

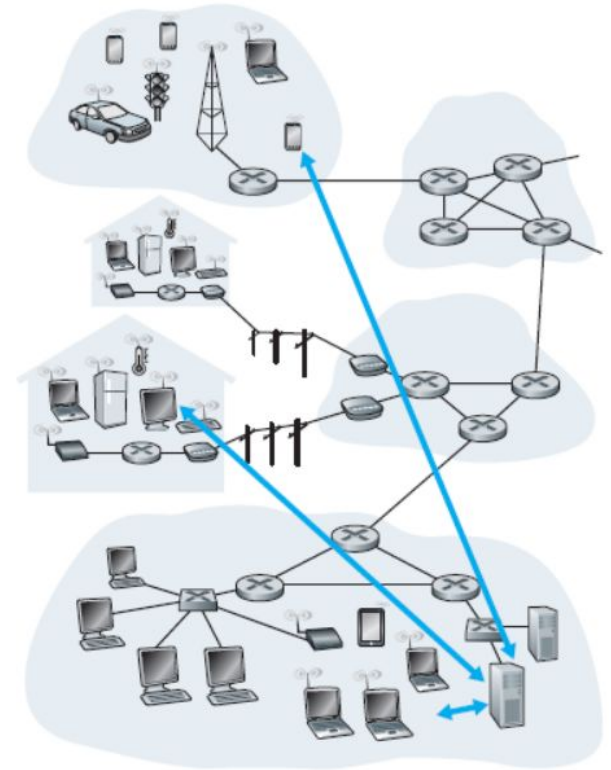
CS-335a: Computer Networking

Tutorial 2: Application Layer

Application Layer Architectures

1. Client-server architecture

- An always-on host (server) services requests from many other hosts (clients)
- Server has a fixed, well-known address (IP address)
- Because the server is always on, a client can always contact the server by sending a packet to the server's IP address
- Clients do not communicate directly with each other
- *E.g. Web, FTP, e-mail*

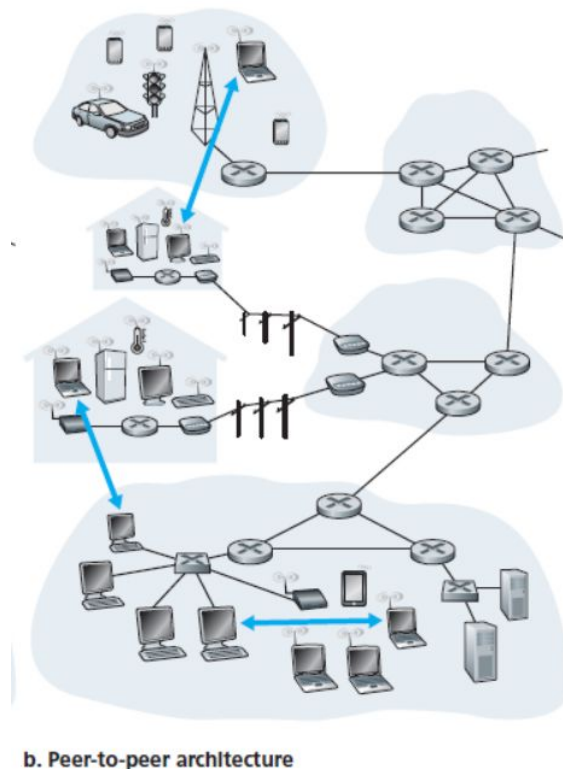


a. Client-server architecture

Application Layer Architectures

2. Peer-2-Peer architecture

- No always-on server
- Arbitrary end systems directly communicate (peers)
- Peers request service from other peers, provide service in return to other peers
- Peers are desktops and laptops controlled by users
- P2P architectures have *self-scalability*
- *E.g. BitTorrent, Skype*



Transport Services Available to Applications

When you develop an application, which of the available transport-layer protocols should you choose ?

Transport Layer Protocols provide:

- i) **Reliable data transfer**
(e.g file transfer, web transactions)
- ii) **Throughput**
(e.g. multimedia applications are bandwidth sensitive)
- iii) **Timing**
(e.g., Internet telephony, interactive games) require low delay to be “effective”
- iv) **Security**
(e.g. encryption of data, end-point authentication)

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Transport Services Provided by the Internet

TCP:

- *reliable transport* between sending and receiving process
- *flow control*: sender won't overwhelm receiver
- *congestion control*: throttle sender when network overloaded
- *connection-oriented*: setup required between client and server processes
- *does not provide*: timing, minimum throughput guarantee, security

UDP:

- *unreliable data transfer* between sending and receiving process
- *does not provide*: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Application Layer Protocol

An application-layer protocol defines how an application's processes, running on different end systems, pass messages to each other.

