We know that...

Every value has a type

Every function has to specify the type of its arguments

So does that mean...

Every function we ever write has to be rewritten to accommodate different types even if the logic in it is identical?

func (d deck) **shuffle**()

Can only shuffle a value of type 'deck'

func (s []float64) **shuffle**()

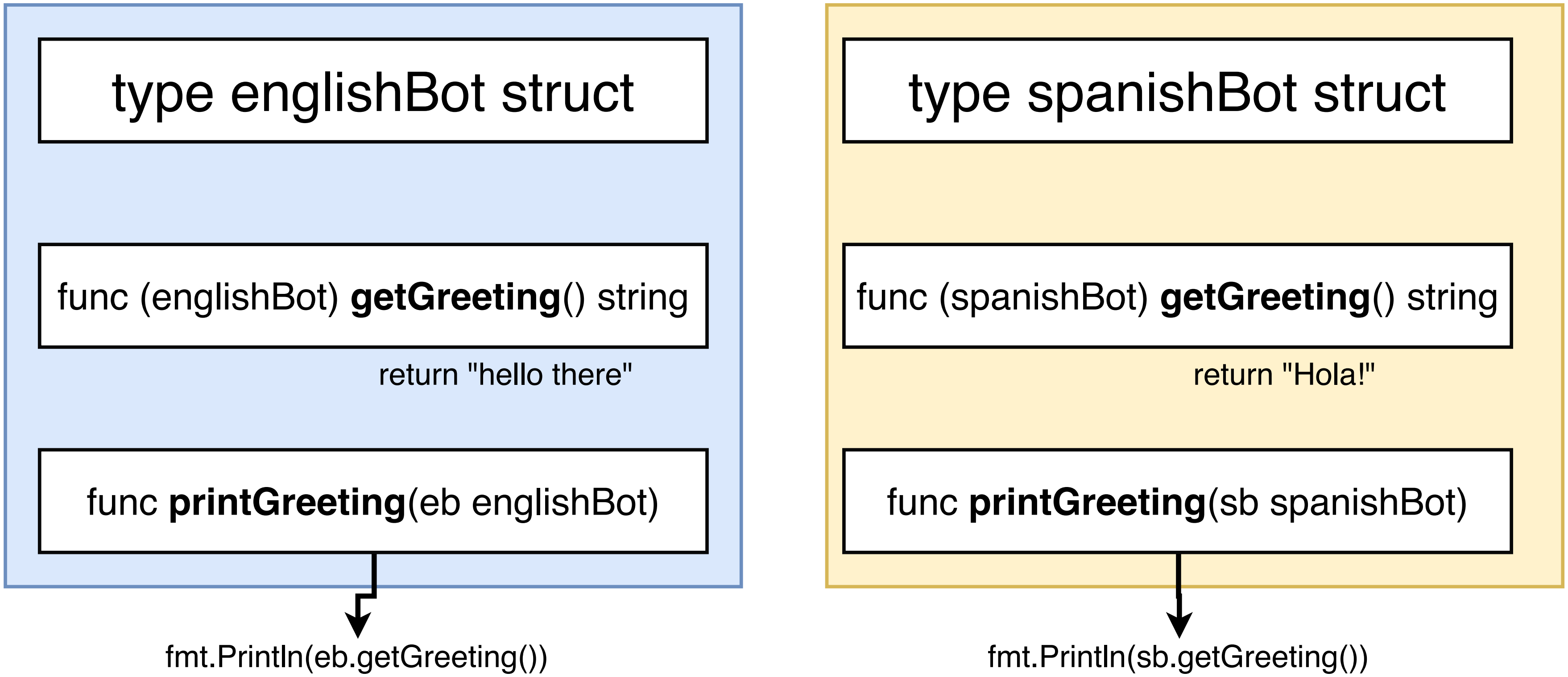
Can only shuffle a value of type '[]float64'

func (s []string) **shuffle**()

Can only shuffle a value of type '[]string'

func (s []int) **shuffle**()

Can only shuffle a value of type '[]int'



type englishBot struct

func (englishBot) **getGreeting**() string

return "hello there"

func **printGreeting**(eb englishBot)

fmt.Println(eb.getGreeting())

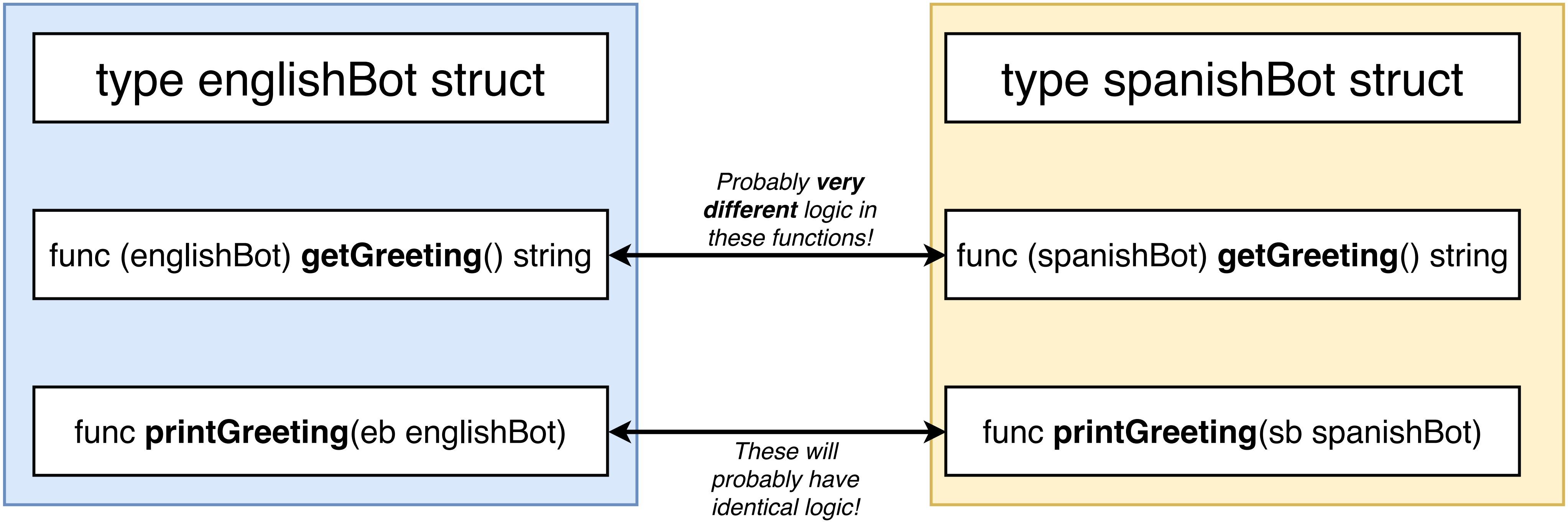
type spanishBot struct

func (spanishBot) **getGreeting**() string

return "Hola!"

func **printGreeting**(sb spanishBot)

fmt.Println(sb.getGreeting())



type englishBot struct

func (englishBot) **getGreeting**() string

func **printGreeting**(eb englishBot)

