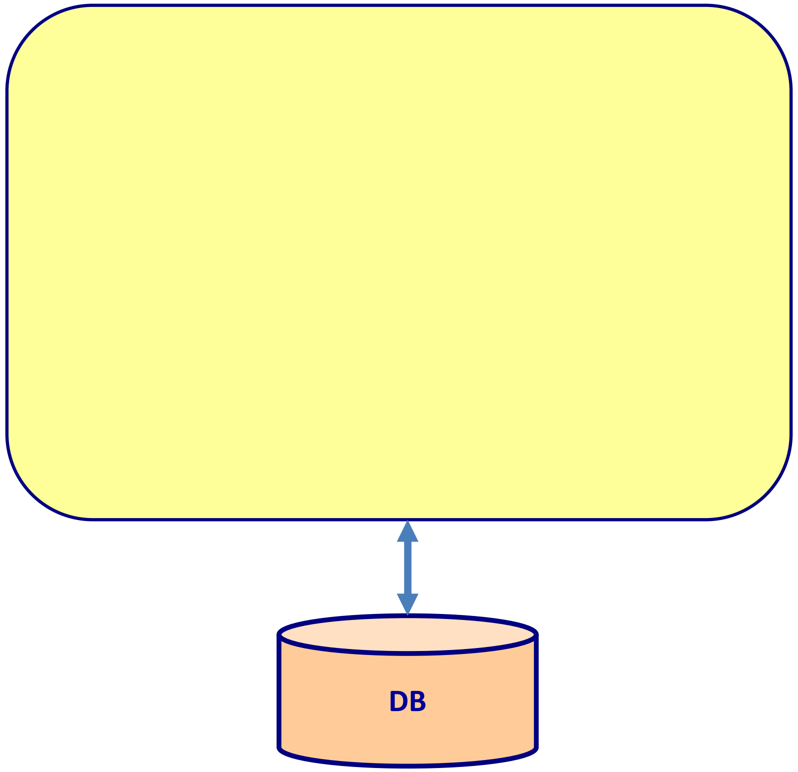
Lesson 7

MICROSERVICES

MONOLITH ARCHITECTURE

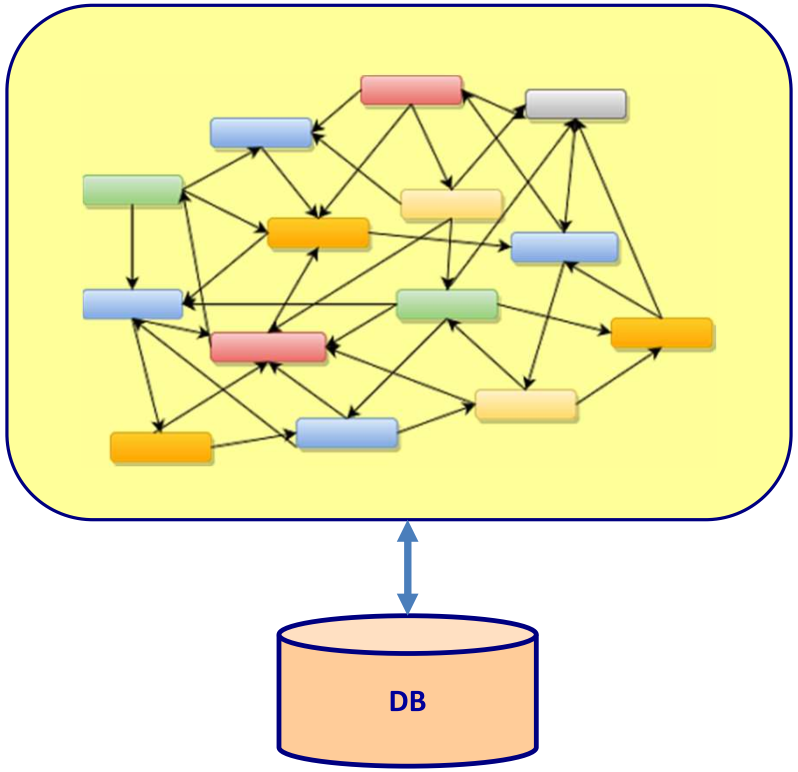
Monolith architecture

- Everything is implemented in one large system



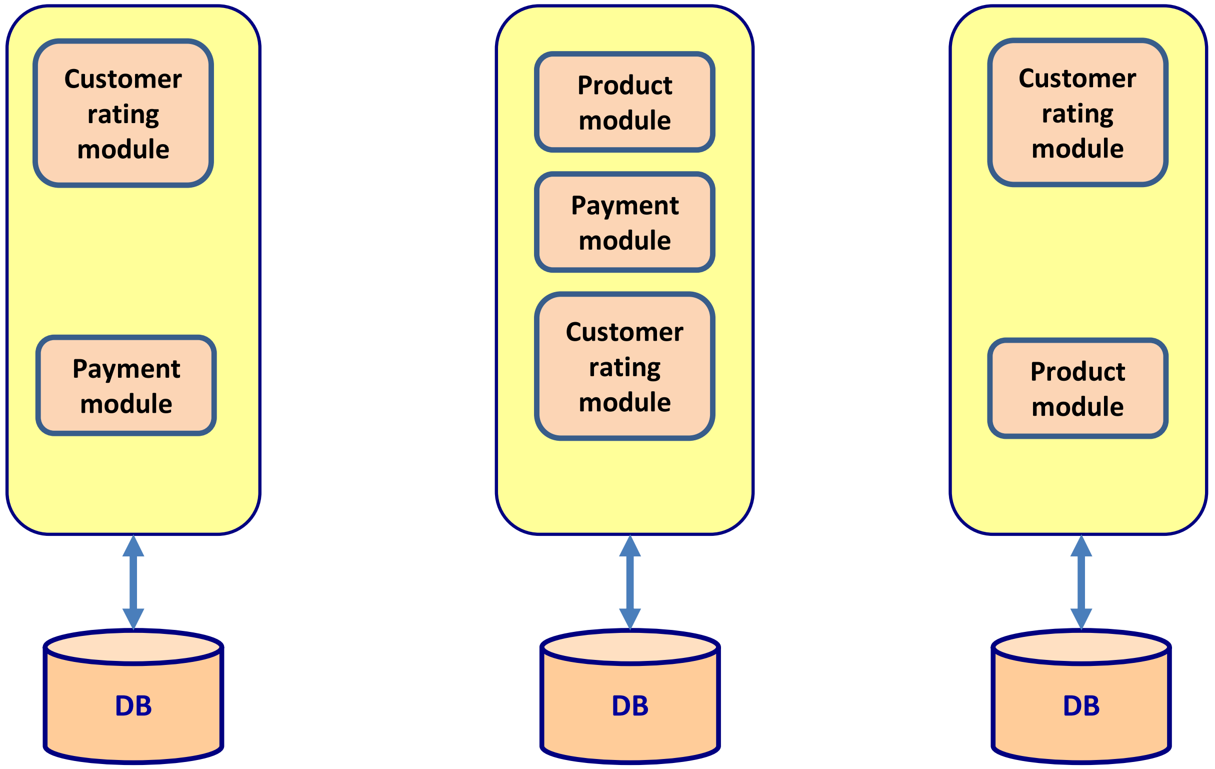
Monolith architecture

- Can evolve in a big ball of mud
  - Large complex system
    - Hard to understand
    - Hard to change



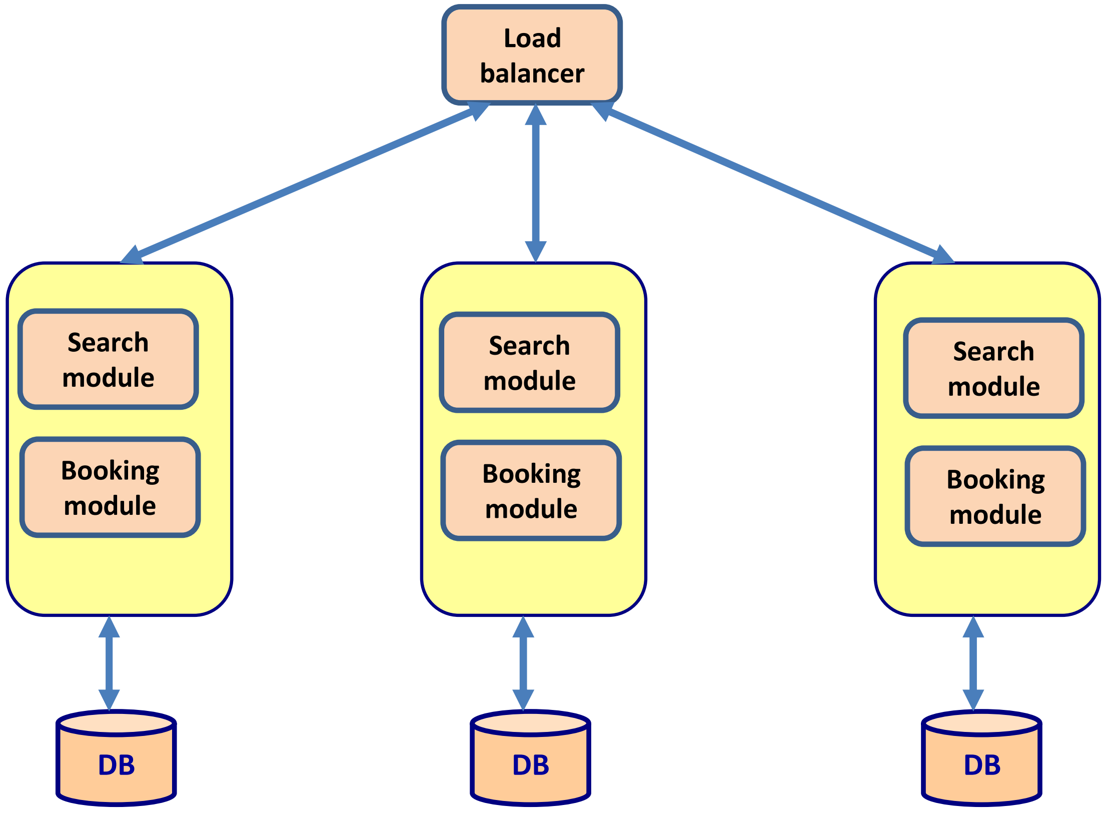
Monolith architecture

- Limited re-use is realized across monolithic applications



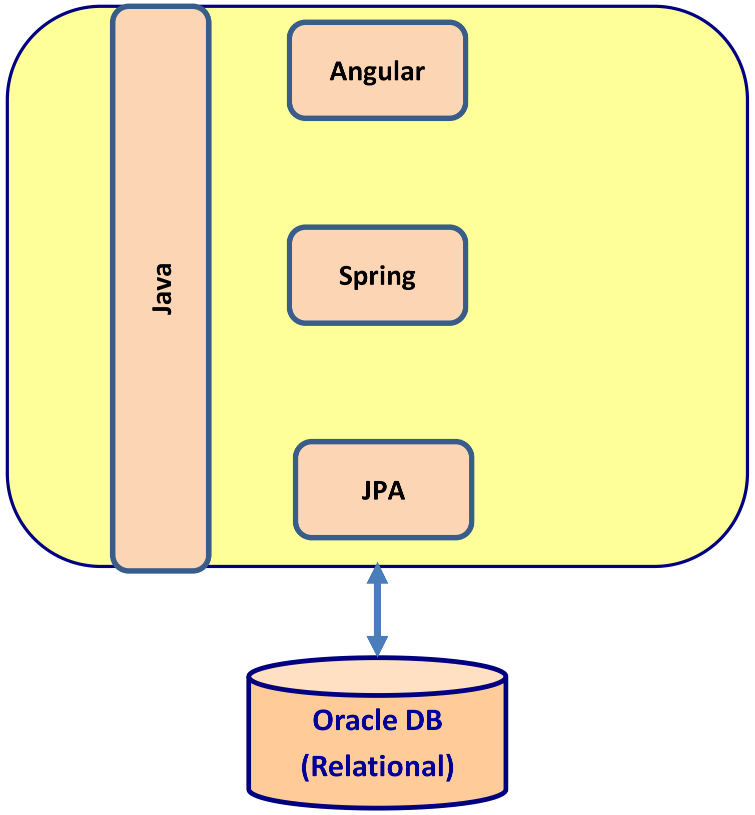
Monolith architecture

- All or nothing scaling
  - Difficult to scale separate parts



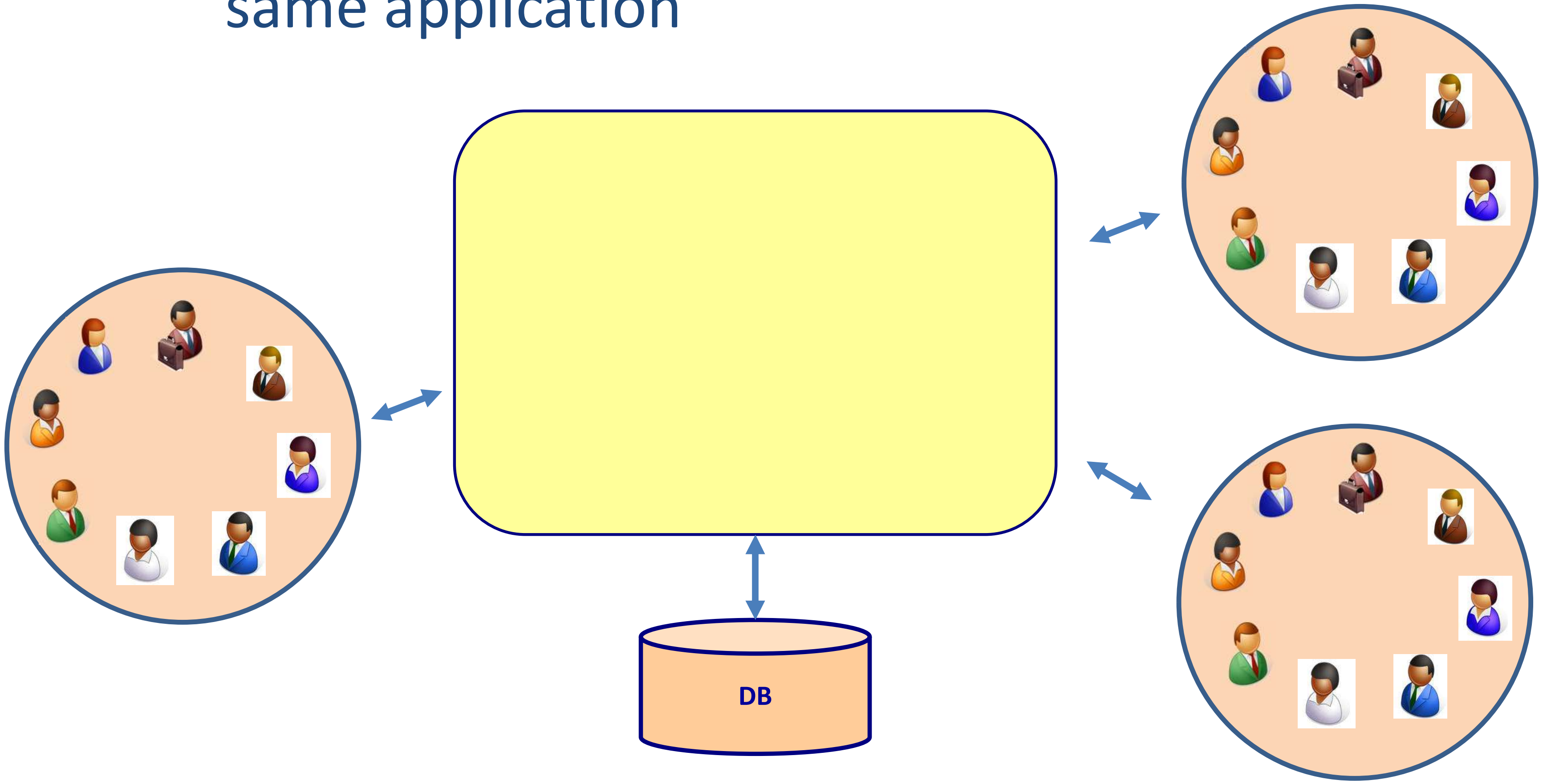
Monolith architecture

- Single development stack
  - Hard to use “the right tool for the job.”



