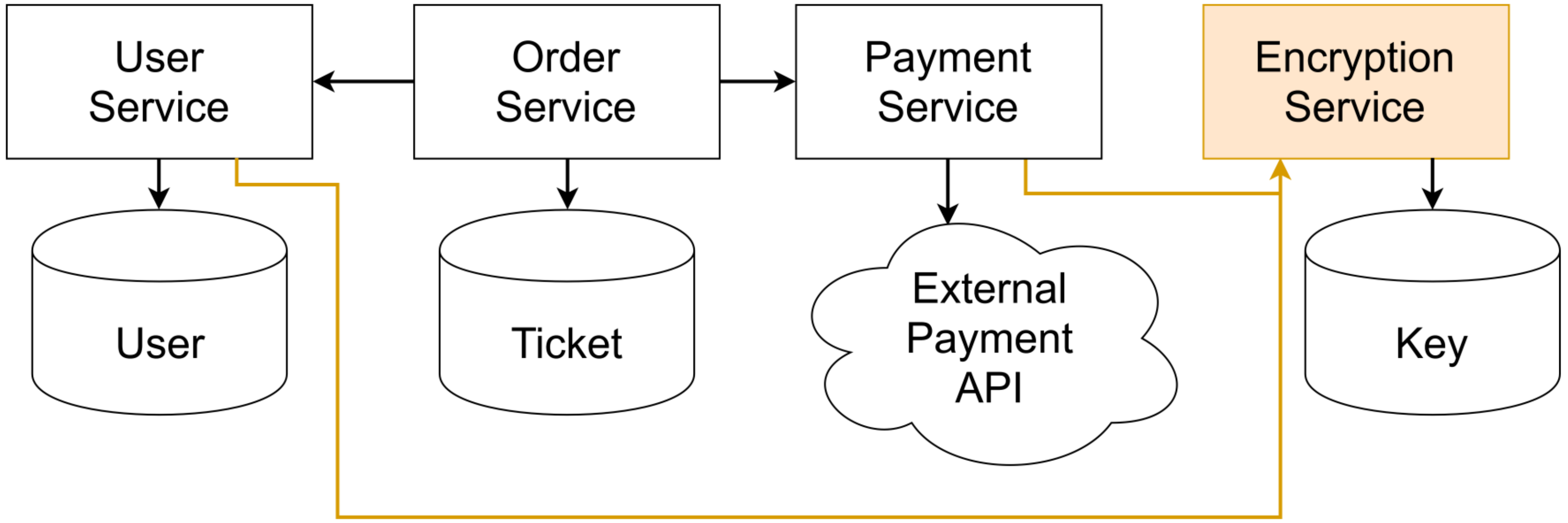
Cloud-based Online Concert Ticketing System

Zhong Xi Lu

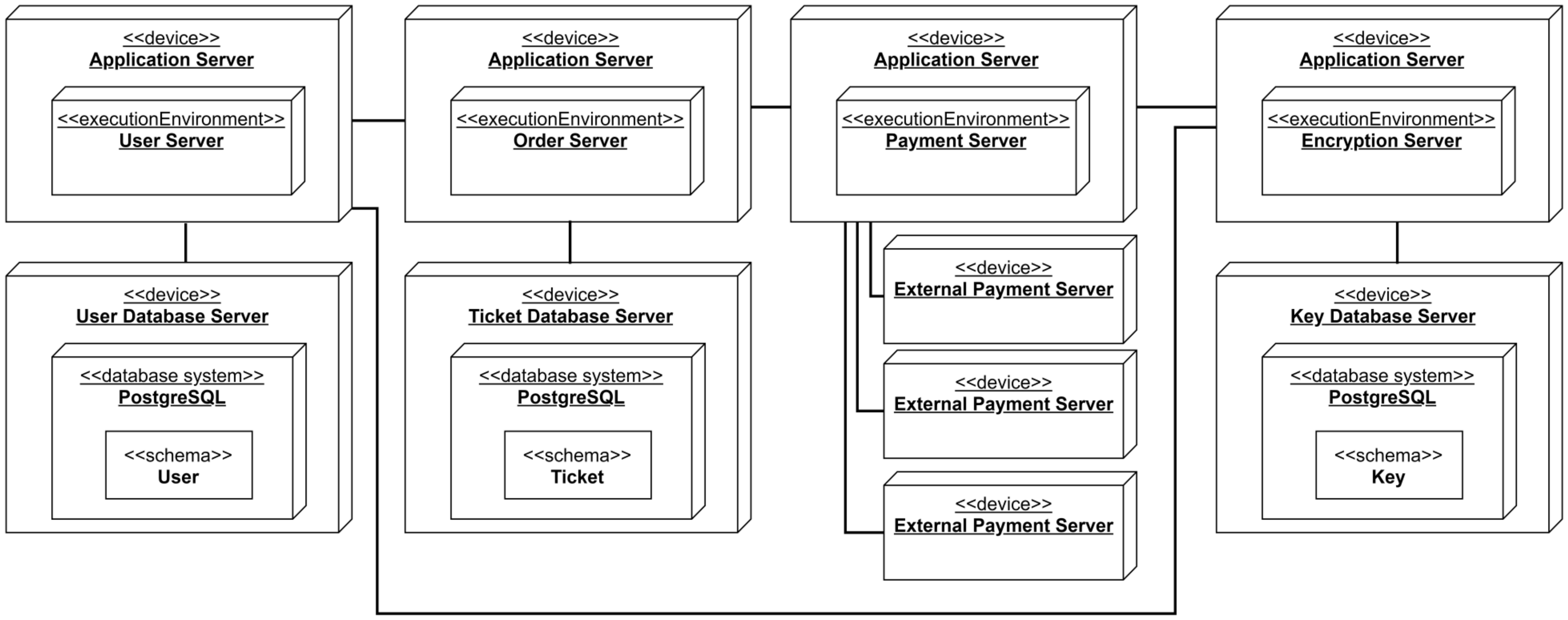
University of Antwerp

Architecture Overview

General Overview



Deployment Diagram



Https (REST API)

