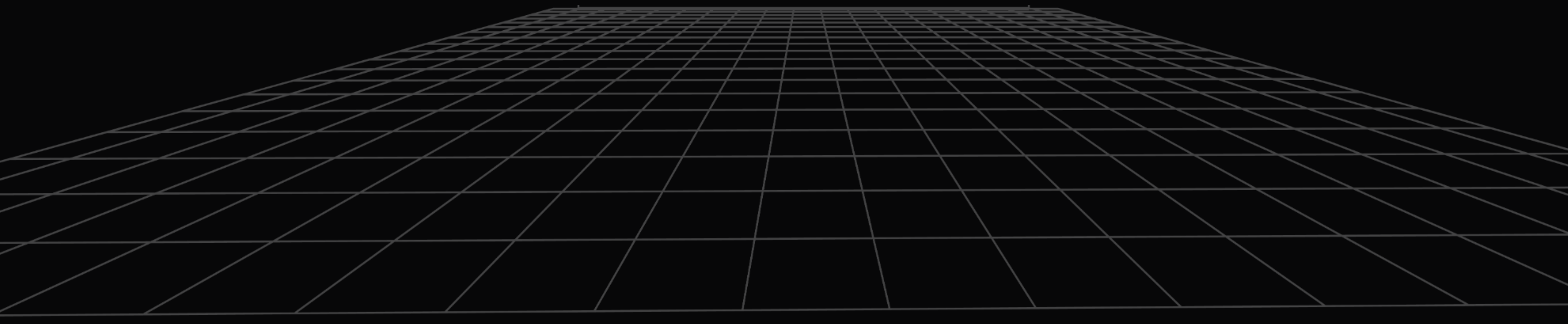


Глубокий Go

ТИПЫ ДАННЫХ

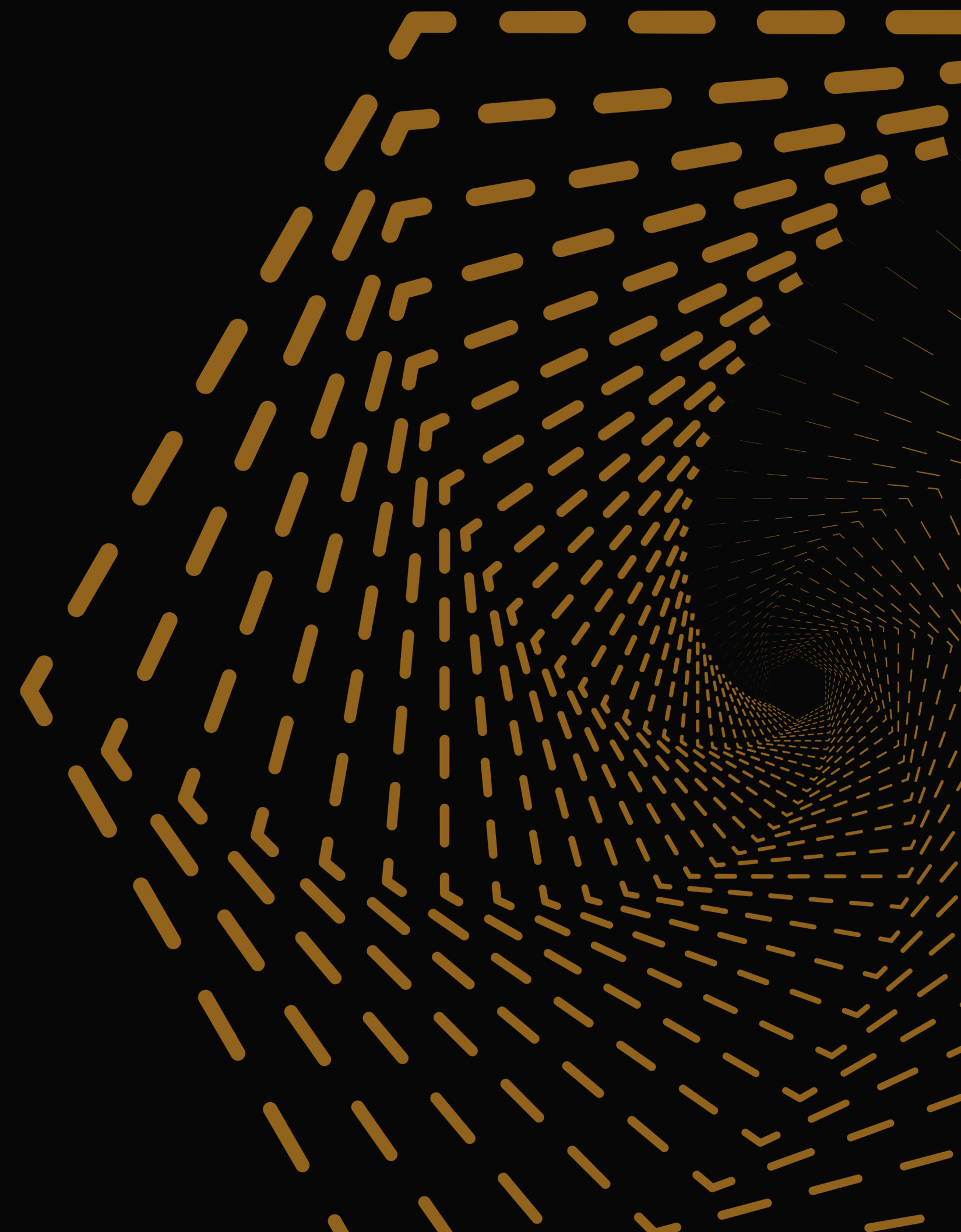


ПРОВЕРЬ ЗАПИСЬ



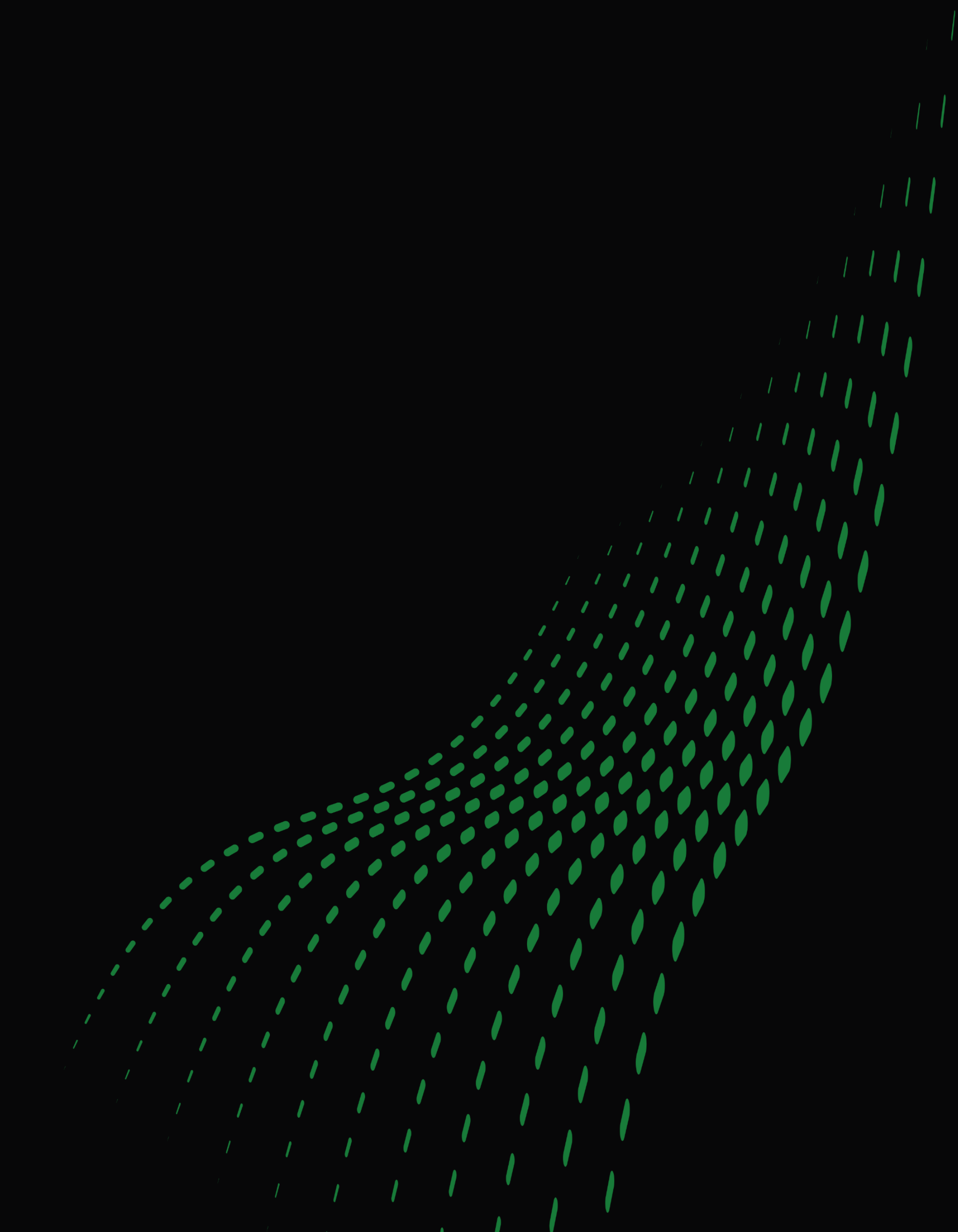
ПРАВИЛА ЗАНЯТИЯ

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. Типы данных
2. Целочисленные типы данных
3. Указатели
4. Порядок следования байтов
5. Битовые операции
6. Битовые операции на практике



ТИПЫ ДАННЫХ

ЧТО ТАКОЕ ТИП ДАННЫХ?

