Software Construction and User Interface (SE/ComS 319)

Ali Jannesari

Department of Computer Science

Iowa State University, Fall 2018

EVENT-DRIVEN PROGRAMMING

Outline

- Event-Driven Programming (EDD):
  - Concepts
    - Event handling
    - Event-driven architecture
    - Asynchronous programming, etc.
  - Web UI and EDD with JavaScript (Node.js)
  - GUI and EDD with JavaFX

Event-Driven programming (1)

- A programming paradigm in which the flow of the program is determined by **events** such as:
    - User actions (mouse clicks, key presses)
    - Sensor outputs (mostly in embedded systems)
    - Messages from other programs/threads (device drivers)

Event-Driven programming (2)

- Event-driven programming
  - … is the dominant paradigm used in **graphical user interfaces** and other applications
    - e.g. JavaScript web applications: performing actions in response to user input.
  - … is used in **Human-computer interaction (HCI)**

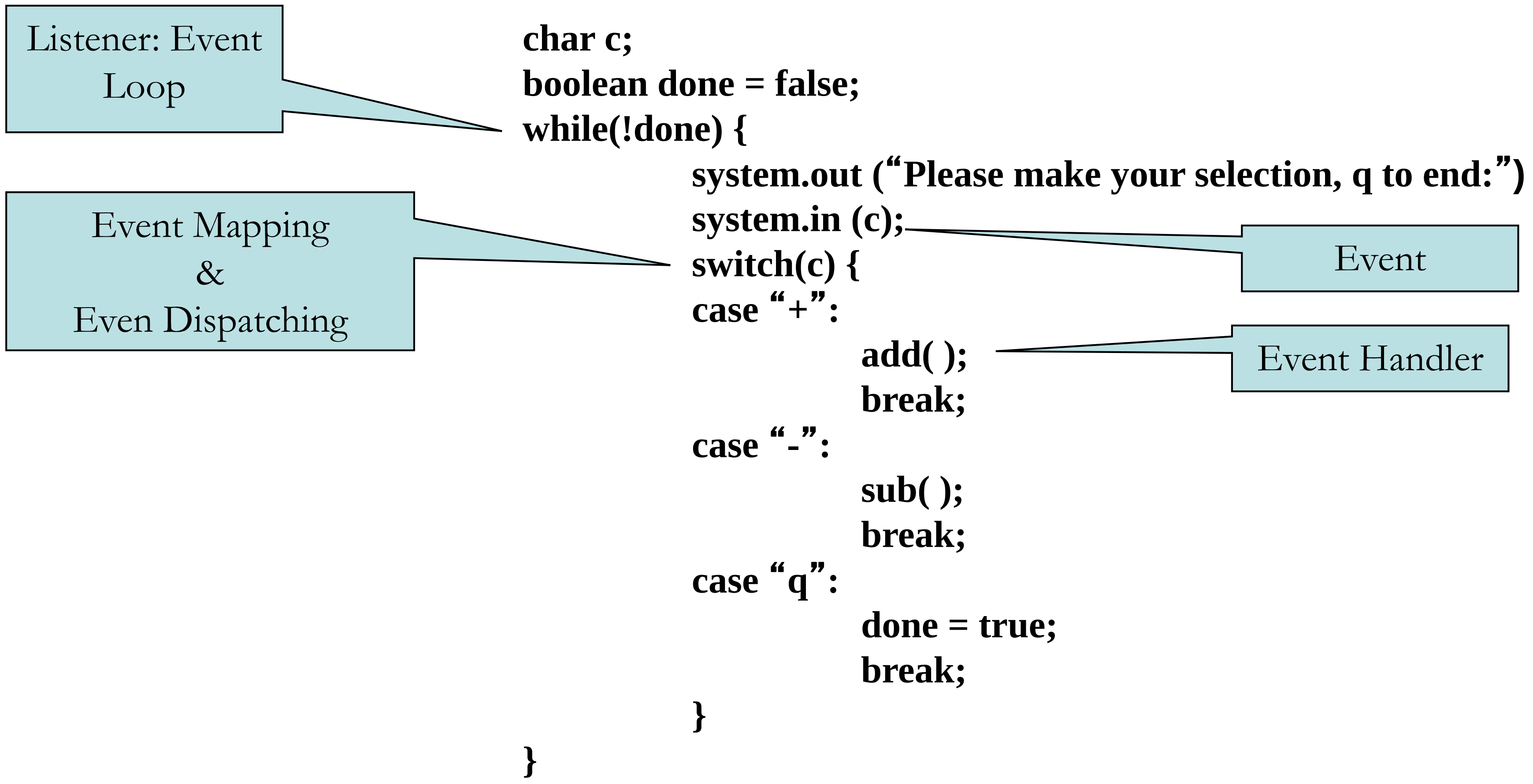
Human-computer interaction (HCI)

