



IF3110 – Web-based Application Development

Internet Application Concept

Objective

- › Students understand the differences Internet App vs Web App
- › Students know various components composing a Web App
- › Students understand typical processes done in a Web App

Main Concept

- › Internet Programming → Programming Internet-based Application (Internet App)
- › Application that is distributed across the place; using Internet as communication means
- › Keywords:
 - › Distributed System
 - › Communication via Internet

Characteristics

- › Concurrency
 - › Handle many tasks at the same time (e.g., send/receive data, process user requests, handle many clients)
- › Synchronize
 - › activity coordination
 - › time/timing handling
- › Exception Handling
 - › a fail in handling a user; won't cause any problems to other users

Distributed System

- › Main Classes
 - › Client Server: service requester (client) and provide services (server)
 - › Peer-to-peer: each component is in the same level in providing and requesting services
- › In a peer-to-peer system, each component plays a role as client and server

Client-Server Variant

› Stateless Client-Server

- › Each client/server doesn't store each other's status
- › Each client-server interaction is independent and stateless

› Stateful Client-Server

- › Store info about client-server interaction
- › Client can send message based on the existing interaction context (i.e., no need to provide such info at the message)

ada informasi / state yg disimpan

konsup state → pakai token (mungkin masukin uname, dsb)

gunaan web yg by default
stateless (http originally stateless),
buat simpen info state dipakai — mechanism
cookie
session ID
autentikasi
auto refresh

Other Flavour of Client-Server

- › RPC: client-server interaction seen as procedure remote invocation
- › N-tier systems: expand a server into various tiers of servers

what denise see ;



Client-Server Trade-Off

- › Advantageous

- › Distributing computation across several machines
- › Client can access services remotely
- › Client & Server can be designed and developed independently
- › Server can process many simultaneously requests
- › Data can be stored in either centrally in a server or distributed across clients

- › Disadvantageous

- › communication delay
- › need to consider sync and parallel/concurrent process in the server

Protocol Communication

- › Definition: a set of rules agreed by client-server in communication to each other
 - › Application protocol
 - › client-server can send messages to each other with following a particular format/syntax with a specific order
 - › Transport protocol
 - › a message is chunked into many packets
 - › send the packets with various network routes
 - › at the destination the packets are constructed to the original message
- › This course emphasizes at application protocol

Application Protocol using Internet

- › Web (protokol aplikasi: HTTP)
- › E-mail (IMAP, POP, SMTP)
- › Chatting
 - › open standard: IRC
 - › non standard: YM, ICQ, MSN chat, AOL, dll
- › File transfer (FTP)
- › Remote terminal (telnet)
- › Directory service (LDAP)
- › Network monitoring (SNMP)
- › Web service (SOAP)
- › Voice (SIP, XMPP, ASTERISX)
- › etc.



Internet-based vs Web-based Application

Internet-based

- › Using Internet Application Protocol or defining own protocol
- › Application at the server communicates directly to client
- › Application can be a standalone or a component for an existing application

Web-based

- › Using HTTP
- › Application communicate to Client via Web Server
- › Application commonly runs in web browser

Technology in Web-based Application Development

Web client (web browser)

Web server

URL : Uniform Resource Locator

HTTP : HyperText Transfer Protocol

HTML : HyperText Markup Language

CSS : Cascading Style Sheet

Web Programming

CGI

server side scripting

client side scripting

plug-in

Web client (web browser)

web browser

a software

runs in client/user's computer

navigate through the web

render/view the web page

Example:

Chrome (Windows)

Internet Explorer (Windows)

Mozilla Firefox (Windows & Linux)

Opera (Windows & Linux)

Safari (Mac)

lynx, berbasis teks (Linux)

Web server

- › web server
 - › a software
 - › runs in a server
 - › store web document so it can be accessed by users across the Internet
- › Example:
 - › Apache (Linux & Windows)
 - › NginX
 - › MS Internet Information Server / IIS (Windows)
 - › Tomcat, untuk Java (Windows & Linux)

URL (Uniform Resource Locator)

URL is locating a resource (file) in the internet

URL format is defined in RFC 1738 (<http://www.ietf.org/rfc/rfc1738.txt>)

URL has protocol type

Example

http://www.if.itb.ac.id/

mailto:fulan@informatika.org

ftp://ftp.itb.ac.id/

http → send & goodbye, stateless
antara req 1 & 2 ga ada hubungannya.

ftp → buat file, statefull

Example of URL in the web:

domain

path

scheme
(menentu-
kan protoco)
http://

http://www.itb.ac.id/campus-life/index.html

menentu-
kan protoco)

http://www.google.com/search?hl=en&q=URL+RFC

query

params

http://www.indymedia.org:8081/

TLD. (top level domain)

ex: .com, .id, .net

domain di resolve oleh resolver
→ recursive query. iterative
sampai ketemu authoritative
server.

GET → fungsi, ga ngubah state
 POST → procedure, state di server berubah

HTTP (HyperText Transfer Protocol)

- HTTP is a standard web protocol communication
- HTTP standard (HTTP 1.1) defined in RFC 2616 (<http://www.ietf.org/rfc/rfc2616.txt>)

Example line request (verb/method)



HTML (HyperText Markup Language)

- › HTML is a standard document in the web
- › HTML standard (HTML 4.01) accessible at <http://www.w3.org/TR/html4/>
- › HTML standard (HTML 5) accessible at <https://html.spec.whatwg.org/multipage/>
- › Example:

```
<html>
<head>
    <title>My first HTML document</title>
</head>
<body>
    <p>Hello world!<br>Welcome to my <b>first</b> HTML
page.
    </p>
</body>
</html>
```

- › Output:

```
Hello world!
Welcome to my first HTML page.
```

CSS (Cascading Style Sheet)

- › CSS is a mechanism to define style (font, template, colour) in a web document
- › CSS standard (CSS 2) is accessible at <http://www.w3.org/TR/REC-CSS2/>
- › Example:

```
<html>
<head>
    <title>My first HTML document</title>
</head>

<body>
    <p>Hello world!<br>Welcome to my <b>first</b> HTML page.
    </p>
</body>
</html>
```

- › Output:

Hello world!
Welcome to my **first** HTML page.

TCP/IP (Internet) Programming

- › Web Programming often uses HTTP as transport protocol and HTML/XML as message format
 - › Text oriented
 - › Stateless, client server oriented
- › More flexibility can be achieved by using transport or network protocol directly
- › IP Network (layer transport):
 - › TCP: connection oriented
 - › UDP: datagram oriented
- › Communication mode:
 - › Unicast, Multicast, Broadcast

Web Programming

Web Programming

- › CGI, executing code in server-side (perl, C) → jembatan server dgn proses apapun
 - › Web server executes a program and the output is piped to HTTP response
- › server side scripting (PHP, ASP, JSP, Phyton)
 - › Web server identifies and runs script/program and insert the outputs as a part of the web document
- › client side scripting (JavaScript, JScript, VBScript)
 - › Web browser identifies and runs the script/program and insert the outputs to the web document (received from the server) or modifies the web document or does what the script orders (e.g., XHR, Open Socket)
- › plug-in, eksekusi program di sisi client (applet, ActiveX, Flash)
 - › Web browser executes the program with the help of plugin and view the outputs at a defined place in the web document

Web Application Evolution

- › Web site – a collection of documents accessible through internet
- › Web app – a client/server app through a web as the client, and uses Internet to communicate
- › Web Server → Web App Server

Typical web app processing

Source:
Shklar, L.
Web Application Architecture:
Principles, Protocols and
Practices
Wiley Publishing, Inc., 2003

JWT
ref ke hotel
checkin → dpt token.
dpt jsl yg
verify bukan org
lobby

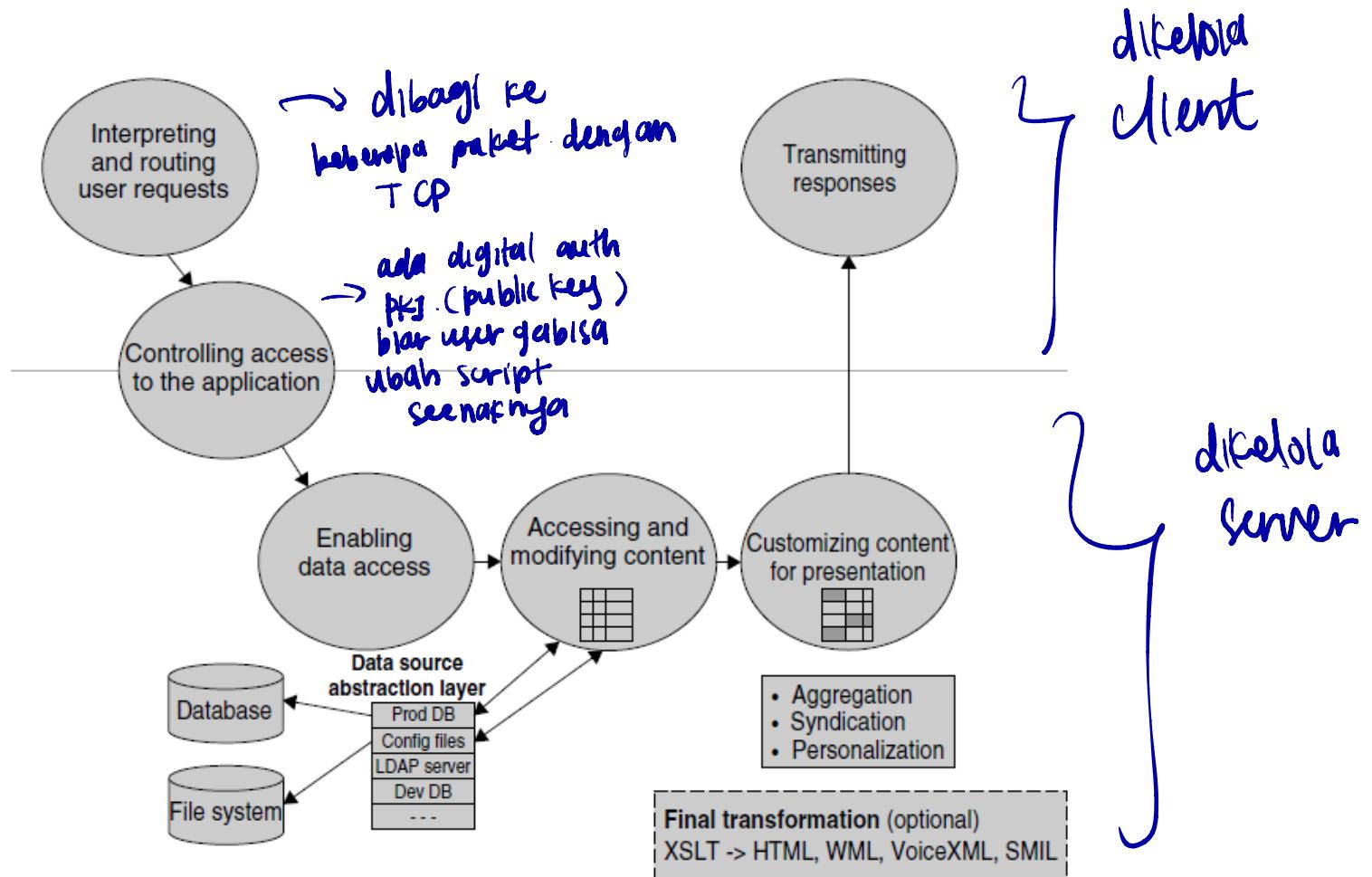


Figure 8.3 Processing flow in a typical Web application (Above the grey line—Web server; below the grey line—Web application)

Web Stack

- › A collection of software required to develop and run web applications
- › Contains all software layers
 - › Operating System
 - › Web Server
 - › Programming Language (frontend & backend)
 - › Web Framework (frontend & backend)
 - › Database Server
- › Examples:
 - › LAMP, MEAN, ...
- › It is **different** from Web Framework

Interpreting & processing request

- › Browser communicate the request via HTTP (info sent by the browser called *request context*)
 - URL, query string, request headers, request body, session information

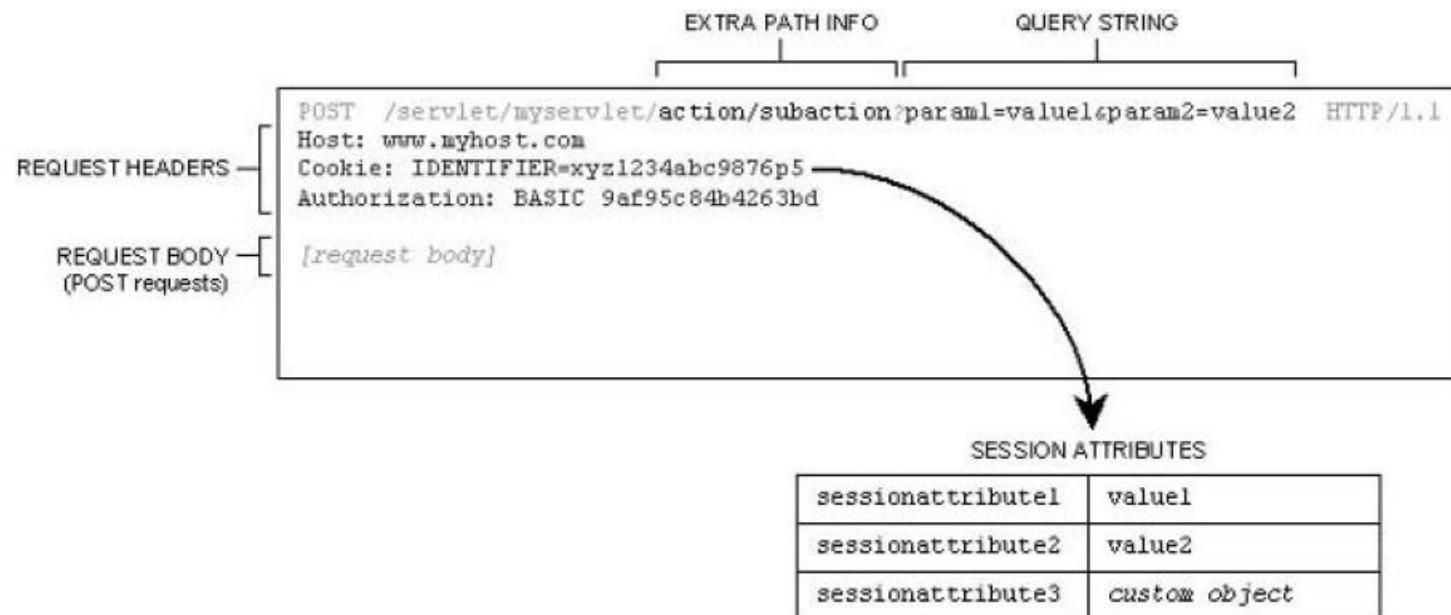


Table 8.1 Selection of server processing modules

Approach	Configuration	URL Examples
CGI	<ol style="list-style-type: none">1. Server provides mechanism for defining CGI path mappings.2. Server provides mechanism to map URL file name extensions to CGI processing.	http://host/cgi-bin/script?... http://host/.../script.cgi?...
Auxiliary processing modules (scripting, template, hybrid)	<ol style="list-style-type: none">1. Server provides mechanism for registering processing modules for files with predefined name extensions.2. Native support may be built into server (e.g., ASP on IIS).	http://host/.../modulename.php?... http://host/.../modulename.cfm?... http://host/.../modulename.asp?...
J2EE Webapp	<ol style="list-style-type: none">1. Server provides mechanism for defining application path mappings.2. Server provides mechanism to map URL file name extensions to be processed as JSP pages.	http://host/servlet/servletname/... http://host/webappname/servletname/... http://host/webappname/modulename.jsp?... http://host/anotherjsp.jsp

Controlling User Access

- › HTTP provides an authentication mechanism
 - › web server ask authentication info; when the request doesn't have such info

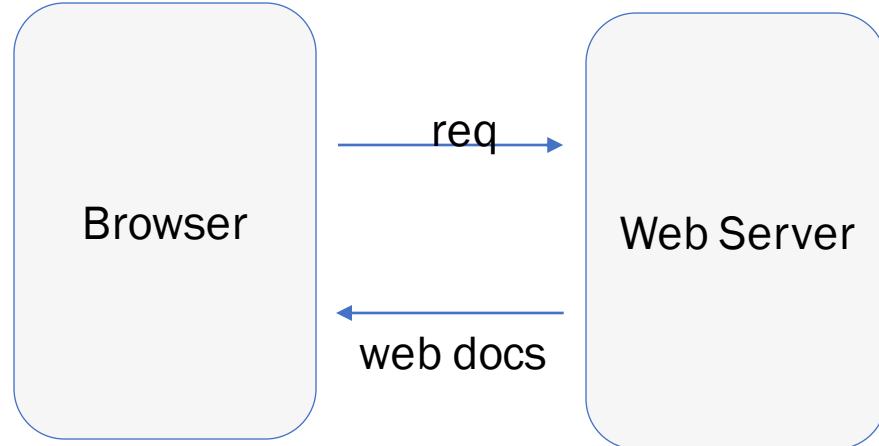
Web Application Architecture

- › presentation layer:
 - › markup / HTML/XHTML
 - › templating system
 - › code segregation
 - › CSS
- › presentation/page logic
- › business/application logic
- › persistent store

Web Site Architecture (Old)

Source“:

Microsoft(r) Application
Architecture Guide, 2nd
Edition (Patterns &
Practices)
Microsoft Press, 2009



Web Application Architecture

Source“:

Microsoft(r) Application
Architecture Guide, 2nd
Edition (Patterns &
Practices)
Microsoft Press, 2009

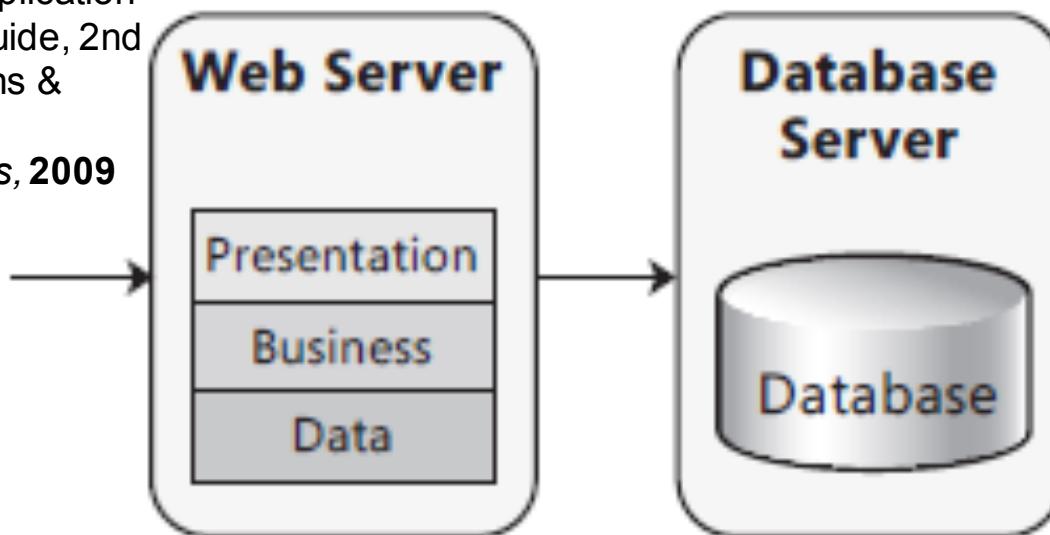


Figure 2
Nondistributed deployment of a Web application

Web Application Architecture

Source“:

Microsoft(r) Application
Architecture Guide, 2nd
Edition (Patterns &
Practices)

Microsoft Press, 2009

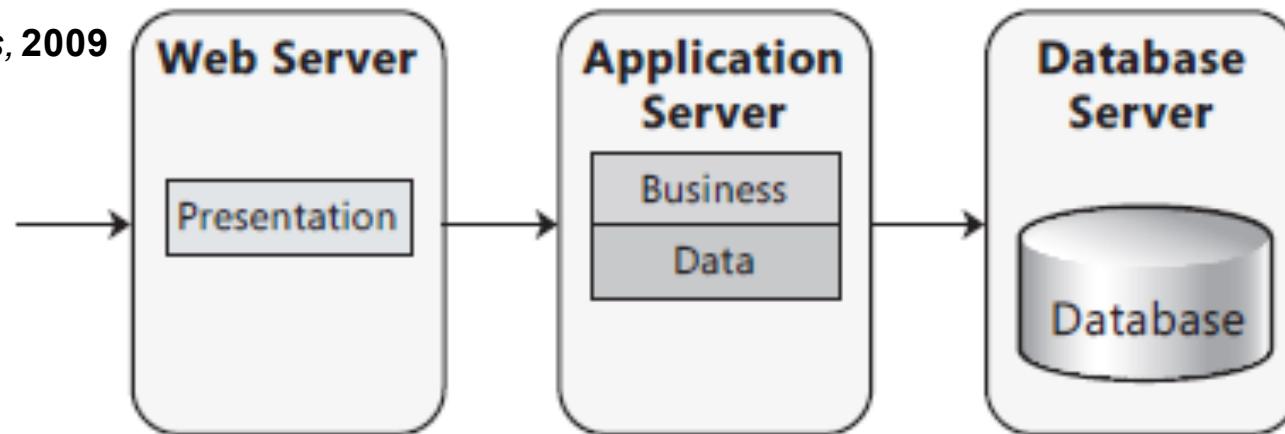


Figure 3

Distributed deployment of a Web application

Application Architecture

Source“:

Microsoft(r) Application
Architecture Guide, 2nd
Edition (Patterns &
Practices)
Microsoft Press, 2009

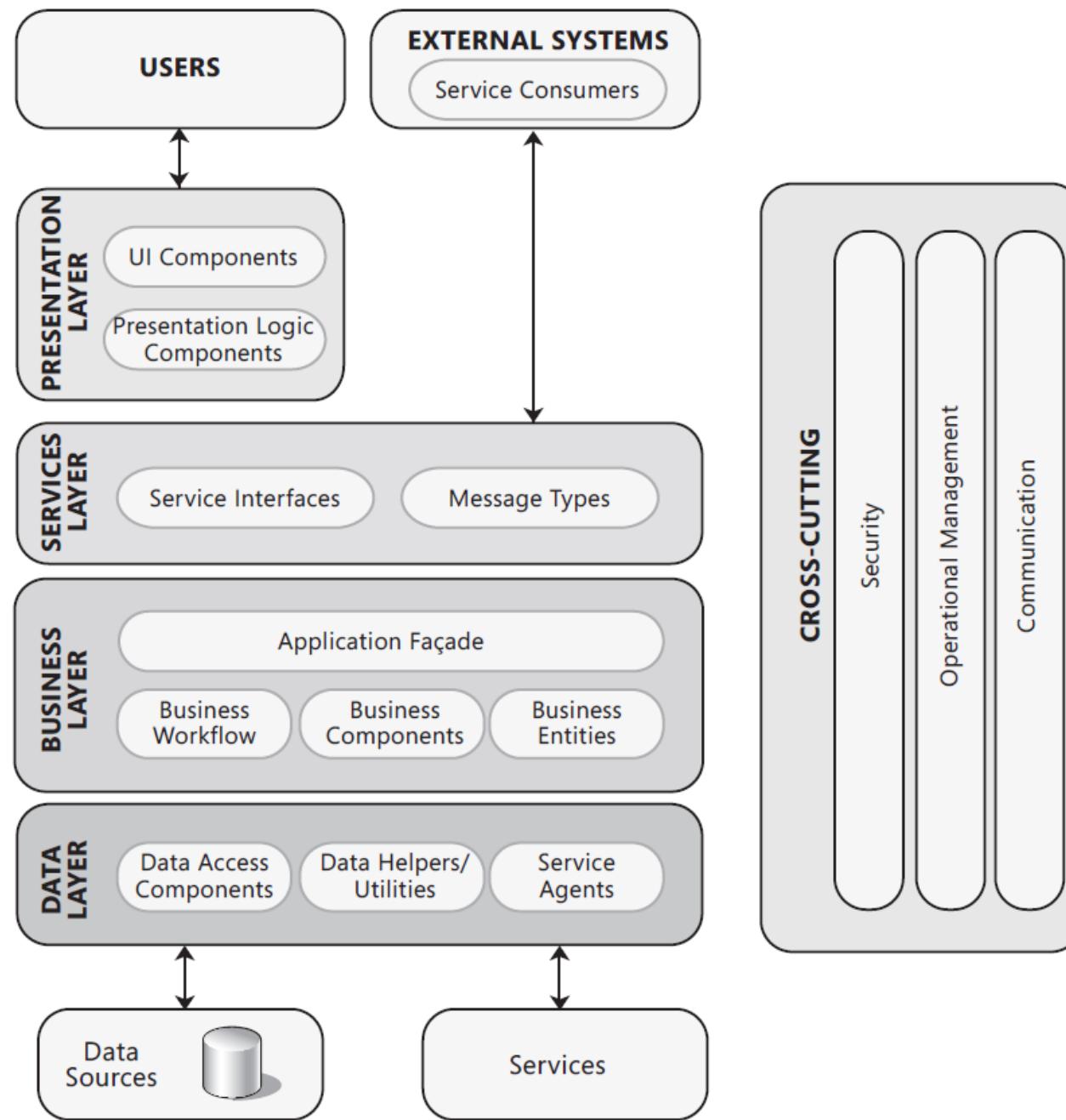


Figure 1
Common application architecture

Common Aspect in Web-App

- › App request processing
- › Authentication
- › Authorization
- › Caching
- › Exception Management
- › Logging & Instrumentation
- › Navigation
- › Page layout
- › Page rendering
- › Session management
- › Validation
- › Internationalization
- › User Experience

Exercise: Under the Hood of Internet App (FTP and Web App)

› Objective

- › Students understand how an Internet App Client communicate with the Server

› Tools

- › FTP Client FileZilla <https://wiki.filezilla-project.org/Using>
- › Simple HTTP Server for Testing <http://neverssl.com>
- › Free FTP Server for Testing: <https://dlptest.com/ftp-test/>
 - › FTP URL: ftp.dlptest.com or <ftp://ftp.dlptest.com/>
 - › FTP User: dlptester
 - › Password: rNrKYTX9g7z3RgJRmxWuGHbeu
- › Capturing Live Network Data with Wireshark https://www.wireshark.org/docs/wsug_html/#ChapterCapture
- › Inspect Web Network Activity using Chrome DevTools <https://developer.chrome.com/docs/devtools/network>

Exercise: Under the Hood of Internet App (FTP and Web App) - contd

› Instruction

- › Access the service via a client (HTTP/FTP) that has been run at a HTTP/FTP server
- › Capture the traffic with Wireshark

Question:

- › What has been done under the hood (in the network)?
- › What are the differences between FTP and HTTP based on what you have discovered?
- › What are the differences between stateful protocol and stateless protocol?

HTTP Protocol

IF3110 – Web-based Application Development
School of Electrical Engineering and Informatics
Institut Teknologi Bandung

Reference

- Leon Shklor and Richard Rosen, **Web application architecture: principles, protocols, and practices**, John Wiley & Sons Ltd (2003)
- <https://httpwg.org/specs/>

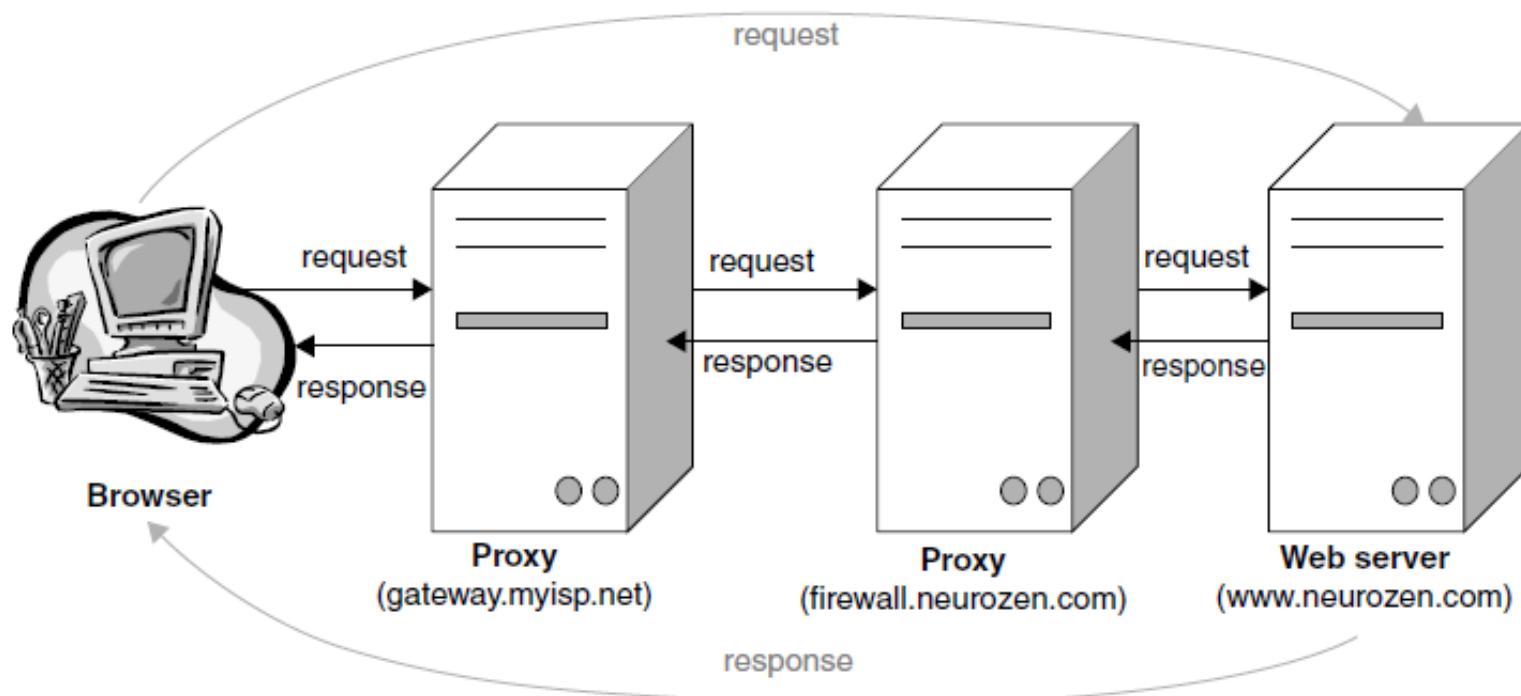
About HTTP

- Basic Ingredient Protocol for World Wide Web
- Simple – strength and weakness
- Doesn't manage the state with limited functionality
- Application Layer Protocol founded by TCP

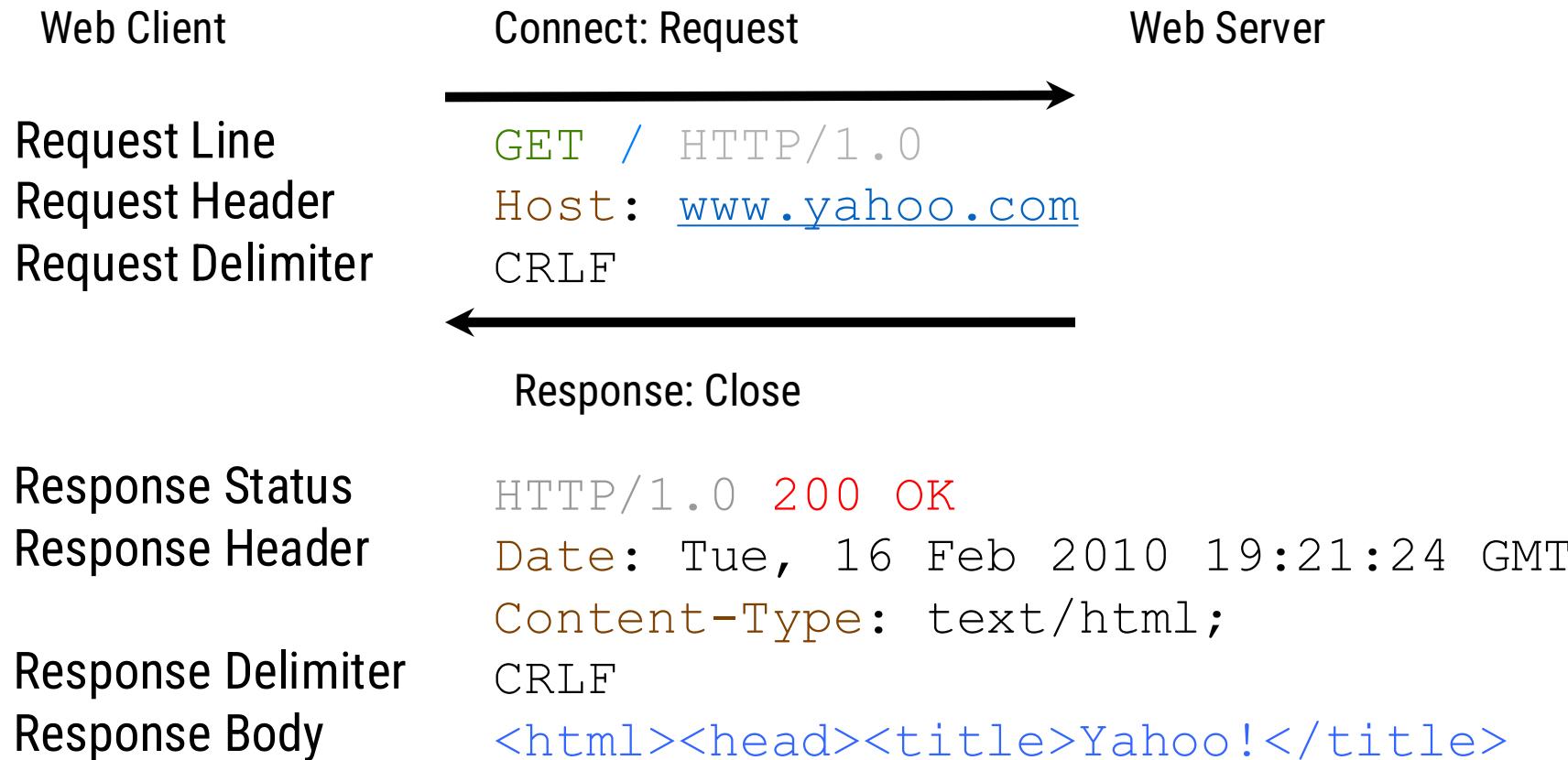
HTTP Protocol and Structure

- *request* and *response* paradigm
- *request* and *response* has similar structure
 - couple lines of *header*
 - empty line, followed *message body*
- *Stateless protocol* – a transaction is composed of a single *request* from *client* and a *response* from *server*

Request Response Virtual Circuit



Anatomy of HTTP 1.0



HTTP Request Structure

```
METHOD /path-to-resource HTTP/version-number  
Header-Name-1: value  
Header-Name-2: value  
  
[ optional request body ]
```

```
GET /q?s=YHOO HTTP/1.1  
Host: finance.yahoo.com  
User-Agent: Mozilla/4.75 [en] (WinNT; U)
```

```
HEAD http://www.cs.rutgers.edu/~shkclar/ HTTP/1.1  
Host: www.cs.rutgers.edu  
User-Agent: Mozilla/4.75 [en] (WinNT; U)
```

HTTP Request Structure

- *Request Line:*
 - *Request Methods:* **GET, POST, HEAD, PUT, DELETE, TRACE, OPTION, CONNECT, PATCH**
 - access URL
 - Version HTTP: 1.0 or 1.1
- Header – variable: value pairs
 - Host
 - Content-Type
 - Content-Length
 - User-Agent
 - Cookie, dll
- *Request body*

HTTP Response Structure

```
HTTP/version-number    status-code   message  
Header-Name-1: value  
Header-Name-2: value  
  
[ response body ]
```

```
HTTP/1.0 200 OK  
Date: Sat, 03 Feb 2001 22:48:35 GMT  
Connection: close  
Content-Type: text/html  
Set-Cookie: B=9ql5kgct7p2m3&b=2; expires=Thu, 15 Apr 2010 20:00:00 GMT;  
path=/; domain=.yahoo.com  
  
<HTML>  
<HEAD><TITLE>Yahoo! Finance - YHOO</TITLE></HEAD>  
<BODY>  
...  
</BODY>  
</HTML>
```

HTTP Response Structure

- *Status line:*
 - Version HTTP: 1.0 or 1.1
 - *Status Code* and some description
- Header – variable: value pairs
 - Content-Type
 - Content-Length
 - Set-Cookie
 - Date, dll
- *Response body*

- Once receive the document; browser parses the doc to define additional resources to be retrieved

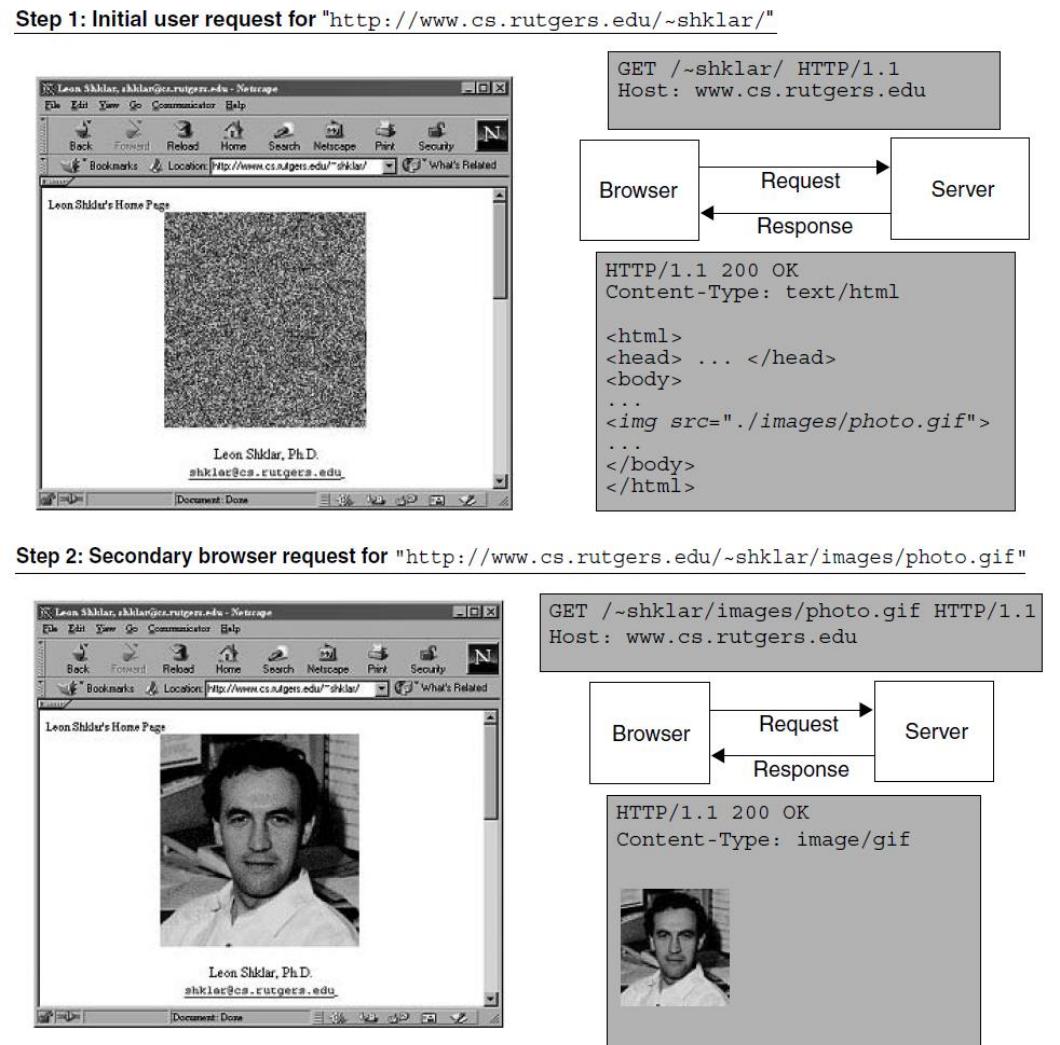


Figure 3.2 Sequence of browser requests for loading a sample page

Request Methods

- GET
 - most simple
 - doesn't contain *request body*
 - *request parameters* will be added in the URL *query string* (after "?")
 - Only retrieve the resource/data, without any other effects (inc. modification/deletion)
- POST
 - *request body* contains *request parameters*
 - Submits data to be processed to a specified resource
 - URL doesn't contain any data (suitable for submit FORM)

Form Processing

- Form w/ POST method, will use HTTP POST to send data, with
content-type: application/x-www-form-urlencoded
- Query parameter will be provided as pairs of type: value
- File upload uses
content-type: multipart/form-data

Form Processing

```
POST /enlighten/calais.asmx/Enlighten HTTP/1.1
Host: api.opencalais.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length

licenseID=string&content=string&paramsXML=string
```

GET vs POST

GET

- can be cached
- remain in the browser history
- can be bookmarked
- Data is visible to everyone in the URL
- have length restrictions
- should be used only to retrieve data
- Only ASCII characters allowed

POST

- never be cached
- do not remain in the browser history
- cannot be bookmarked
- Data is not displayed in the URL
- have no restrictions on data length
- No restrictions. Binary data is also allowed

Request Methods

- HEAD
 - Similar to GET, but server MUST NOT return a message body in the response
 - Server only returns *header*
 - To support *cache* with *content modification information (Last-Modified)*

Request Methods

- **PUT**
 - to store a resource on a particular URI
 - if the URI refers to existing resource then the resource is being updated
- **DELETE**
 - to delete a resource.
- **TRACE**
 - send back the request received by the server
 - client can identify what the additional info added in a HTTP request (e.g., by http proxy)

Request Methods

- OPTIONS
 - return HTTP methods supported by the server on a particular URL
- CONNECT
 - convert request into a transparent TCP/IP tunnel,
 - used in SSL-encrypted used in HTTPS
- PATCH
 - used in modification on a part of the resource

Status Code

- Inform browser or proxy whether *response* is as expected
 - 1xx information
 - 2xx success
 - 3xx redirection
 - 4xx client request error
 - 5xx server error

HTTP Header

- **General Header**
 - Date: Sun, 11 Feb 2001 22:28:31 GMT
Date time created message
 - Connection: Close
Client or Server define whether the connection is maintained/not
- **Request Header**
 - User-Agent: Mozilla/4.75 [en] (WinNT; U)
browser user agent
 - Host: www.neurozen.com
to support virtual host
 - Referer: http://www.cs.rutgers.edu/index.html
URL of the referral