*Probably **very different** logic in these functions!*

These will probably have identical logic!

type spanishBot struct

func (spanishBot) **getGreeting**() string

func **printGreeting**(sb spanishBot)

type englishBot struct

func (englishBot) **getGreeting**() string

type spanishBot struct

func (spanishBot) **getGreeting**() string

To whom it may concern...

type bot interface

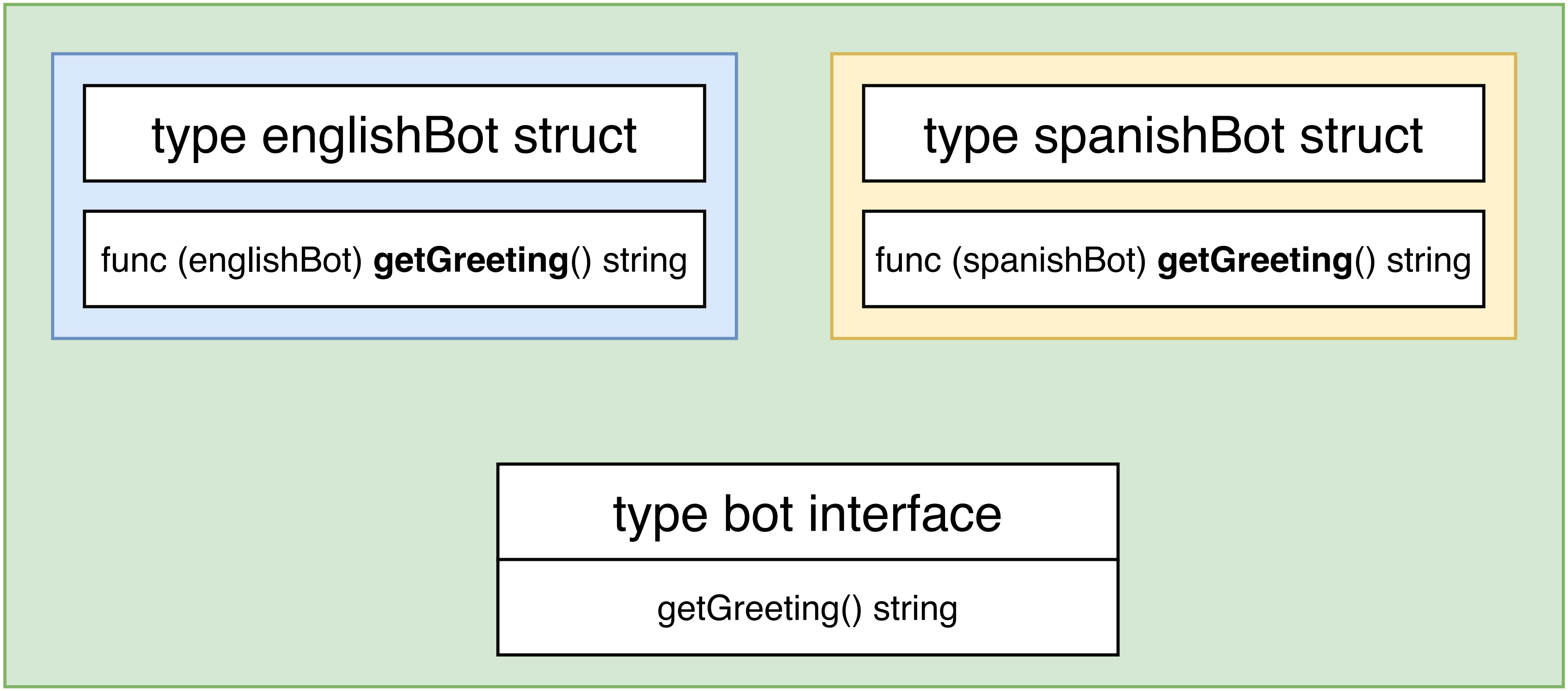
Our program has a new type called 'bot'

getGreeting() string

If you are a type in this program with a function called 'getGreeting' and you return a string then you are now an honorary member of type 'bot'

Now that you're also an honorary member of type 'bot', you can now call this function called 'printGreeting'

func printGreeting(b bot)



type englishBot struct

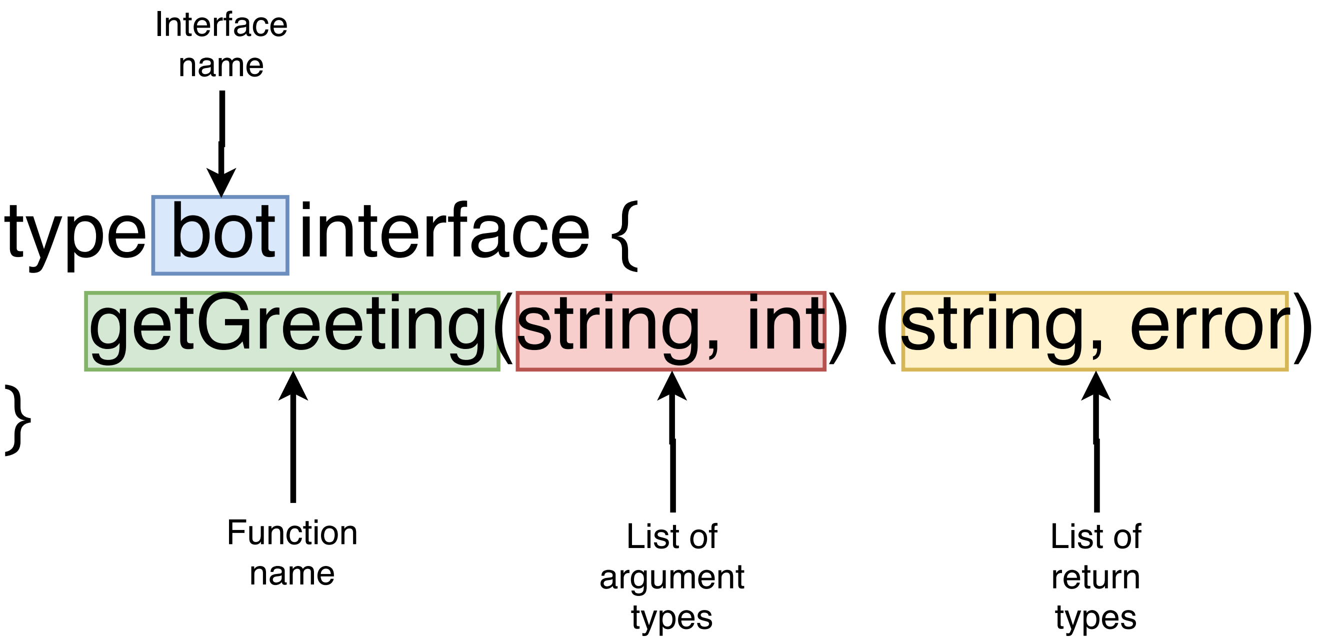
func (englishBot) **getGreeting**() string

