ROYAL INSTITUTE
OF TECHNOLOGY

Application Layer

IK2218/EP2120

Peter Sjödin, psj@kth.se
KTH School of ICT

Acknowledgements

- The presentation builds upon material from
  - Previous slides by György Dan, Markus Hidell, Björn Knutsson and Peter Sjödin
  - *Computer Networking: A Top Down Approach*, 6th ed. Jim Kurose, Keith Ross. Addison-Wesley.
  - *TCP/IP Protocol Suite*, 4th ed, Behrouz Foruzan. McGraw-Hill.

Outline

- Introduction to application layer
  - Principles
  - Client-server
  - Peer-to-peer
- Creating network applications
  - Socket programming API
- Learning by examples
  - Structure of application-layer protocols
- Web
  - Hypertext Transfer Protocol (HTTP)
  - Web documents
  - Cookies
- Remote login
  - Telnet and SSH
- Email
  - SMTP
  - POP and IMAP
  - Email message format,
    - RFC-822, MIME
- Multimedia networking
  - Streaming and real-time media
  - RTP

What Do We Use the Internet For?

- E-mail
- Web
- Text messaging
- Remote login
- P2P file sharing
- Multi-user online games
- Streaming stored video and audio
  - YouTube, Hulu, Netflix, Spotify
- Voice over IP
  - Skype, Rebtel, SIP
- Real-time video conferencing
- Social networking
- Search
- Storage
  - Dropbox, Box, SkyDrive, iCloud, Google Drive
- Remote applications
  - Google Apps, Windows Office 365, iCloud/iWorks
- …

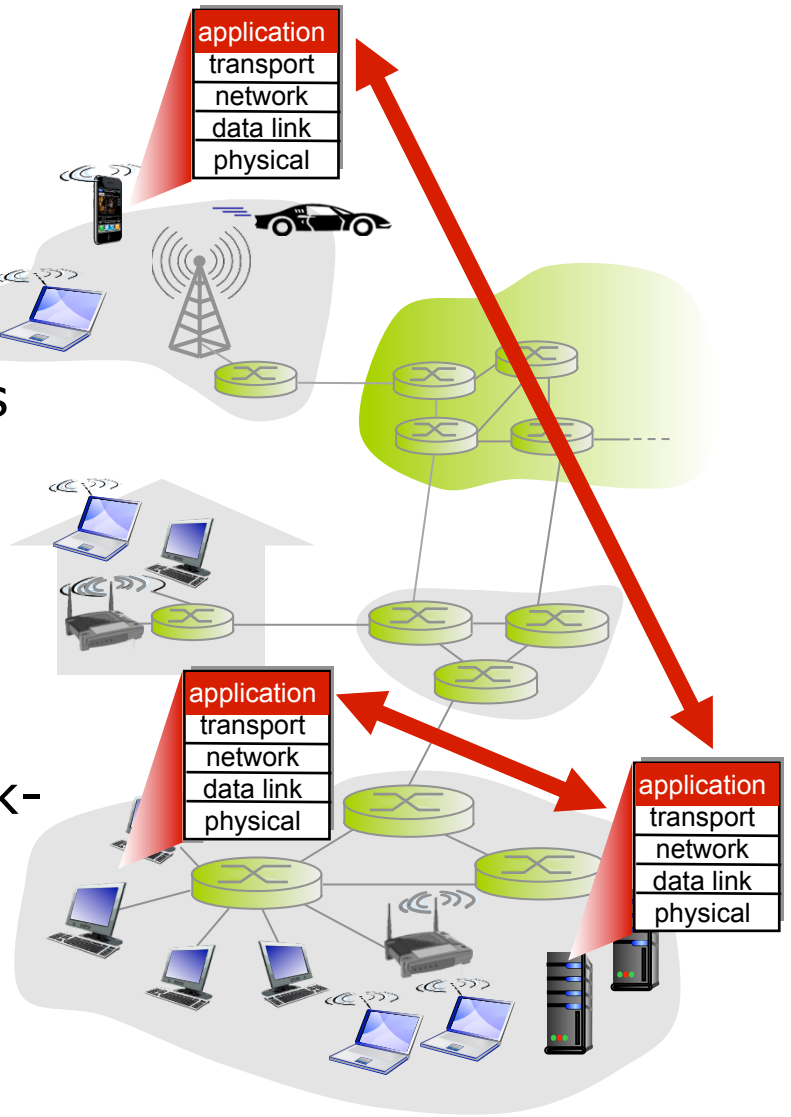
Creating an Application

Write programs that:

- Run on (different) *end systems*
- Communicate over network
  - Web server software communicates with browser software

Applications run on end-systems only

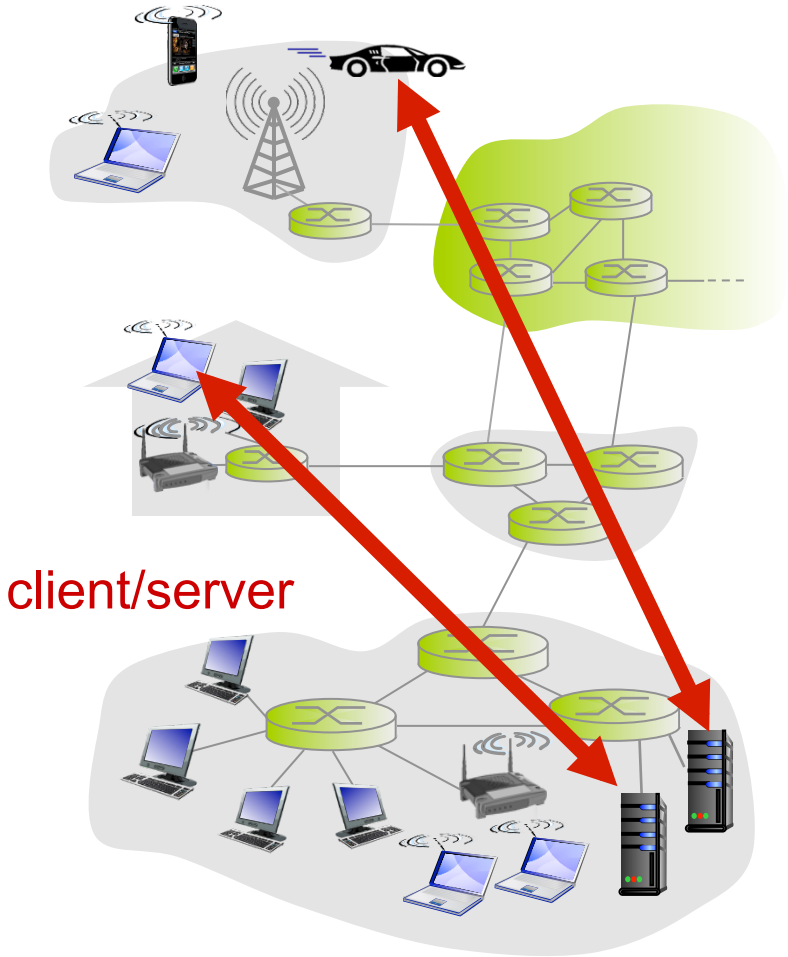
- new services and applications can be designed and deployed quickly
- No need to write software for network-core devices (routers/switches)



Application Architectures

- Possible structure of applications:
  - Client-server
  - Peer-to-peer (P2P)

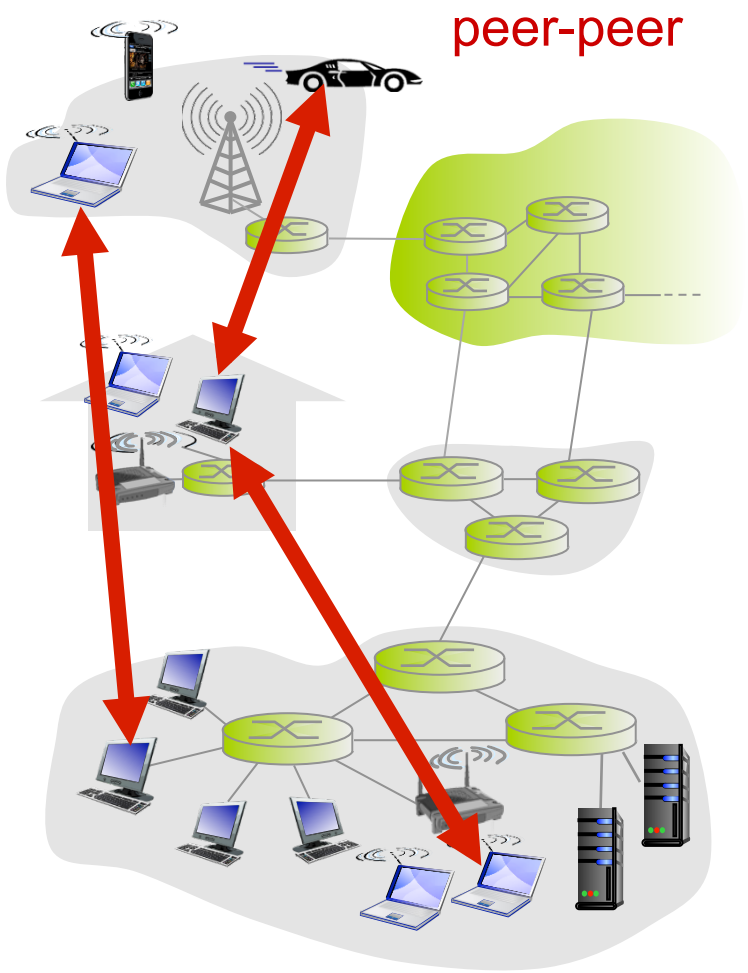
Client-Server Architecture



- Server:
  - Always on
  - Permanent IP address
  - Well-known port
    - E.g. 80 for HTTP
  - Data centers for scaling
- Clients:
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP addresses
  - "Ephemeral" ports
    - Short-lived, dynamic
  - Do not communicate directly with each other

P2P Architecture

- No always-on server
- Arbitrary end-systems directly communicate
- Peers request service from other peers, provide service in return to other peers
  - Self scalability
    - new peers bring new service capacity, as well as new service demands
- Peers are intermittently connected and change IP addresses
  - Complex management

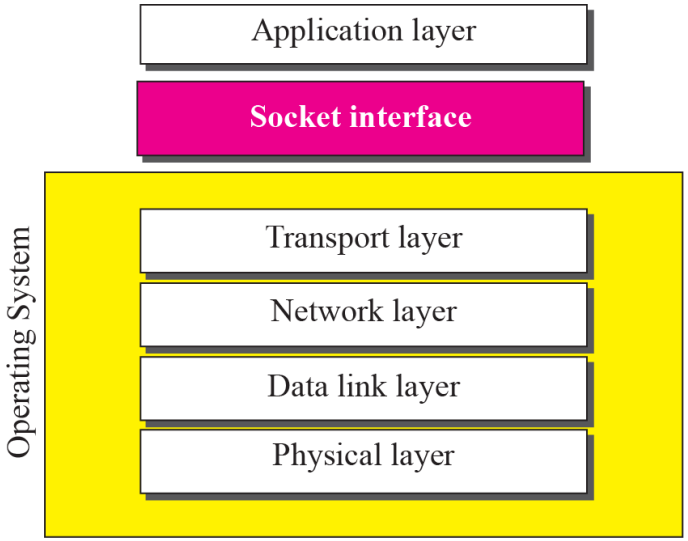


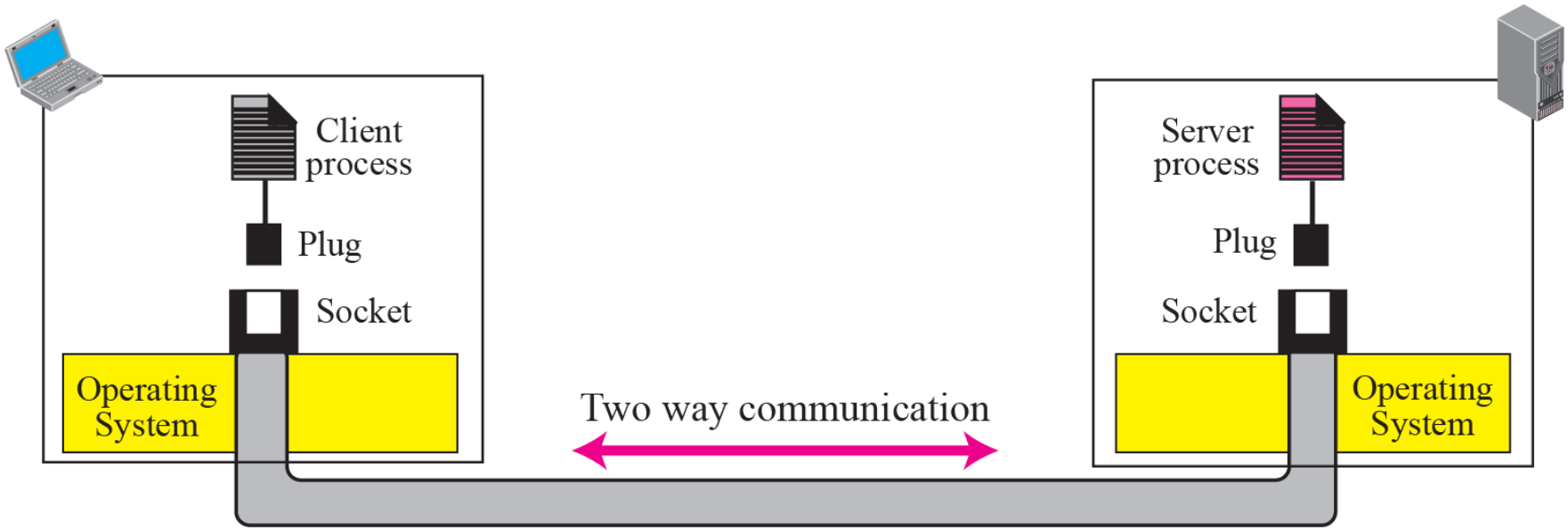
Outline

- Introduction to application layer
  - Principles
  - Client-server
  - Peer-to-peer
- Creating network applications
  - Socket programming API
- Learning by examples
  - Structure of application-layer protocols
- Web
  - Hypertext Transfer Protocol (HTTP)
  - Web documents
  - Cookies
- Remote login
  - Telnet and SSH
- Email
  - SMTP
  - POP and IMAP
  - Email message format,
    - RFC-822, MIME
- Multimedia networking
  - Streaming and real-time media
  - RTP

