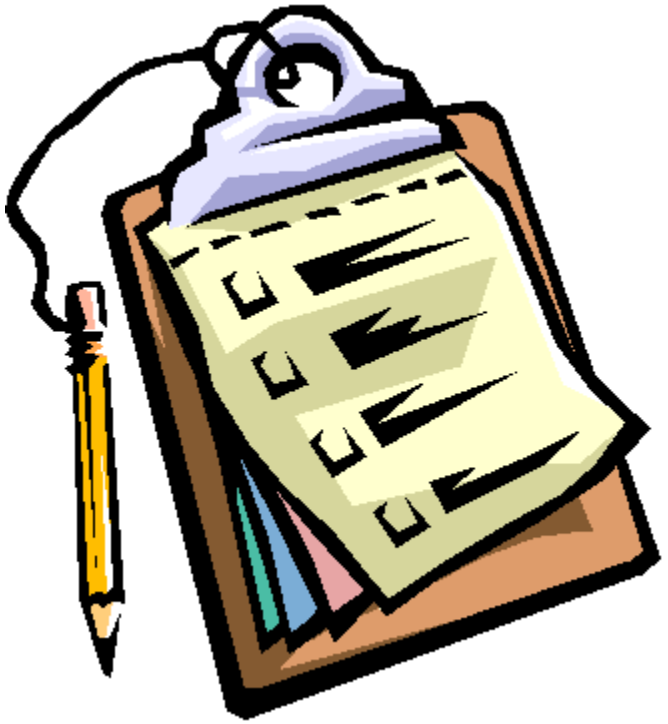
INTERNET ENGINEERING

HTTP Protocol

Sadegh Aliakbary

Agenda

- HTTP Protocol
- HTTP Methods
- HTTP Request and Response
- State in HTTP



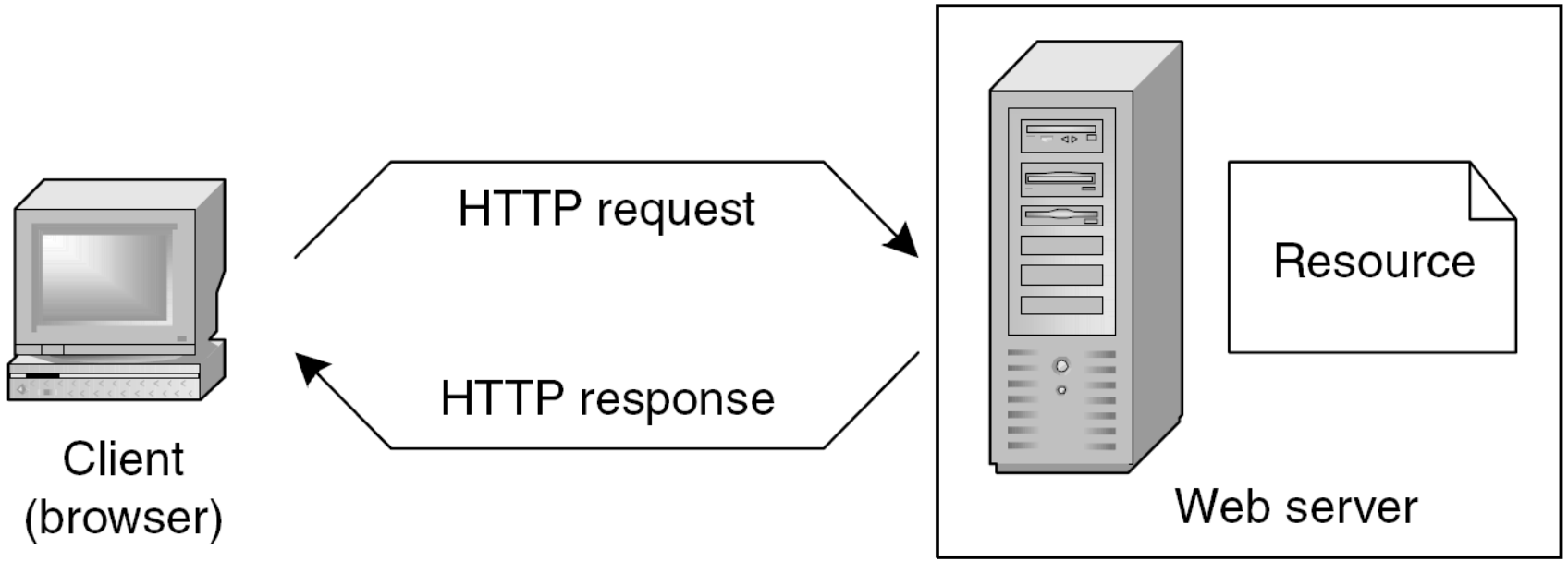
HTTP

HTTP

- Hyper-Text Transfer Protocol (HTTP)
- The fundamental technology at the origin of Web applications
- An *application-level* protocol
- Allows users to make requests to remote servers
- HTTP is a *client-server application protocol*
- A client requests a server's content or service function
- Clients initiate communication sessions
