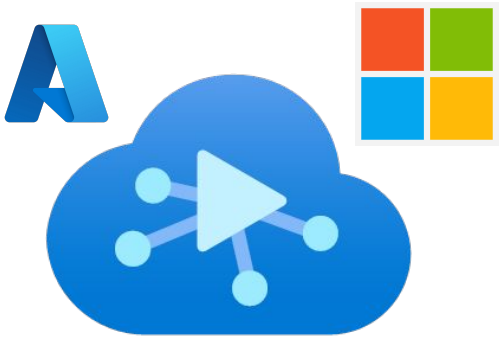
Practical Concurrency

Venkatraman Srikanth

Practical Concurrency (and Parallelism)

Venkatraman Srikanth

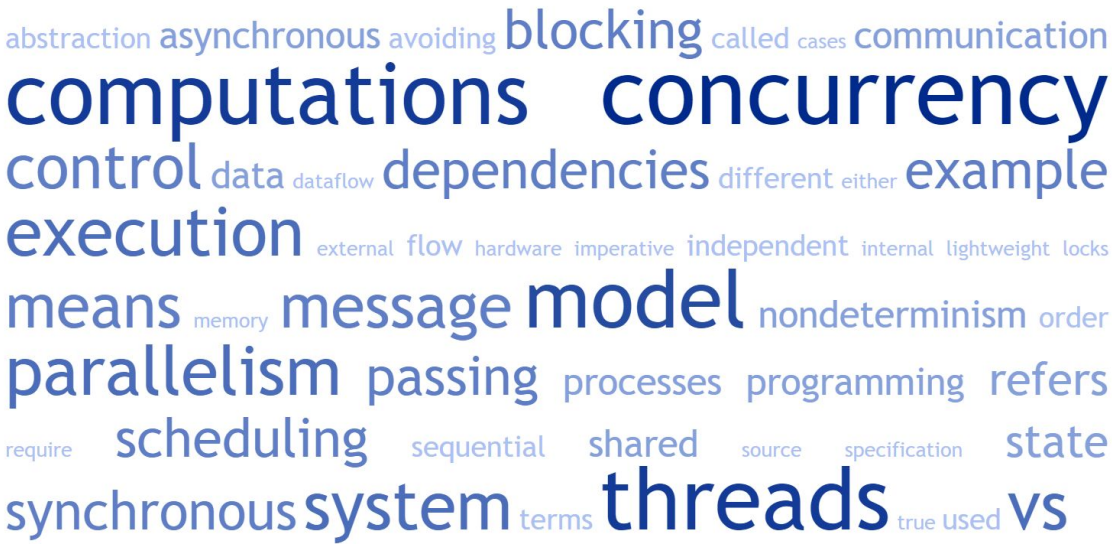
About me



Azure Video Analyzer

Find me at:
linkedin.com/in/**venkatraman24**
github.com/**venkat24**
venkat24@outlook.com

