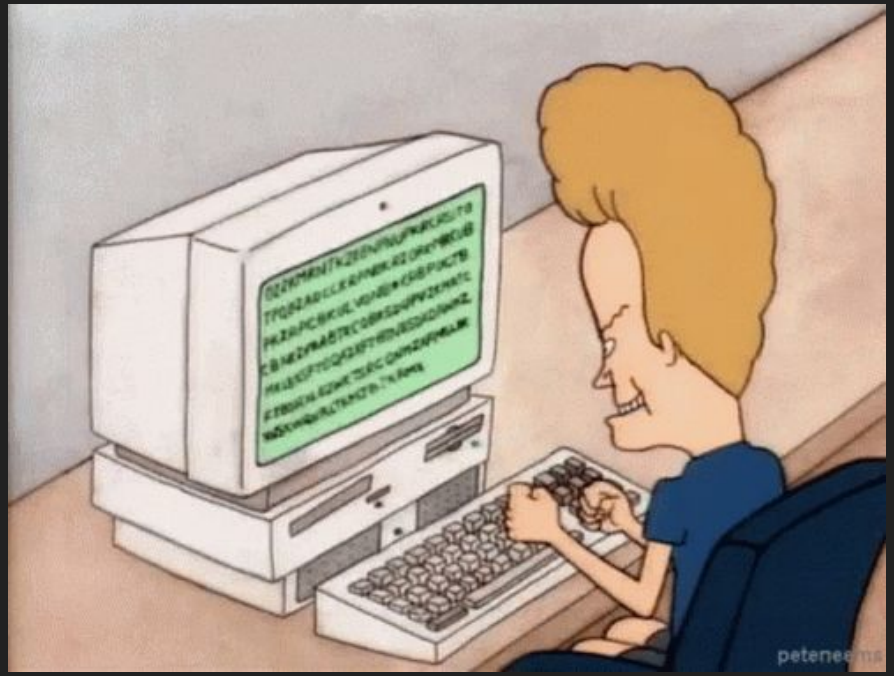
Concurrency, simplified?

(because it’s hard)

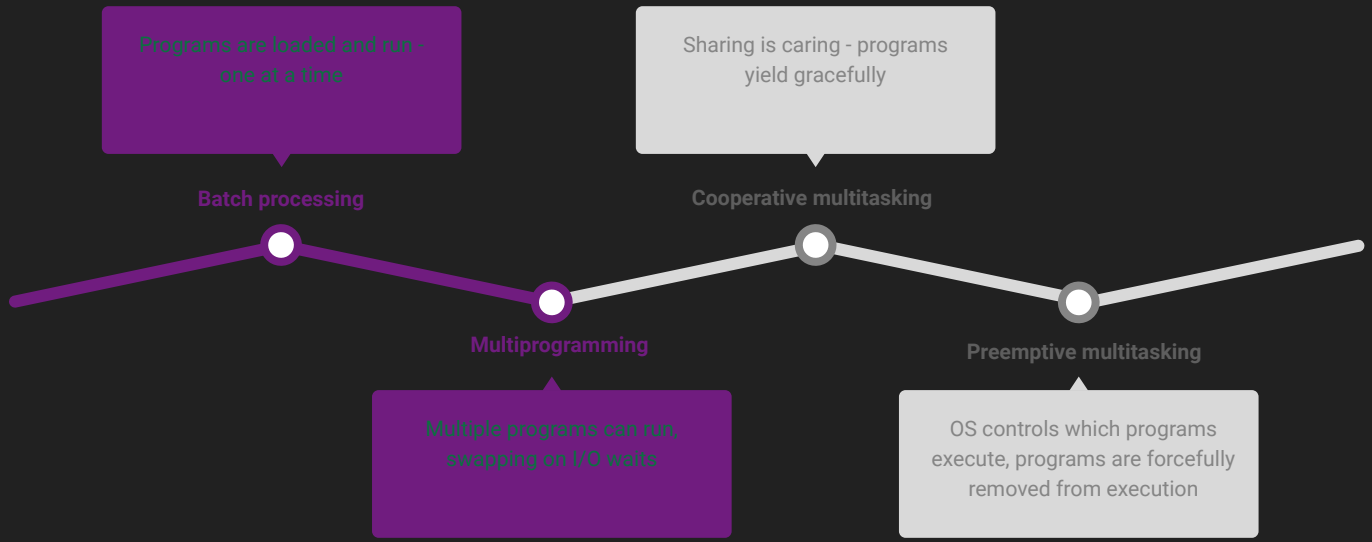
Brian Elliott, Principal Software Engineer, Planet Labs

What is concurrency?