  - with servers which await (*listen* to) incoming requests

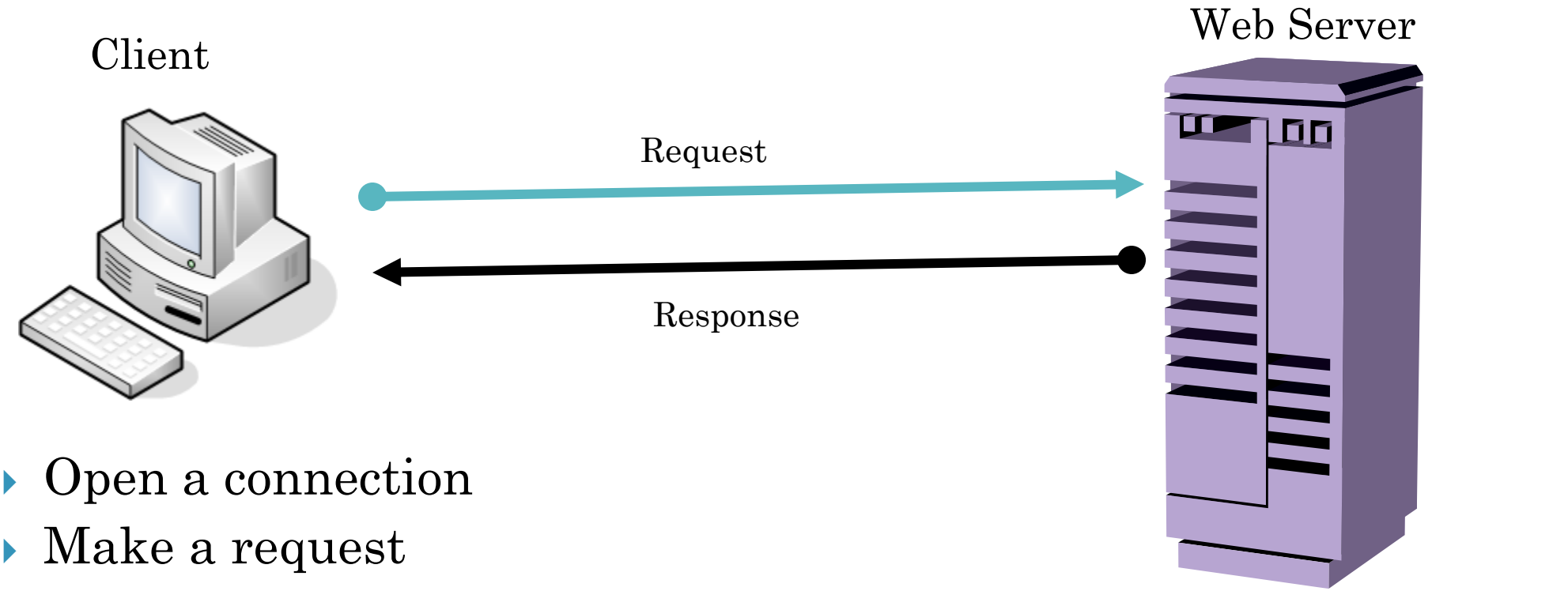
HTTP

- HTTP protocol defines the rules by which a client program (a browser or user agent) and a server program (called Web server) interact
- A request may address a file of any format stored in the Web server, or even the invocation of a program to be executed at the server side



HTTP

- It is connection oriented



  - Open a connection
  - Make a request
  - Server responds
  - Close connection

Listening via a port (80)