Terminal: deep_go × + ∨



ТИП ДАННЫХ

Множество значений и операций, которые
могут быть применимы над этими значениями

ПРИМИТИВНЫЕ ТИПЫ ДАННЫХ

Логические

- `bool`

Строковые

- `string`

Числовые

- `int8`, `uint8 (byte)`, `int16`, `uint16`, `int32`
`(rune)`, `uint32`, `int64`, `uint64`, `int`, `uint`, `uintptr`
- `float32`, `float64`
- `complex64`, `complex128`

СОСТАВНЫЕ ТИПЫ ДАННЫХ

1. Указатели

2. Структуры

3. Функции

4. Контейнеры

- массивы
- срезы
- словари

5. Каналы

6. Интерфейсы

КАЖДЫЙ ТИП ИМЕЕТ НУЛЕВОЕ ЗНАЧЕНИЕ (ZERO VALUE),

которое можно рассматривать как
значение типа по умолчанию

Data Types

```
graph TD; DT[Data Types] --> BT[Basic type]; DT --> AT[Aggregate type]; DT --> RT[Reference type]; DT --> IT[Interface type]; BT --> N[numbers]; BT --> S[strings]; BT --> B[boolean]; AT --> A[array]; AT --> ST[structs]; RT --> SL[slice]; RT --> M[map]; RT --> F[function]; RT --> CH[channel]; IT --> ;
```

Basic type

numbers

strings

boolean

Aggregate type

array

structs

Reference type

slice

map

function

channel

Interface type

Screenshot ×		⚙️ —	
Types whose values each is only hosted on one single memory block		Types whose values each may be hosted on multiple memory blocks	
Solo Direct Value Part		Direct Part → Underlying Part	
boolean types numeric types pointer types unsafe pointer types struct types array types		slice types map types channel types function types interface types string types	

**КАКОЙ РАЗМЕР У ОСНОВНЫХ
ТИПОВ ДАННЫХ?**

Бит – минимальная единица измерения информации, которая принимает значение «ноль» или «один»

Байт (октет) – единица измерения информации, которая состоит из восьми бит или двух тетрад

Машинное слово – основная единица информации, которой оперирует процессор (размер машинного слова равен архитектуре процессора, например 32 или 64 битная архитектура)

Screenshot ×		⚙️	—
type	size in bytes		
-----	-----		
uint8, int8	1		
uint16, int16	2		
uint32, int32, float32	4		
uint64, int64	8		
float64, complex64	8		
complex128	16		
uint, int	implementation-specific, generally 4 on 32-bit architectures, and 8 on 64-bit architectures.		
uintptr	implementation-specific, large enough to store the uninterpreted bits of a pointer value.		

**ПОЧЕМУ РАЗМЕР
ЛОГИЧЕСКОГО ТИПА ДАННЫХ
(BOOL) 1 БАЙТ, А НЕ 1 БИТ?**

Байтовая адресация

0xAACO

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0xAAC1

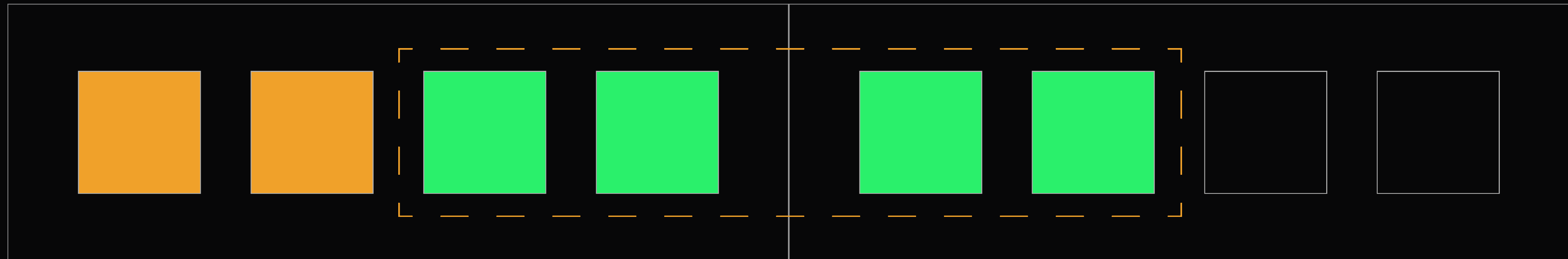
0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Словесная адресация

0xAACO

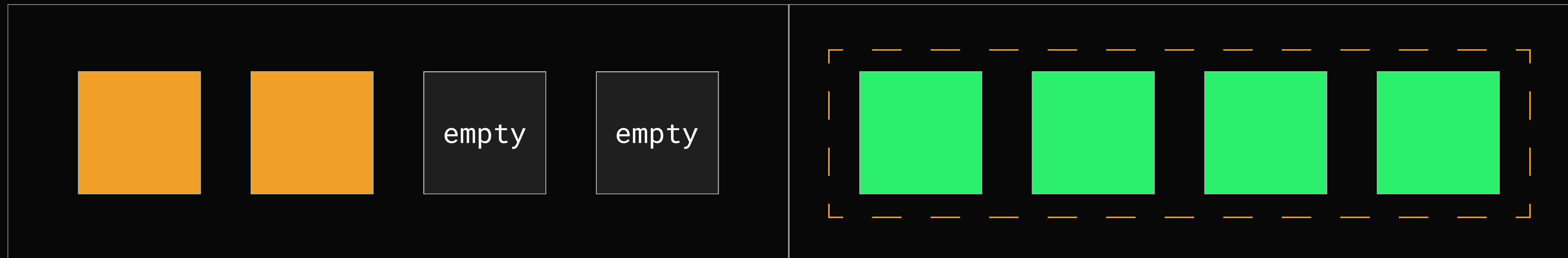
**НУЖНО ЛИ ВЫРАВНИВАТЬ
ТИПЫ ДАННЫХ В ПАМЯТИ?**

machine word



machine word

machine word

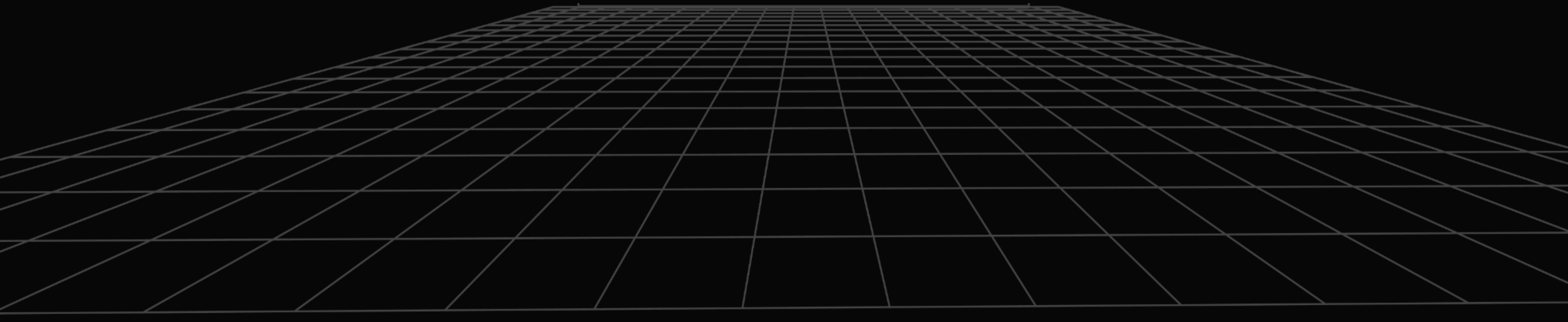


machine word

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

Типы данных



ЦЕЛОЧИСЛЕННЫЕ ТИПЫ ДАННЫХ

Terminal: deep_go × + ∨



ПРЯМОЙ КОД ЧИСЛА

В десятичной системе счисления: 121

В двоичной системе счисления: 0 1 1 1 1 0 0 1

 $2^0 + 2^3 + 2^4 + 2^5 + 2^6 = 1 + 8 + 16 + 32 + 64 = 121$

МОЖНО ЗАКОДИРОВАТЬ $2^N - 1$ ЗНАЧЕНИЙ,

где N — количество бит

8 бит

- 00000000 = 0
- 11111111 = 255

16 бит

- 00000000 00000000 = 0
- 11111111 11111111 = 65535

РАЗРЯДЫ БИТОВ

Седьмой разряд (крайний левый) называется **старшим**,
а нулевой разряд (крайний правый) – **младшим**.

