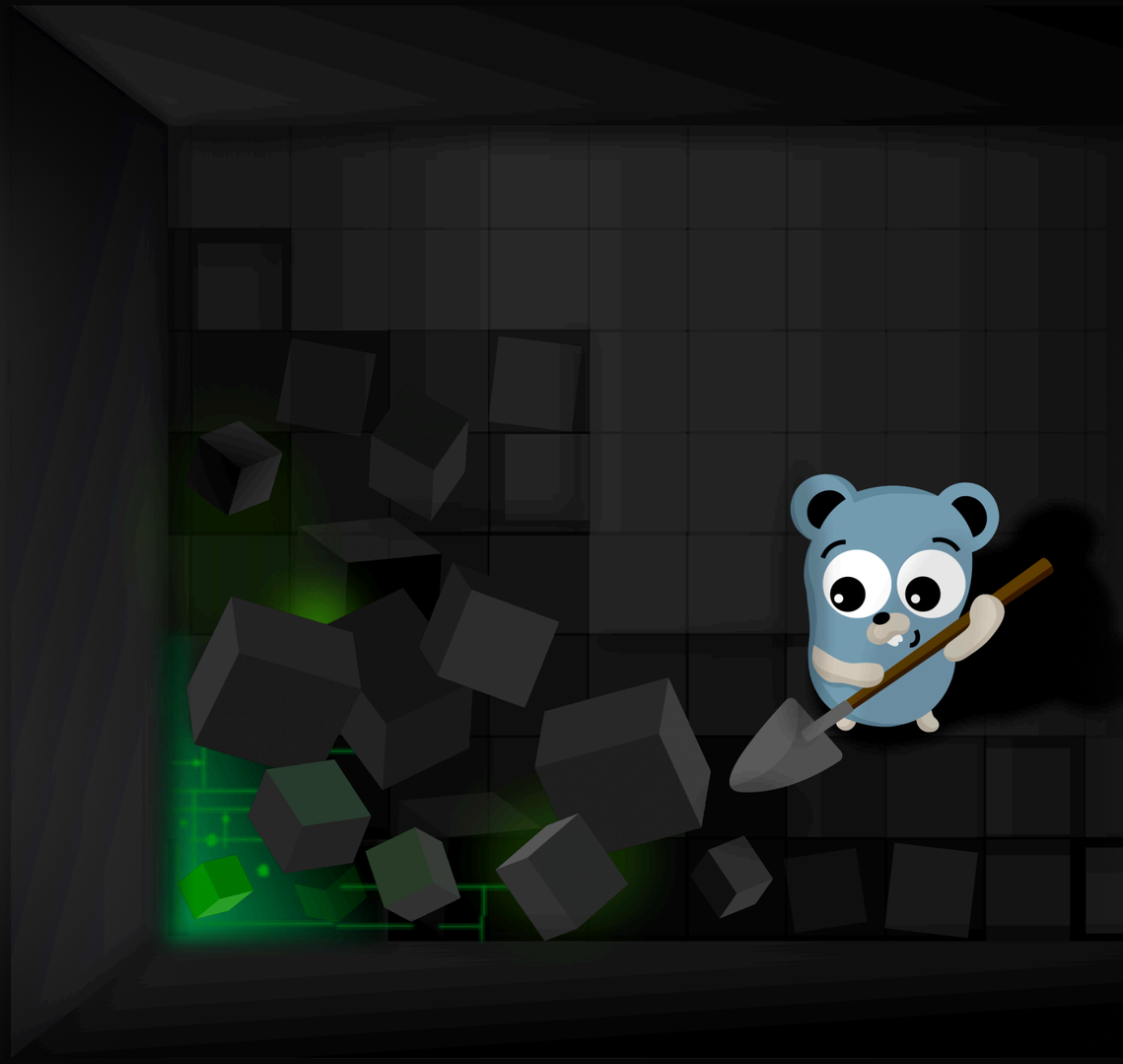
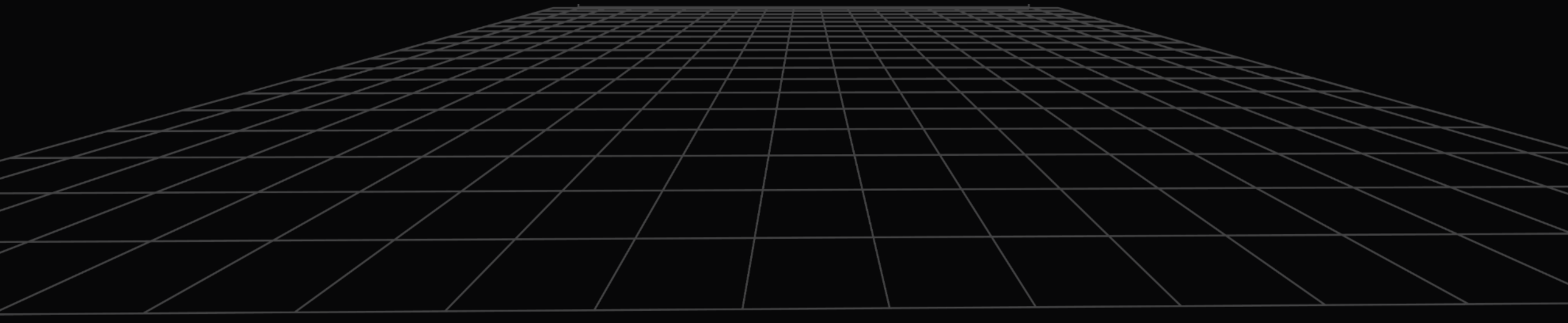
ПРИМИТИВЫ СИНХРОНИЗАЦИИ



Balun.Courses

ПРОВЕРЬ ЗАПИСЬ

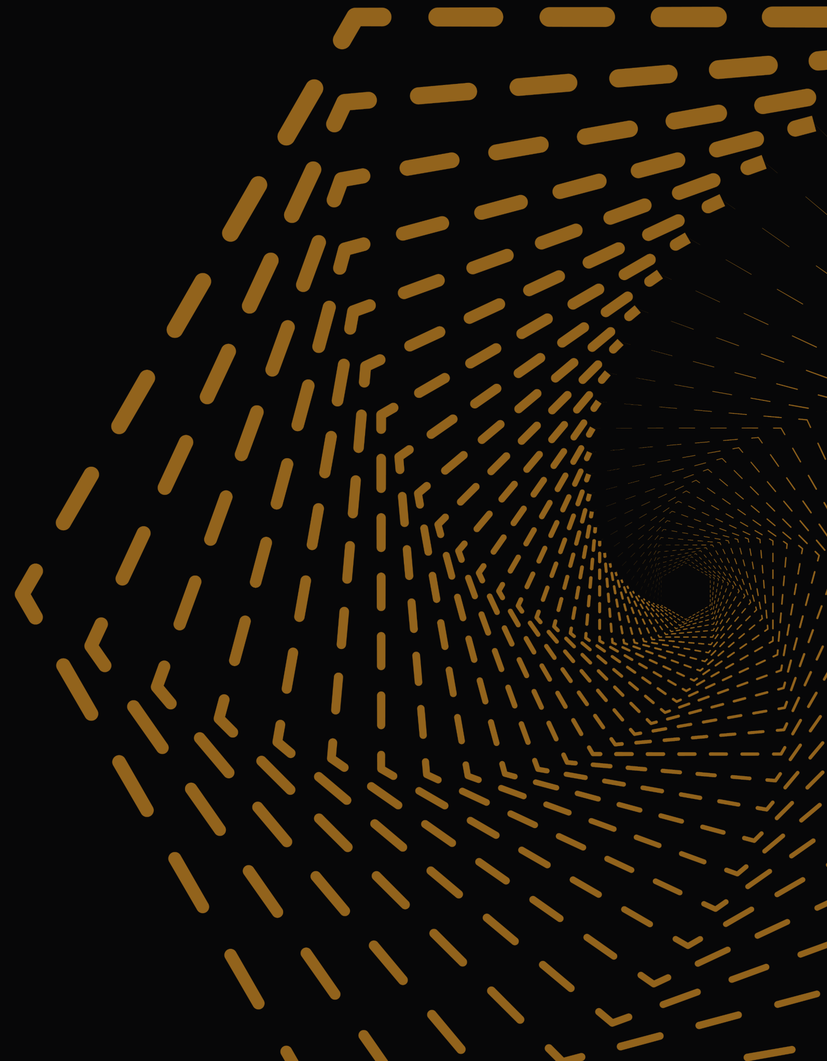


ПРАВИЛА ЗАНЯТИЯ

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**



WAITGROUP

EXAMPLE

Waiting goroutines

API

```go
type WaitGroup struct { ... }

func (wg *WaitGroup) Add(delta int) // увеличивает счетчик
func (wg *WaitGroup) Done()         // уменьшает счетчик на единицу
func (wg *WaitGroup) Wait()         // блокируется, пока счетчик не обнулится
```

EXAMPLE

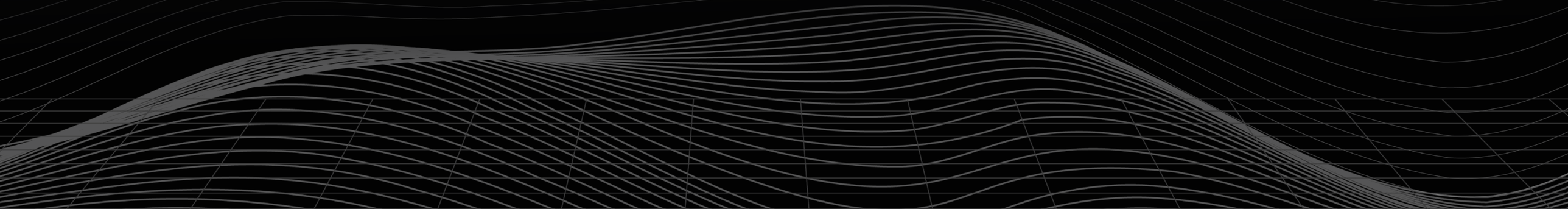
WaitGroup operations

EXAMPLE

WaitGroup copying

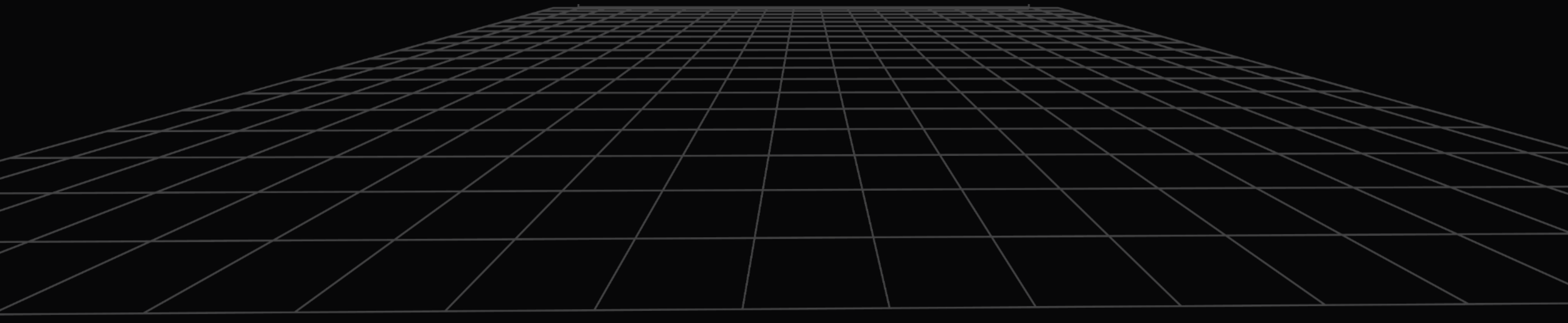
Values containing the types defined
in this package should not be copied!

