TECHNISCHE HOCHSCHULE LÜBECK

DISTRIBUTED SYSTEMS

05.04.2019

COMMUNICATION

LENKA KLEINAU, M.SC.

SLIDES BASED ON MATERIAL FROM PROF. KROHN

CONTENT

TECHNISCHE HOCHSCHULE LÜBECK

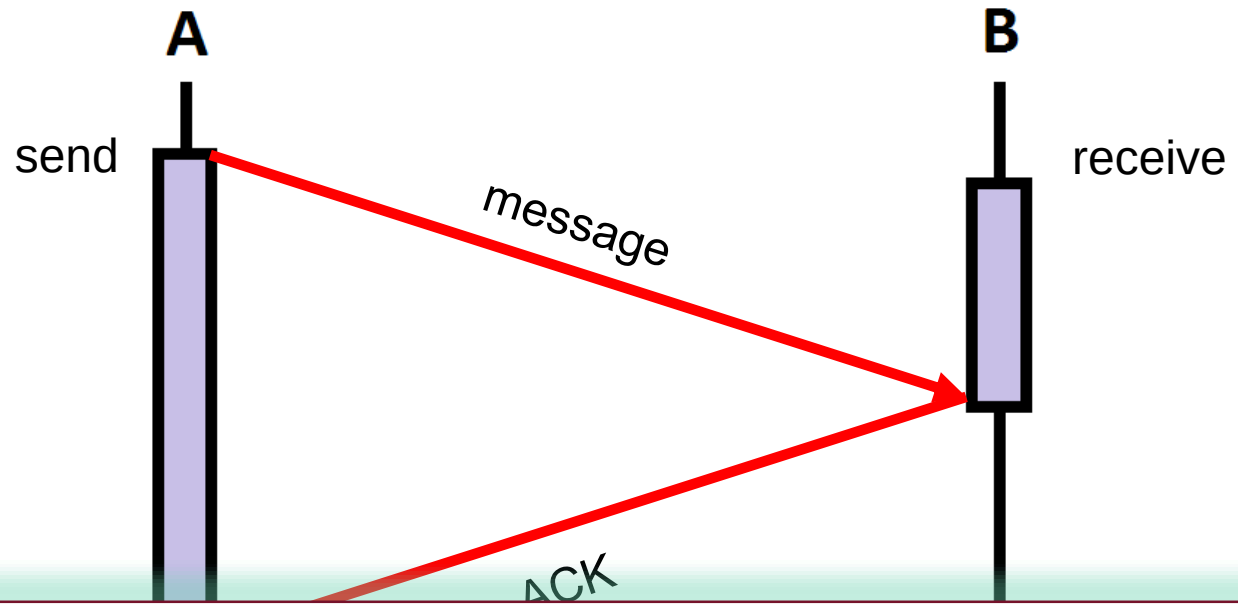
- Repetition
- Data transmission in distributed systems
- Java object serialization
  - Exercise
- Java file handling exercise

TECHNISCHE HOCHSCHULE LÜBECK

- **Communication in Distributed Systems**
  → between software and processes
- **Process**
  - Client/Server
  - Sender/Receiver
  - Producer/Consumer
  - Peers
- **Communication models → characterizing communication in distributed systems**
  - Synchronous vs. asynchronous
  - Direct vs. indirect
  - Persistent vs. transient

TECHNISCHE HOCHSCHULE LÜBECK

- **A** sends message
- **A** waits until the message was sent or..
- until **B** confirms the reception



Called „Rendezvous“

Sender/Receiver are synchronized automatically

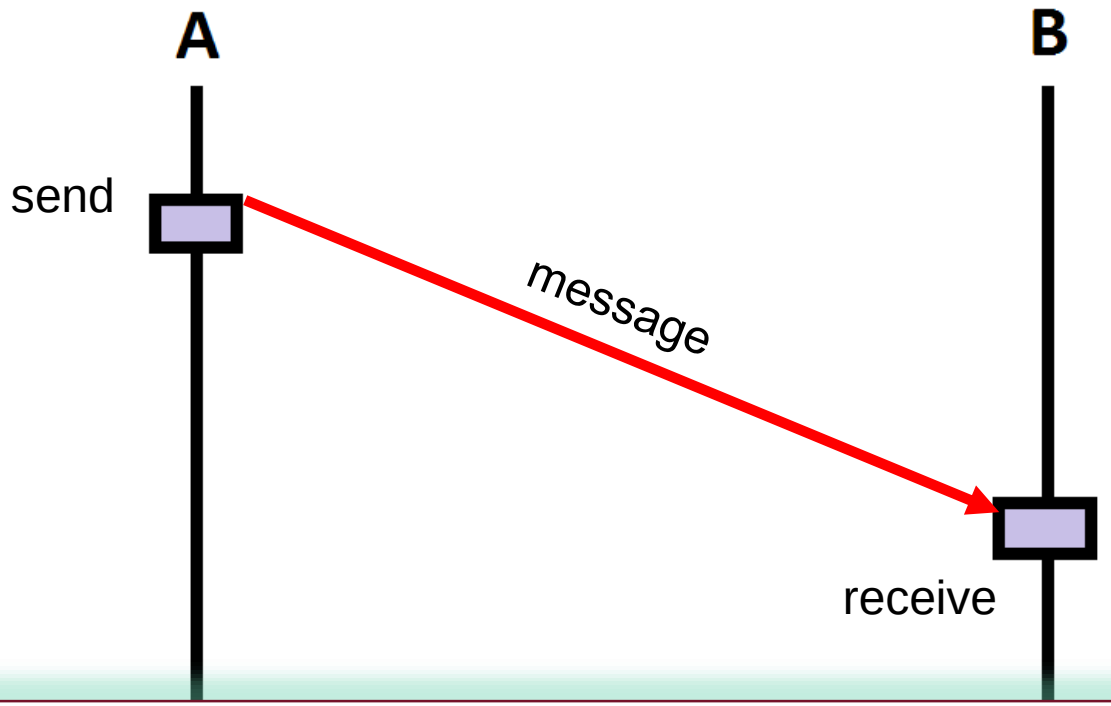
Sender can block activity until ACK is received (possibility of blocking forever!)