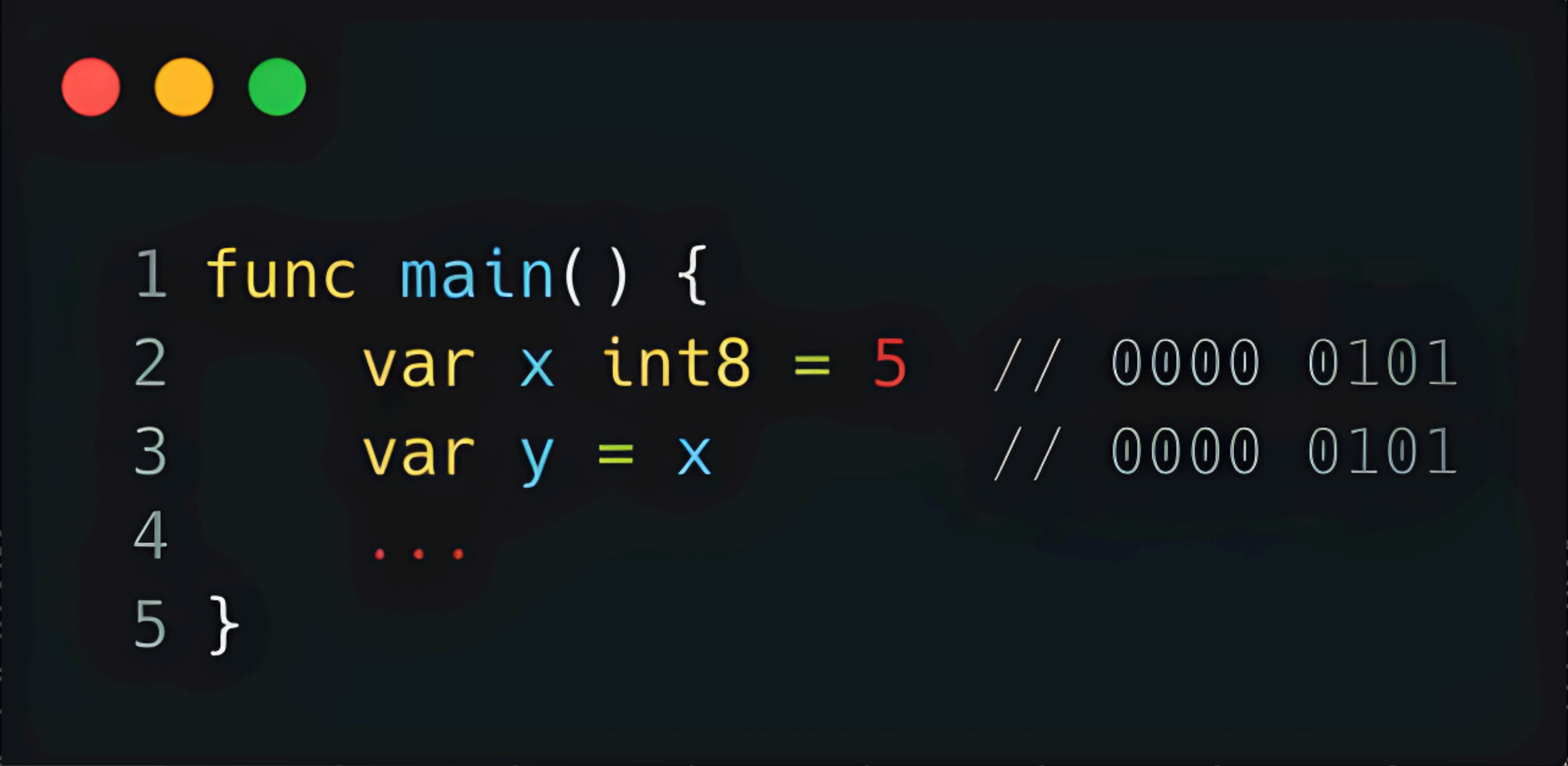
Наименьшему значению соответствует комбинация нулей, а наибольшему значению соответствует комбинация единиц

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

← **Старшие разряды**

Младшие разряды →

КОПИРОВАНИЕ ПЕРЕМЕННЫХ



```
1 func main() {  
2     var x int8 = 5 // 0000 0101  
3     var y = x      // 0000 0101  
4     ...  
5 }
```

КАК БЫТЬ С ОТРИЦАТЕЛЬНЫМИ
ЧИСЛАМИ?

0 – положительное

1 – отрицательное

[illegible]

Можно закодировать $2^N - 1$ (7)
положительных и $2^N - 1$ (7)
отрицательных чисел

Получилось два нуля –
положительный
и отрицательный, а также
-7 получился больше -1

Знак	Число			
0	1	1	1	7
0	1	1	0	6
0	1	0	1	5
0	1	0	0	4
0	0	1	1	3
0	0	1	0	2
0	0	0	1	1
0	0	0	0	0
1	0	0	0	0
1	0	0	1	-1
1	0	1	0	-2
1	0	1	1	-3
1	1	0	0	-4
1	1	0	1	-5
1	1	1	0	-6
1	1	1	1	-7

Не работает сложение

+	0	0	1	0	2
	1	0	1	0	-2
=	1	1	1	0	-6

Можно использовать
обратный код, инвертируя
все биты чисел

Знак	Число			
0	1	1	1	7
0	1	1	0	6
0	1	0	1	5
0	1	0	0	4
0	0	1	1	3
0	0	1	0	2
0	0	0	1	1
0	0	0	0	0
1	1	1	1	0
1	1	1	0	-1
1	1	0	1	-2
1	1	0	0	-3
1	0	1	1	-4
1	0	1	0	-5
1	0	0	1	-6
1	0	0	0	-7

Так работает

+

0

0

1

0

2

1

1

0

1

-2

=

1

1

1

1

-0

Так не работает

+

0

0

1

0

2

1

1

1

0

-1

=

0

0

0

0

0

Можно использовать
дополнительный код,
добавив 1 к обратному
коду числа

Знак	Число			
0	1	1	1	7
0	1	1	0	6
0	1	0	1	5
0	1	0	0	4
0	0	1	1	3
0	0	1	0	2
0	0	0	1	1
0	0	0	0	0
0	0	0	0	0
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7

Исчез **-0**, вместо него
теперь можно добавить
еще одно число

Можно закодировать $2^N-1 - 1$ (7)
положительных и $2^N-1(8)$
отрицательных чисел

	Знак	Число			
	0	1	1	1	7
	0	1	1	0	6
	0	1	0	1	5
	0	1	0	0	4
	0	0	1	1	3
	0	0	1	0	2
	0	0	0	1	1
	0	0	0	0	0
>	1	1	1	1	-1
>	1	1	1	0	-2
	1	1	0	1	-3
	1	1	0	0	-4
	1	0	1	1	-5
	1	0	1	0	-6
	1	0	0	1	-7
	1	0	0	0	-8

Так работает

+	0	0	1	0	2
	1	1	1	0	-2
	=	0	0	0	0

Так тоже работает

+	0	0	1	0	2
	1	1	1	1	-1
	=	1	1	1	-1

Оставшиеся биты у положительных чисел заполняются нулями, а у отрицательных единицами

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

 = 4

1	1	1	1	1	1	0	0
---	---	---	---	---	---	---	---

 = -4

0	0	1	0
---	---	---	---

1	1	1	0
---	---	---	---

Знаковые числа компьютер может сравнивать
лексикографически по первым битам числа
(получилась некая оптимизация)