https://pkg.go.dev/sync



FAQ

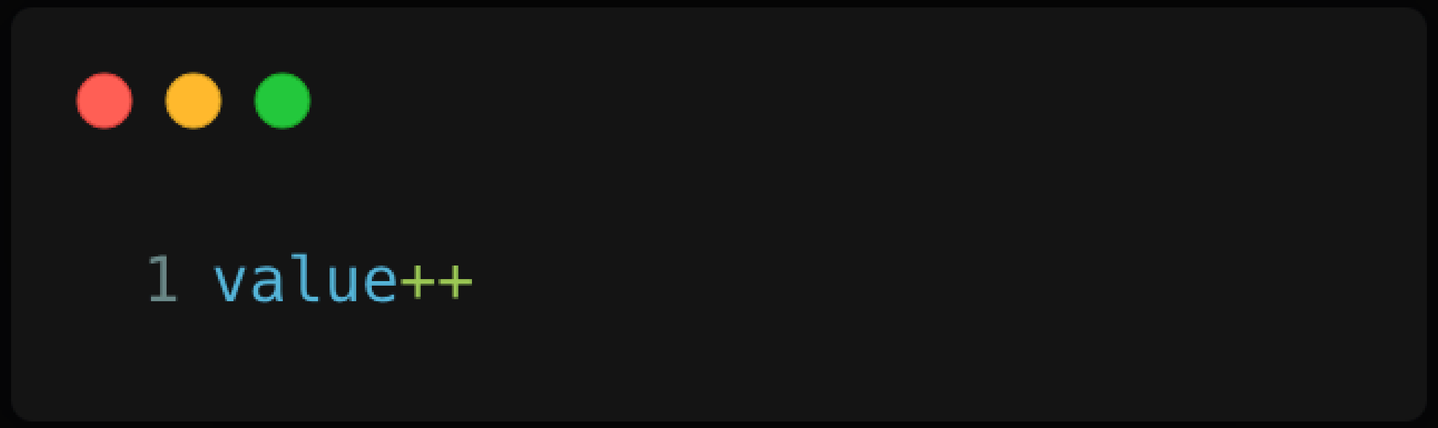
WaitGroup



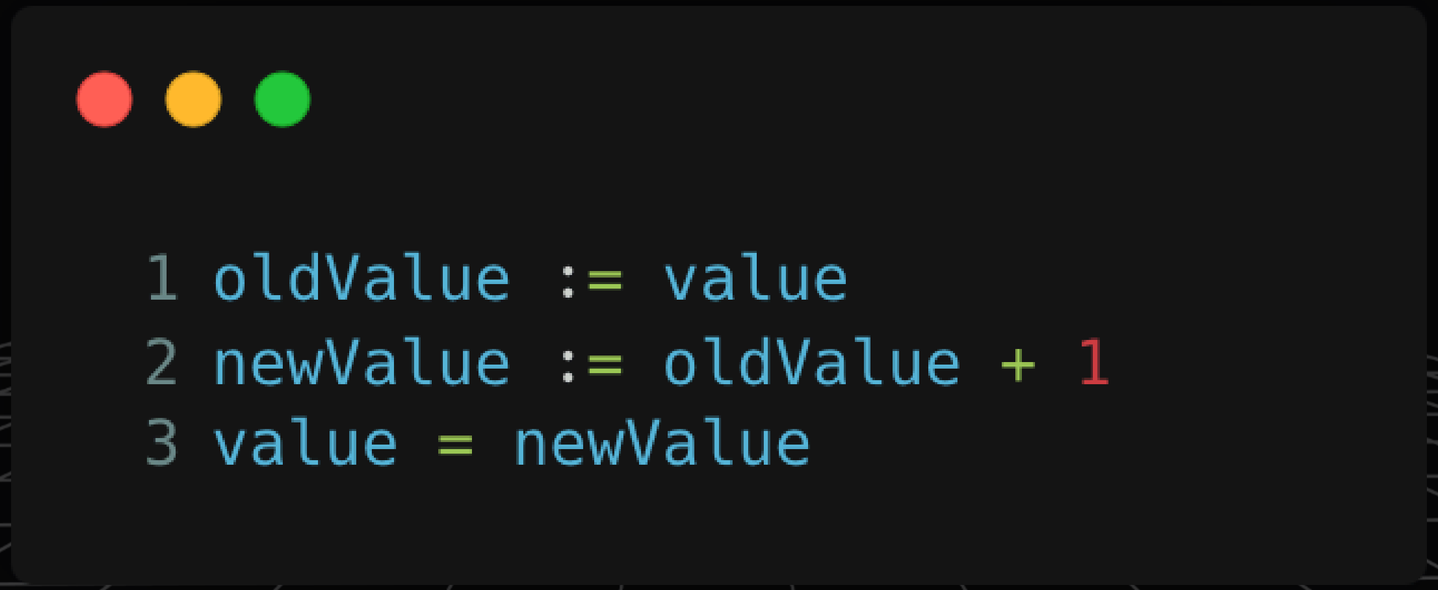
MUTEX

EXAMPLE

Incorrect increment



```
value++
```



```
oldValue := value
newValue := oldValue + 1
value = newValue
```

value = 100

G1

```
oldValue := value // oldValue = 100
```

G2

```
oldValue := value // oldValue = 100
```

G1

```
newValue := oldValue + 1 // newValue = 101
value = newValue         // value = 101
```

G2

```
newValue := oldValue + 1 // newValue = 101
value = newValue         // value = 101
```

value = 101

Terminal: Concurrency

МЬЮТЕКСЫ

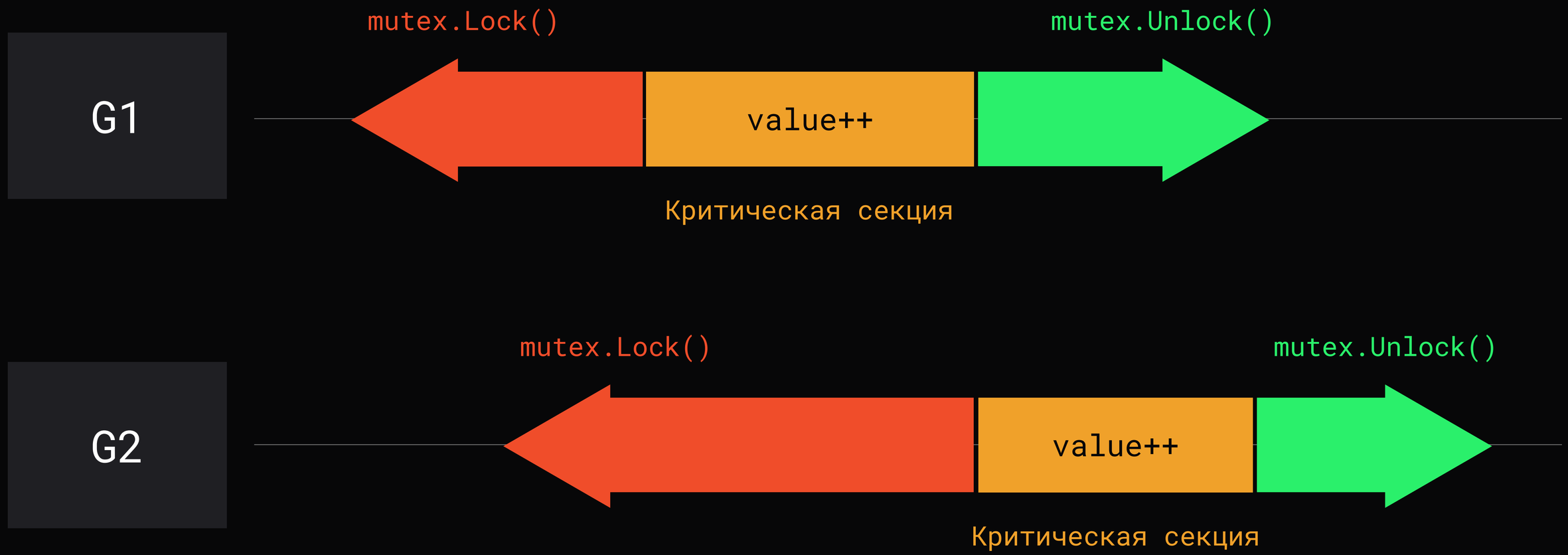
Mutural Exclusion – взаимное исключение

API

```
1 type Mutex struct { ... }
2
3 func (m *Mutex) Lock()          // захватить мьютекс
4 func (m *Mutex) Unlock()        // освободить мьютекс
5 func (m *Mutex) TryLock() bool // попробовать захватить мьютекс
```

EXAMPLE

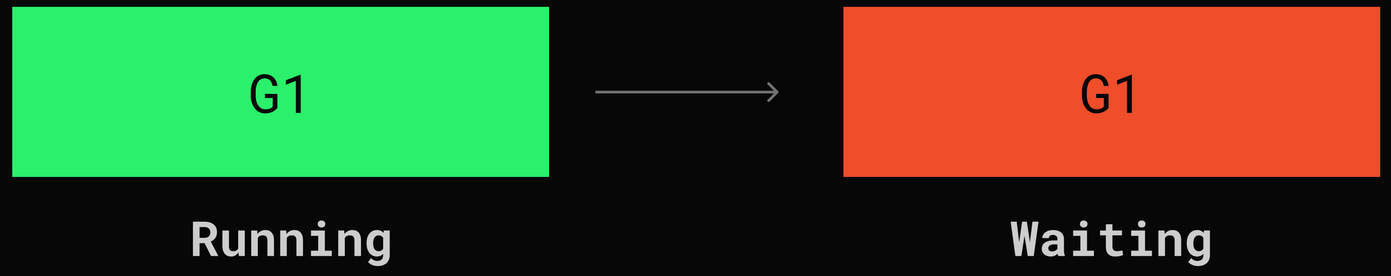
Correct increment



Горутина захватывает мьютекс, затем входит
в критическую секцию, а в конце освобождает мьютекс


mutex.Lock()
value++
mutex.Unlock()
mutex.Lock()
value++
mutex.Unlock()
G1
(running)
mutex.Lock()
value++
G2
(ready)
G1
(ready)
mutex.Lock()
G2
(running)
G1
(running)
mutex.Unlock()
G2
(waiting)
G1
(ready)
value++
mutex.Unlock()
G2
(running)

