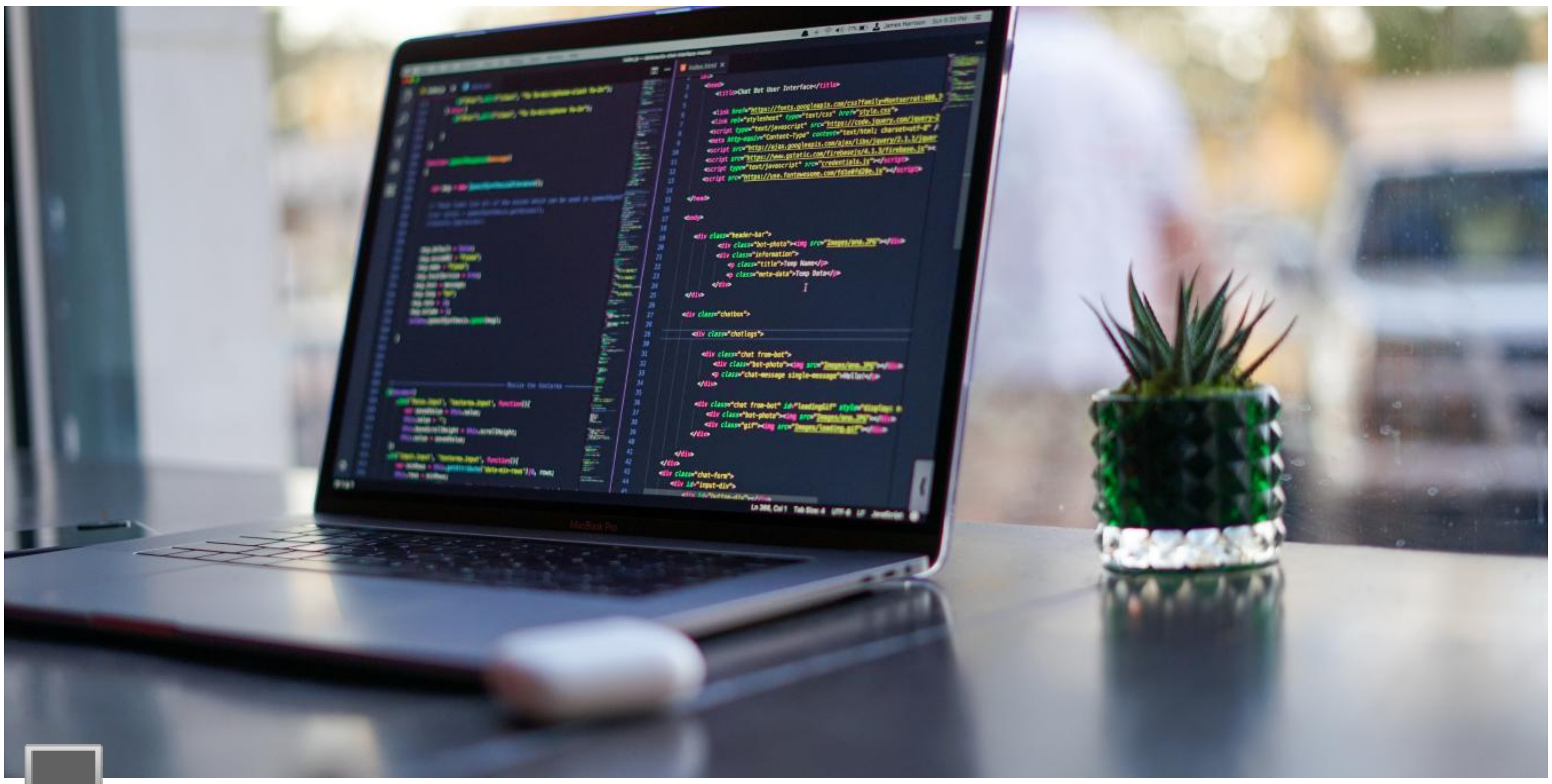


💻

Week 1 Lecture 1

Class	BSCCS2003
Created	@September 4, 2021 7:51 PM
Materials	
Module #	1
Type	Lecture
Week #	1

What is an App?

"An app is a computer software, or a program, most commonly a small, specific one used for mobile devices. The term app originally referred to any mobile or desktop application, but as more app stores have emerged to sell mobile apps to smartphone and tablet users, the term has evolved to refer to small programs that can be downloaded and installed at once"

- Source: Techopedia

Desktop Apps

- Usually standalone
    - Editors / Word processors
    - Web browsers
    - Mail clients
- Often work offline
    - Local data storage
    - Possible network connection
- Software Development Kits (SDK)
    - Custom frameworks
    - OS specific

Mobile Apps

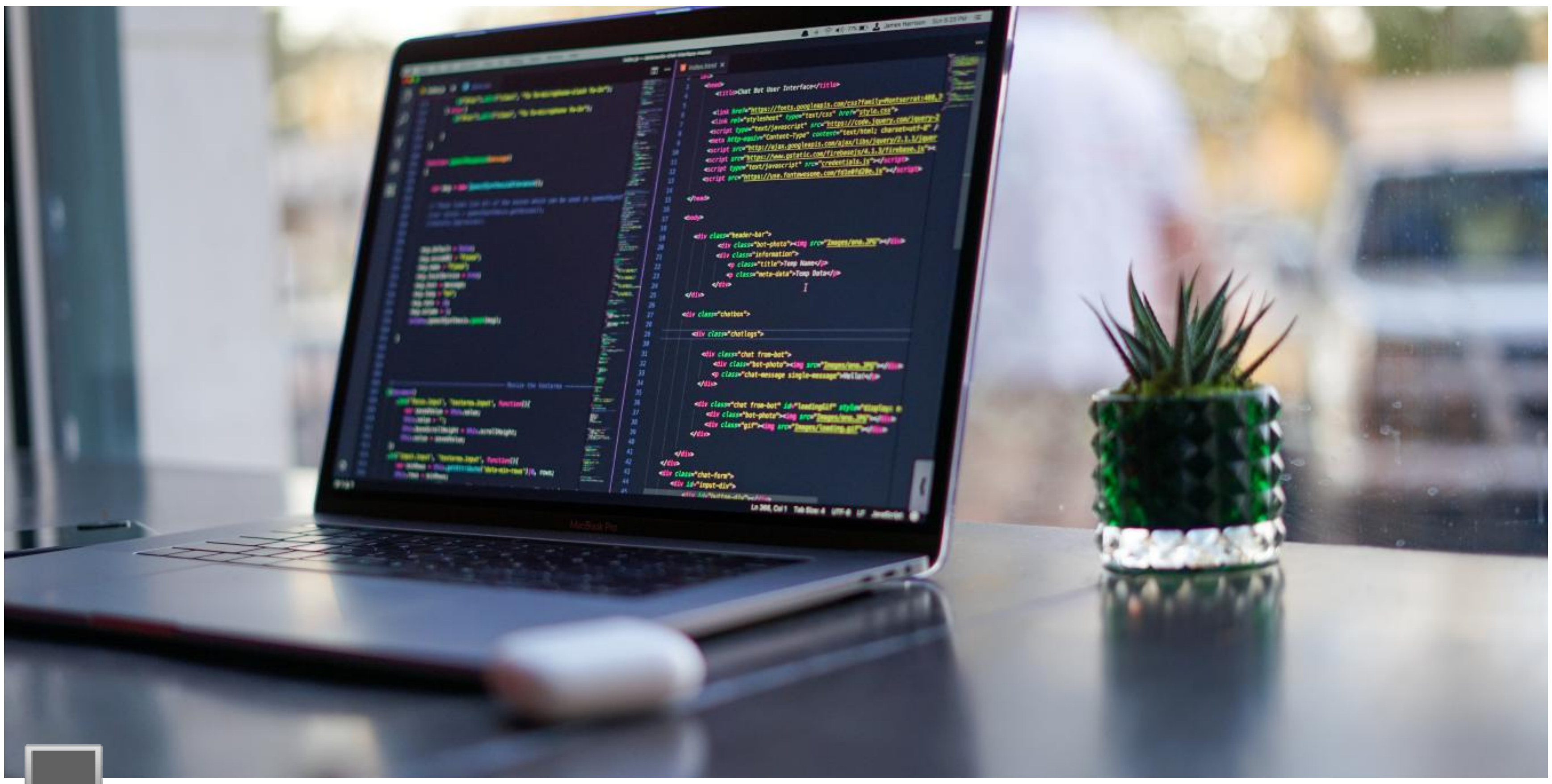
- Targeted at mobile platforms: Phones / Tablets

- Constraints
    - Limited screen space
    - User interaction (touch, audio, camera)
    - Memory / processing
    - Power
- Frameworks
    - OS specific
    - Cross-platform
- Network!
    - Usually network oriented

Web Apps

- The **Platform**
- Works across OS, device: create a common base
- Heavily network dependent
    - Workarounds for offline processing

Our main focus in this course ...