Blocking

ASYNCHRONOUS COMMUNICATION

TECHNISCHE HOCHSCHULE LÜBECK

- **A** sends message
- **A** resumes execution as soon as the message was sent



Called „Datagram“

Parallel activities of Sender/Receiver possible
Buffer needed for reliability of communication

Blocking

TECHNISCHE HOCHSCHULE LÜBECK

Direct addressing

Symmetric addressing

```
send (B, message);
receive (A, message);
```

Asymmetric addressing

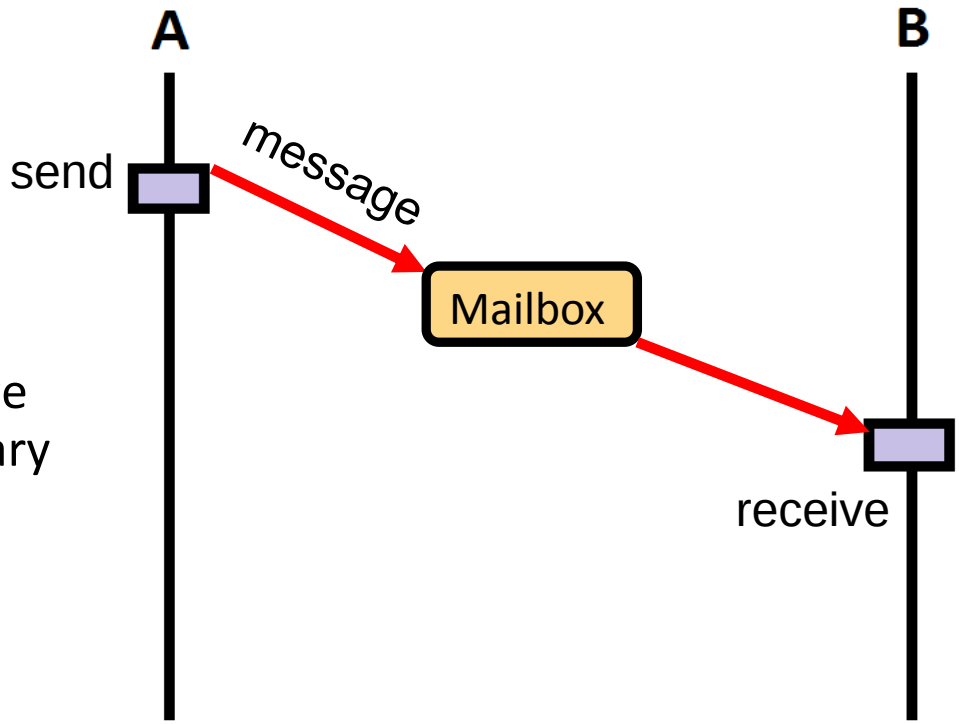
```
send (B1, message);
send (B2, message);
receive (Bi, message);
```

Indirect addressing

Sender transmits message addressing an intermediary

E.g. Mailbox

More flexible than direct addressing!



- **Persistent communication**
  - Sender transmits message to an intermediary
  - Message is held by intermediary until delivering is possible
  - Sender/Receiver don’t need to be active simultaneously!
    - Sender can terminate after transmission
    - Receiver gets message on reactivation
  - Buffer overflow due to too many messages at intermediary possible

- **Transient communication**
  - Messages only exist while Sender and Receiver are active!
  - Communication or Receiver errors cause loss of message
  - Maintenance easy
  - Memory usage low

