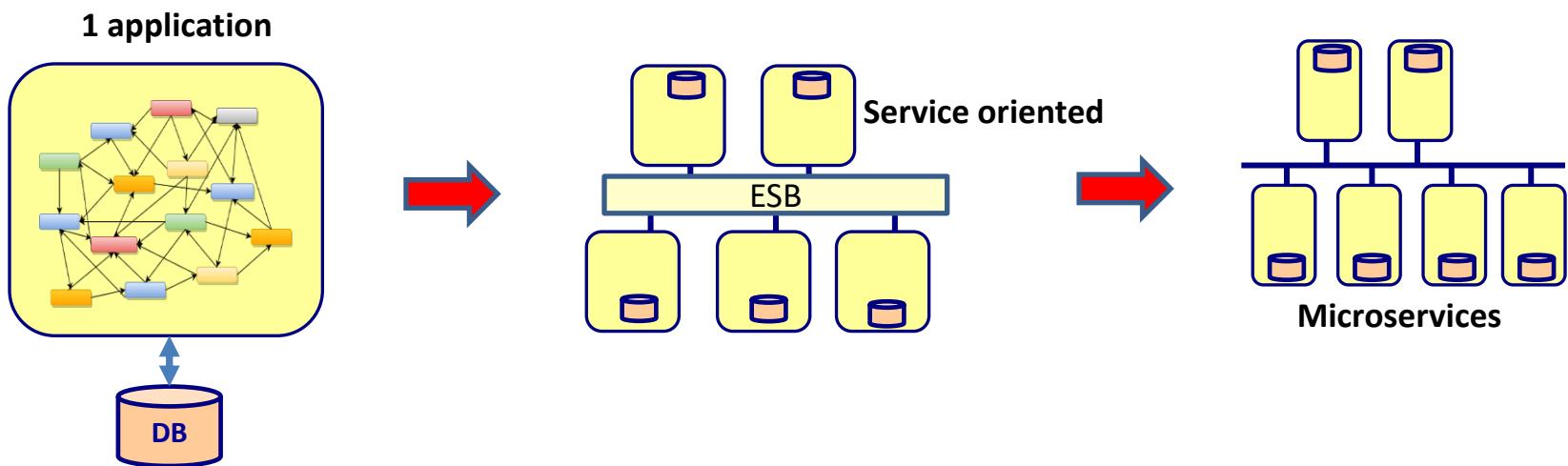


SOFTWARE ARCHITECTURE COURSE OVERVIEW



CS 590 Software Architecture

- Architecture styles
- Architecture patterns
- Architecture principles and best practices



Course agenda

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
October 27 Lesson 1 Software architecture introduction	October 28 Lesson 2 Domain Driven Design	October 29 Lesson 3 Databases	October 30 Lesson 4 Component based design	October 31 Lesson 5 SOA & integration patterns	November 1 Lesson 6 Microservices 1	November 2
November 3 Lesson 7 Microservices 2	November 4 Review	November 5 Midterm Exam	November 6 Lesson 8 Microservices 3	November 7 Lesson 9 Microservices 4	November 8 Lesson 10 Microservices 5	November 9
November 10 Lesson 11 Kafka	November 11 Lesson 12 Kafka + Stream based architecture	November 12 Lesson 13 Finding the right architecture	November 13 Review	November 14 Final Exam	November 15 Project	November 16
November 17 Project	November 18 Project	November 19 Project	November 20 Project Presentations			

LESSON 1

SOFTWARE ARCHITECTURE

INTRODUCTION



Why architecture?



More complexity asks for:

- More abstraction and decomposition
- More principles and guidelines
- More communication
- More processes
- More powerful tooling

More complexity asks for more architecture



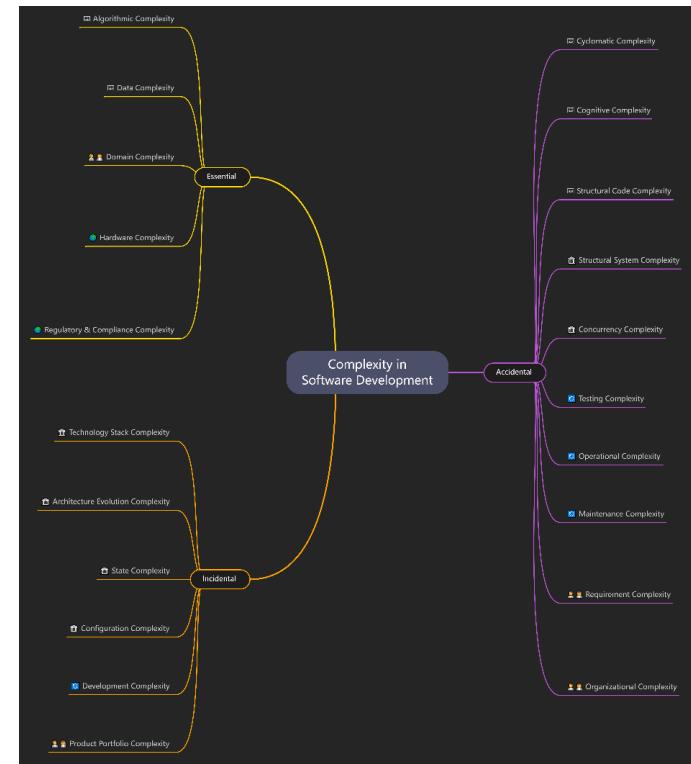
Software complexity

- Essential complexity
 - Inherent to the nature of the problem itself or its environment
- Incidental complexity
 - Accumulated over time as software, project or organization evolves
- Accidental complexity
 - Introduced by human decisions



