Distributed systems and their properties

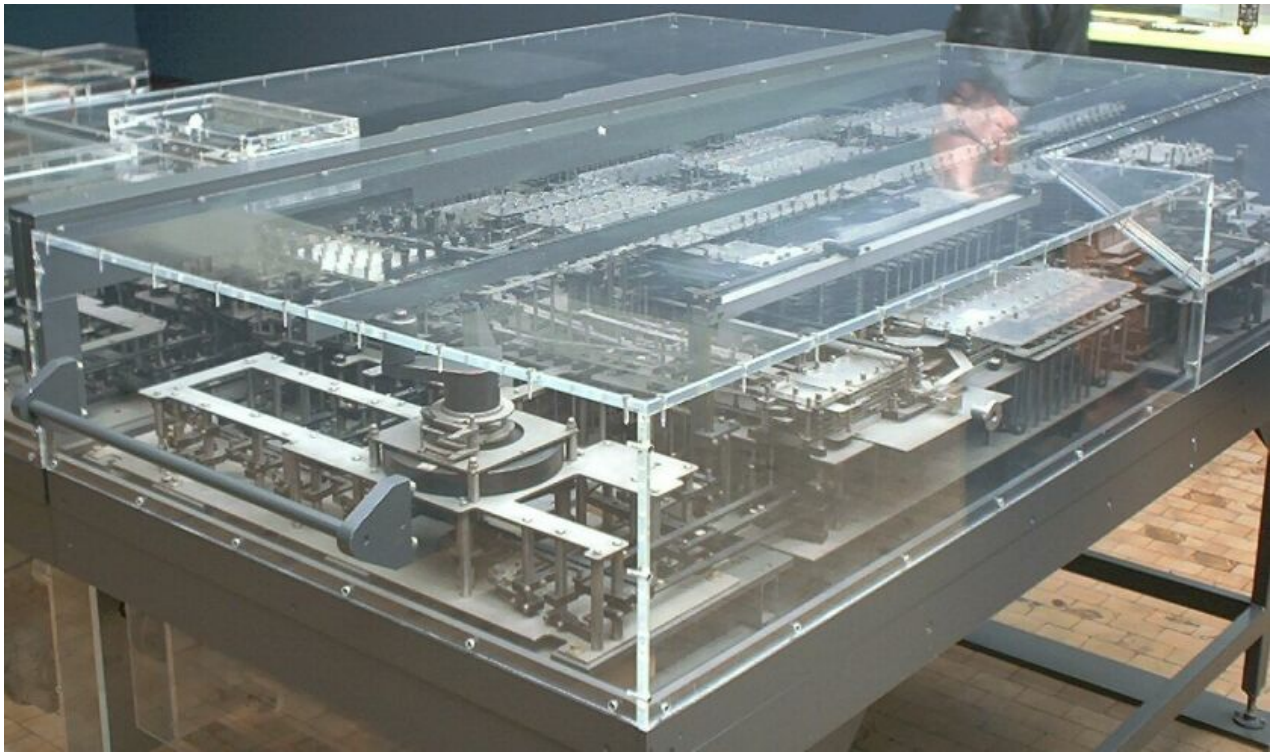
(Lecture 2 for Programming of Interactive Systems)

Fredrik Kilander & Wei Li

Agenda

- Interaction in Interactive Systems
- Evolution in Distributed Systems
- Distributed Systems
  - OSI Model & Middleware
  - Remote Procedural Call (RPC)
  - Remote Method Invocation (RMI)
  - Message Oriented Middleware
  - Sockets
- Summary

Interactive Systems



Early computing – Z1 (1938)

Interactive Systems

Early computing – Manchester Mark I (1949)

Interactive Systems

- Special environments

iLounge (2007)

Interactive Systems

Tangible interfaces – Reactable (2009)

Interactive Systems

Gesture+voice interfaces – Kinect (2010)

Interactive Playground

Banabi, by Maurizio Piraccini

Intangible Systems

Computer networks – peer networks, ad-hoc networks, temporary associations

Definition of a Distributed System

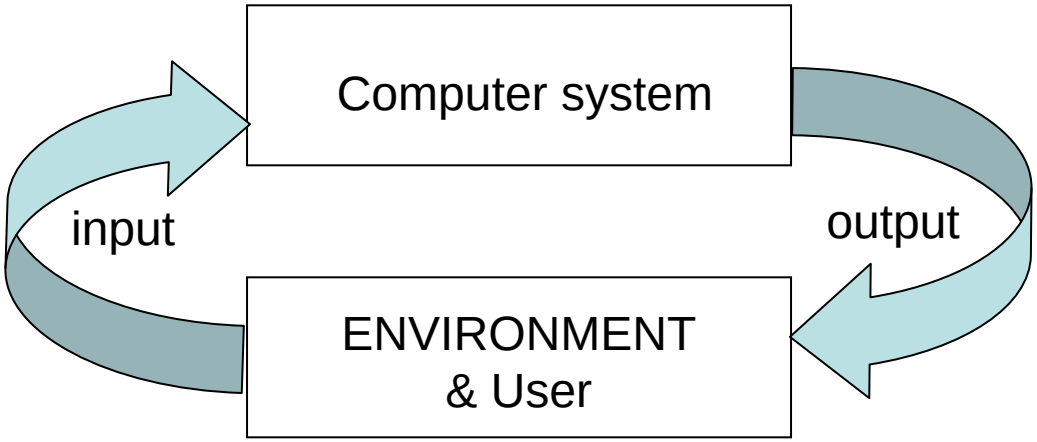
A distributed system is: A collection of independent computers that appears to its users as a single coherent system.