💻

Week 1 Lecture 2

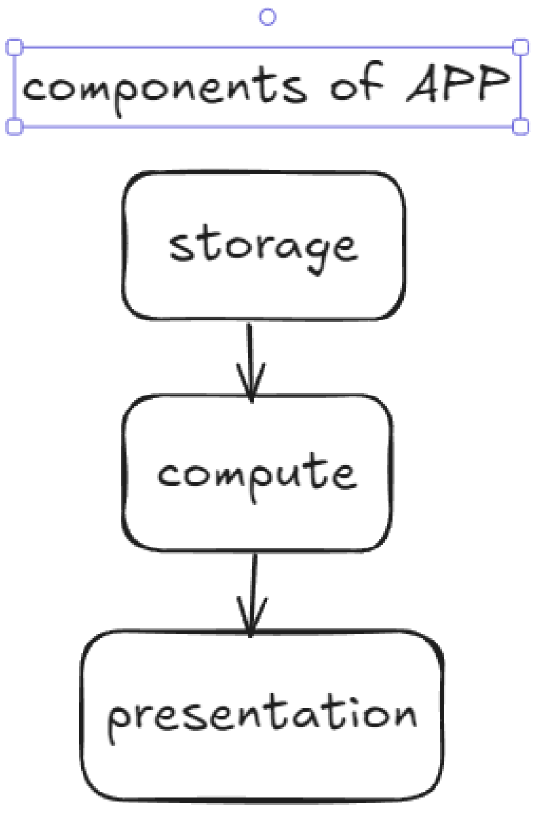
Class	BSCCS2003
Created	@September 4, 2021 8:10 PM
Materials	
Module #	2
Type	Lecture
Week #	1

Components of an App

Storage

Computation

Presentation



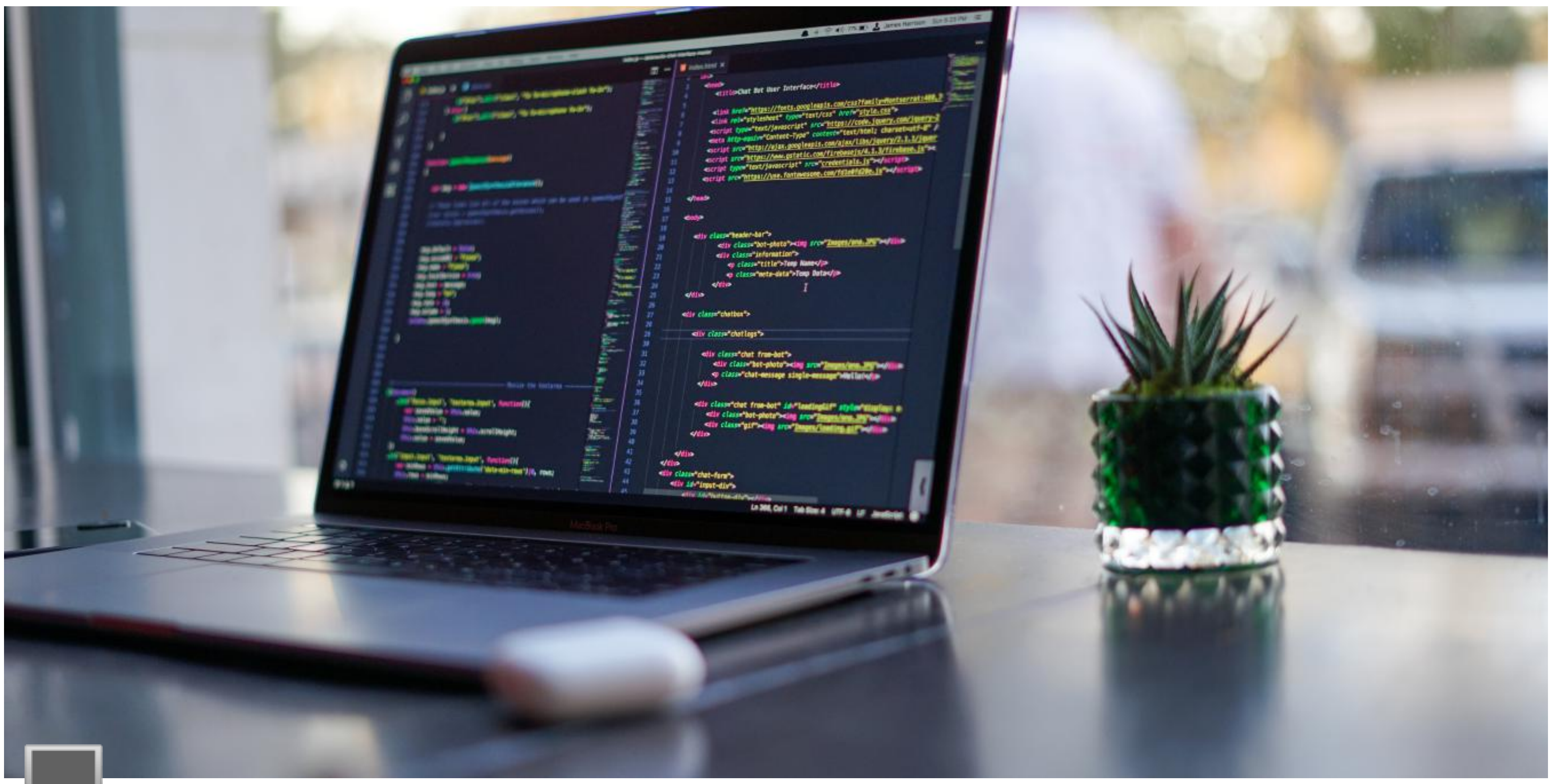
Example

- Storage
    - Where are the e-mails stored?
    - How are they stored on the server? File formats, etc.
- Compute
    - Indexing of e-mails
    - Searching
- Presentation
    - Display list of e-mails
    - Rendering / display of individual e-mails

Platforms

- Desktop
    - Keyboard, Mouse, Video as I / O
    - Desktop paradigm - folders, files, documents

- Mobile
  - Touchscreen as I / O
  - Voice, tilt, camera interfaces
  - Small self-contained apps
- Web-based
  - Datacenter storage - persistent
  - Cloud: access anywhere multi-device
- Embedded
  - Single-function, limited scope



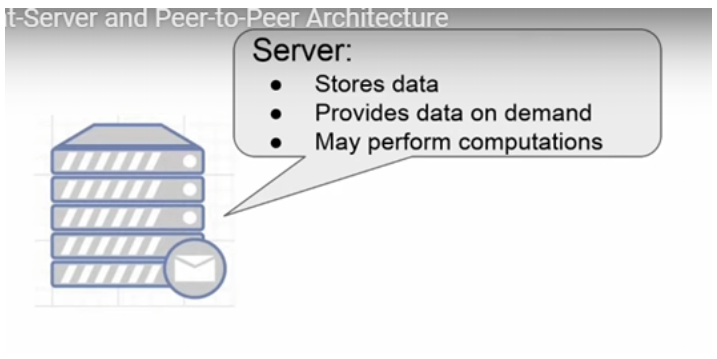
💻

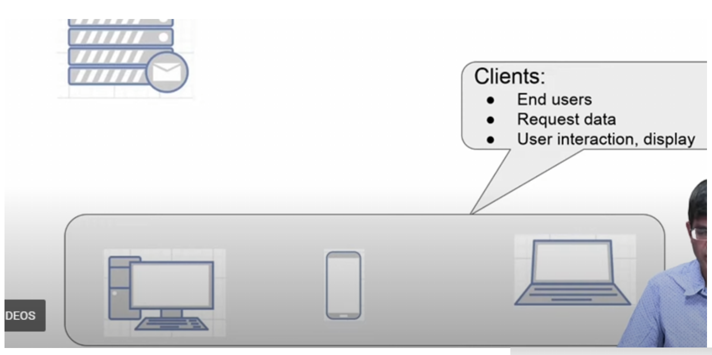
Week 1 Lecture 3

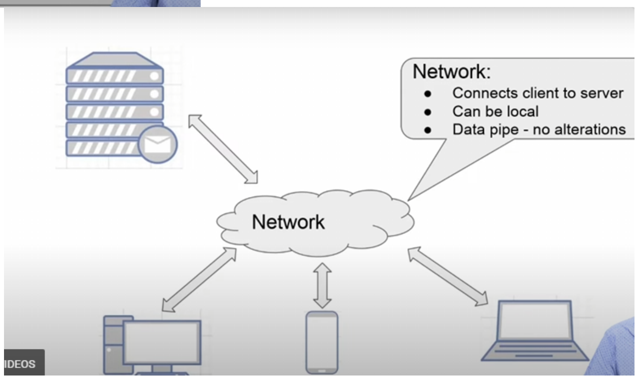
Class	BSCCS2003
Created	@September 5, 2021 7:38 PM
Materials	
Module #	3
Type	Lecture
Week #	1

