UPPSALA UNIVERSITET

Process communication that involves the network

Lars-Åke Nordén

UPPSALA UNIVERSITET

Communicating processes

- There is a common need for processes to communicate
  - Exchange or share information
  - Request services
  - Redirection of data
  - ...
- Interprocess communication comes in many flavors
  - Different services
  - Different delays
  - Different degrees of reliability
  - ...

Focus of this lecture

- Process communication over the network
- Different types of application design
  - Client/server, Peer-to-peer...
- Different types of communication services
  - Reliable vs unreliable
  - Byte stream vs message forwarding
- Different types of communication properties
  - Delay, reliability, robustness...

Applications' communication needs

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-10Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
financial apps	no loss	elastic	yes and no

Applications take whatever they get

The environment

Application

Logical communication between applications

Transport

Network

Best-effort datagram Delivery using IP

Link

- Packets can be
  - Lost
  - Reordered
  - Damaged
  - Delayed
  - Fragmented
- Packets are
  - Unknown in size
  - Sent to a **node**
  - Sent in cleartext
  - Delivered ”asap”

Possible transport layer functionality

- Reliable delivery
- In-order delivery
- Deliver data at the right pace
- Data integrity verification
- Ignoring network layer fragmentation
- Process (Application) multiplexing
- Encryption, Authentication
- Avoiding packet losses
  - Adapt to receiver
  - Adapt to network
  - Efficient operation (low overhead)
- Cooperation with other nodes
  - Share bandwith fairly
  - Reduce speed at congestion
  - Local recovery in case of errors

Communication services

Message forwarding

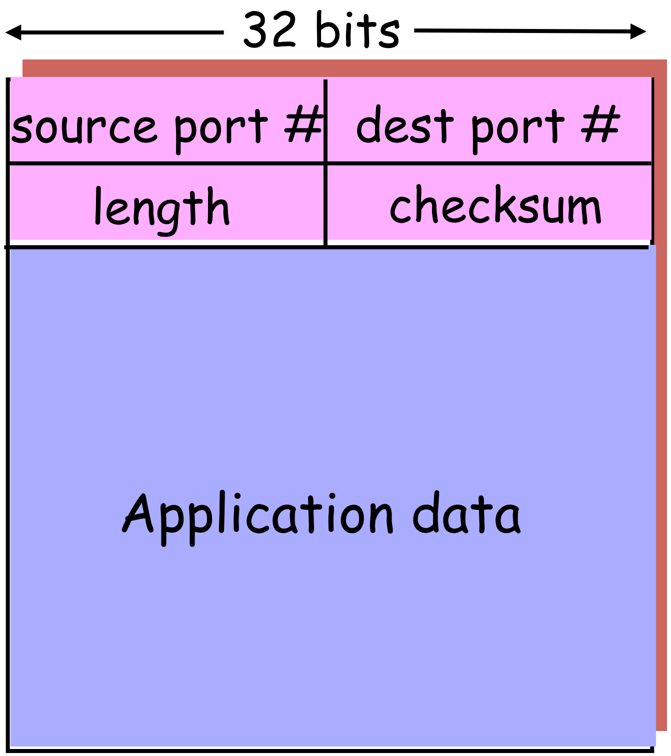
- Connectionless
- Fast
- As reliable as the Internet
  - Best-effort delivery model
- Priority: speed
- Protocol: UDP

Reliable byte stream

- Connection oriented
- “Slow”
- Reliable delivery
  - Retransmissions
  - Data delivered in-order
- Priority: reliability
- Protocol: TCP

UDP (User Datagram Protocol)

- No set-up phase
- No states at endpoints
- Small “head”
- No congestion control
  - Application specifies sending speed
- Only adds process multiplexing



UDP Datagram format

Properties of TCP

- **Byte-oriented**
- Reliability and in-order delivery through a sliding window using cumulative ACK:s
- ACK:s can be piggy-backed on data as they are included in the headers
- Notation of two simplex channels that form a duplex logical channel
- Sender-oriented congestion control
- Assumes that all packet losses are due to congestion
- Congestion-avoidance
- Fast retransmit
- AIMD
- MA estimation of RTT to set timeout timer
- Flow control
- Three-way handshake
- Three-way teardown

Flow control

- Receiver reports available space in receiver buffer
- Sender does not send data that will not fit into receivers' buffer

Congestion control

- Estimates capacity of network share
  - Simple control loop
  - Can exist at both sender and receiver
- Possible input for congestion control algorithms:
  - Variations in RTT
  - Duplicated cumulative ACK:s
  - Timeouts
  - Variations in inter-ACK spacing
  - Timestamps
  - …

UPPSALA UNIVERSITET

The sending window in TCP

- Whenever a segment is sent, the variable *maxwin is updated as follows:*

  `maxwin = min(Adv.win, CWND, Buffersize)`

  - Adv.win = Available space in receivers' buffer
  - CWND = Estimatation of network capacity
  - Buffersize = Maximum sending window size

- Data will not be sent if it will cause the sending window to exceed *maxwin*
  - Sender must then wait until sending window < *maxwin*