In particular, an application-layer protocol defines:

1. The **types of messages** exchanged, for example, request messages and response messages
2. The **syntax** of the various message types, such as the fields in the message and how the fields are delineated
3. The **semantics** of the fields, that is, the meaning of the information in the fields
4. **Rules** for determining when and how a process sends messages and responds to messages

HTTP (HyperText Transfer Protocol): Overview

HTTP is implemented in two programs (**client-server** architecture used by the Web):

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

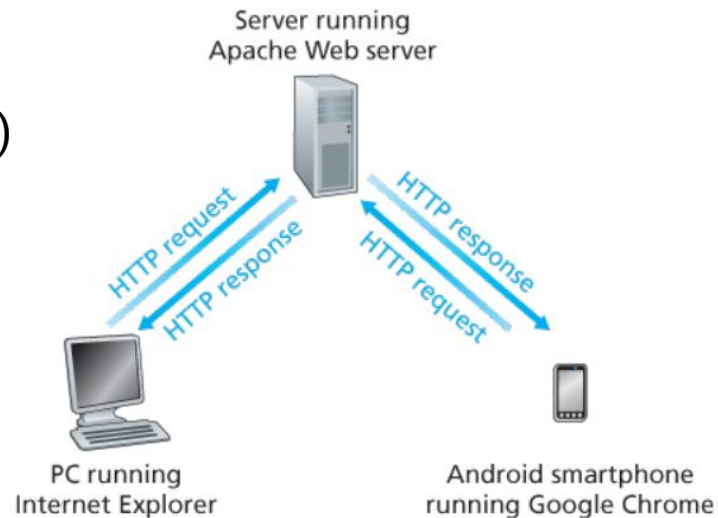
Web objects:

- a web page consists of *objects*, each of which can be stored on different Web servers
- an object can be: HTML file, JPEG image, Java applet, audio file etc.
- web page consists of *base HTML-file* which includes *several referenced objects* (addressable by a *URL*)

HTTP: Overview

HTTP uses **TCP**:

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



HTTP is **stateless**:

- If a client asks for the same object twice, the server re-sends the object, as it has completely forgotten what it did earlier

HTTP: Persistent vs Non-Persistent Connections

Non-persistent HTTP: each request/response pair is sent over a **separate** TCP connection.

Persistent HTTP: each request/response pair is sent over the **same** TCP connection.

- By default HTTP uses *persistent* connections (with *pipelining*), but clients and servers can be configured to use non-persistent connections.

Non-Persistent HTTP Example

```
http://www.someschool.edu/someDepartment/home.index
```

Suppose this webpage consists of:

- 1 base HTML file
- 5 references to JPEG images
- 5 references to other HTML files

→ Total: 11 objects

Non-Persistent HTTP Example

1. Client process initiates a TCP connection to the server `www.someSchool.edu` (port 80).
2. Client sends an HTTP request message to the server via its socket. The request message includes the path name `/someDepartment/home.index`.
3. Server process receives the request message via its socket, retrieves the object `/someDepartment/home.index`, encapsulates the object in an HTTP response message, and sends the response message to the client via its socket.
4. Server process tells TCP to close the TCP connection.
5. The HTTP client receives the response message. The TCP connection terminates.
The client extracts the file from the response message, examines the HTML file, and finds references to the 5 JPEG and 5 HTML objects.
6. Steps 1-4 are then repeated for each of the 10 referenced objects.

