Communication in Distributed Systems – Fundamental Concepts

Lecture Overview

- Understanding basic terminologies in communication in distributed systems

- Understanding key concepts in communication in distributed systems

Outline

- Communication entities, paradigm, roles/responsibilities
- Key issues in communication in distributed systems
- Protocols
- Processing requests
- Summary

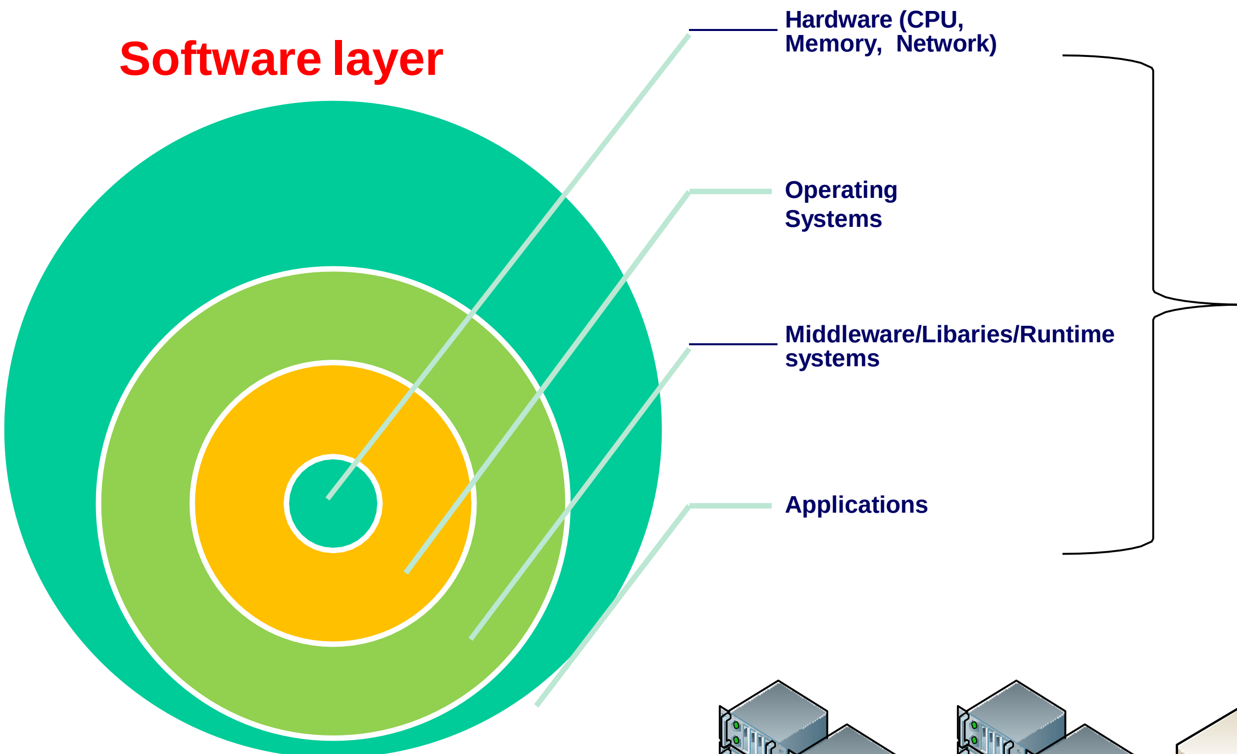
Learning Material

- Main reading:
  - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair„Distributed Systems – Concepts and Design“, 5nd Edition
    - Chapters 2,3, 7.
  - Craig Hunt, TCP/IP Network Administration, 3edition, 2002, O‘Reilly.

COMMUNICATION ENTITIES, PARADIGM, AND ROLES

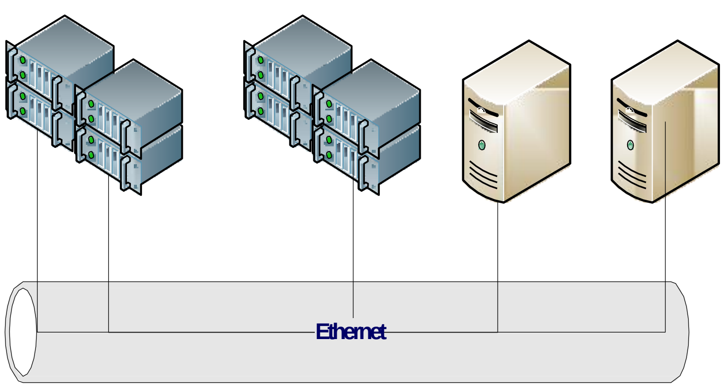
Hardware, software layer, programs

Software layer



Programs/Programming Languages

- C/C++/Java, Python, …
- Different types of programs: systems versus applications; sequential versus parallel ones; clients versus servers/services

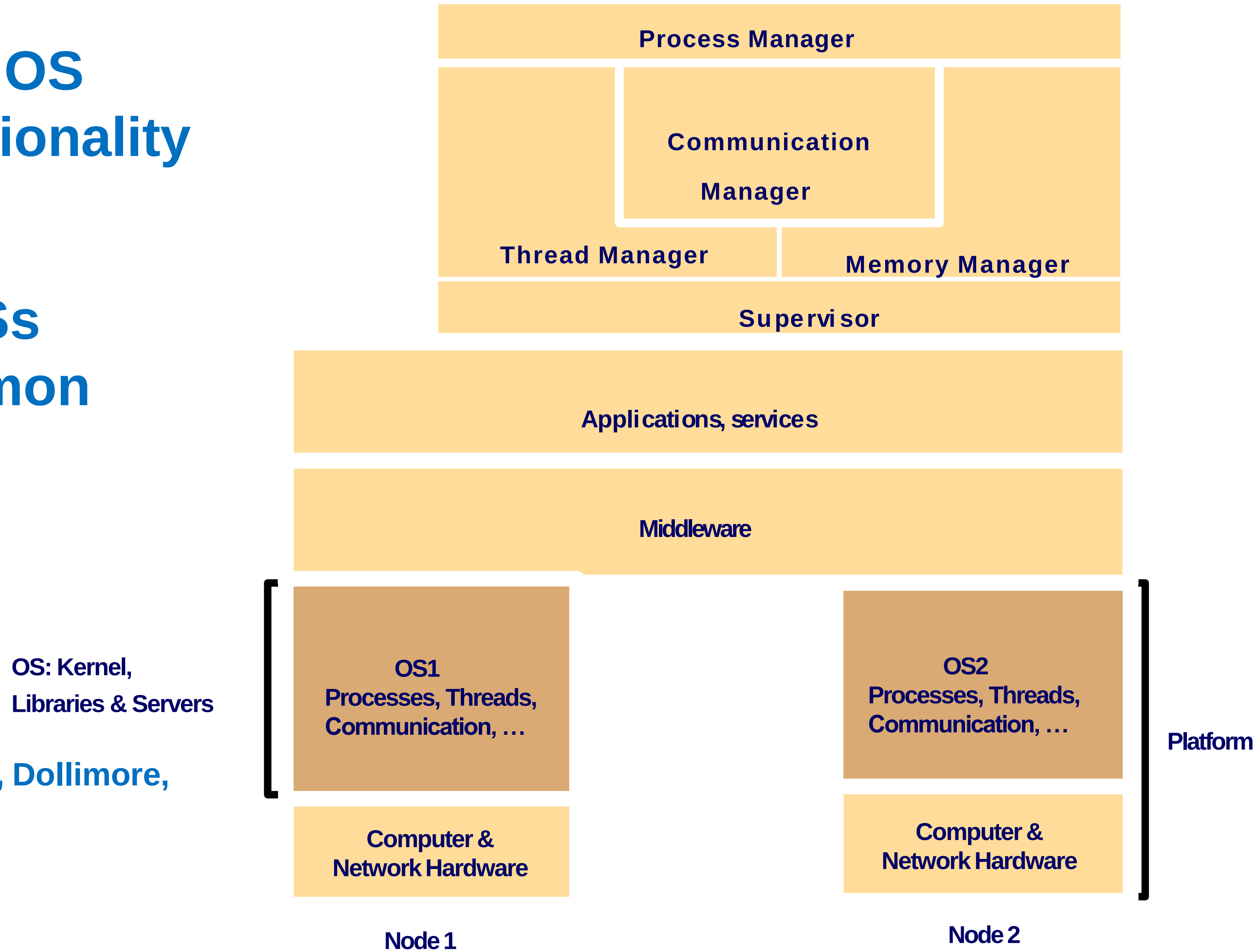


Hardware heterogeneity

Systems layers and Core OS functionality

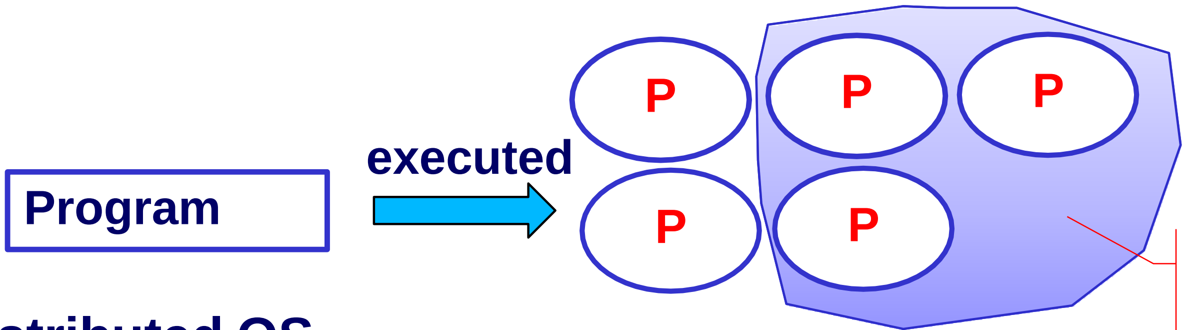
Core OS functionality

Different OSs with a common middleware layer



Source: Coulouris, Dollimore, Kindberg

Process vs Thread

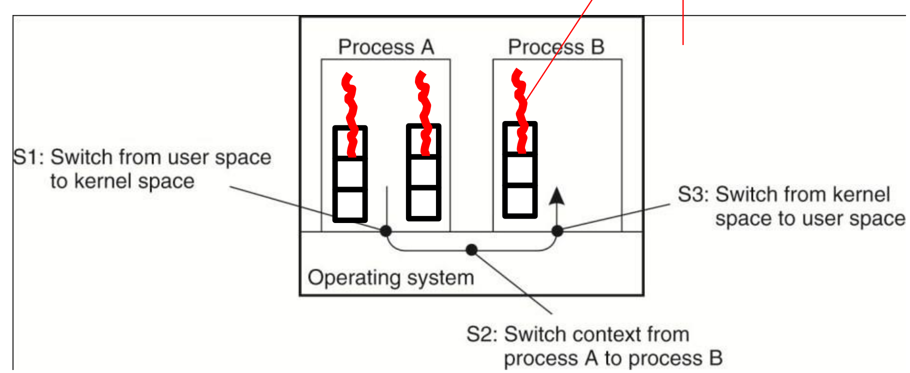


Distributed processes of the same service (coded in the same program)

