LESSON 10

Go

main.go

```
select {
case v := <-ch:
case <-time.After(time.Second):
}
```

select · mutex · ctx

Learn it all @MyWebUniversity



A FREE MYWEBUNIVERSITY COURSE

GO PROGRAMMING

LESSON 10

select, Mutexes and context

select with timeouts, sync.Mutex, the race detector and context cancellation

select

select.go

```
select {
case msg := <-messages:
    fmt.Println("got", msg)
case err := <-errs:
    fmt.Println("error", err)
case <-time.After(2 * time.Second):
    fmt.Println("timeout")
}
```

- select waits on several channel operations at once and runs whichever is ready first. It is …
- Add a case with time dot After to set a timeout, so your program never waits forever.

01

Data Races

terminal

```
$ go run -race counter.go
==================
WARNING: DATA RACE
Write at 0x00c000012345 by goroutine 8:
  main.main.func1()  counter.go:12
Previous write by goroutine 7:
```

- When two goroutines change the same variable at the same time, you get a data race and ...
- Run your program or tests with the dash race flag and Go detects races and shows exactly ...

02

sync.Mutex

counter.go

```
type Counter struct {
    mu sync.Mutex
    n  map[string]int
}
func (c *Counter) Inc(key string) {
    c.mu.Lock()
    defer c.mu.Unlock()
    c.n[key]++
}
```

- A mutex protects shared data: Lock before touching it and Unlock after. Only one goroutine …
- Put the mutex next to the data it protects, and use defer Unlock so you never forget. For …

03

context: Cancellation and Deadlines

context.go

```go
ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
defer cancel()

req, _ := http.NewRequestWithContext(ctx, "GET", url, nil)
resp, err := http.DefaultClient.Do(req)
if errors.Is(err, context.DeadlineExceeded) {
    fmt.Println("too slow, gave up")
}
```

- A context carries a deadline or cancel signal through your program. Pass it as the first …
- When the user cancels or time runs out, ctx dot Done closes, and every goroutine listening …

04

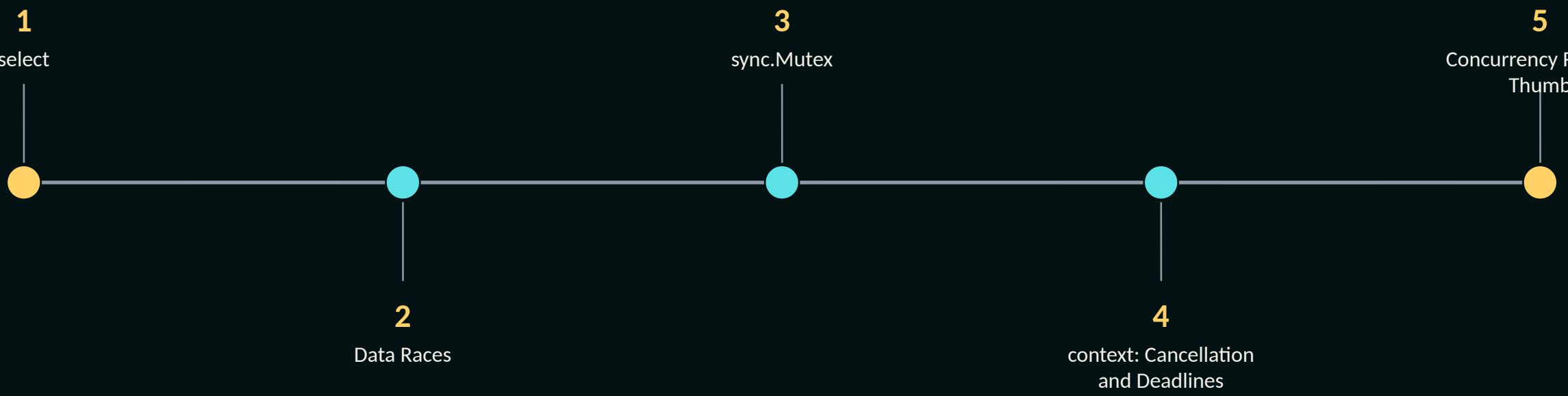
Concurrency Rules of Thumb

Rule	Why
every goroutine can stop	no leaks
channels pass data	one owner at a time
mutex guards state	no data races
`go test -race`	catch races early
ctx first parameter	cancel everywhere

- Know how every goroutine will stop, or it leaks. Use channels to pass ownership of data, …
- Always test concurrent code with the race detector, and add context to anything that does …

05

Lesson Map


1
select
2
Data Races
3
sync.Mutex
4
context: Cancellation and Deadlines
5
Concurrency Rules of Thumb

Sources and Further Reading





Thank you for watching

MyWebUniversity.com

Free technical education since 2000 — English and Dari

@MyWebUniversity · @MyWebUniversity-Dari