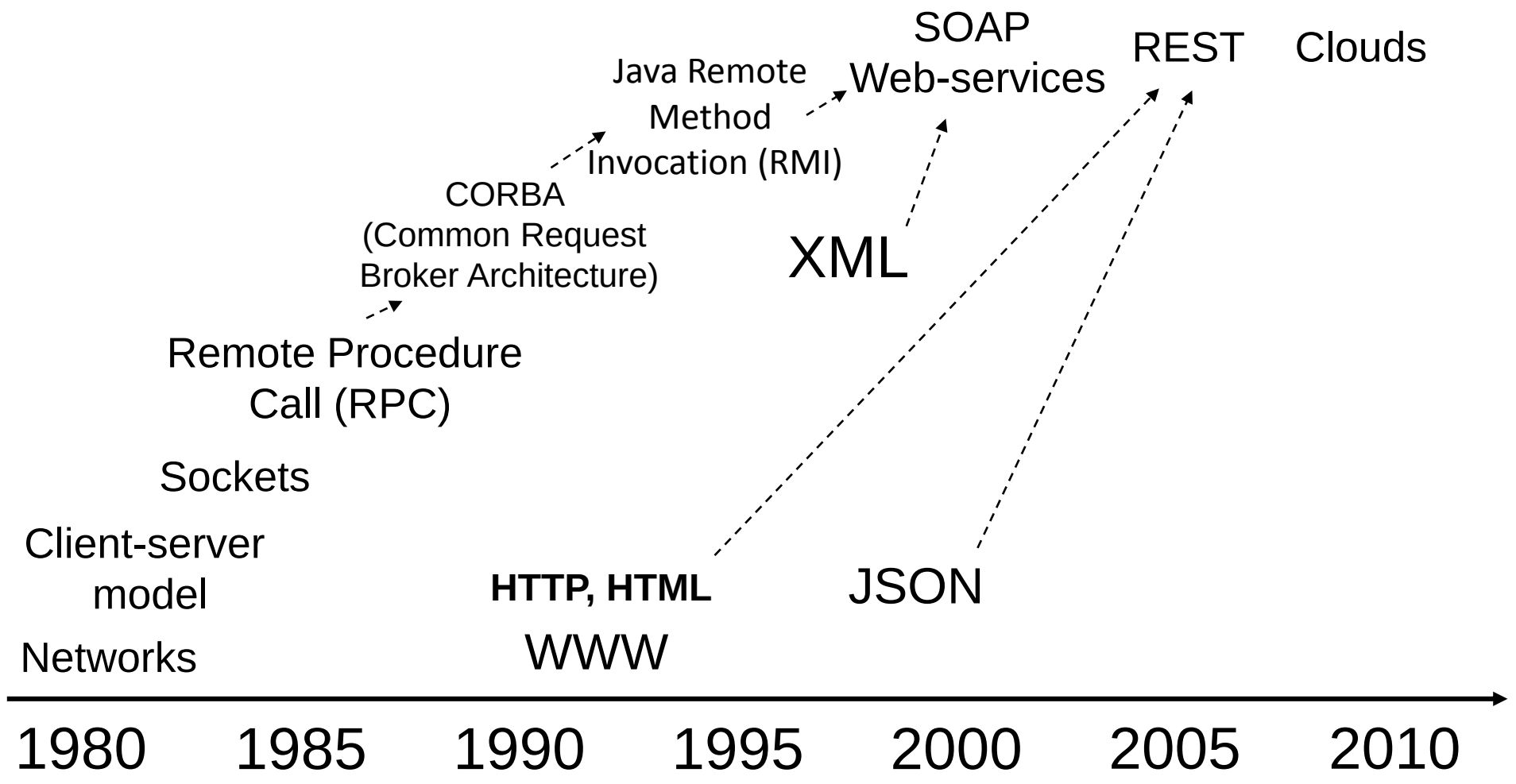
Distributed computing systems

Protocols and middleware

The role of communication in DCS

Distributed computing standards, protocols and architectures

