import golang;
struct Microservice

Giulio De Donato - Giorgio Cefaro

rate our talk!
https://joind.in/14104

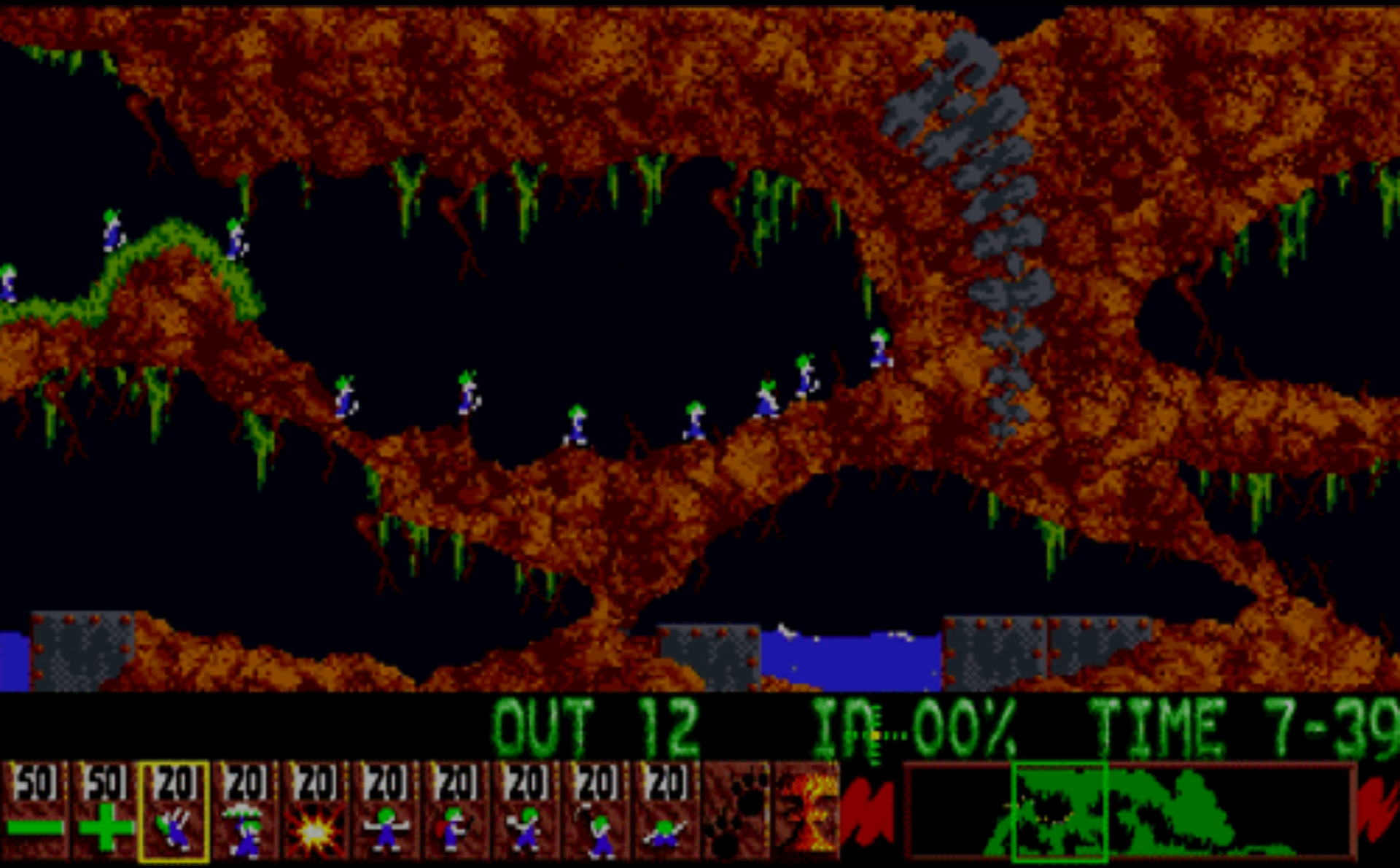
{codemotion}

```
import "giorrrgio"
```

Giorgio Cefaro

Freelance Software Engineer

@giorrrgio



Single Responsability Principle - The Lemmings metaphor

```
import "liuggio"
```

Giulio De Donato

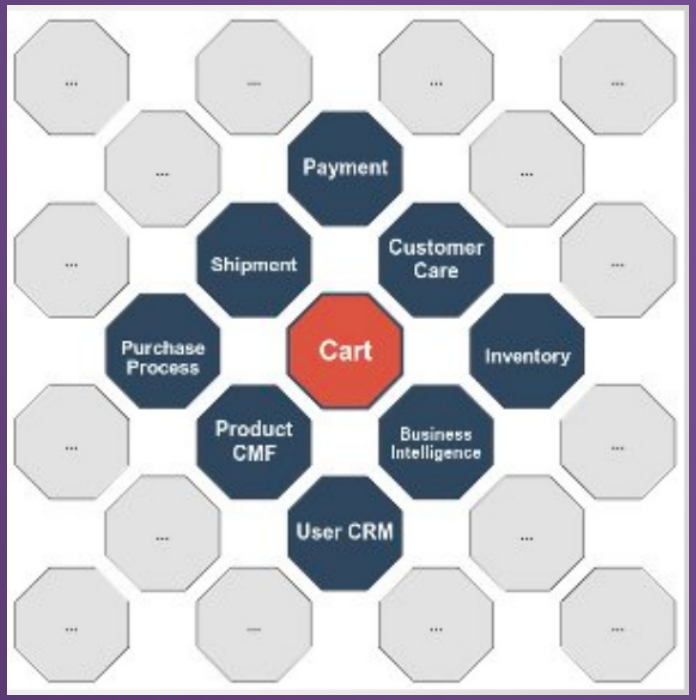
CTO @ chupamobile

@liuggio

Once upon a time ... Monolith

One Company
One Application
One Deploy
One Server
One Database
The developer has a permanent contract with the monolith
is the guardian of all the tricks

BIG complex problem
vs
lots of small simple problems



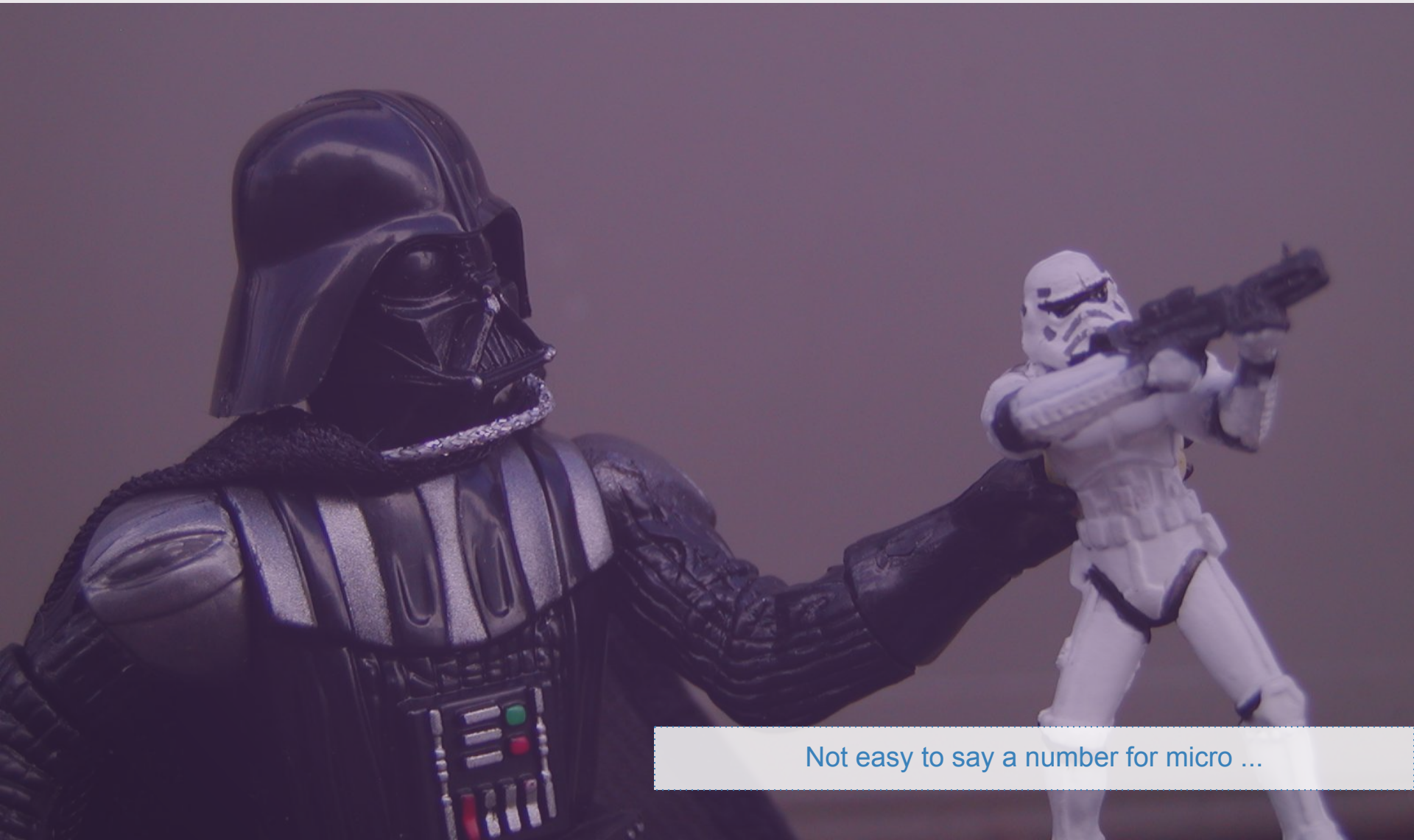
Bounded context
small components
SOA vs ….

