# Outline

- possible structure of applications（2种）： client-server & P2P
- Client-server architecture
- P2P architecture
- Processes
- Sockets
- Addressing processes
- open protocols
- proprietary protocols
- What transport service does an app need & common apps
- Internet transport protocols services TCP/UDP

- HTTP overview
- non-persistent HTTP persistent HTTP
- HTTP request message
- HTTP response status codes
- cookies
- caches
- Electronic mail
- mail servers
- SMTP
- Mail access protocols
- POP3 (more) and IMAP
- DNS: domain name system
- DNS: services, structure
- top-level domain (TLD) servers
- DNS name resolution

# Application architectures

**client-server**

服务器（Server）是始终开启的主机，具有固定的IP地址，使用数据中心进行扩展，用于提供服务和资源。客户端（Client）与服务器进行通信，可能断断续续地连接，具有动态IP地址，并且客户端之间不直接通信

**server:**

- always-on host
- permanent IP address
- data centers for scaling

**clients:**

- communicate with server
- may be intermittent (断断续续connected)
- may have dynamic IP addresses
- do not communicate directly with each other

**peer-to-peer(P2P)[**种子，文件下载**]**

在这种情况下，没有始终开启的服务器。任意终端系统可以直接相互通信，对等节点可以向其他对等节点请求服务，并提供服务作为回报。系统具有**自我可扩展性**，新的对等节点带来新的服务能力和服务需求。对等节点的连接是间断的，并且IP地址会发生变化。管理方面更为复杂。

- no always-on server
- arbitrary end systems directly communicate 任意端系统直接通信
- 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

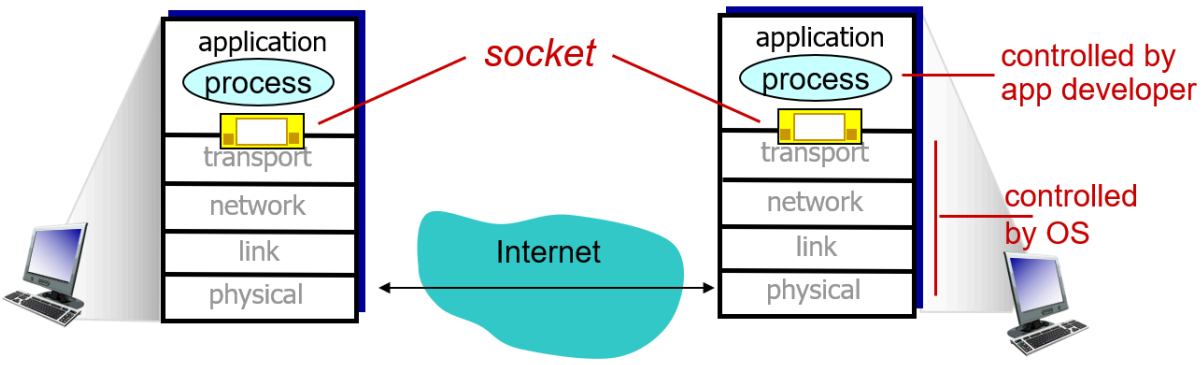
# Processes communicating

- **Process**进程: program running within a host
  - 在同一主机内，两个进程可以使用操作系统定义的进程间通信方式进行通信
  - 而在不同主机上的进程之间通信，则需要通过交换消息进行
  - 在具有对等（P2P）架构的应用中，通常会同时存在客户端进程和服务器进程
- many processes can be running on same host
- example port numbers:（well known）
  - HTTP：80
  - mail：25

# Sockets

套接字是进程与网络通信之间的接口，用于发送和接收消息

套接字作为进程与网络之间的通道，用于消息的进出 - API



# App-layer protocol defines

- **open protocols:**
  - defined in RFCs
  - allows for interoperability
  - e.g., HTTP, SMTP（mail）
- **proprietary protocols：**
  - Skype，FB，ZOOM，MS teams

# What transport service does an app need

| application | data loss | throughput | time sensitive |
|---|---|---|---|
| file transfer | no loss | Elastic(弹性) | no |
| e-mail | no loss | elastic | no |
| Web documents | no loss | elastic | no |
| real-time audio/video | loss-tolerant | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | yes, 100’s msec |
| stored audio/video | loss-tolerant | same as above | |
| interactive games | loss-tolerant | few kbps up | yes, few secs |
| text messaging | no loss | elastic | yes, 100’s msec<br>yes and no |

