Golang design4concurrency

v0.0.1 GoMAD Apr 2015

aleasoluciones

Alea Soluciones

@eferro

@jaimegil

Agenda

Example 1
Example 2
Code samples
References
Conclusion

First, an example

As a customer I put sufficient coins into the vending machine and then press the selection button for coke and then press the button to vend and the robotic arm fetches a coke and dumps it into the pickup tray. The coke is nice and cold because the cooling system keeps the air in the vending machine at 50 degrees.

Nouns (potential clases):
Customer, coins, vending machine, selection button, coke, button to vend, robotic arm, pickup tray, cooling system...

Verbs (methods):
Selection button: push
Vend button: push
Robotic arm: pickup(coke)
cooling system: cool forever
...

OO Solution:
Class, methods

How to implement concurrent behaviours???

OO Solution:
Class, methods

How to implement concurrent behaviours???

Threads, threads pools, locks, debug, debug, debug...


I'LL USE THREADS,
POOLS, MUTEXES...
SHUT UP, DESIGN
FOR CONCURRENCY
memegenerator.es

One goroutine for each truly concurrent activity in the system

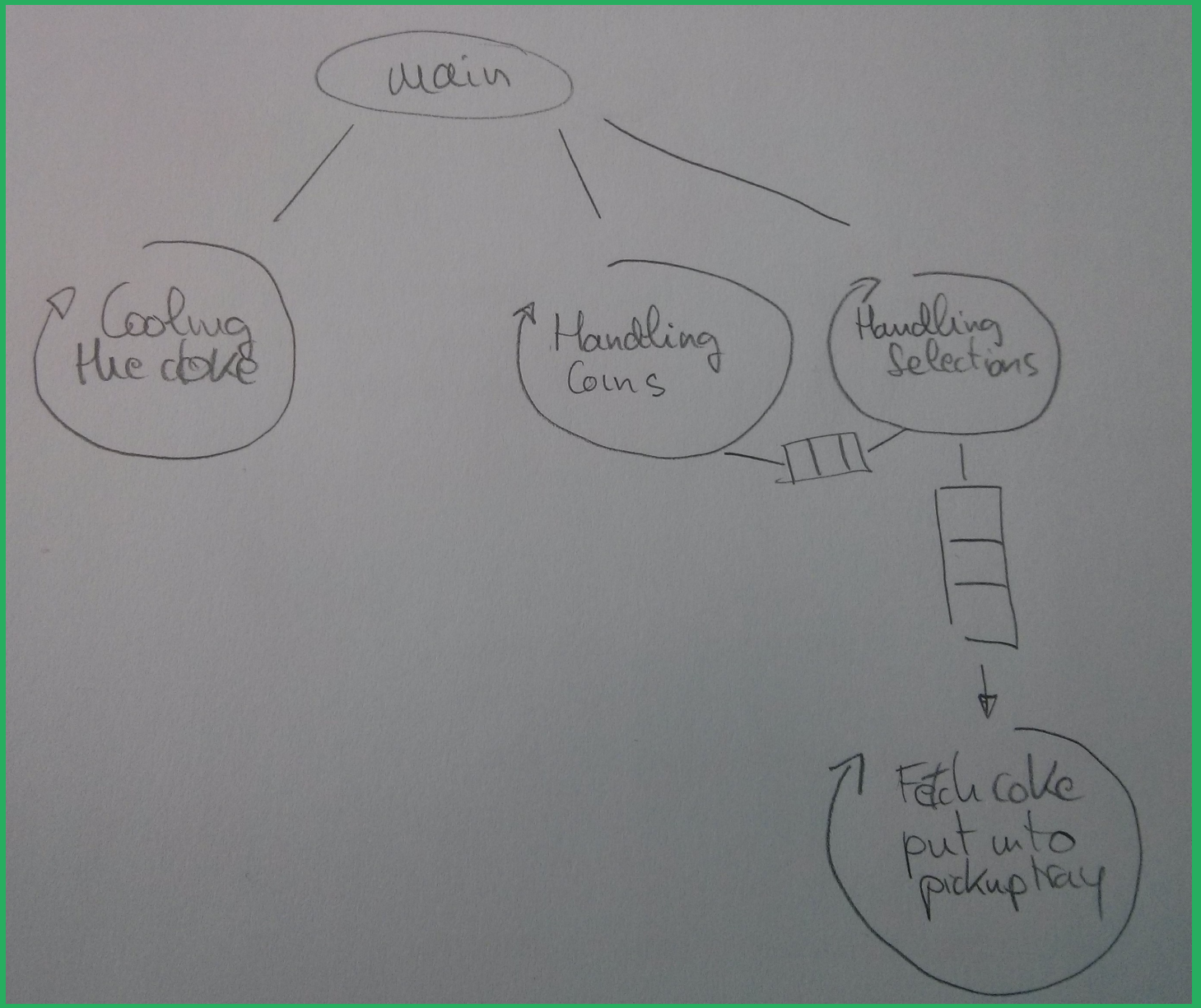
Goroutines are not SO processes

Goroutines are not threads

Goroutines are really cheap

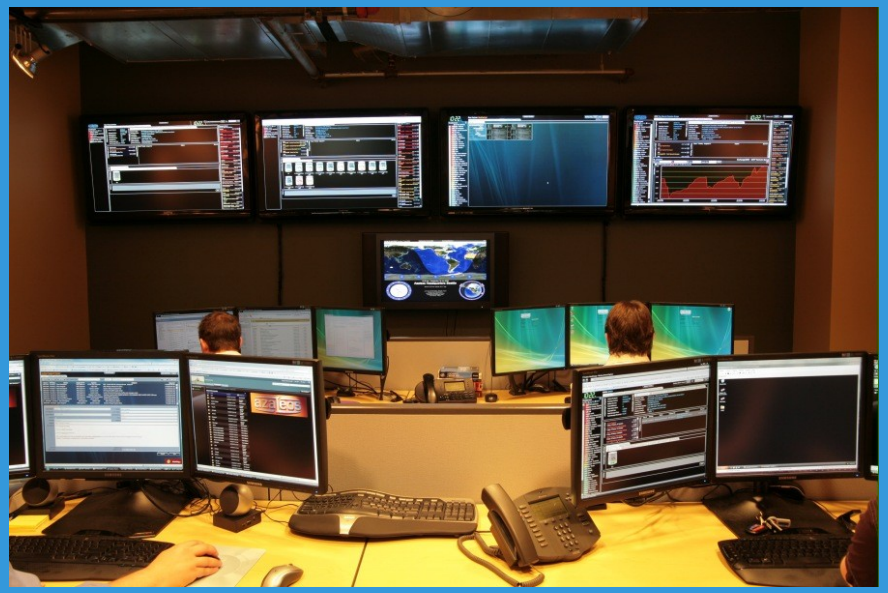
One goroutine for each truly concurrent activity in the system

Handling coins, Putting coins into the slots, Handling selections, Cooling the soda, Fetching the coke and putting into the pickup tray....

