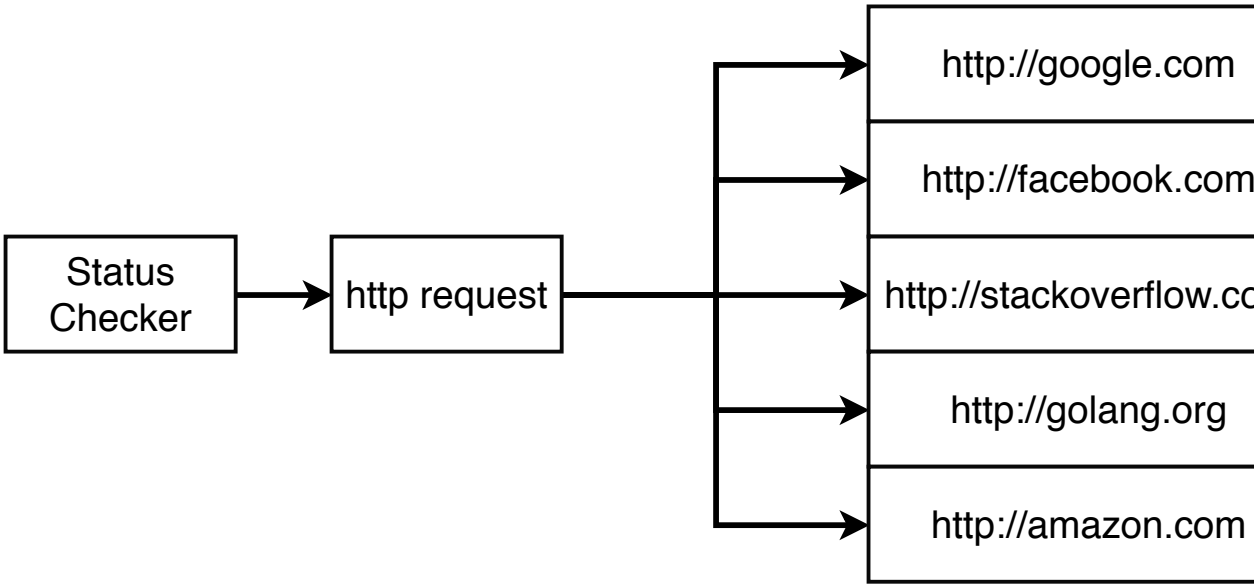
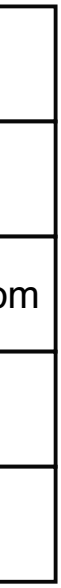
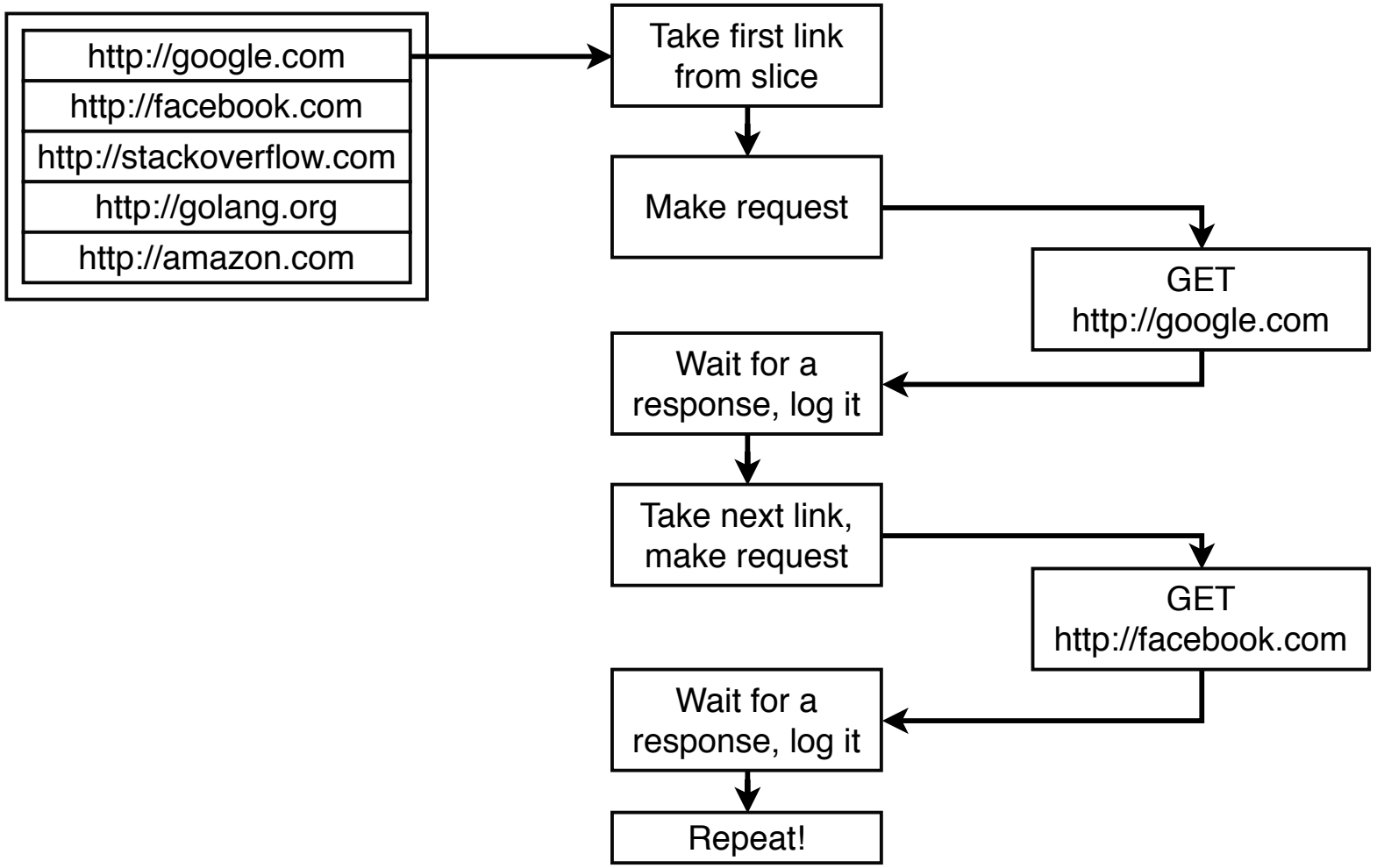
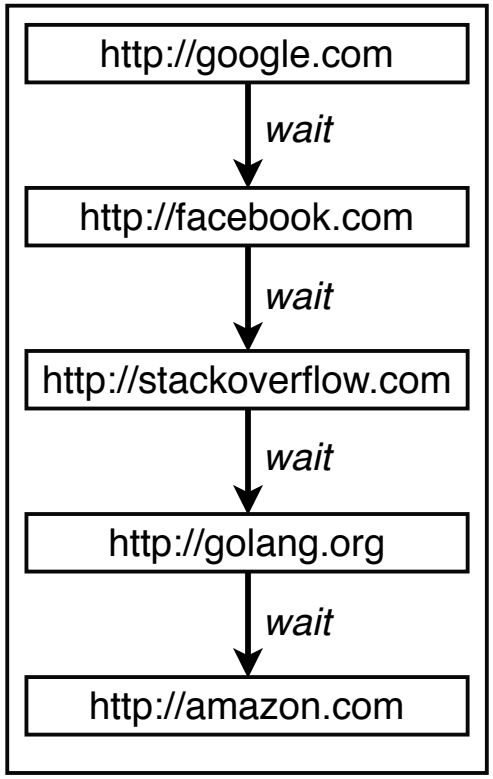
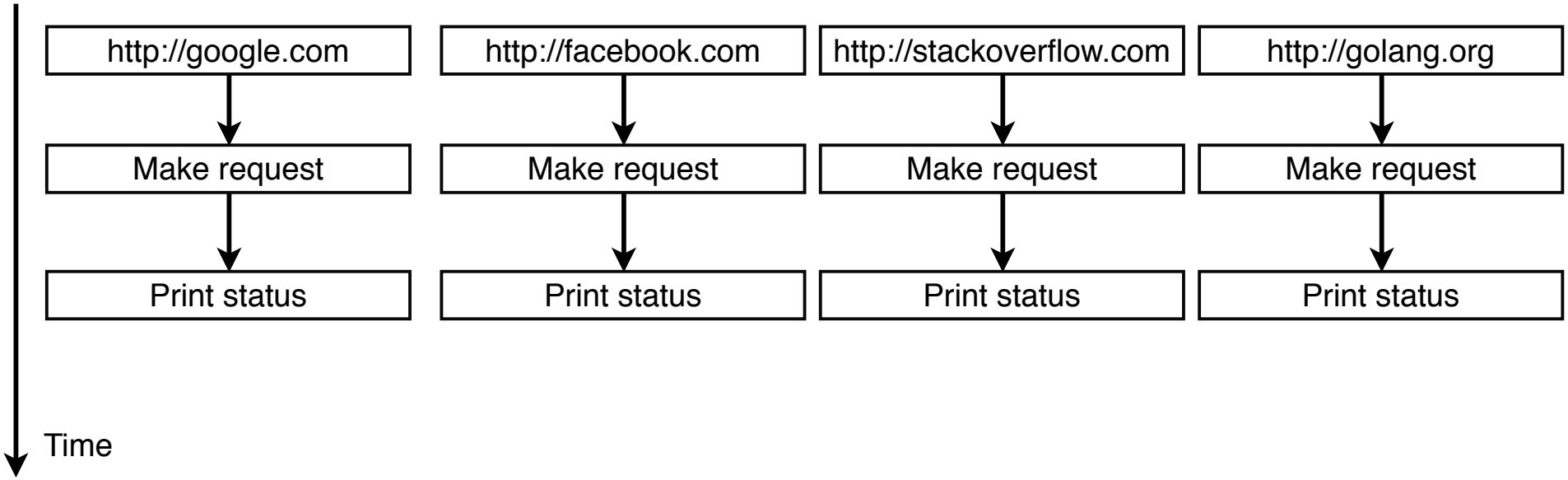
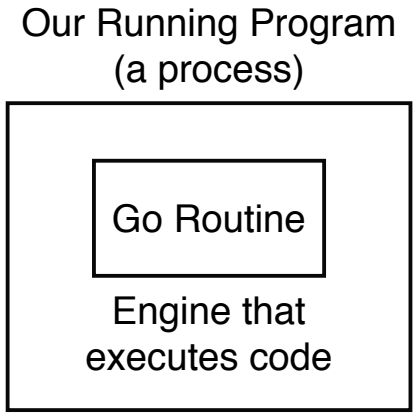

