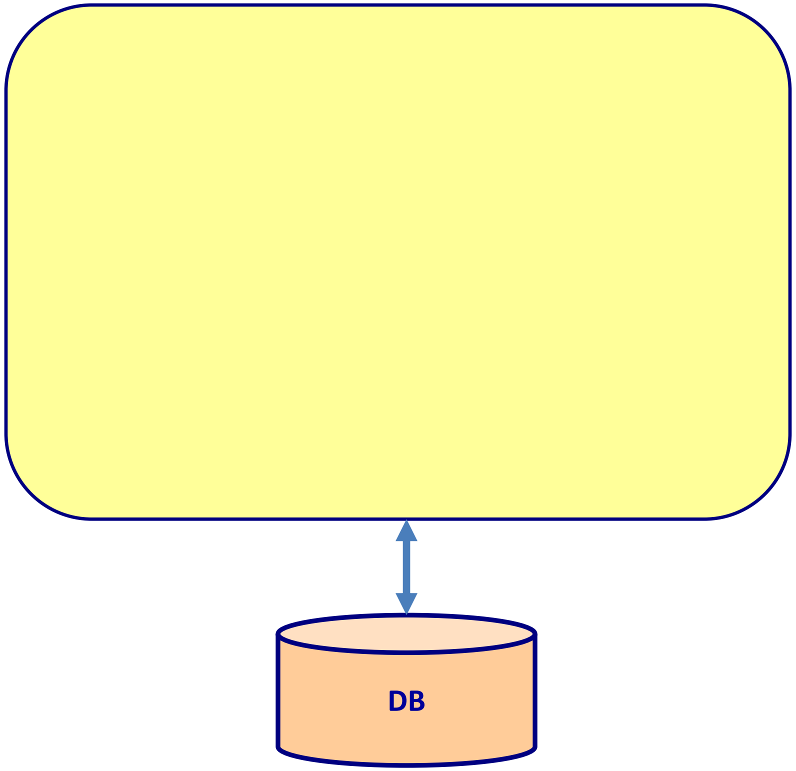
Lesson 7

MICROSERVICES

MONOLITH ARCHITECTURE

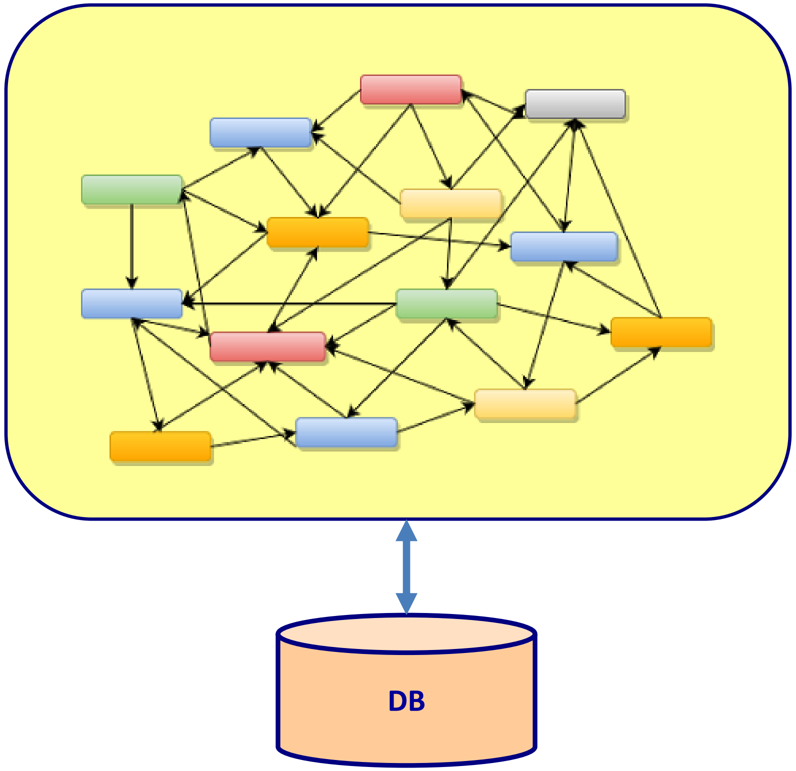
Monolith architecture

- Everything is implemented in one large system



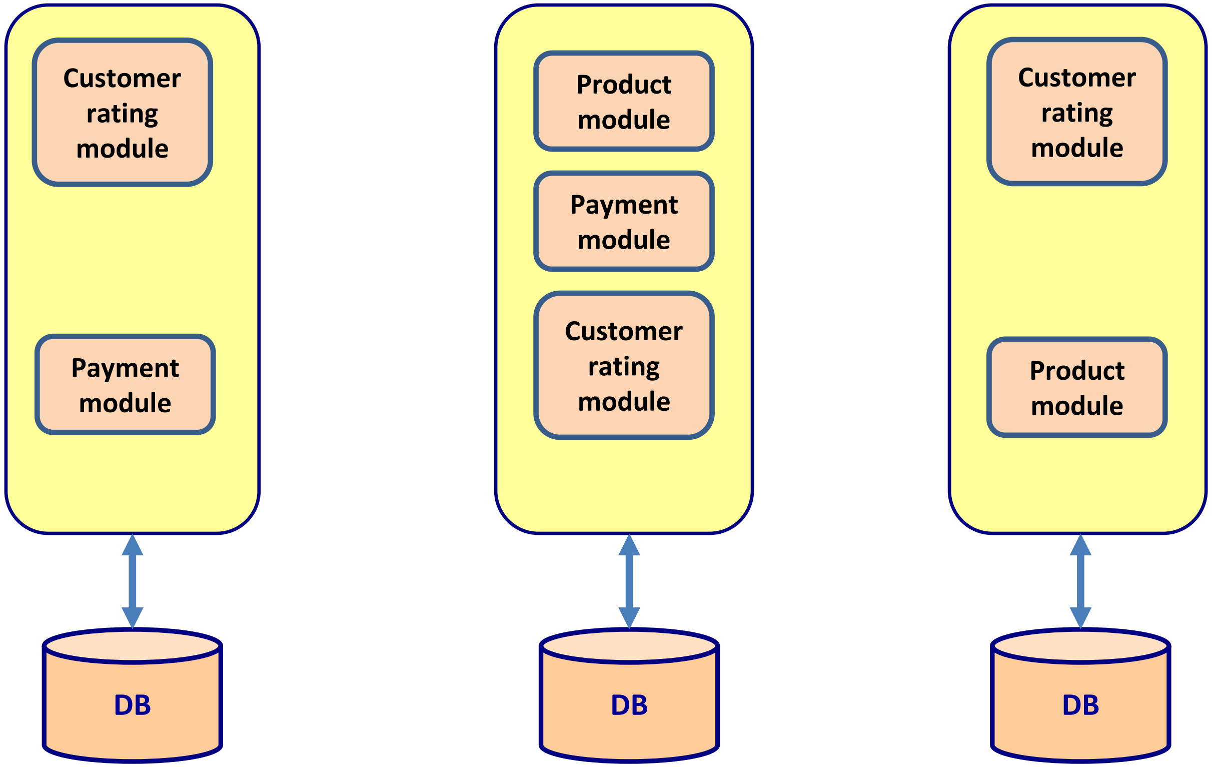
Monolith architecture

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



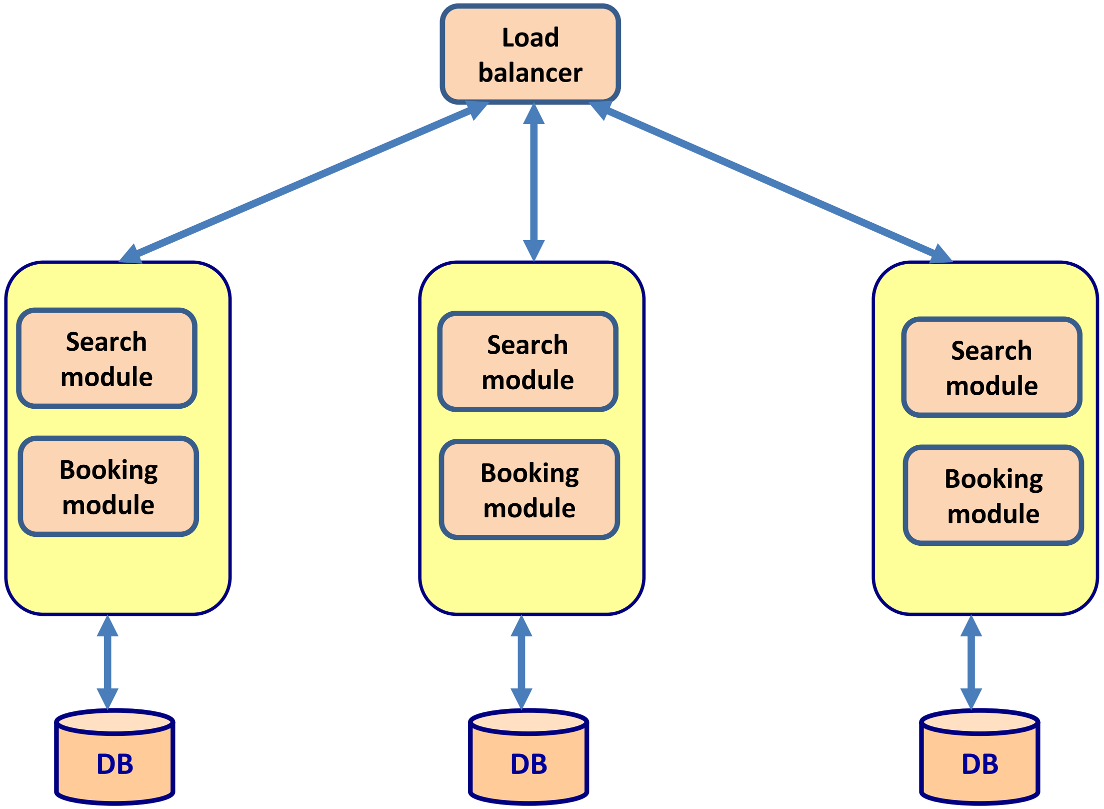
Monolith architecture

- Limited re-use is realized across monolithic applications



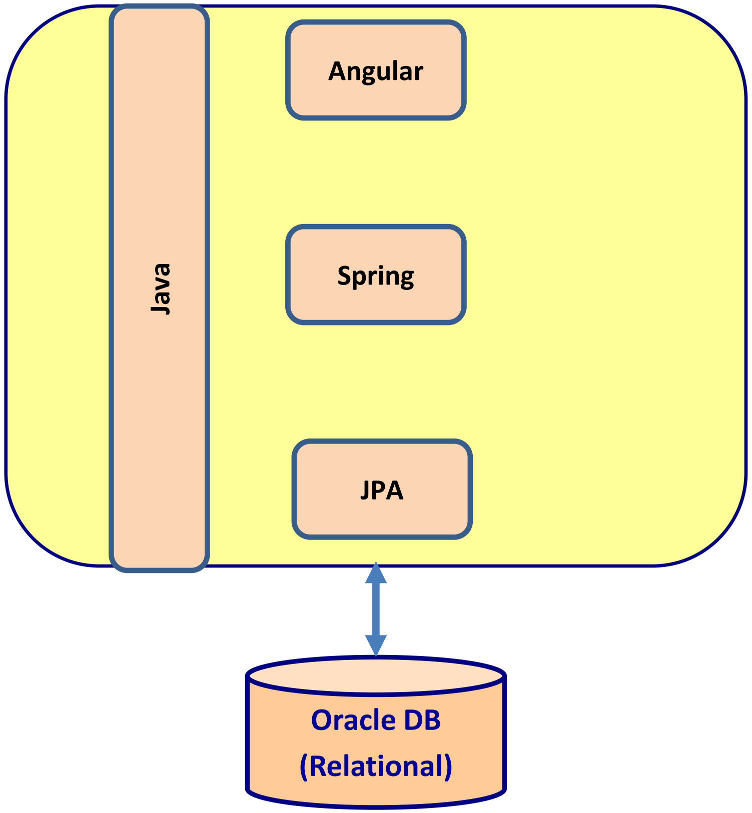
Monolith architecture

- All or nothing scaling
  - Difficult to scale separate parts



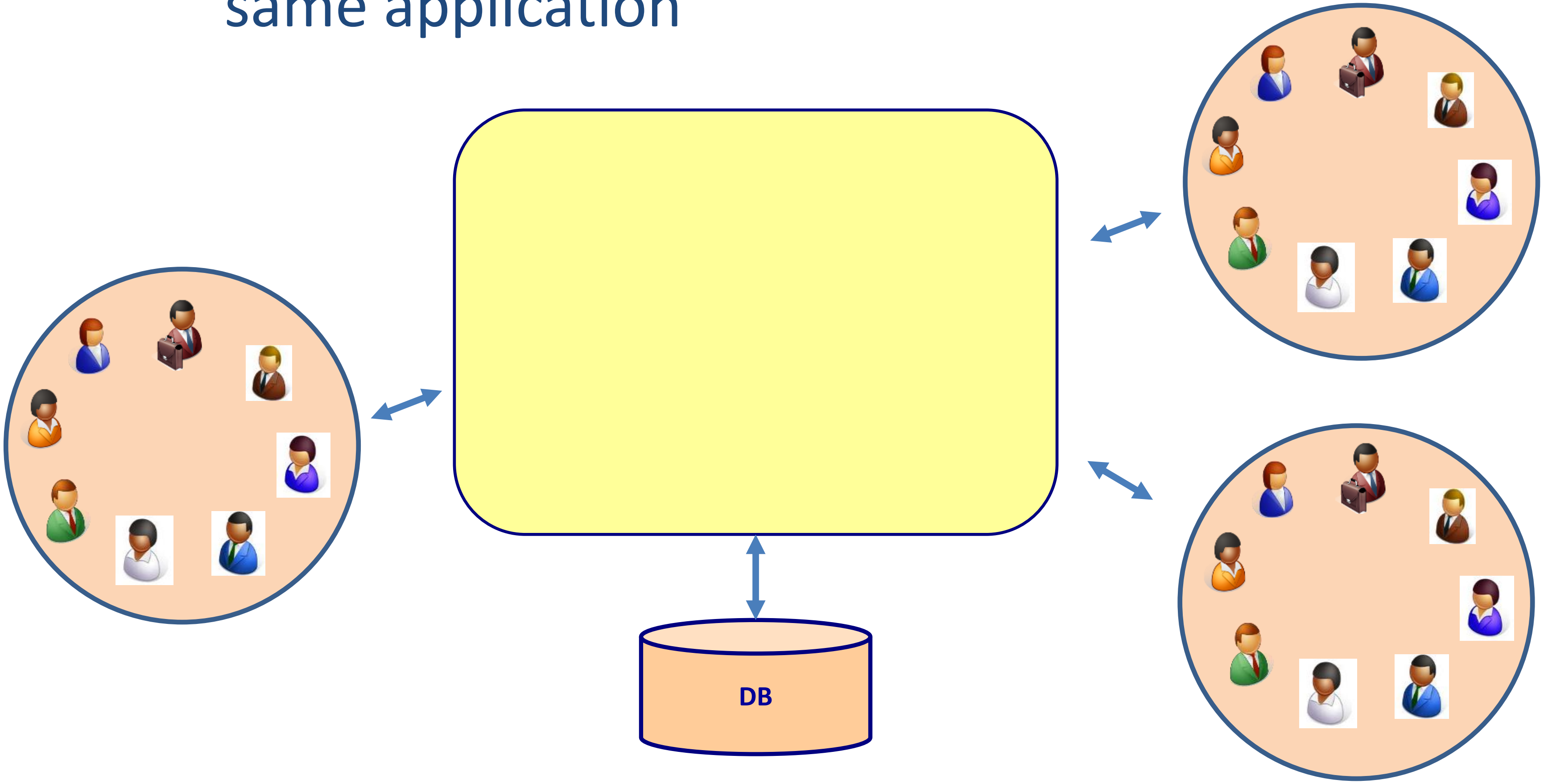
Monolith architecture

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



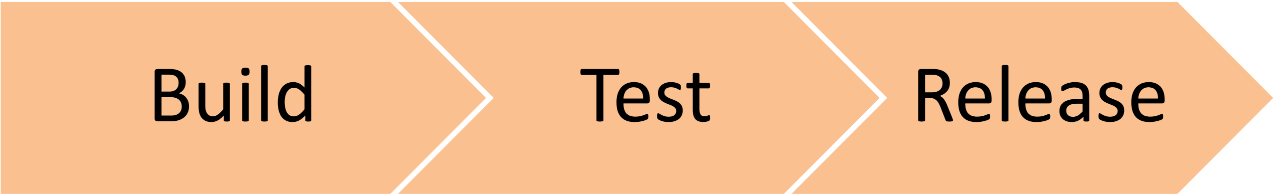
Monolith architecture

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

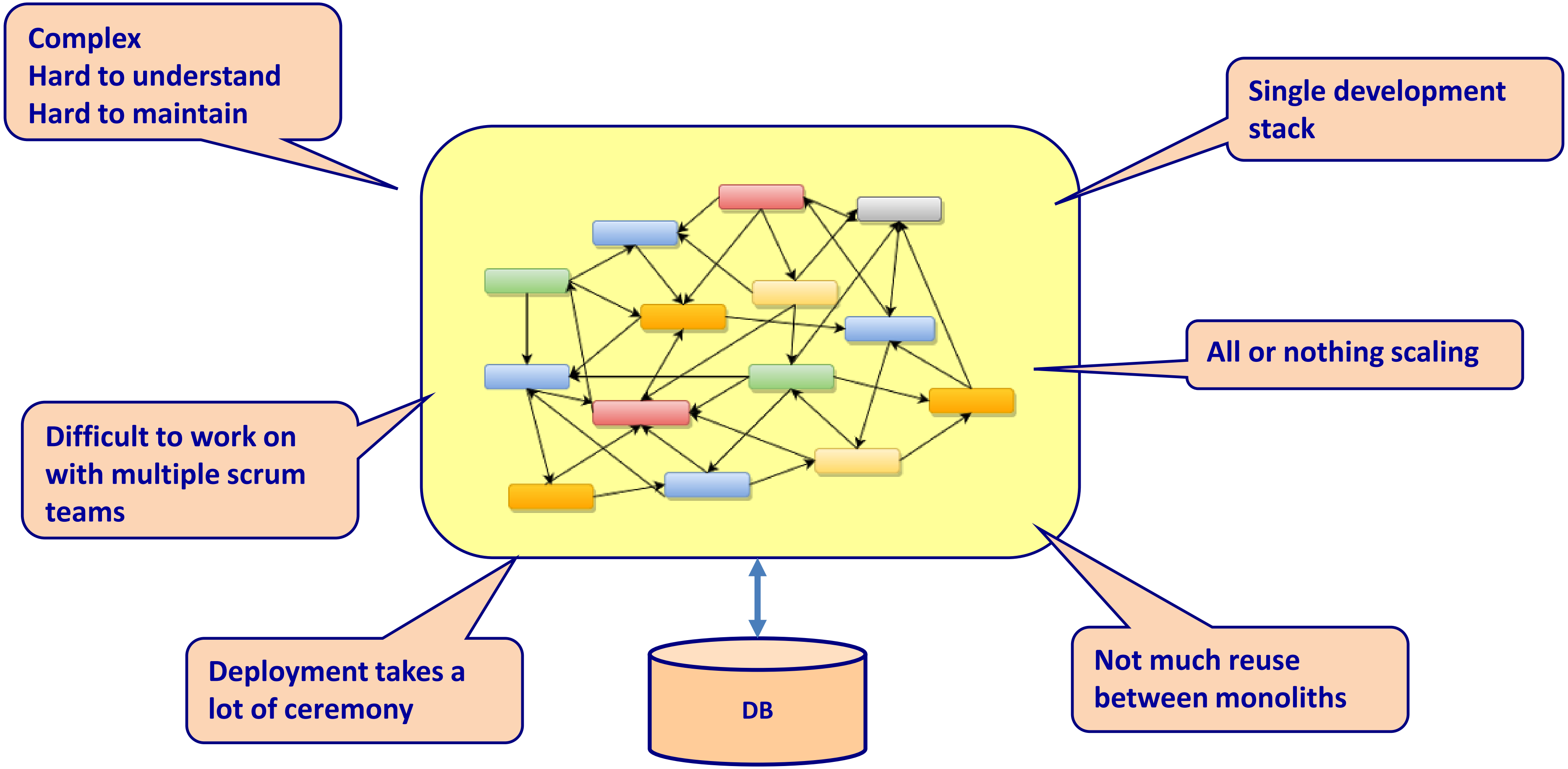


Monolith architecture

- 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

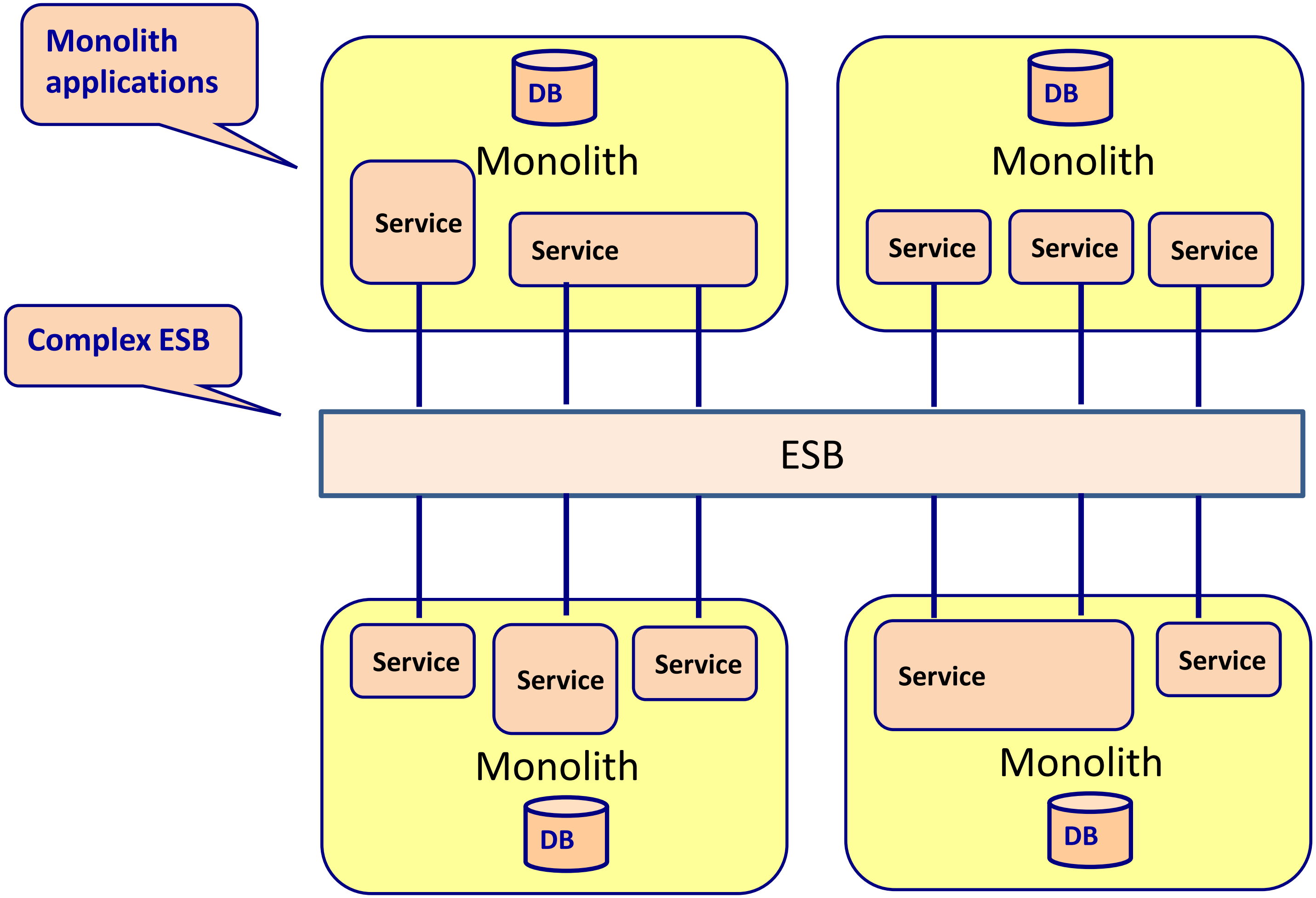
All or nothing scaling