**ЧТО ПРОИЗОЙДЕТ, ЕСЛИ МЫ В ЧИСЛО
ЗАПИШЕМ БОЛЬШЕ ИНФОРМАЦИИ,
чем оно в себя может вместить?**

+
=

0	1	1	1
---	---	---	---

7

0	0	1	1
---	---	---	---

3

1	0	1	0
---	---	---	---

-6

Знак

Число

0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0

7

6

5

4

3

2

1

0

-1

-2

-3

-4

-5

-6

-7

-8

+
=

1	0	0	1
---	---	---	---

-7

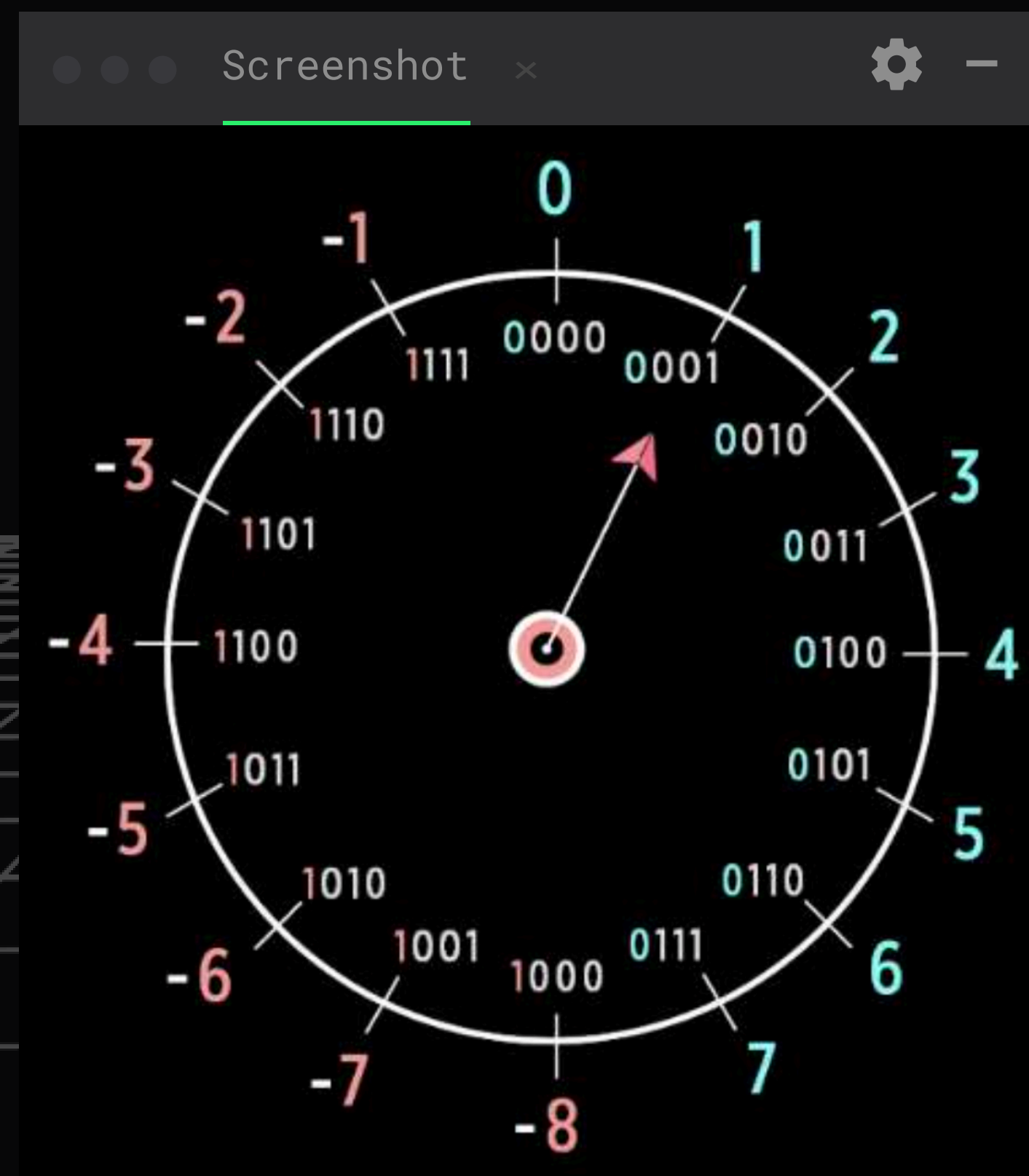
1	1	0	1
---	---	---	---

-3

0	1	1	0
---	---	---	---

6

НА КРУГЕ ВСЕ СТАНОВИТСЯ ПОНЯТНЕЕ



EXAMPLE

Int overflow

Неудачный запуск ракеты Ariane 5 в 1996 году
произошел из-за переполнения, возникшего в результате
преобразования 64-битного числа с плавающей точкой
в 16-битное целое число со знаком

КАК ОБНАРУЖИТЬ ПЕРЕПОЛНЕНИЕ?

EXAMPLE

Overflow detection

ЦЕЛОЧИСЛЕННЫЕ ЛИТЕРАЛЫ

- **Двоичные** – используется префикс `0b` или `0B` (например `0b100`)
- **Восьмеричные** – используется префикс `0` или `0o` (например `010`)
- **Шестнадцатеричные** – используется префикс `0x` или `0X` (например `0xF`)
- **Мнимые** – используется суффикс `i` (например `3i`)

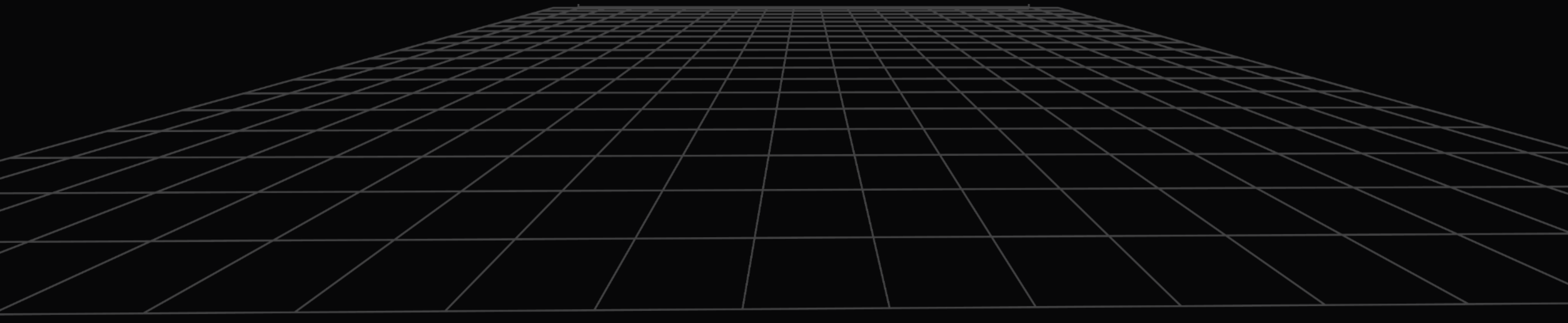
Еще можно использовать `_` в качестве разделителя для удобства чтения, например `1_000_000`

EXAMPLE

Octal system

FAQ

Целочисленные типы данных



УКАЗАТЕЛИ

Terminal: deep_go × + ∨



УКАЗАТЕЛЬ

Переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения – нулевого адреса (**nil**)

КАК ПОЛУЧИТЬ АДРЕС
ПЕРЕМЕННОЙ?



```
1 // 1 way
2 pointer := new(int)
3
4 // 2 way
5 var value int
6 pointer := &value
7
8 // 3 way - only for structs
9 pointer := &int{} // compilation error
```

EXAMPLE

Pointer

value [0xAA00]



pointer [0xAA04]



pointer = &value

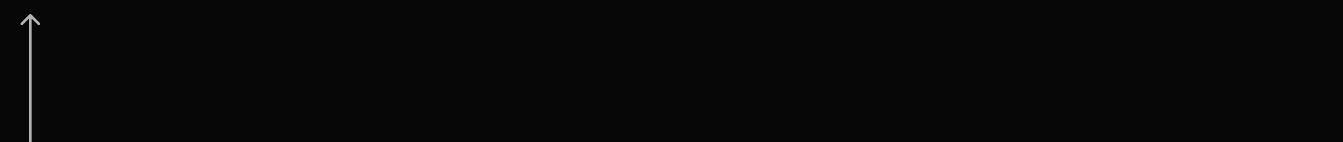
value [0xAA00]



pointer [0xAA04]



***pointer = 500**



Разыменование

**ЗАЧЕМ НУЖНЫ
УКАЗАТЕЛИ?**

EXAMPLE

Pointer with function

А также, чтобы вместо больших объектов передавать
только адреса на них и чтобы строить структуры
данных, основанные на указателях (*такие как связанные
списки, деревья и так далее*)

EXAMPLE

Pointer to pointer incorrect

main

value1 [0xAA00]

100

pointer [0xAA04]

0xAA00

process

value2 [0xAAB0]

200

temp [0xAAB4]

0xAAB0



УКАЗАТЕЛИ НА УКАЗАТЕЛИ

value [0xAA00]



pointer1 [0xAA04]



pointer1 = &value

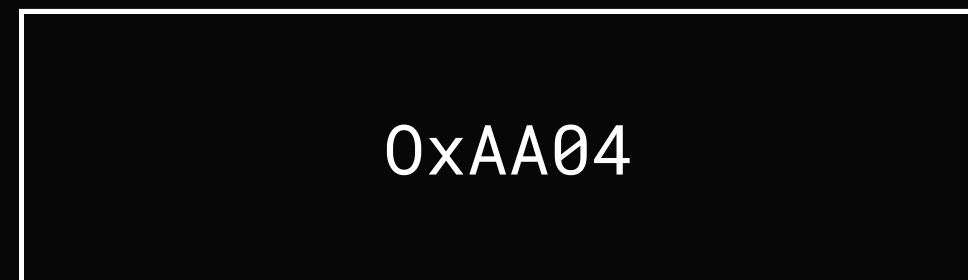
value [0xAA00]



pointer1 [0xAA04]



pointer2 [0xAA08]



pointer2 = &pointer1

value [0xAA00]



pointer1 [0xAA04]



pointer2 [0xAA08]



