Communication in Distributed Systems – Fundamental Concepts

Hong-Linh Truong
Distributed Systems Group,
Vienna University of Technology

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong

Learning Materials

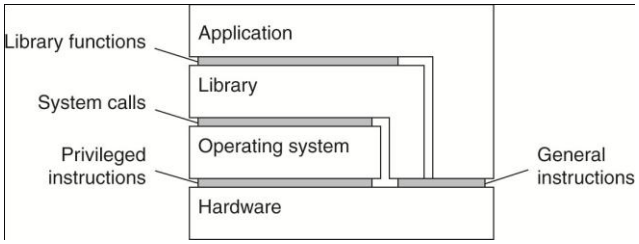
- Main reading:
  - Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall
    - Chapters 3 & 4
- Others
  - George Coulouris, Jean Dollimore, Tim Kindberg, „Distributed Systems – Concepts and Design“, 2nd Edition
    - Chapters 3,4, 6.
  - Craig Hunt, TCP/IP Network Administration, 3edition, 2002, O‘Reilly.
- Test the examples in the lecture

Outline

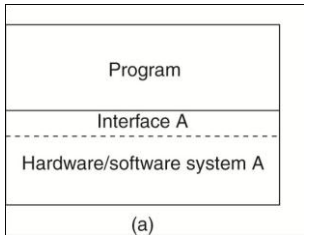
- Programs, interfaces, systems, and communication
- Key issues in communication in distributed systems
- Protocols
- Processing requests
- Summary

Programs, interfaces, systems and communication (1)

Program, interface and systems in typical machines

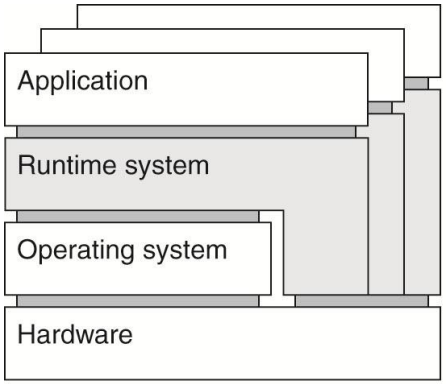


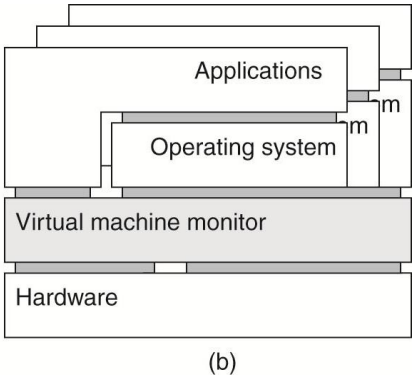
leads to



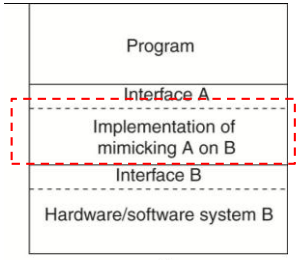
introduce virtualization interfaces

Program, interface and systems in typical virtual machines





leads to

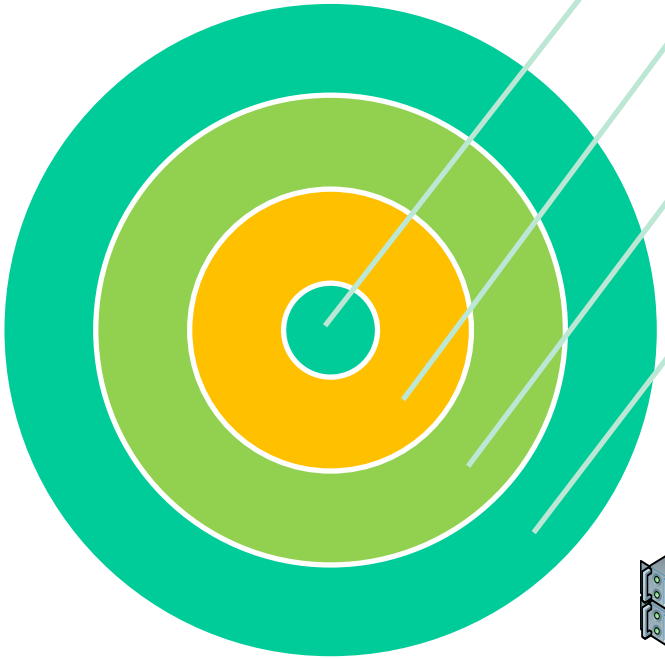


Figures source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Q: Why virtualization techniques are useful?

Programs, interfaces, systems and communication (2)

Software stack



Hardware (CPU, Memory, Network)

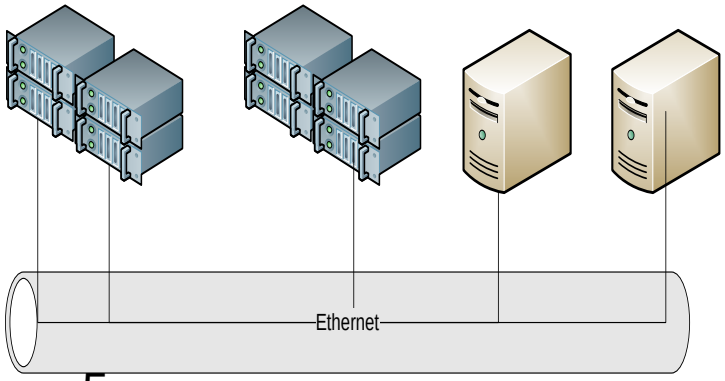
Operating System

Middleware/Libary/Runtime system

Application

Programs

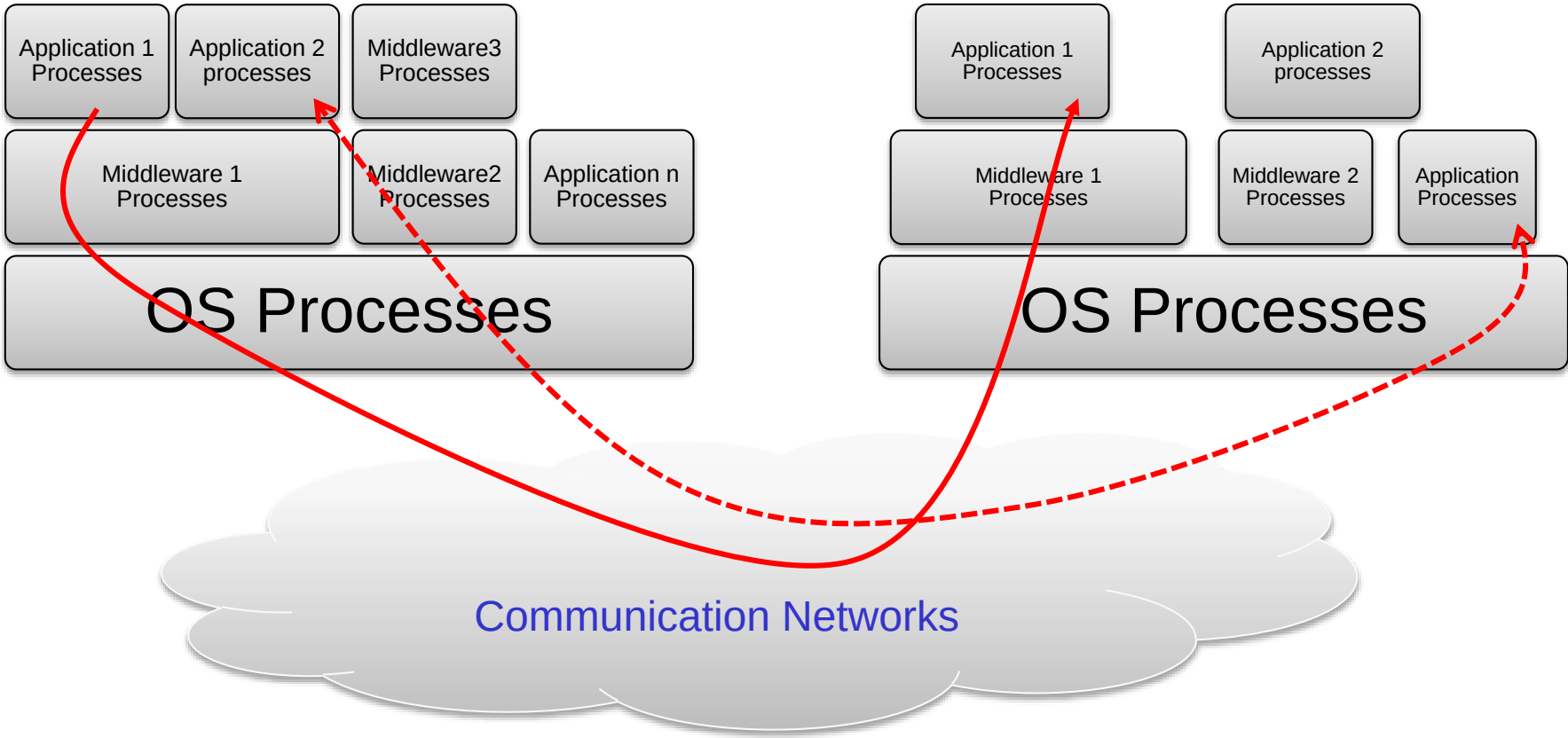
- C/C++/Java, Python, …
- Different types of programs: systems versus applications; sequential versus parallel ones; clients versus servers/services



Programs, interfaces, systems and communication (3)

Communication in distributed systems

- between processes within a single application/middleware/service
- among processes belonging to different applications/middleware/services