type spanishBot struct

func (spanishBot) **getGreeting**() string

type bot interface

getGreeting() string



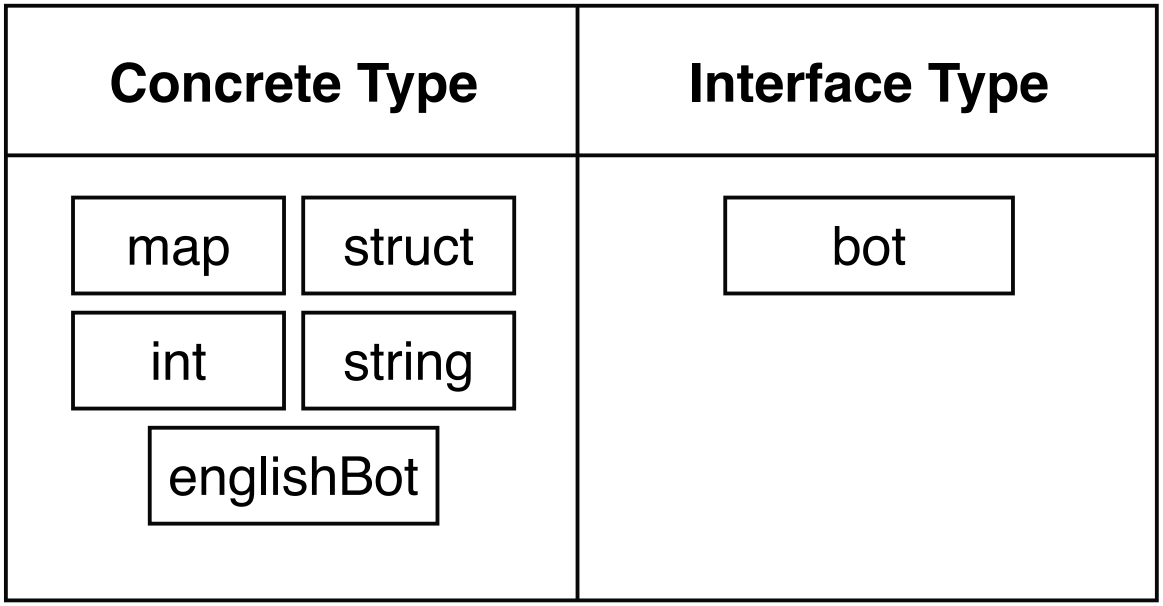
```
type bot interface {
    getGreeting(string, int) (string, error)
}
```

Interface name

Function name

List of argument types

List of return types



Concrete Type	Interface Type
map struct int string englishBot	bot

Interfaces are **not** generic types

Other languages have 'generic' types - go (famously) does not.

Interfaces are 'implicit'

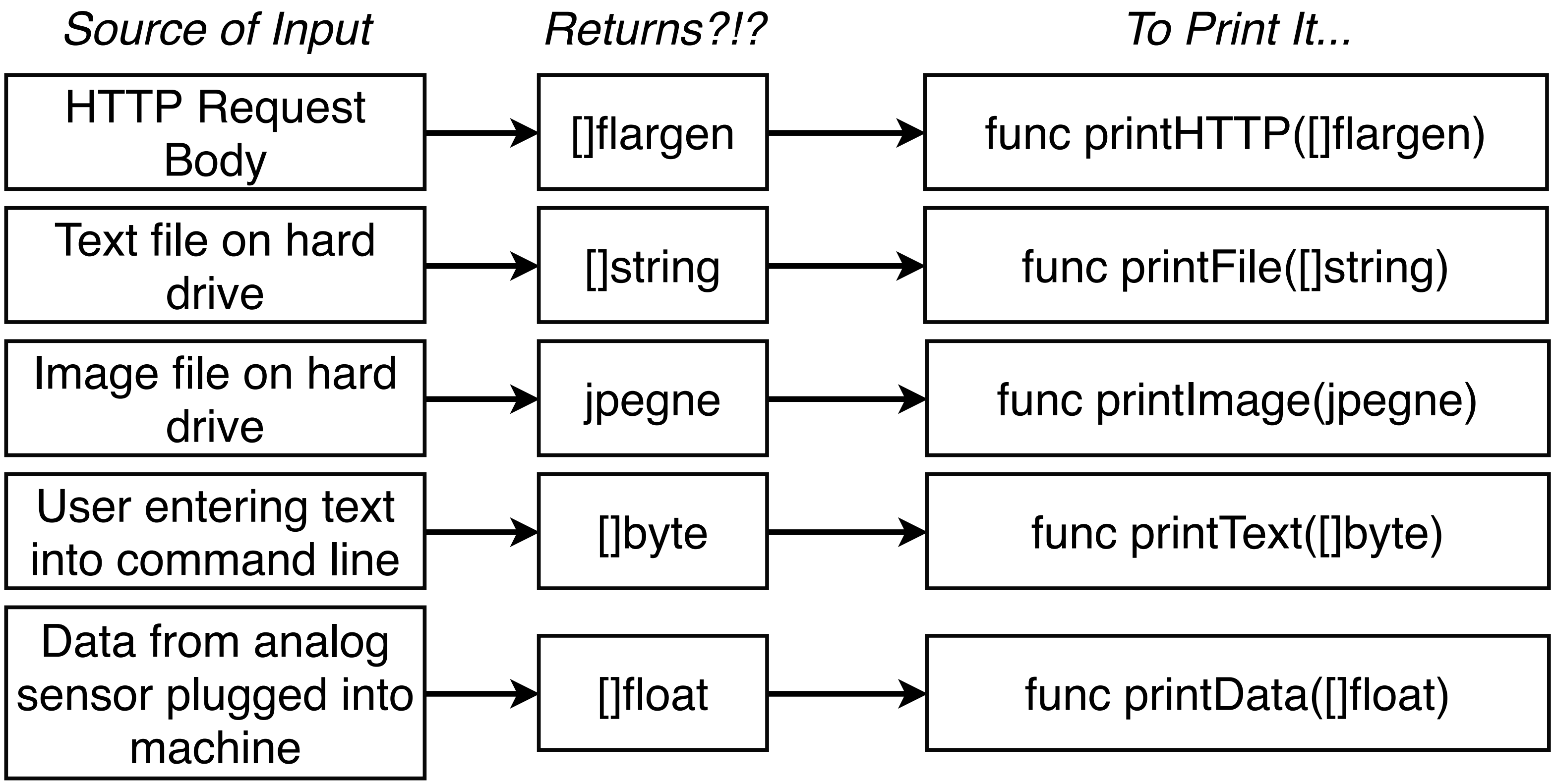
We don't manually have to say that our custom type satisfies some interface.

