
WLUG
COMMUNITY | KNOWLEDGE | SHARE
GO
GoLang

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Tables of Contents

Introduction	Functions and Strings
Variables and Constants	Structures and Pointers
Loops, If/else and Switch statements	File Systems and Error Handling
Arrays and Slices	Interfaces and Channels
Maps	Routines

Introduction

History of Golang

- Developed by Google
- Creators of Go

  a.Rob Pike b.Ken Thomson c.Robert Greisemer
- Go was released

  a. Born - 2007

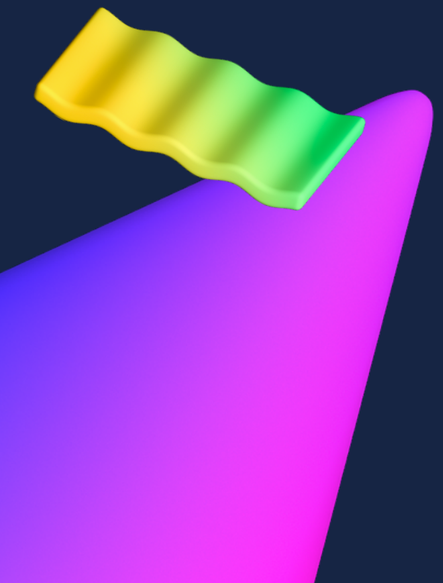
  b. Released as Open-source - 2009

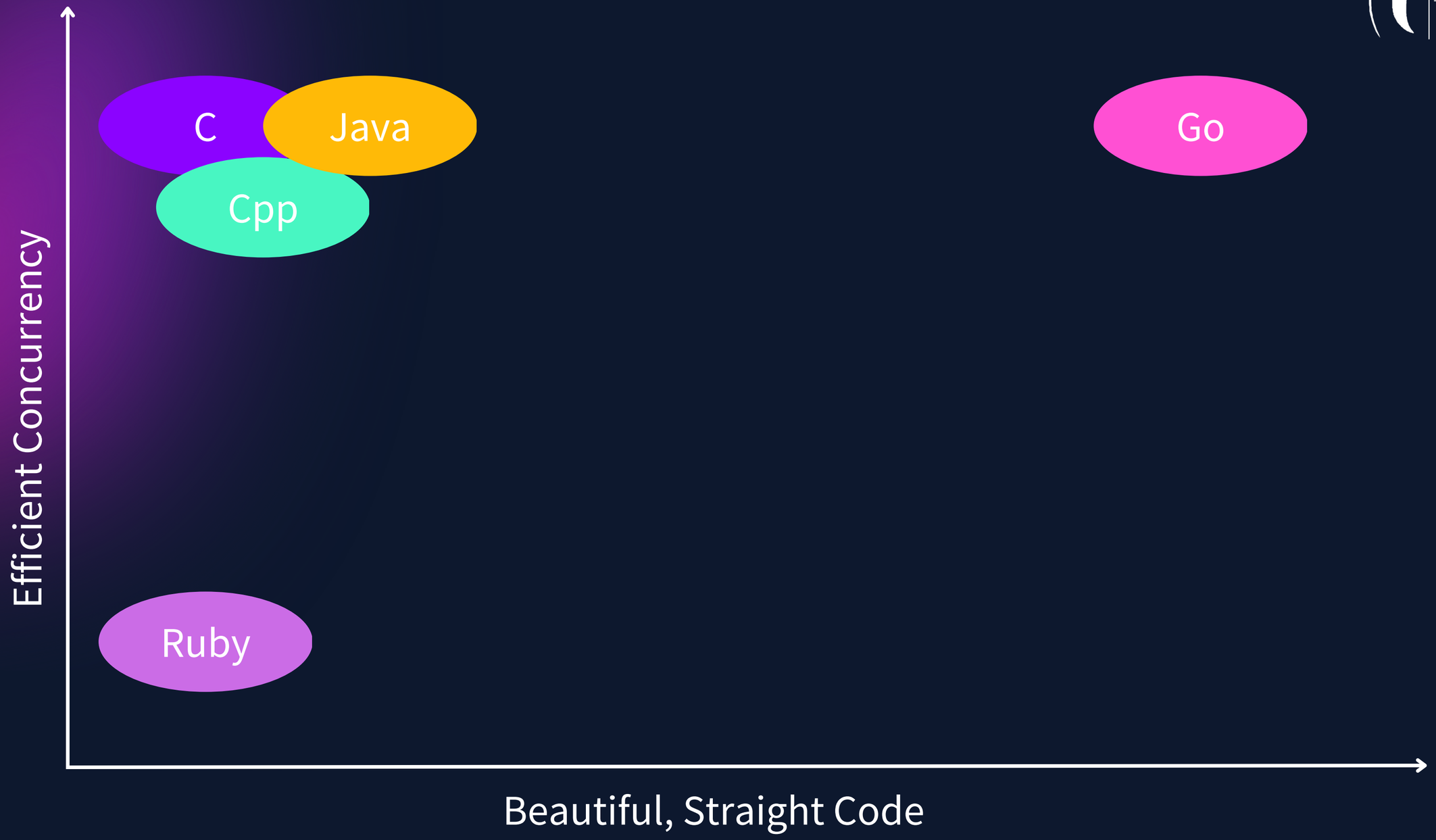
  c. First version was released - 2012

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Features of Golang

- Open-Source Programming language
- Large code bases
- Keep the development productive
- Focus on simplicity




C
Java
Cpp
Go
Ruby
Efficient Concurrency
Beautiful, Straight Code

Why Go?

- Problem with scalability
- Development
- Binaries
- Language Design
- Powerful Standard Library
- Package Management
- Static Typing
- Concurrency Support

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Who uses Go?

- Uber - Microservices
- Netflix - Heavy data processing
- Adobe - Server handling
- Docker - Written in Go
- Other applications are Apple, Youtube, Mozilla Firefox

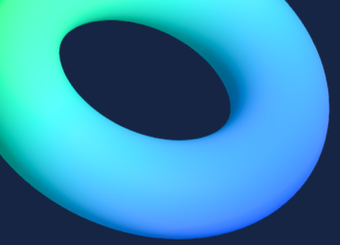






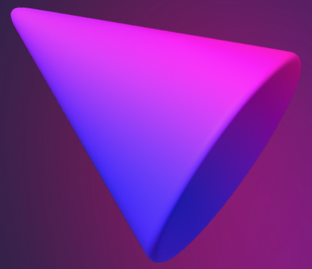
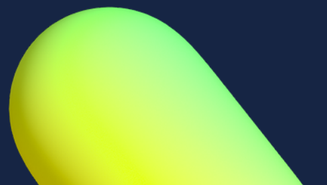
What do you understand?

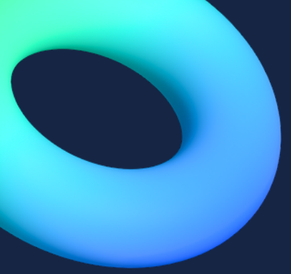
```
package main
import "fmt"
func main() {
    name := "Hello World"
    fmt.Println(name)
}
```



What does package main mean?

- Collection of common source code files
- Inside package there can be multiple files

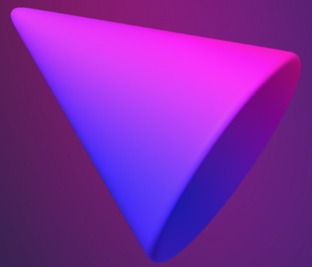




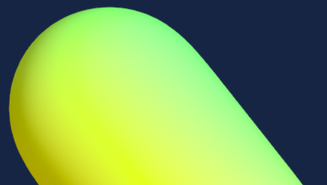


WLUG

COMMUNITY | KNOWLEDGE | SHARE

```
import "fmt"
Main package
Main function
```





WLUG
COMMUNITY | KNOWLEDGE | SHARE

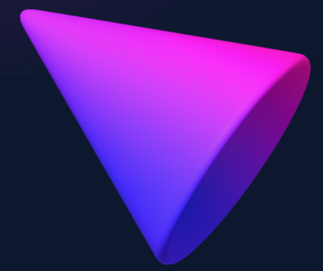
What is func?

- Keyword in Go language used to create a function
- A special type of function that can be defined in a package and it acts as an entry point for the executable program
- Used to programs that are executable

WLUG
COMMUNITY | KNOWLEDGE | SHARE

How do we run the code?

