Introduction of Go Channel

Gaston Chiu

Aug. 02 2020 @ Coscup

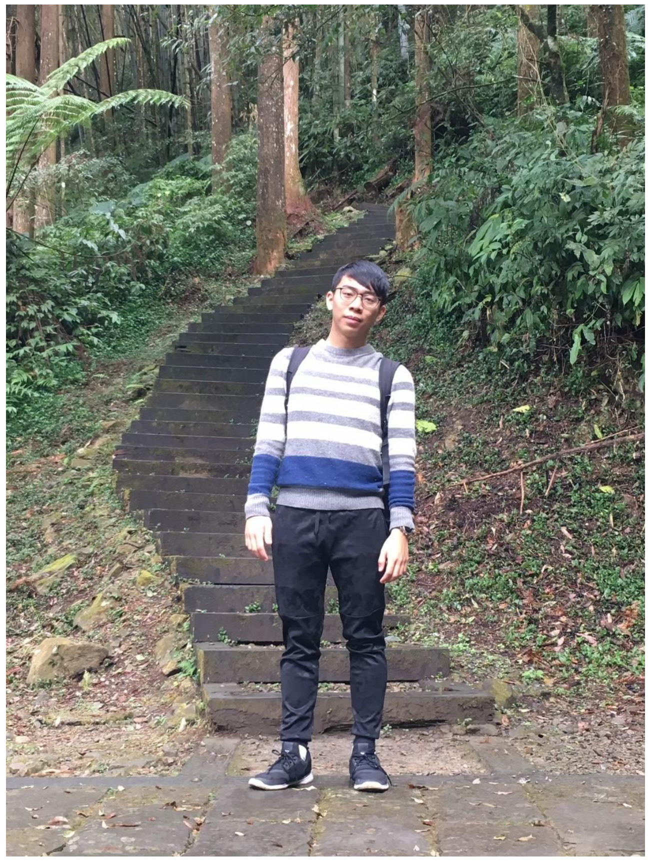
Gaston Chiu

Umbo CV Backend Engineer

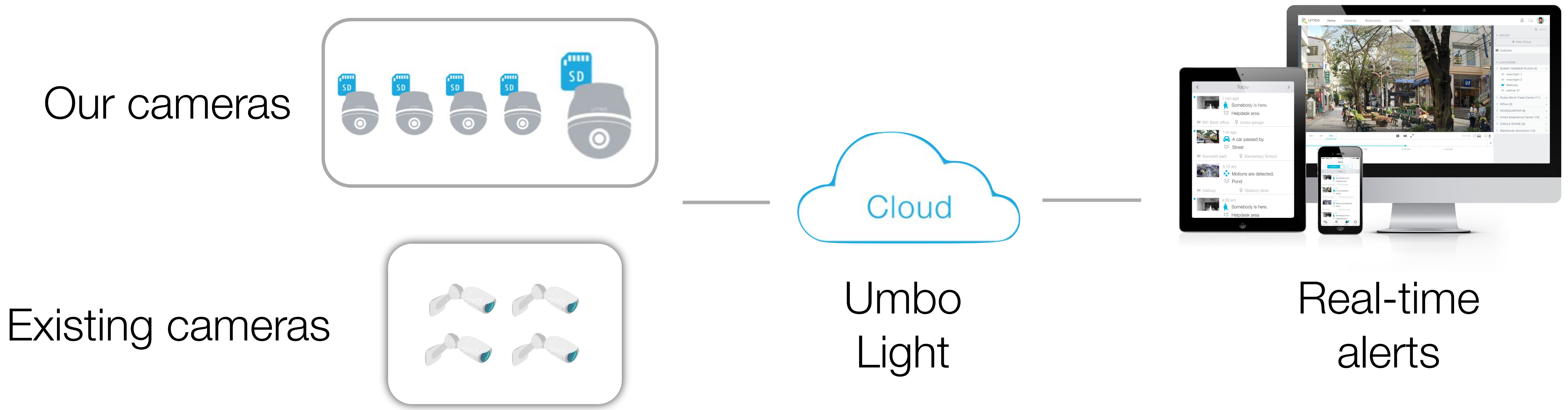
- Umbo CV Backend Engineer 2019/07
- 國軍Online 2019/11 (還好我退了)
- Umbo CV Backend Engineer 2020/03 - Now

Open source: Grafana

Contact:
gaston.qiu@umbocv.com
Github: gastonqiu



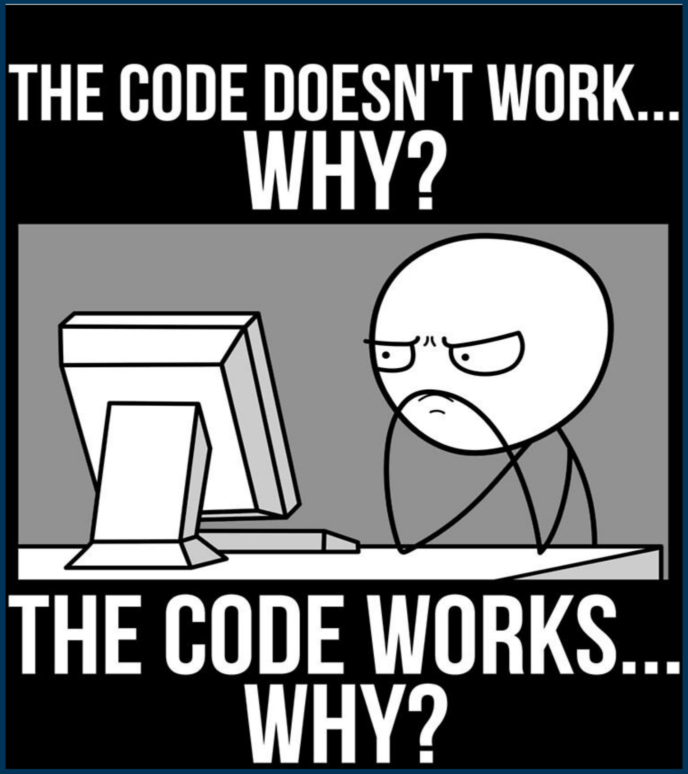
Simplified Hosting Solutions



No NVR, VMS, Local Server

Outline

Channel 101



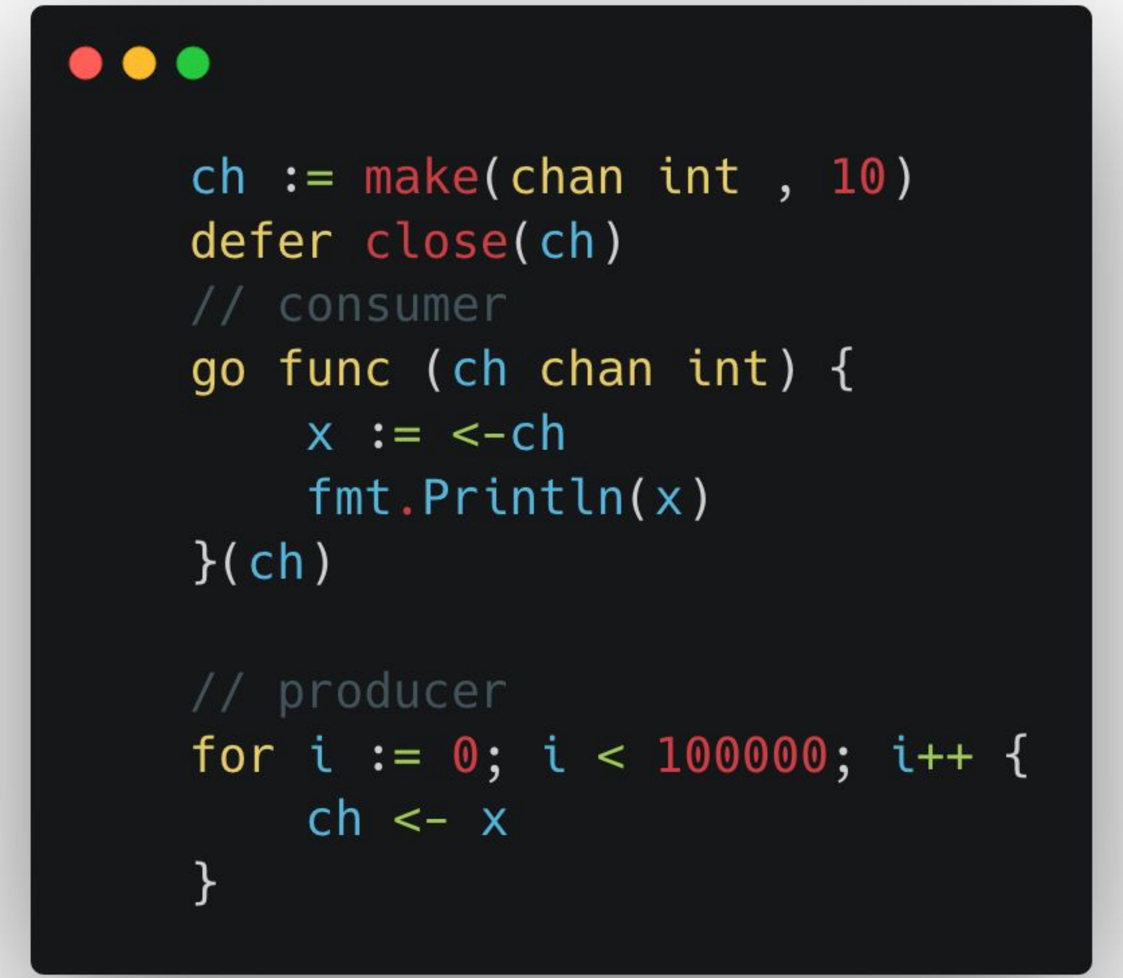
Communication Sequential Process

- CSP first mention on 1978 Tony Hoare Paper
- Passing message via channel. No shared state!
- sending input into process: ch!val
- receiving output from process: ch?val

Channel 101

```
ch := make(chan dataType, size(optional))

ch <- x // send channel

x := <- ch // receive channel
```

How To Use Channel

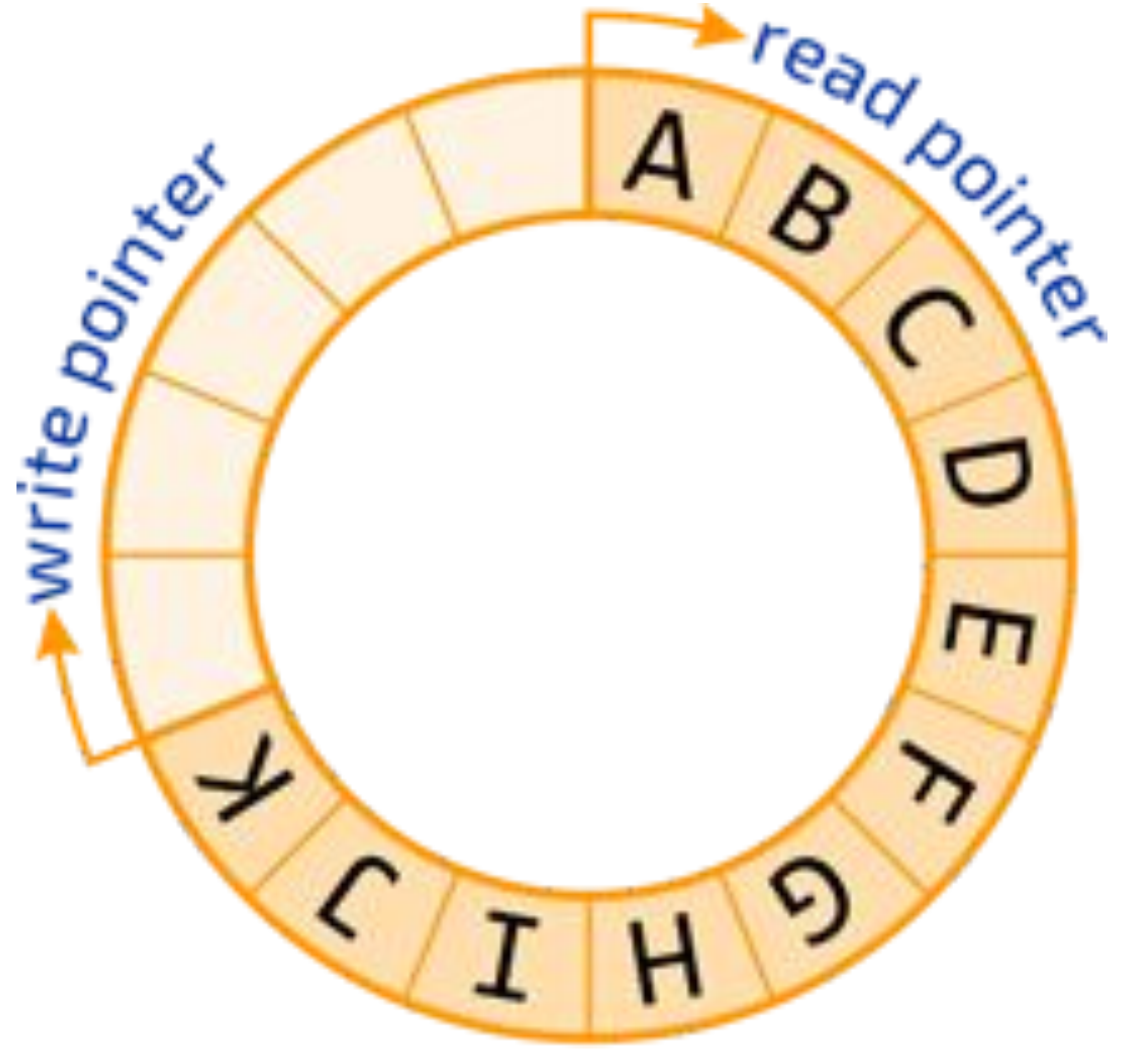


```
ch := make(chan int , 10)
defer close(ch)
// consumer
go func (ch chan int) {
    x := <-ch
    fmt.Println(x)
}(ch)

// producer
for i := 0; i < 100000; i++ {
    ch <- x
}
```