Class Diagram and Database Schemas

UserService

POST users
PUT users
GET users/{user_id}
GET users
POST users/verify

OrderService

POST orders

PaymentService

POST payment

EncryptionService

POST keys
GET keys/{user_id}

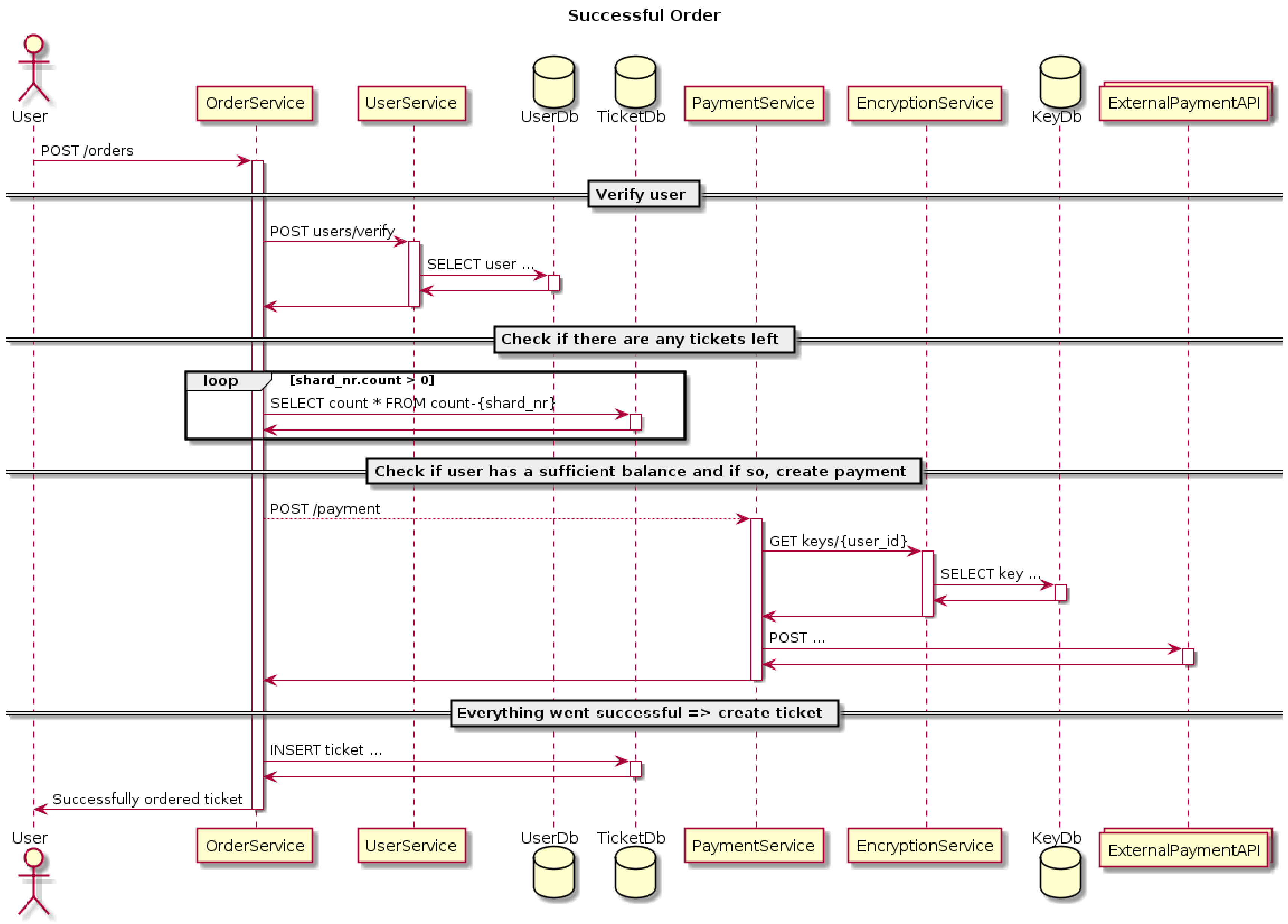
User
id
username
password
gender
token
country
city
zip_code
street
card_type
card_holder_name
card_number
expiration_date_month
expiration_date_year
cvv

