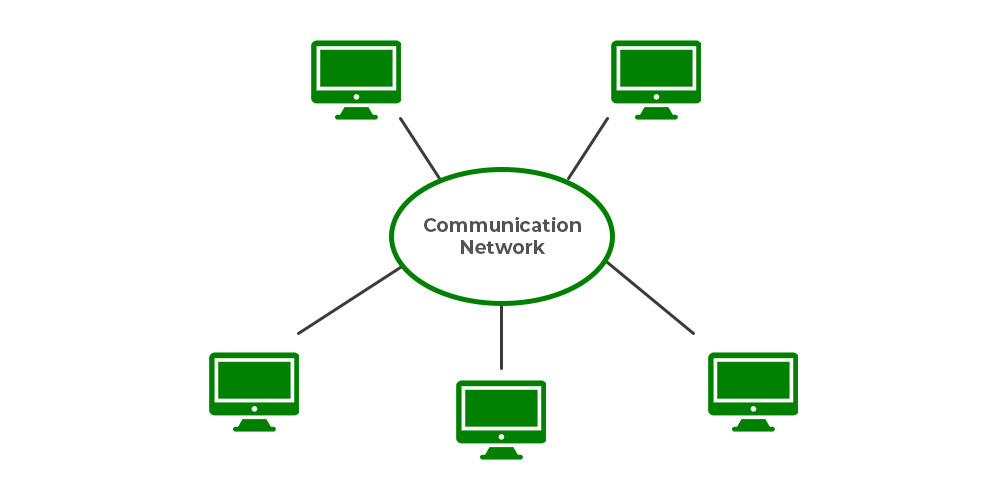
|  |  |
| --- | --- |
| **GDPARCM Lecture – Distributed Systems** | Instructor: Neil Patrick Del Gallego |



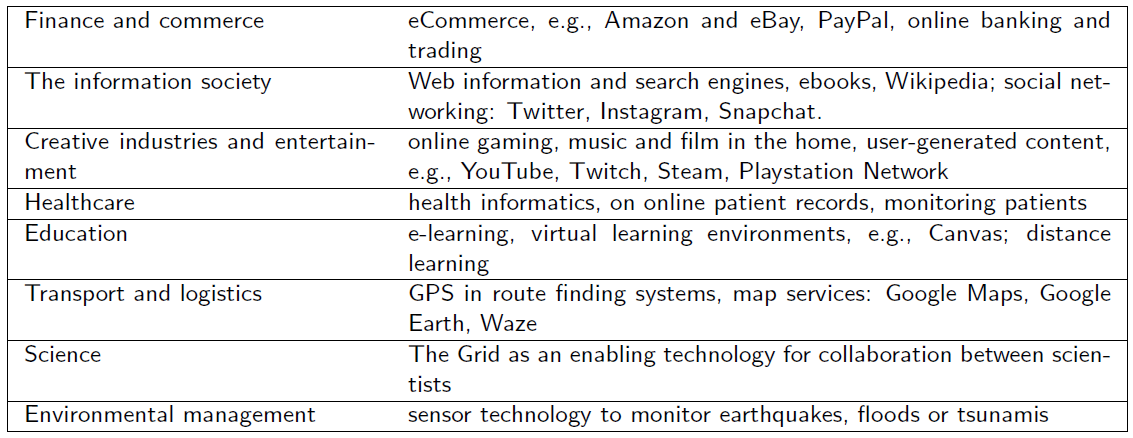
A screen shot of a computer

Description automatically generated

**Distributed Systems**

* A collection of interconnected computers that work together to achieve a common goal.
* Can also be a collection of running programs that work together. E.g. game client + cheat detector, windows application + notification handler.
* Creating distributed systems would require a combination of parallel computing and networking techniques.
* The entities communicate with each other by message passing 🡪 glorified chat.
* Internet is a huge distributed system.

**Applications of distributed systems**



Low-key examples of distributed systems:

* Mobile development IDE - Mobile devices with USB debugging
* Google Home, IOT
* Swarm coordination. E.g. drones.
* Web updates from a game. E.g. Progress trackers, character inventory viewers.

