Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

Algorithms and Programming IV

Communication Paradigms in Distributed Systems

Summer Term 2023 | 19.06.2023

Claudia Müller-Birn, Barry Linnert

Freie Universität Berlin

Our topics today

Recap

Architectural Styles

Layered architectures

Service-oriented architectures

Publish-subscribe architectures

Communication Paradigms

Interprocess Communication

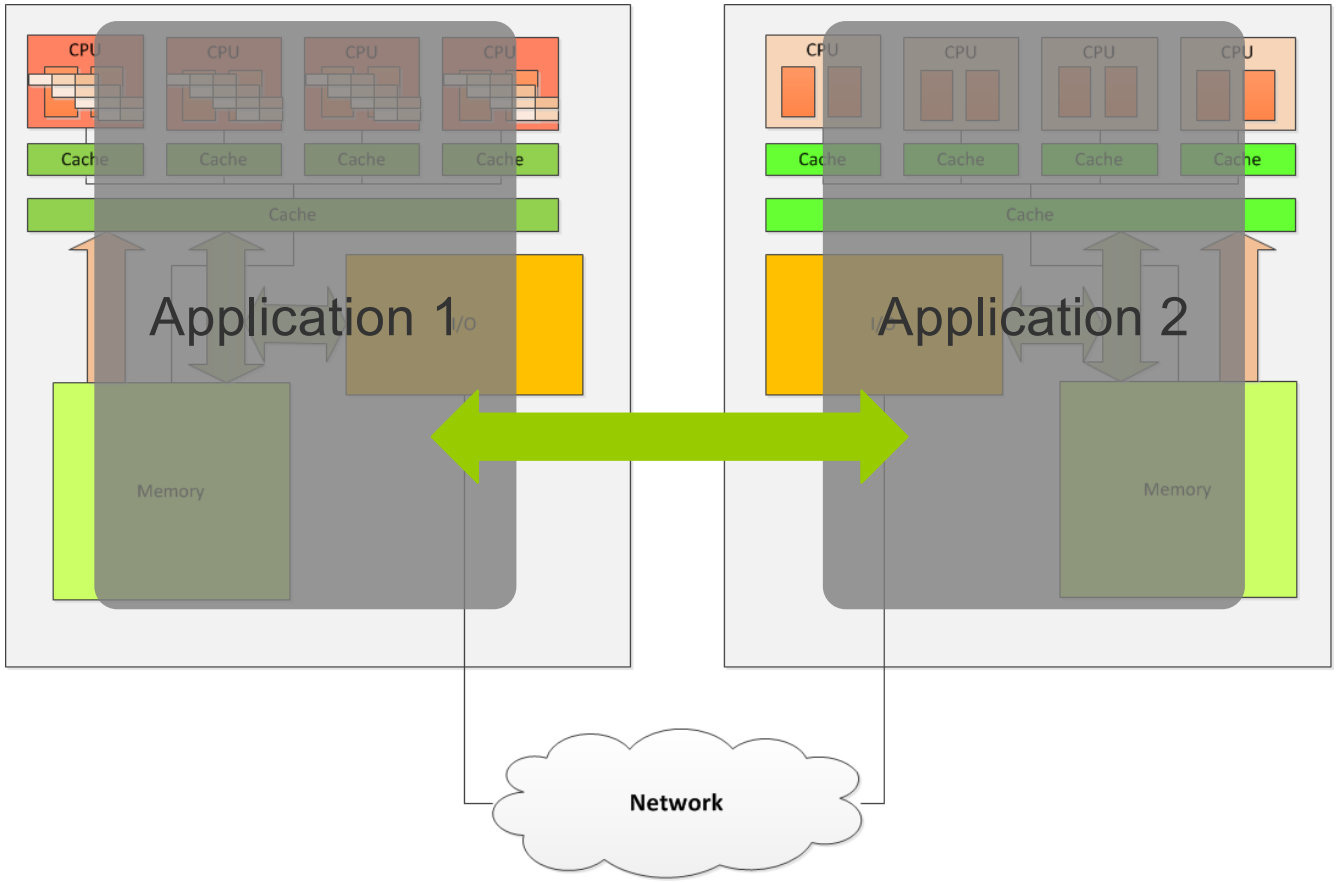
- API for Internet Protocols
- UDP Datagram Communication
- TCP Stream Communication

Freie Universität Berlin

RECAP

Freie Universität Berlin

Recap: Distributed System Model

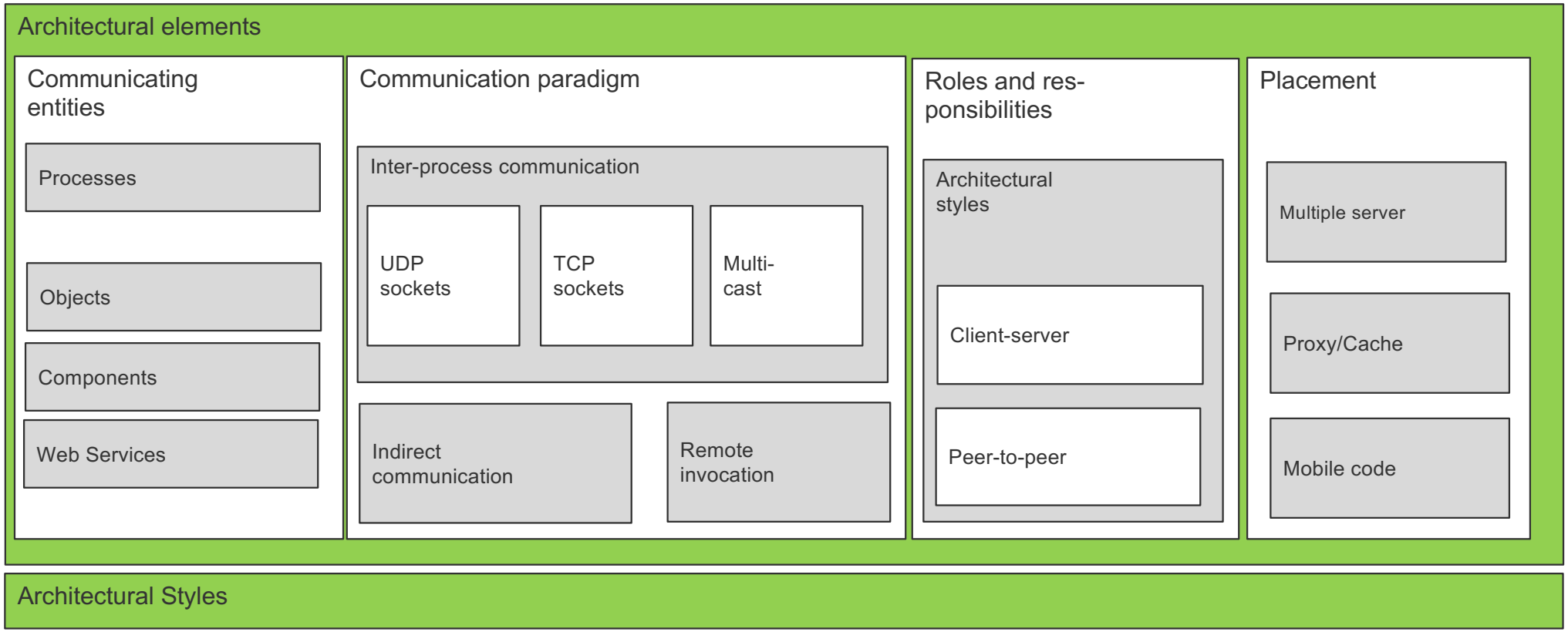


Freie Universität Berlin

A CLASSIFICATION OF DISTRIBUTED SYSTEMS

Freie Universität Berlin

Recap: Architectural Model



Freie Universität Berlin

Communication Paradigms in Distributed Systems

ARCHITECTURAL STYLES

Freie Universität Berlin

Basic Idea

A **style** is formulated in terms of

- (replaceable) components with well-defined interfaces
- the way that components are connected to each other
- the data exchanged between components
- how these components and connectors are jointly configured into a system.

