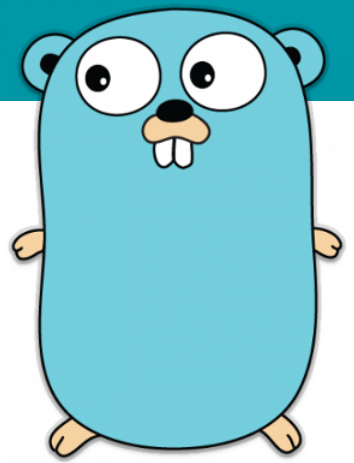
Go



History & Motivation

(https://talks.golang.org/2012/splash.article)

- The goals of the Go project were to eliminate the slowness and clumsiness of software development at Google, and thereby to make the process more productive and scalable

- Go's purpose is therefore not to do research into programming language design; it is to improve the working environment for its designers and their co-workers

- Software at Google (From *Borg, Stubby* .. To ... -> *Kubernetes, Docker, gRPC*)

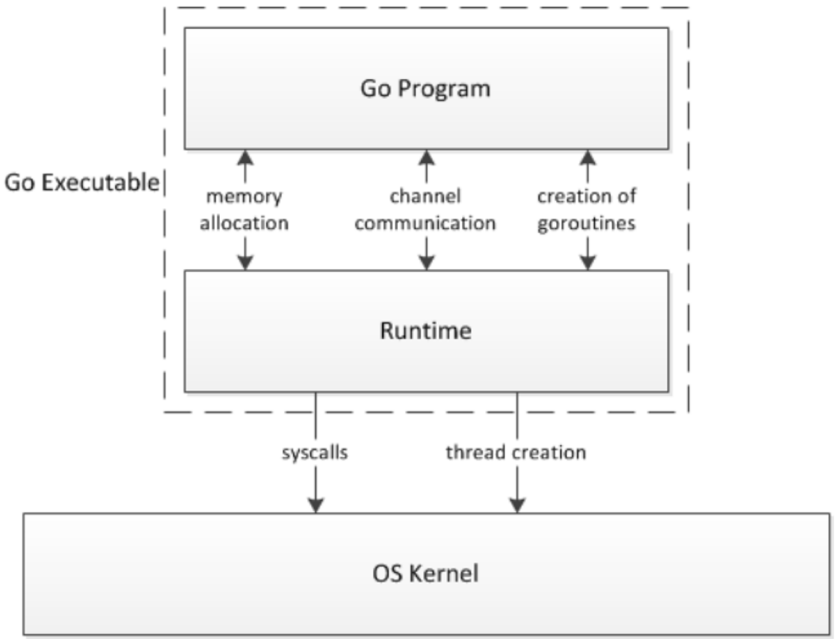
Golang: What it excels at...

- Ease of Concurrency. Highly concurrent light weight go routines. No thread management
- Language designed for productivity and Maintainability
- Ease of Asynchronous operations. Most I/O is async for the runtime however developer has to write simple synchronous style code
- Garbage collected yet natively compiled for efficiency

GoLang: What it excels at...

- New Language designed for productivity and Maintainability
- Faster compile time
- Code portability to cross platform
- Built-in tools for dependency management, code analysis, code documentation
- Composition over inheritance

Golang Runtime

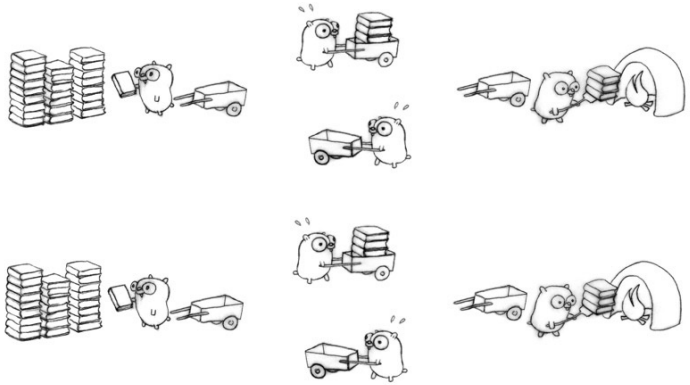


IO can be non blocking for runtime, however the user has to write synchronous code

Go routine (Task or user thread) is lightweight with 8k stack.

