

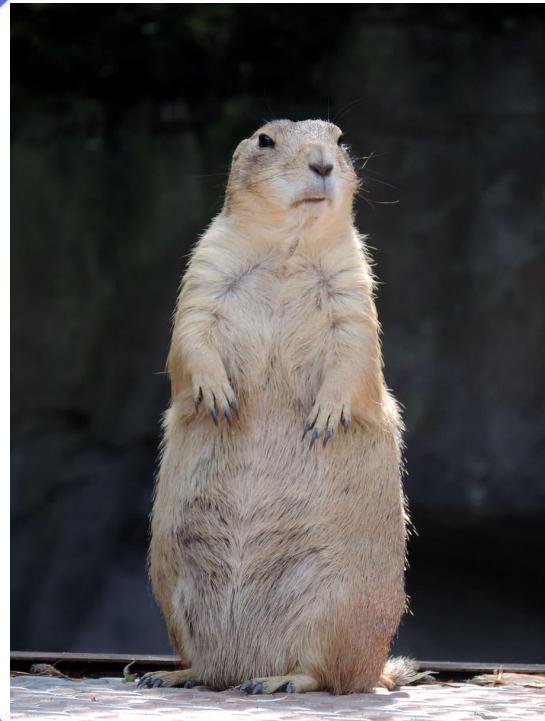
# Scaling Coffee With Goroutines

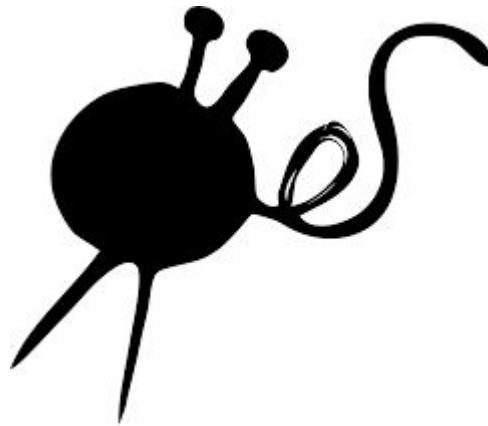
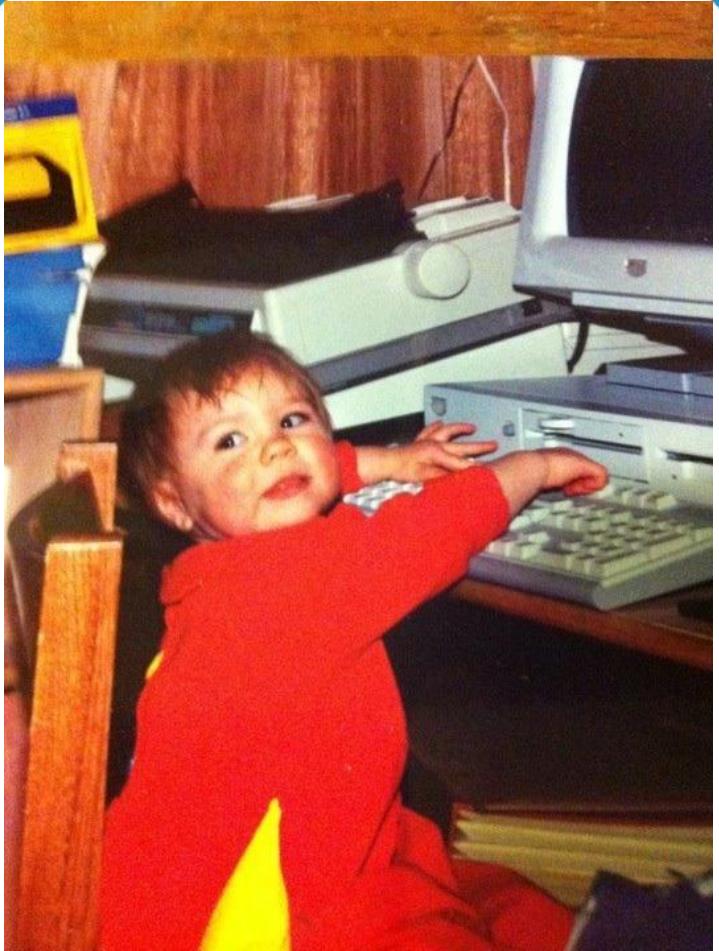


