



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?



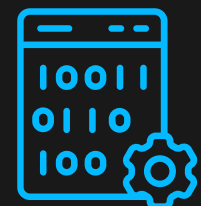
OPEN SOURCE

Open-source language developed and supported by Google



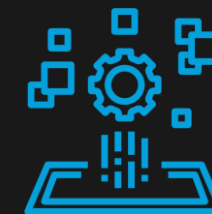
STATICALLY TYPED

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

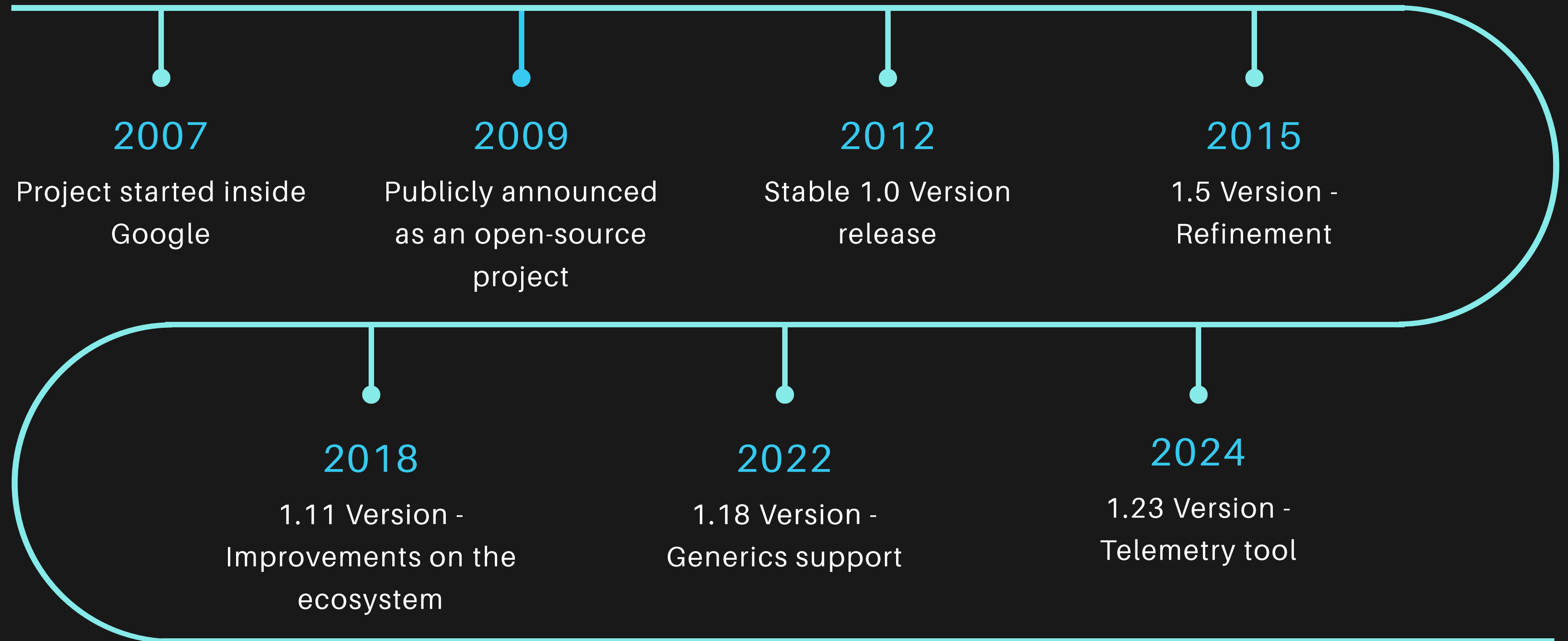


MAINTAINABILITY

Applications easier to maintain, operate, and scale

History of Go

Rob Pike, Robert Griesemer and Ken Thompson



Main Features



CONCURRENCY

Easy ways to handle concurrency



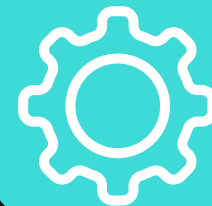