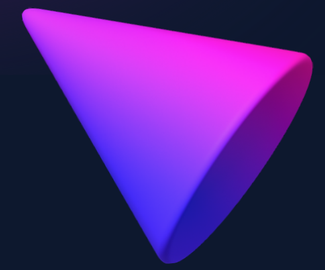
Commands	Functions
go run file_name	Run the program
go build file_name	Build our programs into binaries
./file_name	Execute the program



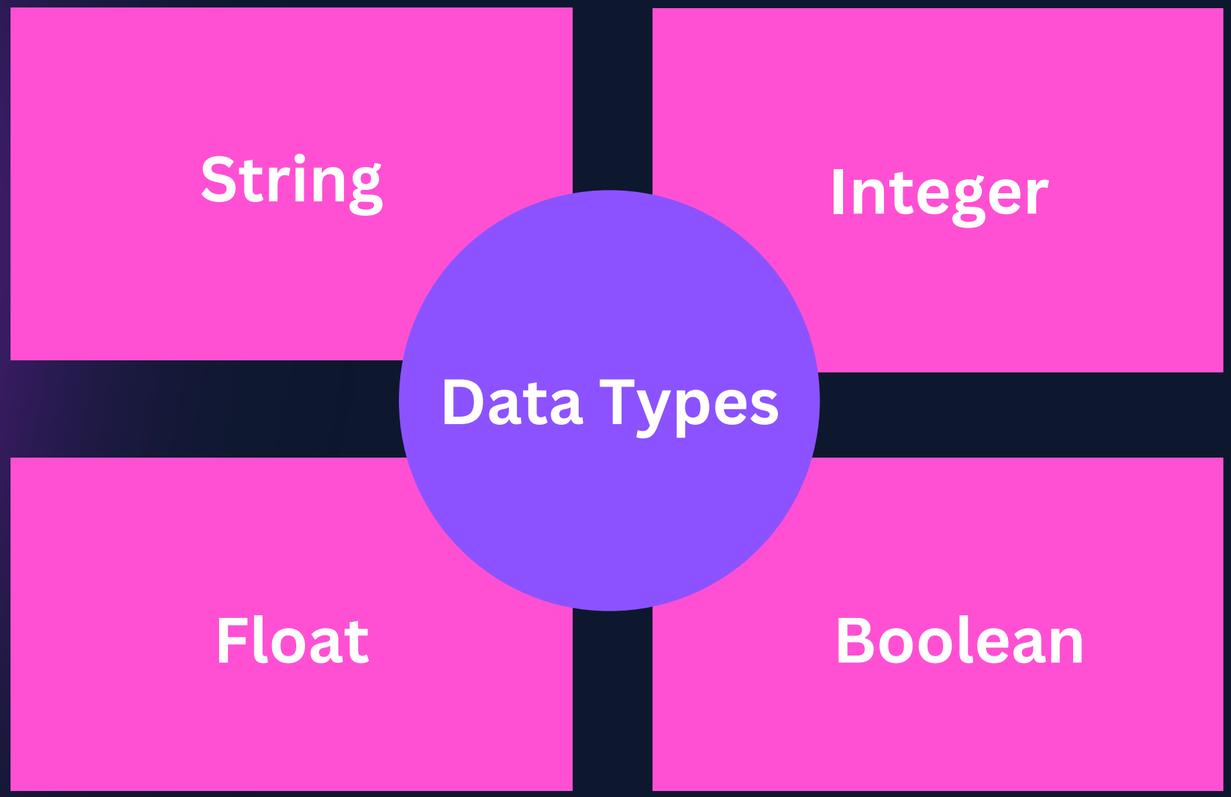
WLUG
COMMUNITY | KNOWLEDGE | SHARE

How is Go file organized?

package main	Package declaration
import "fmt"	Import other packages that we need
func main()	Declare functions, tell Go to do things



Basic Data Types



WLUG
COMMUNITY | KNOWLEDGE | SHARE

Variables & Constants

- Explicit declaration
- Declares 1 or more variables.
- infer the type of initialized variables.
- Variables declared without a corresponding initialization are zero-valued.
- Syntax - var var_name datatype
- Declares a constant value

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Variables

Name of new variable

String will be assigned to the variable

WLUG
COMMUNITY | KNOWLEDGE | SHARE

For Loop

- Go’s only looping construct
- Three types of FOR loop
    a. single condition
    b. classic initial/condition/after for loop
    c. without a condition will loop repeatedly until we break out of the loop

WLUG
COMMUNITY | KNOWLEDGE | SHARE

For Loop

```
for i := 1 ; i < 5 ; i++ {
    fmt.Println("Value of i: ", i)
}
```

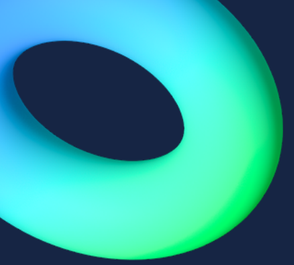
If/Else statement

- If without Else
- Any variables declared in this statement are available in the current and all subsequent branches.
- No need of parentheses around conditions

WLUG
COMMUNITY | KNOWLEDGE | SHARE

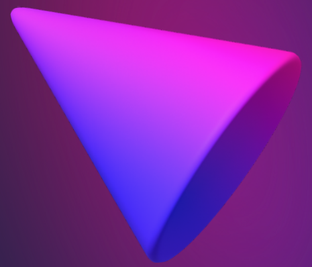
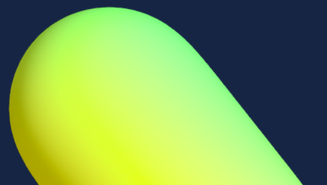
If/Else statement

```
if condition {
    print condition 1 } else {
    print condition 2 }
```



Switch statement

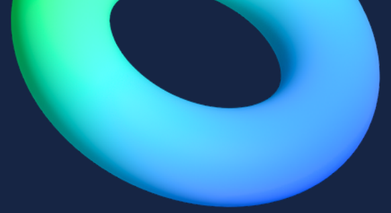
- Express conditionals across many branches.
- Basic switch
- Separate multiple expressions
- Switch without an expression

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Switch statement

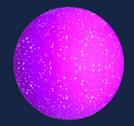
```
switch case_no {
    case_no:
    operation
    default:
    operation
 }
```



WLUG
COMMUNITY | KNOWLEDGE | SHARE

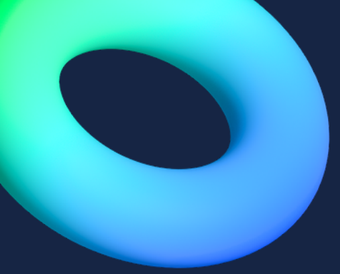
Arrays

- Sequence of elements of a specific length
- By default an array is zero-valued
- len returns the length of an array
- appear in the form [v1 v2 v3 ...] when printed with fmt.Println



Arrays

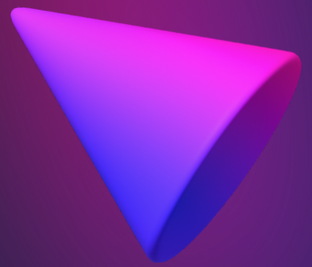
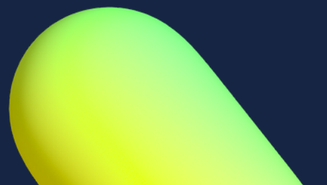
```
var array_name [array_size] data_type
OR
array_name := [array_size] data_type {elements}
```



Slices

a. Basic Operations

- important data type in Go
- lightweight data structure
- variable-length sequence
- To create an empty slice with non-zero length, use the builtin make

Slices

```
slice_name := make([]data_type, size)
```

OR

```
slice_name := [] data_type {elements}
```

Slices

b. Appending a Slice

- Returns a slice containing one or more new values

c. Copying a Slice

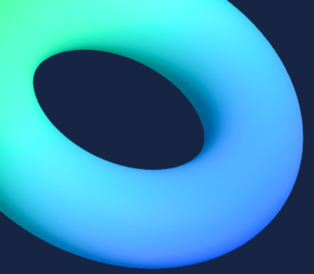
- Create an empty slice c of the same length as s
- Copy into c from s

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Slices

```
slice_name := append(slice_name, new_element)

copy(new_slice_name, old_slice_name)
```



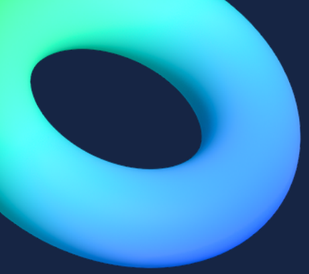
d. Slice Operator

- “slice” operator with the syntax slice [low:high]
- For eg. arr[:7] means all the elements in an array upto index 6


WLUG
COMMUNITY | KNOWLEDGE | SHARE

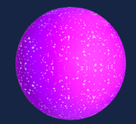
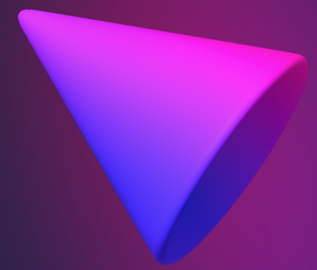
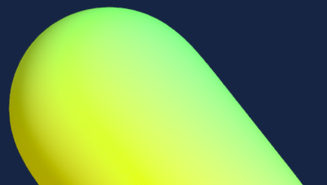
Slices

```
slice_name [lower_index : upper index]
```



Maps

- built-in associative data type
- syntax to create any empty map
  make(map[key-type]val-type)
- Setting the value to map
  name[key]
- len returns the number of key/value pairs when called on a map.
- delete removes key/value pairs from a map

Maps

```
map_name := make(map[key data_type]value_data_type)
```

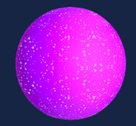
Functions

- Functions are the block of code in a program that can perform a specific task.
- It gives user the ability to reuse the same code.
- Saves excessive use of memory, and provides readability.