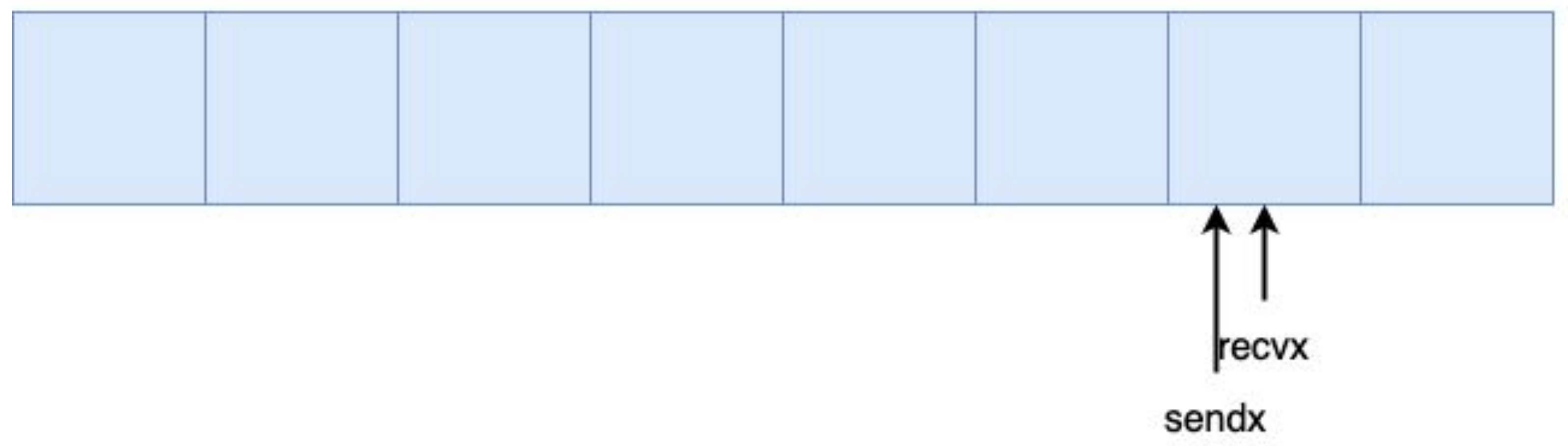
What We Need For Channel

- No dead lock
- No race condition
- FIFO Queue

Buffer (Circular Queue) + Mutex Lock



Blocking v.s. Non-blocking Send, Recv

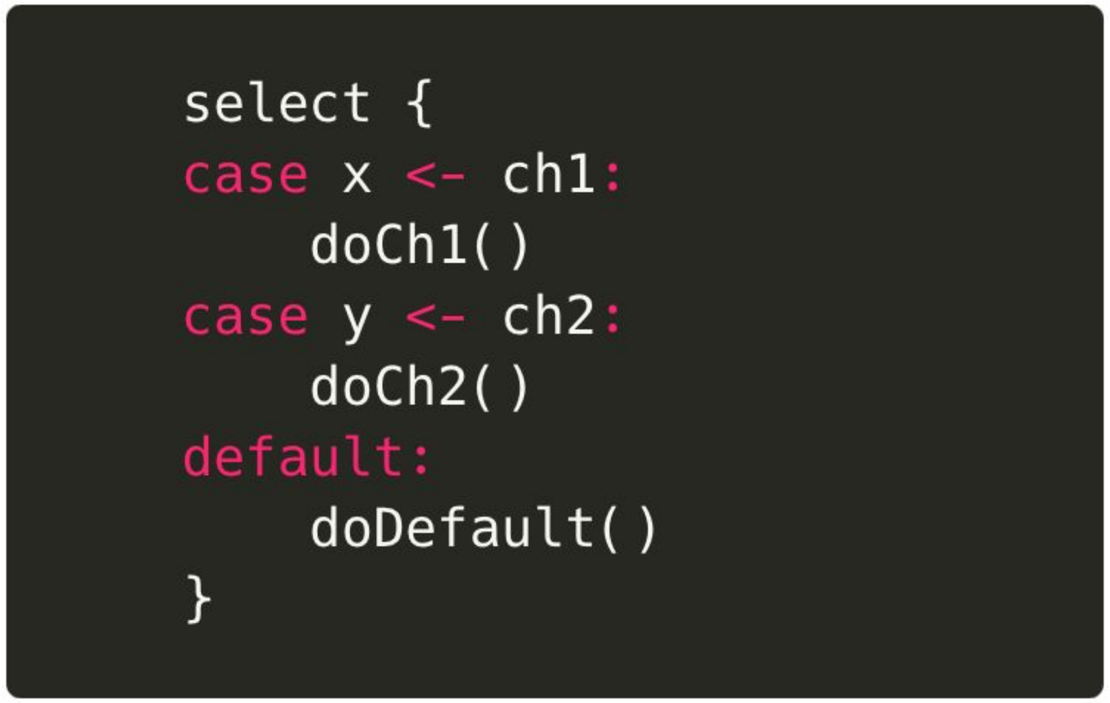


Blocking

- channel is empty when receive (ch := <- x)
- channel is full when send (ch <- x)
- make(chan int) == make(chan int, 0) , every time you try to send data into channel

Non-blocking Recv

- Select

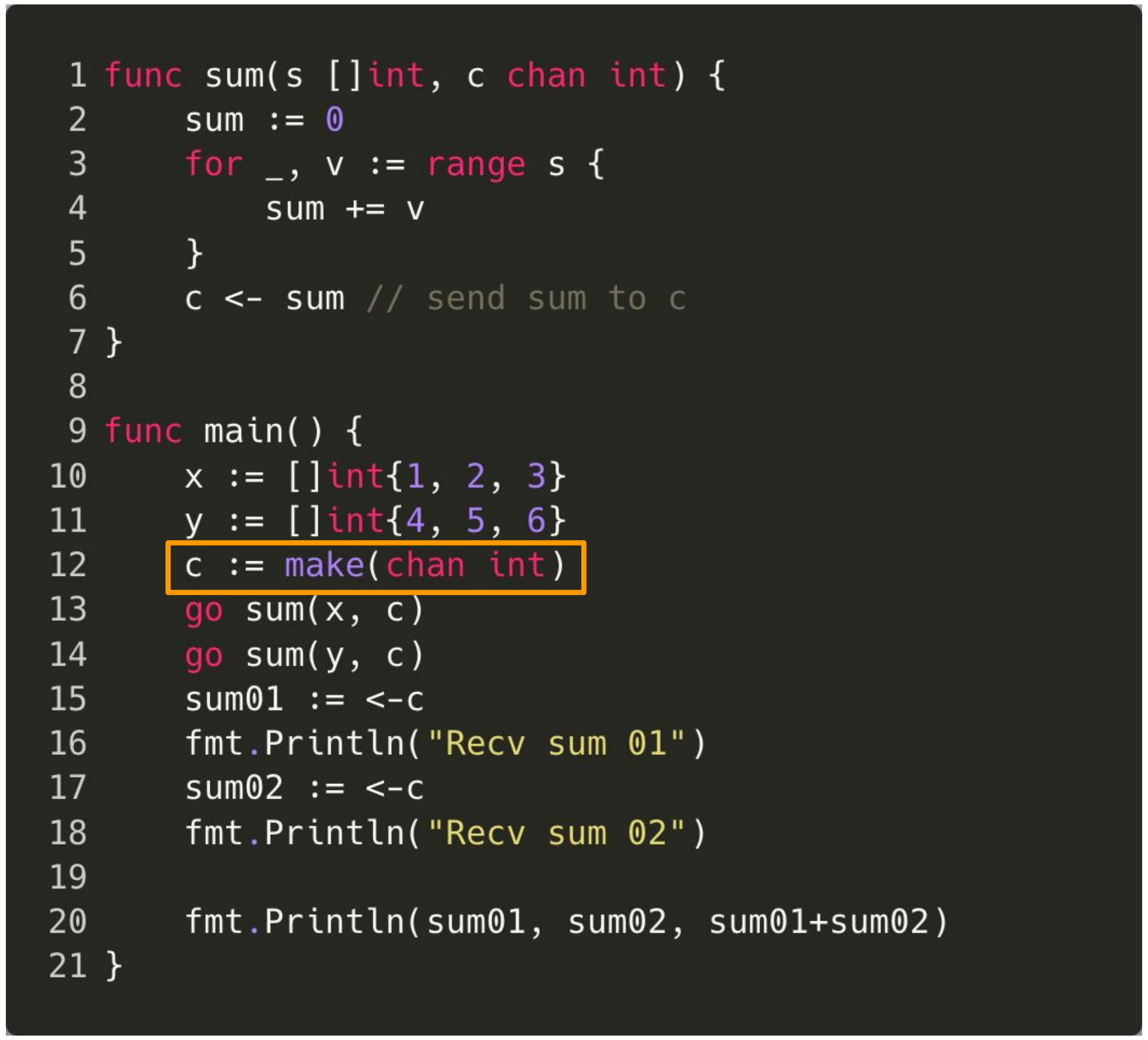


```
select {
case x <- ch1:
    doCh1()
case y <- ch2:
    doCh2()
default:
    doDefault()
}
```

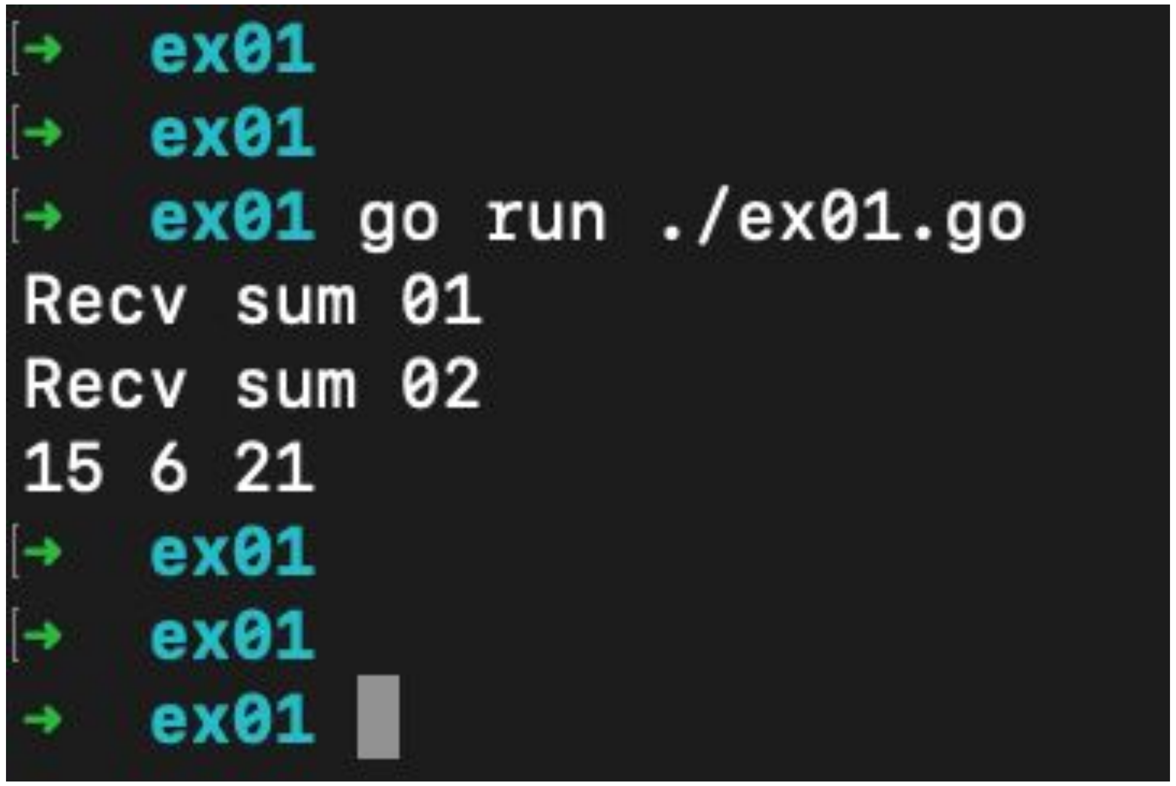
Let’s Check Out Some Examples

Talk is cheap, show me the code.

Return Final Result

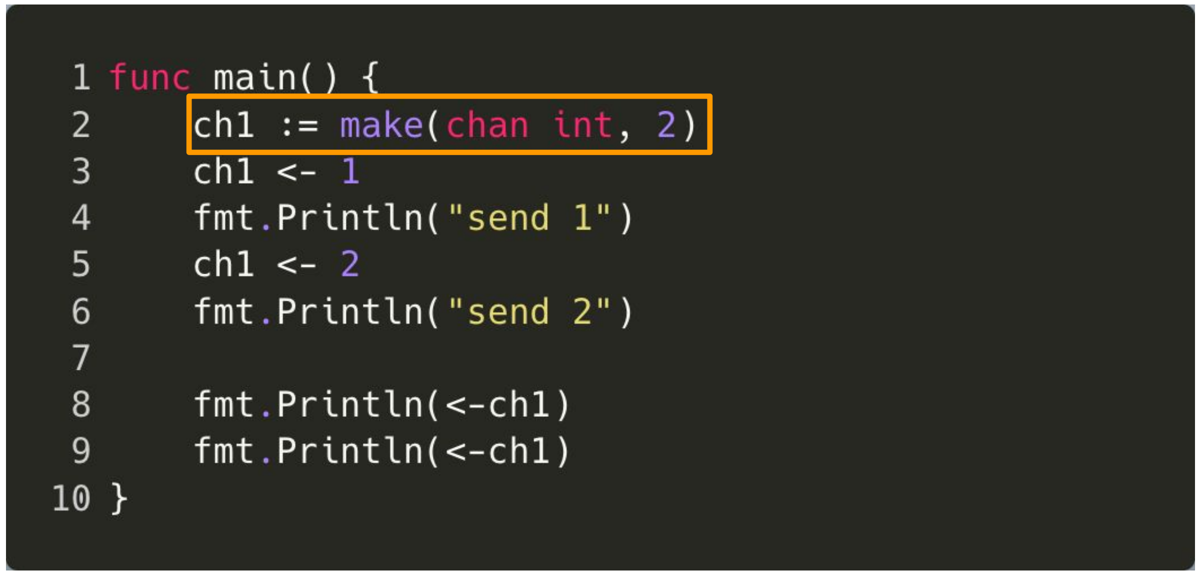


```go
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum // send sum to c
}

func main() {
    x := []int{1, 2, 3}
    y := []int{4, 5, 6}
    c := make(chan int)
    go sum(x, c)
    go sum(y, c)
    sum01 := <-c
    fmt.Println("Recv sum 01")
    sum02 := <-c
    fmt.Println("Recv sum 02")

    fmt.Println(sum01, sum02, sum01+sum02)
}
```

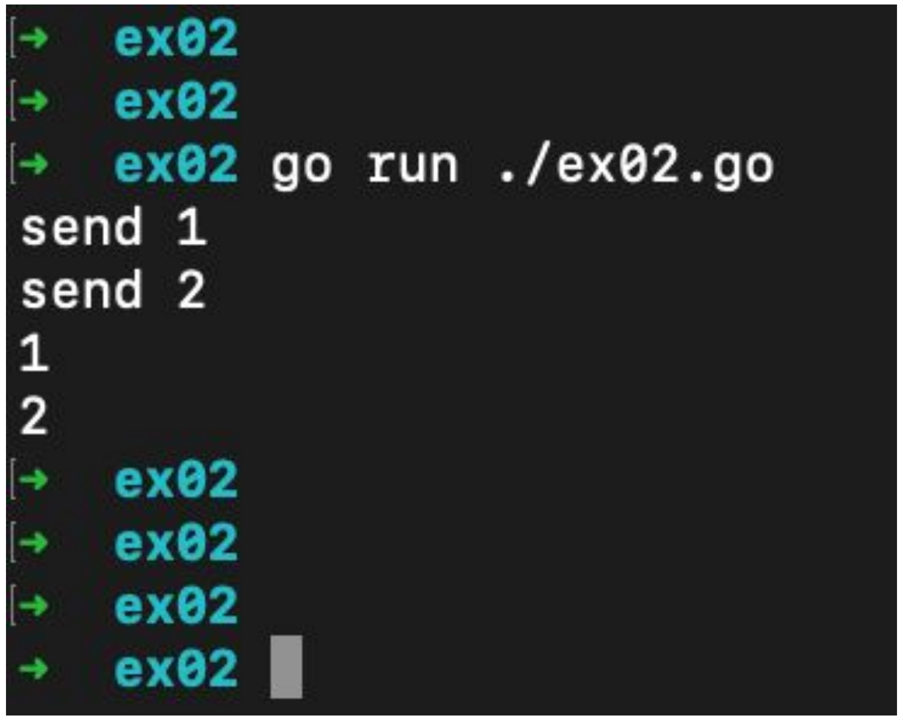


```
➜  ex01
➜  ex01
➜  ex01 go run ./ex01.go
Recv sum 01
Recv sum 02
15 6 21
➜  ex01
➜  ex01
➜  ex01
```

Blocking Receive

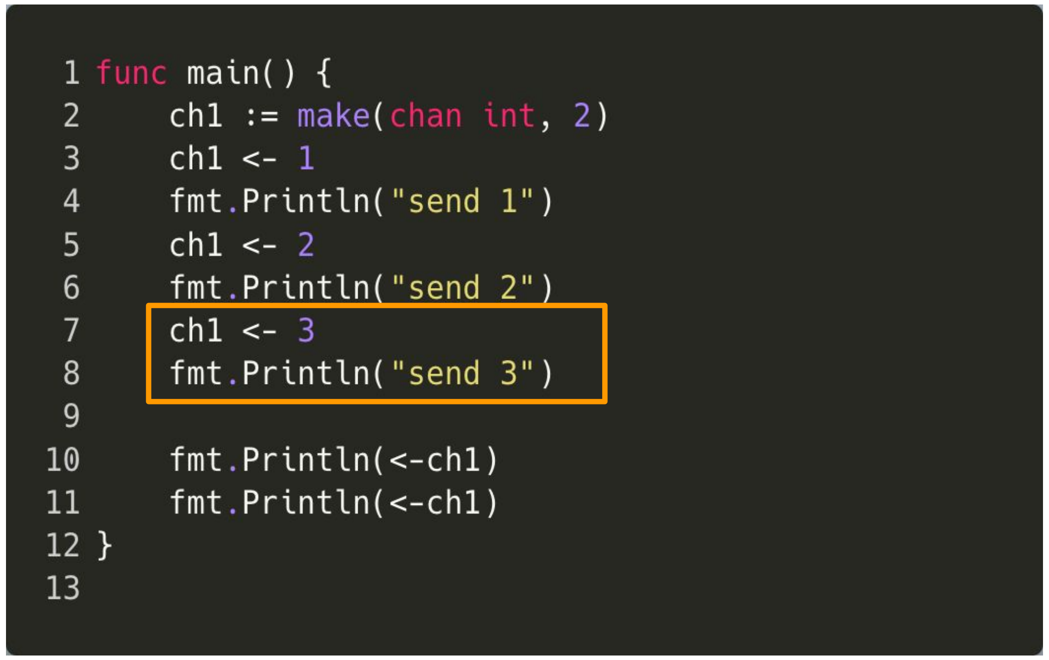


```go
func main() {
    ch1 := make(chan int, 2)
    ch1 <- 1
    fmt.Println("send 1")
    ch1 <- 2
    fmt.Println("send 2")

    fmt.Println(<-ch1)
    fmt.Println(<-ch1)
}
```