TESTING

Built-in test framework



PLATFORM INDEPENDENT

Code can be compiled to almost any platform



SIMPLICITY

Focused on simplicity and productivity



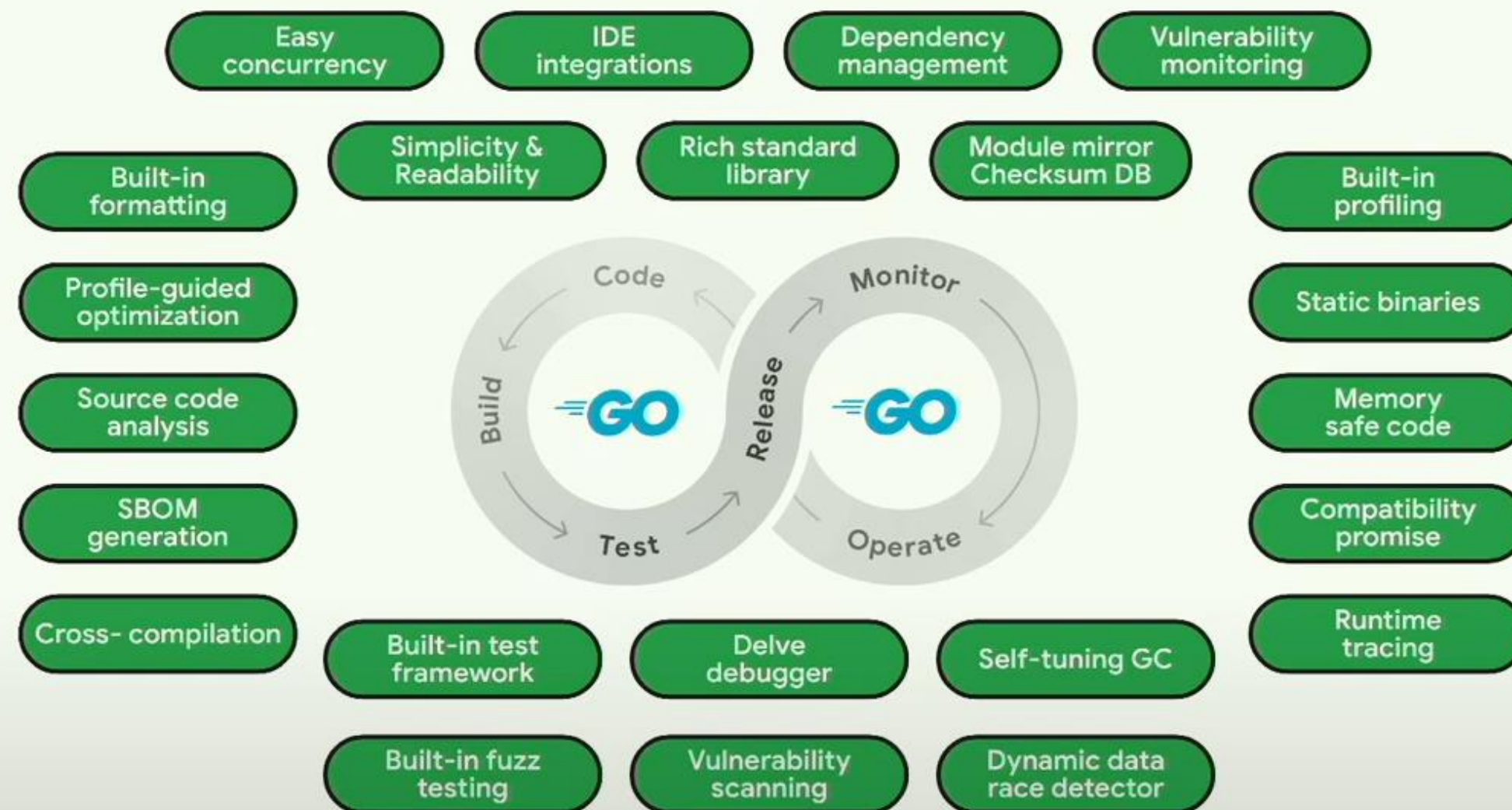
COMPLETE PACKAGE

Complete and robust standard library

