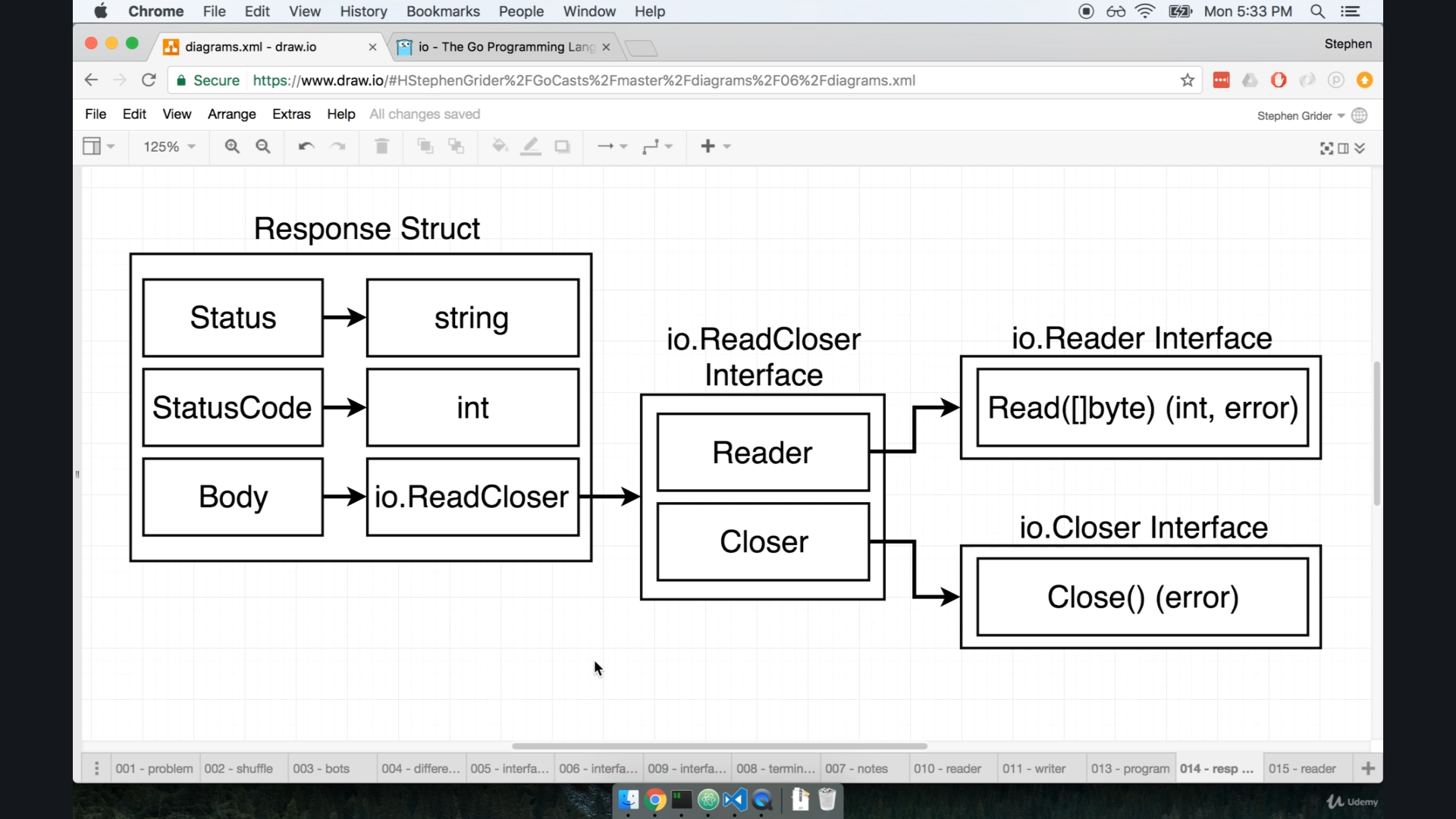
**video no -60, name - more Interface syntax**



In the last lecture we have seen some response struct in go lang documentation.

and it have a Body field which have **ReadCloser as an interaface**

and we use interface kind of reuse code till now.

So here we will talk about **why interface is used inside body of struct ?**

If we specify a interface as value inside struct it means the body field can have any value assign to it so long as it fulfills this interface.

(something like void pointer in C)

So if we create some kind of struct that have functions like:-

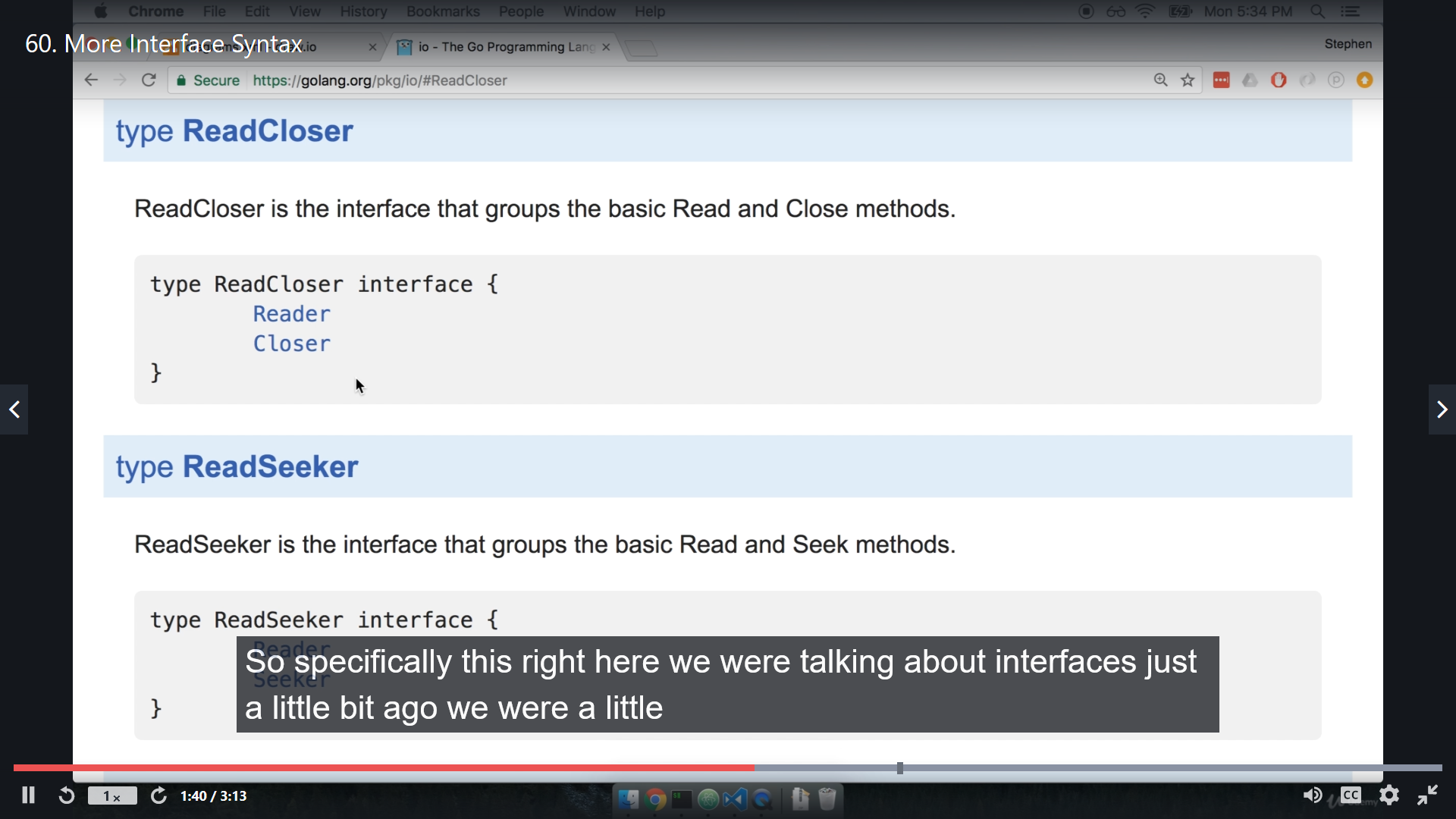
1. Read
2. Close

and obeyed all this type as here.