Status Checker
http request
http://google.com
http://facebook.com
http://stackoverflow.co
http://golang.org
http://amazon.com

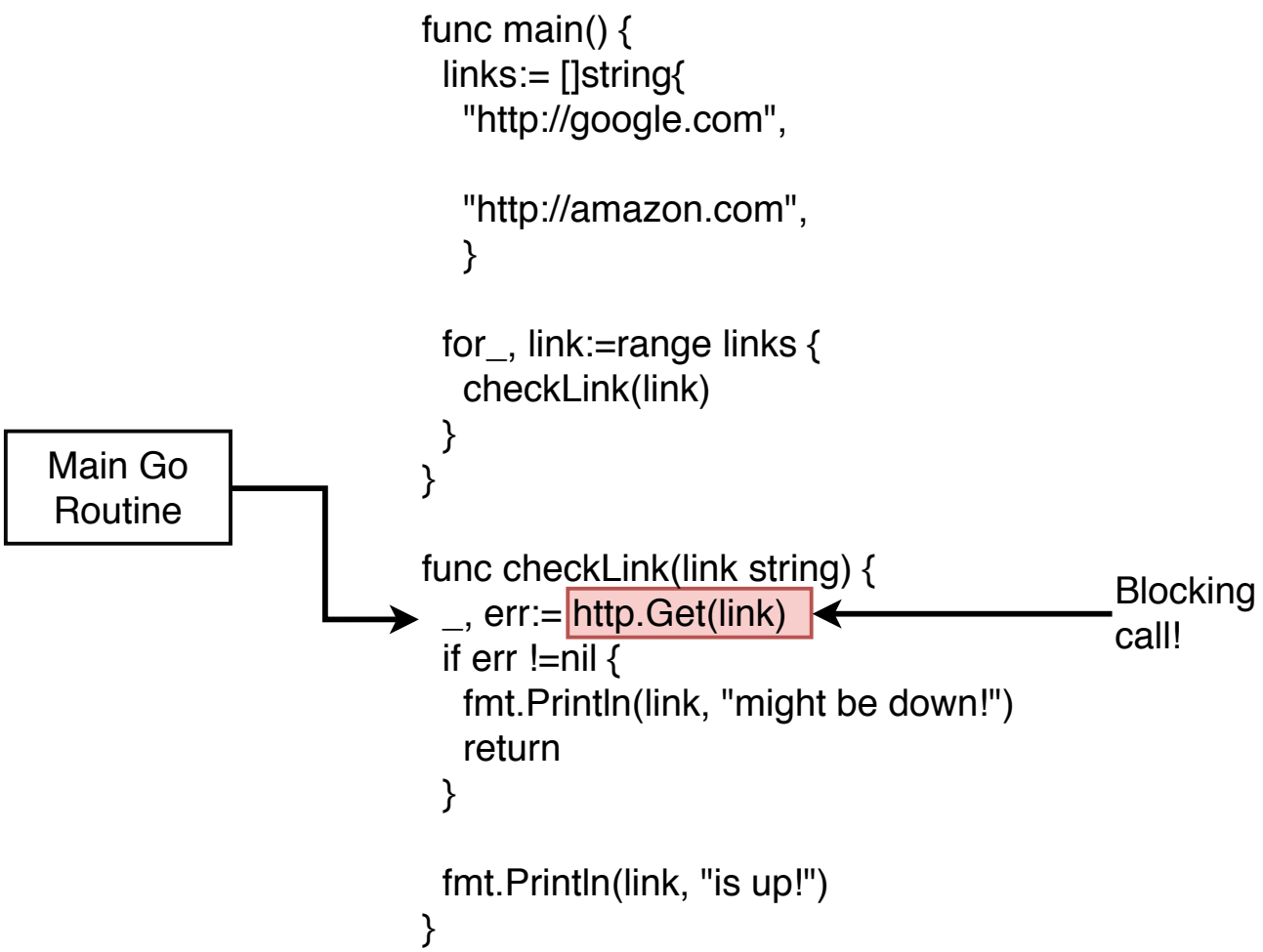

om

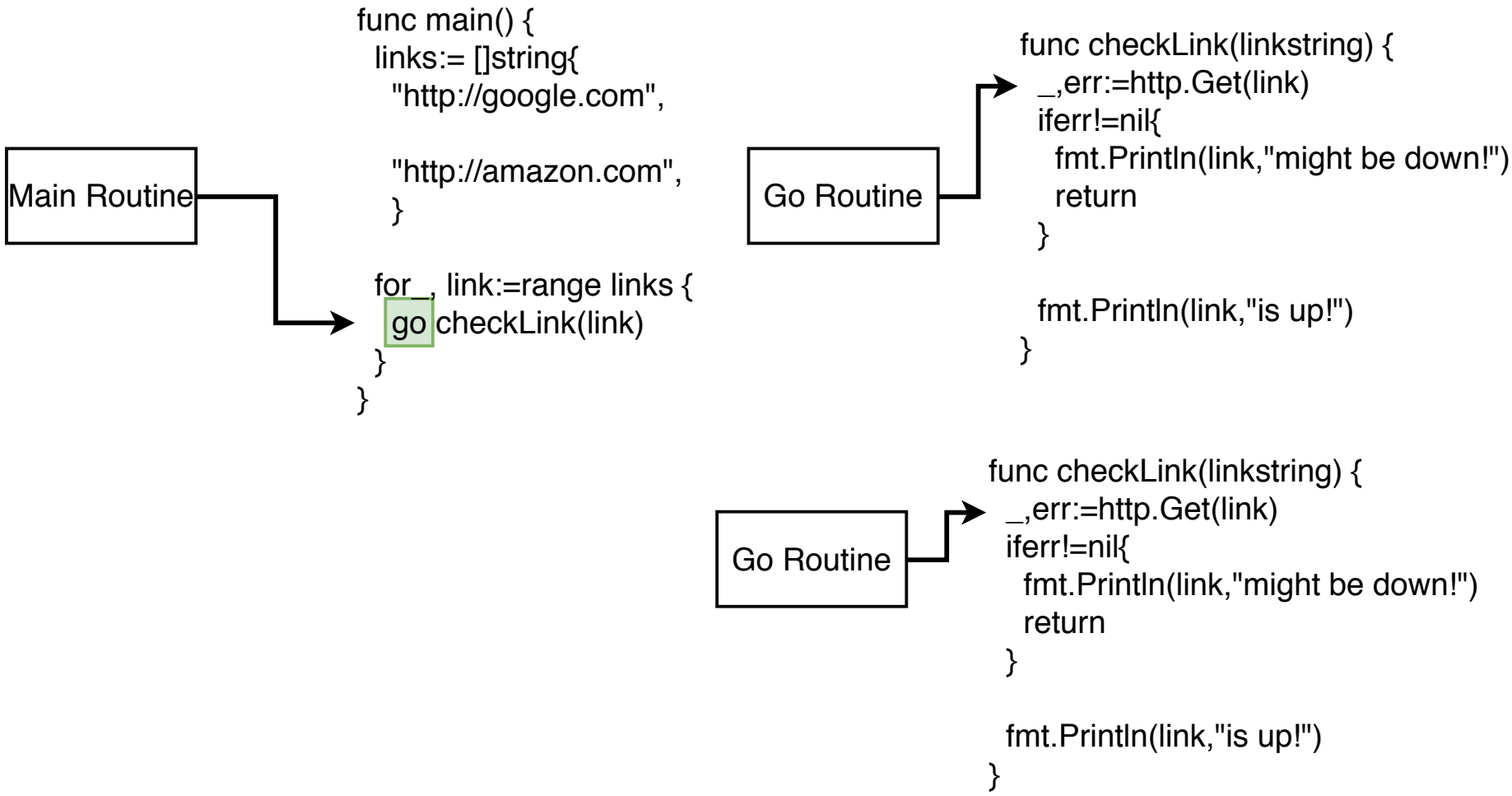

http://google.com
http://facebook.com
http://stackoverflow.com
http://golang.org
http://amazon.com
Take first link from slice
Make request
GET http://google.com
Wait for a response, log it
Take next link, make request
GET http://facebook.com
Wait for a response, log it
Repeat!

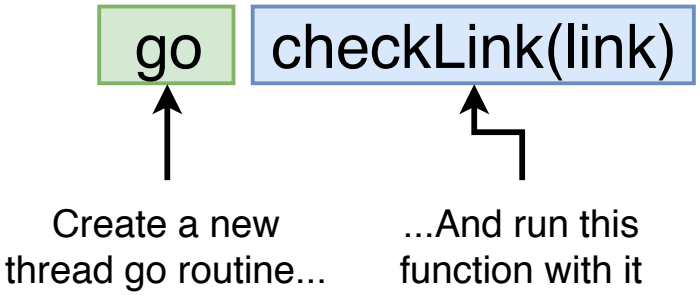

http://google.com
wait
http://facebook.com
wait
http://stackoverflow.com
wait
http://golang.org
wait
http://amazon.com

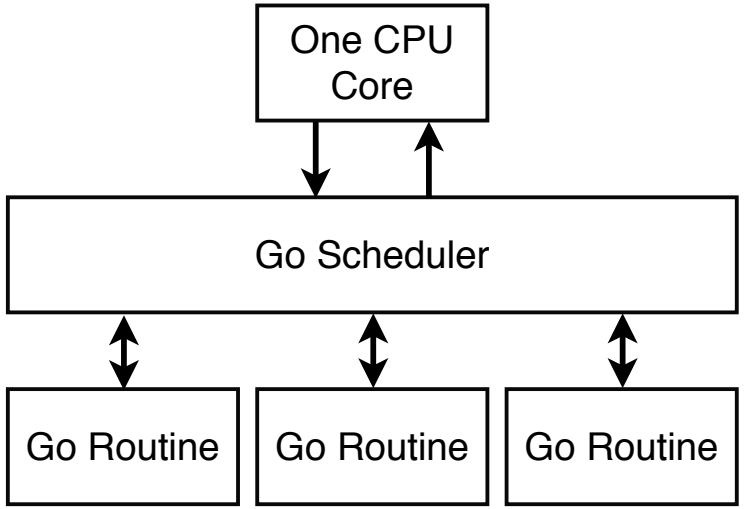

http://google.com
http://facebook.com
http://stackoverflow.com
http://golang.org
Make request
Make request
Make request
Make request
Print status
Print status
Print status
Print status
Time


Our Running Program
(a process)
Go Routine
Engine that
executes code

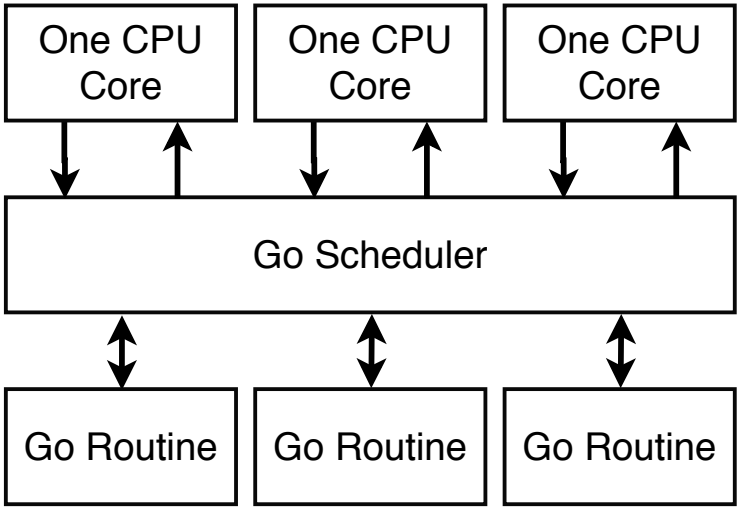

func main() {
links:= []string{
"http://google.com",
"http://amazon.com",
}
for_, link:=range links {
checkLink(link)
}
}
Main Go Routine
func checkLink(link string) {
_, err:= http.Get(link)
Blocking call!
if err !=nil {
fmt.Println(link, "might be down!")
return
}
fmt.Println(link, "is up!")
}


