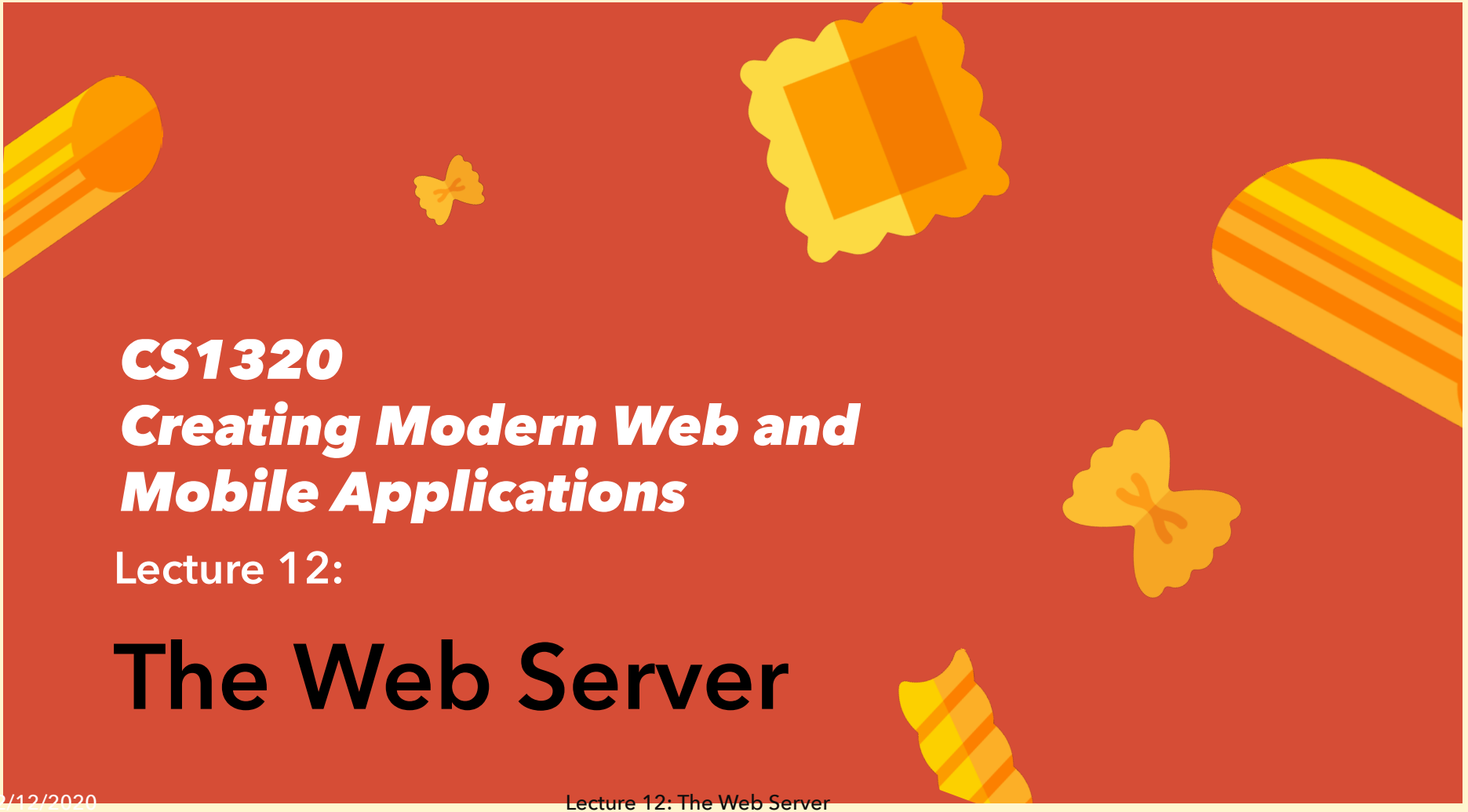


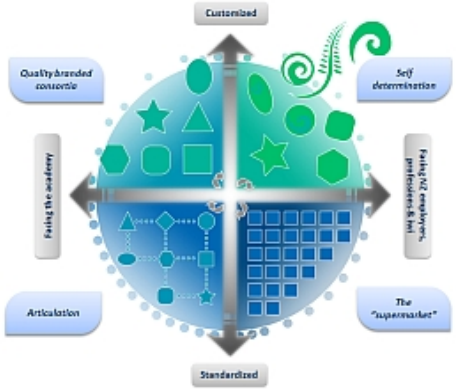
CS1320
Creating Modern Web and Mobile Applications

Lecture 12:

The Web Server

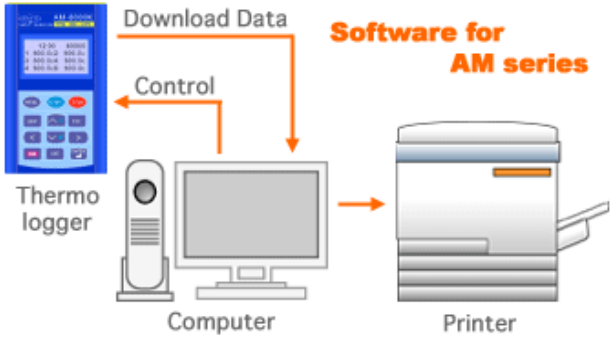
Specifications

- Describe what will be done
  - **Scenarios**
  - Lists of features to implement
  - Note optional versus required (priority)
- Define the user experience
  - Sketches of web pages (not final design)
  - Basically list what should be there as a basis for the design
- Identify any interfaces to existing systems
  - Servers, databases, login, etc.
- Outline of web site and pages
  - List of what pages are needed



Specifications

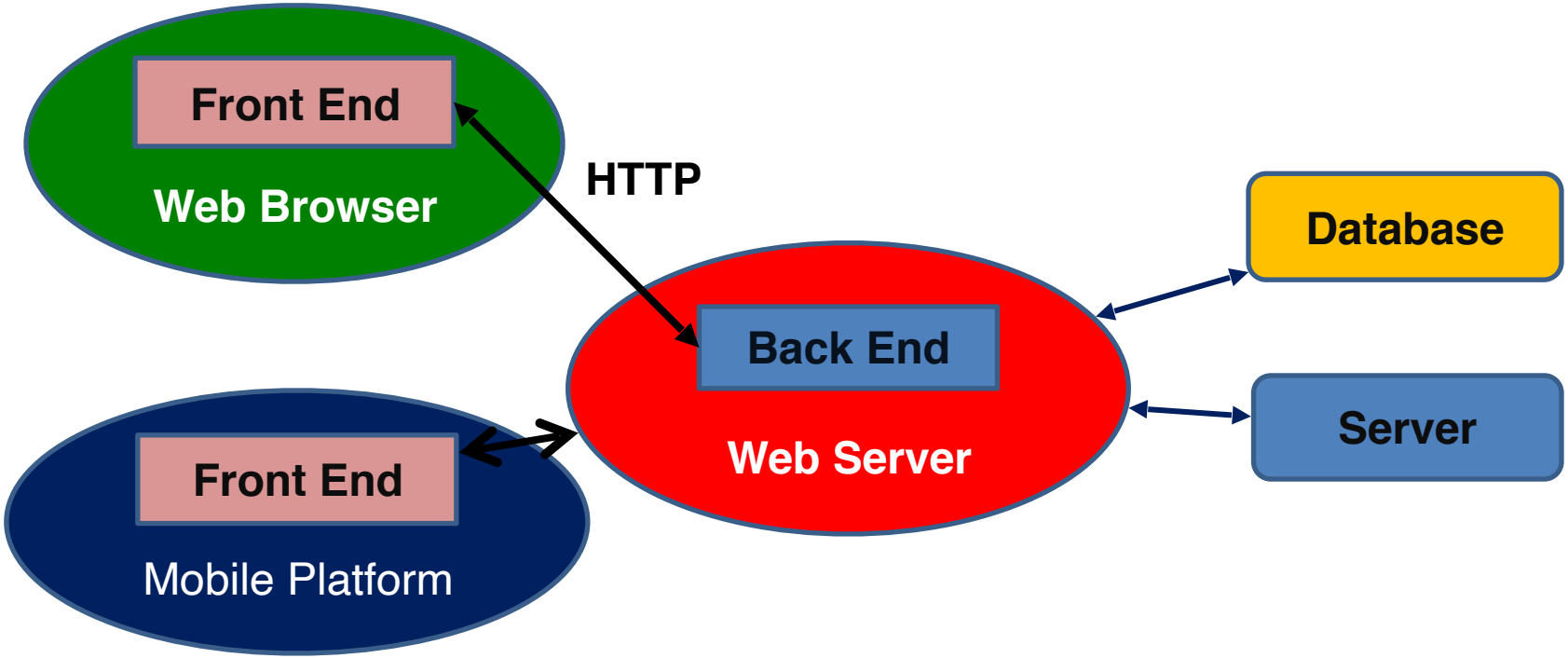
- Detail what the application will do
  - From the programmer's point of view
  - Can talk about other systems, components, modules
  - More likely to talk about commands, inputs, outputs
  - **WHAT** not **HOW**
- Define the inputs and outputs
  - What information is needed
  - What information is used
  - Where does this information come from
  - Where does this information go
- Specifications Document due 2/24