All syscalls go via runtime and scheduler can effectively manage concurrency and fuller utilization of cores

Concurrency



Concurrency

A task can be,

- CPU intensive
- IO intensive
- CPU and IO intensive

> **Concurrency** is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't **necessarily** mean they'll ever both be running at the same time
>
> **Parallelism** is when tasks literally run at the same time, e.g. on a multicore processor.

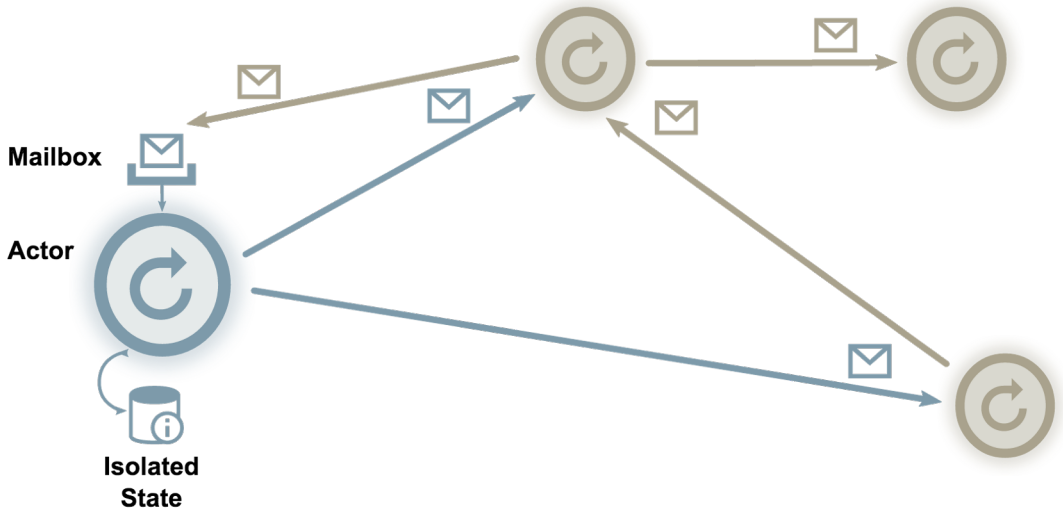
With concurrency => Break long operation into tasks such that you can do something useful while you wait. And such task sequences are parallelizable.

More info: https://blog.golang.org/concurrency-is-not-parallelism

CSP / Actor model

Communicating sequential processes

Principle: Do not communicate by sharing memory; instead, share memory by communicating.



CSP / Actor model

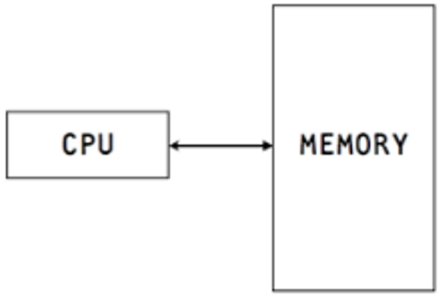
CSP was first described in a 1978 paper by Tony Hoare.

Why CSP or Actor models have become important now?

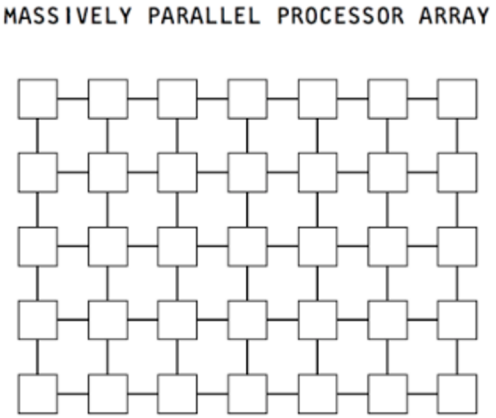
CSP / Actor model

CSP was first described in a 1978 paper by Tony Hoare.

Why CSP or Actor models have become important now?