Main Routine
func main() {
links:= []string{
"http://google.com",
"http://amazon.com",
}
for _, link:=range links {
go checkLink(link)
}
}
Go Routine
func checkLink(linkstring) {
_,err:=http.Get(link)
iferr!=nil{
fmt.Println(link,"might be down!")
return
}
fmt.Println(link,"is up!")
}
Go Routine
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



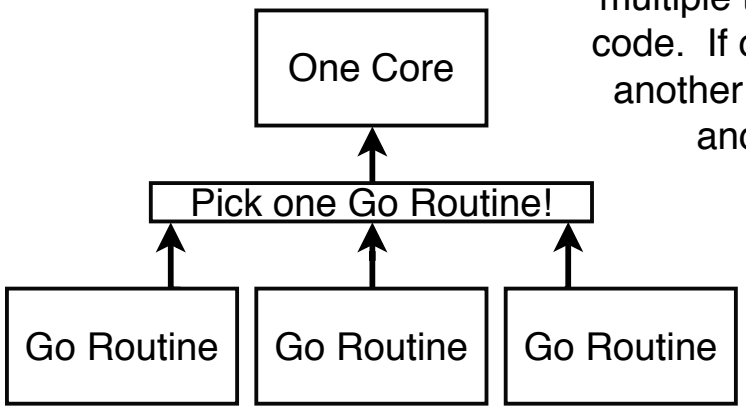
*Scheduler runs **one** routine until it finishes or makes a blocking call (like an HTTP request)*



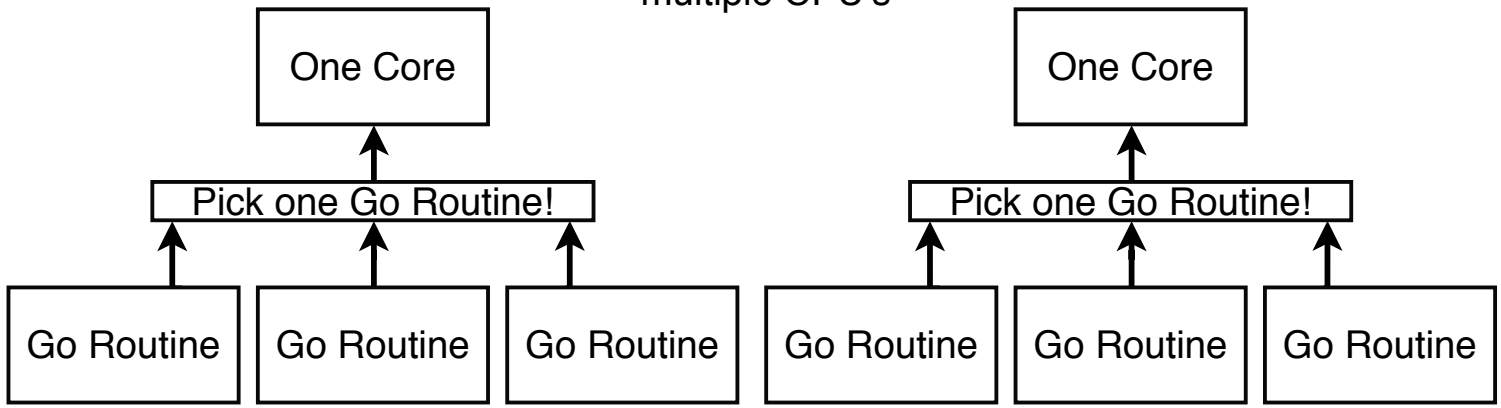
Scheduler runs one thread on each "logical" core

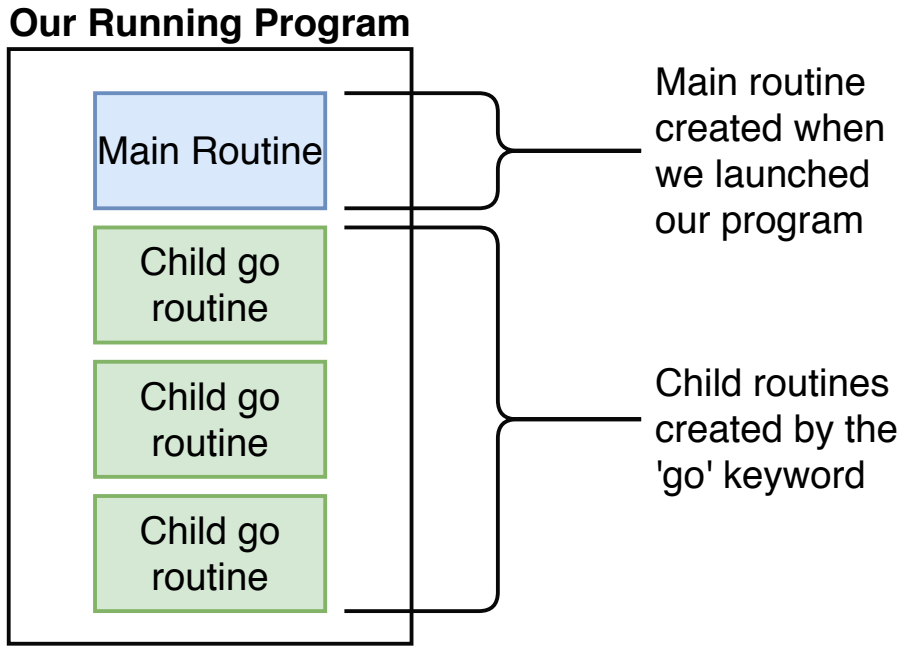
By default Go tries to use one core!

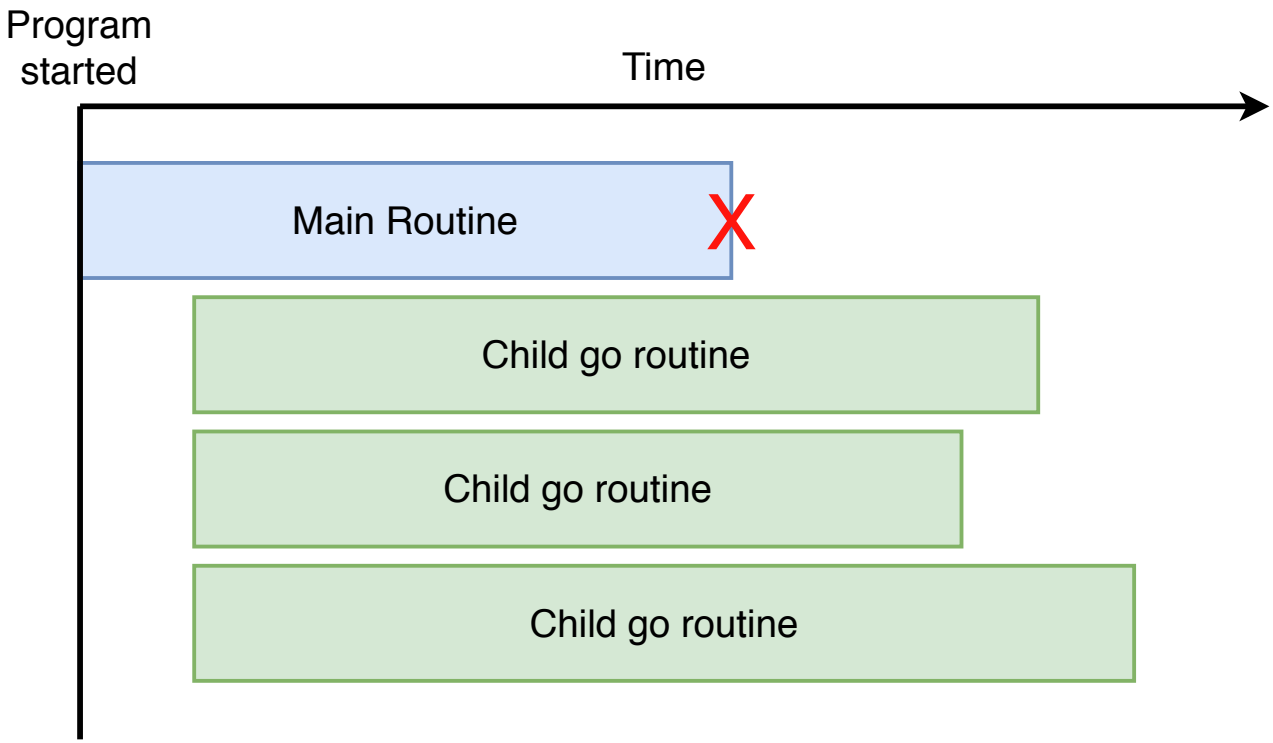
Concurrency - We can have multiple threads executing code. If one thread blocks, another one is picked up and worked on

