CIS 505 – Software Systems
2/7/12

Matt Blaze

(some slides from Boon Loo, Insup Lee., Silberschatz et.al., and Colouris et. al.

RPC issues…

- Transparency:
  - Making RPC “invisible” to the programmer
  - Support heterogeneity (OS, arch., language)
- Parameters:
  - Semantics of parameter passing
  - Call-by-value vs call-by-reference
- Binding:
  - Locate remote process, bind to it during execution
- Dealing with failures

Word of the day:
Idempotent

- What failure behavior do you need? Does it matter?
- Some functions are **idempotent.**
  - It doesn't matter how many times you execute them on the server; the answer is always the same.
  - Addition of two numbers, etc.
- But some aren't…
  - transferring money between bank accounts

RPC Semantics in the Presence of Failures

- Different classes of failures
  - The client is unable to locate the server
  - The request (reply) messages from the client to server (or vice versa) is lost
  - The server crashes after receiving a request
  - The client crashes after sending a request

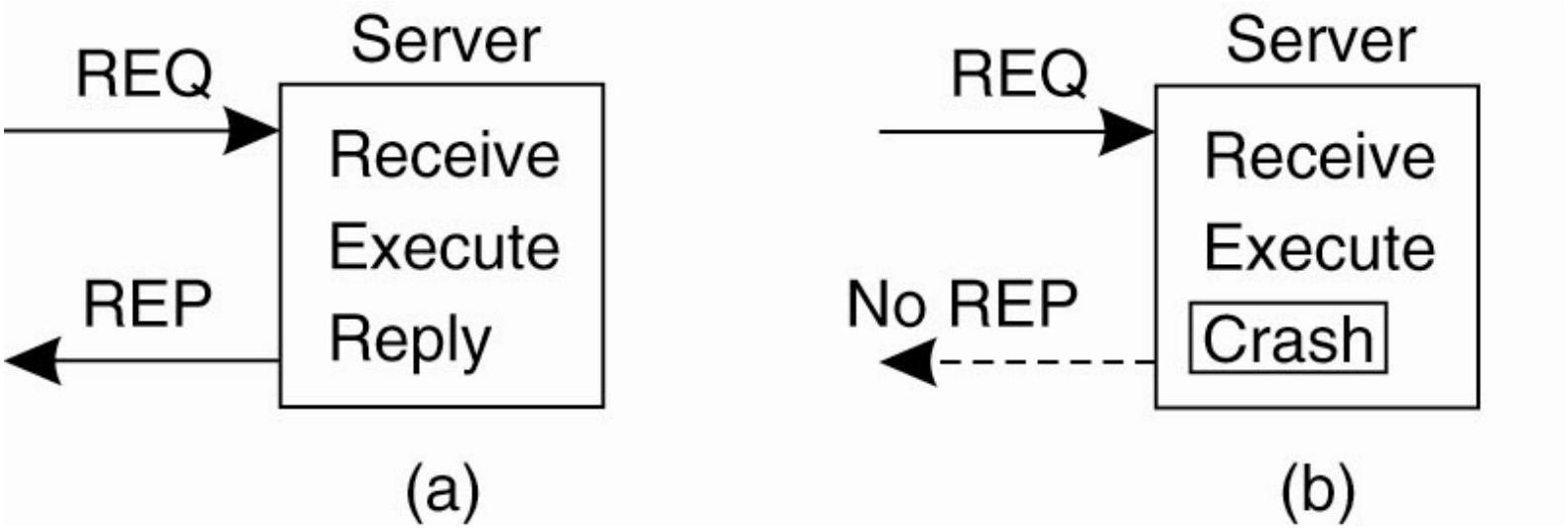
#1: Client might be unable to locate the server

- Causes: server down, different version of server binary, new interfaces, new stubs
- Fixes: write code to handle exceptions
  - Not very transparent, though (not similar to a local procedure)

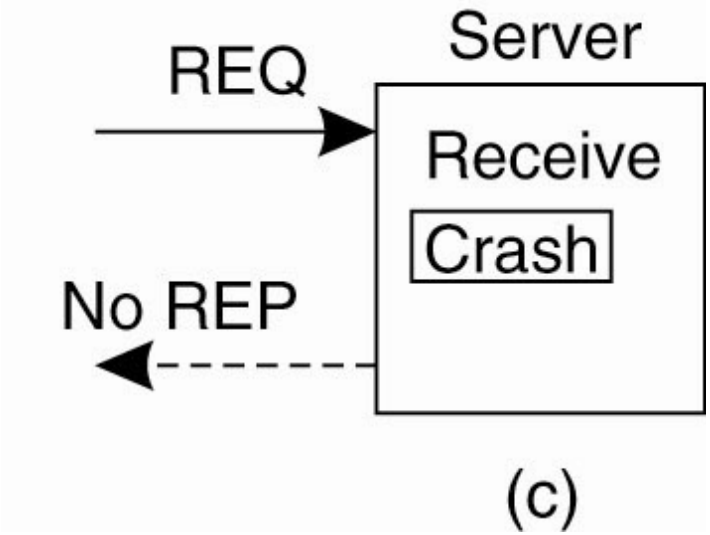
#2 – Messages might be lost (either Request or Reply)

- Complications with lost messages: client doesn't know what happened at server
  - Did server execute the procedure or not?
  - Did request/reply get lost, or is server/network merely slow?
- Solution: client retransmission upon timeout
  - Client stub or OS starts a timer when sending request
  - If server didn't receive previous request, it will not be able to differentiate original and retransmission
  - Only works if operation is idempotent: it's fine to execute it twice
  - What if operation not idempotent?
    - E.g. transferring money from one account to another
    - Solution: assign unique sequence numbers to every request
- If multiple message are lost then
  - "client is unable to locate server" error

#3 – Server might crash



- A server in client-server communication.
  (a) The normal case.
  (b) Crash after execution.
  (c) Crash before execution.



Dealing with Server Crashes

- What should the client do when timer expires?
  - Retransmit or not?
  - Worry about duplicate requests?
- Three possible semantics
  - At least once semantics
    - Client keeps trying until it gets a reply
  - At most once semantics (e.g. Sun RPC)
    - Client gives up on failure
    - Most practical, but RPC may be carried out zero times!
  - Ideal: Exactly once semantics (e.g. Java RMI)
    - Impossible in practice
    - Remote print job example

#4 – Clients might crash

- Let the server computations become “orphans”
- Orphans can
  - Waste CPU cycles
  - Lock files
  - Client reboots and it gets the old reply immediately

#4 - Client Crashes: Possible Solutions

- Extermination:
  - Client keeps a log, reads it when reboots, and kills the orphan
  - Disadvantage: high overhead to maintain the log
- Reincarnation:
  - Divide times in epochs
  - Client broadcasts epoch when reboots
  - Upon hearing a new epoch, servers kills the orphans from previous epochs
  - Disadvantage: doesn't solve problem when network partitioned
- Expiration:
  - Each RPC is given a lease T to finish computation
  - If it does not, it needs to ask for another lease
  - If client reboots after T sec, all orphans are gone

Next....

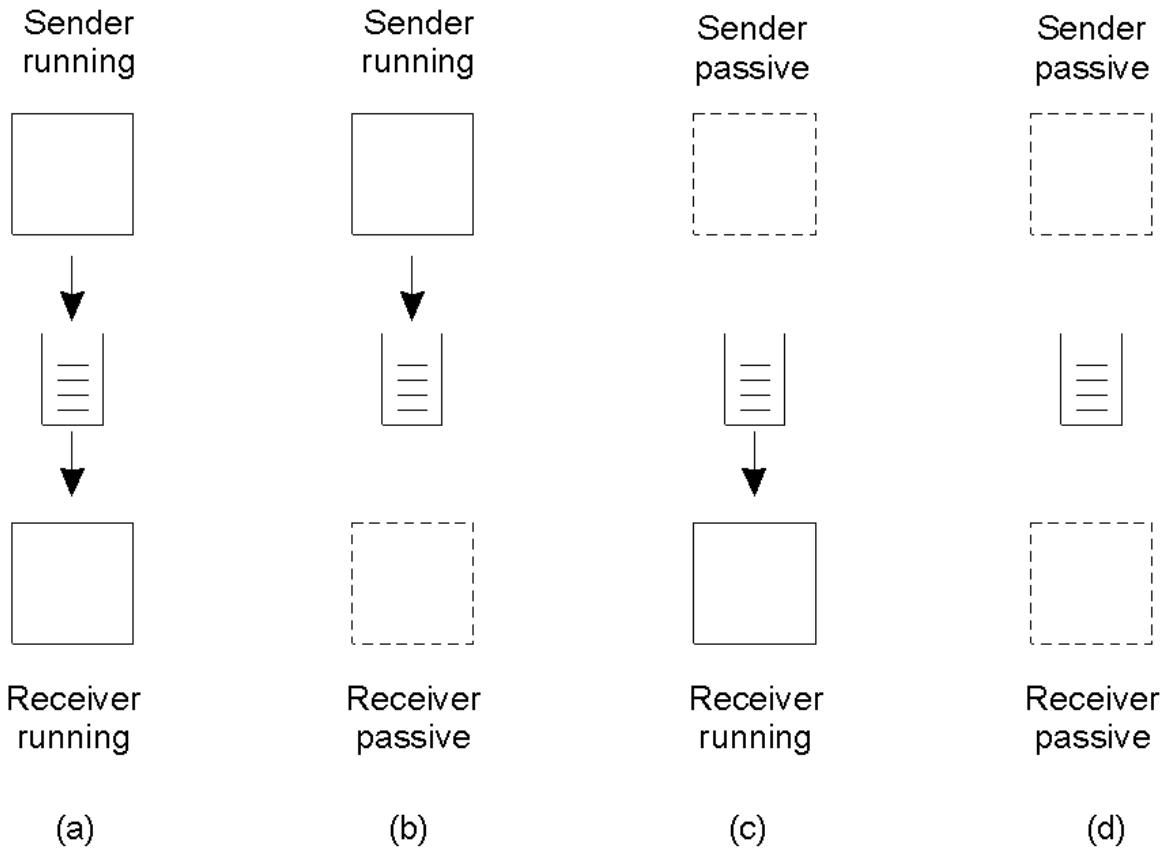
- Remote Procedure Calls (RPC)
- Message queues
- Naming

What's a message queueing system?

- So far, we have seen sockets (send/recv of messages, and RPCs)
- In a message queueing system
  - Senders insert messages to specific queues.
  - Receivers subscribe to queues to receive messages ("publish/subscribe")
  - **Loose-coupling:** receiver need not be around when sender receives message. Receiver may not even bother retrieving message!
  - **Distribution:** sender and receiver can be on different machines
  - **Many-to-one:** there can be multiple receivers per sender, or multiple senders per receiver.
- Textbook refers to this as **asynchronous persistent** communication.
- **Practical examples:** JMS (Java messaging service), IBM MQSeries (now Websphere messaging), pub/sub systems

Message-Queuing Model

- Four combinations for loosely-coupled communications using queues.



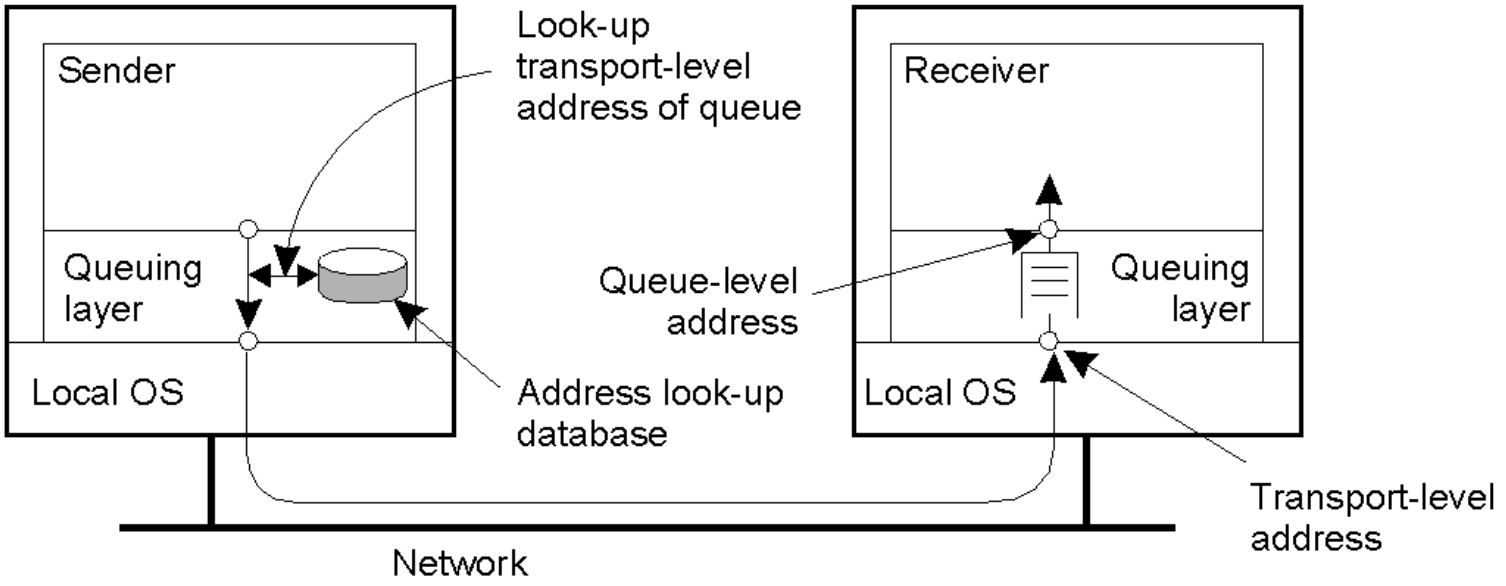
Message-Queuing Model

- Queues correspond to buffers at communication servers.
- Basic interface to a queue in a message-queuing system.

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.