Within a non distributed OS

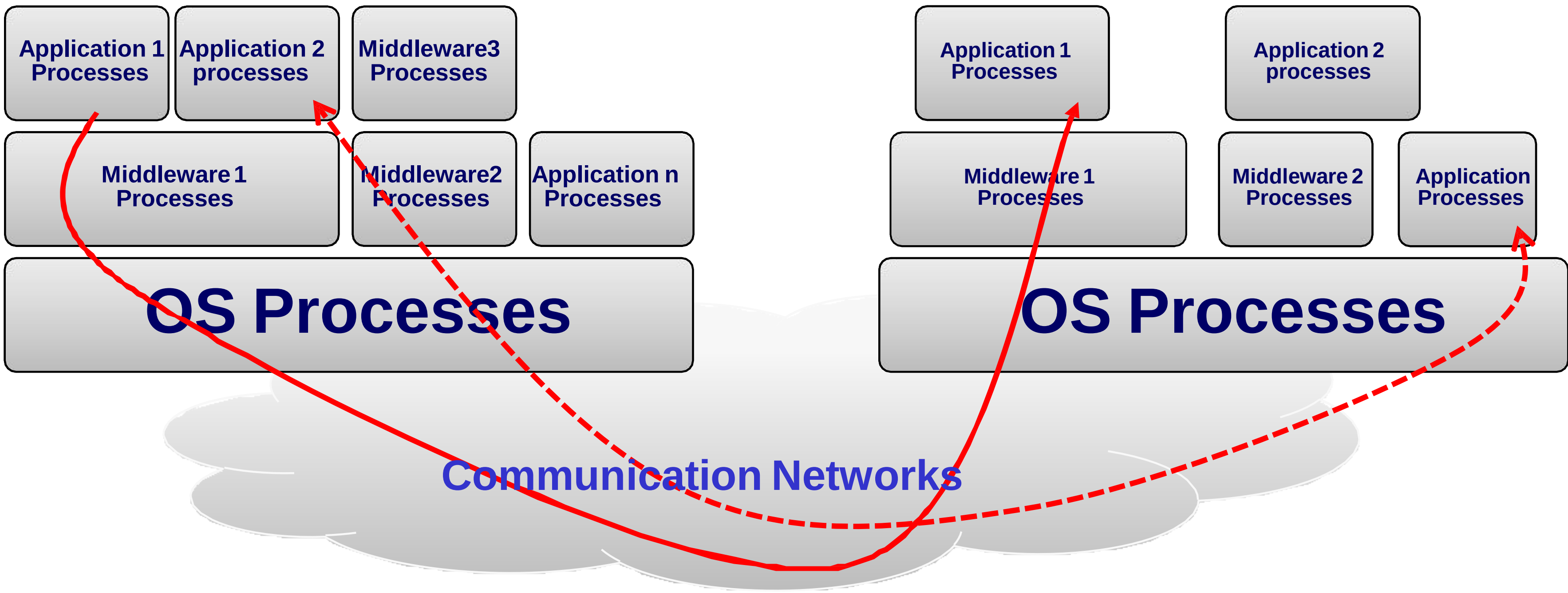
- Process – the program being executed by the OS
- Threads within a process
- Switching thread context is much cheaper than that for the process context
- Blocking calls in a thread do not block the whole process

Thread of execution



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

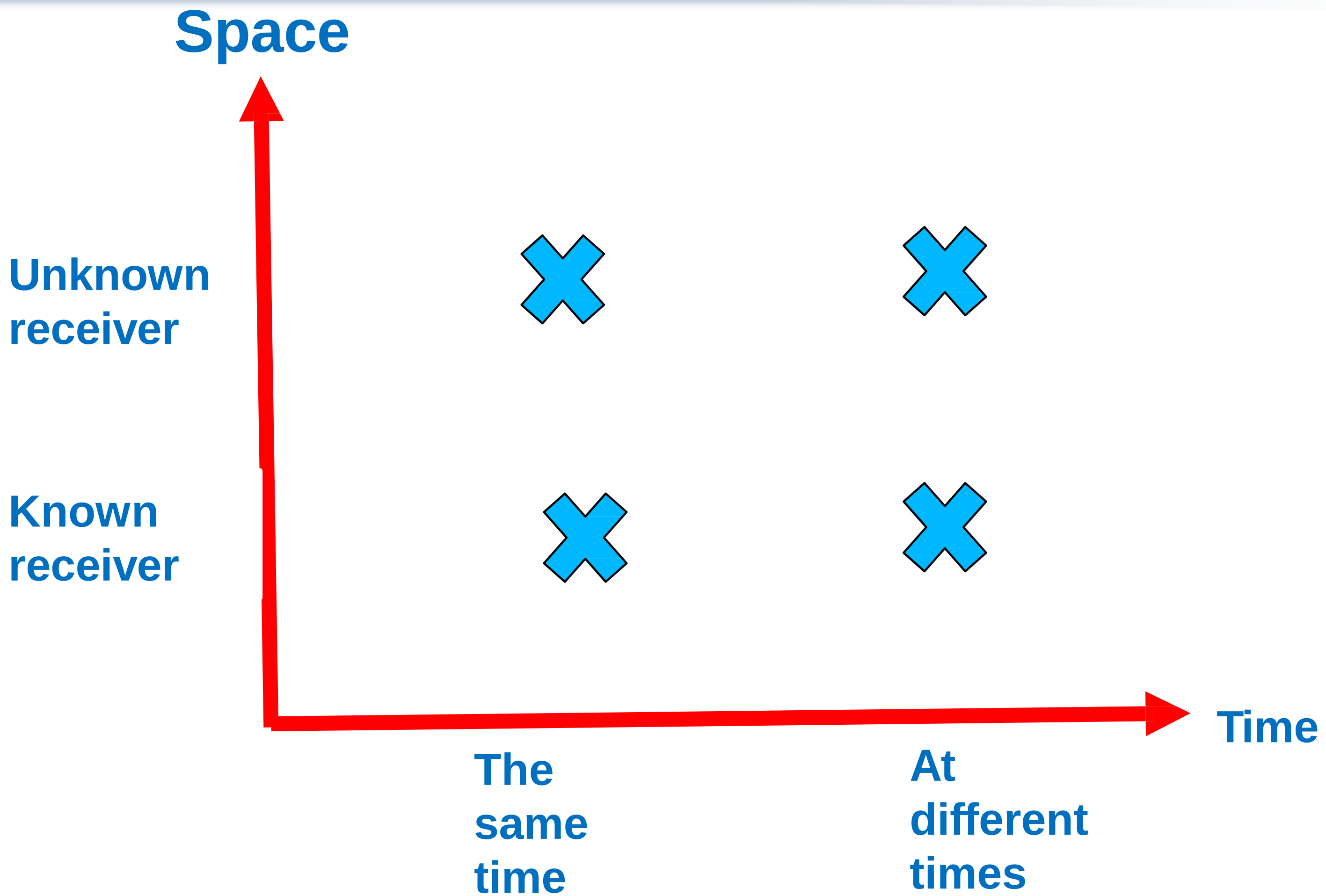
Communication entities



Communication in distributed systems

- between processes within a single application/middleware/service
- among processes belonging to different applications/middleware/services
- Among computing nodes which have no concept of processes (e.g. sensors)

Space and Time in Communication



Q: why is understanding time and space uncoupling important for implementing communication in distributed systems?

