Goroutines demystified.

Mathieu Nativel

goroutines??

A goroutine is a lightweight thread managed by the Go runtime.

goroutines ??

> *A goroutine is a lightweight thread managed by the Go runtime.*

They are called lightweight threads because they require less processing time :

- Smaller default stack size
- Lighter context switching : setup and teardown don't require call to the kernel.

Ok so basically they are just user threads : an implementation of threads and scheduling running on top of the OS.

Ok so basically they are just user threads : an implementation of threads and scheduling running on top of the OS.

But wait how is this a good idea ? Why is it cool to reimplement something already provided by your kernel ?

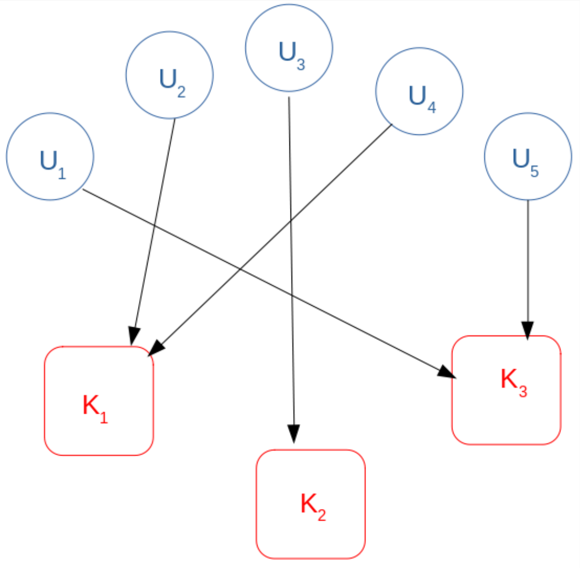
User-threads ideas

Some nice ideas to make profit of user threads :

- Allocate kernel threads when creating the first user threads.
- Park for reuse the kernel threads after the user thread ends.
- Schedule user threads to run on respecting kernel threads.

User-threads implementation

Multiplexing *low-cost* user-threads on *high-cost* kernel threads :



User-threads implementating

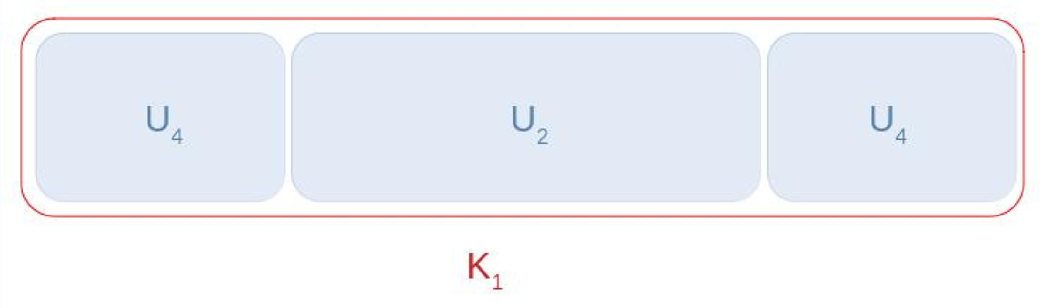


Figure 2: Multiple user thread can run on the same associated kernel thread.

The scheduler manages a runqueue of runnable goroutines.

When it wants to schedule a goroutine it pops it out of the runqueue and schedules it on a available kernel thread (instantiating one if needed and possible).

Scheduling concepts

The scheduler manages a runqueue of runnable goroutines **and** per-core runqueues of runnable goroutines.

Work-stealing scheduling: When it wants to schedule a goroutine on a kernel thread **it first try to pop it from the local runqueue**, if failed **it tries to steal it** from other local runqueues and finally it tries the global runqueue.

Go implements theses concepts through 3 important structures in the runtime code :

The G struct (Goroutine)	The M struct (Kernel Thread)	The P struct (Linked List)
- Represents a runnable goroutine - Contains informations about its stack its current status and its associated P.	- Represents a kernel thread - Contains two important pointers: one to the currently running G and another one to its attached P.	- Represents a scheduling context - Contains a list of runnable Gs.

