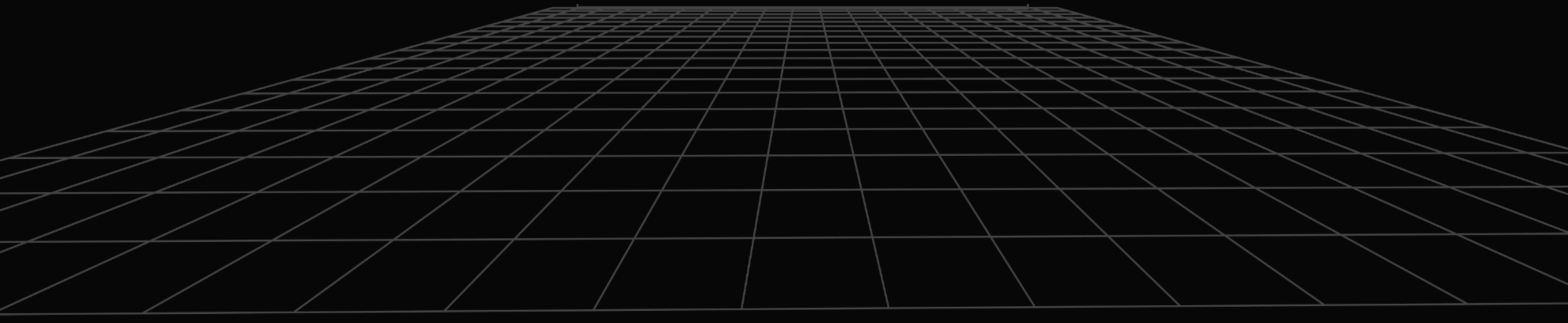
СТРУКТУРЫ

Balun.Courses

ПРОВЕРЬ ЗАПИСЬ

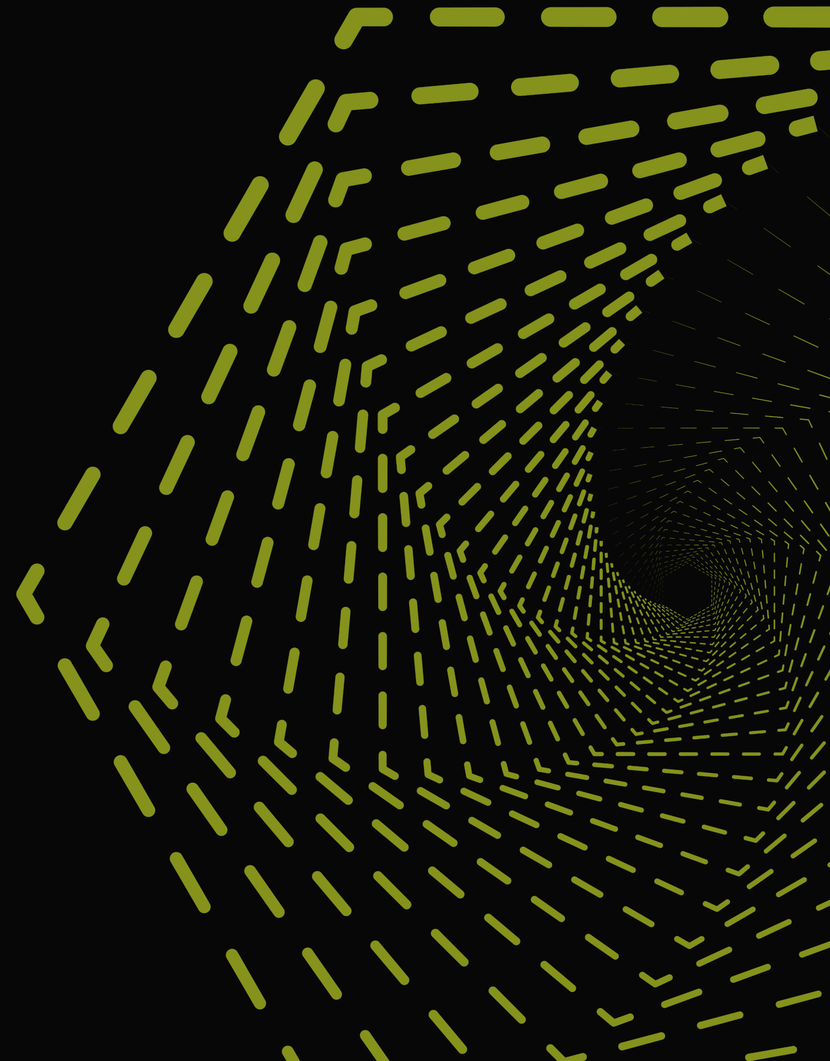


ПРАВИЛА ЗАНЯТИЯ

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

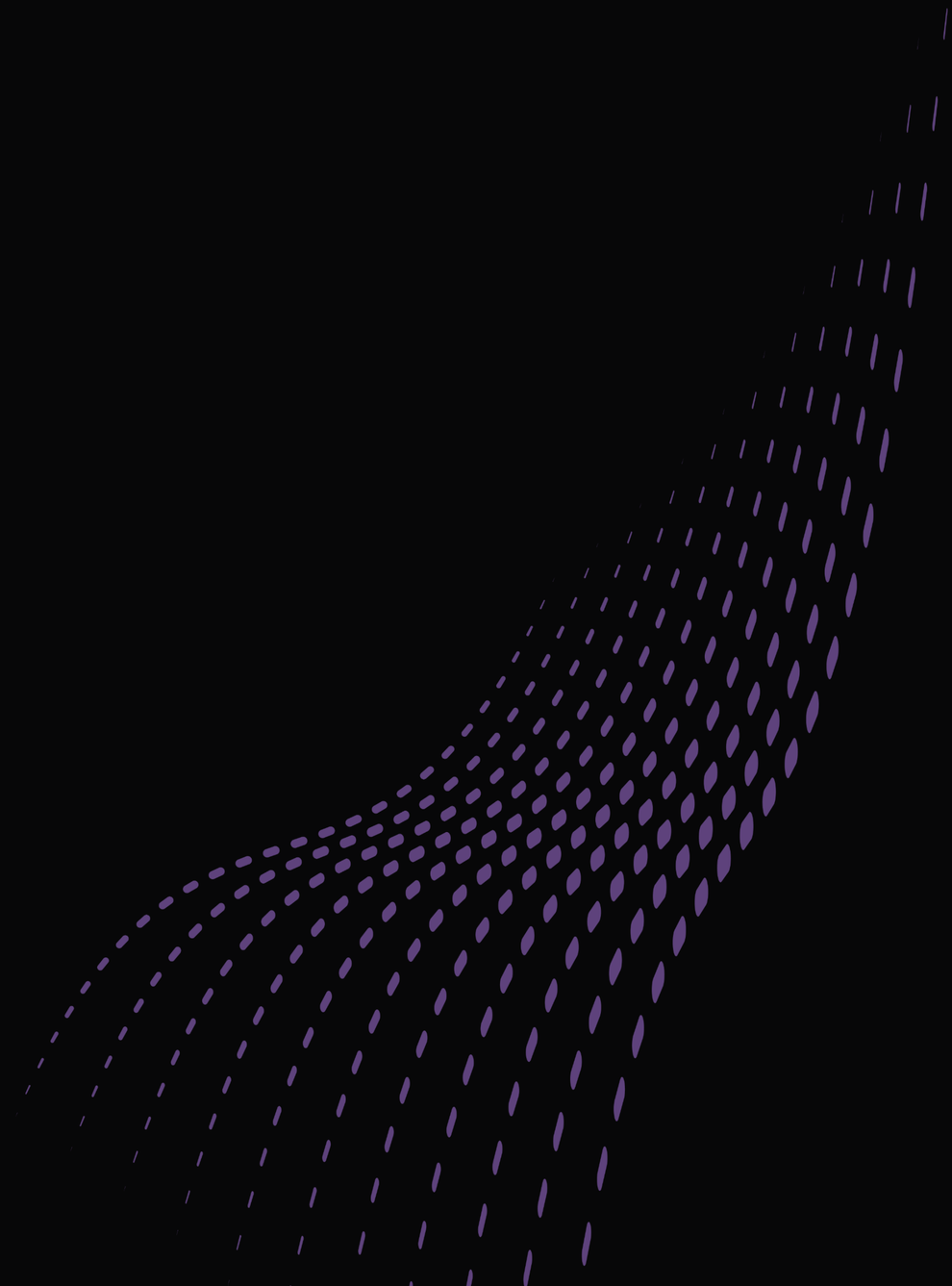
2. вопросы голосом задаем **по поднятой руке в Zoom**