Difficult to work on with multiple scrum teams

Deployment takes a lot of ceremony

Not much reuse between monoliths

Problems with SOA



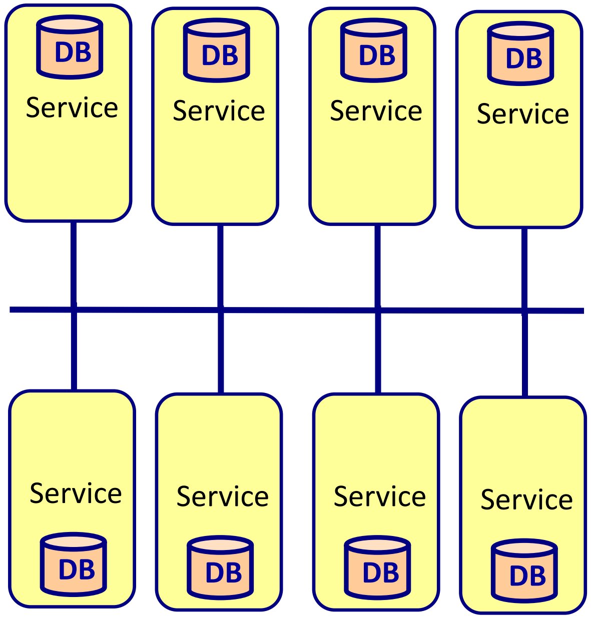
Microservice early adopters

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

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

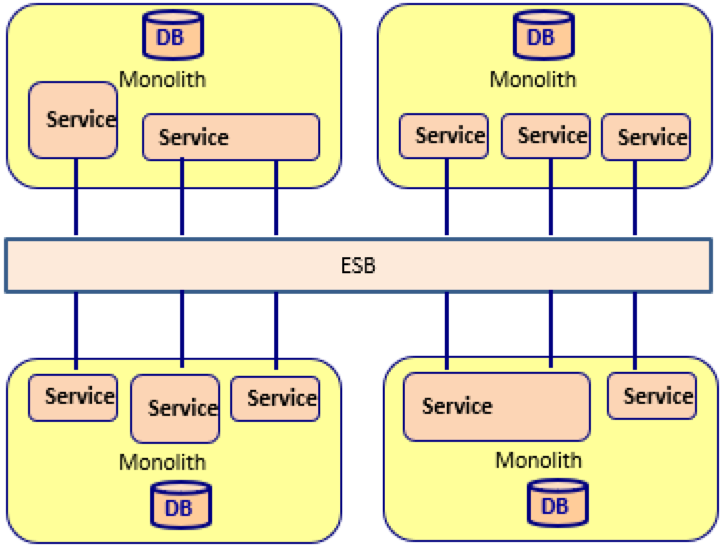
Microservices

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



Orchestration vs. choreography

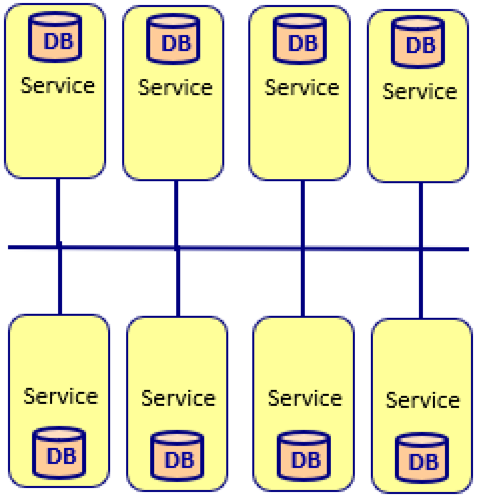
- Orchestration
  - SOA



Easy to follow the process

Does not work well in large and or complex applications

- Choreography
  - Microservices



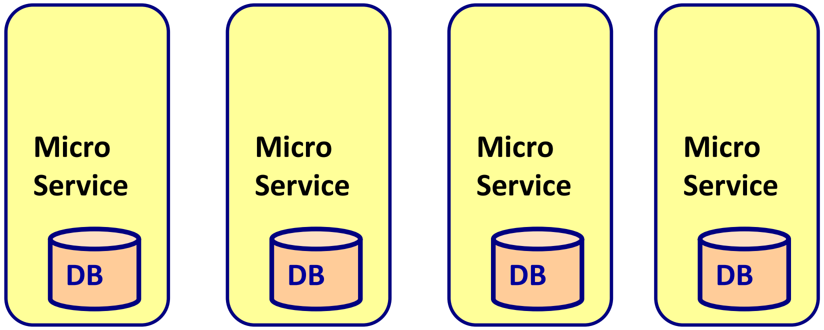
Hard to follow the process

Does work well in large and or complex applications

CHARACTERISTICS OF A MICROSERVICE

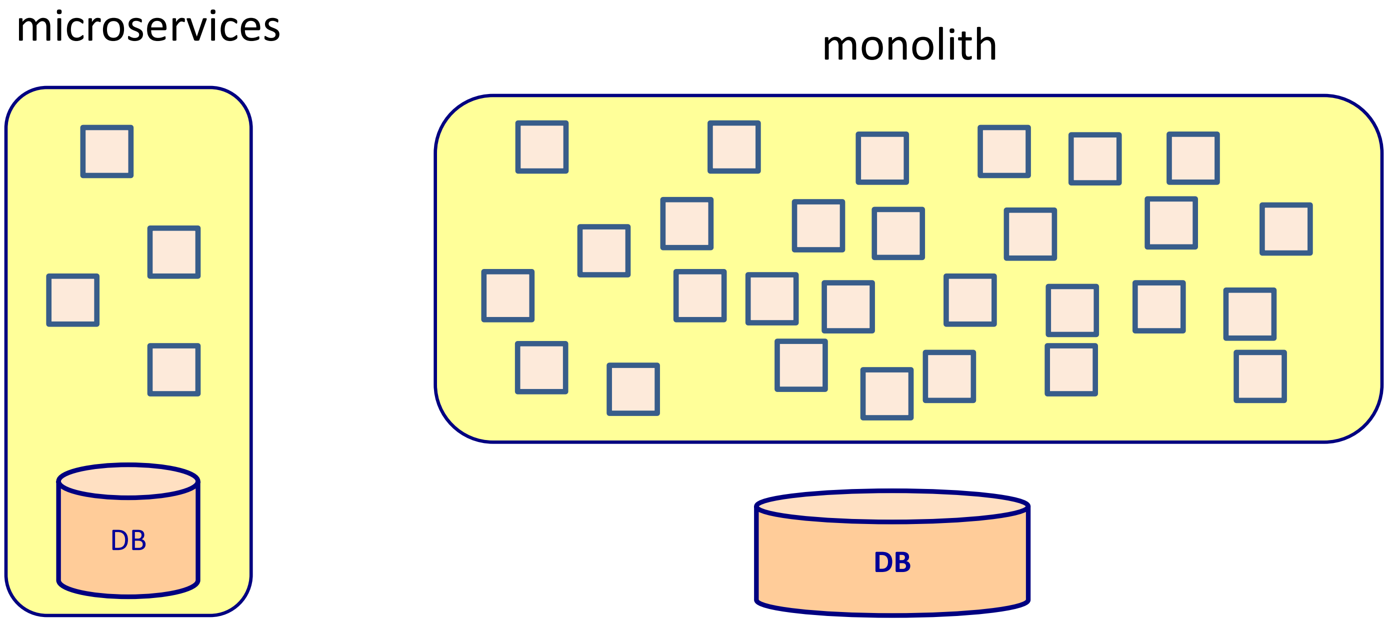
Microservices

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

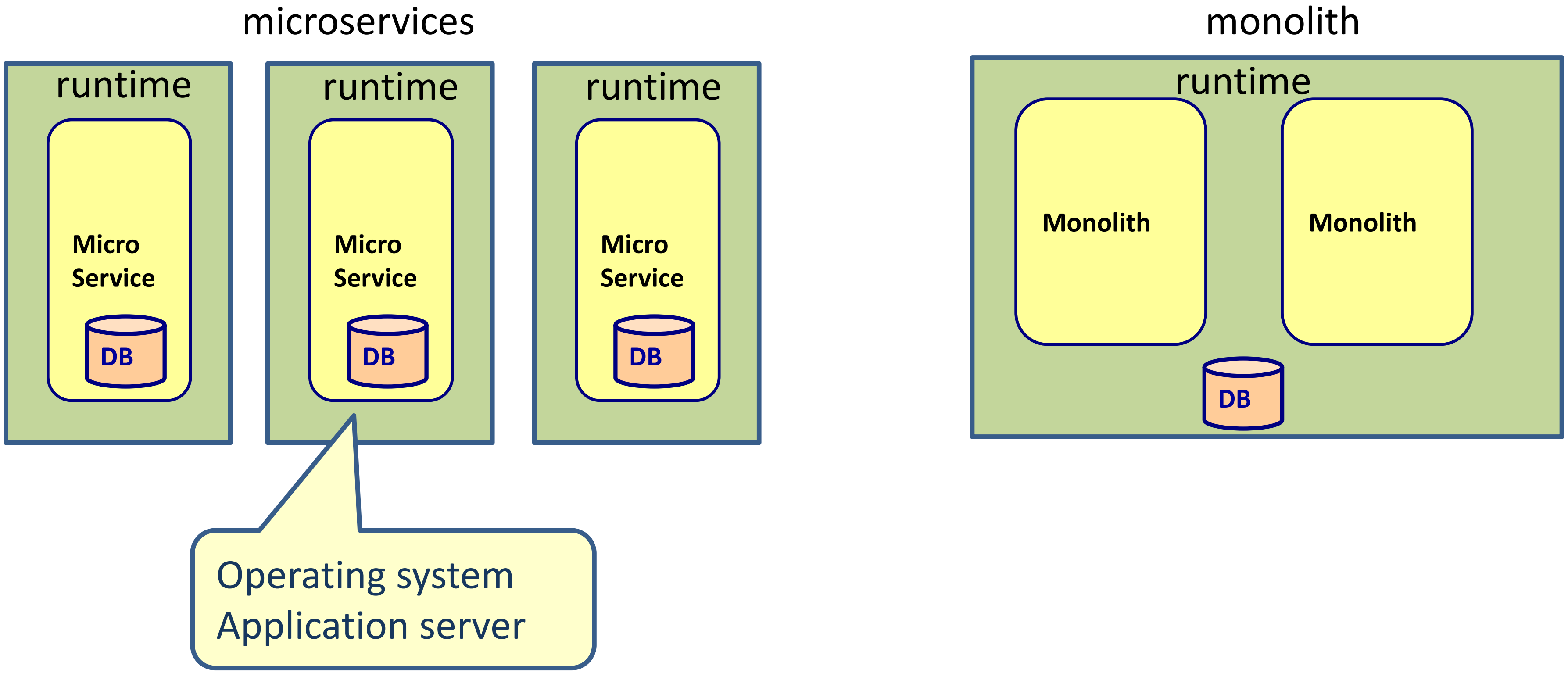


Simple and lightweight

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