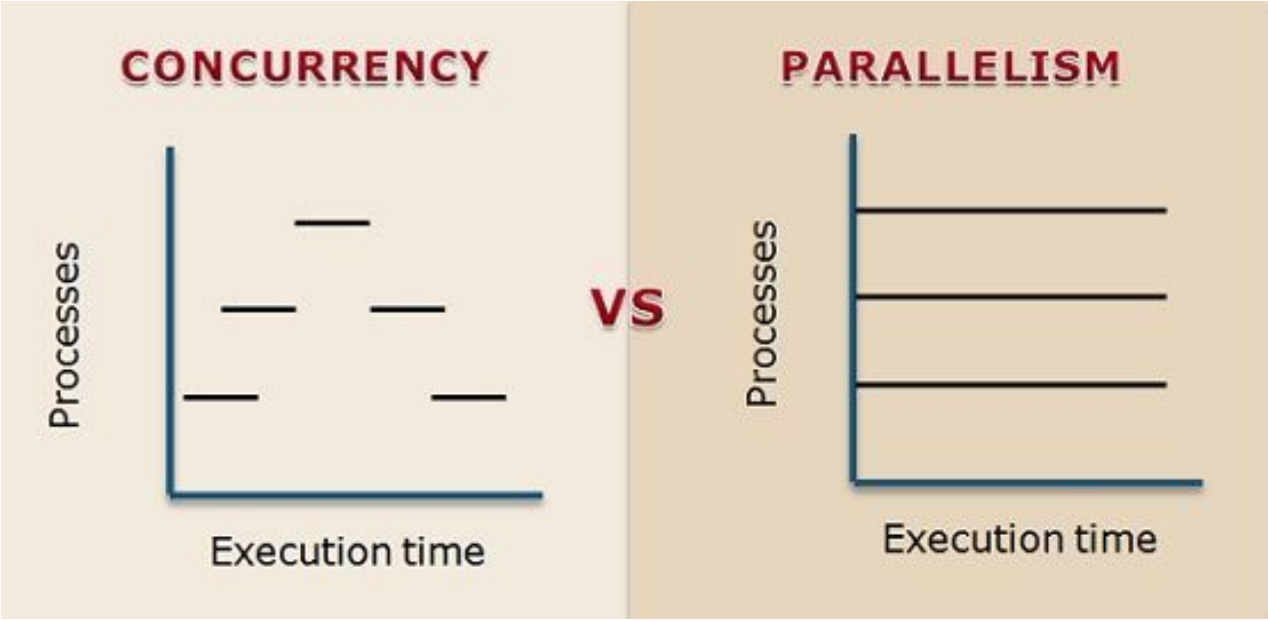
Lots of terminology around concurrency



Concurrency vs Parallelism

- Parallelism
  - Execute several tasks at the same time
  - Parallel execution rather than series
- Concurrency
  - Execute several tasks during overlapping periods of time
  - Concurrent execution rather than sequential

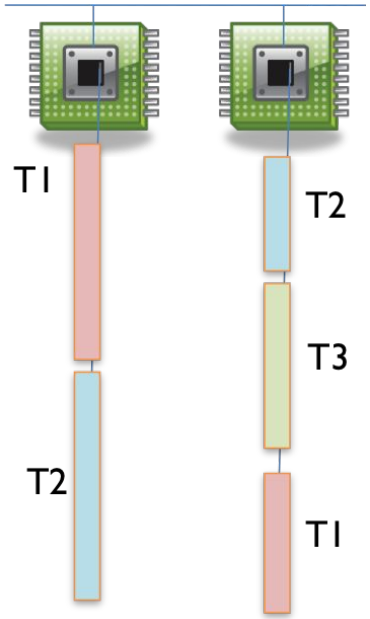
Concurrency vs Parallelism

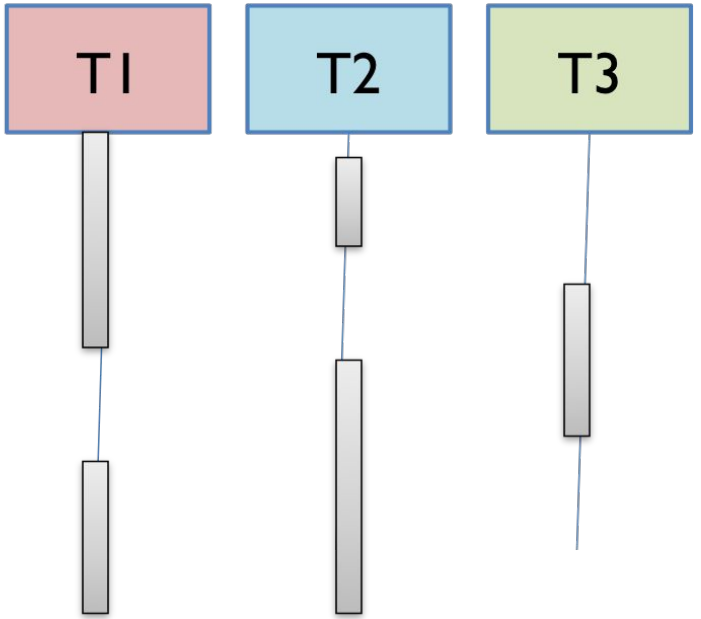


Concurrency vs Parallelism

- Concurrency is about **dealing with** many things at once
- Parallelism is about **actually doing** many things at once

