Discover GO

A Powerful Modern Programming Language

Presented by
Hugo Y. Silva

Hugo Y. Silva

SOFTWARE ENGINEER

- 9+ years of experience
- Specialized in backend development
- Go, PHP, Java, Kotlin, JavaScript
- Passionate about software engineering

AGENDA

1. Introduction to Golang
2. History and Motivations Behind Go
3. Go vs. Other Programming Languages
4. Market Adoption and Use Cases
5. Live Coding Demo: Go Routines and Structs
6. Q&A Session

What is Go?

OPEN SOURCE

Open-source language developed and supported by Google

STATICALLY TYPED

Strongly typed but can be dynamically declared

COMPILED

Powerful and fast compiler, able to compile a large go program within few seconds

CROSS PLATFORM

Supports many different OS and architecture types

Reasons to create a new language

Google was in a period of tremendous growth

1

A lot more people writing a lot more code

2

Struggling with massive and complex codebase

3

Java and C++ led to brutal code, diminishing productivity

4

Problems about scalability

What do we need in a modern language?

HIGH PERFORMANCE

Optimized for networking and multicore processing

HIGH PRODUCTIVITY

Simple and easy to use and understand

MAINTAINABILITY

Applications easier to maintain, operate, and scale

History of Go

Rob Pike, Robert Griesemer and Ken Thompson

2007
Project started inside Google

2009
Publicly announced as an open-source project

2012
Stable 1.0 Version release

2015
1.5 Version - Refinement

