROCESS: Threads: Introduction to Threads, Threads in Distributed Systems, Clients: user Interfaces, Client-Side Software for Distribution Transparency, Servers: General Design Issues, Object Servers, Software Agents: Software Agents in Distributed Systems, Agent Technology, Naming: Naming Entities: Names, Identifiers, and Address, Name Resolution, The Implementation of a Name System, Locating Mobile Entities: Naming verses Locating Entities, Simple Solutions, Home-Based Approaches, Hierarchical Approaches.

PROCESS

A program under execution is called as a process.

THREAD

1. It is a light weight program.
2. **Traditional operating systems**: concerned with the "local" management and scheduling of processes.
3. **Modern distributed systems**: a number of other issues are of equal importance.
4. **There are three main areas of study**
    a. Threads and virtualization within clients/servers.
    b. Process and code migration.
    c. Software agents.
5. Modern OSs provide "virtual processors" within which programs execute.
6. A programs execution environment is documented in the process table and assigned a PID.
7. To achieve acceptable performance in distributed systems, relying on the OS's idea of a process is often not enough – finer granularity is required.
    - The solution: Threading.

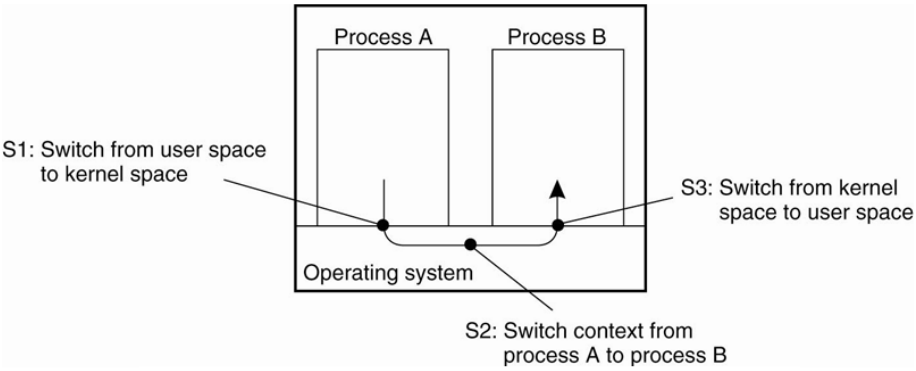
PROBLEMS WITH PROCESSES

1. Creating and managing processes is generally regarded as an expensive task (fork system call).
2. Making sure all the processes peacefully co-exist on the system is not easy (as concurrency transparency comes at a price).
3. **Threads** can be thought of as an "execution of a part of a program (in user-space)".
4. Rather than make the OS responsible for concurrency transparency, it is left to the individual application to manage the creation and scheduling of each thread.

Two Important Implications

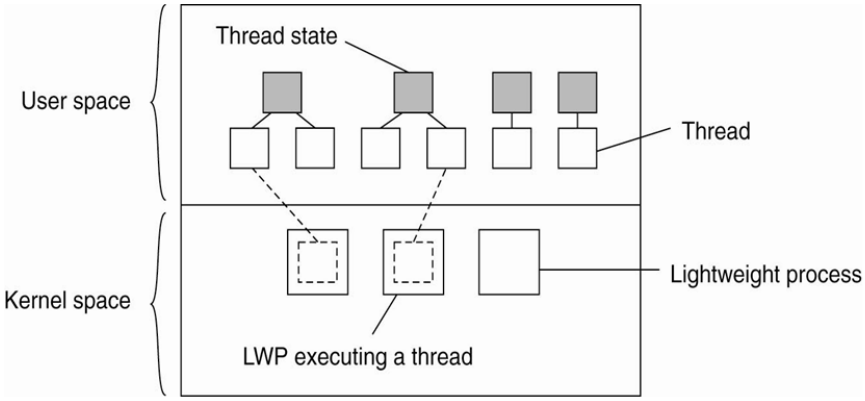
1. Threaded applications often run faster than non-threaded applications (as context-switches between kernel and user-space are avoided).
2. Threaded applications are harder to develop (although simple, clean designs can help here).
Additionally, the assumption is that the development environment provides a Threads Library for developers to use (most modern environments do).

THREAD USAGE IN NON-DISTRIBUTED SYSTEMS



Context switching as the result of IPC

THREAD IMPLEMENTATION



Combining kernel-level lightweight processes and user-level threads

THREADS IN NON-DISTRIBUTED SYSTEMS

Advantages:

1. Blocking can be avoided
2. Excellent support for multi-processor systems (each running their own thread).
3. Expensive context-switches can be avoided.
4. For certain classes of application, the design and implementation is made considerably easier.

THREADS IN DISTRIBUTED SYSTEMS

1. Important characteristic: a blocking call in a thread does not result in the entire process being blocked.

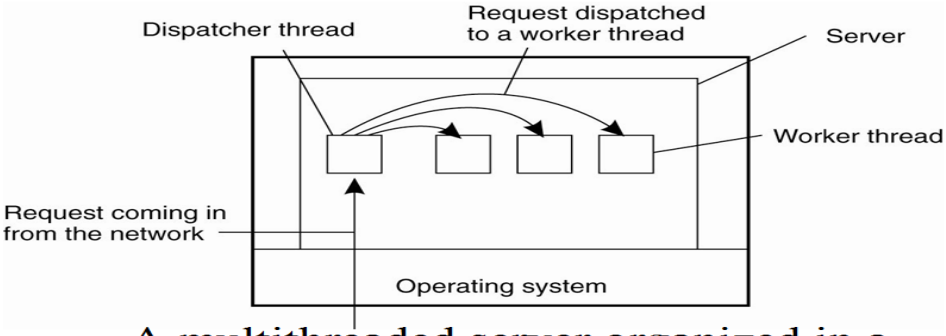
2. This leads to the key characteristic of threads within distributed systems:

"We can now express communications in the form of maintaining multiple logical connections at the same time (as opposed to a single, sequential, blocking process)."

Example: MT Clients and Servers

- Mutli-Threaded Client: to achieve acceptable levels of perceived performance, it is often necessary to hide communications latencies.
- Consequently, a requirement exists to start communications while doing something else.
- Example: modern Web browsers.
- This leads to the notion of "truly parallel streams of data" arriving at a multi-threaded client application.
- Although threading is useful on clients, it is much more useful in distributed systems servers.
- The main idea is to exploit parallelism to attain high performance.
- A typical design is to organize the server as a single "dispatcher" with multiple threaded "workers", as diagrammed overleaf.

MULTITHREADED SERVERS

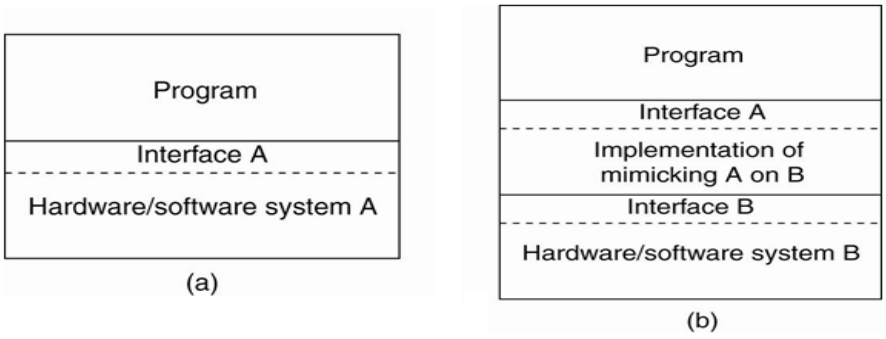


A multithreaded server organized in a dispatcher/worker model

MULTITHREADED SERVERS

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls

THE ROLE OF VIRTUALIZATION IN DISTRIBUTED SYSTEMS

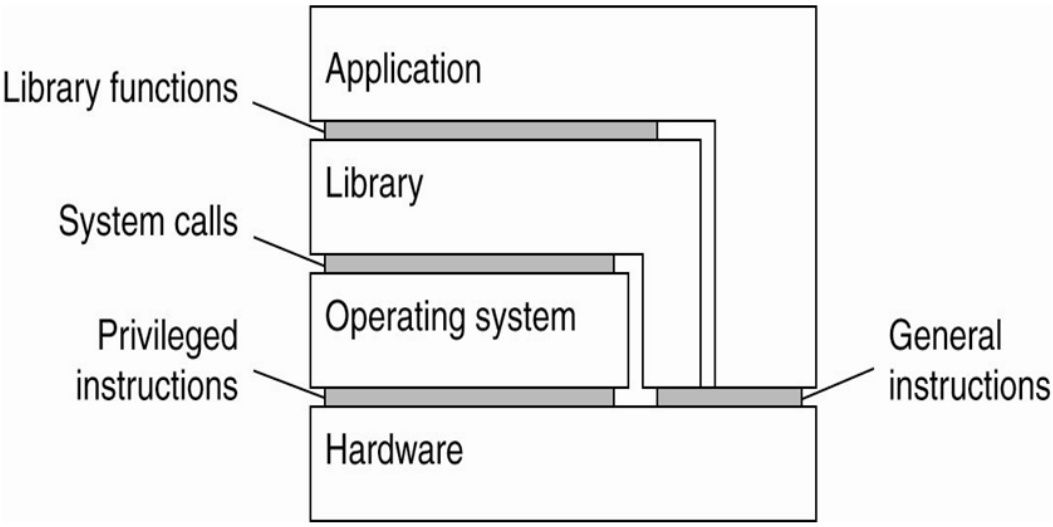


UNIT-3
DISTRIBUTED SYSTEMS

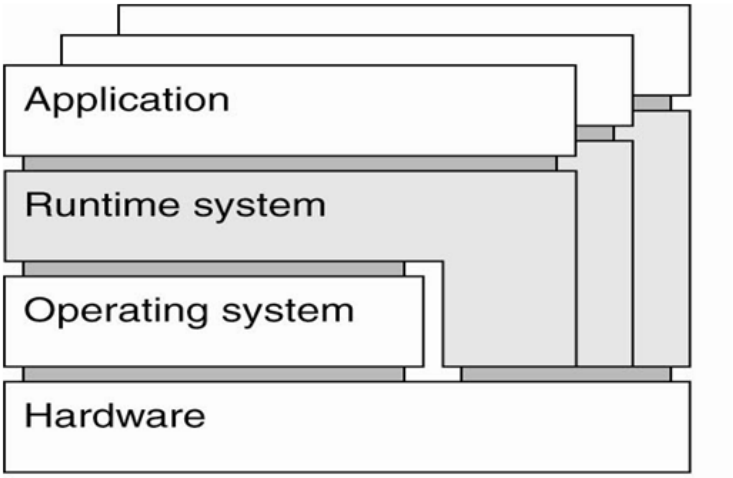
a. General organization between a program, interface, and system
b. General organization of virtualizing system A on top of system B

ARCHITECTURES OF VIRTUAL MACHINES

- There are interfaces at different levels.
- An interface between the hardware and software, consisting of machine instructions
  - that can be invoked by any program.
- An interface between the hardware and software, consisting of machine instructions
  - that can be invoked only by privileged programs, such as an operating system.
- An interface consisting of system calls as offered by an operating system.
- An interface consisting of library calls
  - generally forming what is known as an Application Programming Interface (API).
  - In many cases, the aforementioned system calls are hidden by an API.