Concurrency + Parallelism





Key Realizations (Goals)

- Concurrency is essential, parallelism is usually a bonus
- Concurrency is about design and structure, while parallelism is mainly about performance
- Concurrent systems naturally lend themselves to be exploited by parallelism

Concurrency vs Parallelism

Can you make a program concurrent without having it be parallel?

Absolutely!

Concurrency without Parallelism

Case in point: JavaScript!

JavaScript execution is single threaded

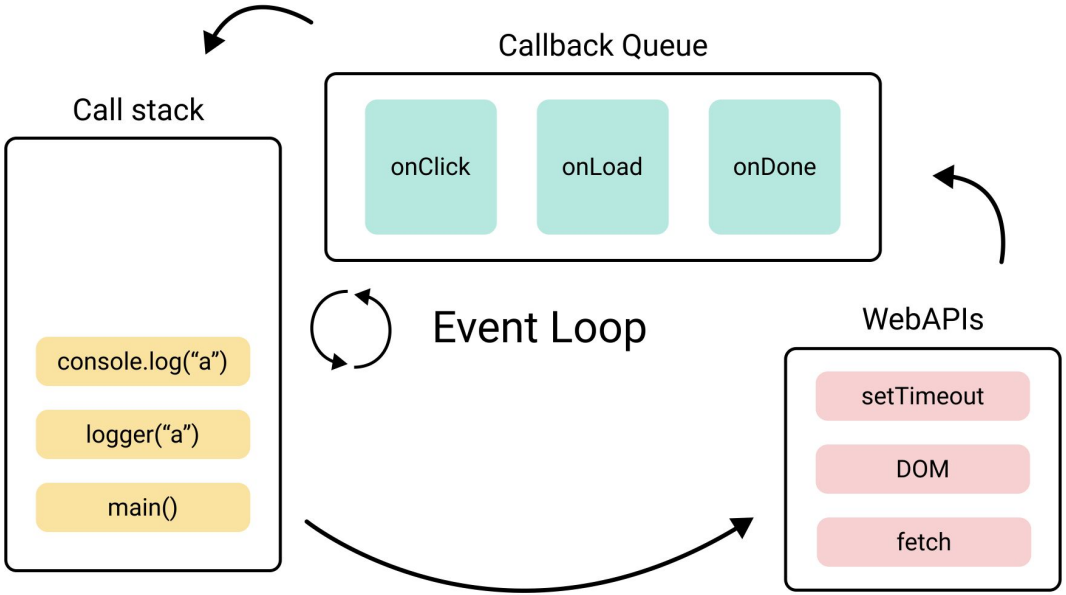
(kinda)

Getting to Async/Await in JavaScript

- Synchronous execution and the JS event loop
- Higher order functions and callbacks
- Promise chaining and error handling
- Callback Hell
- Promises and deferred execution
- Pyramid of Doom
- Async/Await and the illusion of sequential execution

Let’s look at some code!