- HCI: Interactive computing systems for human use
  - CLI: command line interface (with keyboard)
  - **GUI: graphical user interface (mouse)**
  - NUI: natural user interface with Audio/Video (Kinect)
- A main HCI Component: **Interaction**
  - User interaction
  - Event
  - Event Handling
  - Output
- A **GOOD GUI** allows users to perform interactive tasks easily:
  - What you see is what you get

Event-Driven programming (2)

- Application waits (idles) after initialization until the user generates an event trough an input device (keyboard, mouse, …).
- The OS dispatches the event to the application who owns the active window.
- The corresponding event handler(s) of the application is invoked to process the event.

Event-Driven programming (2)



```
char c;
boolean done = false;
while(!done) {
          system.out ("Please make your selection, q to end:")
          system.in (c);
          switch(c) {
          case "+":
                    add( );
                    break;
          case "-":
                    sub( );
                    break;
          case "q":
                    done = true;
                    break;
          }
}
```

Event-Driven programming (4)

1. Event generators: GUI components (e.g. buttons, menus, …)
2. Events/Messages: e.g. MouseClick, …
3. Event loop (Listener) : an infinite loop constantly waits for events.
4. Event mapping / Event registration: inform event dispatcher which event an event hander is for.
5. Event dispatcher: dispatch events to the corresponding event handlers.
6. Event handlers: methods for processing events. E.g. OnMouseClick(), …

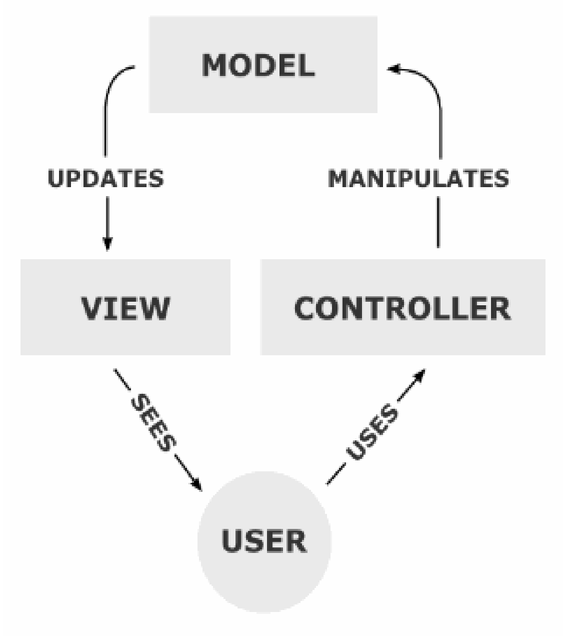
Event-driven programming (5)

- Concepts
- Event-driven programming with
    - JavaScript (Node.js)
    - JavaFX (Java)

EVENT-DRIVEN PROGRAMMING
WEB USER INTERFACES

Event-Driven Programming – Web UI

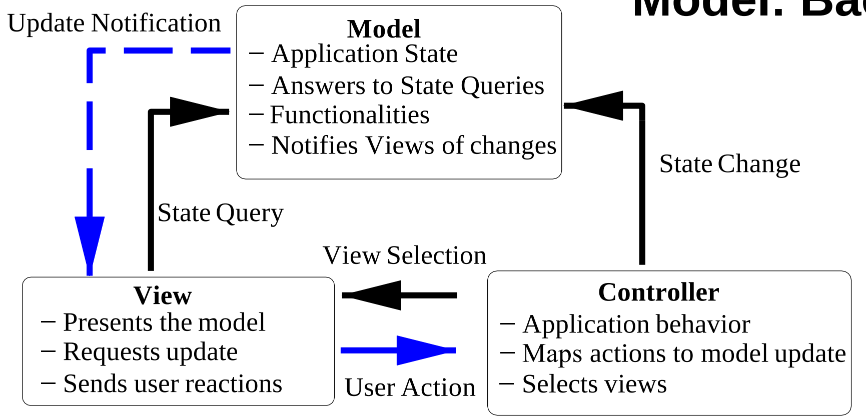
- **MVC (Model – View – Controller) in Web UI:**
  - **View**: Browser presentation (HTML)
  - **Model**: Data (Backend Database or (simple) embedded)
  - **Controller**:
    - Client scripts/programs, e.g. JavaScript
    - Server scripts/programs, e.g. Node.js



