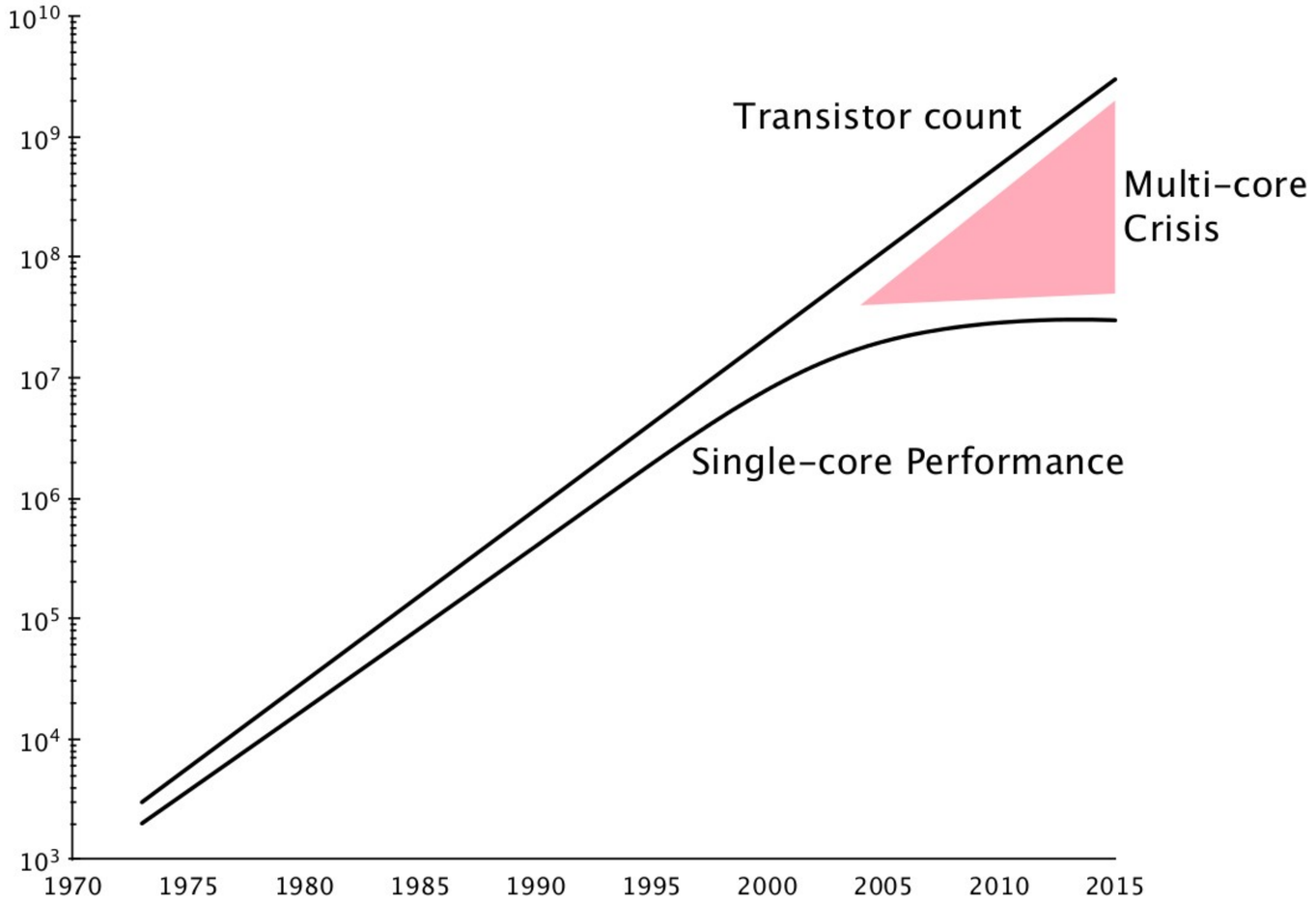
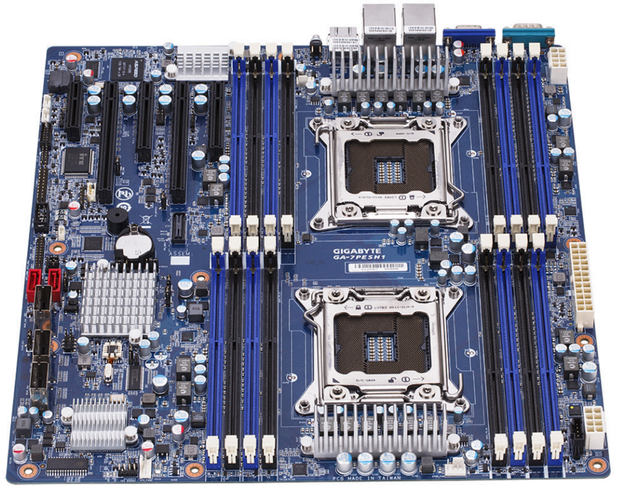
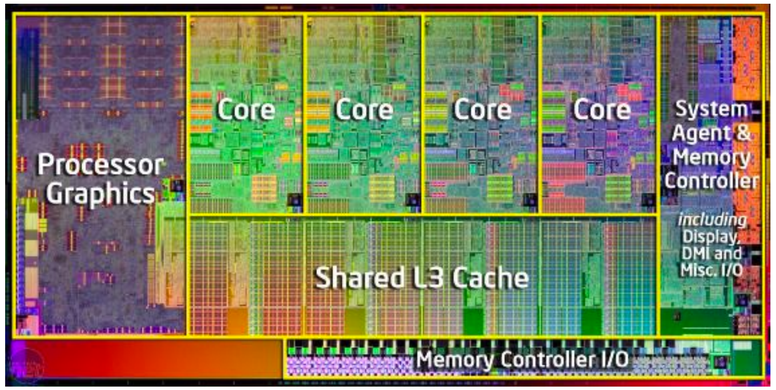
Go Rocks!

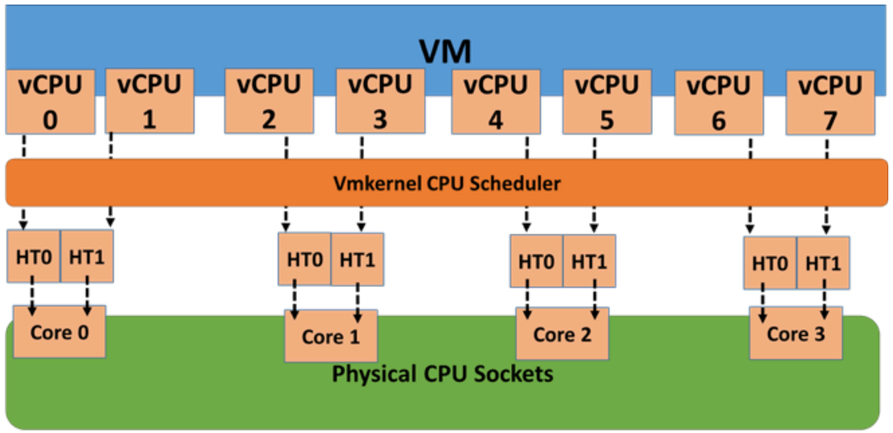
Moore's law


10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
10^{5}
10^{4}
10^{3}
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
Transistor count
Multi-core
Crisis
Single-core Performance

