URL, HTTP

Prof. Sruti S. Singh

- The ***World Wide Web (WWW)***, commonly known as the Web, is an information system where documents and other web resources are identified by ***Uniform Resource Locators*** (**URLs**, such as https://example.com/), which may be interlinked by hypertext, and are accessible over the Internet.
- The resources of the Web are transferred via the ***Hypertext Transfer Protocol (HTTP)***.
- Think of the World Wide Web as a vast collection of electronic files stored on millions of computers all around the world. Hypertext links these files together. The addresses used to locate these files are called Web Addresses or Uniform Resource Locators (URLs) .

Uniform Resource Locator(URL):

- The information contained in a URL gives you the ability to hop from one web page to another with just a click. When you type a URL into your browser or click a hypertext link, your browser sends a request to a remote computer, called a web server, to download one or more files. Every URL is unique, identifying one specific file.
- A web page needs to have a unique identifier to distinguish it from other web pages.
- To define a web page we need four identifiers: **protocol, host, port, path**

- Protocol: The first identifier is the protocol that we need in order to access the web page.
- Host: The host identifier is the IP address of the server which can be defined in dotted decimal notations or as domain name.
- Port: This identifier is used when port number is needed.
- Path: The path identifies the location and name of the file.

URL format: protocol://host/path *OR* protocol://host:port/path

Example: *http://example.org/wiki/Main_Page*

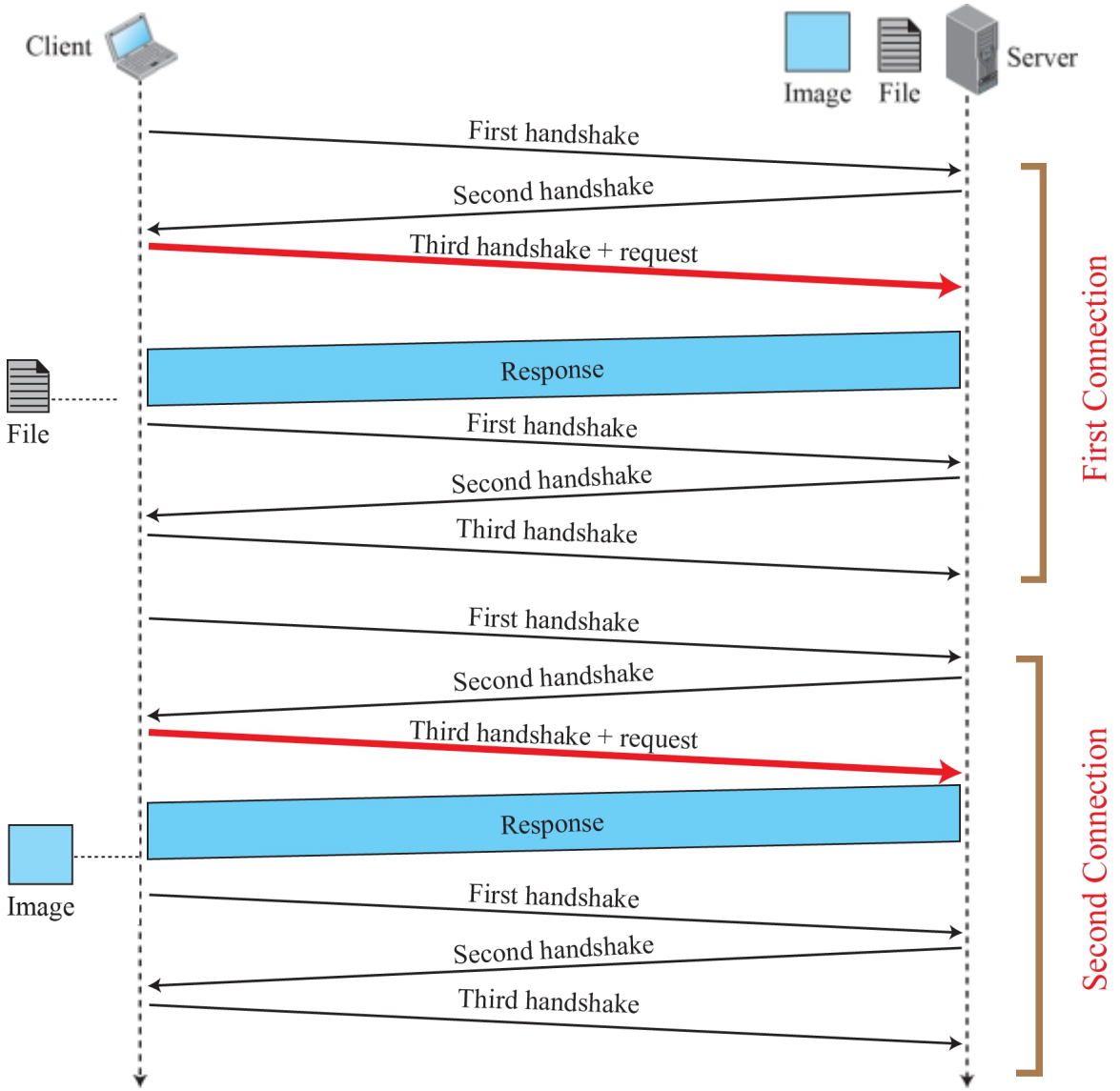
HyperText Transfer Protocol (HTTP):

