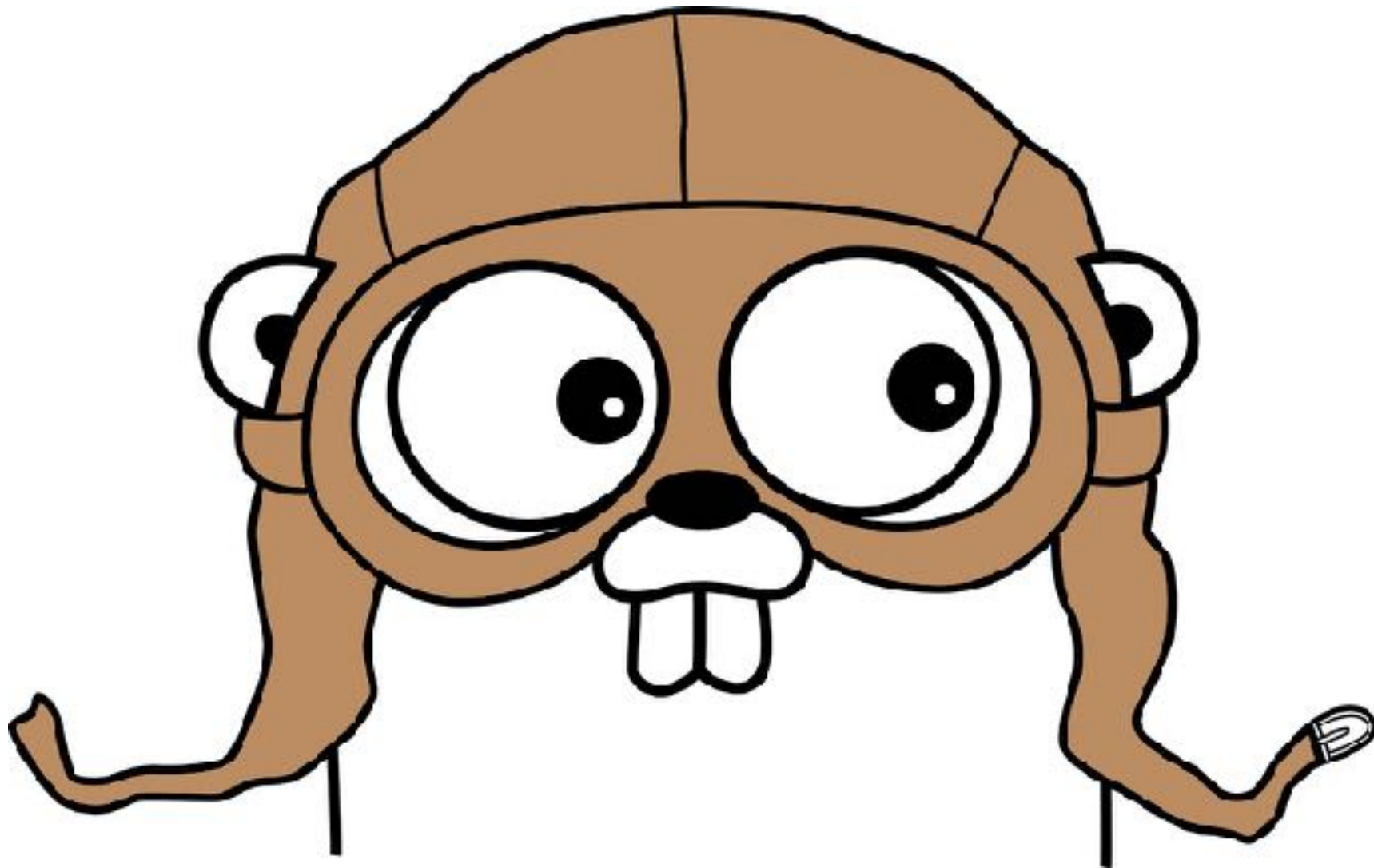


GET GO-ING WITH A NEW LANGUAGE

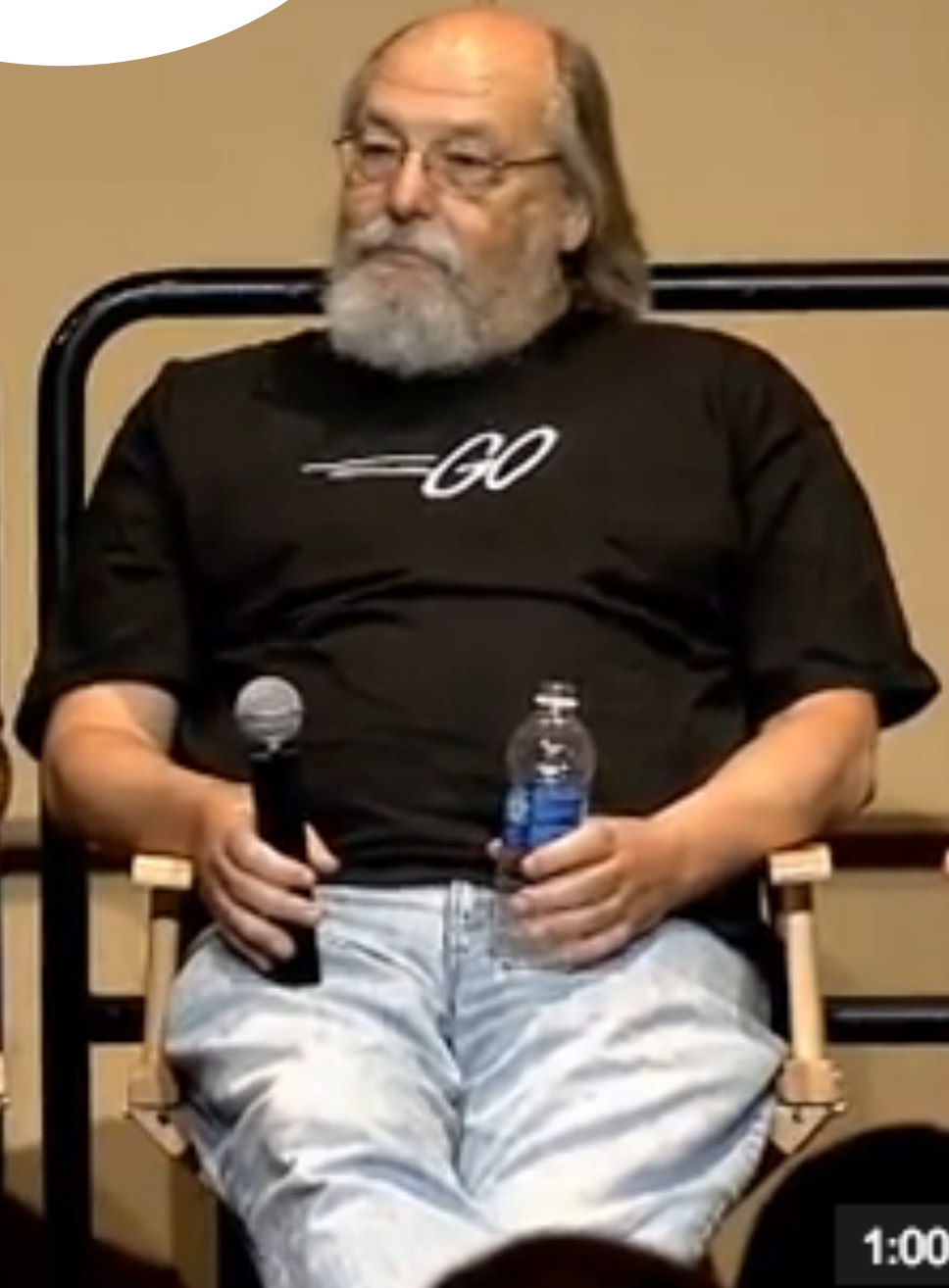