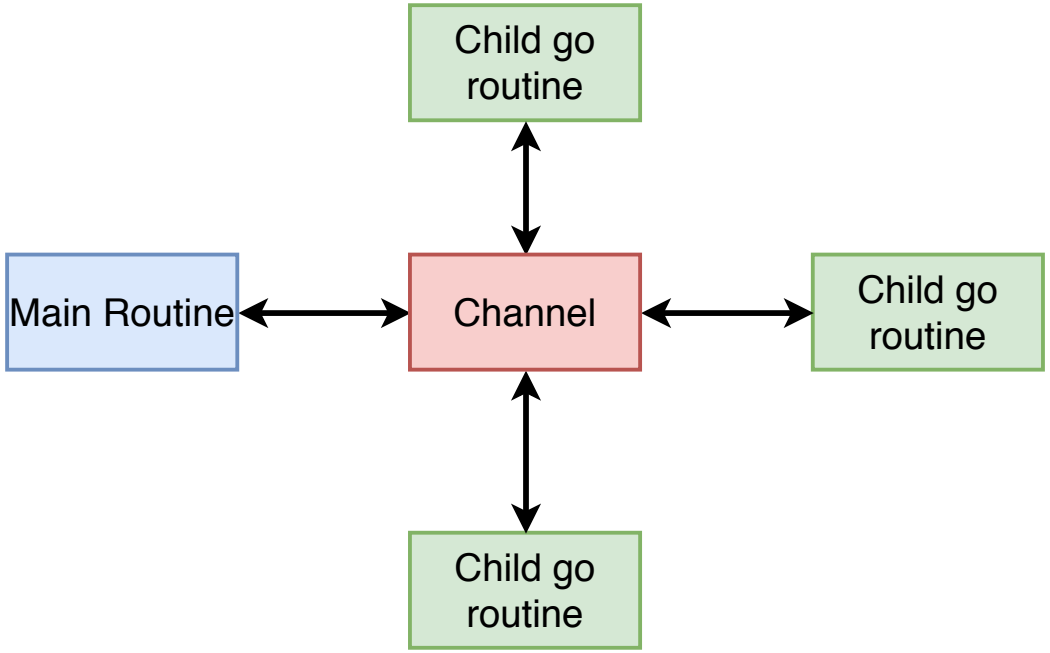


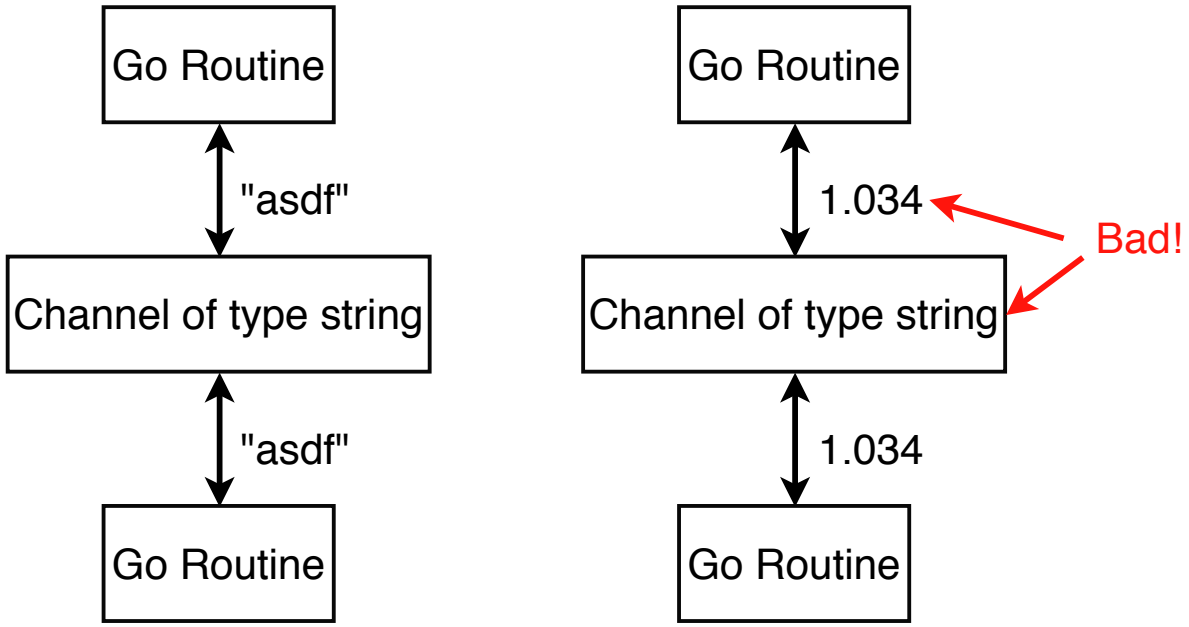
Parallelism - Multiple threads executed at the exact same time. Requires multiple CPU's

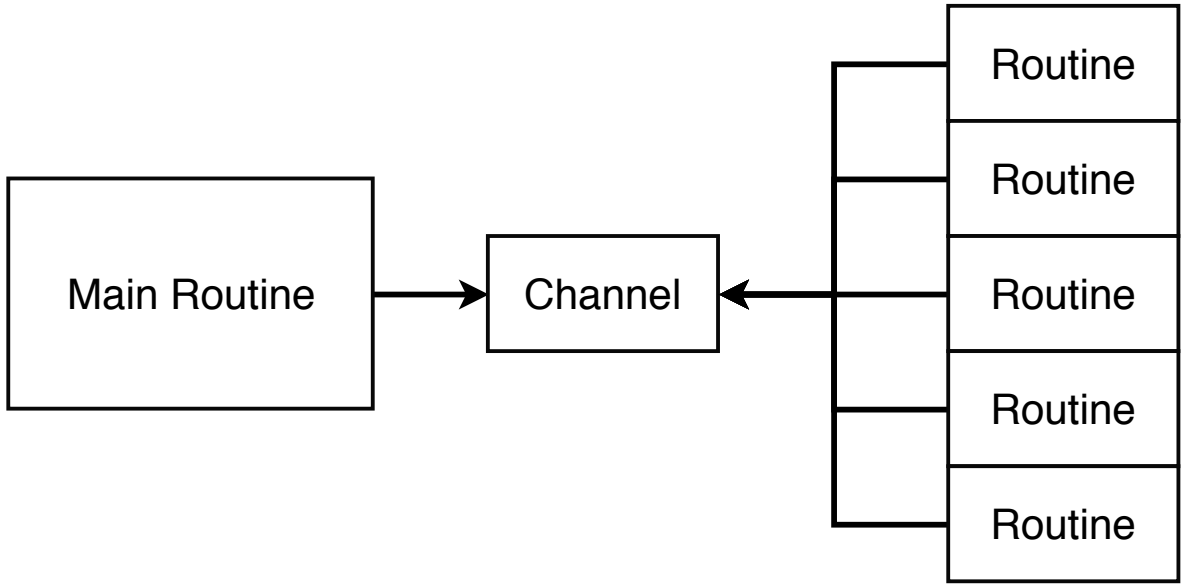



Our Running Program
Main Routine
Child go routine
Child go routine
Child go routine
Main routine created when we launched our program
Child routines created by the 'go' keyword

