

Concurrent Golang

Seokju Hong
PUBG Studio

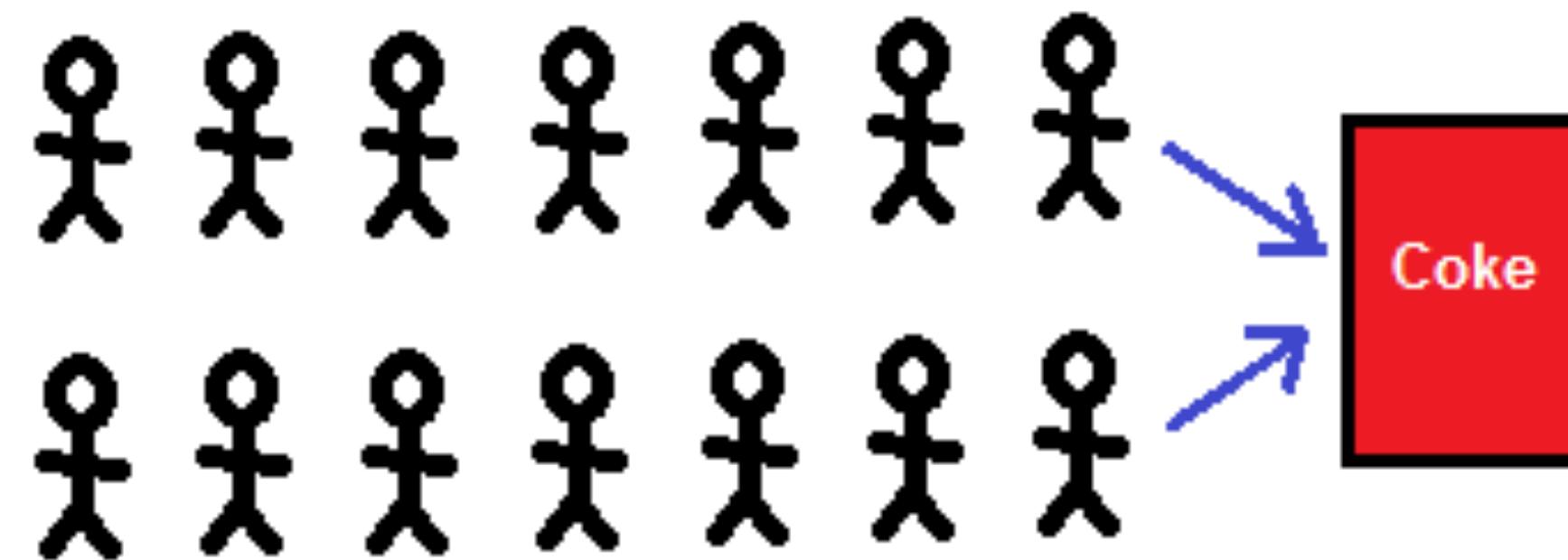
Contents

- Concurrency vs Parallelism
- Go ❤️ Concurrency
- Goroutines Internals, vs POSIX Thread
- Concurrency Patterns

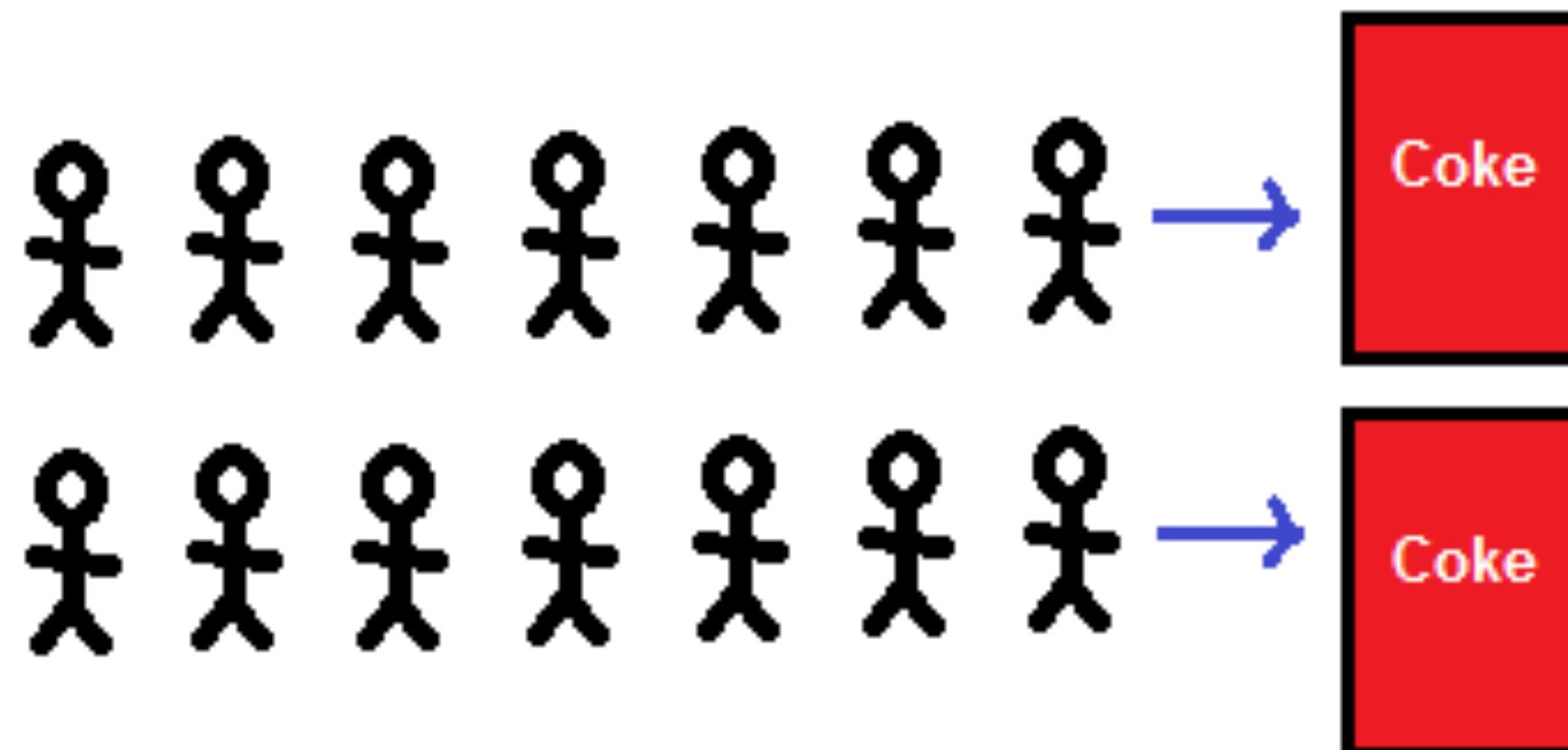
What is Concurrency?