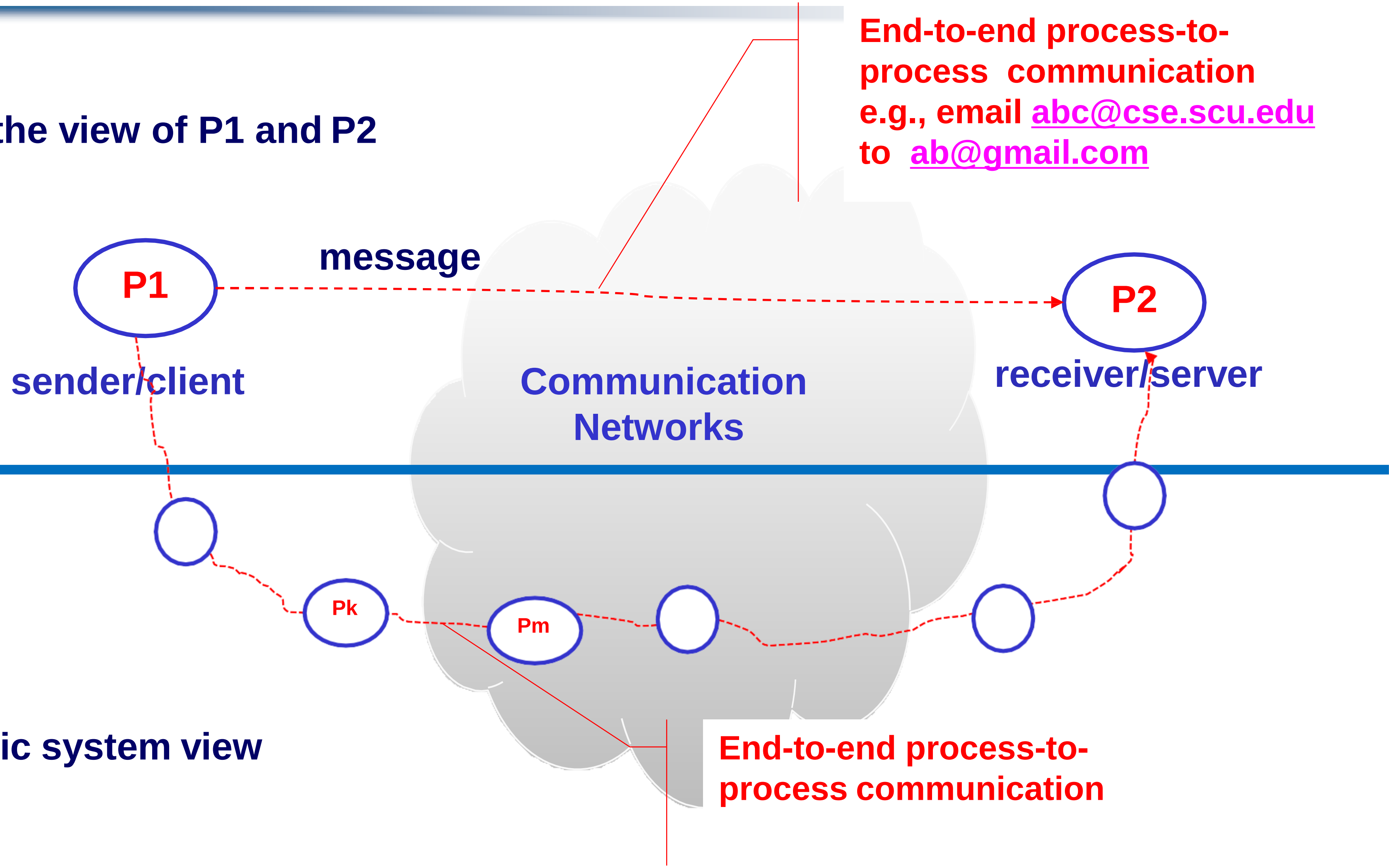
Communication networks in Distributed Systems

- **Maybe designed for specific types of environments**
  - High performance computing, M2M (Machine-to-Machine), building/home/city management, etc.
  - Events, voices, documents, image data, etc.
- **Distributed, different network spans**
  - Personal area networks (PANs), local area networks (LANs), campus area networks (CANs), metropolitan area networks (MANs), and wide area networks (WANs)
  - Communication entities are placed in different locations
- **Different layered networks for distributed systems**
  - Physical versus overlay network topologies (virtual network topologies atop physical networks)

Layered communication

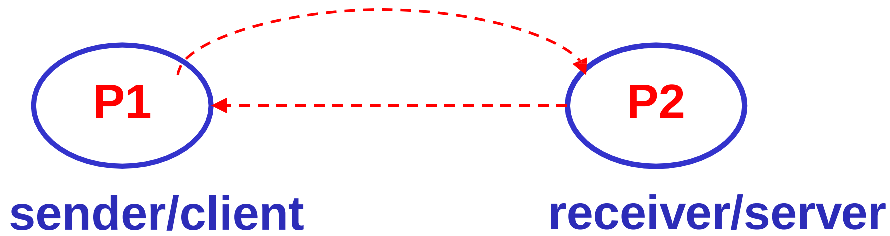
In the view of P1 and P2

Holistic system view

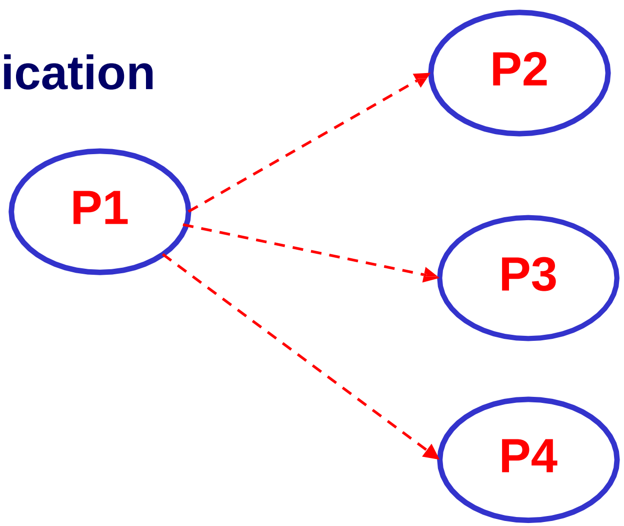


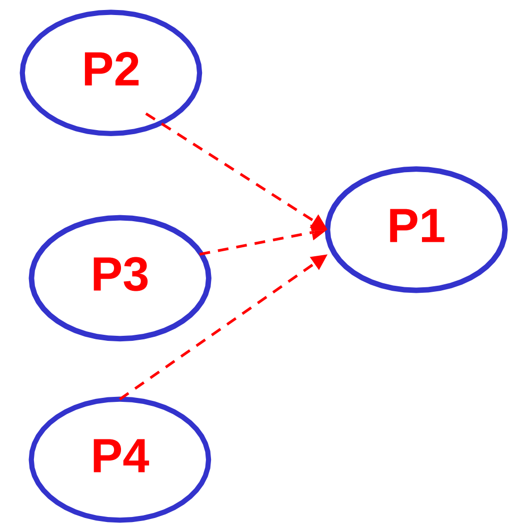
Communication Patterns

One-to- one/client-server



Group communication



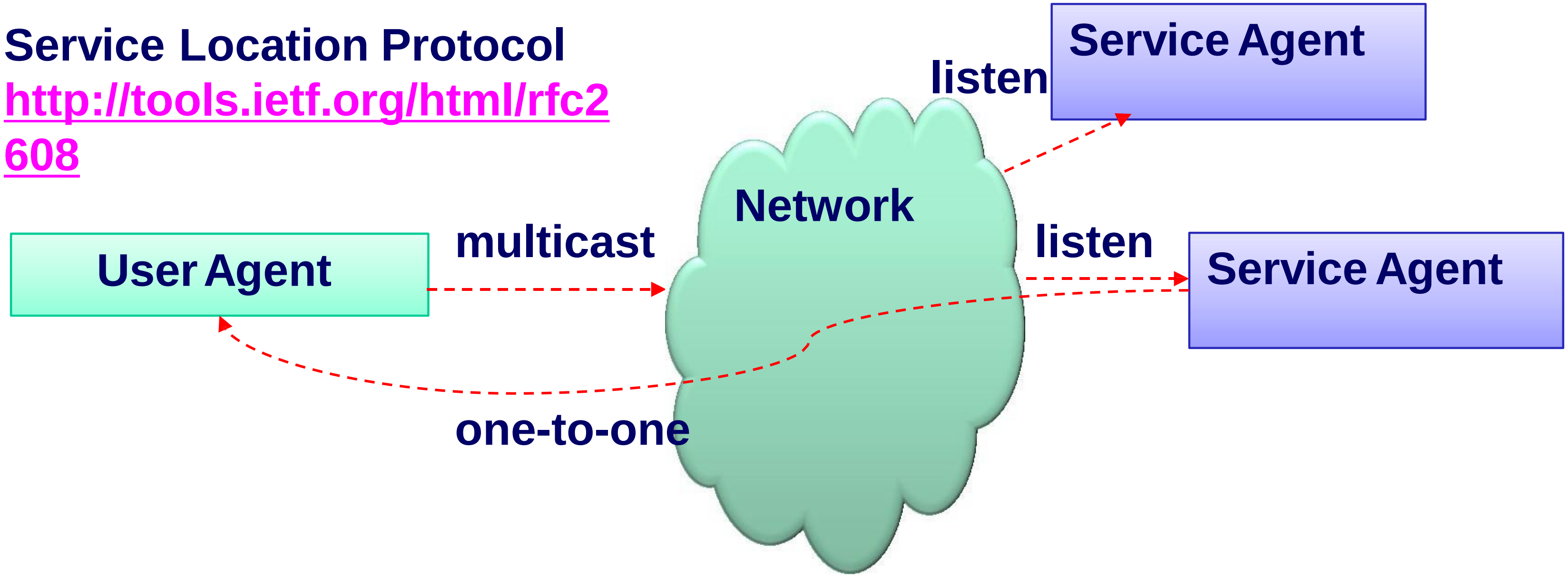


Identifiers of entities participating in communication

- Communication cannot be done without knowing
- identifiers (names) of participating entities
  - Local versus global identifier
  - Individual versus group identifier
- Multiple layers/entities -> different forms of identifiers
  - Process ID in an OS
  - Machine ID: name/IP address
  - Access point: (machine ID, port number)
  - A unique communication ID in a communication network
  - Emails for humans
  - Group ID

