7 идиом Go

и при чем здесь программисты Fortran?

Влад Белогрудов, инженер,
разработчик «вишенок для торта»

О чем?

Рассмотрим наиболее часто встречающиеся фишки Go и как ими правильно пользоваться.

Просто о непростом const

В Go нет неизменяемых объектов. Да ладно, как так?

```
01    package main
02
03    const a int = 10
04
05    func main() {
06        b := &a
07    }

// main.go:6:8: invalid operation: cannot take address of a (untyped int constant 10)
```

Просто о непростом const

Нетипизированные константы, своя "арифметика", просто числа

```
02
03    const (
04        a = 10
05        b  = 20.2
06    )
07
08    func main() {
09        fmt.Println(a * b)    // Нет необходимости: float64(a) * b
10    }

// 202
```

Просто о непростом const

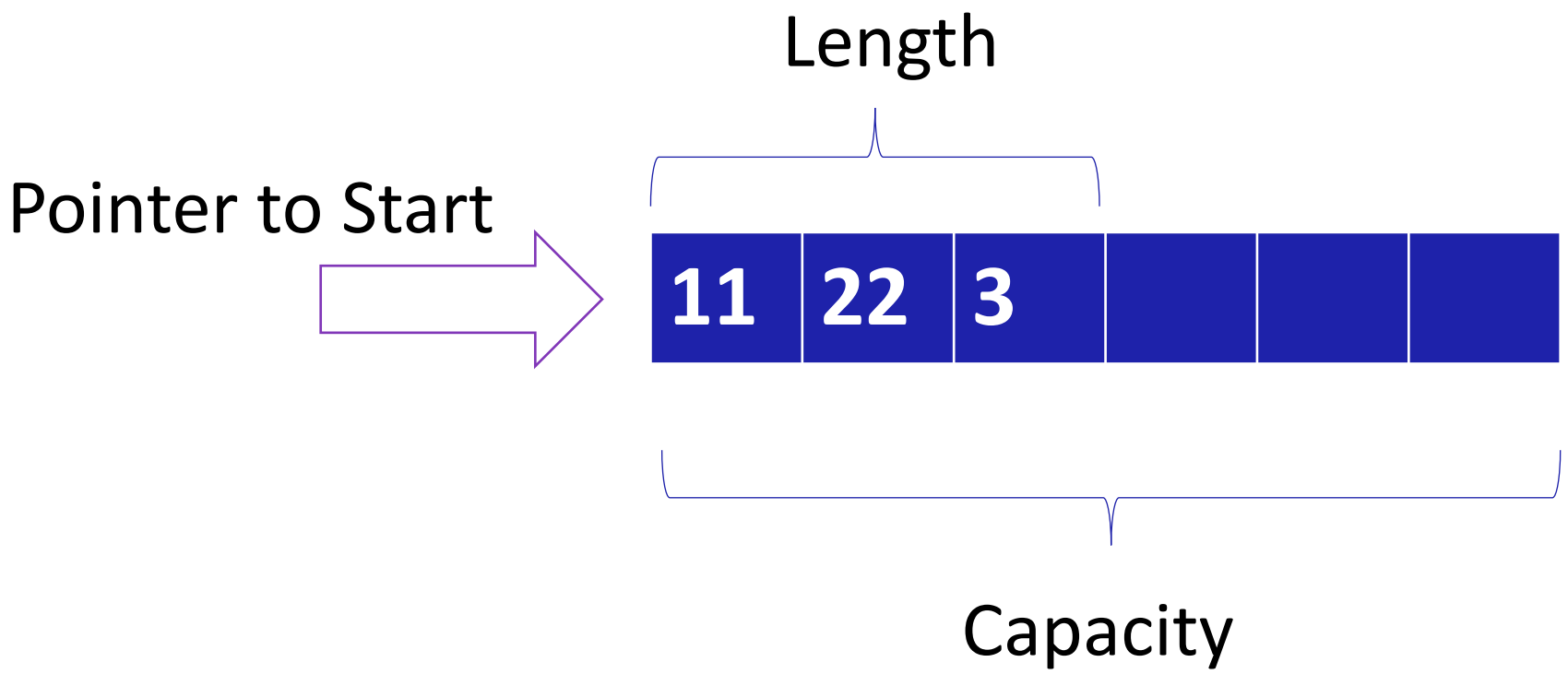
Просто высокоточные числа:

```
02    // https://go.dev/ref/spec#Constants
03    const (
04        a = 1 << 511
05        b = 1 << 510
06    )
07
08    func main() {
09        fmt.Println(a / b)
10    }

// 2
```

