Software Construction and User Interfaces (SE/ComS 319)

Ali Jannesari

Department of Computer Science

Iowa State University, Spring 2019

BASICS

Outline

- Threads
  - Why threads are needed?
  - Threads vs Processes
  - Threads in Java
  - Issues with threads and concurrency
    - How to handle it?
  - Know how to write simple Java programs using threads
- Server and client
- Web server and clients
- HTTP Protocol

PROCESS AND THREADS

Questions (1)

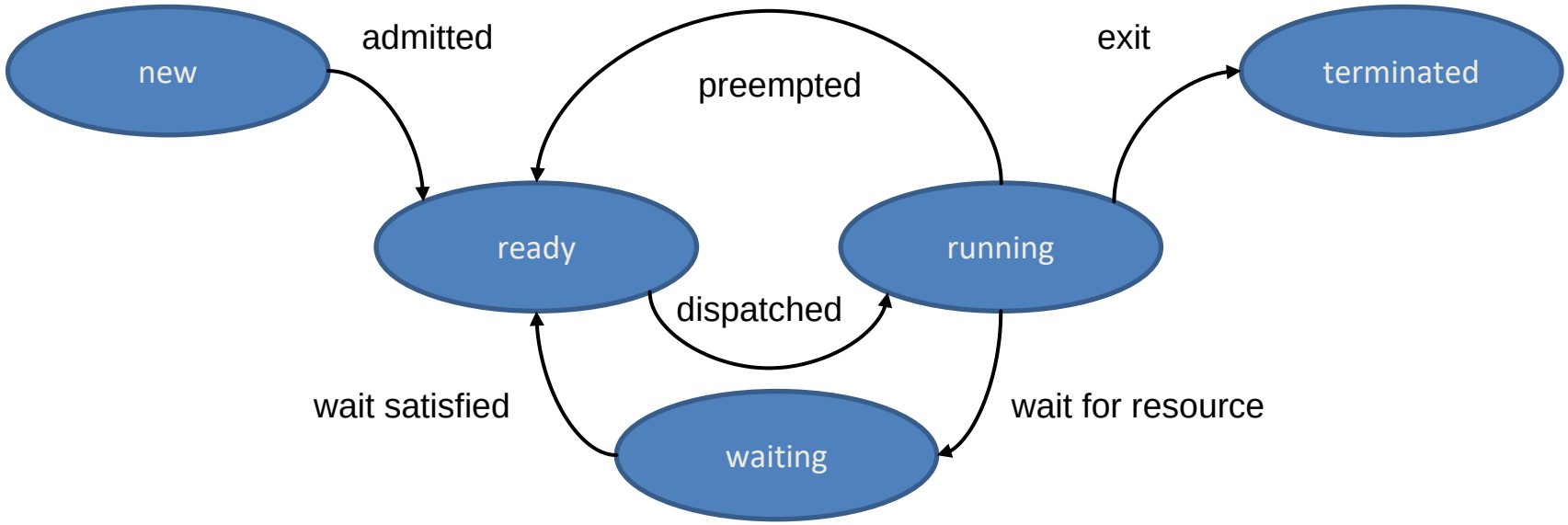
1. What are some reasons that a program cannot execute an instruction immediately?

2. What's a process?

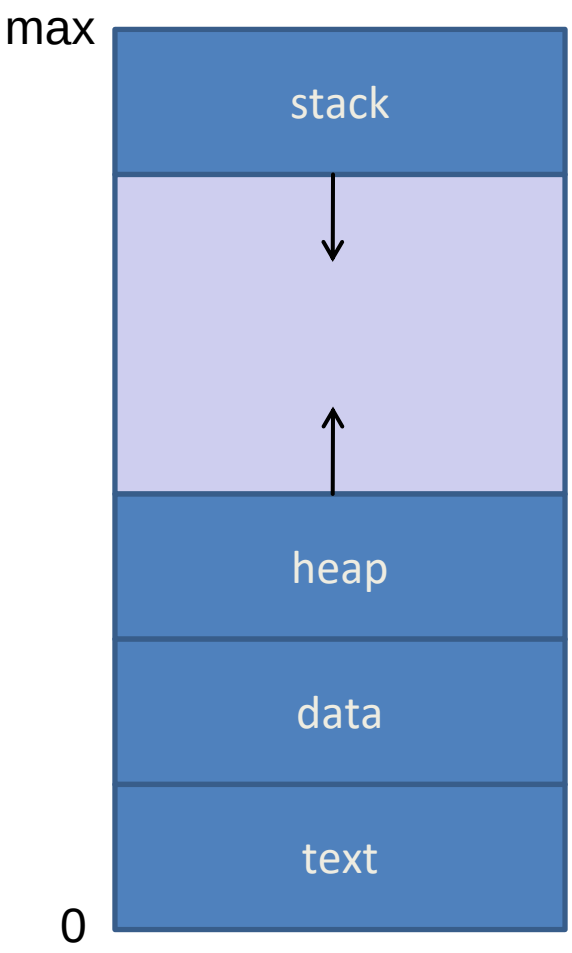
3. How do so many processes execute at the same time?

Process

- A process is a program in execution
- States of a process



Processes in memory



Temporary data: function parameters, return addresses, local variables

Dynamically allocated memory

Global variables

Program code

Questions (2)

1. What's a thread?

2. What are differences between a process and a thread?

3. How are mechanisms for creating threads in Java?

Motivation for threads

- One of the major concepts we learned in programming is that program execution proceeds step by step. Until a step is **completed**, the next step **cannot be started**.

- Consider the following scenarios:
  - Scenario (1): Printing a large file
  - Scenario (2): Loading a web page

Scenario (1): Printing a large file

- You are editing a large file. You ask the document editor to print it for you. While it is being printed, the editor does not allow you to continue editing your file (as the printing needs to be completed first).

- Nowadays, document editors do print jobs in the background - allowing you to continue editing.

Scenario (2): Loading a web page

- You just navigated to a web page and there are many pictures and videos on the web page.

- You have a slow internet connection and have to wait a long while before you can start reading the information on the web page.

- Most good web pages will load text first and allow to start reading while it continues to upload the pictures/videos in the background.

