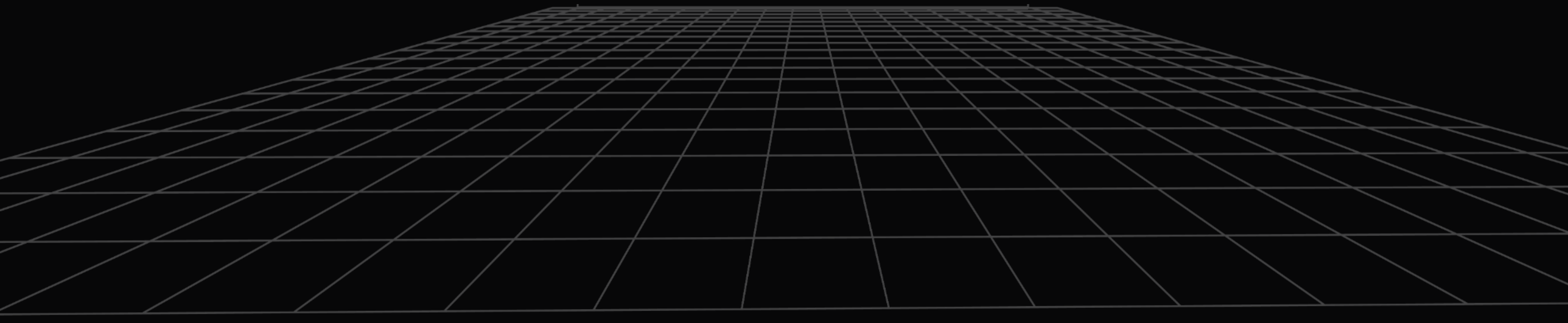
СТРОКИ

Balun.Courses

ПРОВЕРЬ ЗАПИСЬ

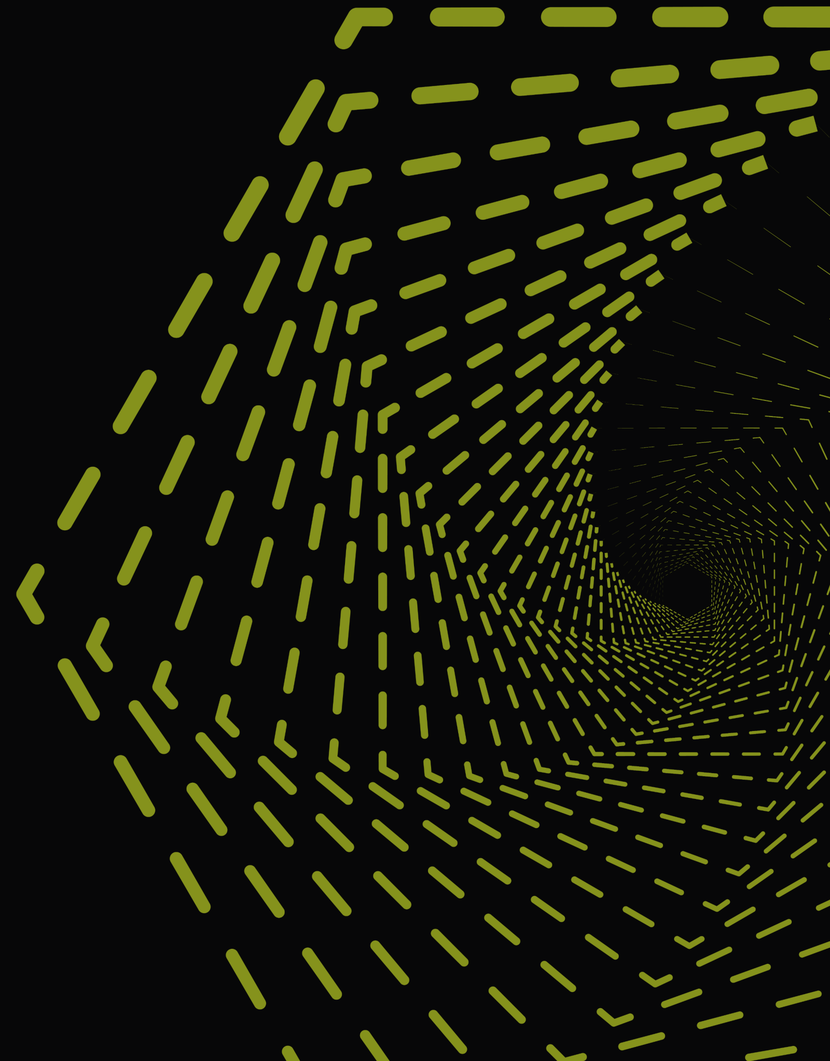


ПРАВИЛА ЗАНЯТИЯ

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

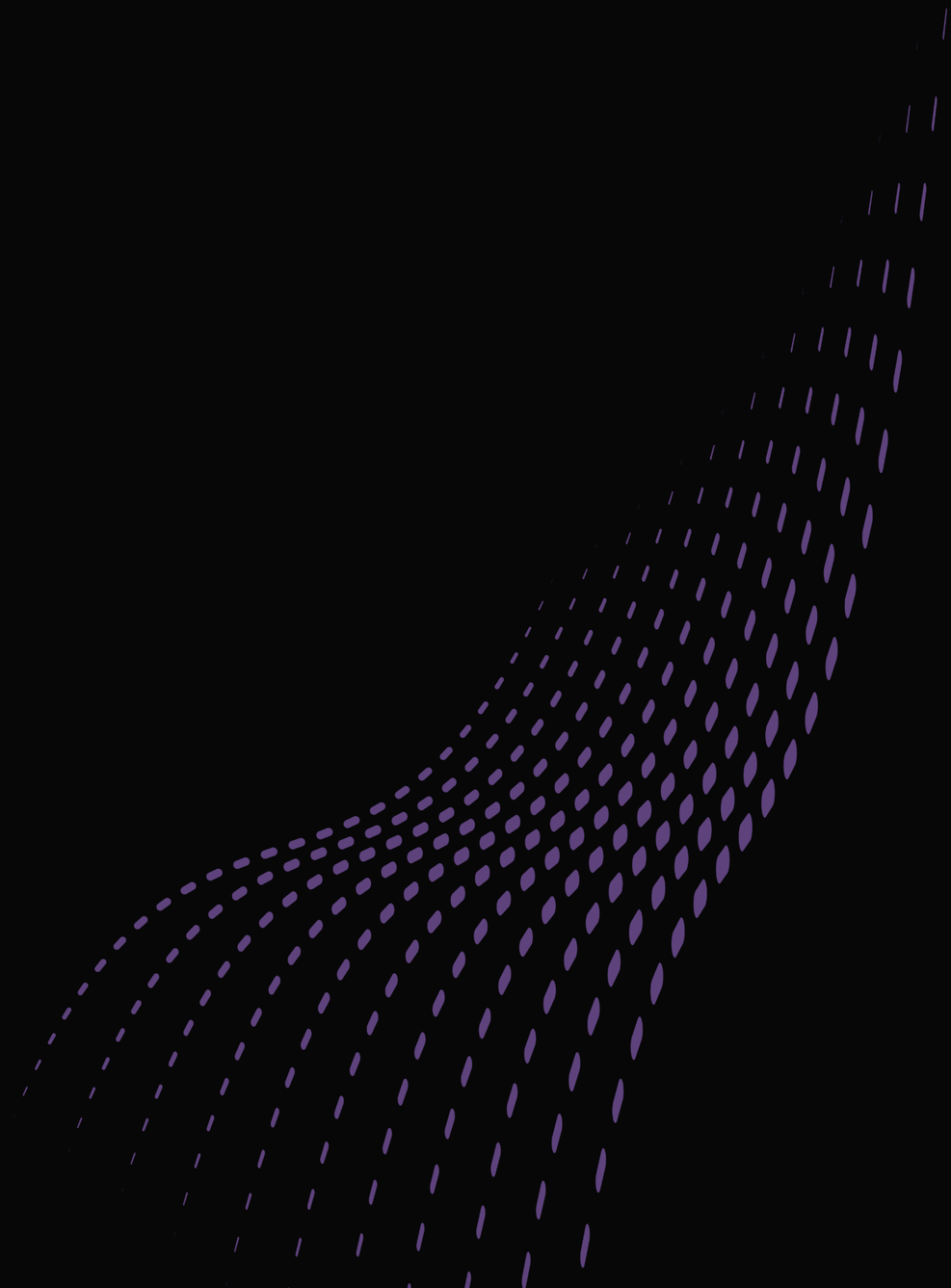
2. вопросы голосом задаем **по поднятой руке в Zoom**

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



ПЛАН ЛЕКЦИИ

1. Кодировки
2. Виды строк
3. Тонкости строк в Go



Terminal: deep_golang

СТРОКА

Тип данных, значениями которого является
произвольная последовательность символов алфавита

КАК РЕАЛИЗОВАТЬ СТРОКУ?