Specifications Are Not Designs

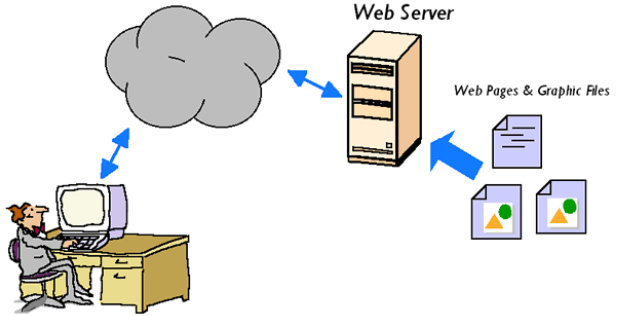
- WHAT not HOW
- Do not identify particular technologies to u
  - Unless mandated by outside requirements
  - Back ends, front ends, databases, …
- Do not determine how or where tasks are d
  - Front end, back end, database
  - Particular algorithms or processing (unless part of the requirements)
- Do not provide detailed web site designs
- Specifications will change as requirements change

Web Applications



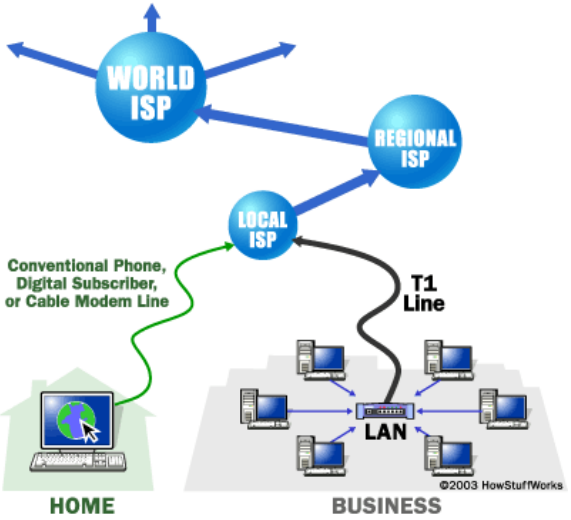
The Web Server

- Sits on the host machine
  - Listens for connections on a particular port (i.e. 80)
  - Gets HTTP requests sent to that port (via a socket connection)
  - Processes each request independently
    - URL tells it how to process a request
    - Sends a response back on the same socket
- Basic requests
  - URL with a file specified
  - Find the file on disk and return it
    - Create an appropriate HTTP response (header)
    - Followed by the data in the file



Web Server Game

- Volunteers (4) to act as clients making requests
  - Can request a page of a given color
    - ORANGE, YELLOW, PURPLE, BLUE
    - RED (pink), GREEN, TAN, GRAY
- Volunteers (4) to act as HTTP connect
  - Interface between clients and server
- Volunteer (1) to act as the web server
  - Pages reside on file system

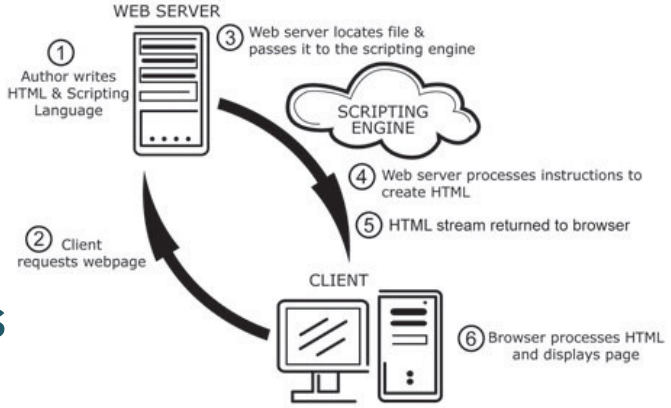


Web Server Game Improvements

- How can we speed this up?

Dynamic Requests

- Static requests are static
  - Don't work for web applications
  - We need to get different data under different circumstances
    - Based on information passed in with the URL
- Recall URLs have a query portion
  - With name-value pairs (or POST data)
  - Set up by HTML forms
  - Can involve interaction with JavaScript
- Web server needs to return different res
  - Based on the query / data



Modified Web Server Game

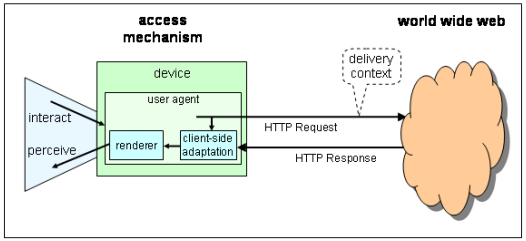
- Client asks for a color and a positive integer <= 100
  - Web server has to return a sheet giving the square of the number
  - Or ERROR (40x) if the input is illegal

Web Server Game Improvements

- How might we speed this up?