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



ПЛАН ЛЕКЦИИ

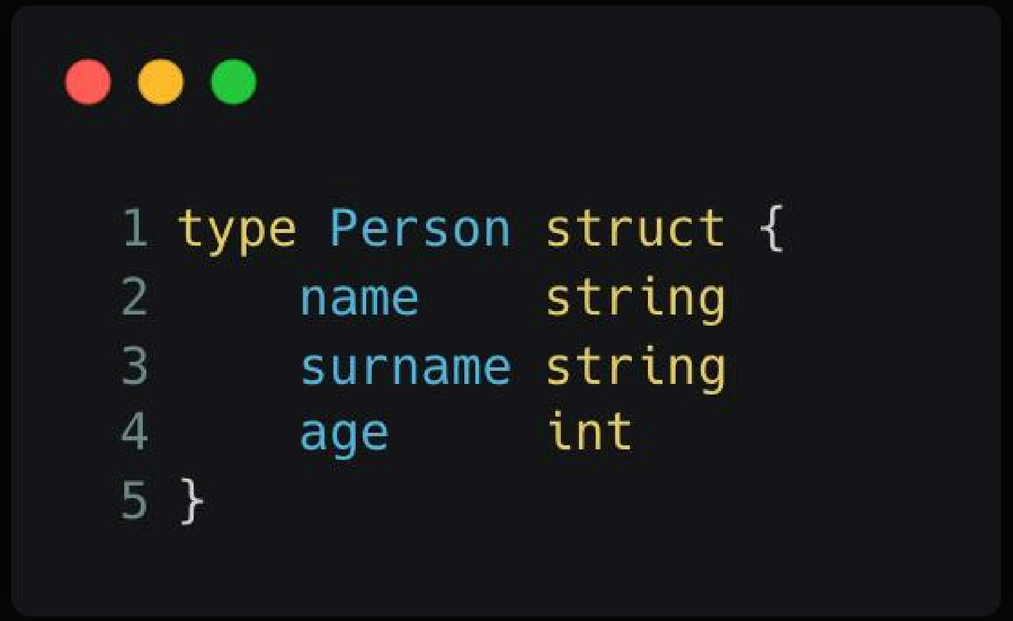
1. Структуры
2. Встраивание типов
3. Выравнивание структур
4. Тонкости структур
5. Паттерны
6. Псевдонимы и определения типов



СТРУКТУРЫ

СТРУКТУРЫ ДАННЫХ

Программная единица, позволяющая хранить и обрабатывать однотипные и/или логически связанные данные



```
type Person struct {
    name    string
    surname string
    age     int
}
```

EXAMPLE

Struct

EXAMPLE

Copy struct

Terminal: deep_go

МЕТОД

В объектно-ориентированном программировании – это **функция или процедура**, принадлежащая какому-то классу или объекту

EXAMPLE

Struct sugar

EXAMPLE

Blank methods

Terminal: deep_go

ПОЛУЧАТЕЛЬ

(Receiver) – представляет собой **указатель на текущий объект или объект структуры** (аналог this в других языка программирования)

EXAMPLE

Different receivers

Получатель должен быть указателем:

- Если метод должен изменить получателя
- Если в получателе есть поле, которое нельзя копировать (например тип из пакета sync)

Получатель следует сделать указателем:

- Если получатель — большой объект (чтобы его не копировать — размер объекта нужно бенчмаркать)

Получатель должен быть значением:

- Если нужно обеспечить неизменность получателя

Получатель следует сделать значением:

- Если получатель является базовым типом (int, float64, string, …)
- Если изменяемые поля не являются частью получателя напрямую, а находятся внутри другой структуры

EXAMPLE

Recursive receiver

По умолчанию **мы можем выбирать тип получателя в виде значения**, если нет веских причин не делать этого, но если возникают сомнения, то использовать указатель

EXAMPLE

Bind receiver

НЕ НУЖНО СМЕШИВАТЬ ТИПЫ ПОЛУЧАТЕЛЕЙ!

EXAMPLE

Nil receiver

Потому что получатель - это синтаксический сахар

```go
type data struct {}

func (d *data) methodWithSugar() {
    // implementation...
}

func methodWithoutSugar(d *data) {
    // implementation...
}
```

EXAMPLE

Implicit method call

GO НЕ ПОДДЕРЖИВАЕТ ОБЪЕДИНЕНИЯ, НО ИХ МОЖНО РЕАЛИЗОВАТЬ

EXAMPLE

Union 1

EXAMPLE

Union 2

У структур в Go могут быть теги (набор пар ключей и значений). Теги опциональные – по умолчанию тег каждого поля равен пустой строке

```go
type User struct {
    Name    string `json:"name" xml:"name"`
    Surname string `json:"surname" xml:"surname"`
}
```

EXAMPLE

Struct inside function

EXAMPLE

Anonymous struct

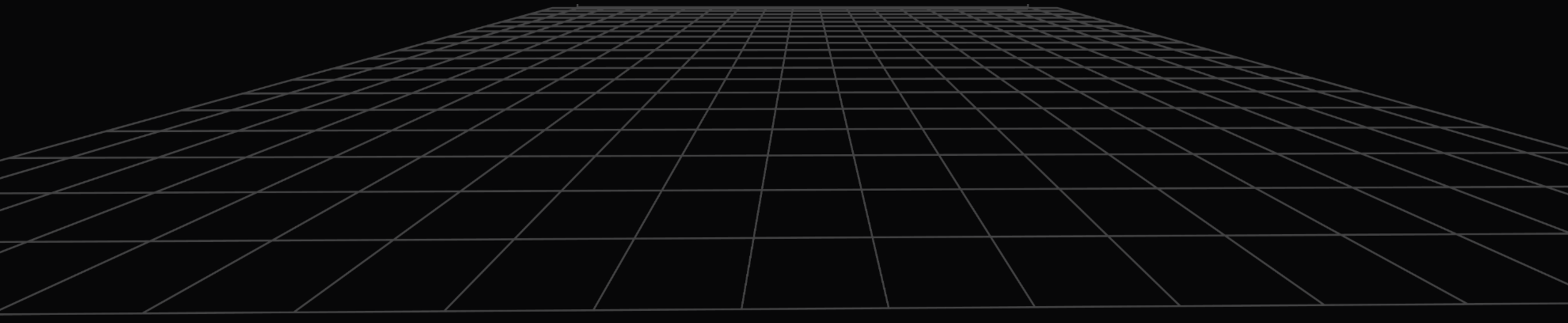
Теги полей и порядок объявлений полей в типе структуры имеют значение для идентификации типа структуры

Два безымянных типов структур идентичны, только если они имеют одинаковую последовательность объявлений полей

Поля идентичны, только если их соответствующие имена, соответствующие типы и соответствующие теги идентичны

FAQ

Структуры

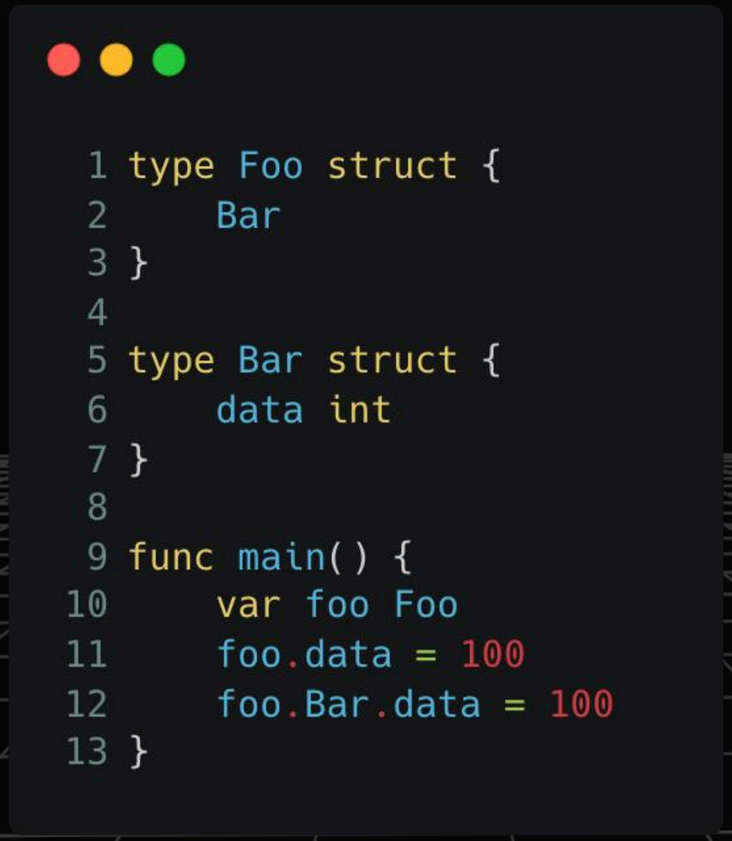


ВСТРАИВАНИЕ ТИПОВ

ВСТРАИВАНИЕ ТИПОВ

```
1 type Foo struct {
2     Bar
3 }
4
5 type Bar struct {
6     data int
7 }
```