# Internet transport protocols services

**TCP service**（不保证时间，延迟远大于**UDP**）

TCP连接是全双工的

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

**UDP service**（不保证时间和服务质量）

无连接的，两个进程通信前没有握手

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

UDP存在的原因主要有以下几点：

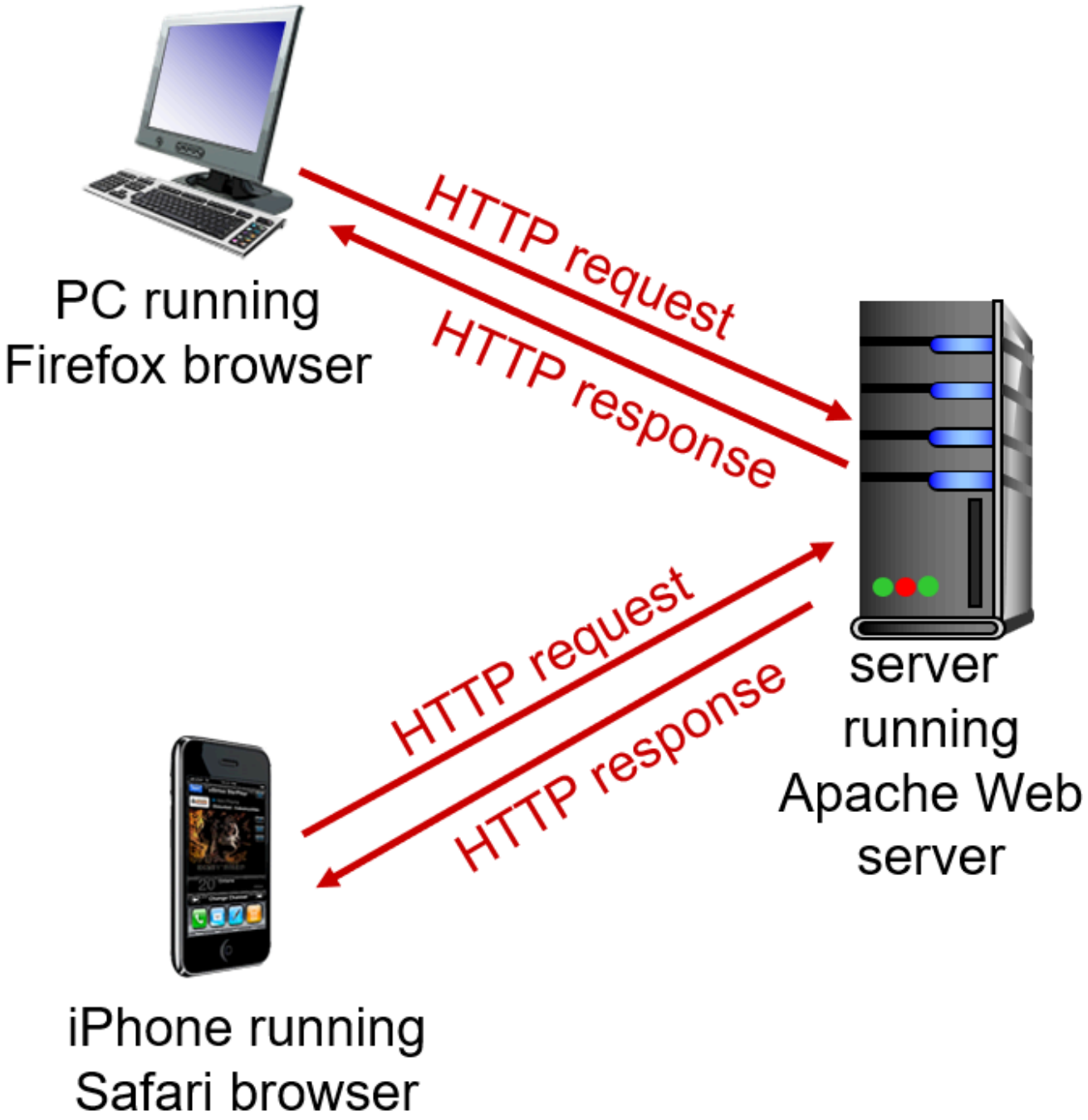
1. **速度和效率**：UDP比TCP更轻量，不需要复杂的握手和确认过程，因此通常更快。这使得它适合需要快速数据传输的应用，如实时音频和视频流。
2. **应用场景**：某些应用程序更关注数据的及时性而不是完整性。例如，语音通话或在线游戏可能更关心降低延迟，而不是确保每个数据包都到达。
3. **控制权**：UDP允许应用程序更多的控制权，可以自己处理数据的完整性和顺序，而不依赖于协议层来管理。
4. **低开销**：由于UDP协议更简单，因此在资源受限的环境中，它可能更具吸引力。

