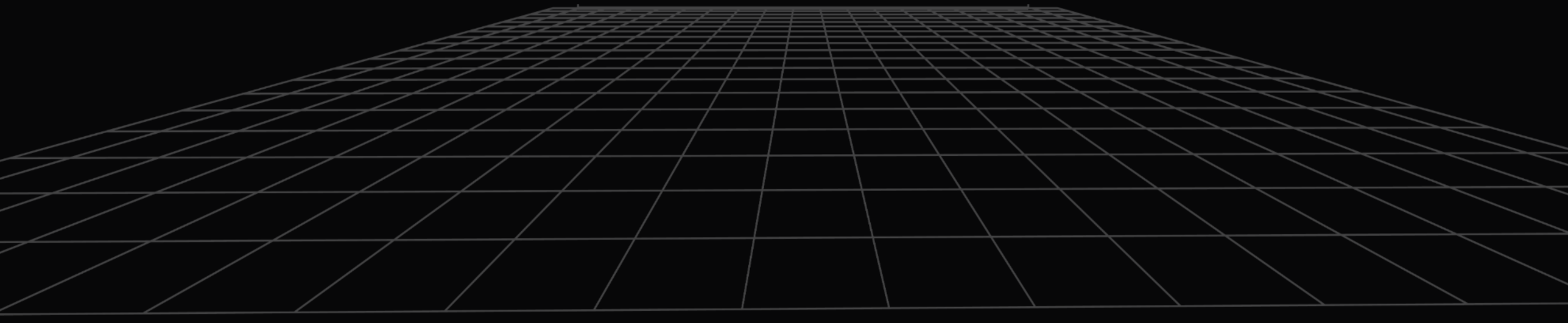
КАНАЛЫ

Balun.Courses

ПРОВЕРЬ ЗАПИСЬ

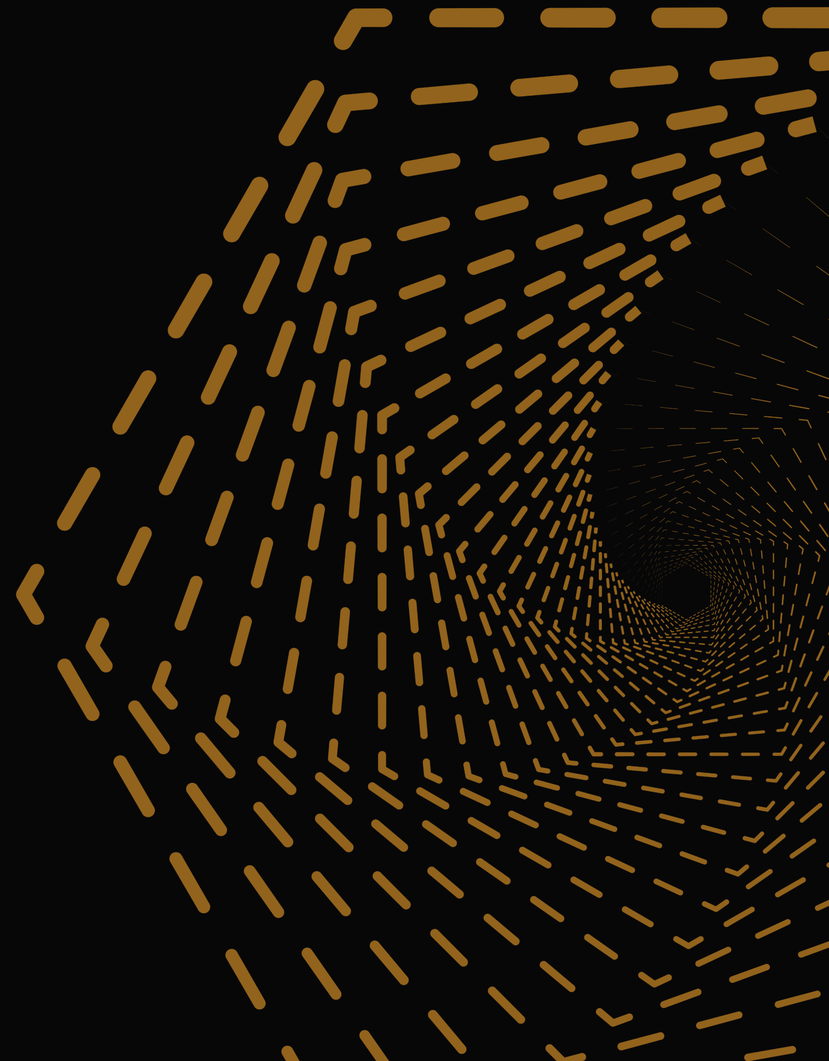


ПРАВИЛА ЗАНЯТИЯ

1. вопросы в чате можно задавать **в любое время**

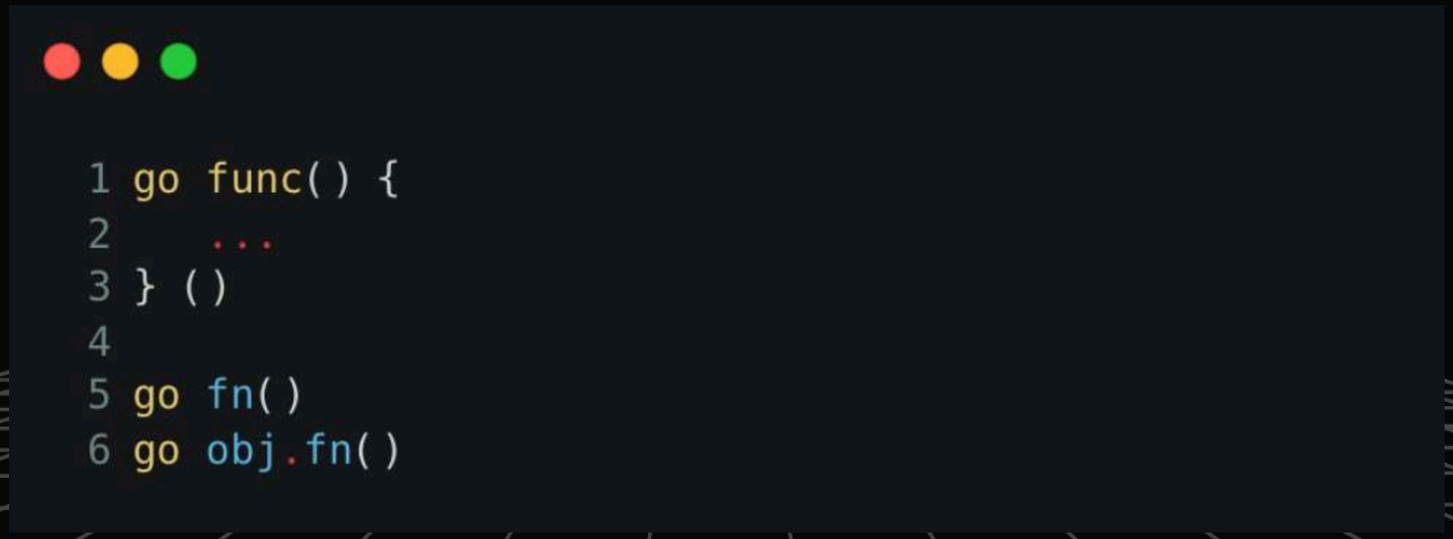
2. вопросы голосом задаем **по поднятой руке в Zoom**

3. ответы на вопросы будут **в запланированных местах**



КАНАЛЫ

КАК ПОЛУЧИТЬ КАКИЕ-ЛИБО ДАННЫЕ ИЗ ГОРУТИНЫ?



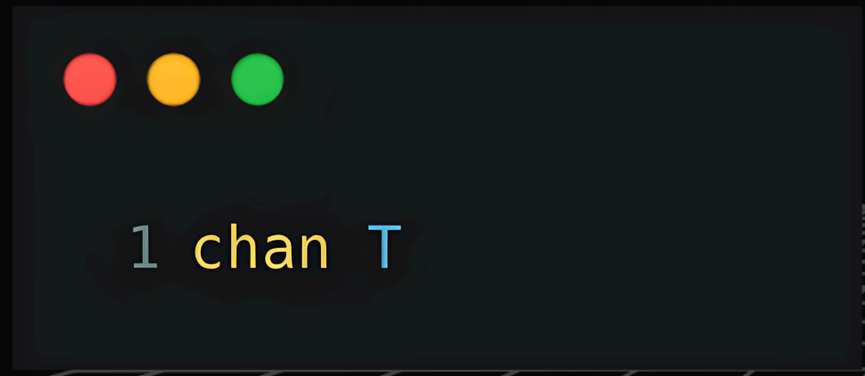
```
1 go func() {
2     ...
3 } ()
4
5 go fn()
6 go obj.fn()
```

КАНАЛЫ

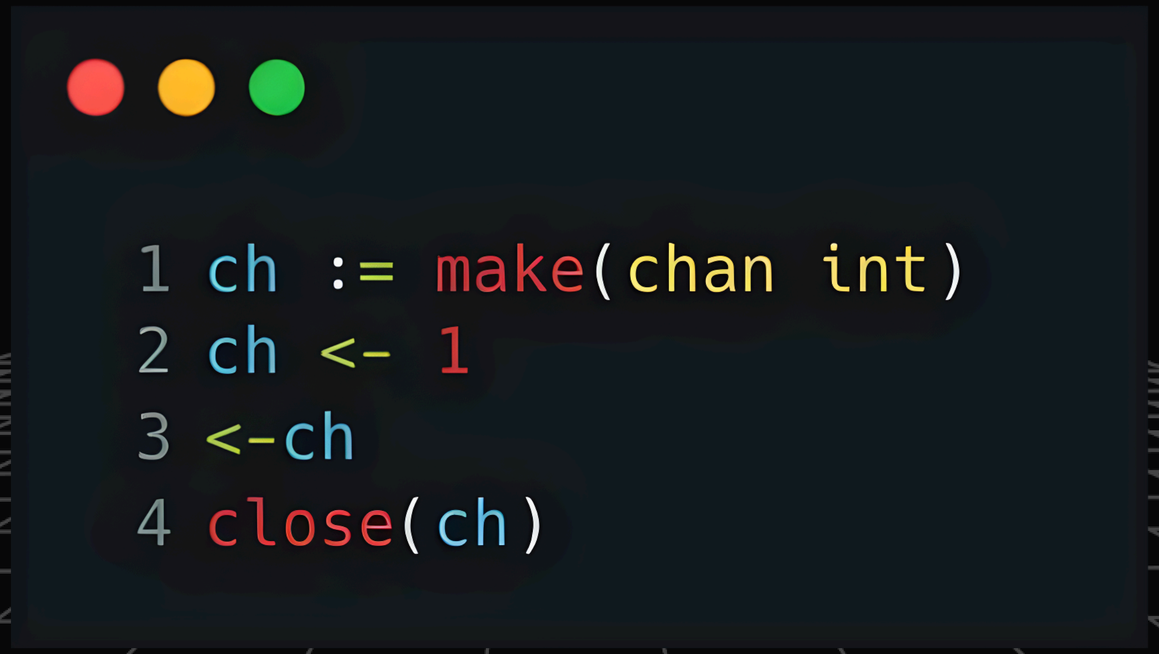
КАНАЛЫ

Некие **примитивы,**
в которые можно писать
и из которых можно читать:

- блокируют/разблокируют горутины
- работают с конкретным типом
- похожи на очереди **FIFO**
- **goroutine-safe**



API



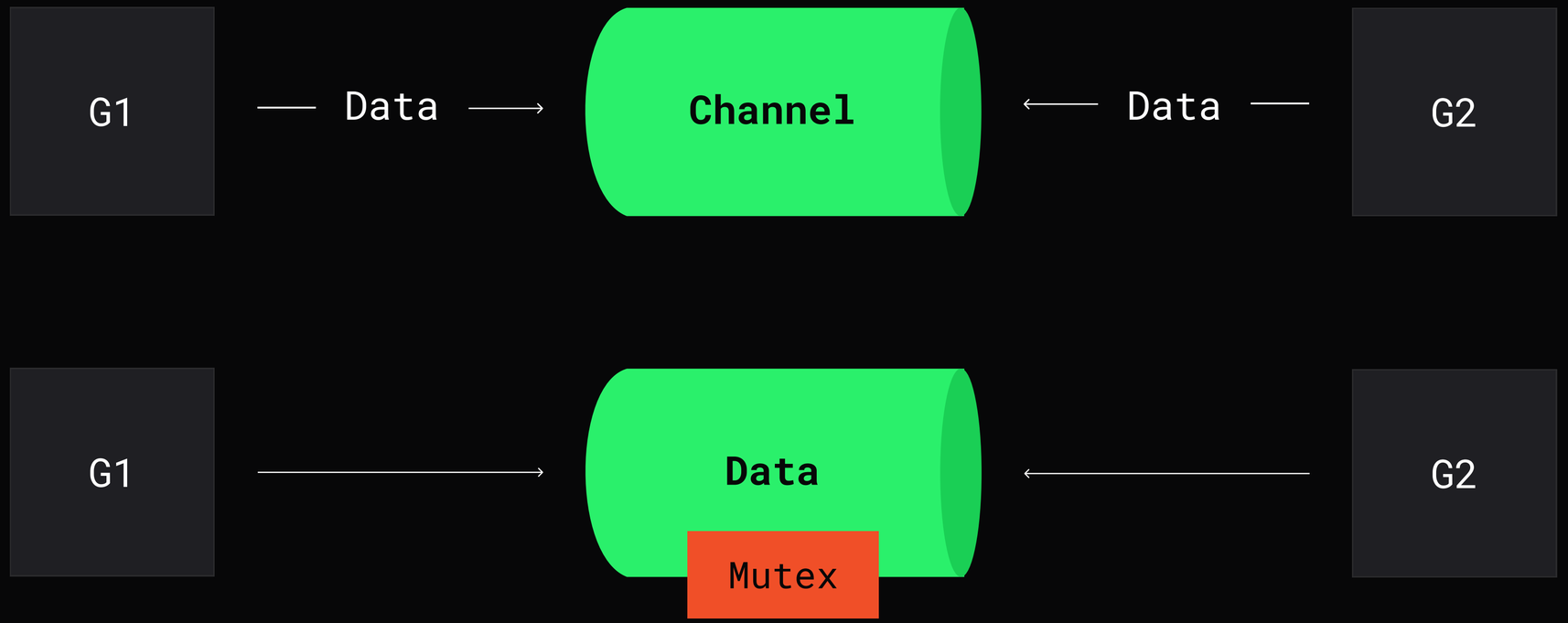
```
1 ch := make(chan int)
2 ch <- 1
3 <-ch
4 close(ch)
```