-- Andrew S. Tanenbaum

- Machines are running autonomously
- Software hides that processes and resources are physically distributed across multiple computers over networks

Goal: Users and applications can access remote resources and share them with other users in a controlled way through the interaction with a DS in a consistent and uniform way

Interactive System



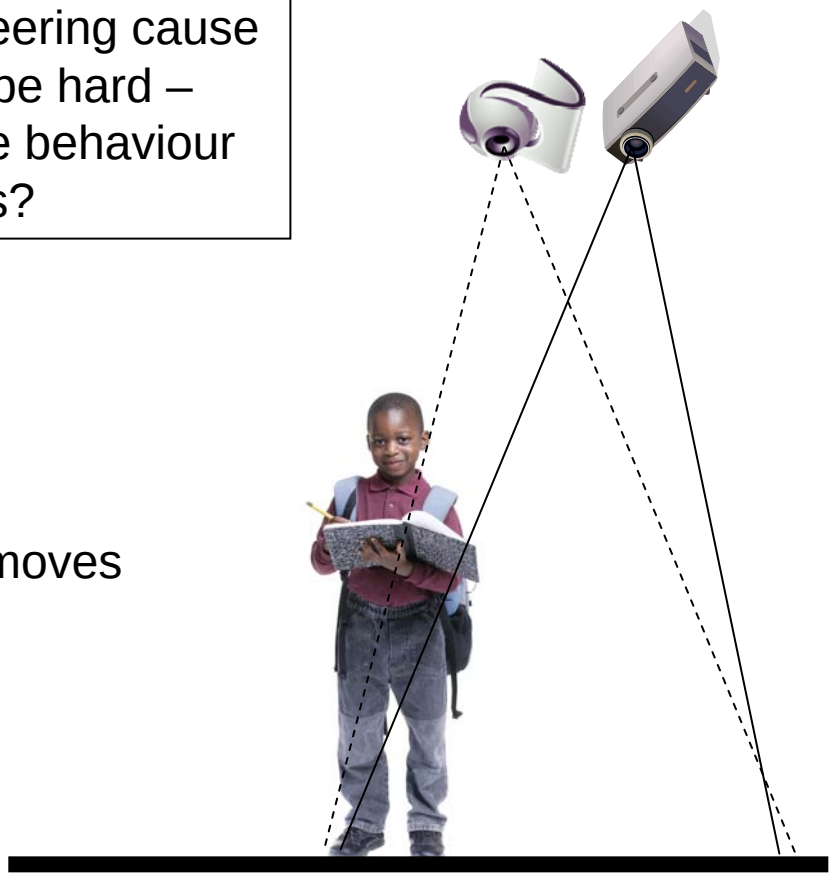
Interactive systems

Interacting with and through different systems

Interactive System

Reverse engineering cause and effect can be hard – what guides the behaviour of the creatures?

H1 – it is afraid of being stepped on
H2 – it is afraid of shadow
H3 – when the camera can't see it, it moves



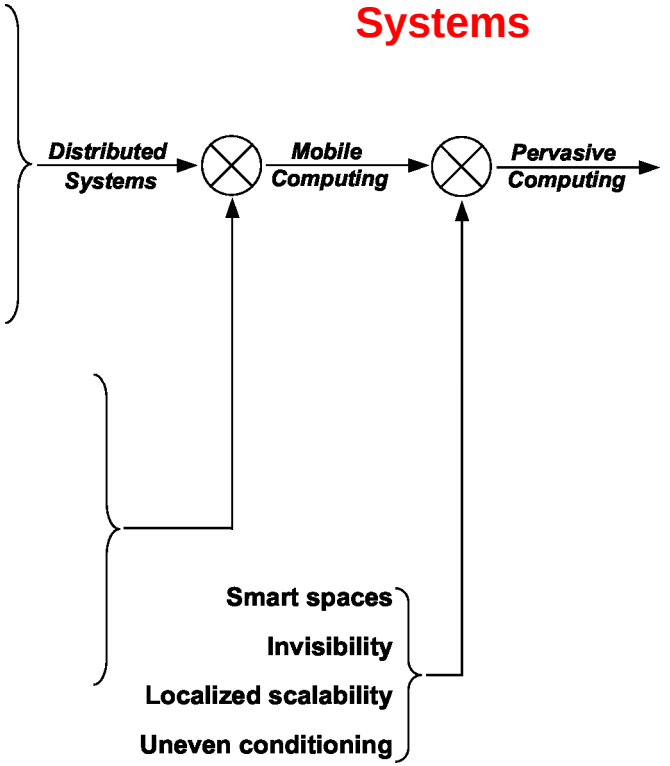
Interactive Systems

- Distributed Systems (DS)
- Mobile Computing (MC)
- Pervasive Computing (PC)

Remote communication
protocol layering, RPC, end-to-end args . . .
Fault tolerance
ACID, two-phase commit, nested transactions
. . .
High Availability
replication, rollback recovery, . . .
Remote information access
dist. file systems, dist. databases, caching, . . .
Distributed security
encryption, mutual authentication, . . .

Mobile networking
Mobile IP, ad hoc networks, wireless TCP fixes, . . .
Mobile information access
disconnected operation, weak consistency, . . .
Adaptive applications
proxies, transcoding, agility, . . .
Energy-aware systems
goal-directed adaptation, disk spin-down, . . .
Location sensitivity
GPS, WiFi triangulation, context-awareness, . . .

Contemporary Interactive Systems



Pervasive Computing: Vision and Challenges
M. Satyanarayanan, School of Computer Science Carnegie Mellon University

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 (static deployment)
Relocation	Hide that a resource may be moved to new location during use (dynamic)
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive 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.