Interfaces are a contract to help us manage types

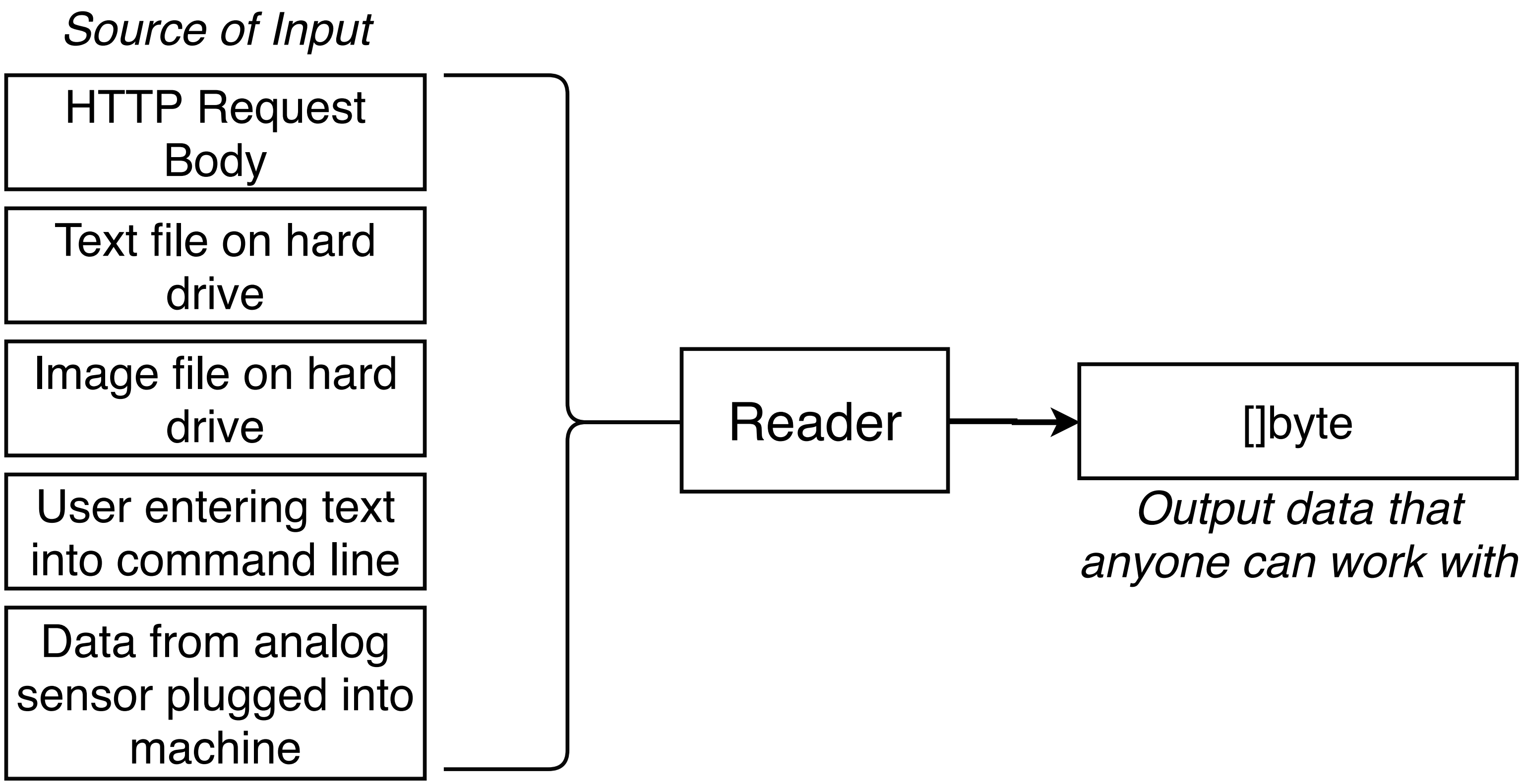
GARBAGE IN -> GARBAGE OUT. If our custom type's implementation of a function is broken then interfaces wont help us!

Interfaces are tough. Step #1 is understanding how to read them

Understand how to read interfaces in the standard lib. Writing your own interfaces is tough and requires experience



Source of Input	*Returns?!?*	*To Print It...*
HTTP Request Body	[]flargen	func printHTTP([]flargen)
Text file on hard drive	[]string	func printFile([]string)
Image file on hard drive	jpegne	func printImage(jpegne)
User entering text into command line	[]byte	func printText([]byte)
Data from analog sensor plugged into machine	[]float	func printData([]float)



