Concurrency

Michael VanSickle

@vansimke

Introduction

Concurrency in Go

Goroutines

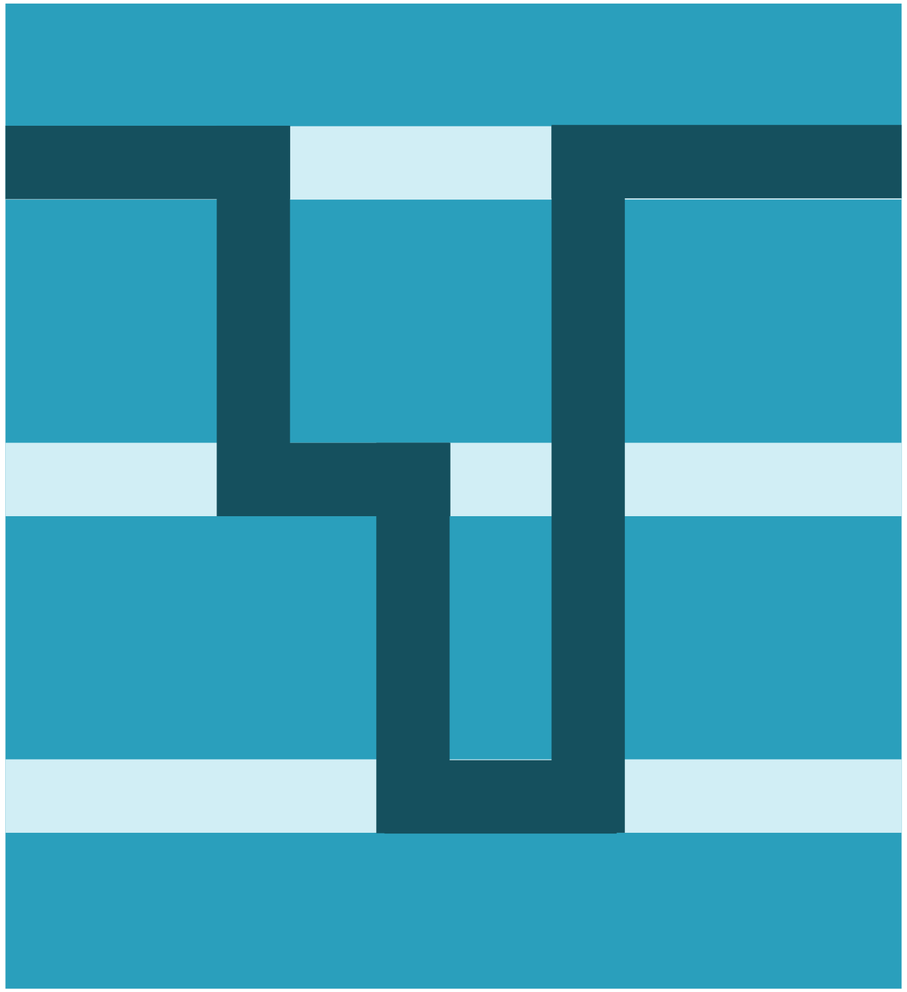