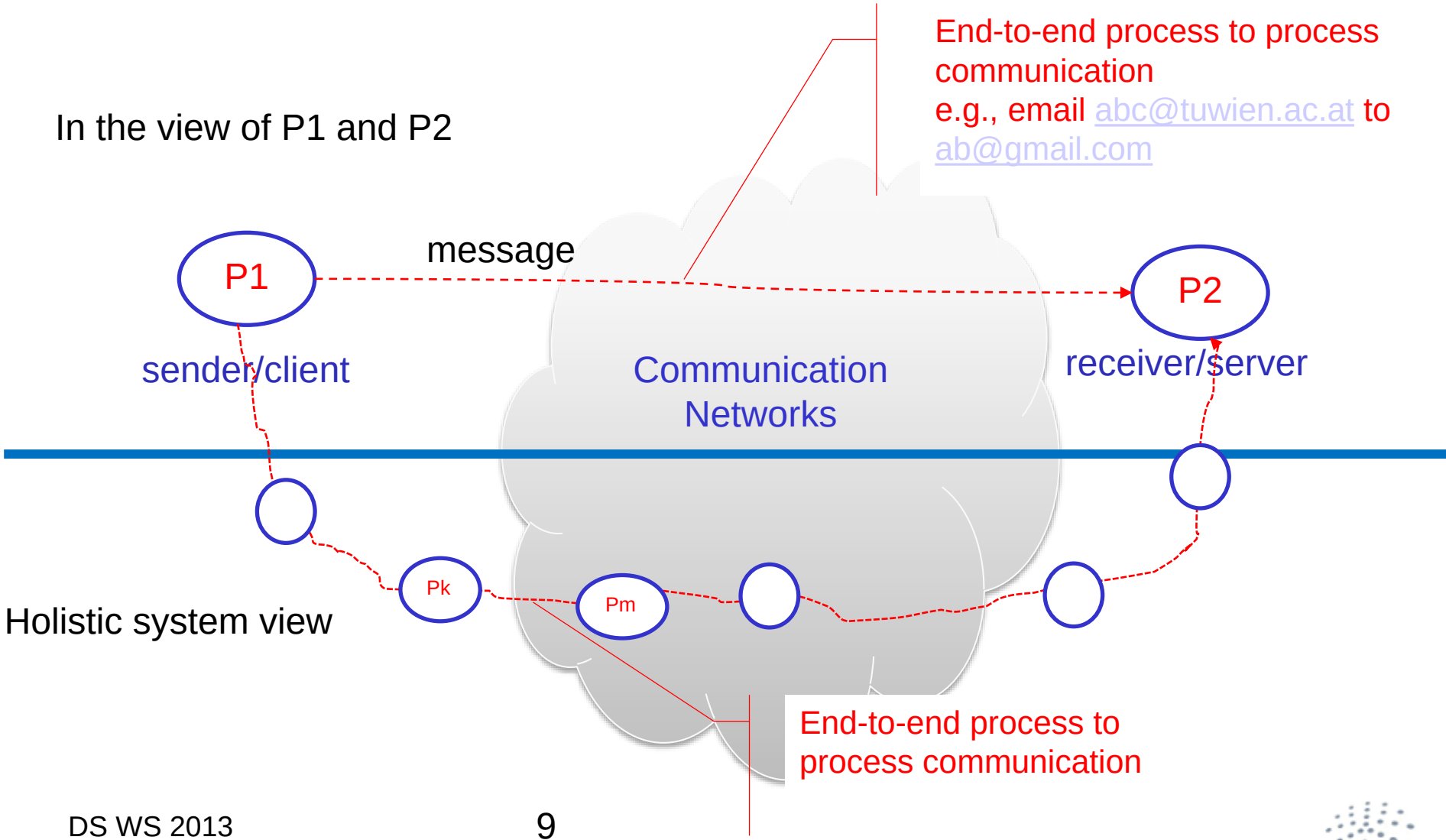


KEY ISSUES

Communication networks in distributed systems

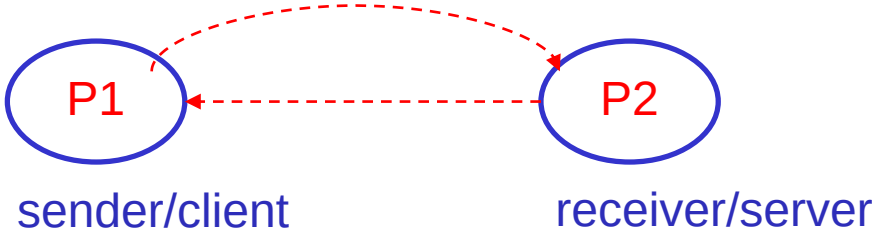
- Maybe designed for specific types of environments
  - High performance computing, M2M, building/home, etc.
  - Voices, documents, sensory data, etc.
- Distributed, different network spans
  - Personal area networks (PANs), local area networks (LANs), campus area networks (CANs), metropolitan area networks (MANs), and wide area networks (WANs)
- Different layered networks for distributed systems
  - Physical versus overlay network topologies (virtual network topologies atop physical networks)

Layered communication

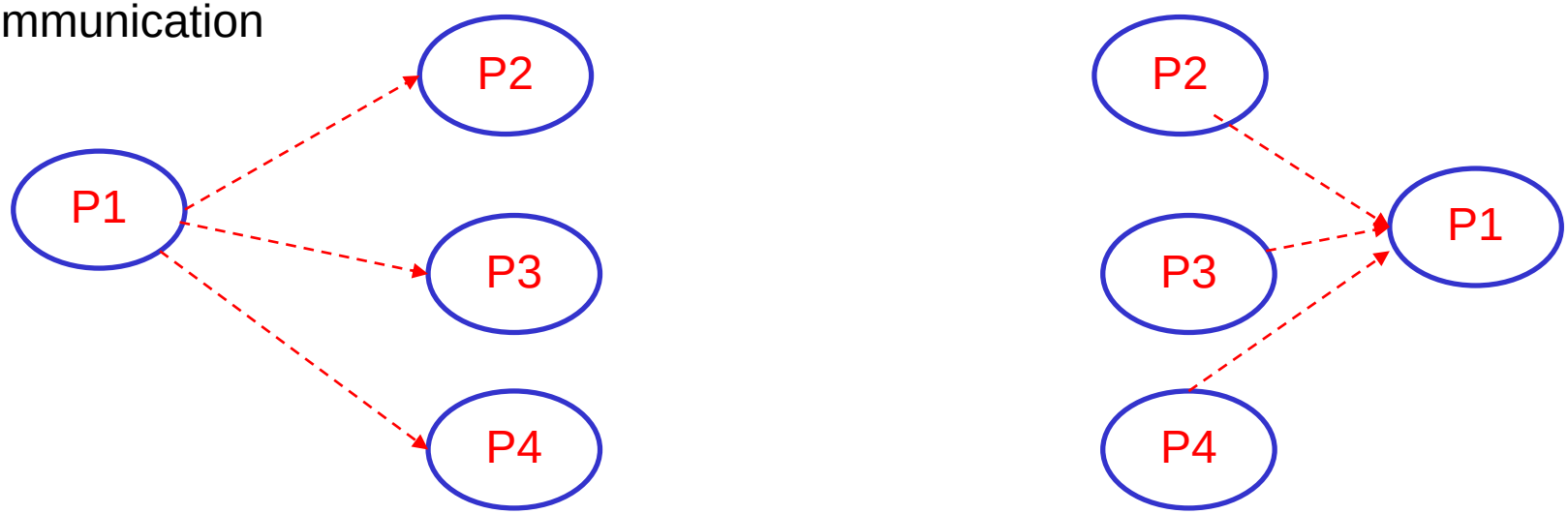


Communication Patterns

One-to- one/client-server



Group communication



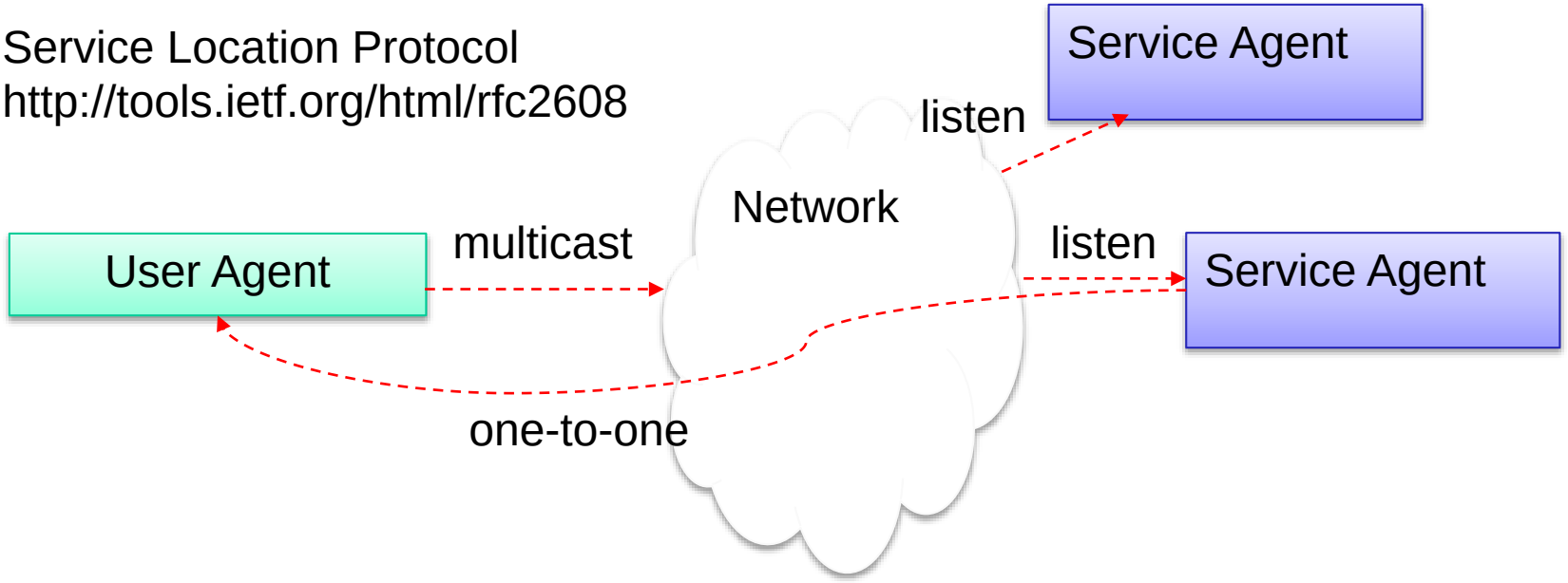
Q: What are the benefits of group communication, give some examples?