2018
1.11 Version - Improvements on the ecosystem

2022
1.18 Version - Generics support

2024
1.23 Version - Telemetry tool

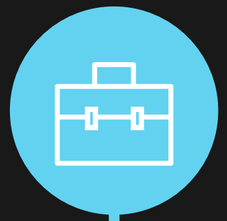
Main Features

CONCURRENCY

Easy ways to handle concurrency

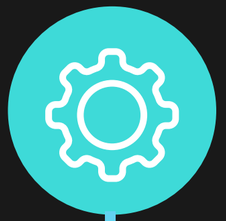
TESTING

Built-in test framework



PLATFORM INDEPENDENT

Code can be compiled to almost any platform



SIMPLICITY

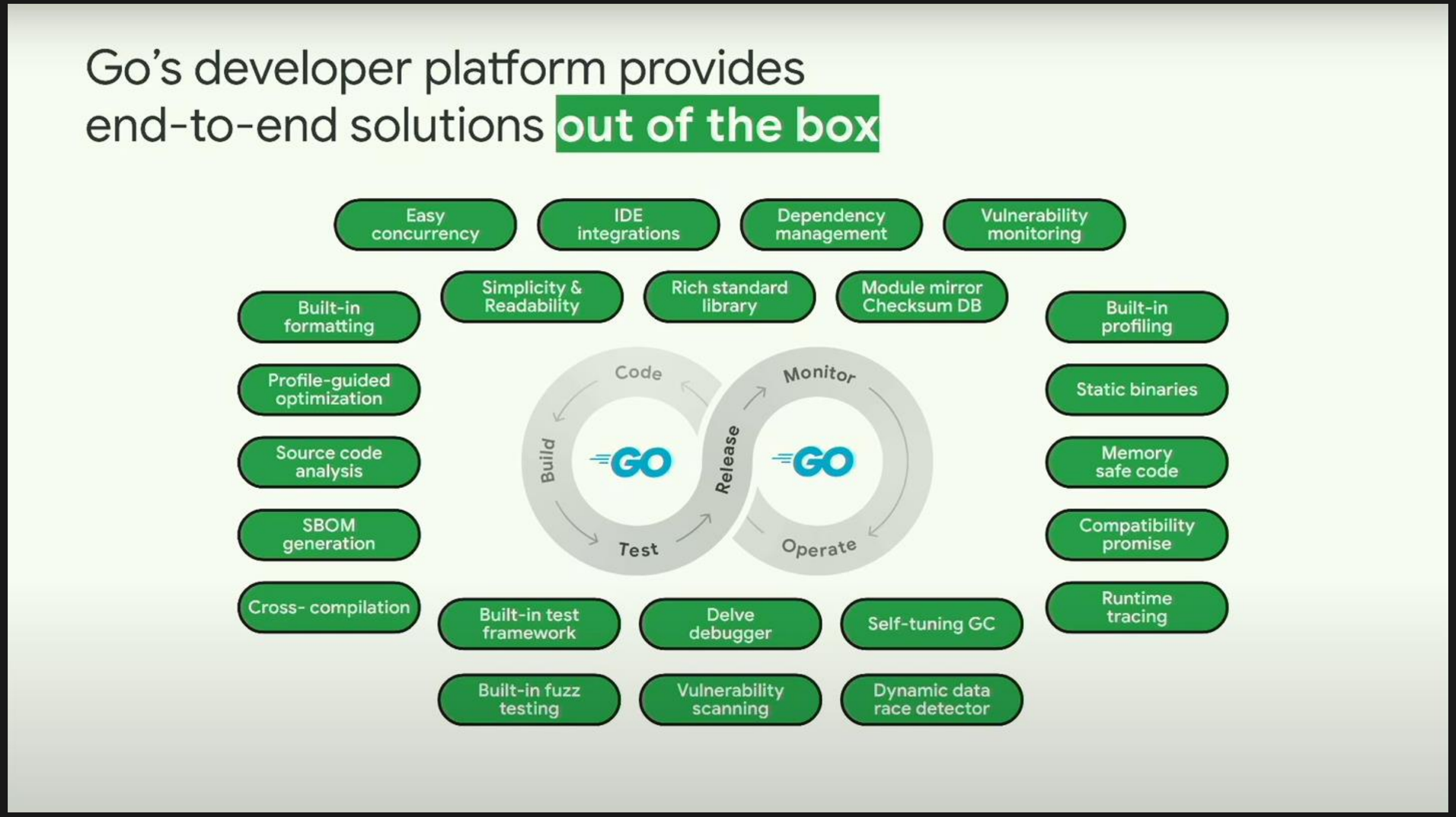
Focused on simplicity and productivity

COMPLETE PACKAGE

Complete and robust standard library

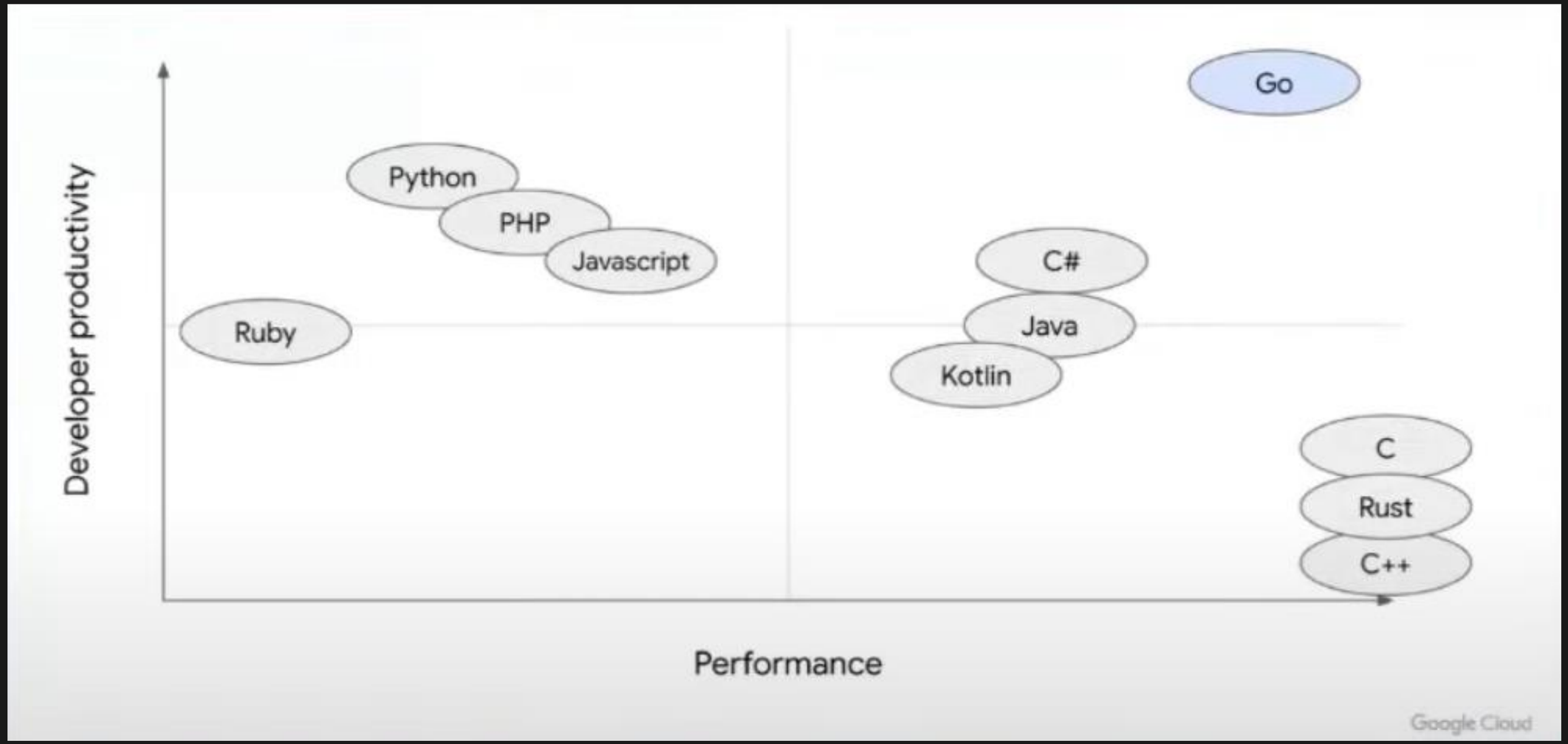
Go is not just a programming language

But a complete develpoment platform



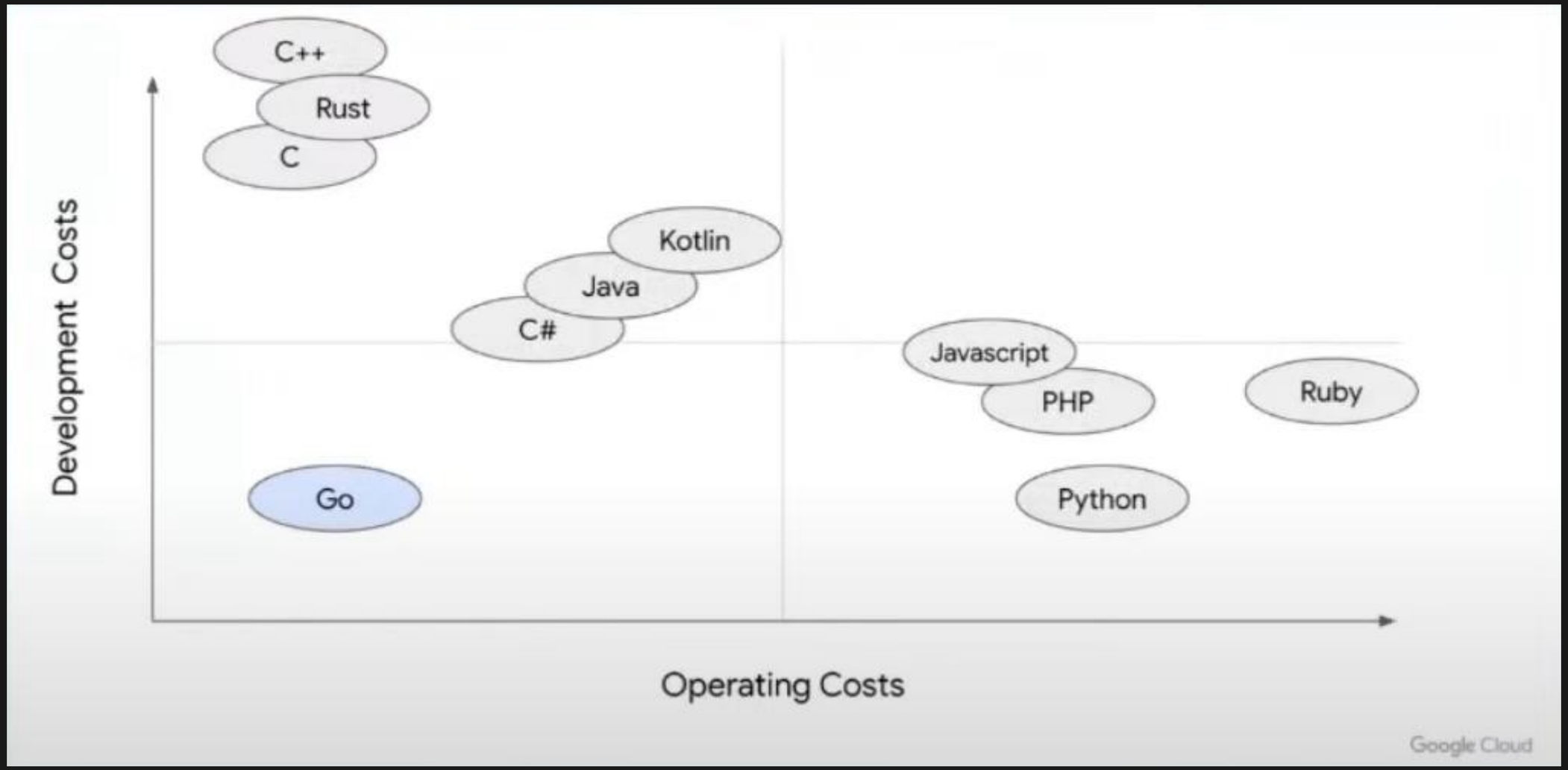
Source: Gophercon Europe 2023 - Keynote: The state of go