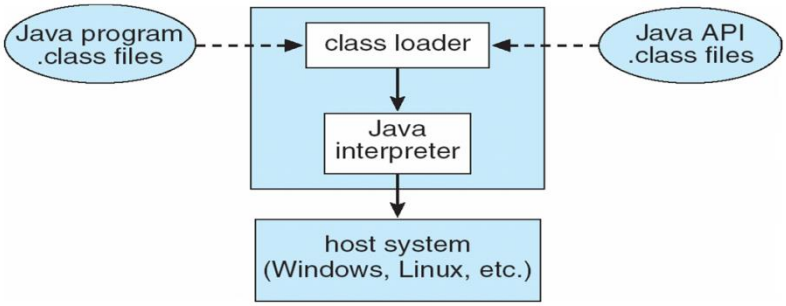


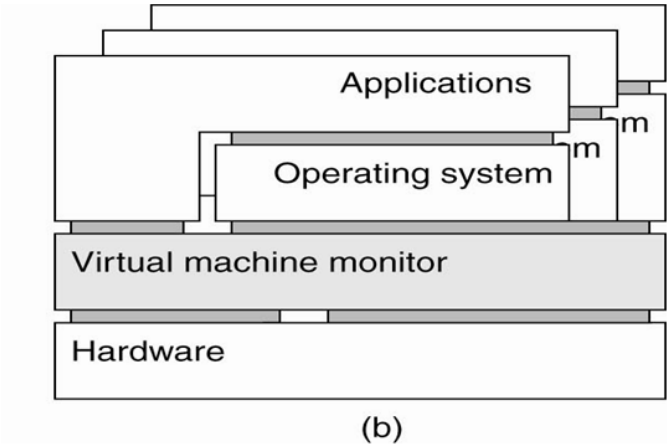
Various interfaces offered by computer



(a) A process virtual machine, with multiple instances of (application, runtime) combinations

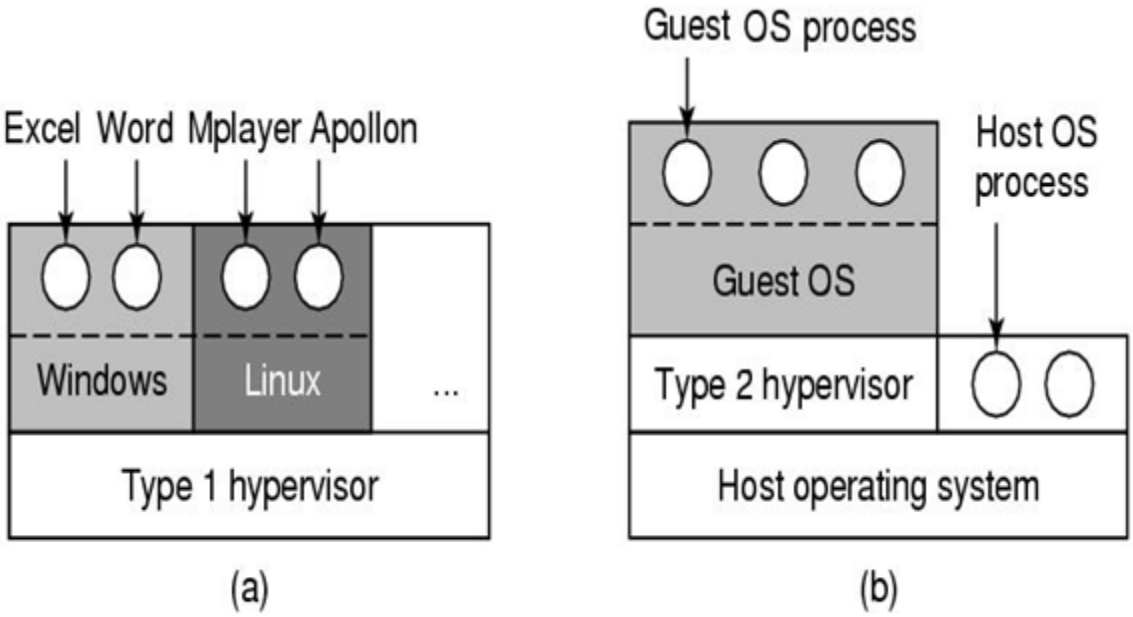
THE JAVA VIRTUAL MACHINE





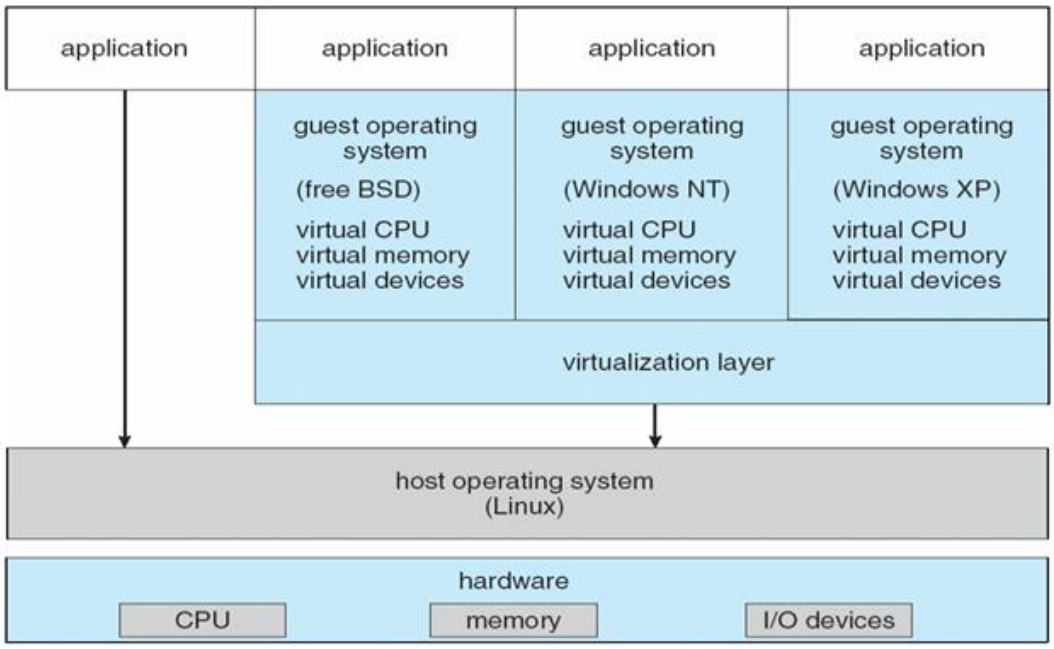
(b) A Virtual Machine Monitor (VMM), with multiple instances of (applications, operating system) combinations.

TYPES OF VMM/HYPERVISORS



(a) A type 1 hypervisor. (b) A type 2 hypervisor

VMWARE ARCHITECTURE

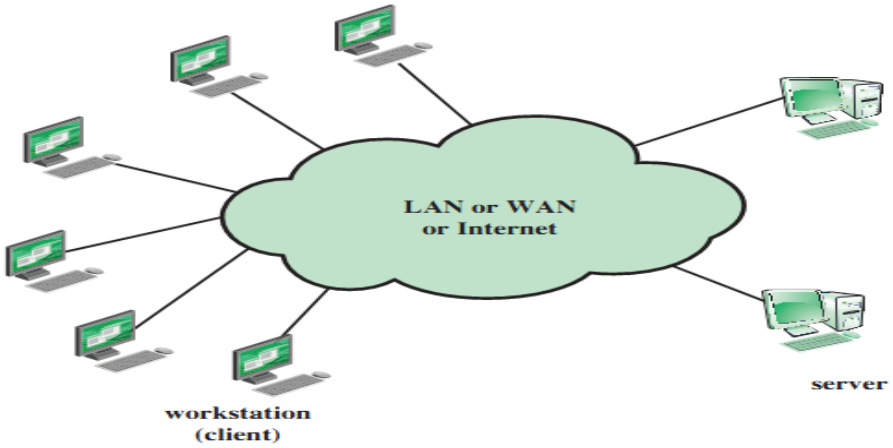


More on Clients

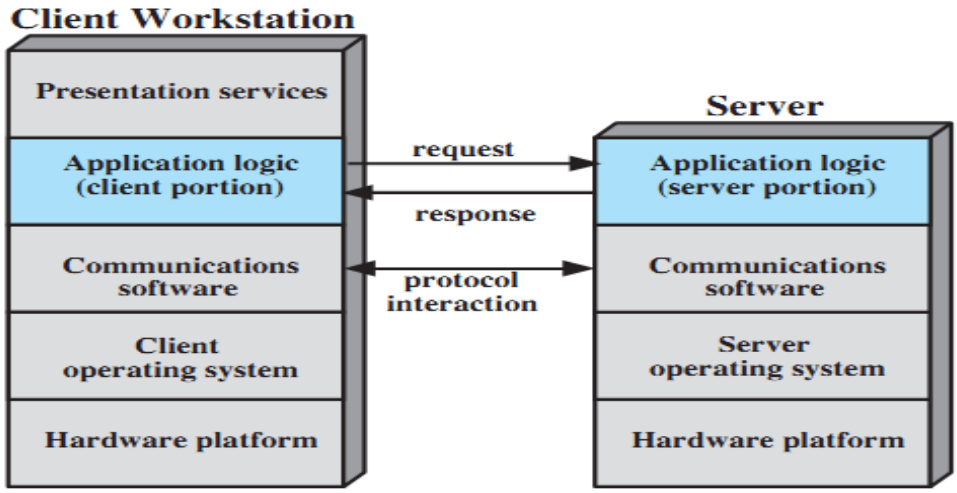
- **What's a client?**
- **Definition: "A program which interacts with a human user and a remote server."**
- **Typically, the user interacts with the client via a GUI.**

- **Of course, there's more to clients than simply providing a UI. Remember the multi-tiered levels of the Client/Server architecture from earlier …**

