

How to avoid Go gotchas

by learning internals

Ivan Danyliuk, Codemotion Milano

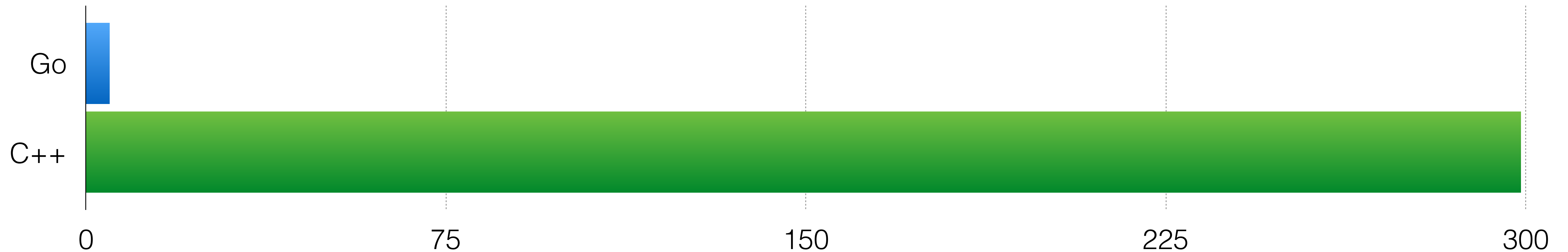
26 Nov 2016

Gotchas

- Go has some gotchas
- Good examples:
 - [50 Shades of Go: Traps, Gotchas, and Common Mistakes for New Golang Devs](#)
 - [Go Traps](#)
 - [Golang slice append gotcha](#)

Gotchas

- Luckily, Go has very few gotchas
- Especially in comparison with other languages



Gotchas

- So, what is gotcha?
- “a gotcha is a **valid construct** in a system, program or programming language that works as documented but **is counter-intuitive** and almost invites mistakes because it is both easy to invoke and unexpected or unreasonable in its outcome”

Gotchas

- Two solutions:
 - “fix” the language
 - fix the intuition.
- Let’s build some intuition to fight gotchas then.

Gotchas

- Let's learn some internals and in memory representations
- It worked for me, should work for you as well.

basic types

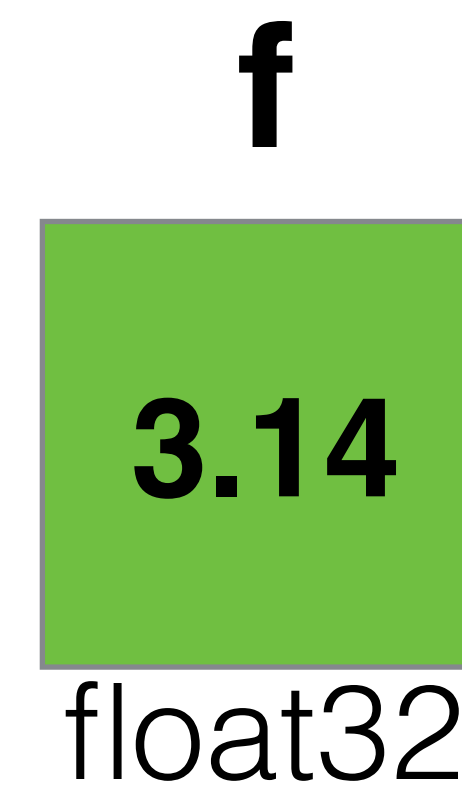
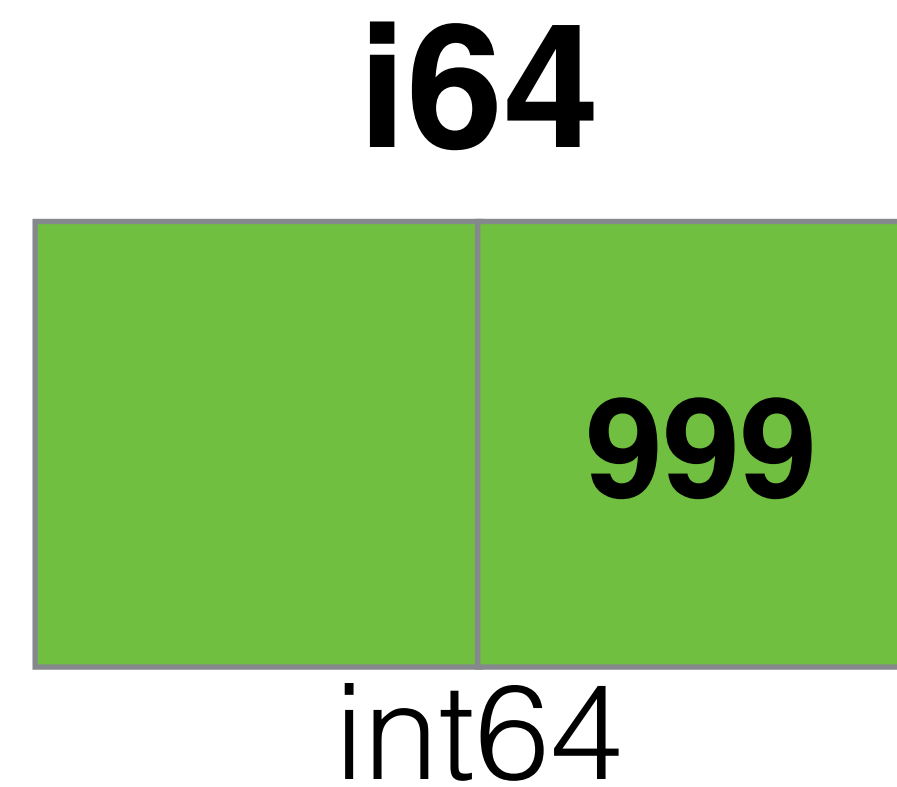
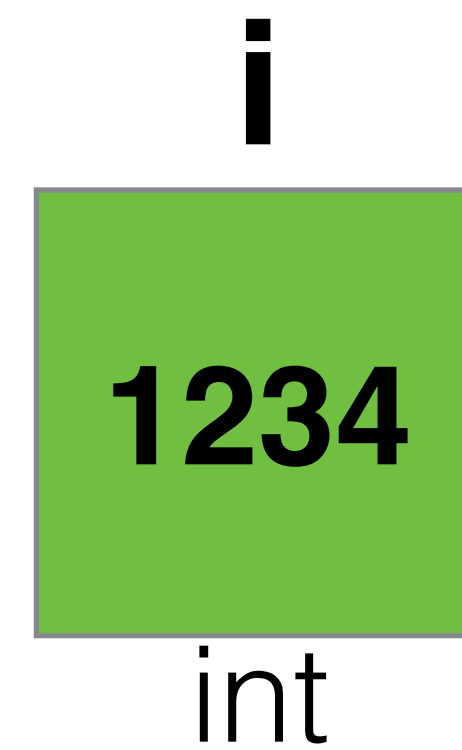
Code:

```
i := 1234  
j := int32(4)  
i64 := int64(999)  
f := float32(3.14)
```

basic types

Code:

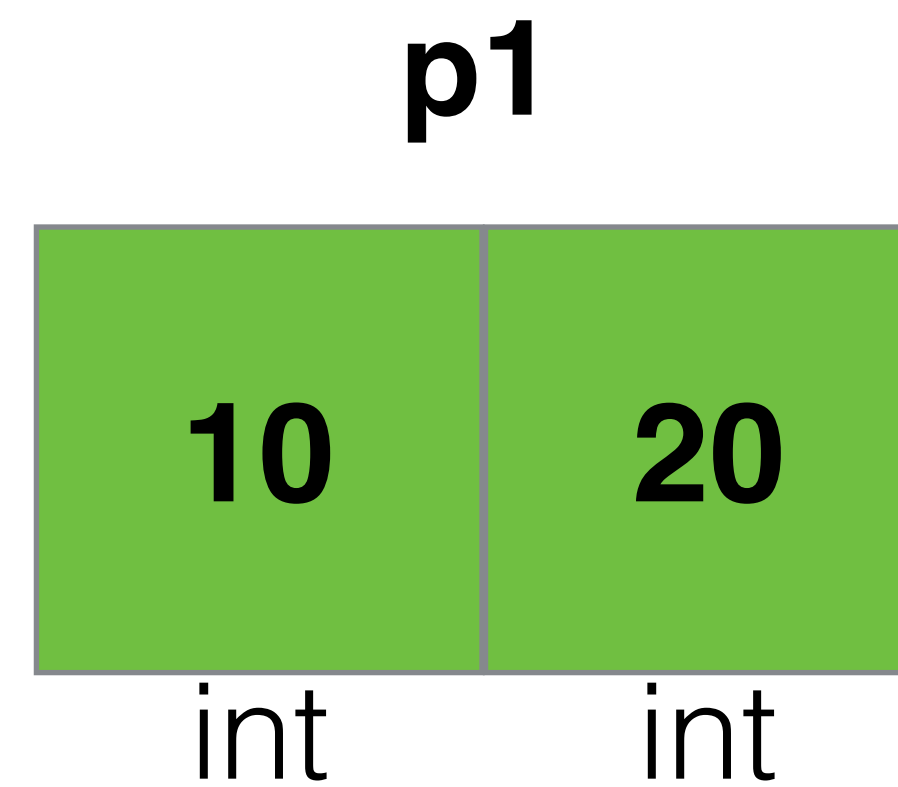
```
i := 1234  
j := int32(4)  
i64 := int64(999)  
f := float32(3.14)
```



structs

Code:

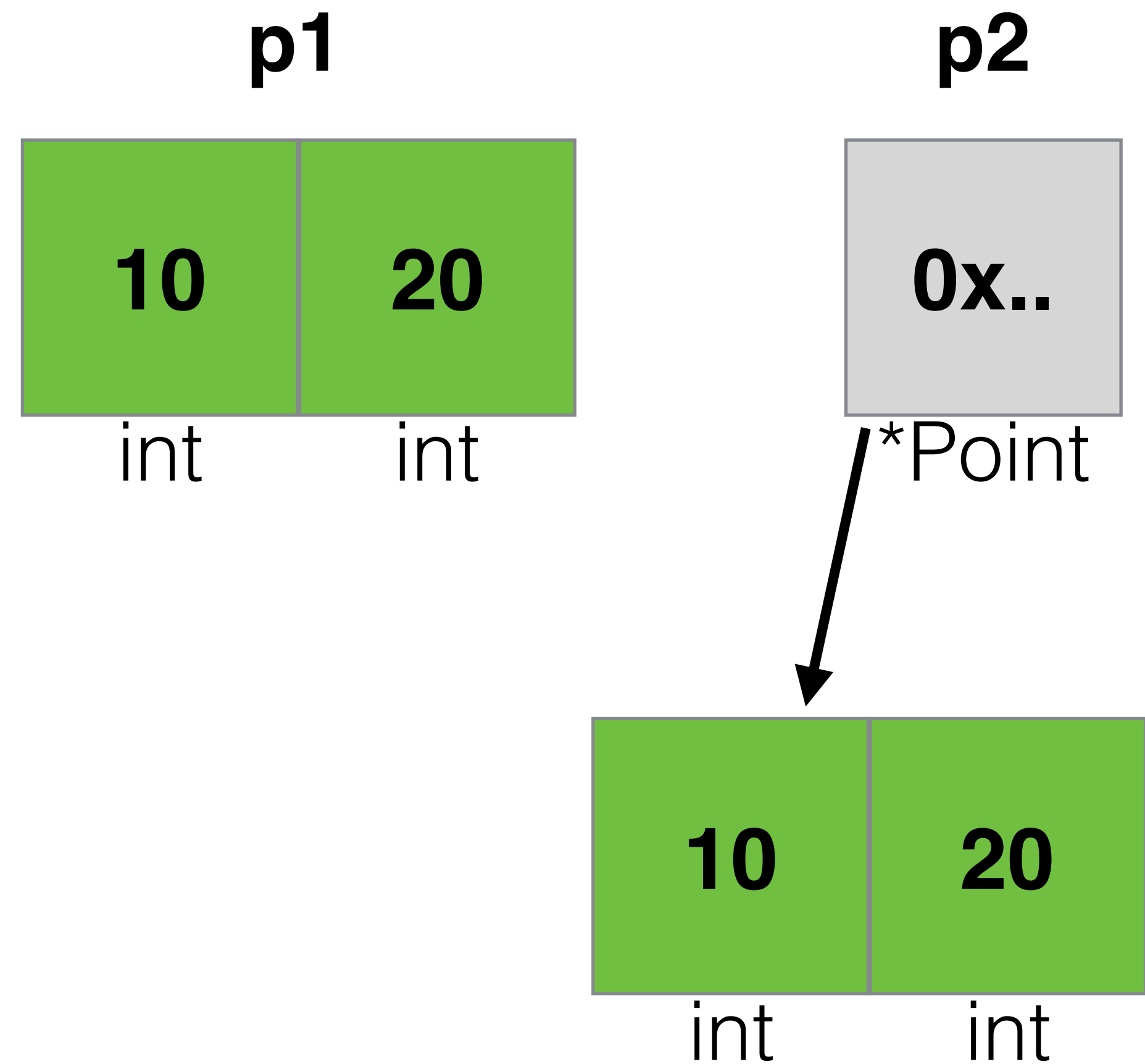
```
type Point struct {  
    X, Y int  
}  
  
p1 := Point{10, 20}
```



basic types

Code:

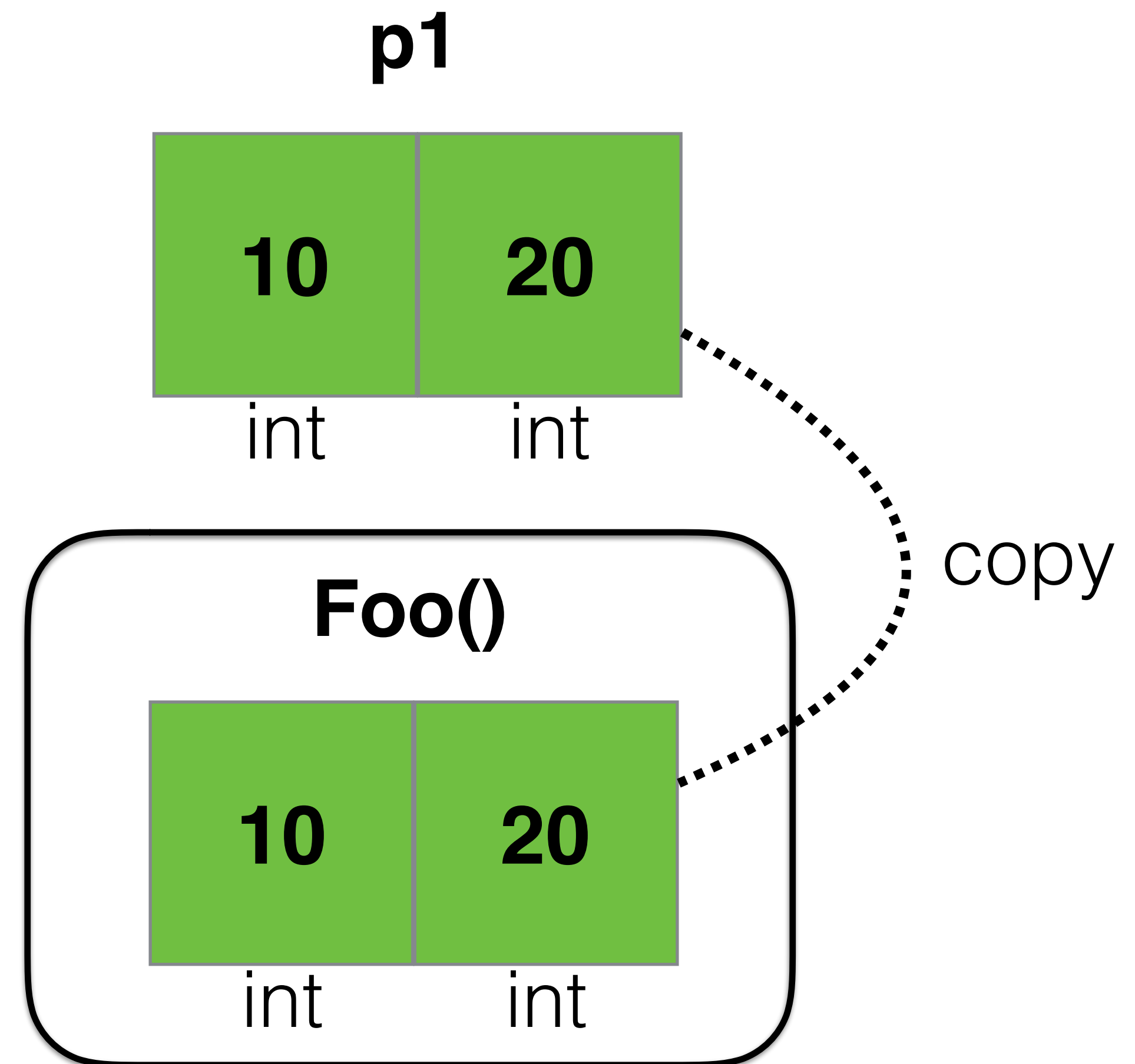
```
type Point struct {  
    X, Y int  
}  
  
p1 := Point{10, 20}  
p2 := &Point{10, 20}
```



structs

Code:

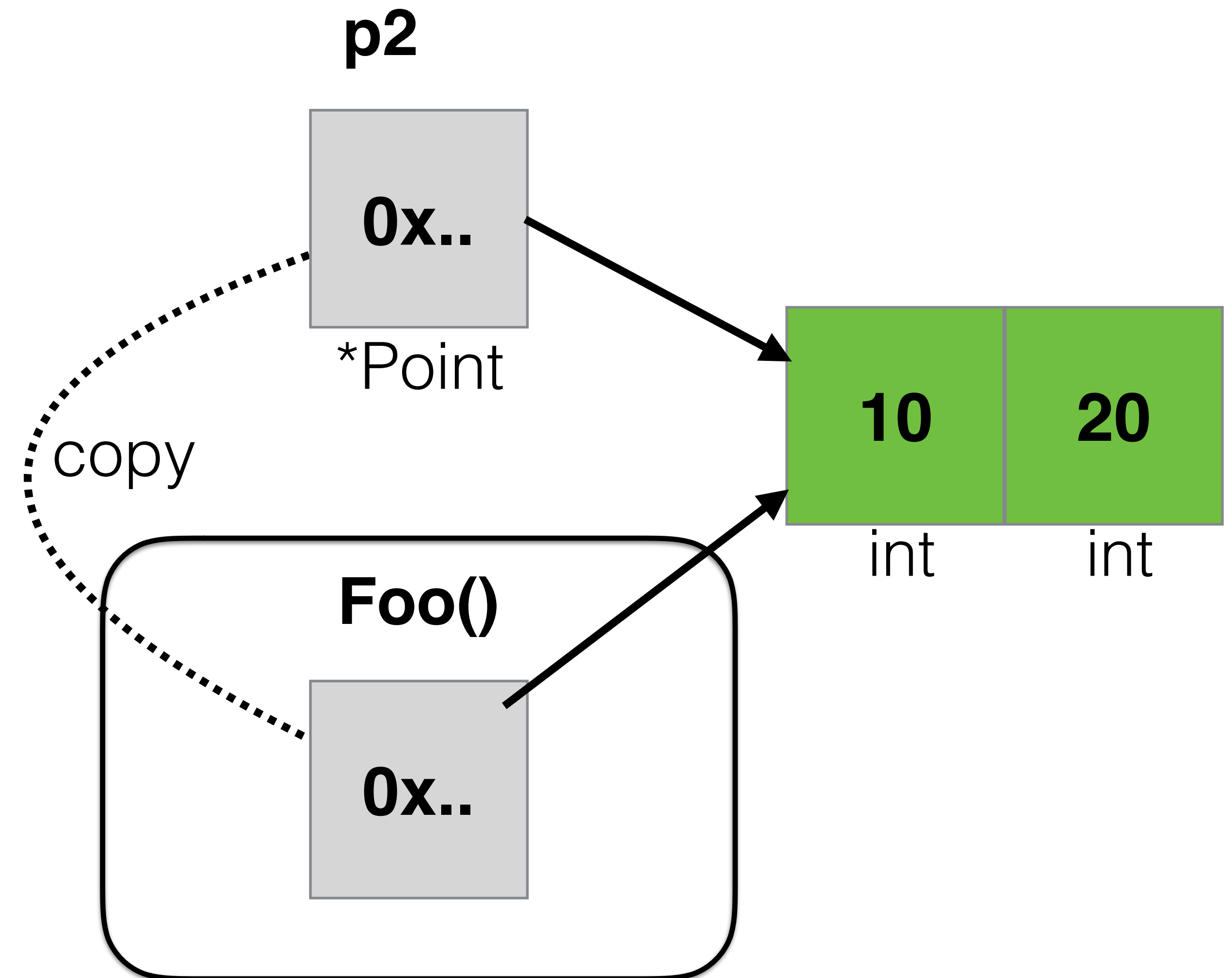
```
func Foo(p Point) {  
    // ...  
}  
  
p1 := Point{10, 20}  
Foo(p1)
```



structs

Code:

```
func Foo(p *Point) {  
    // ...  
}  
  
p2 := &Point{10, 20}  
Foo(p2)
```



array

Code:

```
var arr [5]int
```

array

Code:

```
var arr [5]int
```

Go code: [src/runtime/malloc.go](https://sourcegraph.com/go/src/runtime/malloc.go)

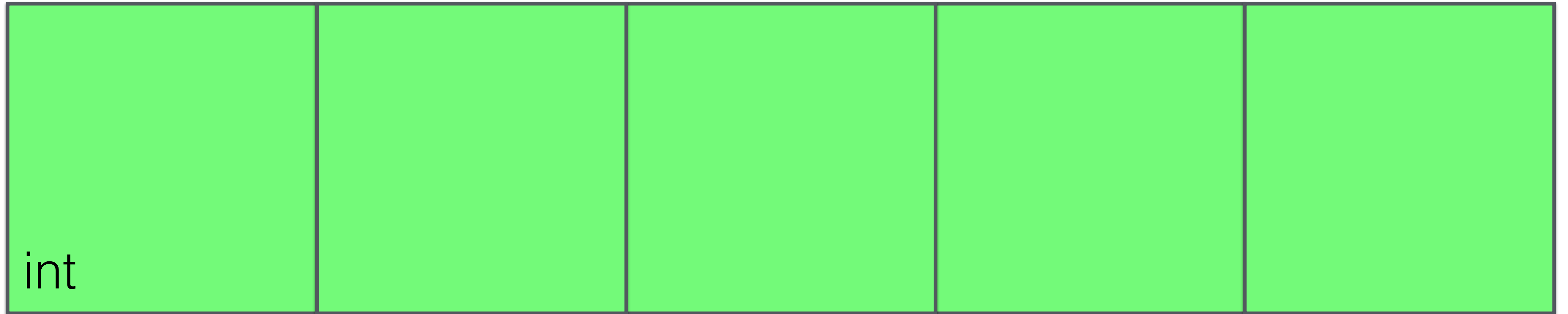
```
// newArray allocates an array of n elements of type typ.
func newArray(typ *_type, n int) unsafe.Pointer {
    if n < 0 || uintptr(n) > maxSliceCap(typ.size) {
        panic(plainError("runtime: allocation size out of range"))
    }
    return mallocgc(typ.size*uintptr(n), typ, true)
}
```

array

Code:

```
var arr [5]int
```

Memory:



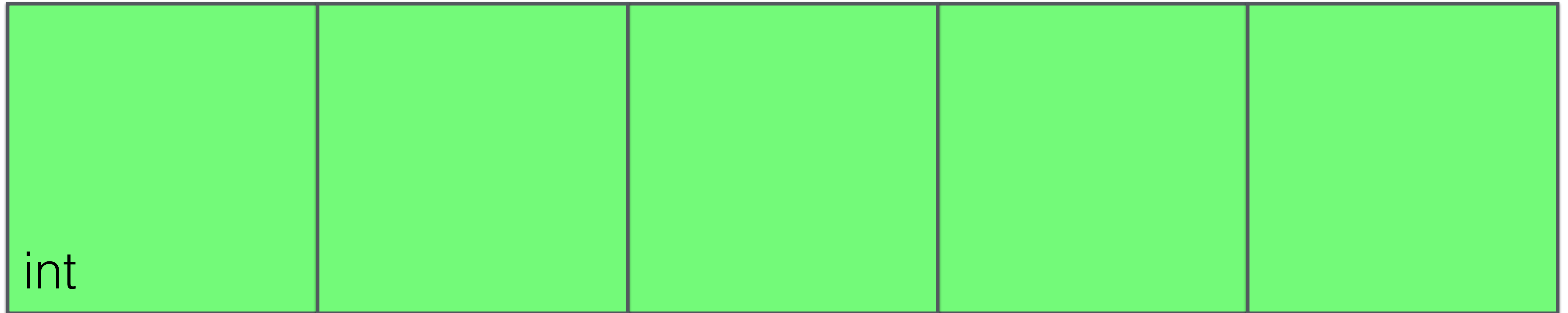
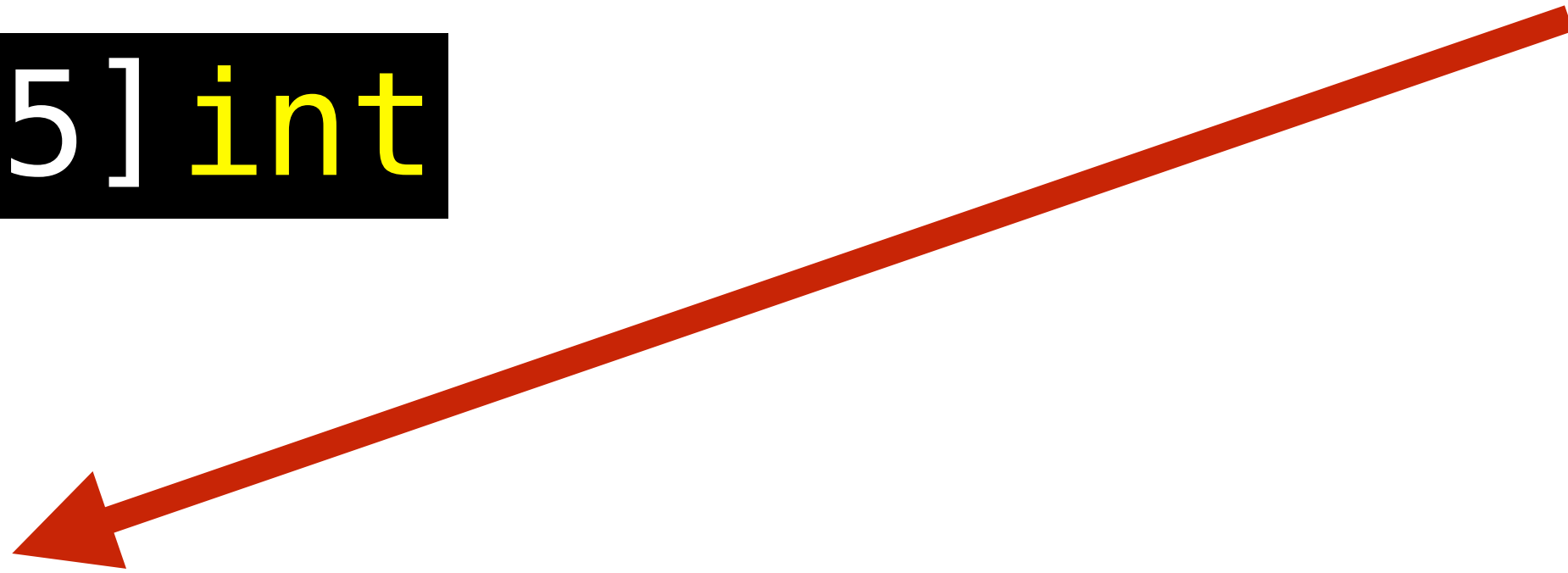
array

Code:

```
var arr [5]int
```

**4 or 8 bytes blocks
(32 or 64 bit arch)**

Memory:

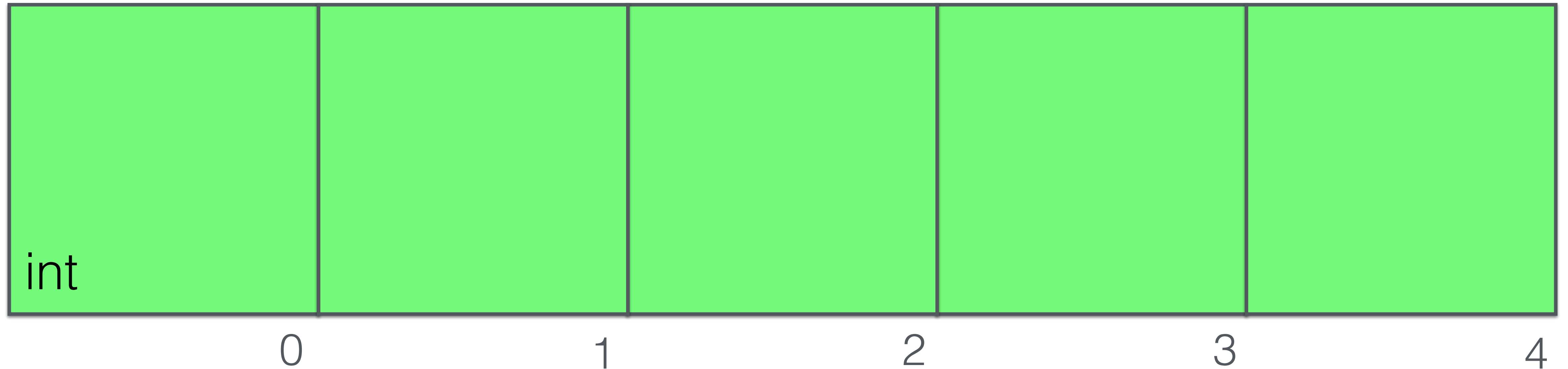


array

Code:

```
var arr [5]int
```

Memory:

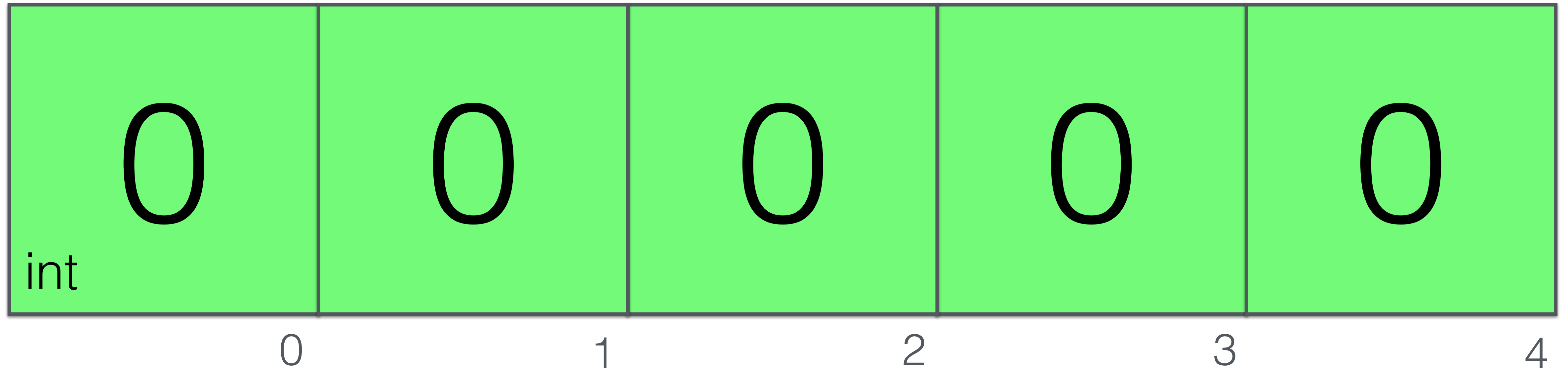


array

Code:

```
var arr [5]int
```

Memory:

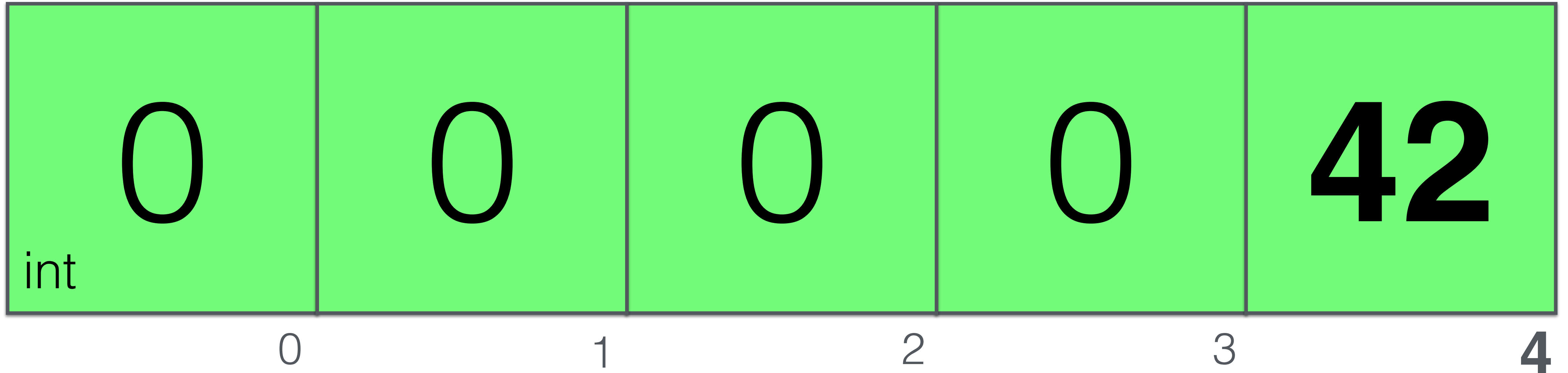


array

Code:

```
var arr [5] int  
arr[4] = 42
```

Memory:



slice

Code:

```
var foo []int
```

slice

Code:

```
var foo []int
```

Go code: [src/runtime/slice.go](https://golang.org/src/runtime/slice.go)

```
type slice struct {  
    array unsafe.Pointer  
    len   int  
    cap   int  
}
```

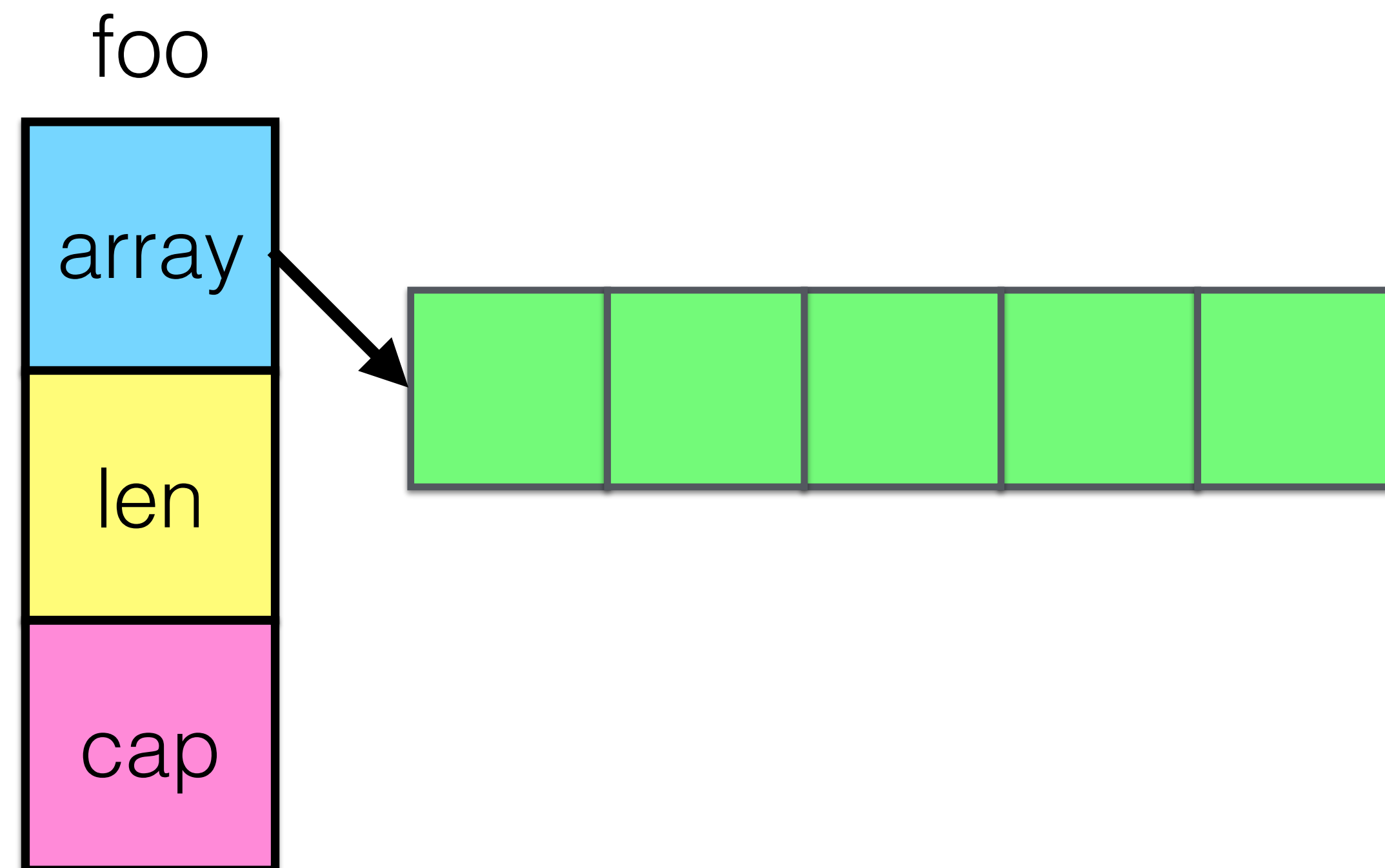
slice

Code:

```
var foo []int
```

Go code: [src/runtime/slice.go](https://golang.org/src/runtime/slice.go)

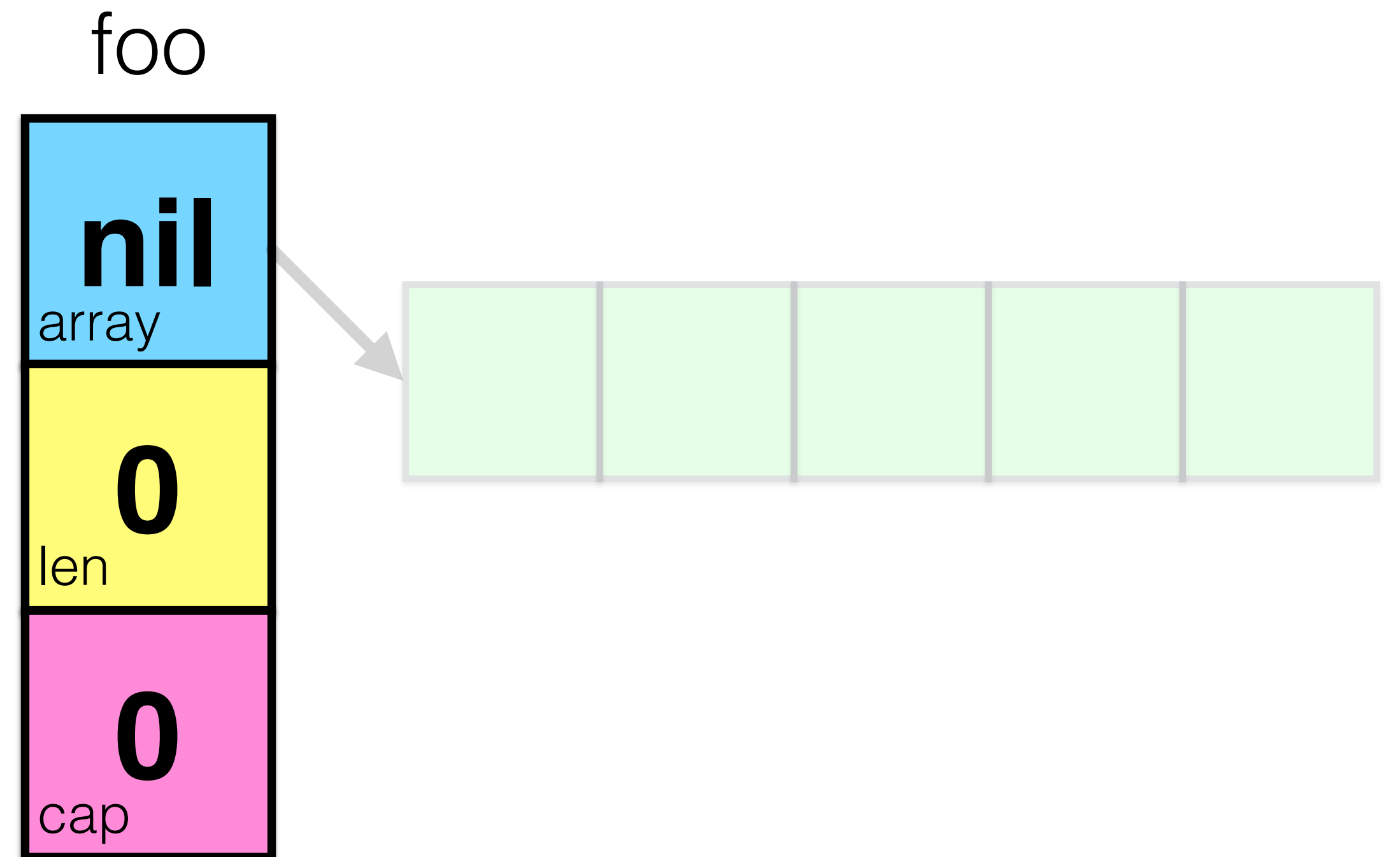
```
type slice struct {  
    array unsafe.Pointer  
    len   int  
    cap   int  
}
```



slice

Code:

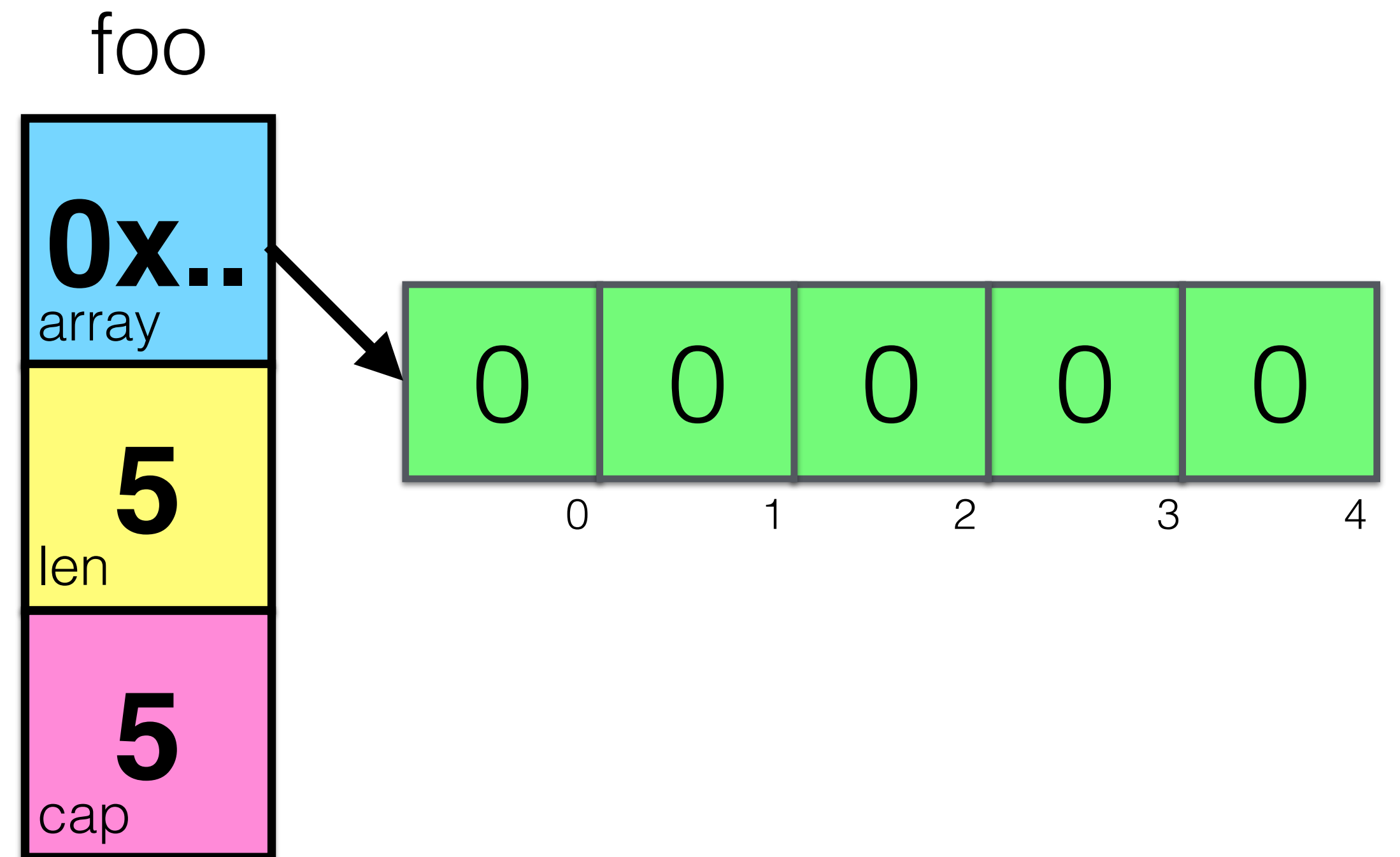
```
var foo []int
```



slice

Code:

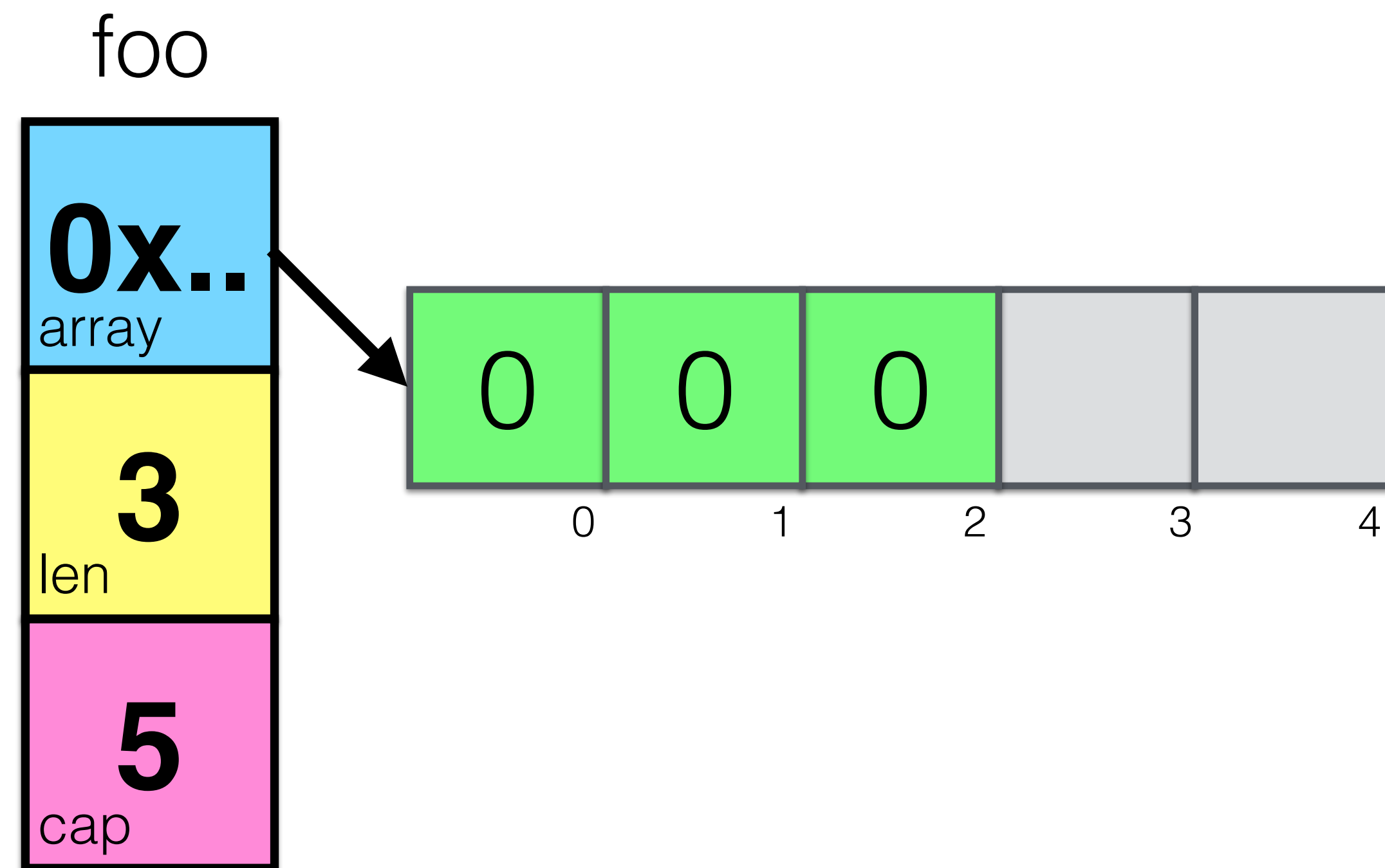
```
var foo []int  
foo = make([]int, 5)
```



slice

Code:

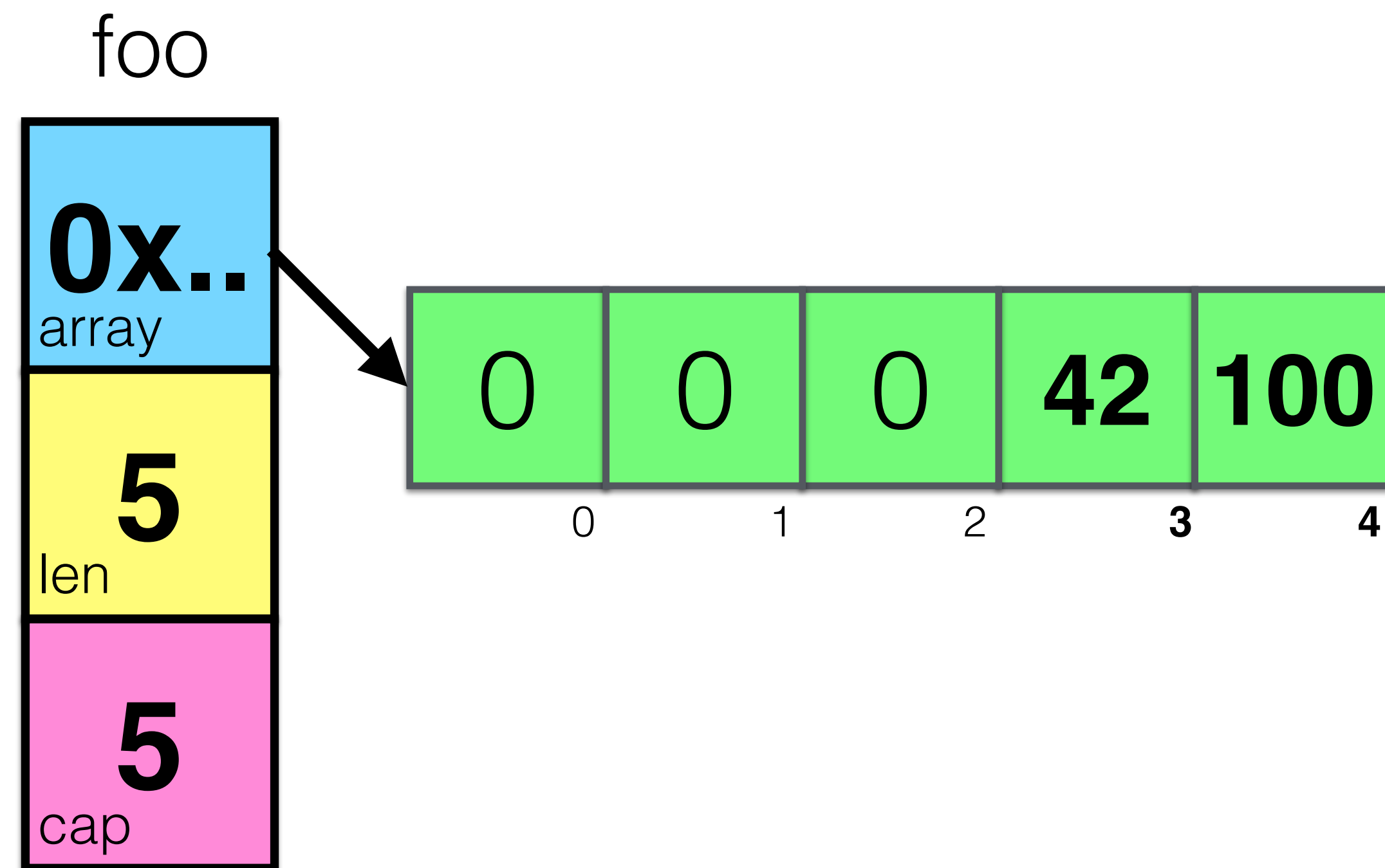
```
var foo []int  
foo = make([]int, 3, 5)
```



slice

Code:

```
var foo []int
foo = make([]int, 5)
foo[3] = 42
foo[4] = 100
```

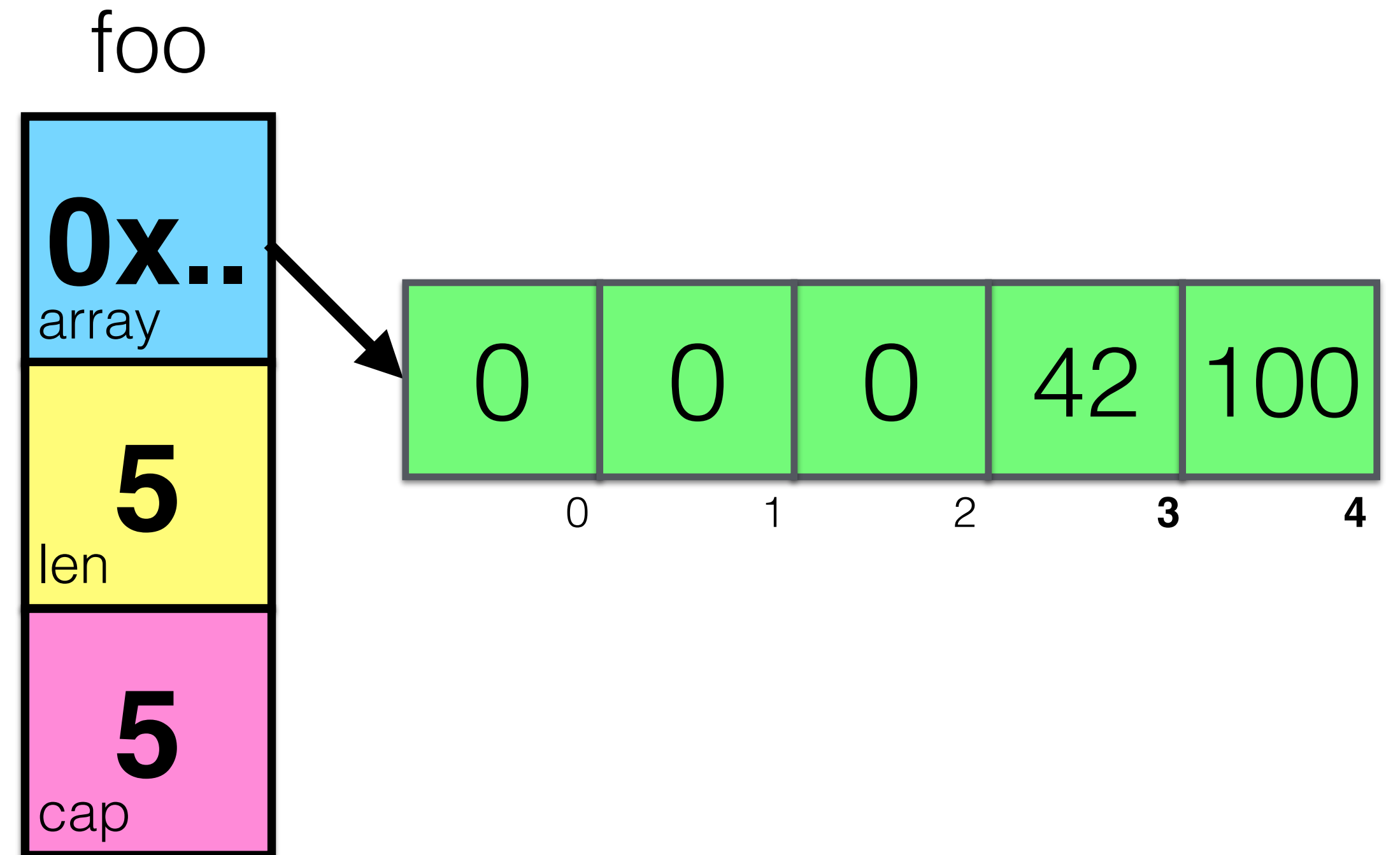


slice

Code:

```
var foo []int
foo = make([]int, 5)
foo[3] = 42
foo[4] = 100

bar := foo[1:4]
```

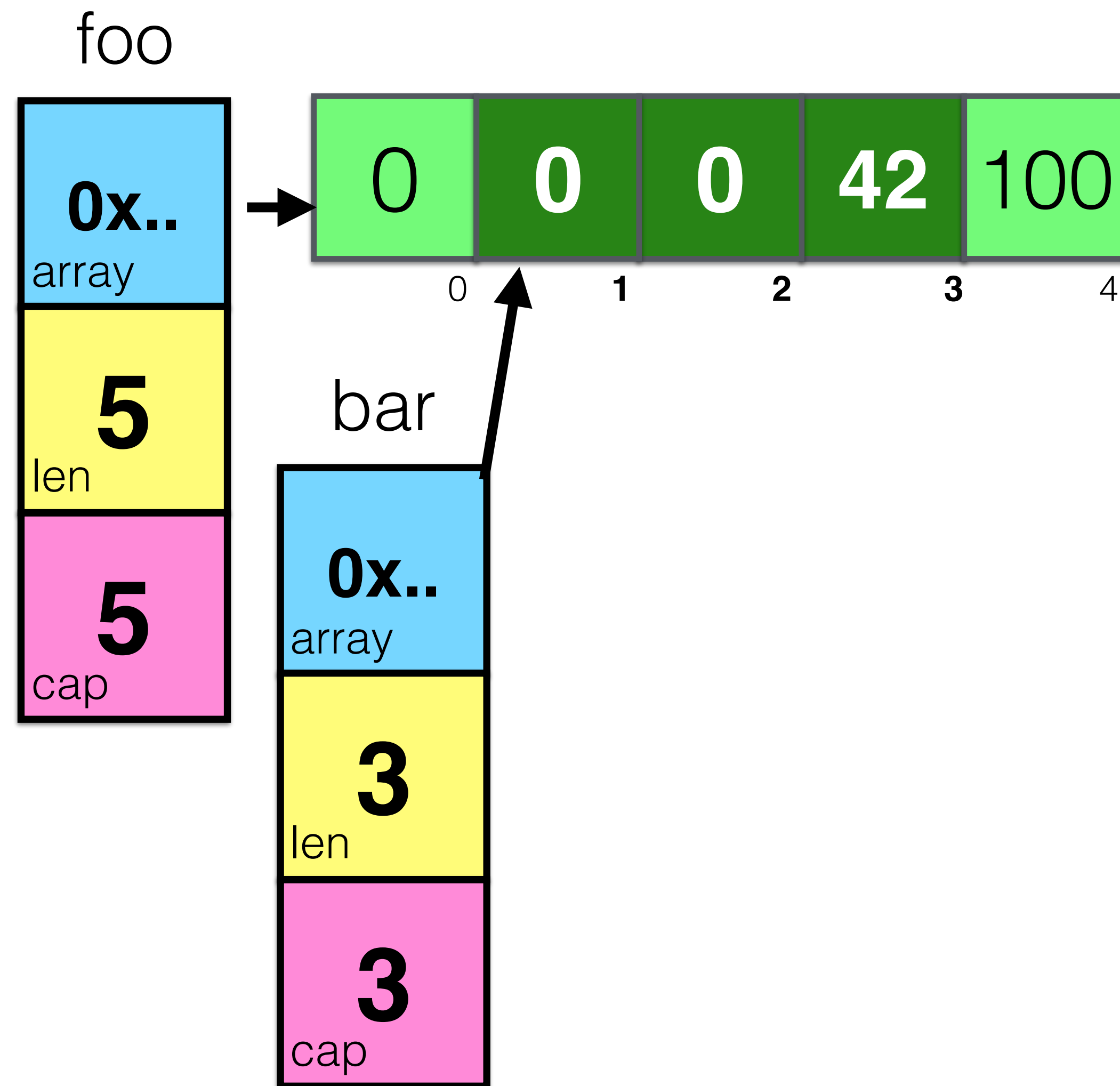


slice

Code:

```
var foo []int
foo = make([]int, 5)
foo[3] = 42
foo[4] = 100

bar := foo[1:4]
```

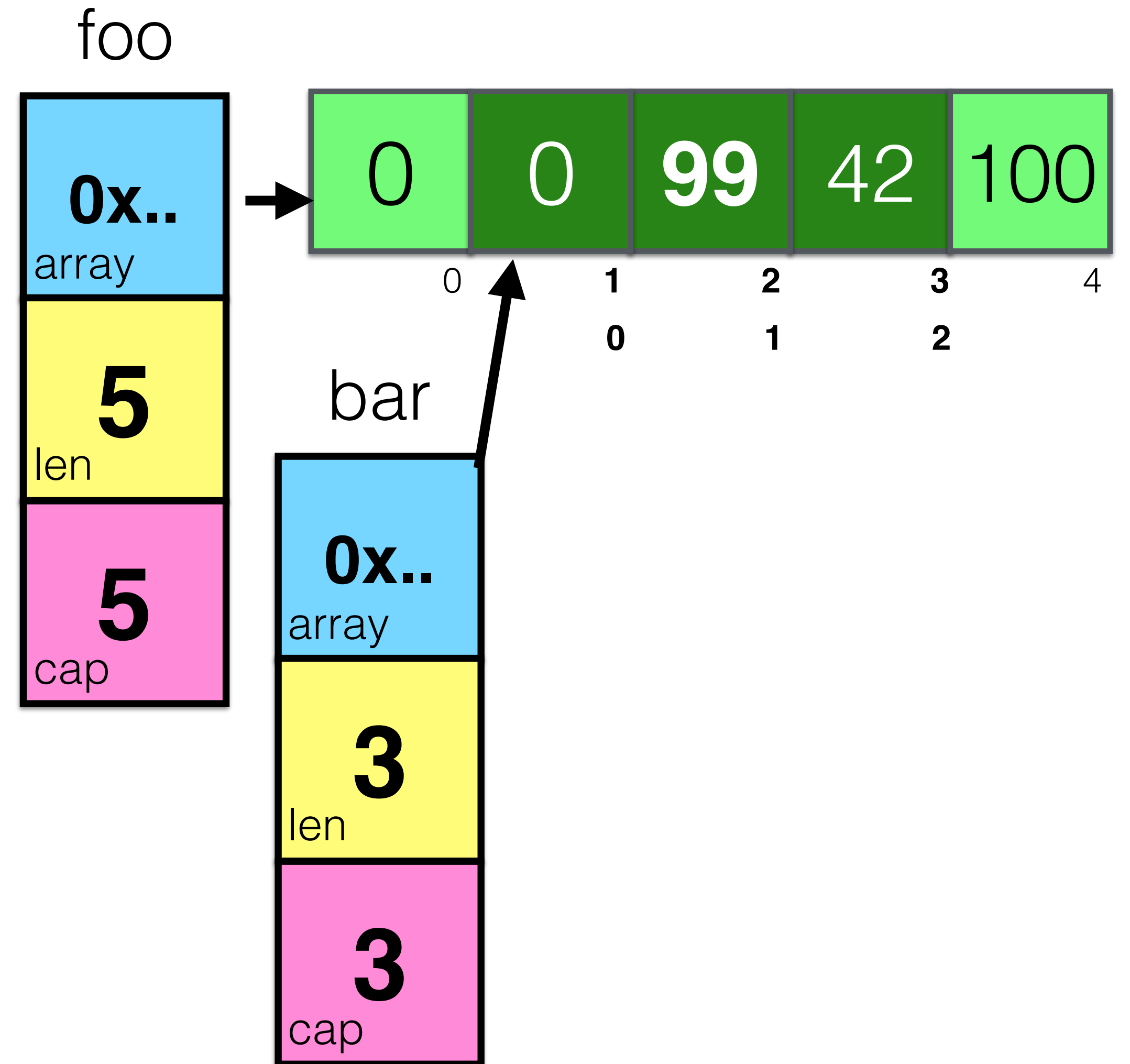


slice

Code:

```
var foo []int
foo = make([]int, 5)
foo[3] = 42
foo[4] = 100

bar := foo[1:4]
bar[1] = 99
```