MVC

- Model-View-Controller architecture:

Model: Backend Database



View: HTML

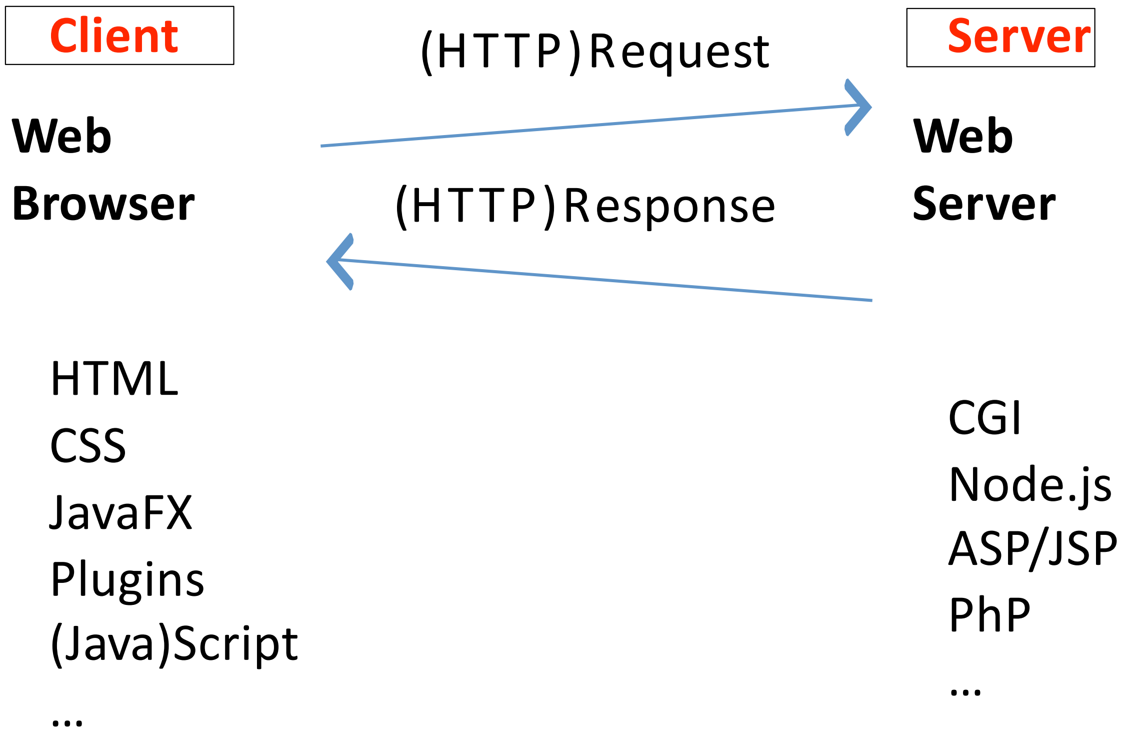
Controller: JavaScript, Node.js code

- Advantages:
  - Loosely coupled, modular
  - Model with different views
  - Controller decides when/how to update the model and/or the view
  - Model can change the view

Client/Server programming

- Use **client-side** programming for
  - Validating user input
  - Prompting users for confirmation, presenting quick information
  - Calculations on the client side
  - Preparing user-oriented presentation
  - Any function that does not require server-side information
- Use **server-side** programming for
  - Maintaining data across sessions, clients, applications

Web software: Client/Server (1)