Runs in an independent process



microservices

runtime
Micro Service
DB

runtime
Micro Service
DB

runtime
Micro Service
DB

Operating system
Application server

monolith

runtime
Monolith
Monolith
DB

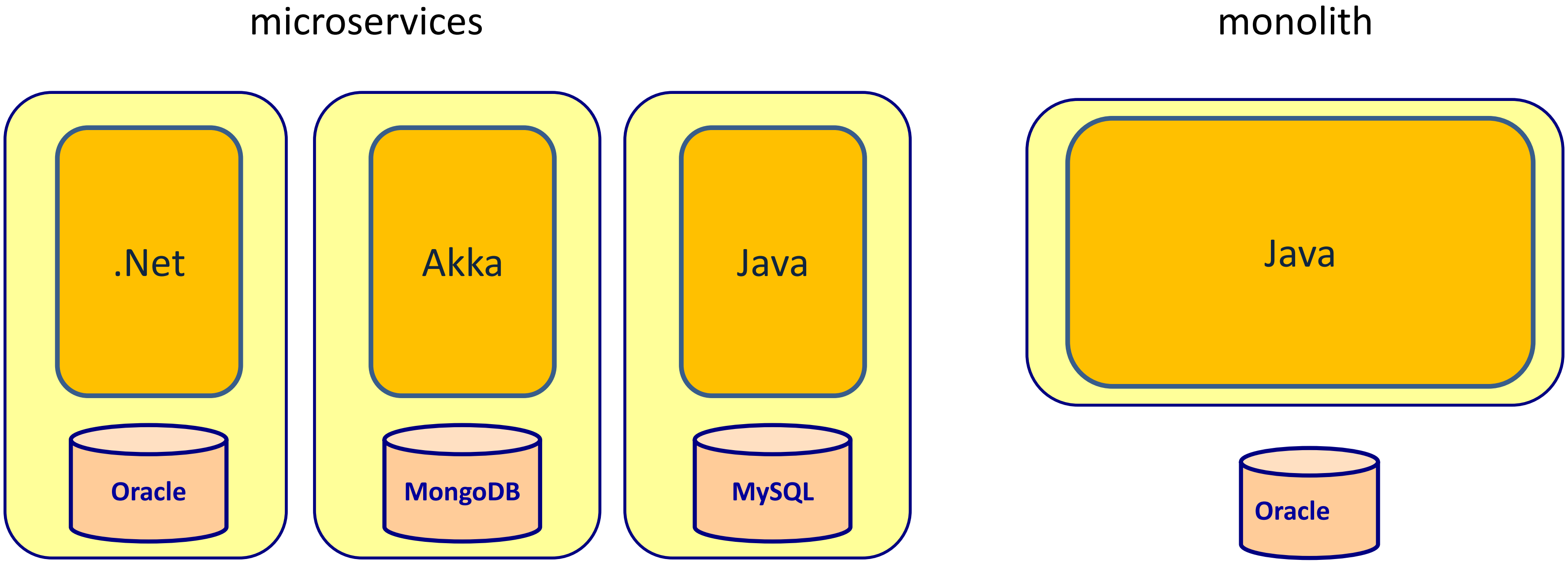
Advantages

- 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

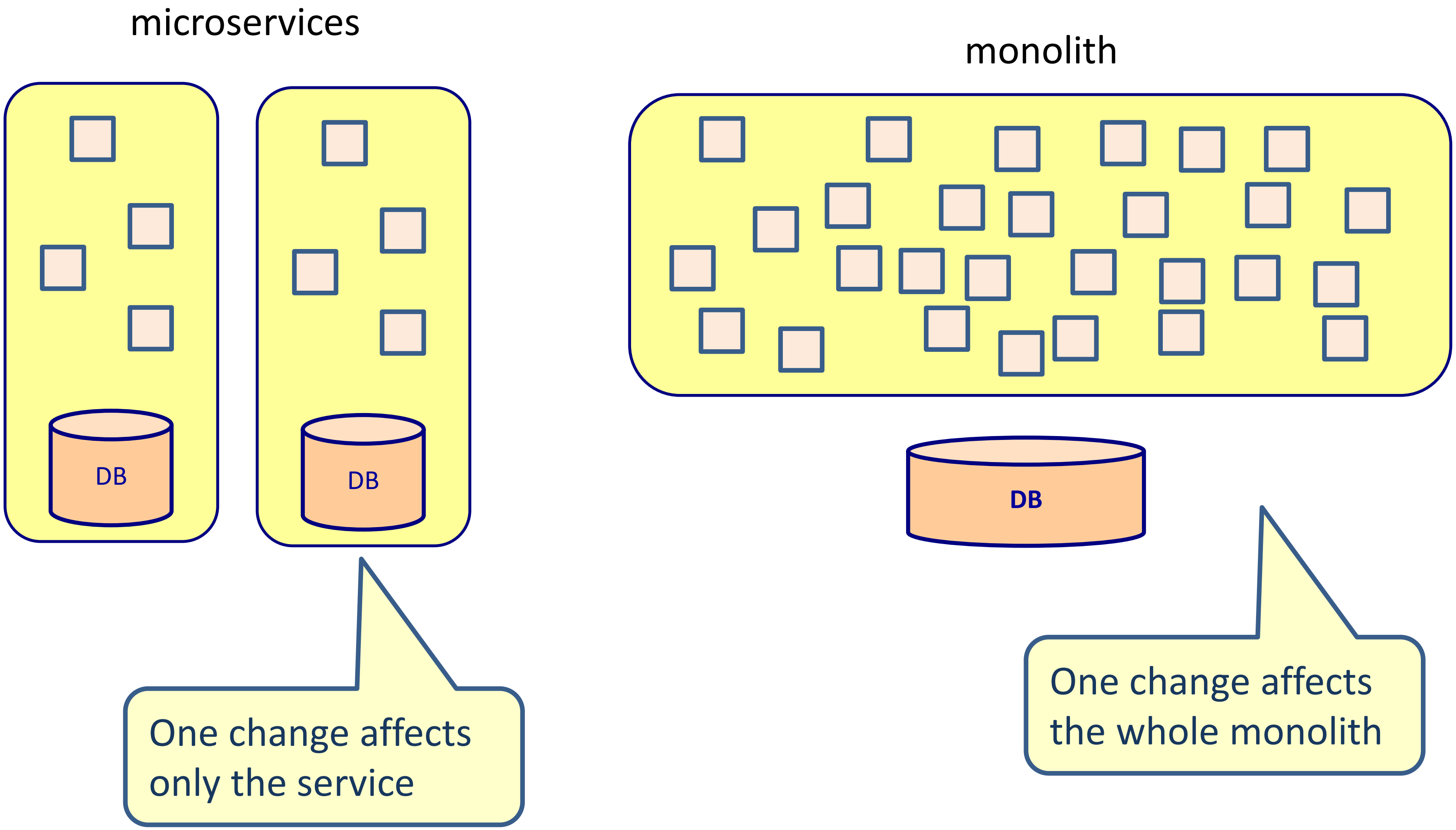
- We need to manage many runtimes

Technology agnostic



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

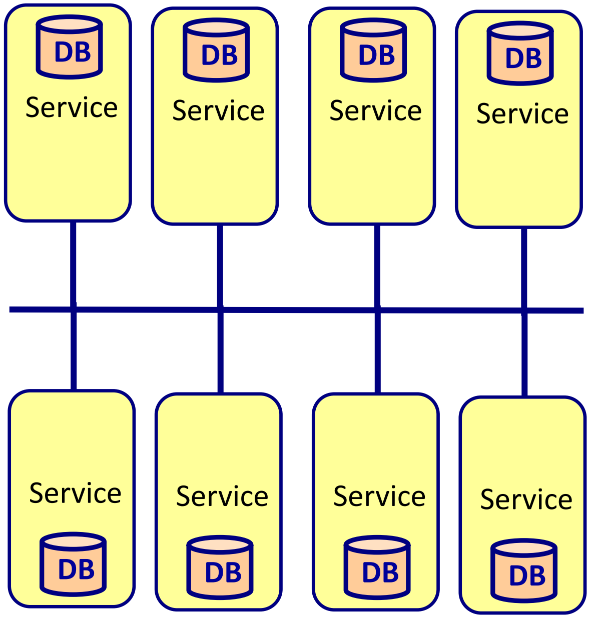
Decoupled



One change affects only the service

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

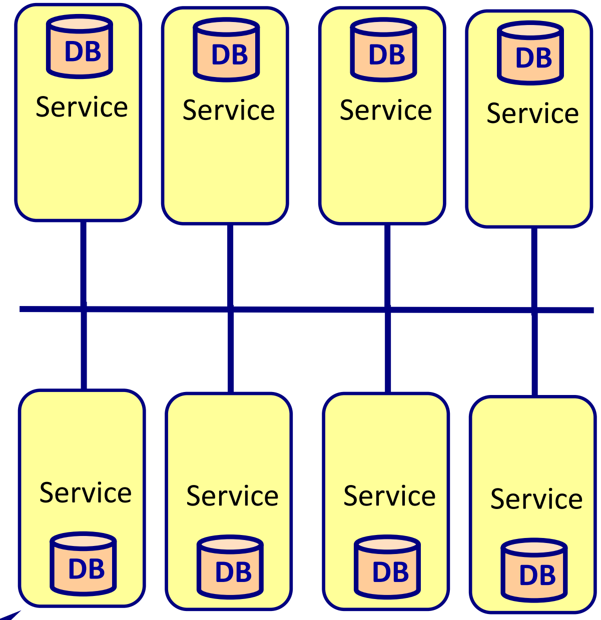
Every scrum teams
owns one or more
services

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

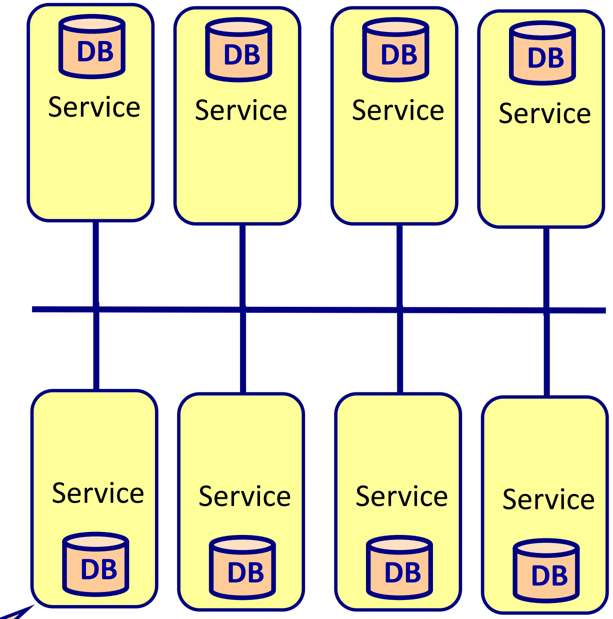
Every scrum/devops team owns one or more services

More reuse of services

Deployment of a microservice is simpler

Easy to add new microservices (Open-closed principle)

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)