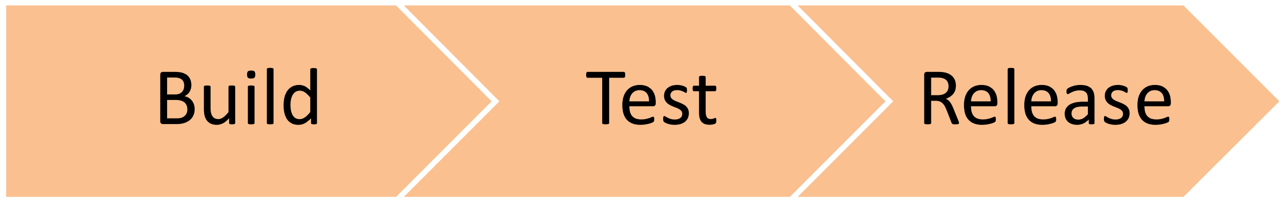
Monolith architecture

- Does no support small agile scrum teams
  - Hard to have different agile teams work on the same application

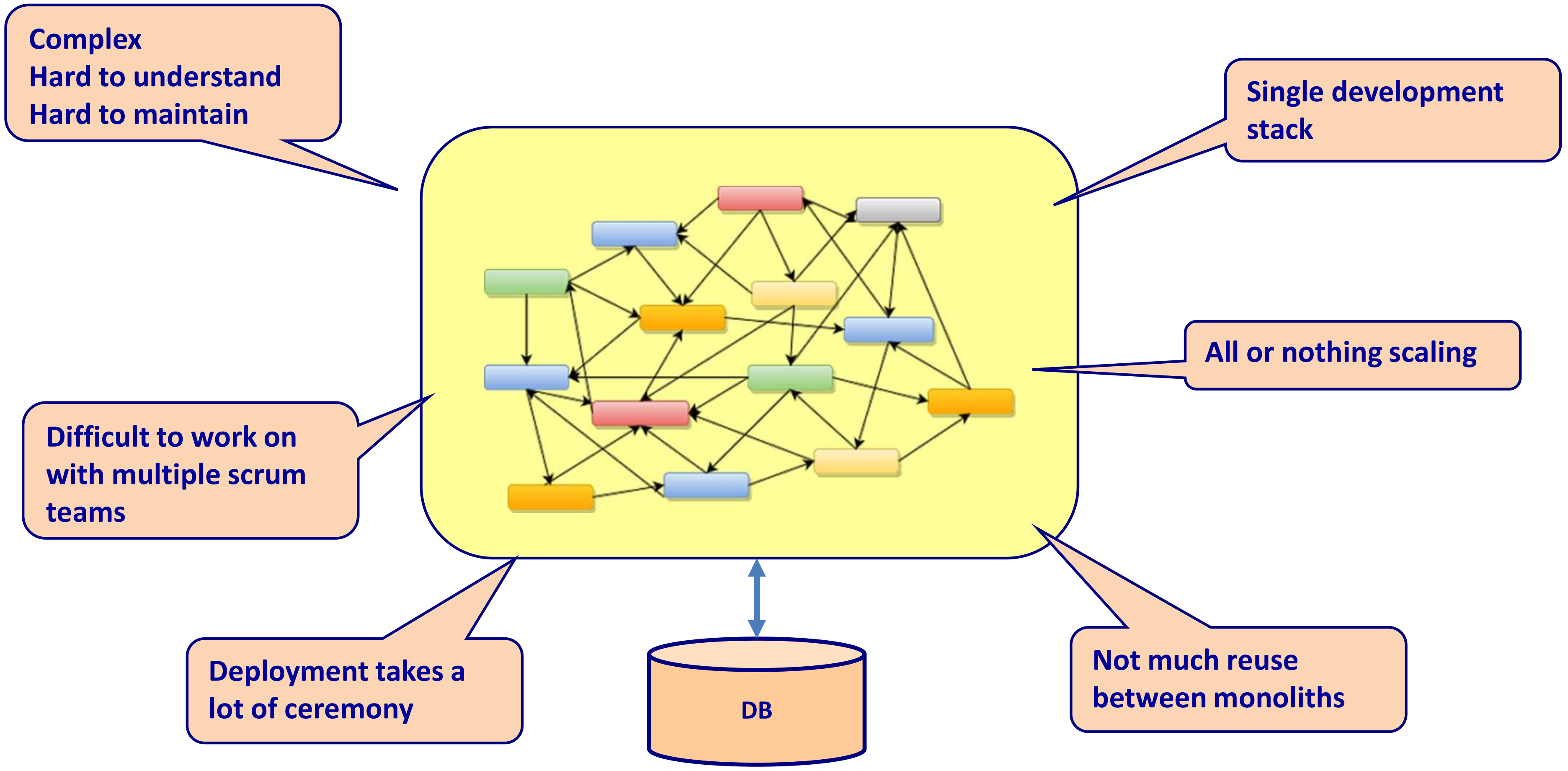


Monolith architecture

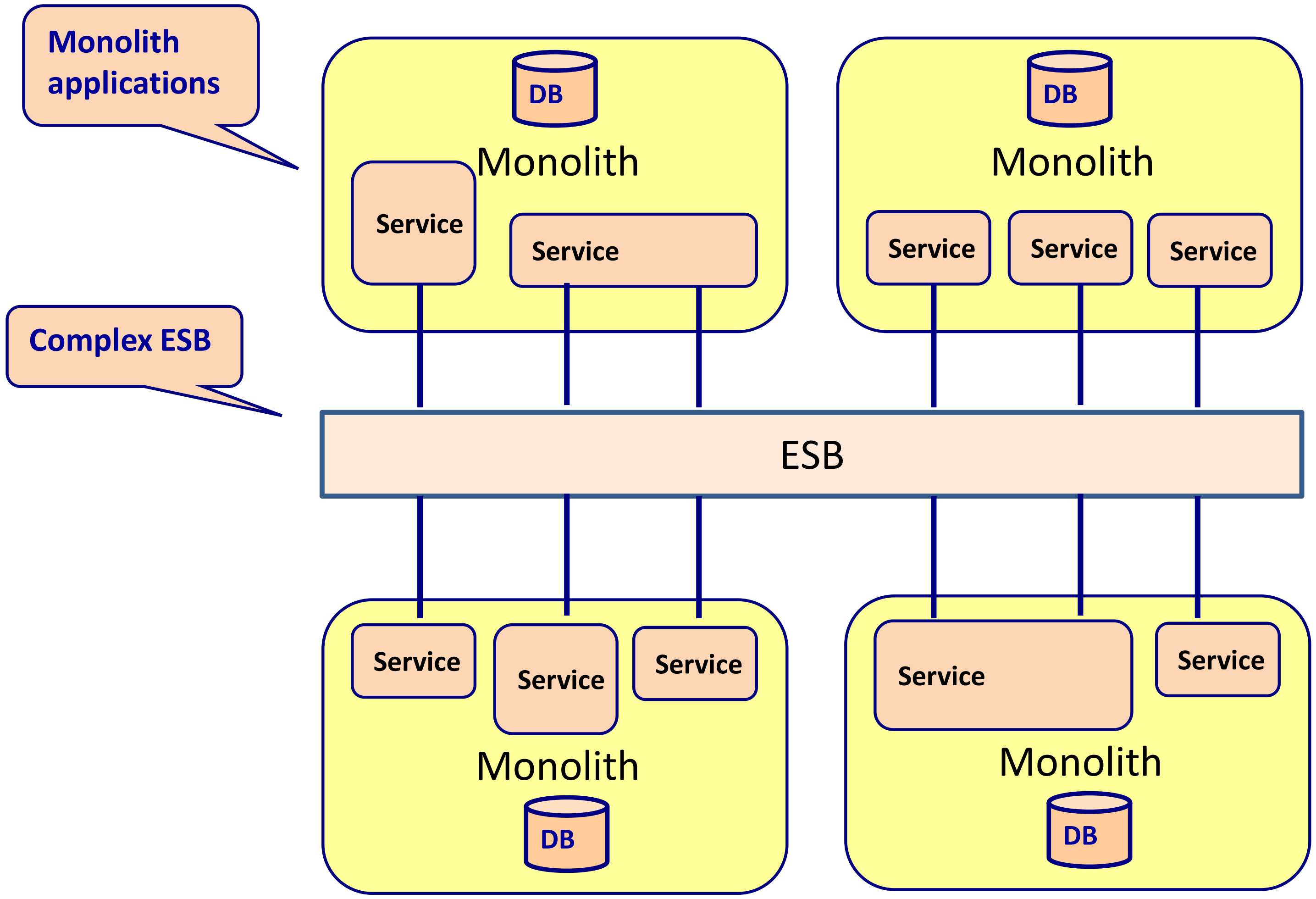
- Deploying a monolith takes a lot of ceremony
  - Every deployment is of high risk
  - I cannot deploy very frequently
  - Long build-test-release cycles



Problems with a monolith architecture


Complex
Hard to understand
Hard to maintain
Single development stack
All or nothing scaling
Difficult to work on with multiple scrum teams
Deployment takes a lot of ceremony
DB
Not much reuse between monoliths

Problems with SOA



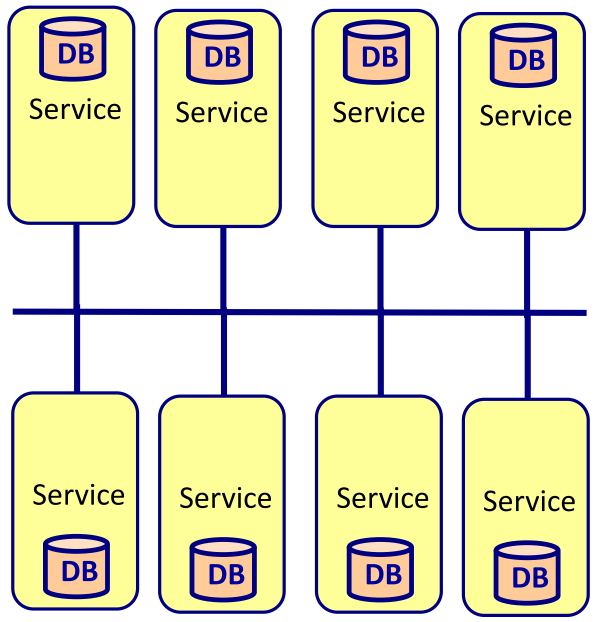
Microservice early adopters

- Netflix
- Uber
- Airbnb
- Orbiz
- eBay
- Amazon
- Twitter
- Nike

Common problem:
How to migrate from a monolith to more scalability, process automation, manageability,…

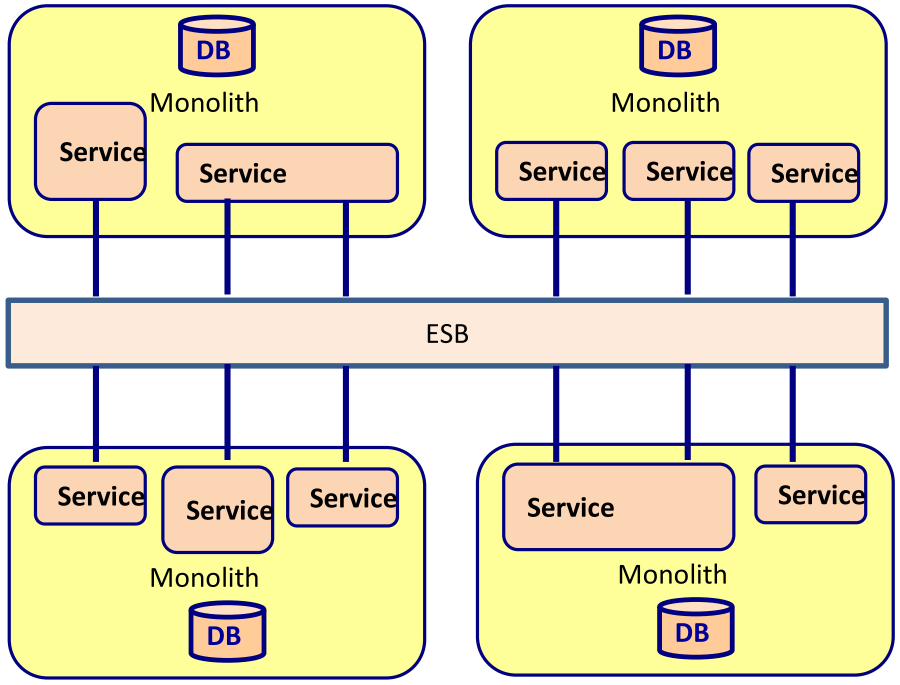
Microservices

- Small independent services
  - Simple and lightweight
  - Runs in an independent process
  - Language agnostic
  - Decoupled

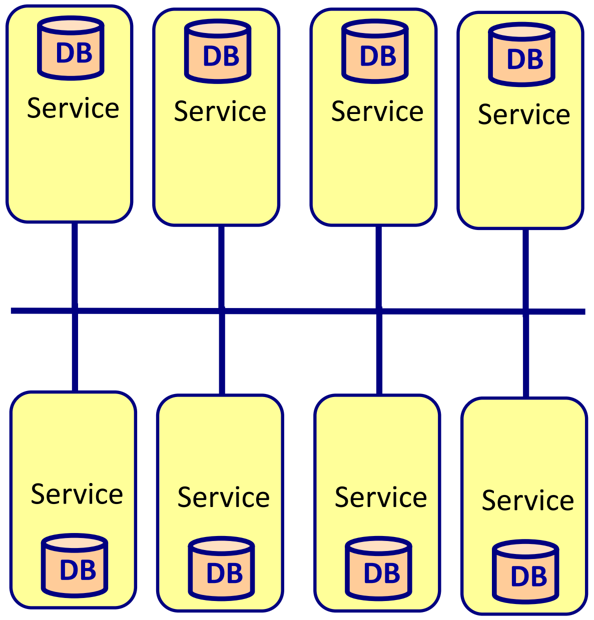


SOA vs Microservice

SOA



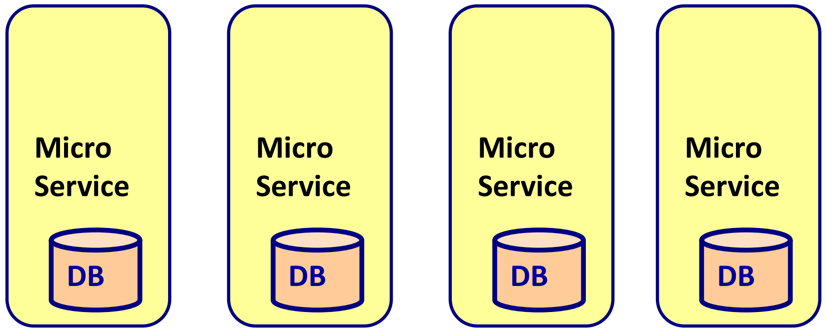
Microservice



CHARACTERISTICS OF A MICROSERVICE

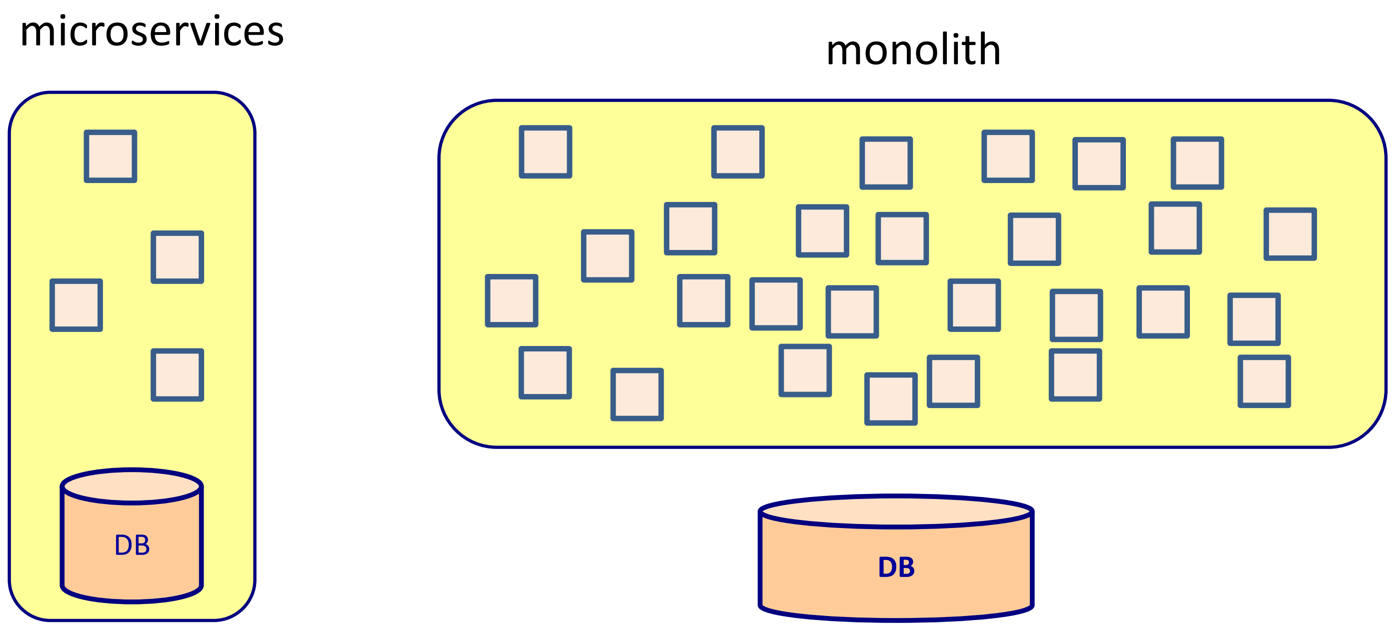
Microservices

- Small independent services
  - Simple and lightweight
  - Runs in an independent process
  - Technology agnostic
  - Decoupled