WaitGroups

Channels

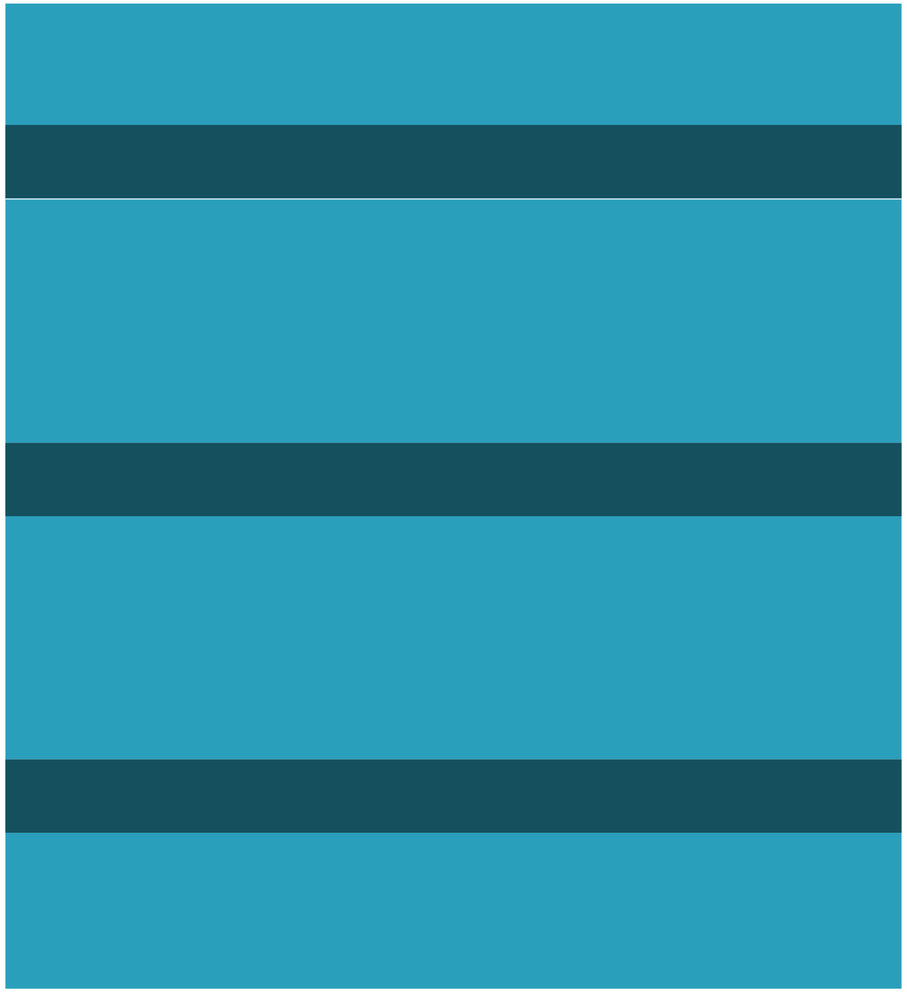
Concurrency is not parallelism.