- The HTTP is the Web's application-layer protocol for transferring various forms of data between server and client like plaintext, hypertext, image, videos and sounds.
- HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is **TCP 80**, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.
- *HOW HTTP WORKS:* HTTP is implemented in two programs: a client program and a server program, executing on different end systems, talk to each other by exchanging HTTP messages. The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces.

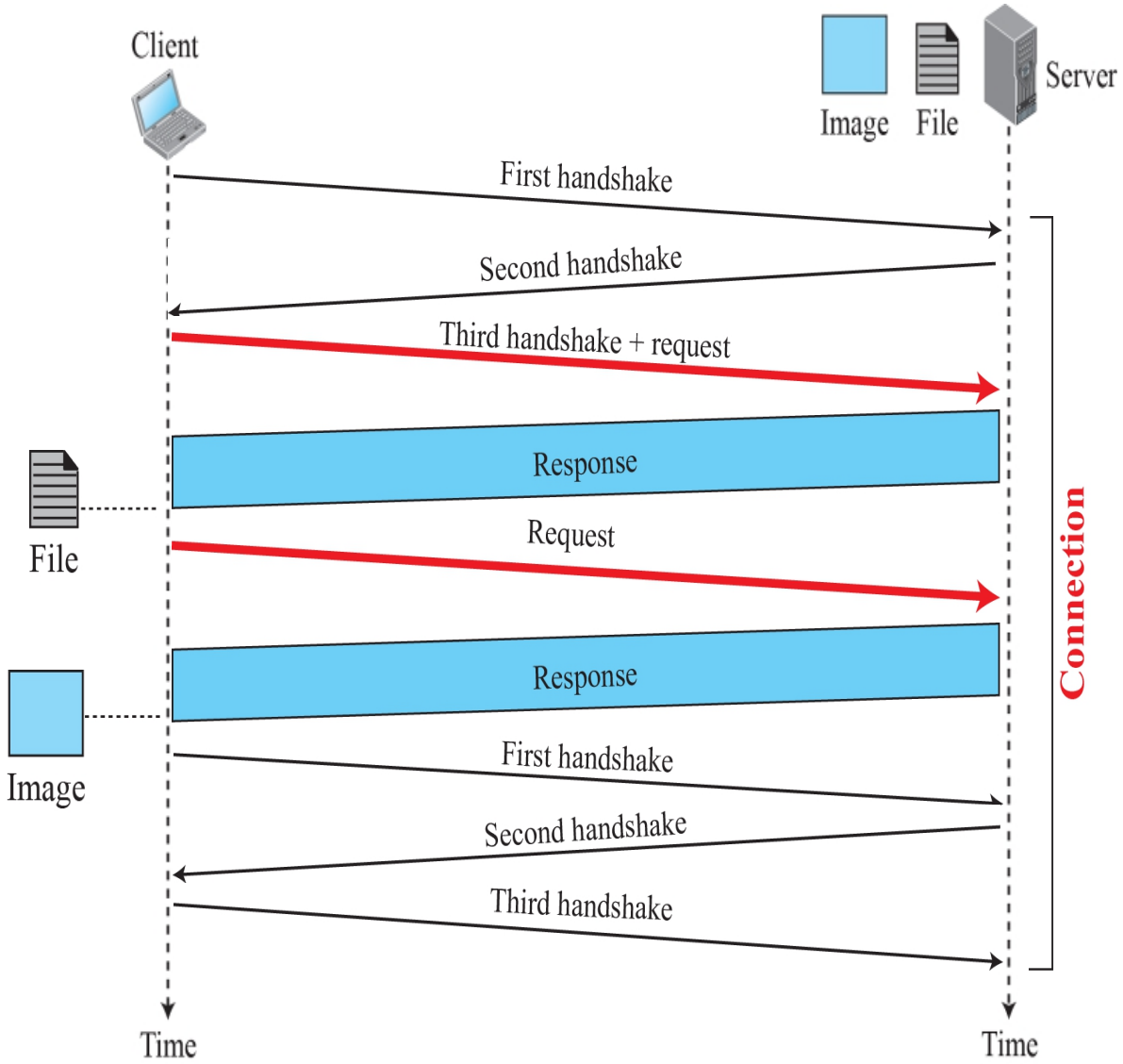
HTTP features:

- HTTP is **connectionless**: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server knows about each other during current request and response only. Further requests are made on new connection like client and server are new to each other.
- HTTP is **media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type.
- HTTP is **stateless:** As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

- HTTP versions: HTTP/0.9, HTTP/1.0, HTTP/1.1
- HTTP can use both **nonpersistent connections and persistent connections**. A nonpersistent connection is the one that is closed after the server sends the requested object to the client. In other words, the connection is used exactly for one request and one response.
- With persistent connections, the server leaves the TCP connection open after sending responses and hence the subsequent requests and responses between the same client and server can be sent. The server closes the connection only when it is not used for a certain configurable amount of time.
- Nonpersistent connections are the default mode for HTTP/1.0 and persistent connections are the default mode for HTTP/1.1. This means HTTP/1.0 uses a new connection for each request/response exchange, where as HTTP/1.1 connection may be used for one or more request/response exchanges.



nonpersistent connections



persistent connections

- For nonpersistent connection: The client needs to access a file that contains one link to an image. The text file and image are located on the same server. Here we need two connections. For each connection, TCP requires at least three handshake messages to establish the connection, but the request can be sent with the third one. After the connection is established, the object can be transferred. After receiving an object, another three handshake messages are needed to terminate the connection.
- For persistent connection: Only one connection establishment and connection termination is used, but the request for the image is sent separately.

Calculation of response time for persistent and non-persistent connection:

- **Round-trip time(RTT)**: Time for a small packet to travel from client to server and back.

  RTT= 2*propagation time

- Total response time = 2RTT+ transmit time
- **Example**: Assume that you have base HTML file with 30 embedded images, images & base file are small enough to fit in one TCP segment. How many RTT are required to retrieve base file & images for non-persistent and persistent connection.

Solution: For **Non-Persistent connection** :

Here for each image 2 RTT are required one for TCP connection and one for image to send.

So transmit time for 30 images = 2*(30 RTT) = 60 RTT

Total time = 2 RTT+60 RTT = 62RTT

For **Persistent connection** :

Here TCP connection is not required again and again.