Client-Server & Peer-to-Peer Architecture

Architectures





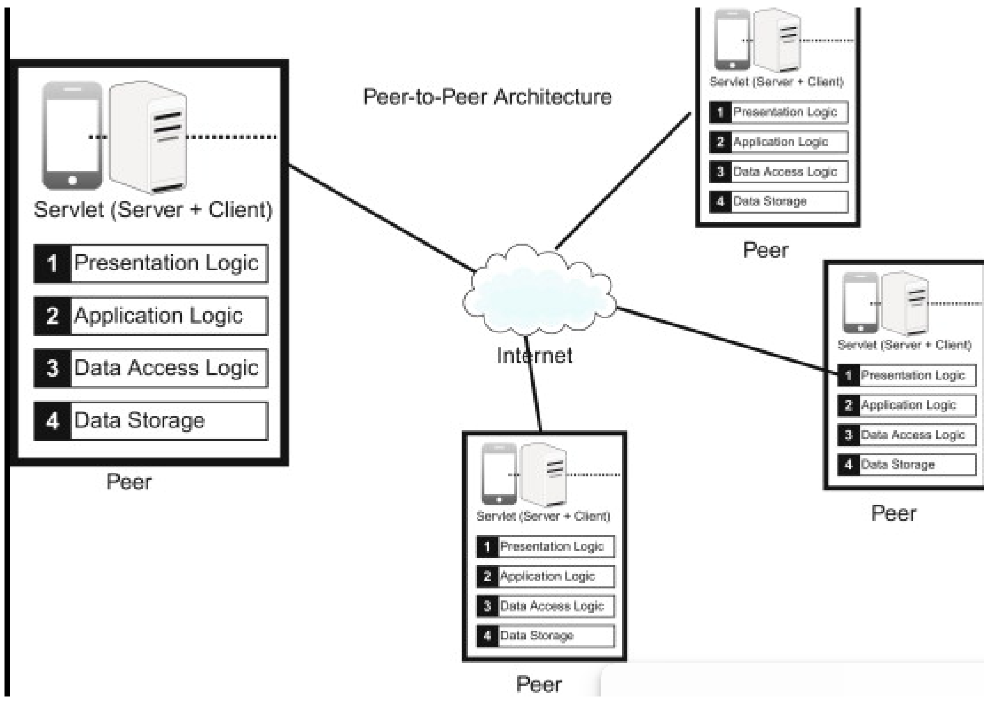


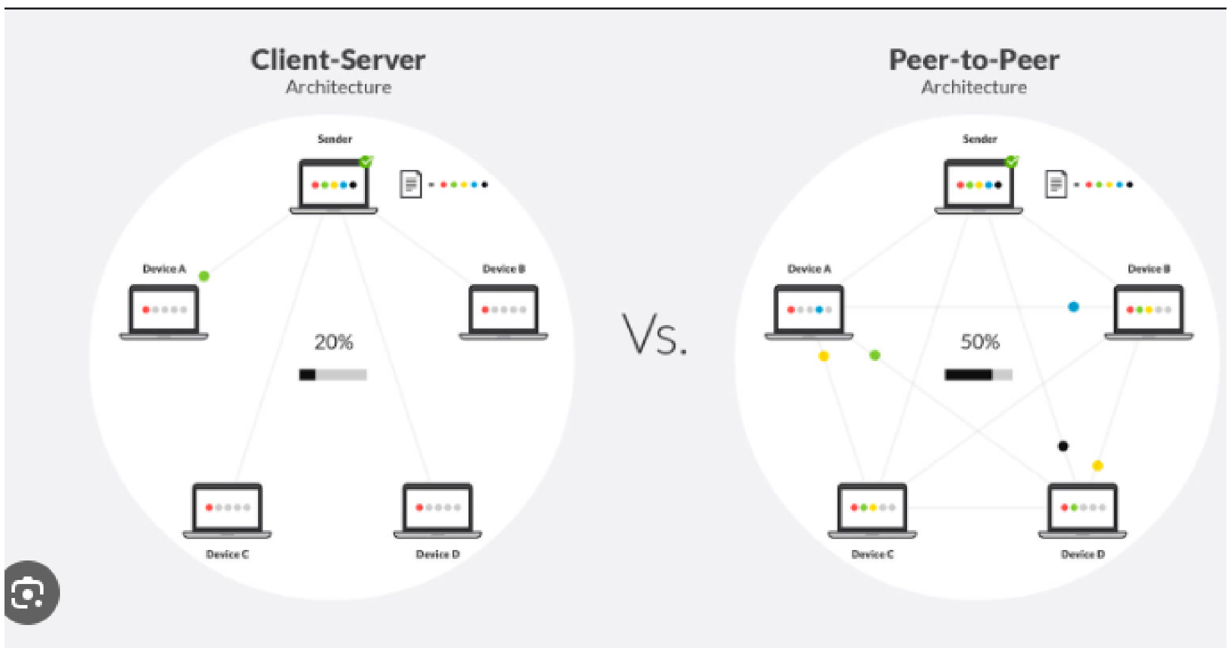
- **Client-Server**
  - Server:
    - Stores data
    - Provides data on-demand
    - May also perform computations
  - Client:
    - End users
    - Requests data
    - User interaction, display
  - Network:
    - Connects the client to the server
    - Can be local
    - Data pipe - no alterations
  - **In a client-server model, the server and client are explicitly defined**
  - Local systems:
    - Both client and the server on the system ⇒ local network / communication

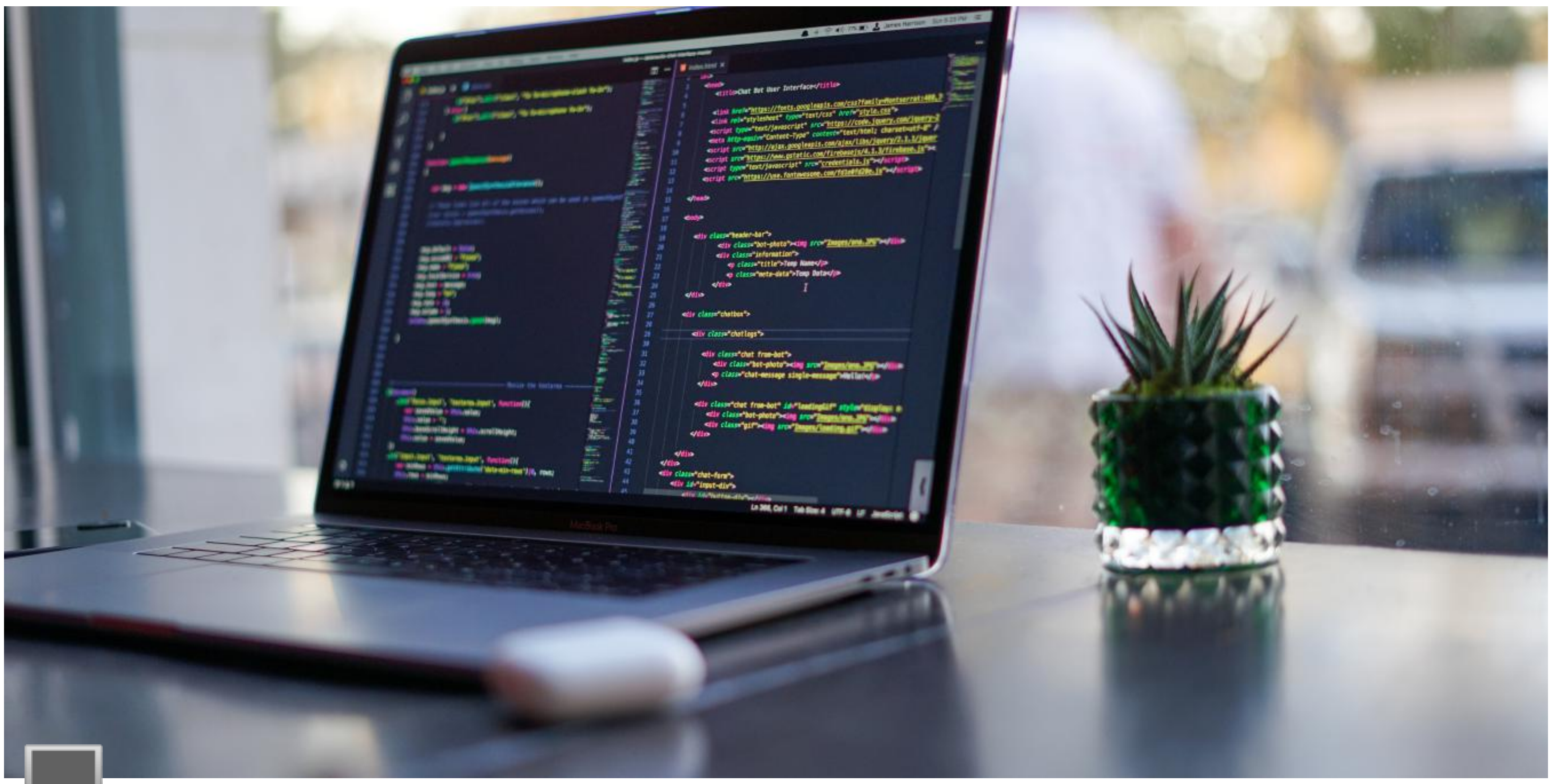
        - Conceptually, it is still a networked system
    - Machine clients
        - eg: Software, antivirus updaters
        - They need not have user interaction
    - Variants
        - Multiple servers, single queue, multiple queues, load balancing front-ends
    - Examples:
        - E-mail
        - Databases
        - WhatsApp / messaging
        - Web browsing
- **Distributed - Peer-to-Peer**

  Data can flow both ways
    - All peers are considered "equivalent"
        - But some peers may be more equal than others

          This just means that some servers with high bandwidth are given higher weightage
    - Error tolerance
        - Masters / Introducers
        - Election / re-selection of masters on failure
    - Shared information
    - Examples:
        - BitTorrent
        - Blockchain-based systems
        - IPFS, Tahoe (distributed file system)







💻

Week 1 Lecture 4

Class	BSCCS2003
Created	@September 5, 2021 8:33 PM
Materials	
Module #	4
Type	Lecture
Week #	1

Software Architectures

What is a design pattern?