Source of Input

HTTP Request Body

Text file on hard drive

Image file on hard drive

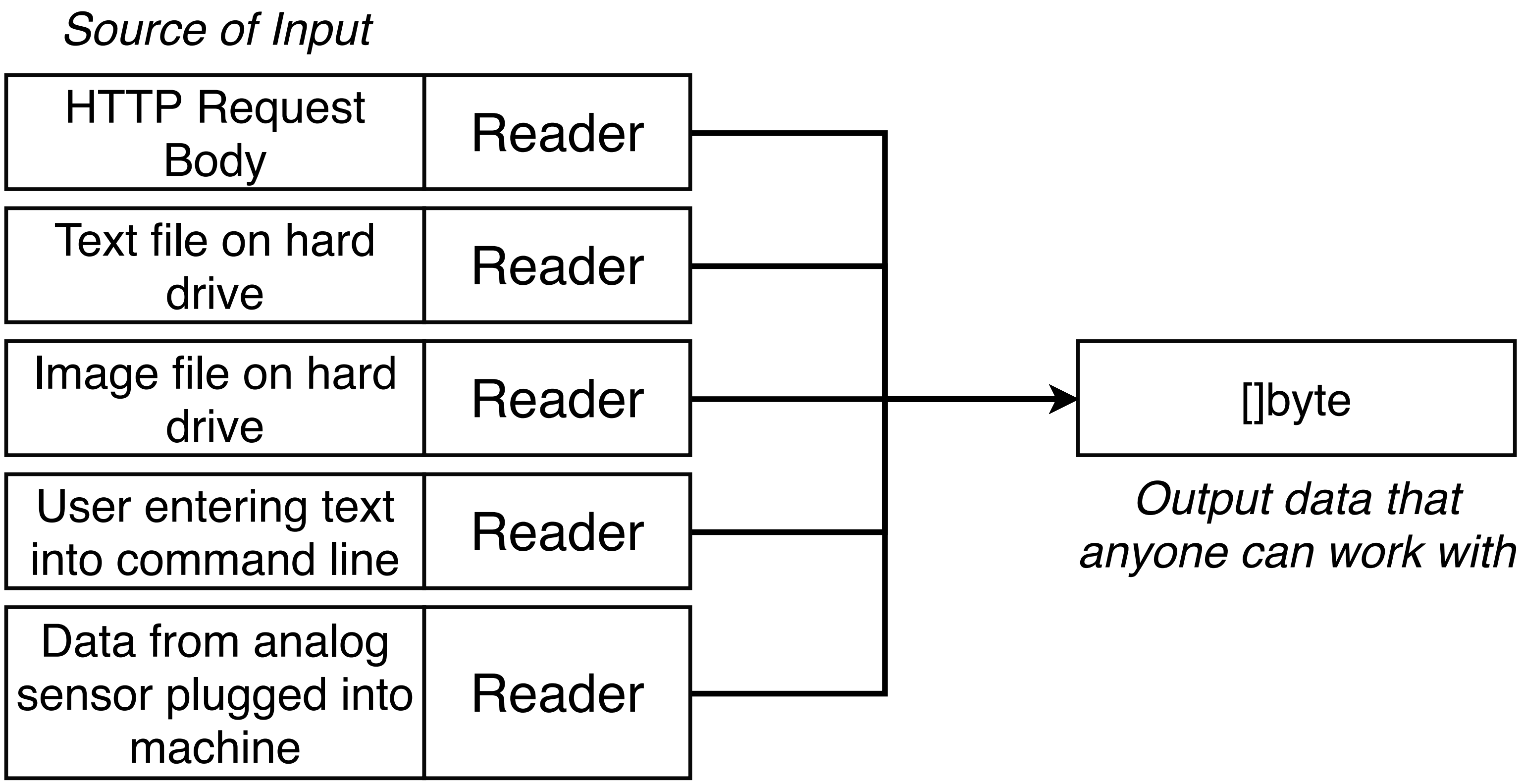
User entering text into command line

Data from analog sensor plugged into machine

Reader

[]byte

Output data that anyone can work with



Source of Input

HTTP Request Body — Reader

Text file on hard drive — Reader

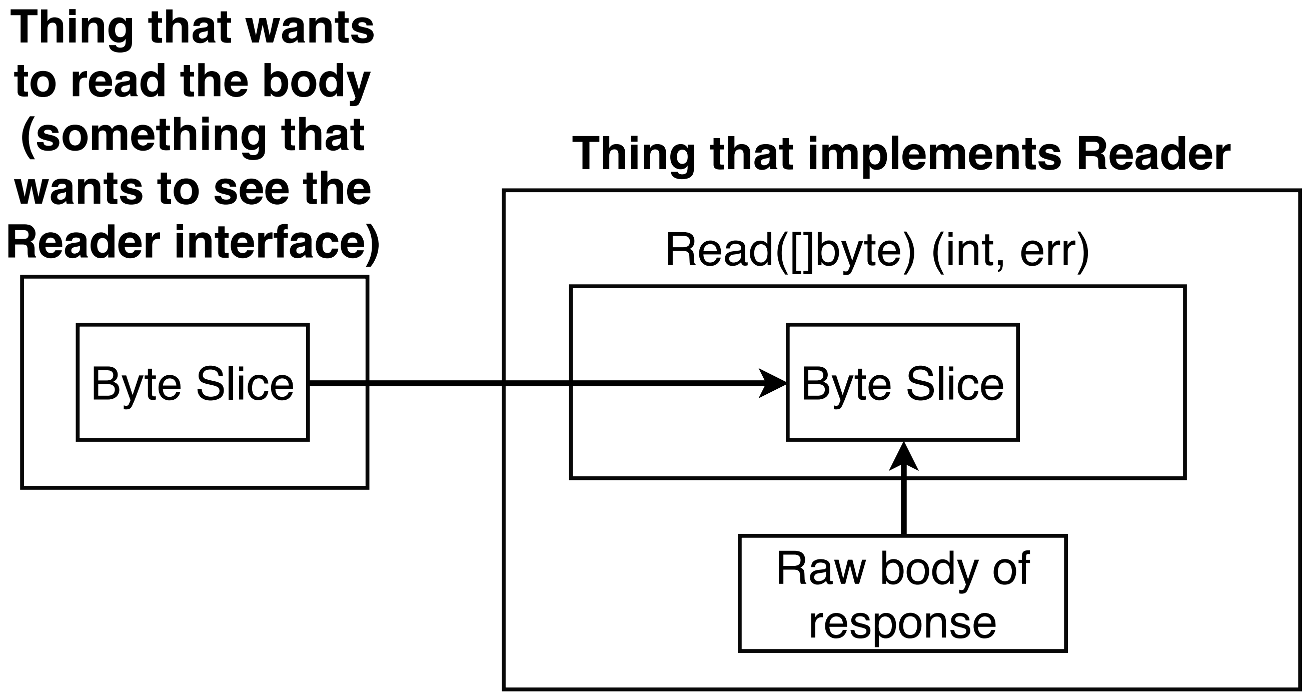
Image file on hard drive — Reader

User entering text into command line — Reader

Data from analog sensor plugged into machine — Reader

[]byte

Output data that anyone can work with



Thing that wants to read the body (something that wants to see the Reader interface)