不同类别的app用的协议

| application | application layer protocol | underlying transport protocol |
|---|---|---|
| e-mail | SMTP [RFC 2821] | TCP |
| remote terminal access | Telnet [RFC 854] | TCP |
| Web | HTTP [RFC 2616] | TCP |
| file transfer | FTP [RFC 959] | TCP |
| streaming multimedia | HTTP (e.g., YouTube), RTP [RFC 1889] | TCP or UDP |
| Internet telephony | SIP, RTP, proprietary (e.g., Skype) | TCP or UDP |

# HTTP：hypertext transfer protocol

是一种基于**TCP协议**的应用层协议，它是**Web的应用层协议**。HTTP协议使用客户端/服务器模型，其中客户端是指发出请求、接收并显示Web对象的浏览器，服务器则是指通过HTTP协议响应请求并发送对象的Web服务器。




在使用TCP时，**客户端会通过创建套接字**向服务器的***端口80***发起TCP连接。服务器接受来自客户端的TCP连接。然后，浏览器（作为HTTP客户端）和Web服务器（作为HTTP服务器）之间交换HTTP消息（应用层协议消息）。完成消息交换后，TCP连接关闭。

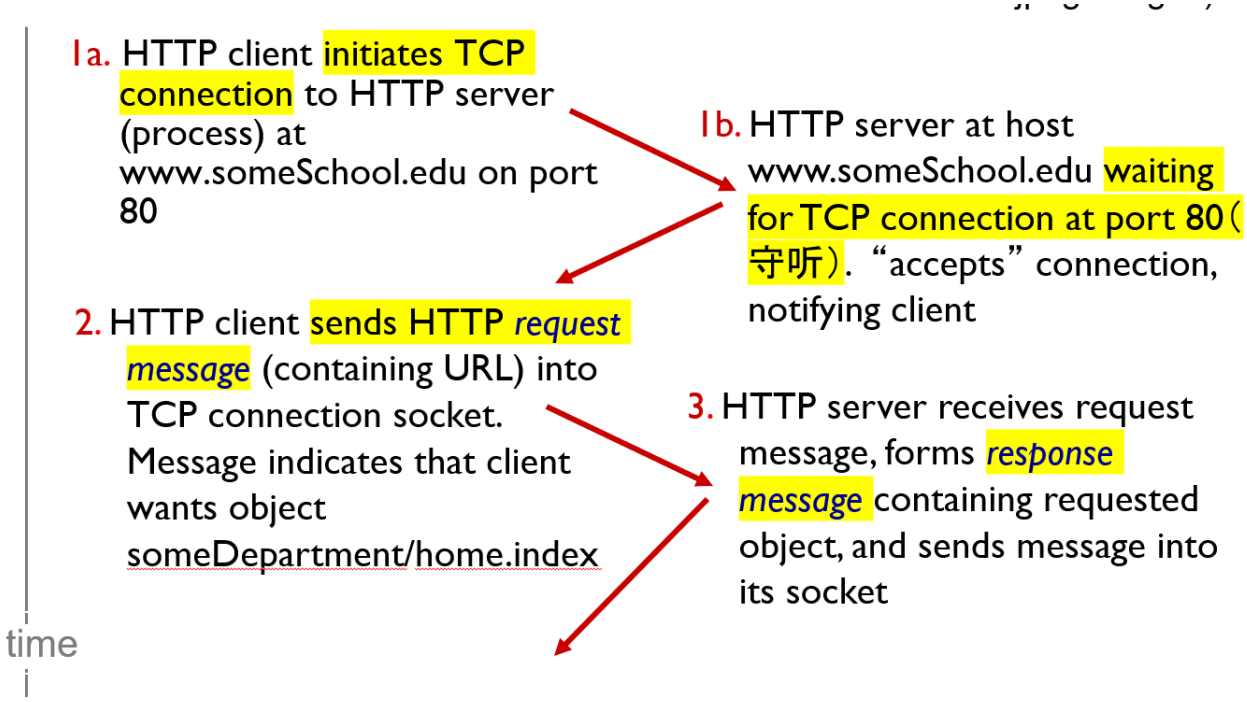
- 早期的HTTP是无状态的

**non-persistent HTTP 和 persistent HTTP**

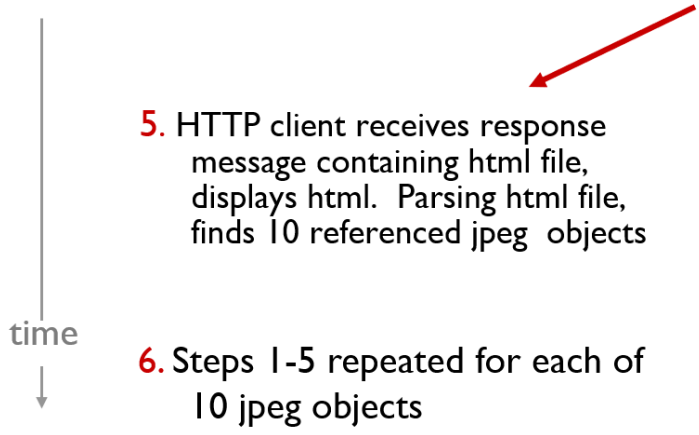
```