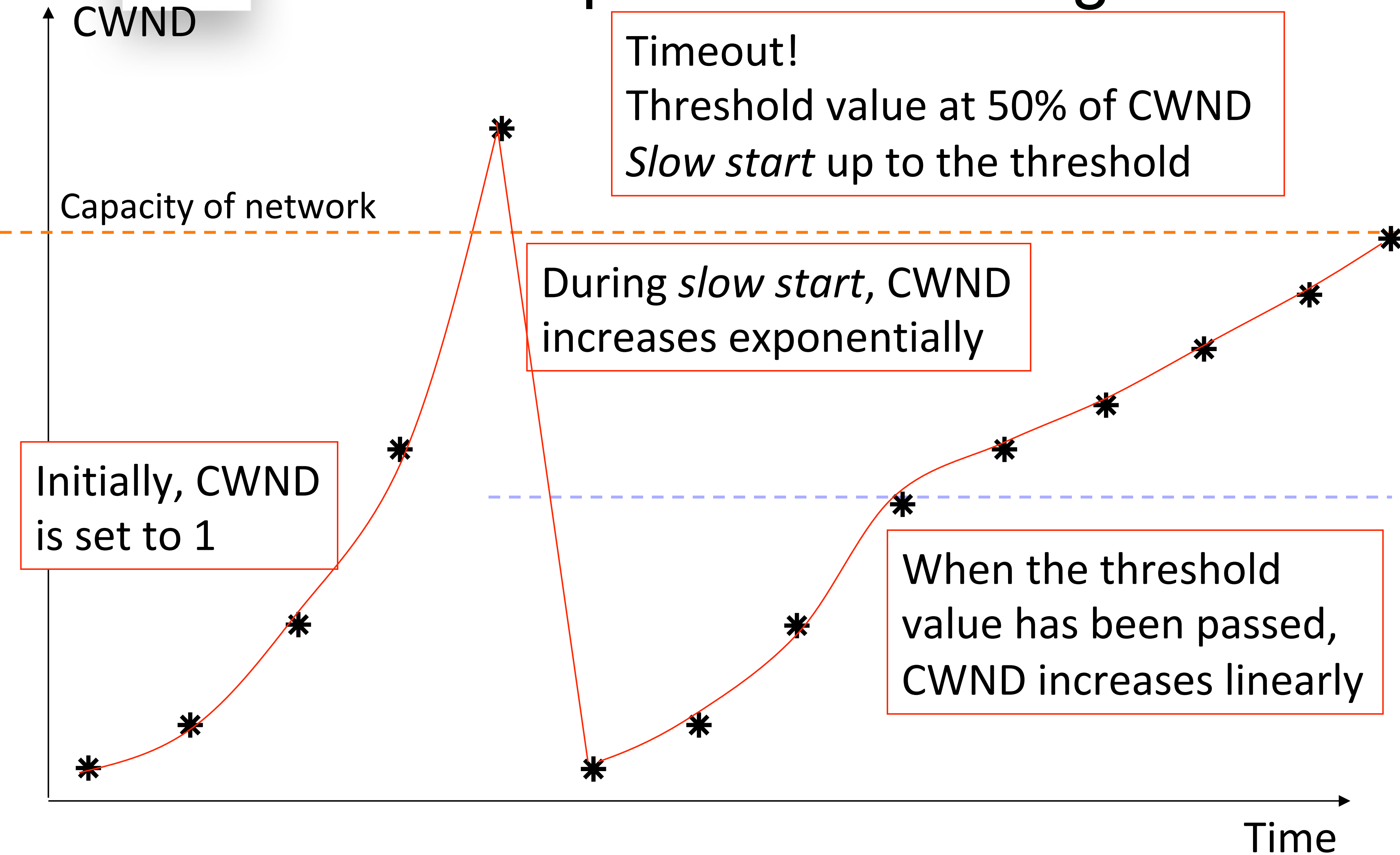
Regulating the sending window in TCP

- Adv.Win:
  - Is reported in each ACK received

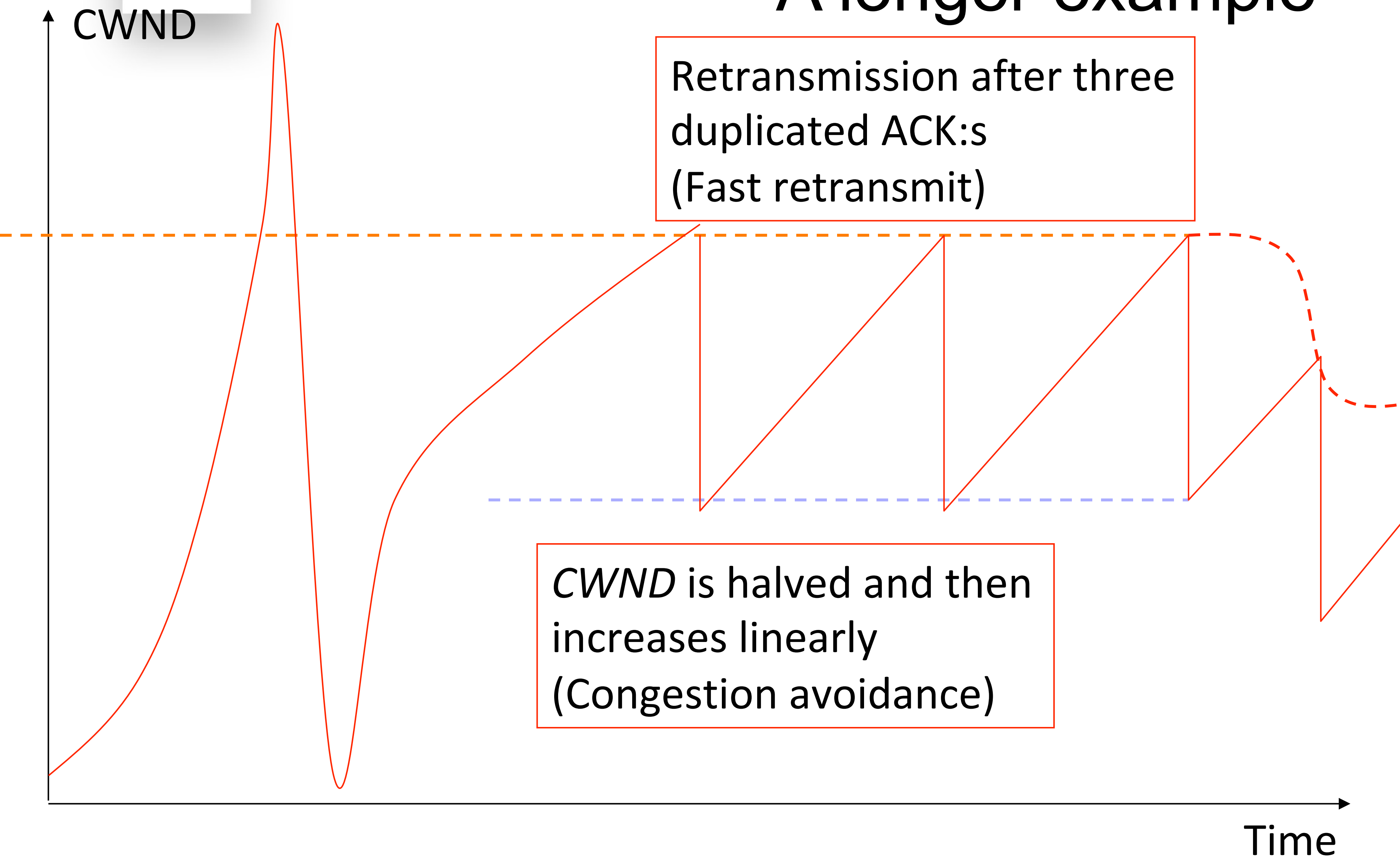
- CWND:
  - Is reduced ”a lot” in the case of a timeout
    - Timeouts are interpreted as critical events that require a significant reduce in sending speed to avoid gridlocks
  - Can be reduced by duplicated (cumulative) ACKs
    - Not considered as a critical event as data still gets through
  - Can be lowered by measured variations in time
    - Assumed to be caused by queues filling up in the network

- **Normally, it is CWND that limits the current sending speed of TCP**

Example of CWND regulation

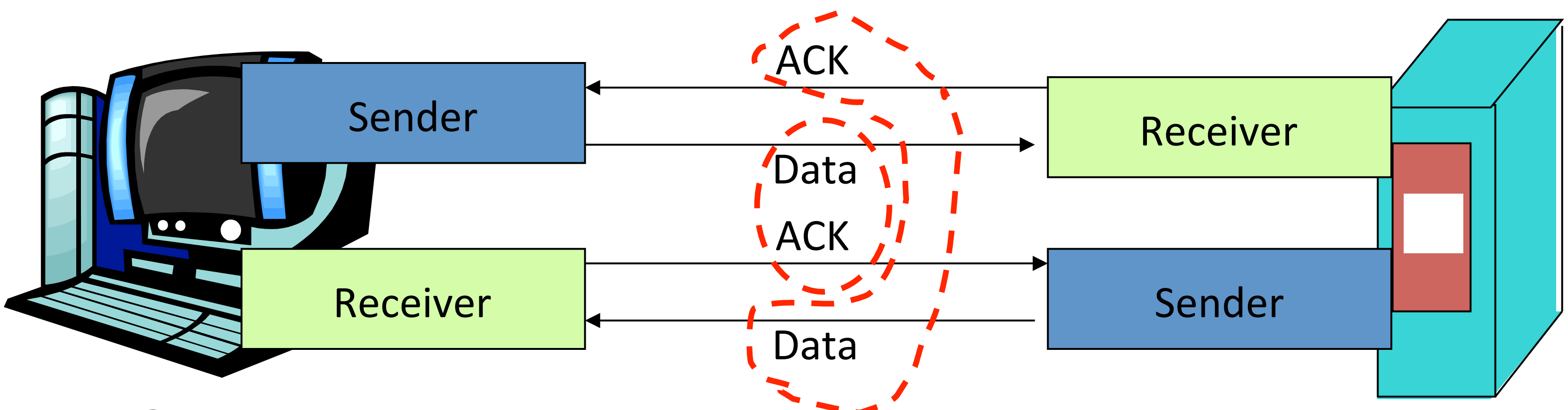


A longer example



A TCP session

- Begins with a 3-way handshake
  - Initiated by the ”client"
- After handshake, full duplex
  - Both sides are both sender and receiver
  - ACK:s can be ”piggybacked”



