

The Era of the Cloud

Florin Barbuceanu

Senior Solutions Architect @ Amazon Web Services Berlin

florin.gabriel.barbuceanu@gmail.com

What is Amazon Web Services?



A broad and deep platform that helps customers build sophisticated, scalable, secure applications

What is the Cloud?



IT as a **commodity**



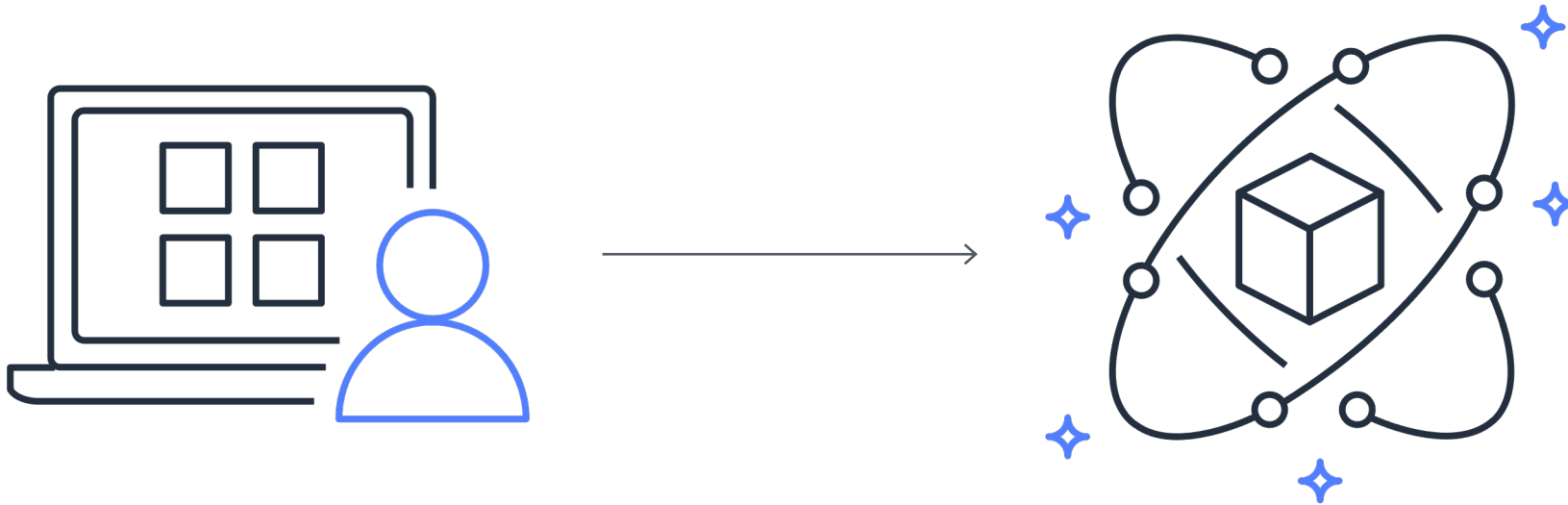
No more **building,**
operating, maintaining
data centers

What is an application?



Software system designed to serve end users
for example web browser, video game

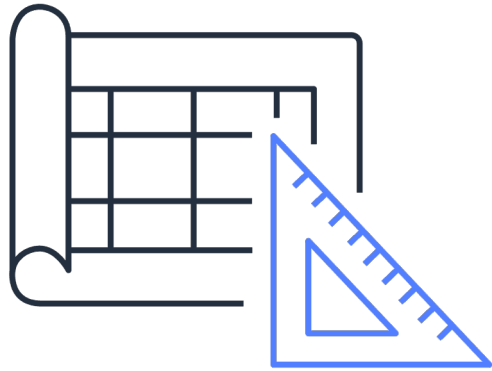
What is a server?



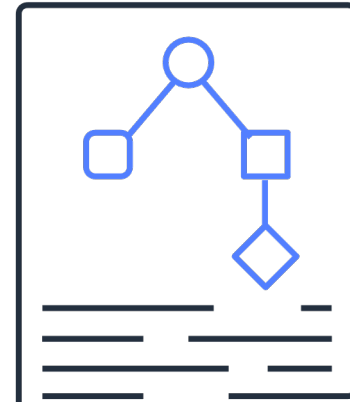
A **server** is an application that exposes functionality for clients

What is an API?

Application Programming Interface



Abstract **standard of interaction**



How do classes, applications, computers **interact** and **communicate**?

What is an API?

```
interface BankingService {  
    double getAccountBalance(String accountId);  
}
```

What do we achieve by using **interfaces**?

- Defined a **contract**
- **Decoupled** clients from implementors

The simplest implementation of this API?

```
class DummyBankingService implements BankingService {  
    public double getAccountBalance(String accountId) {  
        return 42.0;  
    }  
}
```

Does it **work**?

- **Yes***

Is it **correct**?

- **No**

* - The API clients can work with this **mock implementation** until the real implementation is delivered

Another example of API?

