Race Conditions

Two go routines produce a race condition if the result depends on the order in which instructions of the two tasks are interleaved by the go runtime scheduler. For example see the following go code:

```
package main


import (
    "fmt"
    "time"
)


var sharedVar int


func main() {
    go func() {
        sharedVar = 0
        time.Sleep(20000)
        sharedVar++
    }()
    go func() {
        time.Sleep(20000)
        fmt.Println(sharedVar)
    }()
    time.Sleep(2000000)
}
```

There are two go routines in the above code. The first one sets the global variable (sharedVar) to 0 in one instruction and increments it in another instruction (with a sleep in between). The second go routine prints sharedVar (after a sleep). You can see that the value printed out depends on the order in which the instructions of the two go routines are executed. It prints 0, if the print instruction of the 2nd go routine got executed before the increment instruction of the first go routine, and it prints 1 otherwise.