So for 30 images it requires 30 RTTs

Total time = 2 RTT + 30 RTT = 32RTT

Formats of the request and response messages

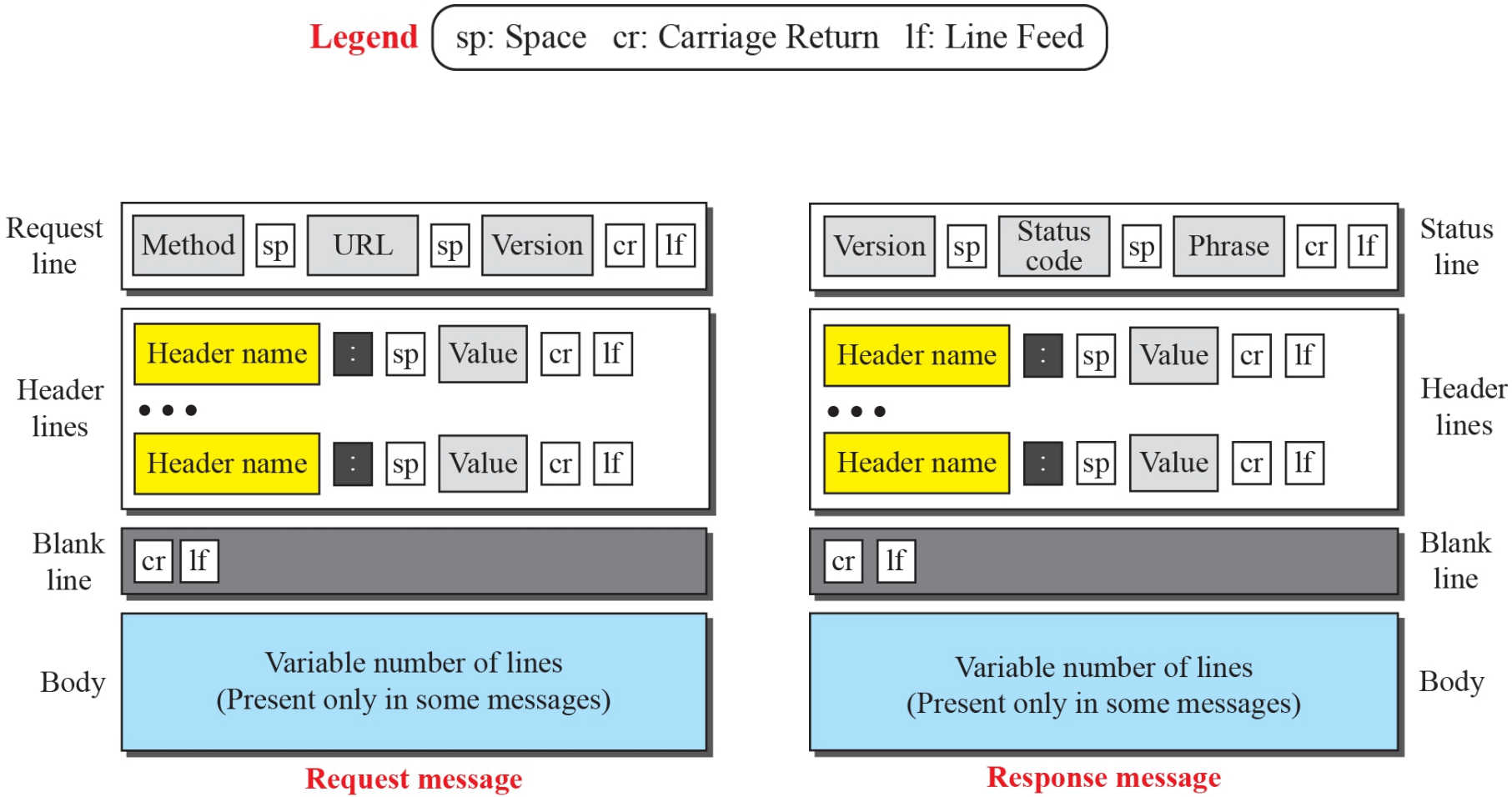


Table 2.1: Methods

Method	*Action*
GET	Requests a document from the server
HEAD	Requests information about a document but not the document itself
PUT	Sends a document from the client to the server
POST	Sends some information from the client to the server
TRACE	Echoes the incoming request
DELETE	Removes the web page
CONNECT	Reserved
OPTIONS	Inquires about available options

Table 2.2: Request Header Names

Header	*Description*
User-agent	Identifies the client program
Accept	Shows the media format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
Host	Shows the host and port number of the client
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Cookie	Returns the cookie to the server (explained later)
If-Modified-Since	If the file is modified since a specific date

Table 2.3: Response Header Names

Header	*Description*
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Server	Gives information about the server
Set-Cookie	The server asks the client to save a cookie
Content-Encoding	Specifies the encoding scheme
Content-Language	Specifies the language
Content-Length	Shows the length of the document
Content-Type	Specifies the media type
Location	To ask the client to send the request to another site
Accept-Ranges	The server will accept the requested byte-ranges
Last-modified	Gives the date and time of the last change

The status code field defines the status of the request. It consists of three digits.

