Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

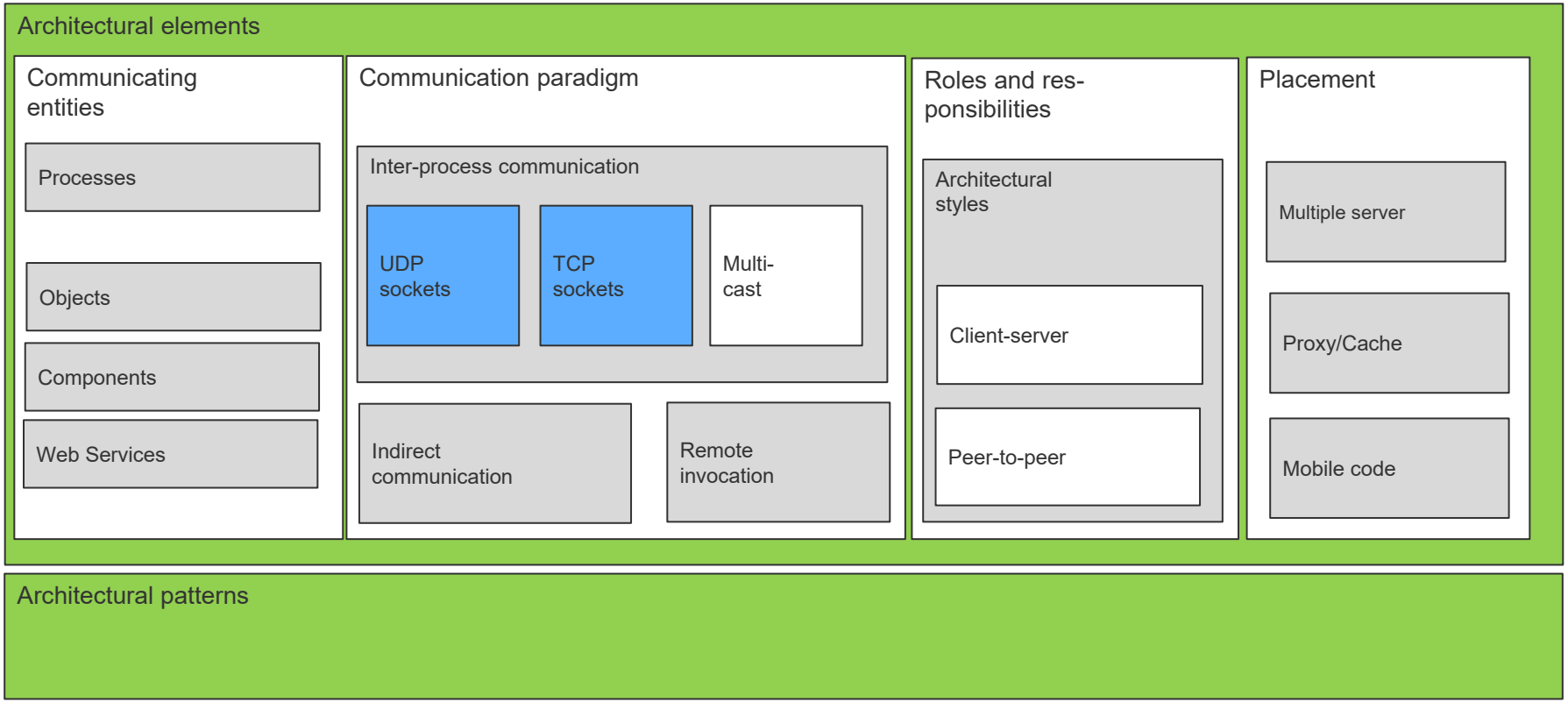
Algorithms and Programming IV

Interprocess Communication (*cont.*)

Summer Term 2021 | 14.06.2021

Barry Linnert

Recap: Architectural Model



Our topics today

Interprocess Communication

- External Data Representation and Marshalling
- Multicast Communication

Remote Invocation

- Request-and-Reply Protocolls
- Remote Procedure Call

Freie Universität Berlin

Interprocess Communication

EXTERNAL DATA REPRESENTATION AND MARSHALLING

What is the Challenge?