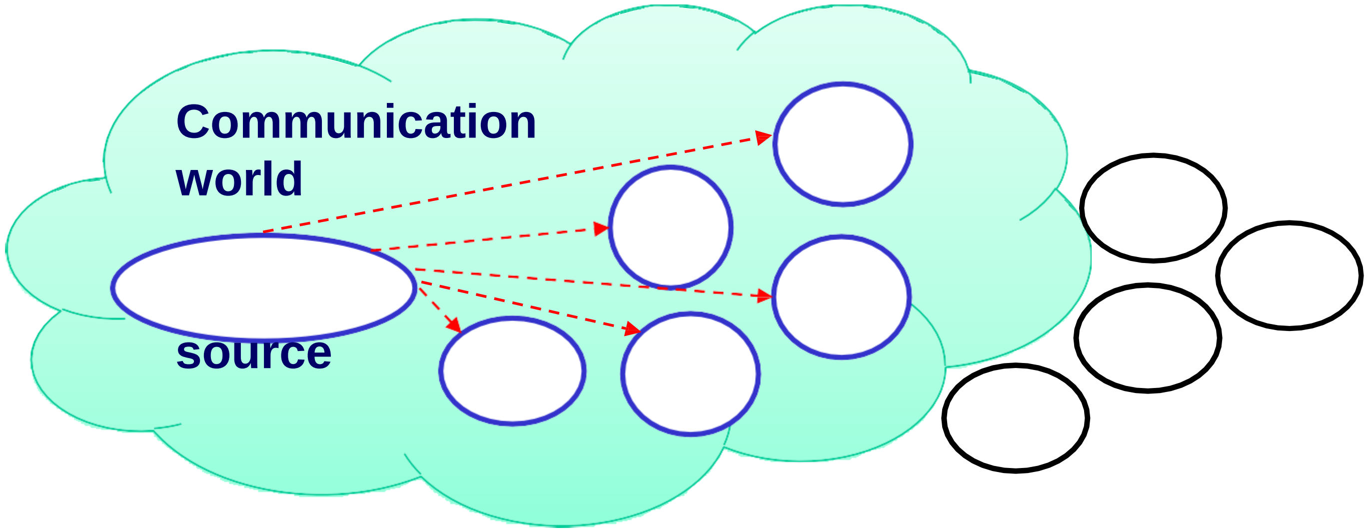
Examples of communication patterns (I)

Service Location Protocol
http://tools.ietf.org/html/rfc2608



- A User Agent wants to find a Service Agent
- Different roles and different communication patterns
- Get http://jslp.sourceforge.net/ and play samples to see how it works

Examples of communication patterns (2)



- **MPI (Message Passing Interface)**

```
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   source=0;
   count=4;
   if(myid == source){
     for(i=0;i<count;i+
     +)  buffer[i]=i;
   }
   MPI_Bcast(buffer,count,MPI_INT,source,MPI_COMM_WORLD);
```

$sudo apt-get install mpich
$mpicc c_ex04.c
$mpirun –np 4 ./a.out

http://geco.mines.edu/workshop/class2/examples/mpi/c_ex04.c

Connection-oriented or connection less communication

P1

The message: “there is a party tonight“

P2

Write the message in a letter

Go to the post office

Send the letter to P2

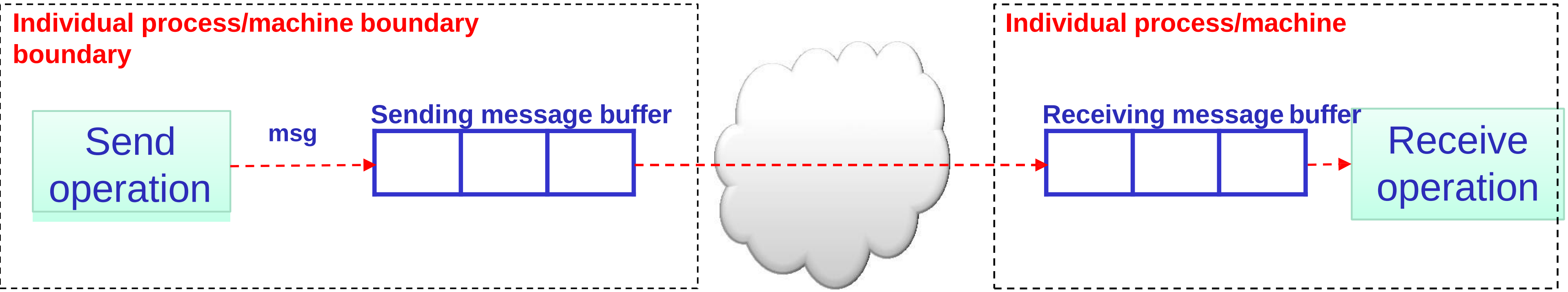
Find the phone number of P2

Call P2

Tell P2 the message

Connection-oriented communication between P1 and P2 requires the setup of communication connection between them first – no setup in connectionless communication

Blocking versus non-blocking communication calls



Send: transmitting a message is finished, it does not necessarily mean that the message reaches its final destination.

- Blocking: the process/thread execution is suspended until the message transmission finishes
- Non-blocking: the process/thread execution continues without waiting until the finish of the message transmission

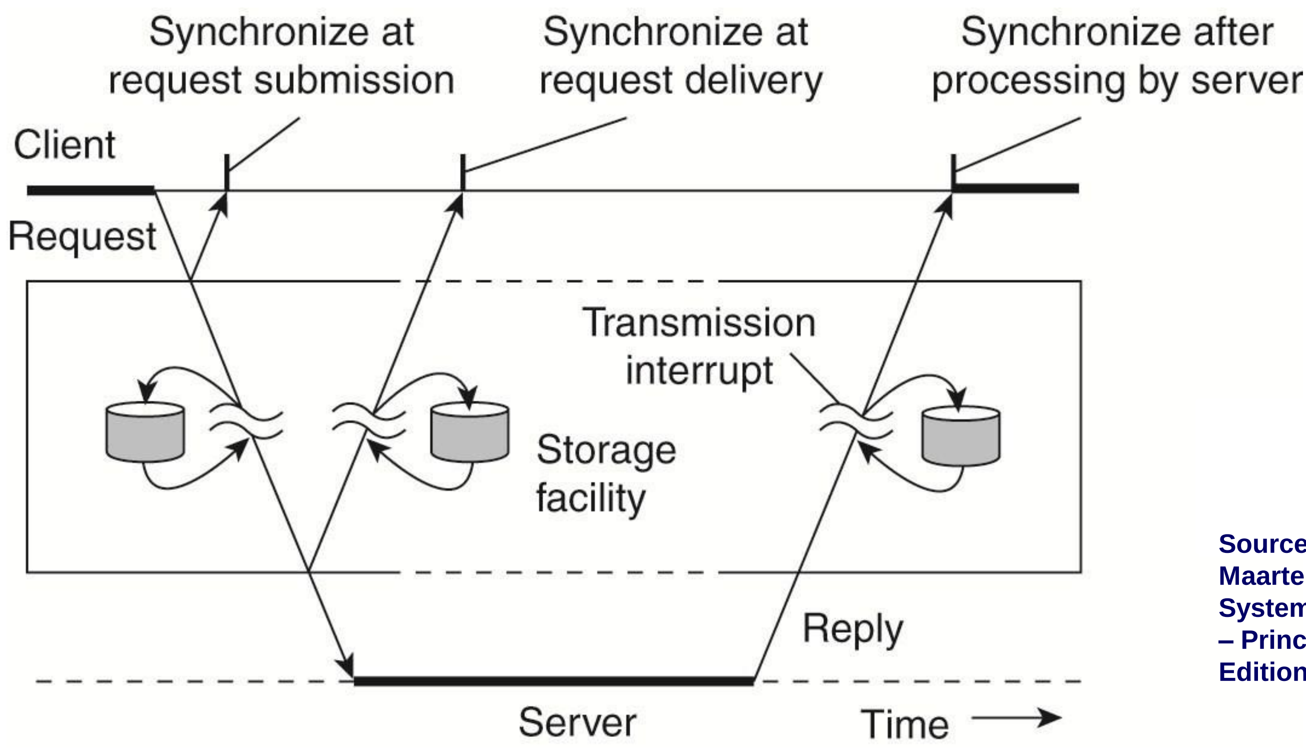
Persistent and transient communication

- Persistent communication
  - Messages are kept in the communication system until they are delivered to the receiver
  - Often storage is needed
- Transient communication
  - Messages are kept in the communication temporary only if both the sender and receiver are live

Asynchronous versus synchronous communication

- Asynchronous: the process continues after as soon as sending messages have been copied to the local buffer
  - Non blocking send; receive may/may not be blocking
  - Callback mechanisms
- Synchronous: the sender waits until it knows the messages have been delivered to the receiver
  - Blocking send/blocking receive
  - Typically utilize connection-oriented and keep-alive connection
  - Blocking request-reply styles

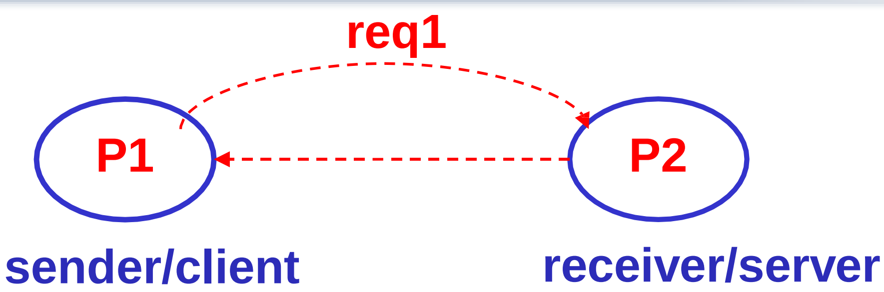
Different forms of communication



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

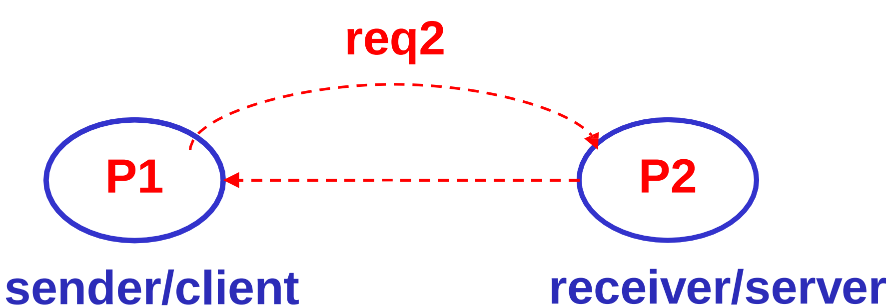
Stateful versus Stateless Server

T0



P2 has no information about P1 before

T1

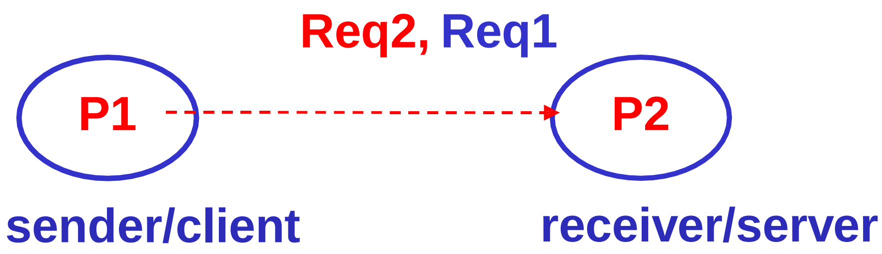


Does P2 keep information about P1 from the previous request?