EXAMPLE

Channel interaction

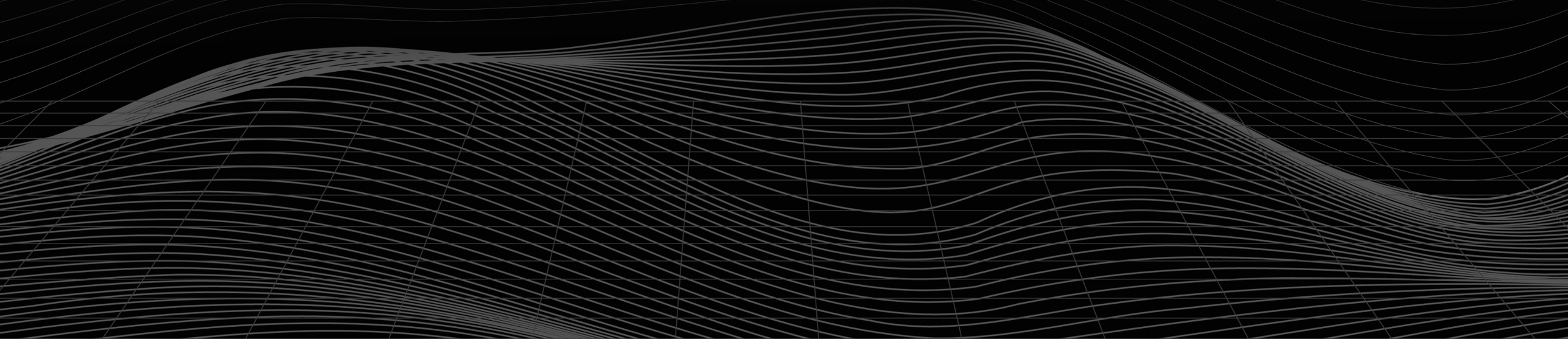
Don’t communicate by sharing memory;
share memory by communicating

(не общайтесь разделением памяти,
разделяйте память через общение)


G1
Data
Channel
Data
G2
G1
Data
Mutex
G2

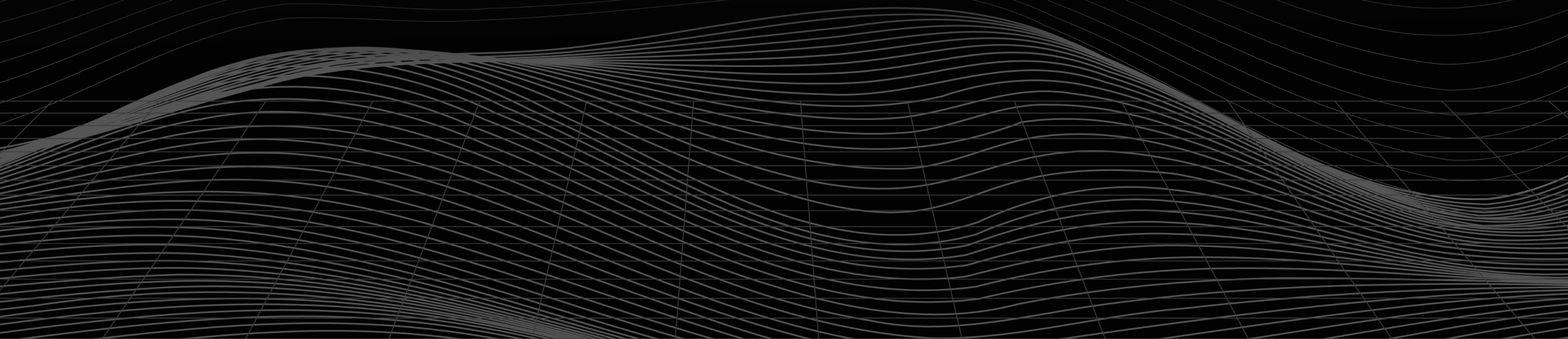
EXAMPLE

Increment with mutex



EXAMPLE

Increment with channel



Terminal: question + 

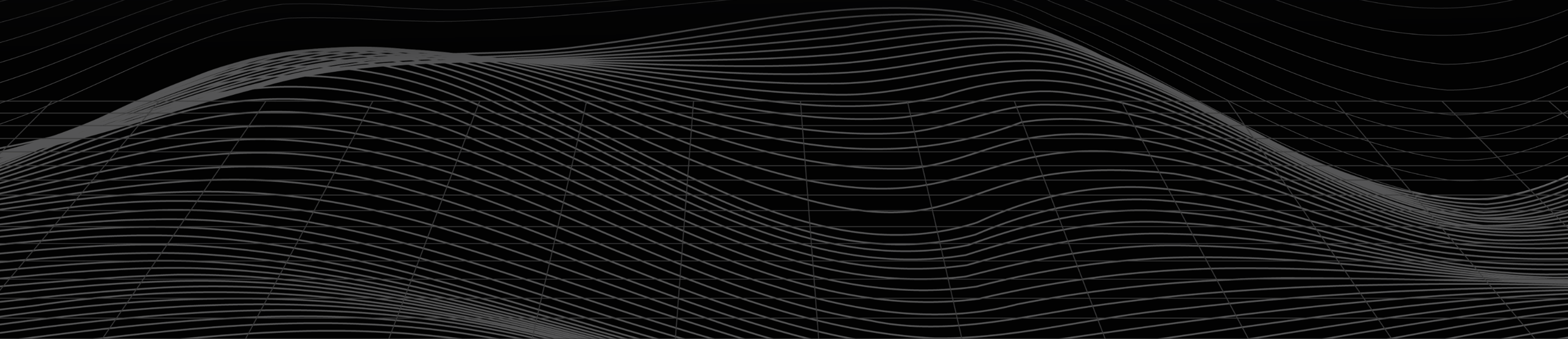
КАК ЖДАТЬ ДАННЫЕ ИЗ НЕСКОЛЬКИХ КАНАЛОВ?

EXAMPLE

Select

EXAMPLE

Select with break and continue



EXAMPLE

Producer and consumer

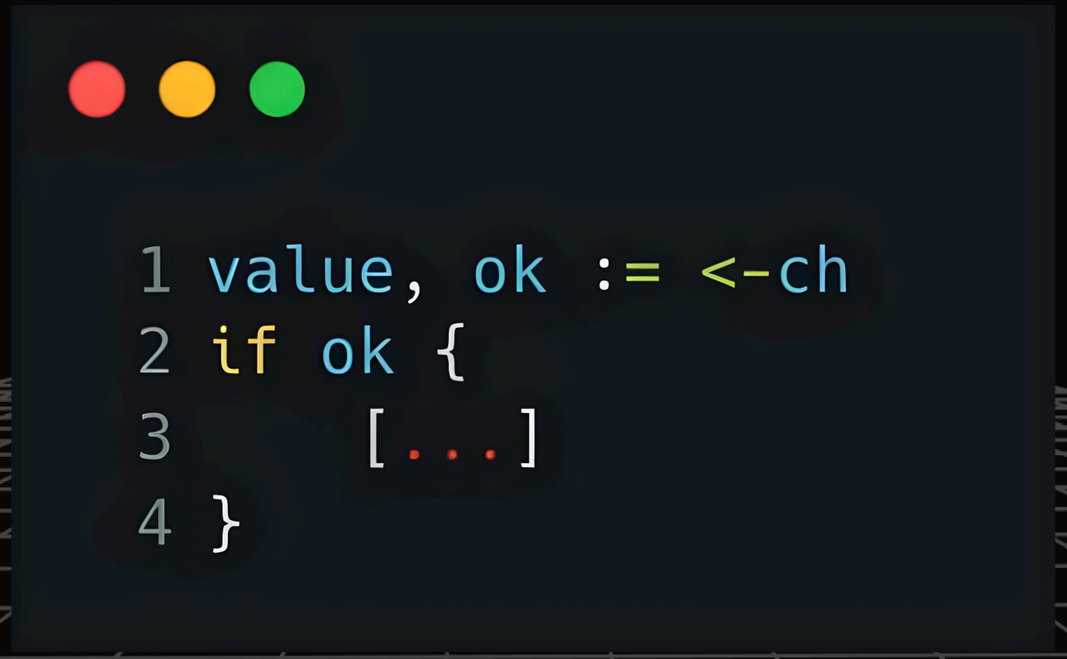
EXAMPLE

Signals

EXAMPLE

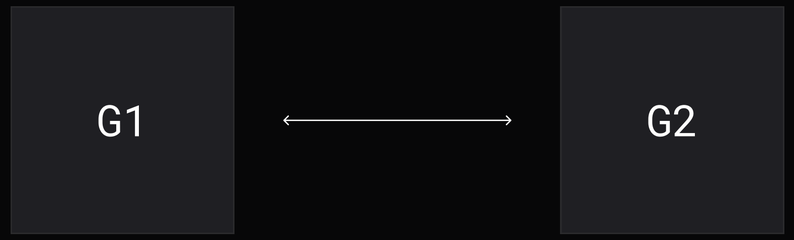
Broadcast

МОЖНО ПРОВЕРИТЬ ЗАКРЫТ КАНАЛ ИЛИ НЕТ



НЕБУФЕРИЗОВАННЫЕ КАНАЛЫ (СИНХРОННЫЕ)

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



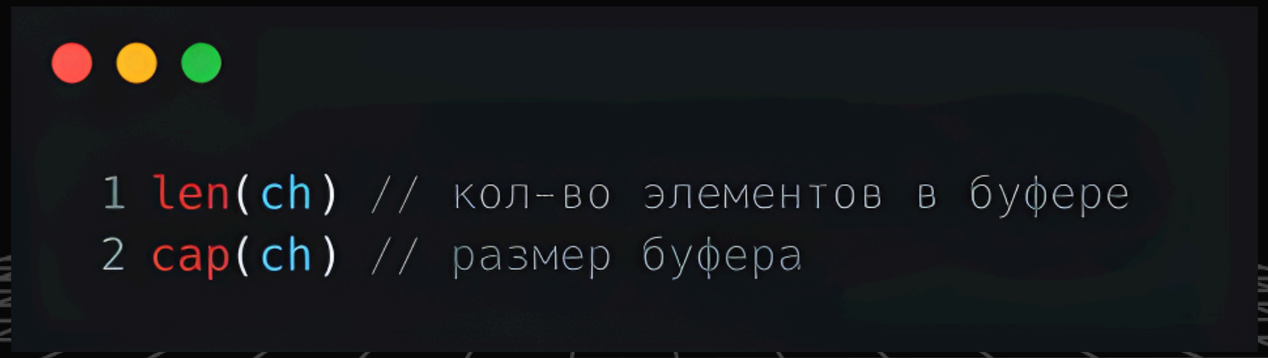
БУФЕРИЗОВАННЫЕ КАНАЛЫ (АСИНХРОННЫЕ)

```
1 ch := make(chan int, 4)
```