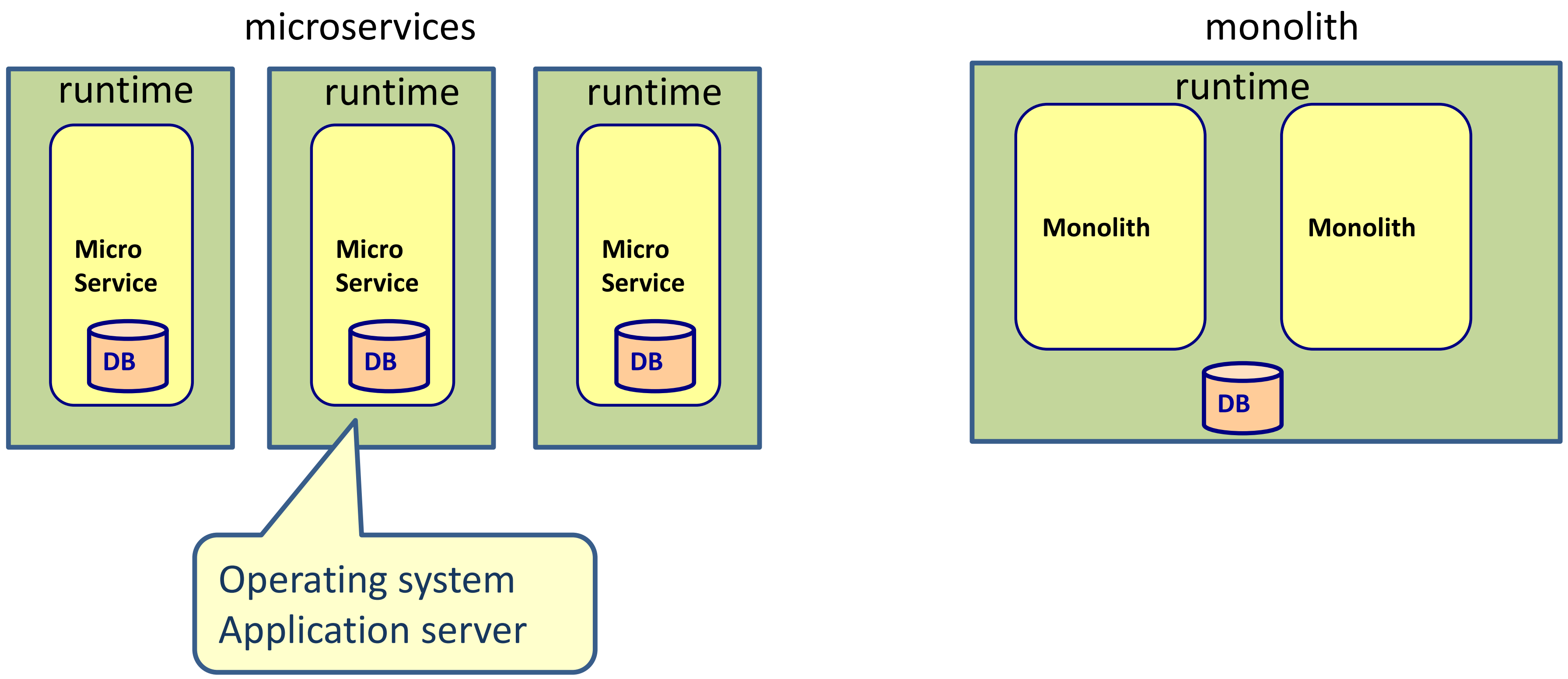


Simple and lightweight

- Small and simple
- Can be build and maintained by 1 agile team



Runs in an independent process



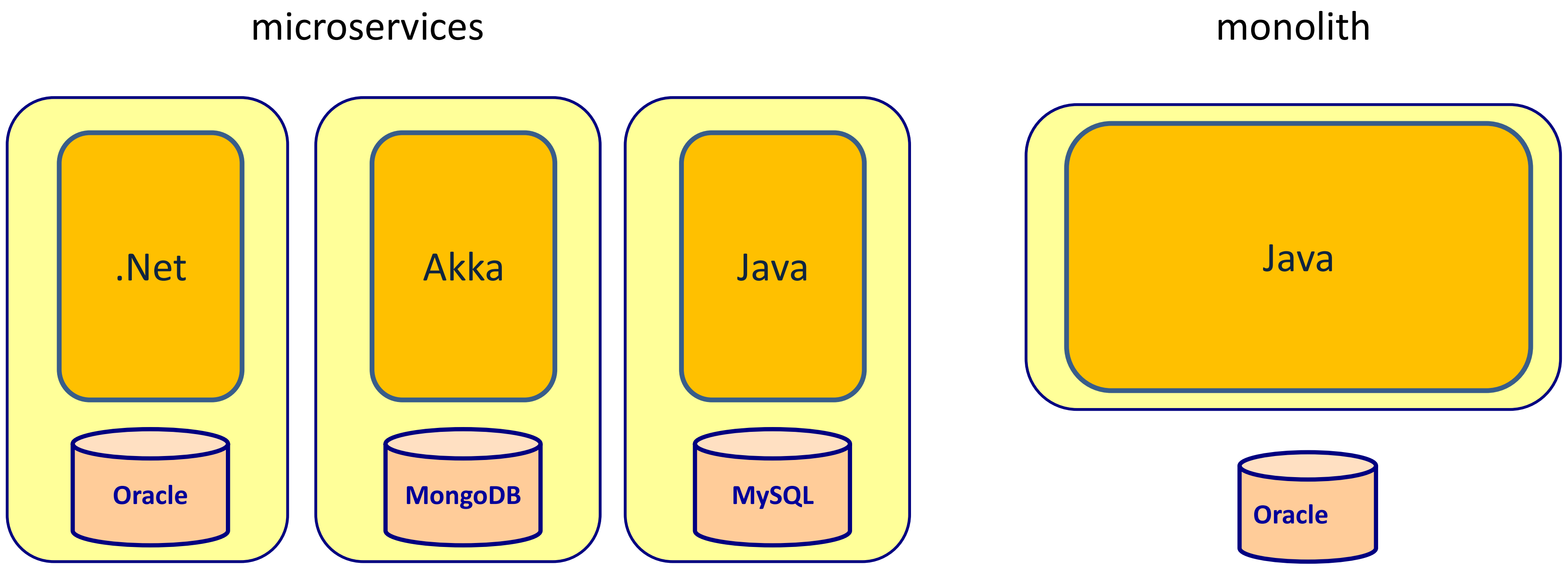
Advantages

- Runtime can be small
  - Only add what you need
- Runtime can be optimized
- Runtime can start and stop fast
- If runtime goes down, other services will still run

Disadvantages

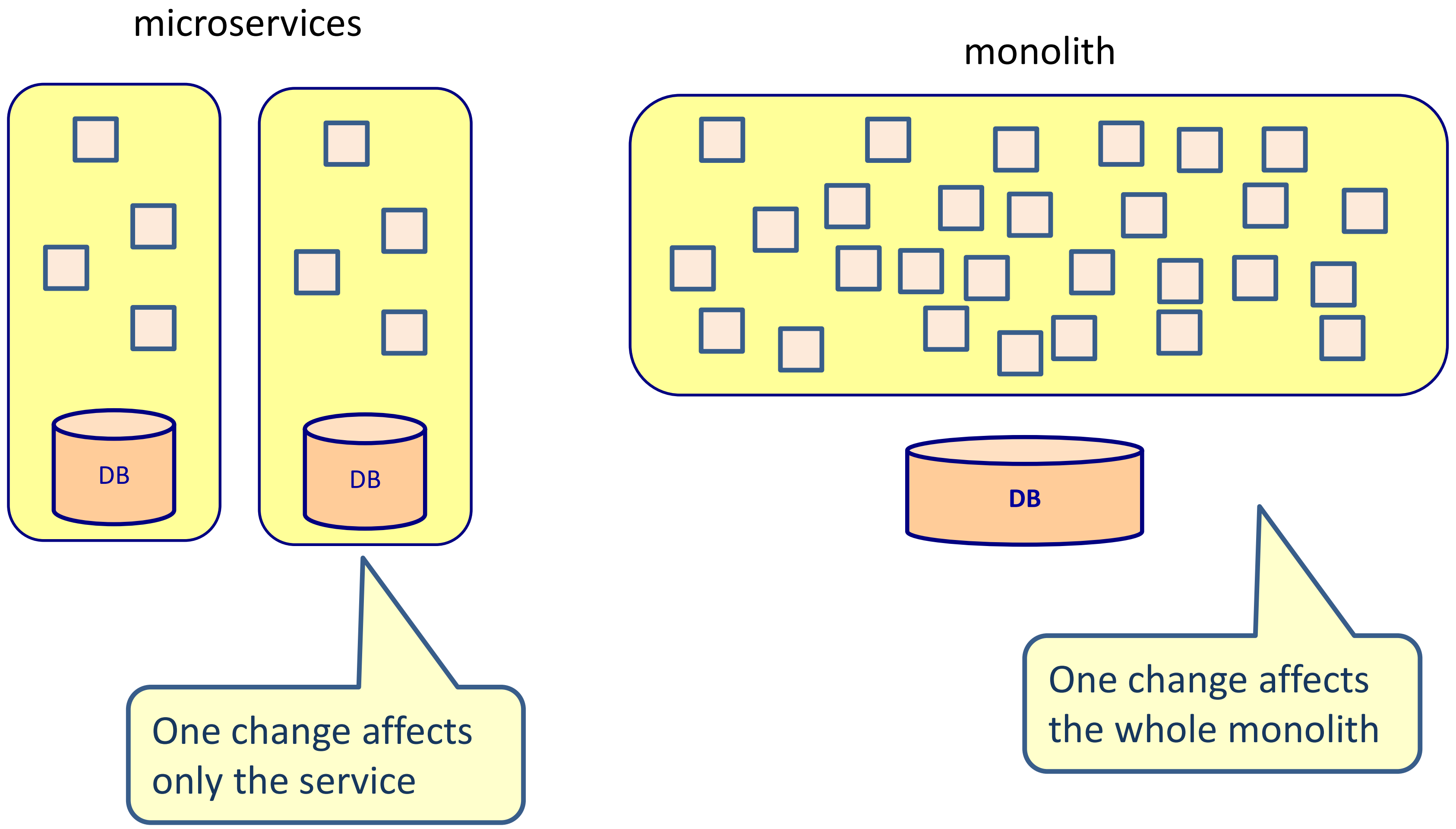
- We need to manage many runtimes

Technology agnostic

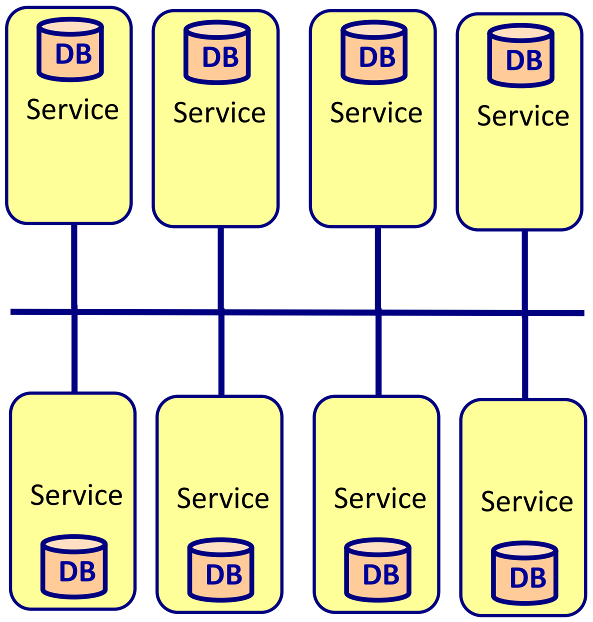


- Use the architecture and technologies that fits the best for this particular microservice

Decoupled


microservices
DB
DB
One change affects only the service
monolith
DB
One change affects the whole monolith

Microservice architecture



Simple microservices
Simpler to understand
Simpler to maintain

Every microservice has its own development stack

Microservices can be individually scaled

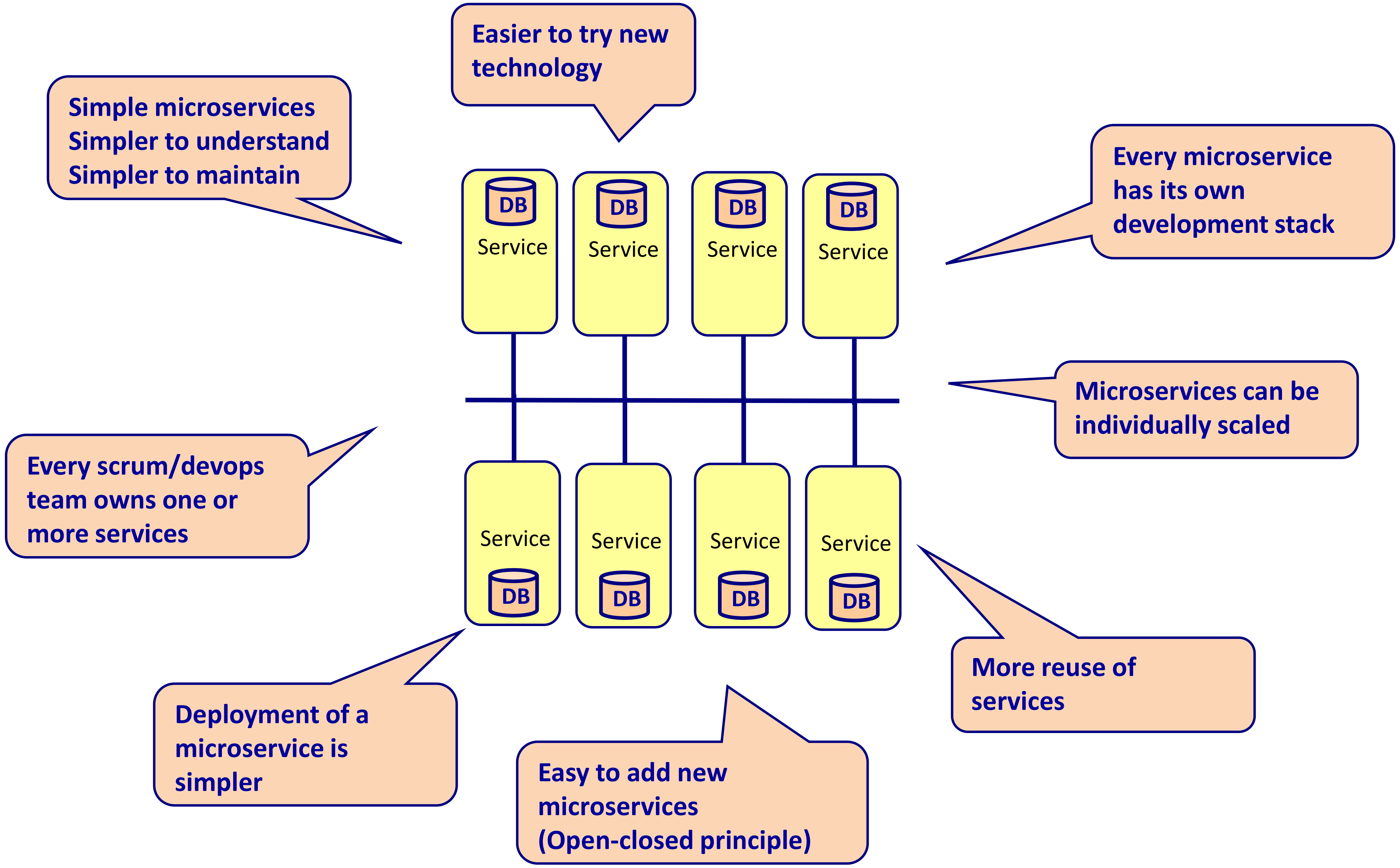
Every scrum teams owns one or more services

Deployment of a microservice is simpler

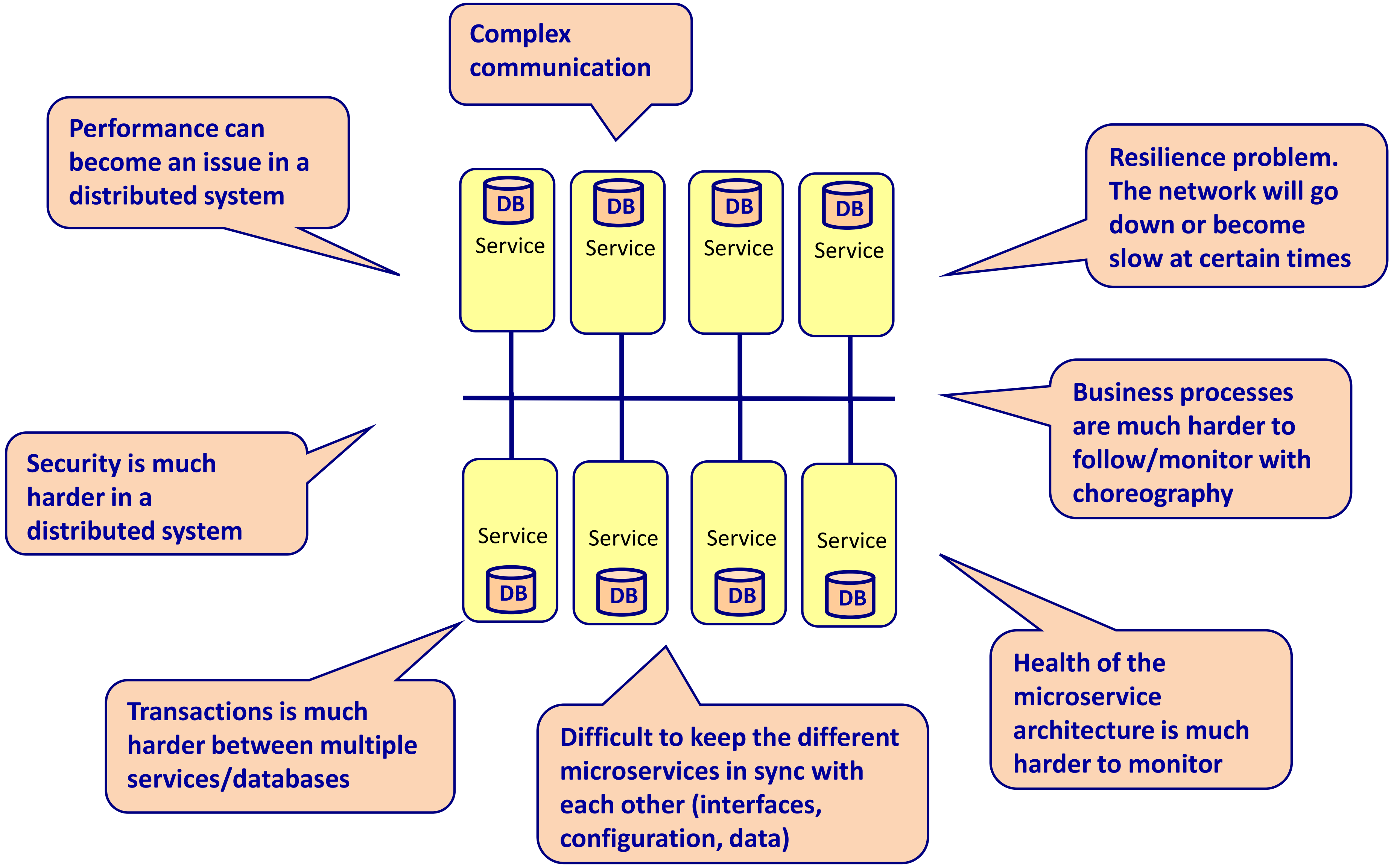
More reuse of services

ADVANTAGES AND DISADVANTAGES OF A MICROSERVICE ARCHITECTURE

Advantages


Easier to try new technology
Simple microservices Simpler to understand Simpler to maintain
Every microservice has its own development stack
Microservices can be individually scaled
Every scrum/devops team owns one or more services
More reuse of services
Deployment of a microservice is simpler
Easy to add new microservices (Open-closed principle)
DB
Service