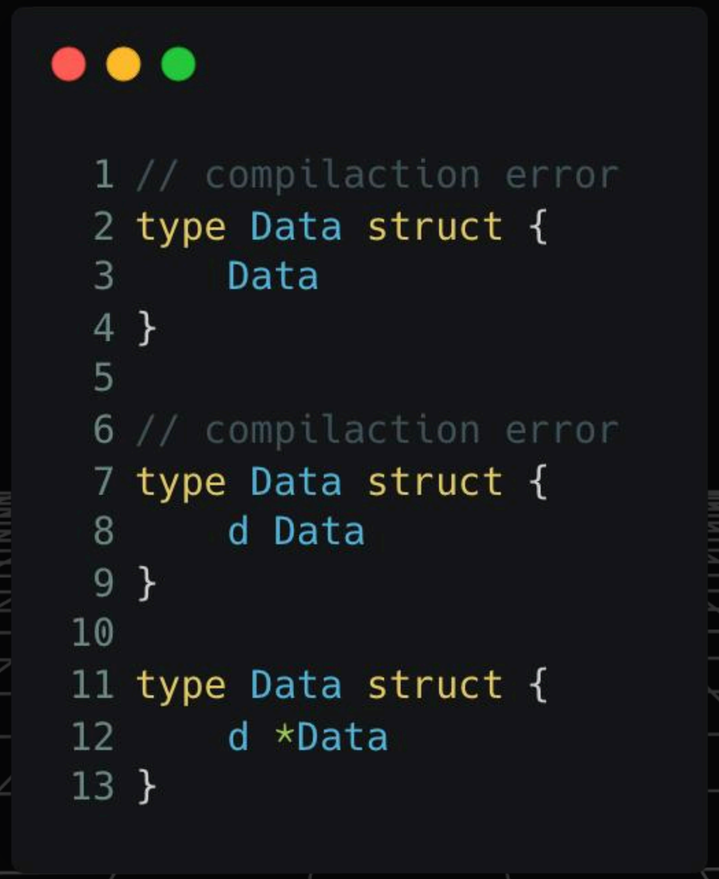
Встроенные поля также еще называют анонимными полями - однако каждое встроенное поле имеет имя, указанное неявно (имя типа встроенного поля действует как имя поля)



```
1 type Foo struct {
2     Bar
3 }
4
5 type Bar struct {
6     data int
7 }
8
9 func main() {
10    var foo Foo
11    foo.data = 100
12    foo.Bar.data = 100
13 }
```

Нельзя рекурсивно встраивать себя

(но можно использовать указатели)



```
1 // compilaction error
2 type Data struct {
3     Data
4 }
5
6 // compilaction error
7 type Data struct {
8     d Data
9 }
10
11 type Data struct {
12     d *Data
13 }
```

EXAMPLE

Skipping selectors

Поэтому их и называют еще **анонимными полями**
– из-за того, что их можно пропустить!

EXAMPLE

Ambiguous selectors

НУЖНО ЯВНО УКАЗЫВАТЬ

к какой встроенной структуре происходит обращение

Terminal: question

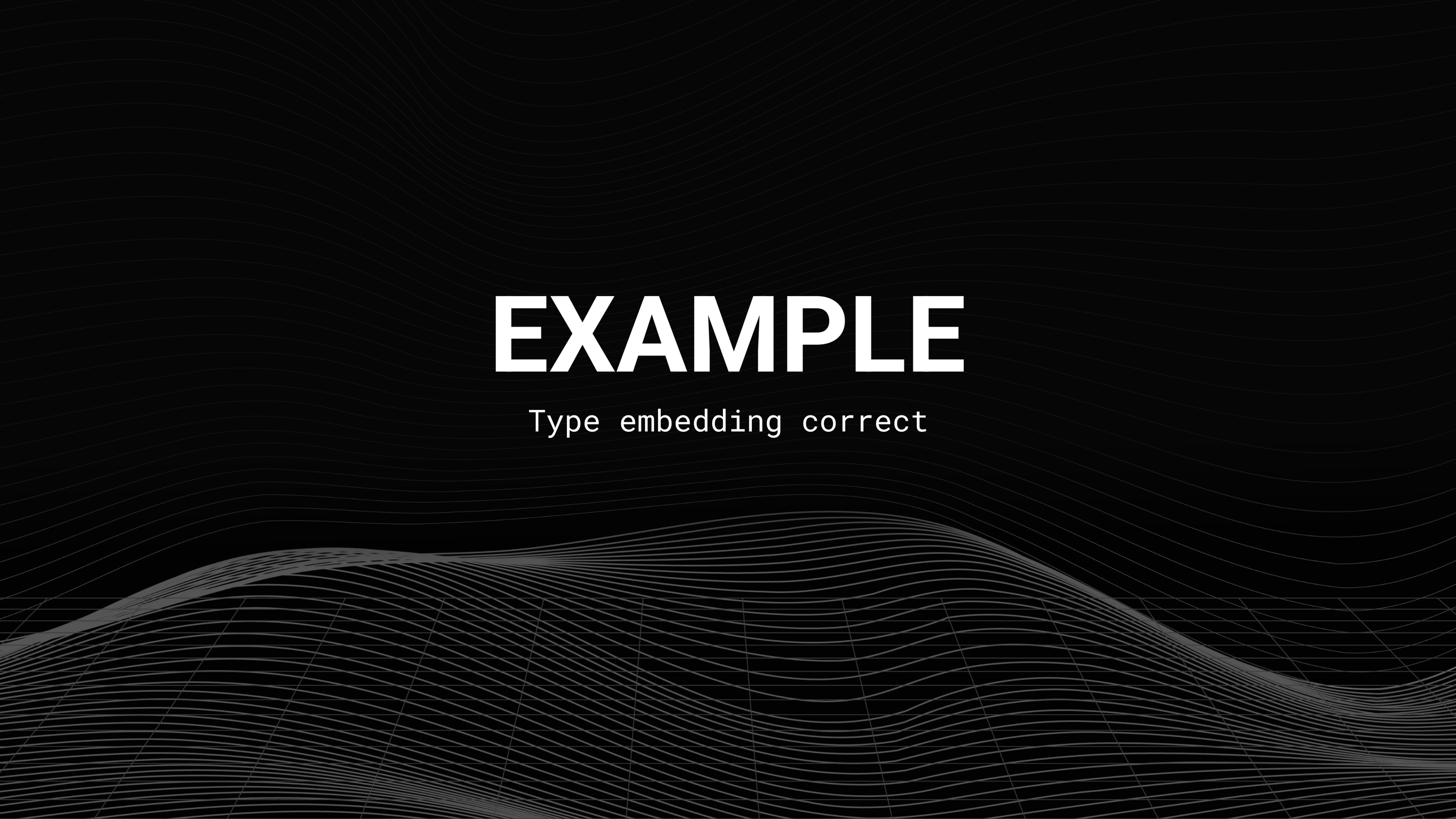
КОГДА ИСПОЛЬЗОВАТЬ ВСТРАИВАНИЕ?

EXAMPLE

Type embedding incorrect

EXAMPLE

Type embedding correct



В GO НЕТ НАСЛЕДОВАНИЯ!

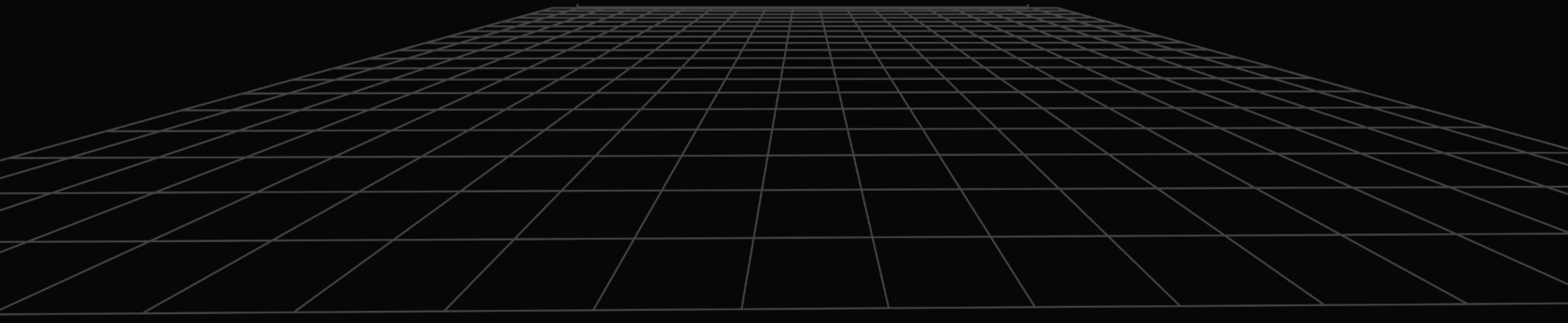
EXAMPLE

Inheritance

Методы с одинаковой сигнатурой «перекрываются» при встраивании

FAQ

Встраивание типов



ВЫРАВНИВАНИЕ СТРУКТУР

Terminal: question + ⌄ ⚙ –

КАК УЗНАТЬ РАЗМЕР ОБЪЕКТА СТРУКТУРЫ?

Terminal: question + ⚙ –

ВЛИЯЕТ ЛИ КОЛИЧЕСТВО МЕТОДОВ НА РАЗМЕР ОБЪЕКТА СТРУКТУРЫ?

EXAMPLE

Struct alignment 1

В GO ИСПОЛЬЗУЕТСЯ

«ТРЕБУЕМОЕ ВЫРАВНИВАНИЕ»

Его значение равно размеру памяти, требующемуся самому большому полю в структуре

> i То есть, если в структуре есть только поля int32, то выравнивание составит 4 байта, а если есть и int32, и int64 — то 8 байтов.