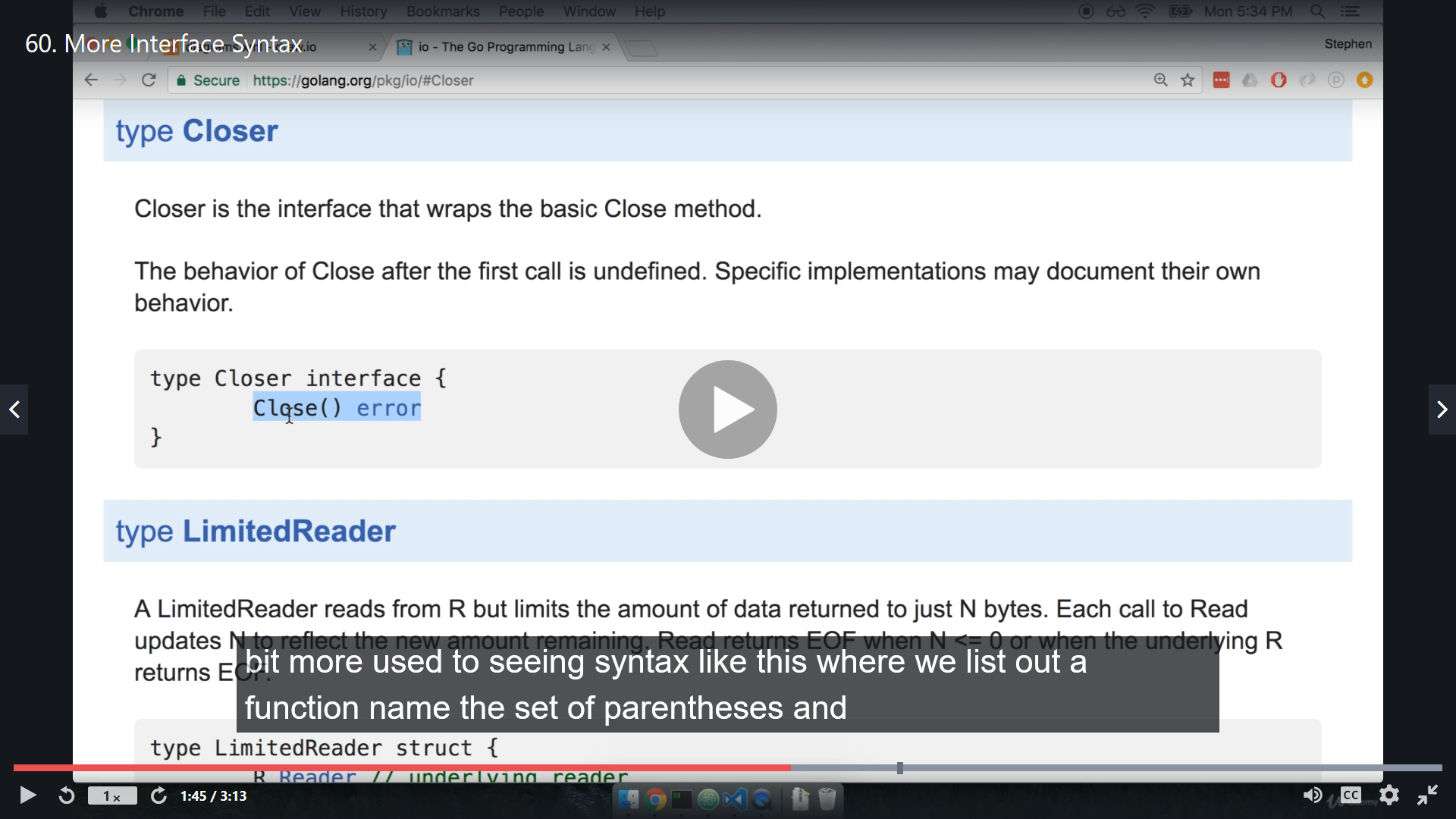
as mention in above pic.

We could then freely make a response struct then assign that field to this **Body** field.

But here we can see some kind funny syntax for interface like:-



we are more familiar with the interface like:-

 Function name set of parentheses and the return type.

Clearly that not a case in above **ReadCloser** interface.

In Go **we can take different interface and put it together to form a new interfaces**.

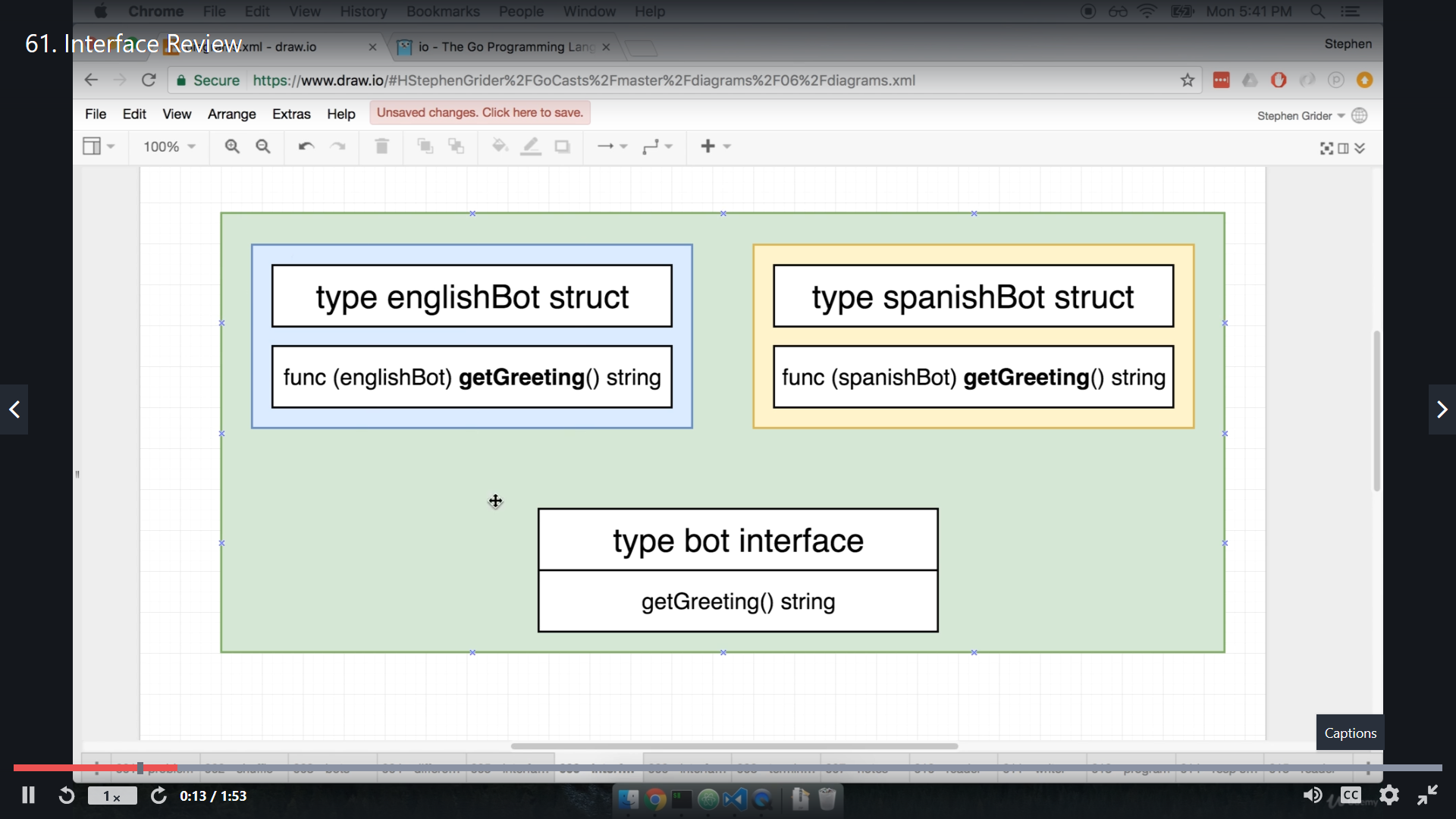
so both **Reader and Closer are interfaces**.