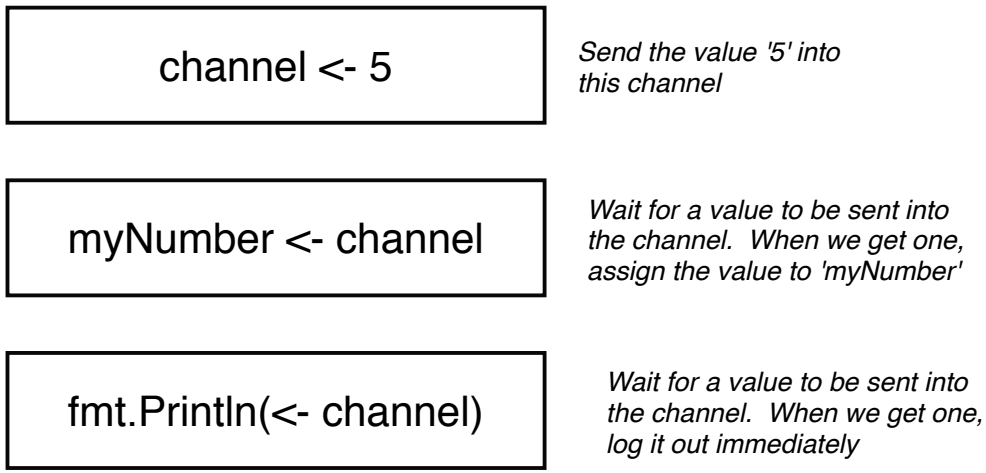

Program
started
Time
Main Routine
X
Child go routine
Child go routine
Child go routine


Child go routine
Main Routine
Channel
Child go routine
Child go routine


Go Routine
"asdf"
Channel of type string
"asdf"
Go Routine
Go Routine
1.034
Bad!
Channel of type string
1.034
Go Routine