Functions Declaration

WLUG
COMMUNITY | KNOWLEDGE | SHARE

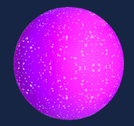
```
func function-name(parameter-list) return-type{
    //function body
}
```



Functions Calling

WLUG
COMMUNITY | KNOWLEDGE | SHARE

```
function-name(parameter-list)
```



Functions Arguments

- The parameters passed to function are called actual parameters, whereas the parameters received by a function are called formal parameters.

Pointers

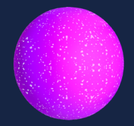
- Go supports pointers.
- Pointers are used in passing variables as parameter to a function.

- **&** -> pointer to a variable that points its address.
- ***** -> value of variable pointed by pointer

Call by value

WLUG
COMMUNITY | KNOWLEDGE | SHARE

- Value of actual parameters is copied into functions formal parameter.
- Any changes made inside function do not impact the actual parameters.



Call by reference

- Both the actual and formal parameters refer to the same location.
- So, changes made inside function are actually reflected in the actual parameters.

Multiple return types

WLUG
COMMUNITY | KNOWLEDGE | SHARE

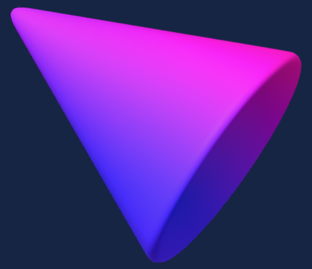
- A function in go can return more than one value of same or different data types

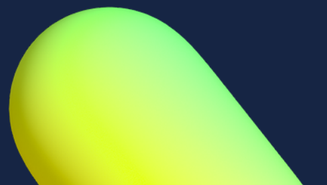
```
func function-name(parameters) (return-types){
            //function body
}
```

Custom type declaration

WLUG
COMMUNITY | KNOWLEDGE | SHARE

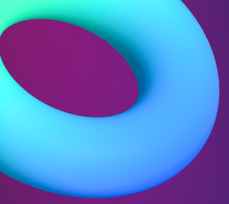
- To define a custom type declaration, we use the type keyword.
- Example: struct, []string, []bytes

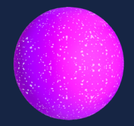
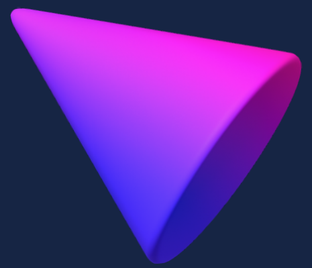
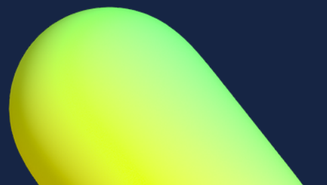




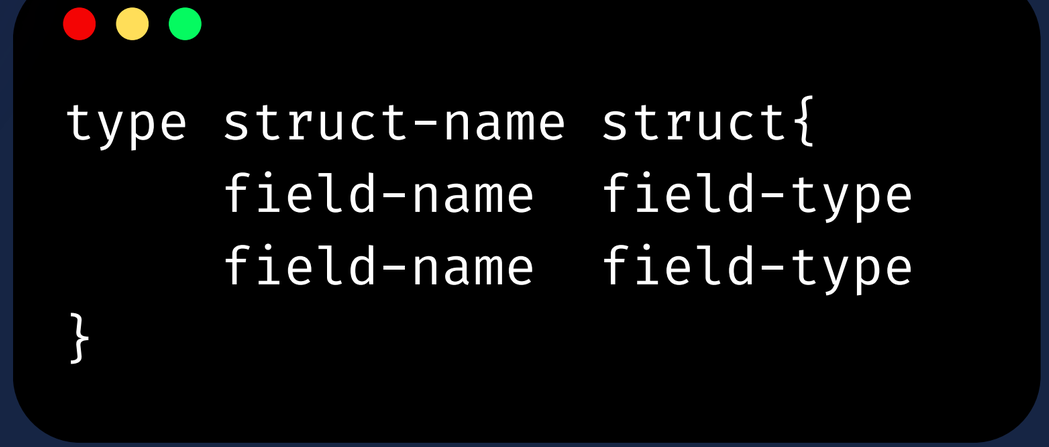
structs

- A struct is a user defined data type that consists of one or more variables.
- It is like a collection of variables, called as fields.
- Fields can have different data types.

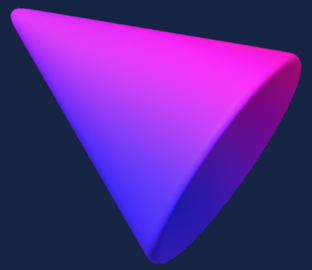


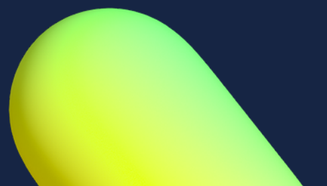




Defining structs



```
type struct-name struct{
     field-name  field-type
     field-name  field-type
}
```

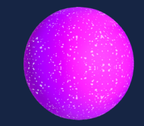




Initializing variables

WLUG
COMMUNITY | KNOWLEDGE | SHARE

```
type Person struct {
    Name string
    Age int
}
item1 := Person{"Rahul",49}
item2 := Person{Age: 49,Name: "Rahul"}
item3 := Person{Name: "Rahul"}
item3.Age=49
item4 := Person{}
item4.Name="Rahul"
item4.Age=49
```



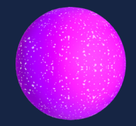
Receiver Function

WLUG
COMMUNITY | KNOWLEDGE | SHARE

- Used for passing custom type declarations as parameter to the function.

Declaration syntax

```
func (parameters) function-name (parameters) return-type{
                //function body
}
```



Functions VS Receiver Function

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Declaration

- Function has parameter after function name
- Receiver function has parameters both before and after function name

Calling

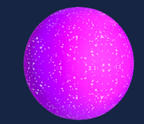
- Function-name(parameter)
- Parameter.reciever-function-name()

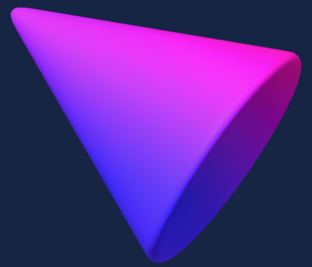
Analogy with C++ class methods

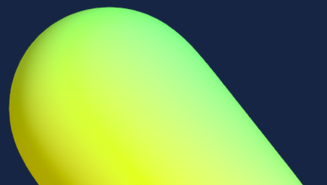
WLUG
COMMUNITY | KNOWLEDGE | SHARE

```
class Person{
   void print(){
      cout<<”hello”<<name;
   }
};

P=Person();
P.print() ->same as receiver function
```



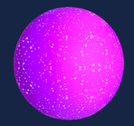


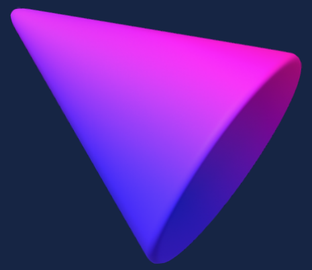


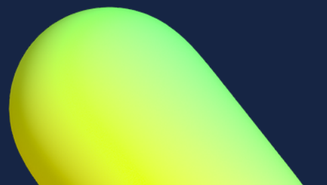
Error handling

WLUG
COMMUNITY | KNOWLEDGE | SHARE

- Errors are a part of any program.
- An error tells if something unexpected happens.
- Errors also help maintain code stability and maintainability.
- Without errors, the programs we use today will be extremely buggy due to a lack of testing.







Creating error

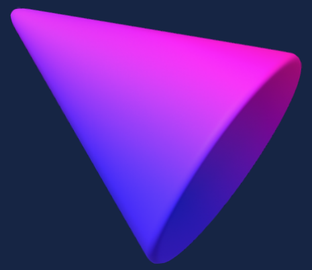
- GoLang errors package has a function called New() which can be used to create errors easily.

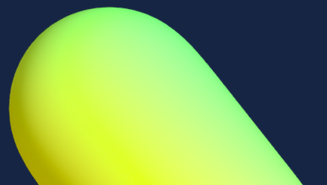
```
func returnError() error {
    return errors.New("This is an Error!")
}
```

Panic and Recover

WLUG
COMMUNITY | KNOWLEDGE | SHARE

- Panic occurs when an unexpected wrong thing happens. It stops the function execution.
- Recover is the opposite of it.
- It allows us to recover the execution from stopping.