EXAMPLE

String implementation

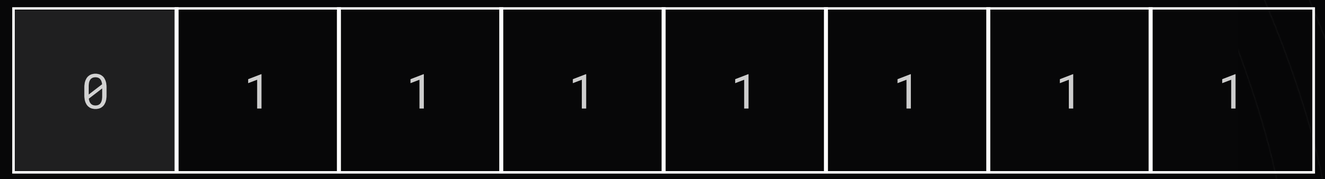
Чтобы одни и те же числа отображались на разных компьютерах одинаково – нужен стандарт по кодированию этих символов

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

American Standart Code for Information Interchange

- от **0** до **127** (128 значений)
- с **0** до **31** - управляющие символы (нечитаемые)
- с **31** до конца символы, имеющие внешний вид
- **128** свободных значений (пустой старший бит)



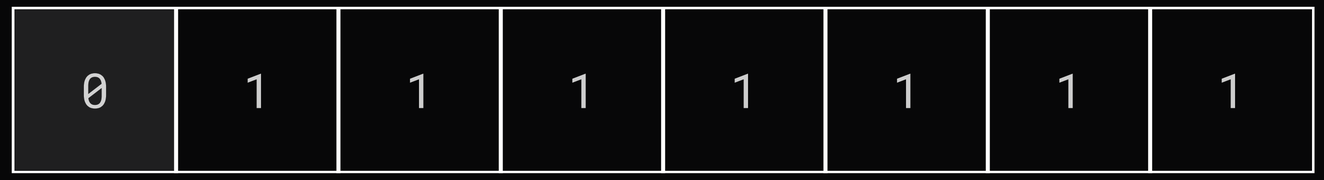
EXAMPLE

Letter order

EXAMPLE

Letter frequencies

КАК БЫТЬ ЕСЛИ В АЛФАВИТЕ МНОГО СИМВОЛОВ?



Решили расширить **ASCII**, введя кодовые страницы. **Кодовая страница** – набор из **256** символов для конкретного языка

(первые 128 символов символов совпадают с ASCII,
а остальные 128 символов для конкретного языка)

256 символов не хватало, чтобы охватить
все возможные языки!

UNICODE

- Первые 128 символов одинаковые с ASCII
- В первой версии два байта для хранения (65 536 значений)

КАКИЕ ПОЯВИЛИСЬ ПРОБЛЕМЫ

когда решили кодировать символы двумя байтам?

ПРОБЛЕМА №1

Излишнее потребление памяти

4 байта

"HI" (Unicode)

2 байта

"HI" (ASCII)

ПРОБЛЕМА №2

Порядок следования байтов

UCS-2

Добавилось два дополнительных байта

BOM (Unicode byte order mask)

Terminal: deep_golang

UTF-8

Новая кодировка, которая стала доминирующей в интернете. Идея заключалась в том, чтобы сделать коды символов не фиксированной, а переменной длины от 1 до 4 байт — первые 128 символов юникода совпадают с символами ASCII

1 байт (от 0 до 7 бит)

2 байта (от 8 до 11 бит)

3 байта (от 12 до 16 бит)

4 байта (от 17 до 21 бита)

Вместо BOM используются маски кодов - если BigEndian,
то первый байт будет начинаться с 0, 110, 1110 или 11110,
а если LittleEndian то первый байт будет начинаться с 10

КАКИЕ ПОЯВИЛИСЬ ПРОБЛЕМЫ У UTF-8?

Переменная длина символов увеличила время обработки строк, так как каждый символ нужно было проверять на его длину, поэтому код стал работать медленнее, из-за этого некоторые языки программирования стали использовать другие кодировки

Terminal: deep_golang

UTF-16

Один из способов кодирования символов
в виде последовательности 16-битных слов

Terminal: deep_golang

UTF-32

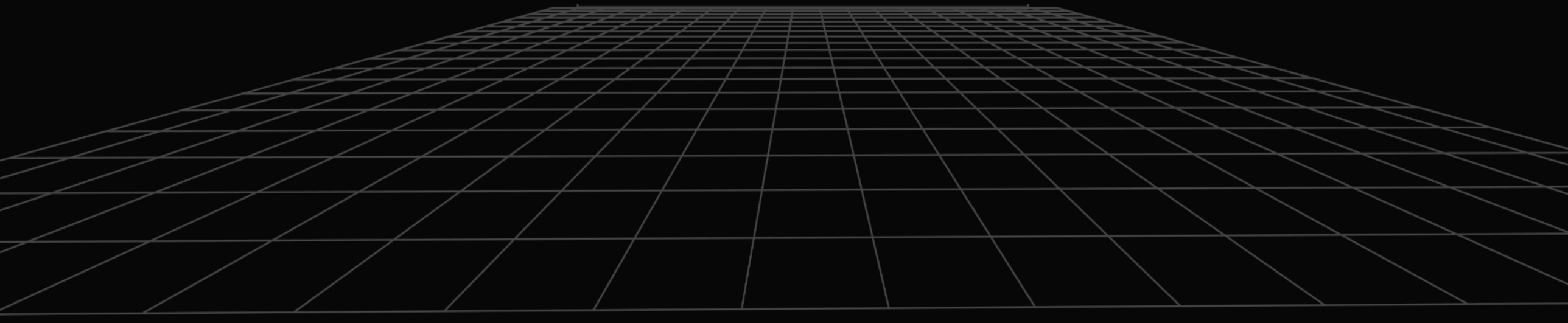
Один из способов кодирования символов, использующий для кодирования любого символа ровно 32 бита

СРАВНЕНИЕ

	Размер	BOM	Скорость обработки
UTF-8	От 1 до 4 байт	Нет	Медленная
UTF-16	2 или 4 байта	Да	Средняя
UTF-32	4 байта	Да	Быстрая

FAQ

Строки

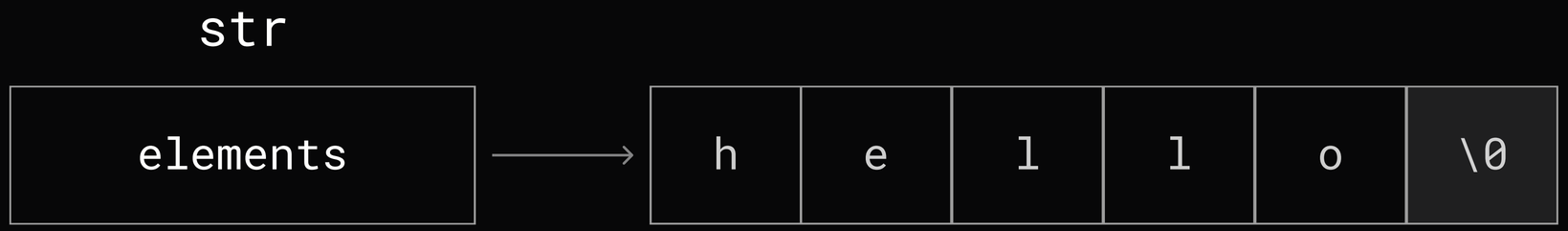


ВИДЫ СТРОК

КАК МОЖНО РЕАЛИЗОВАТЬ СТРОКУ?

НУЛЬ-ТЕРМЕНИРОВАННЫЕ СТРОКИ

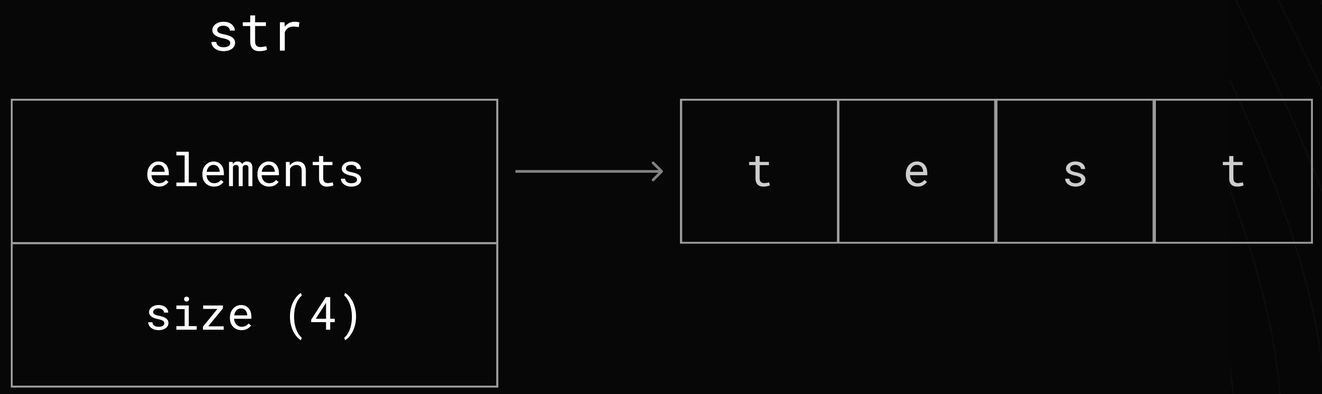
Идея заключается в использовании завершающего байта - как правило, символ с кодом **0** выбирается в качестве признака конца строки, и строка хранится как последовательность байтов от начала до конца (пример — язык C)



КАКИЕ ПРОБЛЕМЫ У НУЛЬ-ТЕРМИНИРОВАННЫХ СТРОК?

