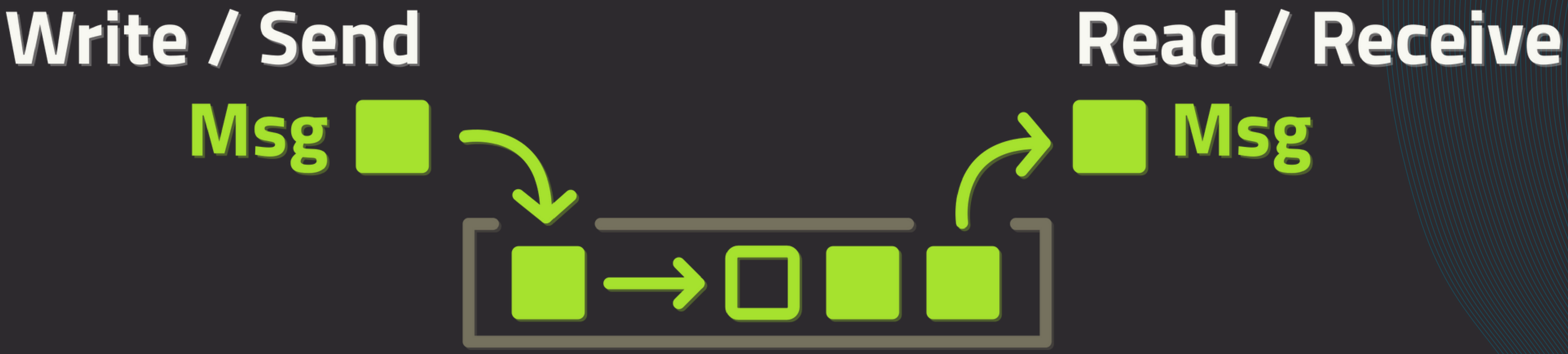
ommunication Chan

# Channels

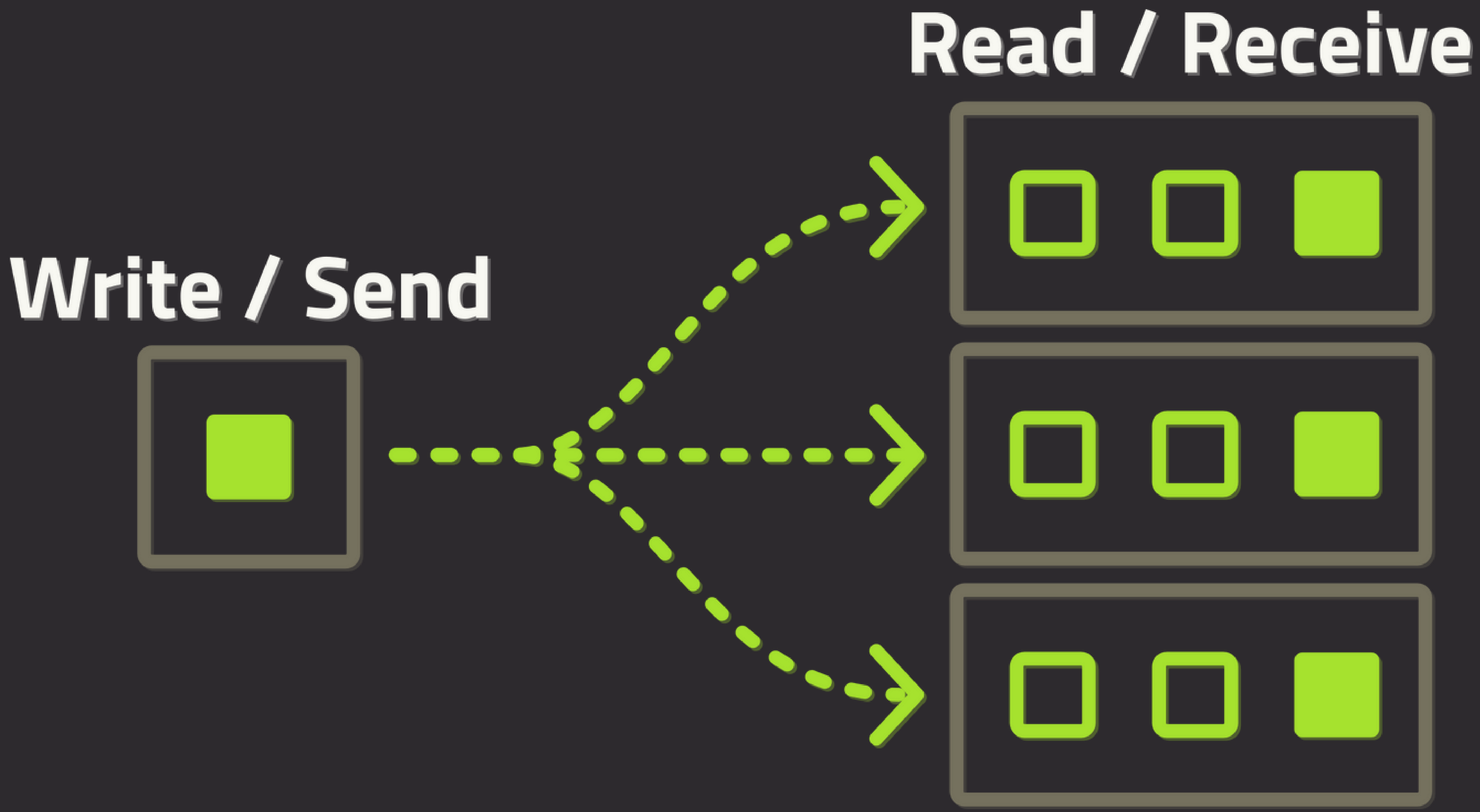
# Channels

- **Channels** offer bidirectional communication
  - Conceptually the same as a two-ended pipe:
    - Write data in one end and read data out the other
    - This is also called **sending** and **receiving**
- Utilizing channels enables goroutines to communicate:
  - Can send/receive messages or computational results
- Channel ends can be duplicated across goroutines

## Visual



# Multiple Receive Ends



## Creation & Usage

```go
channel := make(chan int)

// Send to channel
go func() { channel <- 1 }()
go func() { channel <- 2 }()
go func() { channel <- 3 }()

// Receive from channel
first := <-channel
second := <-channel
third := <-channel
fmt.Println(first, second, third)
```