Ticket-*i*
id
user_id
token

TicketsLeft-*i*
count

Key
id
user_id
key
IV

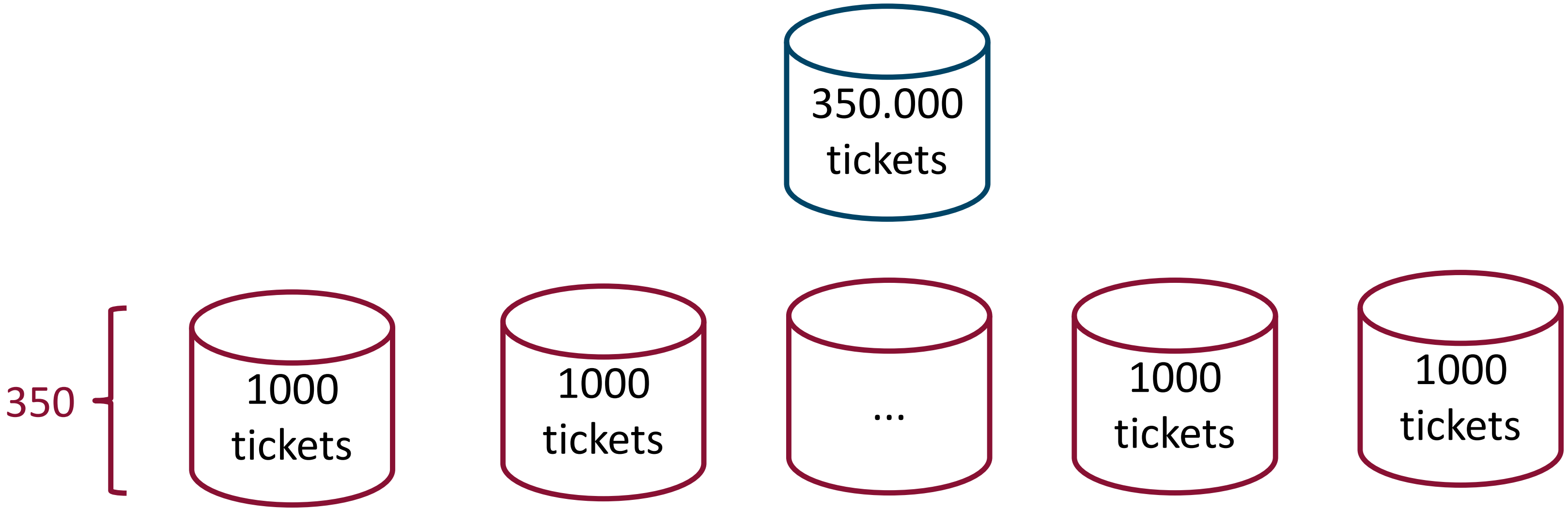
Sequence Diagram



Revisiting Old Conclusions

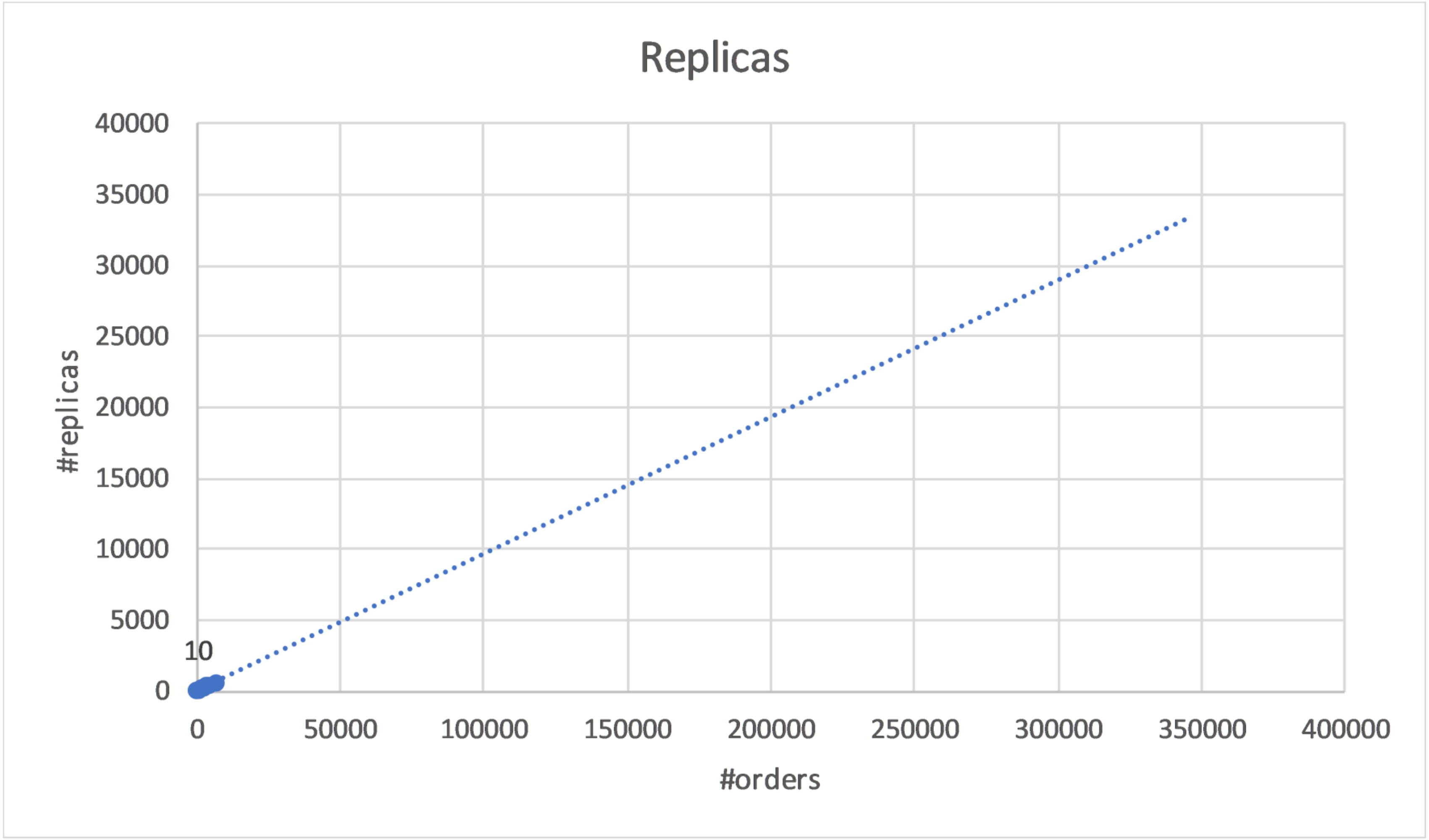
Bottlenecks

- Database => database sharding



- External payment API => solution?