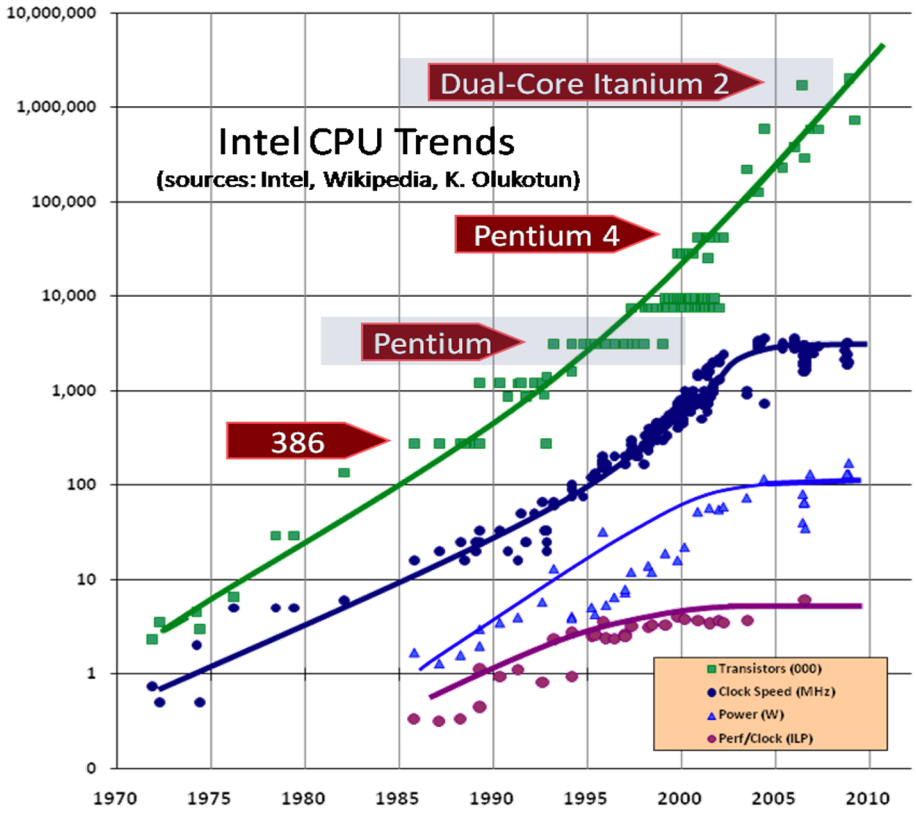


Evolving machine architecture



Multicore

Hardware has become multi core. Free lunch is over: http://www.gotw.ca/publications/concurrency-ddj.htm



Multicore

Sharing is even bigger problem in addition to the context switch.

Locks/mutexes might hinder some or all cores, this is not good for multi core environment.

Sharing is the root of all contention http://herbsutter.com/2009/02/13/effective-concurrency-sharing-is-the-root-of-all-contention/

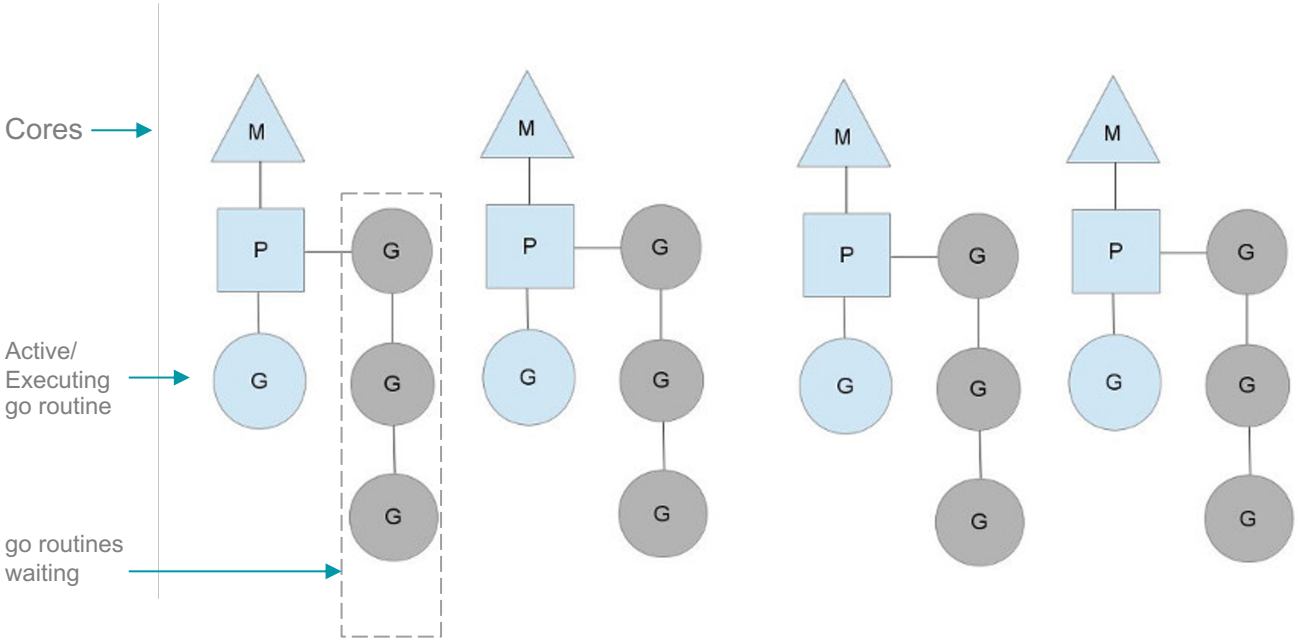
Concurrency without locks (sharing) scales