GET

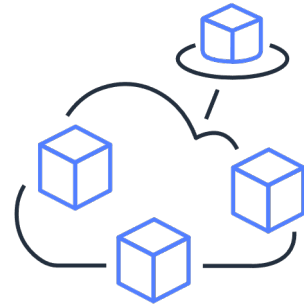
<https://api.apistorebt.ro/bt/sb/bt-psd2-aisp/v2/accounts/K13RONCRT0060214301>

```
{
  "account": {
    "iban": "RO98BTRLRONCRT0ABCDEFGHI",
    "resourceId": "K13RONCRT0060214301",
    "currency": "RON",
    "product": "Cont de debit",
    "name": "Contul meu",
    "cashAccountType": "CurrentAccount",
    "balances": [{
      "balanceType": "expected",
      "creditLimitIncluded": false,
      "balanceAmount": {
        "currency": "RON",
        "amount": 675.502
      },
      "referenceDate": "2019-03-26"
    }
  ],
  "_links": {
    "balances": {
      "href": "https://apistorebt.ro/bt/sb/bt-psd2-aisp/v1/accounts/K13RONCRT0060214301/balances"
    },
    "transactions": {
      "href": "https://apistorebt.ro/bt/sb/bt-psd2-aisp/v1/accounts/K13RONCRT0060214301/transactions"
    }
  }
}
```

What are the Benefits of the Cloud?