Когда горутина блокируется на мьютексе,
она переходит из состояния Running в Waiting

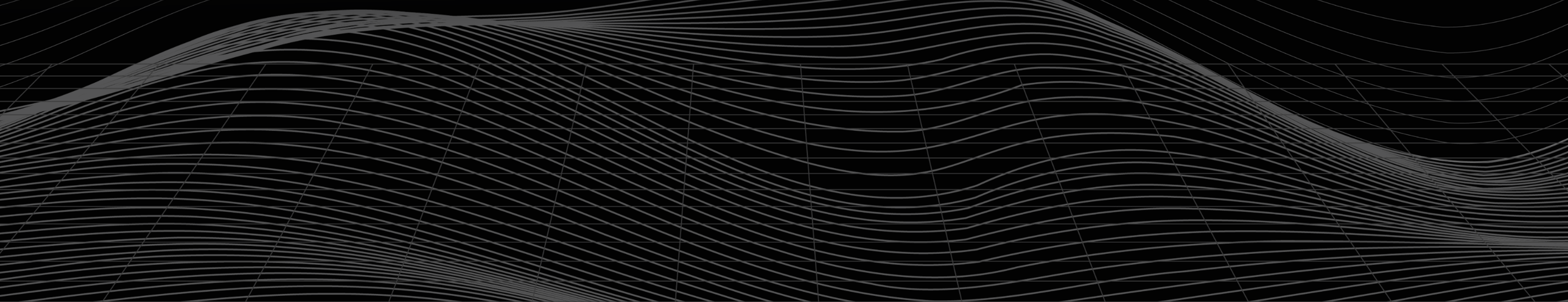


EXAMPLE

Mutex different operations

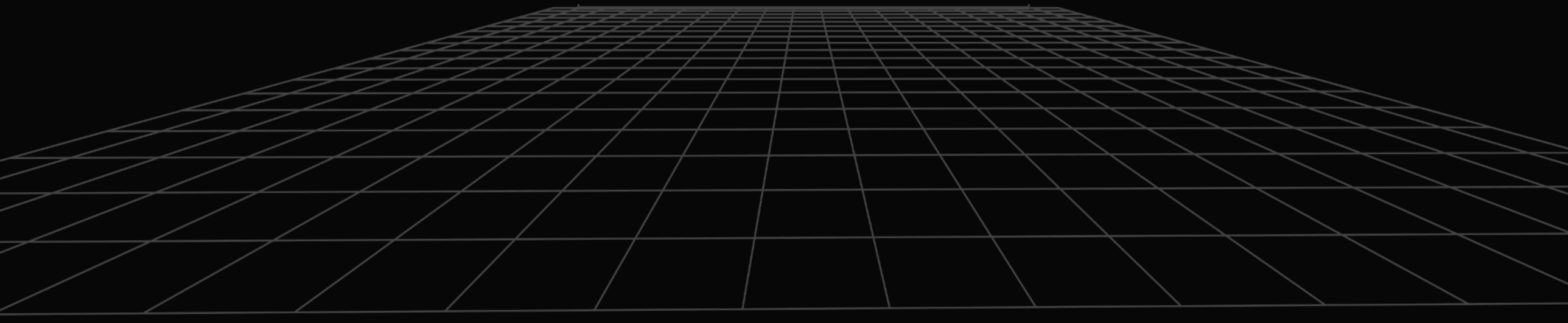
EXAMPLE

Mutex with local values



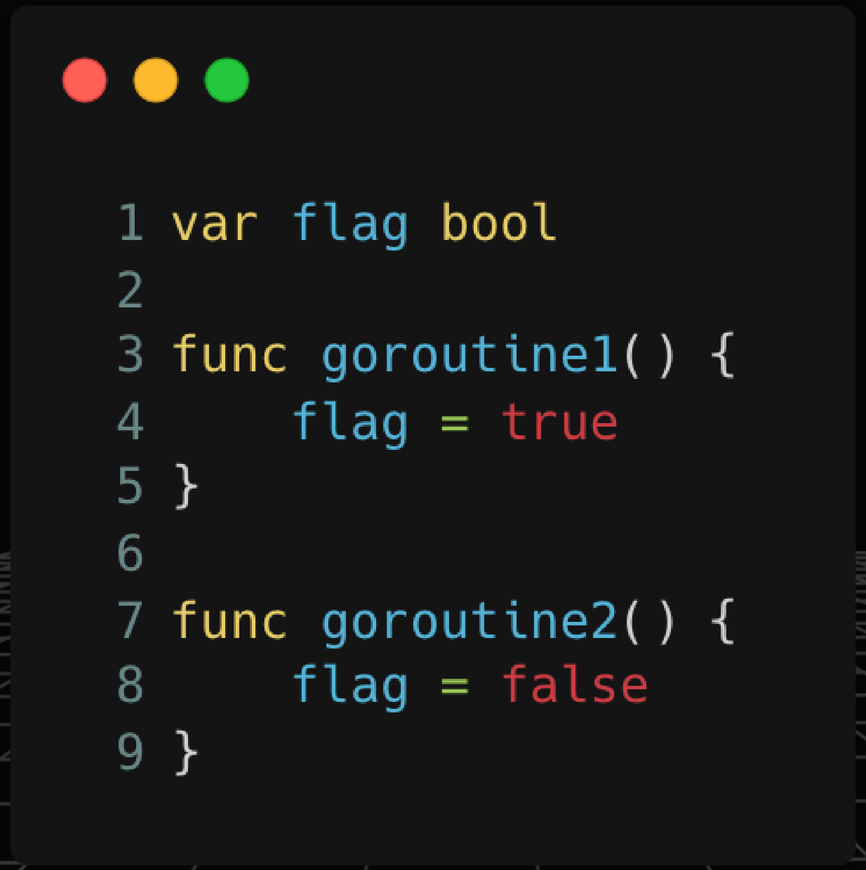
FAQ

Mutex



РАСПРОСТРАНЕННЫЕ ОШИБКИ

КОРРЕКТНО ЛИ ЭТО?



```
1 var flag bool
2
3 func goroutine1() {
4     flag = true
5 }
6
7 func goroutine2() {
8     flag = false
9 }
```

Что может быть загружено\сохранено
в память одной операцией на одной
архитектуре, **не может быть на другой**

Terminal: Concurrency

DATA RACE – ЭТО

Несинхронизированное обращение к одному и тому же участку памяти разных потоков (горутин), как минимум один из которых осуществляет запись

Terminal: Concurrency

RACE CONDITION – ЭТО

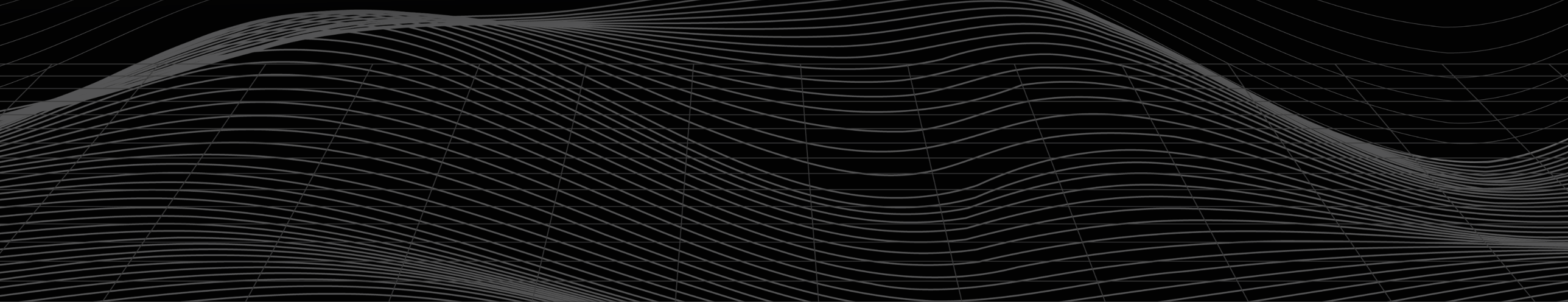
Ошибка проектирования приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода

EXAMPLE

Data race

EXAMPLE

Local mutex



EXAMPLE

Lock granularity

EXAMPLE

Defer with mutex

EXAMPLE

Defer with panic

Инкапсулируйте мьютексы внутри типа,
скрывая всю сложность

EXAMPLE

Synchronized stack

```
data = ["1", "2"]
```

G1

```
stack.Top() // "2"
```

G2

```
stack.Top() // "2"
stack.Pop() // data = ["1"]
```

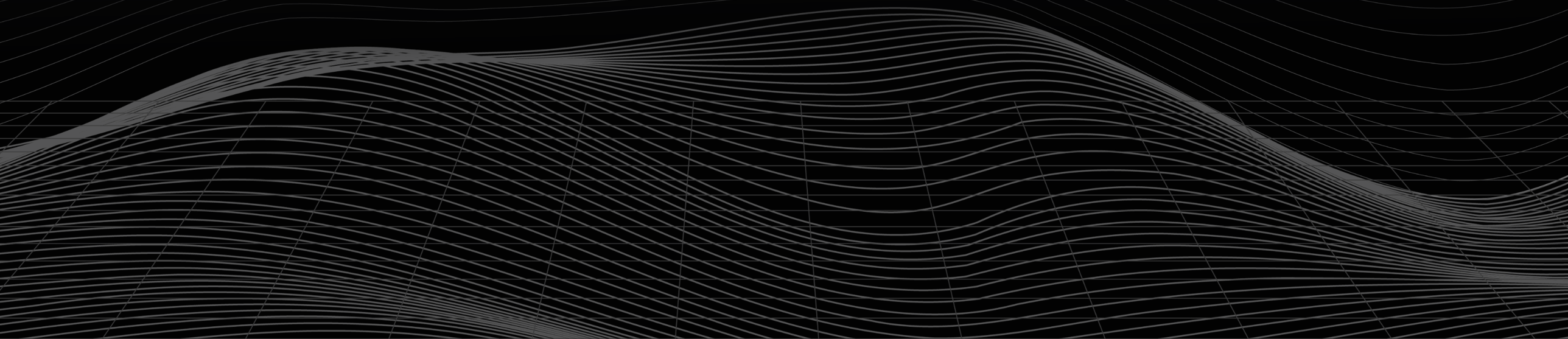
G1

```
stack.Pop() // data = []
```

```
data = []
```