Messages consist of sequences of bytes.

Interoperability problems

- Big-endian, little-endian byte ordering
- Floating point representation
- Character encodings (ASCII, UTF-8, Unicode, EBCDIC)

So, we must either:

- Have both sides agree on an external representation or
- transmit in the sender's format along with an indication of the format used. The receiver converts to its form.

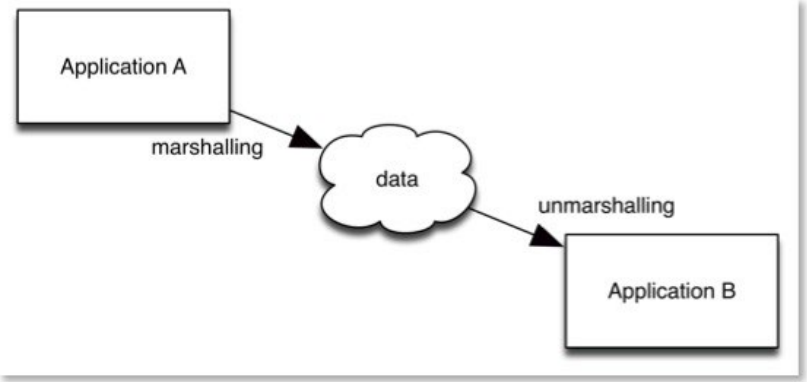
External Data Representation and Marshalling

External data representation

- An agreed standard for the representation of data structures and primitive values

Marshalling

- The process of taking a collection of data items and assembling them into a form suitable for transmission in a message



http://www.breti.org/tech/files/b400feb80f01f69e5cafca5160be5d65-67.html

Unmarshalling

- Is the process of disassembling them on arrival into an equivalent representation at the destination

Approaches for External Data Representation

Java's object serialization

XML (Extensible Markup Language)

Protocol buffer (protobuf)

JSON (JavaScript Object Notation)

Java Object Serialization

```
public class Person implements Serializable {
        private String name;
        private String place;
        private int year;
        public Person(String aName, String aPlace, int aYear) {
                name = aName;
                place = aPlace;
                year = aYear;
        }
        // followed by methods for accessing the instance variables
}
```

A class implements the Serializable interface (which is provided in the java.io package) has the effect of allowing its instances to be serialized.

Example: Java Object Serialization

```
Person p = new Person("Smith", "London", 1984);
```

Serialized values

Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name:	java.lang.String place:	*number, type and name of instance variables*
1984	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers; h0 and h1 are handles

Extensible Markup Language (XML)

- XML is a markup language that was defined by the World Wide Web Consortium (W3C) for general use for writing structured documents for the Web.

- XML data items are tagged with ‘markup’ strings. The tags are used to describe the logical structure of the data and to associate attribute-value pairs with logical structures. For a specification of XML, see the pages on XML provided by W3C [www.w3.org VI].

- XML is used to enable clients to communicate with web services and for defining the interfaces and other properties of web services.

Example: XML definition with namespace

```
<person pers:id="123456789" xmlns:pers = "http://www.nonsense.net/person">
        <pers:name> Smith </pers:name>
        <pers:place> London </pers:place >
        <pers:year> 1984 </pers:year>
        <!-- a comment -->
</person>
```