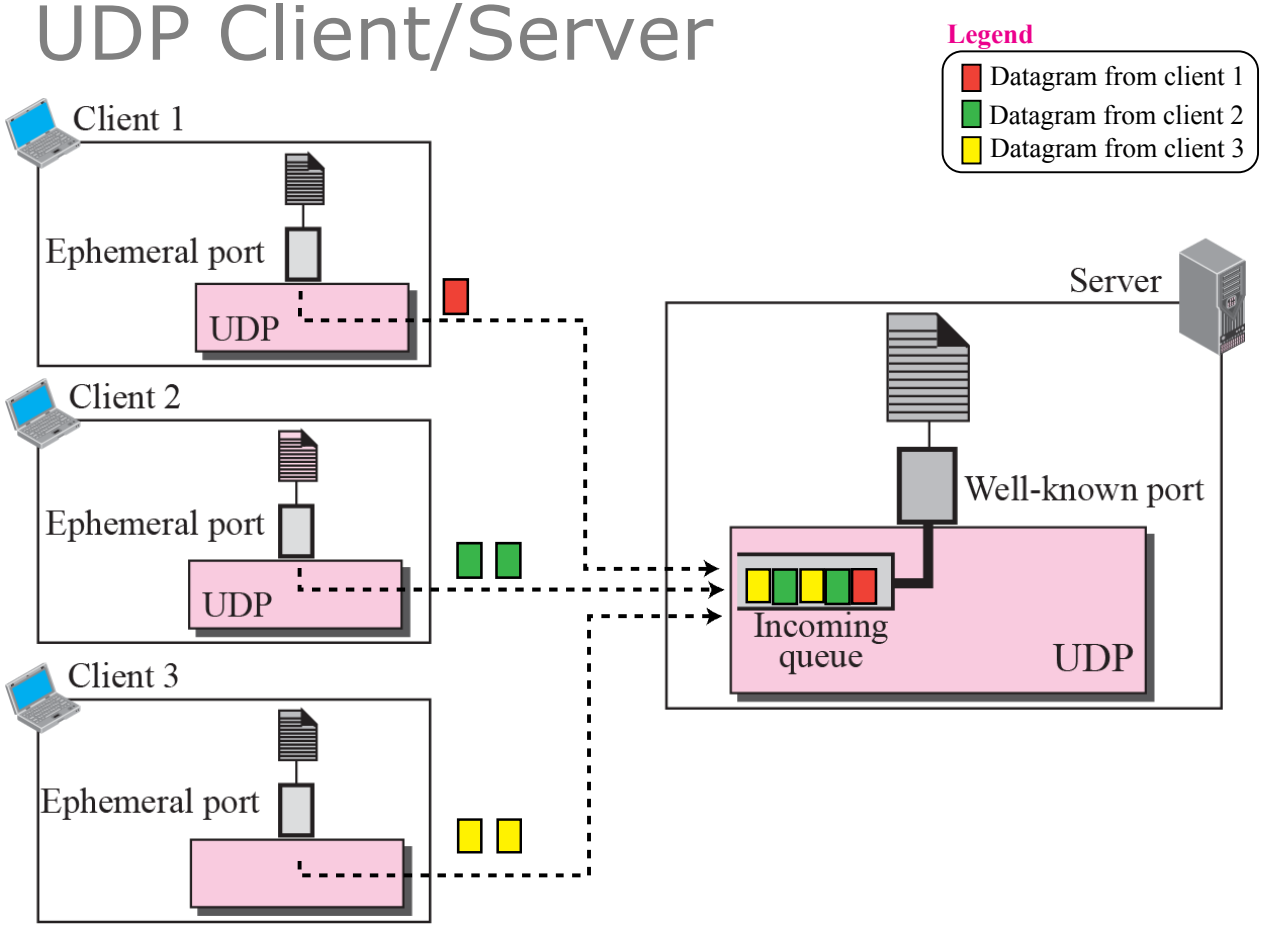
Socket API

- *Socket* – represents endpoint for communication
- Socket API
  - Application Programming Interface for network applications
  - Socket addresses
    - IP address + port number



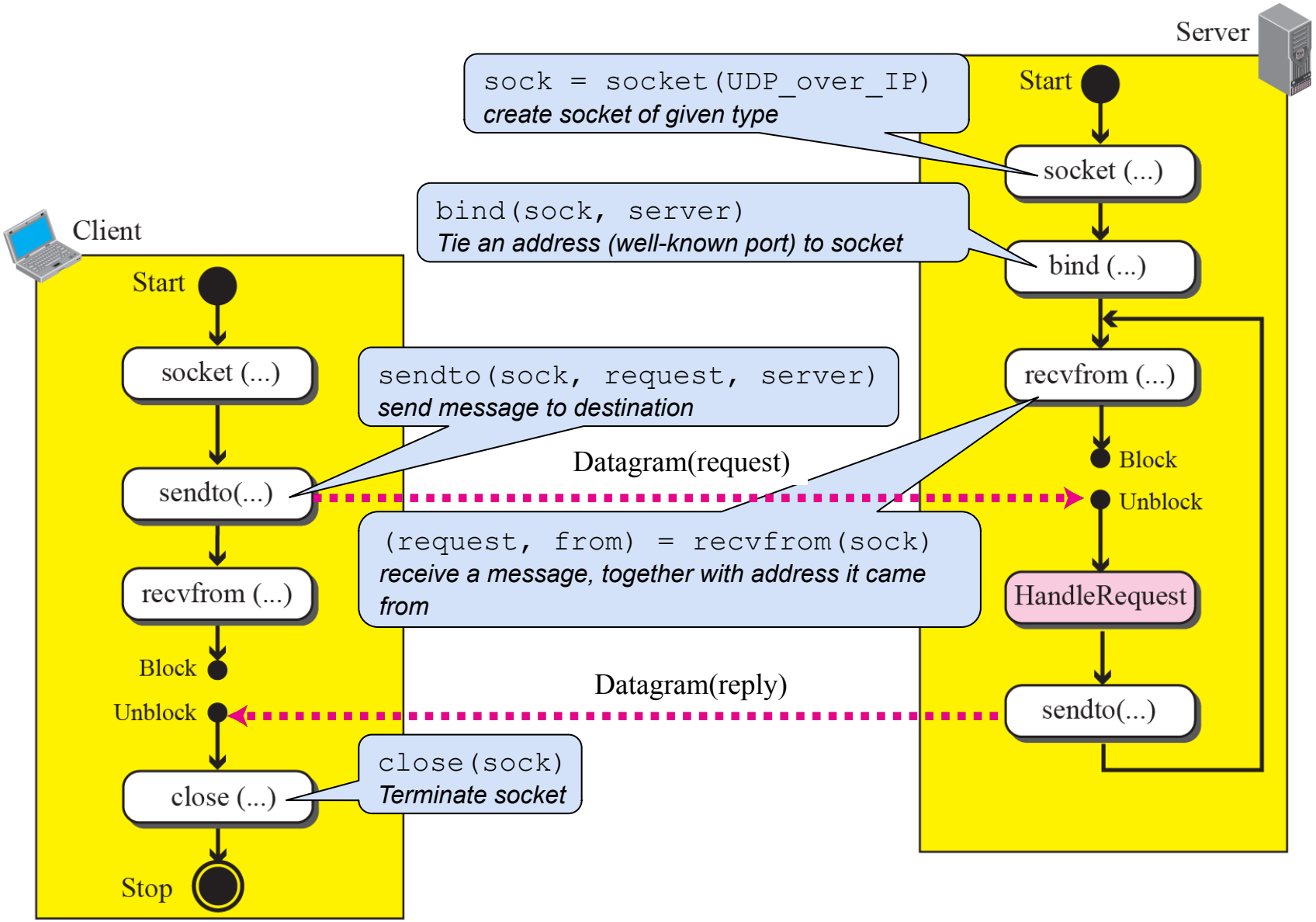


UDP Client/Server



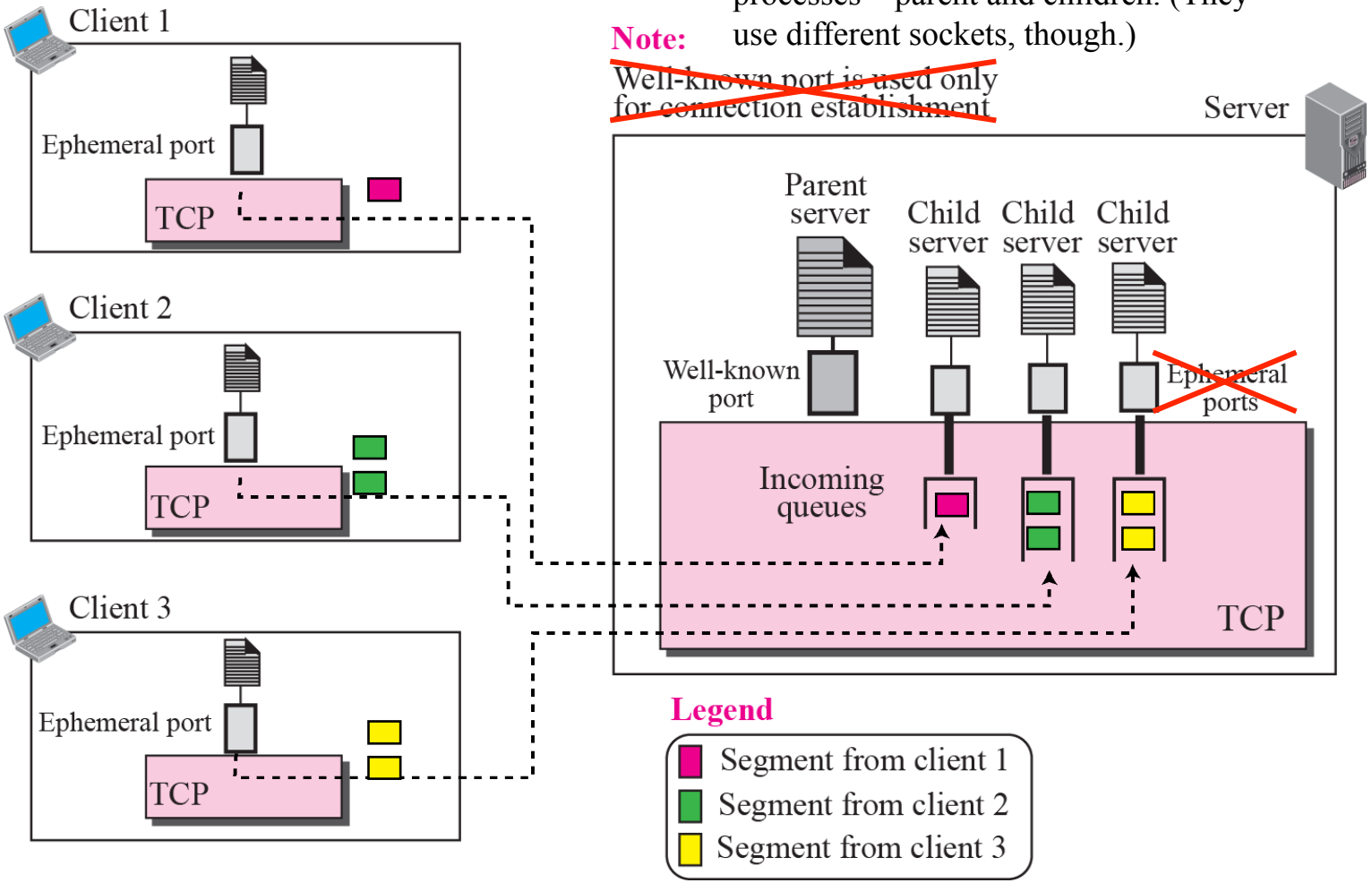
- Sequential server
  - Process one client request at a time

UDP Client/Server Implementation



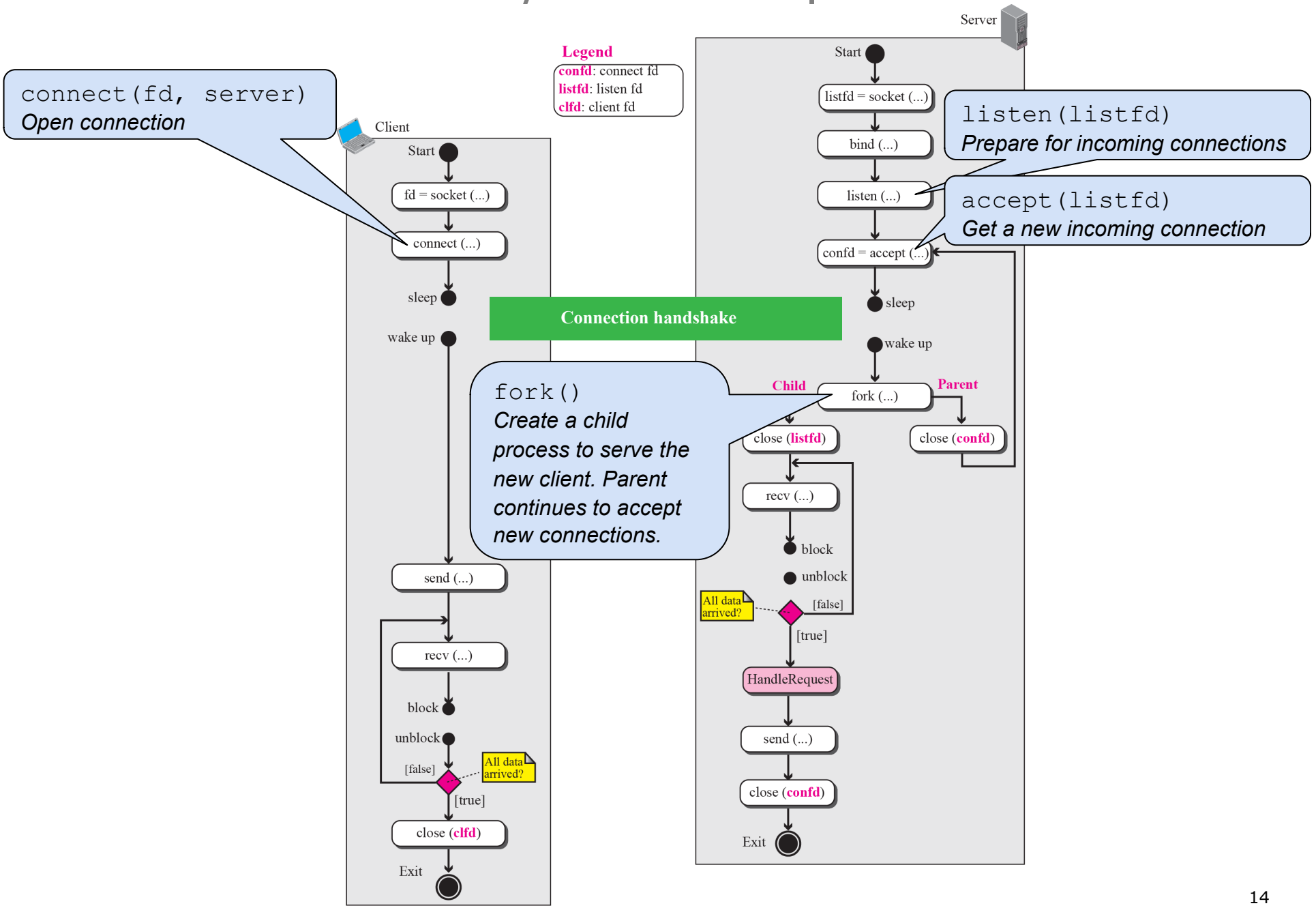
TCP Client/Server

Well-known port is used for all server processes – parent and children. (They use different sockets, though.)