➔ **11 TCP connections in total !**

Round-Trip Time (RTT)

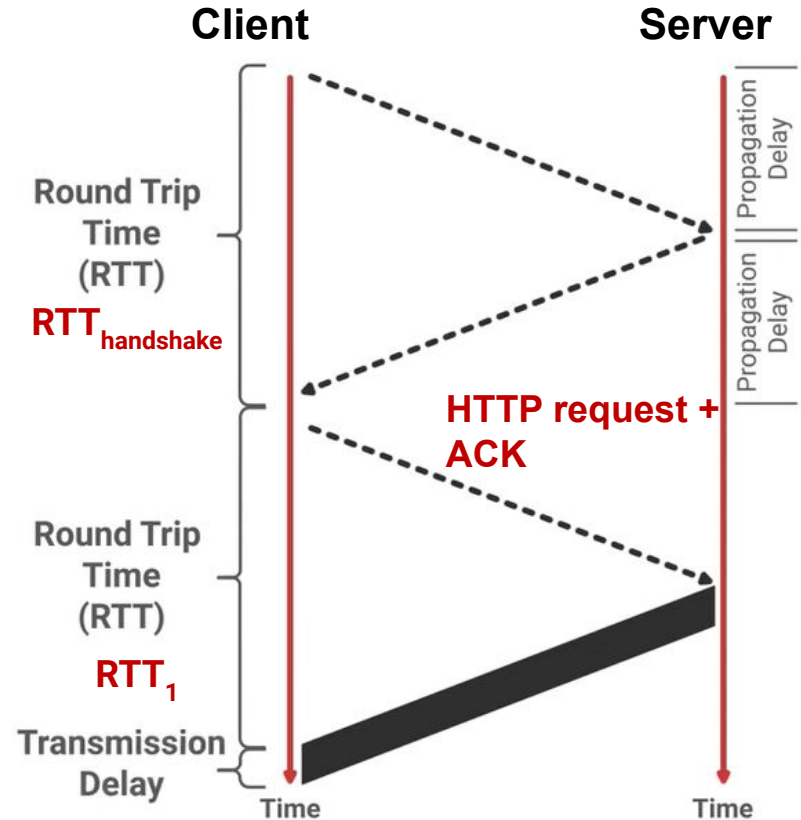
RTT: the time it takes for a small packet to travel from client to server and then back to the client.

RTT includes packet-propagation delays, packet queuing delays in intermediate routers and switches, and packet-processing delays

Total response time =

$RTT_{\text{handshake}} + RTT_1 + \text{Transmission Delay}$

→ What is the total response time in the previous example ?



Persistent HTTP - Pipelining

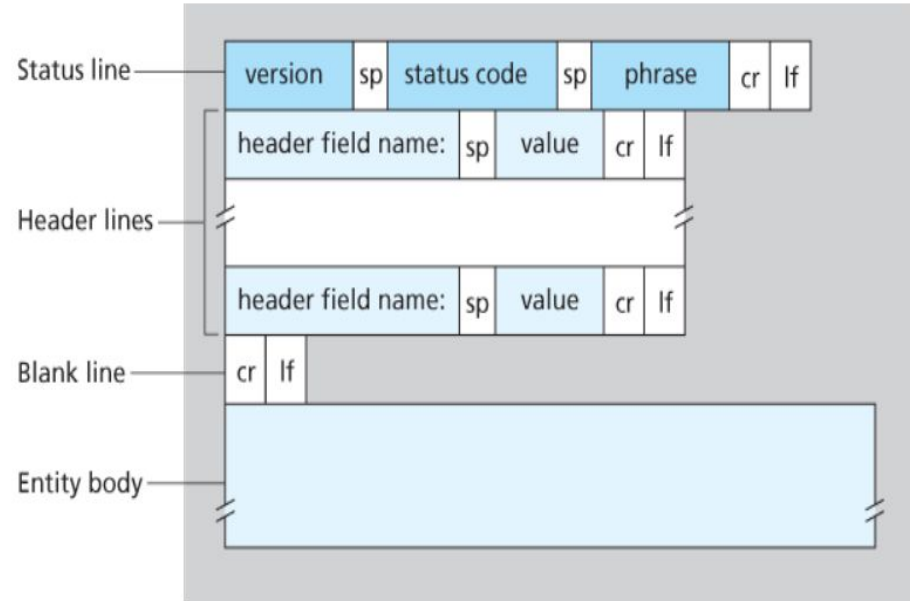
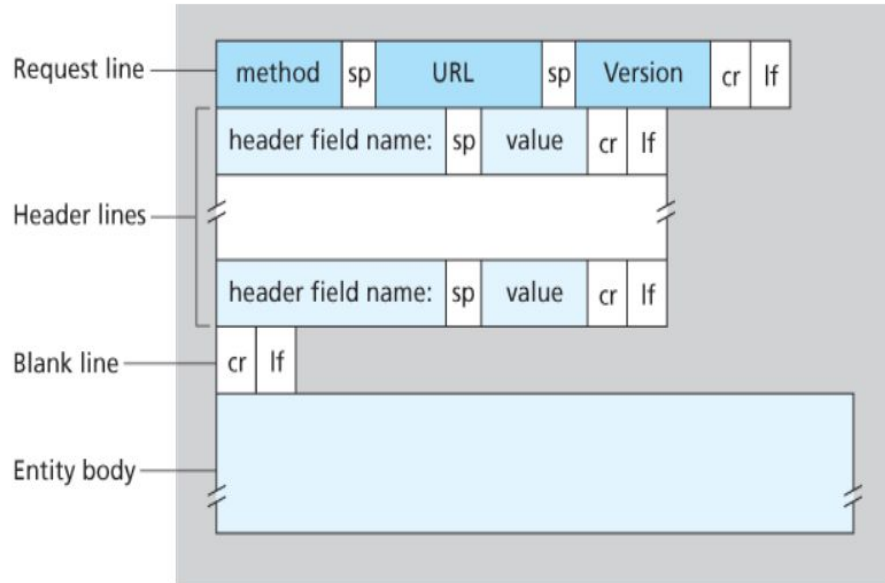
- The server leaves the TCP connection open after sending a response.
- Subsequent requests and responses between the same client and server can be sent over the same TCP connection.
- These requests can be made
 - i. Sequentially