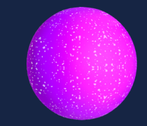
Defer

- defer keyword is used to delay the execution of a function or a statement until the nearby function returns. In simple words, defer will move the execution of the statement to the very end inside a function.

File handling

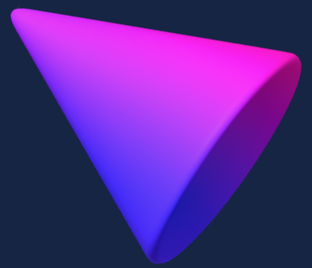
WLUG
COMMUNITY | KNOWLEDGE | SHARE

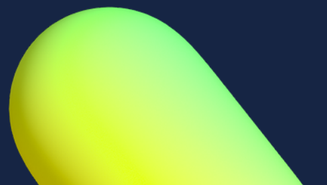
- In order to write something inside a file, we must create the file for writing. We can use the os package Create() function to open the file.
- We also must make sure the file is closed after the operation is done. So, we can use the defer keyword to send the function execution at last.
- To to open the file for reading. We can use the os package Open() function to open the file.



File handling

- Create
- Write
- Open
- Read
- Append





Testing

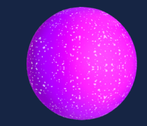
WLUG
COMMUNITY | KNOWLEDGE | SHARE

- We do not need any other dependencies for testing as go lang has its inbuild packages for testing.

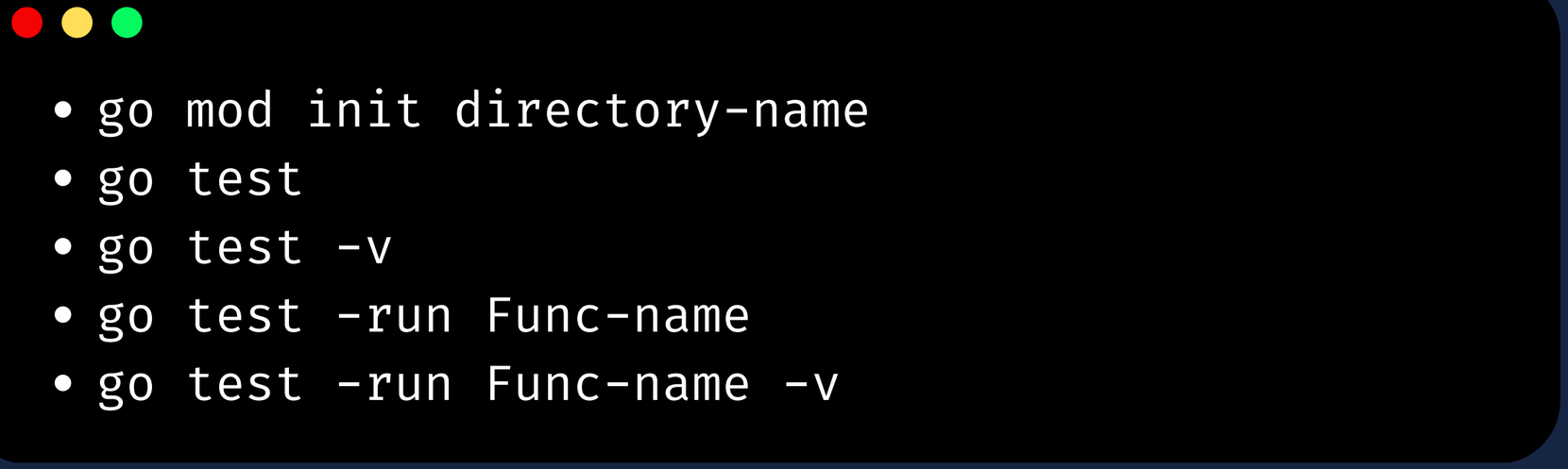
File Structure

```
--File.go
--File_test.go

  • Inside file_test.go we need to import
    “testing”
```



Commands



- go mod init directory-name
- go test
- go test -v
- go test -run Func-name
- go test -run Func-name -v

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Agendas

Following three topics would be discuss

Goal # 1

Interfaces

Goal # 2

Channels

go

Goal # 3

GoRoutines

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Let's Revise Functions

- Rules in Function?
- HINT: Related to Arguments

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Task Time.....

Function 1	Accepts an integer and prints it
Function 2	Accepts a string and prints it
Function 3	Accepts a Float and prints it

Problems Encountered

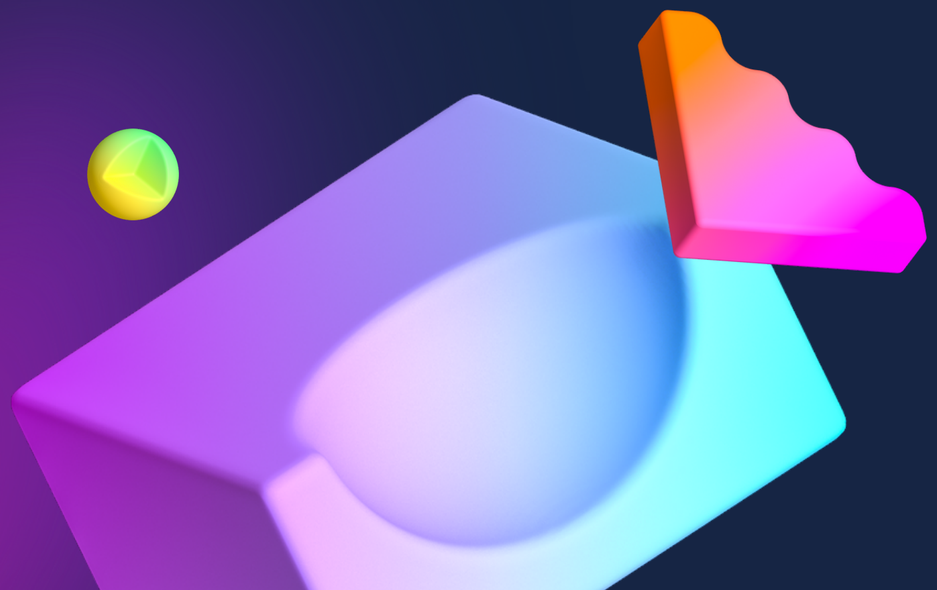
WLUG
COMMUNITY | KNOWLEDGE | SHARE

Functions — More number of Functions Required

Task — Task is to print value in each Function

Code — Code and most importantly Logic inside function is repeated

Conclude — Need for Reusability



META_GOLANG

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Reusability is the Key to any programming Language

Interfaces:The saviours

- An interface is a set of method signatures
- When a type provides definition for all the methods in the interface, it is said to implement the interface

Interface Syntax

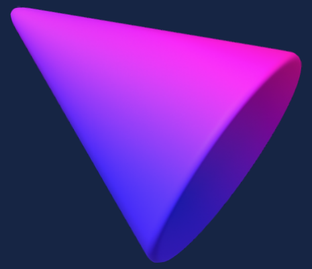
```
type interface_name interface{
  //Method Signatures
}
```

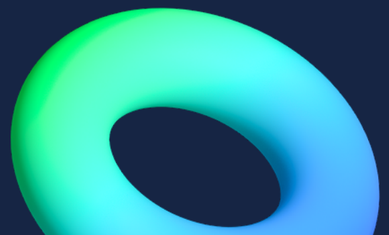
Lets code it!!

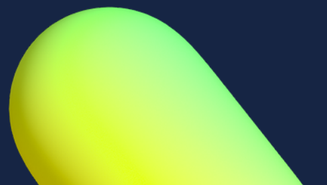
WLUG
COMMUNITY | KNOWLEDGE | SHARE

Types inside Interface

- If you are a type in this programme with a function name the same as functions inside the interface and return the same value as that of the function, then you are a member of the Interface



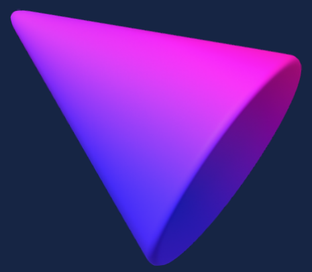
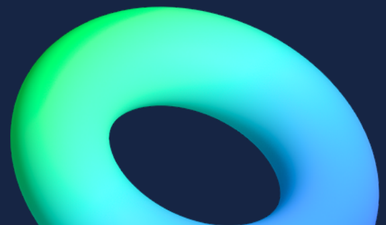
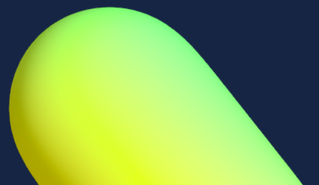




WLUG
COMMUNITY | KNOWLEDGE | SHARE

GoRoutines

- Used to handle concurrent Programmes
- Goroutines can be thought of as lightweight threads
- The cost of creating a Goroutine is tiny when compared to a thread. Hence Go applications mainly have thousands of Goroutines running concurrently