HTTP Header

- **Response Header**
 - Location: `http://www.mywebsite.com/Page.html`
Page intended to visit (*redirect*)
 - Server: Apache/1.2.5
Server ID
- **EntityHeader**
 - Content-Type: mime-type/mime-subtype
type of message body
 - Content-Length: xxx
length of message body
 - Last-Modified: Sun, 11 Feb 2001 22:28:31 GMT
modification date of the content

Virtual Hosting

```
GET http://finance.yahoo.com/q?s=YHOO HTTP/1.1  
Host: finance.yahoo.com
```

```
GET /q?s=YHOO HTTP/1.1  
Host: finance.yahoo.com
```

Authentication

```
HTTP/1.1 401 Authenticate
Date: Mon, 05 Feb 2001 03:41:23 GMT
Server: Apache/1.2.5
WWW-Authenticate: Basic realm="Chapter3"
```

```
GET /book/chapter3/index.html HTTP/1.1
Date: Mon, 05 Feb 2001 03:41:24 GMT
Host: www.neurozen.com
Authorization: Basic eNCoDED-uSErId:pA$swORd
```

Session Management

```
GET /movies/register HTTP/1.1
Host: www.sample-movie-rental.com
Authorization:...
```

```
HTTP/1.1 200 OK
Set-Cookie: CLIENT=Rich; path=/movies
...
```

```
GET /movies/rent-recommended HTTP/1.1
Host: www.sample-movie-rental.com
Cookie: CLIENT=Rich
```

Caching control

```
GET /~shklar/ HTTP/1.1
Host: www.cs.rutgers.edu
If-Modified-Since: Fri, 11 Feb 2001 22:28:00 GMT
```

Persistent Connection

- HTTP 1.0 uses TCP separately for each request
 - not efficient
 - slow (i.e., high-latency)
- HTTP 1.1 uses a persistent connection, can be used by several requests

`Connection: Close`

`Connection: Keep-Alive`

HTTP 1.1 vs 1.0

- Additional Methods (PUT, DELETE, TRACE, CONNECT + GET, HEAD, POST)
- Additional Headers
- Transfer Coding (chunk encoding)
- Persistent Connections (content-length matters)
- Request Pipelining

HTTP Sometime Back

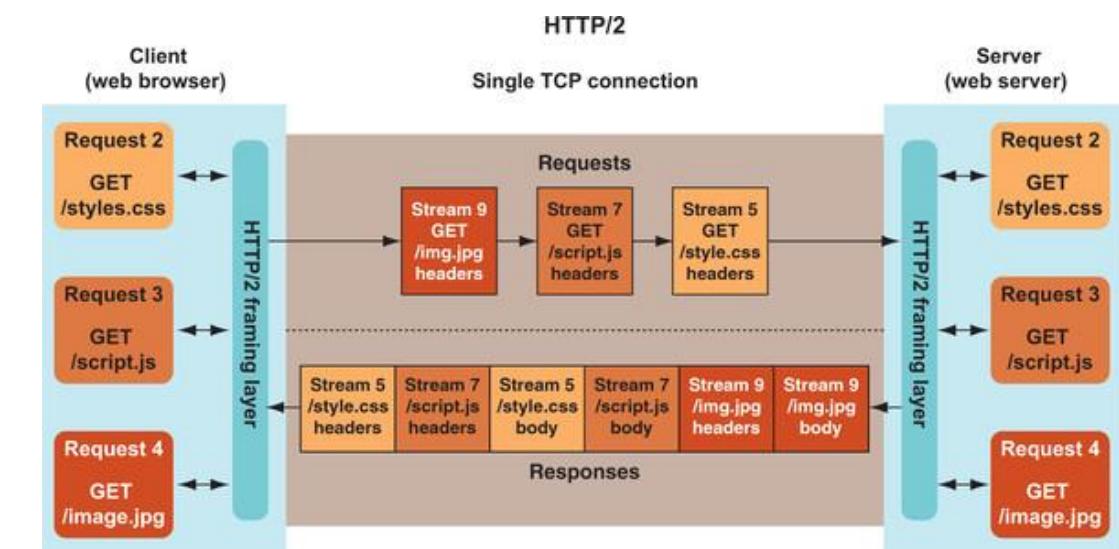
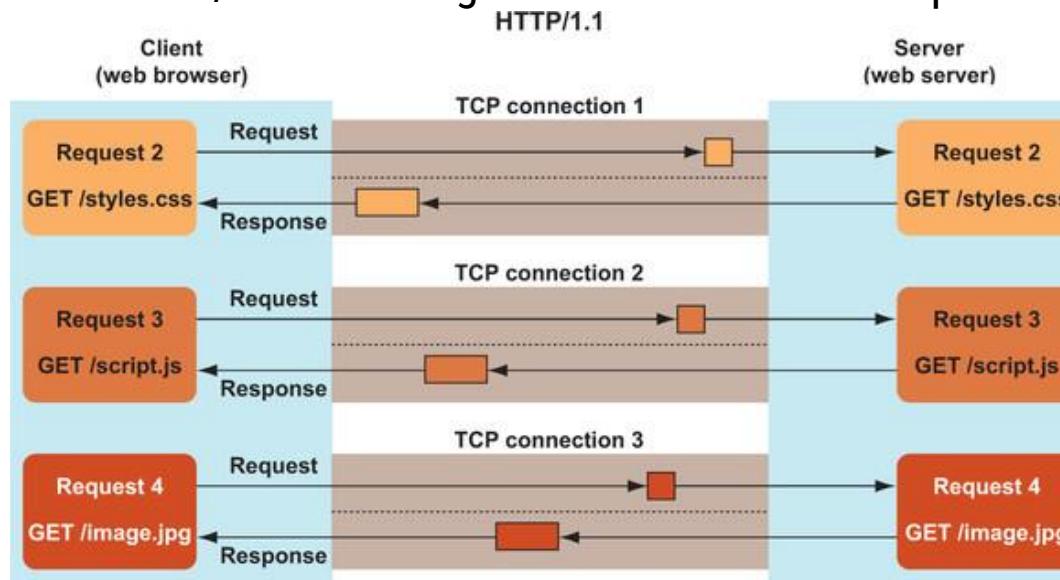
- Using HTTP 1.1 since 1997 / 1999
 - Connection: keep-alive
 - Head of Line Blocking
- But we still use N TCP Connections per origin
- No Header Compression
- And Many Hacks because requests are evil
 - Spriting of Images
 - Resource Inlining
 - Concatenation of files
 - Domain Sharding
 - CDNs

HTTP/2

- Addressing HTTP 1.1 Performance Issues:
 - Binary rather than textual protocol
 - Multiplexed rather than synchronous
 - Flow control
 - Stream prioritization
 - Header compression
 - Server push

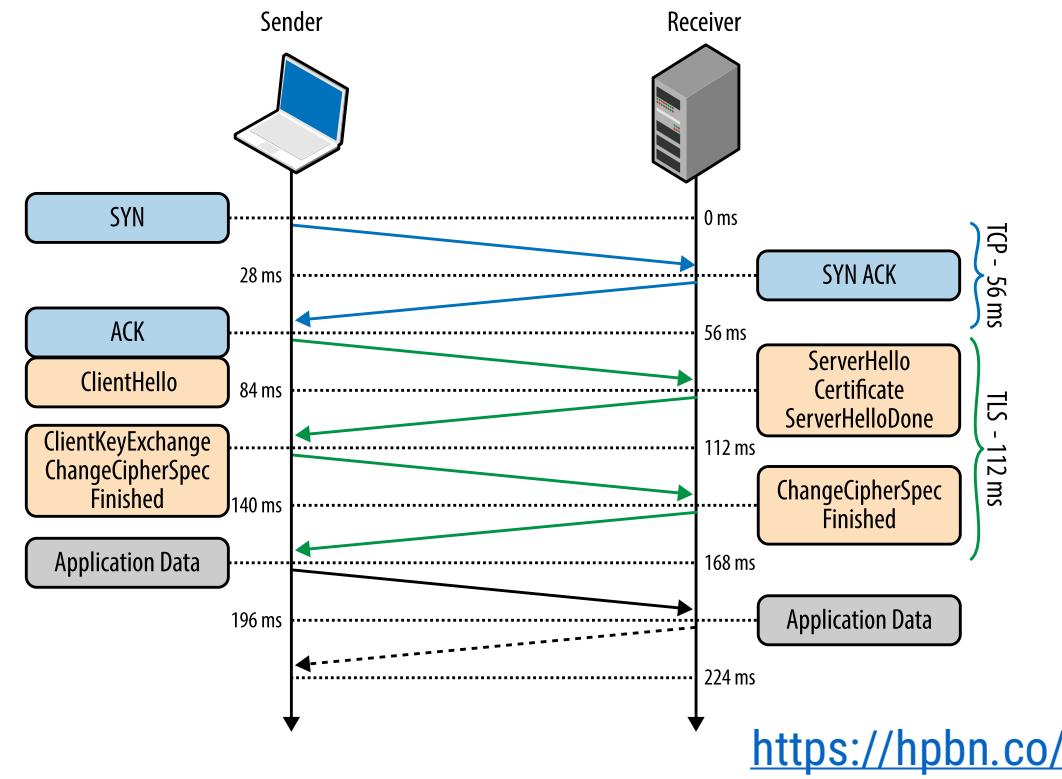
Multiple TCP connection vs Single Connection

- Multiple HTTP/1 requests in parallel require multiple TCP connections.
- Most browsers open up to six connections per domain in parallel for this reason.
- But when the max. connection reached => blocking occurs. Also, TCP connections are expensive.
- A technique called domain sharding splits resources into many domains to address this limitation.
- HTTP/1 may allow "persistent connection": using connection:keep-alive header to use one connection for several requests. However this introduce *head-of-line blocking*: the current request must complete before the next one can be sent.
- HTTP/2 allows single TCP connection multiplex streams of resources over frames which may be interleaved.



HTTPS

- Is HTTP over TLS over TCP
- Hence, TCP 3 way handshake and certificate exchange is performed before actual HTTP protocol application data.



HTML & CSS

HTML



CSS



IF3110 – Web-based Application Development
School of Electrical Engineering and Informatics
Institut Teknologi Bandung

Reference

It is a living document

<https://html.spec.whatwg.org/multipage/>

Additional reference:

<https://developer.mozilla.org/en-US/>

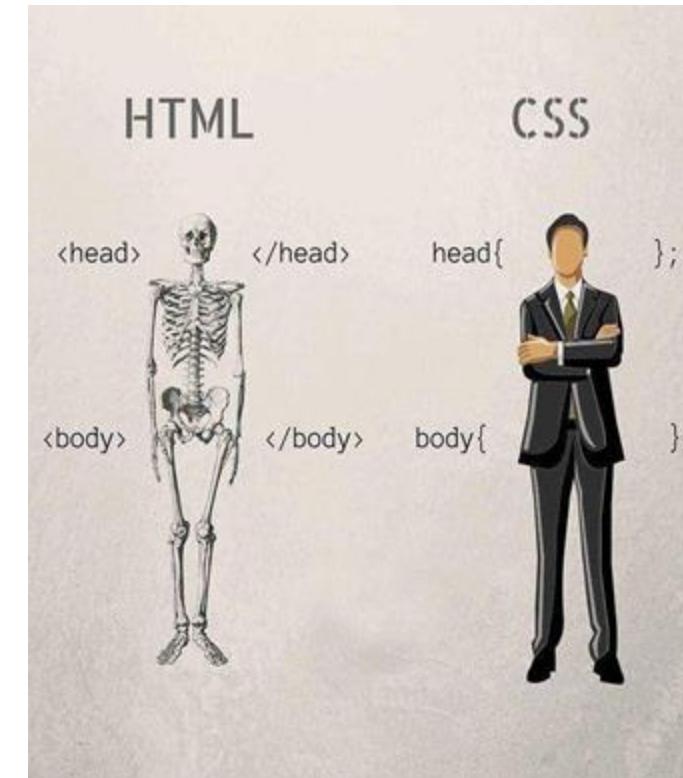
<https://weblab.mit.edu/schedule>

HTML

Hypertext Markup Language

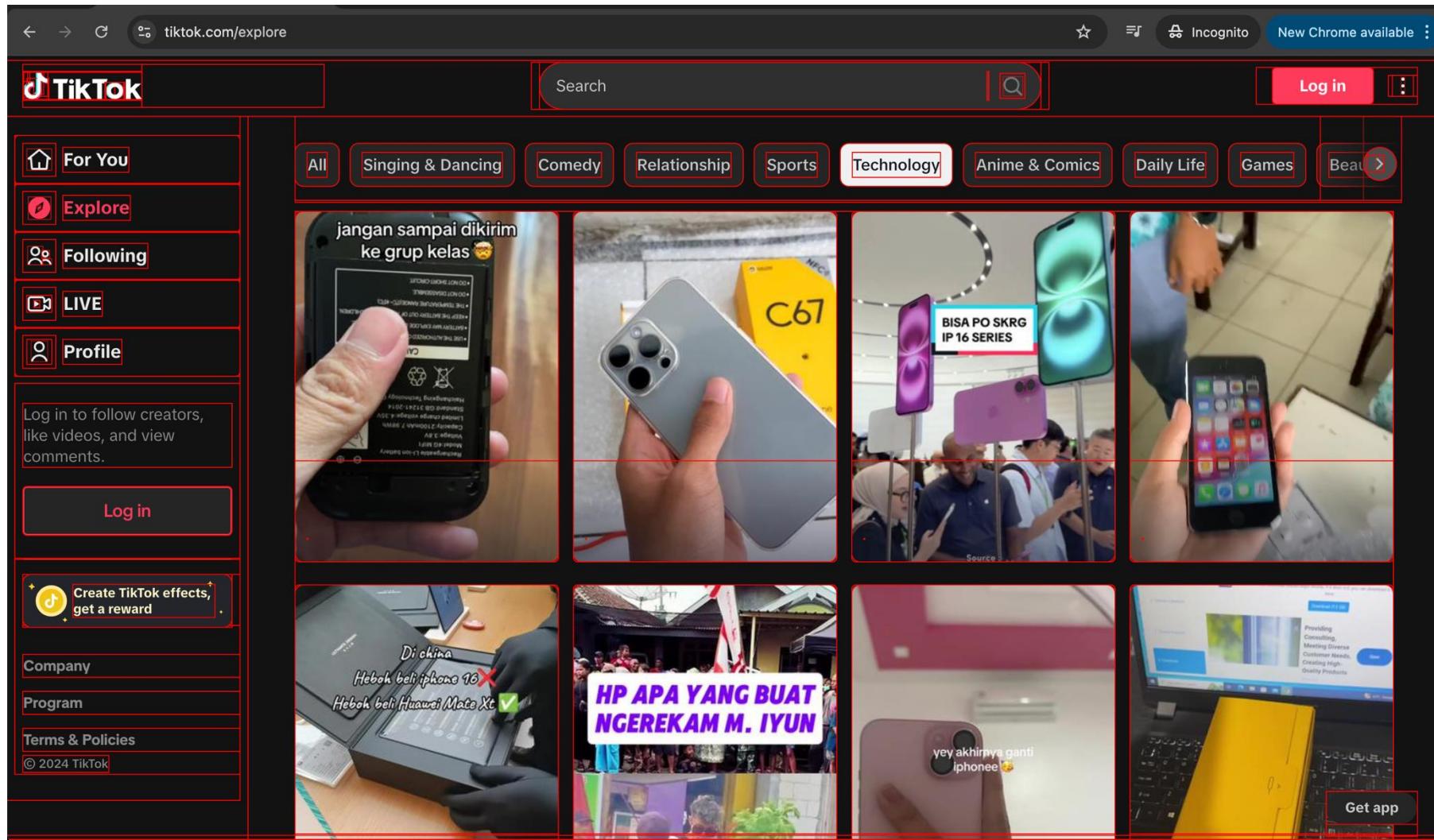
The language your web browser uses to describe the content and structure of web pages

HTML & CSS Analogy



Source: weblab.mit.edu

HTML = Nested Boxes

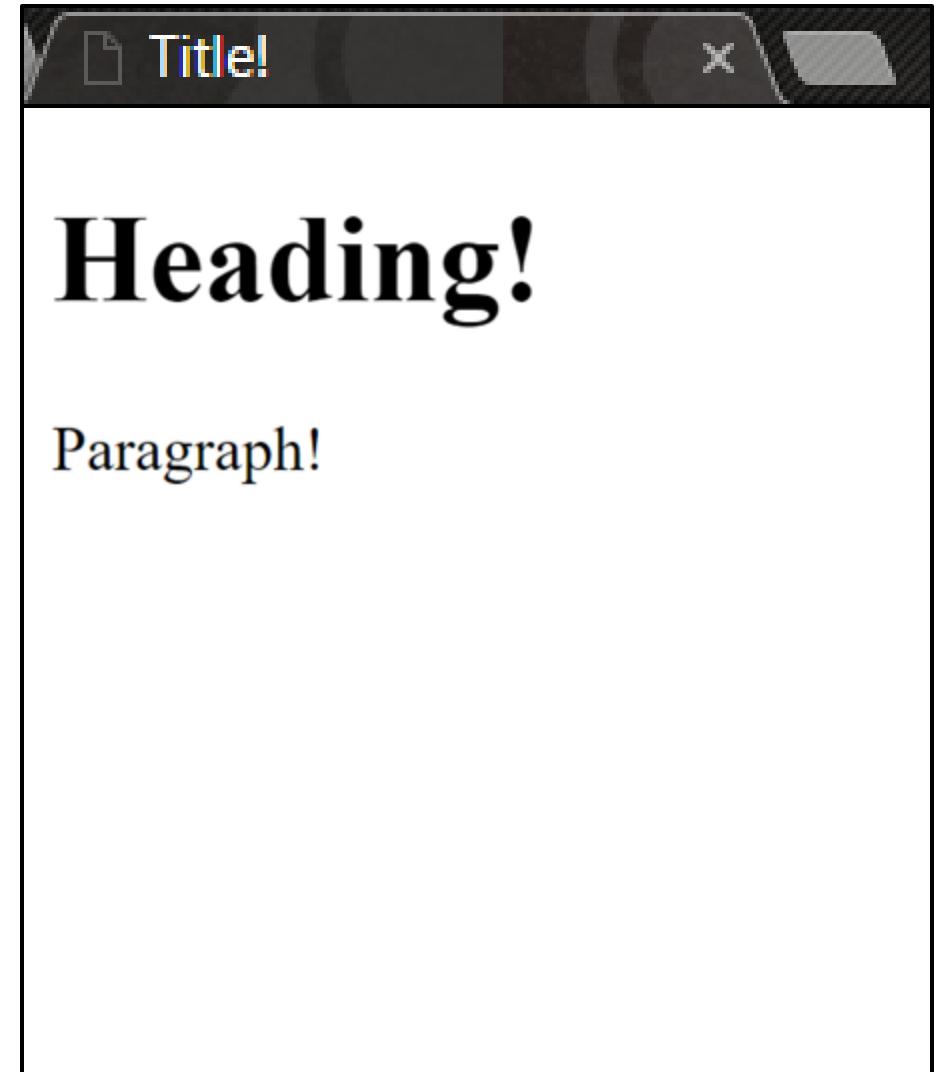


HTML Document Structure

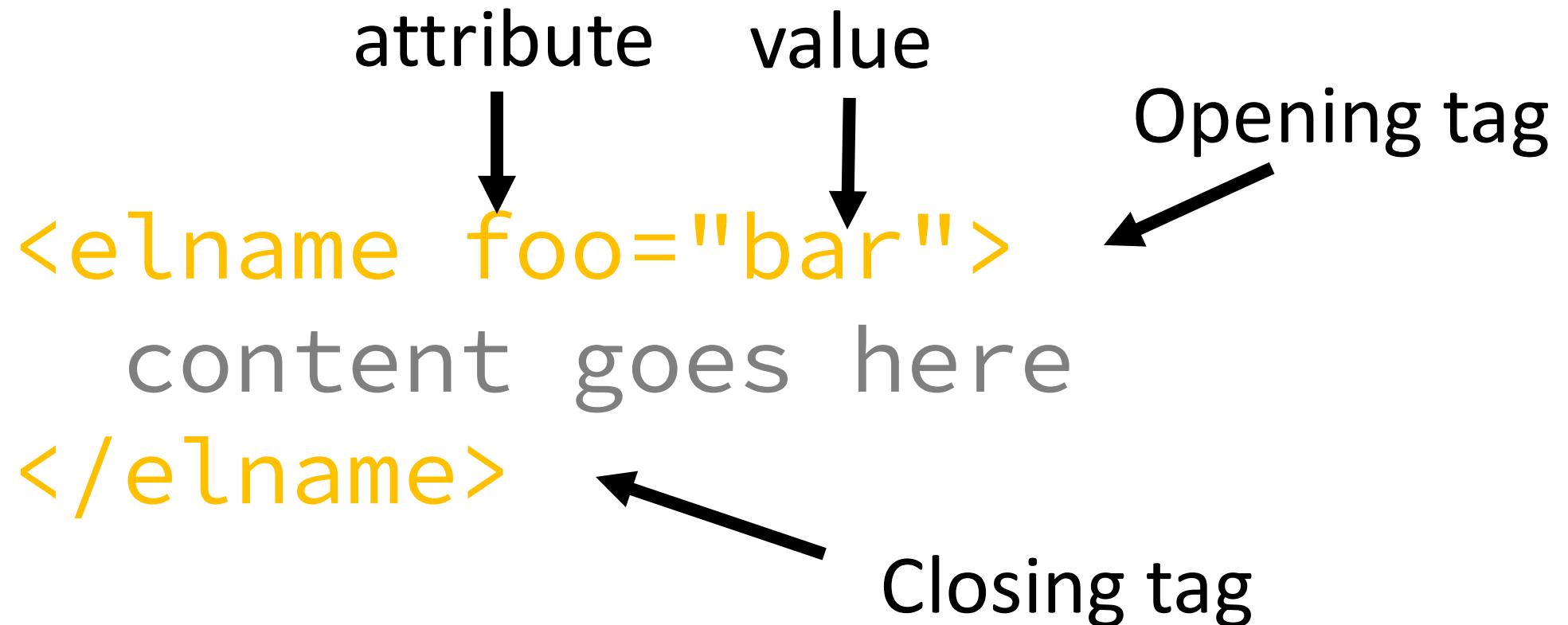
hello.html

this makes sure page uses latest html and
not some random fallback version

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title!</title>
  </head>
  <body>
    <h1>Heading!</h1>
    <p>Paragraph!</p>
  </body>
</html>
```



Anatomy of HTML Element



Elements and tags are *not* the same things. Tags begin or end an element in source code, whereas elements are part of the DOM, the document model for displaying the page in the browser.

Basic HTML Element

html

Root of HTML Document

head

Info about Document

body

Document Body

h1, h2, h3, ...

Header

p

Paragraph

div

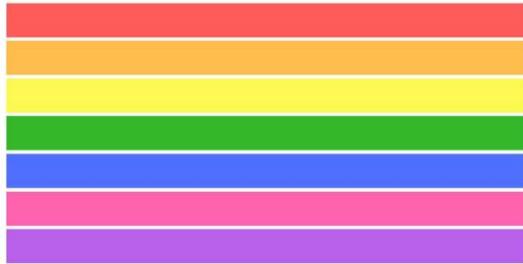
Generic block section

span

Generic inline section

Types: Block vs Inline Element

BLOCK-LEVEL ELEMENTS:



INLINE ELEMENTS:



Block Level Elements

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are: `<p>` and `<div>`.

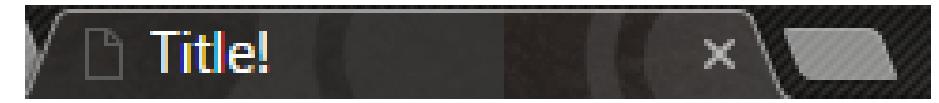
Inline Elements

An inline element does not start on a new line.

An inline element only takes up as much width as necessary.

Note: An inline element cannot contain a block-level element!

Example: Inserting Link



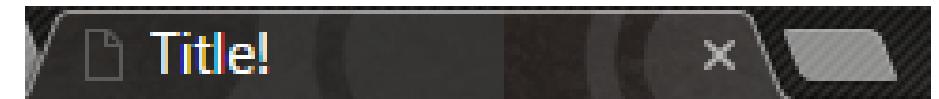
[Link to itb.ac.id!](https://itb.ac.id)

```
<a href="https://itb.ac.id">  
Link to itb.ac.id!</a>
```

Example: Inserting Image

```
</iXng>  
  

```

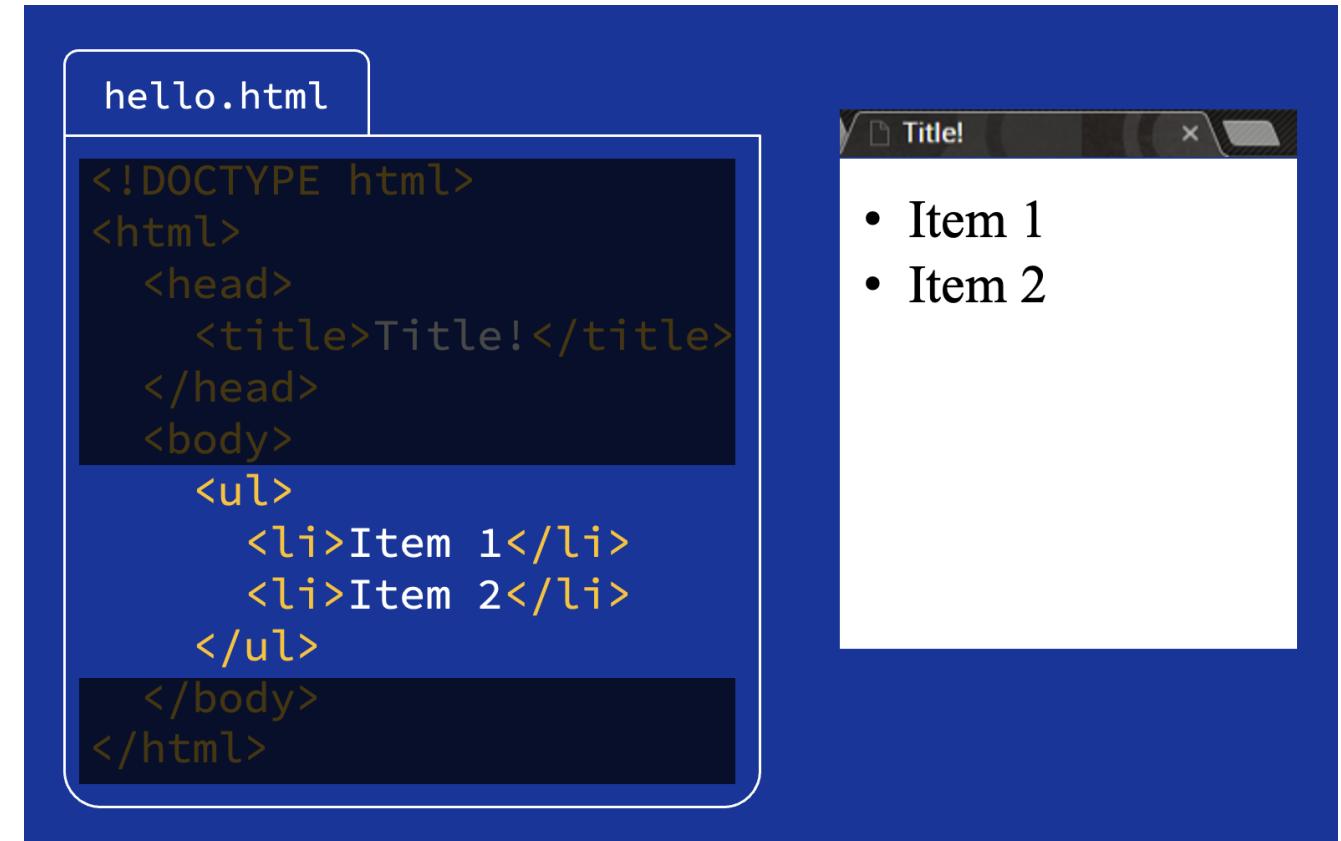


Example: Inserting Lists

 Ordered List (1, 2, 3...)

 Unordered List (bullets)

 List Item



The image shows a code editor on the left and a browser window on the right. The code editor displays the file 'hello.html' with the following content:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title!</title>
  </head>
  <body>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
    </ul>
  </body>
</html>
```

The browser window on the right shows the rendered output: a title bar labeled 'Title!' and a list of two items:

- Item 1
- Item 2

div & span

<div>

Block Section in Document

Inline Section in Document

```
<div id="div-0">
  <h2>Div 1!</h2>
  <p>Paragraph</p>
</div>

<div id="div-1">
  <h2>Div 2!</h2>
  <p>Paragraph!!</p>
</div>

<span id="span-0">This text is inside a span element.</span>
<span id="span-1" span>This text is inside another span element.</span>
```

Div 1!

Paragraph

Div 2!

Paragraph!!

This text is inside a span element. This text is inside another span element.

so many elements...

HTML elements reference

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element>

HTML 5

HTML5 is a markup language used for structuring and presenting content on the **World Wide Web**. It is the fifth and last major HTML version that is a **World Wide Web Consortium (W3C)** recommendation.

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1.

HTML5 standard is a cooperation between the **World Wide Web Consortium (W3C)** and the **Web Hypertext Application Technology Working Group (WHATWG)**.

HTML 5 New Features

- **New Semantic Elements** – These are like `<header>`, `<footer>`, and `<section>`.
- **Forms 2.0** – Improvements to HTML web forms where new attributes have been introduced for `<input>` tag.
- **Persistent Local Storage** – To achieve persistence locally without resorting to third-party plugins.
- **WebSocket** – A next-generation bidirectional communication technology for web applications.
- **Server-Sent Events** – HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

HTML 5 New Features

- **Canvas** – This supports a two-dimensional drawing surface (bitmap) that you can program (manipulate) with JavaScript.
- **Audio & Video** – You can embed audio or video on your webpages without resorting to third-party plugins.
- **Geolocation** – Now visitors can choose to share their physical location with your web application.
- **Microdata** – This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
- **Drag and drop** – Drag and drop the items from one location to another location on the same webpage.

HTML5 Semantic Elements

- **<section>** - This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.
- **<article>** - This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.
- **<aside>** - This tag represents a piece of content that is only slightly related to the rest of the page.
- **<header>** - This tag represents the header of a section.

HTML5 Semantic Elements

- **<footer>** – This tag represents a footer for a section and can contain information about the author, copyright information, et cetera.
- **<nav>** – This tag represents a section of the document intended for navigation.
- **<dialog>** – This tag can be used to mark up a conversation.
- **<figure>** – This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

HTML5 Semantic Tags

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <title>...</title>
  </head>
  <body>
    <header>...</header>
    <nav>...</nav>
    <article>
      <section>
        ...
      </section>
    </article>
    <aside>...</aside>
    <footer>...</footer>
  </body>
</html>
```

Note: `<head>` primarily holds metadata information for machine processing, not human-readability. For human-visible information, like top-level headings and listed authors, see the `<header>` element.

Web Forms 2.0

Web Forms 2.0 is an extension to the forms features found in HTML4. Form elements and attributes in HTML5 provide a greater degree of semantic mark-up than HTML4 and free us from a great deal of tedious scripting and styling that was required in HTML4.

The <input> types element in HTML5

- **datetime** - a date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC
- **datetime-local** - a date and time with no time zone information
- **date** - a date (year, month, day)
- **month** - a date consisting of a year and a month
- **week** - a date consisting of a year and a week number
- **time** - a time (hour, minute, seconds, fractional seconds)

The <input> types element in HTML5

- **number** - accepts only numerical value. The step attribute specifies the precision, defaulting to 1
- **range** - used for input fields that should contain a value from a range of numbers
- **email** - It accepts only email value. This type is used for input fields that should contain an e-mail address.
- **url** - It accepts only URL value. This type is used for input fields that should contain a URL address.

Web Storage

HTML5's **web storage** - feature to store some information locally on the user's computer, similar to cookies, but it is faster and much better than cookies.

HTML5 introduces two mechanisms, similar to HTTP session cookies, for storing structured data on the client side:

- **Session Storage** - designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time.
- **Local Storage** - designed for storage that spans multiple windows, and lasts beyond the current session

Local Storage

```
1 <script>
2 // Check if the localStorage object exists
3 if(localStorage) {
4     // Store data
5     localStorage.setItem("first_name", "Peter");
6
7     // Retrieve data
8     alert("Hi, " + localStorage.getItem("first_name"));
9 } else {
10     alert("Sorry, your browser do not support local storage.");
11 }
12 </script>
```

- **localStorage.setItem(key, value)** stores the value associated with a key.
- **localStorage.getItem(key)** retrieves the value associated with the key.

WebSockets

WebSockets is a next-generation bidirectional communication technology for web applications which operates over a single socket and is exposed via a JavaScript interface in HTML 5 compliant browsers.

WebSockets send data from browser to server by calling a **send()** method, and receive data from server to browser by an **onmessage** event handler.

WebSocket Events

Event	Event Handler	Description
open	Socket.onopen	This event occurs when socket connection is established.
message	Socket.onmessage	This event occurs when client receives data from server.
error	Socket.onerror	This event occurs when there is any error in communication.
close	Socket.onclose	This event occurs when connection is closed.

WebSockets

```
1 let socket = new WebSocket("wss://javascript.info/article/websocket/chat/ws");
2
3 // send message from the form
4 document.forms.publish.onsubmit = function() {
5     let outgoingMessage = this.message.value;
6
7     socket.send(outgoingMessage);
8     return false;
9 };
10
11 // message received - show the message in div#messages
12 socket.onmessage = function(event) {
13     let message = event.data;
14
15     let messageElem = document.createElement('div');
16     messageElem.textContent = message;
17     document.getElementById('messages').prepend(messageElem);
18 }
```

Server-Sent Events

- Using **Server-Sent Events (SSE)** you can push DOM events continuously from your web server to the visitor's browser.
- The event streaming approach opens a persistent connection to the server, sending data to the client when new information is available, eliminating the need for continuous polling.
- It allows a web page to hold an open connection to a web server so that the web server can send a new response automatically at any time, there's no need to reconnect, and run the same server script from scratch over and over again.

Server-Sent Events

```
1 <?php
2 header("Content-Type: text/event-stream");
3 header("Cache-Control: no-cache");
4
5 // Get the current time on server
6 $currentTime = date("h:i:s", time());
7
8 // Send it in a message
9 echo "data: " . $currentTime . "\n\n";
10 flush();
11 ?>
```

Server-Sent Events

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <title>Using Server-Sent Events</title>
5  <script>
6      window.onload = function() {
7          var source = new EventSource("server_time.php");
8          source.onmessage = function(event) {
9              document.getElementById("result").innerHTML += "New time received from
web server: " + event.data + "<br>";
10         };
11     };
12 </script>
13 </head>
14 <body>
15     <div id="result">
16         <!--Server response will be inserted here-->
17     </div>
18 </body>
19 </html>
```

Canvas

- HTML5 element <canvas> gives you an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.

Canvas

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <title>Drawing a Rectangle on the Canvas</title>
6 <script>
7     window.onload = function() {
8         var canvas = document.getElementById("myCanvas");
9         var context = canvas.getContext("2d");
10        context.rect(50, 50, 200, 100);
11        context.stroke();
12    };
13 </script>
14 </head>
15 <body>
16     <canvas id="myCanvas" width="300" height="200"></canvas>
17 </body>
18 </html>
```

SVG

The Scalable Vector Graphics (SVG) is an XML-based image format that is used to define two-dimensional vector based graphics for the web. Unlike raster image (e.g. .jpg, .gif, .png, etc.), a vector image can be scaled up or down to any extent without losing the image quality.

SVG

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>Embedding SVG in HTML</title>
6 </head>
7 <body>
8   <svg width="300" height="200">
9     <text x="10" y="20" style="font-size:14px;">
10       Your browser support SVG.
11     </text>
12     Sorry, your browser does not support SVG.
13   </svg>
14 </body>
15 </html>
```

Differences Canvas vs SVG

Canvas	SVG
Resolution dependent	Resolution independent
No support for event handlers	Support for event handlers
Poor text rendering capabilities	Best suited for applications with large rendering areas (Google Maps)
You can save the resulting image as .png or .jpg	Slow rendering if complex (anything that uses the DOM a lot will be slow)
Well suited for graphic-intensive games	Not suited for game applications

Audio & Video

The HTML5 <audio> and <video> tags make it simple to add media to a website. You need to set src attribute to identify the media source and include a controls attribute so the user can play and pause the media.

```
1 <video controls="controls">
2   <source src="media/shuttle.mp4" type="video/mp4">
3   <source src="media/shuttle.ogv" type="video/ogg">
4   Your browser does not support the HTML5 Video element.
5 </video>
```

Geolocation

The HTML5 **geolocation** feature lets you find out the geographic coordinates (latitude and longitude numbers) of the current location of your website's visitor.

It utilizes the three methods that are packed into the `navigator.geolocation` object – **getCurrentPosition()**, **watchPosition()** and **clearWatch()**.

Properties of position object

Property	Type	Notes
coords.latitude	double	Decimal degrees
coords.longitude	double	Decimal degrees
coords.altitude	double or null	Meters above the reference ellipsoid
coords.accuracy	double	Meters
coords.altitudeAccuracy	double or null	Meters
coords.heading	double or null	Degrees clockwise from true north
coords.speed	double or null	Meters/second
timestamp	DOMTimeStamp	Like a Date() object

Geolocation

```
<script>
var x = document.getElementById("demo");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}

function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
</script>
```

Microdata

- **Microdata** is a standardized way to provide additional semantics in your web pages.
- **Microdata** lets you define your own customized elements and start embedding **custom properties** in your web pages. At a high level, microdata consists of a **group of name-value pairs**.
- The groups are called **items**, and each name-value pair is a property. Items and properties are represented by regular elements.
- If a browser supports the HTML5 **microdata API**, there will be a **getItems()** function on the global **document** object.

Microdata Global Attributes

Attribute	Description
<i>itemscope</i>	Defines the scope of microdata item.
<i>itemprop</i>	Defines the name/value pairs of the microdata.
<i>itemtype</i>	A URL to define the vocabulary used for encoding the microdata.
<i>itemid</i>	To set an unique identifier for microdata item.
<i>itemref</i>	To include itemprop attributes outside the itemscope attribute.

Microdata

```
<div itemscope itemtype="http://schema.org/SoftwareApplication">
  <span itemprop="name">Angry Birds</span> -
  REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
  <link itemprop="applicationCategory" href="http://schema.org/GameApplication"/>

  <div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
    RATING:
    <span itemprop="ratingValue">4.6</span> (
    <span itemprop="ratingCount">8864</span> ratings )
  </div>

  <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
    Price: $<span itemprop="price">1.00</span>
    <meta itemprop="priceCurrency" content="USD" />
  </div>
</div>
```

Drag & drop

- **Drag and Drop** (DnD) is powerful User Interface concept which makes it easy to copy, reorder and deletion of items with the help of mouse clicks.
- In HTML5, drag and drop is part of the standard: Any element can be draggable

Drag & drop

Event	Description
ondragstart	Fires when the user starts dragging an element.
ondragenter	Fires when a draggable element is first moved into a drop listener.
ondragover	Fires when the user drags an element over a drop listener.
ondragleave	Fires when the user drags an element out of drop listener.
ondrag	Fires when the user drags an element anywhere; fires constantly but can give X and Y coordinates of the mouse cursor.
ondrop	Fires when the user drops an element into a drop listener successfully.
ondragend	Fires when the drag action is complete, whether it was successful or not. This event is not fired when dragging a file to the browser from the desktop.

Web Worker

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page

It continues to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

Offline Support for Web App

Use **Service Worker** to fetch contents/files and store data in **local storage**

A **Service Worker** is a script that executes in the background, in a separate thread from the browser UI. Service worker cache makes it possible for a web site to function offline. They are the technical powerhouse that levels up a website to a progressive web app. They enable deep platform integration, like rich caching, push notifications and background sync.

A **Service worker** sits between the browser and the network, acting like a proxy server, handling a collection of non-UI centric tasks. They are event driven and live outside the browser process, enabling them to work without an active browser session.