MICROSERVICES:

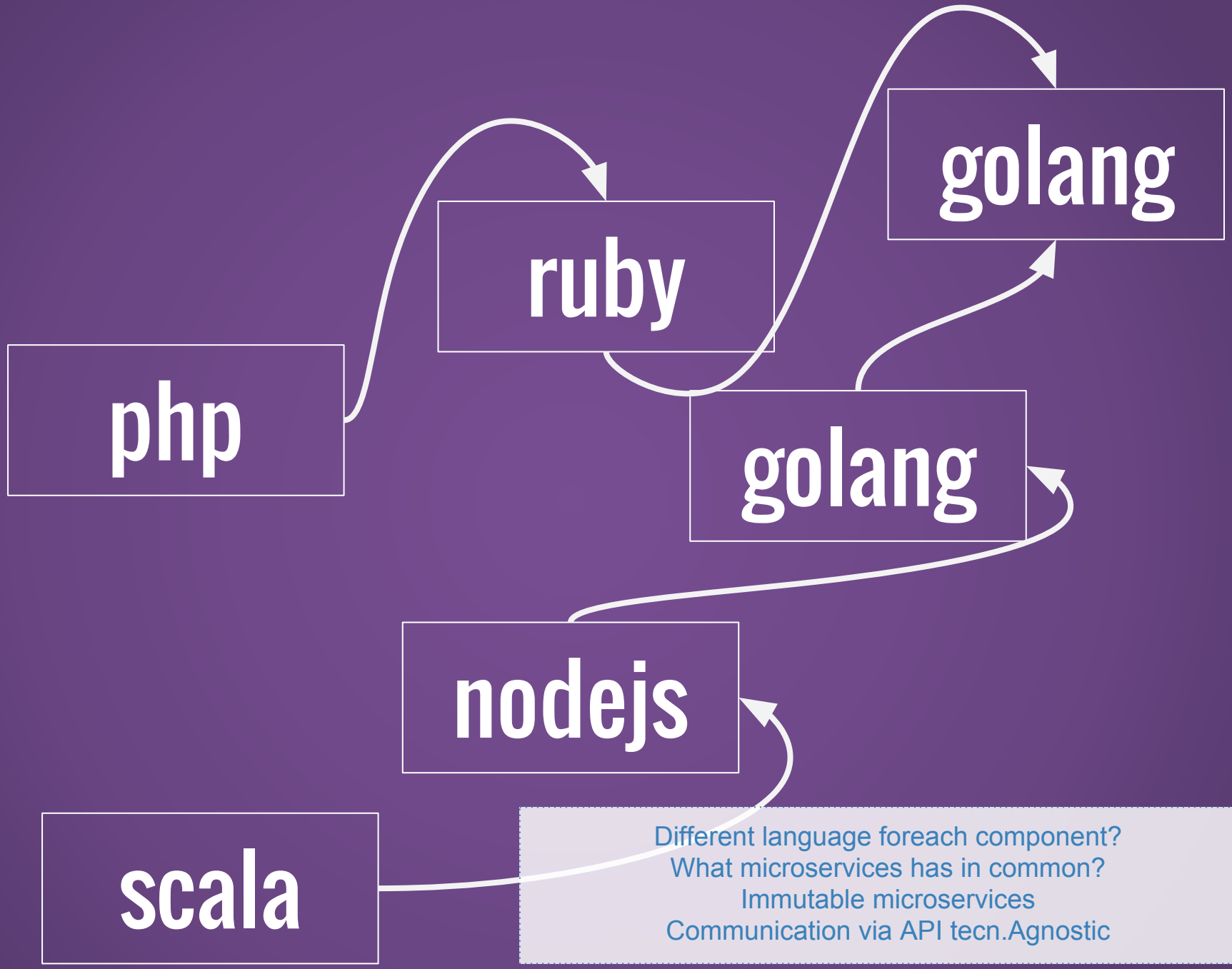
While there is **no precise definition** of this architectural style, there are **certain common characteristics** around **organization** around **business capability**, **automated deployment, intelligence in the endpoints, and decentralized control of languages and data.**

– martin fowler

How small is “micro”?



Not easy to say a number for micro ...

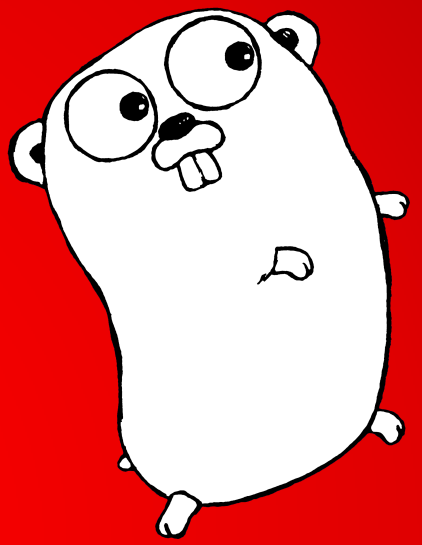

golang
ruby
php
golang
nodejs
scala
Different language foreach component?
What microservices has in common?
Immutable microservices
Communication via API tecn.Agnostic

GOLANG

Why we chose it

2

GOLANG



static typing

fast

less is more

concurrency made easy

simple and concise syntax

short learning curve

GOLANG - PLATFORMS AND ARCHITECTURES

Go programs are statically compiled

Go compiler target multiple platforms and architectures

Linux, Mac OS X, FreeBSD, NetBSD, OpenBSD, Plan 9, and Microsoft Windows OS and i386, amd64, ARM and IBM POWER architectures are currently supported

NO EXTERNAL LIBRARIES

NO VIRTUAL MACHINES

NO JIT-COMPILING

JUST A STATIC EXECUTABLE

GOLANG - CONCURRENCY

Concurrency is easy to implement through **GOROUTINES** , each goroutine having a **SMALL FOOTPRINT** of memory and being **MULTIPLEXED** through OS threads to avoid that blocking routines can block other running goroutines

GOLANG - PACKAGES

Go supports modern web technologies through a set of bundled packages, ready to import.

archive, bufio, builtin, bytes, compress, container, crypto, **database**, debug, **encoding**, errors, expvar, flag, fmt, go, hash, **html**, image, index, io, log, math, mime, **net**, os, path, reflect, regexp, runtime, sort, strconv, strings, suffixarray, sync, syscall, **testing**, text, time, unicode, unsafe

```go
// hello_codemotion.go
package main

import "fmt"

func main() {
  // Create a channel to synchronize goroutines
  done := make(chan bool)

  // Execute println in goroutine
  go func() {
    fmt.Println("Hello Codemotion")

    // Tell the main function everything is done.
    // This channel is visible inside this goroutine because
    // it is executed in the same address space.
    done <- true
  }()

  fmt.Println("Bye Codemotion")
  <-done // Wait for the goroutine to finish. what if we
         // remove it?
}
```

Even if we run the goroutine that will print the “Hello” as first, its output will be echoed once it is synchronized with main (the “<- done” final line)

```
$ go build hello_codemotion.go

$ ./hello_codemotion

Bye Codemotion
Hello Codemotion
```

GOLANG - COMMUNICATION

Network I/O is supported through the **net** package, which comes with **TCP/IP**, **UDP**, **DNS**, and **Unix domain socket** components.