GoLab 2023 - Florence

Sadie Freeman  
Senior Backend Engineer

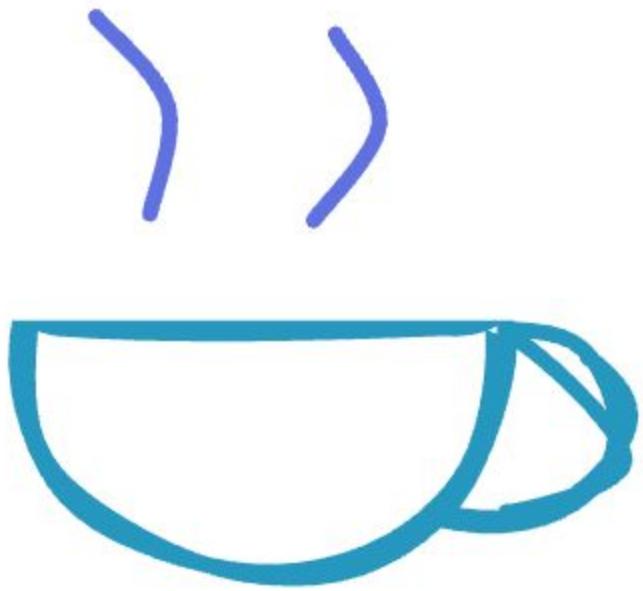
# All Gophers welcome!





// TODO:

- Step by Step
- Scaling for speed
- Scaling for load
- Case study
- Link to tutorial



# How can we serve

1. A lot of coffee
2. To a lot of people
3. As fast as we can

Speed + Load

# Coffee Actions

Take payment

Steam milk

Make espresso

1

# No Concurrency

```
func main() {
    start := time.Now()
    PayForCoffee()
    MakeEspresso()
    SteamMilk()
    log.Printf("Coffee made, 1 customer served")

    timeTaken := time.Since(start)
    log.Printf("Took %s to serve coffee", timeTaken)
}
```

1

# No Concurrency

```
func PayForCoffee() {  
    time.Sleep(2 * time.Second)  
    log.Printf("Coffee paid for 💰")  
}  
  
func MakeEspresso() {  
    time.Sleep(2 * time.Second)  
    log.Printf("Espresso made ☕")  
}  
  
func SteamMilk() {  
    time.Sleep(2 * time.Second)  
    log.Printf("Milk steamed 🥛")  
}
```



## No Concurrency

```
`go run main.go`
```

Coffee paid for 💰

Espresso made ☕

Milk steamed 🥛

Coffee made, 1 customer served

Took 6.02238615s to serve coffee

2

## Serve more customers

```
func ServeCustomer(w http.ResponseWriter, r *http.Request) {
    start := time.Now()

    numCustomers, err := strconv.Atoi(strings.TrimPrefix(r.URL.Path, "/serve-customer/"))
    if err != nil || numCustomers == 0 {
        numCustomers = 1
    }

    count := 0
    for i := 0; i < numCustomers; i++ {
        MakeCoffee()
        count++
    }

    timeTaken := time.Since(start)
    log.Printf("Took %s to serve coffee to %v customer(s)", timeTaken, count)
}

func MakeCoffee() {
    PayForCoffee()
    MakeEspresso()
    SteamMilk()
}
```

2

## Serve more customers

```
`go run main.go`
```

```
`curl http://localhost:8080/serve-customer/3`
```

Took 18.009306471s to serve coffee to 3 customer(s)

Speed

Step 1:

Don't

3

## Use a Goroutine

```
go MakeCoffee()
```

Threads:

Set of instructions that can be run independently

Concurrently: Happening at the same time

3

## Use a Goroutine

```
`go run main.go`
```

```
`curl http://localhost:8080/serve-customer/3`
```

```
Took 12.548µs to serve coffee to 3 customer(s)
```

```
Coffee paid for ☕
```

```
Coffee paid for ☕
```

```
Coffee paid for ☕
```

```
Espresso made ☕
```

```
Espresso made ☕
```

```
Espresso made ☕
```

```
Milk steamed 🍼
```

```
Milk steamed 🍼
```

```
Milk steamed 🍼
```