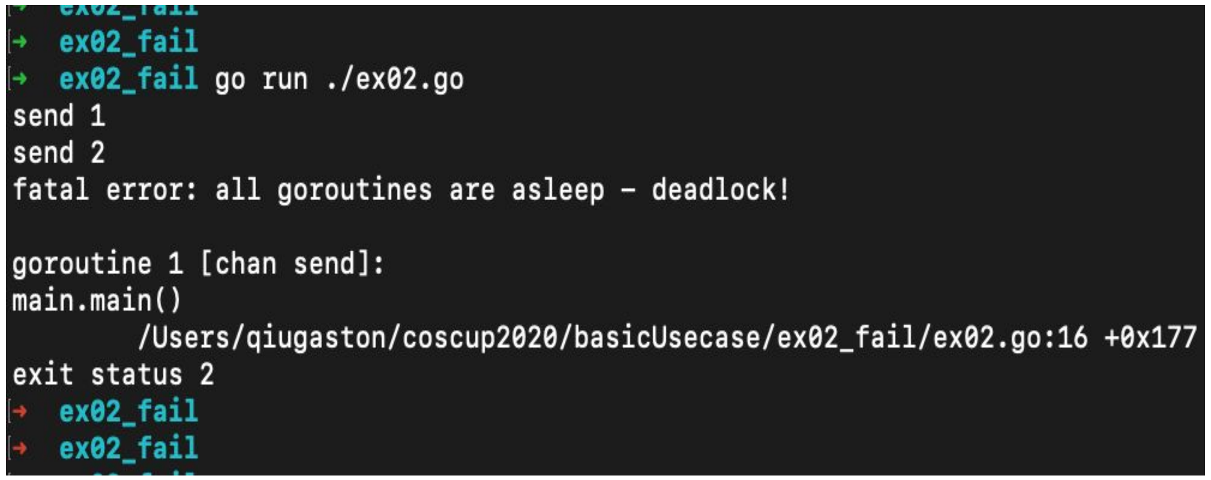


```
→  ex02
→  ex02
→  ex02 go run ./ex02.go
send 1
send 2
1
2
→  ex02
→  ex02
→  ex02
→  ex02
```

# Send Element > Buffer Size

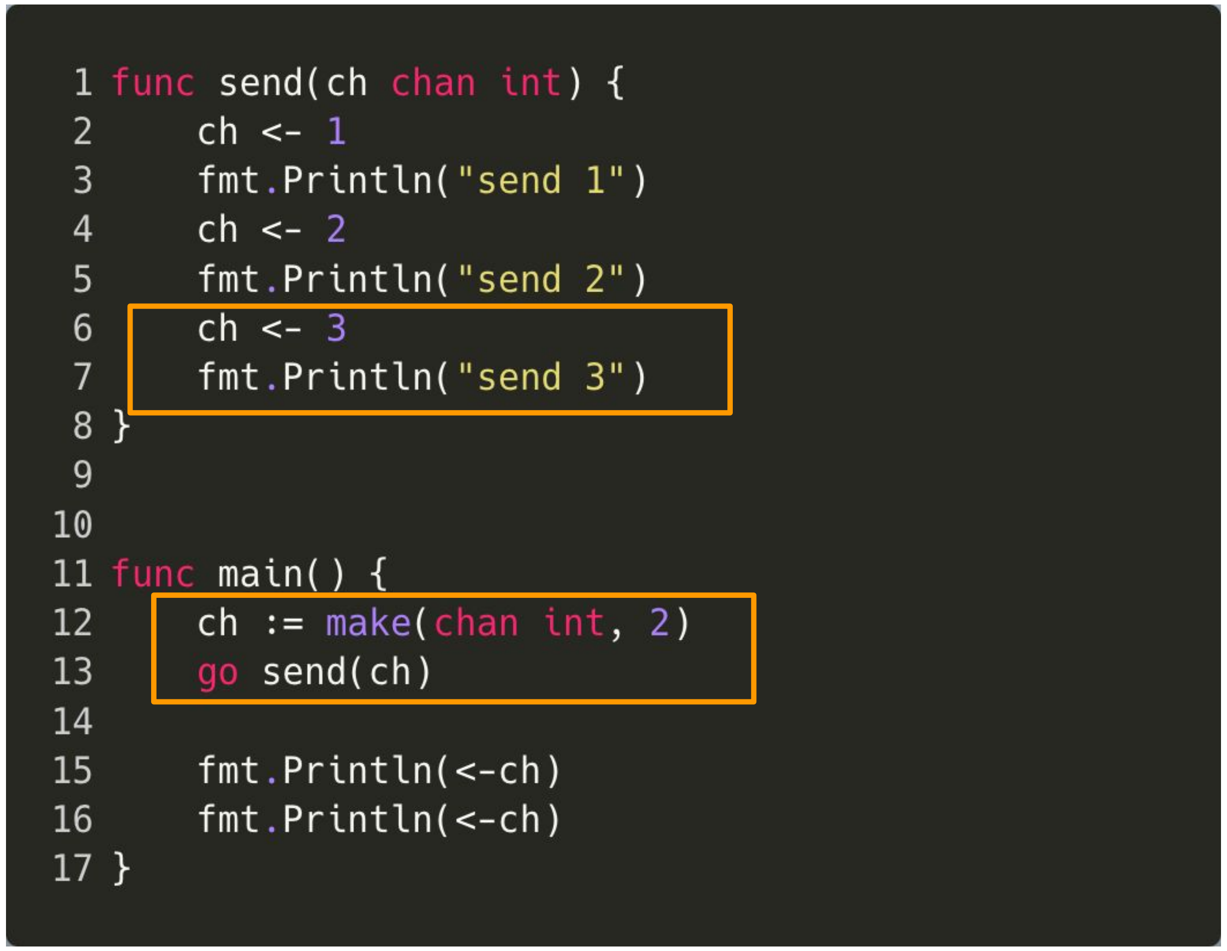


```go
func main() {
    ch1 := make(chan int, 2)
    ch1 <- 1
    fmt.Println("send 1")
    ch1 <- 2
    fmt.Println("send 2")
    ch1 <- 3
    fmt.Println("send 3")

    fmt.Println(<-ch1)
    fmt.Println(<-ch1)
}
```

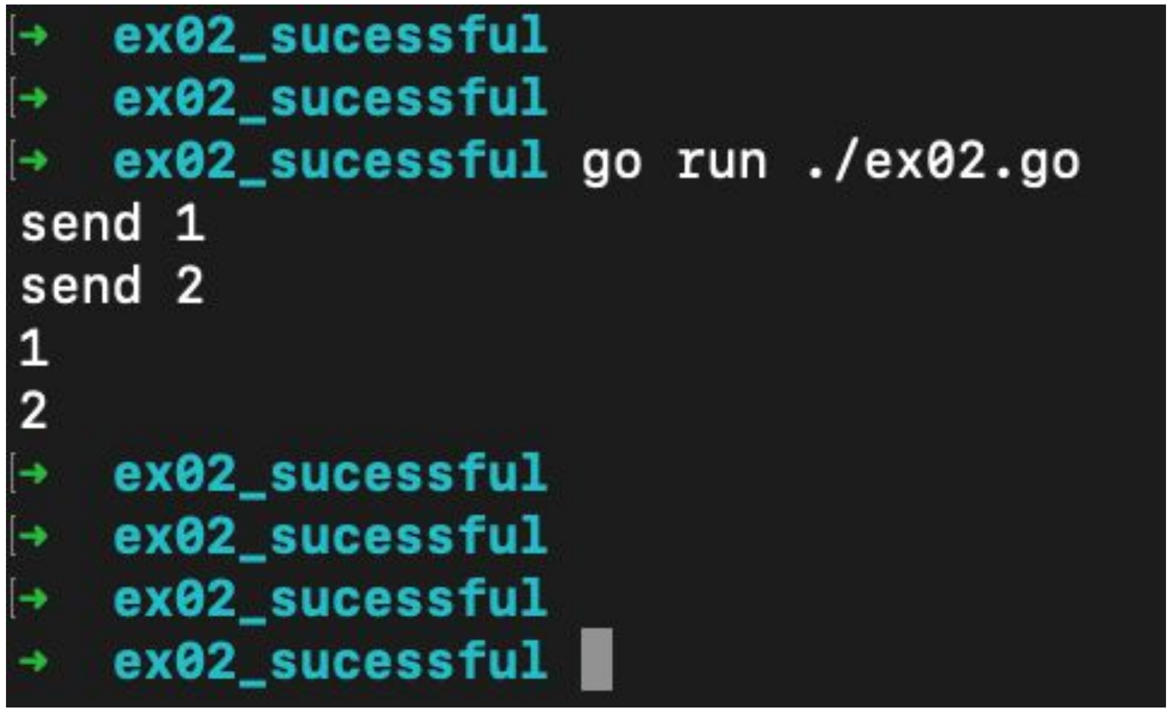


```
→  ex02_fail
→  ex02_fail go run ./ex02.go
send 1
send 2
fatal error: all goroutines are asleep - deadlock!

goroutine 1 [chan send]:
main.main()
        /Users/qiugaston/coscup2020/basicUsecase/ex02_fail/ex02.go:16 +0x177
exit status 2
→  ex02_fail
→  ex02_fail
```

Using Channel Send Values

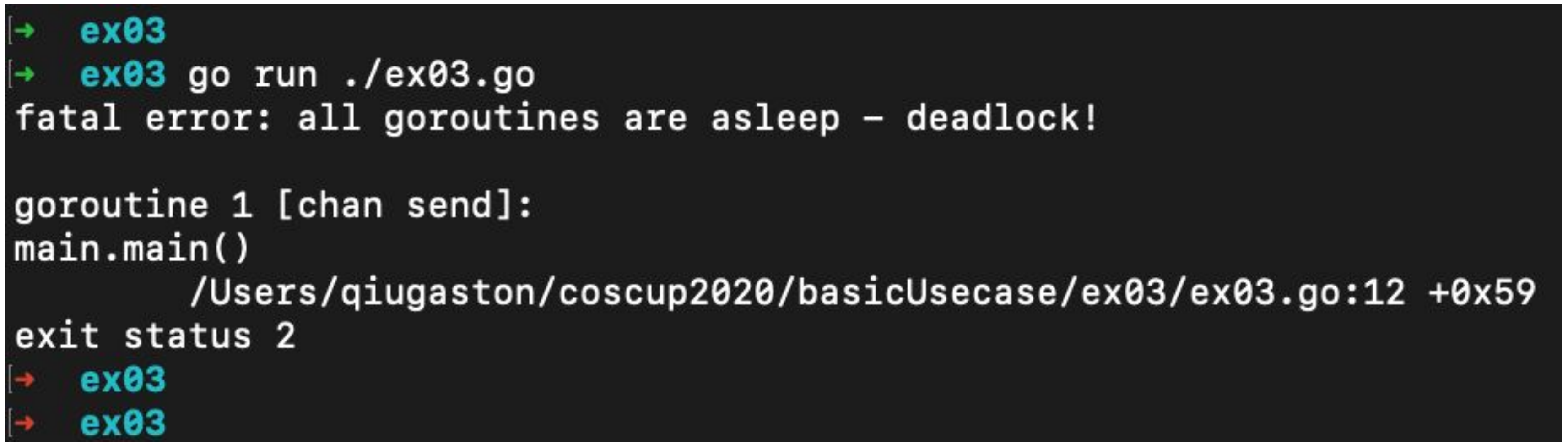


```go
func send(ch chan int) {
    ch <- 1
    fmt.Println("send 1")
    ch <- 2
    fmt.Println("send 2")
    ch <- 3
    fmt.Println("send 3")
}


func main() {
    ch := make(chan int, 2)
    go send(ch)

    fmt.Println(<-ch)
    fmt.Println(<-ch)
}
```

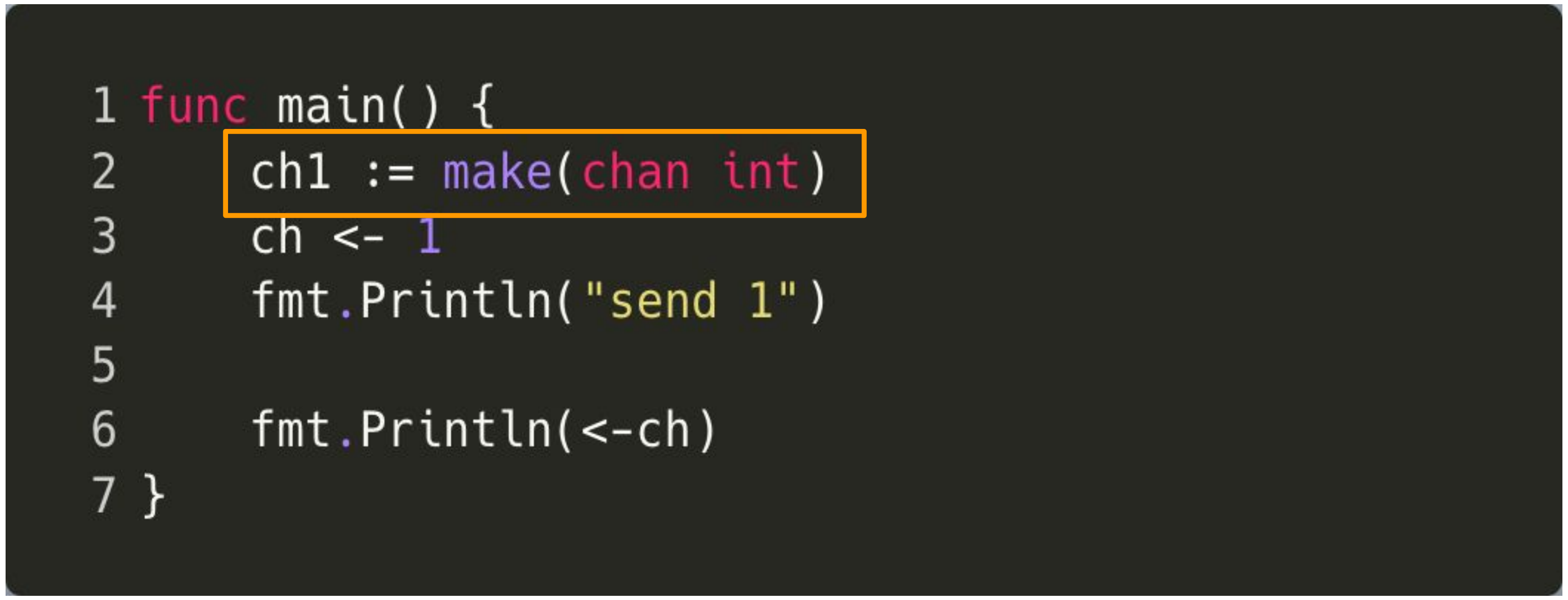


```
→  ex02_sucessful
→  ex02_sucessful
→  ex02_sucessful go run ./ex02.go
send 1
send 2
1
2
→  ex02_sucessful
→  ex02_sucessful
→  ex02_sucessful
→  ex02_sucessful
```