HTTP

- HTTP resources are located by means of *Uniform Resource Locators (URLs),*
- which are structured strings of the format:

  http[s]: // <host> [: <port>] [<path> [? <query>]]

- **E.g.,**
  - *http://sbu.ac.ir/cols/cse*
  - *https://www.google.com:8081/dir/form.php?ln=en&cn=ir*

HTTP Request

- HTTP requests (have a fixed format) consists of 3 parts:

1- Request line, consists of 3 parts:

- HTTP **method**, requested URL, (HTPP) protocol version
- E.g., GET /pub/WWW/TheProject.html HTTP/1.1

2- Message header (optional)

- the details (such as cookies) of **what the browser wants**
- It also contains the type, version and capabilities of the **browser**
- so that server returns **compatible data**

3- The **request body** (also optional)

HTTP Methods

- **GET**
  - Retrieve information from the server using given URL
  - Requests only retrieve data: should have **no effect** on the data
- **HEAD**
  - Same as GET, but transfers the status line and header section only
  - Used for obtaining meta-information about the web page
  - E.g., for testing hypertext links for validity

HTTP Methods (cont'd)

- **POST:** Send data to the server using **HTML forms.** E.g.?
  - Example: customer information
  - The user's input is packaged as an attachment to the request **(request body)**
  - a **sizeable user's input** (e.g., a long text) to be processed by the server
  - The request body **as a new argument** of the resource identified by the Request-URL

Question

- Why should we use POST method?
- What if we embed the parameters in URL (as query string) and use GET method?

- **Answer:**
- HTTP GET request and URL size are limited in size
- The limit is dependent on the server and client (browser)
  - e,.g., 2048 bytes

HTTP Methods (cont'd)

- **PUT**
  - Replaces current target resource with the **upload**ed content
- **DELETE**
  - Removes current target resource given by a URL
- Other Methods
  - CONNECT, OPTIONS, TRACE, …

- The most important **HTTP methods**: GET, POST, PUT

Idempotent Methods

- An idempotent HTTP method:
- A method that can be called many times without different outcomes
- It would not matter if the method is called only once, or ten times over
- The result should be the same
- Idempotent Methods?
  - GET, HEAD, PUT, DELETE
  - POST is not idempotent
- Nowadays, are GET methods necessarily idempotent?

HTTP Response

- HTTP response is structured in 3 parts:

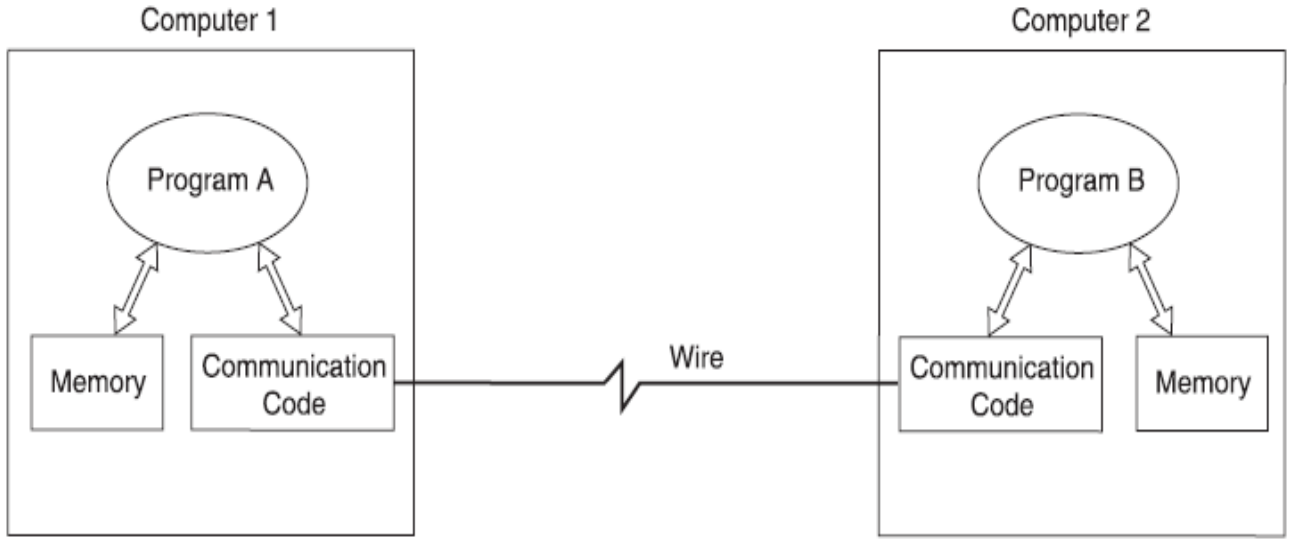
1. A **status line**
   - protocol version + a numeric status code + associated message
     - example: `HTTP/1.1 404 Not found`
     - example: `HTTP/1.0 200 OK`
2. A set of optional **header fields**
   - Example: *Content-Type: text/html.*
3. A **message body**
   - Example: An HTML page.