Threads

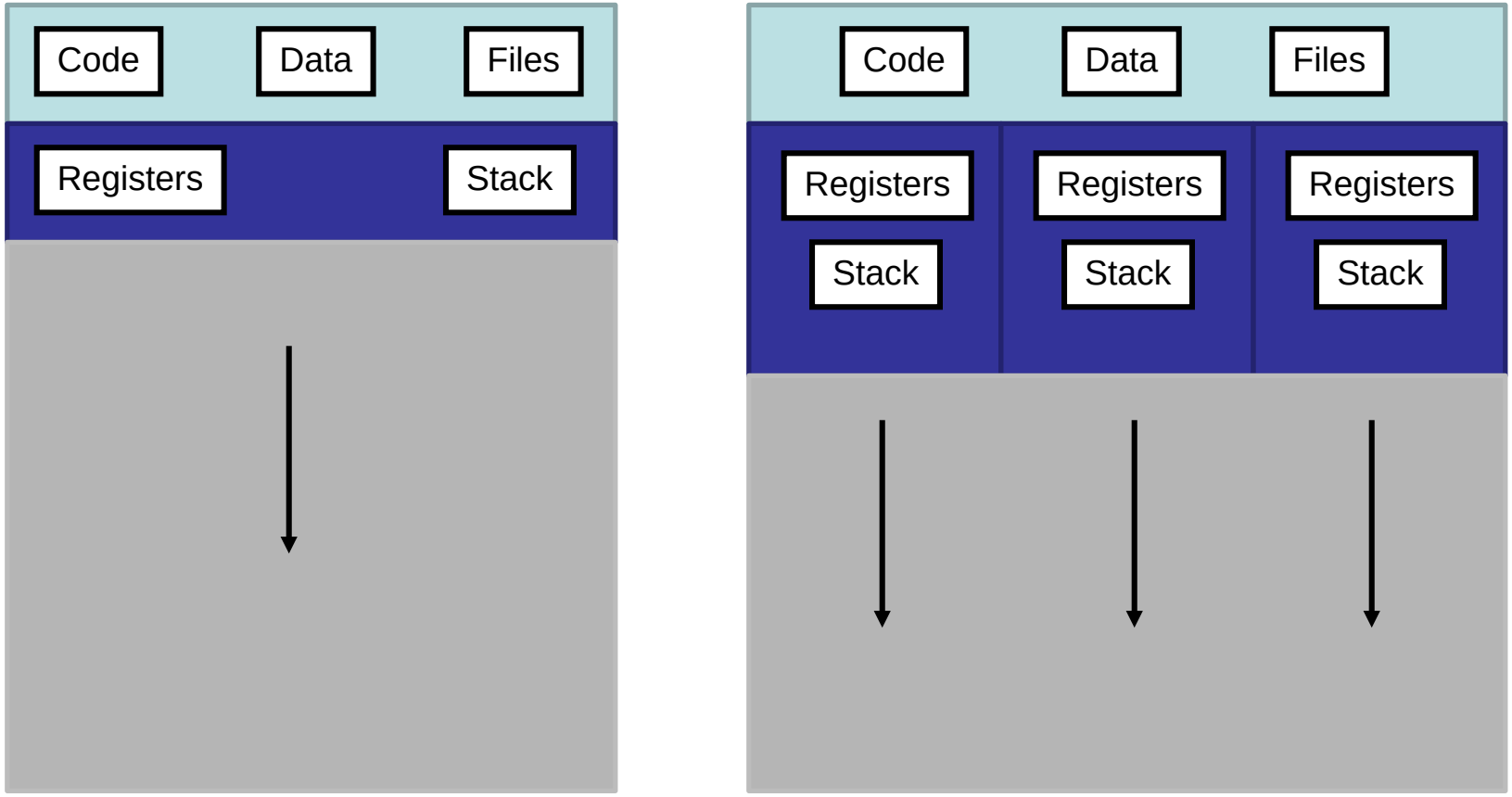
- In both the previous scenarios, we want more than one set of tasks to be going on **at the same time (i.e. concurrently).**
- Typically, there are many programs on a computer and these can be executed by the computer at the same time. Executing programs are called PROCESSES.

- However, it is also possible to have **separate sets of tasks** being **concurrently** executed in the **SAME PROGRAM**. These are called THREADS.

Threads (2)

- Basic unit of CPU utilization
  - Flow of control within a process
- A thread includes
  - Thread ID
  - Program counter
  - Register set
  - Stack
- Shares resources with other threads belonging to the same process
  - Text (i.e., code) section
  - Data section (i.e., address space)
  - Other operating system resources
  - E.g., open files, signals

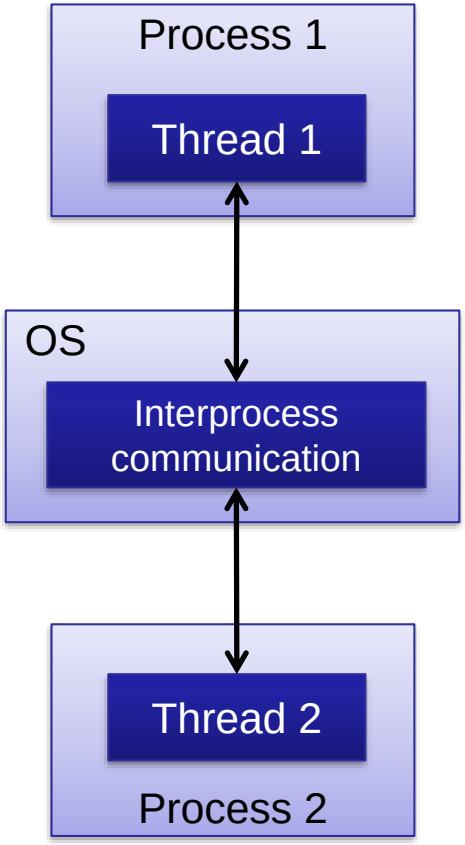
Single-threaded vs. multi-threaded

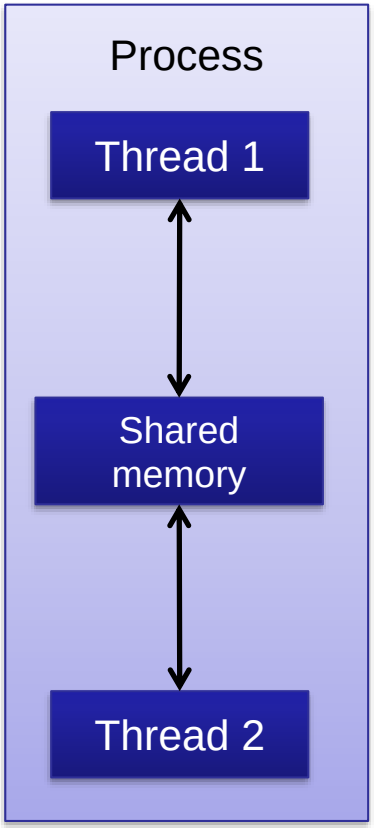


Single-threaded

Multi-threaded

Processes vs. threads





Questions (3)

1. What are some issues with threads (or concurrency in general)?

2. What is a mechanism provided in Java to handle these issues?

Threads in Java

- Java provides TWO ways of creating threads.
- First way: extend Thread class and implement the run method
- Second way: implement Runnable class, create Thread object, and implement the run method

```
public class SimpleThread1 extends Thread {

        String name;

        public SimpleThread1(String s) { name = s; }

        public void run() {
          for (int i = 1; i < 1000; i++) {
                System.out.println("This is thread " + name);
          }
        }

        public static void main(String[] args) {

                Thread firstThread = new SimpleThread1("T1");
                Thread secondThread = new SimpleThread1("T2");

                firstThread.start();
                secondThread.start();
        }
}
```

```
public class SimpleThread2 implements Runnable {

        String name;

        public SimpleThread2(String s) { name = s; }

        public void run() {
          for (int i = 1; i < 1000; i++) {
                    System.out.println("This is thread " + name);
          }
        }

        public static void main(String[] args) {

                Thread firstThread = new Thread( new SimpleThread2("T1"));
                Thread secondThread = new Thread( new SimpleThread2("T2"));

                firstThread.start();
                secondThread.start();
        }

}
```

Notes

- In the previous examples, there were actually **THREE** user threads: the main thread (which stops when the main method is done), the firstThread, and the secondThread.

- When you run the example, you will see that the results from the threads are interleaved. Each thread prints when it gets the time-slice – but stops when it is put back in the ready queue by the scheduler.

What to know about threads!

- Thread lifecycle, Thread states
- Thread methods
- Issues in data sharing (synchronized statement)
- Synchronization between threads (wait/notify)
- Problems in using threading (deadlock, data races)
- We will briefly go over thread states and methods but will skip the other parts now (later sessions).

