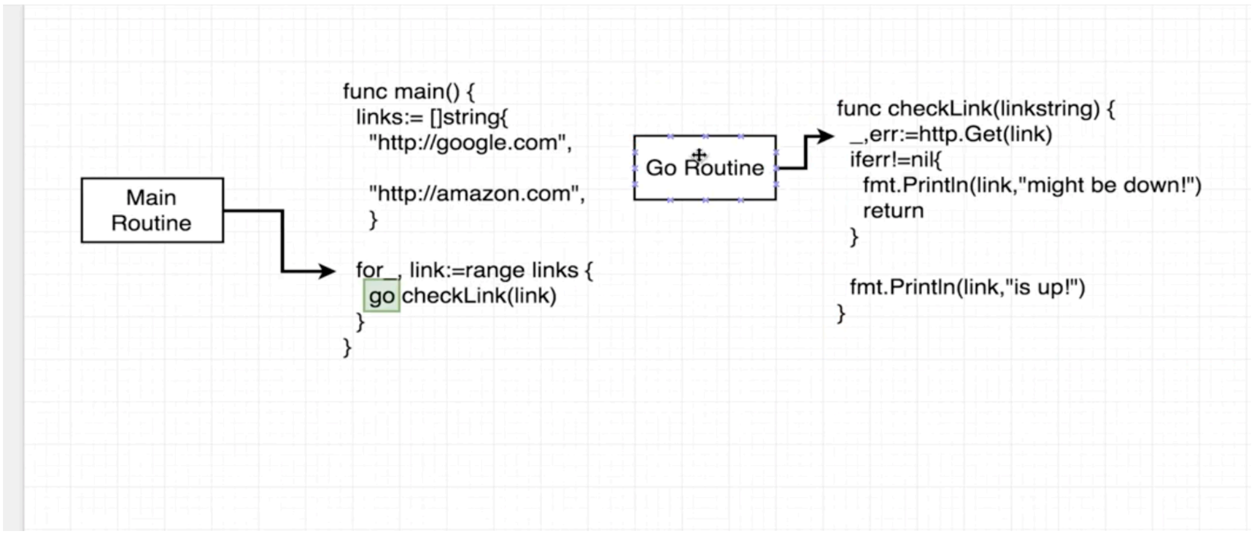
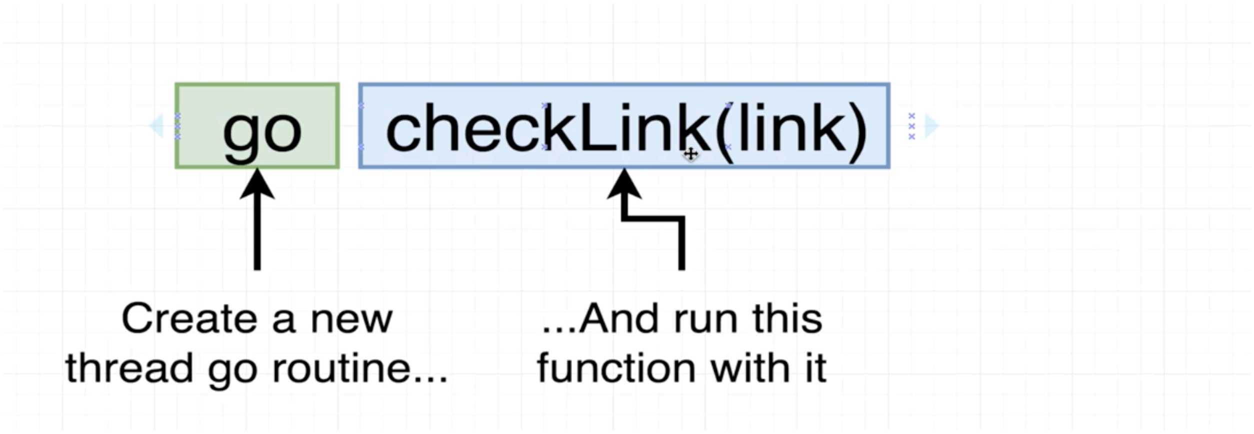
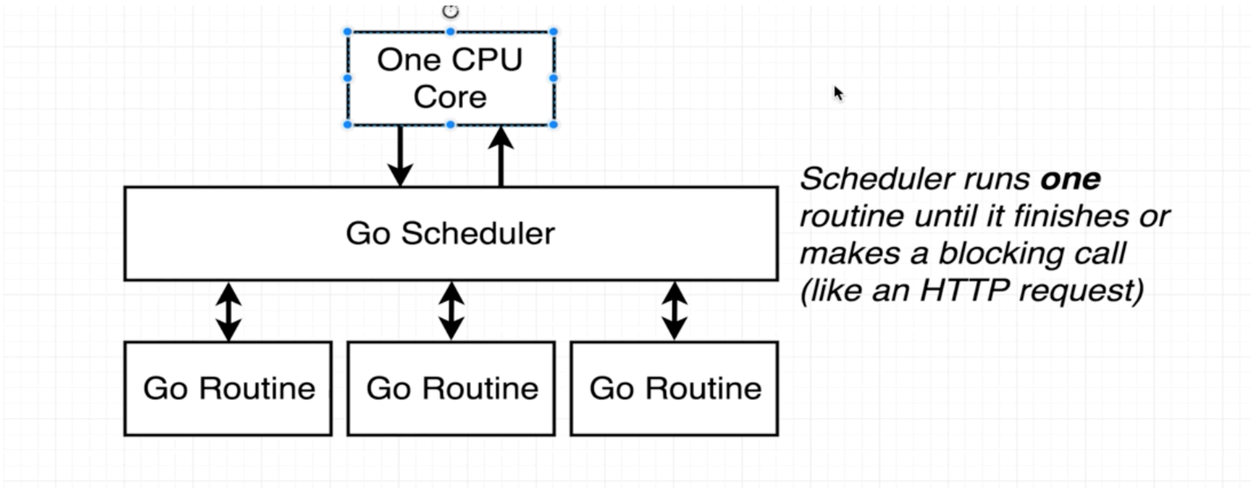
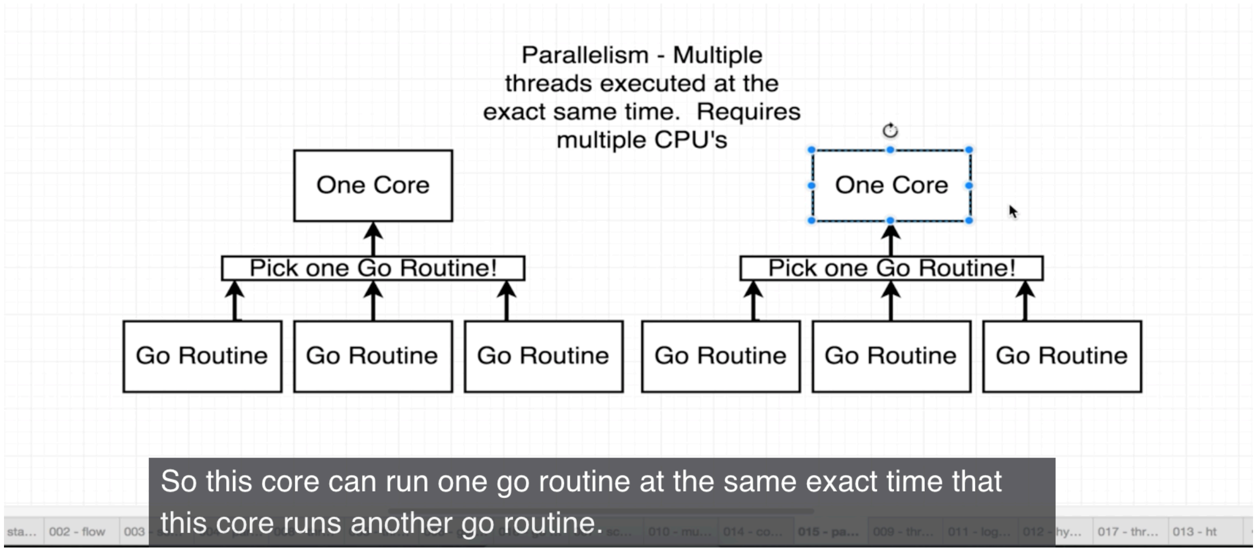


```
func main() {
  links:= []string{
    "http://google.com",

    "http://amazon.com",
    }

  for_, link:=range links {
   go checkLink(link)
  }
}
```