Prove Default Size == 0

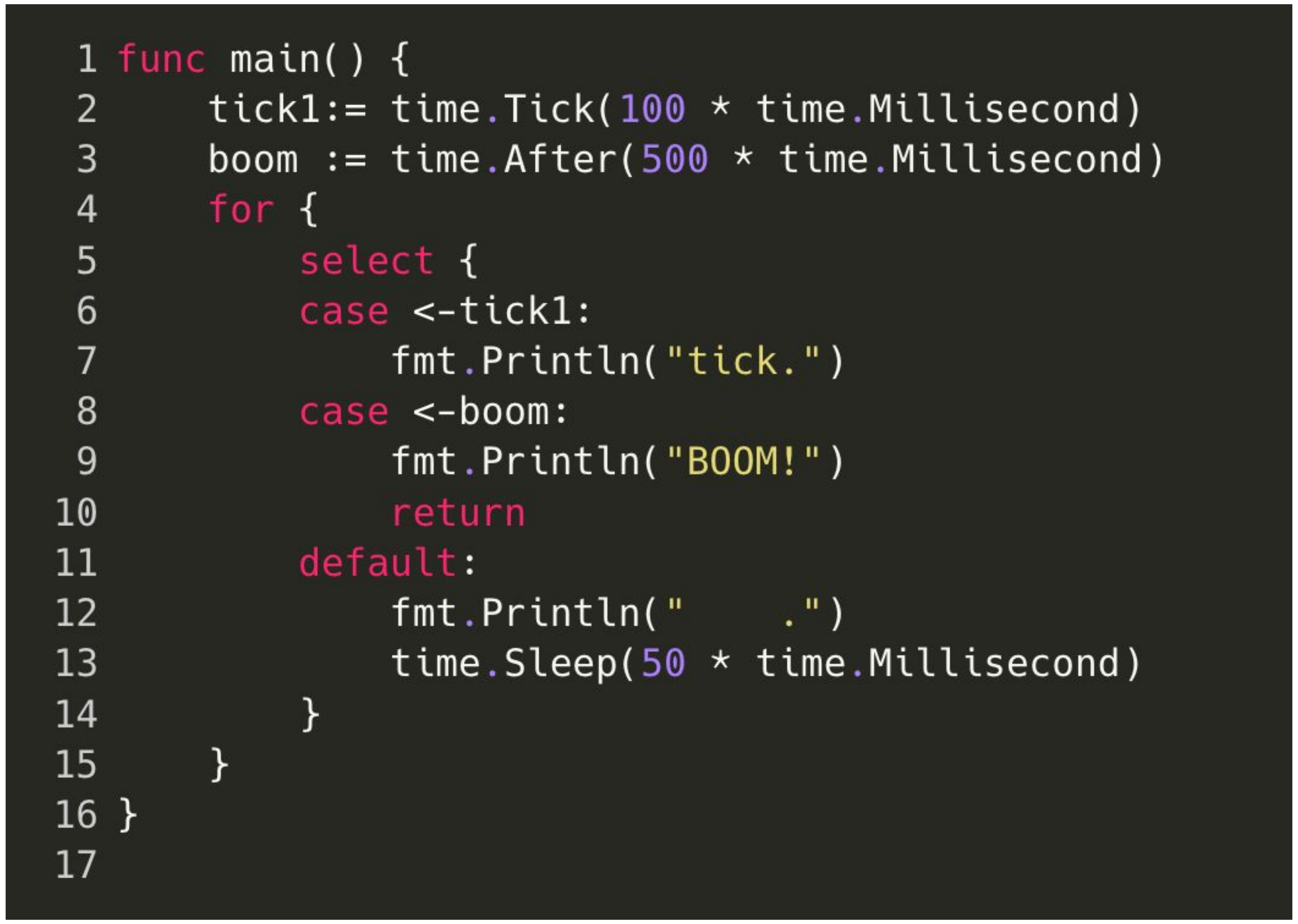


```
→  ex03
→  ex03 go run ./ex03.go
fatal error: all goroutines are asleep - deadlock!

goroutine 1 [chan send]:
main.main()
        /Users/qiugaston/coscup2020/basicUsecase/ex03/ex03.go:12 +0x59
exit status 2
→  ex03
→  ex03
```

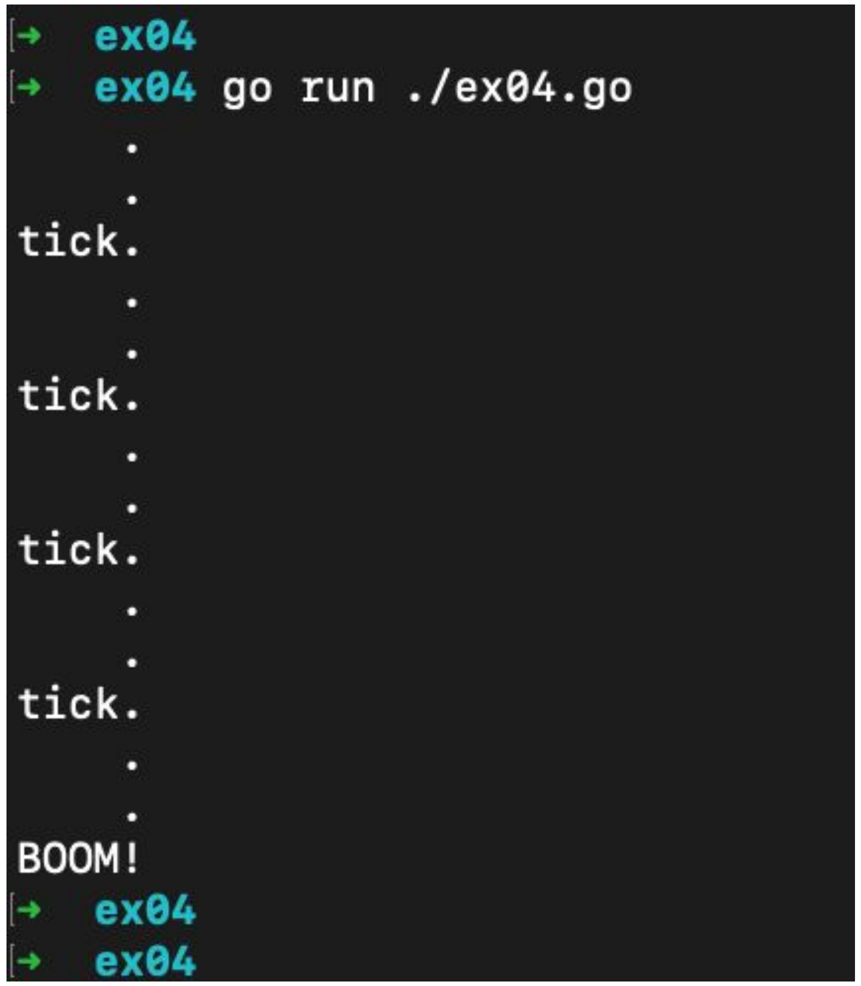


```go
func main() {
    ch1 := make(chan int)
    ch <- 1
    fmt.Println("send 1")

    fmt.Println(<-ch)
}
```

Non-block Recv

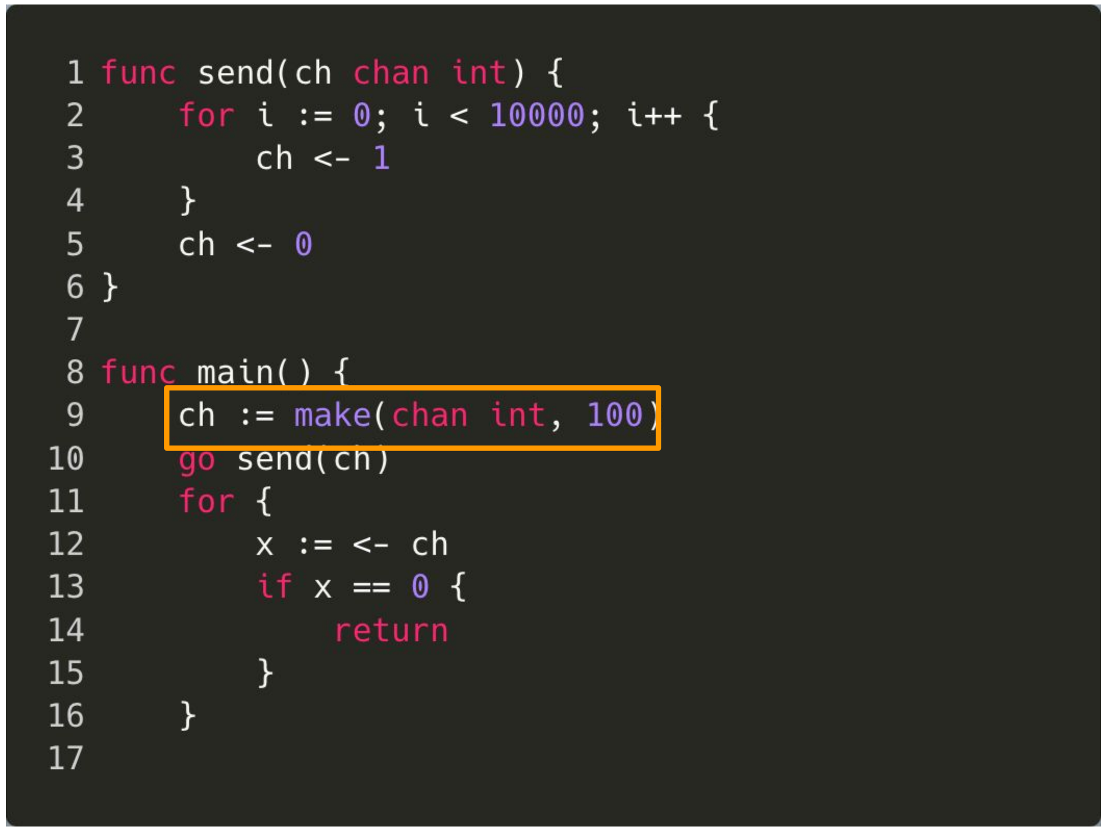


```go
func main() {
    tick1:= time.Tick(100 * time.Millisecond)
    boom := time.After(500 * time.Millisecond)
    for {
        select {
        case <-tick1:
            fmt.Println("tick.")
        case <-boom:
            fmt.Println("BOOM!")
            return
        default:
            fmt.Println("    .")
            time.Sleep(50 * time.Millisecond)
        }
    }
}

```

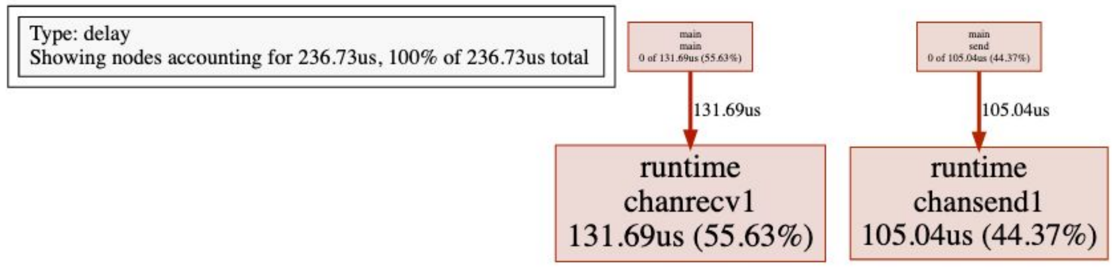


```
➜  ex04
➜  ex04 go run ./ex04.go
    .
    .
tick.
    .
    .
tick.
    .
    .
tick.
    .
    .
tick.
    .
    .
BOOM!
➜  ex04
➜  ex04
```

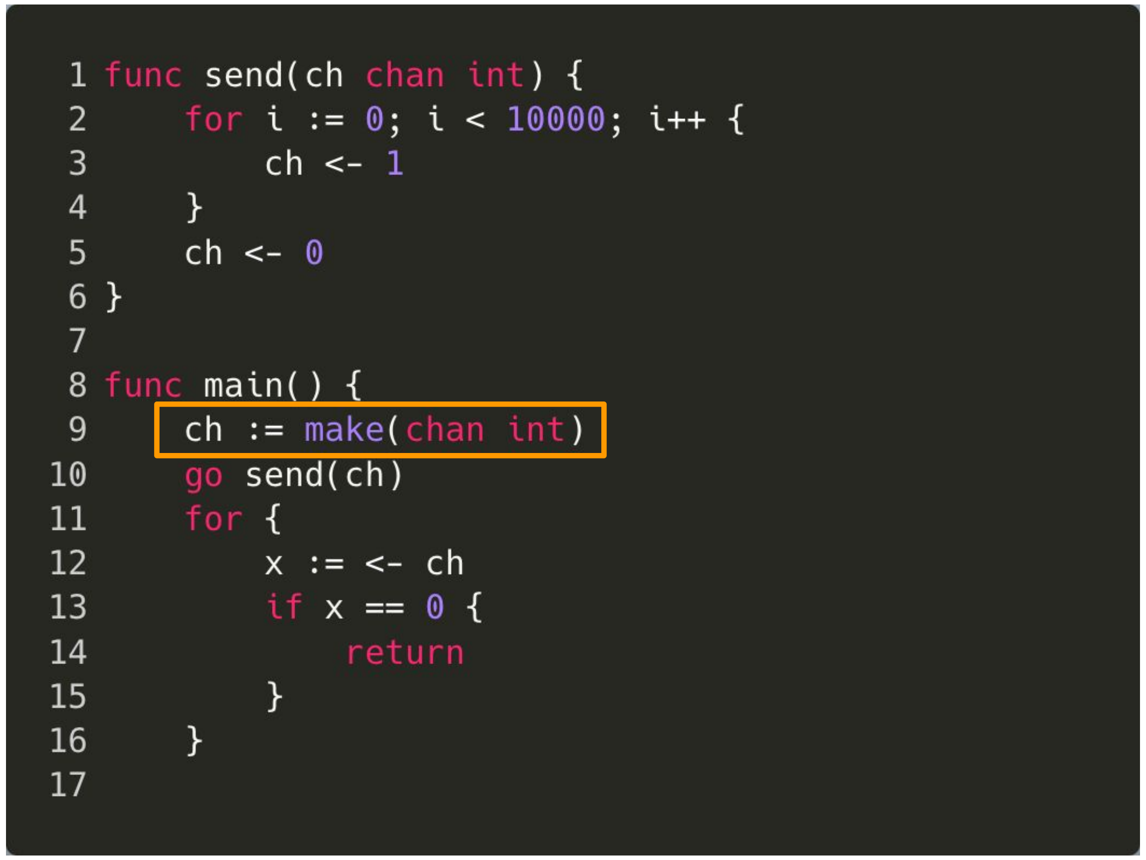
Given Channel Size



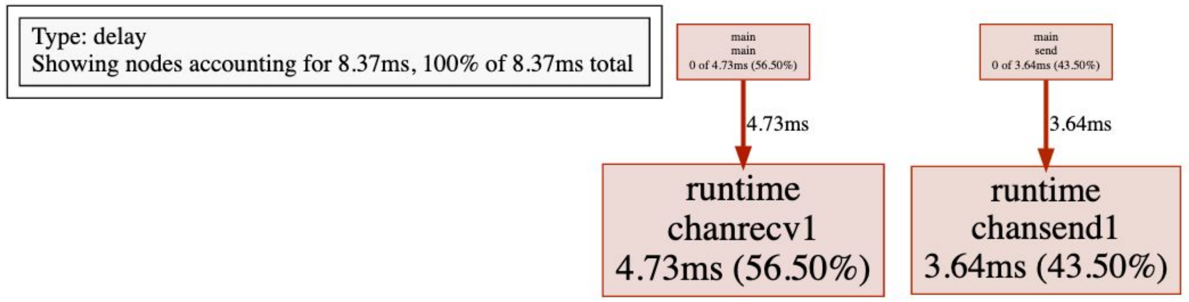
```go
func send(ch chan int) {
    for i := 0; i < 10000; i++ {
        ch <- 1
    }
    ch <- 0
}

func main() {
    ch := make(chan int, 100)
    go send(ch)
    for {
        x := <- ch
        if x == 0 {
            return
        }
    }
}
```



