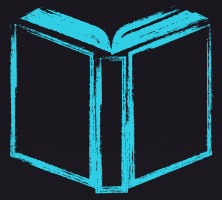
Welcome To The Course

Maximilian Schwarzmüller

@maxedapps

academind.com



Course Prerequisites

NO prior Go knowledge required!

Basic programming knowledge (with any language) is **strongly recommended!**

Go is an open-source programming language developed by Google

Focus on **simplicity**, **clarity** & **scalability**

Inspired by languages like Python

Aims to provide a clean, understandable syntax

High performance & Focus on Concurrency

Similar to C or C++

Popular for tasks that benefit from multi-threading

Batteries included

Go comes with a standard library

Many core features are built-in

Static typing

Go is a type-safe language

Allows you to catch many errors early

Maximilian Schwarzmüller — “Go - The Complete Guide”

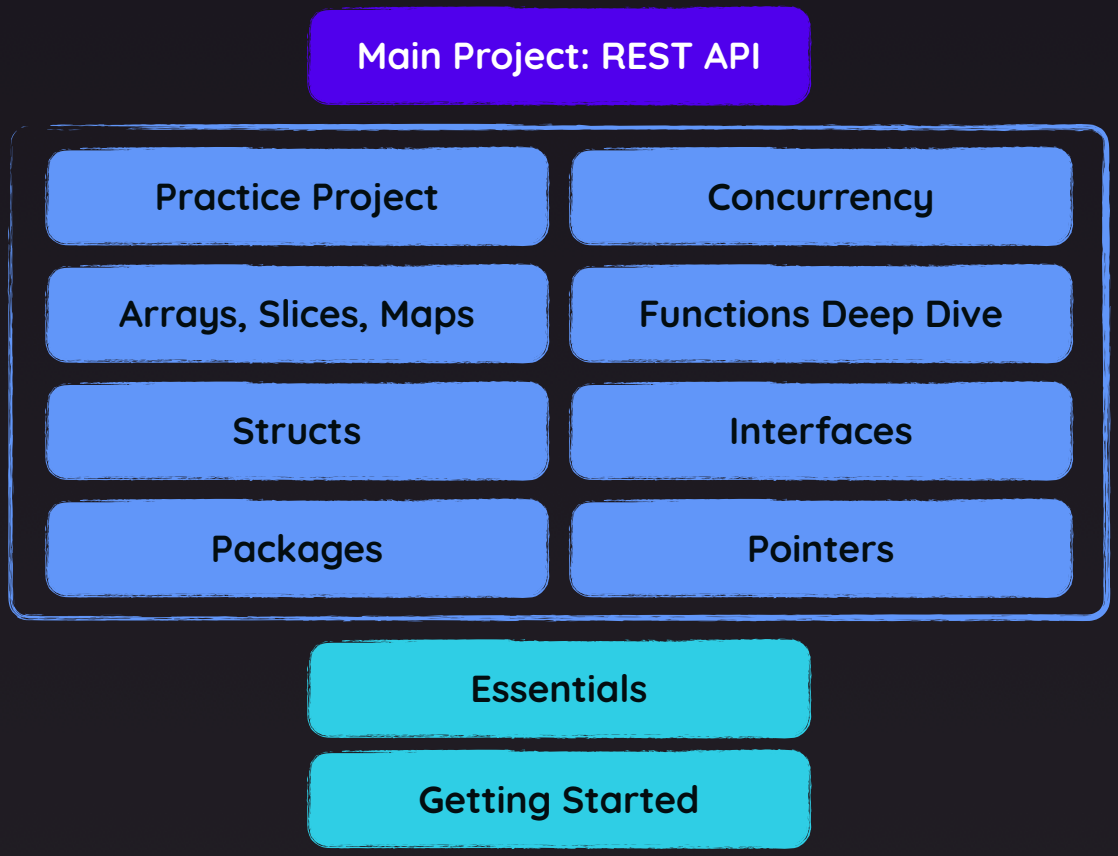
Popular Use-cases

Networking & APIs

Microservices

CLI Tools

About This Course



How To Get The Most Out Of This Course

Meet the Prerequisites

Basic programming knowledge (any language!) is recommended

Watch the Videos

Watch them at your pace

Slow me down or speed me up

Pause & rewind

Repeat sections

Practice!

Pause & practice on your own

Build dummy demo projects

Help each other

Use code attachments if stuck

Ask & answer in the Q&A section

Find fellow developers on our Discord

Go Essentials

Values, Basic Types & Core Language Features

- Understanding the **Key Components** of a Go Program
- Working with **Values & Types**
- Creating & Executing **Functions**
- Controlling Execution with **Control Structures**

Practice Time!