Concurrency in JavaScript



Concurrency in JS - Key Points

- Async/await and Promise based concurrency in JS is ultimately a system of simple callbacks
- There is not much parallelism in simple JS - most operations happen one after the other. However there is a lot of scope for concurrency
- A lot of concurrent code in JavaScript is I/O bound, and not CPU bound
- If we have CPU bound tasks which might take time and energy to execute, it is best to turn to parallelism

Async/Await and other concurrency models

- Languages offering Async/Await
  - C#
  - JavaScript / TypeScript
  - Python
  - C++ (yes!)
  - Rust
  - Swift
- Goroutines in Go can exploit parallelism while offering very light weight concurrency
- Other languages featuring a message passing concurrency model
  - Rust (also has async await)
  - Scala

Parallelism

- Enabled by multiple levels of abstraction from the hardware, OS, up to the programming language level
- Very closely associated with the concept of threading
- When a program or programming framework supports parallelism, we often call it ‘multithreaded’

Threads - at the OS Level

- Basic idea of threads you may have already encountered:
- Multiple threads of execution may run as part of a single process
- Example: pthread on Linux

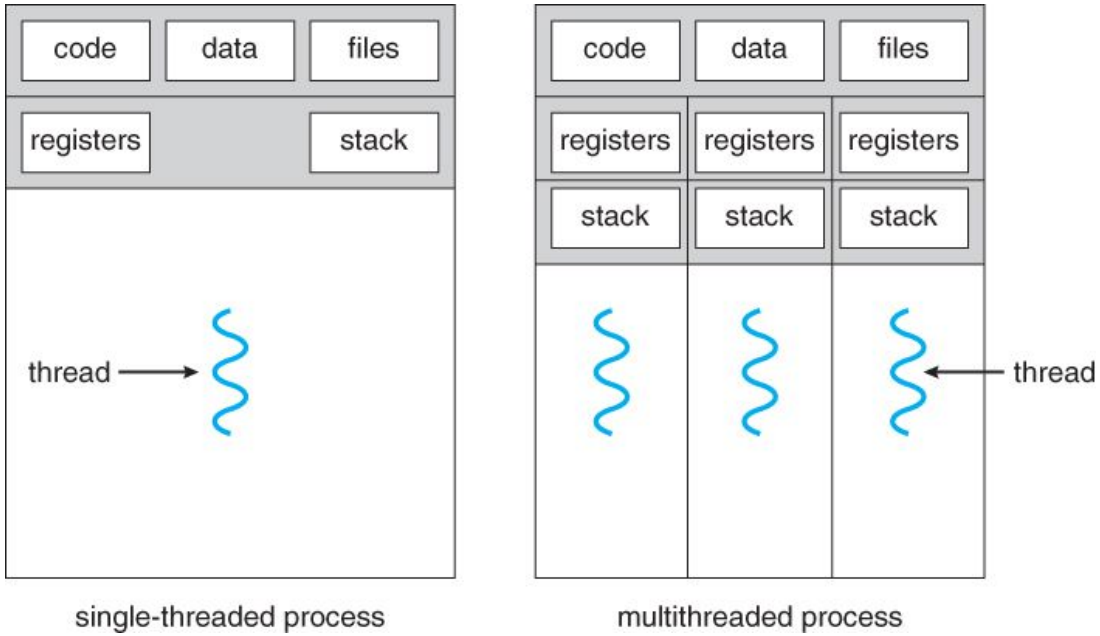
Threads Share:

- Process ID
- Heap memory
- Program code

Threads don't share:

- Same process ID
- Same heap memory
- Same program code

Threads - at the OS Level



Threads - at the Programming language level

- At the programming language level, we can often use threads directly through operating system threading libraries
- Most languages offer some way to create an OS thread
- Example: std::thread in C++, Thread class in Java
- The languages above may be implemented on Linux through pthread behind the scenes