4

## Add Wait Group

```
wg := sync.WaitGroup{}  
count := 0  
for i := 0; i < numCustomers; i++ {  
    wg.Add(1)  
    go MakeCoffee(&wg)  
    count++  
}  
wg.Wait()  
  
func MakeCoffee(wg *sync.WaitGroup) {  
    defer wg.Done()  
    PayForCoffee()  
    MakeEspresso()  
    SteamMilk()  
}
```

4

## Add Wait Group

```
`go run main.go`
```

```
`curl http://localhost:8080/serve-customer/3`
```

```
Coffee paid for $
```

```
Coffee paid for $
```

```
Coffee paid for $
```

```
Espresso made ☕
```

```
Espresso made ☕
```

```
Espresso made ☕
```

```
Milk steamed 🥛
```

```
Milk steamed 🥛
```

```
Milk steamed 🥛
```

```
Took 6.008748085s to serve coffee to 3 customer(s)
```

5

## Add MORE Goroutines

```
func MakeCoffee(wg *sync.WaitGroup) {  
    defer wg.Done()  
  
    newWg := sync.WaitGroup{}  
    newWg.Add(3)  
    go PayForCoffee(&newWg)  
    go MakeEspresso(&newWg)  
    go SteamMilk(&newWg)  
    newWg.Wait()  
}
```

5

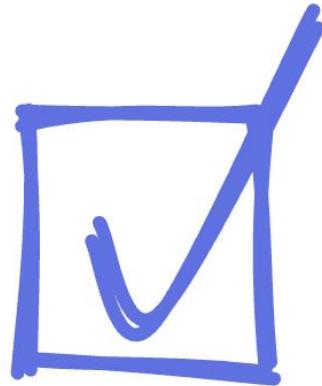
## Add MORE Goroutines

```
`go run main.go`
```

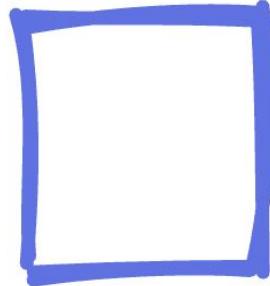
```
`curl http://localhost:8080/serve-customer/3`
```

```
Espresso made ☕  
Coffee paid for 💰  
Coffee paid for 💰  
Milk steamed 🥛  
Espresso made ☕  
Milk steamed 🥛  
Milk steamed 🥛  
Espresso made ☕  
Coffee paid for 💰  
Took 2.002738958s to serve coffee to 3 customer(s)
```

\$ € \$ € \$  
\$ MORE CUSTOMERS!!! \$  
\$ € € \$ €



Speed



Load

6

# Containerize

## Dockerfile

```
FROM golang:1.18.4-alpine

WORKDIR /app

COPY go.mod ./
RUN go mod download

COPY *.go ./

RUN go build -o /coffee-shop

EXPOSE 8080

CMD [ "/coffee-shop" ]
```

6

# Containerize

## Makefile

**build:**

```
docker build --tag coffee-shop .
```

**run:** build

```
docker run -d --name coffee-shop -p 8080:8080 coffee-shop
```

**stop:**

```
docker stop coffee-shop  
docker container rm coffee-shop
```

6

# Containerize

*'make build'*

*'make run'*



## 6

# Containerize

```
< coffee-shop coffee-shop  
RUNNING  
  
2022/08/09 18:37:38 Coffee paid for $  
2022/08/09 18:37:38 Milk steamed   
2022/08/09 18:37:38 Milk steamed   
2022/08/09 18:37:38 Coffee paid for $  
2022/08/09 18:37:38 Coffee paid for $  
2022/08/09 18:37:38 Milk steamed   
2022/08/09 18:37:38 Espresso made   
2022/08/09 18:37:38 Espresso made   
2022/08/09 18:37:38 Took 2.001408751s to serve coffee to 3 customer(s)
```