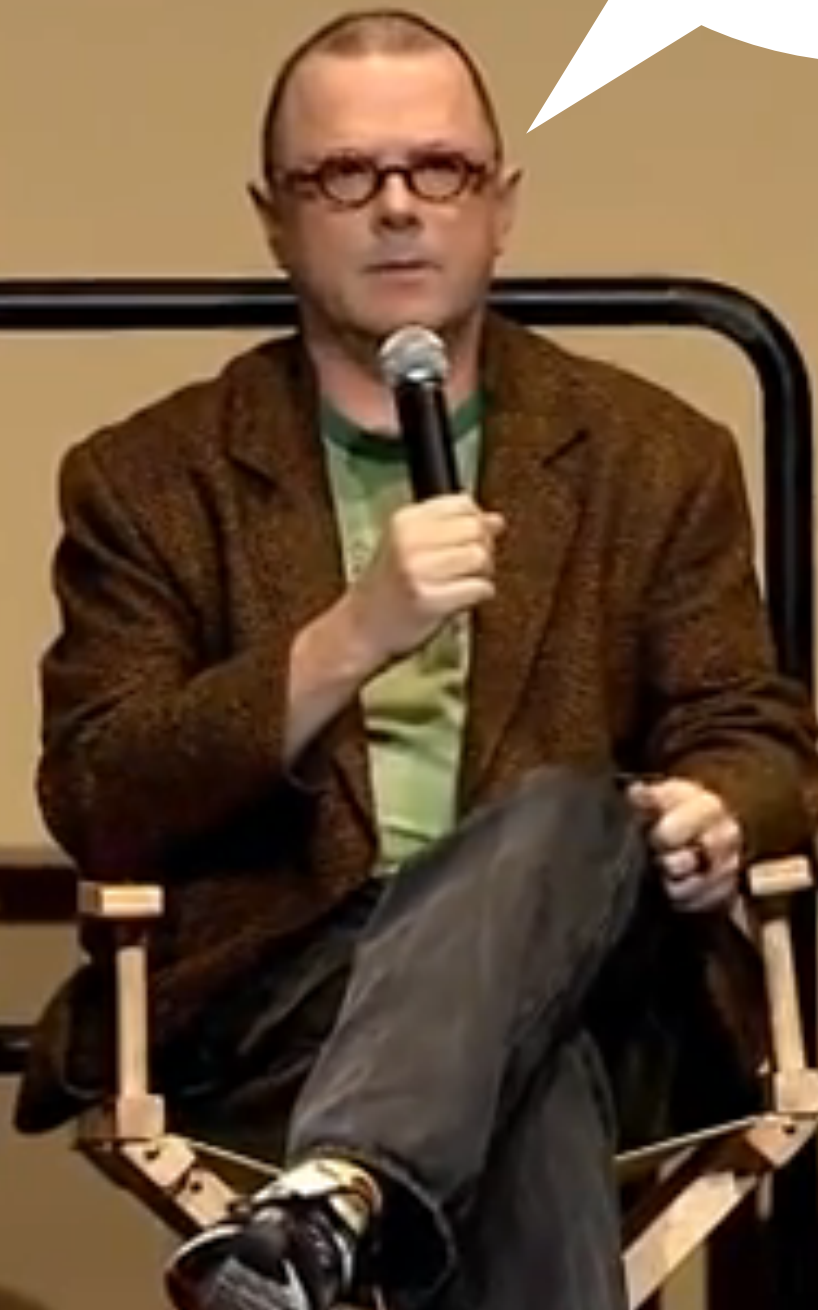
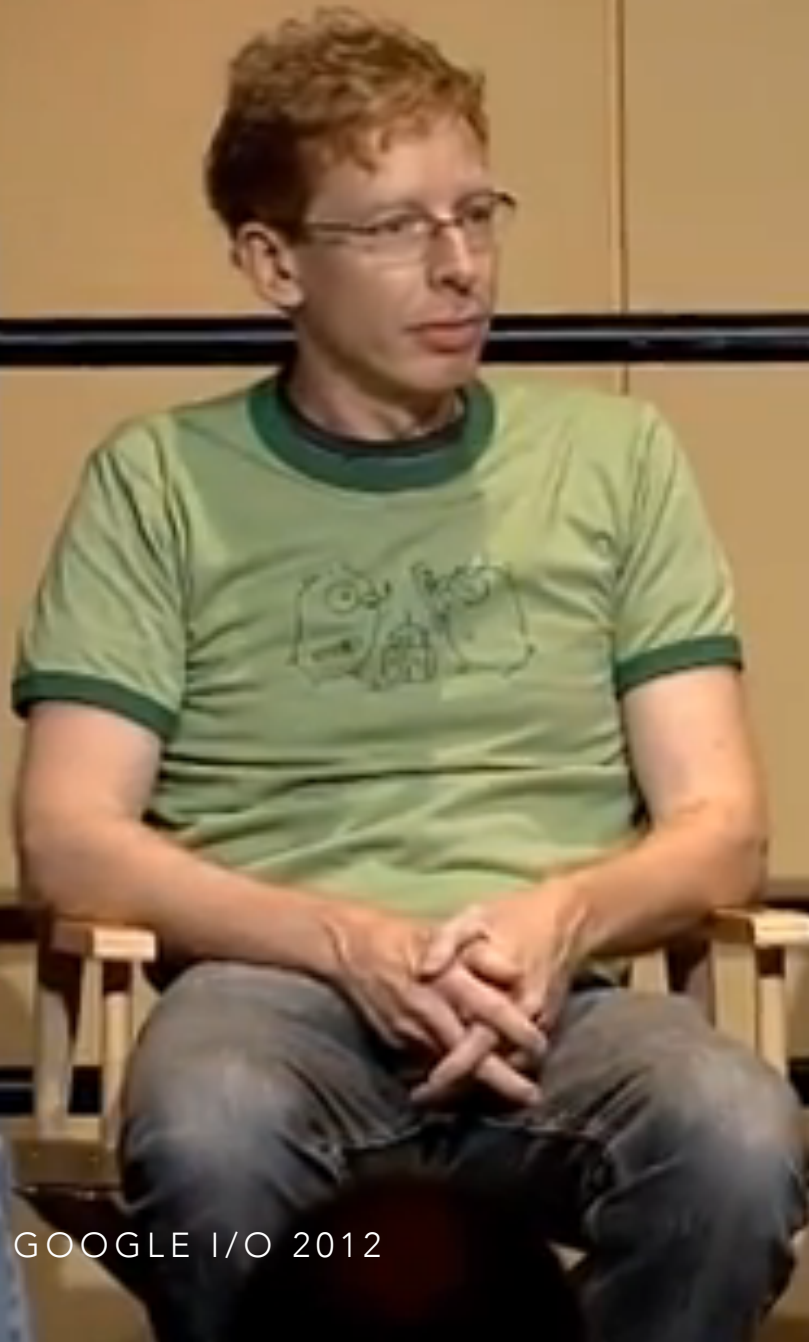