- Session ends with a 3-way handshake

Round-trip times and Timeouts

EstimatedRTT = (1-x)*EstimatedRTT + x*SampleRTT

- Exponential MA-filter
- Typical x value: 0.1

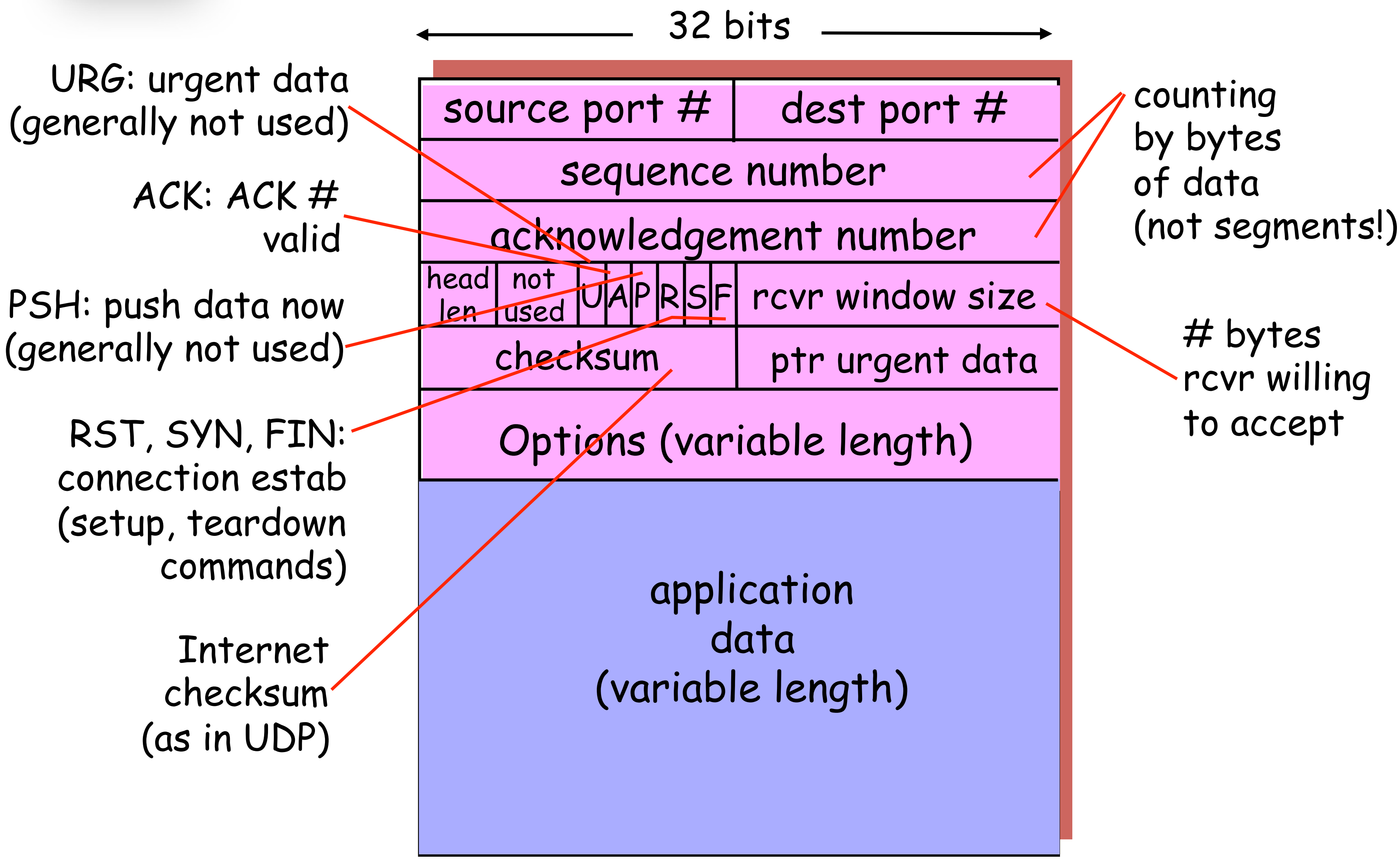
Setting the timeout

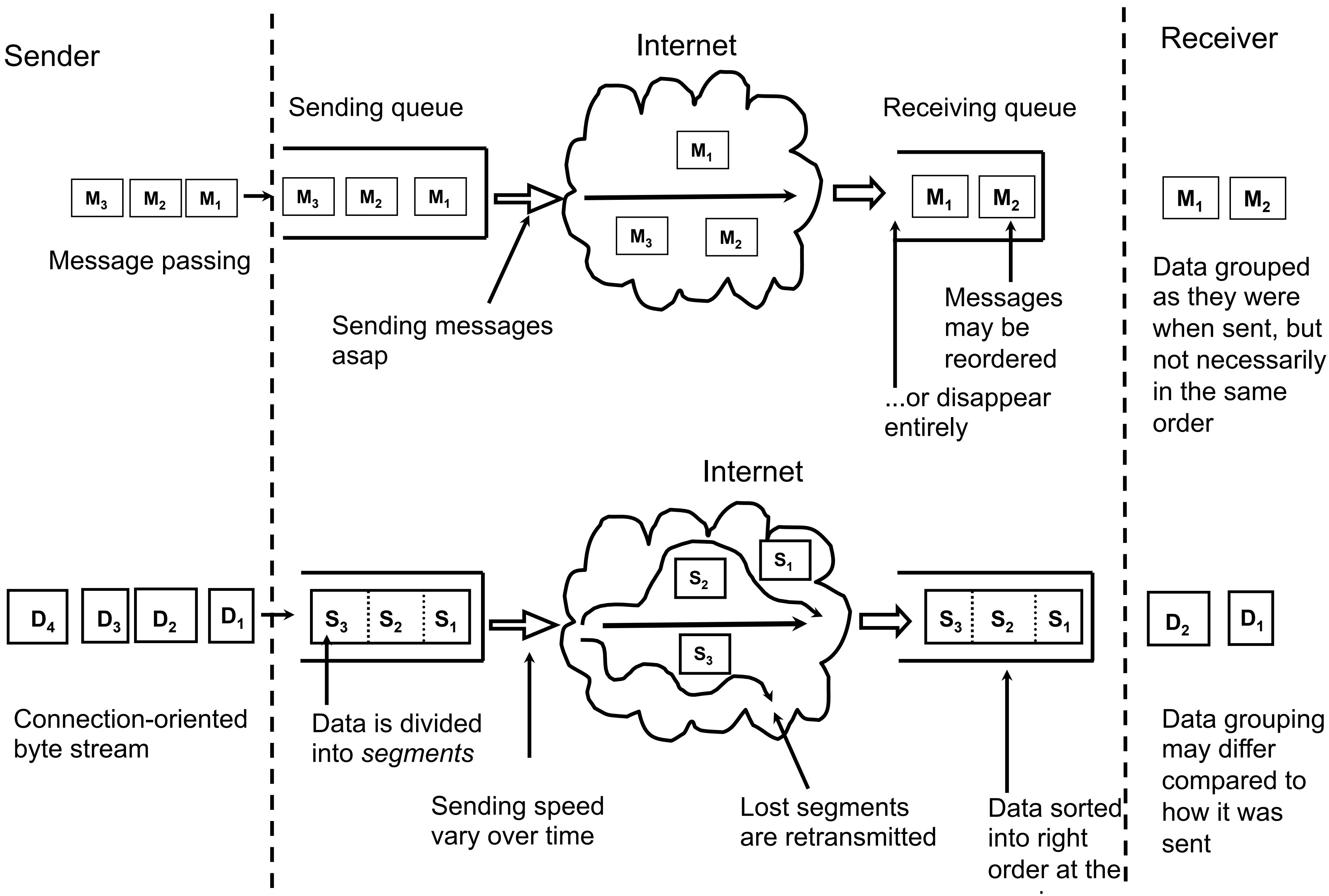
- **EstimatedRTT** plus "security margin"
- Large variations in **EstimatedRTT ->** larger marginal

```
Timeout = EstimatedRTT + 4*Deviation

Deviation = (1-x)*Deviation +
            x*|SampleRTT-EstimatedRTT|
```