- 뭐예요

What is Concurrency?



Concurrent: 2 queues, 1 vending machine

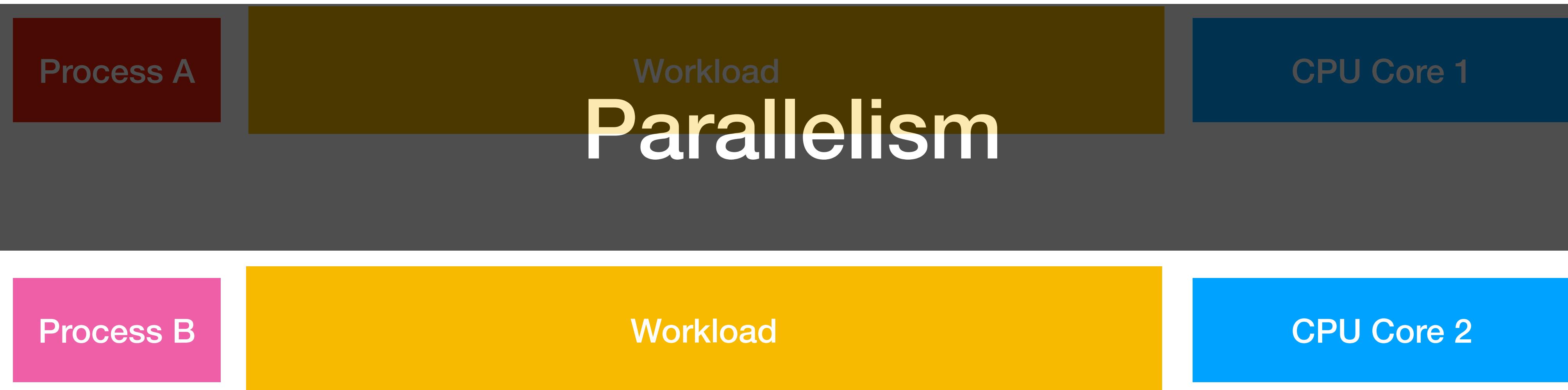


Parallel: 2 queues, 2 vending machines

What is Concurrency?



What is Concurrency?



What is Concurrency?



What is Concurrency?



Go ❤️ Concurrency

- 태생부터 Concurrency를 위해 태어난 언어가 Golang이다.
 - CSP Concurrency Model[1]에서 영향을 받음.
 - 함수 호출 앞에 "go" 두 글자 붙여서 concurrent process를 띄울 수 있음.
 - first-class channel이 존재하여 프로세스간 통신 채널을 변수로 들고 다닐 수 있음.
 - Race detector가 존재함[4].

Concurrent Programming in Go

- Goroutine을 사용한다.
- Goroutine은
 - 함수 앞에 "go" 두 글자만 붙이면 main routine과 별도의 routine으로 실행된다.
 - 가볍다. 100000개 띄워도 된다.
 - 엄청 가벼운 thread 정도로 생각해도 된다.

Concurrent Programming in Go

```
1 package main
2
3 import "fmt"
4
5 func routine(i int) {
6     |   fmt.Println(i)
7 }
8 func main() {
9     |   go routine(1)
10 }
```

Channels

- Channel은
 - first-class Rendezvous Point이다.
 - 데이터를 goroutine에서 goroutine으로 넘기는 징검다리 역할을 한다.
 - Synchronous하다. Goroutine이 channel ops(<-)에서 block된다.

Simple Channel Example

```
package main

import (
    "fmt"
    "time"
    "runtime"
)

var flag int = 0
func routine(ch chan bool) {
    fmt.Println("A very very long task...")
    time.Sleep(1 * time.Second)
    flag = 1
    ch <- true
}
func main() {
    runtime.GOMAXPROCS(2)
    ch := make(chan bool)
    go routine(ch)
    v := <-ch
    fmt.Println(flag, v)
}
```