- Concurrent server
  - Handle multiple clients at the same time

TCP Client/Server Implementation



Outline

- Introduction to application layer
  - Principles
  - Client-server
  - Peer-to-peer
- Creating network applications
  - Socket programming API
- Learning by examples
  - Structure of application-layer protocols
- Web
  - Hypertext Transfer Protocol (HTTP)
  - Web documents
  - Cookies
- Remote login
  - Telnet and SSH
- Email
  - SMTP
  - POP and IMAP
  - Email message format,
    - RFC-822, MIME
- Multimedia networking
  - Streaming and real-time media
  - RTP

Application-Layer Protocols

- Application-layer protocols define
  - Types of messages
    - Request, response, etc
  - Message syntax
    - Fields in messages
    - How fields are delineated
  - Message semantics
    - Meaning of information in fields
  - Rules for sending and responding
    - When and how processes send messages
- Open protocols:
  - Defined in IETF RFCs
  - Allows for interoperability
  - HTTP and SMTP for example
- Proprietary protocols:
  - Skype, for example

Outline

- Introduction to application layer
  - Principles
  - Client-server
  - Peer-to-peer
- Creating network applications
  - Socket programming API
- Learning by examples
  - Structure of application-layer protocols
- Web
  - Hypertext Transfer Protocol (HTTP)
  - Web documents
  - Cookies
  - HTTP/2
- Remote login
  - Telnet and SSH
- Email
  - SMTP
  - POP and IMAP
  - Email message format,
    - RFC-822, MIME
- Multimedia networking
  - Streaming and real-time media
  - RTP

Web and HTTP

First, a review…

- *Web page* consists of *objects*
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of *base HTML-file* which includes *several referenced objects*
- Each object is addressable by a *URL, Uniform Resource Locator,* e.g.,

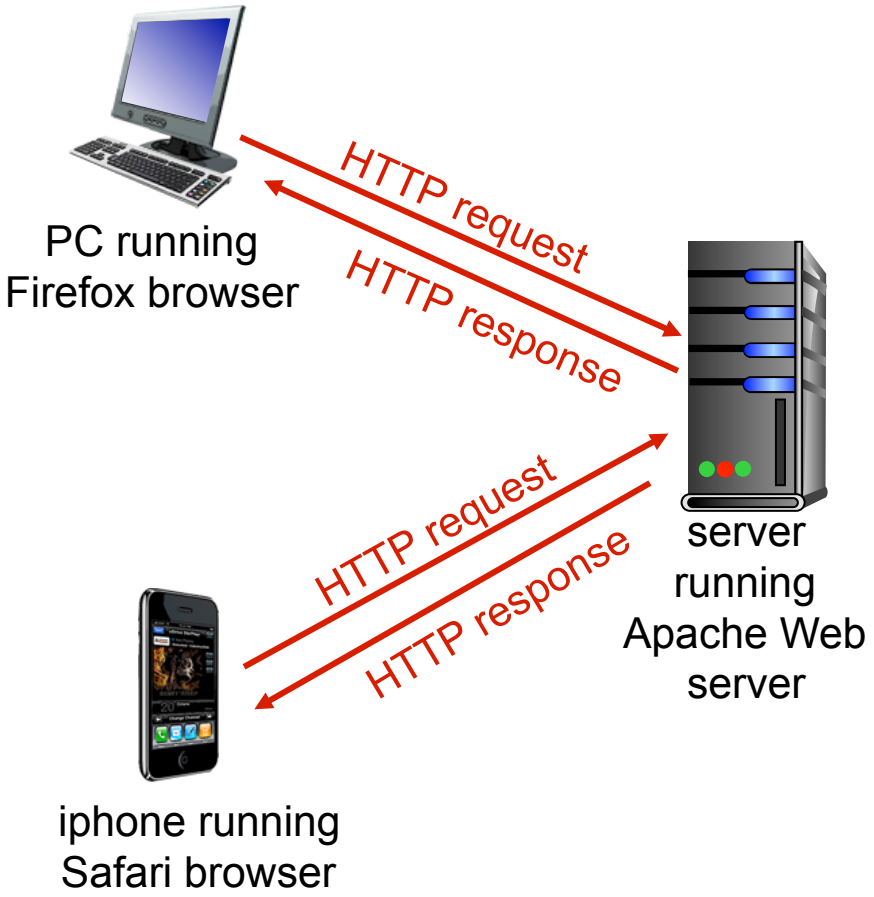
`http://www.someschool.edu/someDept/pic.gif`

protocol host name path name

HTTP Overview

HTTP: hypertext transfer protocol

- Web application layer protocol
- client/server model
  - *client:*
    - browser that requests, receives, (using HTTP) and “displays” Web objects
  - *server:*
    - Web server sends (using HTTP) objects in response to requests



HTTP Overview (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

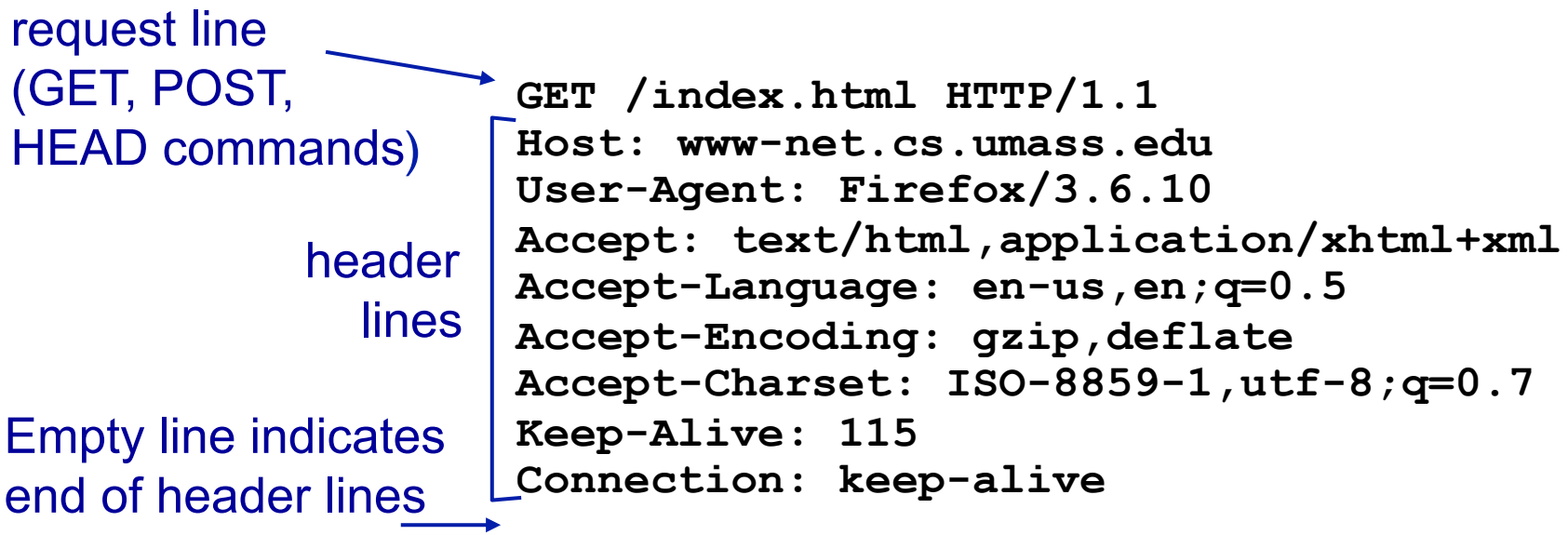
aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP Request Message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII text (human-readable format)



Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Method Types

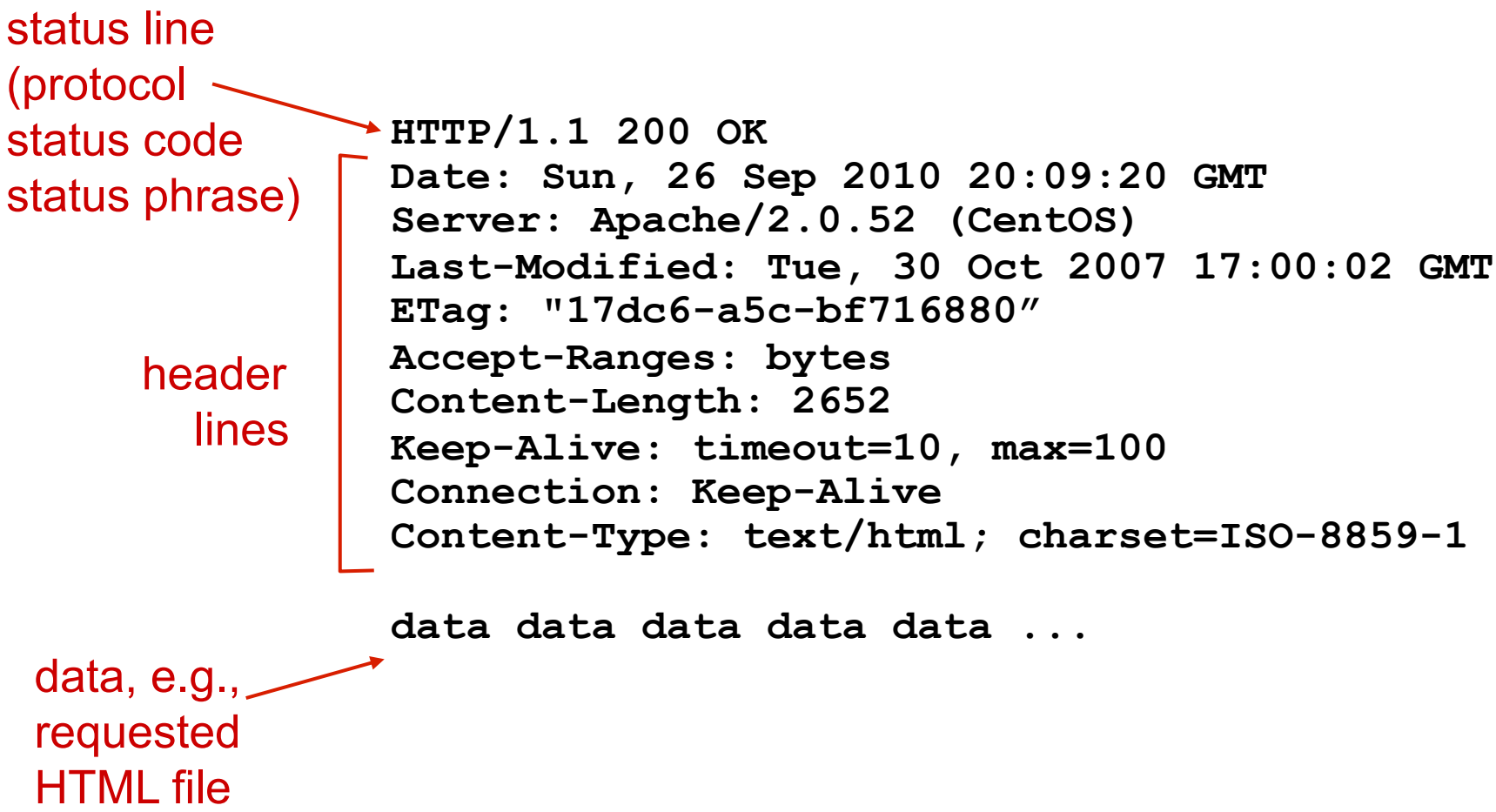
HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP Response Message



HTTP Response Status Codes

- status code appears in first line in server-to-client response message.
- some sample codes:

200 OK
- request succeeded, requested object later in this response

301 Moved Permanently
- requested object moved, new location specified later in this response (Location:)

400 Bad Request
- Request not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

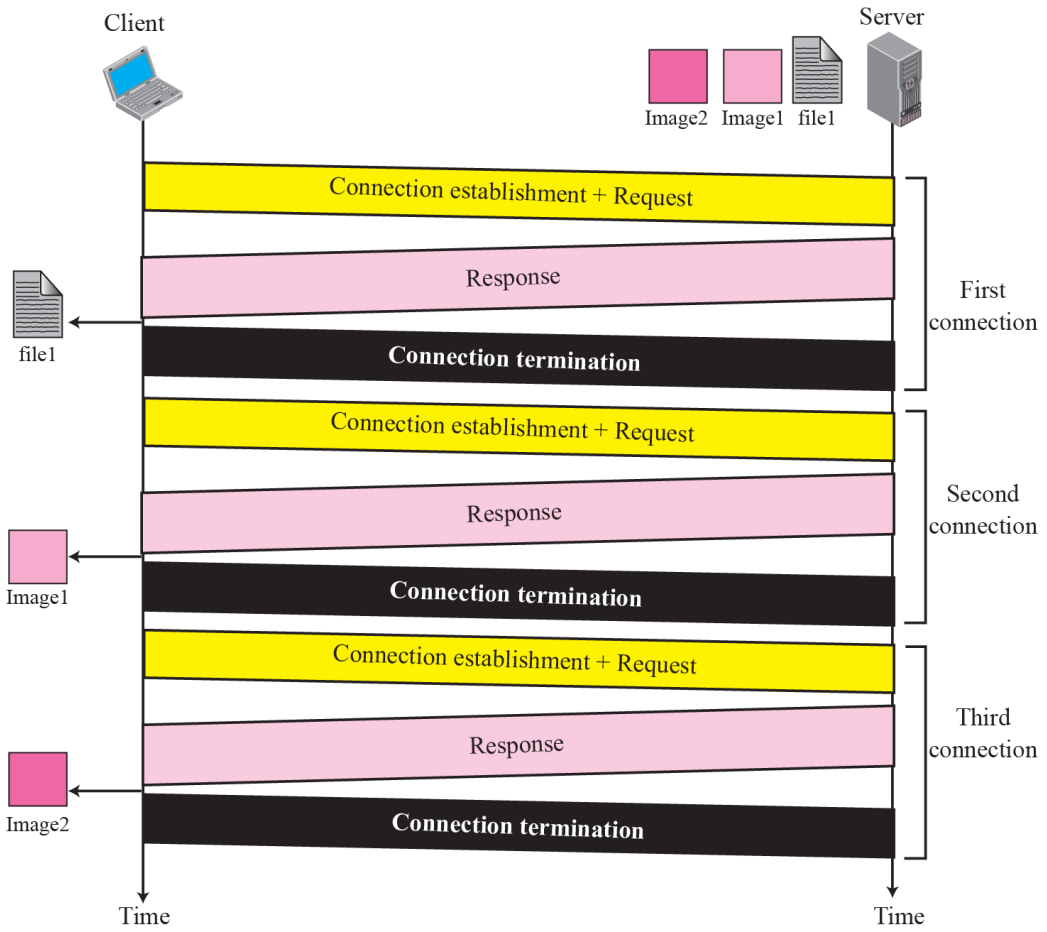
by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

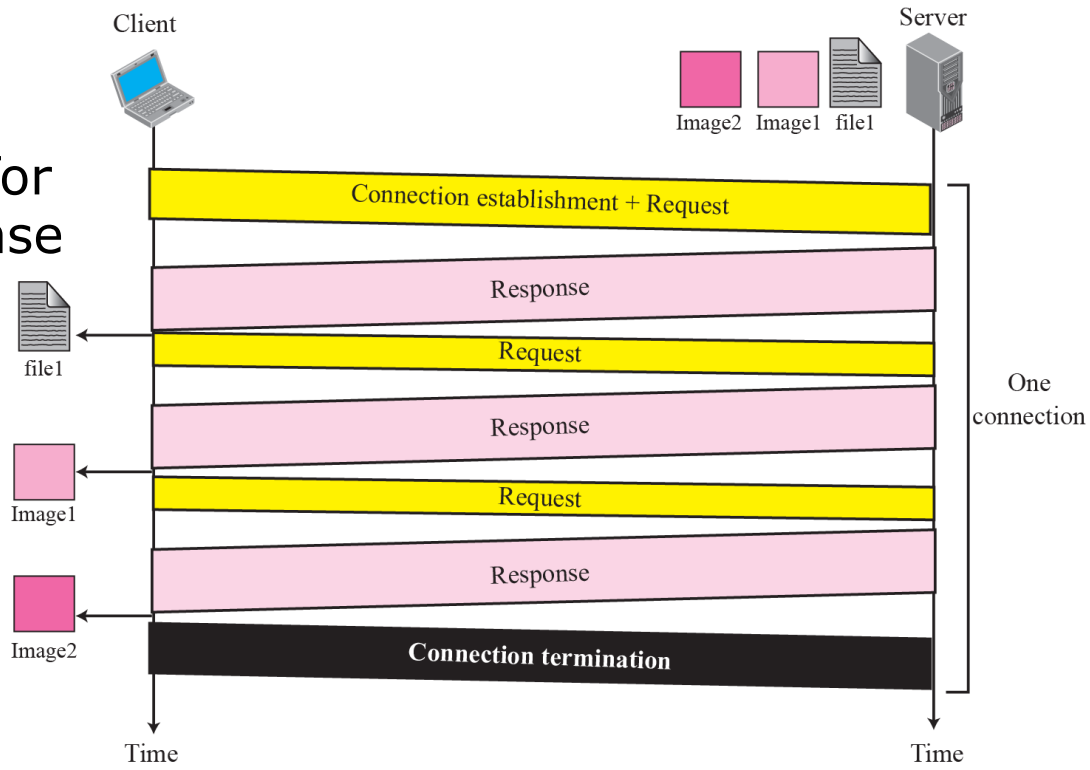
Non-persistent Connection

- One TCP connection per HTTP request/response transaction
- Having several TCP connections could increase load on network (congestion) and server
- TCP handshake and slow start for each HTTP transaction