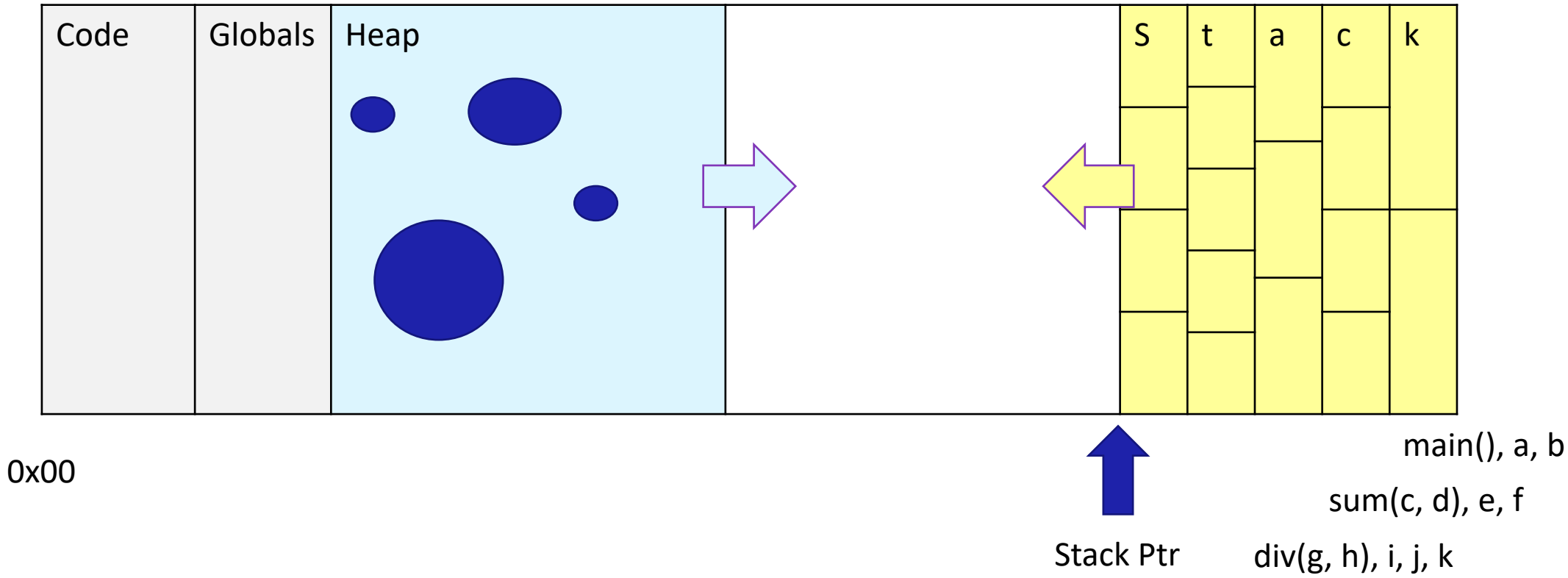
Что такое слайс?