Disadvantages

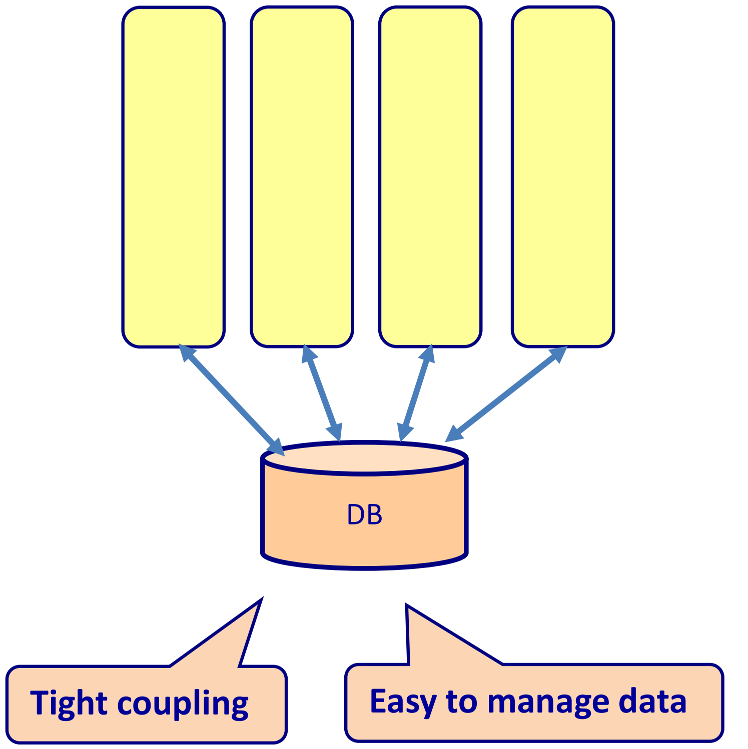

Complex communication
Performance can become an issue in a distributed system
Resilience problem. The network will go down or become slow at certain times
Business processes are much harder to follow/monitor with choreography
Security is much harder in a distributed system
Transactions is much harder between multiple services/databases
Difficult to keep the different microservices in sync with each other (interfaces, configuration, data)
Health of the microservice architecture is much harder to monitor
DB
Service

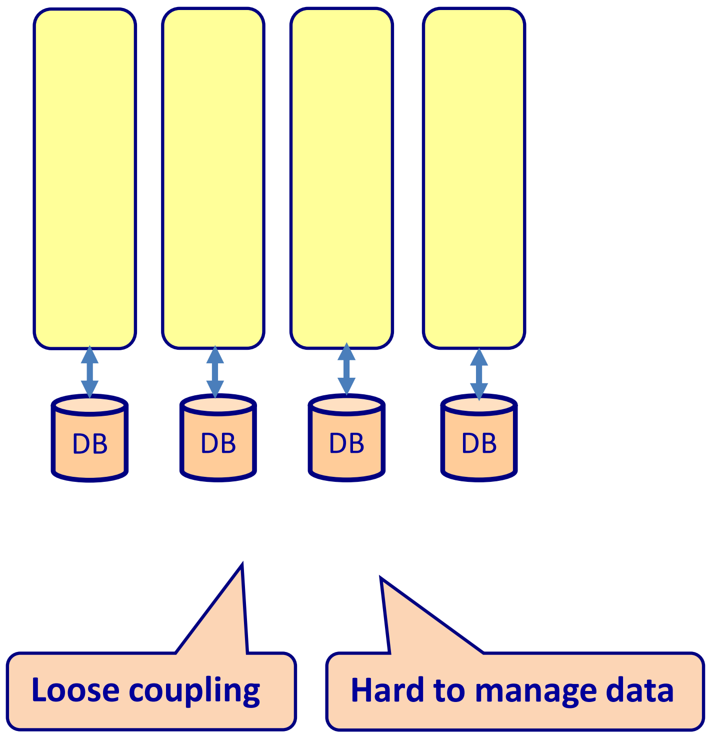
Challenges of a microservice architecture

Challenge	Solution
Complex communication	
Performance	
Resilience	
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	
Follow/monitor business processes	

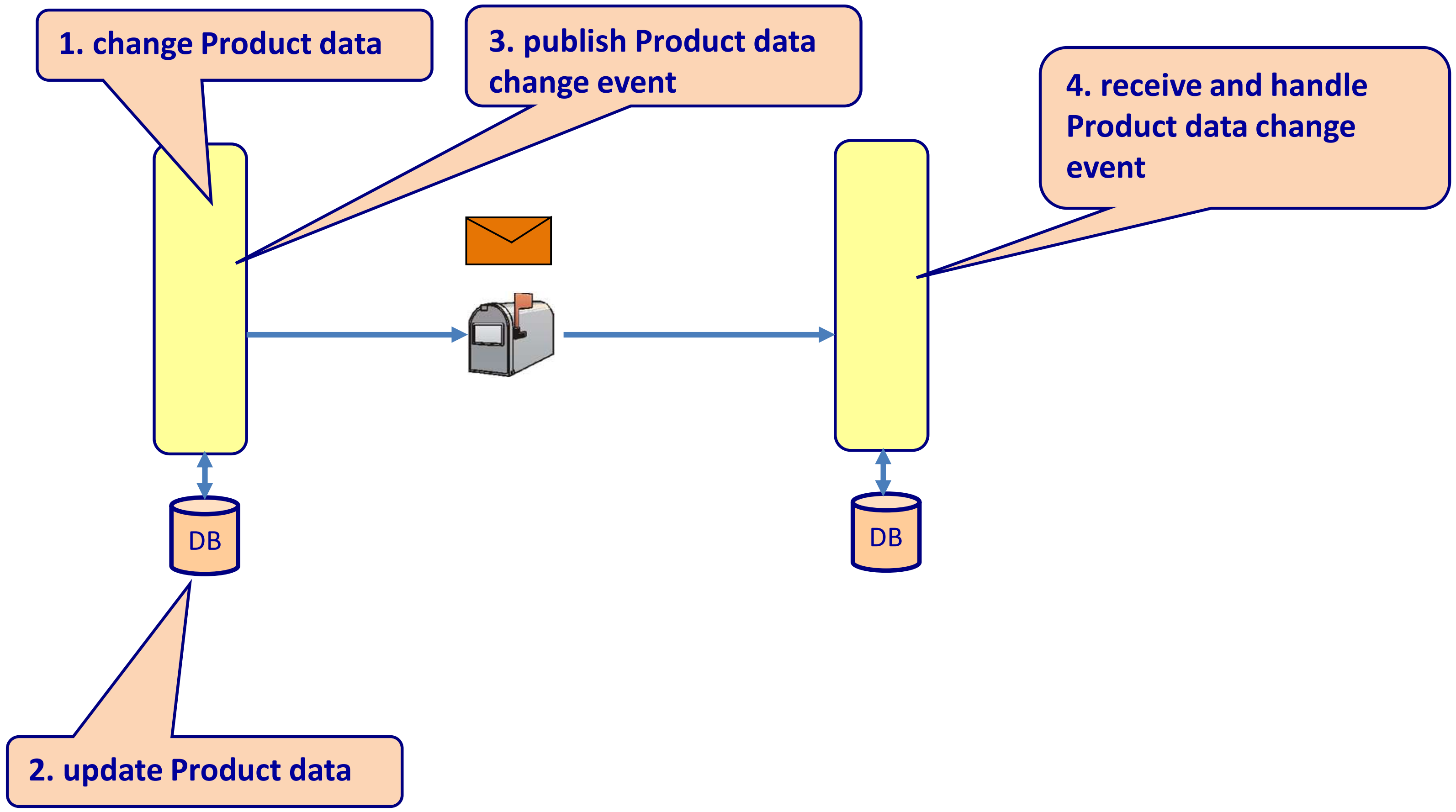
MICROSERVICE AND DATABASES

Every service manages its own data



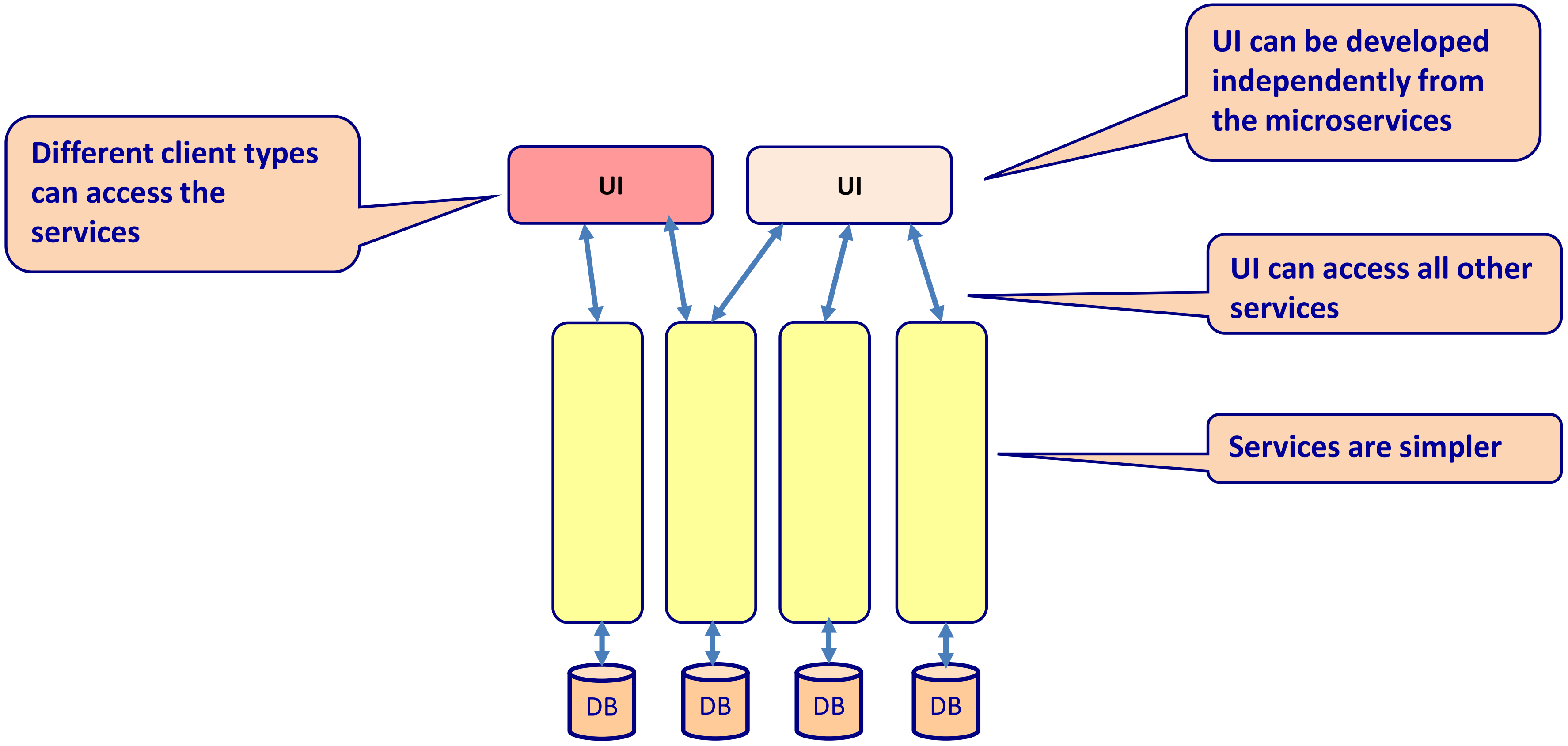


Data consistency


1. change Product data
3. publish Product data change event
4. receive and handle Product data change event
DB
DB
2. update Product data

UI AND MICROSERVICE

Split front-end and back-end


Different client types can access the services
UI
UI
UI can be developed independently from the microservices
UI can access all other services
Services are simpler
DB
DB
DB
DB

MICROSERVICE BOUNDARIES

Appropriate boundaries

- DDD bounded context
  - Isolated domains that are closely aligned with business capabilities
- Autonomous functions
  - Accept input, perform its logic and return a result
    - Encryption engine
    - Notification engine
    - Delivery service that accept an order and informs a trucking service

Appropriate boundaries

- Size of deployable unit
  - Manageable size
- Most appropriate function or subdomain
  - What is the most useful component to detach from the monolith?
  - Hotel booking system: 60-70% are search request
    - Move out the search function
- Polyglot architecture
  - Functionality that needs different architecture
    - Booking service needs transactions
    - Search does not need transactions

Appropriate boundaries

- Selective scaling
  - Functionality that needs different scaling
    - Booking service needs low scaling capabilities
    - Search needs high scaling capabilities
- Small agile teams
  - Specialist teams that work on their expertise
- Single responsibility

Appropriate boundaries

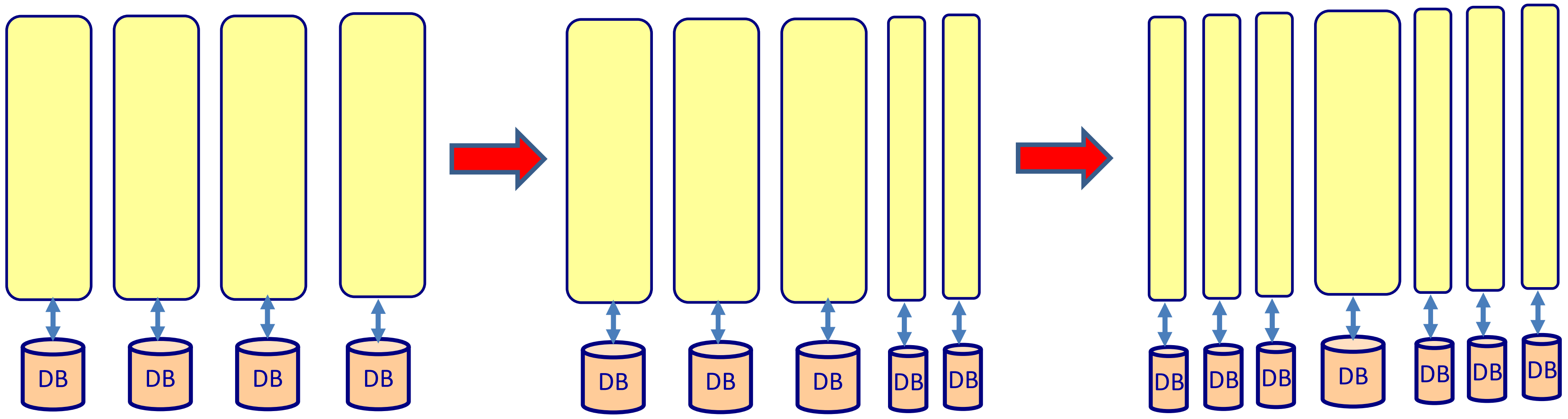
- Replicability or changeability
  - The microservice is easy detachable from the overall system
  - What functionality might evolve in the future?
- Coupling and cohesion
  - Avoid chatty services
  - Too many synchronous request
  - Transaction boundaries within one service

Appropriate boundaries

- DDD bounded context
- Autonomous functions
- Size of deployable unit
- Most appropriate function or subdomain
- Polyglot architecture
- Selective scaling
- Small agile teams
- Single responsibility
- Replicability or changeability
- Coupling and cohesion