Persistent Connection

- Reuse same TCP connection for multiple HTTP request/response transactions
  - Default as of HTTP 1.1
- How long should connection be left open?
  - Occupies server resources
  - Controlled by "Keep-Alive" header



HTTP request

```
GET /index.html HTTP/1.1
Connection: keep-alive
```

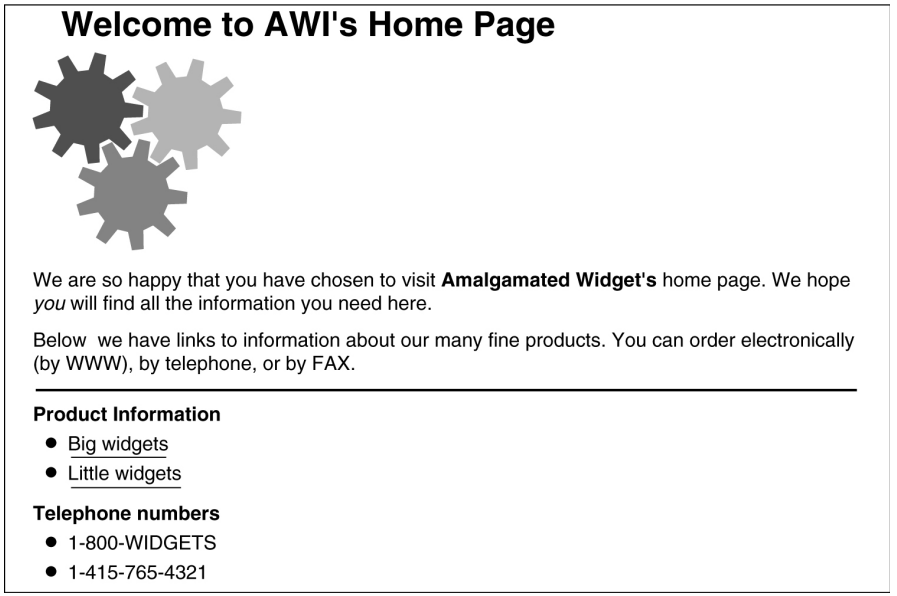
HTTP response

```
HTTP/1.1 200 OK
Keep-Alive: timeout=10, max=100
Connection: keep-alive
```

HyperText Markup Language (HTML)