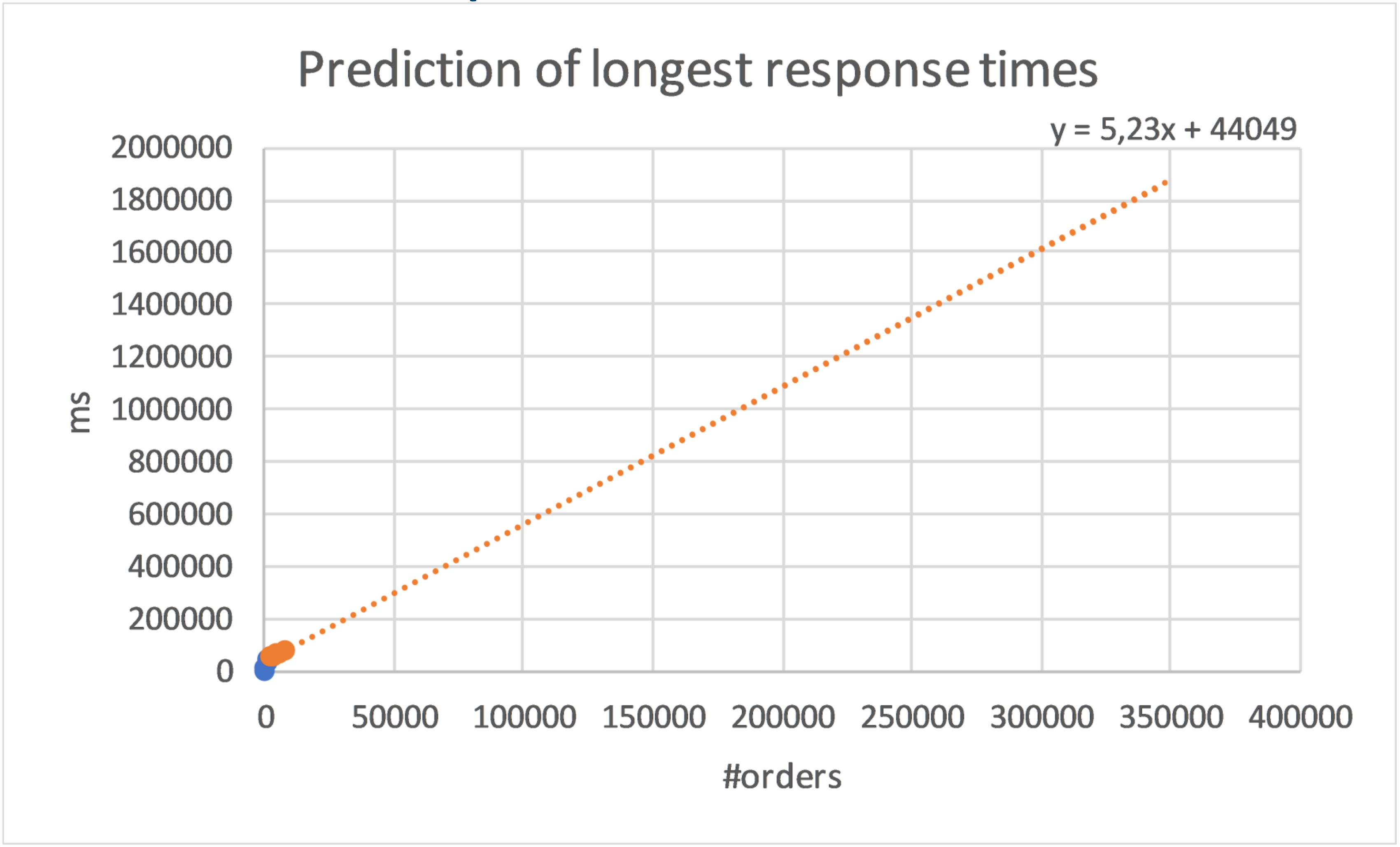
Prediction number of replicas



Prediction number of replicas

- 35.000 replicas?
- Probably only **350** (or more?) replicas = 350 shards
- Maybe even less, one replica can handle **multiple requests concurrently**
- Note: replica ≠ server