PASCAL СТРОКИ

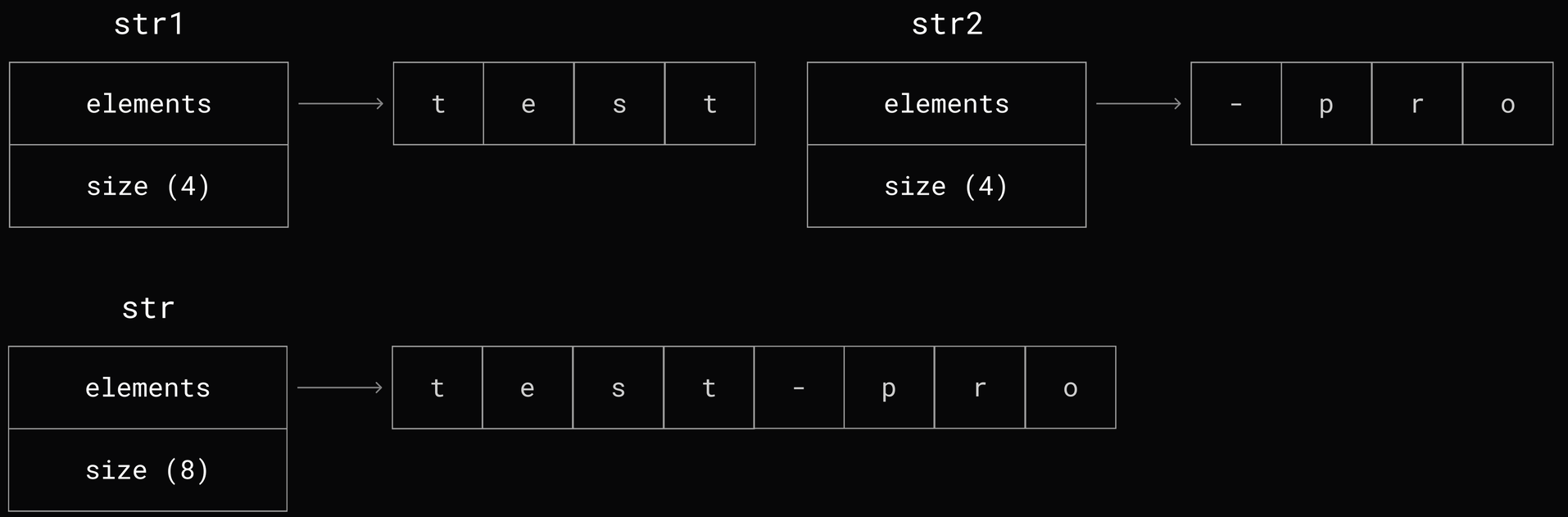
Идея заключается в использовании отдельного места в памяти для хранения размера строки



МОЖНО ЛИ МЕНЯТЬ СИМВОЛЫ В СТРОКАХ?

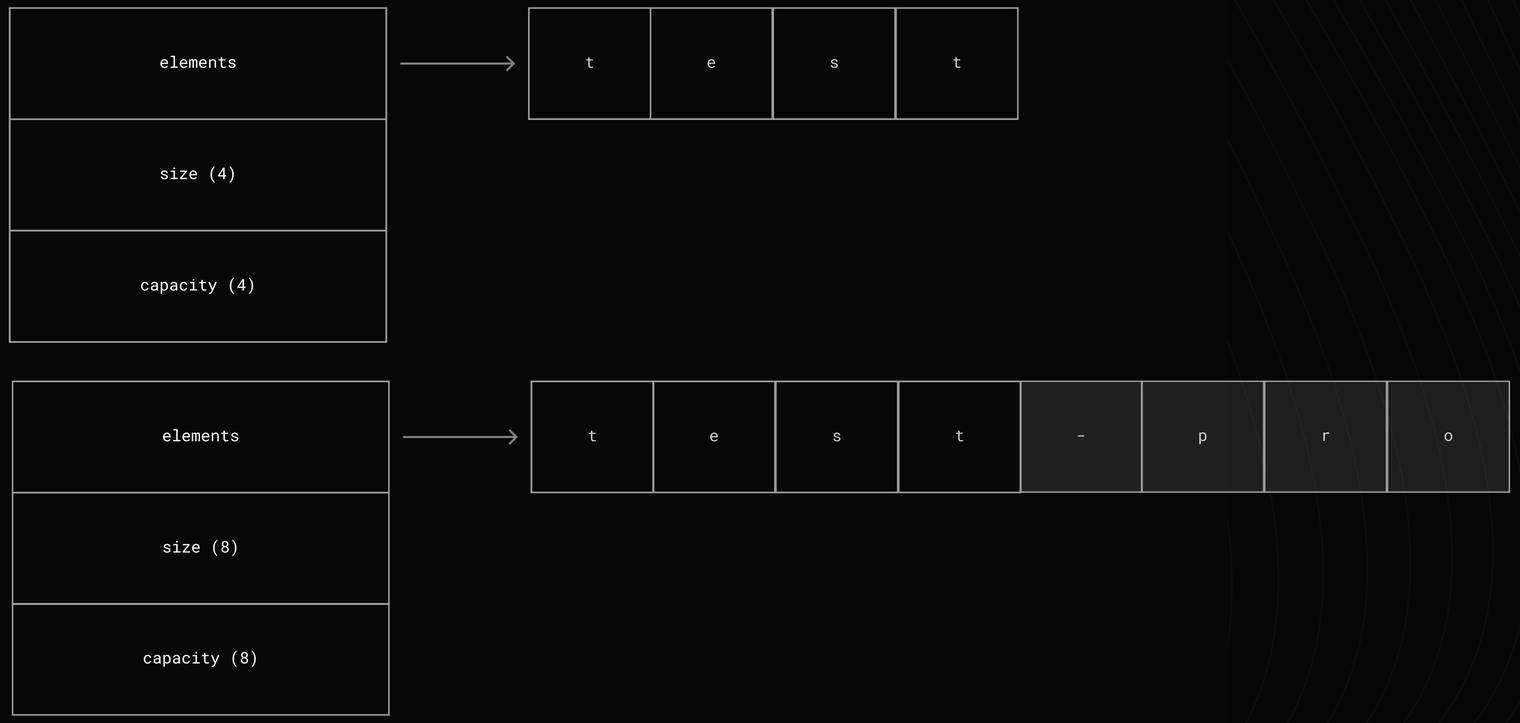
НЕИЗМЕНЯЕМЫЕ СТРОКИ

Идея заключается в том, что строку нельзя изменить – как правило, конкатенация строк возвращает новую строку (пример – язык Go, Java, C#, JavaScript)