Connector

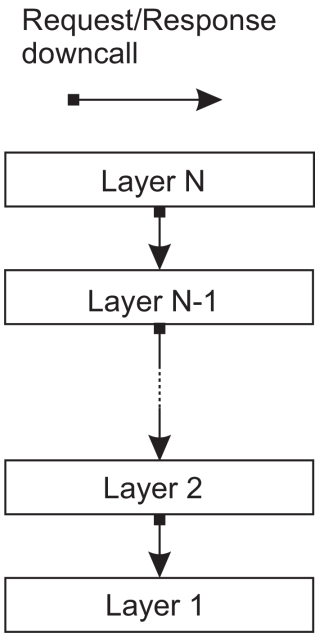
- A mechanism that mediates communication, coordination, or cooperation among components. *Example:* facilities for (remote) procedure call, messaging, or streaming.

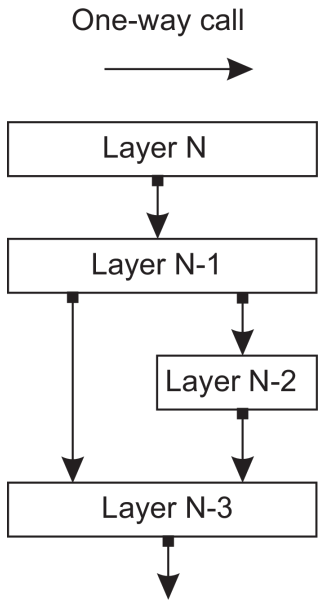
Freie Universität Berlin

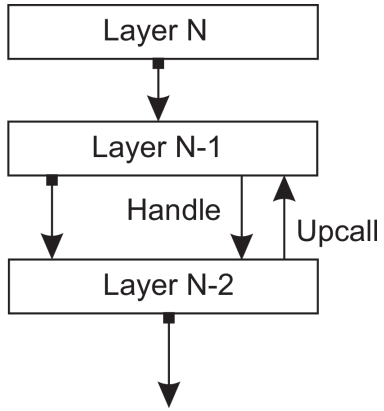
Architectural Styles in Distributed Systems

- Layered architectures
- Service-oriented architectures
- Publish-subscribe architectures

Layered architecture - Different layered organizations

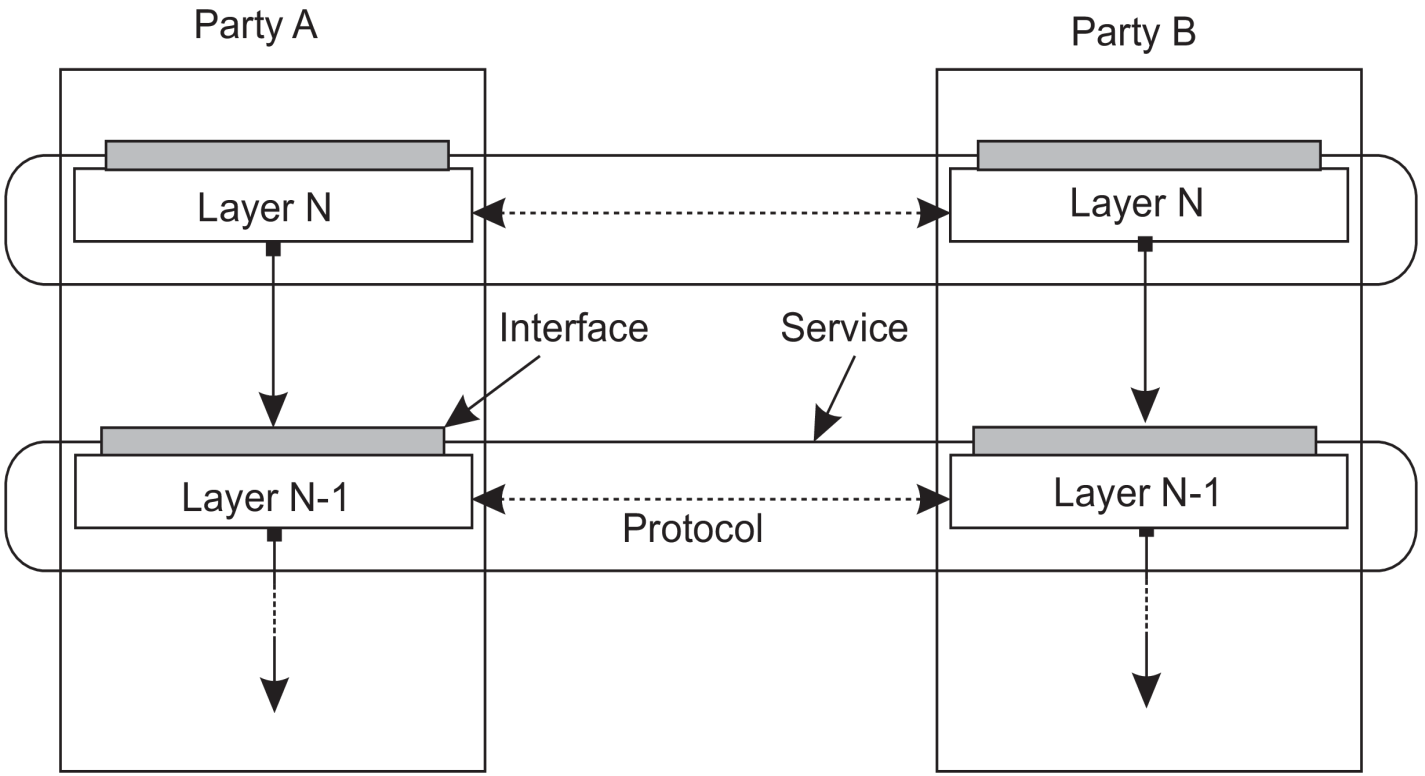






Tanenbaum & van Steen. Distributed Systems. Principles and Paradigms. 2023.

Example: Communication Protocols



Tanenbaum & van Steen. Distributed Systems. Principles and Paradigms. 2023.