Динамический массив, все по классике:



Про стек и кучу.. Вы это конечно знаете ☺

Просто напоминалка-освежалка



Аргументируем правильно

Все передается по значению – числа, строки, структуры, указатели. Измениться могут только объекты переданные с помощью указателей

```
..01
..02    // где-то в коде Golang
..03    type slice struct {
..04        array unsafe.Pointer
..05        len   int
..06        cap int
..07    }
```

```
02
03    func Modify(s []int) {
04        // bubble sort it :p
05        ..
06        // add more - compile, run, but doesn’t work!
07        s = append(s, 99, 100)
08    }
```

Нарезаем правильно

Слайсы – копирование

```
05    a := []int{1, 2, 3, 4}
06
07    b := a[:]  // тонкая копия a, тоже самое, что  = a[0:3]
08
09    c := make([]int, 4)
10    copy(c, a)  // полная копия a, безопасно для a. c должно существовать с тем же размером
11
12    var d []int
13    d = append(d, a...)  // полная копия a, безопасно для a
```

Нарезаем правильно

Слайсы – как тебе такое, Илон Маск?

```
04    a := []int{1, 2, 3, 4}
05
06    copy(a[1:], a[:3])  // ??
07    copy(a[:3], a[1:])  // ??
08
09    b := []int{1, 2, 3, 4}
10    b = append(b[:2], 99, b[2:]...) // ??
11
12    b = append(b[3:], b[:len(b)-1]...) // ??
13
14
```

Аргументируем правильно

Map – хитрый тип, на самом деле это указатель! https://go.dev/src/runtime/map.go

```
..
..    // A header for a Go map.
..    type hmap struct {
..        count     int // # live cells == size of map.
..        hash0     uint32 // hash seed
..        buckets    unsafe.Pointer // array of 2^B Buckets. may be nil if count==0.
..        ..
..    }
..
..    // makemap implements Go map creation for make(map[k]v, hint)
..    func makemap(t *maptype, hint int, h *hmap) *hmap {
.
```

Аргументируем правильно

А если все передавать через указатели? Есть ли какие-то «лучшие практики»?

```
02
03    type TeamScore struct {
04        Team string
05        Score int
06    }
07
08    func main() {
09        var ts TeamScore
10        input := `{“Team”: “Yadro”, “score”: 777}`
11        json.Unmarshal([]byte(input), &ts)
12         ..
13    }
```