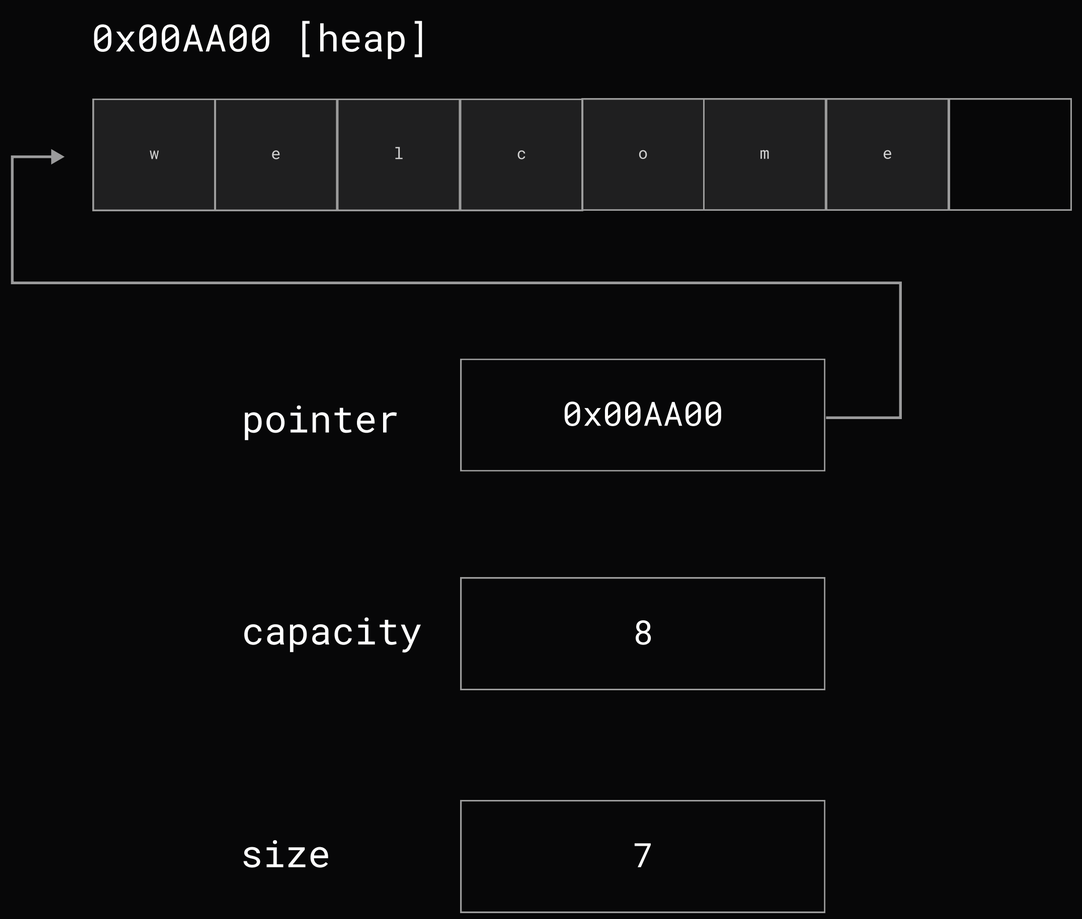
ИЗМЕНЯЕМЫЕ СТРОКИ

Идея заключается в том, что строку можно изменить – как правило, растет по такому же сценарию, как и срез (пример – язык C++)



КАКИЕ ПРЕИМУЩЕСТВА И НЕДОСТАТКИ У НЕИЗМЕНЯЕМЫХ И ИЗМЕНЯЕМЫХ СТРОК?

Какую оптимизацию можно придумать по работе с маленькими строками?



SSO

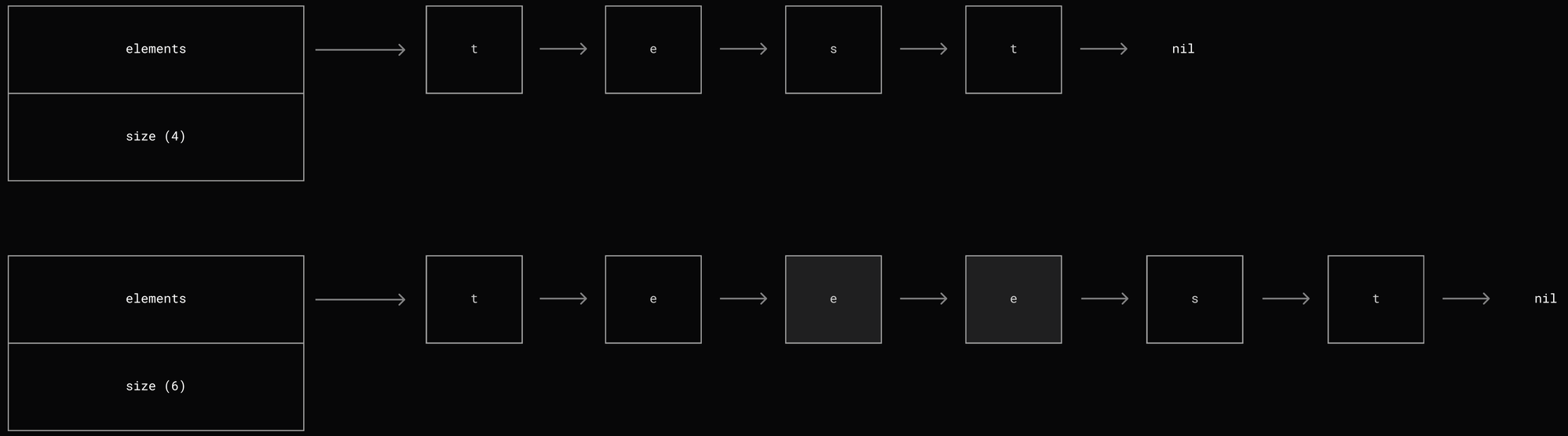
Small String Optimization

ЧТО ДЕЛАТЬ

когда нужно удалять символы из разных мест строки
и добавлять символы в разные места строки?

СПИСКИ СИМВОЛОВ

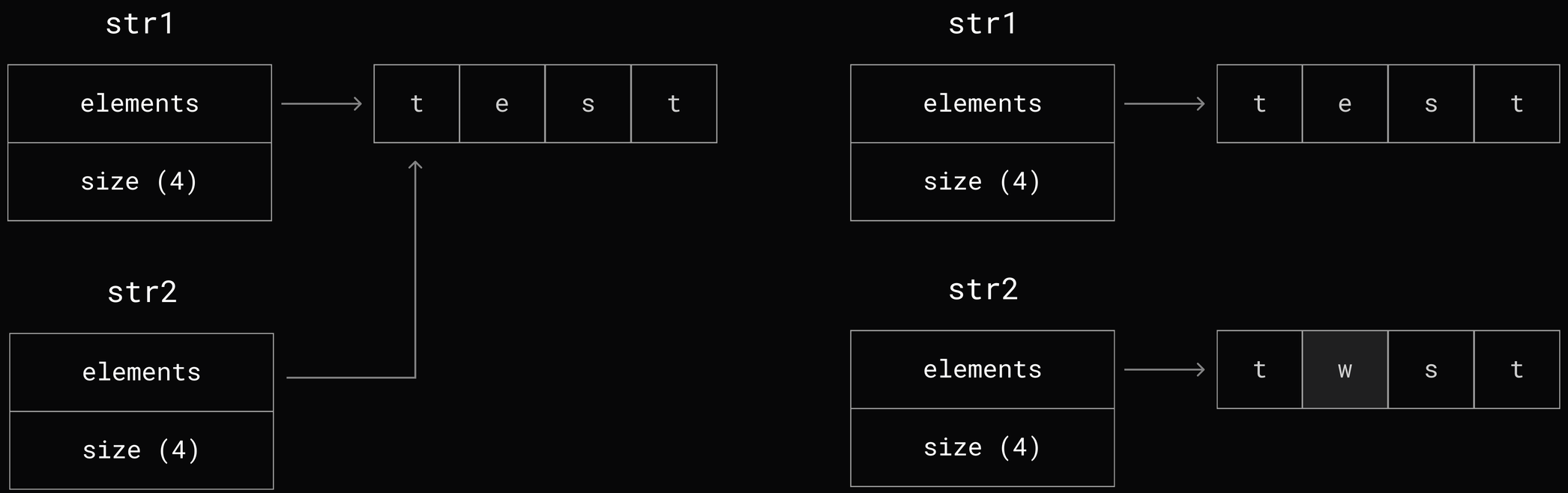
Идея заключается в том, что строку можно представить в виде связного списка символов (пример — язык Erlang и Haskell)



Есть еще интересные виды строк...