Freie Universität Berlin

Application Layering

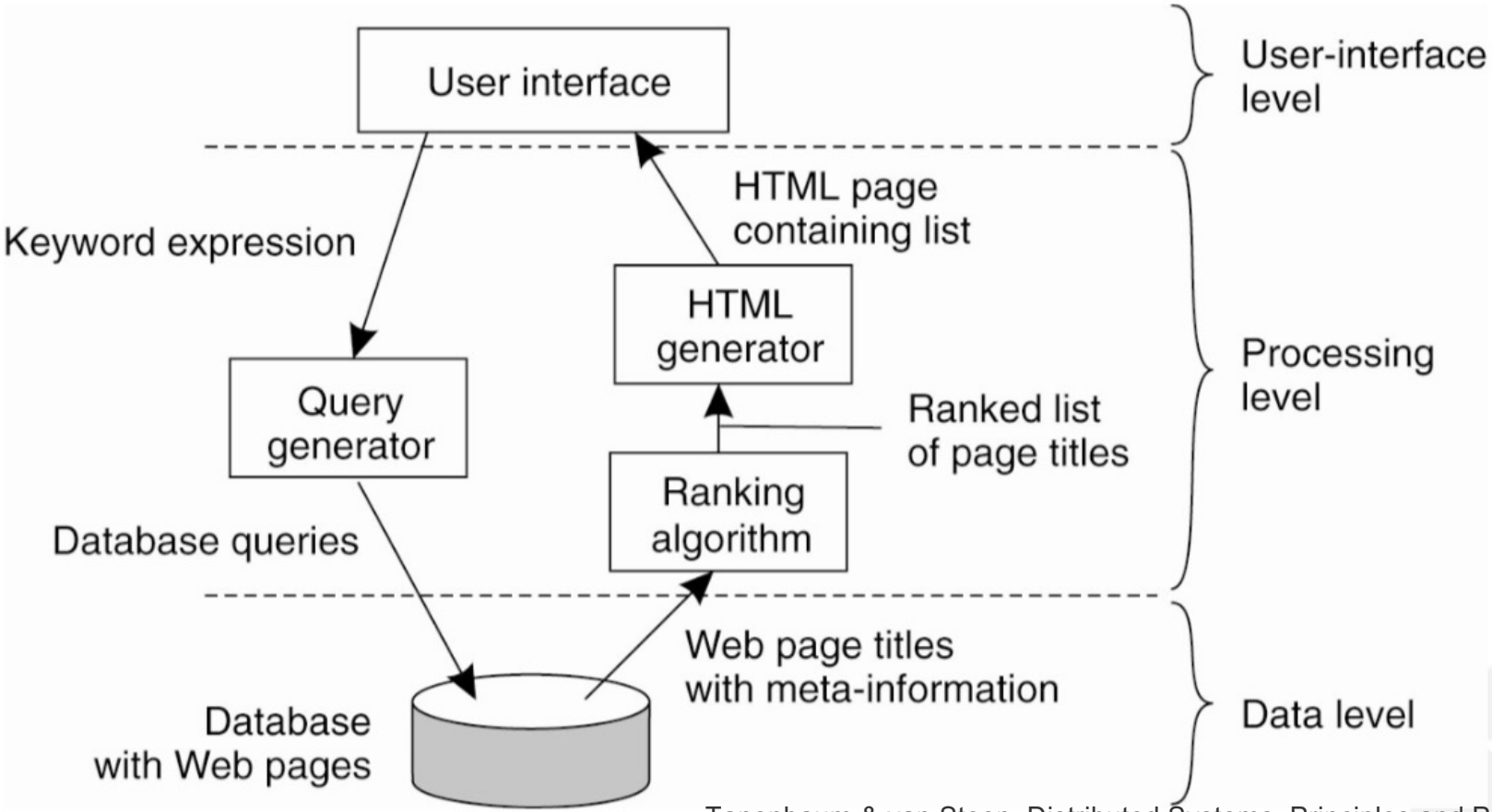
Traditional three-layered view

- The **application-interface** layer contains units for interfacing to users or external applications
- The **processing layer** contains the functions of an application, i.e., without specific data
- The **data layer** contains the data that a client wants to manipulate through the application components

Observation: This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

Tanenbaum & van Steen. Distributed Systems. Principles and Paradigms. 2023.

Example: A Simple Search Engine



Tanenbaum & van Steen. Distributed Systems. Principles and Paradigms. 2007.

Freie Universität Berlin

Service-oriented Architectures

- A layered architectural style‘s significant drawback is the often strong dependency between different layers.
- Such direct dependencies have led to an architectural style reflecting a more loose organization into a collection of separate, independent entities.
- Each entity encapsulates a service (can be called services, objects, or microservices).
- Each service is executed as a separate process (or thread).

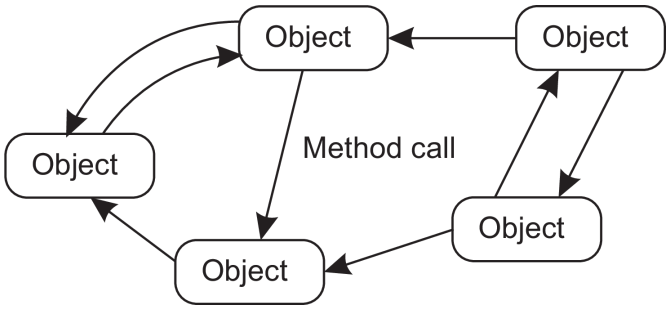
Tanenbaum & van Steen. Distributed Systems. Principles and Paradigms. 2023.

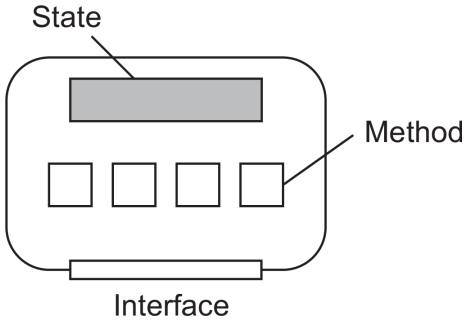
Freie Universität Berlin

Object-based style

Essence

Components are objects, connected to each other through **procedure calls**. Objects may be placed on different machines; calls can thus execute across a network.





Encapsulation

Objects are said to encapsulate data and offer methods on that data without revealing the internal implementation.

Freie Universität Berlin

RESTful architectures

Essence

View a distributed system as a collection of resources, individually managed by components. Resources may be added, removed, retrieved, and modified by (remote) applications.

1. Resources are identified through a single naming scheme
2. All services offer the same interface
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller

Basic operations

Operation	Description
PUT	Create a new resource
GET	Retrieve the state of a resource in some representation
DELETE	Delete a resource
POST	Modify a resource by transferring a new state

Freie Universität Berlin

Example: Amazon's Simple Storage Service

Essence

- Objects (i.e., files) are placed into buckets (i.e., directories). Buckets cannot be placed into buckets. Operations on ObjectName in bucket BucketName require the following identifier:

  http://BucketName.s3.amazonaws.com/ObjectName

Typical operations (all operations are carried out by sending HTTP requests):

- Create a bucket/object: PUT, along with the URI
- Listing objects: GET on a bucket name
- Reading an object: GET on a full URI

Freie Universität Berlin

On interfaces

Issue

- Many people like RESTful approaches because the interface to a service is so simple. The catch is that much needs to be done in the parameter space.

Amazon S3 SOAP interface

Bucket operations	Object operations
ListAllMyBuckets	PutObjectInline
CreateBucket	PutObject
DeleteBucket	CopyObject
ListBucket	GetObject
GetBucketAccessControlPolicy	GetObjectExtended
SetBucketAccessControlPolicy	DeleteObject
GetBucketLoggingStatus	GetObjectAccessControlPolicy
SetBucketLoggingStatus	SetObjectAccessControlPolicy

Insert: Simple Object Access Protocol (SOAP)

SOAP is designed to enable both client-server and asynchronous interaction over the Internet. It defines a scheme for using XML to represent the contents of request and reply messages and a scheme for the communication of documents.

It is used for information exchange and RPC, usually (but not necessarily) over HTTP.

(Very) basic SOAP architecture:

Freie Universität Berlin

On interfaces (*cont.*)

Simplifications

- Assume an interface bucket offering an operation create, requiring an input string such as mybucket, for creating a bucket “mybucket.”

SOAP

```
import bucket
bucket.create("mybucket")
```

RESTful

```
PUT "https://mybucket.s3.amazonsws.com/"
```

Conclusions

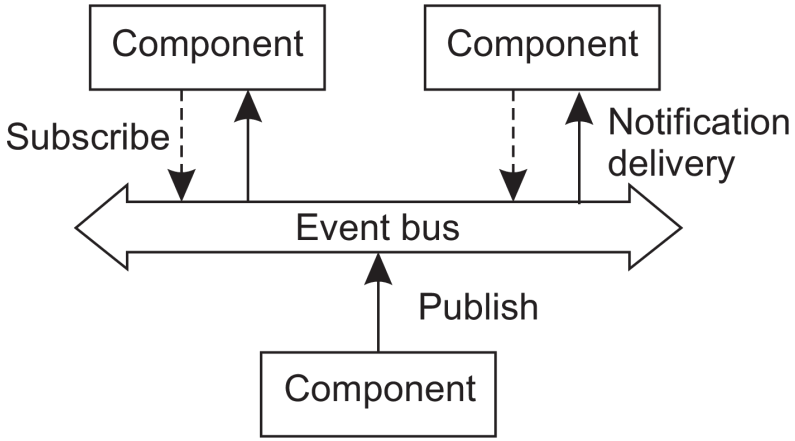
- Are there any to draw?

Freie Universität Berlin

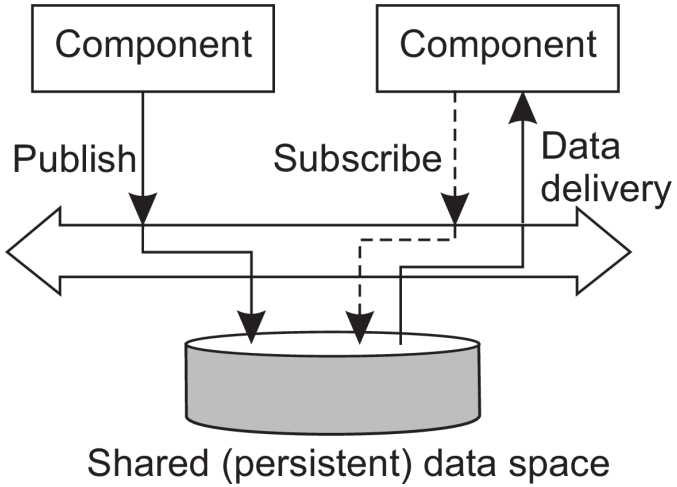
Publish-Subscribe Architectures

Temporal and referential coupling

	Temporally coupled	Temporally decoupled
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space