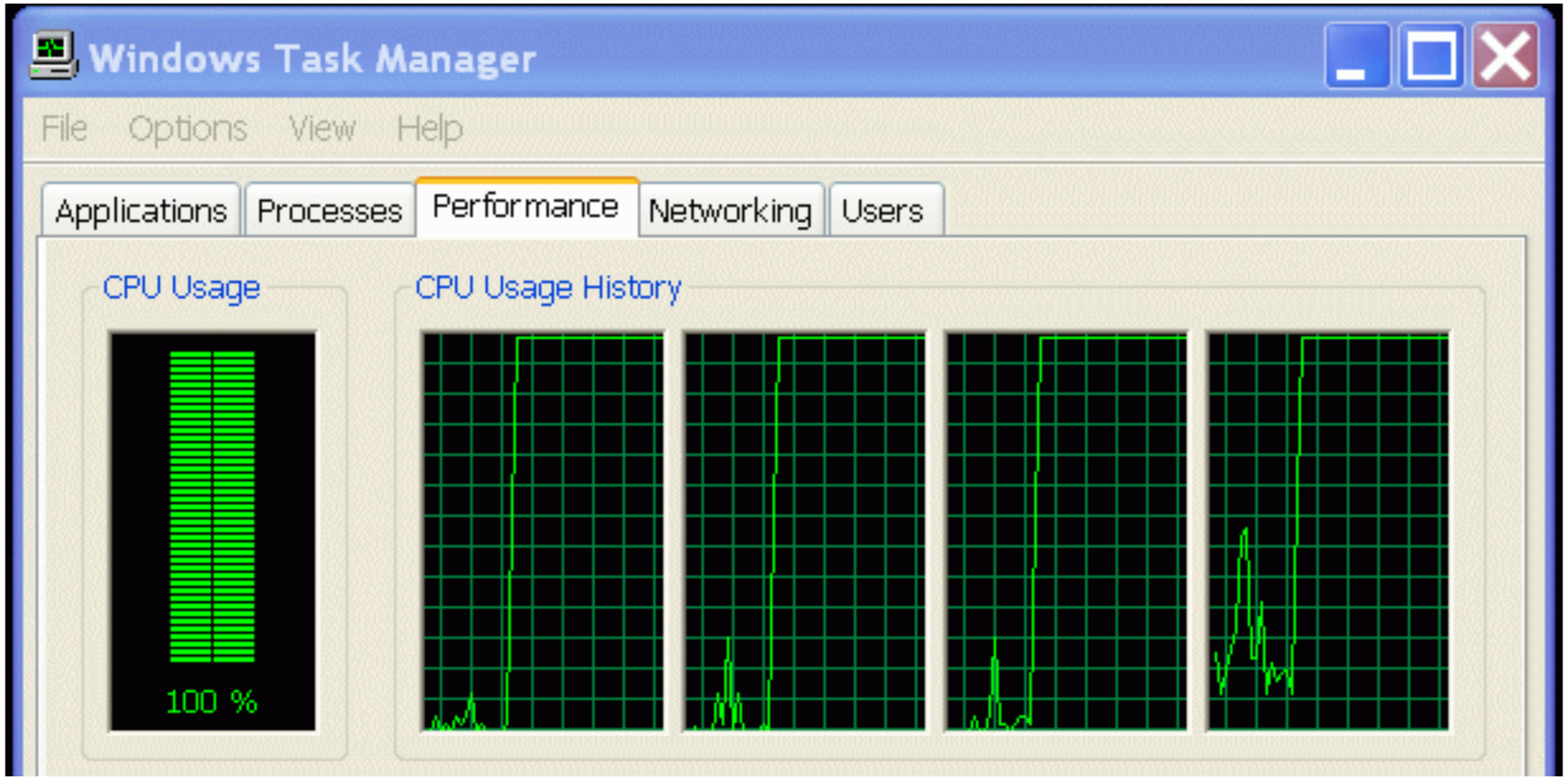
The modern world is parallel



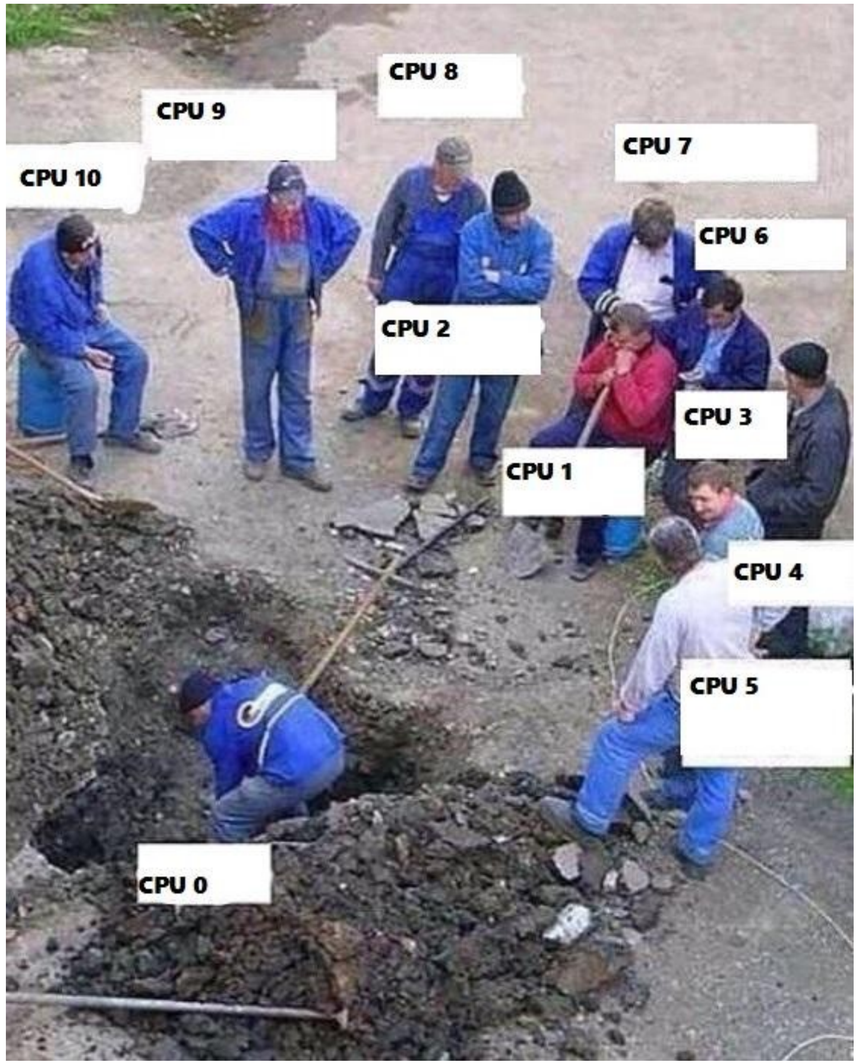




Hyper-Threading


Windows Task Manager
File
Options
View
Help
Applications
Processes
Performance
Networking
Users
CPU Usage
100 %
CPU Usage History

이론

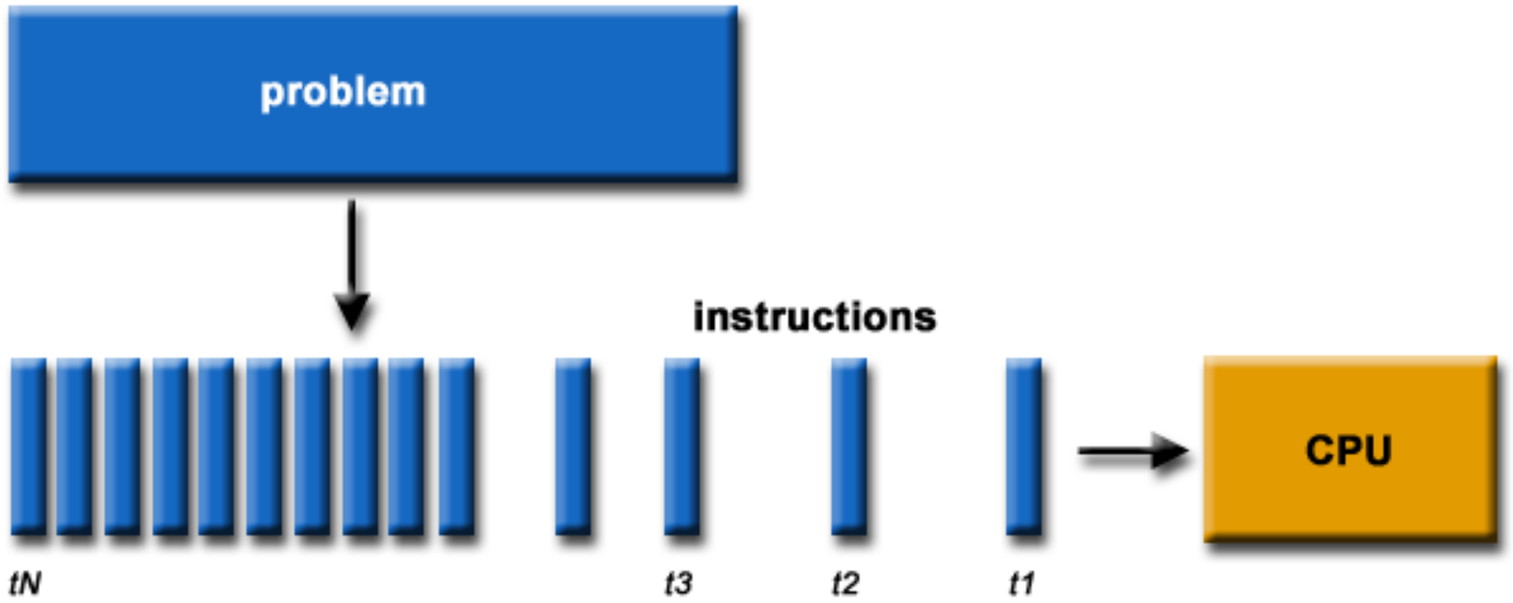


현실

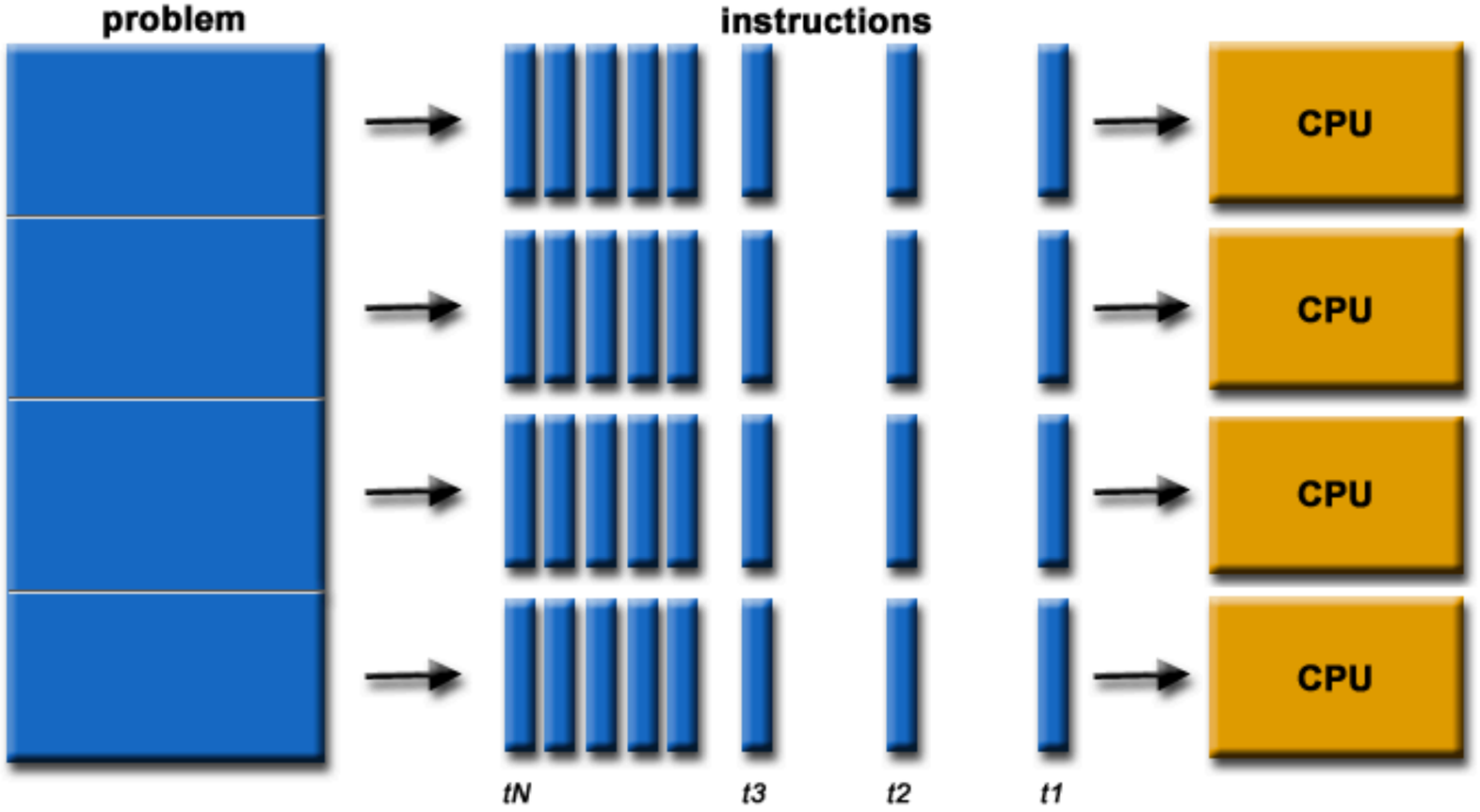

웨건웨건
나는
참견쟁이,
스피드 웨건!
반갑다!!
내 소개를
하도록
하지!!!
런던 빈민가
에서부터 널
따라왔지!
설명을
시작해볼까?