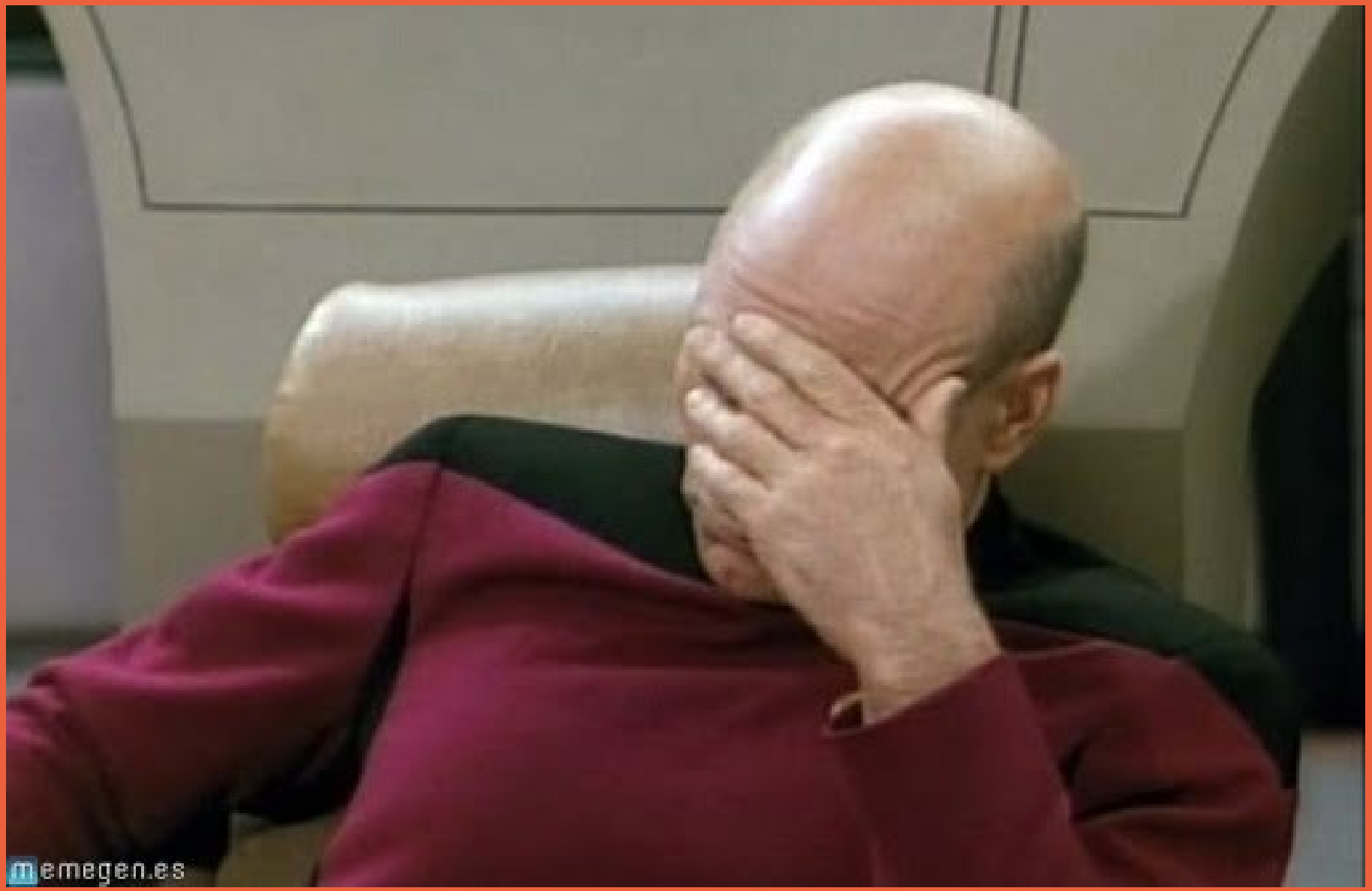

main
Cooling the coke
Handling Coins
Handling Selections
Fetch coke put into pickup tray

Another example

As a network operator I can check in real time the state of each home router of our Fiber to the home network (200,000 routers). The home routers send events with each state change to the central system.