- Plumbing
- Run “processes” independent of one another
- Does not imply parallelism
- Typically people think of OS processes and threads



A little history...



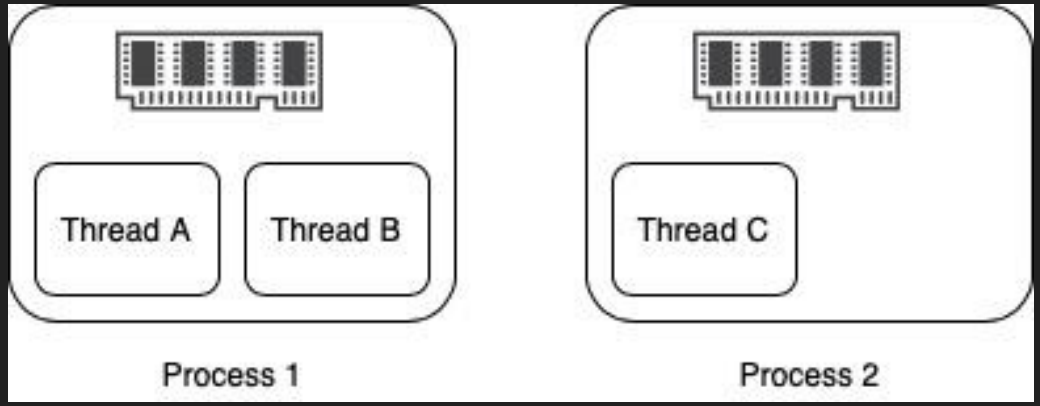
Processes and threads

Process:

- Independently schedule-able program.
- Has a call stack
- Has its own memory
- May contain threads

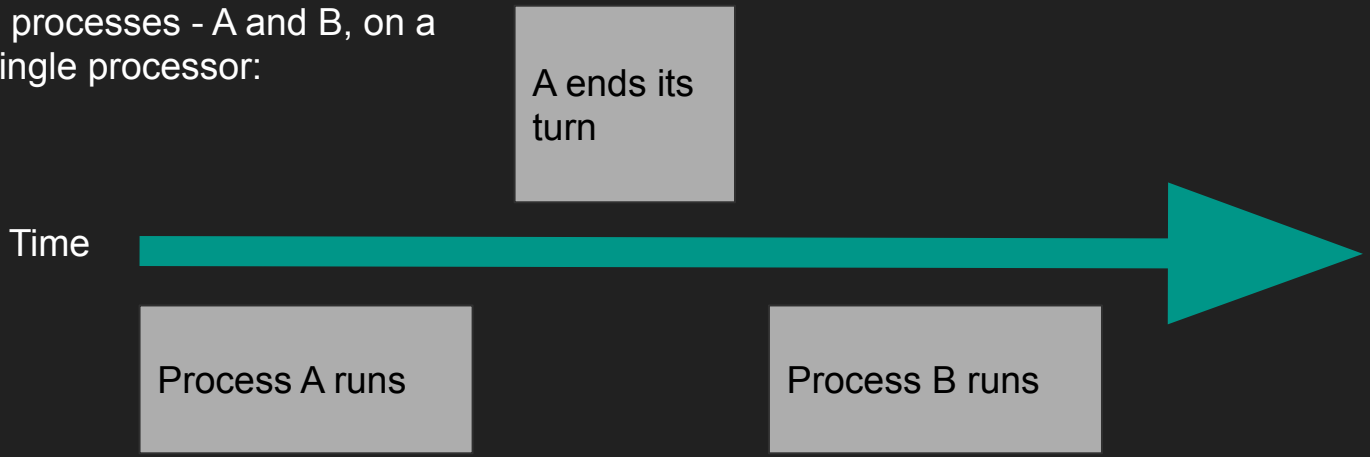
Thread:

- Lives inside a process
- Has a call stack
- OS scheduler may be aware of thread
- **Shares memory** within a process



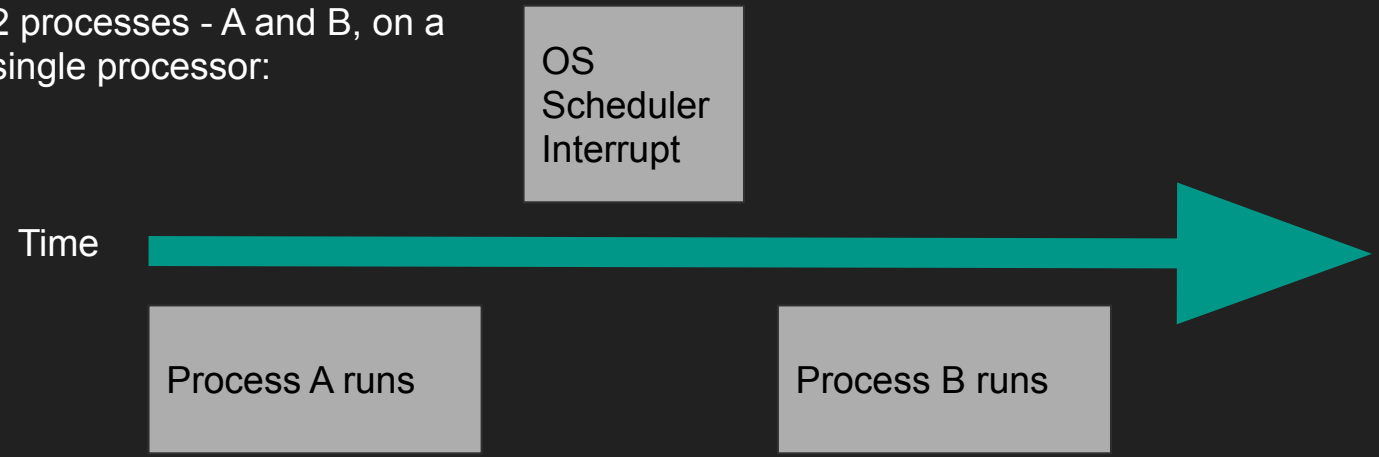
Cooperative Multitasking

2 processes - A and B, on a single processor:



Preemptive Multitasking

2 processes - A and B, on a single processor:



(On a multi-core CPU, both A and B could run simultaneously.)

Single thread example

Contrived example that adds up numbers in individual “buckets”.

See adder.py

Multiple thread example

Modifies previous example to use:

- 1 process
- Multiple threads / workers
- Memory is shared
- Preemptive multitasking

See adder_multi.py

Shall I lock?

Modifies previous example to use:

- Access to “critical section” is protected by a lock.
- Data integrity is maintained.
- What’s the cost?
- Does this scale?

See adder_lock.py

Form an orderly queue

Modifies previous example to:

- Use a lock-less approach.
- Shared memory is removed.
- A queue is used to share data.
- What's the cost?
- Does this scale?

See adder_queue.py

Gotchas

Even in the previous example, locking is still happening!

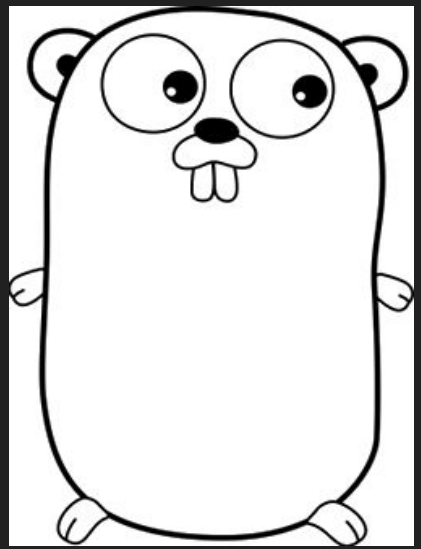
- Global Interpreter Lock (Python interpreter lock)
- One thread at a time, per interpreter!
- Basically, the languages can’t use multiple cores effectively.
- There is an abomination called “multiprocessing” in the Python world.