When it is in 100 range => only informational

When it is in 200 range => a succesful request

When it is in 300 range => to another URL

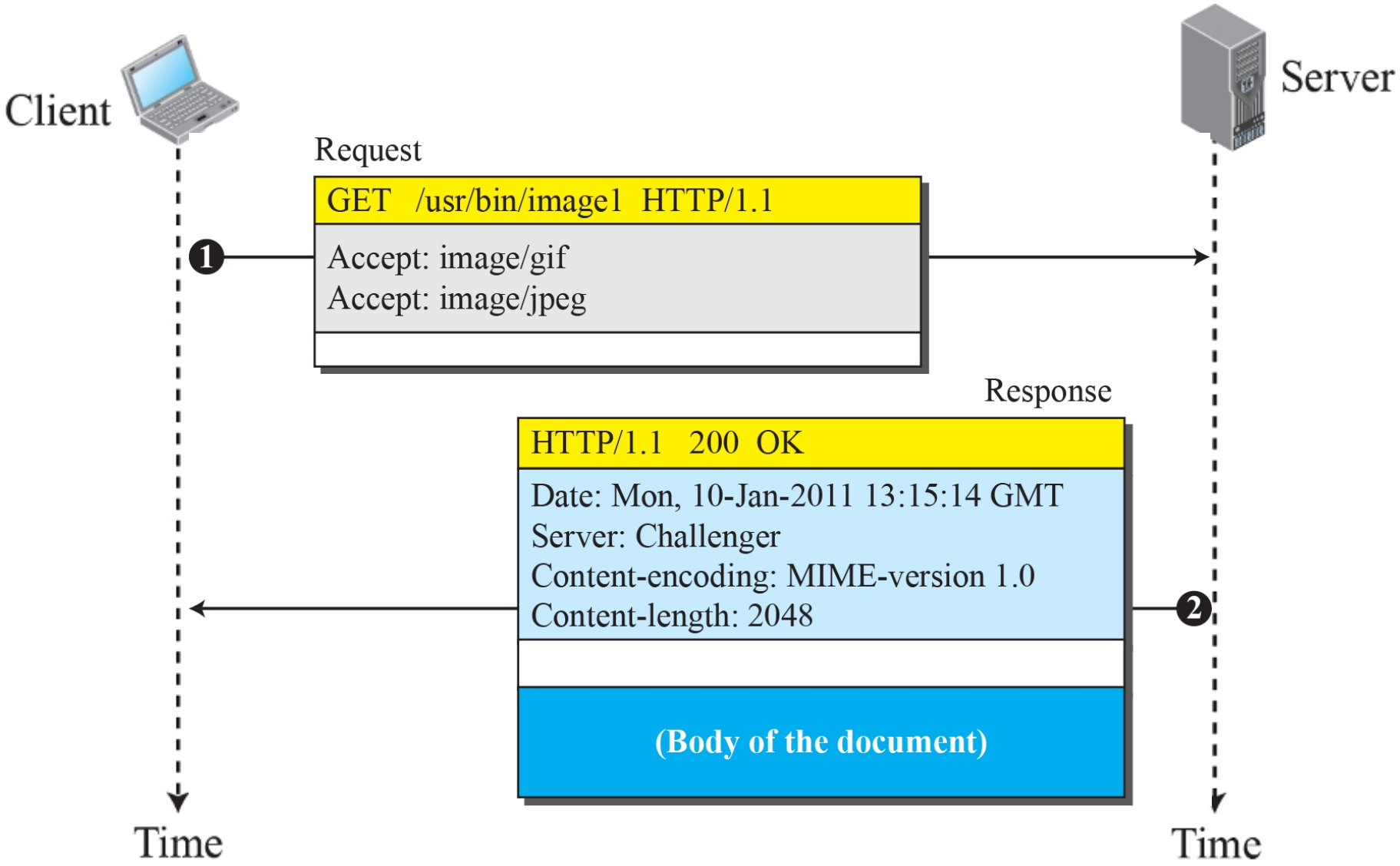
When it is in 400 range => an error at the client side