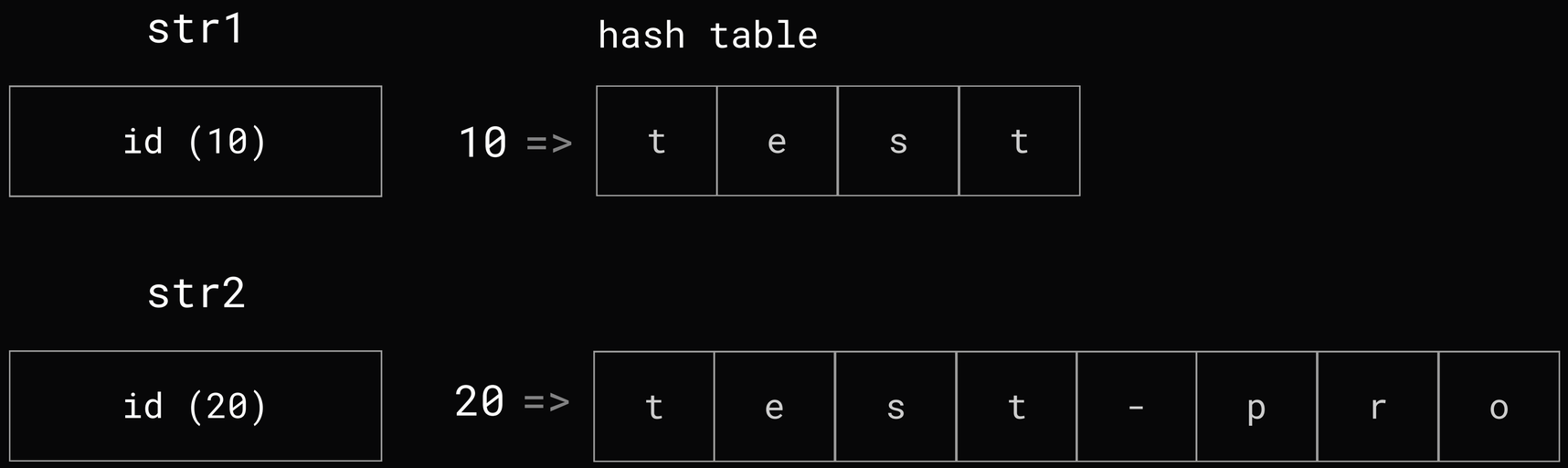
COW СТРОКИ

Copy-on-Write – идея заключается в том, что при чтении строки используется общая память, в случае изменения – создается копия



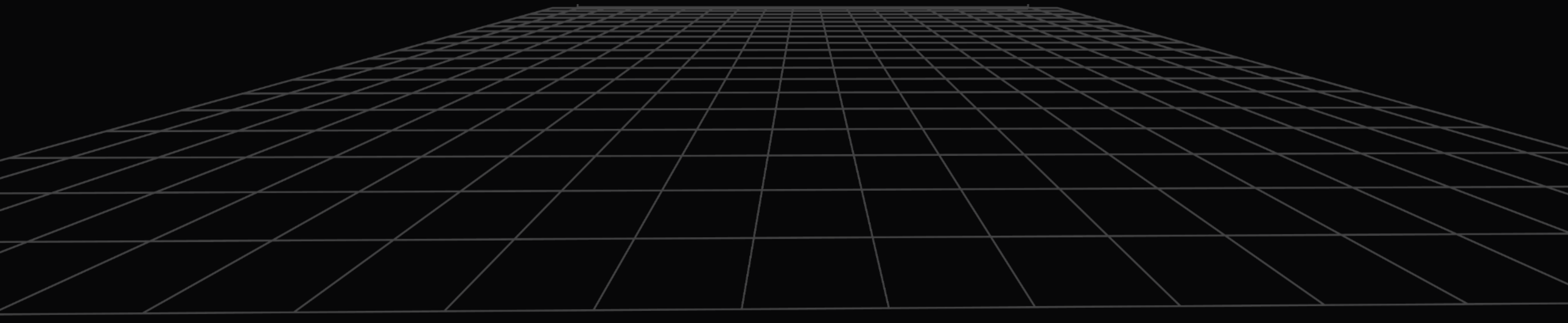
ИНТЕРНИРОВАНИЕ СТРОК

Идея заключается в том, чтобы хранить в памяти только один экземпляр типа строки для идентичных строк



FAQ

Виды строк



ТОНКОСТИ СТРОК В GO

СТРОКА В GO

Последовательность байт
и ничего более

```
type _string struct {
    elements *byte // underlying bytes
    len      int   // number of bytes
}
```

Некоторые считают, что строки в **Go** всегда имеют кодировку **UTF-8**, но это не так — строки представляют собой последовательность произвольных байтов (например, при чтении файла - мы можем прочитать содержимое в абсолютно любой кодировке).

Но исходный код представлен в **UTF-8**, то есть все строковые литеры кодируются в последовательность байтов с использованием **UTF-8**

Terminal: deep_golang

RUNE

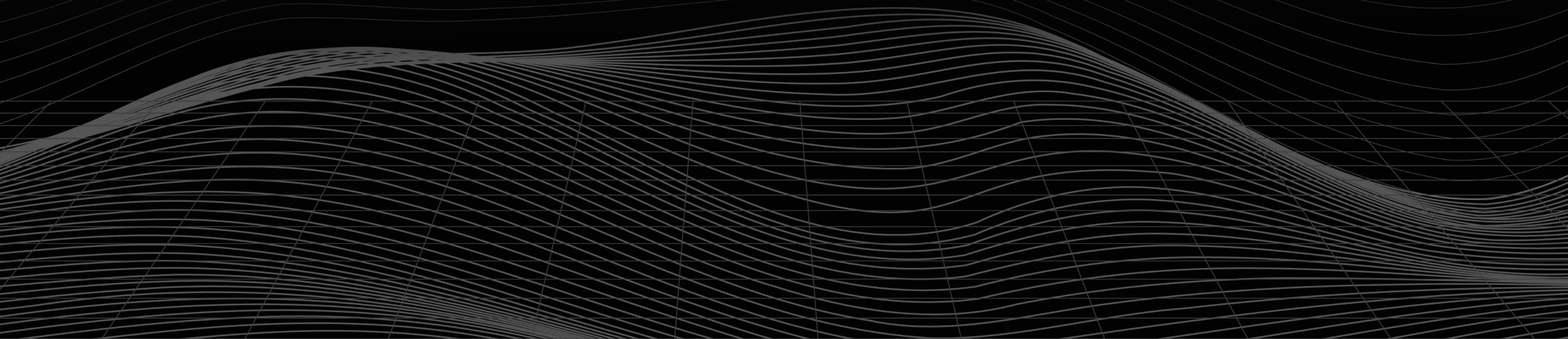
Кодовые точки представлены в Go как значения рун,
в **Go rune** – это не что иное, алиас типа int32

EXAMPLE

Encodings

EXAMPLE

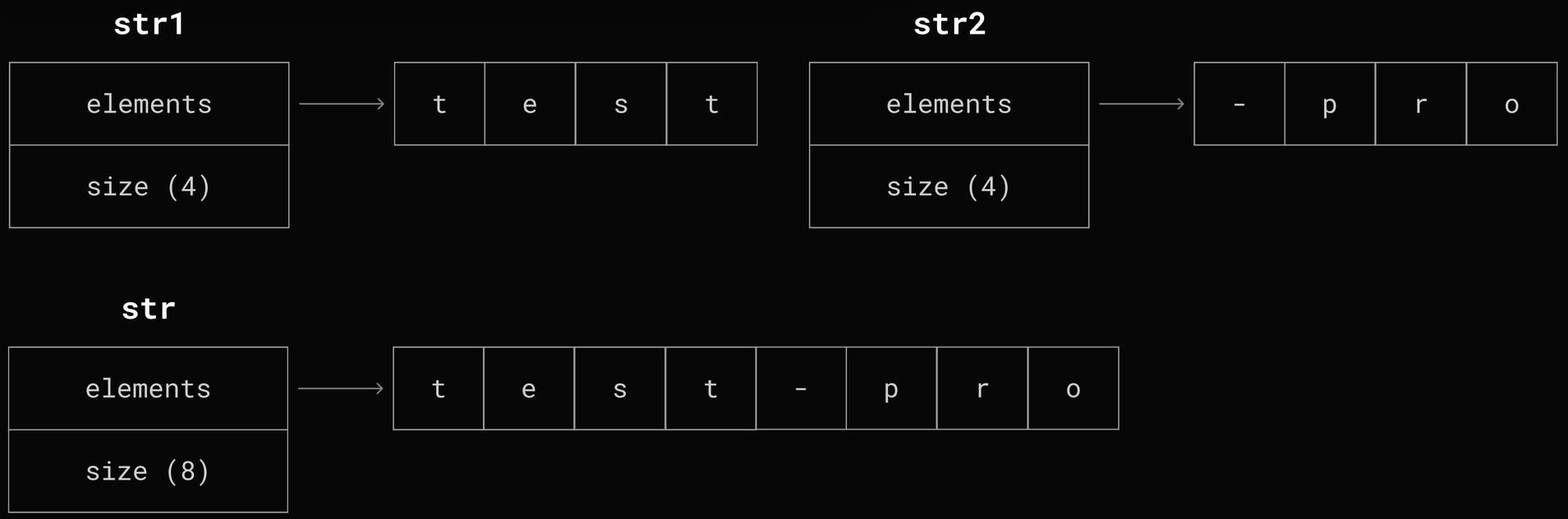
Interpreted and raw strings



СТРОКИ В GO

- строки используются, как константы (неизменяемые)
- строки можно конкатенировать с использованием операторов **+** и **+=**

```go
str1 := "test"
str2 := "-pro"
str := str1 + str2
```



EXAMPLE

Speed concatenation

String concatenations with the + operator are specially optimized so that only one memory allocation is made in each of such concatenations, no matter how many strings are concatenated in a s0 + s1 + ... + sn expression

КАК ДЕЛАТЬ КОНКАТЕНАЦИЮ СТРОК БЫСТРЕЕ?