"A general, re-usable solution to a commonly occuring problem within a given context in software design"

Source: Wikipedia

- Experienced designers observe "patterns" in code
- Re-using these patterns can make design and development faster
- Guide the design and thought process

M-V-C paradigm

- **Model:**
    - Core data to be stored for the application
    - Databases: indexing for easy searching, manipulation, etc;
- **View:**
    - User-facing side of the application
    - Interfaces for finding information, manipulating
- **Controller:**
    - "Business Logic" - how to manipulate the data

Origins in the smalltalk language, 1979

The user uses the **controller** → to manipulate the **model** → that updates the **view** → that the user sees;

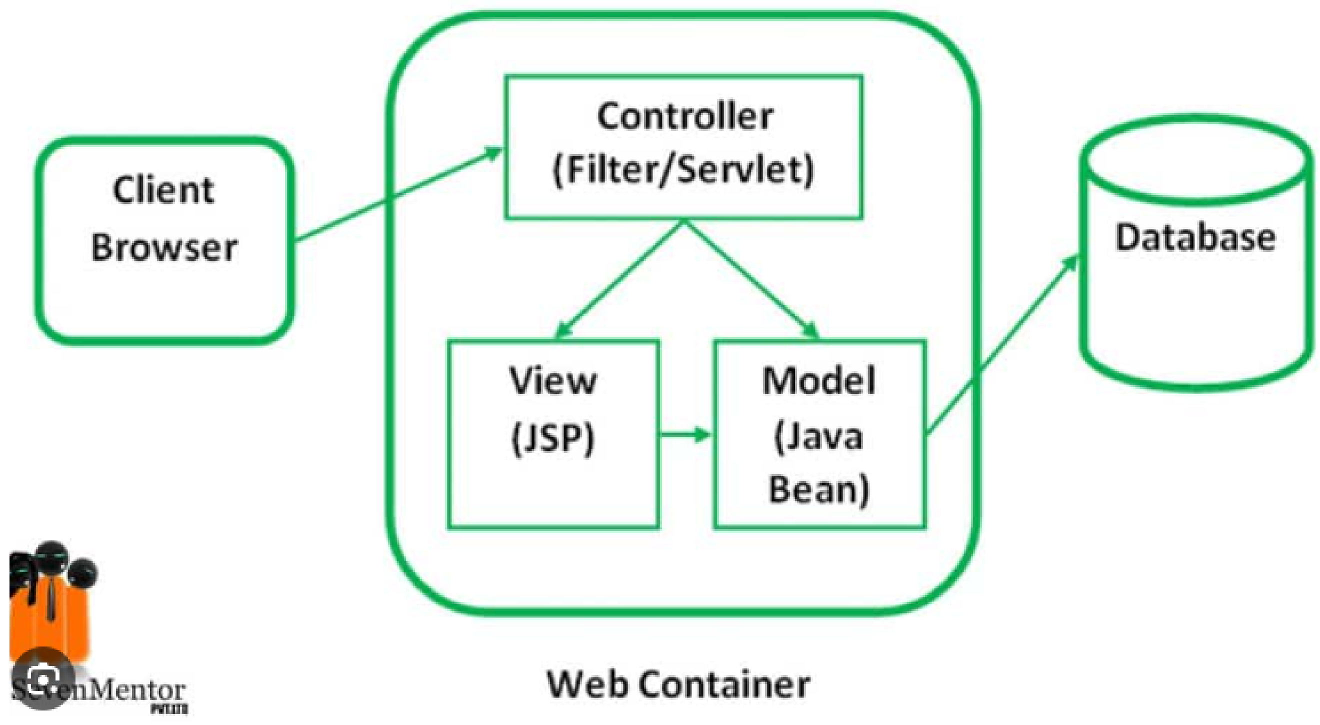
Other design patterns

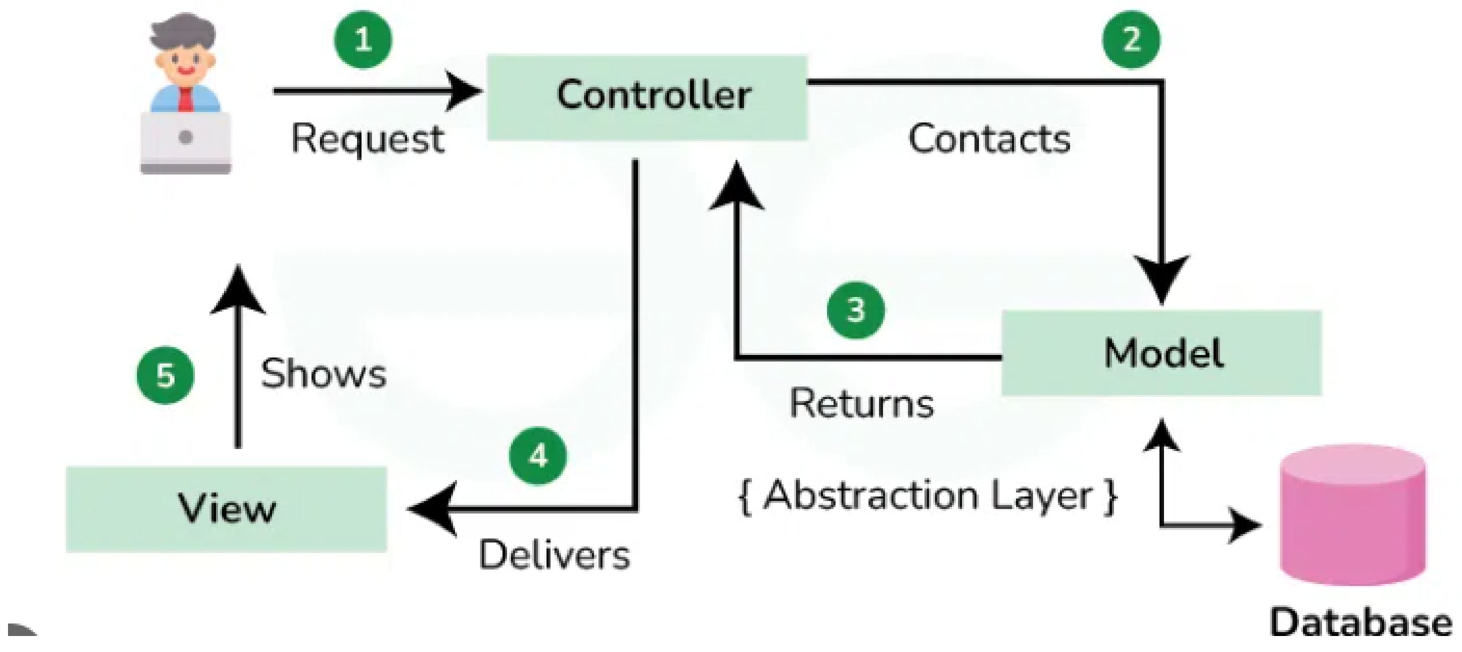
- Model-View-Adapter
- Model-View-Presenter
- Model-View-ViewModel
- Hierarchical MVC
- Presentation-Abstraction-Control

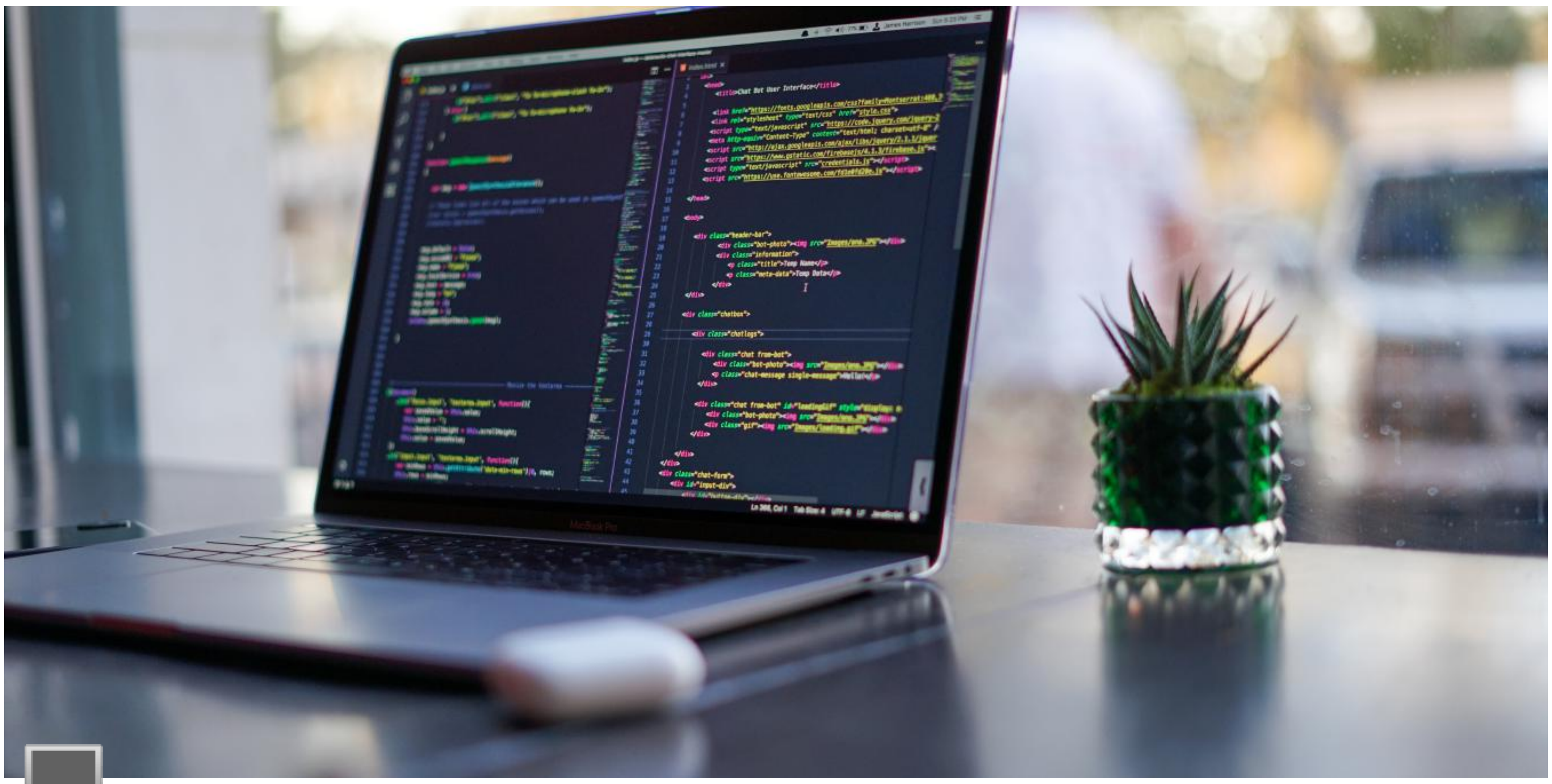
  Each one of them has its own uses, but the fundamentals are very similar