Build a "Profit Calculator"

Ask for **revenue**, **expenses** & **tax rate**

Calculate **earnings before tax** (EBT) and **earnings after tax** (profit)

Calculate **ratio** (EBT / profit)

Output EBT, profit and the ratio

Types

Every value in Go has a specific type

`int`	A number **without** decimal places	`-5, 12, 101, -58, 0, 1`
`float64`	A number **with** decimal places	`-15.9, 0.0, 23.12, 18.1, 122.55`
`string`	A text value	`"Hello", "Hi there", `I'm Max``
`bool`	A value that's either **true** or **false**	`true, false`

Strings are created with double quotes ("") or backticks (``). The latter syntax allows you to write strings across multiple lines

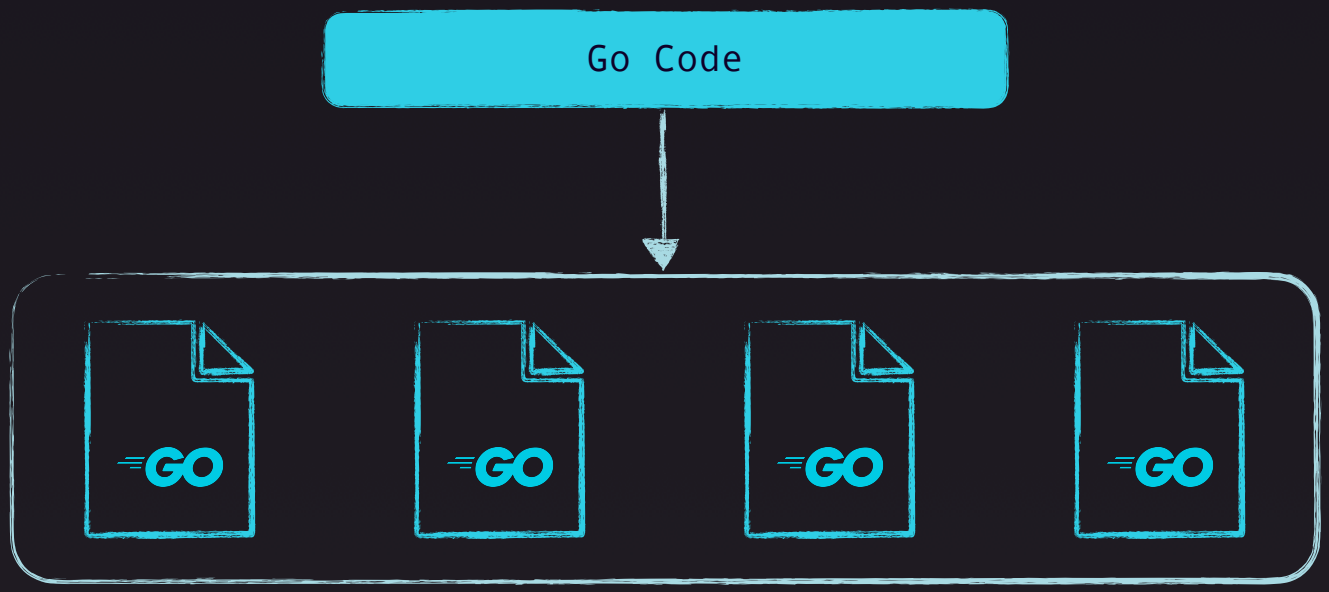
These are the most important basic types in Go (covered later in the course)

Packages — A Closer Look

Working with Go Packages

- Splitting Code Across **Multiple Files**
- Splitting Files Across **Multiple Packages**
- **Importing & Using** Custom Packages