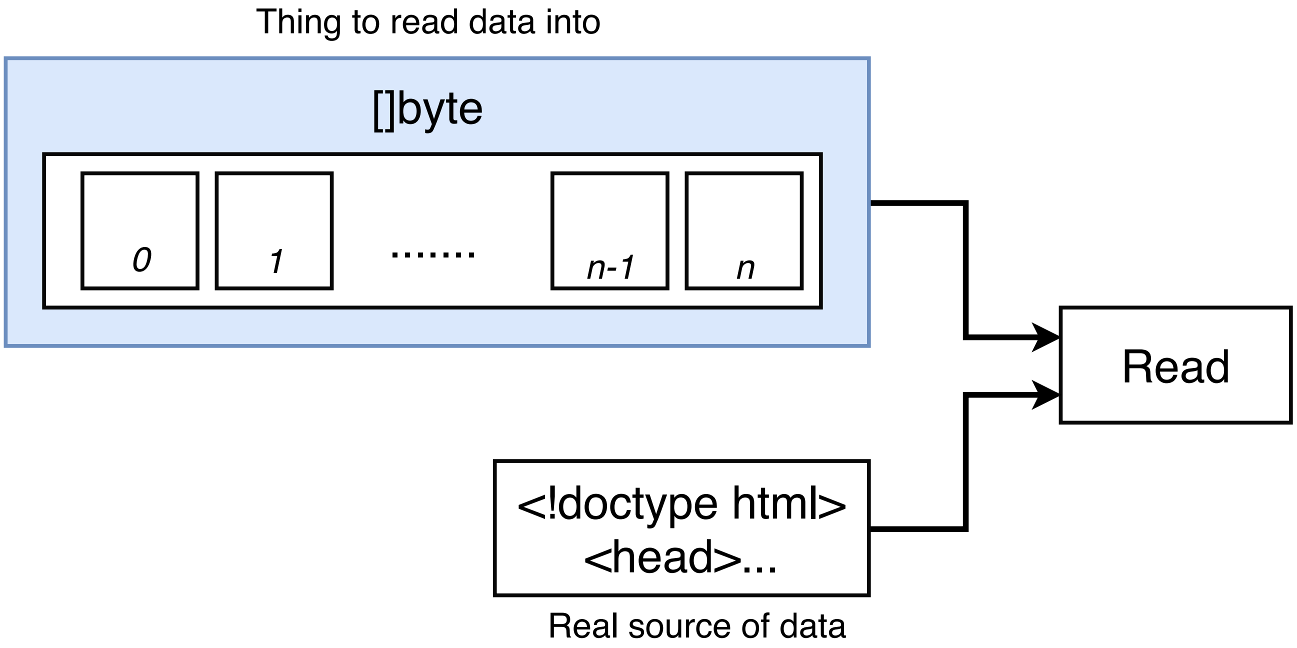
Thing that implements Reader

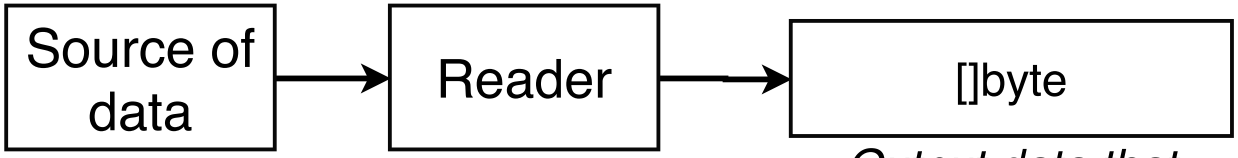
Read([]byte) (int, err)

Byte Slice

Byte Slice

Raw body of response


Thing to read data into
[]byte
0
1
....
n-1
n
Read
<!doctype html>
<head>...
Real source of data

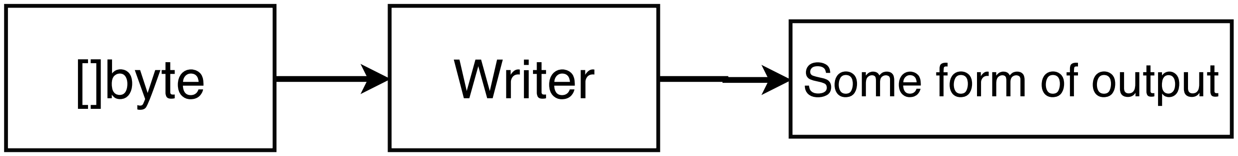


Source of data

Reader

[]byte

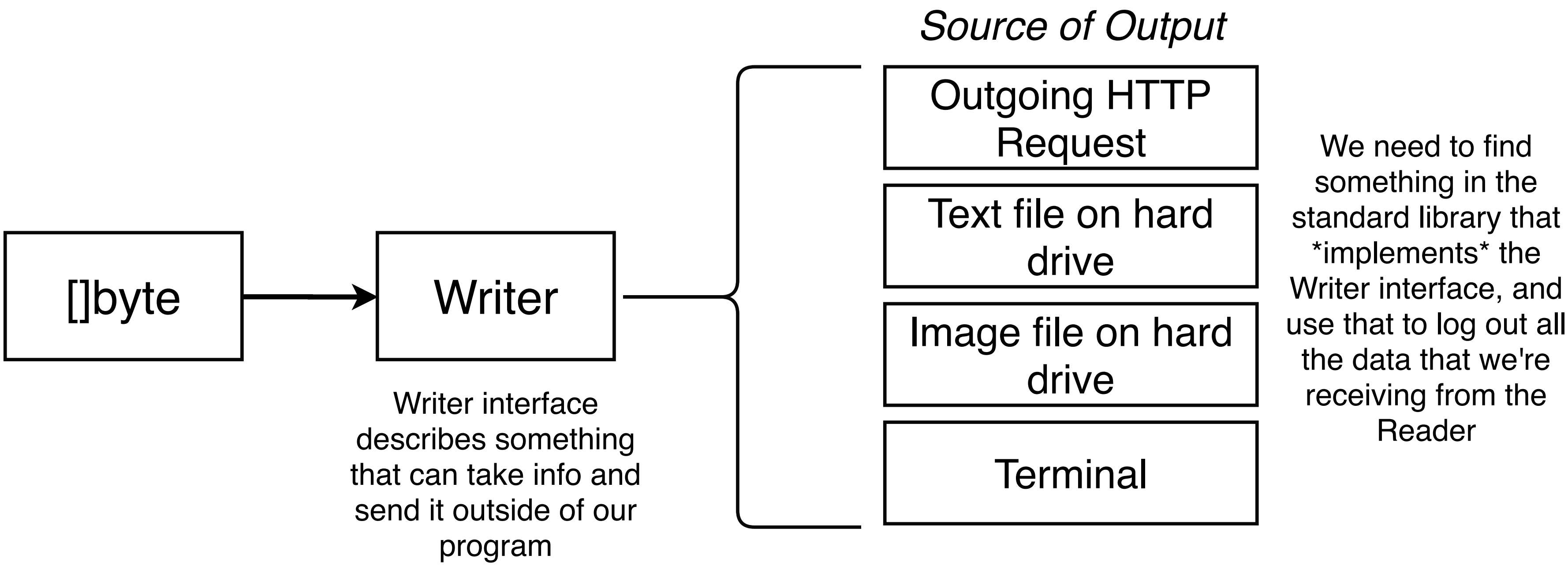
Output data that anyone can work with



[]byte

Writer

Some form of output



[]byte

Writer

Writer interface describes something that can take info and send it outside of our program