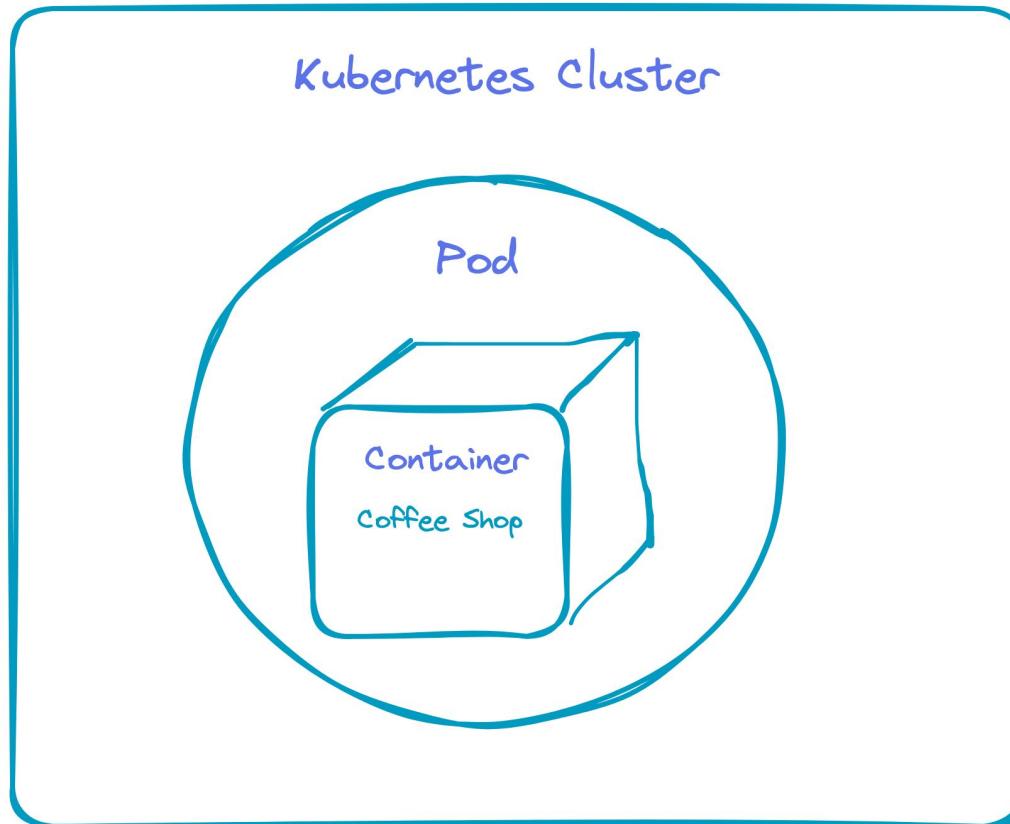
# deployment.yaml

# 7 Deploy on Kubernetes

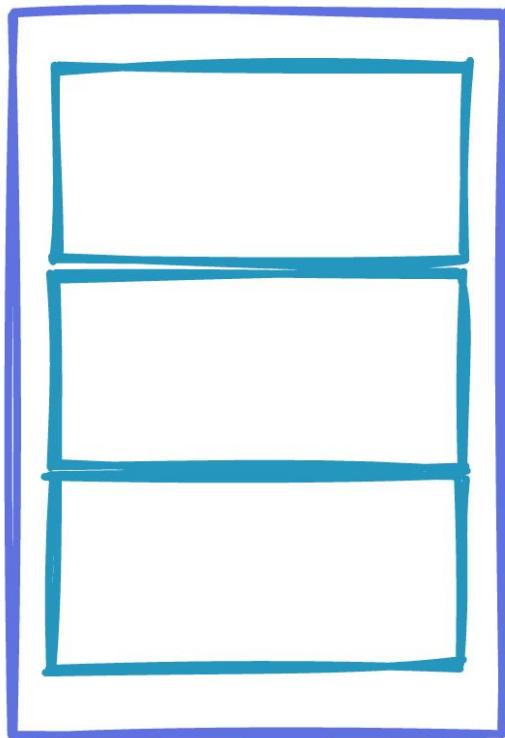
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: coffee-shop
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      coffee-shop: web
  template:
    metadata:
      labels:
        coffee-shop: web
    spec:
      containers:
        - name: coffee-shop
          image: coffee-shop
          imagePullPolicy: Always
```

7

# Deploy on Kubernetes



# Scaling Option 1



Vertical



## Set Resources

```
resources:  
  limits:  
    cpu: 1m  
    memory: 10Mi  
  requests:  
    cpu: 1m  
    memory: 10Mi
```

8

## Set Resources

3

→ 2.002422959s

30

→ 2.001635459s

300

→ 3.78216496s

3000

→ OOM Killed      Restarts: 1

9

# Bump resources