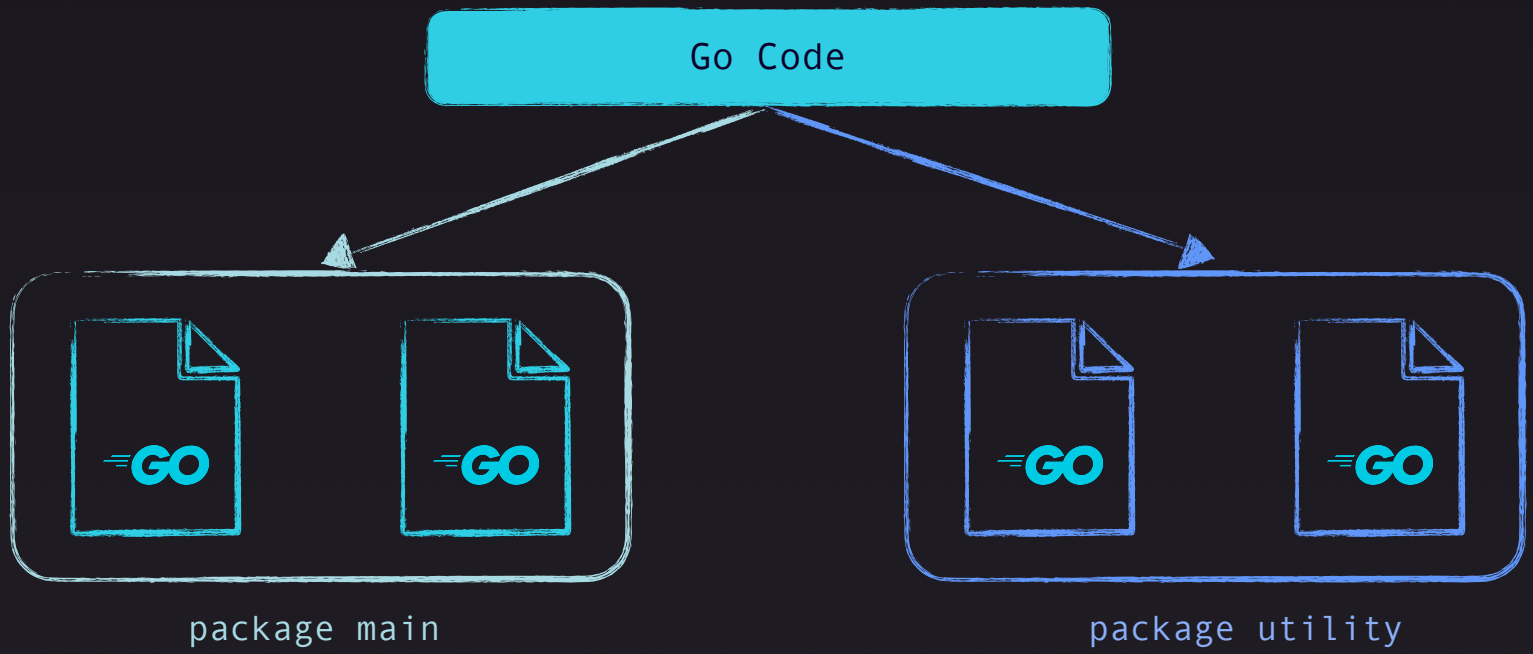
Files & Packages



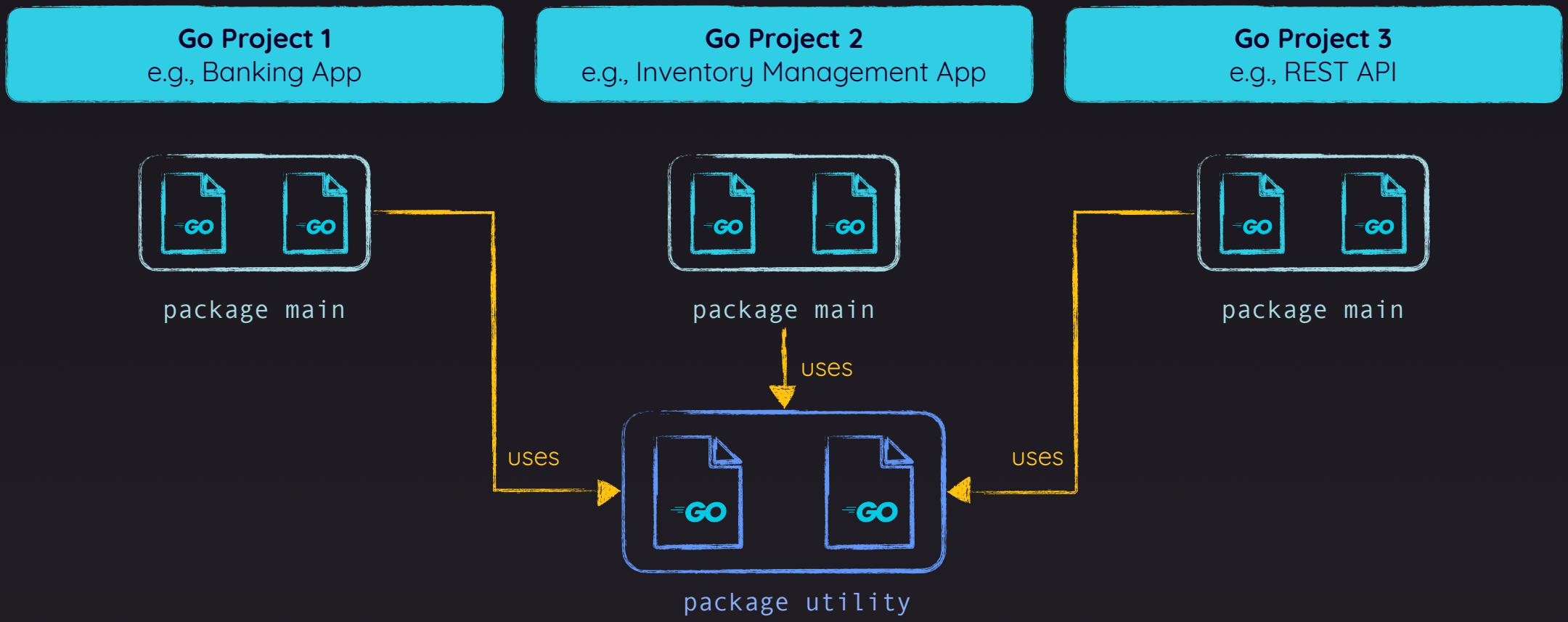
`package main`

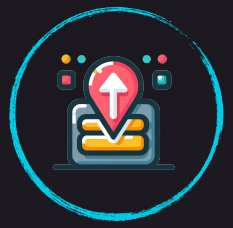
(could be any package name, as long as all files belong to the same package)