Microservice boundaries

- Start with a few services and then evolve to more services



Domains

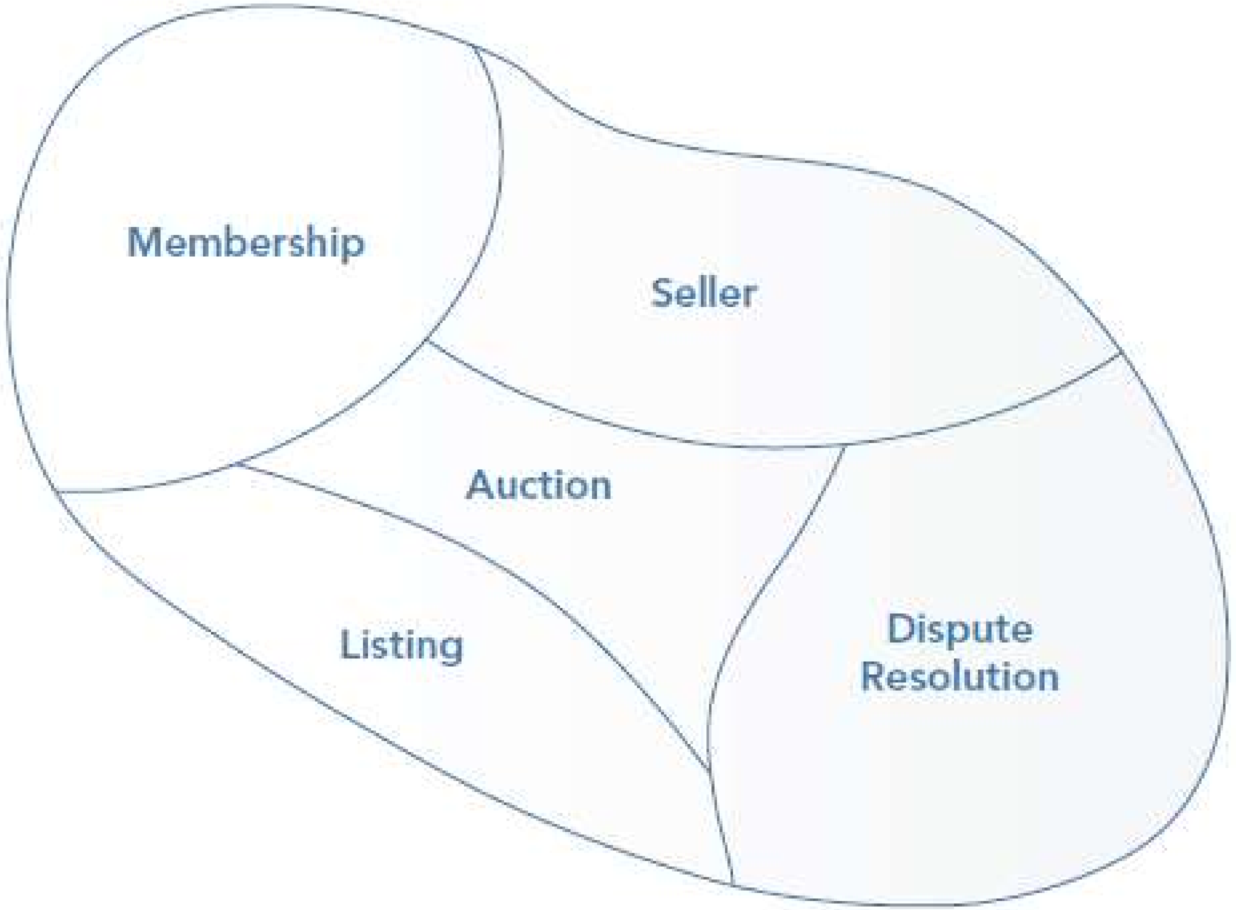
- Core subdomain
  - This is the reason you are writing the software.
- Supporting subdomain
  - Supports the core domain
- Generic subdomain
  - Very generic functionality
    - Email sending service
    - Creating reports service

Distilling the domain

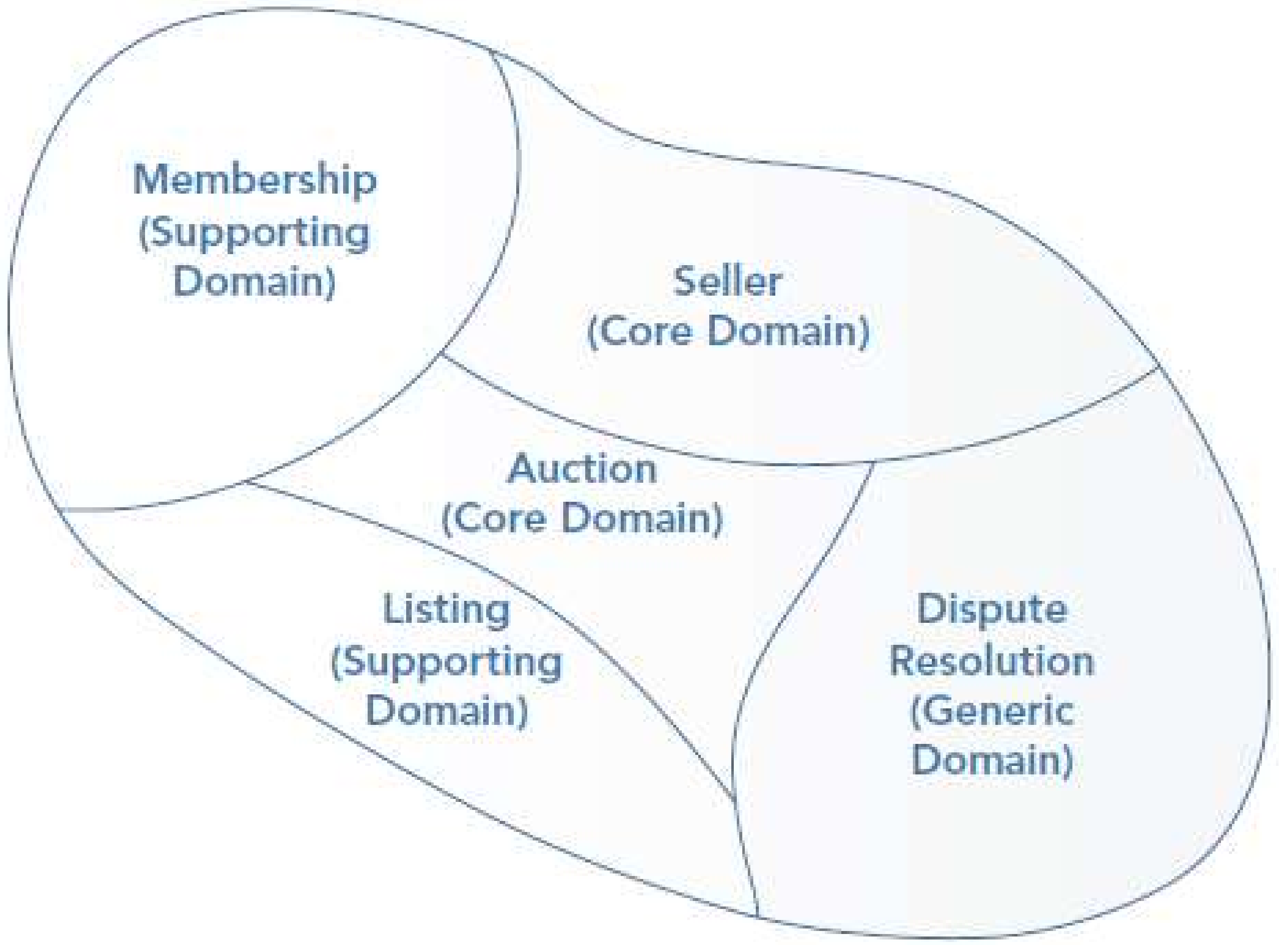
- The large domain of online auction

Online Auction

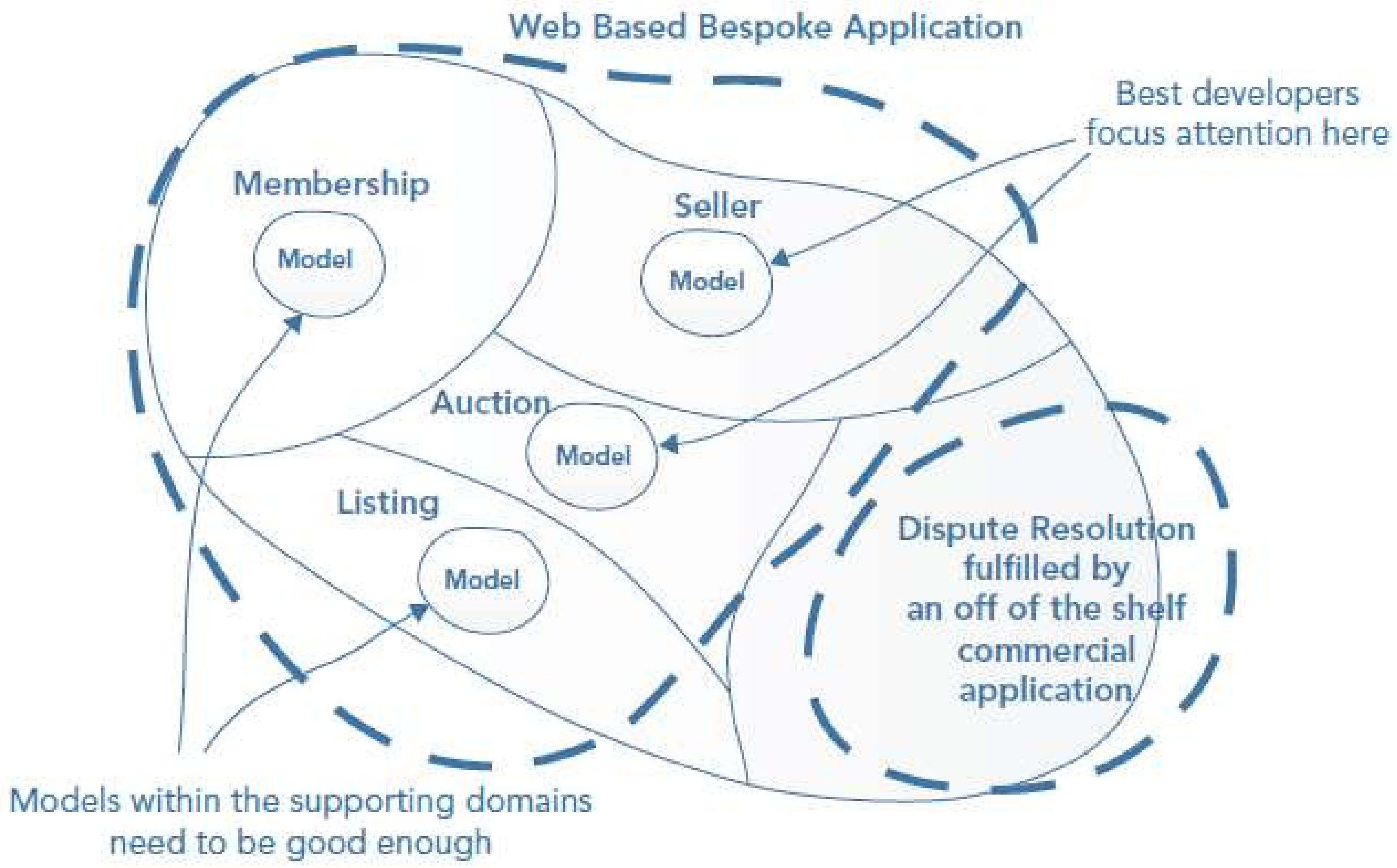
Find the subdomains



Identify the core domain



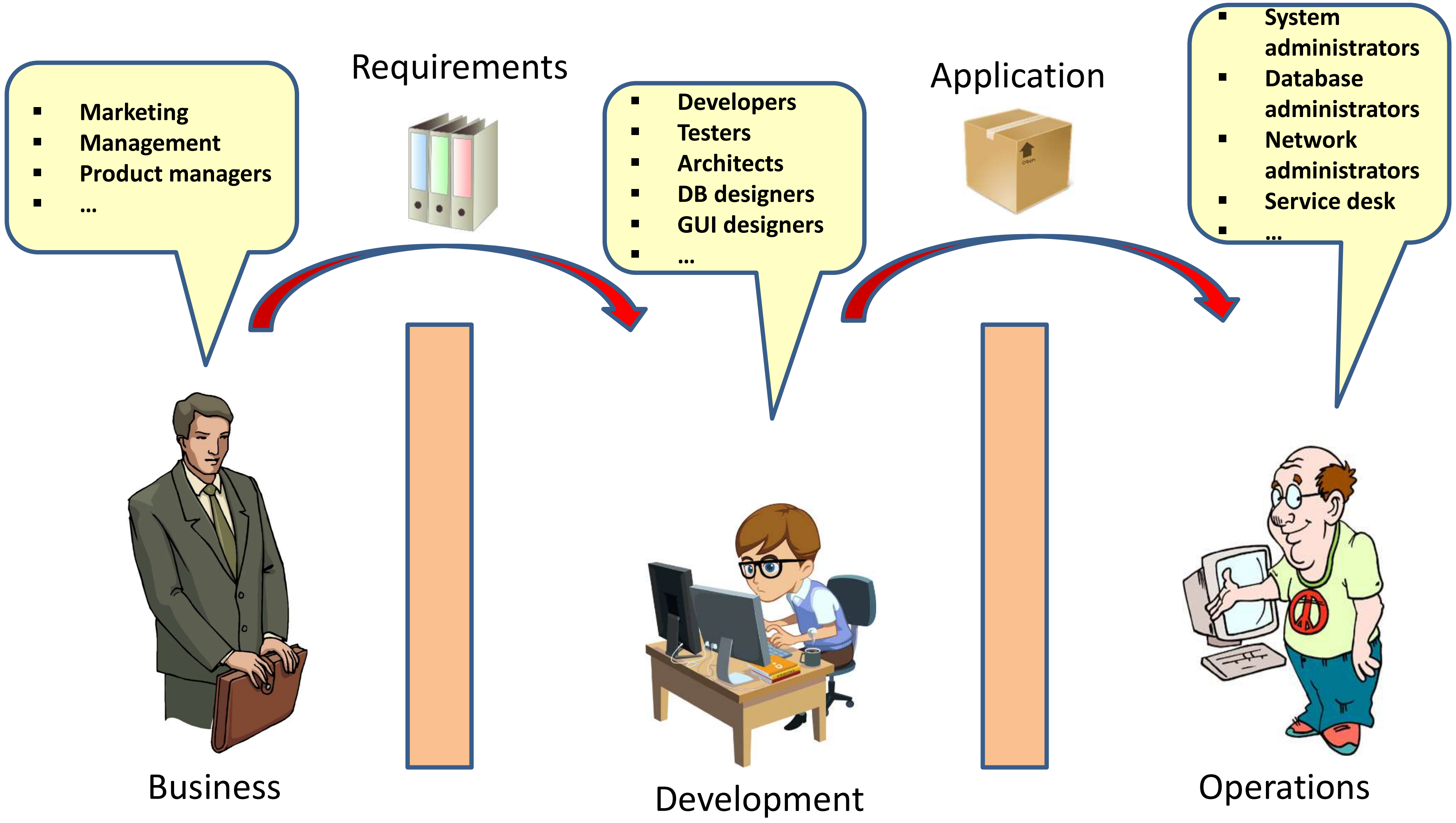
Subdomains shape the solution



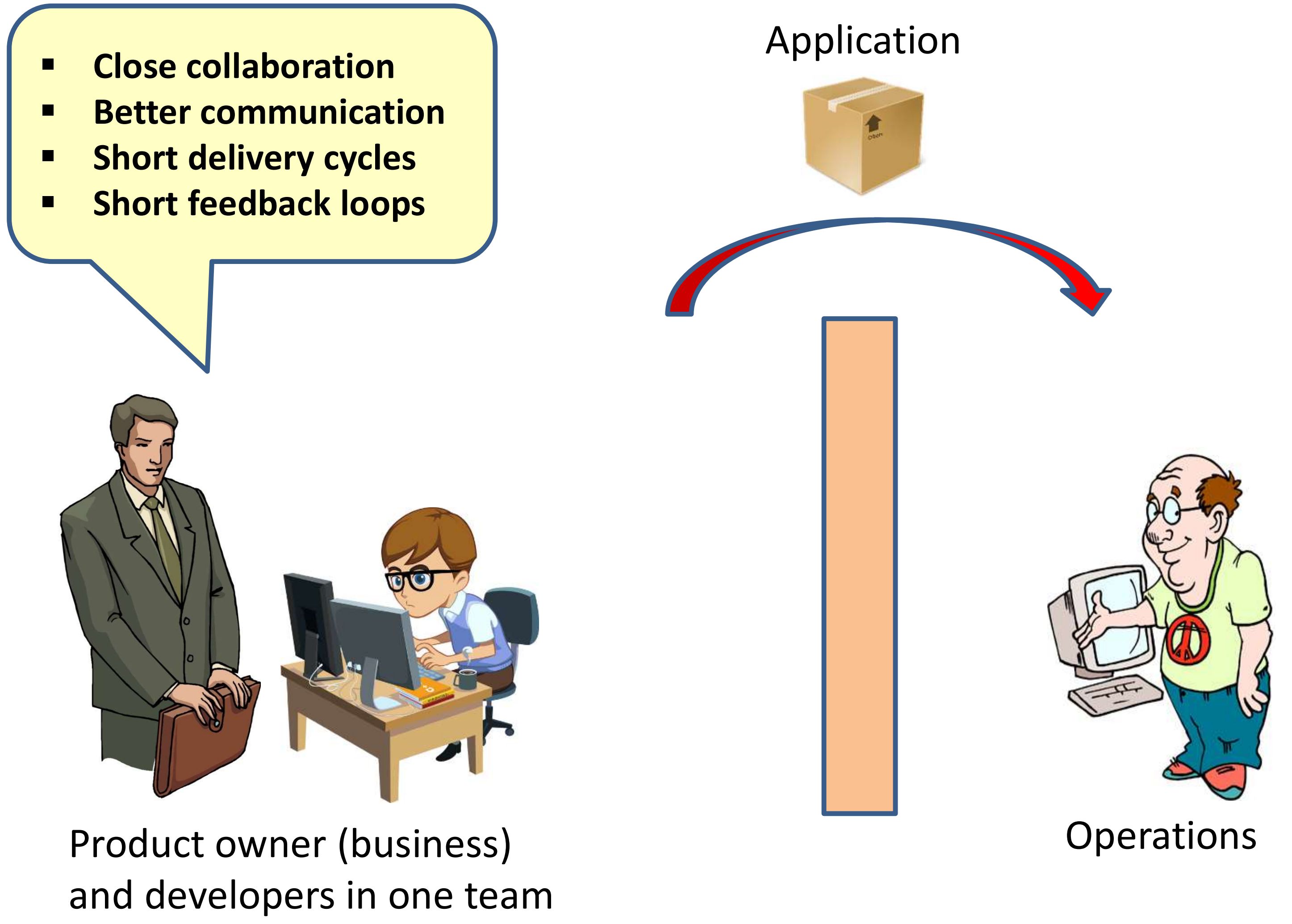
Web Based Bespoke Application
Best developers focus attention here
Membership
Model
Seller
Model
Auction
Model
Listing
Model
Dispute Resolution fulfilled by an off of the shelf commercial application
Models within the supporting domains need to be good enough