Important Issues in Distributed Systems

- **Communication:**
  - the basis of a DS is to support access to remote resources
- **Processes**:
  - communication takes place between processes
  - schedule and manage processes
  - threads, code migration, client/server, software agents

Important Issues in Distributed Systems

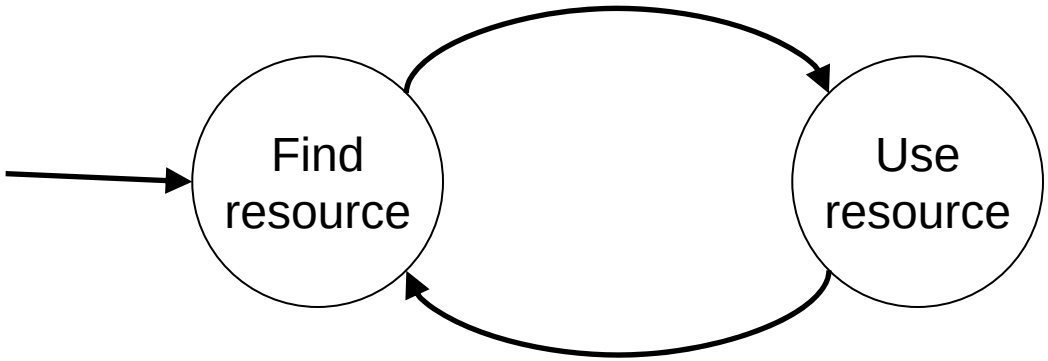
- **Naming:**
  - the shared resources in a DS have to be identified uniquely
  - each identification should be resolvable for retrieving the entity it refers to
  - *Example 1: URL→IPnr:port→MAC*
  - *Example 2: Lisa→telephone nr →GSM tower→terminal (handset)*

Important Issues in Distributed Systems

- **Synchronization:**
  - protect concurrent access from conflicts; one writer; only read from a consistent state
- **Consistency and Replication:**
  - data are replicated to enhance reliability and performance
  - keep replicas consistent (also called data synchronization); cache management.

Important Issues in Distributed Systems

- **Fault Tolerance:**
  - DS are subject to failures as communication spans multiple computers or even networks
  - it is important to have automatic recovery from failures without affecting the overall performance
  - automatic configuration and adaptation



Important Issues in Distributed Systems

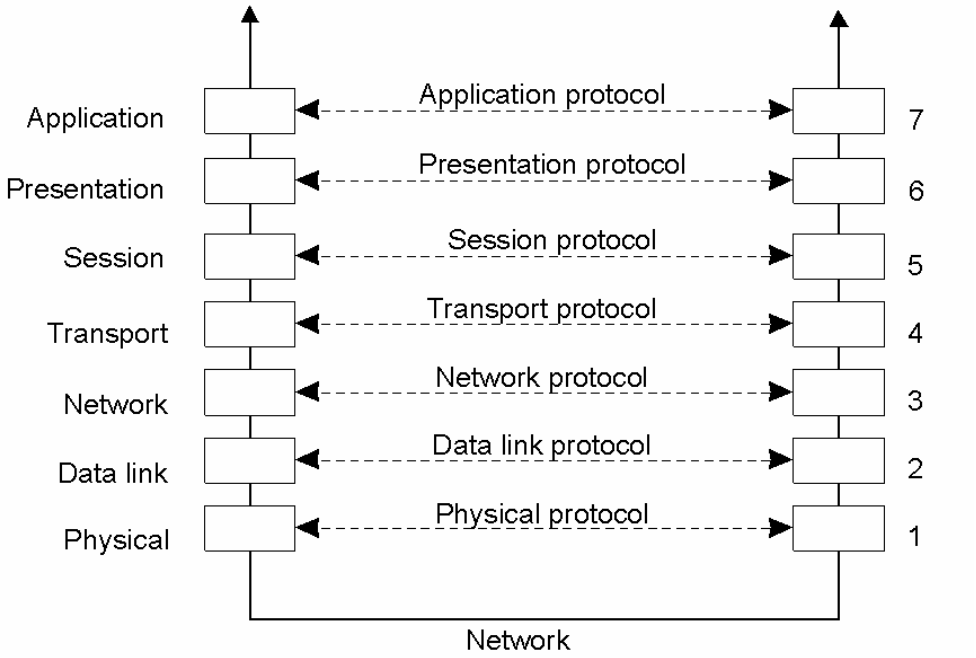
- **Security:**
  - secure communication (secure channel)
  - provide access protection to prevent malicious or unauthorized access
  - *Example: only group members currently in room 4 are allowed to read each other's files*
  - authentication and auditing
  - distributed administration
  - authority in peer-to-peer systems

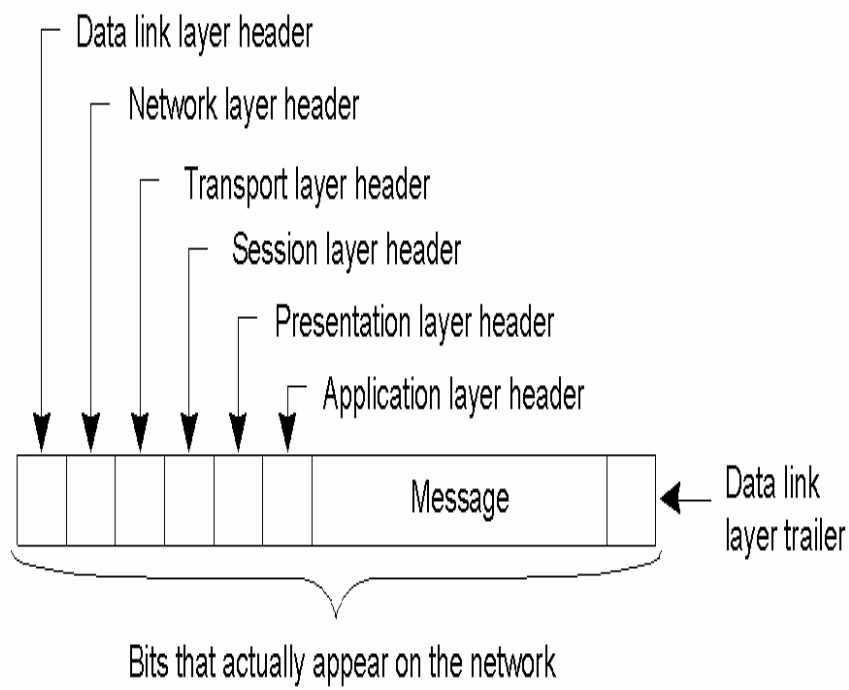
Communication

Layers, interfaces, and protocols in the OSI (Open Systems Interconnection) reference model.