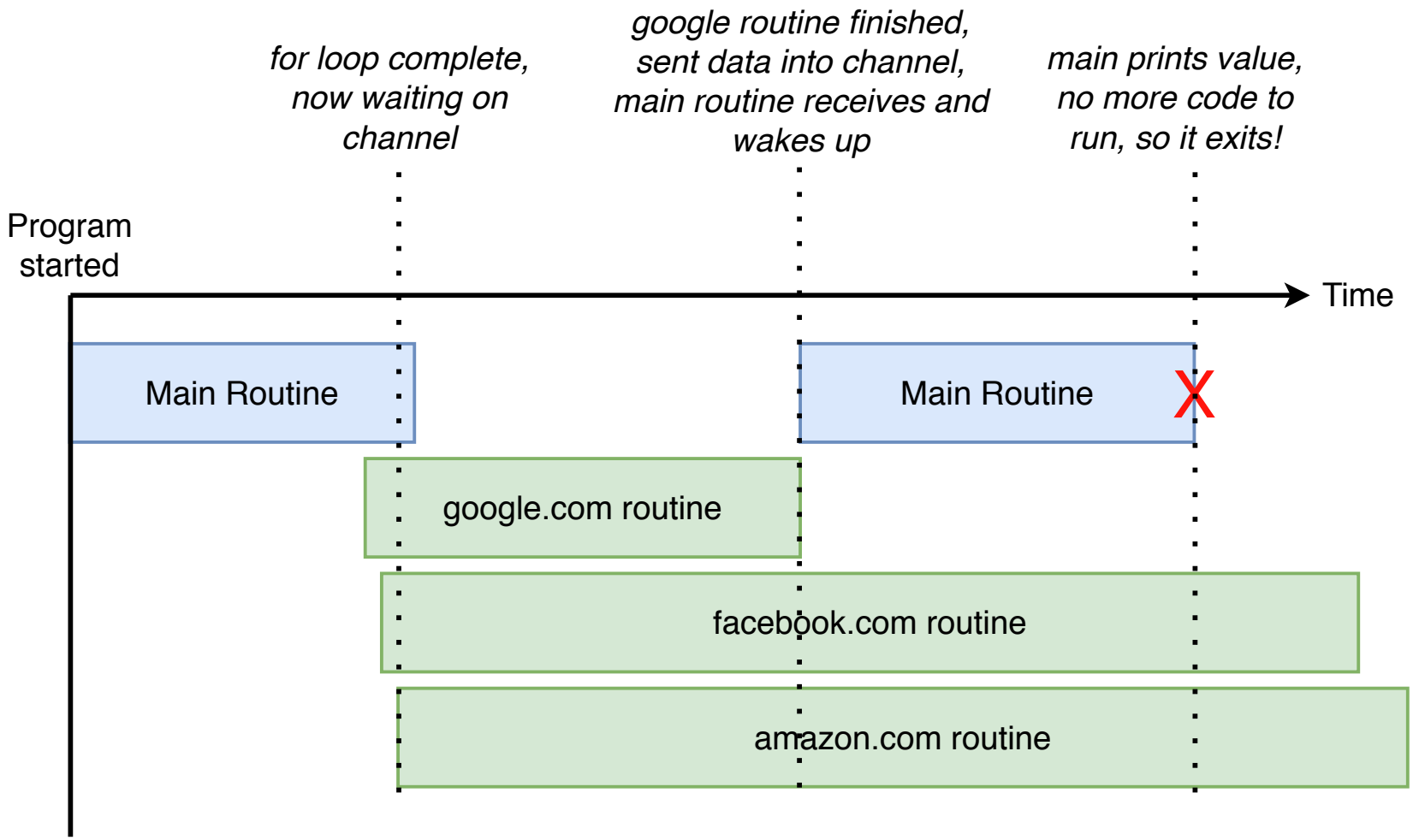

Main Routine
Channel
Routine
Routine
Routine
Routine
Routine

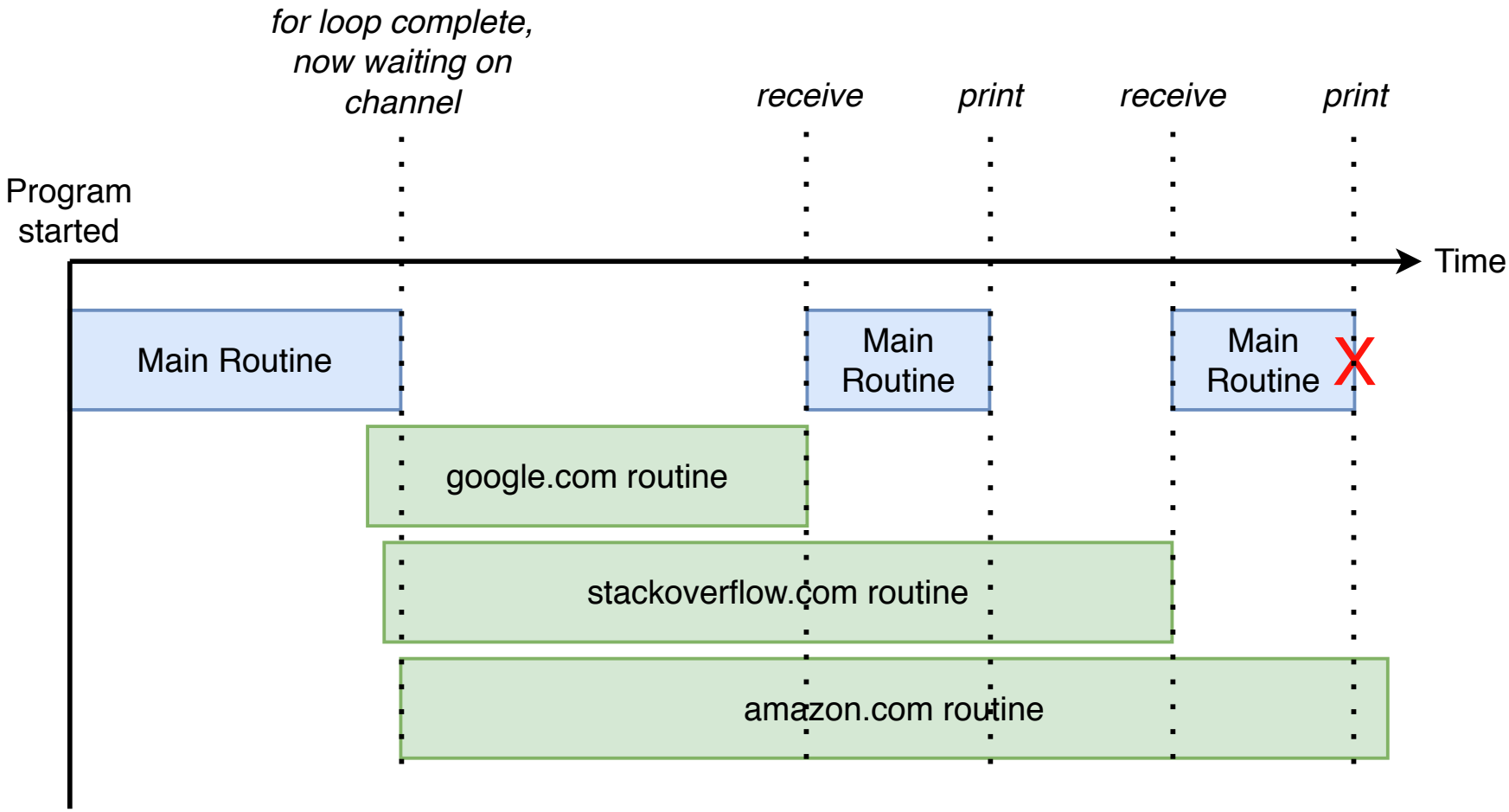
Sending Data with Channels

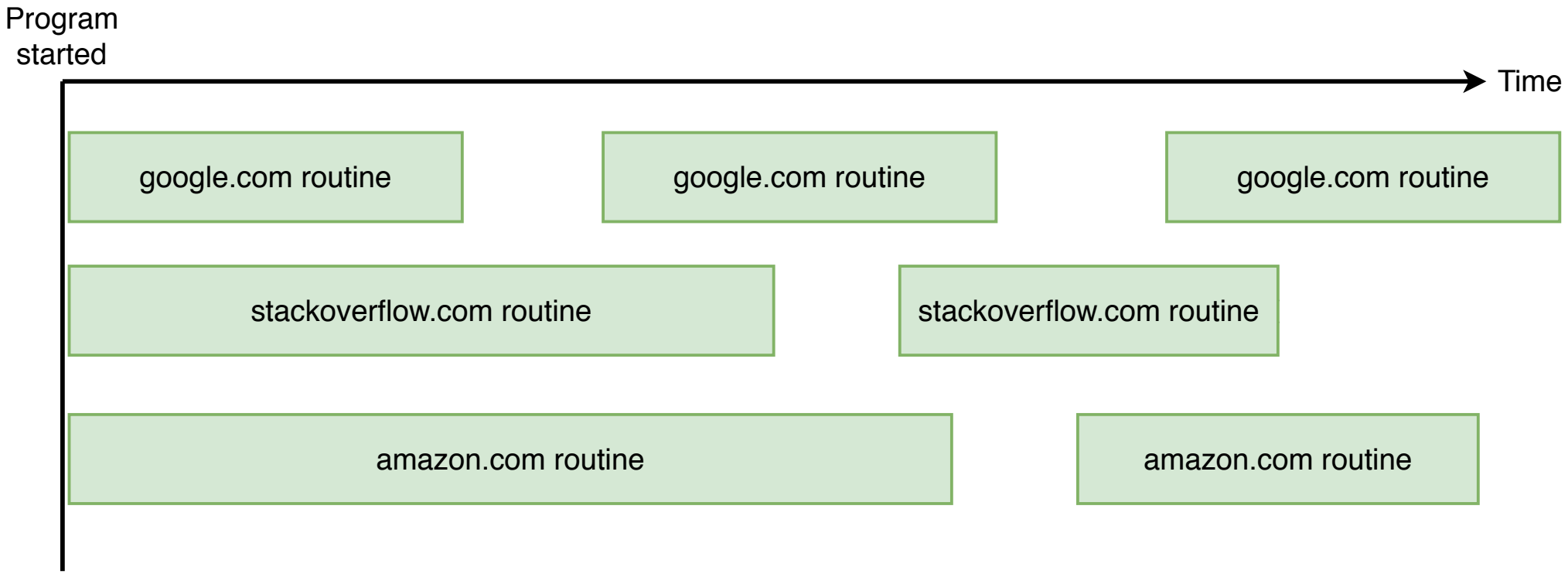


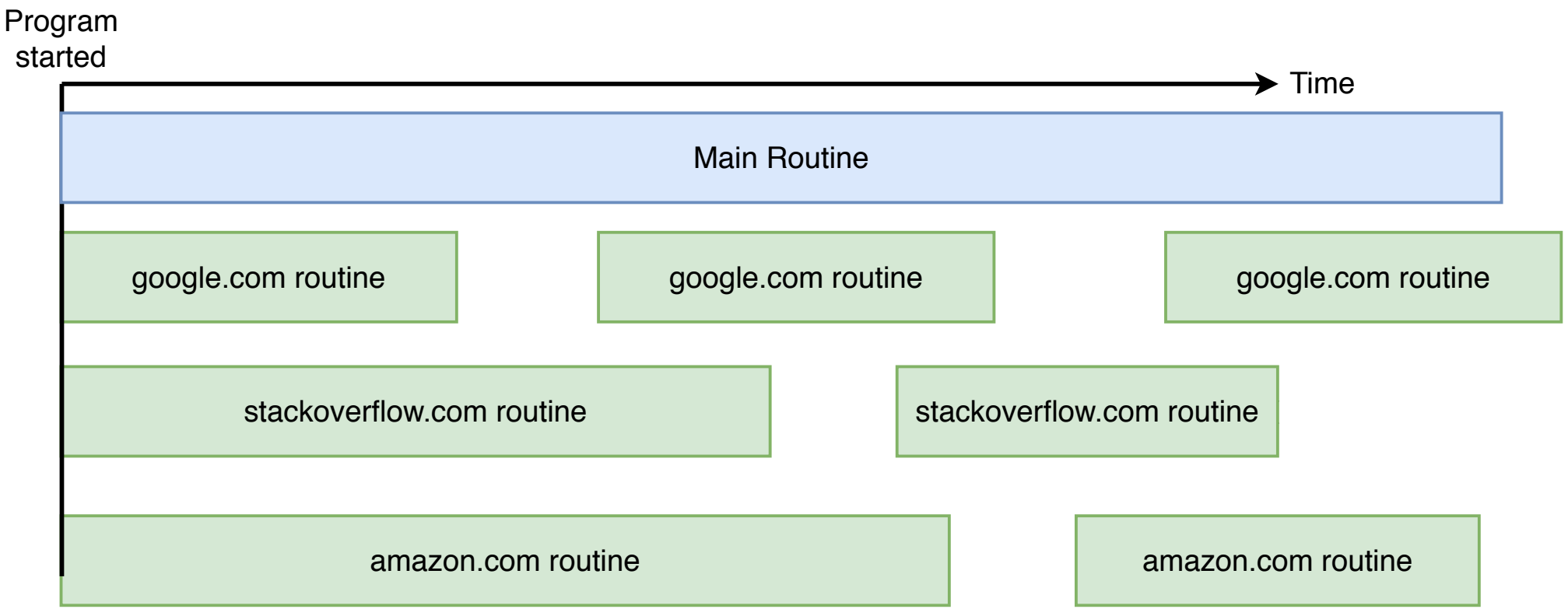

```
func main() {
  links:= []string{
    "http://google.com",

    "http://amazon.com",
    }

  c := make(chan string)

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

  fmt.Println(<- c)
}
```

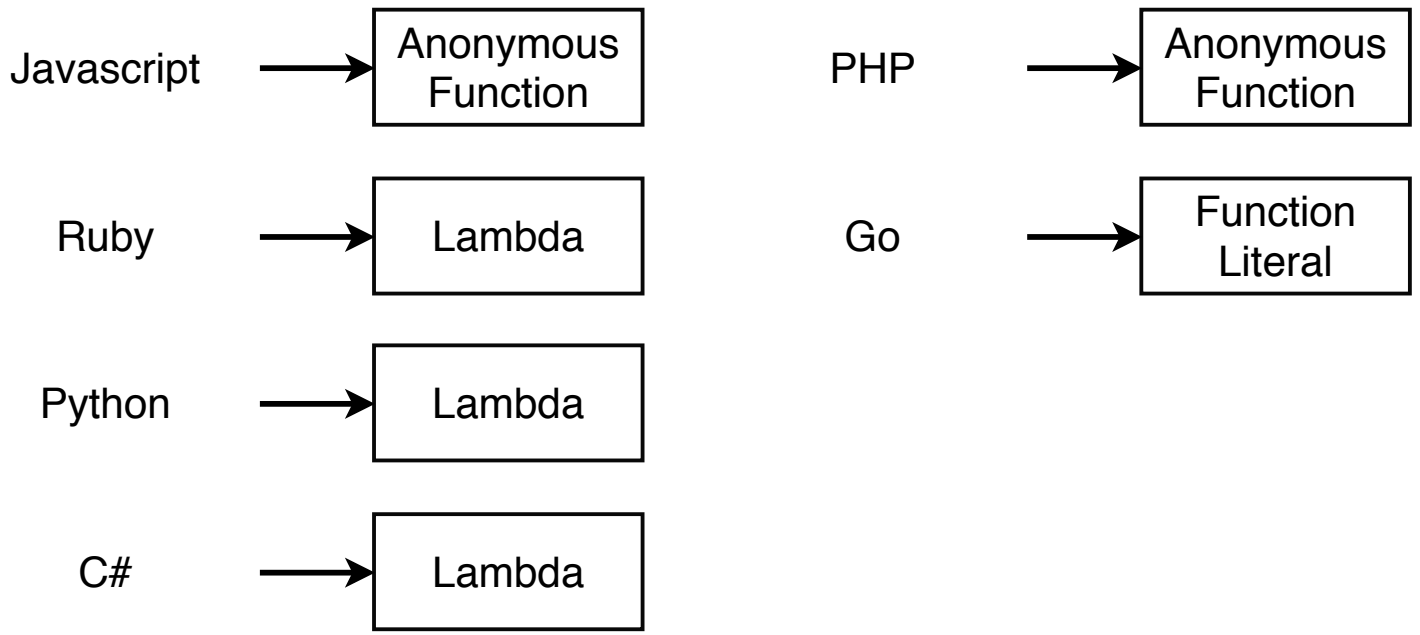

```
func checkLink(link string, c chan string) {
  _,err:=http.Get(link)
  if err!=nil{