KAT ZIEŃ (@KASIAZIEN)
PHPHANTS 11 APRIL 2018



MADE BY IAN BAKER

GO(LANG)



DIFFERENCES



VS

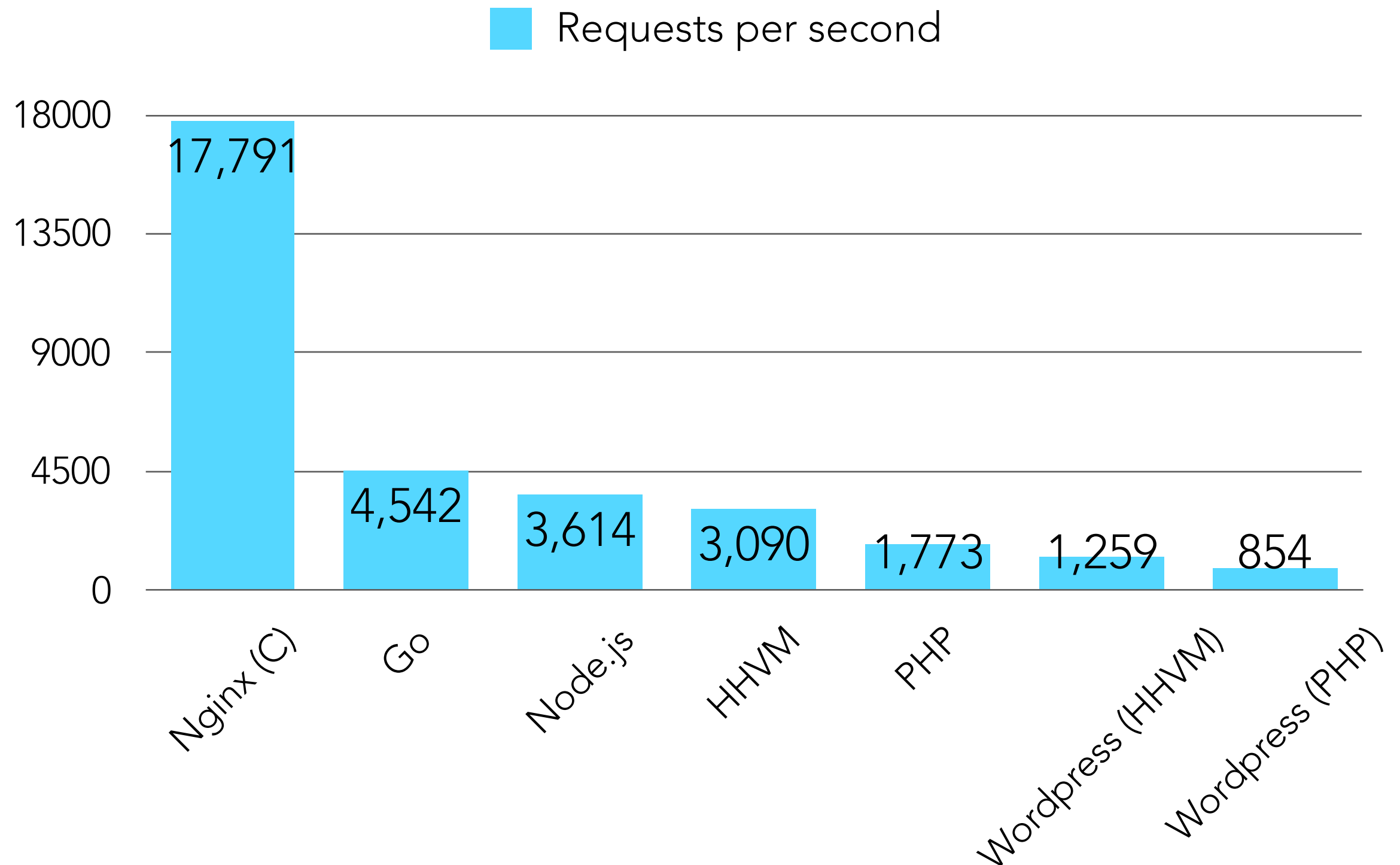


INTERPRETED VS COMPILED



IMAGE: [COMPILERS - PRINCIPLES, TECHNIQUES AND TOOLS](#)