****pointer2 = 200**



Разыменование

Разыменование

EXAMPLE

Pointer to pointer correct

main

value1 [0xAA00]

100

pointer [0xAA04]

0xAAB0

process

value2 [0xAAB0]

200

temp [0xAAB4]

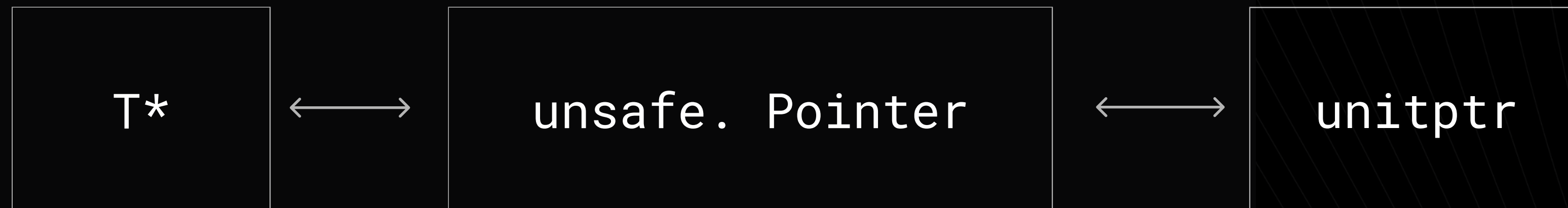
0xAA04

В Go не поддерживается арифметика указателей,
но зато можно использовать пакет **unsafe**



```
1 func main() {  
2     number := 200  
3     pointer := &number  
4     pointer = pointer + 2 // compilation error  
5 }
```


POINTER



Pointer represents a pointer to an arbitrary type. There are four special operations available for type `Pointer` that are not available for other types:

- A pointer value of any type can be converted to a `Pointer`.
- A `Pointer` can be converted to a pointer value of any type.
- A `uintptr` can be converted to a `Pointer`.
- A `Pointer` can be converted to a `uintptr`.

Terminal: deep_go × + ∨



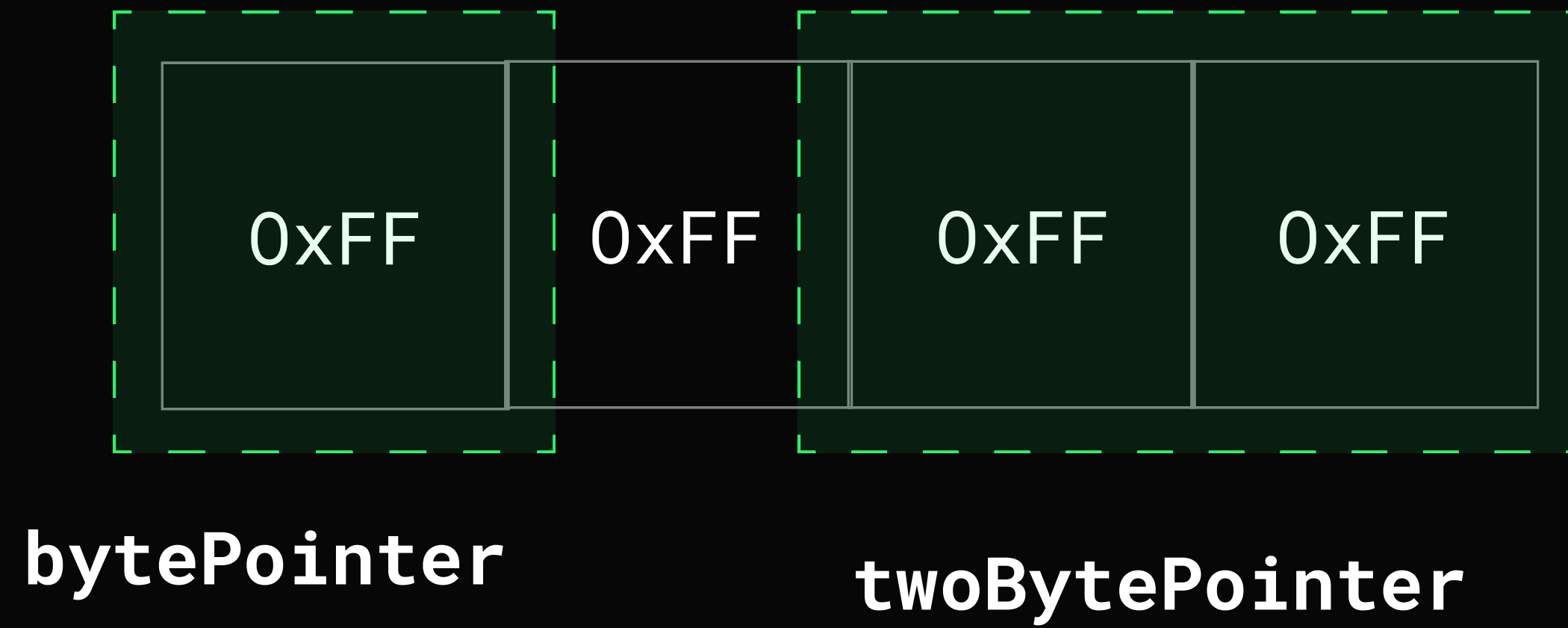
UINTPTR

This is an unsigned integer type which is large enough to hold any pointer address. Therefore its size is platform dependent. It is just an integer representation of an address.

EXAMPLE

Unsafe pointer

value



EXAMPLE

Unsafe size of

We know that, for some cases, the unsafe mechanism can help us write more efficient Go code. However, it is very easy to introduce some subtle bugs which have very low possibilities to produce when using the unsafe mechanism. A program with these bugs may run well for a long time, but suddenly behave abnormally and even crash at a later time. Such bugs are very hard to detect and debug

Another big risk of using unsafe pointers comes from the fact that the unsafe mechanism is not protected by the Go compatibility guideline - **code depending on unsafe pointers works today could break since a later Go version**

Параметр компилятора динамического анализа **-gcflags=all=-d=checkptr** поддерживается начиная с **Go Toolchain 1.14**. При использовании этой опции некоторые (но не все) неправильные использования указателей будут обнаружены во время выполнения (как только такое неправильное использование будет обнаружено, возникнет паника)

