Module 4: Threads in Go
Topic 3.2: Deadlock

Synchronization Dependencies

- Synchronization causes the execution of different goroutines to depend on each other

G1	G2
`ch <- 1`	`x := <- ch`
`mut.Unlock()`	`mut.Lock()`

- G2 cannot continue until G1 does something

Deadlock

- **Circular dependencies** cause all involved goroutines to block
  - G1 waits for G2
  - G2 waits for G1
- Can be caused by waiting on channels

Deadlock Example

```
func dostuff(c1 chan int,
             c2 chan int) {
   <- c1
   c2 <- 1
   wg.Done()
}
```

- Read from first channel
  - Wait for write onto first channel
- Write to second channel
  - Wait for read from second channel

Deadlock Example cont.

```
func main() {
    ch1 := make(chan int)
    ch2 := make(chan int)
    wg.Add(2)
    go dostuff(ch1, ch2)
    go dostuff(ch2, ch1)
    wg.Wait()
}
```

- `dostuff()` argument order is swapped
- Each goroutine blocked on channel read

Deadlock Detection

- Golang runtime automatically detects when all goroutines are deadlocked

```
fatal error: all goroutines are asleep - deadlock!
goroutine 1 [semacquire]:
sync.runtime_Semacquire(0x173e2c, 0x1042ff98)
…
```

- Cannot detect when a subset of goroutines are deadlocked