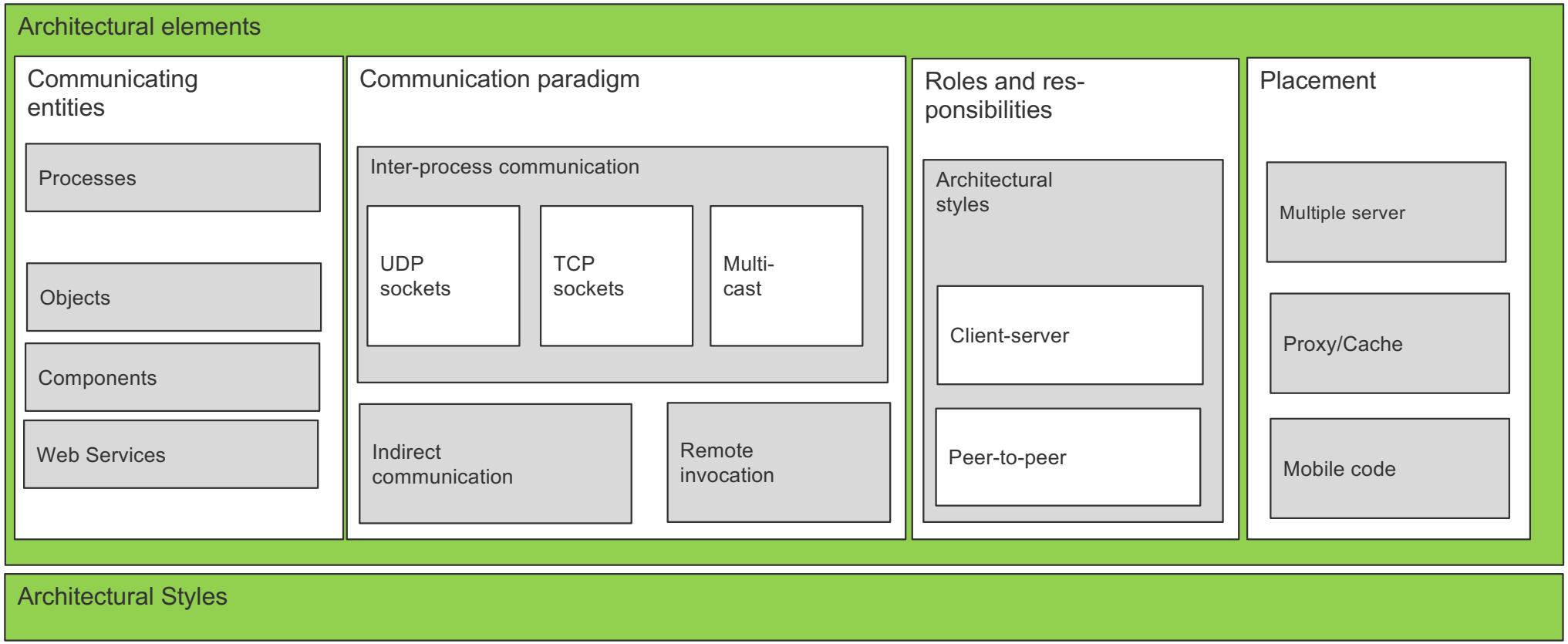
Event-based: communicating parties need to be both online