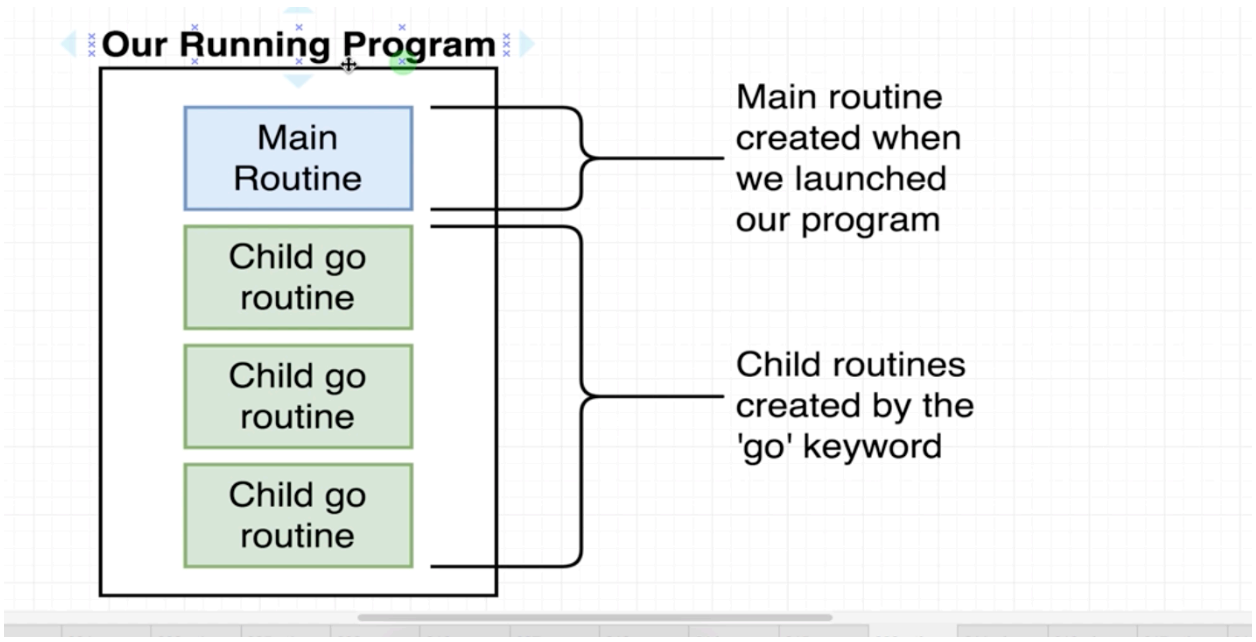
```
func checkLink(linkstring) {
  _,err:=http.Get(link)
  iferr!=nil{
    fmt.Println(link,"might be down!")
    return
  }

  fmt.Println(link,"is up!")
}
```