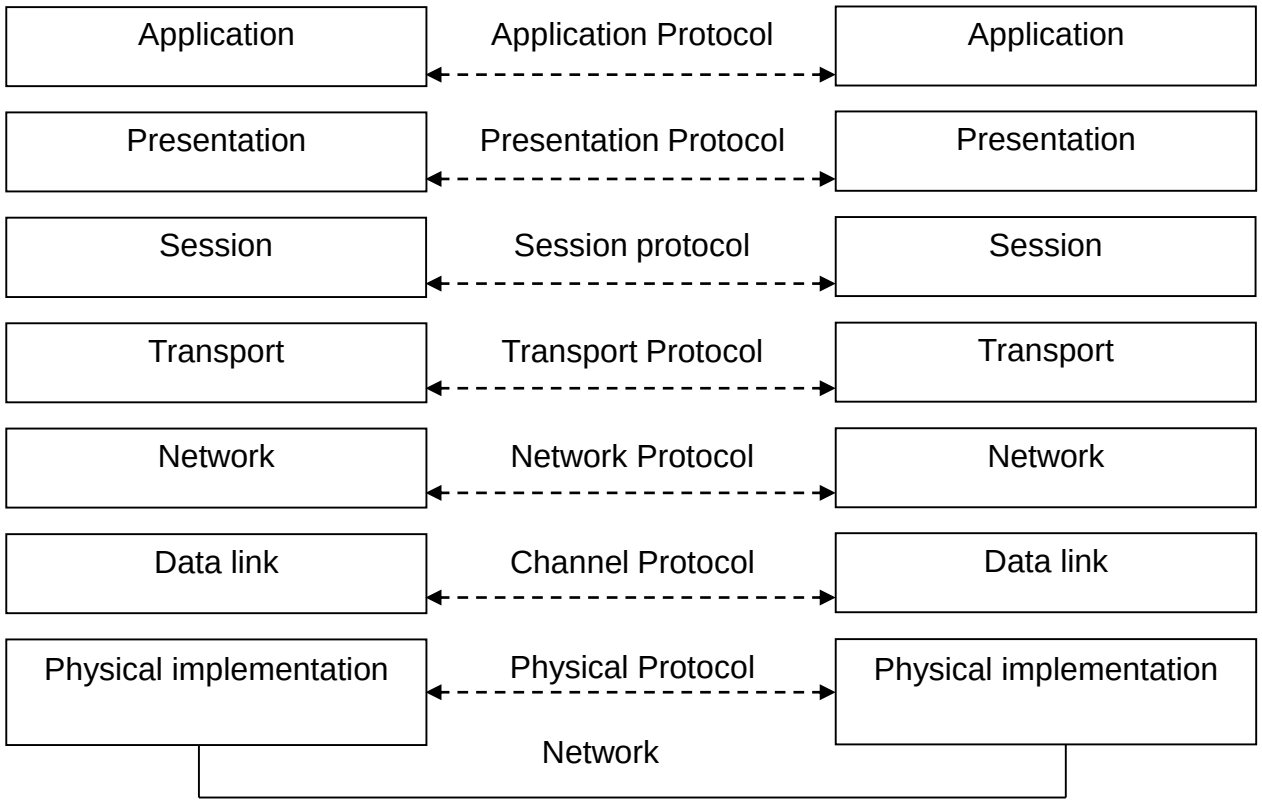


Protocol

Interaction is based on protocols.

A protocol is a set of rules and agreements, describing the procedure for interaction between components of the system.