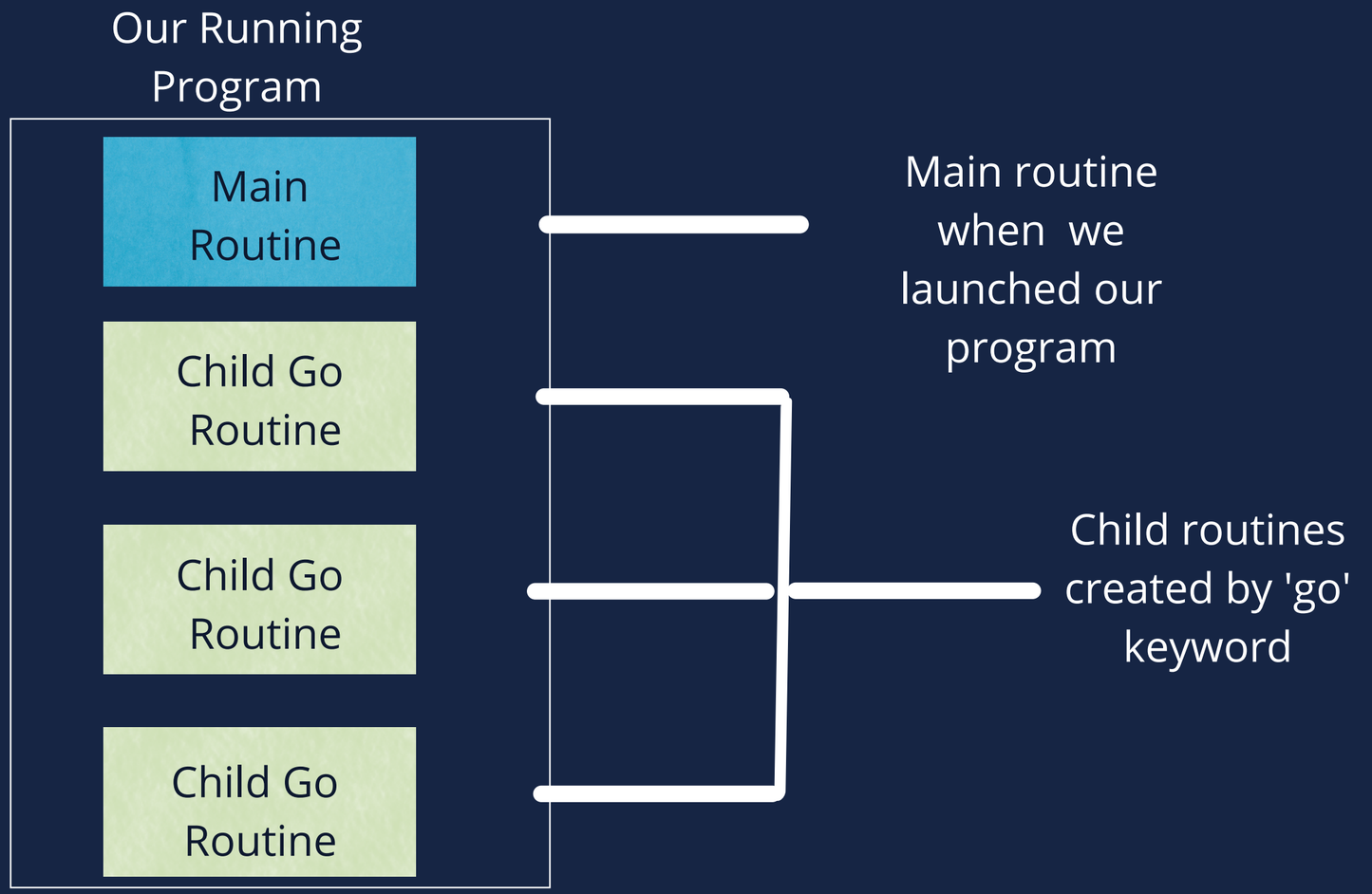




GoRoutines

- **Syntax:**

  Put a keyword go before calling the name of Function

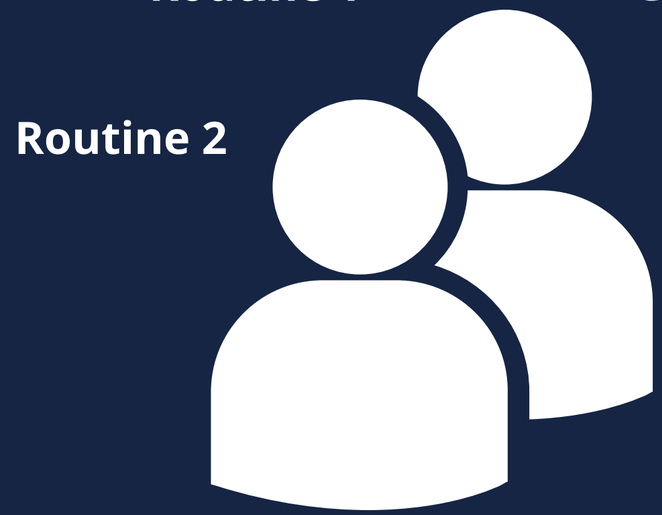
GoRoutines


Our Running Program
Main Routine
Child Go Routine
Child Go Routine
Child Go Routine
Main routine when we launched our program
Child routines created by 'go' keyword

GoRoutines

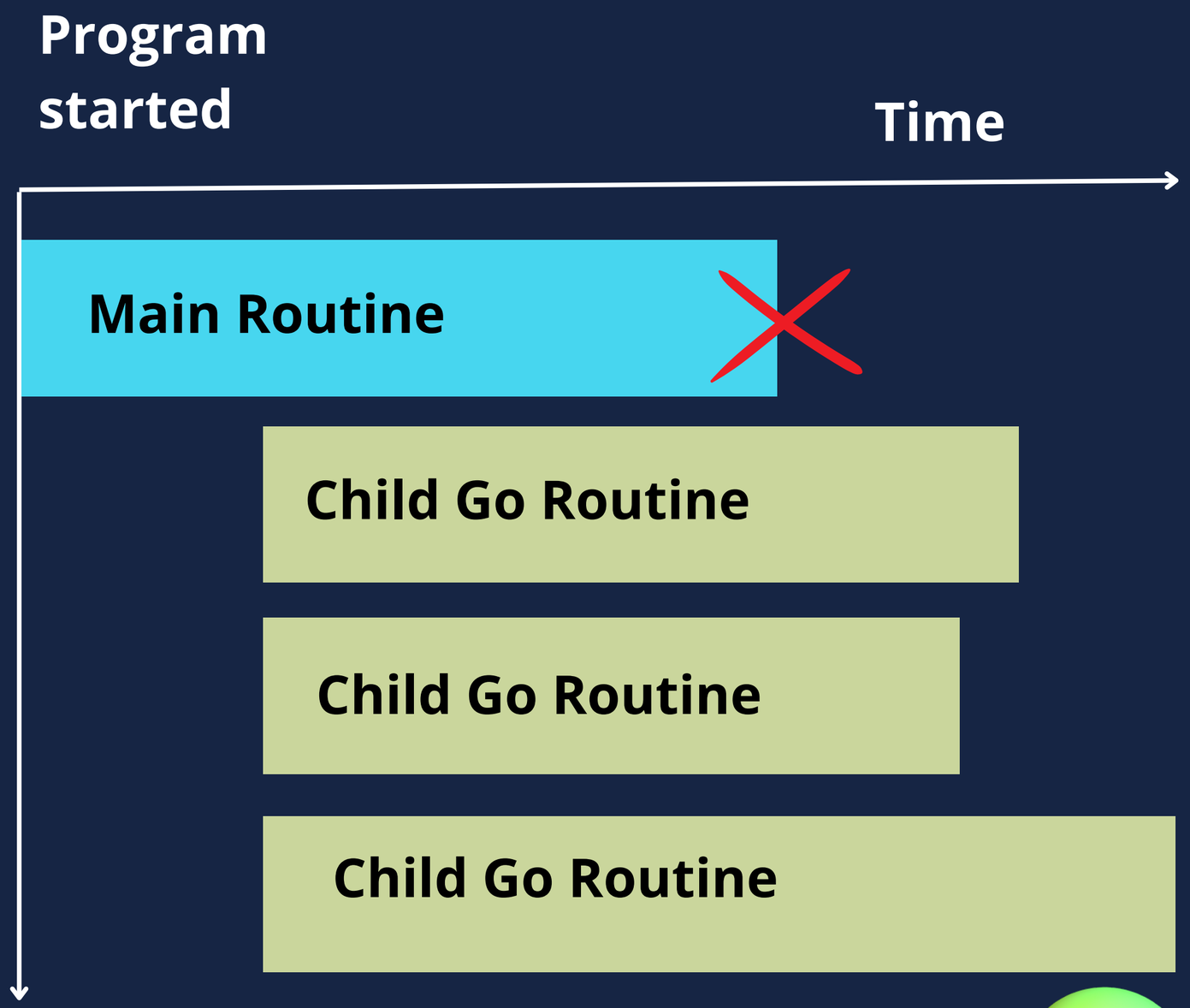
WLUG
COMMUNITY | KNOWLEDGE | SHARE

Solution

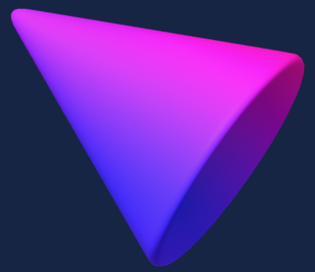


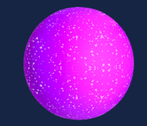
Let's code it!!