- Divided into 7 layers. Each deals with one

specific aspect of the communication



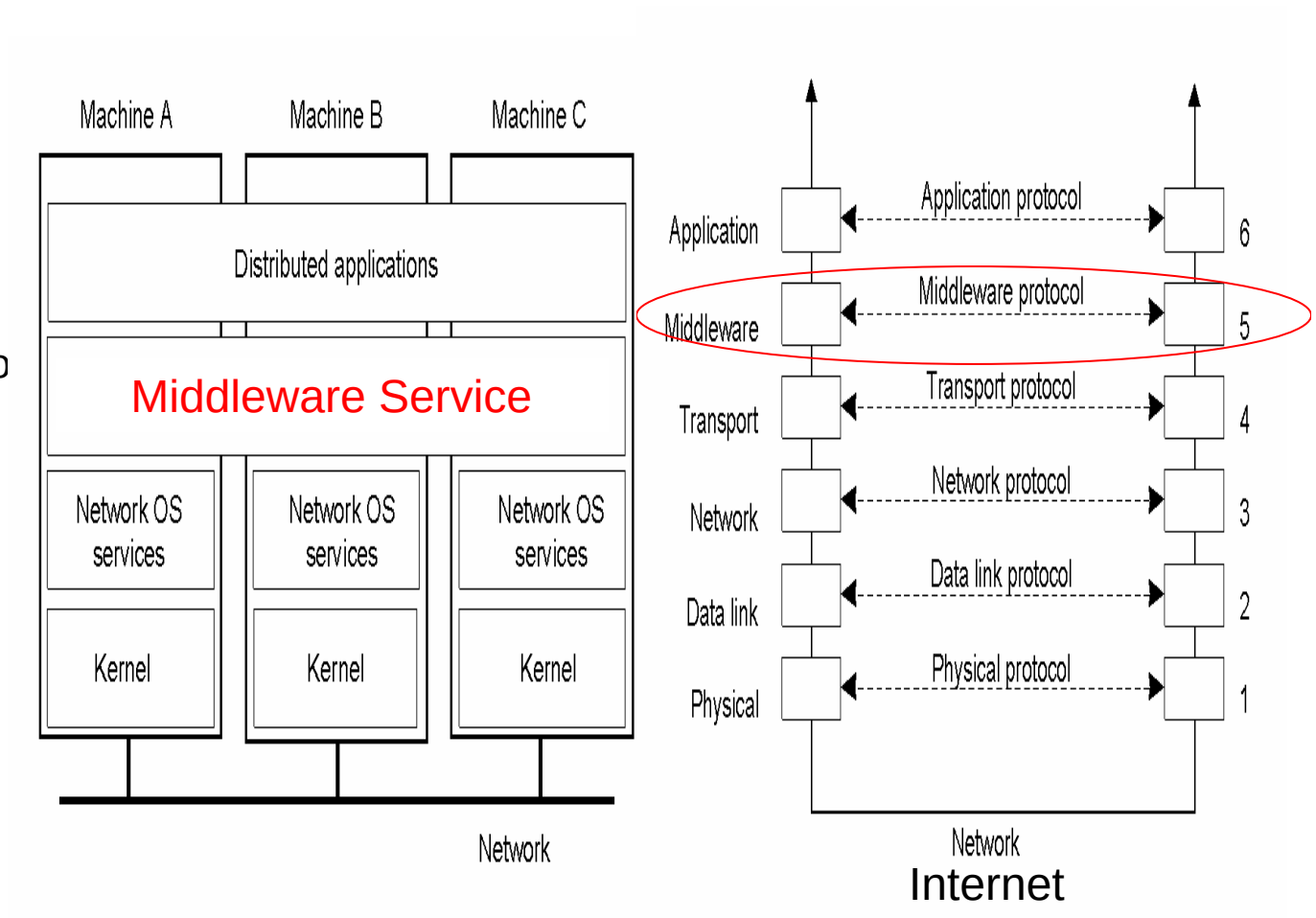


Positioning Distributed System (Middleware)

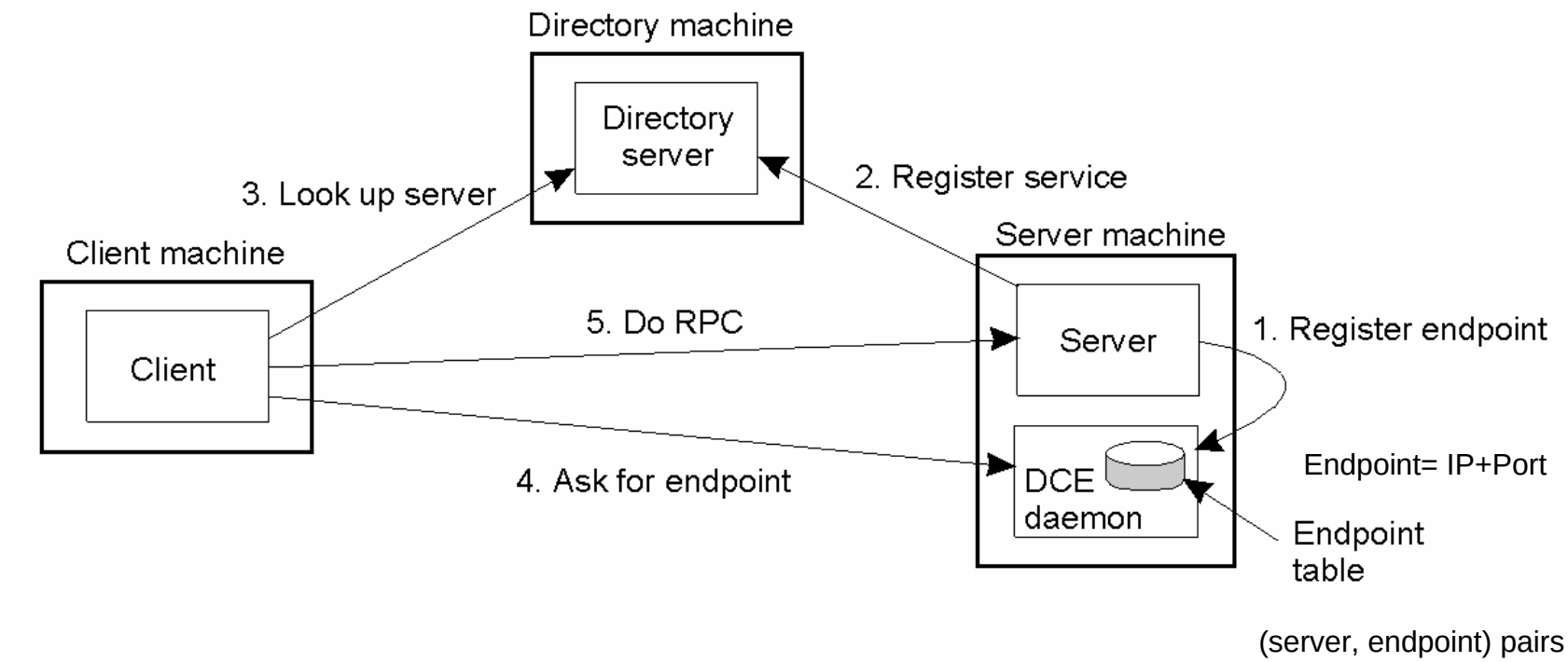
- Distributed systems are often organized as a software layer placed between user and application and the underneath operation system.