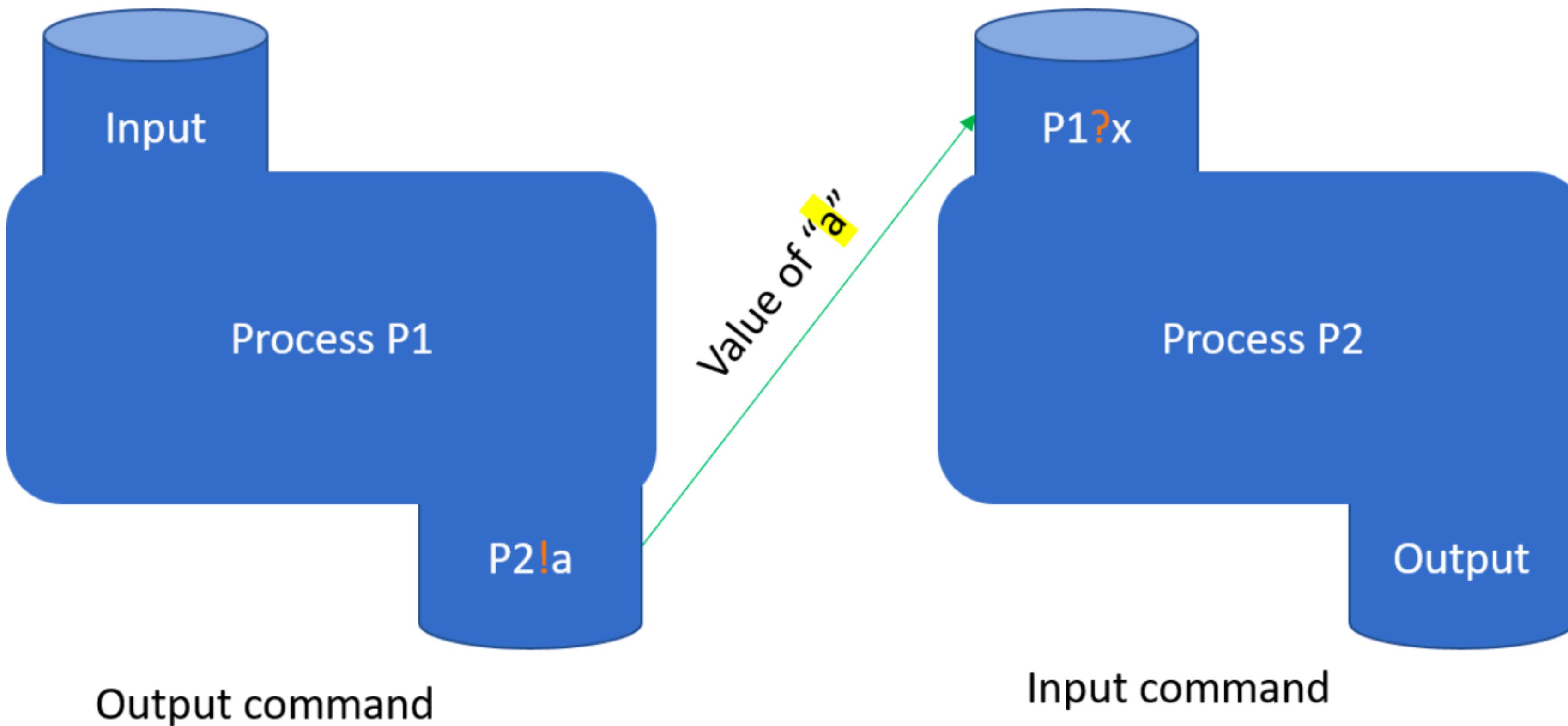
Simple Channel Example

```
package main

import (
    "fmt"
    "time"
    "runtime"
)

var flag int = 0
func routine(ch chan bool) {
    fmt.Println("A very very long task...")
    time.Sleep(1 * time.Second)
    flag = 1
    ch <- true
}
func main() {
    runtime.GOMAXPROCS(2)
    ch := make(chan bool)
    go routine(ch)
    v := <-ch
    fmt.Println(flag, v)
}
```

Simple Channel Example



답은 2천만

```
package main

import (
    "fmt"
    "time"
    "runtime"
)

var sum int = 0
func routine() {
    for i := 0; i < 10000000; i++ {
        sum = sum + 1
    }
}
func main() {
    runtime.GOMAXPROCS(2)
    go routine()
    go routine()
    time.Sleep(1 * time.Second)
    fmt.Println(sum)
}
```

답은 2천만

```
[suckzoo@suckzoo: /Users/suckzoo]
>>> go run aa.go
10127868
[suckzoo@suckzoo: /Users/suckzoo]
>>> go run aa.go
10030282
```

답은 2천만

```
package main

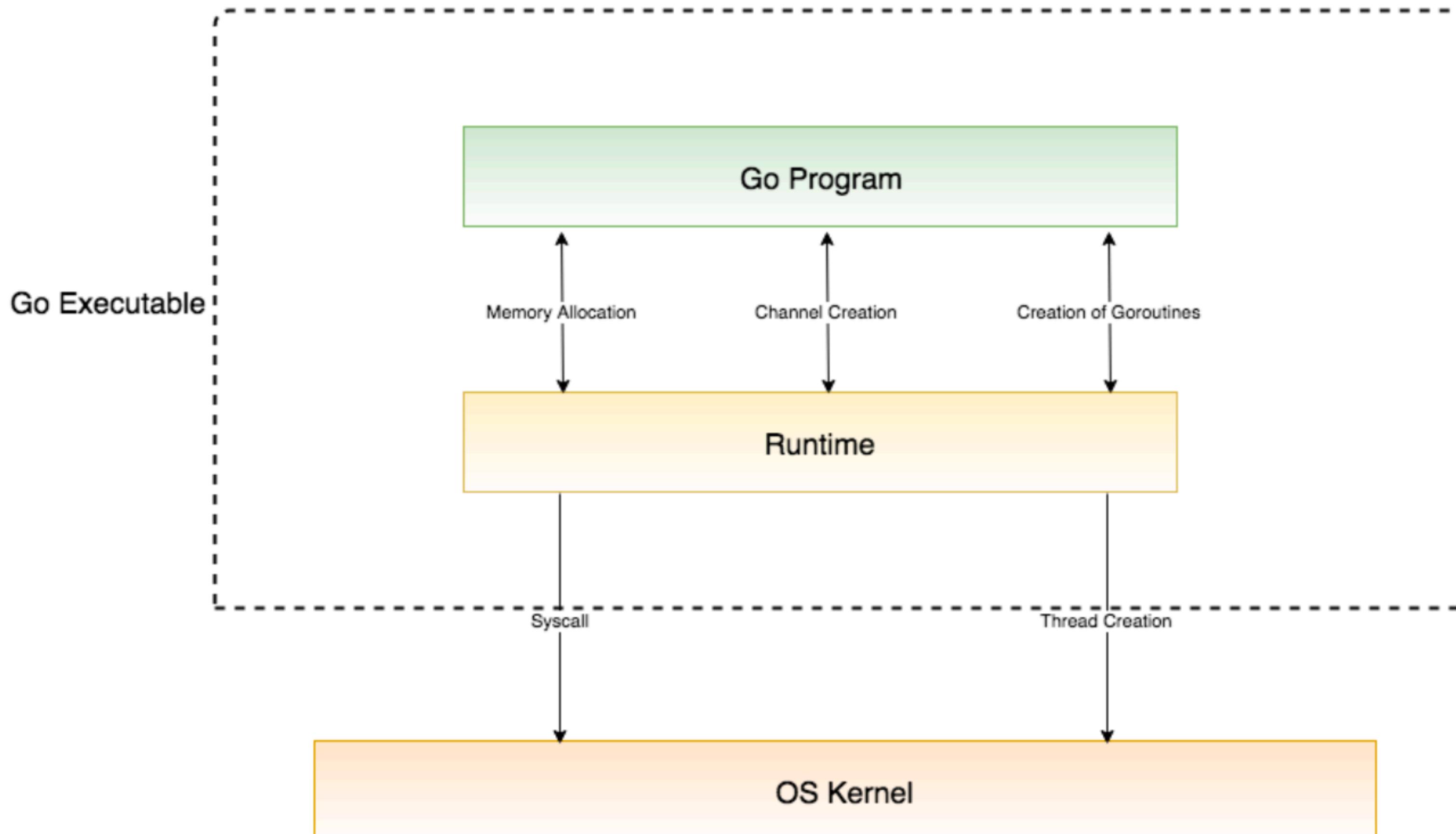
import (
    "fmt"
    "time"
    "runtime"
)

var sum int = 0
func routine(ch chan bool) {
    for i := 0; i < 10000000; i++ {
        <- ch
        sum++
        ch <- true
    }
}
func main() {
    runtime.GOMAXPROCS(2)
    ch := make(chan bool)
    go routine(ch)
    go routine(ch)
    ch <- true
    time.Sleep(1 * time.Second)
    fmt.Println(sum)
}
```