(Ruby does this too.)

(JS is single threaded and avoids all of this.)

(Compiled languages don’t have this “feature”.)

Go is the new hotness



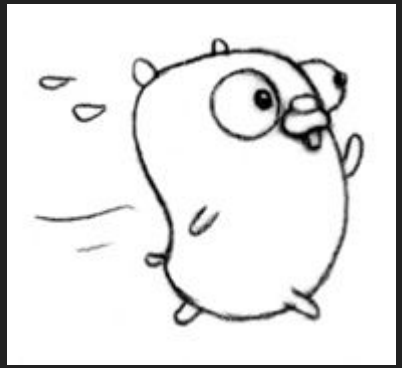
- Fast
- Garbage collected!
- Concurrency primitives baked in
- Scales to many developers
- Static compilation, tiny Docker containers

Goroutines

```
fib(n int) int {

        ...

}

go  fib(n)
```

- Concurrent running entity.
- Thead-ish.
- Cooperatively scheduled!
- Garbage collected!
- User-space scheduler

Go Basics

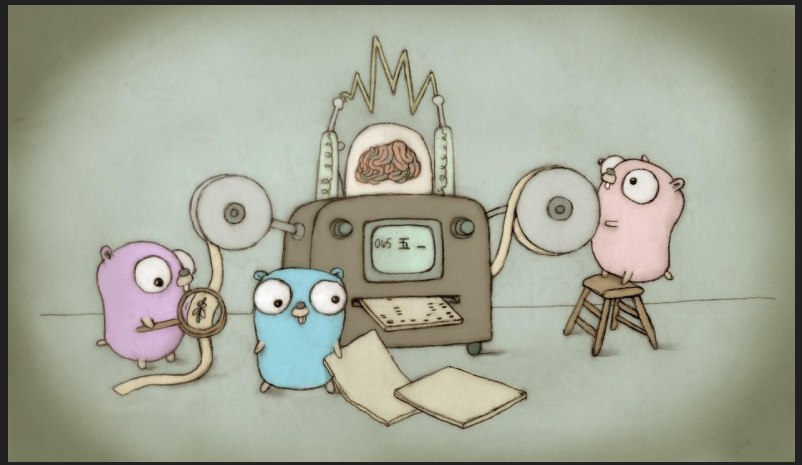


Basic Go version of single threaded adder

See single.go

- Gopher art: https://blog.golang.org/gopher

+ Goroutines

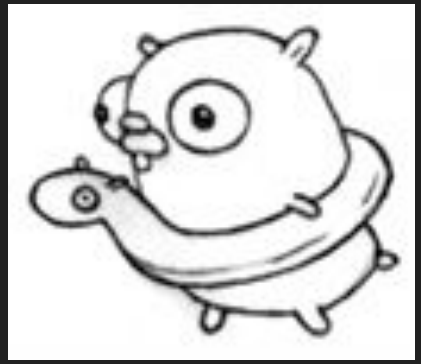


Modified version utilizing goroutines.

(Still not too smart.)

See multi.go

+ Locks

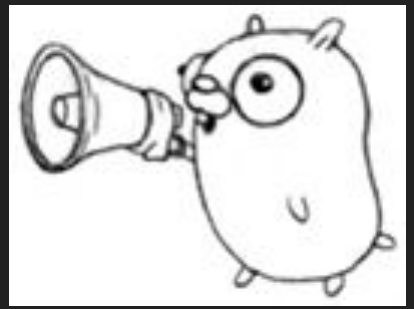


Modified version utilizing goroutines and locks.

(Works, not ideal.)