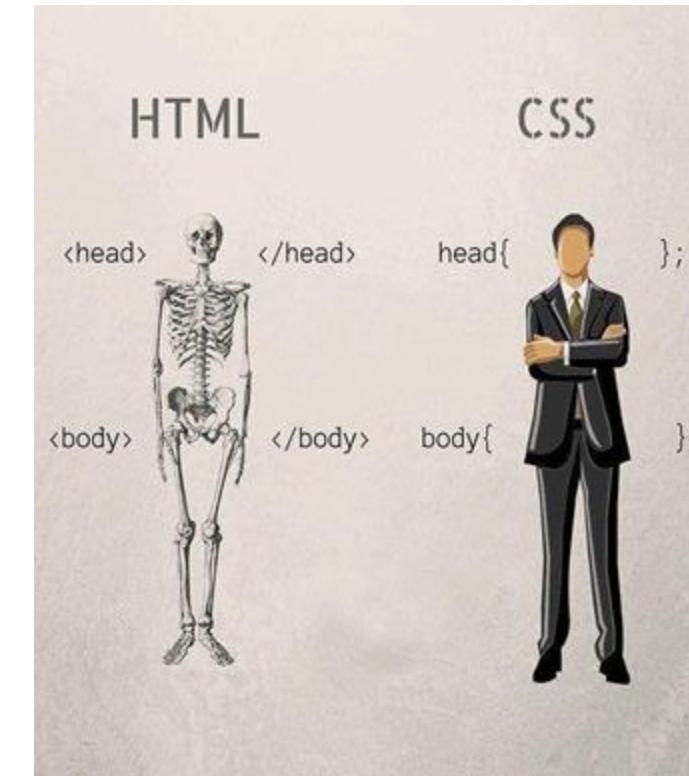
Definition

CSS

Cascading Style Sheets

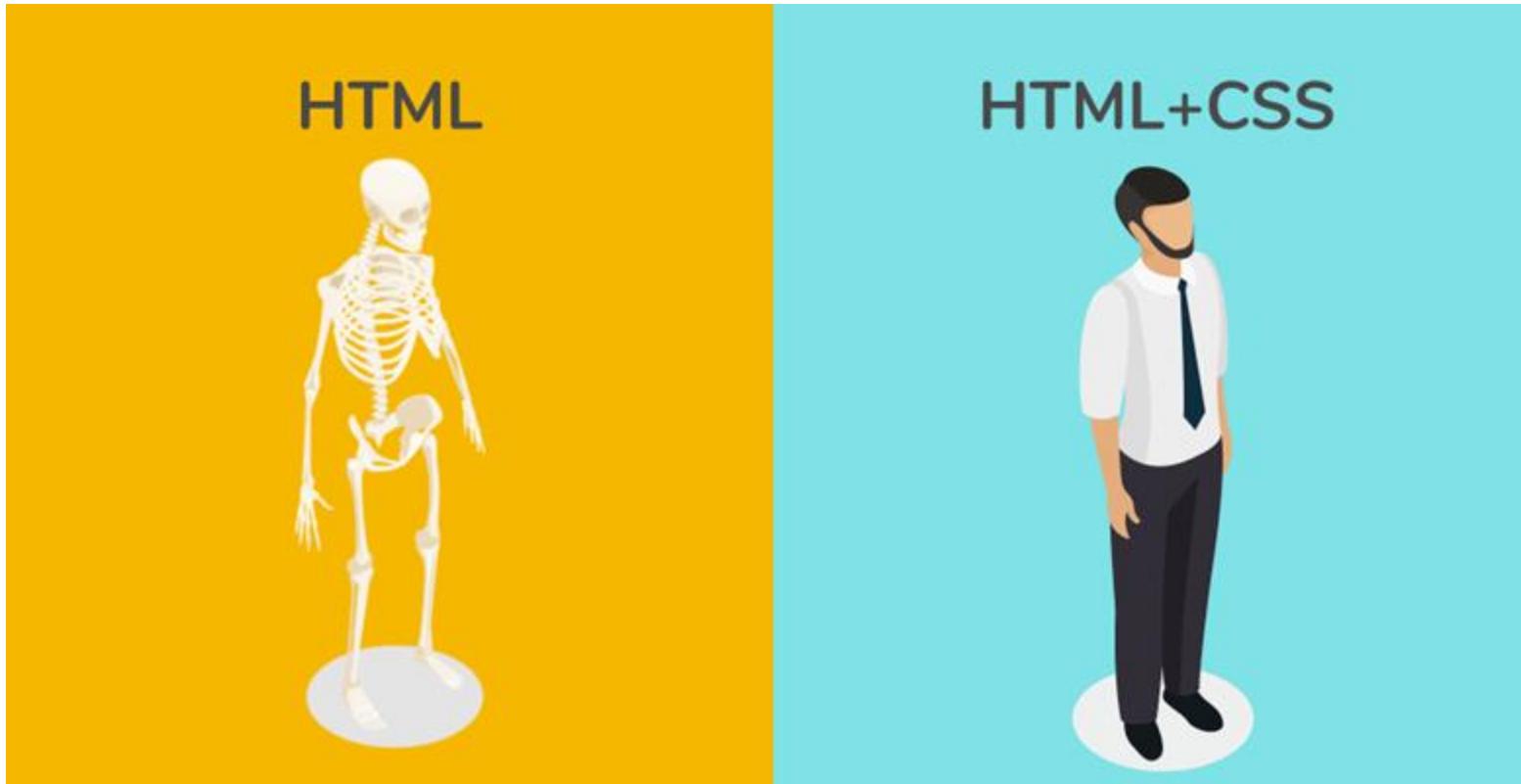
The **rules** that tell your **web browser** **how stuff looks**

HTML & CSS Analogy



Source: weblab.mit.edu

CSS = A list of descriptions

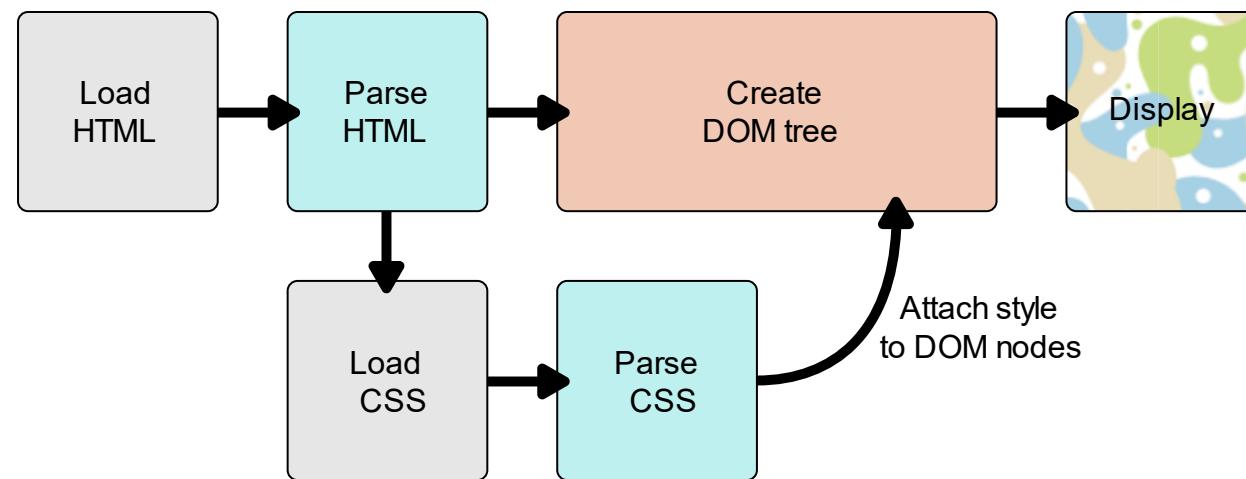


making things look pretty

CSS is used to provide descriptions or rules on how HTML elements should be displayed in the browser.

How does CSS actually work?

When a browser displays a webpage, it merges the content with its styling information. The browser goes through several key steps, and different browsers may handle the process slightly differently. The following diagram provides a simplified illustration of the process.

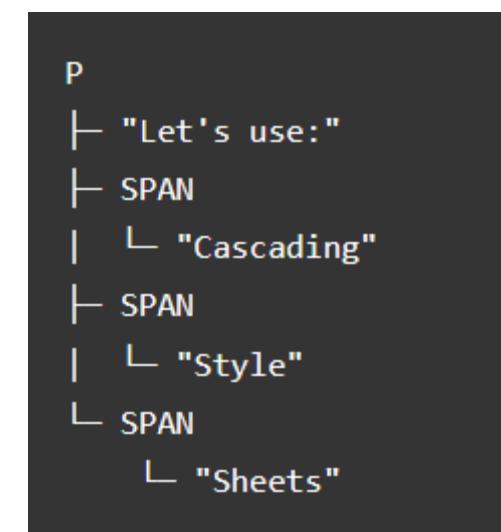


CSS & DOM

- A DOM has a tree-like structure. Each element, attribute, and piece of text in the markup language becomes a [DOM node](#) in the tree structure. The nodes are defined by their relationship to other DOM nodes. Some elements are parents of child nodes, and child nodes have siblings.
- When CSS is applied, the styles are associated with the DOM nodes, determining how each element will look on the page, such as its color, size, and position, without altering the DOM structure itself. This styling process helps shape the visual presentation of the document.

Example:

```
<p>
  Let's use:
  <span>Cascading</span>
  <span>Style</span>
  <span>Sheets</span>
</p>
```



Applying CSS to the DOM

Example:

```
<p>
  Let's use:
    <span>Cascading</span>
    <span>Style</span>
    <span>Sheets</span>
</p>
```

```
span {
  border: 1px solid black;
  background-color: lime;
}
```

Result:

Let's use: Cascading Style Sheets

The CSS Cascade

The cascade is the algorithm for solving conflicts where multiple CSS rules apply to an HTML element.

```
button {  
  color: red;  
}  
  
button {  
  color: blue;  
}
```

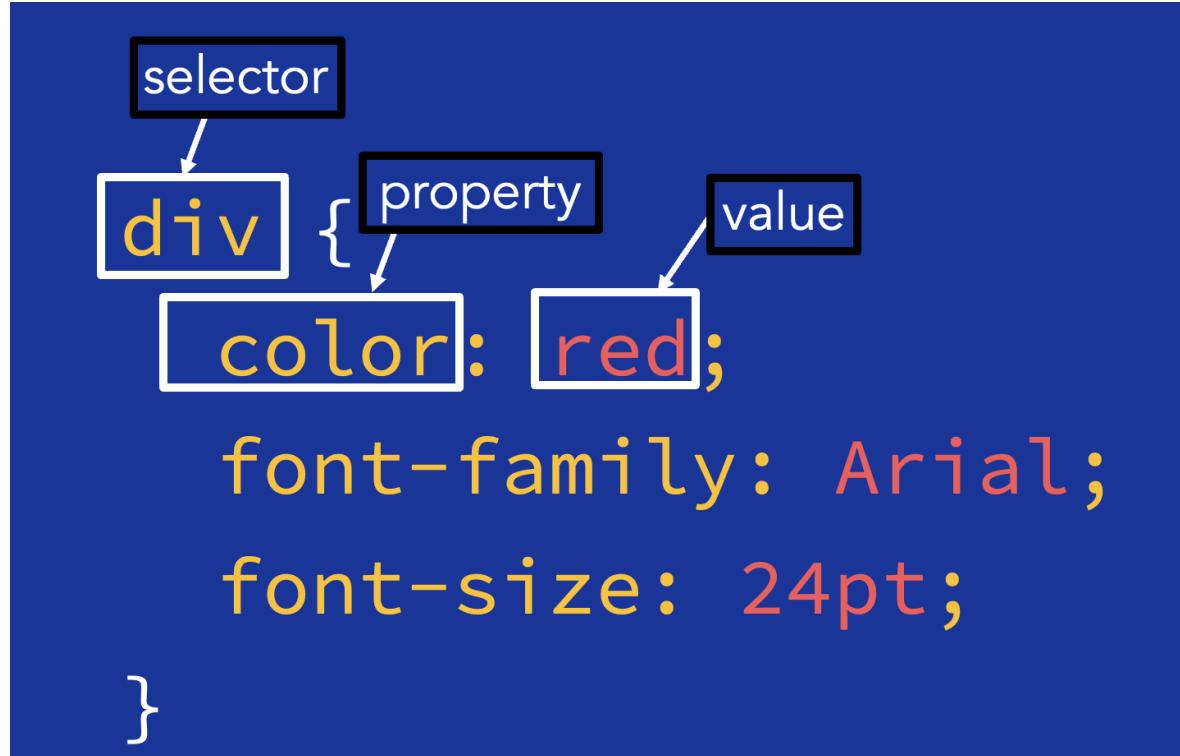
Hello, I have blue text

Understanding the cascade algorithm helps you understand how the browser resolves conflicts like this. The cascade algorithm is split into 4 distinct stages.

- 1. Position and order of appearance:** the order of which your CSS rules appear
- 2. Specificity:** an algorithm which determines which CSS selector has the strongest match
- 3. Origin:** the order of when CSS appears and where it comes from, whether that is a browser style, CSS from a browser extension, or your authored CSS
- 4. Importance:** some CSS rules are weighted more heavily than others, especially with the !important rule type

<https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade>

CSS ruleset (rule)



List of selectors, properties :
<https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

Example 1: Using Element Selector

`hello.html`

```
<h1>Heading!</h1>
<div>Paragraph!</div>
<div>Info</div>
```

`style.css`

```
div {
    color: red;
    font-family: Arial;
    font-size: 24pt;
}
```

A screenshot of a web browser window titled "Title!". The page displays three elements: a large bold black "Heading", a red "Paragraph!", and a red "Info". The browser interface includes a toolbar at the top with icons for file operations and a scroll bar on the right side of the content area.

Example 2: Using Class Selector

hello.html

```
<h1>Heading!</h1>
<div>Paragraph!</div>
<div class="info">Info</div>
```

style.css

```
.info {
    color: red;
    font-family: Arial;
    font-size: 24pt;
}
```

The screenshot shows a browser window titled "Title!". Inside, there is a large black heading "Heading". Below it is a smaller black paragraph "Paragraph!". At the bottom of the page, the word "Info" is displayed in red and in a larger, bold black font.

Example 3: Using ID Selector

hello.html

```
<h1>Heading!</h1>
<div>Paragraph!</div>
<div id="unique">Info</div>
```

style.css

```
#unique {
    color: red;
    font-family: Arial;
    font-size: 24pt;
}
```

The screenshot shows a browser window with a title bar labeled "Title!". The main content area displays three pieces of text: a large black "Heading", a smaller black "Paragraph!", and the word "Info" in red. This visual output corresponds to the rendered HTML and CSS shown on the left.

ID vs Class

ID

- An element can have only one ID
- IDs must be unique in any given HTML document

```
#id {
```

...

```
}
```

Class

- An element can have multiple classes
- Can use the same class on multiple elements

```
.classname {
```

...

```
}
```

CSS Specificity

If there are two or more CSS rules that point to the same element, the selector with the highest specificity value will "win", and its style declaration will be applied to that HTML element.

Think of specificity as a score/rank that determines which style declaration is ultimately applied to an element.

Example 1:

```
<html>
<head>
<style>
  p {color: red;}
</style>
</head>
<body>

<p>Hello World!</p>

</body>
</html>
```

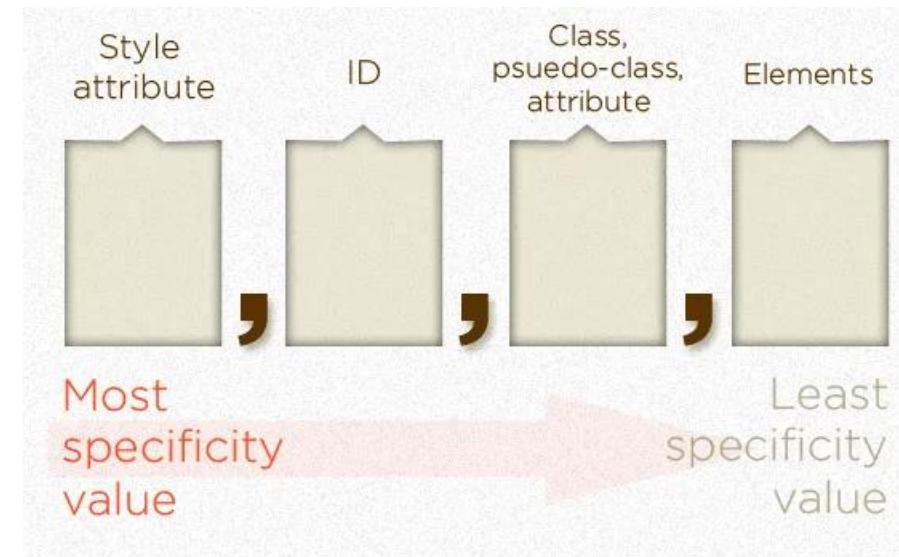
Example 2:

```
<html>
<head>
<style>
  .test {color: green;}
  p {color: red;}
</style>
</head>
<body>

<p class = "test">Hello World!</p>

</body>
</html>
```

CSS Specificity (2)

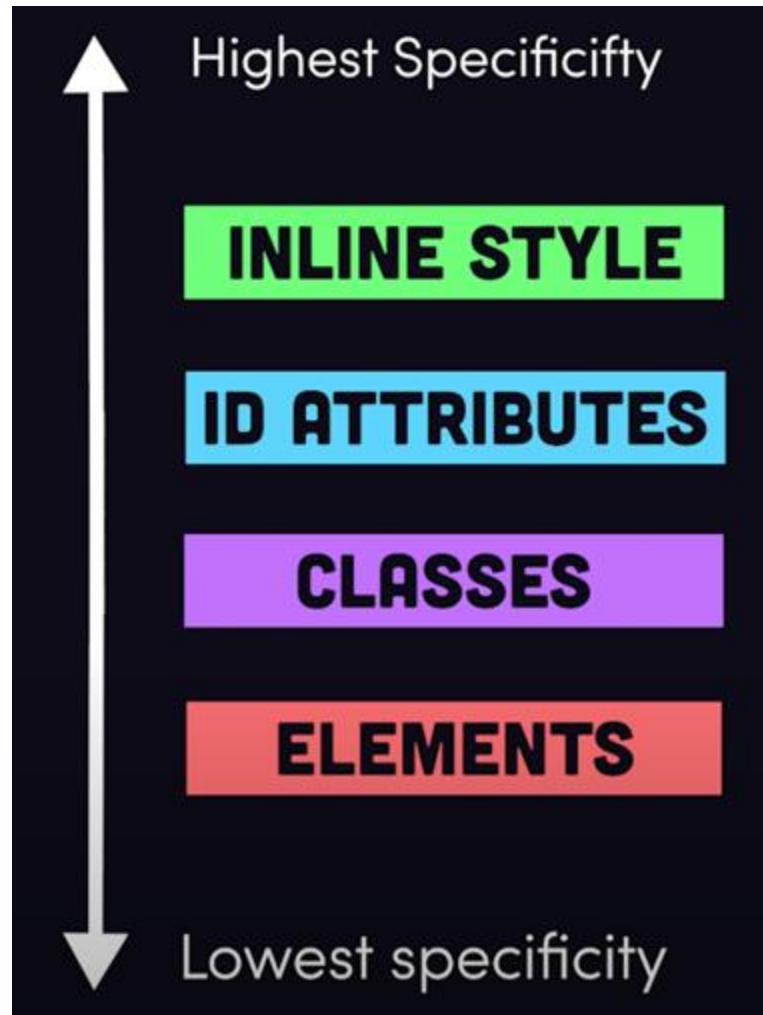


Start at 0, add 100 for each ID value, add 10 for each class value (or pseudo-class or attribute selector), add 1 for each element selector or pseudo-element.

Note 1: Inline style gets a specificity value of 1000, and is always given the highest priority!

Note 2: There is one exception to this rule: if you use the [!important](#) rule, it will even override inline styles!

CSS Hierarchy



read more about **specificity** of selectors [here](#)

Only use classes for CSS styling!



CSS !important

- The **!important** rule in CSS is used to add more importance to a property/value than normal. In fact, if you use the **!important** rule, it will override ALL previous styling rules for that specific property on that element!
- The only way to override an **!important** rule is to include another **!important** rule on a declaration with the same (or higher) specificity in the source code

Example : Using !important

```
#myid {  
    background-color: blue;  
}
```

```
.myclass {  
    background-color: gray;  
}
```

```
p {  
    background-color: red !important;  
}
```

```
<body>  
<p>Hello World!</p>  
<p class="myclass">Hello World!</p>  
<p id="myid" >Hello World!</p>  
</body>
```

```
#myid {  
    background-color: blue; !important;  
}
```

```
.myclass {  
    background-color: gray; !important;  
}
```

```
p {  
    background-color: red !important;  
}
```

```
<body>  
<p>Hello World!</p>  
<p class="myclass">Hello World!</p>  
<p id="myid" >Hello World!</p>  
</body>
```

CSS Combinators

Specifies relationship between CSS selectors. Examples of selectors include HTML tags, such as **div** or **p**

We have 4 different CSS combinator:

- descendant selector (space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~)

HTML Example

Take a look at an example skeleton in the HTML section, followed by some code in the CSS section.



HTML

```
<div class="container">
  <section class="child" id="c1">
    <div>subchild 1</div>
  </section>
  <div class="child" id="c2">child 2</div>
  <div class="child" id="c3">child 3</div>
  <div class="child" id="c4">child 4</div>
</div>
```



CSS

```
.container {  
}  
}
```

Descendant selector (space)

Matches all elements that are **descendants of the specified element**

- In the below example, we're selecting all the **div** elements that are descendants of the **.container** element → can be deeply nested (**subchild 1 is also selected**)

```
...          CSS  
  
.container div {  
    font-size: 20px;  
}
```

subchild 1
child 2
child 3
child 4

Child selector (>)

Matches all elements that are **direct children of the specified element**

- In the below example, only the elements denoted as **child** will be selected → further levels of nesting will not be selected

```
...          CSS  
  
.container > div {  
  font-size: 20px;  
}
```

subchild 1
child 2
child 3
child 4

Adjacent Sibling Selector (+)

Selects a *single* element that is **directly after another specific element**

- In the below example, the **element directly after the element with an id of “c1”** is selected

```
...  
css  
  
#c1 + div {  
    color: red;  
}
```

subchild 1
child 2
child 3
child 4

General Sibling Selector (~)

Selects *all* elements **after another specific element**

- In the below example, **all elements after the element with an id of “c1”** are selected

```
...  
css  
  
#c1 ~ div {  
    color: red;  
}
```

subchild 1
child 2
child 3
child 4

CSS Layout

CSS page layout techniques enable precise control over the positioning of elements within a webpage. These elements can be positioned based on several factors, including their default position in the normal layout flow, their relation to surrounding elements, their parent container, and the main viewport or window

Some CSS layout techniques:

- [Normal flow](#)
- [The display property](#)
- [Flexbox](#)
- [Grid](#)
- [Floats](#)
- [Positioning](#)
- [Table layout](#)
- [Multiple-column layout](#)

Display Types

The **display** property allows us to tell the browser how to display an element and its children on the page.

We've looked at:

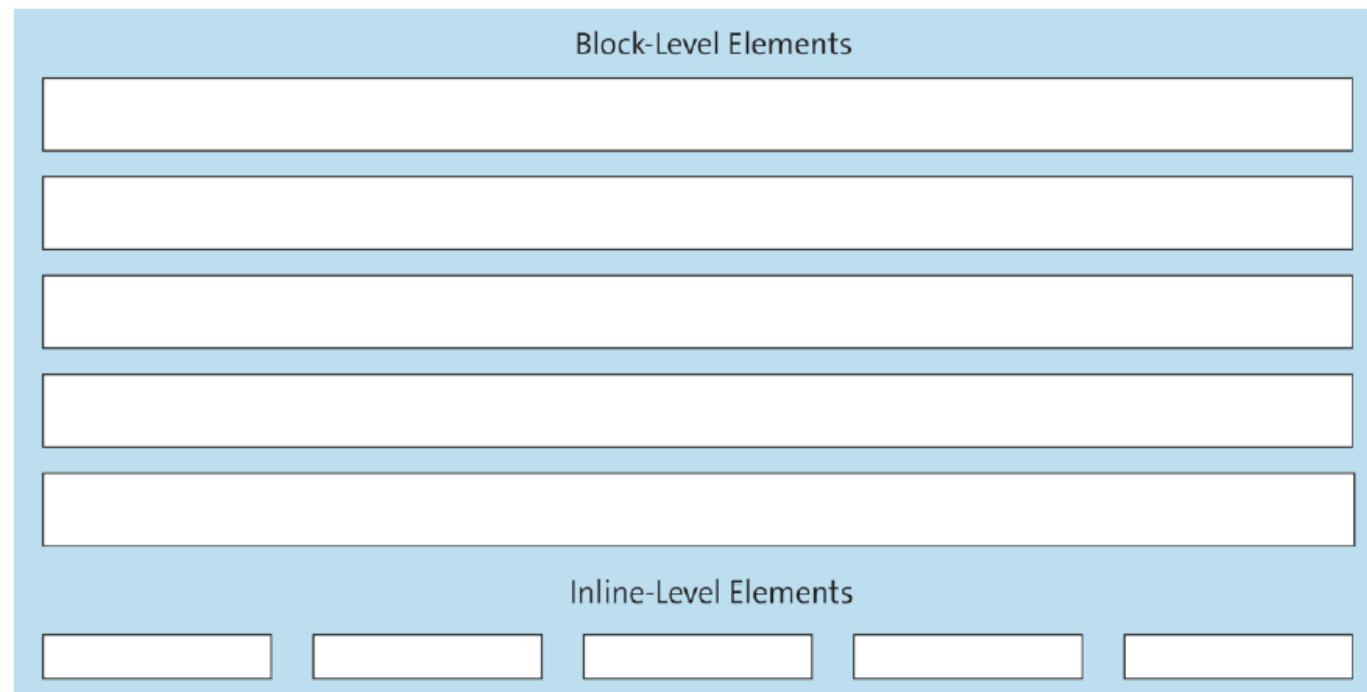
- **block**
- **flex**

Values we'll be looking at:

- **grid**
- **none**

Box model

- HTML categorizes elements as either block-level or inline-level. Block-level elements start on a new line, stacking vertically, while inline-level elements fit into the flow of text, lining up horizontally. With CSS's display property, we can alter these behaviors: setting display: **block** treats the element as block-level, and display: **inline** makes it inline-level.

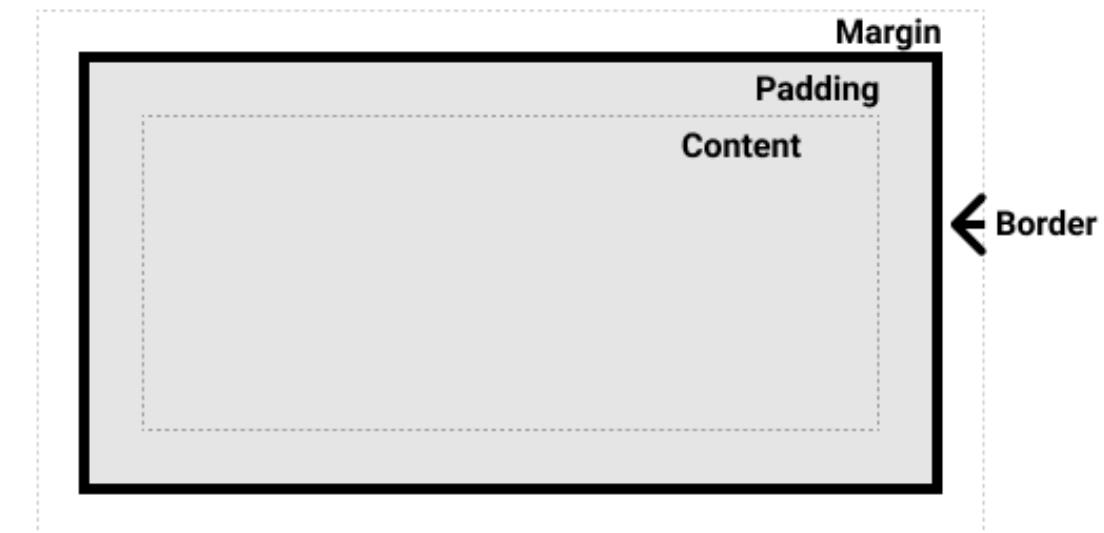


Css box model

- The CSS box model as a whole applies to block boxes and defines how the different parts of a box – margin, border, padding, and content – work together to create a box that you can see on a page. Inline boxes use just some of the behavior defined in the box model.

Parts of a box in CSS:

- Content box: The area where the content is displayed. You can size it with **width**, **height**, or similar properties.
- Padding box: The space around the content. Set its size using **padding**.
- Border box: The layer around the content and padding. Size it with **border** properties.
- Margin box: The outermost space between the element and others. Adjust it using **margin**.



HTML Example

Take a look at an example skeleton in the HTML section, followed by some code in the CSS section.



HTML

```
<div class="container">  
  <div class="child" id="c1">child 1</div>  
  <div class="child" id="c2">child 2</div>  
  <div class="child" id="c3">child 3</div>  
  <div class="child" id="c4">child 4</div>  
</div>
```



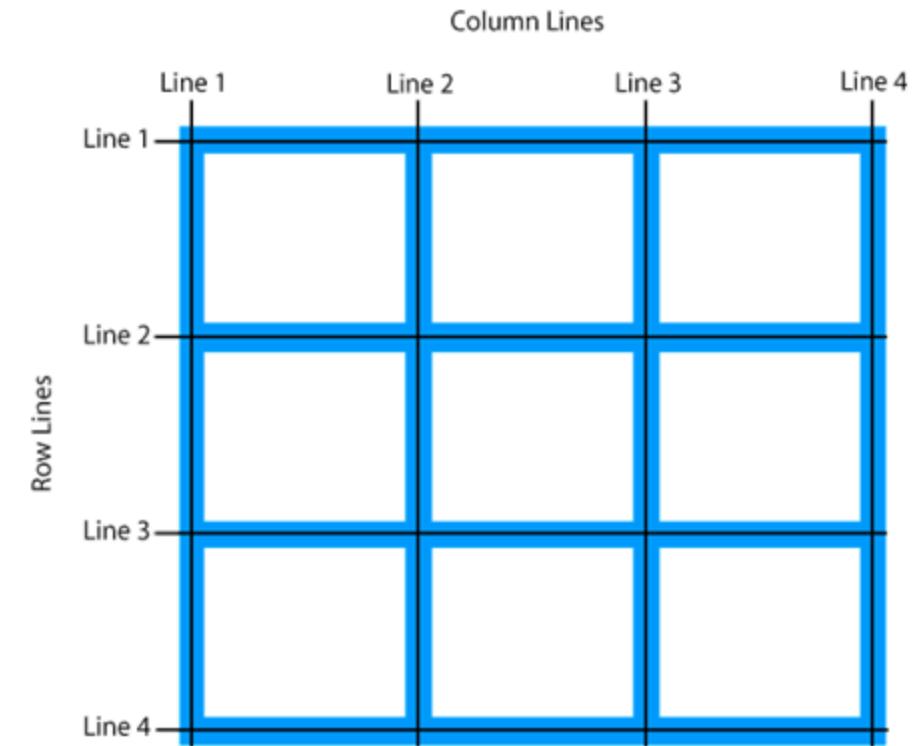
CSS

```
.container {  
  
}
```

display: grid

Tells the browser to display child elements in a two-dimensional grid layout

```
...  
CSS  
  
.container {  
    display: grid;  
}
```



grid-auto-flow

auto-flow gives us the power to tell the browser how to automatically handle child elements that reach the end of our grid layout

row instructs the browser to prioritize adding rows

- By default, **grid** will prioritize rows

columns instructs the browser to prioritize adding columns

grid-auto-flow



CSS

```
.container {  
  display: grid;  
  grid-auto-flow: row;  
}
```



CSS

```
.container {  
  display: grid;  
  grid-auto-flow: column;  
}
```

child 1
child 2
child 3
child 4

child 1	child 2	child 3	child 4
---------	---------	---------	---------

Flexbox

[Flexbox](#) is a one-dimensional layout method for arranging items in rows or columns. Items *flex* (expand) to fill additional space or shrink to fit into smaller spaces. This article explains all the fundamentals.

CSS flexible box layout enables you to:

- Vertically center a block of content inside its parent.
- Make all the children of a container take up an equal amount of the available width/height, regardless of how much width/height is available.
- Make all columns in a multiple-column layout adopt the same height even if they contain a different amount of content.

Example : flexbox

```
section {
```

```
  display: flex;
```

```
}
```

Sample flexbox example

First article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

Second article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

Third article

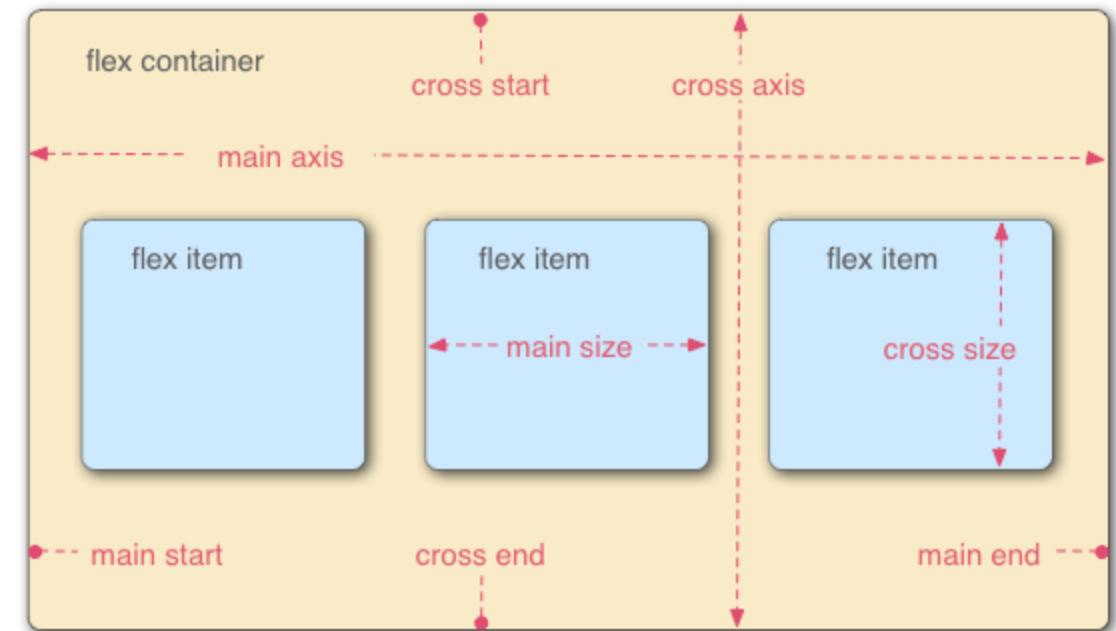
Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

Cray food truck brunch, XOXO +1 keffiyeh pickled chambray waistcoat ennui. Organic small batch paleo 8-bit. Intelligentsia umami wayfarers pickled, asymmetrical kombucha letterpress kitsch leggings cold-pressed squid chartreuse put a bird on it. Listicle pickled man bun cornhole heirloom art party.

flex model

When elements are laid out as flex items, they are laid out along two axes:

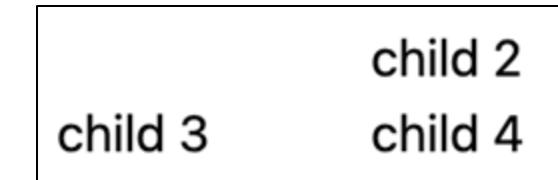
- The main axis is the direction in which the flex items are arranged (like a row or column). The start and end points of this axis are called main start and main end, and the length between them is the main size.
- The cross axis runs perpendicular to the main axis, with start and end points called cross start and cross end, and the length between them is the cross size.
- The element with `display: flex` is the flex container.
- The elements inside the flex container are called flex items.



display: none

Tells the browser to remove an element from the document
→ doesn't take up any space in the layout

- This is different from **visibility: hidden**, which hides the element but still takes up space in our layout



css

```
#c1 {  
  display: none;  
}
```

css

```
#c1 {  
  visibility: hidden;  
}
```

Content Overflow

The **overflow** property allows us to tell the browser how to handle child elements that may exceed the size of a parent element.

Values we'll be looking at:

- visible
- hidden
- scroll
- auto

overflow: visible

Default behavior → tells the browser to display the overflowing content

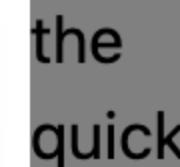
```
...          CSS  
  
.container {  
    width: 50px;  
    height: 50px;  
    background-color: gray;  
    overflow: visible;  
}
```

the
quick
brown
fox
jumps
over
the
lazy
dog

overflow: hidden

Tells the browser to clip the overflowing content → cannot scroll within the parent element.

```
...          CSS  
  
.container {  
    width: 50px;  
    height: 50px;  
    background-color: gray;  
    overflow: hidden;  
}
```



the
quick

overflow: auto

Tells the browser to display a scrollbar for the parent element if needed → this scrollbar will only be present if there is any overflowing content.

```
...  
          CSS  
  
.container {  
  width: 50px;  
  height: 50px;  
  background-color: gray;  
  overflow: auto;  
}
```

overflow content



no overflow content



Exercise: Membuat Web Portofolio Pribadi

Buat halaman web portfolio pribadi sebagai representasi digital dari karya, keterampilan, dan identitas Anda menggunakan HTML dan CSS dari awal (from scratch), tidak diperbolehkan menggunakan template atau CSS framework apapun atau Generative AI.

Kriteria dan Tantangan Utama:

1. Gunakan Struktur HTML yang Semantic
2. Desain Responsif
4. Kreativitas dalam Penggunaan Warna dan Tipografi

Bonus:

3. Desain yang Interaktif (without JavaScript, only HTML+CSS)
5. Aksesibilitas Web
6. Optimisasi Kinerja

Deploy web anda pada penyedian layanan hosting web statis (Netlify, Vercel, Static.app, Github Pages, etc...)

Link dikumpulkan melalui Edunex kelas Parent dan berikan alasan atas keputusan desain yang ada ambil untuk kriteria-kriteria tersebut.

Exercise: Penjelasan Kriteria Lanjutan

Struktur HTML yang Semantik:

- Buatlah struktur halaman yang sesuai dengan standar HTML5, menggunakan elemen-elemen semantik seperti `<header>`, `<nav>`, `<section>`, `<article>`, `<aside>`, dan `<footer>`.
- Jelaskan alasan penggunaan elemen-elemen tersebut dalam konteks portfolio Anda (misalnya, mengapa Anda menggunakan `<section>` pada bagian tertentu dan bukan `<div>` biasa?).

Desain Responsif:

- Gunakan media queries untuk membuat halaman web yang responsif terhadap berbagai ukuran layar (desktop, tablet, dan smartphone).
- Mahasiswa harus menguji dan mempresentasikan hasil tampilan dari halaman yang telah mereka buat di tiga ukuran layar berbeda.
- Tantangan: Pastikan layout tetap menarik dan fungsional pada semua ukuran layar, serta mempertimbangkan kenyamanan pengguna, misalnya: 2 kolom pada desktop dan 1 kolom pada mobile.

Kreativitas dalam Penggunaan Warna dan Tipografi:

- Pilih kombinasi warna dan tipografi yang sesuai dengan tema portfolio pribadi Anda.
- Tantangan: Buatlah skema warna yang ramah mata pengguna dan selaras dengan estetika keseluruhan. Jelaskan bagaimana pilihan warna dan tipografi Anda mendukung citra pribadi yang ingin Anda tampilkan melalui portfolio ini.

An Introduction to JavaScript

IF3110 – Web-based Application Development
School of Electrical Engineering and Informatics
Institut Teknologi Bandung

Introduction

- Interpreter based, used to be only running on the browser – run on server-side (node.js) or in a JVM (rhino)
- Javascript has similar syntax to Java, but it is **not** based on it
- Dynamic typing and supports duck typing
- Objects as general containers
- Function is the first-class (Lambda/closure)
- Linkage via global variables
- Prototypes over class-inheritance (class keyword is syntactic sugar to prototypes)
- Formalized & standardised as **ECMAScript (ECMA-262 and ISO/IEC 16262)**

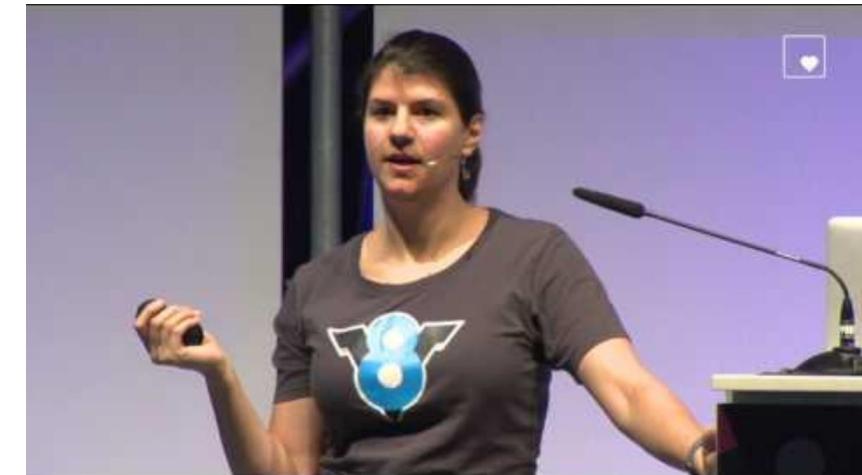
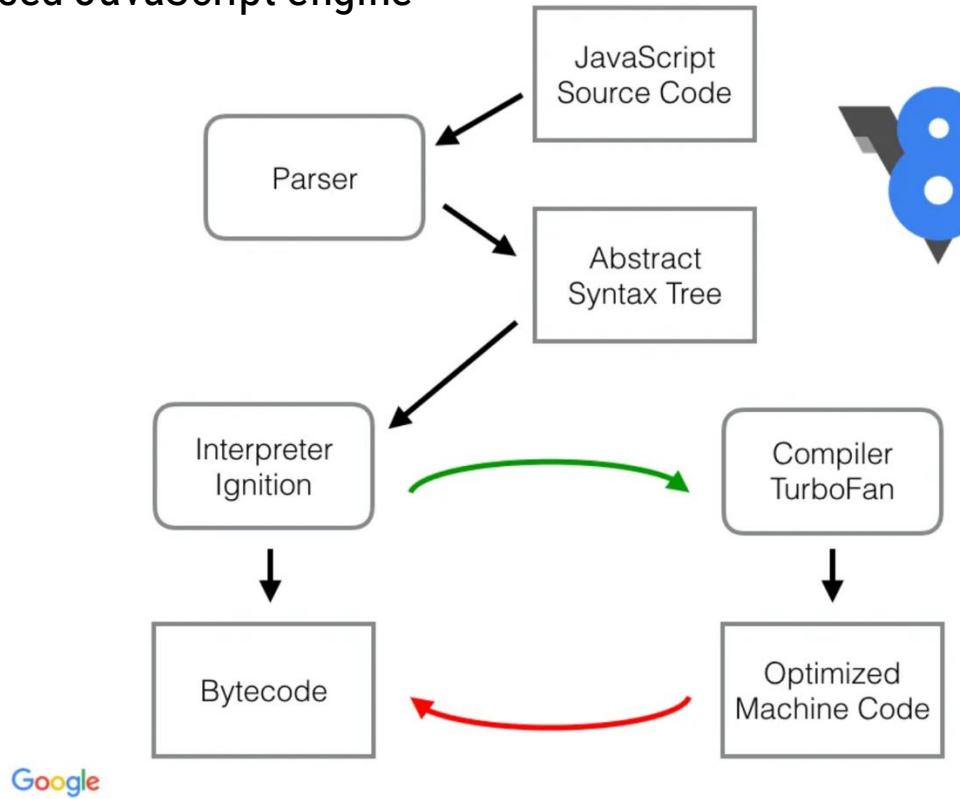
Introduction

- Javascript runs in browser:
 - Core syntax + DOM
- Has various implementation variant (the JavaScript Engine)
 - JavaScript Engine -> a special program embeded in a browser to execute JavaScript code. This program implements ECMAScript standard.
 - V8 -> Chrome, Edge, Opera, **Node.js (not a browser)**
 - SpiderMonkey -> Mozilla Firefox
 - JavaScriptCore -> WebKit/Safari
 - support provided for DOM
 - The DOM is not, however, typically provided by the JavaScript engine but instead by a browser

Inside the JavaScript Engine

V8 from Google is the most used JavaScript engine

written in C++



<https://www.youtube.com/watch?v=p-iiEDtpy6l>

V8 has had an interpreter as its first execution tier. This interpreter "compiles" JavaScript "just in time/JIT" to bytecode (which is then interpreted).