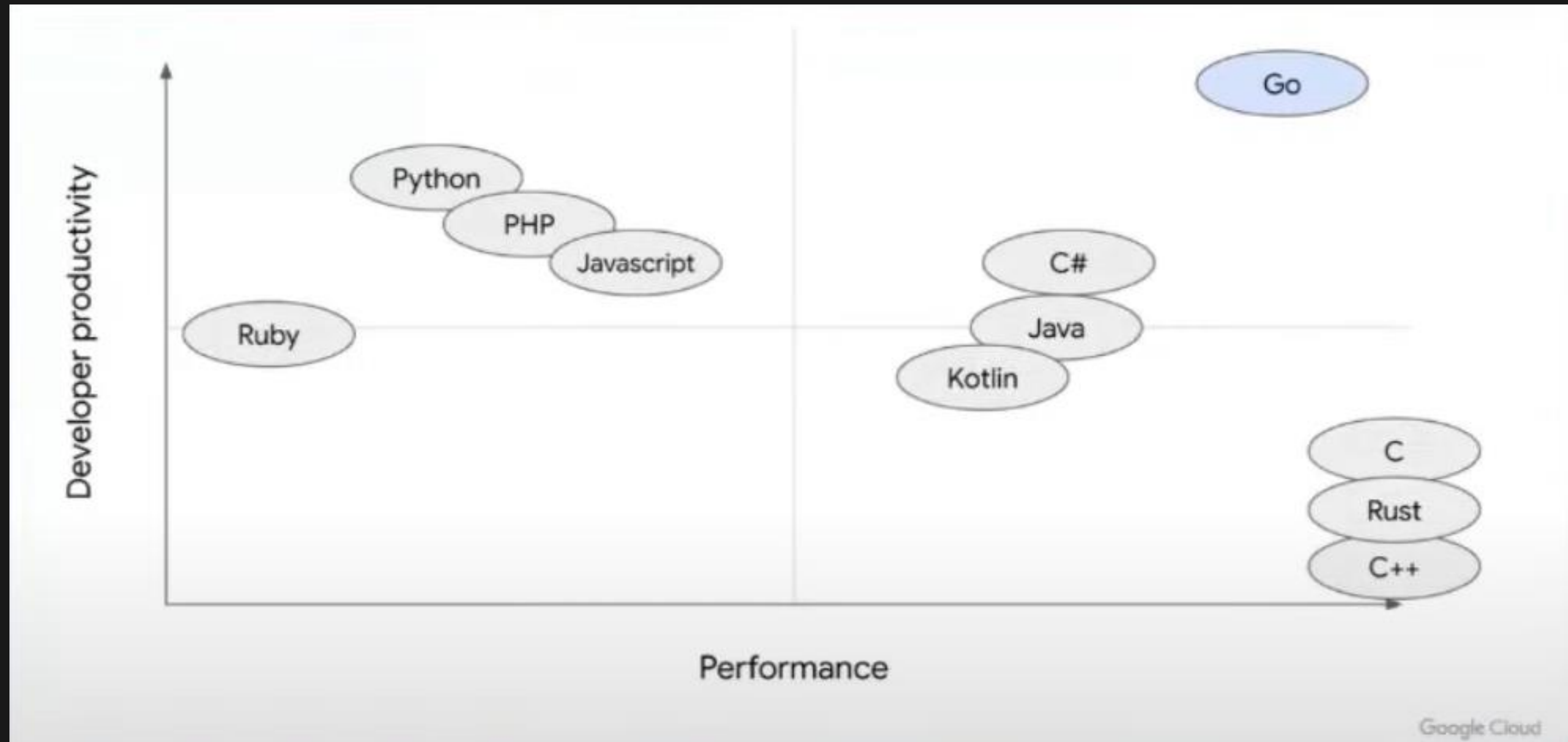
Go is not just a programming language

But a complete development platform

Go's developer platform provides
end-to-end solutions **out of the box**

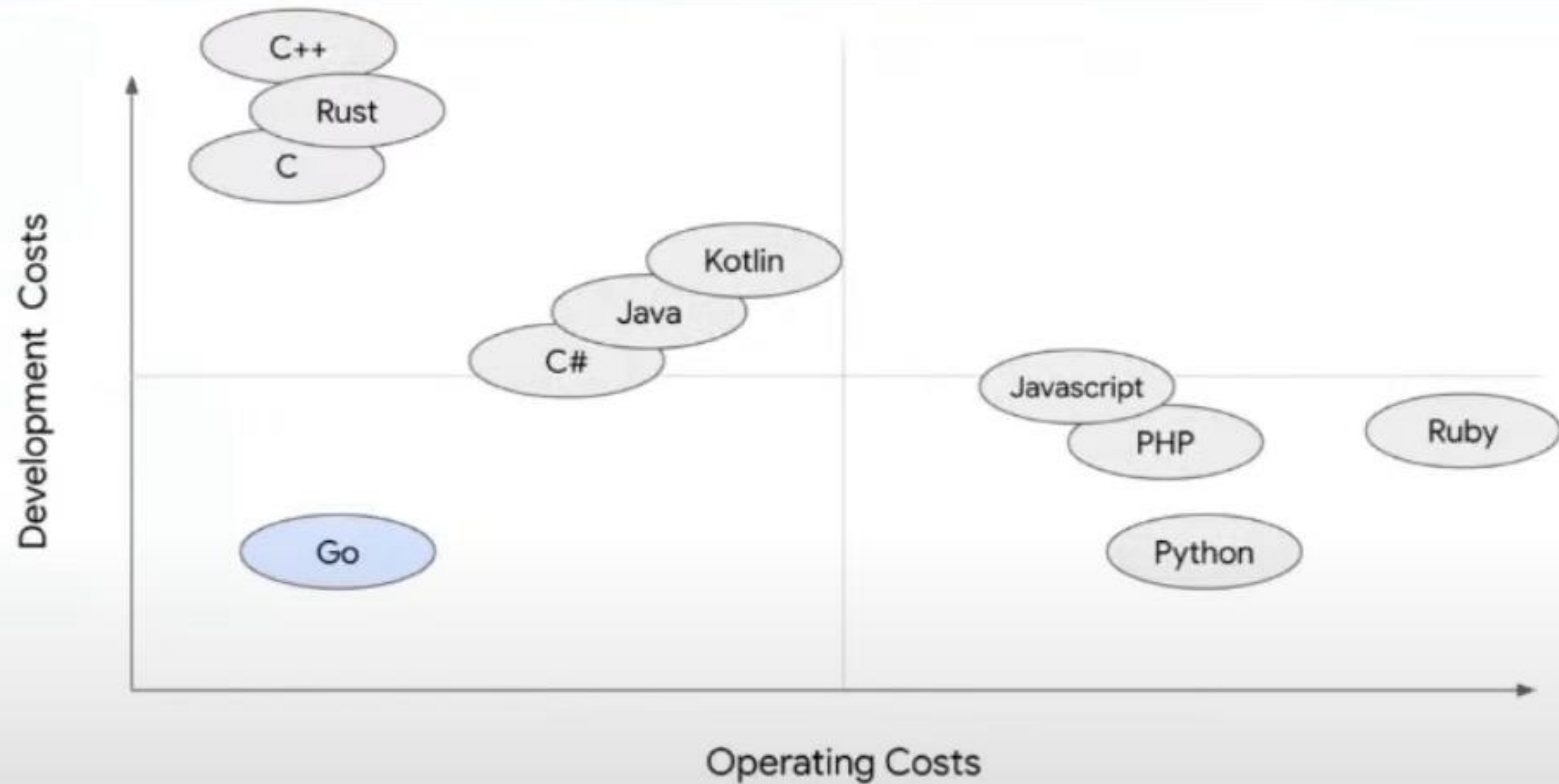


Developer productivity X Performance