- A distributed system is also called **middleware** and the middleware layer extends over multiple machines.

- Middleware is an **application layer protocol** (the layers above transport layer are all categorized into application layer).



Binding a Client to a Server

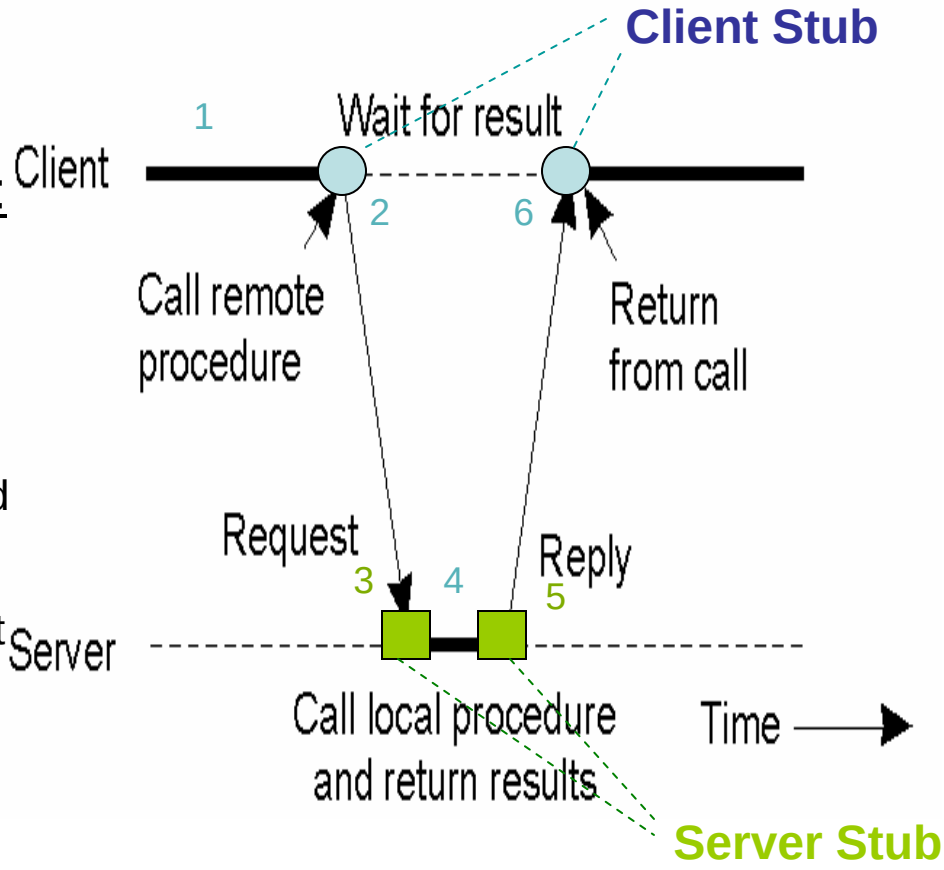


Client-to-server binding in DCE
(distributed computing environment 1990-)

Remote Procedure Call (RPC)

Extend the procedure call over the network by allowing programs to call procedures located on other machines through Stubs:

1. Client program calls *client stub* to place a remote procedure call
2. Client stub builds a request message and sends to remote server
3. *Server stub* receives the message and unpacks parameters, calls the local procedure
4. Procedure executes and returns result to the server stub
5. Server stub packs it in message, and sends back to client stub
6. Client stub unpacks result, returns to client program



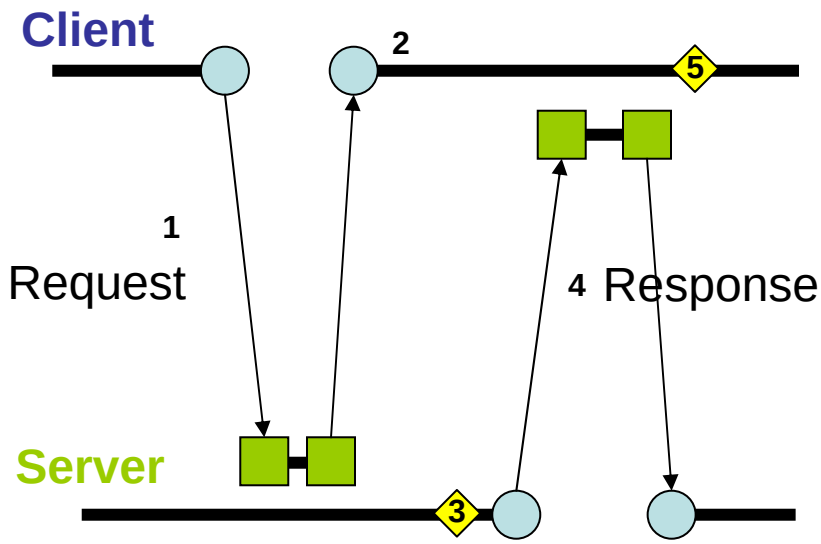
A synchronous Client/Server Model

Remote Procedure Call (RPC)