OSI PROTOCOL STACK



FTP, SMTP, HTTP

XML, JPEG, ASCII

PPTP, SMPP

TCP, UDP

IP, IPsec

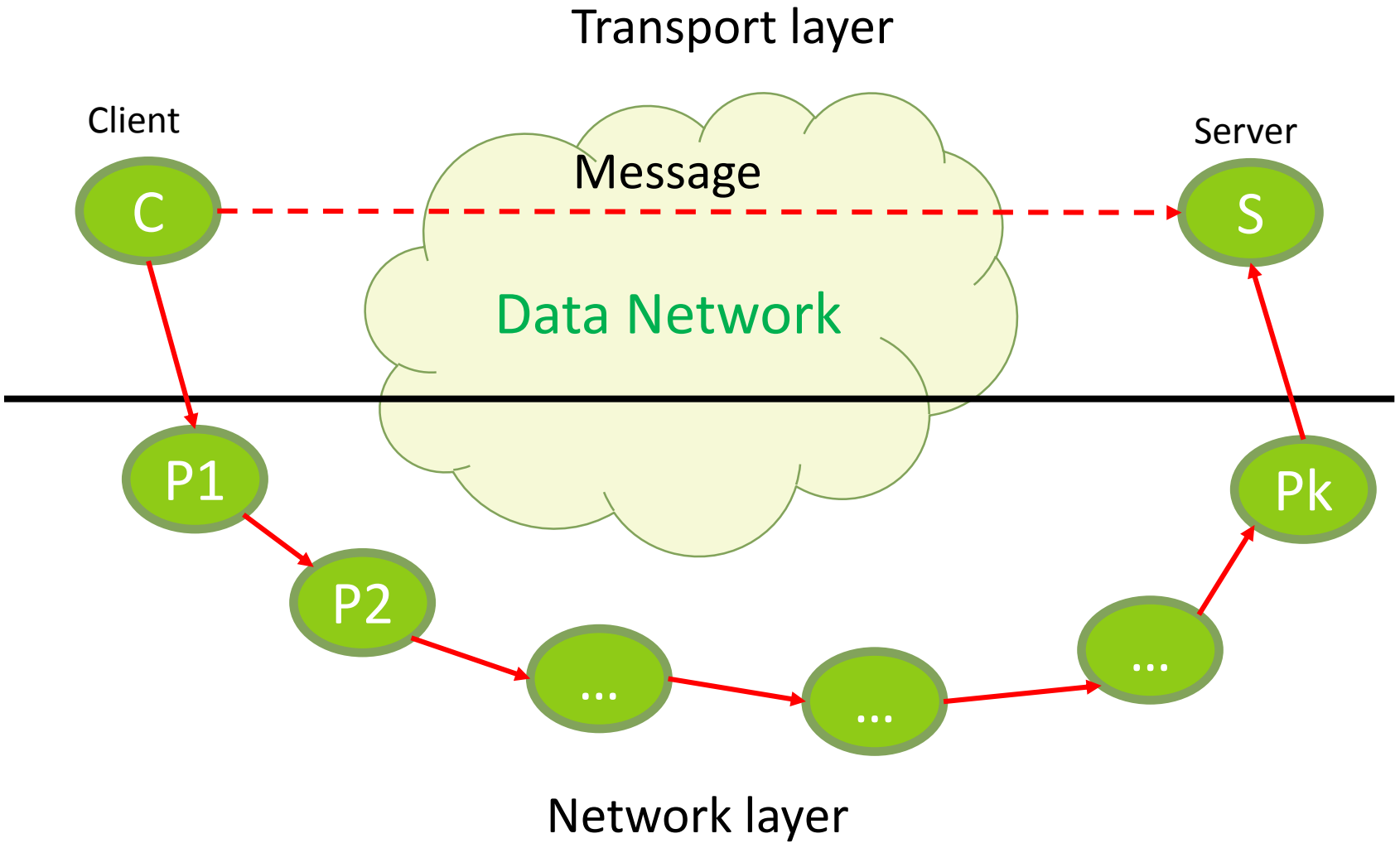
IEEE 802.11, PPP, PPPoE

Bluetooth, DSL, IRDA

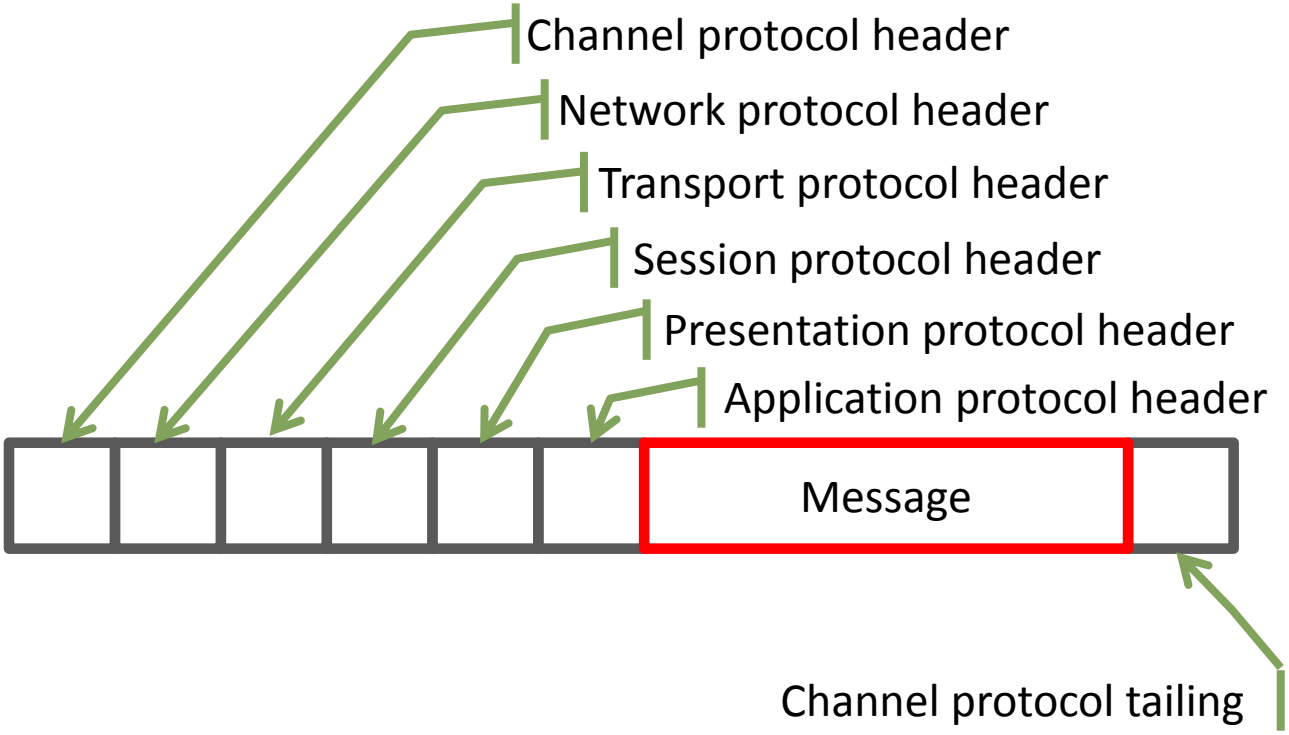
OSI PROTOCOL STACK

1. **Application layer:** specific application needs of the user processes. Examples are email, bulletin boards, chat rooms, web applications, directory services, etc.
2. **Presentation layer:** compatibility problems by addressing the syntactic differences in data representation. Mime encoding, data compression, and encryption are addressed in this layer. Another example is representing structure by using XML.
3. **Session layer:** the connection between peer processes is established and maintained at this level for all connection-oriented communications.
4. **Transport layer:** the goal of the transport layer is to provide end-to-end communication between the sender and the receiver processes.
5. **Network layer:** the network layer provides machine-to-machine communication, and is responsible for message routing.
6. **Data-link layer:** this layer assembles the stream of bits into frames, and appends error-control bits (like cyclic redundancy codes) to safeguard against corruption of messages in transit.
7. **Physical layer:** this layer deals with how a bit is sent across a channel. In electrical communication, the issue is what voltage levels (or what frequencies) are to be used to represent a 0 or a 1.