When it is in 500 range => an error at the server side

Example 2.6

This example retrieves a document (see next Figure). We use the GET method to retrieve an image with the path /usr/bin/image1. The request line shows the method (GET), the URL, and the HTTP version (1.1). The header has two lines that show that the client can accept images in the GIF or JPEG format. The request does not have a body. The response message contains the status line and four lines of header. The header lines define the date, server, content encoding (Multipurpose Internet Mail Extension(MIME) version, which will be described in electronic mail), and length of the document. The body of the document follows the header..

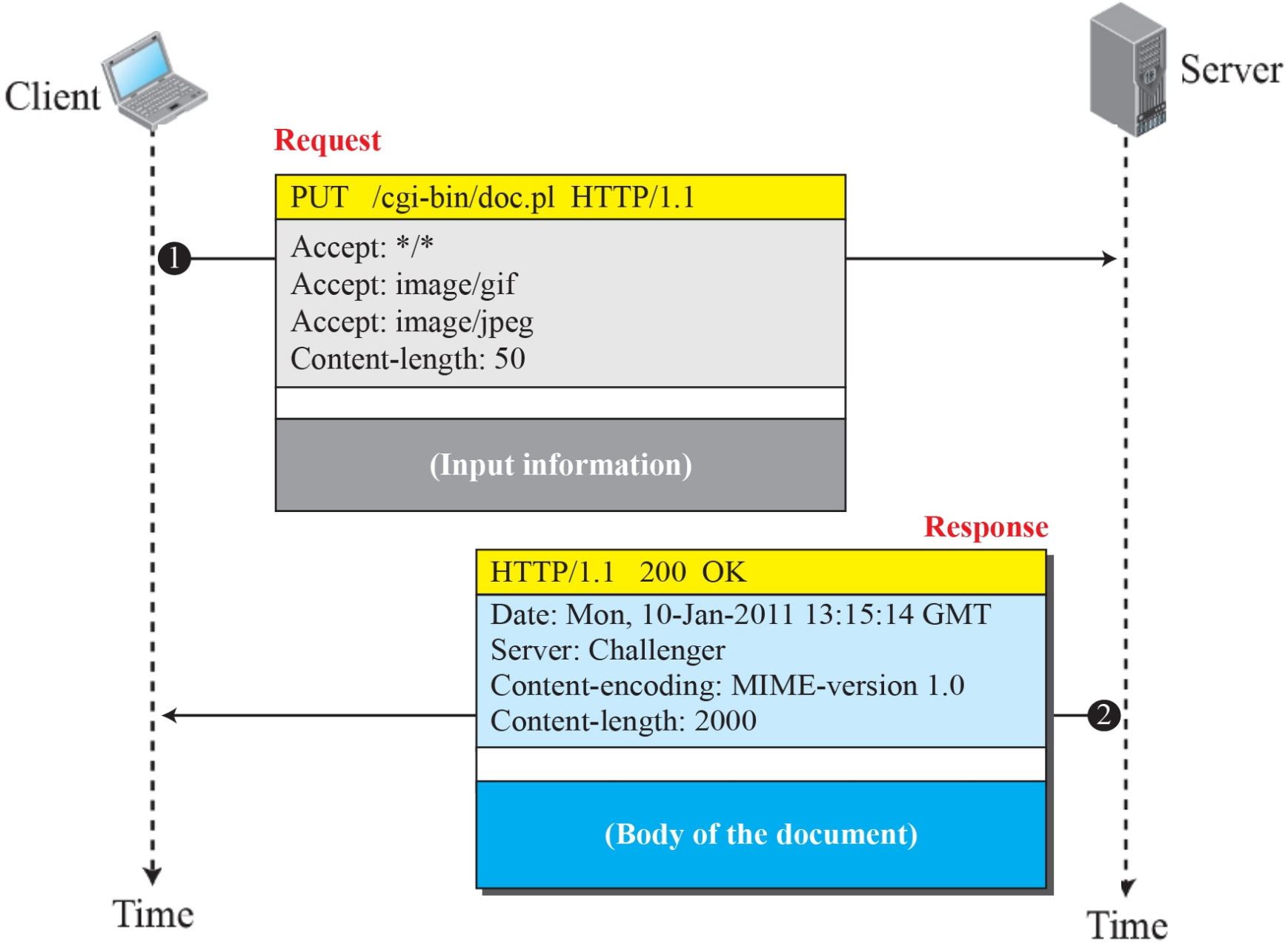
Figure : Example 2.6



Example 2.7

In this example, the client wants to send a web page to be posted on the server. We use the PUT method. The request line shows the method (PUT), URL, and HTTP version (1.1). There are four lines of headers. The request body contains the web page to be posted. The response message contains the status line and four lines of headers. The created document, which is a CGI(Common Gateway Interface) document, is included as the body (see next Figure).

Figure : Example 2.7



Example 2.8 (Conditional Request)

The following shows how a client imposes the modification data and time condition on a request.

GET http://www.commonServer.com/information/file1 HTTP/1.1	**Request line**
If-Modified-Since: Thu, Sept 04 00:00:00 GMT	**Header line**
	Blank line

The status line in the response shows the file was not modified after the defined point in time. The body of the response message is also empty.

HTTP/1.1 304 Not Modified	**Status line**
Date: Sat, Sept 06 08 16:22:46 GMT	**First header line**
Server: commonServer.com	**Second header line**
	Blank line
(Empty Body)	**Empty body**

Cookies:

- An HTTP cookie (also called web cookie, Internet cookie, browser cookie, or simply cookie) is a small piece of data stored on the user's computer by the web browser while browsing a website.
- Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past).
- They can also be used to remember pieces of information that the user previously entered into form fields, such as names, addresses, passwords, and payment card numbers.

Creating and Storing Cookies:

- After receiving an HTTP request, a **server can send one or more Cookie headers with the response**.
- The cookie is usually **stored by the browser in the cookie directoty,** which is sorted by the **server domain name.**