Software complexity

- Essential
- Incidental
- Accidental

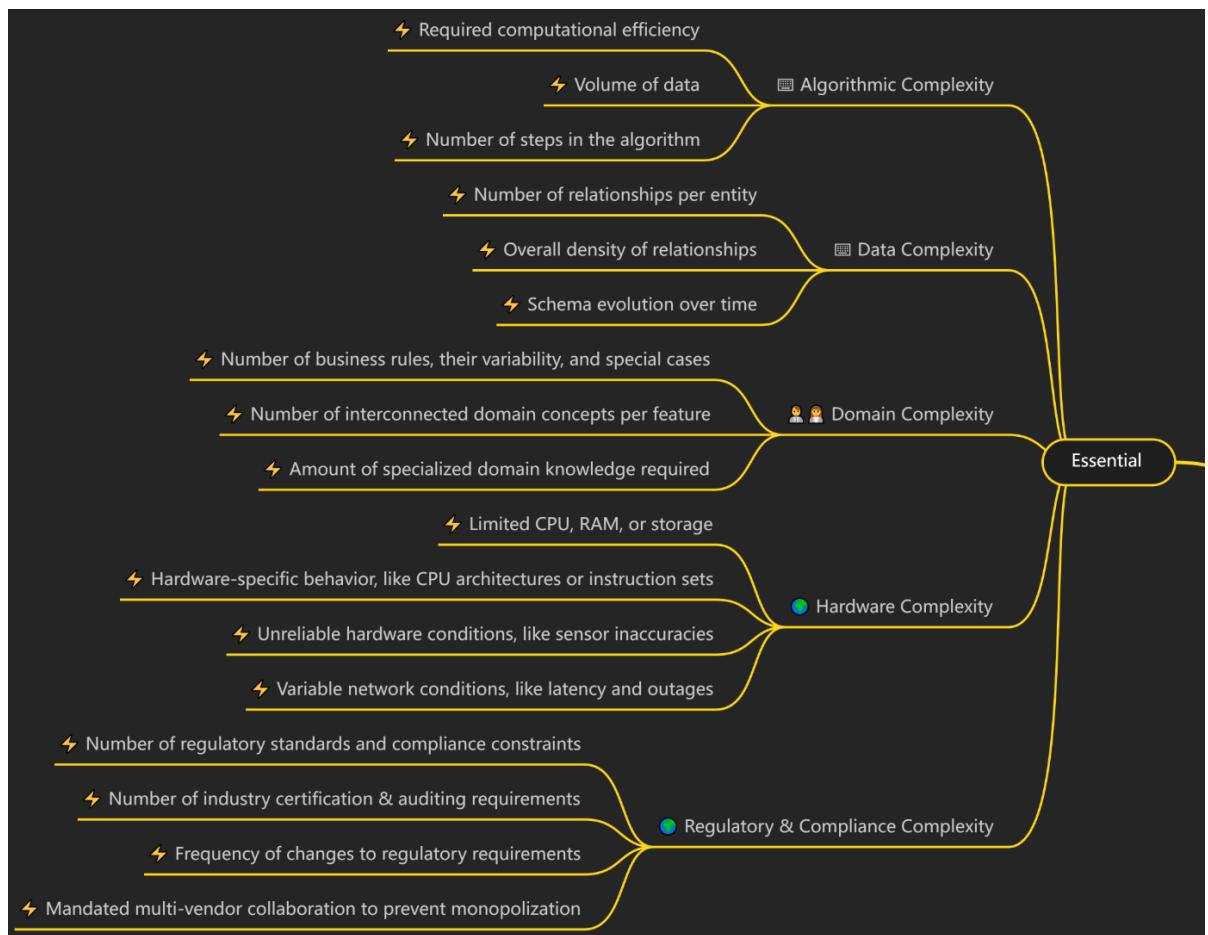


- <https://github.com/plainionist/AboutCleanCode/tree/main/Complexity>



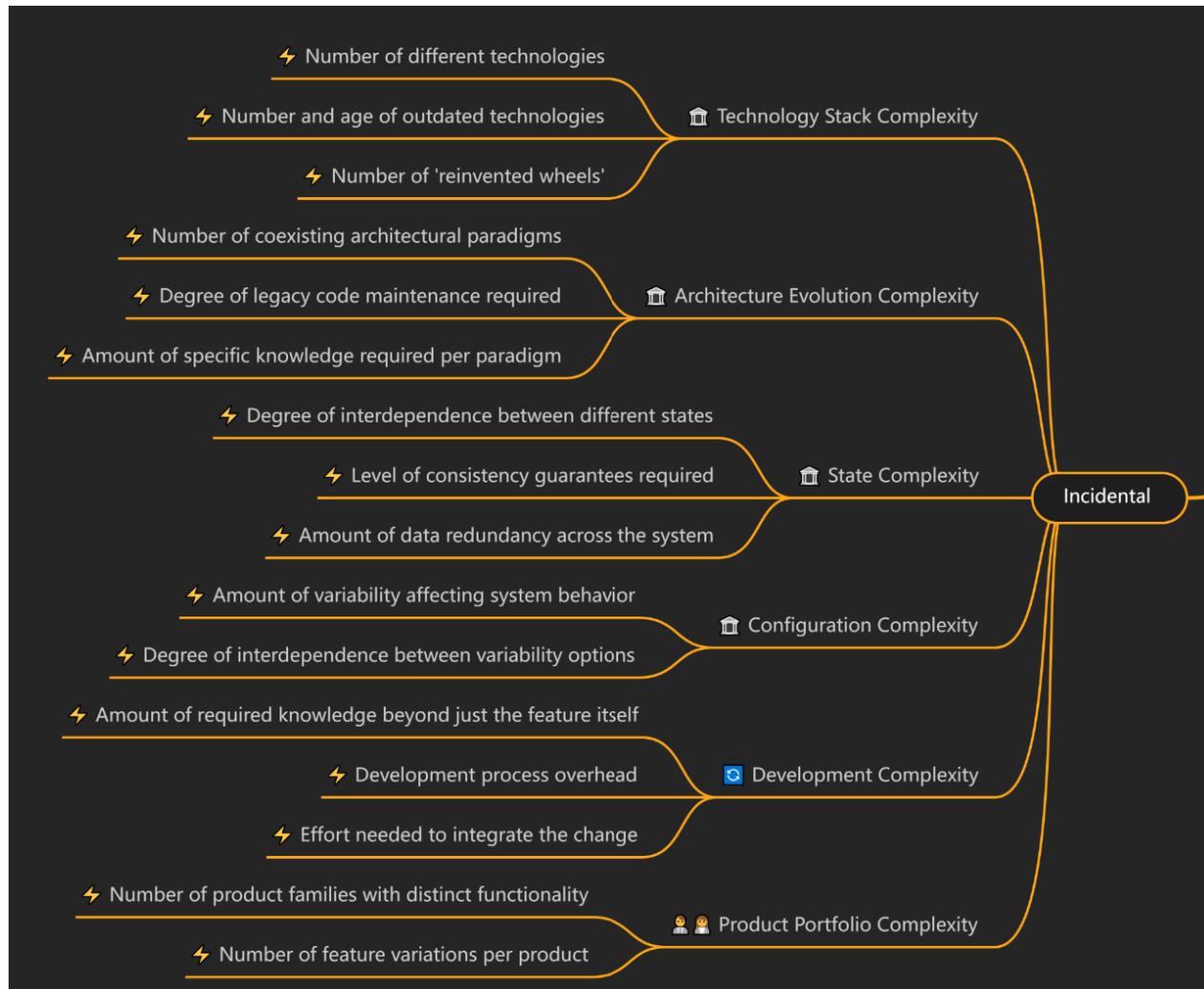
Essential complexity

- Inherent to the nature of the problem itself or its environment



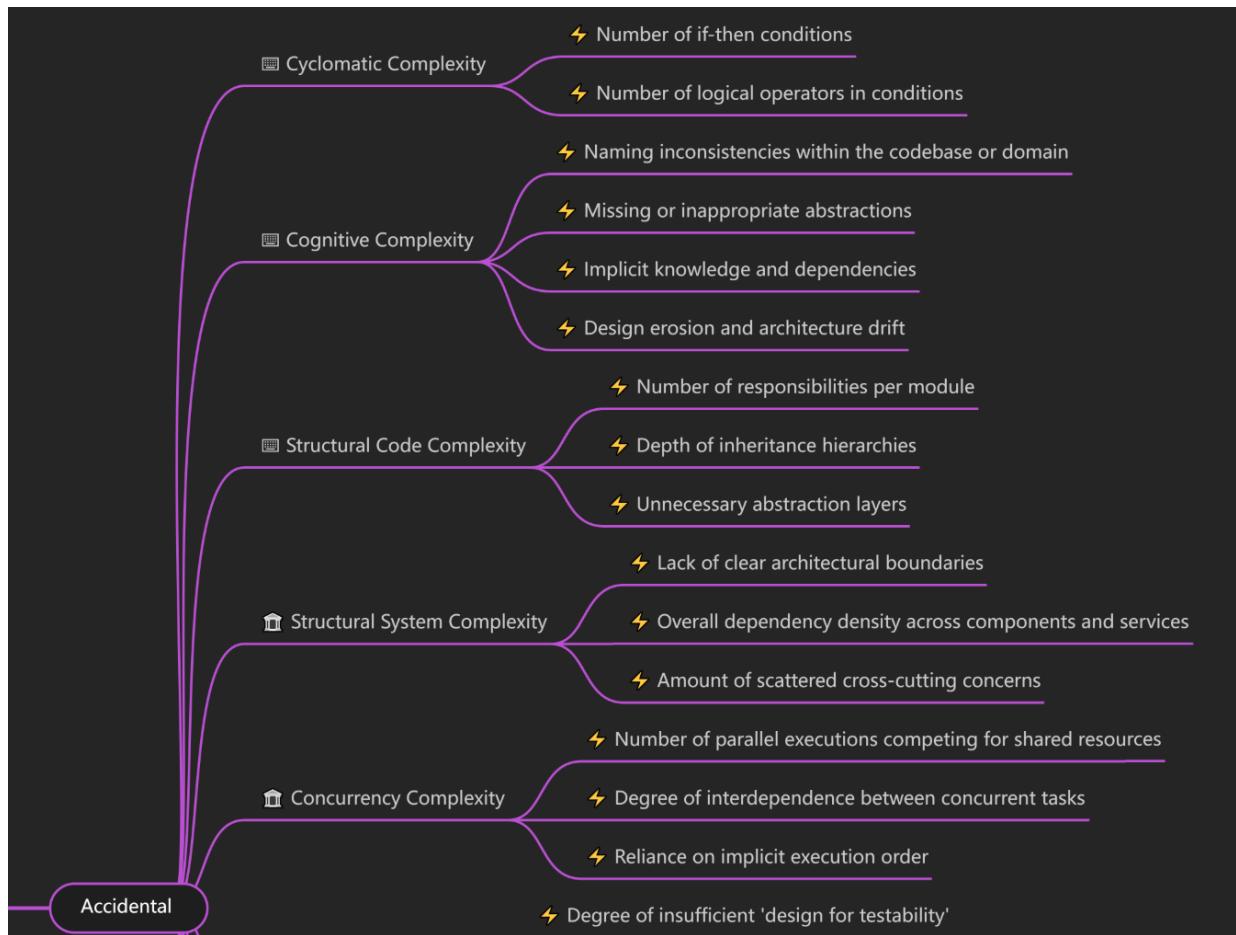
Incidental complexity

- Introduced by human decisions



Accidental complexity

- Accumulated over time as software, project or organization evolves



Accidental complexity

- Accumulated over time as software, project or organization evolves



Software complexity

- Essential complexity
 - Inherent to the nature of the problem itself or its environment
 - This complexity cannot be reduced
- Incidental complexity
 - Accumulated over time as software, project or organization evolves
 - Can be reduced with good software engineering principles
- Accidental complexity
 - Introduced by human decisions
 - Can be reduced with good software engineering principles



Why architecture?

- Winchester “mystery” house
- 38 years of construction work— 147 builders 0 architects
- 160 rooms— 40 bedrooms, 6 litchens, 2 basements, 950 doors
- No architecture description
- 65 doors that don’t go anywhere, 13 stairs that don’t go anywhere, 24 skylights where you cannot see the sky



What is software architecture?

- The fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution
 - ANSI/IEEE std 1471-2000



What is software architecture?