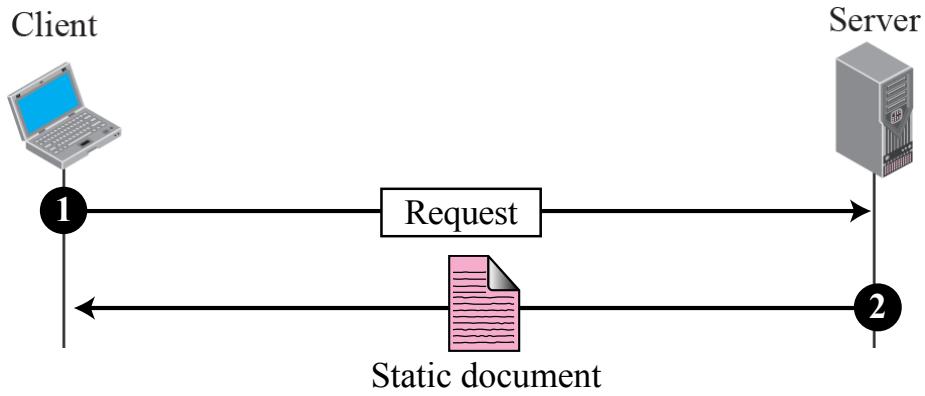
```
<html>
<head><title> AMALGAMATED  WIDGET,  INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page</h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI  Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated  Widget's </b>
home page.We hope <i> you </i> will  find  all the information you need here.
<p>Below we have links to information  about our many fine products.
You can order  electronically (by WWW), by  telephone, or by fax. </p>
<hr>
<h2> Product information  </h2>
<ul>
   <li> <a href="http://widget.com/products/big">  Big widgets </a>
   <li> <a href="http://widget.com/products/little">  Little widgets </a>
</ul>
<h2> Telephone numbers</h2>
<ul>
   <li> By telephone: 1-800-WIDGETS
   <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

HTML for sample web page



Formatted web page

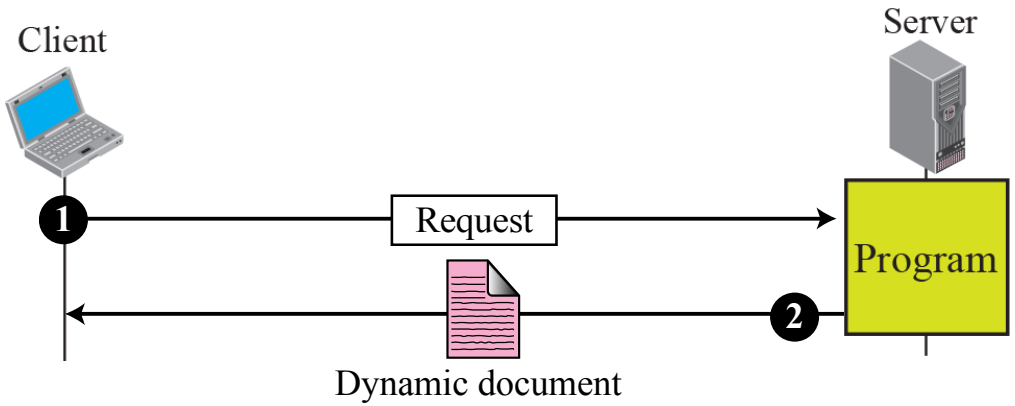
Web Documents



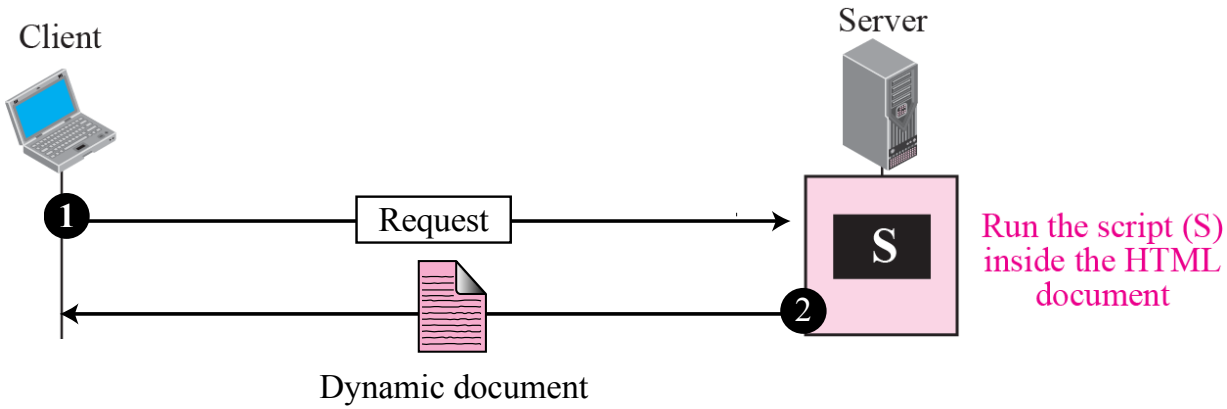
- Static documents
  - Fixed document on server
  - Same for all
- Dynamic documents
  - Generated by running a program on the server
    - For each request
- Active documents
  - Server sends a program to run on the client

Dynamic Documents

- Common Gateway Interface (CGI)
  - Rules and formats for running server program
    - C, C++, sh, bash, perl, …

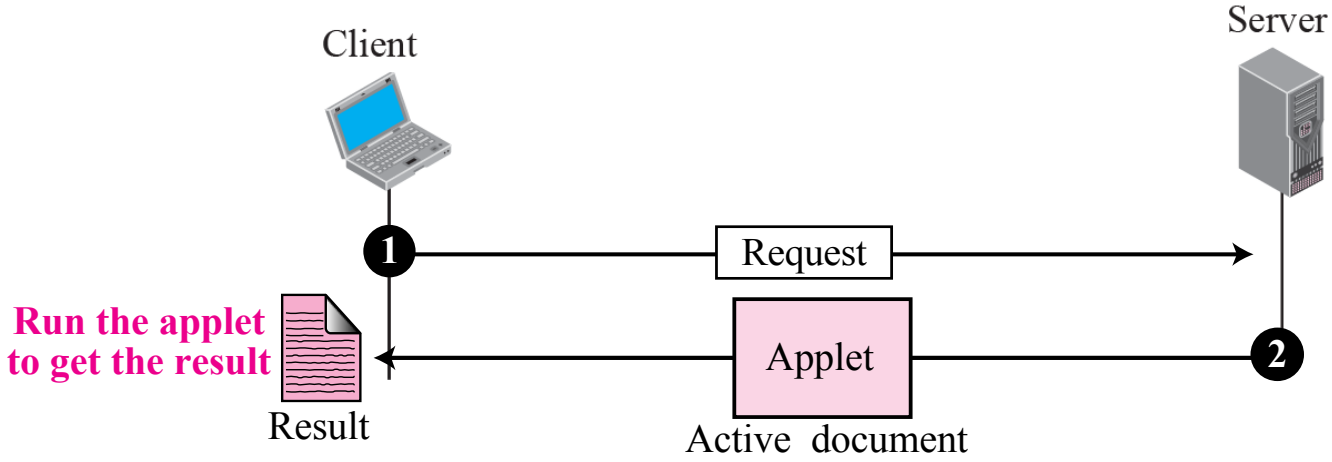


- Server-side script
  - Embedding scripts in HTML code
    - PHP, JSP, ASP, ColdFusion, …

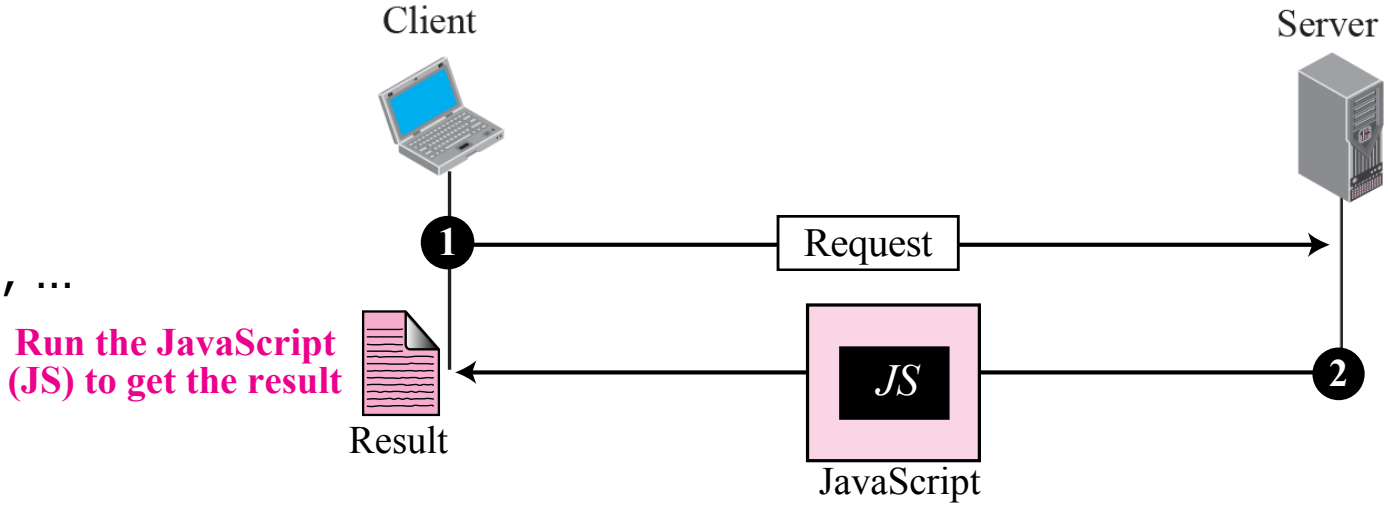


Active Documents

- Java applets
  - Compiled
  - Binary format – Java bytecode



- Client-side script
  - Source code
  - JavaScript, Ajax, …



User-Server State: Cookies

Many Web sites use cookies

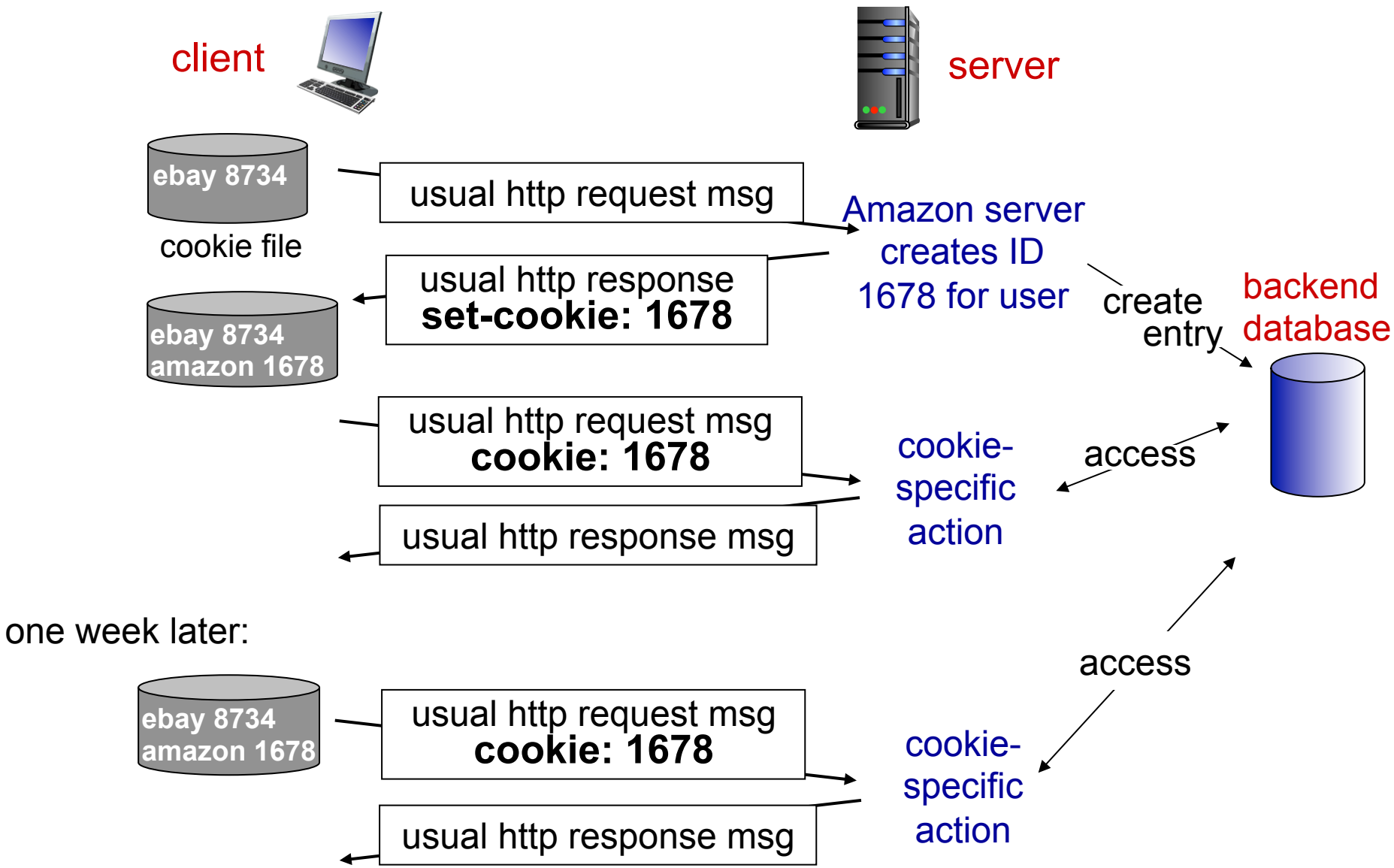
Four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan always accesses Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping “state” (cont.)



Cookies (continued)

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

How to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- For instance, you may supply name and e-mail to sites

aside

Controlling cookies:

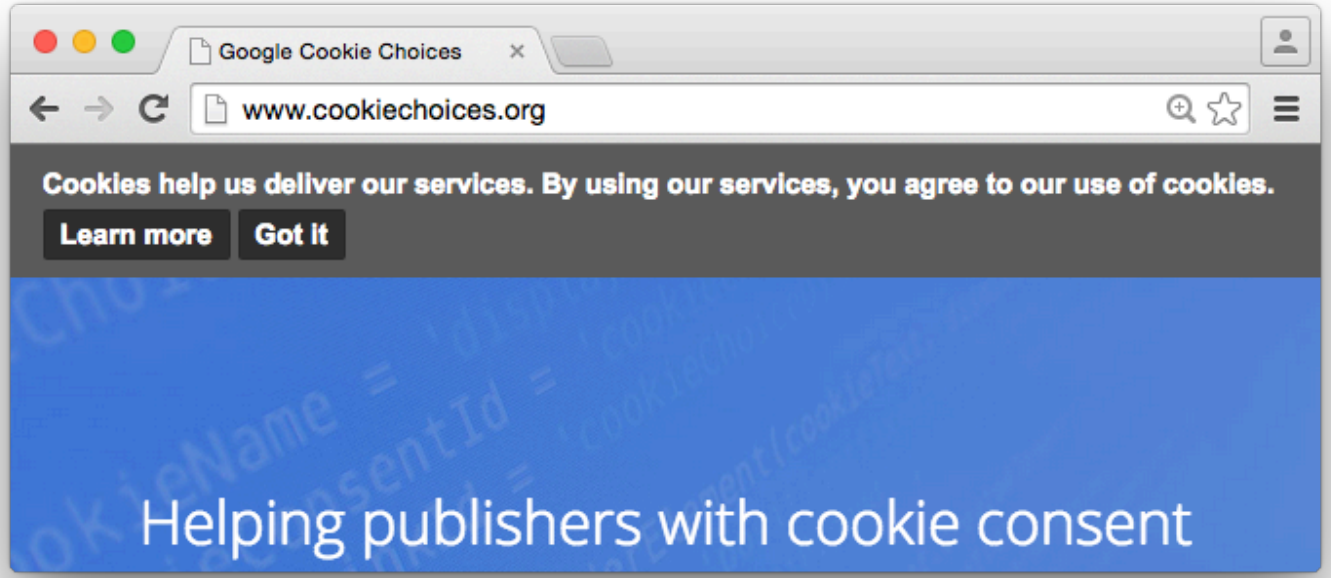
- Cookie acceptance policy in browser
- List and remove cookies manually

Laws and Regulations

In EU, placing a cookie on a user's computer requires that the user is informed, and that the user consents to it. (This applies not only to cookies, but to any similar technology that stores and accesses information on the user's device.)

Lagen om elektronisk kommunikation
http://www.pts.se/sv/Bransch/Regler/Lagar/Lag-om-elektronisk-kommunikation/Cookies-kakor 2015-09-21

EU Cookie Law (ePrivacy directive, 2002/58/EC)
http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:EN:HTML 2015-09-21

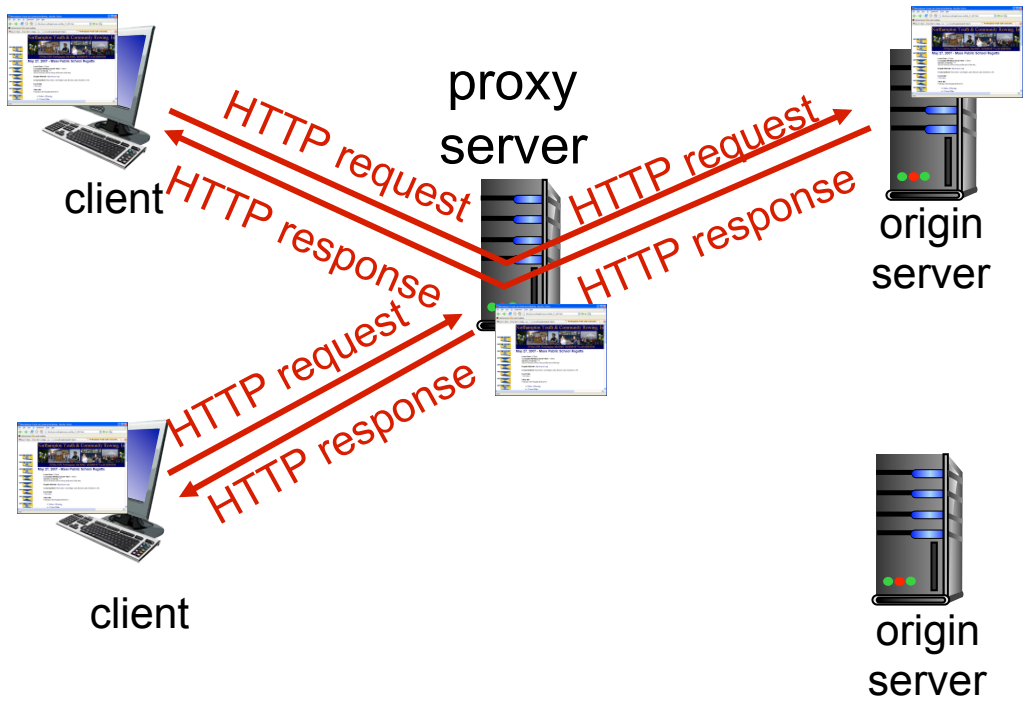