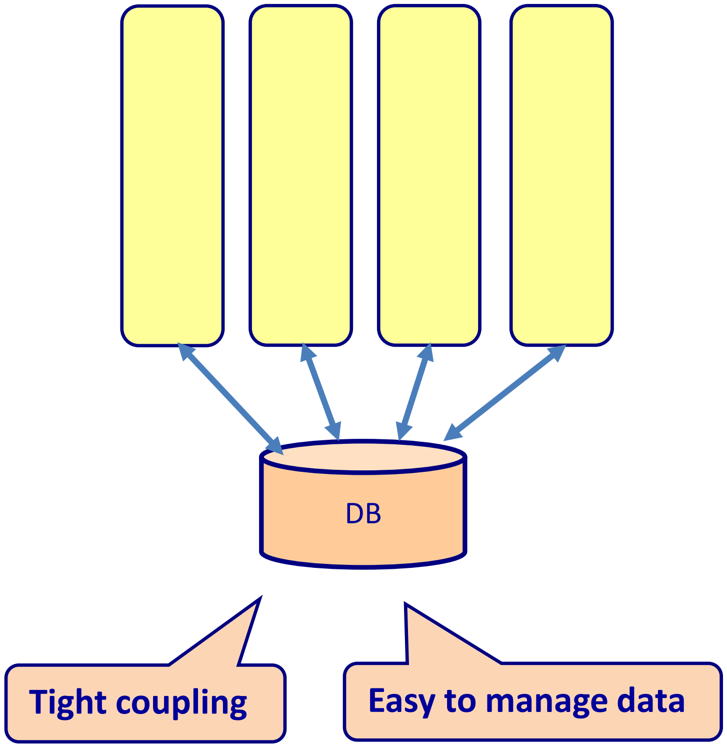
Health of the microservice architecture is much harder to monitor

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	

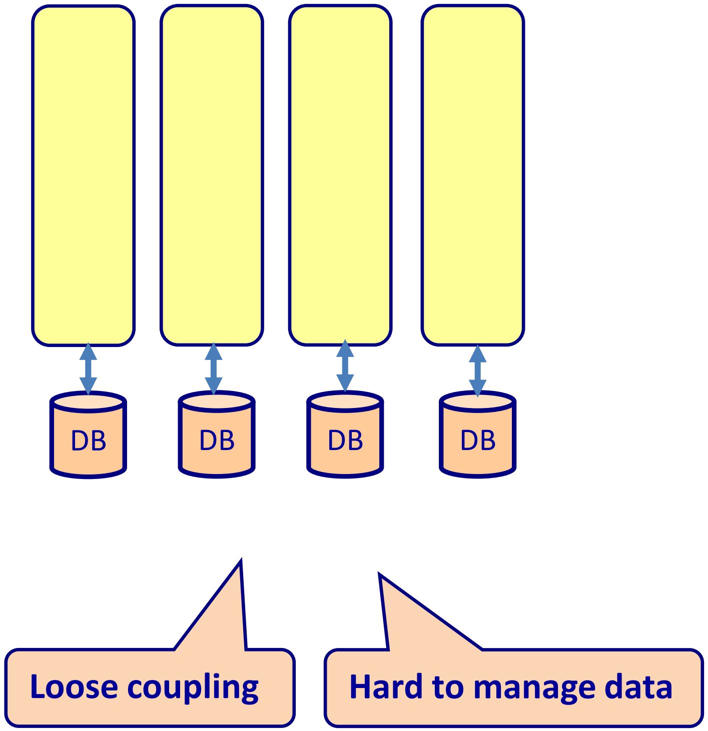
MICROSERVICE AND DATABASES

Every service manages its own data



Tight coupling

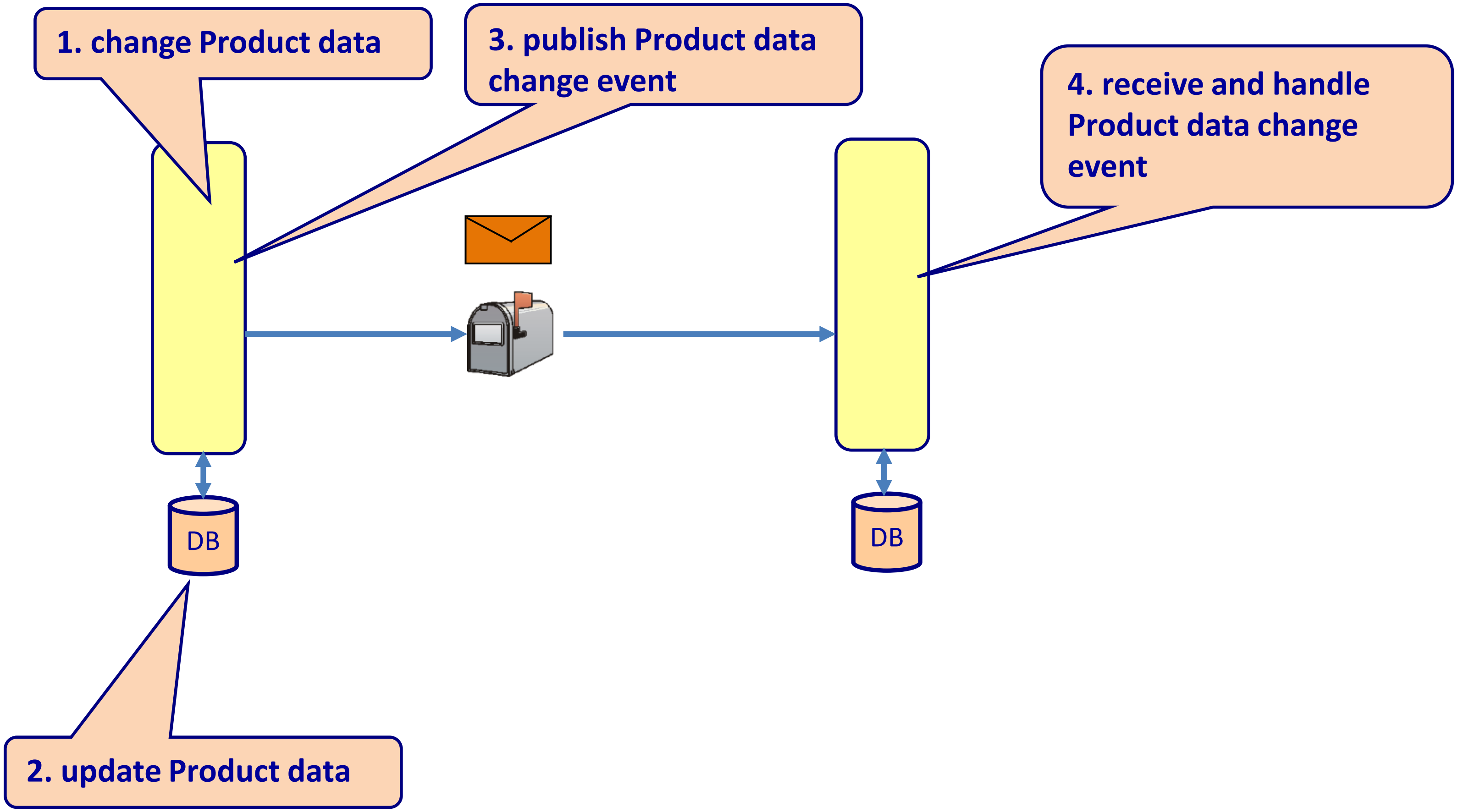
Easy to manage data



Loose coupling

Hard to manage data

Data consistency



1. change Product data

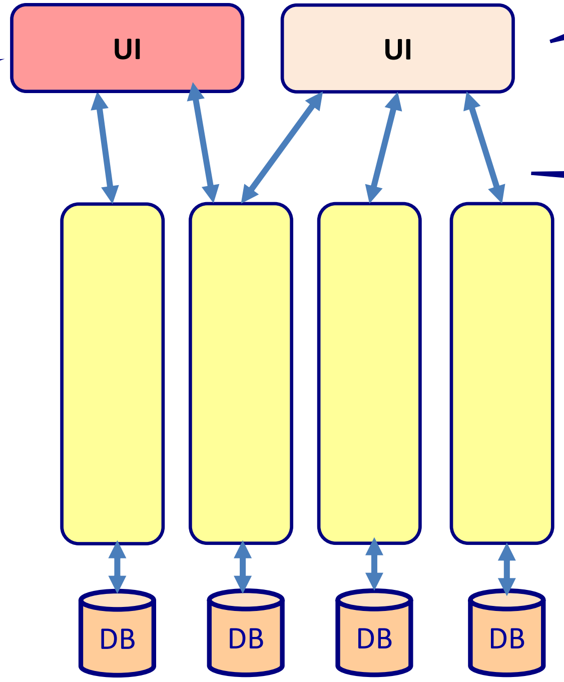
3. publish Product data change event

4. receive and handle Product data change event

2. update Product data

UI AND MICROSERVICE

Split front-end and back-end



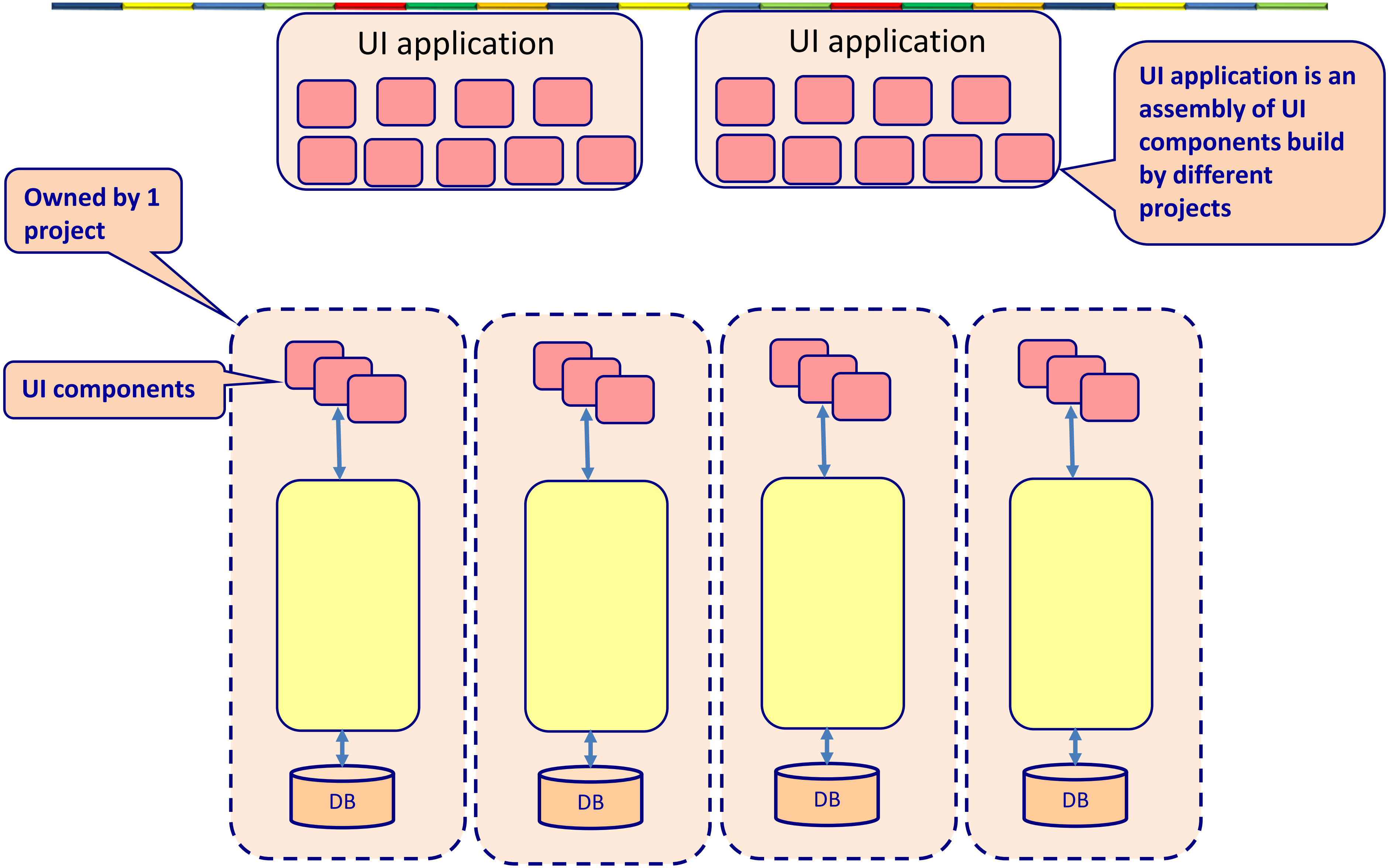
Different client types can access the services

UI can be developed independently from the microservices

UI can access all other services

Services are simpler

Micro frontend



UI application is an assembly of UI components build by different projects

Owned by 1 project

UI components

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

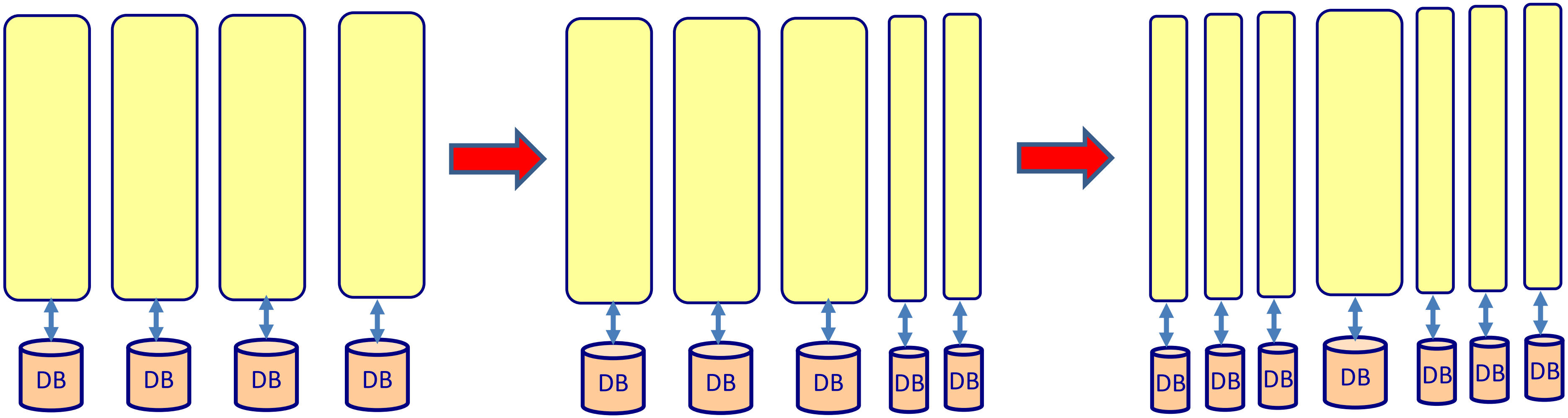
- Replicability or changeability
  - The microservice is easy detachable from the overall system
  - What functionality might evolve in the future?
- Coupling and cohesion
  - Avoid chatty services
  - Too many synchronous request
  - Transaction boundaries within one service