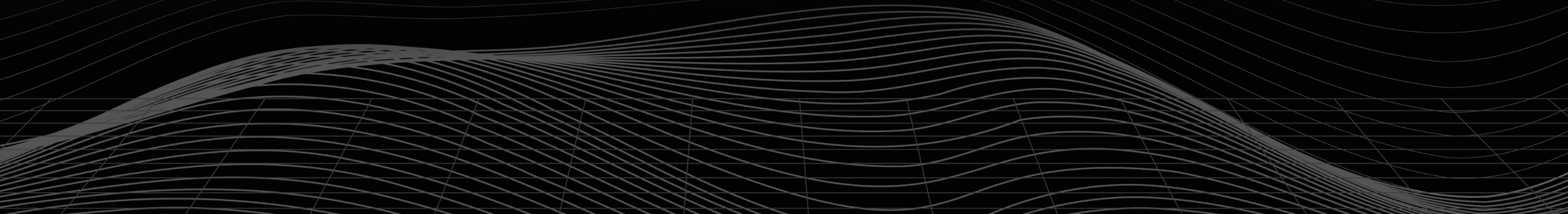

```
type data1 struct {
    aaa bool
    bbb int32
    ccc bool
}
```

4 bytes

4 bytes

```
1 type data2 struct {
2     aaa int32
3     bbb bool
4     ccc bool
5 }
```

Компилятор не меняет порядок полей в структуре
и не может оптимизировать такие случае - **но мы можем**



НЕИСПОЛЬЗУЕМАЯ ПАМЯТЬ ЗАПОЛНЯЕТСЯ НУЛЯМИ

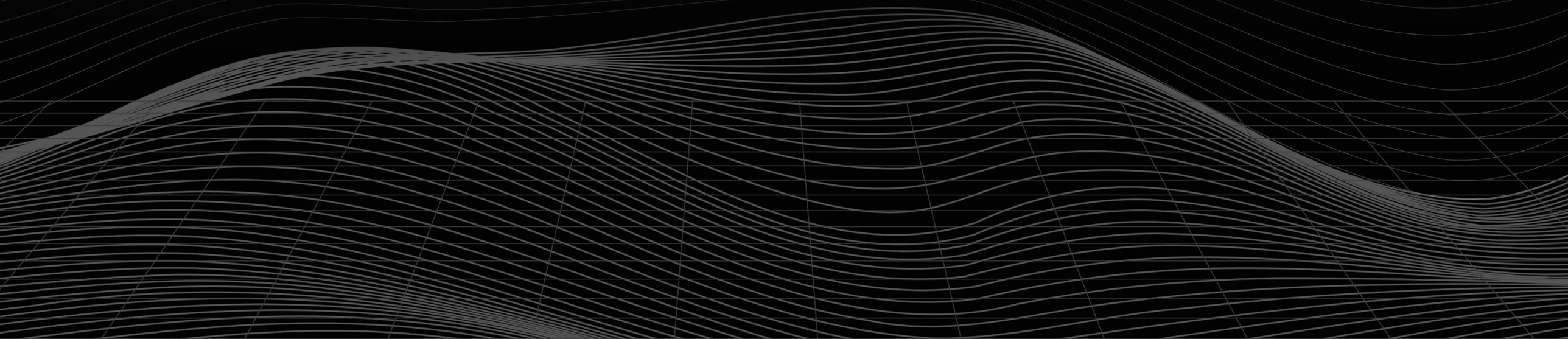
EXAMPLE

Struct alignment 2

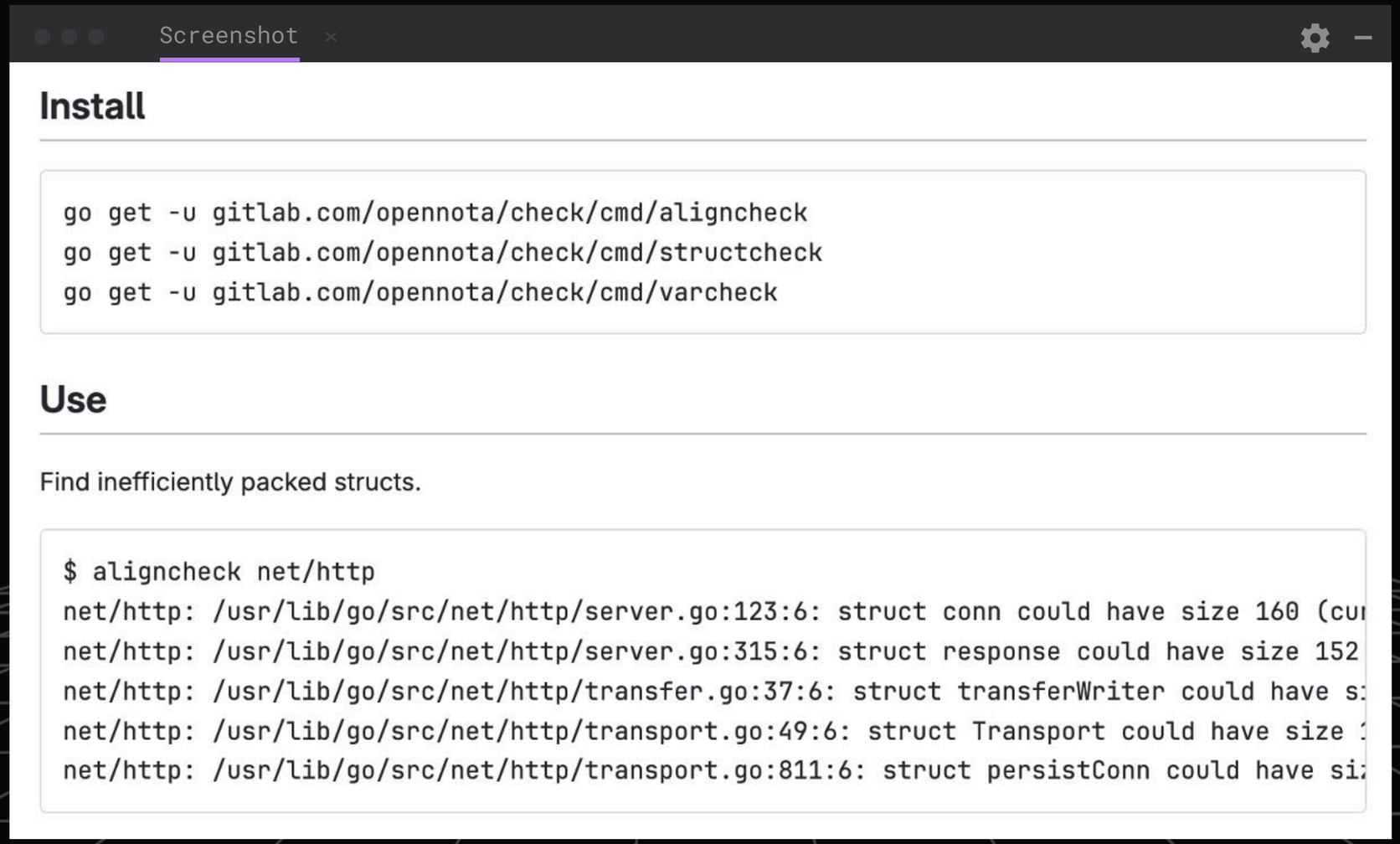
Для массивов выравнивание идентично выравниванию типа элемента массива

EXAMPLE

Alignment functions



ALIGNCHECK


Screenshot
Install
go get -u gitlab.com/opennota/check/cmd/aligncheck
go get -u gitlab.com/opennota/check/cmd/structcheck
go get -u gitlab.com/opennota/check/cmd/varcheck
Use
Find inefficiently packed structs.
$ aligncheck net/http
net/http: /usr/lib/go/src/net/http/server.go:123:6: struct conn could have size 160 (cu
net/http: /usr/lib/go/src/net/http/server.go:315:6: struct response could have size 152
net/http: /usr/lib/go/src/net/http/transfer.go:37:6: struct transferWriter could have s
net/http: /usr/lib/go/src/net/http/transport.go:49:6: struct Transport could have size
net/http: /usr/lib/go/src/net/http/transport.go:811:6: struct persistConn could have si

ВНИМАНИЕ!

НЕ СПЕШИТЕ!