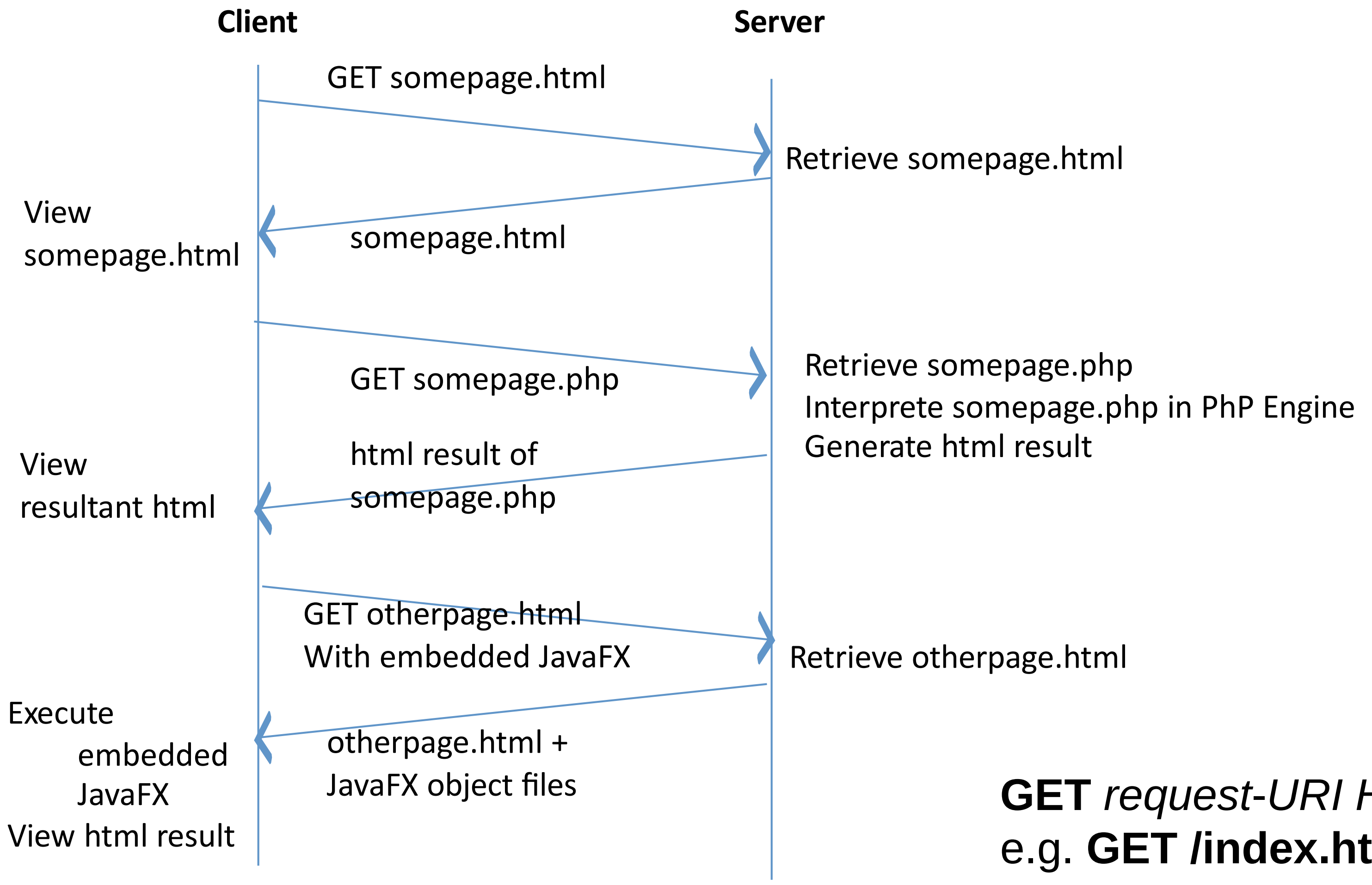


HTTP (Hypertext Transfer Protocol): HTTP is a client-server application-level protocol. It typically runs over a TCP/IP connection.

Web software: Client/Server (2)

- 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

Web software: Client/Server (3)



GET *request-URI HTTP-version*
e.g. **GET /index.html HTTP/1.0**

Common Gateway Interface (CGI) – Classic method

- Standard for the server to communicate with external applications
- Server receives a client (Http) request to access a CGI program
- Server creates a new process to execute the program
- Server passes client request data to the program
- Program executes, terminates, produces data (HTML page)
- Server sends back (Http response) the HTML page with result to the client

HTML – Example

```
<html>
<head></head>
<body>
<form action="<some-server side cgi program>" method="post">
First Name: <input type="text" name="fname"/>
Last Name: <input type="text" name="lname"/>
<input type="submit" value="Submit"/>
</form>
</body>
</html>
```

- Once the user clicks the submit button, the data provided in the form fields are “submitted" to the server where it is processed by a CGI program!