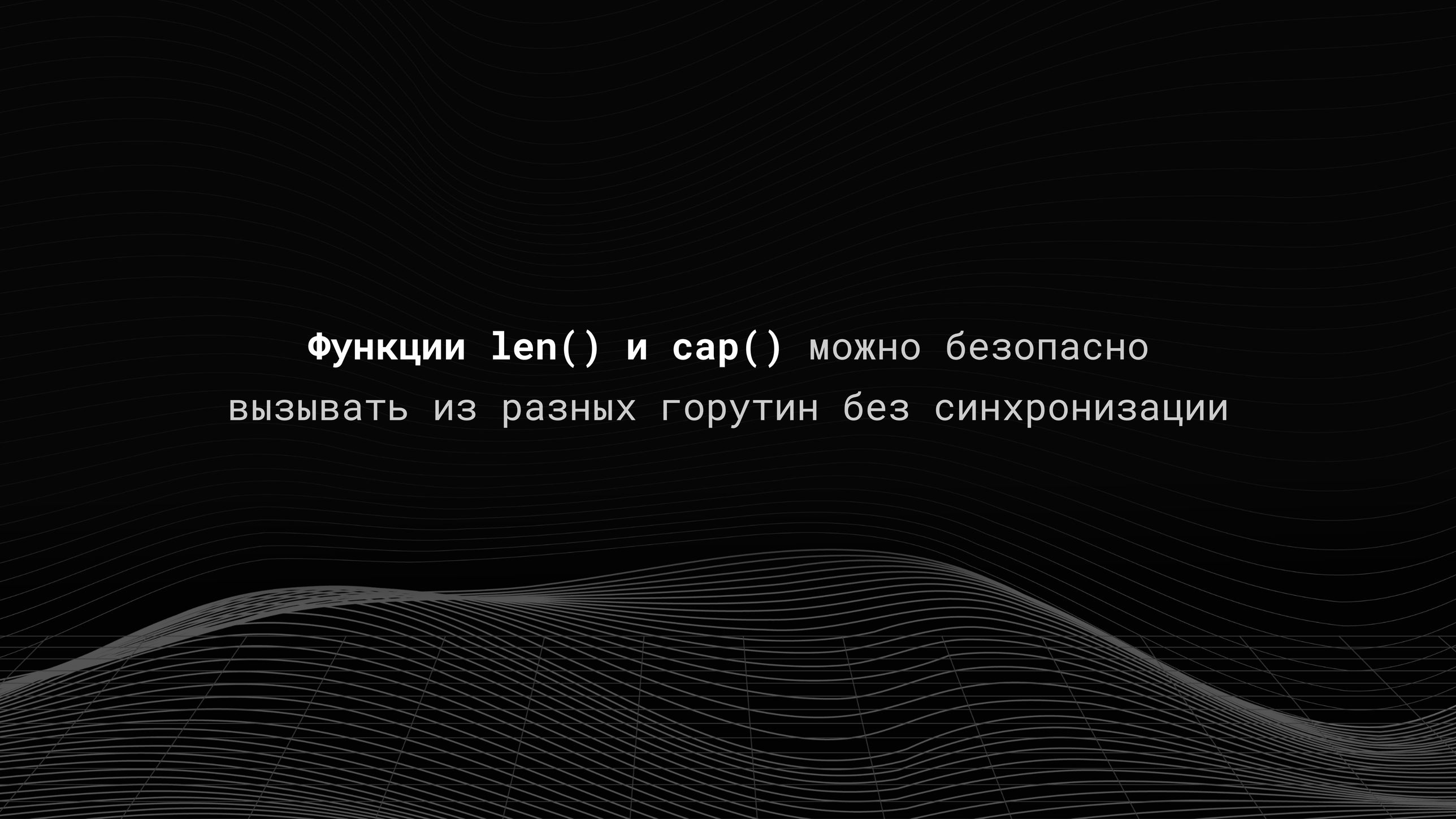
EXAMPLE

Buffered channel

МОЖНО ПРОВЕРИТЬ РАЗМЕР И ЕМКОСТЬ



Функции `len()` и `cap()` можно безопасно вызывать из разных горутин без синхронизации



ЧЕМУ БУДЕТ РАВЕН БУФЕР НЕБУФЕРИЗОВАННОГО КАНАЛА?

EXAMPLE

Operations with channel

	состояние	
операции	nil channel	closed channel
close(c)	panic	panic
read val: <-c	block	default value
write c <- val	block	panic

EXAMPLE

Copy channel

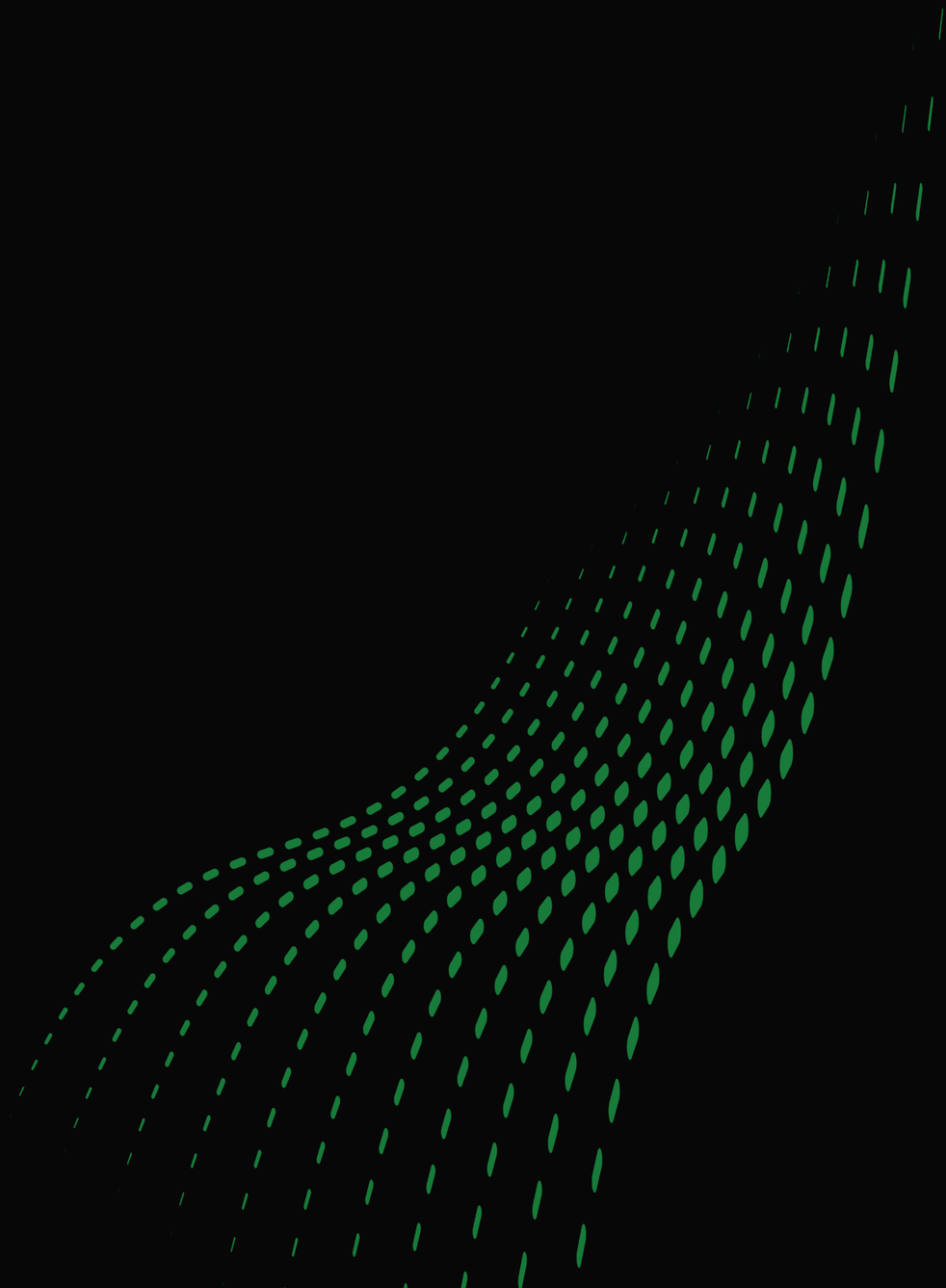
Terminal: question + ⌄ ⚙ –

КТО ДОЛЖЕН ЗАКРЫВАТЬ КАНАЛ?

ПРАВИЛО

1. Канал закрывает тот,
   кто пишет в него

2. Если несколько писателей, то тот,
   кто создал писателей и канал



Terminal: question +  

ЧТО БУДЕТ, ЕСЛИ НЕ ЗАКРЫТЬ КАНАЛ?

EXAMPLE

Goroutine leak 1

EXAMPLE

Goroutine leak 2

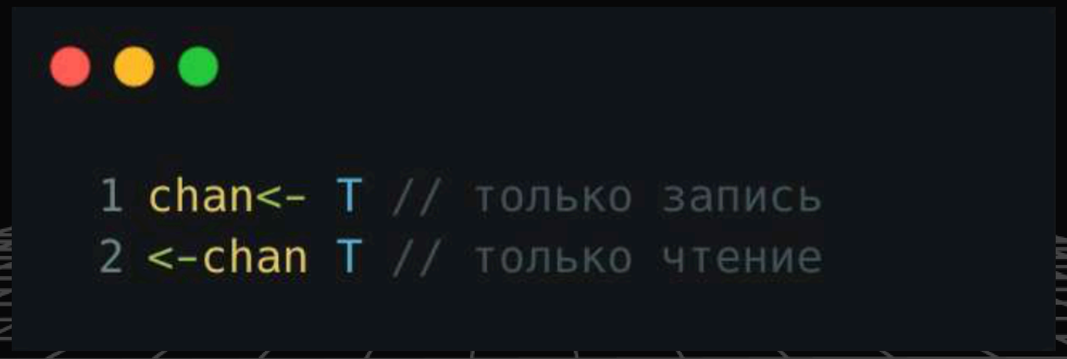
Terminal: deep_golang

HANGING GOROUTINES (ПОДВИСШИЕ ГОРУТИНЫ)

Горутины, которые навсегда остаются в заблокированном состоянии. Например, когда:

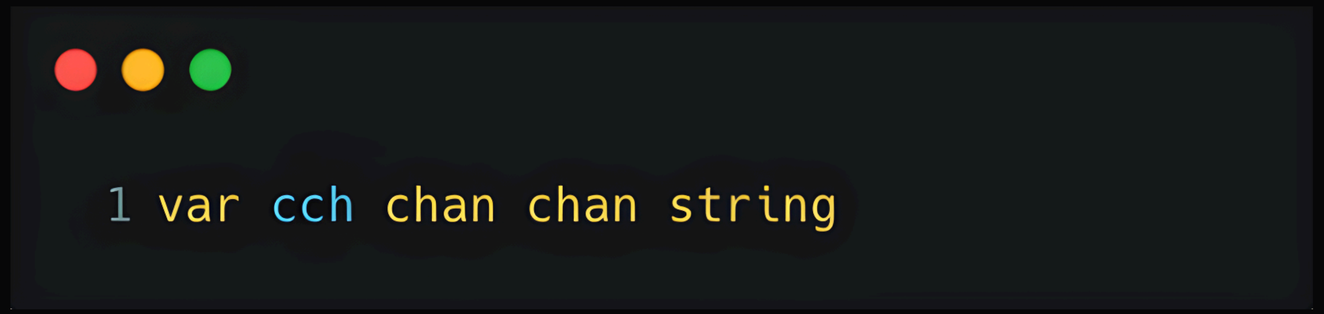
- горутина пытается получить значение из канала, которому больше никто не отправит значение или не закроет
- горутина пытается отправить значение в нулевой канал или в канал, из которого никто больше не будет читать
- горутина сама себя заблокирована при помощи мьютекса
- группа горутин заблокирована друг с другом при помощи нескольких мьютексов

ОДНОНАПРАВЛЕННЫЕ КАНАЛЫ



EXAMPLE

Unidirectional channels



Каналы являются объектами первого класса - то есть они могут быть использованы как значение элемента структуры, или аргументы функции, как возврат значения из функции/метода и даже как тип для другого канала

EXAMPLE

Prioritization

EXAMPLE

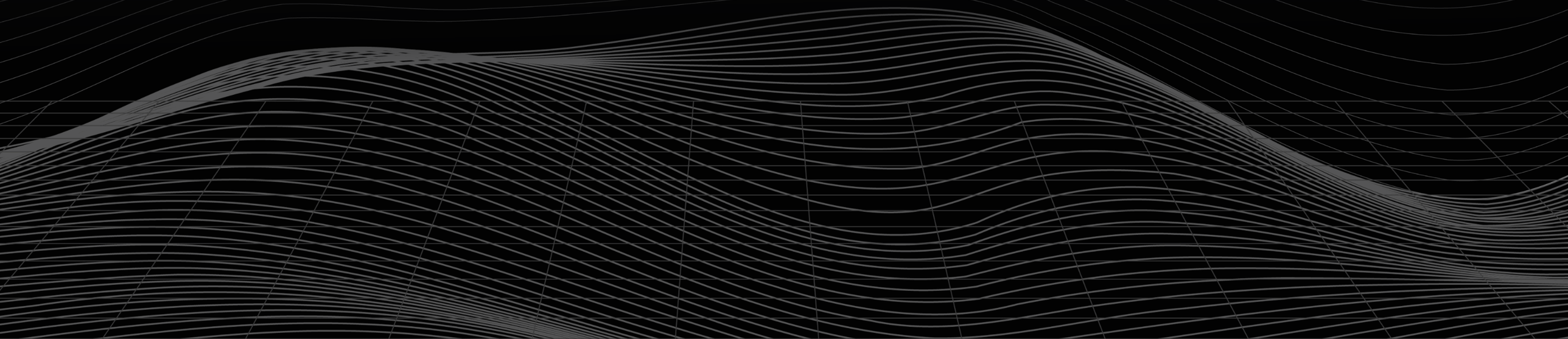
Prioritization weight

Terminal: question + ⚙ –

КАК ПОПРОБОВАТЬ ЗАПИСАТЬ ИЛИ ПРОЧИТАТЬ ИЗ КАНАЛА?