Stateless server	Soft State	Stateful Server
Does not keep client's state information	Keep some limited client's state information in a limited time	Maintain client's state information permanently

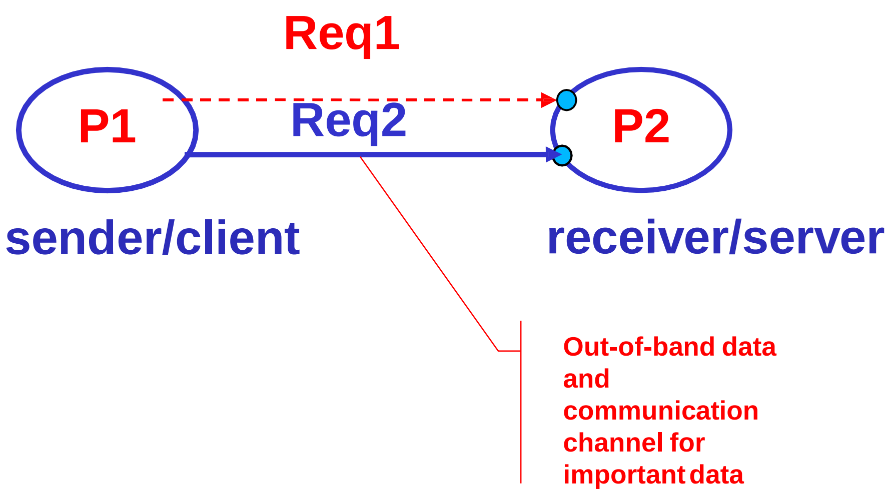
Handling of band data

Normal case



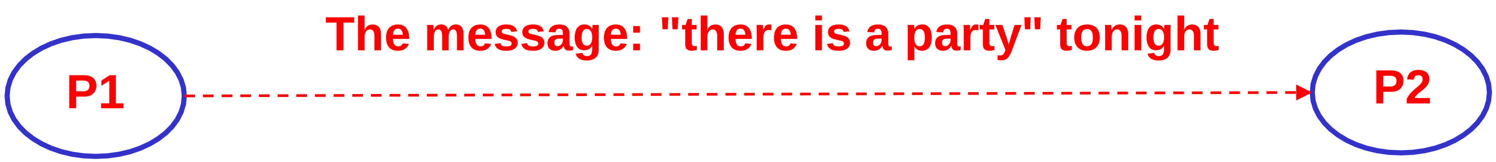
All messages come to P2 in the same port, no clear information about priority

Out of band data with a separate transmission port



COMMUNICATION PROTOCOLS

Some key questions - Protocols

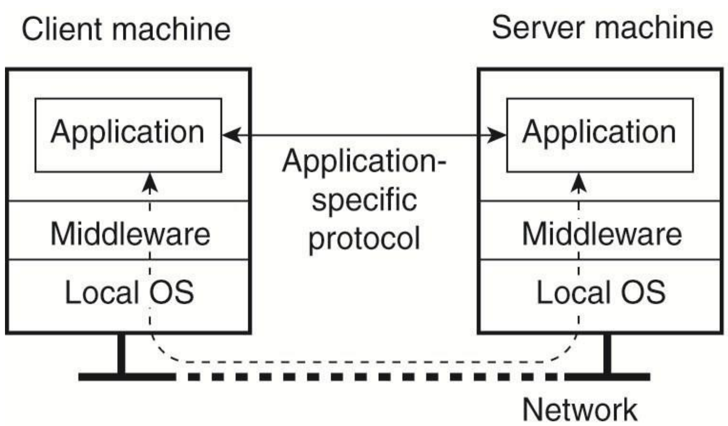


- Communication patterns
  - Can I use a single sending command to send the message to multiple people?
- Identifier/Naming/Destination
  - How do I identify the guys I need to send the message
- Connection setup
  - Can I send the message without setting up the connection
- Message structure
  - Can I use German or English to write the message
- Layered communication
  - Do I need other intermediators to relay the message?
- …

A communication protocol will describe rules addressing these issues

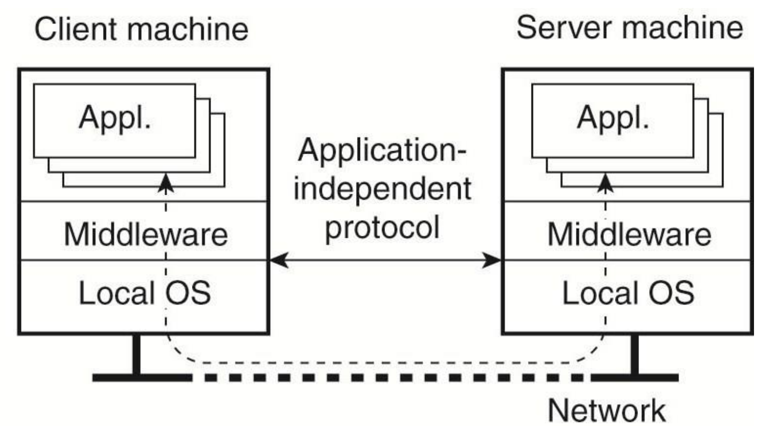
Applications and Protocols

Application-specific protocols



(a)

Application-independent protocols



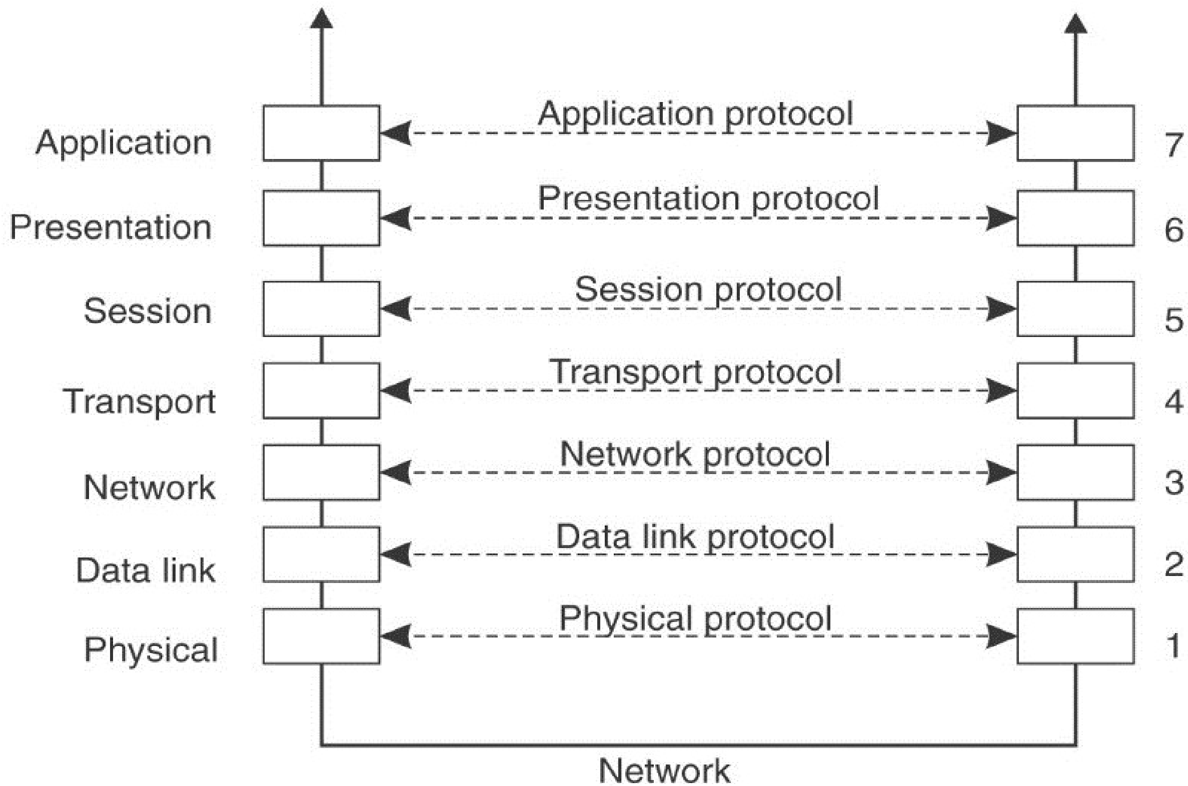
(b)

Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Layered Communication Protocols

- Complex and open communication requires multiple communication protocols
- Communication protocols are typically organized into differ layers: layered protocols/protocol stacks
- Conceptually: each layer has a set of different protocols for certain communication functions
    - Different protocols are designed for different environments/criteria
- A protocol suite: usually a set of protocols used together in a layered model