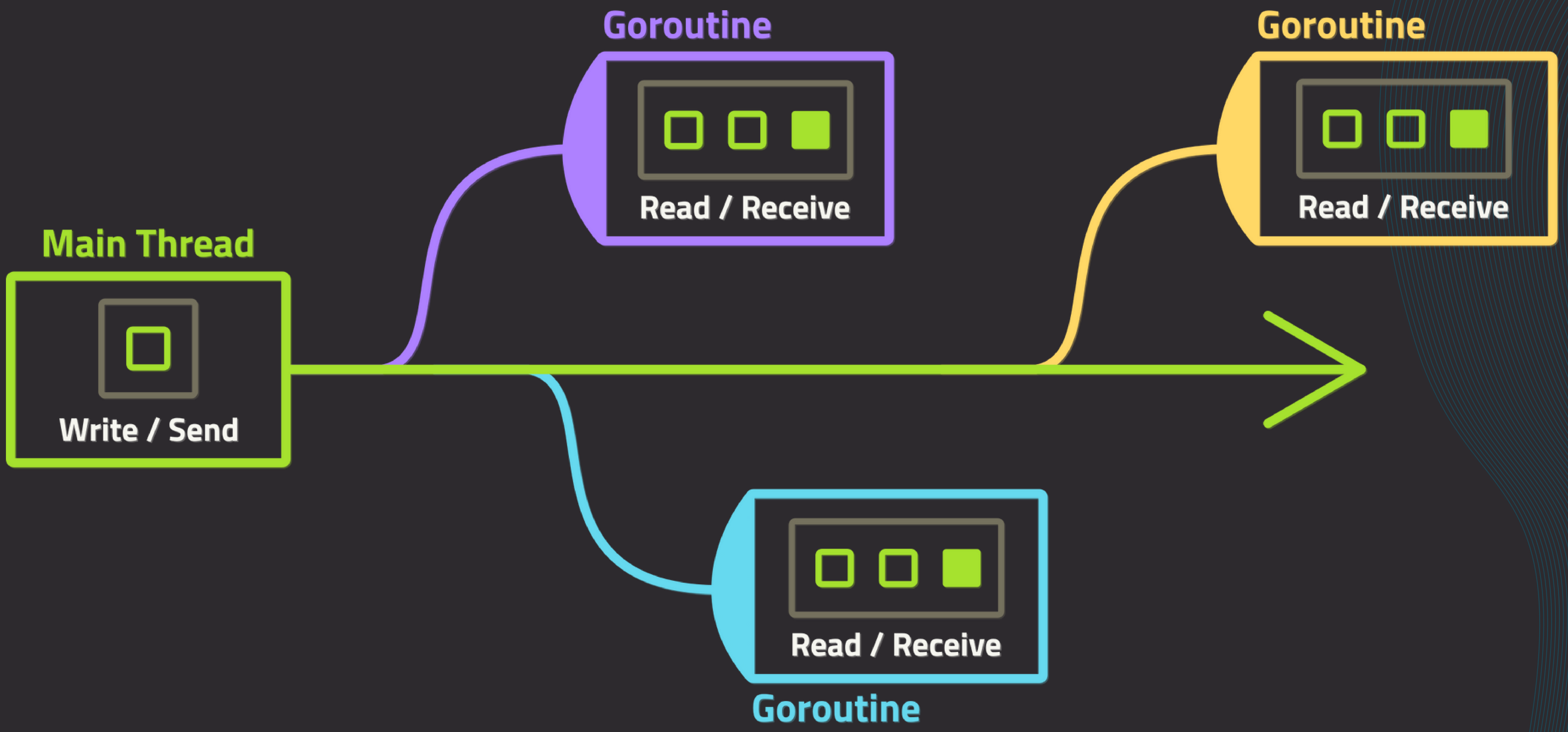
## Details

- Channels can be **buffered** or **unbuffered**
  - Unbuffered channels will **block** when sending until a reader is available
  - Buffered channels have a specified capacity
    - Can send messages up to the capacity, even without a reader
- Messages on a channel are **FIFO** ordering

