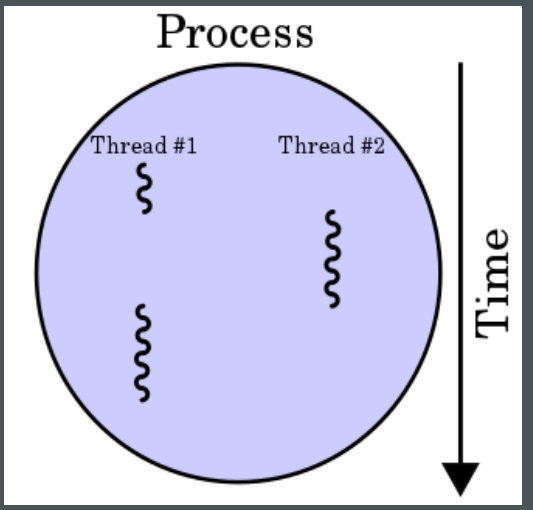
MULTITHREADING

In computer architecture, multithreading is the ability of a central processing unit (or a single core in a multi-core processor) to provide multiple threads of execution concurrently, supported by the operating system. This approach differs from multiprocessing. In a multithreaded application, the threads share the resources of a single or multiple cores, which include the computing units, and CPU cache.



For Example: Consider your internet browser. At any given time, you may have numerous tabs open, each one displaying various types of content. Multiple threads of execution are used to load content, display animations, play a video, and so on.

Another example of a multithreaded program that we are all familiar with is a word processor. While you are typing, multiple threads are used to display your document, asynchronously check the spelling and grammar of your document, generate a PDF version of the document. These are all happening concurrently, with independent threads performing these tasks internally.

USES OF MULTITHREADING

The main reason for incorporating threads into an application is to improve its performance. Performance can be expressed in multiple ways:

- A web server will utilize multiple threads to simultaneous process requests for data at the same time.
- An image analysis algorithm will spawn multiple threads at a time and segment an image into quadrants to apply filtering to the image.
- A ray-tracing application will launch multiple threads to compute the visual effects while the main GUI thread draws the final results.

Multithreading also leads to minimization and more efficient use of computing resources. Application responsiveness is improved as requests from one thread do not block requests from other threads.

Additionally, multithreading is less resource-intensive than running multiple processes at the same time. There is much more overhead, time consumption, and management involved in creating processes as compared to creating and managing threads.

ISSUES IN MULTITHREADING APPLICATION

For all the advantages of using multiple threads, they add complexity and can create tough bugs to solve. There are some common scenarios where you may encounter challenges with debugging multithreaded applications.

These include:

- Investigating data access issues where two threads are reading and modifying the same data. Without the proper use of locking mechanisms, data inconsistency and dead-lock situations can arise.
- Thread starvation and resource contention issues arise if many threads are trying to access a shared resource.
- Display issues can surface if threads are not coordinated correctly when displaying data.

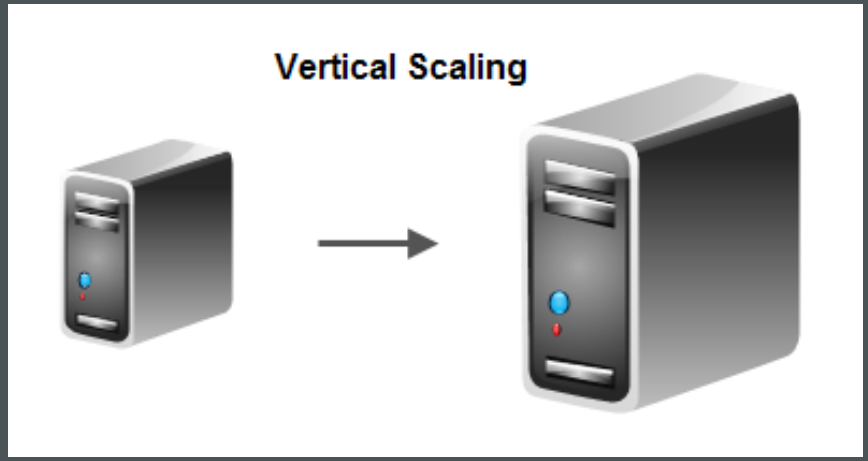
SCALBLE ARCHITECTURE

A scalable architecture is an architecture that can scale up to meet increased work loads. In other words, if the work load all of a sudden exceeds the capacity of your existing software + hardware combination, you can scale up the system (software + hardware) to meet the increased work load.

There are two primary ways to scale up a system: Vertical scaling and horizontal scaling.

Vertical scaling

Vertical scaling means that you scale up the system by deploying the software on a computer with higher capacity than the computer it is currently deployed on. The new computer may have a faster CPU, more memory, faster and larger hard disk, faster memory bus etc. than the current computer.



Horizontal scaling

Horizontal scaling means that you scale up the system by adding more computers with your software deployed on. The added computers typically have about the same capacity as the current computers the system is running on, or whatever capacity you would get for the same money at the time of purchase (computers tend to get more powerful for the same money over time).