The things that are difficult to change

A blueprint

Technical leadership

Standards and guidelines

Building blocks

Technical direction

Systems, subsystems, interactions and interfaces

Satisfying non-functional requirements/quality attributes

The big picture

The skeleton/backbone of the product

The system as a whole

Structure (components and interactions)

The things that are expensive to change



What is software architecture?

The important stuff

Whatever that might be



Defining the architecture

- Define components
 - Define component interfaces
 - Define platform and language(s)
 - Define architectuur styles
 - Define architectuur patterns
 - Define layers and packages
 - Define presentation architecture
 - Define persistency architecture
 - Define security architecture
 - Define transaction architecture
 - Define distribution architecture
 - Define integration architecture
 - Define the deployment architecture
 - Define the clustering architecture
 - Define the hardware
 - Define tools to use
-
- Decide on solutions for
 - Logging
 - Error management
 - Error detection
 - Error reporting
 - Fault tolerance
 - Event management
 - File handling
 - Printing
 - Reporting
 - Resource management
 - Internationalization
 - Licence management
 - Debugging
 - ...



Different kinds of architecture

- Infrastructure
- Security
- Technical
- Solution
- Network
- Data
- Hardware
- Enterprise
- Application
- System
- Integration
- IT
- Database
- Information
- Process
- Business
- Software



Different kinds of architecture

Enterprise Architecture

Define enterprise wide aspects like people, processes, applications, infrastructure, etc.

Business Architecture

Define the processes, strategies and goals of the business.

Information Architecture

Define the information and services needed to support the business

Application Architecture

Define the structure and dynamics of software applications.

Infrastructure Architecture

Define the hardware and software infrastructure

*Software
Architecture*



City planning



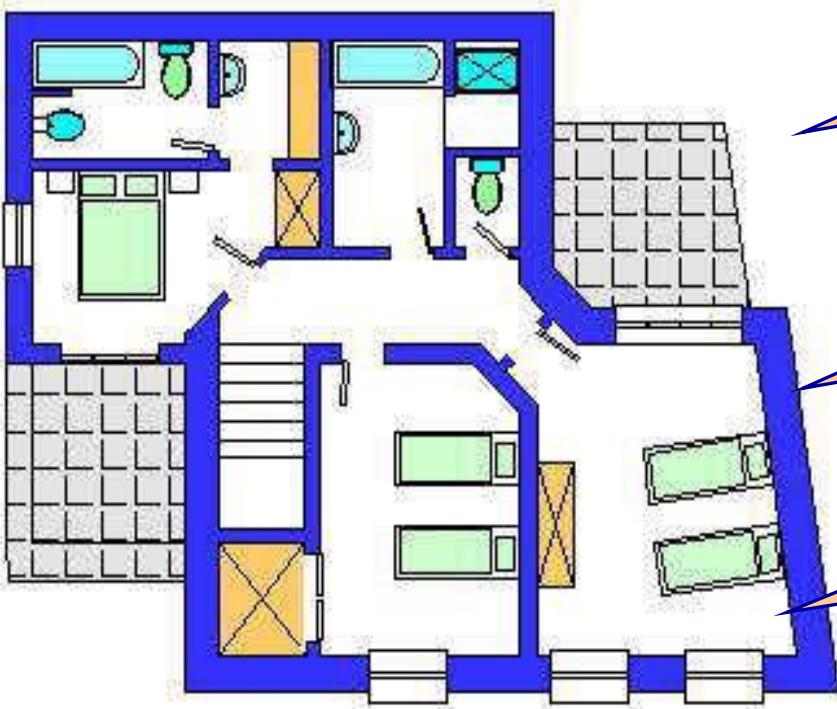
Business Architecture :
Goals of the city and the processes
in a city

Information Architecture:
healthcare, education, water,
electricity, transport, ...

Application Architecture :
hospital, police, library, schools, ...

Infrastructure Architecture :
roads, railroads, harbour, airport, ...

House planning



Information Architecture:
water, electricity, heating, ...

Application Architecture :
Living room, bedrooms, kitchen, ...

Infrastructure Architecture :
Electrical wires, water pipes,
driveway,...



Warship Vasa

- Customer
 - King of Sweden
 - Gustav II Adolf
- Requirements:
 - 70 m long
 - 300 soldiers
 - 64 guns
 - 2 decks
- Architect
 - Hendrik Hybertson



Software architecture is hard!

- Complexity
 - No physical limitations
 - Huge state-space
- Constant change of
 - Business
 - Technology
- The architecture is never ideal

The work of an architect: Make non-optimal decisions in the dark.