OSI – Open Systems Interconnection Reference Model



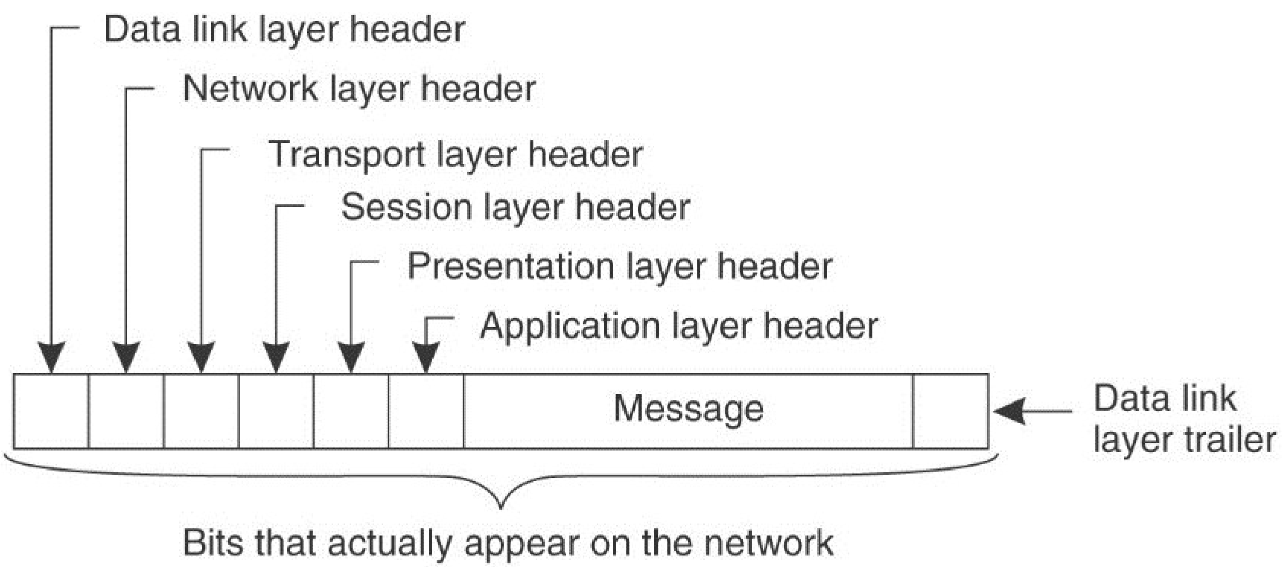
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

OSI Layers

Application	• Support application-specific needs
Presentation	• Process information format and deliver the information for the application layer (e.g., serializing and encryption)
Session	• Manage communication sessions between applications
Transport	• Provide an end-to-end communication for applications by delivering data among applications
Network	• Route data packets among senders/receivers
Data Link	• Deal with sending data frames (units of bits) and detecting/correcting data frames
Physical Layer	• Transfer binarydata (bits) over physical interfaces (e.g., fiber optics)

How layered protocols work – message exchange

- Principles of constructing messages/data encapsulation



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

TCP/IP

- The most popular protocol suite used in the Internet
- Four layers

	Protocol suite
Application Layer	SMTP, HTTP, Telnet, FTP, etc.
Transport Layer	UDP, TCP
Internet Layer	Internet Protocol (IP)
Link Layer	Most network hardware

http://tools.ietf.org/html/rfc1122

Internet Protocol (IP)

- Defines the datagram as the basic data unit
- Defines the Internet address scheme
- Transmits data between the Network Access Layer and Transport Layer
- Routes datagrams to destinations
- Divides and assembles datagrams

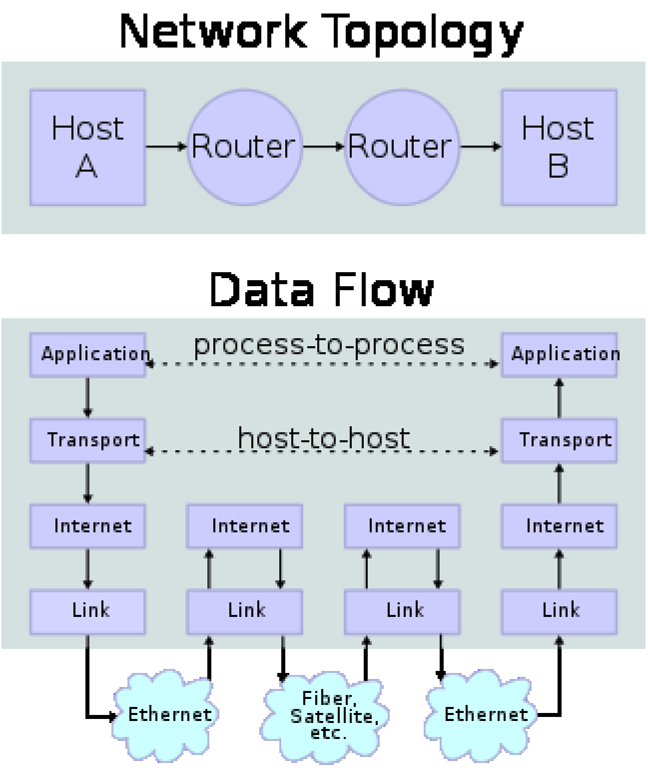


Figure source: http://en.wikipedia.org/wiki/Internet_protocol_suite

TCP/IP – Transport Layer

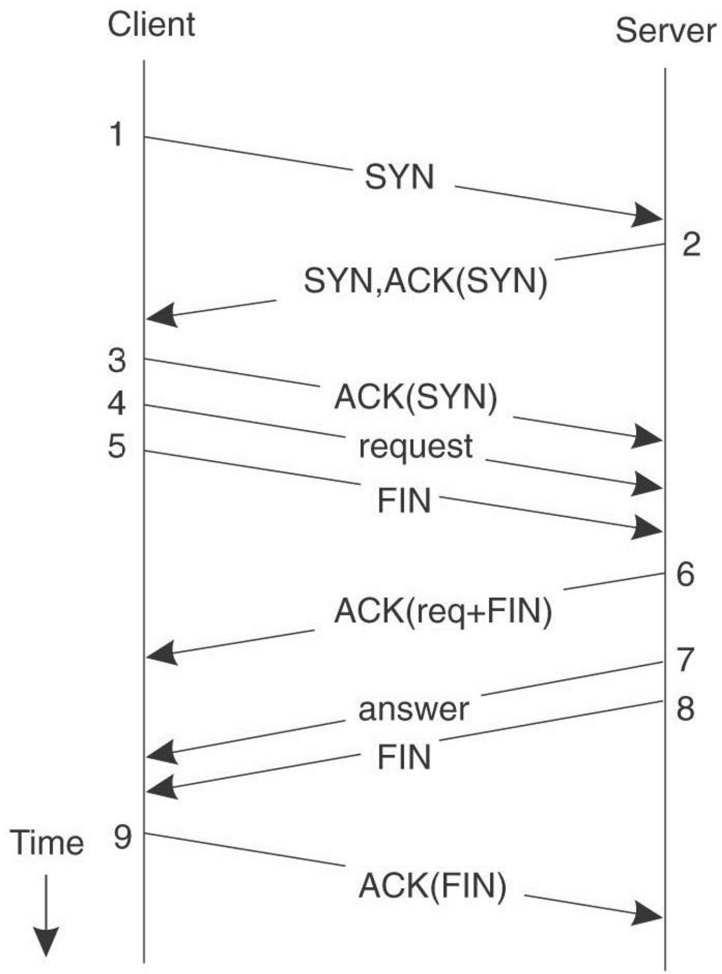
- Host-to-host transport features
- Two main protocols: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol)

Layer\Protocol	TCP	UDP
Application layer	Data sent via Streams of bytes	Data sent in Messages
Transport Layer	Segment	Packet
Internet Layer	Datagram	Datagram
Link Layer	Frame	Frame

Segment is the original data + Transport Layer header.
Packet is a Segment + Network Layer header.
Frame is a Packet + Data Link Layer header.

Note: pay attention with the terms, packet/datagram“ in TCP/IP versus that in the OSI model

TCP Operations



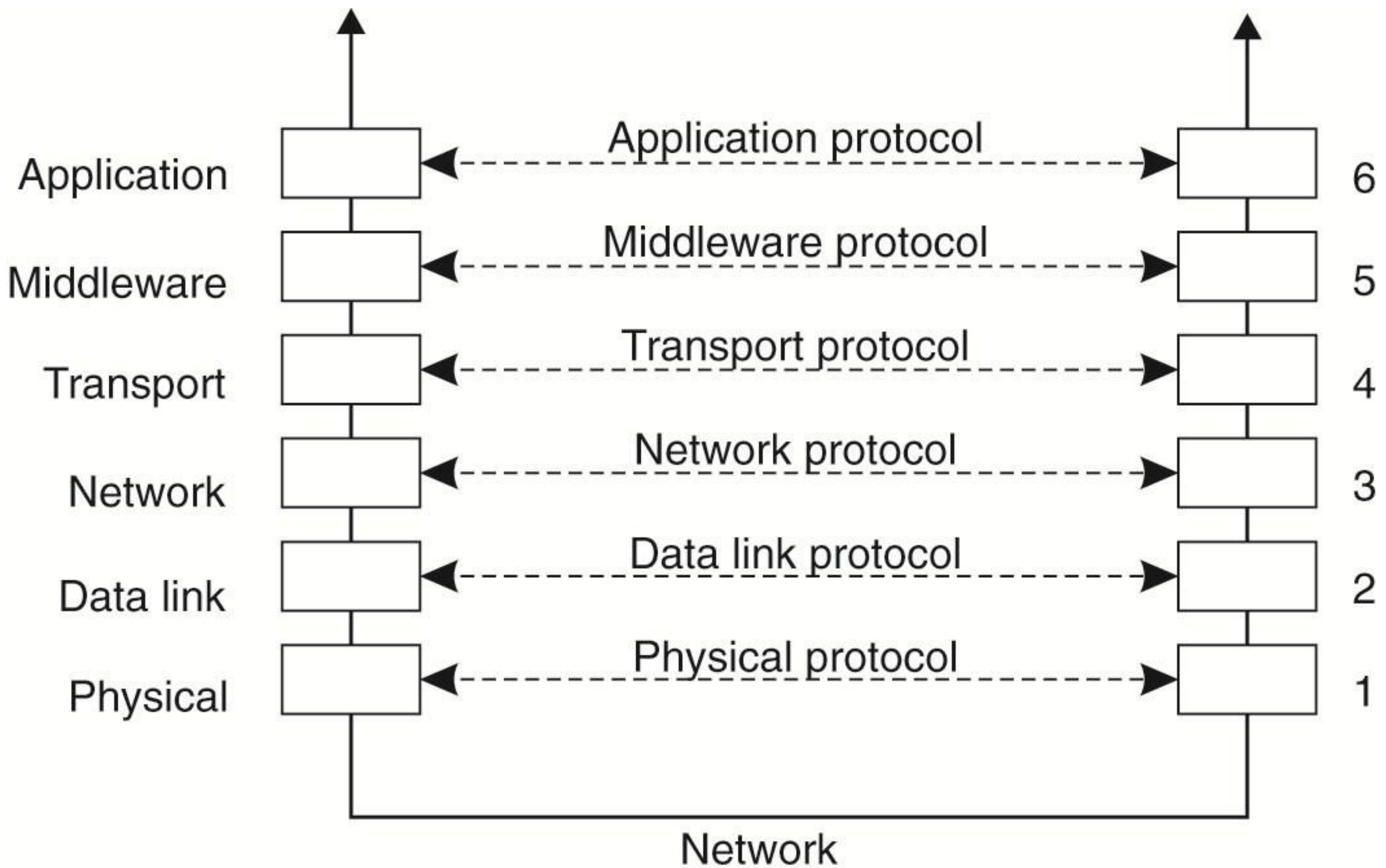
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2002, Prentice-Hall, Inc.