TECHNISCHE HOCHSCHULE LÜBECK

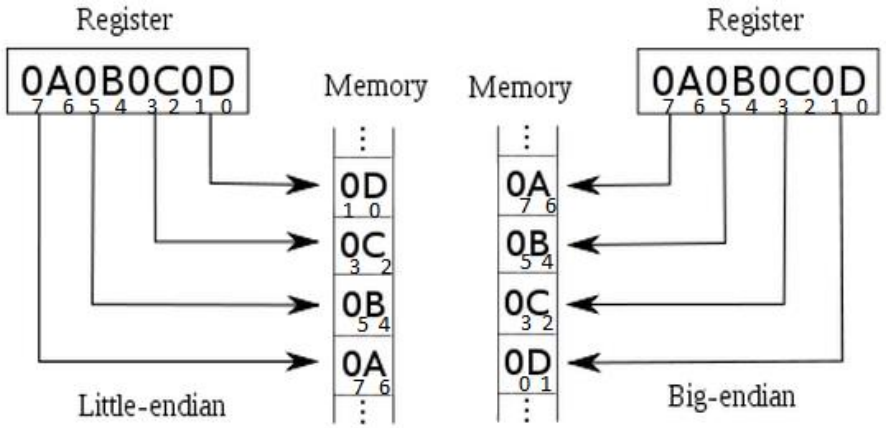
- **Data transmission in distributed systems**
  - Data structures must be flattened (converted to sequence of bytes) before transmission
  - Rebuild of data structures on arrival

TECHNISCHE HOCHSCHULE LÜBECK

- **Marshalling**
  - Process of taking a collection of data items and assembling them into a form suitable for transmission in a message
- **Unmarshalling**
  - Process of disassembling data on arrival to produce an equivalent collection of data items at the destination

TECHNISCHE HOCHSCHULE LÜBECK

- Not all computers store primitive values such as integer in same order!
- **Floating point representation** varies on different architectures
- Big-endian, little-endian
  - send 4 byte using *network byte order*
- **ASCII character coding**
  - one byte per character
- **Unicode**
  - two bytes per character



http://upload.wikimedia.org/wikipedia/en/7/77/Big-little_endian.png

TECHNISCHE HOCHSCHULE LÜBECK

- **Binary data exchange possible with:**
  - Convert values into agreed external format before transmission
  - Convert to local format after reception

  OR
  - Transmission in sender's format, additional indication of which format is used
  - Converting at receiver side if necessary

- **Agreed standard for representation is called "external data representation"**

TECHNISCHE HOCHSCHULE LÜBECK

- **Alternative approaches for external data representation and marshalling:**
  - CORBA's common data representation (discussed later in Prof. Heeren's lecture)
  - XML, textual format for representing structured data (discussed in "Information Systems")
  - **Java's object serialization**

TECHNISCHE HOCHSCHULE LÜBECK

- **Allowing instances of objects to be serialized: need for implementation of Serializable**

```
public class Person implements Serializable {

        private String birthPlace;
        private int age;

        public Person (String bp, int a) {
                this.birthPlace = bp;
                this.age = a;
        }

}
```

TECHNISCHE HOCHSCHULE LÜBECK

- **Serialization:** flattening object into serial form

- **Deserialization:** restoring state of object from serialized form
  - Information about the class of each serialized object included

    → Loading of appropriate class possible

- Contained references to other objects are included in the serialized version of the object, as well as the objects themselves

- **Reflection**
  - Ability to inspect and dynamically call e.g. classes and functions at runtime
  - Enables creating classes from their names
  - Good examples here

- **Java object serialization uses reflection to find out the class names, types and values**

  → All what is needed to serialize/deserialize!

TECHNISCHE HOCHSCHULE LÜBECK

	Byte based		Character based	
	Input	**Output**	**Input**	**Output**
Basic	InputStream	OutputStream	Reader	Writer
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream	FileOutputStream	FileReader	FileWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Strings			StringReader	StringWriter
Objects	ObjectInputStream	ObjectOutputStream		

Adapted from http://tutorials.jenkov.com/java-io/overview.html

TECHNISCHE HOCHSCHULE LÜBECK

- **Serializing an object into a file (ObjectOutputStream)**
- Deserializing an object from a file (ObjectInputStream)

1. Create project in Eclipse named e.g. “YourName_JOS”
2. Create class *Student*, fields *studentNumber* and *degreeCourse*, create constructor using the fields
3. Make the class serializable
4. Create class *TestSerializing*
5. Create *FileOutputStream* for writing into file

TECHNISCHE HOCHSCHULE LÜBECK

- **Serializing an object into a file (ObjectOutputStrream)**
- Deserializing an object from a file (ObjectInputStream)

6. Create *ObjectOutputStream* using *FOS* for serializing
7. Serialize object *Student* by writing it on *OOS*
8. Close streams

TECHNISCHE HOCHSCHULE LÜBECK

- Serializing an object into a file (ObjectOutputStrream)
- **Deserializing an object from a file (ObjectInputStream)**

1. Create class *TestDeserialize*
2. Create *FileInputStream* using the created text file from serialization
3. Create *ObjectInputStream* using *FIS*
4. Read *Student* object from *OIS*
5. Close streams
6. Print *Student* object (including the fields)
7. Upload your project containing both parts!

TECHNISCHE HOCHSCHULE LÜBECK

Next lecture:

12.04.2019, 10:00-11:30, room 1-1.11

Questions?

Lenka Kleinau
Contact: lenka.kleinau@th-luebeck.de
Room 18-2.02