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

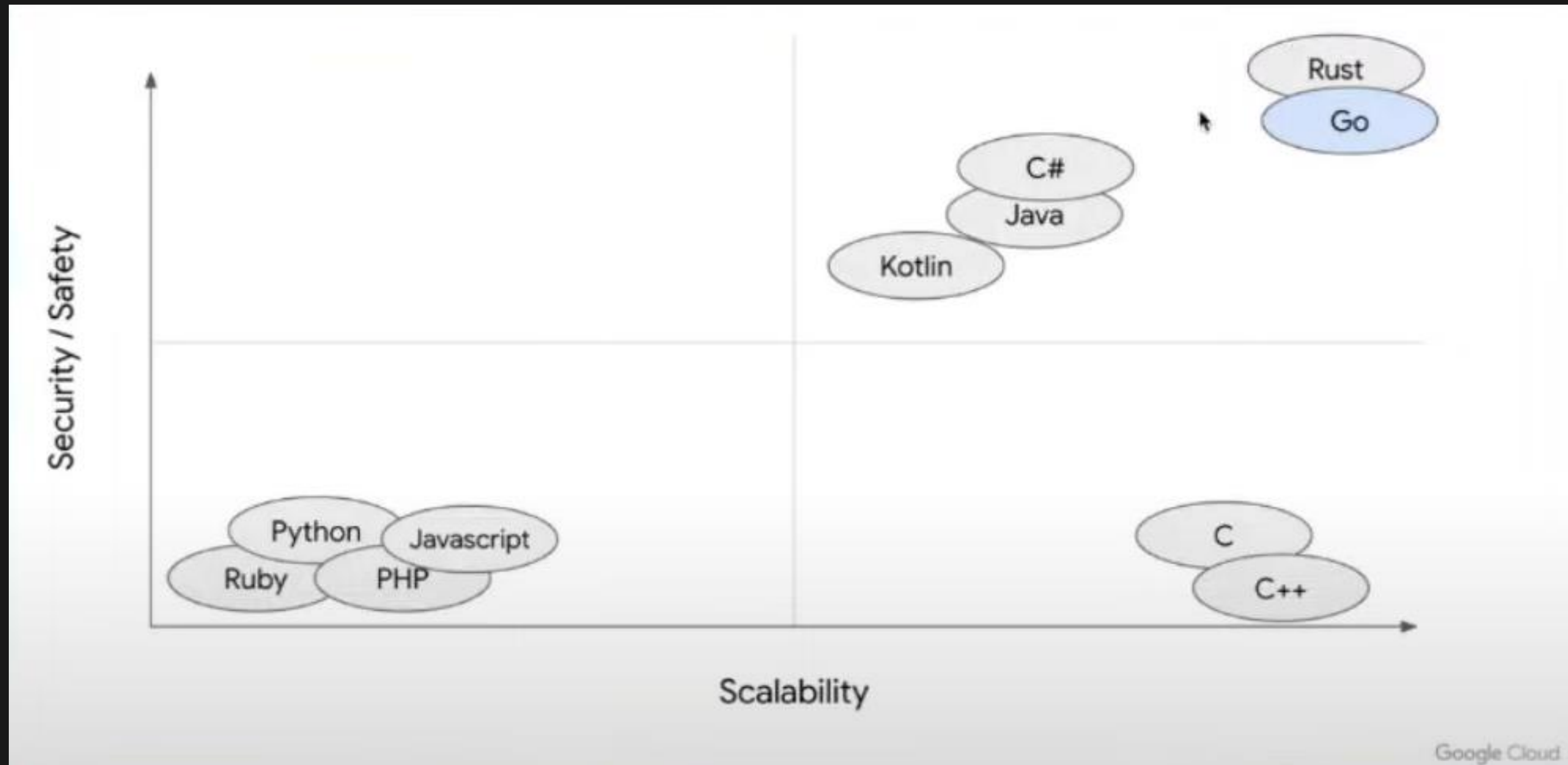
Development Costs X Operating costs



Google Cloud

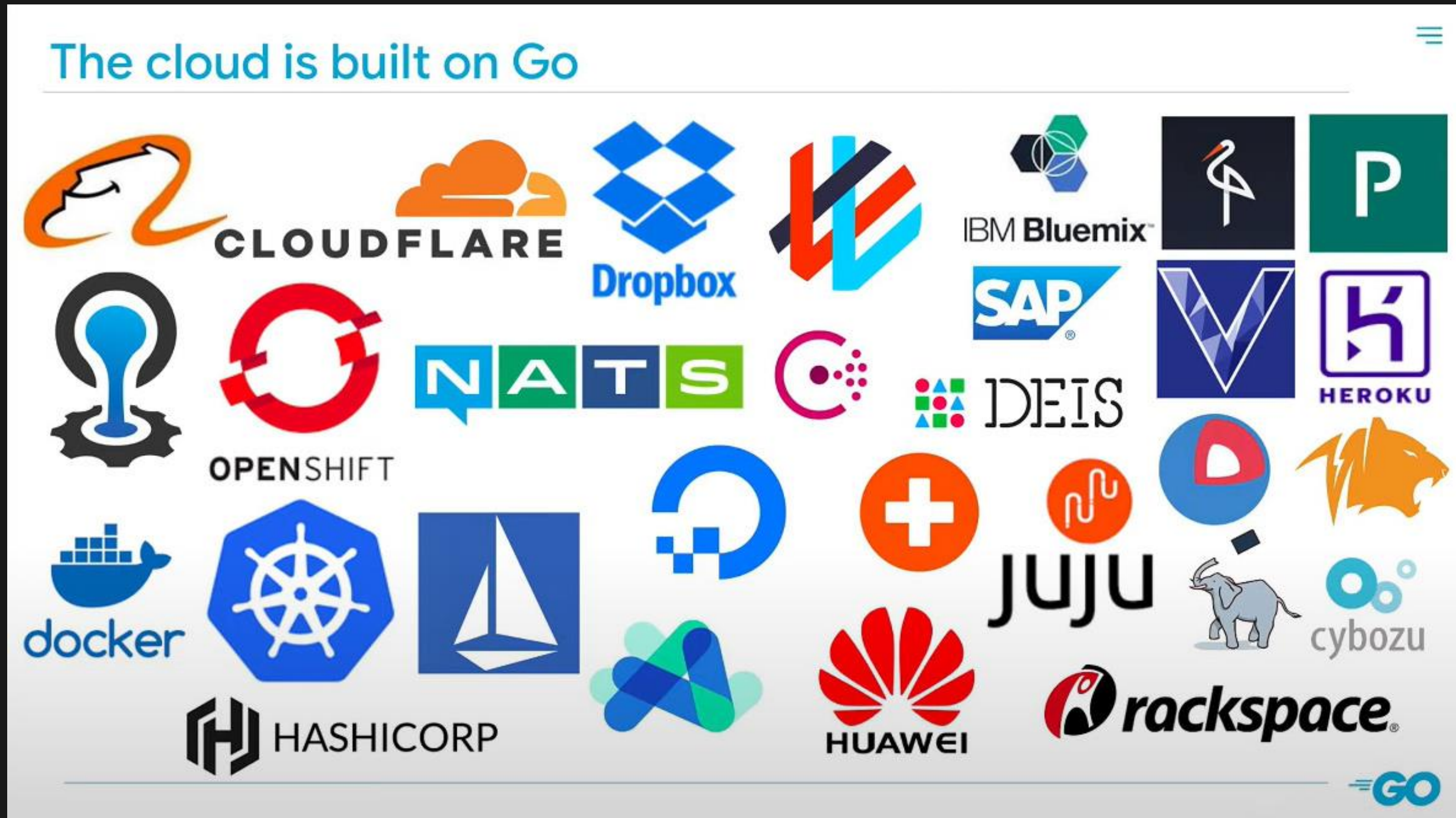
Source: <https://www.youtube.com/watch?v=u56Yp3Egzao>

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

Enterprises adopt Go

Google



redhat

Alibaba Group

UBER



Microsoft



HUAWEI

Hewlett Packard
Enterprise

amazon

Disney

stripe

Capital One

Yandex



Walmart

IBM

LAZADA
GROUP



Dropbox





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.