$$\text{Total transmission time} = \text{RTT}_{\text{handshake}} + (\text{RTT} + \text{Transmission Delay}) * N_{\text{objects}}$$

- ii. back-to-back, without waiting for replies from the server (**pipelining**)

$$\text{Total transmission time} = \text{RTT}_{\text{handshake}} + \text{RTT} + \text{Transmission Delay}$$

HTTP Message Format



HTTP Request

```
GET /somedir/page.html HTTP/1.1<cr><lf>Host:  
www.someschool.edu<cr><lf> Connection: close<cr><lf>User-agent:  
Mozilla/5.0<cr><lf>Accept-language: fr <cr><lf><cr><lf>
```



```
GET /somedir/page.html HTTP/1.1  
Host: www.someschool.edu  
Connection: close  
User-agent: Mozilla/5.0  
Accept-language: fr  
<blank line>
```

HTTP Request

method URL HTTP version

↓ ↓ ↓

where to find object → GET /somedir/page.html HTTP/1.1 request line

Non-persistent connection ! → Host: www.someschool.edu

Connection: close

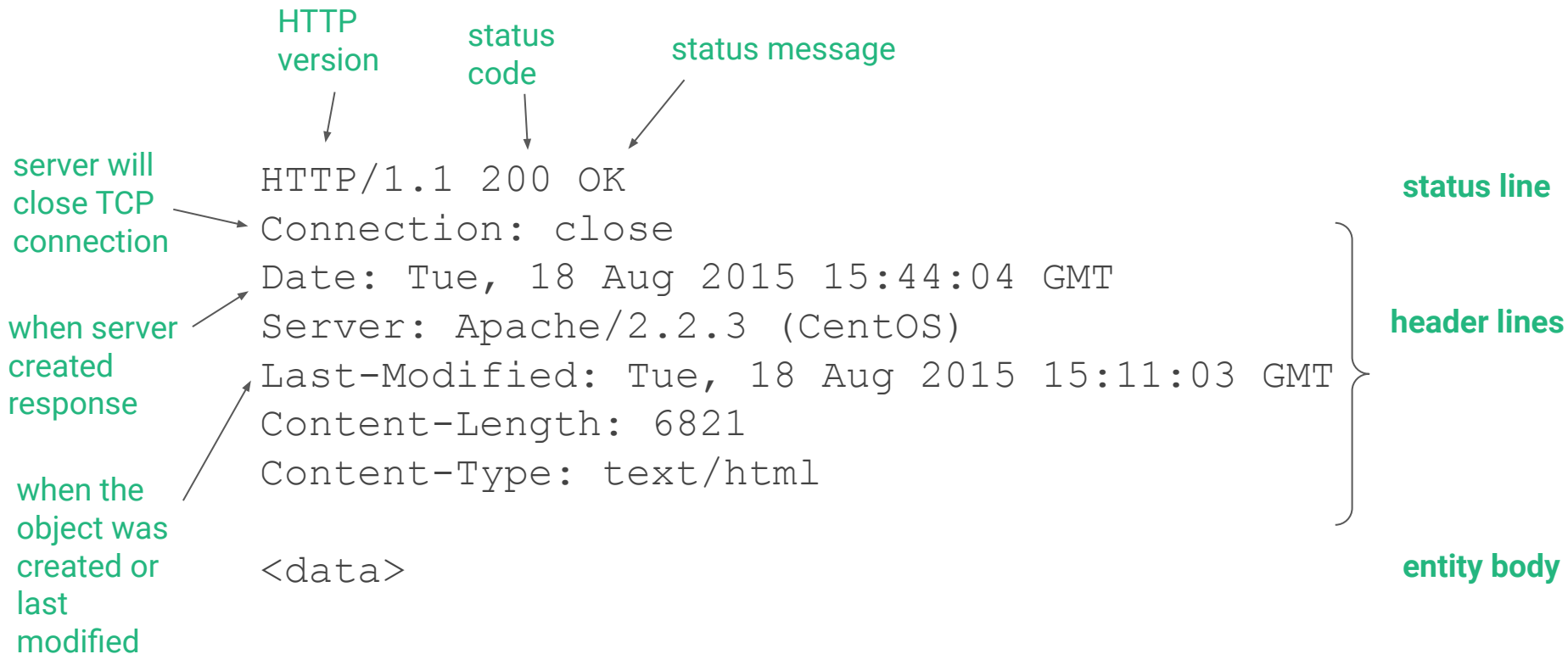
User-agent: Mozilla/5.0

Accept-language: fr

<blank line>

} header lines

HTTP Response



Web Caching

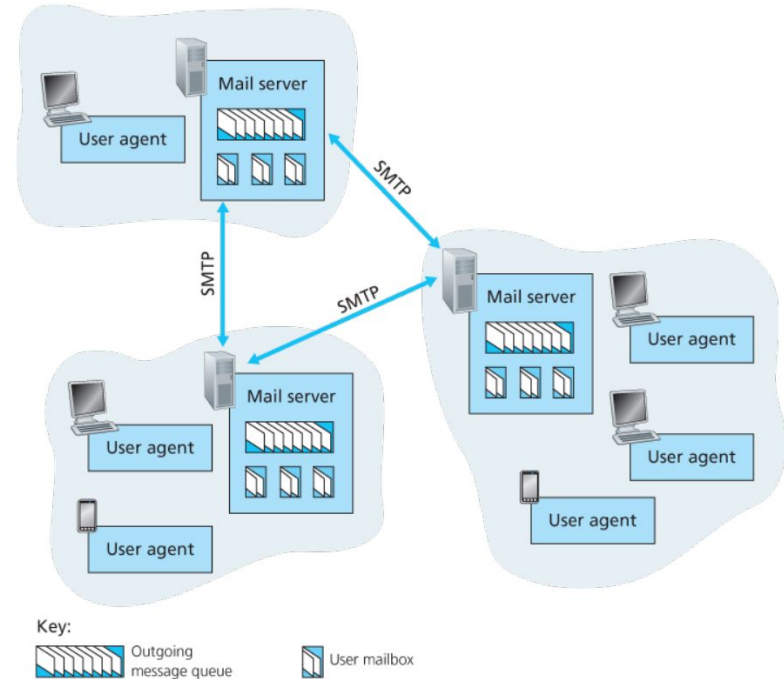
Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) *Web cache*
- **Web cache (or proxy server):** network entity that satisfies HTTP requests on the behalf of an origin Web server. The Web cache has its own disk storage and keeps copies of recently requested objects in this storage
- browser sends all HTTP requests to cache
 - *if* object in cache: cache returns object to client
 - *else* cache requests object from origin server, caches received object, then returns object to client
- A Web cache acts as both *server* and *client*

Electronic Mail in the Internet: Overview

Three major components:

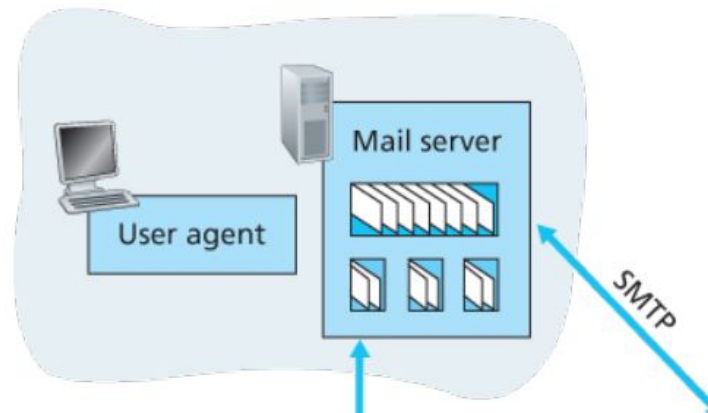
- **user agents:** allow users to read, reply to, forward, save, and compose messages (e.g. Microsoft Outlook)
- **mail servers:** where message are placed
- **Simple Mail Transfer Protocol (SMTP):** transfers messages from senders' mail servers to the recipients' mail servers



E-mail

How will Alice send an e-mail to Bob ?

1. Alice composes her email
2. Her user agent sends the message to her mail server
3. The message is placed in the mail server's outgoing message queue
4. Bob's user agent retrieves the message from his mailbox in his mail server



A user's **mail server** contains:

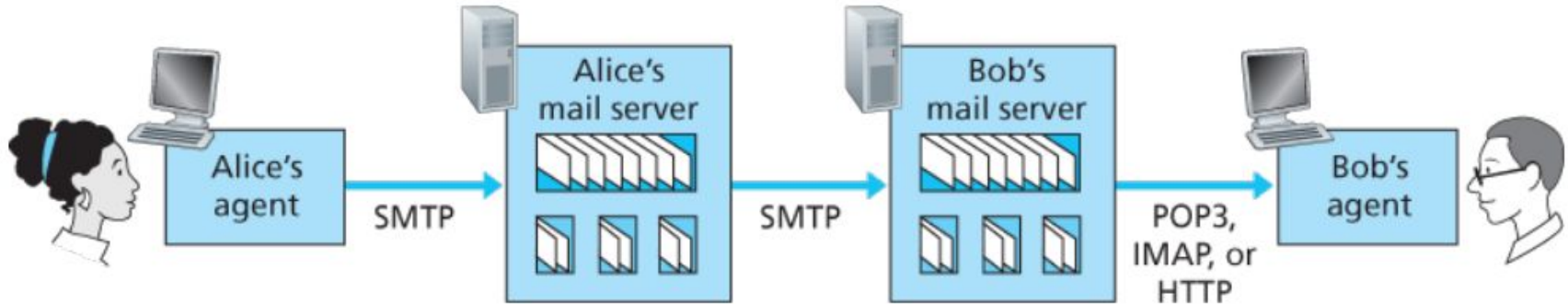
- **Mailbox**: maintains the messages that have been sent to them
- **Message queue**: holds outgoing messages (until they are sent to the recipient)

SMTP

- Principal application-layer protocol for Internet electronic mail
- Uses **TCP** (reliable data transfer)
- Uses **persistent connections**
- Has two sides:
 - i) **client side**: executes on the sender's mail server
 - ii) **server side**: executes on the recipient's mail server
- Both the client and server sides of SMTP run on every mail server

→ SMTP transfers messages from senders' mail servers to the recipients' mail servers

Mail Access Protocols



- Bob's user agent can't use SMTP to obtain the messages because obtaining the messages is a pull operation, whereas **SMTP is a push protocol**
- Instead, Bob can use email access protocols such as **POP3, IMAP or HTTP**

Web-based E-mail

- **User agent** is an ordinary **Web browser**
- User communicates with its remote **mailbox** via **HTTP**
- When a recipient wants to access a message in their mailbox, the e-mail message is sent from the **recipients mail server** to the **recipients browser** using **HTTP** rather than POP3 or IMAP
- When a sender wants to send an e-mail message, the e-mail message is sent from the **senders browser** to the **senders mail server** over **HTTP** rather than over SMTP
- The **senders mail server** still **sends** messages to, and **receives** messages from, other mail servers using **SMTP**.

FTP (File Transfer Protocol)

- Application layer protocol that uses TCP
- Used to transfer files between a local and a remote host
- To transfer a file, FTP uses two parallel connections (channels):
 - i) **Control channel:** controls the conversation between the two hosts
(e.g. user identification, password, commands to change the remote directory, commands to retrieve and store files, etc.)
 - ii) **Data channel:** transmits the file content

FTP Session

- **Client** initiates a **control** TCP connection with the server side and sends control information
- When the **server** receives this, it initiates a **data** connection to the client side (only one file can be sent over one data connection)
- Control connection remains active throughout the user session