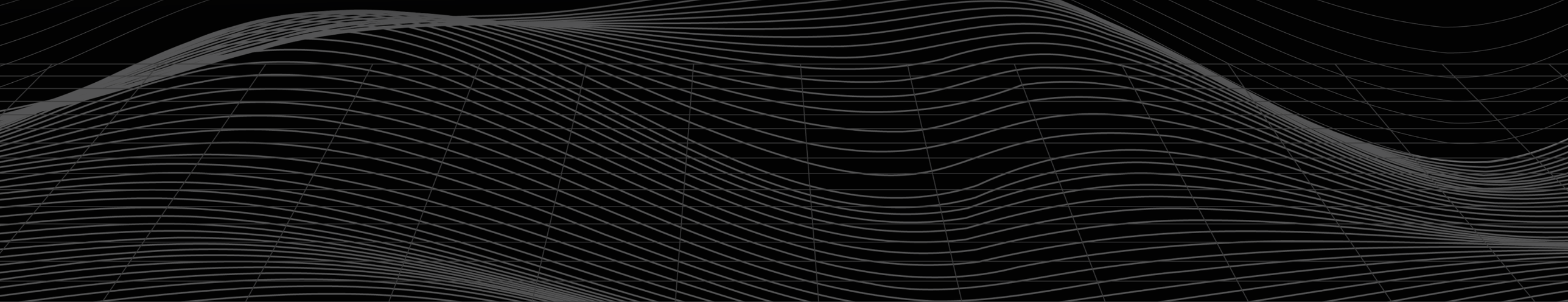
EXAMPLE

Incorrect buffer design



EXAMPLE

Incorrect struct design



EXAMPLE

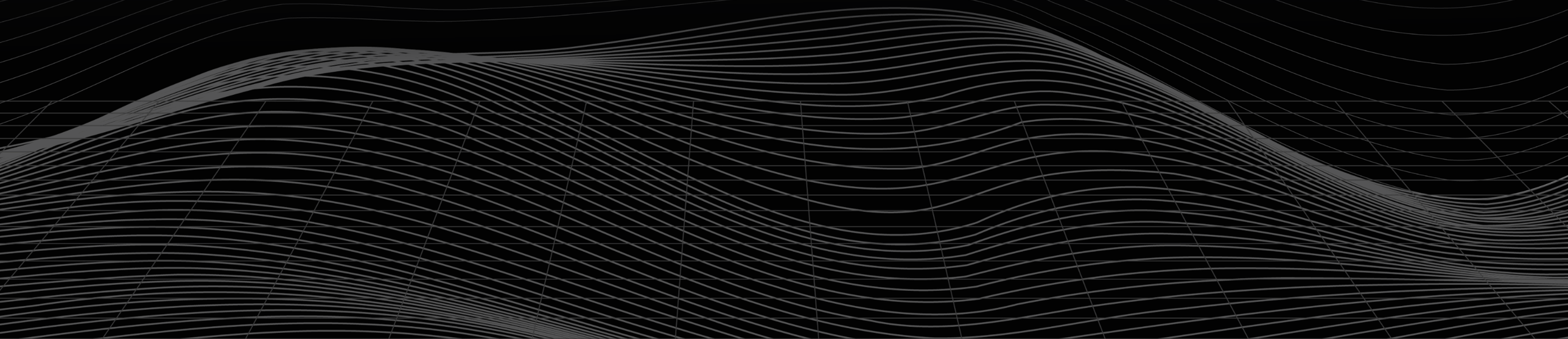
Lockable struct

EXAMPLE

Recursive lock

EXAMPLE

Mutex with common values



Terminal: Concurrency

DEADLOCK

Ситуация в многозадачной среде, при которой несколько потоков (горутин) находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать свое выполнение