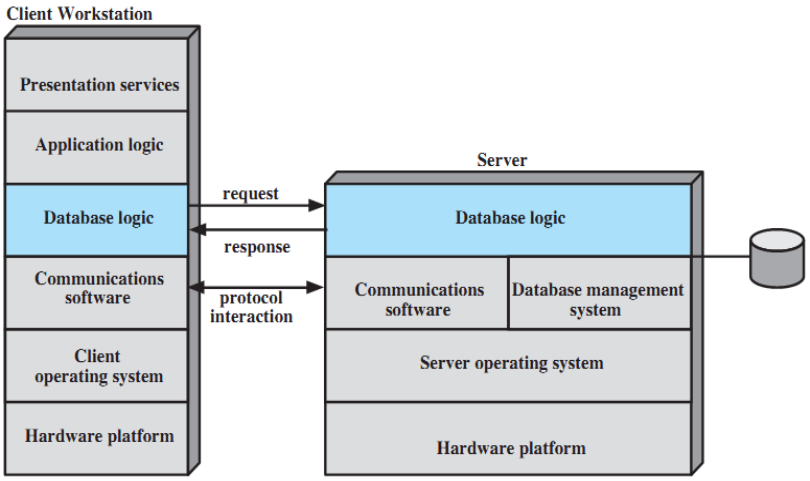
GENERIC CLIENT/SERVER ENVIRONMENT



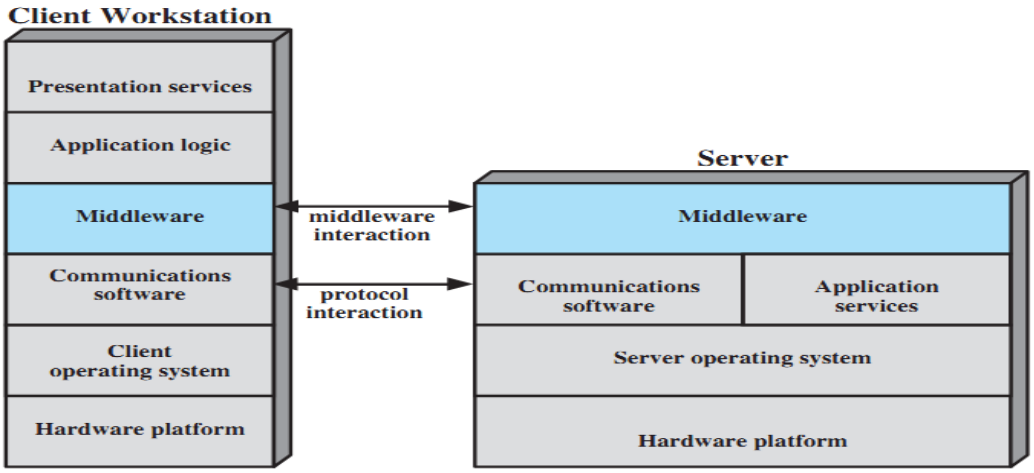
Generic Client/Server Architecture



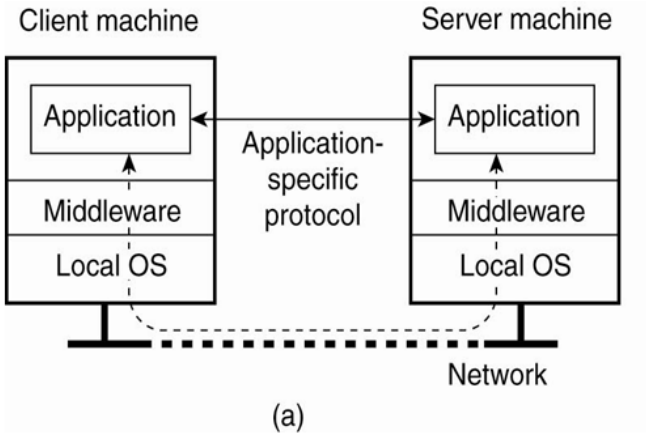
Client/Server Architecture for Database Applications



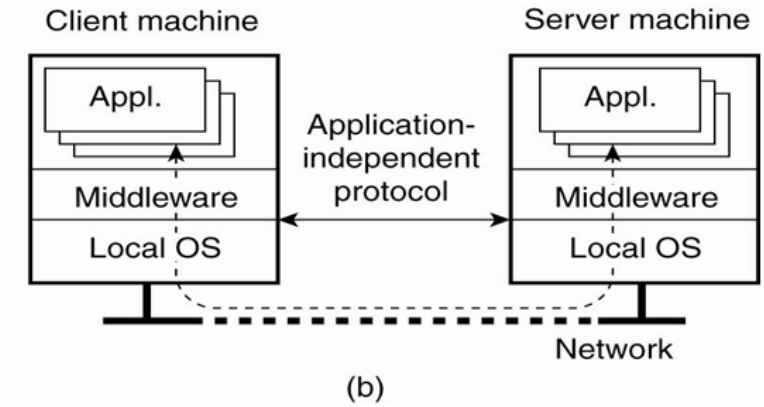
Role of Middleware in Client/Server Architecture



NETWORKED USER INTERFACES

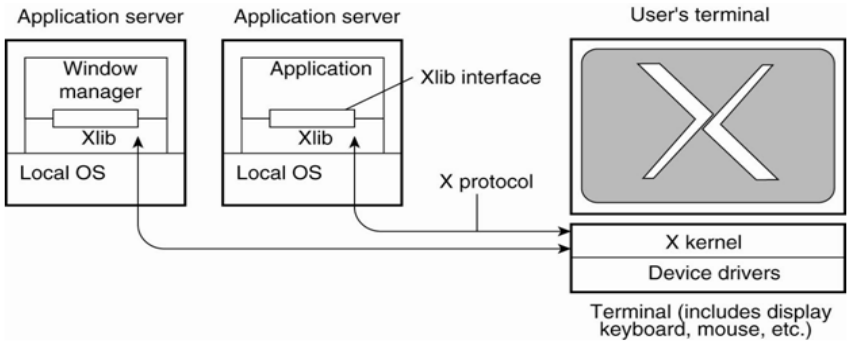


(a) A networked application with its own protocol



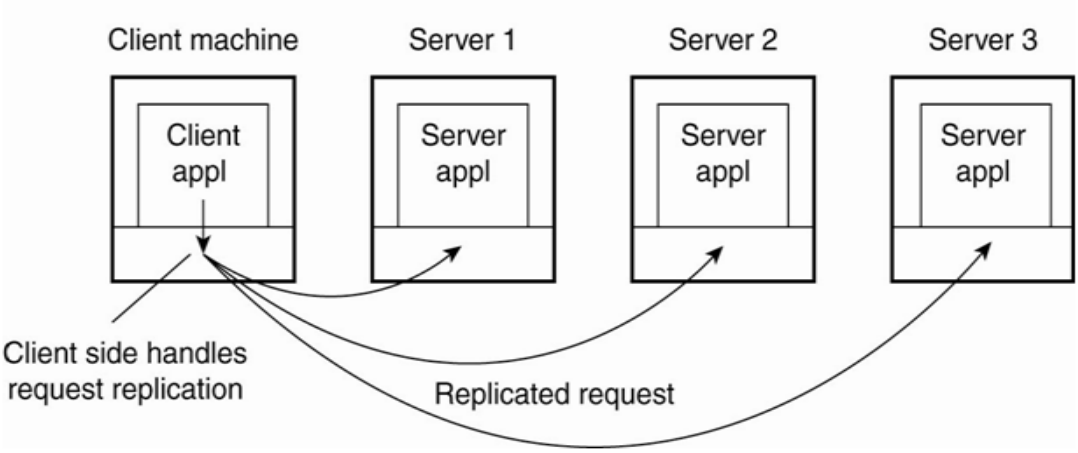
(b) A general solution to allow access to remote applications

Example: The X Window System



The basic organization of the X Window System

CLIENT-SIDE SOFTWARE FOR DISTRIBUTION TRANSPARENCY



Transparent replication of a server using a client-side solution

More on Servers

SERVER

"A process that implements a specific service on behalf of a collection of clients".Typically, servers are organized to do one of two things:

1. Wait. 2. Service.

… Wait … service … wait … service … wait …

SERVERS: ITERATIVE AND CONCURRENT

1. Iterative: server handles request, then returns results to the client; any new client requests must wait for previous request to complete
(also useful to think of this type of server as sequential).

2. Concurrent: server does not handle the request itself; a separate thread or sub-process handles the request and returns any results to the client; the server is then free to immediately service the next client (i.e., there's no waiting, as service requests are processed in parallel).

SERVER "STATES"

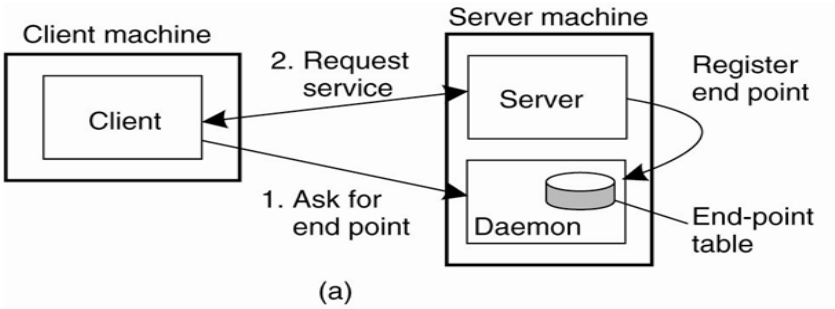
1. **Stateless servers** – no information is maintained on the current "connections" to the server. The Web is the classic example of a stateless service. As can be imagined, this type of server is **easy** to implement.

2. **Stateful servers** – information is maintained on the current "connections" to the server. Advanced file servers, where copies of a file can be updated "locally", then applied to the main server (as it knows the state of things) - more **difficult** to implement.

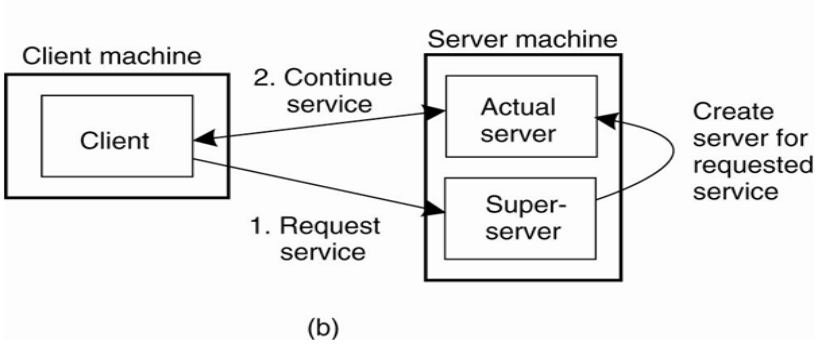
Problem: Identifying "end-points"?

- How do clients know which end-point (or port) to contact a server at?
  How do they "bind" to a server?
  - Statically assigned end-points (IANA).
  - Dynamically assigned end-points (DCE).
  - A popular variation:
    - the "super-server" (inetd on UNIX).