Существует большой соблазн броситься и сразу начать править все свои структуры, чтобы они занимали как можно меньше места, но это темный путь, и не стоит становиться на эту дорожку раньше срока

Почти всегда, **гораздо важней читаемость кода**, чем такие оптимизации. Важно понимать, по какой причине у значения типа именно такой размер и что вообще происходит, а уже в случае необходимости заниматься оптимизацией

Terminal: question + ⚙ –

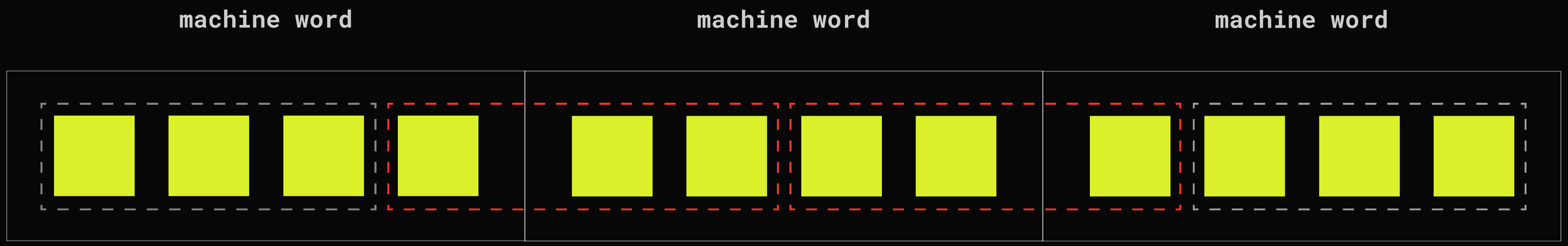
ЗАЧЕМ НУЖНО ВЫРАВНИВАНИЕ?

Большинство современных процессоров читают данные из памяти не отдельными байтами, а блоками по 2-4-8 байт - соответственно, если у вас данные не выравнены в памяти и начало попадает в один блок, а конец в другой, то для чтения надо будет получить два таких блока

Представим, что нет выравнивания и размер структуры занимает 3 байта

```go
type data struct {
    aaa int16
    bbb bool
}

func main() {
    var values []data
    for value := range values {
        _ = value
    }
}
```



Terminal: question

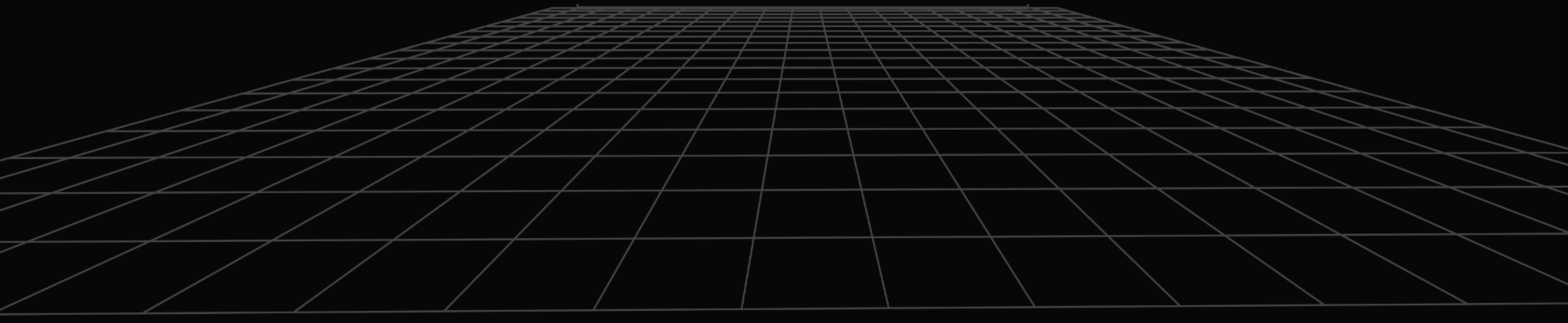
КАКОЕ БУДЕТ ВЫРАВНИВАНИЕ У СТРУКТУРЫ, СОСТОЯЩИХ ИЗ INT64 ПОЛЕЙ НА 32 БИТНОЙ МАШИНЕ?

Screenshot

type	alignment guarantee
bool, uint8, int8	1
uint16, int16	2
uint32, int32	4
float32, complex64	4
arrays	depend on element types
structs	depend on field types
other types	size of a native word

FAQ

Выравнивание структур



ПЕРЕРЫВ 5 МИНУТ

ТОНКОСТИ СТРУКТУР

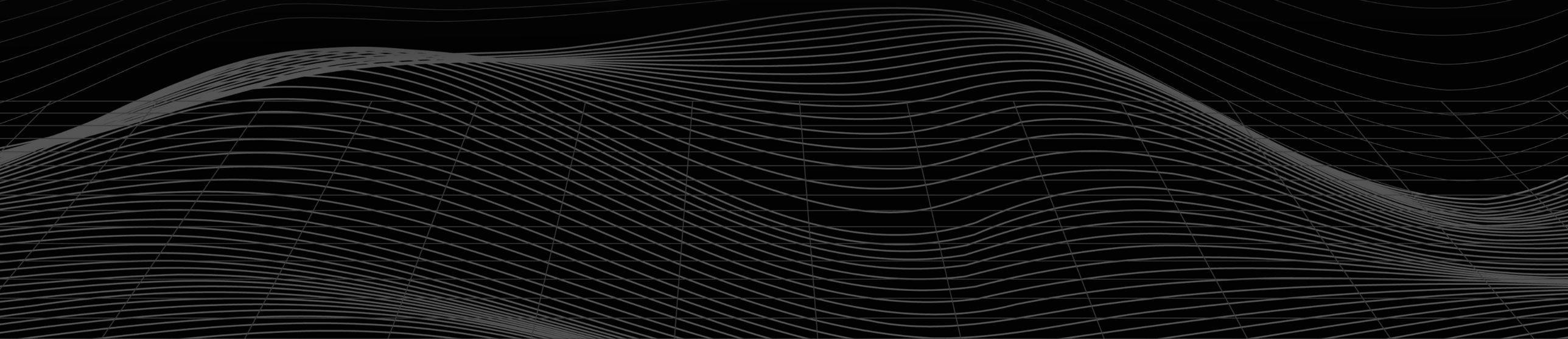
EXAMPLE

Empty struct size

РАЗМЕР ПУСТОЙ СТРУКТУРЫ РАВЕН НУЛЮ!

EXAMPLE

Empty structs



EXAMPLE

Final zero field

ALL THE FIELDS OF AN ADDRESSABLE STRUCT VALUE CAN BE TAKEN ADDRESSES.

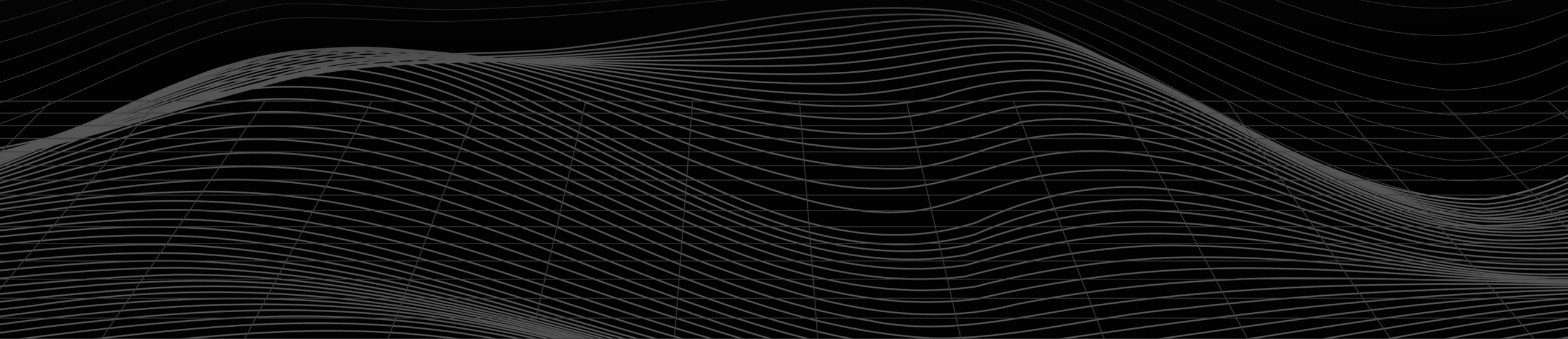
If the size of the final field in a non-zero-sized struct value is zero, then taking the address of the final field
in the struct value will return an address which is beyond the allocated memory block for the struct value. The returned address may point to another allocated memory block which closely follows the one allocated for the non-zero-sized struct value. As long as the returned address is stored in an active pointer value, the other allocated memory block will not get garbage collected, which may cause memory leaking.

To avoid these kinds of memory leak problems, the standard Go compiler will ensure that taking the address of the final field in a non-zero-sized struct will never return an address which is beyond the allocated memory block for the struct.

The standard Go compiler implements this by padding some bytes after the final zero-sized field when needed. If the types of all fields in a struct type are zero-sized (so the struct is also a zero-sized type), then there is no need to pad bytes in the struct, for the standard Go compiler treats zero-sized memory blocks specially.