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



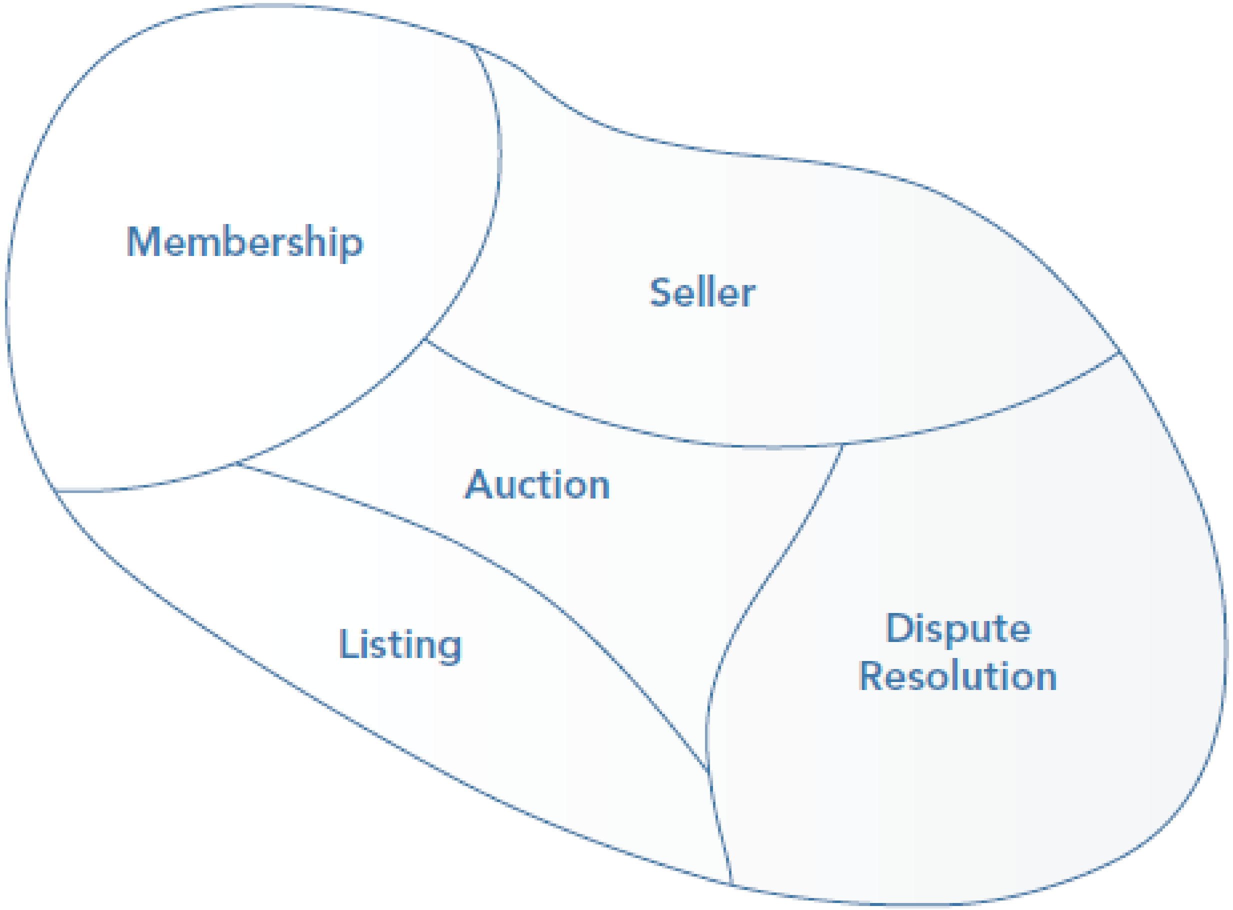
Domains

- 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

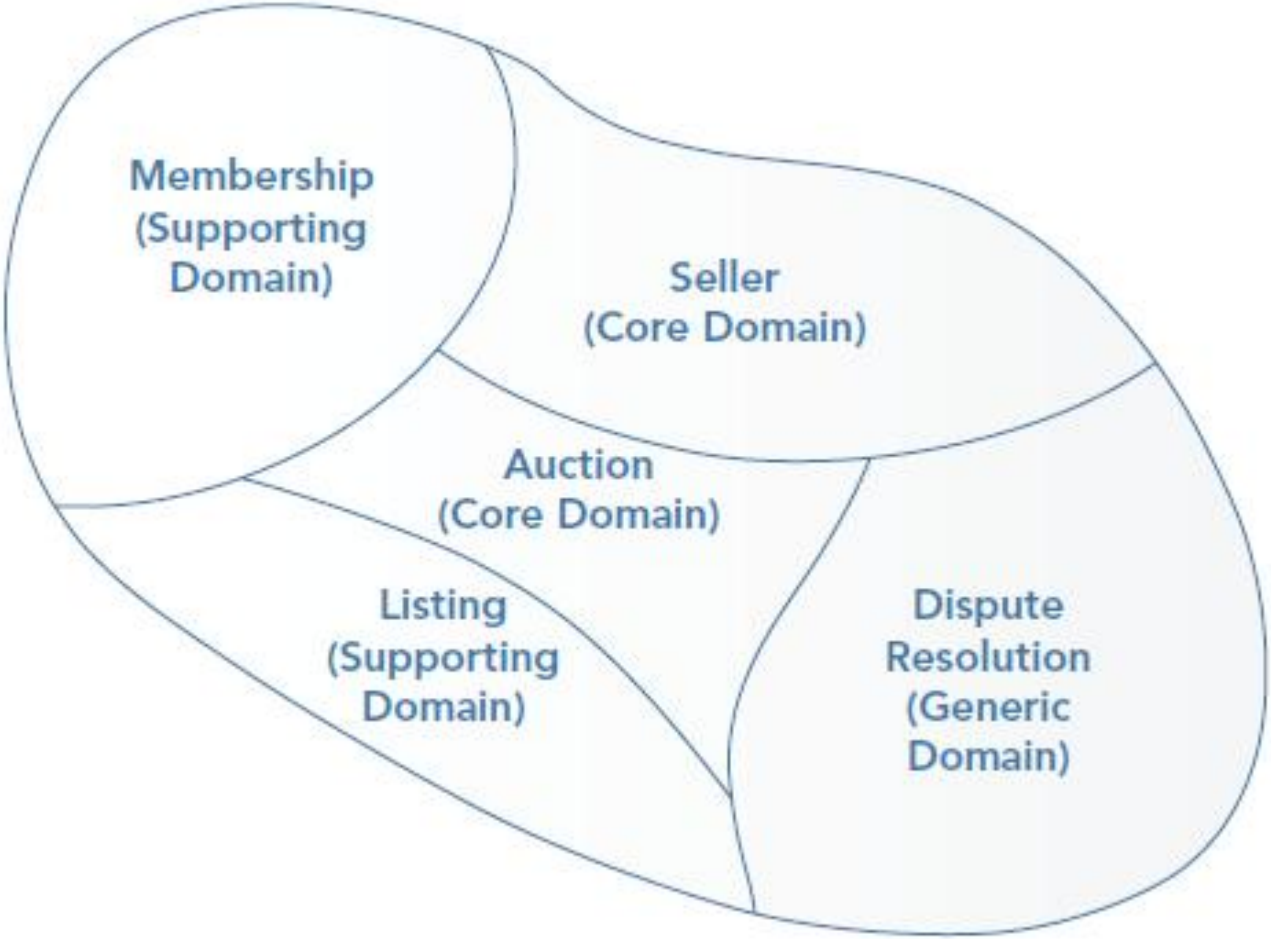
Distilling the domain

- The large domain of online auction

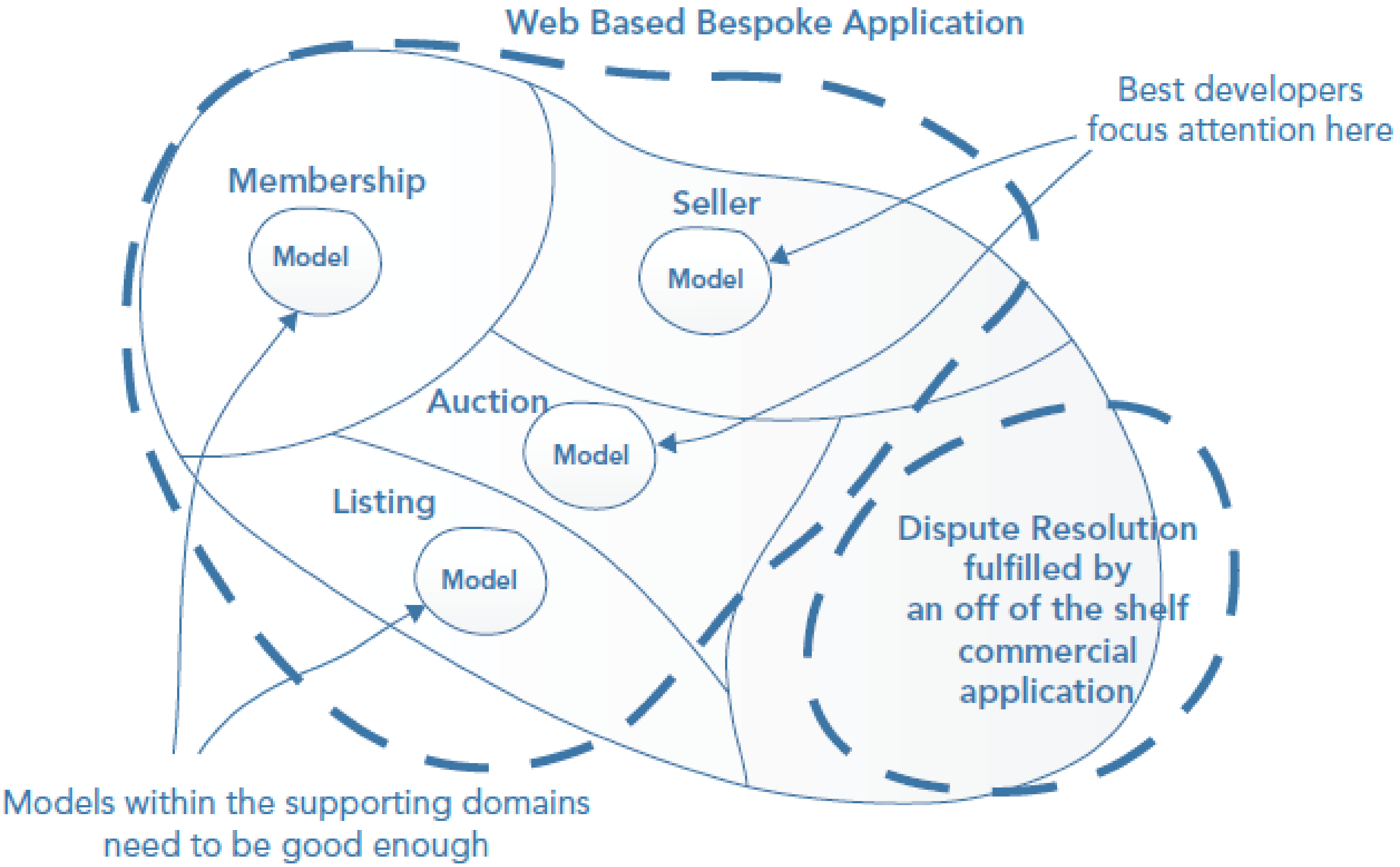
Find the subdomains



Identify the core domain


Membership (Supporting Domain)
Seller (Core Domain)
Auction (Core Domain)
Listing (Supporting Domain)
Dispute Resolution (Generic 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

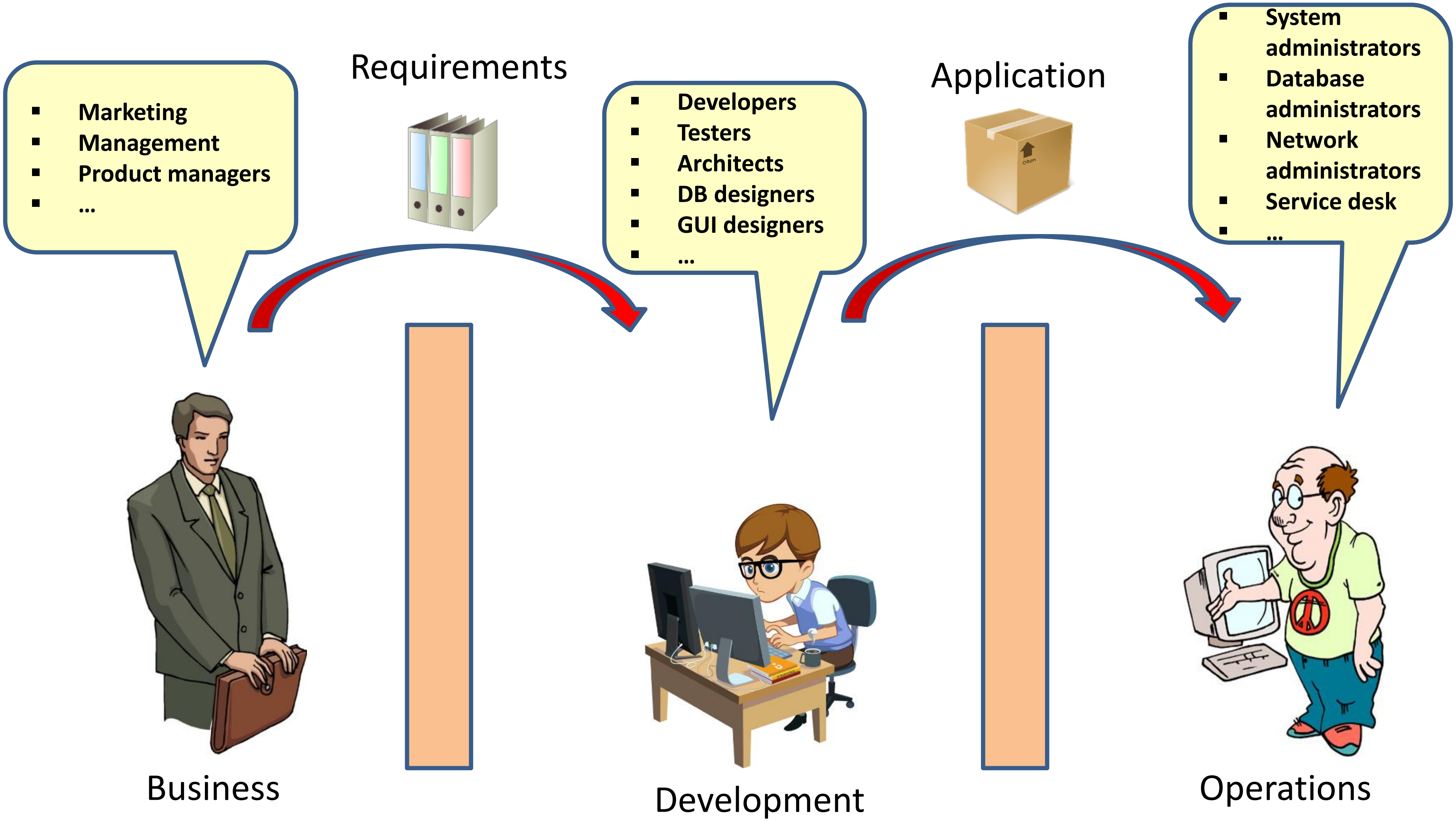
Main point

- An ideal architecture does not exist. An microservice architecture has its own advantages and disadvantages. It is almost impossible to transform every application into microservices.

- Water the root and enjoy the fruit. Problems are hard to solve at the level of the problem. It is much easier to solve problems at its root.