Main point

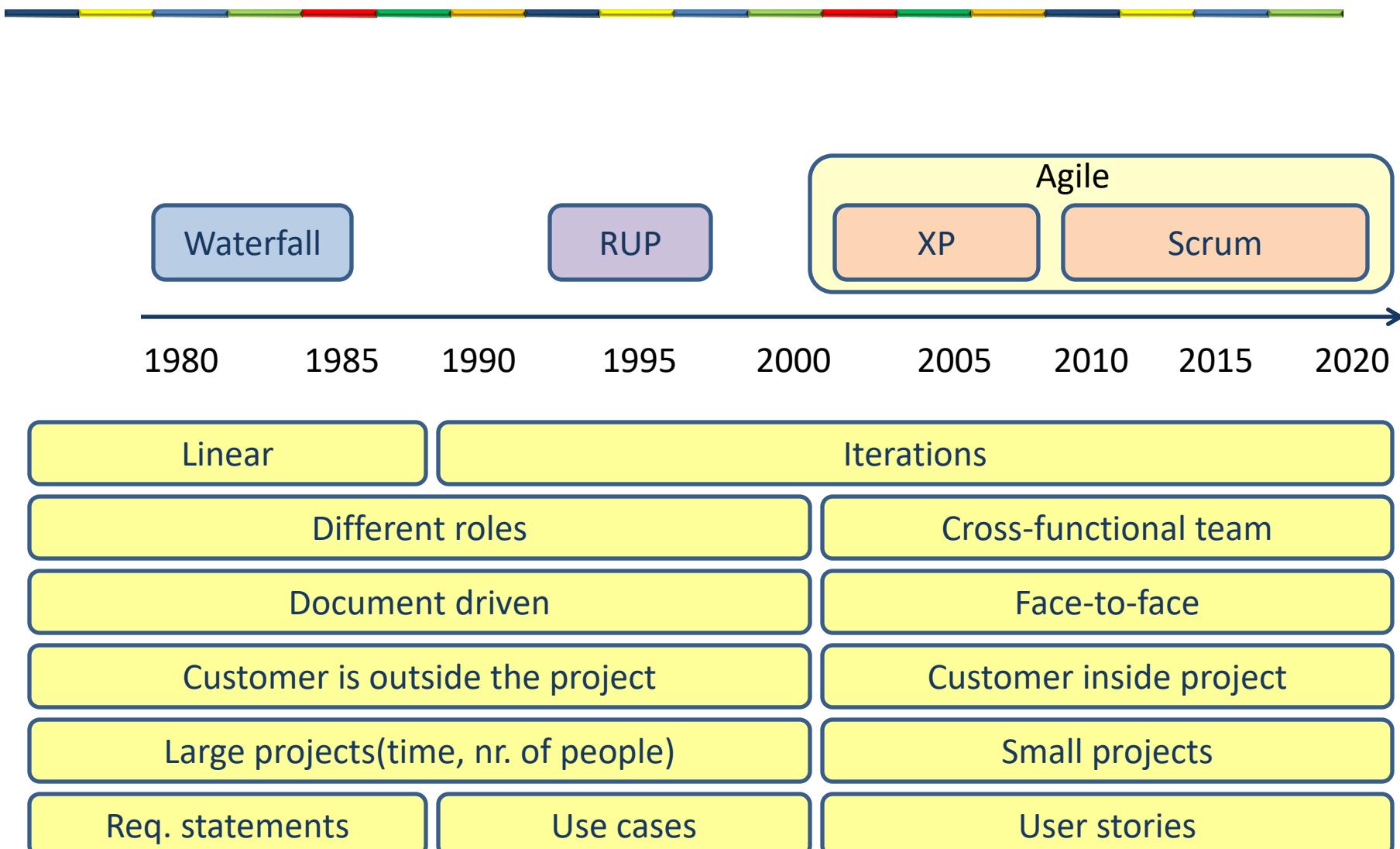
- Software architecture is defining all the **important stuff** in a software development project.
- The human physiology has the same structure as the structure of the Veda and Vedic literature who are expressions of the structure of pure consciousness.



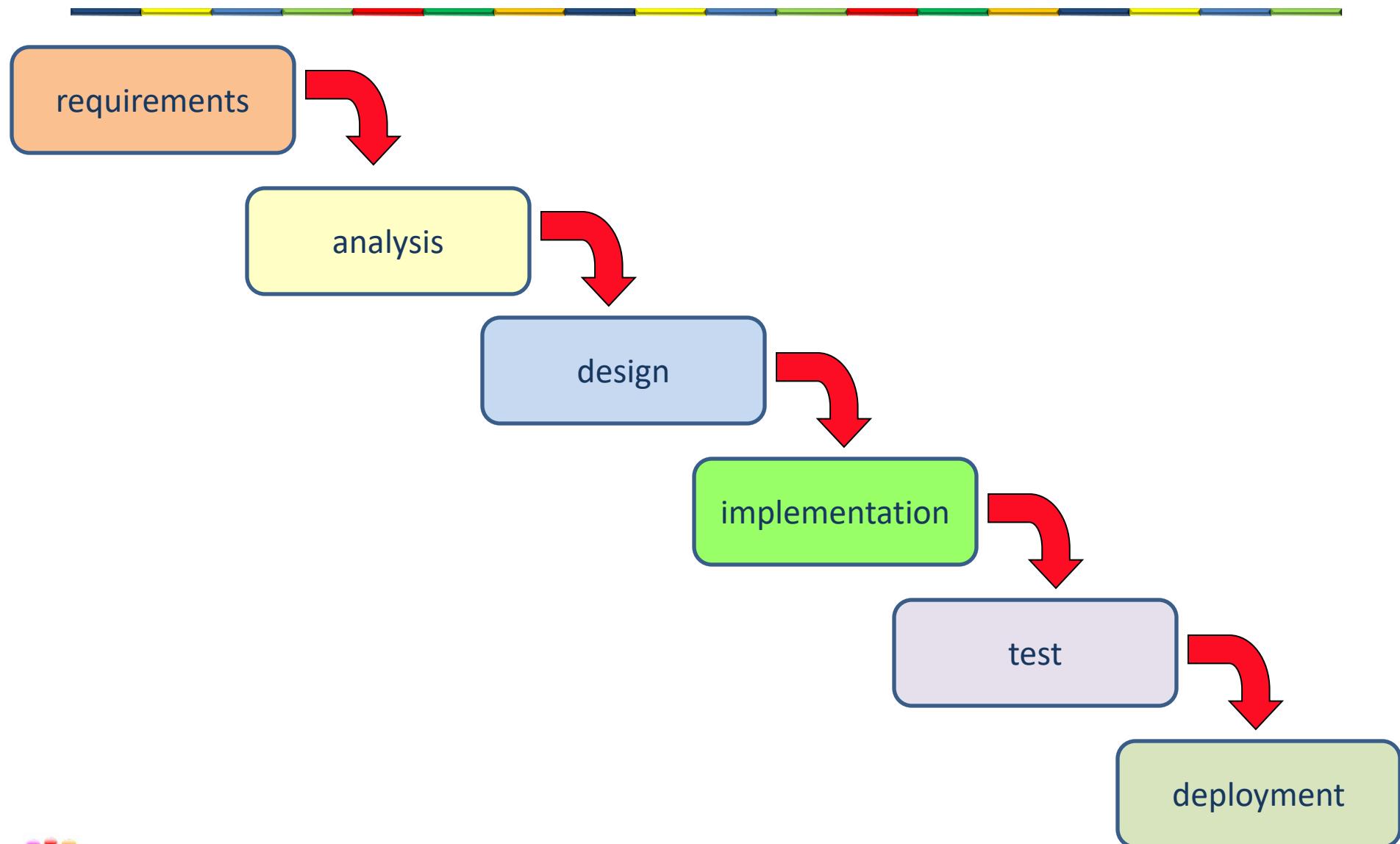
HOW DOES SOFTWARE ARCHITECTURE FIT IN THE PROCESS