Contents

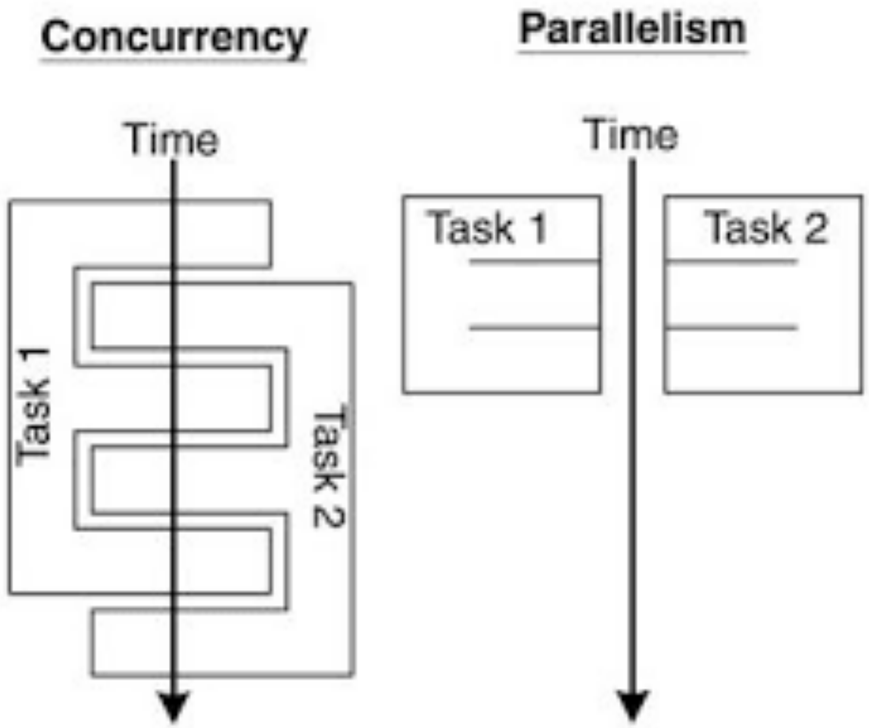
Concurrency



Parallelism



Concurrency vs. Parallelism



Concurrency is not parallelism

Concurre

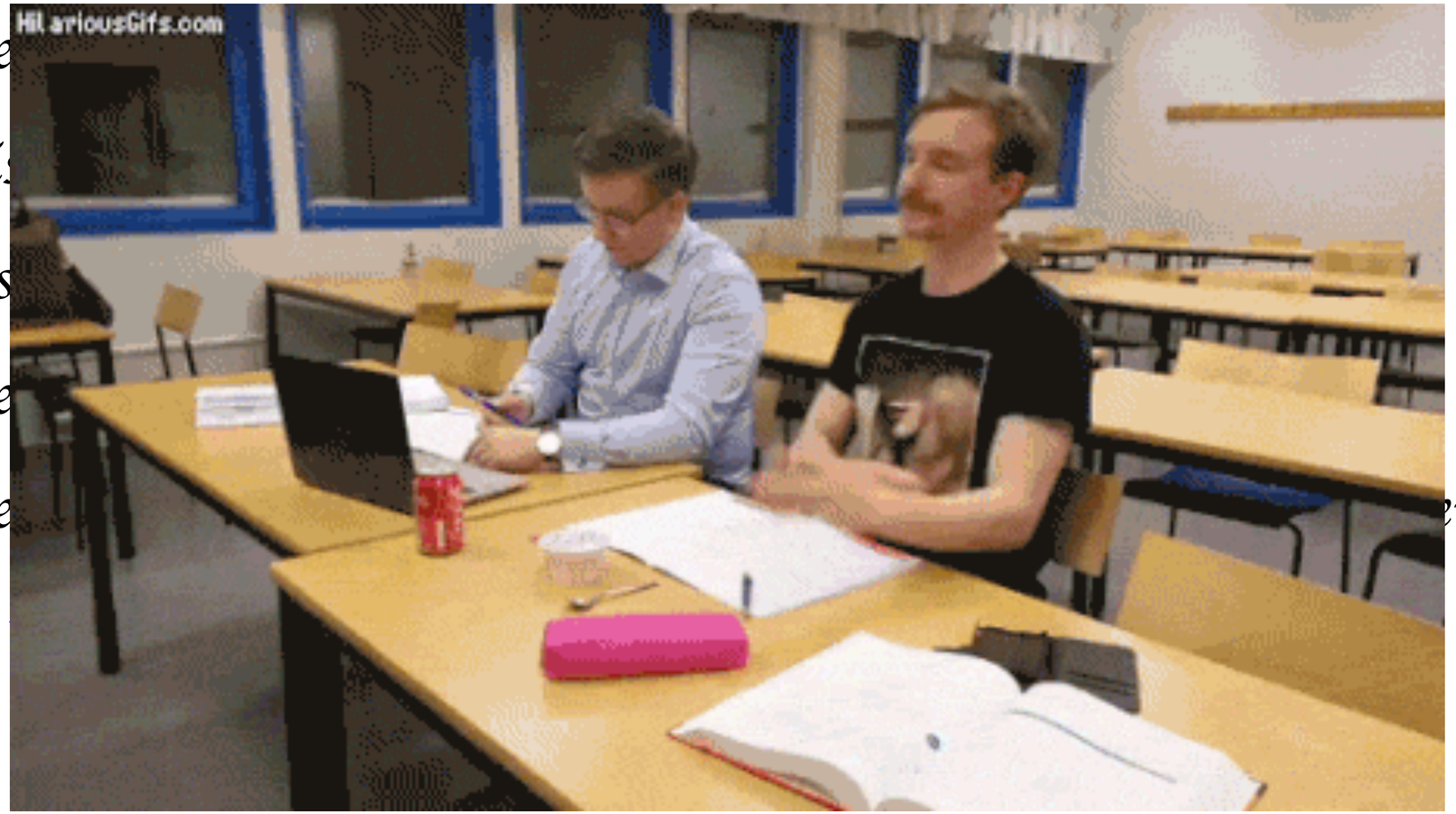
Parallelis

Not the s

Concurre

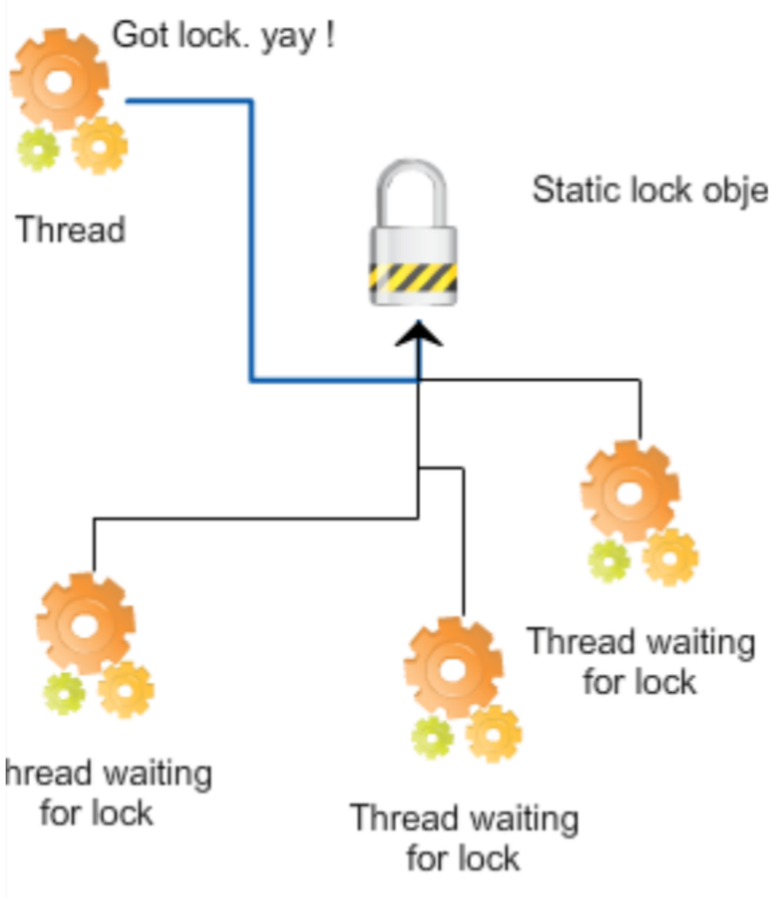
Concurre *em that may*

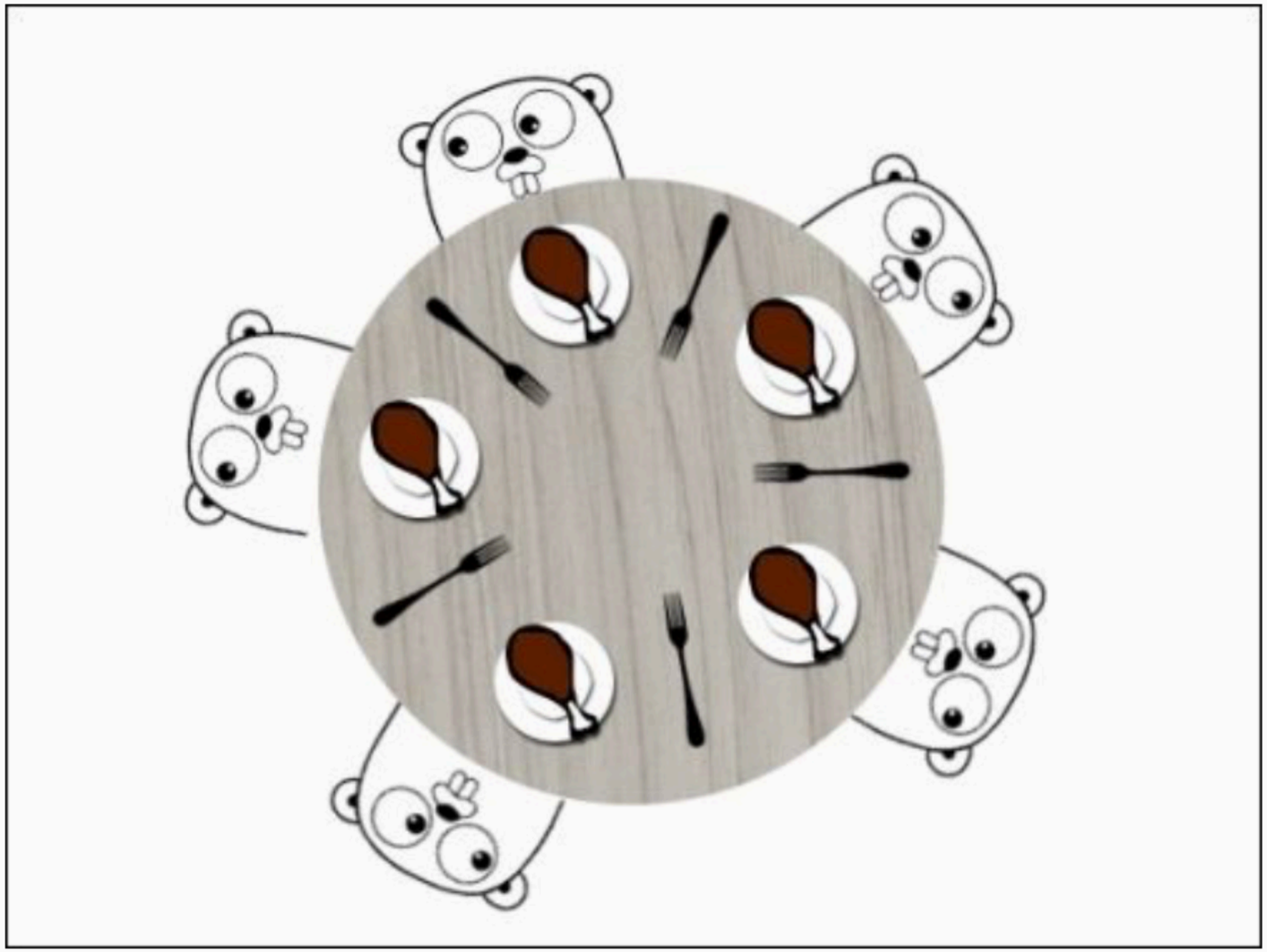
(but not



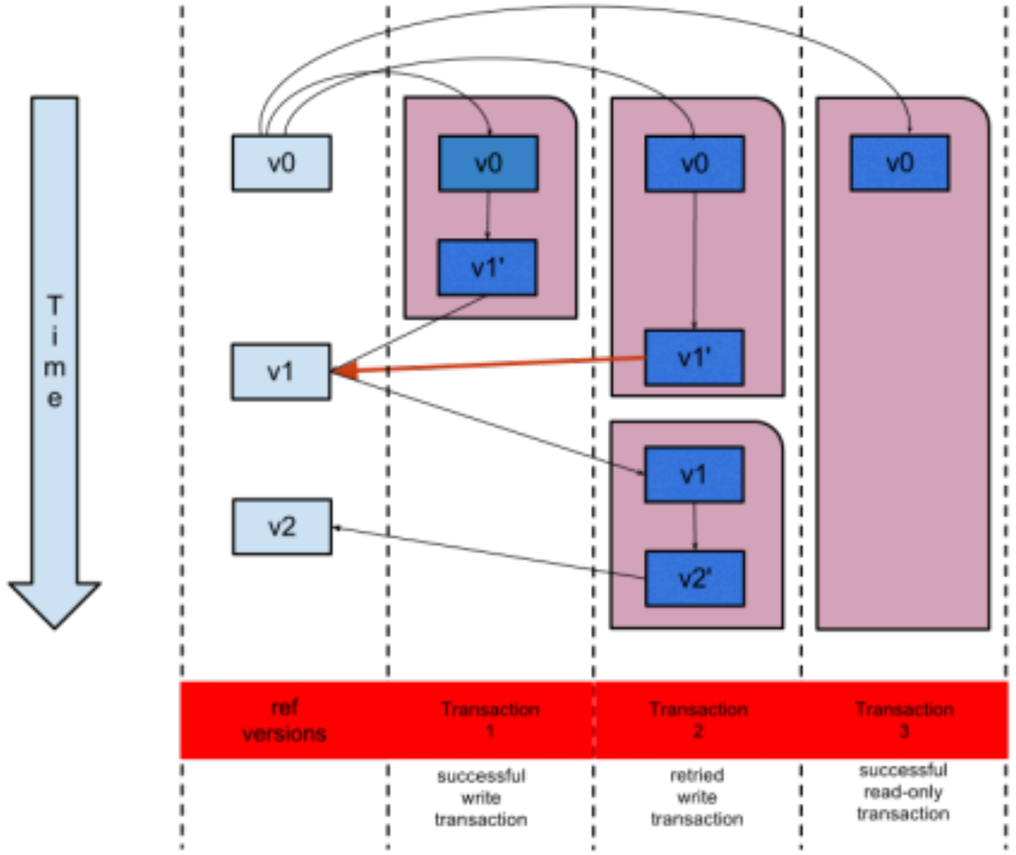
Rob Pike

Lock Condition





Software Transactional Memory



A model for software construction

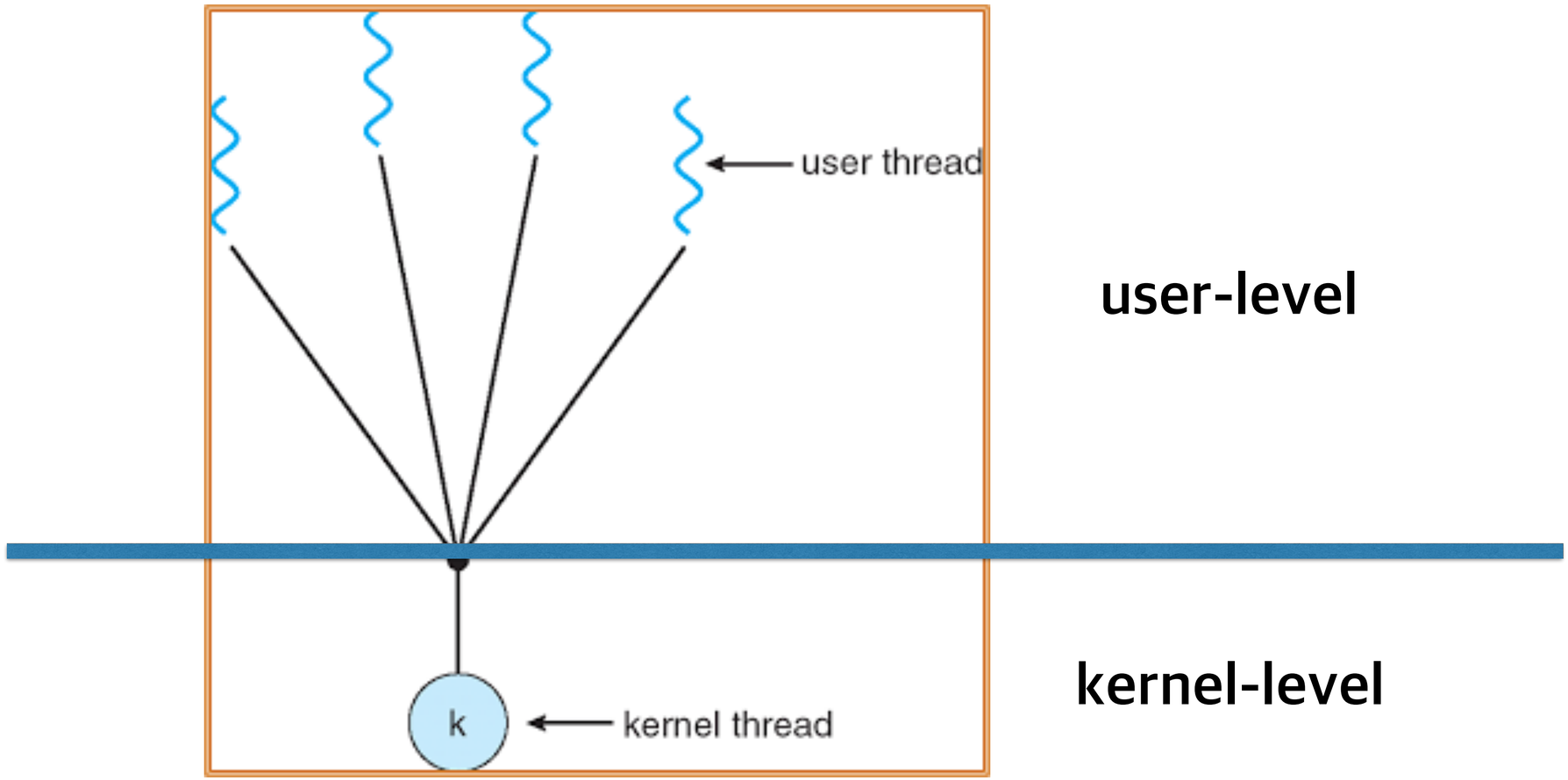