EXAMPLE

Uintptr gc

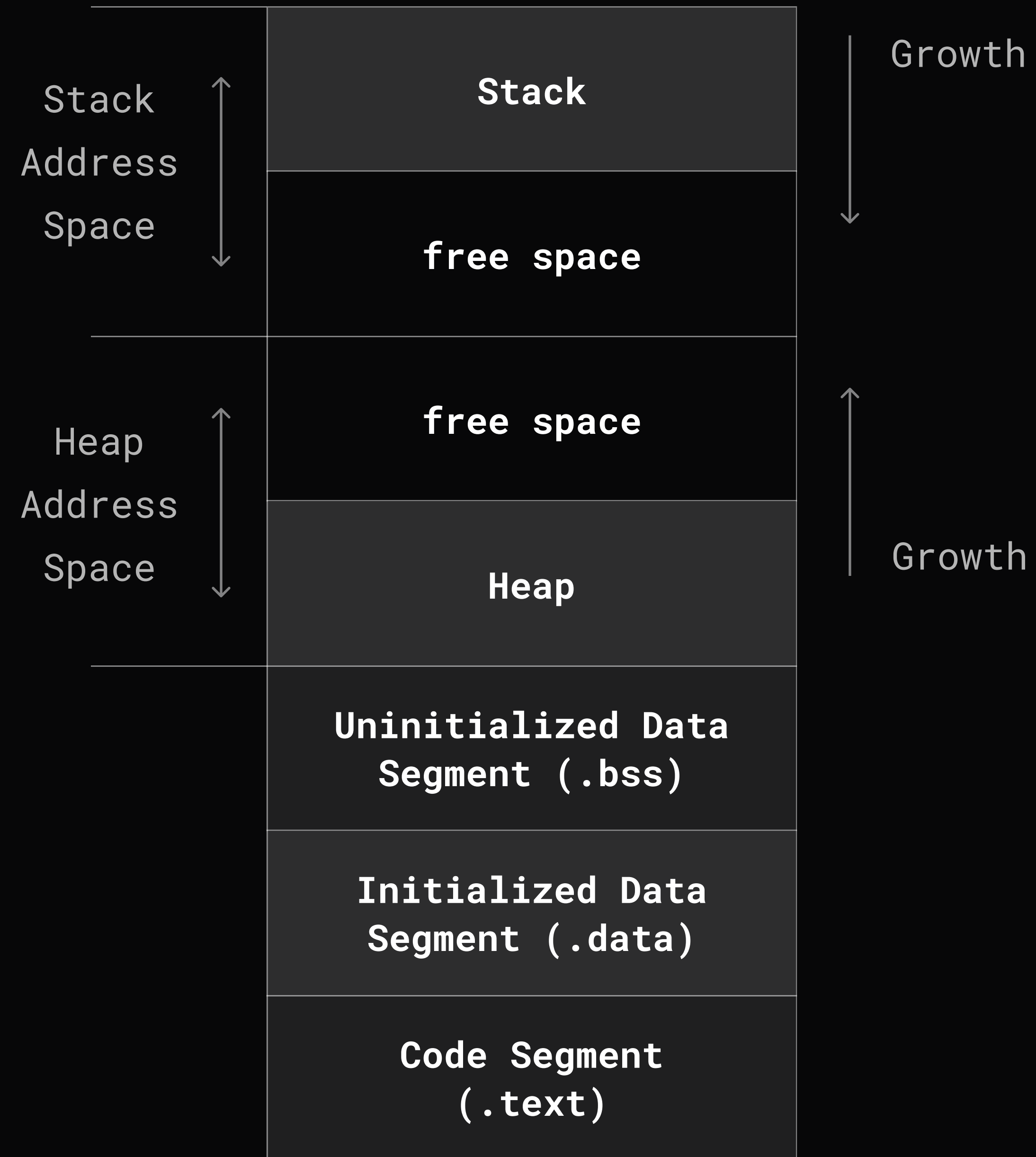
EXAMPLE

Uintptr stack grow

Each of non-nil safe and unsafe pointers references another value. However uintptr values don't reference any values, they are just plain integers, though often each of them stores an integer which can be used to represent a memory address.

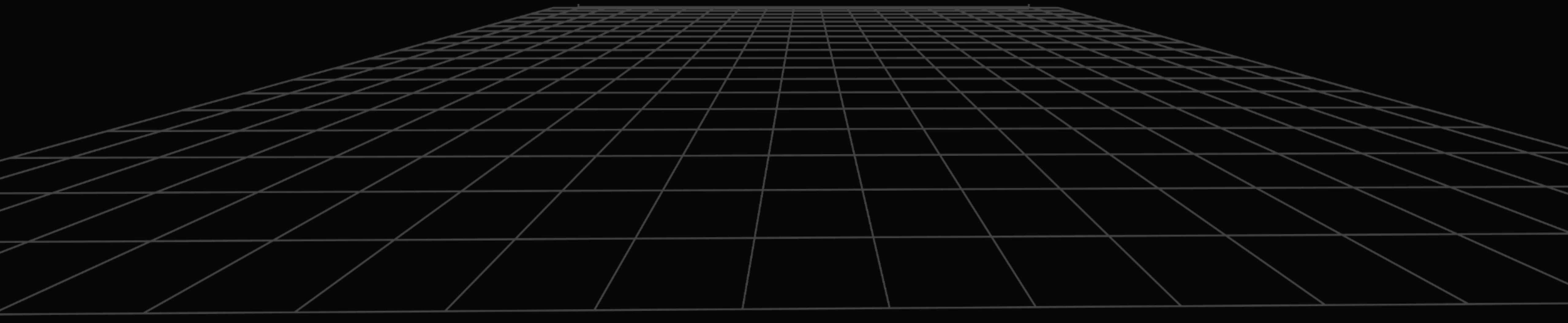
EXAMPLE

Unsafe analysis



FAQ

Указатели



ПЕРЕРЫВ 5 МИНУТ

ПОРЯДОК СЛЕДОВАНИЯ БАЙТОВ

Разряды байтов

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
------	------	------	------	------	------	------	------

← Старшие байты

Младшие байты →

EXAMPLE

Endianness

Terminal: deep_go × + ∨



ENDIANNESS

В том случае, если число не может быть представлено одним байтом, имеет значение, в каком порядке байты записываются в памяти компьютера или передаются по линиям связи

BIG ENDIAN

Порядок от старшего к младшему – этот порядок подобен привычному порядку записи слева направо (*например, число сто двадцать три было бы записано при таком порядке как 123*).

Этот порядок является стандартным для протоколов TCP/IP – он используется в заголовках пакетов данных и во многих протоколах более высокого уровня, поэтому такой порядок часто называют **«сетевым порядком байтов»**

Terminal: deep_go × + ∨



LITTLE ENDIAN

Порядок от младшего к старшему – это обратный привычному порядку записи чисел (*например, число сто двадцать три было бы записано при таком порядке как 321*). Этот порядок записи принят в памяти персональных компьютеров с процессорами архитектуры x86, в связи с чем иногда его называют «интеловским порядком байтов»

Big Endian

0	0	0	1	0	0	1	1	1	0	1	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Little Endian

1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Порядок меняется только для байтов – биты
остаются в той же последовательности

Представьте, что вы отправляете письмо другу в другую страну, и в письме вы хотите сообщить ему о важной дате: 23 апреля 2023 года.

В России дату обычно записывают в формате день/месяц/год (23.04.2023), а в США – месяц/день/год (04.23.2023). Если ваш друг в США откроет письмо и увидит дату в вашем формате, он может неправильно понять, когда именно произойдет событие, особенно если день меньше 13

**ЗАЧЕМ ПРИДУМАЛИ РАЗНЫЕ
ПОРЯДКИ СЛЕДОВАНИЯ БАЙТ?**

**Современные процессоры x86 позволяют
работать с одно, двух, четырёх
и восьмибайтовыми операндами**

В таком порядке следования байтов очень удобно то, что
при увеличении размера (количества байтов) операнда,
значение его первого байта неизменно: **3210 → 3210 0000**.

При порядке от старшего к младшему значение изменилось
бы, например: **0123 → 0000 0123**

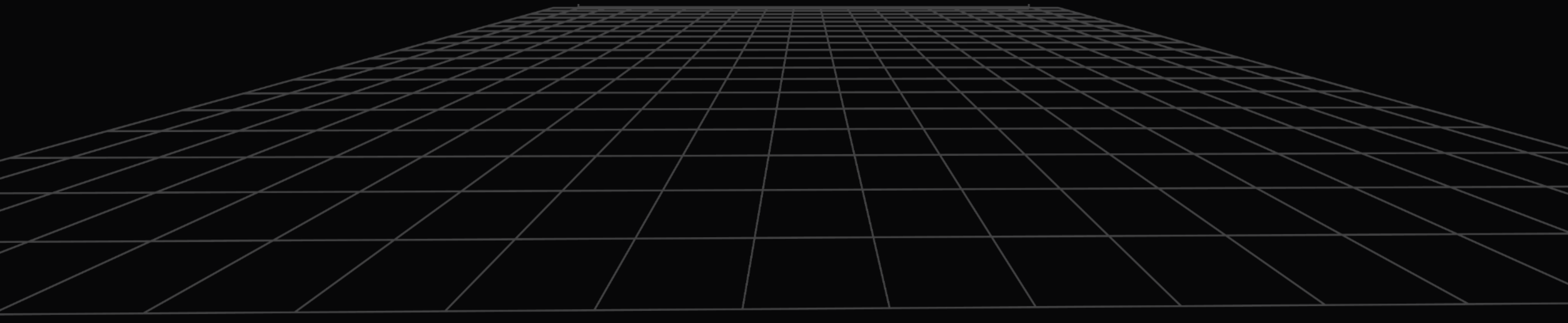
**КАК ПРОВЕРИТЬ ПОРЯДОК
СЛЕДОВАНИЯ БАЙТОВ?**

EXAMPLE

Endianness check

FAQ

Порядок следования байтов



БИТОВЫЕ ОПЕРАЦИИ

Битовое И



```
1 func main() {  
2     var x int8 = 3    // 0011  
3     var y int8 = 5    // 0101  
4     var result = x & y // 0001  
5 }
```

&

=

0	0	1	1
---	---	---	---

0	1	0	1
---	---	---	---

0	0	0	1
---	---	---	---

Битовое ИЛИ



```
1 func main() {  
2     var x int8 = 3    // 0011  
3     var y int8 = 5    // 0101  
4     var result = x | y // 0111  
5 }
```

|

=

0	0	1	1
0	1	0	1
0	1	1	1

Исключающее ИЛИ (XOR)



```
1 func main() {  
2     var x int8 = 3    // 0011  
3     var y int8 = 5    // 0101  
4     var result = x ^ y // 0110  
5 }
```

^

=

0	0	1	1
---	---	---	---

0	1	0	1
---	---	---	---

0	1	1	0
---	---	---	---

Битовое НЕ



```
1 func main() {  
2     var x uint8 = 3 // 0011  
3     var result = ^x // 1100  
4 }
```

^

0	0	1	1
1	1	0	0

EXAMPLE

Inversion

Инверсия для отрицательных и положительных
чисел работает по разному!