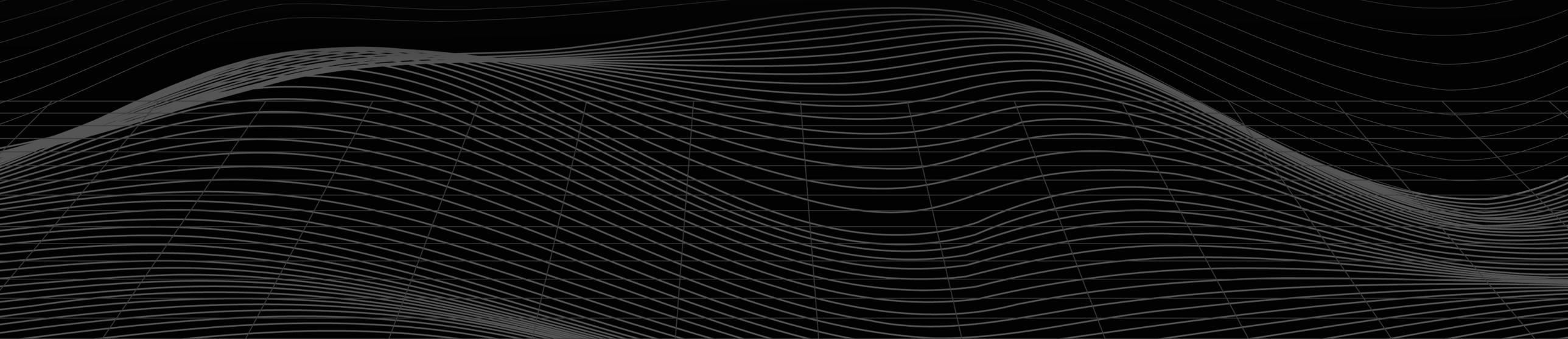
EXAMPLE

Non-blocking channels incorrect



EXAMPLE

Non-blocking channels correct



EXAMPLE

Is closed 1

EXAMPLE

Is closed 2

EXAMPLE

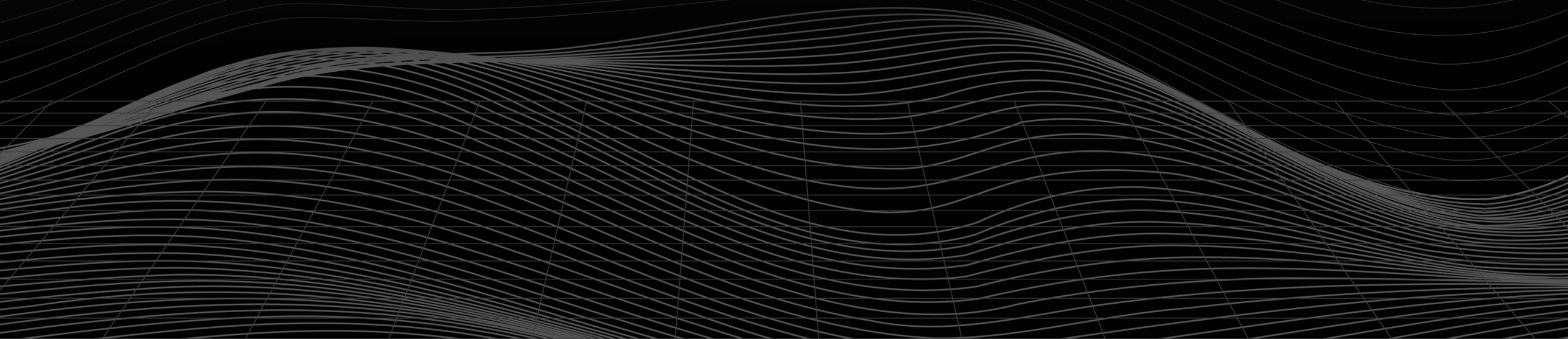
Deadlock

EXAMPLE

Channel data race

EXAMPLE

Sequential execution

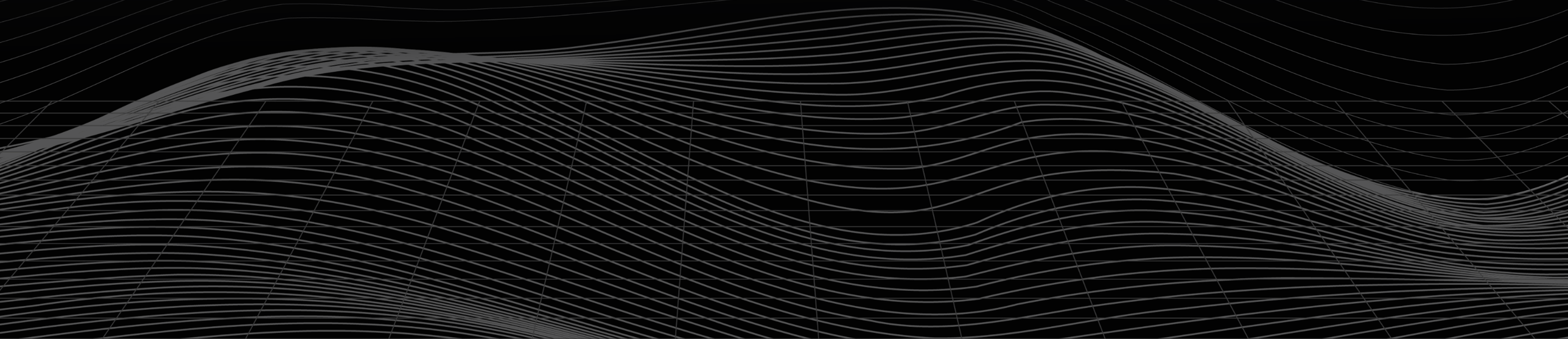


Terminal: question + ⌄ ⚙ –

ЗАЧЕМ НУЖНЫ NIL КАНАЛЫ?

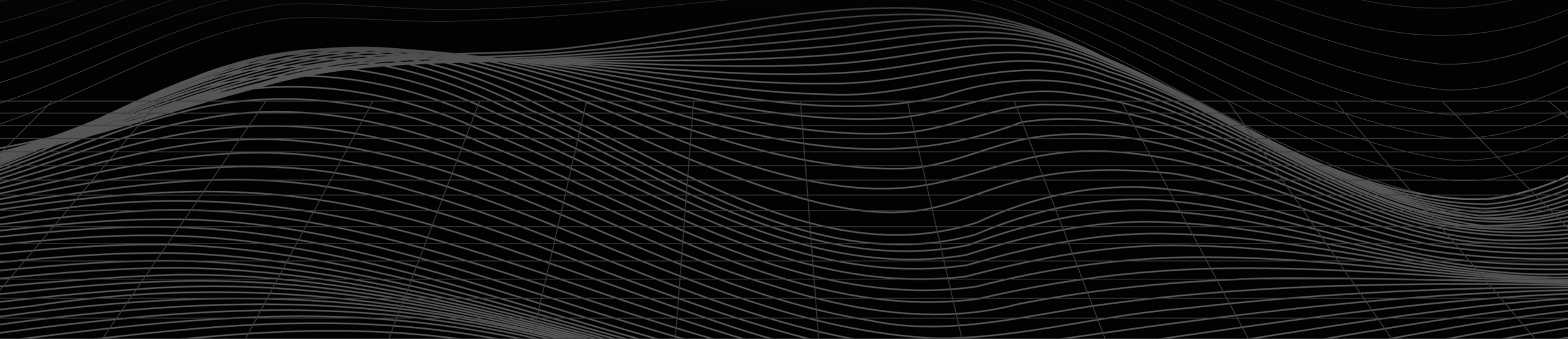
EXAMPLE

Nil channel



EXAMPLE

Nil channel task

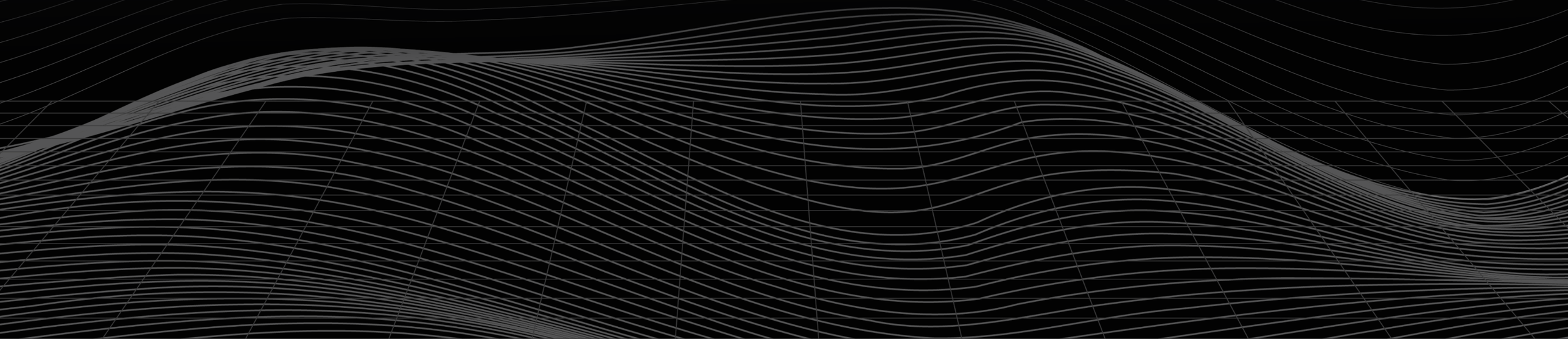


Terminal: question + 

КАК ЗАБЛОКИРОВАТЬ ГОРУТИНУ НАВСЕГДА?

