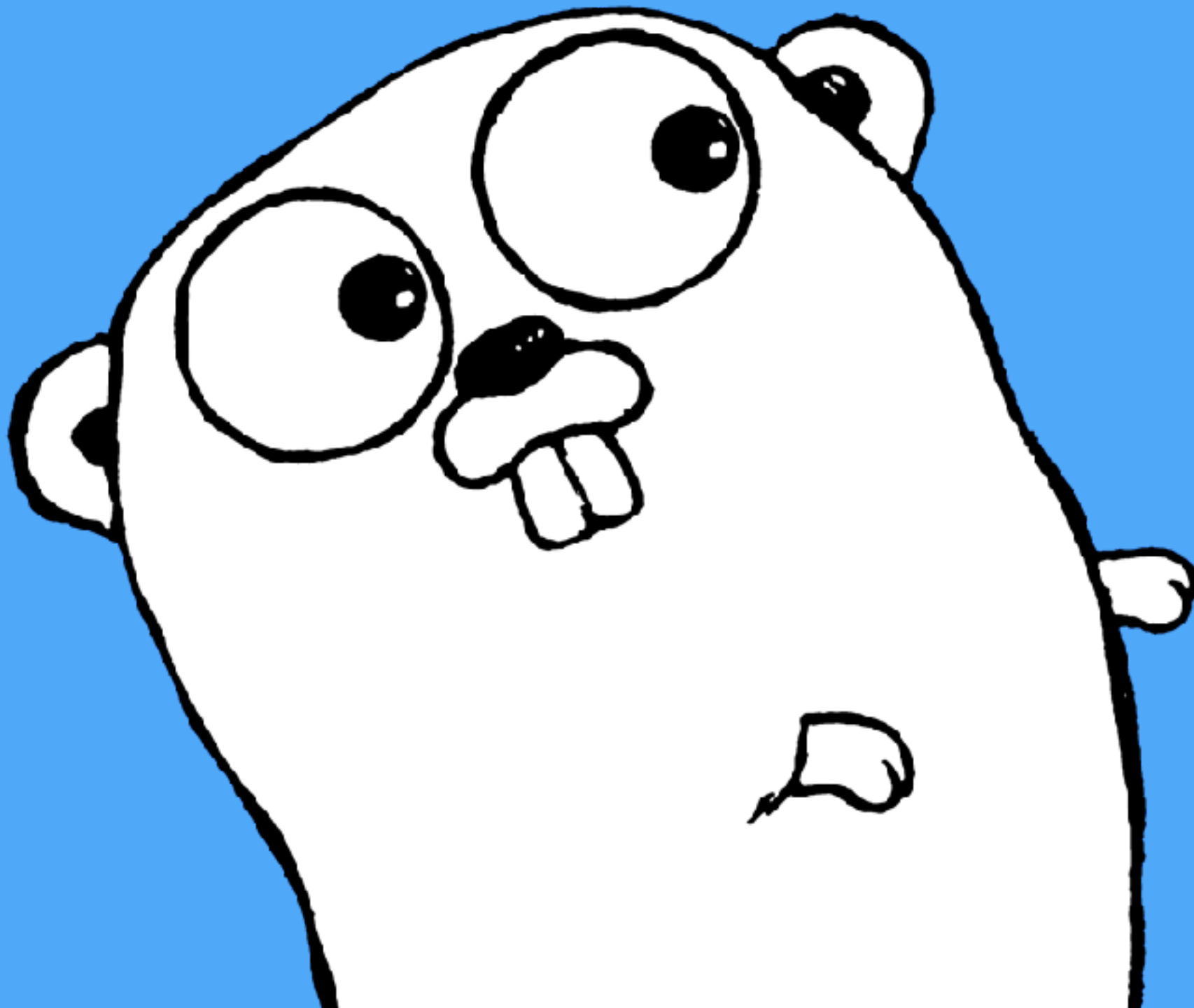
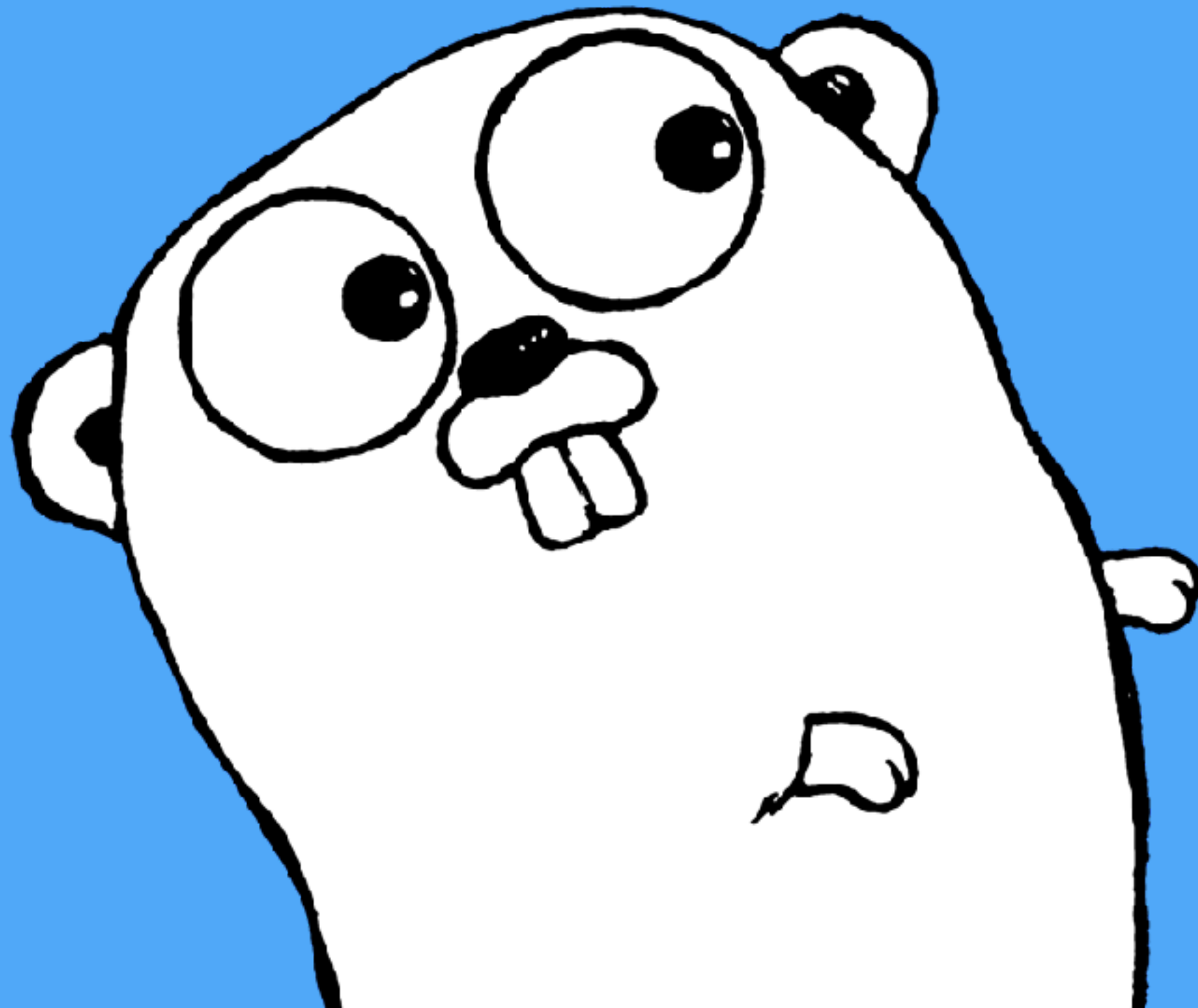