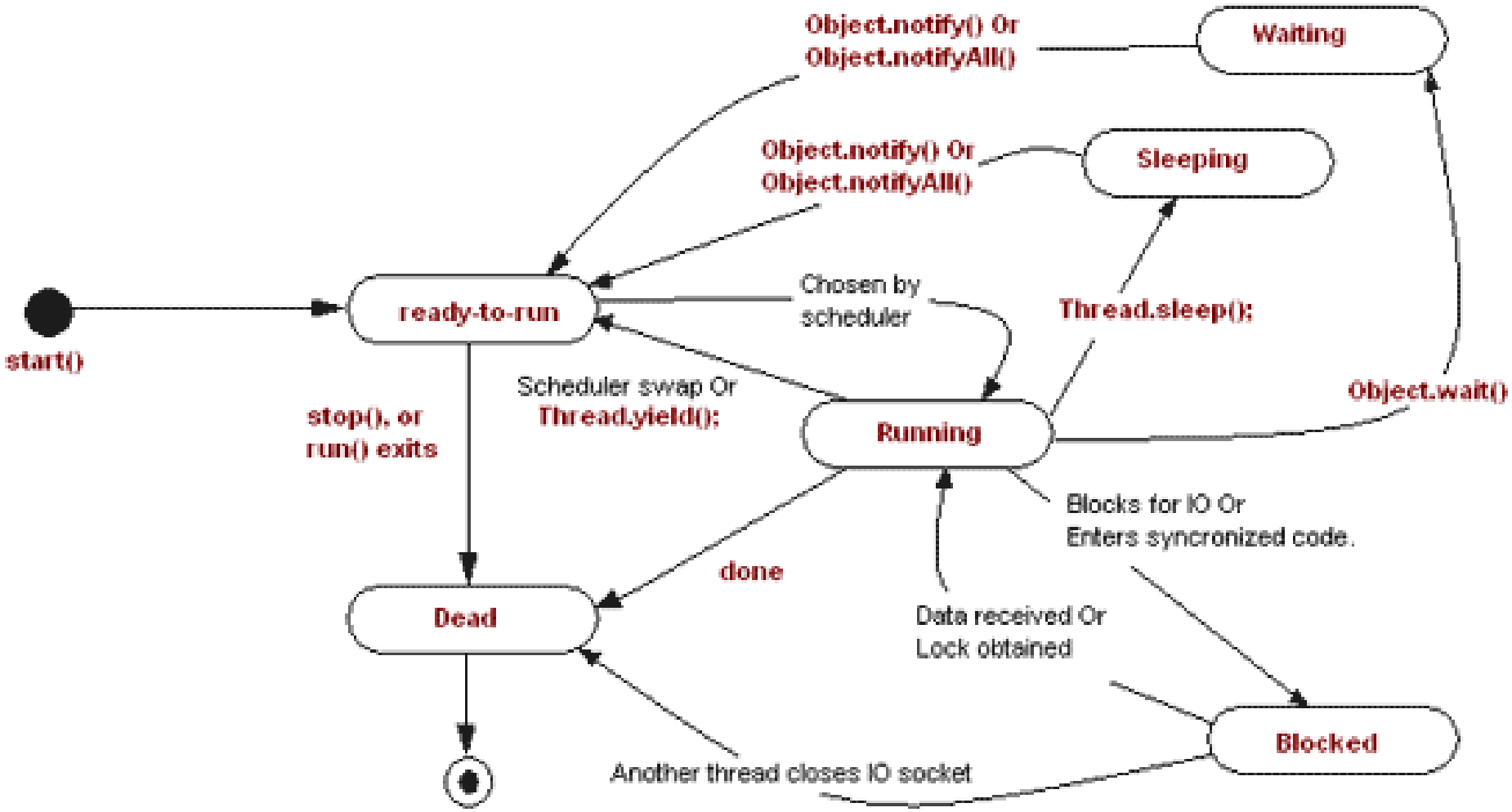
Thread states

- **created**: When we do a new Thread(), the thread has been created.
- **ready-to-run**: When we invoke the start method on the thread object, the scheduler puts the thread in a "ready" Queue. The scheduler lets the first thread on this queue to run for some time, and then the next one on the queue and so on.
- **running**: The run method of the thread object is started. When the thread's time-slice is over, the thread is put back in the ready queue. It can continue with the run method, when it is allowed to execute again by the scheduler.

Thread states

- dead: When the run method is completed, the thread is considered to be done executing and in the dead state.
  - Some threads never die (as their run methods are never-ending loops).
- For now, we will ignore the sleep, waiting, and blocked states.

Thread lifecycle/methods



Thread methods

Some useful thread methods for now are:

- **start():** this puts the thread in the ready queue.
- **yield()**: this allows the thread to give up its time-slice and put back at the end of the ready queue.
- **sleep(x)**: allows the thread to sleep for x milliseconds before being woken up to run again.

Parallelism vs. concurrency
(often used interchangeably)

Parallelism

- Parallel processing of subtasks for the purpose of speedup
- Requires **parallel hardware**

Concurrency

- Overlapping but potentially unrelated activities
- Access to shared resources possible
- Does not require parallel hardware

Concurrent programming

Separation of concerns

- Group related code
- Keep unrelated code apart
- Examples
  - Waiting for input vs. processing input in interactive applications
  - Waiting for requests vs. processing requests in server
  - Background tasks such as monitoring the file system for changes in desktop applications
- Beneficial even when using a single CPU

Web browser

DVD player

Summary

- You can either extend Thread or implement Runnable to create Java Threads.

- Threads can be started by "start" method.
- Then, their run method starts executing. Threads get switched out by the scheduler.

- Multiple threads can be running at the same time.
- Parallelism vs. concurrency

Literature – Threads

- There is a lot to know about threads.
  - More in sessions for parallel software construction
- Good resource:
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

- COMS430 --- good class to take (about concurrency)
- COMS 527x : Concurrent systems (graduate course)

SERVER AND CLIENT TERMS AND DEFINITIONS

Server

- Program that **provides SERVICES** (i.e. useful functionality). Typically, keeps running forever.
  - Typically, *the computer* that runs the server program is called a server. Many server programs can be running on a single computer.
- Examples of services:
  - database services
  - runs a web site

Client

- Program that connects to a SERVER computer - and then to a program that provides services and **USES those services**.
  - Typically, the computer that runs the client program is called a client!
- Multiple clients can typically connect to a server
- Examples:
  - Web browsers on a computer connect to web servers on other computers and is provided with web pages.

MAC ID – IP Address – Hostname

- **MAC ID** is a unique id that is HARD-CODED on every computer (or internet capable device).
  - Already there when you buy the device.
  - Example: c8:bc:c8:9b:c4:0f for ethernet card of a computer.
  - Used by lower protocols to uniquely identify a device.
- **IP ADDRESS** is an address assigned to **computers connected to the internet.**
  - Typically assigned when connecting to the internet.
  - Example: 129.186.252.23
- Unlike IP address, **HOSTNAME** is a human-readable address (like www.google.com). Servers typically have hostnames.

DNS – Localhost

- **DNS (Domain Name Server) –** is like a phone book.
  - Maps Hostnames to IP addresses (many to one)
  - When you want to connect to a website by typing in a hostname, your computer will find the IP address by asking the DNS.

- **Localhost –** each computer can use the hostname *localhost* to refer to itself!