Transport and network layer



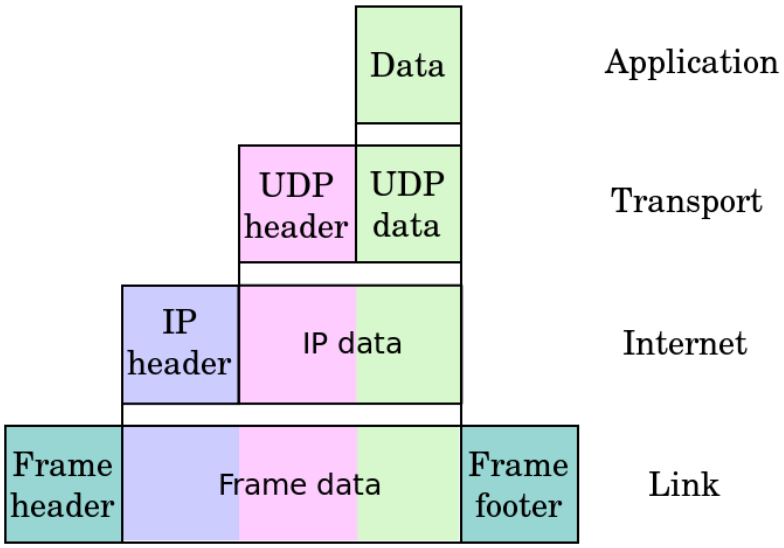
Messages «matrioshka»



TCP/IP

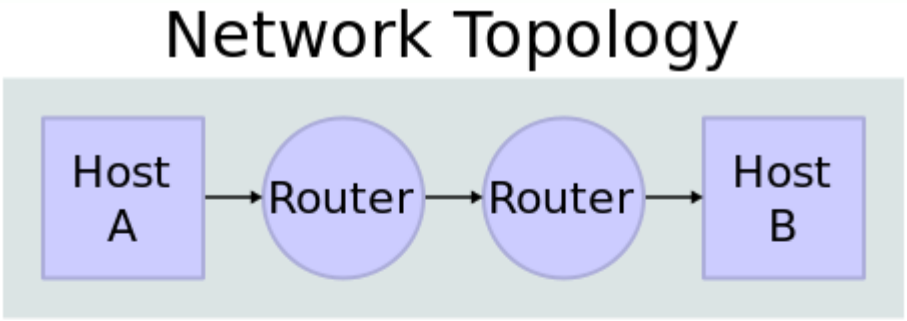
- The most popular protocol stack on the Internet
- Four layers

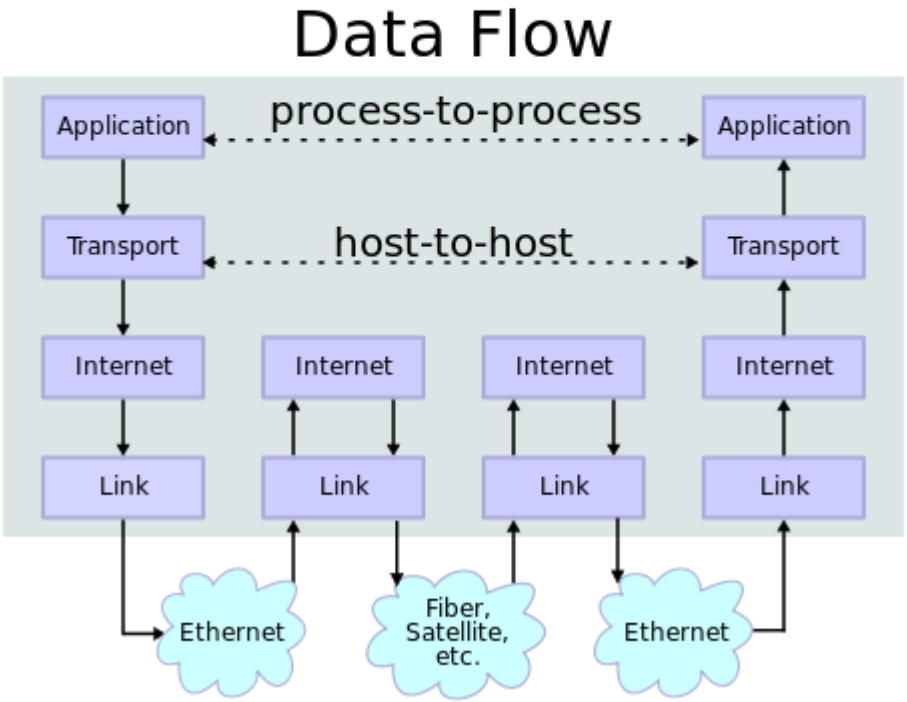
Application layer	SMTP, HTTP, Telnet, FTP etc.
Transport layer	UDP, TCP
Internet layer	IP
Link lyer	Hardware