MICROSERVICES IN THE ORGANIZATION

Traditional software development



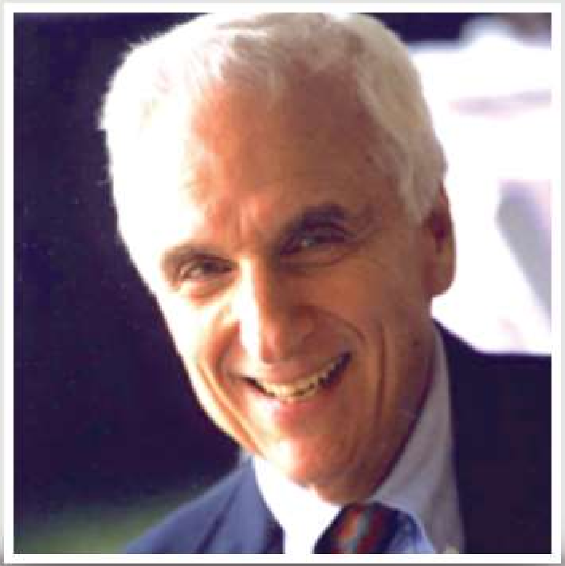
Agile software development: Scrum



DevOps

- Close collaboration between developers and operations
- Streamlines the delivery process of software from business requirements to production
- Better communication
- Identical development and production environment
- Shared tools
  - Automate everything
  - Monitor everything

Conways law



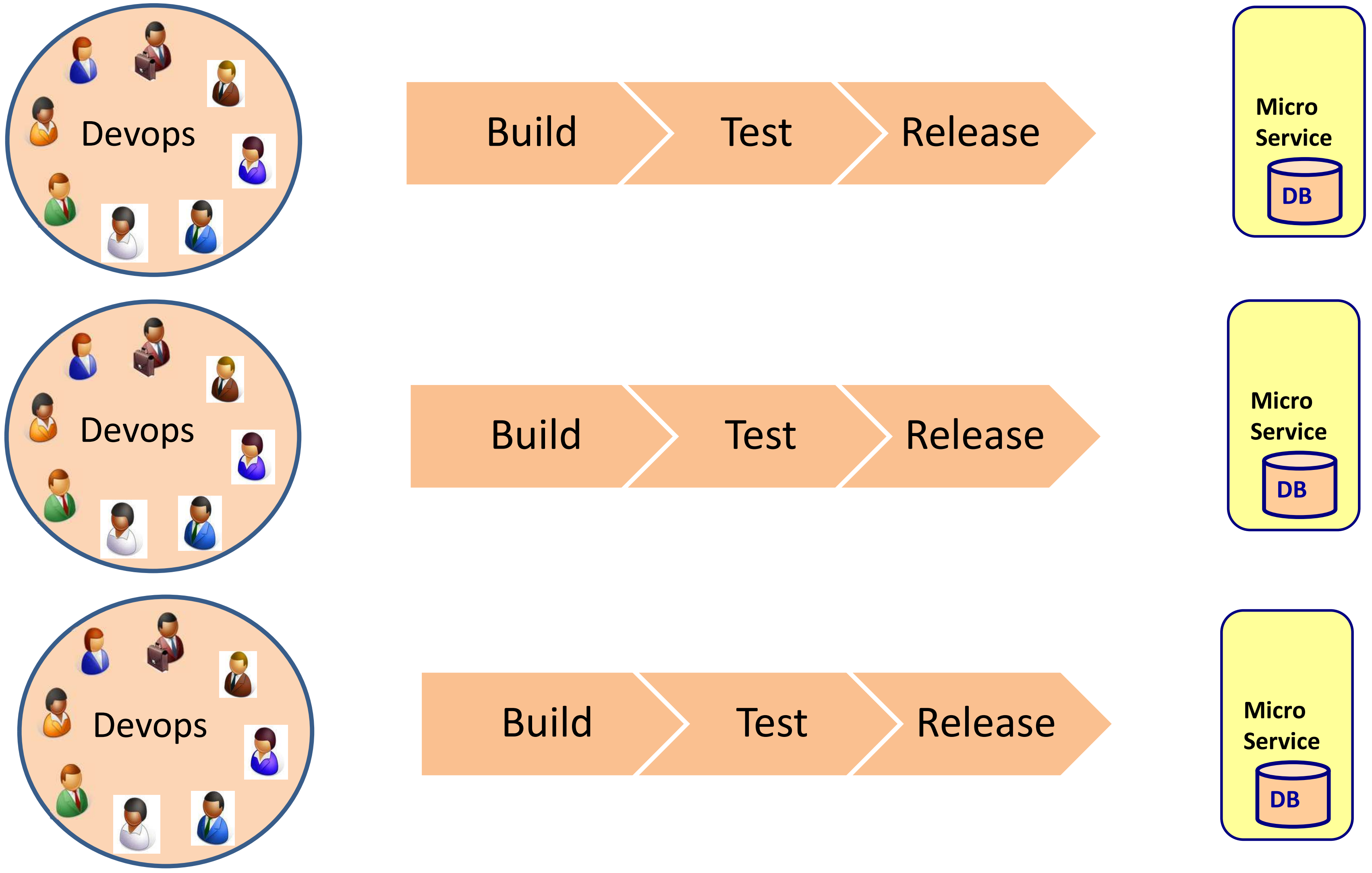
"If you have four groups working on a compiler, you'll get a 4-pass compiler"

—Eric S Raymond

"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations "

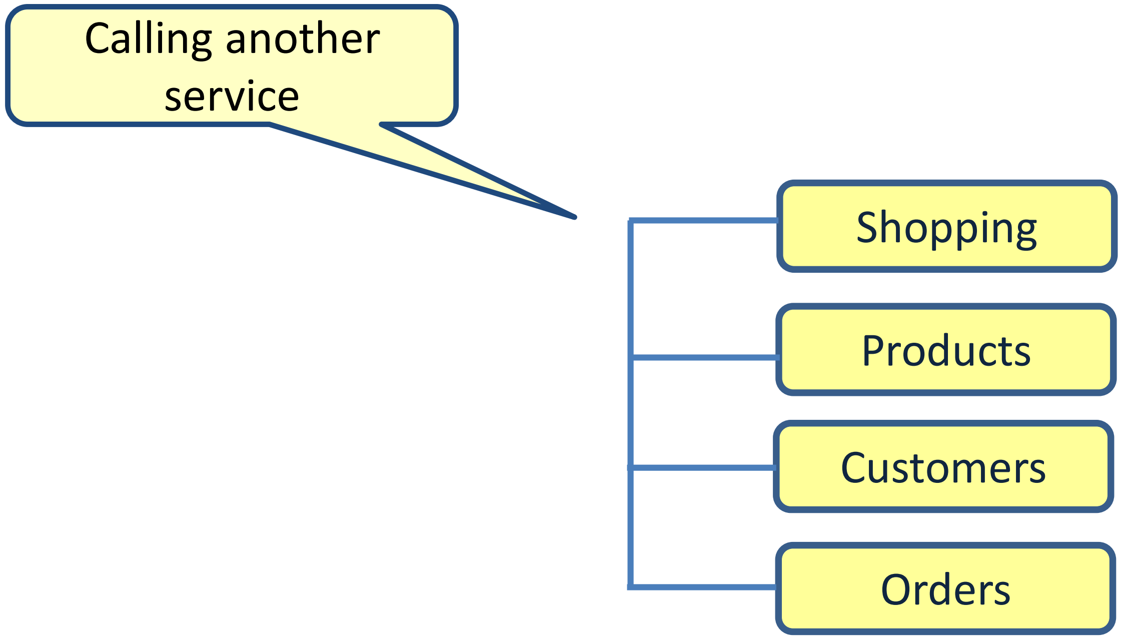
—Melvin Conway

Microservice organization

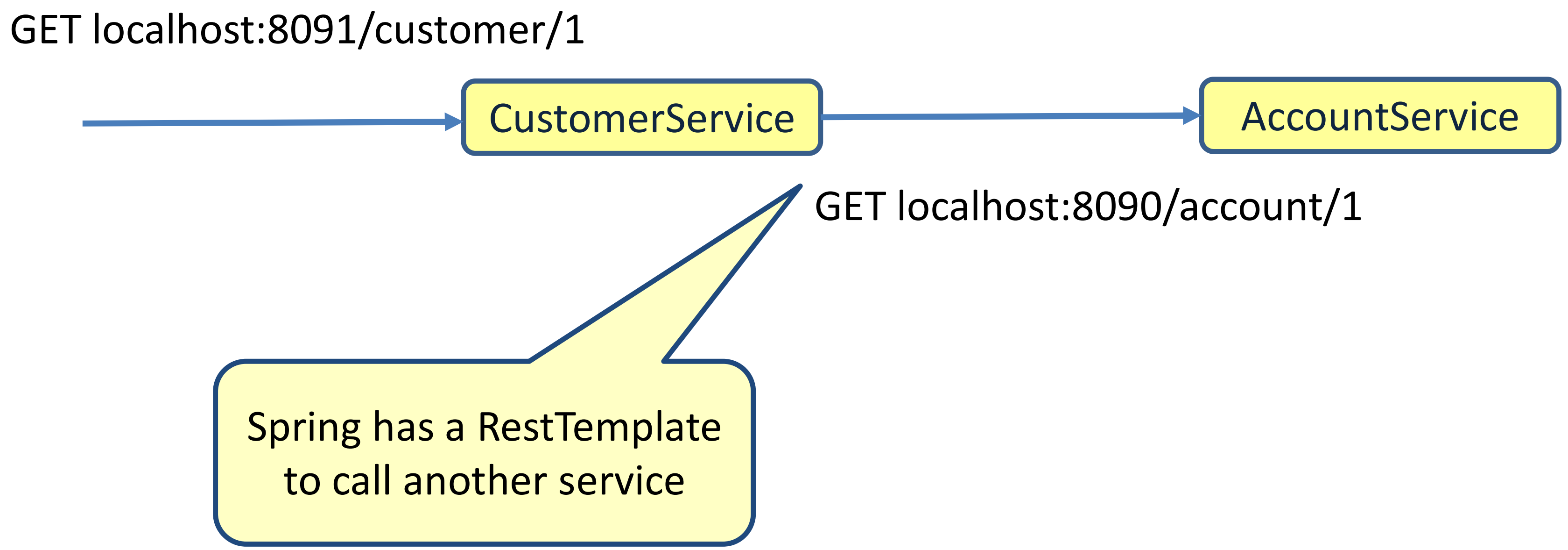


CALLING ANOTHER MICROSERVICE: FEIGN

Implementing microservices



Calling another service


GET localhost:8091/customer/1
CustomerService
AccountService
GET localhost:8090/account/1
Spring has a RestTemplate to call another service

RestTemplate

```java
@Component
public class RestClient {
  @Autowired
  private RestOperations  restTemplate;

  public void callRestServer(){
        Greeting greeting =
           restTemplate.getForObject("http://localhost:8080/greeting", Greeting.class);
        System.out.println("Receiving message:"+greeting.getContent());
  }
}
```

RestTemplate has to be configured.
Developer has to know REST details

```java
@Configuration
public class AppConfig {
   @Bean
   RestTemplate restTemplate(){
        return new RestTemplate();
   }
}
```

Feign

- Declarative HTTP client
  - Simplify the HTTP client
- You only need to declare and annotate the interface

AccountService

```java
@RestController
public class AccountController {
  @RequestMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    return new Account("1234", "1000.00");
  }
}
```

```java
public class Account {
  private String accountNumber;
  private String balance;
  ...
}
```

application.yml

```yaml
server:
  port: 8090
```

```java
@SpringBootApplication
public class AccountServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(AccountServiceApplication.class, args);
  }
}
```

Properties files and yml files

application.properties

```
Mapping single properties
myapp.mail.to=frank@hotmail.com
myapp.mail.host=mail.example.com
myapp.mail.port=250

#Mapping list or array
myapp.mail.cc=mike@gmail.com,david@gmail.com
myapp.mail.bcc=john@hotmail.com,admin@acme.com

#Mapping nested POJO class
myapp.mail.credential.user-name=john1234
myapp.mail.credential.password=xyz@1234
```

application.yml

```
myapp:
  mail:
    to: frank@hotmail.com
    host: mail.example.com
    port: 250
    cc:
      - mike@gmail.com
      - david@gmail.com
    bcc:
      - john@hotmail.com
      - admin@acme.com
    credential:
      user-name: john1234
      password: xyz@1234
```

CustomerService

```java
@SpringBootApplication
@EnableFeignClients
public class CustomerServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(AccountServiceApplication.class, args);
  }
}
```

Use Feign

application.yml

```yaml
server:
  port: 8091
```

```xml
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

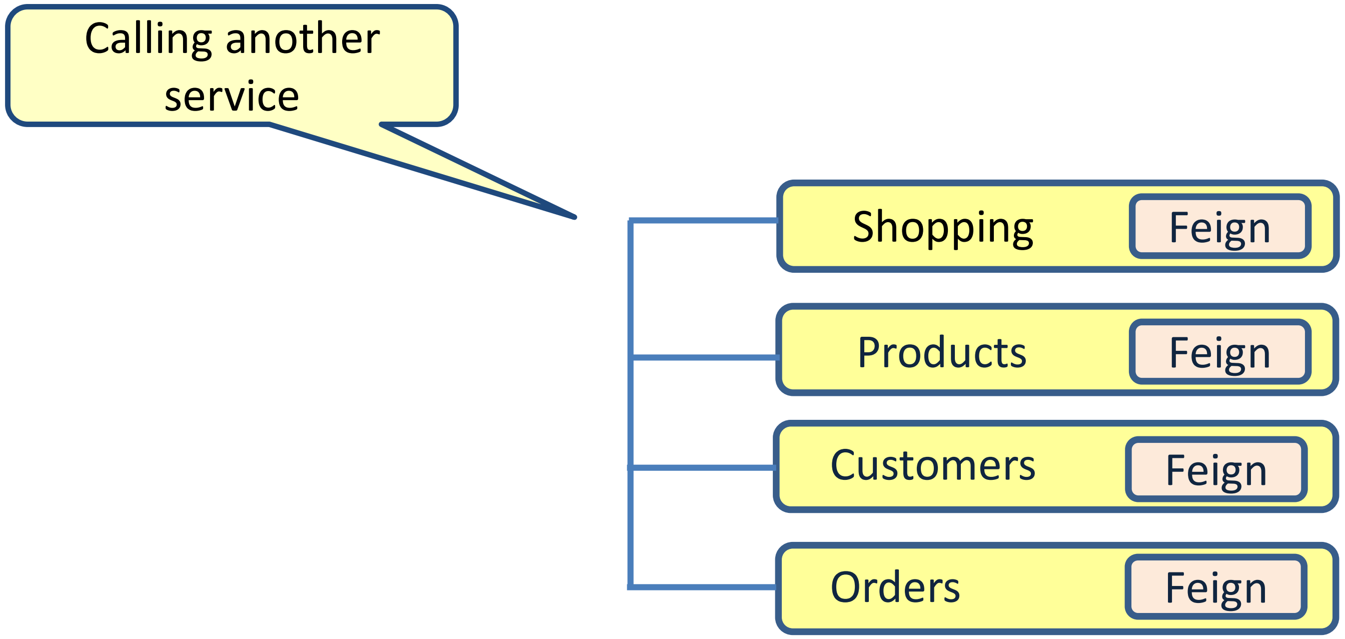
CustomerService: the controller