Files & Packages



Files & Packages





Understanding Pointers

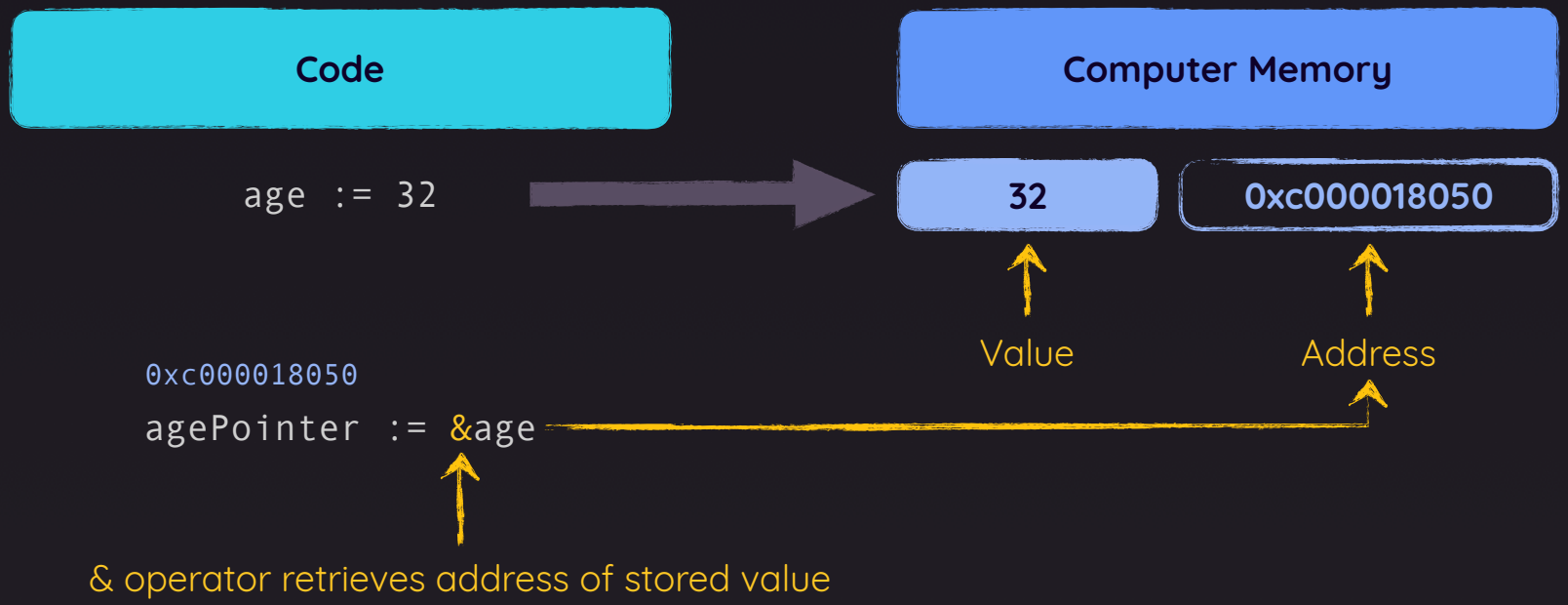
Working With Addresses Instead Of Values

- **What** are Pointers?
- **Why** does this feature exist?
- **How** can you work with Pointers?

What are "Pointers"?

What are "Pointers"?

Variables that store value **addresses** instead of values



Why "Pointers"?

Avoid Unnecessary Value Copies

By default, Go creates a copy when passing values to functions

For very large & complex values, this may take up too much memory space unnecessarily

With pointers, only one value is stored in memory (and the address is passed around)