http://www.cookiechoices.org 2015-09-21

Web caches (proxy server)

goal: satisfy client request without involving origin server

- User configures browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - if object in cache
    - cache returns object
  - else
    - cache requests object from origin server,
    - then returns object to client



More About Web Caching

- Cache acts as both client and server
  - server for original requesting client
  - client to origin server
- Typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- Reduce response time for client request
- Reduce traffic on an organization's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

HTTP/2

- HTTP performance trends
  - More HTTP transfers per page
  - More data per transfer
  - Request/response stop-and-wait
- TCP efficiency
  - Congestion control has little effect with many short connections
  - Redundancy with same information sent many times
  - Stop-and-wait nature of TCP handshakes
- User experience suffers as page load time increases
- Negative influence on server load and performance

HTTP/2 (continued)

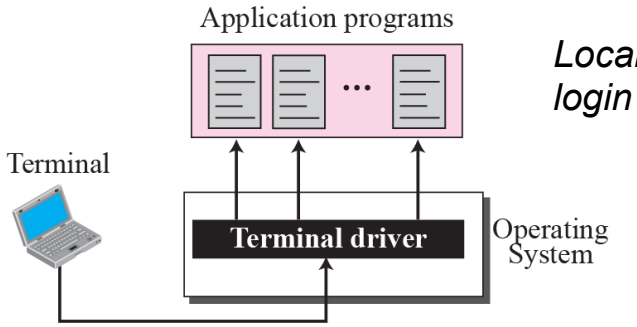
- Multiplexing to support loading of multiple objects at the same time over single connection
- More compact header format
  - Binary format (not text)
  - Compression to remove redundancy
- Advanced features, such as server push
  - Server knows which objects the browser will request next – send them in advance
- Backward compatibility – version negotiation
- Originates from SPDY research project initiative by Google

- HTTP/2 home page https://http2.github.io/
- RFC 7540 – Hypertext Transfer Protocol version 2 (HTTP/2)
- RFC 7541 – HPACK: Header Compression for HTTP/2

Outline

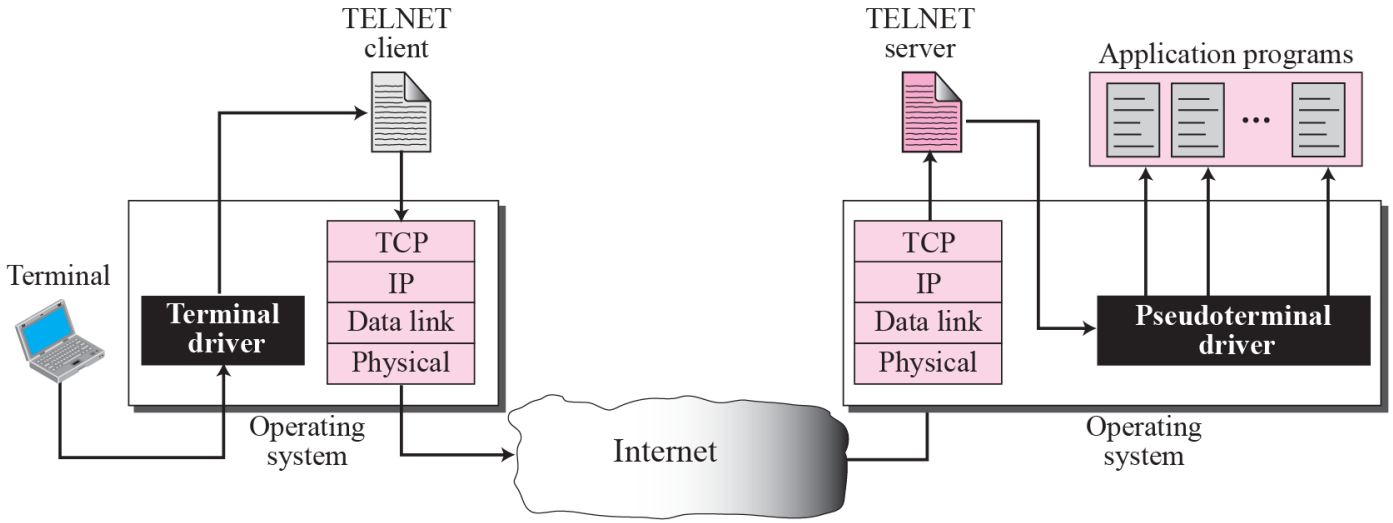
- Introduction to application layer
  - Principles
  - Client-server
  - Peer-to-peer
- Creating network applications
  - Socket programming API
- Learning by examples
  - Structure of application-layer protocols
- Web
  - Hypertext Transfer Protocol (HTTP)
  - Web documents
  - Cookies
- Remote login
  - Telnet and SSH
- Email
  - SMTP
  - POP and IMAP
  - Email message format,
    - RFC-822, MIME
- Multimedia networking
  - Streaming and real-time media
  - RTP

Telnet Remote Login



Local login

- Run applications from a CLI
  - Command Line Interface
    - Text commands
  - Windows Command Prompt and PowerShell
  - Unix/Linux shell
    - sh, ksh, csh, tcsh, bash, …



Remote login

Telnet

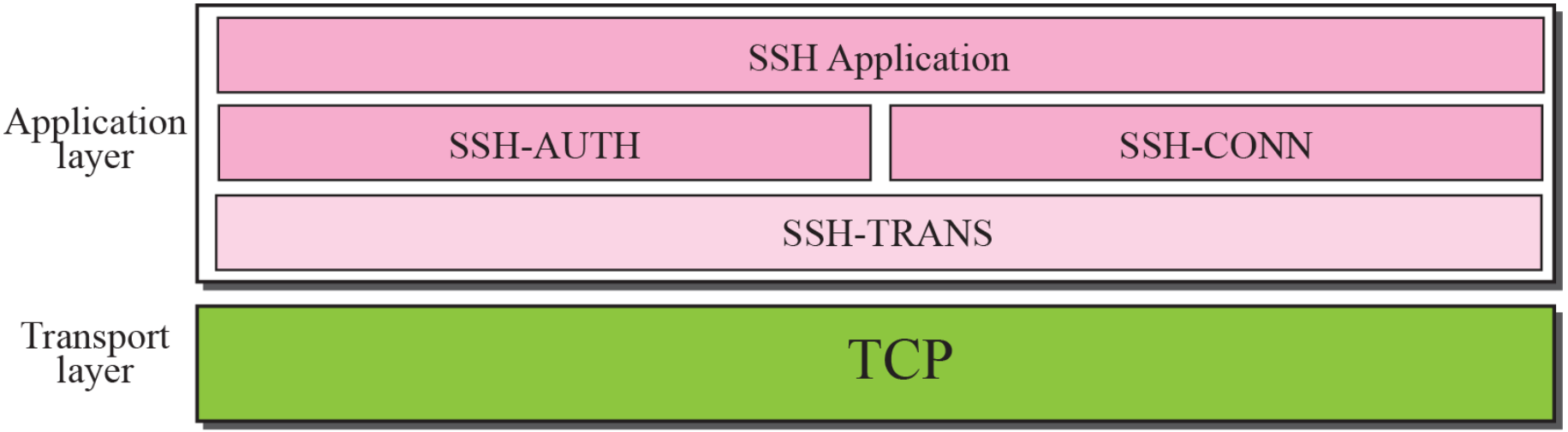
- One of the first Internet protocols
  - RFC 15
- Intended for remote login
  - Text commands over TCP connection
    - Default port 23
- Can be used to test text-based protocols in general
  - SMTP, HTTP, …

```
~$ telnet server
Trying 192.168.13.14...
Connected to server.
Escape character is '^]'.

