Lesson 2.1: Application Layer

CSC450 – COMPUTER NETWORKS | WINTER 2019-20

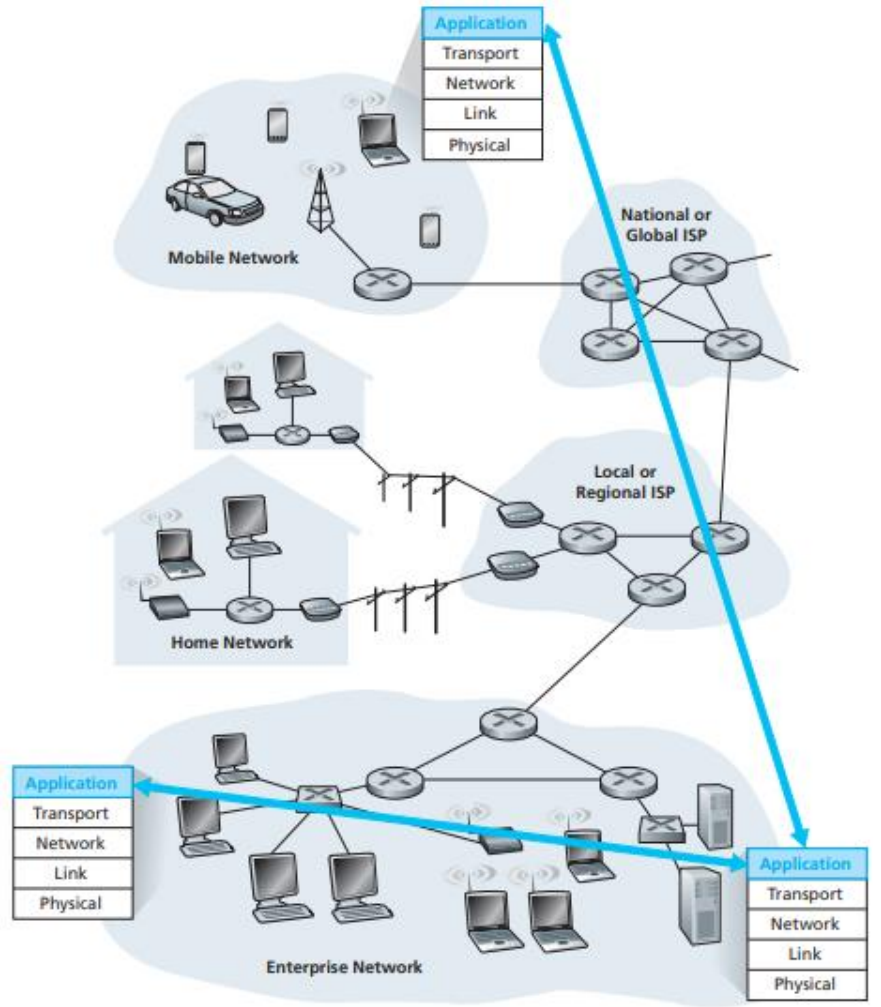
DR. ANDREY TIMOFEYEV

OUTLINE

- Foundation.
  - Architectures.
  - Processes.
  - Services.
  - Protocols.
- World Wide Web.
  - HTTP protocol.
  - Connection types.
  - HTTP message format.
  - Cookies.
  - Caching.

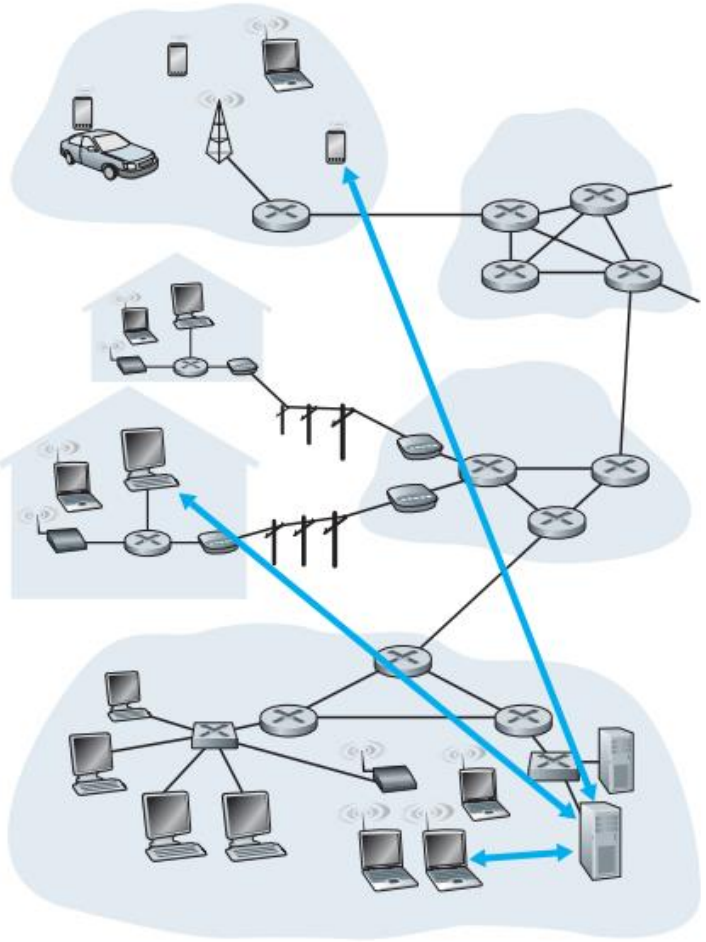
FOUNDATION: INTRO

- **Communication** for a network application takes place between **end systems** at the **application layer**.