Software development methods



Waterfall



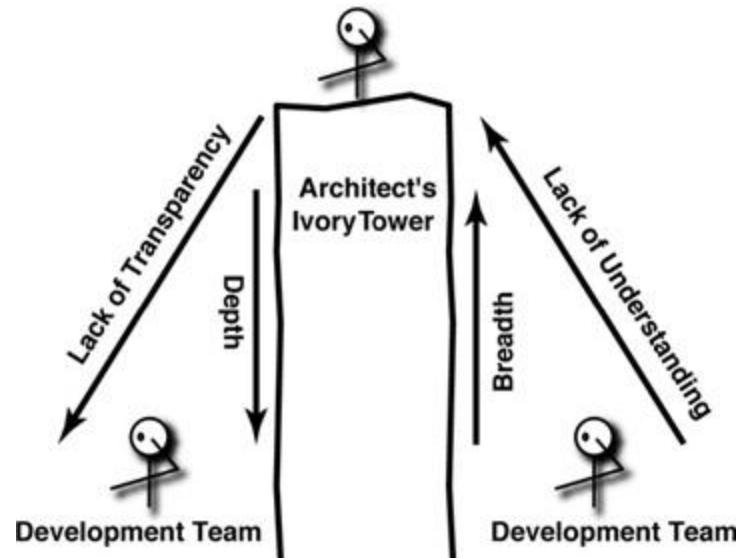
Waterfall software architecture

- The architect designs the system
 - Big upfront architecture
 - Large Software Architecture Document (SAD)
 - Ivory tower architect
 - The architecture is not understood and implemented by the developers
 - The architect does not understand the current technology
 - The architect is only available in the beginning of the project



Ivory tower architect

- It is very hard to truly know the best solutions for design problems if you are not working (coding) on the project
- It takes many iterations of a solution before it finally works - so you can't suggest a solution and then leave