Developer productivity X Performance



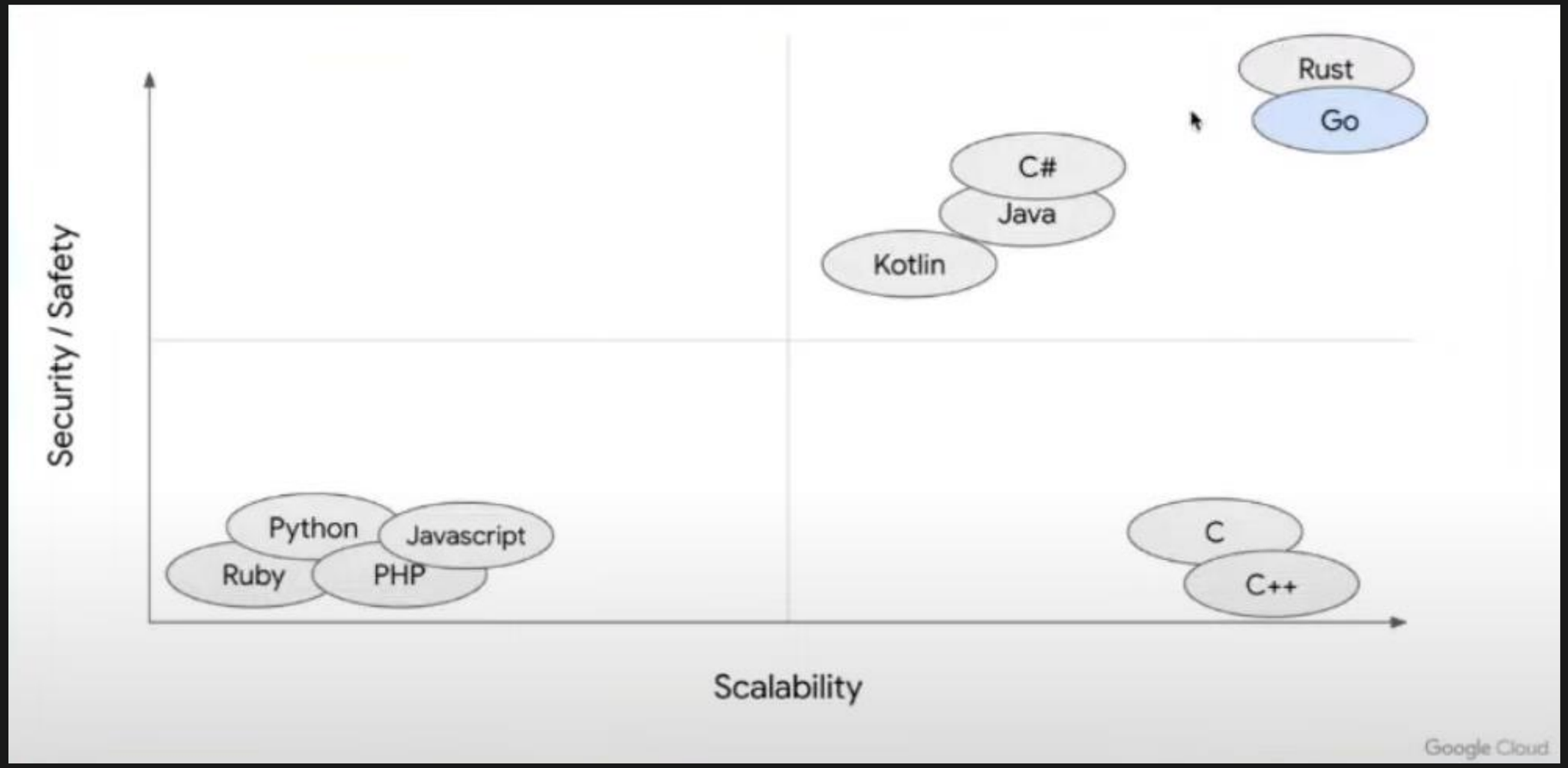
Source: https://www.youtube.com/watch?v=u56Yp3Egzao