Identifiers of entities particiapting 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

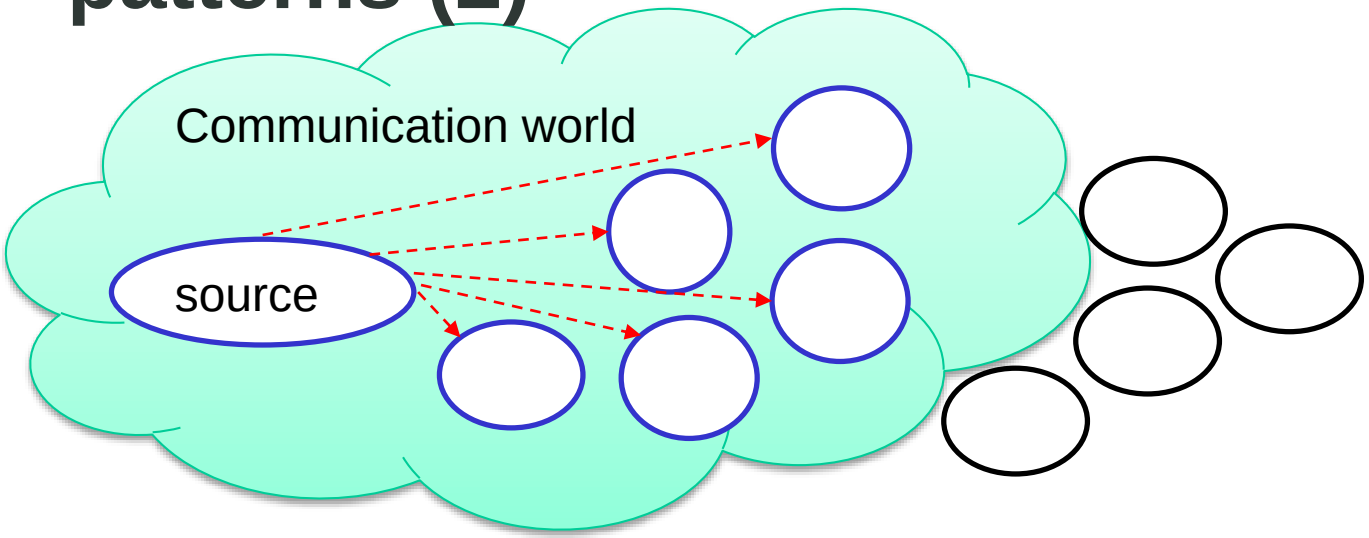
Examples of communication patterns (1)

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)