The Agile Manifesto

Individuals and interactions

over

Processes and tools

Working software

over

Comprehensive documentation

Customer collaboration

over

Contract negotiation

Responding to change

over

Following a plan

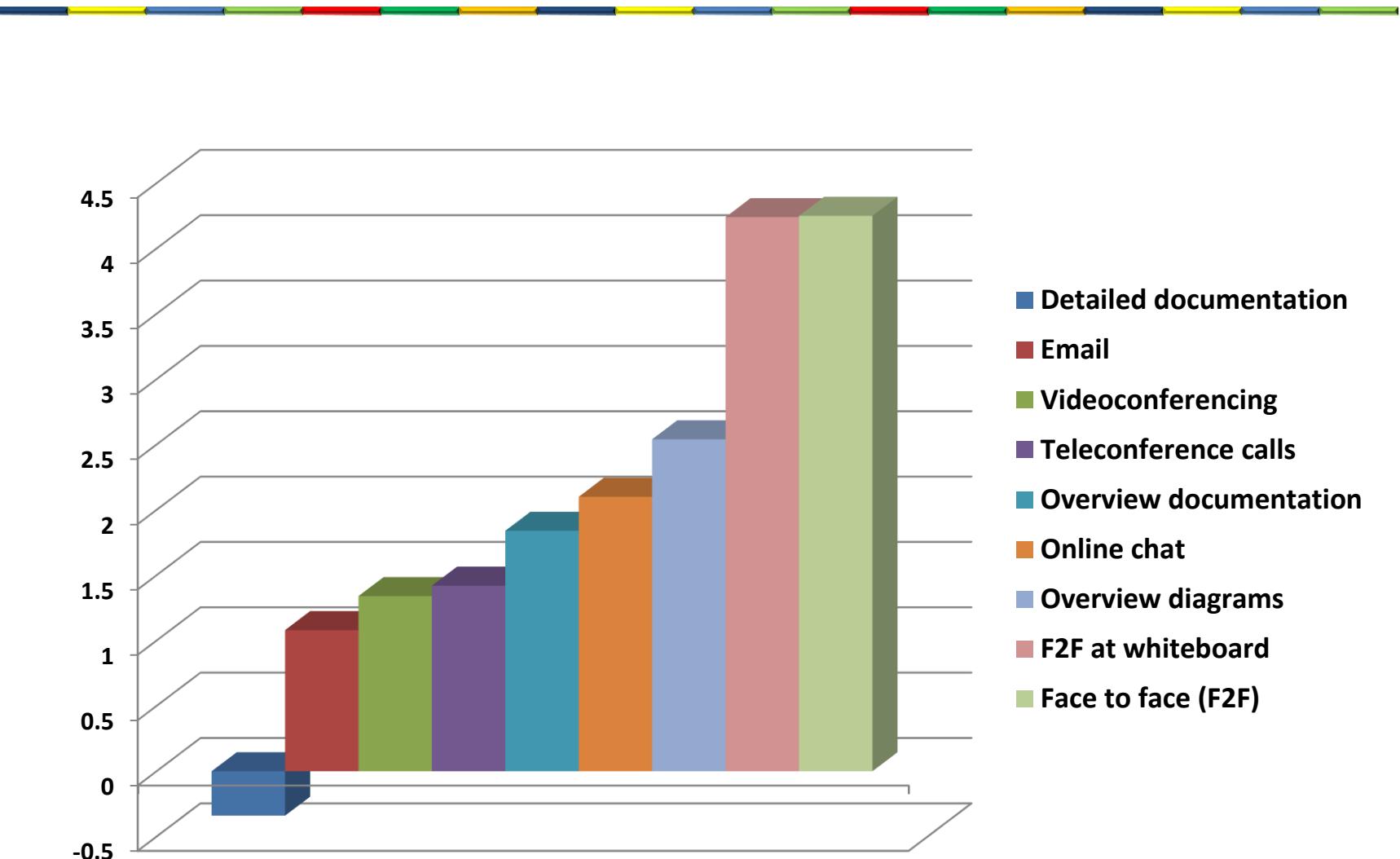


Agile principles

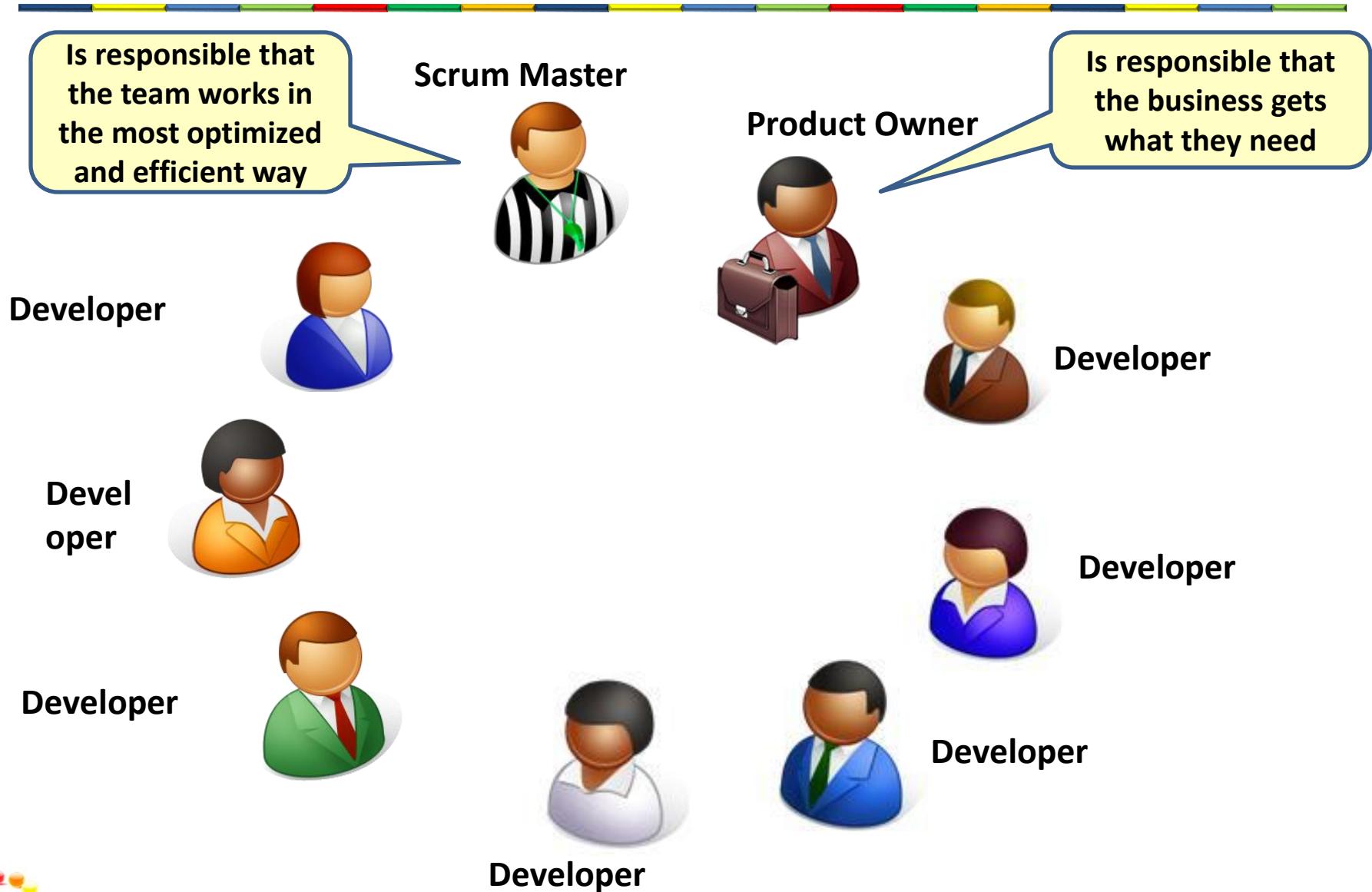
- Early and continuous delivery of valuable software.
- Welcome changing requirements.
- Business people and developers must work together daily.
- Give the team the environment and support they need, and trust them to get the job done.
- Prefer face-to-face conversation.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design
- Simplicity is essential.
- Self-organizing teams.