Focus of this course

- Platform:
  - Web-based
- Architecture:
  - Client-server
- Software architecture
  - Model-View-Controller







💻

Week 1 Lecture 5

Class	BSCCS2003
Created	@September 6, 2021 12:07 PM
Materials	
Module #	5
Type	Lecture
Week #	1

Introduction to the Web

- Platform of choice for this course
- Generic - works across operating systems, hardware architectures
    - Can it be considered as a cross-platform OS?
- Built on sound underlying principles
- Worth understanding
    - Constraints: what can and cannot be done (easily)
    - Costs: storage, network, device sizing, datacenter

Historical background

- Telephone networks ~ 1890+
    - Circuit switched - allow A to talk to B, through complex switching network
    - Physical wires tied up for duration of call even if nothing was said!
        - The bandwidth used by voice signal is pretty low
- Packet switched networks ~ 1960s
    - Wires occupied only when data was to be sent - more efficient use

      Last link: The physical connection between my telephone and my nearest telephone exchange, it has to be tied up to me

      Aggregate all the data in one place (telephone exchange) and transmit it over a single wire at a much higher speed (connection between telephone exchanges)

    - Data is being sent instead of voice over the wires
- ARPANet - 1969
    - acronym for Advanced Research Projects Agency Network
    - Node-to-node network
    - Mostly university driven
- Others:
    - IBM SNA, Digital DECNet, Xerox Ethernet, ...
- Protocol
    - How to format packets; place them on a wire; headers / checksums;
    - Each network had its own protocol
- "Inter" network
    - How to communicate between different network protocols?
    - Or replace them with a single "inter"net protocol
- IP: Internet Protocol - 1983
    - Define headers, packet types, interpretation
    - Can be carried over different underlying networks: ethernet, DECnet, PPP, SLIP
- TCP: Transmission Control Protocol - 1983
    - Establish reliable communication - retry; error control
    - Automatically scale and adjust to network limits
    - It is a connection oriented protocol
- Domain names ~ 1985
    - Use names instead of IP addresses
        - IP addresses were basically a set of 4 numbers, what they called the dotted quad notation
        - There are 4 numbers in the range of 0 - 255, which are separeted by dots (.)
    - Easy to remember - *.com* revolution still in the future
- HyperText ~ 1989+
    - Text documents to be "served"
    - Formatting hints inside documents to "link" to other documents - HyperText
    - It was brought into one consolidated framework by Tim Berners Lee at CERN
    - All this lead to the creation of the World Wide Web

IP (Internet Protocol) is indeed a universal protocol that operates across various types of networks, not just the internet.

Here's how it works:

IP provides the rules for how data packets are structured and routed between devices, no matter what kind of network they're on (like local area networks (LANs), wide area networks (WANs), or the internet itself).
Whether you're on Wi-Fi, Ethernet, or even a mobile network, IP is used to ensure that data is sent to the correct device by assigning unique IP addresses.

TCP (Transmission Control Protocol) is like a set of rules that ensures reliable communication between devices over a network, such as the internet. Think of it as a conversation between two people:

Starting the Conversation (Handshake): Before sending data, TCP makes sure both devices (sender and receiver) are ready to communicate by performing a "handshake." This ensures the connection is reliable.

Sending Data in Chunks: Once the connection is established, TCP breaks large pieces of data into smaller packets and sends them. If any packets get lost along the way, TCP asks for them to be resent.

Reassembling and Confirming: The receiver collects all the packets, reassembles them into the original message, and sends a confirmation back to the sender to let them know everything arrived safely.

Ending the Conversation: After all data is successfully sent, TCP closes the connection, like saying goodbye after a conversation.

In summary, TCP makes sure that data sent

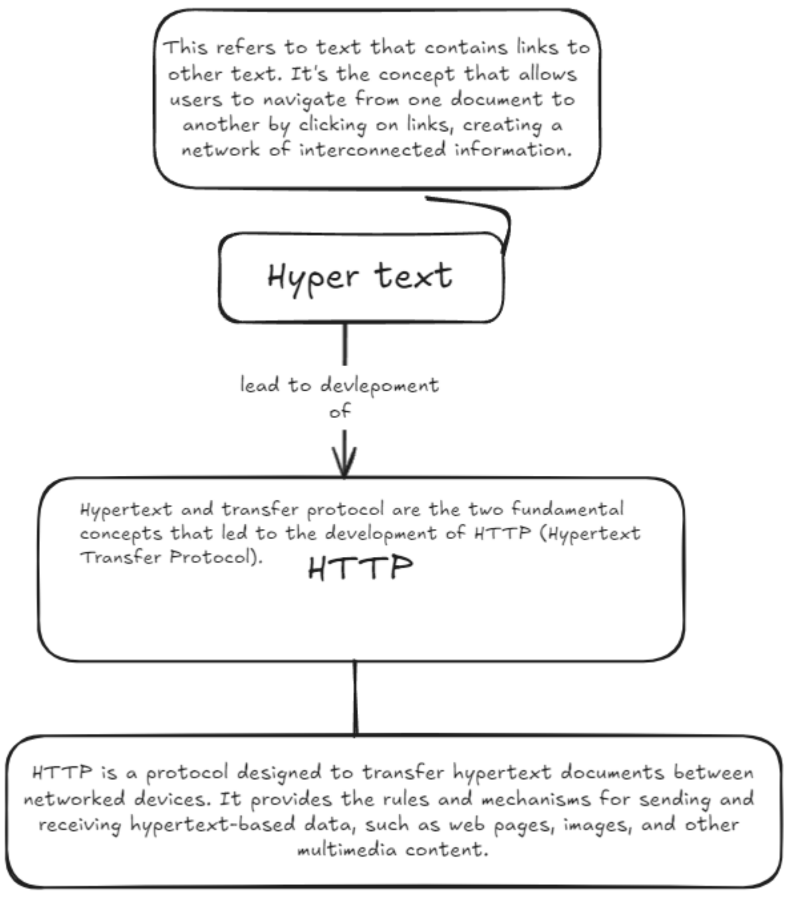
Then an IP address linked with a domain name. A domain name is essentially a human-readable alias for an IP address, which is the unique address assigned to each device or server connected to the internet.

Here's how it works:

When you type a domain name (like example.com) in your browser, a service called DNS (Domain Name System) translates that domain into its corresponding IP address (e.g., 93.184.216.34).

Where are we now?

- Original web was limited to ...
    - static pages
    - complicated executable interfaces
    - limited styling
    - browser compatibility issues
- Web 2.0 ~ 2004+
    - dynamic pages - generated on the fly
    - HTTP as a transport mechanism - binary data; serialized objects
    - client side computation and rendering
    - platform agnostic operating system


This refers to text that contains links to other text. It's the concept that allows users to navigate from one document to another by clicking on links, creating a network of interconnected information.