GENERAL DESIGN ISSUES



(a) Client-to-server binding using a daemon

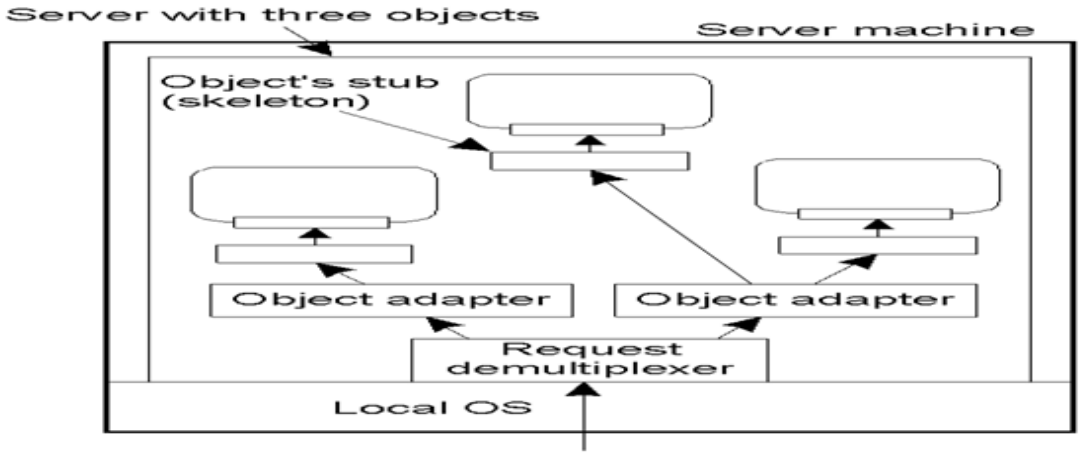


(b) Client-to-server binding using a superserver

A Special Type: Object Servers

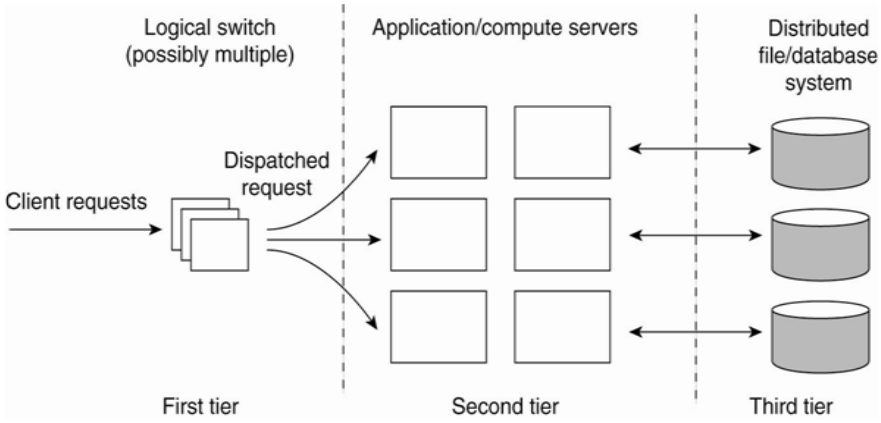
- A server tailored to support distributed objects.
- Does not provide a specific service.
- Provides a facility whereby objects can be remotely invoked by non-local clients.
- Consequently, object servers are highly adaptable.
- "A place where objects live".

Object Adapter

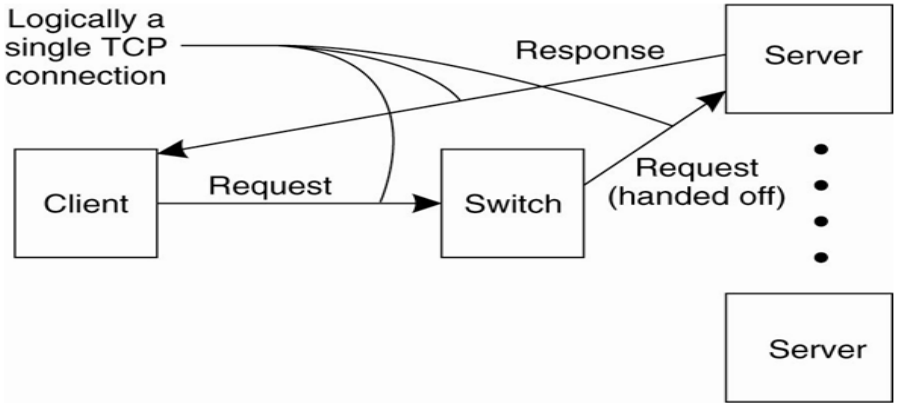


Organization of an object server supporting different activation policies.

SERVER CLUSTERS

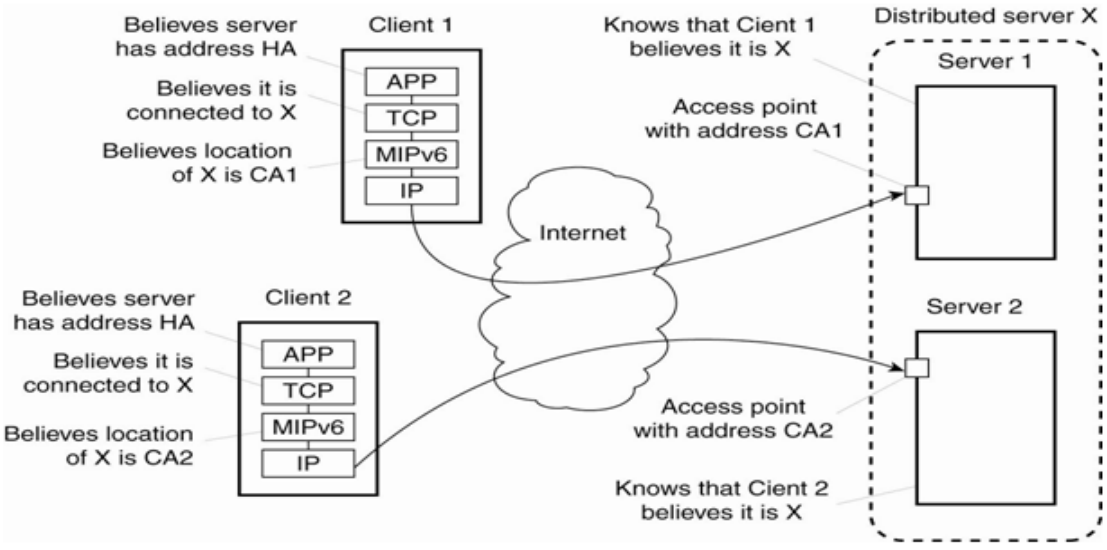


The general organization of a three-tiered server cluster



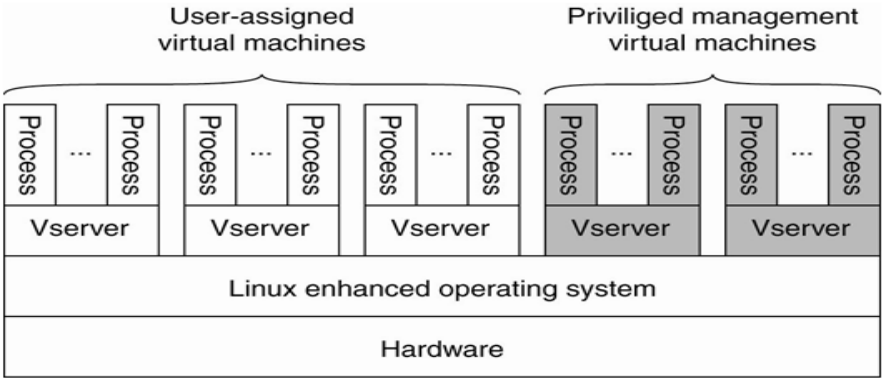
The principle of TCP handoff

DISTRIBUTED SERVERS



Route optimization in a distributed server

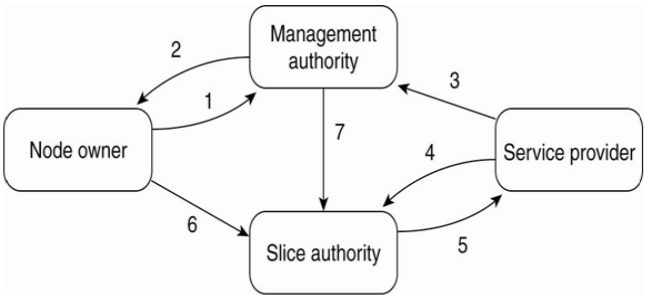
MANAGING SERVER CLUSTERS



The basic organization of a PlanetLab node

PlanetLab

- PlanetLab management issues:
- Nodes belong to different organizations.
  - Each organization should be allowed to specify who is allowed to run applications on their nodes,
  - And restrict resource usage appropriately.
- Monitoring tools available assume a very specific combination of hardware and software.
  - All tailored to be used within a single organization.
- Programs from different slices but running on the same node should not interfere with each other.



The management relationships between various PlanetLab entities

- Relationships between PlanetLab entities:
- A node owner puts its node under the regime of a management authority, possibly restricting usage where appropriate.
- A management authority provides the necessary software to add a node to PlanetLab.
- A service provider registers itself with a management authority, trusting it to provide well-behaving nodes.
- A service provider contacts a slice authority to create a slice on a collection of nodes.
- The slice authority needs to authenticate the service provider.
- A node owner provides a slice creation service for a slice authority to create slices. It essentially delegates resource management to the slice authority.
- A management authority delegates the creation of slices to a slice authority.

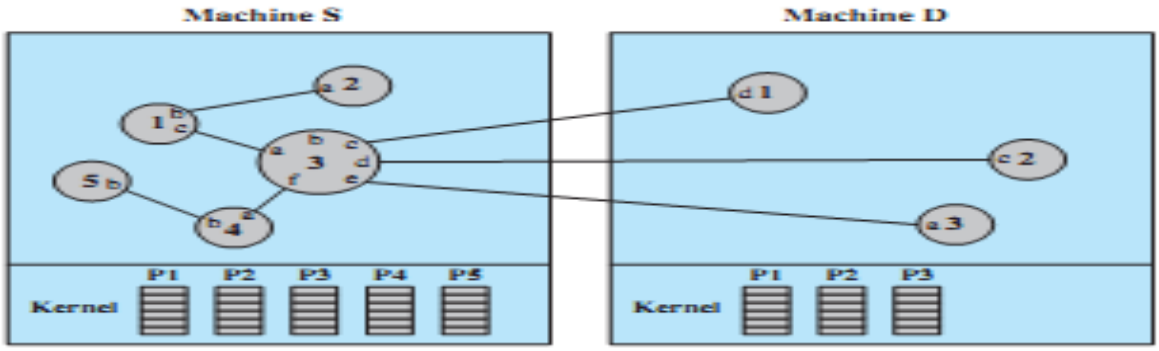
PROCESS AND CODE MIGRATION

- Under certain circumstances, in addition to the usual passing of data, passing code (even while it is executing) can greatly simplify the design of a DS.
- However, code migration can be inefficient and very costly.
- So, why migrate code?

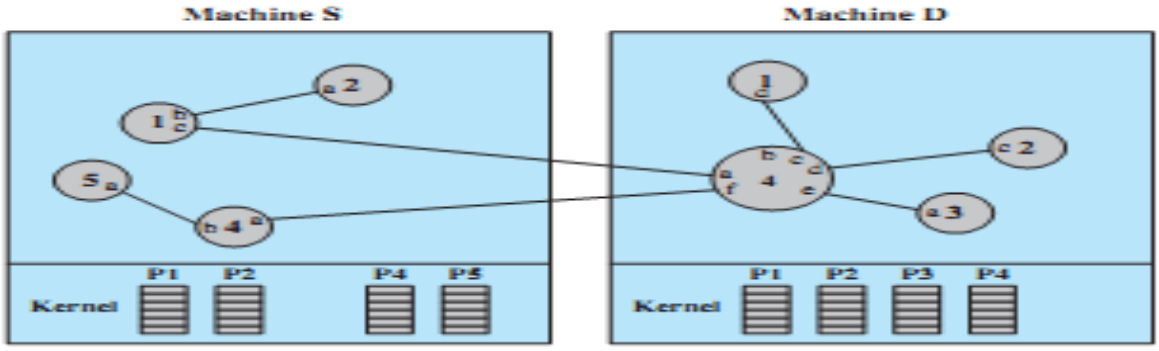
REASONS FOR MIGRATING CODE

- Why? Biggest single reason: **better performance**.
- The big idea is to move a compute-intensive task from a heavily loaded machine to a lightly loaded machine
  "on demand" and "as required".