Shared data space: data to communicate can be stored

Freie Universität Berlin

Recap: Architectural Model



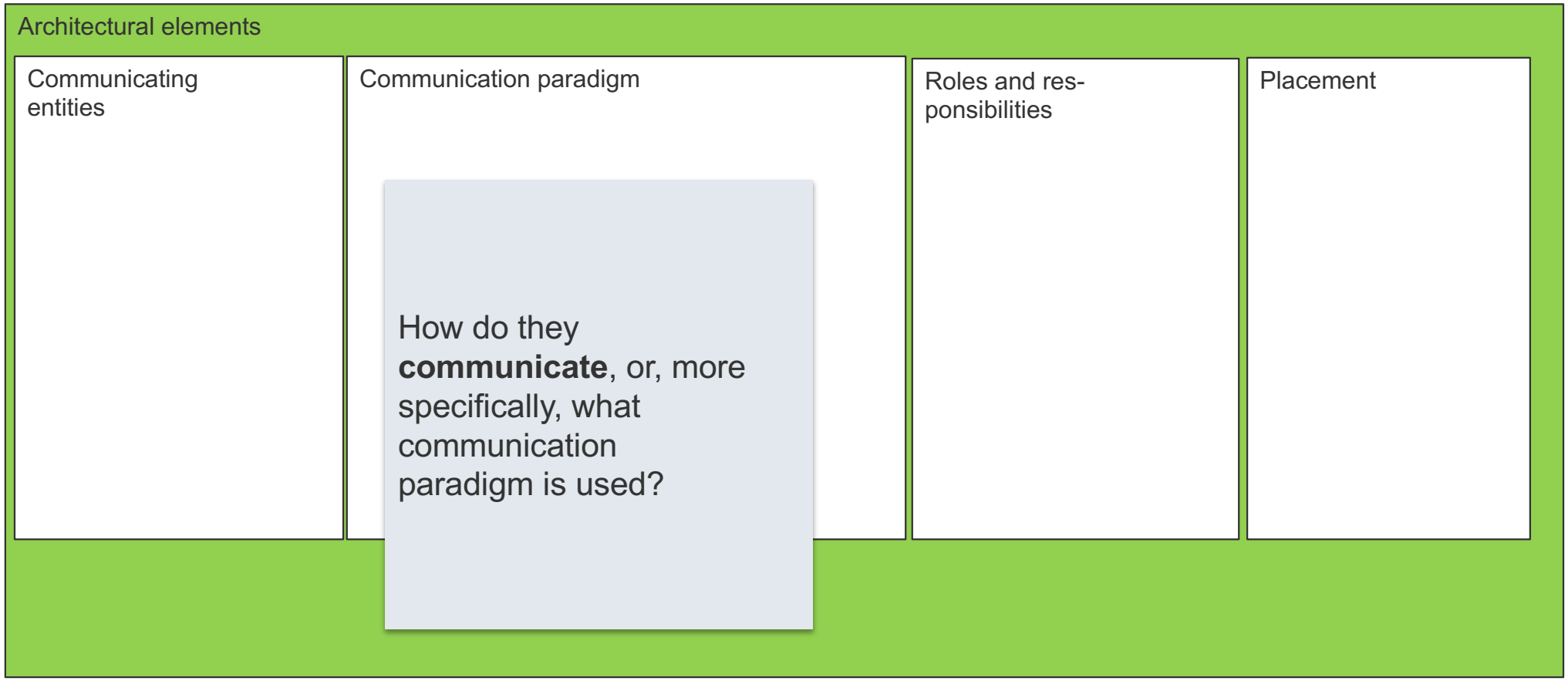
Freie Universität Berlin

Architectural elements

COMMUNICATION PARADIGM

Freie Universität Berlin

An Architectural Model of Distributed Systems



Freie Universität Berlin

Types of Communication Paradigms

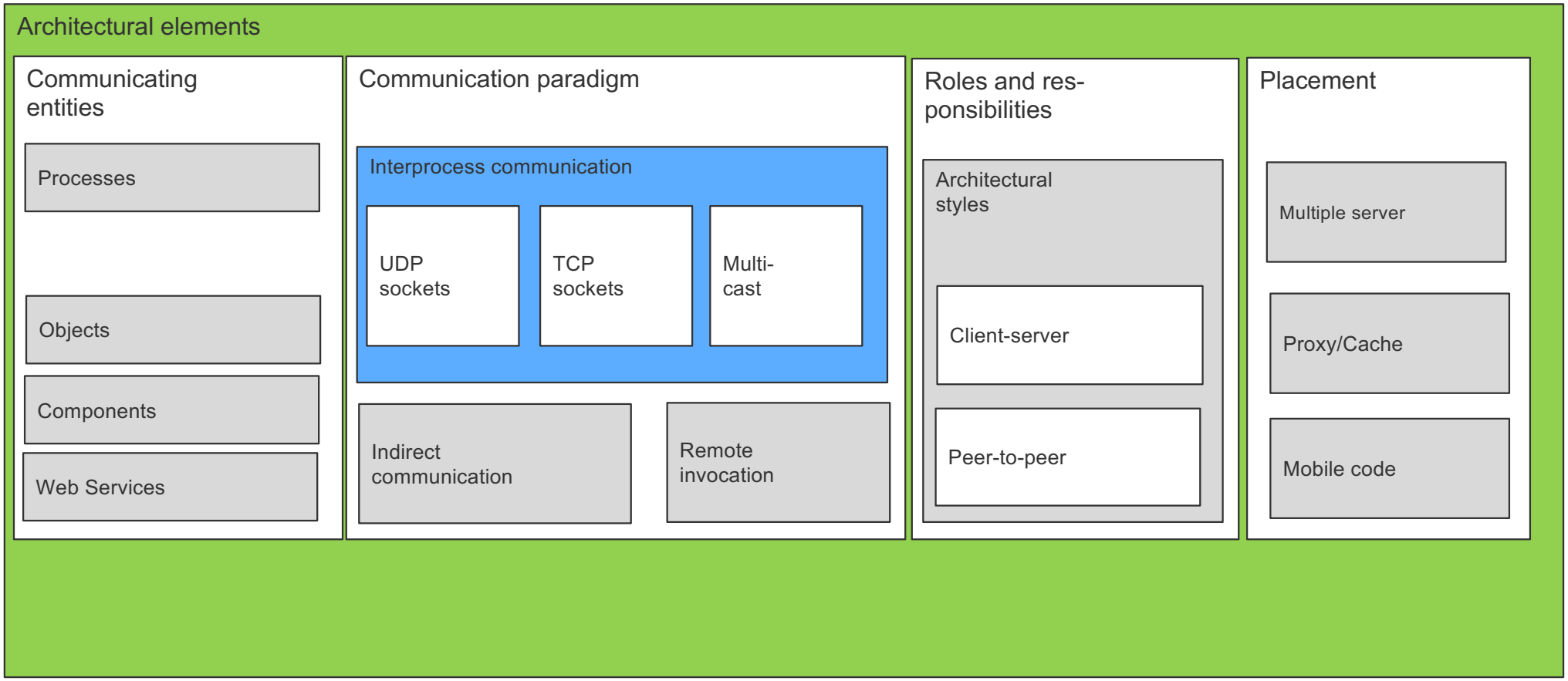
Interprocess communication

Remote invocation

Indirect communication

Freie Universität Berlin

Types of Communication Paradigms



Freie Universität Berlin

Communication Paradigm

INTERPROCESS COMMUNICATION

Freie Universität Berlin

Interprocess Communication

Interprocess Communication (IPC) mechanisms provide a low-level support to enable processes from different address spaces to connect and exchange information.

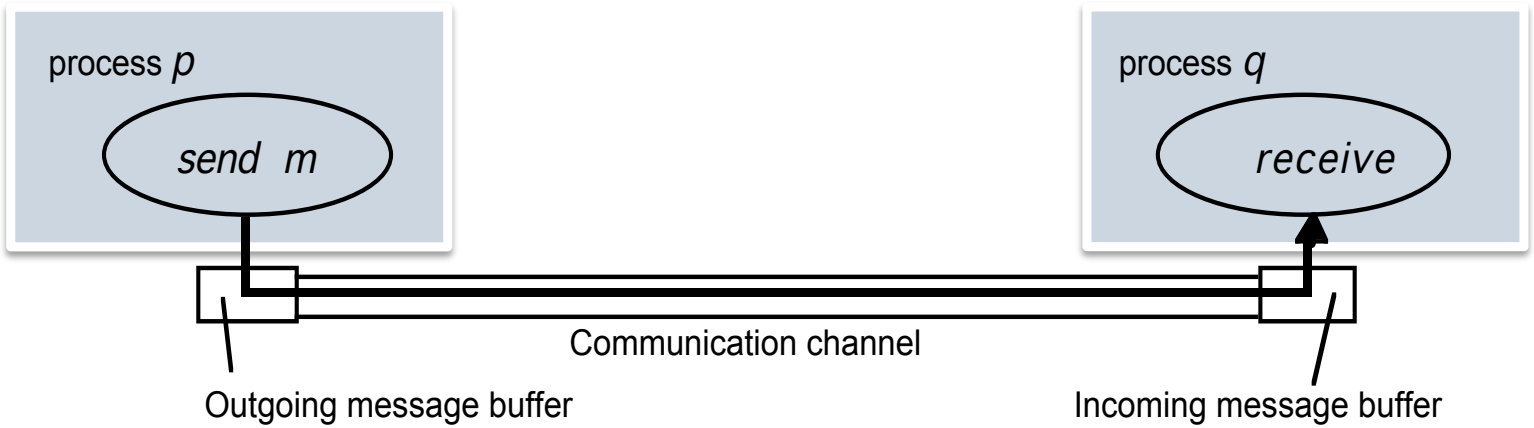
A process is an object of the operating system through which applications gain secure access to computer resources. Individual processes are isolated from each other for this purpose.

IPC is based on the exchange of messages (= bit sequences).

- A message is sent by the one process (the sender).
- It is received by another process (the receiver).