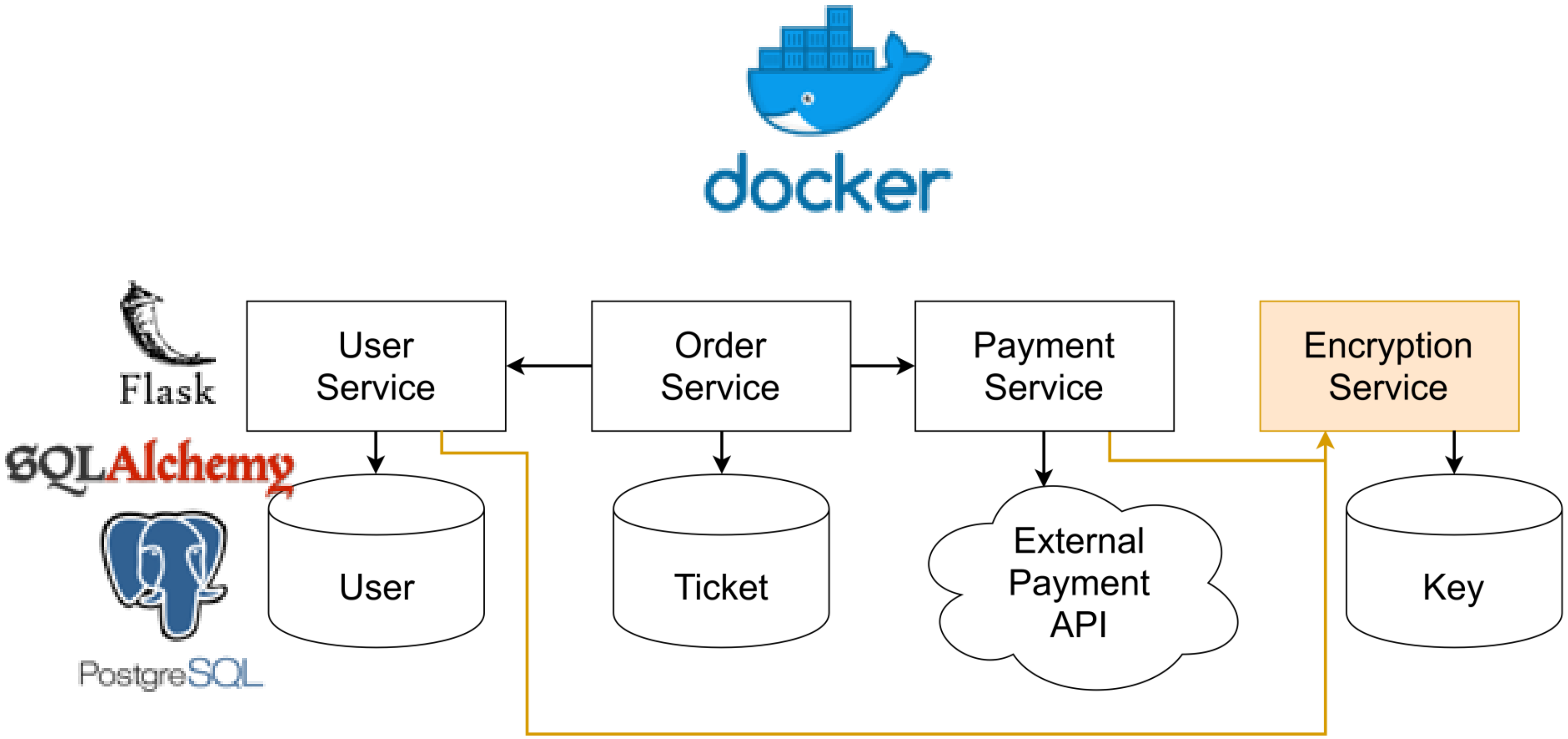
Prediction response times



Ordering 350.000 tickets at same time => longest response time = 1.874.549ms ≈ 31min.
Ordering **106.300 tickets** at same time => longest response time ≈ 600.000ms = **10 min.**

Set up

Architecture

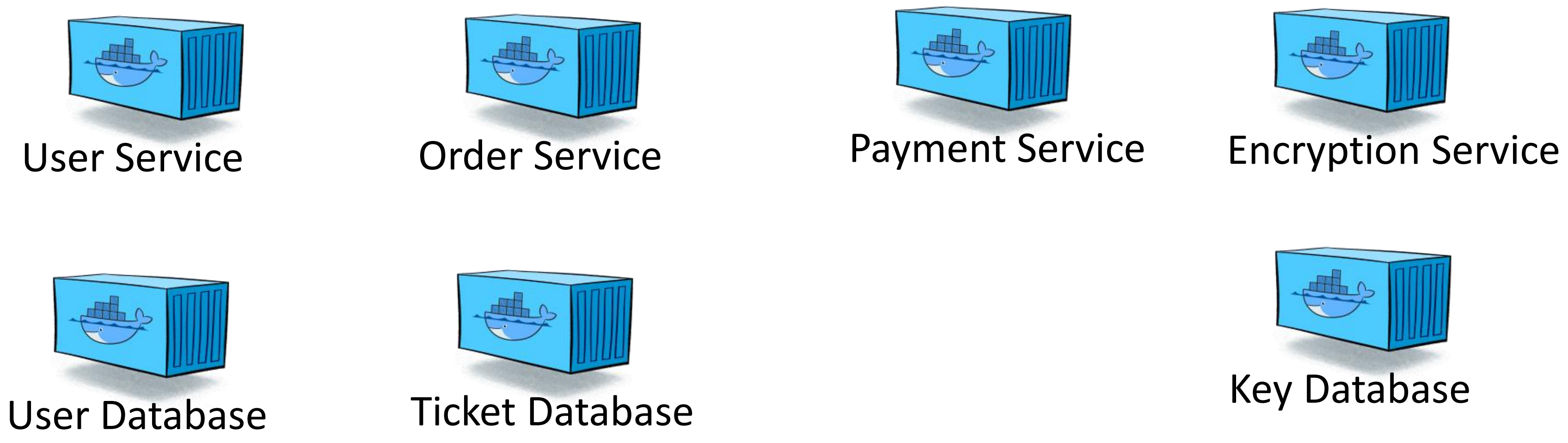


Docker containers

kubernetes

minikube

Note: Everything ran on just **one machine**



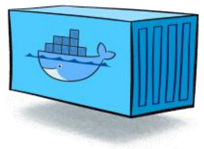
Autoscaler

- Kubernetes **Horizontal Pod Autoscaler** (HPA)