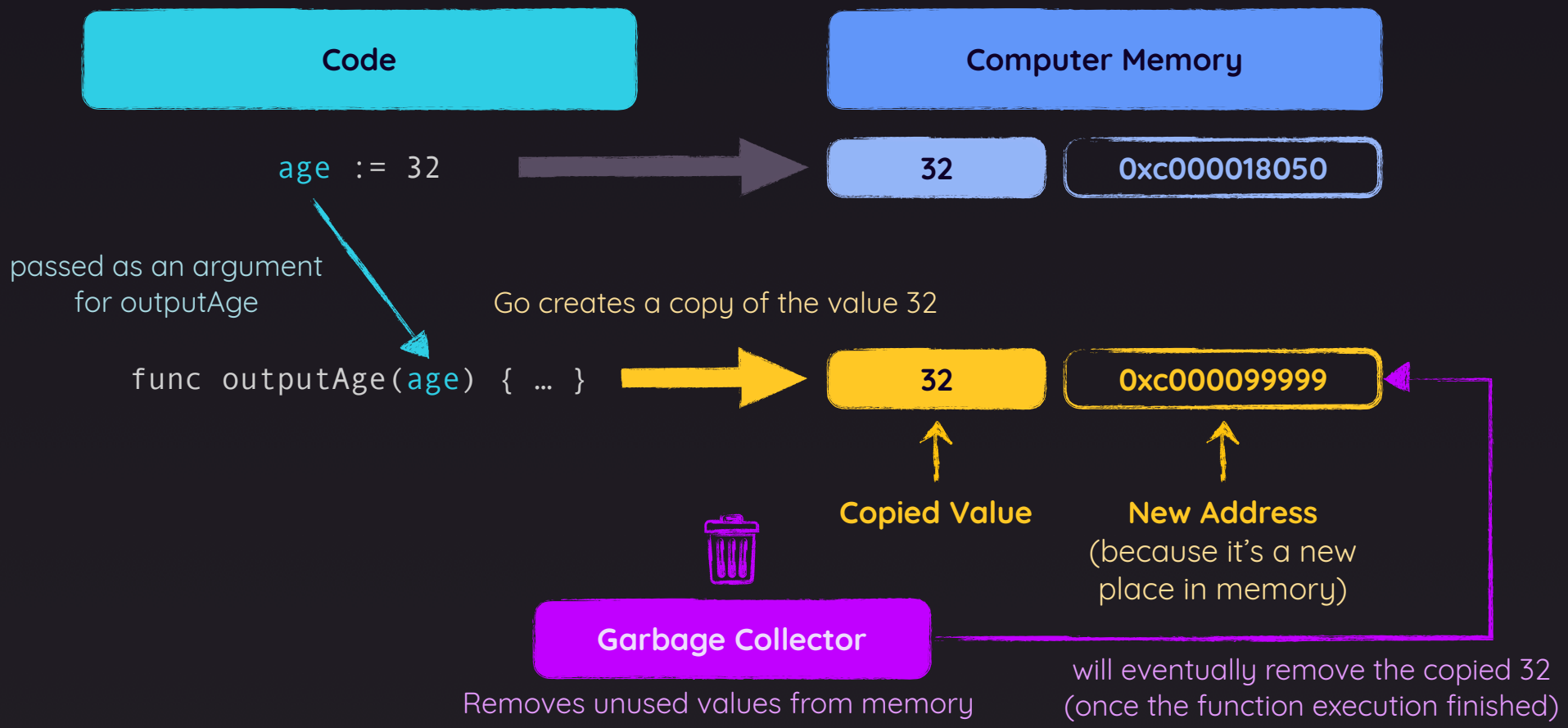
Directly Mutate Values

Pass a pointer (address) instead of a value to a function

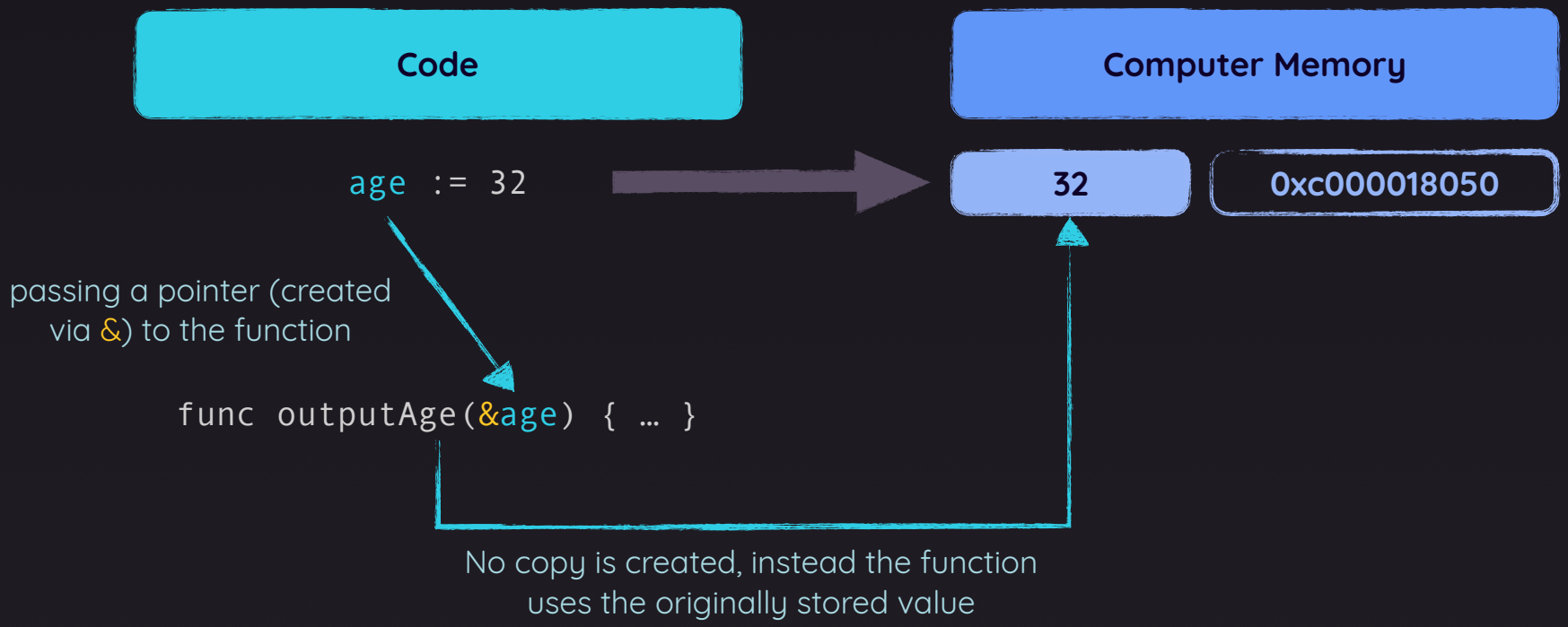
The function can then directly edit the underlying value — no return value is required