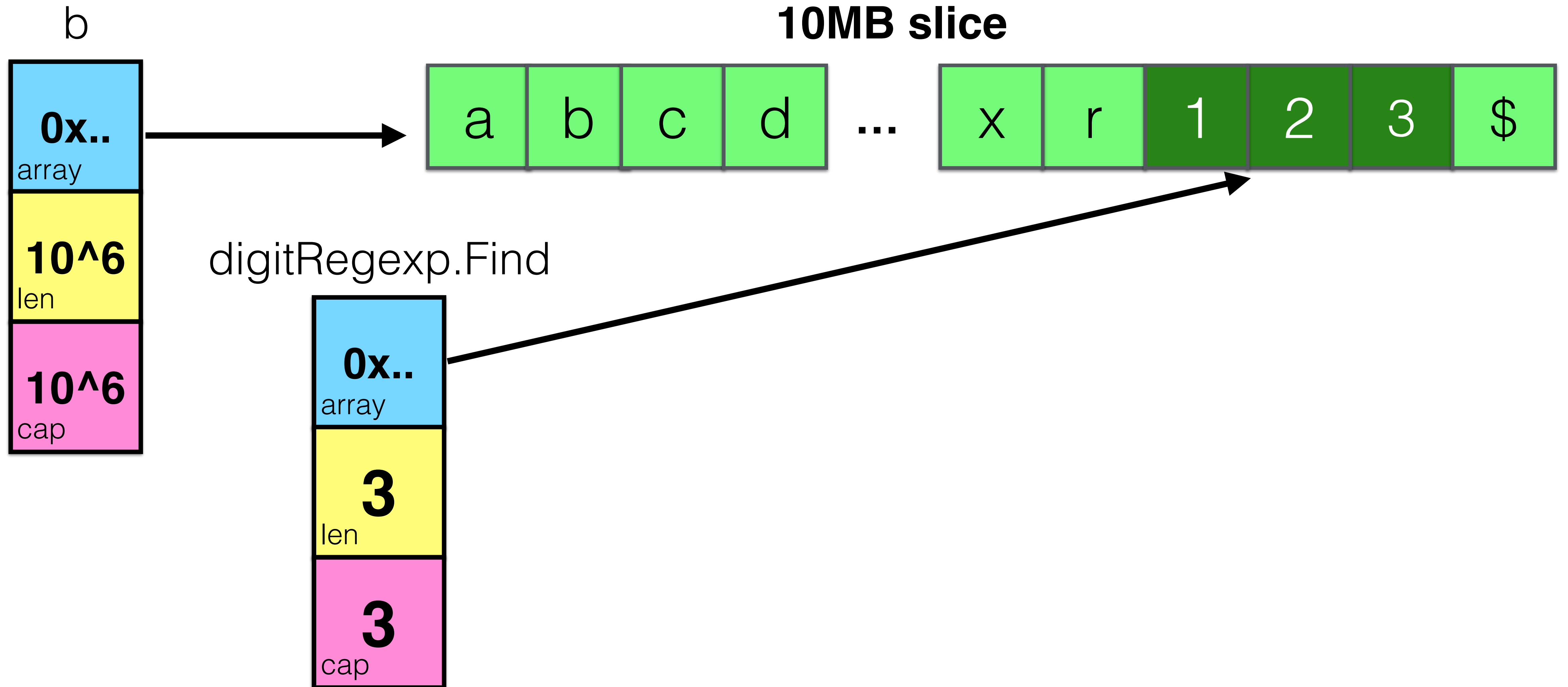
slice

Code:

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    return digitRegexp.Find(b)
}
```

slice



append

Code:

```
a := make([]int, 32)  
a = append(a, 1)
```


append

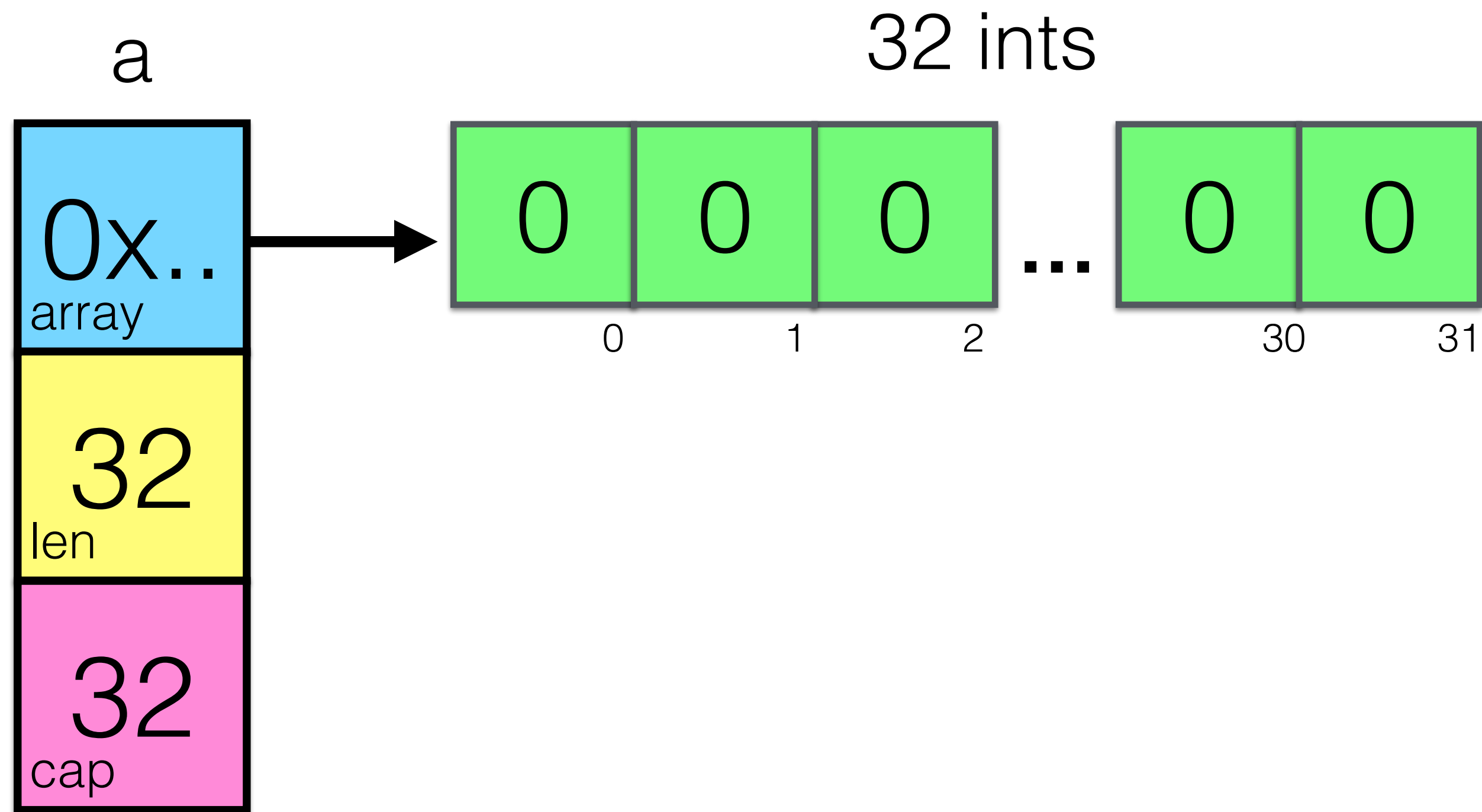
Code:

```
a := make([]int, 32)
a = append(a, 1)
fmt.Println("len:", len(b), "cap:", cap(b))
```

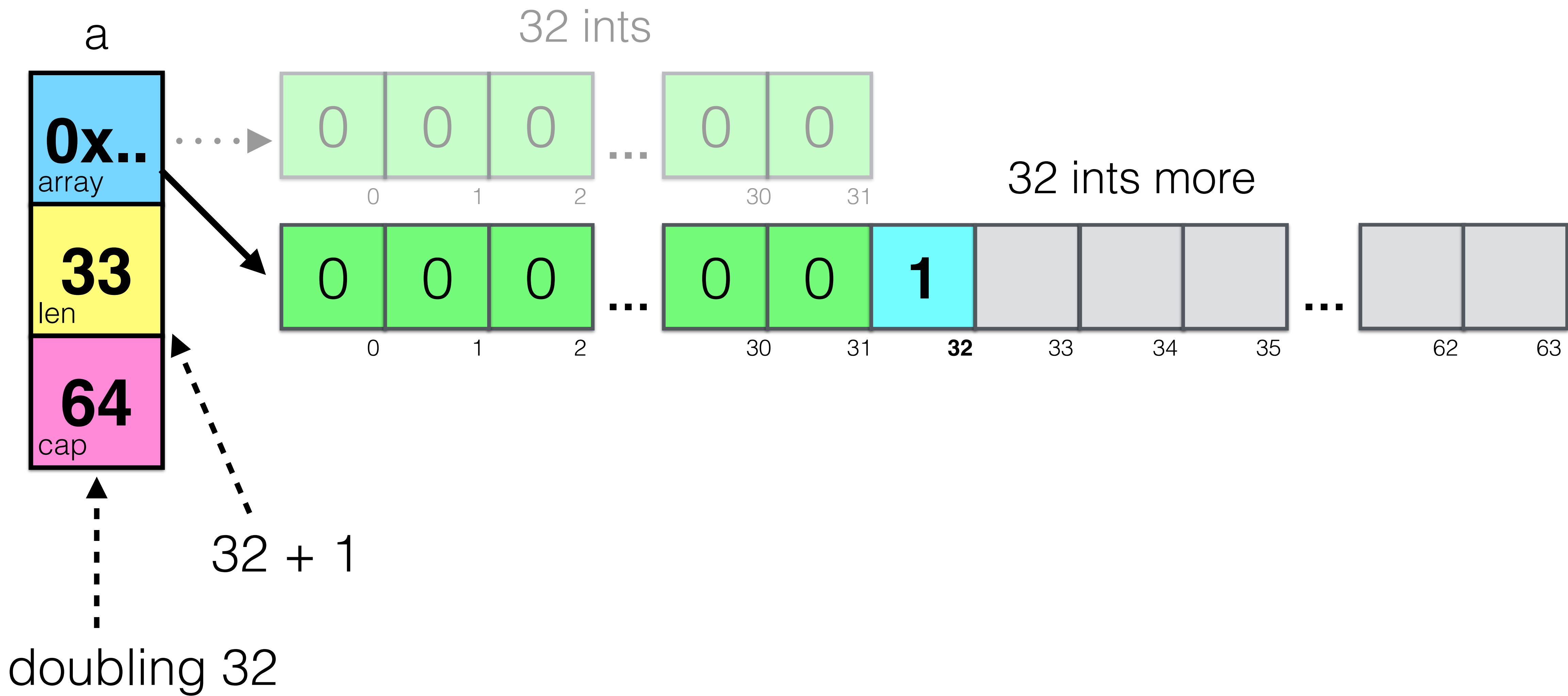
Output:

```
len: 33 cap: 64
```

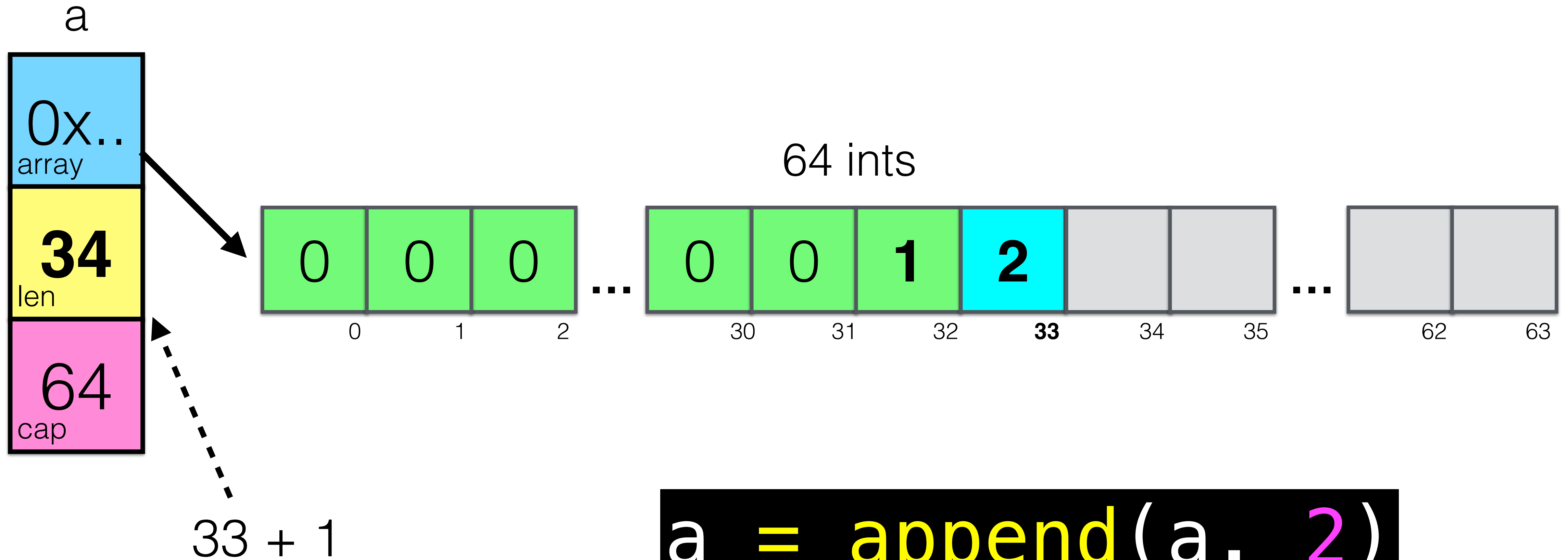
append



```
a = append(a, 1)
```



append



```
a = append(a, 2)
```

interfaces

Code:

```
type error interface {  
    Error() string  
}
```

interfaces

Code:

```
type error interface {  
    Error() string  
}
```

Go code: [src/runtime/runtime2.go](https://golang.org/src/runtime/runtime2.go)

```
type iface struct {  
    tab *itab  
    data unsafe.Pointer  
}
```

interfaces

Code:

```
type error interface {  
    Error() string  
}
```

Go code: [src/runtime/runtime2.go](https://sourcegraph.com/go/src/runtime/runtime2.go)