```
@RestController
public class CustomerController {
  @Autowired
  AccountFeignClient accountClient;

  @RequestMapping("/customer/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
        Account account = accountClient.getName(customerId);
        return account;
  }

  @FeignClient(name = "account-service", url = "http://localhost:8090")
  interface AccountFeignClient {
    @RequestMapping("/account/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId);
  }
}
```

Autowire the client

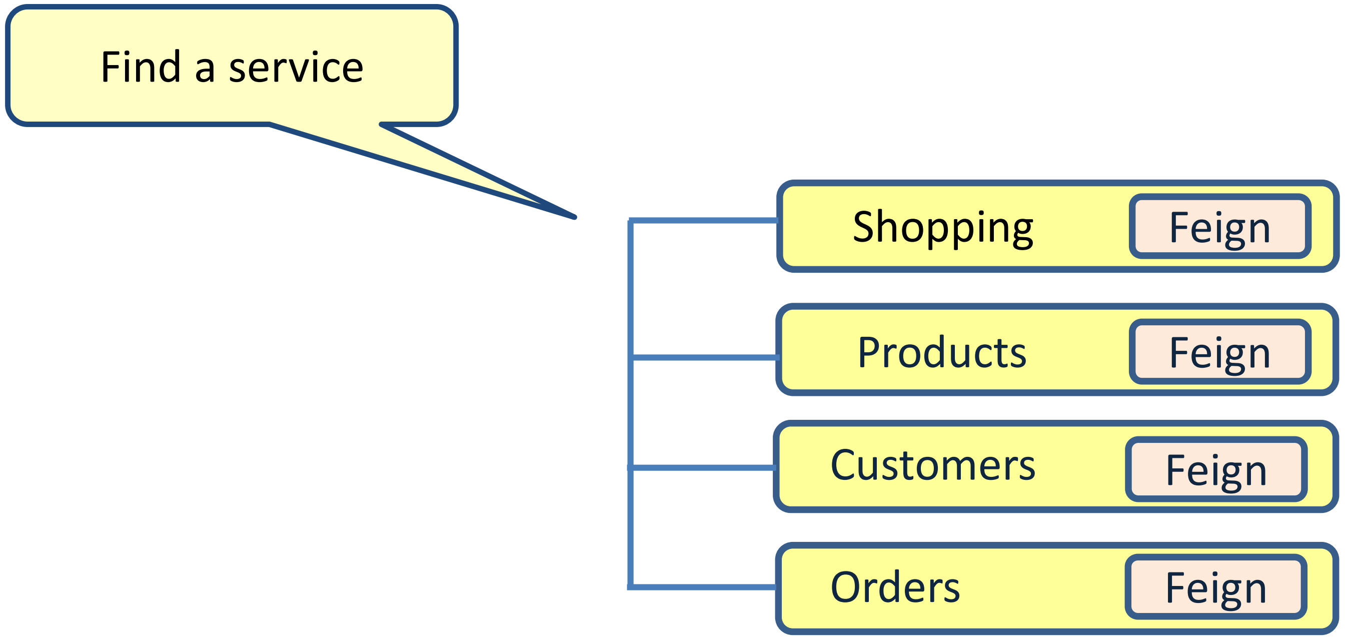
Declare the interface, Spring creates the implementation

Implementing microservices

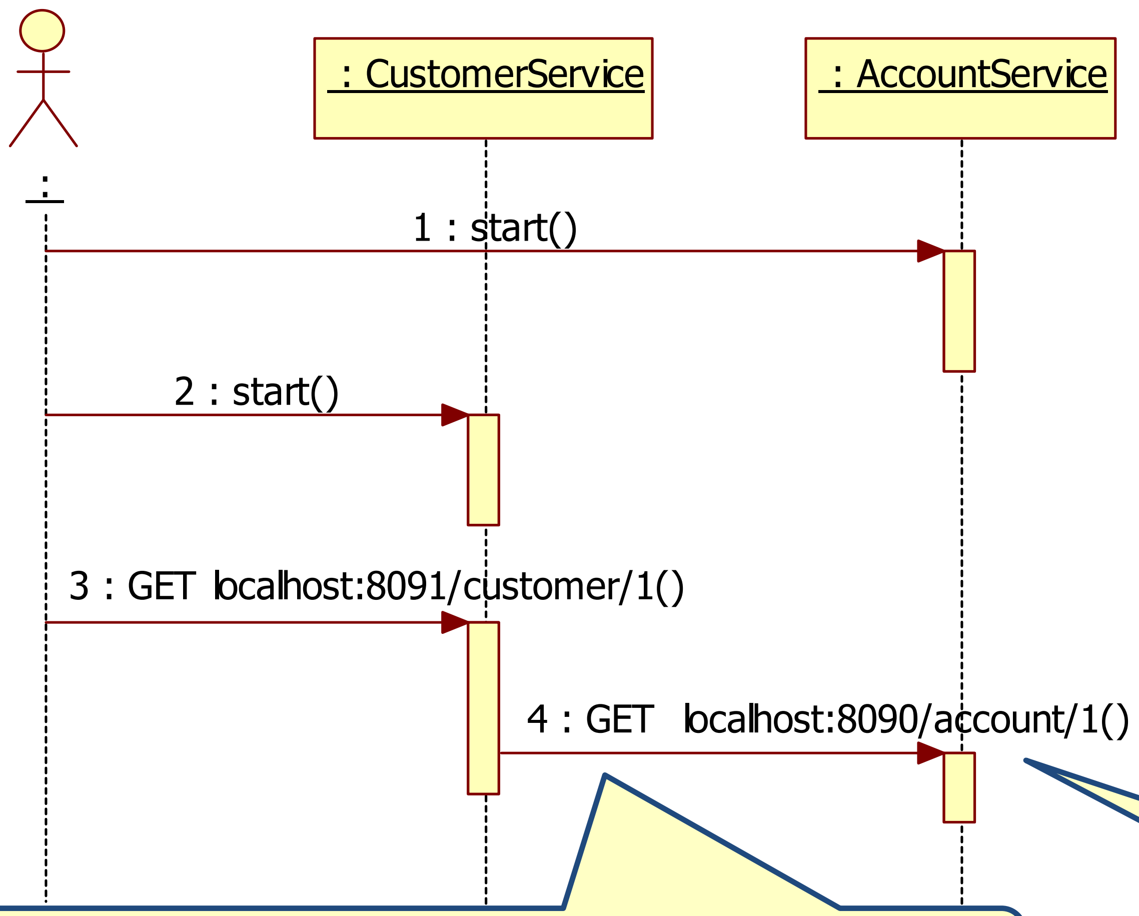


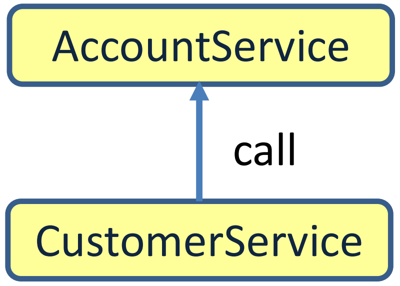
SERVICE REGISTRY: EUREKA

Implementing microservices



One service calling another service


: CustomerService
: AccountService
:
1 : start()
2 : start()
3 : GET localhost:8091/customer/1()
4 : GET localhost:8090/account/1()



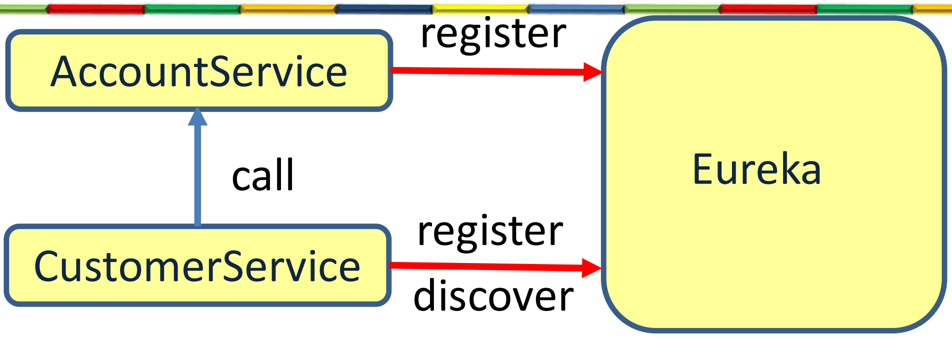
Address is hard coded

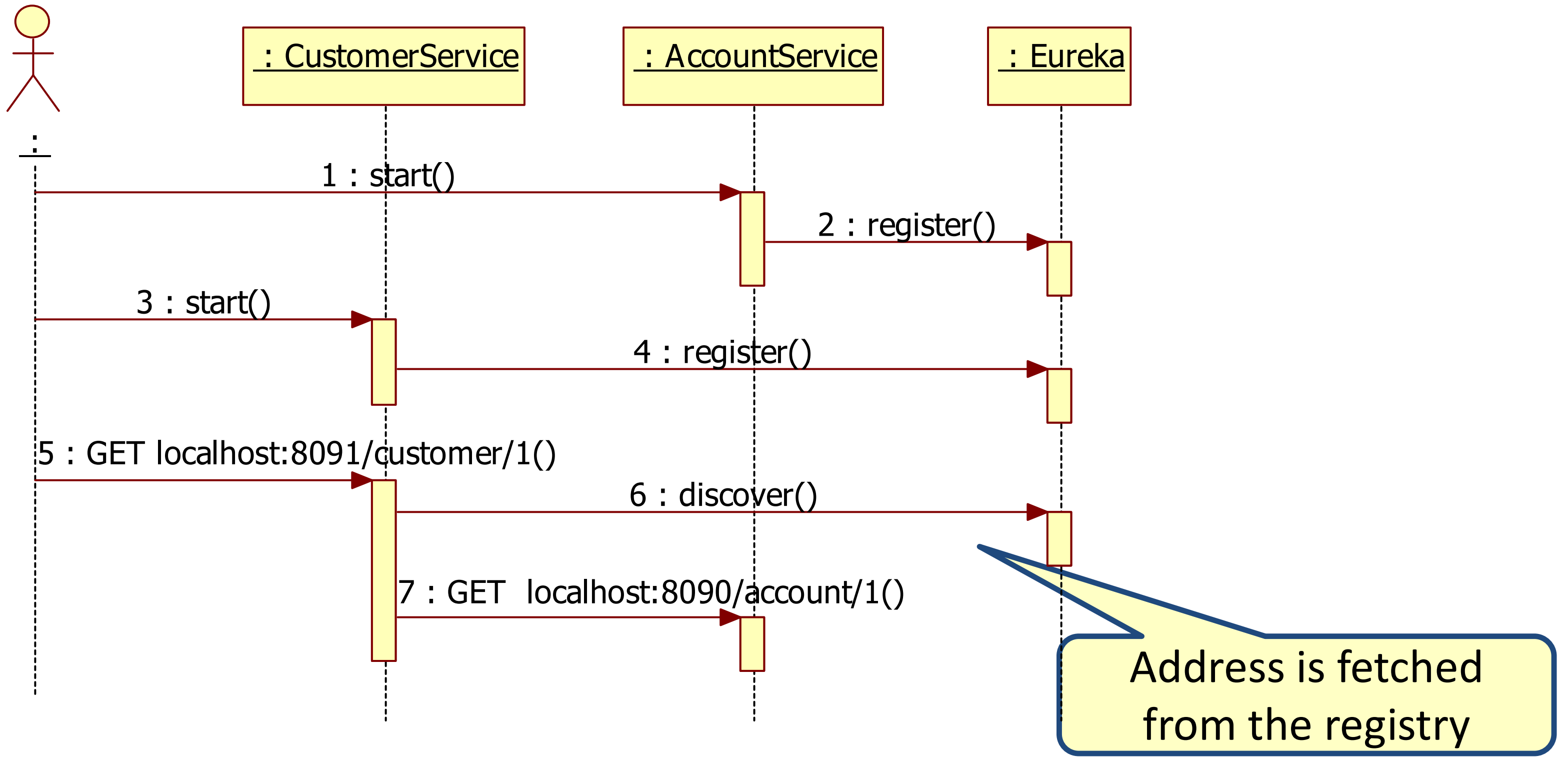
If I change the URL of the AccountService, I need to change the configuration of the CustomerService

Service Registry

- Like the phone book for microservices
  - Services register themselves with their location and other meta-data
  - Clients can lookup other services
- Netflix Eureka

Using Eureka





Why service registry/discovery?

1. Loosely coupled services
   - Service consumers should not know the physical location of service instances.
     - We can easily scale up or scale down service instances
2. Increase application resilience
   - If a service instance becomes unhealthy or unavailable, the service discovery engine will remove that instance from the list of available services.

Eureka Server

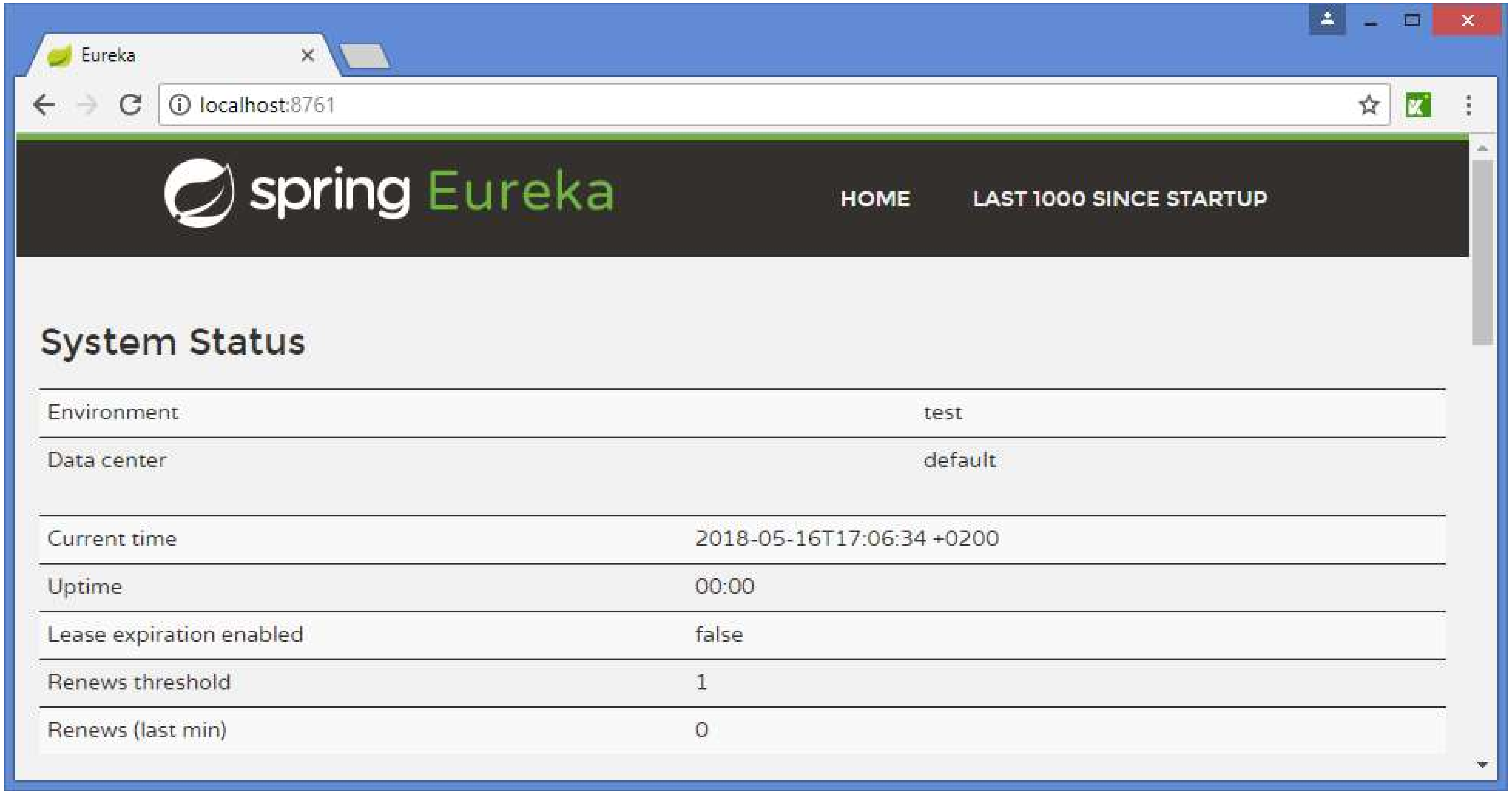
```java
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

  public static void main(String[] args) {
    SpringApplication.run(EurekaServerApplication.class, args);
  }
}
```

application.yml

```yaml
server:
  port: 8761

eureka:
  client:
    registerWithEureka: false   #telling the server not to register himself
    fetchRegistry: false
```

```xml
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Running Eureka



System Status

Environment	test
Data center	default

Current time	2018-05-16T17:06:34 +0200
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

AccountService

```java
@SpringBootApplication
@EnableDiscoveryClient
public class AccountServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(AccountServiceApplication.class, args);
  }
}
```

application.yml

```yaml
server:
  port: 8090

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/

spring:
  application:
    name: AccountService
```