BENCHMARKS



```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    var greeting string = "Hello! Is it %s you're looking for?\n"

    words := []string{"me", "tea"}

    rand.Seed(42)

    for i := 0; i < 5; i++ {
        fmt.Printf(greeting, words[rand.Intn(len(words))])
    }
}
```

```
package main
```

```
import (  
    "fmt"  
    "math/rand"  
)
```

```
func main() {  
    var greeting string = "Hello! Is it %s you're looking for?\n"  
  
    words := [2]string{"me", "tea"}  
  
    rand.Seed(42)  
  
    for i := 0; i < 5; i++ {  
        fmt.Printf(greeting, words[rand.Intn(len(words))])  
    }  
}
```



```
package main
```

```
import (  
    "fmt"  
    "math/rand"  
)
```

```
func main() {  
    var greeting string = "Hello! Is it %s you're looking for?\n"  
  
    words := [2]string{"me", "tea"}  
  
    rand.Seed(42)  
  
    for i := 0; i < 5; i++ {  
        fmt.Printf(greeting, words[rand.Intn(len(words))])  
    }  
}
```

```
package main
```

```
import (  
    "fmt"  
    "math/rand"  
)
```

```
func main() {
```

```
    var greeting string = "Hello! Is it %s you're looking for?\n"
```

```
    words := [2]string{"me", "tea"}
```

```
    rand.Seed(42)
```

```
    for i := 0; i < 5; i++ {
```

```
        fmt.Printf(greeting, words[rand.Intn(len(words))])
```

```
    }
```

```
}
```

```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    var greeting string = "Hello! Is it %s you're looking for?\n"

    words := []string{"me", "tea"}

    rand.Seed(42)

    for i := 0; i < 5; i++ {
        fmt.Printf(greeting, words[rand.Intn(len(words))])
    }
}
```



```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    var greeting string = "Hello! Is it %s you're looking for?\n"

    words := [2]string{"me", "tea"}

    rand.Seed(42)

    for i := 0; i < 5; i++ {
        fmt.Printf(greeting, words[rand.Intn(len(words))])
    }
}
```

```
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    var greeting string = "Hello! Is it %s you're looking for?\n"

    words := []string{"me", "tea"}

    rand.Seed(42)

    for i := 0; i < 5; i++ {
        fmt.Printf(greeting, words[rand.Intn(len(words))])
    }
}
```

```
$ go run lionel.go
```

```
Hello! Is it tea you're looking for?
```

```
Hello! Is it tea you're looking for?
```

```
Hello! Is it me you're looking for?
```

```
Hello! Is it me you're looking for?
```

```
Hello! Is it tea you're looking for?
```



```
$ go build lionel.go
```

```
$ ls
```

```
lionel      lionel.go
```

```
$ ./lionel
```

```
Hello! Is it tea you're looking for?
```

```
Hello! Is it tea you're looking for?
```

```
Hello! Is it me you're looking for?
```

```
Hello! Is it me you're looking for?
```

```
Hello! Is it tea you're looking for?
```

TOOLS

OUT OF THE BOX

- ▶ formatting
- ▶ linting
- ▶ testing
- ▶ documenting
- ▶ running
- ▶ profiling



IMAGE: [LEGO.COM](https://www.lego.com)

TOOLS

OUT OF THE BOX



GOPATH

`$GOPATH/src/<vendor>/<project-name>`

e.g.