```
type iface struct {  
    tab *itab  
    data unsafe.Pointer  
}
```

itab = interface table



interfaces

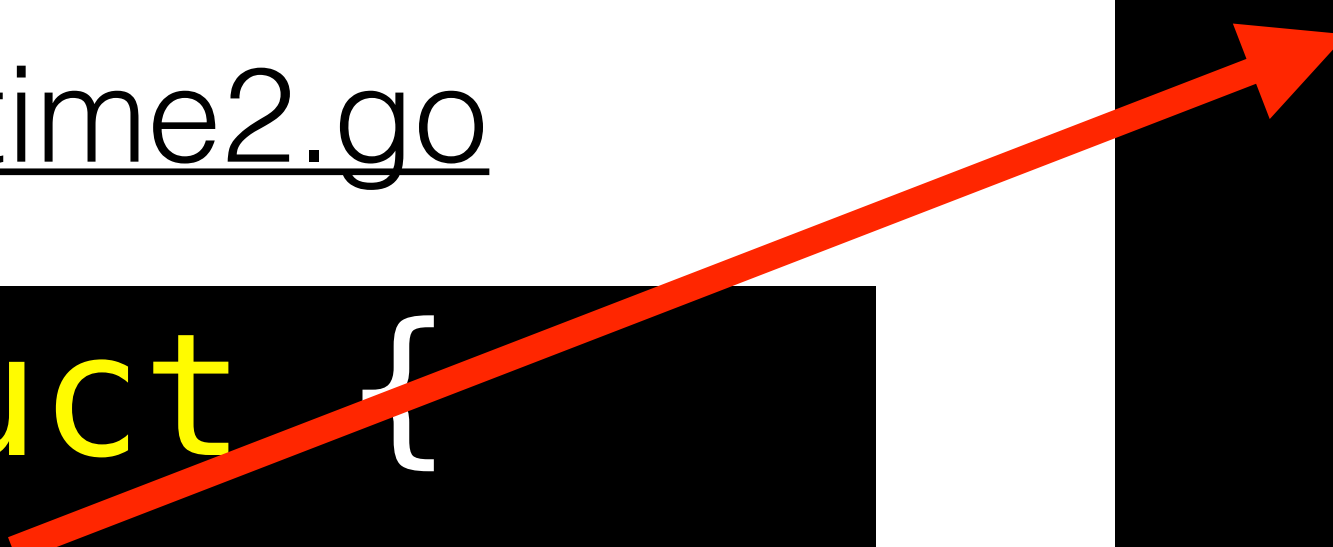
Code:

```
type error interface {  
    Error() string  
}
```

Go code: [src/runtime/runtime2.go](https://golang.org/src/runtime/runtime2.go)

```
type iface struct {  
    tab *itab  
    data unsafe.Pointer  
}
```

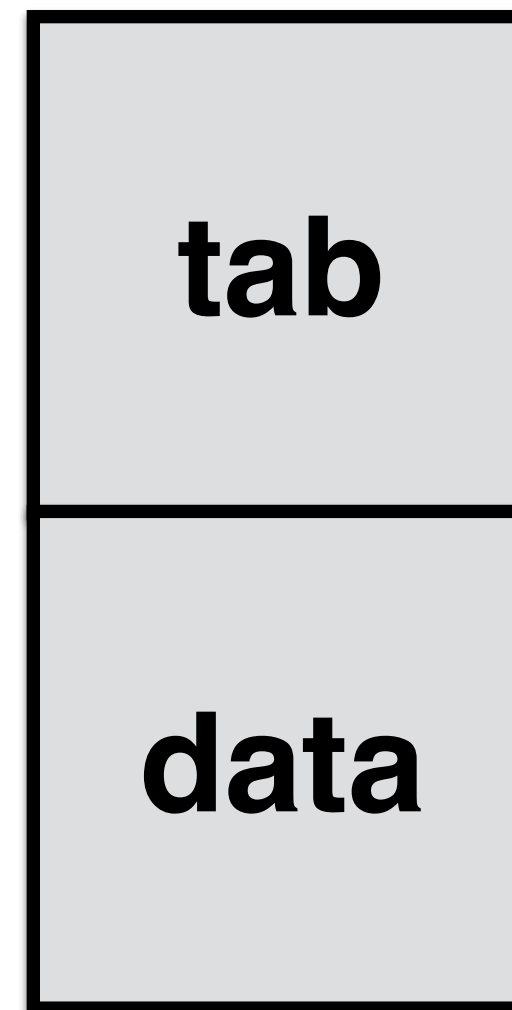
```
type itab struct {  
    inter *interfacetype  
    _type *_type  
    link *itab  
    bad int32  
    unused int32  
    fun [1]uintptr  
}
```



interfaces

Code:

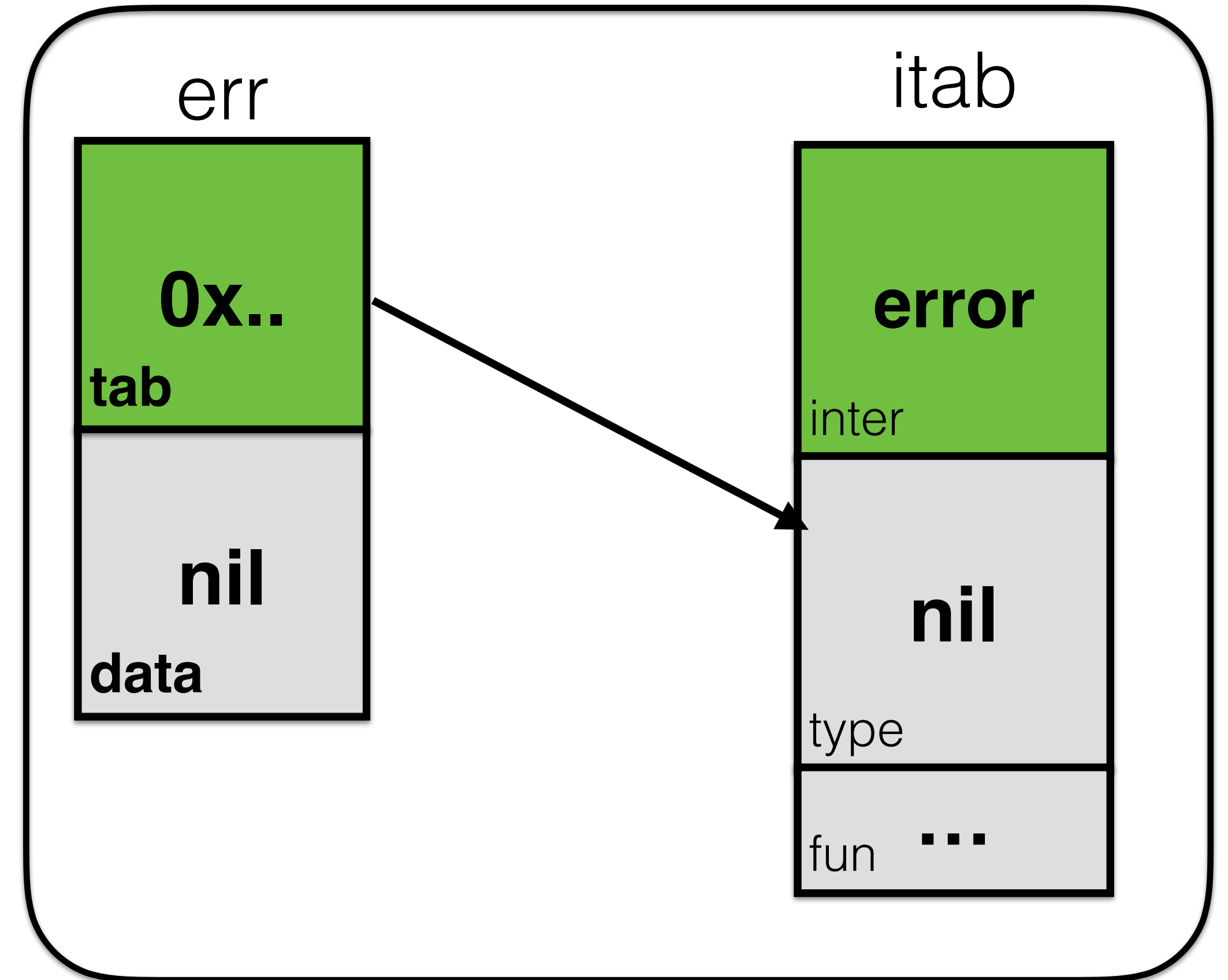
```
type error interface {  
    Error() string  
}
```



interfaces

Code:

```
type error interface {  
    Error() string  
}  
  
var err error
```

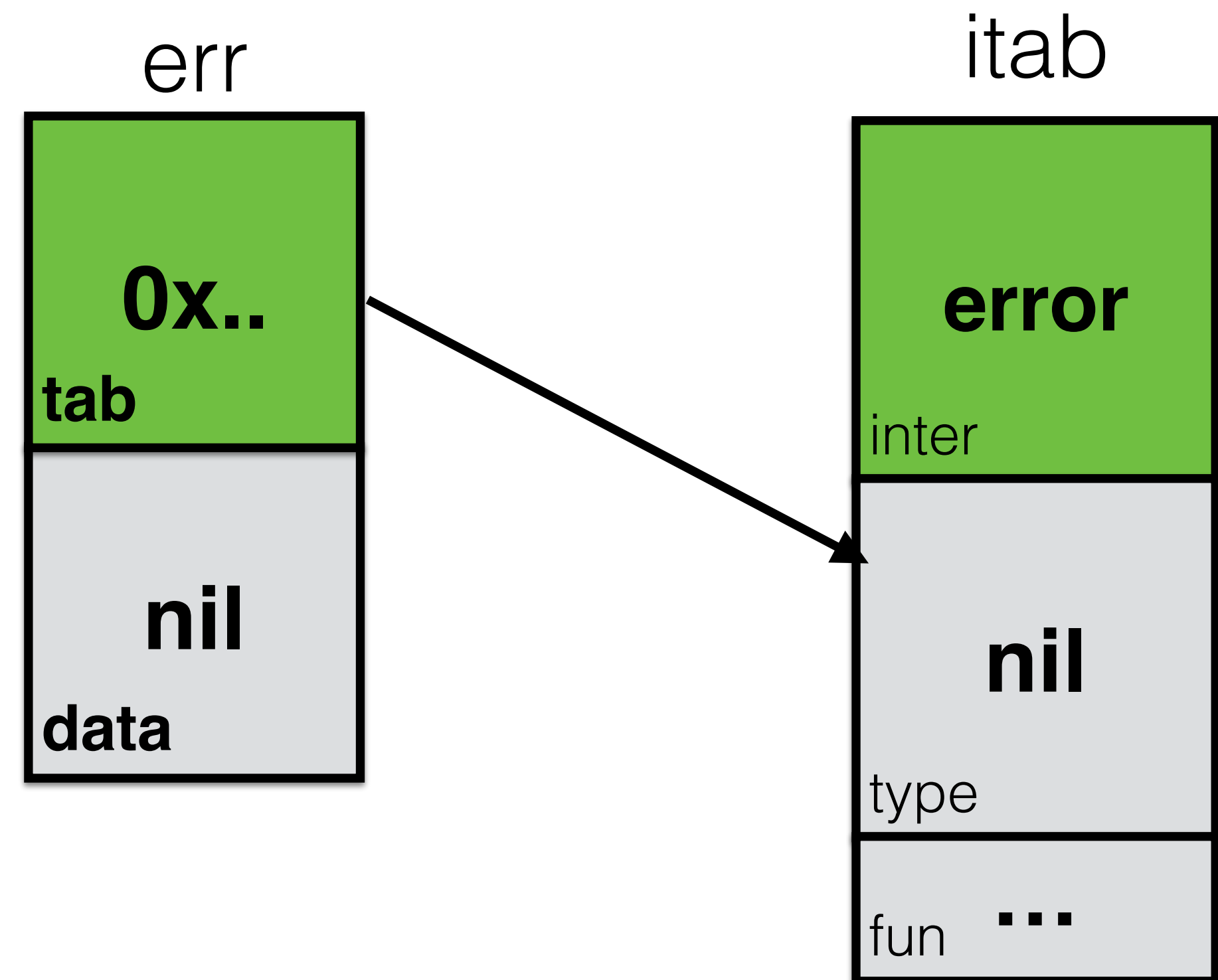


nil interface →

interfaces

Code:

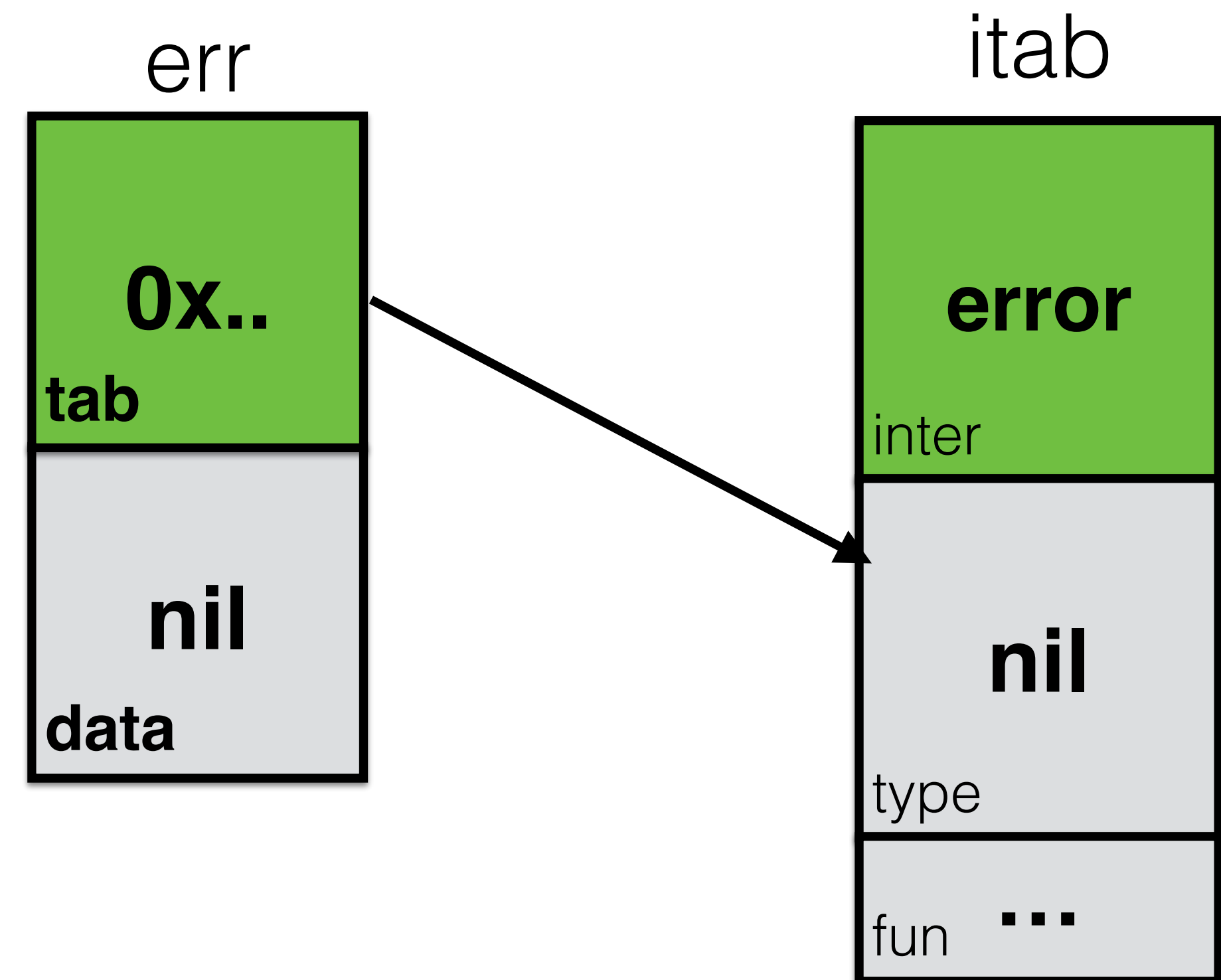
```
type error interface {  
    Error() string  
}  
  
func foo() error {  
    return nil  
}
```



interfaces

Code:

```
func foo() error {  
    var err error  
    // err == nil  
    return err  
}  
  
err := foo()  
if err != nil { // false  
}
```



interfaces

Code:

```
func foo() error {  
    var err *os.PathError  
    // err == nil  
    return err  
}  
  
err := foo()  
if err != nil { // ???  
}
```

interfaces

Code:

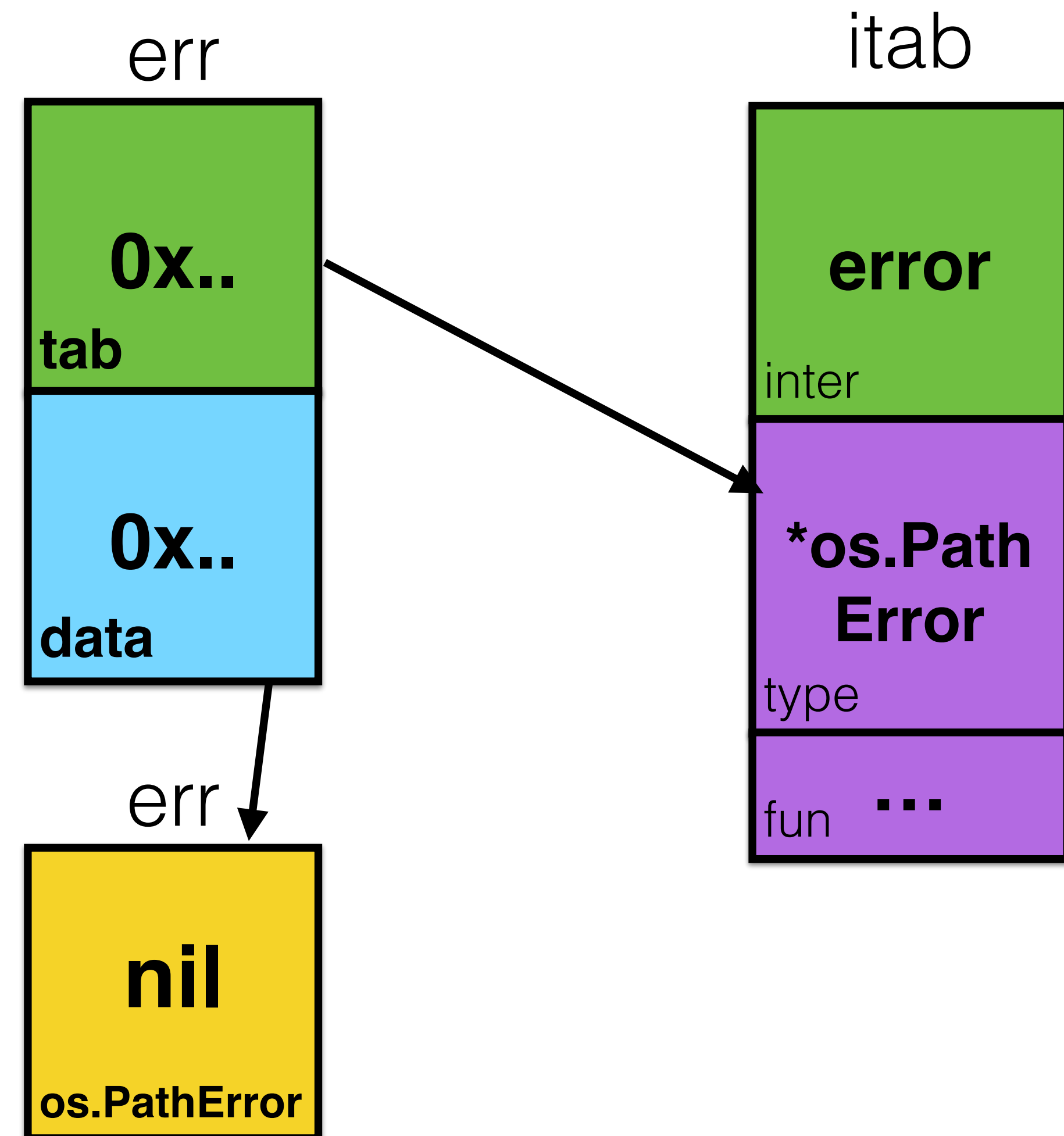
```
func foo() error {  
    var err *os.PathError  
    // err == nil  
    return err  
}  
  
err := foo()  
if err != nil { // true  
}
```

interfaces

Code:

```
func foo() error {  
    var err *os.PathError  
    // err == nil  
    return err  
}
```

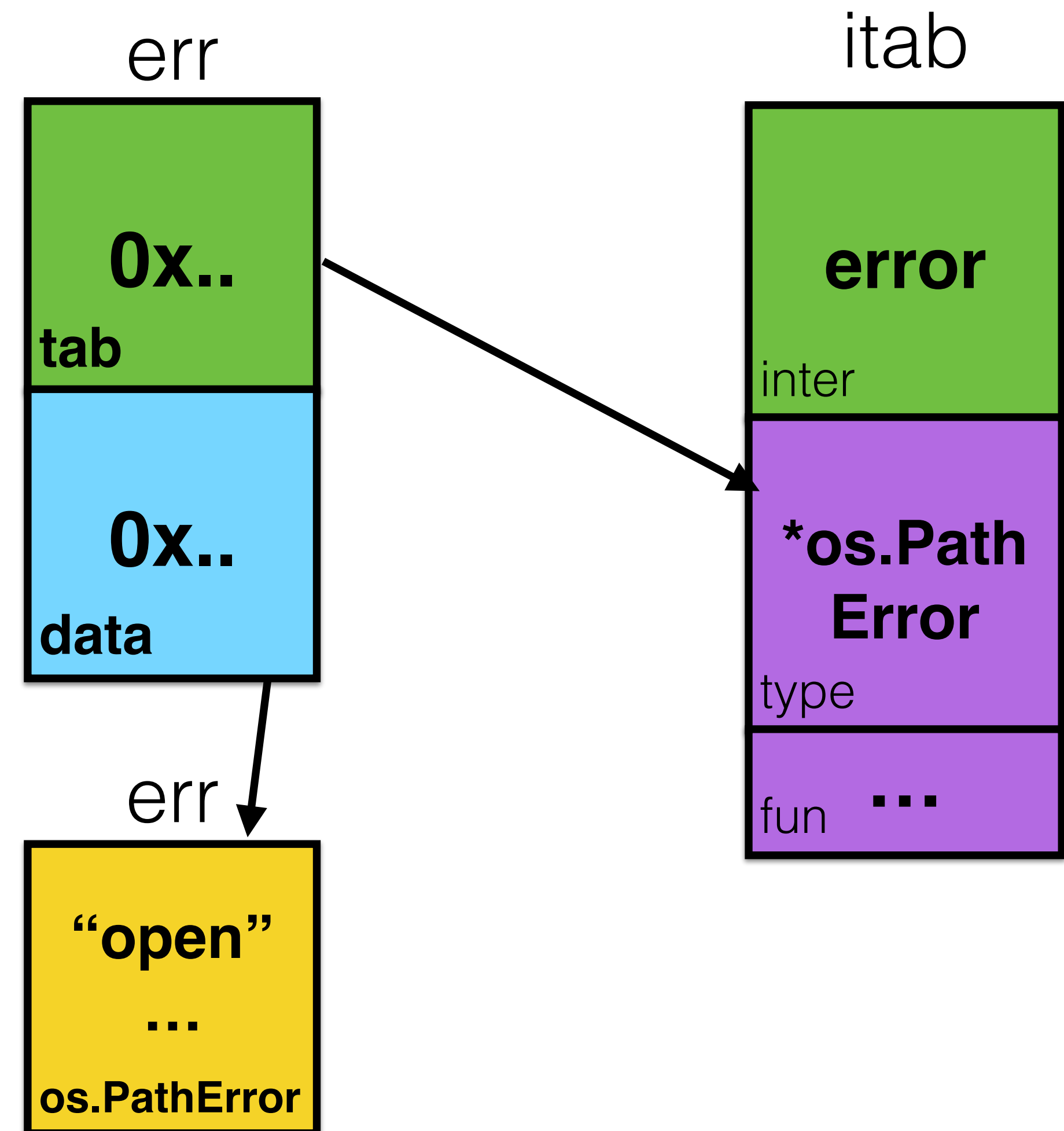
```
err := foo()  
if err != nil { // true  
}
```



interfaces

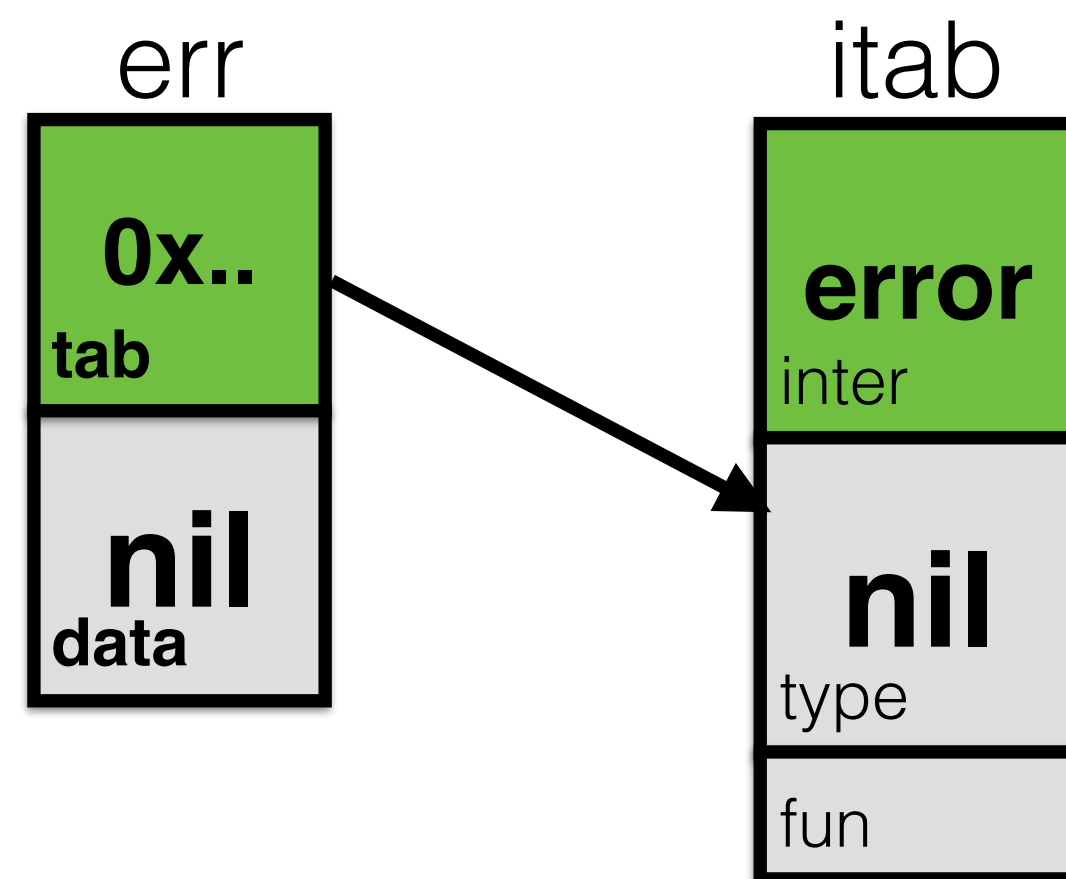
Code:

```
func foo() error {  
    err := &os.PathError{  
        "open", name, e  
    }  
    return err  
}  
  
err := foo()  
if err != nil { // true  
}
```

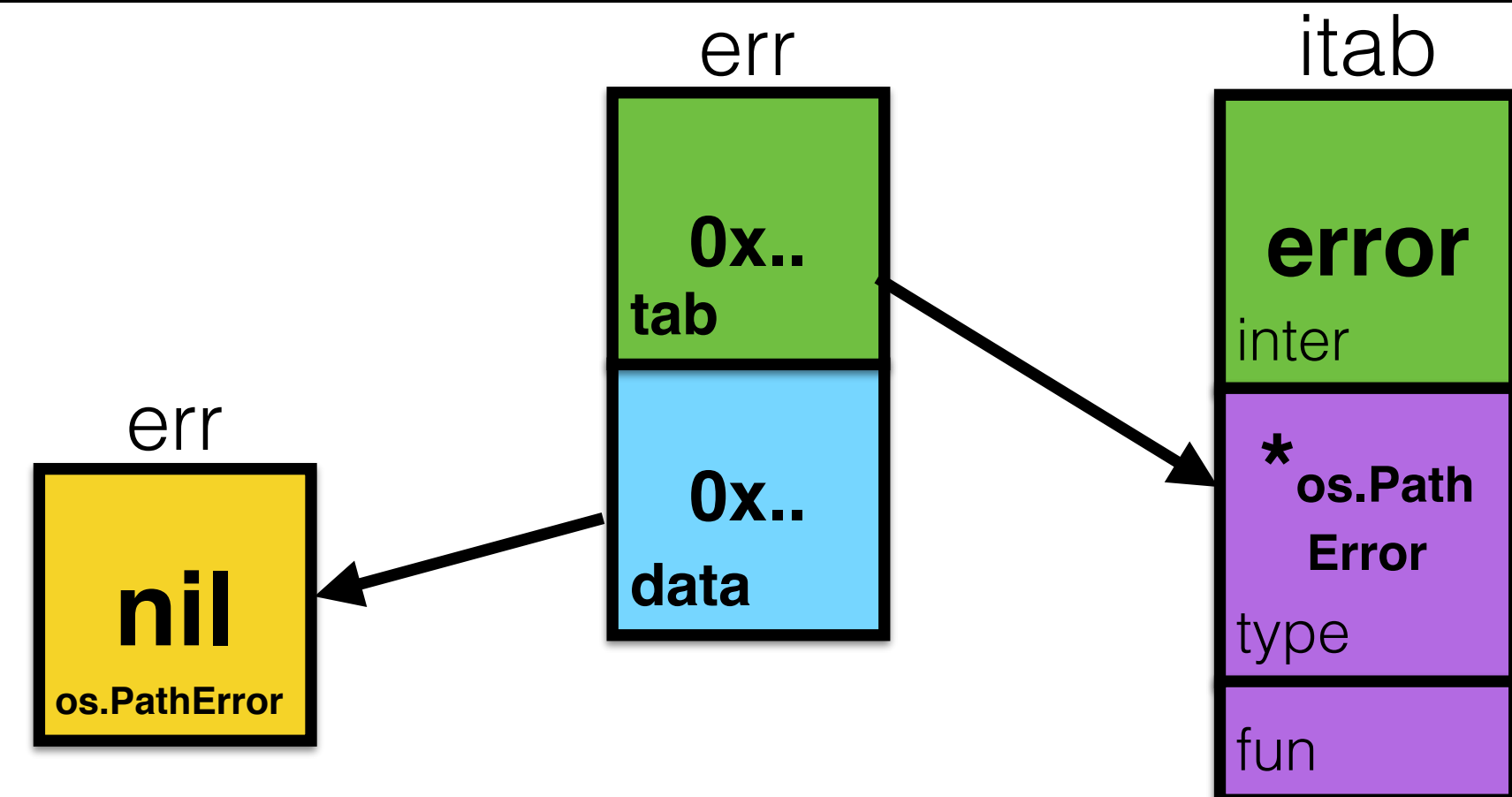


interfaces

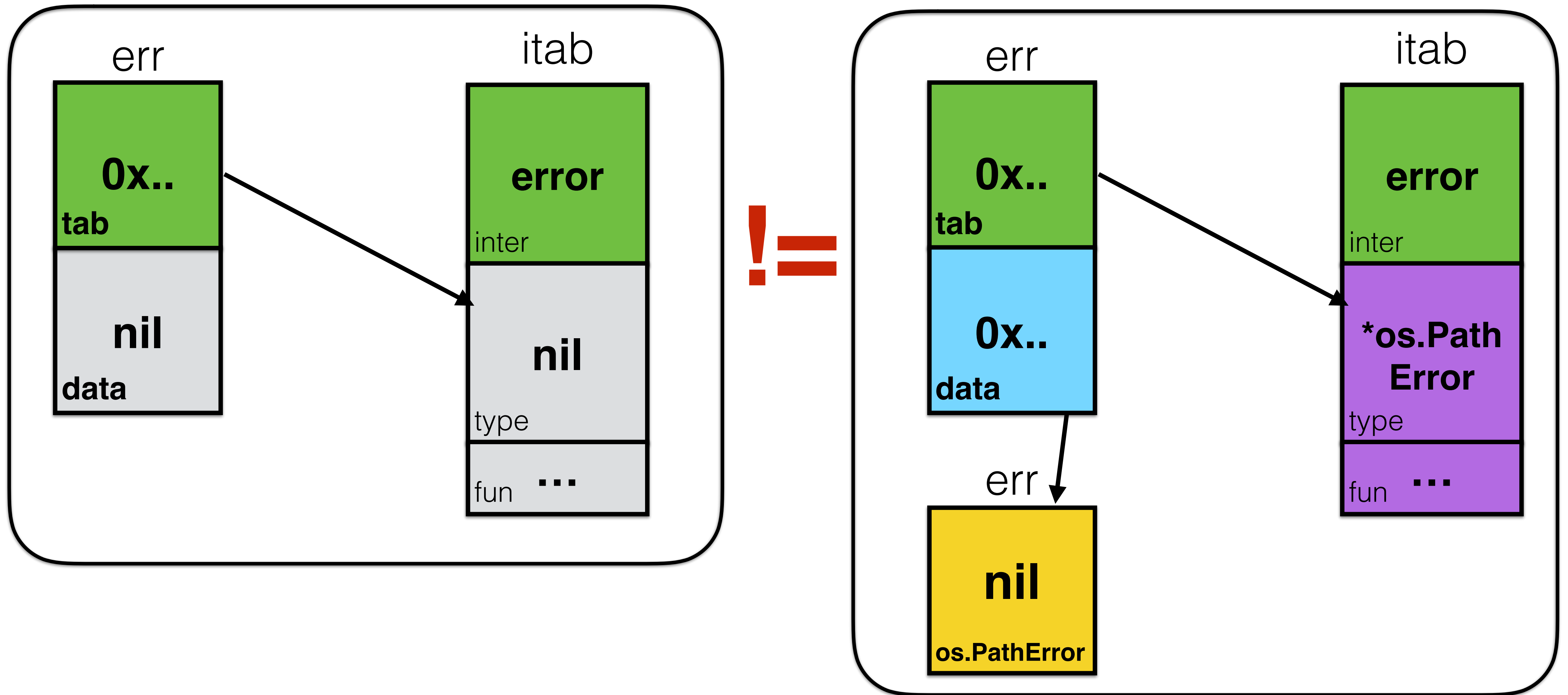
```
func foo() error {  
    var err error  
    // err == nil  
    return err  
}
```