Developement Costs X Operating costs



Source: https://www.youtube.com/watch?v=u56Yp3Egzao

Safety / Security X Scalability



Source: https://www.youtube.com/watch?v=u56Yp3Egzao

Go Platform Adoption

Source: [Gophercon Europe 2023 - Keynote: The state of go]

Go Platform Adoption

Source: [Gophercon Europe 2023 - Keynote: The state of go]

Uber

Uber’s GPU-powered Analytics Engine

01.

AresDB: Written in Go, powers Uber's real-time data analytics dashboards.

02.

Enables data-driven decisions at scale across various business operations.

03.

Chosen for its ability to efficiently handle real-time analytics with large datasets.

Source: https://go.dev/solutions/case-studies

NETFLIX

Netflix’s Application Data Caching

01.

Utilizes Go for application data caching using SSDs.

02.

Why Go? Lower latency than Java and more productivity than C.

03.

Handles tens of thousands of client connections, so Go fits this space very well.

Source: https://go.dev/solutions/case-studies

Cloudflare’s Use of Go for High-latency HTTP Connections

01.

Go is at the heart of essential services: HTTP compression, DNS infrastructure, SSL, and load testing.

02.

Chosen for its ability to handle compression for high-latency HTTP connections

03.

Go is integral to Cloudflare's mission to protect and speed up millions of online properties.

Source: https://go.dev/solutions/case-studies

Riot Games’ Backend Microservices with Go

01.

Go is leveraged to develop and operate backend microservices at a global scale.

02.

Used for game development and operational tasks to ensure scalability and reliability.

03.

Riot shares insights into Go's value for efficient operations and community support.

Source: https://go.dev/solutions/case-studies

ipead FACE • UFMG 75 ANOS

Migrated PHP spreadsheet data ingestion

01.