Go Proverbs - https://go-proverbs.github.io/

Concurrency



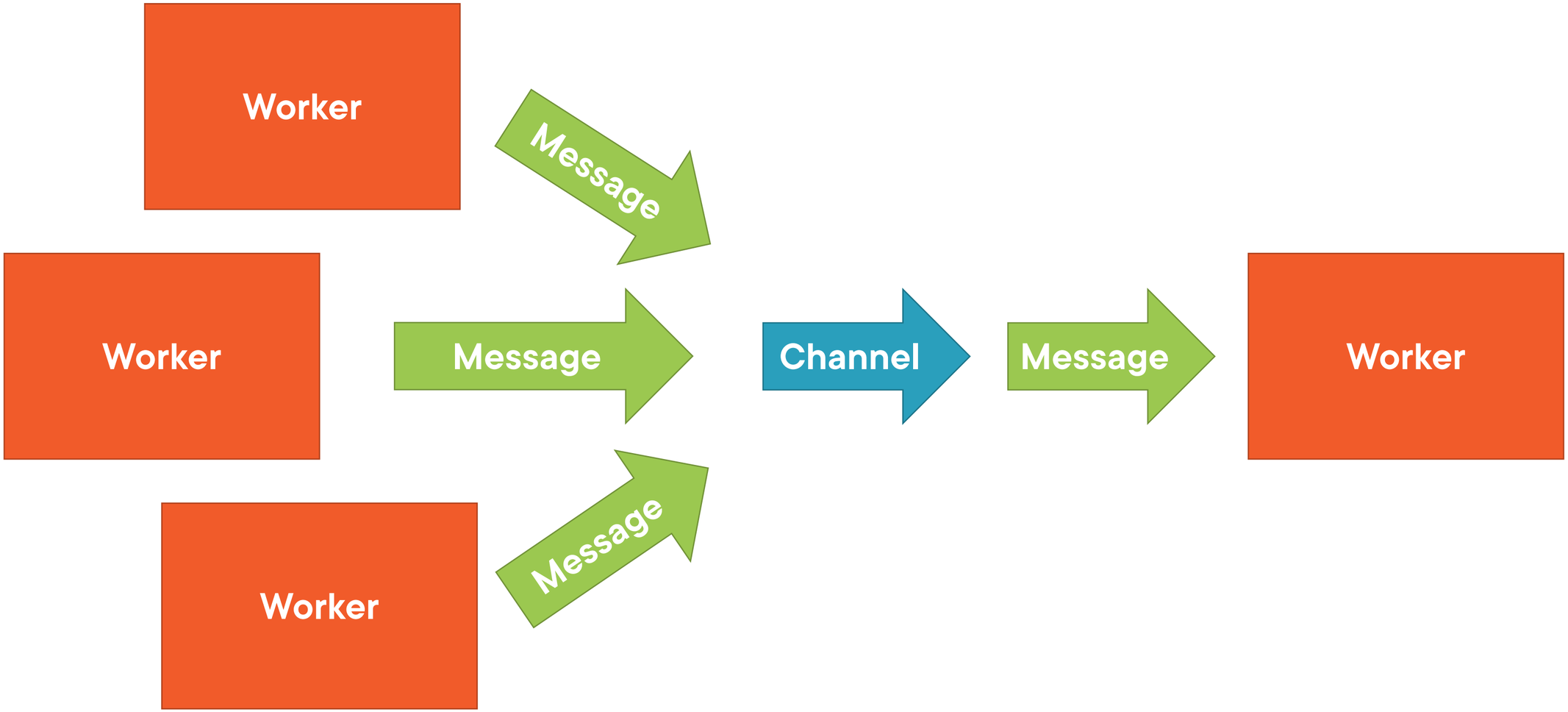
Parallelism



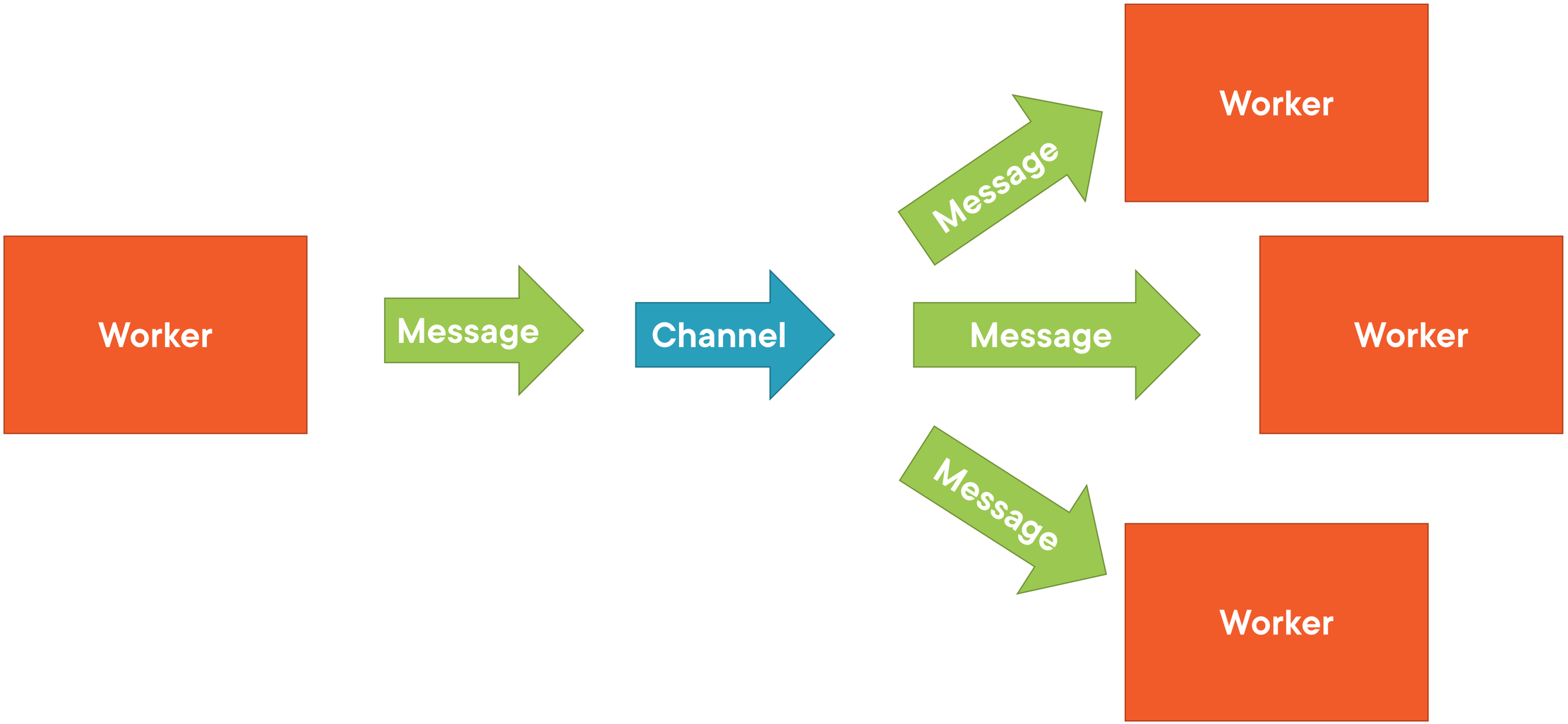
Concurrency in Go

Communicating Sequential Processes (CSP)
https://en.wikipedia.org/wiki/Communicating_sequential_processes

Concurrency in Go



Concurrency in Go



Concurrency in Go

goroutine

channel

goroutine

A *goroutine* is a function executing concurrently with other goroutines in the same address space. It is lightweight, costing little more than the allocation of stack space.

https://go.dev/doc/effective_go#goroutines

WaitGroups are simply counters that have special behavior when their value is zero.

WaitGroups

```
type WaitGroup ...

func (wg WaitGroup) Add(delta int)      // increment counter by delta
func (wg WaitGroup) Done()              // decrement counter by 1
func (wg WaitGroup) Wait()              // wait till counter is zero
```

WaitGroups

```
func main() {


    go func() {
        fmt.Println(“do some async thing”)

    }()


}
```

WaitGroups

```
func main() {
     var wg sync.WaitGroup

     wg.Add(1)
     go func() {
          fmt.Println(“do some async thing”)
          wg.Done()
     }()

     wg.Wait()
}
```