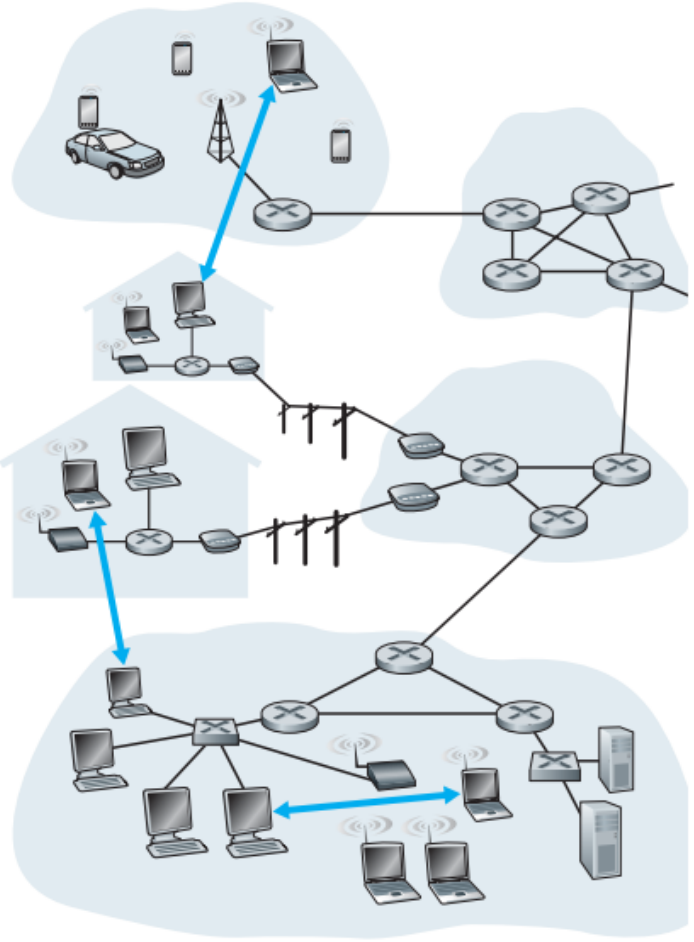


FOUNDATION: ARCHITECTURES

- Network application **architectures**:
  - **Client-server** architecture.
  - **Peer-to-peer** (**P2P**) architecture.