Extend the procedure call over the network by allowing programs to call procedures located on other machines

1. The client calls the server to place a service request
2. While the server processes the request, the client keeps executing
3. The server sees the request, and responds to it
4. The server calls back to the client with the result
5. The client sees the response and acts upon it

+ Higher parallelism
- Higher complexity



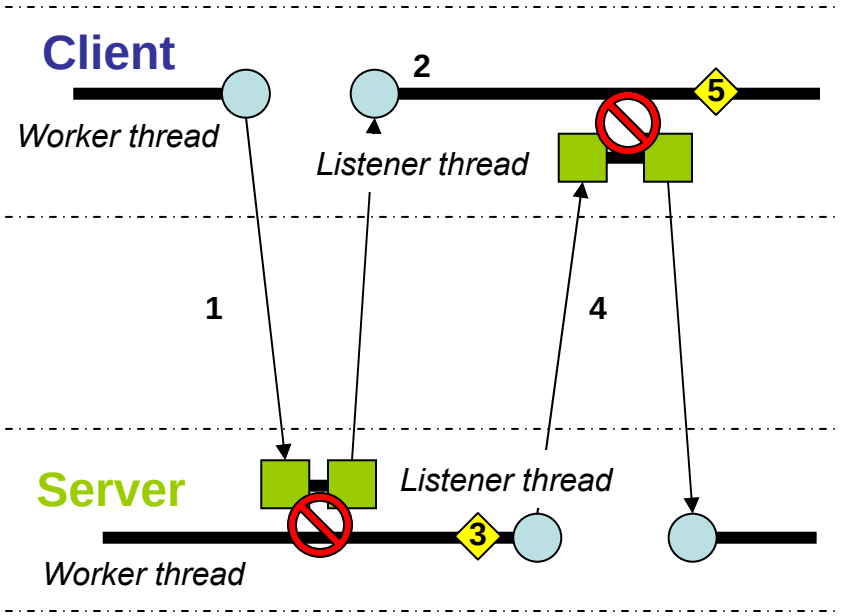
An asynchronous and symmetric Client/Server Model

Remote Procedure Call (RPC)

Extend the procedure call over the network by allowing programs to call procedures located on other machines

1. The client calls the server to place a service request
2. While the server processes the request, the client keeps executing
3. The server sees the request, and responds to it
4. The server calls back to the client with the result
5. The client sees the response and acts upon it

+ Higher parallelism
- Higher complexity



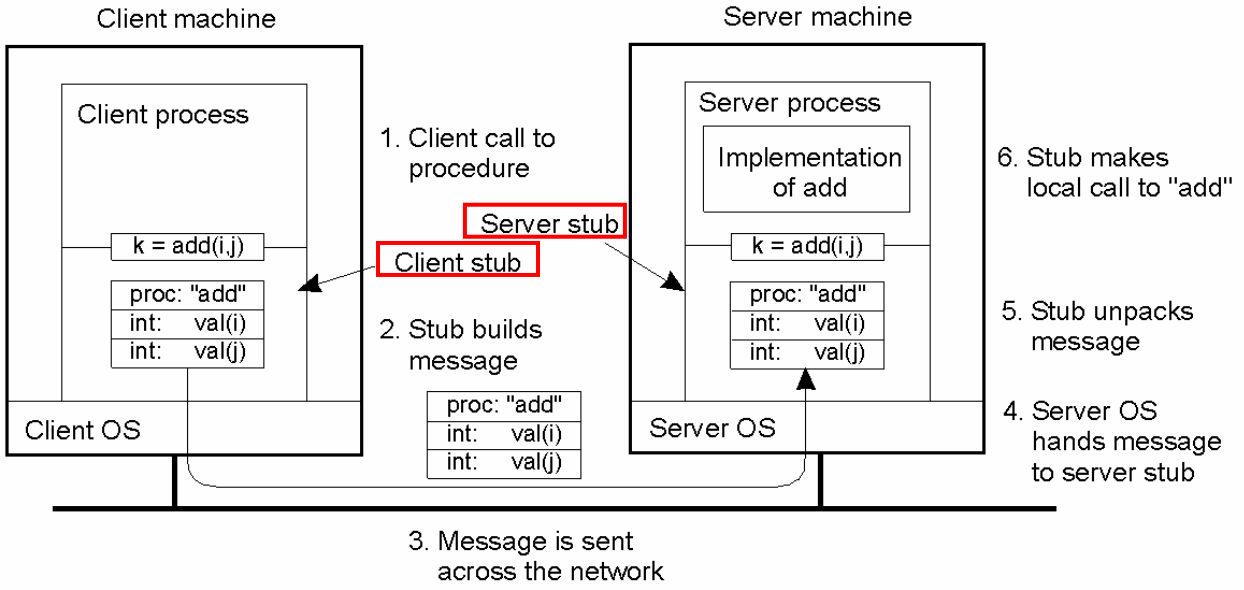
An asynchronous and symmetric Client/Server Model

Client & Server Stubs

The Stubs take charge of:

1) Building the RPC message (parameters and results), also called marshaling and unmarshaling