The G struct

From the go runtime source code : https://golang.org/src/runtime/runtime2.go

```
type g struct {
  stack       stack   // offset known to runtime/cgo
      ...
  param          unsafe.Pointer // passed parameter on wakeup
  atomicstatus   uint32
  goid           int64
  schedlink      guintptr
  waitsince      int64      // approx time when the g become blocked
  waitreason     waitReason // if status==Gwaiting
      ...
  tracelastp     puintptr   // last P emitted an event for this goroutine
    };
```

The M struct

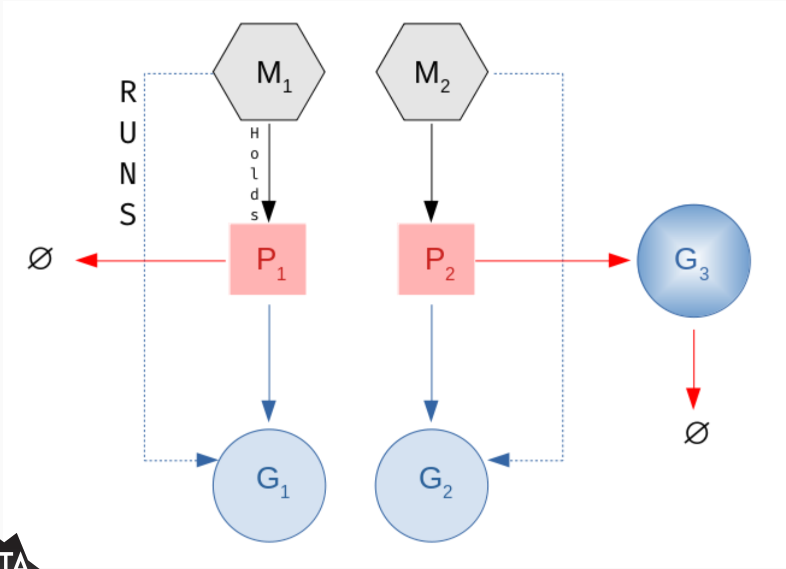
```go
type m struct {
      ...
  curg          *g       // current running goroutine
  caughtsig     guintptr // goroutine running during fatal signal
  p             puintptr // attached p for executing go code (nil if not executing go code)
  nextp         puintptr
  oldp          puintptr // the p that was attached before executing a syscall
  id            int64
  spinning      bool // m is out of work and is actively looking for work
      ...
      };
```

go

The P struct

```go
type p struct {
      id          int32
  status      uint32 // one of pidle/prunning/...
  m           muintptr   // back-link to associated m (nil if idle)
  runqhead uint32
  runqtail uint32
  runq     [256]guintptr
  // runnext, if non-nil, is a runnable G that was ready'd by
  // the current G and should be run next instead of what's in
  // runq if there's time remaining in the running G's time
  // slice. It will inherit the time left in the current time
  // slice ...
  runnext guintptr
      ...
  };
```

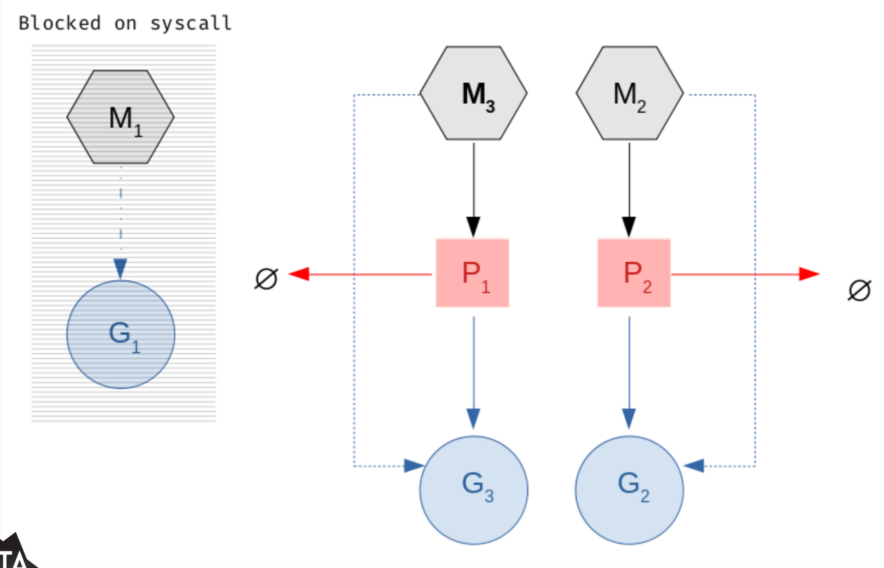
Maybe with a drawing ?



Why would we separe M from G?

The idea is that when a M (a kernel thread) is blocked (on a syscall for exemple), the go scheduler can take its runqueue *(its P)* and give it to an other M.

What happens when blocking



Questions ?

— author: LSE title: Presentation ...