Easy to understand.

Easy to use.

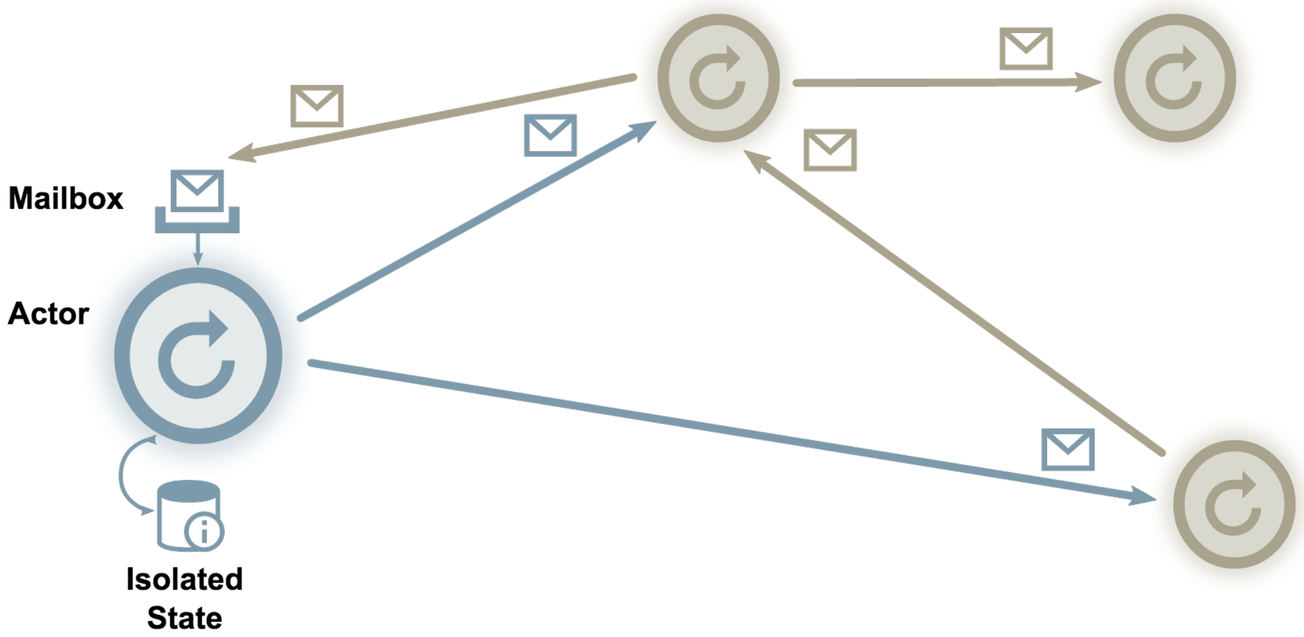
Easy to reason about.

You don't need to be an expert!

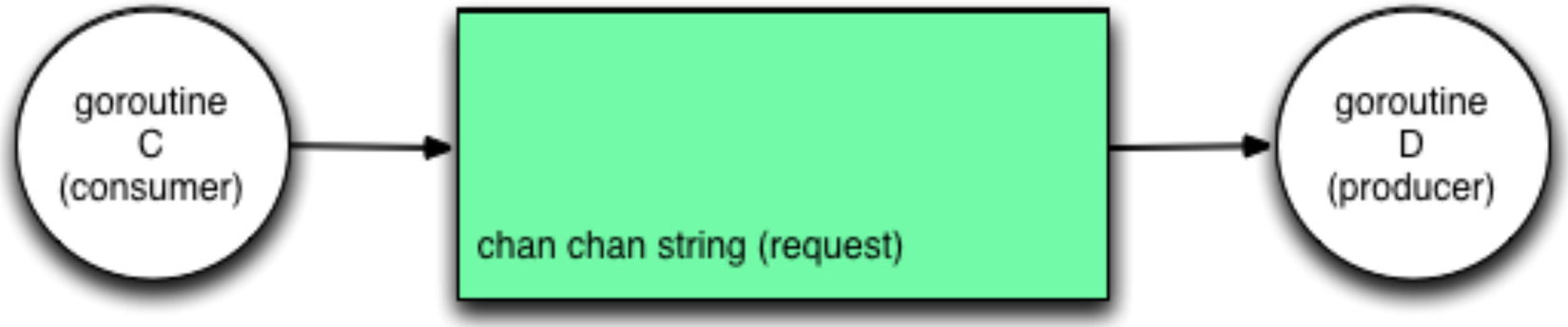
Green Threads



Actor



Communicating Sequential Processes



Actor와 다른점은 Actor 는 pid에게 전달을 하지만 CSP는 anonymous로 channel로 전달함

Go루틴(Goroutines)

- OS 쓰레드보다 경량화된 Go 쓰레드
- 같은 메모리 공간을 공유해서 다른 Goroutines을 병렬로 실행한다.
- 스택에서 작은 메모리로 시작 필요에 따라 힙 메모리에 할당/해제
- 프로그래머는 쓰레드를 처리하지 않고 마찬가지로 OS도 Goroutine의 존재를 모른다.
- OS관점에서는 Goroutine은 C program의 event-driven형식처럼 작동한다.