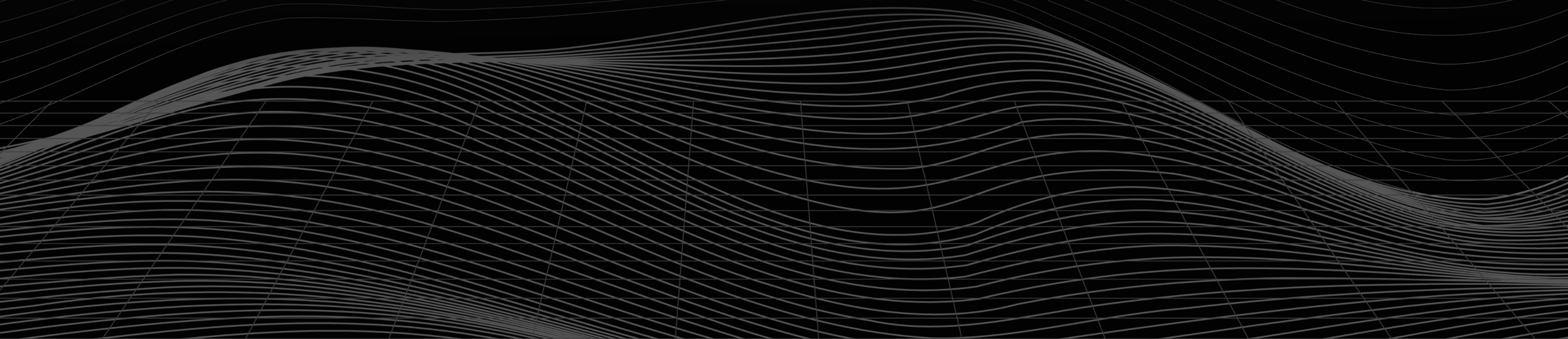
EXAMPLE

Select forever



EXAMPLE

Select with main goroutine

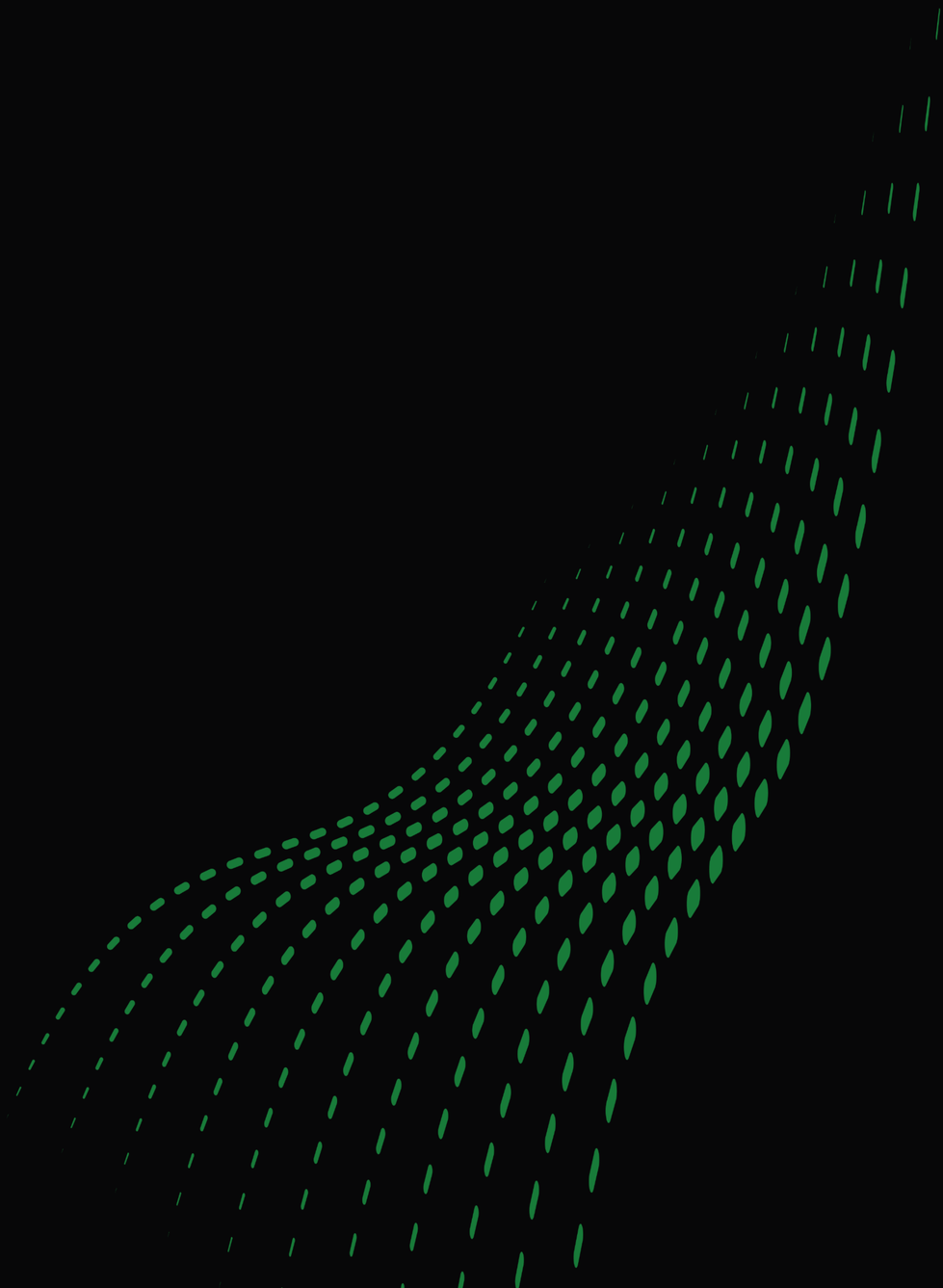


Когда использовать каналы:

1. передача владения данными
2. распределение вычислений
3. передача асинхронных результатов

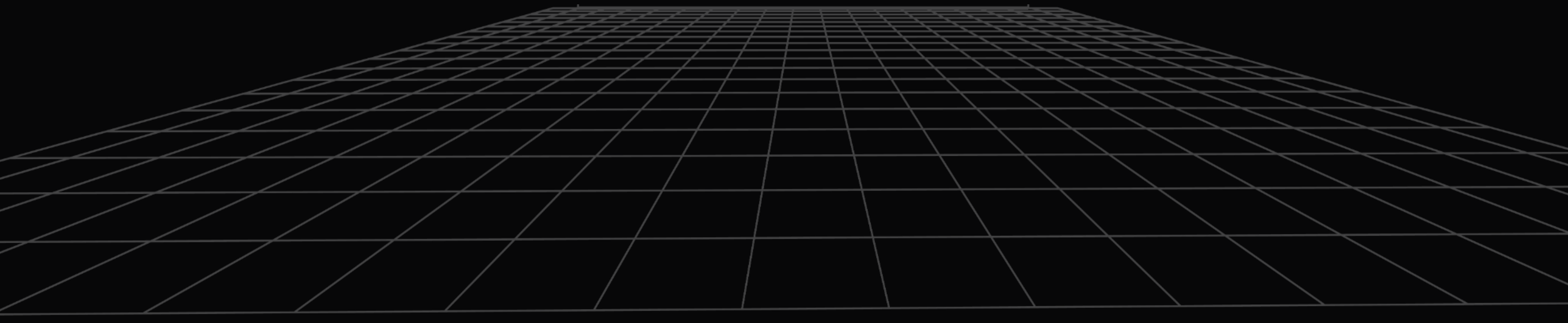
Когда использовать мьютексы:

1. кэши
2. состояния

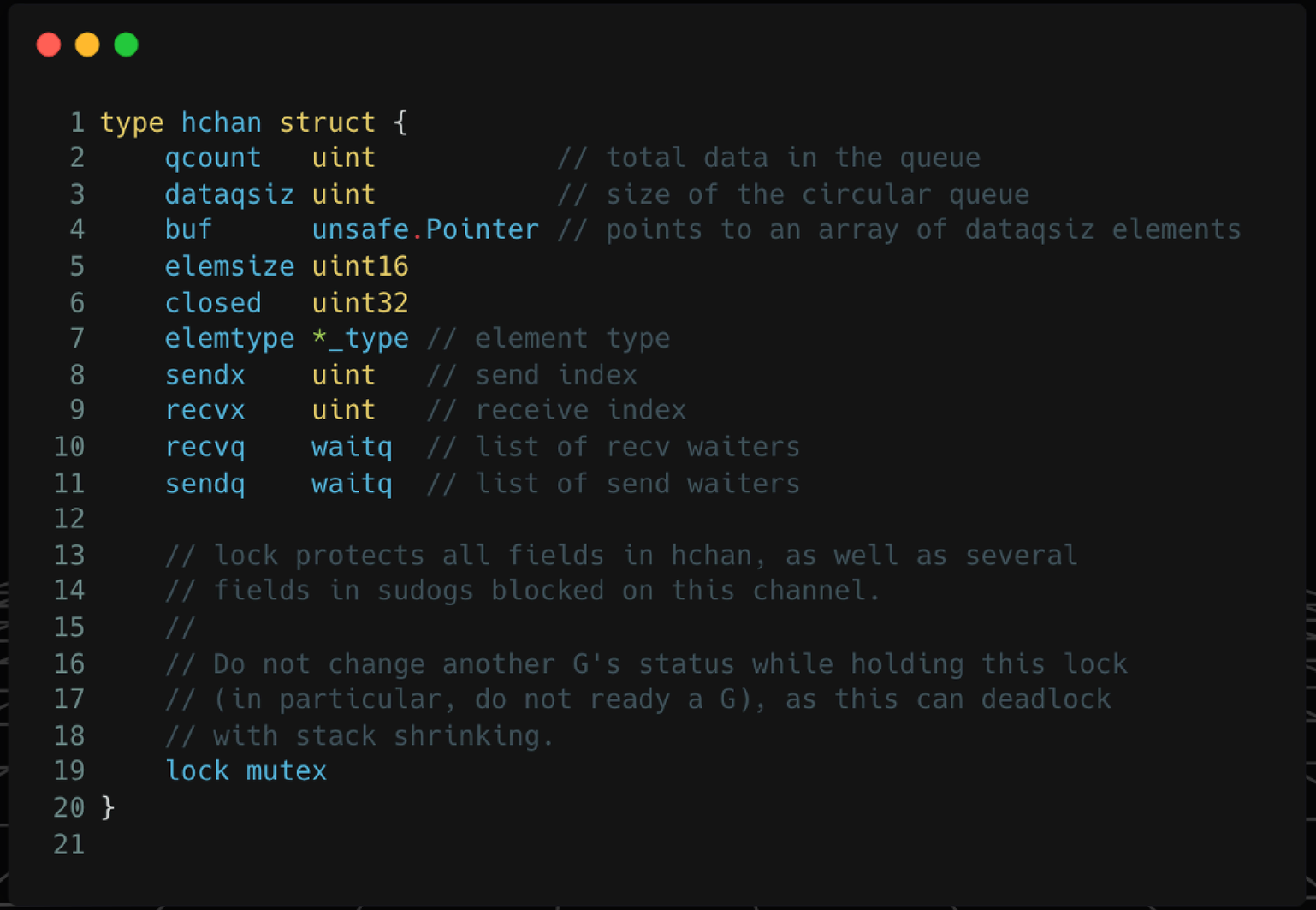


FAQ

Каналы

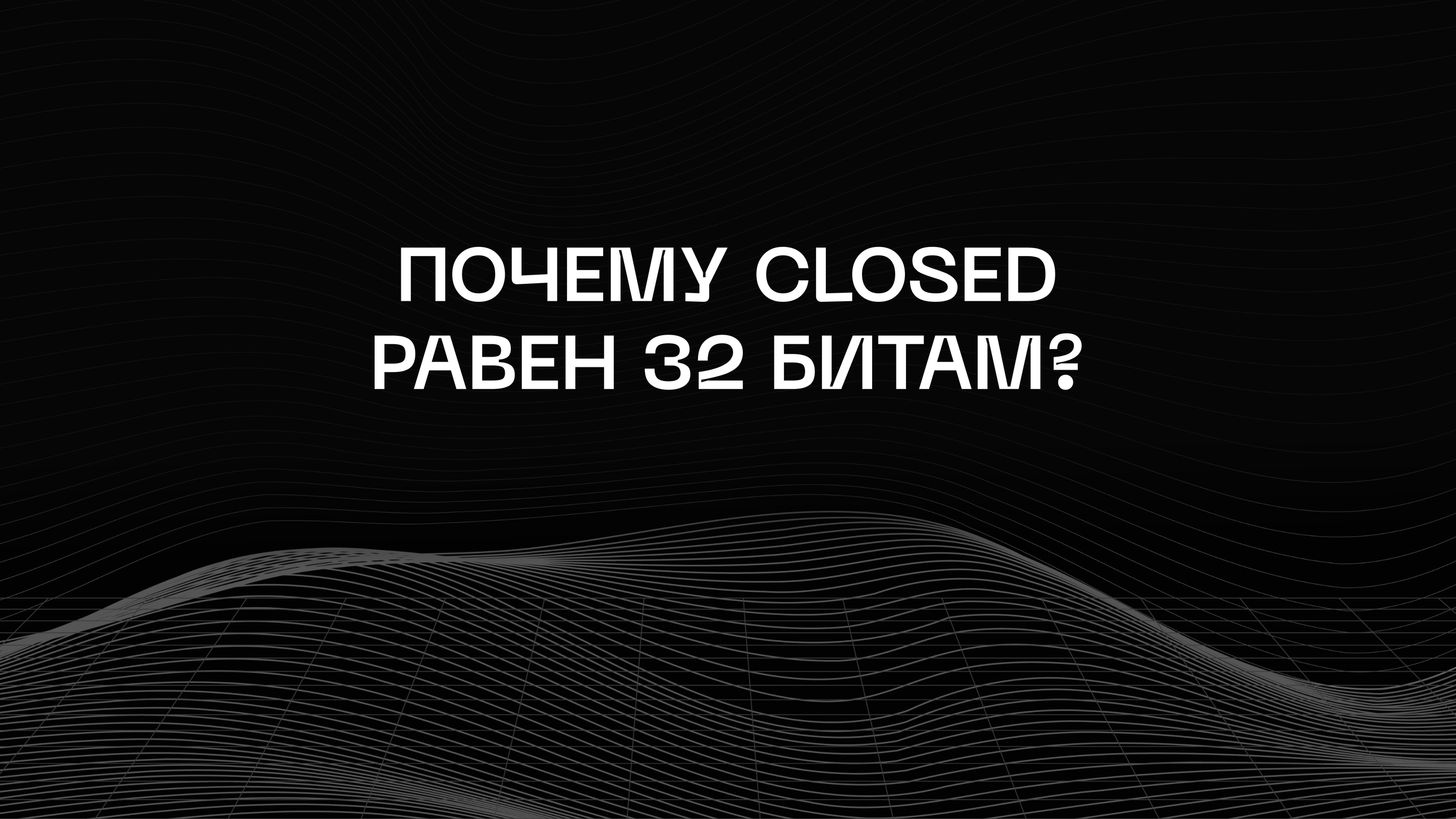


ВНУТРЕННЕЕ УСТРОЙСТВО



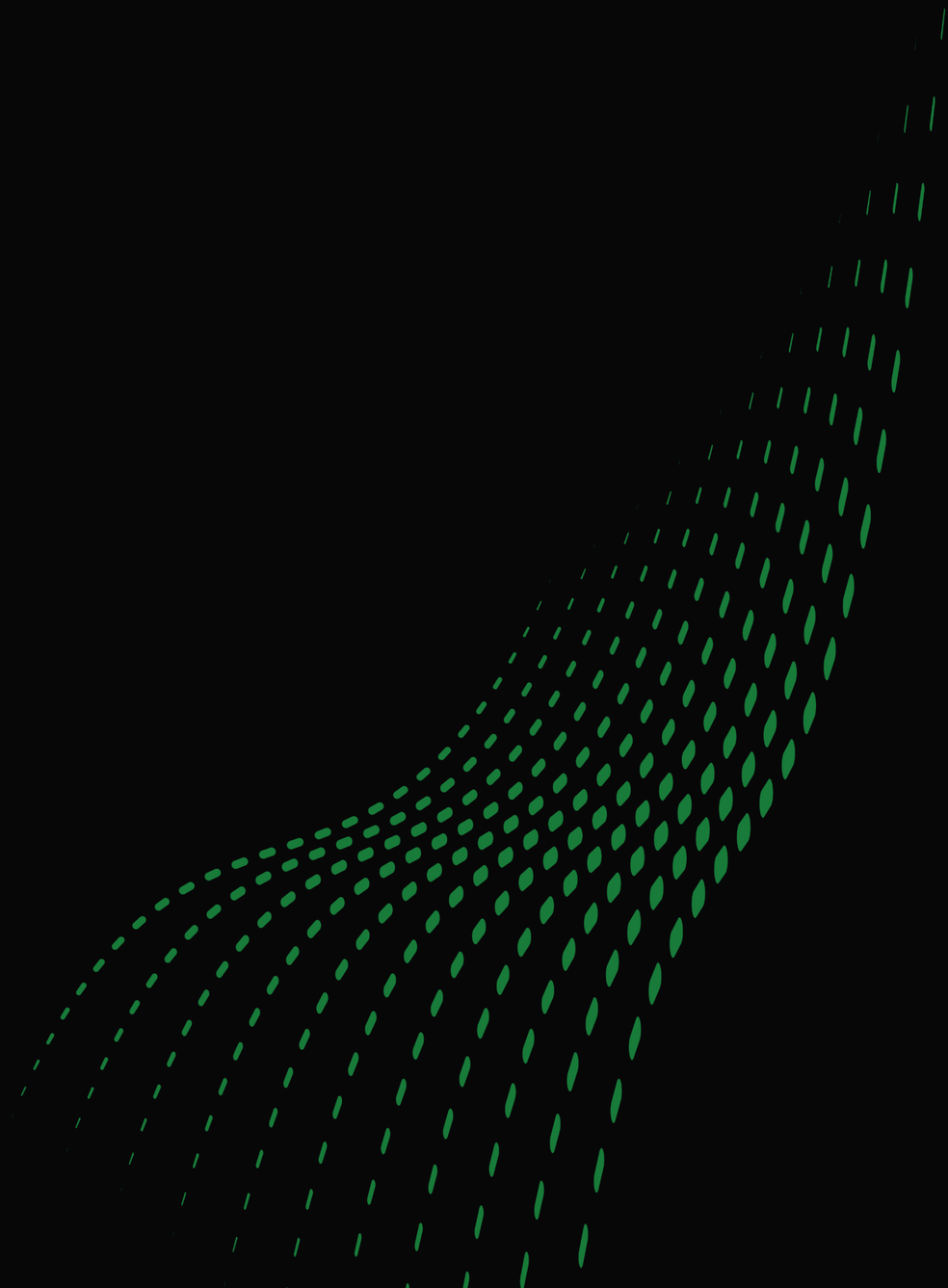
```
type hchan struct {
    qcount   uint           // total data in the queue
    dataqsiz uint           // size of the circular queue
    buf      unsafe.Pointer // points to an array of dataqsiz elements
    elemsize uint16
    closed   uint32
    elemtype *_type // element type
    sendx    uint   // 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
}

```

ПОЧЕМУ CLOSED РАВЕН 32 БИТАМ?



ОСОБЕННОСТИ

1. Канал — это указатель на объект структуры
2. Функция make возвращает указатель на него
3. Мы работаем с указателем на объект структуры hchan

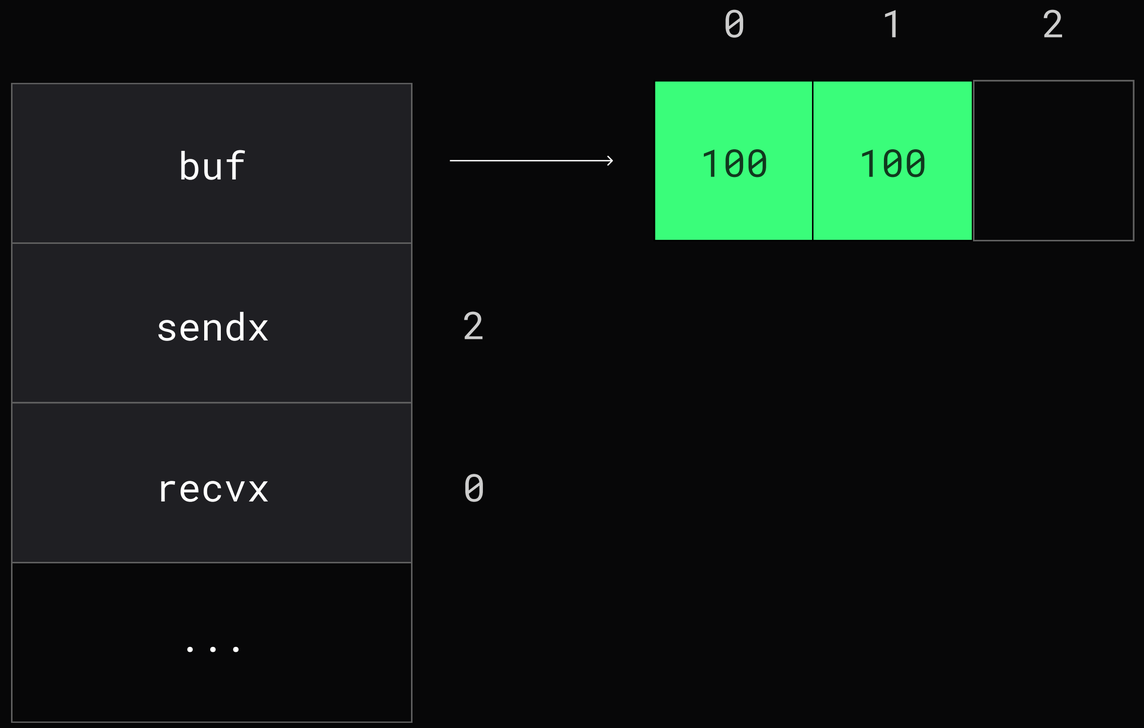


КАК УСТРОЕН БУФЕР ВНУТРИ КАНАЛА?

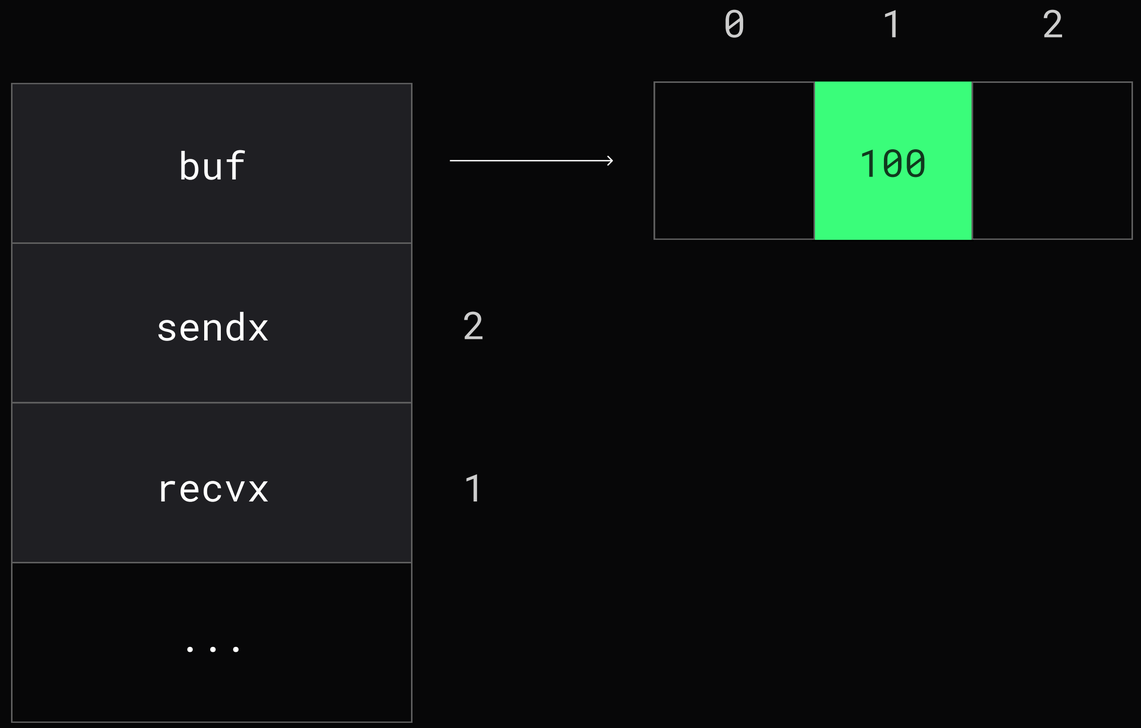
```go
1 ch := make(chan int, 3)
```

```
1 ch <- 100
```

```
1 ch <- 100
```

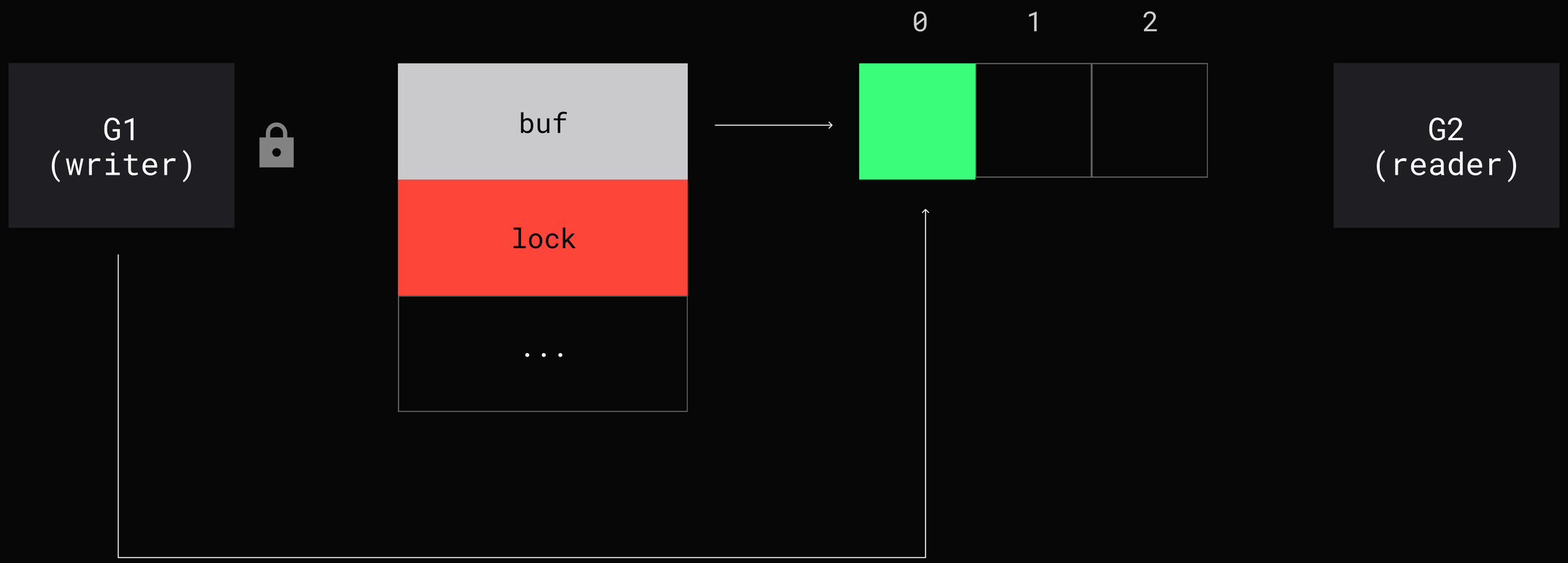


```
1 value := <-ch
```

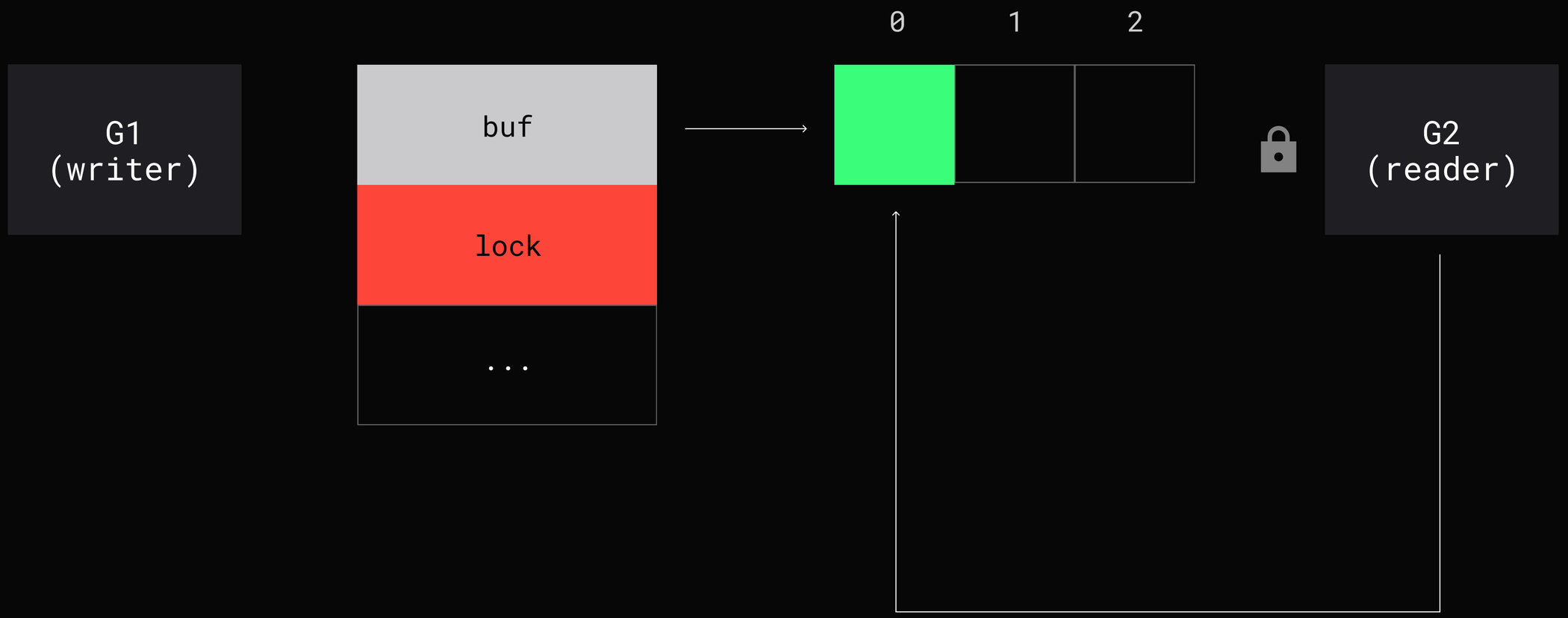


КАК ПРОИСХОДИТ БЕЗОПАСНЫЕ ЗАПИСЬ И ЧТЕНИЕ?