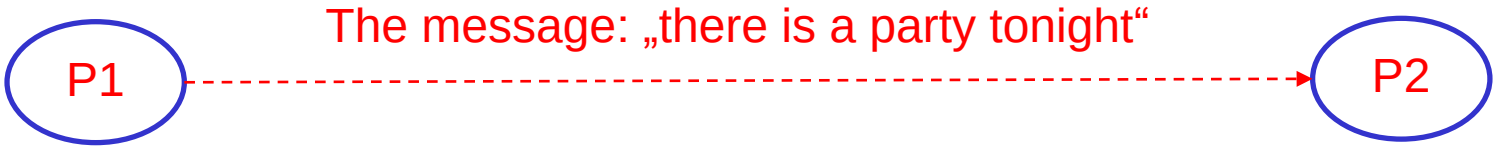
- MPI (Message Passing Interface)

```
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



Write the message in a letter

Go to the post office

Send the letter to P2

Find the phone number of P2

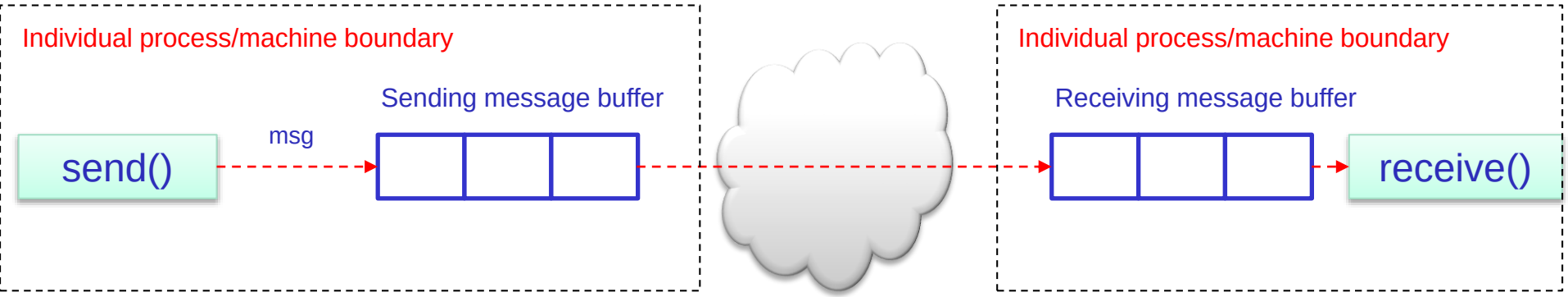
Call P2

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

Q: What are the pros/cons of connection-oriented/connectionless communications? Is it possible to have a connectionless communication between (P1,P2) through some connection-oriented connections?

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

Q: Analyze the benefits of non-blocking communication. How non-blocking receive() works?

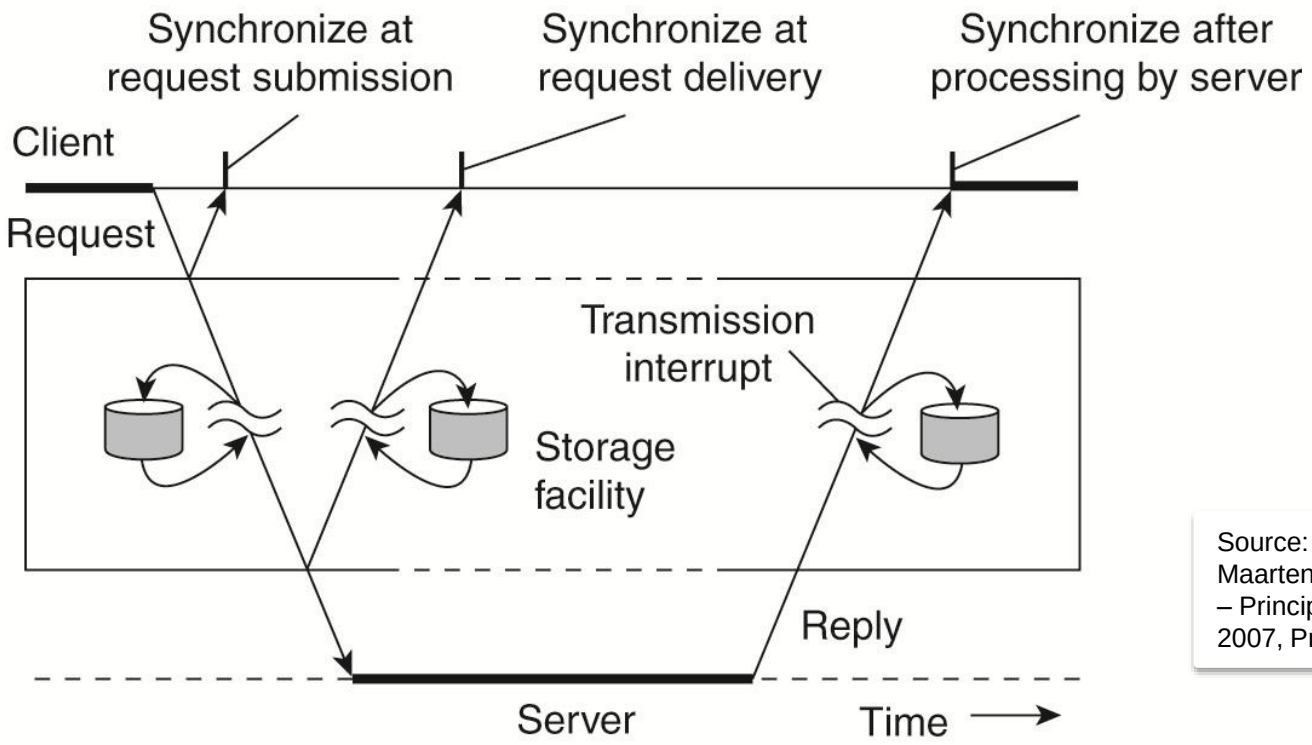
Persistent and transient communication

- 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: continues after sending messages
  - Non blocking send
  - Receive may/may not be blocking
  - Callback mechanisms
- Synchronous: the sender waits until it knows the messages delivered to the receiver
  - Blocking send/blocking receive
  - Typically utilize connection-oriented and keep-alive connection
  - Blocking request-reply styles

Different forms of communication

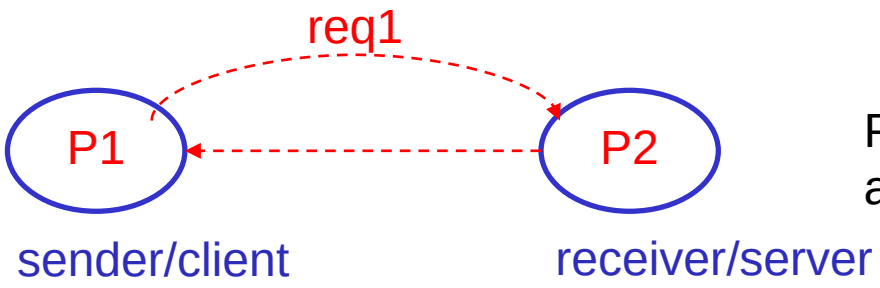


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Q: How can we achieve the „persistence“? What are possible problems if a server sends a accepted/replied/ACK message before processing the request?

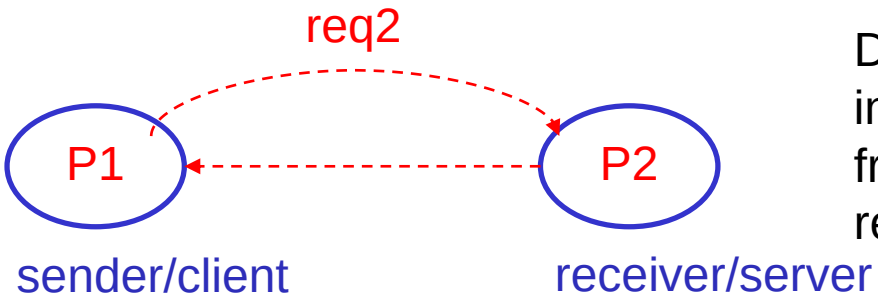
Stateful versus Stateless Server

T0



P2 has no information about P1 before

T1

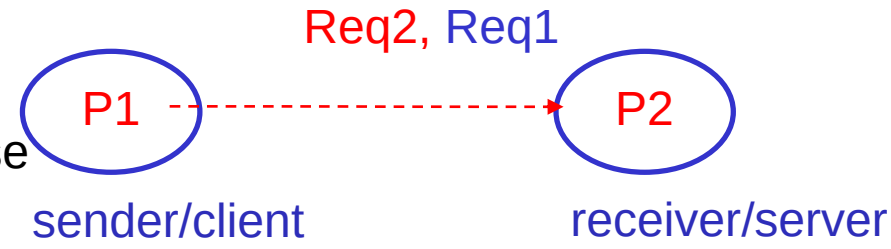


Does P2 keep information about P1 from the previous request?

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

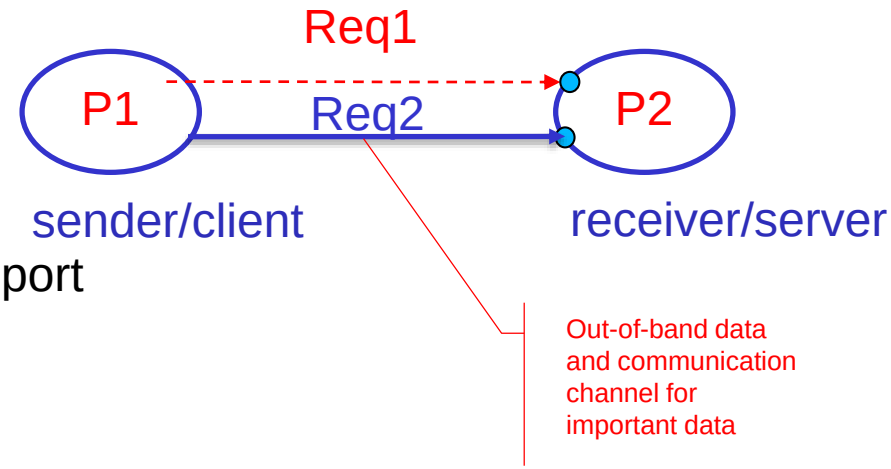
Handling out of band data

Normal case



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

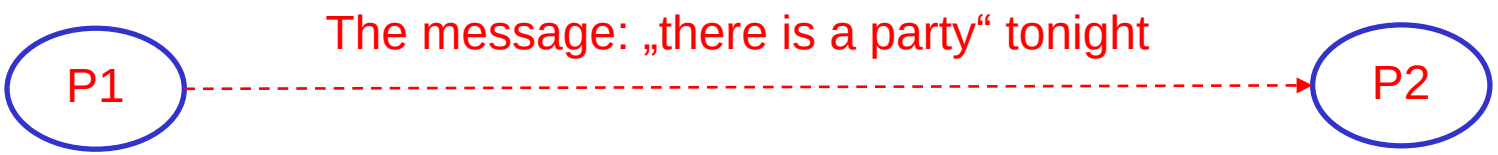
Out of band data with a separate transmission port



Q: How can out-of-band data and normal data be handled by using the same transmission channel?

COMMUNICATION PROTOCOLS

Some key questions – Protocols

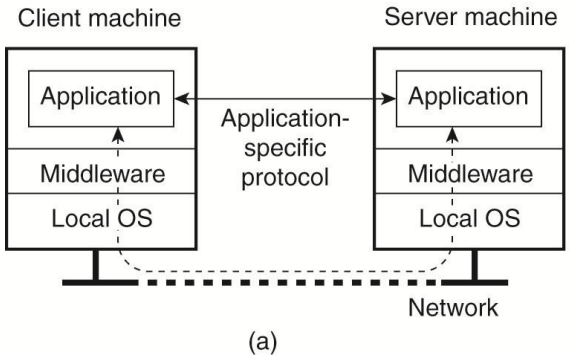


- Communication patterns
  - Can I use a single sending command to send the message to multiple people?
- Identifier/Naming/Destination
  - How do I identify the guys I need to send the message
- Connection setup
  - Can I send the message without setting up the connection
- Message structure