ПЕРЕРЫВ 5 МИНУТ

EXAMPLE

String Builder

КАК РЕАЛИЗОВАТЬ STRINGS.BUILDER?

EXAMPLE

String Builder implementation

EXAMPLE

String Builder copy

СТРОКИ В GO

Строки можно сравнивать между собой
с использованием операторов **!=, ==, <, >, <= и >=**
(строки сравниваются друг с другом байт за байтом)

Сравнение строк на равенство реализовано таким образом, что сначала проверяются размеры строк, затем адреса указателей и только потом итерация по каждому байту

EXAMPLE

Speed comparison

СТРОКИ В GO

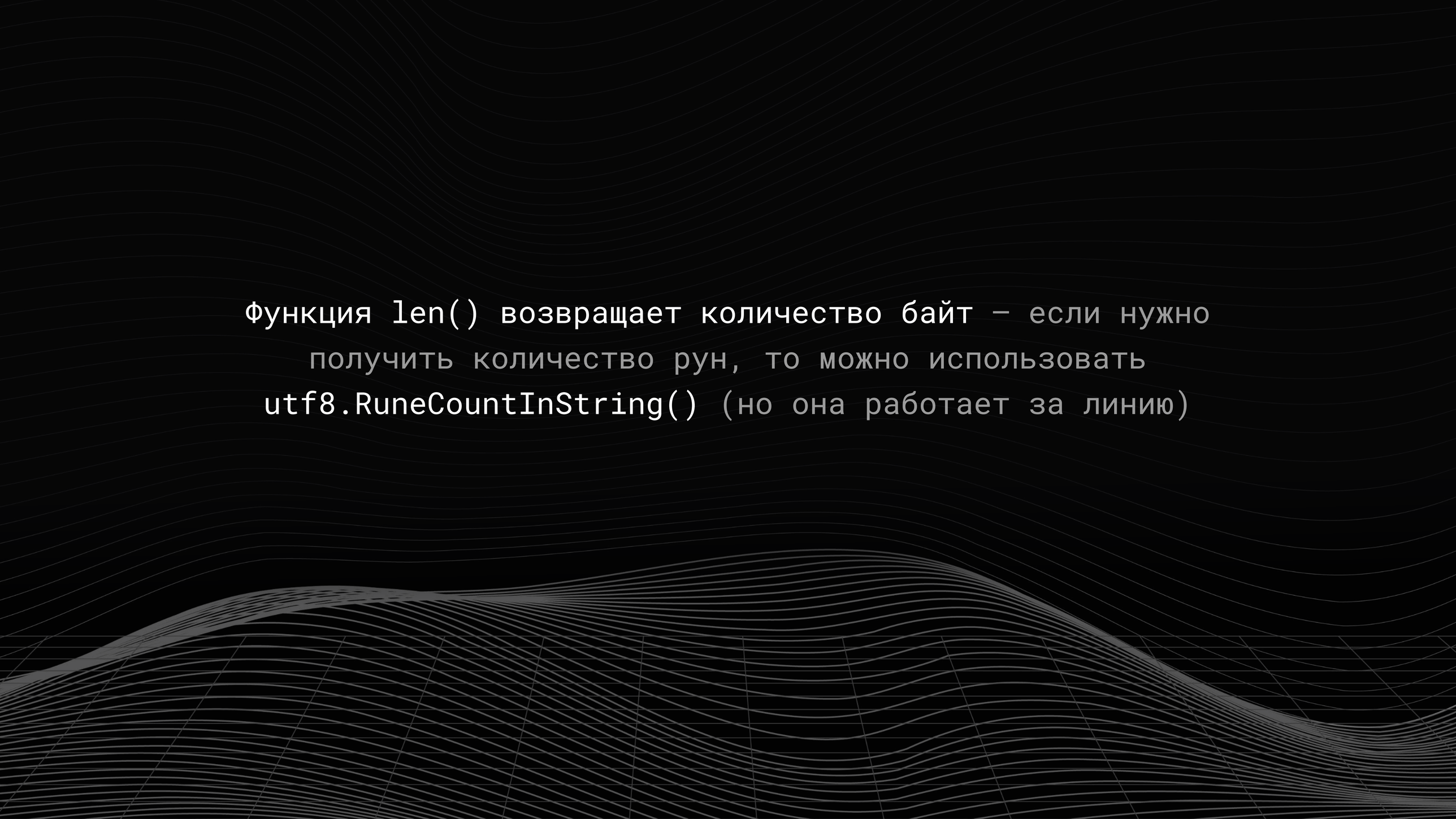
- можно узнать длину строки
  с использованием функции **len(x)**
- можно обратиться к n-ому байту строки
  с использованием оператора **[x]**
- можно получить подстроку
  с использованием оператора
  слайсинга **[x:x]**
- нельзя взять указатель
  на один из байт строки

ЧТО ВОЗВРАЩАЕТ ФУНКЦИЯ LEN() ДЛЯ СТРОКИ?

EXAMPLE

String len

Функция len() возвращает количество байт – если нужно получить количество рун, то можно использовать utf8.RuneCountInString() (но она работает за линию)

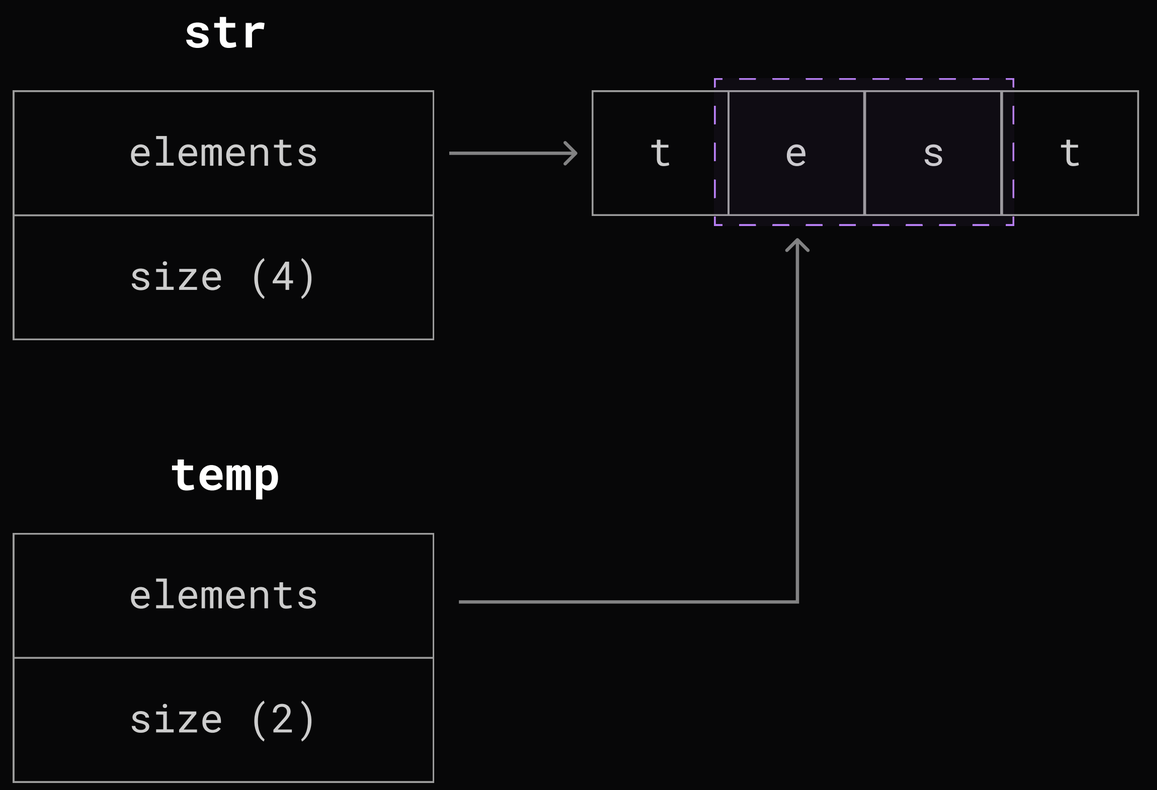


Вызов функции **len()** для константной сроки вычисляется на этапе компиляции, а не в рантайме

КАК ПОЛУЧИТЬ ПОДСТРОКУ ИЗ СТРОКИ?

В результате получается строка, а не срез

```
1 str := "test"
2 substr := str[1:3]
```



EXAMPLE

Substring

ПОЧЕМУ НЕЛЬЗЯ ВЗЯТЬ УКАЗАТЕЛЬ НА ОДИН ИЗ БАЙТОВ СТРОКИ?