JavaScript in HTML

- Locating JavaScript
 - internal HTML document
 - Standar HTML 4.01
 - ```
<script type="text/javascript">
...statement...
</script>
```
    - Old Tag
    - ```
<script language="JavaScript">
...statement...
</script>
```
 - file external
 - Script JavaScript dituliskan pada file tersendiri (ekstensi file .js)
 - Pemanggilan file JavaScript:
 - ```
<script src="namafайл.js"></script>
```
- Tips: place js script at the end of a html, to improve the performance

# Simple Example

```
<HTML>
<HEAD>
<TITLE>Hello javascript</TITLE>
</HEAD>
<BODY onload="createDoc();">
<H1>Hello javascript...</H1>
<HR>
<SCRIPT type="text/javascript">
 function createDoc() {
 document.write("Hello...")
 }
</SCRIPT>
</BODY>
</HTML>
```

# Basic Syntax

- Alike to Java or C
- Case-sensitive
- One line represents a statement
- “;” is optional to end a statement
- Comment: // and /\* \*/

# Basic Syntax: Type

- Data Type
  - Number
  - String
  - Boolean
  - Object: e.g.:
    - Function
    - Array
    - Date
    - Regexp
  - Null
  - undefined

# Number

All numbers are represented in floating-point 64-bit IEEE-754 (double), but can be operated in integer

0, 3, 10000000, 0xF3, 031

3.14, 1.23e10, 3.14E-14

Be careful with **rounding errors**

Special values

NaN, Infinity

Number.MAX\_VALUE, Number.MIN\_VALUE

Number.POSITIVE\_INFINITY

Number.NEGATIVE\_INFINITY

# Number

NaN: Not a Number, result from a wrong operation

Every operation that has NaN, will result in NaN

NaN != NaN

```
>>> var a = 0/0;
```

```
>>> a
```

NaN

```
>>> typeof a
```

"number"

# Number function

`Number (value)`

Cast some value into number

Result in `NaN` if there is some problem,

Alternative: + prefix & `parseInt`

`+value`

`parseInt (value, radix)`

# Math

Math object provides a collection of standard function in arithmetic

**abs** absolute value

**floor** integer

**log** logarithm

**max** maximum

**pow** raise to a power

**random** random number

**round** nearest integer

**sin** sine

**sqrt** square root

# String

- Series of character written with some delimiter " or '  
'this is string', "this also a string", "ini bisa 'kan", "nama='amir'", 'This string\n consists of 2 lines'
- Immutable
- Some basic operations:

s = "hello"

s.length, s.charAt( 2 ), sub = s.substring( 2, 3 )

i = s.indexOf('e')

"hello, world".replace("hello", "goodbye")

"hello".toUpperCase()

# String Methods

`charAt`

`concat`

`indexOf`

`lastIndexOf`

`match`

`replace`

`search`

`slice`

`split`

`substring`

`toLowerCase`

`toUpperCase`

# boolean

- Value: true atau false
- Boolean(value): cast some value into boolean. Alternative: use !!
- Falsy: 0, **false**, **null**, **undefined**, "" (empty string), **NaN**
- Truthy: other than falsy (including “null”, “0”, “false”)

# null

A value that isn't anything

# **undefined**

A value that isn't even that

The default value for variables and parameters

The value of missing members in objects

# Loosely and Dynamically Typed

## Loosely typed

Any of these types can be stored in a variable, or passed as a parameter to any function.

The language is not "untyped".

Conversion of types happen automatically.

## Dynamically typed

Types are not required in advance.

Types are not checked at compile-time.

# Type Conversion

`10 + " objects" // => "10 objects". Number 10 converts to a string`

`"7" * "4" // => 28: both strings convert to numbers`

`var n = 1 - "x"; // => NaN: string "x" can't convert to a number`

`n + " objects" // => "NaN objects": NaN converts to string "NaN"`

# Variable

- Case sensitive
- Declare by var, const, or let (ES6, 2015)

```
var i; var x = 2;
```

```
let y = 3; // having a block scope, and cannot be redeclared
```

```
const z = 4; // having a block scope, and cannot be redeclared & reassigned
```

- Variable without a declaration, will be declared automatically as a global variable

```
a = 5;
```

- Scope: global dan local. local used when variable declared within a function with var
- Convention: variable name, parameter, member, function starts with lowercase. Constructor starts with uppercase

# Operator Aritmatika

Operator	Usage	Example	Result
+	addition	3+4	7
-	subtraction	4-3	1
*	multiplication	4*3	12
/	division	4/3	1.33333333
%	modulus	4 % 3	1
++	increment	x=5 x++	x=6
--	decrement	x=5 x--	x=4

# Assignment Operator

operator	example	explanation
=	a = b	assignment b to a
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

# Comparative Operator

operator	example	explanation
<code>==</code>	<code>a == b</code>	a equal to b
<code>!=</code>	<code>a != b</code>	a not equal to b
<code>&lt;</code>	<code>a &lt; b</code>	a lesser than b
<code>&gt;</code>	<code>a &gt; b</code>	a bigger than b
<code>&lt;=</code>	<code>a &lt;= b</code>	a lesser or equal to b
<code>&gt;=</code>	<code>a &gt;= b</code>	a bigger or equal to b
<code>====</code>	<code>'10' === 10</code>	comparaison w/o automatic type conversion

# Logic Operator

operator	usage	example
&&	and	$x = 6$ $y = 3$ $(x < 7 \&& y < 4)$
	or	$x = 6$ $y = 3$ $(x < 7    y < 2)$
!	not	$x = 6$ $y = 3$ $x != y$

==

!=

Equal and not equal operators

These operators can do type correction

It is better to use === (strict equality operator) and !==, which do not do type correction.

**Examples:**

```
"5" == 5 // true, because of type coercion
```

```
"5" === 5 // false, no type coercion occurs
```

# & &

- The guard operator, aka *logical and*
- If first operand is truthy
  - then result is second operand
  - else result is first operand
- It can be used to avoid null references

```
if (a) {
 return a.member;
} else {
 return a;
}
```

- can be written as

```
return a && a.member;
```



The default operator, aka *logical or*

If first operand is truthy

then result is first operand

else result is second operand

It can be used to fill in default values.

```
var last = input || nr_items;
```

(If `input` is truthy, then `last` is `input`, otherwise set `last` to `nr_items`.)

!

Prefix *logical not* operator.

If the operand is truthy, the result is **false**. Otherwise, the result is **true**.

**!!** produces booleans.

# Bitwise

&   |   ^   >>   >>>   <<

The bitwise operators convert the operand to a 32-bit signed integer, and turn the result back into 64-bit floating point.

# string operator

“ini ”+”buku”

“ini buku”

“jumlah adalah ”+ 5

“jumlah adalah 5”

“jumlah adalah ” + 5 + 3

“jumlah adalah 53”

5 + 3 + “ adalah bilangan”

“8 adalah bilangan”

# Break statement

Statements can have labels.

Break statements can refer to those labels.

```
loop: for (;;) {
 ...

 if (...) {

 break loop;

 }

 ...
}
```

# For statement

Iterate through all of the elements of an array:

```
for (var i = 0; i < array.length; i += 1) {

 // within the loop,

 // i is the index of the current member

 // array[i] is the current element

}
```

# For statement

Iterate through all of the members of an object:

```
for (var name in object) {

 if (object.hasOwnProperty(name)) {

 // within the loop,

 // name is the key of current member

 // object[name] is the current value

 }
}
```

# Switch statement

Multiway branch

The switch value does not need to be a number. It can be a string.

The case values can be expressions.

# Switch statement

```
switch (expression) {

 case ';':

 case ',':

 case '.':

 punctuation();

 break;

 default:

 noneOfTheAbove();

}
```

# Throw statement

```
throw new Error(reason);
```

```
throw {
 name: exceptionName,
 message: reason
};
```

# Try statement

```
try {
 ...
} catch (e) {
 switch (e.name) {
 case 'Error':
 ...
 break;
 default:
 throw e;
 }
}
```

# Try statement

The JavaScript implementation can produce these exception names:

'**Error**'

'**Evaluator**'

'**RangeError**'

'**SyntaxError**'

'**TypeError**'

'**URIError**'

# With statement

- Intended as a short-hand

```
with (o) {

 foo = null;

}
```

- Ambiguous

```
□ o.foo = null;

□ foo = null;
```

- Error-prone

- Don't use it

# Function

- mechanism to structure a program
  - modular
  - reusable
- kind
  - built-in
  - user defined
- function can be
  - have parameters
  - return a value

# Function

## syntax

```
function namaFungsi ([parameter]) {
 ... statements
 [return value]
}
```

function can be defined within another function

# Many ways to define a function

- typical definition

```
function f(x, y) { return x*y; }
```

- constructor Function()

```
var f = new Function("x", "y", "return x*y;");
```

- function literal

```
var f = function(x, y) { return x*y; }
```

- arrow function

```
var f = (x, y) => return x*y;
```

# function as data

function can be treated as data, stored in variable

```
function square(x) { return x*x; }
```

```
var b = square;
```

```
var c = b(5);
```

# function: parameter

function can have parameter/argument

```
function square(x) { return x*x; }
```

argument can be access from the function w/ name or object arguments

```
function square(x) { return arguments[0]*x; }
```

argument checking/verification in Javascript done at runtime

# Return statement

`return expression;`

or

`return;`

If there is no *expression*, then the return value is `undefined`.

Except for constructors, whose default return value is `this`.

# Object

Object – entity that composed of states represented as attributes (properties) value, and behavior represented in term of methods



# JavaScript Object

- In JavaScript, almost everything is object
  - Unless the 6 primitive types (string, number, boolean, null, undefined, symbol in ES6, bigint in ES11)
- Object in Javascript is a collection
  - similar to hashtable – can be accessed by a key (member name)
- Member is accessable by dot or subscript ( a.b or a['b'])
- `new Object()` produces an empty container of name/value pairs
- A name can be any string, a value can be any value except `undefined`
- A function is also an object

# Object

- Java Script is prototype-based object language

- Object created by constructor

```
var now = new Date()
```

- Object can be created from literal

```
var circle = { x:0, y:0,
radius:2 }
```

- class keyword is introduced in ES6 (2015)

- Attributed & methods access by . (dot)

```
var book = new Object();
```

```
book.title = "Javascript: The
Definitive Guide"
```

```
book.author = "David
Flanagan"
```

# Object

- property (attribute) can be enumerated w/ for loop

```
for(var name in obj) {

 document.write(name + "
");

}
```

- property of an object can be removed

```
delete book.title;
```

# property access

- using dot

```
object.property
```

- as associative array

```
object["property"]
```

# Constructor

- special function to create/initiate an object
- called with “new” command

```
var now = new Date()
```

# prototype

- prototype: mechanism to share properties and methods of objects Javascript
- every object has prototype
- properties & methods, to be share with other objects, place in prototype.

```
Circle.PI = 3.14;

Circle_area() { return Circle.PI * this.r * this.r; }

Circle.prototype.area = Circle_area;

c = new Circle();

document.write(c.area());
```

# prototype

- properties & method, in object instance, allocated ONLY for that particular instance
  - 'this' refers to the instance
- properties & method, in constructor, allocated ONLY for that constructor
  - can be used from other object
    - 'this' refers to constructor
- properties & method, in prototype, can be used by other object
  - 'this' refers to the instance

# Circle class prototype

```
function Circle(radius) {
 this.r = radius;
 this.min = function() {}
}

Circle.PI = 3.14159;

Circle.prototype.area = function() {
 return Circle.PI * this.r * this.r;
}

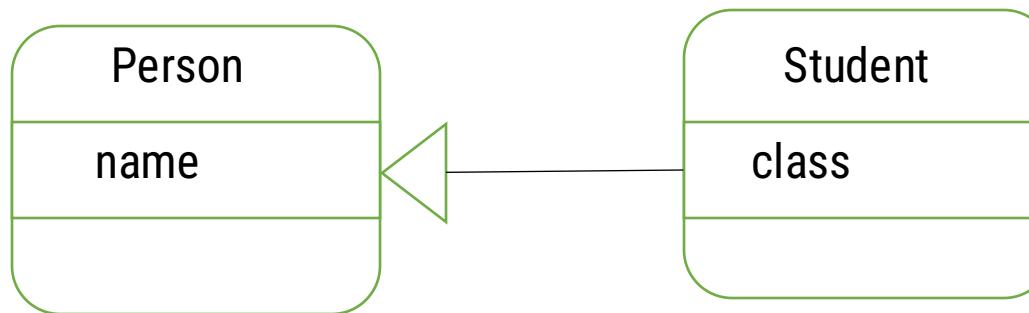
Circle.max = function(a,b) {
 if (a.r > b.r) return a;
 else return b;
}

var c = new Circle(1.0);
var d = Circle(2.0)
Circle.prototype.name = "a";
c.r = 2.2;
var a = c.area();
var x = Math.exp(Circle.PI);
var d = new Circle(1.2);
var bigger = Circle.max(c,d);
```

# Inheritance

Inheritance: OO concept for reusability

using an existing class to define a new class



# Inheritance through prototype

JavaScript provides inheritance through prototype:

```
function Person(n) { this.name = n; }
function Student(n, c) {
 this.name = n;
 this.class = c;
}
Student.prototype = new Person();
```

# Class and Inheritance

Since ES6 (2015) :

```
class Person {
 constructor(n) {
 this.name = n;
 }
}
class Student extends Person {
 constructor(n, c) {
 super(n);
 this.class = c;
 }
}
```

# Object

## constructor

`o.constructor` consists of function constructor used to initialize `o`

## `toString()`

represents string of an object. Called automatic when an object is casted to string

## `valueOf()`

represents object other than string. Called automatic when an object is casted to other than string

## `isPrototypeOf( x )`

verify whether an object is prototype from another object

# Array

- by Array constructor

```
var a = new Array();
var a = new Array(1, 2, "tiga");
```

- by literal

```
var a = [1, 2, "tiga"];
```

- array access

```
a[0] = 1;
a[7] = 8;
```

- length of array

```
a.length
```

# Array methods

join()

convert array values into string

reverse()

reverses the value of array

sort()

sort the array values alphabetically

concat()

combining array values with the value of parameter

slice()

slice the value of array array

splice()

remove and add the value of array

push(), pop(), shift(), unshift()

# Array Example

```
<html>
<script type="text/javascript">
//cara 1 pendefinisian array
mhs = new Array();
mhs[0] = "Bevin";
mhs[1] = "Andini";
mhs[2] = "Citra";
//cara 2 pendefinisian array
mhs = new Array("Bevin", "Andini", "Citra");
//cara 3 pendefinisian array
mhs = ["Bevin", "Andini", "Citra"];
//cara pengaksesan array
document.write("Mahasiswa pertama adalah "+mhs[0]+".
"); //Bevin
document.write("Mahasiswa terakhir adalah "+mhs[2]+".
"); //Citra
//cara pengaksesan array menggunakan loop
for (i=0; i<mhs.length; i++) {
 document.write(mhs[i] +"
");
}
//mengurutkan array
mhs.sort();
//menggabungkan array
document.write(mhs.join("-")); //Andini-Bevin-Citra
</script>
</html>
```

# Regular expression

- used to process text, search and convert text with particular pattern
- Literal:

```
var pattern = /s$/
```

- RegExp object

```
var pattern = new RegExp ("s$");
```

# Browser object

window

navigator

screen

history

location

Not standard, but provided in every browser

# Browser object

Window object: global execution context,

```
var x = 5;
```

sama dengan

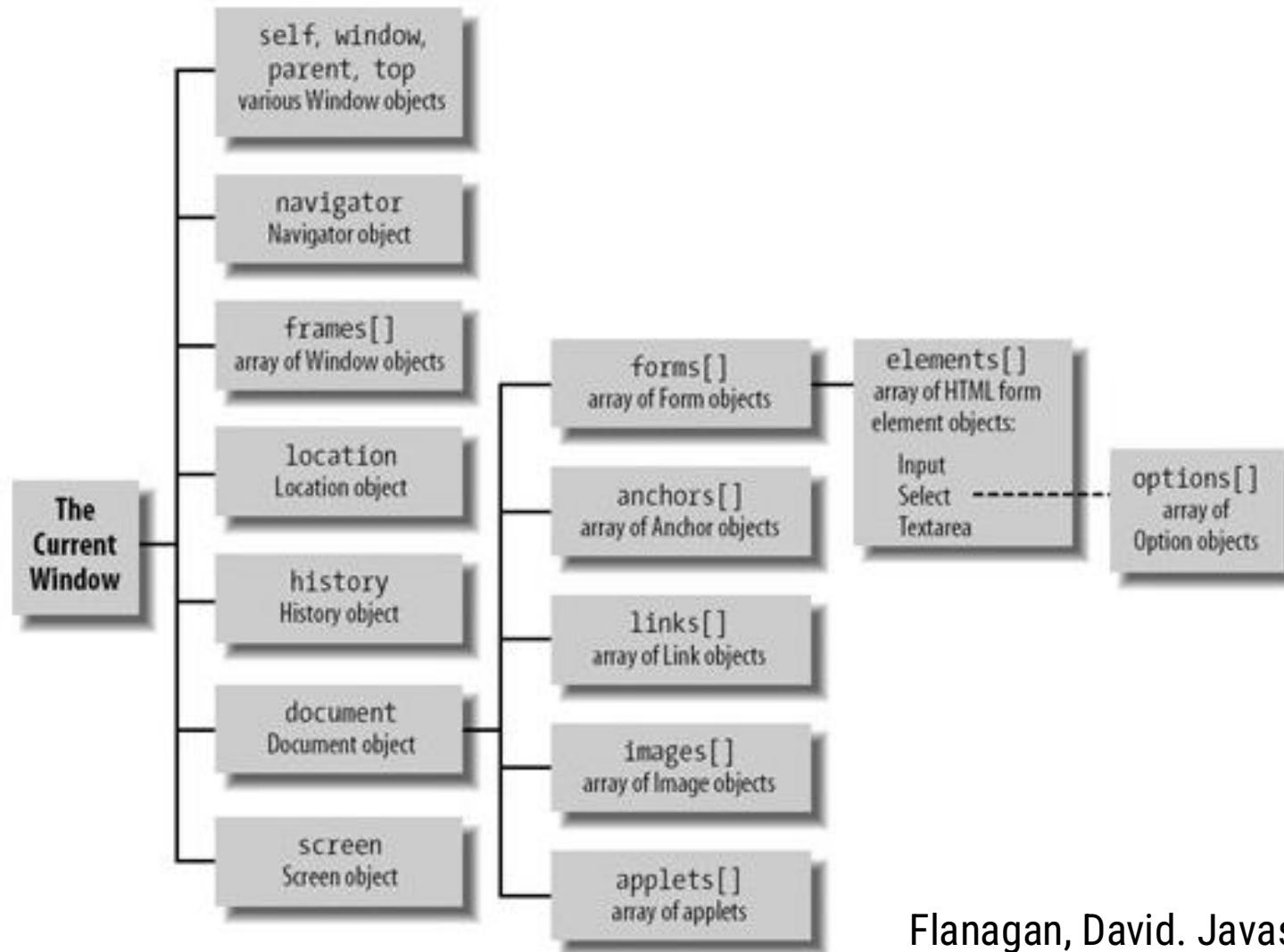
```
window.x = 5;
```

It refers to window browser (or frame) that renders the page

Document object: page that is being rendered

```
window.document
```

# DOM



Flanagan, David. Javascript: The Definitive Guide, 5th Ed.

# Document object

`getElementById( id )`

`getElementsByName( tag )`

`createElement( tag )`

`HTMLDocument`, derived from `Document`

body, forms, anchors, images, links

title, lastModified, referrers

etc

`close()`, `open()`, `write()`

# Javascript as event handler

HTML Element HTML can define to run script if an event occurs

onclick, onmousedown, onmouseup, onchange, onload

In-line HTML:

```
<input type="checkbox" name="options" value="giftwrap"
onclick="giftwrap = this.checked;" >
```

# Javascript as event handler

Script:

```
<script>
 // Define a function to display function displayTime() {
 // A script of js code the current time
 var elt = document.getElementById("clock");
 // Find element with id="clock"
 }
 window.onload = displayTime;
 // Start displaying the time when document loads.
</script>
```

# Javascript as URL and Bookmarklet

Javascript can be executed as a URL link

```
javascript:alert("Hello World!")
javascript:var now = new Date(); "<h1>The time is:</h1>" + now;
```

URL containing javascript can be saved as a bookmark

```
<a href='javascript:var e='', r='';
do{e=prompt ("Expression: "+e+"\n"+r+"\n",e);
try{r="Result: "+eval(e); }catch(ex) {r=ex; } }
while(e);void 0;'> JS Evaluator
```

JavaScript URL often used by hackers to run malicious codes, e.g., Cross Site Scripting (XSS)

# Javascript Execution in HTML

Javascript is executed as soon as the element, containing script, has been processed by the browser.

To postpone the execution: use `defer` attributed,

OR

write a script as a function that is called by `onload` attributes from the body element

# Javascript threading model

Javascript runs sequentially; with a single thread model.

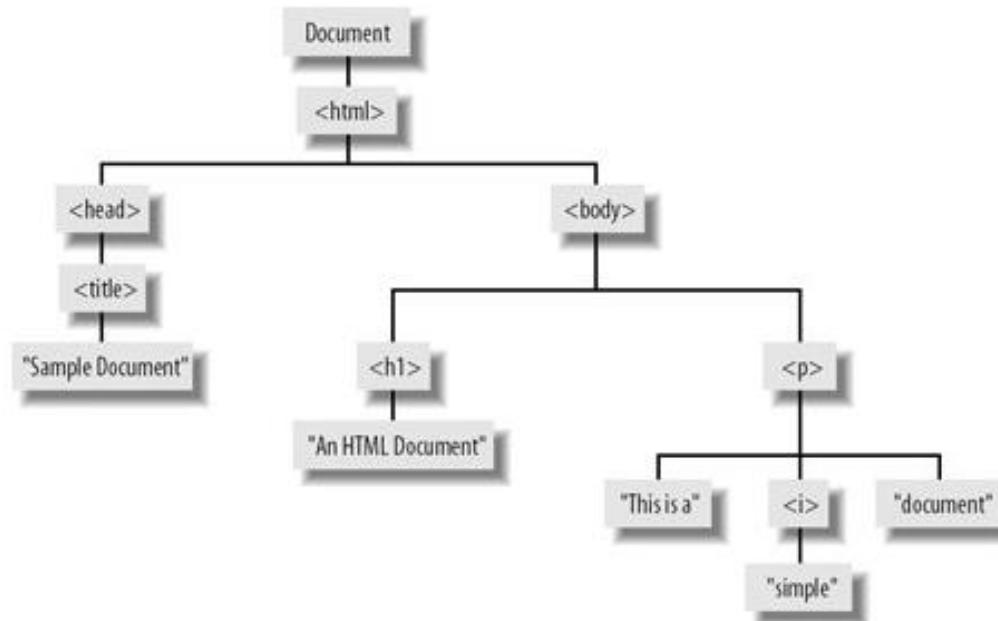
On loading HTML document HTML, executing a script will suspend the loading process until the execution finishes

event-handler javascript SHOULD NOT be long, it creates a browser freeze.

# W3C DOM

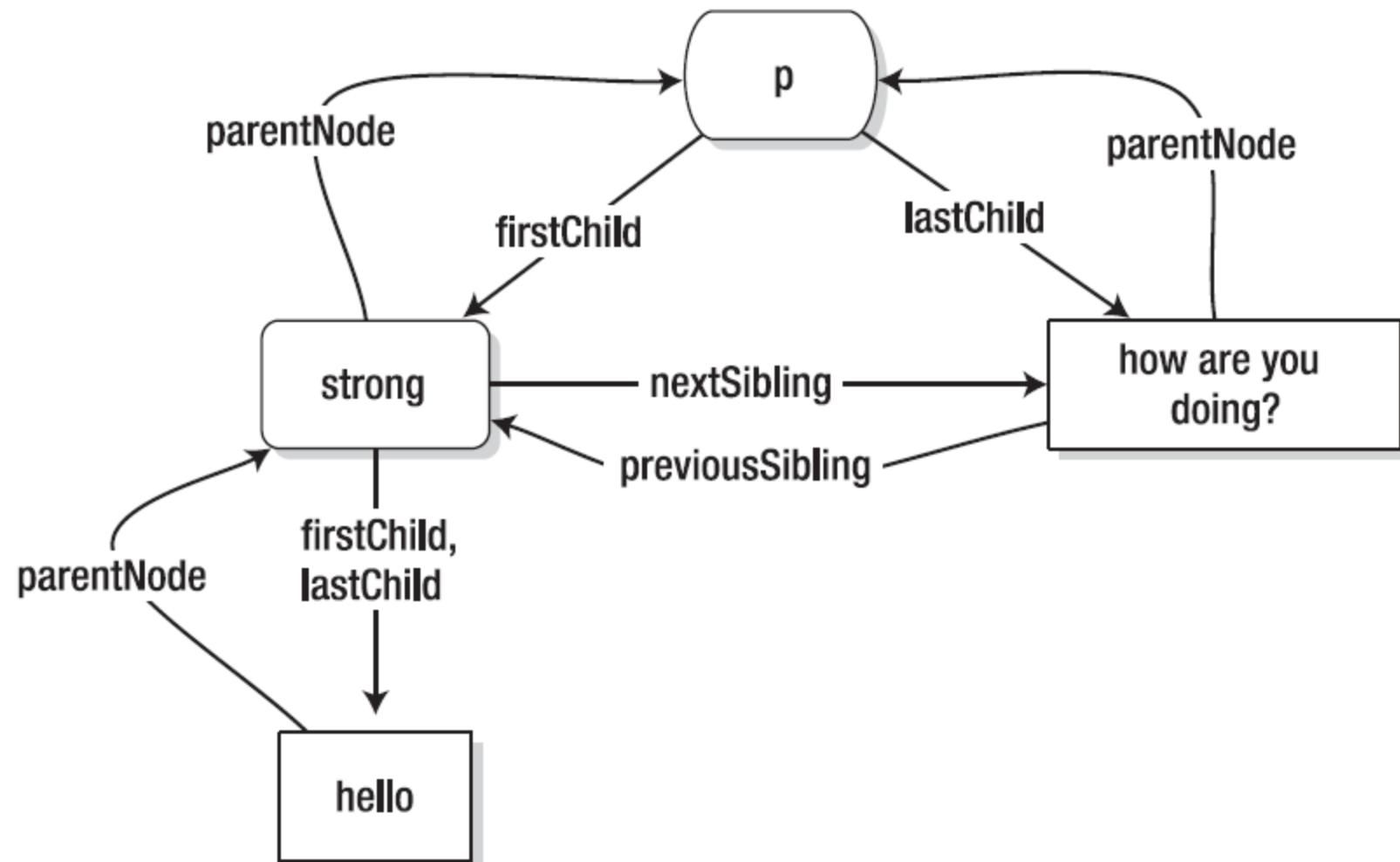
HTML Document is accessed by javascript via DOM

```
<html> <head> <title>Sample Document</title> </head> <body> <h1>An
HTML Document</h1> <p>This is a <i>simple</i> document.</p> </html>
```



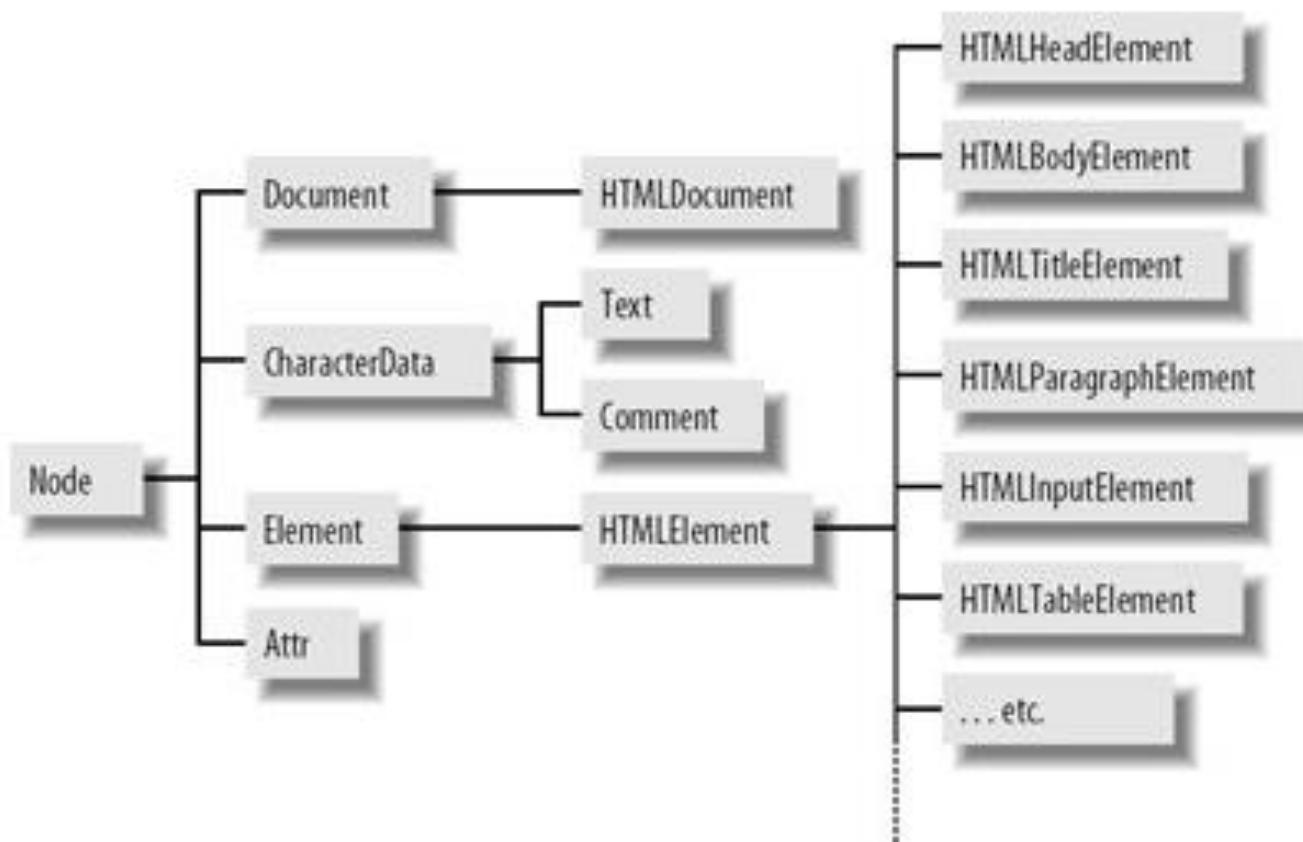
# W3C DOM

```
<p>Hello how are you doing?</p>
```



# W3C DOM

HTML element in DOM is descendant of Node



# W3C DOM

DOM Level: standardisation DOM by W3C in 2 version (level)

DOM Level 1, 1998

Definition of Node, Element, Attr, Document etc

DOM Level 2, 2000

Modular. DOM Level 1 becomes Core module, there are optional modules, such as Event, CSS

DOM Level 3

# W3C DOM

Accessing element:

```
document.getElementById(id_name);
```

return object Node object, if existed

```
Document.getElementByTagName(tag_name);
```

return NodeList, can be accessed as array

```
document.getElementsByTagName("h1") [0]
var li = document.getElementsByTagName("li");
for (var j = 0; j < li.length; j++) {
 li[j].style.border = "1px solid #000";
}
```

# Encode/Decode URIs

- In a URL (Uniform Resource Locator) or a URI (Uniform Resource Identifier), some characters have special meanings.
- If you want to "escape" those characters, you can use the functions `encodeURI()` or `encodeURIComponent()`.
  - The first one will return a usable URL.
  - The second one assumes you're only passing a part of the URL, like a query string for example, and will encode all applicable characters.
- The opposites of `encodeURI()` and `encodeURIComponent()` are `decodeURI()` and `decodeURIComponent()` respectively.
- Sometimes, in older code, you might see the similar functions `escape()` and `unescape()` but these functions have been deprecated and should not be used.

```
>>> var url = 'http://www.packtpub.com/script.php?q=this and that';
>>> encodeURIComponent(url)
'http://www.packtpub.com/scr%20ipt.php?q>this%20and%20that'
>>> encodeURIComponent(url);
'http%3A%2F%2Fwww.packtpub.com%2Fscr%20ipt.php%3Fq%3Dthis%20and%20that'
```

# Timers

Timer berguna untuk membuat animasi atau efek tampilan

```
<script>

var WastedTime = {
 start: new Date(),
 displayElapsedTime: function() {
 var now = new Date();
 var elapsed = Math.round((now - WastedTime.start)/60000);
 window.defaultStatus = "You have wasted " + elapsed +
 " minutes.";
 }
}
setInterval(WastedTime.displayElapsedTime, 60000);
</script>
```

# Forms

- Form can be accessed via javascript using `document.forms[]`
- Form object has `submit()` and `reset()` method
- Javascript can be called from a form via event:
  - `onsubmit`, `onreset`
  - `onchange`, `onblur`, `onfocus`

# Form Element

Object	HTML tag	type property	Description and events
Button	<code>&lt;input type="button"&gt;</code> or <code>&lt;button type="button"&gt;</code>	"button"	A push button; <code>onclick</code> .
Checkbox	<code>&lt;input type="checkbox"&gt;</code>	"checkbox"	A toggle button without radio-button behavior; <code>onclick</code> .
File	<code>&lt;input type="file"&gt;</code>	"file"	An input field for entering the name of a file to upload to the web server; <code>onchange</code> .
Hidden	<code>&lt;input type="hidden"&gt;</code>	"hidden"	Data submitted with the form but not visible to the user; no event handlers.
Option	<code>&lt;option&gt;</code>	none	A single item within a Select object; event handlers are on the Select object, not on individual Option objects.
Password	<code>&lt;input type="password"&gt;</code>	"password"	An input field for password entrytyped characters are not visible; <code>onchange</code> .
Radio	<code>&lt;input type="radio"&gt;</code>	"radio"	A toggle button with radio-button behavioronly one selected at a time; <code>onclick</code> .
Reset	<code>&lt;input type="reset"&gt;</code> or <code>&lt;button type="reset"&gt;</code>	"reset"	A push button that resets a form; <code>onclick</code> .
Select	<code>&lt;select&gt;</code>	"select-one"	A list or drop-down menu from which one item may be selected; <code>onchange</code> . (See also Option object.)
Select	<code>&lt;select multiple&gt;</code>	"select-multiple"	A list from which multiple items may be selected; <code>onchange</code> . (See also Option object.)
Submit	<code>&lt;input type="submit"&gt;</code> or <code>&lt;button type="submit"&gt;</code>	"submit"	A push button that submits a form; <code>onclick</code> .
Text	<code>&lt;input type="text"&gt;</code>	"text"	A single-line text entry field; <code>onchange</code> .
Textarea	<code>&lt;textarea&gt;</code>	"textarea"	A multiline text entry field; <code>onchange</code> .

# TypeScript

- What is TypeScript:
  - Programming Language: A language that includes all the existing JavaScript syntax, plus new TypeScript-specific syntax for defining and using types
  - Type Checker: A program that takes in a set of files written in JavaScript and/or TypeScript, checks for incorrect syntaxes
  - Compiler (Transpiler) : A program that runs the type checker, reports any issues, then outputs the equivalent JavaScript code.
  - Language Service: A program that uses the type checker to tell editors such as VS Code how to provide helpful utilities to developers

# TypeScript

TS

JS

1. TypeScript source -> TypeScript AST
2. AST is checked by typechecker
3. TypeScript AST -> JavaScript source

4. JavaScript source -> JavaScript AST
5. AST -> bytecode
6. Bytecode is evaluated by runtime

# TypeScript

- JS

- Unconstrained
- Checked at runtime
- Errors surfaced at runtime (mostly)

```
function squareOf(n) {
 return n * n
}
squareOf(2) // evaluates to 4
squareOf('z') // evaluates to NaN
```

- TS

- Constrained or bound statically
- Checked at compile time
- Errors surfaced at compile time (mostly)

```
function squareOf(n: number) {
 return n * n
}
squareOf(2) // evaluates to 4
squareOf('z') // Error TS2345: Argument of type '"z"' is not
 // parameter of type 'number'.
```

# Advance JavaScript

# Closure

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives a function access to its outer scope. In JavaScript, closures are created every time a function is created, at function creation time.

```
function makeFunc() {
 const name = "Mozilla";
 function displayName() {
 console.log(name);
 }
 return displayName;
}

const myFunc = makeFunc();
myFunc();
```

```
function makeAdder(x) {
 return function (y) {
 return x + y;
 };
}

const add5 = makeAdder(5);
const add10 = makeAdder(10);

console.log(add5(2)); // 7
console.log(add10(2)); // 12
```

# Memory Management

Regardless of the programming language, the memory life cycle is pretty much always the same:

1. Allocate the memory you need
2. Use the allocated memory (read, write)
3. Release the allocated memory when it is not needed anymore

JavaScript automatically allocates memory when objects are created and frees it when they are not used anymore (garbage collection)

# Memory Management: Allocation

## Value initialization

```
const n = 123; // allocates memory for a number
const s = "azerty"; // allocates memory for a string

const o = {
 a: 1,
 b: null,
}; // allocates memory for an object and contained values

// (like object) allocates memory for the array and
// contained values
const a = [1, null, "abra"];

function f(a) {
 return a + 2;
} // allocates a function (which is a callable object)

// function expressions also allocate an object
someElement.addEventListener(
 "click",
 () => {
 someElement.style.backgroundColor = "blue";
 },
 false,
);
```

## Function Call

```
const d = new Date(); // allocates a Date object

const e = document.createElement("div"); // allocates a DOM element
```

# Memory Management: Using Values

Using values basically means reading and writing in allocated memory. This can be done by reading or writing the value of a variable or an object property or even passing an argument to a function.

# Memory Management: Garbage collection

The majority of memory management issues occur at this phase. The most difficult aspect of this stage is determining when the allocated memory is no longer needed.

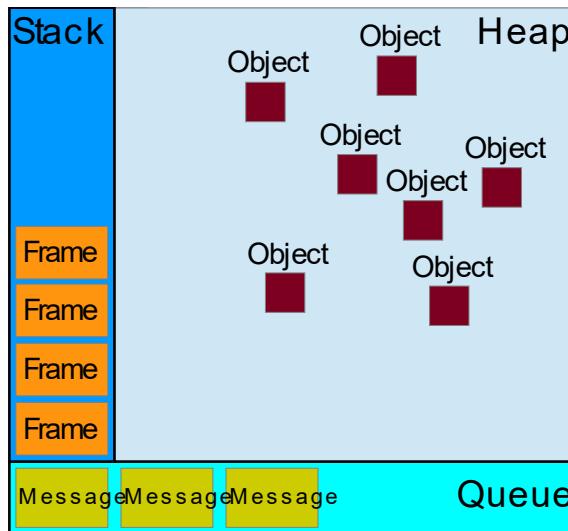
The main concept that garbage collection algorithms rely on is the concept of reference. Within the context of memory management, an object is said to reference another object if the former has access to the latter (either implicitly or explicitly).

**Reference-counting** Algorithm -> determining whether or not an object is still needed to determine if an object still has any other objects referencing it (not used anymore in modern JavaScript engine)

**Mark-and-sweep** Algorithm -> reduces the definition of "an object is no longer needed" to "an object is unreachable" (all modern engines ship a mark-and-sweep garbage collector)

# Concurrency model and Event Loop

JavaScript has a runtime model based on an **event loop**, which is responsible for executing the code, collecting and processing events, and executing queued sub-tasks. This model is quite different from models in other languages like C and Java.



JavaScript runtime concepts (only a theoretical model)

Modern JavaScript engines implement and heavily optimize this.

JavaScript program is *single-threaded*. A thread is a sequence of instructions that a program follows. Because the program consists of a single thread, it can only do one thing at a time: so if it is waiting for our long-running synchronous call to return, it can't do anything else.

# Runtime Concept: Stack

Function calls form a stack of frames.

Order of operations:

1. When calling **bar**, a first frame is created containing references to bar's arguments and local variables.
2. When bar calls **foo**, a second frame is created and pushed on top of the first one, containing references to foo's arguments and local variables.
3. When **foo** returns, the top frame element is popped out of the stack (leaving only bar's call frame).
4. When **bar** returns, the stack is empty.