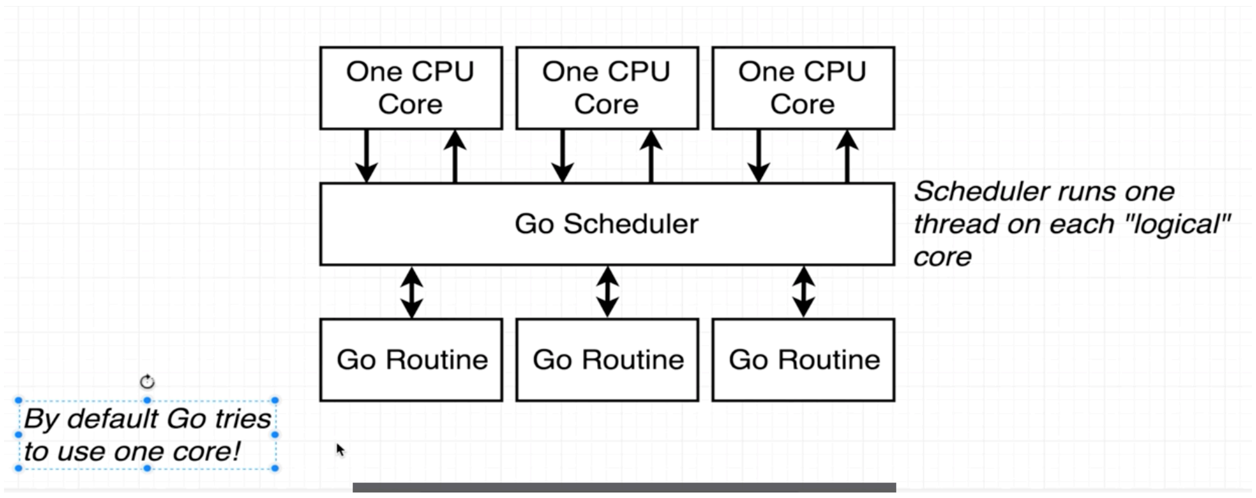

go
checkLink(link)
Create a new thread go routine...
...And run this function with it


One CPU Core
Go Scheduler
Go Routine
Go Routine
Go Routine
Scheduler runs ***one*** *routine until it finishes or makes a blocking call (like an HTTP request)*