Client-server architecture



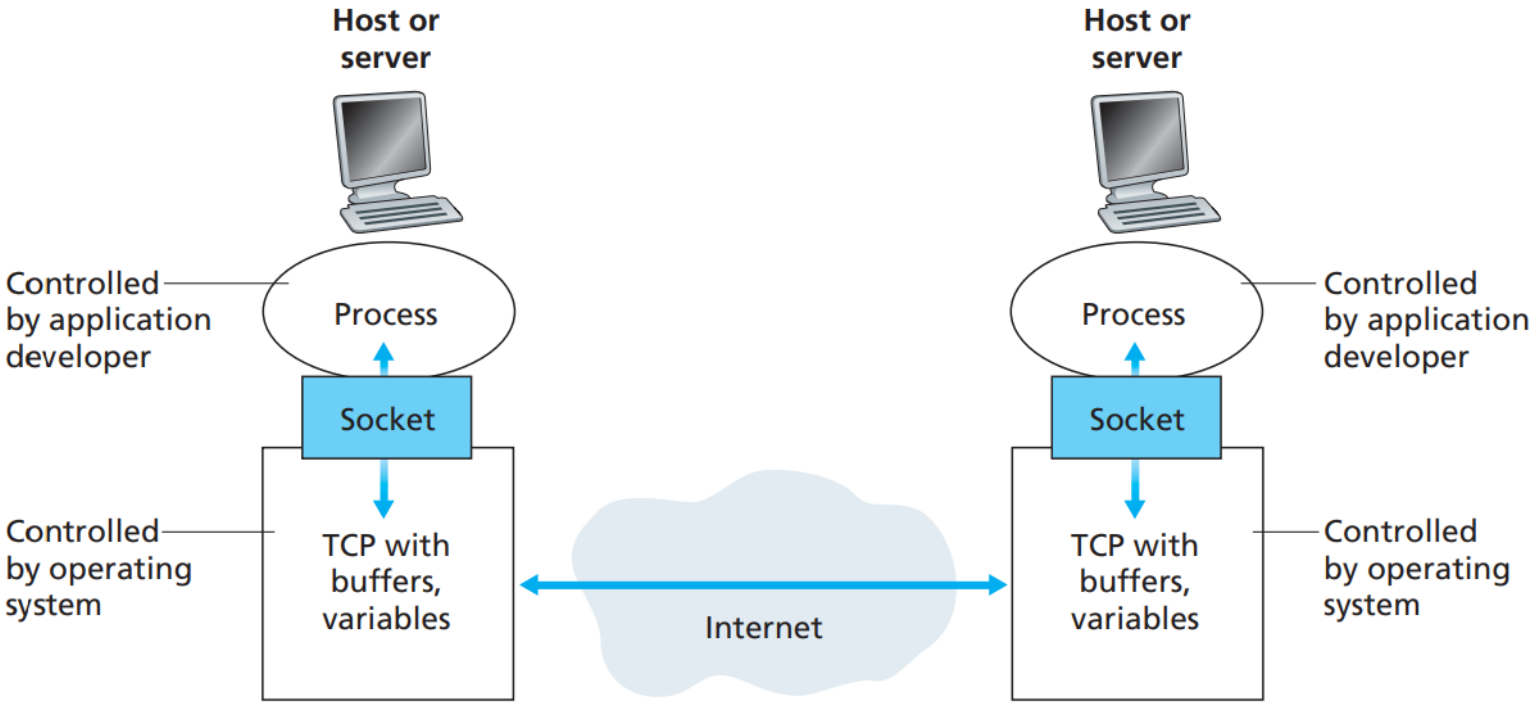
Peer-to-peer architecture

FOUNDATION: PROCESSES (1)

- **Application layer** provides **process-to-process communication** by allowing messages **exchange**.
  - **Process** – part of a **program** that is running within the **end system**.
- **Network application** consists of **pairs** of **processes** that send messages to each other.
  - **Client** – process that **initiates** the communication.
  - **Server** – process that **wait** to be contacted.
- Two pieces of **information** needed to **identify** a process:
  - **IP address** of the host.
  - **Port number** of the process.
    - **Identifier** that specifies the **process** on the **host**.

FOUNDATION: PROCESSES (2)

- **Message** sent from one process to another must go **through** underlying **network**.
- **Socket** – software **interface** that is used by **process** to **send** message into and **receive** message from the **network**.
  - Application Programming Interface (**API**) between the **application** and the **network**.



Application processes and sockets

FOUNDATION: SERVICES

- Two **transport-layer** protocols provide **services** to applications:
  - **Transmission Control Protocol** (**TCP**).
    - Connection-oriented.
    - Reliable data transfer.
    - Flow control.
    - Congestion control.
  - **User Datagram Protocol** (**UDP**).
    - Lightweight.
    - Connectionless.
    - Unreliable data transfer.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	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)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Internet applications and transport protocols

FOUNDATION: PROTOCOLS

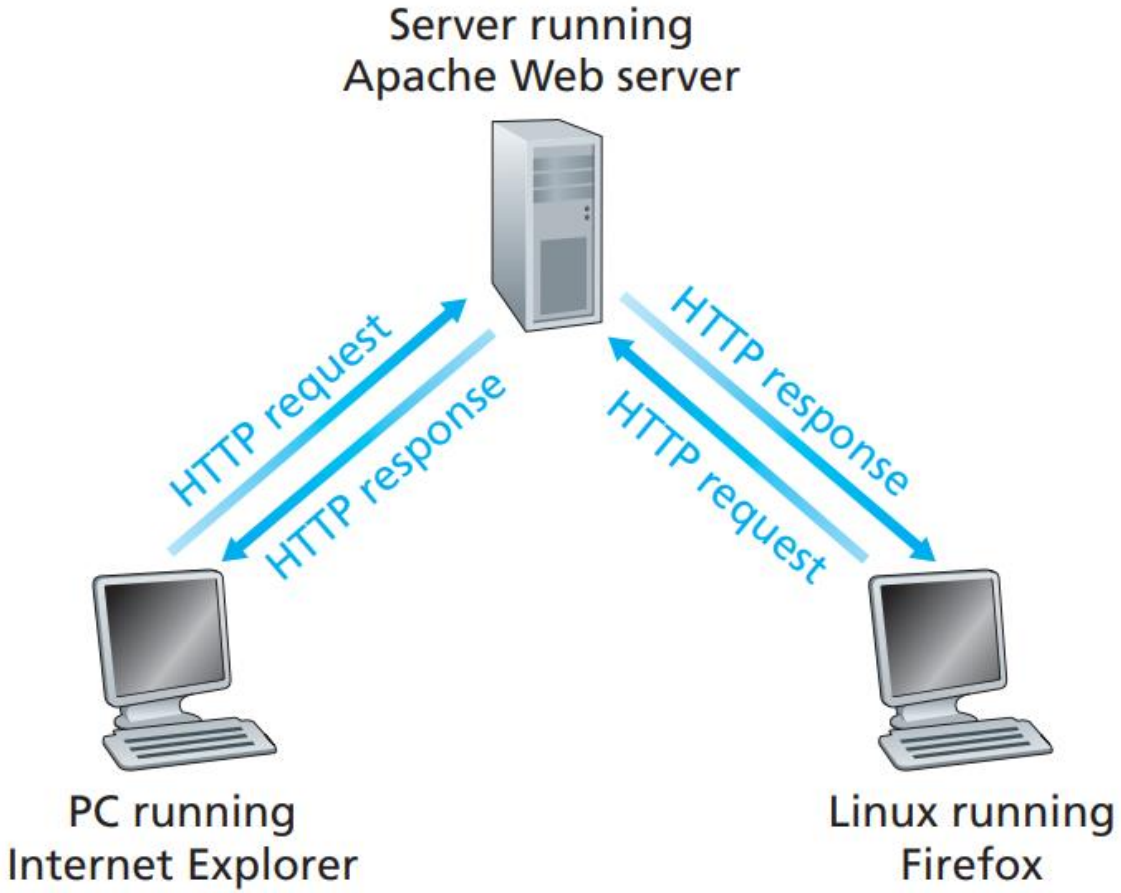
- **Application-layer protocols** define:
  - **Type** of messages exchanged.
    - **Request** messages / **response** messages.
  - **Syntax** of various messages.
    - **Fields** in messages and how they are **delineated**.
  - **Semantics** of the message fields.
    - **Meaning** of the information in the fields.
  - **Rules** on how process **sends** messages and **responds** to messages.
- Network **application** ≠ Application-layer **protocol**.
  - World Wide Web application vs. HTTP protocol.