The **Reader** Reads **Closer** interfaces and said if you want to fulfill **ReadCloser** interface, then you have to satisfy the requirement of both reader and closer interface.

so we can put interface into other interface as many time as and as depth as it satisfy our requirement.

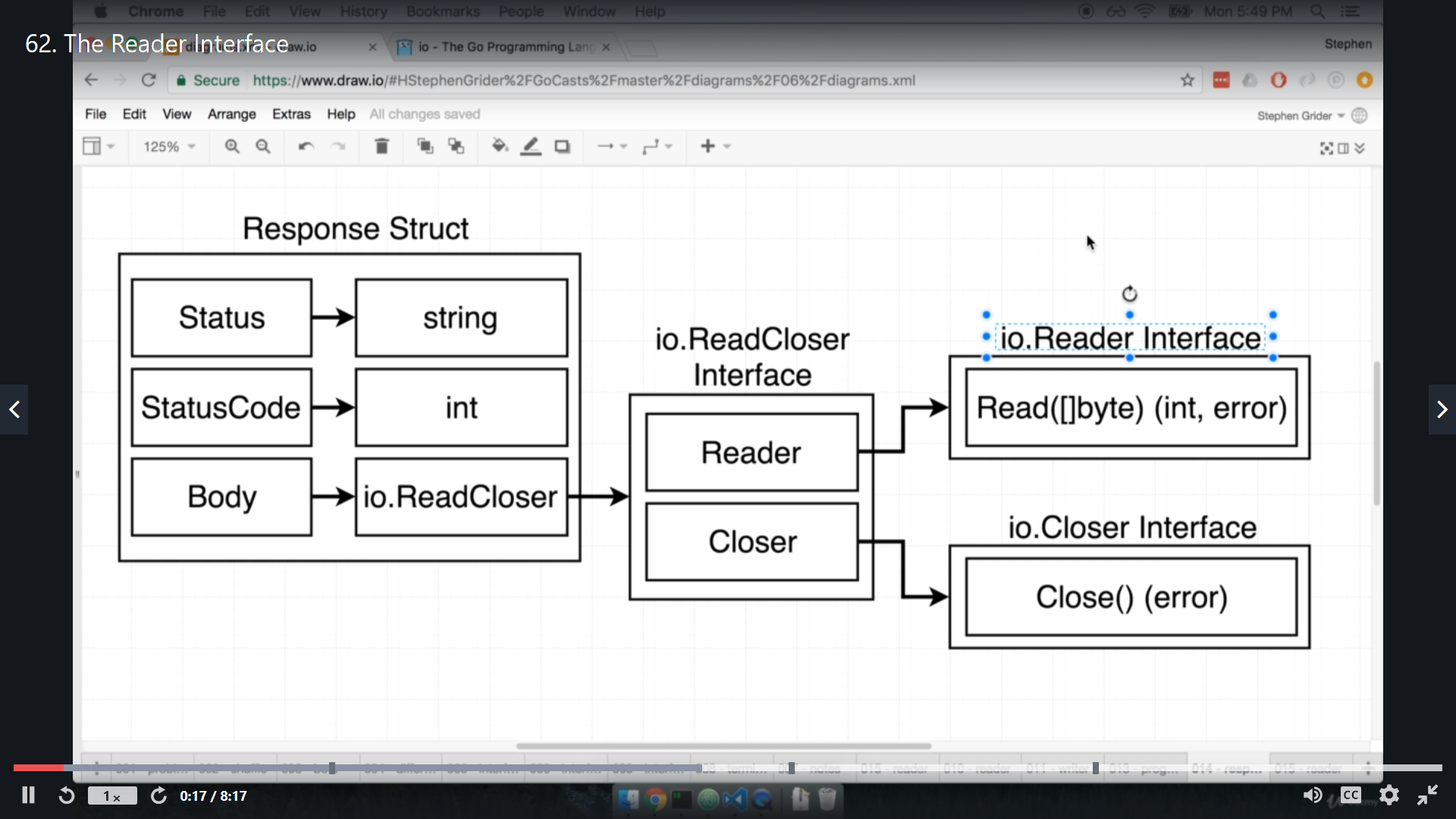
so in our case we need to full fill what **Reader and Closer interface** requiring from us to full fill **ReadCloser** interface.