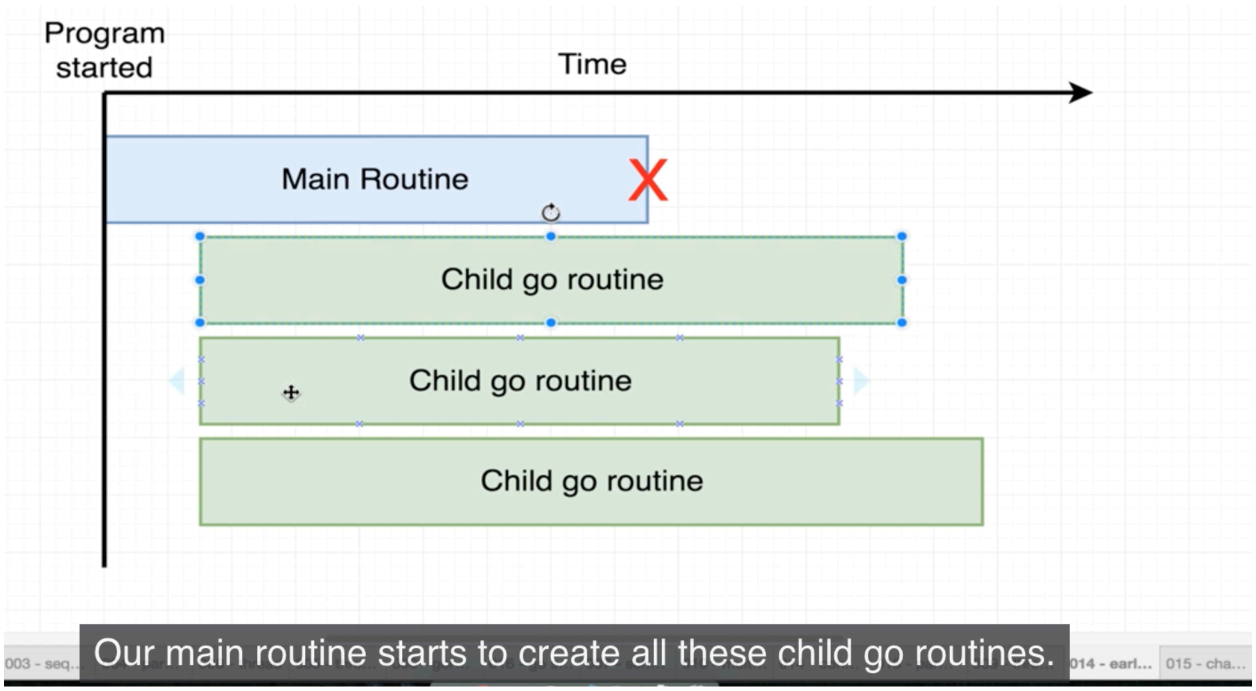


So this core can run one go routine at the same exact time that this core runs another go routine.


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

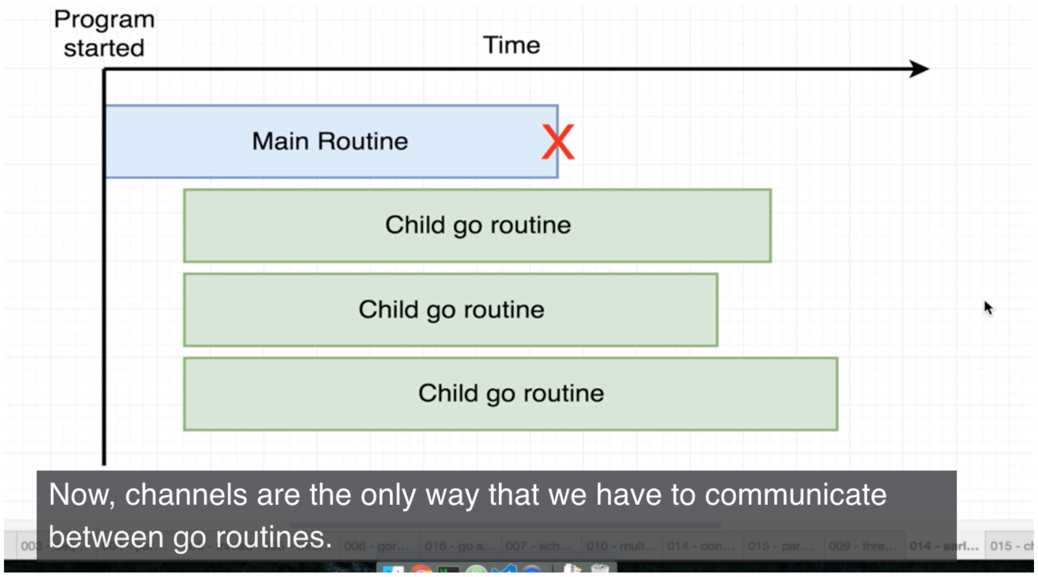

One CPU Core
One CPU Core
One CPU Core
Go Scheduler
Go Routine
Go Routine
Go Routine
Scheduler runs one thread on each "logical" core
By default Go tries to use one core!