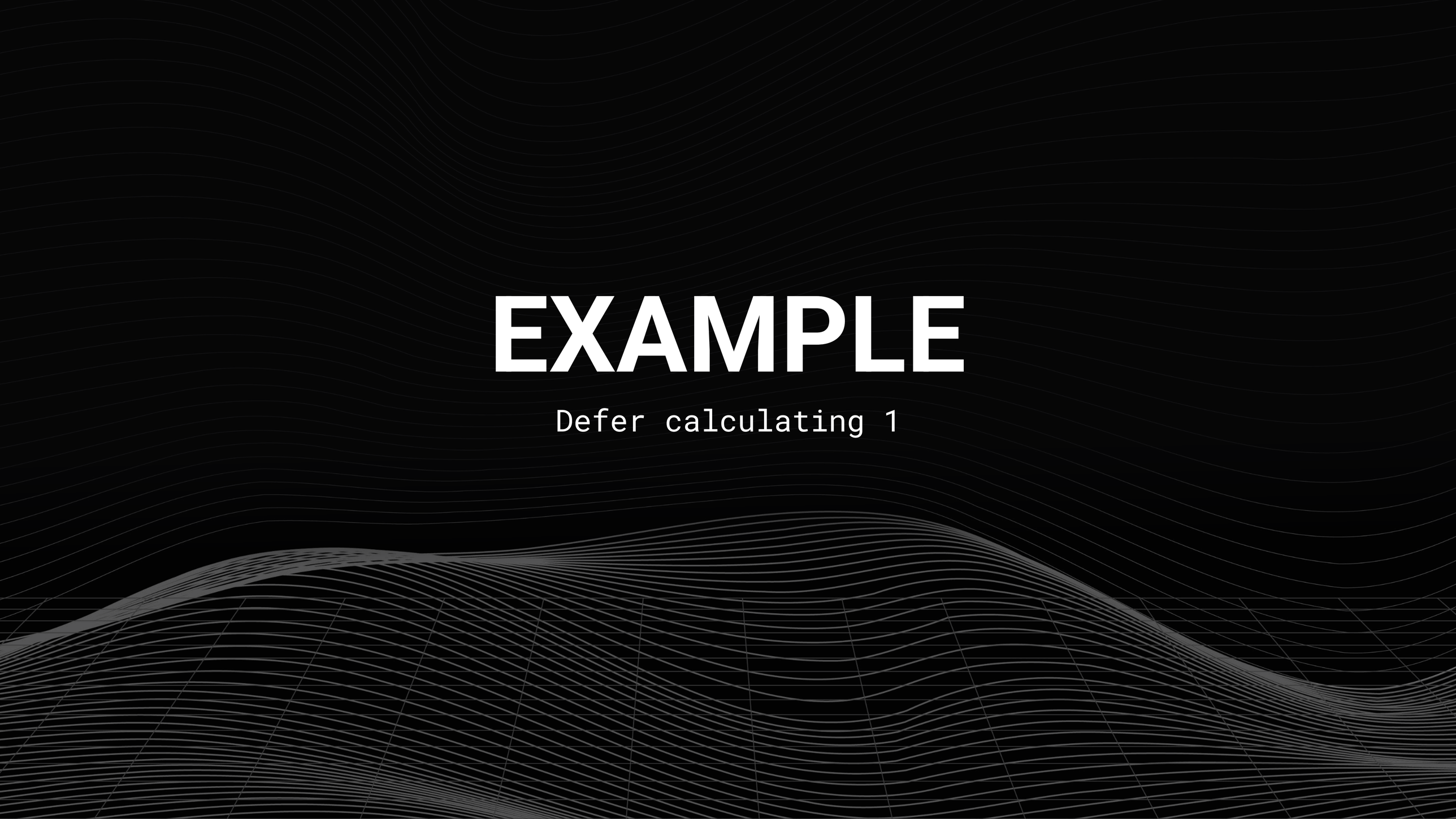
EXAMPLE

Empty structs internal



EXAMPLE

Defer calculating 1



EXAMPLE

Defer calculating 2

EXAMPLE

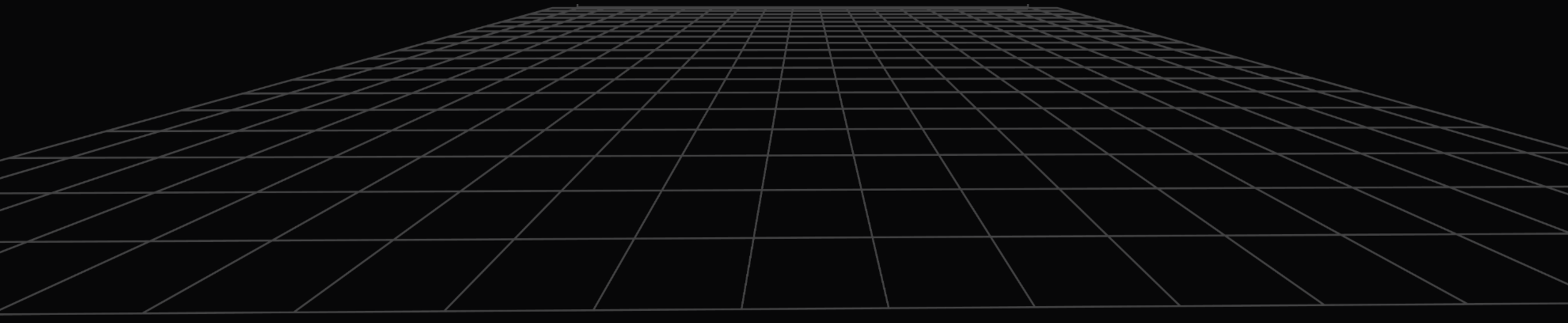
Comparison benchmark

EXAMPLE

Check struct size

FAQ

Тонкости структур



ПАТТЕРНЫ

Terminal: deep_golang

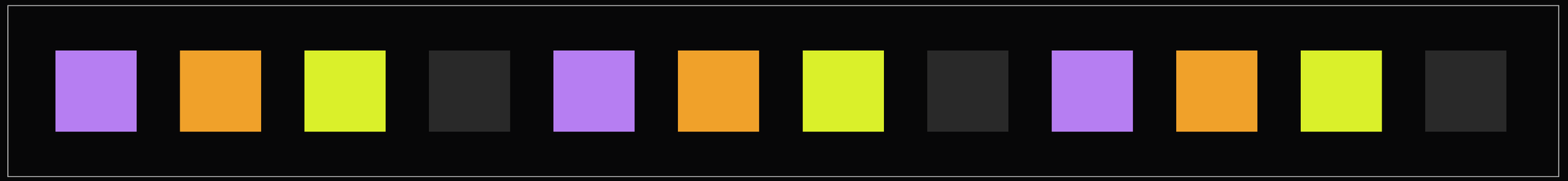
DOD (DATA ORIENTED DESIGN)

