Lecture 2 RPC & Threads

- C++ vs Go
  - C++: non-GC, requires book keeping to keep a track of threads & free objects, error messages are a nightmare
  - Go: typesafe, memory safe, GC, straightforward — "defective Go"

- Threads
  - → manage I/O concurrency { one program talks to many diff servers }
  - → "go routines"

1 Thread — 1 PC, 1 stack, 1 address space

← 1 address space
diff threads could refer other threads' stacks

→ I/O Concurrency : Can have one program launch RPC's to many servers & wait for replies.

→ Parallelism : One program can use multiple cores of CPU, truly in parallel
multicore

→ Convenience : sleep → periodic thing → sleep

★ event-driven : single thread in a single loop sitting and waiting for any kind of input.

- Threads vs process
  - → inside a process → multiple threads
  - → OS keeps processes separate.
  - → one go program runs one UNIX process which then all go routines are inside that process

- Thread challenges
  - → sharing memory (same address space) but very easy to get bugs

`n = n + 1` → multiple threads might be executing the same code

"RACE"

solution: locks (mutex in go)

`mu.Lock()`, `mu.Unlock()`

- → should locks be private?
  - not always a good idea → embedded inside DS methods not known to programmer
  - might not want to use locks at all?

→ Problems with Threads

① Race conditions

② Co-ordination: when locking, diff threads dont know others exist, but we might intentionally want diff threads to interact

- "channels" in go to send data between threads
- sync.cond :- condition variables (if other thread is waiting for you, give it a kick)
- wait group: launching an unknown no. of routines & waiting for them to finish
- deadlock: T1 ⟶ T2 ⟲

→ Web crawler demo

- avoid cycles, not fetch a page twice, fetch pages in parallel
- final challenge: knowing when to stop the crawl?
- Serial Crawler
  - DFS on web graph, keeps mapped (set) to remember what it fetched.
    ↳ passed by ref and not copy
    (in go all calls share pointer to obj in memory)

- Concurrent Crawler
  - shared data objects & locks
  - Non-shared but synced by channels