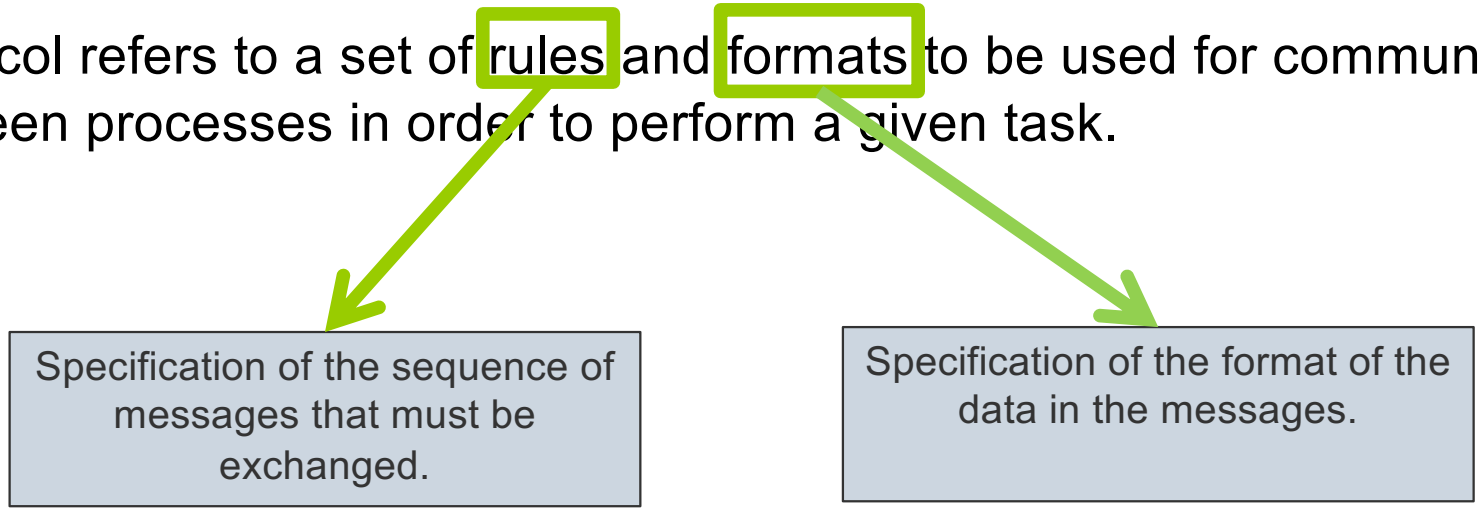
Freie Universität Berlin

Motivation for IPC

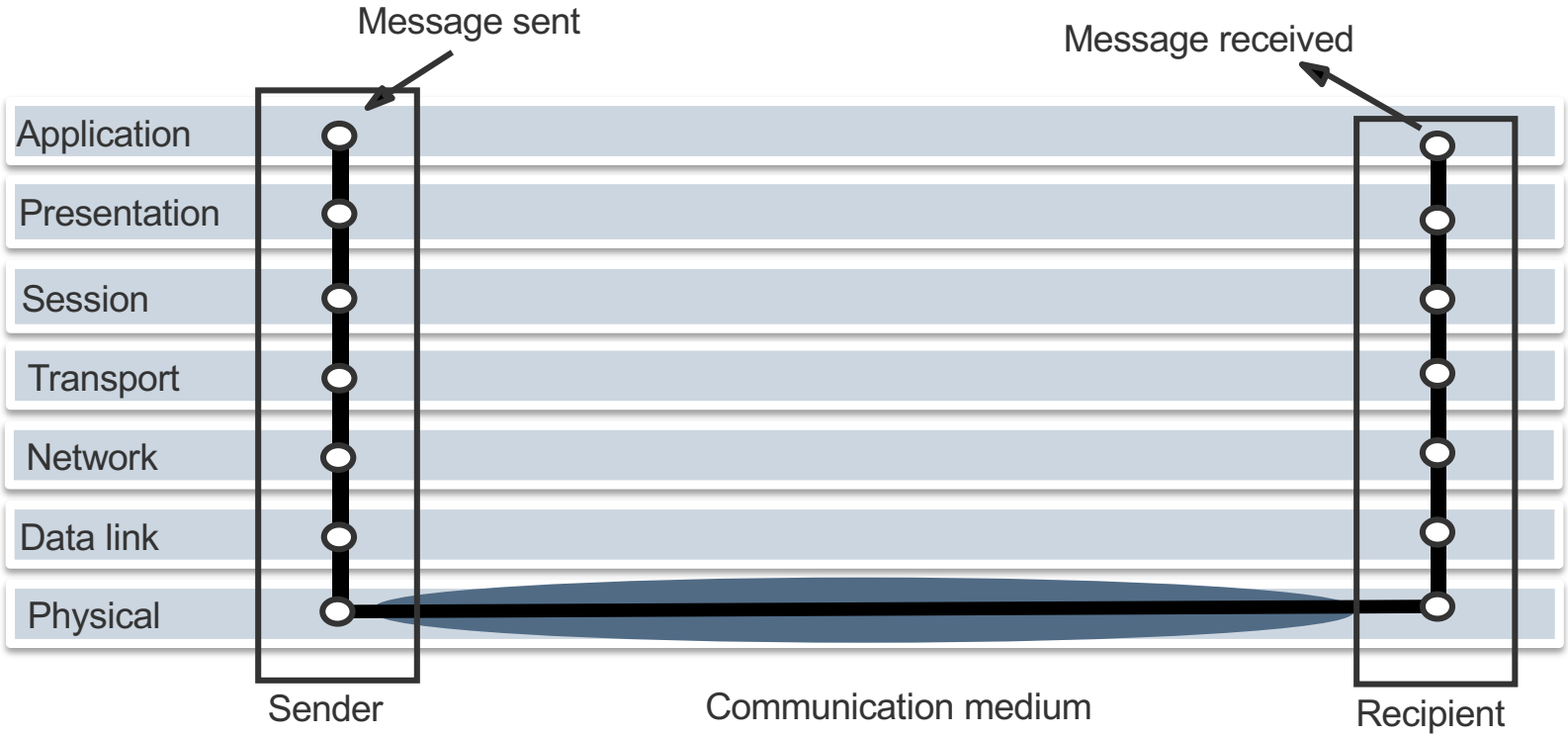


Protocols

- Protocol refers to a set of rules and formats to be used for communication between processes in order to perform a given task.



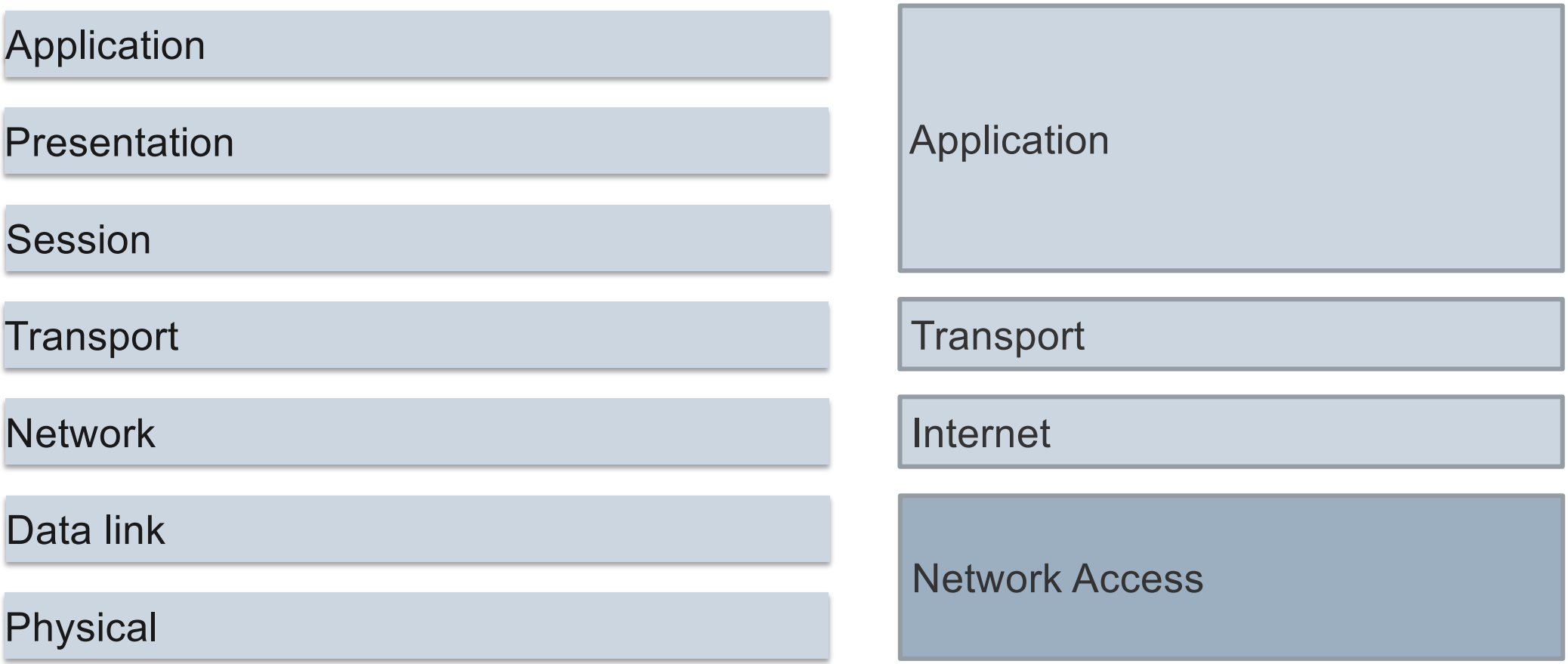
Protocol layers in the ISO Open Systems Interconnection (OSI) model



Connection-oriented communication: sending and receiving processes synchronize at every message = send and receive are blocking operation
Connectionless communication: send and receive operations are non-blocking

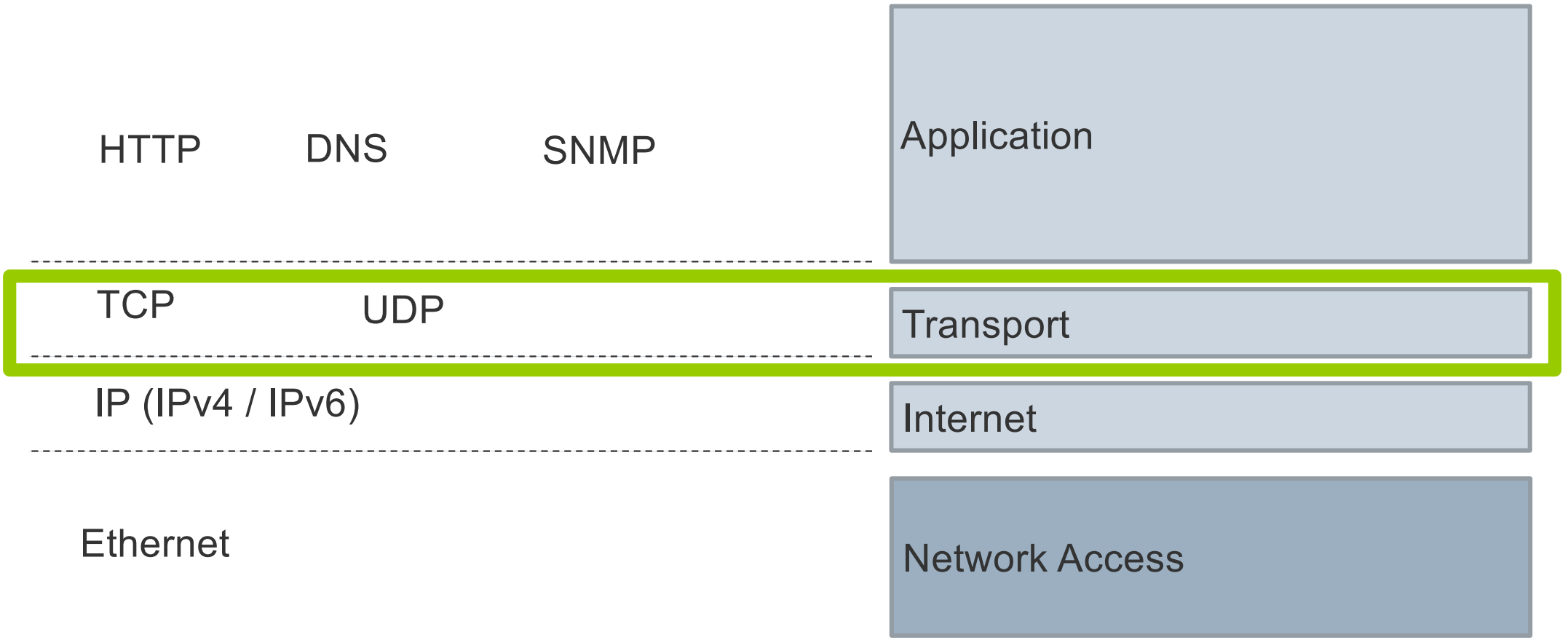
Freie Universität Berlin

Layers ISO Model vs. TCP/IP Model



Freie Universität Berlin

TCP/IP Model



Freie Universität Berlin

UDP vs. TCP

UDP (User Datagram Protocol)