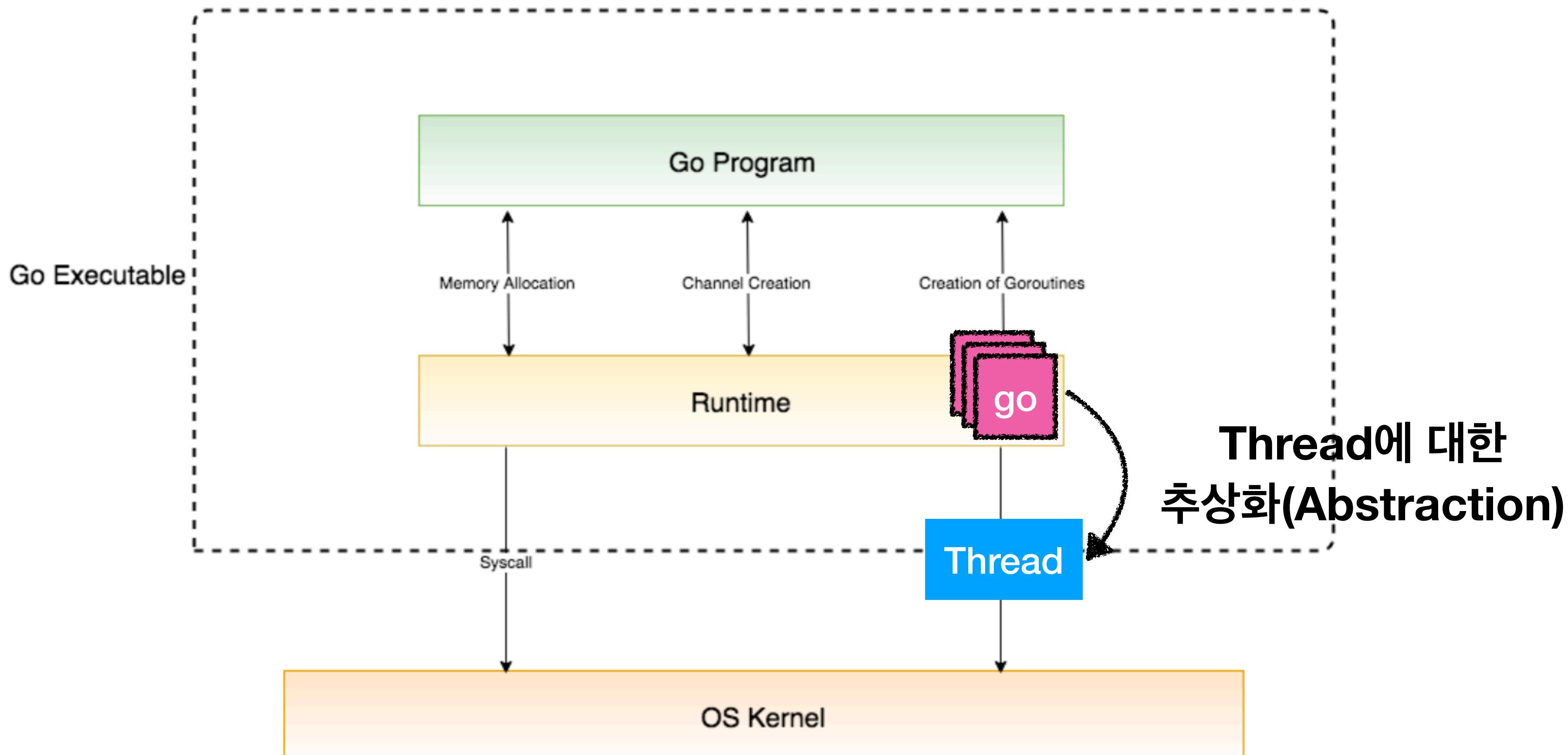
답은 2천만

```
[suckzoo@suckzoo: /Users/suckzoo]
>>> go run aa.go
20000000
```