TCP – Transmission Control Protocol

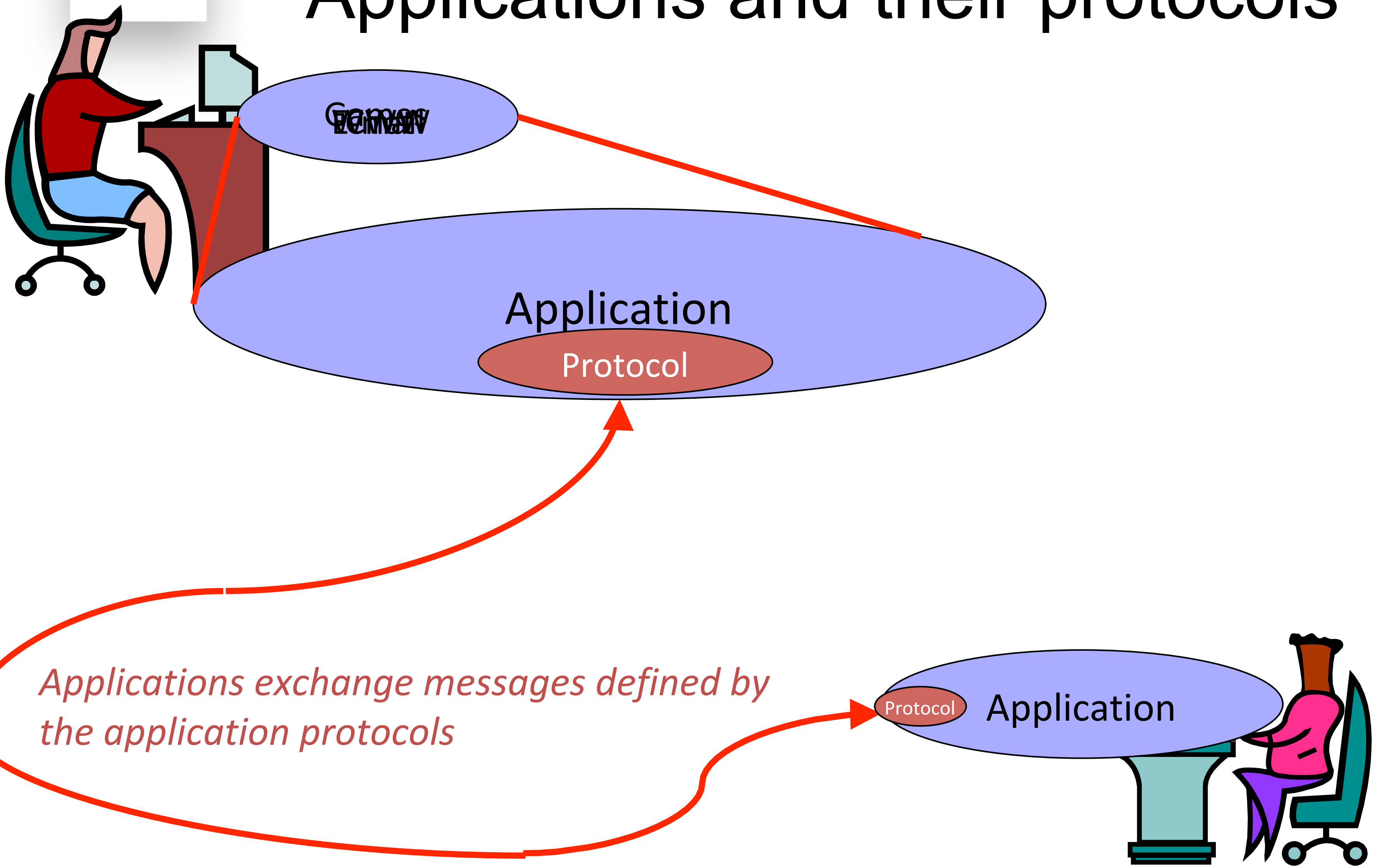



Sender
Internet
Receiver
Sending queue
Receiving queue
M3 M2 M1
M1 M2
Message passing
Sending messages asap
Messages may be reordered
...or disappear entirely
Data grouped as they were when sent, but not necessarily in the same order
Internet
D4 D3 D2 D1
S3 S2 S1
D2 D1
Connection-oriented byte stream
Data is divided into segments
Sending speed vary over time
Lost segments are retransmitted
Data sorted into right order at the receiver
Data grouping may differ compared to how it was sent

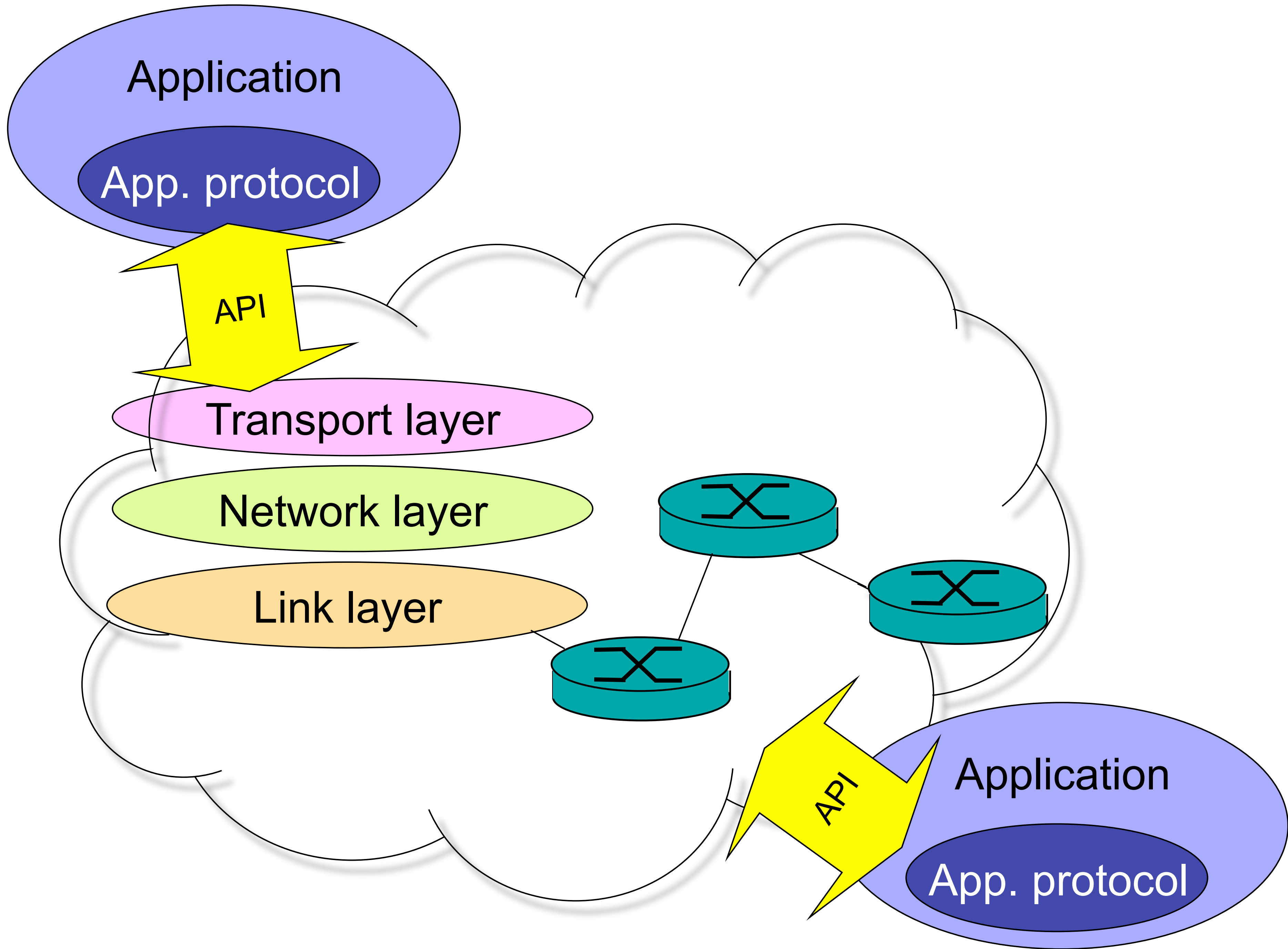
Delivery models

- **Unicast**
  - One sender, one receiver (default)
- **Broadcast**
  - Mass distribution, usually with limited reachability
- **Multicast**
  - Mass distribution to multicast *groups*
  - Receivers subscribe to groups to receive data
- **Anycast**
  - IP address identifies a service (DNS, nearest printer...)
  - Not necessary to know the identity of the receiver