```
resources:  
limits:  
    cpu: 4  
    memory: 3092M  
requests:  
    cpu: 2  
    memory: 1024M
```

9

Bump resources

30,000



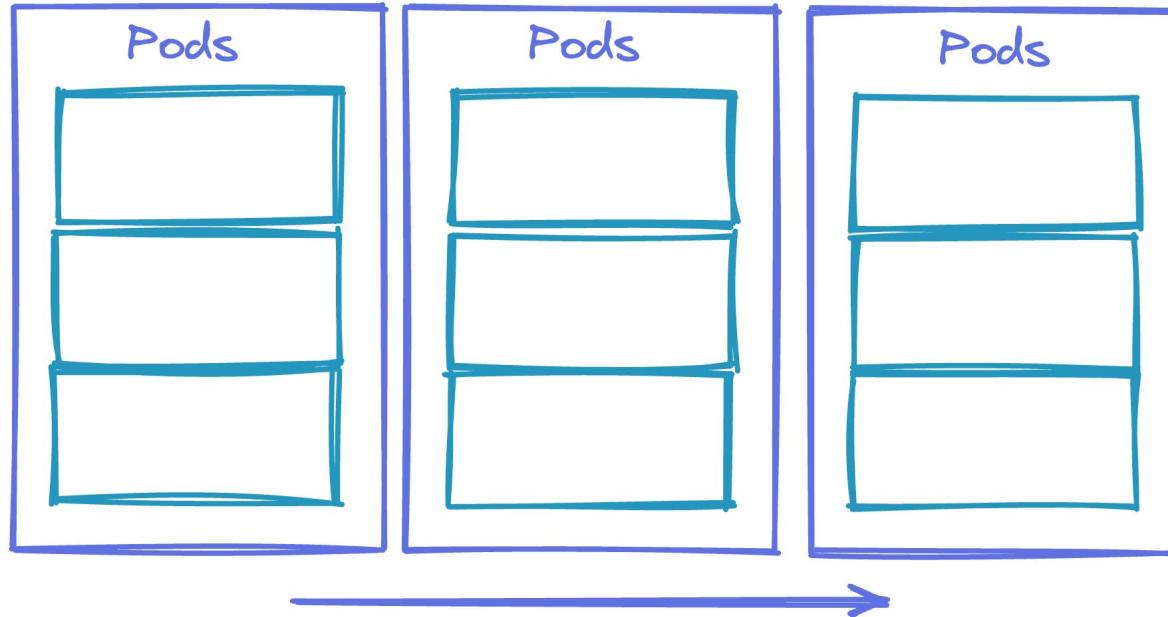
3.797142293s

3,000,000



OOM killed

# Scaling Option 2



Horizontal

10

Scale up pods

replicas: 2

10

# Scale up pods

## ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress

metadata:
  namespace: default
  name: coffee-shop
  labels:
    ingress-controller: nginx
  annotations:
    kubernetes.io/ingress.class: "nginx"

spec:
  rules:
  - host: "localhost"
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: coffee-shop
            port:
              number: 80
```