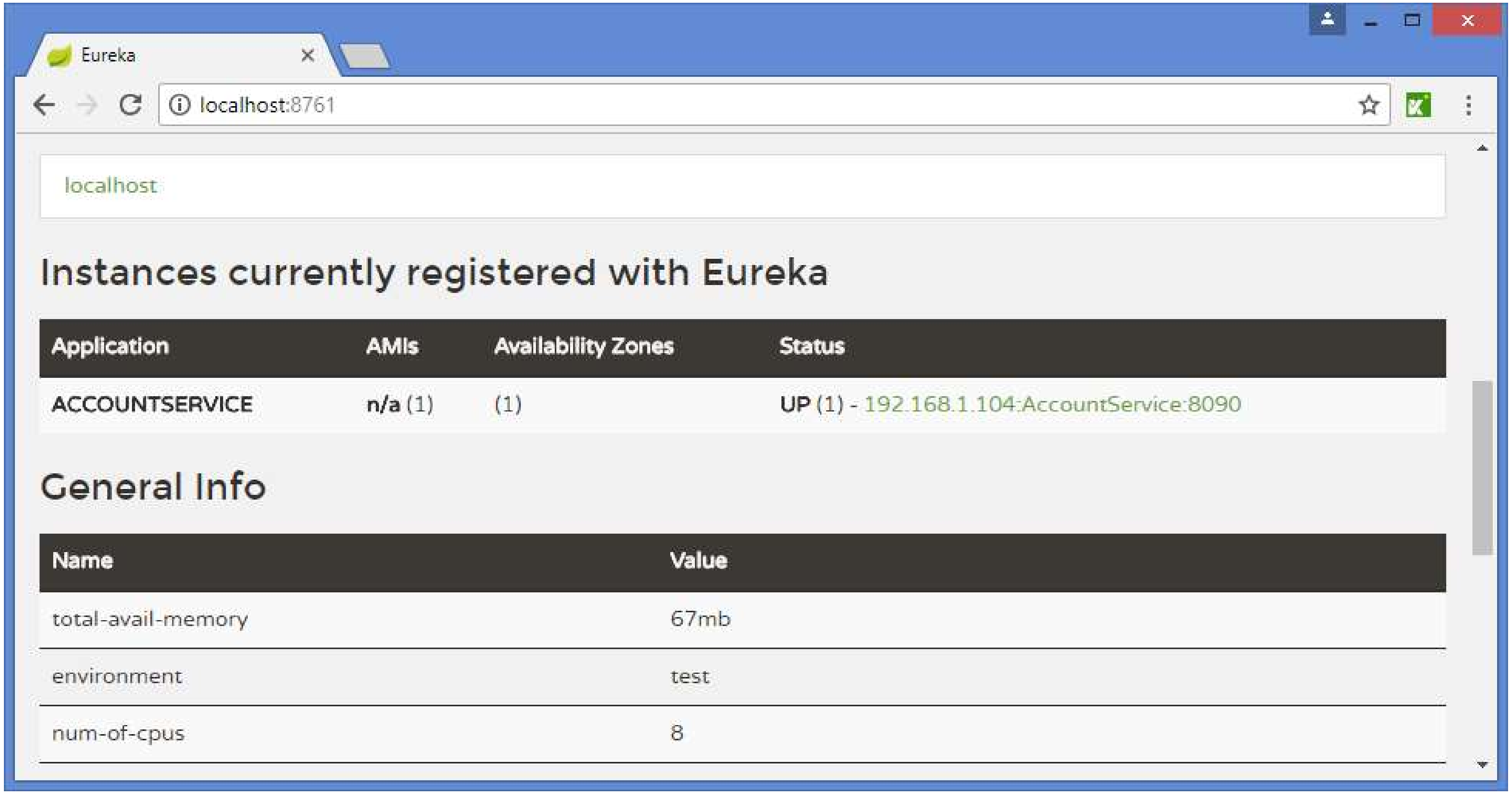
AccountService

```java
@RestController
public class AccountController {
  @RequestMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    return new Account("1234", "1000.00");
  }
}
```

```java
public class Account {
  private String accountNumber;
  private String balance;
  ...
}
```

```xml
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

Running the AccountService



localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 192.168.1.104:AccountService:8090

General Info

Name	Value
total-avail-memory	67mb
environment	test
num-of-cpus	8

CustomerService

```java
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class CustomerServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(AccountServiceApplication.class, args);
  }
}
```

Use Feign and Eureka

application.yml

```yaml
server:
  port: 8091

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/

spring:
  application:
    name: CustomerService
```

CustomerService: the controller

```
@RestController
public class CustomerController {
  @Autowired
  AccountFeignClient accountClient;

  @RequestMapping("/customer/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    Account account = accountClient.getName(customerId);
    return account;
  }

  @FeignClient("AccountService")
  interface AccountFeignClient {
    @RequestMapping("/account/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId);
  }
}
```

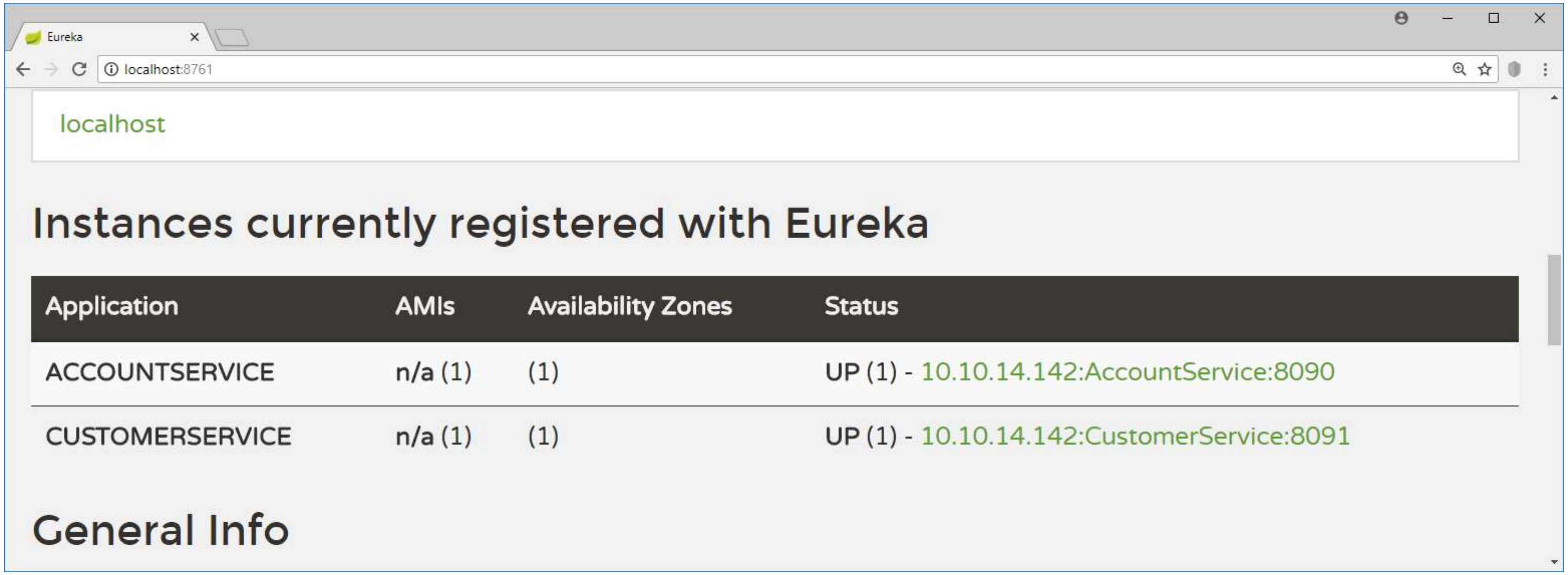
Name of the service instead of the URL

Feign works together with Eureka

application.yml

```
server:
  port: 8091
```

Running the CustomerService



localhost

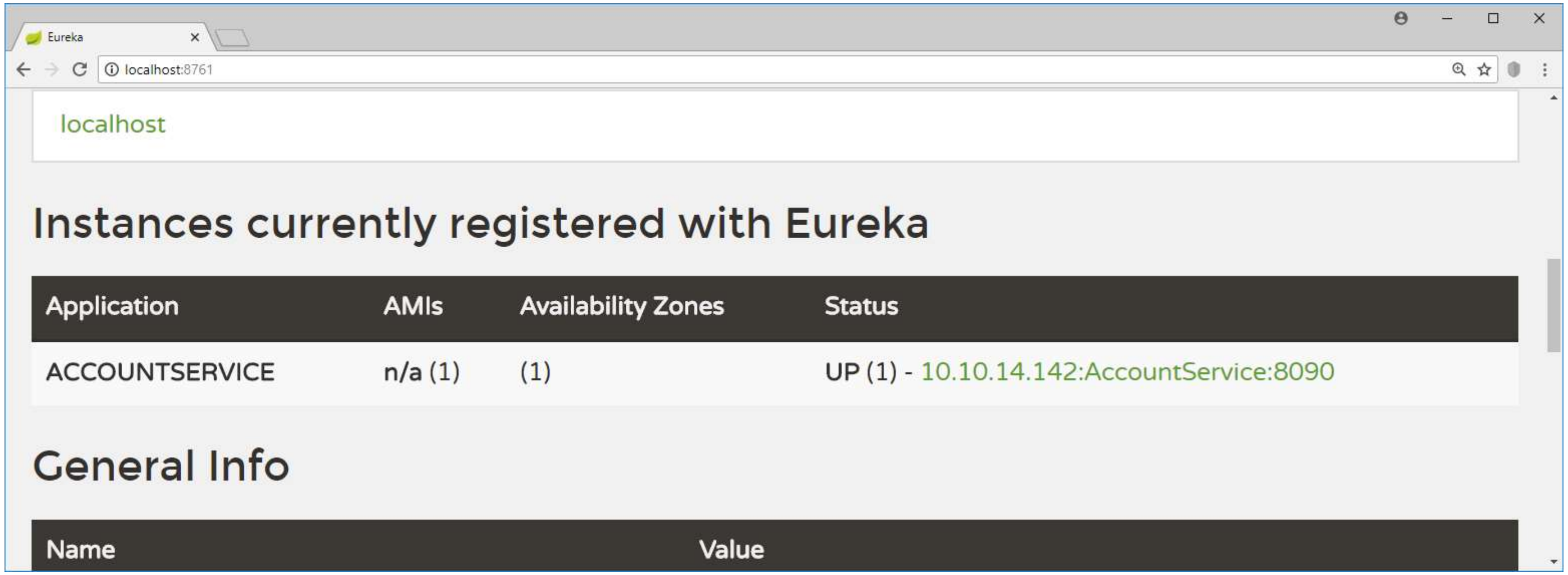
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:AccountService:8090
CUSTOMERSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:CustomerService:8091

General Info

Stopping the CustomerService

- Eureka monitors the health of registered services.
- If we stop the CustomerService, Eureka will notice that automatically



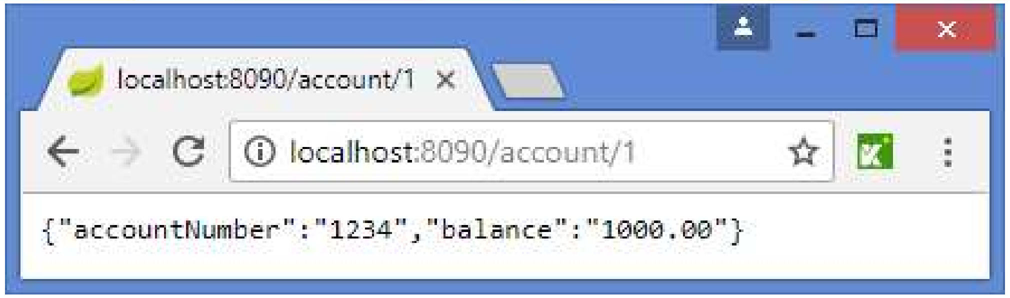
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:AccountService:8090

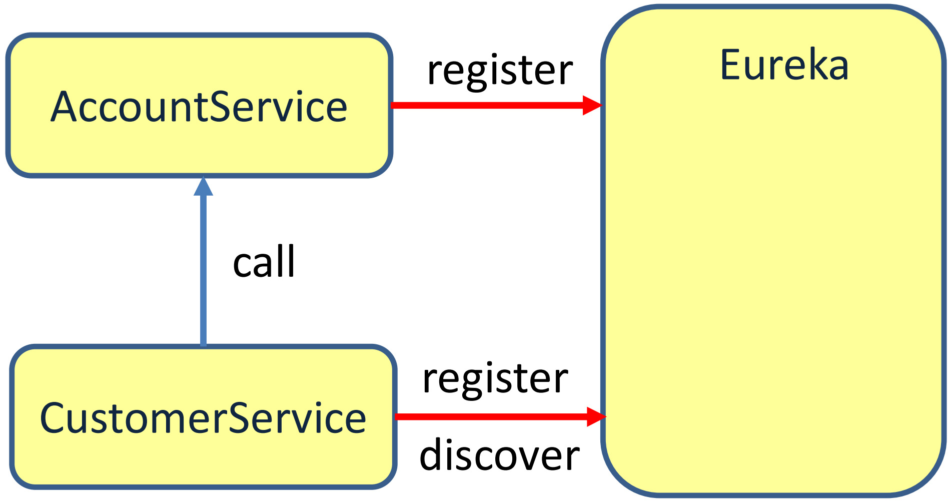
General Info

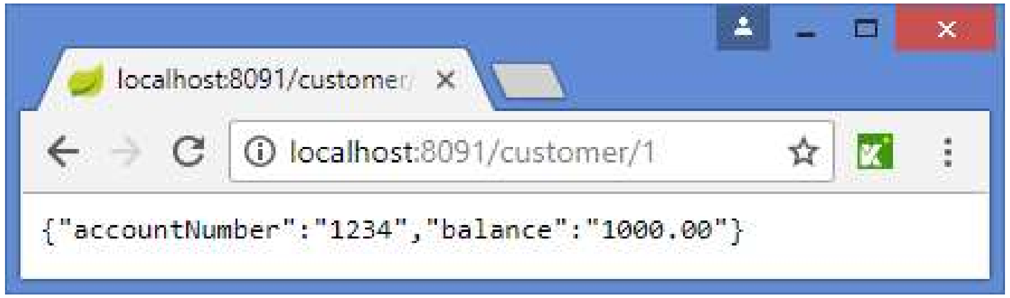
Name	Value

Using Eureka


localhost:8090/account/1

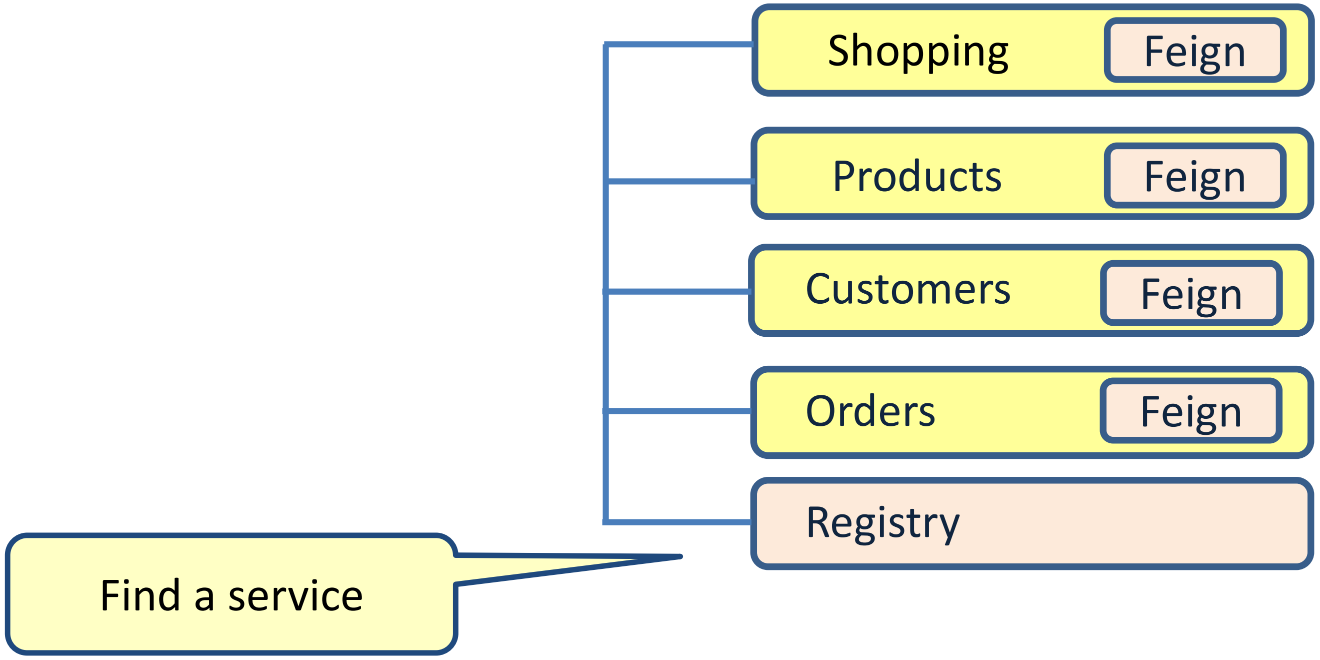
```
{"accountNumber":"1234","balance":"1000.00"}
```




localhost:8091/customer/1

```
{"accountNumber":"1234","balance":"1000.00"}
```

Implementing microservices


Shopping
Feign
Products
Feign
Customers
Feign
Orders
Feign
Registry
Find a service