HTTP Request/Response Message

- Message Header
  - Who is the requester/responder
  - Time of request/response
  - Protocol used …
- Message Body
  - Actual message being exchanged

HTTP Request

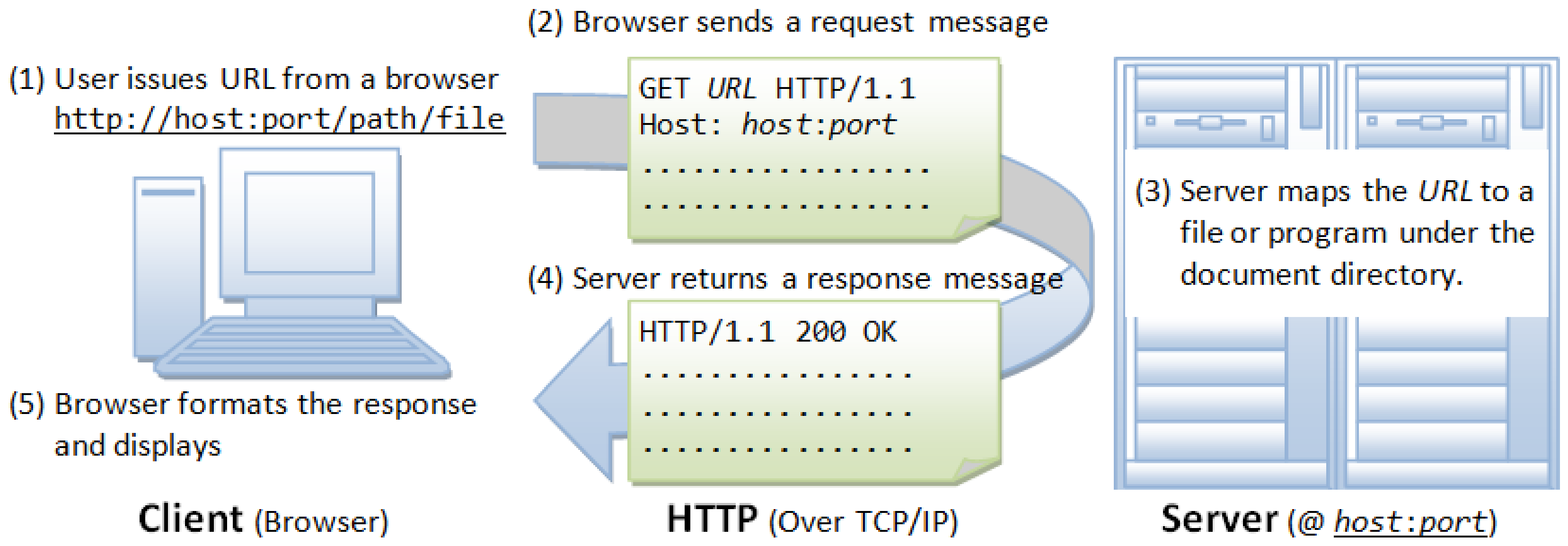
```
GET /index.html HTTP/1.1
Host: http://www.se.iastate.edu
Accept-Language: en
User-Agent: Mozilla/8.0
Query-String: ...
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Sat, 27 Oct 2007 16:00:00 GMT
Server: Apache
Content-Type: text/html
```

- Response Codes:
  - 200s: good request/response
  - 300s: redirection as the requested resource is not available
  - 400s: bad request leading to failure to respond
  - 500s: server failure