Welcome to Ubuntu 14.04.1 LTS

server login:
```

```
~$ telnet www.kth.se 80
Trying 130.237.32.143...
Connected to www.kth.se (130.237.32.143).
Escape character is '^]'.
GET / HTTP/1.1
Host: www.kth.se

HTTP/1.1 200 OK
…
```

SSH – Secure Shell

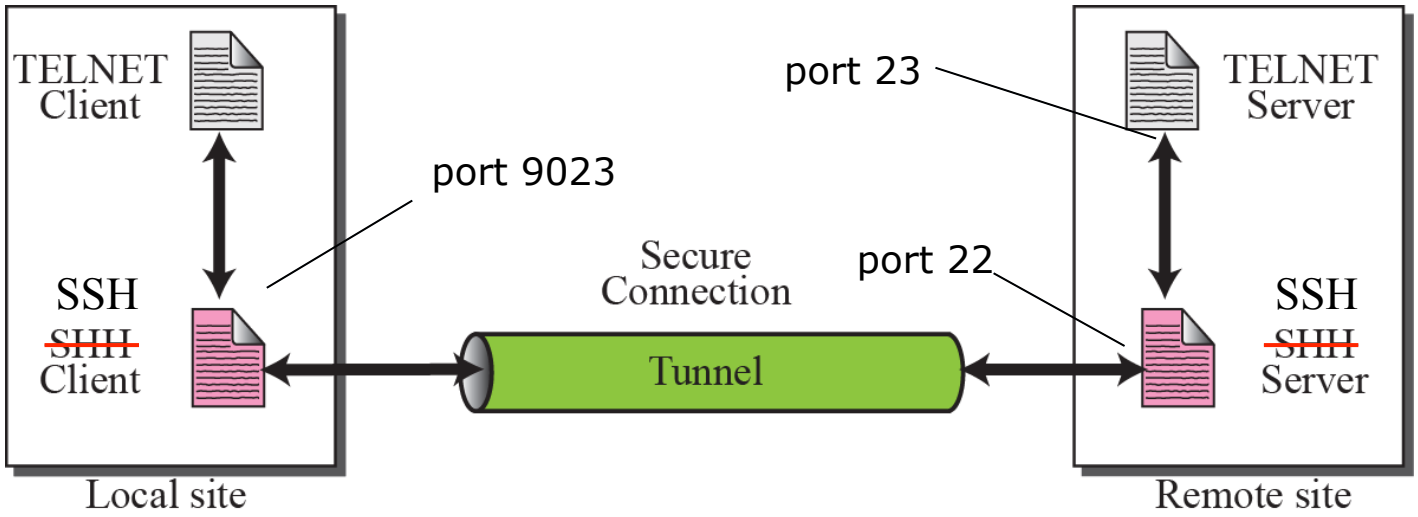
- Telnet considered insecure
  - No encryption – eavesdropping
  - No authentication of client/server
- SSH
  - Encryption and authentication
  - Create a secure (encrypted and authenticated) channel over TCP
  - Default port 22



Port Forwarding

- Create a secure connection, *tunnel*, between (TCP) ports on two machines
- SSH client/server act as *proxies*
- In this way, any legacy (insecure) application can run over a secure connection

  - For example, telnet over a secure tunnel:
    - Set up SSH tunnel with port forwarding from port 9023 on "Client" to port 23 (telnet) on "Server"
    - Command "telnet localhost 9023" on "Client" will connect to telnet server on port 23 on "Server" via SSH tunnel



Remote Login Applications

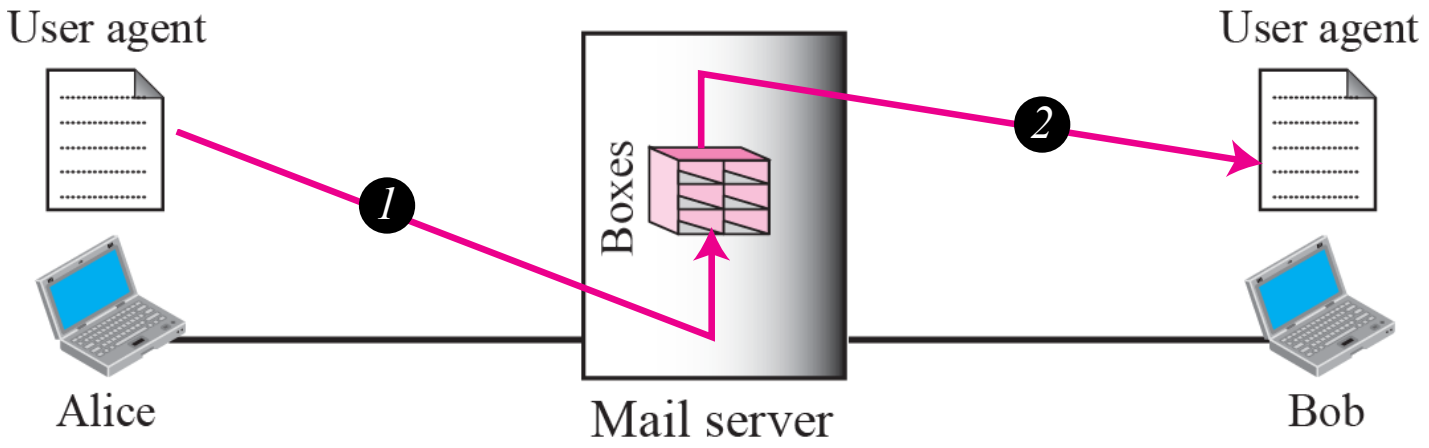
- "telnet" application for command line interface (CLI)
  - Linux/Unix, Windows, …
- OpenSSH
  - Open source, part of OpenBSD project
  - CLI
    - "ssh" command
  - Ported to most platforms
  - http://www.openssh.org/
- PuTTY
  - GUI-based Windows application
  - SSH and telnet
  - Open source, http://www.putty.org/

Outline

- Introduction to application layer
  - Principles
  - Client-server
  - Peer-to-peer
- Creating network applications
  - Socket programming API
- Learning by examples
  - Structure of application-layer protocols
- Web
  - Hypertext Transfer Protocol (HTTP)
  - Web documents
  - Cookies
- Remote login
  - Telnet and SSH
- Email
  - SMTP
  - POP and IMAP
  - Email message format,
    - RFC-822, MIME
- Multimedia networking
  - Streaming and real-time media
  - RTP

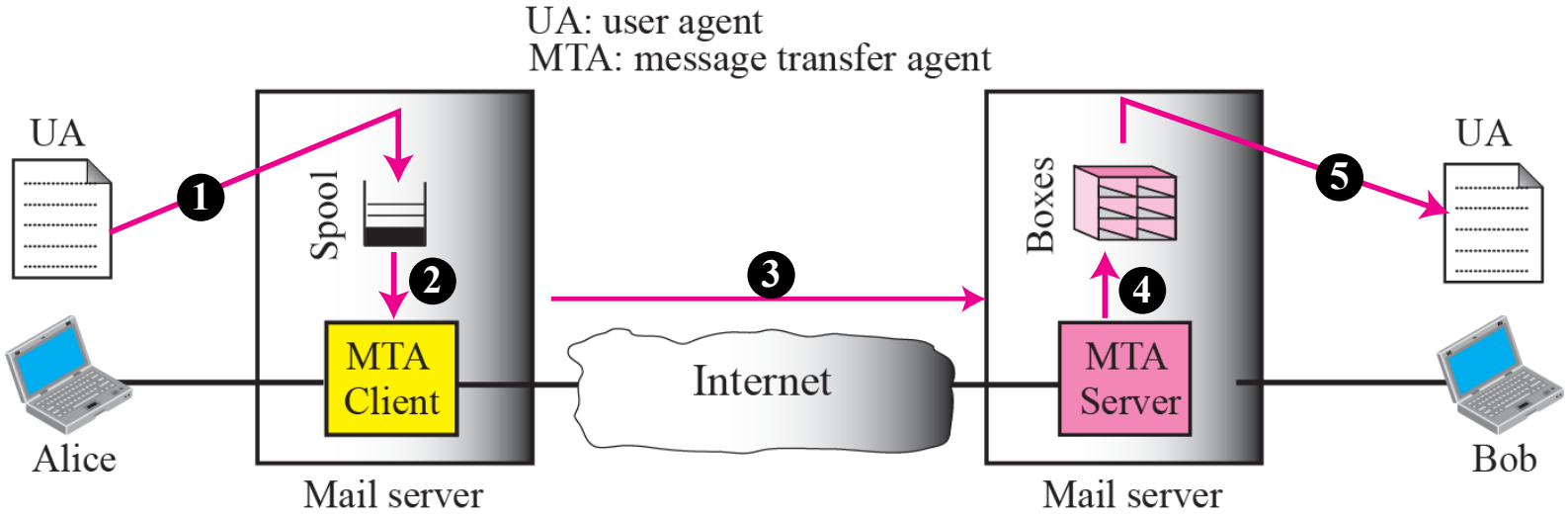
User Agent and Mail Server

- User Agent
  - "Mail reader"
  - Program to create and read e-mail
  - Examples: Outlook, Thunderbird, Kmail, Envelope, …
- Mail server
  - Keeps users' e-mail in *mailboxes*



Message Transfer Agent

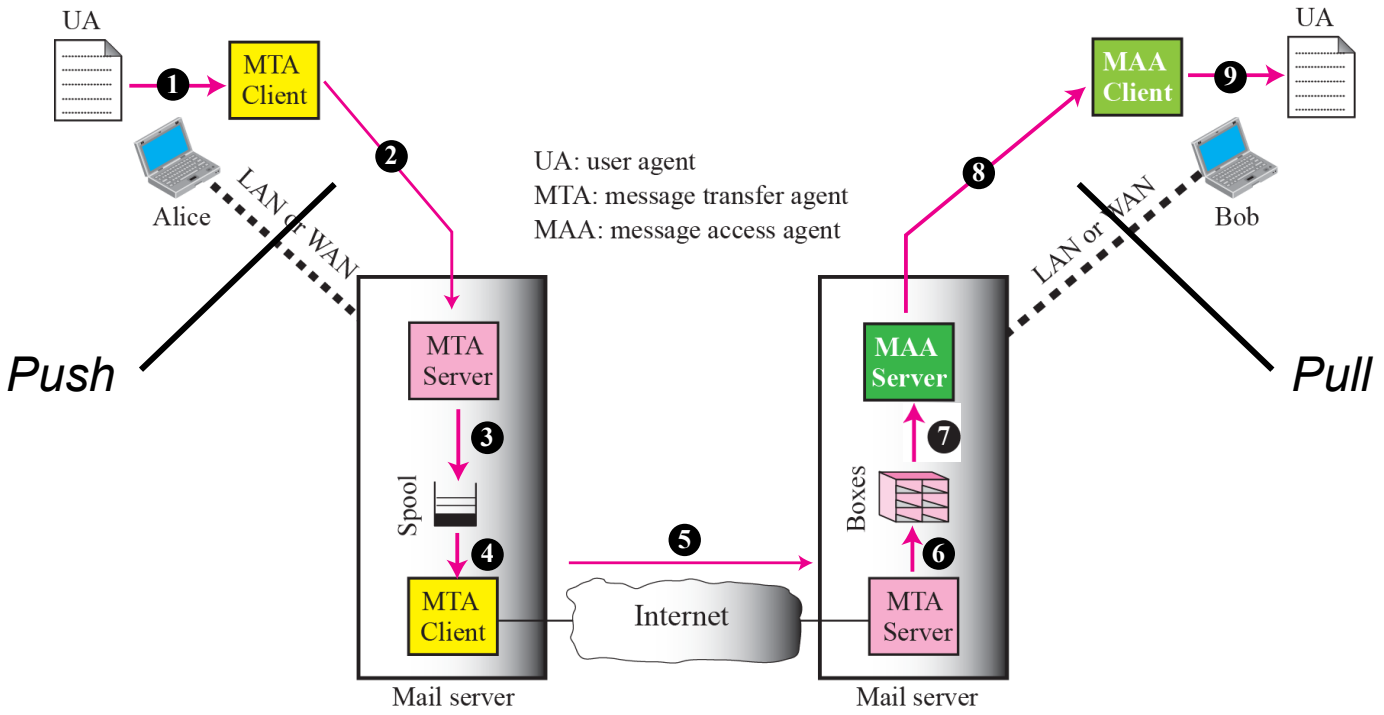
- Transfers message to mail server



Sender and receiver on different mail servers

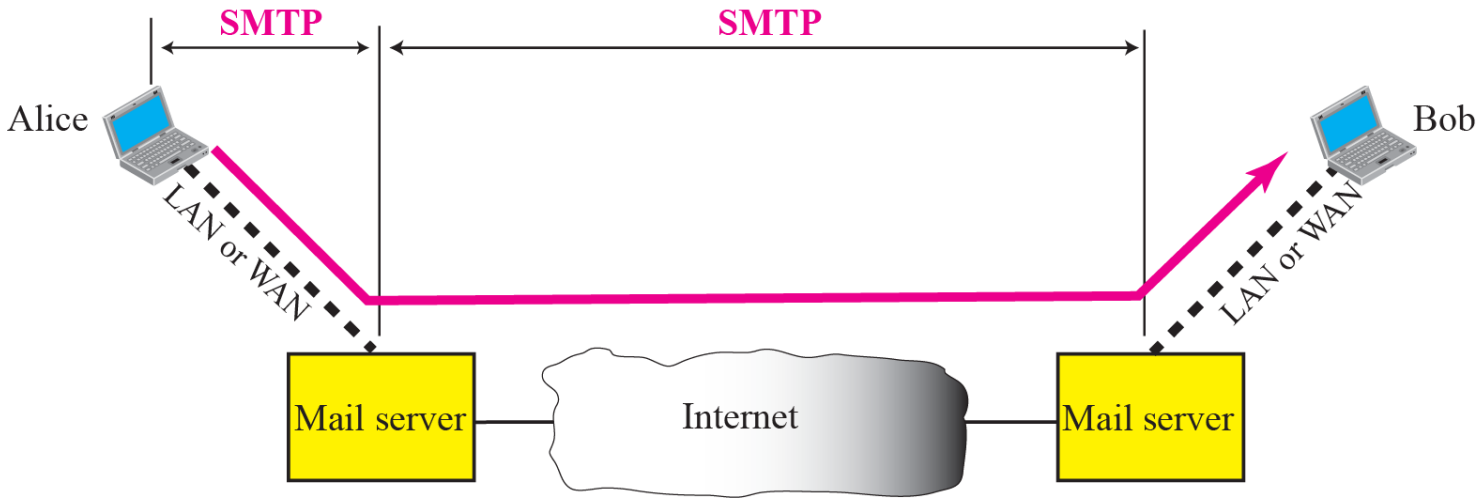
Message Push and Pull

- Sender (Alice) transfers message to *outgoing* mail server
  - MTA
- Receiver (Bob) accesses mail on *incoming* mail server
  - MAA



Sender and receiver separate from mail servers (network between)

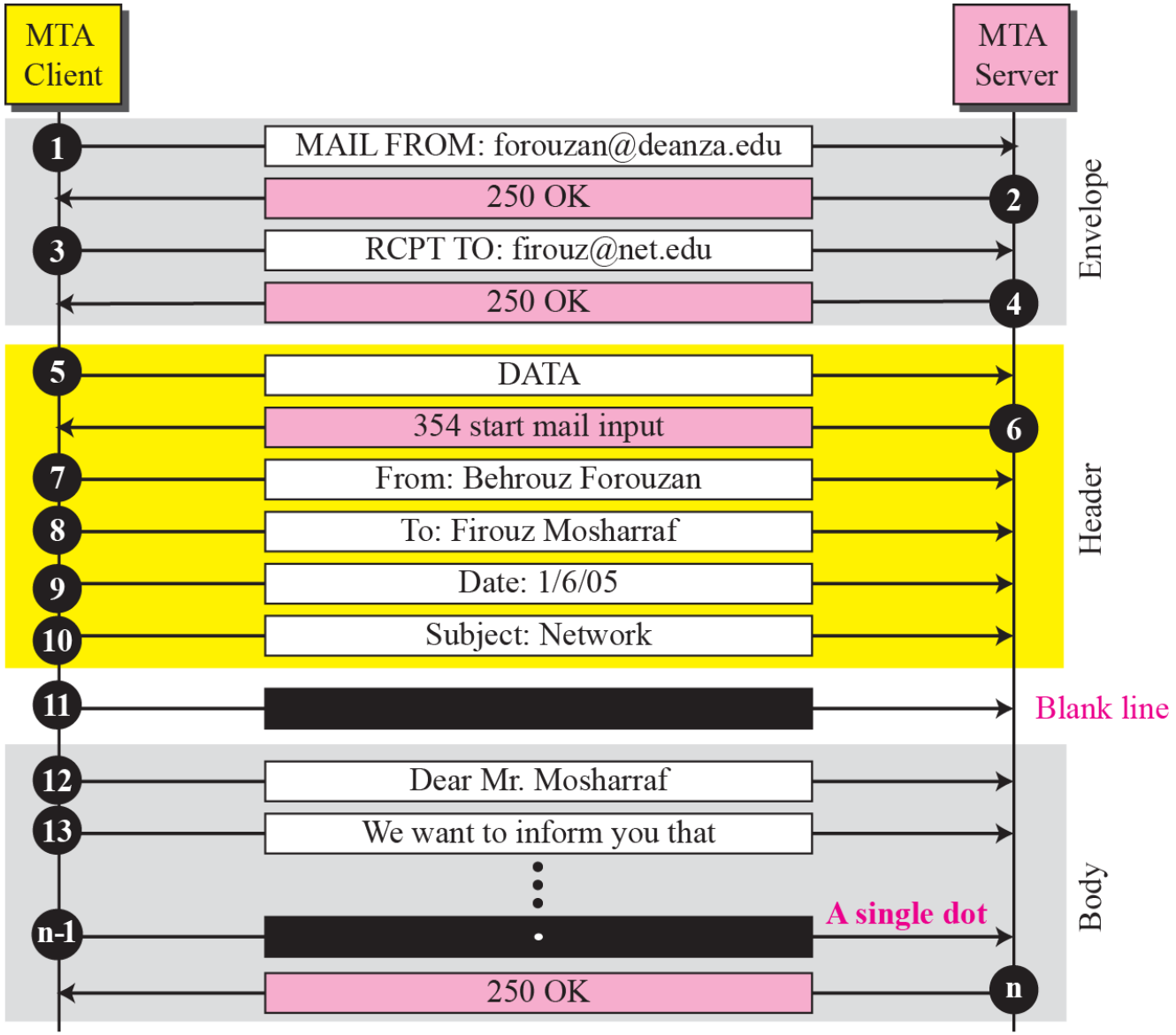
Simple Mail Transfer Protocol (SMTP)



- Uses TCP to reliably transfer email message from client to server
  - port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - Closure

- Command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- Messages must be in 7-bit ASCII

Message Transfer



Try SMTP Interaction for Yourself

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

SMTP Remarks

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses line with single period (`CRLF.CRLF`) to determine end of message

comparison with HTTP:

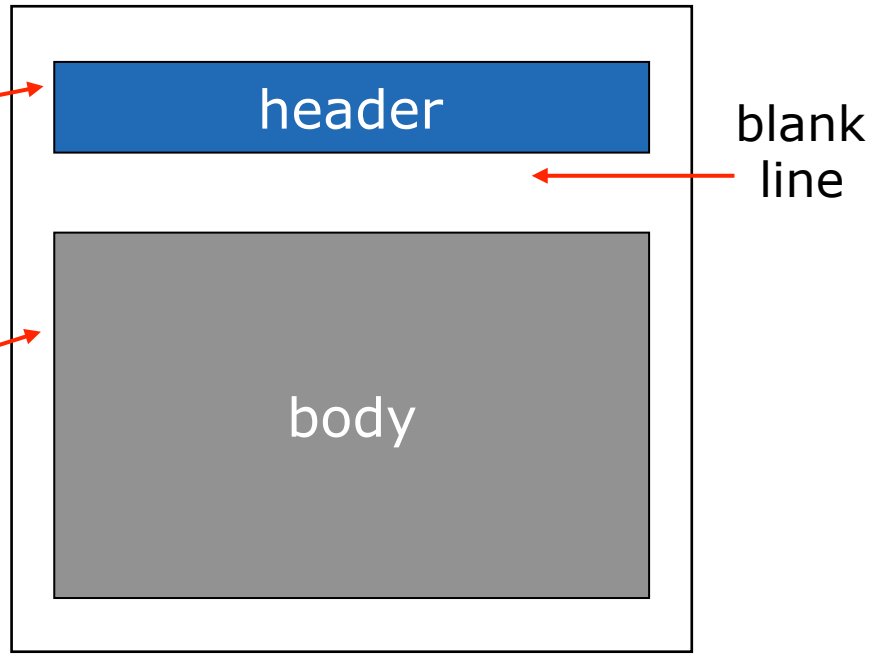
- HTTP: pull
- SMTP: push
- both have ASCII command/ response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message

Mail Message Format

RFC 5322 (RFC 822): standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



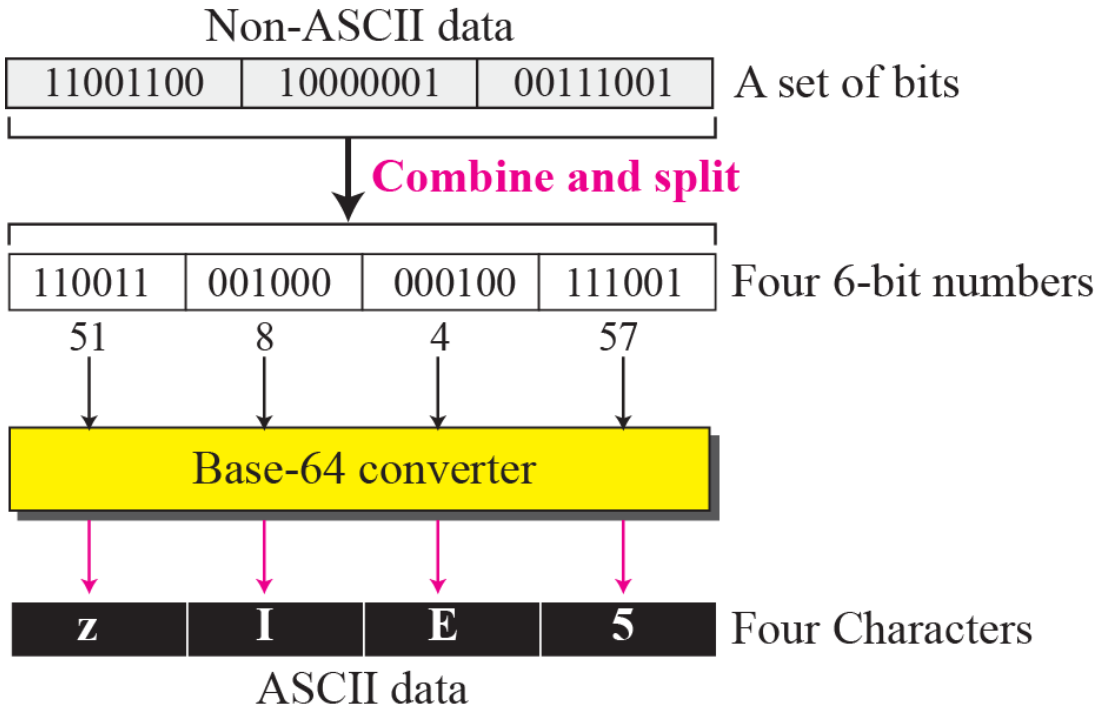
MIME

- Multipurpose Internet Mail Extensions
  - RFC 2045 and more
- Content formats and encodings for SMTP (7-bit ASCII)
  - Binary (non-text) objects (binary files)
  - Non-ASCII text (“Å”, “Ä”, “Ö” for instance)
  - Multi-part message bodies
- Extensions for secure email – S/MIME, PGP, …