Low level support is provided, but you will probably only need **Dial**, **Accept** and **Listen**

GOLANG - COMMUNICATION

The **net/http** package provides **http client** and **server** implementations.

Building a web server is quite easy!

```
package main

import "net/http"
import "log"
import "fmt"\

func main() {

   http.HandleFunc("/hello", func(w http.ResponseWriter, r
*http.Request) {
     fmt.Fprintln(w, "Hello, Codemotion!")
   })

   log.Fatal(http.ListenAndServe(":8080", nil))
}
```

GOLANG - COMMUNICATION

The **encoding** package provides **interfaces** shared by other packages that convert data to and from byte-level and textual representations:

encoding/json

encoding/xml

encoding/gob

encoding/json package provides structures to read and write JSON data

```
type Message struct {
    Name string
    Body string
    Time int64
}

m := Message{"Codemotion", "Hello", 1294706395881547000}

b, err := json.Marshal(m)

// b will be
// {"Name":"Alice","Body":"Hello","Time":1294706395881547000}
```

encoding/gob package provides native golang RPC with binary transmission of structures

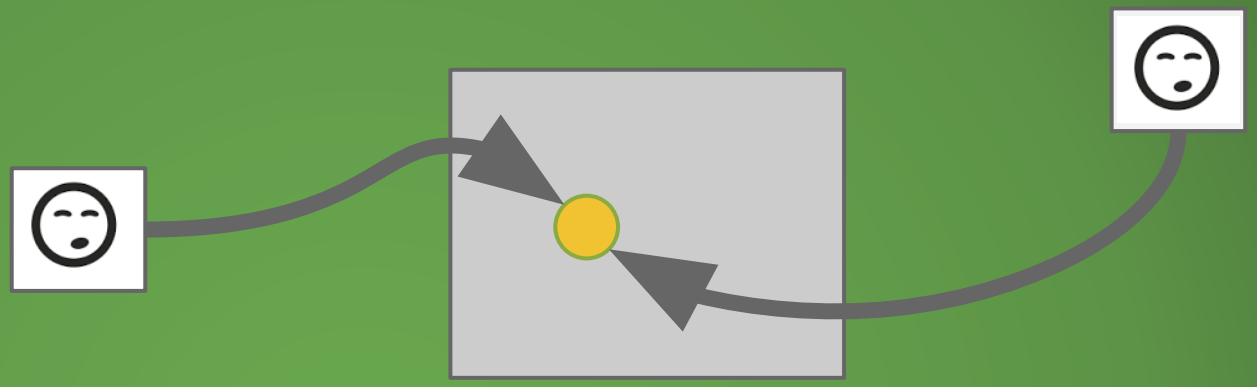
```
type Foo struct {
    A, B int64
}

func main() {
    conn, err := net.Dial("tcp", "localhost:8080")
    if err != nil {
        log.Fatal("Connection error", err)
    }
    encoder := gob.NewEncoder(conn)
    foo := &Foo{1, 2}
    encoder.Encode(foo)
    conn.Close()
 }
```

INTERFACE ALL THE THINGS

3

explicit is **always** better than implicit

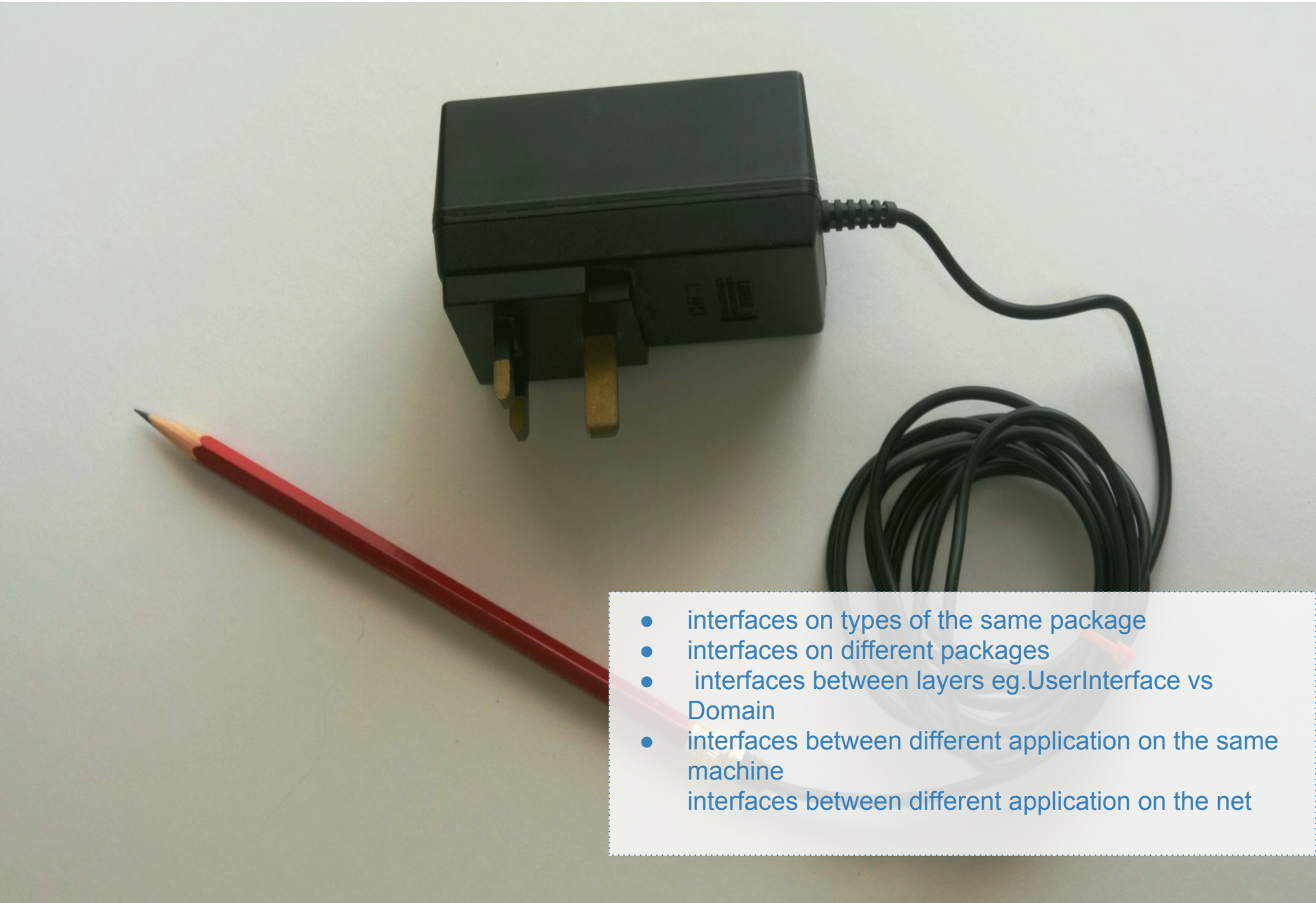
My always True sentence



Expose behaviour

person.GoAhed() vs person.foot += 1meter

Interface all the things



- interfaces on types of the same package
- interfaces on different packages
- interfaces between layers eg.UserInterface vs Domain
- interfaces between different application on the same machine

  interfaces between different application on the net

```
cat access.log | grep “porn” | wc -l
```