Cost
Optimisation



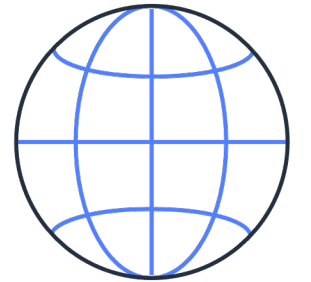
Elasticity



Agility



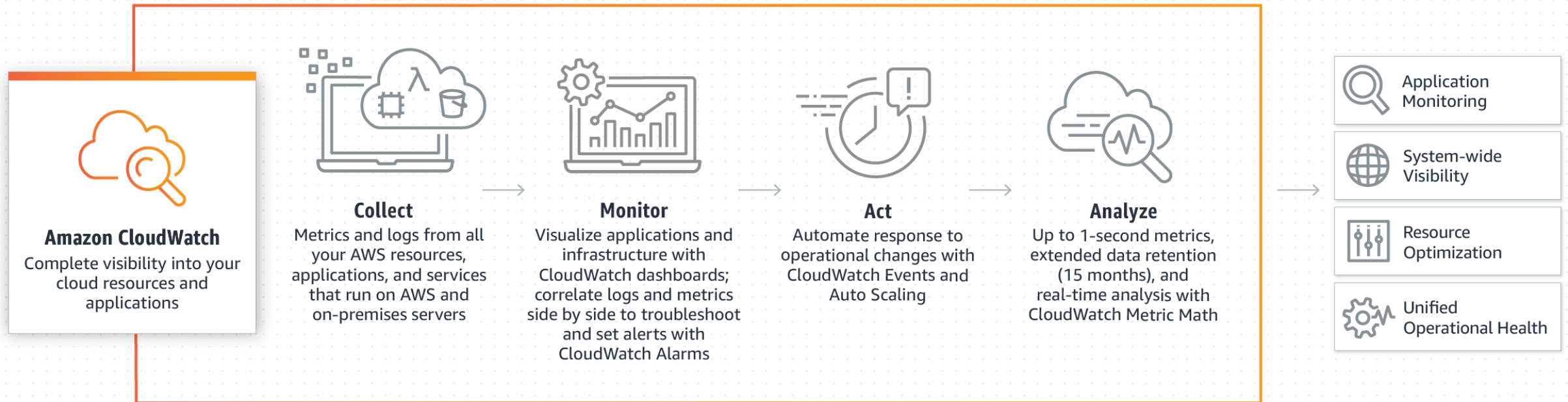
Security



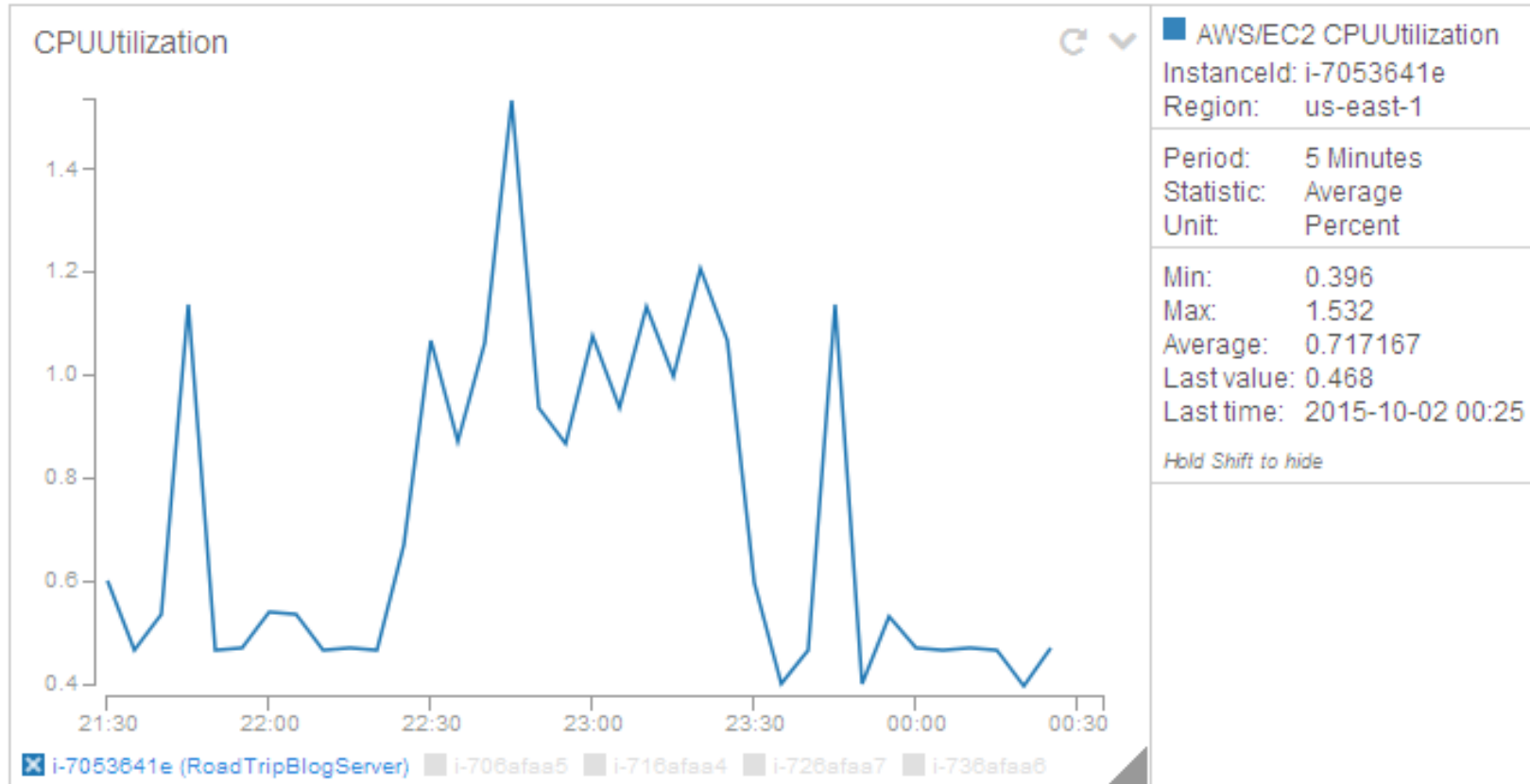
Global Footprint

How is all this elasticity possible?

- **Observability** – metrics such as CPU utilisation
- **Orchestration** – create and release virtualized resources



Example CloudWatch Metric



Case Study – Netflix



NETFLIX

Netflix on AWS

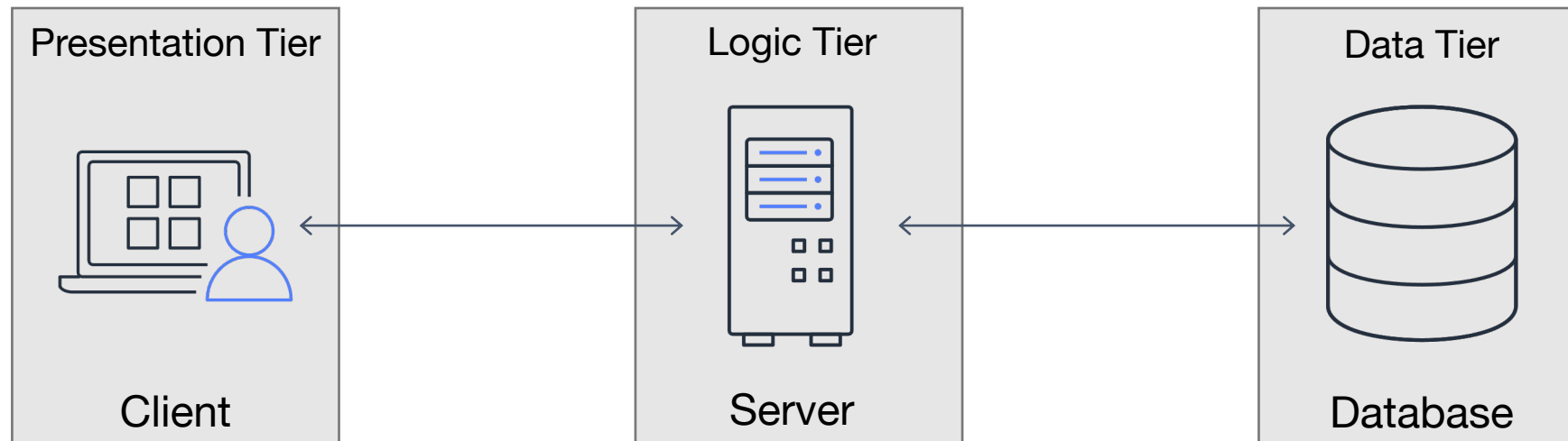
Netflix is the world's leading internet television network, with more than 200 million members in more than 190 countries enjoying 125 million hours of TV shows and movies each day. Netflix uses AWS for nearly all its computing and storage needs, including databases, analytics, recommendation engines, video transcoding, and more—hundreds of functions that in total use more than 100,000 server instances on AWS.