`/Users/Kat/go-code/src/gitlab.com/katzien/hello`

`/Users/Kat/go-code/src/github.com/golang/go`

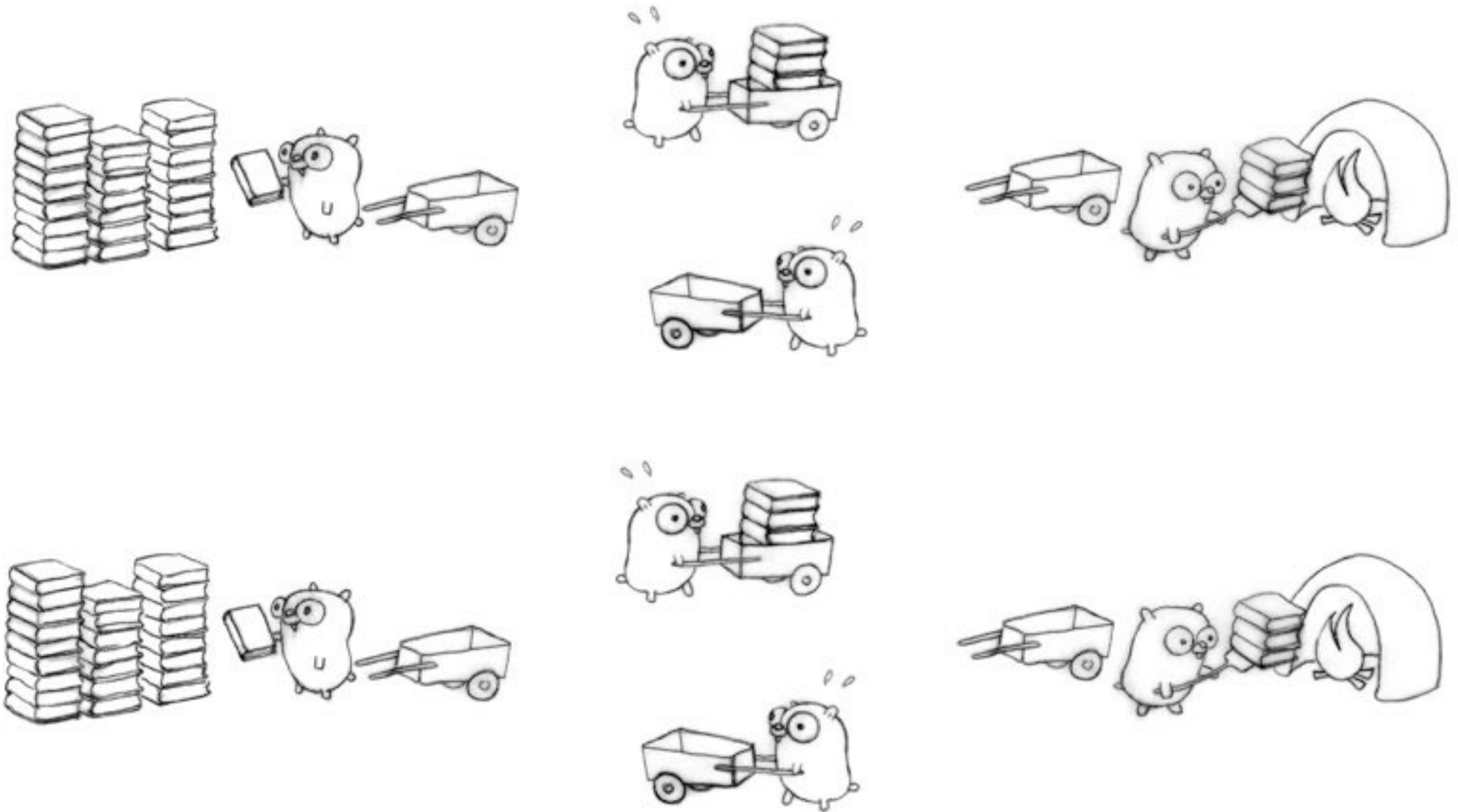


CONCURRENCY VS PARALLELISM

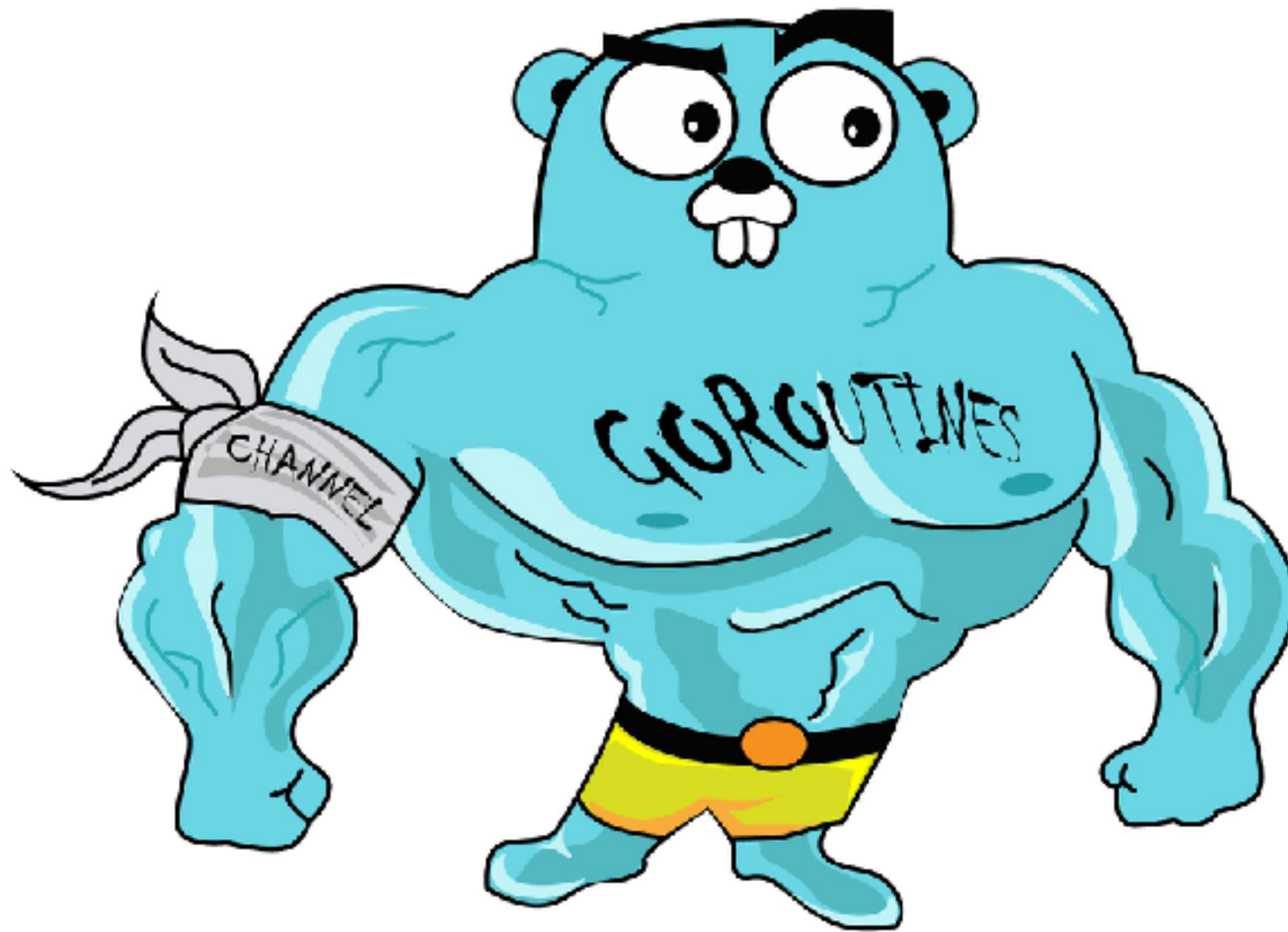
Concurrency: two threads are making progress

Parallelism: two threads are executing simultaneously

CONCURRENCY != PARALLELISM



BUILT-IN CONCURRENCY



GOROUTINES

```
go myFunction()
```

```
func say(word string) {  
    for i := 0; i < 5; i++ {  
        time.Sleep(100 * time.Millisecond)  
        fmt.Println(word)  
    }  
}
```

```
func main() {  
    go say("world")  
    say("hello")  
}
```

```
func say(word string) {  
    for i := 0; i < 5; i++ {  
        time.Sleep(100 * time.Millisecond)  
        fmt.Println(word)  
    }  
}
```

```
func main() {  
    go say("world")  
    say("hello")  
}
```



```
func say(word string) {  
    for i := 0; i < 5; i++ {  
        time.Sleep(100 * time.Millisecond)  
        fmt.Println(word)  
    }  
}
```

```
func main() {  
    go say("world")  
    say("hello")  
}
```

```
func say(word string) {  
    for i := 0; i < 5; i++ {  
        time.Sleep(100 * time.Millisecond)  
        fmt.Println(word)  
    }  
}
```

```
func main() {  
    go say("world")  
    say("hello")  
}
```

```
$ go run goroutines.go
```

world

hello

world

hello

hello

world

world

hello

world

hello

```
$ go run goroutines.go
```

world	world	hello	world
hello	hello	world	hello
world	hello	hello	world
hello	world	world	hello
hello	hello	world	world
world	world	hello	hello
world	hello	hello	hello
hello	world	world	world
world	world	hello	world
hello	hello	world	hello

```
$ go run goroutines.go
```

world	world	hello	world
hello	hello	world	hello
world	hello	hello	world
hello	world	world	hello
hello	hello	world	world
world	world	hello	hello
world	hello	hello	hello
hello	world	world	world
world	world	hello	world
hello	hello	world	hello