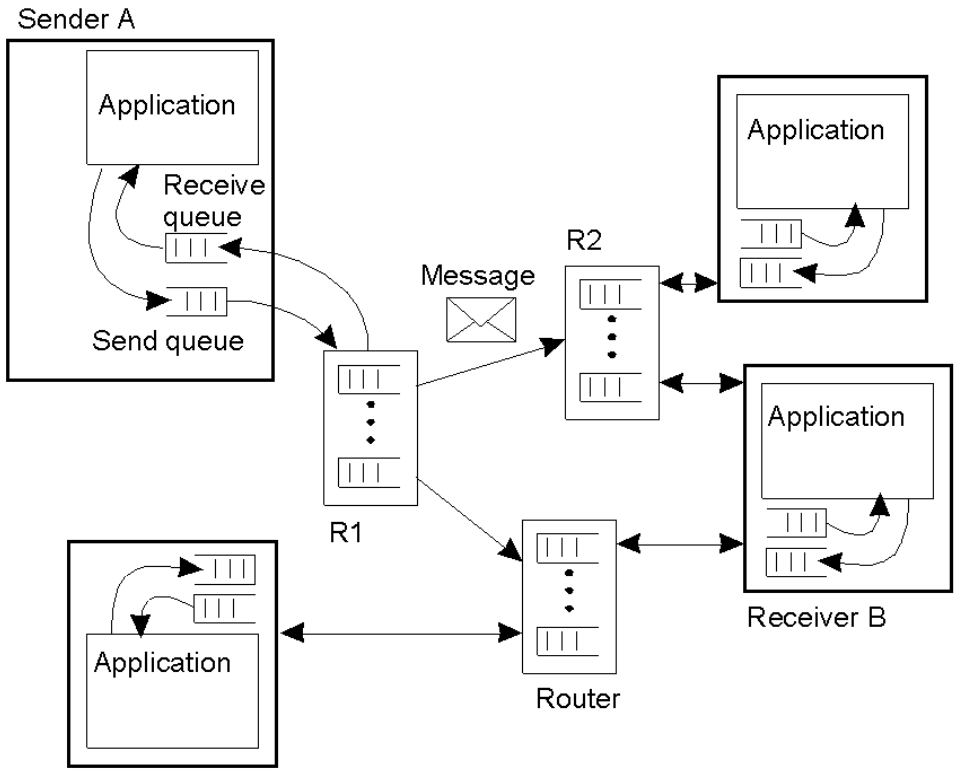
General Architecture of a Message-Queuing System

- The relationship between queue-level addressing and network-level addressing.
- A queueing layer middleware maintains mapping from queue names to network location.
- Delivers message to remote queues (in practice, there can be relays in between)



Architecture of a Message-Queuing System with Routers

- Network of queue managers => content-based routing
- Maintain routes => similar to running routing protocols
- Scalability, ease of management (why? what advantages?)

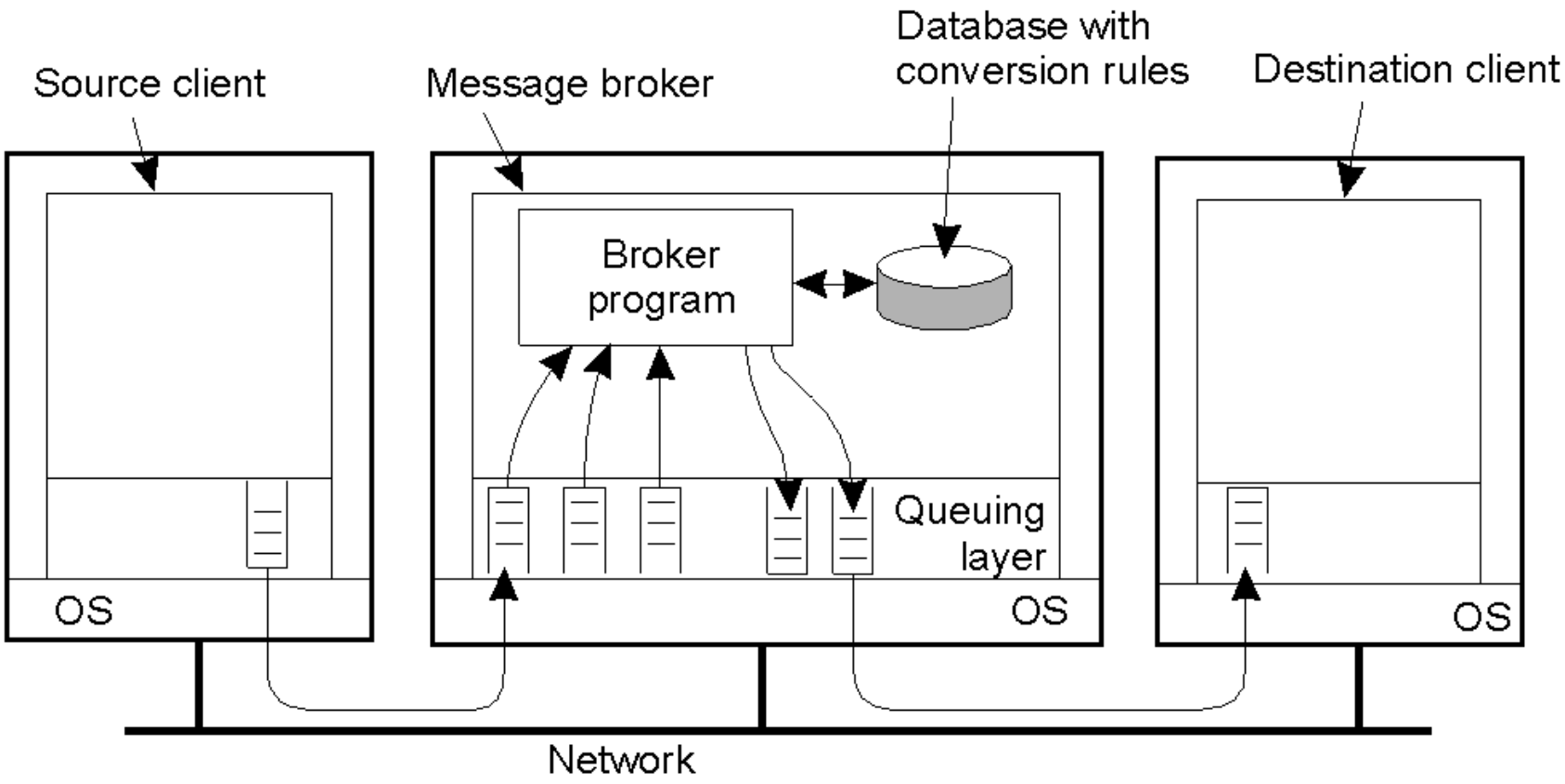


Message Broker

- Message queuing systems assume common messaging protocol:
  - All applications agree on message format (i.e., structure and data representation)
- Message broker:
  - Centralized component that takes care of application heterogeneity in an MQ system:
    - Transforms incoming messages to target format
  - May provide subject-based routing capabilities
    - Publish/subscribe systems

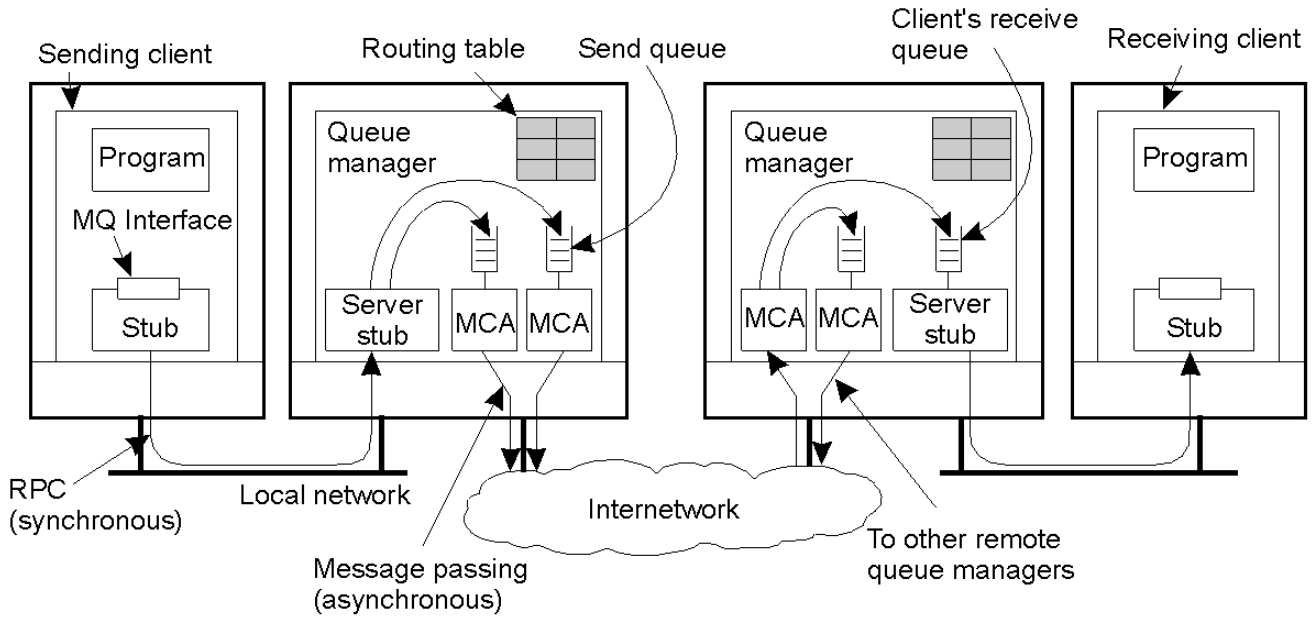
Message Brokers

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



Example: IBM WebSphere MQ

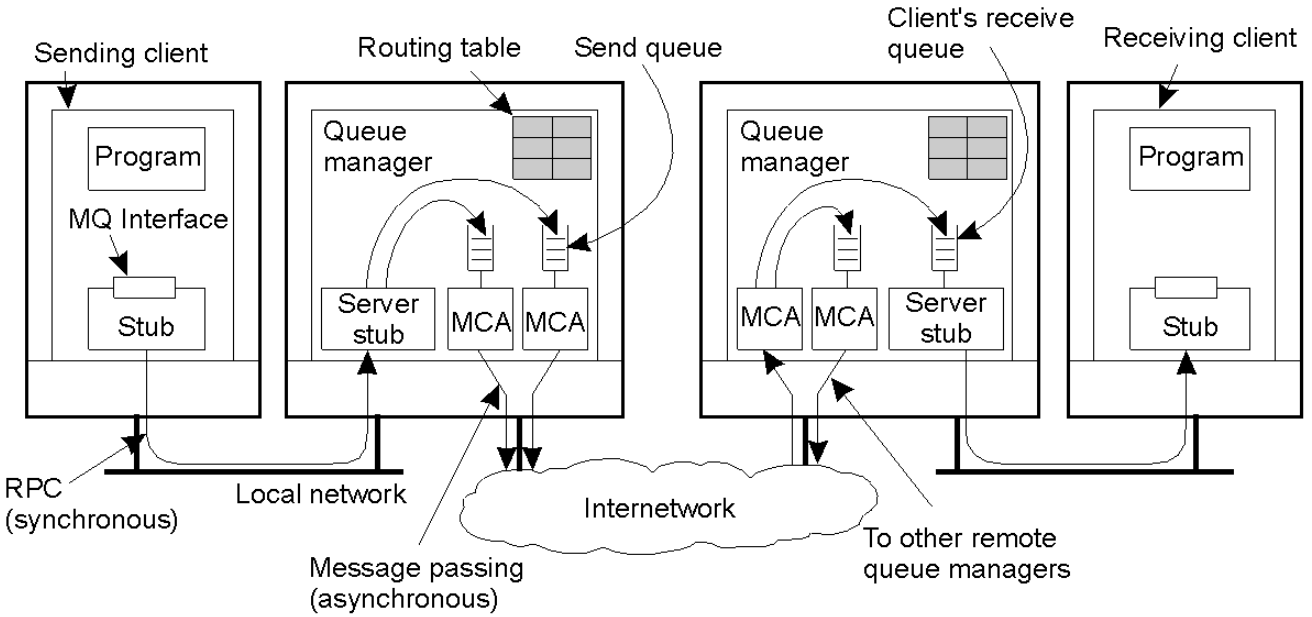
Formerly known as “IBM MQSeries”, part of the whole WebSphere application server middleware suite



Basic concept:

- Application-specific messages are put into, and removed from queues
- Queues always reside under the regime of a queue manager
- Processes can put messages only in local queues, or through an RPC mechanism

Example: IBM WebSphere MQ



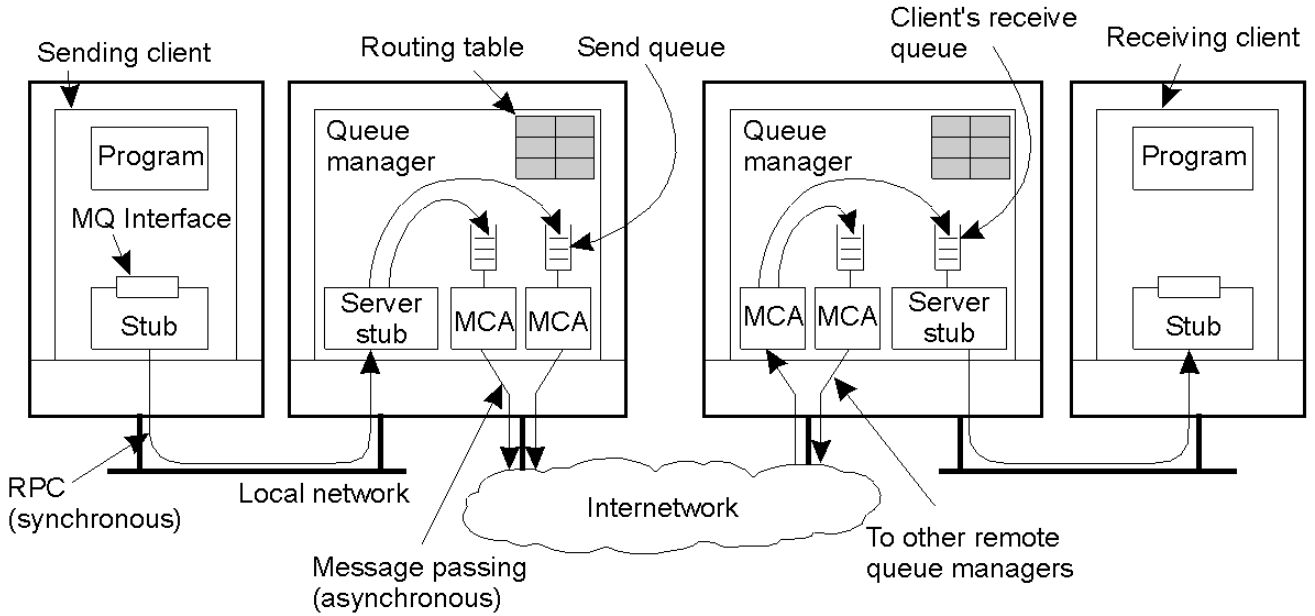
- Channels:
  - Unidirectional transfer of messages between queues
  - At each endpoint of channel is a message channel agent (MCA)
  - MCA are responsible for:
    - Sending/receiving packets
    - Setting up channels using lower-level network communication facilities (e.g., TCP/IP)
    - (Un) wrapping messages from/in transport-level packets