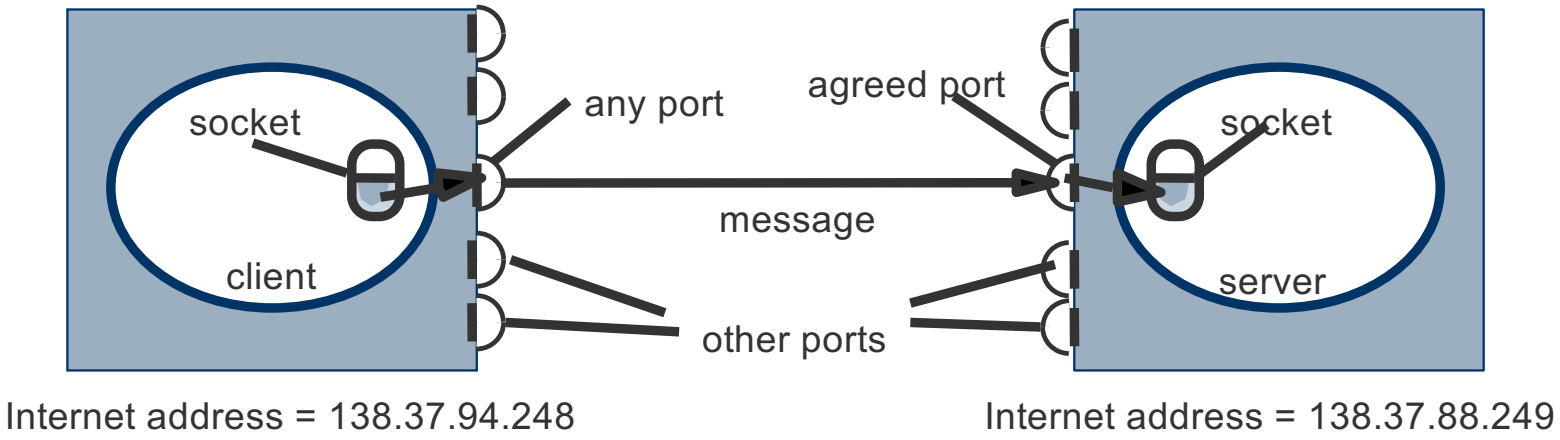
- UDP differs from the IP service only in the additional specification of the ports: a message is sent as a datagram through the network without the arrival at the destination port being guaranteed (connectionless service).

TCP (Transmission Control Protocol)

- TCP establishes a virtual connection between a client port and a provider port and thus two opposing, reliable, sequence-true (FIFO) byte streams (connection-oriented service).

Sockets

On approach to realize interprocess communication consists of transmitting a message between a socket in one process to a socket in another process.



Coulouris, Dollimore, Kindberg: *Distributed Systems: Concepts and Design*. 2011.

Freie Universität Berlin

Interprocess communication

API FOR INTERNET PROTOCOLS

Freie Universität Berlin

Java API: package java.net

Java provides class `InetAddress` that represents Internet addresses.

Method static `InetAddress getByName(String host)`

Can throw an `UnknownHostException`

Example

```
System.out.println(InetAddress.getByName("www.fu-berlin.de"));
        www.fu-berlin.de/160.45.170.10
System.out.println(InetAddress.getByName("localhost"));
        localhost/127.0.0.1
System.out.println(InetAddress.getLocalHost());
        wing.local/192.168.183.35
```

Slide adapted from Peter Löhr/Robert Tolksdorf
http://download.oracle.com/javase/6/docs/api/java/net/InetAddress.html

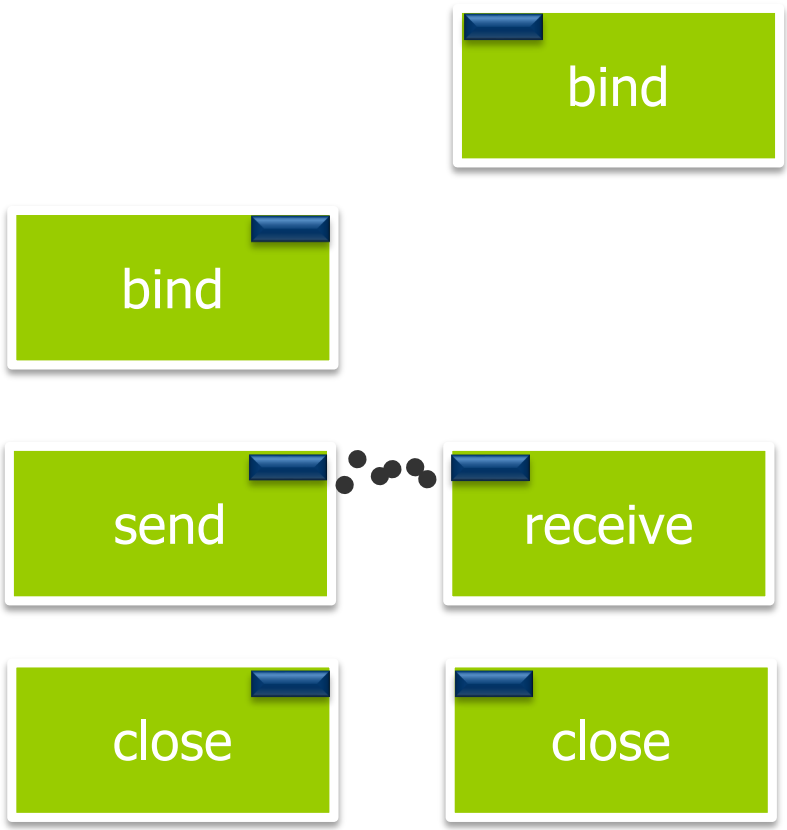
Freie Universität Berlin

API for Internet protocols

UDP DATAGRAM COMMUNICATION

UDP Sockets

1. Client creates socket bound to a local port
2. Server binds its socket to a server port
3. Client/Server send and receive datagrams
4. Ports and sockets are closed



Slide adapted from Peter Löhr/Robert Tolksdorf

Freie Universität Berlin

Using UDP for Applications

Advantage of UDP datagrams is that they do not suffer from overheads associated with guaranteed message delivery.