Source of Output

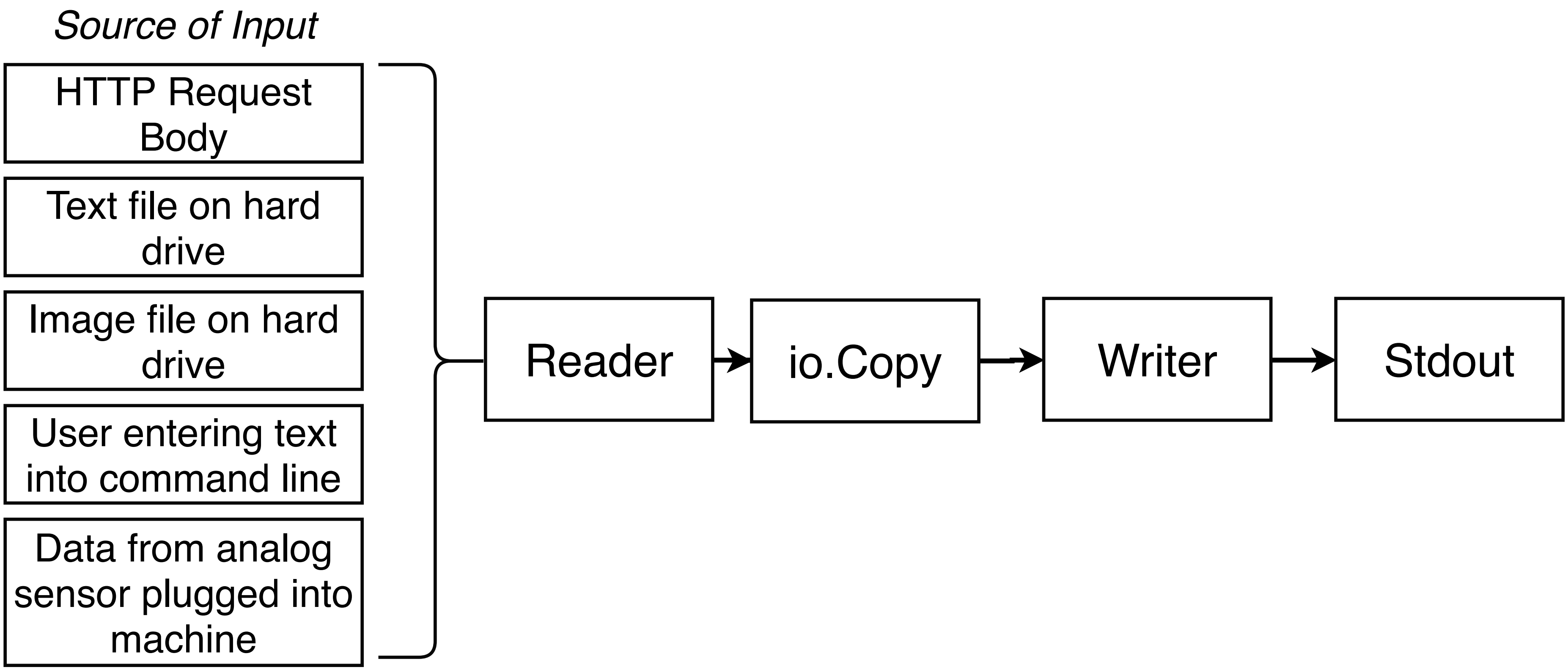
Outgoing HTTP Request

Text file on hard drive

Image file on hard drive

Terminal

We need to find something in the standard library that *implements* the Writer interface, and use that to log out all the data that we're receiving from the Reader



Source of Input

HTTP Request Body

Text file on hard drive

Image file on hard drive

User entering text into command line

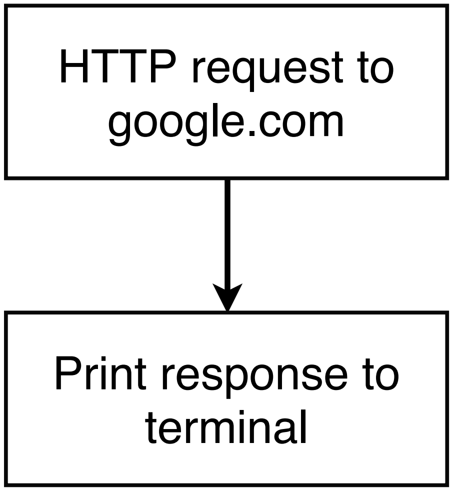
Data from analog sensor plugged into machine