Message Channel Agents (MCA)

- MCA started via manual configuration (or triggered by first message)
- Start one MCA, send control message to start the other
- Sender and Receiver MCA negotiate attributes:

Attribute	**Description**
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Specifies maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue

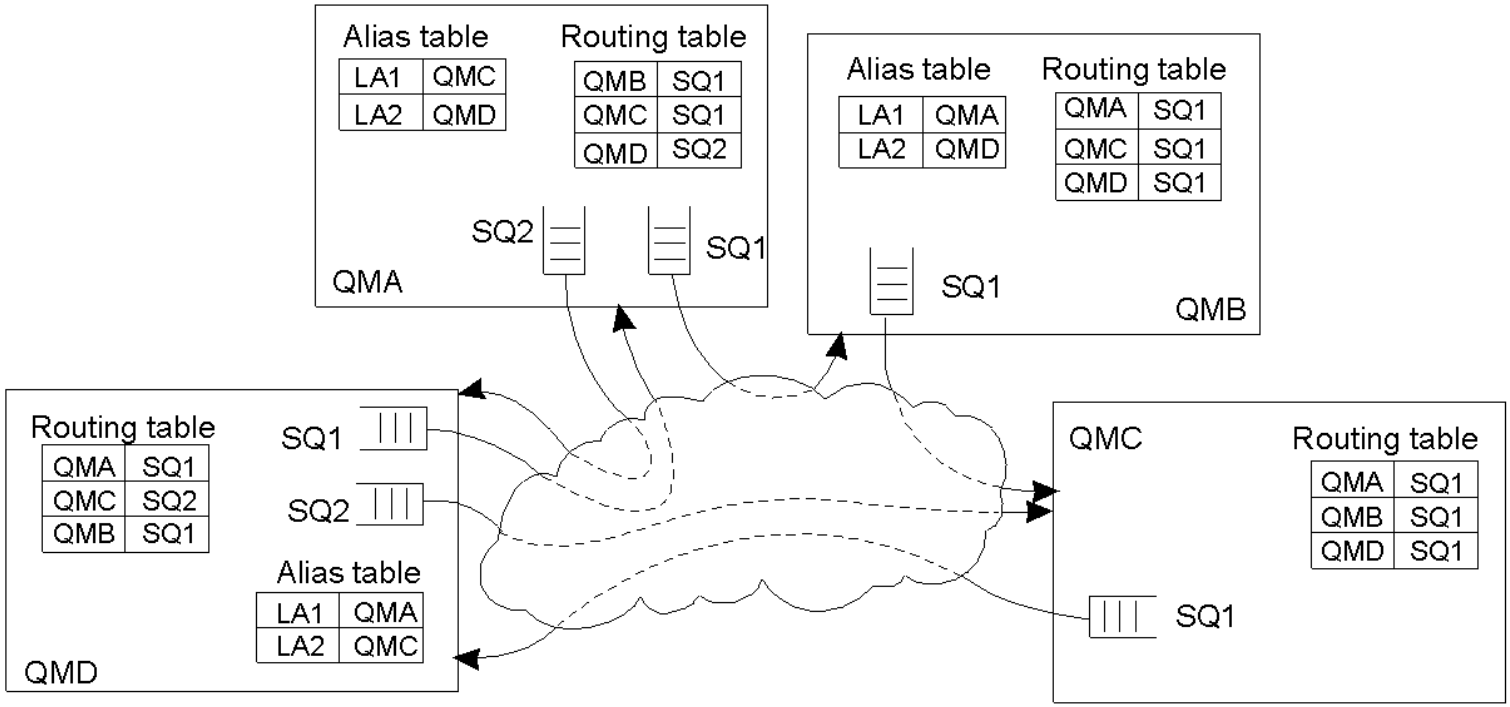
Message Transfer API



Primitive	**Description**
MQopen	Open a (possibly remote) queue
MQclose	Close a queue
MQput	Put a message into an opened queue
MQget	Get a message from a (local) queue

MQSeries Queuing Network

- Address: (Destination Queue Manager, Remote Receive Queue)
- Routing entry (Destination Queue Manager, local Send Queue)
- Local alias: avoid changing applications when queue manager changes



Communication in a distributed system

- Layering
- Sockets (Unreliable datagrams, reliable in-order)
- Remote procedure call / Remote object invocation
- Distributed message queues

Next few lectures

- Naming in distributed systems (Chapter 5)
- Synchronization
  - Logical clocks (Chapter 6.1-6.2)
  - Mutual exclusion and elections (Chapter 6.3, 6.5)

Rest of Today

- Naming (general concepts)
- Examples:
  - DNS
  - LDAP

Names, Entities and Identities

- Names are use to share resources, uniquely identify entities, to refer to locations, etc.
- Entity:
  - Hosts, printers, disks, and files.
- Name:
  - Unique identifier for entity
  - Human-friendly name: string of characters

Names & Addresses in the Network

- Link layer:
  - MAC (Ethernet) addresses
  - 48 bits, assigned by device maker
  - used only on local network
- Network layer
  - IPv4 address
  - 32 bits, based on *position* in network
- Domain names
  - human readable, used by applications
  - “www.cis.upenn.edu”
  - used by applications, translated to IP addresses

Some Terminology…

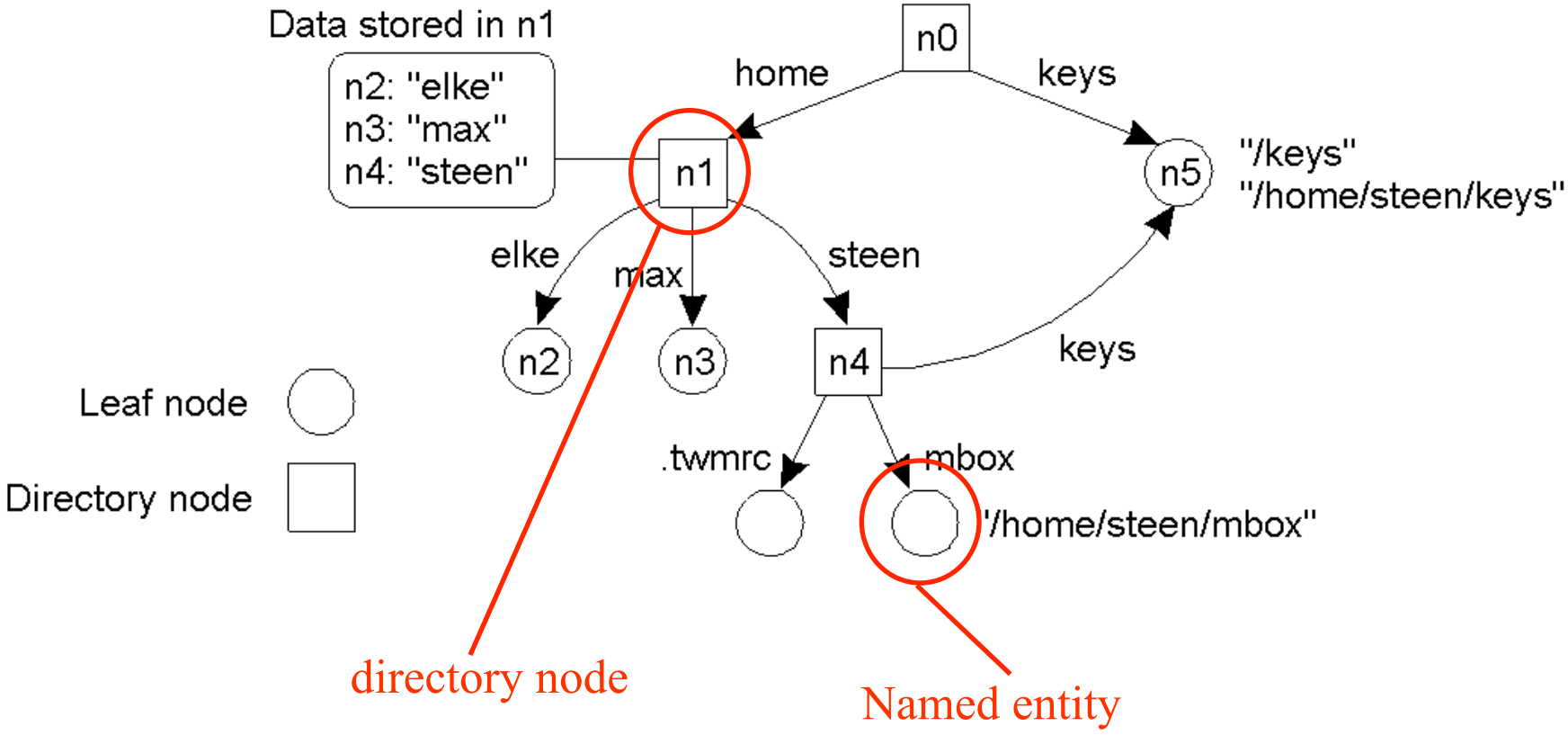
- Name space: define the set of possible names and their relationship
  - Hierarchical (e.g., Unix and Windows file names)
  - Flat (MAC address, distributed hash table)
- Bindings:
  - The mapping between names and values (e.g., addresses or other names)
  - Bindings can be implemented by using tables
- Resolution: procedure that, when invoked with a name, returns the corresponding value
- Name service: specific implementation of a resolution mechanism that is available on the network and that can be queried by sending messages

Structured Naming: Name Spaces

- Name space – organization of names
  - Represented as a labeled, directed graph with two types of nodes:
    - Leaf node – a named entity (stores address or state, e.g. file)
    - Directory node – a directory table of (edge label, node id)
  - A naming path – *"N: <label-1,label-2, …,label-n>"*
  - Path name
    - Absolute path name: *label-1* is the root node
    - Relative path name
- Example use: file naming and host naming on the Internet

Example: Hierarchical Name Spaces

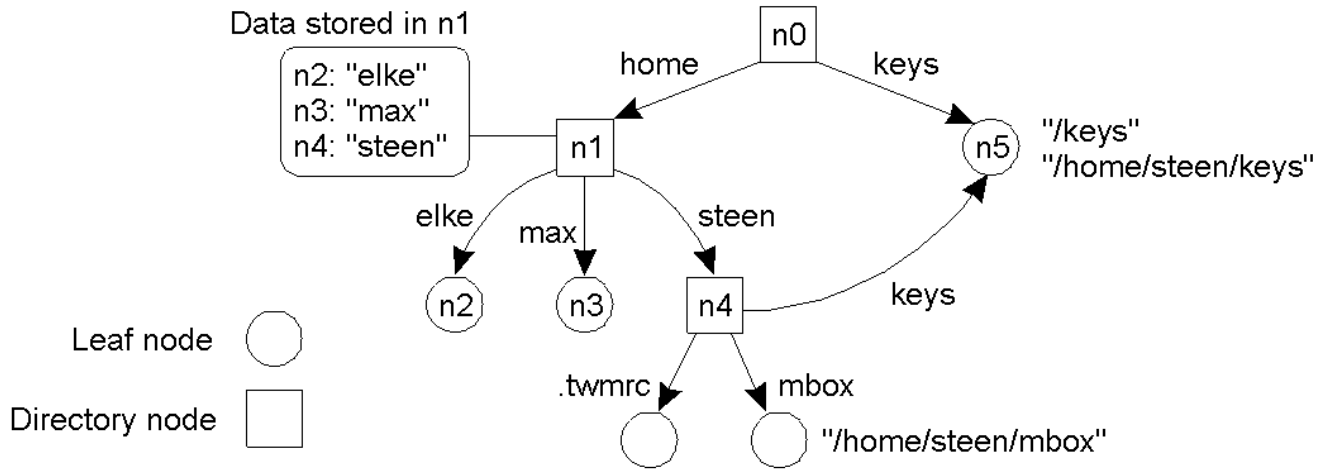
- A graph in which a **leaf node** represents a (named) entity. A **directory node** is an entity that refers to other nodes.