HTTP is Stateless

- If you view 10 web pages, your browser makes 10 independent HTTP requests
- At any time in between those requests, you are free to restart your browser program
- At any time in between those requests, the publisher is free to restart its server program (Web server)

Stateful Communications Protocols

- Two programs establish a connection and **proceed to use it**
  - for as long as necessary
  - typically until one of the programs terminates
- Program B can easily remember what Program A has said already



HTTP is *Stateless*

- Each HTTP request is treated by the Web server as an atomic and independent call
- There is no difference between **a sequence of two requests** by different users or by the same user
- When the connection is over, **it is over**
- If the user follows a hyperlink, that's **a new HTTP request** is created

HTTP is *Stateless (cont'd)*

- What if Yahoo is using multiple servers to operate its site?
  - The second request might go to an entirely different machine
- This sounds fine for browsing Yahoo
- What about shopping at an e-commerce site (Digikala)?
- If you put something in your shopping cart on one HTTP request, you still want it to be there ten clicks later
- **Engineering Challenge**: Creating a stateful application on top of a fundamentally stateless protocol

Anatomy of a Typical HTTP Session

- User types www.yahoo.com into browser
- Browser translates www.yahoo.com into an IP address and tries to open a TCP connection with port 80 of that address
- Browser sends the following byte stream:
  - Get / HTTP/1.1
  - What is the meaning?
    - The "GET" means that the browser is requesting a file
    - The "/" is the name of the file, in this case simply the root index page

Anatomy of a Typical HTTP Session

- Yahoo responds with a set of headers indicating
  - Which protocol is actually being used
  - Whether or not the file requested was found
  - How many bytes are contained in that file
  - Kind of information in the file, such as application (Multipurpose files), audio, video, image, text, multipart,…
- Yahoo's server sends a blank line to indicate the end of the headers
- Yahoo sends the contents of its index root
- The **TCP connection is closed** when the file has been received by the browser