=> CSP (Erlang processes, Golang “go routines”)

GoLang scheduler

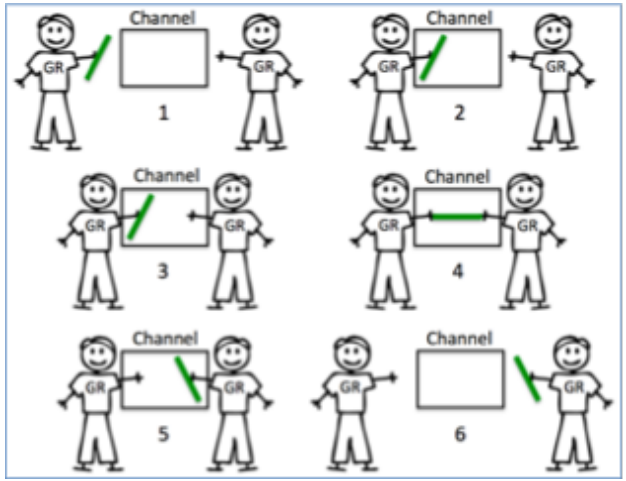
http://morsmachine.dk/go-scheduler

On core **M** the context **P** (virtual proc) executing several goroutines G

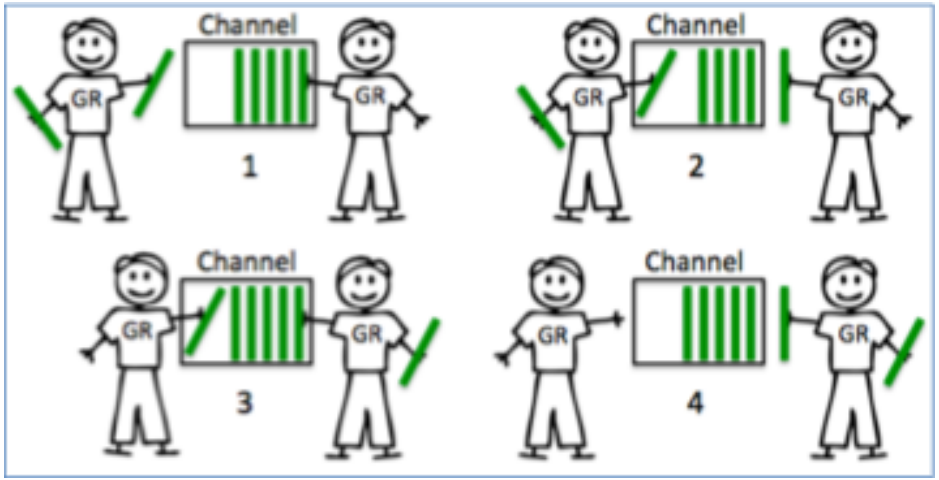


Channels

Unbuffered



Buffered



(https://www.ardanlabs.com/blog/2014/02/the-nature-of-channels-in-go.html)