CHANNELS

```
messenger := make(chan string)
```

```
// Send value to channel  
messenger <- "Ohai!"
```

```
// Receive from channel and assign to variable  
message := <-messenger
```

```
func playSound(name string, freq time.Duration) <-chan string {  
    c := make(chan string)  
    go func() {  
        for i := 1; i <= 3; i++ {  
            c <- fmt.Sprintf("%s %d", name, i)  
            time.Sleep(time.Second * freq)  
        }  
    }()  
    return c  
}
```

```
func main() {  
    t1 := playSound("trumpet", 3)  
    t2 := playSound("guitar", 2)  
    t3 := playSound("drum", 1)  
  
    for i := 1; i <= 3; i++ {  
        println(<-t1)  
        println(<-t2)  
        println(<-t3)  
    }  
}
```

```
func playSound(name string, freq time.Duration) <-chan string
{
    c := make(chan string)
    go func() {
        for i := 1; i <= 3; i++ {
            c <- fmt.Sprintf("%s %d", name, i)
            time.Sleep(time.Second * freq)
        }
    }()
    return c
}

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

```
func playSound(name string, freq time.Duration) <-chan string
{
    c := make(chan string)
    go func() {
        for i := 1; i <= 3; i++ {
            c <- fmt.Sprintf("%s %d", name, i)
            time.Sleep(time.Second * freq)
        }
    }()
    return c
}

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

```
func playSound(name string, freq time.Duration) <-chan string
{
    c := make(chan string)
    go func() {
        for i := 1; i <= 3; i++ {
            c <- fmt.Sprintf("%s %d", name, i)
            time.Sleep(time.Second * freq)
        }
    }()
    return c
}

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



```
func playSound(name string, freq time.Duration) <-chan string
{
    ...
}

func main() {
    t1 := playSound("trumpet", 3)
    t2 := playSound("guitar", 2)
    t3 := playSound("drum", 1)

    for i := 1; i <= 3; i++ {
        println(<-t1)
        println(<-t2)
        println(<-t3)
    }
}
```

```
func playSound(name string, freq time.Duration) <-chan string
{
    ...
}
```

```
func main() {
    t1 := playSound("trumpet", 3)
    t2 := playSound("guitar", 2)
    t3 := playSound("drum", 1)
```

```
    for i := 1; i <= 3; i++ {
        println(<-t1)
        println(<-t2)
        println(<-t3)
    }
```

```
}
```

```
$ go run concurrency.go
```

```
trumpet 1
```

```
guitar 1
```

```
drum 1
```

```
trumpet 2
```

```
guitar 2
```

```
drum 2
```

```
trumpet 3
```

```
guitar 3
```

```
drum 3
```

```
func playSound(name string, freq time.Duration) <-chan string
{
    ...
}

func main() {
    t1 := playSound("trumpet", 3)
    t2 := playSound("guitar", 2)
    t3 := playSound("drum", 1)

    for i := 1; i <= 3; i++ {
        println(<-t1)
        println(<-t2)
        println(<-t3)
    }
}
```

```
func playSound(name string, freq time.Duration) <-chan string {  
    ...  
}
```

```
func main() {  
    t1 := playSound("trumpet", 3)  
    t2 := playSound("guitar", 2)  
    t3 := playSound("drum", 1)
```

```
    for i := 1; i <= 9; i++ {  
        select {  
            case msg := <-t1:  
                println(msg)  
            case msg := <-t2:  
                println(msg)  
            case msg := <-t3:  
                println(msg)  
        }  
    }
```

```
}
```



```
$ go run concurrency.go
```

```
drum 1  
guitar 1  
trumpet 1
```

```
drum 2  
drum 3  
guitar 2  
trumpet 2  
guitar 3  
trumpet 3
```

```
guitar 1  
trumpet 1  
drum 1
```

```
drum 2  
drum 3  
guitar 2  
trumpet 2  
guitar 3  
trumpet 3
```

```
$ go run concurrency.go
```

```
drum 1  
guitar 1  
trumpet 1
```

```
drum 2  
drum 3  
guitar 2  
trumpet 2  
guitar 3  
trumpet 3
```

```
guitar 1  
trumpet 1  
drum 1
```

```
drum 2  
drum 3  
guitar 2  
trumpet 2  
guitar 3  
trumpet 3
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    for j := range jobs {  
        log.Println("worker", id, "started job", j)  
        time.Sleep(time.Second)  
        log.Println("worker", id, "finished job", j)  
        results <- j * 2  
    }  
}
```

```
func main() {  
    jobs := make(chan int, 100)  
    results := make(chan int, 100)  
  
    for w := 1; w <= 3; w++ {  
        go worker(w, jobs, results)  
    }  
  
    for j := 1; j <= 5; j++ {  
        jobs <- j  
    }  
  
    close(jobs)  
  
    for r := 1; r <= 5; r++ {  
        fmt.Println("Result #", r, ": ", <-results)  
    }  
}
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    for j := range jobs {  
        log.Println("worker", id, "started job", j)  
        time.Sleep(time.Second)  
        log.Println("worker", id, "finished job", j)  
        results <- j * 2  
    }  
}  
  
func main() {  
    ...  
}
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    for j := range jobs {  
        log.Println("worker", id, "started job", j)  
        time.Sleep(time.Second)  
        log.Println("worker", id, "finished job", j)  
        results <- j * 2  
    }  
}  
  
func main() {  
    ...  
}
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    for i := range jobs {  
        log.Println("worker", id, "started job", j)  
        time.Sleep(time.Second)  
        log.Println("worker", id, "finished job", j)  
        results <- j * 2  
    }  
}  
  
func main() {  
    ...  
}
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    ...  
}  
  
func main() {  
    jobs := make(chan int, 100)  
    results := make(chan int, 100)  
  
    for w := 1; w <= 3; w++ {  
        go worker(w, jobs, results)  
    }  
  
    for j := 1; j <= 5; j++ {  
        jobs <- j  
    }  
  
    close(jobs)  
  
    for r := 1; r <= 5; r++ {  
        fmt.Println("Result #", r, ": ", <-results)  
    }  
}
```



