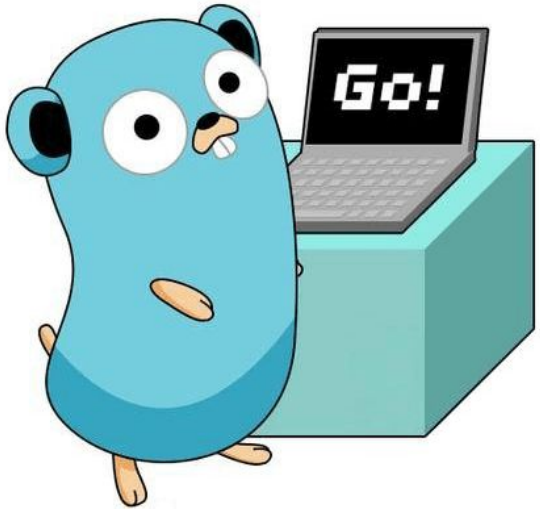
Intro To Golang



Presented By: Raza Z Sayed.

✉️ raza@heady.io

https://github.com/razasayed

Why another programming language ?

- Developed at Google in 2007 to solve performance and scalability issues with their large C++ codebase. The Go team at Google who designed the language consisted of Rob Pike, Ken Thompson and Robert Griesemer.

- Go team wanted the performance benefits of statically typed languages like C++ combined with the productivity benefits of dynamic languages like Javascript, Python, Ruby etc.

- End of Moore’s Law - It was not possible to make faster processors any more so multicore architectures started becoming common. Go team wanted to be able to take full advantage of multicore processor architectures by offering concurrency and “true” parallelism and without the safety and scalability challenges of multithreaded programming.

Features Of Go

- Statically typed. Therefore we get all the benefits associated with type safety.
- Compiled with fast compile time. This is makes it more performant and faster than interpreted languages.
- Automatic garbage collection.
- Lots in common with C without the complexity of C.
- Supports object oriented “style” of programming but is “not” object oriented. Therefore type system is lightweight as there is no type hierarchy as in OO languages.
- A small language with not too many “language features”. To compensate for that has a powerful standard library. Developers don't need to rely much on third party packages.
- Package management using Go modules. Similar to NPM, PIP, Rubygems.
- Inbuilt support for testing. No need to install a separate package for testing.
- Concurrency system which is much more powerful and scalable than using threads.
- Highly opinionated.

Some differences between Javascript and Go

Typing

Strongly typed.

Datatypes: String, Float, Int, Byte, Struct etc.

Dynamically typed.

Typescript.

Structures to define attributes and behavior

Structs, Pointers, Methods,Interfaces.

Prototypes, ES6 classes.

Concurrency

Multi-threaded.

Goroutines and Channels.

Single-threaded.

Callbacks, async await etc.

Error Handling

Explicit.

Errors are values which need to be explicitly handled.

Built In

Try catch, Exceptions.

Opinionated-ness

Strongly opinionated.

Conventions.Built in tooling and linters.

Fluid Opinions.

Can be different across projects.

Talk is cheap. Show me the code.

— Linus Torvalds —

Things you need to get started

- **Go compiler:** Download the binary for your operating system from https://golang.org/dl/
- **IDE options:** Microsoft VSCode with Go plugin installed, JetBrains GoLand, Vim 😉 etc.
- **REPL:** No inbuilt REPL. But we can use Go playground at https://play.golang.org/ which offers same functionality as a REPL.

Creating a Go project demo 

Go Playground

The Go Playground | Run | Format | Imports | Share | Hello, playground | About

```
1  package main
2
3  import (
4          "fmt"
5  )
6
7  func main() {
8          fmt.Println("Hello, world")
9  }
10
11
12
13
14
```

```
Hello, world

Program exited.
```

https://play.golang.org

Defining a simple greeting function

https://github.com/razasayed/intro_to_golang_workshop/blob/main/greet/greet.go

Writing test for the greeting function

https://github.com/razasayed/intro_to_golang_workshop/blob/main/greet/greet_test.go

A simple webserver

https://github.com/razasayed/intro_to_golang_workshop/blob/main/webserver/server.go

Calling external api

https://github.com/razasayed/intro_to_golang_workshop/blob/main/external_api/api.go