ERROR

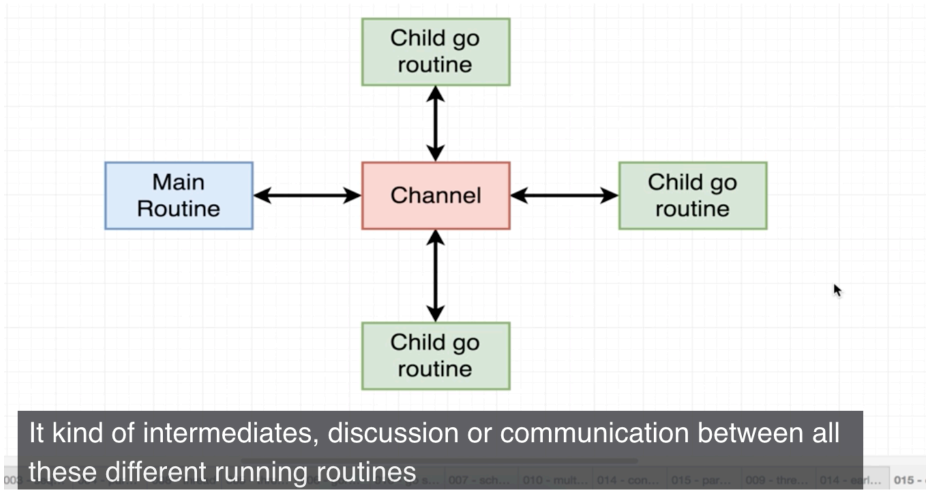


Our main routine starts to create all these child go routines.