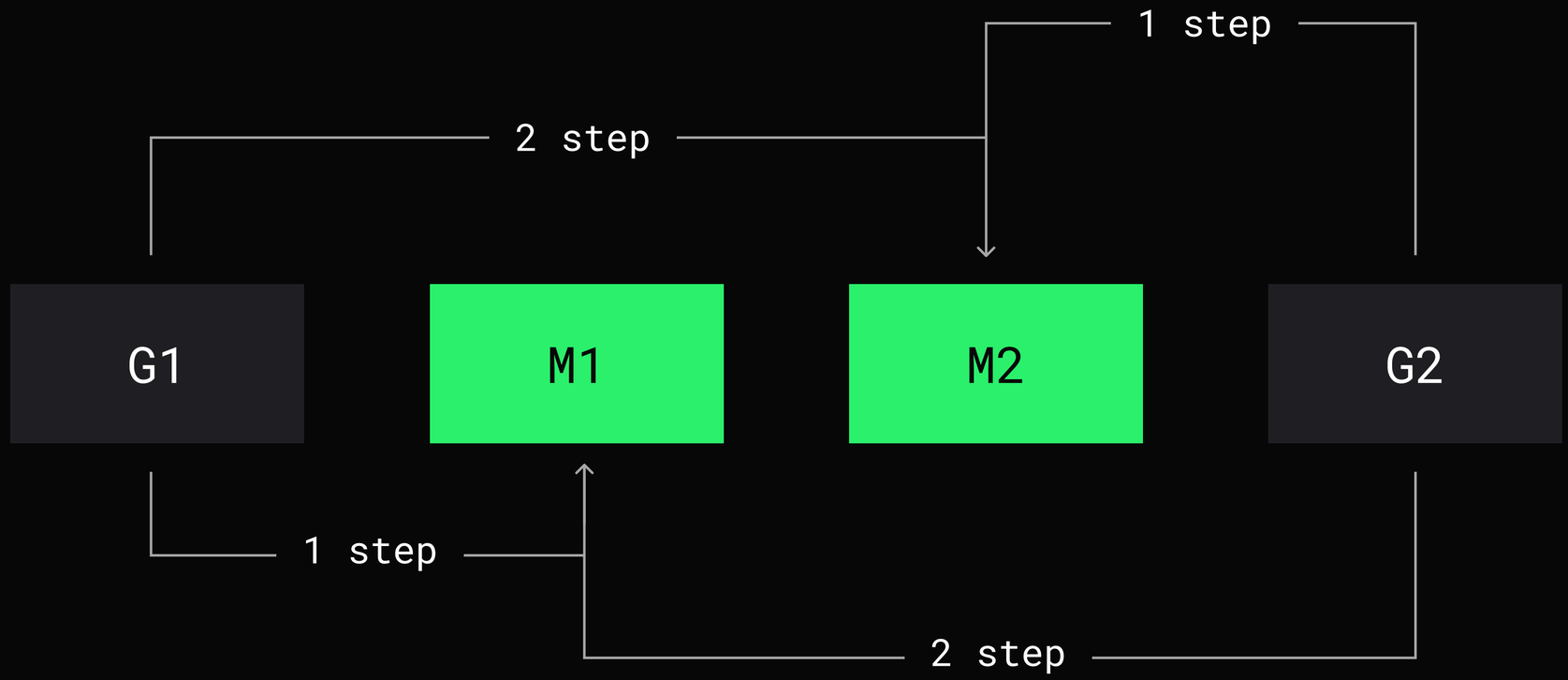
EXAMPLE

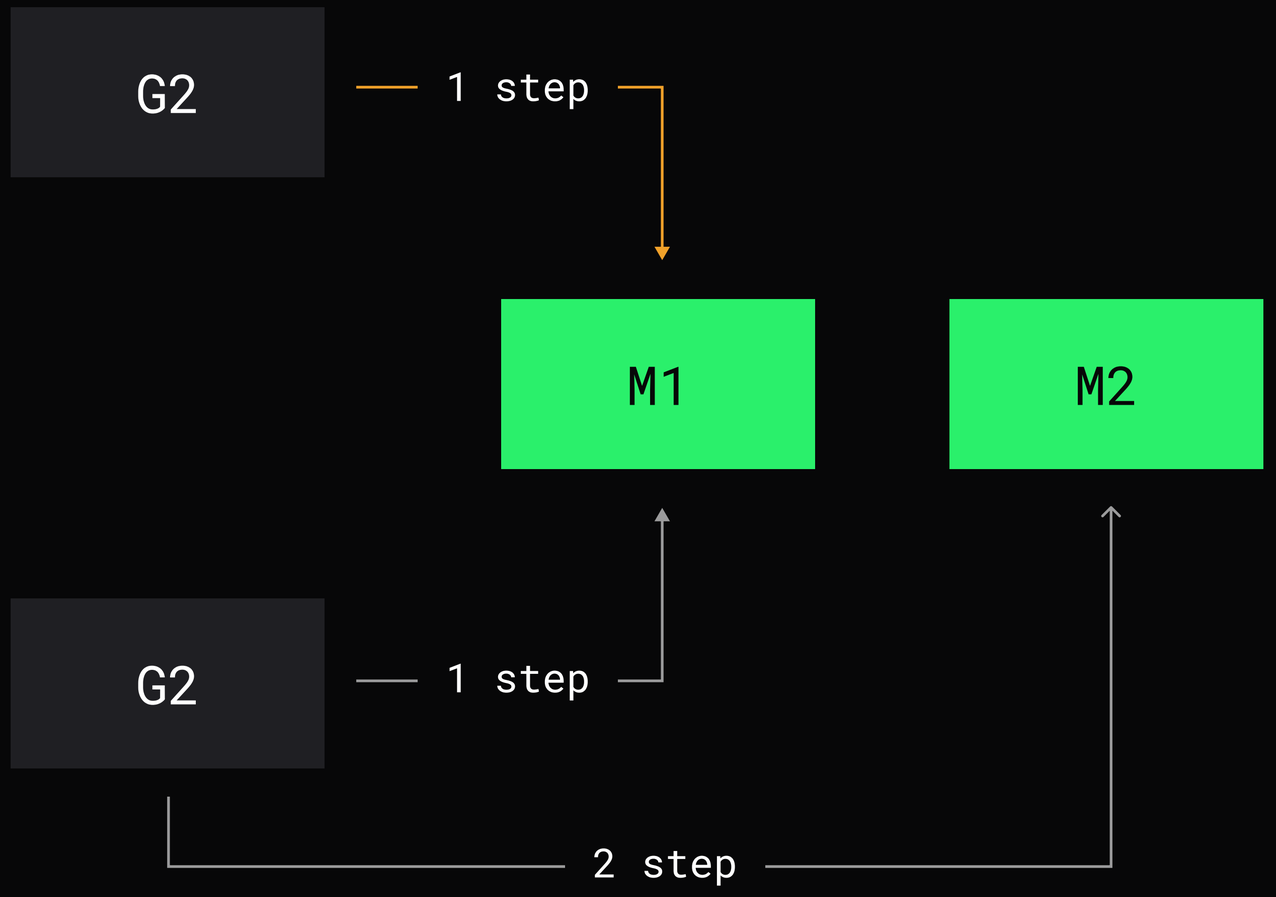
Deadlock

EXAMPLE

Deadlock with work

ВЗАИМНАЯ БЛОКИРОВКА

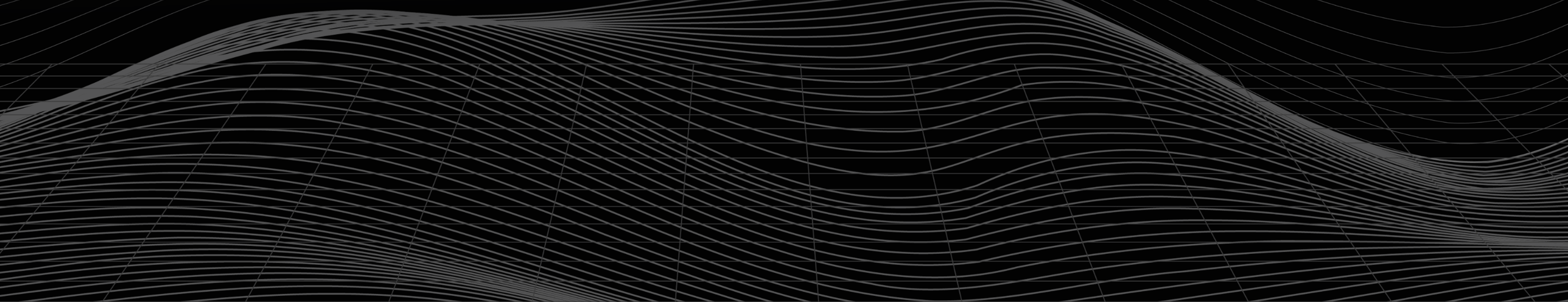



G2
1 step
M1
M2
G2
1 step
2 step

КАКОЕ МИНИМАЛЬНОЕ КОЛИЧЕСТВО МЬЮТЕКСОВ НЕОБХОДИМО ДЛЯ ВЗАИМНОЙ БЛОКИРОВКИ?

EXAMPLE

Mutex operations



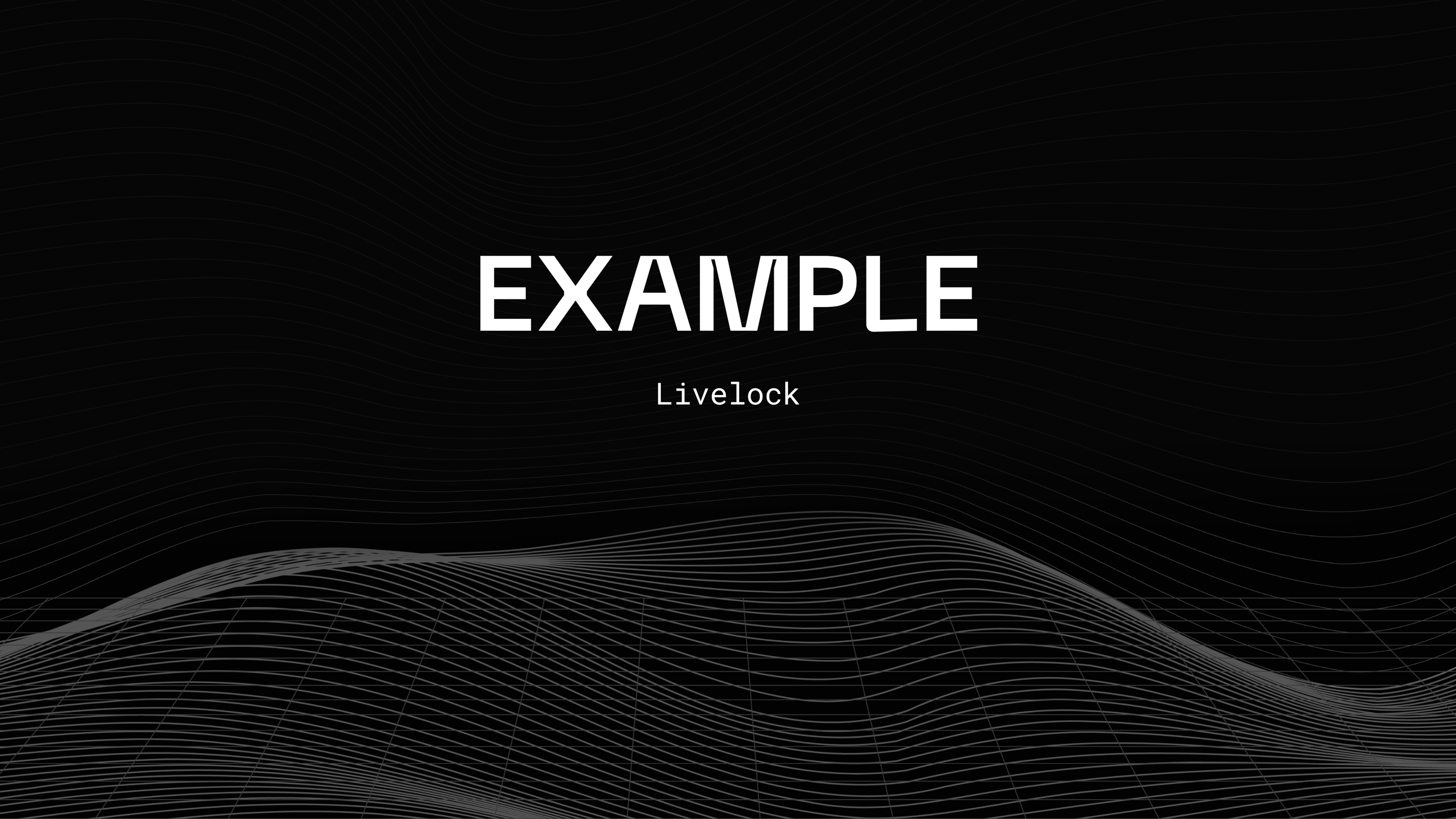
Terminal: Concurrency

LIVELOCK

Ситуация, в которой система не «застревает», а занимается бесполезной работой. Её состояние постоянно меняется, но, тем не менее, она не производит никакой полезной работы

EXAMPLE

Livelock



Terminal: Concurrency

STARVATION

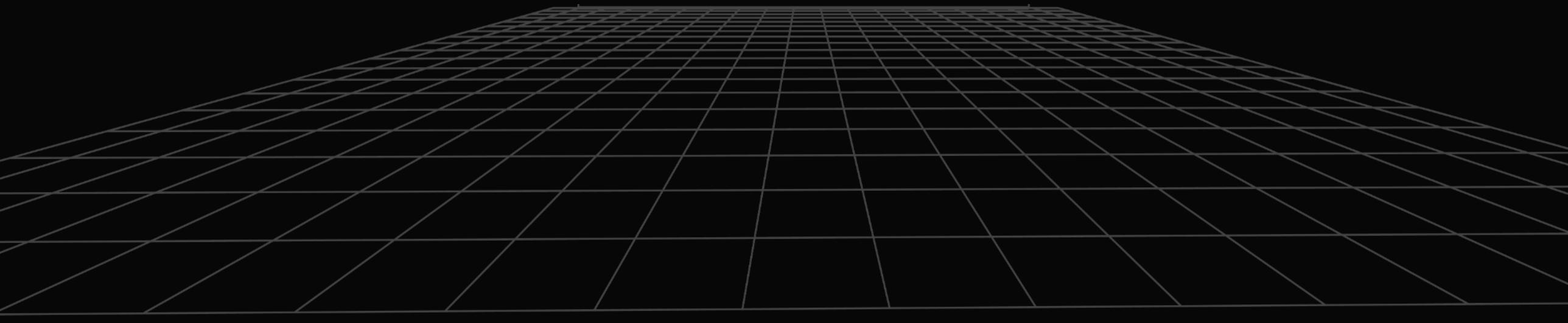
Ситуация, в которой поток (горутина) не может получить все необходимые ресурсы для выполнения его работы (из-за чего он начинает голодать)

EXAMPLE

Starvation

FAQ

Распространенные ошибки



COND

Terminal: Concurrency

УСЛОВНАЯ ПЕРЕМЕННАЯ

Примитив синхронизации, обеспечивающий блокирование одного или нескольких потоков до момента поступления сигнала от другого потока о выполнении некоторого условия

API

```
1 type Cond struct {
2     L Locker
3 }
4
5 func NewCond(l Locker) *Cond // создает Cond с использованием мьютекса
6 func (c *Cond) Broadcast()   // оповещает все горутины
7 func (c *Cond) Signal()      // оповещает одну горутину
8 func (c *Cond) Wait()        // ожидает наступления сигнала
```

EXAMPLE

Cond

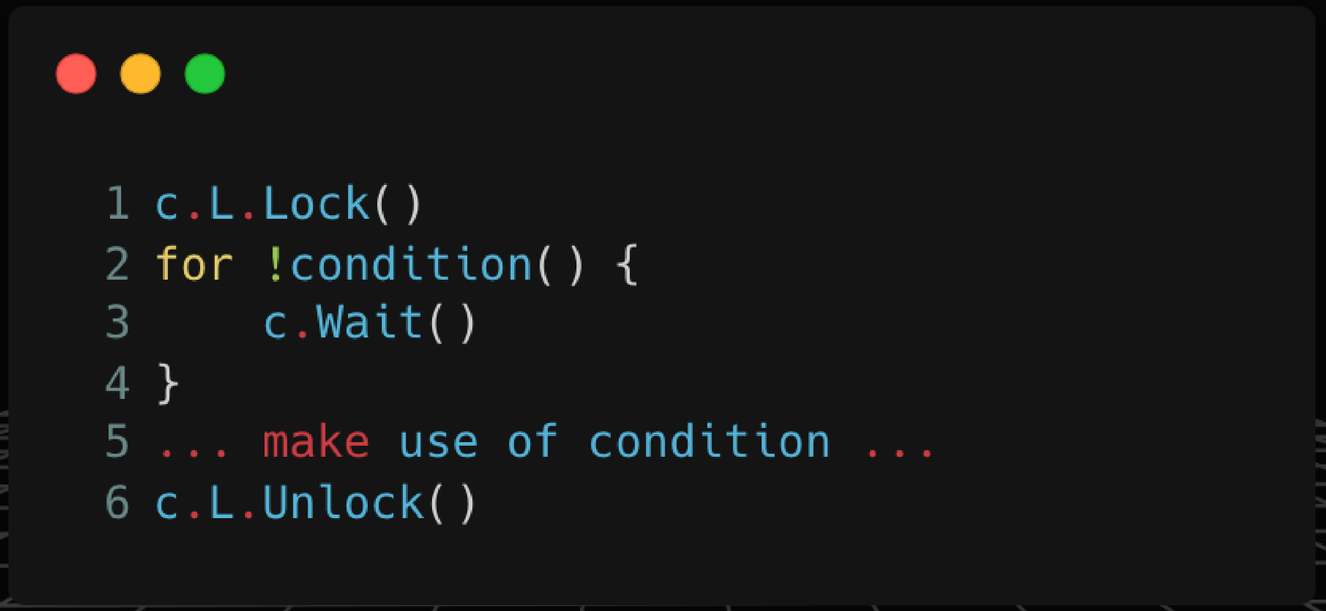
ЧТО ПРОИСХОДИТ ВНУТРИ WAIT()

```
1 c.checker.check()
2 t := runtime_notifyListAdd(&c.notify)
3 c.L.Unlock()
4 runtime_notifyListWait(&c.notify, t)
5 c.L.Lock()
```