```
function foo(b) {
 const a = 10;
 return a + b + 11;
}

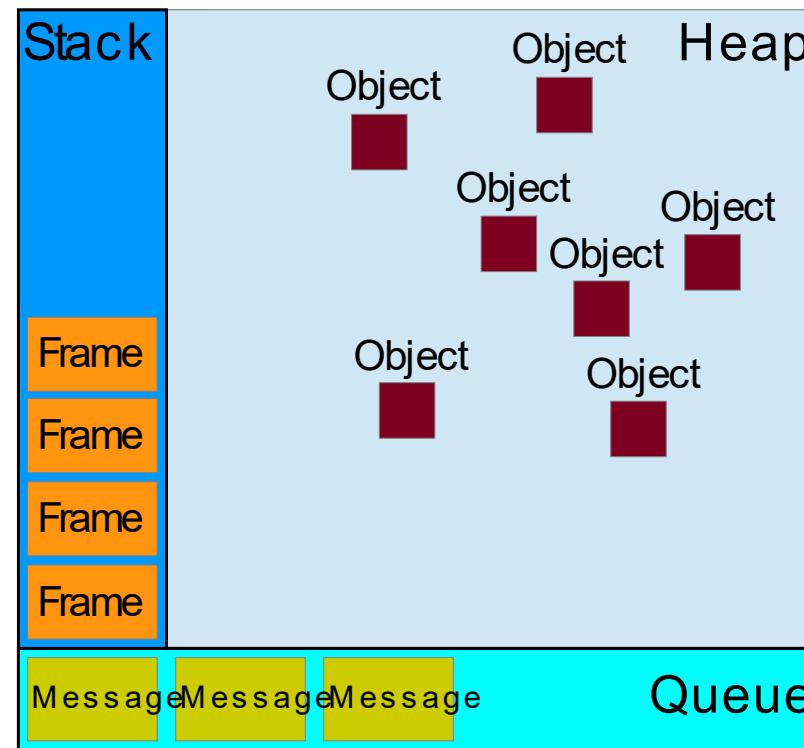
function bar(x) {
 const y = 3;
 return foo(x * y);
}

const baz = bar(7); // assigns 42 to baz
```

Note that the arguments and local variables may continue to exist, as they are stored outside the stack

# Runtime Concept: Heap

Objects are allocated in a heap which is just a name to denote a large (mostly unstructured) region of memory.



# Runtime Concept: Queue

A JavaScript runtime uses a message queue, which is a list of messages to be processed. Each message has an associated function that gets called to handle the message.

At some point during the event loop, the runtime starts handling the messages on the queue, starting with the oldest one. To do so, the message is removed from the queue and its corresponding function is called with the message as an input parameter. As always, calling a function creates a new stack frame for that function's use.

The processing of functions continues until the stack is once again empty. Then, the event loop will process the next message in the queue (if there is one).

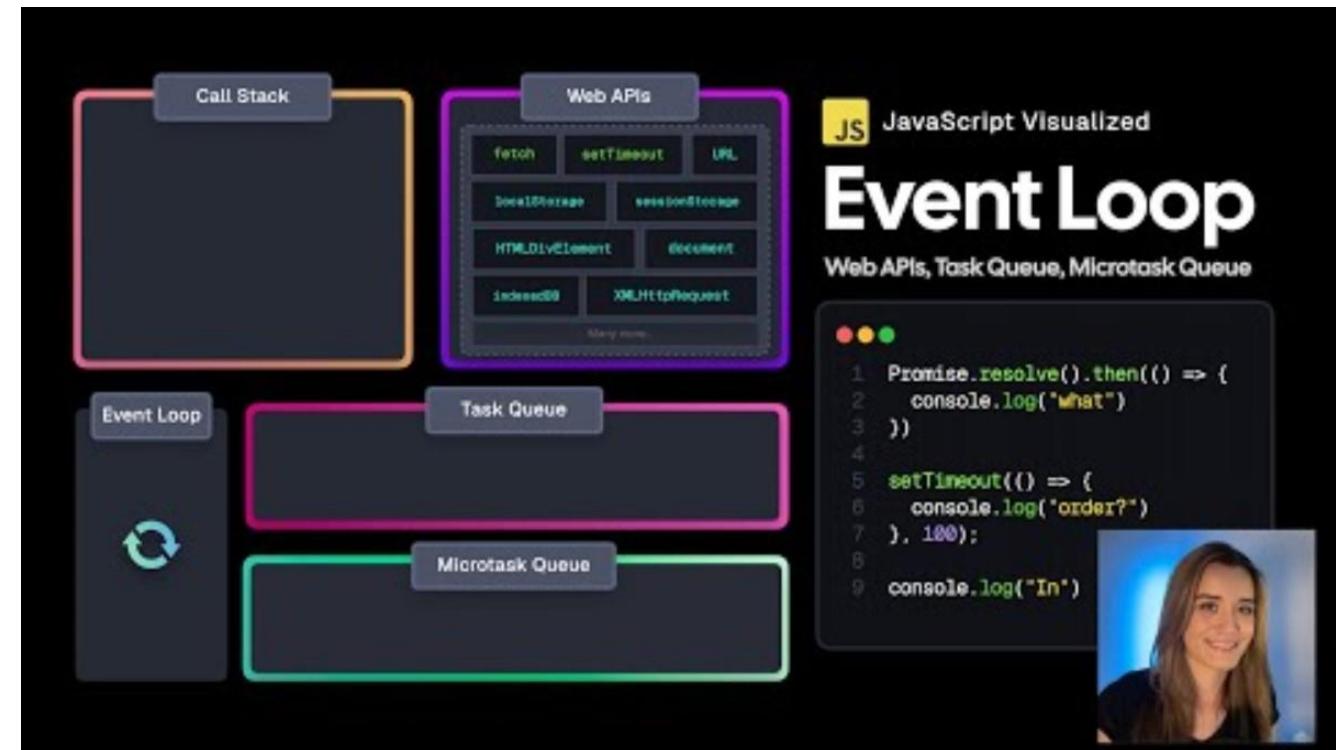
# The Event Loop

The event loop got its name because of how it's usually implemented, which usually resembles:

```
while (queue.waitForMessage()) {
 queue.processNextMessage();
}
```

A very interesting property of the event loop model is that JavaScript, unlike a lot of other languages, never blocks.

Legacy exceptions exist like alert or synchronous XHR, but it is considered good practice to avoid them



Good Visualization of the Event Loop:  
<https://www.youtube.com/watch?v=eiC58R16hb8>

# Asynchronous JavaScript

When multiple related things happen without any being dependent on the completion of previous happenings, they are asynchronous.

```
const MAX_PRIME = 1000000;

function isPrime(n) {
 for (let i = 2; i <= Math.sqrt(n); i++) {
 if (n % i === 0) {
 return false;
 }
 }
 return n > 1;
}

const random = (max) => Math.floor(Math.random() * max);

function generatePrimes(quota) {
 const primes = [];
 while (primes.length < quota) {
 const candidate = random(MAX_PRIME);
 if (isPrime(candidate)) {
 primes.push(candidate);
 }
 }
 return primes;
}
```

We have synchronous program that a very inefficient algorithm to generate multiple large prime numbers.

What if the synchronous function takes a long time?

A downside of the JavaScript runtime model is that if a function takes too long to complete, the web application is unable to process user interactions like click or scroll. The browser mitigates this with the "a script is taking too long to run" dialog.

# Asynchronous JavaScript: Event Handler

Event handlers are really a form of asynchronous programming: you provide a function (the event handler) that will be called, not right away, but whenever the event happens. If "the event" is "the asynchronous operation has completed", then that event could be used to notify the caller about the result of an asynchronous function call.

Example using  
asynchronous API:  
XMLHttpRequest

```
const log = document.querySelector(".event-log");

document.querySelector("#xhr").addEventListener("click", () => {
 log.textContent = "";

 const xhr = new XMLHttpRequest();

 xhr.addEventListener("loadend", () => {
 log.textContent = `${log.textContent}Finished with status: ${xhr.status}`;
 });

 xhr.open(
 "GET",
 "https://raw.githubusercontent.com/mdn/content/main/files/en-us/_wikihistory.json",
);
 xhr.send();
 log.textContent = `${log.textContent}Started XHR request\n`;
});
```

# About Event Listeners in JavaScript

- Event Handler concept

Event Handler is function that handles when particular Event happened at specified target. Object representing the event is passed as the first argument to the event handler. This event object either implements or is derived from the [Event](#) interface.

Used not only on UI, but also in workers, websocket, etc.

- Two approaches:
  - onevent: simplicity and one to one.
  - addEventListener and removeEventListener:

Very Flexible and Specific, Allows multiple event listener attached to a target. Also enable Event Bubbling/Capture phase to be specified (for DOM event)

```
const btn = document.querySelector('button');

function greet(event) {
 console.log('greet:', event)
}

btn.onclick = greet;
```

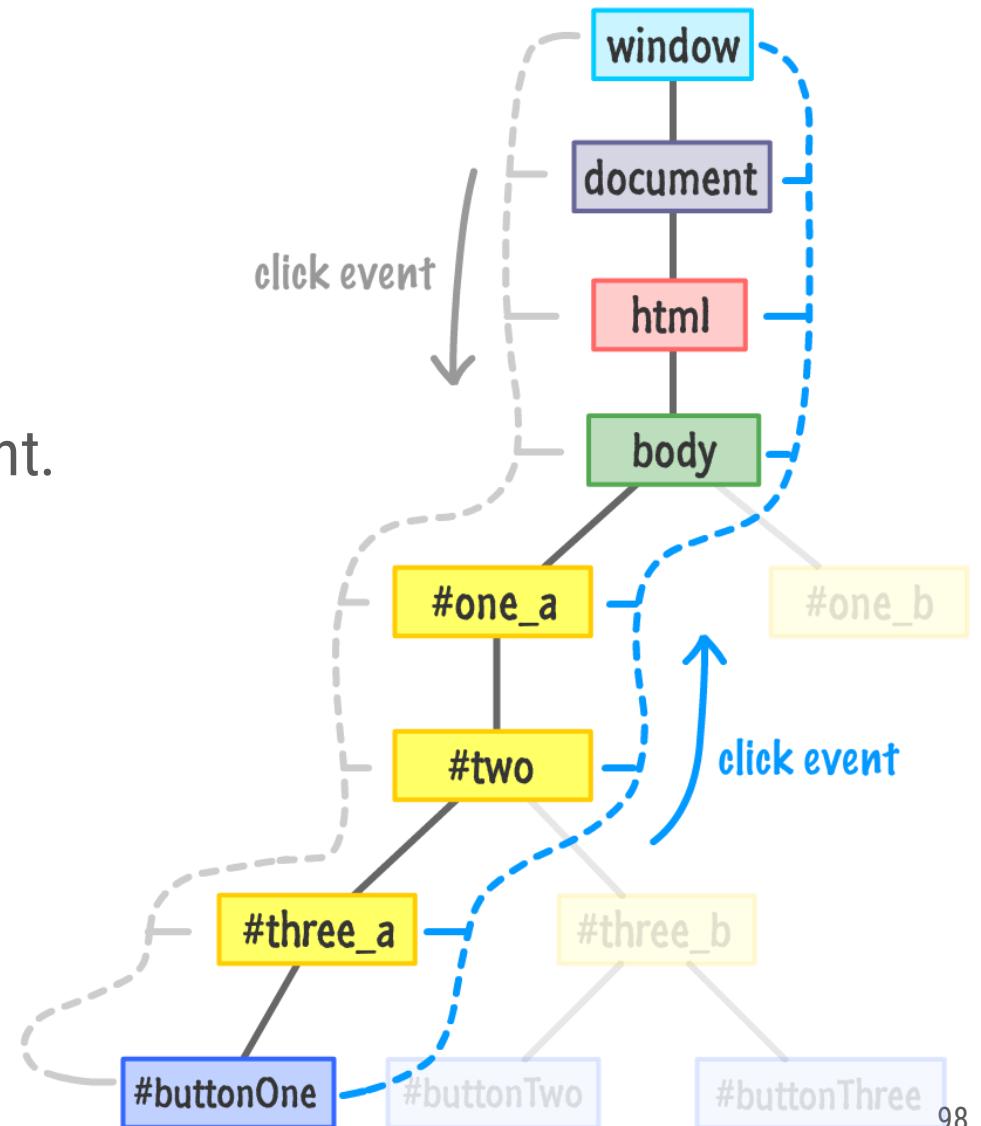
```
const btn = document.querySelector('button');

function greet(event) {
 console.log('greet:', event)
}

btn.addEventListener('click', greet);
```

# Event Bubble/Capture

- The standard DOM Events describes 3 phases of event propagation:
  - Capturing phase – the event goes down to the element.
  - Target phase – the event reached the target element.
  - Bubbling phase – the event bubbles up from the element.
- `Event.target` is the deepest nested element being the target.
- "this" refer to the current element.



# Asynchronous JavaScript: Callback

A callback is just a function that's passed into another function, with the expectation that the callback will be called at the appropriate time.

```
function doStep1(init, callback) {
 const result = init + 1;
 callback(result);
}

function doStep2(init, callback) {
 const result = init + 2;
 callback(result);
}

function doStep3(init, callback) {
 const result = init + 3;
 callback(result);
}

function doOperation() {
 doStep1(0, (result1) => {
 doStep2(result1, (result2) => {
 doStep3(result2, (result3) => {
 console.log(`result: ${result3}`);
 });
 });
 });
}

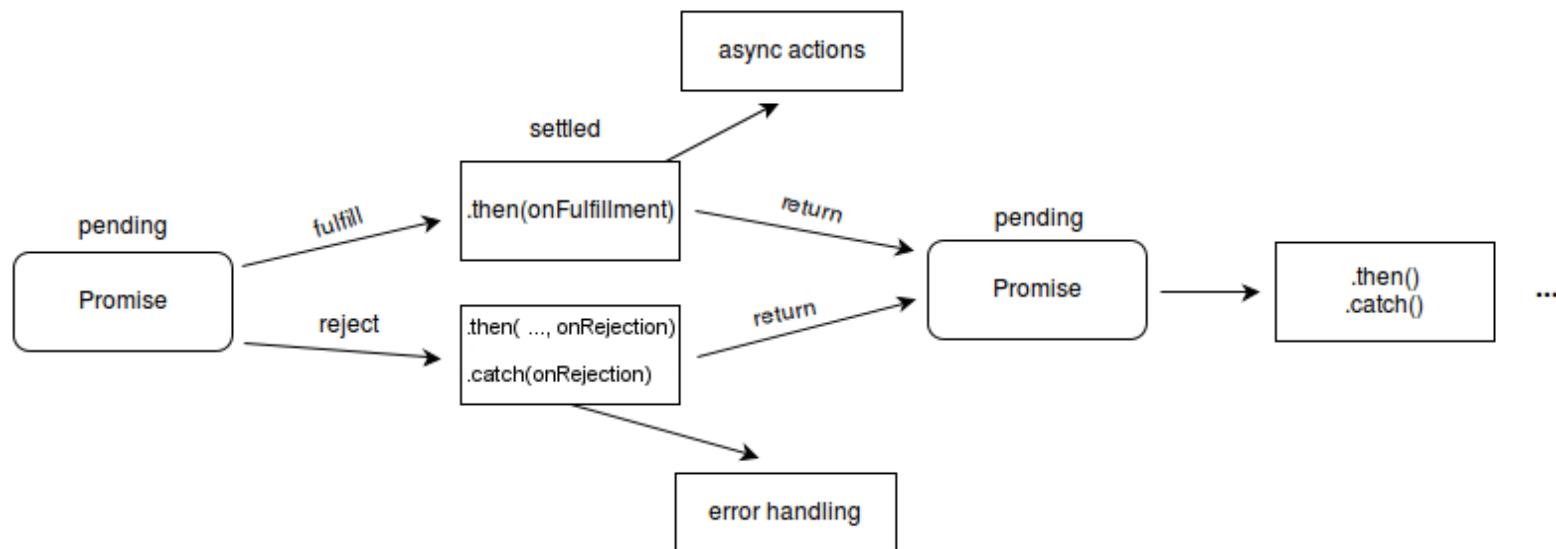
doOperation();
```

However, callback-based code can get hard to understand when the callback itself has to call functions that accept a callback. This is a common situation if you need to perform some operation that breaks down into a series of asynchronous functions.

This is sometimes called "callback hell" or the "pyramid of doom".

# Asynchronous JavaScript: Promise

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason



**pending:** initial state, neither fulfilled nor rejected.

**fulfilled:** meaning that the operation was completed successfully.

**rejected:** meaning that the operation failed.

# Asynchronous JavaScript: Promise creation and chaining

Example below create and return new Promise Object. Then we handle or reject using then interface.

```
const myPromise = new Promise((resolve, reject) => {
 setTimeout(() => {
 resolve("foo");
 }, 300);
});

myPromise
 .then(handleFulfilledA, handleRejectedA)
 .then(handleFulfilledB, handleRejectedB)
 .then(handleFulfilledC, handleRejectedC);
```

The promise chain is what you need when your operation consists of several asynchronous functions, and you need each one to complete before starting the next one.

This special then() function returns another promise p. Depends on the return of value of the handler which may:

- returns a value: p gets **fulfilled** with the returned value as its value.
- doesn't return anything: p gets **fulfilled** with undefined as its value.
- throws an error: p gets **rejected** with the thrown error as its value.
- returns another pending promise: p is **pending** and becomes fulfilled/rejected with that promise's value as its value immediately after that promise becomes fulfilled/rejected.

# Asynchronous JavaScript: Fetch example

```
// Make a request for user.json
fetch('/article/promise-chaining/user.json')
 // Load it as json
 .then(response => response.json())
 // Make a request to GitHub
 .then(user => fetch(`https://api.github.com/users/${user.name}`))
 // Load the response as json
 .then(response => response.json())
 // Show the avatar image (githubUser.avatar_url) for 3 seconds (maybe
 .then(githubUser => {
 let img = document.createElement('img');
 img.src = githubUser.avatar_url;
 img.className = "promise-avatar-example";
 document.body.append(img);

 setTimeout(() => img.remove(), 3000); // (*)
 });
}
```

Fetch is the modern replacement for XMLHttpRequest: unlike XMLHttpRequest (XHR), which uses callbacks, Fetch is promise-based.

The `fetch()` function returns a Promise which is fulfilled with a Response object representing the server's response. You can then check the request status and extract the body of the response in various formats, including text and JSON, by calling the appropriate method on the response.

# Asynchronous JavaScript: Async Await

The `async` keyword gives you a simpler way to work with asynchronous promise-based code. Adding `async` at the start of a function makes it an `async` function:

```
async function fetchProducts() {
 try {
 // after this line, our function will wait for the `fetch()` call to be settled
 // the `fetch()` call will either return a Response or throw an error
 const response = await fetch(
 "https://mdn.github.io/learning-area/javascript/apis/fetching-data/can-
 store/products.json",
);
 if (!response.ok) {
 throw new Error(`HTTP error: ${response.status}`);
 }
 // after this line, our function will wait for the `response.json()` call to be
 // settled
 // the `response.json()` call will either return the parsed JSON object or throw
 // an error
 const data = await response.json();
 console.log(data[0].name);
 } catch (error) {
 console.error(`Could not get products: ${error}`);
 }
}

fetchProducts();
```

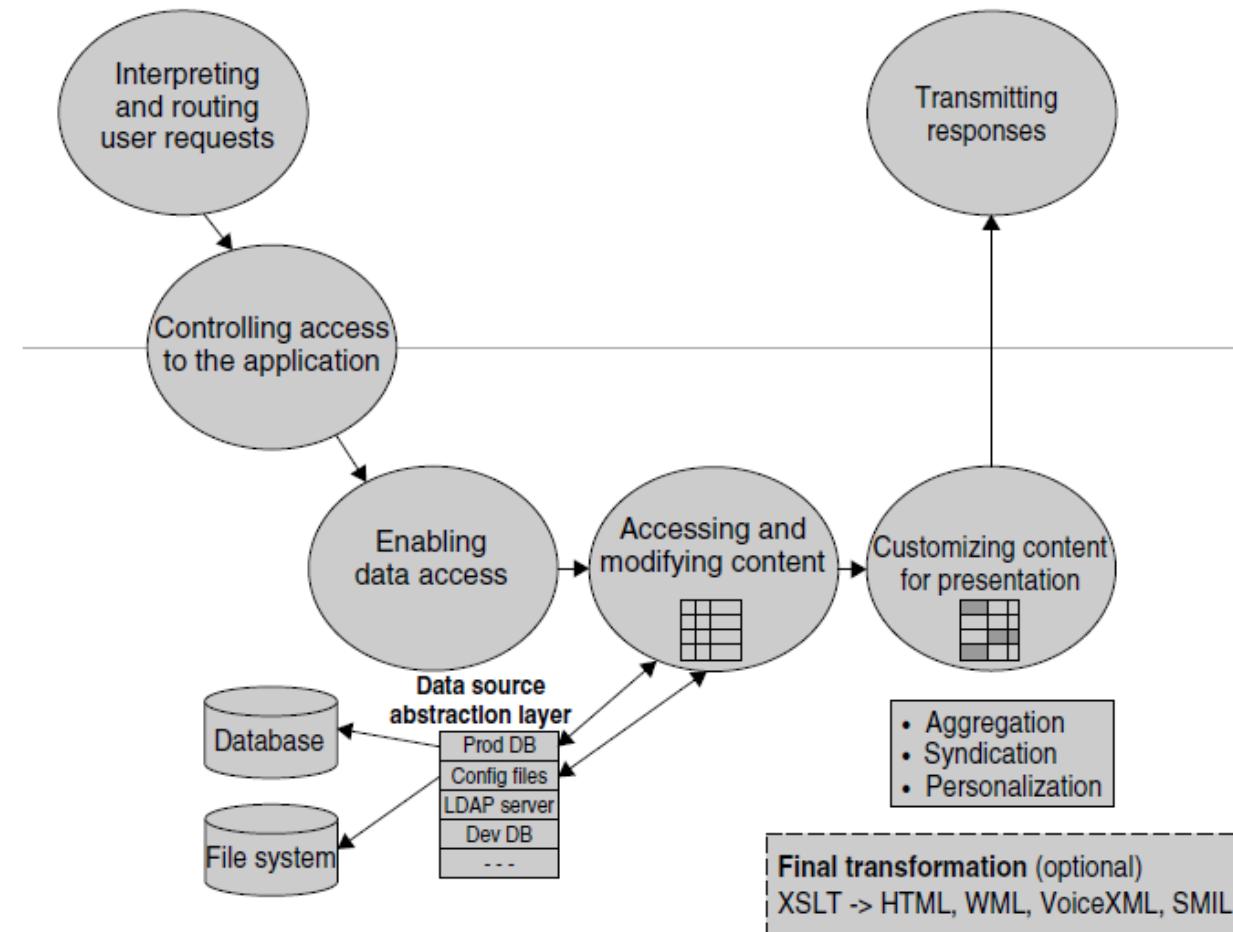
Inside an `async` function, you can use the `await` keyword before a call to a function that returns a promise.

The `async` and `await` keywords make it easier to build an operation from a series of consecutive asynchronous function calls, avoiding the need to create explicit promise chains, and allowing you to write code that looks just like synchronous code (a cleaner style).

# Server Side Scripting

IF3110 – Web-based Application Development  
School of Electrical Engineering and Informatics  
Institut Teknologi Bandung

# Typical Web Application Processing



**Figure 8.3** Processing flow in a typical Web application (Above the grey line—Web server; below the grey line—Web application)

# Web Stack

A collection of software required to develop and run web applications

Contains all software layers

Operating System

Web Server

Programming (& Markup) Language (frontend & backend)

Web Framework (frontend & backend)

Database Server

Examples:

LAMP, MEAN, ...

# Common Stack

- MEAN/MERN/MEVN
  - MongoDB, ExpressJS, Angular/React/Vue.js, NodeJS
- WINS
  - Windows Server, IIS, .NET, (Microsoft) SQL Server
- LAMP
  - Linux, Apache, MySQL/MariaDB, PHP
- XAMPP
  - (X), Apache, MySQL/MariaDB, PHP & Perl
- etc.

# Server Side Scripting Languages

- PHP (Laravel, Zend, CodeIgniter, etc.)
- Python (Flask, Django)
- Ruby (Rails)
- Java (Spring, etc.)
- C# (ASP.NET)
- Scala (Play)
- Go
- etc.

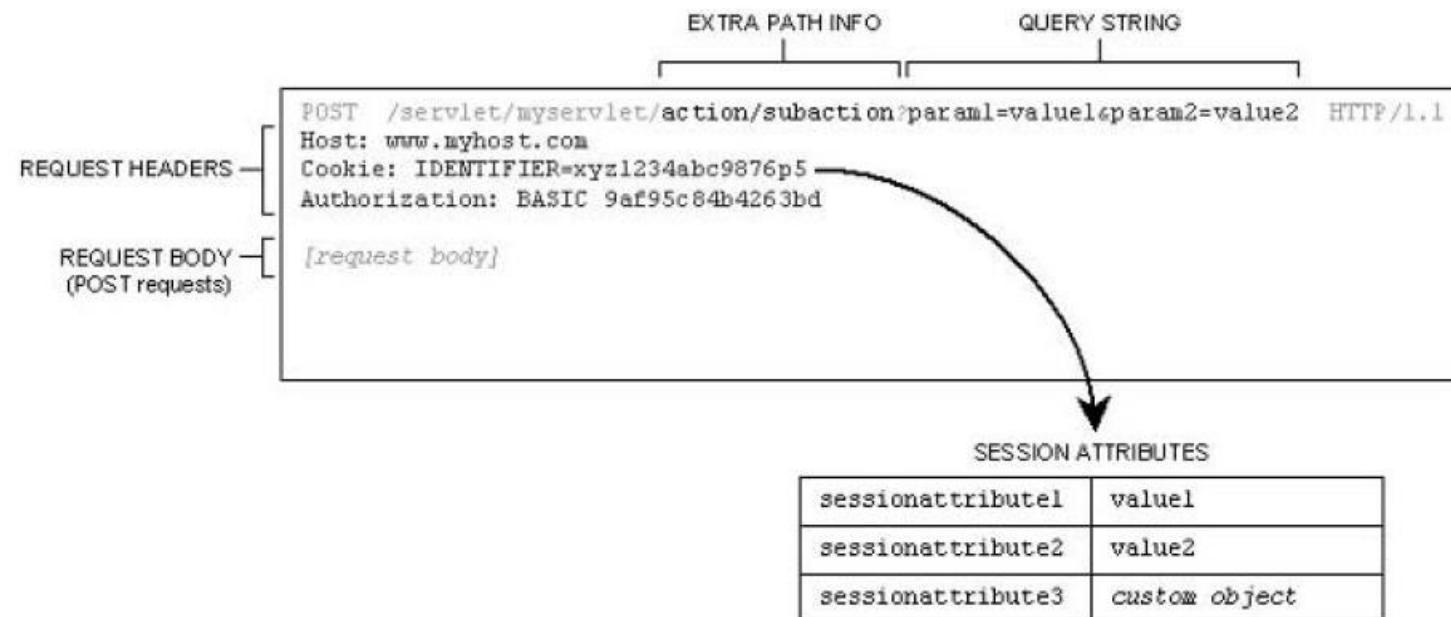
# Server Side Aspects

- Routing
- Templating
- Form Processing
- Data Access

# Interpreting & processing request

Browser communicate the request via HTTP (info sent by the browser called *request context*)

- URL, query string, request headers, request body, session information



**Table 8.1** Selection of server processing modules

Approach	Configuration	URL Examples
<b>CGI</b>	<ol style="list-style-type: none"><li>1. Server provides mechanism for defining CGI path mappings.</li><li>2. Server provides mechanism to map URL file name extensions to CGI processing.</li></ol>	<a href="http://host/cgi-bin/script?...">http://host/cgi-bin/script?...</a> <a href="http://host/.../script.cgi?...">http://host/.../script.cgi?...</a>
<b>Auxiliary processing modules</b> (scripting, template, hybrid)	<ol style="list-style-type: none"><li>1. Server provides mechanism for registering processing modules for files with predefined name extensions.</li><li>2. Native support may be built into server (e.g., ASP on IIS).</li></ol>	<a href="http://host/.../modulename.php?...">http://host/.../modulename.php?...</a> <a href="http://host/.../modulename.cfm?...">http://host/.../modulename.cfm?...</a> <a href="http://host/.../modulename.asp?...">http://host/.../modulename.asp?...</a>
<b>J2EE Webapp</b>	<ol style="list-style-type: none"><li>1. Server provides mechanism for defining application path mappings.</li><li>2. Server provides mechanism to map URL file name extensions to be processed as JSP pages.</li></ol>	<a href="http://host/servlet/servletname/...">http://host/servlet/servletname/...</a> <a href="http://host/webappname/servletname/...">http://host/webappname/servletname/...</a> <a href="http://host/webappname/modulename.jsp?...">http://host/webappname/modulename.jsp?...</a> <a href="http://host/anotherjsp.jsp">http://host/anotherjsp.jsp</a>

# Routing

- Map the specific URL with an associated function
- URI syntax
  - scheme:[//authority]path[?query][#fragment]
  - authority = [userinfo@]host[:port]
  - <https://website.org/one/certain/path.html>
  - <https://myweb.app/view>
- Without routing through web application, that simply returns an HTML file
- Using a web application, an HTML file *may* be returned, among other return types.

# Routing Example

- Python + Flask

```
from flask import Flask
from flask import jsonify, render_template, request

app = Flask(__name__)

@app.route("/")
def hello_world():
 return "<p>Hello, World!</p>"

@app.route("/view")
def view_route():
 return render_template("view.html")

@app.route("/analyze", methods=[POST])
def analyze_sentiment():
 text = request.form['text']
 #analyze sentiment using NLP library and save to a variable "sentiment"
 return jsonify({'text': text, 'sentiment': sentiment})
```

# Template

- Templates are files that contain static data as well as placeholders for dynamic data.
- A template is rendered with specific data to produce a final document.

```
from flask import Flask
from flask import jsonify, render_template

app = Flask(__name__)

@app.route("/analyze", methods=[POST])
def analyze_sentiment():
 text = request.form['text']
 #analyze sentiment using NLP library and save to a variable "sentiment"
 return render_template("result.html", sentiment=sentiment, text=text)
```

# Template (2)

```
<!--pre-rendered result.html-->

<html>
<head><!--necessary header components--></head>
<body>
 <h2>Text</h2>
 <p>{{ text }}</p>
 <h2>Sentiment</h2>
 <p>{{ sentiment }}</p>
</body>
</html>
```

# Form Processing

- It is a usual practice for POST-type form requests to be processed using server-side programs
- The data values coming out of the form are passed to the function to be further processed

```
@app.route('/login', methods=['GET', 'POST'])
def login():
 error = None
 if request.method == 'POST':
 if request.form['username'] != 'admin' or request.form['password'] != 'admin':
 error = 'Invalid Credentials. Please try again.'
 else:
 return redirect(url_for('home'))
 return render_template('login.html', error=error)
```

# Form Processing (2)

```
<form action="" method="post">
 <input type="text" placeholder="Username" name="username"
 value="{{ request.form.username }}">
 <input type="password" placeholder="Password" name="password"
 value="{{ request.form.password }}">
 <input class="btn btn-default" type="submit" value="Login">
</form>
```

# PHP Programming

# What is PHP?

- PHP (recursive acronym for *PHP: Hypertext Preprocessor*) is a widely-used **open source general-purpose scripting language** that is especially suited for web development and **can be embedded into HTML**.
- What distinguishes PHP from something like client-side JavaScript is that the code is **executed on the server**, generating HTML which is then sent to the client.

# Relation between the versions

- PHP/FI 2.0 is an early and no longer supported version of PHP. PHP 3 is the successor to PHP/FI 2.0 and is a lot nicer.
- PHP 5 uses the Zend engine 2 which, among other things, offers many additional OOP features.
- PHP 6 was experimental, and never released
- PHP 7 is the current generation of PHP, twice faster than PHP 5

# What can PHP do?

- Anything.
- PHP is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. But PHP can do much more.

*Table 1-1. Sampling of major websites that use PHP*

Website name	Description	URL
Facebook	Social networking	<a href="http://www.facebook.com">http://www.facebook.com</a>
Flickr	Photograph sharing	<a href="http://www.flickr.com">http://www.flickr.com</a>
Wikipedia	Online collaborative encyclopedia	<a href="http://www.wikipedia.org">http://www.wikipedia.org</a>
SugarCRM	Customer relationship management tool	<a href="http://www.sugarcrm.com">http://www.sugarcrm.com</a>
Dotproject	Project management tool	<a href="http://www.dotproject.org">http://www.dotproject.org</a>
Drupal	Website construction template engine	<a href="http://drupal.org">http://drupal.org</a>
Interspire	Newsletter and email marketing product	<a href="http://www.interspire.com">http://www.interspire.com</a>

# Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
 <title>Example</title>
 </head>
 <body>

 <?php
 echo "Hi, I'm a PHP script!";
 ?>

 </body>
</html>
```

# hello.php

```
<html>
 <head>
 <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

```
<html>
 <head>
 <title>PHP Test</title>
 </head>
 <body>
 <p>Hello World</p>
 </body>
</html>
```

# Other examples

## Example #1 Printing a variable (Array element)

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];
?>
```

A sample output of this script may be:

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

## Example #2 Example using control structures and functions

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
 echo 'You are using Internet Explorer.
';
}
?>
```

A sample output of this script may be:

You are using Internet Explorer.<br />

### Example #3 Mixing both HTML and PHP modes

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
?>
<h3>strpos() must have returned non-false</h3>
<p>You are using Internet Explorer</p>
<?php
} else {
?>
<h3>strpos() must have returned false</h3>
<p>You are not using Internet Explorer</p>
<?php
}
?>
```

A sample output of this script may be:

```
<h3>strpos() must have returned non-false</h3>
<p>You are using Internet Explorer</p>
```

### Example #1 A simple HTML form

```
<form action="action.php" method="post">
<p>Your name: <input type="text" name="name" /></p>
<p>Your age: <input type="text" name="age" /></p>
<p><input type="submit" /></p>
</form>
```

### Example #2 Printing data from our form

```
Hi <?php echo htmlspecialchars($_POST['name']); ?>.
You are <?php echo (int)$_POST['age']; ?> years old.
```

A sample output of this script may be:

```
Hi Joe. You are 22 years old.
```

# Language Overview

- **script tag**

```
<?php php_statements.. ?>
<? php_statements.. ?>
```

- **comment**

```
// komentar
/* komentar */
komentar
```

- **statement**

- ended by “;”

```
$nama = "amir";
```

# Data Types

- Scalar Type:
  - boolean, integer, float (or double), string
- Compound Type:
  - array, object
- Special Type:
  - resource, NULL
- pseudo-types:
  - mixed, number, callback

# Examples

```
<?php
$a_bool = TRUE; // a boolean
$a_str = "foo"; // a string
$a_str2 = 'foo'; // a string
$an_int = 12; // an integer

echo gettype($a_bool); // prints out: boolean
echo gettype($a_str); // prints out: string

// If this is an integer, increment it by four
if (is_int($an_int)) {
 $an_int += 4;
}

// If $bool is a string, print it out
// (does not print out anything)
if (is_string($a_bool)) {
 echo "String: $a_bool";
}
?>
```

# Array

- An array in PHP is actually an ordered map.
- A map is a type that associates **values** to **keys**.
- This type is optimized for several different uses; it can be treated as an:
  - array,
  - list (vector),
  - hash table (an implementation of a map),
  - dictionary, collection, stack, queue, and probably more.
- As array values can be other arrays, trees and multidimensional arrays are also possible.

# Example

```
<?php
$arr = array("foo" => "bar", 12 => true);

echo $arr["foo"]; // bar
echo $arr[12]; // 1
?>
```

```
<?php
$arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));

echo $arr["somearray"][6]; // 5
echo $arr["somearray"][13]; // 9
echo $arr["somearray"]["a"]; // 42
?>
```

```
<?php
// This array is the same as ...
array(5 => 43, 32, 56, "b" => 12);

// ...this array
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

# Array operations

## Array access

```
$buah[0] = "apel";
$buah[1] = "mangga";
```

## Adding array element

```
$buah[] = "apel";
$buah[] = "mangga";
```

## length of array

```
$len = count($buah);
```

## associative array

```
$pengguna["nama"] = "amir";
$pengguna["alamat"] = "ganesha 10";
```

# Array Init

- Initialization

```
$hari = array("senin", "selasa", "rabu", "kamis",
"jumat", "sabtu", "minggu");
```

```
$hari = array(1=>"senin", "selasa", "rabu", "kamis",
"jumat", "sabtu", "minggu");
```

```
$days = array("mon"=>"monday", "tue"=>"tuesday",
"wed"=>"wednesday", "thu"=>"thursday",
"fri"=>"friday",
"sat"=>"saturday", "sun"=>"sunday");
```

# Array access (1)

```
<?php
$cities = array(
 "Jawa Barat"=>array(
 "Bandung",
 "Cianjur",
 "Cirebon"
),
 "Jawa Tengah"=>array(
 "Semarang",
 "Magelang"
)
);
print($cities["Jawa Barat"] [1]);
?>
```

# Array access (2)

```
<?php

$buah [0] = "apel";
$buah [1] = "mangga";
$buah [2] = "jambu";
$len = count($buah);
for($i = 0; $i < $len; $i++) {
 echo $buah[$i], "
";
}
?>
```

# Array access (3)

```
<?php

$days = array ("mon"=>"monday", "tue"=>"tuesday", "wed"=>"wednesday",
"thu"=>"thursday", "fri"=>"friday", "sat"=>"saturday",
"sun"=>"sunday");

foreach($days as $key=>$value) {
 echo "key: ", $key, ", value: ", $value, "
";
}
?>
```

# Function

**definition:**

```
function f($param1, $param2 ..) { statements.. }
```

**return value**

```
return $val;
```

**parameter by reference**

```
function f(&$param1, &$param2 ..) { statements.. }
```

**return by reference**

```
function &f($param1, $param2 ..) { .. return $v }
```

**dynamic parameter access**

```
func_get_arg($i), func_num_args()
```

# Function

- variable scope
  - use `global` to access global variable with in a function

```
<?php
 function assignName() {
 // // global $name;
 echo $nglobal $name = "Zeev";
 ame;
 }
 global $name;
 $name = "Leon";
 assignName();
 print($name);
?
?>
```

# Function

- static variable
  - store variable state with in a function
- dynamic function call
  - function execution can be done dynamically, by store the function name in a variable and call the variable as a function

```
<?php
 function printBold($text)
 {
 print("$text") ;
 }

 print("This Line is not Bold
\n");
 printBold("This Line is Bold");
 print("
\n");
 print("This Line is not Bold
\n");

?>
```

```
<?php
 function makeBold($text)
 {
 $text = "$text";
 return($text);
 }

 print("This Line is not Bold
\n");
 print(makeBold("This Line is Bold") . "
\n");
 print("This Line is not Bold
\n");
?>
```

```
<?php
 function stripCommas(&$text)
 {
 $text = str_replace(", ", "", $text);
 }

$myNumber = "10,000";

stripCommas($myNumber);
print($myNumber);

?>
```

```
<?
function useColor()
{
 static $ColorValue = "#00FF00";

 if($ColorValue == "#00FF00") {
 $ColorValue = "#CCFFCC";
 } else {
 $ColorValue = "#00FF00";
 }

 return($ColorValue);
}

print("<table width=\"300\">\n");
for($count=0; $count < 10; $count++) {
 $RowColor = useColor();

 print("<tr>" .
 "<td style=\"background: $RowColor\">" .
 "Row number $count" .
 "</td>" .
 "</tr>\n");
}
print("</table>\n");
?>
```

```
<?php
 function write($text)
 {
 print($text);
 }

 function writeBold($text)
 {
 print("$text");
 }

 $myFunction = "write";
 $myFunction("Hello!");
 print("
\n");

 $myFunction = "writeBold";
 $myFunction("Goodbye!");
 print("
\n");
?>
```

# Object

- Starting with PHP 5, the object model was rewritten to allow for better performance and more features. This was a major change from PHP 4. PHP 5 has a full object model.
- Among the features in PHP 5 are the inclusions of visibility, abstract and final classes and methods, additional magic methods, interfaces, cloning and typehinting.
- PHP treats objects in the same way as references or handles, meaning that each variable contains an object reference rather than a copy of the entire object. See Objects and References

# Example

```
<?php
class foo
{
 function do_foo()
 {
 echo "Doing foo.";
 }
}

$bar = new foo;
$bar->do_foo();
?>
```

# Resource

- A resource is a special variable, holding a reference to an external resource.
- Resources are created and used by special functions.
- As resource variables hold special handlers to opened files, database connections, image canvas areas and the like.

## Example #1 mysql\_connect() example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
 die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

# NULL

- The special NULL value represents a variable with no value. NULL is the only possible value of type NULL.
- A variable is considered to be null if:
  - assigned the constant NULL
  - has not been set to any value yet
  - has been unset()

# Pseudo-types

- mixed indicates that a parameter may accept multiple (but not necessarily all) types
- number indicates that a parameter can be either integer or float
- Some functions like `call_user_func()` or `usort()` accept user-defined callback functions as a parameter

# Type Juggling

- PHP does not require (or support) explicit type definition in variable declaration; a variable's type is determined by the context in which the variable is used. That is to say, if a string value is assigned to variable \$var, \$var becomes a string. If an integer value is then assigned to \$var, it becomes an integer.