**video no -61, name - Interface Review**



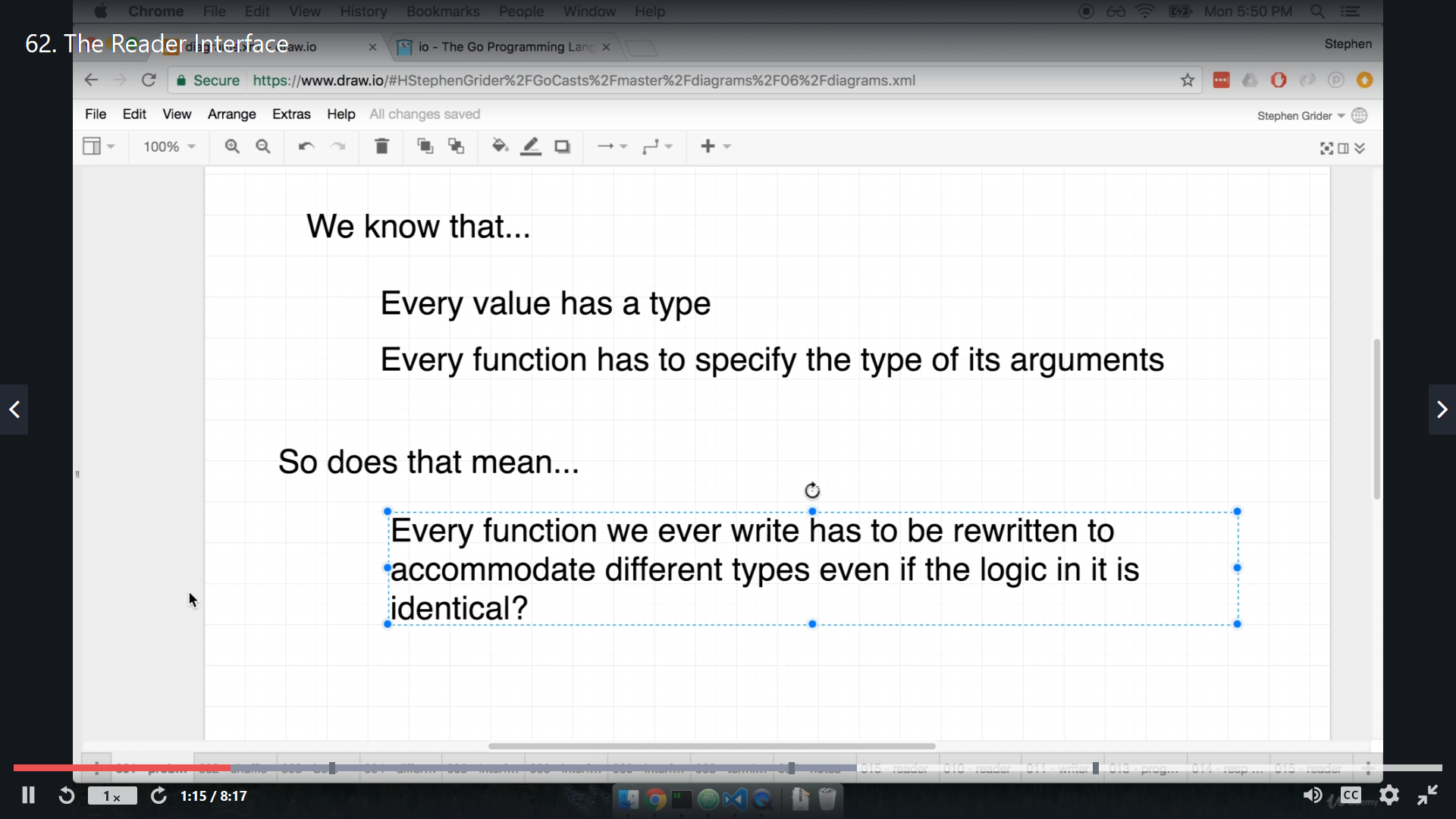
**Video no -62, name - Reader interface**

Currently we have seen interfaces are doing more harm than helping, but it not as such.



Now we are going to deep drive into **Reader Interface**

Just for review:-



so interfaces are introduce to get rid of this type issue.

Okay, this is diagram lets go to step by step

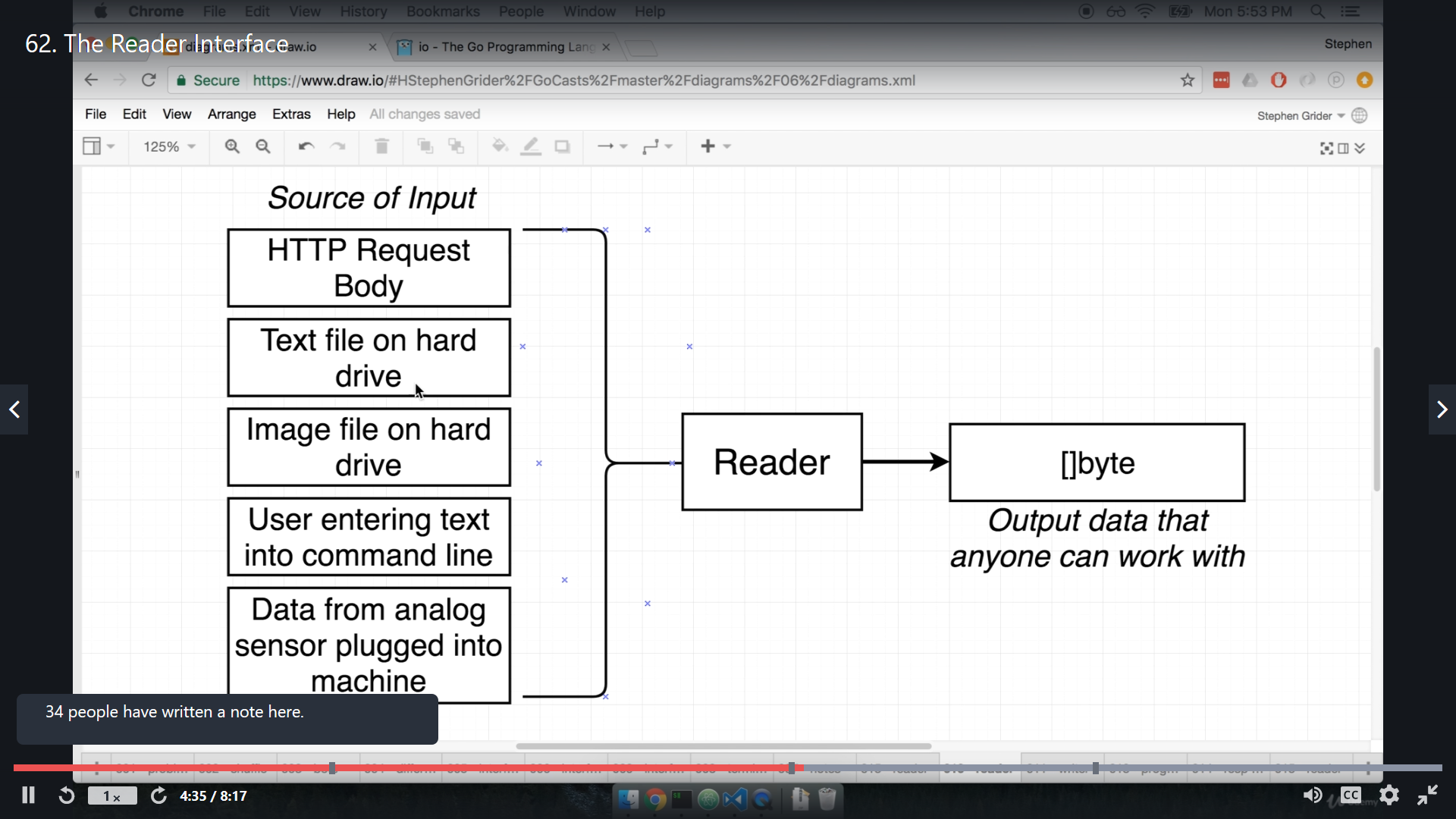
so this diagram should how things works without interface

for every input we have to define separate function even all function have same fucntionality.



so solution of above query is **Reader interface** .