Web software: Client/Server

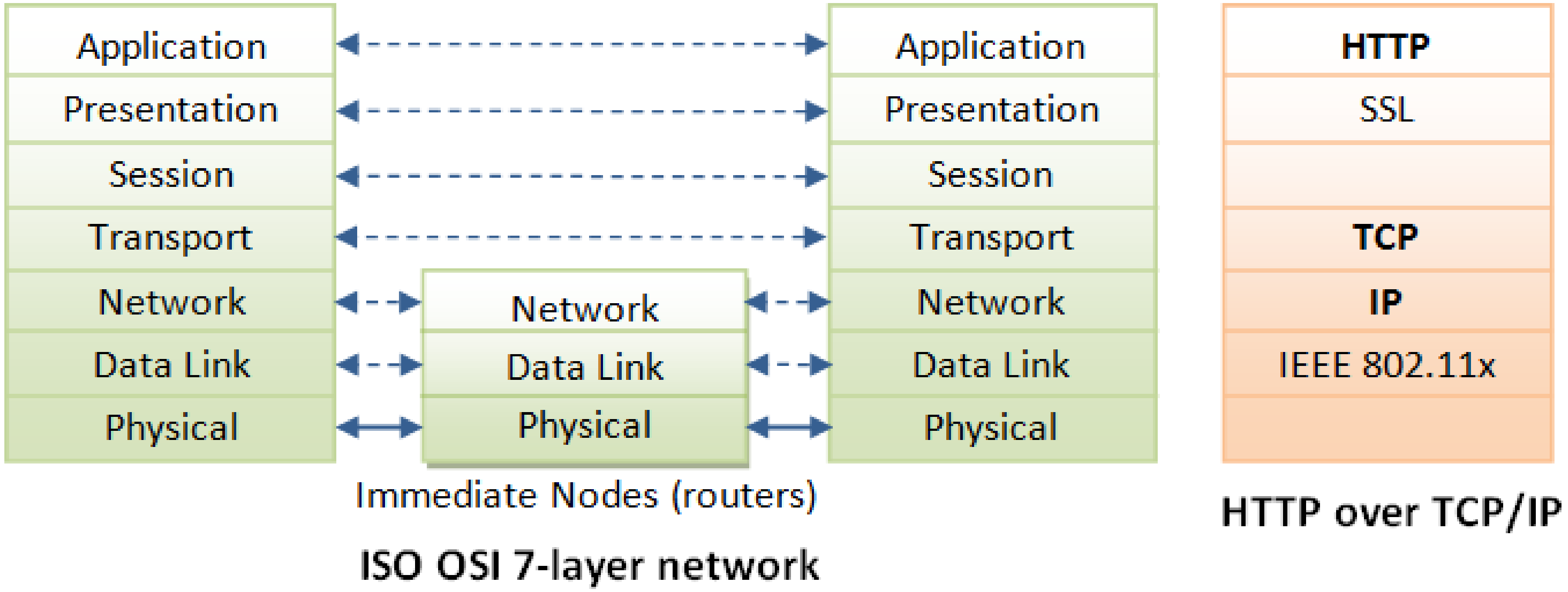


(1) User issues URL from a browser http://host:port/path/file
(2) Browser sends a request message
GET URL HTTP/1.1
Host: host:port
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

