Consider the following two goroutines, which represent two transactions on a joint bank account:

/ Alice:

go func() {

bank.Deposit(200) // A1

fmt.Println("=", bank.Balance()) // A2

}()

// Bob:

go bank.Deposit(100) // B

Alice deposits $200, then checks her balance, while Bob deposits $100. Since the steps A1 and A2 occur concurrently with B, we cannot predict the order in which they happen.

Intuitively, it might seem that there are only three possible orderings:

1. "Alice first"
2. "Bob first"
3. "Alice/Bob/Alice"

In all cases the final balance is $300. The only variation is whether Alice's balance slip includes Bob's transaction or not, but the customers are satisfied either way.

However, this intuition is wrong. There is a fourth possible outcome, in which Bob's deposit occurs in the middle of Alice's deposit, after the balance has been read (balance + amount) but before it has been updated (balance = ...), causing Bob's transaction to disappear. This is because Alice's deposit operation A1 is really a sequence of two operations, a read and a write; call them A1r and A1w. The following is the problematic interleaving:

After A1r, the expression balance + amount evaluates to 200, so this is the value written during A1w, despite the intervening deposit.

This program contains a particular kind of race condition called as Data Race.

**RACE CONDITION:**

A race condition is a situation in which the program does not give the correct result for some interleaving of the operations of multiple goroutines. Race conditions are harmful because they may remain latent in a program and appear infrequently, perhaps only under heavy load or when using certain compilers, platforms, or architectures. This makes them hard to reproduce and diagnose.

**When does Race Condition Occur?**

 A **race condition occurs** when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data.