**Observation:** Any program, that requires communication with other programs, require distributed computing techniques.

**Challenges when designing distributed systems**

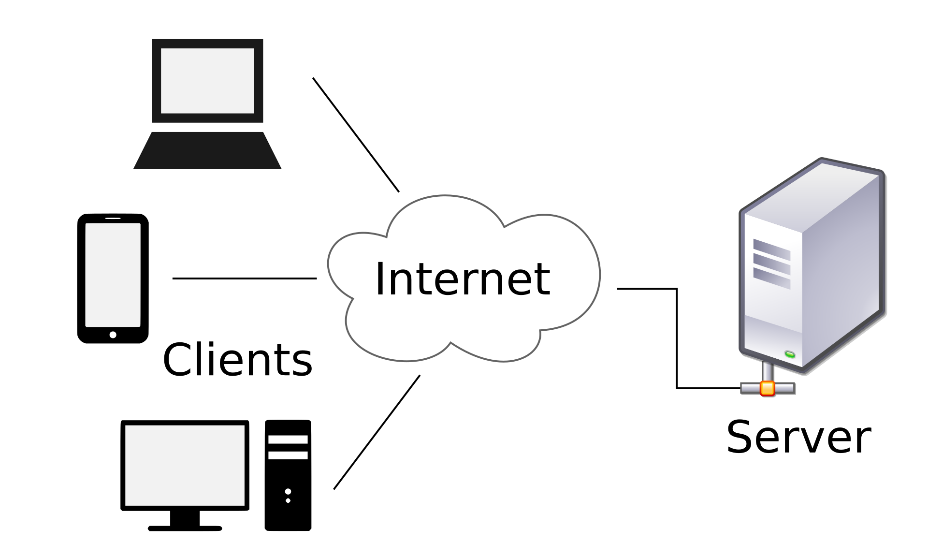
1. Concurrency – since multiple programs, across different hardware, different locations, are required, then concurrency techniques are needed.
2. Tolerance to failures – when one program fails, there is a risk of cascading failures – the cluster fails. One must design a fault-tolerant distributed system.
3. Scalability – Can we easily add new clusters to further improve the performance of the distributed system?
4. Concealment – A user must perceive the distributed system as only 1 whole program, rather than a collection of independent components.
5. Quality of service – how fast can the distributed system provide results?

**Got distributed system skills? What are the possible jobs?**

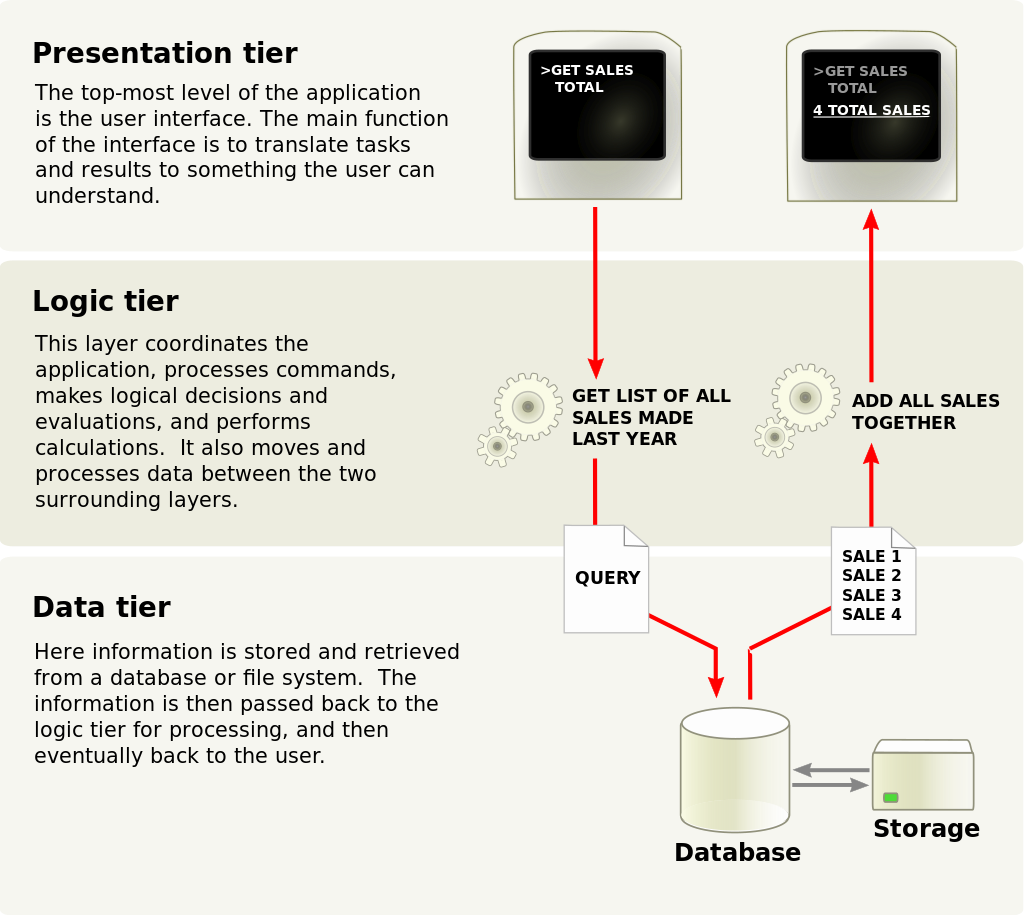
* File storage systems for servers 🡪 Google Drive
* Cloud compute 🡪 Amazon Web Servers, Google Cloud
* Real-time services 🡪 Banking transactions, MMO services

**System models**

There can be many ways to design distributed systems. It all depends on the problem and the best solution required. Common system models are: client-server, three-tier, n-tier, peer-to-peer.