[Customer Stories](#) | [Architecture](#) | [Additional Resources](#)

Read more: <https://aws.amazon.com/solutions/case-studies/netflix/>

Modernising Applications

The three-tier web application



- How can we scale this architecture?
 - **Monitor utilisation** – build our own or use existing solutions
 - But who **monitors the monitoring** system?
 - How can we ensure we add hardware **in time**?
 - What if we reach **limits**? (storage, CPU, space in datacenter facility?)

Scalability Patterns

Vertical Scaling



Increasing the machine size, but..

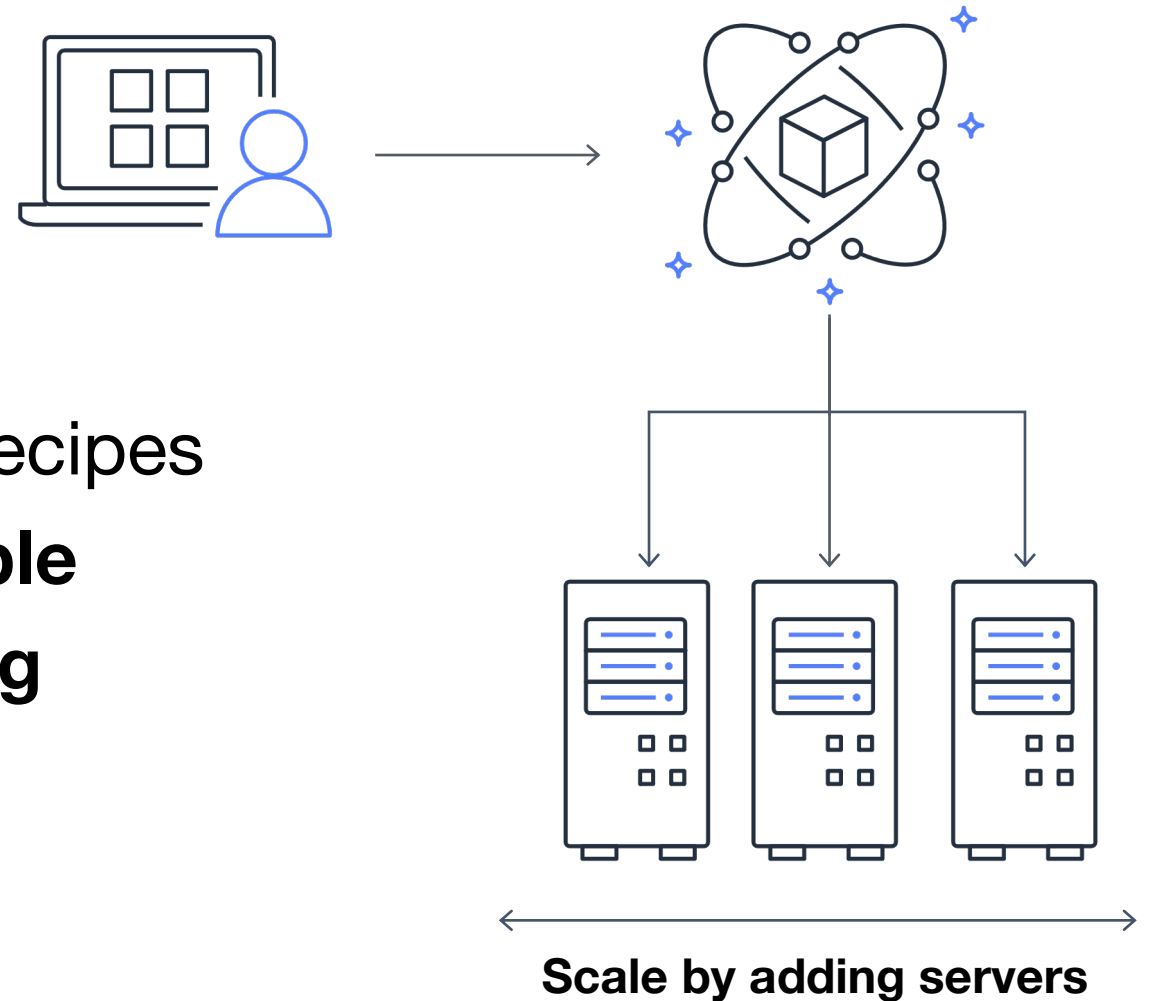
- We run out of bigger processors to upgrade to, or
- They get so expensive that it's not worth it anymore

Scale by growing servers

Scalability Patterns

Horizontal Scaling

- Stateless applications
- Easy to recreate and deploy by recipes
- Infrastructure becomes **immutable**
- **Load Balancer** and **Auto Scaling**
- Better cost optimisations!



What are Microservices?

Well, it's very easy...

“Share-nothing distributed architecture, where each microservice is bounded by domain, and relies on APIs to interact and implement functionality, therefore streamlining independent scalability of components and autonomy of developer teams.”

Yeah, I know some of those words

“Share-nothing distributed architecture, where each microservice is bounded by domain, and relies on APIs to interact and implement functionality, therefore streamlining independent scalability of components and autonomy of developer teams.”

What are Microservices?