SYN: Synchronize
TCP packet sent to server requesting that a connection be established
ACK(SYN)
If the SYN is received by the server, an SYN/ACK is sent back to the address requested by the SYN

Communication protocols are not enough

- We need more than just communication protocols
  - E.g., resolving names, electing a communication coordinator, locking resources, and synchronizing time
- Middleware
  - Including a set of general-purpose but application-specific protocols, middleware communication protocols, and other specific services.

Middleware Protocols



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

HANDLING COMMUNICATION MESSAGES/REQUESTS

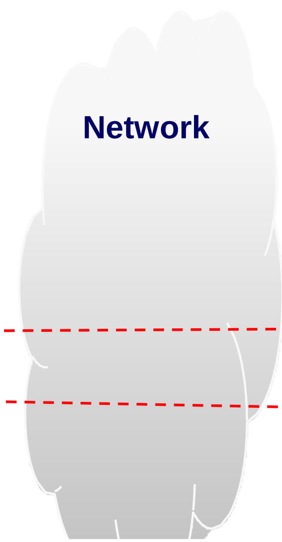
Where communication takes place?

- Message passing – send/receive
  - Processes send and receive messages
    - Sending process versus receiving process
    - Communication is done by using a set of functions for communication implementing protocols
- Remote method/procedure calls
  - A process calls/invokes a (remote) procedure in another process
    - Local versus remote procedure call, but in the same manner
- Remote object calls
  - A process calls/invokes a (remote) object in another process

Basic send/receive communication

```
# Echo client program
import socket

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)
```




```
# Echo server program
import socket

HOST = ''                 # Symbolic name meaning the local host
PORT = 50007              # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

Python source: http://docs.python.org/release/2.5.2/lib/socket-example.html

Remote procedure calls (RPC)

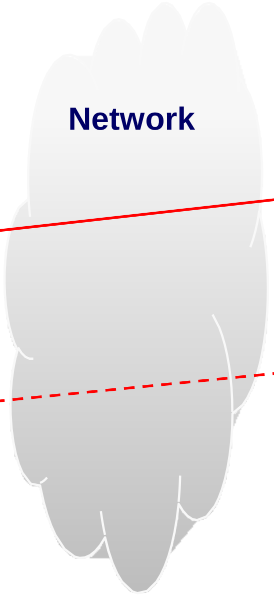
```
void
hello_prog_1(char *host)
{
        CLIENT *clnt;
        char * *result_1;
        char *hello_1_arg;

#ifndef   DEBUG
        clnt = clnt_create (host, HELLO_PROG, HELLO_VERS, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif    /* DEBUG */

        result_1 = hello_1((void*)&hello_1_arg, clnt);
        if (result_1 == (char **) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef   DEBUG
        clnt_destroy (clnt);
#endif     /* DEBUG */
     printf("result is: %s\n",(*result_1));
}


int
main (int argc, char *argv[])
{
        char *host;

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        hello_prog_1 (host);
exit (0);
}
```



Procedure in a remote server

```
char **
hello_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result ="Hello";