Битовые сдвиги

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

$1 \ll 3$

0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

$255 \ll 4$

1	1	1	1	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---

Битовые сдвиги

16 >> 3

0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---

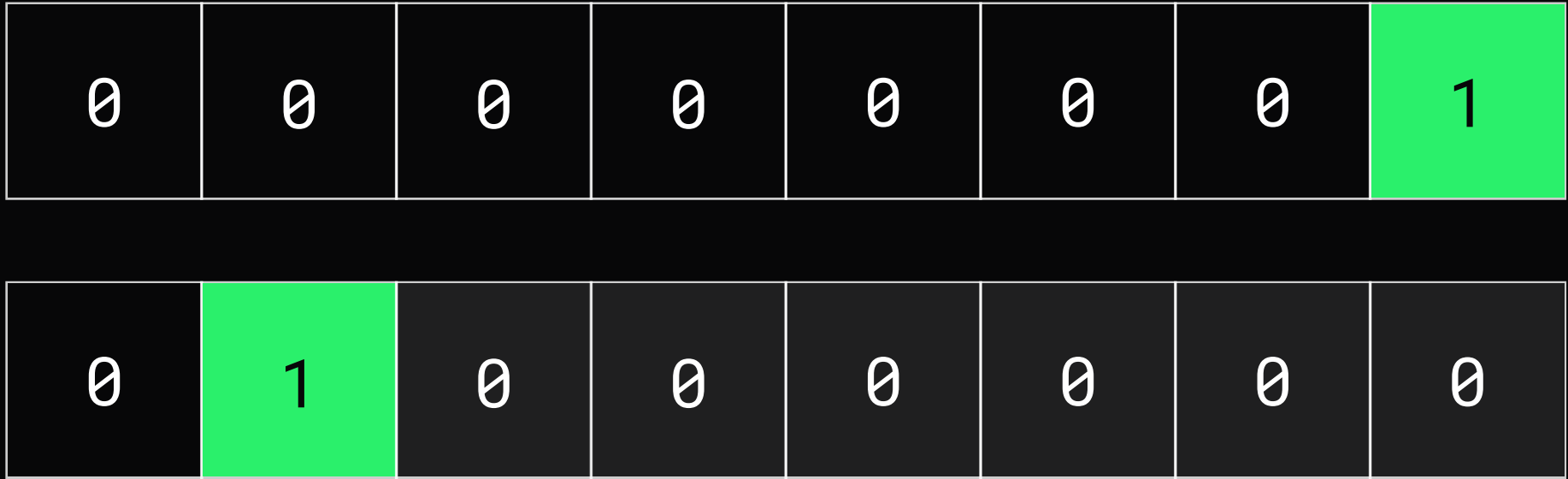
0	0	0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---

255 >> 4

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

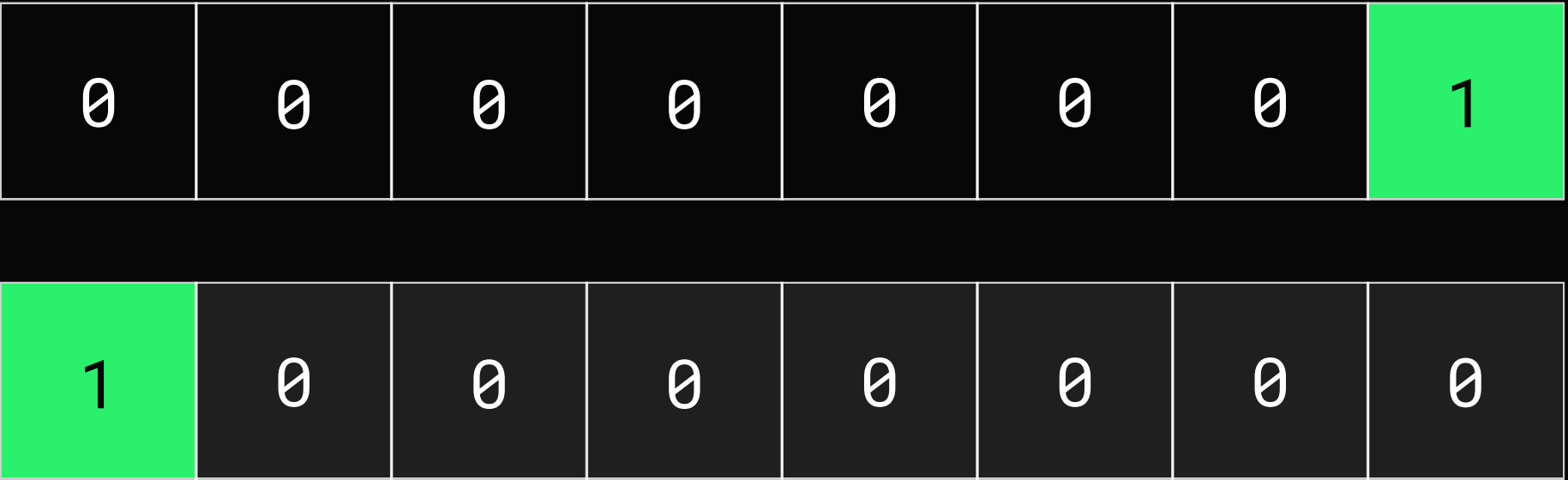
0	0	0	0	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---

```
1 func main() {
2     var value int8 = 1
3     value = value << 6
4     fmt.Println(value) // 64
5 }
```



1<<6

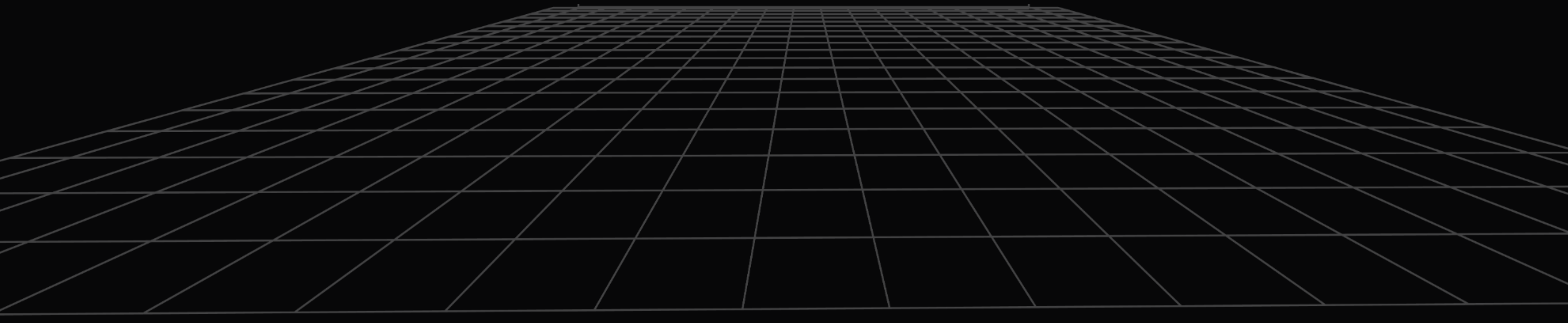
```
1 func main() {
2     var value int8 = 1
3     value = value << 7
4     fmt.Println(value) // -128
5 }
```



1<<7

FAQ

Битовые операции



БИТОВЫЕ ОПЕРАЦИИ НА ПРАКТИКЕ

КАК УЗНАТЬ, ЯВЛЯЕТСЯ ЛИ
ЧИСЛО СТЕПЕНЬЮ ДВОЙКИ?

0	0	0	0	0	0	0	1	$2^0 = 1$
0	0	0	0	0	0	1	0	$2^1 = 2$
0	0	0	0	0	1	0	0	$2^2 = 4$
0	0	0	0	1	0	0	0	$2^3 = 8$
0	0	0	1	0	0	0	0	$2^4 = 16$
0	0	1	0	0	0	0	0	$2^5 = 32$
0	1	0	0	0	0	0	0	$2^6 = 64$
1	0	0	0	0	0	0	0	$2^7 = 128$

Только одна единица в бинарном представлении числа

EXAMPLE

Power of two

ЗАЧЕМ ЭТО НУЖНО ЗНАТЬ?

EXAMPLE

Bool mask

EXAMPLE

Bit mask

**КАК НАЙТИ ЧИСЛО, КОТОРОЕ
ВСТРЕЧАЕТСЯ ВО МНОЖЕСТВЕ
ТОЛЬКО ОДИН РАЗ,**

когда все остальные числа встречаются два раза?