```
<?php
$foo = "0"; // $foo is string (ASCII 48)
$foo += 2; // $foo is now an integer (2)
$foo = $foo + 1.3; // $foo is now a float (3.3)
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)
$foo = 5 + "10 Small Pigs"; // $foo is integer (15)
```

```
<?php
$foo = 10; // $foo is an integer
$bar = (boolean) $foo; // $bar is a boolean
?>
```

# Variable

- Scope default: local
- To access global variable use **global** keyword
- Support **static** variable
- Can be variable of variable:

```
$a="hallo";
$$a="world"; // sama dg $hallo
```

# Predefined Variable

- `$_REQUEST`: variabel http request
- `$_GET`: variabel http GET
- `$_POST`: variabel http POST
- `$_FILES`: http file upload
- `$_SESSION`: variabel sesi
- `$_COOKIE`: http cookie
- `$_ENV`: variabel environment
- `$_SERVER`: variabel server

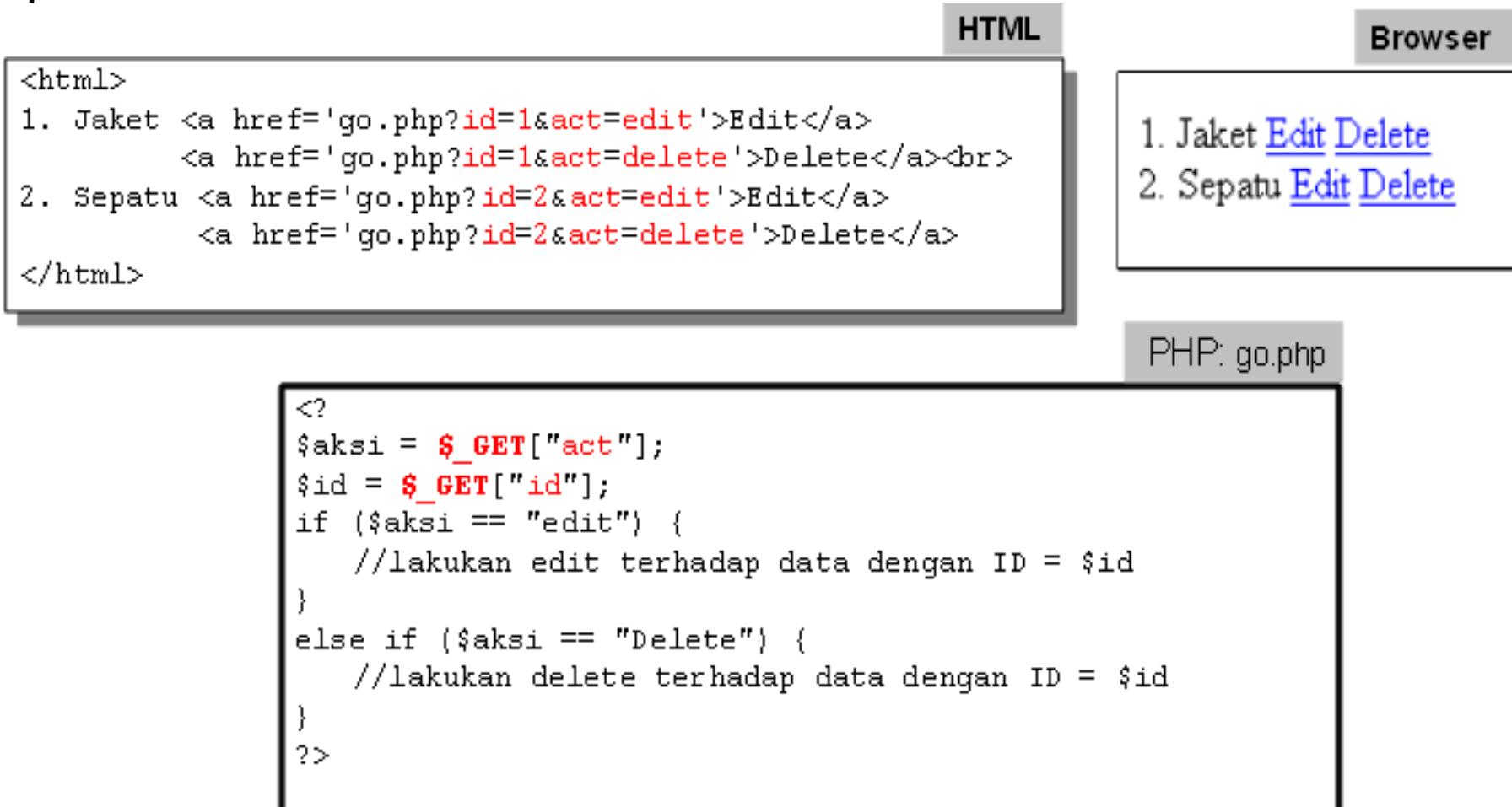
# Input Data

- Alternative of data input source:
  - Parameter URL: `$_GET`, `$_REQUEST`
  - Form handling: `$_POST`, `$_REQUEST`, `$_FILES`
  - Cookie: `$_COOKIE`
  - Session: `session_start()`, `$_SESSION`
  - File: `fopen()`, `fread()`, `fclose()`, `dll`
  - Database: `connect`, `select_db`, `query`, `fetch`

# Output

- Output alternative:
  - HTML: echo
  - Image: imagejpeg(), imagegif(), imagepng()
  - File: fopen(), fwrite(), fclose()
  - Cookie: setcookie()
  - Session: session\_start(), \$\_SESSION
  - Database: connect, select\_db, query

- Used to indicate which link is clicked by the user
- Each link represents data/action



# Input from HTML form

HTML	Browser
<pre>&lt;html&gt; &lt;form action='save.php' method='POST'&gt;     Nama&lt;br&gt;     &lt;input type='text' name='nama'&gt;&lt;br&gt;     Jenis&lt;br&gt;     &lt;input type='radio' name='jenis' value='L'&gt;Laki-laki&lt;br&gt;     &lt;input type='radio' name='jenis' value='P'&gt;Perempuan&lt;br&gt;     &lt;input type='submit' value='Simpan'&gt; &lt;/form&gt; &lt;/html&gt;</pre>	<p>Nama <input type="text"/></p> <p>Jenis</p> <p><input type="radio"/> Laki-laki</p> <p><input type="radio"/> Perempuan</p> <p><input type="button" value="Simpan"/></p>

PHP: save.php

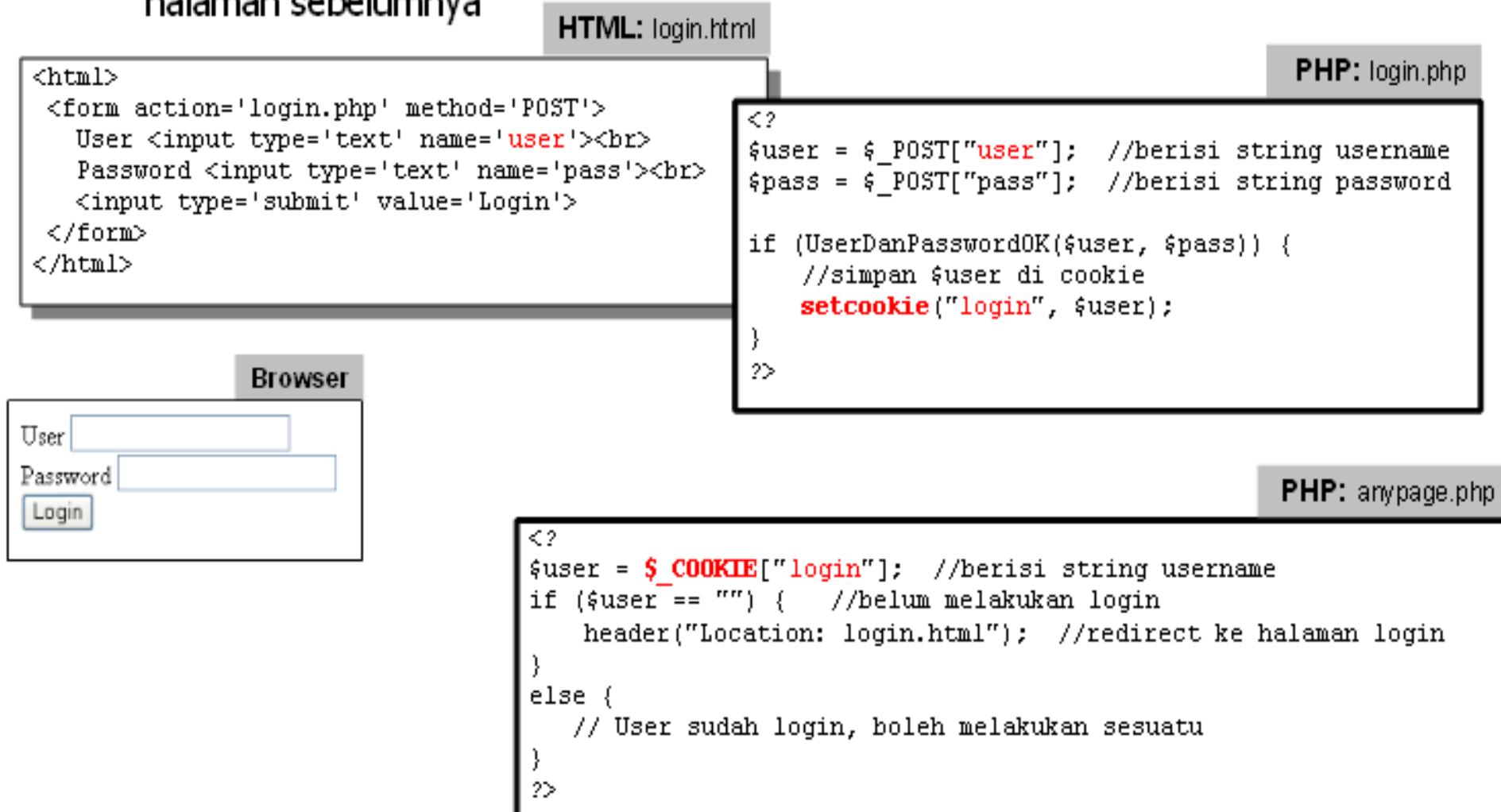
```
<?
$nama = $_POST["nama"]; //berisi string nama
$jenis = $_POST["jenis"]; //berisi "L" atau "P"

//simpan data $nama dan $jenis

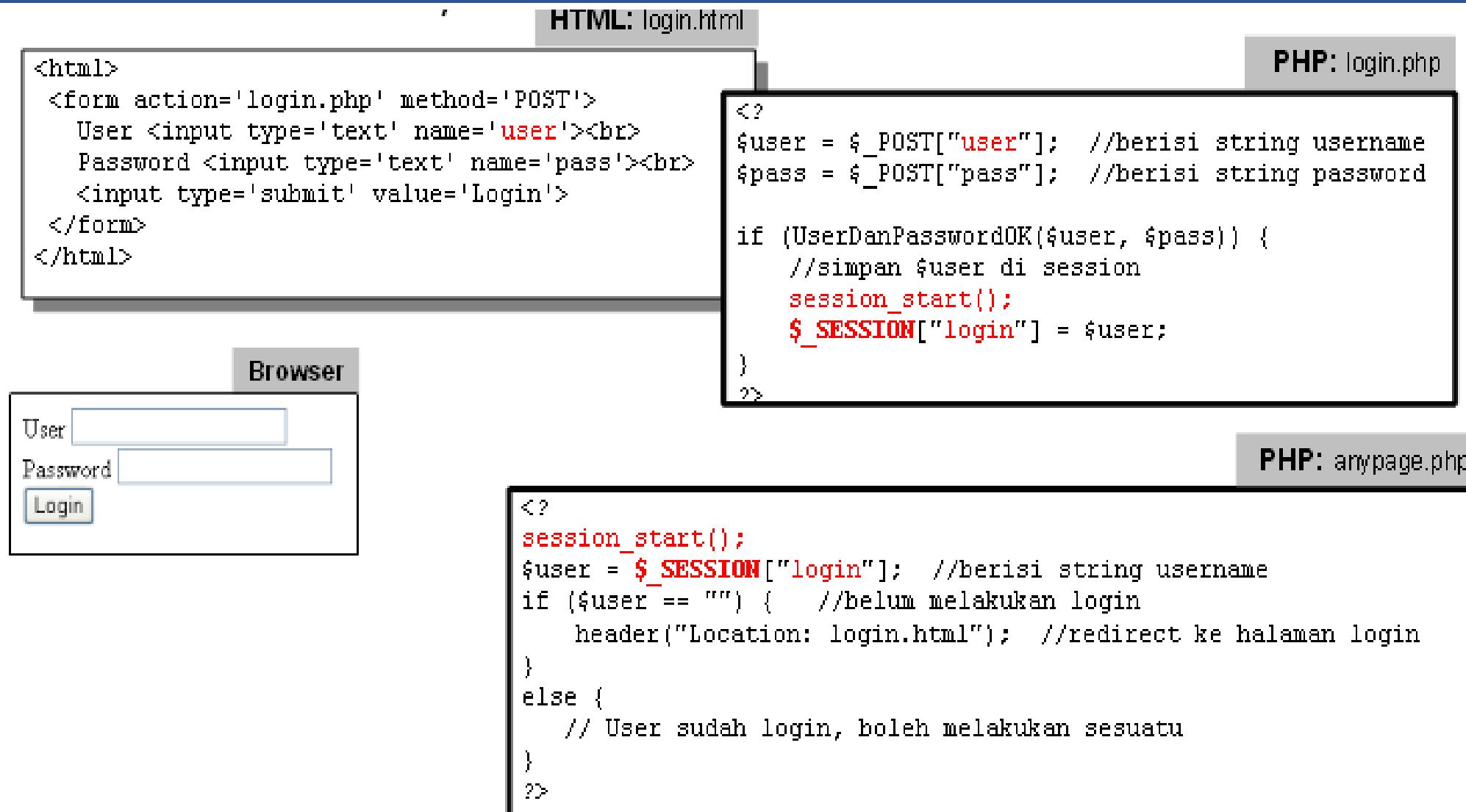
?>
```

# Input from Cookie

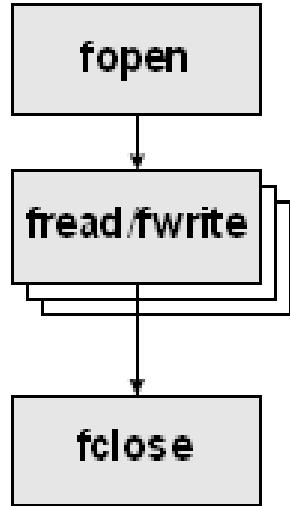
- Dapat digunakan untuk mendapatkan data yang dimasukkan oleh user pada halaman sebelumnya



# Input from Session



# File Access



```
<?
$namafile = "log.txt";

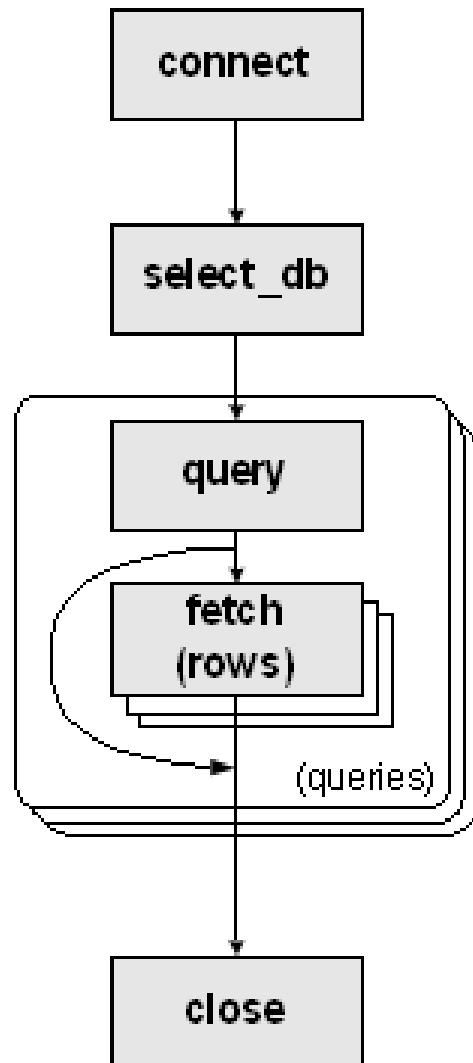
//Contoh menulis ke file
$fw = fopen($namafile, "w"); //buka untuk ditulisi
fwrite($fw, "2006-02-12 User Budi melakukan login\n");
fwrite($fw, "2006-02-15 User Toni melakukan login\n");
fwrite($fw, "2006-02-17 User Budi menambah data\n");
fclose($fw);

//Contoh membaca file
$fr = fopen($namafile, "r"); //buka untuk dibaca
while ($line = fread($fr)) {
 echo $line;
}
fclose($fr);

//Contoh membaca isi file dan memasukkan isinya ke sebuah variabel
$sisifile = file_get_contents($namafile);
?>
```

PHP

# Database Access



PHP

```
$server = "167.205.1.2"; //database server
$userid = "tedi";
$password = "asdf";
$basisdata = "mhs";
$link = mysql_connect($server, $userid, $password);

mysql_select_db($basisdata, $link);

//contoh menyimpan data
$query = "insert into t_mahasiswa values('135', 'Budi', 'L')";
mysql_query($query);

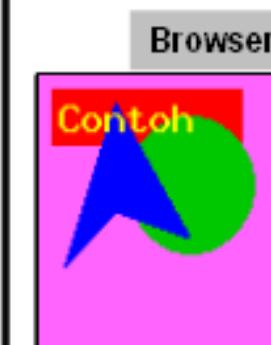
//contoh membaca data
$query = "select nim, nama, jenis from t_mahasiswa";
$result = mysql_query($query, $link);
while ($row = mysql_fetch_array($result)) {
 echo $row["nama"]."
";
}

mysql_close($link);
```

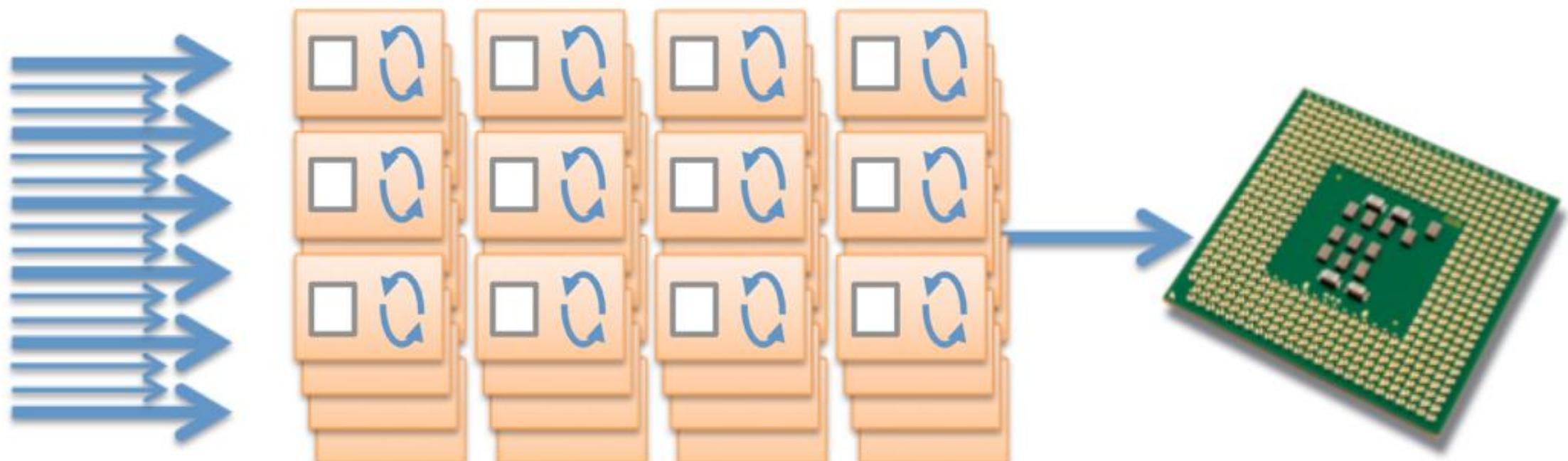
# Image

PHP

```
<?
//HTTP header yang menyatakan bahwa output mempunyai format GIF
header("Content-type: image/gif");
//membuat image baru di memory
$im = imagecreate(100, 100); //width,height
//definisi warna pertama untuk background
$backgroundcolor = imagecolorallocate($im, 255, 0, 255); //purple
//contoh definisi warna lainnya sesuai kebutuhan
$redcolor = imagecolorallocate($im, 255, 0, 0); //red
$greencolor = imagecolorallocate($im, 0, 200, 0); //green
$bluecolor = imagecolorallocate($im, 0, 0, 255); //blue
$yellowcolor = imagecolorallocate($im, 255, 255, 0); //yellow
//contoh menggambar persegi panjang
imagefilledrectangle($im, 5, 5, 80, 25, $redcolor); //x1,y1,x2,y2,color
//contoh menggambar lingkaran
imagefilledellipse($im, 60, 40, 50, 50, $greencolor); //xcenter,ycenter,width,height
//contoh menggambar poligon
$points = array(30,10,60,60,30,50,10,70); //x1,y1,x2,y2,x3,y3,x4,y4
imagefilledpolygon($im, $points, 4, $bluecolor); //arraypoints,numpoints,color
//contoh menggambar teks
imagestring($im, 5, 8, 8, "Contoh", $yellowcolor); //fontsize,x,y,color
//outputkan ke browser
imagegif($im);
//hapus dari memory
imagedestroy($im);
?>
```



# PHP Execution Model: Apache HTTPD



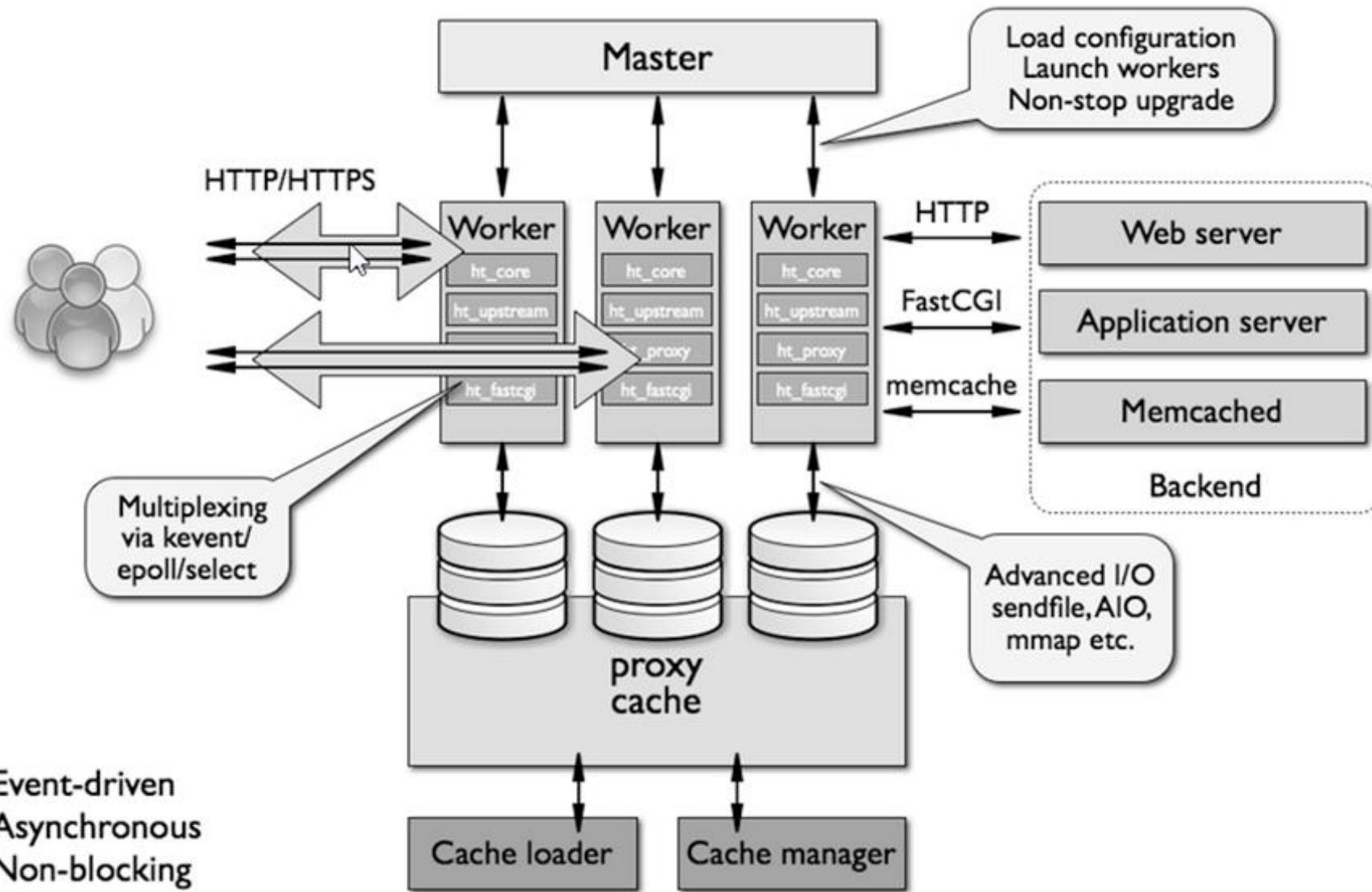
Hundreds of concurrent  
connections...

- Process-driven Approach
- Create a new thread for each request

require hundreds of heavyweight  
threads or processes...

competing for limited  
CPU and memory

# PHP Execution Model: NGINX



# Additional Stuffs in PHP 7

- Speed: benchmarks for PHP 7 consistently show speeds twice as fast as PHP 5.6, and even faster
- Optional strict typing
- Error/exception handling
- New operators
  - “Spaceship” operator
  - Null coalesce operator

```
$compare = 2 <= 1
2 < 1? return -1
2 = 1? return 0
2 > 1? return 1

$name = $firstName ?? "Guest";
```

# State Handling

IF3110 – Web-based Application Development  
School of Electrical Engineering and Informatics  
Institut Teknologi Bandung

# Reference

## RFC 6265 – HTTP State Management

<https://tools.ietf.org/html/rfc6265>

# Task in Web Application

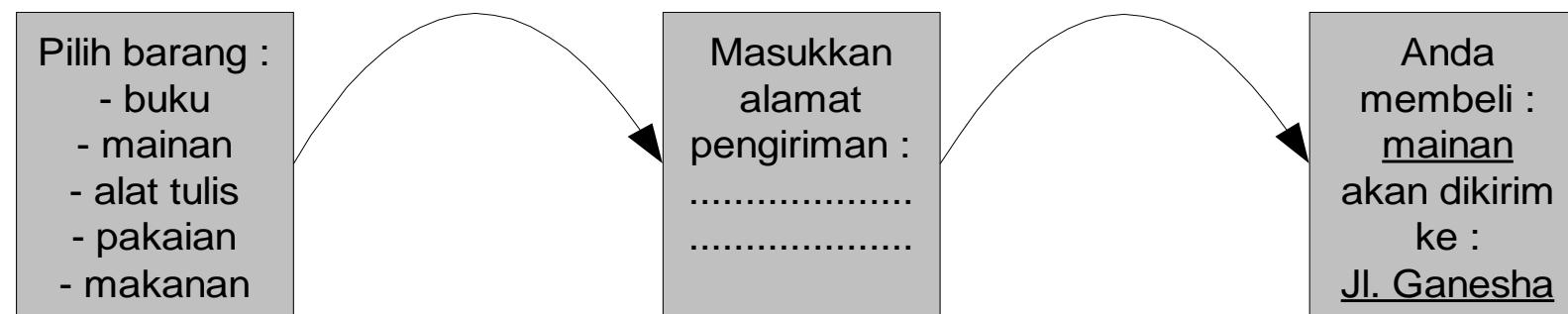
Application's features allow users to perform various tasks

To perform a task might involve one or more pages (as interface)

Example

1 page: read an article, add entry in a guestbook

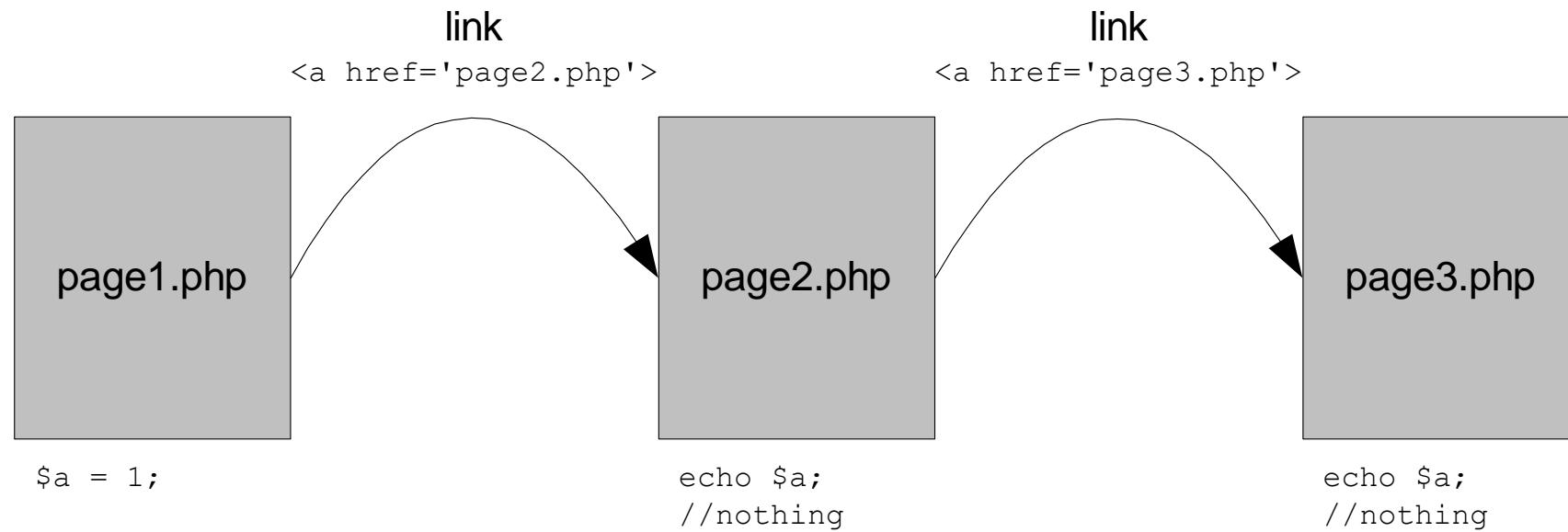
more pages: buying things online



Between pages should have a mechanism to manage the *state* of the task

# Stateless HTTP

HTTP is *stateless*; no *state* is preserved between requests (at protocol level)

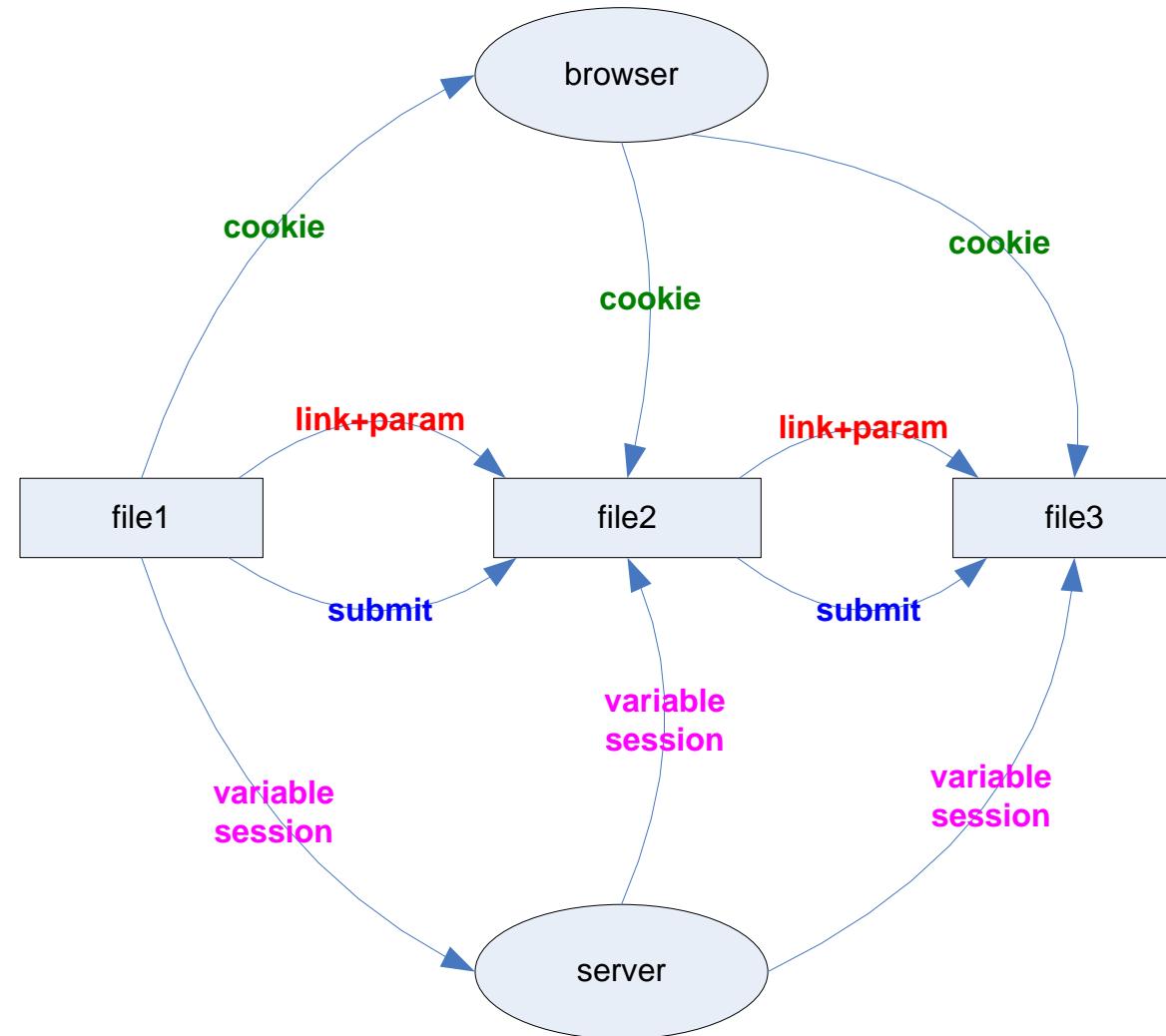


# State Management

## Definition

a process to maintain state and page information over multiple requests for the same/different pages

# State Handling



# Common Techniques

Technique to overcome this limitation

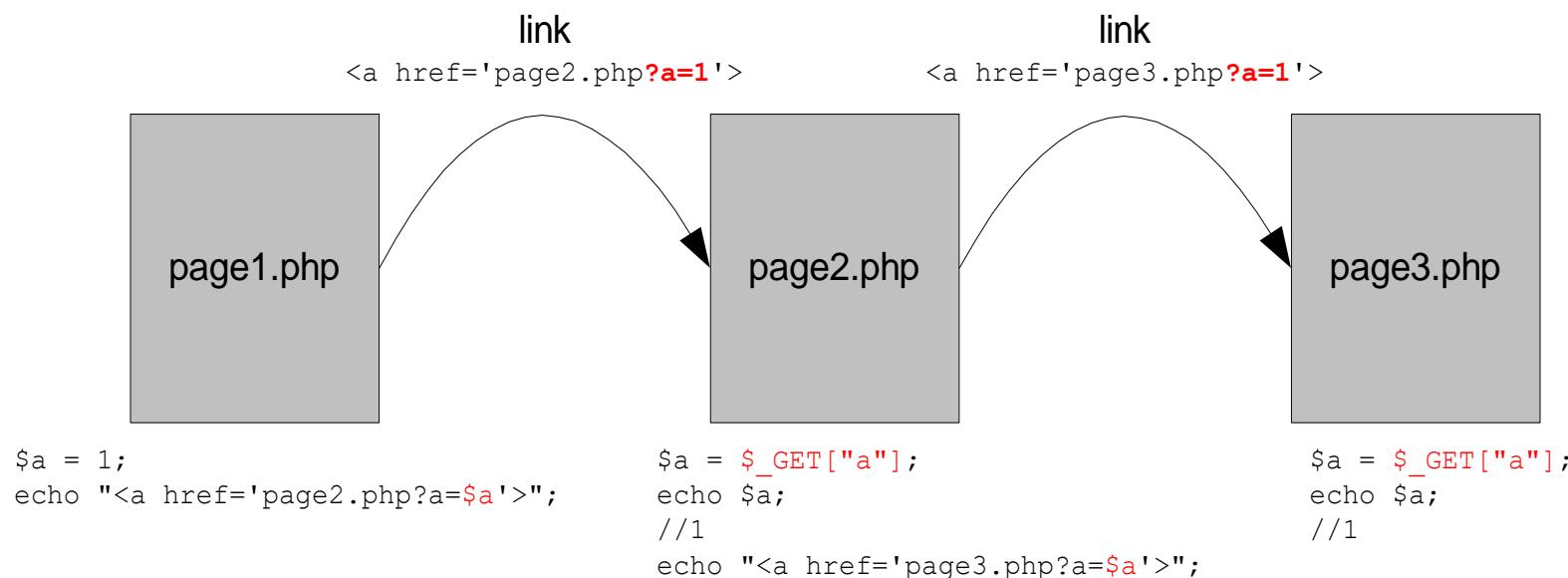
- message passing via URL/form
- Hidden field
- Cookie
- Session

# Message Passing via URL

Variable is passed in the URL's parameter

Request: construct the parameter(s) in a URL

Response: access via `$_GET`



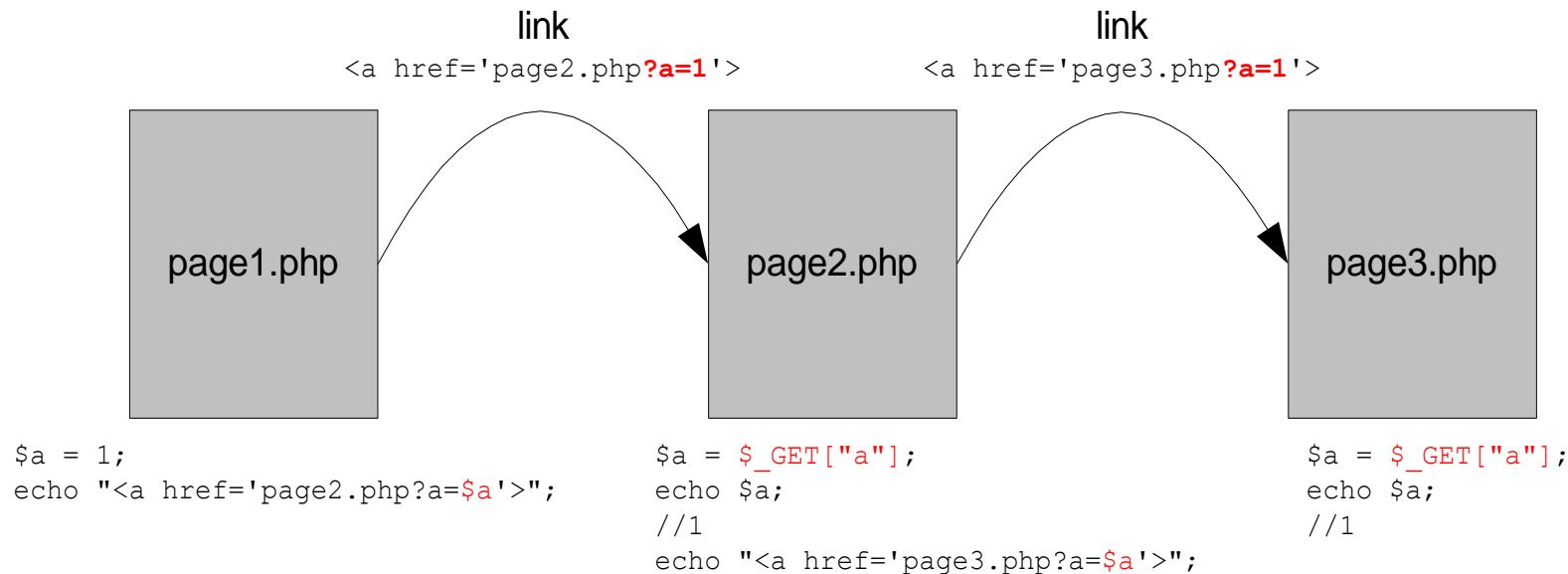
# Hidden Field

Variable is passed in the hidden field