Писатель блокирует мьютекс во время записи

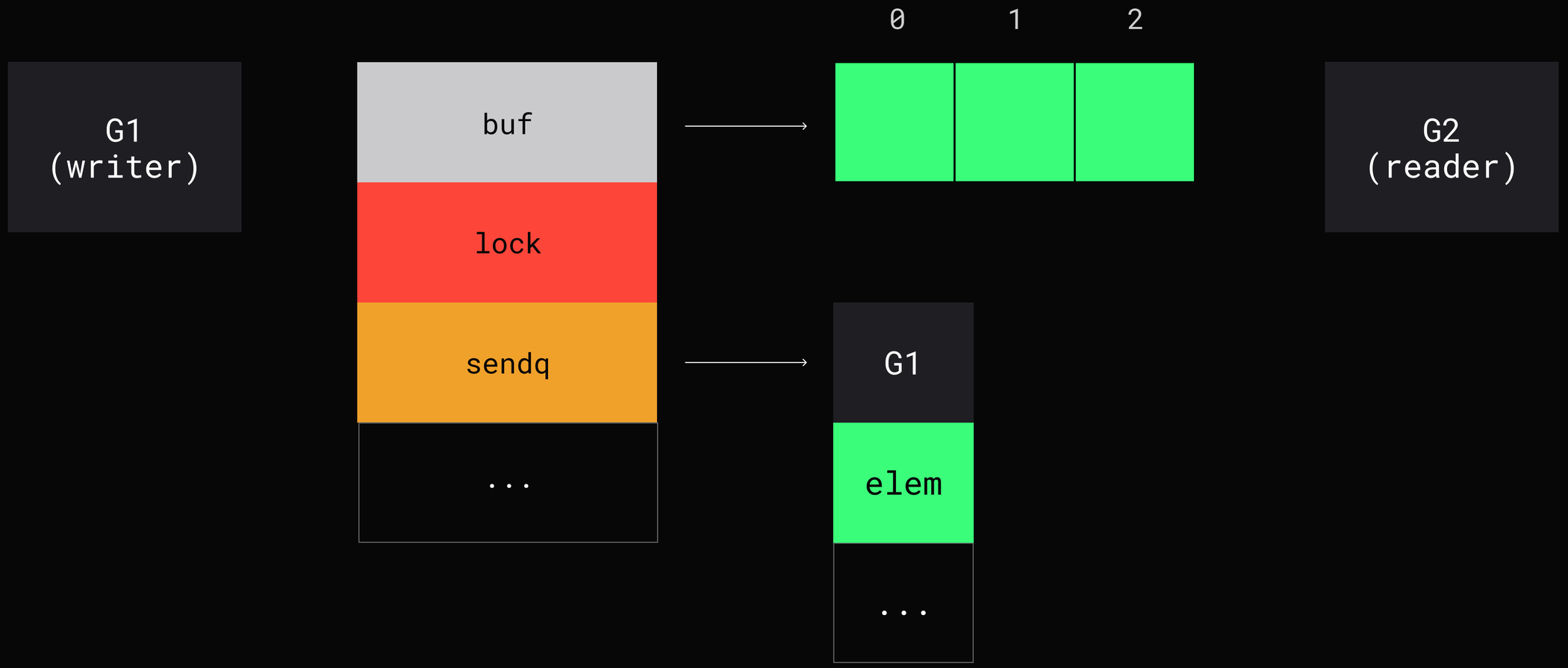


Читатель блокирует мьютекс во время чтения



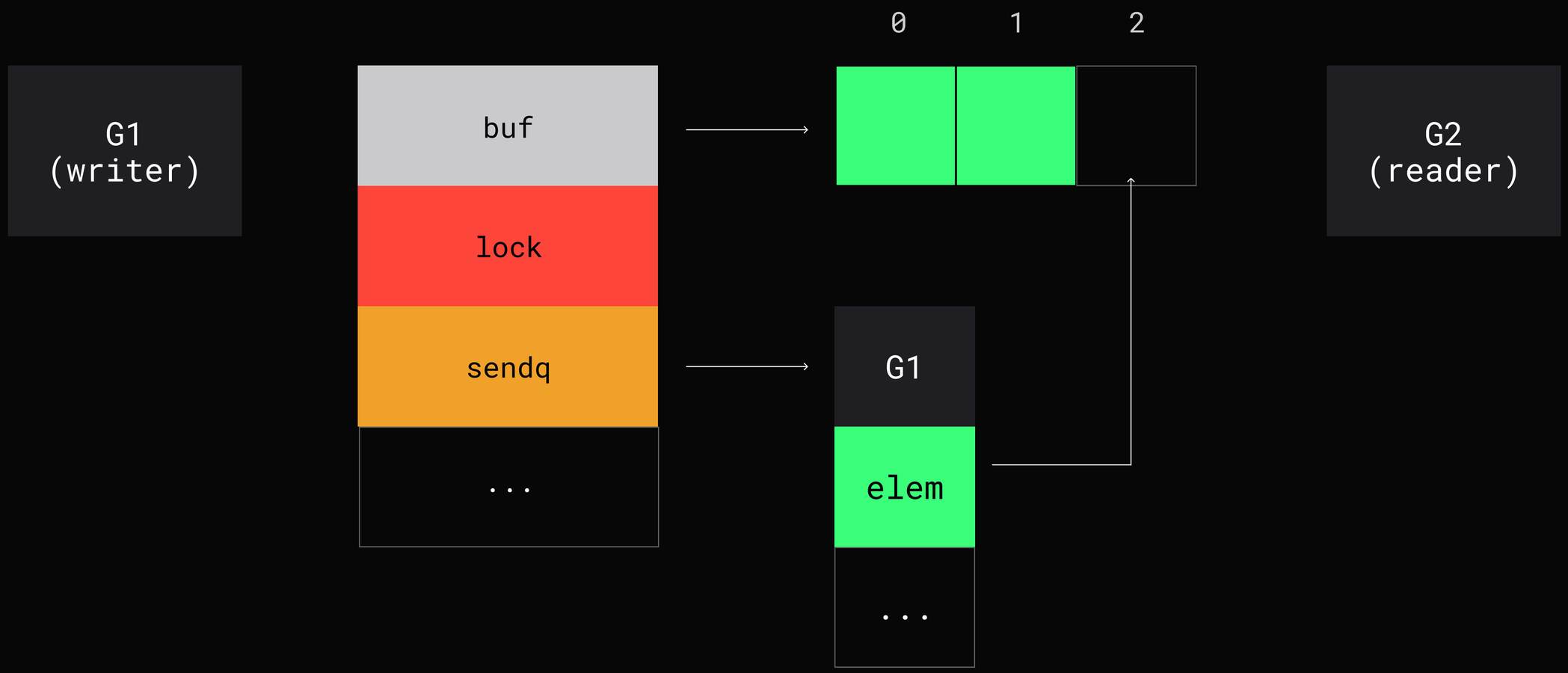
КУДА ПИСАТЬ, ЕСЛИ БУФЕР ЗАПОЛНЕН?

elem – указатель на данные

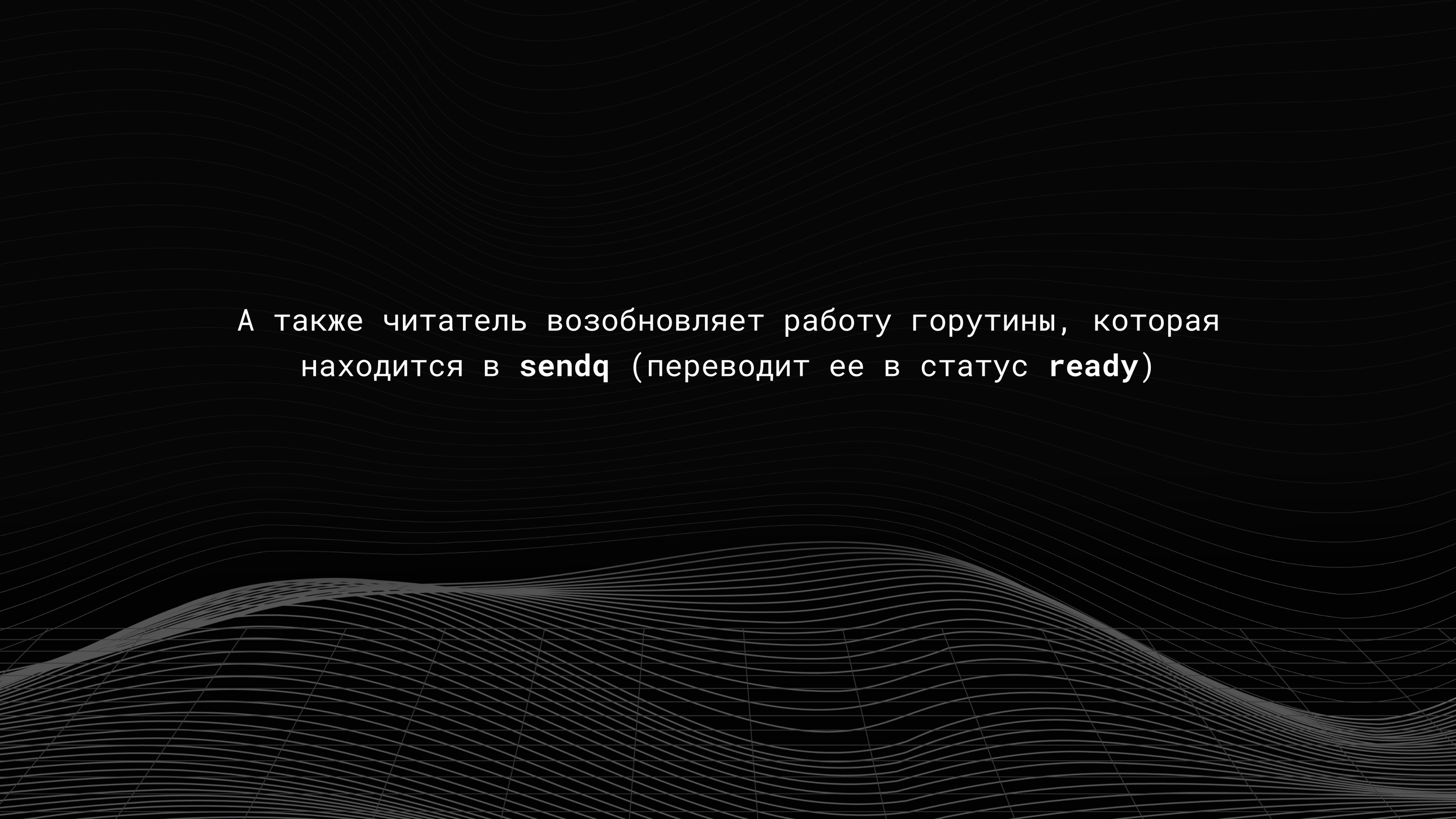


ПОЧЕМУ В SENDQ ХРАНИТСЯ УКАЗАТЕЛЬ НА ЭЛЕМЕНТ, А НЕ САМ ЭЛЕМЕНТ?