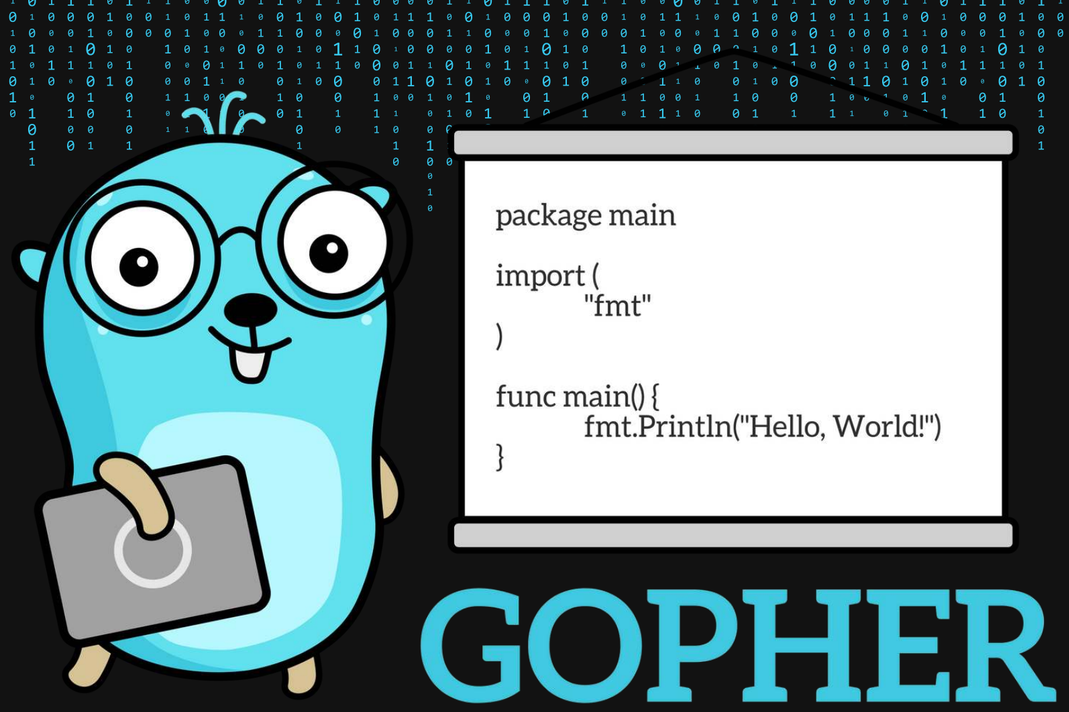
Synchronous service that imports tens of thousands of medical data from government

02.

The process consists in reading a large dataset from excel spreadsheet and insert into a cloud database

03.

The whole process was improved using golang and parallelism reducing time **from 1 hour to just 8 minutes (7.5x faster)**



GOPHER

LETS DIVE INTO THE CODE

Structs

main.go

```go
package main

import "fmt"

type Movie struct {
    Name   string
    Rating int
}

func main() {
    m := Movie{
        Name:   "Deadpool",
        Rating: 10,
    }
    fmt.Println(m)
}

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
~/projects/vanlug-gopresentation > go run main.go
{Deadpool 10}
```

Pointers

main.go > updateStringPointer

```go
func main() {
    var movie string
    movie = "Deadpool"

    fmt.Println(movie)

    updateString(movie)
    fmt.Println(movie)

    updateStringPointer(&movie)
    fmt.Println(movie)
}

func updateString(str string) {
    str = fmt.Sprintf("This won't be updated: %s", str)
}

func updateStringPointer(str *string) {
    *str = fmt.Sprintf("Your movie is: %s", *str)
}
```

PROBLEMS 50 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
~/projects/vanlug-gopresentation ❯ go run main.go
Deadpool
Deadpool
Your movie is: Deadpool
```

Webserver

main.go

3 webserver > main.go > main

```go
package main

import "net/http"

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

func HelloVanlug(w http.ResponseWriter, r *http.Request) {
    hello := []byte("<h1>Hello VanLUG!</h1>")
    w.Write(hello)
}

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS 1 COMMENTS

```
~/projects/vanlug-gopresentation > curl http://localhost:8080
<h1>Hello VanLUG!</h1>%
```

