Module 4: Threads in Go
Topic 2.1: Mutual Exclusion

Goroutines Sharing Variables

- Sharing variables concurrently can cause problems
- Two goroutines writing to a shared variable can interfere with each other

Concurrency-Safe

- Function can be invoked concurrently without interfering with other goroutines

Variable Sharing Example

```
var i int = 0
var wg sync.WaitGroup
func inc() {
    i = i + 1
    wg.Done()}
func main() {
wg.Add(2)
    go inc()
    go inc()
    wg.Wait()
    fmt.Println(i)
}
```

- Two goroutine write to i
- i should equal 2

Possible Interleavings

- Seems like there is no problem

Task 1	Task 2	i
		0
`i= i + 1`		
		1
	`i= i + 1`	
		2

Task 1	Task 2	i
		0
	`i= i + 1`	
		1
`i= i + 1`		
		2

Granularity of Concurrency

- Concurrency is at the machine code level
- i = i + 1 might be three machine instructions

```
read i
increment
write i
```

- Interleaving machine instructions causes unexpected problems

Interleaving Machine Instructions

- Both tasks read 0 for i value

	Task 1	Task 2	i
			0
1:	`read i`		
2:		`read i`	
3:	`inc`		
4:	`write i`		
5:			1
6:		`inc`	
7:		`write i`	
8:			**1**