```
func worker(id int, jobs <-chan int, results chan<- int) {  
    ...  
}
```

```
func main() {  
    jobs := make(chan int, 100)  
    results := make(chan int, 100)
```

```
    for w := 1; w <= 3; w++ {  
        go worker(w, jobs, results)  
    }
```

```
    for j := 1; j <= 5; j++ {  
        jobs <- j  
    }
```

```
    close(jobs)
```

```
    for r := 1; r <= 5; r++ {  
        fmt.Println("Result #", r, ": ", <-results)  
    }
```

```
}
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    ...  
}
```

```
func main() {  
    jobs := make(chan int, 100)  
    results := make(chan int, 100)  
  
    for w := 1; w <= 3; w++ {  
        go worker(w, jobs, results)  
    }
```

```
    for j := 1; j <= 5; j++ {  
        jobs <- j  
    }
```

```
    close(jobs)
```

```
    for r := 1; r <= 5; r++ {  
        fmt.Println("Result #", r, ": ", <-results)  
    }  
}
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    for j := range jobs {  
        log.Println("worker", id, "started job", j)  
        time.Sleep(time.Second)  
        log.Println("worker", id, "finished job", j)  
        results <- j * 2  
    }  
}
```

```
func main() {  
    ...  
    close(jobs)  
    for r := 1; r <= 5; r++ {  
        fmt.Println("Result #", r, ": ", <-results)  
    }  
}
```

```
func worker(id int, jobs <-chan int, results chan<- int) {  
    ...  
}
```

```
func main() {  
    jobs := make(chan int, 100)  
    results := make(chan int, 100)  
  
    for w := 1; w <= 3; w++ {  
        go worker(w, jobs, results)  
    }
```

```
    for j := 1; j <= 5; j++ {  
        jobs <- j  
    }
```

```
    close(jobs)
```

```
    for r := 1; r <= 5; r++ {  
        fmt.Println("Result #", r, ": ", <-results)  
    }
```

```
}
```

```
$ go run workerpool.go
```

```
worker 3 started job 1
```

```
worker 1 started job 2
```

```
worker 2 started job 3
```

```
worker 3 finished job 1
```

```
worker 3 started job 4
```

```
worker 2 finished job 3
```

```
worker 2 started job 5
```

```
worker 1 finished job 2
```

```
Result # 1 : 2
```

```
Result # 2 : 6
```

```
Result # 3 : 4
```

```
worker 2 finished job 5
```

```
Result # 4 : 10
```

```
worker 3 finished job 4
```

```
Result # 5 : 8
```

```
$ go run workerpool.go
```

```
worker 3 started job 1
```

```
worker 1 started job 2
```

```
worker 2 started job 3
```

```
worker 3 finished job 1
```

```
worker 3 started job 4
```

```
worker 2 finished job 3
```

```
worker 2 started job 5
```

```
worker 1 finished job 2
```

```
Result # 1 : 2
```

```
Result # 2 : 6
```

```
Result # 3 : 4
```

```
worker 2 finished job 5
```

```
Result # 4 : 10
```

```
worker 3 finished job 4
```

```
Result # 5 : 8
```

```
$ go run workerpool.go
```

```
worker 3 started job 1
```

```
worker 1 started job 2
```

```
worker 2 started job 3
```

```
worker 3 finished job 1
```

```
worker 3 started job 4
```

```
worker 2 finished job 3
```

```
worker 2 started job 5
```

```
worker 1 finished job 2
```

```
Result # 1 : 2
```

```
Result # 2 : 6
```

```
Result # 3 : 4
```

```
worker 2 finished job 5
```

```
Result # 4 : 10
```

```
worker 3 finished job 4
```

```
Result # 5 : 8
```



```
$ go run workerpool.go
```

```
worker 3 started job 1
```

```
worker 1 started job 2
```

```
worker 2 started job 3
```

```
worker 3 finished job 1
```

```
worker 3 started job 4
```

```
worker 2 finished job 3
```

```
worker 2 started job 5
```

```
worker 1 finished job 2
```

```
Result # 1 : 2
```

```
Result # 2 : 6
```

```
Result # 3 : 4
```

```
worker 2 finished job 5
```

```
Result # 4 : 10
```

```
worker 3 finished job 4
```

```
Result # 5 : 8
```

WE CAN SPEED THINGS UP



SOURCE: [HTTP://ENGLISH.SINA.COM/SPORTS/P/2011/0519/374212.HTML](http://english.sina.com/sports/p/2011/0519/374212.html)

NEITHER IS BETTER OR WORSE:
THEY'RE JUST DIFFERENT



FAVOURITE THINGS

PHP

- ▶ quick to get things done
- ▶ mature frameworks and libraries
- ▶ PHP 7

GO

- ▶ strong types
- ▶ keeping things simple
- ▶ error checking policy
- ▶ multiple return values
- ▶ interfaces
- ▶ built-in tools
- ▶ easy to run
- ▶ concurrency
- ▶ speed
- ▶ strong leadership



COMMUNITY

COULD BE BETTER

PHP

- ▶ inconsistency
- ▶ type juggling
- ▶ no dead code checking
- ▶ bit verbose?

GO

- ▶ “Is there anything at least v1.0?”
- ▶ dependency injection
- ▶ “Ugh, do I really have to write it from scratch?”

FROM PHP TO GO AND BACK:

DID ANYTHING CHANGE?



IMAGE FROM [GRACEHOPPERFILM.COM](http://gracehopperfilm.com)

SHORTER, SENSIBLE NAMES