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.



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.



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.



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 go lang and parallelism reducing time **from 1 hour to just 8 minutes (7.5x faster)**

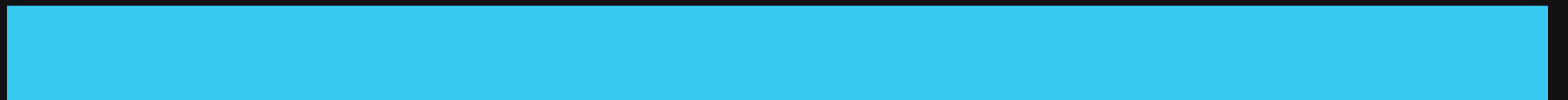


```
package main

import (
    "fmt"
)

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

GOPHER



LETS DIVE INTO THE CODE

Structs

GO main.go X

GO main.go > main

```
1  package main
2
3  import "fmt"
4
5  type Movie struct {
6      Name    string
7      Rating int
8  }
9
10 func main() {
11     m := Movie{
12         Name:    "Deadpool",
13         Rating: 10,
14     }
15     fmt.Println(m)
16 }
17
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

COMMENTS

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

Pointers

~GO main.go >  updateStringPointer

```
5 func main() {
6     var movie string
7     movie = "Deadpool"
8
9     fmt.Println(movie)
10
11    updateString(movie)
12    fmt.Println(movie)
13
14    updateStringPointer(&movie)
15    fmt.Println(movie)
16 }
17
18 func updateString(str string) {
19     str = fmt.Sprintf("This won't be updated: %s", str)
20 }
21 
22 func updateStringPointer(str *string) {
23     *str = fmt.Sprintf("Your movie is: %s", *str)
24 }
25
```

PROBLEMS 50 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

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

Webserver

GO main.go X

3 webserver > GO main.go > main

```
1  package main
2
3  import "net/http"
4
5  func main() {
6      http.HandleFunc("/", HelloVanlug)
7      http.ListenAndServe(":8080", nil)
8  }
9
10 func HelloVanlug(w http.ResponseWriter, r *http.Request) {
11     hello := []byte("<h1>Hello VanLUG!</h1>")
12     w.Write(hello)
13 }
14
```

PROBLEMS 2

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS 1

COMMENTS

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

Routines

GO main.go X

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

```
7
8 func main() {
9     /*
10      Whenever we want to run a function asynchronously (in a separated lightweight process)
11      we just add the reserved word "go" in front of it
12      and Voilà
13      */
14     go counter(10)
15     go counter(10)
16     counter(10)
17 }
18
19 func counter(n int) {
20     for i := range n {
21         fmt.Println(i)
22         time.Sleep(time.Second)
23     }
24 }
25
```

Channels

GO main.go X

5 channels > GO main.go > main

```
1  package main
2
3  import "fmt"
4
5  type Movie struct {
6      Name    string
7      Rating  int
8  }
9
10 func (m *Movie) Format() string {
11     return fmt.Sprintf("%s rating is %v", m.Name, m.Rating)
12 }
13
14 func main() { //Main process - T1
15     /*
16      * A channel has a type of your choice, where you are can send / receive data
17      * between go routines (separated processes)
18      */
19     channel := make(chan Movie)
20
21     go func() { // Another process - T2
22         //Sending data
23         channel <- Movie{Name: "Deadpool", Rating: 9}
24     }()
25
26     //T1
27     // Reading data
28     m := <-channel
29     fmt.Println(m.Format())
30 }
```