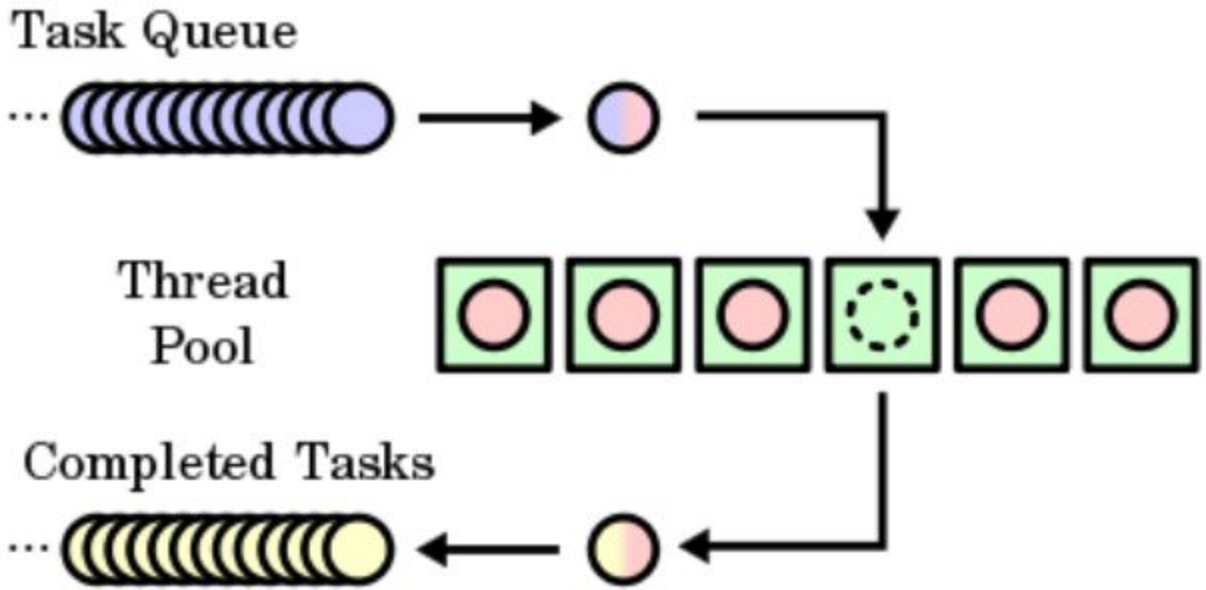
Threads - at the Programming language level

- However, many languages also offer layers of abstraction over threading - often through concurrency mechanisms
- For example: Async/Await in C# can run on a thread pool (instead of just being a substitute for callbacks like in JavaScript)

Languages with async/await

- C#
- JavaScript / TypeScript
- Python
- C++ (yes!)
- Rust
- Swift

Threads - at the Program level



More about Async/Await - C#

- Let’s look more closely at the async/await model in C#
- Almost identical in syntax to JavaScript - but implementation uses real parallelism on a thread pool
- Some names are changed:
- Promise -> Task
- Promise.all -> Task.WhenAll
- Promise.any -> Task.WhenAny
- Promise.then -> Task.ContinueWith

More about Async/Await - C#

- Tasks are strongly types - like everything else in C#
- A simple Task returns nothing, a Task<int> returns an integer
- You can force any piece of code to run in a separate Task by using Task.Run()
- Functions which await Tasks must be async, just like JS

Let’s look at some code!
https://replit.com/teams/join/ttkdfodktslcsekwmrzbeyjdatzorglk-concurrency-talk

Let's write some code!

https://replit.com/teams/join/ttkdfodktslcsekwmrzbeyjdatzorglk-concurrency-talk

Key Realizations (Goals)

- Concurrency is essential, parallelism is usually a bonus
- Concurrency is about design and structure, while parallelism is mainly about performance
- Concurrent systems naturally lend themselves to be exploited by parallelism

References and Sources

https://stackoverflow.com/questions/42412145/what-is-a-state-machine-in-terms-of-javascript-promises-and-c-sharp-asyc-await

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await

https://www.digitalocean.com/community/tutorials/understanding-the-event-loop-callbacks-promises-and-async-await-in-javascript

https://www.youtube.com/watch?v=oV9rvDllKEg

https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/

https://alexyakunin.medium.com/go-vs-c-part-1-goroutines-vs-async-await-ac909c651c11

Thank you!

Find me at:
linkedin.com/in/**venkatraman24**
github.com/**venkat24**
venkat24@outlook.com