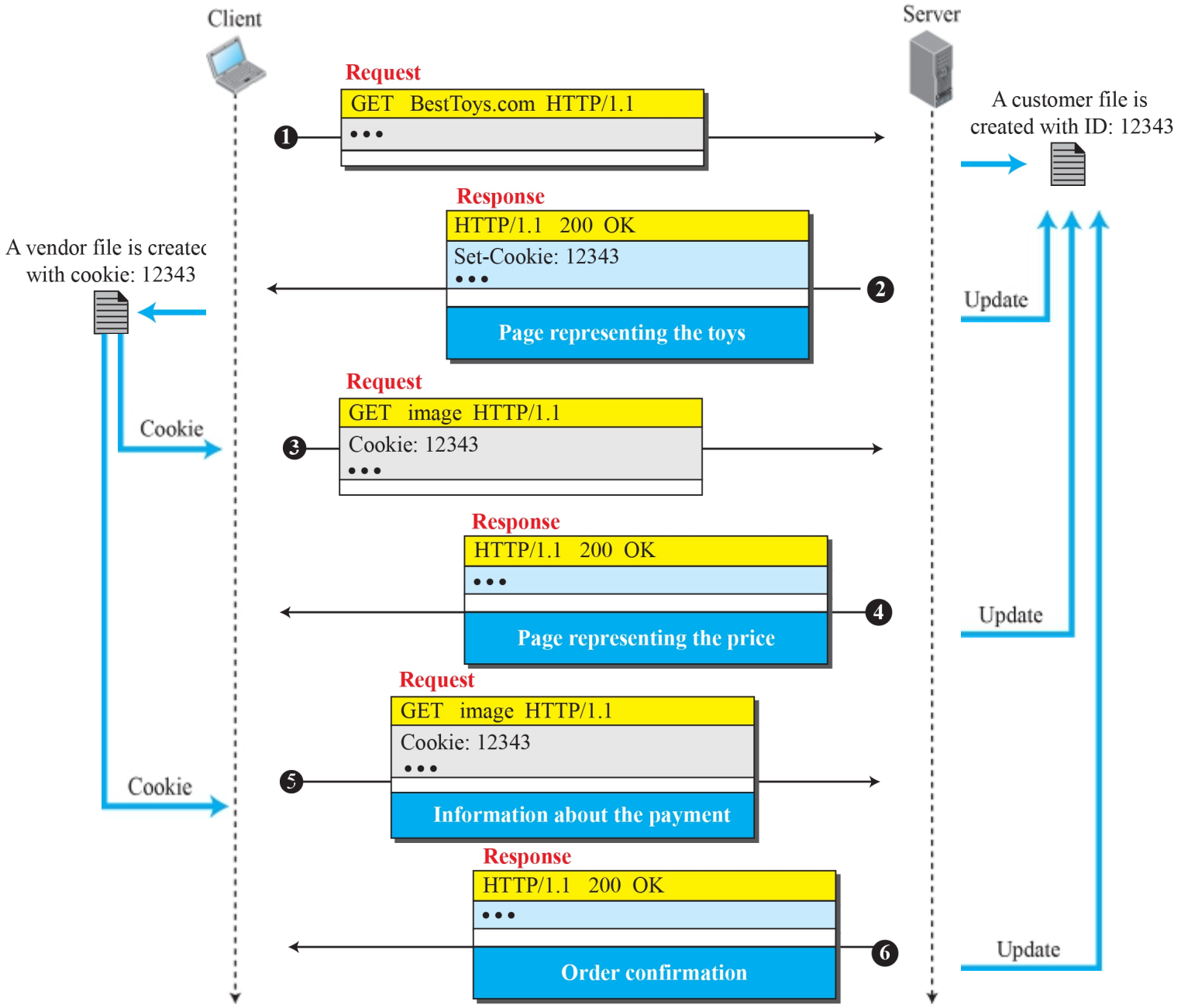
Using cookies:

- When a client sends a request to the server, the browser looks in the cookie directory to see if it can find a cookie sent by that server. If found, the cookie is included in the request.
- When the server receives the request, it comes to know that the client is an old client, not a new one.
- Note: The contents of the cookie are never read by the browser or disclosed to the user.

Example 2.9

Next Figure shows a scenario in which an electronic store can benefit from the use of cookies. Assume a shopper wants to buy a toy from an electronic store named BestToys. The shopper browser (client) sends a request to the BestToys server. The server creates an empty shopping cart (a list) for the client and assigns an ID to the cart (for example, 12343). The server then sends a response message, which contains the images of all toys available, with a link under each toy that selects the toy if it is being clicked. This response message also includes the Set-Cookie header line whose value is 12343. The client displays the images and stores the cookie value in a file named BestToys.

Figure : Example 2.9



Proxy server:

- HTTP supports proxy servers. A proxy server is a computer that keeps copies of responses to recent requests.
- The HTTP client sends a request to the proxy server. The proxy server checks it's cache. If the response is not stored there, the proxy server sends the request to the corresponding server.
- Incoming responses are sent to the proxy server and stored for future requests from other clients.
- The proxy server acts as both server and client.
- It is always located at the client side.

Advantages of Proxy Server:

- Reduces the load on the original server
- Decreases traffic
- Improves latency

Cache Update:

- An important question is **how long a response should ramain in a proxy server before being deleted or replaced**.
- One solution is to store the list of sites whose information remains the same for a while.
- Another solution is to add some headers to show the last modification time of the information.

Example 2.10

Next Figure shows an example of a use of a proxy server in a local network, such as the network on a campus or in a company. The proxy server is installed in the local network. When an HTTP request is created by any of the clients (browsers), the request is first directed to the proxy server If the proxy server already has the corresponding web page, it sends the response to the client. Otherwise, the proxy server acts as a client and sends the request to the web server in the Internet. When the response is returned, the proxy server makes a copy and stores it in its cache before sending it to the requesting client.

Figure : Example of a proxy server