Now seriously

“Share-nothing distributed architecture, where each microservice is bounded by domain, and relies on APIs to interact and implement functionality, therefore streamlining independent scalability of components and autonomy of developer teams.”

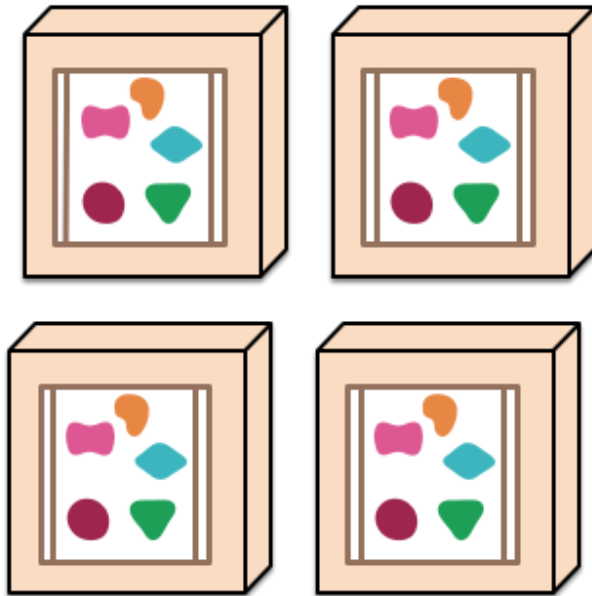
Scalability Patterns

Microservices

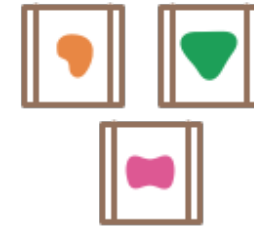
A monolithic application puts all its functionality into a single process...



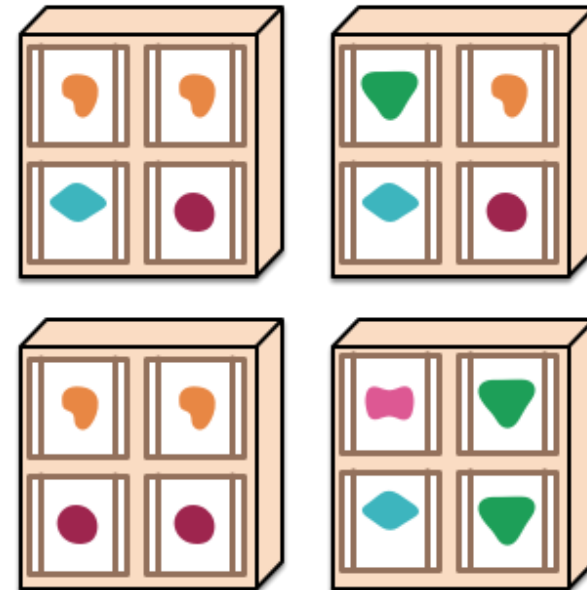
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...

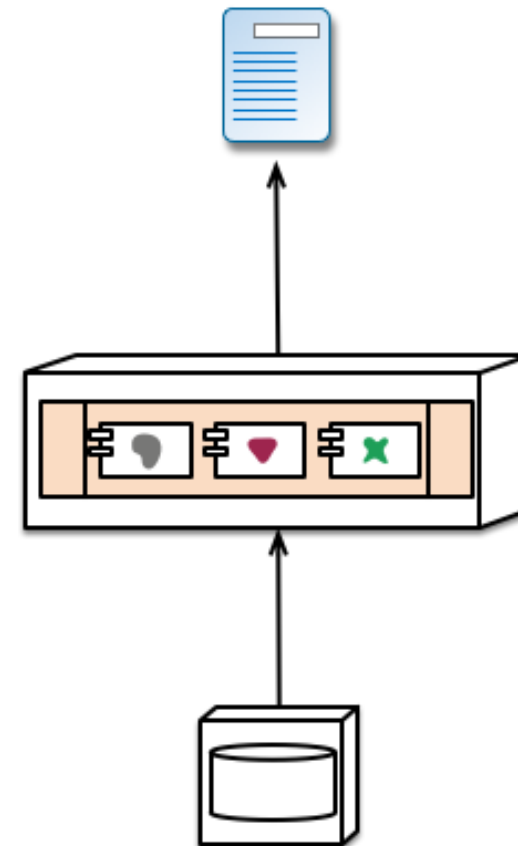
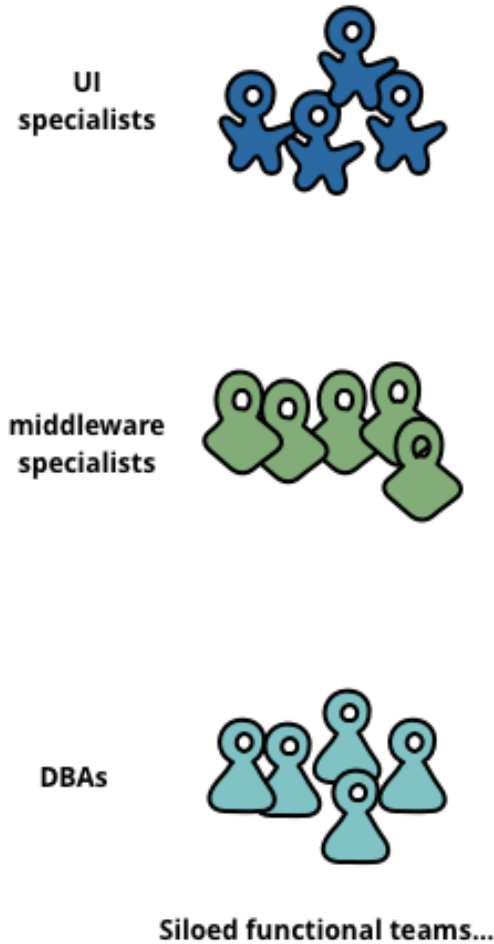