Can lead to less code (but also to less understandable code or unexpected behaviors)

Passing Values To Functions



Passing Pointers To Functions



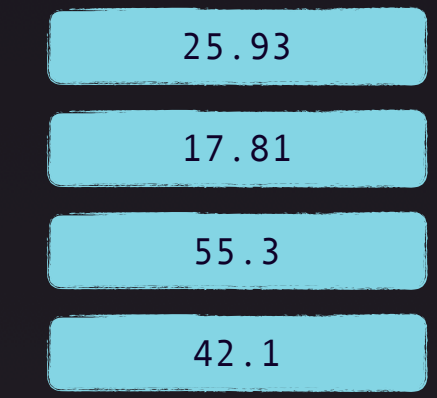
Arrays, Slices & Maps

Storing Data In Collections

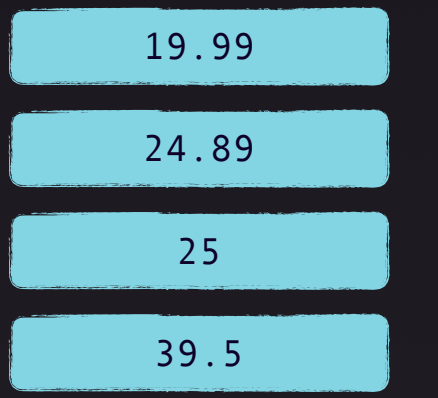
- Understanding **Arrays** & Array **Limitations**
- Understanding & Using **Slices**
- Slices **Deep Dive**
- Understanding & Using **Maps**

Dealing With List Data

Example: Weather Data



Example: Product Prices



Managing List Data With Arrays

Functions Deep Dive

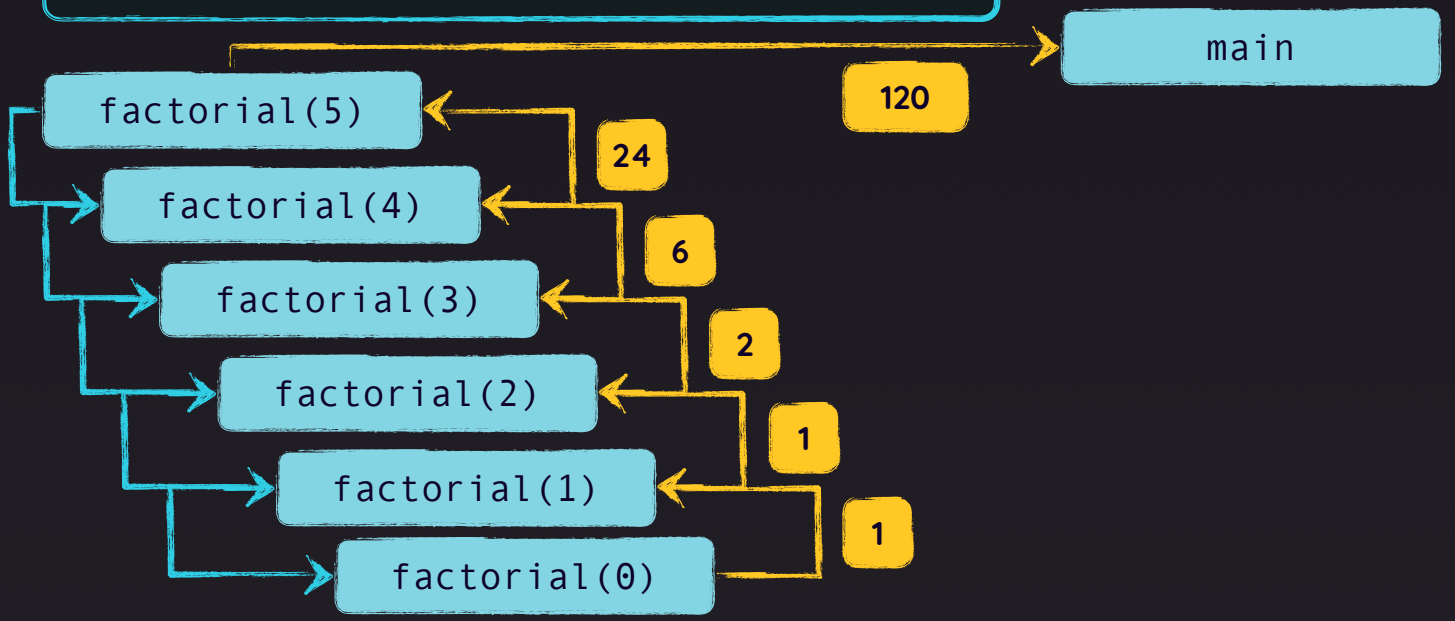
Beyond The Basics

- Using Functions **as Values**
- **Anonymous** Functions
- **Recursion**
- **Variadic** Functions