  - Can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

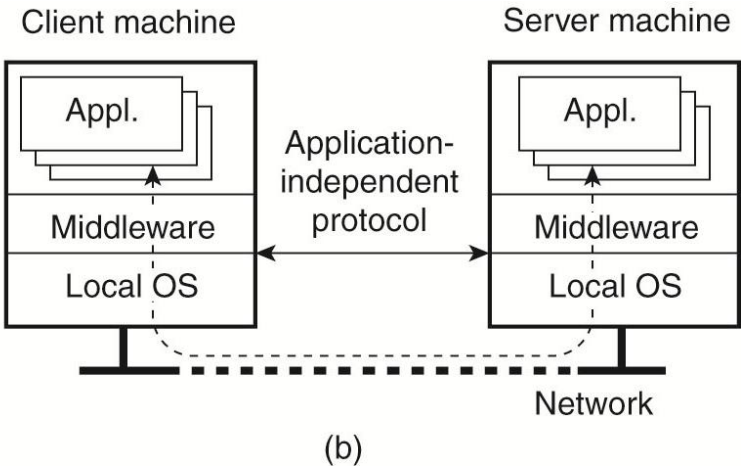
A communication protocol will describe rules addressing these issues

Applications and Protocols

Application-specific protocols



Application-independent protocols

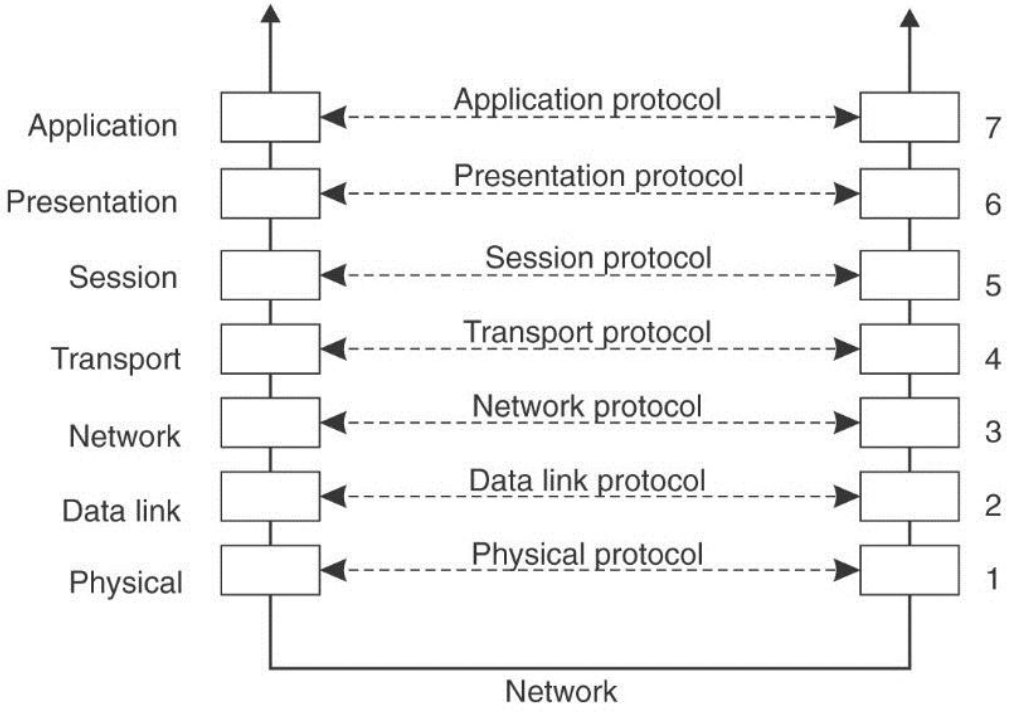


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Layered Communication Protocols

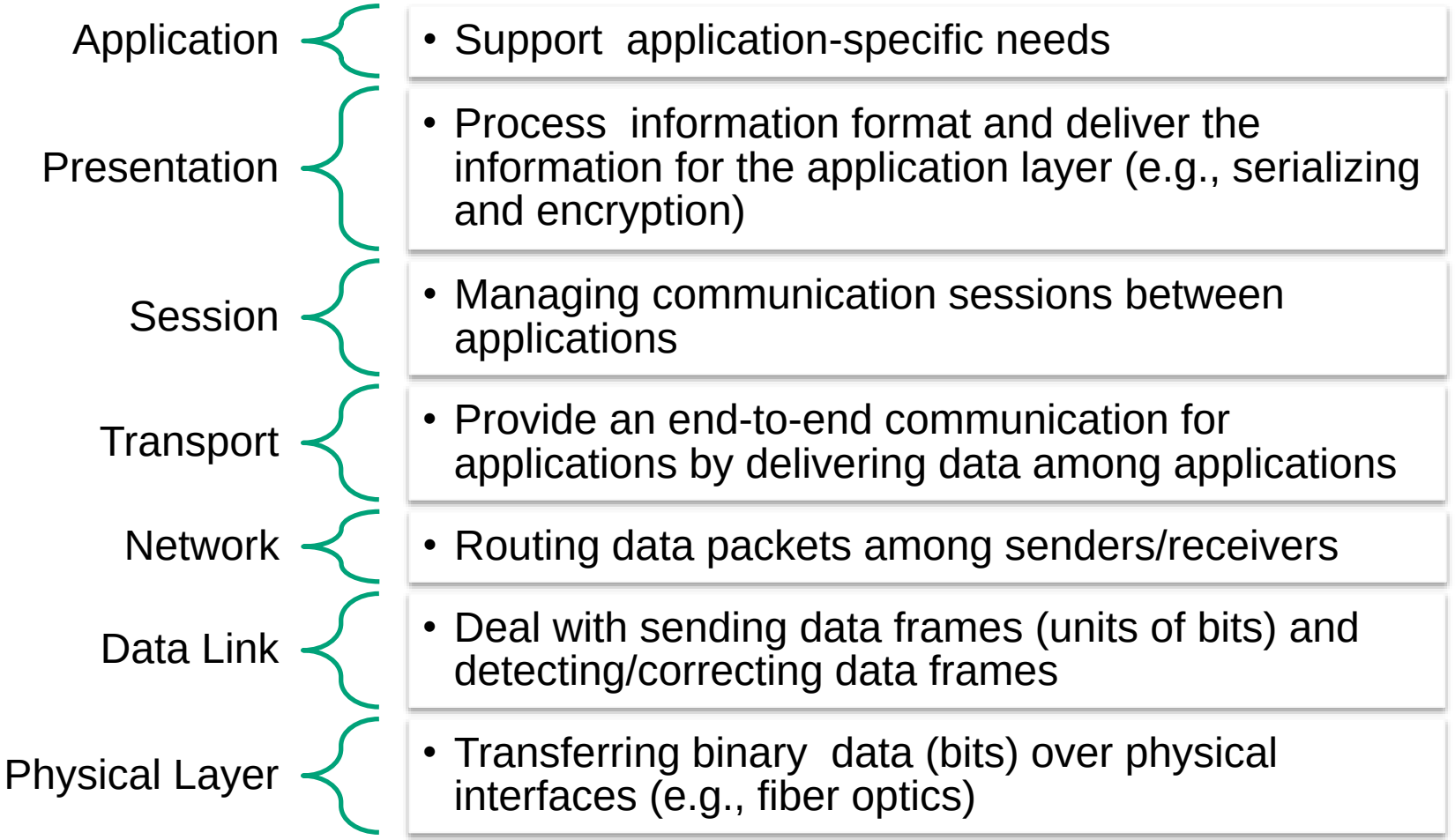
- Complex and open communication requires multiple communication protocols
- Communication protocols are typically organized into differ layers: layered protocols/protocol stacks
- Conceptually: each layer has a set of different protocols for certain communication functions
  - Different protocols are 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



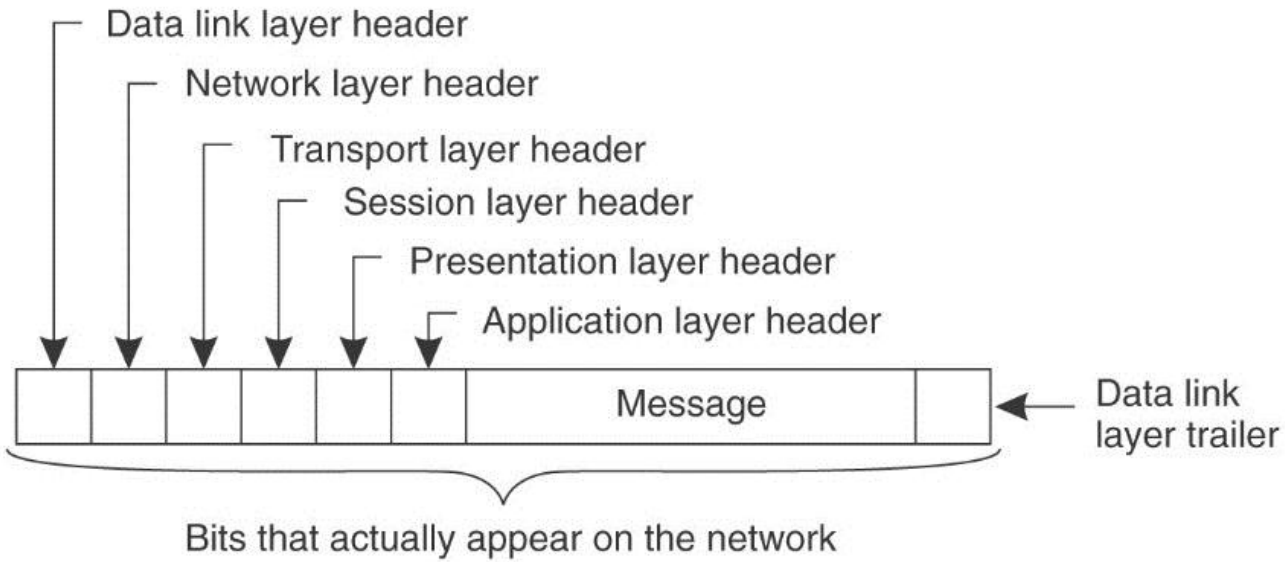
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

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
• Managing communication sessions between applications
Transport
• Provide an end-to-end communication for applications by delivering data among applications
Network
• Routing data packets among senders/receivers
Data Link
• Deal with sending data frames (units of bits) and detecting/correcting data frames
Physical Layer
• Transferring binary data (bits) over physical interfaces (e.g., fiber optics)

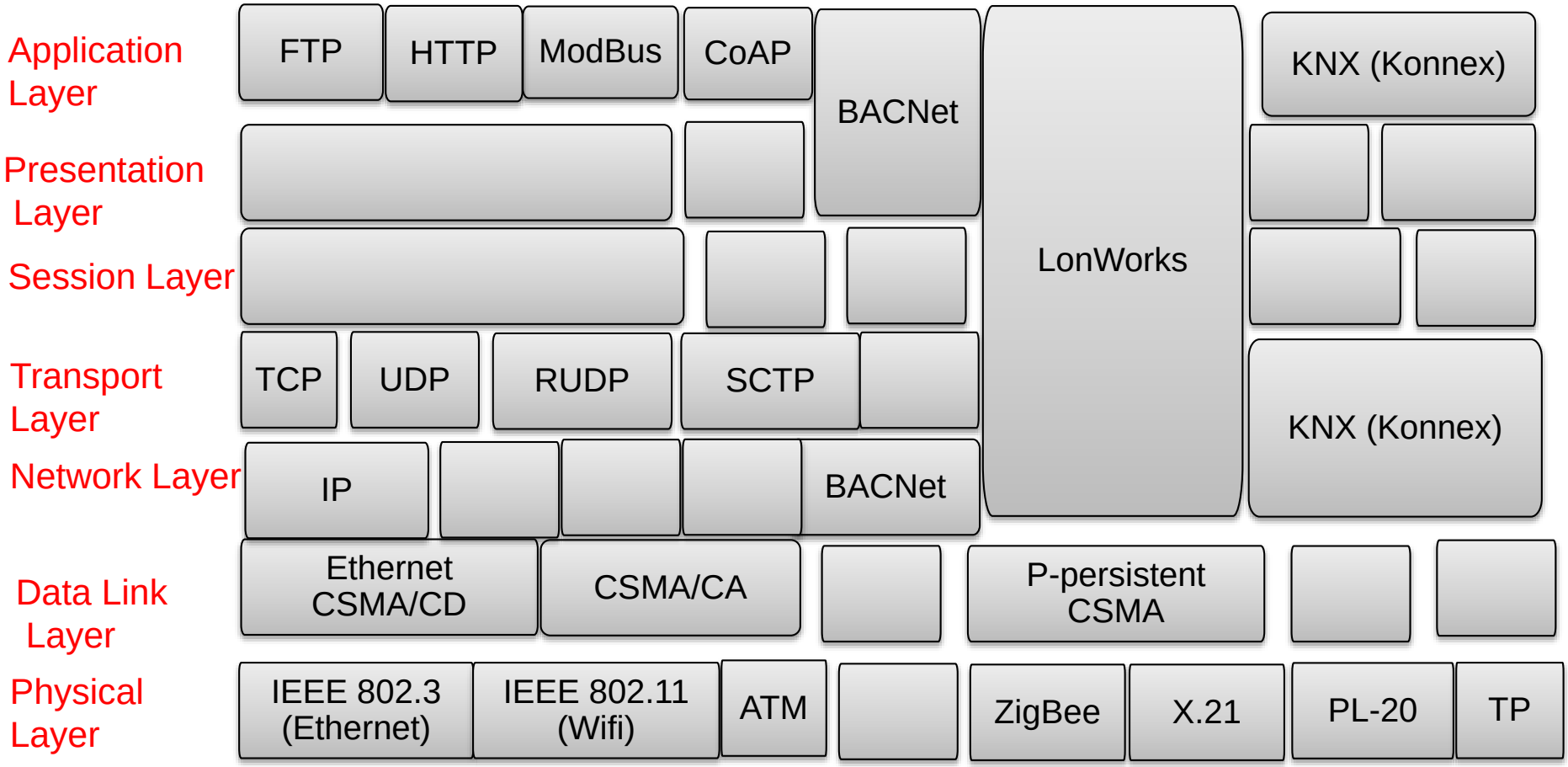
How layered protocols work – message exchange

- Principles of constructing messages/data encapsulation



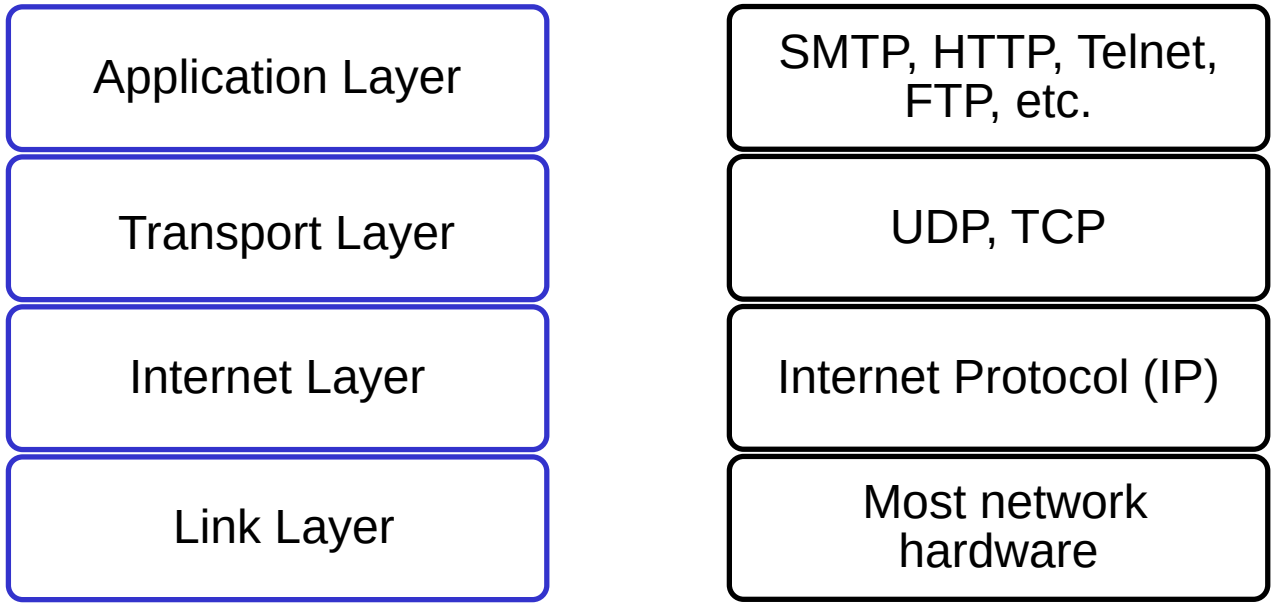
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Examples of Layered Protocols



TCP/IP

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



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

Internet Protocol (IP)

- Define the datagram as the basic data unit
- Define the Internet address scheme
- Transmit data between the Network Access Layer and Transport Layer
- Route datagrams to destinations
- Divide and assemble datagrams

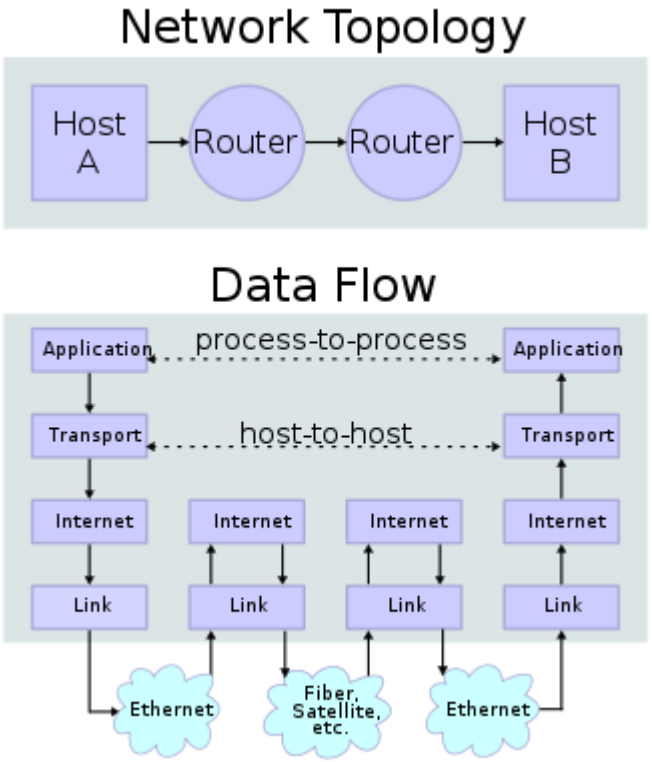


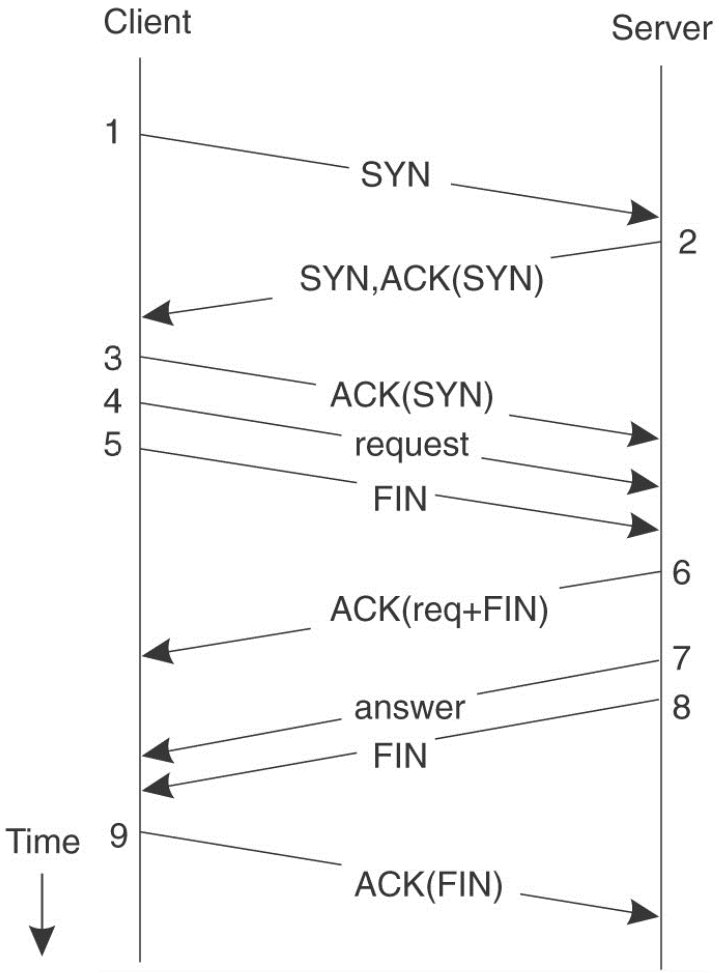
Figure source:
http://en.wikipedia.org/wiki/Internet_protocol_suite

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	Data sent in Messages
Transport Layer	Segment	Packet
Internet Layer	Datagram	Datagram
Link Layer	Frame	Frame

TCP operations



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

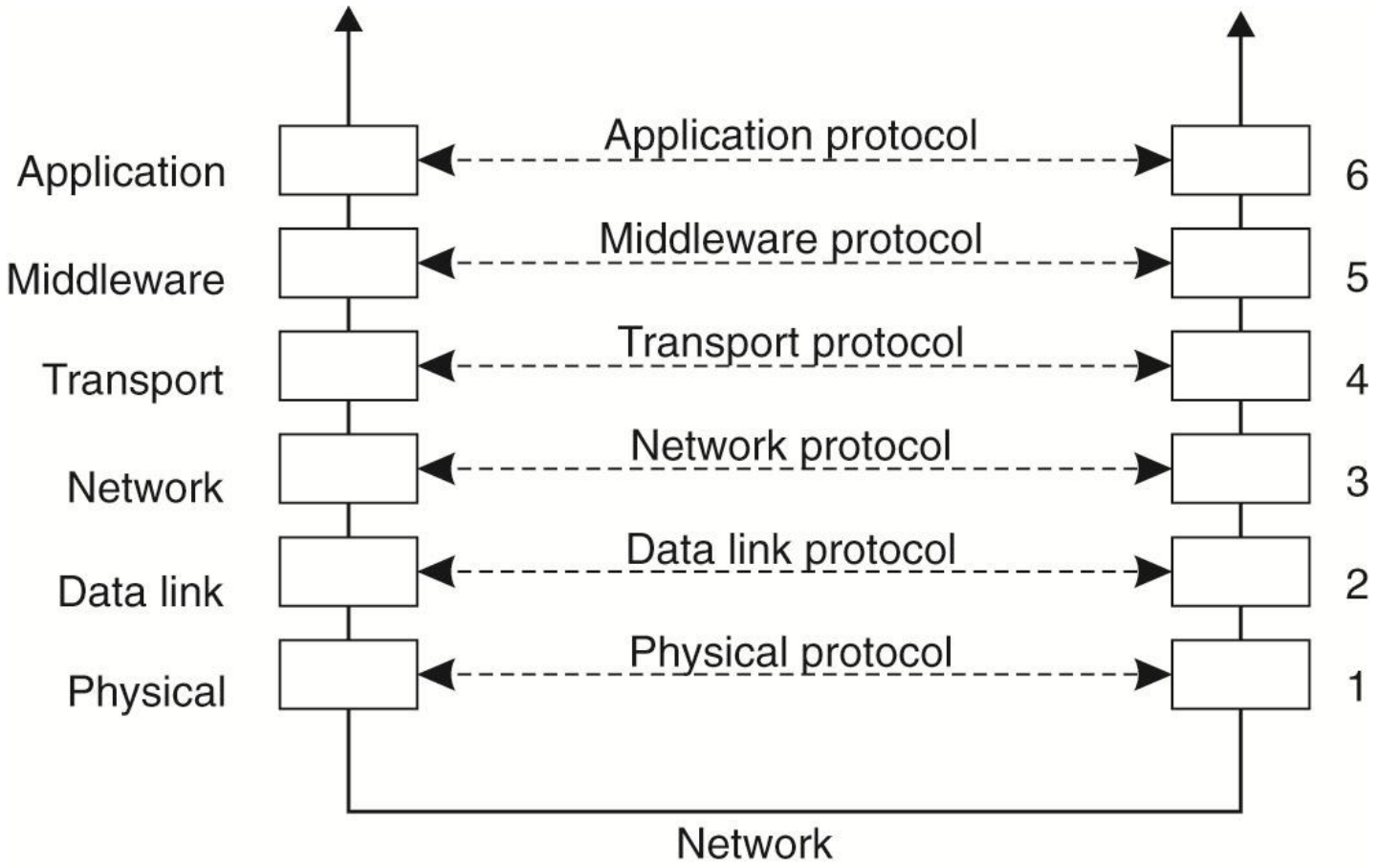
```
$sudo nast -d -T iptest >ip.out
$wget www.tuwien.ac.at
```

