Le Langage Go

# Goroutine & Channels

## Comprendre les Channels

Robin Penea

# On a des Goroutines !

Robin Penea

# Comment les synchroniser ?

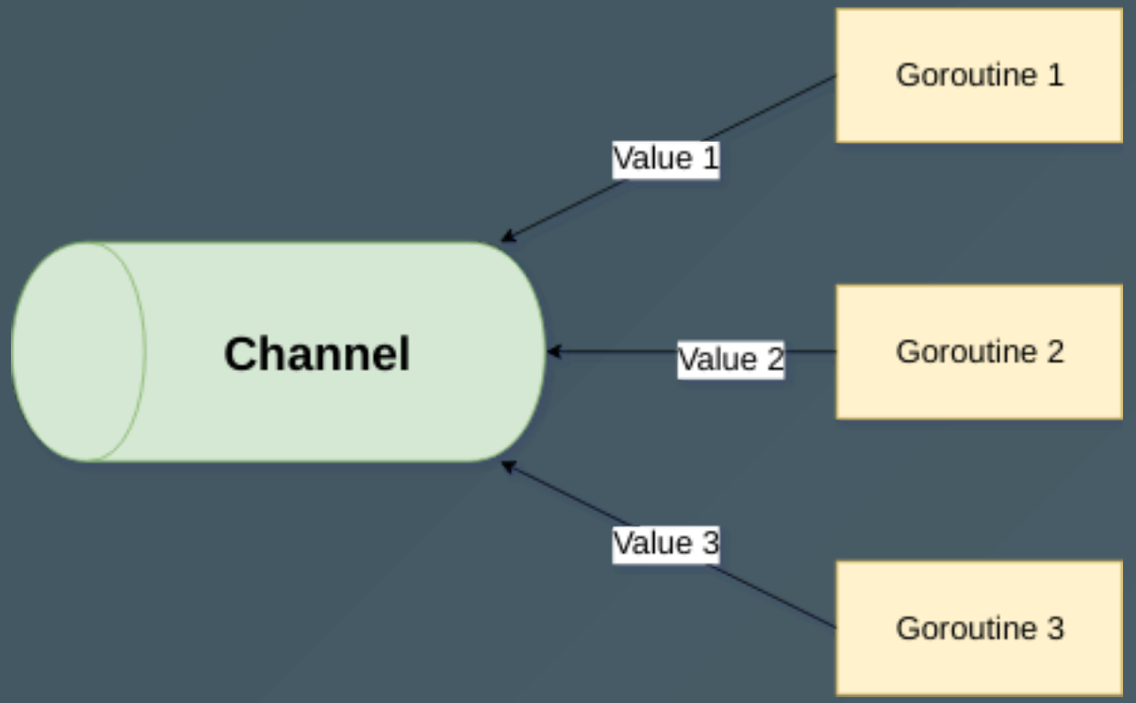
# Les Channels servent à cela !