IP

- Defines the datagram as the unit of data transmission
- Specifies the Internet address scheme
- Transmits datagrams from sender to receiver





TCP/IP

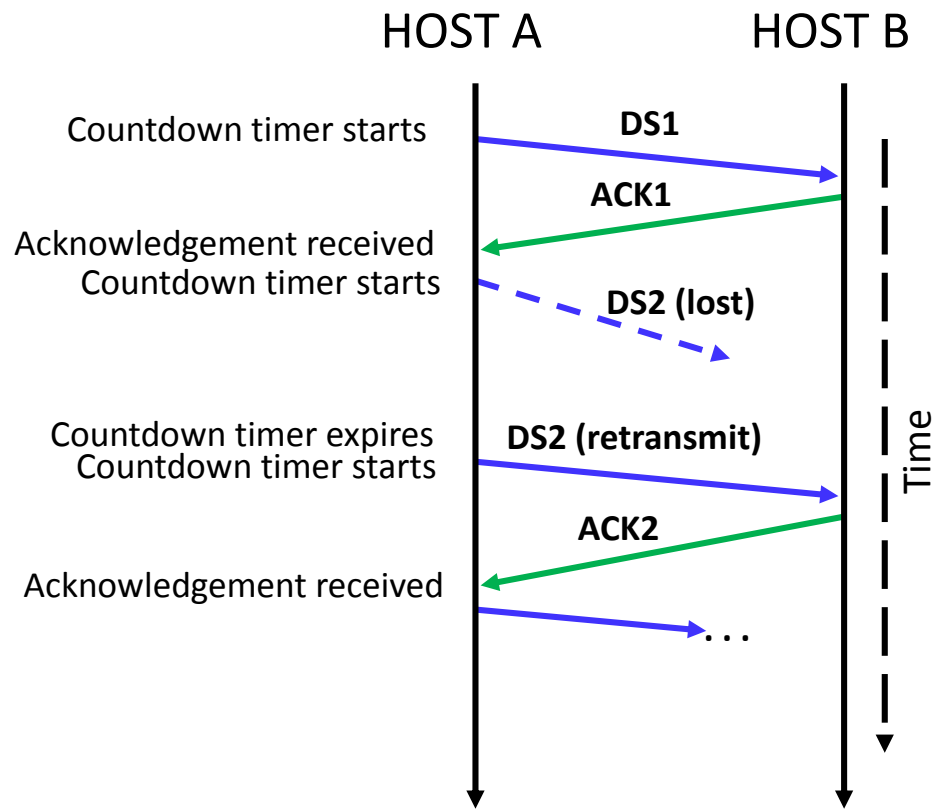
- TCP/IP - transport layer, providing transfer of data from the client to the server.
- Two main protocols: TCP and UDP.