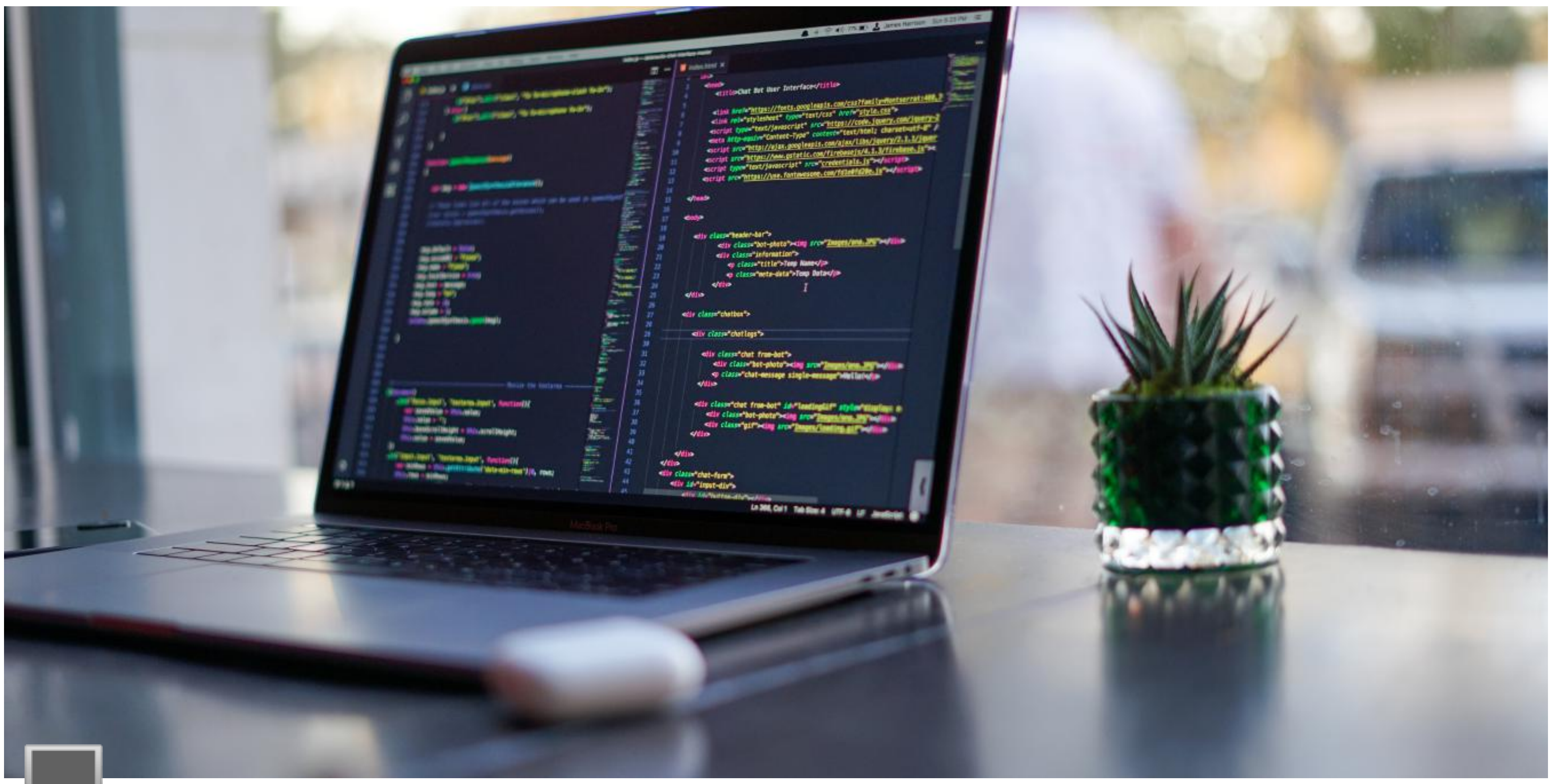
Hyper text

lead to devlepoment of

Hypertext and transfer protocol are the two fundamental concepts that led to the development of HTTP (Hypertext Transfer Protocol).

HTTP

HTTP is a protocol designed to transfer hypertext documents between networked devices. It provides the rules and mechanisms for sending and receiving hypertext-based data, such as web pages, images, and other multimedia content.



💻

Week 1 Lecture 6

Class	BSCCS2003
Created	@September 6, 2021 2:01 PM
Materials	
Module #	6
Type	Lecture
Week #	1

How does the Web work?

Digging deeper

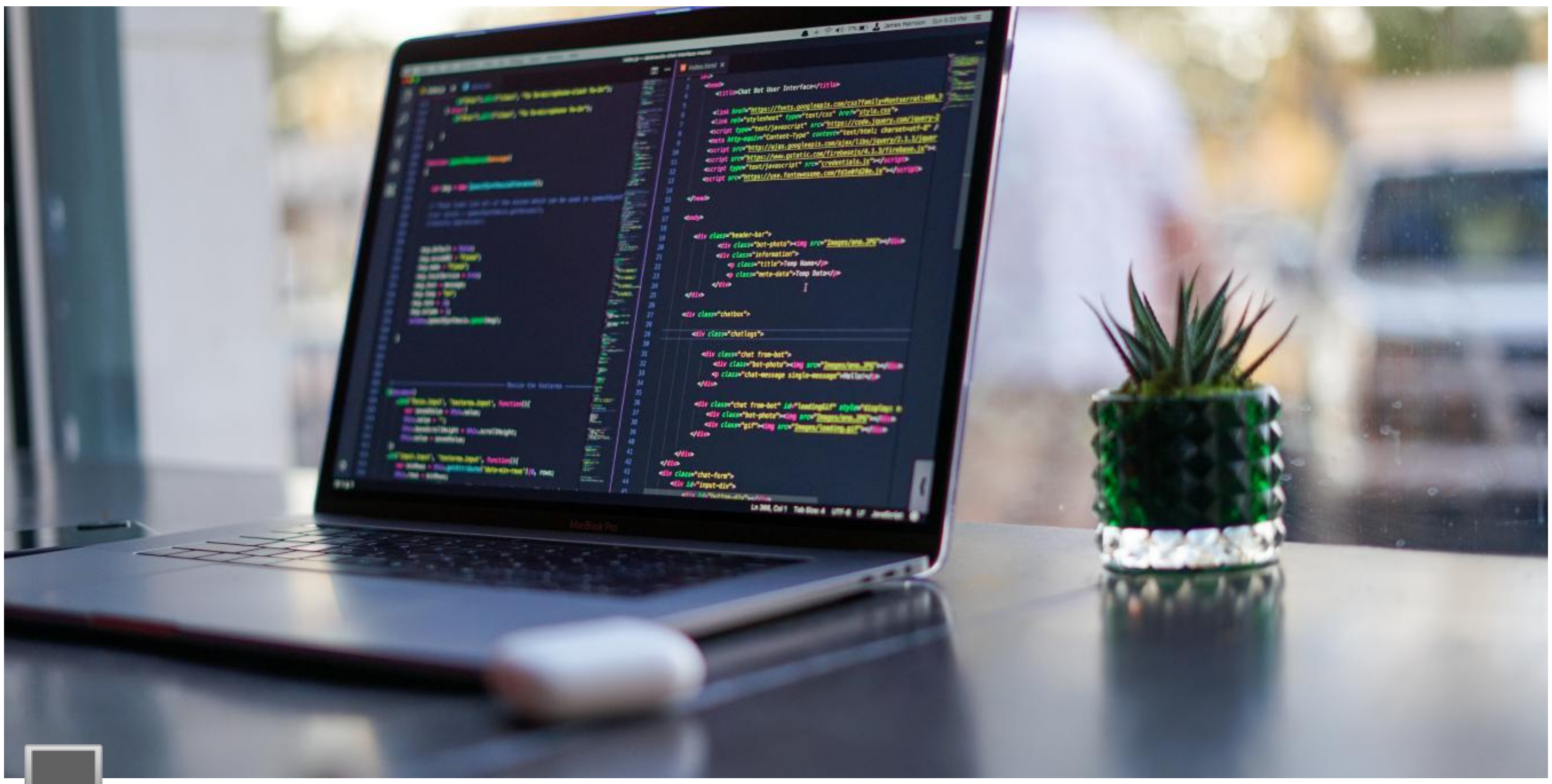
- What is a server?
- What is a transport protocol?
- Impact on performance:
    - Network
    - Server compute resources
    - Storage requirements
    - Client compute resources

Web Server

- Any old computer with a network connection can act as a web server
- Software:
    - Listen for incoming network connections on a fixed port
    - Respond in specific ways
    - Opening network connections, ports etc already known to OS
- Protocol:
    - What should client ask server
    - How should the server respond to the client

HTTP

- HyperText
    - Regular text documents
    - Containes "codes" inside that indicate special functions - how to "link" to other documents
- HyperText Transfer Protocol
    - Largely text based: client sends requests, server responds with hypertext documents



💻

Week 1 Lecture 7

Class	BSCCS2003
Created	@September 6, 2021 2:43 PM
Materials	
Module #	7
Type	Lecture
Week #	1

Simple Web Server

Simplest server

```
#!/bin/bash

while true; do
  echo -e "HTTP/1.1 200 OK\n\n $(date)" |
    nc -l localhost 1500;
done
```

Create a new file on the linux terminal

```
$ nano server.sh
```

Write the above mentioned server bash script in that file, and then to run the script, type the following code on the linux terminal

```
$ bash server.sh
```

We can send a request to the running server from another instance of linux shell

```
$ curl http://localhost:1500
```

We can also add a verbose flag to get a ton more info

```
$ curl -v http://localhost:1500
```

What the general web server is actually doing?

- Listening on a fixed port
- On incoming requests, it ran some code, generated results on the fly and returned the results to the client
    - Standard headers to be sent as a part of the result
    - Output can be in text or other format - MIME (Multipurpose Internet Mail Extensions)

What does a typical request look like?

```
GET / HTTP/1.1
Host: localhost:1500
User-Agent: curl/7.64.1
Accept: */*
```

Viewing responses with curl

- curl, wget, etc - simple command line utilities
- Can perform full HTTP requests
- Verbose output includes all the headers
- Very useful for debugging!

What does a typical response look like?

When you make a simple http request to a web server via curl, the following response appears ...