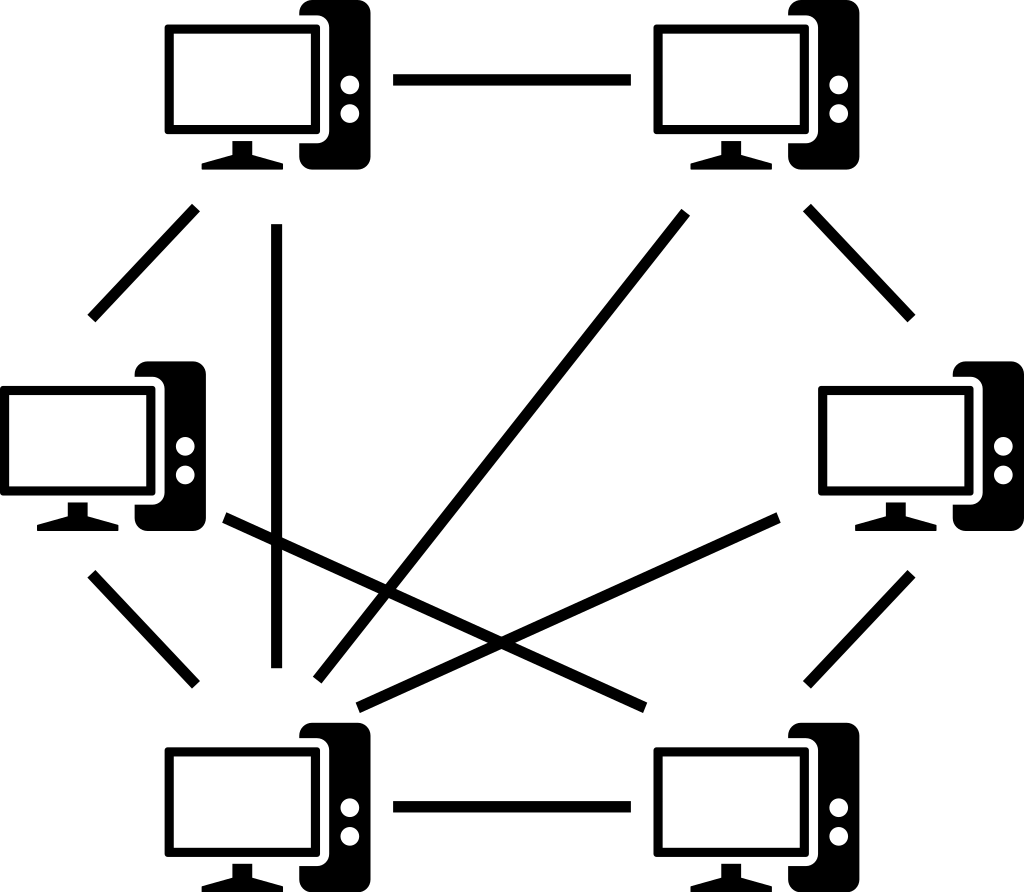


**Client-server architecture** is the most common. Server handles most of the demanding computations, clients generally handle light-weight computations, or generally receive the results from the server. Major cons of this is the server is a possible single point of failure.



The **three-tier** architecture attempts to separate the presentation/viewing processes, the logical operations, and the data storage in three different groups. Each tier can have its own collection of entities.

The **N-tiers** introduces additional tiers, aside from presentation, logic and data tiers.



The peer-to-peer architecture doesn’t have a server. Each client also acts as its own server. A portion of computing resources is allocated for processing and sending/receiving data from other network participants. Good example: torrents.

**Inter-process Communication**

Basic idea: Pass messages between applications. Message passing is the core operation for all distributed systems.

You can do a lot with message passing:

* Chat system
* Send images, videos through file streams.
* Trigger remote operations

Different libraries are available for supporting inter-process communications for distributed systems: Elixir, Netflix Hystrix, Resilience4J, LizardFS.

**Sockets**

For a process to receive messages:

* It’s a socket must be bound to a local port and one of the internet addresses of the computer on which it runs.
* Processes may use the same socket for sending and receiving messages 🡪 there can be 655365 possible port numbers to use: <https://networkappers.com/tools/open-port-checker>
* Any process may make use of multiple ports to receive messages, but a process cannot share ports with other processes on the same computer.



A white background with black text

Description automatically generated

**Hands-on activities**

* Demonstration of simple chat system using C++ sockets. GDPARCM\_HO7-SocketsDemo.