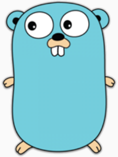
Programming in Go

Matt Holiday
Christmas 2020



What is Concurrency?

Some definitions of concurrency

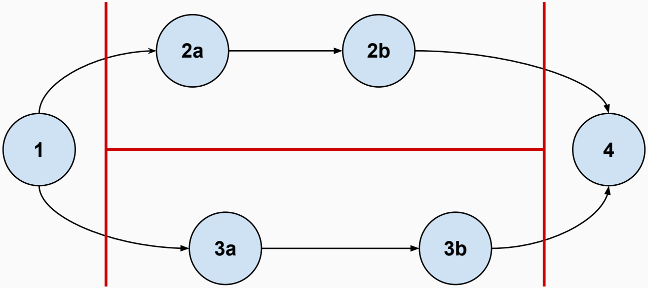
“Execution happens in some non-deterministic order”

“Undefined out-of-order execution”

“Non-sequential execution”

“Parts of a program execute out-of-order or in partial order”

Partial order



- part 1 happens before parts of 2 or 3
- both 2 and 3 complete before part 4
- the parts of 2 and 3 are ordered among themselves

Non-deterministic

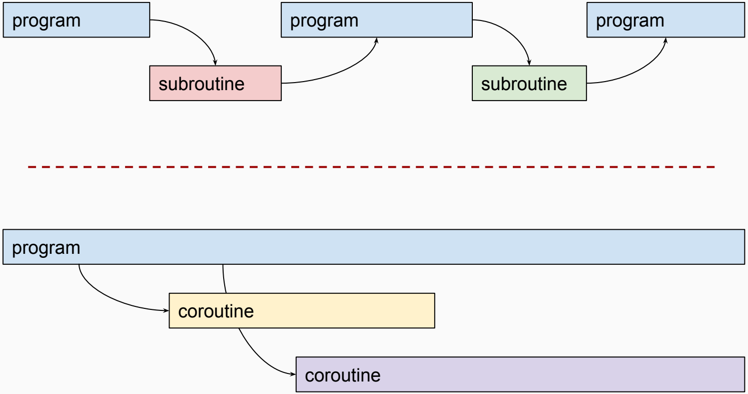
“Different behaviors on different runs, even with the same input”

1. {1, 2a, 2b, 3a, 3b, 4}
2. {1, 2a, 3a, 2b, 3b, 4}
3. {1, 2a, 3a, 3b, 2b, 4}
4. {1, 3a, 3b, 2a, 2b, 4}
5. {1, 3a, 2a, 3b, 2b, 4}

Note we don’t necessarily mean different results, but a different trace of execution

Execute independently

Subroutines are subordinate, while coroutines are co-equal



Concurrency

So let’s try a new definition of concurrency:

Parts of the program may execute independently in some non-deterministic (partial) order

Parallelism

Parts of a program execute independently at the same time

You can have concurrency with a single-core processor
(think interrupt handling in the operating system)

Parallelism can happen only on a multi-core processor

Concurrency doesn’t make the program faster, parallelism does

Concurrency vs Parallelism

Concurrency is about *dealing with* things happening out-of-order

Parallelism is about things actually happening *at the same time*

A single program won't have parallelism without concurrency

We need concurrency to allow parts of the program to execute independently

And that's where the fun begins . . .

Race condition

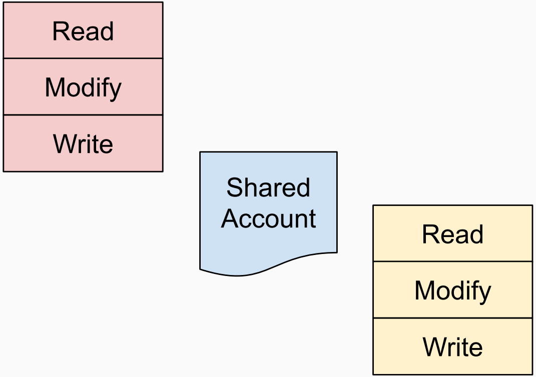
“System behavior depends on the *(non-determistic)* sequence or timing of parts of the program executing independently, where some possible behaviors *(orders of execution)* produce invalid results”

What if interleaving parts of 2 and 3 is wrong?

1. {1, 2a, 2b, 3a, 3b, 4}
2. ~~{1, 2a, 3a, 2b, 3b, 4}~~
3. ~~{1, 2a, 3a, 3b, 2b, 4}~~
4. {1, 3a, 3b, 2a, 2b, 4}
5. ~~{1, 3a, 2a, 3b, 2b, 4}~~

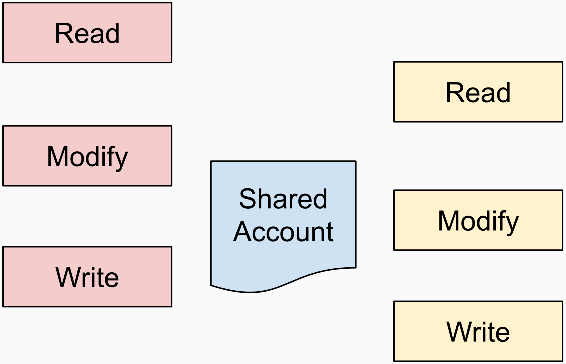
Race condition example

Two deposits to a bank account



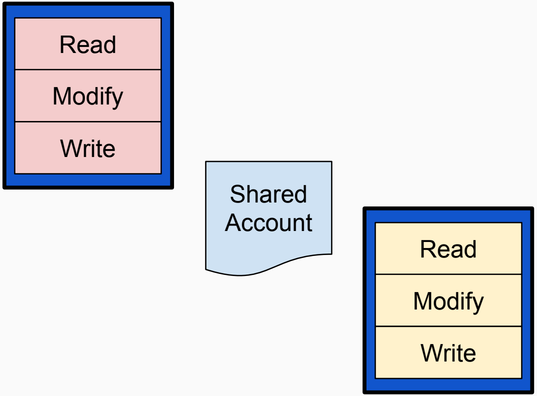
Race condition example fails

Parts of the two deposits are interleaved

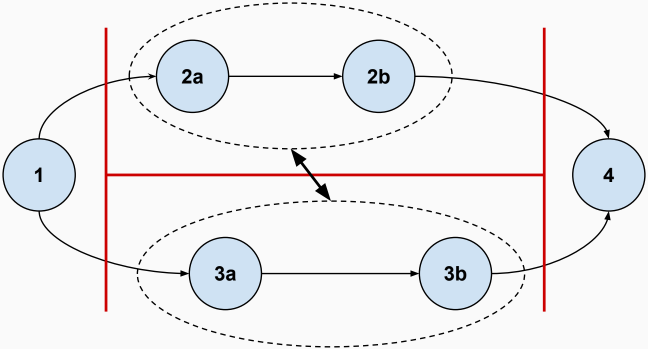


Race condition example fixed

We must actively prevent the parts interleaving



Race condition example fixed



Some ways to solve race conditions

Race conditions involve independent parts of the program changing things that are shared

Solutions making sure operations produce a consistent state to any shared data

- don’t share anything
- make the shared things read-only
- allow only one writer to the shared things
- make the read-modify-write operations atomic

In the last case, we’re adding more (sequential) order to our operations