Applications and their protocols

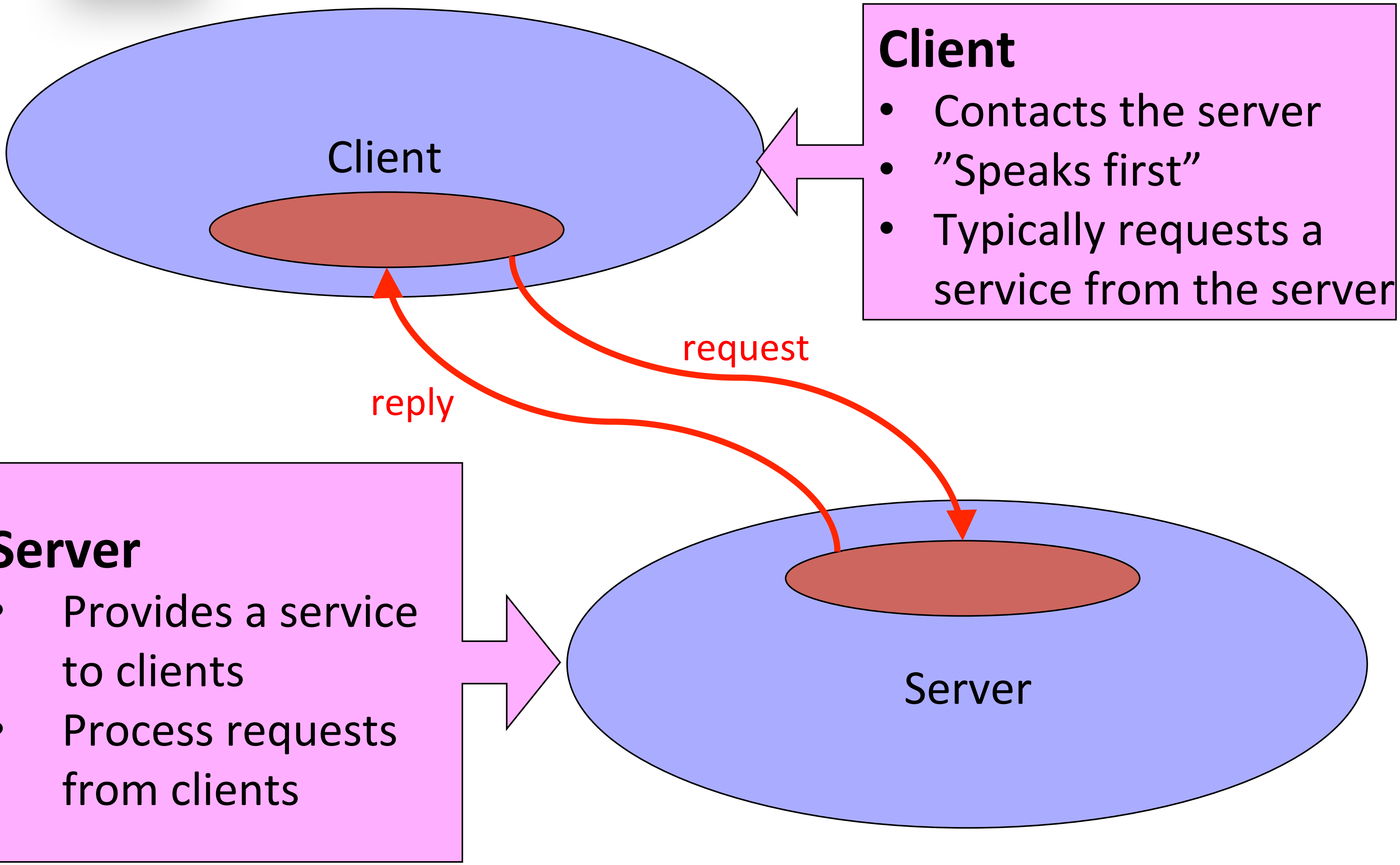


Applications exchange messages defined by the application protocols

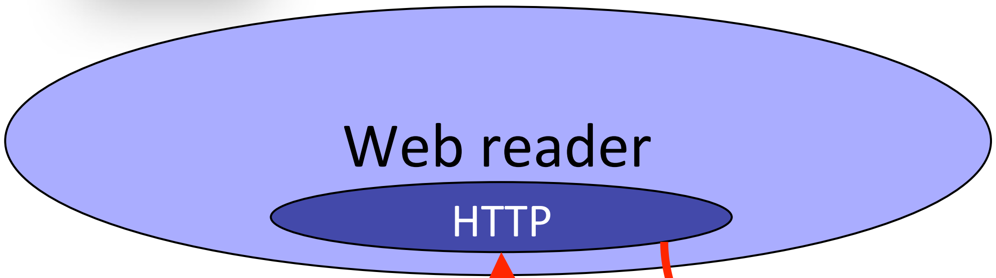


Application
App. protocol
API
Transport layer
Network layer
Link layer
API
Application
App. protocol