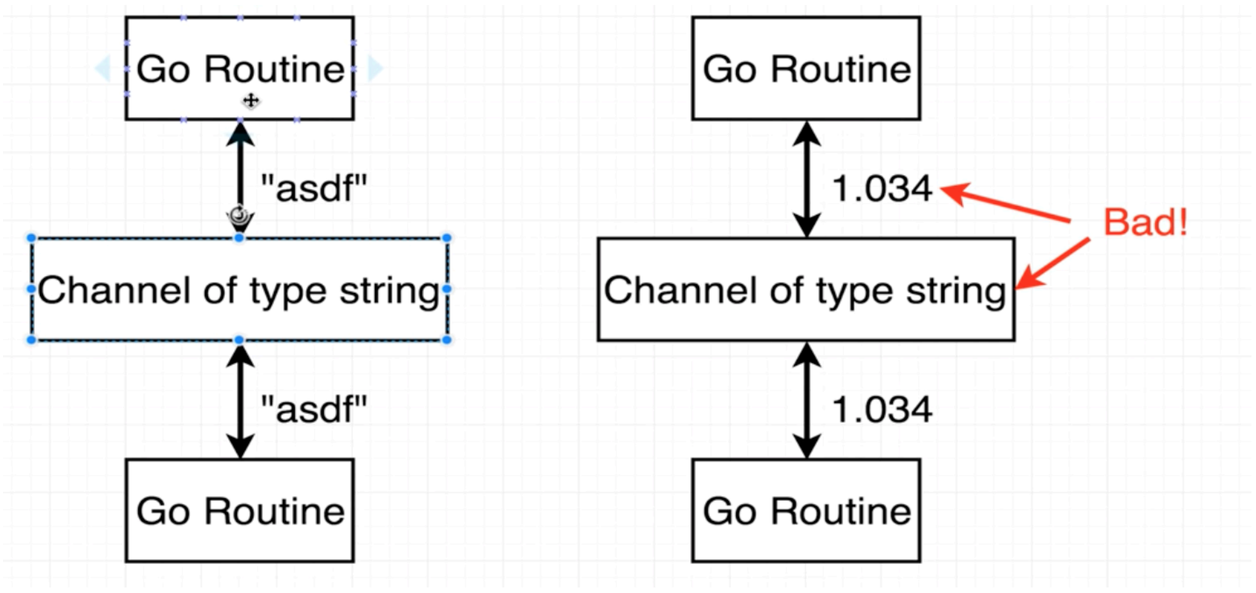
Channels

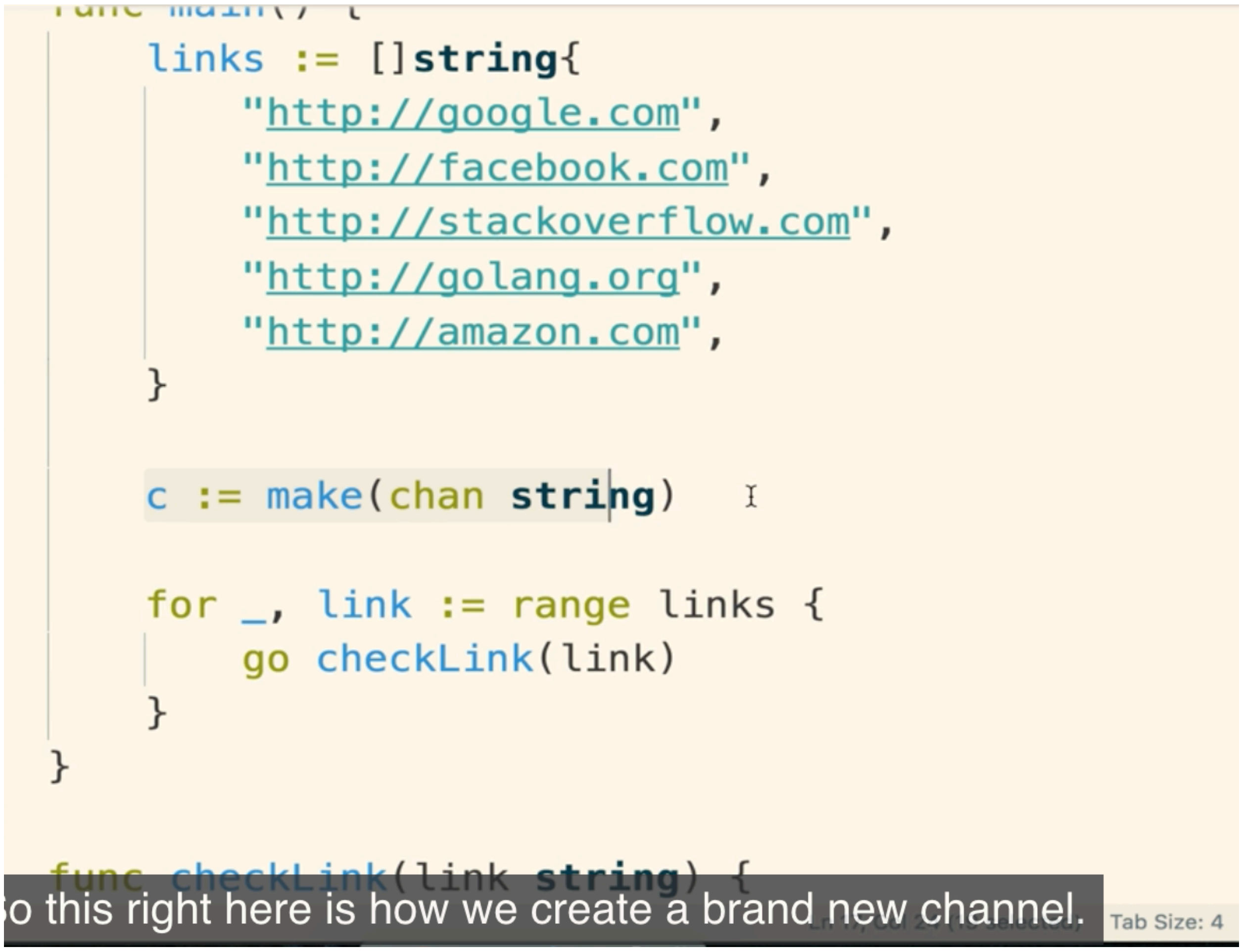


Now, channels are the only way that we have to communicate between go routines.