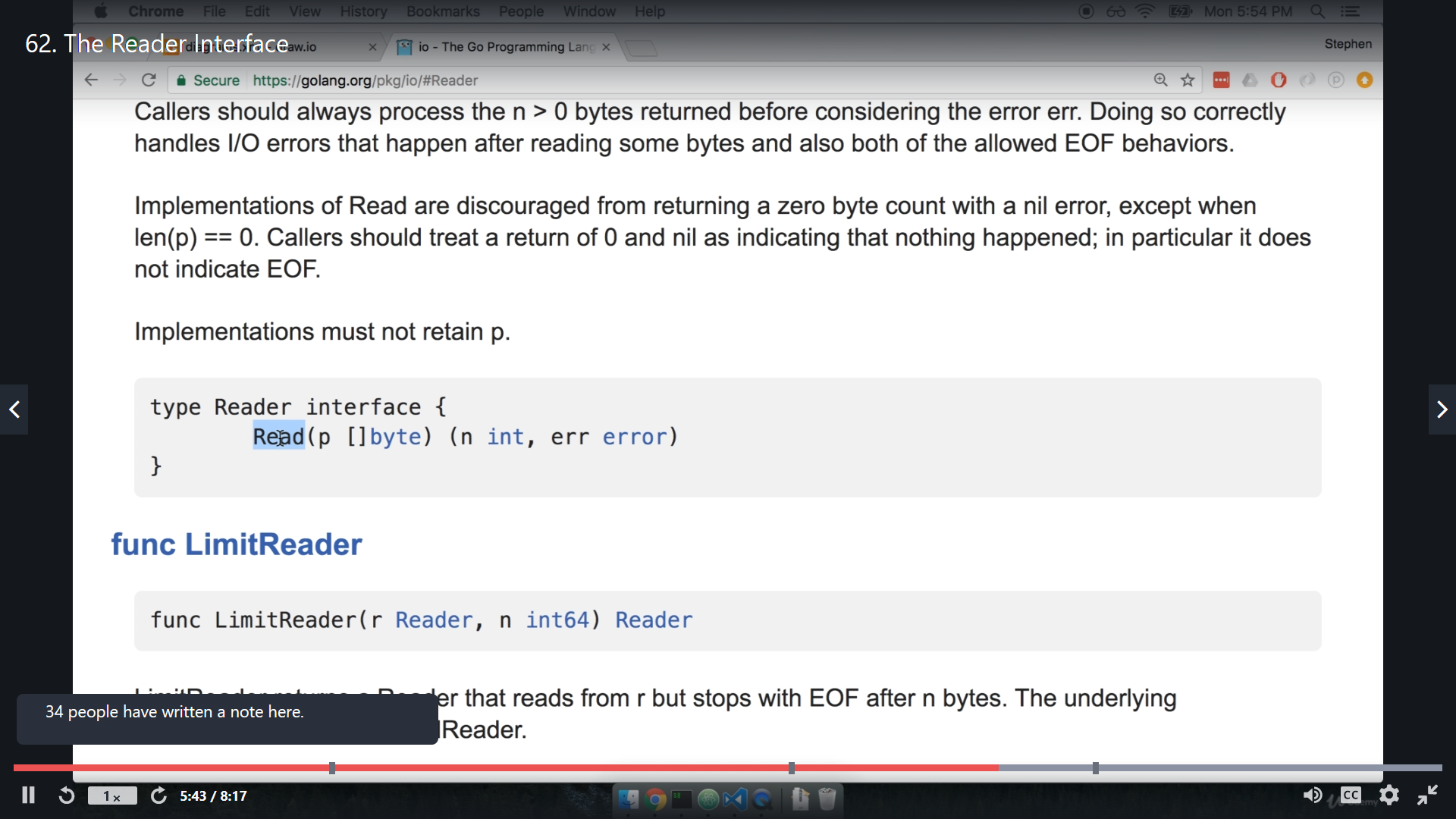
so we are actaully working little bit like this



so no matter what is source of input we have this reader interface all over the place.

but reader interface, also required us to define a function

So for any **type** to be **member of this reader interface** has to implement a **read function**.



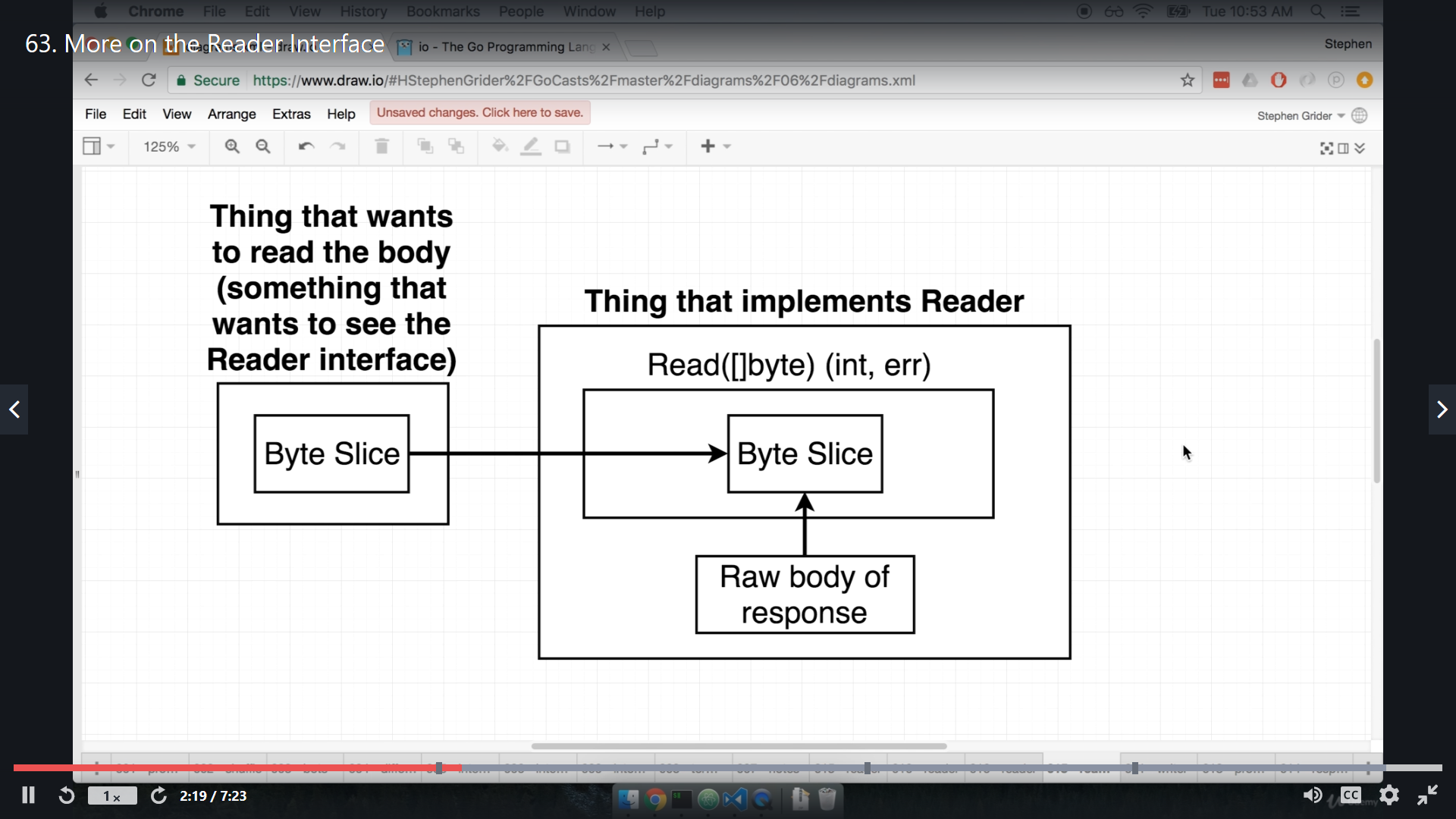
Basically all read function is actually returning something, but here we can see that we are passing some byteSlice to this function.

this rad function actually takes **byteSlice** as argument.