WEB: OVERVIEW

- Overview of Web **components**:
  - **Web page** (document) consists of **objects**.
    - Objects are files (e.g. HTML file, JPEG image, Java applet, video).
  - **Web pages** consist of base **HTML-file**, which includes several **referenced** objects.
  - **Objects** are **addressable** by single **URL**.
    - www.someschool.edu/someDept/pic.gif
      - Host name / file path.
  - Web **browsers**.
    - **Request**, **receive** and **interpret** Web objects.
  - Wed **servers**.
    - **Store** and **send** Web objects.

WEB: HTTP PROTOCOL (1)

- **HyperText Transfer Protocol** (**HTTP**) – **Web's** application-layer **protocol**.
  - **Defines** how Web **clients request** Web pages from Web **servers** and how Web **servers transfer** Web pages to **clients**.
  - **Web browser** – **client** side of HTTP.
  - **Web servers** – **server** side of HTTP.



HTTP request-response behavior

WEB: HTTP PROTOCOL (2)

- **HTTP** runs on **TCP** transport-layer protocol.
  - **Client initiates** TCP connection to the **server** through **socket**.
    - Port number **80**.
  - **Server accepts** TCP connection from **client**.
  - HTTP **messages exchanged** between **browser** and **Web server**.
  - TCP **connection closed**.
- HTTP is **stateless** protocol.
  - **Server** maintains **no information** about **past** client **requests**.

WEB: HTTP CONNECTIONS (1)

- **HTTP** provides two types of **connections**:
  - **Non-persistent** connection.
    - At **most one object** sent over TCP **connection.**
      - Connection then **closed**.
    - Requesting **multiple objects** requires **multiple** TCP **connections**.
  - **Persistent** connection.
    - **Multiple objects** can be sent over **single** TCP **connection** between client and server.
    - **Default** HTTP mode.

WEB: HTTP CONNECTIONS (2)

- **Non-persistent** connection.
  - User enters URL: *www.someSchool.edu/someDepartment/home.index*
    - Contains references for 10 objects.

1a. HTTP client initiates TCP connection to HTTP server at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client.

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

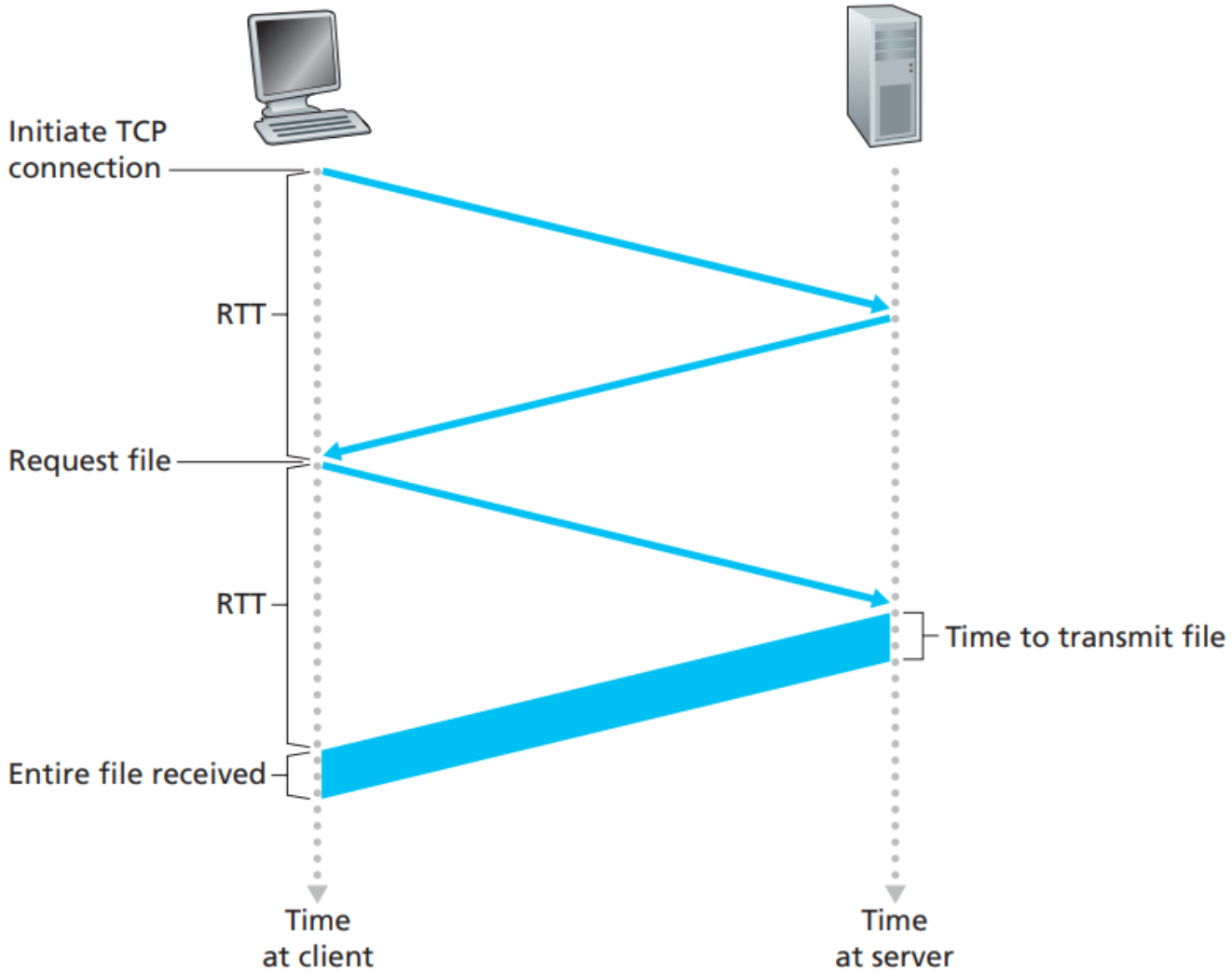
4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