go get better



# 3 Day Go Training Course



# Dom Davis

@idomdavis



GOD

# GO Developer

# Goal Oriented Developer

# Goal Oriented Developer

# Goal Oriented Developer



# Goal Oriented Developer

# Goal Oriented Developer

# Deliberate Practice

# Deliberate Practice

# Goals



# Goals



# Introductions



# House Rules



# House Rules

There is no such thing as a stupid question

# House Rules

There is no such thing as a stupid question

We go at your pace

# House Rules

There is no such thing as a stupid question

We go at your pace

This is your course

# House Rules

There is no such thing as a stupid question

We go at your pace

This is your course

This is not school

# House Rules

There is no such thing as a stupid question

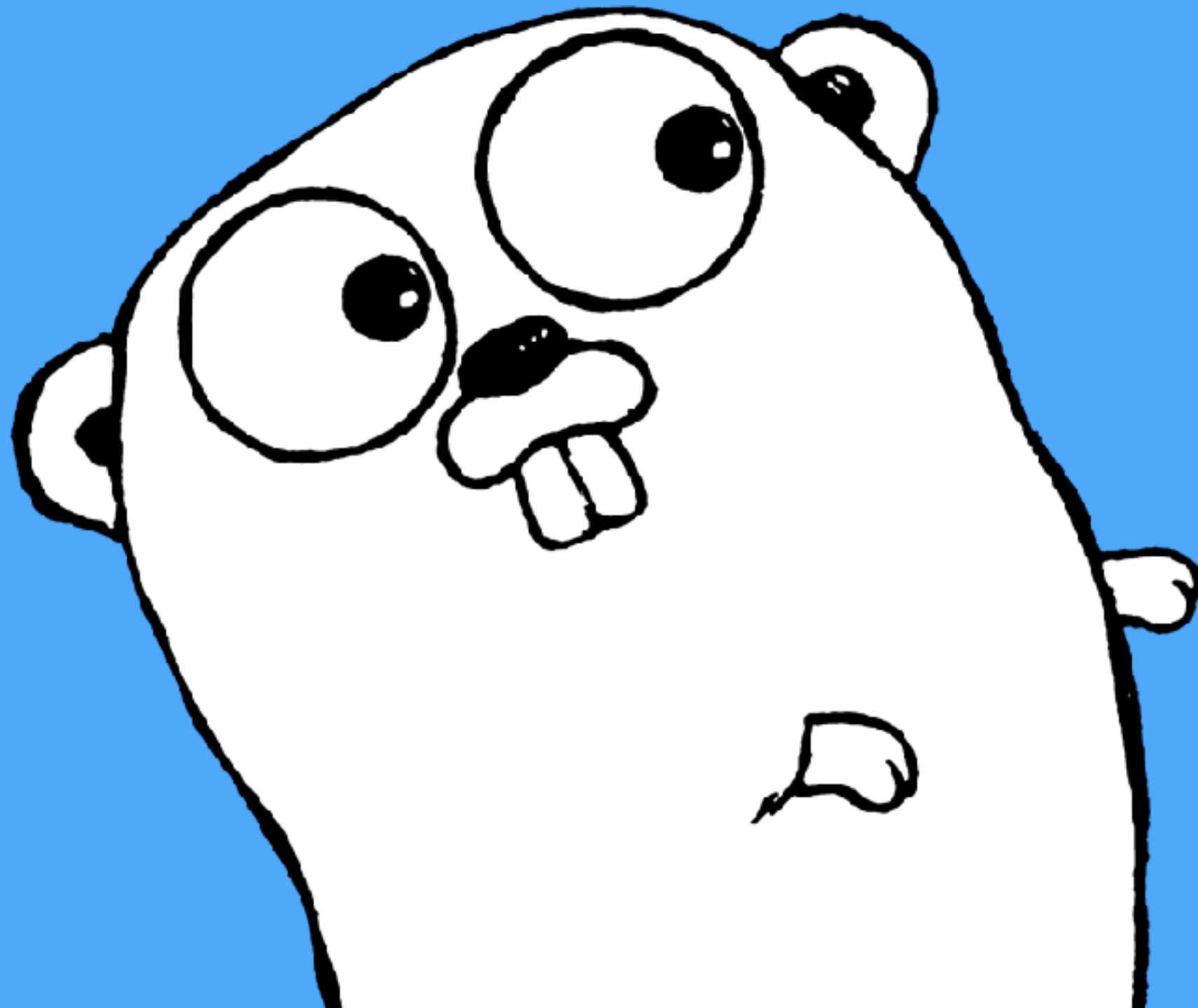
We go at your pace

This is your course

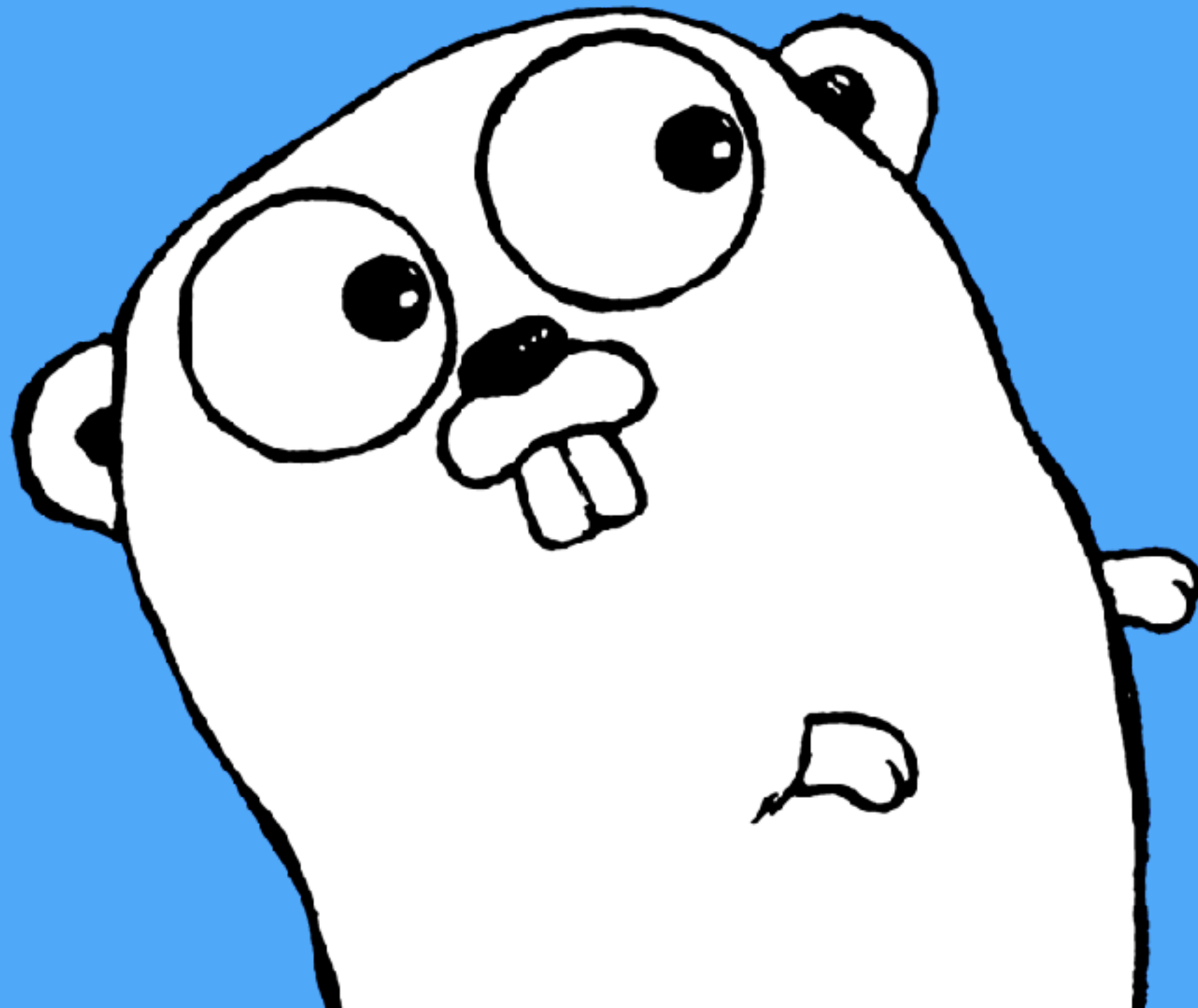
This is not school

Have fun

# Go



# Go



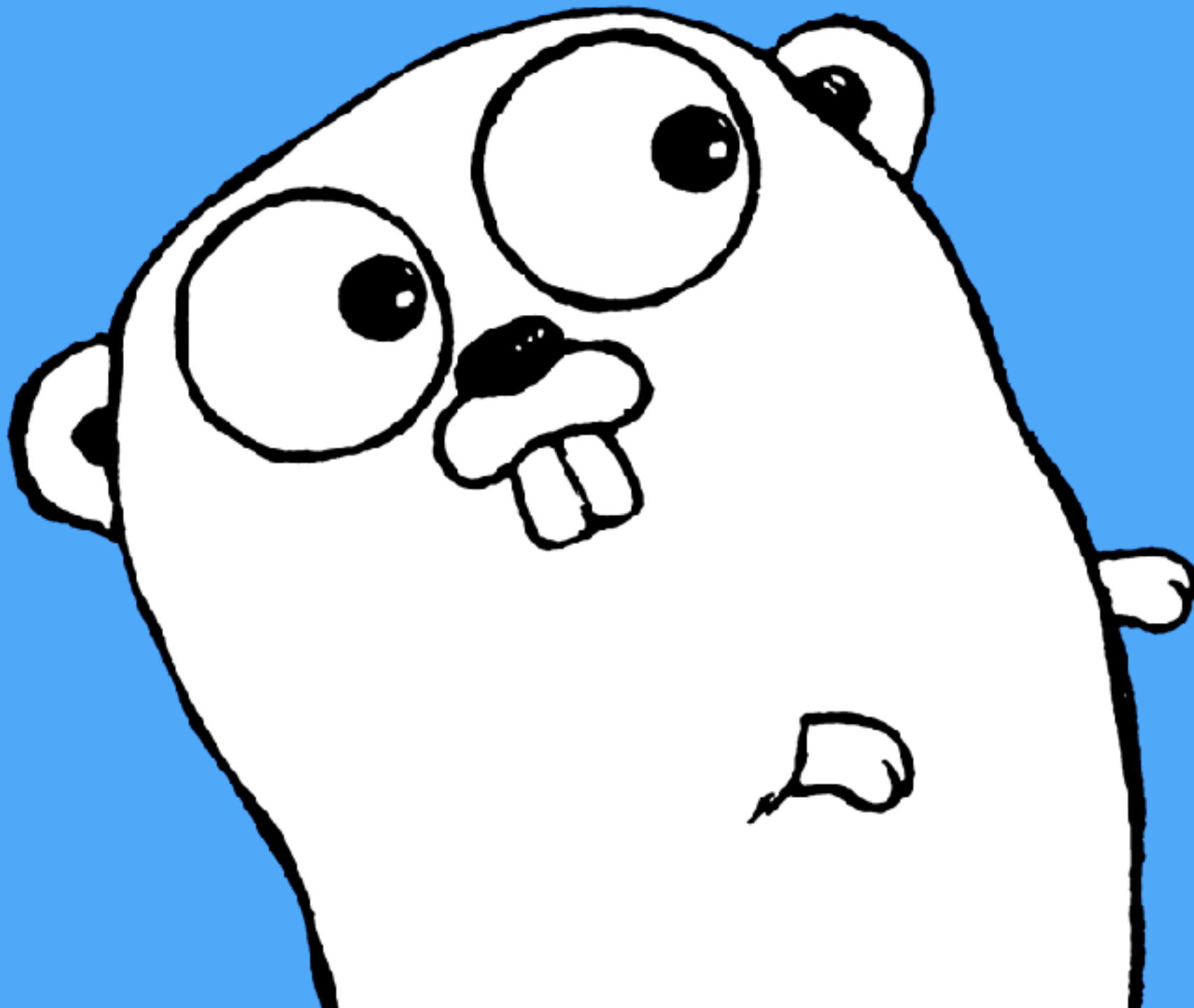
Simple, orthogonal features, that must carry their weight



Simple, orthogonal features, that must carry their weight  
Prefer maintainability over expressiveness

Simple, orthogonal features, that must carry their weight  
Prefer maintainability over expressiveness  
Typing is not so difficult

# Gophers



# 0: Goals

Make a list of goals for this course  
Make a list of your programming experience



Introduction, Setup & Go Katas  
Go Idiosyncrasies  
Types Interfaces and Polymorphism  
Concurrency and Parallelism  
Networking  
Packaging and Distribution  
Go Nuts

Introduction, Setup & Go Katas  
Go Idiosyncrasies  
Types Interfaces and Polymorphism  
Concurrency and Parallelism  
Networking  
Packaging and Distribution  
Go Nuts

Introduction, Setup & Go Katas  
Go Idiosyncrasies  
Types Interfaces and Polymorphism  
Concurrency and Parallelism  
Networking  
Packaging and Distribution  
Go Nuts

# The Basics





```
package main
```

```
func main() {  
    var s string = "Hello"  
    println(s)  
}
```



```
package main

func main() {
    var s1 string = "Hello"
    var s2 = "World"

    println(s1, s2)
}
```



```
package main

func main() {
    var s1 string = "Hello"
    s2 := "World"

    println(s1, s2)
}
```



```
package main
```

```
func main() {  
    s1, s2 := "Hello", "World"  
    println(s1, s2)  
}
```



```
package main

func main() {
    var (
        s1 = "Hello"
        s2 = "World"
    )
    println(s1, s2)
}
```

bool  
byte  
uintptr  
int, int8, int16, int32, int64  
uint, uint8, uint16, uint32, uint64  
float32, float64  
complex64, complex128  
rune, string

```
if a == "A" {  
    println("Good")  
} else if a == "B" {  
    println("Close enough")  
} else {  
    println("Nope!")  
}
```



```
if a, b := 1, 2; a == b {  
    println(a, "equal to", b)  
} else {  
    println(a, "not equal to", b)  
}
```

*// a, b are undefined here*





```
for i := 0; i < 100; i++ {  
    println(i)  
}
```

# FizzBuzz



1, 2, Fizz, 4, Buzz, Fizz, 7, 8, 9, Buzz, 11, Fizz, 13, 14, FizzBuzz

If a number is divisible by 3 output Fizz  
If a number is divisible by 5 output Buzz  
If a number is divisible by 3 and 5 output FizzBuzz  
Otherwise output the number

# 1: FizzBuzz

For the numbers 1 to 100

If a number is divisible by 3 output Fizz

If a number is divisible by 5 output Buzz

If a number is divisible by 3 and 5 output FizzBuzz

Otherwise output the number

<https://play.golang.org/>



```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    for i := 1; i <= 100; i++ {
        if i%15 == 1 {
            rand.Seed(int64(176064004))
        }
        fmt.Println([]string{fmt.Sprintf("%d", i),
            "fizz", "buzz", "fizzbuzz"}[rand.Int63()%4])
    }
}
```



```
package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    for i := 1; i <= 100; i++ {
        if i%3 == 0 && i%5 == 0 {
            fmt.Printf("%-5d - %s%s", i, fizz, buzz)
        } else if i%3 == 0 {
            fmt.Printf("%-5d - %s", i, fizz)
        } else if i%5 == 0 {
            fmt.Printf("%-5d - %s", i, buzz)
        } else {
            fmt.Print(i)
        }
        fmt.Println()
    }
}
```

FizzBuzz Enterprise Edition is a no-nonsense implementation of FizzBuzz made by serious businessmen for serious business purposes. <http://www.fizzbuzz.enterprises>

🕒 161 commits

🌿 1 branch

📦 0 releases

👤 30 contributors

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾



emiln committed on GitHub Merge pull request #281 from tkellogg/master ...

Latest commit cdfac75 on 4 Feb

📁 gradle/wrapper	Include Gradle config closes #230	2 years ago
📁 resources/assets/configuration/sp...	Installed spring and restructured for dependency injection.	3 years ago
📁 src	FizzStringReturner may not have copied all characters	8 months ago
📄 .gitattributes	Started the project. Time to learn!	5 years ago
📄 .gitignore	Add unsupported platform files to the .gitignore	2 years ago
📄 .travis.yml	Reverting the revert commit, since it clearly did not revert as inten...	2 years ago
📄 CONTRIBUTING.md	Reverting the revert commit, since it clearly did not revert as inten...	2 years ago
📄 README.md	Reverting the revert commit, since it clearly did not revert as inten...	2 years ago
📄 build.gradle	Include Gradle config closes #230	2 years ago
📄 gradlew	Include Gradle config closes #230	2 years ago
📄 gradlew.bat	Include Gradle config closes #230	2 years ago
📄 pom.xml	Merge pull request #232 from kristianperkins/patch-1	a year ago
📄 settings.gradle	Include Gradle config closes #230	2 years ago



```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    for i := 1; i <= 100; i++ {
        if i%15 == 1 {
            rand.Seed(int64(176064004))
        }
        fmt.Println([]string{fmt.Sprintf("%d", i),
            "fizz", "buzz", "fizzbuzz"}[rand.Int63()%4])
    }
}
```

```
package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    for i := 1; i <= 100; i++ {
        if i%3 == 0 && i%5 == 0 {
            fmt.Printf("%-5d - %s%s", i, fizz, buzz)
        } else if i%3 == 0 {
            fmt.Printf("%-5d - %s", i, fizz)
        } else if i%5 == 0 {
            fmt.Printf("%-5d - %s", i, buzz)
        } else {
            fmt.Print(i)
        }
        fmt.Println()
    }
}
```

```
package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    for i := 1; i <= 100; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            fmt.Printf("%-5d - %s%s", i, fizz, buzz)
        case i%3 == 0:
            fmt.Printf("%-5d - %s", i, fizz)
        case i%5 == 0:
            fmt.Printf("%-5d - %s", i, buzz)
        default:
            fmt.Print(i)
        }

        fmt.Println()
    }
}
```

```
switch i {  
case 1:  
    println("One")  
case 2:  
    println("A couple")  
default:  
    println("Many")  
}
```

```
switch i {  
  case 1:  
    println("One")  
  case 2, 3, 4, 5:  
    println("Some")  
  default:  
    println("Many")  
}
```

```
switch {  
  case i == 1:  
    println("One")  
  case i <= 5:  
    println("Some")  
  default:  
    println("Many")  
}
```

# Go Is Opinionated



# Go Is Opinionated

Go is not a language to express your inner artist



# Go Is Opinionated

Go is not a language to express your inner artist  
There is [almost] always one idiomatic way to do something



# Go Is Opinionated

Go is not a language to express your inner artist  
There is [almost] always one idiomatic way to do something  
There is only one way to format your code: `go fmt`



```
package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    for i := 1; i <= 100; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            fmt.Printf("%-5d - %s%s", i, fizz, buzz)
        case i%3 == 0:
            fmt.Printf("%-5d - %s", i, fizz)
        case i%5 == 0:
            fmt.Printf("%-5d - %s", i, buzz)
        default:
            fmt.Print(i)
        }

        fmt.Println()
    }
}
```

go vet  
goimports  
golint

go get setup



## 2: FizzBuzz Binary

For the numbers 1 to 100

If a number is divisible by 3 output Fizz

If a number is divisible by 5 output Buzz

If a number is divisible by 3 and 5 output FizzBuzz

Otherwise output the number

Run your program as a compiled binary from the command line



```
package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    for i := 1; i <= 100; i++ {
        txt := ""

        if i%3 == 0 {
            txt += fizz
        }

        if i%5 == 0 {
            txt += buzz
        }

        fmt.Printf("%-5d - %s\n", i, txt)
    }
}
```

```
package main

import "fmt"

var pattern = []int{0, 0, 1, 0, 2, 1, 0, 0, 1, 2, 0, 1, 0, 0, 3}

func main() {
    for i := 1; i <= 100; i++ {
        fmt.Println([]string{fmt.Sprintf("%d", i),
            "fizz", "buzz", "fizzbuzz"}[pattern[(i-1)%15]])
    }
}
```



# loops



```
for i := 0; i < 100; i++ {  
    println(i)  
}
```

```
i := 0  
for i < 100 {  
    i++  
}
```

```
for true {  
    println("spam")  
}
```

```
for {  
    println("spam")  
}
```

framework training:



```
for i := range []int{1, 1, 2, 3, 5, 8} {  
    println(i)  
}
```



```
for _, v := range []int{1, 1, 2, 3, 5, 8} {  
    println(v)  
}
```

```
m := map[string]int{  
    "one": 1,  
    "two": 2,  
}  
  
for k, v := range m {  
    println(k, v)  
}
```





# Arrays and Slices



```
var a [4]int  
a[0] = 1
```

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

```
a := [2]int{1, 2}  
b := [4]int{1, 2}  
c := [...]int{1, 2}
```

```
var a []int // read "slice of int"  
a = append(a, 123)  
a = append(a, 456)
```



```
a := []int{123, 456, 789}
```

```
fmt.Printf("%v\n", a[0])
```

```
a[1] = 555
```

```
fmt.Printf("%v\n", a[1:])
```

```
fmt.Printf("%v\n", a[:1])
```

```
fmt.Printf("%v\n", a[1:2])
```



```
a := [][]int{  
    []int{1, 2, 3},  
    []int{4, 5},  
    []int{6, 7, 8, 9},  
}
```

```
a = append(a, []int{10, 11, 12})
```

```
a := [] []int{  
    {1, 2, 3},  
    {4, 5},  
    {6, 7, 8, 9},  
}
```

```
a = append(a, []int{10, 11, 12})
```