Actually the output of this Read function is actually a byte slice.

Look at Below diagram to understand read function. why it is taking byte slice as argument.

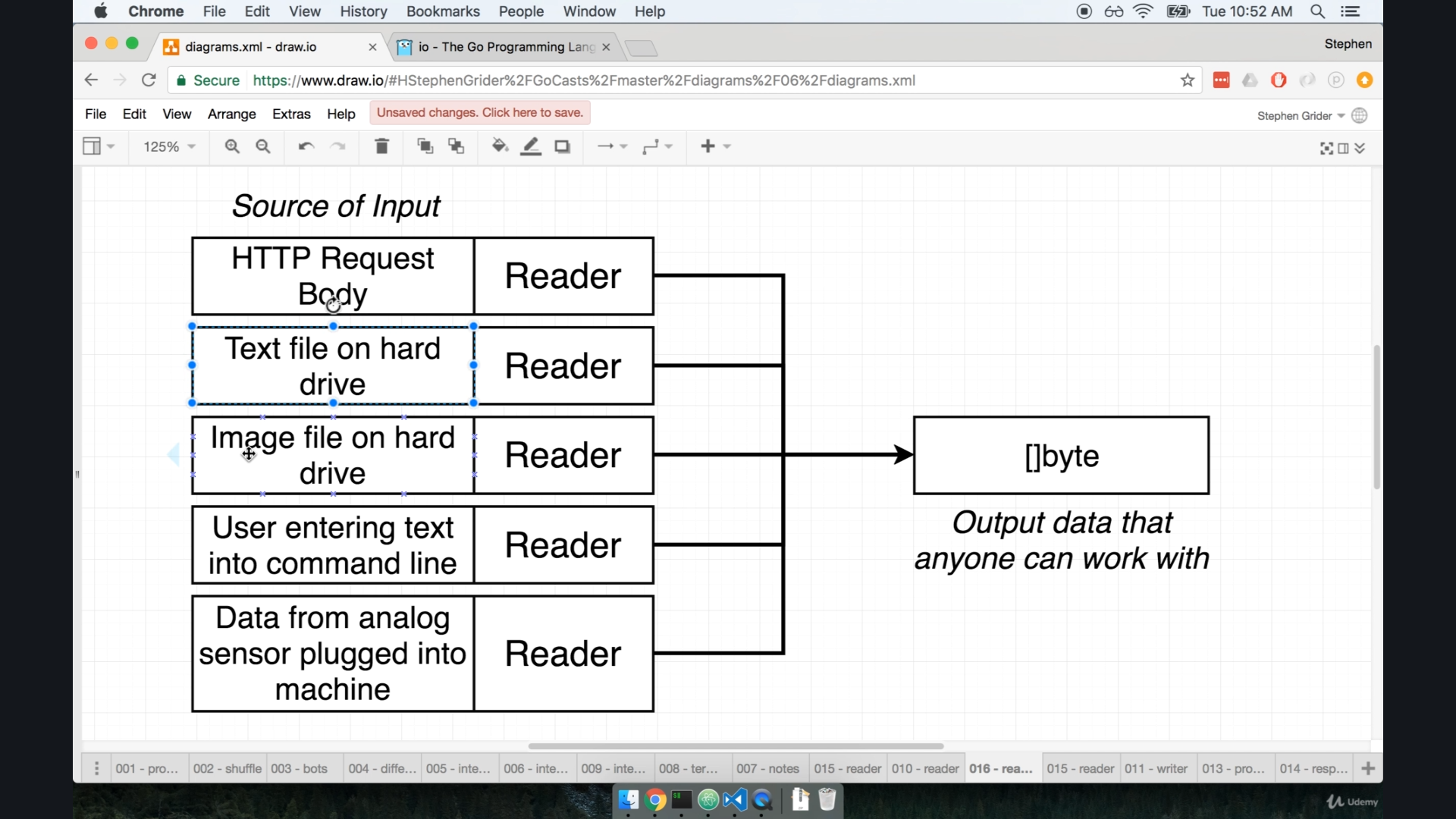
We know that **Slice is Reference type** , this mean when we pass a slice to a function (call by reference), and if we modify its content inside that function, it will be reflected on the original slice.



So here we can see that the user prepared new byteSlice (inital/dummy/garbage Slice), and pass that to read function, now this read function will take response lets say response json from google and write that into new byteSlice that we provided in argument. and this is Reference Type, so our original byteSlice will get modified and have response.

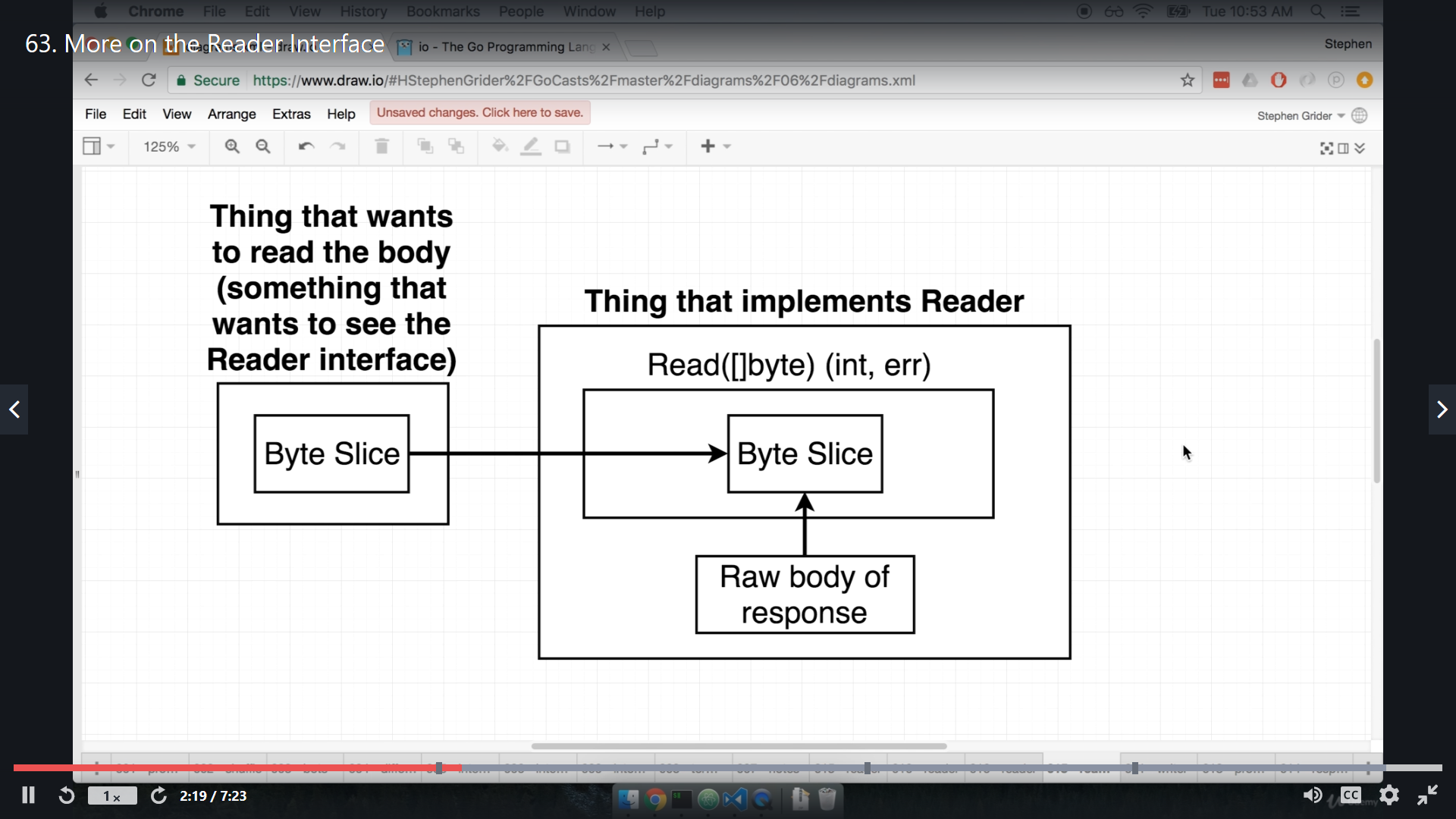
Read function also return "int" -> number of byte read. and err for error.