Example of Process Migration



(a) Before migration



(b) After migration

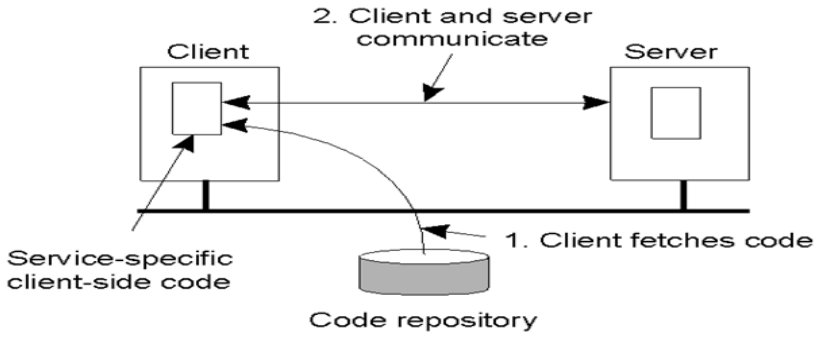
CODE MIGRATION EXAMPLES

- Moving (part of) a client to a server – processing data close to where the data resides. It is often too expensive to transport an entire database to a client for processing, so move the client to the data.
- Moving (part of) a server to a client –
  checking data prior to submitting it to a server. The use of local error-checking (using JavaScript) on Web forms is a good example of this type of processing. Error-check the data close to the user, not at the server.

"Classic" Code Migration Example

- Searching the Web by "roaming".
- Rather than search and index the Web by requesting the transfer of each and every document to the client for processing, the
  client relocates to each site and indexes the documents it finds "in situ". The index is then transported from site to site, in addition to the executing process.

ANOTHER BIG ADVANTAGE: FLEXIBILITY



The principle of dynamically configuring a client to communicate to a server. The client first fetches the necessary software, and then invokes the server. This is a very flexible approach.

Major Disadvantage

- **Security Concerns**.
- "Blindly trusting that the downloaded code implements only the advertised interface while accessing your unprotected hard-disk and does not send the juiciest parts to heaven-knows-where may not always be such a good idea".

CODE MIGRATION MODELS

A RUNNING PROCESS CONSISTS OF THREE "SEGMENTS":

1. Code – instructions.
2. Resource – external references.
3. Execution – current state.

MIGRATION IN HETEROGENEOUS SYSTEMS

Three ways to handle migration (which can be combined):

1. Pushing memory pages to the new machine and resending the ones that are later modified during the migration process.
2. Stopping the current virtual machine; migrate memory, and start the new virtual machine.
3. Letting the new virtual machine pull in new pages as needed, that is, let processes start on the new virtual machine immediately and copy memory pages on demand.

CODE MIGRATION CHARACTERISTICS

Weak Mobility: just the code is moved – and it always restarts from its initial state.

- e.g., Java Applets.
- Comment: simple implementation, but limited applicability.

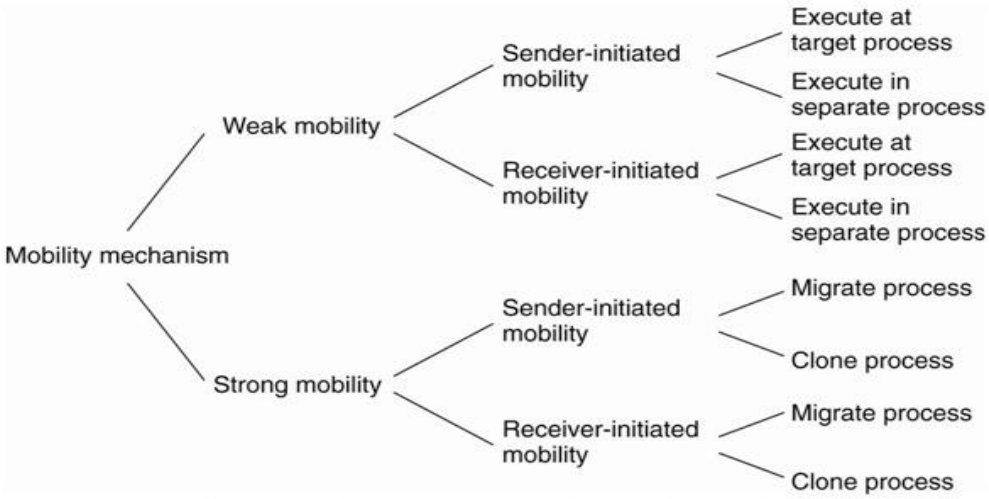
Strong Mobility: code & state is moved – and execution restarts from the next statement.

  - e.g., D'Agents.
  - Comment: very powerful, but hard to implement.
- Sender-Initiated vs. Receiver-Initiated.
- Which side of the communication starts the migration?
  - The machine currently executing the code (known as sender-initiated)
  - The machine that will ultimately execute the code (known as receiver-initiated).

HOW DOES THE MIGRATED CODE RUN?

- Another issue surrounds where the migrated code executes:
  1. Within an existing process (possibly as a thread)
  2. Within it's own (new) process space.
- Finally, strong mobility also supports the notion of "remote cloning": an exact copy of the original process, but now running on a different machine.

MODELS FOR CODE MIGRATION



Alternatives for code migration

What About Resources?

- What makes code migration difficult is the requirement to migrate resources.
- Resources are the external references that a process is currently using, and includes (but is not limited to):
  - Variables, open files, network connections, printers, databases, etc...

TYPES OF PROCESS-TO-RESOURCE BINDING

1. Strongest: Binding-by-Identifier (BI) – precisely the referenced resource, and nothing else, has to be migrated.

2. Binding-by-Value (BV) –
weaker than BI, but only the value of the resource need be migrated.

3. Weakest: Binding-by-Type (BT) –
nothing is migrated, but a resource of a specific type needs to be available after migration (e.g., a printer).

MORE RESOURCE CLASSIFICATION

- Resources are further distinguished as one of:
    1. **Unattached**: a resource that can be moved easily from machine to machine.
    2. **Fastened**: migration is possible, but at a high cost.
    3. **Fixed**: a resource is bound to a specific machine or environment, and cannot be migrated.
- Refer to following diagram for a good summary of resource-to-binding characteristics (to find out what to do with which resource when).

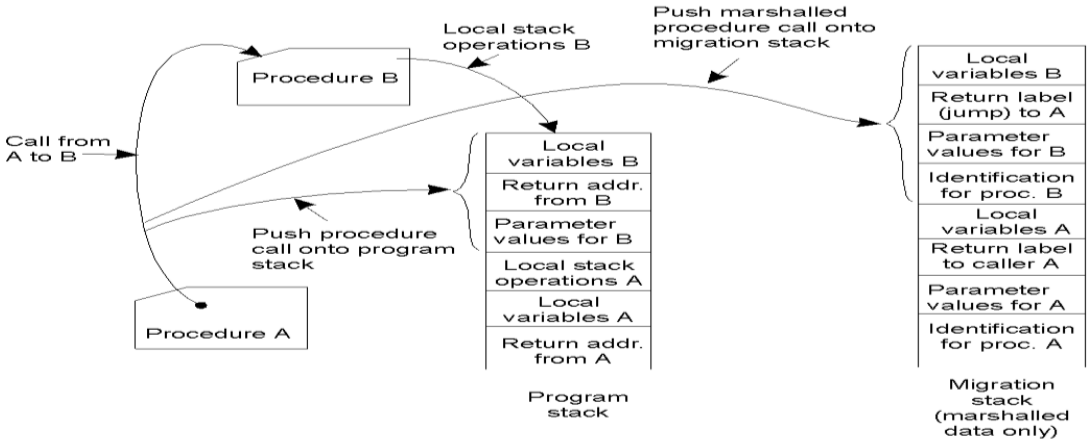
MIGRATION AND LOCAL RESOURCES

		Resource-to-machine binding		
		Unattached	Fastened	Fixed
Process-to-resource binding	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV,GR)	GR (or CP)	GR
	By type	RB (or MV,CP)	RB (or GR,CP)	RB (or GR)

GR Establish a global systemwide reference
MV Move the resource
CP Copy the value of the resource
RB Rebind process to locally-available resource

Actions to be taken with respect to the references to local resources when migrating code to another machine

Migration in Heterogeneous DS's



Using a migration stack: the principle of maintaining a migration stack to support migration of an execution segment in a heterogeneous environment. Usually requires changes to the programming language and its environment.

Overview of Code Migration in D'Agents (1)

```
proc factorial n {
   if ($n ≤ 1) { return 1; }                  # fac(1) = 1
   expr $n * [ factorial [expr $n – 1] ]      # fac(n) = n * fac(n – 1)
}
set number …        # tells which factorial to compute
set machine …       # identify the target machine

agent_submit $machine –procs factorial –vars number –script {factorial $number }

agent_receive …     # receive the results (left unspecified for simplicity)
```

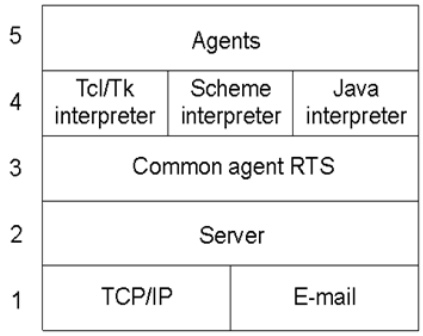
A simple example of a Tel agent in D'Agents submitting a script to a remote machine (adapted from [Gray 95])

Overview of Code Migration in D'Agents (2)

```
all_users $machines

proc all_users machines {
   set list ""                          # Create an initially empty list
   foreach m $machines {  # Consider all hosts in the set of given machines
      agent_jump $m                     # Jump to each host
      set users [exec who]  # Execute the who command
      append list $users    # Append the results to the list
   }
   return $list             # Return the complete list when done
}
set machines …                          # Initialize the set of machines to jump to
set this_machine                        # Set to the host that starts the agent

# Create a migrating agent by submitting the script to this machine, from where
# it will jump to all the others in  $machines.

agent_submit $this_machine –procs all_users
                                        -vars  machines
                                        -script { all_users $machines }
agent_receive …                         #receive the results  (left unspecified for simplicity)
```

An example of a Tel agent in D'Agents migrating to different machines where it executes the UNIX *who* command.

Implementation Issues



The architecture of the D'Agents system.

Implementation Issues

Status	Description
Global interpreter variables	Variables needed by the interpreter of an agent
Global system variables	Return codes, error codes, error strings, etc.
Global program variables	User-defined global variables in a program
Procedure definitions	Definitions of scripts to be executed by an agent
Stack of commands	Stack of commands currently being executed
Stack of call frames	Stack of activation records, one for each running command

The parts comprising the state of an agent in D'Agents.

SOFTWARE AGENTS

- What is a software agent?
  - "An autonomous unit capable of performing a task in collaboration with other, possibly remote, agents".
- The field of Software Agents is still immature, and much disagreement exists as to how to define what we mean by them.
- However, a number of types can be identified.

TYPES OF SOFTWARE AGENTS

1. Collaborative Agent – also known as "multi-agent systems", which can work together to achieve a
   common goal (e.g., planning a meeting).
2. Mobile Agent – code that can relocate and continue executing on a remote machine.

3. Interface Agent – software with "learning abilities" (that damned MS paperclip, and the ill-fated "bob").
4. Information Agent – agents that are designed to collect and process geographically dispersed data and information.