Example: XML schema

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions      >
        <xsd:element name= "person" type ="personType" />
            <xsd:complexType name="personType">
                <xsd:sequence>
                        <xsd:element name = "name"  type="xs:string"/>
                        <xsd:element name = "place"  type="xs:string"/>
                        <xsd:element name = "year"  type="xs:positiveInteger"/>
                </xsd:sequence>
                <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
            </xsd:complexType>
</xsd:schema>
```

Google Protocol Buffer

- Google Protocol Buffer (protobuf) is a common serialization format for storing and interchanging all kinds of structured information. It serves as a basis for a remote procedure call (RPC) system that is used for nearly all inter-machine communication at Google.

- The goal of Protocol Buffer is to provide a language- and platform-neutral way to specify and serialize data, it has been released as open source.

- Protobuf is 3-10 times smaller than an XML and 10-100 times faster than an XML.

http://code.google.com/apis/protocolbuffers

JSON (JavaScript Object Notation)

- JavaScript Object Notation (JSON) is a language-independent data format.
- It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data.
- Example:

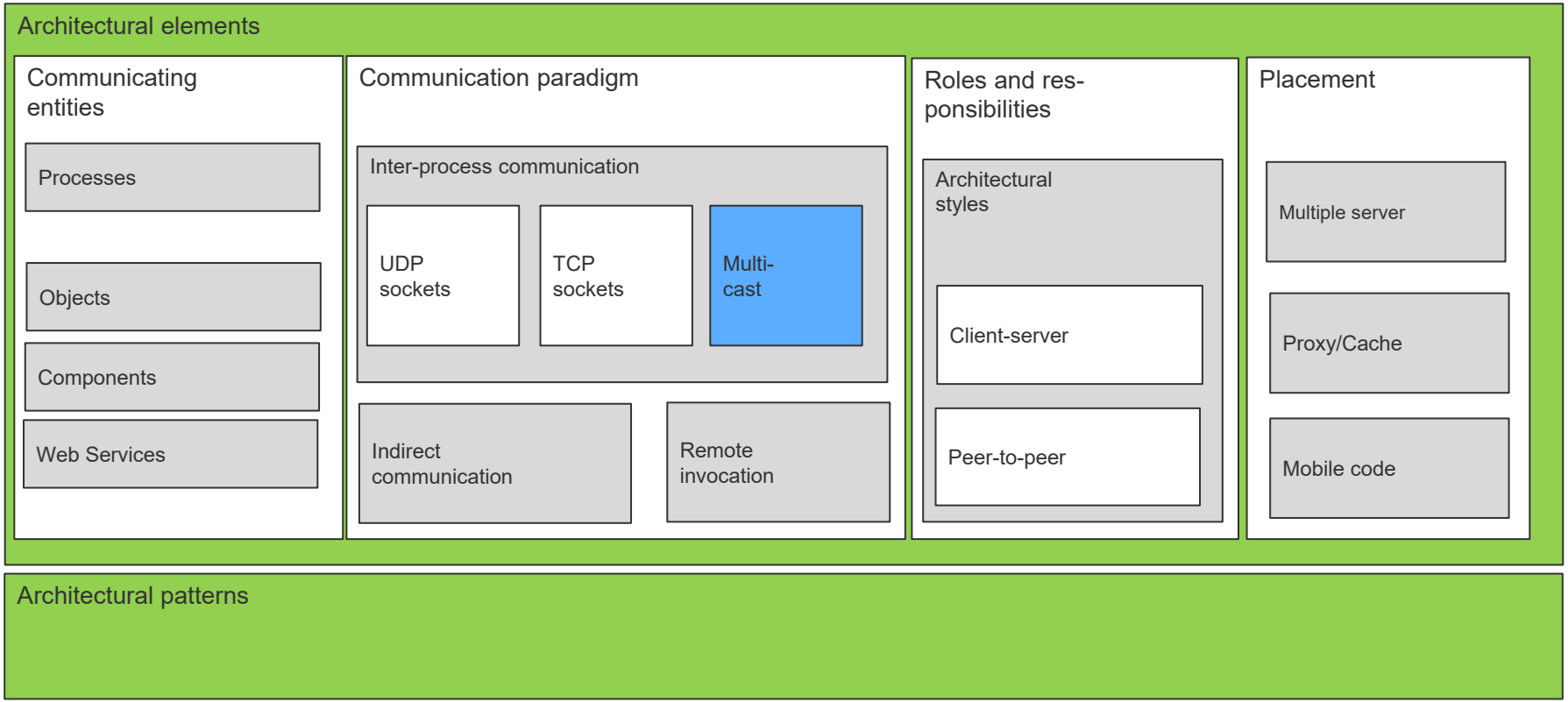
```
{
    "firstName": "John",
    “lastName": "Smith",
    “birthyear”: “1984”,
    "address": {"city": "New York", "state": "NY},
}
```

Comparison of Data-Serialization Formats

	Standardized	Binary	Human-Readable	Standard-API
Java	Yes	Yes	No	Yes
XML	Yes	Partial	Yes	Yes
protobuf	No	Yes	Partial	For example, C++, Java, C#, Python, Ruby, C, PHP, R
JSON	Yes	No	Yes	Partial (JSON-LD)

Source: https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats

Recap: Architectural Model

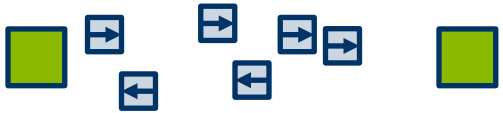


Interprocess Communication

MULTICAST COMMUNICATION

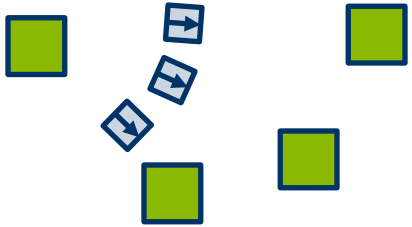
Possiblities to Communicate

- Connectionless 1:1
  UDP



- Connection-oriented 1:1
  TCP

- Connectionless 1:n
  Multicast



Multicast Communication

Efficient group communication is becoming increasingly important in modern multimedia applications such as video conferencing or joint editing of documents.

The standard solution is called multicast and provides 1-to-n communication:

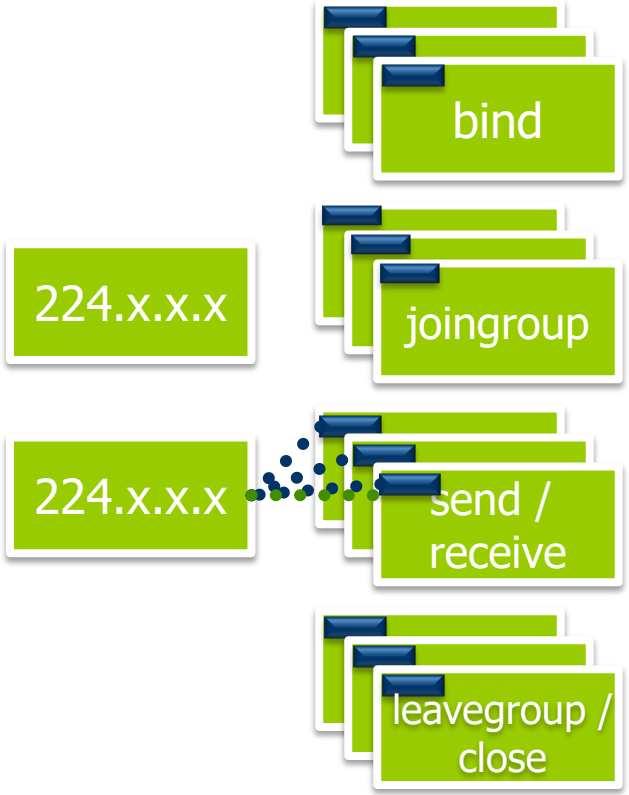
- The application only needs to manage one connection per group.
- The resources in the network are used more efficiently.

Effect of using Multicast for building Distributed Systems

- Fault tolerance based on replicated services
- Discovering services in spontaneous networking
- Better performance through replicated data
- Propagation of event notifications

Multicast Sockets

1. Participants bind socket
2. Participants join group
3. Particpants receive messages from sender
4. Partcipants leave group and release socket



IP Multicast

- Is built on top of the Internet Protocol (IP) and allow the sender to transmit a single IP packet to a set of computers that form a multicast group.
- Multicast group is specified by a Class D Internet Address. Every IP datagram whose destination address starts with "1110" (in IPv4) is an IP Multicast datagram.
- IP packets can be multicast on a local and wider network. In order to limit the distance of operation, the sender can specify the number of routers that can be passed (i.e. time to live, or TTL)
- Multicast addresses can be permanent (e.g. 224.0.1.1 is reserved for the Network Time Protocol (NTP))

Java API: java.net.MulticastSocket