Routines

main.go

~/projects/vanlug-gopresentation/4 routines/main.go

```go
func main() {
    /*
        Whenever we want to run a function asynchronously (in a separated lightweight process)
        we just add the reserved word "go" in front of it
        and Voilá
    */
    go counter(10)
    go counter(10)
    counter(10)
}

func counter(n int) {
    for i := range n {
        fmt.Println(i)
        time.Sleep(time.Second)
    }
}
```

Channels

main.go

5 channels > main.go > main

```go
package main

import "fmt"

type Movie struct {
    Name   string
    Rating int
}

func (m *Movie) Format() string {
    return fmt.Sprintf("%s rating is %v", m.Name, m.Rating)
}

func main() { //Main process - T1
    /*
        A channel has a type of your choice, where you are can send / receive data
        between go routines (separated processes)
    */
    channel := make(chan Movie)

    go func() { // Another process - T2
        //Sending data
        channel <- Movie{Name: "Deadpool", Rating: 9}
    }()

    //T1
    // Reading data
    m := <-channel
    fmt.Println(m.Format())
}
```

Simple load balancer simulation

main.go

6 load balancer > main.go > main

```go
package main

import (
    "fmt"
    "time"
)

func worker(workerId string, data chan int) {
    // This will loop the channel until this channel is closed
    // otherwise it will keep reading it
    for x := range data {
        fmt.Printf("WorkerID: %s | Value: %v\n", workerId, x)
        time.Sleep(time.Second)
    }
}

func main() {
    channel := make(chan int) //channel := make(chan int, 10) - Example of a channel with a "buffer" size of 10

    // go worker("John", channel)
    // go worker("Hugo", channel)
    // go worker("Rachel", channel)

    // Start amount of workers that will listen the channel
    // whoever is available is going to read the channel first
    workers := 10000
    for i := range workers {
        go worker(GenWorkerName(i), channel)
    }

    // Simulating a large amount of data being pushed to the channel
    dataset := 100000
    for i := range dataset {
        /*
            Only one single value can be pushed to channel at a time.
            Execution will blocked until someone read from channel
            It is possible to create a channel with a "buffer" of size X
        */
        // sending data
        channel <- i
    }
}

func GenWorkerName(x int) string {
    return fmt.Sprint("John", x)
}
```

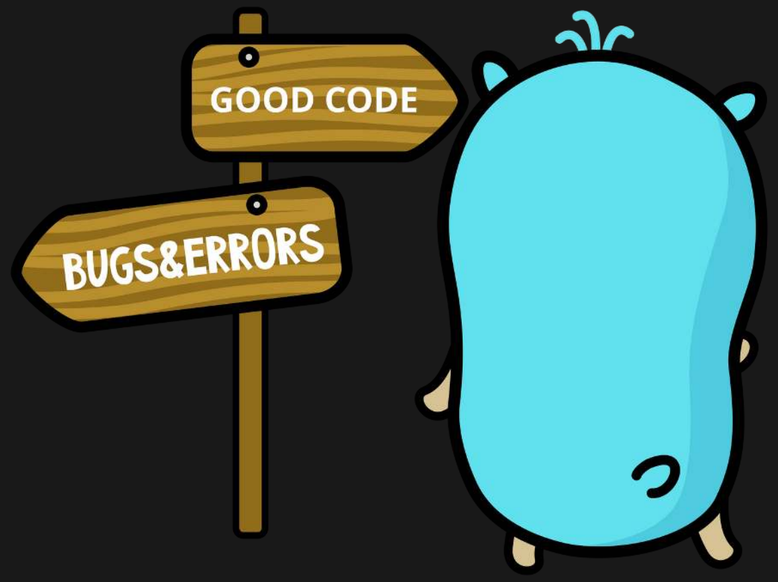
WRAPPING UP

01 Go is a simple, efficient, and powerful language.

02 It Is widely adopted by major tech companies for scalable and performant applications.

03 Invite for you to explore Go through official documentation, tutorials, and community resources.

Learn more about GO



Learn Go on official documentation

Online coding tour

Go online Playground

Golang Course with Bonus Projects - FreeCodeCamp

ANY QUESTIONS?

Hugo Yamauthi Silva

Senior Software Engineer // Meeting Business Goals for Software Products by ...

THANK YOU