Software Agents in Distributed Systems

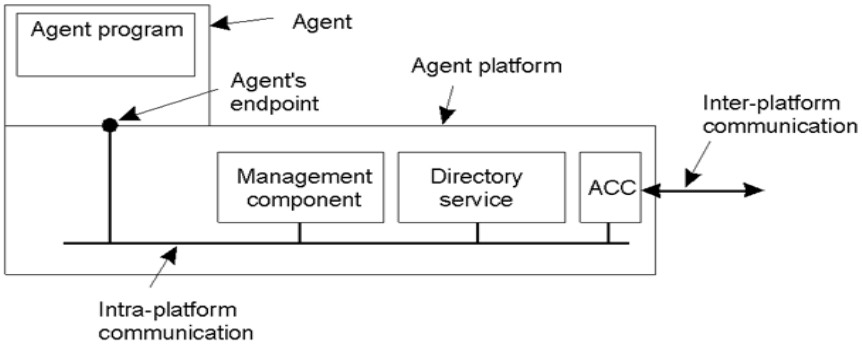
Property	Common to all agents?	Description
Autonomous	Yes	Can act on its own
Reactive	Yes	Responds timely to changes in its environment
Proactive	Yes	Initiates actions that affects its environment
Communicative	Yes	Can exchange information with users and other agents
Continuous	No	Has a relatively long lifespan
Mobile	No	Can migrate from one site to another
Adaptive	No	Capable of learning

Some important properties by which different types of agents can be distinguished.

Agent Technology – Standards

- The general model of an agent platform has been standardized by FIPA ("Foundation for Intelligent Physical Agents") located at http://www.fipa.org
- Specifications include:
  - Agent Management Component.
  - Agent Directory Service.
  - Agent Communication Channel.
  - Agent Communication Language.

AGENT TECHNOLOGY



The general model of an agent platform (adapted from [FIPA 1998])

AGENT COMMUNICATION LANGUAGES

Message purpose	Description	Message Content
INFORM	Inform that a given proposition is true	Proposition
QUERY-IF	Query whether a given proposition is true	Proposition
QUERY-REF	Query for a give object	Expression
CFP	Ask for a proposal	Proposal specifics
PROPOSE	Provide a proposal	Proposal
ACCEPT-PROPOSAL	Tell that a given proposal is accepted	Proposal ID
REJECT-PROPOSAL	Tell that a given proposal is rejected	Proposal ID
REQUEST	Request that an action be performed	Action specification
SUBSCRIBE	Subscribe to an information source	Reference to source

Examples of different message types in the FIPA ACL [FIPA 1998], giving the purpose of a message, along with the description of the actual message content.

AGENT COMMUNICATION LANGUAGES

Field	Value
Purpose	INFORM
Sender	max@http://fanclub-beatrix.royalty-spotters.nl:7239
Receiver	elke@iiop://royalty-watcher.uk:5623
Language	Prolog
Ontology	genealogy
Content	female(beatrix),parent(beatrix,juliana,bernhard)

A simple example of a FIPA ACL message sent between two agents using Prolog to express genealogy information.

Naming Entities

A name in a distributed system is a string of bits or characters that is used to refer to an entity

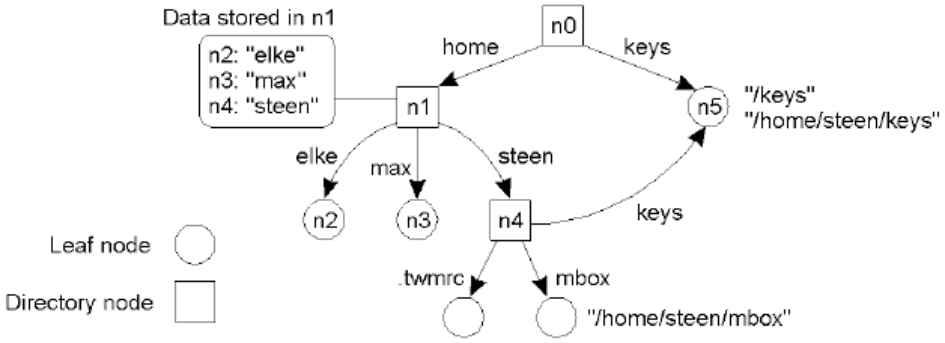
- Types of names
  - Address: an access point of an entity
  - Identifiers: a name that uniquely identifies an entity
    - An identifier refers to at most one entity
    - Each entity is referred to by at most one identifier
    - An identifier always refers to the same entity
  - Human-friendly names
  - Location-independent name: a name that is independent from its addresses

Name Spaces and Name Resolution

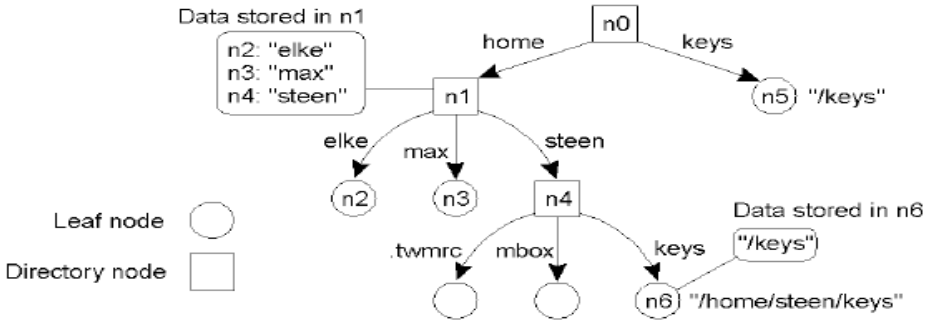
- Names are organized into name spaces
- A name space can be represented as a labeled, directed graph with two types of nodes
  - Leaf nodes and directory nodes
  - Absolute vs relative path names
  - Local names vs global names
- Name Resolution: the process of looking up a name
  - Closure mechanism: knowing where and how to start name resolution

Name Spaces cont'd

A general naming graph with a single root node.

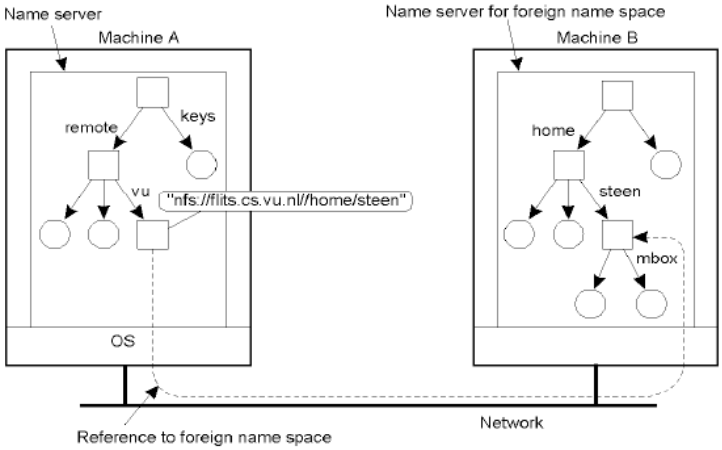


Linking and Mounting



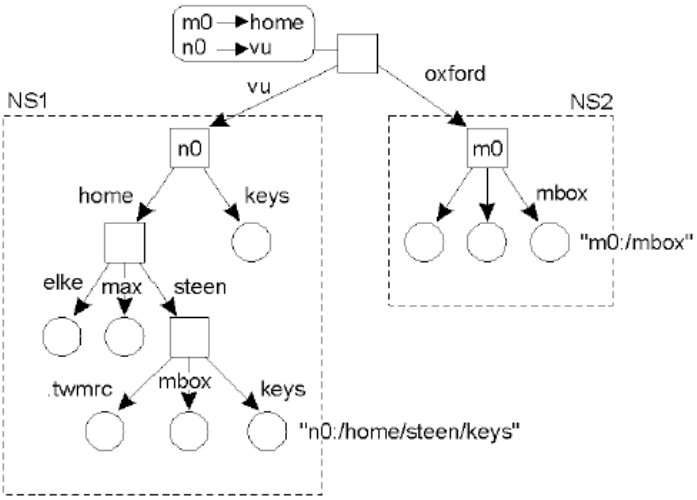
The concept of a symbolic link explained in a naming graph.

Linking and Mounting



Mounting remote name spaces through a specific process protocol.

Merging Name Spaces

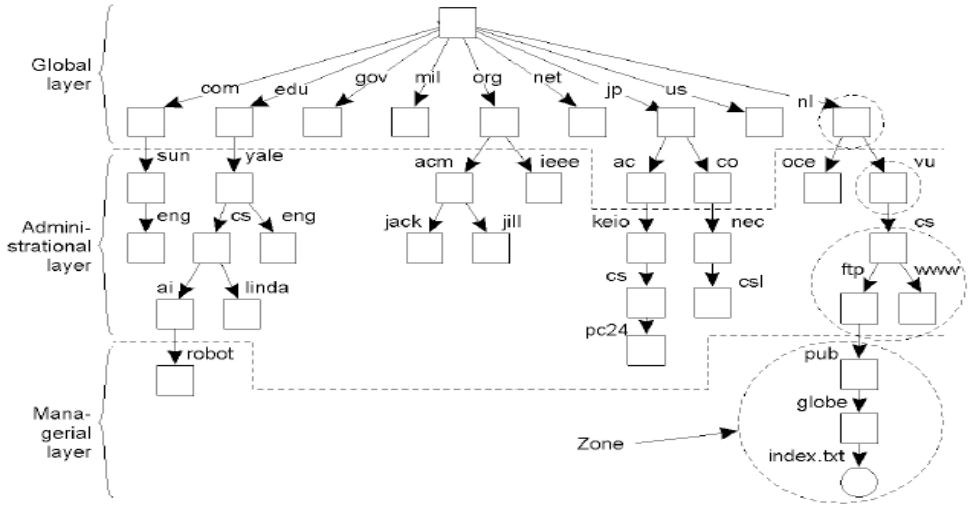


Organization of the DEC Global Name Service

Implementing Name Spaces

- Naming service: a service that allows users and processes to add, remove, and lookup names
- Name spaces for large-scale widely distributed systems are typically organized hierarchically
- Three layers used to implement such distributed name spaces
  - Global layer: root node and its children
  - Administrational layer: directory nodes within a single organization
  - Managerial layer

Name Space Distribution



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

Name Space Distribution

Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

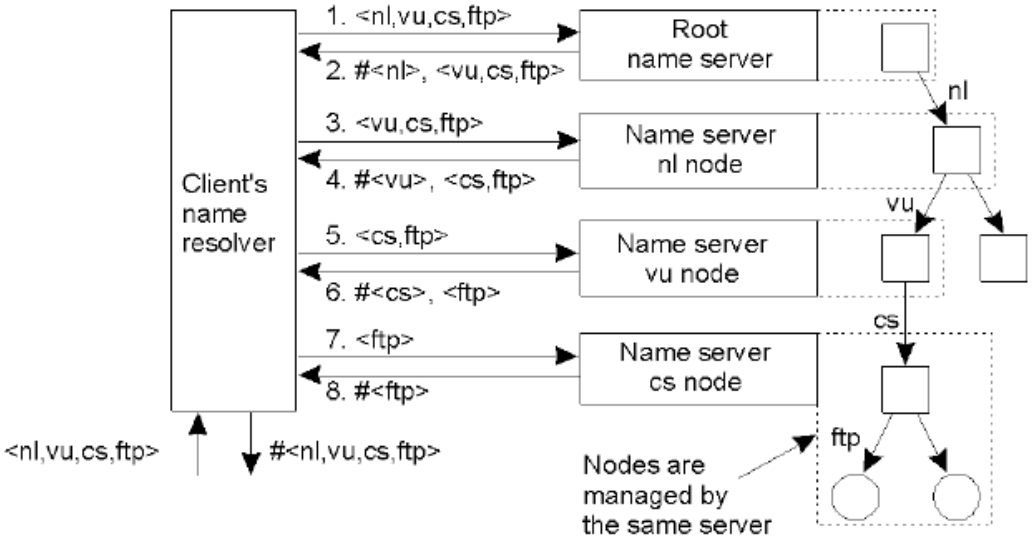
A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, as an administrational layer, and a managerial layer.