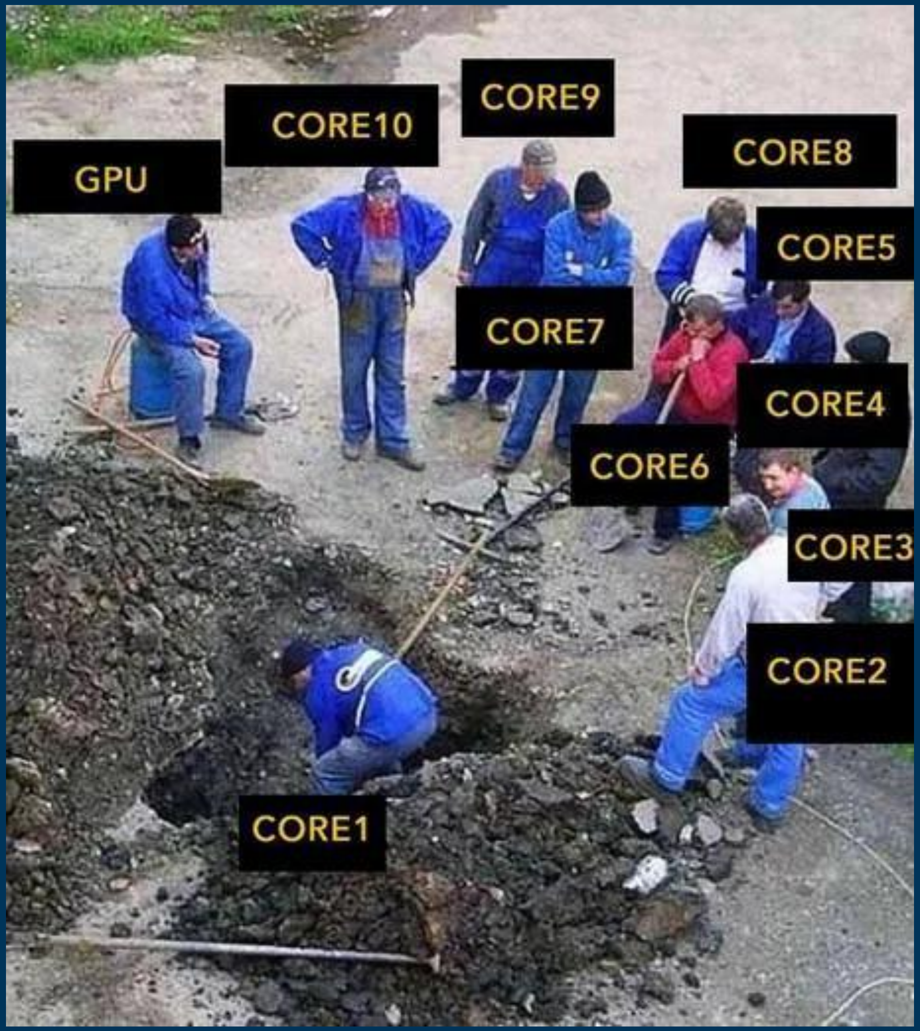
Use Default Channel Size



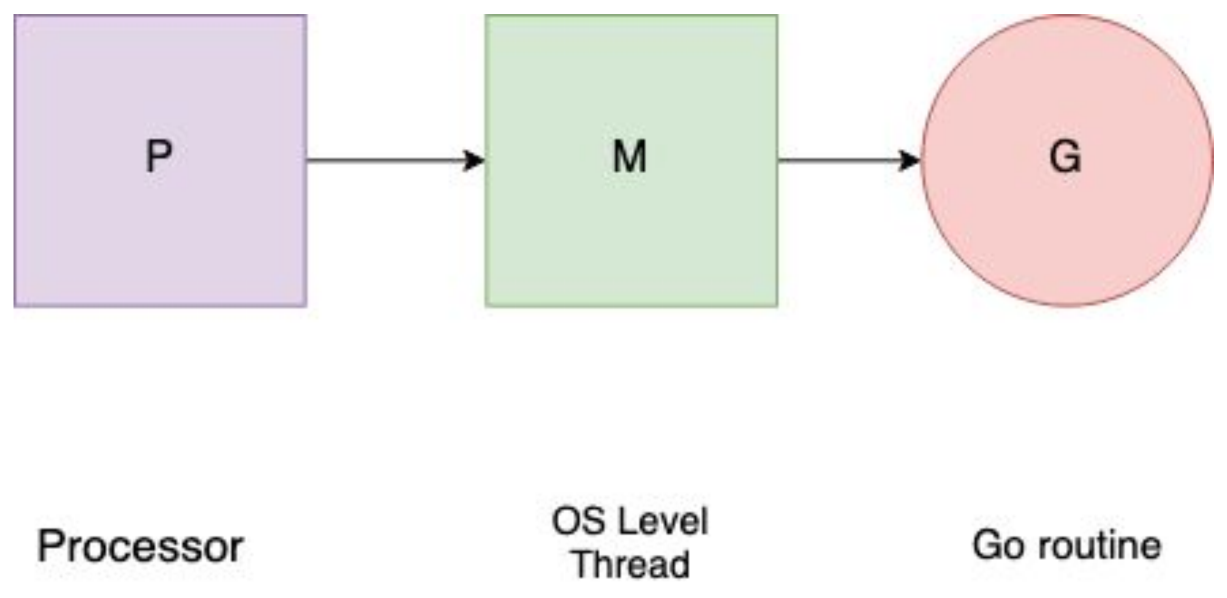
```go
func send(ch chan int) {
    for i := 0; i < 10000; i++ {
        ch <- 1
    }
    ch <- 0
}

func main() {
    ch := make(chan int)
    go send(ch)
    for {
        x := <- ch
        if x == 0 {
            return
        }
    }
```



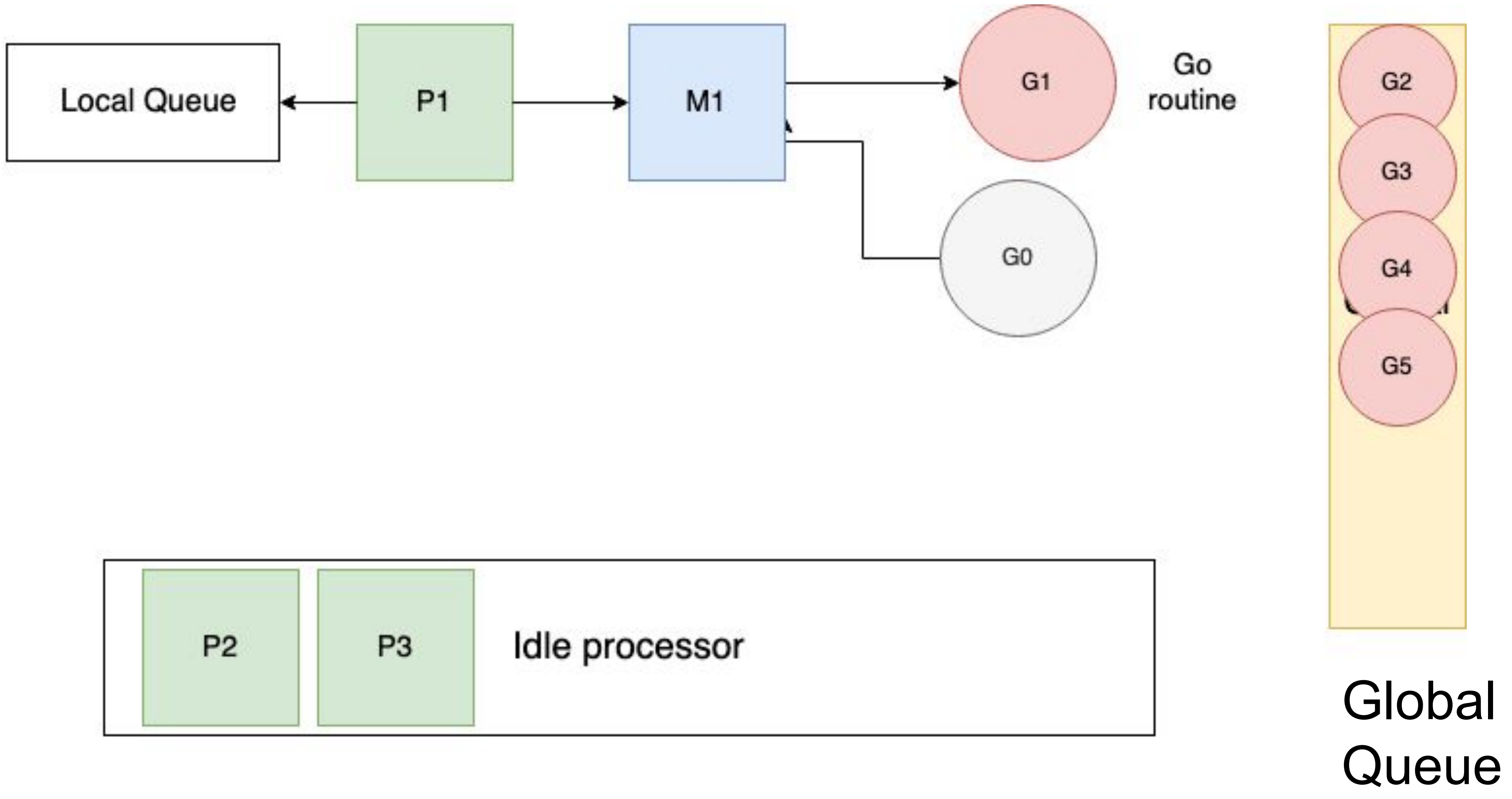
Go Scheduler



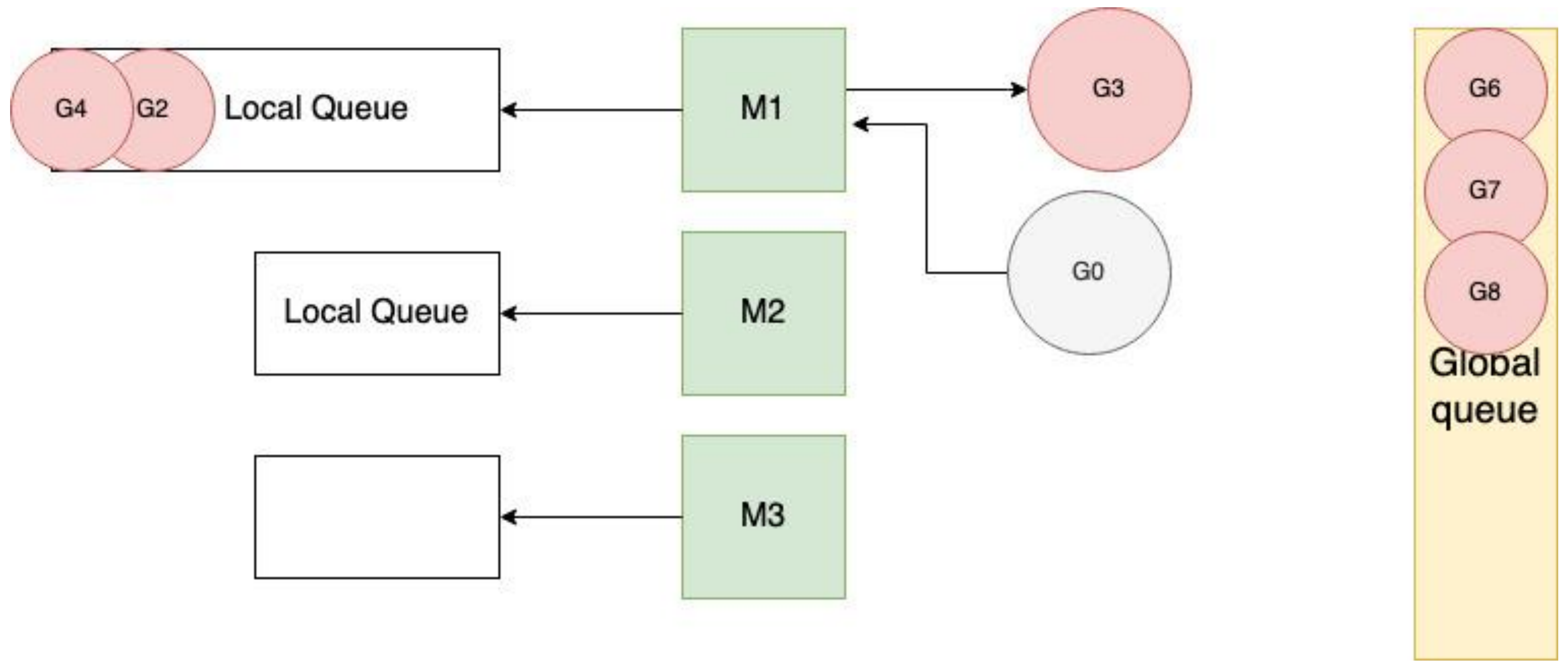
Basic elements



Scheduler Initialization

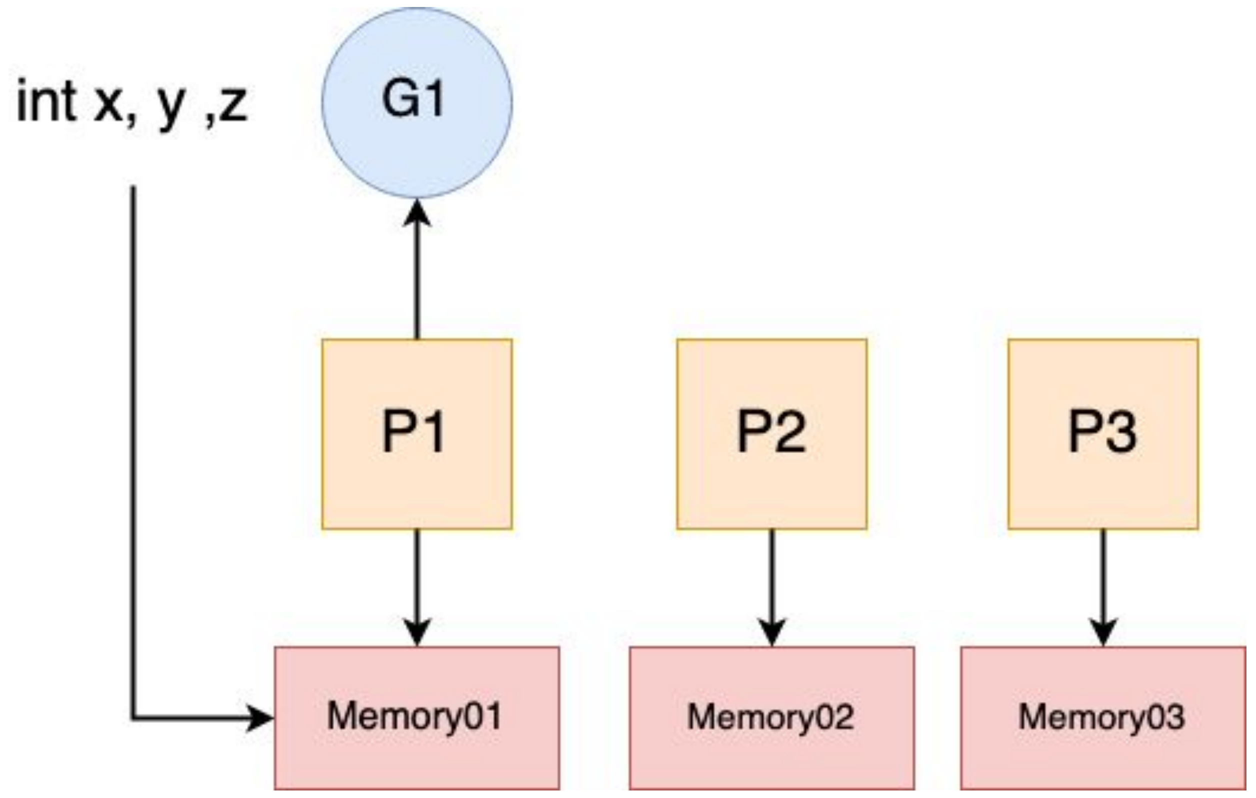


Scheduler (M:N scheduler)