WEB: HTTP CONNECTIONS (3)

- **Non-persistent** connection.
  - User enters URL:
    www.someSchool.edu/someDepartment/home.index
    - Contains references for 10 objects.
  - HTTP **response time**:
    - **2 RTT** to **establish** TCP connection.
      - "Three-way handshake" process.
    - File **transmission time**.
    - **Repeated** for all 11 objects.



Three-way handshake

WEB: HTTP CONNECTIONS (4)

- **Persistent** connection.
  - Server **leaves** connection **open** after sending **response**.
  - **Subsequent** HTTP **messages** between same client/server **sent** over **open connection**.
  - Client sends **requests** as soon as it encounters a **referenced object**.
    - Requests for objects are made **back-to-back** (*pipelining*).
  - Server **closes** a connection after a configurable **timeout interval**.

WEB: HTTP MESSAGE FORMAT (1)

- Two **types** of HTTP **messages**:
  - **Request** messages.
    - Request line.
    - Header line.
  - Example:

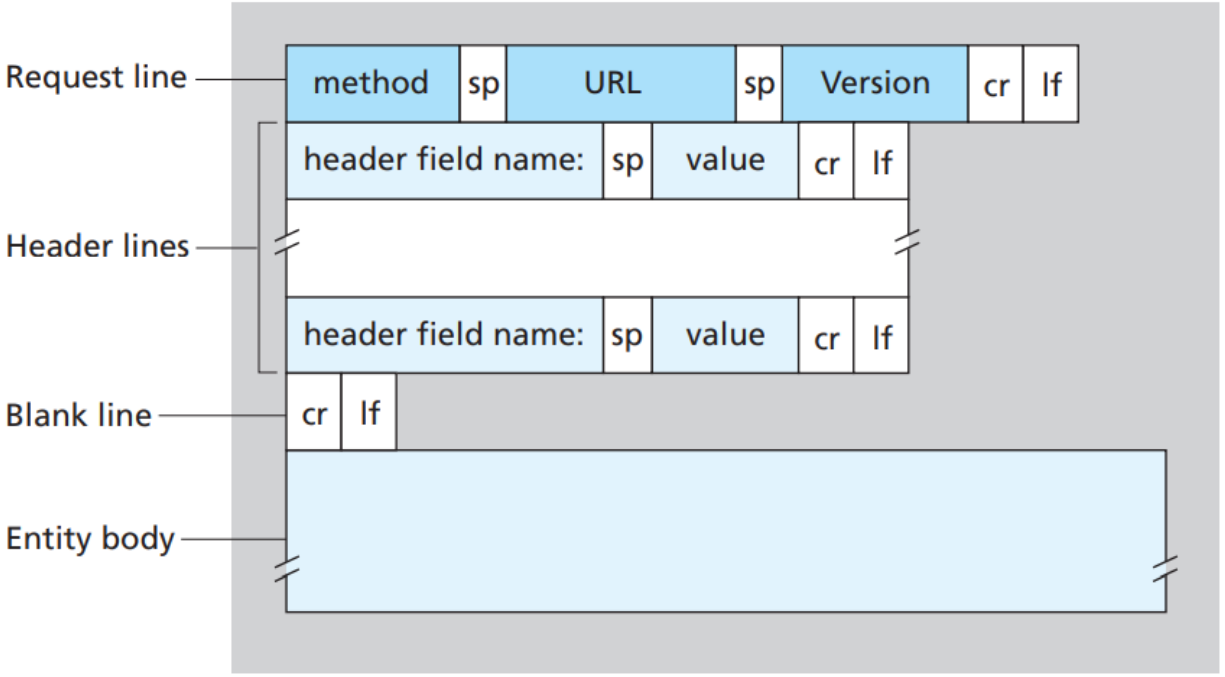
    GET /somedir/page.html HTTP/1.1
    Host: www.someschool.edu
    Connection: close
    User-agent: Mozilla/5.0
    Accept-language: en

  - **Response** messages.
    - Status line.
    - Header line.
    - Entity body (data).
  - Example:

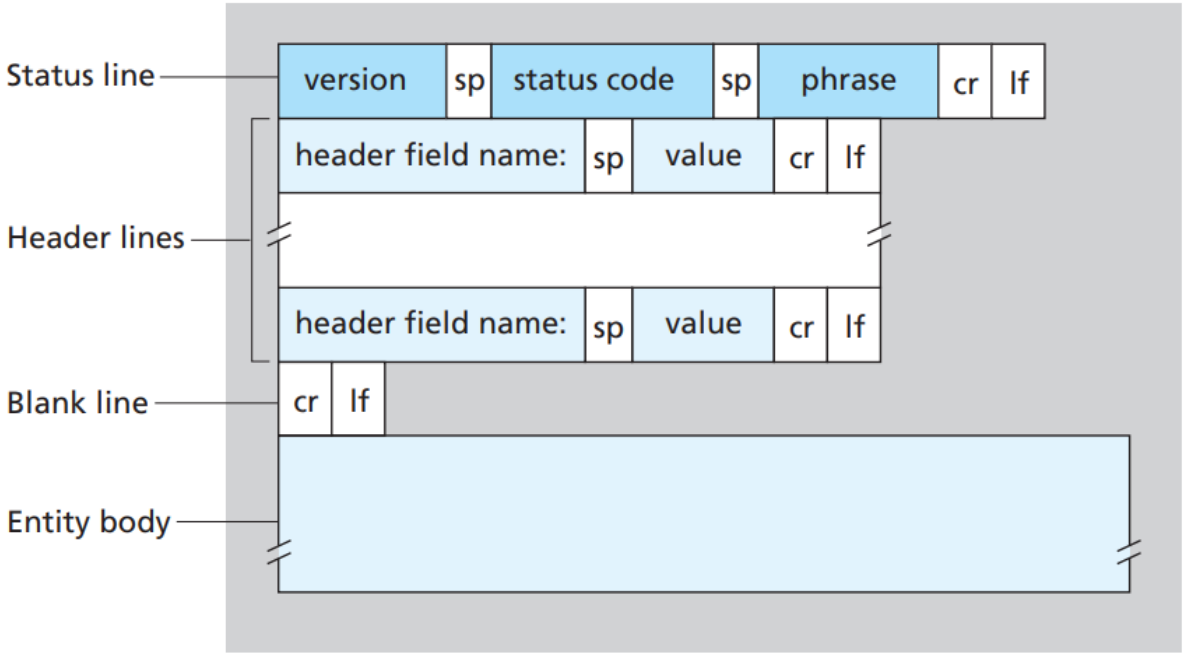
    HTTP/1.1 200 OK
    Connection: close
    Date: Tue, 09 Aug 2011 15:44:04 GMT
    Server: Apache/2.2.3 (CentOS)
    Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
    Content-Length: 6821
    Content-Type: text/html
    (data)

WEB: HTTP MESSAGE FORMAT (2)

- Two **types** of HTTP **messages**:
  - **Request** messages.
  - **Response** messages.