1.非持久连接的HTTP (non-persistent HTTP)
- 在非持久连接的HTTP中，每个HTTP请求/响应都建立一个独立的连接。
- 当客户端发送一个HTTP请求到服务器时，客户端和服务器之间建立一个连接，服务器处理请求并发送响应后，连接立即关闭。
- 下次客户端再次发送请求时，需要重新建立一个新的连接。这种连接方式对于每个请求都需要建立连
```

接和释放连接，导致了
较高的连接开销。
2.持久连接的HTTP (persistent HTTP)
- 在持久连接的HTTP中，多个HTTP请求/响应可以共享同一个连接。
- 当客户端发送一个HTTP请求到服务器时，客户端和服务器之间建立连接，服务器处理请求并发送响应后，连接并不立即关闭，而是保持打开的状态。
- 在保持连接打开的状态下，客户端可以发送更多的请求，而无需重新建立连接。
- 服务器可以在一段时间内保持连接打开，以便处理后续的请求从而减少了连接的建立和释放开销。
- 持久连接通过复用连接，提供了更高的效率和较低的延迟

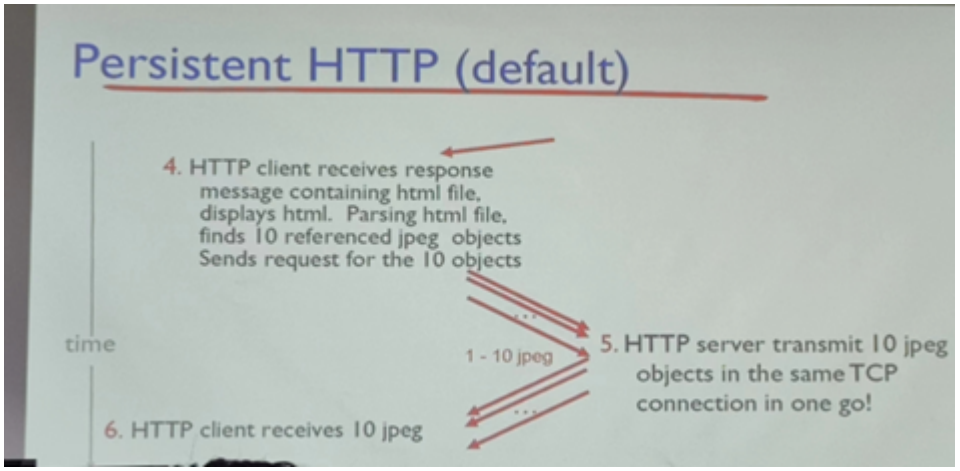
**(Non)persistent HTTP状态图**




NON




PER




**Non-persistent HTTP: response time**

- **RTT (definition)**: time for a small packet to travel from client to server and back

HTTP响应时间是指完成一次HTTP请求并接收到完整HTTP响应所需的时间。

1. 建立TCP连接：客户端发起TCP连接请求到服务器，这需要一个RTT
2. 发送HTTP请求和接收部分响应：客户端发送HTTP请求到服务器，并接收到HTTP响应的前几个字节，这也需要一个RTT。
3. 文件传输时间：在收到部分响应后，服务器开始传输完整的文件内容到客户端。这个时间取决于文件大小和网络带宽。