Context-Based Requests

- Most dynamic requests have a context
  - Shopping cart
  - Previous searches
  - Previous inputs and pages
  - User id
- The web server needs know the context
  - Map users to contexts
  - Use the context in creating the resultant output



Modified Web Server Game

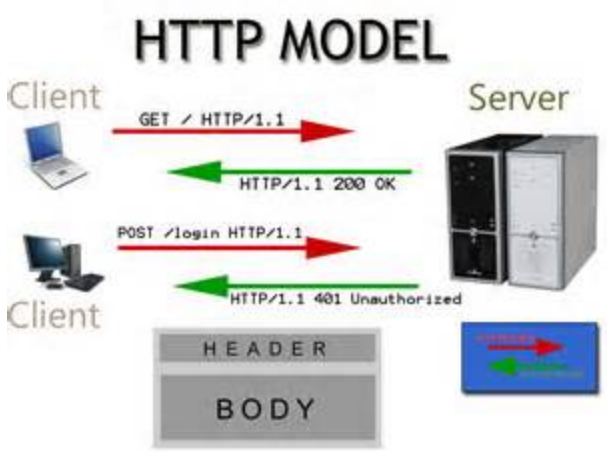
- Client asks for a color and provides positive integer <= 100
  - Server provides the sum of their previous numbers plus the new one
- Server provides the client with an ID
  - Same ID for same client
  - Client has to return the ID as part of their request

Modified Web Server Game

- How might we speed this up?

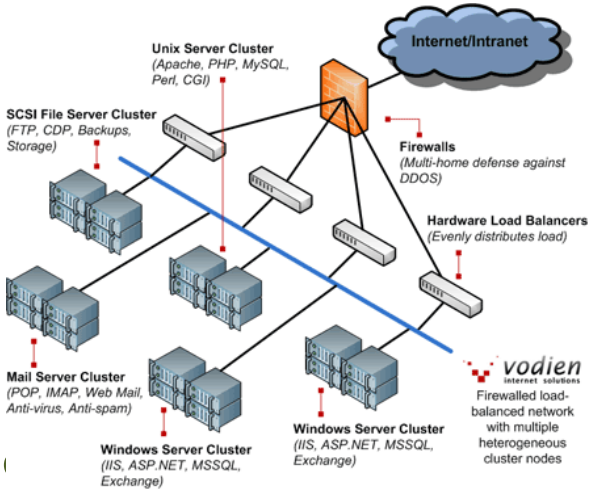
What the Web Server Does

- Given a HTTP Request
  - Return a HTTP Response
- Given a URL
  - Return the corresponding page
- Given a URL plus parameters / data
  - Compute and return the resultant data
  - Compute and return a HTML page



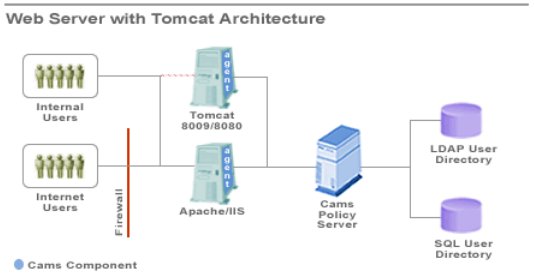
Web Server Issues

- Handling large numbers of clients
  - Multiple threads, caching, multiple servers
- Managing context or state
- Generating HTML output containing compute
- Doing the actual computations
  - We need to describe these
  - We need a program (and hence a programming language)
- Where are the computations done
  - By the web server
  - Externally



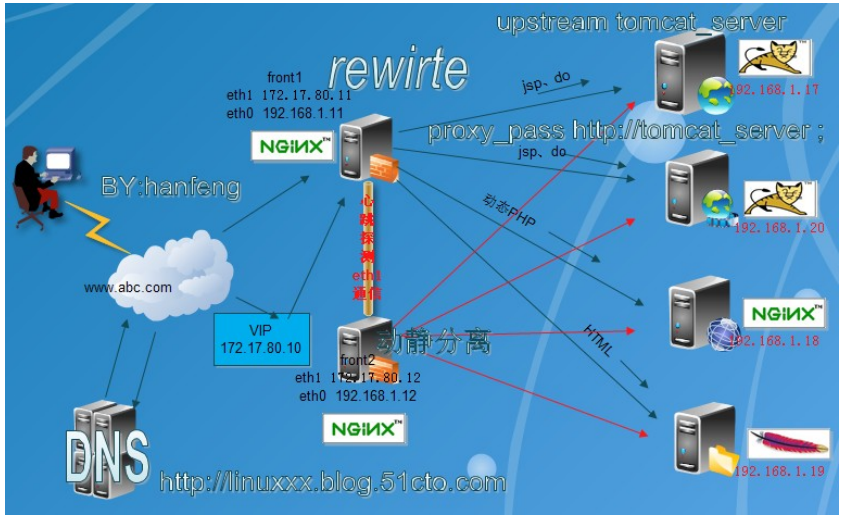
Web Servers

- General purpose servers
  - Handle static pages; designed to scale
  - Examples: Apache, NginX, Microsoft IIS
- Extensions to handle Computation
  - Modules: PHP, Ruby, Python, Perl, FCGI, C#
  - External Calls: CGI
- Special purpose servers
  - TOMCAT: Java servlets
  - NODE.JS: Event-based JavaScript
  - Django, Flask: Python; Ruby on Rails: Ruby
- Embedded Servers
  - Nanohttpd.java; micro-httpd for arduino

