SHOULD I STAY OR SHOULD I GO?

Ralf Wirdemann - kommitment GmbH & Co. KG

```
- (RACSignal *)getAllGroupsV2 {
    RACSignal *signal = [self.communicator getGroupsV2];
    return [RACSignal createSignal:^RACDisposable *(id subscri
        [signal subscribeNext:^(NSDictionary *data) {
            NSManagedObjectContext *context =
                [PQCoreDataManager sharedManager].managedConte
            [PQGroup createOrUpdateWithContext:context data:da
        } error:^(NSError *error){
            [subscriber sendError:error];
        } completed:^{
            [[PQCoreDataManager sharedManager] saveContext];
            [subscriber sendCompleted];
        }];
        return nil;
    }];
}
```

```go
func getAllGroupsV2() {
    if resp, err := http.Get("http://picue.de/groups"); err !=
        panic(err)
    }

    var groups []*domain.Group
    json := resp.Body
    decoder := json.NewDecoder(json)
    decode.Decode(&groups)
    database.CreateGroups(&groups)
}
```

LESBARKEIT

MEINE GESCHICHTE

1992

```
int main() {
    printf("Hello World\n");
    return 0;
}
```

1994

```
main() {
     cout << "Hello World" << endl;
}
```

2000

```
public class HelloWorld {

    public static void main (String[] args) {
        System.out.println("Hello World");
    }
}
```

2008

```
puts 'Hello World'
```

2016

```
int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init
    NSLog(@"Hello World!");
    [pool drain];
    return 0;
}
```

2016

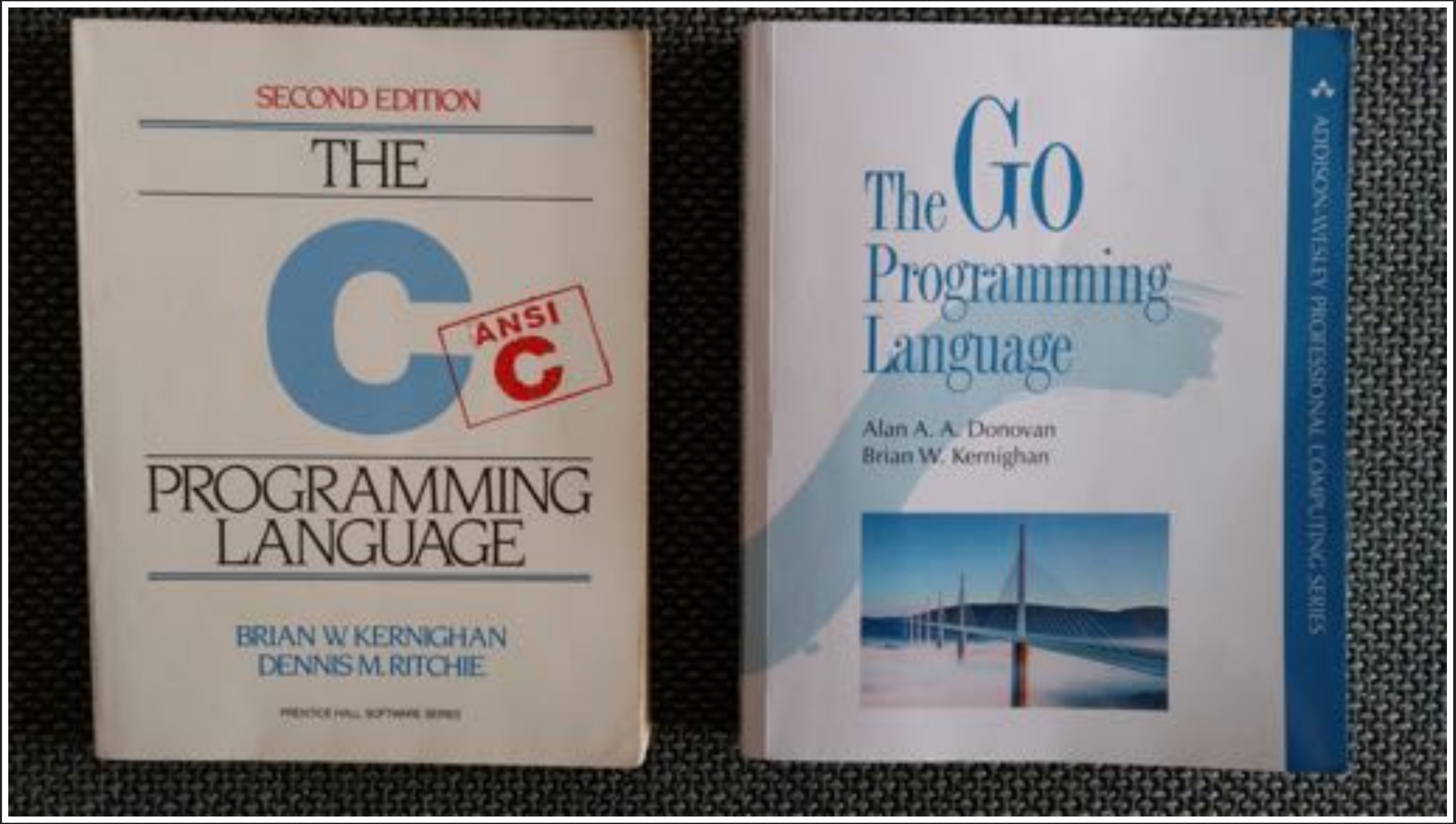
```
print("Hello World")
```

2016

```
func main() {
    print("Hello World\n")
}


int main() {
    printf("Hello World\n");
    return 0;
}
```

40 JAHRE SPÄTER



WORUM ES MIR HEUTE GEHT

Fragen beantworten, die ich mir selber gestellt habe

10% der Anwesenden wollen Go probieren

DIE PROGRAMMIERSPRACHE GO

Geschichte

Tour

Eigenschaften

Toolchain

Wer, wofür, warum?

GESCHICHTE

2007: Frust Komplextität von Google-Software

Ken Thompson, Rob Pike, Robert Griesemer

Idee: Einfache, compilierbare Programmiersprache, die heutigen Rechensystemen gerecht wird, z.B. für einfache skalierbare Netzwerkdienst

2009: Erste öffentliche Version im November

STATISCH GETYPT

```go
s := "Hallo"        // string
i := 42             // int
f := 3.142          // float64
g := 0.867 + 0.5i // complex128
```

25 KEYWORDS

`break`	`default`	`func`	`interface`	`sele`
`case`	`defer`	`go`	`map`	`stru`
`chan`	`else`	`goto`	`package`	`swit`
`const`	`fallthrough`	`if`	`range`	`type`
`continue`	`for`	`import`	`return`	`var`

A TOUR OF GO

PACKAGES

```
$GOROOT/picue/domain/groups.go
                     media.go
```

```
$GOROOT/picue/domain/groups.go
                     media.go

package domain

type Group struct {
...
}
```

```
$GOROOT/picue/domain/groups.go
                     media.go

package domain

type Group struct {
...
}

package domain

type Media struct {
...
}
```

FUNCTIONS

```go
package main

func main() {
    println("Hello")
}
```

IMPORTS

```go
package main

import "fmt"

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

```go
package main

import "fmt"

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

VARIABLEN

```go
package main

import "fmt"

var name string

func main() {
    name = "Jax"
    fmt.Println("Hello, ", name)
}
```

```go
package main

import "fmt"

var name string

func main() {
    name = "Jax"
    fmt.Println("Hello, ", name)
}
```

```
package main

import "fmt"

func main() {
    name := "Jax"
    fmt.Println("Hello, ", name)
}
```

TYPEN

```go
package main

import "fmt"

type message string

var m message

func main() {
    m = "Hello, Jax"
    fmt.Println(m)
}
```

```go
package main

import "fmt"

type message string

var m message

func main() {
    m = "Hello, Jax"
    fmt.Println(m)
}
```

```
package main

import "fmt"

type message string

var m message

func main() {
    m = "Hello, Jax"
    fmt.Println(m)
}
```

ZUSAMMENGESETZTE TYPEN

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    fmt.Println(g.message, g.greetee)
}
```

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    fmt.Println(g.message, g.greetee)
}
```

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    fmt.Println(g.message, g.greetee)
}
```

METHODS

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func print(g greeting) {
    fmt.Println(g.message, g.greetee)
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    print(g)
}
```

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func print(g greeting) {
    fmt.Println(g.message, g.greetee)
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    print(g)
}
```

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func print(g greeting) {
    fmt.Println(g.message, g.greetee)
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    print(g)
}
```

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func (g greeting) print() {
    fmt.Println(g.message, g.greetee)
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    g.print()
}
```

```go
package main

import "fmt"

type greeting struct {
    message string
    greetee string
}

func (g greeting) print() {
    fmt.Println(g.message, g.greetee)
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    g.print()
}
```

```go
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```go
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```go
package main

import "fmt"

type greeting struct {
    message     string
    greetee     string
    timesPrinted int
}

func (g greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)  => 0
}
```

POINTER

```go
package main

import "fmt"

func inc(px *int) {
    *px++
}

func main() {
    x := 1
    p := &x
    inc(p)
    fmt.Printf("x: %d", *p)
}
```

```
package main

import "fmt"

func inc(px *int) {
    *px++
}

func main() {
    x := 1
    p := &x
    inc(p)
    fmt.Printf("x: %d", *p)
}
```

```go
package main

import "fmt"

func inc(px *int) {
    *px++
}

func main() {
    x := 1
    p := &x
    inc(p)
    fmt.Printf("x: %d", *p)
}
```

```
package main

import "fmt"

func inc(px *int) {
    *px++
}

func main() {
    x := 1
    p := &x
    inc(p)
    fmt.Printf("x: %d", *p)
}
```

```
package main

import "fmt"

func inc(px *int) {
    *px++
}

func main() {
    x := 1
    p := &x
    inc(p)
    fmt.Printf("x: %d", *p)
}
```

```
package main

import "fmt"

func inc(px *int) {
    *px++
}

func main() {
    x := 1
    p := &x
    inc(p)
    fmt.Printf("x: %d", *p)
}
```

```
package main

import "fmt"

func inc(px *int) {
    *px++
}

func main() {
    x := 1
    p := &x
    inc(p)
    fmt.Printf("x: %d", *p)  => x: 2
}
```

POINTER RECEIVER

```go
package main

import "fmt"

type greeting struct {
    message     string
    greetee     string
    timesPrinted int
}

func (g *greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g *greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```go
package main

import "fmt"

type greeting struct {
    message     string
    greetee     string
    timesPrinted int
}

func (g *greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```go
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g *greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```go
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g *greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted)
}
```

```go
package main

import "fmt"

type greeting struct {
    message      string
    greetee      string
    timesPrinted int
}

func (g *greeting) print() {
    fmt.Println(g.message, g.greetee)
    g.timesPrinted++
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    g.print()
    fmt.Printf("times printed: %d", g.timesPrinted) => 1
}
```

INTERFACES

```
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print());
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

```go
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print());
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

```
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print());
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

```go
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print())
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

```go
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print())
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

```go
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print())
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

```go
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print());
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

```go
type Printable interface {
    print() string
}

type greeting struct {
    message      string
    greetee      string
}

func (g *greeting) print() string {
    return fmt.Sprintf("", g.message, g.greetee)
}

func printOnConsole(printable Printable)  {
    fmt.Printf("=> %s", printable.print());
}

func main() {
    g := &greeting{message: "Hello", greetee: "Jax"}
    printOnConsole(g)
}
```

GO ONE PAGER

Packages

Typen

Interfaces

Funktionen

Pointer

```
package main
import "fmt"

type Dog struct {
    name        string
    timesBarked int
}

type Animal interface { bark() }

func (d *Dog) bark() {
    fmt.Printf("I am %s", d.name)
    d.timesBarked++
}

func sayHello(animal Animal) {
    animal.bark()
}

func main() {
    kalle := &Dog{name: "Kalle"}
    sayHello(kalle)
}
```

```go
package main
import "fmt"

type Dog struct {
    name        string
    timesBarked int
}

type Animal interface { bark() }

func (d *Dog) bark() {
    fmt.Printf("I am %s", d.name)
    d.timesBarked++
}

func sayHello(animal Animal) {
    animal.bark()
}

func main() {
    kalle := &Dog{name: "Kalle"}
    sayHello(kalle)
}
```

```go
package main
import "fmt"

type Dog struct {
    name        string
    timesBarked int
}

type Animal interface { bark() }

func (d *Dog) bark() {
    fmt.Printf("I am %s", d.name)
    d.timesBarked++
}

func sayHello(animal Animal) {
    animal.bark()
}

func main() {
    kalle := &Dog{name: "Kalle"}
    sayHello(kalle)
}
```

```go
package main
import "fmt"

type Dog struct {
    name        string
    timesBarked int
}

type Animal interface { bark() }

func (d *Dog) bark() {
    fmt.Printf("I am %s", d.name)
    d.timesBarked++
}

func sayHello(animal Animal) {
    animal.bark()
}

func main() {
    kalle := &Dog{name: "Kalle"}
    sayHello(kalle)
}
```

```go
package main
import "fmt"

type Dog struct {
    name        string
    timesBarked int
}

type Animal interface { bark() }

func (d *Dog) bark() {
    fmt.Printf("I am %s", d.name)
    d.timesBarked++
}

func sayHello(animal Animal) {
    animal.bark()
}

func main() {
    kalle := &Dog{name: "Kalle"}
    sayHello(kalle)
}
```

```go
package main
import "fmt"

type Dog struct {
    name        string
    timesBarked int
}

type Animal interface { bark() }

func (d *Dog) bark() {
    fmt.Printf("I am %s", d.name)
    d.timesBarked++
}

func sayHello(animal Animal) {
    animal.bark()
}

func main() {
    kalle := &Dog{name: "Kalle"}
    sayHello(kalle)
}
```

```go
package main
import "fmt"

type Dog struct {
    name        string
    timesBarked int
}

type Animal interface { bark() }

func (d *Dog) bark() {
    fmt.Printf("I am %s", d.name)
    d.timesBarked++
}

func sayHello(animal Animal) {
    animal.bark()
}

func main() {
    kalle := &Dog{name: "Kalle"}
    sayHello(kalle)
}
```

GO - SPRACHEIGENSCHAFTEN

KAPSELUNG

VERERBUNG

JAVA

```
public class A {

    public void print() {
        ...
    }
}

public class B extends A {
}

B b = new B();
b.print();
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

func (a A) print() {
    ...
}

b := B{}
b.print()  // Not allowed in Java
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

func (a A) print() {
    ...
}

b := B{}
b.print()  // Not allowed in Java
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

func (a A) print() {
    ...
}

b := B{}
b.print()  // Not allowed in Java
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

func (a A) print() {
    ...
}

b := B{}
b.print()  // Not allowed in Java
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

func (a A) print() {
    ...
}

b := B{}
b.print()  // Not allowed in Java
```

POLYMORPHISMUS

JAVA

```
public class A {

    public void print() {
        ...
    }
}

public class B extends A {
}

List<A> l = new ArrayList<>();
l.add(new B());
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

list := [2]A{}
list[0] = A{}
list[1] = B{} // doesn't compile
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

list := [2]A{}
list[0] = A{}
list[1] = B{} // doesn't compile
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

list := [2]A{}
list[0] = A{}
list[1] = B{} // doesn't compile
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

list := [2]A{}
list[0] = A{}
list[1] = B{} // doesn't compile
```

GO

```
type A struct {
    a int
}

type B struct {
    A
}

list := [2]A{}
list[0] = A{}
list[1] = B{} // doesn't compile
```

POLYMORPHISMUS VIA INTERFACES

```
type Animal interface {
    Talk() string
}

...

kalle := Dog{"Kalle"}
felix := Cat{"Felix"}

animals := [2]Animal{kalle, felix}
for _, v := range animals {
    fmt.Printf("Hey, %s\n", v.Talk())
}
```

```
type Animal interface {
    Talk() string
}

...

kalle := Dog{"Kalle"}
felix := Cat{"Felix"}

animals := [2]Animal{kalle, felix}
for _, v := range animals {
    fmt.Printf("Hey, %s\n", v.Talk())
}
```

```
type Animal interface {
    Talk() string
}

...

kalle := Dog{"Kalle"}
felix := Cat{"Felix"}

animals := [2]Animal{kalle, felix}
for _, v := range animals {
    fmt.Printf("Hey, %s\n", v.Talk())
}
```

```
type Animal interface {
    Talk() string
}

...

kalle := Dog{"Kalle"}
felix := Cat{"Felix"}

animals := [2]Animal{kalle, felix}
for _, v := range animals {
    fmt.Printf("Hey, %s\n", v.Talk())
}
```

```
type Animal interface {
    Talk() string
}

...

kalle := Dog{"Kalle"}
felix := Cat{"Felix"}

animals := [2]Animal{kalle, felix}
for _, v := range animals {
    fmt.Printf("Hey, %s\n", v.Talk())
}
```

```
type Animal interface {
    Talk() string
}

...

kalle := Dog{"Kalle"}
felix := Cat{"Felix"}

animals := [2]Animal{kalle, felix}
for _, v := range animals {
    fmt.Printf("Hey, %s\n", v.Talk())
}
```

FUNKTIONALE PROGRAMMIERUNG

```
func main() {
    f := func(x int) int {
        return x * x
    }
    result := f(2)
    print(result)
}
```

```
func main() {
    f := func(x int) int {
        return x * x
    }
    result := f(2)
    print(result)
}
```

```go
func foo(fn func(x int) int) int {
    return fn(2)
}

func main() {
    f := func(x int) int {
        return x * x
    }
    result := foo(f)
    print(result)
}
```

```
func foo(fn func(x int) int) int {
    return fn(2)
}

func main() {
    f := func(x int) int {
        return x * x
    }
    result := foo(f)
    print(result)
}
```

```
type myfunction func(x int) int

func foo(fn myfunction) int {
    return fn(2)
}

func main() {
    f := func(x int) int {
        return x * x
    }
    result := foo(f)
    print(result)
}
```

```
type myfunction func(x int) int

func foo(fn myfunction) int {
    return fn(2)
}

func main() {
    f := func(x int) int {
        return x * x
    }
    result := foo(f)
    print(result)
}
```

CONCURRENCY

```
func sum(s []int) int {
    sum := 0
    for _, v := range s {
        sum += v
    }
    return sum
}

println("sum: ", sum(values)) // sum:  124999999750000000
```

```
func sum(s []int) int {
    sum := 0
    for _, v := range s {
        sum += v
    }
    return sum
}

println("sum: ", sum(values)) // sum:  12499999750000000
```

```
func sum(s []int) int {
	sum := 0
	for _, v := range s {
		sum += v
	}
	return sum
}

s1 := sum(values[0:size/2-1])
s2 := sum(values[size/2-1:])
println("sum: ", s1 + s2)    // sum: 124999999750000000
```

```
func sum(s []int) int {
    sum := 0
    for _, v := range s {
        sum += v
    }
    return sum
}

s1 := sum(values[0:size/2-1])
s2 := sum(values[size/2-1:])
println("sum: ", s1 + s2)    // sum: 1249999997500000000
```

```
func sum(s []int) int {
    sum := 0
    for _, v := range s {
        sum += v
    }
    return sum
}

s1 := sum(values[0:size/2-1])
s2 := sum(values[size/2-1:])
println("sum: ", s1 + s2)    // sum: 1249999997500000000
```

```go
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum                     // send result to channel
}

c := make(chan int)
go sum(values[0:size/2-1], c)
go sum(values[size/2-1:], c)
s1, s2 := <-c, <-c               // receive value from channel
println("sum: ", s1 + s2)        // sum: 124999999750000000
```

```
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum                        // send result to channel
}

c := make(chan int)
go sum(values[0:size/2-1], c)
go sum(values[size/2-1:], c)
s1, s2 := <-c, <-c                  // receive value from channel
println("sum: ", s1 + s2)           // sum: 12499999750000000
```

```
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum                        // send result to channel
}

c := make(chan int)
go sum(values[0:size/2-1], c)
go sum(values[size/2-1:], c)
s1, s2 := <-c, <-c                  // receive value from channel
println("sum: ", s1 + s2)           // sum: 12499999750000000
```

```
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum                  // send result to channel
}

c := make(chan int)
go sum(values[0:size/2-1], c)
go sum(values[size/2-1:], c)
s1, s2 := <-c, <-c            // receive value from channel
println("sum: ", s1 + s2)     // sum: 12499999975000000
```

```
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum                        // send result to channel
}

c := make(chan int)
go sum(values[0:size/2-1], c)
go sum(values[size/2-1:], c)
s1, s2 := <-c, <-c                  // receive value from channel
println("sum: ", s1 + s2)       // sum: 12499999750000000
```

ZUSAMMENFASSUNG

Komposition statt Vererbung

Interfaces

Polymorphismus

Funktionale Programmierung

Concurrency

GO TOOL CHAIN

DER GO WORKSPACE

```
$GOPATH/src
       /bin
       /pkg
```

EIN WORKSPACE, VIELE PROJEKTE

```
$GOPATH/src/bitbucket.org/rwirdemann/hello/main.go
                                    /hello/domain/models.go
                                    /picue/main.go
                                    /picue/controllers/groupli
       /bin/hello
            picue

       /pkg/linux_amd64/bitbucket.org/rwirdemann/hello/domain.
```

EIN TOOL: GO

```
go run src/bitbucket.org/rwirdemann/hello/main.go
```

```
go install bitbucket.org/rwirdemann/hello
```

```
cd $GOPATH/src/bitbucket.org/rwirdemann/hello/domain
go test
```

```
go test groups_test.go
```

DEPENDENCY MANAGEMENT

```
$GOPATH/src/github.com/
           /golang.org
           ...
```

```
$ go get github.com/go-sql-driver/mysql
```

```
$GOPATH/src/github.com/go-sql-driver/mysql
```

```
import "github.com/go-sql-driver/mysql"
```

WEITERE TOOLS

gofmt

goimport

godef

golint

gocode

...

WER NUTZT GO?

Google

Docker

Wunderkinder

Booking.com

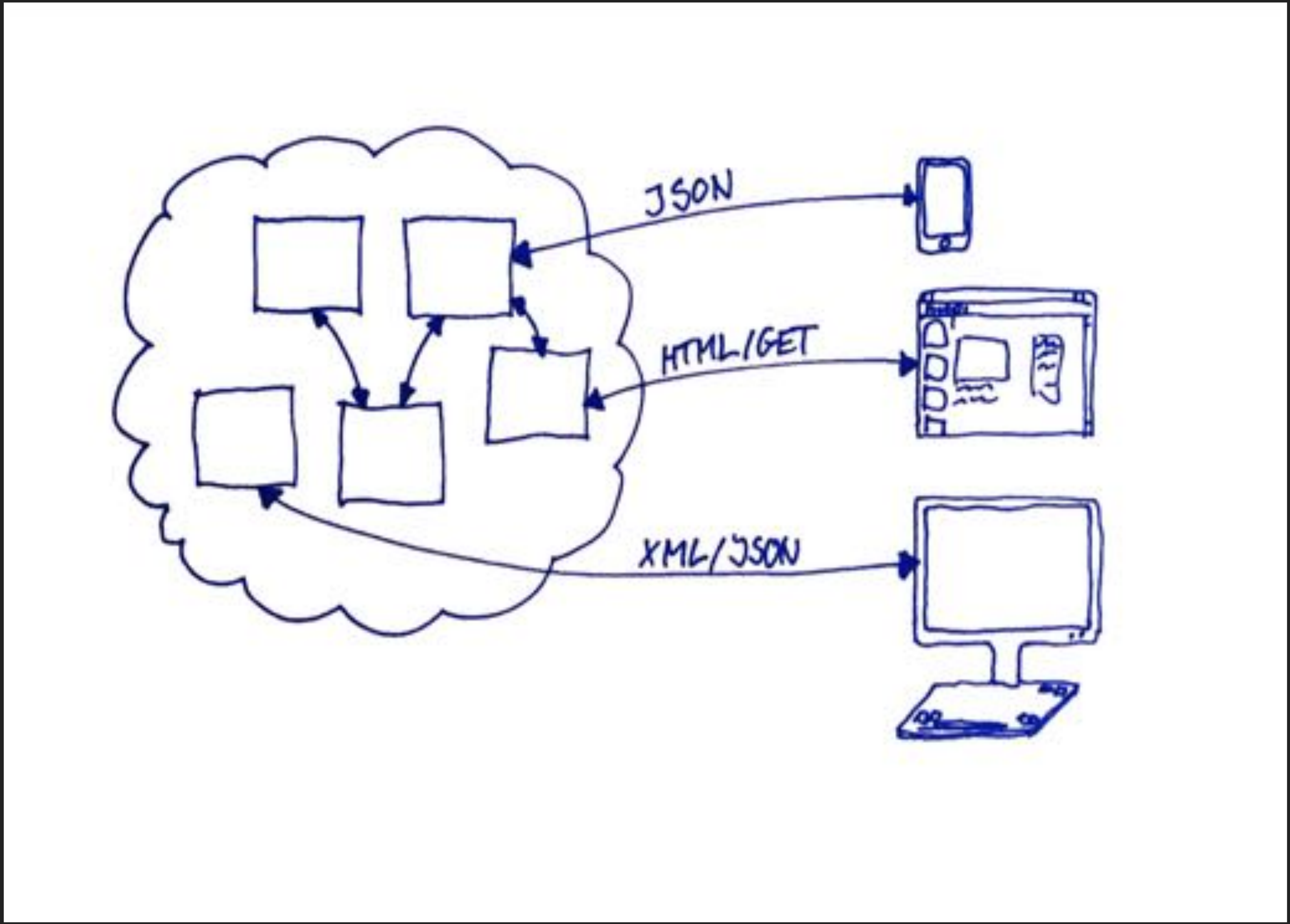
Crashlytics

Dropbox

Soundcloud

WOFÜR GO?

APIS UND MICROSERVICES



WARUM GO?

PRODUKTIVITÄT

Erlernbarkeit

Standards

Einheitlicher Workspace

Lesbarkeit

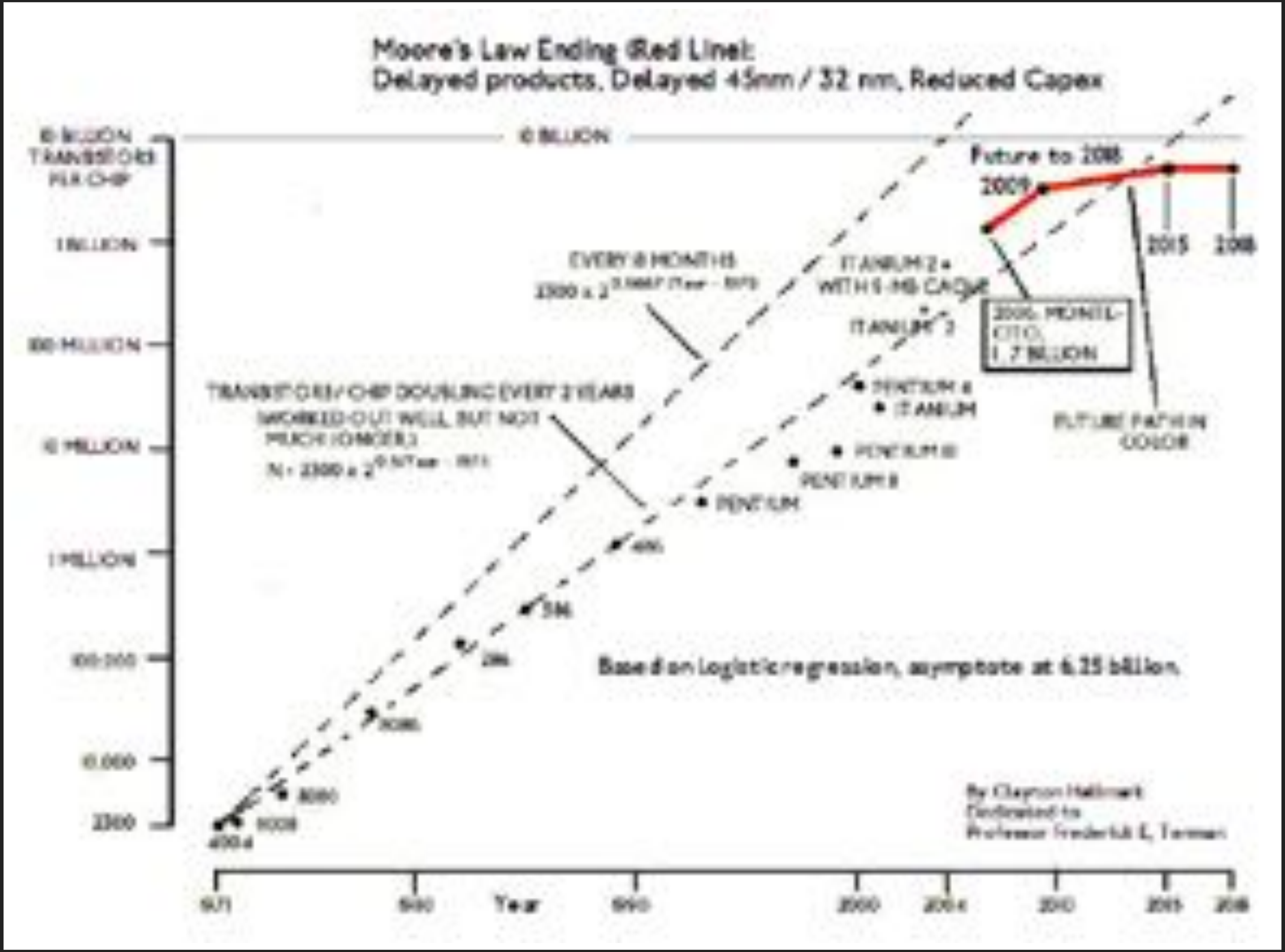
TYPSICHERHEIT

```
Map<String, List<Note>> sections = new HashMap<String, List<No
```

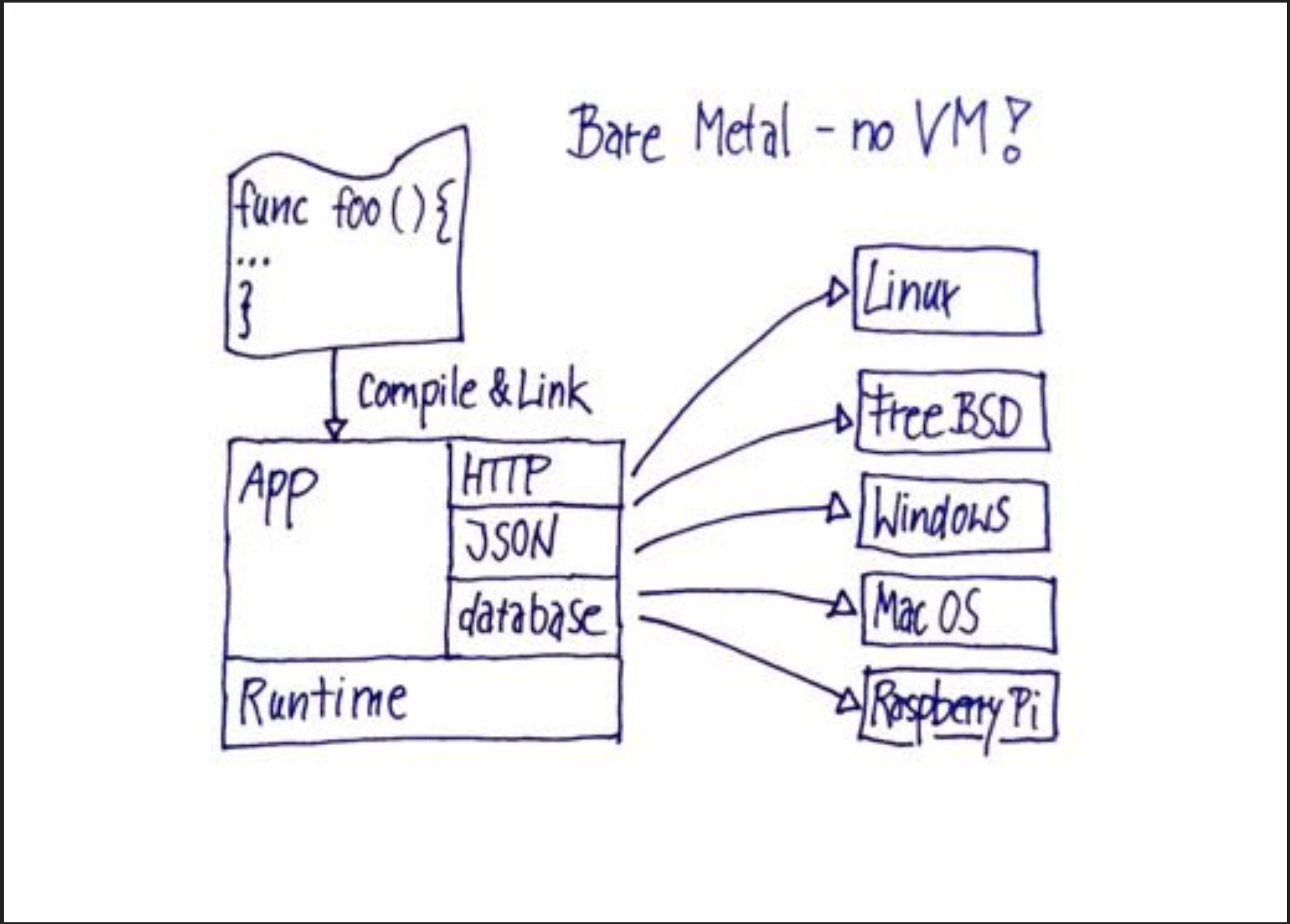
```
Map<String, List<Note>> sections = new HashMap<>();
```

```
sections := make(map[string][]Note)
```

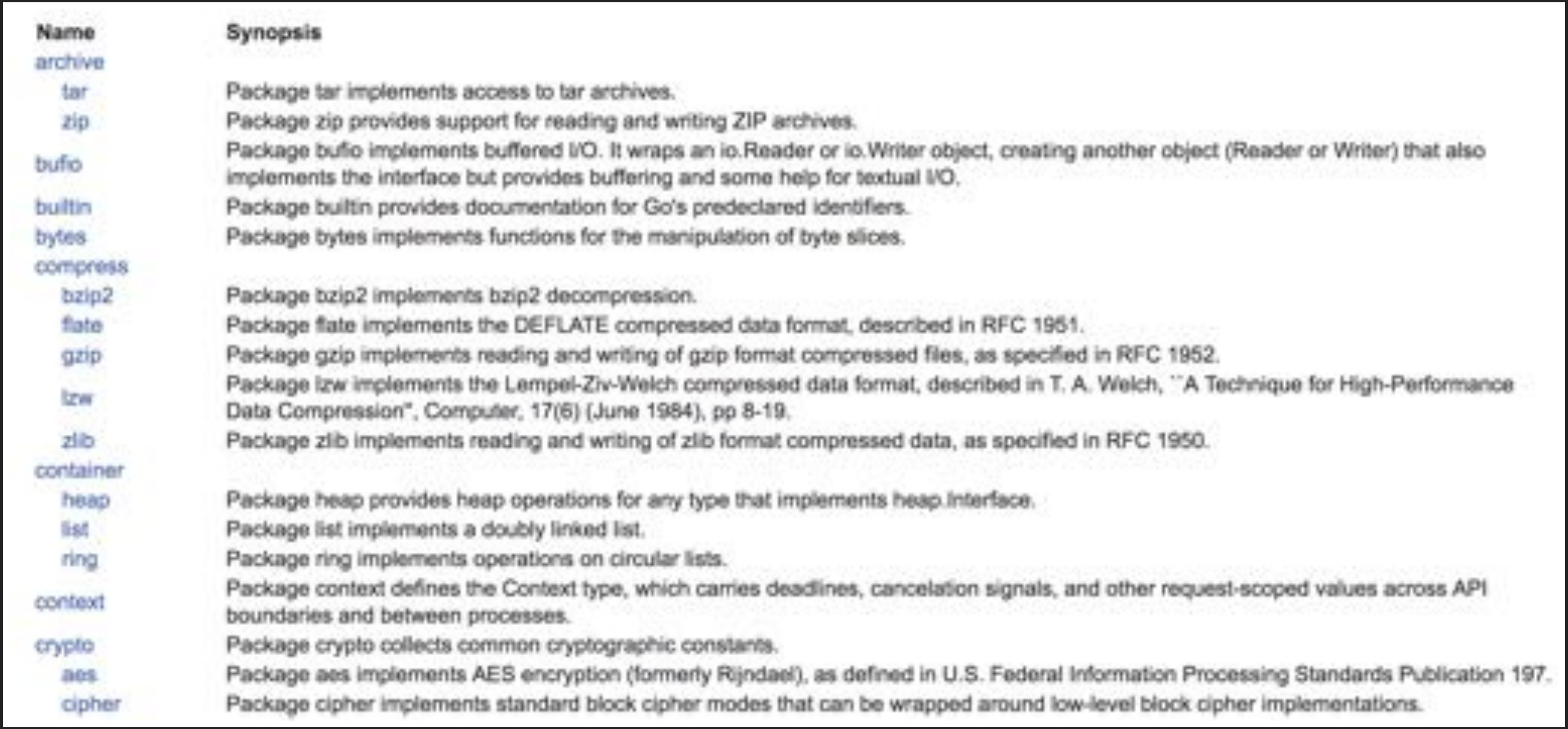
PERFORMANCE



DEPLOYMENT


Bare Metal - no VM!
func foo(){
...
}
Compile & Link
App
HTTP
JSON
database
Runtime
Linux
FreeBSD
Windows
Mac OS
Raspberry Pi

WORLD CLASS STANDARD LIBRARY



Name	Synopsis
archive	
tar	Package tar implements access to tar archives.
zip	Package zip provides support for reading and writing ZIP archives.
bufio	Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.
builtin	Package builtin provides documentation for Go's predeclared identifiers.
bytes	Package bytes implements functions for the manipulation of byte slices.
compress	
bzip2	Package bzip2 implements bzip2 decompression.
flate	Package flate implements the DEFLATE compressed data format, described in RFC 1951.
gzip	Package gzip implements reading and writing of gzip format compressed files, as specified in RFC 1952.
lzw	Package lzw implements the Lempel-Ziv-Welch compressed data format, described in T. A. Welch, ``A Technique for High-Performance Data Compression'', Computer, 17(6) (June 1984), pp 8-19.
zlib	Package zlib implements reading and writing of zlib format compressed data, as specified in RFC 1950.
container	
heap	Package heap provides heap operations for any type that implements heap.Interface.
list	Package list implements a doubly linked list.
ring	Package ring implements operations on circular lists.
context	Package context defines the Context type, which carries deadlines, cancelation signals, and other request-scoped values across API boundaries and between processes.
crypto	Package crypto collects common cryptographic constants.
aes	Package aes implements AES encryption (formerly Rijndael), as defined in U.S. Federal Information Processing Standards Publication 197.
cipher	Package cipher implements standard block cipher modes that can be wrapped around low-level block cipher implementations.

NET/HTTP

```
package main

import (
    "fmt"
    "net/http"
)

func handleRequest(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World\n")
}

func main() {
    http.HandleFunc("/hello", handleRequest)
    http.ListenAndServe(":8080", nil)
}
```

```go
package main

import (
    "fmt"
    "net/http"
)

func handleRequest(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World\n")
}

func main() {
    http.HandleFunc("/hello", handleRequest)
    http.ListenAndServe(":8080", nil)
}
```

```go
package main

import (
	"fmt"
	"net/http"
)

func handleRequest(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintf(w, "Hello, World\n")
}

func main() {
	http.HandleFunc("/hello", handleRequest)
	http.ListenAndServe(":8080", nil)
}
```

```go
package main

import (
    "fmt"
    "net/http"
)

func handleRequest(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World\n")
}

func main() {
    http.HandleFunc("/hello", handleRequest)
    http.ListenAndServe(":8080", nil)
}
```

```go
package main

import (
    "fmt"
    "net/http"
)

func handleRequest(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World\n")
}

func main() {
    http.HandleFunc("/hello", handleRequest)
    http.ListenAndServe(":8080", nil)
}
```

EINFACHHEIT

Reduzierte Syntax

Sprachstabilität als Feature

Nur ein Weg

Mindset

WAS NERVT?

ERROR HANDLING

```go
func create(a *Activity) {

    var res Result
    var err Error

    res, err = DB.Exec("insert ...", a.Name)
    if err != nil {
        panic(err)
    }

    var id int64

    id, err = res.LastInsertId()
    if err != nil {
        panic(err)
    }

    activity.Id = id

    ...
}
```

NO RAILS

HTTP / JSON Support ist top

gobuffalo

POINTER

Waren wir froh, als wir sie los waren

NullPointerException

Immutability for free

Entscheide dich: by value oder by reference

Keine Pointer-Arithmetik

Garbage Collection

KEINE GENERICS

```
List<String> l = new ArrayList<>();
Map<String, Media> m = new HashMap<>();


l := [5]string
m := make(map[string]Media)
```

DEPENDENCY MANAGEMENT

```
$GOPATH/src/bitbucket.org/rwirdemann/gornale
                                     picue

            github.com/go-sql-driver/mysql
```

KEINE KLASSEN

Keine Vererbung

Fehlende Modellierungsgrundlage

WIE GEHTS WEITER?

Mein nächstes Go-Projekt

Workshop

Schulung

Vortrag

Cooles Freizeitprojekt

RESOURCEN

https://tour.golang.org

https://gobyexample.com/

The Go Programming Language

HEY HO, LET'S GO

VIELEN DANK!

ralf.wirdemann@kommitment.biz