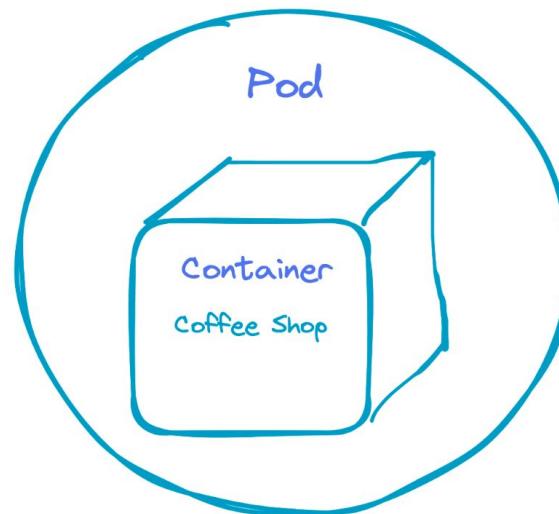
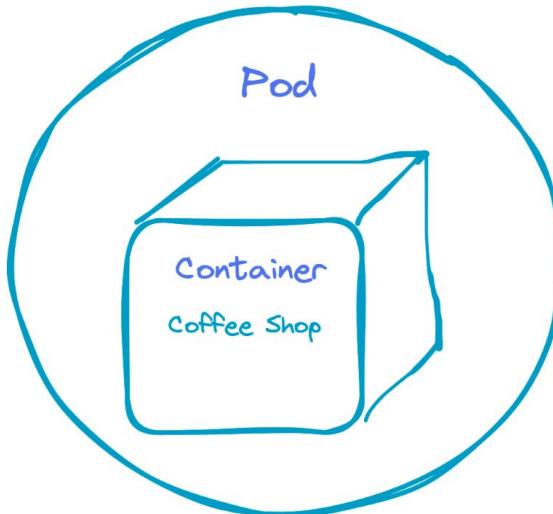
## deployment.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: coffee-shop
  labels:
    run: coffee-shop
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    run: coffee-shop
  type: NodePort
  sessionAffinity: None
```

10

# Scale up pods

Kubernetes Cluster



10

Scale up pods

30,000

→ 2.265168584s

30,000

→ 2.210111751s

# Watch out...

- Do you need this?
- Use Wait Groups
- Resources have limits
- Infinite goroutines?

Take Payment

2 Pods

Steam Milk

2 Pods

Make Espresso

4 Pods

# Kubernetes Cluster

Main



Take Payment



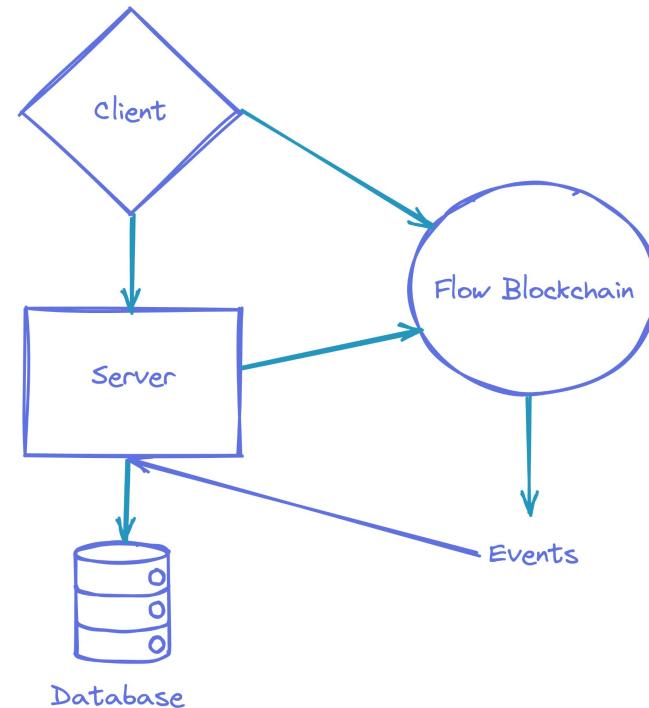
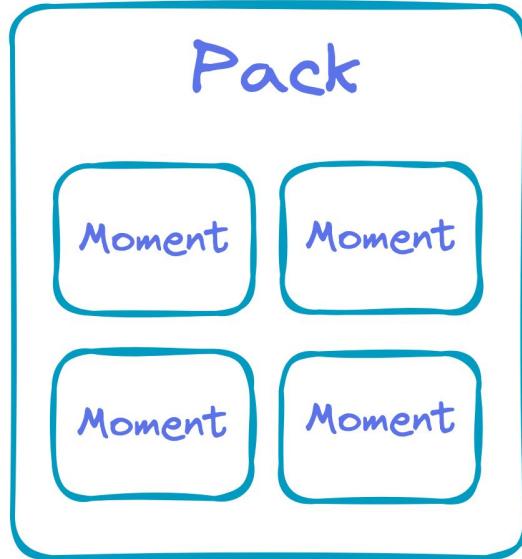
Steam Milk



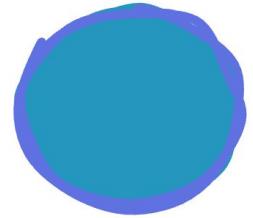