EXAMPLE

Cond operations

THE CALLER SHOULD WAIT IN A LOOP



```
1 c.L.Lock()
2 for !condition() {
3     c.Wait()
4 }
5 ... make use of condition ...
6 c.L.Unlock()
```

Terminal: Concurrency

СЕМАФОР

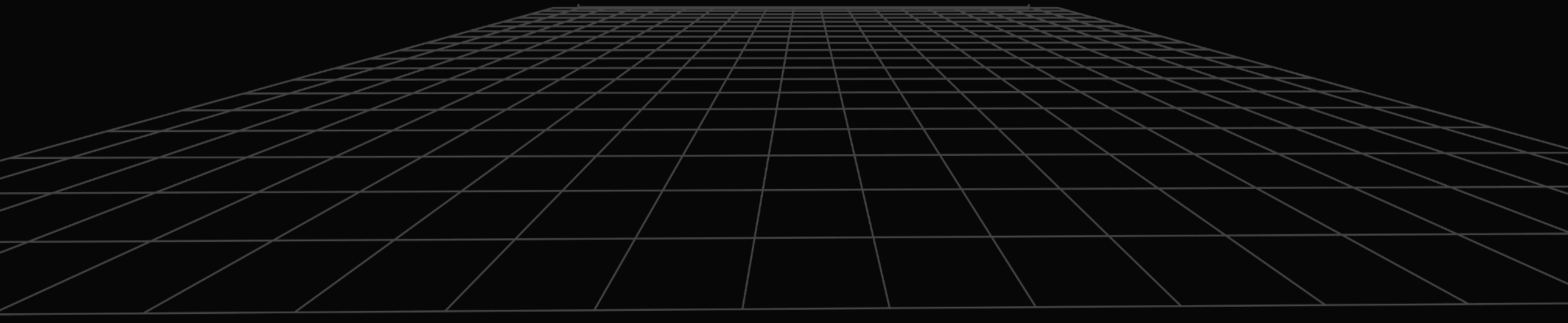
Примитив синхронизации, который используется для управления доступом к общим ресурсам несколькими потоками (горутинами)

EXAMPLE

Semaphore

FAQ

COND



ATOMIC

Terminal: Concurrency

АТОМАРНЫЕ ОПЕРАЦИИ

Операции, которые либо выполняется целиком, либо не выполняется вовсе

EXAMPLE

Incorrect increment

API

```go
func AddInt32(addr *int32, delta int32) (new int32)          // добавить значение
func LoadInt32(addr *int32) int32                            // получить значение
func StoreInt32(addr *int32, val int32)                      // установить значение
func SwapInt32(addr *int32, new int32) (old int32)           // заменить значение и вернуть старое
func CompareAndSwapInt32(addr *int32, old, new int32) bool // заменить значение на new, если оно равно old
```

API

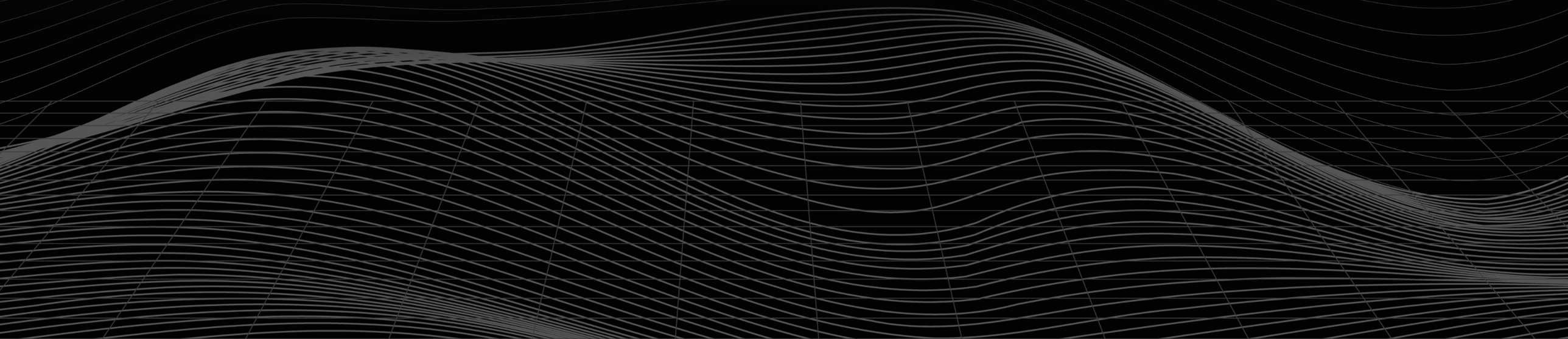
```
1 type Bool struct { ... }
2
3 func (x *Bool) CompareAndSwap(old, new bool) (swapped bool)
4 func (x *Bool) Load() bool
5 func (x *Bool) Store(val bool)
6 func (x *Bool) Swap(new bool) (old bool)
```

EXAMPLE

Atomic performance

EXAMPLE

Compare and swap



initialized = false

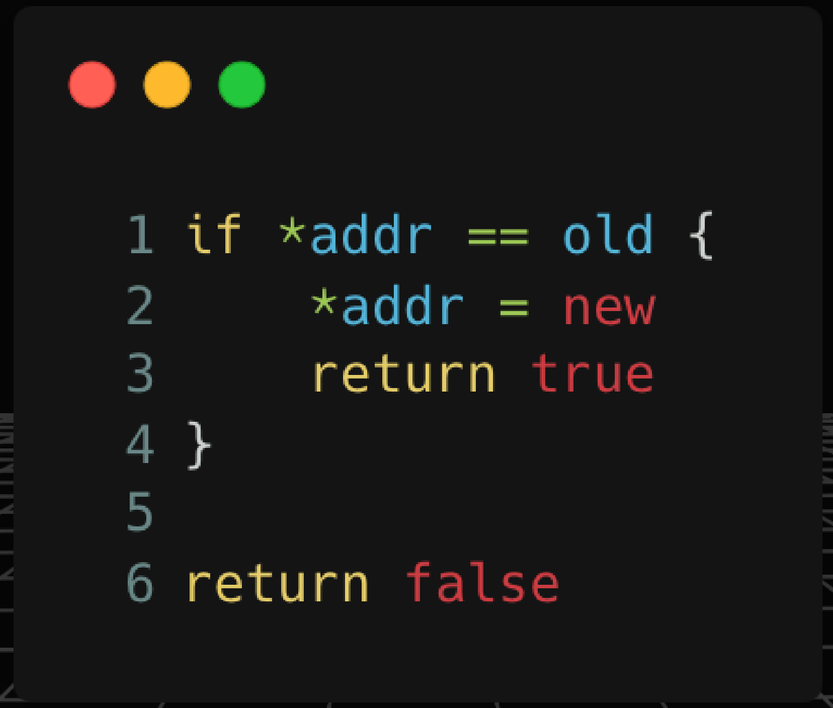
G1 **initialized.Load()** // false

G2 **initialized.Load()** // false

G1 **initialized.Store()**

G2 **initialized.Store()**

УСТРОЙСТВО CAS



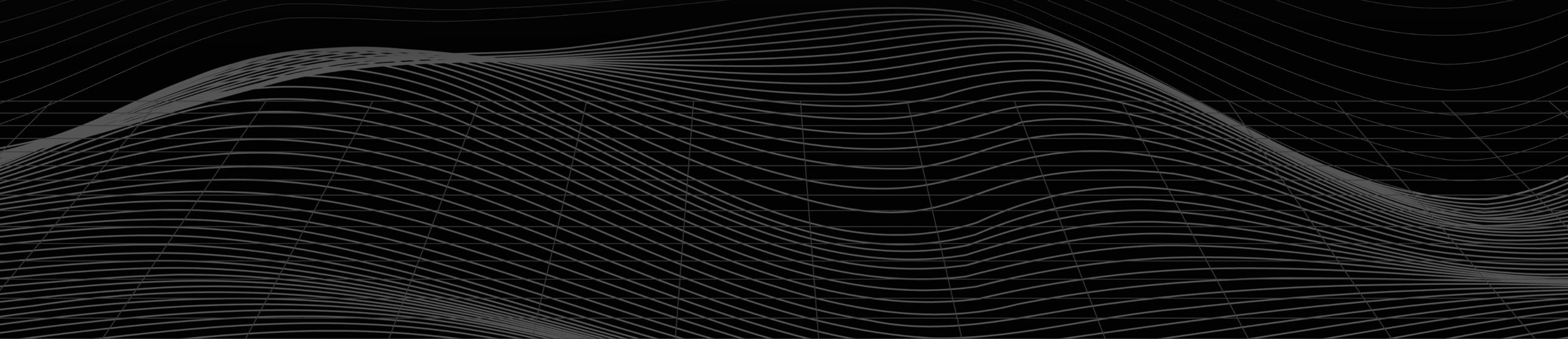
```
1 if *addr == old {
2     *addr = new
3     return true
4 }
5
6 return false
```