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 1500 (#0)
> GET / HTTP/1.1
> Host: localhost:1500
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
* no chunk, no close, no size. Assume close to signal end
<
 Thu Jun 17 08:14:55 IST 2021
```

NOTE: The $::1$ is an IPv6 address which represents $localhost$

1. bin/bash:
This line specifies the Bash shell to execute the script. It's optional but often used for clarity.

2. while true; do ... done:
This creates an infinite loop. The script will continue to execute the commands within the loop until it's manually stopped.

3. echo -e "HTTP/1.1 200 OK\n\n$(date)":

   a. echo -e: This command is used to print text to the standard output. The -e flag enables escape sequences.

   b. "HTTP/1.1 200 OK\n\n$(date)":
      1. "HTTP/1.1 200 OK\n\n": This is the HTTP response header. It indicates a successful response (200 OK) and includes a blank line to separate the header from the body. HTTP/1.1 is included in the HTTP response header to specify the version of the HTTP protocol being used.

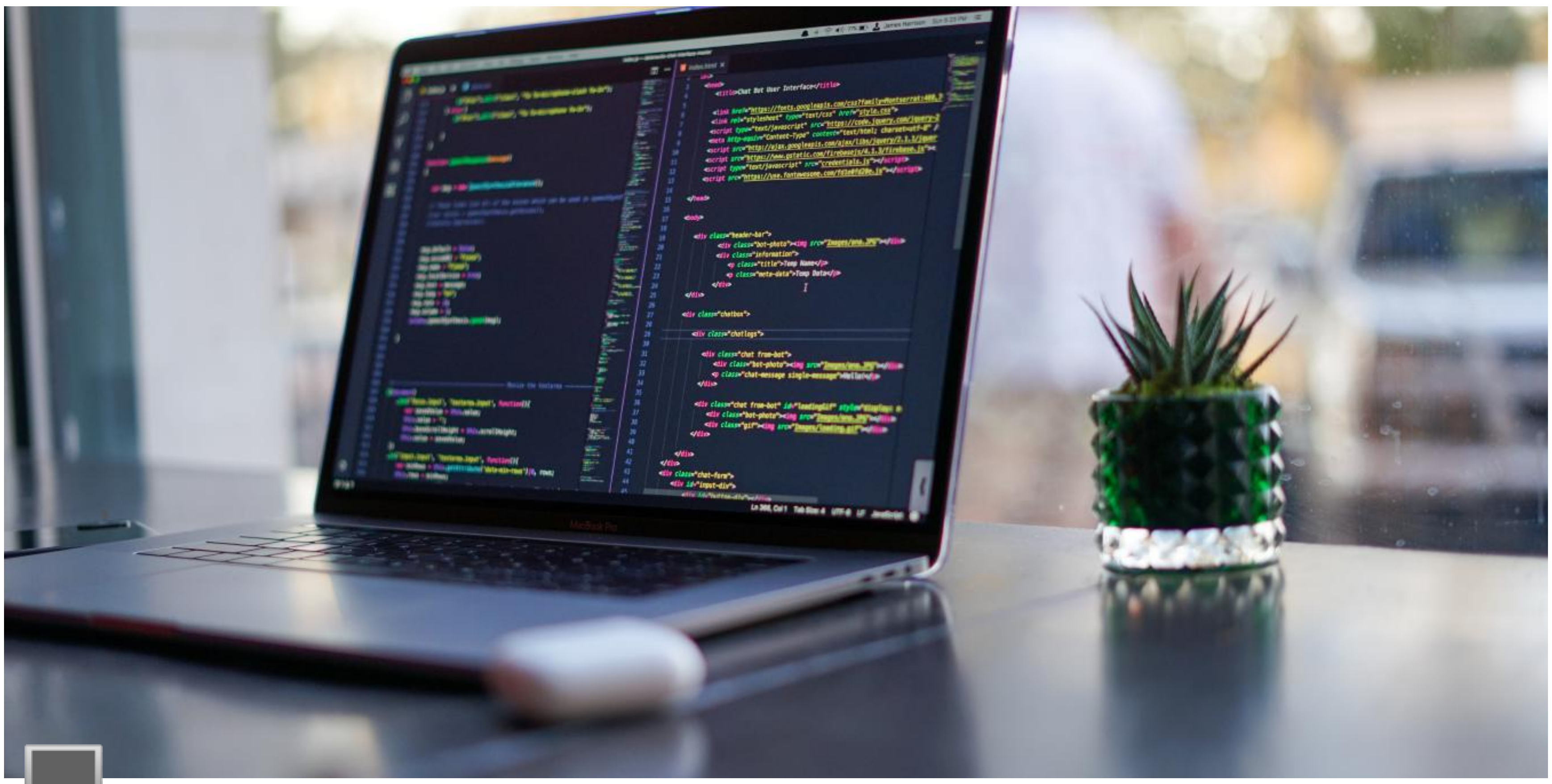
      2. $(date): This is a Bash command substitution. It executes the date command and replaces the $(date) with the current date and time.

4. | nc -l localhost 1500:

   |: This is a pipe character. It takes the output of the previous command and feeds it as input to the next command.

   nc -l localhost 1500:
      nc: This is the netcat command, a network utility that can be used to create network connections.
      -l: This flag indicates that netcat should listen on a specific port.
      localhost: This specifies the local machine as the server.
      1500: This is the port number on which the server will listen.

Overall Functionality:
Infinite Loop: The script enters an infinite loop, continuously executing the commands inside the loop.
HTTP Response Generation: The echo command generates an HTTP response with a 200 OK status code and the current date.
Listening on Port: The nc command starts listening on port 1500 of the local machine.
Sending Response: When a client connects to port 1500, the nc command sends the generated HTTP response to the client.
Loop Continuation: The script then returns to the beginning of the loop, waiting for another client connection.
In essence, this script creates a simple web server that listens on port 1500 and responds to every incoming request with a 200 OK status and the current date.



💻

Week 1 Lecture 8

Class	BSCCS2003
Created	@September 6, 2021 3:26 PM
Materials	
Module #	8
Type	Lecture
Week #	1

What is a Protocol?

- When both sides agree on how to talk
- Server expects "requests"
    - Nature of requests
    - Nature of client
    - Types of results that the client can deal with
- Client asks "responses"
    - Asks server for something
    - Convey what you can accept
    - Read the result and process

HyperText Transfer Protocol (HTTP)

- Primarily text based
- Requests specified as "GET", "POST", "PUT", etc
    - Headers can be used to convey acceptable response types, languages, encoding
    - Which host to connect to (if multiple hosts are there on a single server)
- Response headers
    - convey message type, data
    - cache information

    - status codes: 404 not found, etc

Use cases

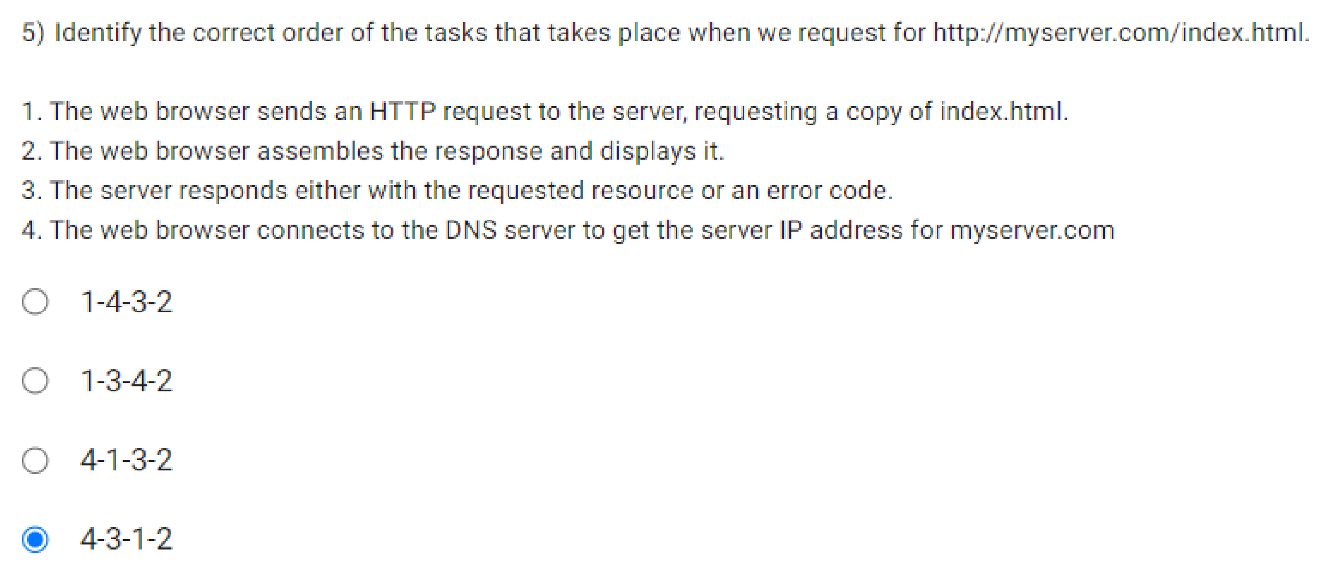
- **GET**: simple requests, search queries, etc
- **POST**: more complex form data, large text blocks, file uploads
- **PUT** / **DELETE** / ...
    - Rarely used in Web 1.0
    - Extensively used in Web 2.0
    - Basis of most APIs - REST, CRUD

Another web server