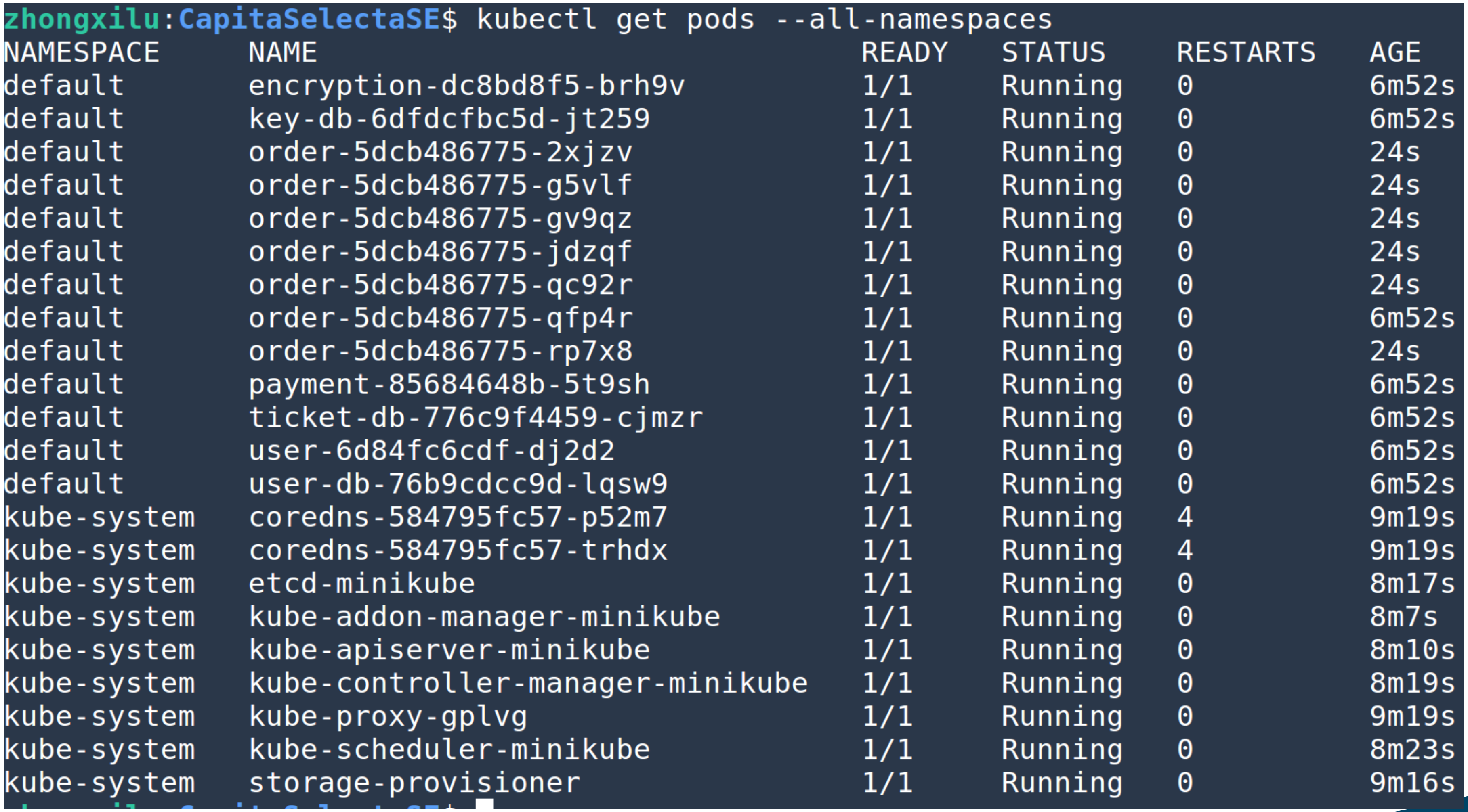
```
desiredReplicas =
  ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )]
```

(https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#algorithm-details)