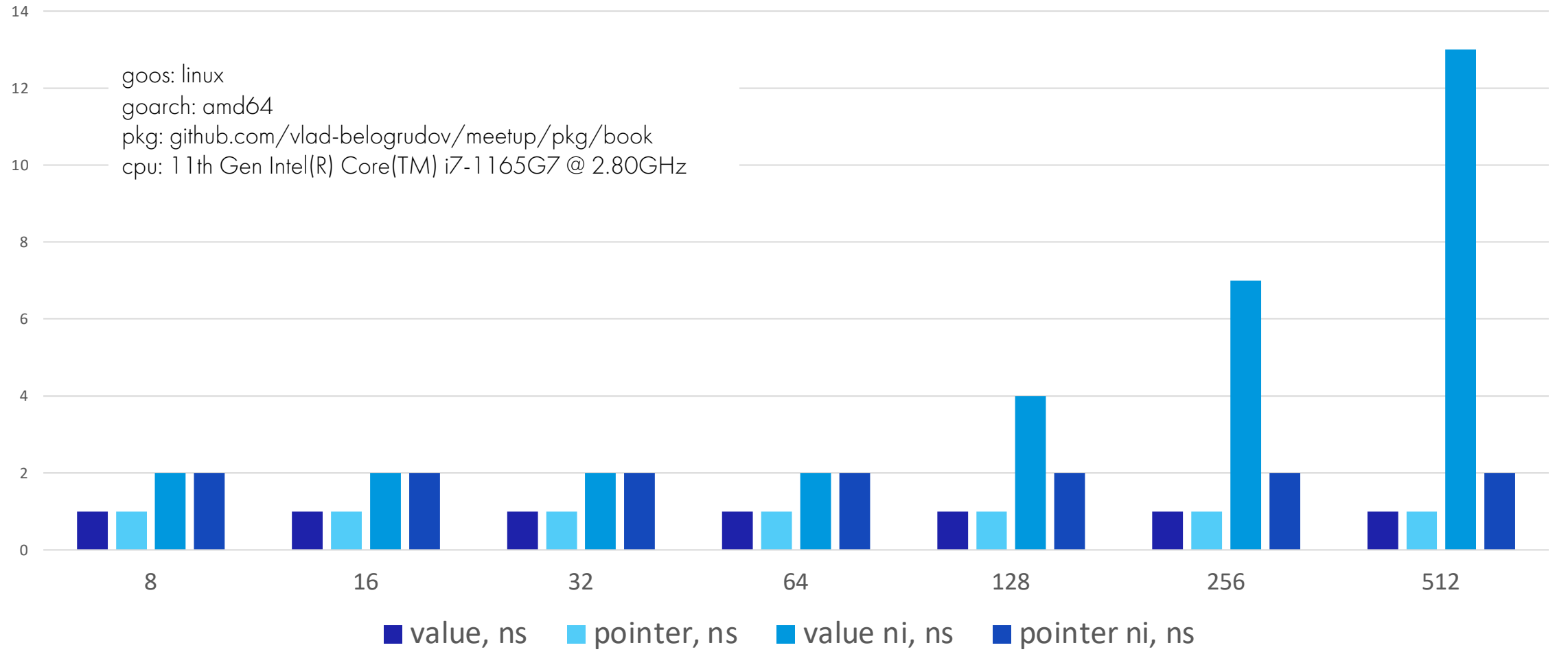
Аргументируем правильно

Когда нужно передавать через указатель:

- Есть необходимость модифицировать входные параметры и избежать дупликатов
- Очень большие данные

```
06    type Book struct {
07        Text [10]byte
08    }
09
10    func ReadBookByValue(book Book, n int) byte {
11        return book.Text[n]
12    }
13
14    func ReadBookByPointer(book *Book, n int) byte {
15        return book.Text[n]
16    }
```