Port

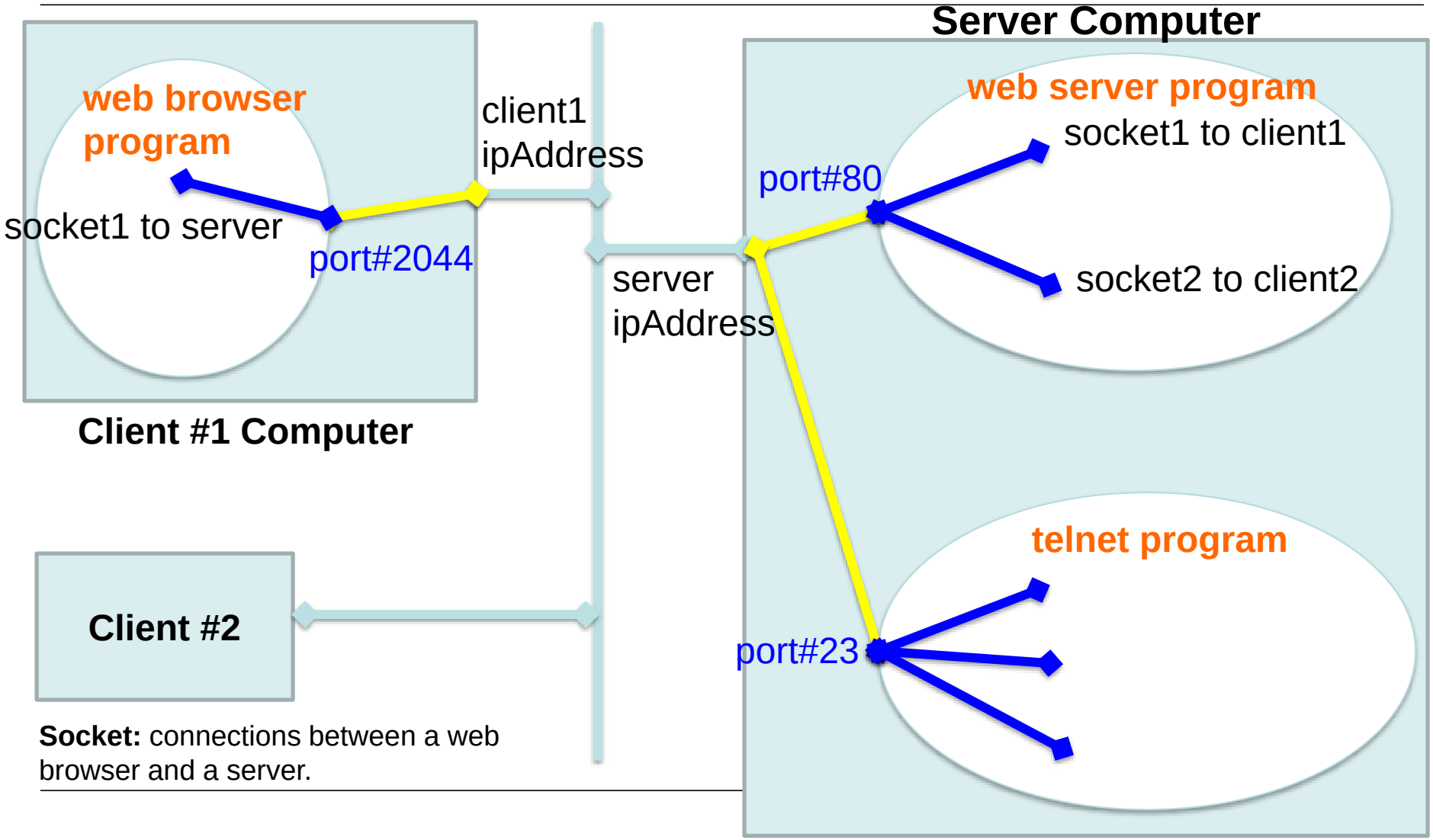
- This is a **NUMBER** that refers to a **specific process** running on a computer.
- Many port numbers are reserved:
  - 80: http
  - 23: telnet (communication for computers on internet/local area networks)
  - 22: ssh
  - 110: pop3 (for email delivery)
- You will be able to create ports only from 1024 onwards.
- Once a port is being **used by a server**, you cannot use that same port for other programs.
- Multiple clients can talk to a server through that port.

Socket

- A socket contains connection information between two computers
- LOCAL ADDRESS
  - local computer's IP address
  - local program's port#
- REMOTE ADDRESS
  - remote computer's IP address
  - remote program's port#
- PROTOCOL
  - means the "LANGUAGE" or "RULES" that the two computers will use to communicate.
  - typically this is TCP/IP protocol.

Socket = host name (IP) and port number

Server and client connection



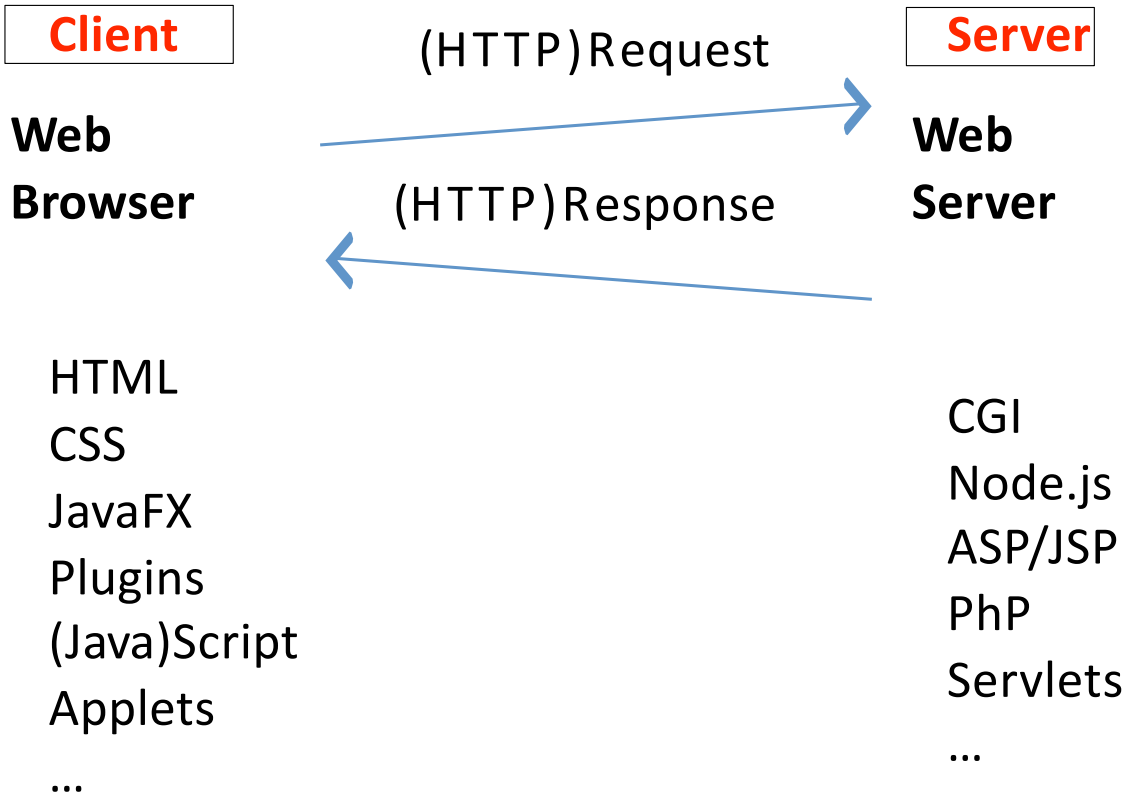
Socket: connections between a web browser and a server.

Demonstration

- Telnet is disabled on most servers for security reasons
- Server @ CS Dep ISU
  - To login to pyrite, you must use the Secure Shell (ssh)
  - ssh <ISU NetID>@pyrite.cs.iastate.edu
  - **Ctrl - d** usually allows you to exit the ssh session normally (logout)
- Condo Cluster @ ISU (High-performance computing cluster)
  - ssh <ISU NetID>@condo2017.its.iastate.edu