```
---[ TCP ]---------------------------------------------------------
192.168.1.7:46023(unknown) -> 128.130.35.76:80(http)
TTL: 64    Window: 14600       Version: 4              Length: 60
FLAGS: -S-----        SEQ: 3308581872 - ACK: 0
Packet Number: 16

---[ TCP ]---------------------------------------------------------
128.130.35.76:80(http) -> 192.168.1.7:46023(unknown)
TTL: 54    Window: 14480       Version: 4              Length: 60
FLAGS: -S--A--        SEQ: 3467332359 - ACK: 3308581873
Packet Number: 17

---[ TCP ]---------------------------------------------------------
192.168.1.7:46023(unknown) -> 128.130.35.76:80(http)
TTL: 64    Window: 115         Version: 4              Length: 52
FLAGS: ----A--        SEQ: 3308581873 - ACK: 3467332360
Packet Number: 18

---[ TCP ]---------------------------------------------------------
192.168.1.7:46023(unknown) -> 128.130.35.76:80(http)
TTL: 64    Window: 115         Version: 4              Length: 166
FLAGS: ---PA--        SEQ: 3308581873 - ACK: 3467332360
Packet Number: 19

---[ TCP Data ]----------------------------------------------------

GET / HTTP/1.1


---[ TCP ]---------------------------------------------------------
128.130.35.76:80(http) -> 192.168.1.7:46023(unknown)
TTL: 54    Window: 114         Version: 4              Length: 52
FLAGS: ----A--        SEQ: 3467332360 - ACK: 3308581987
Packet Number: 20

---[ TCP ]---------------------------------------------------------
128.130.35.76:80(http) -> 192.168.1.7:46023(unknown)
TTL: 54    Window: 114         Version: 4              Length: 1500
FLAGS: ----A--        SEQ: 3467332360 - ACK: 3308581987
Packet Number: 21