MICROSERVICES IN THE ORGANIZATION

Traditional software development



- Marketing
- Management
- Product managers
- …

Requirements

- Developers
- Testers
- Architects
- DB designers
- GUI designers
- …

Application

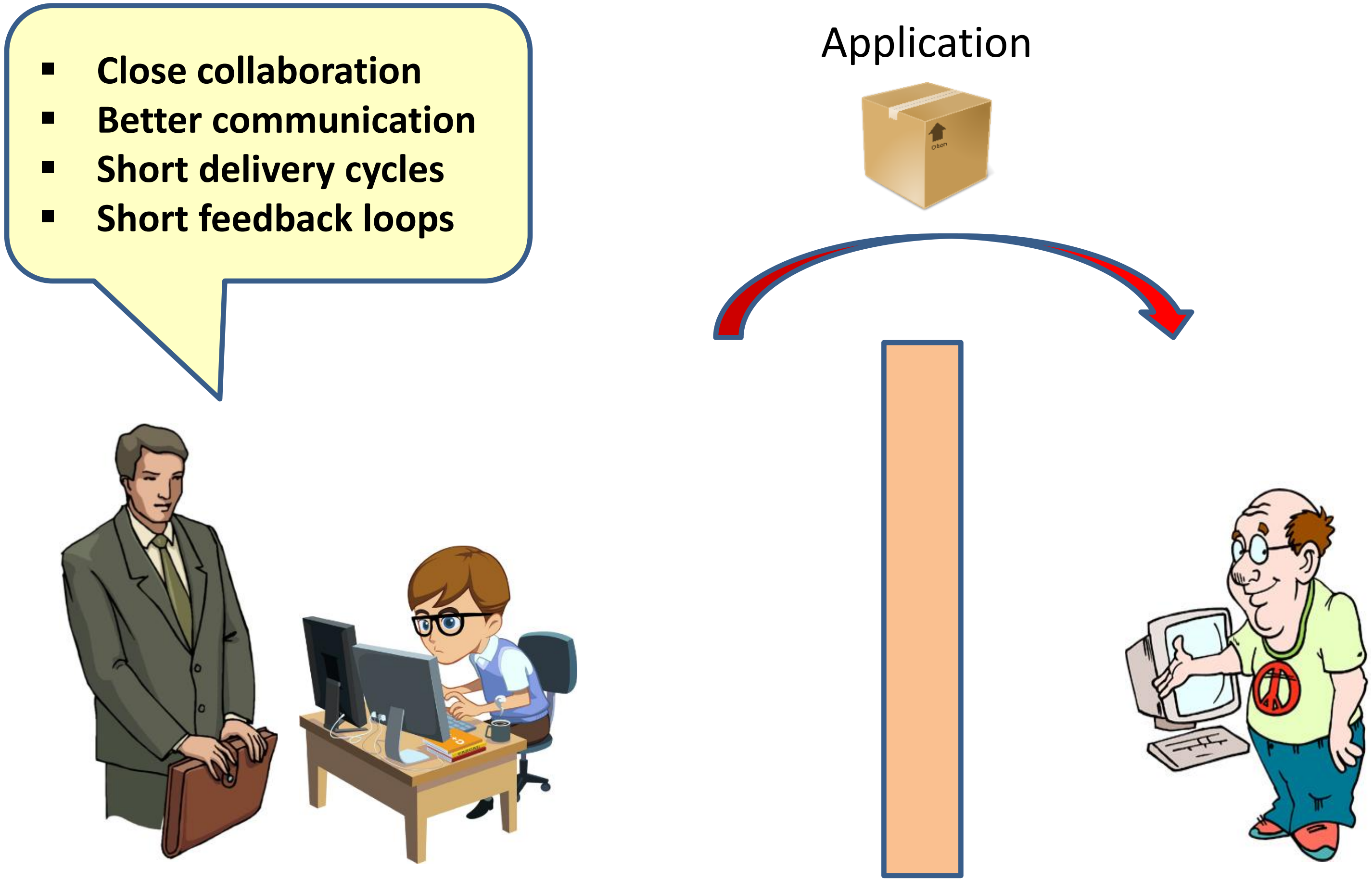
- System administrators
- Database administrators
- Network administrators
- Service desk
- …

Business

Development

Operations

Agile software development: Scrum



- **Close collaboration**
- **Better communication**
- **Short delivery cycles**
- **Short feedback loops**

Application

Product owner (business)
and developers in one team

Operations

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

Product owner (business) and developers in one team

Operations

Conways law

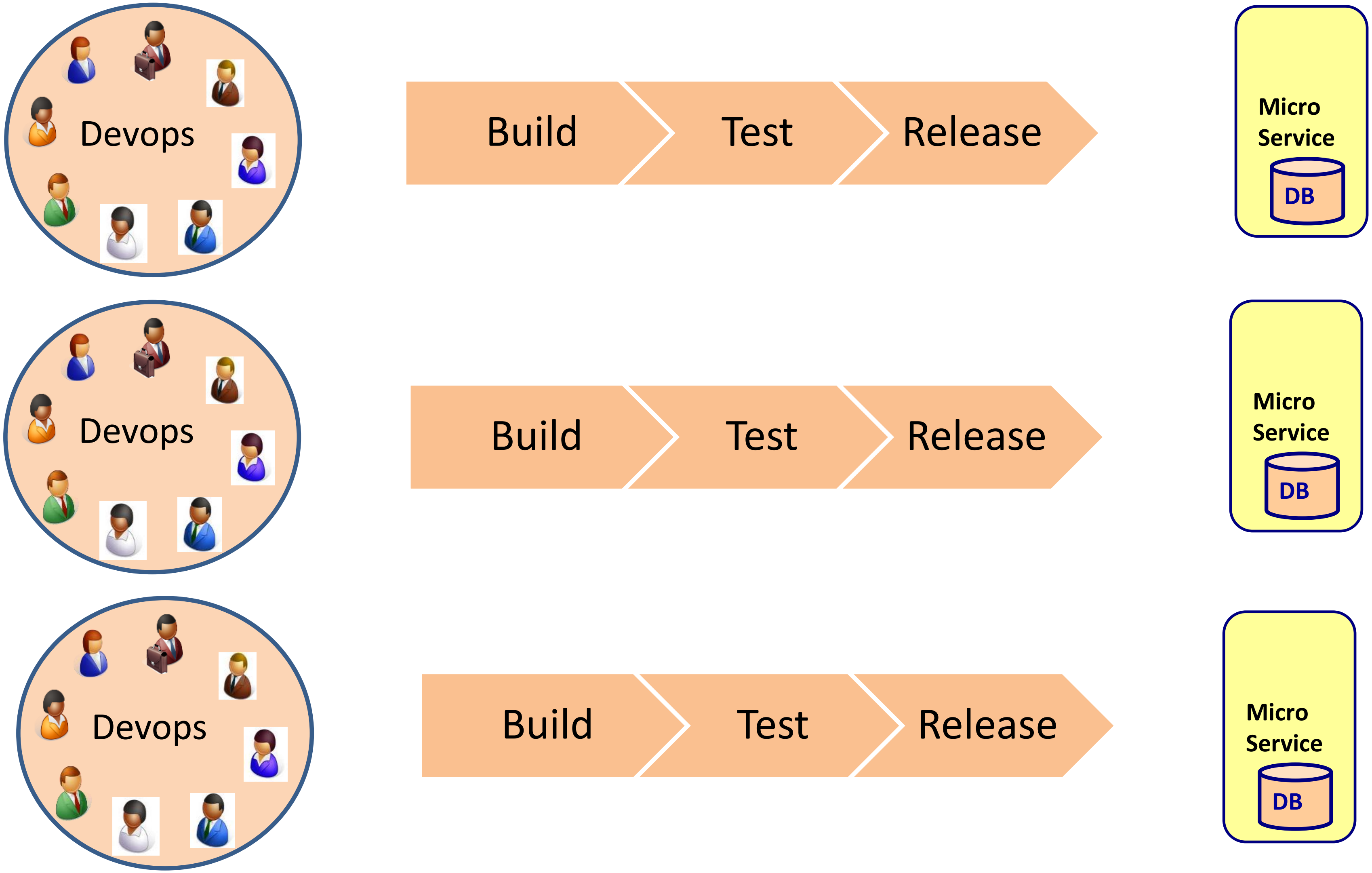
“organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations ”

—Melvin Conway

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

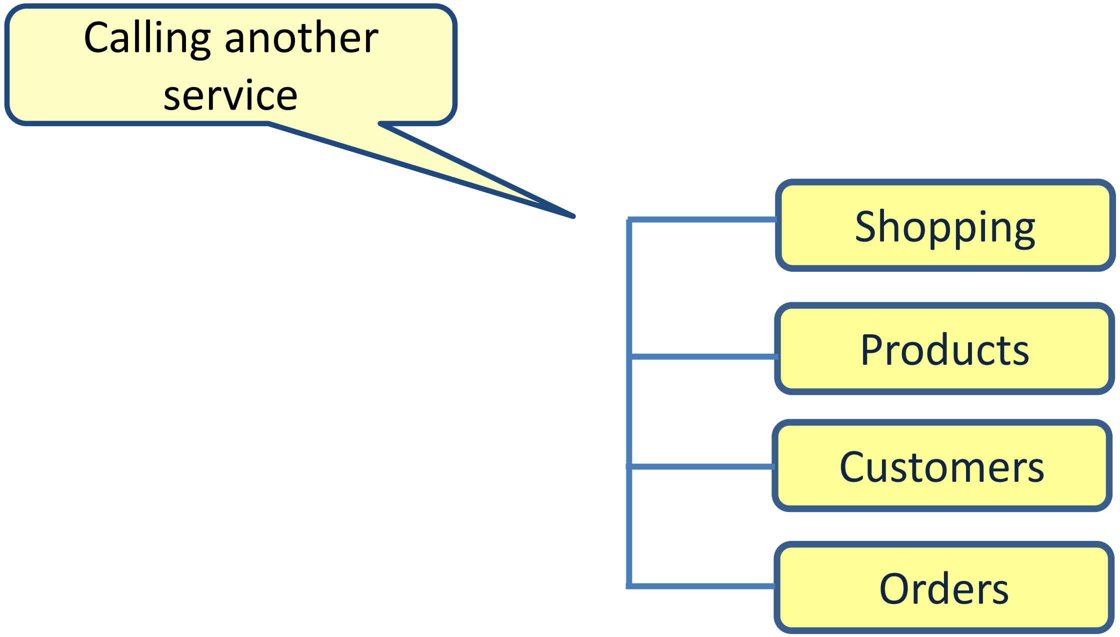
—Eric S Raymond

Microservice organization

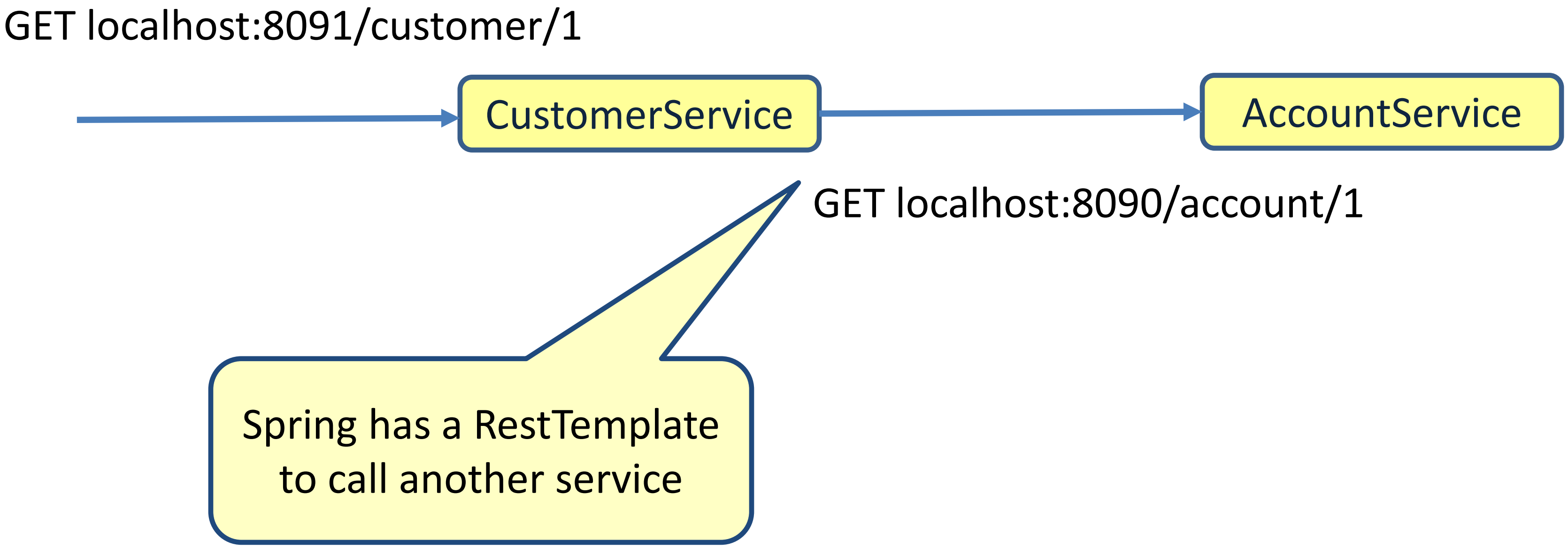


CALLING ANOTHER MICROSERVICE: FEIGN

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

```
@Component
public class RestClient {
  private RestTemplate  restTemplate = new RestTemplate();

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

RestTemplate does not work automatically with registry, load balancer, etc.

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

Feign

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

AuthorService

```
@RestController
public class AuthorController {
  @RequestMapping("/authors/{isbn}")
  public Author getAuthor(@PathVariable("isbn") String isbn) {
    return new Author("Joanne", "Rowling");
  }
}
```

```
public record Author (String firstname, String lastname){
}
```

```
@SpringBootApplication
public class AuthorServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(AuthorServiceApplication.class, args
  }
}
```

application.yml

```
spring:
  application:
    name: Authorservice

server:
  port: 8093
```

BookService

```java
@SpringBootApplication
@EnableFeignClients
public class BookServiceApplication {
   public static void main(String[] args) {
      SpringApplication.run(BookServiceApplication.class, args);
   }
}
```

Use Feign

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

application.yml

```yaml
spring:
  application:
    name: Bookservice

server:
  port: 8092
```

BookService: the controller

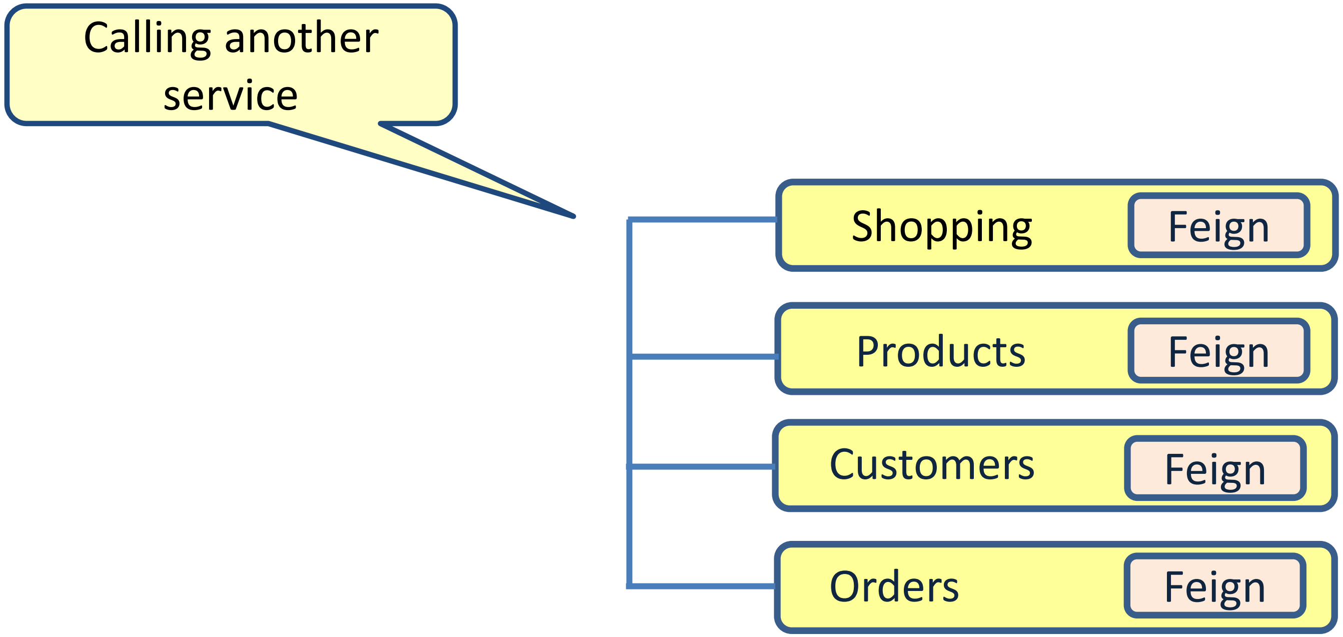
```
@RestController
public class BookController {
  @Autowired
  AuthorFeignClient authorClient;