это способ оперировать данными
в cache friendly манере

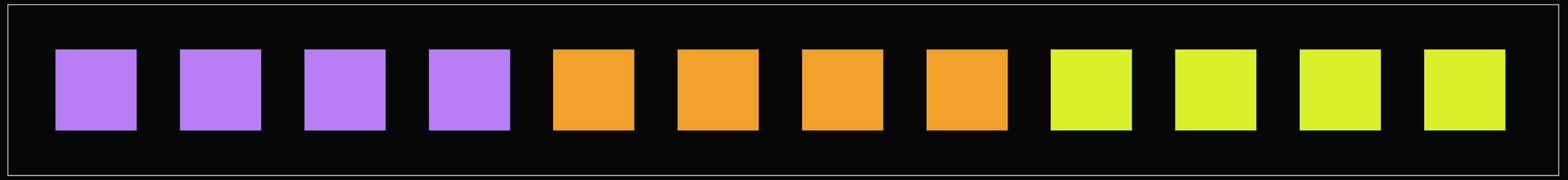
EXAMPLE

DOD

Object Oriented Design



Data Oriented Design



Closer

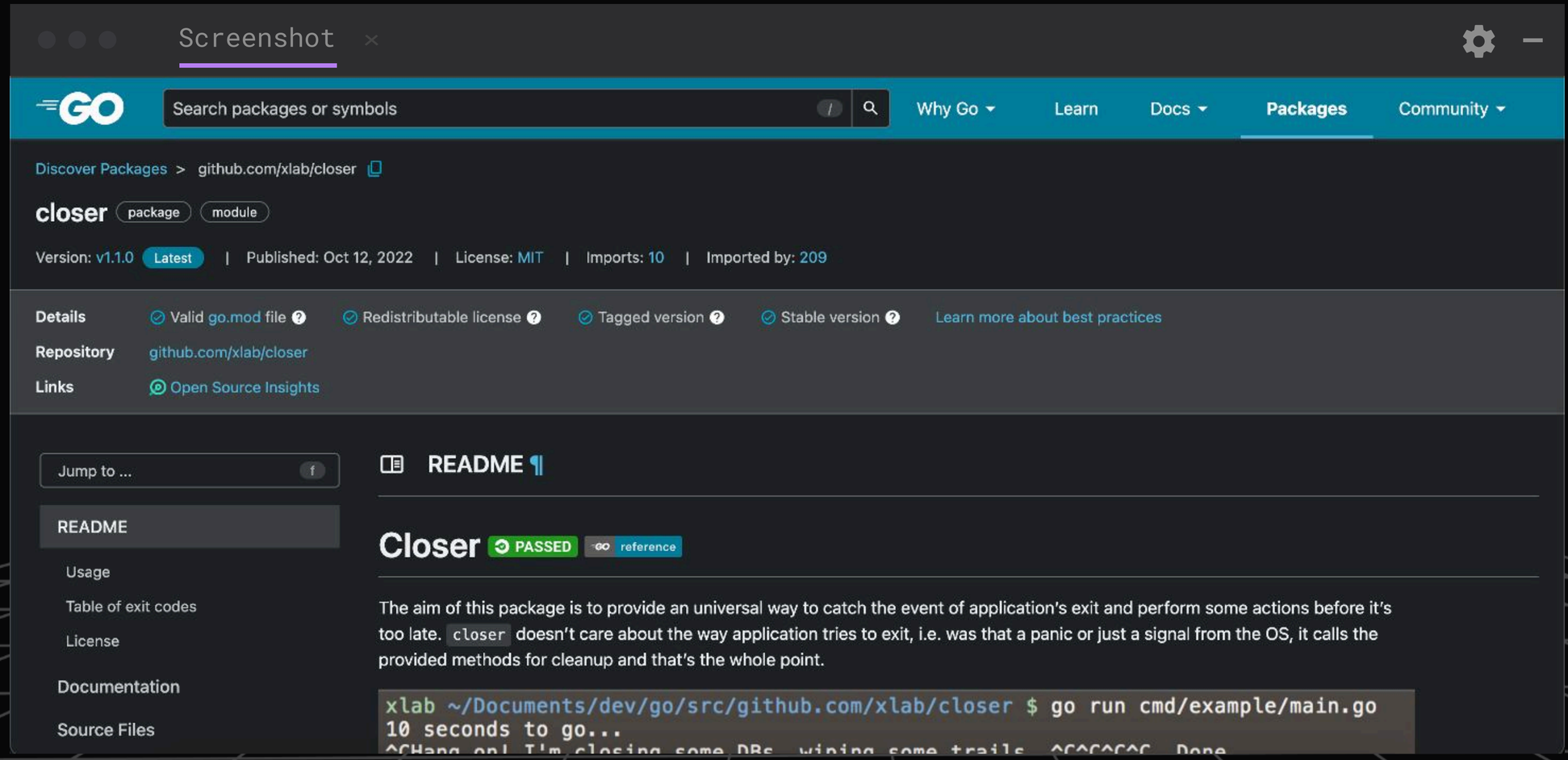
Очищать все ресурсы в одном месте программы не всегда удобно или вообще возможно сделать, не ухудшив архитектуру приложения

Для решения этой проблемы существует отдельный паттерн. Пусть каждый компонент, заинтересованный в том, чтобы ему сообщили, когда необходимо очистить ресурсы и прекратить работу, сообщит об этом отдельному компоненту, зарегистрировав у него обработчик

EXAMPLE

Closer

Можно не писать свое,
а использовать готовую реализацию



Terminal: question

КАК В GO ПРИНЯТО КОНСТРУИРОВАТЬ ОБЪЕКТЫ С ОПЦИОНАЛЬНЫМИ АРГУМЕНТАМИ?

EXAMPLE

Optional parameters

EXAMPLE

Functional options

Functional Options

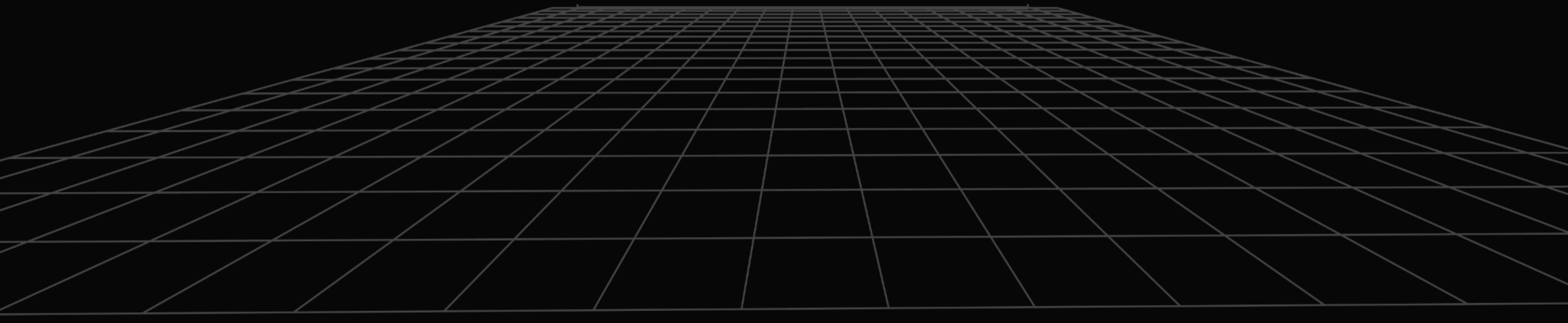
- Не надо рефакторить имеющиеся вызовы в случае изменений, они останутся без изменений
- Вызовы функций менее громоздкие, когда весь набор опций требуется очень редко, либо никогда
- Параметры становятся именованными и их становится проще читать