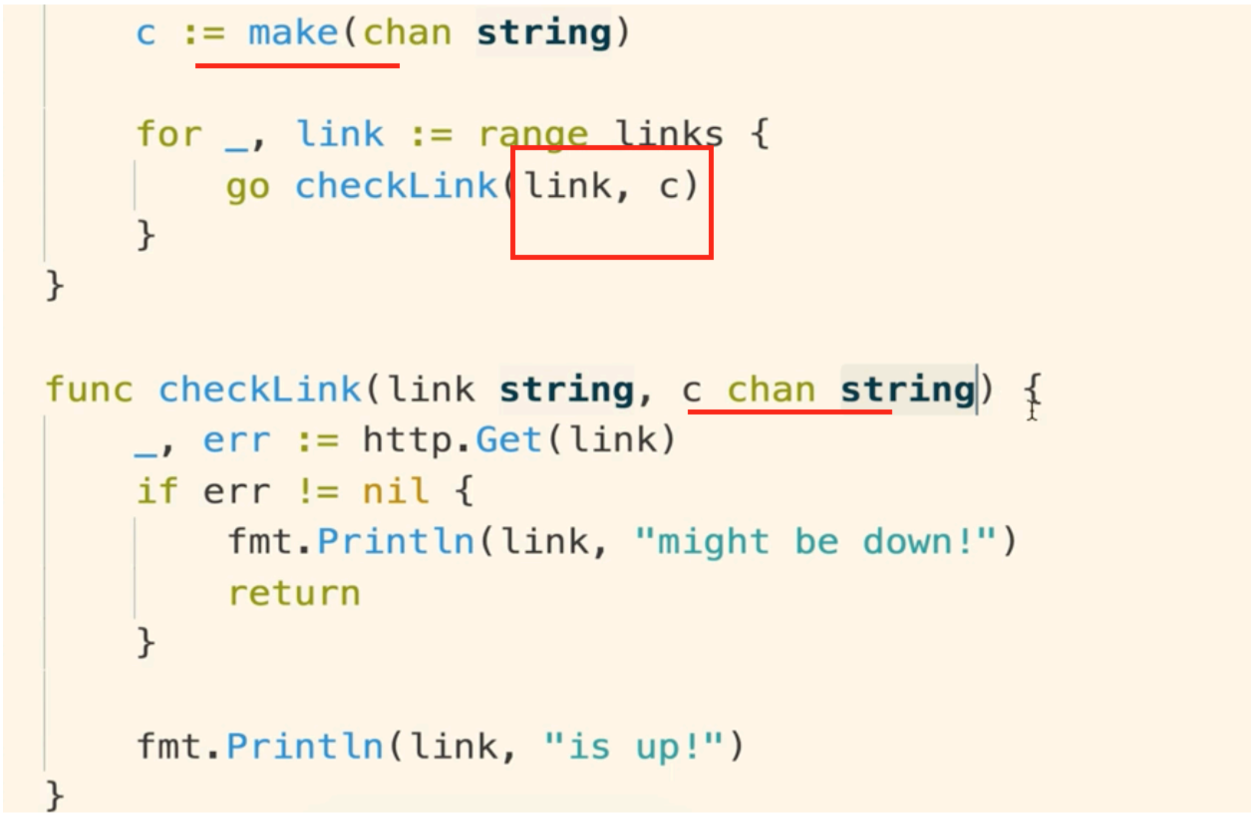
It kind of intermediates, discussion or communication between all these different running routines



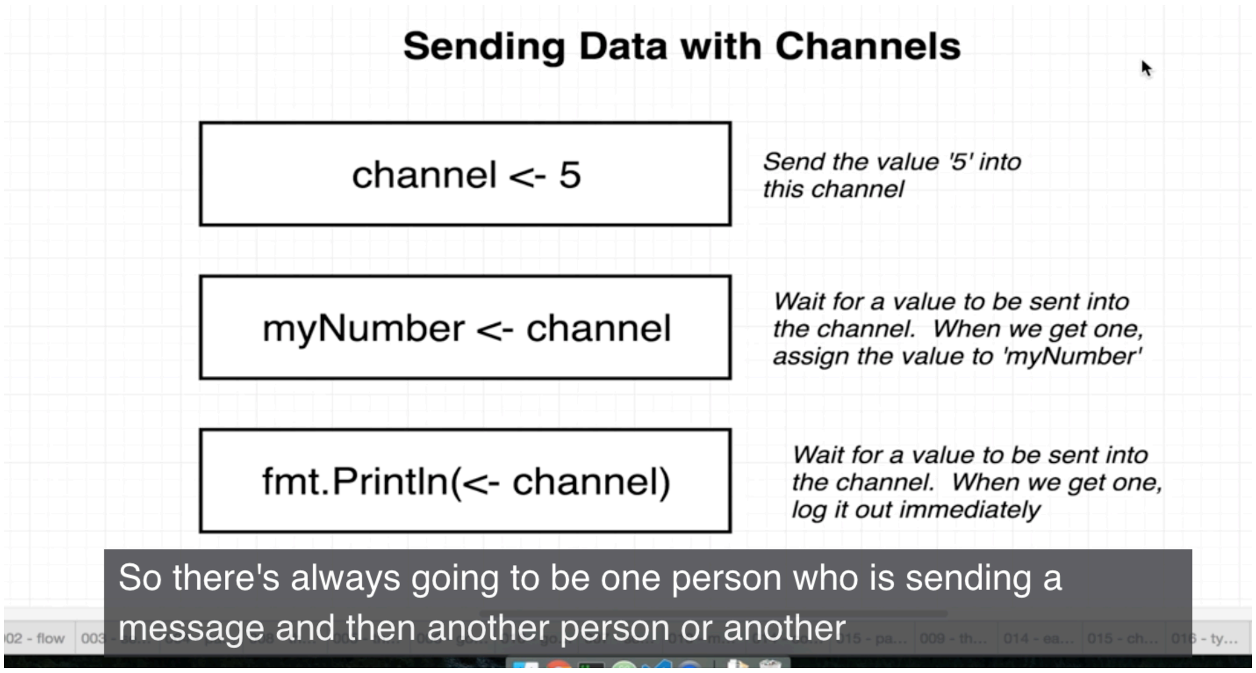


```
func main() {
    links := []string{
        "http://google.com",
        "http://facebook.com",
        "http://stackoverflow.com",
        "http://golang.org",
        "http://amazon.com",
    }

    c := make(chan string)

    for _, link := range links {
        go checkLink(link)
    }
}

func checkLink(link string) {
```