Source: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

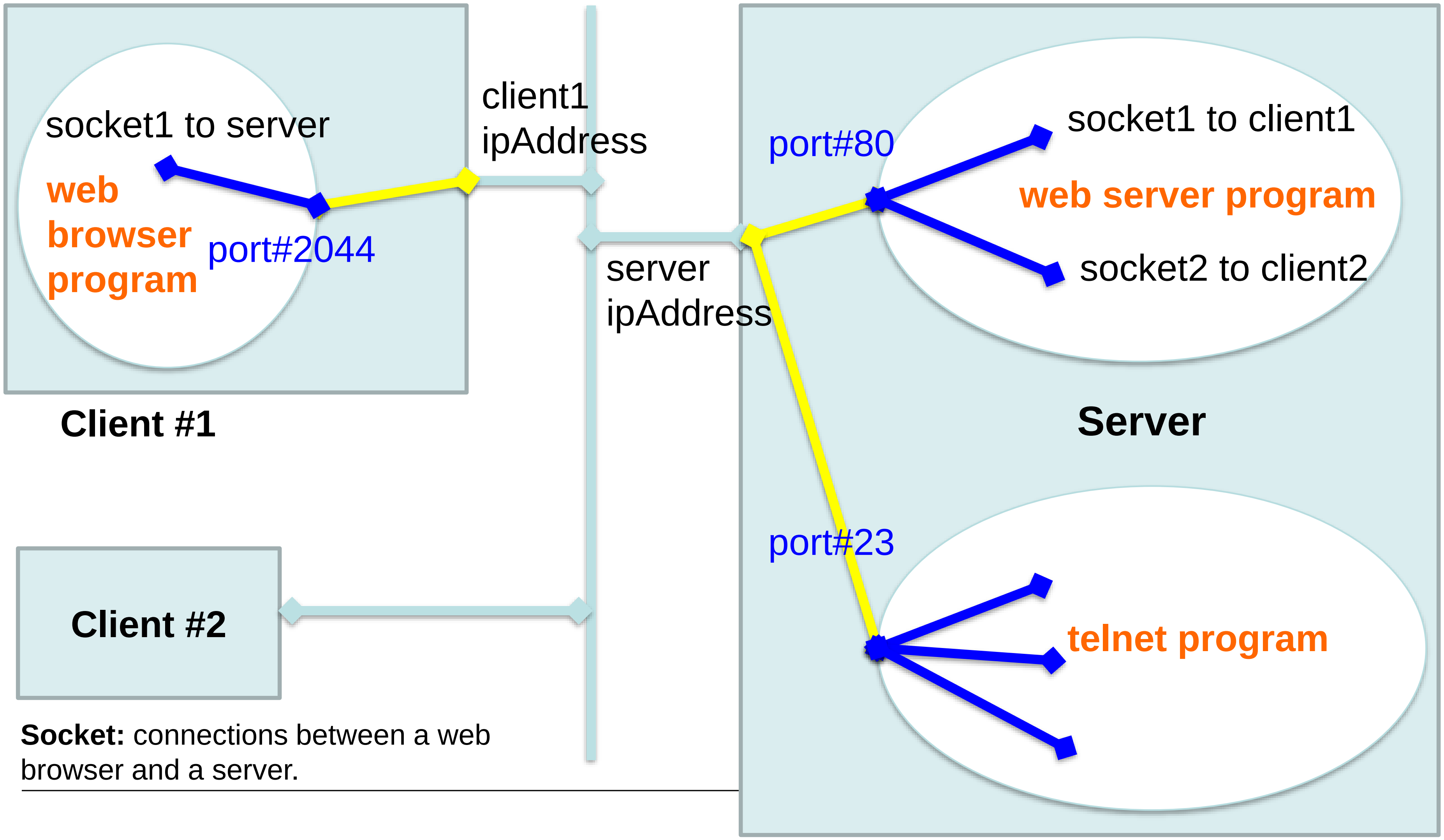
- **GET**: The GET method is used to retrieve information from the given server using a given URI.
  - Requests using GET should only retrieve data and should have no other effect on the data.

Client/Server: HTTP over TCP/IP



Source: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Web software: Client/Server – Connections



Socket: connections between a web browser and a server.

Client-Side Dynamics (1)

- HTML + Javascript
- Html elements: **forms**
- Html style elements: fonts, headings, breaks
- CSS: uniformly manipulate styles
- JavaScript:
  - manipulate styles (CSS)
  - manipulate html elements
  - validate user data
  - communicate with the server-side programs
- In HTML: `<input id="clkb" type="button" value="Click” onclick="clkF()"/>`
- In Javascript file: `function clkF() { alert("Hello"); }`

Client-Side Dynamics (2)

- Html elements: **View**
- CSS: **Model**
- Javascript: **Controller**
- CSS: A simple mechanism for adding style to Web documents.
  - Look & feel of Webpages
  - Layouts, fonts, text, image size, location
  - Objective: Uniform update
- Javascript as a client side event-driven programming
  - Client-side computations
  - Form validation + warnings
  - Dynamic views

How to add JavaScript to html file?

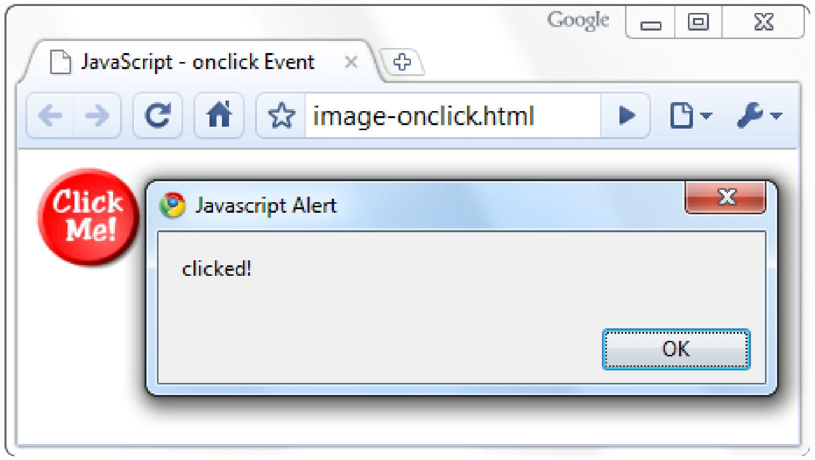
- Include in html file:
  - `<script> your javascript code goes in here </script>`