EXAMPLE

Configurable object

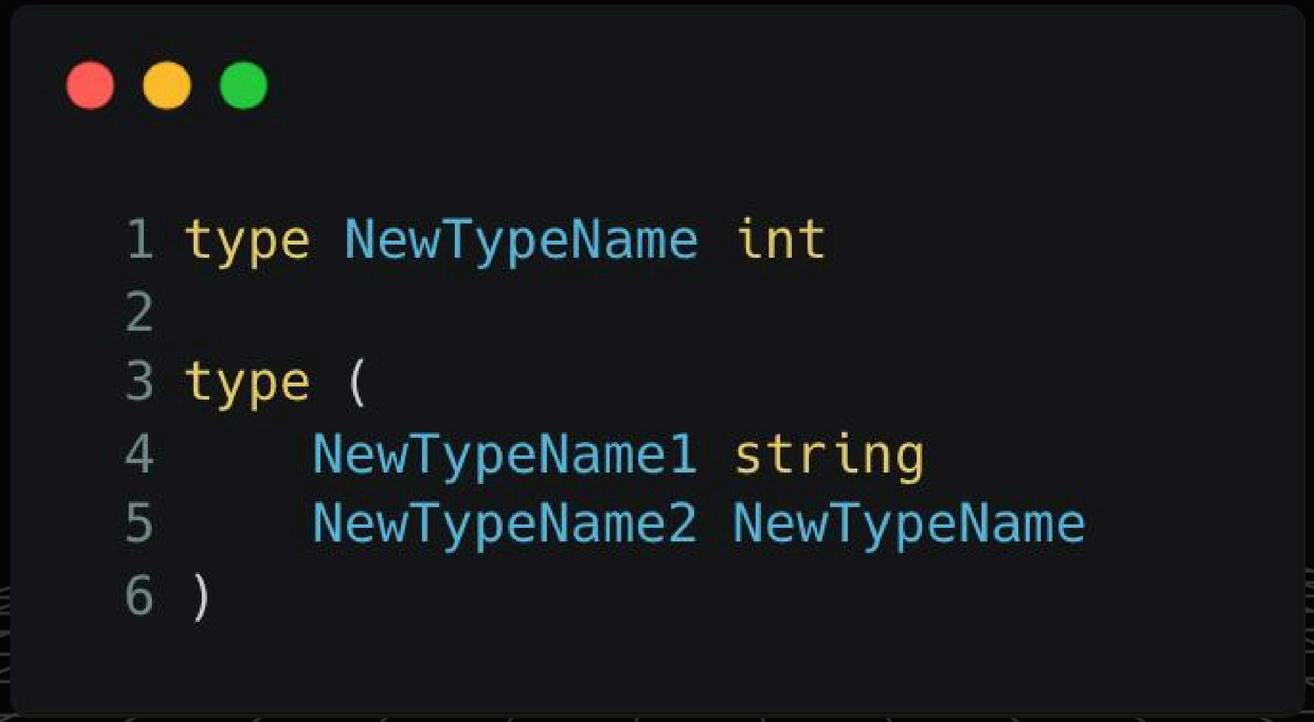
FAQ

Паттерны



ПСЕВДОНИМЫ И ОПРЕДЕЛЕНИЯ ТИПОВ

TYPE DEFENITION

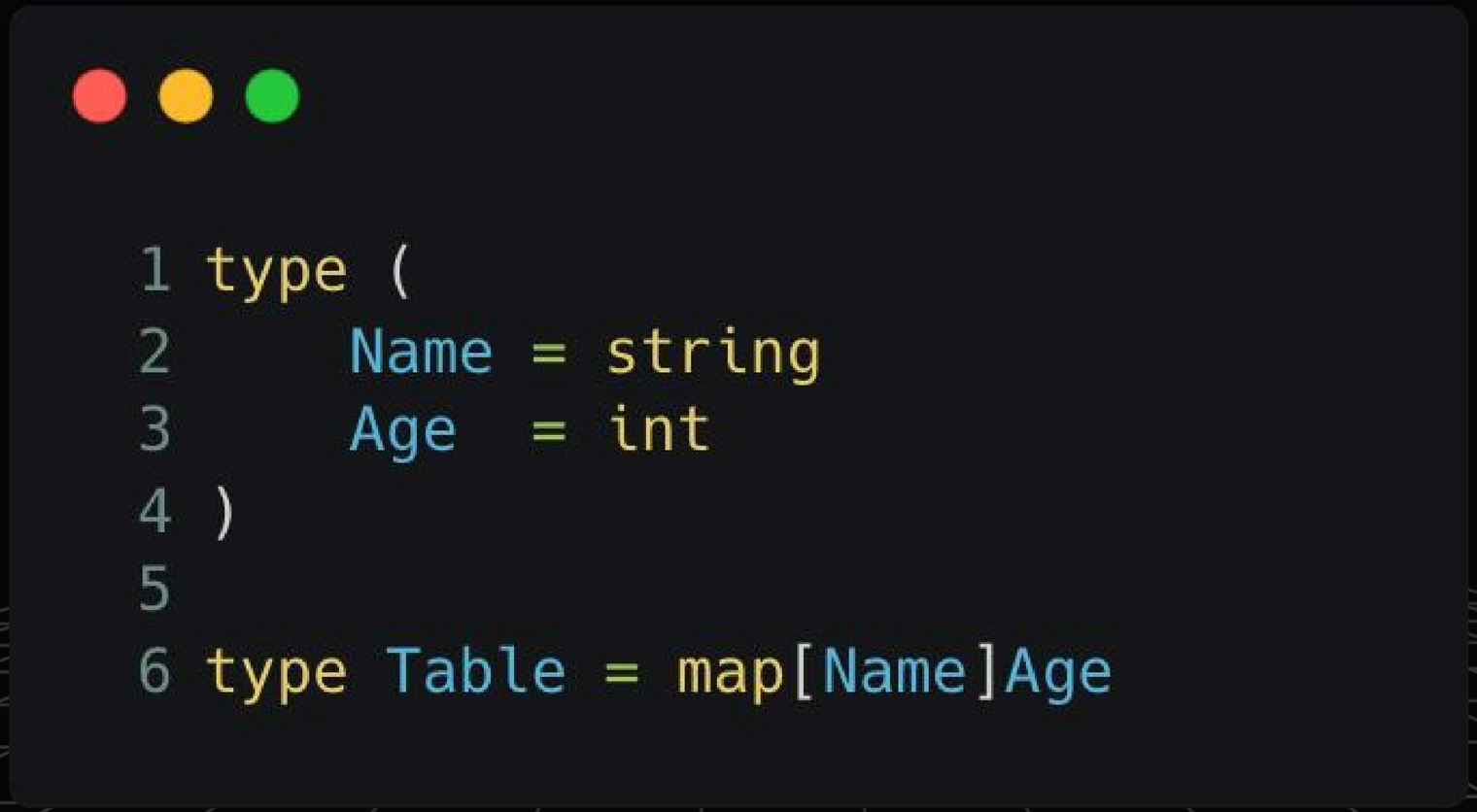


```
type NewTypeName int

type (
    NewTypeName1 string
    NewTypeName2 NewTypeName
)
```

Новый определенный тип и соответствующий ему исходный тип в определениях типов — это два разных типа. Новый определенный тип и исходный тип будут иметь один и тот же базовый тип (underlying type)

TYPE ALIAS



```
type (
    Name = string
    Age  = int
)

type Table = map[Name]Age
```

- **Type alias** – не создает новый тип, просто псевдоним типа
- **Type definition** – создает новый тип

EXAMPLE

Type alias and definition

При присваивании:

- **type alias** — ничего не нужно делать
- **type definition** — нужно кастить, потому что это разные типы данных

EXAMPLE

Type alias and definition with method

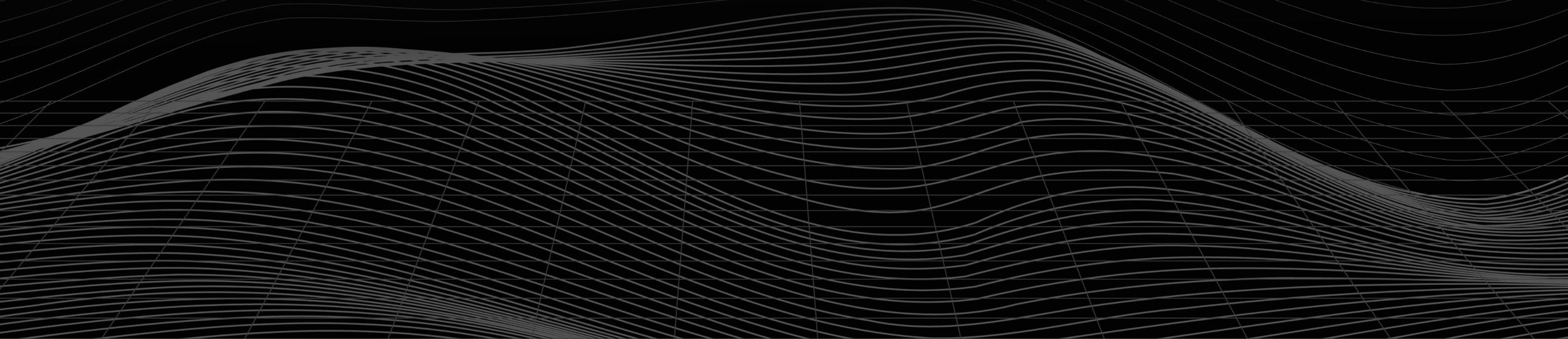
МОЖНО ОБЪЯВИТЬ

МЕТОД ДЛЯ ТИПА T, ЕСЛИ:

- тип T не является базовым типом данных и его алиасом
- тип T находится в том же пакете, где и метод
- тип T не должен быть указателем
- тип T не должен быть интерфейсом

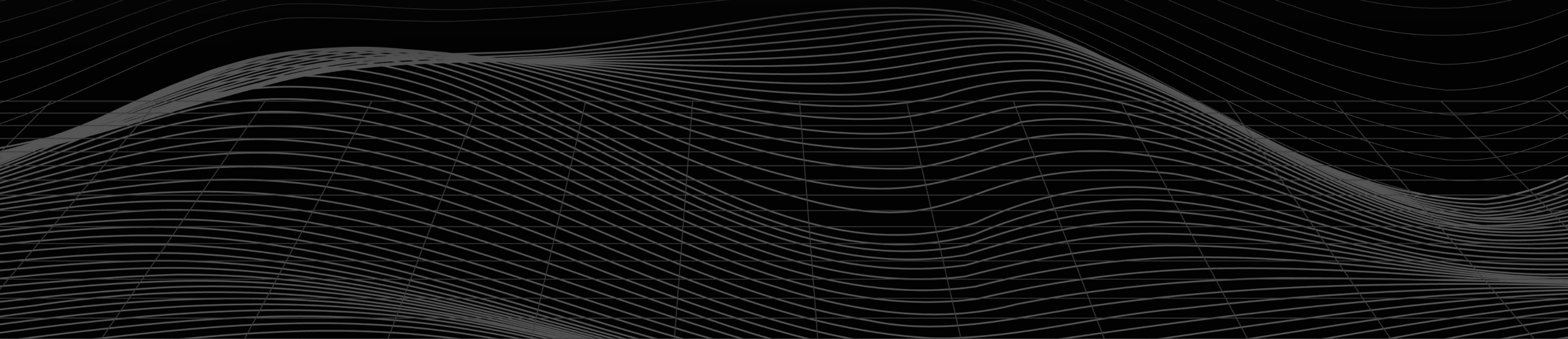
EXAMPLE

Methods with types



EXAMPLE

Methods for basic types

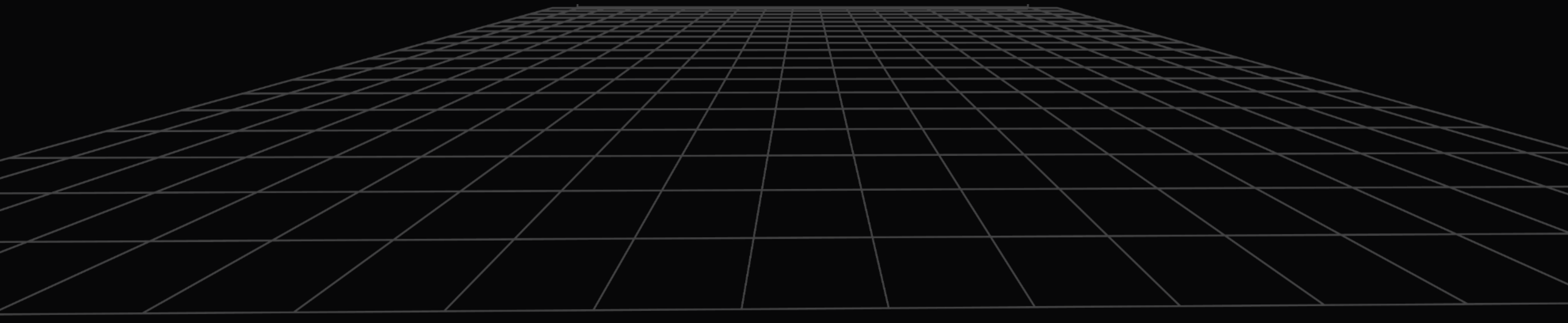


EXAMPLE

Unsafe cast

FAQ

Псевдонимы и определения типов



ПОЖАЛУЙСТА, ЗАПОЛНИ ОПРОС О ЗАНЯТИИ

Ссылка в чате и в группе участников