Example 1: Domain Name System

- DNS primarily uses UDP on port number 53 to serve requests
- DNS queries consist of a single UDP request from the client followed by a single UDP reply from the server

Example 2: VOIP

- No reason to re-transmit packets with bad speech data
- Speech data must be processed at the same rate as it is sent - there is no time to retransmit packets with errors

Freie Universität Berlin

UDP datagram communication

JAVA API FOR UDP DIAGRAMS

Freie Universität Berlin

Java API for UDP diagrams

Datagram communication is provided by two classes
`DatagramPacket` and `DatagramSocket`

`DatagramPacket`

- Constructor that makes an instance out of an array of bytes comprising a message
- Constructor for use when receiving a message, message can be retrieved by the method `getData`

`DatagramSocket`

- Constructor that takes port number as argument for use by processes
- No-argument constructor for choosing a free local port

Freie Universität Berlin

Example: UDP Echo Server

Create datagram socket at port 9876

```
import java.net.*;

class UDPServer {

    public static void main(String args[]) throws Exception {

    DatagramSocket serverSocket = new DatagramSocket(9876);

    byte[] receiveData = new byte[1024];
    byte[] sendData = new byte[1024];

    ...
```

```
...
        while (true) {
            DatagramPacket receivePacket = new
                        DatagramPacket(receiveData,
                        receiveData.length);

            serverSocket.receive(receivePacket);

            String sentence = new String(receivePacket.getData());
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();

            String capitalizedSentence = sentence.toUpperCase();

            sendData = capitalizedSentence.getBytes();

            DatagramPacket sendPacket = new DatagramPacket(sendData,
                    sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

Create space for new datagram

Receive datagram

Get IP addr port #, of sender

Create datagram to send to client

Example: Java Client (UDP)

```
import java.io.*;
import java.net.*;


class UDPClient {
    public static void main(String args []) throws Exception {

        BufferedReader inFromUser = new BufferedReader(new
            InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();

        InetAddress IPAddress =
InetAddress.getByName("localhost");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
```

Create input stream

Create client socket

Translate host-name to IP address using DNS

Freie Universität Berlin

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
. . .
            sendData = sentence.getBytes();
            DatagramPacket sendPacket = new
                                        DatagramPacket(sendData,
                    sendData.length, IPAddress, 9876);
            clientSocket.send(sendPacket);

            DatagramPacket receivePacket = new
                                        DatagramPacket(receiveData,
                    receiveData.length);
            clientSocket.receive(receivePacket);

            String modifiedSentence = new
                                    String(receivePacket.getData());

            System.out.println("FROM SERVER: " + modifiedSentence);

            clientSocket.close();
        }
    }
```

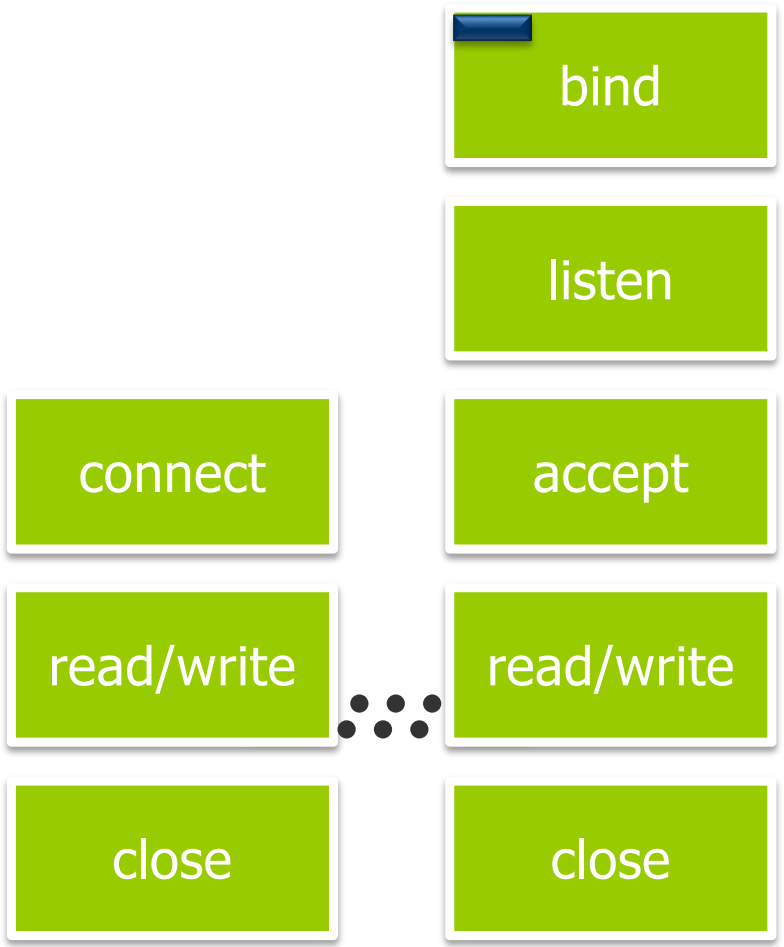
Freie Universität Berlin

API for Internet protocols

TCP STREAM COMMUNICATION

TCP Sockets Communication

1. Server bind port
2. Server is ready and listening
3. Server is waiting for request, client sends request, server accepts
4. Client and server are connceted - bidirectional!
5. Connection is closed



Freie Universität Berlin

Use of TCP for Applications

Many frequently used services run over TCP connections with reserved port numbers.

- **HTTP** [RFC 2068]: The Hypertext Transfer Protocol is used for communication between web browser and web server.
- **FTP** [RFC 959]: The File Transfer Protocol allows directories on a remote computer to be browsed and files to be transferred from one computer to another over a connection.
- **Telnet** [RFC 854]: Telnet provides access by means of a terminal session to a remote computer.
- **SMTP** [RFC 821]: The Simple Mail Transfer Protocol is used to send mail between computer.

http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Freie Universität Berlin

TCP Stream Communication

JAVA API FOR TCP

Freie Universität Berlin

Java API for TCP streams

Java interface provides two classes `ServerSocket` and `Socket`

`ServerSocket`

- Class is intended to be used by server to create a socket at a server port for listening for connect requests from clients.

`Socket`

- Class is for use by a pair of processes with a connection
- The client uses a constructor to create a socket, specifying the DNS hostname and port of a server

Freie Universität Berlin

TCP Echo Server

```
public class EchoServer {
public static void main(String args[]) throws IOException {

    ServerSocket listen = new ServerSocket(1234);

    while (true) {
        Socket socket = listen.accept();
        BufferedReader in = new BufferedReader(new
                            InputStreamReader(socket.getInputStream()));
        PrintStream out = new PrintStream(socket.getOutputStream());
        . . .
```

Create Server Socket

Listens for a connection and accepts it.

Input-Stream

Output-Stream

```
while (true) {
    String message = in.readLine();

    if (message == null) {
            break;
    }
    String answer = message.replace('i', 'o');
    out.println(answer);
}
in.close();
out.close();

socket.close();
System.out.println("Socket closed.");

}}}
```

Read Input stream

Send Output stream

TCP Client

```
public class Client {

public static void main(String args[]) throws IOException {

    Socket socket = new Socket("localhost", 1234);

    PrintStream out = new PrintStream(socket.getOutputStream());

    BufferedReader in = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));

    BufferedReader keyboard = new BufferedReader(new
            InputStreamReader(System.in));

. . .
```

Create Socket And connect to server

BufferedReader reads Keyboard Input

```
            while (true) {
                    String message = keyboard.readLine();

                    if (message == null)
                            break;

                    out.println(message);
                    String answer = in.readLine();
                    System.out.println("echo: " + answer);
            }

        in.close();
        out.close();
        socket.close();
        }
}
```

Send text via output stream

Freie Universität Berlin

Notice!

However, the service `echo` is quite limited in that it cannot have several sessions at the same time. If you want to use the service, you may have to wait until an active user closes the session.

What might be a solution?

Freie Universität Berlin

Example EchoServer Extended

```
public class EchoServerExtended extends Thread {
    private Socket socket;
    private BufferedReader in;
    private PrintStream out;


    public EchoServerExtended(Socket socket) throws IOException{
        this.socket = socket;
        this.in = new BufferedReader(new
                  InputStreamReader(socket.getInputStream()));
        this.out = new PrintStream(socket.getOutputStream());
    }
```

Freie Universität Berlin

Summary…

Sockets only provide the basic mechanisms, there is still work to be done, for example the implementation of more complex system models such as Request-Reply (for client-server) or group communication.

Above all, however, the necessity of homogeneous data representation in heterogeneous environments is a major issue.

These are the basic techniques for more complex middleware such as RPC, Java RMI. We will talk about it next ☺

Freie Universität Berlin

Open Topics in IPC – we discuss them in the next lecture

- External data representation and marshalling
- Multicast communication
- Network virtualization: Overlay networks

Freie Universität Berlin

References

Main resources for this lecture:

- George Coulouris, Jean Dollimore, Tim Kindberg: *Distributed Systems: Concepts and Design*. 5th edition, Addison Wesley, 2011.
- Andrew S. Tanenbaum and Marteen van Steen.Distributed Systems. Principles and Paradigms.Pearson Prentice Hall, 2nd edition, 2007.
- Marteen van Steen and Andrew S. Tanenbaum. Distributed Systems. Principles and Paradigms.Pearson Prentice Hall, 4th edition, 2023.

Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

Algorithms and Programming IV

Remote Invocation: Remote Procedure Calls

Summer Term 2023 | 21.06.2023

Claudia Müller-Birn, Barry Linnert