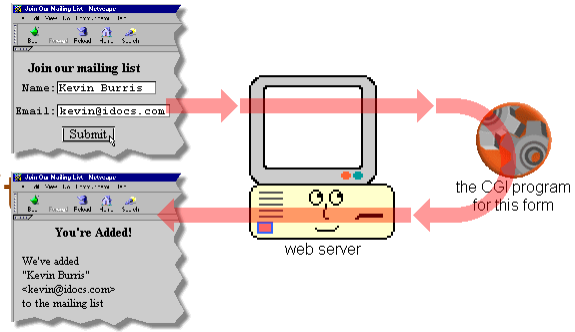


Server Organization

- Server needs to handle multiple requests at once
  - Several alternative designs are possible for this
- Use threads
- Use multiple servers
- Use asynchronous I/O
- Combinations of these



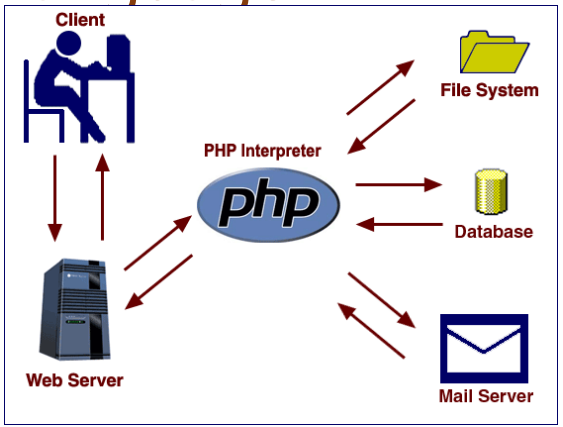
CGI Programs



- First way that servers provided programmabili
- URL: http://host/cgi-bin/cmd?args
  - cgi-bin is a special directory on the web server
  - cmd is the name of a normal executable in that directory
    - Shell script, perl, php, python, java jar file, c/c++ binary, …
  - args are named arguments passed to command
- The program 'cmd' is run on the web server
  - Any program output is passed back to client
  - Typical Use: Format a request and pass it on to server
- Problems: efficiency, security, safety
  - Used in very limited applications

PHP

- PHP is a simple string-oriented scripting language
  - Similar capabilities as Python, JavaScript
  - Designed to make string processing easy
- Web server runs PHP internally
  - As a module or plug-in
  - Automatically when a page has a .php extension

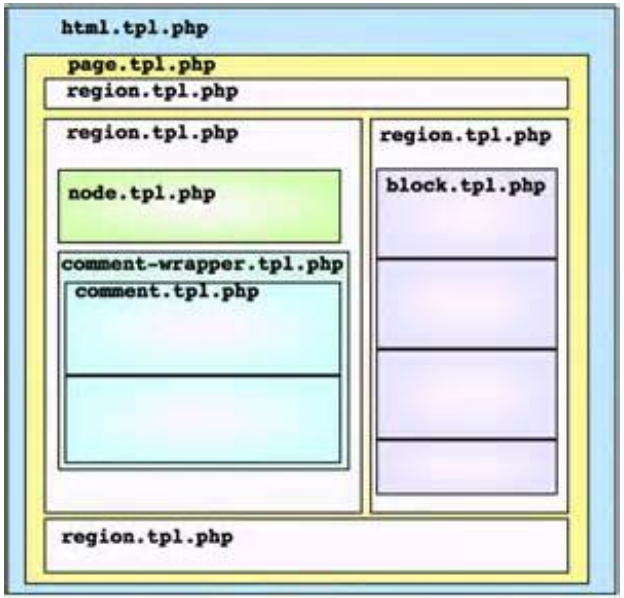


PHP and HTML

- What does the web server normally generate
  - HTML pages
  - With lots of HTML (text)
- What's different is based on query part of URL
  - Some fraction of the page
- Most of the output is fixed text
  - Header, navigation, footer
  - Parts of the contents
- Why should we write code to output this
  - In any language