Reader

io.Copy

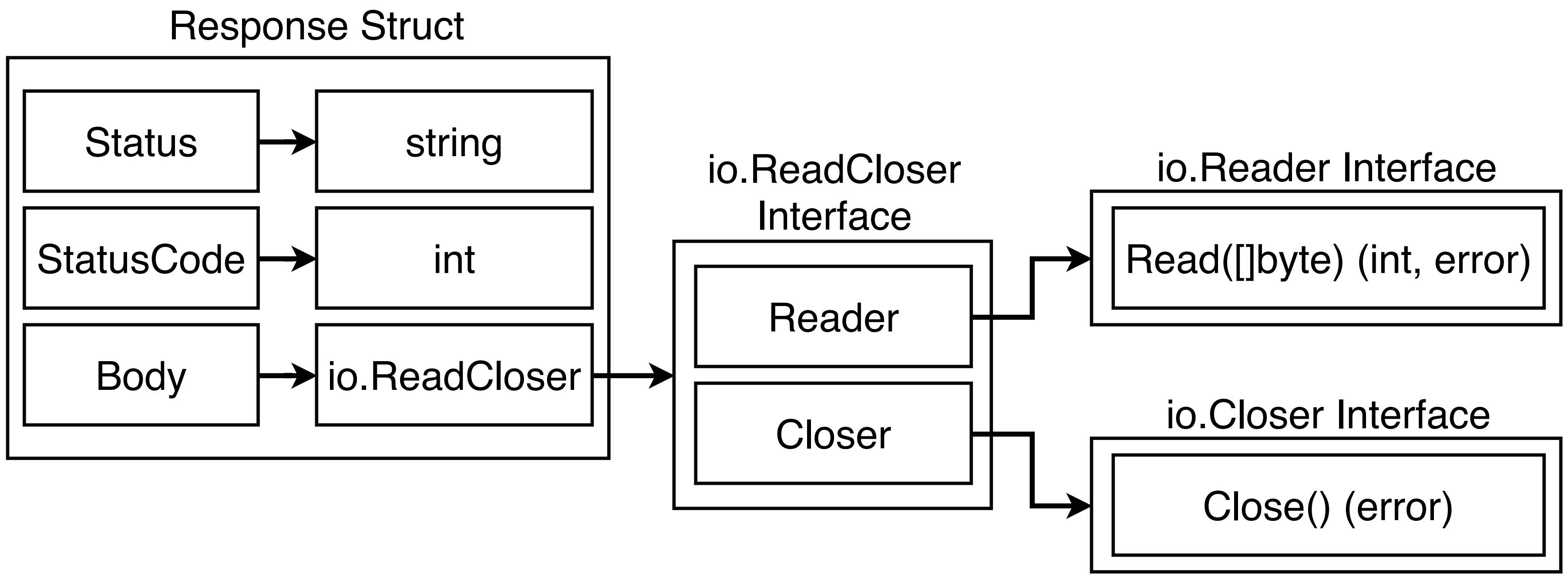
Writer

Stdout



HTTP request to google.com

Print response to terminal



Response Struct

Status → string

StatusCode → int

Body → io.ReadCloser

io.ReadCloser Interface

Reader

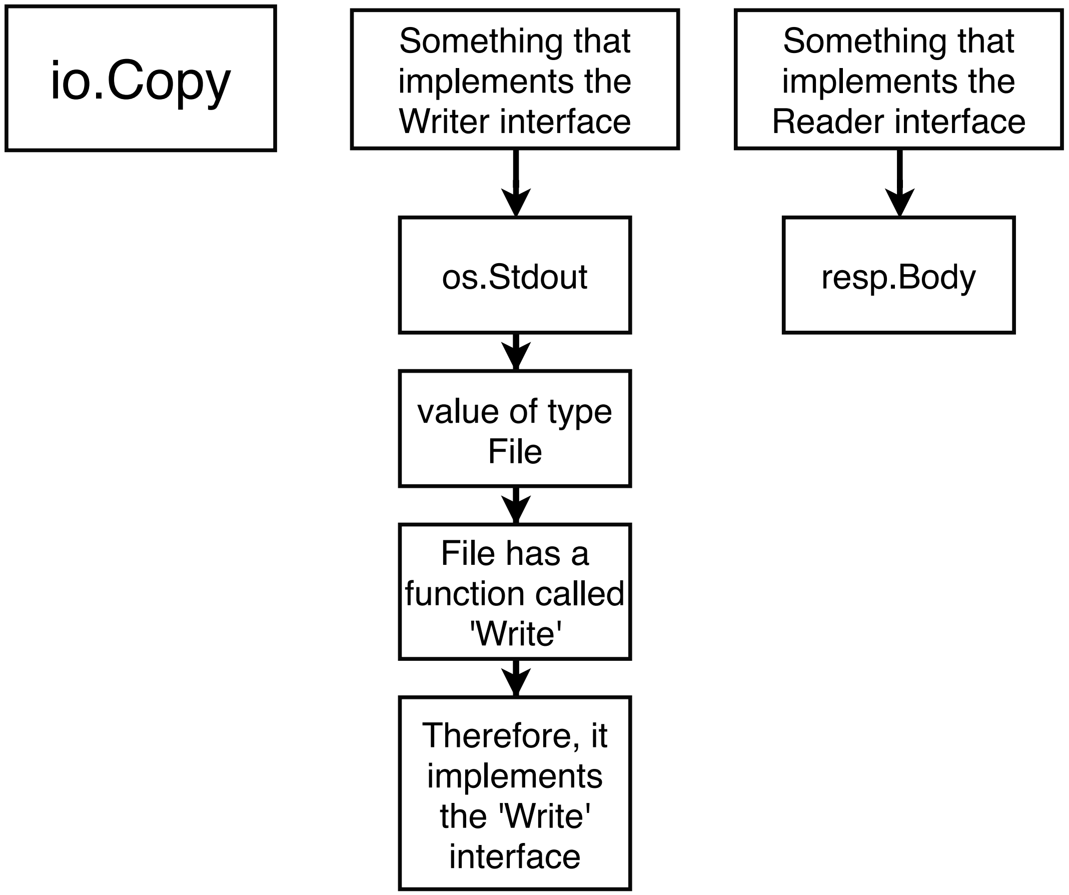
Closer

io.Reader Interface

Read([]byte) (int, error)

io.Closer Interface

Close() (error)



io.Copy

Something that implements the Writer interface

os.Stdout

value of type File

File has a function called 'Write'

Therefore, it implements the 'Write' interface

Something that implements the Reader interface

resp.Body

Assignment

Write a program that creates two custom struct types called 'triangle' and 'square'

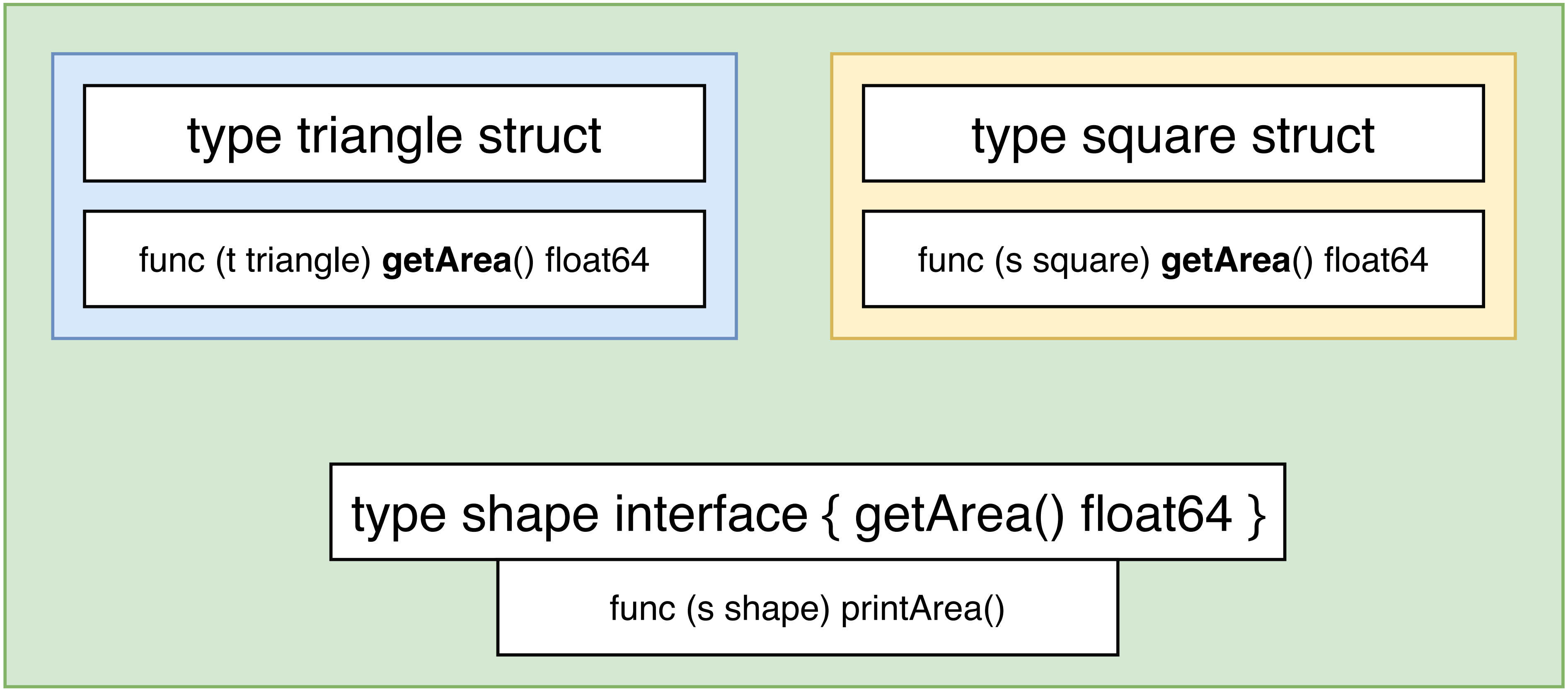
The 'square' type should be a struct with a field called 'sideLength' of type float64

The 'triangle' type should be a struct with a field called 'height' of type float64 and a field of type 'base' of type float64

Both types should have function called 'getArea' that returns the calculated area of the square or triangle

Area of a triangle = 0.5 * base * height.
Area of a square = sideLength * sideLength

Add a 'shape' interface that defines a function called 'printArea'. This function should calculate the area of the given shape and print it out to the terminal Design the interface so that the 'printArea' function can be called with either a triangle or a square.



type triangle struct

func (t triangle) **getArea**() float64

type square struct

func (s square) **getArea**() float64

type shape interface { getArea() float64 }

func (s shape) printArea()

Hard Mode
Assignment

Create a program that reads the contents of a text file then prints its contents to the terminal.

The file to open should be provided as an argument to the program when it is executed at the terminal. For example, '*go run main.go myfile.txt*' should open up the *myfile.txt* file

To read in the arguments provided to a program, you can reference the variable '*os.Args*', which is a slice of type string

To open a file, check out the documentation for the '*Open*' function in the '*os*' package - https://golang.org/pkg/os/#Open

What interfaces does the 'File' type implement?

If the '*File*' type implements the '*Reader*' interface, you might be able to reuse that *io.Copy* function!