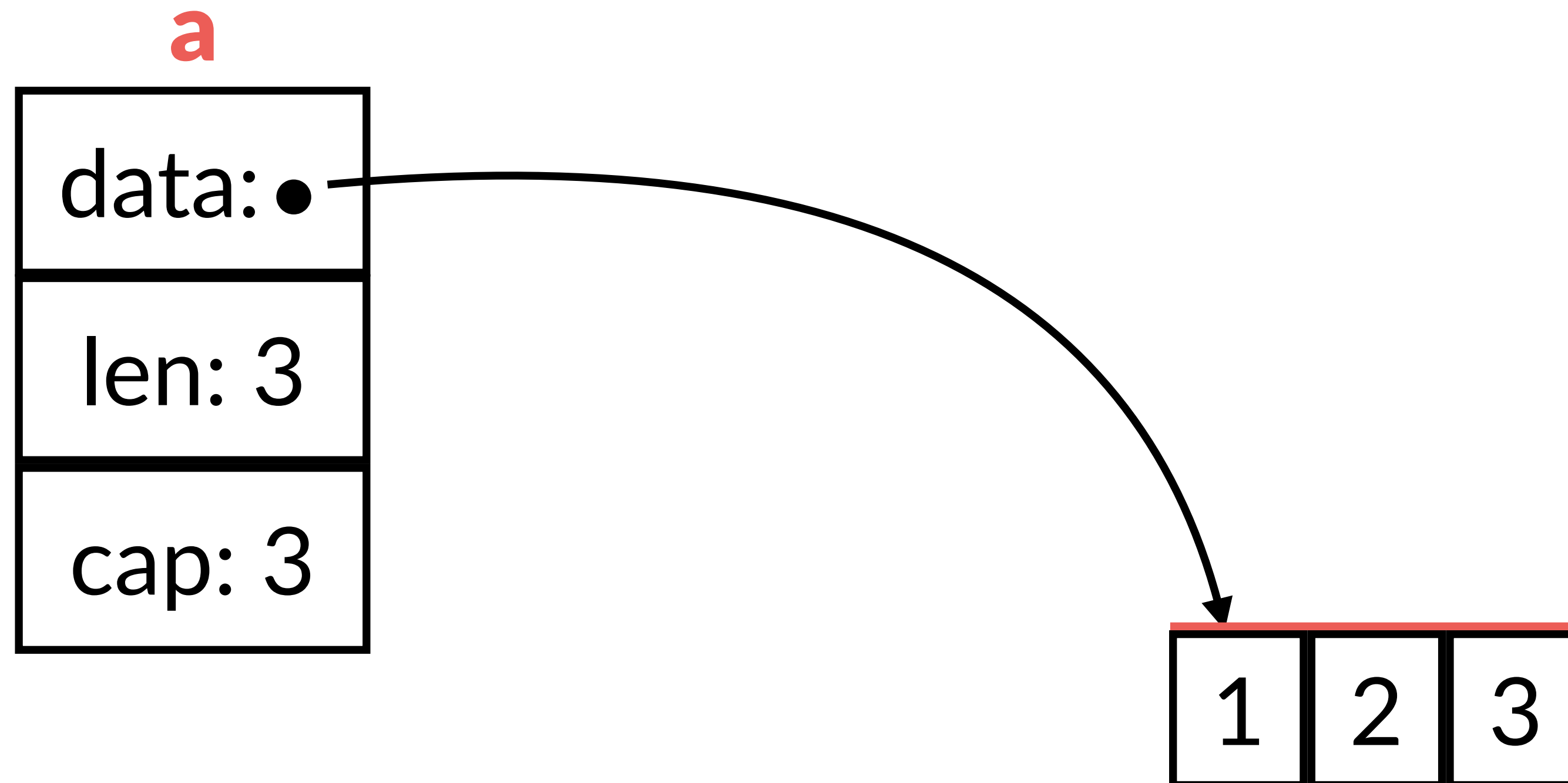
```
a := make([]int, 5)
a[3] = 123 // OK
a = append(a, 456)
```

```
b := make([]int, 0, 5)
b[3] = 123 // panic!
b = append(b, 789)
```



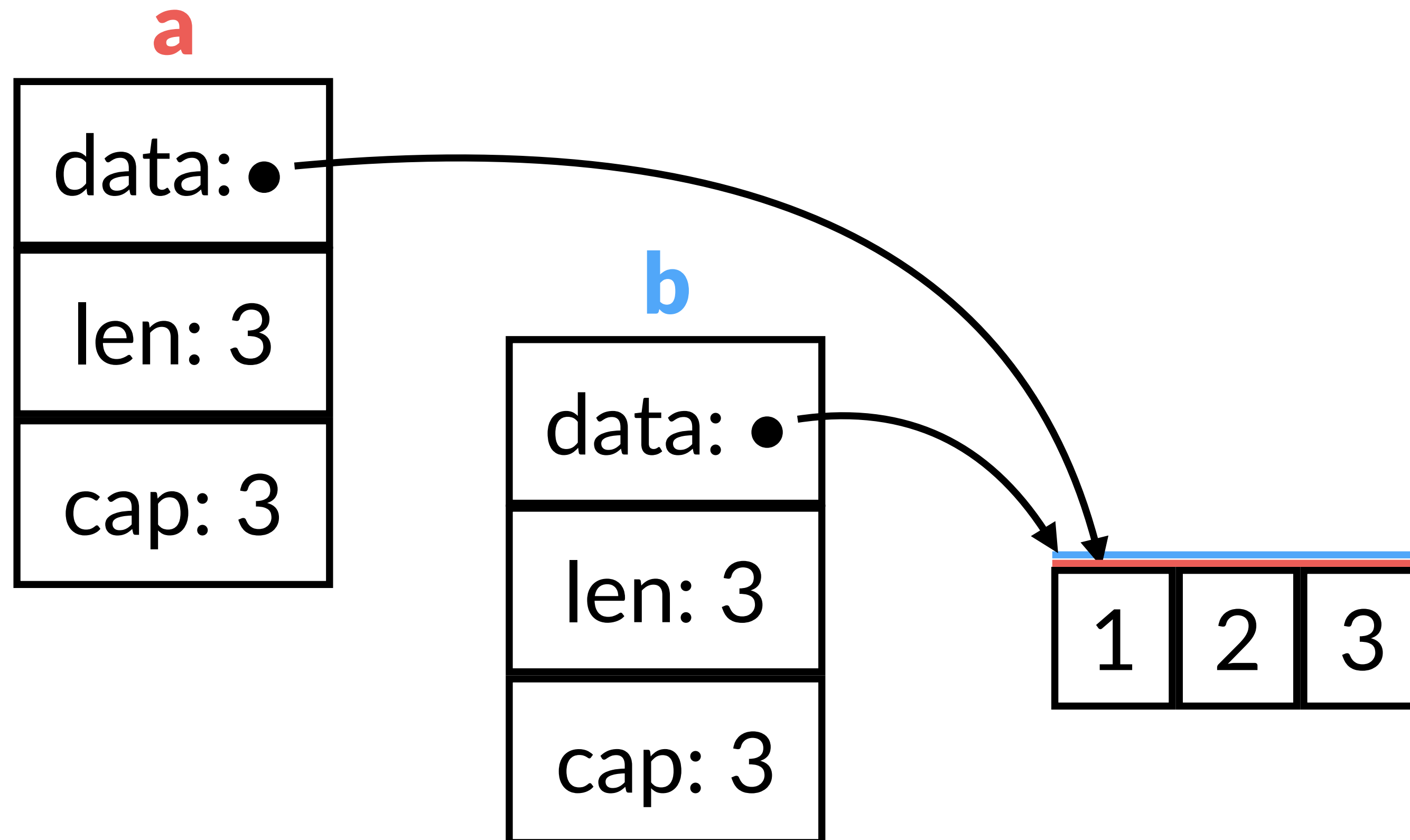
# Slice Implementation

```
a := []int{1, 2, 3}
```



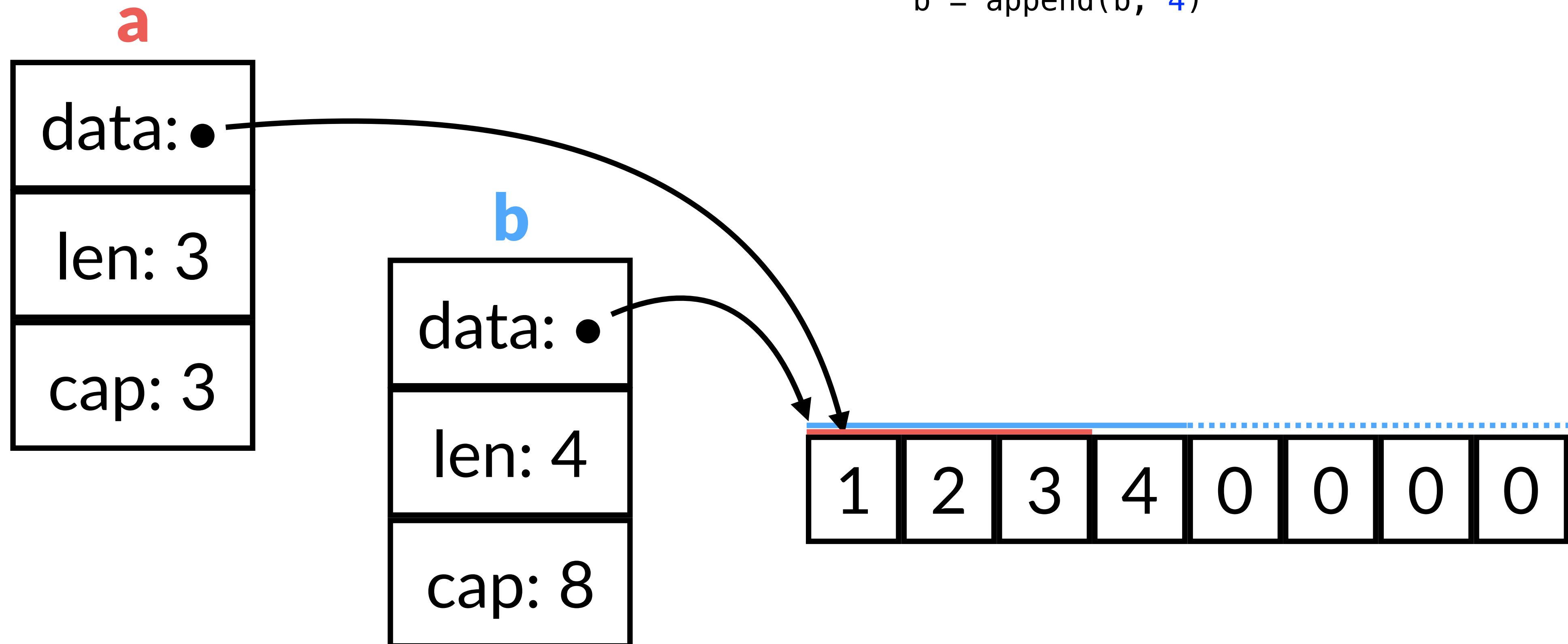
# Slice Implementation

```
a := []int{1, 2, 3}  
b := a
```



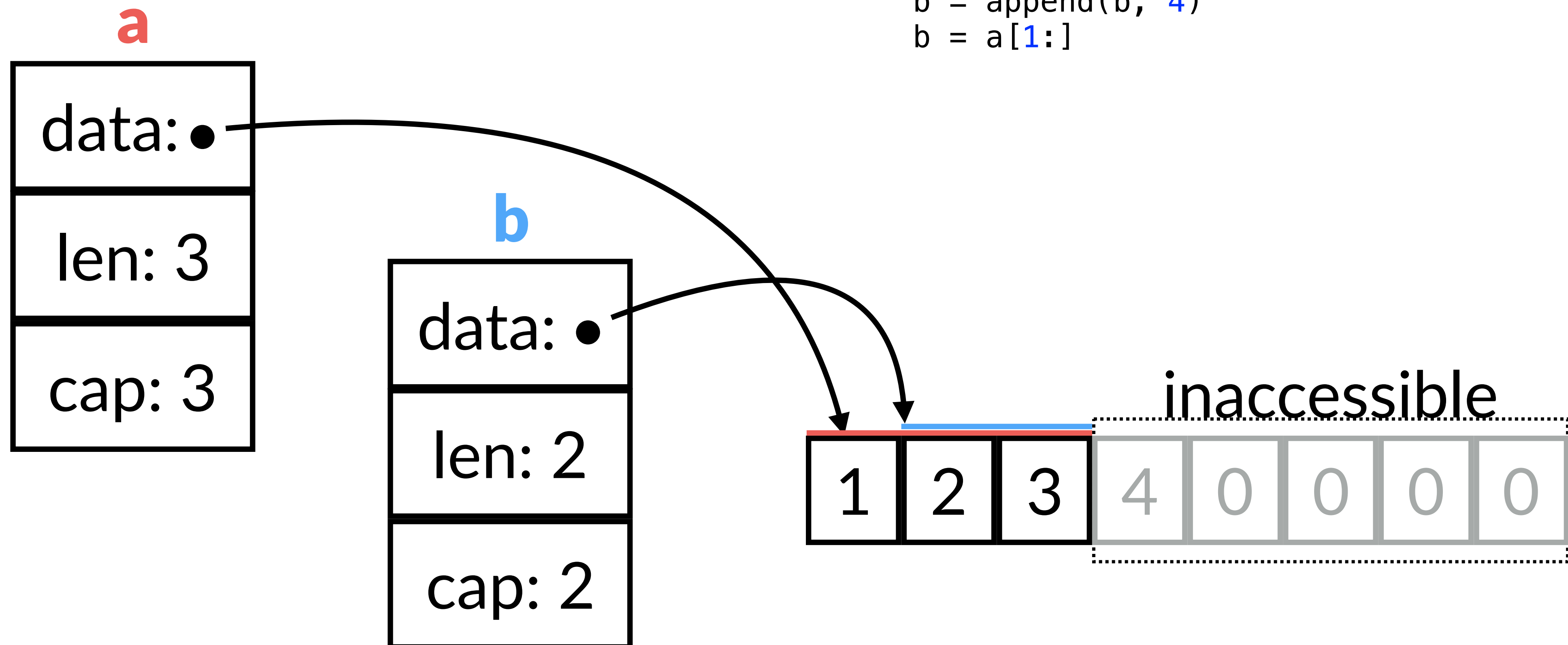
# Slice Implementation

```
a := []int{1, 2, 3}  
b := a  
b = append(b, 4)
```