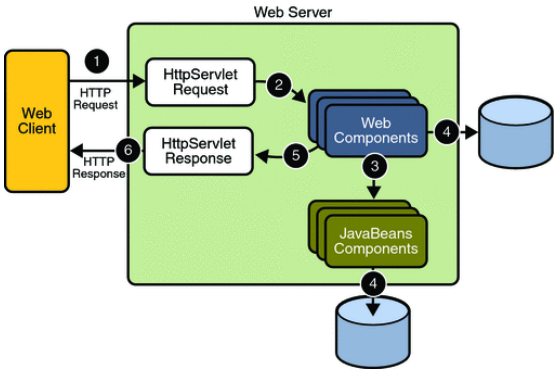
PHP Pages

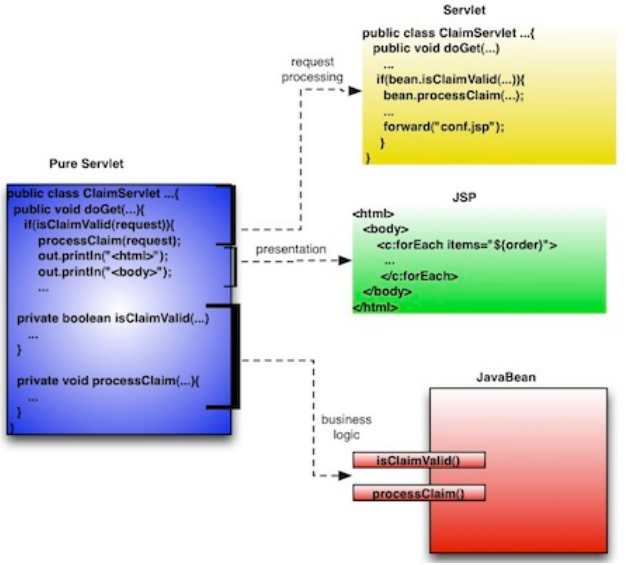
- Normal URLS where the file has a .php extension
  - The plug in doesn't run PHP directly on the file
  - The page is actually a mixture of text and code
- HTML pages with embedded PHP code
  - PHP module reads the page
  - The HTML portion is passed on directly
  - The PHP code is embedded in <?php … ?> constructs
    - <? … ?>
  - Where the code appears, it is run & replaced by its output
    - PHP print or echo statements
- This concept, *templating*, is very useful
  - Used to some extent in React, angular, vue, …



Servlets and JSP

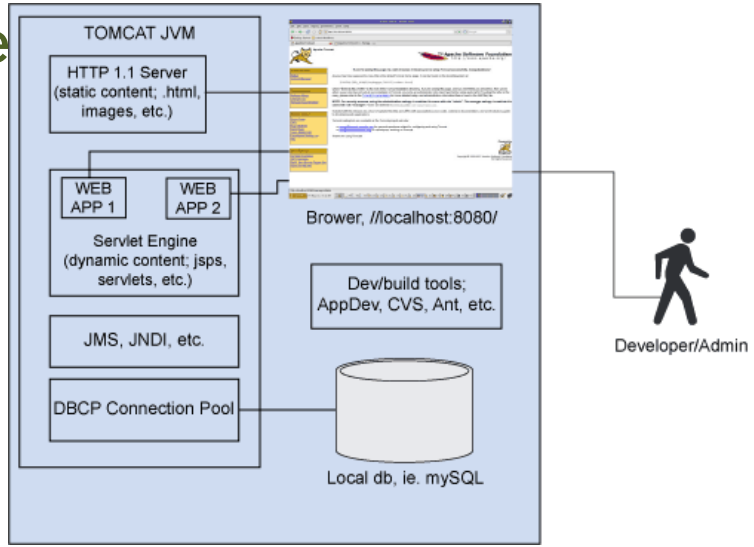
- Why add a new language
  - Programmers know Java
  - Back end applications are often written in Java
- Use Java as the processing language
  - Not ideal for string processing, but acceptable
  - Multiple threads already accommodated
- Servlet
  - Standard interface invoked directly by URL
    - Path name = class name, parameters accessible
- Java Server Pages
  - Pages with embedded Java <? … ?>





Java Servlets and JSP

- Handled by a separate web serve
  - TOMCAT is the most common
  - Runs on a different port
  - URL: host:8080/servlet/class?parms
- JSP handled by file extension
  - URL: host:8080/page.jsp



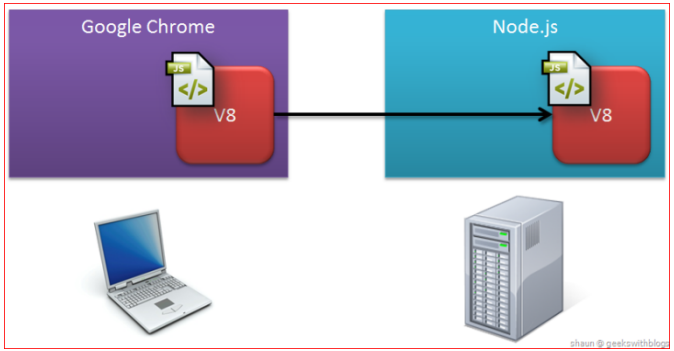
ASP.Net

- Supported by Microsoft IIS
- Use C# (or C++) to write the back e
- Web pages use templating
  - With embedded C#