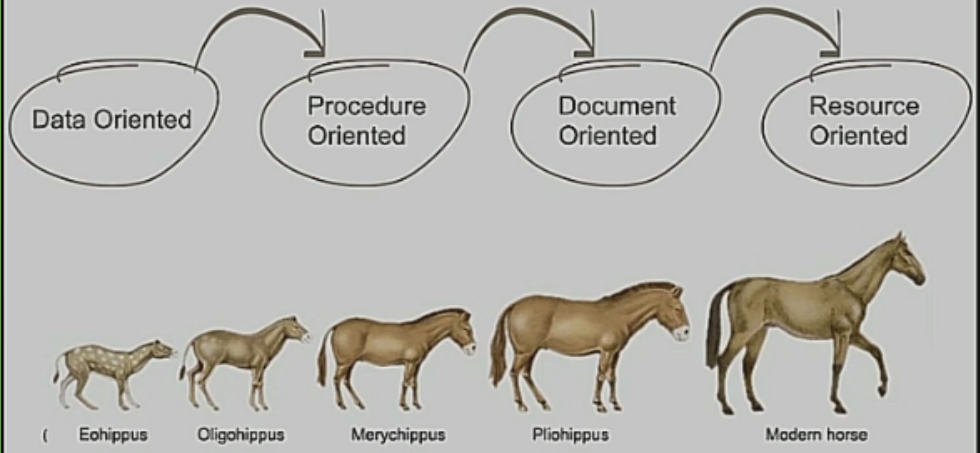
Unix Pipe revolution

ThoughtWorks®

PRACTICAL CONSIDERATIONS FOR MICRO SERVICES

Sam Newman
Javazone 2014, September 2014

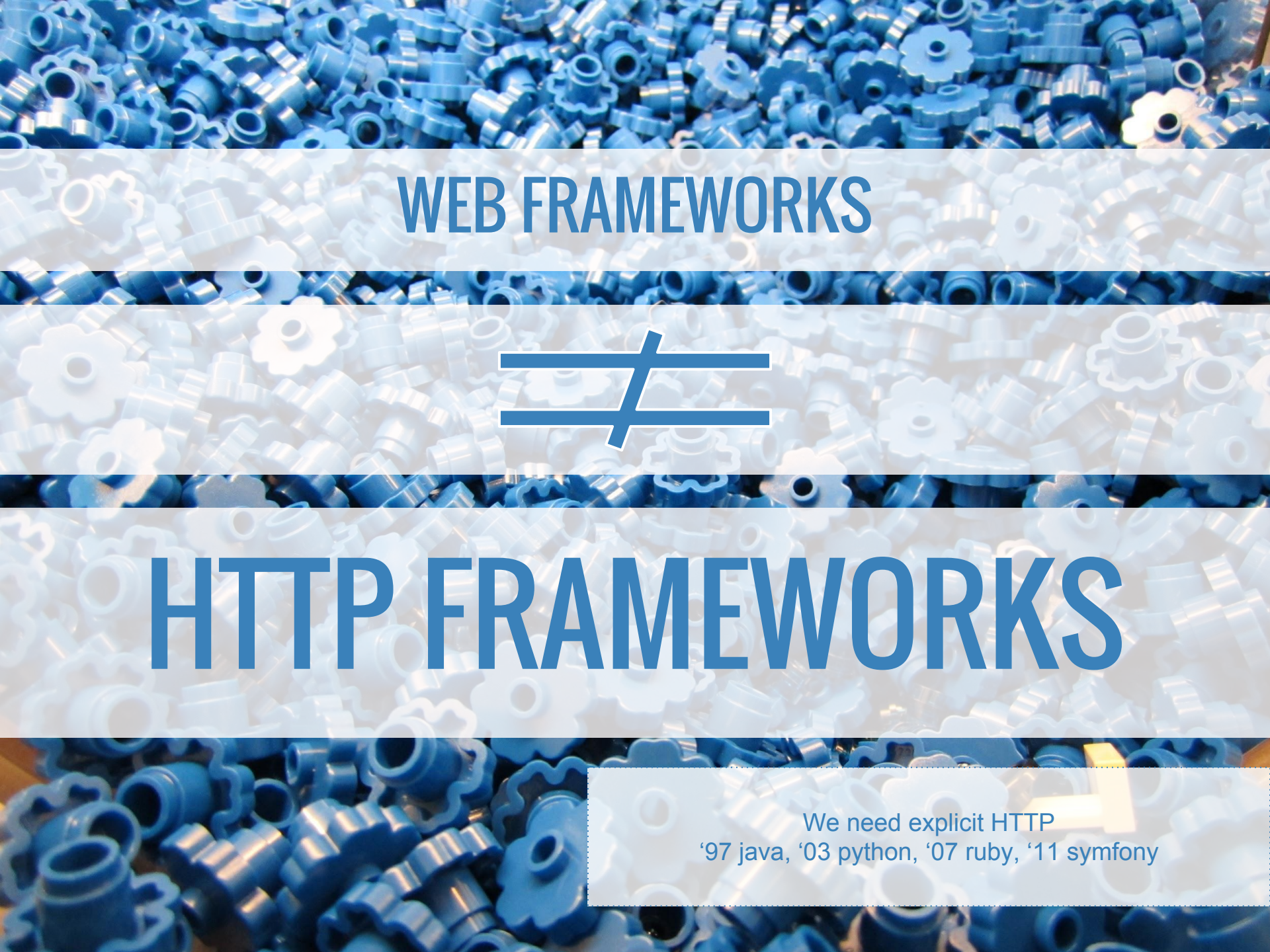
Integration Styles An Evolutionary View




HTTP WINS
HTTP is the protocol, everybody has to study its RFC
HTTP is not easy, Request/Response

WEB FRAMEWORKS

There’s no more web framework in Golang,
Martini example
Web Framework implicit net protocol


WEB FRAMEWORKS
≠
HTTP FRAMEWORKS
We need explicit HTTP
‘97 java, ‘03 python, ‘07 ruby, ‘11 symfony

```go
func helloHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello World!\n")
}

func main() {
    log.Println("Starting 8080")
    http.HandleFunc("/", helloHandler)
    http.ListenAndServe(":8080", nil)
}
```

```
$ go run main.go
2015/03/24 22:08:55 Starting 8080
```

```
$ curl localhost:8080
Hello World!
```

We are using explicit interface for Request/Response

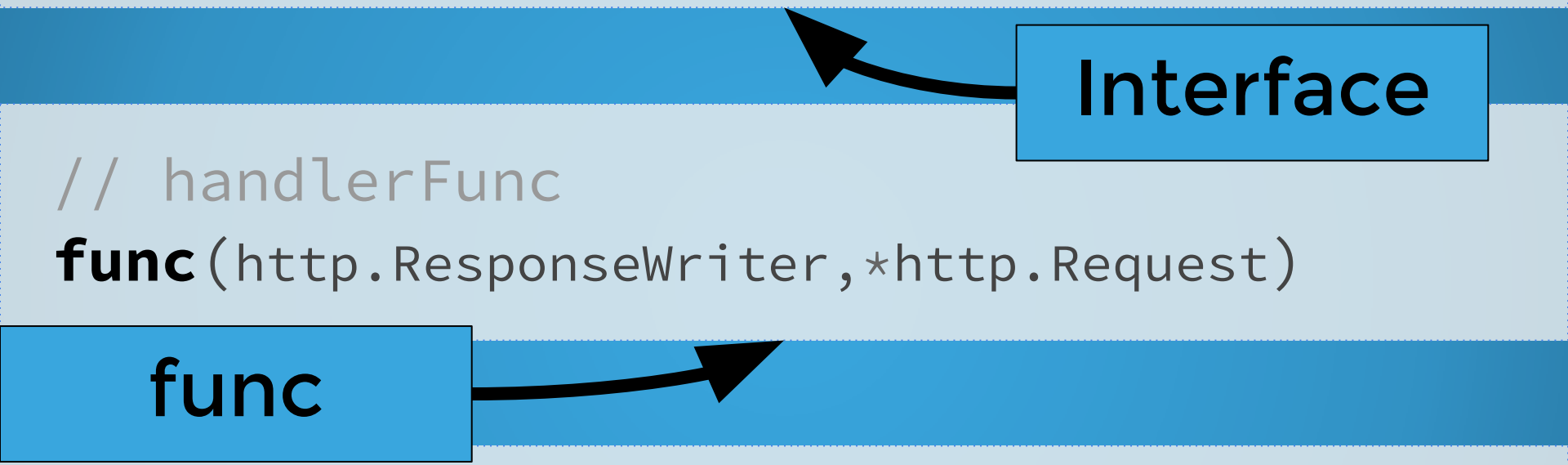
GOLANG
HTTP INTERFACE

```
type Handler interface {
    ServeHTTP(ResponseWriter, *Request)
}
```

The interface needs a ServeHTTP method
but we use a function with no ServeHTTP interface
how?

```
func helloHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello World!")
}
```

```
type Handler interface {
   ServeHTTP(http.ResponseWriter,*http.Request)
}
```




```
// handlerFunc
func(http.ResponseWriter,*http.Request)
```



```
handler := http.HandlerFunc(handlerFunc)
handler.ServeHTTP(w, r)
```

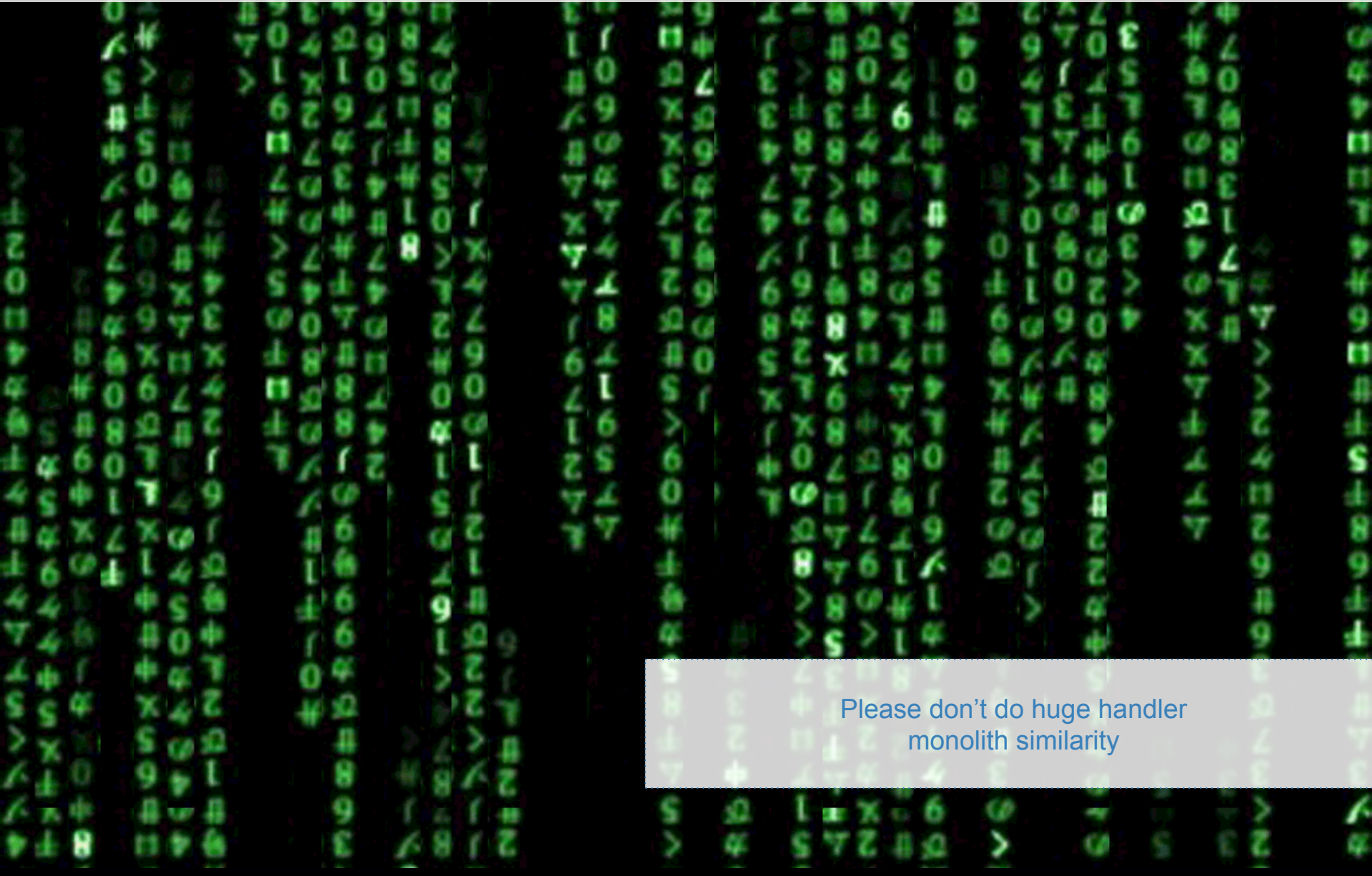
Unit test on handlers

```go
func helloHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello World!")
}
```

```go
func TestShouldResponse(t *testing.T) {

    req, _ := http.NewRequest("GET", "/liuggio", nil)
    res := httptest.NewRecorder()

    helloHandler(res, req)

    bodyBytes, _ := ioutil.ReadAll(res.Body)
    if !bytes.Contains(bodyBytes, "Hello World!") {
        t.Error("Hello World not found on response body")
    }
}
```

We can also unit test the handler

Huge handler



Please don’t do huge handler
monolith similarity

Compositions

```
func f(val int) int {
    return val+1
}

func g(val int) int {
    return val*2
}

f(g(1)) // 3

g(f(1)) // 4
```

Higher-order function

```
func makeFunction(name string) func() {
    return func() {
        fmt.Printf("Hello %s", name)
    }
}

func main() {
    f := makeFunction("liuggio")
    f()
}
```

```go
func logger(handlerFunc http.HandlerFunc) http.HandlerFunc {

    return func(w http.ResponseWriter, r *http.Request) {
        start := time.Now()
        log.Printf("Access [%s] %s", r.Method, r.URL.Path)

        handler := http.HandlerFunc(handlerFunc)
        handler.ServeHTTP(w, r)

        log.Printf("Completed %q\n", time.Since(start))
    }
}
```

EXECUTE handlerFunc

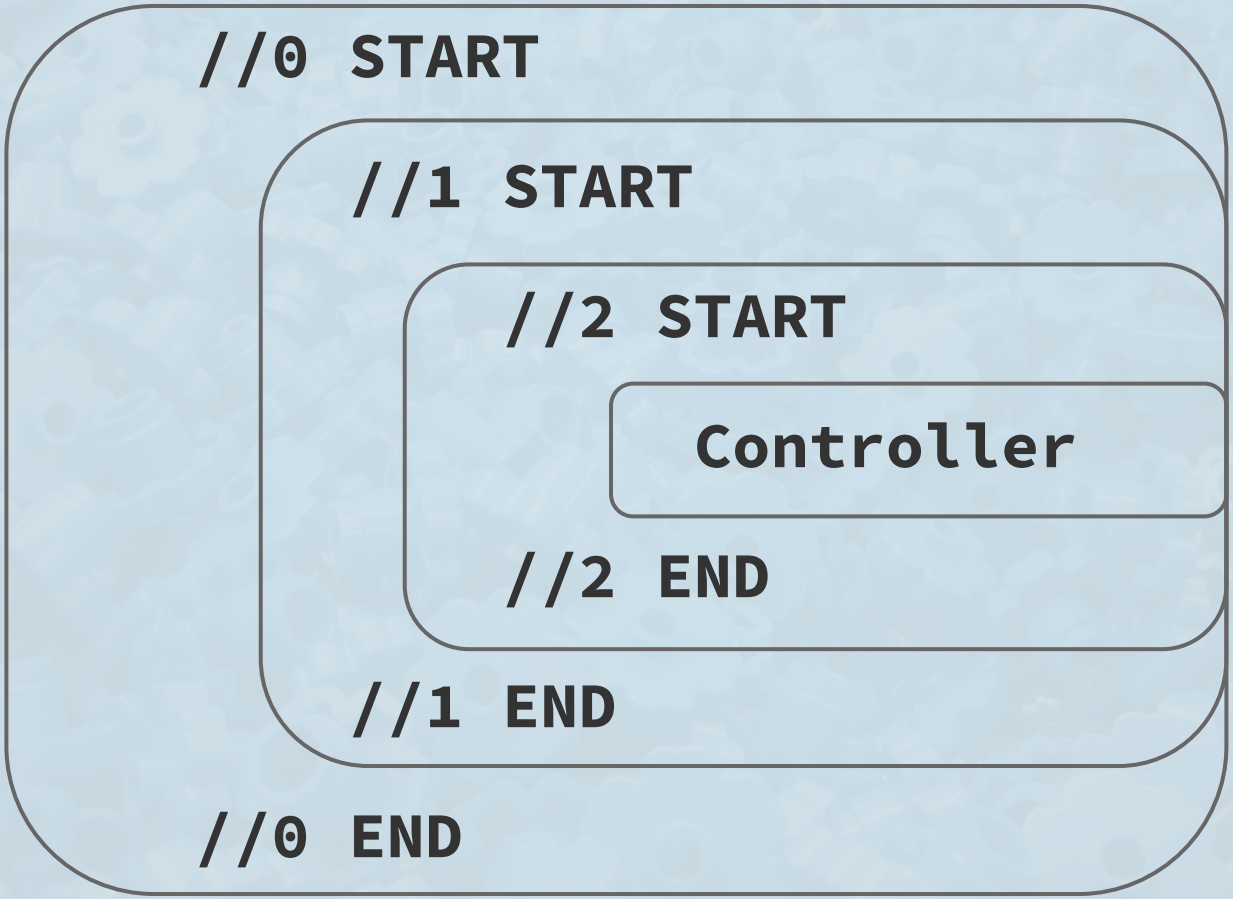
```go
func main() {
    log.Println("Starting 8080")
    http.HandleFunc("/", logger(helloHandler))
    http.ListenAndServe(":8080", nil)
}
```

```go
func logger(handlerFunc http.HandlerFunc) http.HandlerFunc {

    return func(w http.ResponseWriter, r *http.Request) {
        start := time.Now()
        log.Printf("Access [%s] %s", r.Method, r.URL.Path)

        handler := http.HandlerFunc(handlerFunc)
        handler.ServeHTTP(w, r)

        log.Printf("Completed %q\n", time.Since(start))
    }
}
```

```
$ go run main.go middleware_logger.go
2015/03/24 22:18:42 Starting 8080
2015/03/24 22:18:44 Access [GET] /
2015/03/24 22:18:44 Completed "35.06µs"
```

```
$ curl localhost:8080
Hello World!
```

middleware



middleware

- **logger** request logger with custom format support
- **csrf** Cross-site request forgery protection
- **compress** Gzip compression middleware
- **basicAuth** basic http authentication
- **bodyParser** extensible request body parser
- **json** application/json parser
- **urlencoded** application/x-www-form-urlencoded parser
- **multipart** multipart/form-data parser
- **timeout** request timeouts
- **cookieParser** cookie parser
- **session** session management support with bundled MemoryStore
- **cookieSession** cookie-based session support
- **methodOverride** faux HTTP method support
- **responseTime** calculates response-time and exposes via X-Response-Time
- **staticCache** memory cache layer for the static() middleware
- **static** streaming static file server supporting Range and more
- **directory** directory listing middleware
- **vhost** virtual host sub-domain mapping middleware
- **favicon** efficient favicon server (with default icon)
- **limit** limit the bytesize of request bodies
- **query** automatic querystring parser, populating req.query
- **errorHandler** flexible error handler
- …. many more

middleware

```
http.HandleFunc("/", logger(helloHandler)))
http.HandleFunc("/admin", logger(httpAuth(helloHandler))))
http.HandleFunc("/api", logger(httpAuth(jsonEncode(helloHandler))
http.ListenAndServe(":8080", nil)
```

The future is already arrived —

it's just not very evenly

distributed

-- william gibson

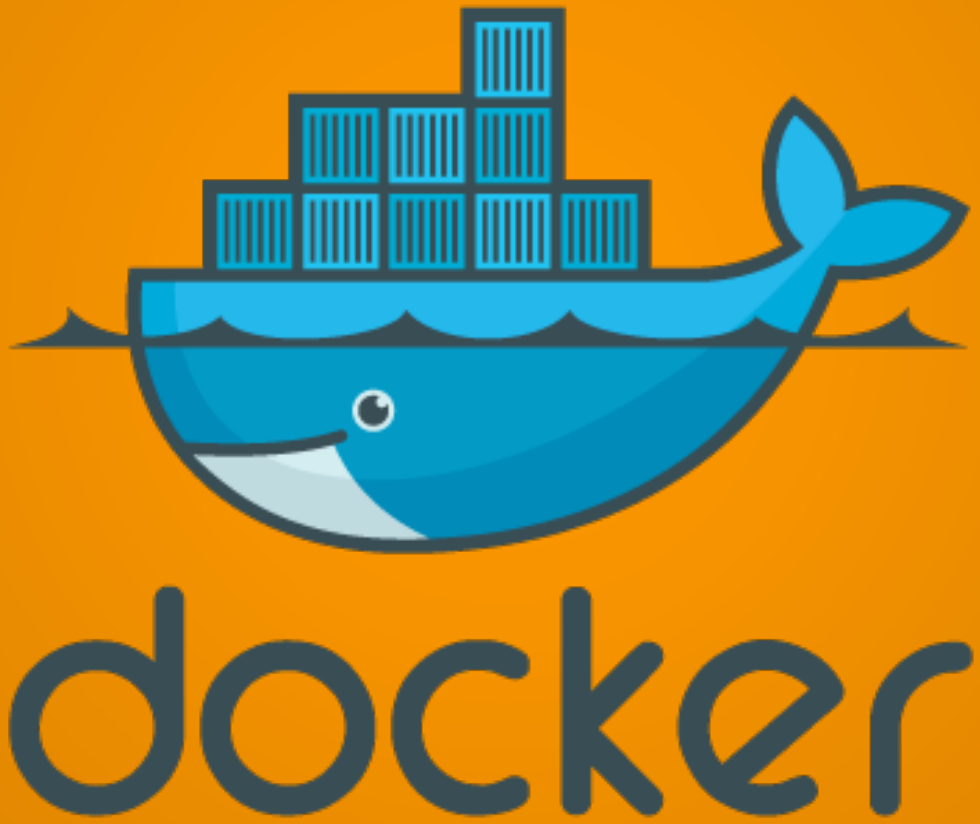
Microservices need AUTOMATION

AUTOMATION

Build, Deploy, Scale

4

ARCHITECTURE - DOCKER



ARCHITECTURE - DOCKER

Docker is an open-source project that automates the deployment of applications inside software containers.

http://en.wikipedia.org/wiki/Docker_(software)

ARCHITECTURE - DOCKER

Docker allows independent "containers" to run within a single Linux instance, avoiding the overhead of starting virtual machines.

http://en.wikipedia.org/wiki/Docker_(software)

A **container** can be configured through a **Dockerfile**

In this basic configuration, we specify that our image is derived **FROM** the **golang** image, available through the **Docker Hub,** and that our container will **EXPOSE** port 3000

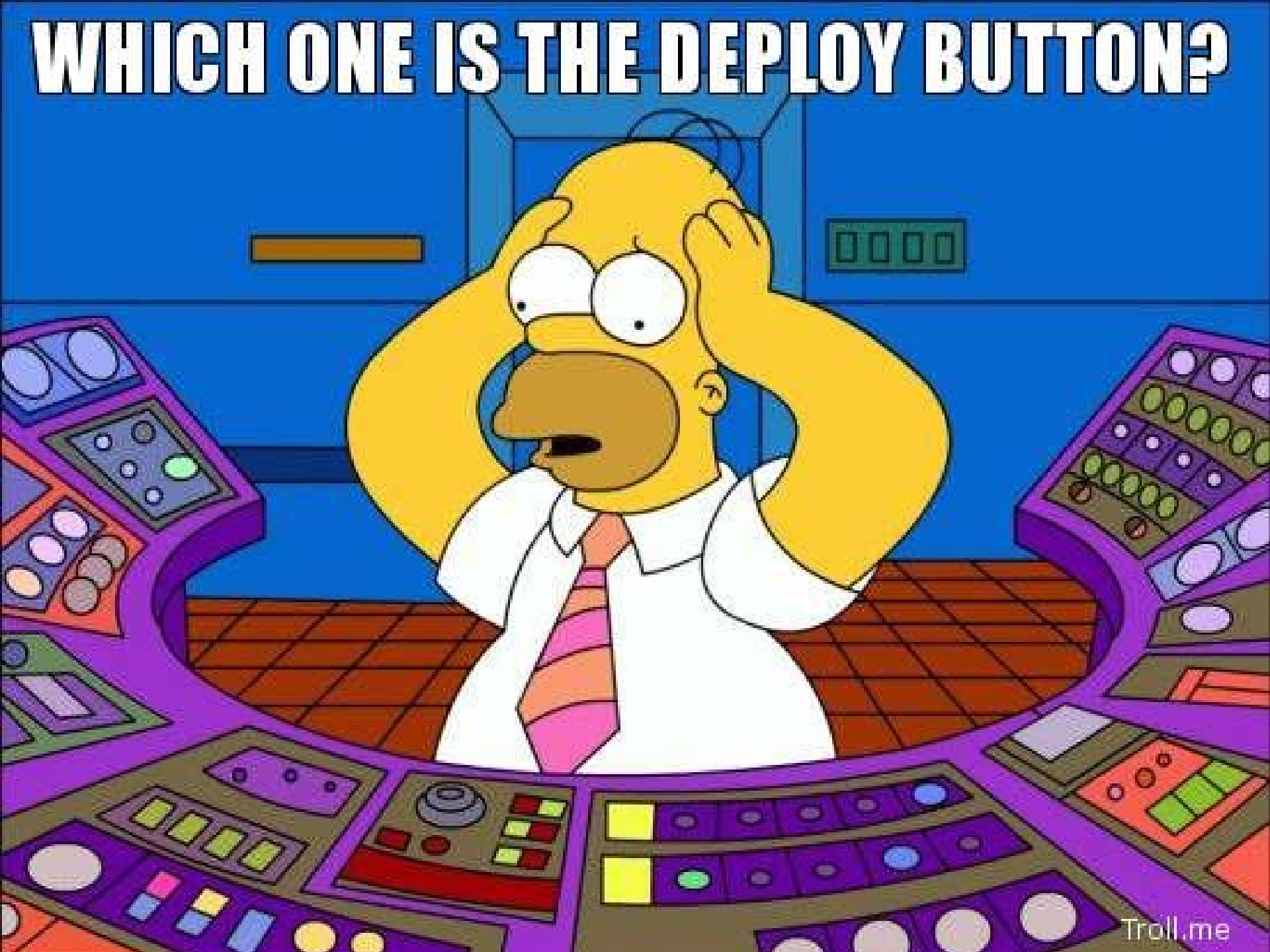
```
# Dockerfile
# golang image where workspace (GOPATH) configured at /go.
FROM golang:1.4-onbuild
EXPOSE 3000
```

https://docs.docker.com/reference/builder/

The golang **image** is a great solution for go microservices, it is configured to take care of compiling our program and copy it inside the container, ready to **go** with a single command. Cross compiling is supported too!

```
$ sudo docker build -t codemotion_container .
```

https://registry.hub.docker.com/_/golang/


WHICH ONE IS THE DEPLOY BUTTON?
Troll.me

Microservices need AUTOMATION

remember???

Microservices need ORCHESTRATION

ARCHITECTURE - DOCKER ORCHESTRATION TOOLS

- Provision Docker on any infrastructure, from laptop to public cloud instance
- Compose an app using both proprietary containers and Docker Hub Official Repos
- Manage all containers of an app as a single group
- Cluster an application's containers to optimize resources and provide high-availability

http://blog.docker.com/2015/02/orchestrating-docker-with-machine-swarm-and-compose/

ARCHITECTURE - DOCKER MACHINE

- Provision Docker Engines across various providers both local and remote, secured with TLS, or not.
- Lightweight management of machines: Starting, stopping, removing, etc.
- Run remote commands or log in to machines via SSH
- Upgrade the Docker Engine when a new version is released

http://blog.docker.com/2015/02/announcing-docker-machine-beta/

With **Machine** docker hosts can be spawned and controlled on different computers and virtual machines from you local docker client. Different clouds too!

Machine supports Amazon EC2, Microsoft Azure, Microsoft Hyper-V DigitalOcean, Google Compute Engine, OpenStack, Rackspace, SoftLayer, VirtualBox, VMware Fusion, VMware vCloud Air, VMware vSphere and counting!

```
$ docker-machine create -d virtualbox dev
[info] Downloading boot2docker...
[info] Creating SSH key...
[info] Creating VirtualBox VM...
[...]
$ docker run busybox echo hello world
hello world
```

ARCHITECTURE - DOCKER SWARM

- Native clustering for Docker
- Swarm is a standard Docker image
- Run one command to create a cluster.
- Run another command to start Swarm.
- On each host where the Docker Engine is running, run a command to join said cluster.

http://blog.docker.com/2015/02/scaling-docker-with-swarm/

```
$ docker run swarm create
5d9bc2f39ccc00500b36f23d90994f5f # <- cluster_id

# swarm master
$ docker-machine create -d virtualbox --swarm --swarm-master --
swarm-discovery token://5d9bc2f39ccc00500b36f23d90994f5f my-
swarm

# swarm nodes
$ docker-machine create -d virtualbox --swarm --swarm-discovery
token://5d9bc2f39ccc00500b36f23d90994f5f my-swarm-node1

$ docker-machine create -d virtualbox --swarm --swarm-discovery
token://5d9bc2f39ccc00500b36f23d90994f5f my-swarm-node3
```

Run two redis servers, but don’t run both on the same machine:

```
$ docker run -d --name redis_1 -e ‘affinity:container!=redis_*’ redis
$ docker run -d --name redis_2 -e ‘affinity:container!=redis_*’ redis
```

Add a constraint on the type of storage, we want a machine that has SSD storage:

```
$ docker run -d -e constraint:storage==ssd mysql
```

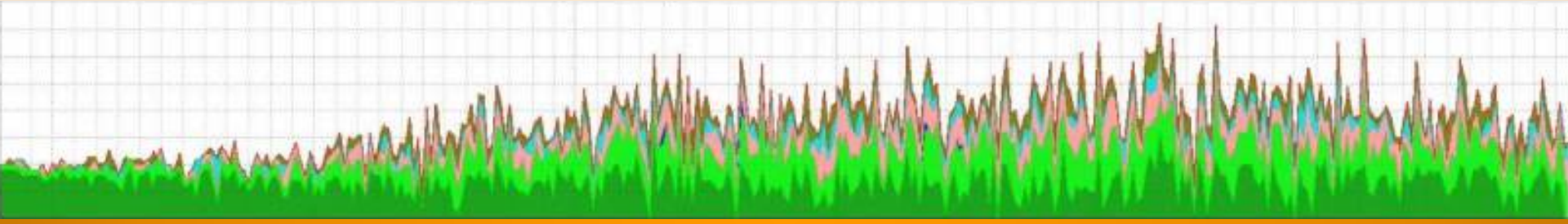
ARCHITECTURE - DOCKER COMPOSE

- Define your application's components in a single file
- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

https://blog.docker.com/2015/02/announcing-docker-compose/

```
# docker-compose.yml
cart:
  build: ./cart
  links:
    - redis
  ports:
    - "5000:5000"
shipment:
  build: ./shipment
  links:
    - mongo
  ports:
    - "5000:5000"
redis:
  image: redis
mongo:
  image: mongodb
```

Microservices need MONITORING



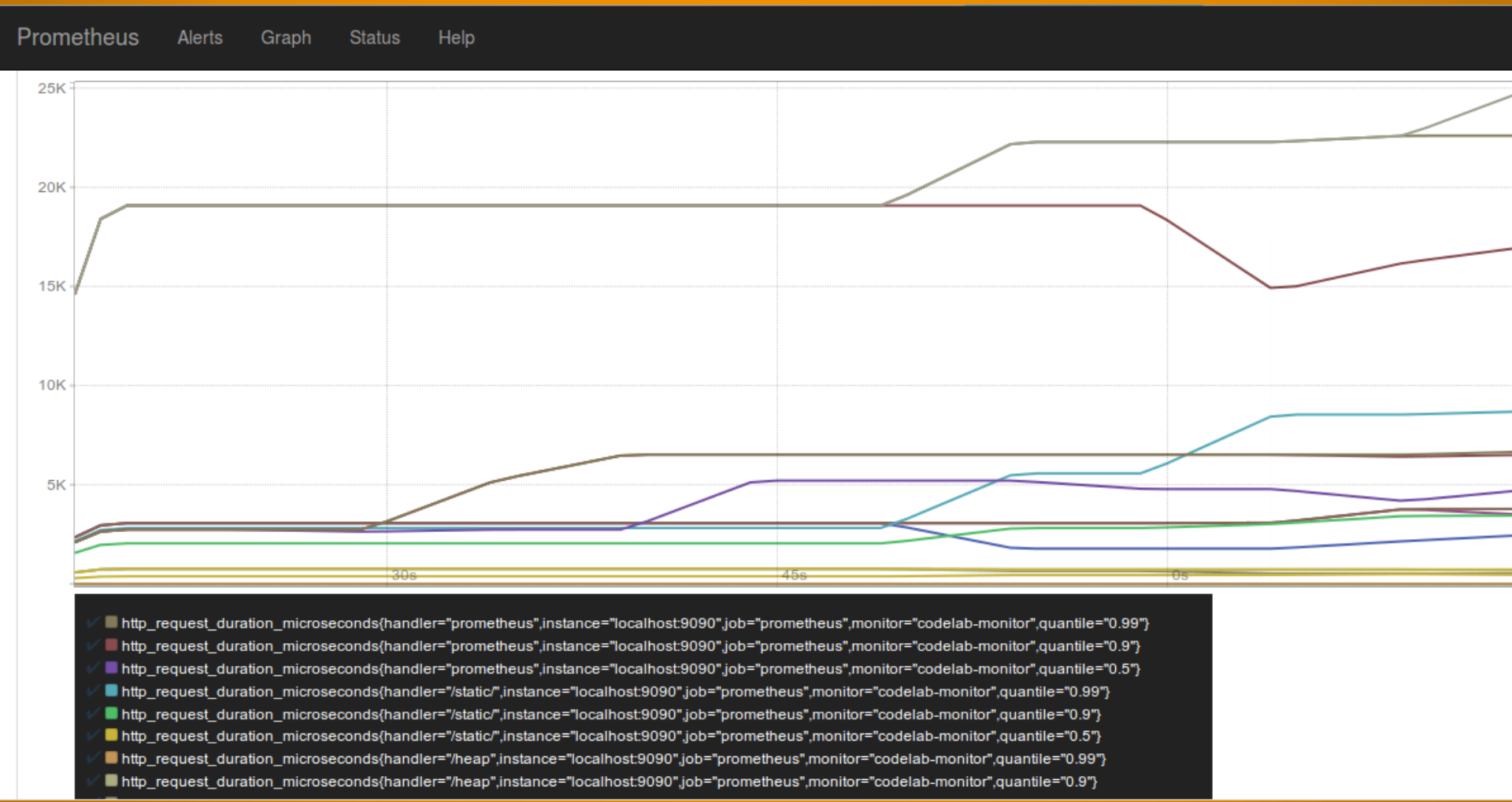

SO YOU WANT A
NEW MONITORING TOOL
TELL ME MORE ABOUT WHAT YOU
DID WITH TODAY'S 32456 ALERTS
imgflip.com

MONITORING - SOUNDCLOUD'S PROMETHEUS

- Recently open-sourced by SoundCloud
- Written in GO with native client for in-application monitoring
- Easy docker container monitoring
- Highly dimensional data model
- Flexible query language
- Powerful metrics types
- Alerting on any expression
- No fracking dependencies

http://5pi.de/2015/01/26/monitor-docker-containers-with-prometheus/

```
$ docker run -p 9090:9090 prom/prometheus
```



THERE'S NO GOOD TALK WITH NO REFERENCES

https://vimeo.com/105751281 PCFMA

http://www.youtube.com/watch?v=QY8mL6WARIE HTTP interface is a lie

http://martinfowler.com/articles/consumerDrivenContracts.html

http://www.infoq.com/news/2014/07/building-deploying-microservices

https://talks.golang.org/2014/readability.slide#27

http://www.morewords.com/contains/go/

http://martinfowler.com/articles/microservices.html (come non citarlo :))

https://blog.golang.org/docker (figata che docker ha il suo env golang)

https://speakerdeck.com/stilkov/microservices-talk-berlin

http://www.infoq.com/articles/microservices-practical-tips

http://nginx.com/blog/microservices-at-netflix-architectural-best-practices/

http://www.reddit.
com/r/golang/comments/252wjh/are_you_using_golang_for_webapi_development_what/

https://justinas.org/embrace-gos-http-tools/

https://news.ycombinator.com/item?id=6869710

https://crate.io/blog/deploying-crate-with-docker-machine-swarm/

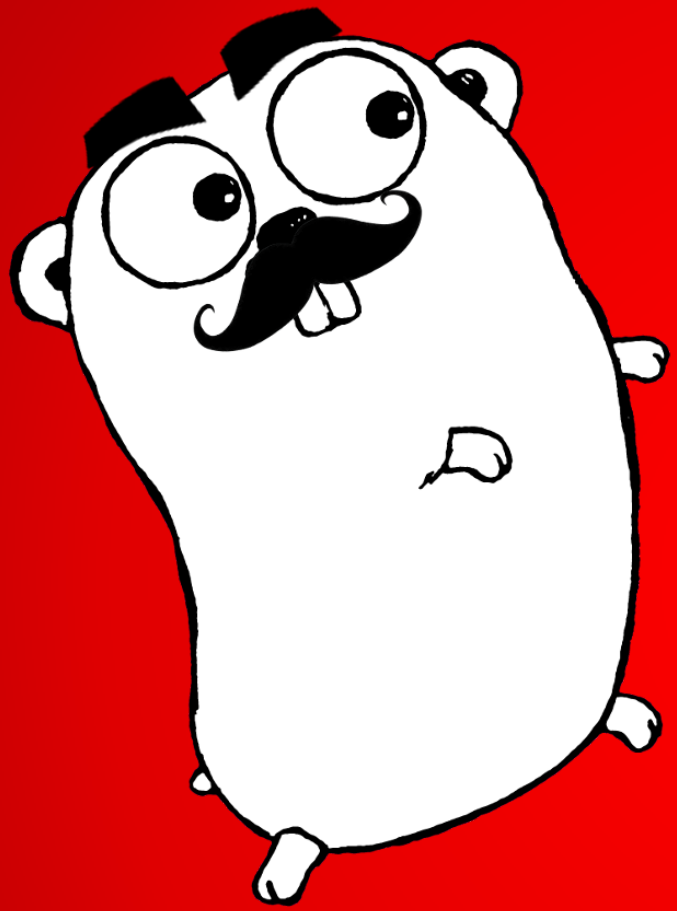
CREDITS

http://en.wikipedia.org/wiki/List_of_largest_monoliths_in_the_world

https://www.flickr.com/photos/robwatling/3411172879

https://www.flickr.com/photos/dsevilla/139656712_FLOWER

https://www.flickr.com/photos/nickpiggott/5212359135_BRICKS



JOIN GOLANGIT!

http://goo.gl/9am5v0

Questions? Answers?

https://joind.in/14104