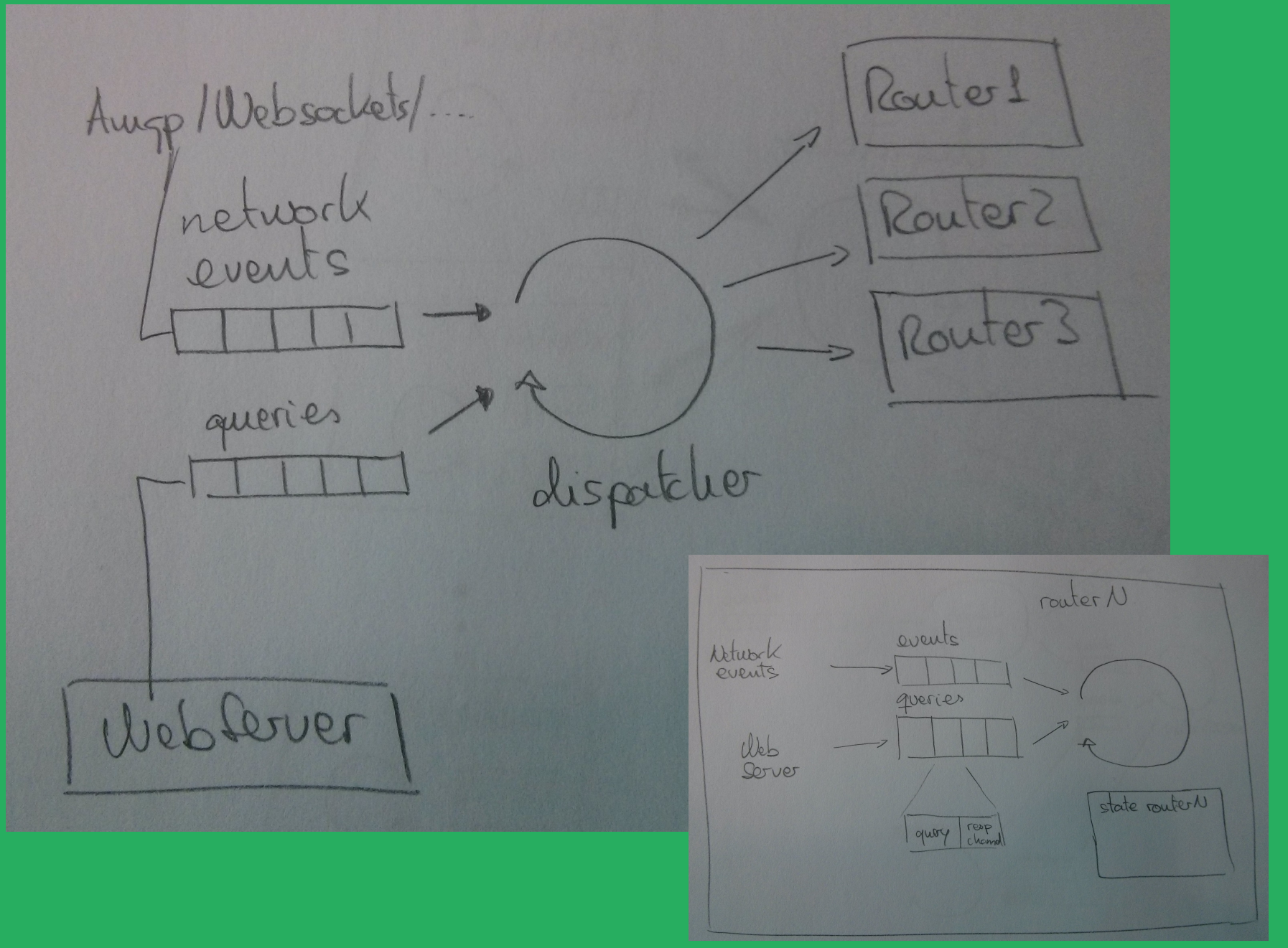
EventProcessor, DBEventSerializer, RouterStateCalculator, EventProcessorsPool...


memegen.es

One goroutine for each truly concurrent activity in the system

Receive events from home routers, Dispatch events, compute state for a home router...