Anatomy of a Typical HTTP Session

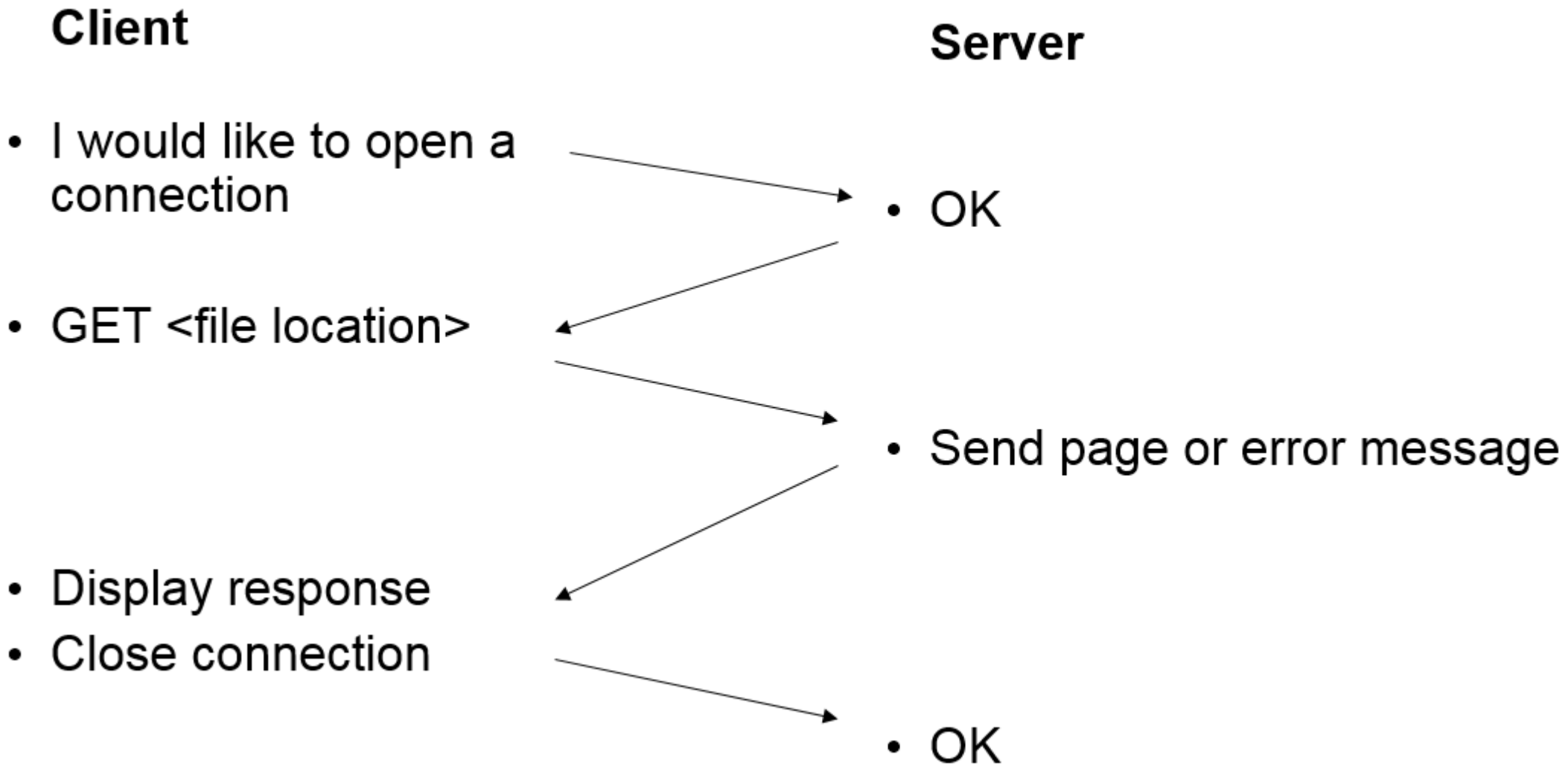
- The Unix telnet command with an optional argument specifying the port number for the target host
- The HTTP status code of 200 means that the file was found (“OK”)

```
bash-2.03$ telnet www.yahoo.com 80
Trying 216.32.74.53...
Connected to www.yahoo.akadns.net.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 200 OK
Content-Length: 18385
Content-Type: text/html

<html><head><title>Yahoo!</title><base href=http://www.yahoo.com/>...
```

Overview of an HTTP Connection



Example Session

Client Request

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Request line

A header field

Server Response

Status Line

```
HTTP/1.1 200 OK
```

Headers

```
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Accept-Ranges: bytes
Connection: close
```

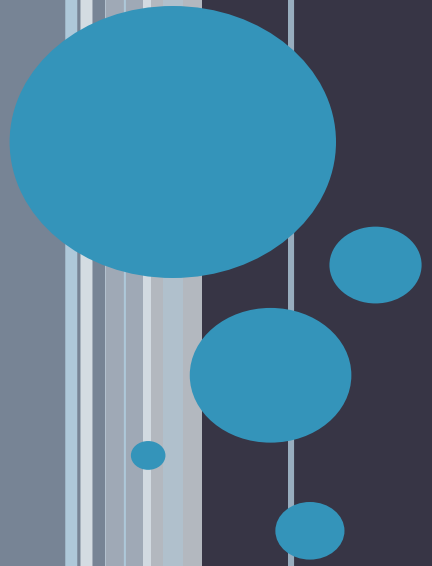
An empty line

Body

```
<html>
<head> <title>An Example Page</title> </head>
<body>  Hello World </body>
</html>
```

HTTPS

- **Secure** HTTP: HTTP over **SSL**
- HTTPS consists of communication over HTTP, within an encrypted connection by SSL
- Benefits:
  - Authentication of the visited website
  - **Privacy** and **integrity** of the exchanged data
- Which web pages should be redirected to HTTPS?
  - E.g., Login Pages



Web Sessions and Stateful Web Applications

Web Session

- A **web session**: a sequence of HTTP request and response transactions associated to **the same user**
- Modern web applications require the **retaining of status about each user** for the duration of multiple requests
- **Sessions** provide the ability to establish **variables**
  - E.g., localization settings, Shopping basket, …
- WebApps will make use of sessions once the user has authenticated
- Additionally, webapps can create sessions to keep track of **anonymous users**
  - E.g., maintaining the user language preference.