Goroutines vs OS threads

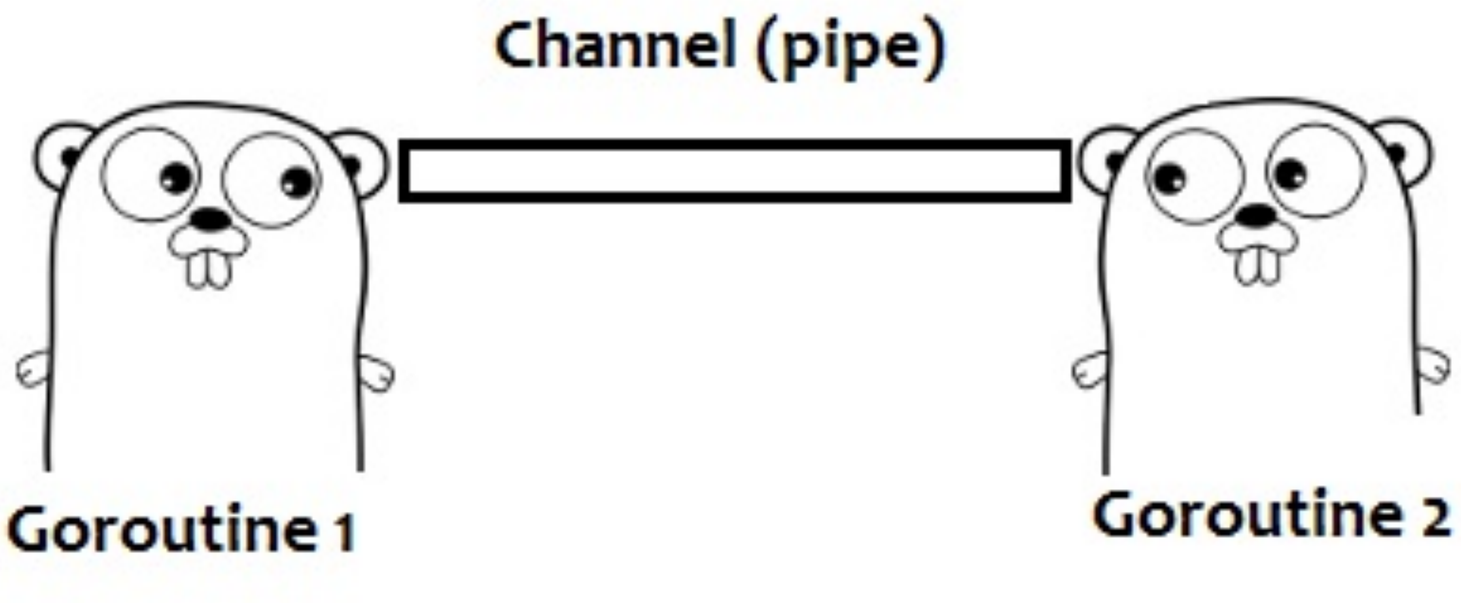
	Goroutines	OS threads
Memory consumption	2kb	1mb
Setup and teardown costs	pretty cheap	high-priced
Switching costs **(saved/restored)**	Program Counter, Stack Pointer and DX.	ALL registers, that is, 16 general purpose registers, PC (Program Counter), SP (Stack Pointer),

Goroutine stack size was decreased from 8kB to 2kB in Go 1.4.

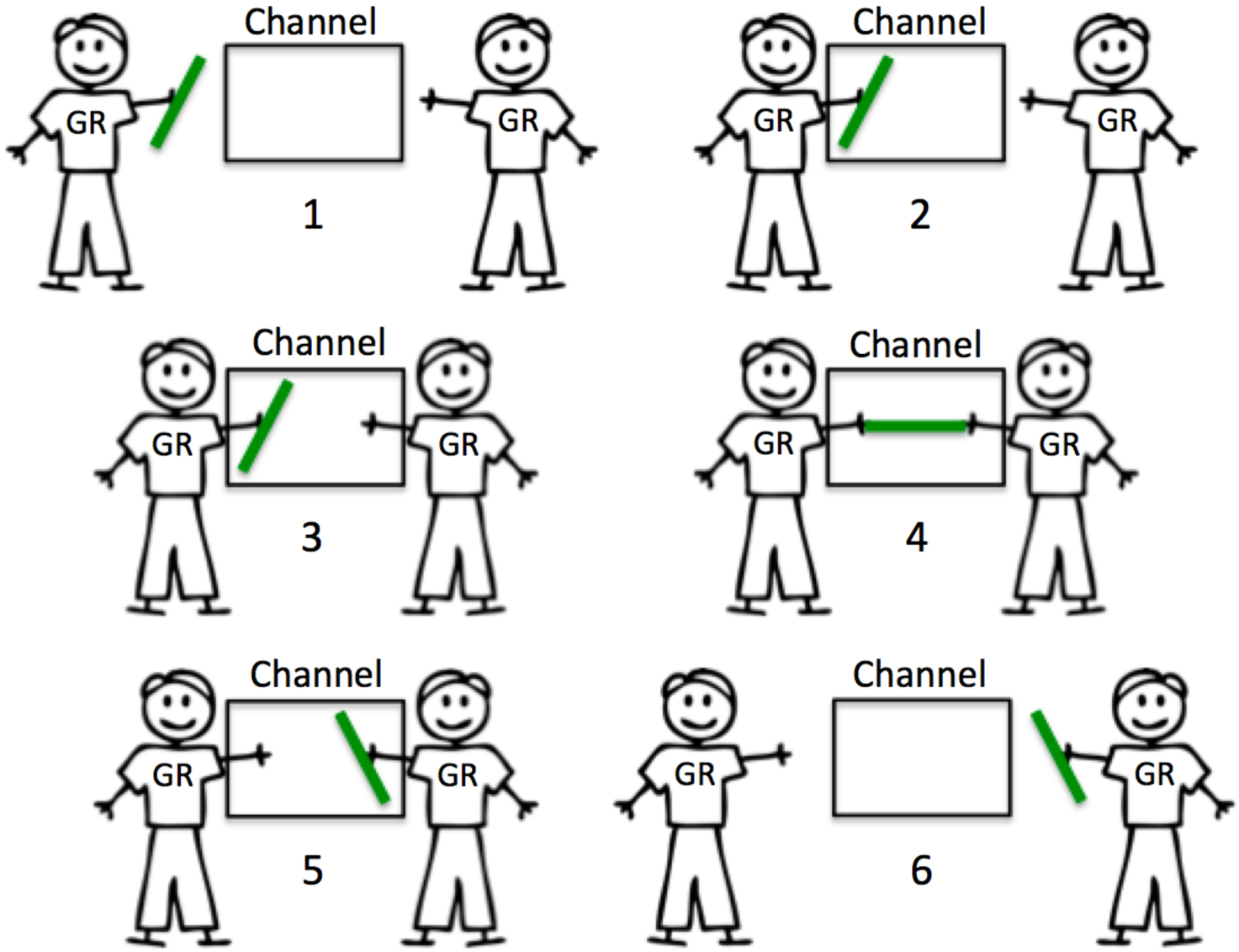
Goroutines blocking

- network input
- sleeping
- channel operations or
- blocking on primitives in the sync package.
- call function (case by case)