... and scales by distributing these services across servers, replicating as needed.



Microservices - Organisational Change

From This

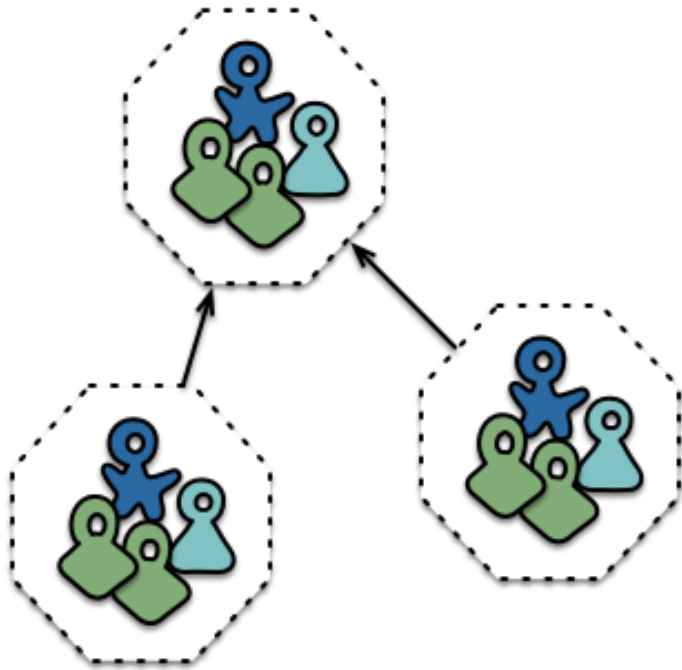


... lead to siloed application architectures.
Because Conway's Law

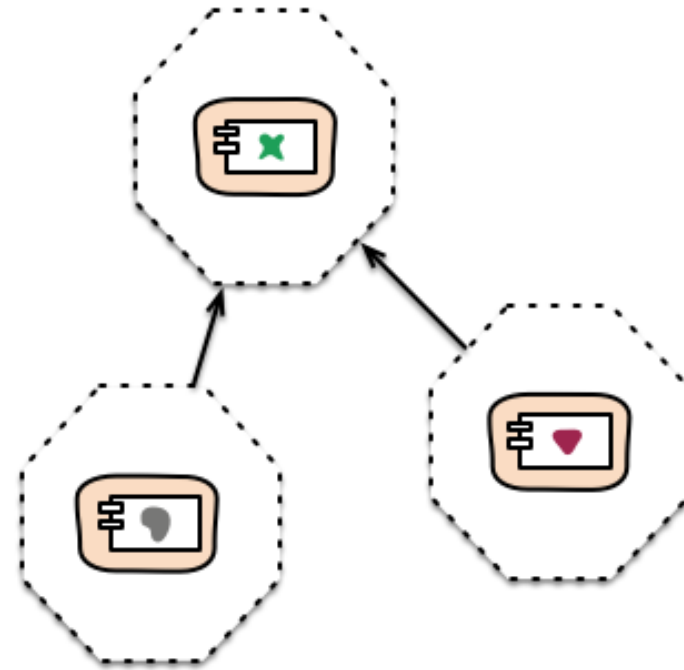
Read more: <https://martinfowler.com/articles/microservices.html>

https://en.wikipedia.org/wiki/Conway%27s_law

Microservices - Organisational Change To This



Cross-functional teams...



... organised around capabilities
Because Conway's Law

Distributed Systems

Quick Detour

```
board.move(pacman, user.joystickDirection());
```

In how many ways can this code fail?