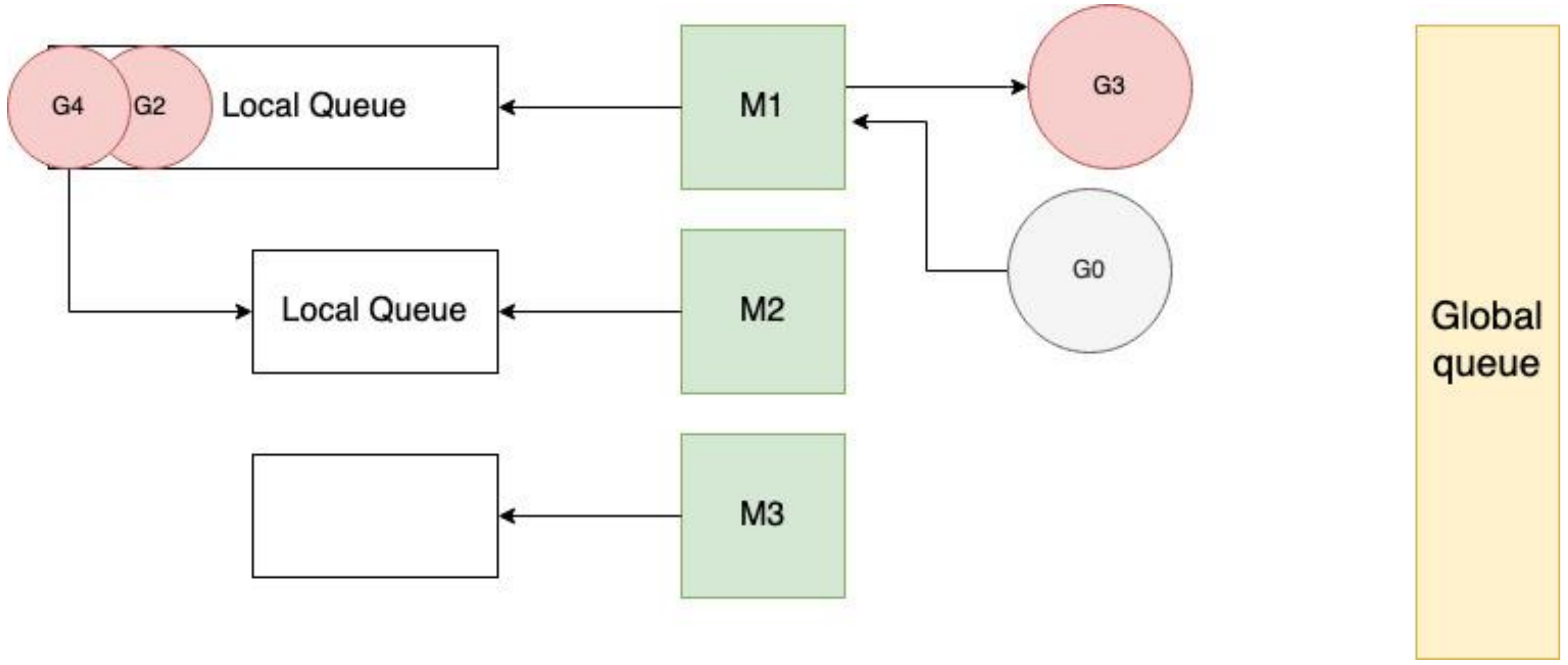


Why local queue and global queue

NEMU (non uniform memory access)



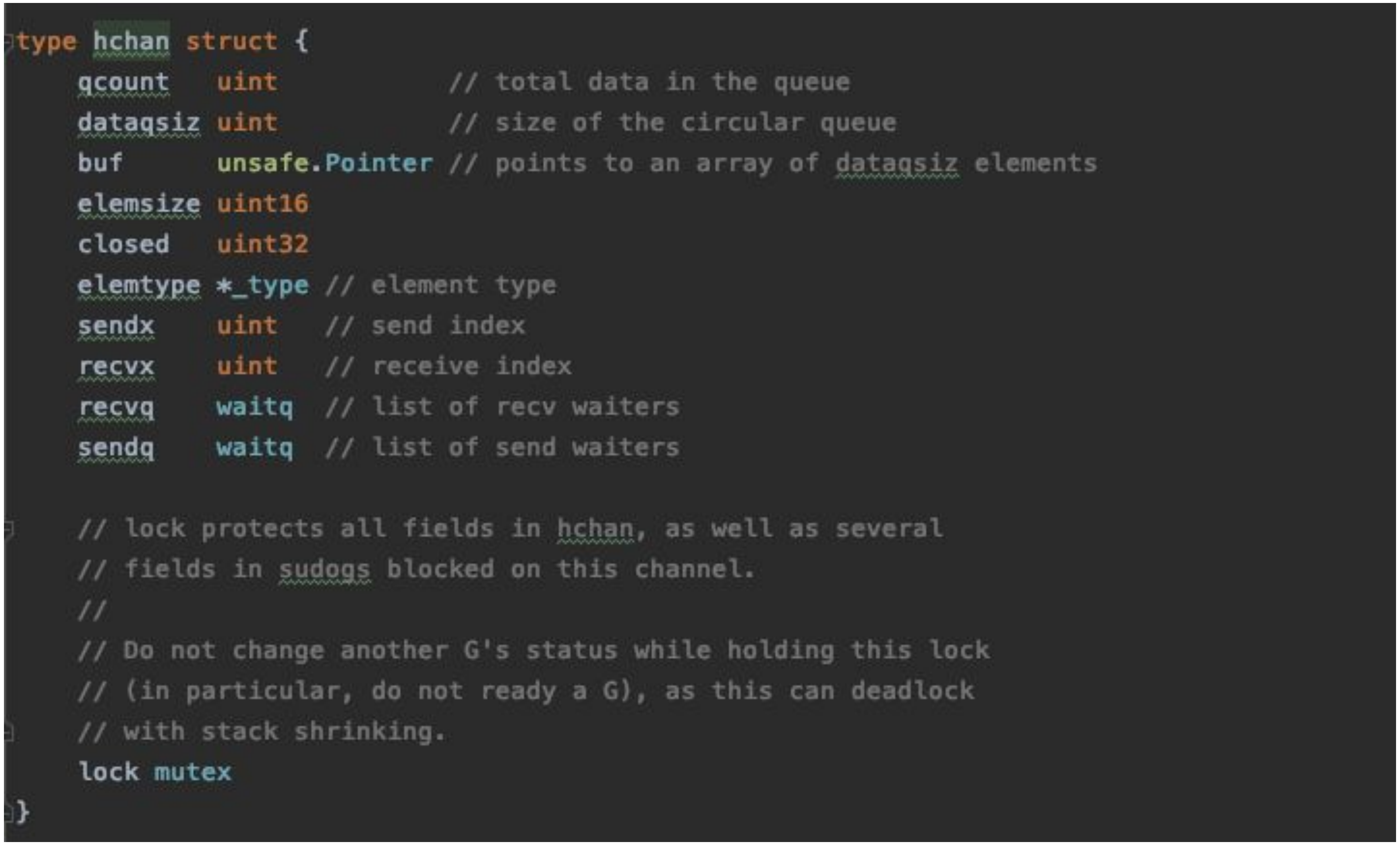
Scheduler work stealing



Dive Into Channel Implementation

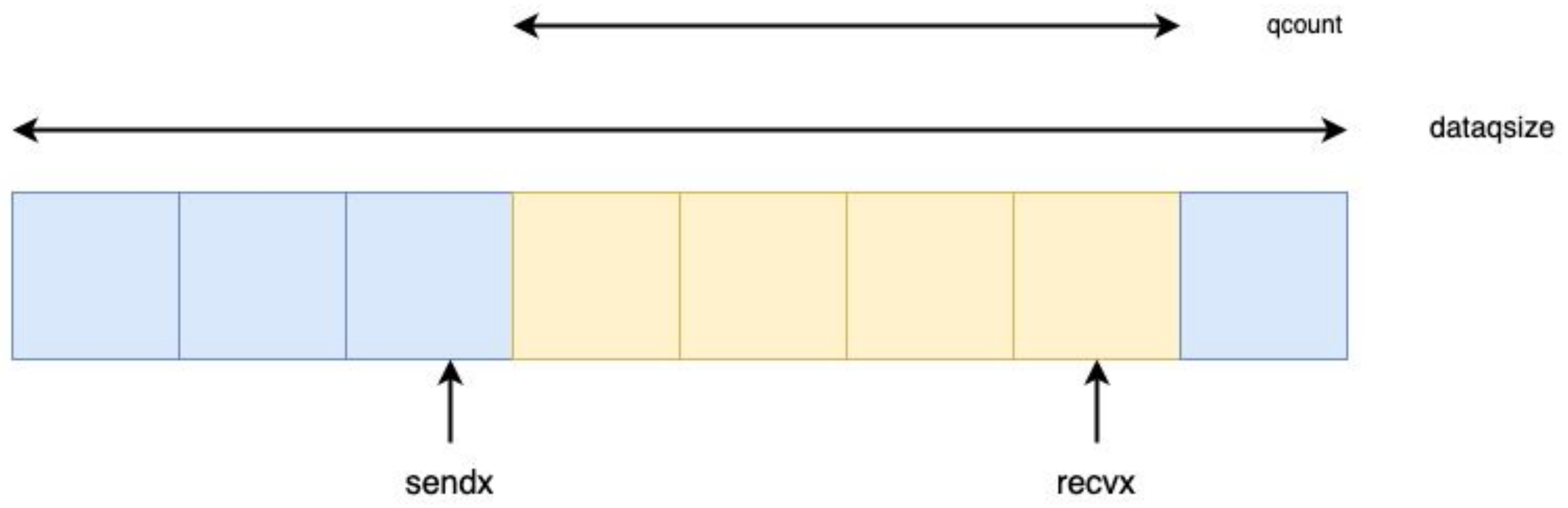
有些程式compile不過,
有些程式compile過了不會動
,
還是別想談愛情了,
你老闆在你後面很火。

hchan (src/runtime/chan.go)

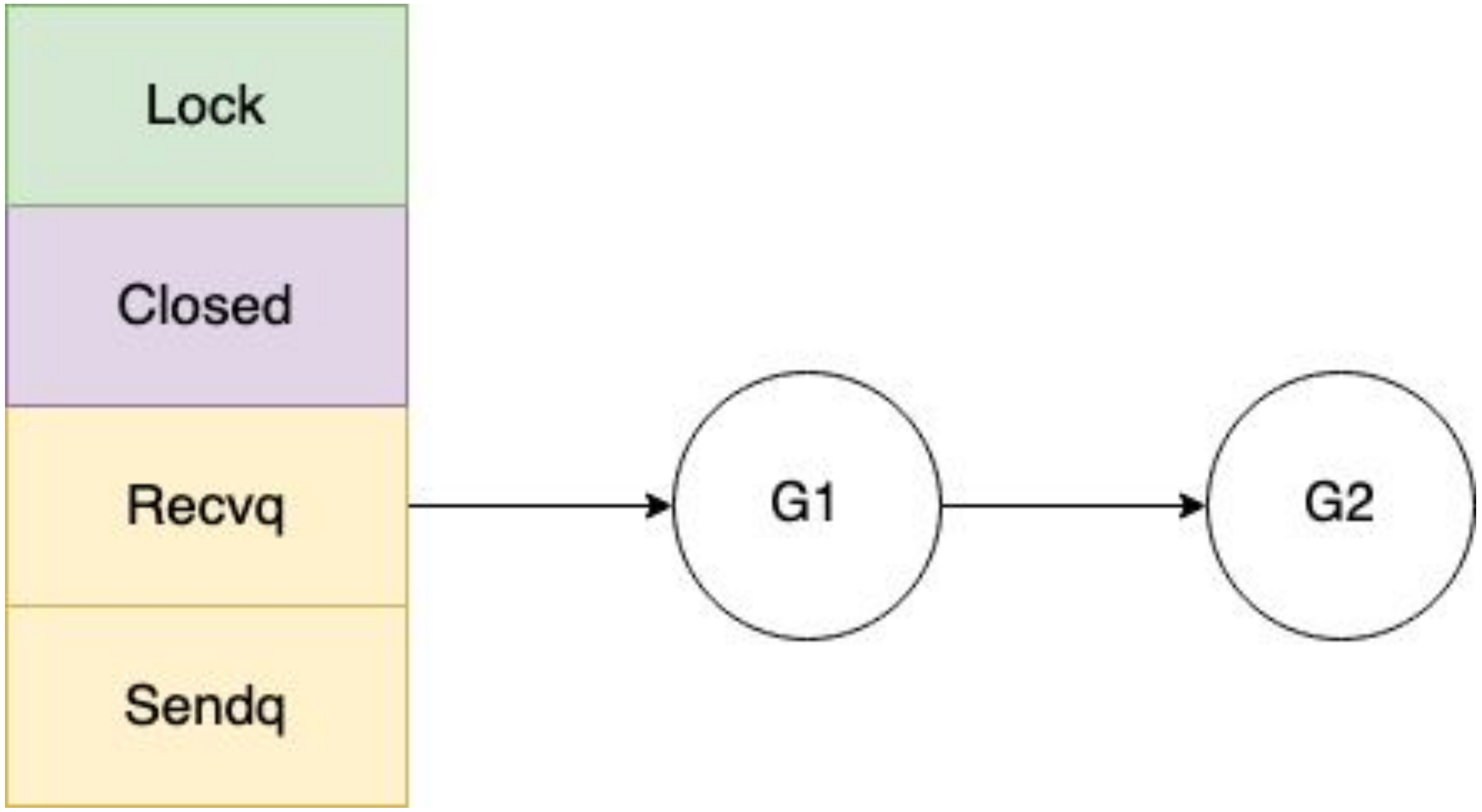


```go
type hchan struct {
    qcount   uint           // total data in the queue
    dataqsiz uint           // size of the circular queue
    buf      unsafe.Pointer // points to an array of dataqsiz elements
    elemsize uint16
    closed   uint32
    elemtype *_type // element type
    sendx    uint   // send index
    recvx    uint   // receive index
    recvq    waitq  // list of recv waiters
    sendq    waitq  // list of send waiters

    // lock protects all fields in hchan, as well as several
    // fields in sudogs blocked on this channel.
    //
    // Do not change another G's status while holding this lock
    // (in particular, do not ready a G), as this can deadlock
    // with stack shrinking.
    lock mutex
}
```