General format of HTTP request message



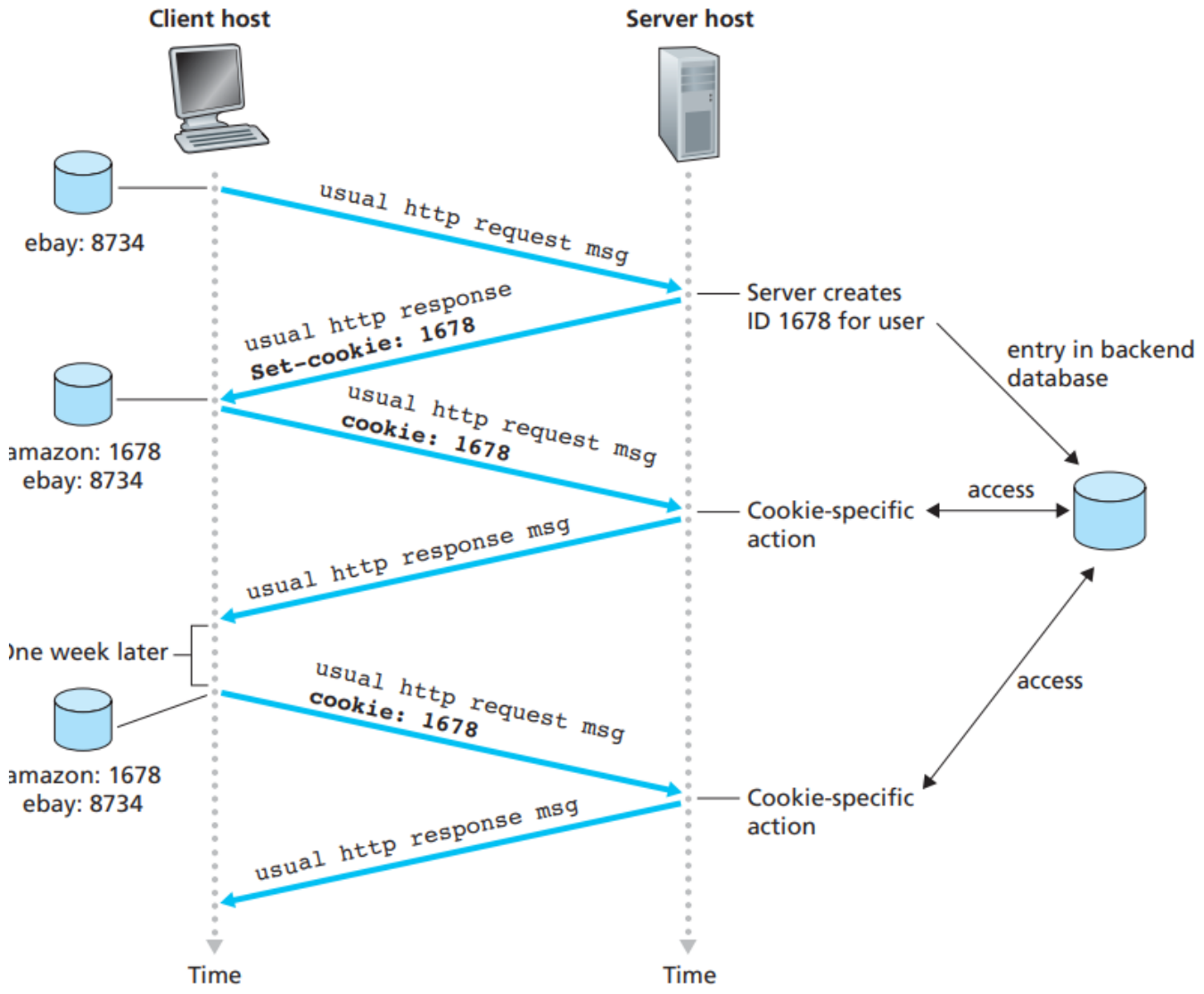
General format of HTTP response message

WEB: HTTP MESSAGE FORMAT (3)

- HTTP **response** message **status codes**:
  - **200 OK.**
    - Request succeeded, requested object is later in this message.
  - **301 Moved Permanently.**
    - Requested object moved, new location specified later in this message (*Location:*)
  - **400 Bad Request.**
    - Request message is not understood by server.
  - **404 Not Found.**
    - Requested document does not exist on this server.
  - **505 HTTP Version Not Supported.**

WEB: COOKIES

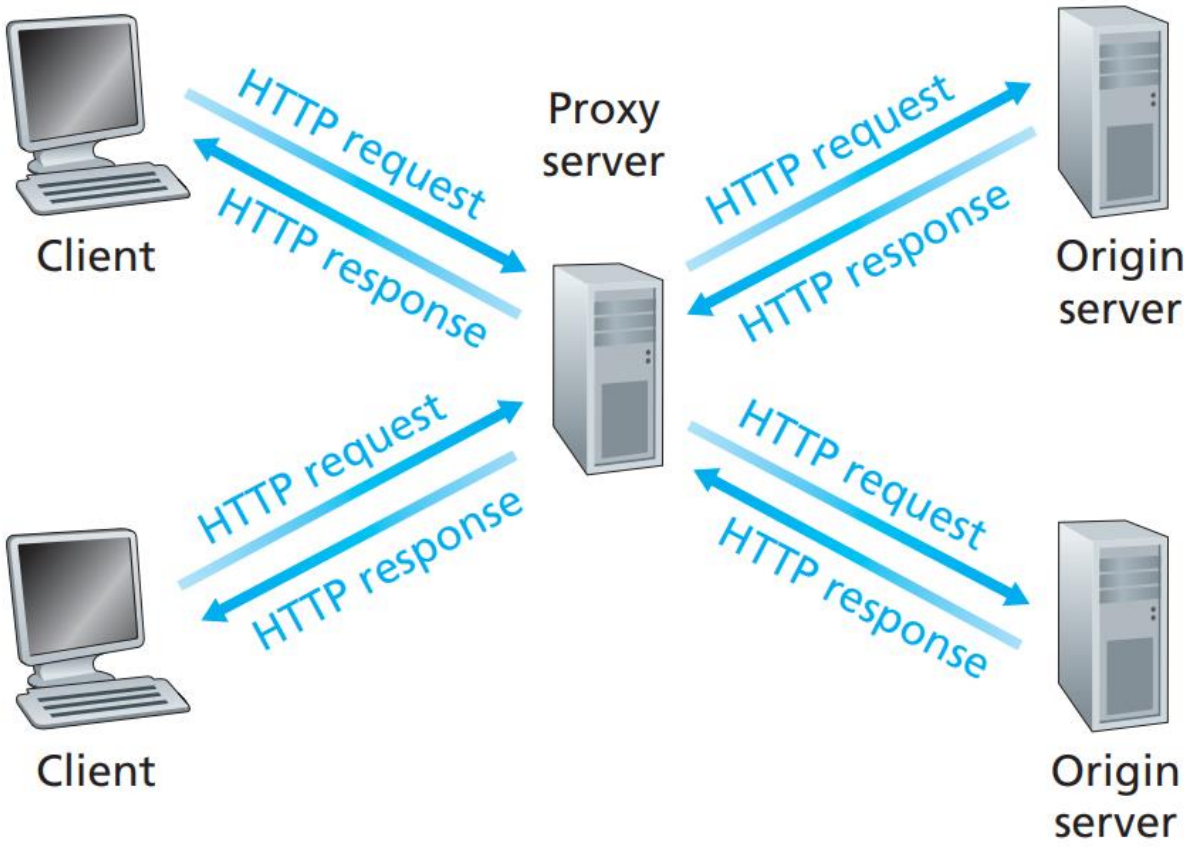
- **Cookies** are used by Web sites to keep **track** of users.
- Four main **components**:
  - **Cookie header** line in **response** message.
  - **Cookie header** line in **request** message.
  - **Cookie file** on **client**, managed by **browser**.
  - **Back-end database** at Web site.
- Cookies are **controversial**, since they can potentially invade of **privacy**.