Implementation of Name Resolution

- Iterative vs recursive name resolution
- Recursive name resolution puts a higher performance demand on each name server
  - Too high for global layer name servers
- Advantages of recursive name resolution
  - Caching is more effective
  - Communication costs may be reduced

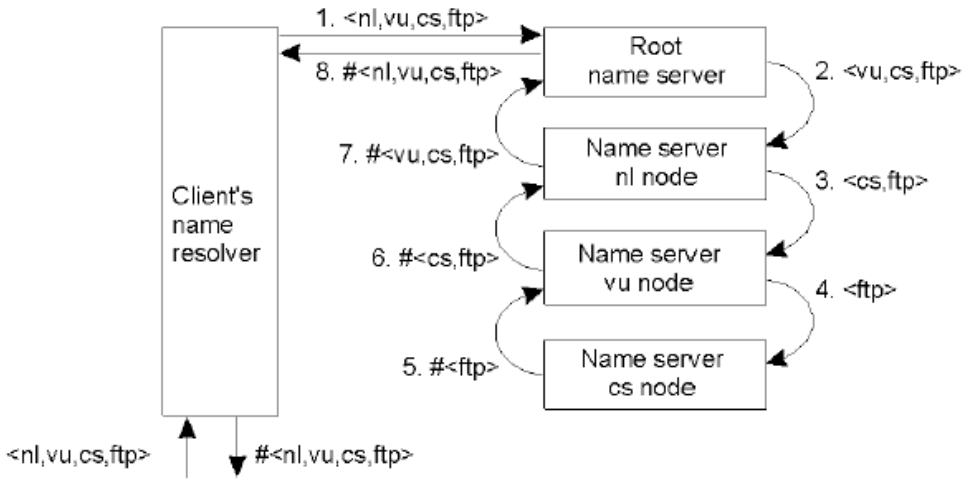
Implementation of Name Resolution

The principle of iterative name resolution.



Implementation of Name Resolution

The principle of recursive name resolution.

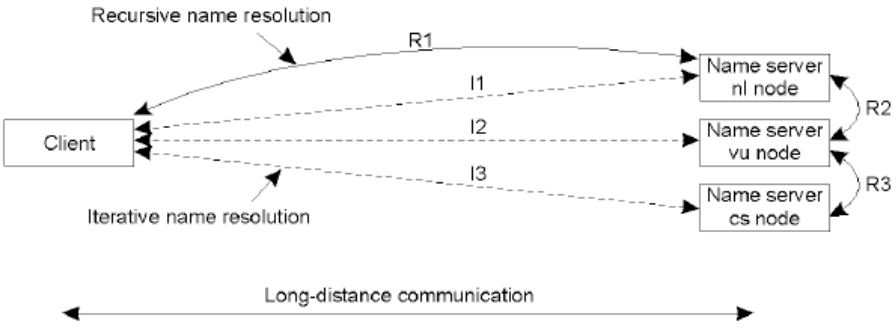


Implementation of Name Resolution

Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	<ftp>	#<ftp>	–	–	#<ftp>
vu	<cs,ftp>	#<cs>	<ftp>	#<ftp>	#<cs> #<cs, ftp>
ni	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
root	<ni,vu,cs,ftp>	#<nl>	<vu,cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>	#<nl> #<nl,vu> #<nl,vu,cs> #<nl,vu,cs,ftp>

Recursive name resolution of *<nl, vu, cs, ftp>*. Name servers cache intermediate results for subsequent lookups.

Implementation of Name Resolution



The comparison between recursive and iterative name resolution with respect to communication costs.

Example System: DNS

- Domain Name System (DNS)
  - Host name to IP address translation
  - Name space organized as a hierarchical rooted tree
    - Name space divided into non-overlapping **zones**
  - Name servers implement the global and administrational layers
    - Managerial layer not part of DNS
    - Each zone has a name server, which is typically replicated
    - Updates take place at the primary name server for a zone
      - Secondary name servers request the primary name server to transfer its content

The DNS Name Space

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

The most important types of resource records forming the contents of nodes in the DNS name space.

DNS Implementation

An excerpt from the DNS database for the zone *cs.vu.nl*.

Name	Record type	Record value
cs.vu.nl	SOA	star (1999121502,7200,3600,2419200,86400)
cs.vu.nl	NS	star.cs.vu.nl
cs.vu.nl	NS	top.cs.vu.nl
cs.vu.nl	NS	solo.cs.vu.nl
cs.vu.nl	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl	MX	1 zephyr.cs.vu.nl
cs.vu.nl	MX	2 tornado.cs.vu.nl
cs.vu.nl	MX	3 star.cs.vu.nl
star.cs.vu.nl	HINFO	Sun Unix
star.cs.vu.nl	MX	1 star.cs.vu.nl
star.cs.vu.nl	MX	10 zephyr.cs.vu.nl
star.cs.vu.nl	A	130.37.24.6
star.cs.vu.nl	A	192.31.231.42
zephyr.cs.vu.nl	HINFO	Sun Unix
zephyr.cs.vu.nl	MX	1 zephyr.cs.vu.nl
zephyr.cs.vu.nl	MX	2 tornado.cs.vu.nl
zephyr.cs.vu.nl	A	192.31.231.66
www.cs.vu.nl	CNAME	soling.cs.vu.nl
ftp.cs.vu.nl	CNAME	soling.cs.vu.nl
soling.cs.vu.nl	HINFO	Sun Unix
soling.cs.vu.nl	MX	1 soling.cs.vu.nl
soling.cs.vu.nl	MX	10 zephyr.cs.vu.nl
soling.cs.vu.nl	A	130.37.24.11
laser.cs.vu.nl	HINFO	PC MS-DOS
laser.cs.vu.nl	A	130.37.30.32
vucs-das.cs.vu.nl	PTR	0.26.37.130.in-addr.arpa
vucs-das.cs.vu.nl	A	130.37.26.0

DNS Implementation

Name	Record type	Record value
cs.vu.nl	NS	solo.cs.vu.nl
solo.cs.vu.nl	A	130.37.21.1

Part of the description for the *vu.nl* domain which contains the *cs.vu.nl* domain.

Example System: X.500

- An example of a directory service
  - Analogy: X.500 is to DNS as the yellow pages are to a telephone book
- Each directory entry is made up of a collection of (attribute, value) pairs
  - Attributes can be single-valued or multiple-valued
- Collection of all directory entries is called a Directory Information Base (DIB)
- Each entry has a globally unique name formed by a sequence of naming attributes (Relative Distinguished Names or RDN)
- Lookup operations
  - Read: Read a single record given its pathname in the Directory Information Tree (DIT), I.e. hierarchical name space formed by directory entries
  - List: return the names of all outgoing edges of a given node in the DIT

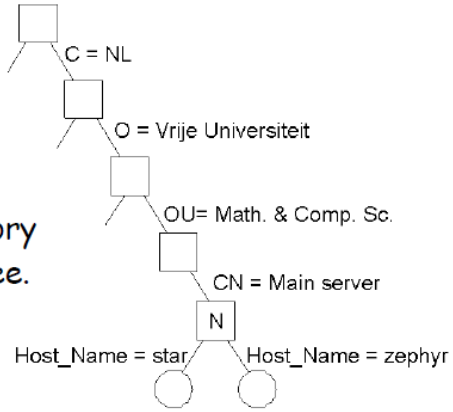
The X.500 Name Space

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	L	Vrije Universiteit
OrganizationalUnit	OU	Math. & Comp. Sc.
CommonName	CN	Main server
Mail_Servers	--	130.37.24.6, 192.31.231,192.31.231.66
FTP_Server	--	130.37.21.11
WWW_Server	--	130.37.21.11

A simple example of a X.500 directory entry using X.500 naming conventions.

The X.500 Name Space

Part of the directory information tree.



The X.500 Name Space

Two directory entries having *Host_Name* as RDN.

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Math. & Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Math. & Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	192.31.231.66

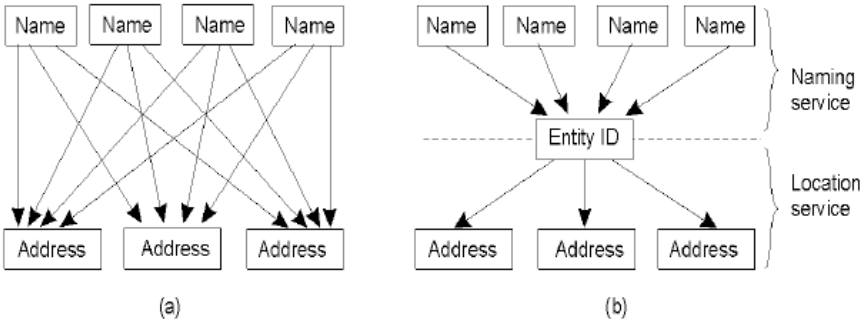
X.500 implementation

- Similar to DNS
  - The DIT is partitioned and distributed across several servers known as Directory Service Agents (DSA)
  - Clients are represented by name resolvers called Directory User Agents (DUA)
- Differences from DNS
  - Operations for searching through a DIB given a set of criteria that attributes should meet
  - Searching is an expensive operation since several leaf nodes of a DIT will need to be accessed
- Lightweight Directory Access Protocol (LDAP) is an application-level protocol that is a simplified version of X.500
  - Becoming a de facto standard for Internet-based directory services

Locating Mobile Entities

- Consider an entity that changes its location
  - E.g. ftp.cs.vu.nl moves to another domain
    - Cannot change name
  - Two choices
    - Record the address of the new machine in the DNS database for cs.vu.nl
      - If name changes again, DNS entry will have to be changed again
    - Record the name of the new machine in the database, I.e. use a symbolic link
      - Inefficient lookups
- Traditional naming services such as DNS cannot cope well with mobile entities
  - Problems arise because of the direct mapping between human-friendly names and the address of entities

Naming versus Locating Entities

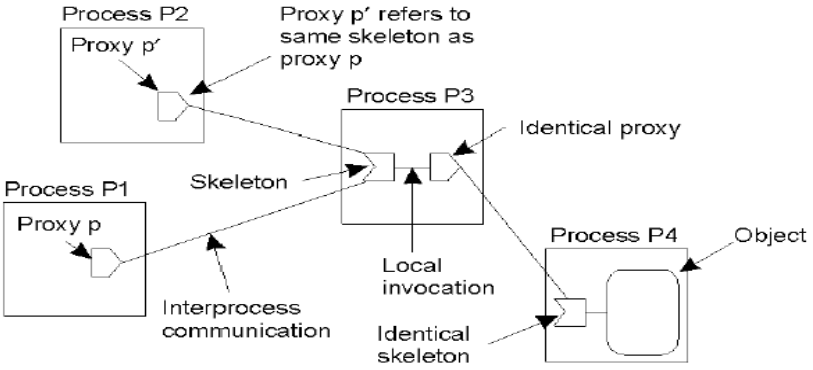


a) Direct, single level mapping between names and addresses.
b) Two-level mapping using identities.