EXAMPLE

CAS loop

EXAMPLE

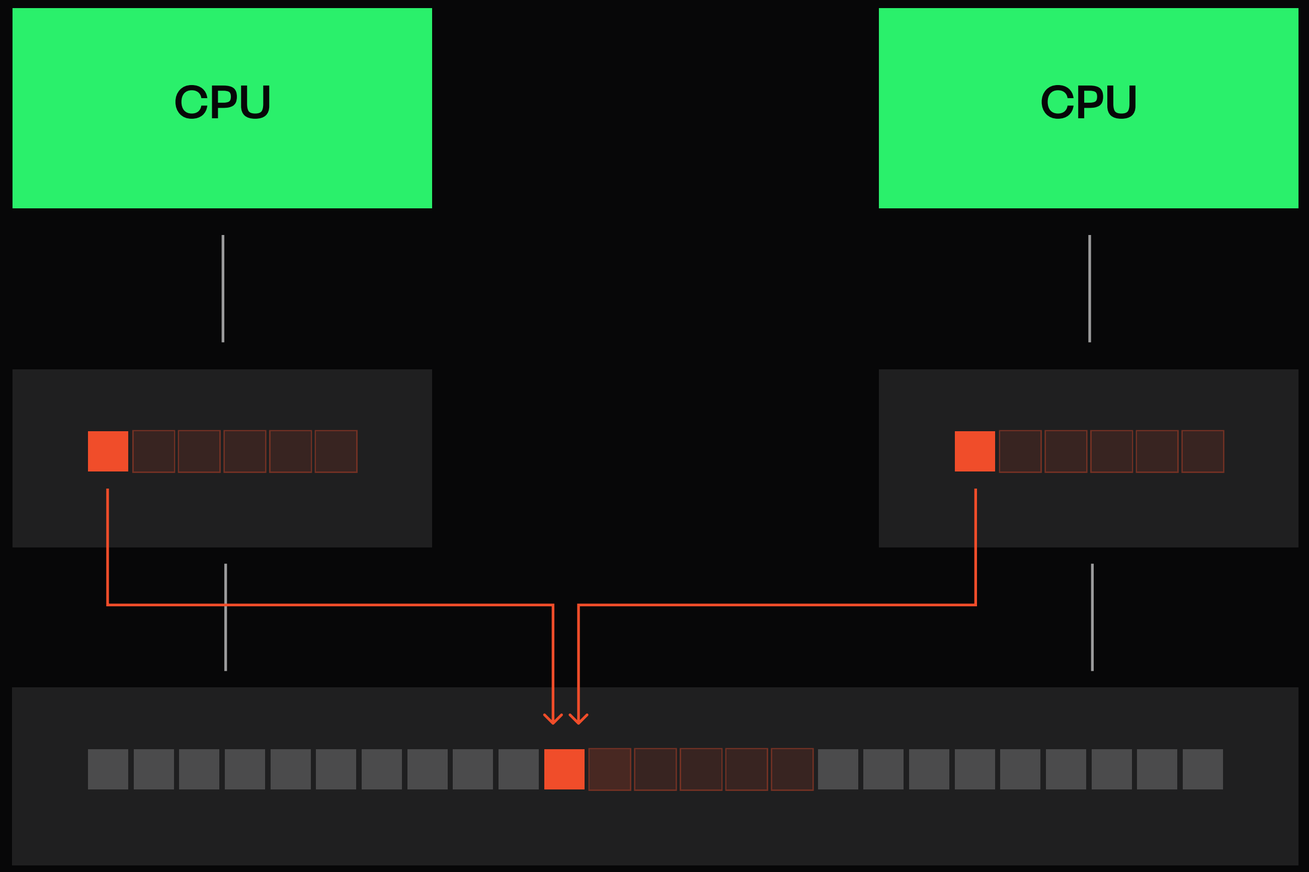
Action during actions



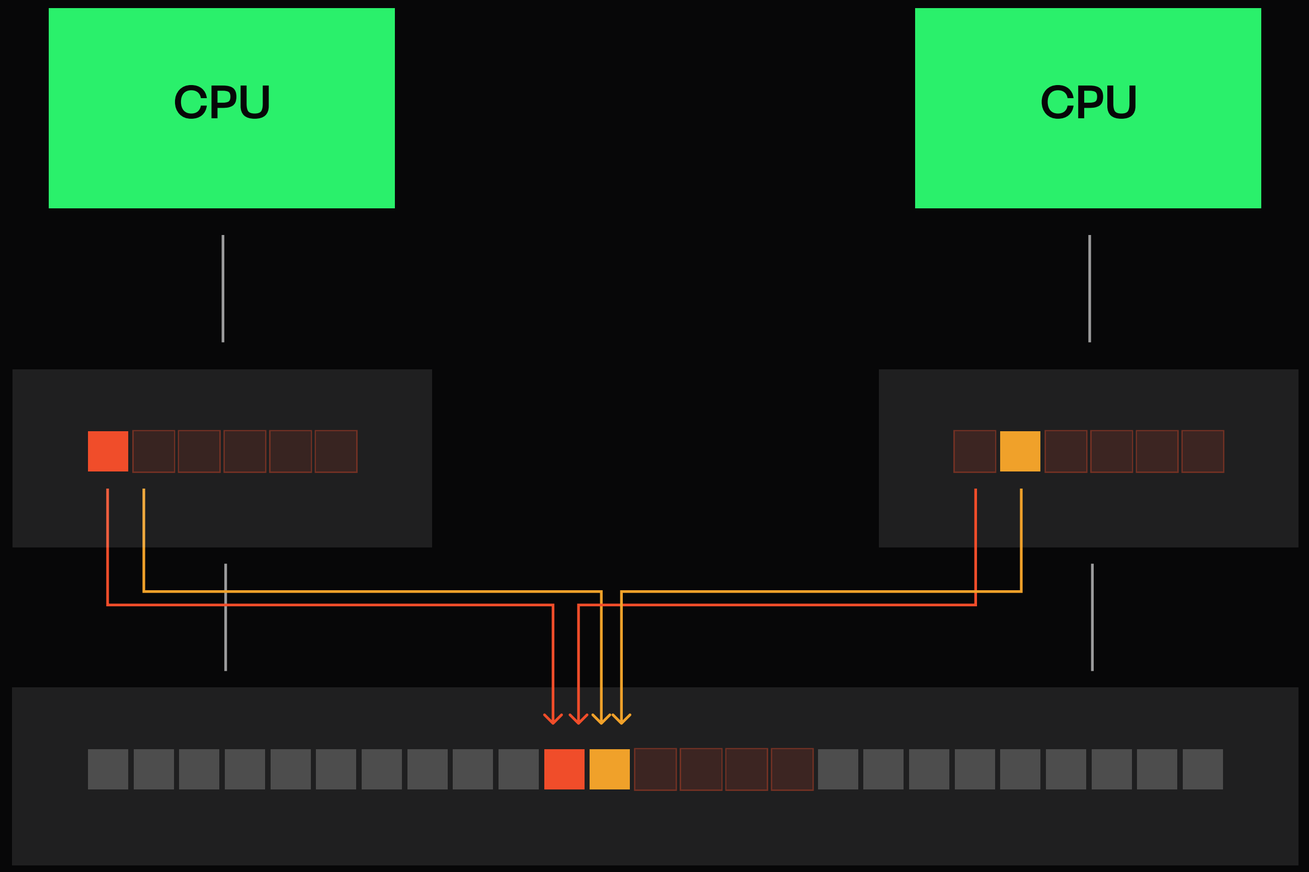
EXAMPLE

False sharing

TRUE SHARING

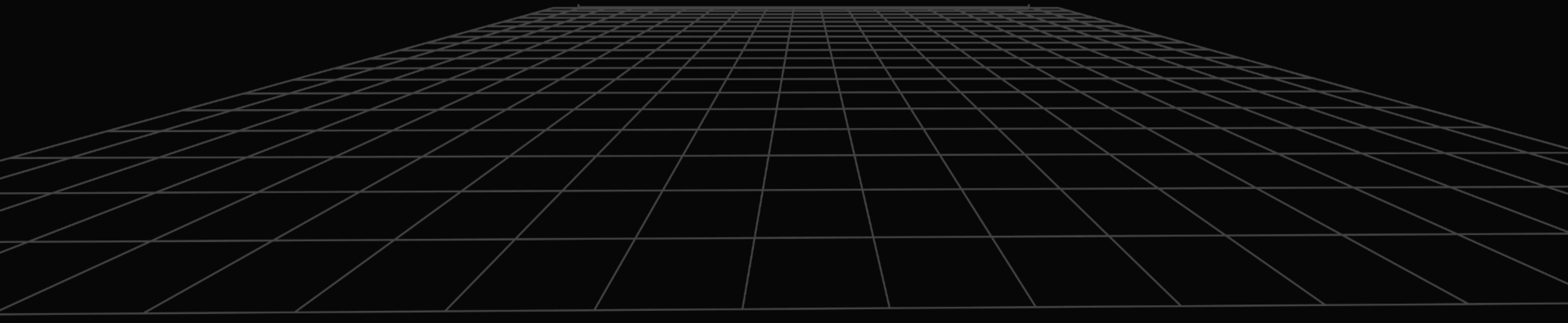


FALSE SHARING



FAQ

Atomic



RWMutex

Terminal: Concurrency

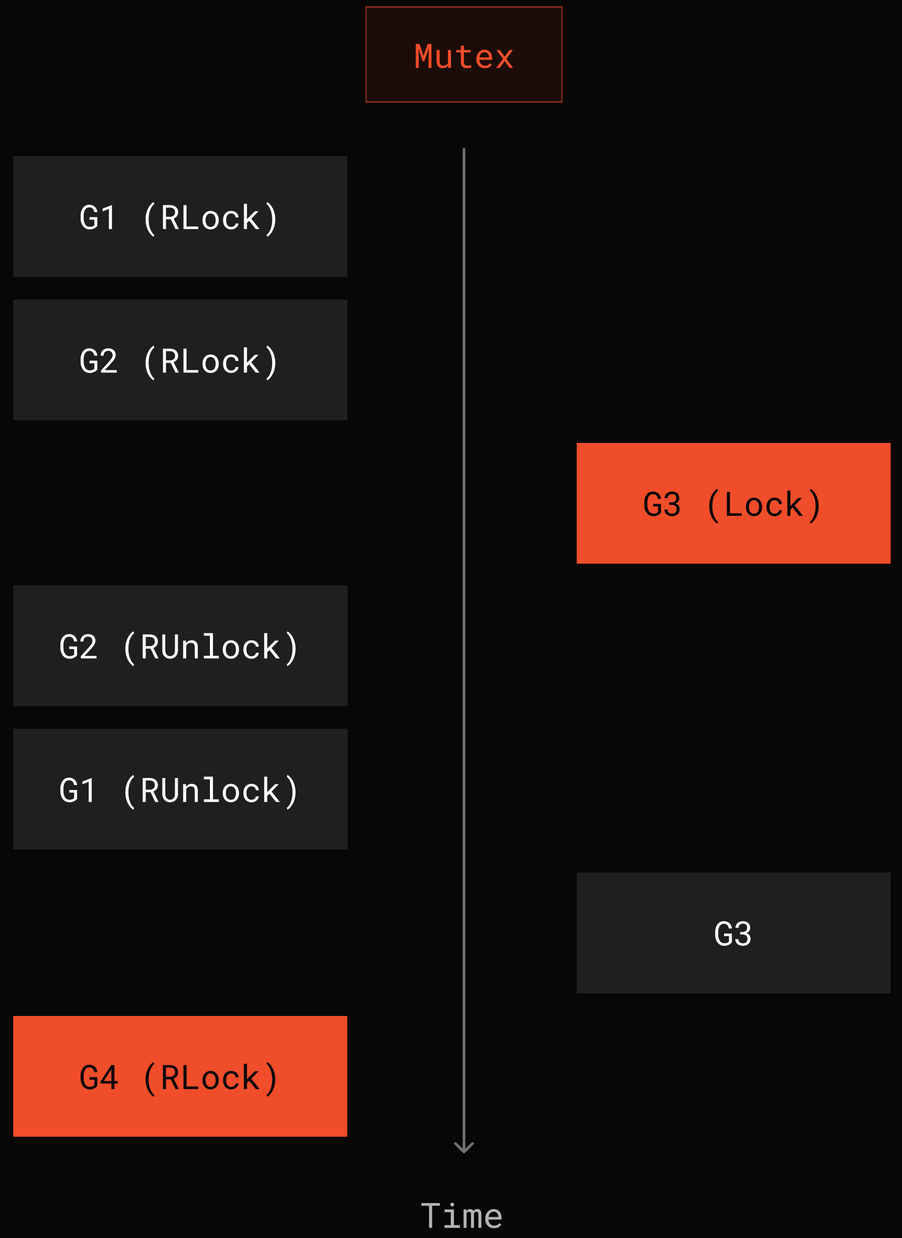
SHARED MUTEX (RW)