对于**非持久连接的HTTP**请求，即每次请求都会建立新的TCP连接，HTTP响应时间可以计算为:

$$responsetime = 2RTT + filetransmissiontime$$

非持久**HTTP**的问题

非持久连接的HTTP存在以下问题：

每个对象需要两个RTT：由于每次请求都需要建立新的TCP连接，所以对于每个HTTP对象（如图片、脚本、样式表等），都需要额外的RTT来建立连接。

每个TCP连接的操作系统开销：每次建立和关闭TCP连接都会带来操作系统的开销，包括资源分配、连接管理等。

浏览器常常会打开并行的TCP连接来获取引用的对象：为了提高性能，浏览器通常会同时打开多个TCP连接，以便并行地获取引用的对象。这样做可以减少等待时间，但也带来了额外的开销和复杂性。

持久连接的HTTP解决了上述问题：

服务器在发送响应后保持连接开放：服务器在发送完HTTP响应后保持连接开放，并且可以在同一连接上进行后续的HTTP消息传递。

同一客户端/服务器之间的后续HTTP消息在打开的连接上发送：客户端可以立即发送请求，而不需要等待

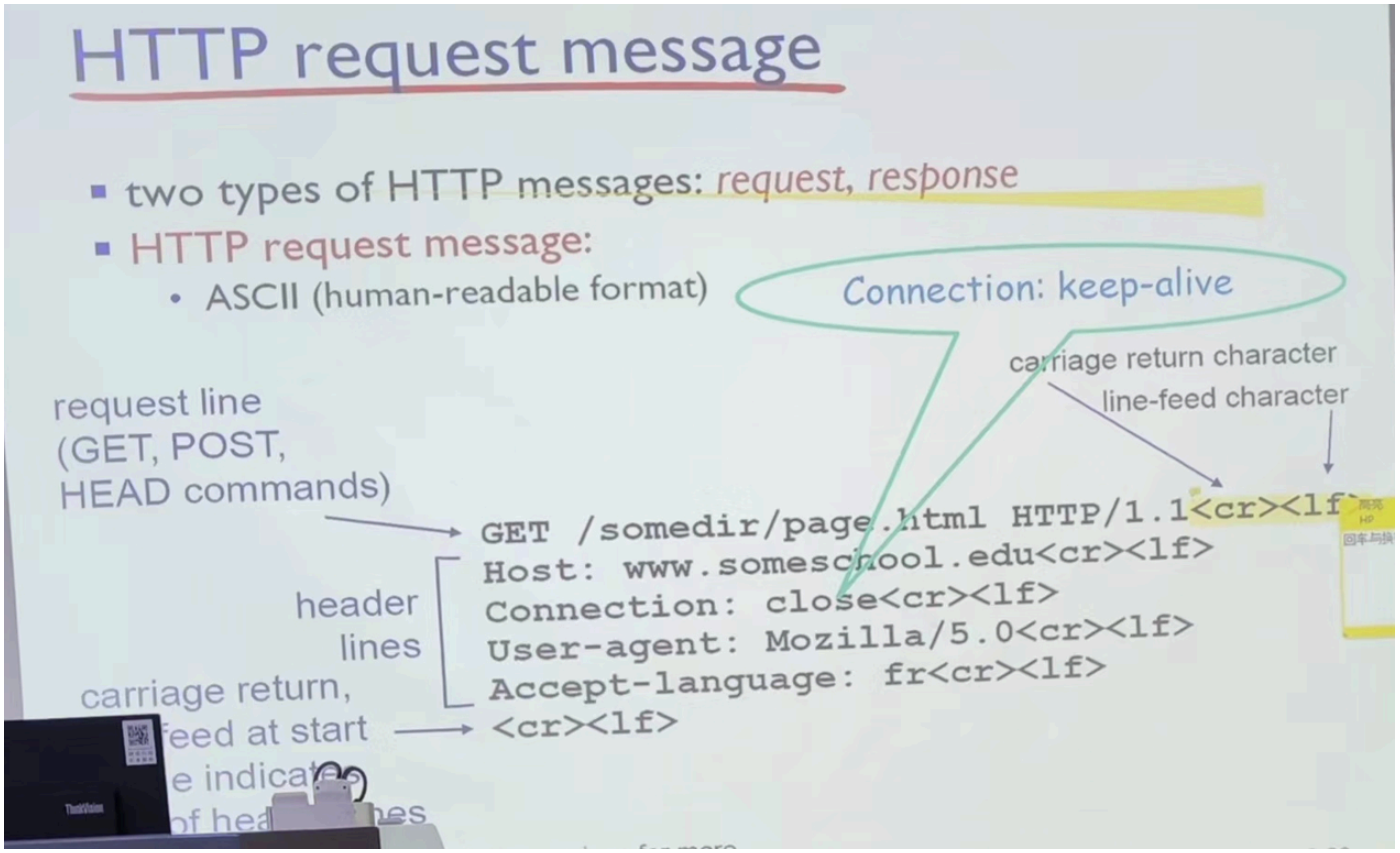
建立新的TCP连接。

所有引用的对象只需一个RTT：由于连接已经建立并保持打开状态，所以在同一连接上获取所有引用的对象只需要一个RTT。

这样，持久连接可以减少RTT的数量，减少操作系统开销，并提高性能。然而，持久连接也需要进行适当的管理，以避免资源占用和过度的连接数。

**HTTP request message**

- two types of HTTP messages: request, response




**HTTP response status codes**

- 200 OK：请求成功，服务器成功处理了请求，并返回了请求的对象。响应消息中会包含请求的对象。