  @RequestMapping("/books/{isbn}")
  public Book getName(@PathVariable("isbn") String isbn) {
    Author author = authorClient.getAuthor(isbn);
    return new Book("isbn", "1000.00", author.firstname()+" "+author.lastname());
  }

  @FeignClient(name = "author-service", url = "http://localhost:8093")
  interface AuthorFeignClient {
    @RequestMapping("/authors/{isbn}")
    public Author getAuthor(@PathVariable("isbn") String isbn);
  }
}
```

Autowire the client

Remote REST call

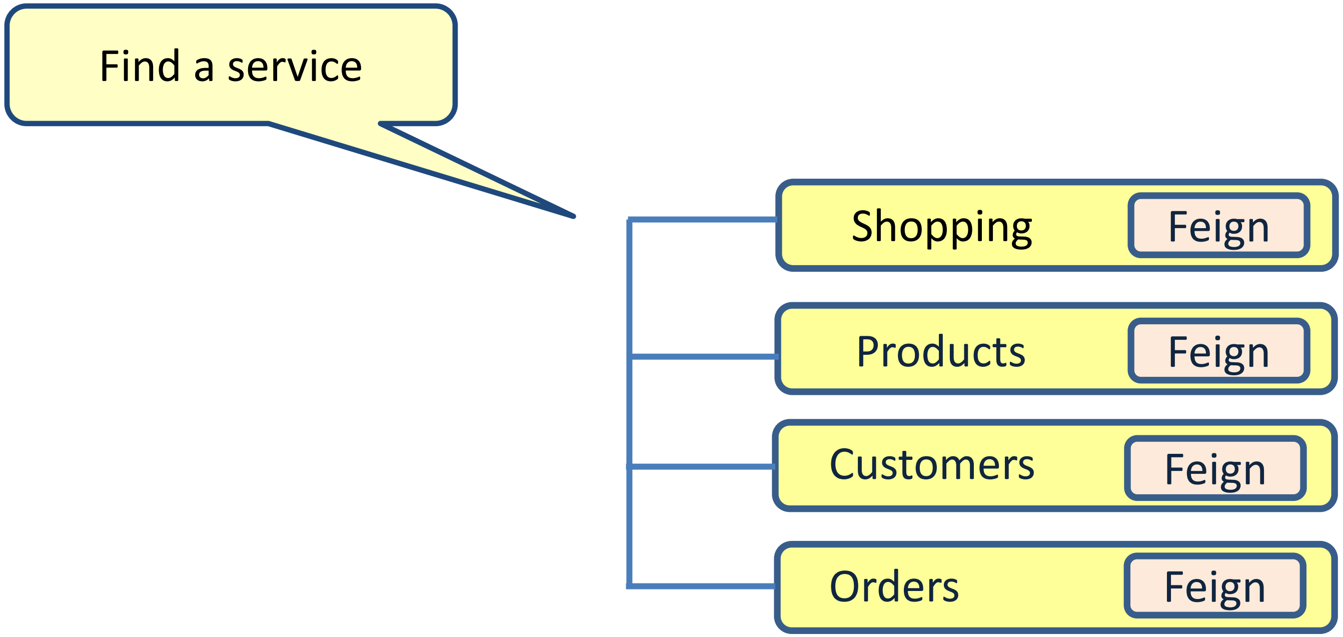
Declare the interface, Spring creates the implementation

Implementing microservices

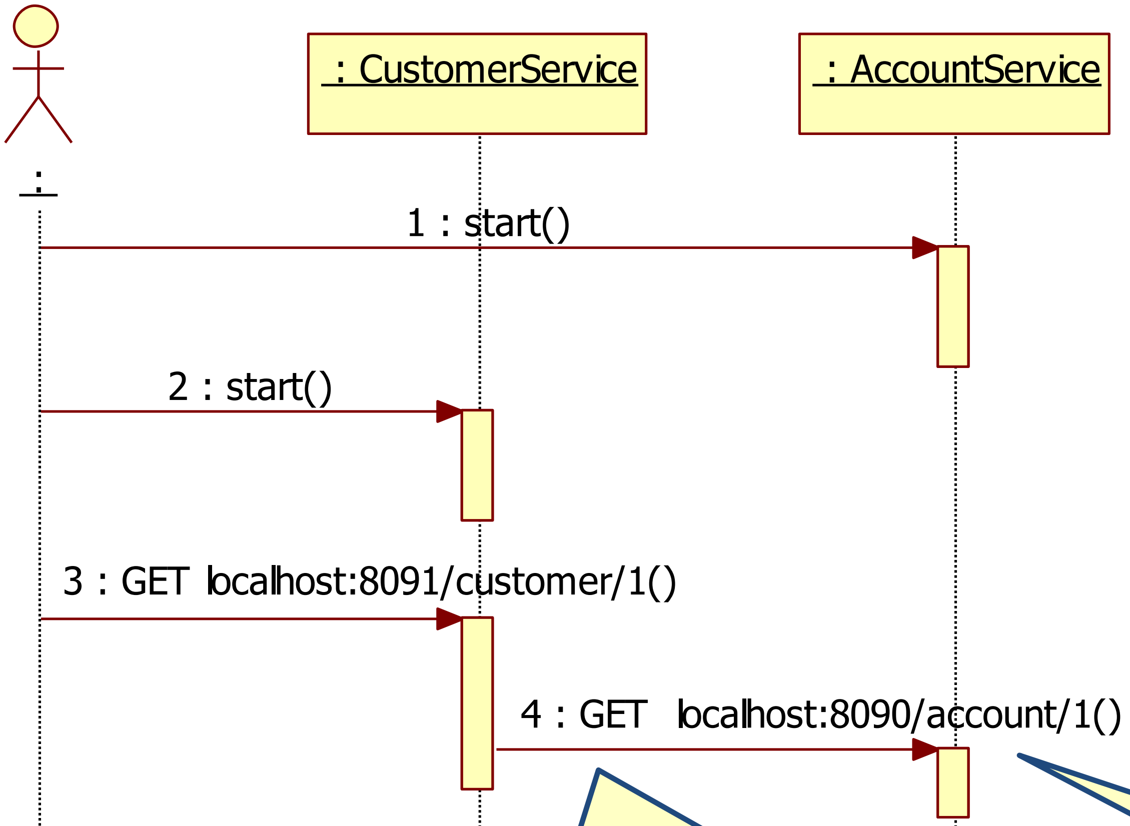


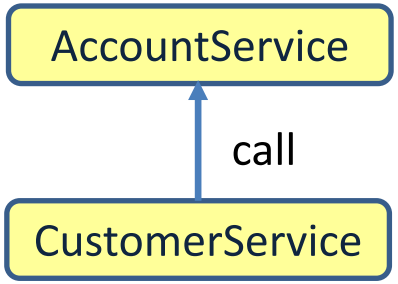
SERVICE REGISTRY: CONSUL

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()



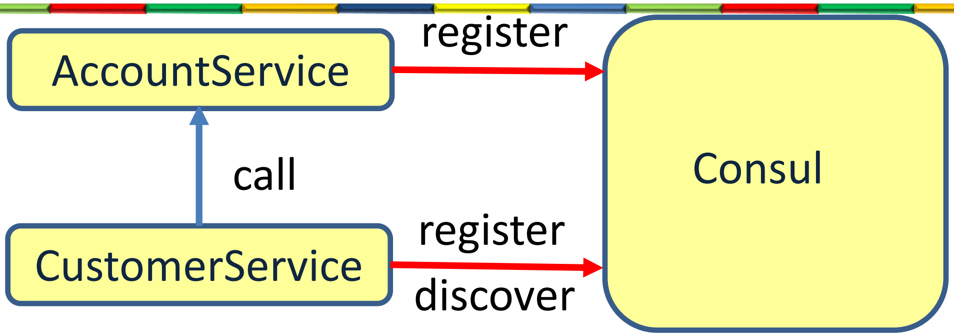
Address is hard coded

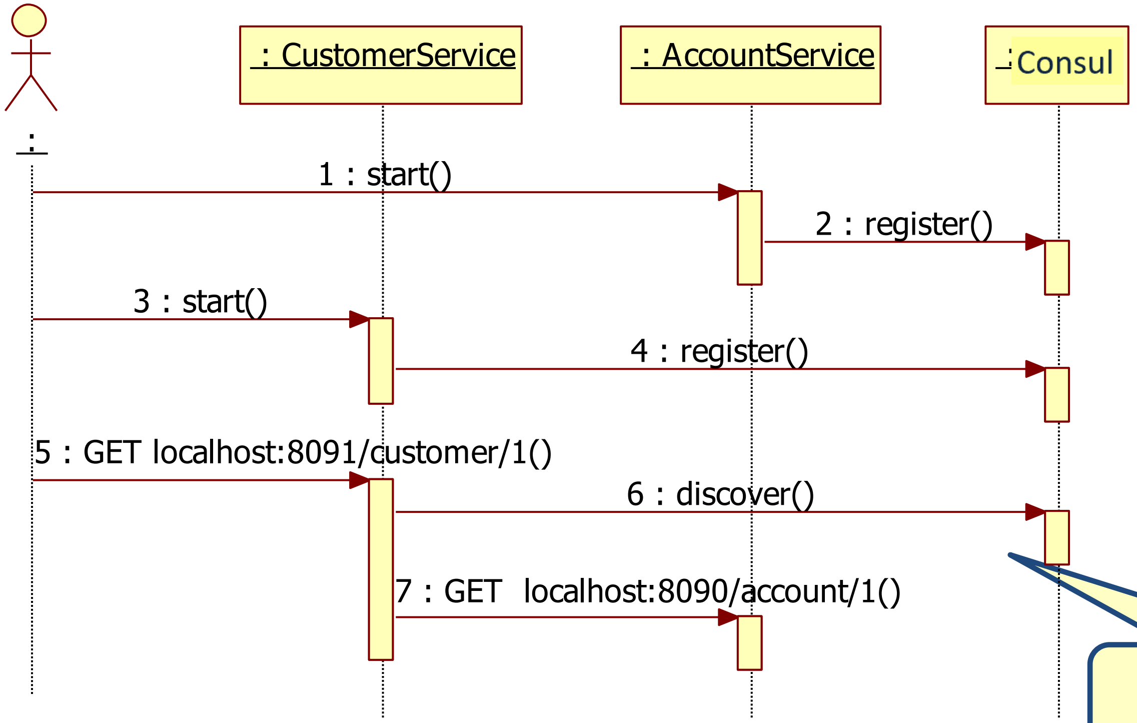
If I change the URL of the AccountService, I need to change the configuration of the CustomerService

Service Registry

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

Using Eureka





Address is fetched from the registry

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.

AccountService

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

The service will register itself in the registry

```yaml
spring:
  application:
    name: Accountservice
  cloud:
    consul:
      host: localhost
      port: 8500

server:
  port: 8091
```

application.yml

AccountService

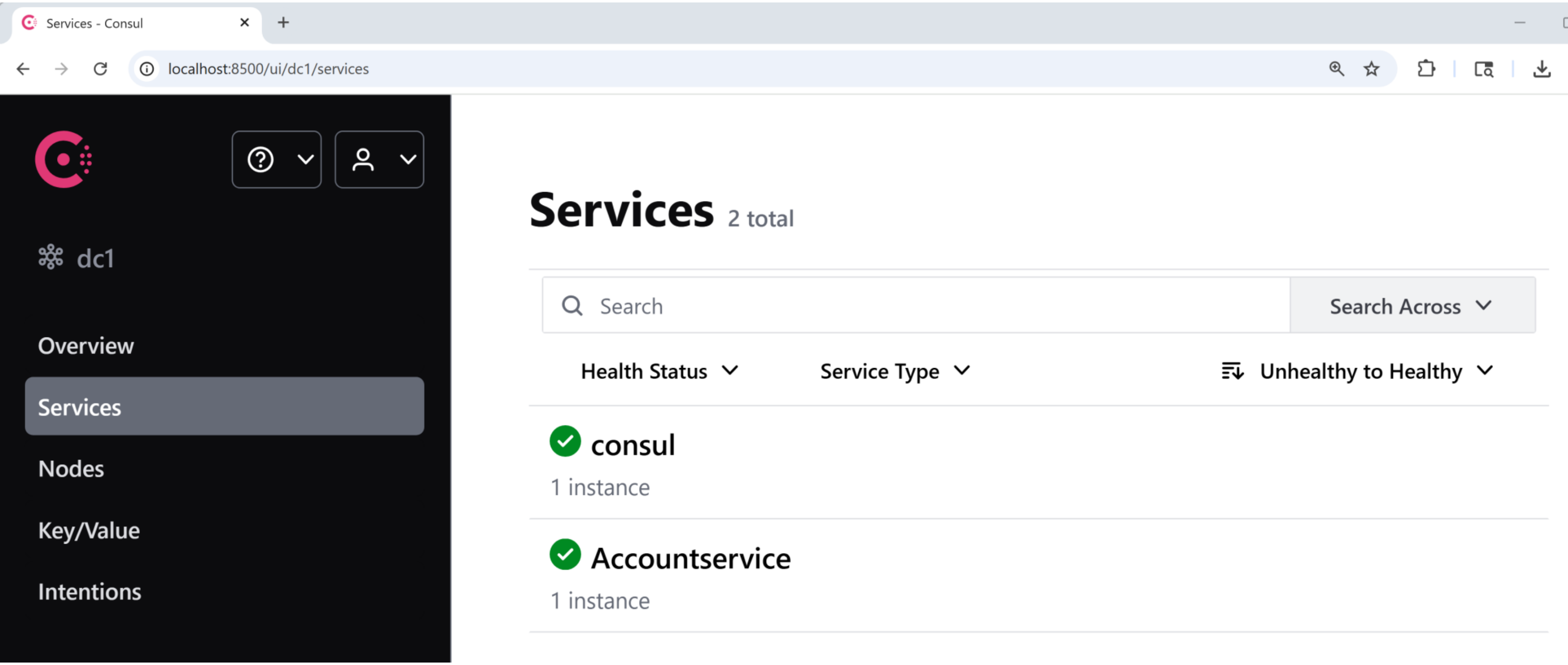
```
@RestController
public class AccountController {
  @RequestMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    return new Account("1234", "1000.00");
  }
}
```

```
public record Account (String accountNumber, String balance){
}
```

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-consul-discovery</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Needed so that Consul can call the /actuator/health actuator

Running the AccountService



Services 2 total

consul
1 instance

Accountservice
1 instance

CustomerService

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

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

Use Feign and the Registry

```xml
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-consul-discovery</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<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 Customer getName(@PathVariable("customerid") String customerId) {
        Account account = accountClient.getName(customerId);
        return new Customer("Frank Brown", account.accountNumber(), account.balance());
    }