```
$ python -m http.server
```

- Serve files from local directory
- Understands basic HTTP requests
- Gives more detailed headers and responses

When the above command is run in a given directory, if I find a file called $index.html$, return me that file else return me the directory listing

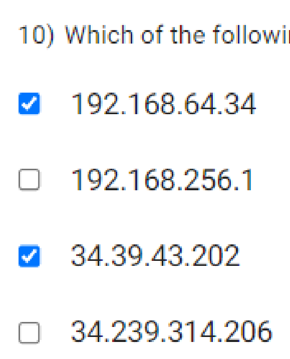


5) Identify the correct order of the tasks that takes place when we request for http://myserver.com/index.html.

1. The web browser sends an HTTP request to the server, requesting a copy of index.html.
2. The web browser assembles the response and displays it.
3. The server responds either with the requested resource or an error code.
4. The web browser connects to the DNS server to get the server IP address for myserver.com

- 1-4-3-2
- 1-3-4-2
- 4-1-3-2
- [✓ 4-3-1-2]

Rules for IPv4 Address:
An IPv4 address consists of four numbers separated by dots.
Each number (called an octet) must be in the range 0 to 255 (inclusive).



10) Which of the following is/are valid IPv4 address(es)?

- [✓ 192.168.64.34]
- 192.168.256.1
- [✓ 34.39.43.202]
- 34.239.314.206

Domain, Root Domain, and Subdomain Concepts
1. Domain Name:
A domain name is the address people use to access a website on the internet. It's made up of several parts, separated by dots, each serving a specific purpose. For example, in www.example.com, the domain name is example.com.

2. Root Domain:
The root domain refers to the highest level of a domain that consists of:

Top-Level Domain (TLD): This is the last part of the domain name (e.g., .com, .org, .net).
Second-Level Domain (SLD): This is the name you register and choose for your site (e.g., example in example.com).
Root Domain Structure:

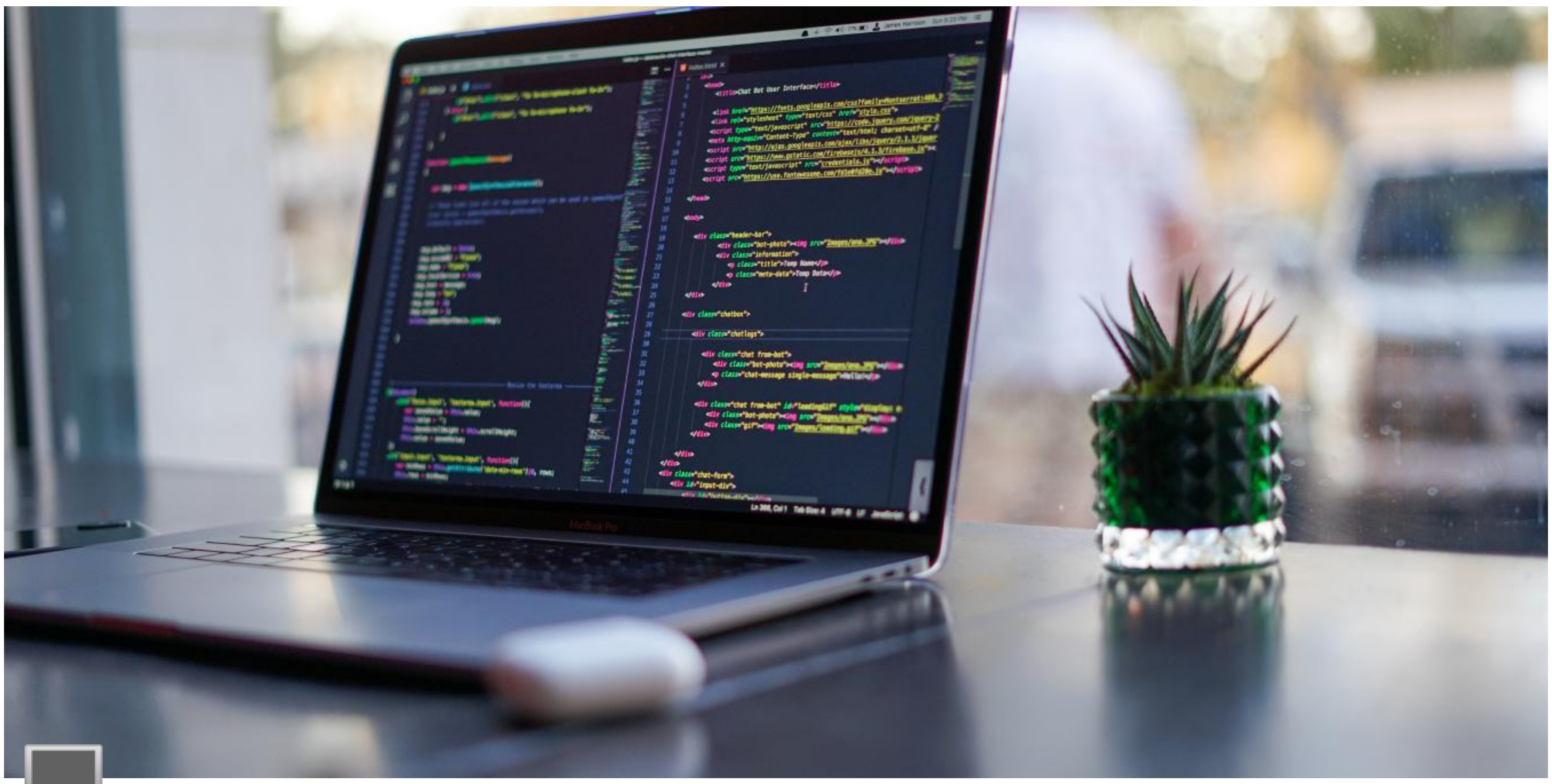
example.com (Root Domain) = Second-Level Domain (example) + Top-Level Domain (.com)
The root domain is essential for identifying a website uniquely on the internet.

3. Subdomain:
A subdomain is a prefix added to the root domain, creating a separate, distinct section of the website. It comes before the root domain. Subdomains are often used to organize different parts of a website or host specific content.

For example:

blog.example.com
blog is the subdomain.
example.com is the root domain.
Subdomains act like a "child" domain under the root domain and can point to different sections or services of a website, such as:

blog.example.com (for a blog section)
shop.example.com (for an e-commerce section)
support.example.com (for a support section)



💻

Week 1 Lecture 9

Class	BSCCS2003
Created	@September 6, 2021 3:44 PM
Materials	
Module #	9
Type	Lecture
Week #	1

Performance of a Website

- How fast a site can be?
- What limits the performance?
- Some basic observations ...

Latency

How much time does it take to get the response for a given request

- Speed of light: $3 \times 10^8 m/s$ in vacuum
- ~$2 \times 10^8 m/s$ on a cable
  - ~5ns for 1m $\implies$ ~5ms for 1000km
- Datacenter is 2000km away
  - One-way request: ~10ms
  - Round-trip: 20ms

So, what does this mean?

If I am continuously sending requests to a server, and I want it such that whenever I send a request, I get a response back before I can then proceed, then

I am limited to a maximum of 50 requests per second

What about a server that needs to send back responses?

Response size

- Response: 1KB of text (headers, HTML, CSS, JS)
- Network connection = 100Mbps (Megabits per second) ≈ 10MBps (Megabytes per second) [Theoretically, it should be 12.5MB/s]
- This limits us to ~10,000 requests per second

  For eg: Just imagine, that the result of some entrance exams were declared in India and if it is all trying to come out of one server with the bandwidth of 100 Megabits per second

  The server would simply crash when it is bombarded with more than 10,000 requests per second

A simple case study: Google response

- Headers: ~100B (bytes)
- Content: 144kB (kilobytes)
- Approx. 60,000 requests per second

Which means that Google server effectively need more than 80Gbps of bandwidth

Memory: YouTube

- One python HTTP server process: ~6MB (measured)
- Multiple parallel requests: multiple processes
  - eg: YouTube will have long running server processes
- 2016 Presidential debate in the US:
  - 2M+ concurrent viewers

Storage: Google

- Index 100s of billions of web pages
- Cross-reference, pagerank
- Total index size estimate:
  - 100,000,000 Gigabytes $\Longrightarrow$ 100 Petabytes
- Storage?
- Retrieval

Summary

- The Web is a useful device / OS agnostic platform for apps
- Built for HTTP for transport, HTML and related tech for presentation
- Servers trivial at most basic
- Scaling requires careful design

General URL Structure:
A URL is generally composed of these main parts:

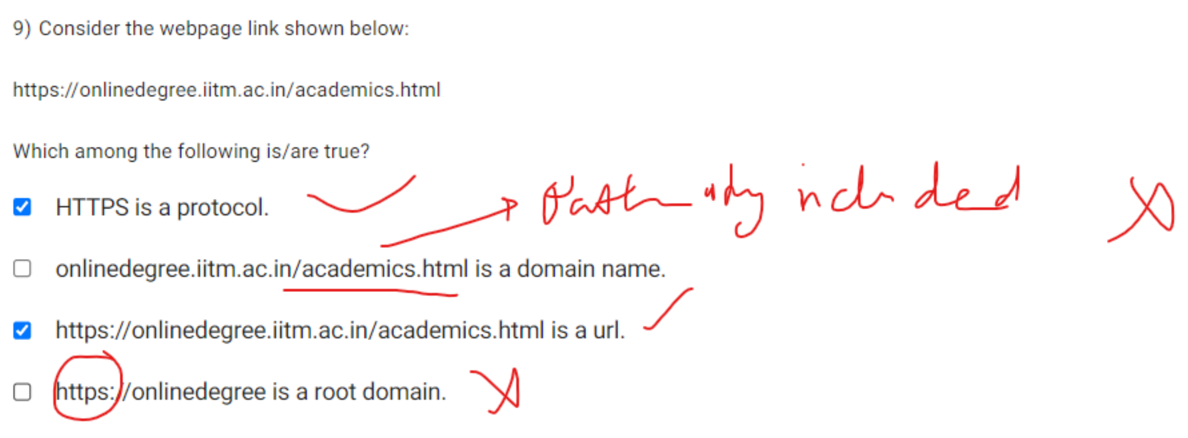
Protocol: Specifies the communication protocol (e.g., http://, https://).

Domain Name: The website's address (e.g., www.example.com).

Path: The specific location within the website (e.g., /about).

Query Parameters: Information passed to the server (e.g., ?q=search), which includes:

Key-Value Pairs: Like q=30+day+code where q is the key, and 30+day+code is the value.
Multiple parameters are separated by &.



9) Consider the webpage link shown below:

https://onlinedegree.iitm.ac.in/academics.html

Which among the following is/are true?

- [✓ HTTPS is a protocol.] ✓ → Path only included X
- onlinedegree.iitm.ac.in/academics.html is a domain name.
- [✓ https://onlinedegree.iitm.ac.in/academics.html is a url.] ✓
- https://onlinedegree is a root domain. X