EXAMPLE

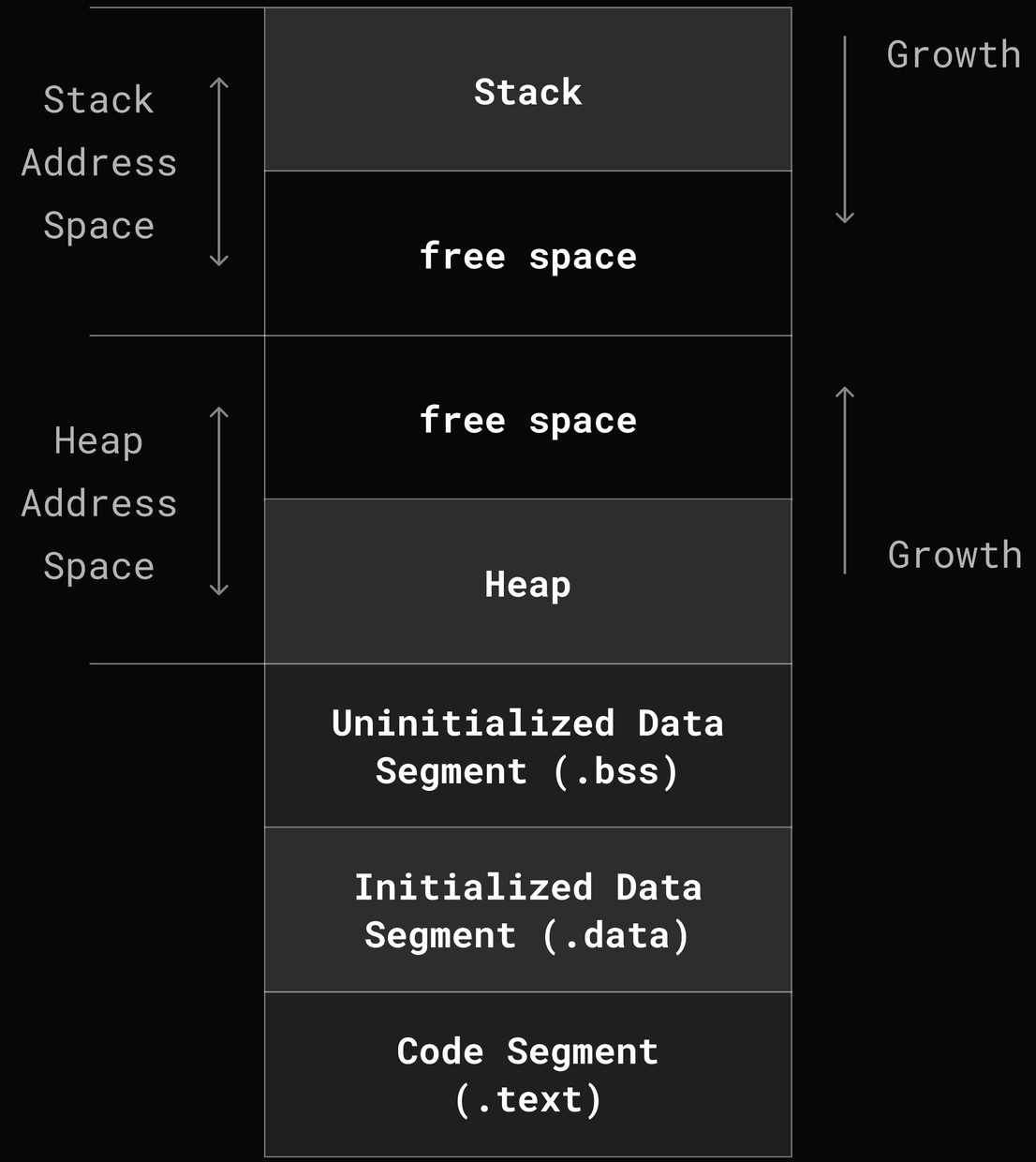
Byte address

EXAMPLE

Mutate string 1

EXAMPLE

Mutate string 2


Stack
Address
Space
Stack
free space
Growth
Heap
Address
Space
free space
Heap
Growth
Uninitialized Data
Segment (.bss)
Initialized Data
Segment (.data)
Code Segment
(.text)

EXAMPLE

Const strings

СТРОКИ В GO

- можно конвертировать в **[]byte** и наоборот
- можно конвертировать в **[]rune** и наоборот

При конвертации будет **аллокация памяти** - так как строки неизменяемые в отличии от срезов, поэтому они не должны разделять данные

```
1 str := "test"
2 slice := []byte(str)
```