```
<input type="hidden" name="a" value="1">
```

Request: construct the parameter(s) in a URL or body

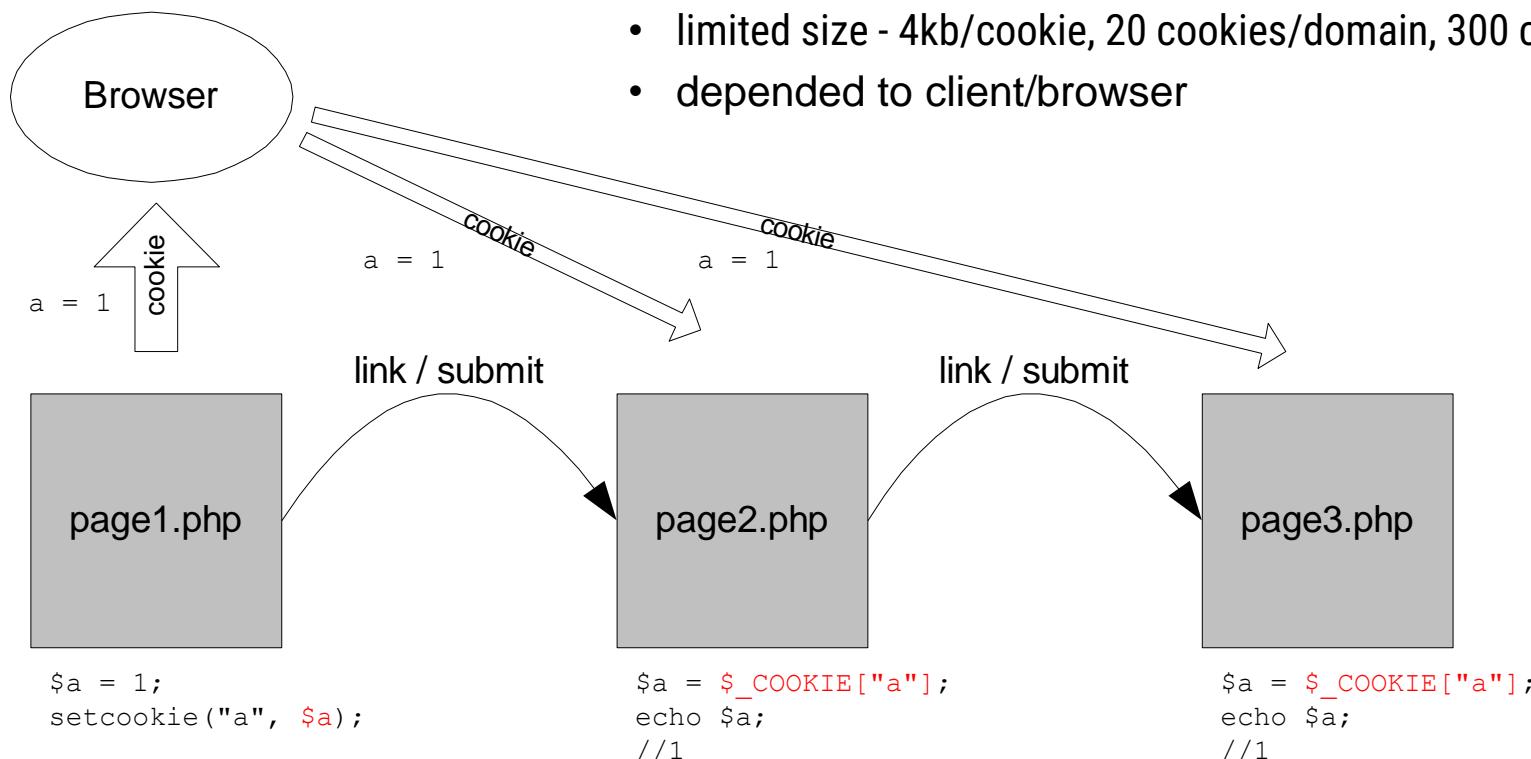
Response: access via `$_GET` or `$_POST`



# Cookie

Variable is stored in the browser

- Request: setcookie ("name", "value")
- Response: access via `$_COOKIE`
- Limitation:
  - limited size - 4kb/cookie, 20 cookies/domain, 300 cookies/client
  - depended to client/browser

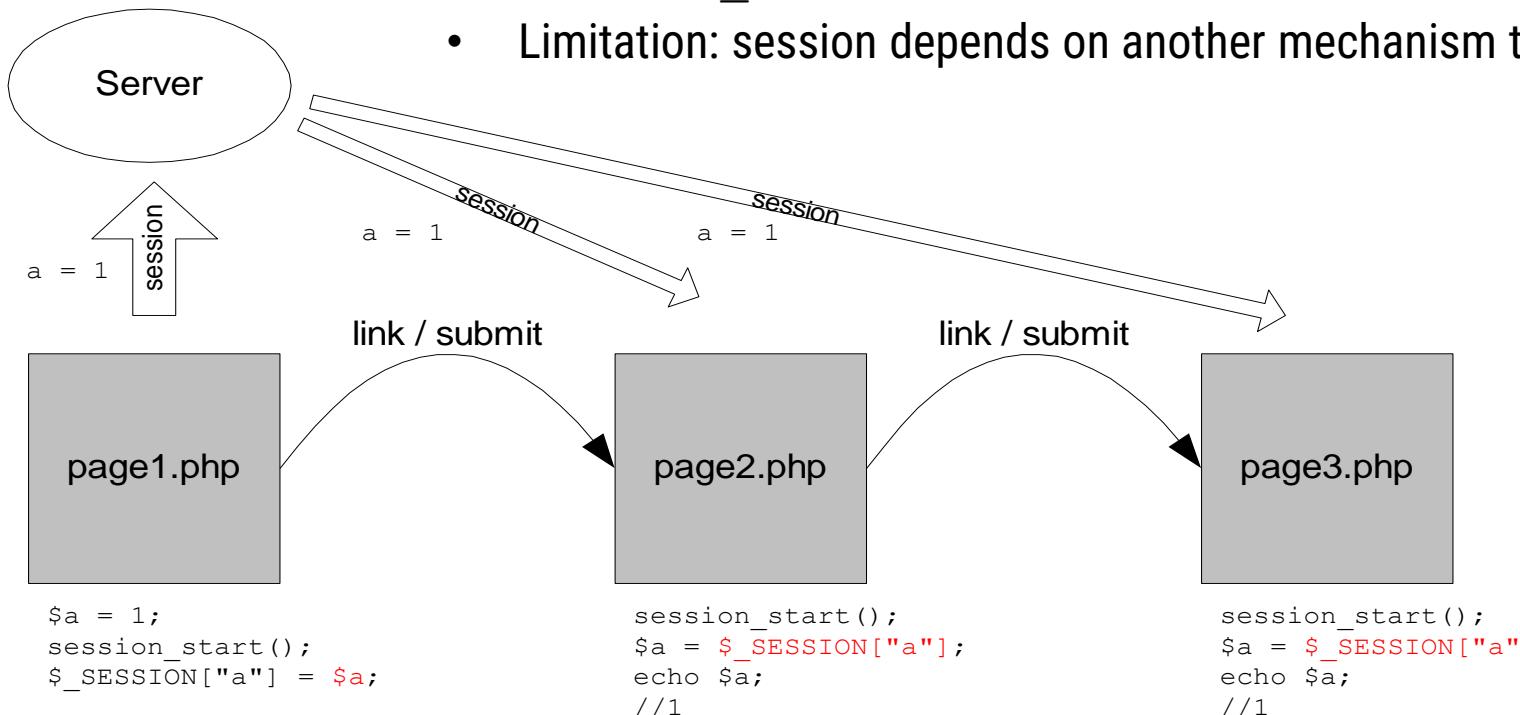


# Session

Variable is stored in server

Request and Response via `$_SESSION`

- Function related to session handling: `session_start()`, `session_id()`, `session_destroy()`, etc
- Limitation: session depends on another mechanism to store session id



# Short Question

Can you tell me the advantage & disadvantage of using

- cookies
- url
- hidden input
- session

Which one is the most secure?

# Consideration

- Size of State Representation
- 3<sup>rd</sup> Party-Cookies
- User Access to State Representation

# Server Side Data Access, Authentication

IF3110 – Web-based Application Development  
School of Electrical Engineering and Informatics  
Institut Teknologi Bandung

# Objective

- Students understand the role of data access on web application
- Students understand the concepts of authentication and relevant technologies.

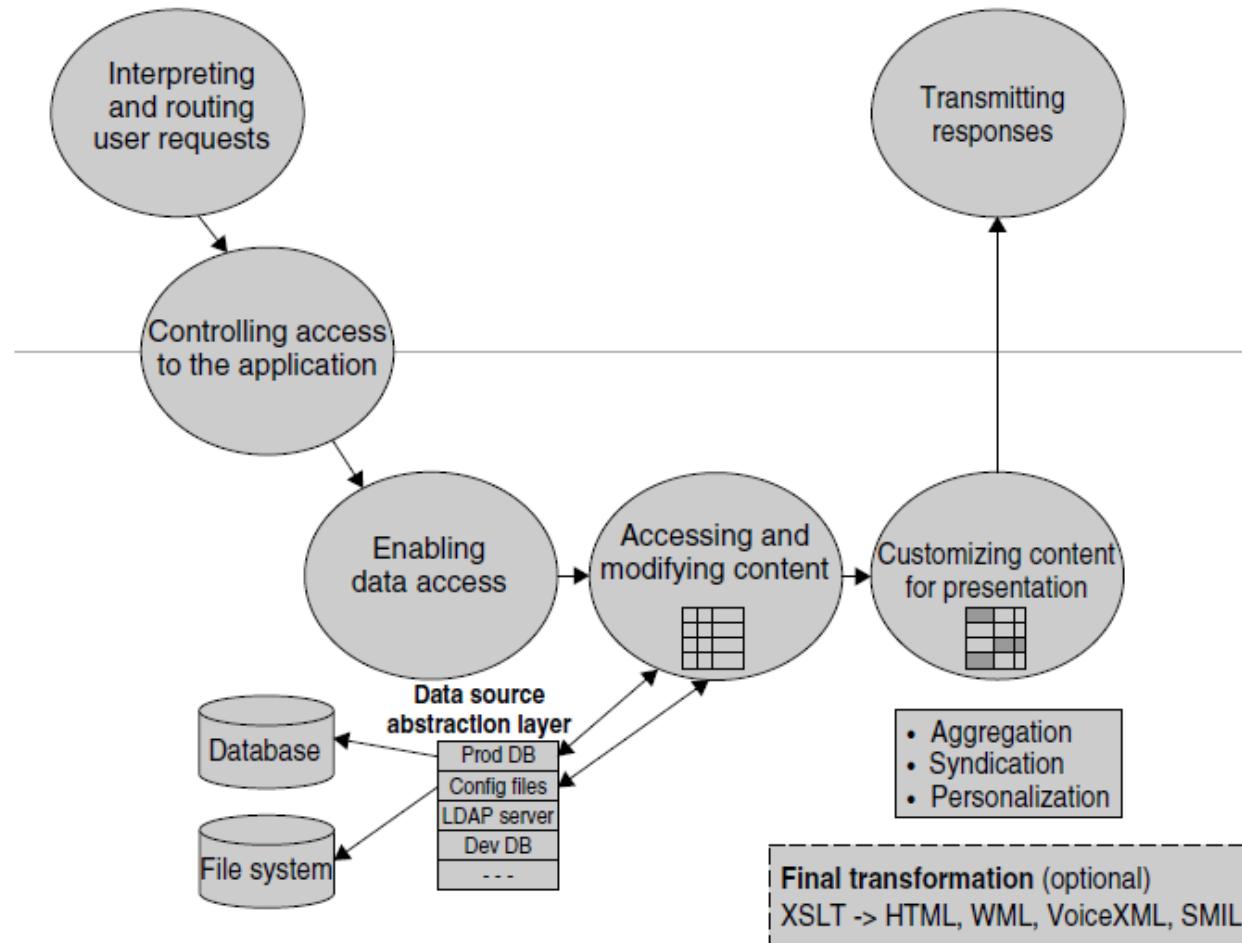
## What we have learned so far

- HTTP Protocol
- State-handling
- Form handling
- Templating

# Typical Web App Processing

Source:

Shklar, L.  
Web Application Architecture:  
Principles, Protocols and  
Practices  
*Wiley Publishing, Inc., 2003*



**Figure 8.3** Processing flow in a typical Web application (Above the grey line—Web server; below the grey line—Web application)

# Reference

- Learning PHP, David Sklar, Oreilly

# Data Access

- Why do we need to have a data access mechanisms
  - Remember your users (and their attributes)
  - Which users do these transactions
  - The “stuff” that our site manages/sells
- Why the following mechanisms are insufficient
  - URL parameters
  - Cookies
  - Session attributes

# Possible Mechanisms

- Database
- File

## Advanced

- Services
- Object Storage
- Memcache

# Database

# Things commonly done

- Connecting to

```
$db = new PDO
 ('mysql:host=db.example.com;dbname=dbname',
 'johndoe', 'secretpwd');
```

- Setting attributes (if needed)
- Handling errors (e.g., database failure, network failure)
- Putting Data into
  - Data modification
- Retrieving Data

# Things commonly done

- Putting Data into

```
try {
 $db = new PDO('sqlite:/tmp/restaurant.db');
 $db->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
 $affectedRows = $db->exec("INSERT INTO dishes
 (dish_name, price, is_spicy) VALUES ('Sesame Seed
 Puff', 2.50, 0)");
} catch (PDOException $e) {
 print "Couldn't insert a row: " . $e->getMessage();
}
```

# What it does?

```
try {
 $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
 print "Couldn't connect: " . $e->getMessage(); }
 $result = $db->exec("INSERT INTO dishes (dish_size,
dish_name, price, is_spicy) VALUES ('large', 'Sesame Seed
Puff', 2.50, 0)");
if (false === $result) {
 $error = $db->errorInfo();
 print "Couldn't insert!\n";
 print "SQL Error={$error[0]}, DB Error={$error[1]},
Message={$error[2]}\n"; }
```

# Things commonly done

- Data Modification

```
$rows = $db->exec("UPDATE dishes SET is_spicy = 1 WHERE dish_name = 'Eggplant with Chili Sauce'");
```

- Data Deletion

```
if ($make_things_cheaper) {
 $rows = $db->exec("DELETE FROM dishes WHERE price > 19.95");
} else { // or, remove all dishes
 $rows = $db->exec("DELETE FROM dishes");
}
```

- \$rows is number of affected rows

# Things commonly done

- Retrieving Data

```
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch()) {
 print "$row[dish_name], $row[price] \n";
}
```

- What is the difference from

```
$q = $db->query('SELECT dish_name, price FROM dishes');
$rows = $q->fetchAll();
```

Which one is better?

# Things commonly done

## ■ Retrieving Data as OBJECT

```
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch(PDO::FETCH_OBJ)) {
 print "{$row->dish_name} has price {$row->price}\n";
}
```

# Things commonly done... more

- Data Insertion from Request Attributes (e.g., `$_POST`)

```
$stmt = $db->exec('INSERT INTO dishes
 (dish_name) VALUES (...)');
```

- Be careful ...for injection, imagine
  - `$_POST[dish_name] = 'Fried Rice'`
  - `$_POST[dish_name] = 'Uncle Wong's Fried Rice'`

```
$stmt = $db->prepare('INSERT INTO dishes
 (dish_name) VALUES (?)');
```

```
$stmt->execute(array($_POST['dish_name']));
```

- Not using `exec()` or `query()` but using `prepare()` `execute()`

# Things commonly done... more

- Putting data in from a web form
  1. Read request parameters as inputs
  2. Validate and process the inputs
  3. Sanitize the inputs (before being SQL parameters)
  4. Prepare the SQL Statements (DML)
  5. Put the inputs as the parameters of the prepared statement
  6. Execute

# Things commonly done... more

- Presenting data in a web from database
  1. Read request parameters as inputs
  2. Validate and sanitize the inputs
  3. Prepare the SQL Statements (Query)
  4. Put the inputs as the parameters of the prepared statement
  5. Execute
  6. Put select resultset into a template of the web form

# Things commonly done... more (retrieval)

```
$stmt = $db->prepare($sql);
$stmt->execute(array($input['min_price'],
$input['max_price']));
$dishes = $stmt->fetchAll();
if (count($dishes) == 0) {
 print 'No dishes matched.';
} else {
 print '<table>';
 print '<tr><th>Dish Name</th><th>Price</th><th>Spicy?
</th></tr>';
 foreach ($dishes as $dish) {
 if ($dish->is_spicy == 1) {
 $spicy = 'Yes'; } else { $spicy = 'No'; }
 printf('<tr><td>%s</td><td>$%.02f</td>
<td>%s</td></tr>', htmlentities($dish->dish_name),
$dish->price, $spicy); }}
```

# Remarks

- Often you only have a single instance of SQL DBMS
  - Chocking point to the Web App Performance, Single Point of Failure
- Be careful with SQL wildcard
  - Lead to slow query
  - Unnecessary big result sets
- Never trust other inputs
  - Sanitize before store in Database
  - Sanitize/escape before printing to client (after retrieval from database)
- Open and Close Database Connection wisely

# File

# Things commonly done

- Reading a File
- Writing a File
- Working with CSV

# Read a File

```
$page = file_get_contents('page-template.html');
$page = str_replace('{page_title}', 'Welcome', $page);
if (date('H' >= 12)) {
 $page = str_replace('{color}', 'blue', $page);
} else { $page = str_replace('{color}', 'green', $page); }

$page = str_replace('{name}', $_SESSION['username'],
$page);
print $page;
```

- What it does?
- Handle the error/exception wisely

# Write a File

```
$page = file_get_contents('page-template.html');
$page = str_replace('{page_title}', 'Welcome', $page);
if (date('H' >= 12)) {
 $page = str_replace('{color}', 'blue', $page);
} else { $page = str_replace('{color}', 'green', $page); }

$page = str_replace('{name}', $_SESSION['username'],
$page);
file_put_contents('page.html', $page);
```

- What it does?
- Handle the error/exception wisely

# Read a File

```
$fh = fopen('people.txt', 'rb') ;
while ((!feof($fh)) && ($line = fgets($fh))) {
 $line = trim($line) ;
 $info = explode('|', $line) ;
 print '' .
$info[1] . "\n" ;
}
fclose($fh) ;
```

- rb is a flag to read from beginning; return false if doesn't exist

# Working with CSV

- How to put data in CSV into Database?
  1. Open the CSV
  2. Prepare the SQL statement
  3. Parse CSV line into array to be SQL Parameters
  4. Close file (and close DB Connection if necessary)

# Remarks

- Handle file errors correctly (open, permission, space, etc.)

```
$page = file_get_contents('page-template.html');
if ($page === false) {
 print "Couldn't load template: $php_errormsg";
} else { // ... process template }
```

- Sanitized supplied filenames

- Filename

```
$_POST['user'] =
'/usr/local/data/../../../../etc/passwd'
```

```
$user = str_replace('/', ' ', $_POST['user']);
$user = str_replace('..', ' ', $user);
```

```
$user = $_POST['user'];
if (is_readable("/usr/local/data/$user")) {
 print 'User profile ' . htmlentities($user) . ':
';
 print file_get_contents("/usr/local/data/$user"); }
```

# Object Storage

- It is a computer data storage that manage data as objects, as opposed to file system, data blocks, or records
- Common flows for Upload
  - WebApp requests to Object Storage for Singed URL to upload a file
  - Client uploads the file via the Signed URL
  - Signed URL is only valid for a short of time
- Flow Download
  - WebApp requests to Object Storage for Singed URL to download a file
  - Client shows/downloads the file via the signed URL

# Authentication

# Authentication

- Identity Verification
  - Verify the claims
- How can Bob be sure that he is communicating with Alice?
- Three General Ways:
  - Something you *know* (i.e., *Passwords*)
  - Something you *have* (i.e., *Tokens*)
  - Something you *are* (i.e., *Biometrics*)

# Why do we need Authentication

- Web App uses an HTTP (Stateless Protocol)
- A web app is commonly used by many users
  - I am Yudis and I want a new book
  - I am Riza, a lecture, and I need to access the lectures that I teach
  - I am Catur and I need to upload new materials

# Where to store

- Cookie
- Session Attribute
- HTTP Header



# How a web app do Authentication

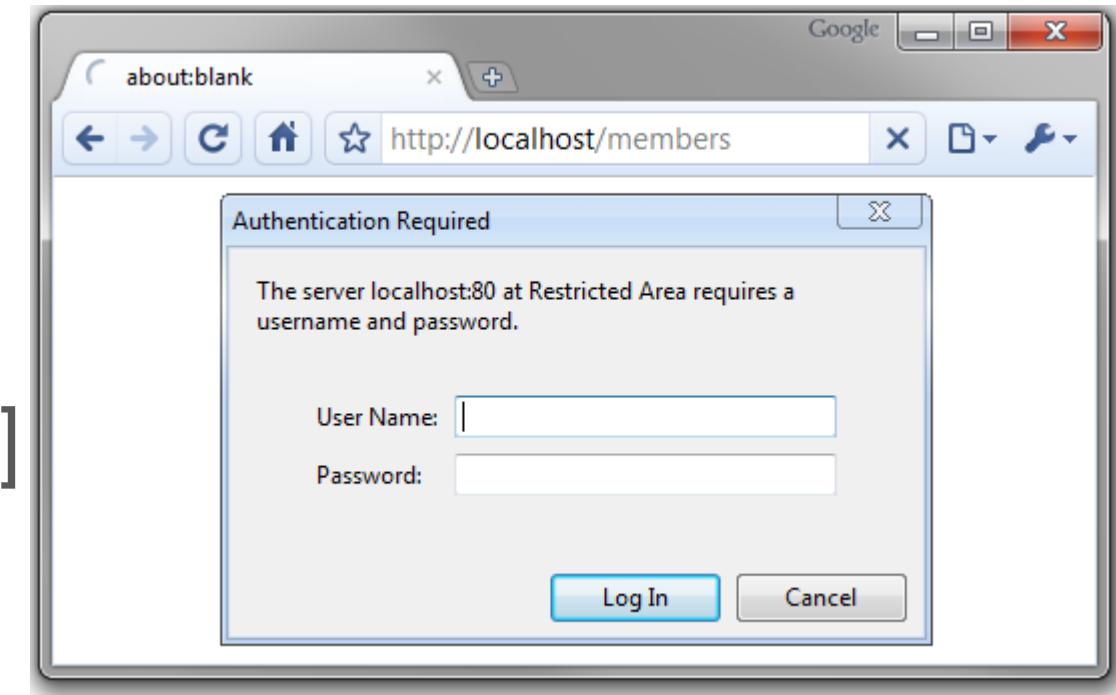
- Backed by internal user database
  - HTTP Basic Auth
  - Database-backed Auth
- External services
  - LDAP
  - OpenID+Connect (Open ID, OAuth2)
  - SAML
  - CAS
  - ...

# Common flow

1. Display web form (or Dialog in the case HTTP Basic Auth)
2. Enter the credential
3. Checking the form submission
4. (if the submitted credential is correct) adding the username in the session
5. ....browsing...
6. Remove the username from the session when the user logout/timeout

# HTTP Basic Authentication

- PHP will define the params
  - `$_SERVER['PHP_AUTH_USER']`
  - `$_SERVER['PHP_AUTH_PW']`
  - `$_SERVER['PHP_AUTH_DIGEST']`



# Database-backed Authentication [simplified]

- Checking the credential in Database

```
session_start();
$password_ok = false;
$input['username'] = $_POST['username'] ?? '';
$submitted_password = $_POST['password'] ?? '';
$stmt = $db->prepare('SELECT password FROM users WHERE
username = ?');
$stmt->execute($input['username']); $row = $stmt->fetch();
if ($row) { $password_ok =
 password_verify($submitted_password, $row[0]); }
if (! $password_ok) { $errors[] = 'Please enter a valid
username and password.'; }
else { $_SESSION['valid_user'] = $input['username'] }
```

# Database-backed Authentication [simplified]

- If success

```
session_start();
if (isset($_SESSION['valid_user'])) {
 do_html_menu();
 //contents
 do_html_footer();
} else {
 //redirect to login page
}
```

- End user session

```
session_start();
unset($_SESSION['valid_user']);
$res = session_destroy();
```

# Database-backed Authentication [simplified]

- Insert/Update Password
  - Update query in SQL with prepared statement
  - Never store the password in plaintext
    - Hash – sha1, sha2
    - Hash+Salt – password\_hash()
- Reset password
  - ?

# Common Protocols on Web

# Using LDAP for Authentication

- LDAP – Lightweight Directory Access Protocol
  - A “database” for user directory

```
$ldap = ldap_connect("ldap://ldap.mydomain.com") or die('Could
not connect to LDAP server.');//
ldap_set_option($ldap, LDAP_OPT_PROTOCOL_VERSION, 3);
ldap_set_option($ldap, LDAP_OPT_REFERRALS, 0);
$bind = @ldap_bind($ldap, $ldapuser, $ldappass);
if ($bind) {
 $filter="(&sAMAccountName=$username)";
 $result = ldap_search($ldap,"dc=MYDOMAIN,dc=COM",$filter);
 ldap_sort($ldap,$result,"sn");
 $info = ldap_get_entries($ldap, $result);
 if($info['count'] > 0)
 // exists entries
```

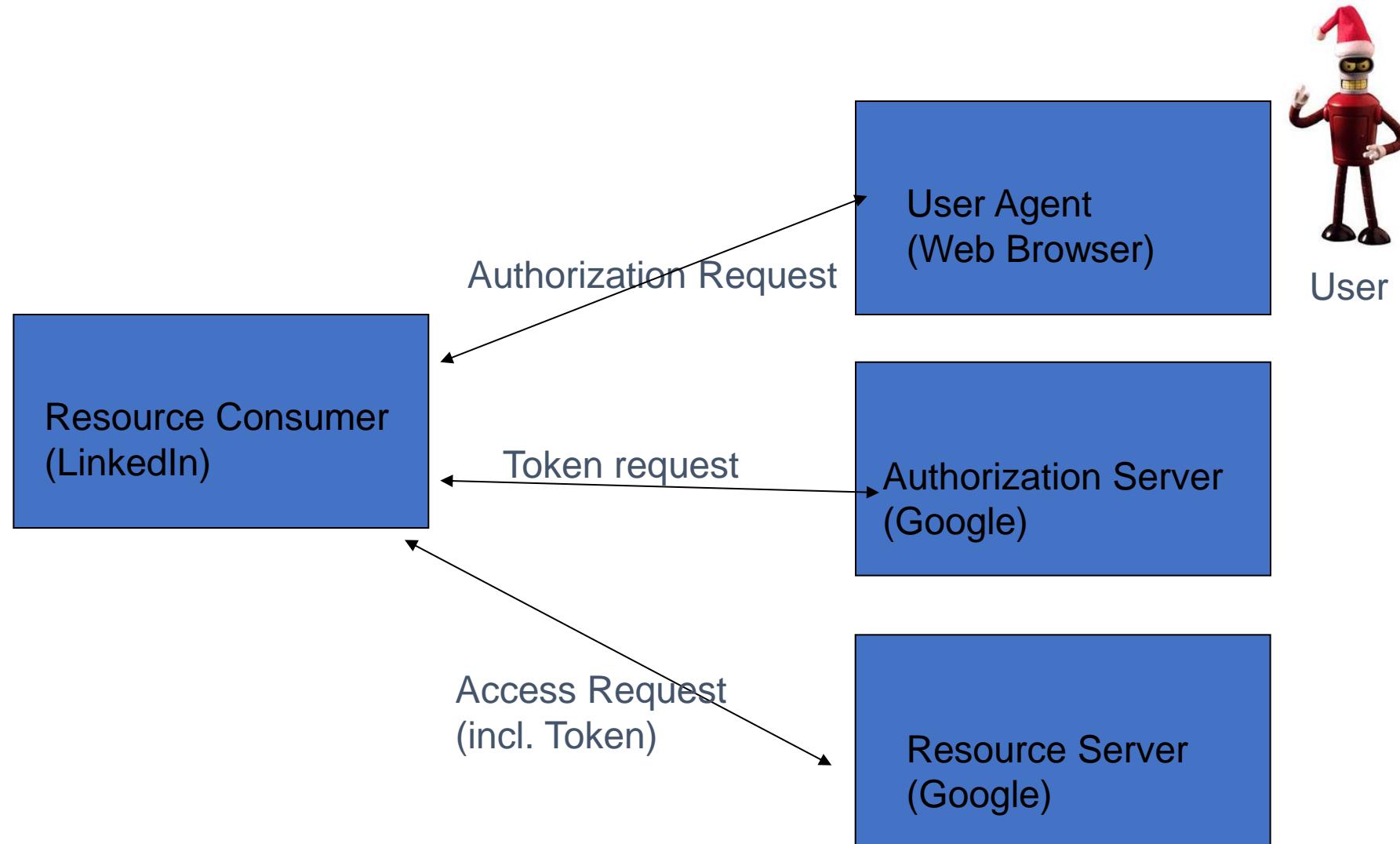
# OAuth - Features

- A third party app can access user's data stored at service provider without requiring username and password.
- Delegated authorization protocol
- Explicit user consent is mandatory.
- Light-weight\*
- Use Case
  - Website X can access your protected data at API Y
    - All without sharing your password off-site
    - especially when there isn't one like with OpenID
- Approach
  - Signed HTTP Requests
  - Safe, Password-less Token Exchange

# Three things

- Actors
  - User
  - Service Provider
  - Consumer
- Token
  - Access Token
  - Request Token
  - Consumer Key
- URLs
  - Request Token Issuer
  - Authorization Page
  - Access Token Exchanger

# Entities



# User navigates to Resource Client

**Build your network (Why?)** X

**Find contacts who are already on LinkedIn**

---

**Web email contacts**  
Check your address book to find contacts who are on LinkedIn.

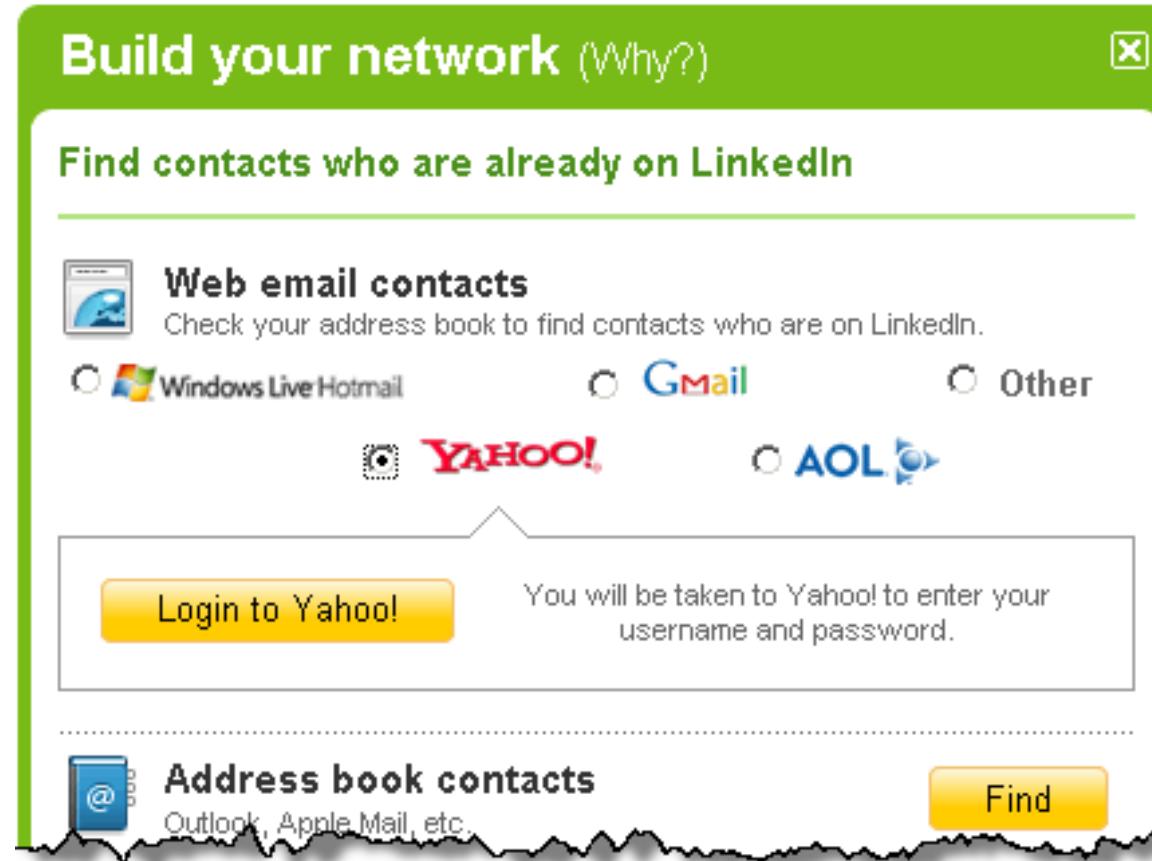
Windows Live Hotmail       Gmail       Other

YAHOO!       AOL

**Login to Yahoo!**      You will be taken to Yahoo! to enter your username and password.

---

**Address book contacts**  
Outlook, Apple Mail, etc. Find



# User authenticated by Authorization Server

The screenshot shows a Microsoft Internet Explorer window titled "Sign in to Yahoo! - Microsoft Internet Explorer provided by NOKIA". The address bar contains the URL [https://login.yahoo.com/config/login?.src=appuser&.done=https%3A%2F%2Fapi.login.yahoo.com%2FWSLogin%2FV1%2Fwslogin%3Fappid%3DcVa\\_PAfIkY2KjGtd2KZejlTUNp8fLBX4KmA%26ts%3D1](https://login.yahoo.com/config/login?.src=appuser&.done=https%3A%2F%2Fapi.login.yahoo.com%2FWSLogin%2FV1%2Fwslogin%3Fappid%3DcVa_PAfIkY2KjGtd2KZejlTUNp8fLBX4KmA%26ts%3D1). The main content area displays the Yahoo! login interface. The page starts with "To start using this service...". It lists two steps: "Step 1: Sign in to Yahoo!" and "Step 2: Give your permission.". The "Sign in to Yahoo!" step includes a yellow box for "Are you protected? Create your sign-in seal. (Why?)". The login form has fields for "Yahoo! ID:" and "Password:", a "Keep me signed in" checkbox, and a "Sign In" button. Below the form are links for "Forget your ID or password? | Help" and "Don't have a Yahoo! ID? Signing up is easy. Sign Up". The bottom section features a box for "One Yahoo! ID. So much fun!".

# User authorizes Resource Consumer to access Resource Server

The screenshot shows a Microsoft Internet Explorer window with the title bar "Yahoo! - Terms - Microsoft Internet Explorer provided by NOKIA". The address bar contains the URL [https://api.login.yahoo.com/WSLogin/V1/wslogin?appid=cVa\\_PAfIkY2KjGtd2iKZejITUNp8FLBX4KmA&ts=1215323357&sig=a1401bd223c0127d681911983863a633&scrumb=hmMDaryi.3I](https://api.login.yahoo.com/WSLogin/V1/wslogin?appid=cVa_PAfIkY2KjGtd2iKZejITUNp8FLBX4KmA&ts=1215323357&sig=a1401bd223c0127d681911983863a633&scrumb=hmMDaryi.3I). The main content is a Yahoo! login page for LinkedIn. The page features the Yahoo! logo and the heading "Now we need your permission to grant access to your Yahoo! account". It explains that LinkedIn is asking for permission to automatically log the user into their Yahoo! account. The requested permissions are listed as:

- read your data in **Yahoo! Address Book**
- read and write to your data in **Yahoo! Address Book**

Below this, it states: "By clicking "I Agree" below, you give Yahoo! permission to enable <http://www.linkedin.com> to access your Yahoo! account for this purpose, and further agree to the Automatic Login Terms of Service below."

Under "Keep in mind:", the following points are listed:

- <http://www.linkedin.com> will not be able to access any data you keep on Yahoo! other than the data identified above.
- The permission will expire in 2 weeks.
- You can change this permission by visiting the [My Account](#) page and selecting the [Partner Accounts](#) link. Note that revoking permission may take up to 24 hours.
- If you change your password, you may be required to give permission again.
- The Yahoo! privacy policy does not apply to <http://www.linkedin.com>; please read their privacy policy to learn more about how they treat your personal information.
- Yahoo! has no affiliation with <http://www.linkedin.com> and cannot guarantee the security of any user data that you permit <http://www.linkedin.com> to access.

A modal dialog box titled "Sign-in Permissions" is displayed, containing the "Automatic Login Terms of Service - Please read carefully" and a scrollable text area with the following content:

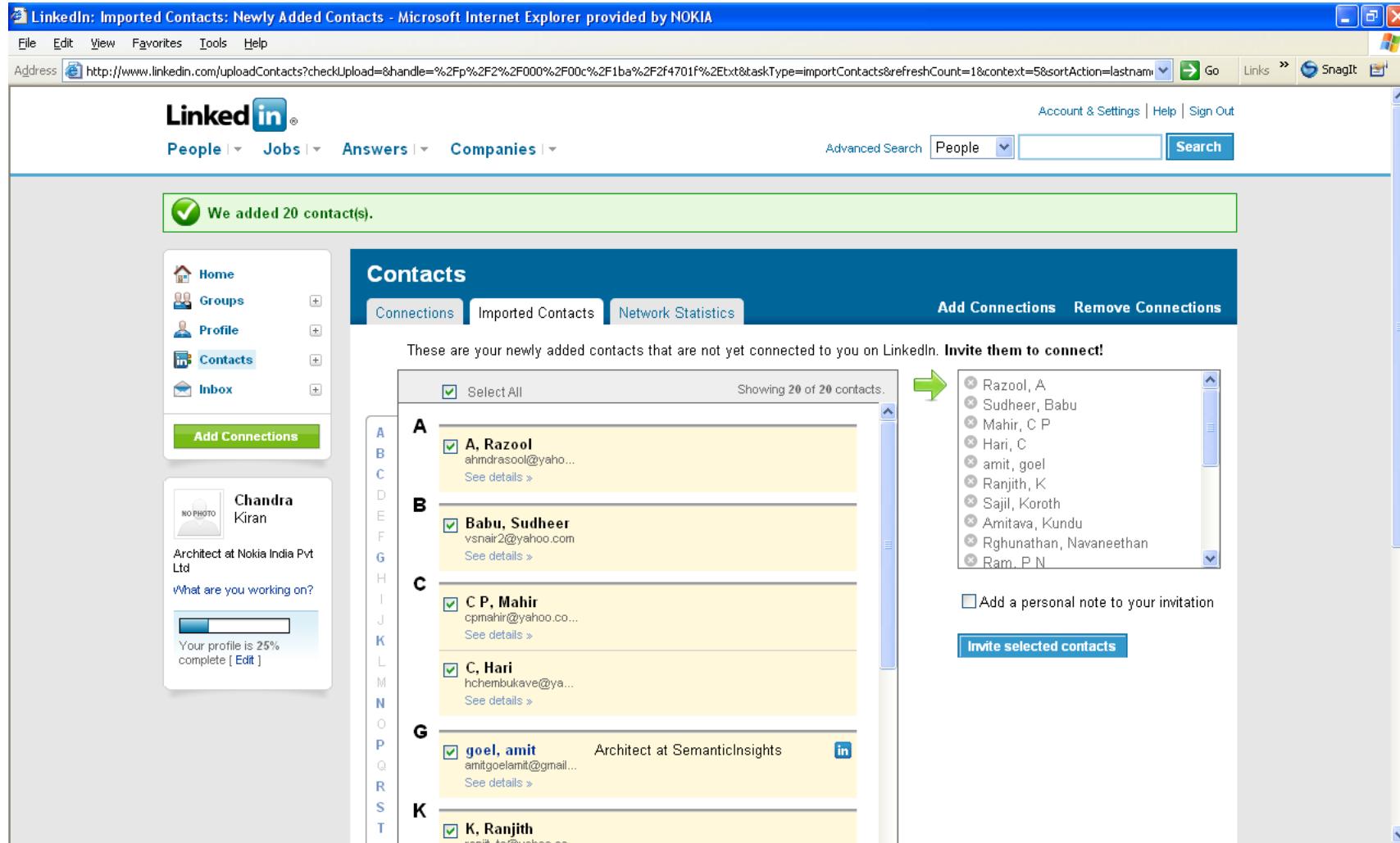
Please review the following terms and indicate your agreement below.  
View all and print

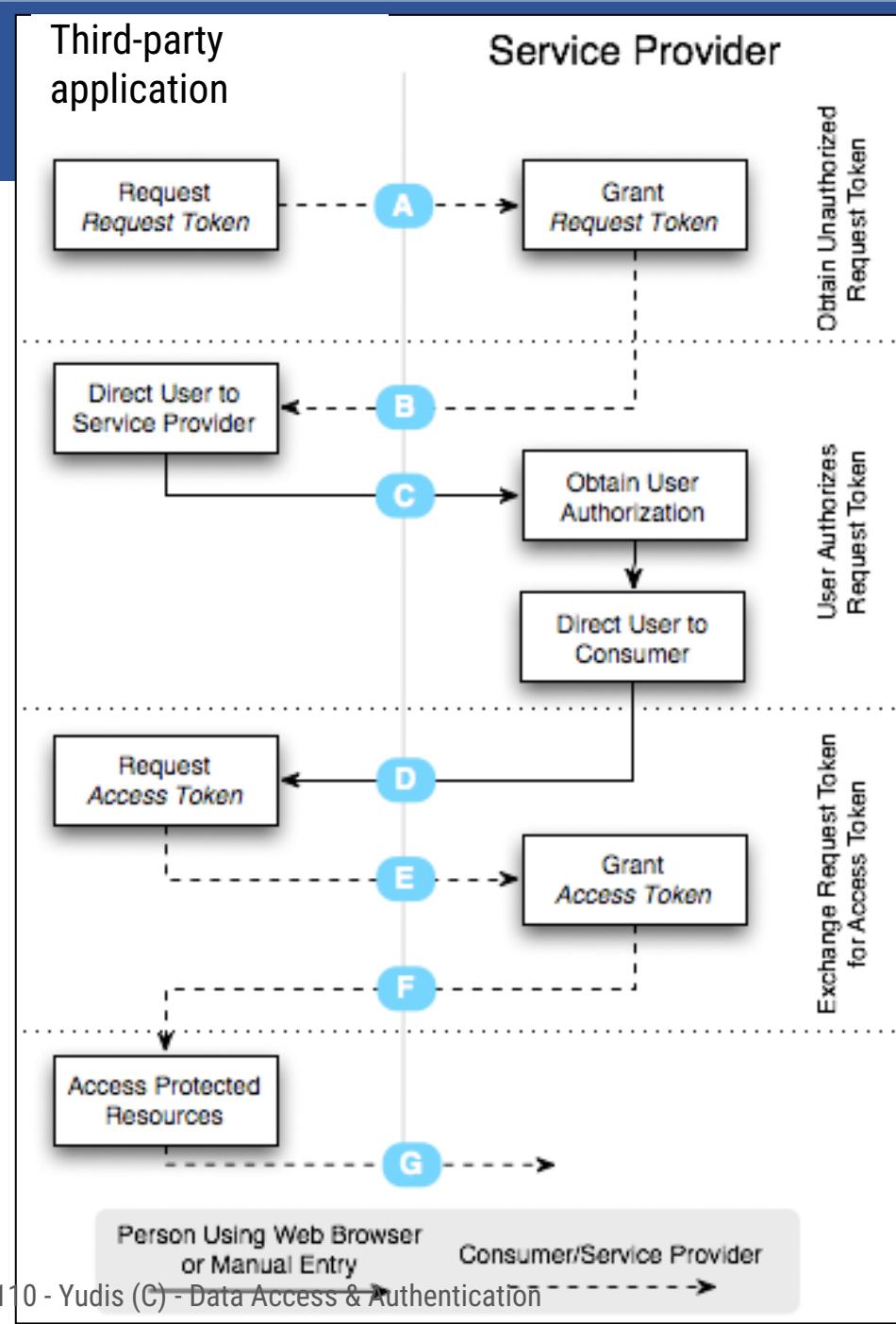
Your use of automatic login with third party sites is at your sole risk. While Yahoo! takes measures to protect the privacy and

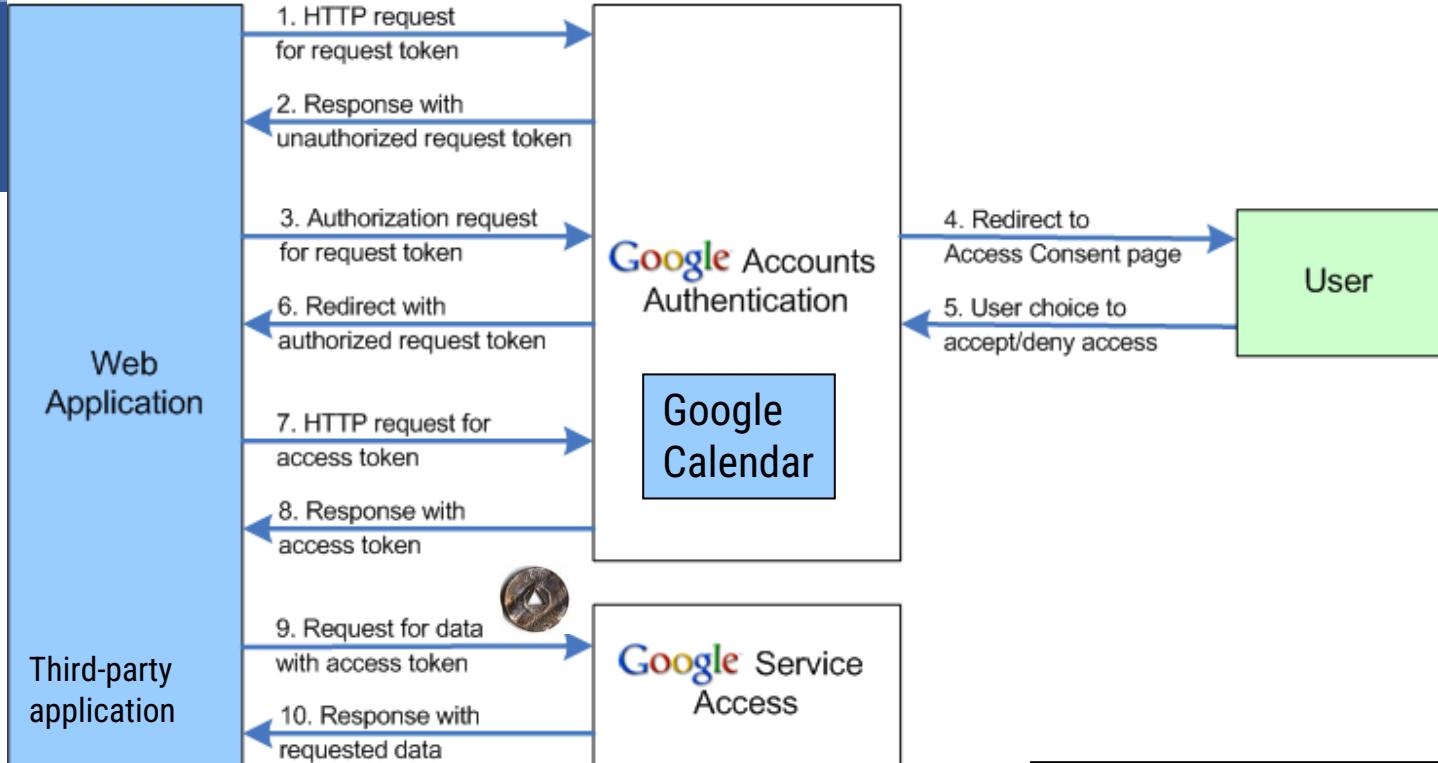
By clicking "I agree", you agree that you have read and understand these terms.

I Agree   I Do Not Agree

# Resource Client calls the Resource Server API





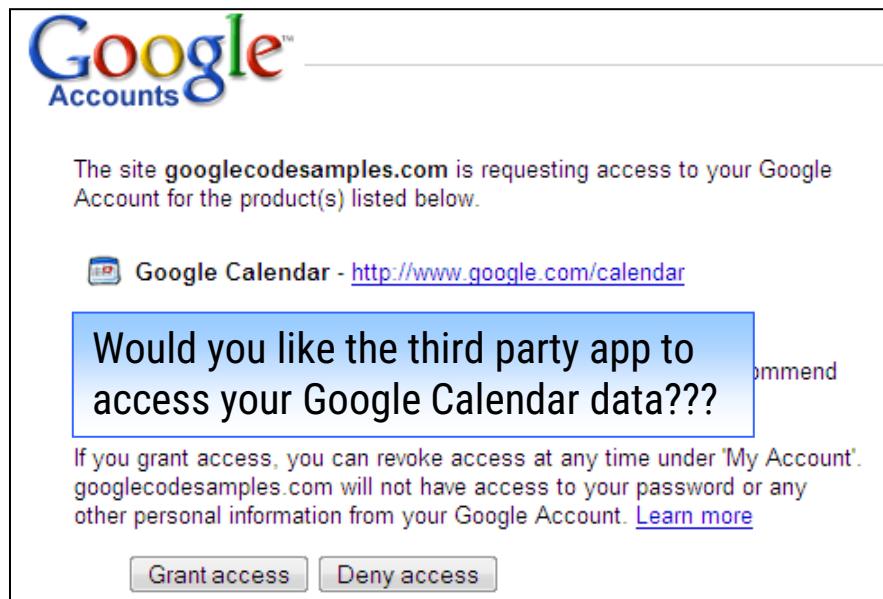


Your google calendar data is:

```

<pre><?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:openSearch="http://a9.com/-/spec/opensearchrss/1.0/">
 <id>http://www.google.com/calendar/feeds/default/public/basic</id>
 <updated>2009-06-29T16:44:00.229Z</updated>
 <category scheme="http://schemas.google.com/g/2005#list"></category>
 <title>Guo Zhenhua's Calendar List</title>
 <link rel="alternate" type="text/html" href="http://www.google.com/calendar/</link>
 <link rel="http://schemas.google.com/g/2005#feed" type="application/atom+xml" href="http://www.google.com/calendar/feeds/default/public/basic</link>
 <link rel="http://schemas.google.com/g/2005#post" type="application/atom+xml" href="http://www.google.com/calendar/feeds/default/public/basic</link>
 <link rel="self" type="application/atom+xml" href="http://www.google.com/calendar/feeds/default/public/basic</link>
 <author>
 <name>Guo Zhenhua</name>
 <email>jenvor@gmail.com</email>
 </author>
 <generator version="1.0" uri="http://www.google.com">Google Calendar</generator>
</feed>

```



# OAuth - Drawbacks

- Delegation granularity
- Error handling
- Token expiration vs revocation

# JWT

- JSON Web Tokens
  - jwt.io
  - Example

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6Ikpvag4gRG9
lIiwiZWFOIjoxNTE2MjM5MDIyfQ.
Sf1KxwRJSMeKKF2QT4fwpMeJf36P
0k6yJV_adQssw5c|
```

## HEADER:

```
{
 "alg": "HS256",
 "typ": "JWT"
}
```

## PAYOUT:

```
{
 "sub": "1234567890",
 "name": "John Doe",
 "iat": 1516239022
}
```

## VERIFY SIGNATURE

```
HMACSHA256(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload),
 your-256-bit-secret
) □ secret base64 encoded
```

# Remarks

- Auto log-off (session destroy) after x minutes idle
  - `session.gc_maxlifetime`
  - `session.cookie_lifetime`
- Beware cookie can be accessible from JS
  - `HttpOnly` (`session.cookie_httponly`)

# Web Service

IF3110 Web-Based Application Development  
School of Electrical Engineering and Informatics  
Institut Teknologi Bandung

# Objective

- Students understand the basic idea of web service and its basic principles
- Students understand various usages and approaches of Web Service
- Students know various techs to develop a Web Service
- Students are able to develop/consume a Web Service

# Motivation

- We need to use a service offered by others; and they are connected to Internet
- We intend to develop an application and it reuses/composes other running components
- We intend to access data that are provided by the other application

# Definition

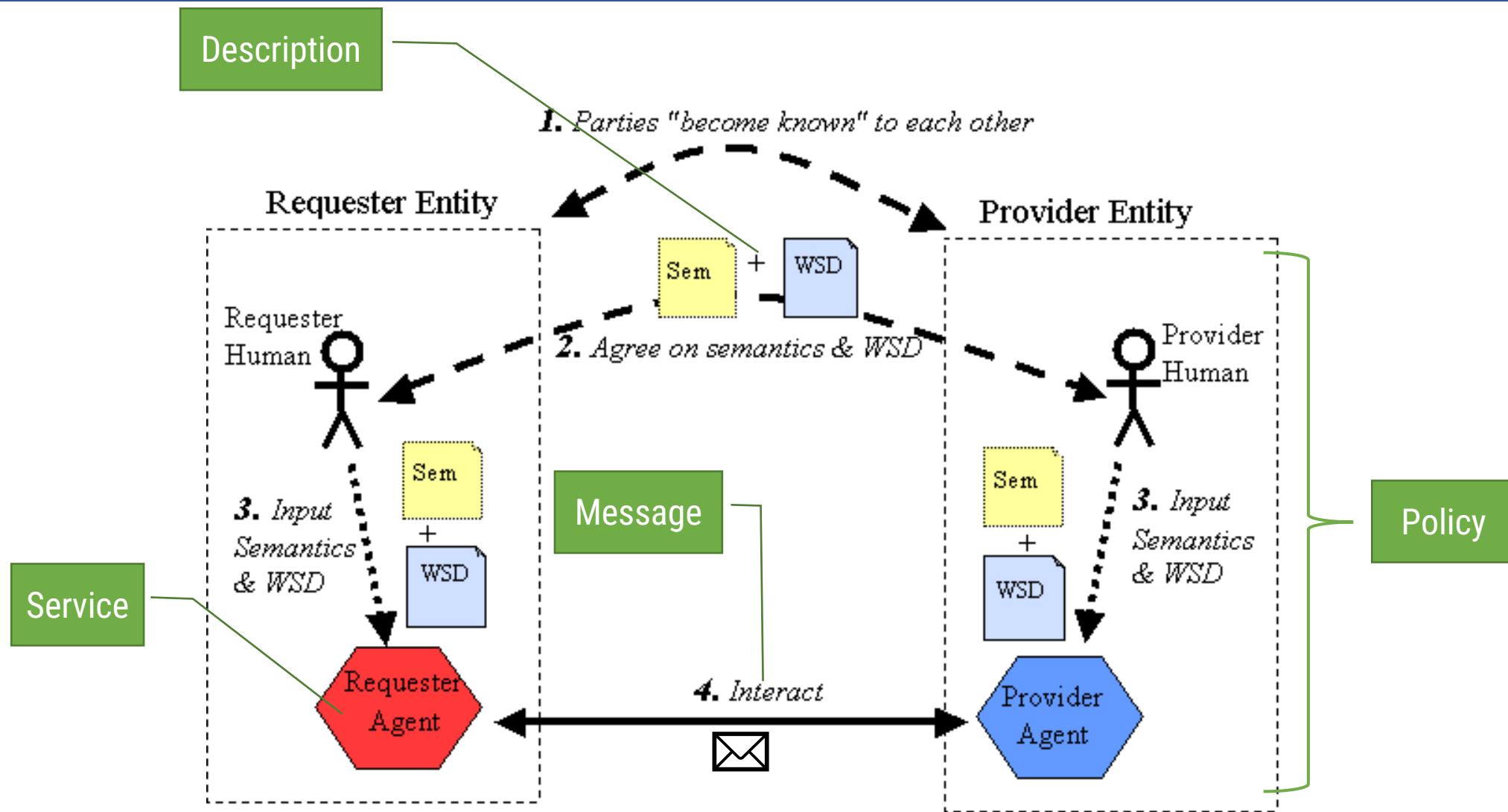
A web service is

- a software system designed
- to support interoperable machine-to-machine interaction over a network.
- It has an interface described in a machine-processable format (specifically [WSDL](#)).
- Other systems interact with the web service in a manner prescribed by its description using SOAP-messages, typically conveyed using [HTTP](#) with an [XML serialization](#) in conjunction with other web-related standards.
- – *W3C, Web Services Glossary*

# Characteristics

- Simple: application service delivered via Web protocol
- Web friendly: HTTP, XML based format, TCP/IP
- Contract-oriented:
  - Two approaches:
    - SOAP Based (WS-\*) Web Services
    - REST style web services
- Technology/language agnostic
- Discoverable

# Application Interaction with Web Service



# Rationale

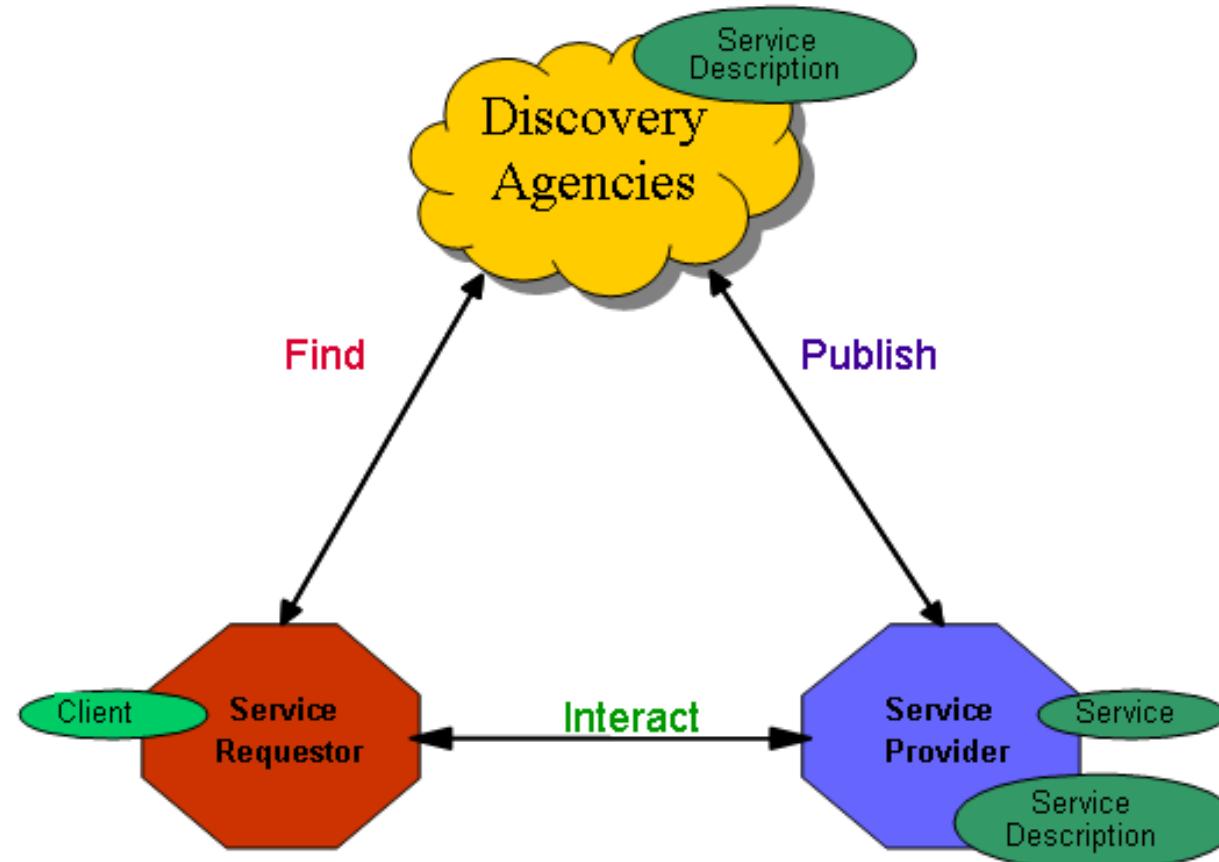
- Web protocol (HTTP, TCP/IP) is popular
- Text based
- loosely coupled
  - each component runs independently/autonomously

# Web services

web services is a service-oriented **middleware/technology**

- standards (WSDL, SOAP, WS-Security, WS-Policy, WS-\*)
- loose coupling:
  - is a service providing independent functionalities
  - service defined in WSDL without any dependency to the implementation
  - based on asynchronous messaging
- using existing internet technology (HTTP, SMTP)

# Service-Oriented Architecture



# Service-Oriented Middleware

Middleware: broker

- object-oriented: based on distributed object, synchronous access
- message-oriented: based on message exchange
- event-oriented: similar to message-oriented, but transient
  
- service-oriented middleware
  - based on message exchange, and has clear definition of the service (contract)

# Using Internet Technology

- service discovery: why not using DNS?
- interaction:
  - synchronous communication: HTTP
  - asynchronous communication: SMTP
- HTTP & SMTP is often non-blocked traffic by a firewall

# XML

```
<?xml version="1.1" encoding="UTF-8" ?>
<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
]>

<greeting>Hello, world!</greeting>
```

# Web Service Standard

- Most (if not all) Web Service standard defined in XML; thus refer to XML Schema definition
- Example
  - Info about offered goods → uses XML tag .... (XSD of SOAP)
  - Define how to use a web service via WSDL
- XML Schema Definition (.xsd)
  - Verify XML document – supported XML tags and the order of usage
  - Develop a program consuming a service without defining which is the service producer

# Web services

- Implementation means to realize services
- Based on open standards:
  - XML
  - SOAP: Simple Object Access Protocol
  - WSDL: Web Services Description Language
  - UDDI: Universal Description, Discovery and Integration
  - BPEL4WS: Business Process Execution Language for Web Services
- Main standardization
- bodies: OASIS, W3C

composition  
description  
messages  
network



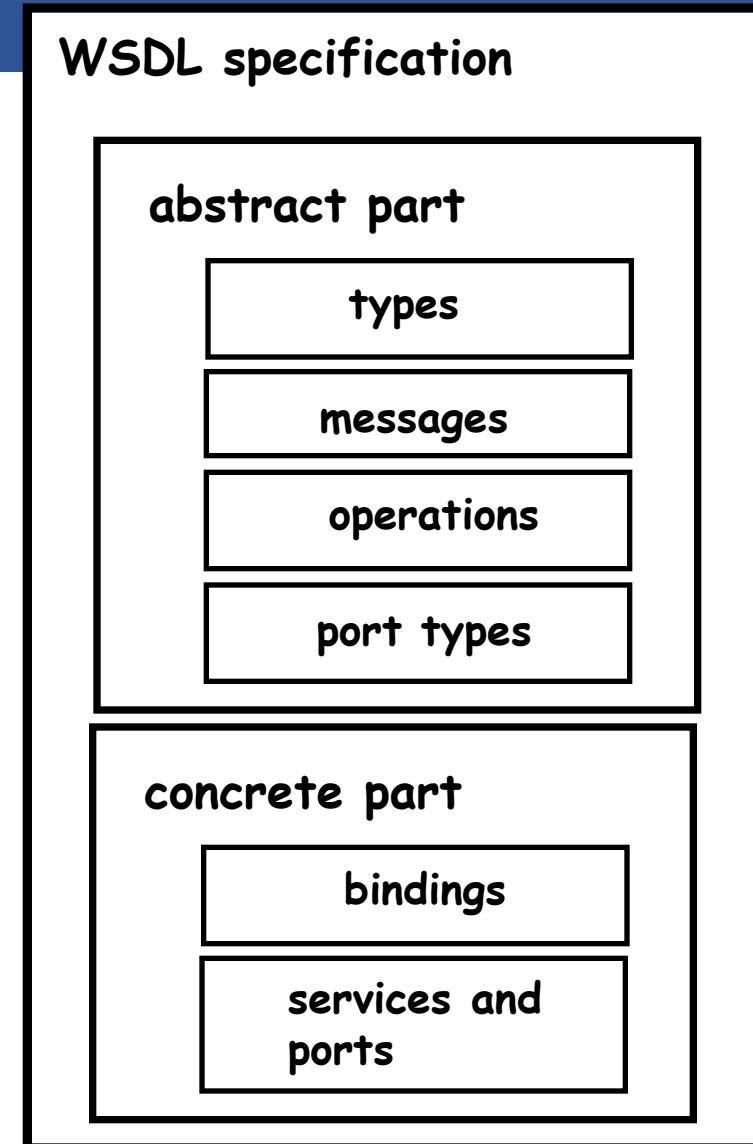
BPEL4WS		discovery
WSDL	UDDI	
SOAP		
HTTP, FTP, ...		

# WSDL specification

- WSDL: defined service provided by a service provider
- WSDL contains 6 elements

<http://schemas.xmlsoap.org/wsdl/>

- Note: WSDL uses XML, defined by an XML Schema



# WSDL example

```
<?xml version="1.0"?>
<definitions name="Procurement"
 targetNamespace="http://example.com/procurement/definitions"
 xmlns:tns="http://example.com/procurement/definitions"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/" >
```

```
<message name="OrderMsg">
 <part name="productName" type="xs:string"/>
 <part name="quantity" type="xs:integer"/>
</message>
```

```
<portType name="procurementPortType">
 <operation name="orderGoods">
 <input message = "OrderMsg"/>
 </operation>
</portType>
```

**abstract part**  
messages

operation and  
port type

```
<binding name="ProcurementSoapBinding" type="tns:procurementPortType">
 <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="orderGoods">
 <soap:operation soapAction="http://example.com/orderGoods"/>
 <input>
 <soap:body use="literal"/>
 </input>
 <output>
 <soap:body use="literal"/>
 </output>
 </operation>
</binding>
```

**concrete part**  
binding

```
<service name="ProcurementService">
 <port name="ProcurementPort" binding="tns:ProcurementSoapBinding">
 <soap:address location="http://example.com/procurement"/>
 </port>
</service>
```

port and  
service

```
</definitions>
```

# Communication patterns in WSDL <portType>

there are 4 classes of operations:

- one way (the client initiates this)
  - request-response (the client initiates this)
  - solicit-response (the service initiates this)
  - notification (the service initiates this)
- 
- in the previous example, a request-response type was used  
(input before output)

# WSDL <binding>

WSDL binding defines how SOAP engine does the service binding, e.g., to HTTP

```
<binding name="ProcurementSoapBinding" type="tns:procurementPortType">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="orderGoods">
 <soap:operation soapAction="http://example.com/orderGoods"/>
 <input>
 <soap:body use="literal"/>
 </input>
 <output>
 <soap:body use="literal"/>
 </output>
</operation>
</binding>
```

# SOAP (and SOAP engines)

SOAP is a standard to define:

- Message exchange form
  - a set of rules for SOAP engines to exchange SOAP messages
  - Convention to implement RPC
- 
- SOAP uses HTTP or SMTP to deliver the message
  - Defined in term of XML Schema

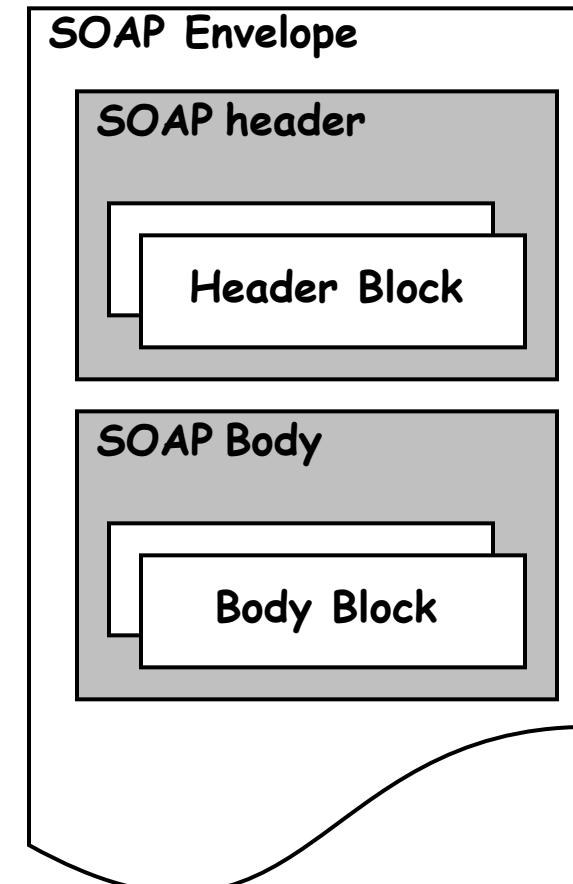
# SOAP messages

**SOAP envelop** wrap data to be sent:

- defined **encoding rules**
- contains **SOAP header**
  - optional
  - infrastructure-level data (e.g., security tokens, routing info, transaction IDs, ...)
- contains **SOAP body**
  - mandatory
  - application-level data (e.g., XML messages according to the WSDL of a service)

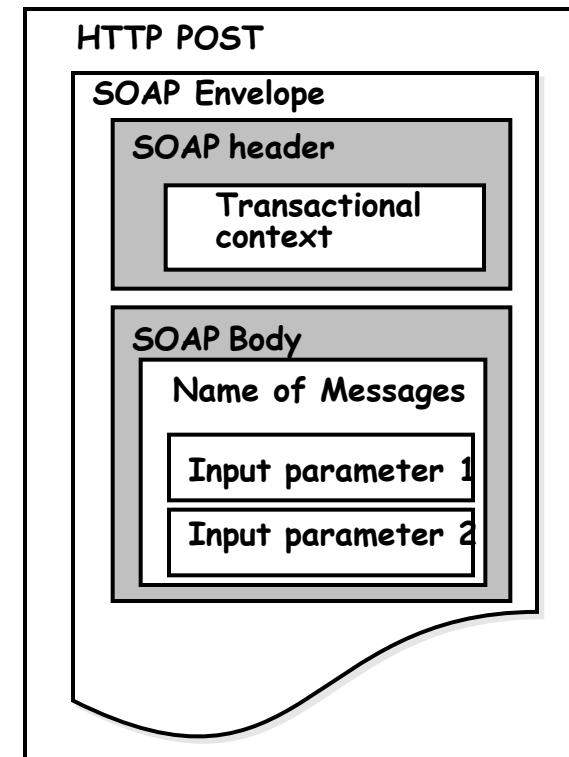
note: SOAP message is an XML document, refer to SOAP XML Schema

<http://schemas.xmlsoap.org/soap/>



# SOAP over HTTP

- SOAP standard defined how to send SOAP message via HTTP;
- SOAP (also SOAP engines) follows HTTP error codes
- HTTP POST
  - request using SOAP
  - response using SOAP, but in asynchronous messaging only containing the receipt acknowledgment HTTP 202-Accepted



# SOAP header and body: example

```
<?xml version='1.0' ?>
```



# Web Service Access in PHP

```
<?php

$wsdl_url =
 'http://services.xmethods.net/soap/urn:
 xmethods-delayed-quotes.wsdl';

$client = new SOAPClient($wsdl_url);
$quote = $client->getQuote('EBAY');

print $quote;
?>
```

# Web Service Provider in PHP

```
<?php
 class pc_SOAP_return_time {
 public function return_time() {
 return date('Ymd\THis');
 }
 }

$server = new SOAPServer(null,
 array('uri'=>'urn:pc_SOAP_return_time'));
$server->setClass('pc_SOAP_return_time');
$server->handle();
?>
```

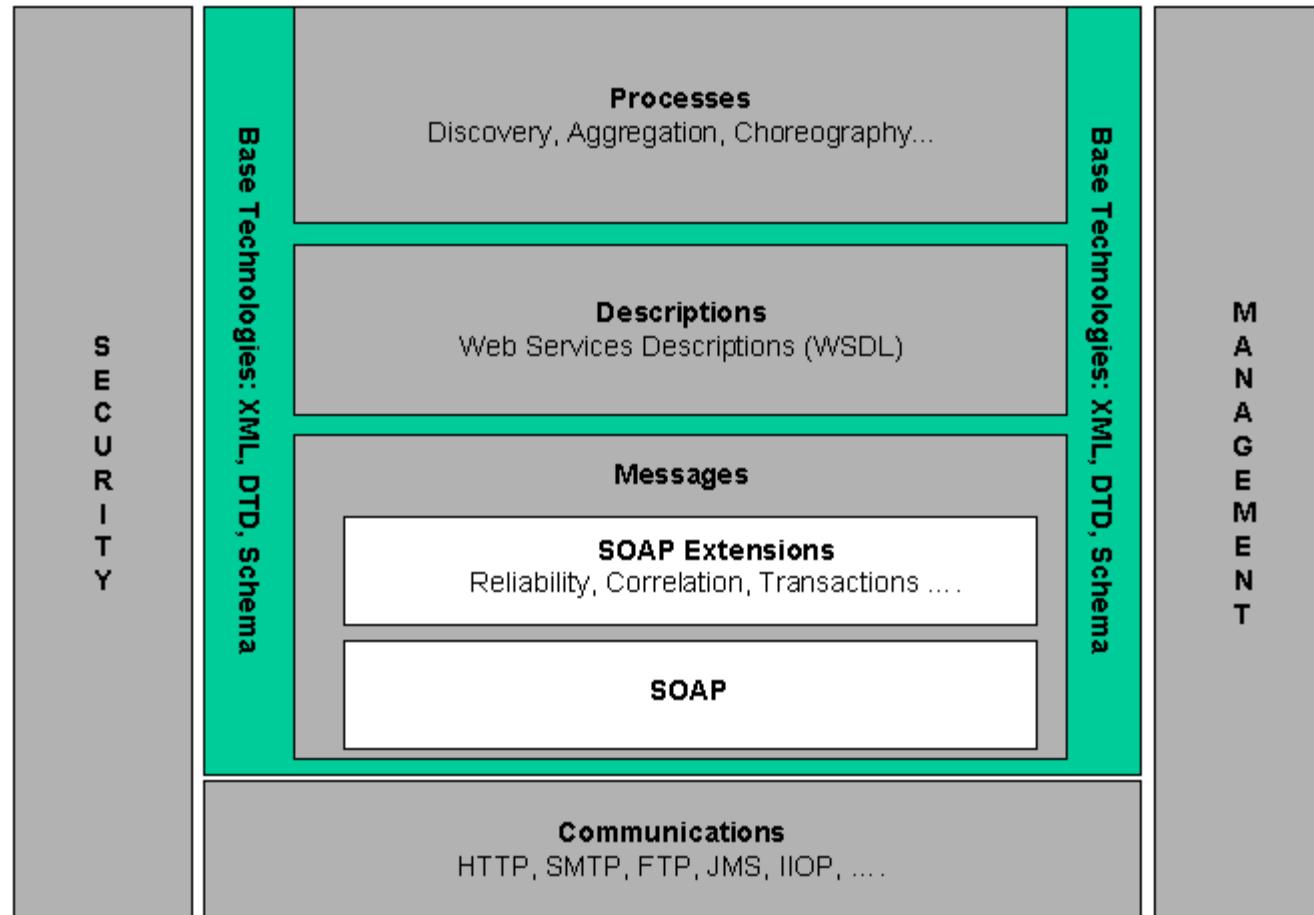
```
<?php
$opts = array('location' => 'http://localhost/contoh/soapserver.php',
 'uri' => 'http://pc_SOAP_return_time');

$client = new SOAPClient(null, $opts);

$result = $client->__soapCall('return_time', array());

print "The local time is $result.\n";
?>
```

# WS Architecture for SOA-system



# Discovery

- the act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria.
- It involves matching a set of functional and other criteria with a set of resource descriptions.

# WS Composition

## Orchestration

- Executable Process
- the arrangement of **execution sequence of independent agent** to support a business process
- defines the sequence of actions performed by a set of agents
- Language: WS-BPEL

## Choreography

- Multi-party collaboration
- the sequence and conditions under which multiple cooperating **independent agents exchange messages** in order to perform a task to achieve a goal state.
- defines the pattern of possible interactions between a set of agents.
- Language: WS-CDL

# Web service & distributed objects

- Distributed object systems problems:
  - Problems introduced by latency and unreliability of the underlying transport.
  - The lack of shared memory between the caller and object.
  - The numerous problems introduced by partial failure scenarios.
  - The challenges of concurrent access to remote resources.
  - The fragility of distributed systems if incompatible updates are introduced to any participant.

# Web service & distributed objects

- web service is suitable for:
  - platform/vendor neutrality is important; and overhead on performance is tradeable
  - service needs to operate via Internet where reliability and speed un-assured
  - loosely coupled – each component is developed independently
  - multiplatform & vendor
  - has an application that is accessible via network

# WS Standard

- W3 Consortium, OASIS,
- Orchestration: WS-Choreography, BPEL
- Transactions: WS-Transaction, RosettaNet
- Reliability: WS-ReliableMessaging, WS-Reliability
- Registries: ebXML, UDDI

# ReSTful webservice

# Web Service Alternatives

- REST: Representation State Transfer
  - Roy Fielding and his doctoral thesis on “Architectural Styles and the Design of Network-based Software Architectures”.
  - resource is accessible via URI
  - standard operation/method: create, retrieve, update and delete (similar to HTTP)
  - interaction: stateless, simple

# Why REST?

- What makes the Web scale?
- How can I apply the architecture of the Web to my own applications?
- Fielding (along with Tim Berners-Lee) designed HTTP and URI's.
- The question he tried to answer in his thesis was “Why is the web so viral”? What is its architecture? What are its principles?

Notes from “RESTful Java with JAX-RS” by Bill Burke.

# Understanding REST

- REST is not protocol specific. It is usually associated with HTTP but its principles are more general.
- SOAP and WS-\* use HTTP strictly as a transport protocol.
- But HTTP may be used as a rich application protocol.
- Browsers usually use only a small part of HTTP.
- HTTP is a synchronous request/response network protocol used for distributed, collaborative, document based systems.
- Various message formats may be used – XML, JSON,...
- Binary data may be included in the message body.

# REST vs SOAP

- So far we have:
  1. Created a SOAP based web service.
  2. Tested it and retrieved its WSDL.
  3. Generated code based on the WSDL.
  4. Called that code from a client.
- Let's look at the REST design philosophy...

# REST vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user
- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
 <pb:GetUserDetails>
 <pb:UserID>12345</pb:UserID>
 </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

# REST vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user
- And with REST? The query will probably look like this:  
<http://www.acme.com/phonebook/UserDetails/12345>
- GET /phonebook/UserDetails/12345 HTTP/1.1  
Host: [www.acme.com](http://www.acme.com)  
Accept: application/xml
- Complex query:  
<http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe>

# REST Style WS

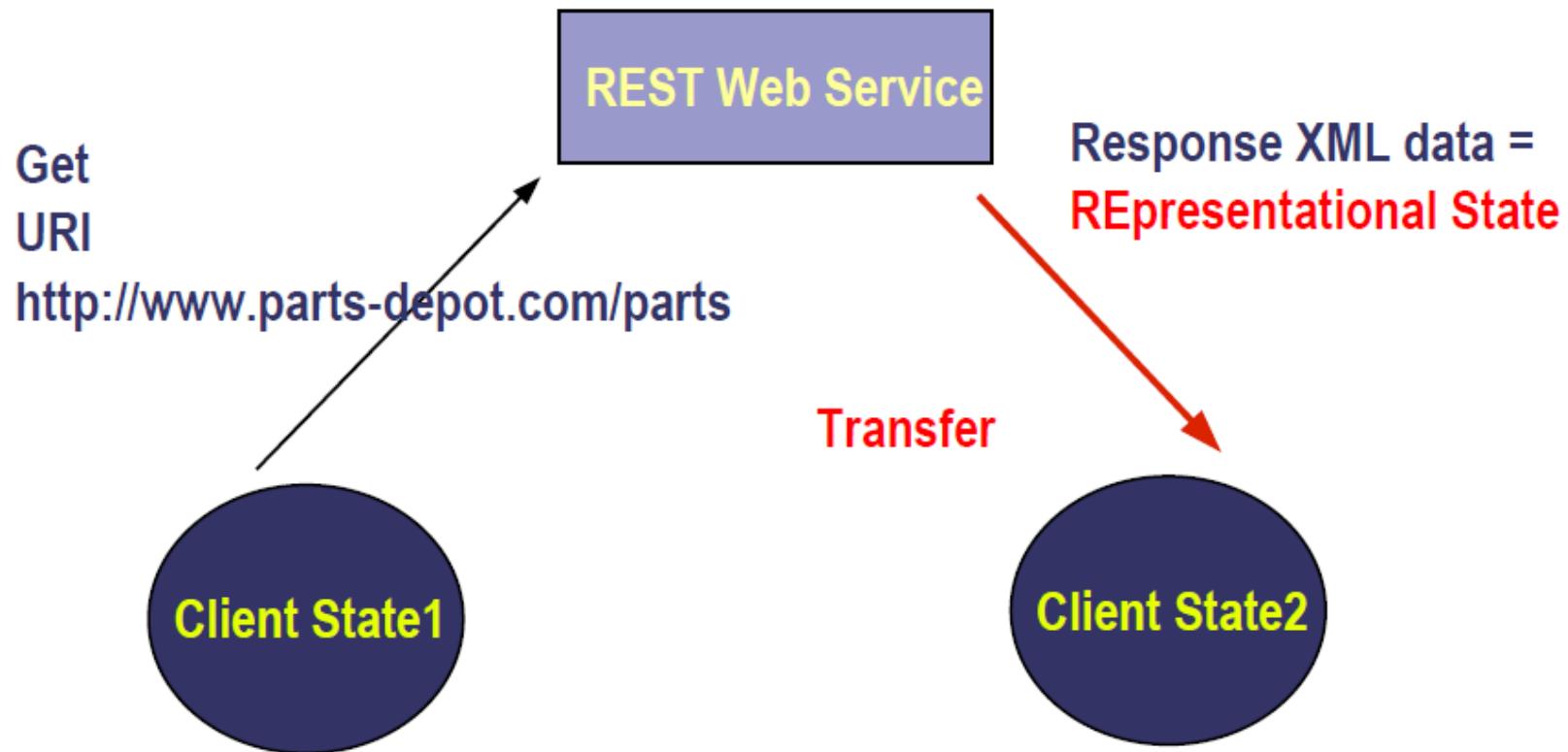
*POST /examples/stringer  
Host: www.cdk4.net*

Use a URI and an HTTP method  
to select what needs to be done.

*Drop the SOAP and use name value pairs in the request.  
Use XML or JSON for the response. Don't provide a new  
set of operations - use HTTP methods instead. Use the same  
set of principles that made the web go viral!*

# What is REST?

## REpresentationational State Transfer



# HTTP Request/Response As REST



# REST Architectural Principles

- The web has **addressable resources**.  
Each resource has a URI.
- The web has a **uniform and constrained interface**.
  - HTTP, for example, has a small number of methods. Use these to manipulate resources.
- The web is **representation oriented** – providing diverse formats.
  - Interaction with services using representations of that service.
  - Different platforms, different formats - browsers -> HTML, JavaScript -> JSON and a Java application -> XML?
- The web may be used to **communicate statelessly** – providing scalability
- **Hypermedia** is used as the **engine of application state**.
  - *Hypermedia As The Engine Of Application State (HATEOAS)*
  - Let your data formats drive state transitions in your applications.

# Principle: Addressability

- **Addressability** (not restricted to HTTP)

Each HTTP request uses a URI.

The format of a URI is well defined:

scheme://host:port/path?queryString#fragment

The **scheme** need not be HTTP. May be FTP or HTTPS.

The **host** field may be a DNS name or a IP address.

The **port** may be derived from the scheme. Using HTTP implies port 80.

The **path** is a set of text segments delimited by the “/”.

The **queryString** is a list of parameters represented as

name=value pairs. Each pair is delimited by an “&”.

The **fragment** is used to point to a particular place in a document

# Principle: Uniform Interface (1)

- A uniform constrained interface:
    - No action parameter in the URI
    - HTTP
      - GET - read only operation
        - idempotent (once same as many)
        - safe (no important change to server's state)
        - may include parameters in the URI
- <http://www.example.com/products?pid=123>

# Principle: Uniform Interface (2)

## HTTP

PUT - store the message body

- insert or update
- idempotent
- not safe

# Principle: Uniform Interface (3)

HTTP

POST

- Not idempotent
- Not safe
- Each method call may modify the resource in a unique way
- The request may or may not contain additional information
- The response may or may not contain additional information
  - The parameters are found within the request body (not within the URI)

# Principle: Uniform Interface (4)

HTTP

DELETE - remove the resource

- idempotent
- Not safe
- Each method call may modify the resource in a unique way
- The request may or may not contain additional information
- The response may or may not contain additional information

HTTP HEAD, OPTIONS, TRACE and CONNECT are less important.

# Principle: Uniform Interface (5)

- Does HTTP have too few operations?
- Note that SQL has only four operations: SELECT, INSERT, UPDATE and DELETE
- 
- JMS and MOM have, essentially, two operations: SEND and RECEIVE
- SQL and JMS have been very useful.

# REST over HTTP – Uniform interface

- CRUD operations on resources
  - Create, Read, Update, Delete
- Performed through HTTP methods + URI

CRUD Operations	4 main HTTP methods	
	Verb	Noun
Create (Single)	POST	Collection URI
Read (Multiple)	GET	Collection URI
Read (Single)	GET	Entry URI
Update (Single)	PUT	Entry URI
Delete (Single)	DELETE	Entry URI

# Why a uniform interface?

## Familiarity

We do not need a general IDL that describes a variety of method signatures.

We already know the methods.

## Interoperability

WS-\* has been a moving target.

HTTP is widely supported.

## Scalability

Since GET is idempotent and safe, results may be cached by clients or proxy servers.

Since PUT and DELETE are both idempotent neither the client or the server need worry about handling duplicate message delivery.

# Principle: Representation Oriented(1)

- Representations of resources are exchanged.
- GET returns a representation.
- PUT and POST passes representations to the server so that underlying resources may change.
- Representations may be in many formats: XML, JSON, YAML, etc., ...

# Principle: Representation Oriented(2)

- HTTP uses the CONTENT-TYPE header to specify the message format the server is sending.
- The value of the CONTENT-TYPE is a MIME typed string. Versioning information may be included.
- Examples:
  - text/plain
  - text/html
  - application/json
  - application/vnd+xml;version=1.1
- “vnd” implies a vendor specific MIME type

# Principle: Representation Oriented(3)

- The ACCEPT header in content negotiation.
- An AJAX request might include a request for JSON.
- A Java request might include a request for XML.
- Ruby might ask for YAML.

# Principle: Communicate Statelessly

- The application may have state but there is no client session data stored on the server.
- If there is any session-specific data it should be held and maintained by the client and transferred to the server with each request as needed.
- The server is easier to scale. No replication of session data concerns.

# Principle: HATEOAS

- **Hypermedia As The Engine Of Application State**
- Hypermedia is document centric but with the additional feature of links.
- With each request returned from a server it tells you what interactions you can do next as well as where you can go to transition the state of your application.

```
<order id = "111">
 <order-uri>http://.../order/111
 <customer>Alice
 <customer-uri>http://.../customers/3214
 <order-entries>
 <order-entry>
 <qty>5
 <product>Noodle
 <product-uri>http://.../products/111
```

```
<order id = "111">
 <customer>Alice 
 <order-entries>
 <order-entry>
 <qty>5
 <product>Noodle
```

# OpenAPI Specification (OAS)

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including:

- Available endpoints (e.g. /users) and operations on each endpoint ( e.g. GET /users, POST /users)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use, and other information.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines.

<https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.1.0.md>

# Basic Structure: Metadata

## Metadata

- Every API definition must include the version of the OAS

```
openapi: 3.0.0
```

## Info

- The info section contains API information: title, description (optional), version, contact information, license, terms of service, and other details.

```
info:
 title: Sample API
 description: Optional multiline or single-line description in
 [CommonMark](http://commonmark.org/help/) or HTML.
 version: 0.1.9
```

# Basic Structure: Servers

## Servers

- The servers section specifies the API server and base URL.

```
servers:
 - url: http://api.example.com/v1
 description: Optional server description, e.g. Main (production) server
 - url: http://staging-api.example.com
 description: Optional server description, e.g. Internal staging server
 for testing
```

# Basic Structure: Paths

## Paths

- The paths section defines individual endpoints (paths) in your API, and the HTTP methods (operations) supported by these endpoints. For example, GET /users can be described as:

```
paths:
 /users:
 get:
 summary: Returns a list of users.
 description: Optional extended description in CommonMark or HTML
 responses:
 "200":
 description: A JSON array of user names
 content:
 application/json:
 schema:
 type: array
 items:
 type: string
```

# OAS Complete Guide

- API Server and Base Path
- Media Types
- Paths and Operations
- Describing Parameters
- Parameter Serialization
- Describing Request Body
- Describing Responses
- Data Models
- Adding Examples
- Authentication
- Links
- Callbacks
- Components Section
- Using \$ref
- API General Info
- Grouping Operations With Tags
- OpenAPI Extensions



**OPENAPI**  
**INITIATIVE**

Read here:

[https://swagger.io/docs/specification/v3\\_0/about/](https://swagger.io/docs/specification/v3_0/about/)

# Misc

- Service vs API vs Endpoint vs Back End
  - What are their similarities?
  - What are their particularities?