Demo

Goroutines and WaitGroups

Channels

```
// create a channel
ch := make(chan string)

// send a message
ch <- “hello!”

// receive a message
msg := <- ch
```

Channel operations block until the complementary operation is ready

Channels

```
func main() {
     var wg sync.WaitGroup
     ch := make(chan int)

     wg.Add(1)

     go func() {
          ch <- 42
     }()

     go func() {
          fmt.Println(<-ch)
          wg.Done()
     }()

     wg.Wait()
}
```

Demo

Channels

send and receive messages

Select Statements

```
select {
     case channel operation:
          statements
     case channel operation:
          statements
     default:                          // optional
          statements
}
```

In a select statement, if more than one case can be acted upon then one case is chosen **randomly**.

Demo

Channels

select statements

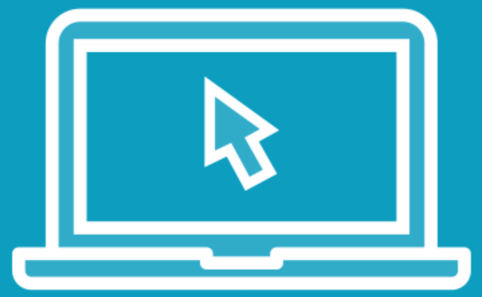
Looping

```
ch := make(chan int)

go func() {
    for i := 0; i < 10; i++ {
        ch <- i
    }
    close(ch)                          // no more messages can be sent
}()

for msg := range ch {
    statements
}
```

Demo



Channels

ranging over channels

Summary

Concurrency in Go

Goroutines

WaitGroups

Channels