WEB SERVER AND CLIENTS

Web server and client

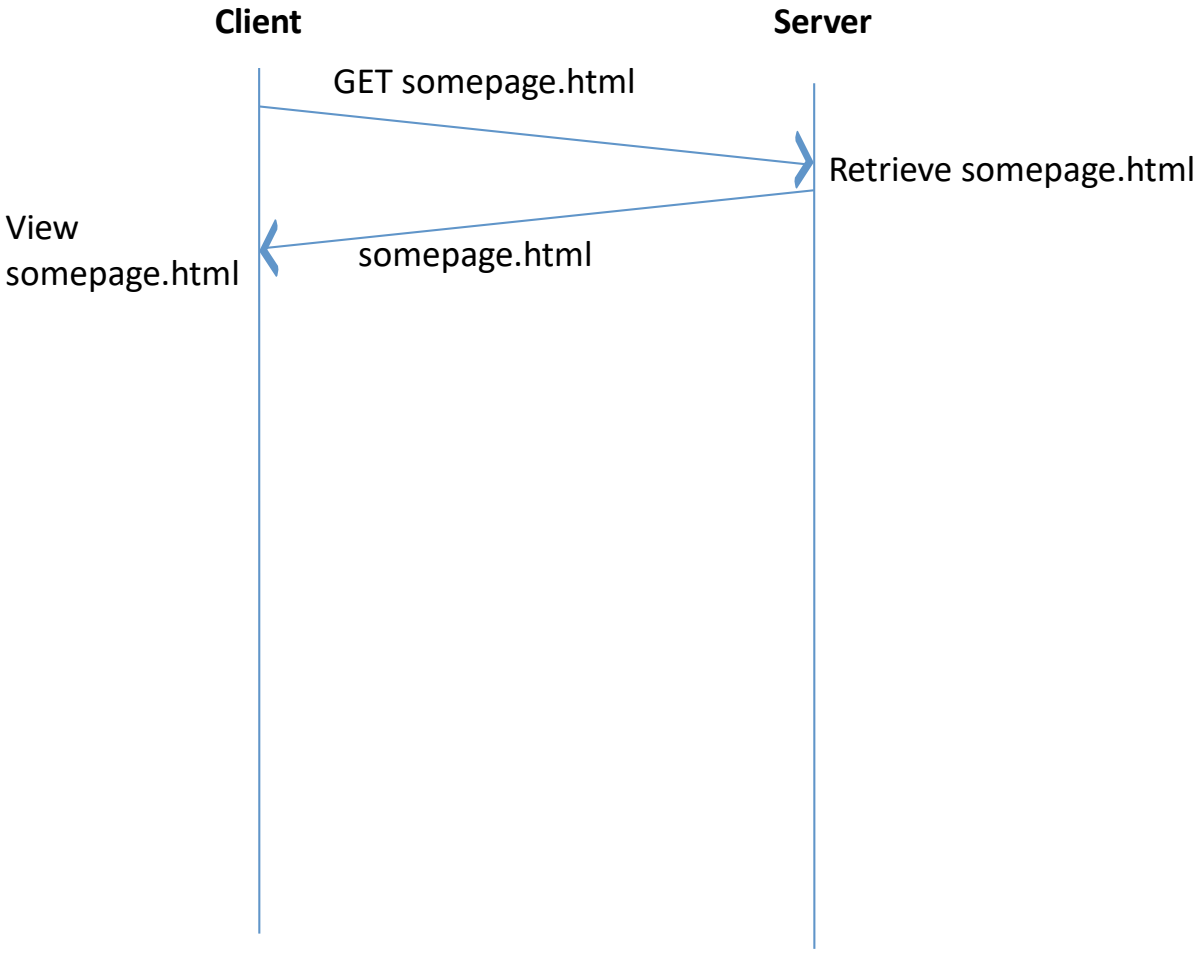


HTTP (Hypertext Transfer Protocol): HTTP is a client-server application-level protocol. It typically runs over a TCP/IP connection.

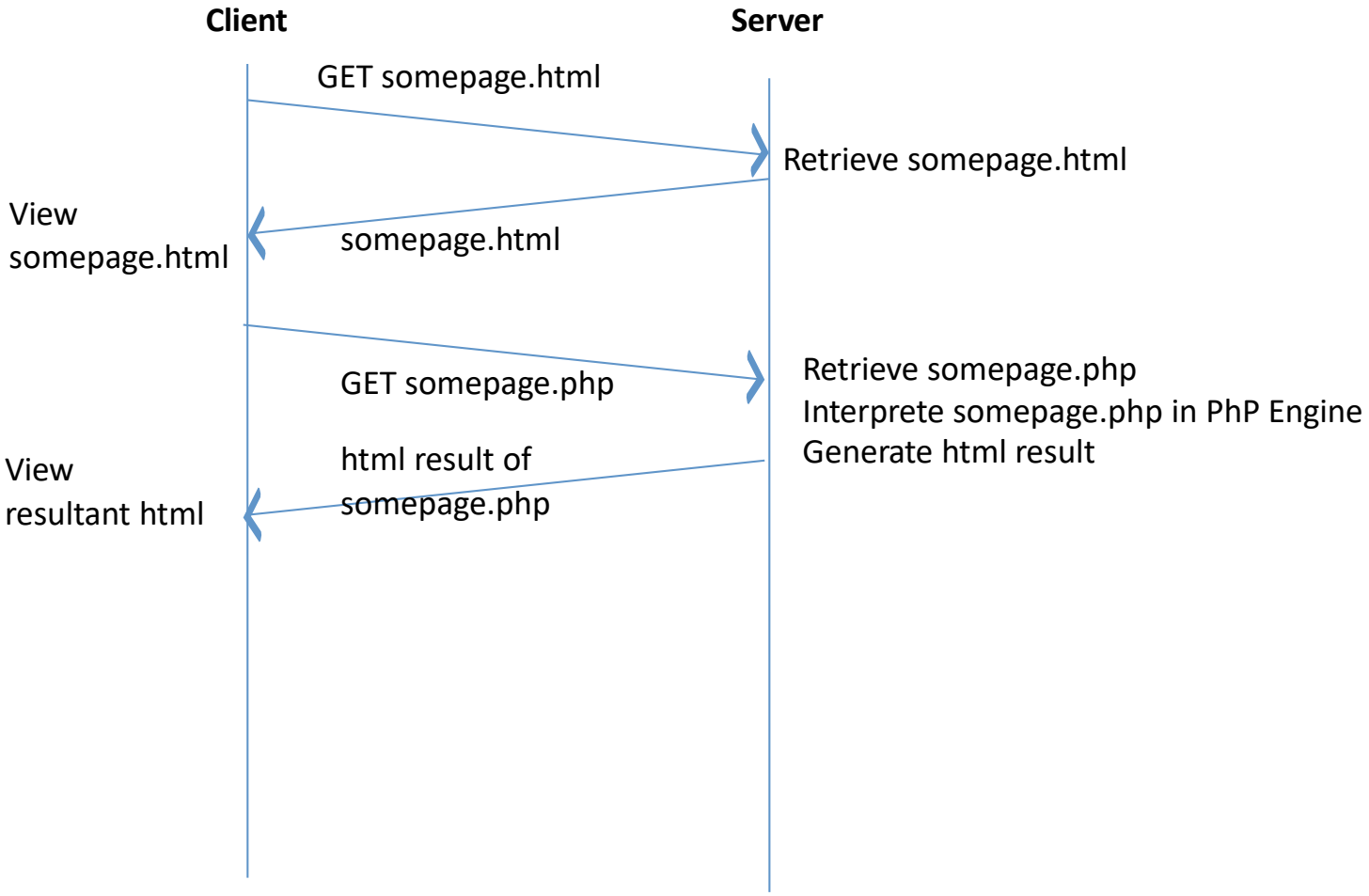
Web server and client – HTTP request

- Web-client and Web-server communicates using HTTP protocol
  - Client can send a HTTP request: method “`get`" or “`post`"
  - Server can read a HTTP request and produce HTTP response
- Server side programs should be capable of reading HTTP request and producing HTTP response
- Command: **GET** *request-URI HTTP-version*
  - e.g. **GET /index.html HTTP/1.0**