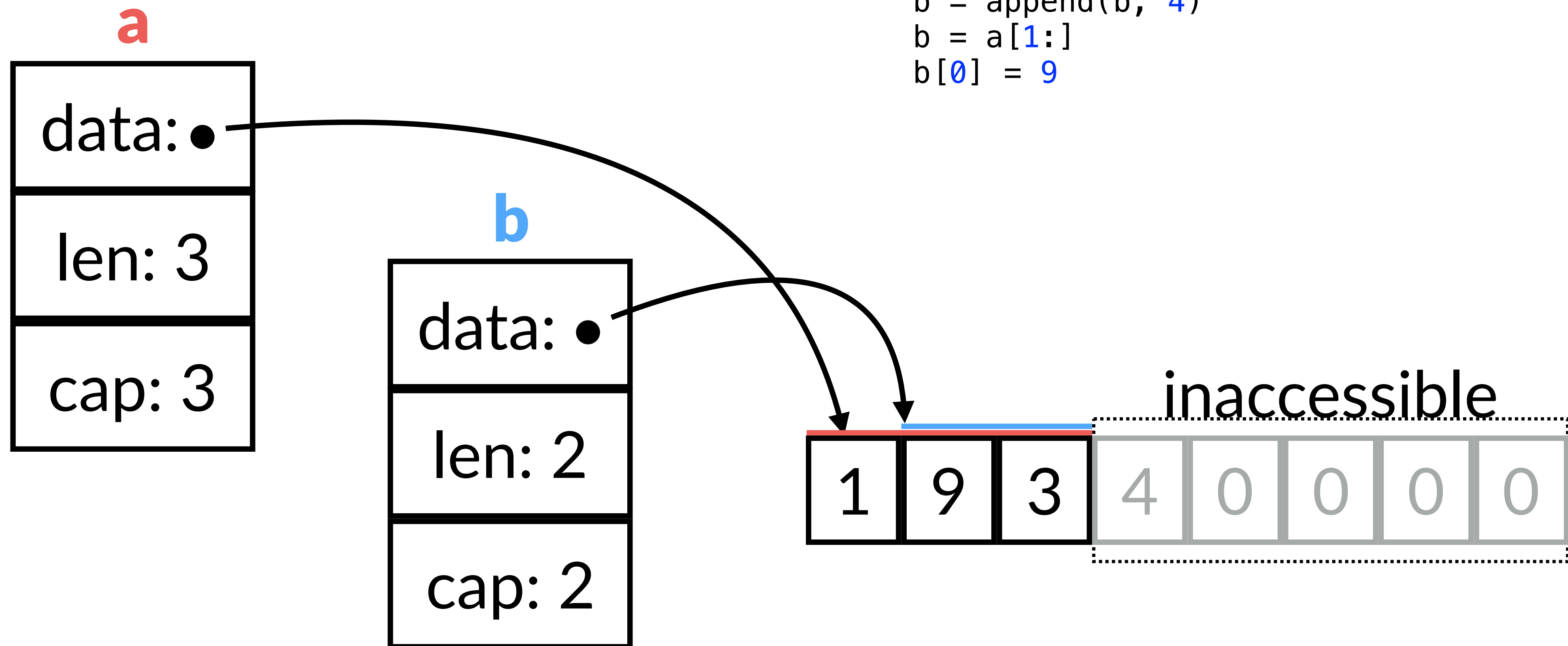
# Slice Implementation

```
a := []int{1, 2, 3}  
b := a  
b = append(b, 4)  
b = a[1:]
```



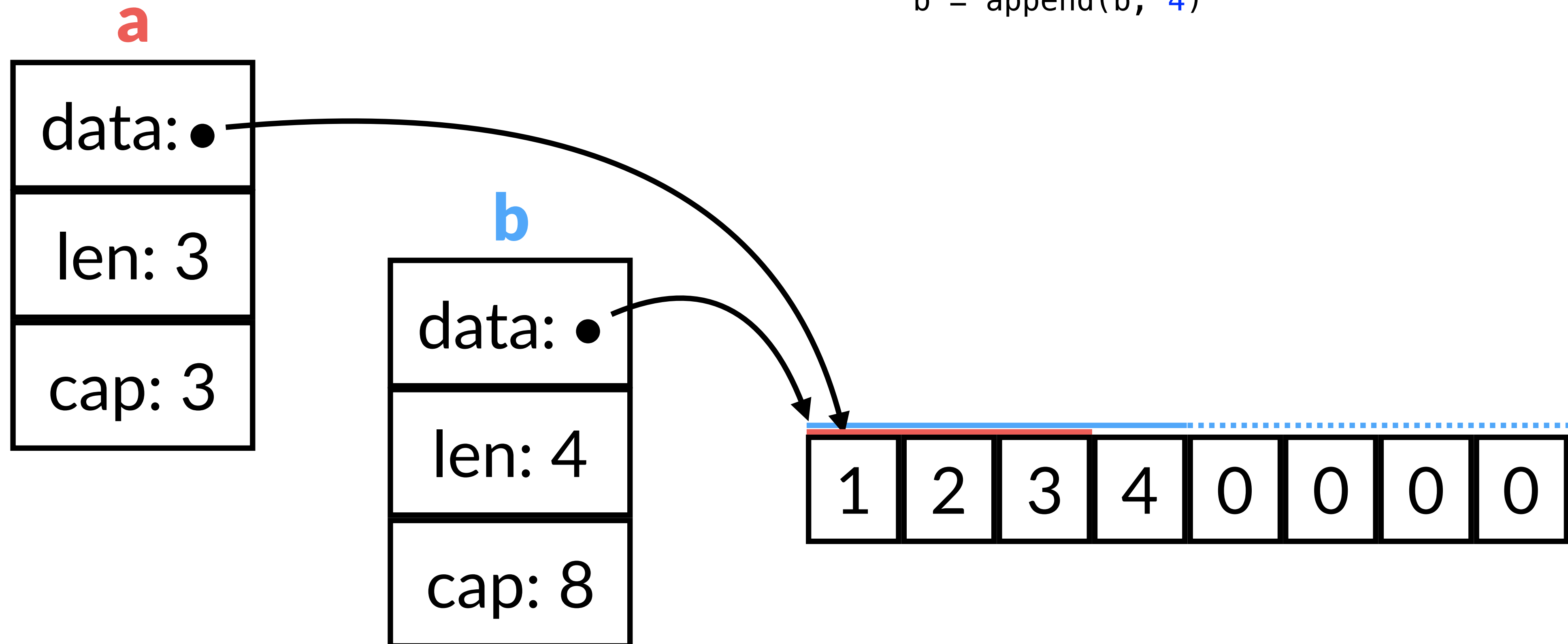
# Slice Implementation

```
a := []int{1, 2, 3}  
b := a  
b = append(b, 4)  
b = a[1:]  
b[0] = 9
```



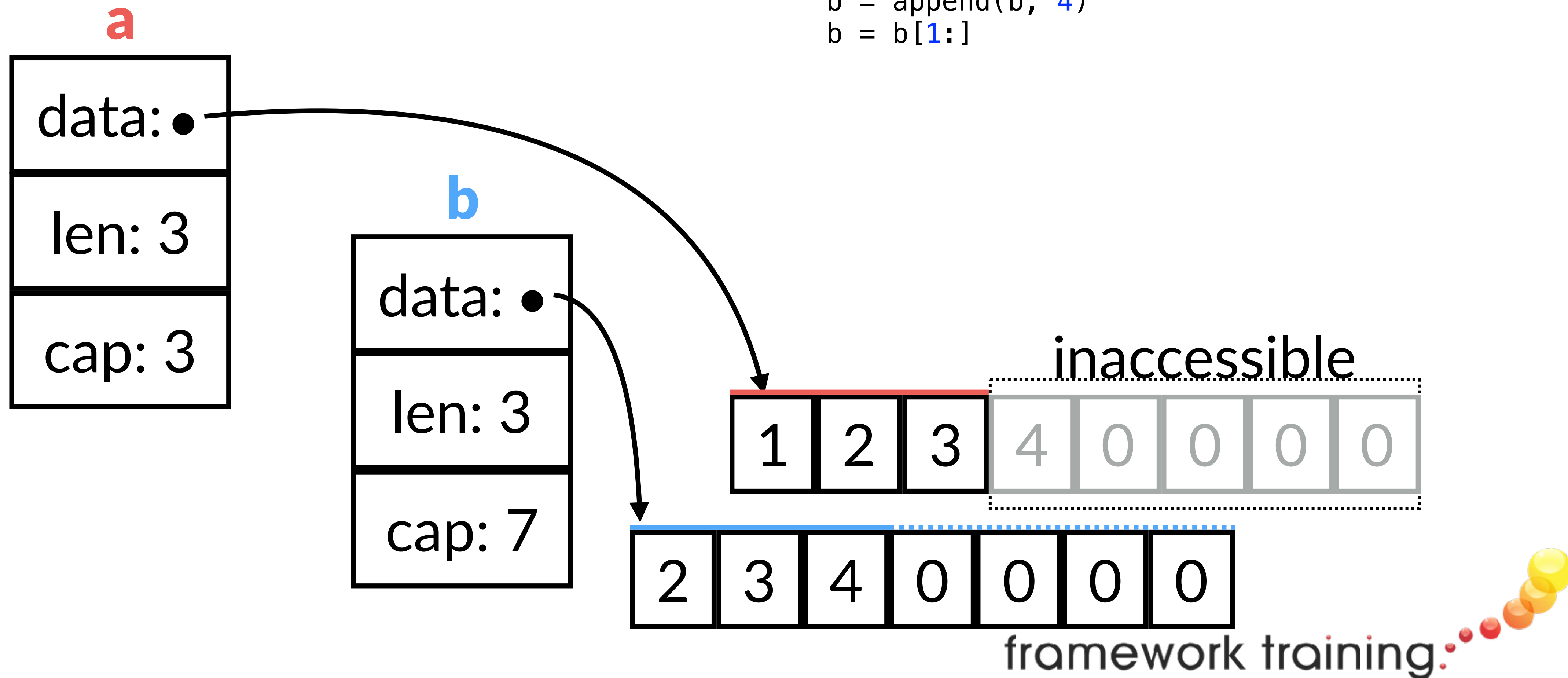
# Slice Implementation

```
a := []int{1, 2, 3}  
b := a  
b = append(b, 4)
```



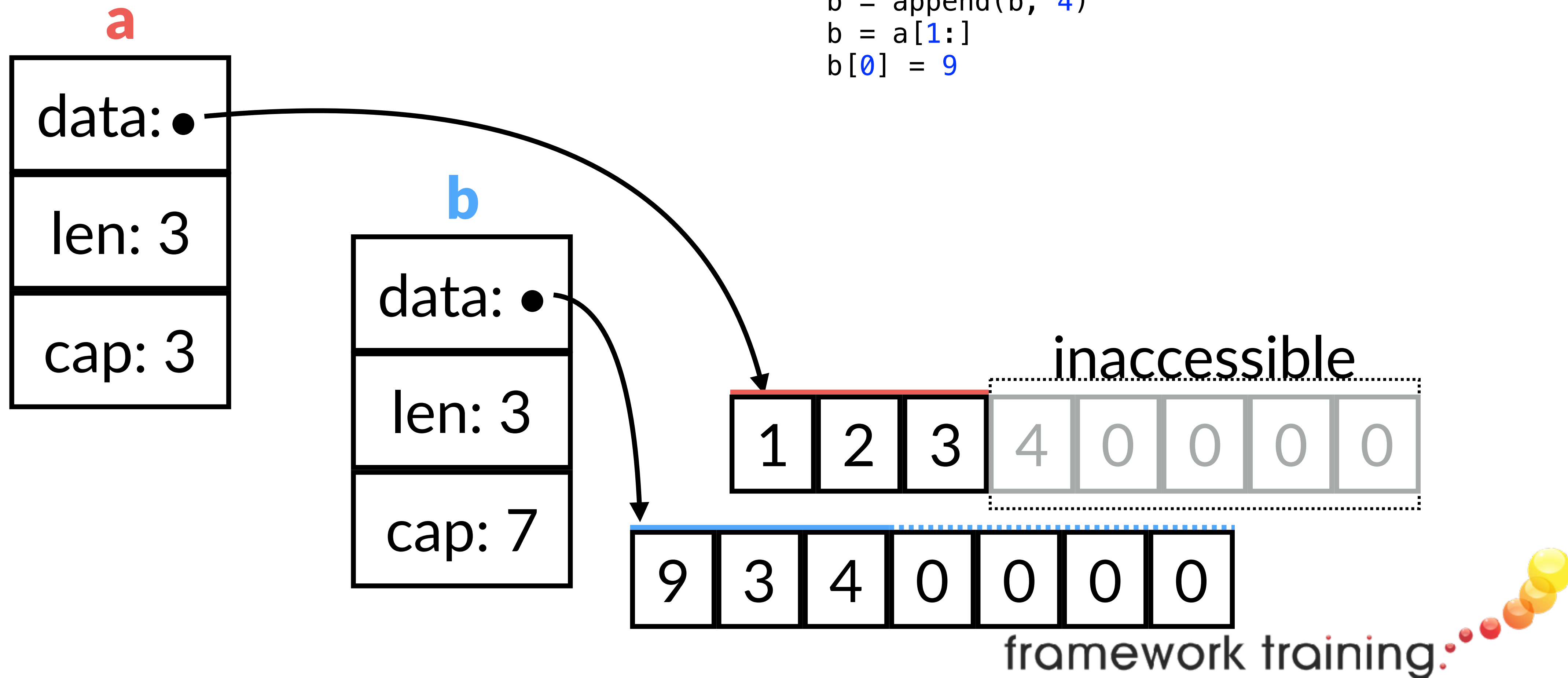
# Slice Implementation

```
a := []int{1, 2, 3}  
b := a  
b = append(b, 4)  
b = b[1:]
```



# Slice Implementation

```
a := []int{1, 2, 3}
b := a
b = append(b, 4)
b = a[1:]
b[0] = 9
```





```
for _, v := range []int{1, 1, 2, 3, 5, 8} {  
    println(v)  
}
```

# 3: Partial FizzBuzz

For the numbers 1, 2, 4, 5, 6, 7, 8, 10, 21, 42

If a number is divisible by 3 output Fizz

If a number is divisible by 5 output Buzz

If a number is divisible by 3 and 5 output FizzBuzz

Otherwise output the number

Run your program as a compiled binary from the command line



```
package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    for _, i := range []int{1, 2, 4, 5, 6, 7, 8, 15, 21, 42} {
        switch {
        case i%3 == 0 && i%5 == 0:
            fmt.Printf("%-5d - %s%s", i, fizz, buzz)
        case i%3 == 0:
            fmt.Printf("%-5d - %s", i, fizz)
        case i%5 == 0:
            fmt.Printf("%-5d - %s", i, buzz)
        default:
            fmt.Print(i)
        }

        fmt.Println()
    }
}
```