Sendx, Recvx, Buf

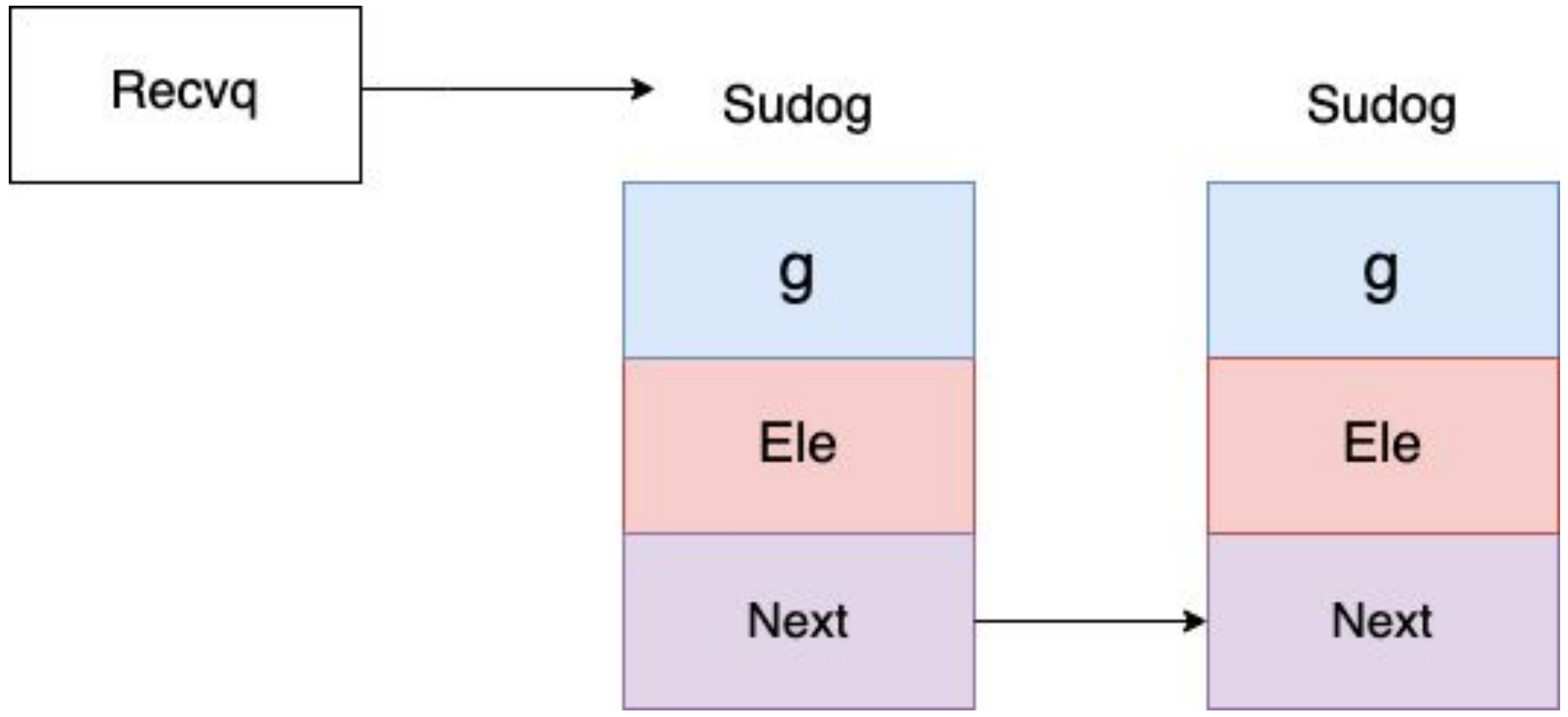


Recvq, Sendq, Closed, Lock

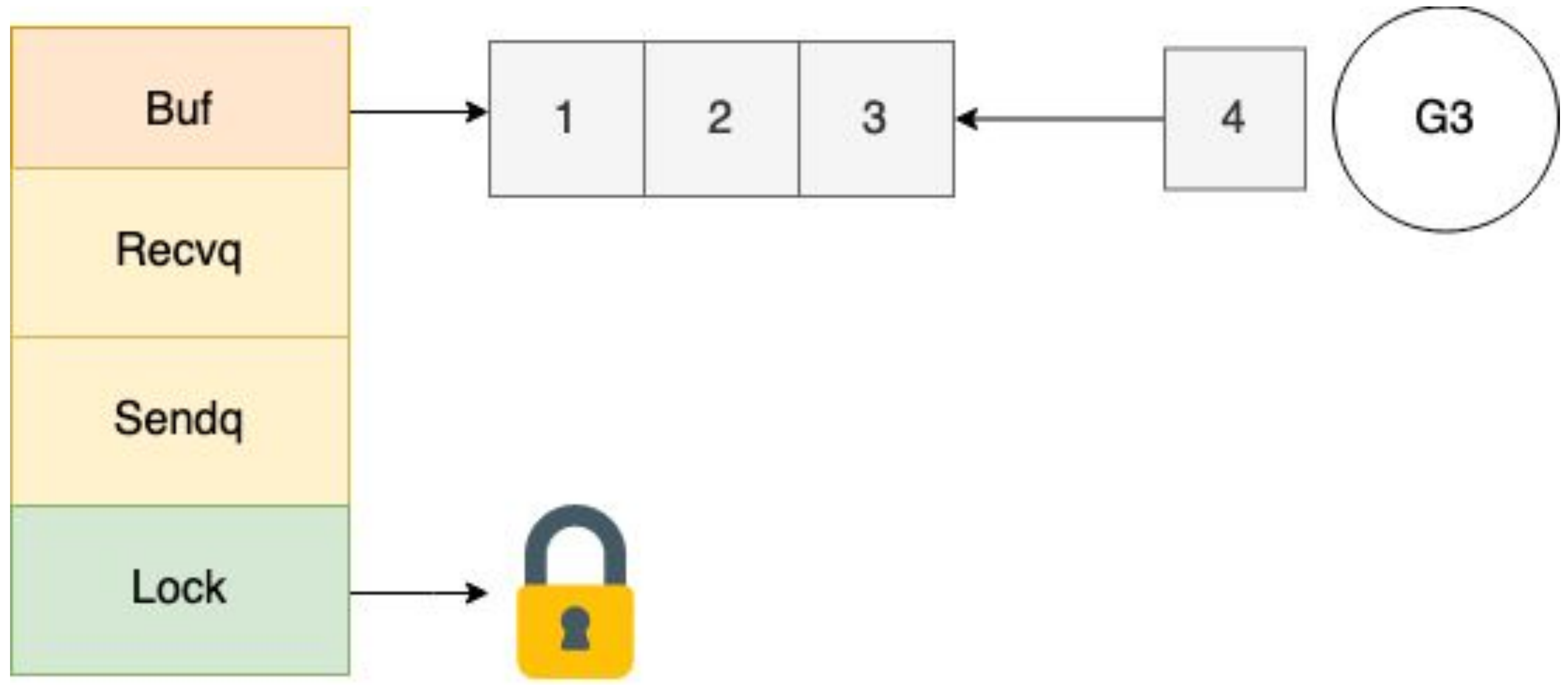


Waiting Sender

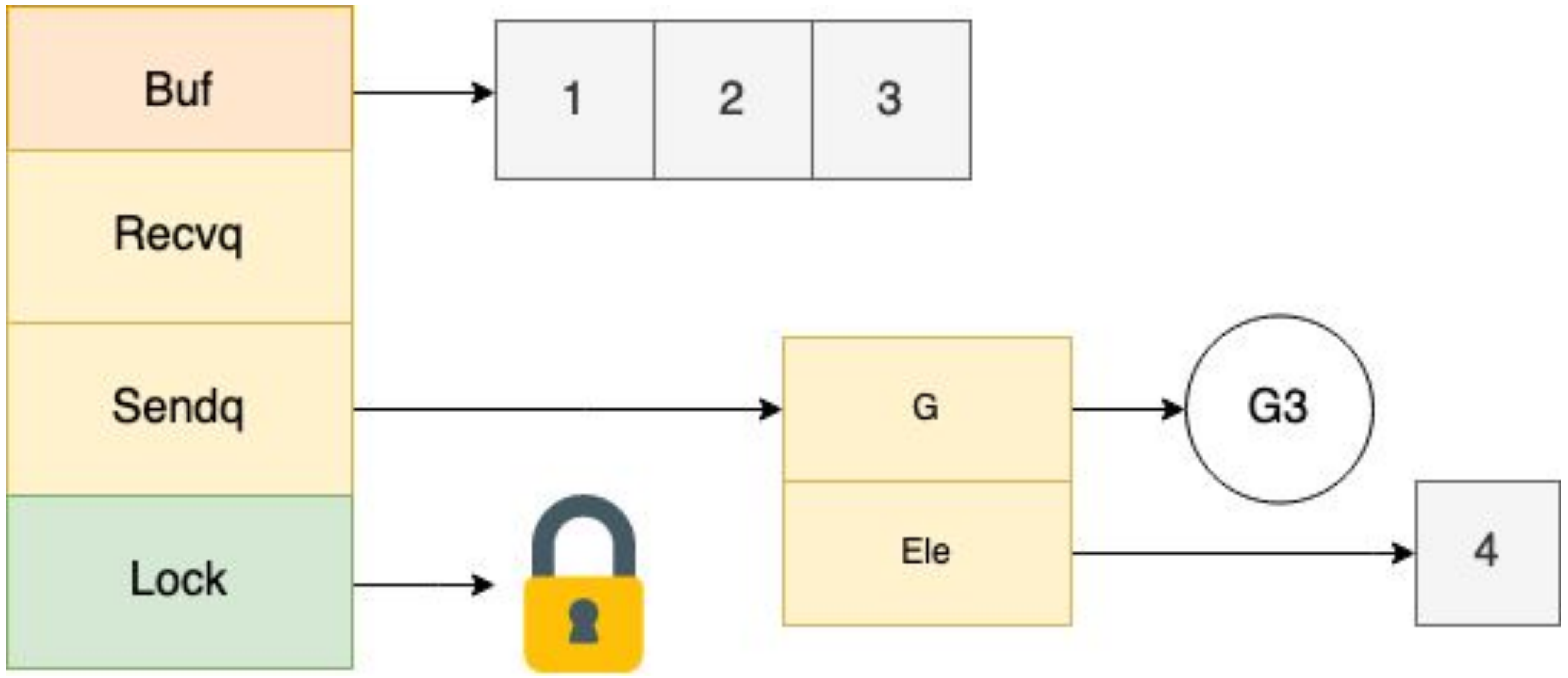
- The buffer is empty so gopark and enqueue to recvq