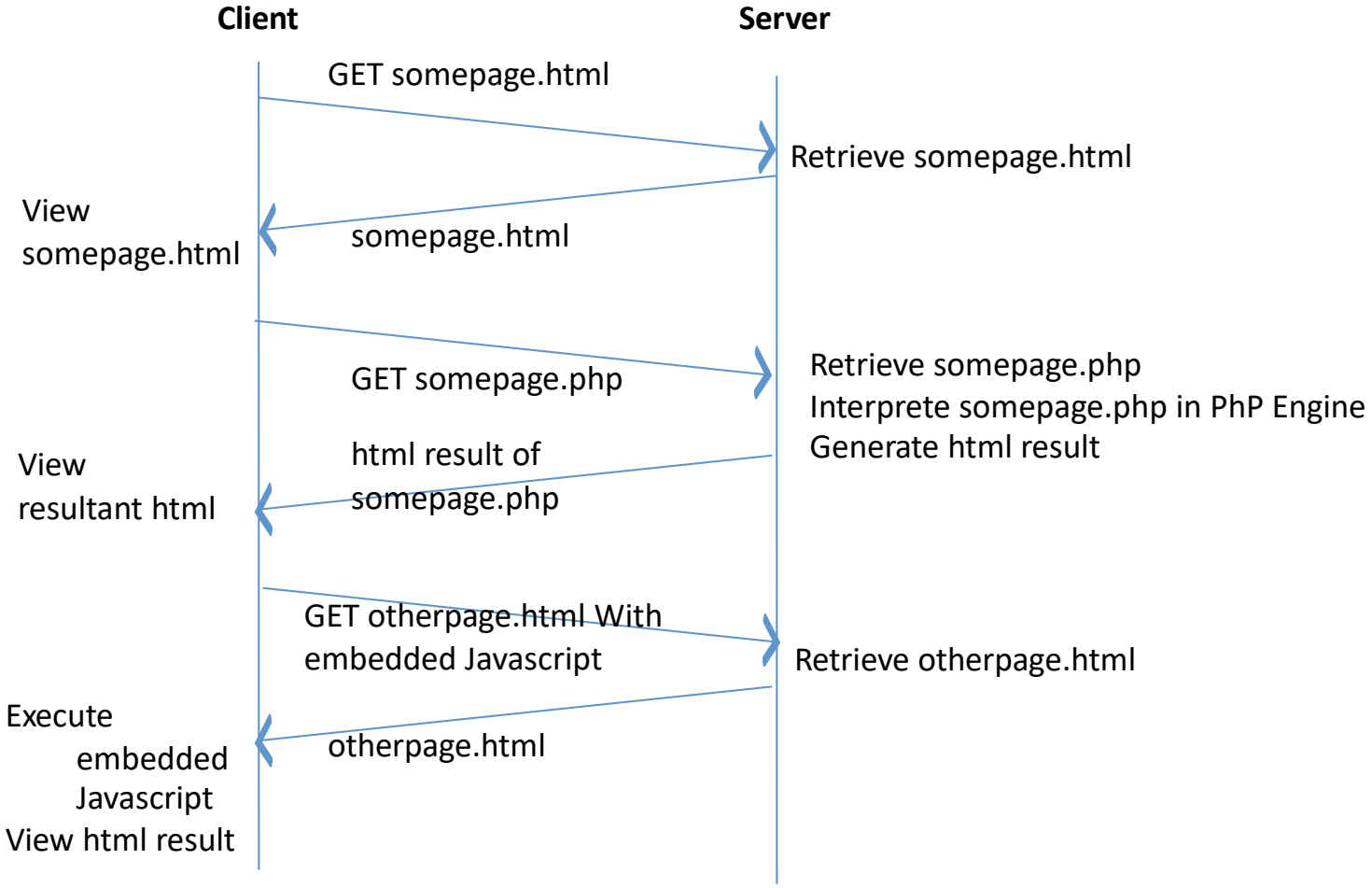
Web server and client interaction (1)



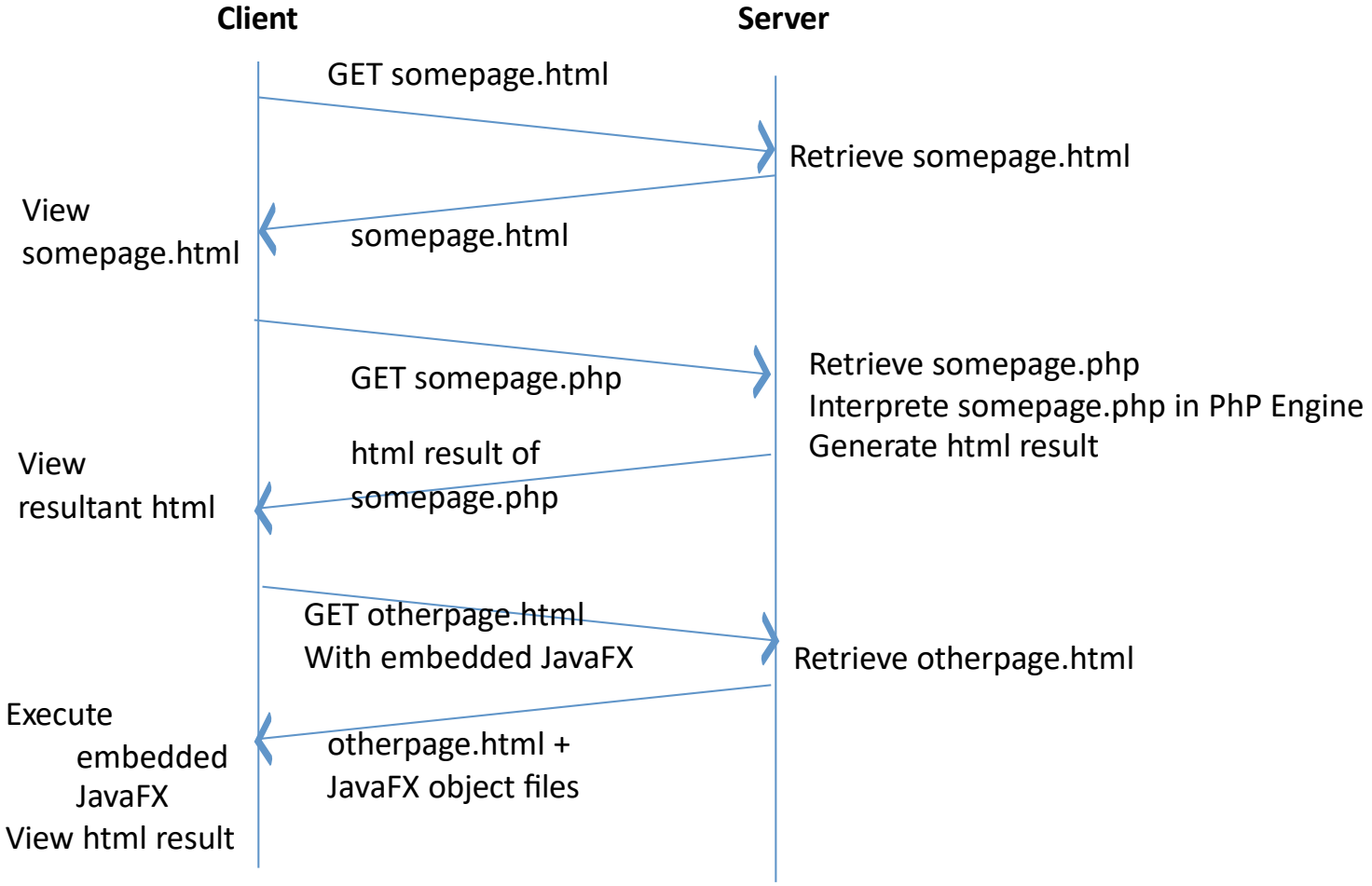
Web server and client interaction (2)



Web server and client interaction (3)

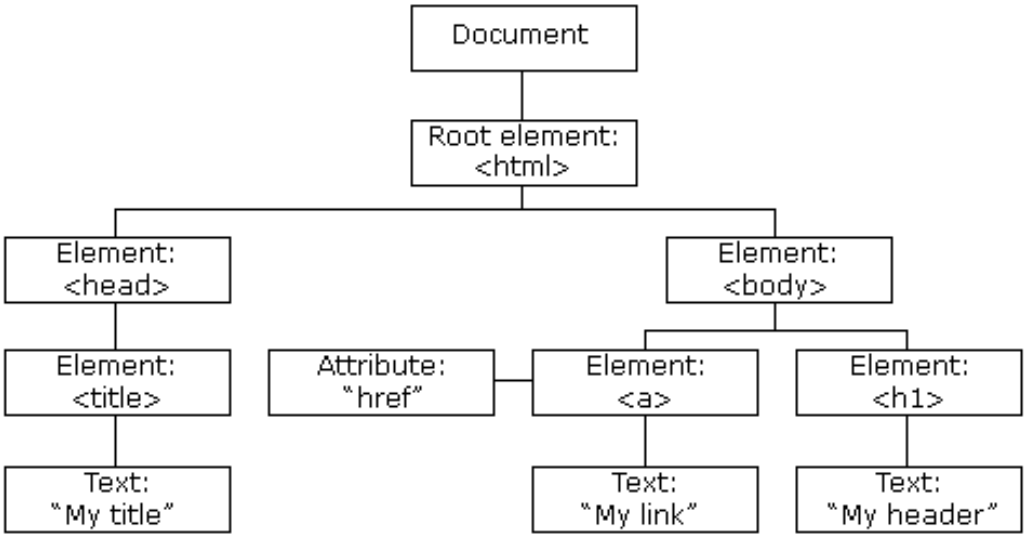


Web server and client interaction (4)