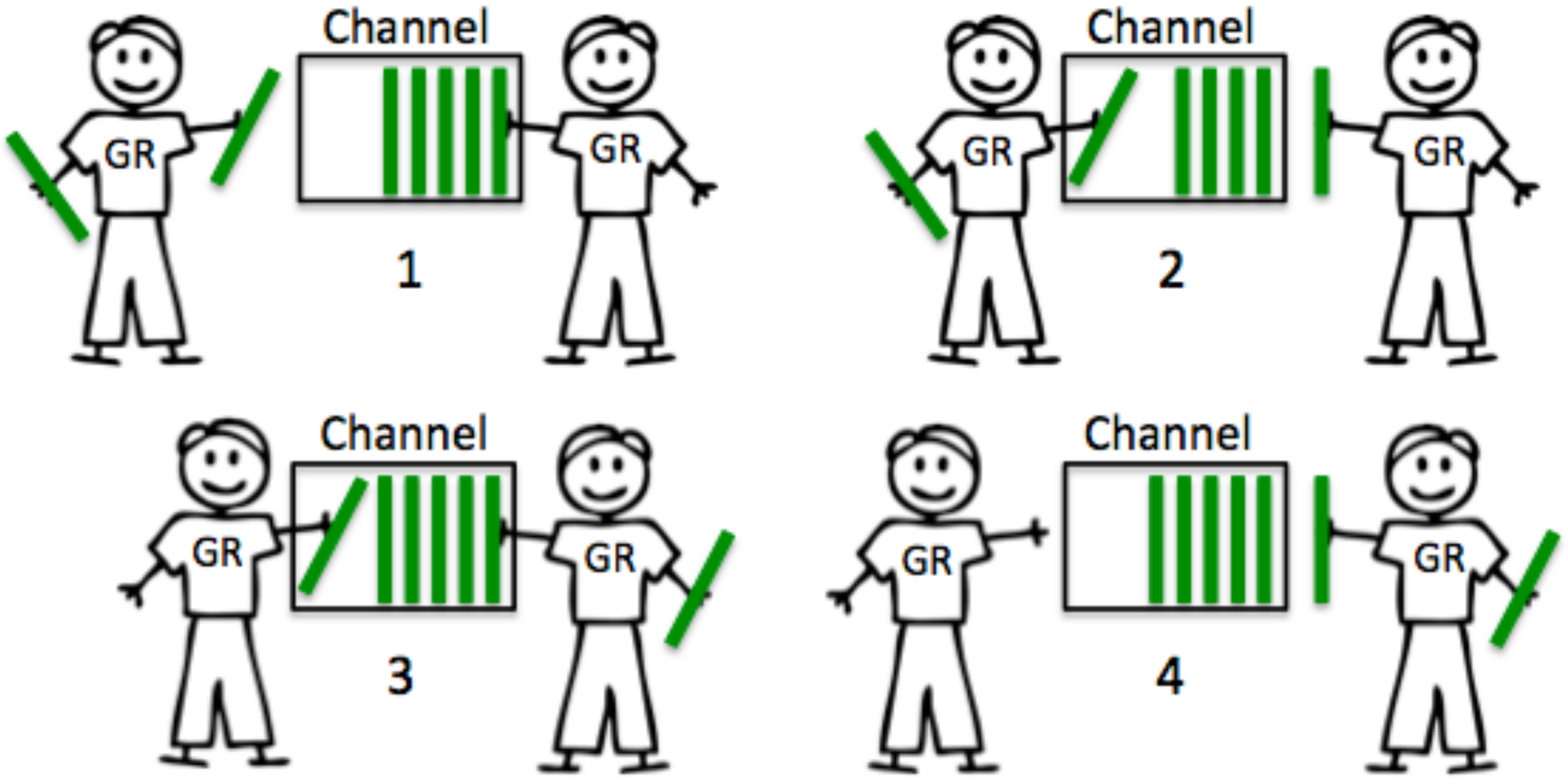
Channel



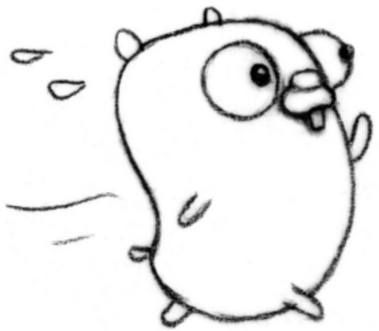
Unbuffered Channel



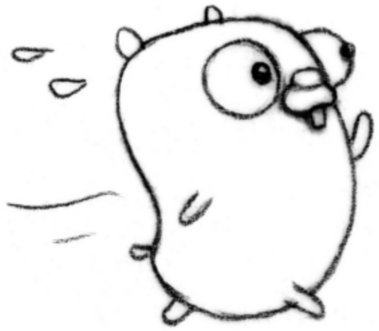
Buffered Channel



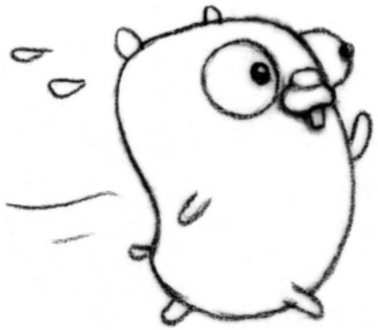
Select



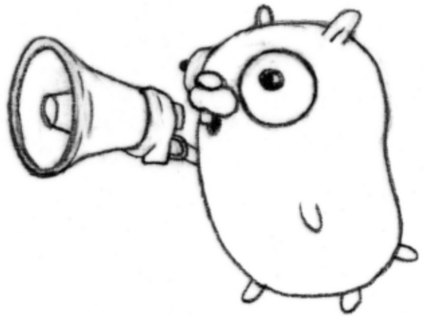
channel



channel



channel



Select

Select

```
func main() {
  tick := time.Tick(100 * time.Millisecond)
  boom := time.After(500 * time.Millisecond)

  for {
    select {
    case <-tick:
      fmt.Println("tick.")
    case <-boom:
      fmt.Println("BOOM!")
      return
    default:
      fmt.Println("    .")
      time.Sleep(50 * time.Millisecond)
    }
  }
}
```

예제

Example

```
package main

func main() {
    // create new channel of type int
    ch := make(chan int)

    // start new anonymous goroutine
    go func() {
        // send 42 to channel
        ch <- 42
    }()
    // read from channel
    <-ch
}
```

Syntax of native CSP in Go

```
ch := make(chan interface{})
ch <- 1
<-ch
close(ch)
```

Communication

```
select {
case <-ch:
case ch <- s:
default:
}
```

Synchronization

```
go func() { … }
```

Process

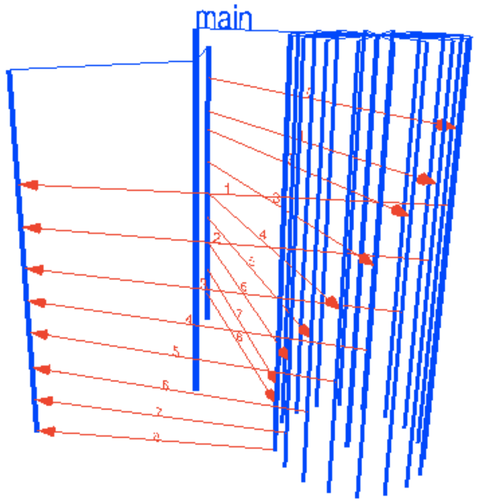

Sit down
We're gonna have fun

```go
func main() {
  runtime.GOMAXPROCS(1)

  go func() {
    for {
    }
  }()

  log.Println("Bye")
}
```