- A directory node contains a (directory) table of *(edge label, node identifier)* pairs. Path is “n0: <home, streen, mbox>” or “/home/steen/mbox”

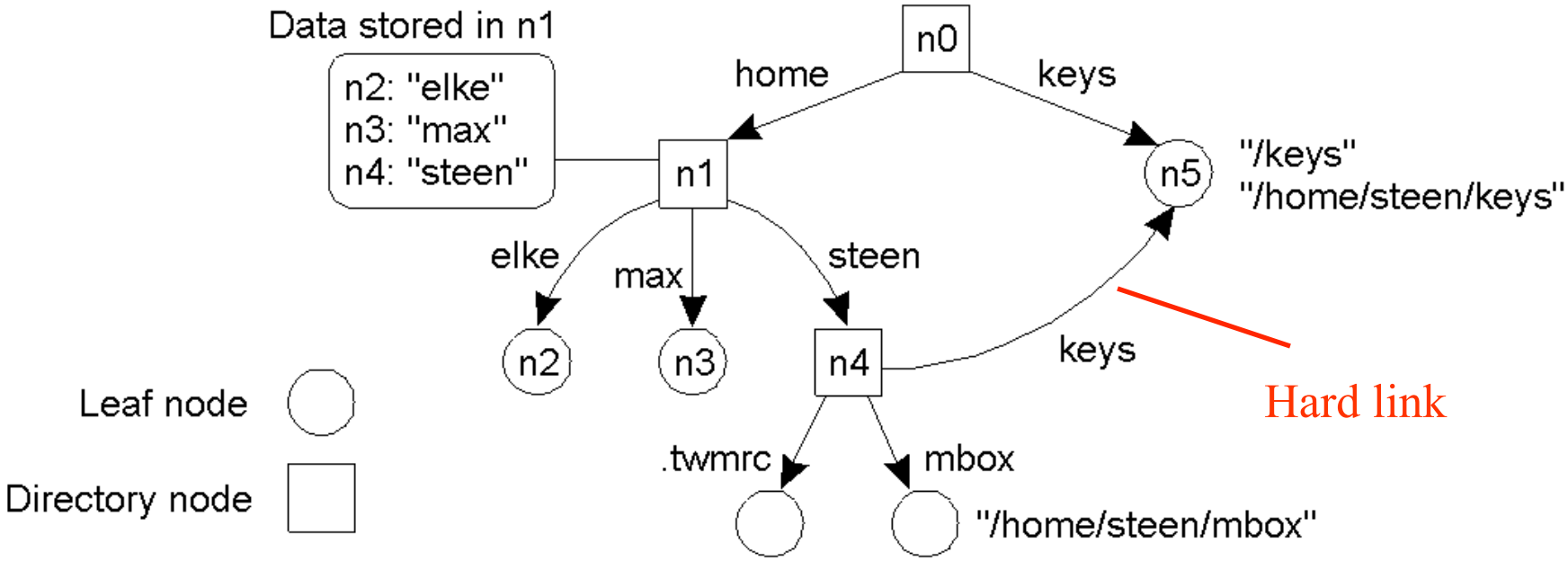
Name Resolution

- Closure mechanism
  - Where to start name resolution?
    - Root, HOME
- Link
  - Aliases
    - Node n5 can be referred two different path names
    - /keys and /home/steen/keys
  - Hard links vs symbolic links



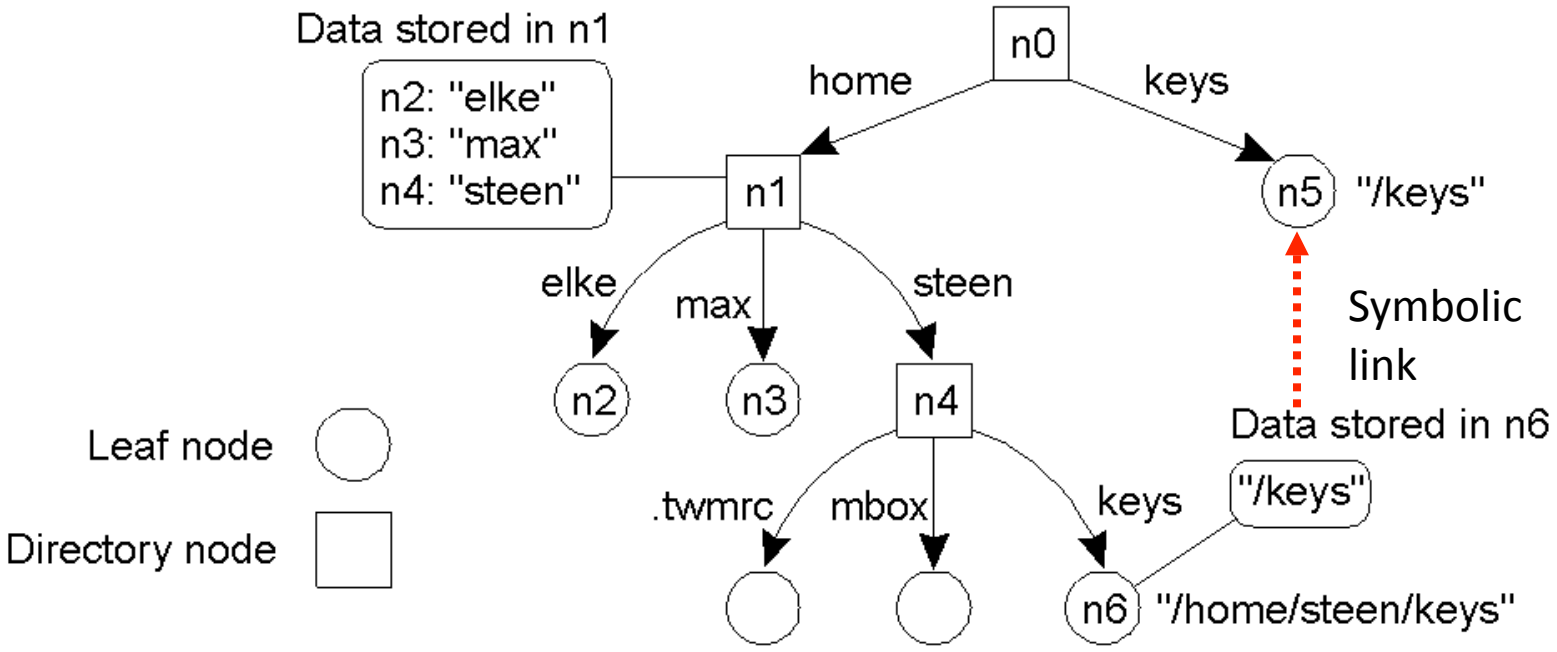
Hard links

- Both /keys and /home/steen/keys are *hard links* to node n5



Linking and Mounting

- Symbolic link in a naming graph
  - Instead of storing address or state of entity, stores absolute path name

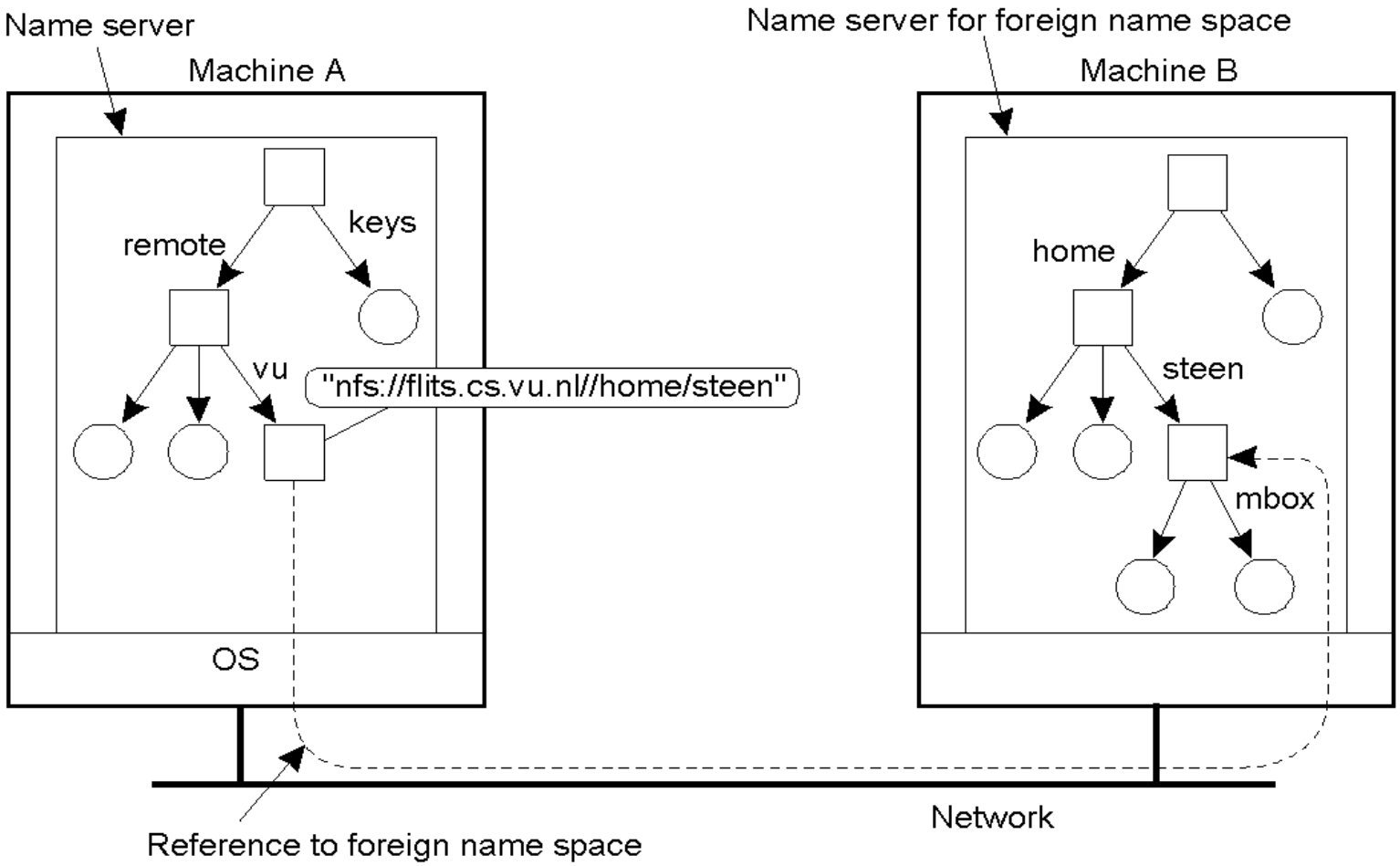


Mounting name space

- Mount a foreign name space NS2 into local NS1
- Requires:
  - The name of an access protocol
  - The name of the server
  - The name of the mounting point in the foreign name space
- Sun's NFS (Network File System)
  - nfs://flits.cs.vu.nl//home/steen
  - Local namespace /remote/vu/mbox refers to remote name space