Аргументируем правильно

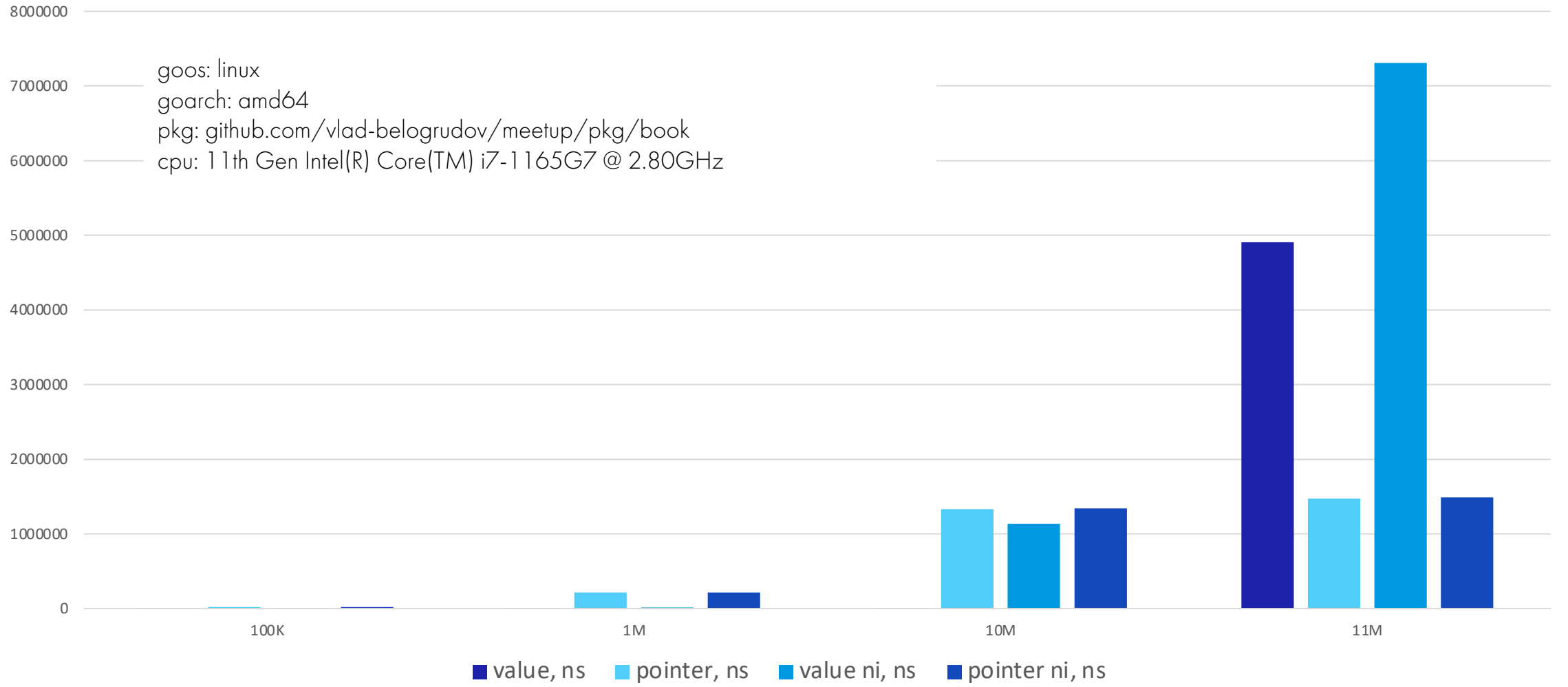


Аргументируем правильно

Что лучше возвращать, указатель или сам объект?

```
05    type Book struct {
06        Name string
07        Text [1000]byte
08    }
09
10    func GetBookByValue() Book {
11        return Book{}
12    }
13
14    func GetBookByPointer() *Book {
15        return new(Book)
16    }
```

Аргументируем правильно



Побег из функций

Стек медленно превращается, превращается стек..

```
05    type Car struct {
06    }
07
09    func BuildCar() *Car {
10        car := Car{}
11        return &car
12    }

// to escape or not to escape, https://pkg.go.dev/cmd/compile/internal/ir#pkg-variables
// MaxStackVarSize = int64(10 * 1024 * 1024)
```

Приемники и звездочки

Звездочка – изменяемый объект

```
05    type Car struct {
06        X float64
07        Y float64
08        Gas int
09    }
10
11    func (c *Car) Move(x, y foat64) { c.X = x; c.Y = y; c.Gas--; }
12
13    func (c *Car) CheckGas() int { return c.Gas; }
14
```

Как пользоваться «ничем»?

nil – значение по умолчанию для «непростых» типов

```
05
06    var s []int
07    var m map[string]bool
08
09    fmt.Println(len(s))
10    fmt.Println(m[“nothing”])
11

// 0
// false
```

```
05    if s != nil {
06        for _, v := range s {
07            ..
08        }
09    if s != nil && len(s) > 0 && s[0] == ‘A’ {
10        ..
11    }
12    if m != nil  {
13        If v, ok := m[“nothing”]; ok { ..

// неидиоматично, вот это словечко!
```

Как пользоваться ничем?

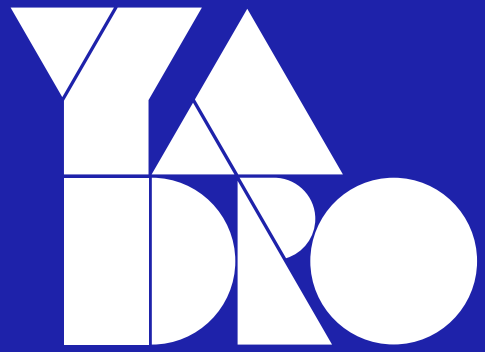
Приемники также могут быть nil

```
05    type Car struct {
06        X, Y float64
07    }
08    func (c *Car) Move(x, y foat64) error {
09        if car == nil {
10            return errors.New(“Приделай мне колеса!”)
11        }
12        c.X, c.Y = x, y
13        return nil
..
..    func main() {
..       var car *Car
..       car.Move(12.345, 6.789) // попытка поехать на еще непостроенной машине
```

При чем здесь программисты Fortran?

OMG!!!
Я ждала этого момента
все выступление!!

https://en.wikiquote.org/wiki/Fortran
https://www.pbm.com/~lindahl/real.programmers.html



Москва,
ул. Родчельская, 15, стр. 13
+7 800 777-06-11

yadro.com