Make Espresso



# Case Study

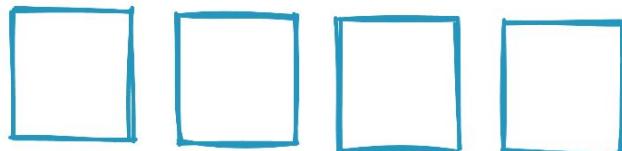


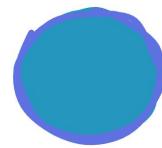
- Check Events
- Minting
- Sending Transactions
- Checking Transactions



# No Concurrency

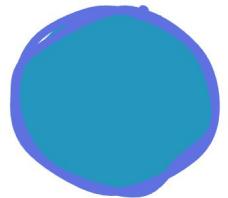
One at a time





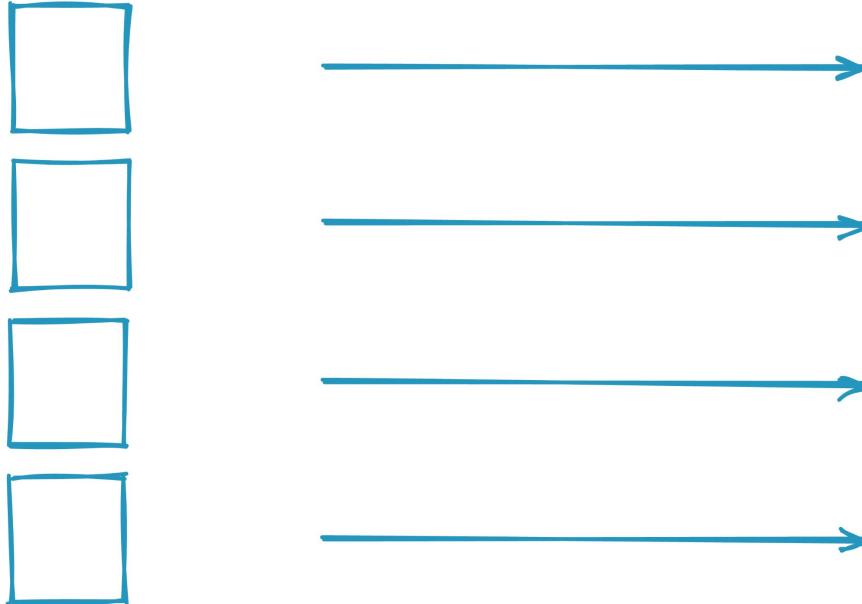
# No Concurrency

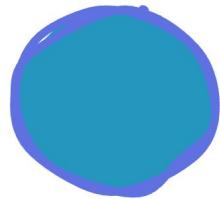
```
for {
    select {
        case <-ticker.C:
            checkEvents()
            mint()
            sendTransactions()
            checkTransactions()
        case <-app.quit:
            cancel()
            ticker.Stop()
            return
    }
}
```



# Use Goroutines

Four at a time



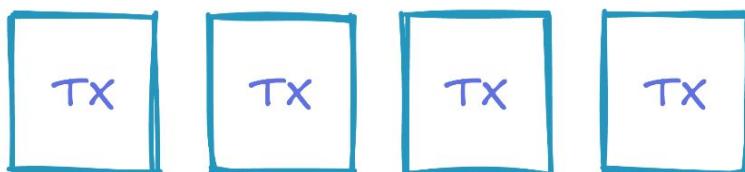


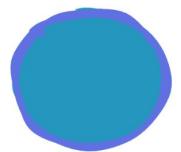
# Use Goroutines