200,000 routers = min 200,000 goroutines


Amqp / Websockets / ...
network events
queries
dispatcher
Router 1
Router 2
Router 3
Webserver
router N
Network events
events
queries
Web Server
query
reap channel
state routerN

One goroutine for each truly concurrent activity in the system

Goroutines are really cheap

Goroutines are really cheap

Goroutines are really cheap

Goroutines

Small mem footprints
Fast creation
Fasts scheduling


SAY GOROUTINE AGAIN!!!
memecrunch.com

Communicate using channels

Never share memory
Only send messages using channels

Code Samples

https://github.com/aleasoluciones/golang_design4concurrency_talk/

```go
func square(input chan int, output chan int) {
        for value := range input {
                time.Sleep(1 * time.Second)
                output <- value * value
        }
}

func printResult(output chan int) {
        for result := range output {
                fmt.Println("Result", result)
        }
}

func main() {
        input := make(chan int, 100)
        output := make(chan int, 100)

        for i := 0; i < 10; i++ {
                go square(input, output)
        }

        for i := 0; i < 100; i++ {
                input <- i
        }
        close(input)

        go printResult(output)

        time.Sleep(15 * time.Second)
        close(output)

}
```

```
func square(value int, output chan int) {
        time.Sleep(1 * time.Second)
        output <- value * value
}

func printResult(output chan int) {
        for result := range output {
                fmt.Println("Result", result)
        }
}

func main() {
        output := make(chan int, 100)

        for i := 0; i < 100; i++ {
                go square(i, output)
        }

        go printResult(output)

        time.Sleep(15 * time.Second)
}
```

```
func (r *Router) processEvents() chan RouterEvent {
        input := make(chan RouterEvent)
        go func() {
                for event := range input {
                        fmt.Println("Router ", r.Id, "processing", event)
                        // change status or calculate something...
                }
        }()
        return input
}

func dispatch() chan RouterEvent {
        input := make(chan RouterEvent)
        go func() {
                dispatchingMap := make(map[string]chan RouterEvent)
                for event := range input {
                        _, exists := dispatchingMap[event.RouterId]
                        if !exists {
                                router := Router{event.RouterId, ""}
                                routerInputChannel := router.processEvents()
                                dispatchingMap[event.RouterId] = routerInputChannel
                        }
                        dispatchingMap[event.RouterId] <- event
                }
        }()
        return input
}
```

```
func (r *InMemoryRepository) Get(id int) (string, bool) {
	getResponses := make(chan getResponse)

	r.gets <- getRequest{
		Id:       id,
		Response: getResponses,
	}

	response := <-getResponses
	return response.Value, response.Found
}

func (r *InMemoryRepository) dispatch() {
	storage := map[int]string{}

	for {
		select {
		case put := <-r.puts:
			storage[put.Id] = put.Value
		case get := <-r.gets:
			value, found := storage[get.Id]
			get.Response <- getResponse{
				Value: value,
				Found: found,
			}
		}
	}
}

func NewInMemoryRepository() *InMemoryRepository {
	repo := InMemoryRepository{
		puts: make(chan putRequest),
		gets: make(chan getRequest),
	}
	go repo.dispatch()
	return &repo
}
```

```
func NewSafeMap() SafeMap {
        sm := make(SafeMap)
        go sm.run()
        return sm
}

func (sm SafeMap) run() {
        st := make(store)
        for command := range sm {
                command(st)
        }
}

func (sm SafeMap) Insert(key Key, value Value) {
        sm <- func(st store) {
                st[key] = value
        }
}

func (sm SafeMap) Delete(key Key) {
        sm <- func(st store) {
                delete(st, key)
        }
}
```

Conclusions

Design for concurrency
Don't share memory
Lot of goroutines / they are cheap

References

Go concurrency patterns
https://talks.golang.org/2012/concurrency.slide

Advanced Go concurrency patterns
https://blog.golang.org/advanced-go-concurrency-patterns

@martinjlogan Designing for Actor Based Systems
http://blog.erlware.org/designing-for-actor-based-systems/

Golang patterns for serving on-demand, generated content
http://blog.gitorious.org/2014/08/11/golang-patterns-for-serving-on-demand-generated-content/

Questions???

@eferro

@jaimegil

aleasoluciones

Thanks !!!

@eferro

@jaimegil

aleasoluciones