Node.JS

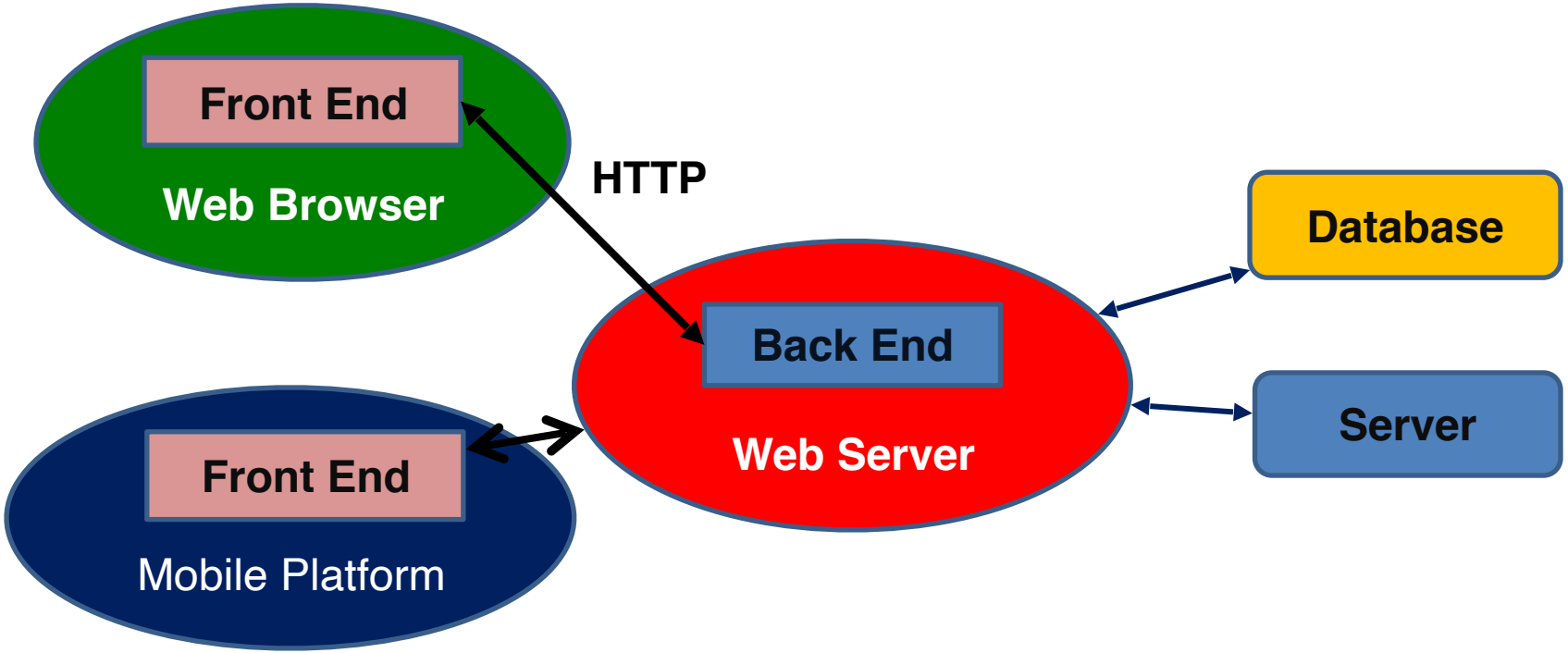
- Why learn a new language (PHP)
  - We already know JavaScript
  - PHP is too slow; JavaScript is now compiled and fast
  - It has most of what is needed
- What's wrong with Java (C#)
  - Too complex, not string-oriented
  - Too much baggage
- Straight line code is inefficient
  - Querying database, servers, file system all take time
  - Multiple threads complicate processing
  - Difficult to load balance with diverse threads

Node.JS



- JavaScript Web Server
  - Separate server (like TOMCAT for Java)
    - Each application has its own server
  - App back end is written in JavaScript
- Event-Based
  - Computation is done in small pieces
    - Complex interactions are done asynchronously
  - JavaScript code is associated with events
    - The code is executed when the event occurs
    - Code can initiate asynchronous computations with later events
    - Code supplies a continuation invoked when action completes

Web Applications



Databases

- Most web applications need to store information
  - Much of what they do is information based
  - Shopping site as an example
  - The security, integrity, … of the information is important
- The server code talks to a database system
  - All languages have code to make this relatively easy
- Database operations
  - Setting up the database
  - Adding and removing information from the database
  - Getting (querying) information from the database

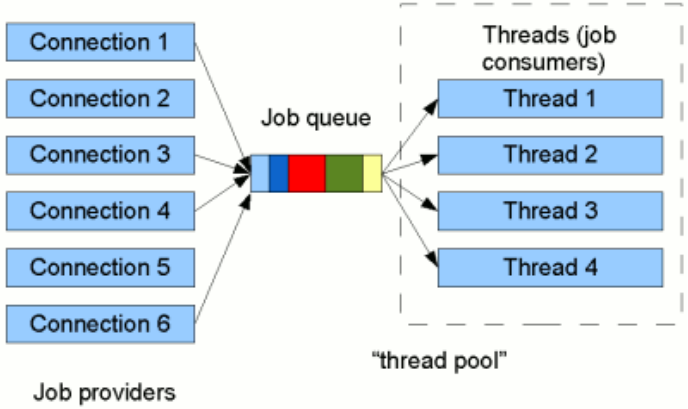
Frameworks

- All this sounds complex to set up and operate
  - A lot of the work is common and straightforward
    - Communications, setting up pages, database access, …
  - It can be simplified by extracting these
    - Leaving only the code specific to the particular application
- Frameworks are attempts to do this
  - Provide common code to plug in the application
  - Provide all the glue code; simplify database access
  - Ruby on Rails, Django, Flask, GWT
  - Express (and other plug-ins) for Node.JS