Читатель после чтения забирает данные из другой горутины по указателю и кладет в буффер (чтобы не копировать лишний раз)

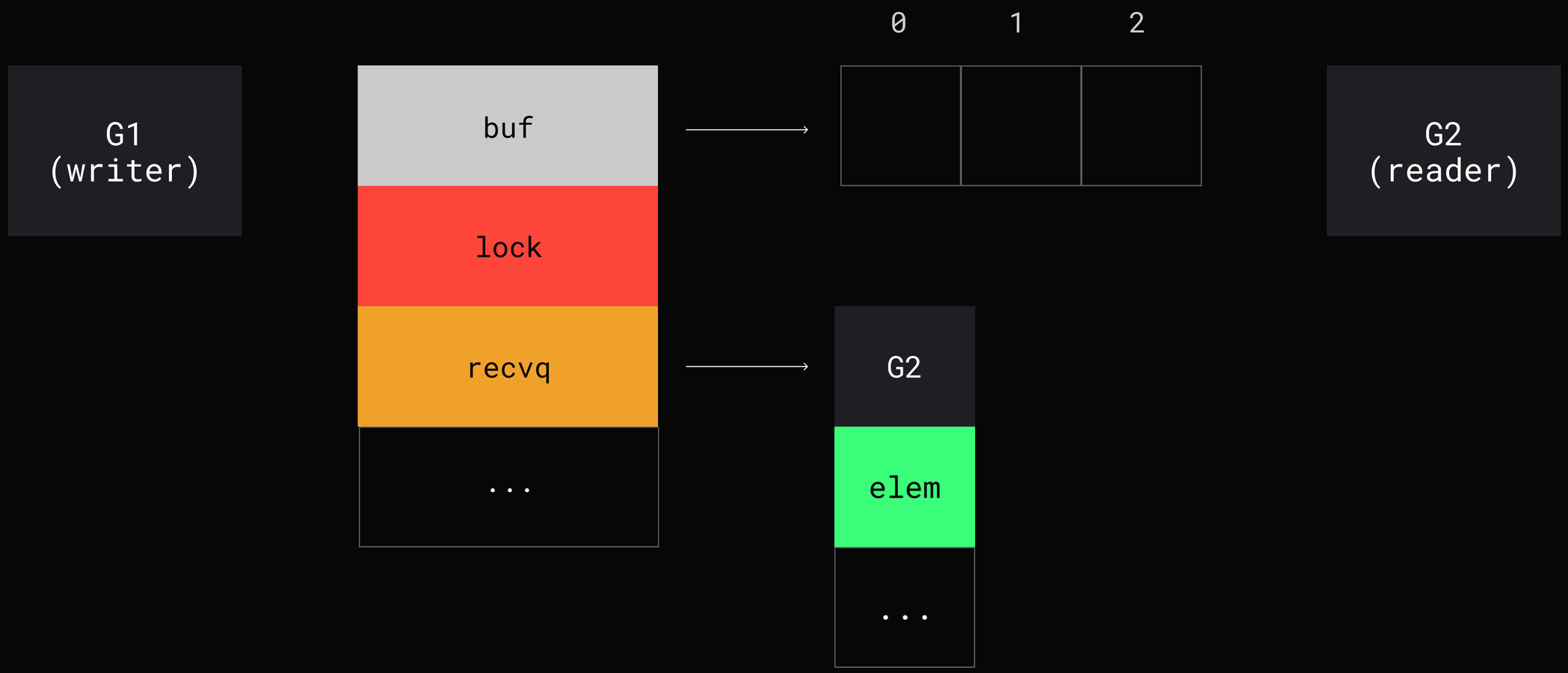


А также читатель возобновляет работу горутины, которая находится в **sendq** (переводит ее в статус **ready**)

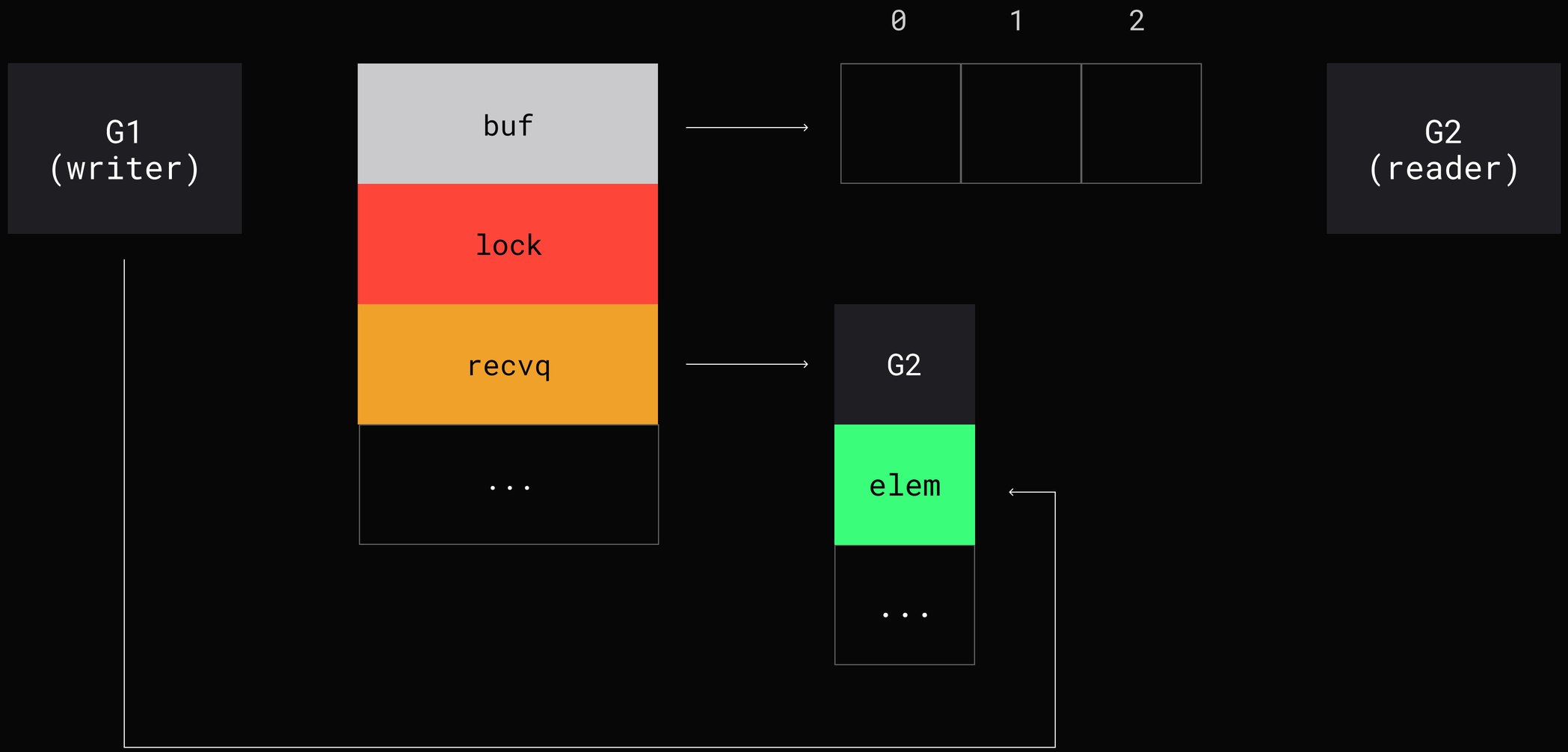


ЧТО ДЕЛАТЬ ЧИТАТЕЛЮ, ЕСЛИ В БУФЕРЕ НЕТ НИЧЕГО?

elem – указатель на то место, куда нужно читать

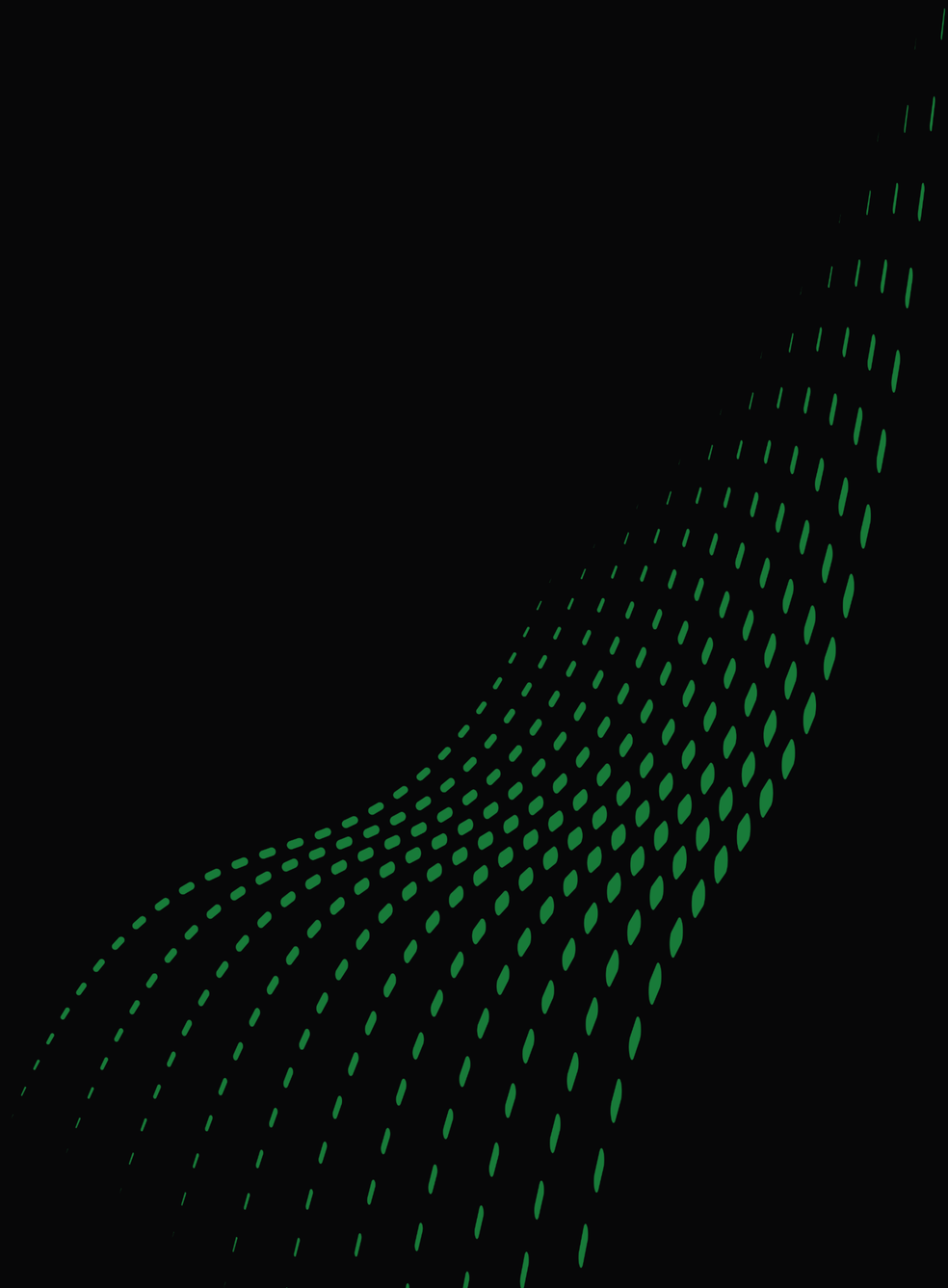


Писатель **пишет данные в стек** другой горутины по указателю



G1 пишет в стек G2

1. Не надо делать операции с буфером
2. Отсутствует дополнительное копирование



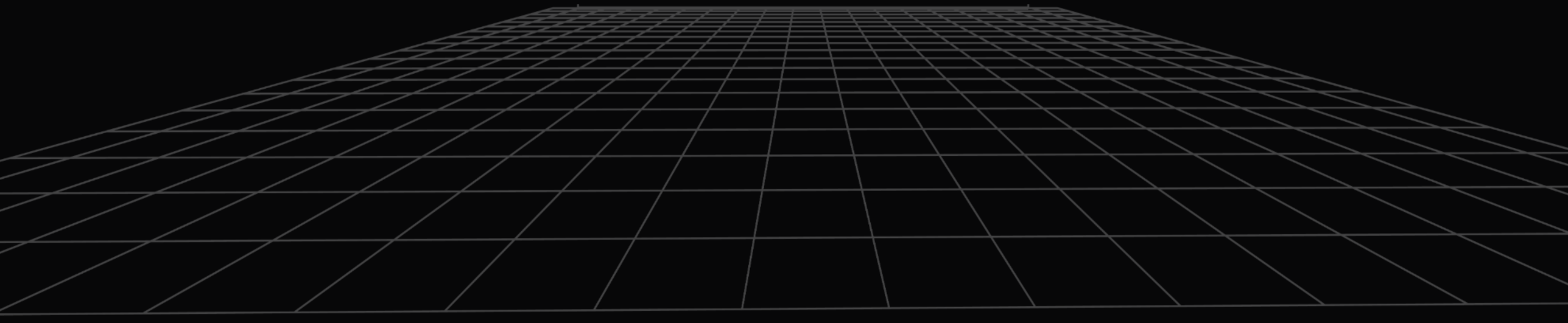
EXAMPLE

Write after close

НЕБУФЕРИЗОВАННЫЕ КАНАЛЫ РАБОТАЮТ БЫСТРЕЕ!

FAQ

Внутреннее устройство



ПОЖАЛУЙСТА, ЗАПОЛНИ ОПРОС О ЗАНЯТИИ

Ссылка в чате и в группе участников