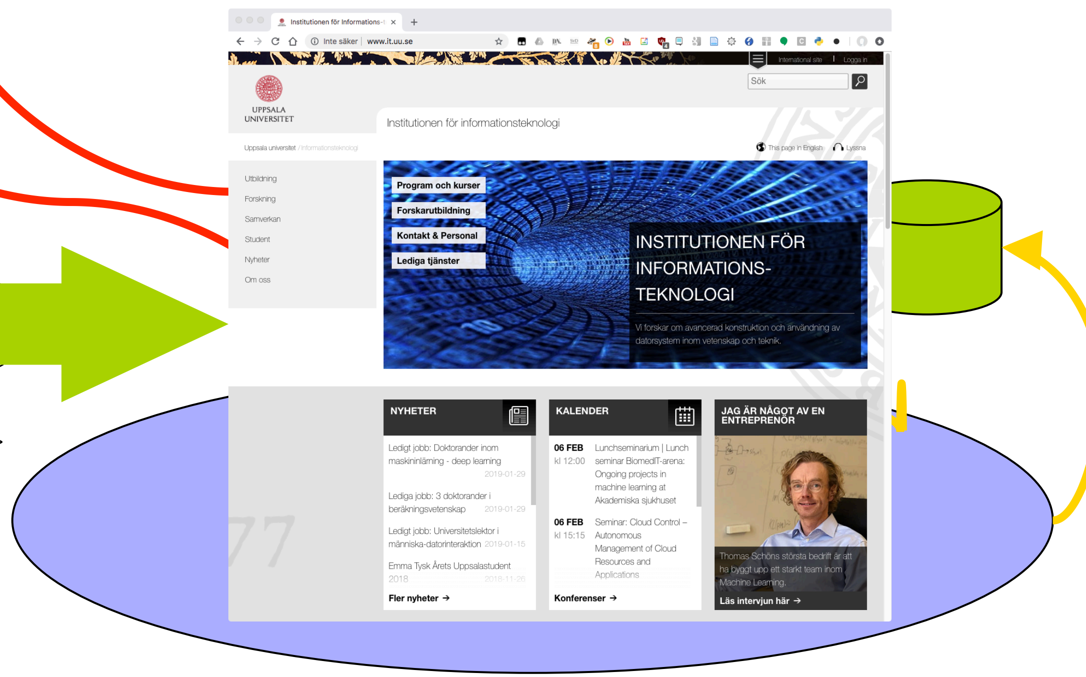
Client/Server communication



Example: WWW
www.it.uu.se



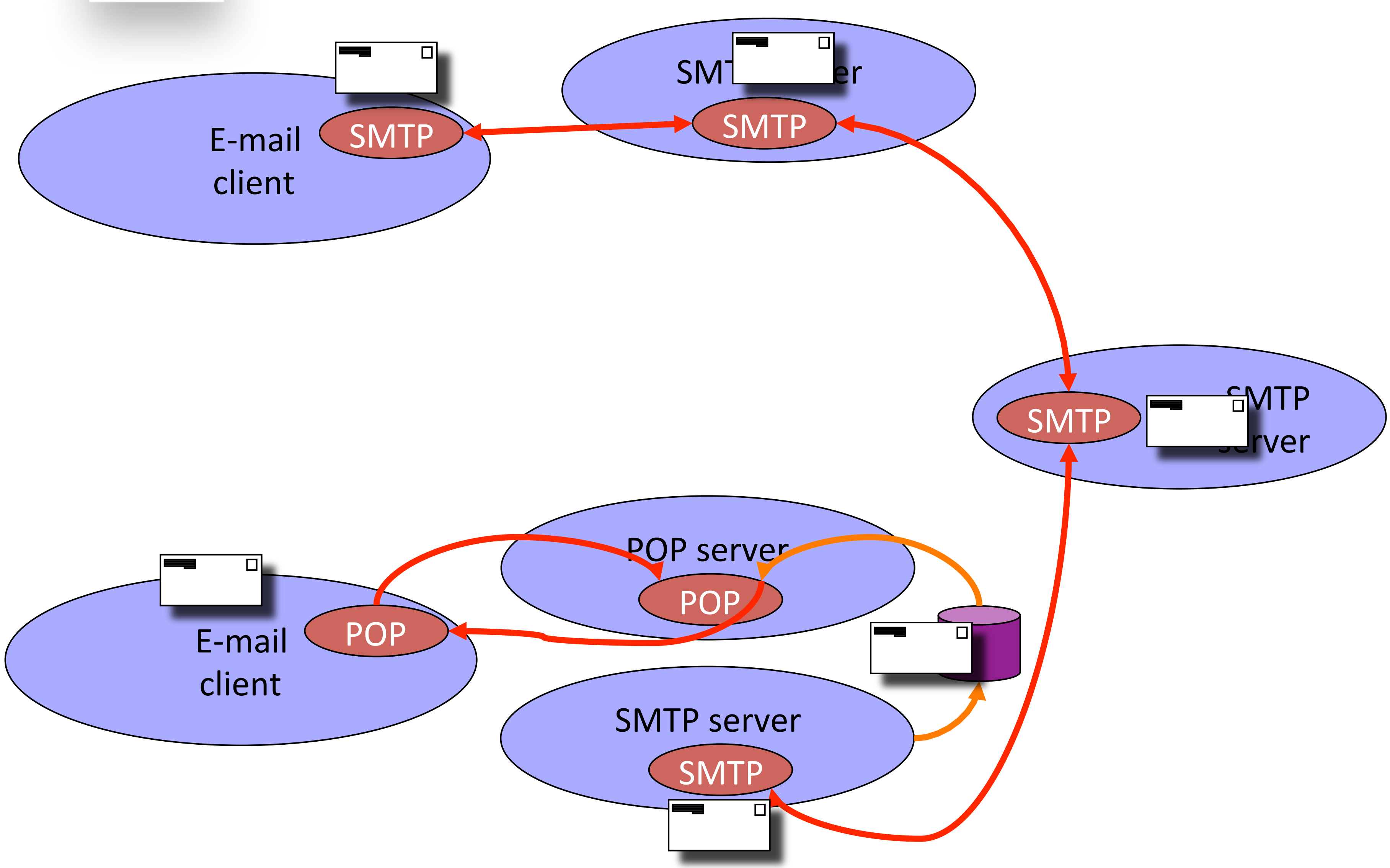

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "about:legacy-compat">
<html lang="sv">
<head>
<meta http-equiv="content-type" content="text/htm
<meta http-equiv="X-UA-Compatible" content="IE=e
<meta http-equiv="content-language" content="en">
<meta name="DC.Identifier" content="http://it.uu.se
<meta name="DC.Publisher" content="Uppsala Unive
<link href="/css/uu-common/css-org-staff.css" rel="st                t/css">
<link href="/css/uu-common/v2-aggregated-css-base.cs
<link href="/css/uu-common/v2-aggregated-css-modules.cs
css">
<link href="/css/uu-common/v2-aggregated-css-post.css" rel="stylesheet" type="text/css">
<link href="/css/uu-common/v2-aggregated-css-media-queries.css" rel="stylesheet"
type="text/css">
<link href="/css/uu-common/css-education-v604420.css" rel="stylesheet" type="text/css">
<link rel="stylesheet" type="text/css" title="style" href="/css/it-uu.css">
<link rel="apple-touch-icon-precomposed" href="https://www.uu.se/digitalAssets/
19/19350_1appletouch57.png">
<link rel="apple-touch-icon-precomposed" sizes="72x72" href="https://www.uu.se/
digitalAssets/19/19350_1appletouch72.png">
<link rel="apple-touch-icon-precomposed" sizes="114x114" href="https://www.uu.se/
digitalAssets/19/19350_1appletouch114.png">
<link rel="apple-touch-icon-precomposed" sizes="144x144" href="https://www.uu.se/
digitalAssets/19/19350_1appletouch144.png">
<link rel="shortcut icon" href="https://www.uu.se/digitalAssets/14/14093_1favicon.ico">
<title>Institutionen för Informations-teknologi - Institutionen för informationsteknologi -
Uppsala universitet</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<script src="https://www.uu.se/script/uu/modernizer-custom-2.6.2.js"></script><script
language="javascript" type="text/javascript" src="https://www.uu.se/jsp/errorjs.jsp"></
script><script type="text/javascript" src="/css/jquery.js"></script><script type="text/
```



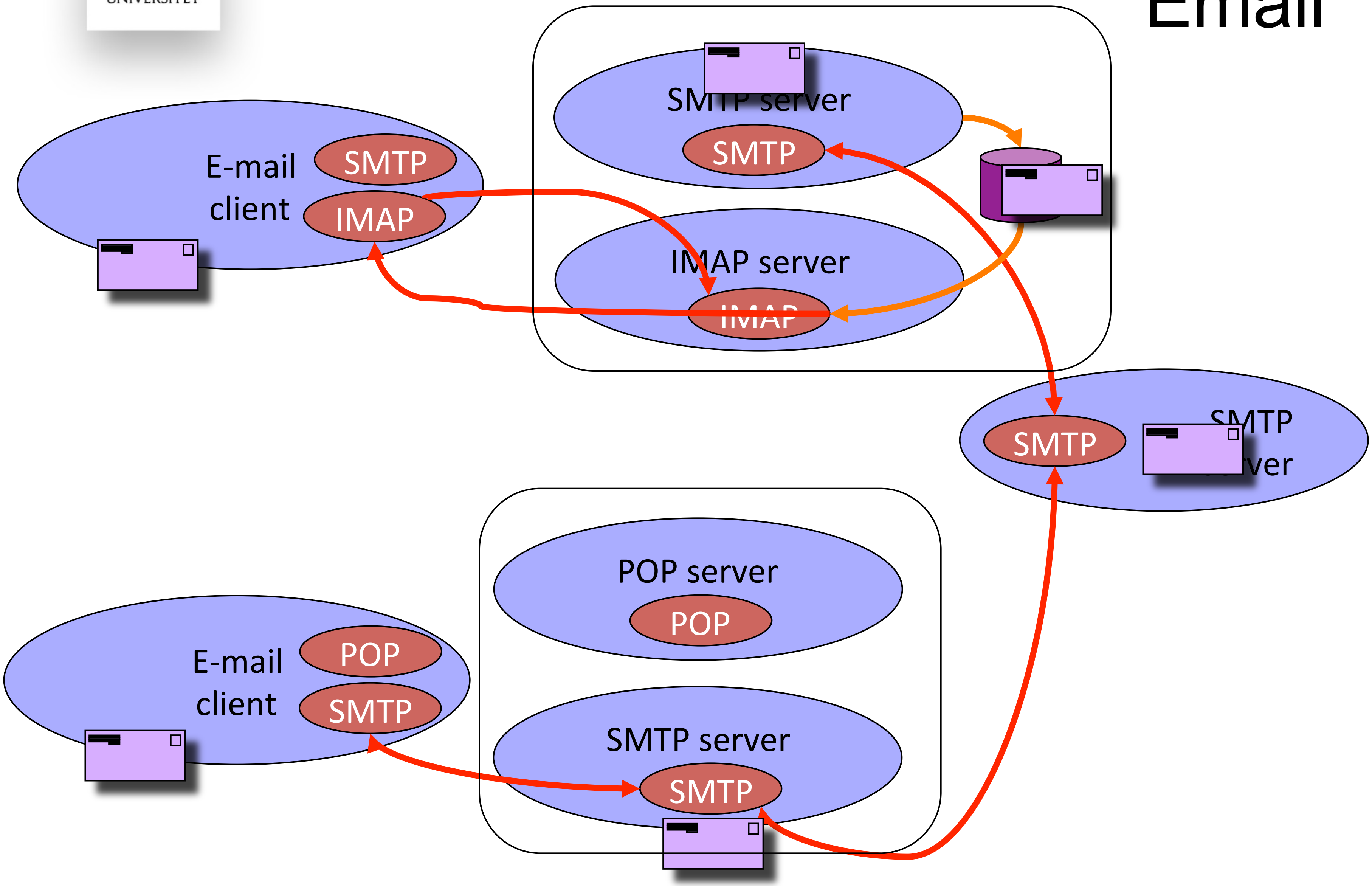
WWW

- 4 different components
  - – Web reader (Safari, Chrome, Explorer, Firefox...)
  - – Web server