---[ TCP Data ]----------------------------------------------------

HTTP/1.1 200 OK
```

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.

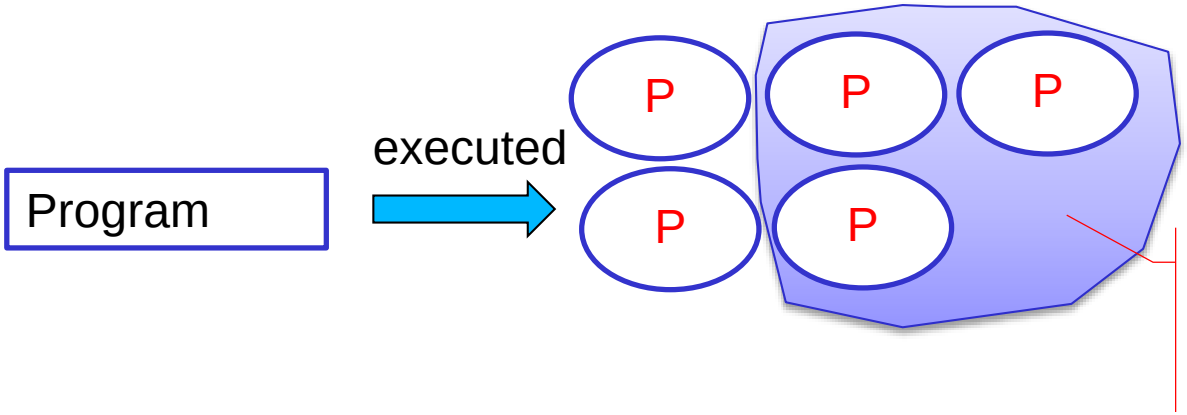
Middleware Protocols



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

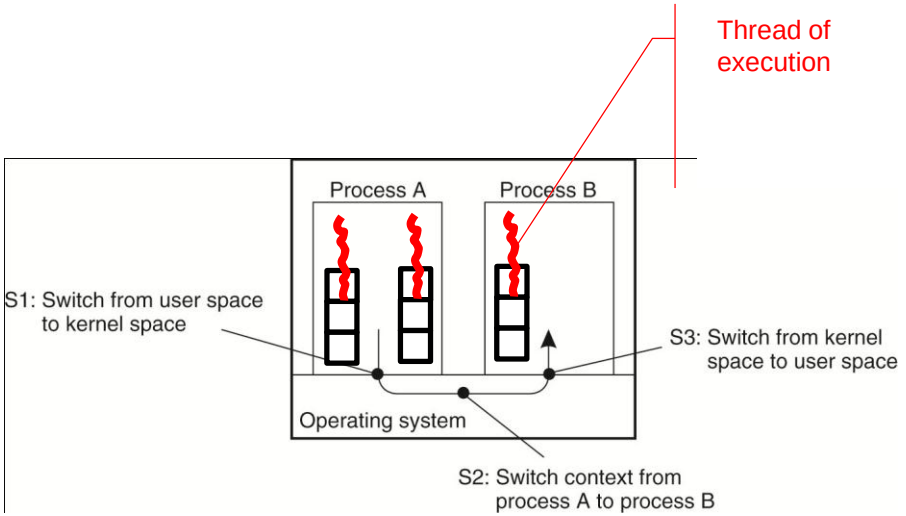
HANDLING COMMUNICATION MESSAGES/REQUESTS

Process versus thread



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



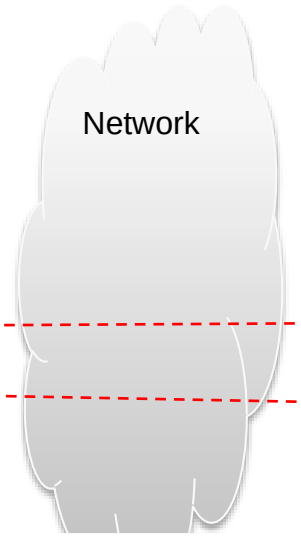
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Where communication tasks take 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)
```

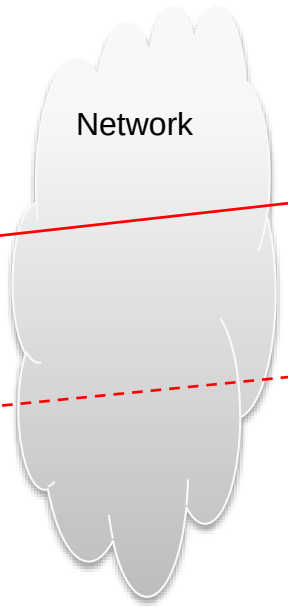



```
# Echo server program
import socket

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

```
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 object calls

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();
      }
   }
}
```

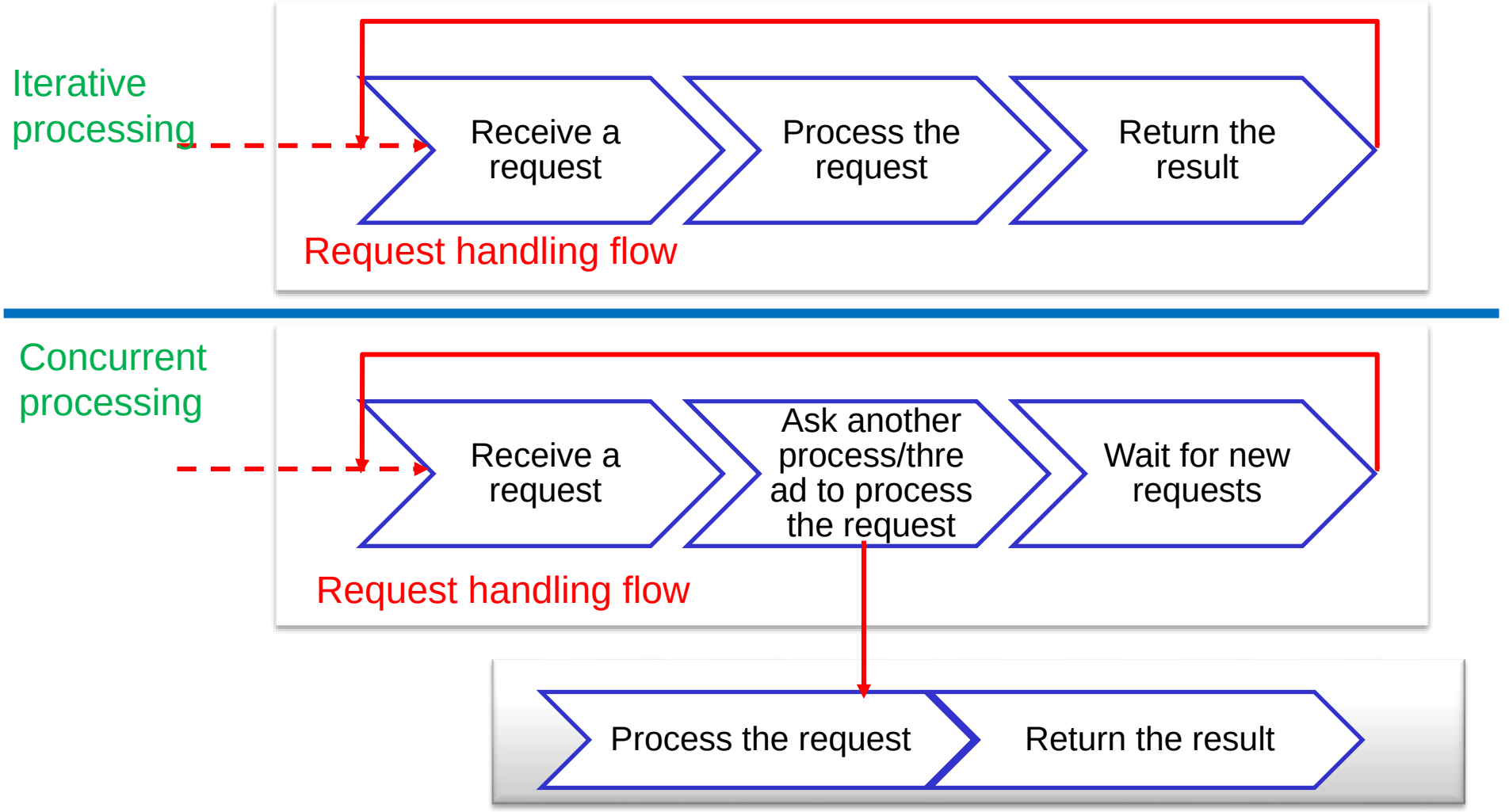
```
public interface Compute extends Remote {
   <T> T executeTask(Task<T> t) throws RemoteException;
}
….
public class ComputeEngine implements Compute {

   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:
http://docs.oracle.com/javase/tutorial/rmi/overview.html

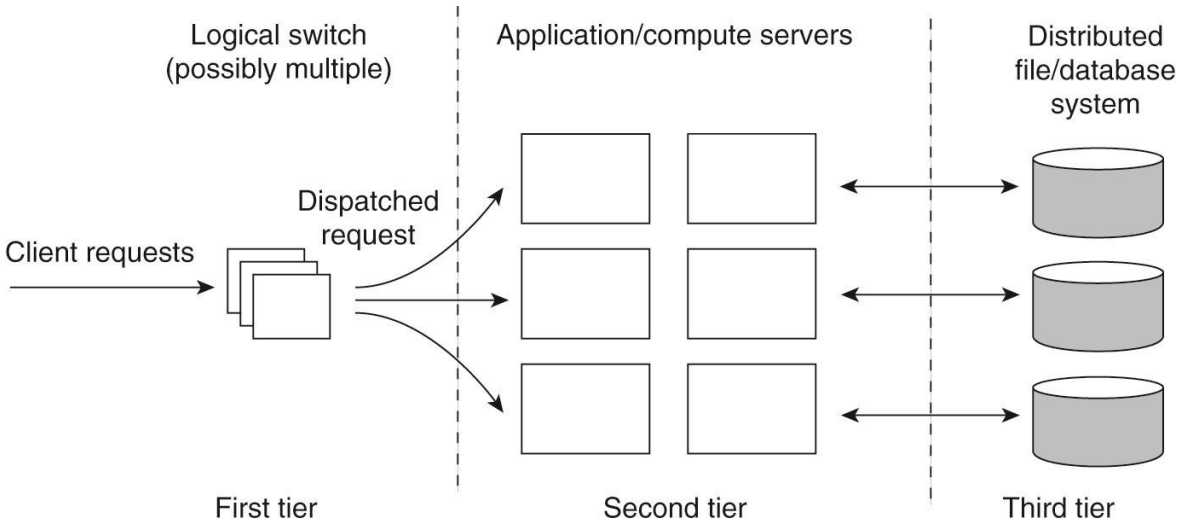
Processing multiple requests

- 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 versus concurrent processing