    @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 the Registry

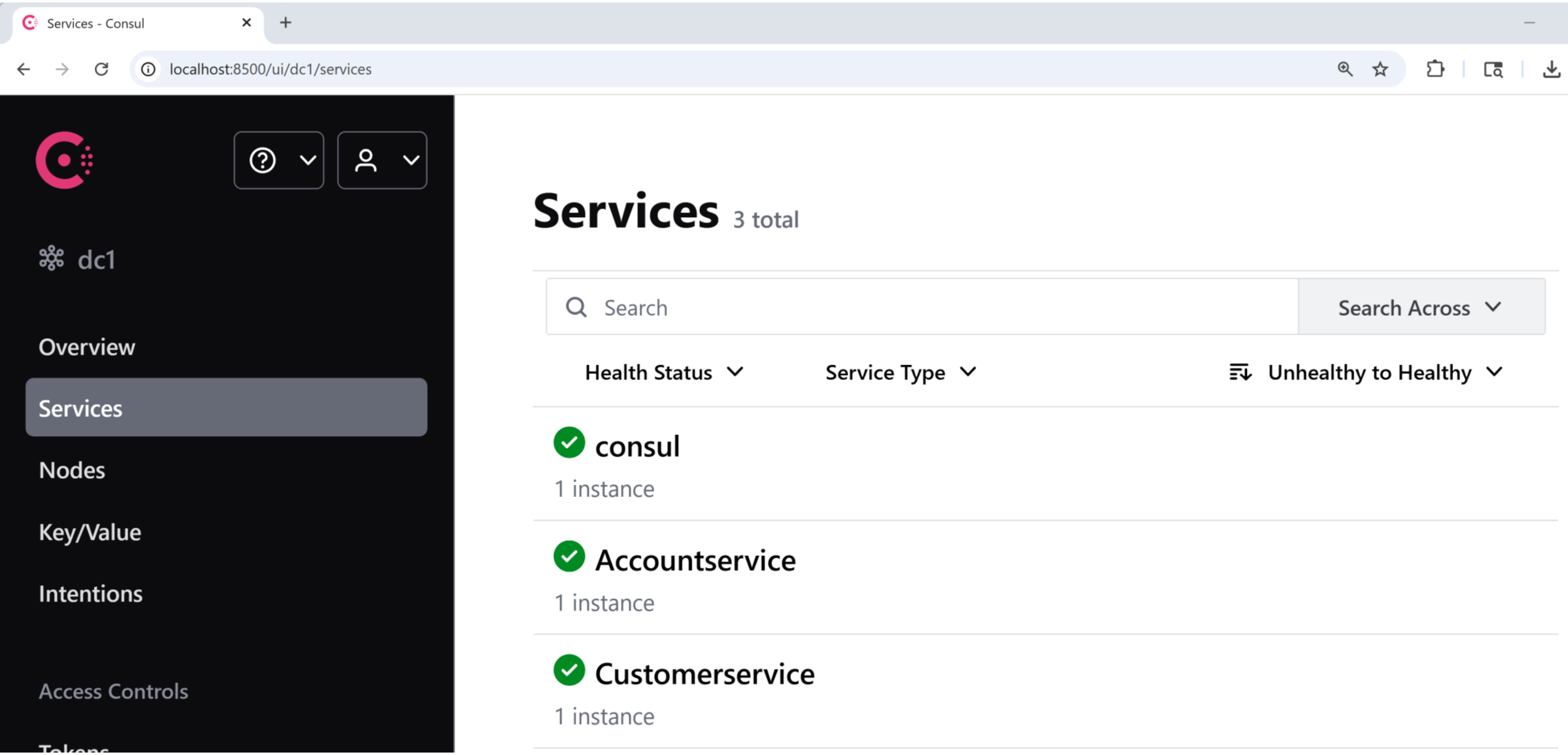
CustomerService configuration

application.yml

```yaml
spring:
  application:
    name: Customerservice
  cloud:
    consul:
      host: localhost
      port: 8500
      discovery:
        enabled: true
        prefer-ip-address: true
        instance-id: ${spring.application.name}:${random.value}


server:
  port: 8090
```

By default, Consul will do a health check every 10 seconds

Running the CustomerService



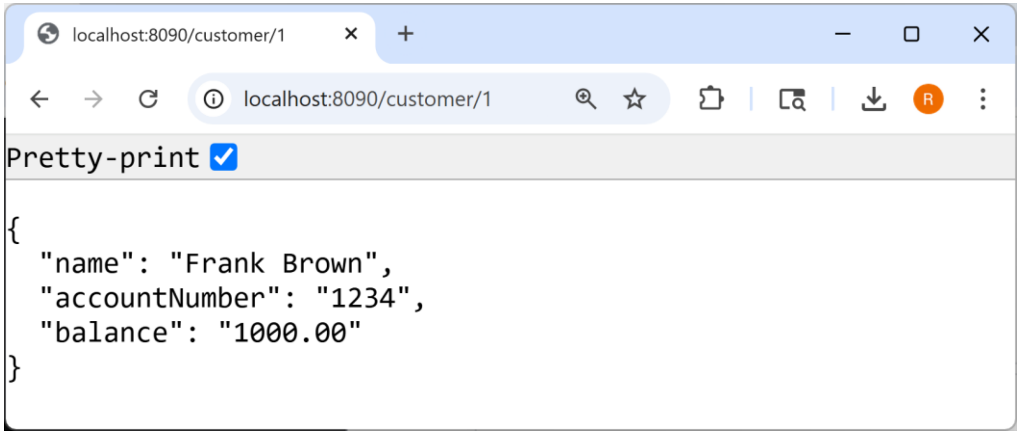
Services 3 total

✓ consul
1 instance

✓ Accountservice
1 instance

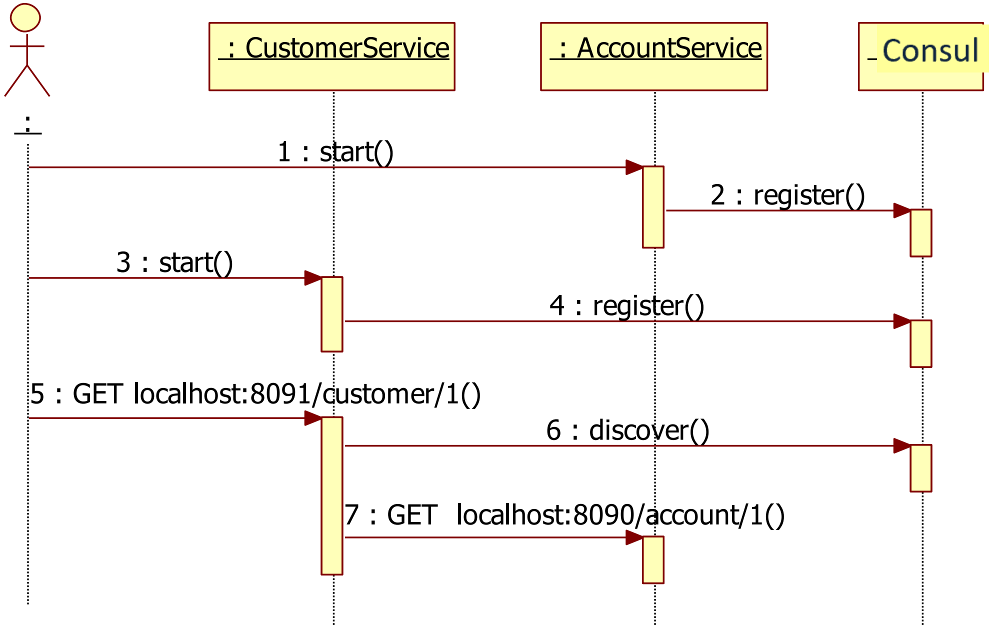
✓ Customerservice
1 instance

Calling the CustomerService



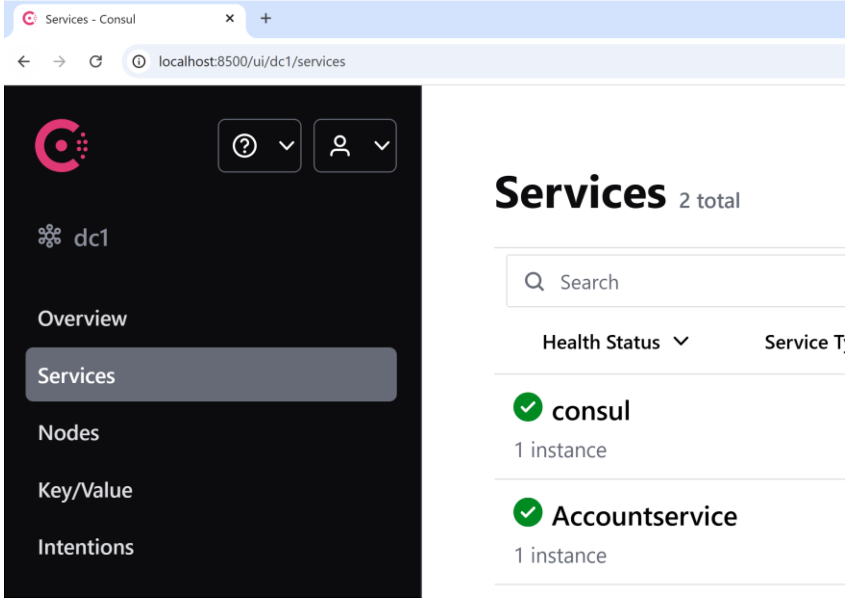
Pretty-print ☑

```
{
  "name": "Frank Brown",
  "accountNumber": "1234",
  "balance": "1000.00"
}
```



Stopping the CustomerService

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

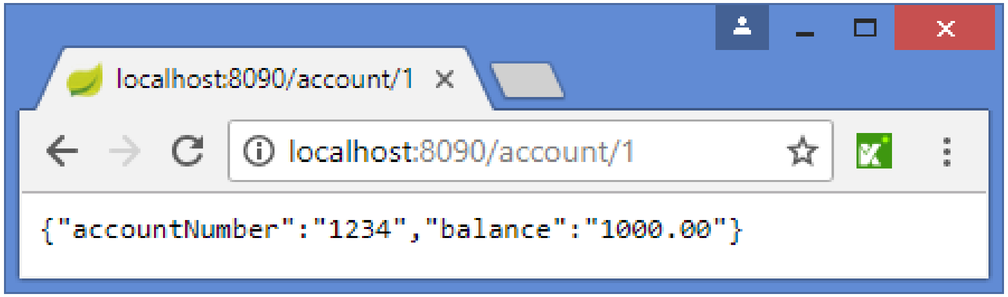


Services 2 total

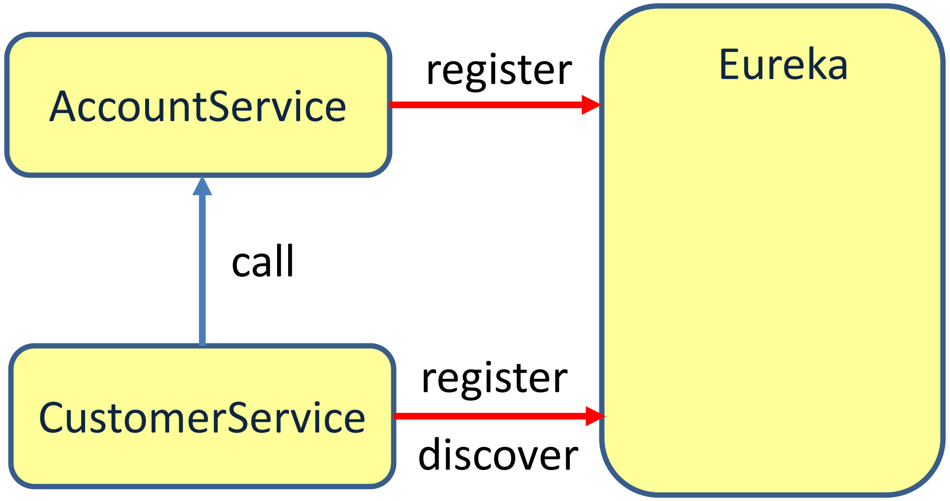
consul
1 instance

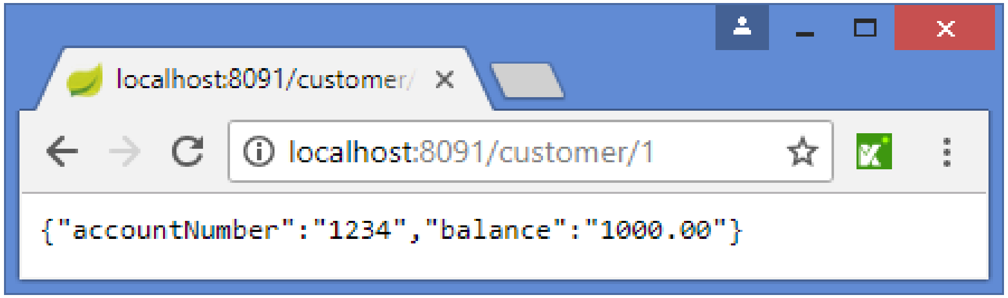
Accountservice
1 instance

Using Eureka



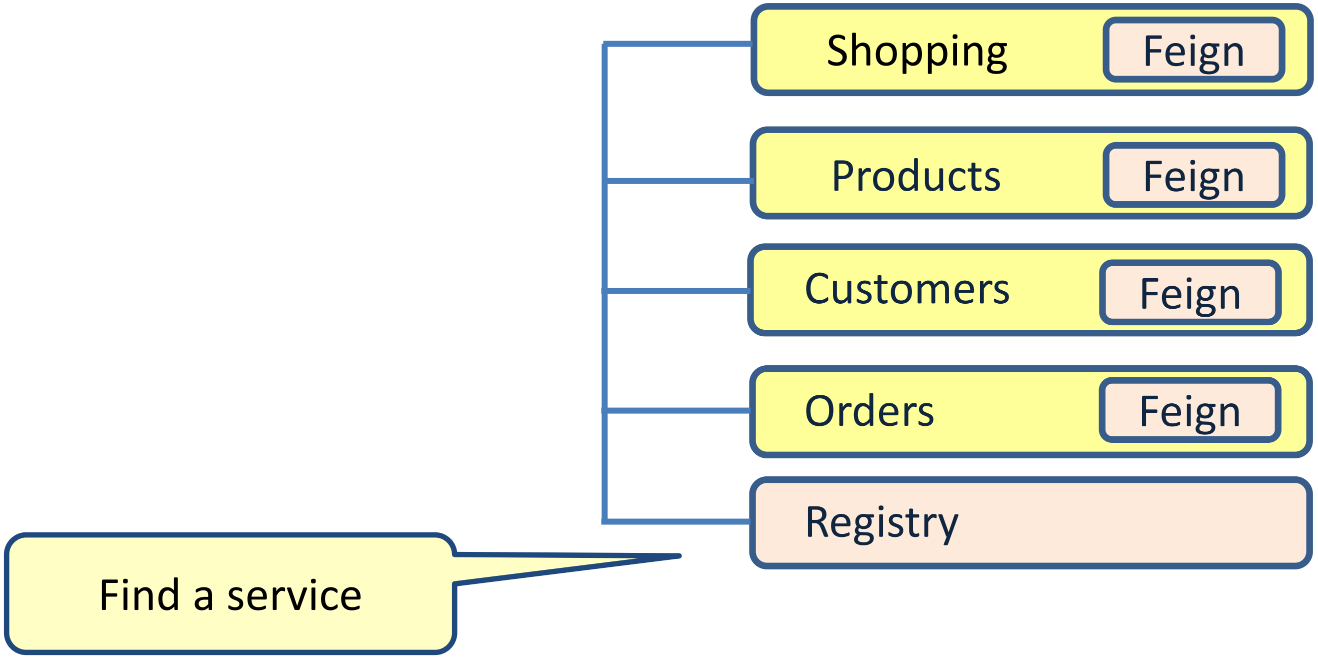
```
{"accountNumber":"1234","balance":"1000.00"}
```





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

Implementing microservices



Find a service

Challenges of a microservice architecture

Challenge	Solution
Complex communication	Registry
Performance	
Resilience	Registry
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	

Main point

- To keep microservices loosely coupled a central registry is needed so that microservices can find each other.

- Pure consciousness is the central registry of all intelligence who is available to every human being.

Connecting the parts of knowledge with the wholeness of knowledge

1. A microservice is an autonomous application owned by 1 team.
2. A microservice architecture is a distributed architecture which is complex by nature.

3. **Transcendental consciousness** is the source from which the whole complex world is created.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that all distributed components in creation are just expressions of ones own Self.