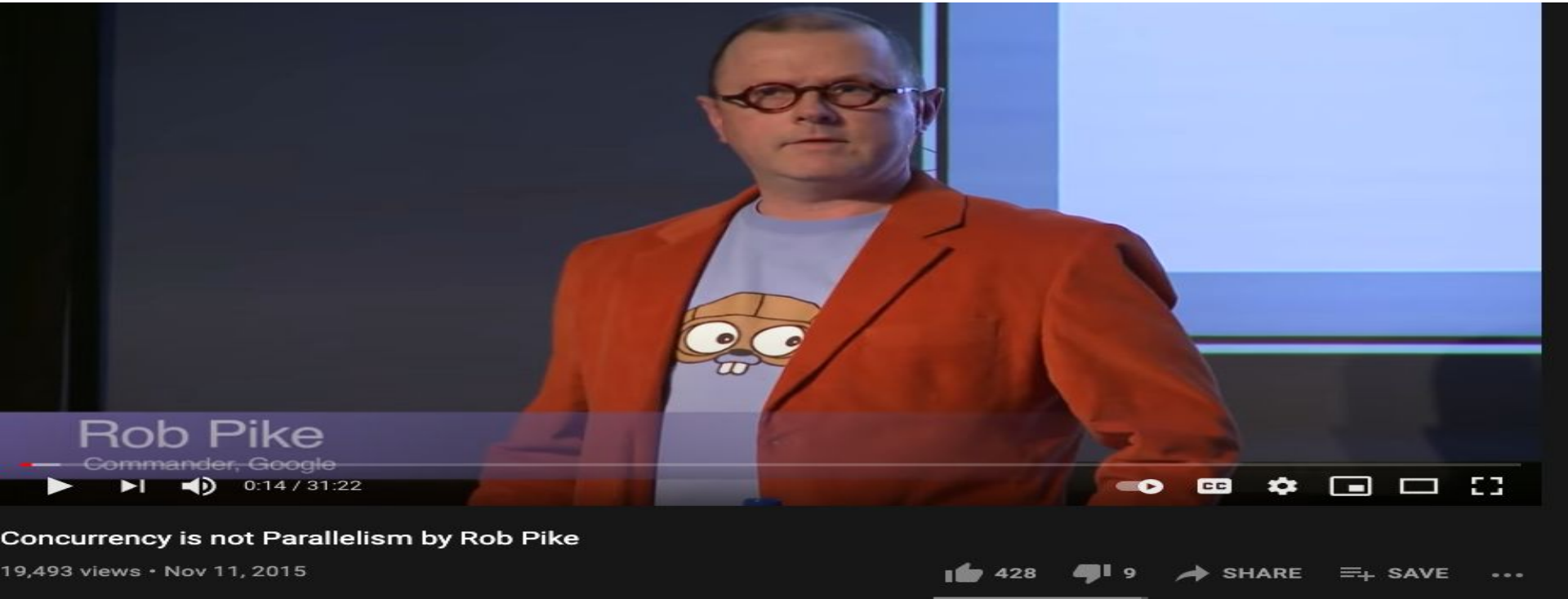
Using external dependency

https://github.com/razasayed/intro_to_golang_workshop/blob/main/using_dependency/uuid.go

Concurrency

Concurrency is not parallelism

https://www.youtube.com/watch?v=oV9rvDllKEg



The Problem 🔥

```
[error] (run-main-0) java.lang.OutOfMemoryError: unable to create native thread:
[error] java.lang.OutOfMemoryError: unable to create native thread:
[error]         at java.base/java.lang.Thread.start0(Native Method)
[error]         at java.base/java.lang.Thread.start(Thread.java:813)
...
[error]         at java.base/java.lang.Thread.run(Thread.java:844)
```