So this right here is how we create a brand new channel.



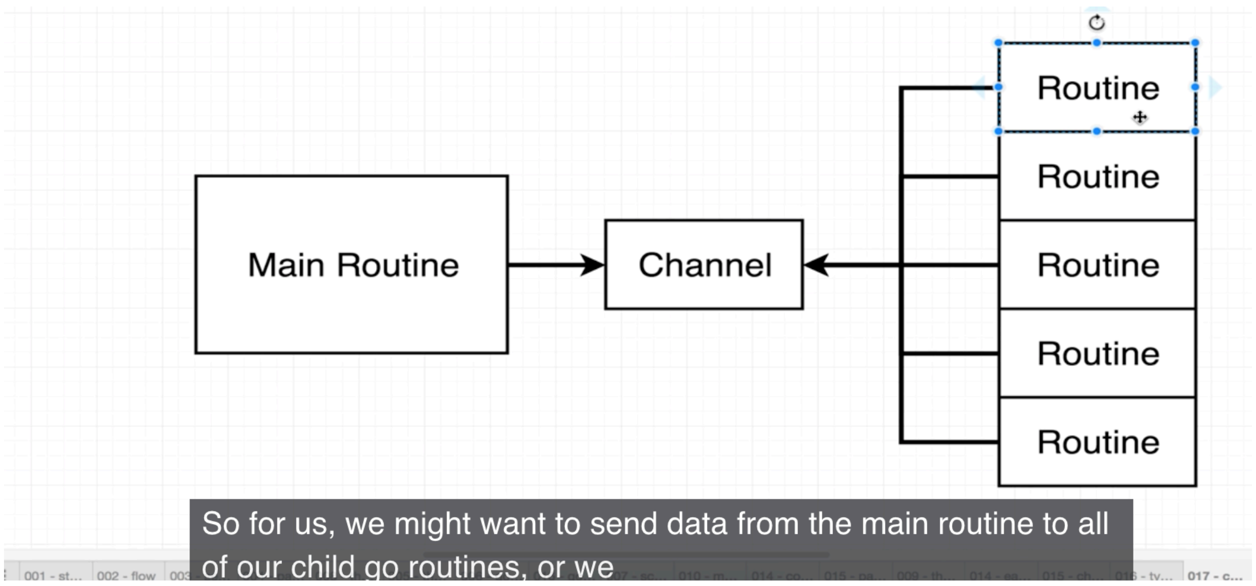
```
    c := make(chan string)

    for _, link := range links {
        go checkLink(link, c)
    }
}

func checkLink(link string, c chan string) {
    _, err := http.Get(link)
    if err != nil {
        fmt.Println(link, "might be down!")
        return
    }

    fmt.Println(link, "is up!")
}
```



Sending Data with Channels

channel <- 5	*Send the value '5' into this channel*
myNumber <- channel	*Wait for a value to be sent into the channel. When we get one, assign the value to 'myNumber'*
fmt.Println(<- channel)	*Wait for a value to be sent into the channel. When we get one, log it out immediately*

So there's always going to be one person who is sending a message and then another person or another



So for us, we might want to send data from the main routine to all of our child go routines, or we