  - – Protocol for data exchange (HTTP)
  - – Document format (HTML)
    - Supported by JavaScript, CSS...
- Operation
  - – The client requests a web page using HTTP
  - – The server responds with requested data
  - – Clients can usually handle more protocols than HTTP

Email

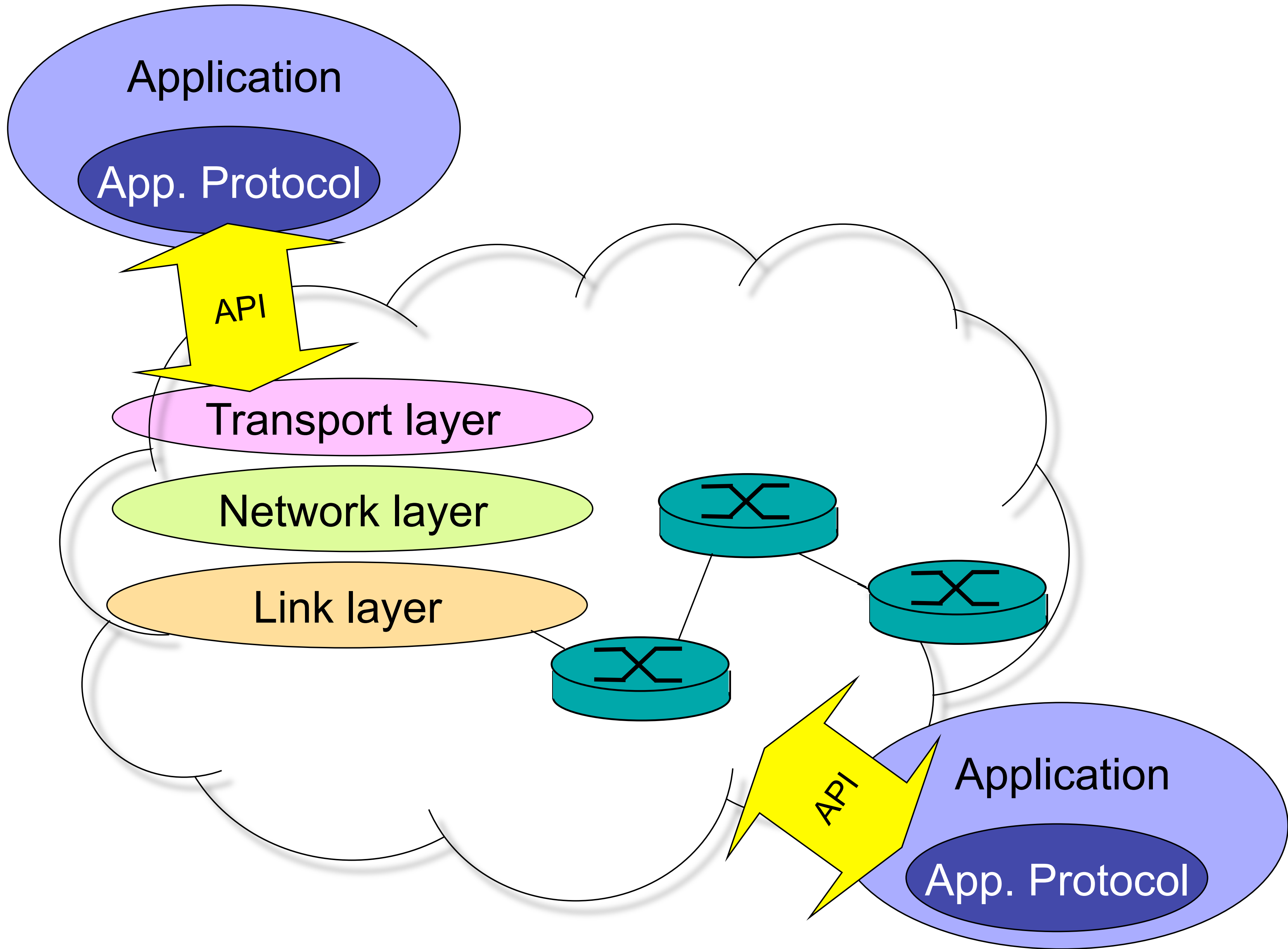

E-mail client
SMTP
SMTP
SMTP
POP server
POP
POP
E-mail client
SMTP server
SMTP

Email


E-mail client
SMTP
IMAP
SMTP
IMAP server
SMTP
E-mail client
POP
SMTP
POP server
POP
SMTP server
SMTP

"Killer apps" for the Internet

- 1980's FTP
  - Long-lived TCP flows, relatively large files
- 1990's WWW
  - Many asymmetric TCP flows
  - Motivated asymmetric bandwidths (ADSL etc.)
- 2000's File sharing & VoIP
  - Long, traffic-intense, bandwidth-hogging flows
  - Short messages, low delay required
- 2010's Video streaming
  - Data-intense flows with flexible delay requirements
- 2020's ?


Application
App. Protocol
API
Transport layer
Network layer
Link layer
API
Application
App. Protocol

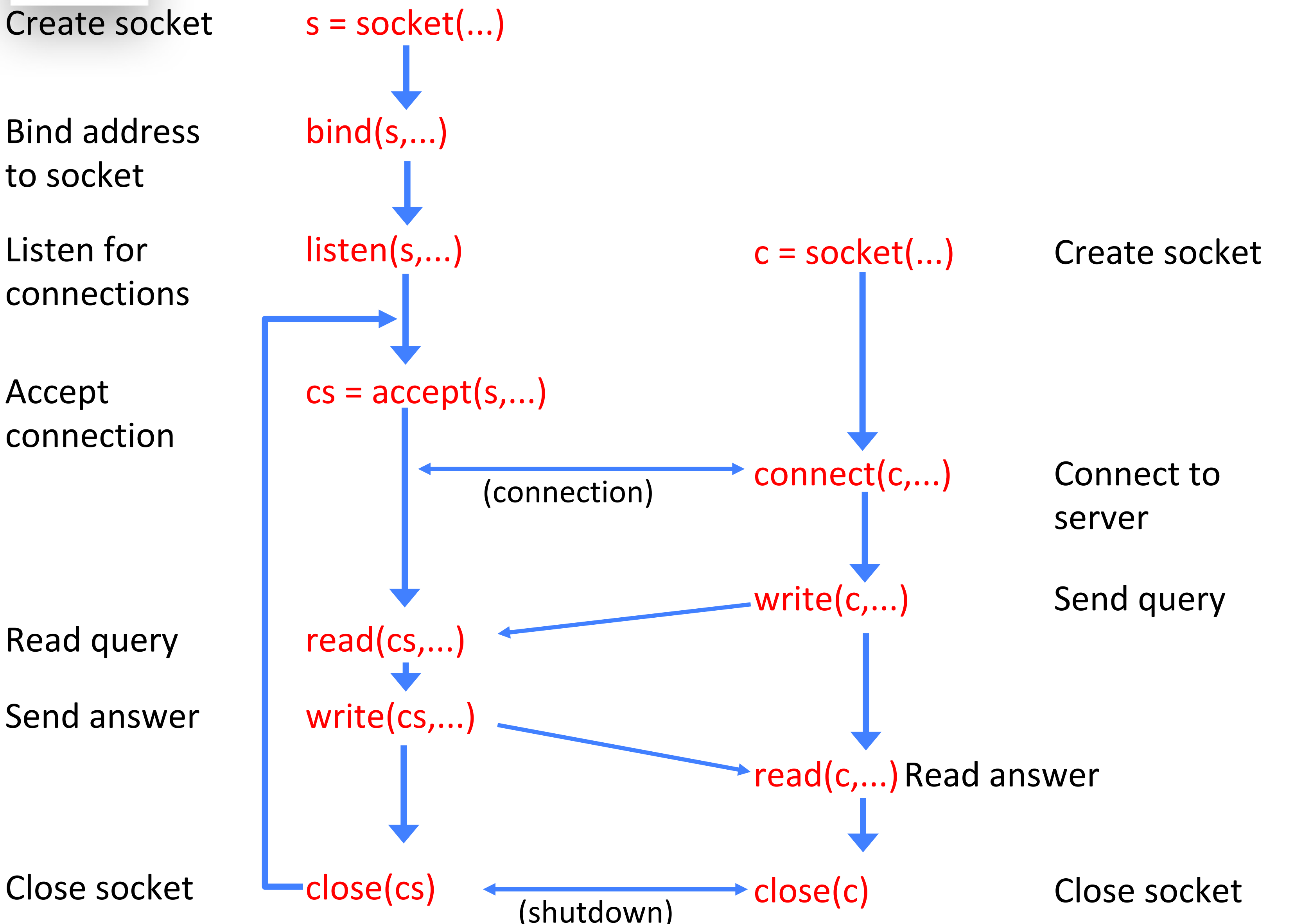
Sockets API

- API for communication between applications
  - Applications create sockets and later use them for sending and receiving data
  - Communication peer identified with:
    - IP adress
      - A 32-bit binary number (IPv4)
      - Identifies destination node
    - Port number
      - 16-bit binary number
      - Identifies application process at each side
  - A session is defined by the 5-tuple:
    - IP addresses of the endpoints
    - Port numbers of the endpoints
    - **Transport protocol**

Services provides by Sockets API

- General
  - Adressing an application through IP/port numbers
- TCP
  - Connection oriented
  - Reliable delivery
  - Byte stream
- UDP
  - Connectionless
  - Unreliable delivery
  - Message oriented

TCP (C)


Create socket
s = socket(...)
Bind address to socket
bind(s,...)
Listen for connections
listen(s,...)
c = socket(...)
Create socket
Accept connection
cs = accept(s,...)
(connection)
connect(c,...)
Connect to server
write(c,...)
Send query
Read query
read(cs,...)
Send answer
write(cs,...)
read(c,...)
Read answer
Close socket
close(cs)
(shutdown)
close(c)
Close socket