Demonstration

- HTML
- DOM (Document Object Model) : An application programming interface (API) for HTML
  - It defines the logical structure of documents and the way a document is accessed and manipulated:
  - https://www.w3schools.com/



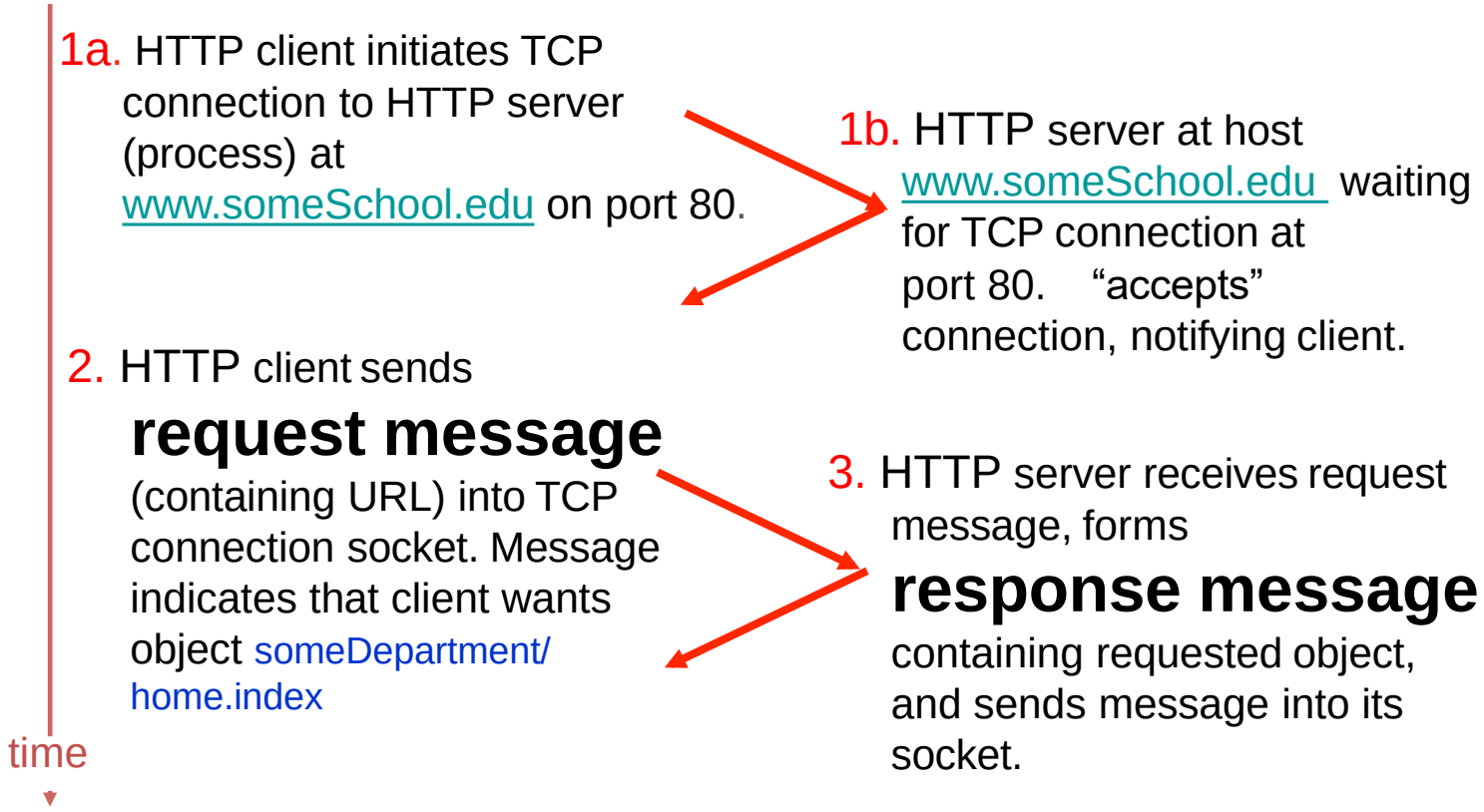
Source: https://www.w3.org

- HTML, DOM, etc. in Safari/Chrome

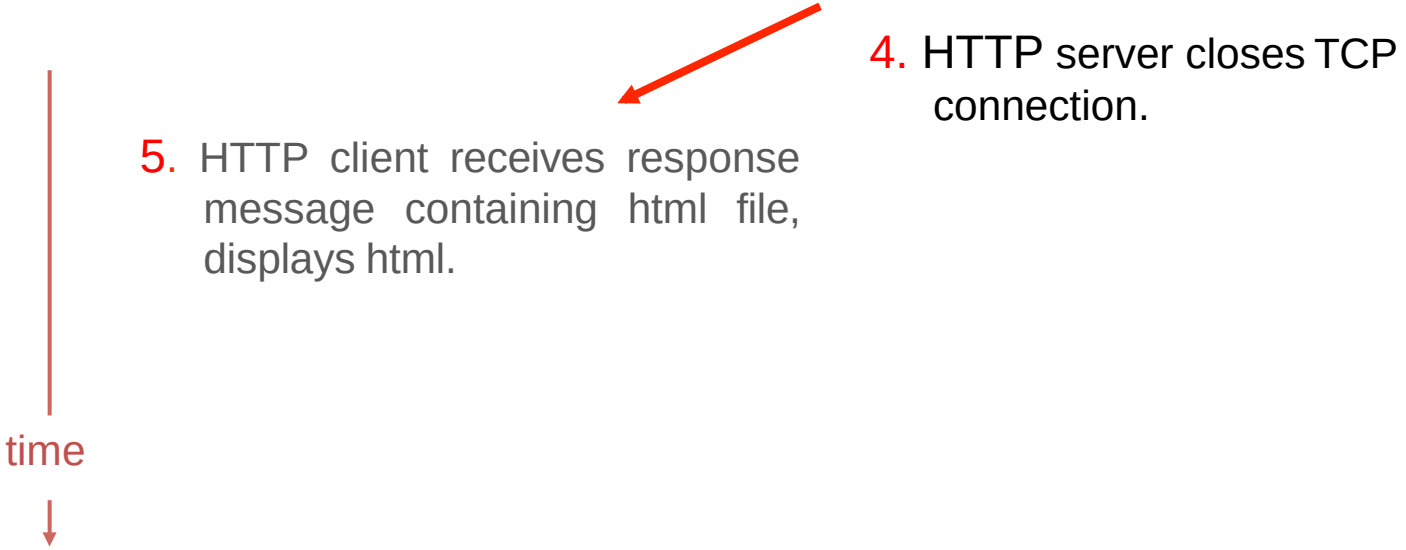
HTTP PROTOCOL

Normal HTTP requests (1)

Suppose user enters URL:
www.someSchool.edu/someDepartment/home.index



Normal HTTP requests (2)



- HTTP works as a **request-response protocol** between a client and server
- A web browser may be the client, and an application on a computer that hosts a web site may be the server

HTTP request message

FORMAT

method URL version<cr><lf>

header1:value<cr><lf>

header2:value<cr><lf>

…

headerN:value<cr><lf>

<cr><lf>

BODY OF HTTP REQUEST

EXAMPLE

GET /somedir/page.html HTTP/1.1

Host: www.someschool.edu

User-agent: Mozilla/4.0

Connection: close

Accept-language: fr

(extra carriage return, line feed)

