- Scheduler runs go routines

- Pauses/resumes when goroutine is blocked on a channel or mutex operation
- Co-ordinates blocking system calls (file I/O)
- Takes care of runtime tasks
  - ↳ Garbage Collection

go routines are USER-Space threads

- Not kernel OS threads
  - ↳ similar though
  - ↳ Lighter + faster to create + smaller mem. footprint
  - + easier to switch context

- OS only knows how to schedule threads on hardware (CPU)

go routines are managed by go runtime

---

go schedule puts go routines on the kernel threads!

---

go scheduler wants to utilise hardware parallelism

ie Scale upto more cores!

work stealing ; Contention

Co-operative premption

=> sys mon unschedules long running go-routines when possible

↳ Background monitor thread

↳ Moves them to a global run queue

↳ A Lower priority runque checked less frequently than local thread run queue

Thread Spinning

- Threads without work "spin" and look for work in global runqueue, poll the network and look at other runqueues to perform work stealing
- This uses up more CPU cycles, but leverages parallelism

Runqueues

- Run queues for cores are stored in a heap allocated struct `p`

  ↳ Also stores resources the thread needs to run goroutines
  => like a memory cache.

- A thread will claim a runqueue to run goroutines, sysmon will handover when thread is blocked.

Limitations of go scheduler

- FIFO runqueues ⇒ no way to prioritise goroutines
  - ↳ Not like linux scheduler which uses priority queues.

- No strong preemption
  - ↳ No strong fairness/latency guarantee

- Scheduler is not aware of hardware topology
  - ↳ Use LIFO runqueues for better cache utilisation?