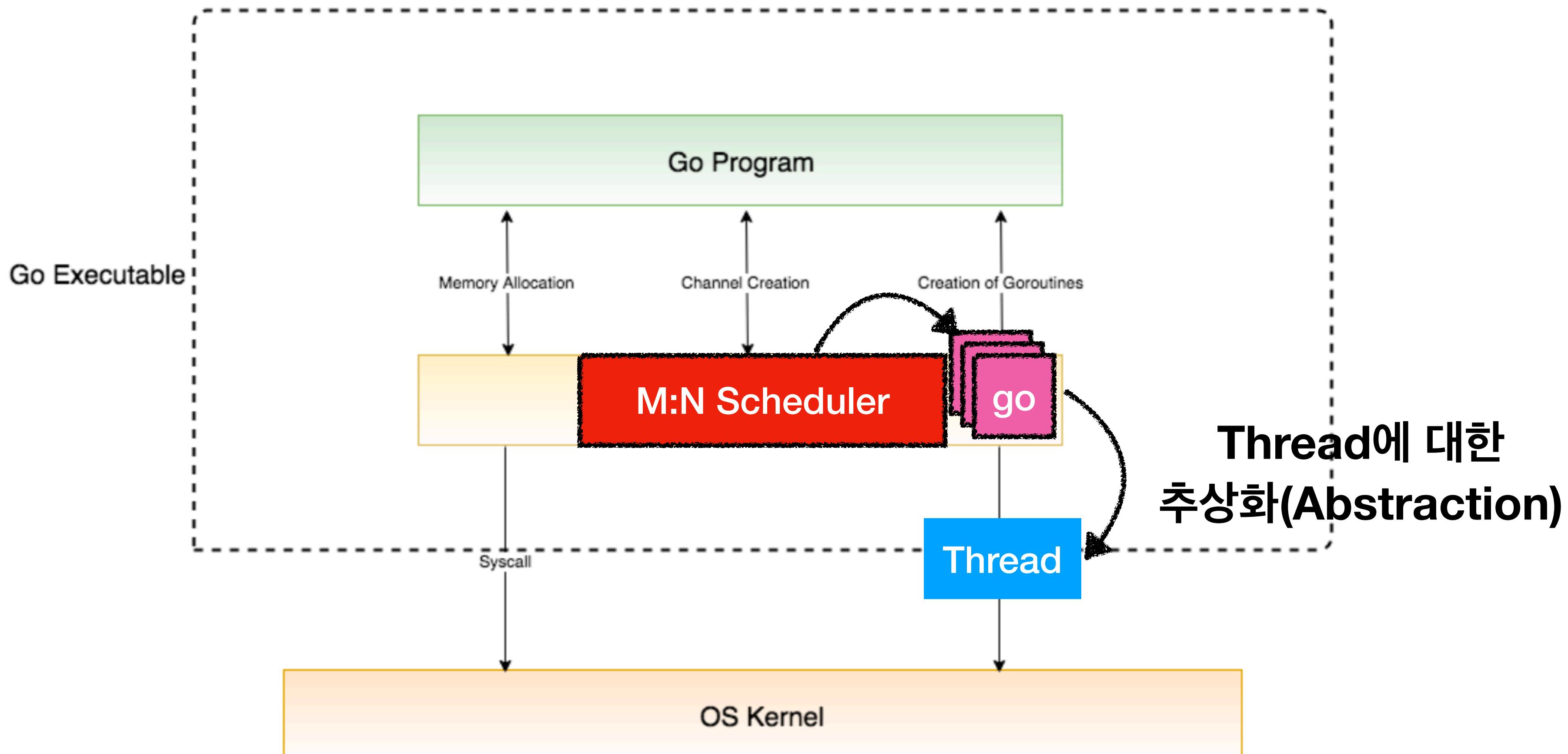
Go Runtime



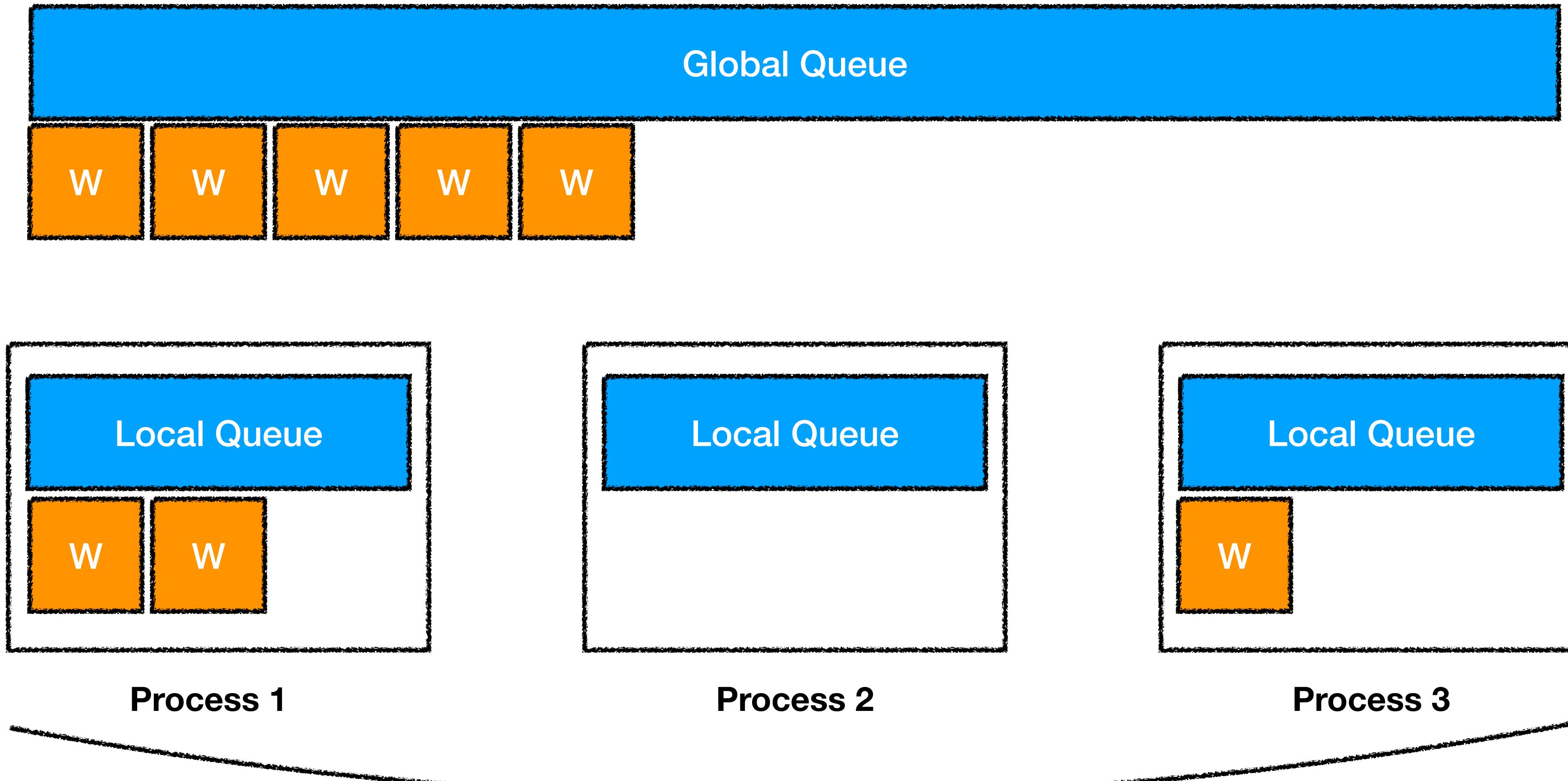
Go Runtime



Go Runtime

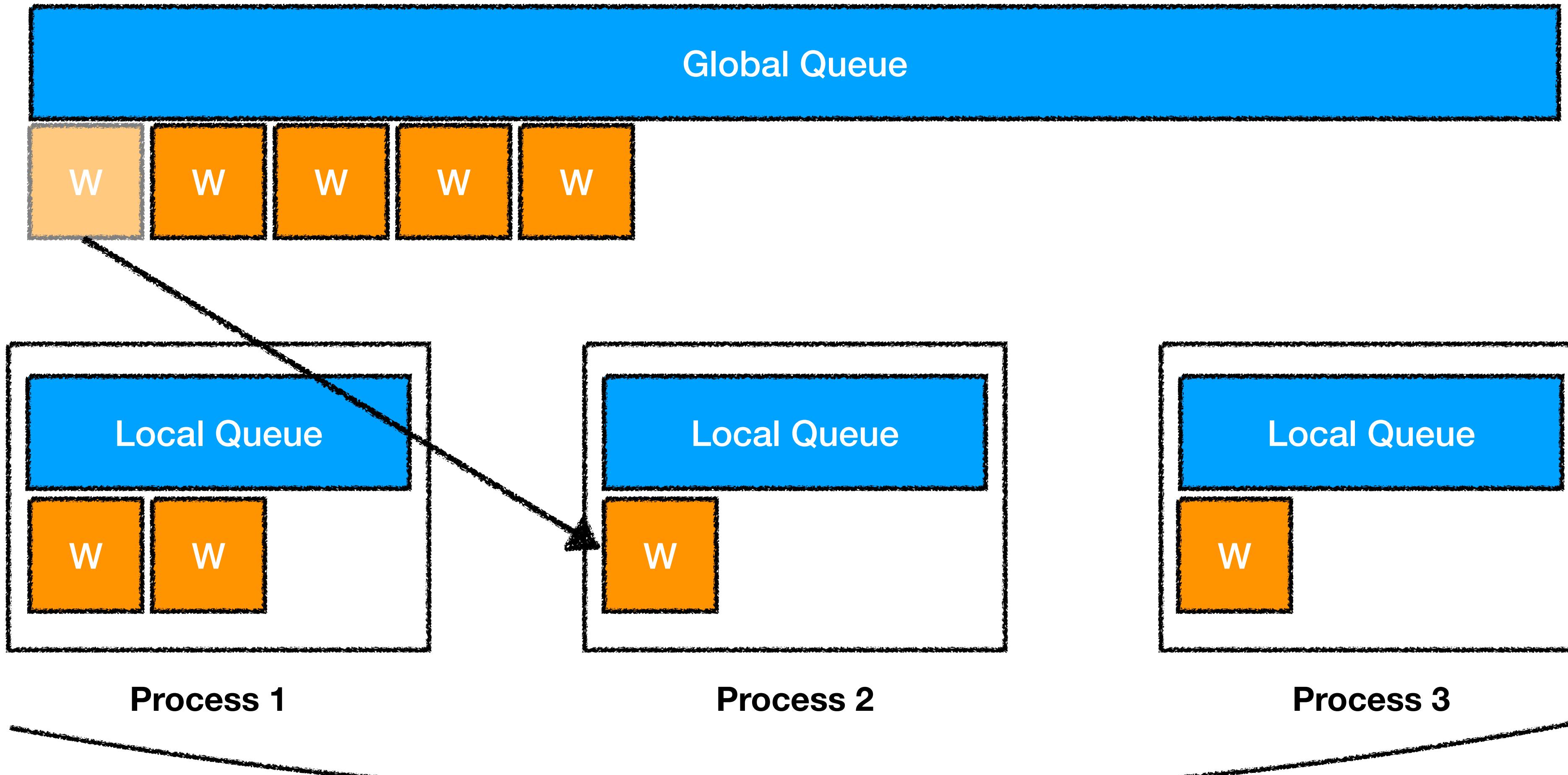


Go Scheduler



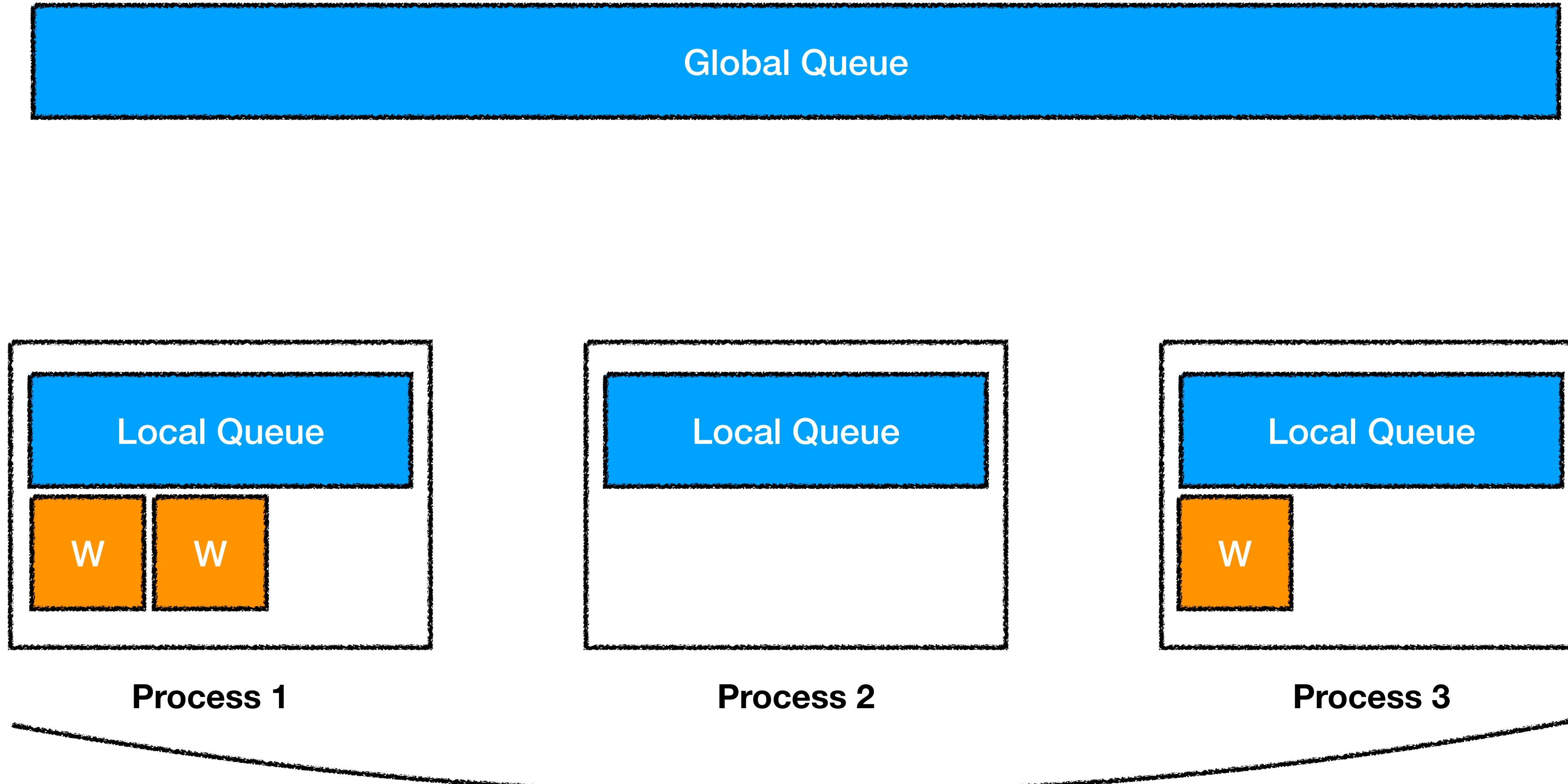