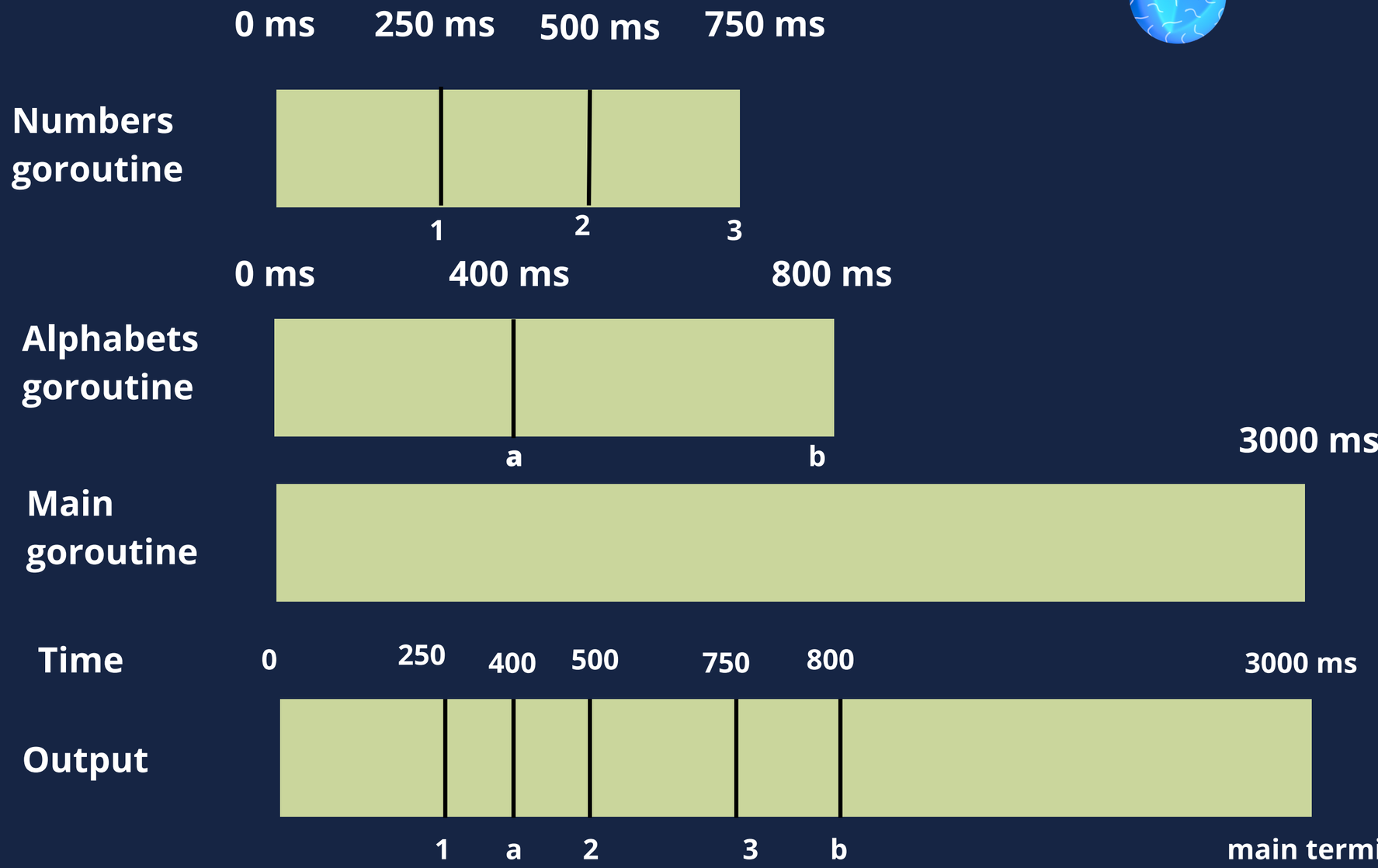
Issue


Program started
Time
Main Routine
Child Go Routine
Child Go Routine
Child Go Routine

Multiple GoRoutines



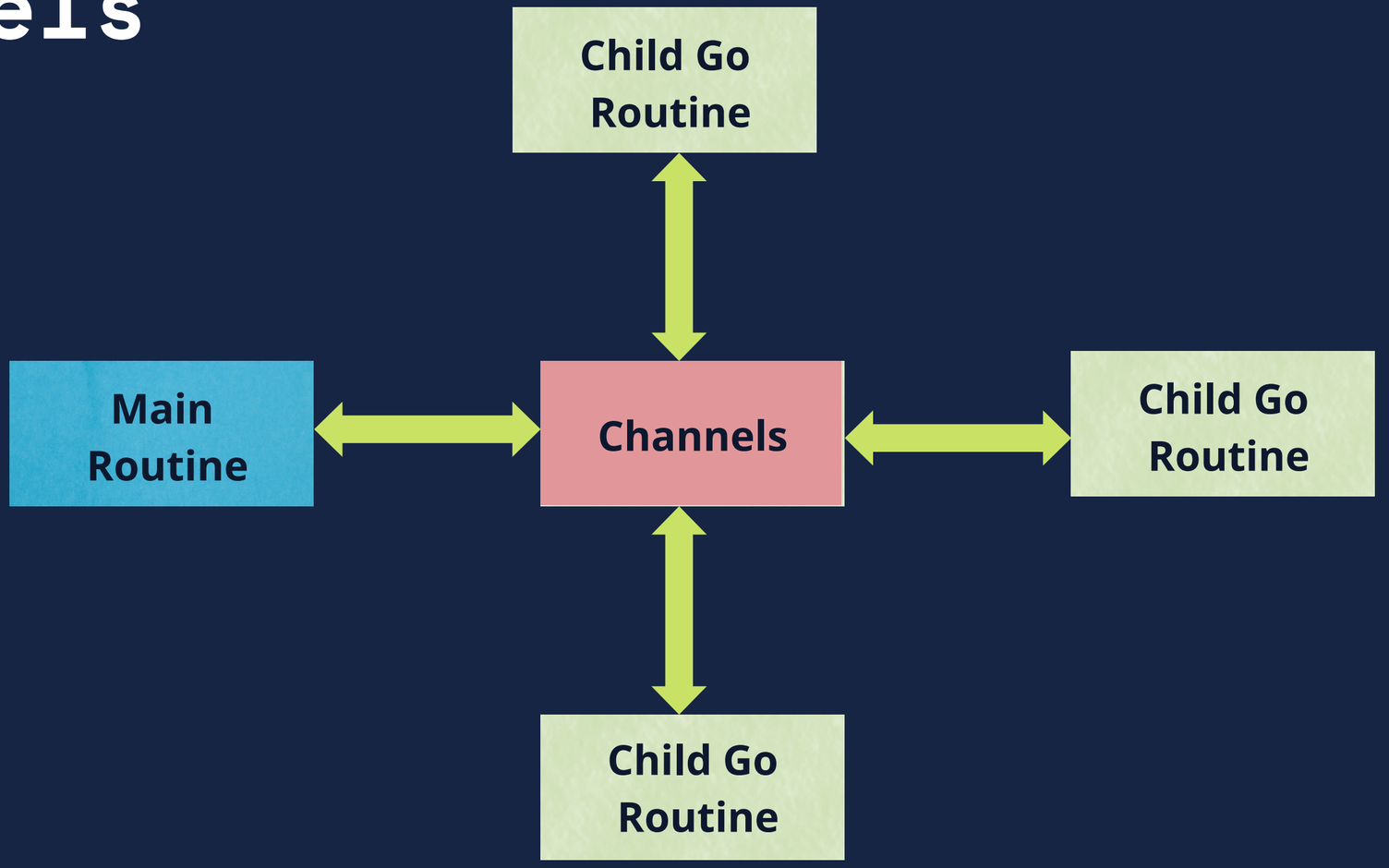




Channels

- Pipes using which GoRoutines communicate
- A programming construct that allows us to move data between different goroutines
- Similar to how water flows from one end to another in a pipe, data can be sent from one end and received from the other end using channels