# Définition

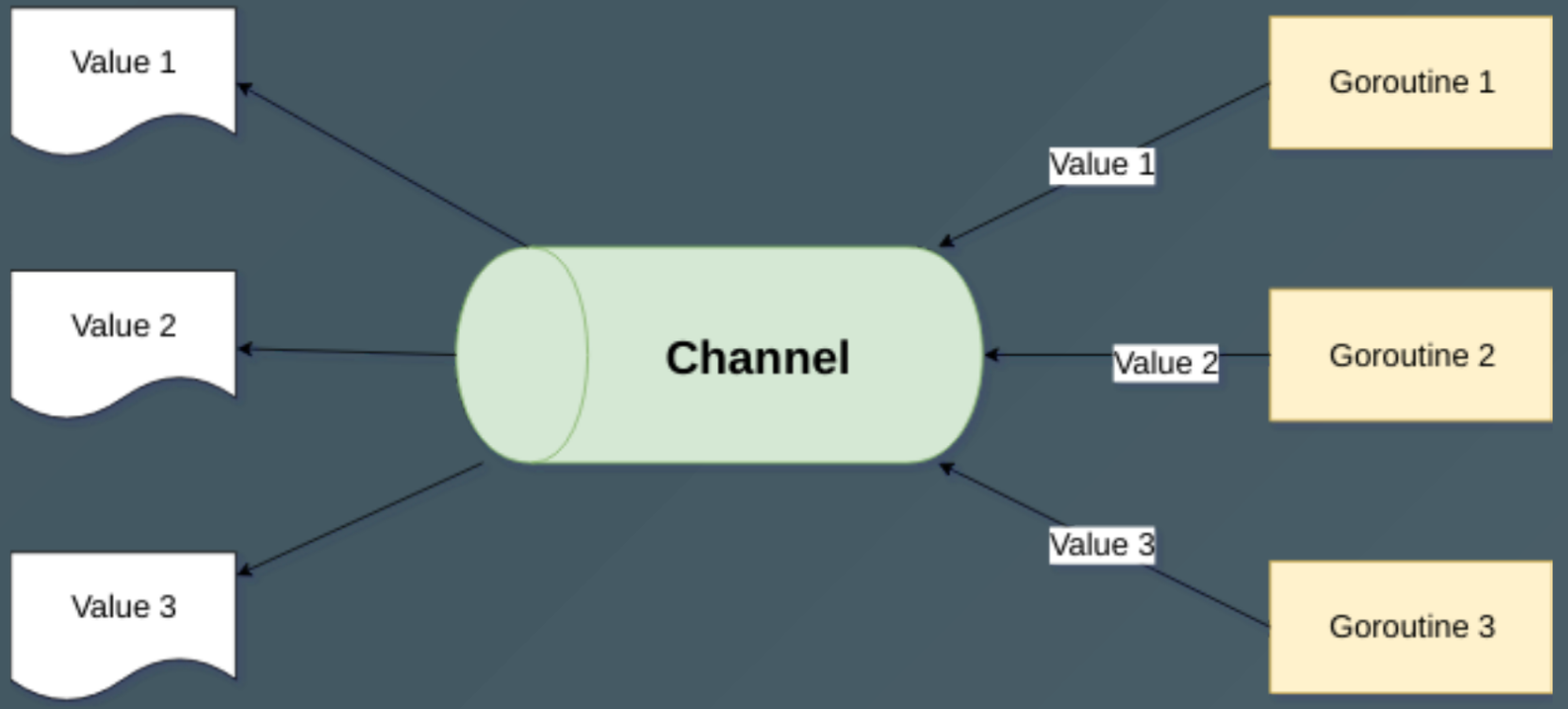
# Des tuyaux qui font communiquer les Goroutines

Robin Penea

# Ecriture dans un channel



# Lecture dans un channel



# La lecture/écriture est synchrone !

# Dans quel cas on a besoin de channels ?

# Aggréger les résultats des Goroutines

Robin Penea

# Séquencer l'exécution des Goroutines

Robin Penea

# Syntaxe

Robin Penea

# Syntaxe

```
c := make(chan <Type>) // type est le type de variable qui transitera
```

# Syntaxe

```
c := make(chan int)
```

# Syntaxe

```
c := make(chan int)
c <- 1 // écrire dans un channel
```

# Syntaxe

```
c := make(chan int)
c <- 1 // écrire dans un channel
i := <-c // lire dans un channel, i=1
```

# Exemple avec une Goroutine

# Exemple avec une Goroutine

```
func calculatePi(c chan float64) {
    // long calculation...
    c <- resPi
}
```

# Example avec une Goroutine

```
func calculatePi(c chan float64) {
    // long calculation...
    c <- resPi
}

c := make(chan float64)
```

# Exemple avec une Goroutine

```
func calculatePi(c chan float64) {
    // long calculation...
    c <- resPi
}

c := make(chan float64)
go calculatePi(c) // exécuté dans une goroutine
```

# Exemple avec une Goroutine

```
func calculatePi(c chan float64) {
    // long calculation...
    c <- resPi
}

c := make(chan float64)
go calculatePi(c) // exécuté dans une goroutine
pi := <-c   // bloquant jusqu'à ce que calculatePi() écrive dans c
```

# Exemple avec une Goroutine

```
func calculatePi(c chan float64) {
    // long calculation...
    c <- resPi
}

c := make(chan float64)
go calculatePi(c) // exécuté dans une goroutine
pi := <-c   // bloquant jusqu'à ce que calculatePi() écrive dans c
fmt.Println("Pi value is", pi) // output 3.1415926535...
```