        /*
         * insert server code here
         */

        return &result;
}
```

Remote procedure calls (RPC)

Objects in a remote server

```
public class ComputePi {
   public static void main(String args[]) {
      if (System.getSecurityManager() == null) {
         System.setSecurityManager(new SecurityManager());
      }
      try {
         String name = "Compute";
         Registry registry = LocateRegistry.getRegistry(args[0]);
         Compute comp = (Compute) registry.lookup(name);
         Pi task = new Pi(Integer.parseInt(args[1]));
         BigDecimal pi = comp.executeTask(task);
         System.out.println(pi);
      } catch (Exception e) {
         System.err.println("ComputePi exception:");
         e.printStackTrace();
      }
   }
}
```

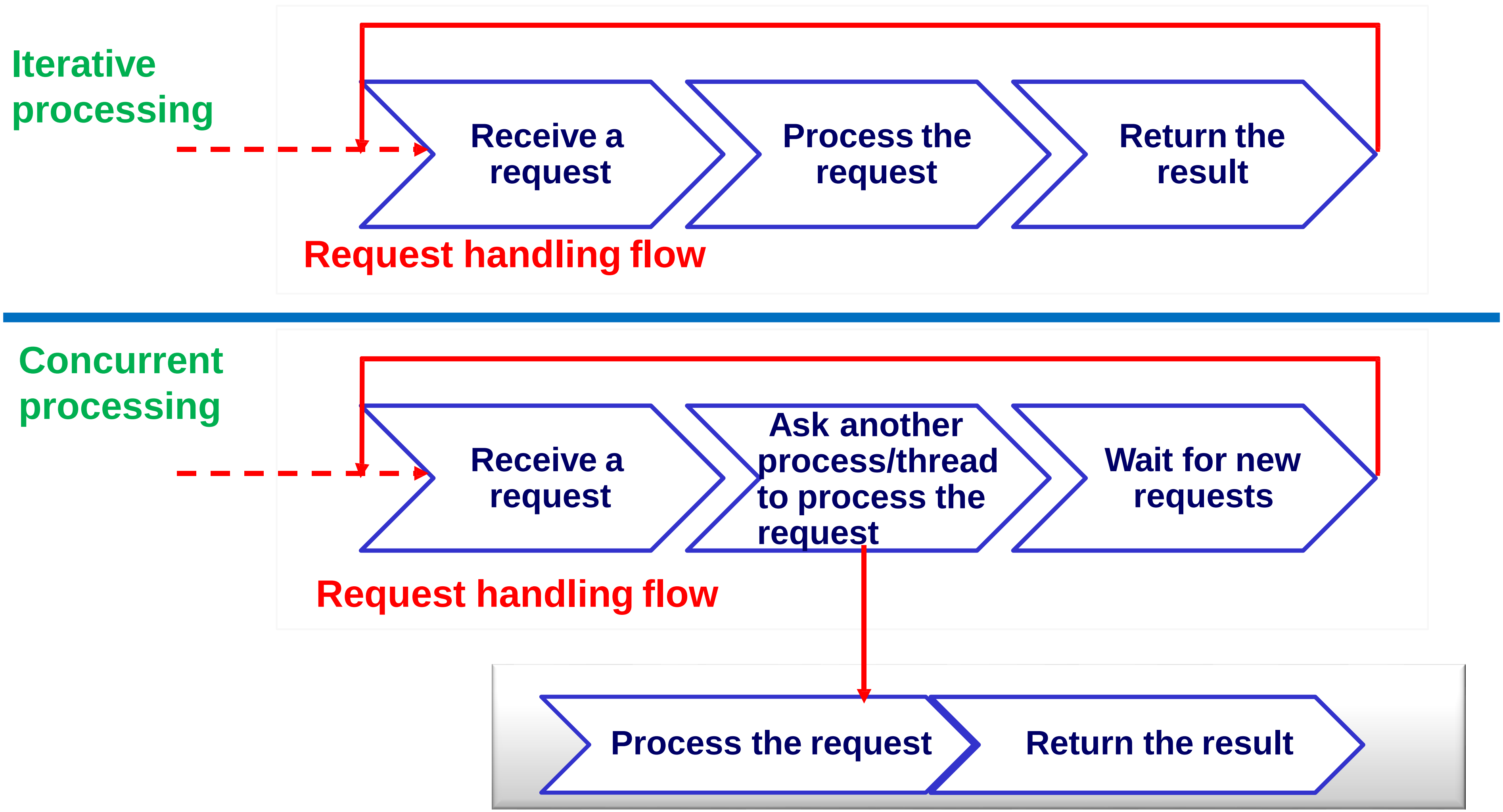
```
public interface Compute extends Remote {
<T> T executeTask(Task<T> t) throws RemoteException;
}
….
public class ComputeEngine implements Compute {
   public ComputeEngine() {
         super();
   }
   public <T> T executeTask(Task<T> t) {
         return t.execute();
   }
   public static void main(String[] args) {
      if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
      }
      try {
         String name = "Compute";
         Compute engine = new ComputeEngine();
         Compute stub =
            (Compute) UnicastRemoteObject.exportObject(engine, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("ComputeEngine bound");
      } catch (Exception e) {
            System.err.println("ComputeEngine exception:");
            e.printStackTrace();
      }
}
```

Java Source: https://docs.oracle.com/javase/tutorial/rmi/overview.html

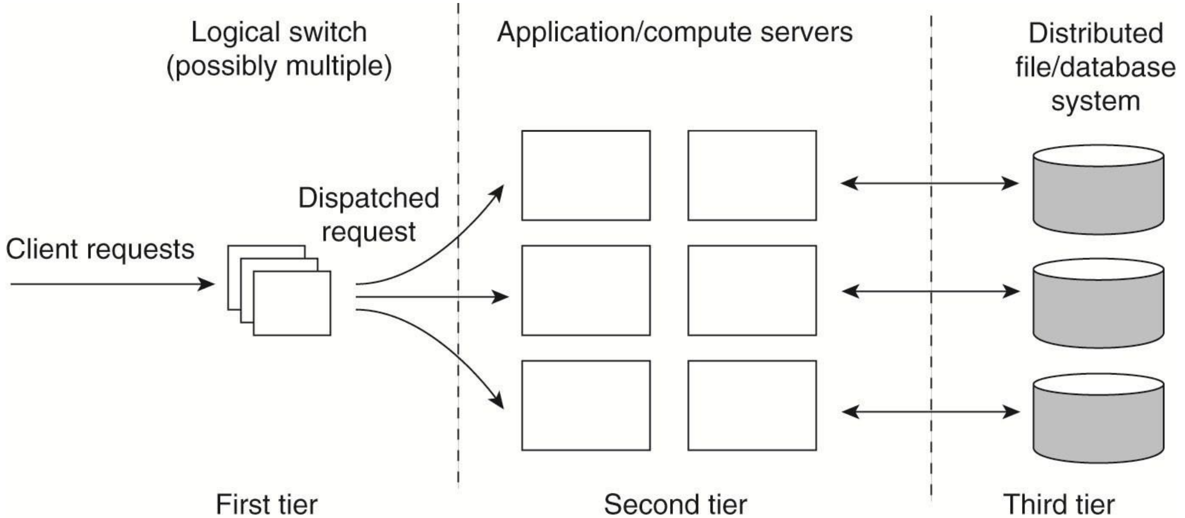
Processing multiple requests

- How to deal with multiple, concurrent messages received?
- Problems:
  - Different roles: clients versus servers/services
    - A large number of clients interact with a small number of servers/services
    - A single process might receive a lot of messages at the same time
- Impacts
  - performance, reliability, cost, etc.

Iterative vs Concurrent processing



Using replicated processes

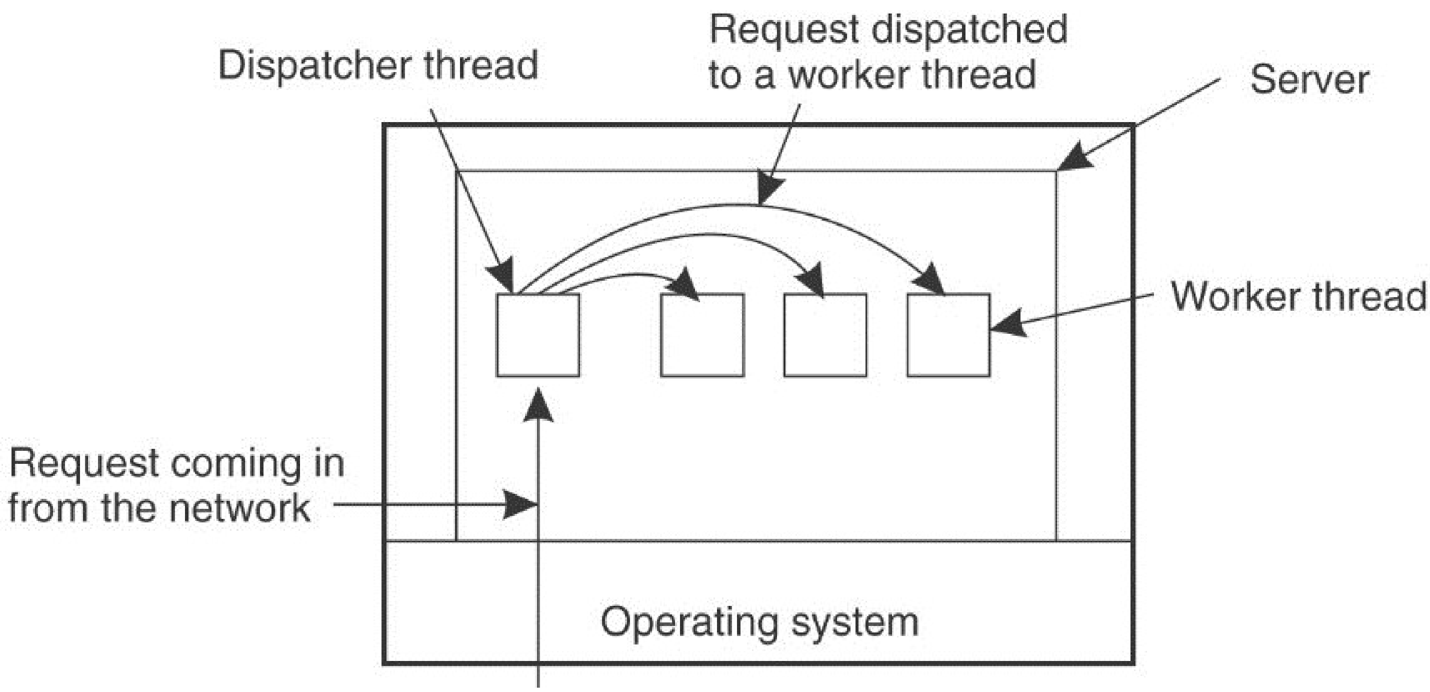


Often load balancing mechanisms are needed

Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Q: How does this model help to improve performance and fault-tolerance? What would be a possible mechanism to reduce costs based on the number of client requests?

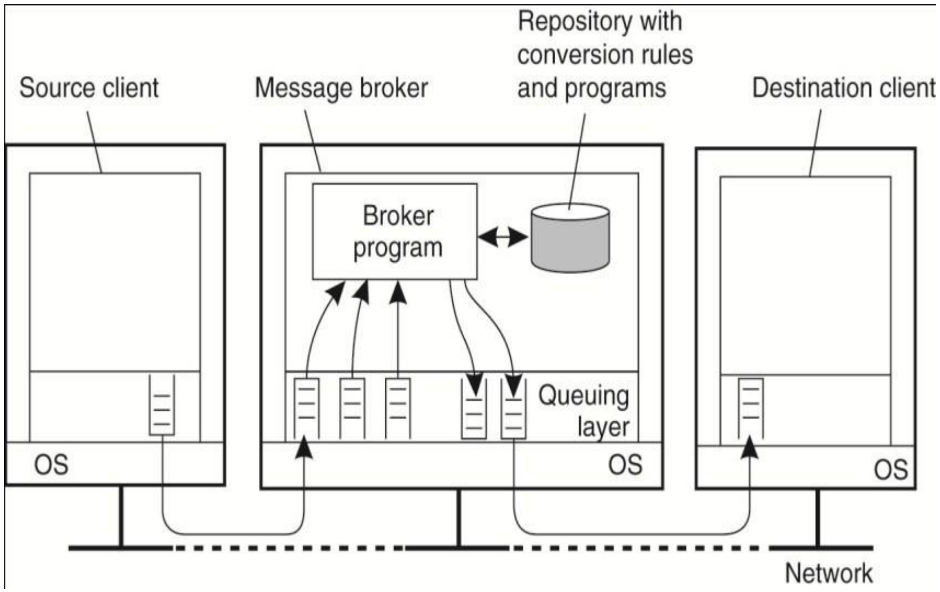
Using multiple threads



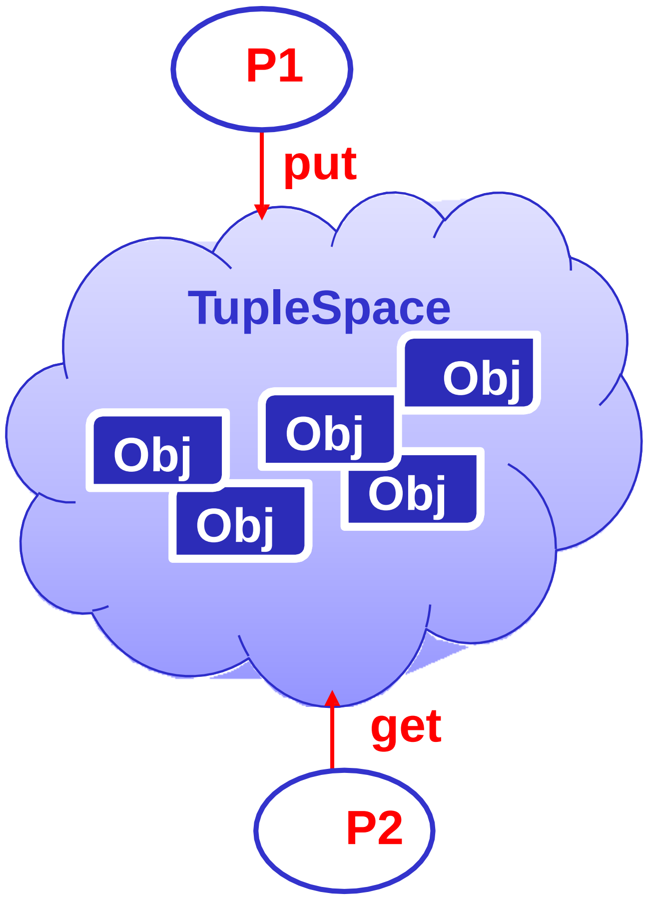
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Q: Compare this architectural model with the super-server model?

Using message brokers/space repository



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall



Example

- Get a free instance of RabbitMQ from cloudamqp.com
- Get code from: https://github.com/cloudamqp/java-amqp-example
- First run the test sender, then run the receiver

Summary

- Complex and diverse communication patterns, protocols and processing models
- Choices are based on communication requirements and underlying networks
  - Understand their pros/cons
  - Understand pros and cons of their technological implementations
- Dont forget to play with some simple examples to understand existing concepts