Recursion In Action

```
func factorial(number int) int {
  if number == 0 {
    return 1
  }

  return number * factorial(number - 1)
}
```



Multiple executions of the same function

(with different) values

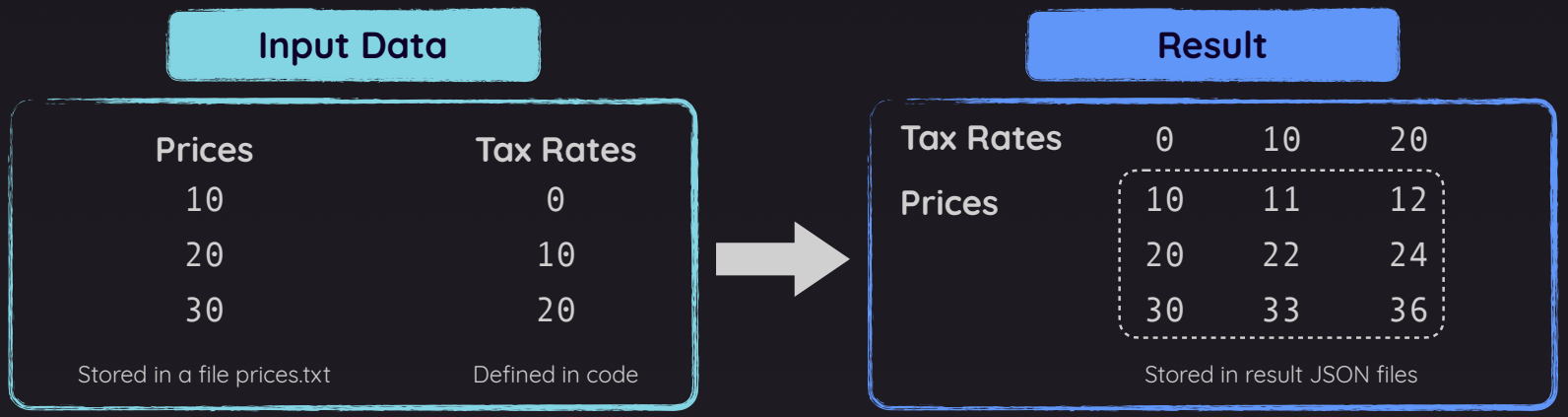
Practice Time!

Practice Working with Structs, Slices, Maps, Interfaces & More

- Build a **Complete Demo App** From The Ground Up

Project Description

A tool that calculates “tax included” prices for a given list of prices & tax rates



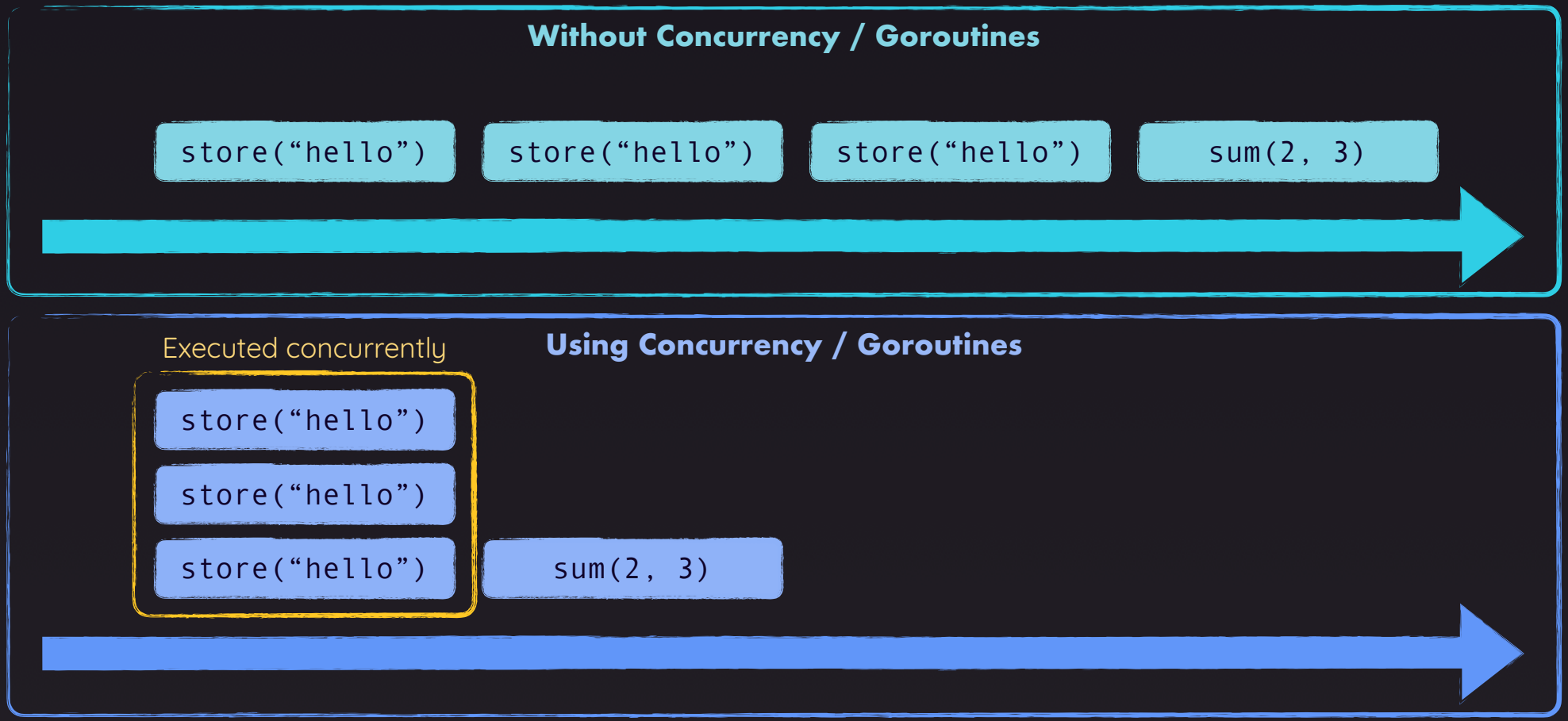
Code will use interfaces, structs, functions, methods, file access, multiple packages & more