```
go checkEvents()
go mint()
go sendTransactions()
go checkTransactions()
```

# Send Transactions.

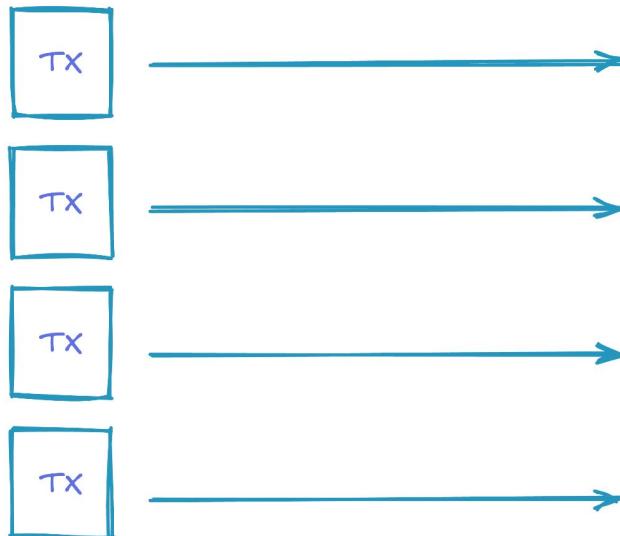
One at a time





# Add More Goroutines

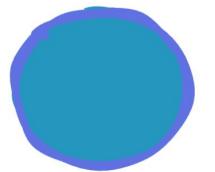
MORE at a time



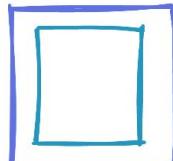


# Add More Goroutines

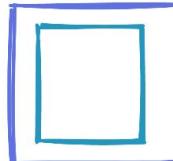
```
for _, transaction := range ts {
    wg.Add(1)
    go func(wg *sync.WaitGroup, tx *transactions.SendableTransaction) {
        defer wg.Done()
        if err := processSendableTransaction(); err != nil {
            logger.Warn("error processing tx", err)
        }
    }(&transaction)
}
wg.Wait()
```



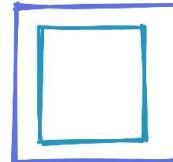
# Split Deployments



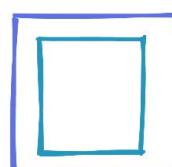
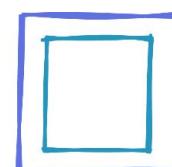
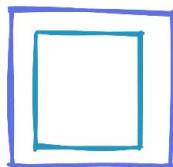
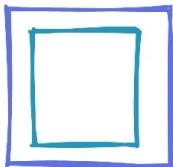
Check Events



Minting

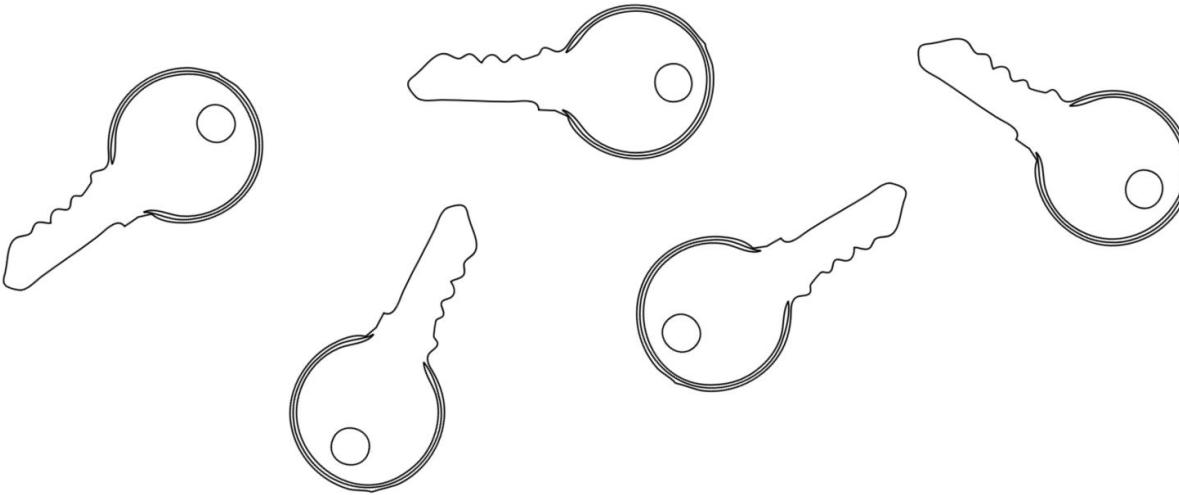


Check Transactions



Send Transactions

# Keys?



# How did we scale?

- By using Goroutines (just a few)
- Scale horizontally + vertically
- Split different responsibilities  
in different deployments

- Start with no concurrency
- Use Goroutines
- Use Goroutines inside of Goroutines
- Scale vertically - add resource
- Scale horizontally - add pods
- Split deployments according to Goroutine usage

# Thank you & Grazie



<https://bit.ly/golab-coffee>

Code examples  
Slides  
Tutorial

sadiefreeman@gmail.com

# THANK YOU!

@Sadieff  
on Github

sadiefreeman@gmail.com



[bit.ly/killerGo](http://bit.ly/killerGo)