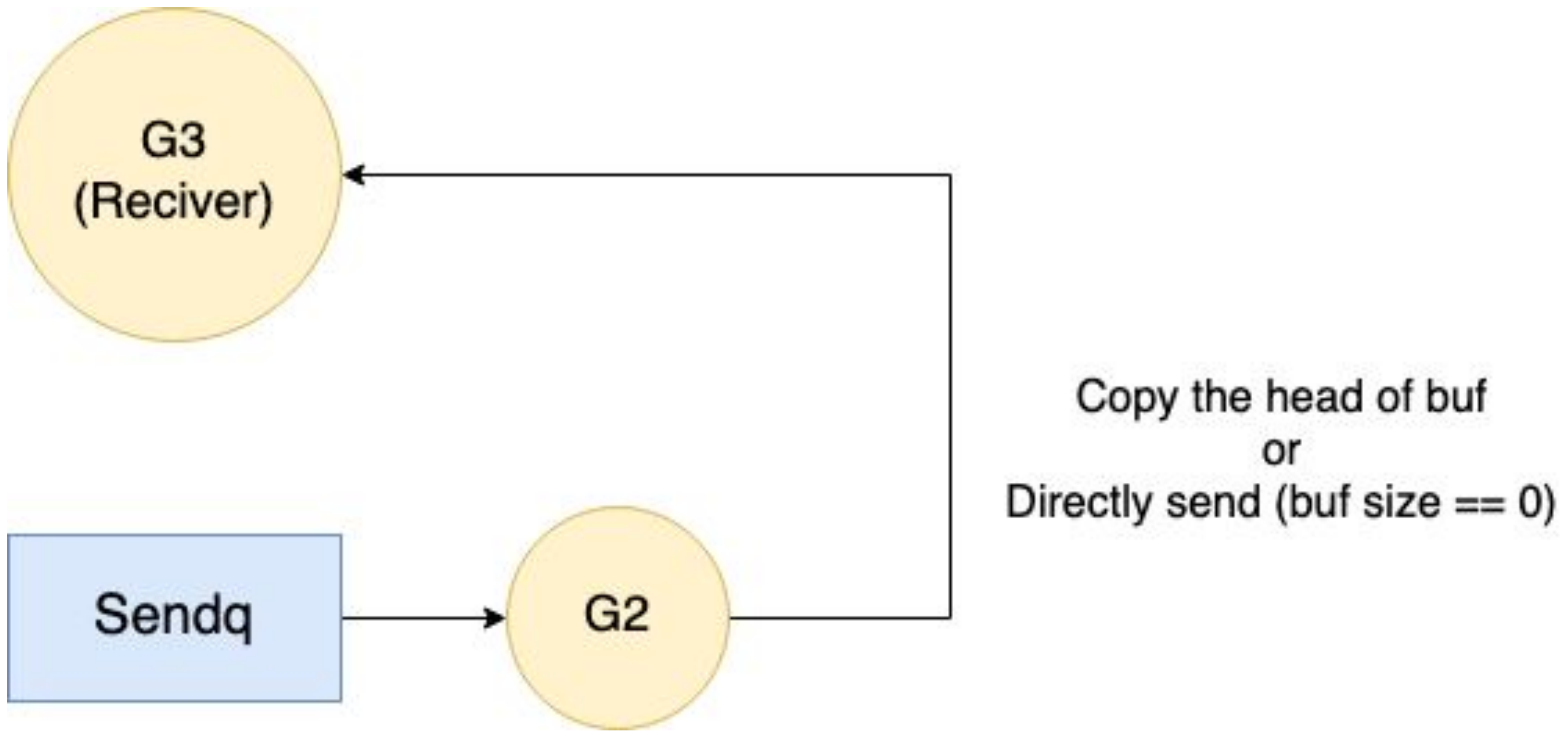
Sending Value Into Full Channel



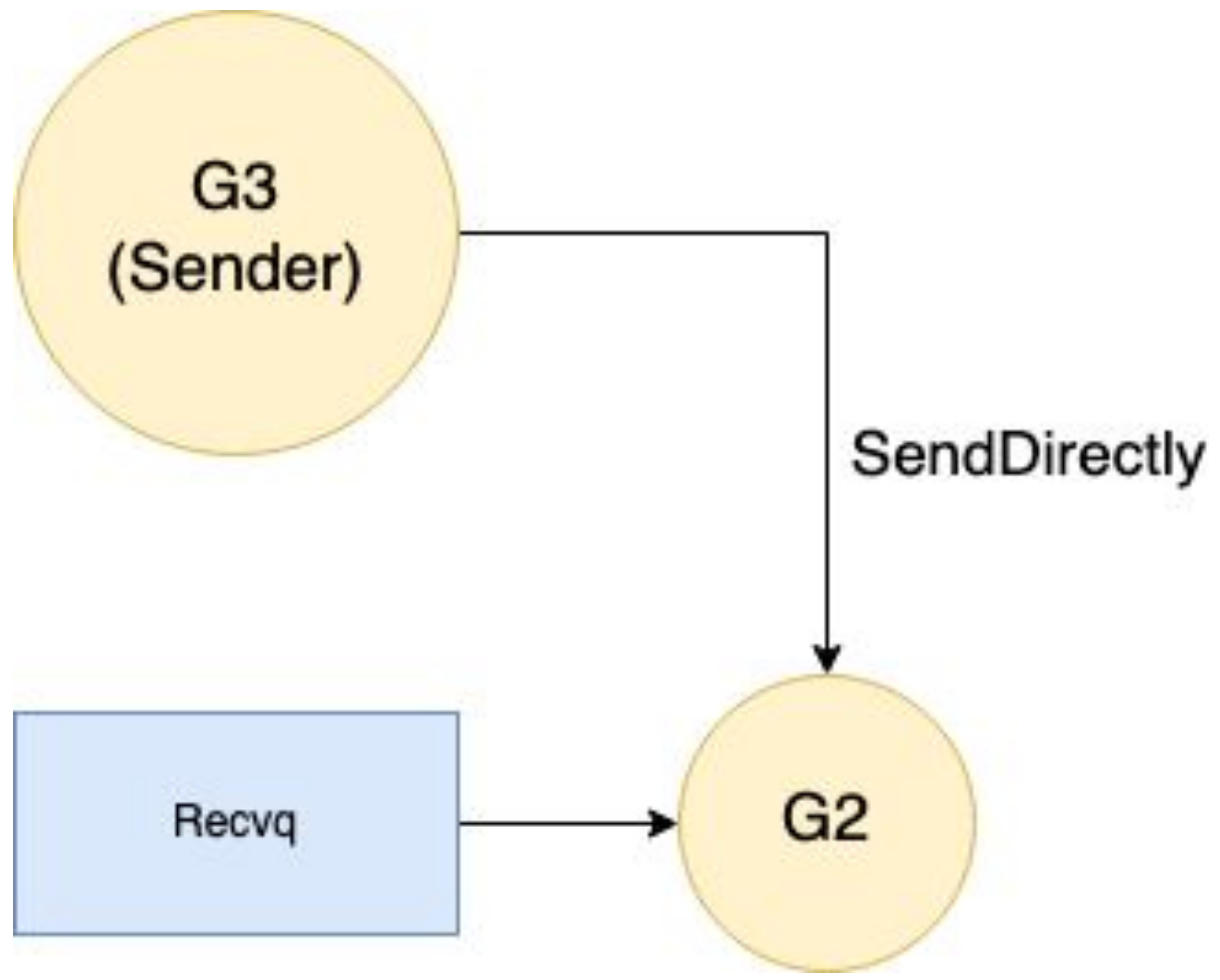
Send Waiting Queue



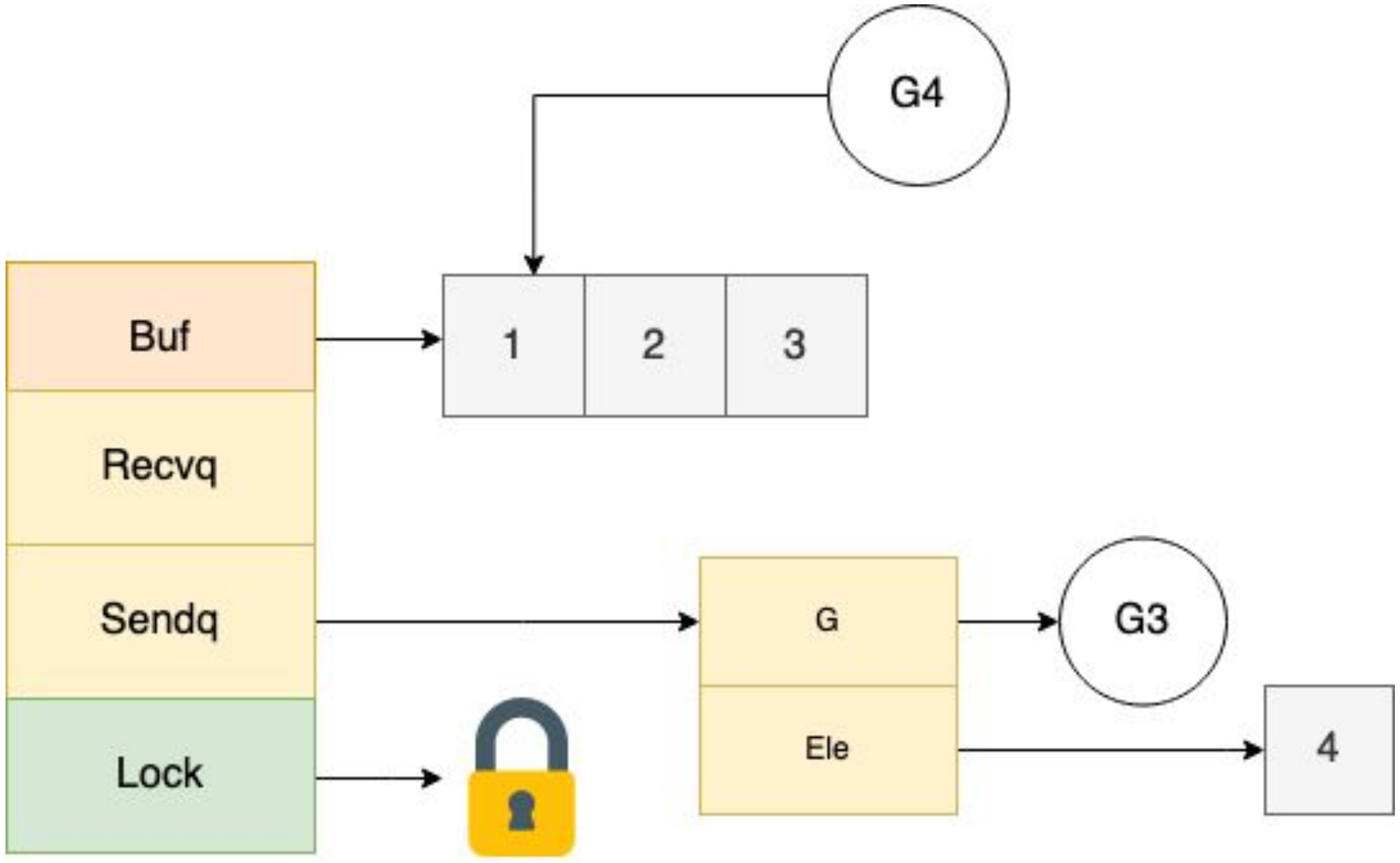
Recv from waiting queue



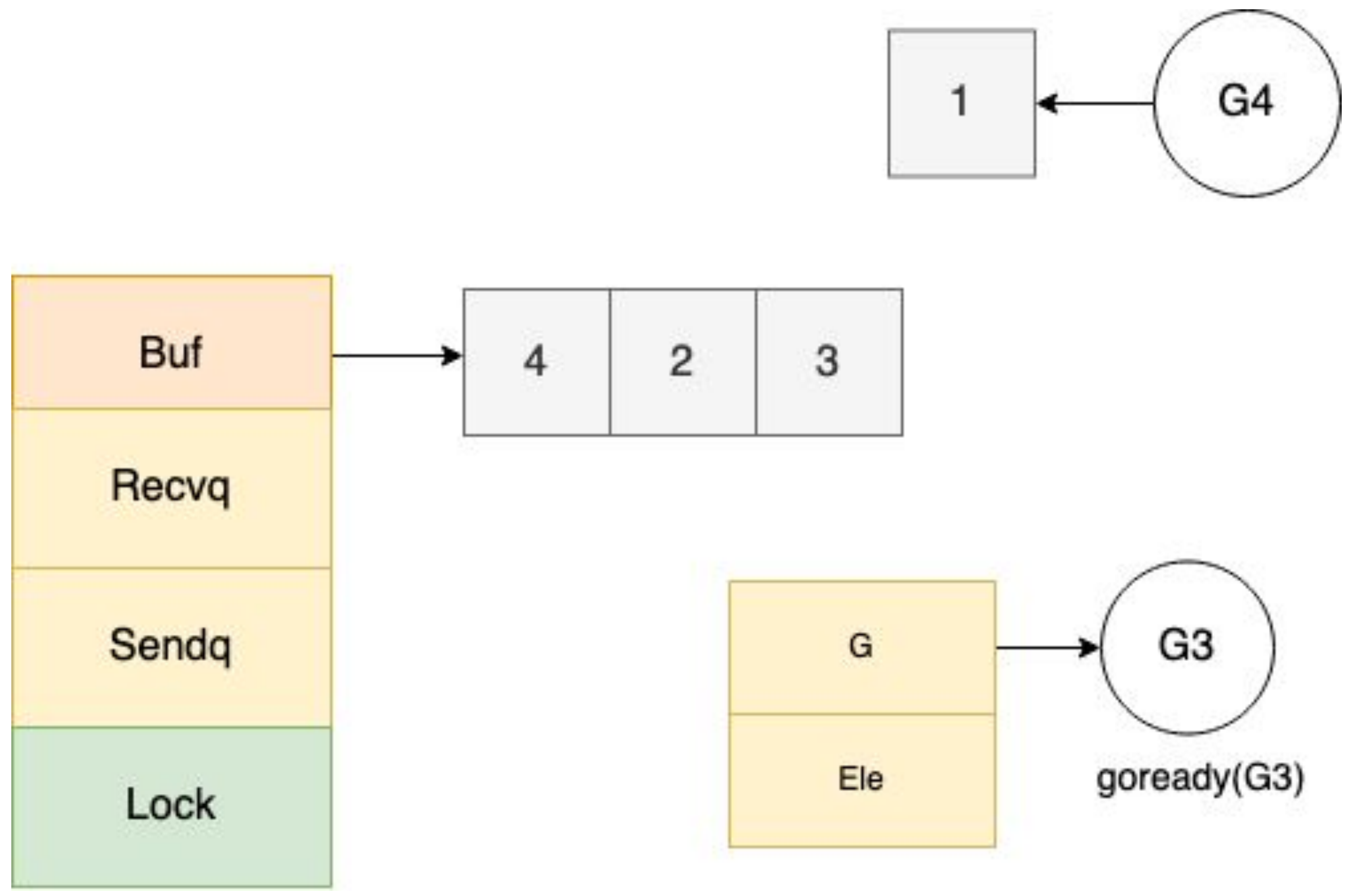
Send directly to waiting queue



New Receiver Come In

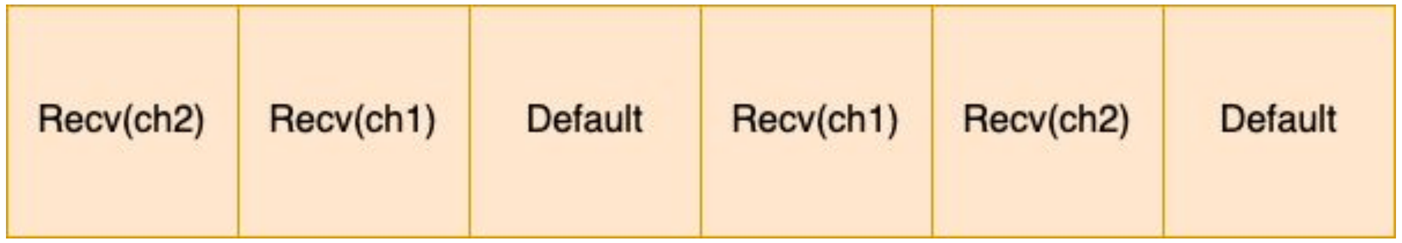


Receive Finished

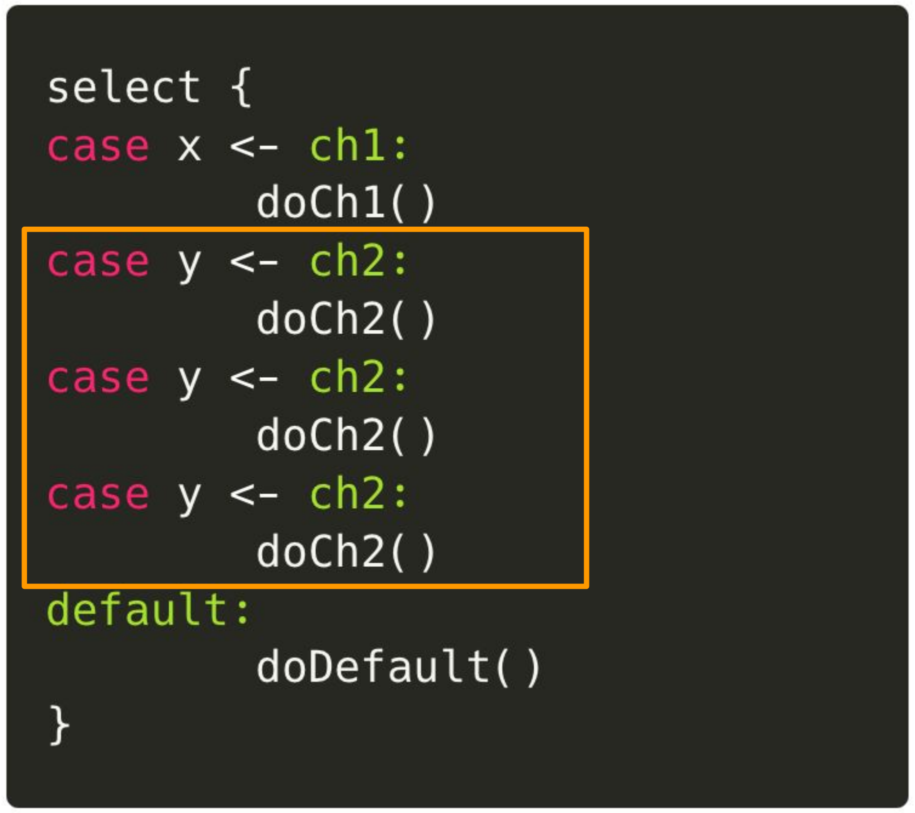


Select Scheduler

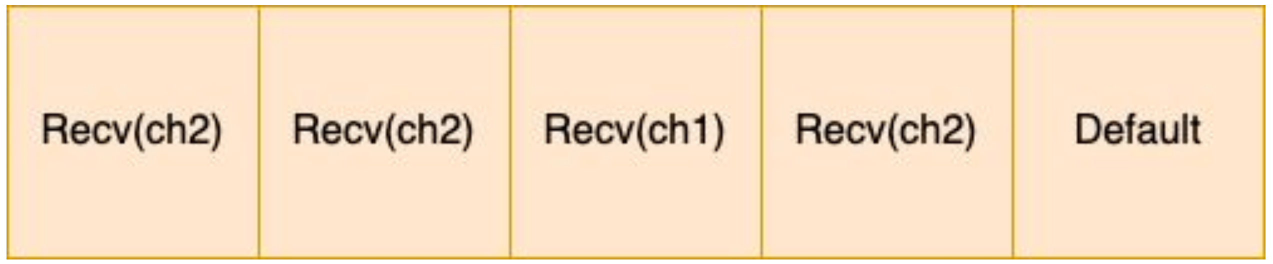
```
select {
case x <- ch1:
    doCh1()
case y <- ch2:
    doCh2()
default:
    doDefault()
}
```



Select Scheduler

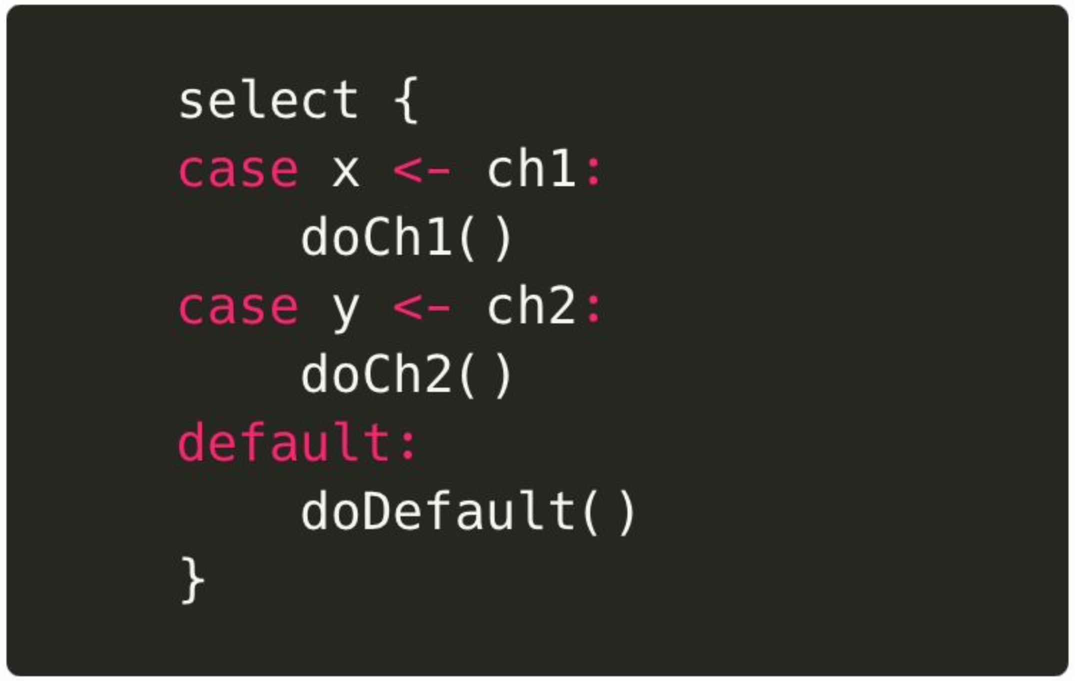


```
select {
case x <- ch1:
        doCh1()
case y <- ch2:
        doCh2()
case y <- ch2:
        doCh2()
case y <- ch2:
        doCh2()
default:
        doDefault()
}
```



Close fast forward on non-blocking recv

- If the channel is closed and empty, return false immediately.



```
select {
case x <- ch1:
    doCh1()
case y <- ch2:
    doCh2()
default:
    doDefault()
}
```

Use Channel

- Max concurrent control by buffer size
- Producer and consumer pattern

Q & A

Contact:
gaston.qiu@umbocv.com
Github: gastonqiu

Reference

The special goroutine go:
https://medium.com/a-journey-with-go/go-g0-special-goroutine-8c778c6704d8

GopherCon 2017: Kavya Joshi - Understanding Channels:
https://www.youtube.com/watch?v=KBZlN0izeiY

Channel & select 源码分析【 Go 夜读 】
https://www.youtube.com/watch?v=d7fFCGGn0Wc&t=2268s

Channel source code
https://github.com/golang/go/blob/master/src/runtime/chan.go

Go: Asynchronous Preemption
https://medium.com/a-journey-with-go/go-asynchronous-preemption-b5194227371c

Go: Ordering in Select Statements
https://medium.com/a-journey-with-go/go-ordering-in-select-statements-fd0ff80fd8d6

Go: Goroutine, OS Thread and CPU Management
https://medium.com/a-journey-with-go/go-goroutine-os-thr

Reference

Go: Buffered and Unbuffered Channels

https://medium.com/a-journey-with-go/go-buffered-and-unbuffered-channels-29a107c00268

Go Scheduler 源码阅读【 Go 夜读 】

https://www.youtube.com/watch?v=B-ozWjqnX24&t=543s

Go tour concurrency

https://tour.golang.org/concurrency/2

Communicating sequential processes

https://levelup.gitconnected.com/communicating-sequential-processes-csp-for-go-developer-in-a-nutshell-866795eb879d