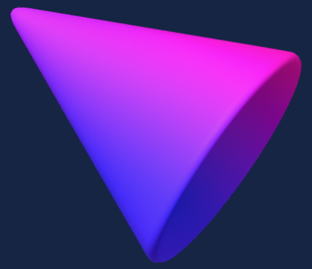
Channels


Child Go Routine
Main Routine
Channels
Child Go Routine
Child Go Routine

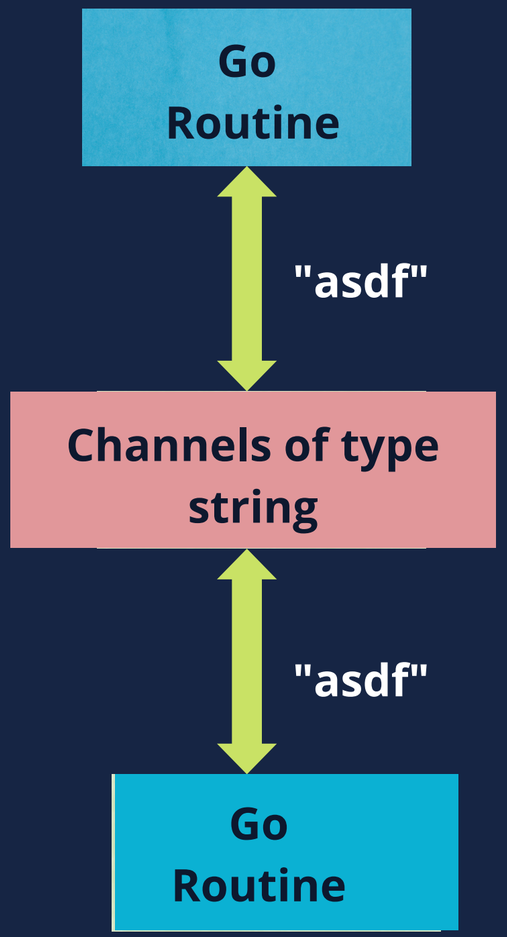
Channels

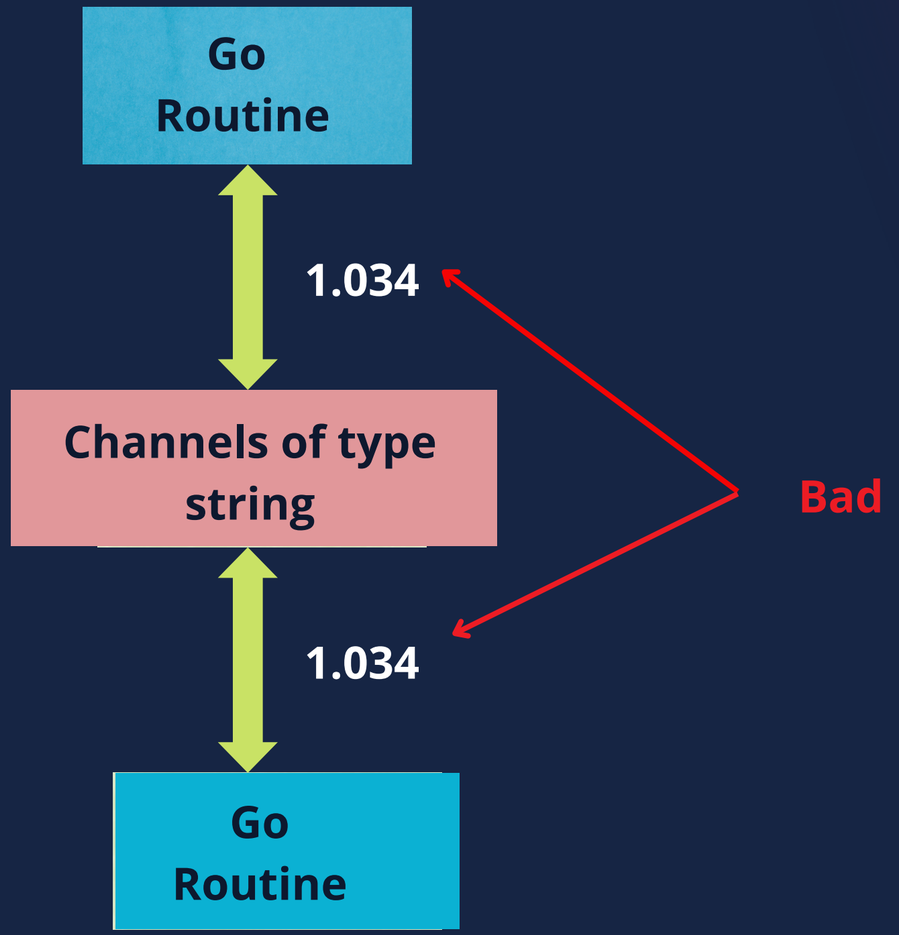
- Data can be sent from one end and received from the other end using channels

- Each channel has a type associated with it. This type is the type of data that the channel is allowed to transport



Channels


Go
Routine
"asdf"
Channels of type
string
"asdf"
Go
Routine


Go
Routine
1.034
Channels of type
string
Bad
1.034
Go
Routine

Channels: Syntax

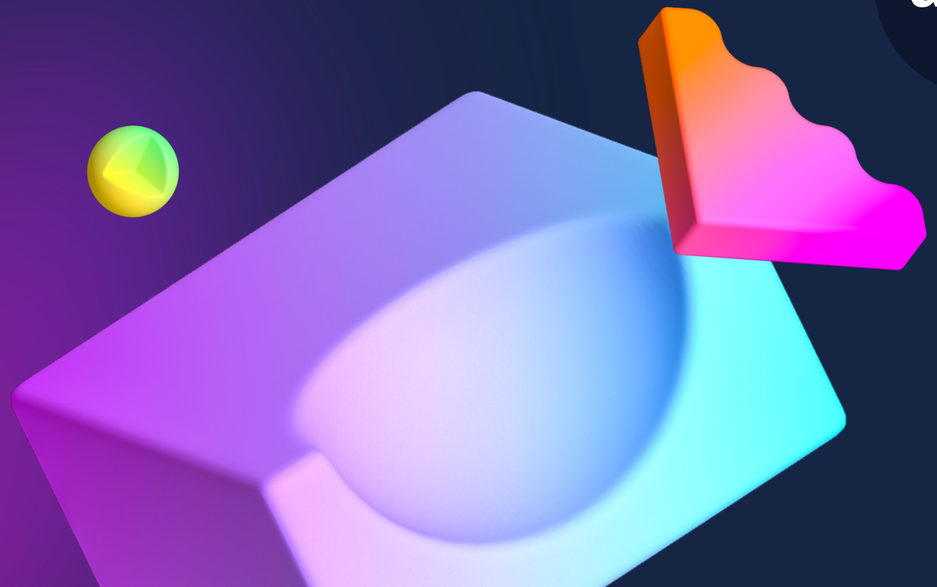
WLUG
COMMUNITY | KNOWLEDGE | SHARE

```
chan T
```

Channel of type T

```
a := make(chan int)
```

Defines an int channel



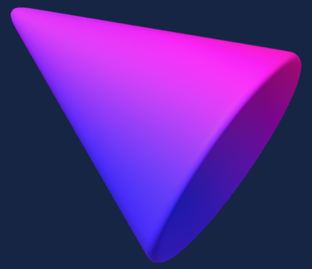
Sending, Receiving data

WLUG
COMMUNITY | KNOWLEDGE | SHARE

- Syntax:

  ```
  data := <- a // read from channel a
  a <- data // write to channel a
  ```

- The direction of the arrow with respect to the channel specifies whether the data is sent or received

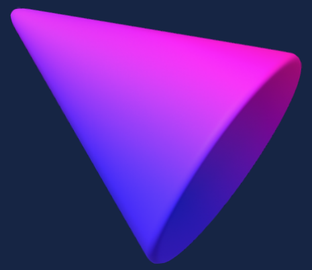


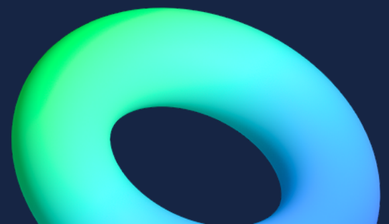
Let's Code It!!

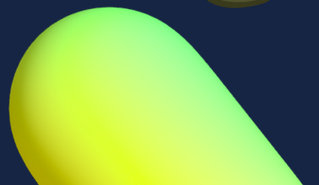
Channels

WLUG
COMMUNITY | KNOWLEDGE | SHARE

- Sends and receives are blocking by default
- A blocking statement is a statement which blocks the execution of further lines of code after it



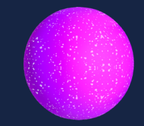


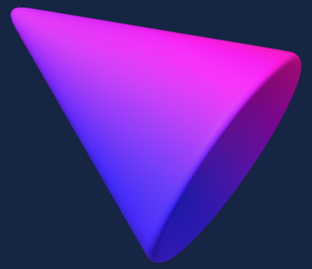


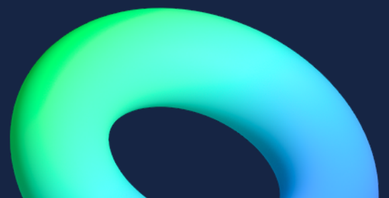
Channels: DeadLock

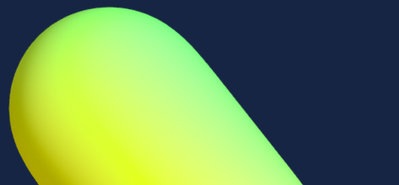
- If a Goroutine is sending data on a channel, then it is expected that some other Goroutine should be receiving the data

Let's Code!!