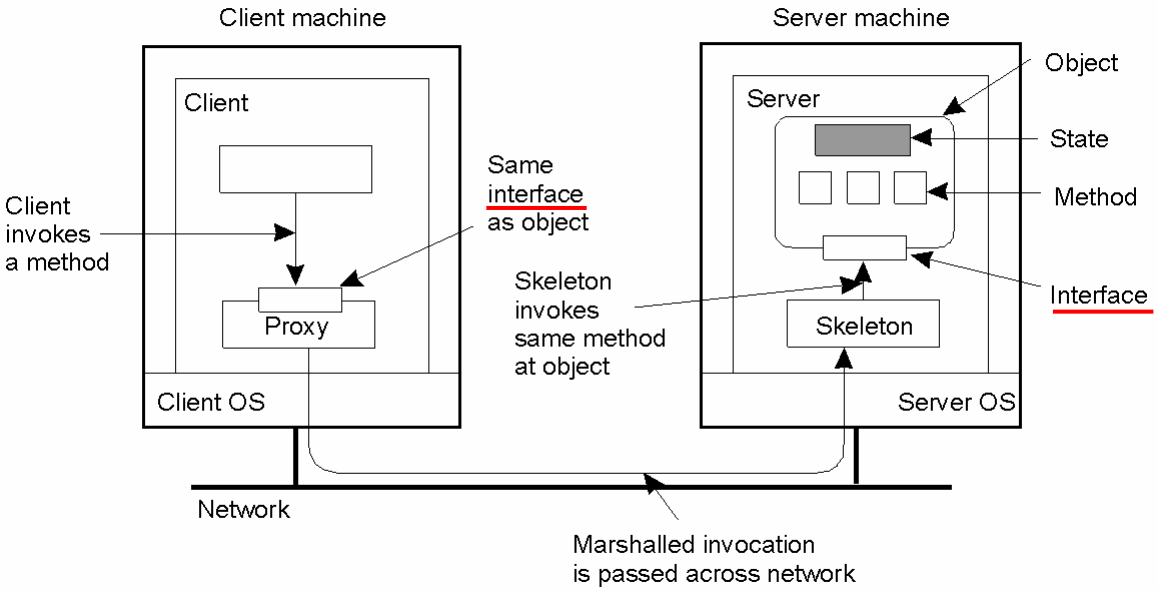
2) Establishing the connection to transfer messages.



Remote Method Invocation

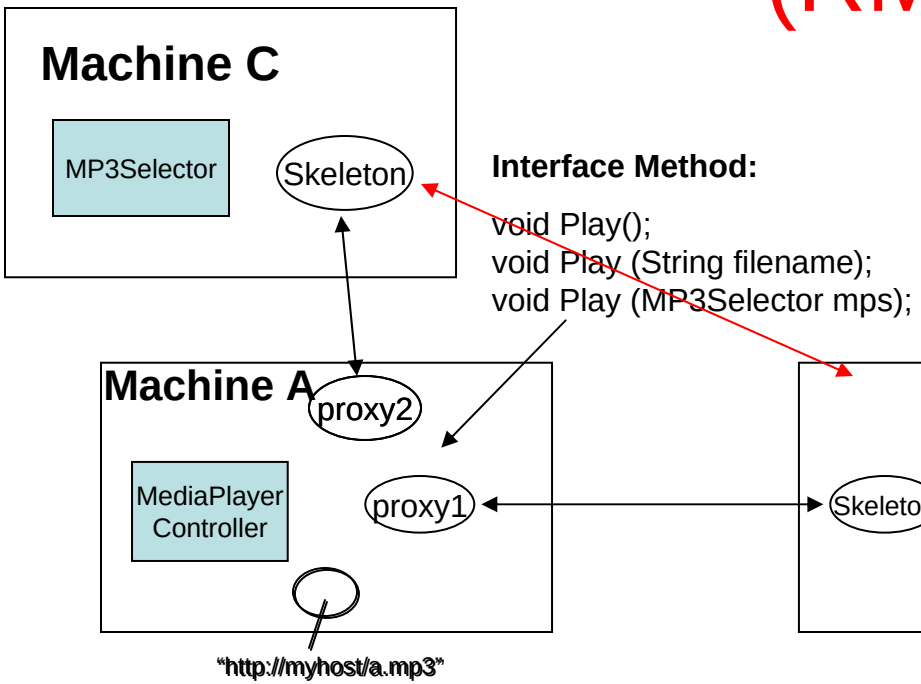
Java

- Object-oriented technology encapsulates data, (**state/Property)** and operations (**method**) on those data
- This encapsulation offers a better transparency for system design and programming
- The principle in RPC can be equally applied to objects



- Client uses **proxy** (a local representative of the remote object) to operate with the remote one.
- Proxy/Skeleton is analog to the stubs in RPC, in addition, it presents an object view.

Passing Object by Value or Reference (RMI)



- The object to send to another machine has to be *Serializable*.
- The object to send has to have its class definition in a publicly accessible *codebase*.
- Setup security policy file

Three cases:

1) Play () without parameters. // only method name will be sent

2) Play ("http://myhost/a.mp3") // send filename as a copied object (value/copy)

3) Play (mps) { // send the copy of proxy2
play(mps.getLatestMP3()) // (reference to MP3Selector)
}

http://java.sun.com/developer/onlineTraining/rmi/RMI.html

Conclusion (RPC & RMI)

- To be able to access a remote object, a local stub (proxy) which refers to the remote object is required.
- The stub appears as a local object, but delivers the received accesses to the remote object.
- The stub can be passed (e.g. in Java RMI) to other programs (on remote computers) to share the access to the same remote object.

Conclusion (RPC & RMI)

- Another way to access a remote object is to make a cloned local copy.
- This improves performance by removing the call delay over the network, but …
- Consistency becomes an issue if they need to be synchronized since they are now two independent objects (from the same class) in the network.

Conclusion (RPC & RMI)

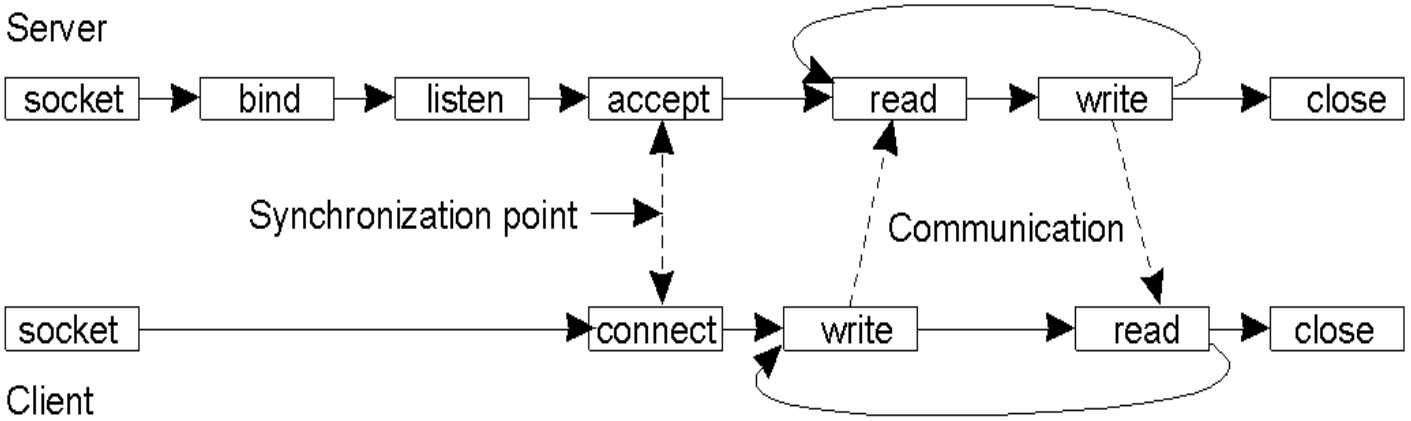
- Stubs, Proxies and Skeletons …
  - hides the complexity of marshaling and unmarshaling.
  - hides the network communication
  - enhances the access transparency to the upper-layer applications.