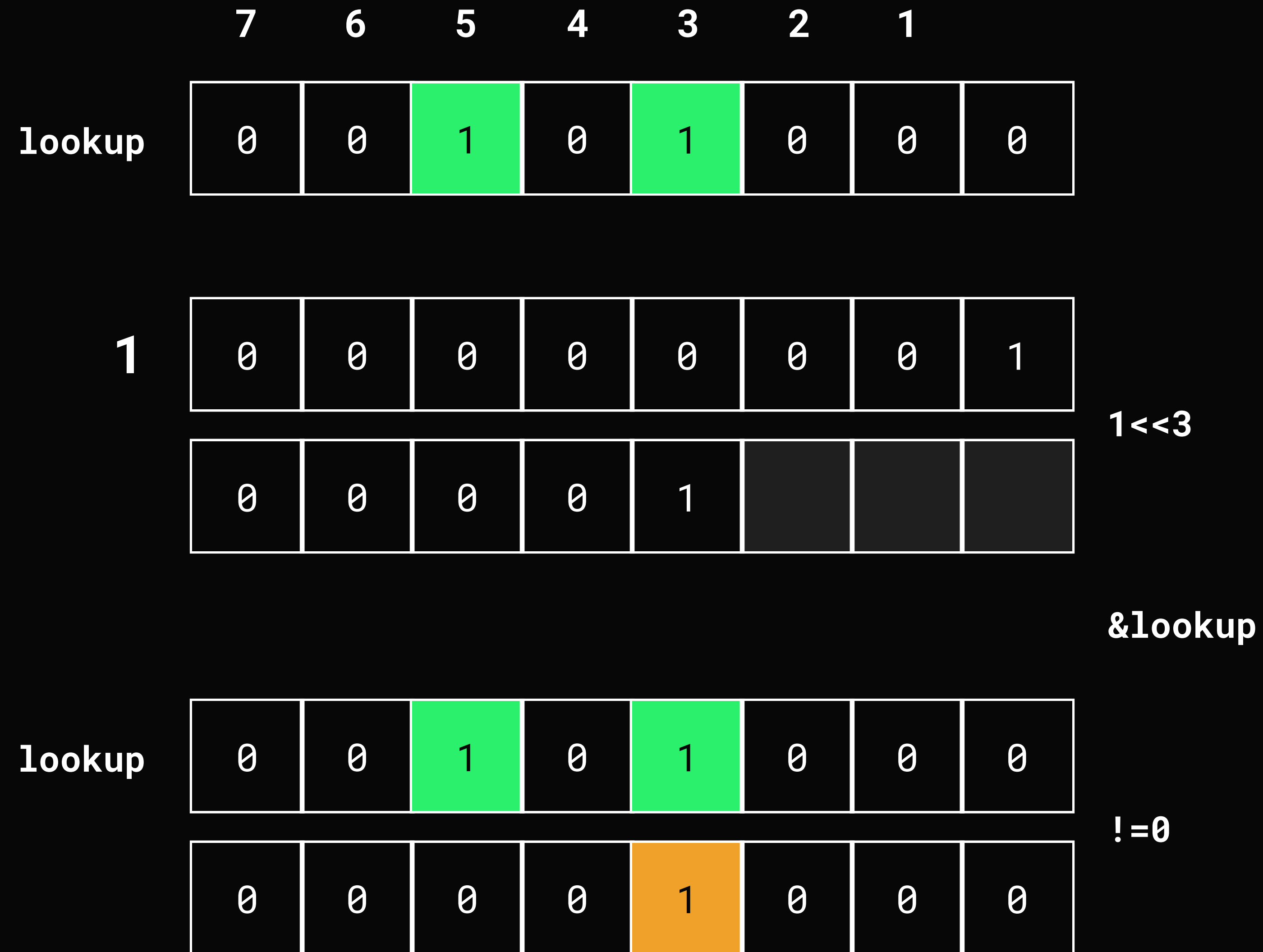
EXAMPLE

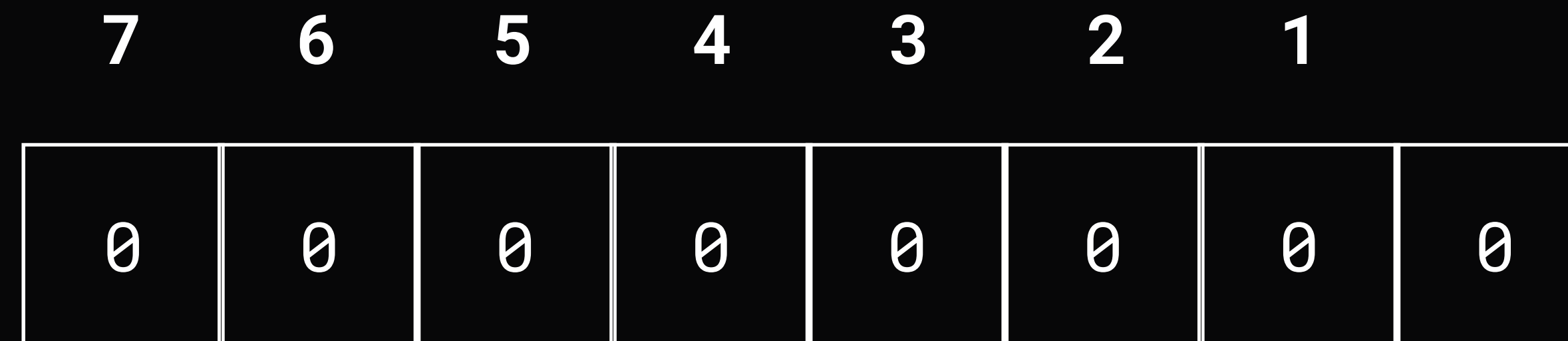
Find unique

**КАК ПРОВЕРИТЬ, ЕСТЬ ЛИ
ДУБЛИКАТЫ ВО МНОЖЕСТВЕ ЧИСЕЛ?**

EXAMPLE

Duplicate check





Битовый карта (*bitset*, *bitmap*, *bit array*) – набор последовательно записанных двоичных разрядов, то есть последовательность битов

EXAMPLE

Bitmap index

**КАК ПРОВЕРИТЬ ЯВЛЯЕТСЯ
ЛИ ЧИСЛО ЧЕТНЫМ?**



```
1 func IsEven(number int) bool {  
2     return number & 1 == 0  
3 }
```

У четных чисел последний бит равен нулю

0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

EXAMPLE

Even test

**КАК КОНВЕРТИРОВАТЬ БАЙТЫ,
КИЛОБАЙТЫ, МЕГАБАЙТЫ?**

EXAMPLE

Conversion test

$$n \ll m == n * 2^m$$

1КБ = 1024 Б

$$1 \ll 10 = 2^{10}$$

1024 Б

1МБ = 1024 КБ

$$1 \ll 20 = 2^{20}$$

1048576 Б

1ГБ = 1024 МБ

$$1 \ll 30 = 2^{30}$$

1073741824 Б

1ТБ = 1024 ГБ

$$1 \ll 40 = 2^{40}$$

1099511627776 Б

1ПБ = 1024 ТБ

$$1 \ll 50 = 2^{50}$$

1125899906842624 Б

КАК ХРАНИТЬ И ПЕРЕДАВАТЬ IPv4 АДРЕС?

Terminal: deep_go x + ∨



IPV4 (INTERNET PROTOCOL V4)

Адрес, записанный в 32-битном формате,
имеет вид четырех 8-битных чисел (минимум
0, максимум 255), которые разделены друг
от друга точками (пример - 172.16.255.2)

EXAMPLE

IPv4 address

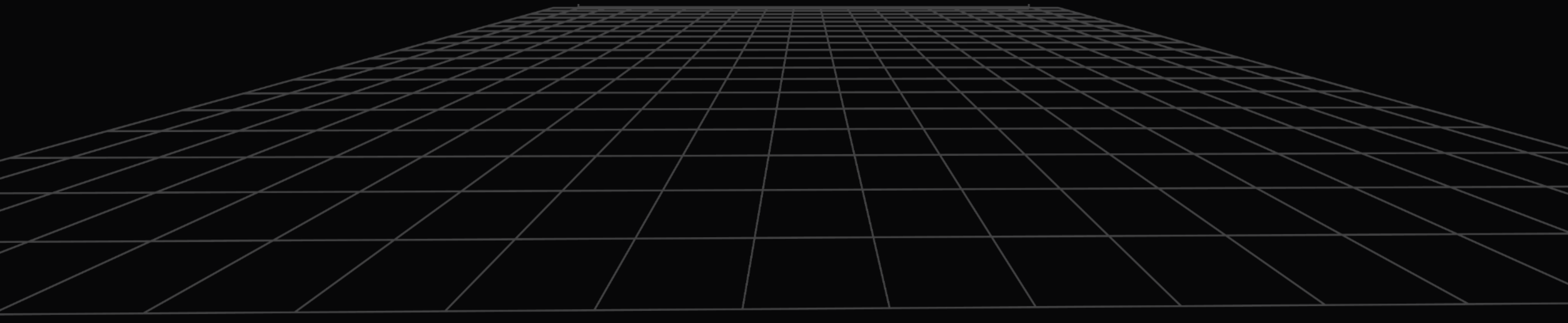
Если вы используете битовые операции
– **скрывайте их за понятными абстракциями!**

EXAMPLE

Bit wrappers

FAQ

Битовые операции на практике



ПОЖАЛУЙСТА, ЗАПОЛНИ ОПРОС О ЗАНЯТИИ

Ссылка в чате и в группе участников