<= N 개, `runtime.GOMAXPROCS(N)`으로 지정 가능

Go Scheduler



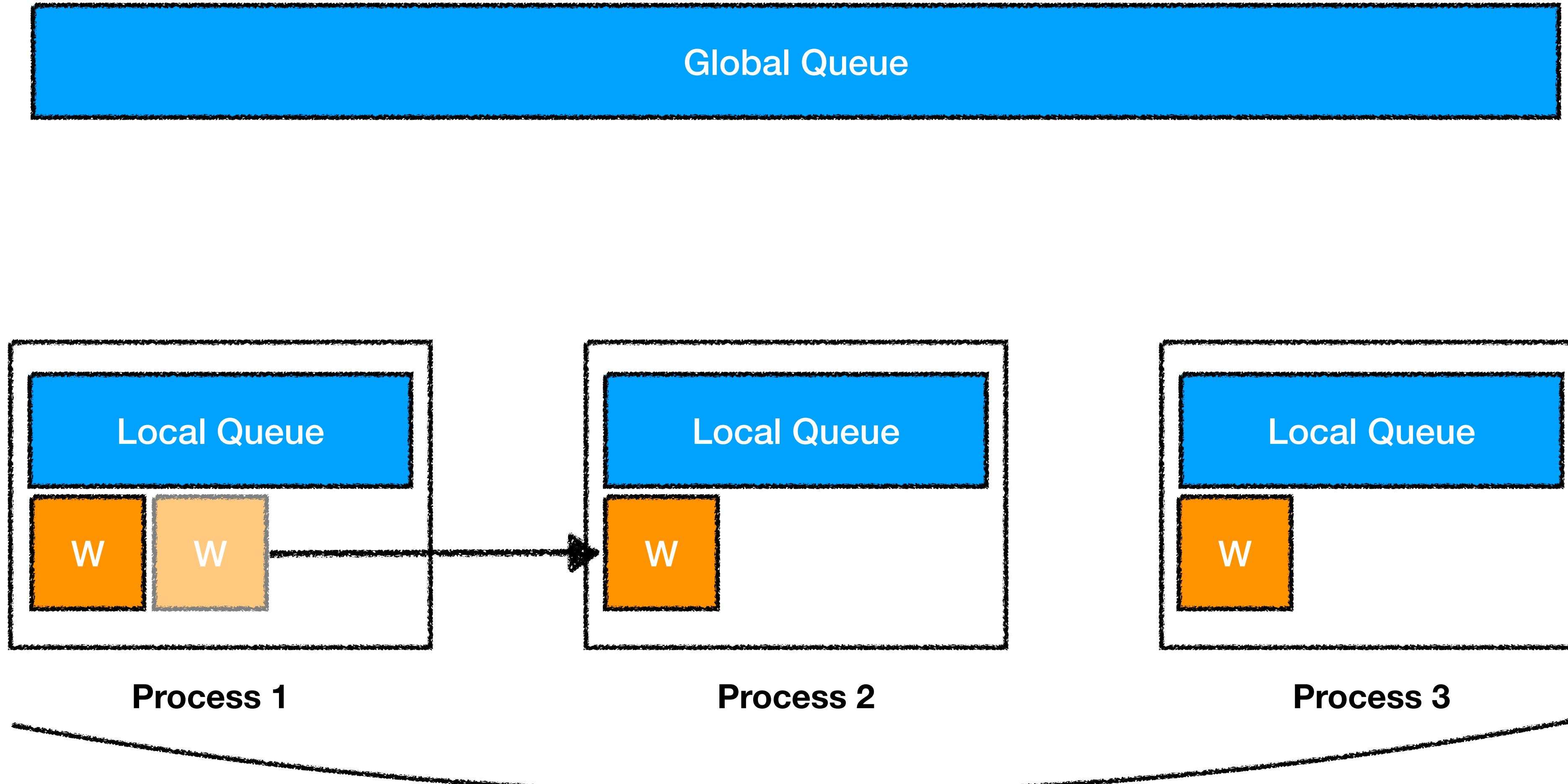
<= N 개, `runtime.GOMAXPROCS(N)`으로 지정 가능

Go Scheduler



<= N 개, `runtime.GOMAXPROCS(N)`으로 지정 가능

Go Scheduler



<= N 개, `runtime.GOMAXPROCS(N)`으로 지정 가능

Go Scheduler

- goroutine들을 협동적(cooperatively)으로 스케줄링한다.
- 아래의 경우 실행되던 goroutine이 block되며 다른 goroutine에게 yield된다.
 - channel operations (<-)
 - go func()
 - syscall
 - garbage collection cycle