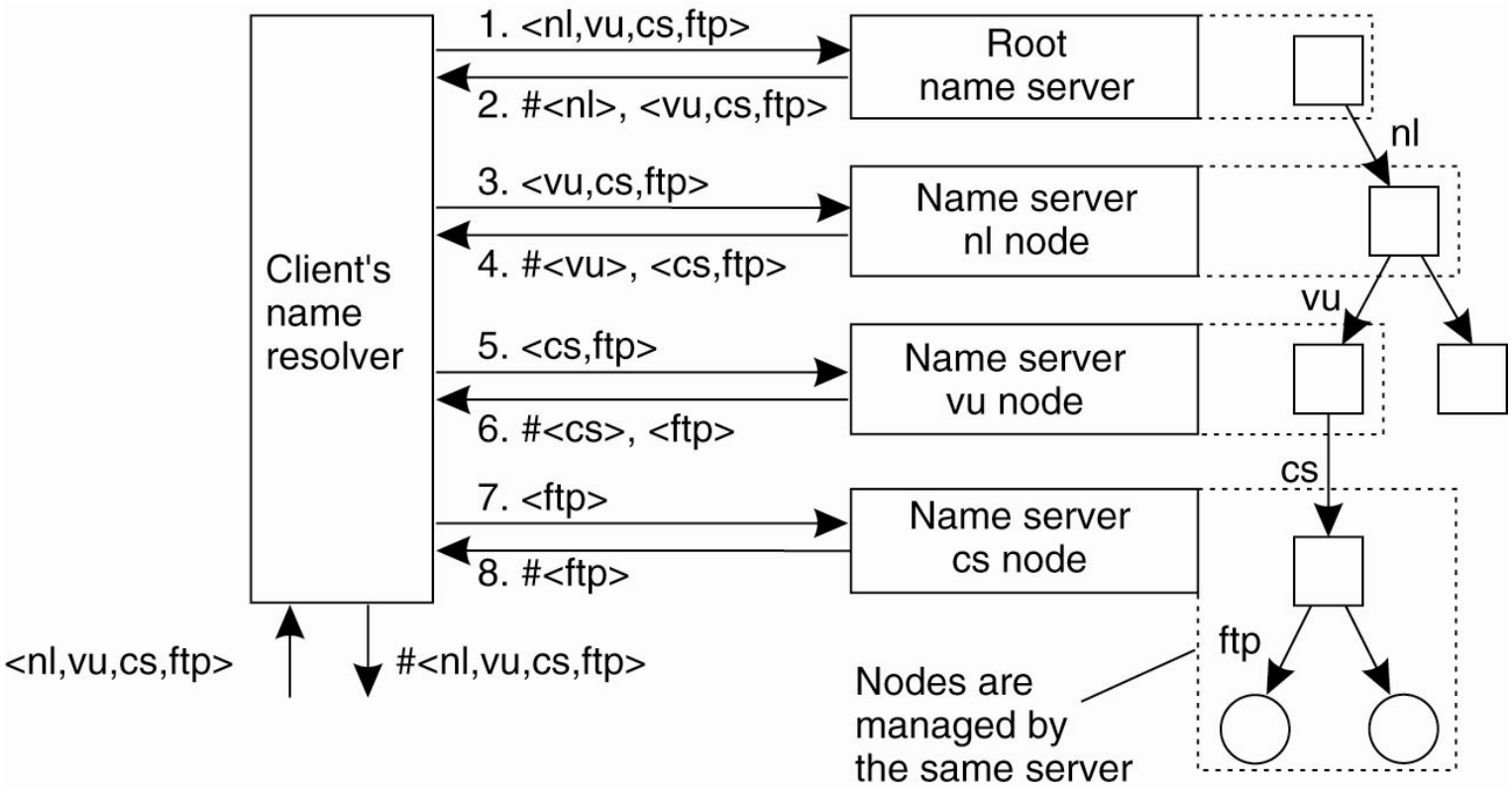
Linking and Mounting

- Mounting remote name spaces through a specific process protocol



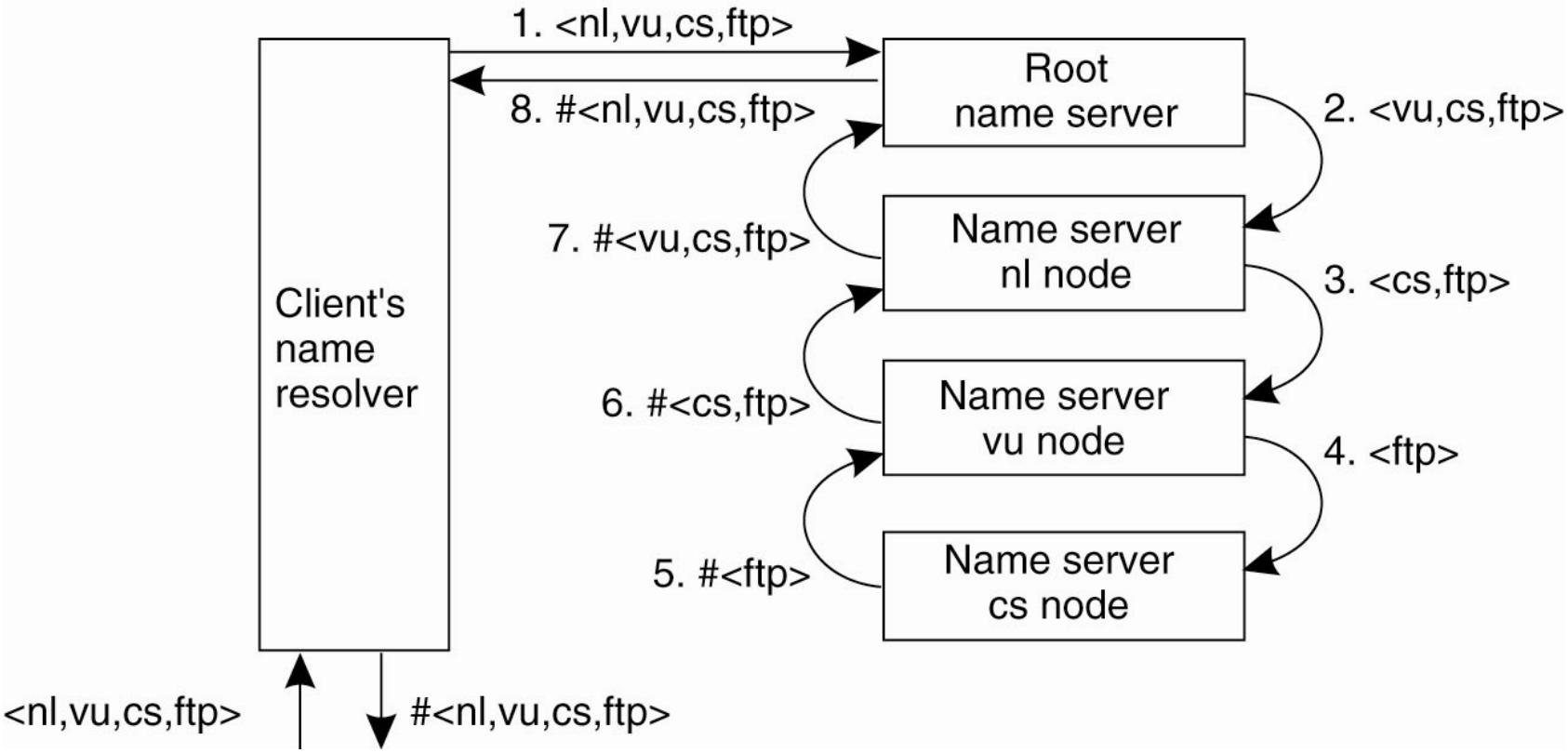
Implementation of Name Resolution

- Iterative name resolution.
- URL: ftp://ftp.cs.vu.nl/pub/globe/index.html.
- #<ftp> is the address of the name server responsible for ftp.cs.vu.nl



Implementation of Name Resolution

- Recursive name resolution. URL: ftp://ftp.cs.vu.nl/pub/globe/index.html
- Pros/cons?



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>
nl	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
root	<nl,vu,cs,ftp>	#<nl>	<vu,cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>	#<nl> #<nl,vu> #<nl,vu,cs> #<nl,vu,cs,ftp>

- Figure 5-17. Recursive name resolution of <*nl, vu, cs, ftp*>. Name servers cache intermediate results for subsequent lookups.

Outline

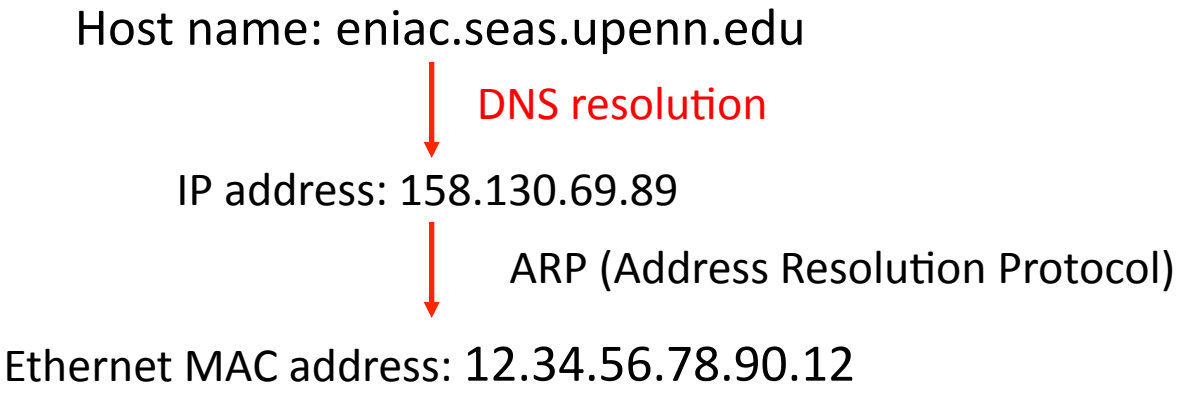
- Naming (general concepts)
- Examples:
  - DNS
  - LDAP

Network Centric View

- Entity: node in network
- Addresses:
  - Says how to reach an object → it has location semantics associated to it
  - Usually, a format easy to process by computers
  - E.g. IP address: 158.130.69.89
- Name:
  - Does not have any location semantics associated to it
  - Usually, a format easier to understand/read/remember by people
  - E.g. eniac.seas.upenn.edu

Binding and Resolution in the Internet

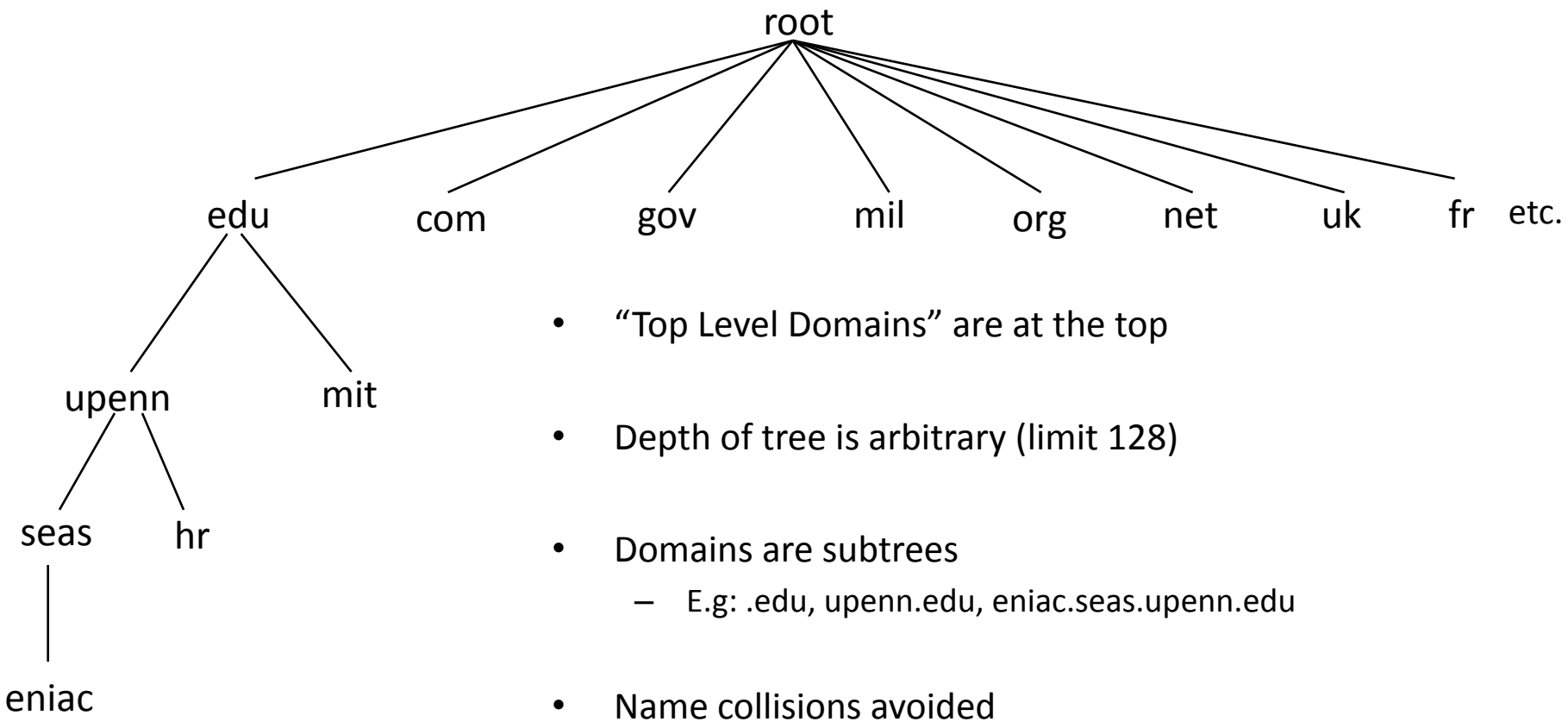
- In general there are multiple mappings



Basic DNS Features

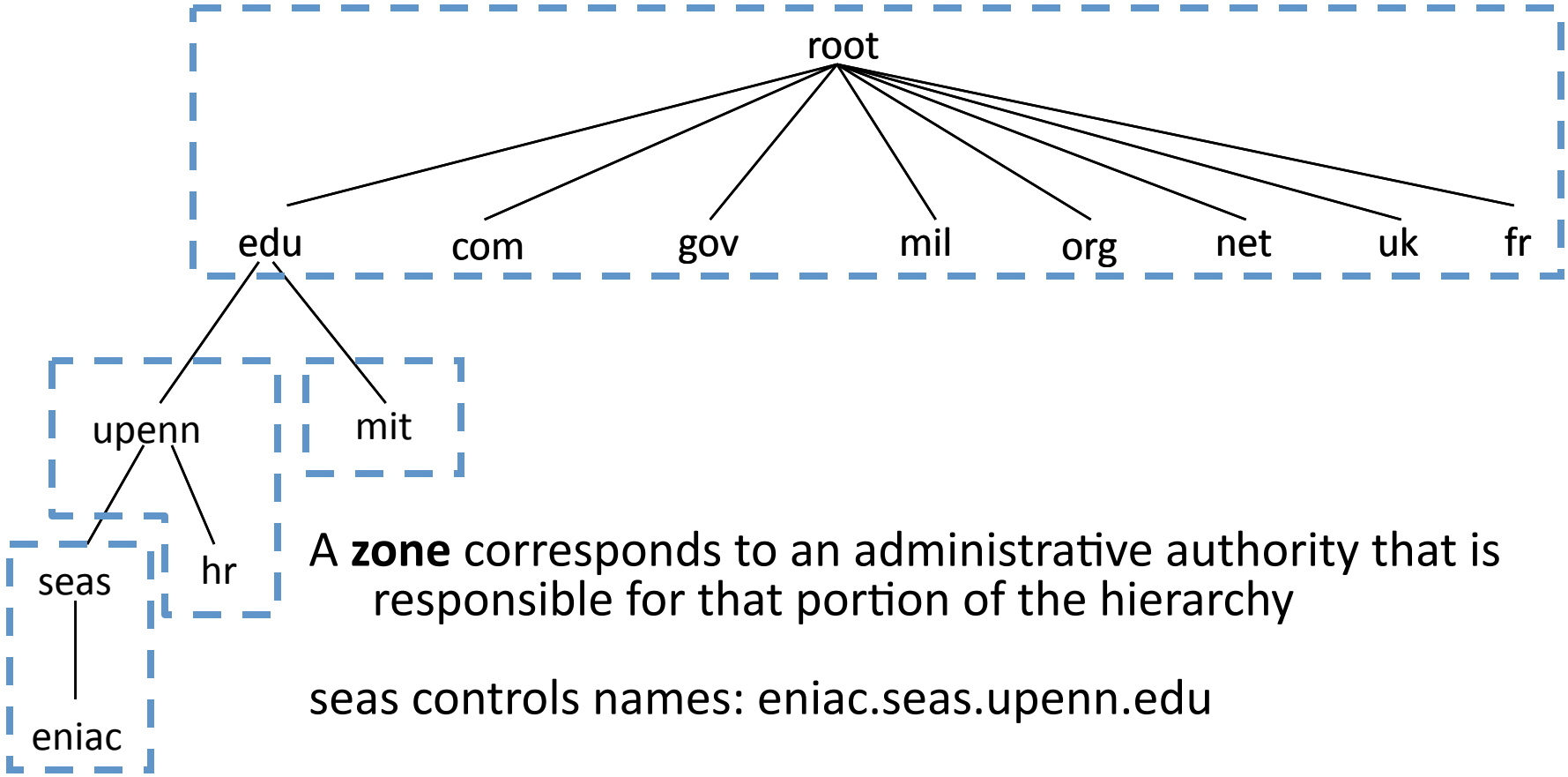
- Hierarchical namespace
  - As opposed to original “flat” namespace
- Distributed storage architecture
  - As opposed to centralized storage (plus replication)
- Client--server interaction on UDP Port 53
  - But can use TCP if desired