**video no - 63, name - More on reader interface**



This is more sophisticated diagram for Reader input.

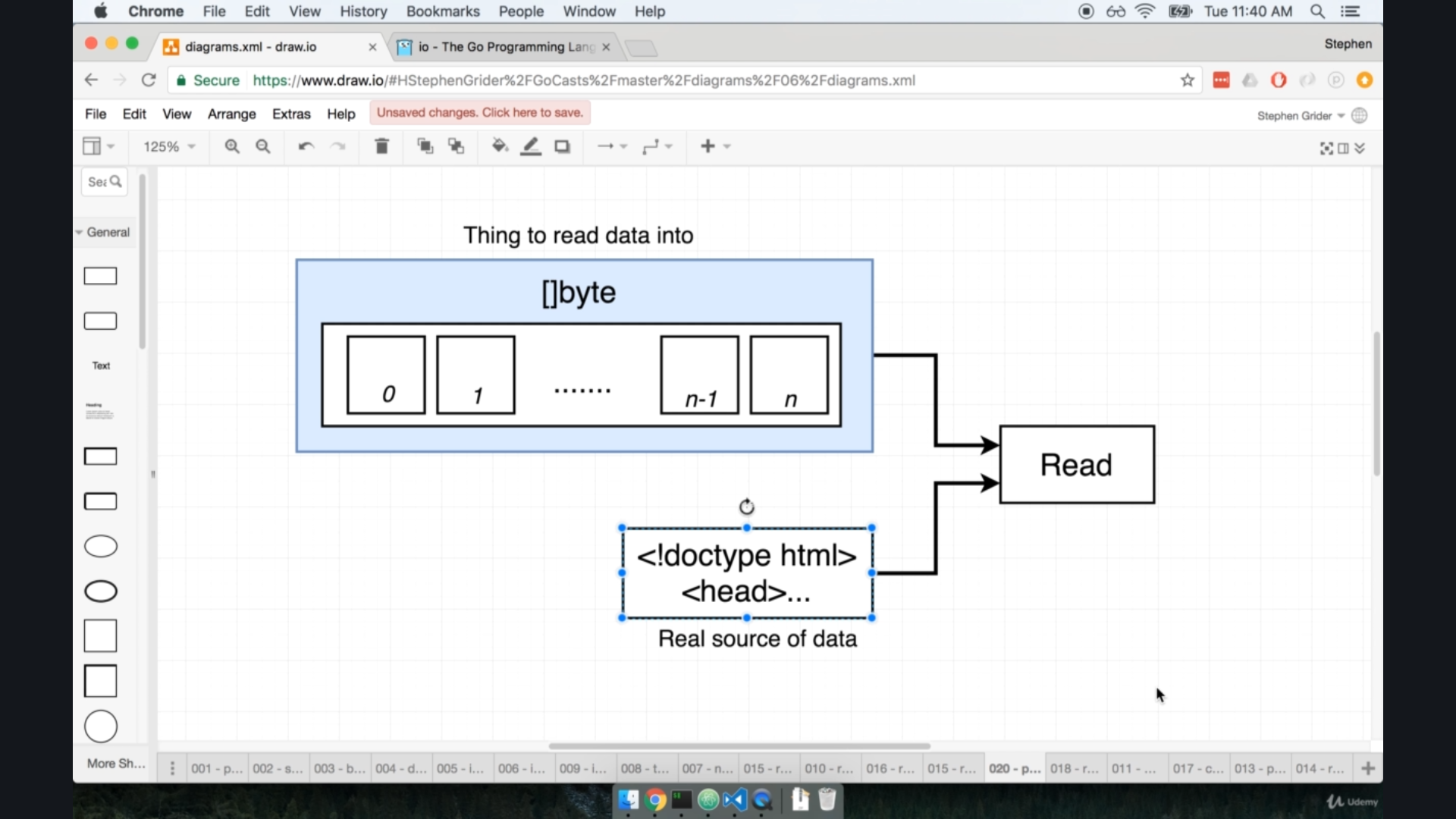
Sp what this read function is actually doing:-



As we have explain above . same thing.

**video no -64, name - working with read function**

we use g=> bs := make([]byte, 9999) => this syntax to create byte slice .



So what is happen here is that, we create a byte slice it will go to read function and read take reponse from net and put that into our byteSlice.

Also one thing to notice, we make **byteSlice of large size** because this read() function does **not** have capacity to increase size of byteSlice automatically.

Now to try some code :-

if we only want response:-

import (

    "fmt"

    "net/http"

)

func main() {

    resp, err := http.Get("http://google.com")

    if err != nil {

        fmt.Println("some error occur")

    }

    fmt.Println(resp)

}

O/p:- &{200 OK 200 HTTP/1.1 1 1 map[Cache-Control:[private, max-age=0] Content-Type:[text/html; chnly] X-Frame-Options:[SAMEORIGIN] X-Xss-Protection:[0]] 0xc00008c0c0 -1 [] false true map[] 0xc000214000 <nil>}

so this is response, But it is not actual body, for response body we need to use reader interface,

resp.Body => is interface that have Read function so we can do like

resp.Body.Read(byteSlice)

lets try that in code.

package main

import (

    "fmt"

    "net/http"

)

func main() {

    resp, err := http.Get("http://google.com")

    if err != nil {

        fmt.Println("some error occur")

    }

    bs := make([]byte, 99999)

    resp.Body.Read(bs)

    fmt.Println(string(bs))

}

O/P :- <!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg\_standard\_color\_128dp.png" itemprop="image"><title>Google</title><script nonce="l

Now we get body.

Also we don’t need to define this byteSlice manually, there is built in support for that which we will see, little later.