See lock.go

Channels



Modified version utilizing goroutines and channels.

- Go queue equivalent
- Fast/safe
- No shared memory

See channels.go

Something more fun

- Proxy Dad jokes
- Parallelize with goroutines and channels
- Observe built-in muscles of net/http server

See go/cmd/joke/*

net/http Server

```
ctx := context.WithValue(baseCtx, ServerContextKey, srv)
for {
    rw, e := l.Accept()
    if e != nil {...}
    if cc := srv.ConnContext; cc != nil {...}
    tempDelay = 0
    c := srv.newConn(rw)
    c.setState(c.rwc, StateNew) // before Serve can return
    go c.serve(ctx)
}
```

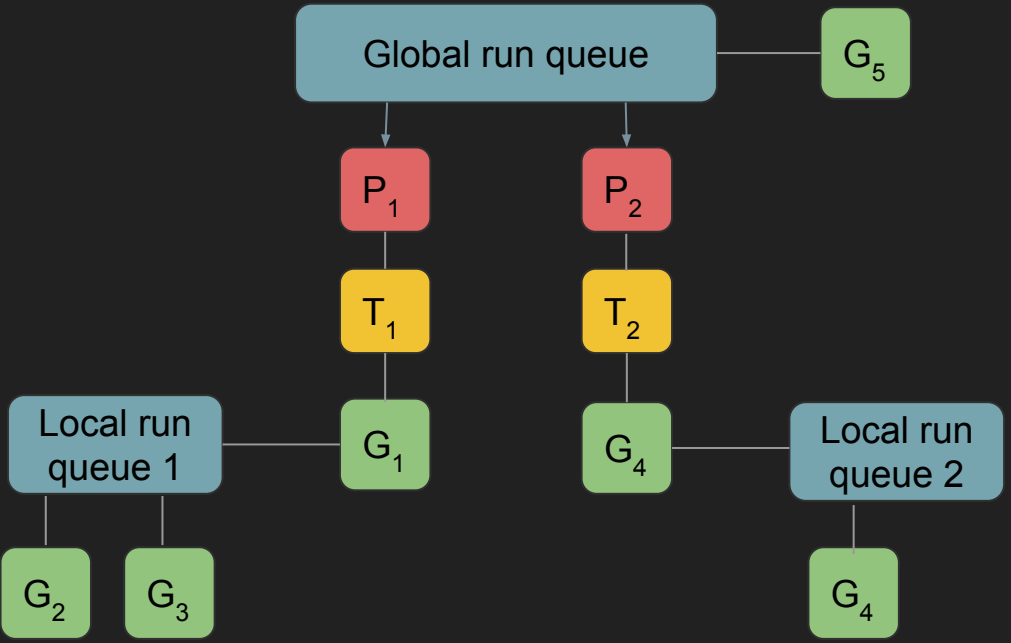
Go scheduler

The Go scheduler augments the OS-level scheduler.

- Runs in user-space
- Goroutines are assigned to a thread / processor
- Scheduler manages which goroutine runs
- Cooperative scheduling - goroutines on syscalls, etc.
- Optimized to avoid OS context switches (thread/process swaps)
- Network I/O poller
- Local run queue / Global run queue
- Garbage collection

Go scheduler, cont'd

P = CPU
T = OS Thread
G = Goroutine



Scale - how high?

- Goroutines are very cheap - a little bit of stack
- Scheduler intelligently switches on blocked goroutines
- Still effectively limited by number of CPU cores, but there’s no artificial blocking due to language constraints.

Some ramblings.

- Aren’t microservices the answer to everything?
- Not so simple once you take into account fault tolerance, rate limiting, versioning, etc.
- I can queue in SQS/Google Pubsub/Rabbit/etc. Yes, you can. And you will pay for the cloud resources.
- Does it solve all the problems? Nope, you still need to be an engineer, but it’s a nice hammer.

Code samples

https://github.com/bdelliott/concurrency-intro

Questions?

What is the airspeed velocity of an unladen swallow?

bdelliott@protonmail.com

Scan Me