- 301 Moved Permanently：请求的对象已经永久移动到了新的位置。响应消息中会包含新的位置信息（Location:）供客户端重新发起请求。
- 400 Bad Request：服务器无法理解客户端发送的请求消息。这可能是由于请求消息格式错误、缺少必要的信息等引起的。
- 404 Not Found：在服务器上未找到请求的文档或对象。服务器无法找到与请求URI匹配的资源。
- 505 HTTP Version Not Supported：服务器不支持请求中所指定的HTTP协议版本。这通常表示服务器不支持客户端使用的HTTP协议版本。

# cookies

- 在HTTP报文里
- keep User-server state

# Caches

- satisfy client request without involving origin server

# Electronic mail

Three major components:

- user agents / clients
- mail servers
- simple mail transfer protocol:
  - SMTP
  - POP3，IMAP，HTTP

**mail servers:**（边缘网络）

- mailbox contains incoming messages for user
- message queue of outgoing (to be sent) mail messages
- **SMTP** protocol between mail servers to send email messages

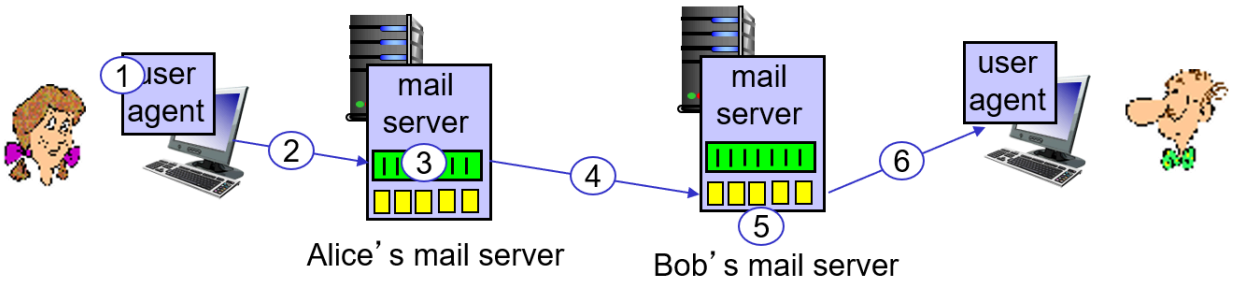
# SMTP

- uses TCP to reliably transfer email message from client to server, port 25
- ASCII
- persistent connections