User state with cookies

WEB: CACHING (1)

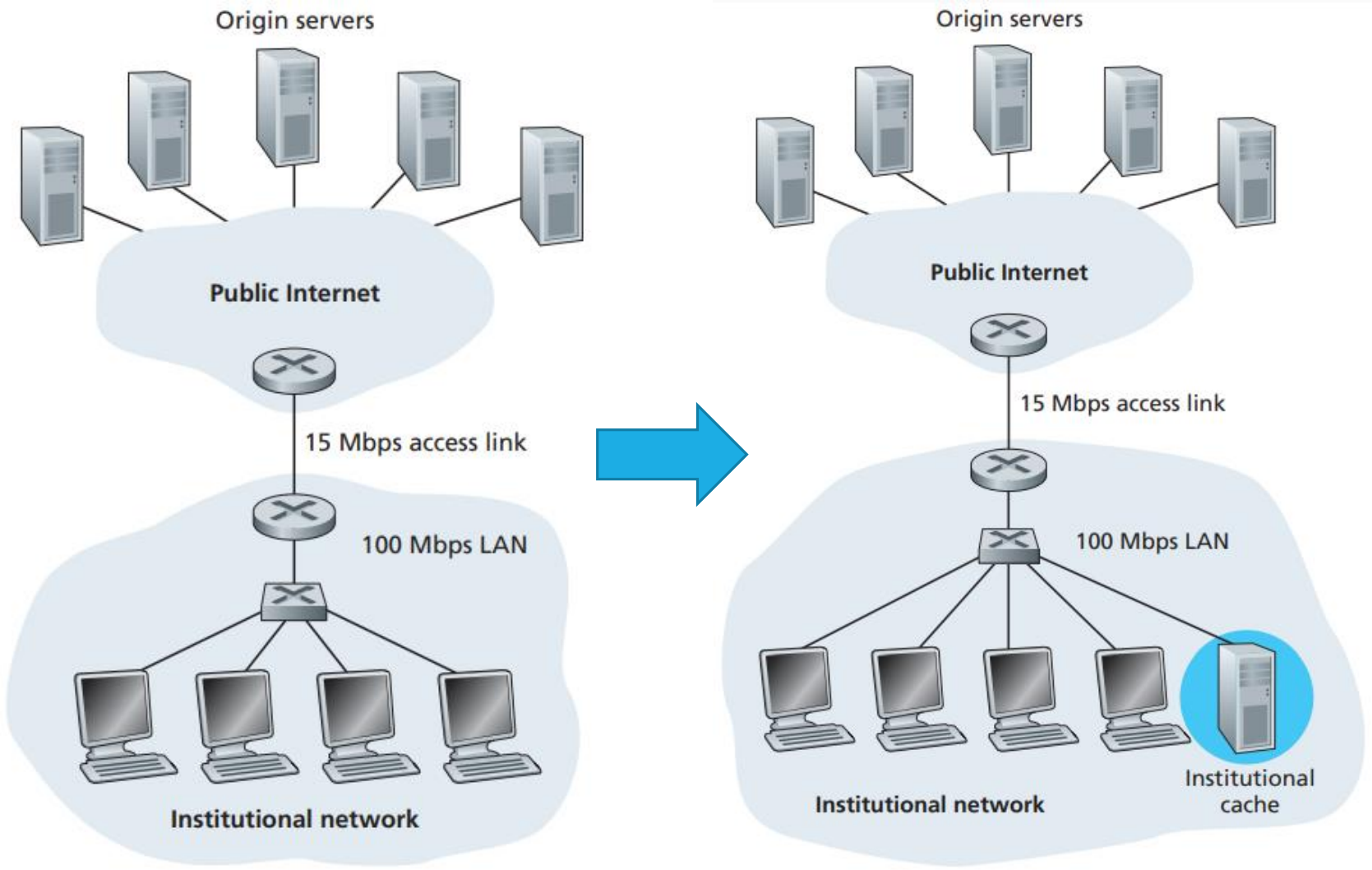
- **Web cache** (**proxy server**) – network entity that **satisfies** HTTP **requests** on the behalf of an **origin** Web **server**.
  - **Cache** is a **server** and a **client** at the same time.
- **Advantages** of Web cache:
  - **Reduces** the **response time** for a client request.
  - **Reduces traffic** on access link to the Internet.
    - Web caches reduce traffic in Internet as a whole, improving **performance** for all applications.



Clients requesting objects through a Web cache

WEB: CACHING (2)

- **Example**:
  - Link between network router and Internet = *15 Mbps.*
  - Average object size = *1 Mbit.*
  - Average request rate = *15 r/sec*.



Bottleneck between network and Internet

Adding cache to network

WEB: CONDITIONAL GET

- **Cache** introduces the problem of **stale objects**.
  - **Object** housed in Web **server** might have been **modified** since the last copy was **cached**.
- **Conditional GET** HTTP **request** mitigates this issue.
  - Cache **includes** "*If-Modified-Since:*" **header** line into **request** message.
    - **Equal** to *"Last-Modified:"* **header** line of **requested** object.
  - If the object has **not been modified**, server sends an **empty response** message with "**304 Not Modified**" response status code.

SUMMARY

- Client-server & peer-to-peer architectures.
- Process.
- Port number.
- Socket.
- Network application vs. Application-layer protocol.
- HTTP protocol.
  - Connection types.
  - HTTP messages.
    - Request.
    - Reply.
  - Cookies.
  - Caching.
  - Conditional GET.