Conclusion (RPC & RMI)

- RPC and RMI use a transient synchronous communication model:
  - The sender blocks until it receives a reply from the other side.
  - This model is not suitable for pervasive computing scenarios where time is critical.

Berkeley Sockets

TCP/UDP Network communication like plug-in sockets



Connection-oriented communication (TCP) pattern using sockets.

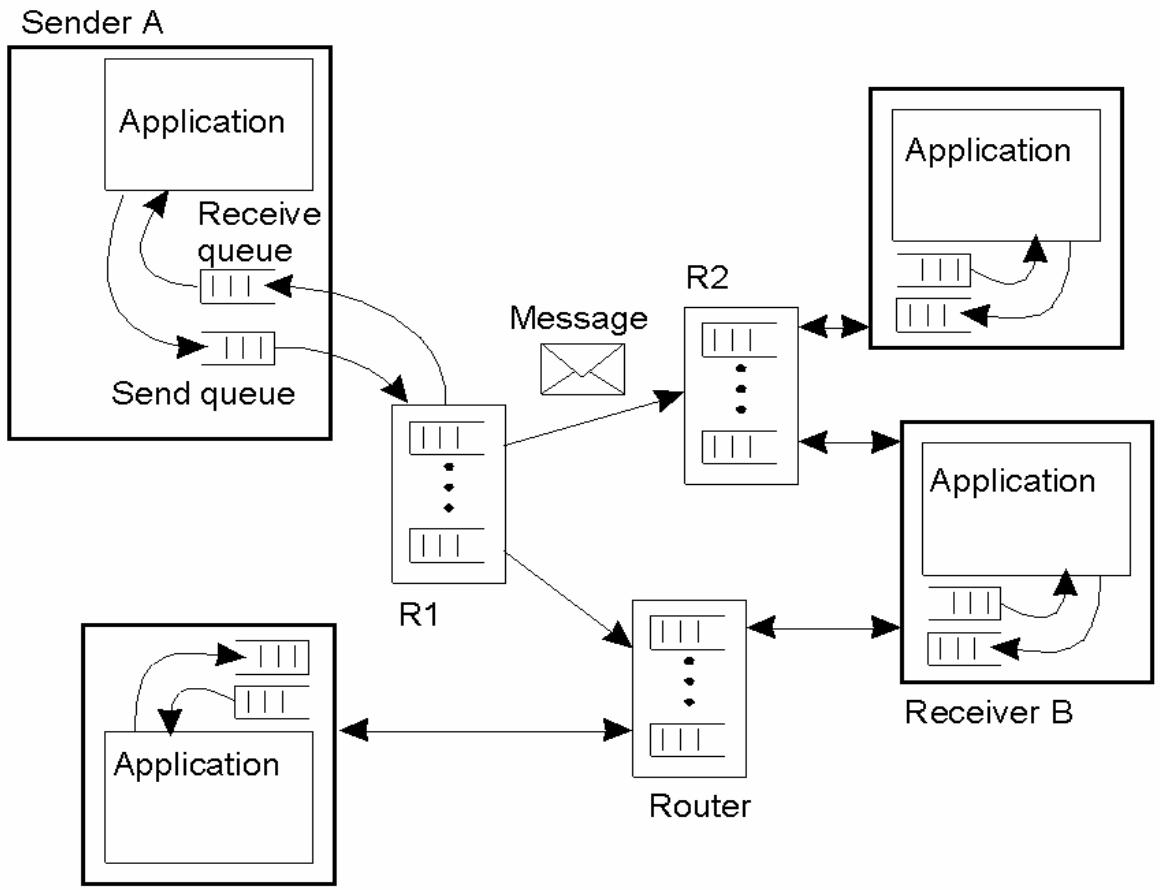
- UDP communication is asynchronous, so does not have the synchronization point as in TCP
- UDP server just creates a UDP socket and then receives (blocking), and UDP client has no “connect” phase to block, but just sends.
- UDP port =/= TCP port, they may use the same port number without conflict

Message-Oriented Middleware

- Socket communication gives an easy-to-use abstraction to network programming.
- Sockets are supported by most programming languages and operating systems supporting networks.
- To achieve efficiency and simplicity, many middlewares are implemented in terms of message delivery based on (hidden) socket communication.
- This is called Message-oriented middleware (MOM).
- *Examples: IBM MQSeries, Tuple Space, JavaSpace.*

General Architecture of a Message-Queuing System

- Messages are delivered in a sorting-storing-forwarding fashion
- Applications are loose-coupled by asynchronous messages (events)
- R1, R2 are Message Servers in MOM
- In email systems, R1, R2 are email servers



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

Message-Oriented Communication

time	Synchronous	Asynchronous
communication	Transient	Persistent

Summary

- OSI Model & Middleware
- Remote Procedural Call (RPC)
- Remote Method Invocation (RMI)
- Sockets
- Message Oriented Middleware

Distributed systems and their properties

(Lecture 2 for Programming of Interactive Systems)

Fredrik Kilander & Wei Li