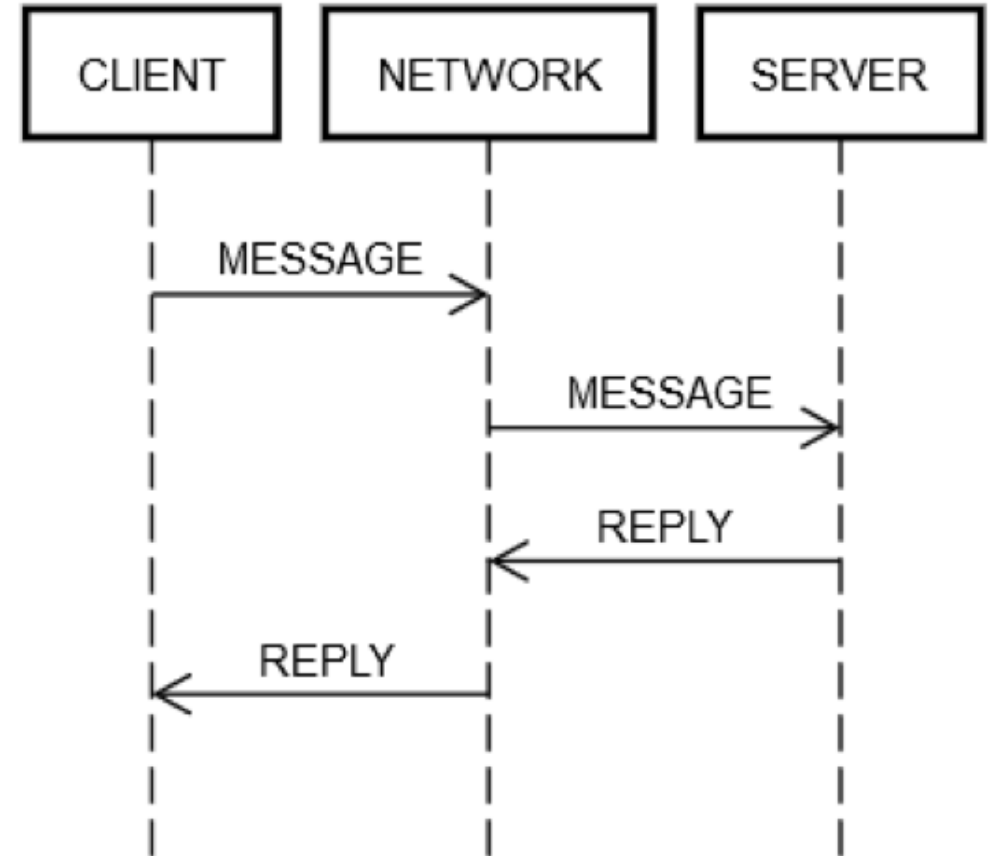
- Consider a local Java application
- Can we handle all of them?

Distributed Systems

What if the game was multiplayer?

8 additional steps:

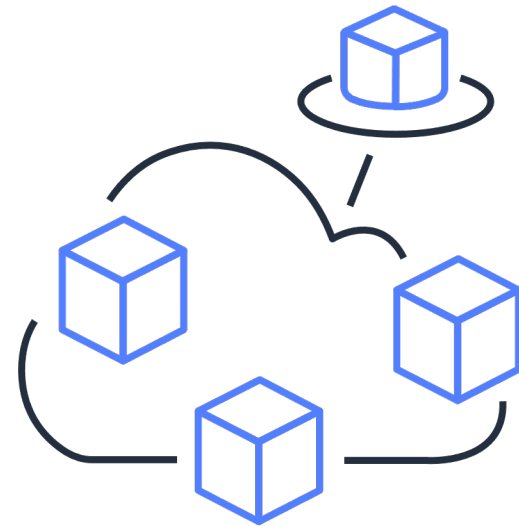
1. POST REQUEST
2. DELIVER REQUEST
3. VALIDATE REQUEST
4. UPDATE SERVER STATE
5. POST REPLY
6. DELIVER REPLY
7. VALIDATE REPLY
8. UPDATE CLIENT STATE