**Video no - 65, name - the writer interface**

Now we are going to condense the code which we write earlier. this will do like it will take all response body automatically and put that into terminal.

we are going to write a single line of code for that like :-

this three line of code I  
bs := make([]byte, 99999)

    resp.Body.Read(bs)

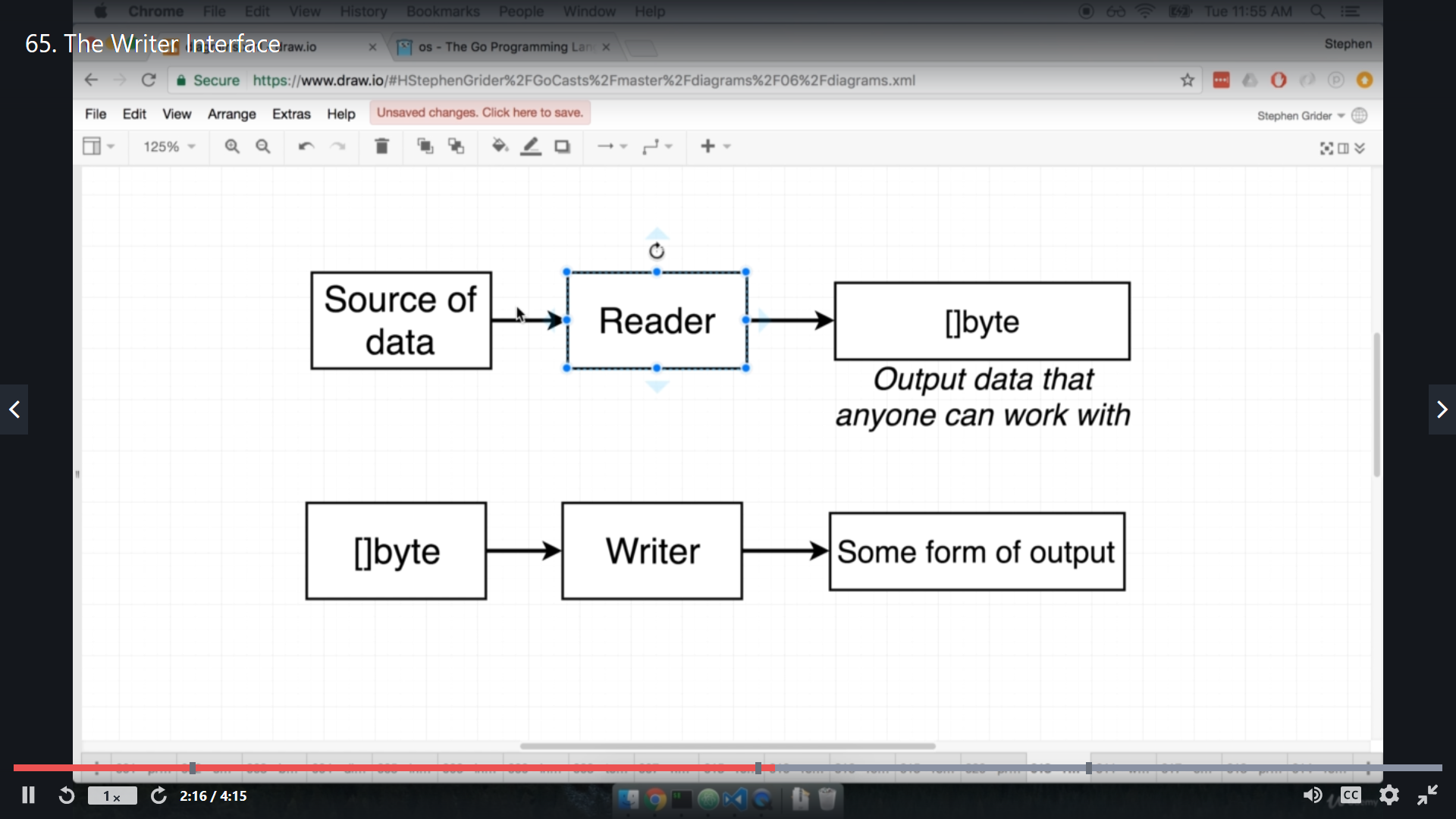
    fmt.Println(string(bs))

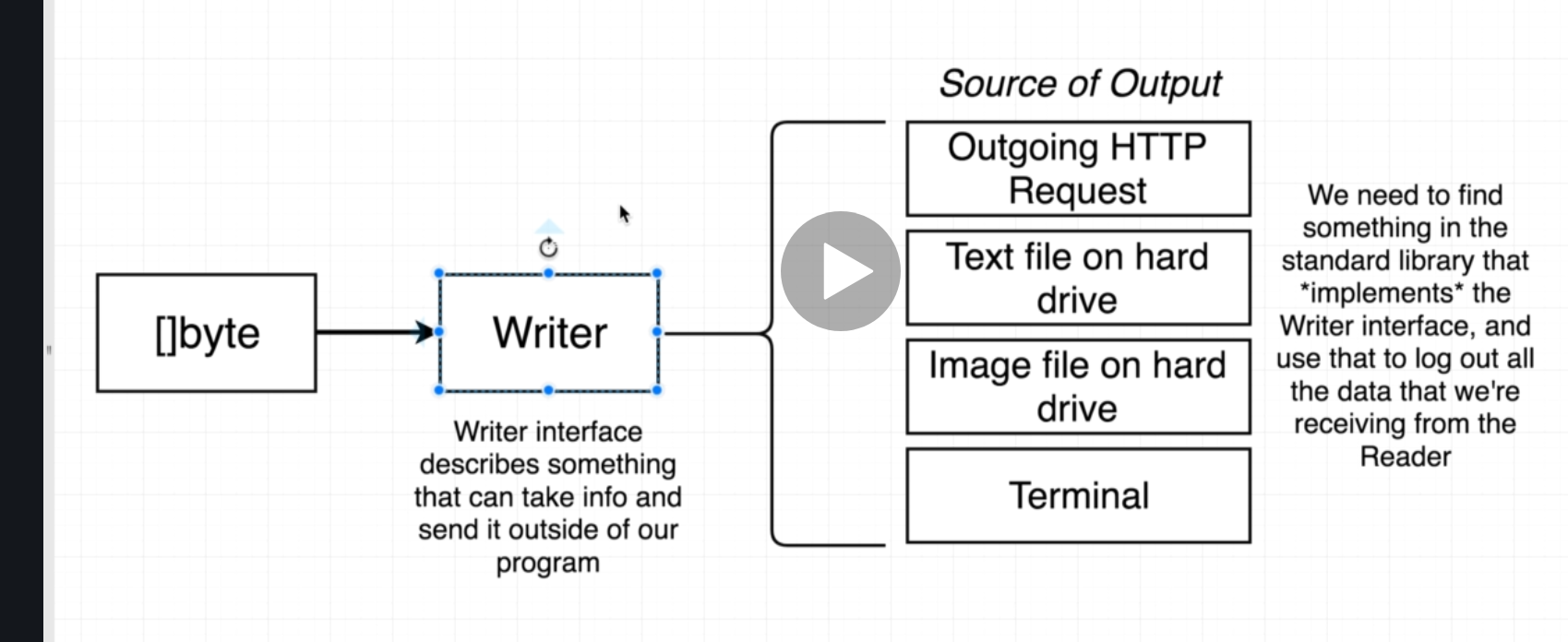
is replaced to this one

g=> io.copy(os.Stdout, resp.Body)

so now where read function, byteSlice all gone ??

lets see this diagram:-





package main

import (

    "fmt"

    "io"

    "net/http"

    "os"

)

func main() {

    resp, err := http.Get("http://google.com")

    if err != nil {

        fmt.Println("some error occur")

    }

    io.Copy(os.Stdout, resp.Body)

}