Слой	TCP	UDP
Application	Data is transmitted in streams	Data is transmitted in messages
Transport	Segment	Packet
Internet	Datagram	Datagram
Link	Frame	Frame

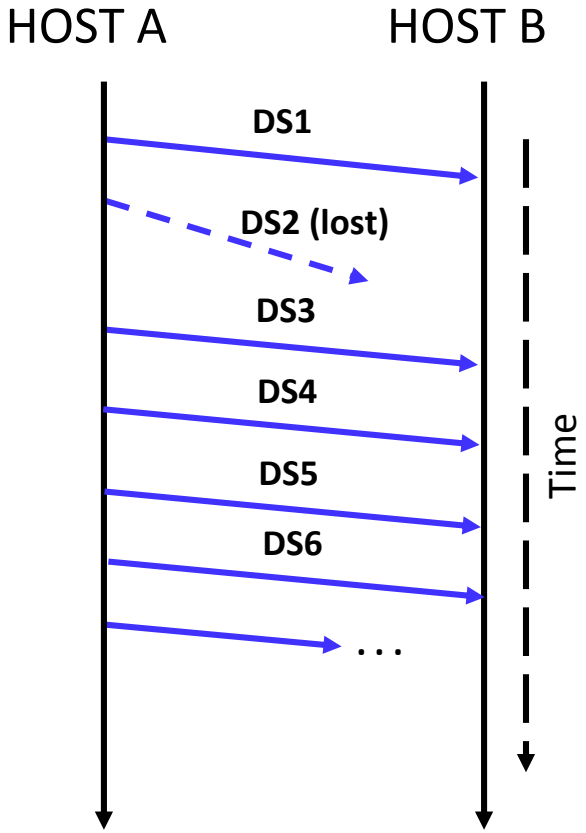
TCP vs UDP DATA TRANSFER

TCP Data Transfer

ACK – Acknowledgement
DS – Data stream



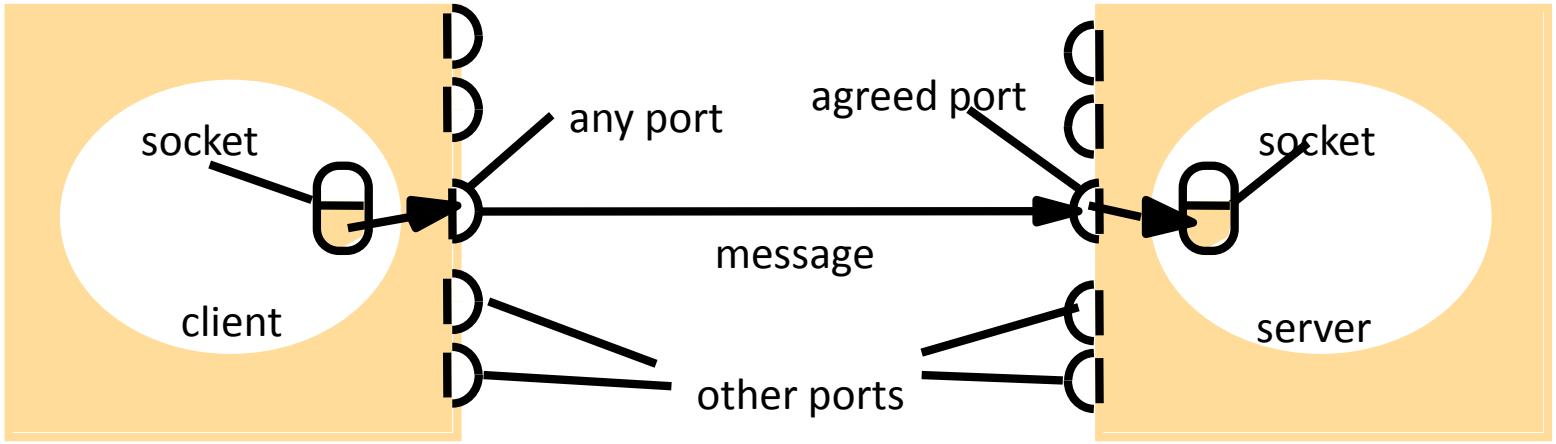
UDP Data Transfer



Direct message transmission: Sockets

Direct message transmission: Sockets

- Uses transport layer directly in the form of Middleware.