```
public class MulticastSocket extends DatagramSocket {
        public MulticastSocket()...
        public MulticastSocket(int port)...
        // create socket and select port number explicitely or implicitely

        public void joinGroup(InetAddress mcastaddr) throws ...
        // join group under the address mcastaddr
        public void leaveGroup(InetAddress mcastaddr) throws ...
        // leave group
        public void setTimeToLive(int ttl) ...
        // define Time to Live – default is 1 !
}
```

Please note: send, receive, ... are inherited from class DatagramSocket

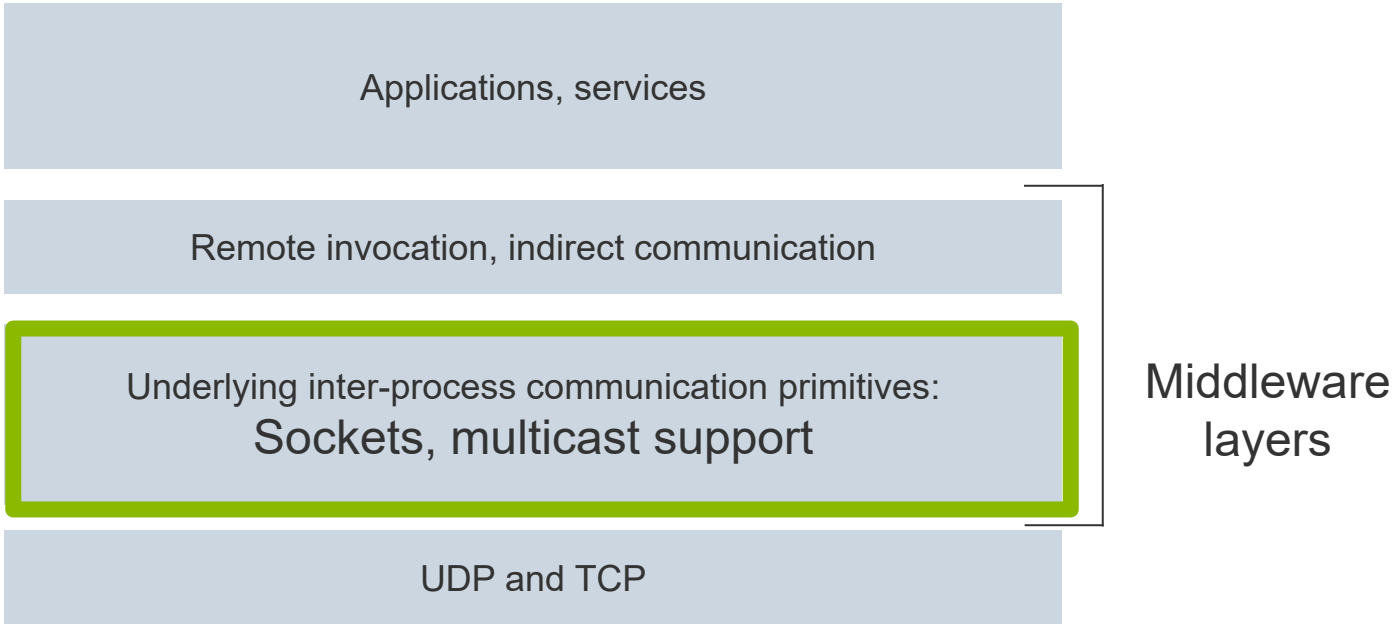
Failure Model of (*unreliable*) Multicast

- Datagrams multicast over IP multicast have the same failure characteristics as UDP datagrams – that is, they suffer from omission failures.

- The effect on a multicast is that messages are not guaranteed to be delivered to any particular group member in the face of even a single omission failure.

- This model can be extended by a reliable multicast that considers:
  - Integrity: the message received is the same as the one sent, and no messages are delivered twice
  - Validity: any outgoing message is eventually delivered
  - *Agreement*, stating that if the message is delivered to one process, then it is delivered to all processes in the group

Middleware Layers



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