EXAMPLE

Byte slice to string

EXAMPLE

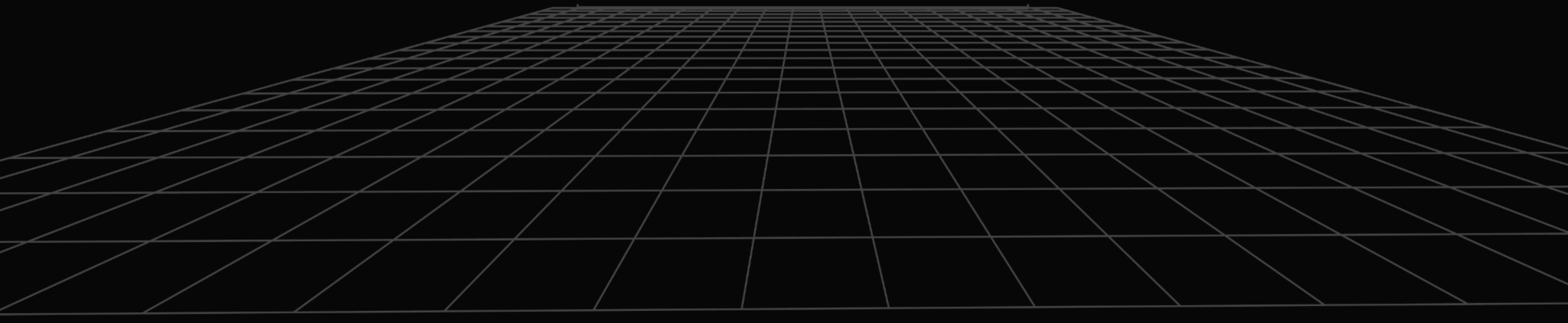
String to byte slice

EXAMPLE

Conversion deprecated

FAQ

Тонкости строк в Go



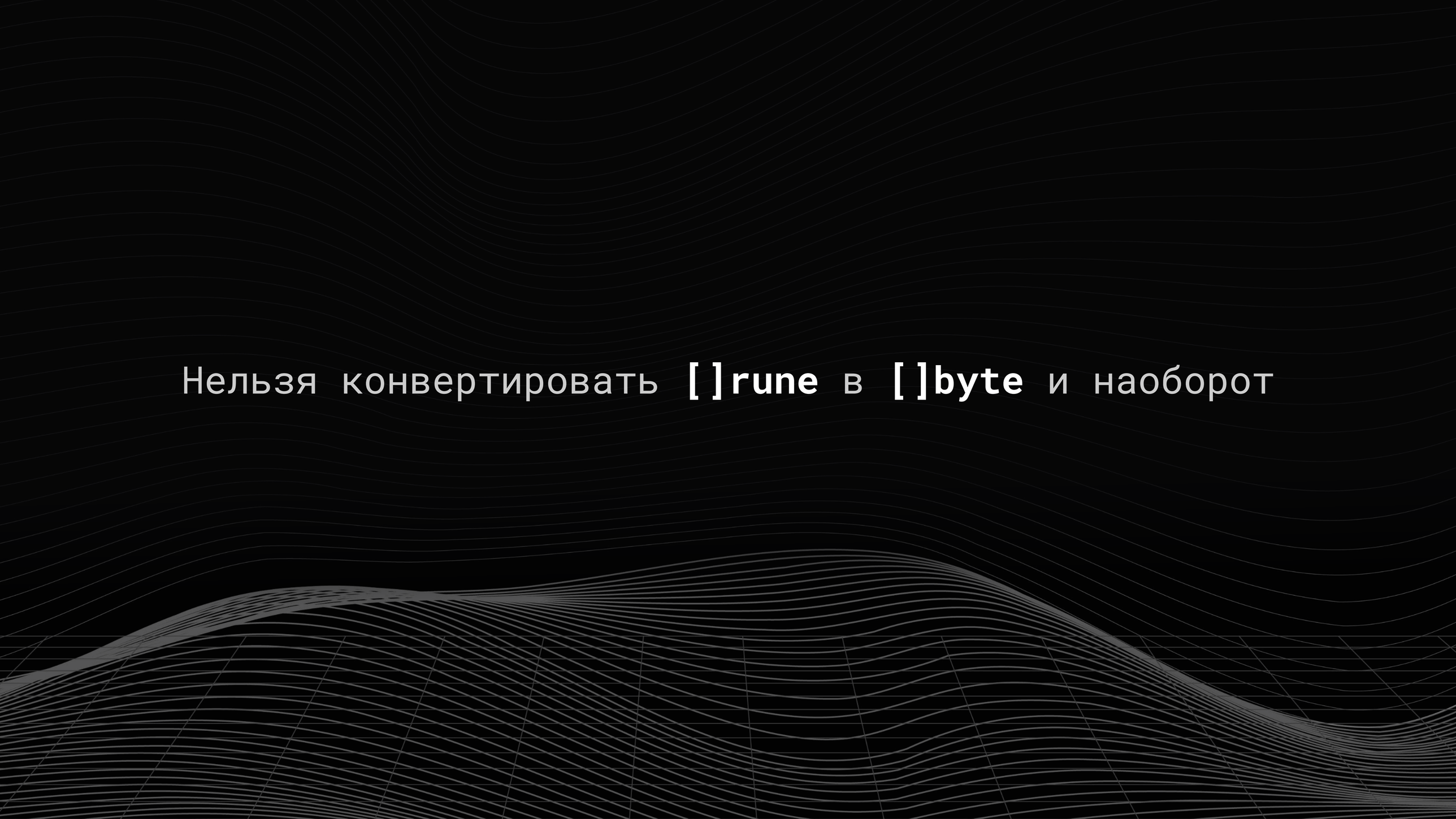
ОПТИМИЗАЦИИ КОМПИЛЯТОРА

- преобразование из строки в байтовый срез, которое происходит внутри range
- преобразование из среза байт в строку, которое используется в качестве ключа словаря во время чтения из словаря
- преобразование из среза байт в строку, которое используется при сравнении
- преобразование из среза байт в строку, которое используется при конкатенации строк (по крайней мере одно из значений объединенной строки должно являться непустой строковой константой)

EXAMPLE

Conversion optimizations

Нельзя конвертировать **[]rune** в **[]byte** и наоборот



EXAMPLE

Runes to bytes

EXAMPLE

Rune to bytes test

КАК МОЖНО ИТЕРИРОВАТЬСЯ ПО СТРОКЕ?

EXAMPLE

Strings range

ЕСТЬ ЛИ РАЗНИЦА

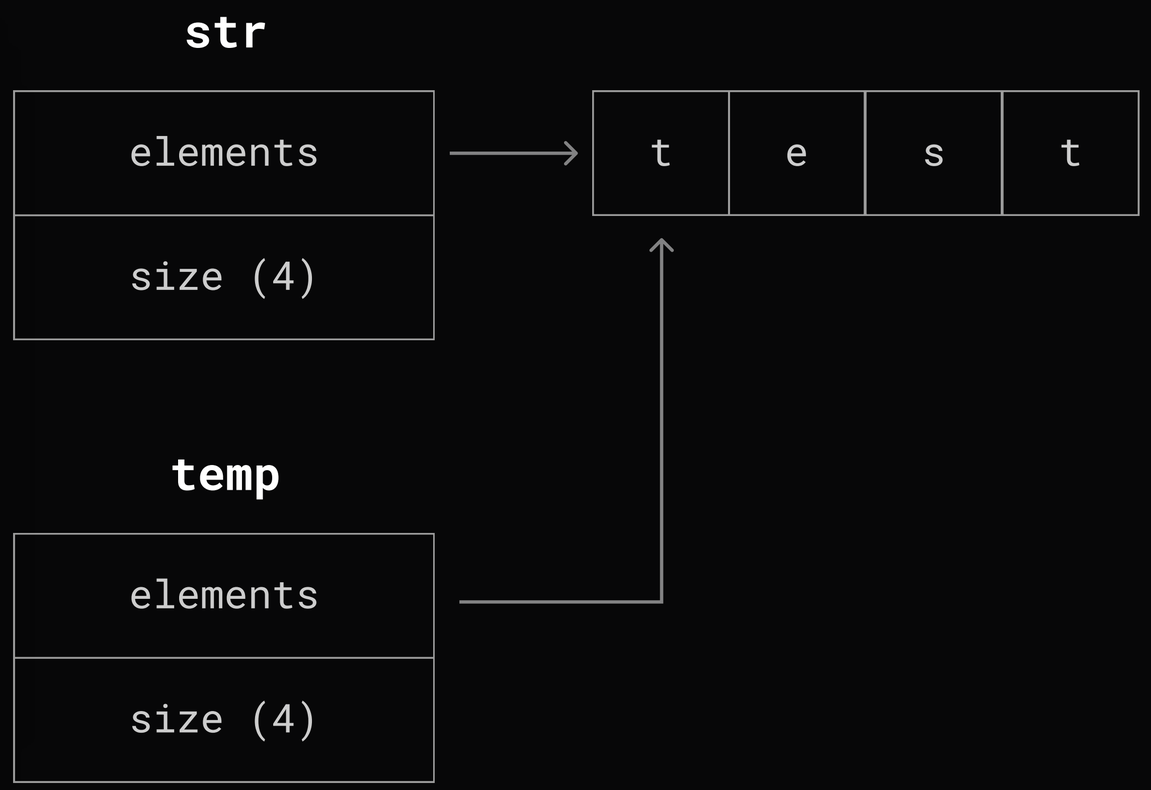
в производительности при итерации
по срезу байт и строке?

EXAMPLE

Range ASCII vs UTF-8

ЧТО ПРОИСХОДИТ, КОГДА СТРОКА ПЕРЕДАЕТСЯ В ФУНКЦИЮ?

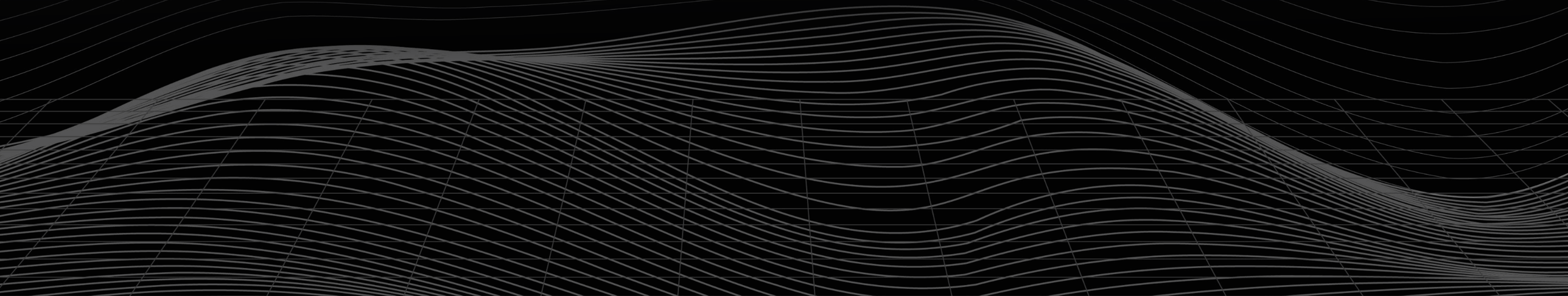
```go
func process(temp string) {
    ...
}

func main() {
    str := "test"
    process(str)
}
```



Terminal: question +

ГДЕ АЛЛОЦИРУЮТСЯ СТРОКИ?

Строковые литералы могут храниться в TEXT (read only) или DATA сегментах, а строки, созданные динамически во время выполнения, хранятся либо в STACK, либо в HEAP - в зависимости от использования



EXAMPLE

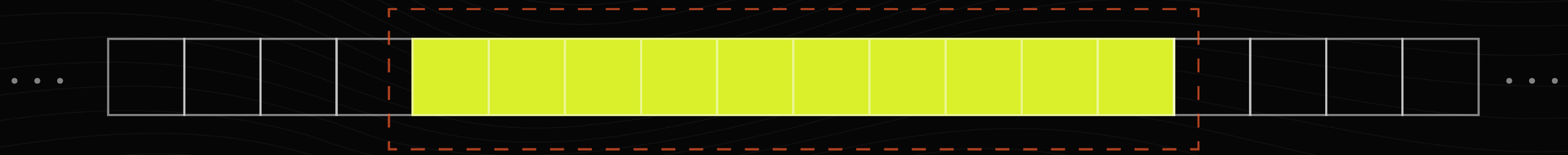
String operations

EXAMPLE

Append and copy

EXAMPLE

Leak with string



Предотвращение утечки

```
1 str := string([]byte(data[i+2 : i+22]))
```

EXAMPLE

BCE optimization

EXAMPLE

Trim right and trim suffix

Documentation

Overview

The unique package provides facilities for canonicalizing ("interning") comparable values.

Index

EXAMPLE

Unique

ПОЖАЛУЙСТА, ЗАПОЛНИ ОПРОС О ЗАНЯТИИ

Ссылка в чате и в группе участников