Concurrency

Running Tasks In Parallel

- Understanding **Goroutines**
- Sending Data with **Channels**
- Controlling **Code Flow** & **Simultaneous Tasks**

Concurrency & Goroutines

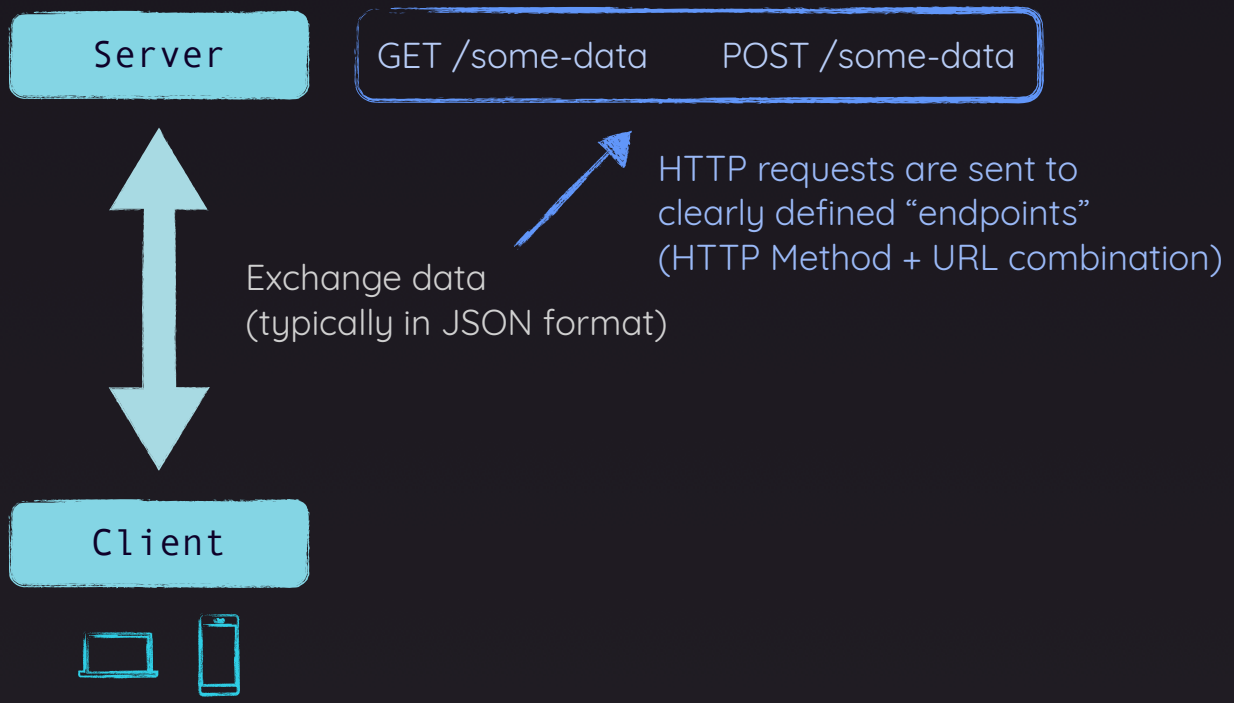


Project: REST API

Building a REST API with Go

- **Planning** The Project
- **Building** the REST API Step By Step

What's A REST API?



Project Description

A Go-powered “Event Booking” REST API