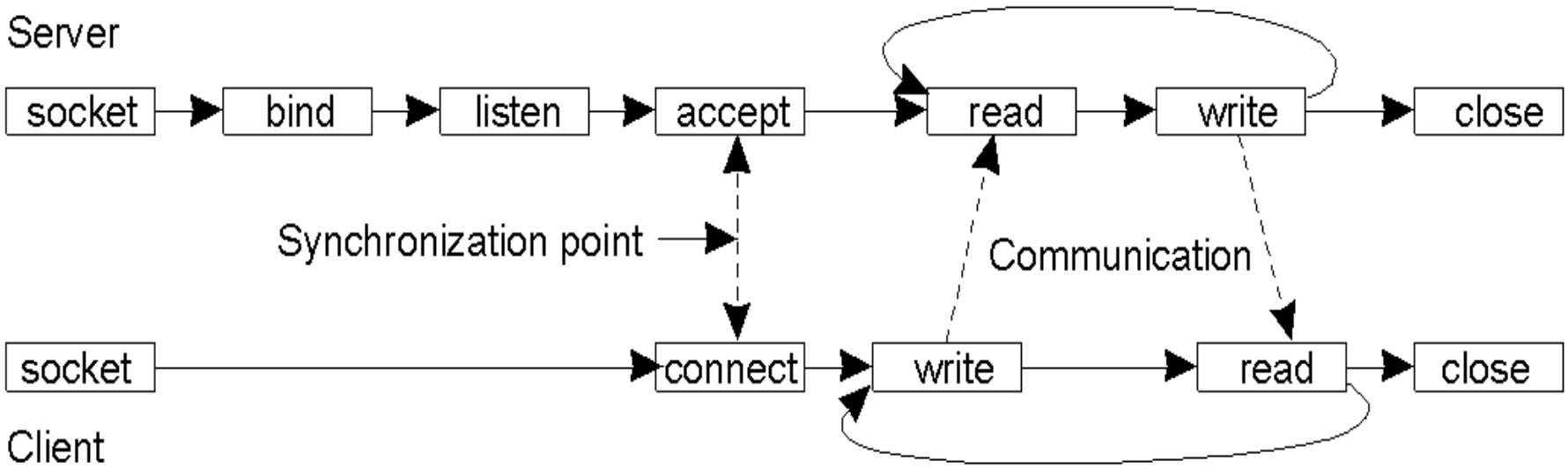
- **A socket** is an abstract object that represents the endpoint of the connection.
- TCP/IP socket is a combination of IP address and port number, for example, 10.10.10.10: 80.
- Socket interface first appeared in BSD Unix.

Berkeley Sockets API (1)

Socket primitives for TCP/IP.

Primitive	**Meaning**
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Berkeley Sockets (2)



Socket implementation example

C # supports two types of network connections:

- Server using the TcpListener class objects;
- the client implemented by using objects of the TcpClient class.

TcpListener and TcpClient objects

- An object of TcpListener class allows **only to listen** to a specific port on your computer.
- Any processes of data transmission via this socket are carried out using the TcpClient object.
- The AcceptTcpClient() method of the TcpListener class returns the TcpClient object that provides the listening port.

Server example

```
using System.Net;
using System.Net.Sockets;

Int32 port = 13000;

IPAddress localAddr = IPAddress. Parse
   ("127.0.0.1");

TcpListener server = new TcpListener (localAddr,
   port);

server.Start ();

//Start listening on port
TcpClient client = server.AcceptTcpClient ();
//After connection create message flow
NetworkStream stream = client.Getstream();
```

Messaging

Writing messages

```
Byte [] bytes = new Byte
  [256];

String data = "text";

bytes =
  System.Text.Encoding.UTF.
  GetBytes (data);

stream.Write (bytes, 0,
  bytes.Length);
```

Reading messages

```
Byte [] bytes = new Byte
  [256];

String data = null;

int i = stream.Read (bytes,
  0, bytes.Length);


data = system.text.
  encoding.UTF8.GetString
  (bytes, 0, i);
```