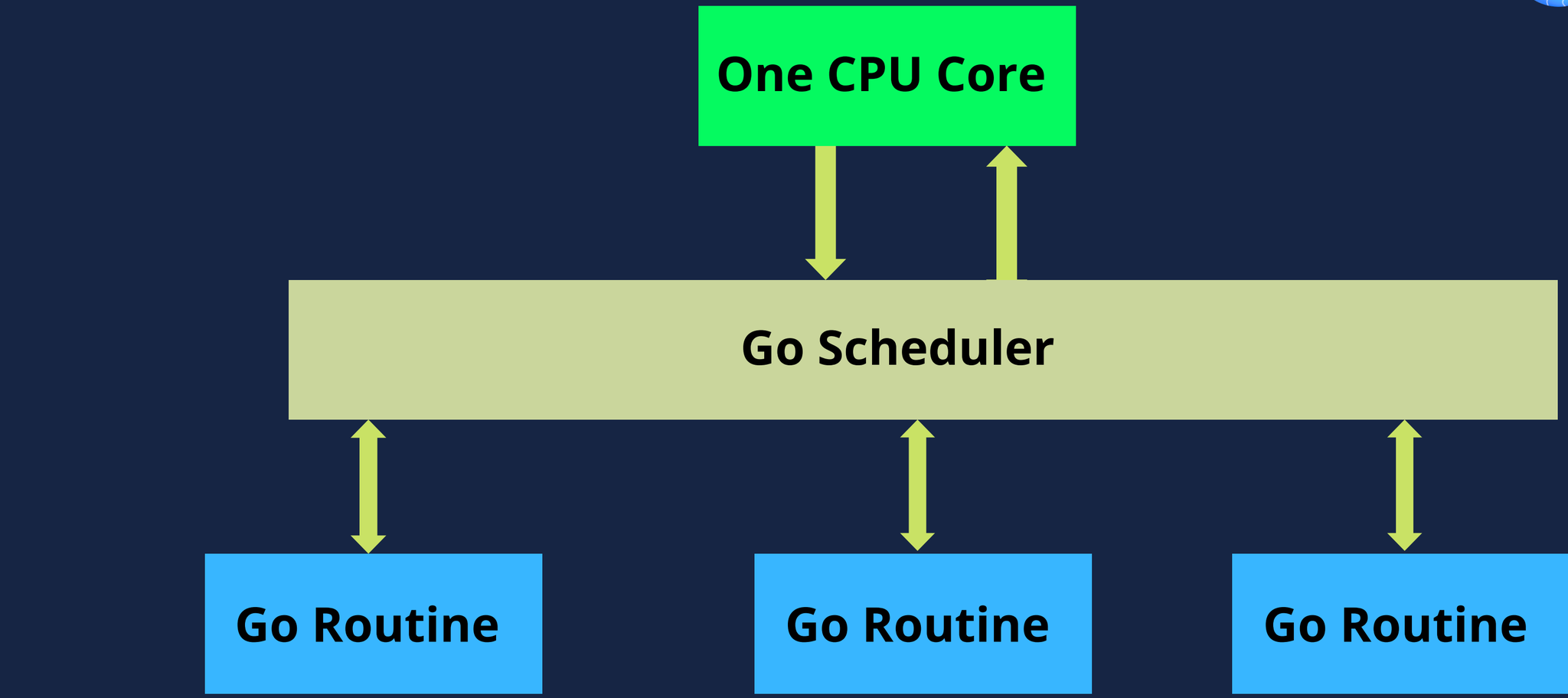


Concurrency Vs Parallelism

WLUG
COMMUNITY | KNOWLEDGE | SHARE

- Concurrency involves structuring a program so that two or more tasks may be in progress simultaneously

Concurrency

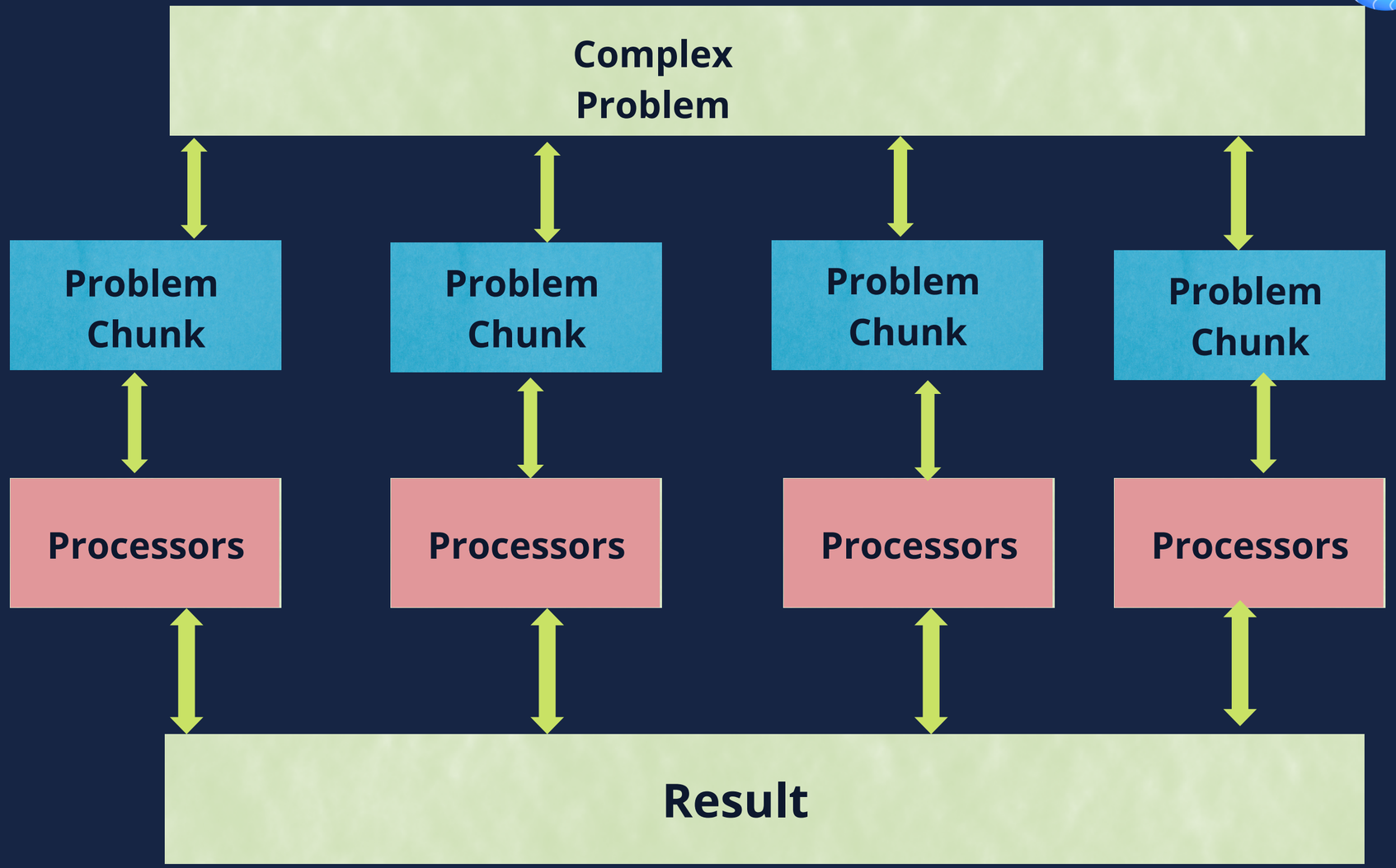


Parallelism

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Solution

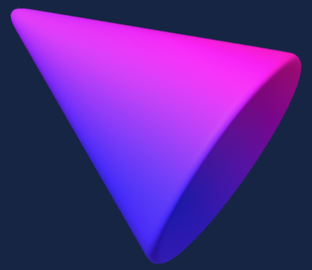
Concurrency Vs Parallelism

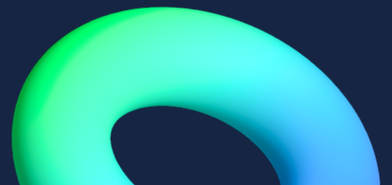


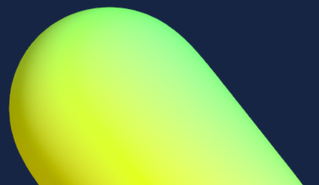
Parallelism

- Parallelism allows for two or more tasks to be executed simultaneously

- Parallelism requires more than one processor or thread, concurrency does not







WLUG
COMMUNITY | KNOWLEDGE | SHARE

Thank You!!