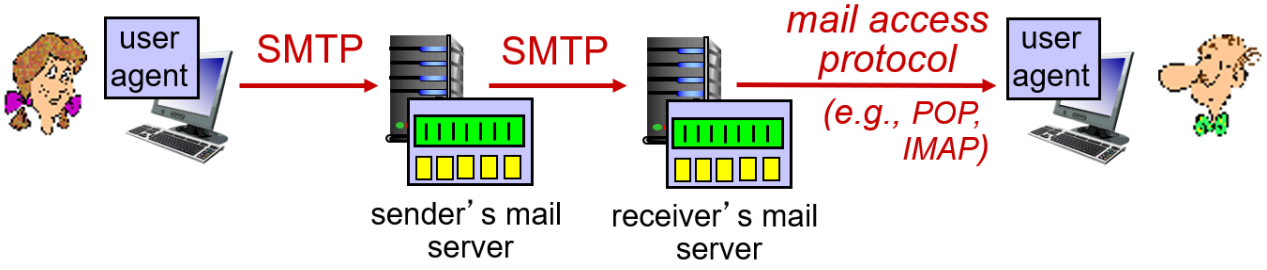
```
comparison with HTTP:
HTTP: pull
SMTP: push
- both have ASCII command/response interaction, status codes
HTTP: each object encapsulated in its own response message
SMTP: multiple objects sent in multipart message
```

e.g.

1) Alice uses UA to compose message "to" bob@someschool.edu
2) Alice's UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message




# Mail access protocols




- **SMTP只负责邮件的投递和存储，而不包括邮件的访问和检索**
- 邮件访问协议是用于从邮件服务器上检索邮件的协议。在这方面，有两个常用的协议：
  - POP（邮局协议）：POP是一种授权和下载协议，它允许用户通过客户端应用程序（如邮件客户端）从邮件服务器上下载邮件到本地设备。一旦邮件被下载，它通常就会从服务器上删除。
  - IMAP（Internet邮件访问协议）：IMAP是一种更高级的邮件访问协议，它提供了更多的功能。IMAP允许用户在邮件服务器上管理和操作存储的邮件，包括创建文件夹、标记邮件、搜索、同步等。与POP不同，IMAP在客户端和服务器之间保持邮件的同步状态，这意味着多个设备可以同时访问和管理同一邮箱。

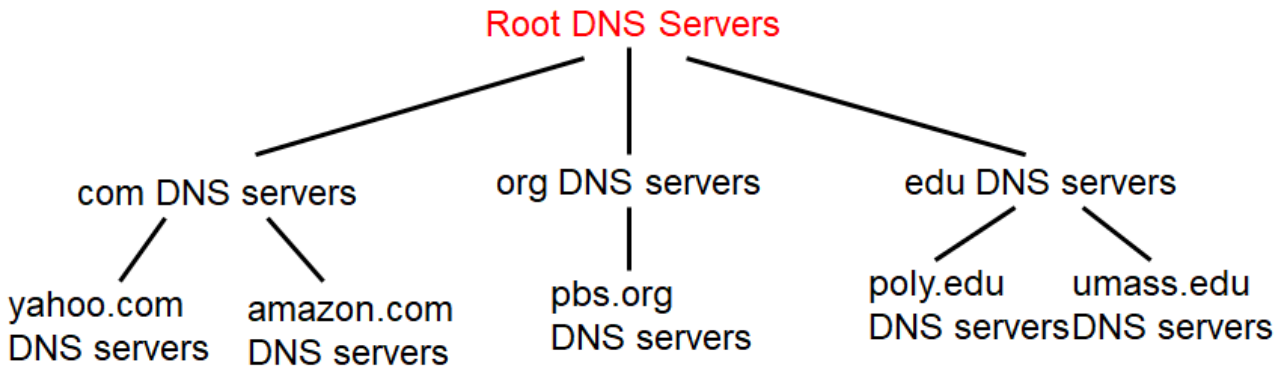
HTTP也可以用于访问电子邮件服务提供商的Web邮件界面，例如Gmail、Hotmail和Yahoo! Mail等。这些Web界面使用HTTP作为通信协议，允许用户通过浏览器访问和管理他们的电子邮件。

- **POP3:**
  - download-and-keep/download and delete
  - POP3 is stateless across sessions 如果Bob在不同的会话中使用不同的客户端连接到邮件服务器，他将无法访问之前下载的邮件
- **IMAP：**
  - Download-and-keep
  - keeps user state across sessions

# DNS: domain name system

一个分布式数据库，通过多个名称服务器的层次结构来实现。它是一个**应用层协议**，用于主机和名称服务器之间的通信，以解析域名和实现地址与名称的转换。

- hostname to IP address translation
- 域名解析：迭代Iterative &递归recursion
- a distributed, hierarchical database




顶级域名服务器（Top-Level Domain servers，TLD servers）是DNS架构中的一部分，它们负责管理顶级域名（Top-Level Domains，TLDs）的域名解析