31

Simple load balancer simulation

main.go

6 load balancer > main

```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func worker(workerId string, data chan int) {
9     // This will loop the channel until this channel is closed
10    // otherwise it will keep reading it
11    for x := range data {
12        fmt.Printf("WorkerID: %s | Value: %v\n", workerId, x)
13        time.Sleep(time.Second)
14    }
15 }
16
17 func main() {
18     channel := make(chan int) //channel := make(chan int, 10) - Example of a channel with a "buffer" size of 10
19
20     // go worker("John", channel)
21     // go worker("Hugo", channel)
22     // go worker("Rachel", channel)
23
24     // Start amount of workers that will listen the channel
25     // whoever is available is going to read the channel first
26     workers := 10000
27     for i := range workers {
28         go worker(GenWorkerName(i), channel)
29     }
30
31     // Simulating a large amount of data being pushed to the channel
32     dataset := 100000
33     for i := range dataset {
34         /*
35          * Only one single value can be pushed to channel at a time.
36          * Execution will be blocked until someone reads from the channel
37          * It is possible to create a channel with a "buffer" of size X
38          */
39         // sending data
40         channel <- i
41     }
42 }
43
44 func GenWorkerName(x int) string {
45     return fmt.Sprintf("John", x)
46 }
47
```



WRAPPING UP

01

Go is a simple, efficient, and powerful language.

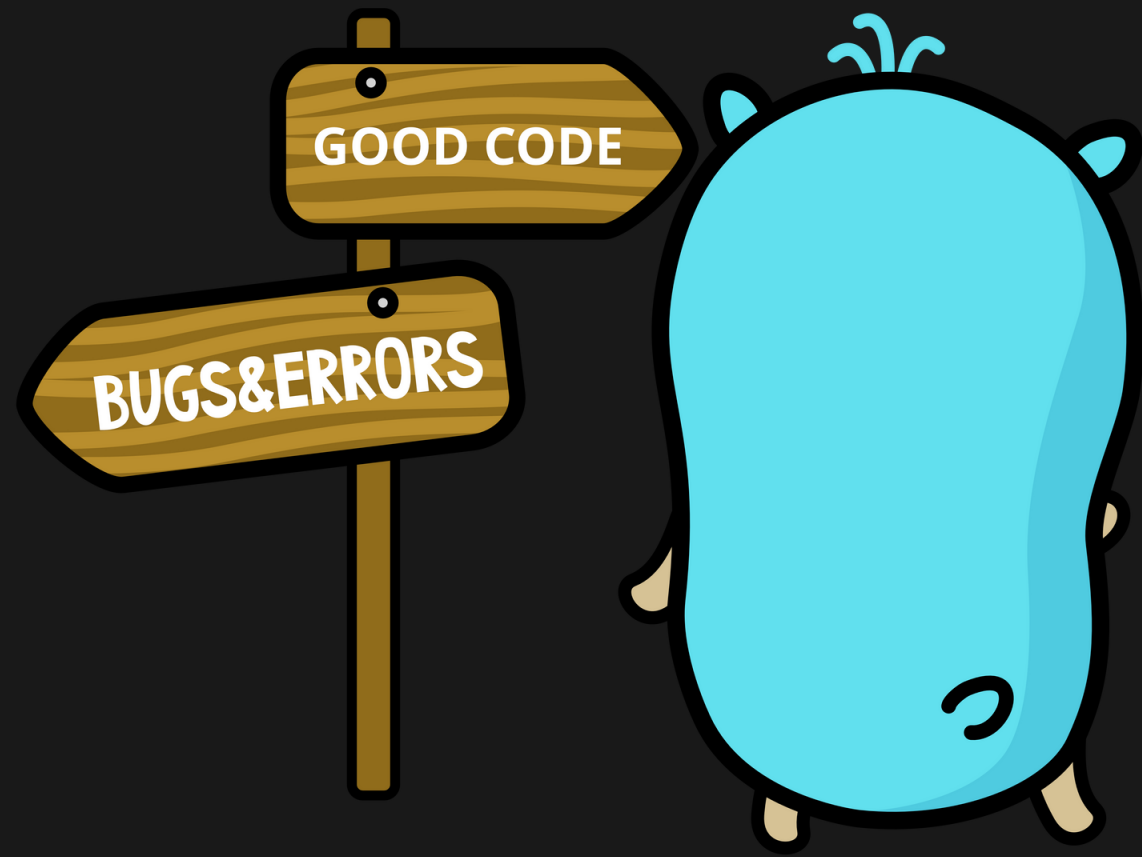
02

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