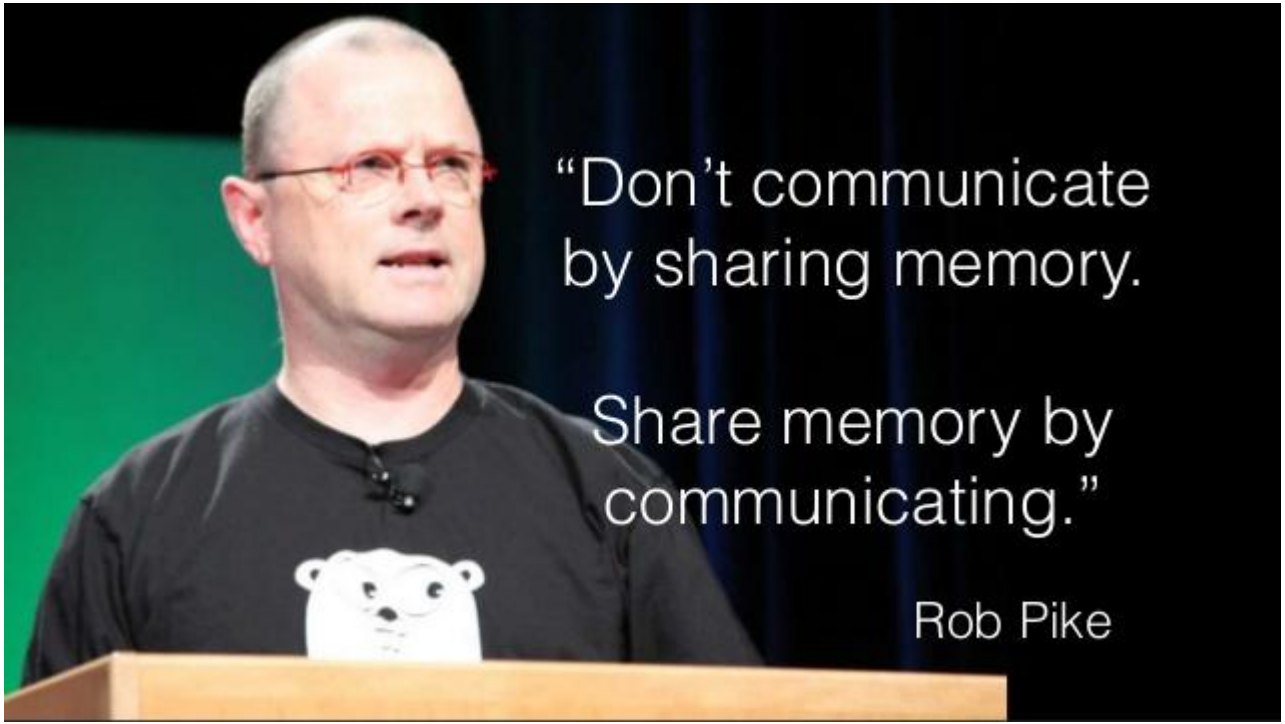
Issues with Threads 💣 🧵

- **Large memory cost per thread** - JVM languages or any language that uses threads uses operating system threads. Threads are created by the kernel/operating system and each thread consumes about 1 MB of RAM and is a fixed stack size. So, in 1 GB of memory we can only have 1000 threads. So, if a program creates a lot of threads it can soon run out of memory.

- **Context switching delay** - Threads are preemptively scheduled by the operating system. There is significant delay when switching from one thread to another as the operating system needs to save the context of the current thread in multiple CPU registers. Therefore threads dont get much time to do useful work.

- **Memory sharing** - Threads communicate by sharing memory. This can lead to lot of synchronization problems, accidental memory modification and hard to track bugs.Solutions involve using locks, semaphores, mutexes etc. but still it is the responsibility of the programmer to use these correctly.

The Go solution



No threads in Go. Use Goroutines !

- Threads are created and scheduled by the operating system. Goroutines are created and scheduled by Go runtime on top of operating system threads.
- Each OS thread has >= 1 MB of fixed stack size. A Goroutine in contrast has dynamic stack size and starts out with only 2 KB of memory and can use more if required. Hence Go schedules multiple Goroutines on a single OS thread. With 1 GB of memory you get 1000 threads but millions of Goroutines!.
- Unlike threads Goroutines don’t communicate by sharing memory (This refers to dont communicate by sharing memory in previous slide). Goroutines communicate by sending messages over “channels” to other Goroutines (Share memory by communicating).

Goroutine demo

https://github.com/razasayed/intro_to_golang_workshop/blob/main/concurrency/goroutines.go

Channels

- Goroutines unlike threads dont communicate by sharing memory. They communicate by sending messages to each other over a channel.
- Channels are based on the idea of CSP (Communicating Sequential Processes) specified by Tony Hoare in 1978.
- Goroutines and Channels is different from Actors in languages like Erlang and Scala which also communicate by sending messages. Actors use a mailbox so one actor can send message to another even when another is not available to receive it. Channels which are based on CSP concept are analogous to a telephone communication between two goroutines. One can send message to another only when the other is ready to receive.

Difference between Actors and CSP models (Pseudocode)

Actors model in Erlang, Scala etc.

a = new Actor

b = new Actor

a -> b(“message”) // a sends message to b’s mailbox

CSP model followed by Goroutines in Go.

a = new Goroutine

b = new Goroutine

c = new Channel

a -> c(“message”) // a sends message over channel c to b

b <- c(“message”) // b receives message from a over channel c

Channels demo

https://github.com/razasayed/intro_to_golang_workshop/blob/main/concurrency/channels.go

Documentation 

Building REST API's

- **Frameworks:** Gorilla Mux (https://www.gorillatoolkit.org/), Gin (https://gin-gonic.com/), Martini (https://github.com/go-martini/martini) etc.
- **ORM's/ODM's:** GORM (https://gorm.io), sqlx (https://github.com/jmoiron/sqlx), mgm for MongoDB (https://github.com/Kamva/mgm) etc.
- **Full Stack Web frameworks:** There are also full stack web frameworks available with templating and other features built in. Some of the popular ones are Revel (http://revel.github.io/), Beego (https://beego.me) etc.

REST API boilerplate and demo

- Uses the Gin web framework (https://gin-gonic.com/) to create a simple REST api for movies. Gin is one of the highly performant framework for building REST api's in Go.
- Uses the GORM ORM (https://gorm.io) to connect to a sqlite database. GORM can also be used with the following relational databases: mysql, postgres and sql server.
- Code: https://github.com/razasayed/golang_rest_api_boilerplate
- PR's are welcome 🙂

Useful Resources To Learn Go

- Tour Of Go (https://tour.golang.org)
- http://www.golang-book.com/
- Go By Example (https://gobyexample.com/)
- Learn Go With Tests (https://quii.gitbook.io/learn-go-with-tests/)

Etc !

One of my favorite talks...

To understand why Go is designed the way it is:

Simplicity Is Complicated by Rob Pike

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

Questions ?

Thank You 🙏

Source Code:

https://github.com/razasayed/intro_to_golang_workshop

https://github.com/razasayed/golang_rest_api_boilerplate