Basic Client-Server Computing Terminology:

- At its core, client-server computing involves a client (usually running a browser) that cooperates with a web server (usually running a program)
    - The server maintains state typically through databases, and the client inquiries may require the current state or request a change in state
- Server programs face similar issues that normal programs deal with - CPU time performance, real-time performance, memory usage, etc.
- However, server programs also deal with new issues:
    - They have to try to attain a higher *throughput*, which is the number of client requests handled per second
        - Throughput may be improved if the server does actions out of order, allowing for easier tasks to be done first
            - However, this may increase latency due to requests having an unlucky ordering having an overall longer delay
        - The server can also leverage parallelism to better improve throughput
        - Out-of-order processing and parallelism may both lead to correctness issues (i.e. you try to delete an item from a database before it is even added)
    - They also have to try to attain a lower *latency*, which is the delay between the request to a server and the response back
        - Latency can be improved by building a better connection between the server and the client, though this is not feasible
        - Latency is more commonly improved by caching previous results so that, if a request can be found in cache, it can be sent immediately
            - A stale cache may occur, though, and give rise to another correctness issues - leading to the need for cache validation

Networking

- Circuit switching involves connecting networks via physical wires. When a signal is sent, it travels via these wires, and possibly through intermediary devices.
    - This has a guaranteed arrival of data, but requires physical connections
- Packet switching involves the data being sent being broken down into tiny "packets" such that they can be handled by local routers. These packets may take different paths as they are handed off to other routers, which follow a variety of protocols to help them reach their destination.
    - Packet switching has no guaranteed arrival (at least on the first try), as it is possible for packets to be lost
    - Additional, packets may arrive out of order (and even duplicated), requiring reassembly upon arrival

Internet Protocol Suite

- The Internet Protocol Suite acts as a foundational set of procedures to handle packet switching and its issues
- The Internet Protocol Suite is a layering of different protocols on top of each other:
    - **Link Layer:** Handles the basic point to point connection of two *adjacent* routers. This layer tends to be more hardware oriented, defining communication standards for adjacent hardware
    - **Internet Layer:** Handles with the process of *just sending* packets
        - It does not assume information about other routers, it simply forwards them packets
        - The Internet Layer is based off of the **Internet Protocol** (IP), which defines how a packet is structured - it consists of a **header**, containing metadata, and a **payload** containing the actual contents being sent
        - Common header metadata:
            - Length of packet
            - Protocol number
                - This indicates the type of data being sent, which can be useful for prioritizing certain packets or figuring out what to do when the data is received
            - Source IP address
                - In IPv4, this is a 32-bit long address
                - In IPv6, this address has been expanded to 128-bits (with backwards compatibility still being able to work)
            - Checksum
                - This is used as a basic check to determine if the data has been corrupted
            - Time-to-Live
                - This is a counter that measures the number of routers that the packet has "hopped". If it has hopped too many routers, then it assumed to be in a loop and is killed
    - **Transport Layer:** Handles data channels consisting of *multiple* packets
        - **User Datagram Protocol (UDP):** Used for applications that need single, short data messages over the internet where it does not necessarily matter if packets are lost
            - UDP is *quick* but may be *unreliable*
        - **Transmission Control Protocol (TCP):** Used for applications that need a reliable, typically large stream of packets. It ensures that the packets are reliable by getting an acknowledgement that it was sent (and resending if no acknowledgement was gotten)
            - TCP makes use of flow-control to ensure that a single router is not overloaded with too many packets
            - TCP also orders and error-checks the data before it is given to the next layer
    - **Application Layer:** Deals with individual applications, each of which may have their own unique application layer (web, voice, video, etc.)
        - **Realtime Transmission Protocol (RTP):** This protocol is built upon UDP and is used for applications that need realtime data (i.e. consider Zoom). As such, it needs data to be sent fast rather than completely reliably (it is fine if a Zoom call loses a frame or two)
        - **Hypertext Transfer Protocol (HTTP):** This protocol is built upon TCP and is used to send web pages. Here, it is more important that data is transmitted reliably rather than quickly.
            - HTTP is used to define web applications, where a client connects to a server through a TCP connection and makes various HTTP requests (GET, POST, etc.) to which the server responds - typically by sending back HTML content

    - HTTPS is a "Secure" version of HTTP in which third-party organizations provide certificates to ensure that a server is secure. It also involves encryption.
    - HTTP2 builds upon HTTP by adding features such as:
        - Header compression
        - Server push (the server can send a response without an initial request from the client)
        - Pipelining
        - Multiplexing (multiple connections can be run over the same TCP channel, possibly allowing for a user to easily navigate through multiple websites if they are all ran by the same server)
    - HTTP3, which is not released yet, will no longer be based on TCP but instead UDP

Alternatives to Client-Server Computing:

- Single computer applications
- Peer-to-peer connections:
    - Rather than there being a single, central server, clients (peers) just communicate with each other directly.
    - If a peer goes down, the system can still be reliable by connecting to another peer
    - While more reliable in terms of connectivity, this approach makes it difficult to manage a shared state among all peers
- Primary / Secondary
    - A primary machine keeps track of all work that needs to be done and then farms those tasks off to secondary servers (which return any results back to the primary server)