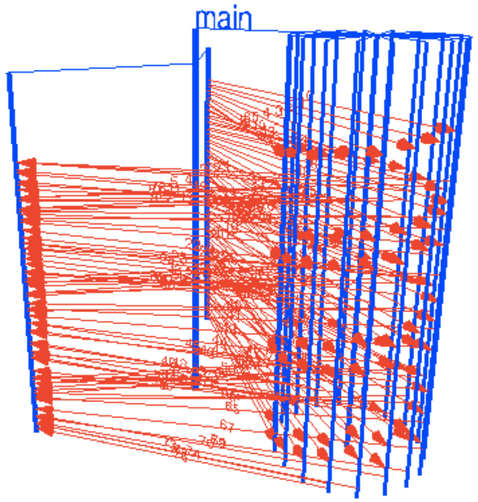
예제

GOMAXPROCS

GOMAXPROCS는 동시에 일을 시킬수 있는
CPU의 최대 수를 설정하는 변수

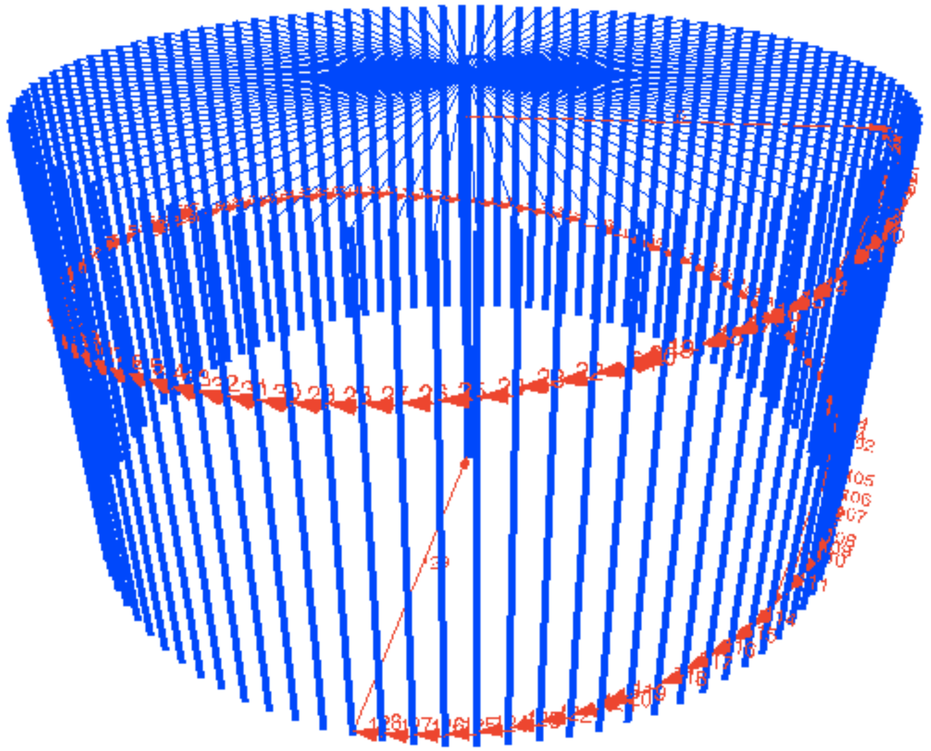


GOMAXPROCS=1

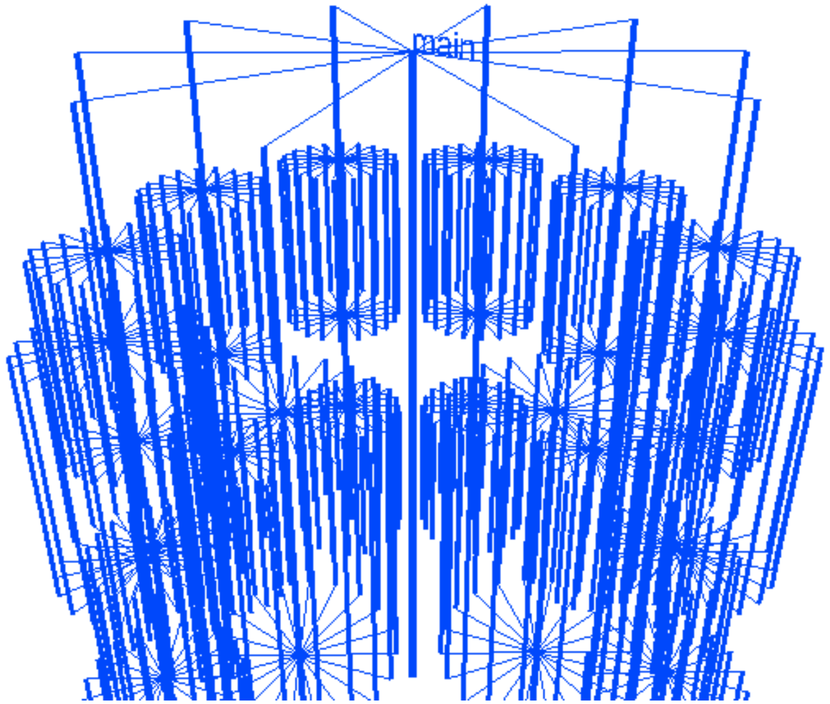


GOMAXPROCS=24

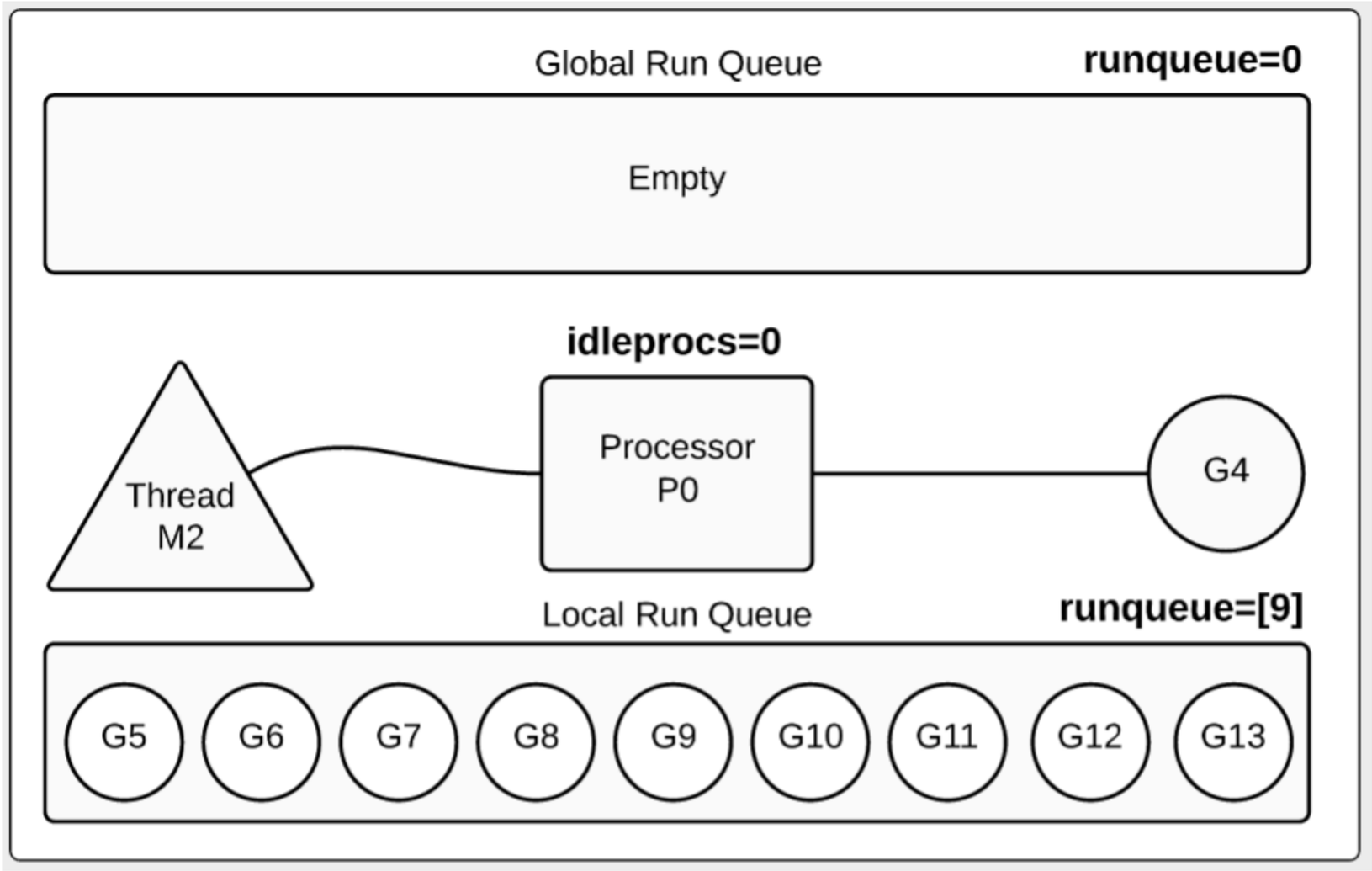
Concurrency



Parallelism

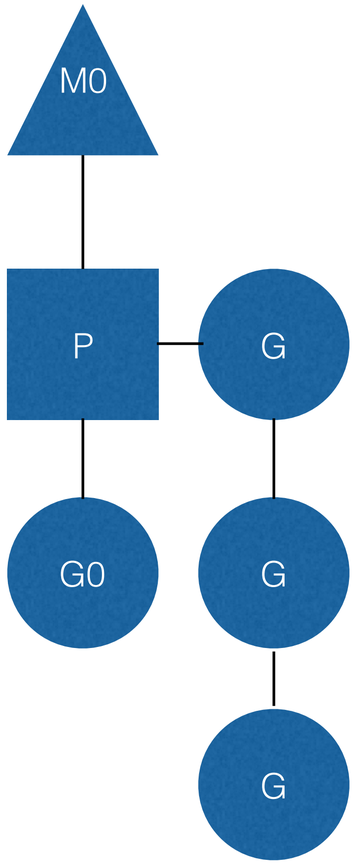


Go Scheduler



https://www.goinggo.net/2015/02/scheduler-tracing-in-go.html

Go Scheduler

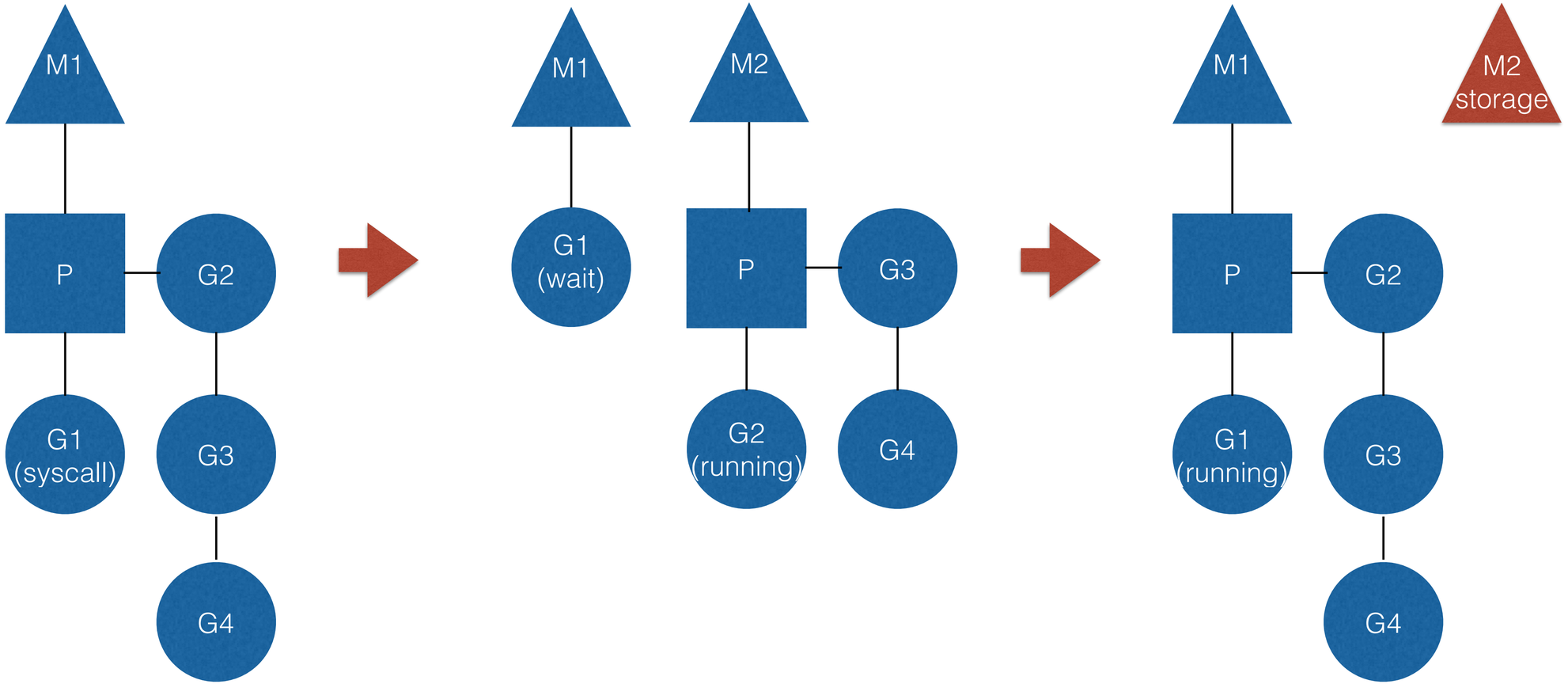


- M: OS threads.
- P: Context to run Go routines.
- G: Go routine.

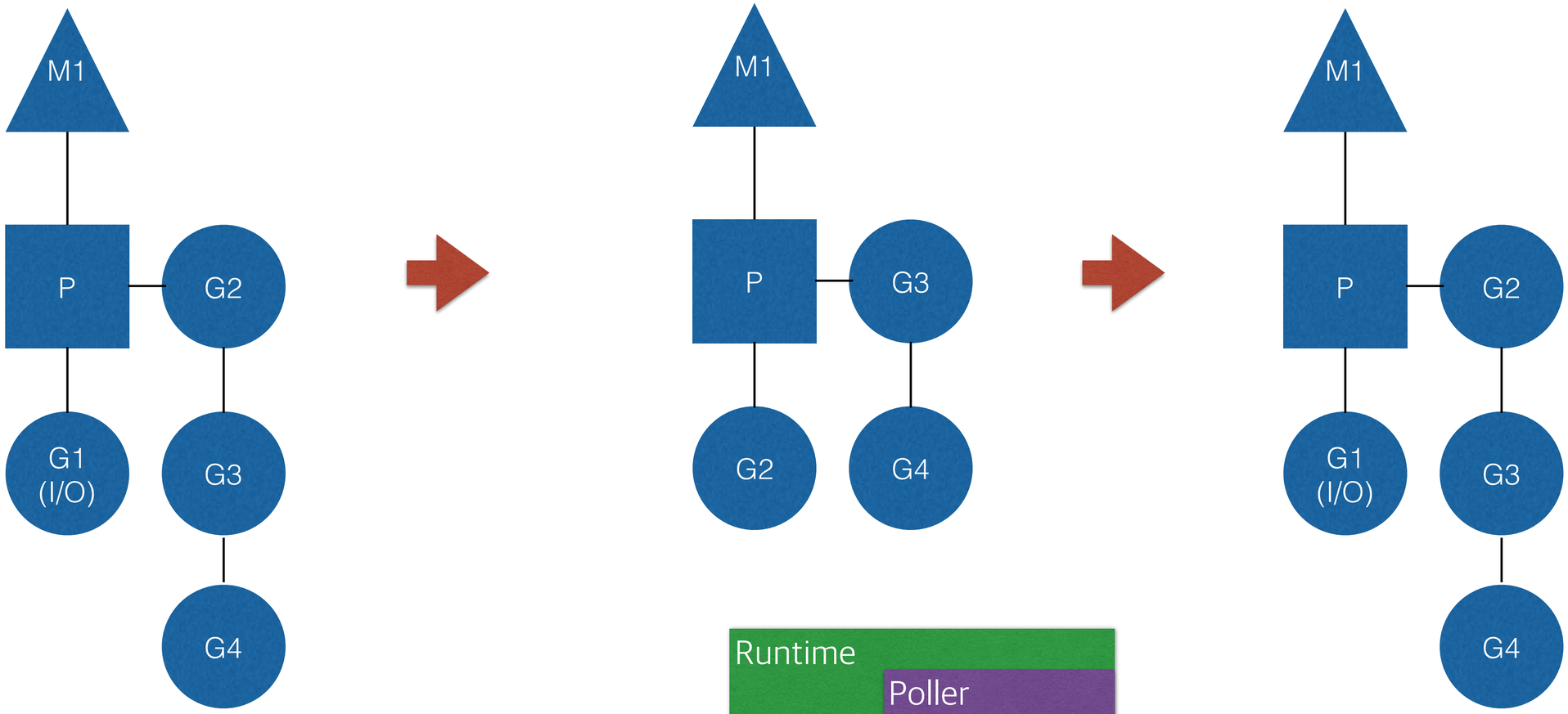
Goroutines blocking

- network input
- sleeping
- channel operations or
- blocking on primitives in the sync package.
- call function (case by case)