```
func foo() error {  
    var err *os.PathError  
    // err == nil  
    return err  
}
```



interfaces



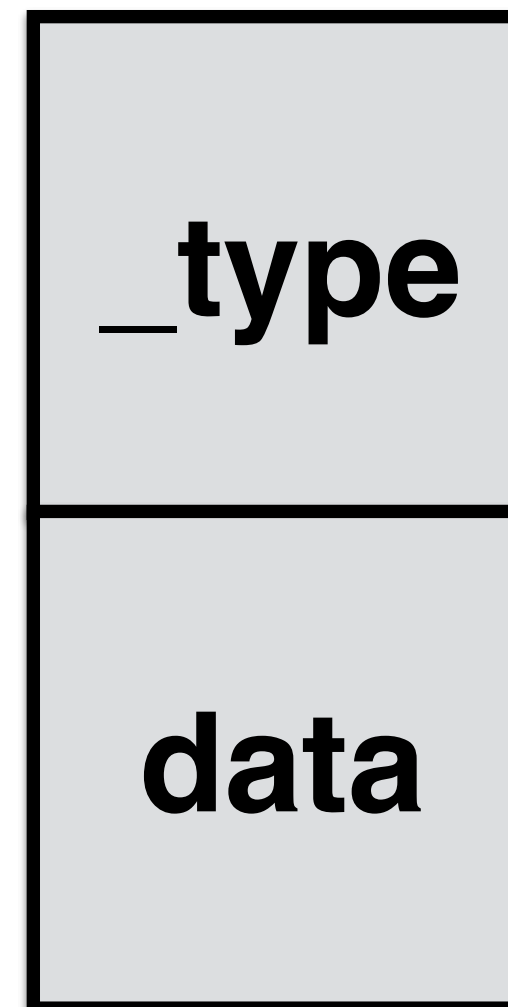
interfaces

Code:

```
type empty interface{}
```

Go code: [src/runtime/runtime2.go](https://sourcegraph.com/go/src/runtime/runtime2.go)

```
type eface struct {  
    _type *_type  
    data unsafe.Pointer  
}
```

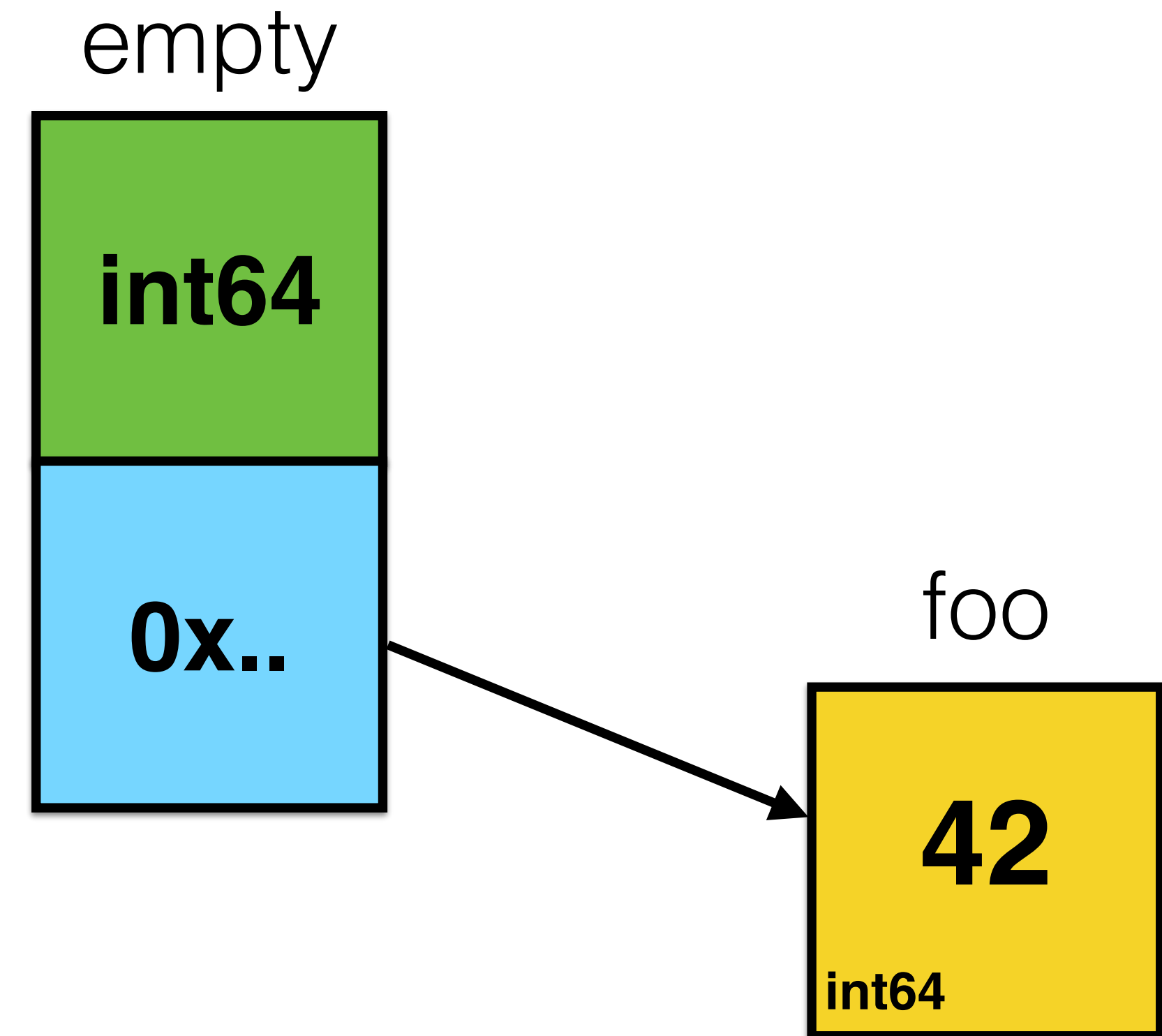


interfaces

Code:

```
var foo int64 = 42

func bar() interface{} {
    return foo
}
```



interfaces

Code:

```
func bar() interface{} {  
    return int64(42)  
}
```

interfaces

Code:

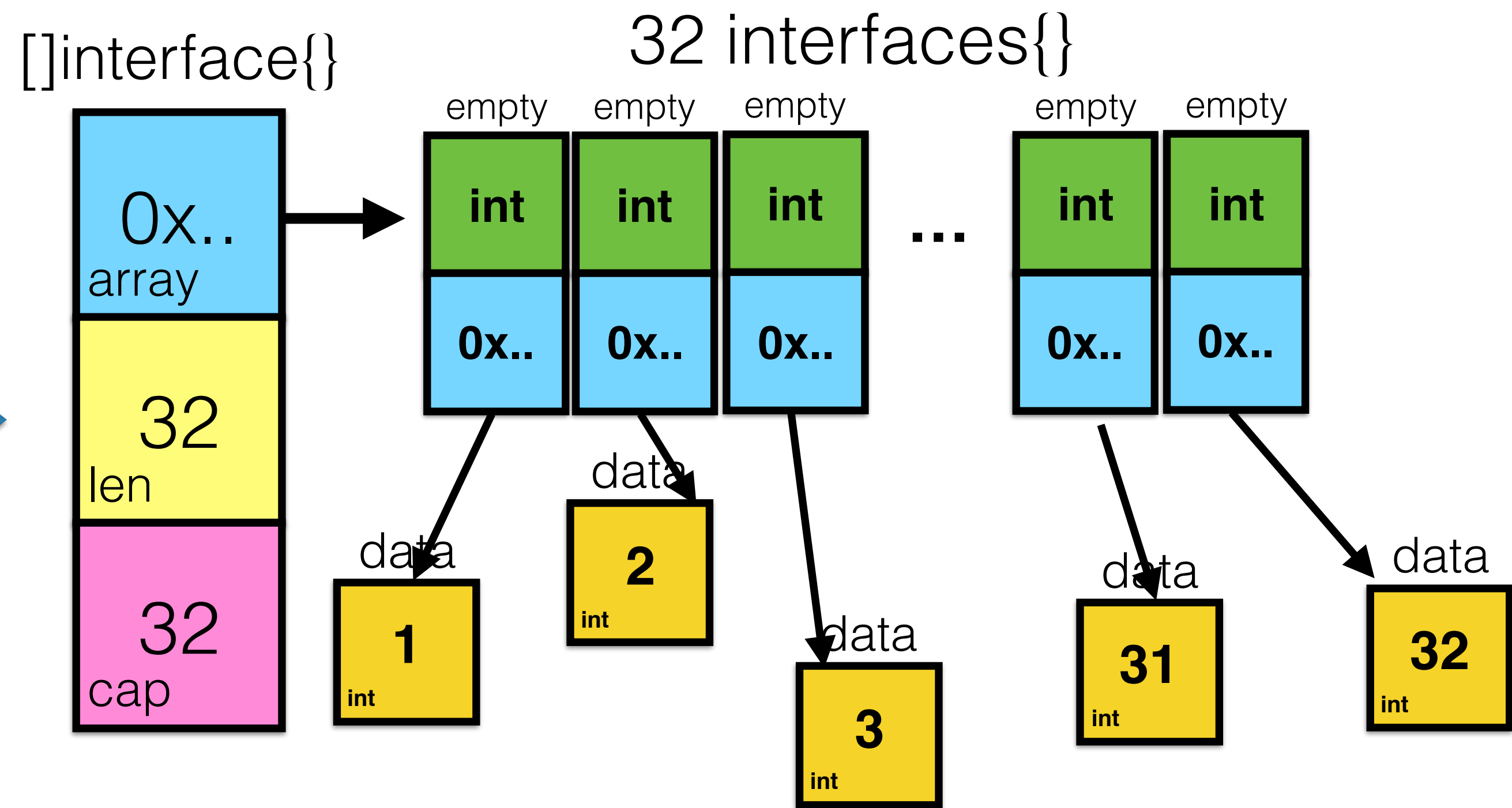
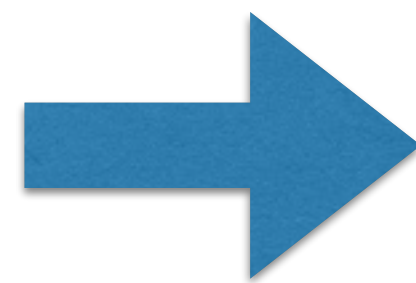
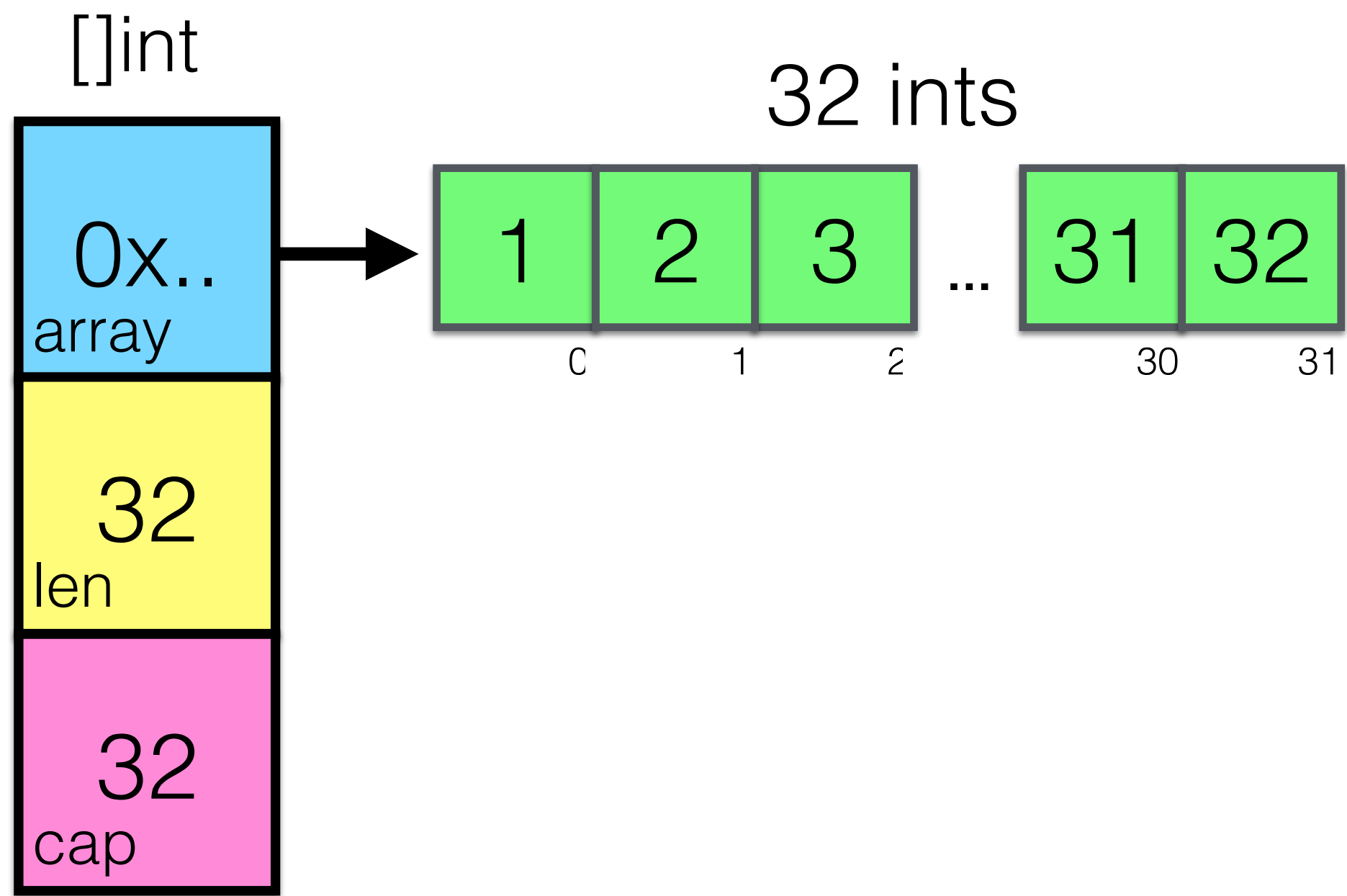
```
func bar() []interface{} {  
    return []int64{1, 2, 3, 4}  
}
```

interfaces

Code:

```
func bar() []interface{} {  
    return []int64{1, 2, 3, 4}  
}
```

```
$ go build  
cannot use []int literal (type []int) as type  
[]interface {} in return argument
```



*If you want to do something
expensive - do it explicitly*

Links

https://divan.github.io/posts/avoid_gotchas/

Links

- Must read:
 - [Go Data Structures](#)
 - [Go Data Structures: Interfaces](#)
 - [Go Slices: usage and internals](#)
 - [Gopher Puzzlers](#)
- And, of course:
 - [Go source code](#)
 - [Effective Go](#)
 - [Go spec](#)

Thank you
[@idanyliuk](#)