Using replicated processes

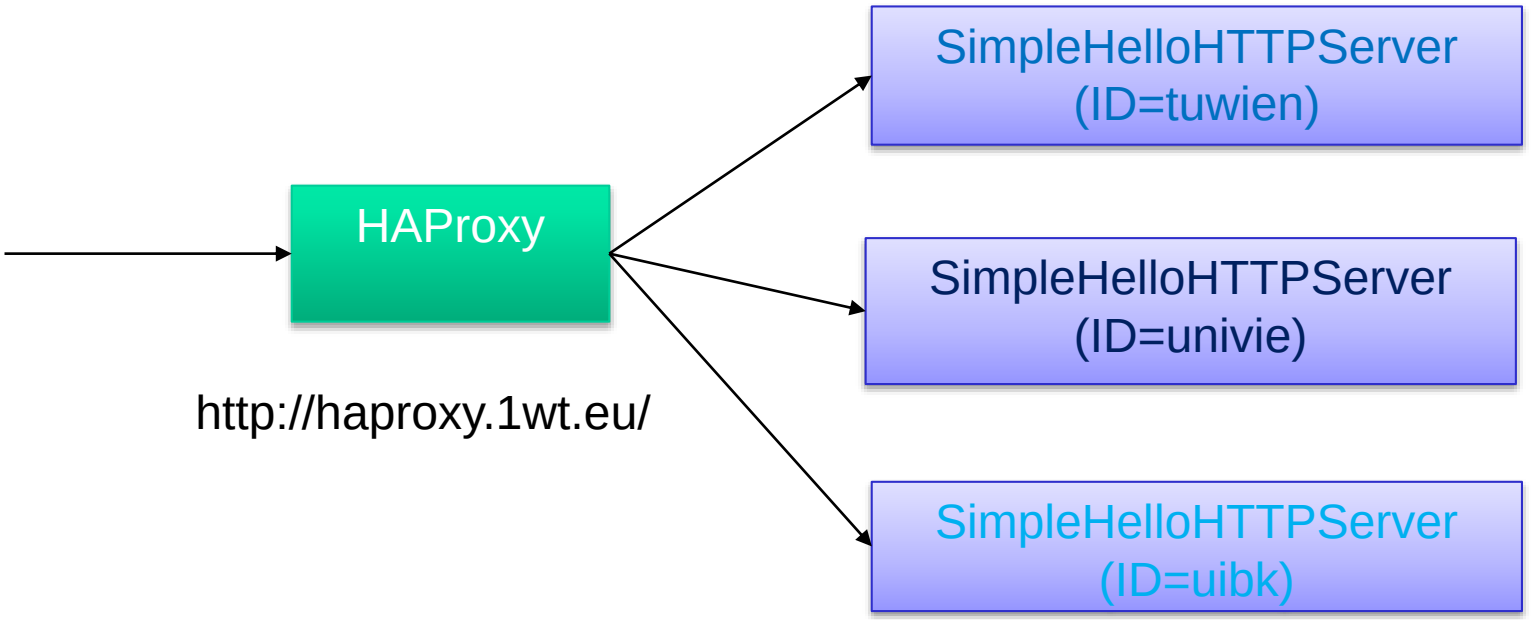


Often loadbalancing mechanisms are needed

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

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

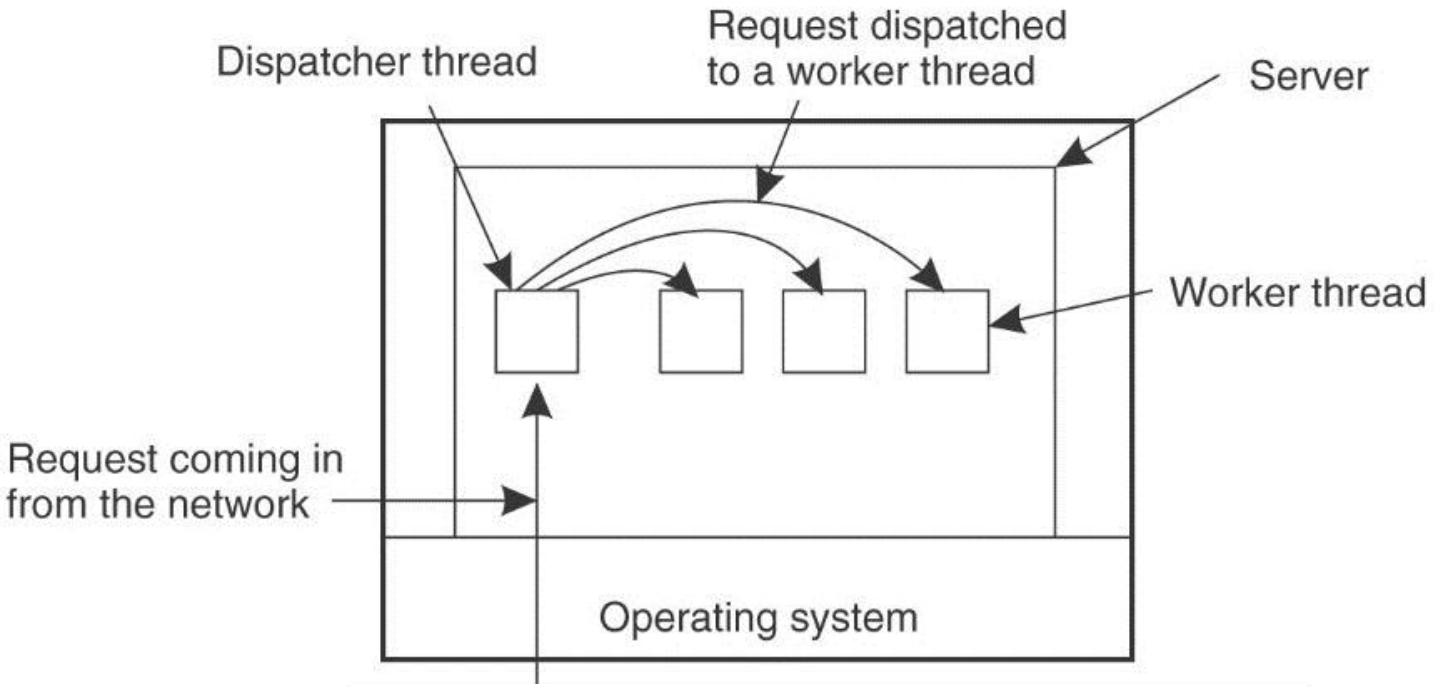
Example



- Get a small test
  - Download haproxy, e.g.

  $sudo apt-get install haproxy
  - Download SimpleHelloHTTPServer.java and haproxy configuration
    - http://bit.ly/19xFDRC
  - Run 1 haproxy instance and 3 http servers
    - Modify configuration and parameters if needed
  - Run a test client

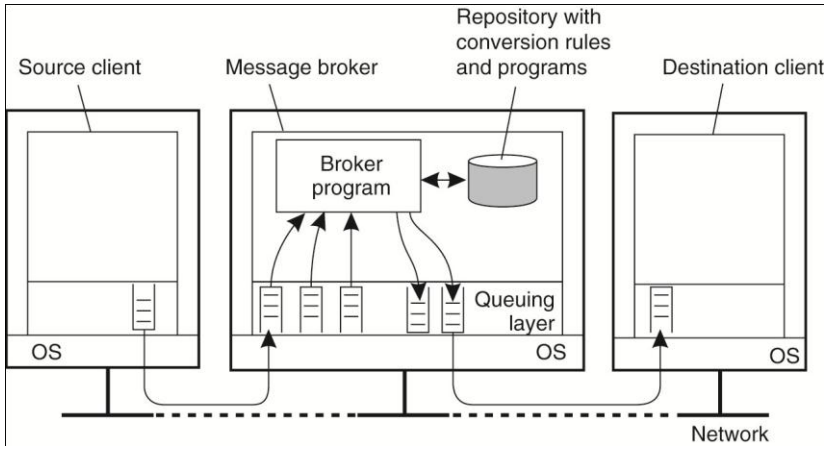
Using multiple threads



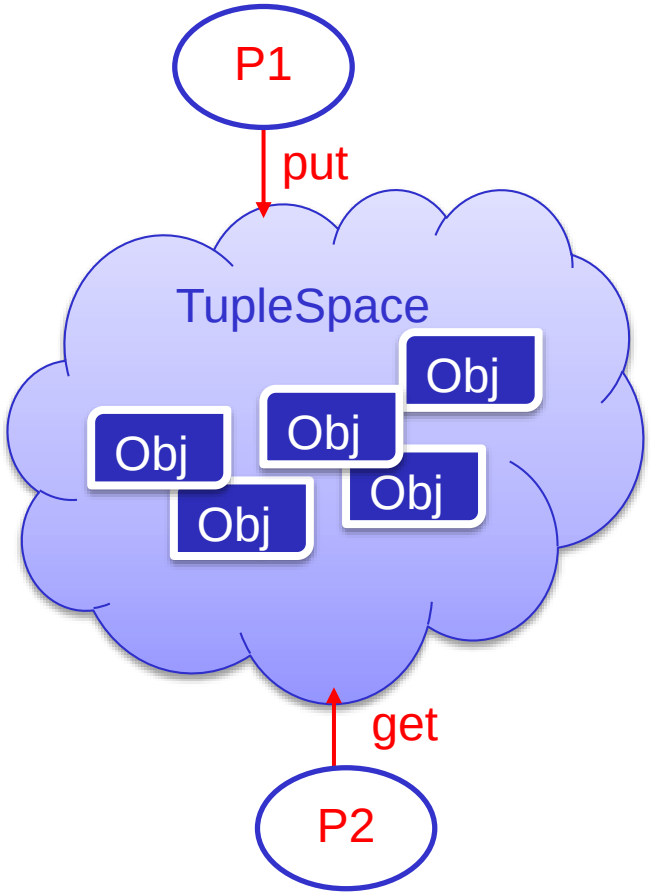
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Q: How this architectural model would be applied/similar to worker processes or 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


```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    for (int i=0; i<100; i++) {
        String message = "Hello distributed systems guys:  "+i;
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
        System.out.println(" [x] Sent '" + message + "'");
        new Thread().sleep(5000);
    }
```

```
while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    String message = new String(delivery.getBody());
    System.out.println(" [x] Received '" + message + "'");
  }
```

Note: i modified the code a bit

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 some simple examples to understand existing concepts

Thanks for your attention

Hong-Linh Truong
Distributed Systems Group
Vienna University of Technology
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong