Go-ing Forward

With Golang

Me

Roy Pearl

19+ years of h.o. dev
12 years of R&D management
9 years of technical coaching

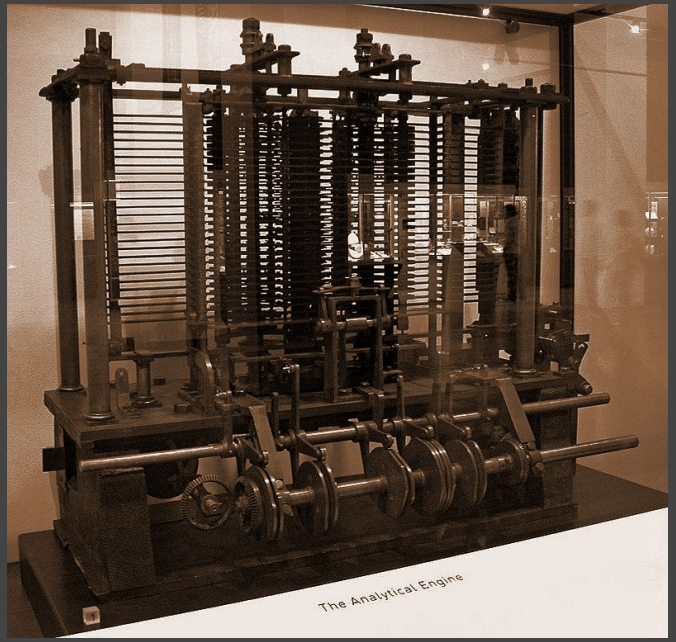
Currently

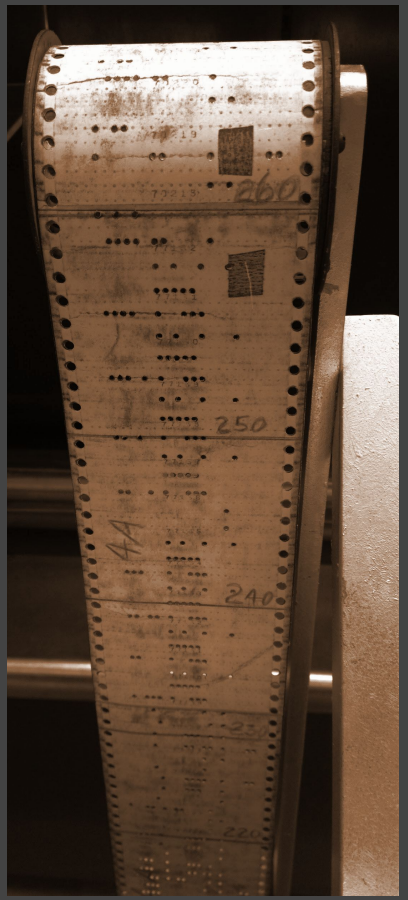
Co-Founder & CTO @ Vica

Once upon a time...

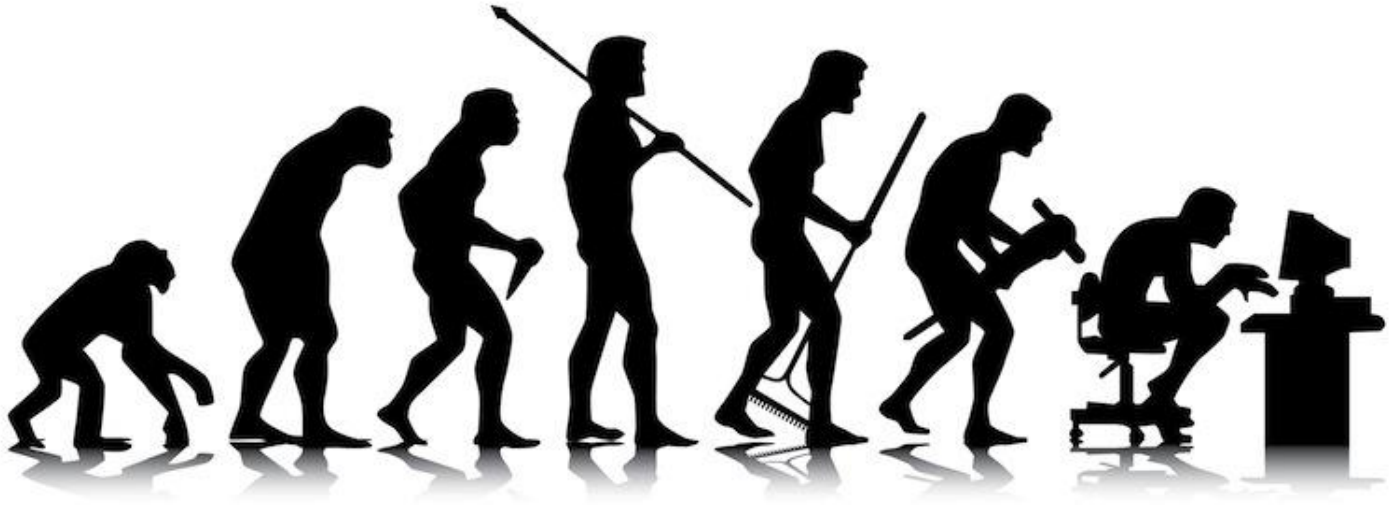
1842 - 1940

Once upon a time...





Software development evolution



Software development evolution

- First programming languages: 1940 - 1967
- Fundamental Paradigms: 1968 - 1978
- Software Gets Bigger: 1980 - 1990
- The Internet Era: 1990 - 2000
- Global Computing: 2000 - Today

Global Computing

2000 - Today

Global Computing

- Cloud Platforms - AWS / GCP
- Multi-Service Architecture (MSA)
- Reactive Programming

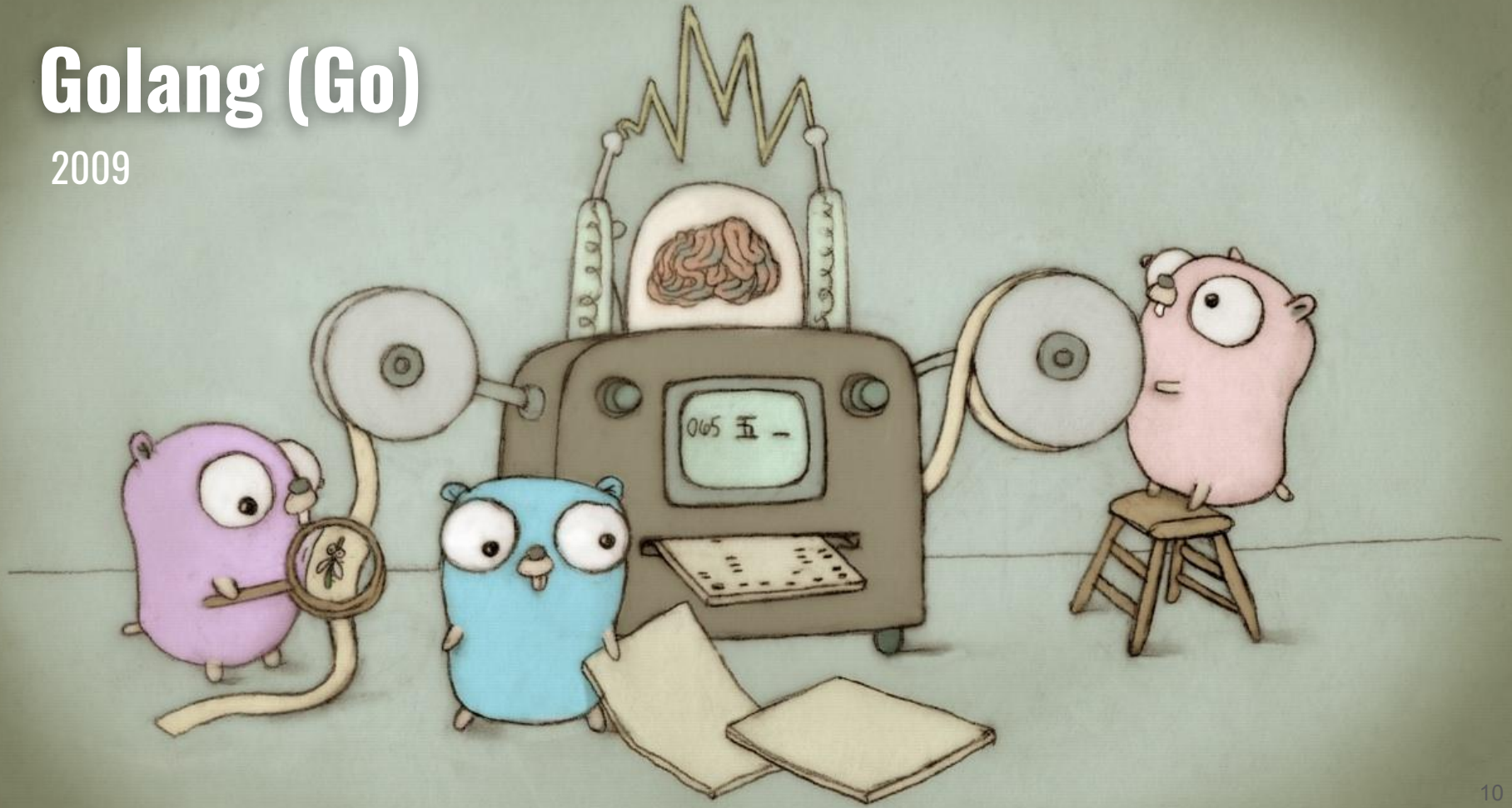
- C#
- Groovy
- Scala
- Clojure
- Go

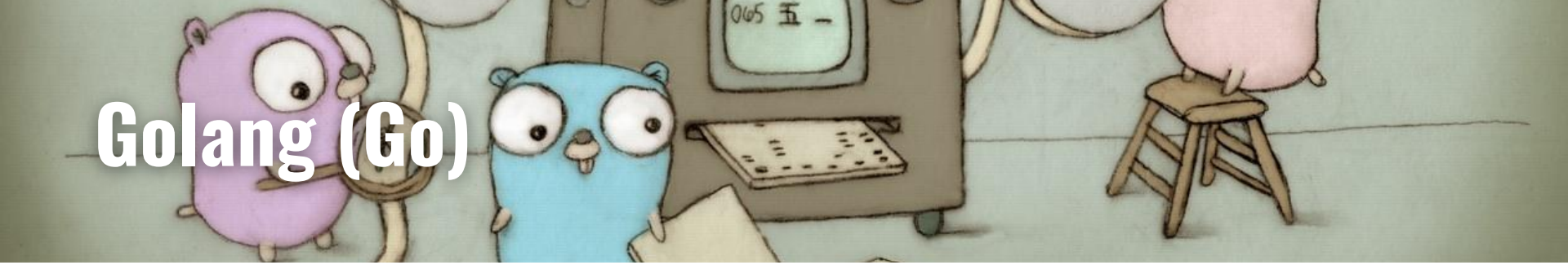
*** Quest For Simplicity ***


“The quest for simplicity
has to pervade every part of the process.
It really is fundamental.”
Jonathan Ive

Golang (Go)

2009





Golang (Go)

- Open Source, backed by Google
- Functional with Some OOP support
- Strongly Typed
- Compiled
- Scale-Ready
- Safe
- Simple! - extremely fast on-boarding

Notable Users

HashiCorp Terraform

OPENSHIFT

docker

Google

mongoDB

InfluxDB

Dropbox

kubernetes

Picking the best of all

- Simplicity
  - JavaScript
  - Python
- Performance
  - C / C++
- Parallelism
  - Akka
  - Vert.x

Go - Important resources

- Formal Documentation:
  - https://golang.org/doc/
- The Go Playground:
  - https://play.golang.org/
- Go by Examples:
  - https://gobyexample.com/

How does it Look Like?

Basic Syntax

```
package main

import "fmt"

func main() {
    var textFormat string = "Hello %s \n"
    name := "Miles David"

    fmt.Printf(textFormat, name)
}
```

Structs

```
type Person struct {
    FullName string        `json:"full_name"`
    uniqueId string        // private field
}

func main() {
    person := Person{FullName: "Miles Davis"}
    fmt.Printf("%+v", person)
    // {FullName:Miles Davis}
}
```

Functions

```
func divide(a, b *int) (int, int) {
    result := *a / *b
    residue := *a % *b
    return result, residue
}

func main() {
    a := 8
    b := 3
    fmt.Println(divide(&a, &b))   // 2 2
}
```

Interfaces

```
type (
    Rectangle struct {
        Length, Height float64
    }

    Shape interface {
        GetArea() float64
        GetType() string
    }
)
func (r *Rectangle) GetArea() float64{
    return r.Height * r.Length
}
func (r *Rectangle) GetType() string{
    return "rectangle"
}
```

```
func PrintShapeArea(shape Shape) {
    fmt.Printf("I'm a %s, area is %f",
     shape.GetType(), shape.GetArea())
}

func main() {
    rect := &Rectangle{3, 4}
    PrintShapeArea(rect)
}
```

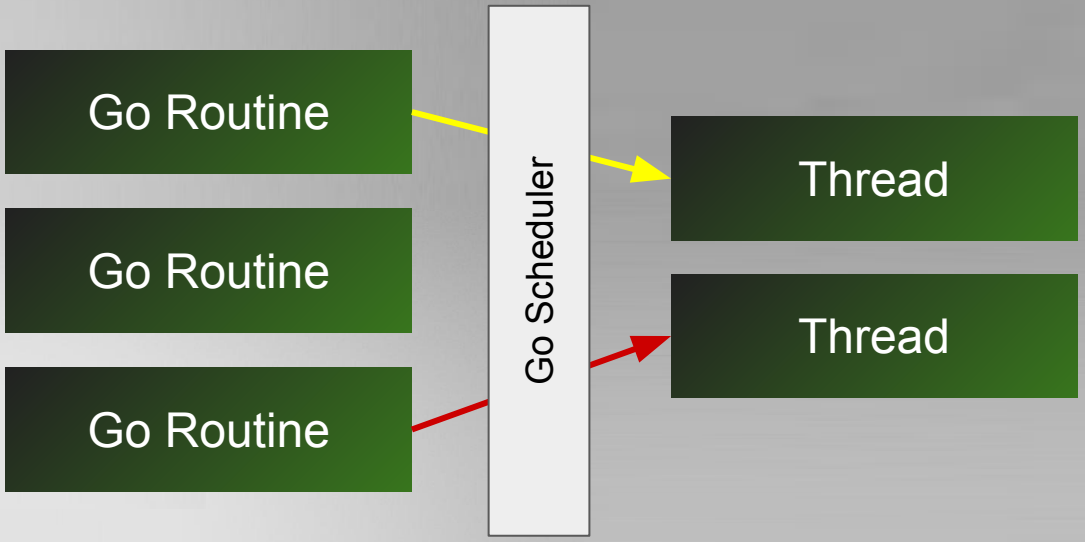
Closures

```
func TestStepper(t *testing.T){
    stepper1 := getStepper(6)
    stepper2 := getStepper(0)
    fmt.Println(stepper1())    // 6
    fmt.Println(stepper1())    // 7
    fmt.Println(stepper1())    // 8

    fmt.Println(stepper2())    // 0
    fmt.Println(stepper2())    // 1
    fmt.Println(stepper2())    // 2
}
```

```
func getStepper(init int) func() int
{
    return func() int {
        init++
        return init-1
    }
}
```

C[24]oncurrency M[8]odel

Concurrency - Go Routines



Concurrency - Go Routines

Go Routine

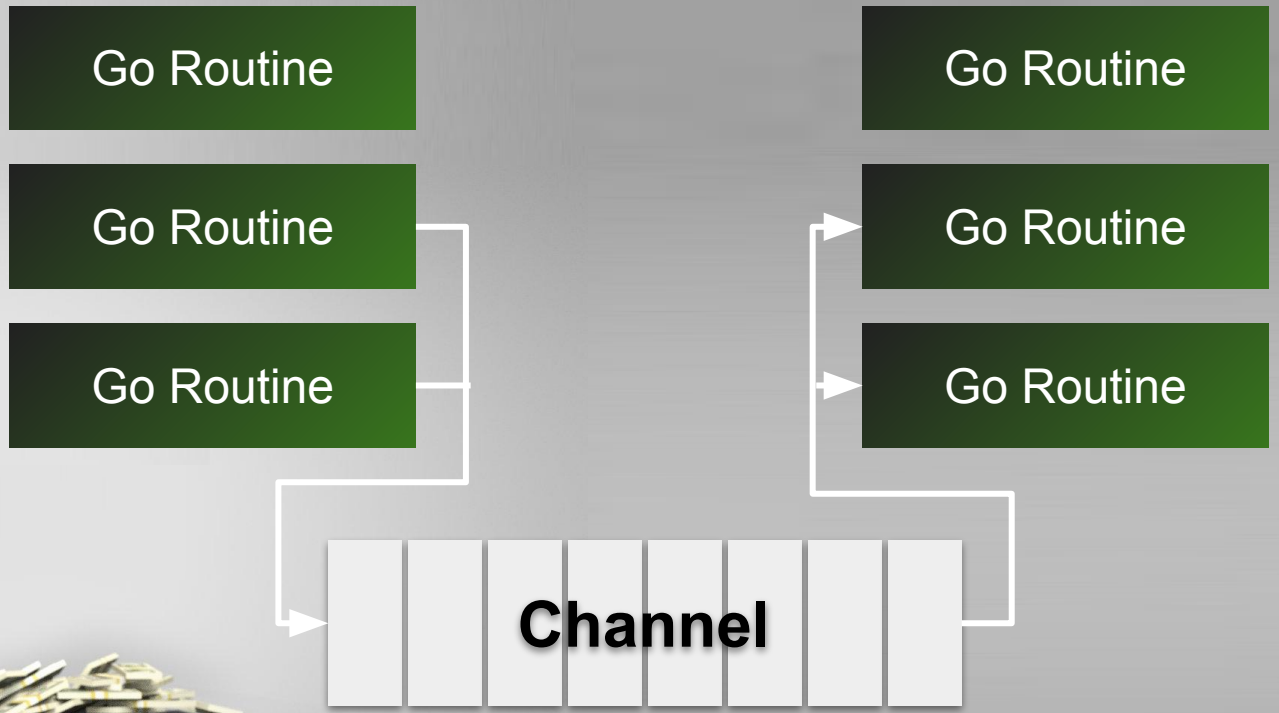
Go Routine

Go Routine

```
for i := 0; i < 10; i++ {
    go func(i int){
        fmt.Printf("%d ", i)
    }(i)
}

time.Sleep(2 * time.Second)
```

```
0 1 7 6 3 2 4 8 9 5
Process finished with exit
code 0
```

Concurrency - Channels



Concurrency - Channels

```
cn := make(chan string, 10)

for i := 0; i < 10; i++ {
    go func(i int, input chan string){
        fmt.Printf("%d: %s\n", i, <-input)
    }(i, cn)
}
for i := 0; i < 5; i++ { cn <- "Hi" }

time.Sleep(2 * time.Second)
```

```
0: Hi
4: Hi
1: Hi
2: Hi
6: Hi

Process finished
with exit code 0
```

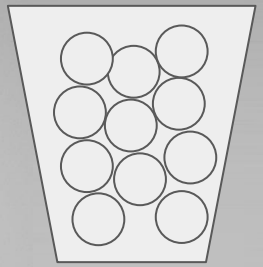
Concurrency - Waiting for task completion

```
ret := make(chan bool)

go func(done chan string){
    // do something long...
    done <- true   // success status
}(ret)

success := <- ret  // waiting...
if !success  {
    fmt.Println("task failed!")
}
```

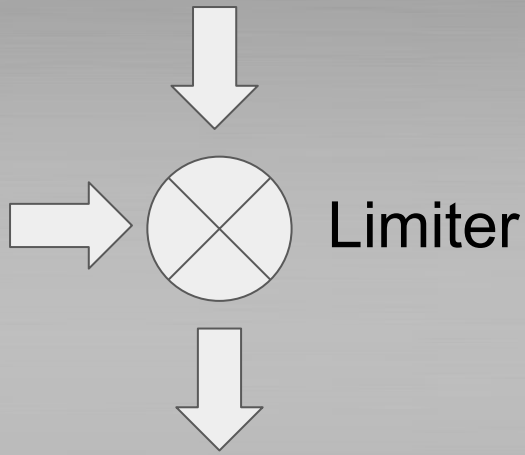
Rate limiter - Concept

Token Flow:
4 token / sec

Token Bucket:
Capacity: 11



Incoming Flow



Limiter

200 / 429

Concurrency - Channel Selection (Rate limiter)

```
output := make(chan int, 100)
input := make(chan string, 100)

go func(input chan string) {
    for _ := range input {
        select {
            case <-rateBucket:
                output <- 200
            default:
                output <- 429
        }
    }
}(input)
```

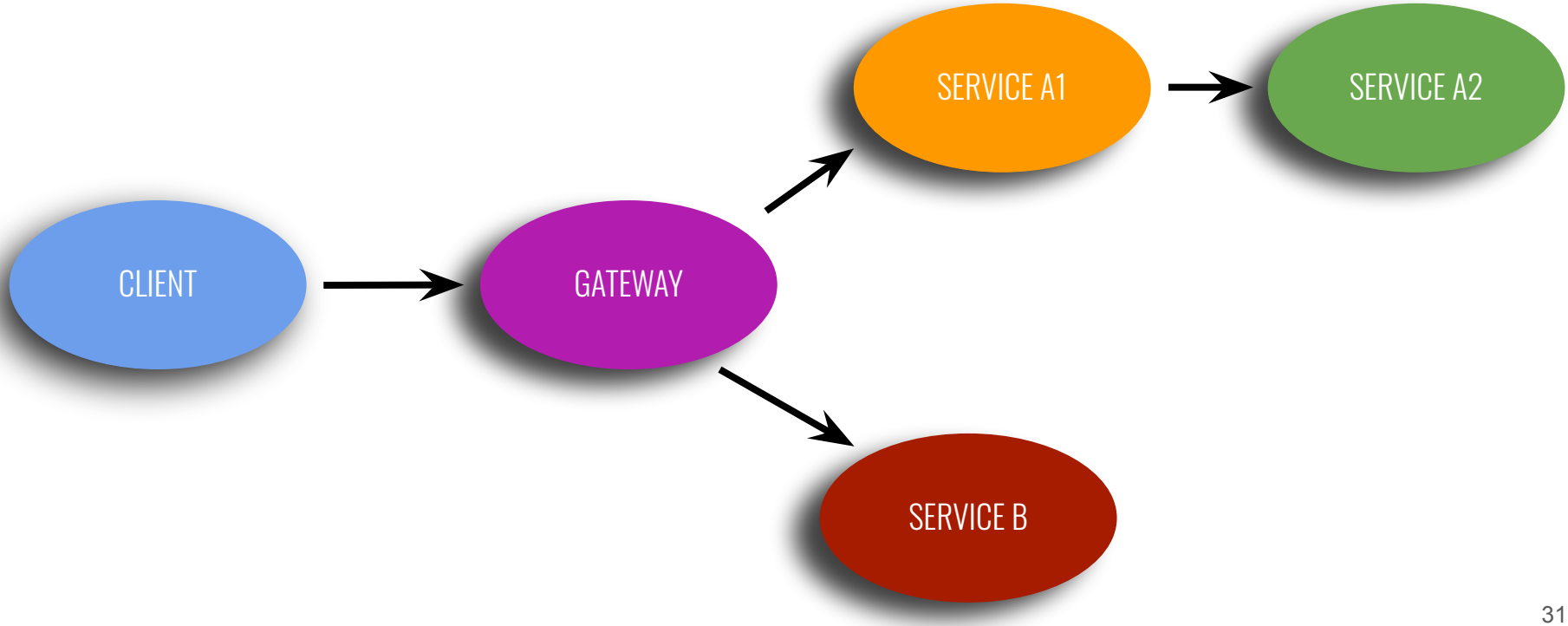
Concurrency - Token Flow (Rate Limiter)

```
var rateBucket= make(chan time.Time, 11)
for i := 0; i < 11; i++ {
    rateBucket <- time.Now()
}

rateTick := time.Tick(250 * time.Millisecond)
go func() {
    for {
        rateBucket <- <-rateTick
    }
}()
```

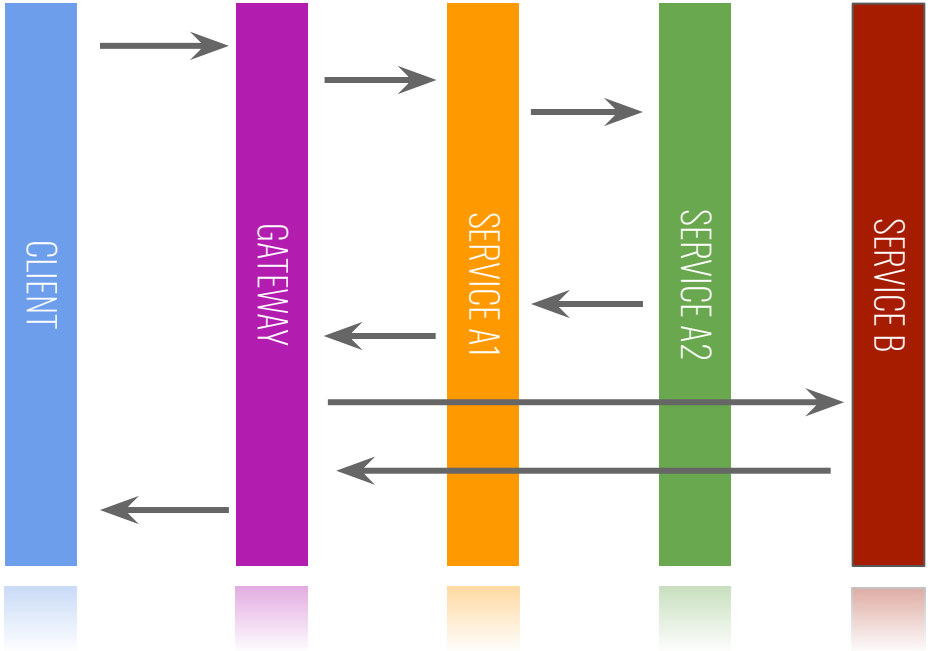
Case Study

Parallelism in the MSA world

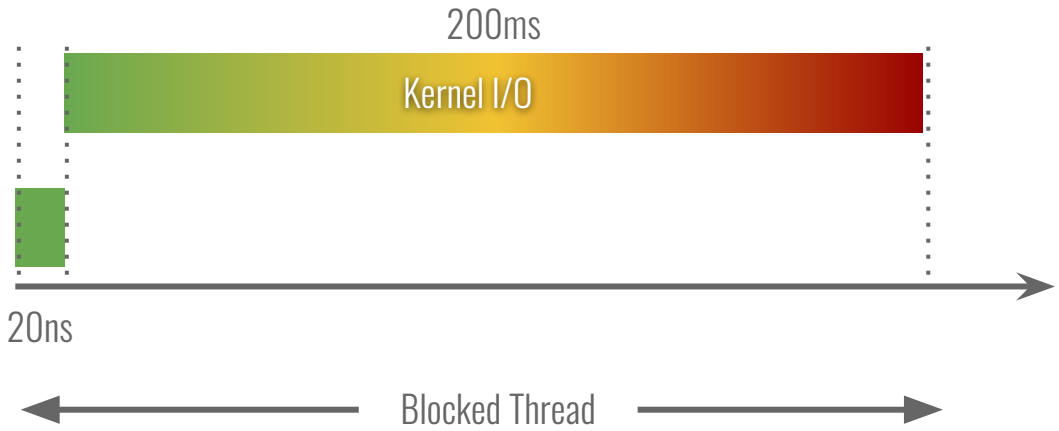
System Overview



Flow

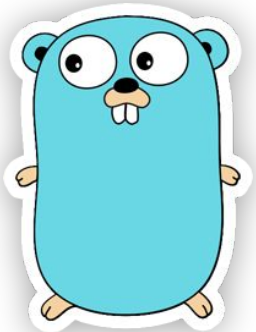


Request Timeline

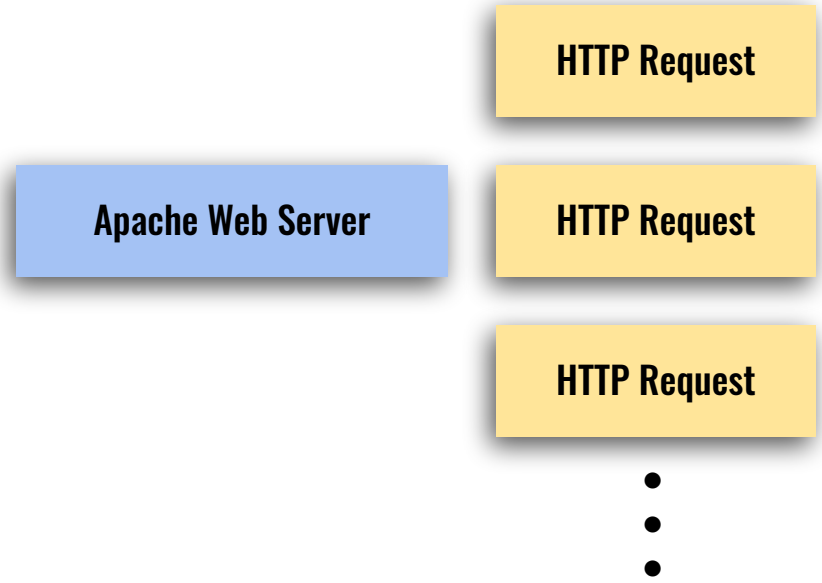


x 10,000,000!

Concurrency Models

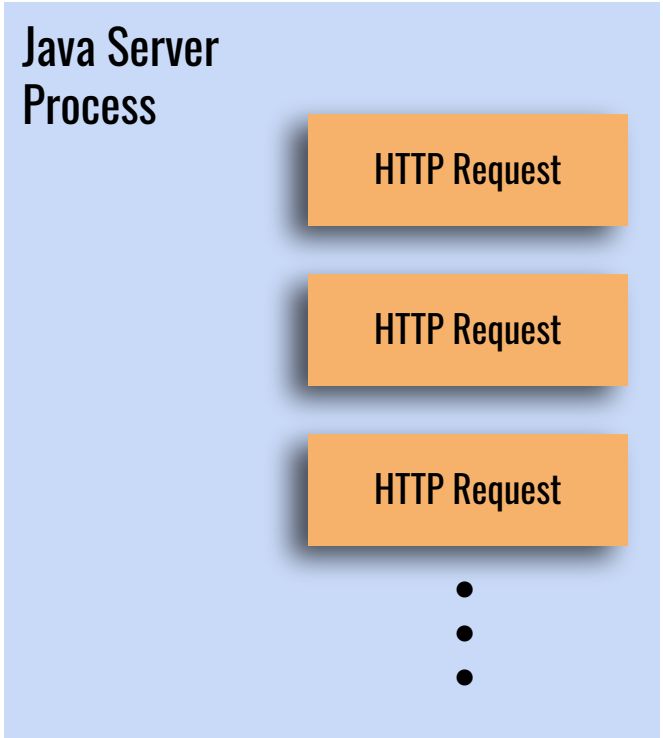


Multi Process Model



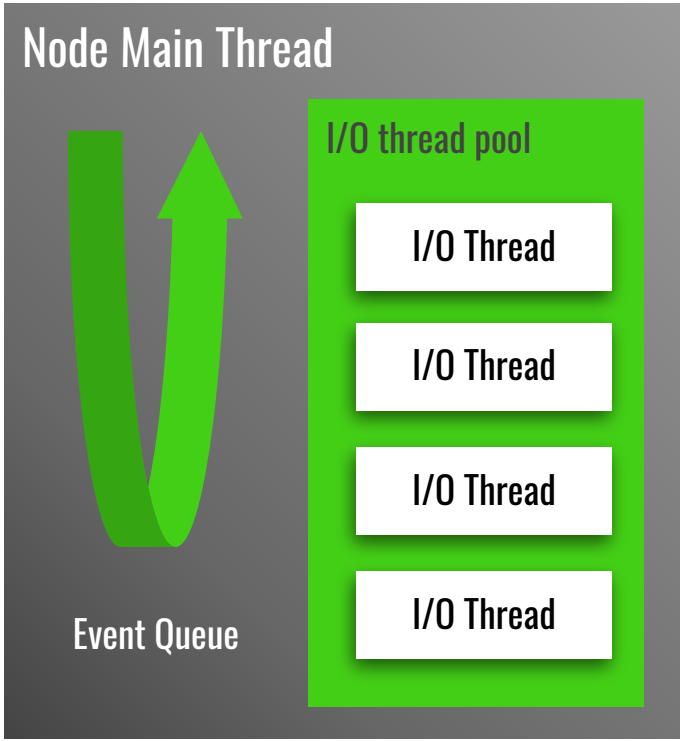

```php
<?php

    // blocking file I/O
    $file_data = file_get_contents('/path/to/file.dat');

    // blocking network I/O
    $curl = curl_init('http://example.com/example-microservice');
    $result = curl_exec($curl);

    // some more blocking network I/O
    $result = $db->query('SELECT id, data FROM examples
                          ORDER BY id DESC limit 100');

?>
```

Multi Thread Model



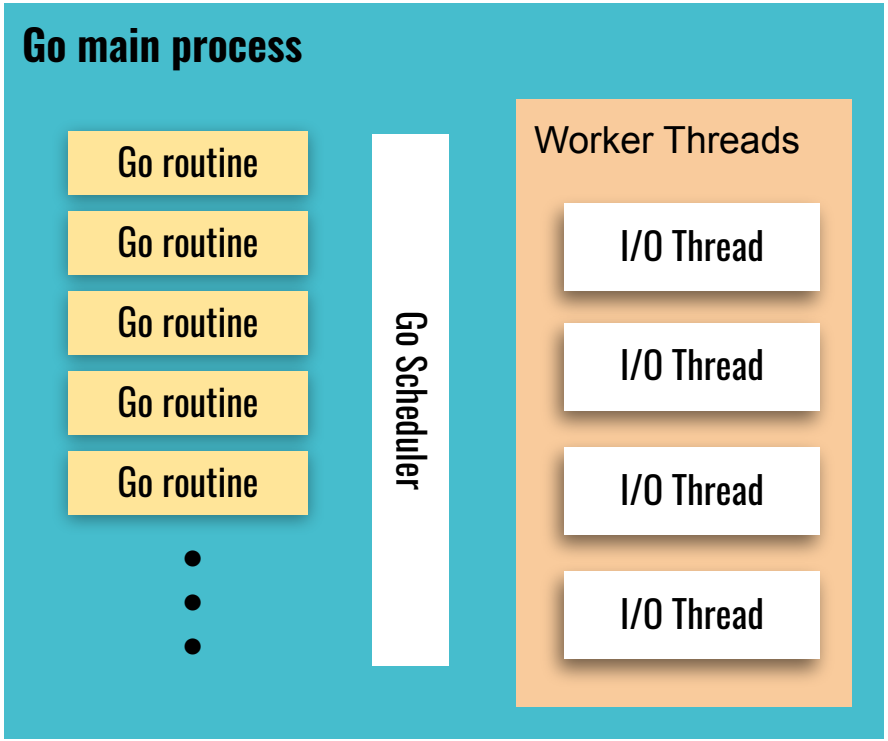

```
public void doGet
      (HttpServletRequest request, HttpServletResponse response) {

      // blocking file I/O
      InputStream fileIs = new FileInputStream("/path/to/file");

      // blocking network I/O
      URLConnection urlConnection =
         (new URL("http://example.com/microservice"))
         .openConnection();

      InputStream netIs = urlConnection.getInputStream();

      // some more blocking network I/O
      out.println("...");
}
```

Async I/O Model




```
var handler = function(request, response) {

        connection.query('SELECT ...', function (err, rows)
        {
                if (err) { throw err };

                for (var i = 0; i < rows.length; i++) {
                        // do processing on each row
                }

                response.end(...); // write out the result
        })
};
```

Goroutines Model



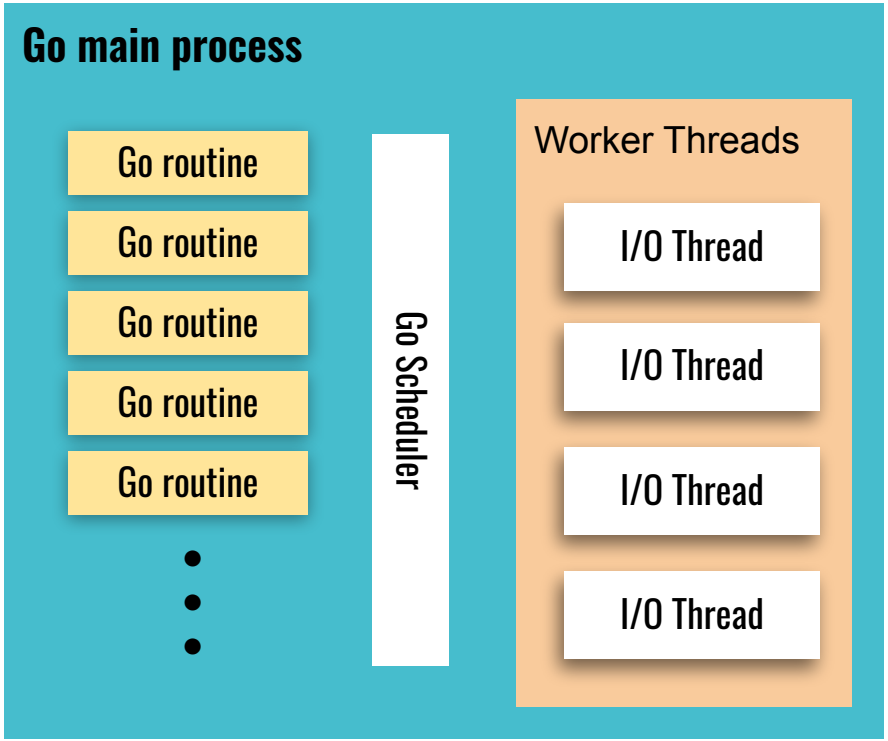

```
func ServeHTTP
(w http.ResponseWriter, r *http.Request) {

        // the underlying network call here is async
        rows, err := db.Query("SELECT ...")

        for _, row := range rows {
                // do something with the rows,
                // each request in its own goroutine
        }

        w.Write(...) // write the response, also async
}
```

Benchmarks

Basic Setup

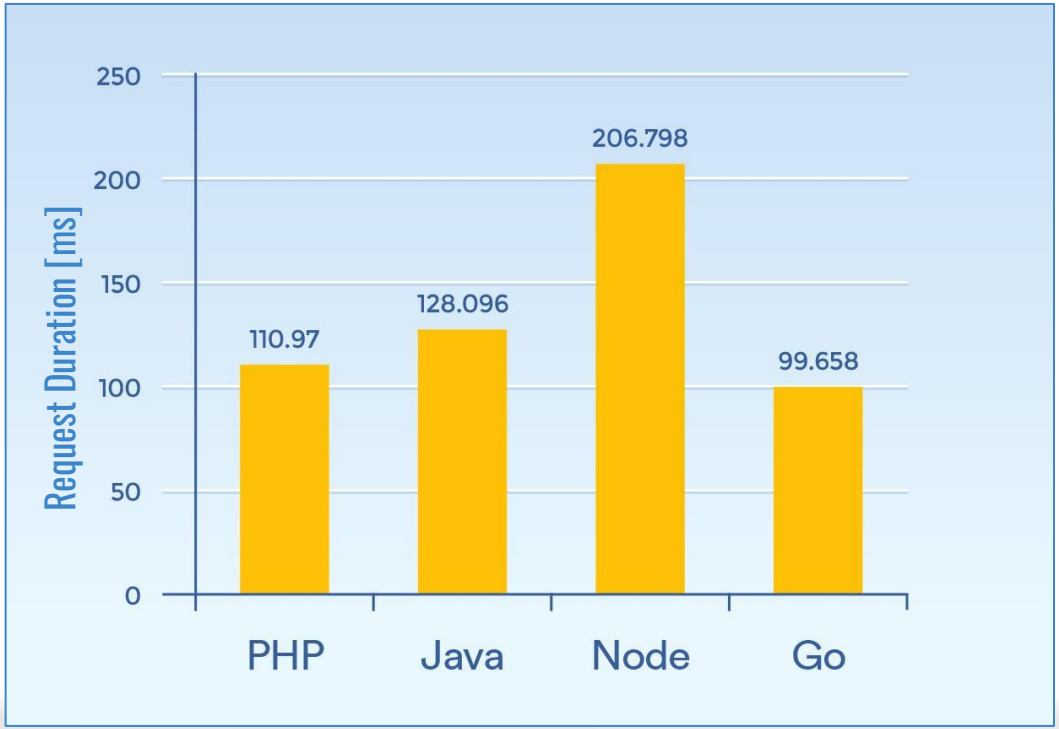
N=1
300 RPS

Goroutines Model




```
func ServeHTTP
(w http.ResponseWriter, r *http.Request) {

        // the underlying network call here is async
        rows, err := db.Query("SELECT ...")

        for _, row := range rows {
                // do something with the rows,
                // each request in its own goroutine
        }

        w.Write(...) // write the response, also async
}
```

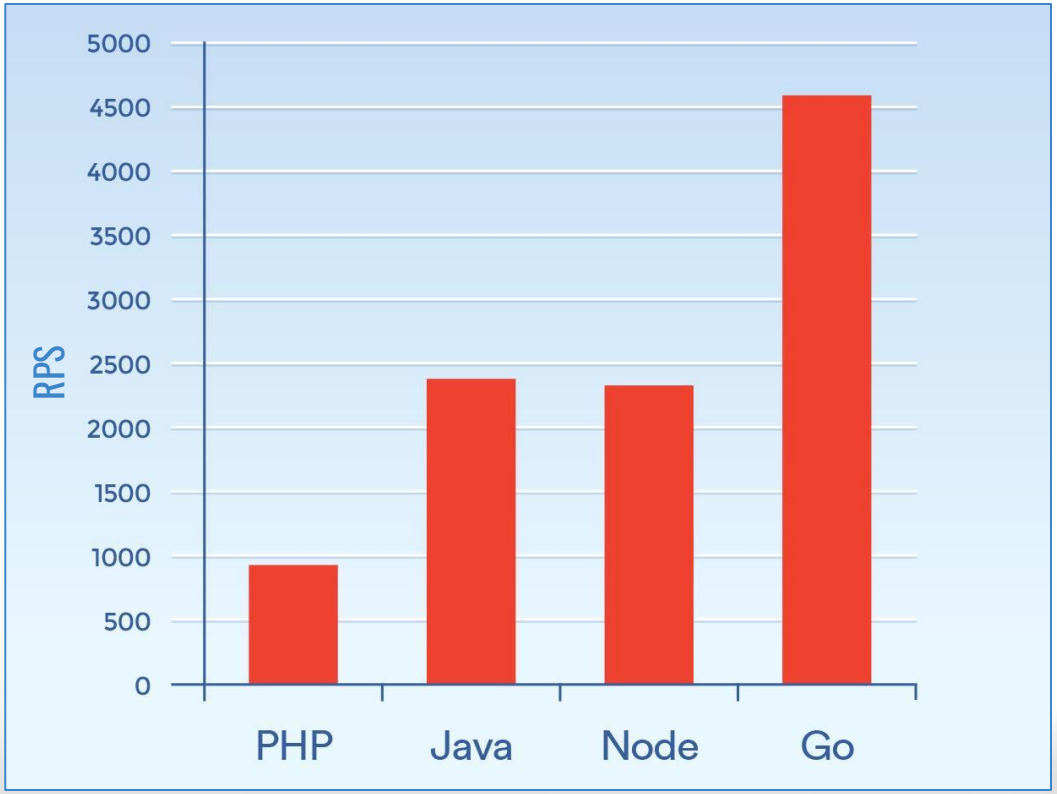
CPU Intensive Setup



N=1000
300 RPS

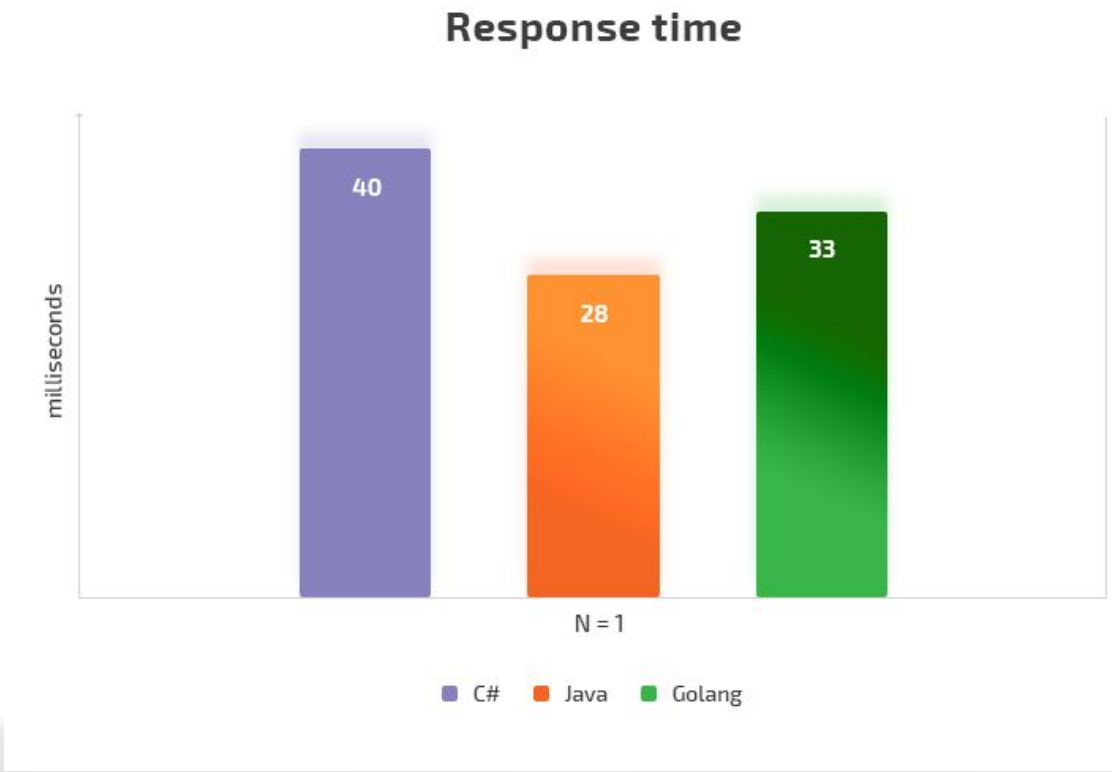
I/O Intensive Setup

N=1
5000 RPS

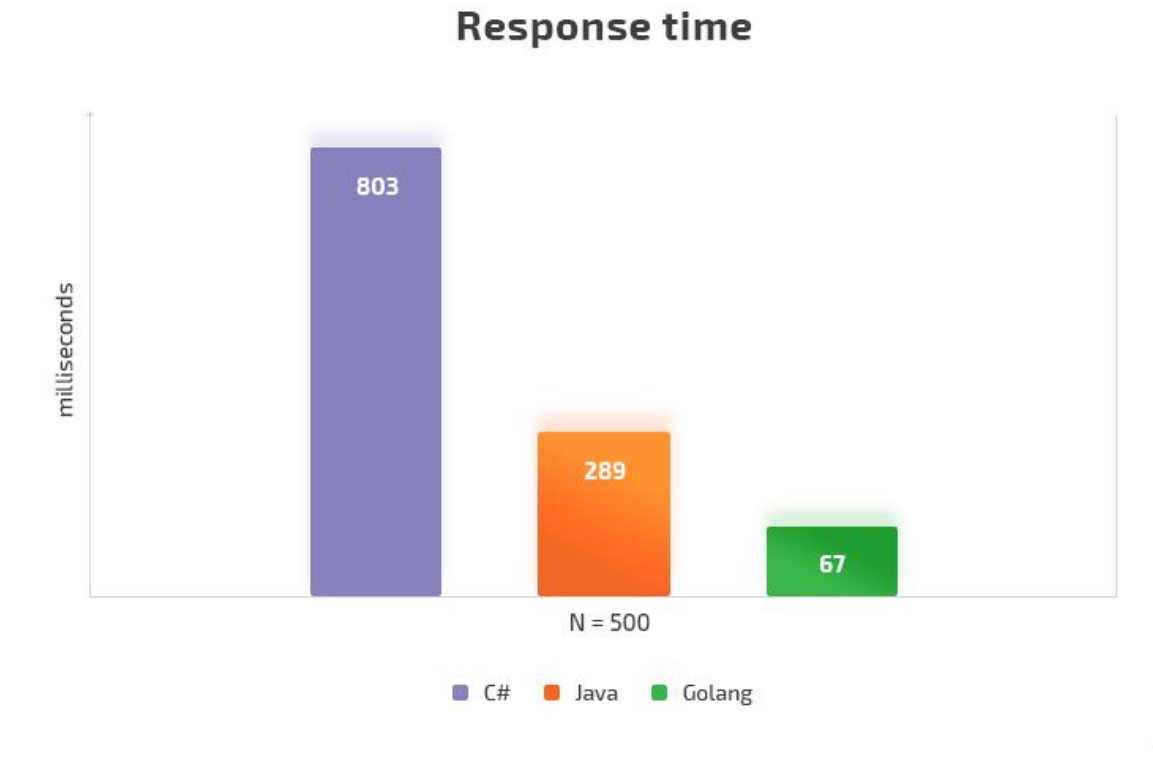


What About C#?

Introducing C# (single request)



Introducing C# (500 requests)



The downsides...

- Limited OOP
- No generics<T>
- Platform-specific compilation
- Explicit error checks
- No function overloading
- Yet, smaller ecosystem

When Golang might be a good choice?

- Philosophy of freshness and simplicity
- Performance of a compiled binary
- “Write now, scale later...”
- Focus in Business Logic
- Fast on-boarding and collaboration

Some other cool stuff

- GoDoc
- Testing, Benchmarking, Profiling
- Race Detector
- Learning Curve
- Reflection
- Opinionatedness
- Culture

“Most of the appeal for me is not the features that Go has, but rather the features that have ***been intentionally*** *left out.”*

Txxxxd @ Hacker News

“Go is not meant to innovate programming theory.
*It’s meant to innovate programming **practice**.”*

Samuel Tesla

Go is all about:

“There should be only

One Way

Of doing a thing”

What About Us?

Questions?