- However, best **scale to maximum at start**
- Scale Order Service


Order Service

Minikube Setup

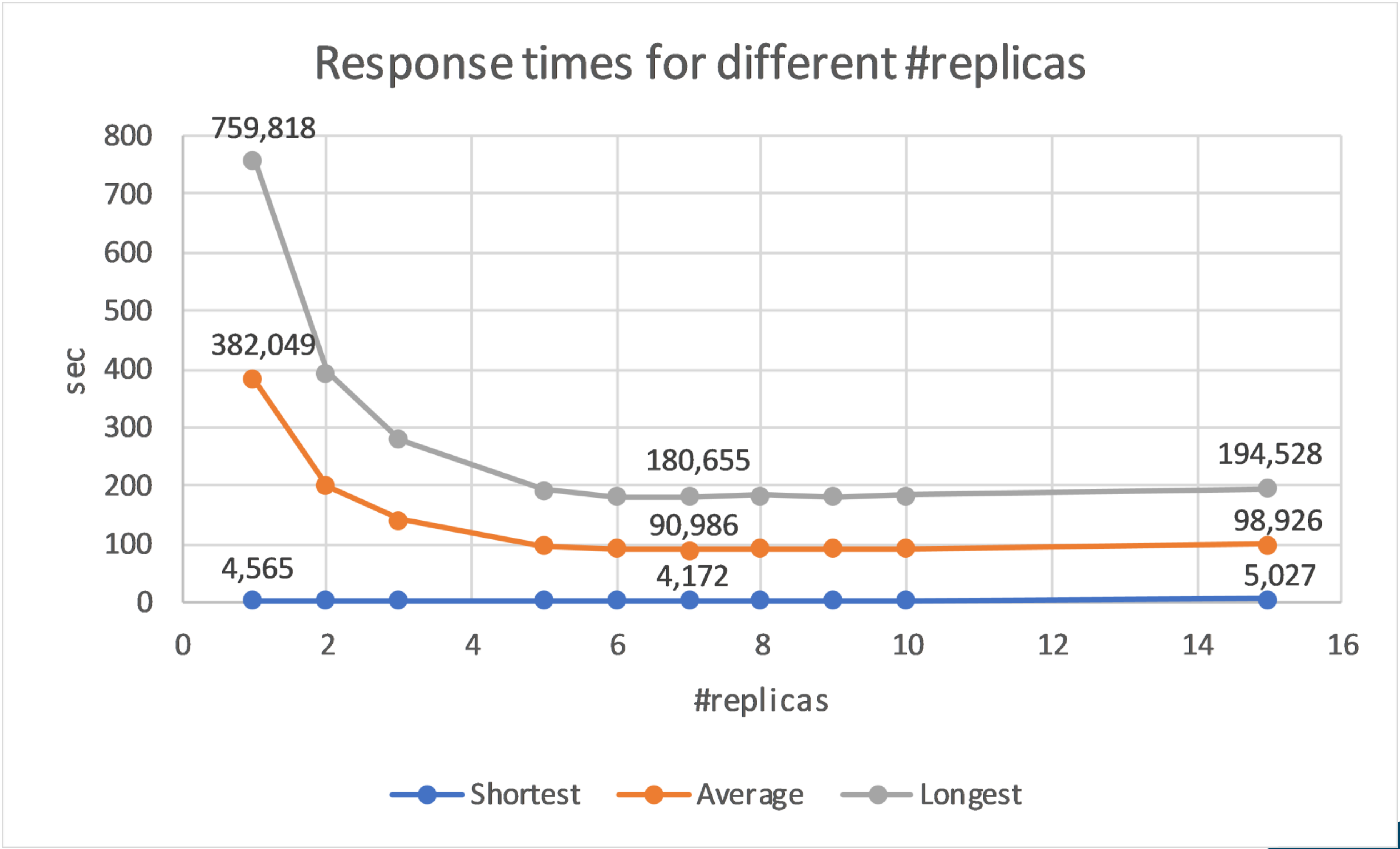

```
zhongxilu:CapitaSelectaSE$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY   STATUS    RESTARTS   AGE
default       encryption-dc8bd8f5-brh9v              1/1     Running   0          6m52s
default       key-db-6dfdcfbc5d-jt259                1/1     Running   0          6m52s
default       order-5dcb486775-2xjzv                 1/1     Running   0          24s
default       order-5dcb486775-g5vlf                 1/1     Running   0          24s
default       order-5dcb486775-gv9qz                 1/1     Running   0          24s
default       order-5dcb486775-jdzqf                 1/1     Running   0          24s
default       order-5dcb486775-qc92r                 1/1     Running   0          24s
default       order-5dcb486775-qfp4r                 1/1     Running   0          6m52s
default       order-5dcb486775-rp7x8                 1/1     Running   0          24s
default       payment-85684648b-5t9sh                1/1     Running   0          6m52s
default       ticket-db-776c9f4459-cjmzr             1/1     Running   0          6m52s
default       user-6d84fc6cdf-dj2d2                  1/1     Running   0          6m52s
default       user-db-76b9cdcc9d-lqsw9               1/1     Running   0          6m52s
kube-system   coredns-584795fc57-p52m7               1/1     Running   4          9m19s
kube-system   coredns-584795fc57-trhdx               1/1     Running   4          9m19s
kube-system   etcd-minikube                          1/1     Running   0          8m17s
kube-system   kube-addon-manager-minikube            1/1     Running   0          8m7s
kube-system   kube-apiserver-minikube                1/1     Running   0          8m10s
kube-system   kube-controller-manager-minikube       1/1     Running   0          8m19s
kube-system   kube-proxy-gplvg                       1/1     Running   0          9m19s
kube-system   kube-scheduler-minikube                1/1     Running   0          8m23s
kube-system   storage-provisioner                    1/1     Running   0          9m16s
```

Results

Load Test Script

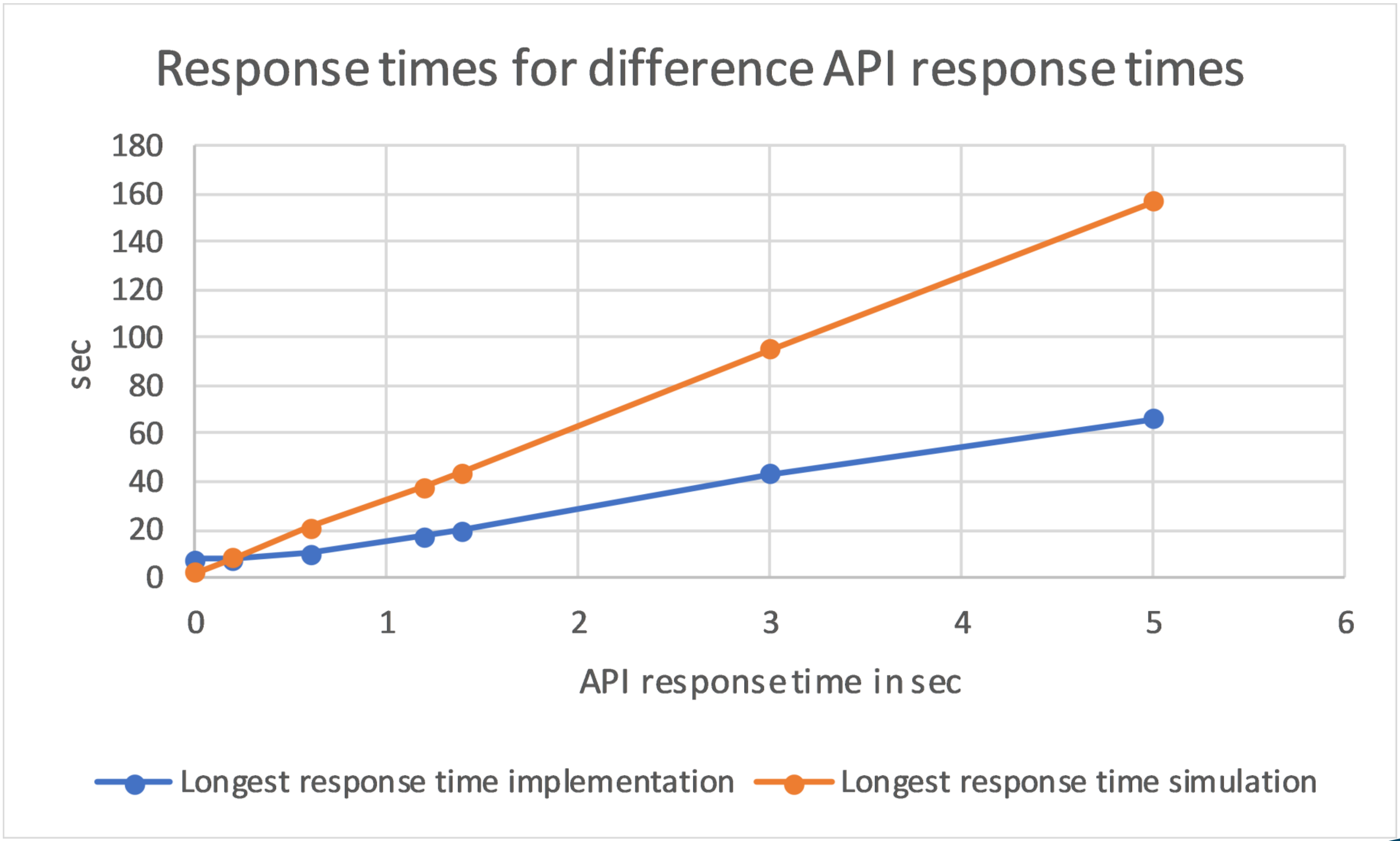
- Before: *n* registered users
- Retrieve all users
- Order ticket for each registered user at the same time (***n* requests**)