    fmt.Println(link,"might be down!")
    c <-"Might be down I think"
    return
  }

  fmt.Println(link,"is up!")
  c <-"Yep its up"
}
```

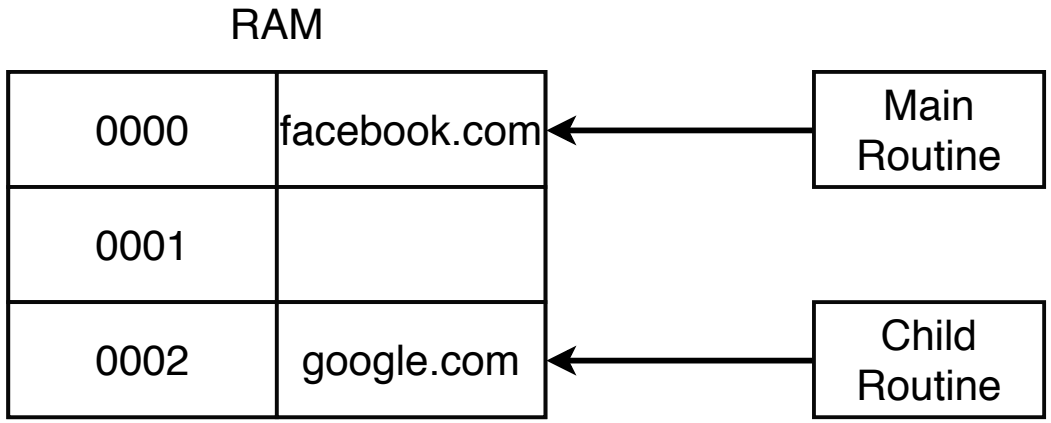

for loop complete, now waiting on channel
google routine finished, sent data into channel, main routine receives and wakes up
main prints value, no more code to run, so it exits!
Program started
Time
Main Routine
Main Routine
X
google.com routine
facebook.com routine
amazon.com routine


for loop complete, now waiting on channel
receive
print
receive
print
Program started
Time
Main Routine
Main Routine
Main Routine
X
google.com routine
stackoverflow.com routine
amazon.com routine


Program started
Time
google.com routine
google.com routine
google.com routine
stackoverflow.com routine
stackoverflow.com routine
amazon.com routine
amazon.com routine


Program started
Time
Main Routine
google.com routine
google.com routine
google.com routine
stackoverflow.com routine
stackoverflow.com routine
amazon.com routine
amazon.com routine


Javascript
Anonymous Function
PHP
Anonymous Function
Ruby
Lambda
Go
Function Literal
Python
Lambda
C#
Lambda


RAM
0000
facebook.com
0001
0002
google.com
Main Routine
Child Routine