# 4: FizzBuzz Function

For an arbitrary set of integers

If a number is divisible by 3 output Fizz

If a number is divisible by 5 output Buzz

If a number is divisible by 3 and 5 output FizzBuzz

Otherwise output the number



```

package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    fizzbuzz([]int{1, 2, 4, 5, 6, 7, 8, 15, 21, 42})
}

func fizzbuzz(values []int) {
    for _, i := range values {
        switch {
        case i%3 == 0 && i%5 == 0:
            fmt.Printf("%-5d - %s%s", i, fizz, buzz)
        case i%3 == 0:
            fmt.Printf("%-5d - %s", i, fizz)
        case i%5 == 0:
            fmt.Printf("%-5d - %s", i, buzz)
        default:
            fmt.Print(i)
        }

        fmt.Println()
    }
}

```

```

package main

import "fmt"

const (
    fizz = "Fizz"
    buzz = "Buzz"
)

func main() {
    fizzbuzz(1, 2, 4, 5, 6, 7, 8, 15, 21, 42)
}

func fizzbuzz(values ...int) {
    for _, i := range values {
        switch {
        case i%3 == 0 && i%5 == 0:
            fmt.Printf("%-5d - %s%s", i, fizz, buzz)
        case i%3 == 0:
            fmt.Printf("%-5d - %s", i, fizz)
        case i%5 == 0:
            fmt.Printf("%-5d - %s", i, buzz)
        default:
            fmt.Print(i)
        }

        fmt.Println()
    }
}

```

# Pointers

# Maps





```
var m map[string]int  
m["A"] = 1
```

nil

framework training:

A decorative graphic consisting of a series of overlapping spheres in red, orange, and yellow, arranged in a curved path that extends from the text 'framework training:' towards the top right corner of the slide.

```
var m map[string]int = make(map[string]int)
m["A"] = 1
```

```
m := make(map[string]int)
m["A"] = 1
```

```
m := map[string]int{}  
m["A"] = 1
```

```
m := make(map[string]int, 100)  
m["A"] = 1
```

```
m := map[string]int{  
    "A": 1,  
}
```

MapType = "map" "[" **KeyType** "]" ElementType

"The comparison operators == and != must be fully defined for operands of the key type; thus the key type must not be a function, map, or slice."

<https://golang.org/doc/ref#KeyType>



```
m := map[string]int{  
    "A": 1,  
}  
  
println(m["A"])
```

```
m := map[string]int{  
    "A": 1,  
}  
  
println(m["B"])
```

```
m := map[string]int{  
    "A": 1,  
}  
  
i, ok := m["B"]  
println(i, ok)
```

```
m := map[string]int{  
    "A": 0,  
}  
  
_, ok := m["A"]  
println(ok)
```

```
m := map[string]int{  
    "A": 1,  
}  
delete(m, "A")
```

```
m := map[int]string{  
    1: "A",  
    2: "B",  
    3: "C",  
}  
  
for k := range m {  
    println(k)  
}
```

```
m := map[int]string{  
    1: "A", 2: "B", 3: "C"}  
  
for k := range m {  
    println(k)  
}
```



```
func main() {  
    m := map[int]string{  
        1: "A", 2: "B", 3: "C"}  
  
    for k, v := range m {  
        println(k, v)  
    }  
}
```





```
m := map[int]string{
    1: "A", 2: "B", 3: "C"}

for k := range m {
    if k%2 == 0 {
        delete(m, k)
    }
}
```



The iteration order over maps is not specified and is not guaranteed to be the same from one iteration to the next. If map entries that have not yet been reached are **removed during iteration**, the corresponding iteration values will not be produced. If map entries are **created during iteration**, that entry may be produced during the iteration or may be skipped. The choice may vary for each entry created and from one iteration to the next. If the map is nil, the number of iterations is 0.

# 5: FizzBuzz Results

For an arbitrary set of integers return a map such that

The key is the integer

If a number is divisible by 3 the value is Fizz

If a number is divisible by 5 the value is Buzz

If a number is divisible by 3 and 5 the value FizzBuzz

Otherwise the value is the number

Output the map



```
func main() {  
    for k, v := range fizzbuzz(1, 2, 4, 5, 6, 7, 8, 15, 21, 42) {  
        println(k, v)  
    }  
}  
  
func fizzbuzz(values ...int) map[int]string {  
    m := make(map[int]string, len(values))  
  
    for _, i := range values {  
        switch {  
        case i%3 == 0 && i%5 == 0:  
            m[i] = fmt.Sprintf("%s%s", fizz, buzz)  
        case i%3 == 0:  
            m[i] = fizz  
        case i%5 == 0:  
            m[i] = buzz  
        default:  
            m[i] = fmt.Sprintf("%d", i)  
        }  
    }  
  
    return m  
}
```

```

func main() {
    for k, v := range fizzbuzz(1, 2, 4, 5, 6, 7, 8, 15, 21, 42) {
        println(k, v)
    }
}

func fizzbuzz(values ...int) map[int]string {
    m := make(map[int]string, len(values))

    for _, i := range values {
        switch {
        case i%3 == 0 && i%5 == 0:
            m[i] = fmt.Sprintf("%s%s", fizz, buzz)
        case i%3 == 0:
            m[i] = fizz
        case i%5 == 0:
            m[i] = buzz
        default:
            m[i] = strconv.Itoa(i)
        }
    }

    return m
}

```

```

func main() {
    for k, v := range fizzbuzz(1, 2, 4, 5, 6, 7, 8, 15, 21, 42) {
        println(k, v)
    }
}

func fizzbuzz(values ...int) map[int]string {
    m := make(map[int]string, len(values))

    for _, i := range values {
        switch {
        case i%3 == 0 && i%5 == 0:
            m[i] = fmt.Sprintf("%s%s", fizz, buzz)
        case i%3 == 0:
            m[i] = fizz
        case i%5 == 0:
            m[i] = buzz
        default:
            m[i] = strconv.Itoa(i)
        }
    }

    return m
}

```

# Testing

# 6: Testing FizzBuzz

Write `fizzbuzz(int) string`  
Create a test suite for the function





# Go Convey



func



```
package main
```

```
func main() {  
    println("Hello, World!")  
}
```

```
package main
```

```
func main() {  
    hello()  
}
```

```
func hello() {  
    println("Hello, World!")  
}
```

```
package main
```

```
func main() {  
    hello("World!")  
}
```

```
func hello(n string) {  
    println("Hello,", n)  
}
```



```
package main
```

```
func main() {  
    println(square(3))  
}
```

```
func square(i int) int {  
    return i * i  
}
```



```
package main
```

```
func main() {  
    i, r := divide(5, 2)  
    println(i, r)  
}
```

```
func divide(x, y int) (int, int) {  
    i := x / y  
    r := x % y  
  
    return i, r  
}
```

```
package main
```

```
func main() {  
    i, r := divide(5, 2)  
    println(i, r)  
}
```

```
func divide(x, y int) (i int, r int) {  
    i = x / y  
    r = x % y  
  
    return  
}
```





```
package main
```

```
func main() {  
    i, r := divide(5, 2)  
    println(i, r)  
}
```

```
func divide(x, y int) (i int, r int) {  
    i = x / y  
    r = x % y  
  
    return i, r  
}
```



```
package main
```

```
func main() {  
    i, r := divide(5, 2)  
    println(i, r)  
}
```

```
func divide(x, y int) (i, r int) {  
    i = x / y  
    r = x % y  
  
    return i, r  
}
```



```
package main
```

```
func main() {  
    i, r := divide(5, 2)  
    println(i, r)  
}
```

```
func divide(x, y int) (i, r int) {  
    i, r = x/y, x%y  
    return i, r  
}
```



```
package main
```

```
func main() {  
    i, r := divide(5, 2)  
    println(i, r)  
}
```

```
func divide(x, y int) (int, int) {  
    return x/y, x%y  
}
```

```
package main
```

```
func main() {  
    i, r := divide(5, 0)  
    println(i, r)  
}
```

```
func divide(x, y int) (int, int) {  
    return x/y, x%y  
}
```



# Error Handling



```
package main
```

```
import "errors"
```

```
func main() {  
    i, r, err := divide(0, 5)  
    println(i, r, err)  
}
```

```
func divide(x, y int) (int, int, error) {  
    if y == 0 {  
        return 0, 0, errors.New("divide by zero")  
    }  
    return x / y, x % y, nil  
}
```



```
func main() {  
    r, err := f()  
  
    if err != nil {  
        println(err)  
    }  
  
    println(r)  
}  
  
func f() (int, error) {  
    if someCondition {  
        return 0, errors.New("some error condition has occurred")  
    }  
  
    return 1, nil  
}
```





```
func main() {  
    if r, err := f(); err != nil {  
        println(err)  
    } else {  
        println(r)  
    }  
}  
  
func f() (int, error) {  
    if someCondition {  
        return 0, errors.New("some error condition has occurred")  
    }  
  
    return 1, nil  
}
```



"Values can be programmed, and since errors are values, errors can be programmed."

<https://blog.golang.org/errors-are-values>



```
scanner := bufio.NewScanner(input)
for scanner.Scan() {
    token := scanner.Text()
    // process token
}
if err := scanner.Err(); err != nil {
    // process the error
}
```



```
scanner := bufio.NewScanner(input)
for scanner.Scan() {
    token := scanner.Text()
    if err := scanner.Err(); err != nil {
        // process the error, or maybe break
    }
    // process token
}
```



```
var (  
    CustomError = errors.New("some custom error")  
    DivideByZero = errors.New("divide by zero")  
)
```

```
var (  
    TryAgain = errors.New("try again")  
    SlowDown = errors.New("slow down")  
)  
  
func main() {  
    for {  
        switch err := doSomething(); err {  
        case nil:  
            break  
        case TryAgain:  
            continue  
        case SlowDown:  
            time.Sleep(1 * time.Second)  
        default:  
            return err // give up  
        }  
    }  
}
```



# defer



```
package main
```

```
func main() {  
    defer println("goodbye")  
    println("hello")  
}
```





```
func main() {  
    f, err := os.Open("/etc/passwd")  
    if err != nil {  
        log.Fatal(err)  
    }  
    defer f.Close()  
    // use f  
}
```



```
func main() {  
    defer println("one")  
    println("two")  
    defer println("three")  
}
```



```
package main
```

```
func main() {  
    i := 1  
    defer println(i)  
    i++  
    defer println(i)  
    println(i)  
}
```

```
package main
```

```
func main() {  
    for i := 0; i < 5; i++ {  
        println("open file", i)  
        defer println("close file", i)  
        println("use file", i)  
    }  
}
```



# 7: Deferred FizzBuzz

Iterate through a set of numbers, deferring the call to FizzBuzz  
Output the result in the call to defer



# closures

```
func main() {  
    a := func() int { return 123 }  
    b := func() int { return 123 }()  
}
```



```
func apply(f func(int) int, i int) int {  
    return f(i)  
}  
  
func main() {  
    double := func(x int) int { return x+x }  
    square := func(x int) int { return x*x }  
    i := apply(double, apply(square, 3))  
    println(i)  
}
```





```
func apply(f func(int) int, i int) int {  
    return f(i)  
}
```

```
func main() {  
    i := apply(func(x int) int { return x+x }, 3)  
    println(i)  
}
```



```
func apply(f func(int) int, i int) int {  
    return f(i)  
}
```

```
func main() {  
    n := 123  
    f := func(x int) int { return n }  
    i := apply(f, 3)  
    println(i)  
}
```



# 8: FizzBuzz Closures

- Iterate through a set of numbers
- Pass the number to a fizz closure
- Pass the number to a buzz closure
- Pass the number to a number closure
- Have the program output FizzBuzz
- Verify through testing



```

package main

import (
    "strconv"
    "fmt"
)

const (
    Fizz = "Fizz"
    Buzz = "Buzz"
)

var fizz func(int) string = func(i int) string {
    if i%3 == 0 {
        return Fizz
    }

    return ""
}

var buzz func(int) string = func(i int) string {
    if i%5 == 0 {
        return Buzz
    }

    return ""
}

```

```

var number func(int) string = func(i int) string
{
    if i%3 != 0 && i%5 != 0 {
        return strconv.Itoa(i)
    }

    return ""
}

func main() {
    for _, i := range []int{1, 2, 4, 5, 6, 7, 8,
15, 21, 42} {
        fmt.Printf("%d - %s%s%s\n", i, fizz(i),
buzz(i), number(i))
    }
}

```

# Panic and Recover



```
package main
```

```
func foo() {  
    panic("kaboom")  
}
```

```
func main() {  
    foo()  
    println("hello")  
}
```



```
func foo() {  
    defer func() {  
        if x := recover(); x != nil {  
            fmt.Printf("recovered: %v\n", x)  
        }  
    }()  
    panic("kaboom")  
}
```



# Don't Panic!





# Packages

```
package main
```

```
func main() {  
    println("Hello, world!")  
}
```



```
package foo
```

```
func bar() {  
    println("Hello, world!")  
}
```



```
package foo
```

```
func bar() {  
    println("Hello, world!")  
}
```

```
func Bar() {  
    println("Hello, world!")  
}
```



```
package foo
```

```
func bar() {  
    println("Hello, world!")  
}
```

```
func Bar() {  
    bar()  
}
```



```
package foo
```

```
var Hello = "Hello"
```

```
const World = "World"
```

```
var internal = "foo"
```



```
package main
```

```
import "fmt"
```

```
func main() {  
    fmt.Println("Hello, World")  
}
```

```
package main
```

```
import (  
    "fmt"  
    "time"  
)
```

```
func main() {  
    fmt.Printf("The time is now: %s", time.Now())  
}
```



```
package main
```

```
import (  
    "fmt"  
    stdtime "time"  
)
```

```
func main() {  
    fmt.Printf("The time is now: %s", stdtime.Now())  
}
```

```
package main
```

```
import (  
    "fmt"  
    _ "github.com/go-mysql/mysql"  
)
```

```
func main() {  
    // ...  
}
```



```
package main
```

```
import (  
    "fmt"  
    . "time"  
)
```

```
func main() {  
    fmt.Printf("The time is now: %s", Now())  
}
```

# Creating Packages





# godoc

# 9: Roman Numerals

Create a package to handle Roman numerals  
Allow conversion from int to Roman numeral string  
Allow conversion from Roman numeral string to int  
Should be fully documented

Should be fully tested

Bonus: Allow mathematical functions on Roman numerals



go get





```
$ go get github.com/domdavis/foo
```

```
# roughly equivalent to...
```

```
$ cd $GOPATH/src/github.com/domdavis
```

```
$ git clone https://github.com/domdavis/foo
```

```
$ go install github.com/domdavis/foo
```



# Pointer Dereferencing



```
foo := &[]int{1, 2, 3}  
// bar := &foo[1]  
// bar := *foo[1]  
bar := *foo  
println(bar[1])
```



# types



```
package main

type name string
type username string

func main() {
    n := name("Dom")
    u := username("david")

    if n == u { // compile error
        println("Items match!")
    }
}
```



```
package main
```

```
type name string
```

```
type username string
```

```
func main() {
```

```
    n := name("Dom")
```

```
    u := username("david")
```

```
    if n == name(u) { // fixed  
        println("Items match!")  
    }
```

```
}
```



```
package main

type name string
type username string

func main() {
    n := name("Dom")
    u := username("david")

    if username(n) == u {
        println("Items match!")
    }
}
```



```
package main
```

```
type name string
```

```
type username string
```

```
func main() {
```

```
    n := name("Dom")
```

```
    u := username("david")
```

```
    if string(n) == string(u) {
```

```
        println("Items match!")
```

```
    }
```

```
}
```





```
type binaryOp func(int, int) int

func add(i, j int) int { return i+j }
func sub(i, j int) int { return i-j }
func mul(i, j int) int { return i*j }
func div(i, j int) int { return i/j }
func mod(i, j int) int { return i%j }

func calc(op binaryOp, i, j int) int {
    return op(i, j)
}
```



# 10: FizzBuzz Type

```
package main

import (
    "fmt"

    "path/to/your/fizzbuzz"
)

func main() {
    for i := 0; i < 100; i++ {
        fmt.Printf("%d - %s%s%s\n", i,
            fizzbuzz.FizzBuzz(fizzbuzz.Fizz, i),
            fizzbuzz.FizzBuzz(fizzbuzz.Buzz, i),
            fizzbuzz.FizzBuzz(fizzbuzz.Number, i),
        )
    }
}
```



# structs



```
type FizzBuzzed struct {  
    input int  
    output string  
}
```

```
package main

import "fmt"

type FizzBuzzed struct {
    input  int
    output string
}

func main() {
    v := FizzBuzzed{}
    fmt.Printf("%v/n", v)
}
```



```
package main

import "fmt"

type FizzBuzzed struct {
    input  int
    output string
}

func main() {
    v := FizzBuzzed{}
    v.input = 15
    v.output = "FizzBuzz"
    fmt.Printf("%v/n", v)
}
```

```
package main

import "fmt"

type FizzBuzzed struct {
    input  int
    output string
}

func main() {
    v := FizzBuzzed{
        input: 15,
        output: "FizzBuzz",
    }
    fmt.Printf("%v/n", v)
}
```



```
package main

import "fmt"

type FizzBuzzed struct {
    input  int
    output string
}

func main() {
    v := FizzBuzzed{input: 15}
    v.output = "FizzBuzz"
    fmt.Printf("%v/n", v)
}
```





```
type Value struct {  
    input  int  
    output string  
}
```

```
type Conversion struct {  
    Value  
    Converter string  
}
```



```
v := Value{1, "1"}  
c := Conversion{v, "FizzBuzz"}
```

```
c := Conversion{Value{1, "1"}, "FizzBuzz"}
```

```
c := Conversion{}  
c.input = 1  
c.output = "1"  
c.Converter = "FizzBuzz"
```



```
package main

type Value struct {
    input  int
    output string
}

func Print(v Value) {
    println(v.input, v.output)
}

func main() {
    v := Value{1, "1"}
    Print(v)
}
```

```
package main

type Value struct {
    input  int
    output string
}

func (v Value) Print() {
    println(v.input, v.output)
}

func main() {
    v := Value{1, "1"}
    v.Print()
}
```

```
package main

type Value struct {
    input  int
    output string
}

func (v Value) Print() {
    println(v.input, v.output)
}

func (v Value) Output(o string) {
    v.output = o
}

func main() {
    v := Value{1, "1"}
    v.Output("FizzBuzz")
    v.Print()
}
```

```
package main

type Value struct {
    input  int
    output string
}

func (v *Value) Print() {
    println(v.input, v.output)
}

func (v *Value) Output(o string) {
    v.output = o
}

func main() {
    v := &Value{1, "1"}
    v.Output("FizzBuzz")
    v.Print()
}
```



```
package main

type Value struct {
    input  int
    output string
}

func (v *Value) Print() {
    println(v.input, v.output)
}

func (v *Value) Output(o string) {
    v.output = o
}

func main() {
    v := Value{1, "1"}
    v.Output("FizzBuzz")
    v.Print()
}
```

# Pointers

// Value types

```
var a thing
```

```
var b thing = make(thing)
```

```
var c thing = thing{}
```

```
var d = make(thing)
```

```
var e = thing{}
```

```
f := make(thing)
```

```
g := thing{}
```

// Pointer types

```
var a *thing
```

```
var b *thing = new(thing)
```

```
var c *thing = &thing{}
```

```
var d = new(thing)
```

```
var e = &thing{}
```

```
f := new(thing)
```

```
g := &thing{}
```



# Pointers

// Value types

~~var a thing~~

~~var b thing = make(thing)~~

~~var c thing = thing{}~~

~~var d = make(thing)~~

~~var e = thing{}~~

~~f := make(thing)~~

g := thing{}

// Pointer types

~~var a \*thing~~

~~var b \*thing = new(thing)~~

~~var c \*thing = &thing{}~~

~~var d = new(thing)~~

~~var e = &thing{}~~

~~f := new(thing)~~

g := &thing{}



# Welcome Back



# 11: FizzBuzz Type Method

```
package main

import (
    "fmt"

    "path/to/your/fizzbuzz"
)

func main() {
    for i := 0; i < 100; i++ {
        fmt.Printf("%d - %s%s%s\n", i,
            fizzbuzz.Fizz.Translate(i),
            fizzbuzz.Buzz.Translate(i),
            fizzbuzz.Number.Translate(i),
        )
    }
}
```



# Interfaces



```
type duck interface {  
    walk()  
    quack()  
}
```

```
type duck interface {  
    walk()  
    quack()  
}
```

```
type goose struct{}
```

```
func (g goose) walk() { println("waddle") }  
func (g goose) quack() { println("HONK") }  
func (g goose) eat()   { println("om nom nom") }
```





```
var d duck // initially nil  
d = goose{} // d may now be used  
d.walk()  
d.quack()
```

```
func annoy(d duck) {  
    d.walk()  
    d.quack()  
    //d.eat() // doesn't work  
}  
  
func main() {  
    g := goose{}  
    annoy(g)  
}
```



```
var walker interface{  
    walk()  
}
```

```
walker = goose{}  
walker.walk()
```



```
type Reader interface {  
    Read(p []byte) (n int, err error)  
}
```

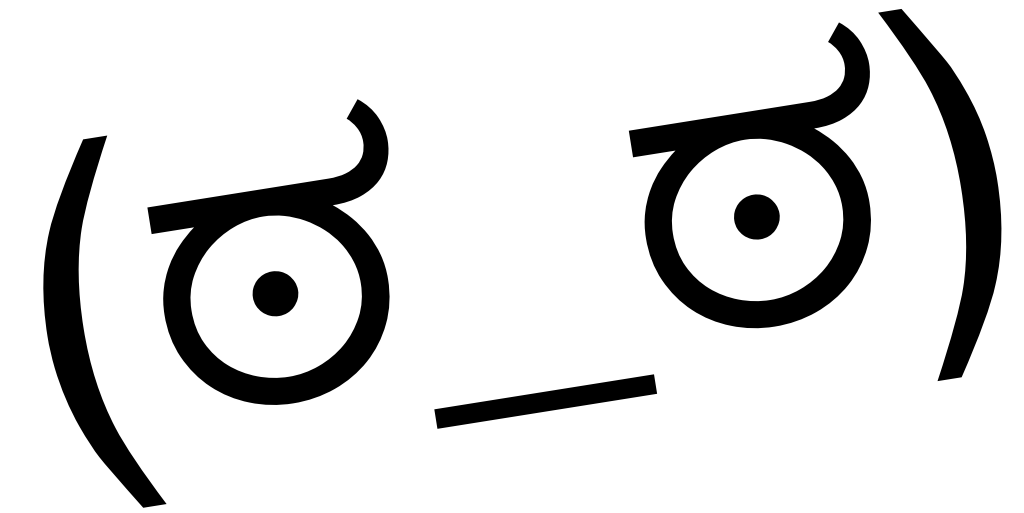
```
type Writer interface {  
    Write(p []byte) (n int, err error)  
}
```

```
type ReadWriter interface {  
    Reader  
    Writer  
}
```



# Interfaces Should Be Small

```
type Store interface {  
    Put(key string, value []byte, options *WriteOptions) error  
    Get(key string) (*KVPair, error)  
    Delete(key string) error  
    Exists(key string) (bool, error)  
    Watch(key string, stopCh <-chan struct{}) (<-chan *KVPair, error)  
    WatchTree(directory string, stopCh <-chan struct{}) (<-chan []*KVPair, error)  
    NewLock(key string, options *LockOptions) (Locker, error)  
    List(directory string) ([]*KVPair, error)  
    DeleteTree(directory string) error  
    AtomicPut(key string, value []byte, previous *KVPair, options *WriteOptions) (bool, *KVPair, error)  
    AtomicDelete(key string, previous *KVPair) (bool, error)  
    Close()  
}
```



```
type NotVeryUseful interface{}
```

```
var i interface{  
i = 123          // valid  
i = "hello"     // valid  
i = true        // valid  
fmt.Println(i)
```



```
func foo(v interface{}) {  
    // ???  
}
```



```
func foo(v interface{}) {  
    i := v.(int)  
    println("int", i)  
}
```



```
func foo(v interface{}) {  
    if i, ok := v.(int); ok {  
        println("int", i)  
    } else {  
        println("not an int")  
    }  
}
```



```
func foo(v interface{}) {  
    switch v := v.(type) {  
    case int:  
        println("int", v)  
    case string:  
        println("string", v)  
    case io.Reader:  
        println("io.Reader")  
    default:  
        println("unknown")  
    }  
}
```



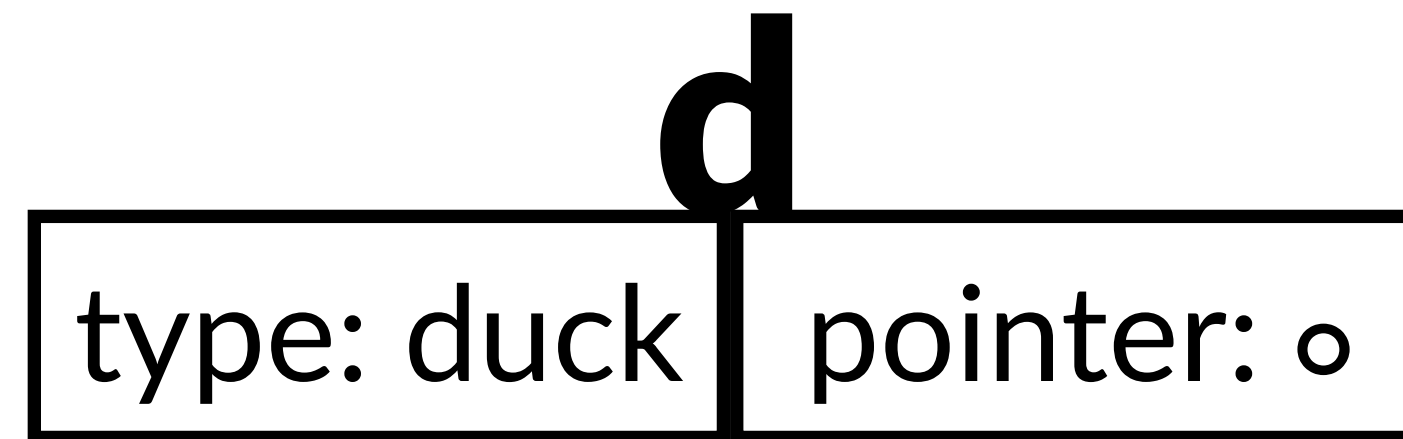
```
switch x := v.(type) {  
case int, float32:  
    println("int", x + x)  
}
```



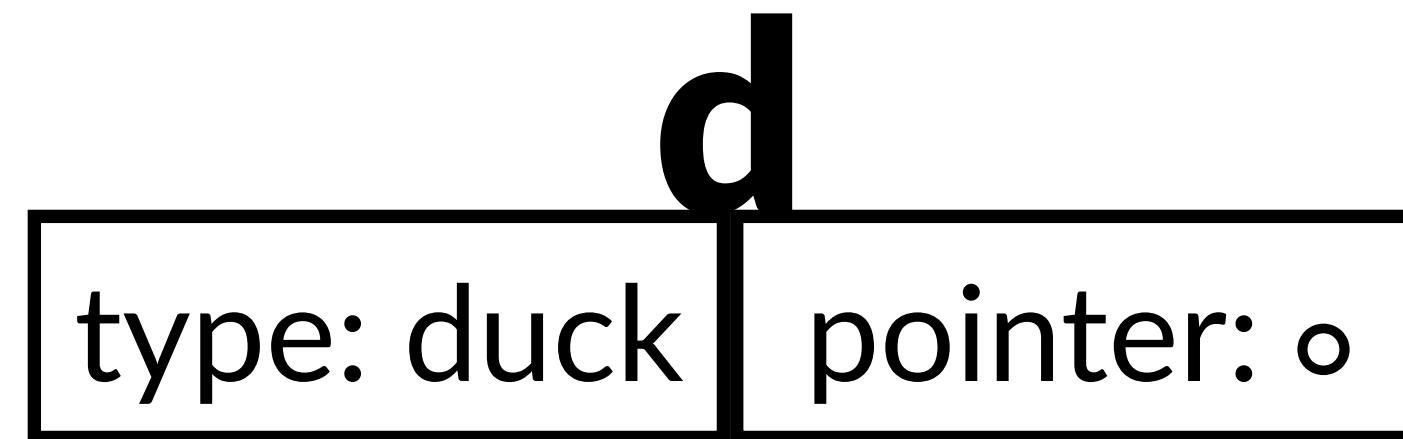
# This is not generics



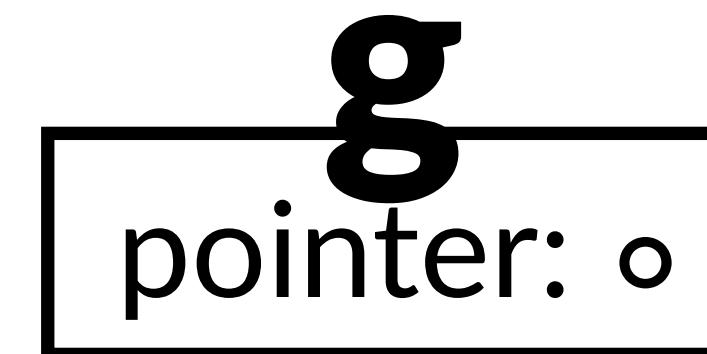
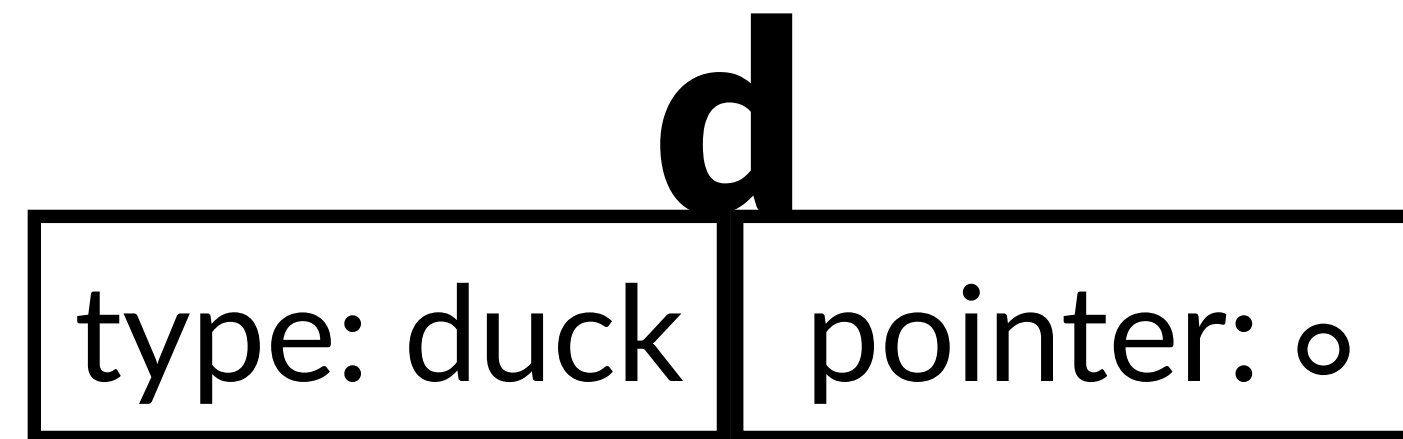
**var** d duck



```
var d duck  
//d.quack()
```

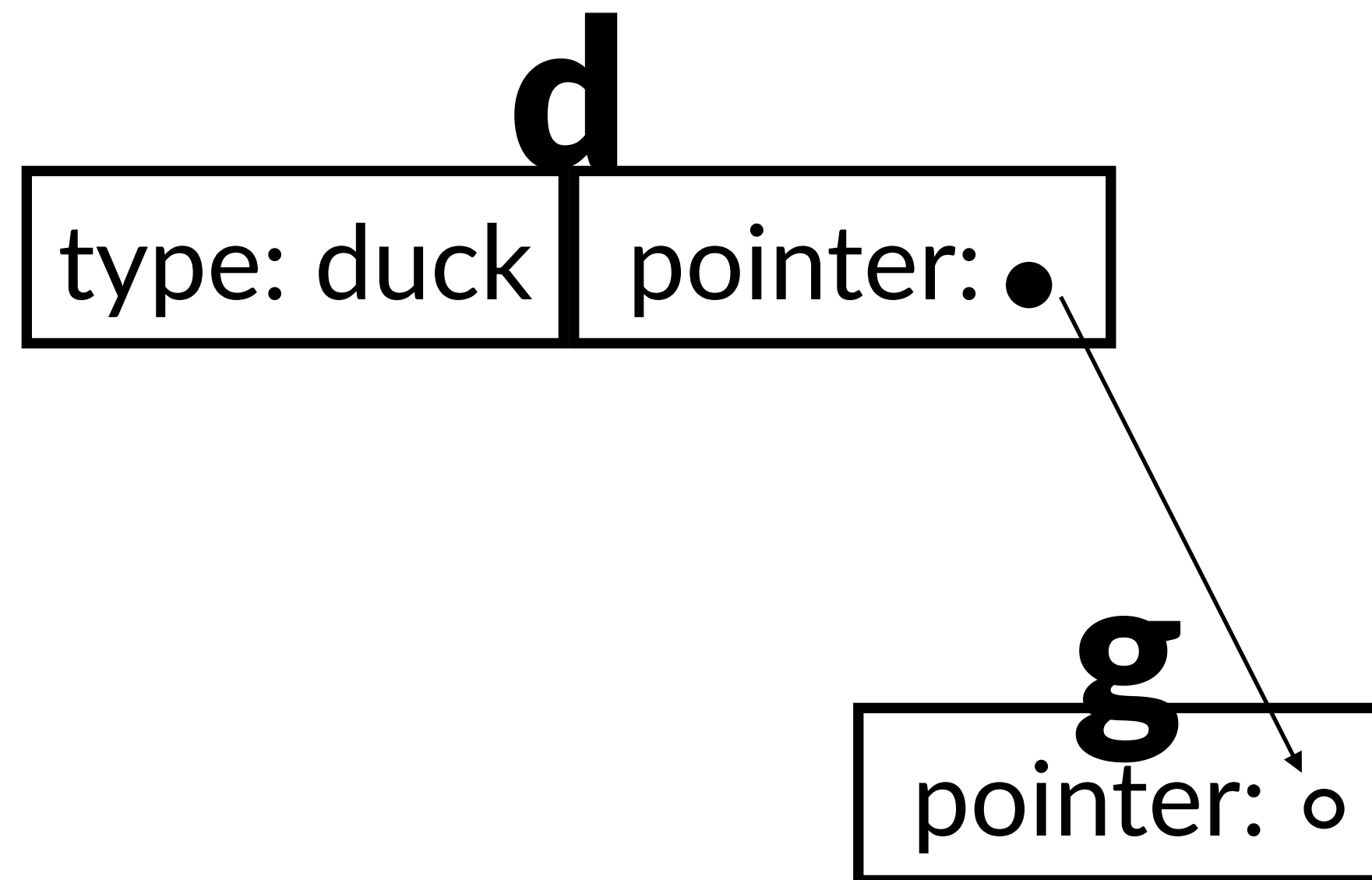


```
var d duck  
//d.quack()  
var g *goose
```

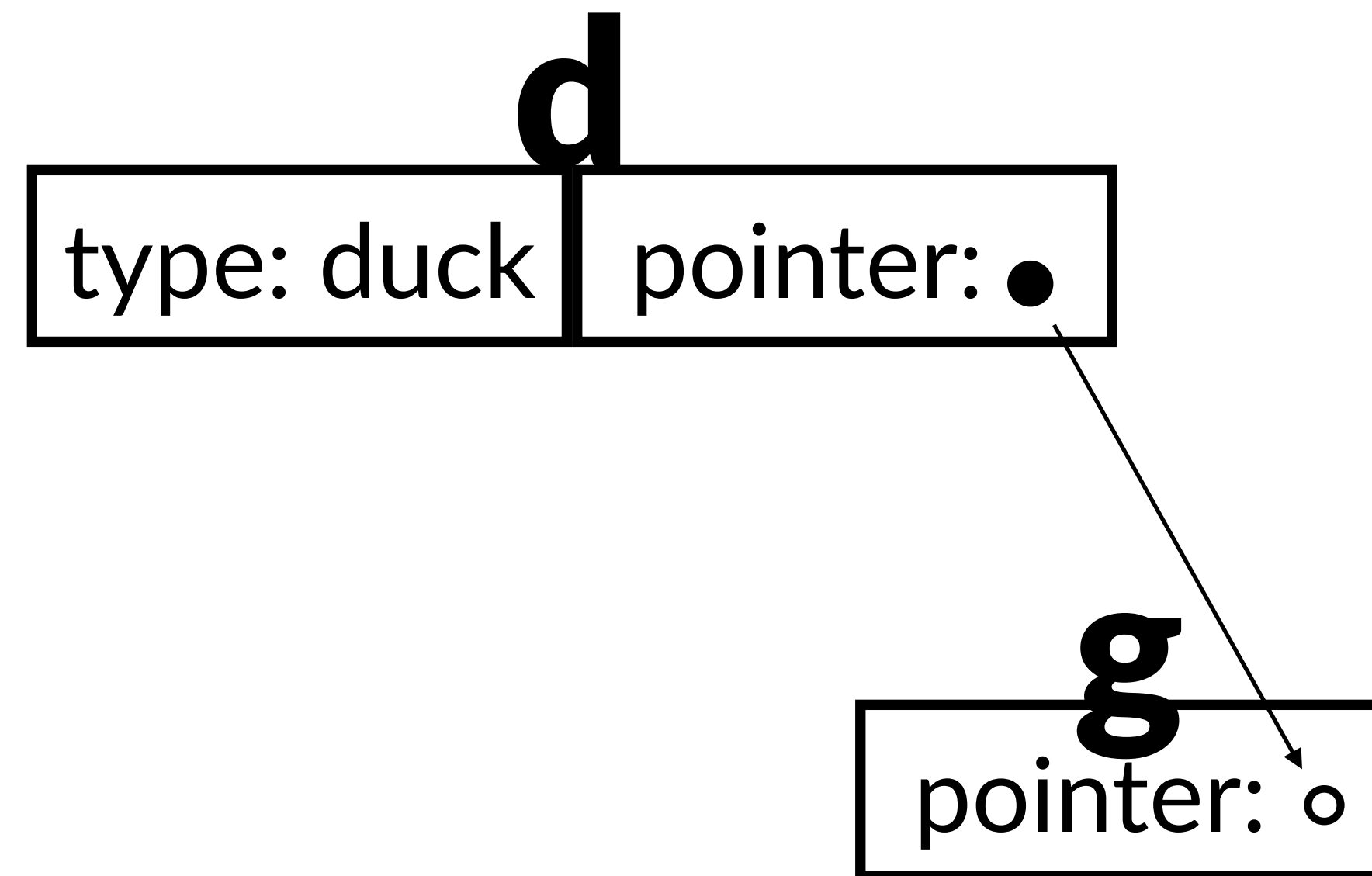




```
var d duck  
//d.quack()  
var g *goose  
d = g
```



```
var d duck
//d.quack()
var g *goose
d = g
d.quack() // ?
```



```
func (g goose) quack() {  
    // it's not possible to get here via a nil ptr  
    // "value method main.goose.quack called using  
    // nil *goose pointer"  
}
```

```
func (g *goose) quack() {  
    // it is possible to get here with a nil ptr!  
    // g may be nil  
}
```



# Interfaces and Testing



Whenever possible, write code in a functional style.

Whenever possible, write code in a functional style.  
Take all dependencies as parameters.

Whenever possible, write code in a functional style.  
Take all dependencies as parameters.  
Avoid depending on or (especially) mutating global state.

Whenever possible, write code in a functional style.  
Take all dependencies as parameters.  
Avoid depending on or (especially) mutating global state.  
Make heavy use of interfaces!



```
package main

func process(db *database) (result, error) {
    rows, err := db.Query("SELECT foo")
    if err != nil {
        return result{}, err
    }
    defer rows.Close()
    var r result
    if err := rows.Scan(&r); err != nil {
        return result{}, err
    }
    return r, nil
}

func main() {
    db := newDatabase()
    r, err := process(db)
}
```

```
package main

type queryer interface {
    Query(s string) (rows, error)
}

func process(db *queryer) (result, error) {
    rows, err := db.Query("SELECT foo")
    if err != nil {
        return result{}, err
    }
    defer rows.Close()
    var r result
    if err := rows.Scan(&r); err != nil {
        return result{}, err
    }
    return r, nil
}

func main() {
    db := newDatabase()
    r, err := process(db)
}
```

```
type fakeQueryer struct{}  
  
func (q fakeQueryer) Query(s string) (rows, error) {  
    return []row{"fakewrow"}, nil  
}
```

```
type fakeQueryer struct{}

func (q fakeQueryer) Query(s string) (rows, error) {
    return []row{"fakewrow"}, nil
}

func TestProcess(t *testing.T) {
    q := fakeQueryer{}
    have, err := process(q)
    if err != nil {
        t.Fatal(err)
    }
    want := result{"fakedata"} // or whatever
    if want != have {
        t.Errorf("process: want %v, have %v", want, have)
    }
}
```

# 12: Putting it all together

Create an integer translation package

Package will convert an int to a string

Package uses an arbitrary conversion function

The returned result should contain the input

The returned result should contain the conversion type

Provide FizzBuzz and Roman numeral converters



```
f ( ) {  
    sleep "$1"  
    echo "$1"  
}  
while [ -n "$1" ]  
do  
    f "$1" &  
    shift  
done  
wait
```



```
package main

import (
    "fmt"
    "sort"
)

func main() {
    strs := []string{"c", "a", "b"}
    sort.Strings(strs)
    fmt.Println("Strings:", strs)

    ints := []int{7, 2, 4}
    sort.Ints(ints)
    fmt.Println("Ints: ", ints)

    s := sort.IntsAreSorted(ints)
    fmt.Println("Sorted: ", s)
}
```

```
package main

import (
    "fmt"
    "sort"
)

type ByLength []string

func (s ByLength) Len() int {
    return len(s)
}

func (s ByLength) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}

func (s ByLength) Less(i, j int) bool {
    return len(s[i]) < len(s[j])
}

func main() {
    fruits := []string{"peach", "banana", "kiwi"}
    sort.Sort(ByLength(fruits))
    fmt.Println(fruits)
}
```



KISS  
DRY  
SOLID



@KevlinHenney

# JSON Parsing



framework training



# Concurrency & Parallelism

# Concurrency $\neq$ Parallelism

Concurrency is about designing your program so that multiple things can execute independently of each other.

Concurrency is about designing your program so that multiple things can execute independently of each other.  
Parallelism is executing those things at the same time.



Concurrency is about designing your program so that multiple things can execute independently of each other.

Parallelism is executing those things at the same time.

Go programs should be written for concurrency, but parallelism is a decision for the runner!



```
package main
```

```
func main() {  
    foo("a")  
    go foo("b")  
}
```

```
func foo(s string) {  
    println(s)  
}
```

```
package main

import "time"

func main() {
    foo("a")
    go foo("b")
    time.Sleep(time.Second)
}

func foo(s string) {
    println(s)
}
```



# Channels

Don't communicate by sharing memory

Don't communicate by sharing memory  
Share memory by communicating

Don't communicate by sharing memory  
Share memory by communicating  
A channel is like a UNIX pipe

Don't communicate by sharing memory  
Share memory by communicating  
A channel is like a UNIX pipe  
Typed conduit for information, typically between goroutines



```
package main
```

```
func main() {  
    c := make(chan int)  
    go compute(c)  
    println(<-c)  
}
```

```
func compute(c chan int) {  
    c <- 123  
}
```



```
package main
```

```
func main() {  
    c := make(chan int, 100)  
    go compute(c)  
    println(<-c)  
}
```

```
func compute(c chan int) {  
    c <- 123  
}
```



```
package main
```

```
func main() {  
    c := make(chan int)  
    for i := 0; i < 10; i++ {  
        go compute(i, c)  
    }  
    for i := 0; i < 10; i++ {  
        println(i, <-c)  
    }  
}
```

```
func compute(id int, c chan int) {  
    c <- id  
}
```



```
package main

func main() {
    c := make(chan int)
    for i := 0; i < 10; i++ {
        go readOne(c)
    }
    c <- 123
    c <- 456
    c <- 789
}

func readOne(c chan int) {
    println(<-c)
}
```

```
package main

func main() {
    c := make(chan int)
    for i := 0; i < 10; i++ {
        go readOne(c)
    }
    c <- 123
    c <- 456
    c <- 789
    close(c)
}

func readOne(c chan int) {
    println(<-c)
}
```

```
package main

import "time"

func main() {
    c := make(chan int)
    for i := 0; i < 10; i++ {
        go readOne(c)
    }
    c <- 123
    c <- 456
    c <- 789
    close(c)
    time.Sleep(time.Second)
}

func readOne(c chan int) {
    println(<-c)
}
```

```
v, ok := <-c
if ok {
    println("received value", v)
} else {
    println("channel was closed")
}
```



```
for v := range c {  
    println("received value", v)  
}
```





# 13: Concurrent FizzBuzz

Implement FizzBuzz so it solves for 1-15 concurrently

# Select



```
select {  
  case v1 := <-c1:  
    // use value v1  
  case v2 := <-c2:  
    // use value v2  
}
```

```
select {  
  case v1 := <-c1:  
    // use value v1  
  case v2 := <-c2:  
    // use value v2  
  default:  
    // no channel was ready  
}
```



```
func doWork(i chan int, s chan string, t chan thing) {  
    for {  
        select {  
        case v := <-i:  
            println("do work with int", i)  
        case v := <-s:  
            println("do work with string", s)  
        case v := <-t:  
            println("do work with thing", t)  
        }  
    }  
}
```



```
func (a *Actor) SendEvent(e Event) {  
    a.eventc <- e  
}  
  
func (a *Actor) SendReq(r *Request) {  
    a.requestc <- r  
}  
  
func (a *Actor) Stop() {  
    close(a.quitc)  
}  
  
func (a *Actor) loop() {  
    for {  
        select {  
            case e := <-a.eventc:  
                // process event  
            case r := <-a.requestc:  
                // process request  
            case <-a.quitc:  
                return  
        }  
    }  
}
```

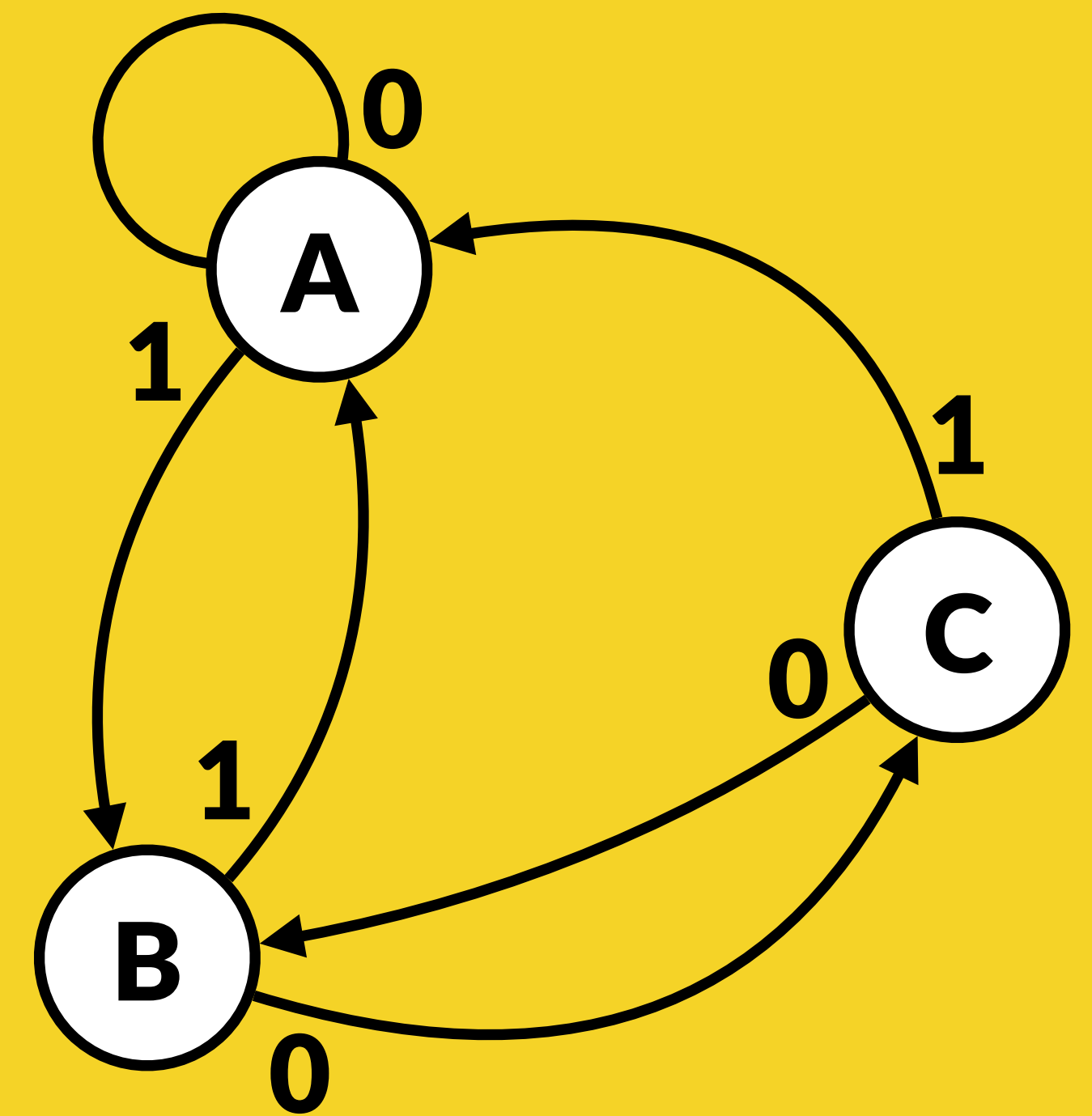
```
type Actor struct {  
    eventc chan Event  
    requestc chan *Request  
    quetc chan struct{  
    }  
}
```

# 14: State Machine

```
package main

// Implement stateMachine.
type stateMachine struct{}

func main() {
    sm := newStateMachine()
    sm.send(1)           // "state A + 1 => state B"
    sm.send(0)           // "state B + 0 => state C"
    println(sm.state()) // "state C"
}
```





# Pipelines



```
package main

import "strings"

func trim(in chan string, out chan string) {
    for s := range in {
        s = strings.TrimSpace(s)
        out <- s
    }
}

func capitalize(in chan string, out chan string) {
    for s := range in {
        s = strings.ToUpper(s)
        out <- s
    }
}
```



```
package main

import "strings"

func trim(in <-chan string, out chan<- string) {
    for s := range in {
        s = strings.TrimSpace(s)
        out <- s
    }
}

func capitalize(in <-chan string, out chan<- string) {
    for s := range in {
        s = strings.ToUpper(s)
        out <- s
    }
}
```

```
func main() {  
    a, b, c := make(chan string), make(chan string), make(chan string)  
    go trim(a, b);  
    go capitalize(b, c)  
  
    a <- " hello world "  
    fmt.Printf("%q", <-c)  
  
    close(a)  
    close(b)  
    close(c)  
}
```

```
func main() {  
    a, b, c := make(chan string), make(chan string), make(chan string)  
    go trim(a, b); go trim(a, b); go trim(a, b); go trim(a, b);  
    go capitalize(b, c)  
  
    a <- " hello world "  
    fmt.Printf("%q", <-c)  
  
    close(a)  
    close(b)  
    close(c)  
}
```

# 15: FizzBuzz Pipelines

```
type converter func(in <-chan int, out chan<- int)
var converters = []converter{fizz, buzz, number}

func main() {
    chans := make([]chan int, 4)

    for i := range chans {
        chans[i] = make(chan int)
        defer close(chans[i])
    }

    for i := 0; i < len(chans)-1; i++ {
        go f(chans[i], chans[i+1])
    }

    for i := 1; i < 100; i++ {
        chans[0] <- i
        fmt.Printf("%d\n", <-chans[len(chans)-1])
    }
}
```

# Mutex



```
package main

type thing struct {
    m map[int]int
}

func newThing() *thing {
    return &thing{
        m: map[int]int{},
    }
}

func (t *thing) set(k, v int) {
    t.m[k] = v
}

func (t *thing) get(k int) int {
    return t.m[k]
}
```



```
package main

import "sync"

type thing struct {
    mtx sync.Mutex
    m    map[int]int
}

func newThing() *thing {
    return &thing{
        m: map[int]int{},
    }
}

func (t *thing) set(k, v int) {
    t.mtx.Lock()
    defer t.mtx.Unlock()
    t.m[k] = v
}

func (t *thing) get(k int) int {
    t.mtx.Lock()
    defer t.mtx.Unlock()
    return t.m[k]
}
```

```
package main

import "sync"

type thing struct {
    mtx sync.RWMutex
    m    map[int]int
}

func newThing() *thing {
    return &thing{
        m: map[int]int{},
    }
}

func (t *thing) set(k, v int) {
    t.mtx.Lock()
    defer t.mtx.Unlock()
    t.m[k] = v
}

func (t *thing) get(k int) int {
    t.mtx.RLock()
    defer t.mtx.RUnlock()
    return t.m[k]
}
```

```
package main
```

```
func main() {
```

```
    n := 10
```

```
    for i := 0; i < n; i++ {
```

```
        go println(i)
```

```
    }
```

```
    // when is everything done?
```

```
}
```

```
package main

import "sync"

func main() {
    var wg sync.WaitGroup
    n := 10
    for i := 0; i < n; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            println(i)
        }()
    }

    wg.Wait()
}
```

```
package main

import (
    "fmt"
    "sync"
)

func main() {
    var wg sync.WaitGroup
    n := 10
    wg.Add(n)
    for i := 0; i < n; i++ {
        go func() {
            defer wg.Done()
            fmt.Println(i)
        }()
    }

    wg.Wait()
}
```

```
package main

import (
    "fmt"
    "sync"
)

func main() {
    var wg sync.WaitGroup
    n := 10
    wg.Add(n)
    for i := 0; i < n; i++ {
        go func(i int) {
            defer wg.Done()
            fmt.Println(i)
        }(i)
    }

    wg.Wait()
}
```

# Parallelism



goroutines are multiplexed onto OS threads



goroutines are multiplexed onto OS threads  
GOMAXPROCS controls number of threads available

goroutines are multiplexed onto OS threads  
GOMAXPROCS controls number of threads available  
By default, GOMAXPROCS = num CPUs



# 16: Playtime

[https://divan.github.io/posts/go\\_concurrency\\_visualize/](https://divan.github.io/posts/go_concurrency_visualize/)



# Networking



```
resp, err := http.Get("http://google.com")  
if err != nil {  
    panic(err)  
}
```



```
resp, err := http.Get("http://google.com")
if err != nil {
    panic(err)
}

defer resp.Body.Close()

buf, err := ioutil.ReadAll(resp.Body)
if err != nil {
    panic(err)
}
fmt.Printf("%s\n", buf)
```

```
resp, err := http.Get("http://google.com")
if err != nil {
    panic(err)
}

defer resp.Body.Close()

io.Copy(os.Stdout, resp.Body)
```



```
c := http.Client{} // zero value is usable
```

```
req, err := http.NewRequest("GET", "http://google.com", nil)  
if err != nil {  
    panic(err)  
}
```

```
resp, err := c.Do(req)  
if err != nil {  
    panic(err)  
}
```



```
func main() {  
    http.HandleFunc("/", h)  
    http.ListenAndServe(":8080", nil)  
}  
  
func h(w http.ResponseWriter, r *http.Request) {  
    fmt.Fprintf(w, "Hello world\n")  
}
```



```
func main() {  
    s := &server{msg: "Hello from server"}  
    http.Handle("/", s)  
    http.ListenAndServe(":8080", nil)  
}  
  
type server struct{ msg string }  
  
func (s *server) ServeHTTP(w http.ResponseWriter, r *http.Request) {  
    log.Printf("%s %s from %s", r.Method, r.URL, r.RemoteAddr)  
    fmt.Fprintf(w, s.msg+"\n")  
}
```

```
mux := http.NewServeMux()  
mux.HandleFunc("/foo", handleFoo)  
mux.HandleFunc("/bar", handleBar)  
log.Fatal(http.ListenAndServe(":8080", mux))
```



# Other Routers and Mixers

Usability: [github.com/gorilla/mux](https://github.com/gorilla/mux)

Raw speed: [github.com/julienschmidt/httprouter](https://github.com/julienschmidt/httprouter)

# 17: FizzBuzz Microservice



```
ln, err := net.Listen("tcp4", ":1234")
    if err != nil {
        panic(err)
    }
    defer ln.Close()

    for {
        c, err := ln.Accept()
        if err != nil {
            break
        }
        go handle(c)
    }
}
```



```
func handle(c net.Conn) {  
    log.Printf("%s: start conn", c.RemoteAddr())  
    defer log.Printf("%s: close conn", c.RemoteAddr())  
  
    s := bufio.NewScanner(c)  
    for s.Scan() {  
        log.Printf("%s: %s", c.RemoteAddr(), s.Text())  
    }  
}
```



```
func handle(c net.Conn) {  
    log.Printf("%s: start conn", c.RemoteAddr())  
    defer log.Printf("%s: close conn", c.RemoteAddr())  
  
    s := bufio.NewScanner(c)  
    for s.Scan() {  
        log.Printf("%s: %s", c.RemoteAddr(), s.Text())  
        fmt.Fprintf(c, "%s\n", strings.ToUpper(s.Text()))  
    }  
}
```



# Go Nuts