Примитив синхронизации, который позволяет нескольким потокам иметь доступ к общему ресурсу для чтения, но блокирует доступ для записи только одному потоку (горутине)

API

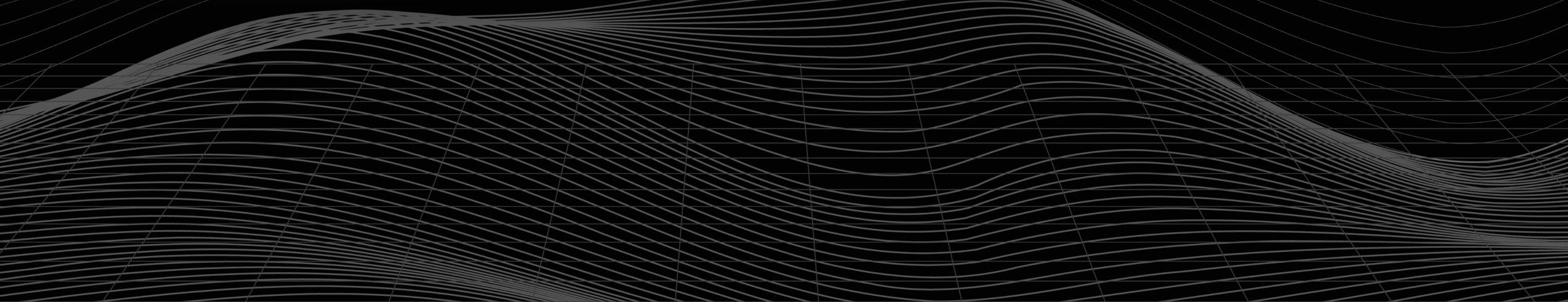
```go
type RWMutex struct { ... }

func (rw *RWMutex) Lock()           // блокировать на запись
func (rw *RWMutex) RLock()          // блокировать на чтение
func (rw *RWMutex) RLocker() Locker // возвращает интерфейс Locker, реализую методы RLock и RUnlock
func (rw *RWMutex) RUnlock()        // разблокировать запись
func (rw *RWMutex) TryLock() bool   // попробовать заблокировать на запись
func (rw *RWMutex) TryRLock() bool  // попробовать заблокировать на чтение
func (rw *RWMutex) Unlock()         // разблокировать чтение
```


Mutex
G1 (RLock)
G2 (RLock)
G3 (Lock)
G2 (RUnlock)
G1 (RUnlock)
G3
G4 (RLock)
Time

EXAMPLE

RWMutex with map

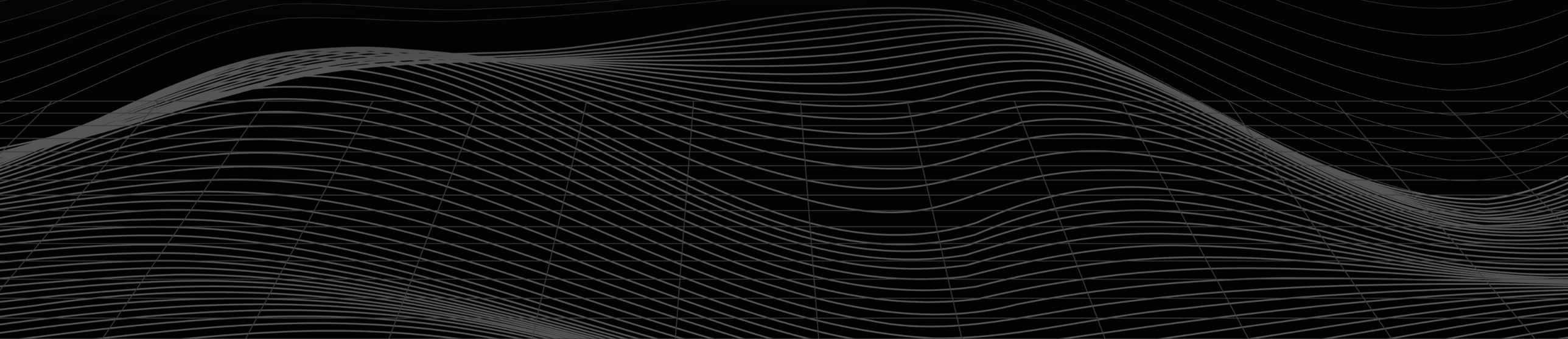


EXAMPLE

RLocker

EXAMPLE

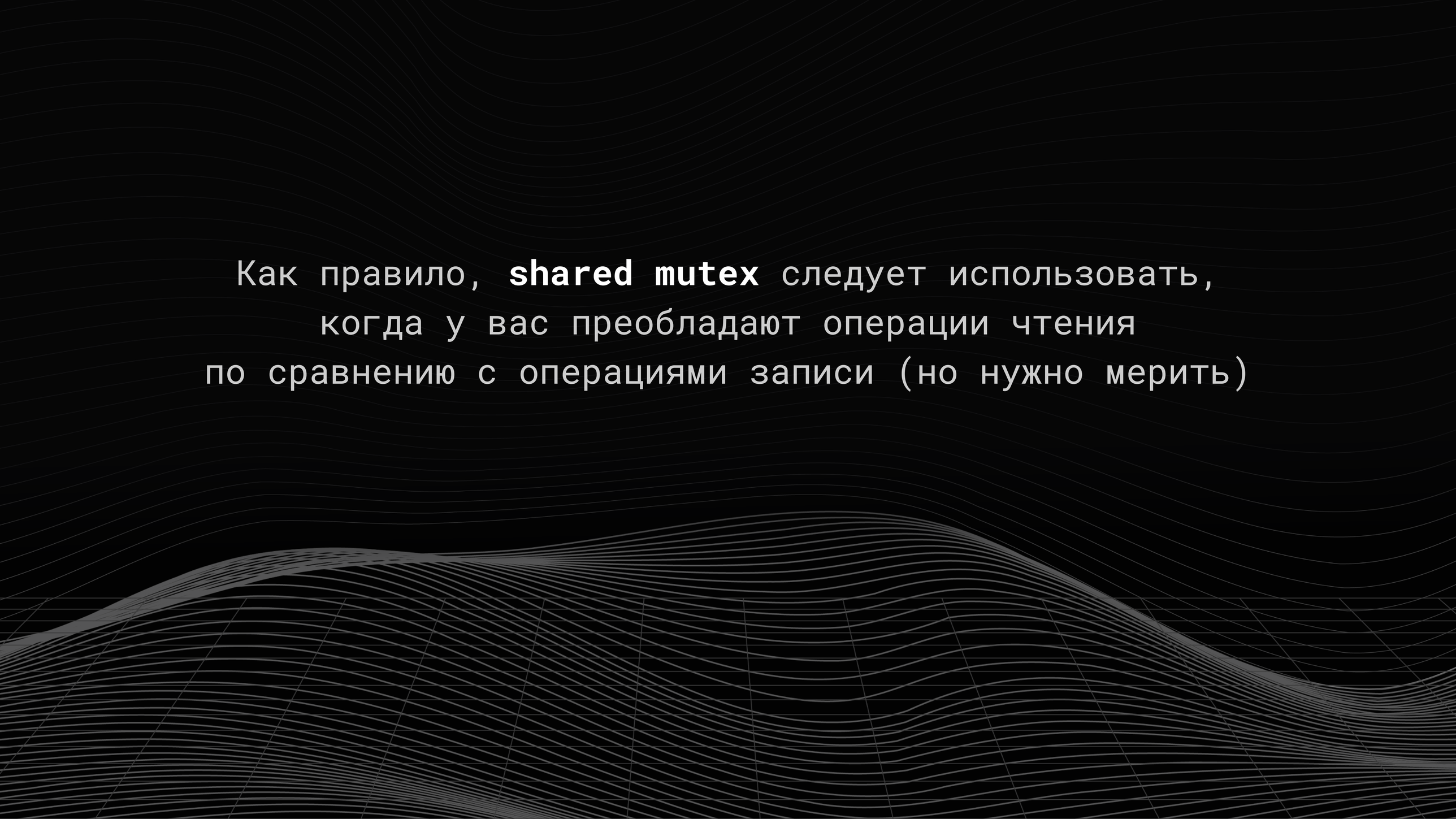
RWMutex operations



EXAMPLE

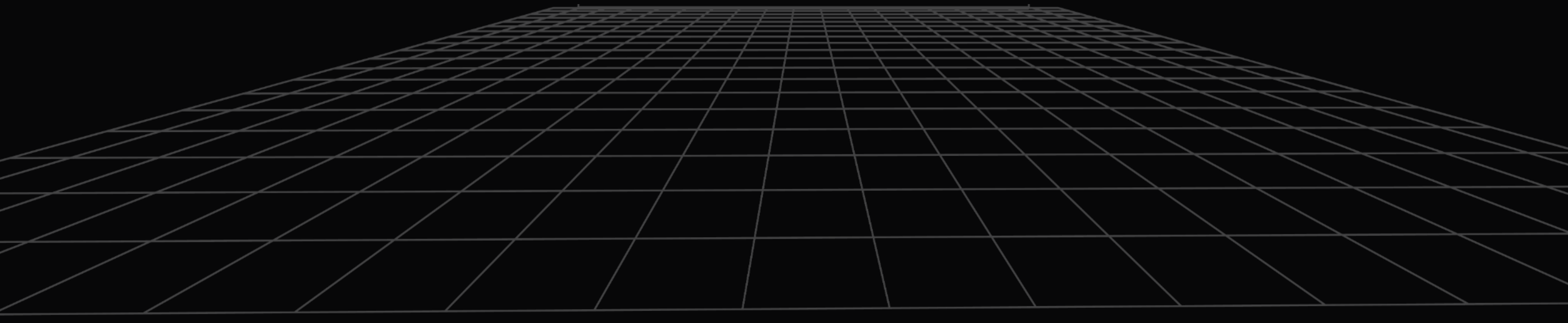
RWMutex performance

Как правило, **shared mutex** следует использовать,
когда у вас преобладают операции чтения
по сравнению с операциями записи (но нужно мерить)



FAQ

RWMutex



ПОЖАЛУЙСТА, ЗАПОЛНИ ОПРОС О ЗАНЯТИИ

Ссылка в чате и в группе участников