Number of replicas for Order service



Ordering 10.000 tickets

External API bottleneck



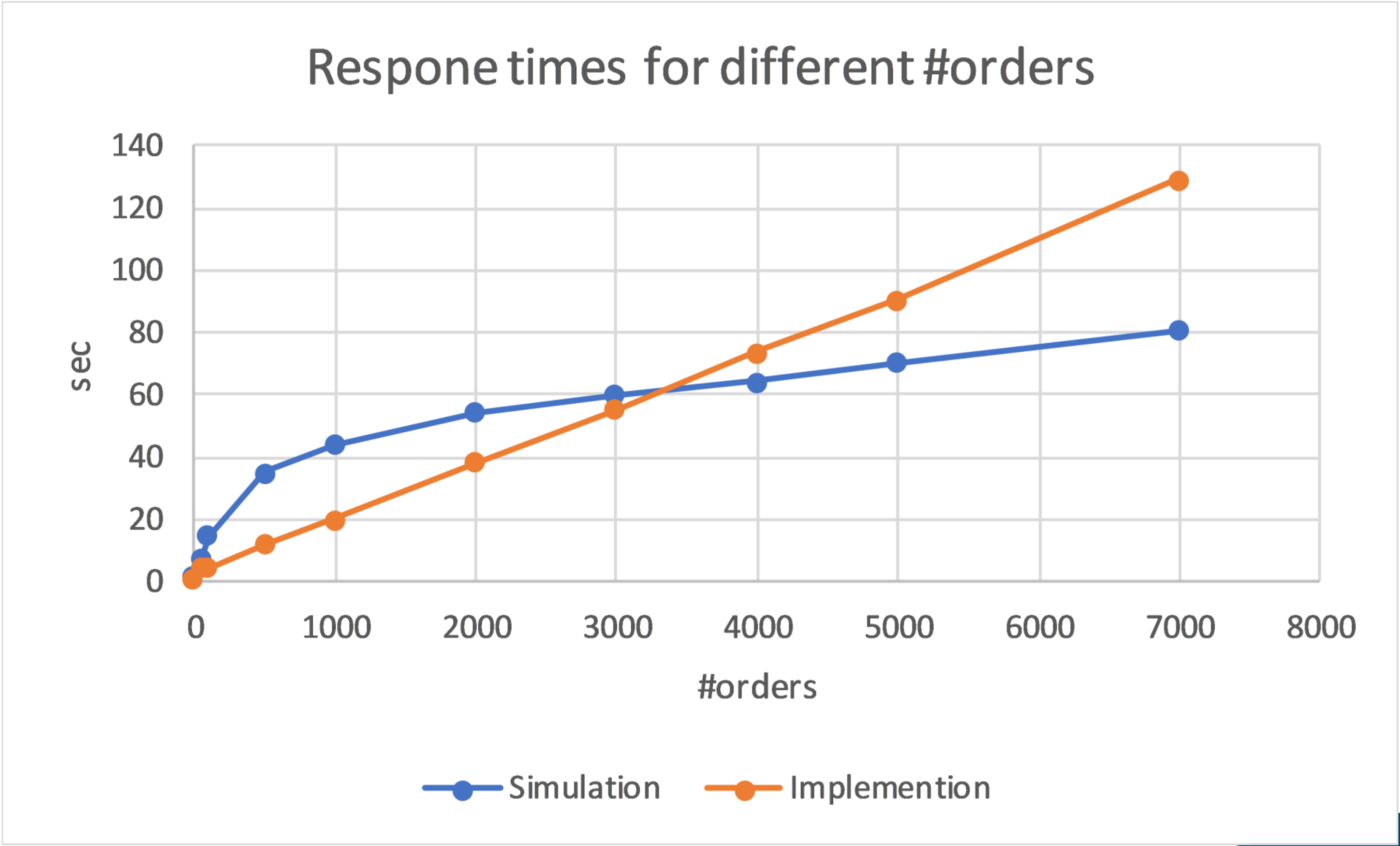
Ordering 1000 tickets

Encryption

- Ordering 10.000 tickets at the same time:

	w/ encryptions	w/o encryptions
Shortest response time	4.5 sec	4.29 sec
Average response time	90.57 sec	87.711 sec
Longest response time	178.023 sec	172.408 sec

Comparison with ABS simulations



Peak Load

- **31.186 tickets** ordered at same time for response time less than 10 min
- *Important*: only using **one machine**

Conclusion (again)

- So does it respond in less than 10 min during peak load?
  - *It depends...*
    - *How much is the actual **peak load**?*
    - *How much **resources** available?*
    - *Processing time **external payment API**?*
  - But system **scales dynamically** on cloud