Naming Hierarchy



- "Top Level Domains" are at the top
- Depth of tree is arbitrary (limit 128)
- Domains are subtrees
  - E.g: .edu, upenn.edu, eniac.seas.upenn.edu
- Name collisions avoided
  - E.g. upenn.edu and upenn.com can coexist, but uniqueness is job of domain

Host names are administered hierarchically



A **zone** corresponds to an administrative authority that is responsible for that portion of the hierarchy

seas controls names: eniac.seas.upenn.edu

Upenn controls names: eniac.seas.upenn.edu and hr.upenn.edu

Server Hierarchy

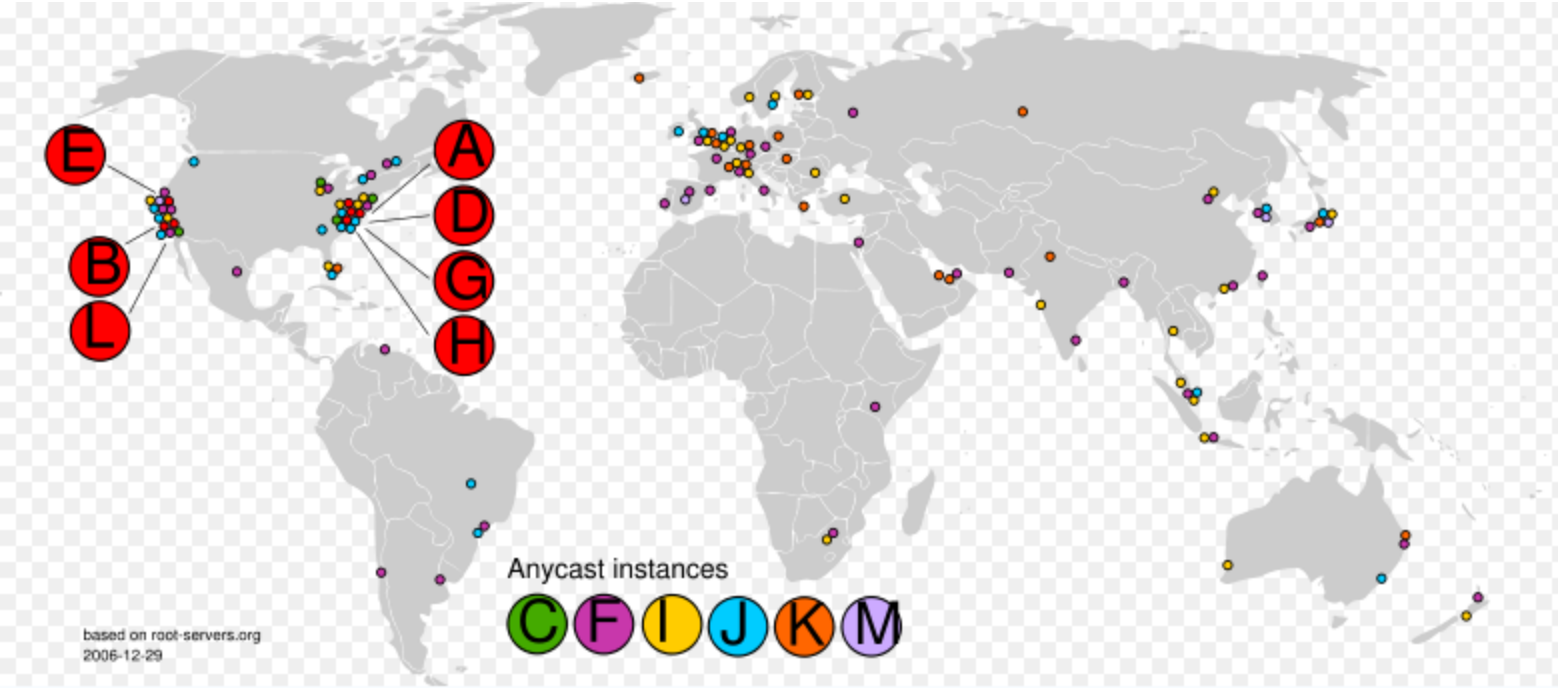
- Each server has authority over a portion of the hierarchy
  - A server maintains only a subset of all names
- Each server contains all the records for the hosts in its zone
  - might be replicated for robustness
- Each server needs to know other servers that are responsible for the other portions of the hierarchy
  - Every server knows the root
  - Root server knows about all top-level domains

DNS Name Servers

- Local name servers:
  - Each ISP (company) has local default name server
  - Host DNS query first goes to local name server

- Authoritative name servers:
  - For a host: stores that host's (name, IP address)
  - Can perform name/address translation for that host's name

DNS: Root Name Servers

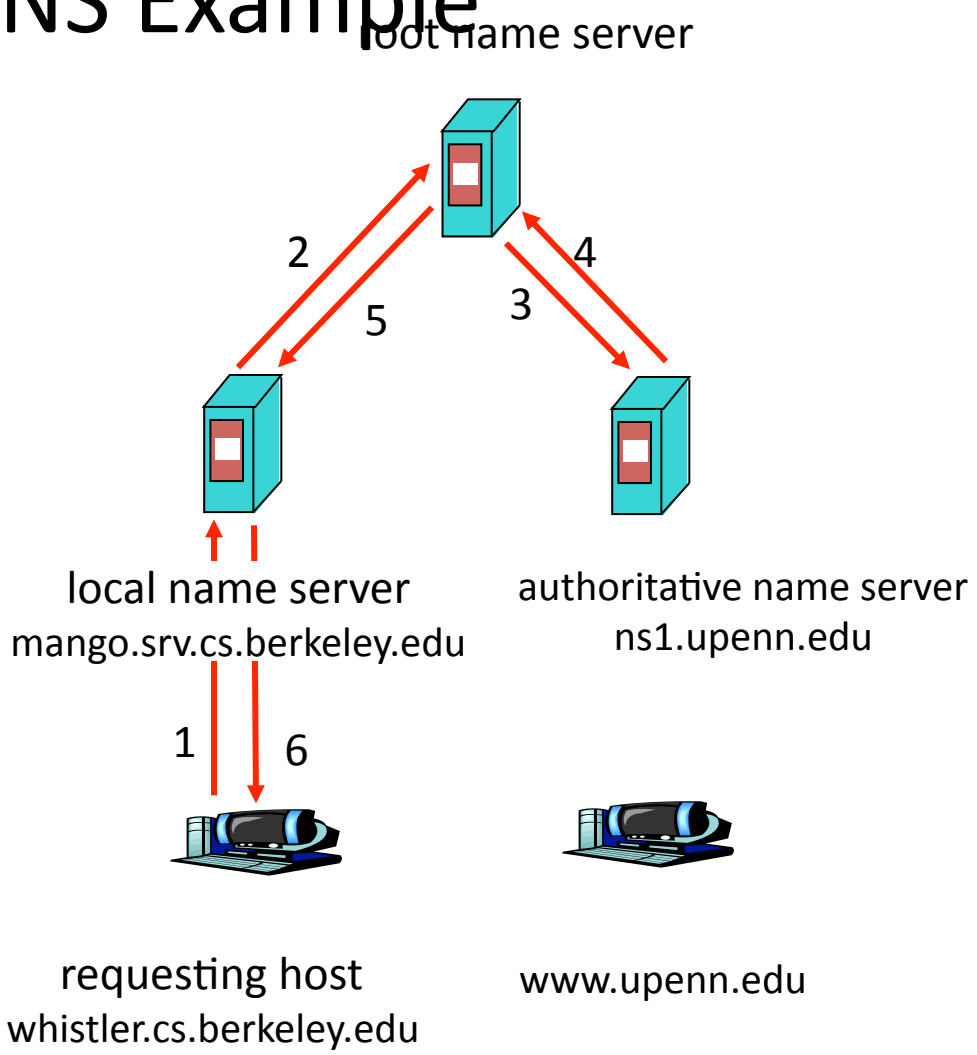


- Contacted by local name server that can not resolve name
- Root name server:
  - Contacts authoritative name server if name mapping not known
  - Gets mapping
  - Returns mapping to local name server
- Currently, there are 13 logical root name servers worldwide (many more physically)
- Lookups to root are extremely rare (only 2% due to caching)

Simple DNS Example

Host **`whistler.cs.cmu.edu`** wants IP address of **`www.upenn.edu`**

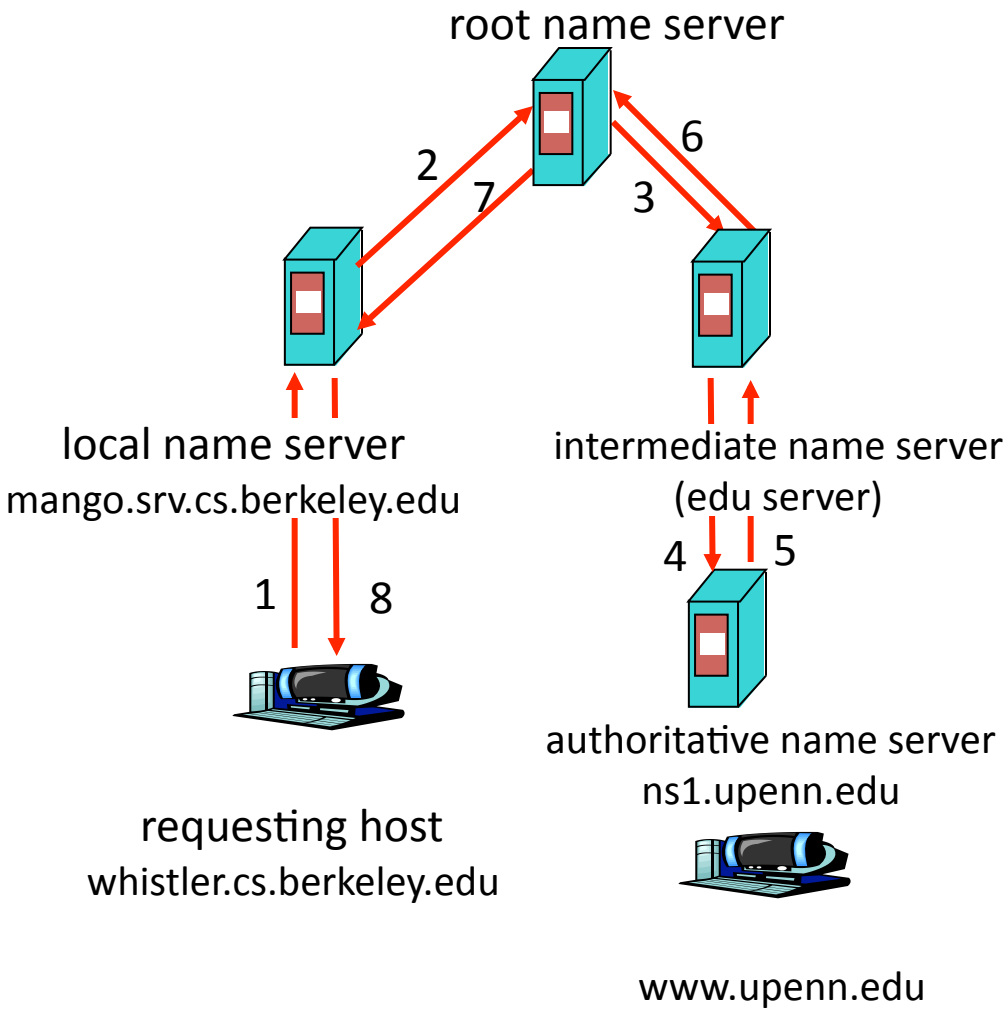
1. Contacts its local DNS server, **`mango.srv.cs.cmu.edu`**
2. **`mango.srv.cs.cmu.edu`** contacts root name server, if necessary
3. Root name server contacts authoritative name server, **`ns1.upenn.edu`**, if necessary



Example of Recursive DNS Query

Root name server:

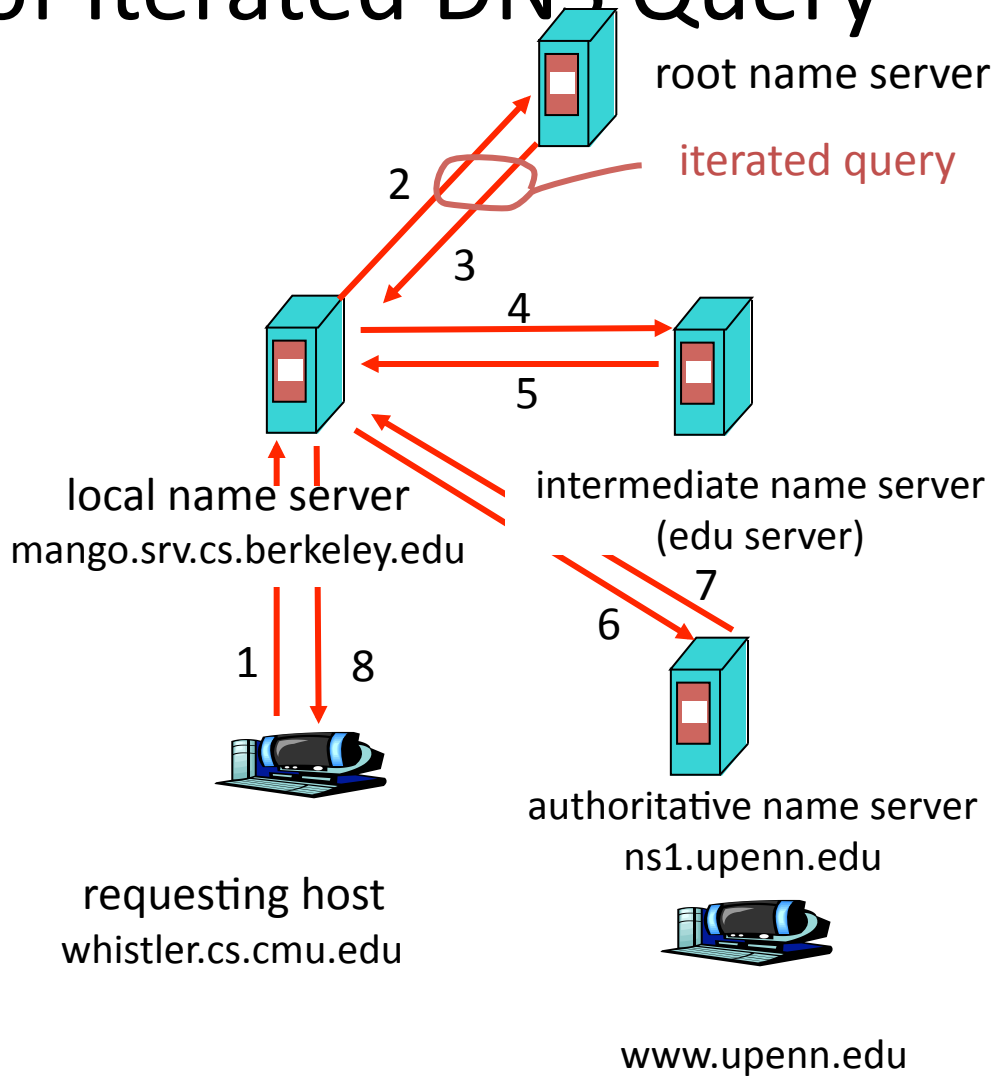
- May not know authoritative name server
- May know intermediate name server: who to contact to find authoritative name server
- DNS servers are not required to support this



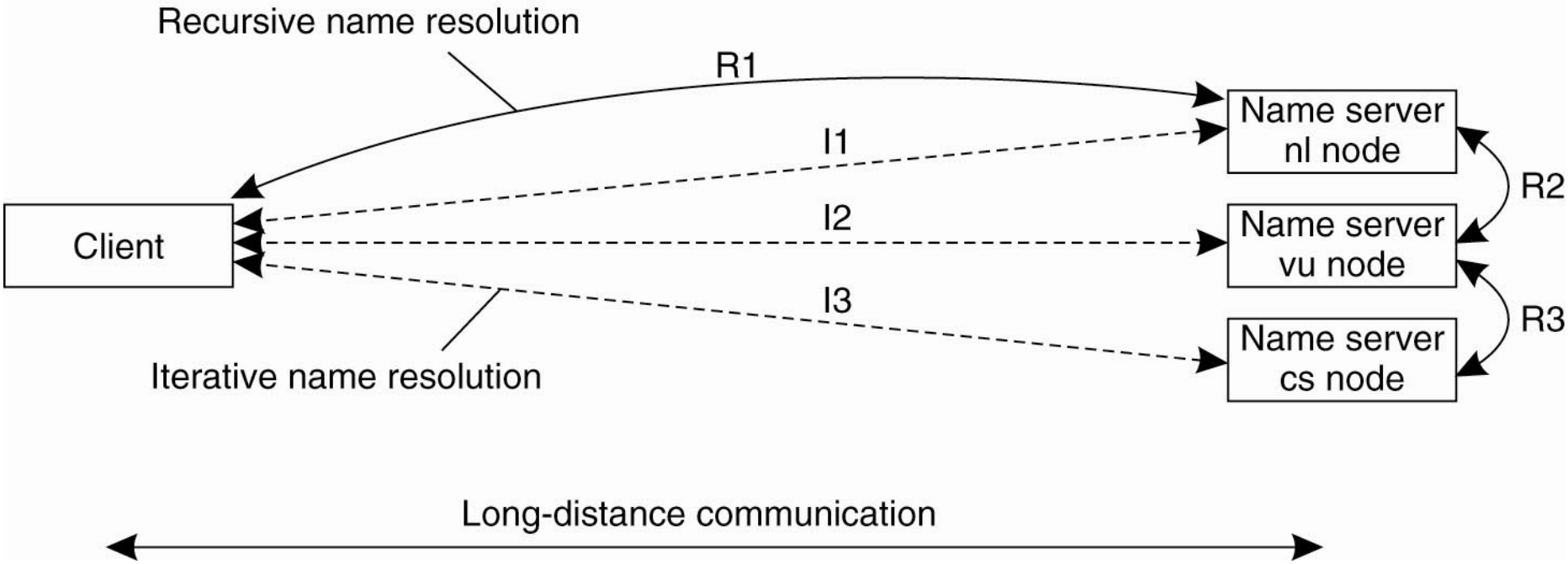
Example of Iterated DNS Query

Iterated query:

- Contacted server replies with name of server to contact
- "I don't know this name, but ask this server"
- All DNS servers supports iterative



DNS: Iterative vs Recursive



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

DNS Records

- Each name server stores a collection of DNS records
- Four fields: <name, value, type, TTL>
- Type = A:
  - name = hostname
  - value = IP address
  - E.g. <example.com., 69.9.64.11, A, 60>
- Type = NS:
  - name = domain
  - value = name of authoritative DNS server for domain
  - E.g. <example.com., ns1.live.secure.com., NS, 60>,
    <com, a.gtld-servers.net, NS, 60>

Types of DNS Records

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

DNS Implementation

Name	Record type	Record value
cs.vu.nl.	SOA	star.cs.vu.nl. hostmaster.cs.vu.nl. 2005092900 7200 3600 2419200 3600
cs.vu.nl.	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl.	MX	1 mail.few.vu.nl.
cs.vu.nl.	NS	ns.vu.nl.
cs.vu.nl.	NS	top.cs.vu.nl.
cs.vu.nl.	NS	solo.cs.vu.nl.
cs.vu.nl.	NS	star.cs.vu.nl.
star.cs.vu.nl.	A	130.37.24.6
star.cs.vu.nl.	A	192.31.231.42
star.cs.vu.nl.	MX	1 star.cs.vu.nl.
star.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
star.cs.vu.nl.	HINFO	"Sun" "Unix"
zephyr.cs.vu.nl.	A	130.37.20.10
zephyr.cs.vu.nl.	MX	1 zephyr.cs.vu.nl.
zephyr.cs.vu.nl.	MX	2 tornado.cs.vu.nl.
zephyr.cs.vu.nl.	HINFO	"Sun" "Unix"

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

DNS Implementation

ftp.cs.vu.nl.	CNAME	soling.cs.vu.nl.
www.cs.vu.nl.	CNAME	soling.cs.vu.nl.
soling.cs.vu.nl.	A	130.37.20.20
soling.cs.vu.nl.	MX	1 soling.cs.vu.nl.
soling.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
soling.cs.vu.nl.	HINFO	"Sun" "Unix"
vucs-das1.cs.vu.nl.	PTR	0.198.37.130.in-addr.arpa.
vucs-das1.cs.vu.nl.	A	130.37.198.0
inkt.cs.vu.nl.	HINFO	"OCE" "Proprietary"
inkt.cs.vu.nl.	A	192.168.4.3
pen.cs.vu.nl.	HINFO	"OCE" "Proprietary"
pen.cs.vu.nl.	A	192.168.4.2
localhost.cs.vu.nl.	A	127.0.0.1

Outline

- Naming (general concepts)
- Examples:
  - DNS
  - LDAP

Attribute-based Naming

- Motivation:
  - Sometimes more convenient to name, and look up entities by means of their *attributes*
  - Traditional directory services (aka yellow pages).
  - May make sense to think of names as entries in a distributed database.
- Lightweight Directory Access Protocol (LDAP)
  - Widely implemented
  - Implement basic directory service as database, and combine with traditional structured (hierarchical) naming system.
  - Enables *attribute-based naming*
  - Adopted by Microsoft's Active Directory service
  - Derived from OSI X.500 directory service
  - Used not just for host names
    - user names / password lookups
    - certificate distribution

LDAP <Attribute. Value> pairs

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
CommonName	CN	Main server
Mail_Servers	—	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	—	130.37.20.20
WWW_Server	—	130.37.20.20

Typical use:

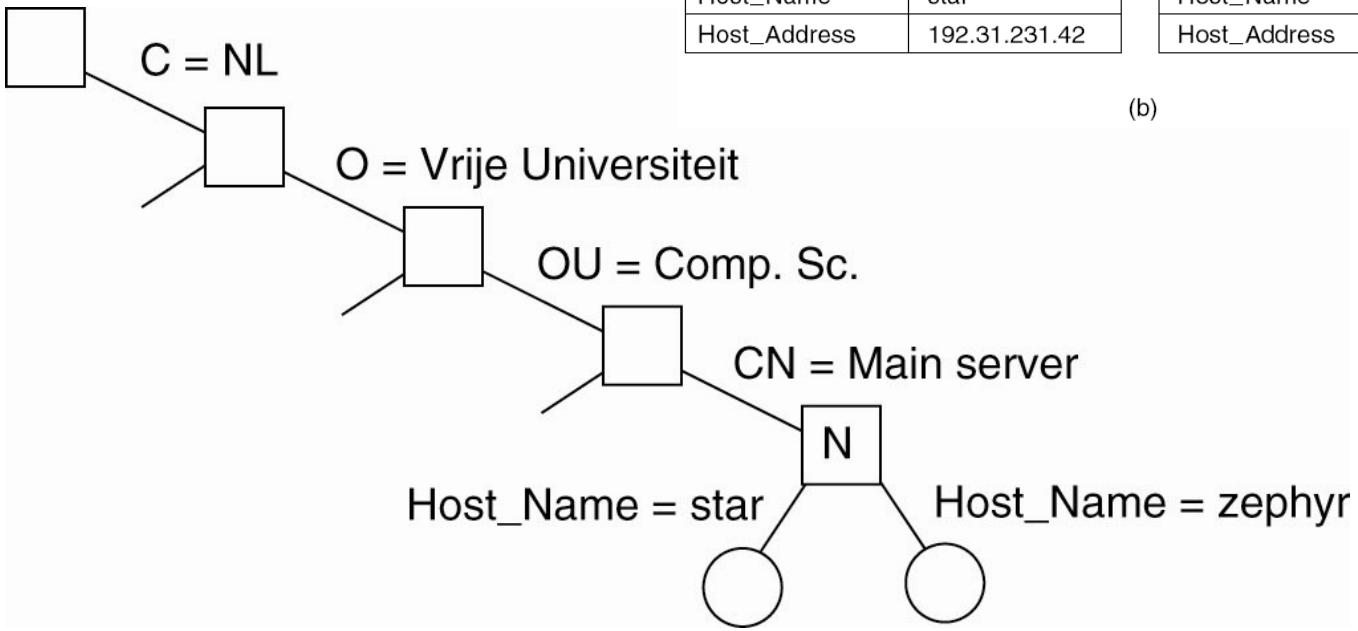
Client connects to LDAP server at TCP port 389. Authorized Client can add, modify, or delete entries.

- A simple example of an LDAP directory entry using LDAP naming conventions.
  - Each entry has a set of attributes. Each attribute is a <name,value> pair. Single vs multi-valued attributes
  - Collection of directory entries is called a directory information base (DIB)
- First five attributes (C,L,O,OU,CN) is the sequence of *naming attributes* (also called relative distinguished name, RDN).
  - Used to globally uniquely identify record.
  - E.g .Names of the form /C=NL/O=Vrije Universiteit/OU=Comp.Sc.

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

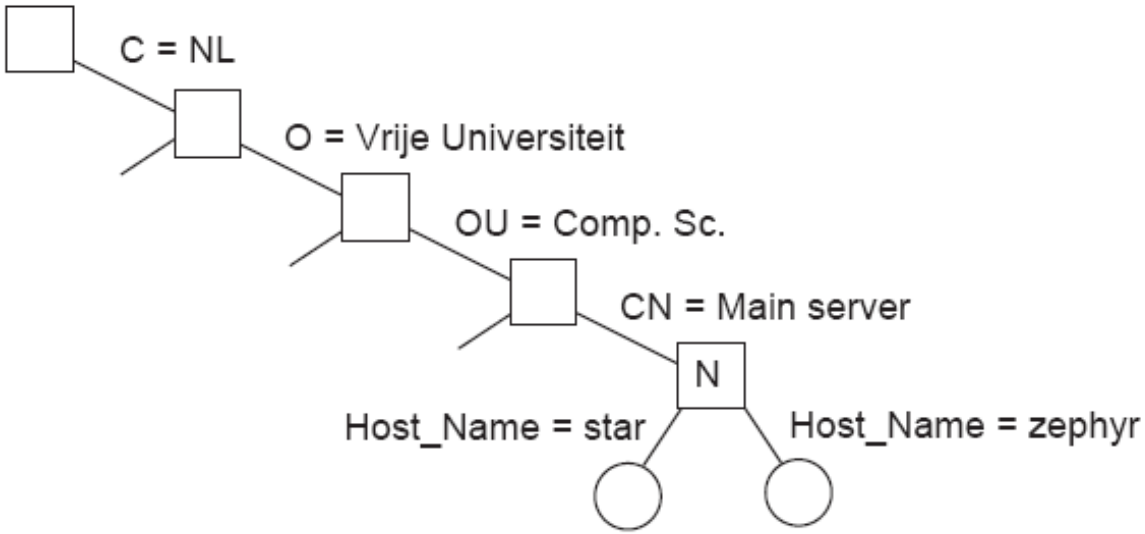
Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

(b)



- Directory information tree (DIT): naming graph of an LDAP directory service in which each node represents a directory entry
- Distribute authority across machines, similar to zones in DNS
- N corresponds to earlier entry. Additional RDN Host_Name
- /C=NL/O=Vrije Universiteit/OU=Comp. Sc./CN=Main Server
- Two operations: *read* (retrieve single record), *list* (all outgoing edges)