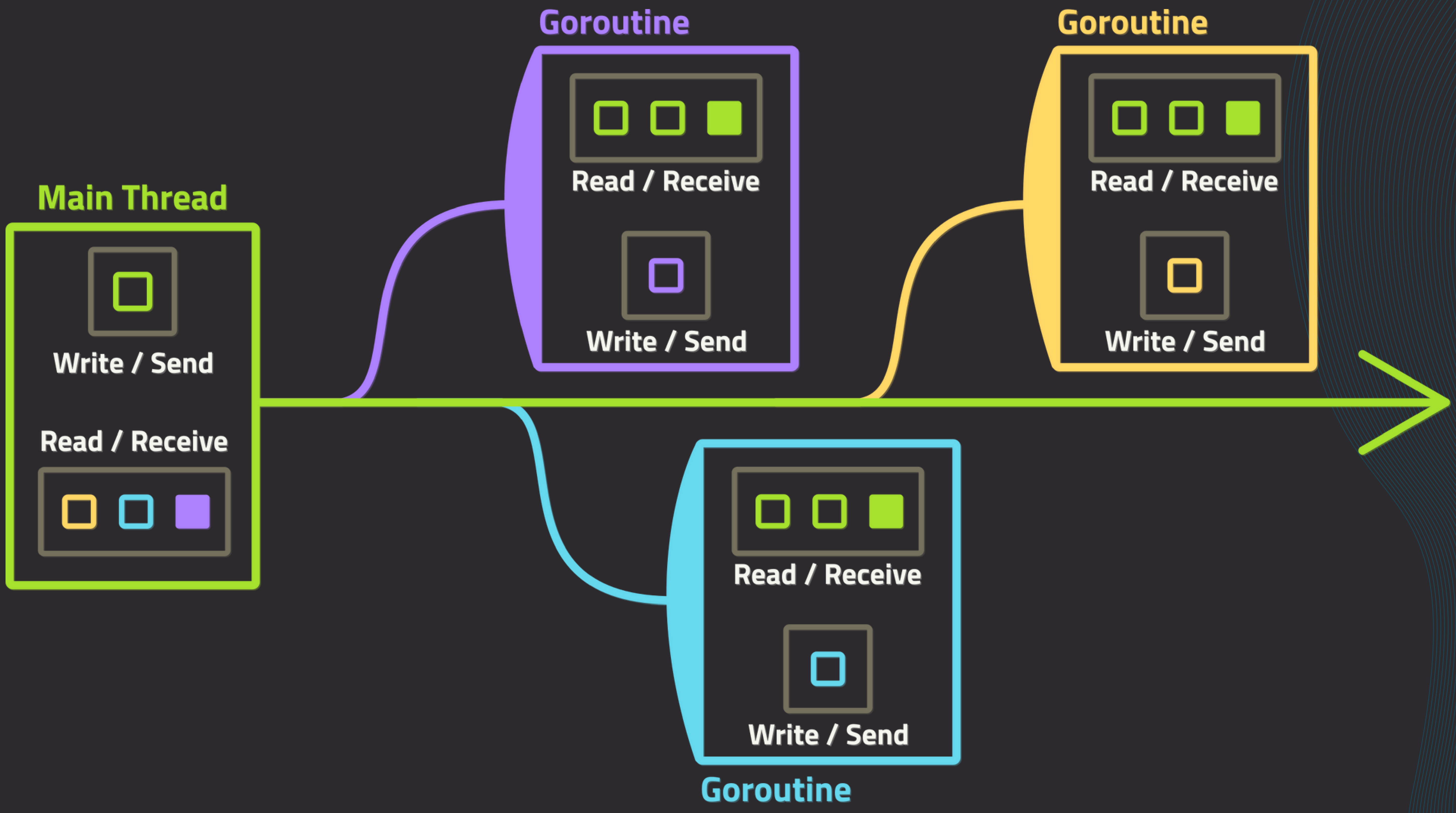
# Buffered Channel

```go
channel := make(chan int, 2)

channel <- 1
channel <- 2

go func() { channel <- 3 }()

first := <-channel
second := <-channel
third := <-channel
fmt.Println(first, second, third)
```

# Goroutines: Unidirectional



# Goroutines: Control Channel



# Channel Selection

- The **select** keyword lets you work with multiple, potentially blocking, channels
- Send/Receive attempts are made, regardless of blocking status

```go
one := make(chan int)
two := make(chan int)

for {
    select {
    case o := <-one:
        fmt.Println("one:", o)
    case t := <-two:
        fmt.Println("two:", t)
    default:
        fmt.Println("no data to receive")
        time.Sleep(50 * time.Millisecond)
    }
}
```

# Timeouts

- The **time** package can be combined with **select** to create timeouts

```go
one := make(chan int)
two := make(chan int)

for {
    select {
    case o := <-one:
        fmt.Println("one:", o)
    case t := <-two:
        fmt.Println("two:", t)
    case <-time.After(300 * time.Millisecond):
        fmt.Println("timed out")
        return
    }
}
```

## Recap

- **Channels** are bidirectional communication pipes
  - They have a **send/write** end and a **receive/read** end
- The ends of a channel can be duplicated across goroutines
- **select** can be used to send or receive on multiple different channels
- Buffered channels are non-blocking, unbuffered channels will block