Goroutine vs POSIX Thread

- Goroutine
 - Go runtime이 만드는 thread에 대한 abstraction이다.
 - 메모리 사용량이 2KB 수준이다.
 - OS를 거칠 필요가 없어 create/destroy가 싸다.
 - 협동적(cooperatively)으로 스케줄링된다.
 - 컨텍스트 스위치가 싸다.
 - OS를 직접 거치지 않고, 레지스터 3개 정도만 바꾸면 된다.
- Pthread
 - clone(2)를 통해 OS가 만들어주는 프로세스다.
 - 메모리 사용량이 1MB 수준이다.
 - OS를 거쳐야 해서 create/destroy가 비싸다.
 - 선점적(preemptively)으로 스케줄링된다.
 - 컨텍스트 스위치가 비싸다.
 - 바꿔줘야 할 레지스터 개수가 많다.
 - OS를 직접 거쳐야 한다.

Go Concurrency Patterns

- Sharing Memory by Communicating
- Generator Patterns
- Multiplexing
- Select
 - Timeout
 - Quit Channel and Cleanups

"Share Memory by Communicating"

- "Do not communicate by sharing memory; instead, share memory by communicating."
- 아래의 과제를 해결해보자.
 - goroutine A와 B를 띄운다.
 - A는 B에게 영문 string을 주고, B는 A에게 strings.ToUpper(s) 하여 돌려준다.

Generator Pattern

- Concurrent하게 실행시킬 job을 함수 안에서 실행
- 새 channel을 만들어 해당 channel을 통해 job의 결과를 확인
- 함수는 channel을 return하여 결과를 외부에서 확인할 수 있게 함

Merging Channels

- 여러 channel에서 오는 결과물을 한 channel에서 받게 할 수 있을까?

Select

- Select는 channel의 switch-case라고 생각하면 편하다.
- 각 case는 channel에 대한 communication이다.
- 모든 case의 channel communication이 evaluate된다.
- 여러 channel이 communication 할 수 있다면, 랜덤하게 하나 골라서 진행된다.
- default case가 있고 ready인 channel이 없으면 바로 default절이 실행된다.

Closed Channel

- Communicate 하려는 채널이 closed면,
 - $v, ok := <- ch$: 해당 타입의 "0"이 바로 나온다. $ok == \text{false}$
 - $ch <- x$: panic
 - $\text{close}(ch)$: panic

Nil Channel

- Communicate 하려는 채널이 nil이면,
 - <-ch: 영원히 block된다.
 - ch <- x: 영원히 block된다.
 - close(ch): panic

Daisy Chain

- 서로 물고있는 goroutine 100000개 만들어봅시다

Getting Stacktrace

- panic시 모든 goroutine의 stack trace를 얻을 수 있다.
(GOTRACEBACK=all 설정 시)
- deadlock이 발생해도 잡을 수 있다.

Race Detector

- go의 내장 race detector가 존재한다.
- race가 발생하는지 여부를 -race 플래그를 통해 확인할 수 있다.

Example: BRDM v1

- 스텝블록이 주어진다.
- 스텝을 모두 실행시키자.

Example: BRDM v2

- 스텝블록이 주어진다.
- 스텝을 모두 실행시키자.
- 스텝끼리의 연관성이 없어 동시에 실행해도 된다.

Example: BRDM v2.1

- 스텝블록이 주어진다.
- 스텝을 모두 실행시키자.
- 스텝끼리의 연관성이 없어 동시에 실행해도 된다.
- 모든 스텝에 500ms의 제한시간이 주어진다.

Example: BRDM v3

- 스텝블록이 주어진다.
- 스텝을 모두 실행시키자.
- 스텝끼리의 연관성이 없어 동시에 실행해도 된다.
- 모든 스텝에 1초의 제한시간이 주어진다.
- 스텝이 모두 idempotent해서 여러번 실행해도 된다.
- 스텝을 실행시킬 replica가 여러 벌 있다. 이 중 하나라도 먼저 끝나면 그 결과를 사용해도 좋다.

Example: BRDM v4

- 패치 리퀘스트가 주어지면 정해진 스텝블록을 실행시킨다.
- 외부에서 5초 뒤 차단모드로 돌입하는 Close() 콜을 한다.
- Close()를 호출한 부분에서 마지막 에러를 알 수 있어야한다.
- 차단모드에서는 리퀘스트를 받아도 동작하지 않아야 한다.
- 차단모드 진입 후 10초 뒤 서비스가 종료된다.

이 모든 것이

- lock, condition variable 없이 만들어졌다
- 물론, lock 쓰는 게 더 쉬울 때가 있다.
 - 그런 여러분들을 위해 "sync" 패키지가 있다.

References

- [1] Communicating sequential processes (Hoare, 1978)
- [2] <https://blog.golang.org/codelab-share>
- [3] <https://godoc.org/github.com/thomas11/csp>
- [4] <https://blog.golang.org/race-detector>
- [5] <https://medium.com/@riteeksrivastava/a-complete-journey-with-goroutines-8472630c7f5c>
- [6] <https://www.youtube.com/watch?v=f6kdp27YZs> (Go Concurrency Patterns, Rob Pike, Google I/O 2012)
- [7] <https://www.youtube.com/watch?v=QDDwwePbDtw> (Advances Go Concurrency Patterns, Sameer Ajmani, Google I/O 2013)