Putting it all together: LDAP



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

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

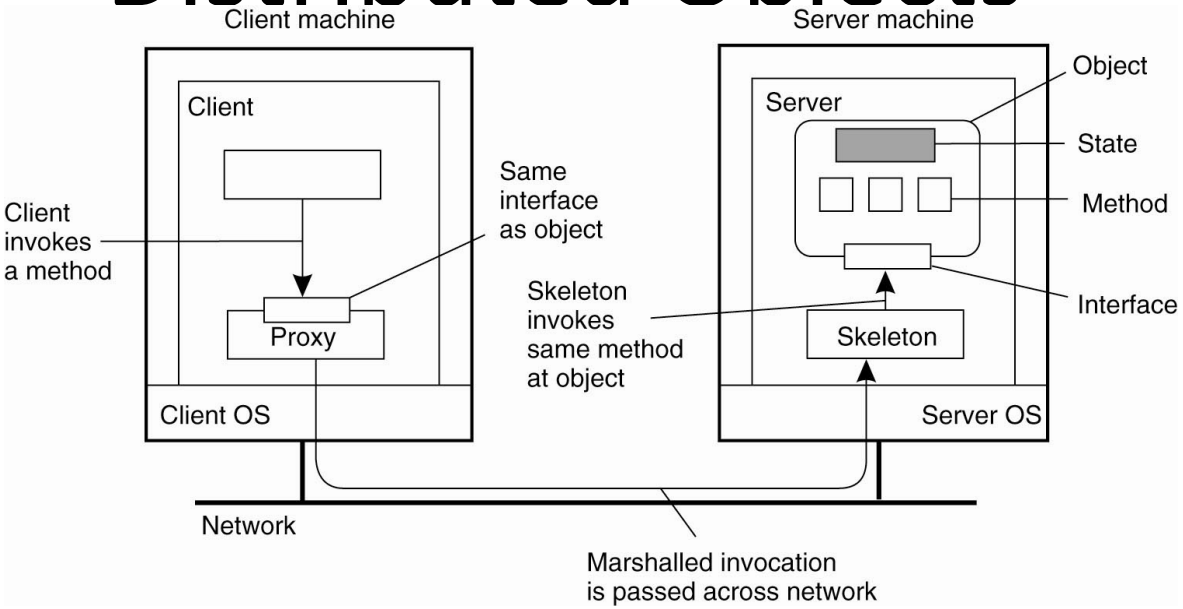
Find the list of all main servers at Vrije Universiteit:

```
answer = search("&(C  = NL)
                 (O  = Vrije Universiteit)
                 (OU = *)
                 (CN = Main server)")
```

Communication in a distributed system

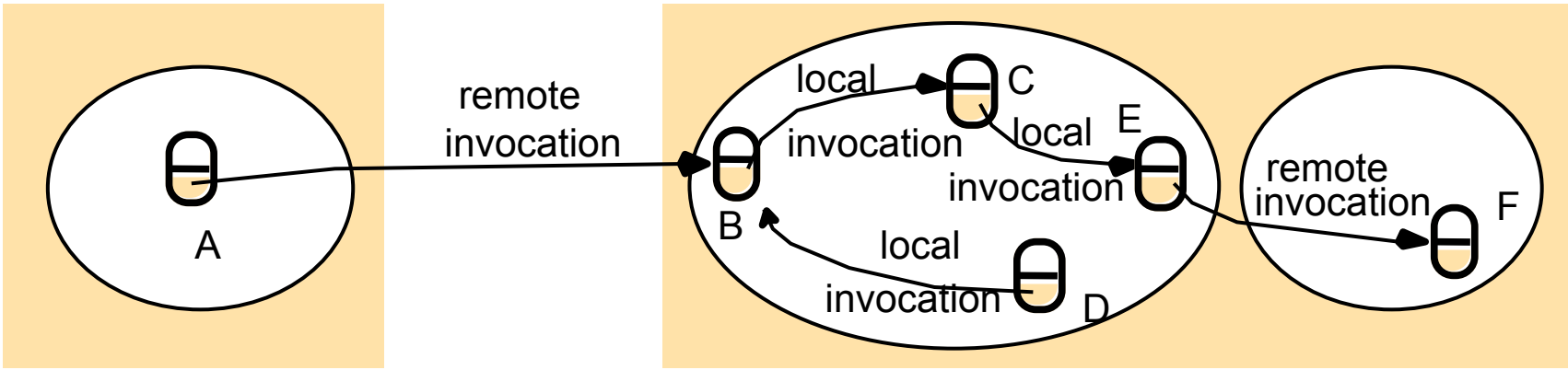
- Layering
- Sockets (Unreliable datagrams, reliable in-order)
- Remote procedure call / Remote object invocation
- Distributed message queues

Distributed Objects



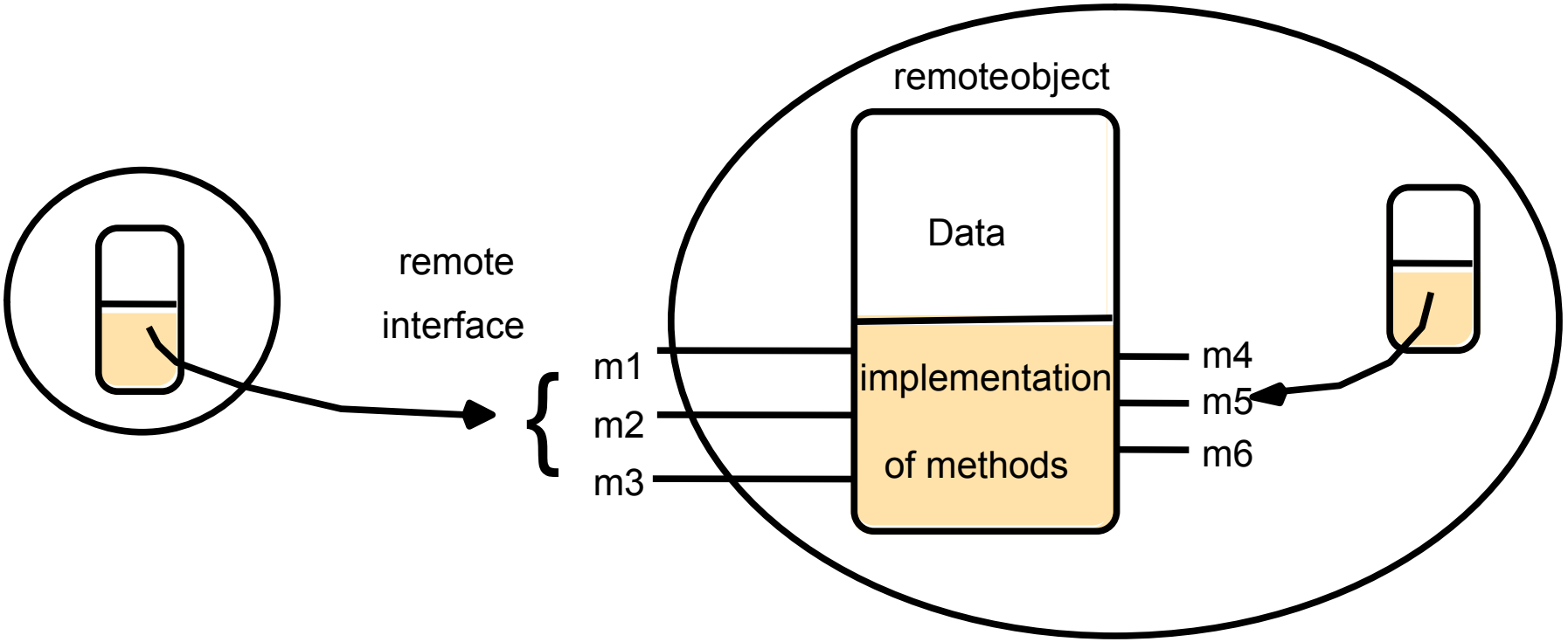
- Object – encapsulate data (state), and operations on data (methods)
- Common organization of a remote object with client-side proxy.
- Client proxy = client-side stub
- Server skeleton = server-side stub
- State is local, but interfaces are remote

Distributed object model



- Each process contains objects, some of which can receive remote invocations, others only local invocations
- Local vs remote object references
- Concurrent access to objects -> concurrency control (monitors, and CVs)
- Those that can receive remote invocations are called *remote objects*
- Objects need to know the *remote object reference* of an object in another process in order to invoke its methods. How do they get it?
- The *remote interface* specifies which methods can be invoked remotely

A remote object and its remote interface



The *Naming* class of Java RMIregistry

void rebind (String name, Remote obj)

This method is used by a server to register the identifier of a remote object by name,

void bind (String name, Remote obj)

This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)

This method removes a binding.

Remote lookup(String name)

This method is used by clients to look up a remote object by name. A remote object reference is returned.

String [] list()

This method returns an array of Strings containing the names bound in the registry.

Name - //host:port/name. Defaults to localhost, port 1099

Hello-world example (interface)

http://java.sun.com/j2se/1.5.0/docs/guide/rmi/hello/hello-world.html

Remote interface:

```
package example.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
```

Hello-world example (server)

Server implementation:

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Hello {

    public Server() {}

    public String sayHello() {
        return "Hello, world!";
    }

    public static void main(String args[]) {

        try {
            Server obj = new Server();
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);

            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Hello-world example (client)

Client implementation:

```
package example.hello;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

    private Client() {}

    public static void main(String[] args) {

        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Hello-world example (Registry service)

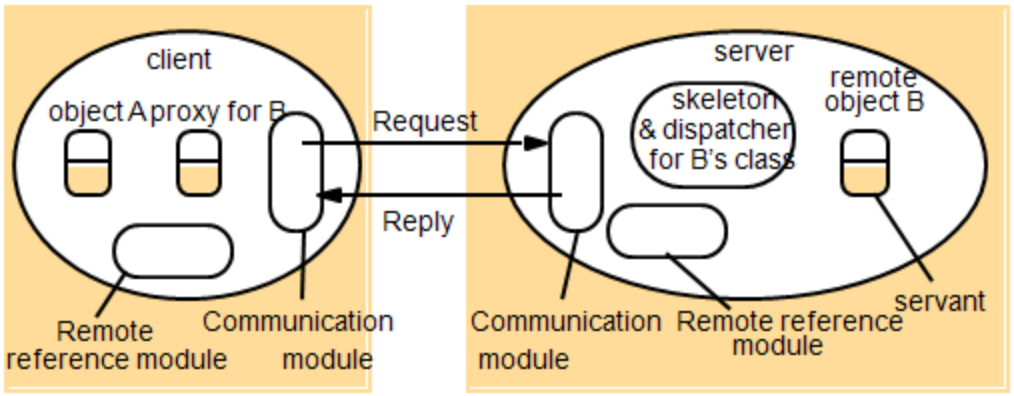
Starting the RMI Registry:

By default, the registry runs on TCP port 1099. To start a registry on a different port, specify the port number from the command line. For example, to start the registry on port 2001 on a Windows platform:

```
start rmiregistry 2001
```

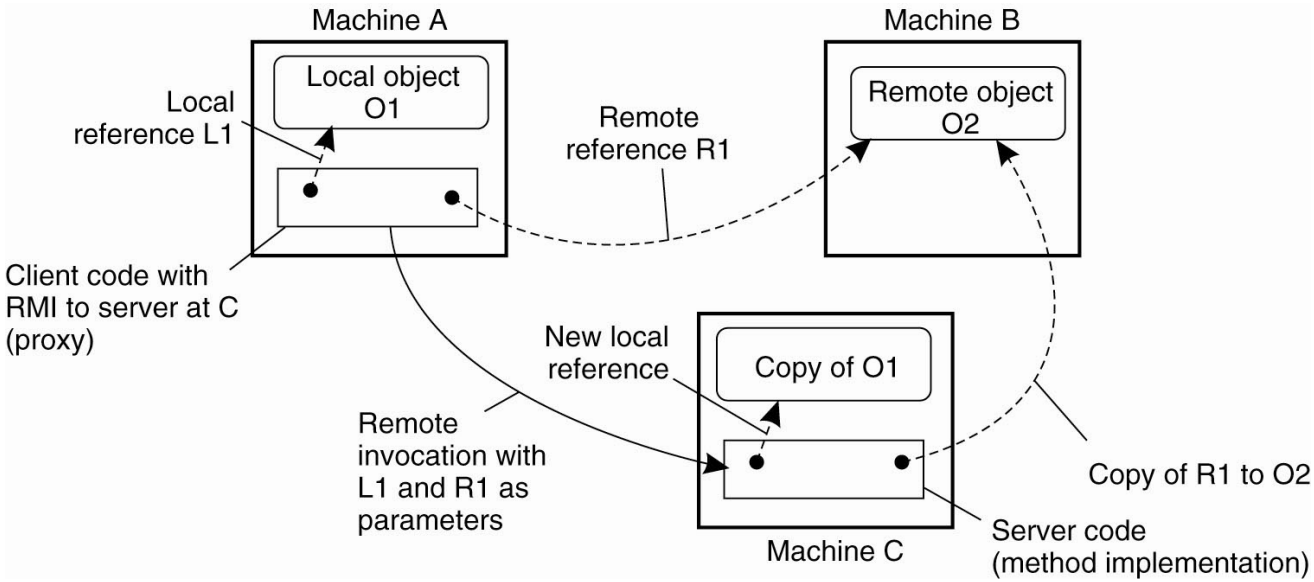
If the registry will be running on a port other than 1099, you'll need to specify the port number in the calls to `LocateRegistry.getRegistry` in the `Server` and `Client` classes. For example, if the registry is running on port 2001 in this example, the call to `getRegistry` in the server would be:

```
Registry registry = LocateRegistry.getRegistry(2001);
```



Textbook page 460

Parameter Passing



- The situation when passing an object by reference or by value.
- Consider A making a call C.proc(O1,O2)
- Local objects can be passed by value. E.g. machine A's O1 is copied to machine C
- Remote objects only pass pointers. Hence, C will need to do a remote invocation on O2.