Web Sessions

How to Implement Web Session?

- HTTP is Stateless
- **Engineering Challenge**: Creating a stateful application on top of a fundamentally stateless protocol
- How to Maintain State over the Stateless HTTP?
- An idea: Client IP address
  - Does the server know it? Yes
  - Is it a good idea as the session identity? No
    - Many computers with the same IP (e.g., NAT), Many browsers on the same IP, …

How to Maintain State over HTTP?

- A general idea: Write some information out to an individual user that will be returned on user's next requests
- How?
- URL Rewriting
  - Redirect the request, and add some query string for the session variables
  - Drawbacks?
  - URL length constraint, Security problems, …
- Modern WebApps use **Cookies**

Cookie

- Remember the general Idea:
  - Write some information out to an individual user that will be returned on that user's next request
- An **HTTP cookie**, also called **web cookie**, **Internet cookie**, and **browser cookie**
- A small piece of data **sent from a website** and **stored in the user's web browser** while the user is browsing it
- WebApps can use **cookies** to both store and retrieve information on the **client side**
- A simple, persistent, **client-side state**
- How does it work???

How Cookies Work

- After you add a book to your shopping cart, the server writes "Set-Cookie: cart_contents=1588750019"
- As long as you do not quit your browser, on every subsequent request to the server, the browser adds a header "Cookie: cart_contents=1588750019"
- The server-side scripts can read this header and extract the current contents of the shopping cart
- Cookies are limited:
  - each browser to store no more than 20 cookies on behalf of a server
  - and each of those cookies must be no more than 4 kb
  - The reason? Cookie information will be passed back up to server.

Cookies Common Solution

- Modern webapps use cookies, but not for the entire state variables
  - E.g., cookies usually do not persist the permissions, shopping basket, …
  - Why?
    - Security, cookie size limitations, …
- In many webapps, Cookies simply maintain a **session ID**
  - Using unique identifier for the data rather than the data
- Information about the contents (variable values) will be kept **on the server**

Cookies: Example

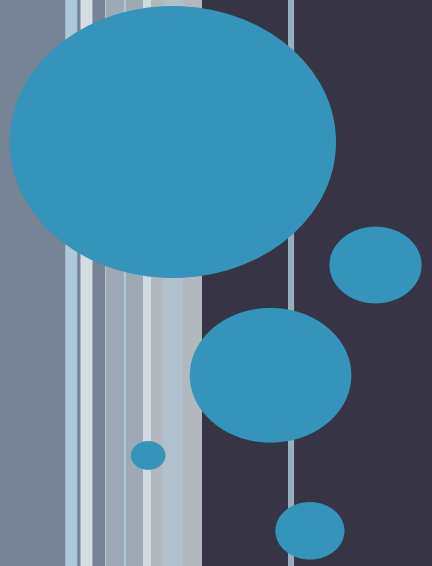
```
GET /index.html HTTP/1.1
Host: www.example.org ...
```

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=09 Jun 2016
...
```

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123 ...
```

Server-Side Storage

- You've got ID information going out to and coming back from browsers, via either the cookie extension to HTTP or URL rewriting
- How to keep associated information on the Web server?
  - Memory? File? RDBMS?
- Memory: good performance
- File: to ensure that sessions can survive a server restart
- RDBMS (or shared file): to cluster, failover, load-balance the servers



Server to Client Communication

The Need for Server to Client Communications

- Many applications need communication from server to client.
  E.g.?
  - Chats, games, status updates, …
  - Examples in Facebook, Gmail, Gmail chats, Youtube, …
- But HTTP 1.1 is ***pull-based***
  (HTTP 1.1 is the current common HTTP protocol version)
  - Interaction occurs only when the **client calls the server**
  - There is no way in which the server may call back the client
  - Notification of clients cannot be implemented using HTTP 1.1 alone