Next Time

- Node.JS
- Homework:
  - Pre-Lab 4

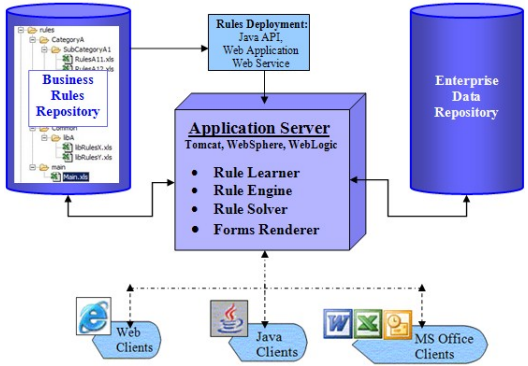
Server Organization



- Internal processing
  - Queue of tasks to be done
  - Thread pool to handle multiple requests
  - Internal requests can be queued if necessary
- Handling initial requests
  - Single thread to read web socket
- Multithreaded versus Single threaded processing
  - Using non-blocking (asynchronous) I/O

Handling Complex Applications

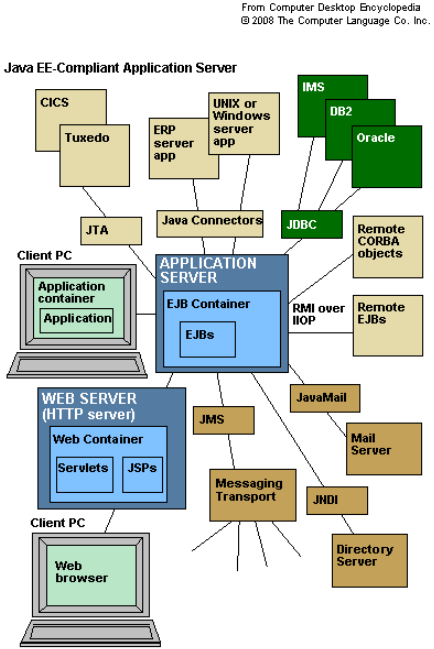
- The web server
  - Can handle PHP, Servlets, etc.
  - But these have limited capabilities
  - These run in limited environments
  - Don't want to overwhelm the server
    - The server has other responsibilities



- What if your application is more complex
  - You need to provide complex services (e.g. machine learning, data mining, search)
  - Then you might want to have your own server

User Server Organization

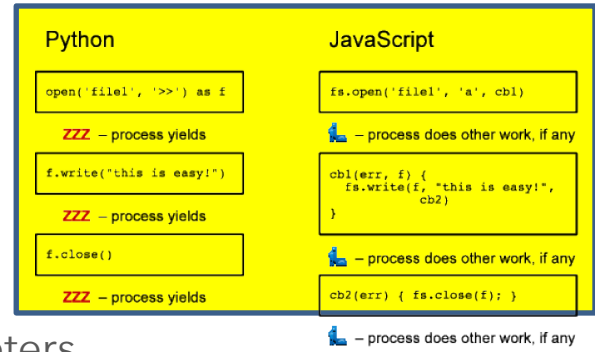
- Based on a client-server model
- Client: app code in the web server
  - Each request is its own client
  - Can be done via PHP or other server side code
- Socket-based communication
  - Server runs on a host and accepts connections on a port
  - Client connects to that host-port
    - Sends command/request
    - Reads response, processes it to HTML/JSON
    - Returns it to the browser
- Server: self-standing system



PHP Language

- Simple interpreted (scripting) language
- Untyped
  - Basic data types: string, int (long), float (double)
  - Complex data types: associative arrays, classes
- Lots of built-in functions
- Good string support
  - "hello $var, this is a ${expr}. "
- Good documentation (esp. for libraries)

Node.JS Event Example



- Request comes in
  - JavaScript code creates database query based on parameters
  - Starts query and registers continuation
- When query completes (done asynchronously)
  - Continuation is invoked. Template file is opened and a new continuation is provided
- When file is ready to read (done asynchronously))
  - A stream from the file to the client is established
  - The file is output asynchronously
- We'll get into this in detail next week

What Services Did You Guess

- What does a back end have to do for a web application?
  - Storage (database)
  - Accounts (login, authentication)
  - Computation (search, processing)
  - Security (transactions, secure processing)