```
#include        <stdio.h>
#include        <sys/types.h>
#include        <sys/socket.h>
#include        <netinet/in.h>
#include        <arpa/inet.h>

int                     sock;
struct sockaddr_in      sa;

bzero((char *) &sa, sizeof(sa));
sa.sin_family = AF_INET;
sa.sin_addr.s_addr = inet_addr(...);
sa.sin_port = ...;

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
       error("Can't create TCP socket");

if (connect(sock, (struct sockaddr *) &sa, sizeof(sa)) < 0)
       error("Can't connect to server");

write(sock, ...);
read(sock, ...);

close(sock);
```

Some important things

- Always translate byte ordering
  - htons, htonl, ntohs, ntohl
- Always capture return values (and handle them!)
  - Some system calls are non-blocking
- Always close a TCP connection
  - Avoid pending connection states from old connections
- Some OS:s does not permit reusing ports
  - Can be resolved with a *socket option*

Socket Options

- Used to control socket behavior
  - OS/Protocol stack specific
  - Generic options
  - Protocol specific options
- Types
  - Boolean flags
  - Complex types
    - `int`
    - `Timeval`
    - `in_addr`
    - `Sockaddr`
    - …
- Some options are read-only

(a few) Generic socket options

- SO_BROADCAST
  - Defines whether broadcast is possible or not
- SO_DONTROUTE
  - Bypass normal routing, used by routing daemons
- SO_ERROR
  - Read-only option similar to `errno`
- SO_KEEPALIVE
  - Used by TCP to keep connection up in case of low traffic
- SO_LINGER
  - Controls ACK waiting time at close
- SO_OOBINLINE
  - Enable OOB data to be sent
- SO_RCVBUF, SO_SNDBUF
  - Controls advertised window and sending buffer
- SO_REUSEADDR
  - Enables `bind()` :ing to address that are already in use

Writing a concurrent server (C)

- Design alternatives
  - One child/client
  - One thread/client
  - Preforking processes
  - Prethreading
- Important to understand the options
- Test before you decide what to use

One child/client

- Traditional solution:
  - After `accept()`/`recvfrom()`, call `fork()`
  - Each process needs only a few sockets.
  - Small requests can be serviced in a small amount of time.
- Parent process needs to clean up!!!
  - call `wait()`

One thread/client

- Almost like using `fork()`
  - Call `pthread_create()` instead
- Less overhead when sharing data with other processes
  - Must be done carefully, using `pthread_mutex`

Preforking and Prethreading

- The initial server
  - calls `socket()` and `bind()`,
  - `fork()` or `pthread_create()` a number of children
- Each process is an iterative server
- All children call `accept()`
  - Next incoming connection handled to a child
- Number of children is a performance tradeoff
- Preforking: Server doesn't bother about clients
  - Only manages the children
- Prethreading: Server can do all the `accept()`:s
  - Hand over incoming connection to an existing thread

What is the best alternative?

- Consider
  - Number of simultaneous clients
  - Transaction size (incl. variability)
  - Available system resources

Multi-input checking with `select()`

- Blocking I/O on a set of descriptors
  - Files, Devices, Sockets...
- Create an empty `fd_set`
  - `FD_ZERO(fd_set *fdset);`
- Add descriptors to be monitores
  - `FD_SET(int fd, fd_set *fdset);`
- Call `select()`
- Check the set when `select()` returns
  - `FD_ISSET(int fd, fd_set *fdset);`

UPPSALA UNIVERSITET

Summary

- Different applications have different communication needs
- Two basic services:
  - message forwarding: unreliable, fast, UDP
  - reliable byte stream: reliable, slower, in-order, TCP
- TCP uses several control loops
  - Adaptive timeouts
  - Regulation of the maximum sending window
  - Flow control
- Four different delivery models: uni/broad/multi/any-cast
- The client/server model is common in application design
- Sockets API can be used for network programming
  - A socket is a communication handle abstraction
  - Properties of a socket can be set through socket options
- Different design alternatives for a concurrent server
- select() system call can be used to monitor I/O