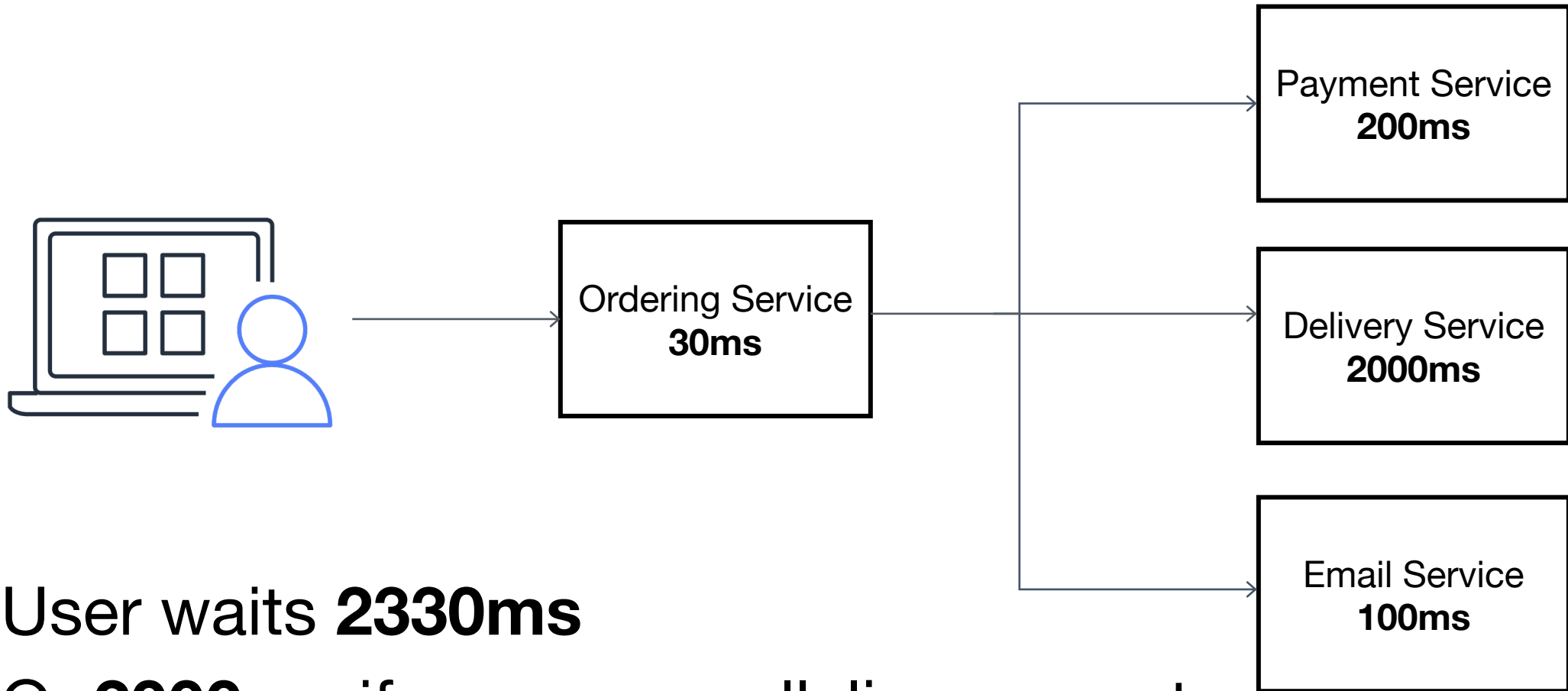
Serverless

Make it somebody else's problem...

- You don't manage servers
- Focus on what matters
- From few requests per day to millions per second



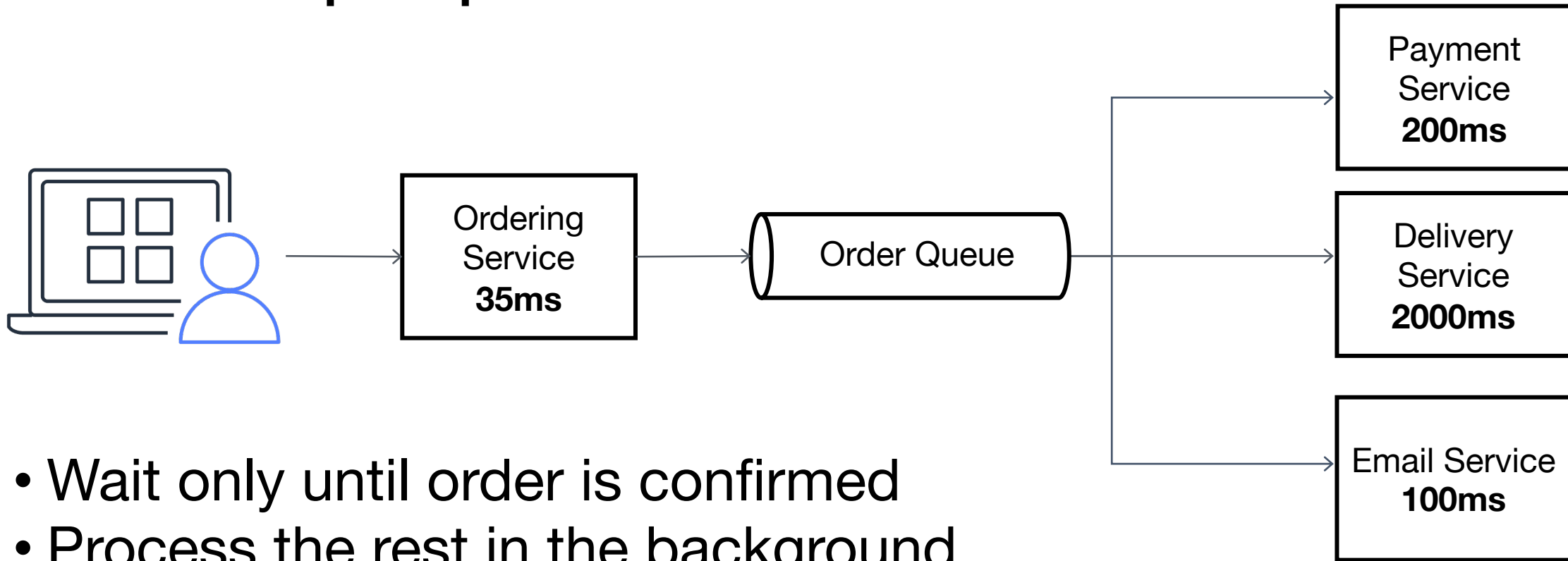
Synchronous Communication



- User waits **2330ms**
- Or **2000ms** if we can parallelise requests

Asynchronous Communication

Decouple producers and consumers



- Wait only until order is confirmed
- Process the rest in the background
- Avoid overwhelming slow consumers
- User now only waits **35ms**

Questions? Complaints? Objections?

Florin Barbuceanu

Senior Solutions Architect @ Amazon Web Services Berlin

florin.gabriel.barbuceanu@gmail.com ← for extra complaints/thoughts