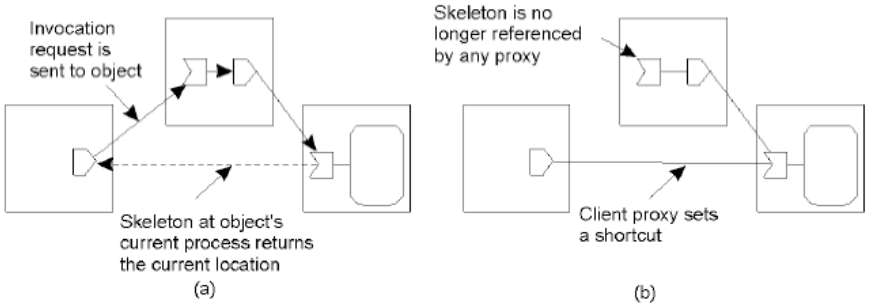
Locating Entities

- Simple Solutions that work in a LAN environment
  - Broadcasting & Multicasting
    - Message containing identifier of the entity is broadcast; machine with an access point for the entity replies with the address of the access point
      - ARP protocol for finding the data-link address of a machine given the IP address
  - Forwarding pointers
    - When an entity moves from A to B, it leaves behind a reference to its new location at B
- Home-based Approaches
  - Home agent keeps track of current location of mobile entity
- Hierarchical Approaches

Forwarding Pointers

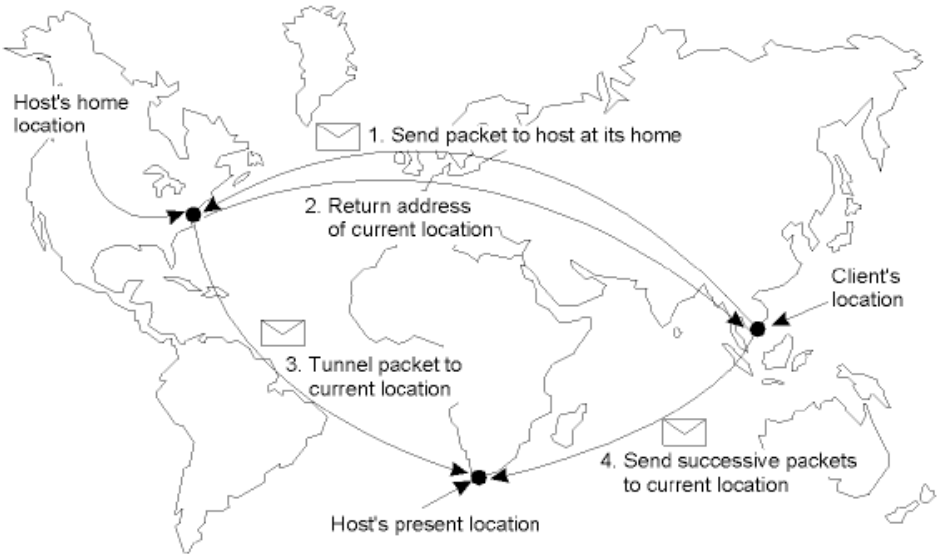


Forwarding Pointers

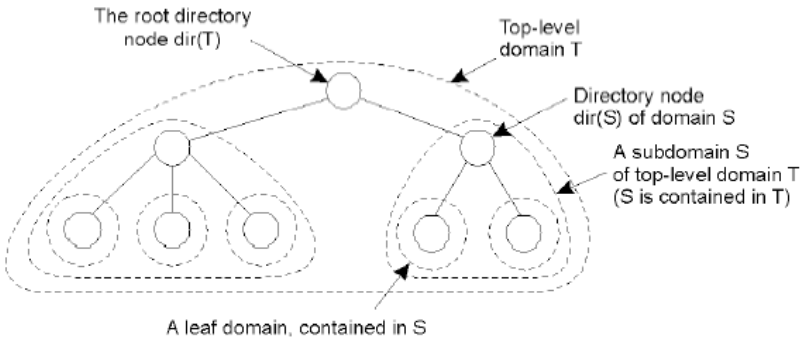


Redirecting a forwarding pointer, by storing a shortcut in a proxy.

Home-Based Approaches

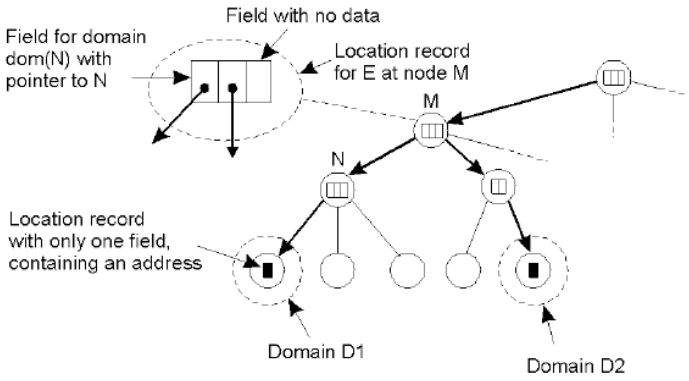


Hierarchical Approaches



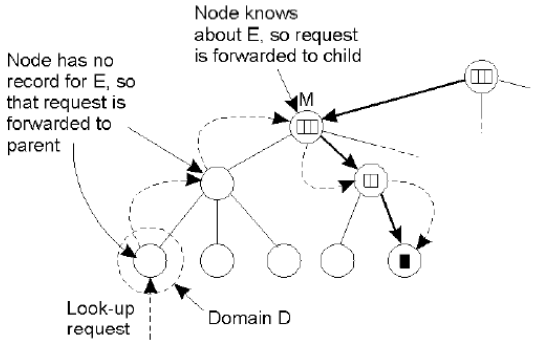
Hierarchical organization of a location service into domains, each having an associated directory node.

Hierarchical Approaches



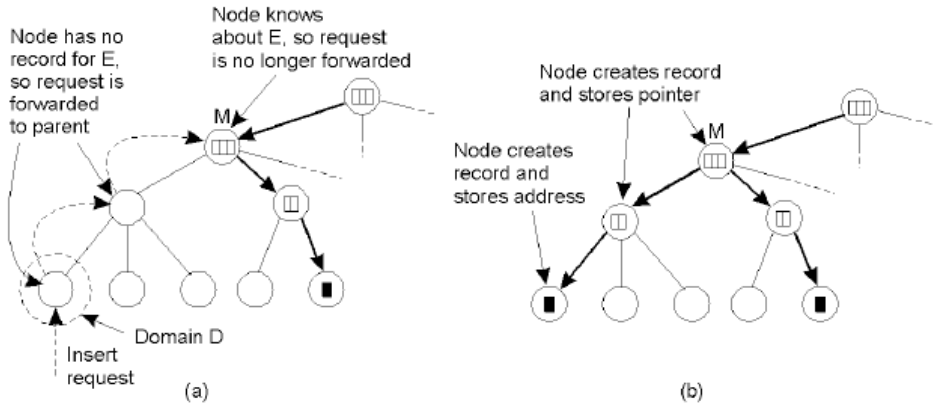
An example of storing information of an entity having two addresses in different leaf domains.

Hierarchical Approaches



Looking up a location in a hierarchically organized location service.

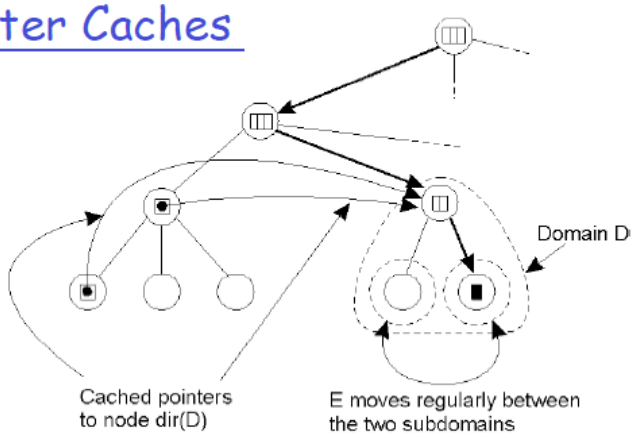
Hierarchical Approaches



a) An insert request is forwarded to the first node that knows about entity *E*.

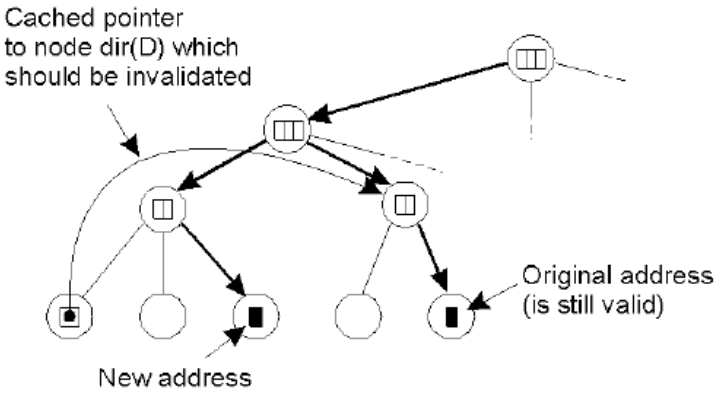
b) A chain of forwarding pointers to the leaf node is created.

Pointer Caches



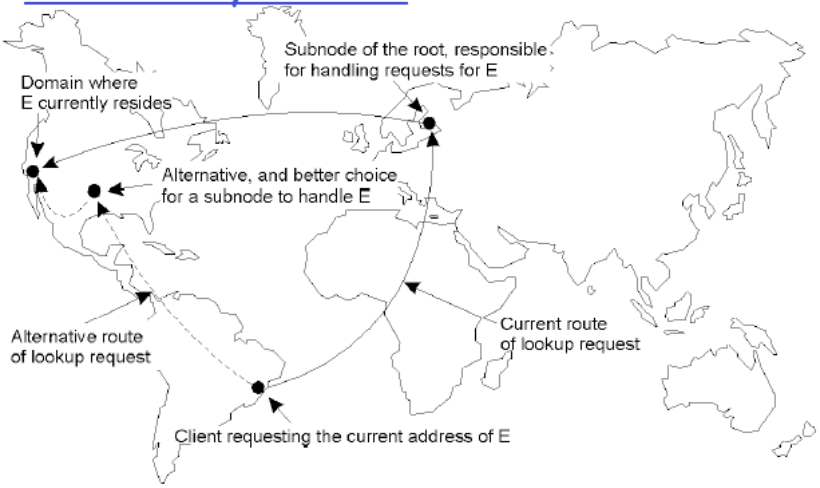
Caching a reference to a directory node of the lowest-level domain in which an entity will reside most of the time.

Pointer Caches



A cache entry that needs to be invalidated because it returns a nonlocal address, while such an address is available.

Scalability Issues



The scalability issues related to uniformly placing subnodes of a partitioned root node across the network covered by a location service.