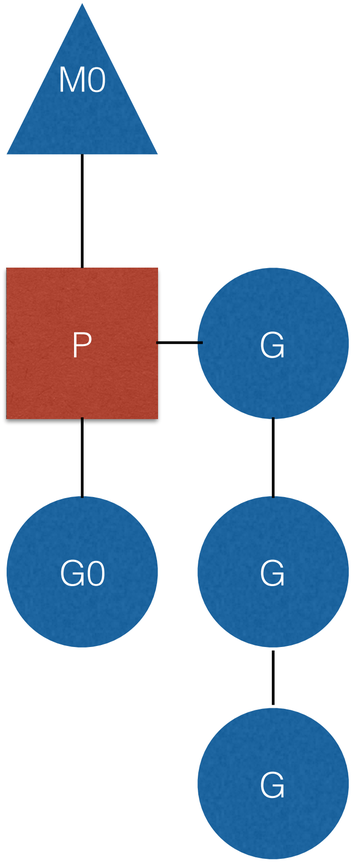
Blocking syscalls

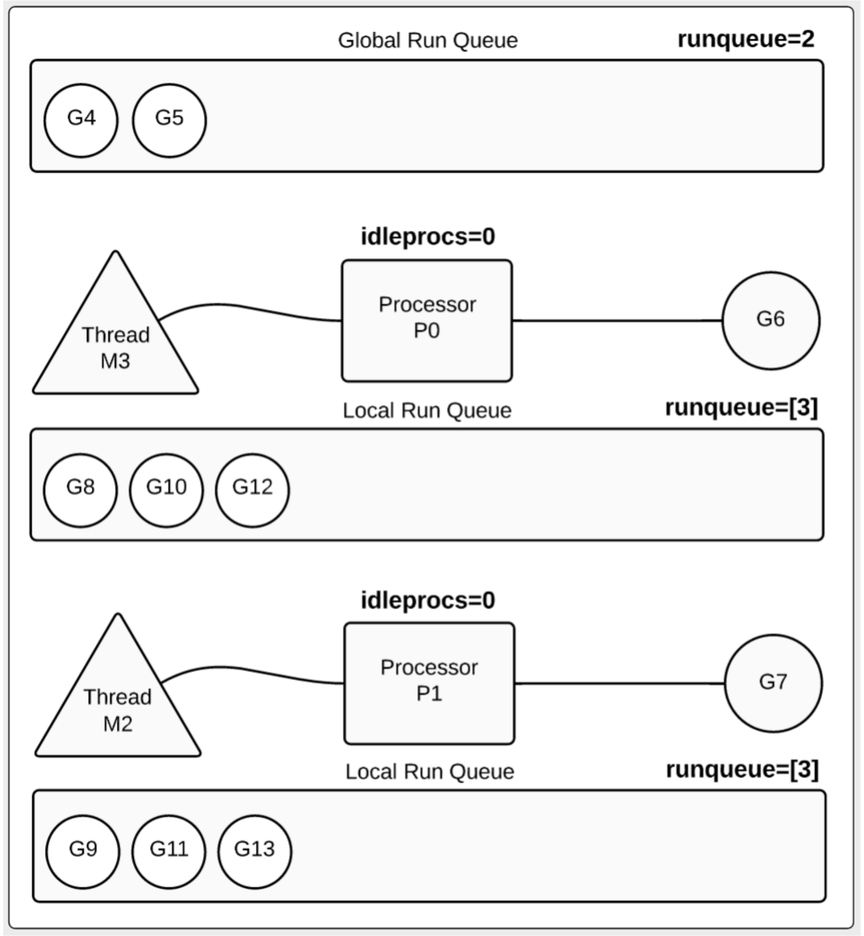


Network I/O call



GOMAXPROCS





GOMAXPROCS = 2

이론

현실

Best practices for concurrency

```
func main() {

  var wg sync.WaitGroup

  for i := 0; i < 10; i++ {
    wg.Add(1)
    go func(n int) {
      log.Println(n)
      wg.Done()
    }(i)
  }

  wg.Wait()
  fmt.Println("the end")
}
```

예제

Best practices for concurrency

```
func main() {

  ch := make(chan int)
  go func(c chan int) {
    for {
      select {
      case <-c:
        log.Println("Hi")
      }
    }
  }(ch)

  go func() {
    for {
      ch <- 1
      time.Sleep(1000 * time.Millisecond)
    }
  }()

  time.Sleep(5000 * time.Millisecond)
}
```

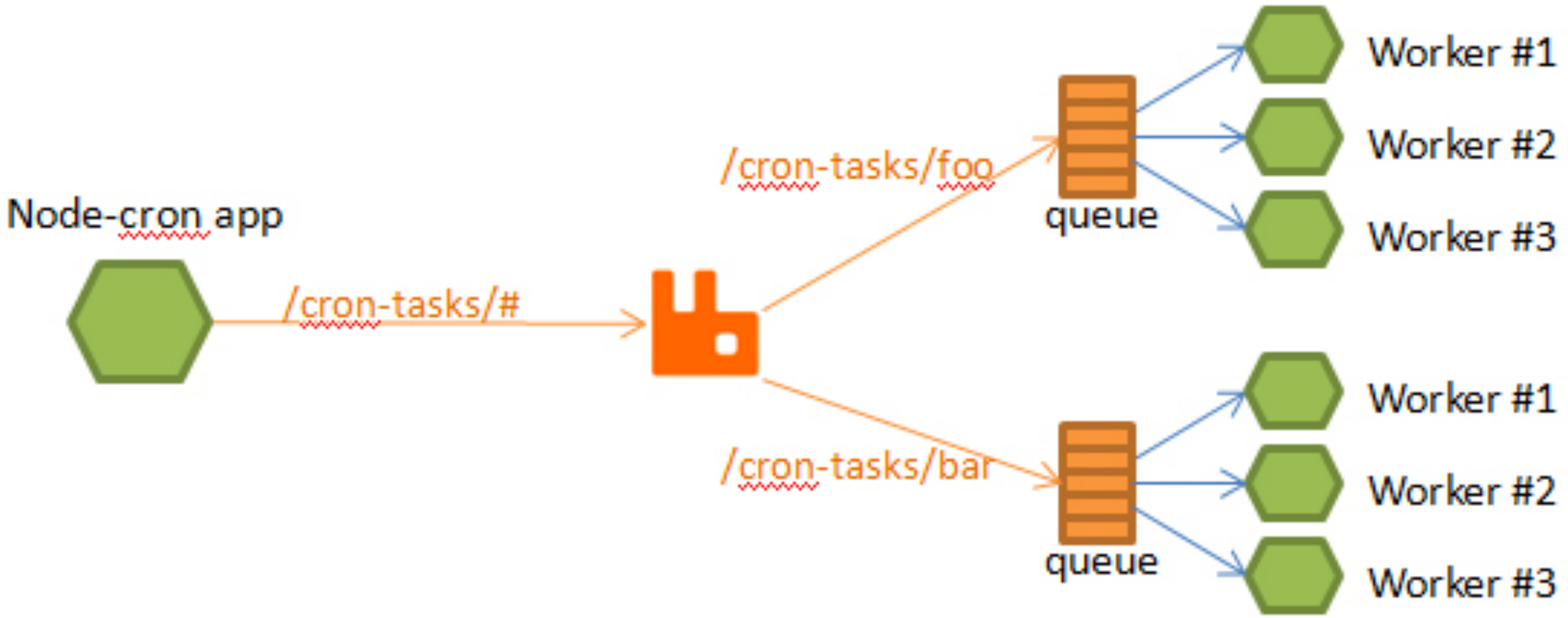
예제

Best practices for concurrency

```
func main() {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()

  for n := range gen(ctx) {
    fmt.Println(n)
    if n == 5 {
      cancel()
      break
    }
  }
}
```

Best practices for concurrency

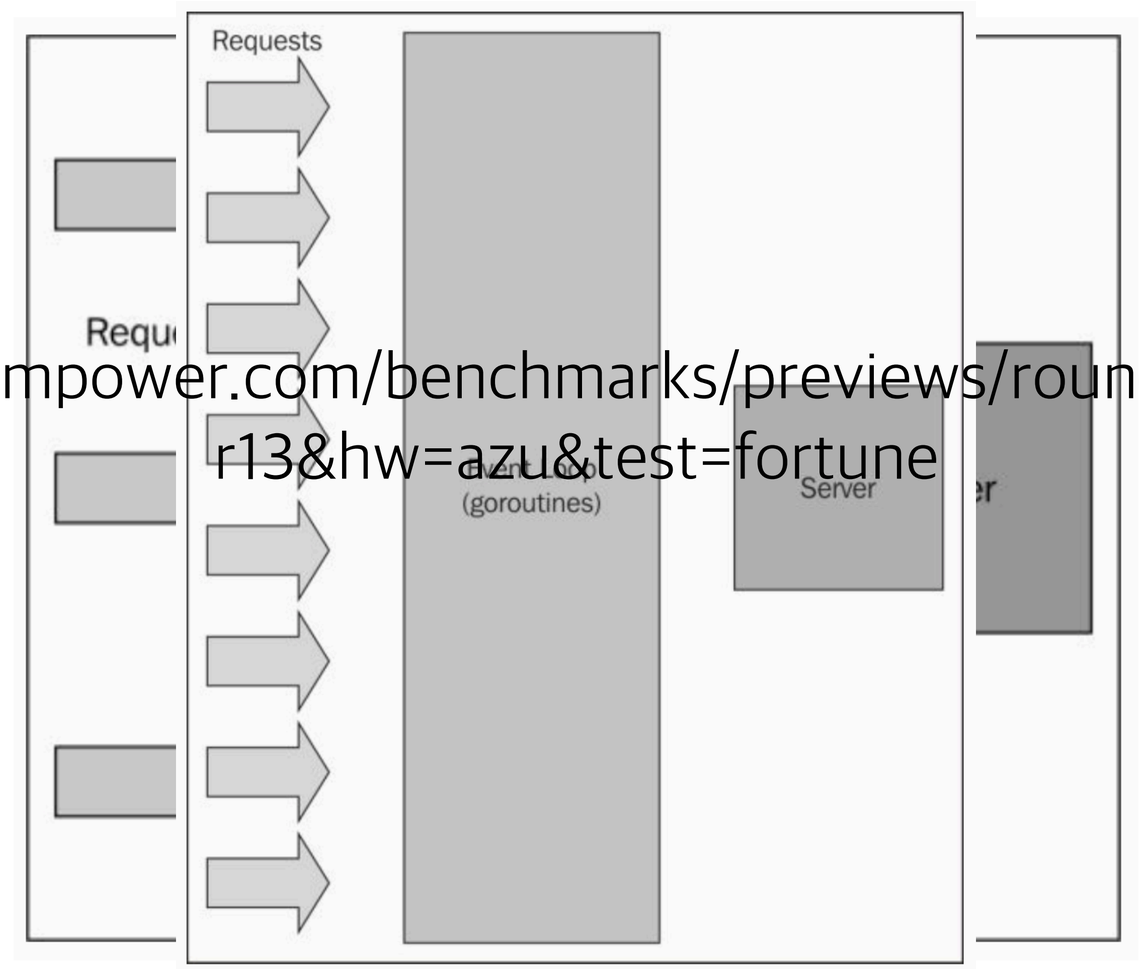
MessageQueue(RabbitMQ) Worker




```
}()

log.Printf(" [*] Waiting for messages. To exit press CTRL+C")
<-forever
```

Best practices for concurrency

HTTP Server



https://www.techempower.com/benchmarks/previews/round13/#section=data-r13&hw=azu&test=fortune

끝

Q&A