```
namespace Controllers;
```

```
class HomeController { ... }
```


HANDLE ERRORS FIRST

```
if (!empty($name)) {  
    if (!is_string($name)) {  
        return false;  
    }  
    return true;  
} else {  
    return false;  
}
```

```
if (empty($name)) {  
    return false;  
}  
  
if (!is_string($name)) {  
    return false;  
}  
  
return true;
```

INTERFACES

Go's implicit interfaces:

has a, rather than *is a*

If a struct **has** methods A, B, C, then it implements interface X.

If an object **is** of type X, then it has methods A, B, C.

INTERFACES

GO

```
type Shape interface {  
    Area() float64  
}  
type Circle struct {  
    radius float64  
}  
  
func (c Circle) Area() float64 {  
    return math.Pi * c.radius * c.radius  
}
```

PHP

```
class Circle implements Shape { ... }
```

INTERFACES

GO

```
type Shape interface {  
    Area() float64  
}  
type Circle struct {  
    radius float64  
}  
  
func (c Circle) Area() float64 {  
    return math.Pi * c.radius * c.radius  
}
```

PHP

```
class Circle implements Shape { ... }
```

INTERFACES

GO

```
type Shape interface {  
    Area() float64  
}  
type Circle struct {  
    radius float64  
}  
  
func (c Circle) Area() float64 {  
    return math.Pi * c.radius * c.radius  
}
```

PHP

```
class Circle implements Shape { ... }
```

WHERE DO I START?

<https://golang.org/>

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

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

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

<https://gobyexample.com>

@KASIAZIEN

TALKS

Concurrency is not parallelism

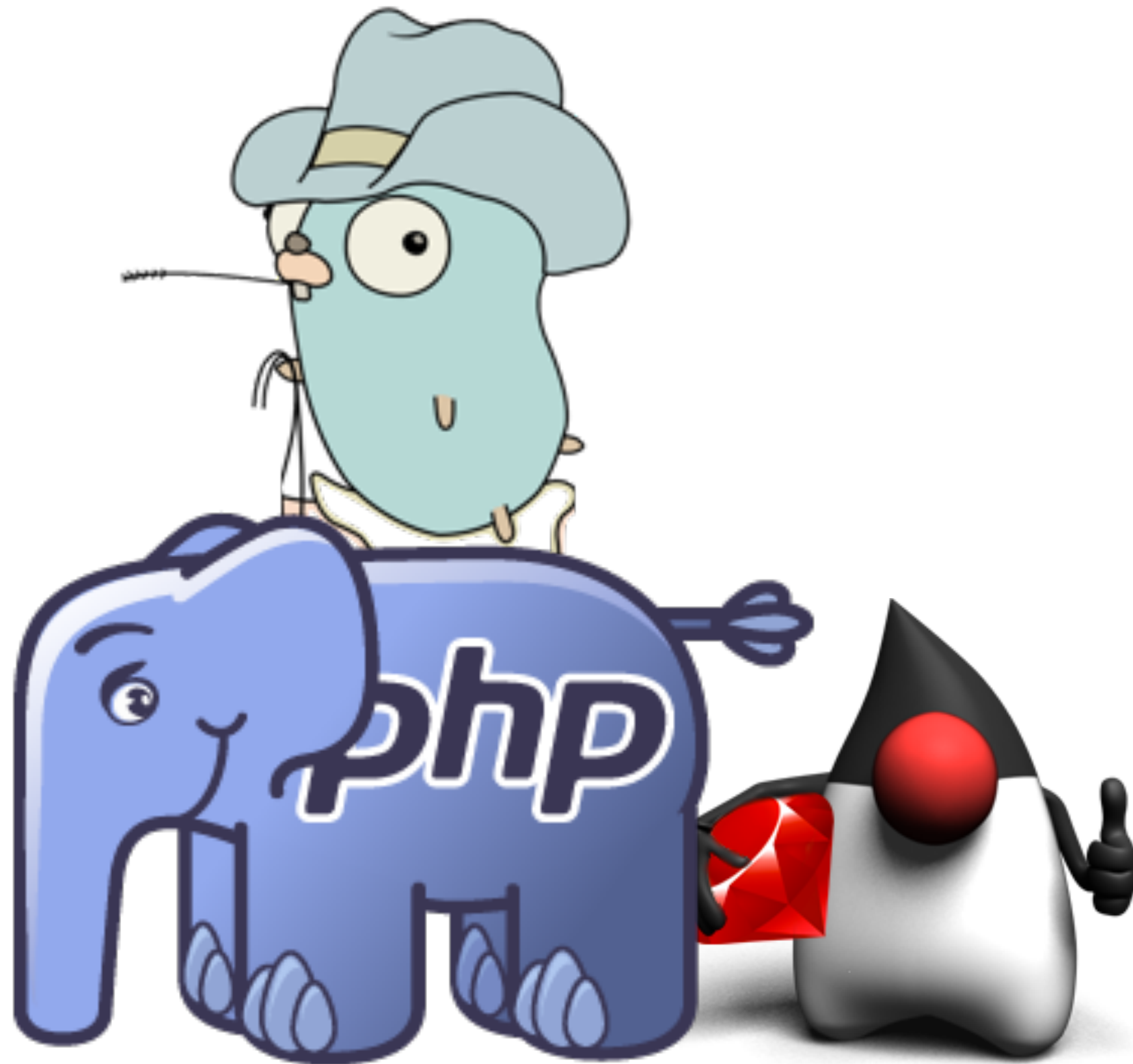
<https://blog.golang.org/concurrency-is-not-parallelism>

Simplicity is complicated

https://www.youtube.com/watch?v=rFejpH_tAHM

Building containers in Go

<https://www.youtube.com/watch?v=HPuvDm8IC-4>



THANKS FOR LISTENING!



SLIDES: [HTTPS://GITHUB.COM/KATZIEN/TALKS/TREE/MASTER/GET-GOING-WITH-A-NEW-LANGUAGE/PHPHANTS-2018-04-11/SLIDES.PDF](https://github.com/katzien/talks/tree/master/get-going-with-a-new-language/phpphants-2018-04-11/slides.pdf)

@KASIAZIEN

IMAGE FROM [HTTPS://GITHUB.COM/GENKO/GOSHIP](https://github.com/genko/goship)