HTTP request message – GET

EXAMPLE OF GET METHOD:

```
GET /form.php?username=Joe HTTP/1.1

Host: www.cs.iastate.edu

<CRLF>
```

- The query string (name/value pairs) is sent in the URL of a GET request
- GET passes arguments on URL
- Data limited

- GET is used to request data from a specified resource
- one of the most common HTTP methods
- length restrictions
- only used to request data (not modify)
- Data is visible to everyone in the URL (not safe)
- Only ASCII characters allowed

HTTP request message – POST

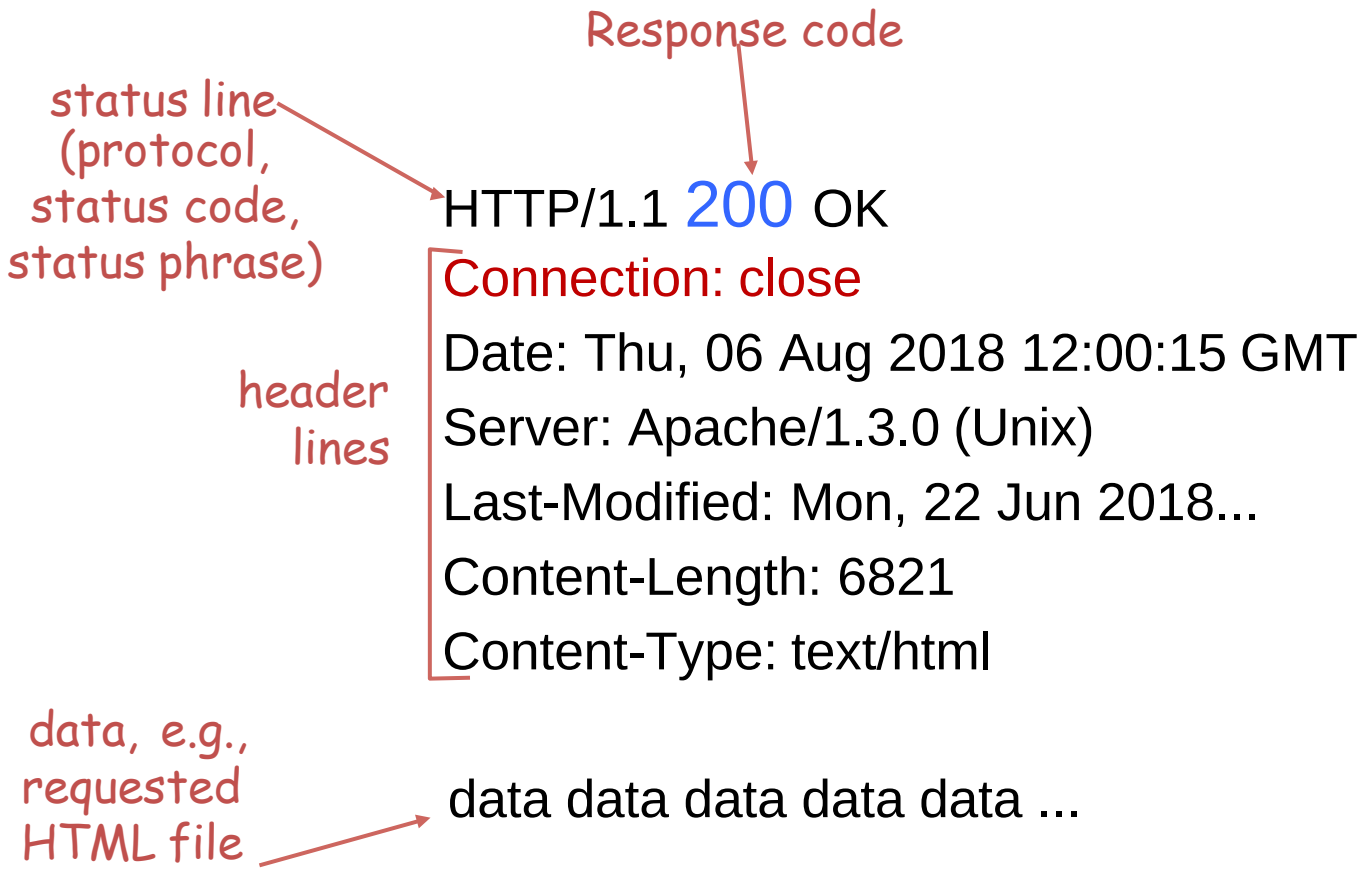
EXAMPLE OF POST METHOD:

```
POST /form.php HTTP/1.1
Host: www.cs.iastate.edu
Content-Length: 12
<CRLF>
username=Joe // This is the body of the request
```

- POST passes arguments in body
- Unlimited data (files) can be sent

- POST is used to send data to a server to **create/update** a resource
- The data sent to the server with POST is stored in the request body of the HTTP request

HTTP response message



HTTP response status codes

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

400 Bad Request

- request message not understood by server

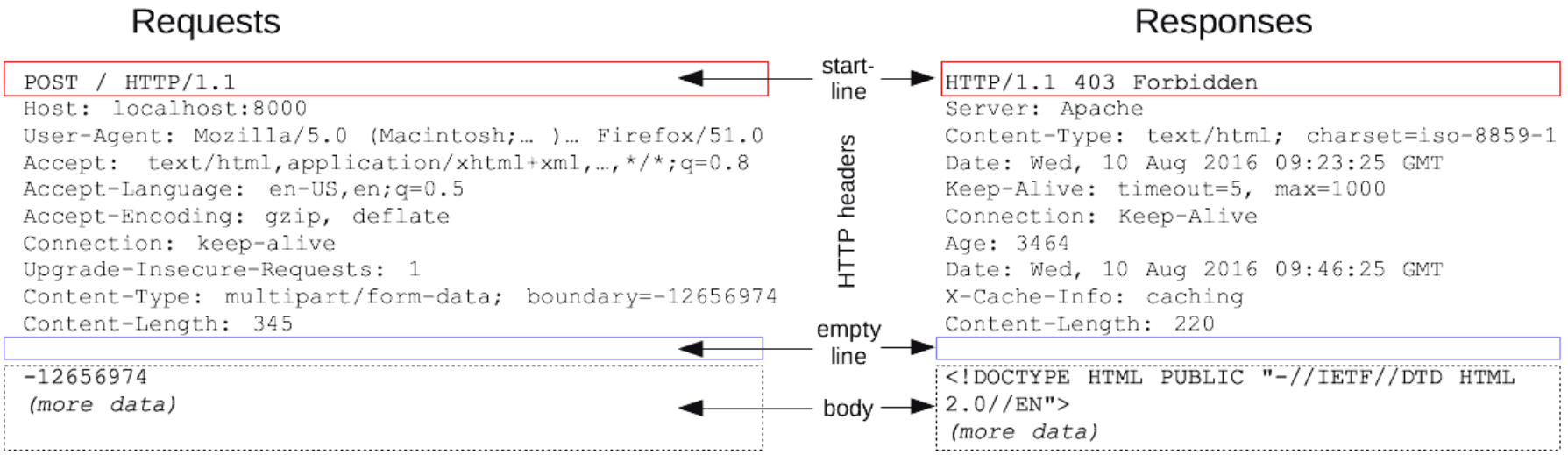
404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

HTTP requests / responses

- **Similar structure:**



Source: https://developer.mozilla.org/

Trying out HTTP (client side)

1. Telnet to your favorite Web server:

```
telnet www.iastate.edu 80
```

Opens TCP connection to port 80 (default HTTP server port)

2. Type in a GET HTTP request:

```
GET /  HTTP/1.1
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Type in another GET HTTP request:

```
GET /  HTTP/1.1
Host: www.iastate.edu
```

Literature – HTTP, HTML, …

Good resources:

- https://www.w3.org
- https://www.w3schools.com/
- http://www.htmldog.com/
- https://www.quackit.com/
- http://www.landofcode.com/