- Can also include from a separate file:
  - `<script src="./01_example.js"></script>`

- Can include from a remote web site:
  - `<script src="http://…./a.js"></script>`

JavaScript Event Handler – Example

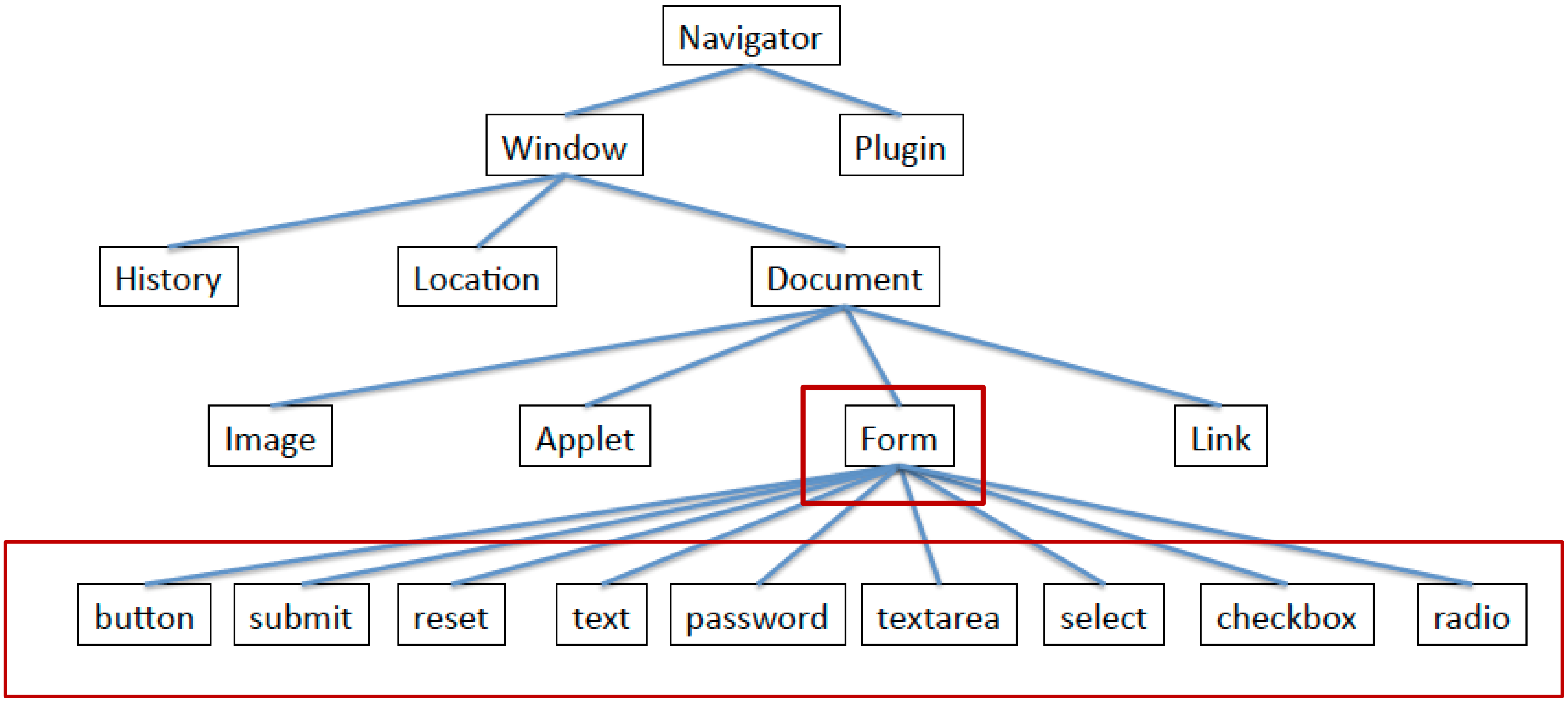
```
<html>
<head>
<script type="text/javascript">
  function test (message) {
    alert(message);
  }
</script>
</head>

<body>
  <img src="logo.gif"
    onclick="test('clicked!')" />
</body>
</html>
```



clicked!

Using **onclick**, we attach **event handlers**.

JavaScript accessibility hierarchy


Navigator
Window
Plugin
History
Location
Document
Image
Applet
Form
Link
button
submit
reset
text
password
textarea
select
checkbox
radio