How to Implement Server to Client Communications

- HTTP 1.1 is pull-based, so what is the Solution?
- Possible solution: Polling
  - The client frequently asks the server for the new information
  - Drawback?
  - Performance, Implementation Complexity
- Possible solution: non-standard approaches
  - Flash, Applet, Silverlight, …
  - Drawback? Benefits?
  - Compatibility problems

A Common Solution: WebSockets

- **WebSocket** is a protocol providing full-duplex communication channels over a single TCP connection
- The WebSocket protocol and API are **standardized** (IETF and W3C)
- WebSocket is designed to be implemented in web browsers and web servers
- The WebSocket protocol is currently supported in most major browsers (Chrome, IE, Firefox, …)
- Unlike HTTP, WebSocket provides full-duplex communication
  - WebSocket is not based on HTTP
- **HTML5** also includes WebSockets
  - HTML5 is based on HTTP+WebSockets Protocols

HTTP/2

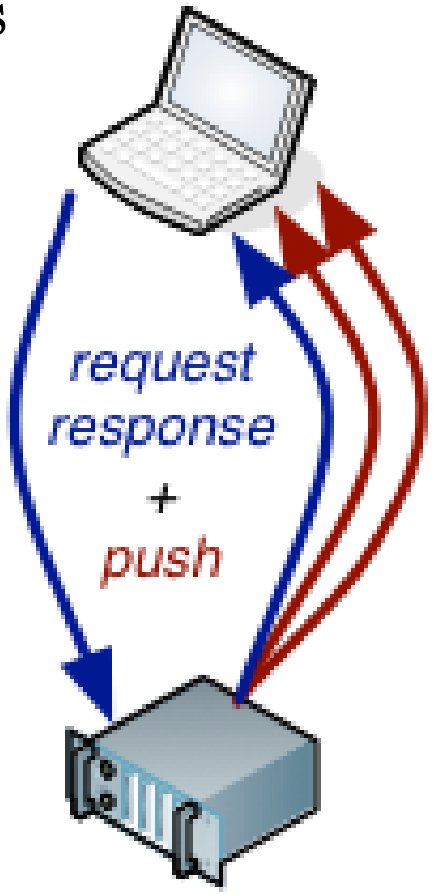
- HTTP/2 is the first new version of HTTP since HTTP 1.1, which was standardized in 1997
- (HTTP 1.1 is the current common HTTP protocol version)
- The HTTP/2 specification published as RFC 7540 in May 2015
- Most major browsers added HTTP/2 support by the end of 2015
- 6.3% of the top 10 million websites supported HTTP/2 (as of January 2016)

Some of the HTTP 2 Goals

- Allows clients and servers to elect to use HTTP 1.1, 2.0
- Maintain high-level compatibility with HTTP 1.1
  - E.g., methods, status codes, and most header fields
- Decrease latency to **improve speed** in web browsers
  - Data compression of HTTP headers
  - **Server push** technologies
  - Pipelining of requests
    - multiple HTTP requests sent on a single TCP connection without waiting for the corresponding responses
- …

Server Push

- With HTTP/1.1, the server had to wait until the client sends request
- With HTTP 2.0, the server can send multiple responses for a single client request
- Useful scenarios?
- Average page requires dozens of additional assets, such as JavaScript, CSS, and images
  - With HTTP 2, server may send these resources instead of waiting for the client to discover them
- Send events from server to client
  - E.g., A new email is received, your friend logged in, …



Server to Client Communication Techniques

- Related terms:
  - WebSockets
  - Server Push
  - Comet
  - Ajax Push
  - Reverse Ajax
  - Two-way-web
  - HTTP Streaming
- **Note**: the common needed technique is the request-response model
  - Most web usecases do not need server events or full-duplex communication

Quiz

1- Assume This Scenario:

- Client enters the DigiKala.com
- adds some products to his/her basket
- signs in
- chats with the support-team (online chat)
- signs out
- exits the browser

- Explain how these concepts are used:
  (what happens on each of the mentioned steps)
  - HTTPS
  - Cookie (name cookie variables)
  - Server side session (name the session variables)
  - WebSockets or Push technology

Further Reading

- http://www.tutorialspoint.com/http/
- Search for:
  - HTTP 2
  - WebSockets
  - Flash, Applet, Silverlight

The End