```
MIME-version: 1.0
Content-type: multipart/mixed; boundary="frontier"

This is a multi-part message in MIME format.
--frontier
Content-type: text/plain

This is the body of the message.
--frontier
Content-type: application/octet-stream
Content-transfer-encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--frontier--
```

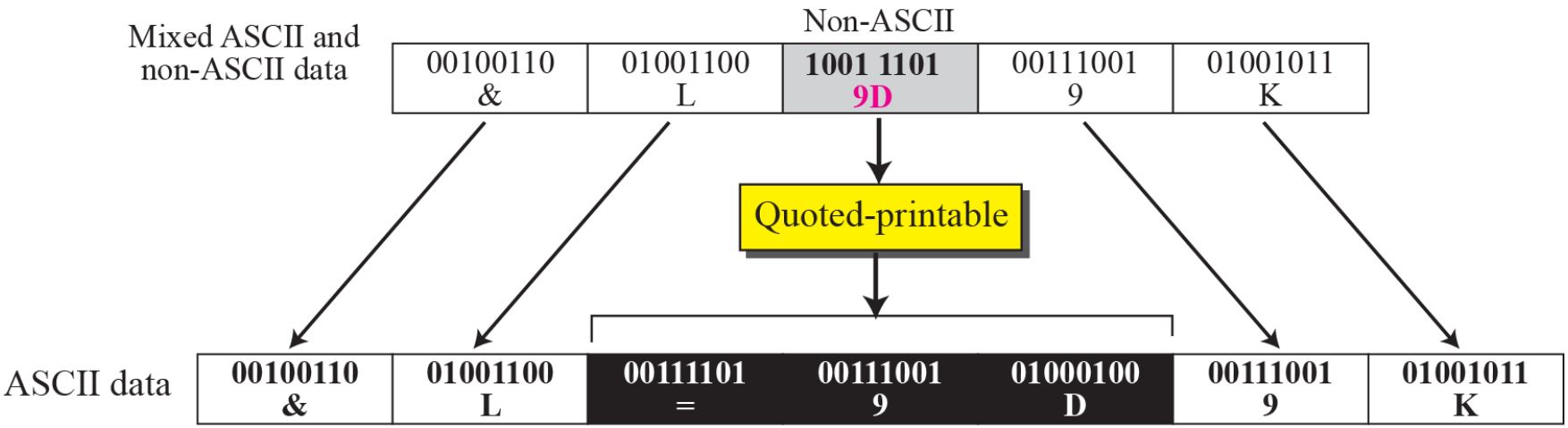
From Wikipedia

Base64 Content Encoding



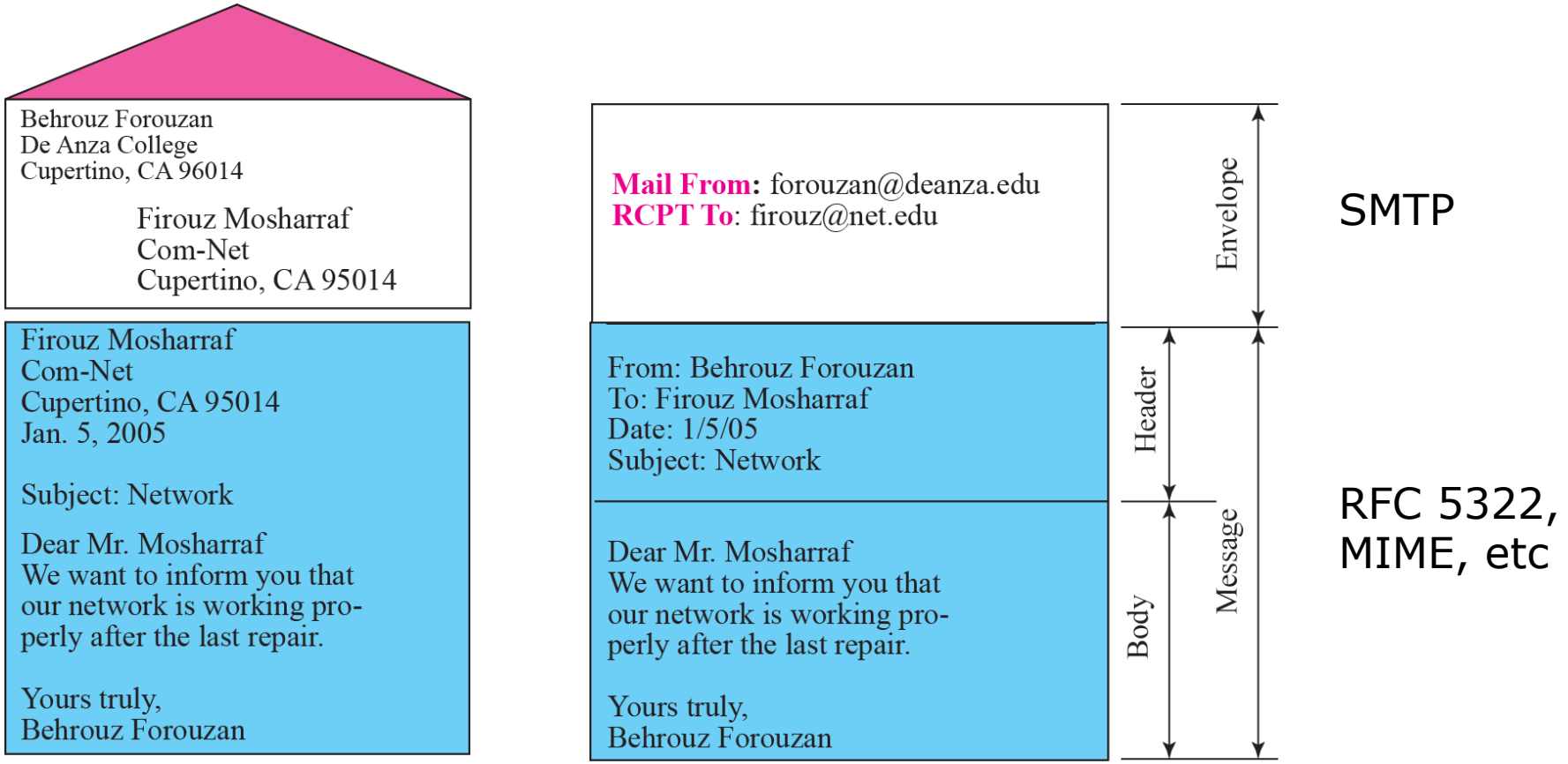
- Three 8-bit words becomes four (7-bit) ASCII characters
- Intended for binary data

Quoted-Printable Content Encoding

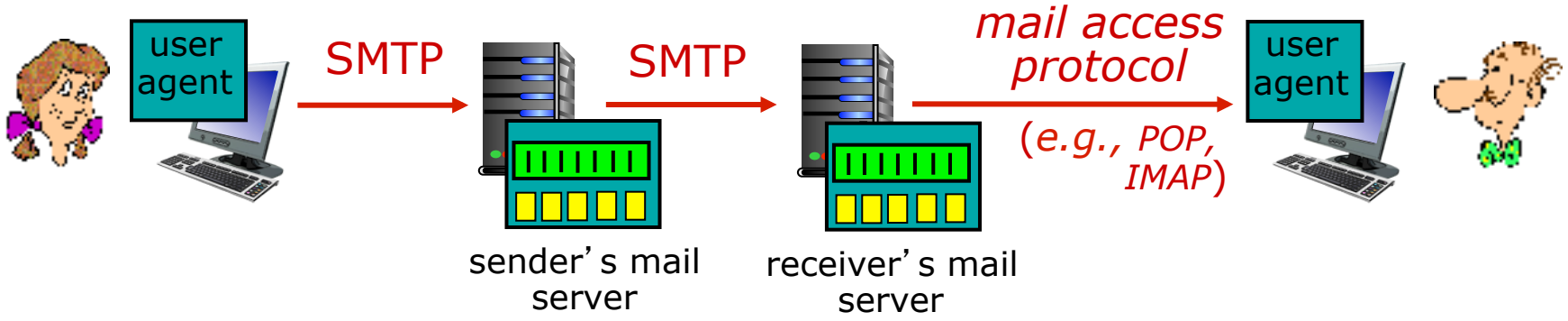


- Non-ASCII 8-bit data "qouted"
  - Translated to sequence of 7-bit ASCII data
  - Equals sign "=" is escape character
- Intended for text interspersed with non-ASCII
  - Such as Swedish…

Format of an Email



Mail Access Protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3

authorization phase

- client commands:
  - user: declare username
  - pass: password
- server responses
  - +OK
  - -ERR

transaction phase, client:

- list: list message numbers
- retr: retrieve message by number
- dele: delete
- quit

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

- Previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- Keeps all messages in one place: at server
- Allows user to organize messages in folders
- Keeps user state across sessions:
  - Names of folders and mappings between message IDs and folder name

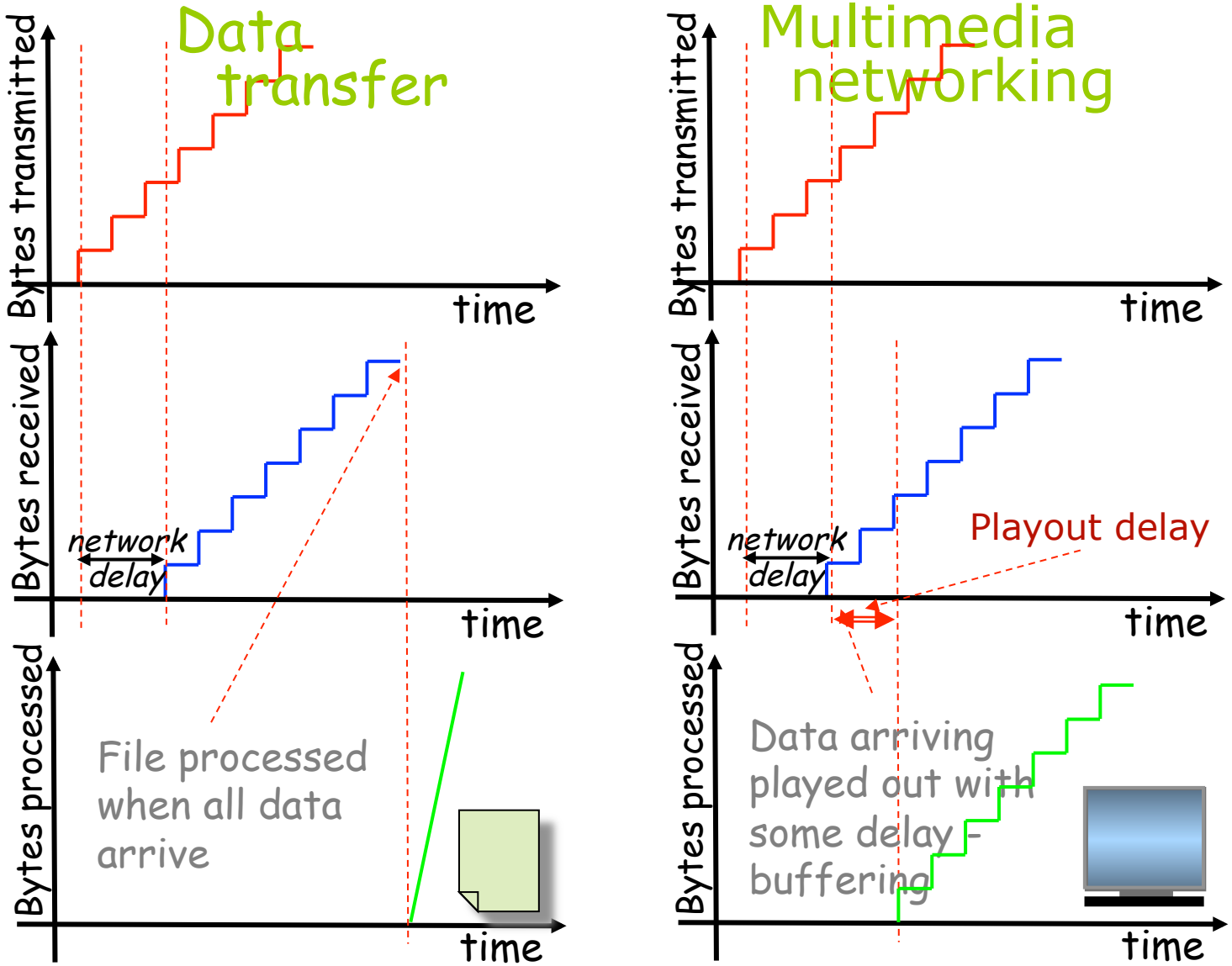
Outline

- Introduction to application layer
  - Principles
  - Client-server
  - Peer-to-peer
- Creating network applications
  - Socket programming API
- Learning by examples
  - Structure of application-layer protocols
- Web
  - Hypertext Transfer Protocol (HTTP)
  - Web documents
  - Cookies
- Remote login
  - Telnet and SSH
- Email
  - SMTP
  - POP and IMAP
  - Email message format,
    - RFC-822, MIME
- Multimedia networking
  - Streaming and real-time media
  - RTP

Multimedia Networking

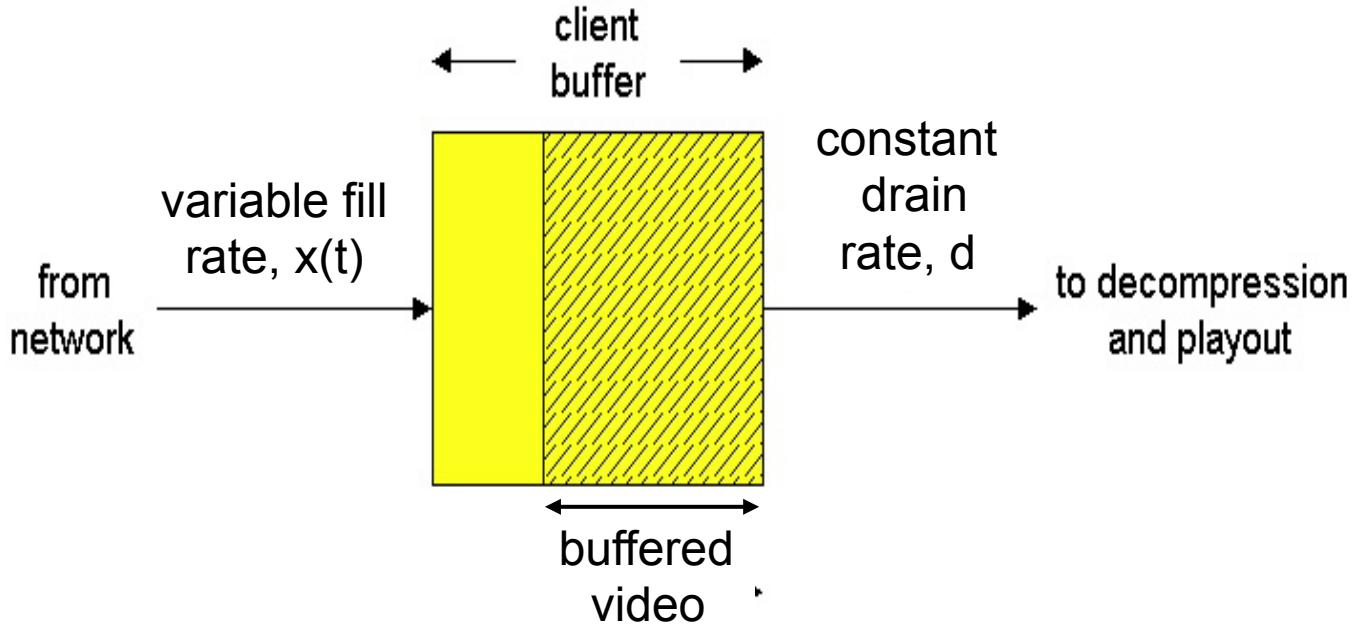
- Multimedia
  - Integration of multiple forms of media
  - Can refer to audio or video only
- Transmission can be
  - Human to human
  - Human to machine
  - Machine to machine
- Continuous consumption
  - Timely delivery
  - Limited loss acceptable

Multimedia Networking vs. Data Transfer



Jitter and Playout Buffer

- Media play out at fixed rate
- Jitter – Internet end-to-end delay varies over time
  - Jitter -> late delivery -> appears as loss



- Client-side buffering to delay the playout – Playout buffer
  - Compensates for delay jitter
  - Playout delay can be dynamically adapted

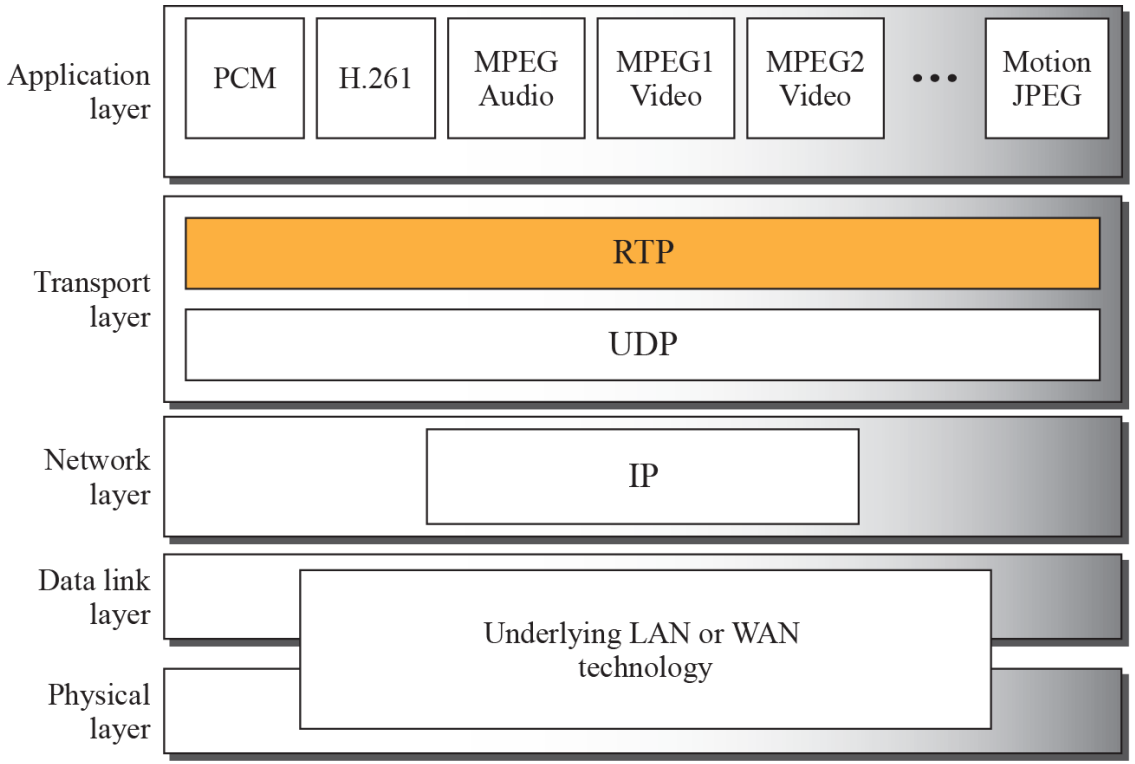
Multimedia Networking Applications

- Interactive applications
  - Strict delay constraints (e.g., telephony, videoconferencing)
  - Session management, locating users, option negotiation
    - H.323 (ITU) and SIP (IETF)
- Streaming applications
  - Limited delay-sensitivity (e.g., Video-on-demand, live streaming)
  - Interactivity creates strict delay constraints
    - VCR like functionality: FF, Rewind, Pause, Seek
    - Acceptable response times and initial delay

TCP or UDP?

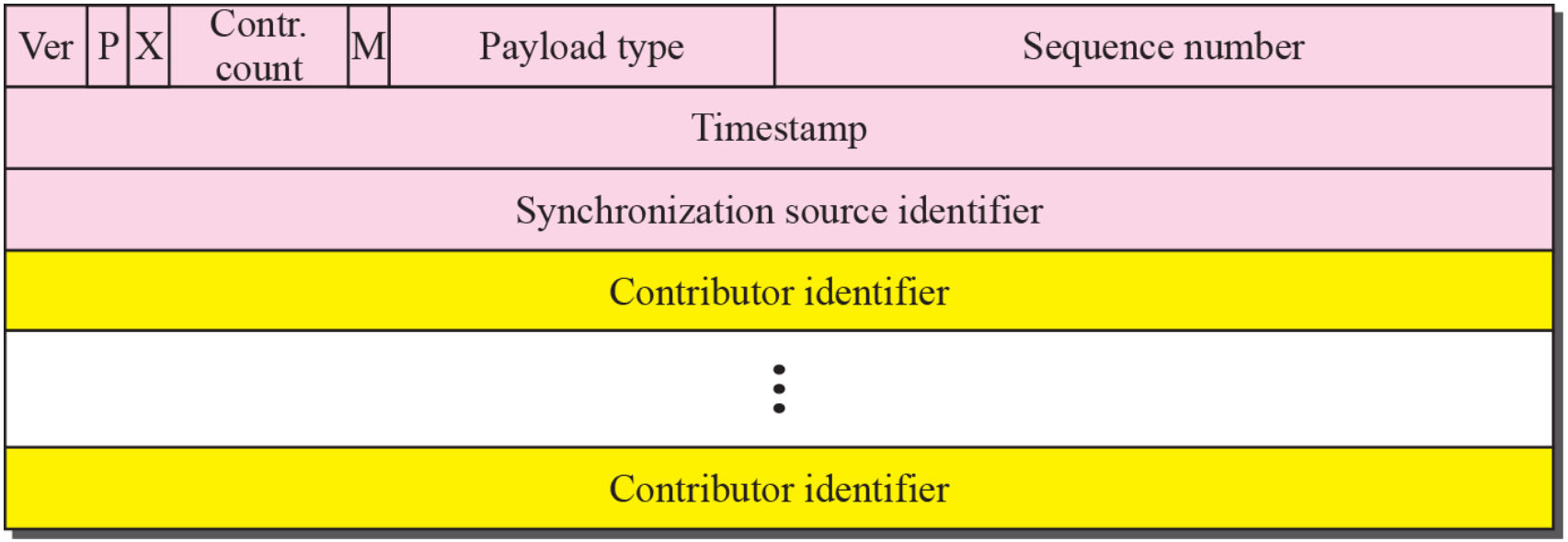
- Arguments for UDP
  - Multimedia can tolerate some loss
  - Minimum overhead
  - Short delay
    - No retransmissions
    - No congestion control, slow start, etc.
- Arguments for TCP
  - Multimedia can tolerate some delay
  - Congestion control
    - Fairness towards other TCP applications
    - Helps to avoid overloading the network
  - Well integrated with HTTP
    - Multimedia on the web
  - Passes through firewalls

Real-time Transport Protocol



- RTP, RFC 3550 and more
- Mechanisms to synchronize and order data
- Runs on top of UDP

RTP Message



- *Payload type* – different encoding formats
- *Sequence number* – for receiver to detect out-of-order delivery
- *Timestamp* – allows receiver to schedule playback (relative)
- *Synchronization source* – session coordinator/mixer
- *Contributor* – allows for multiple sources (contributors) per session

RTP

- Often used together with Real-Time Control Protocol
  - RTCP
  - Feedback to sender about channel state
- Examples
  - SIP (Voice over IP, Video conferencing)
  - Digital Video Broadcast (DVB) over IP

Application Layer Summary

- Client-server and peer-to-peer
- Application design
  - Concurrency – client-server
  - Socket API
- Example application layer protocols
  - Web and HTTP
  - Remote login
    - Telnet and SSH
  - Email
    - SMTP, POP, IMAP for message transfer and access
    - RFC-822, MIME for message formats
- Multimedia networking
  - TCP or UDP
  - RTP