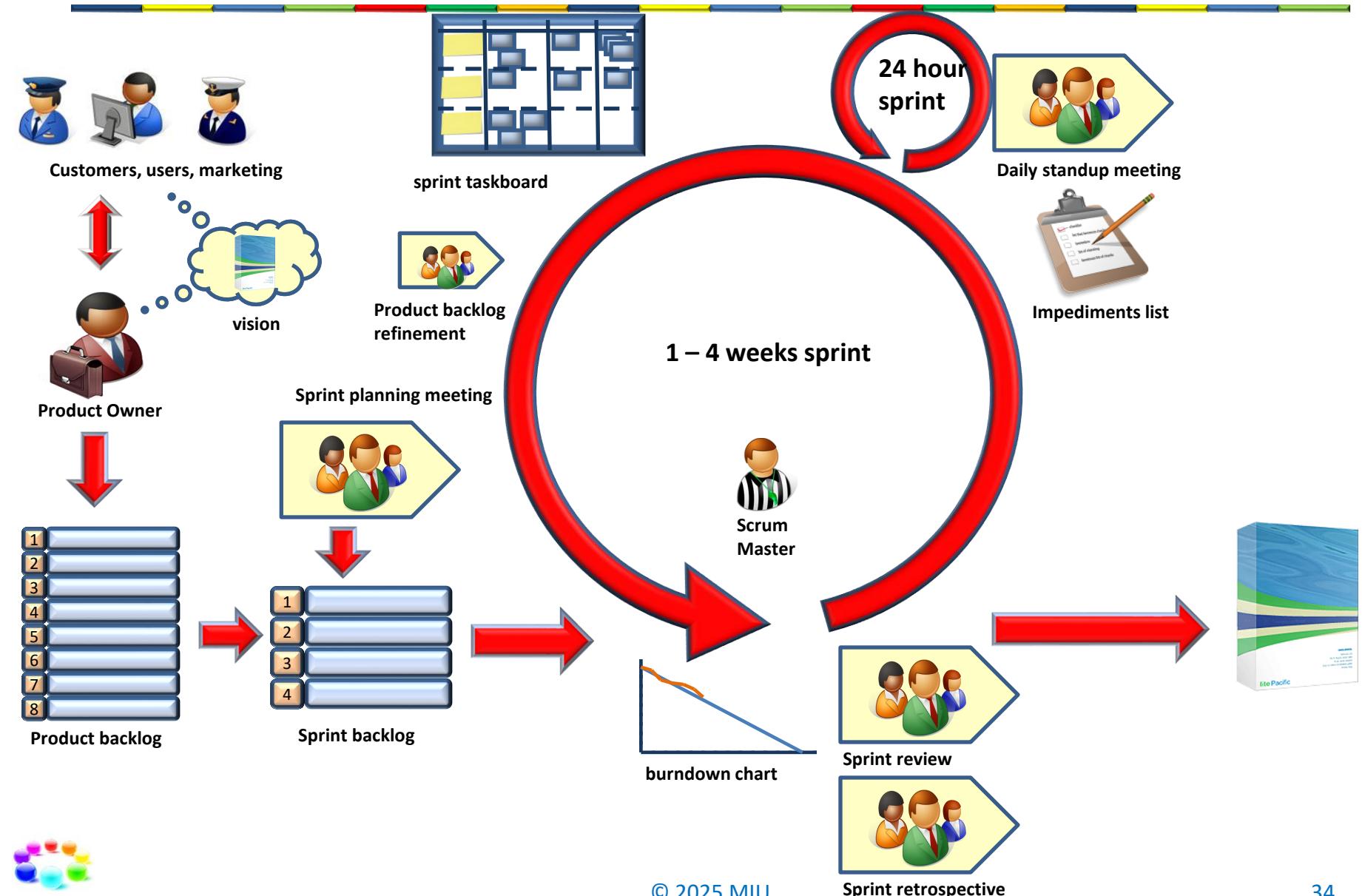
Effectiveness of communication



Scrum team



Scrum in action



Comparing waterfall and agile

PROJECT SUCCESS RATES AGILE VS WATERFALL

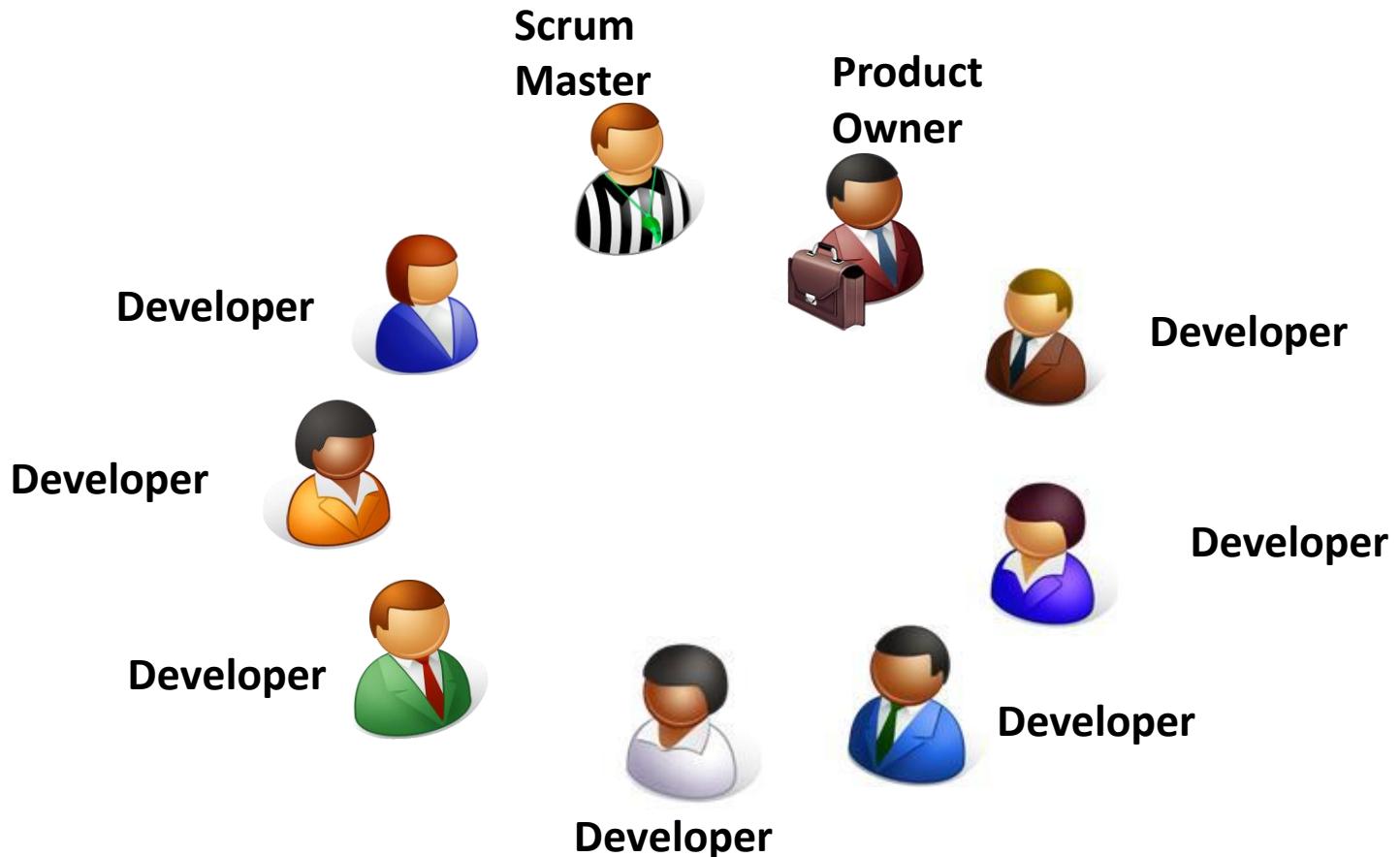


The chaos manifesto 2017

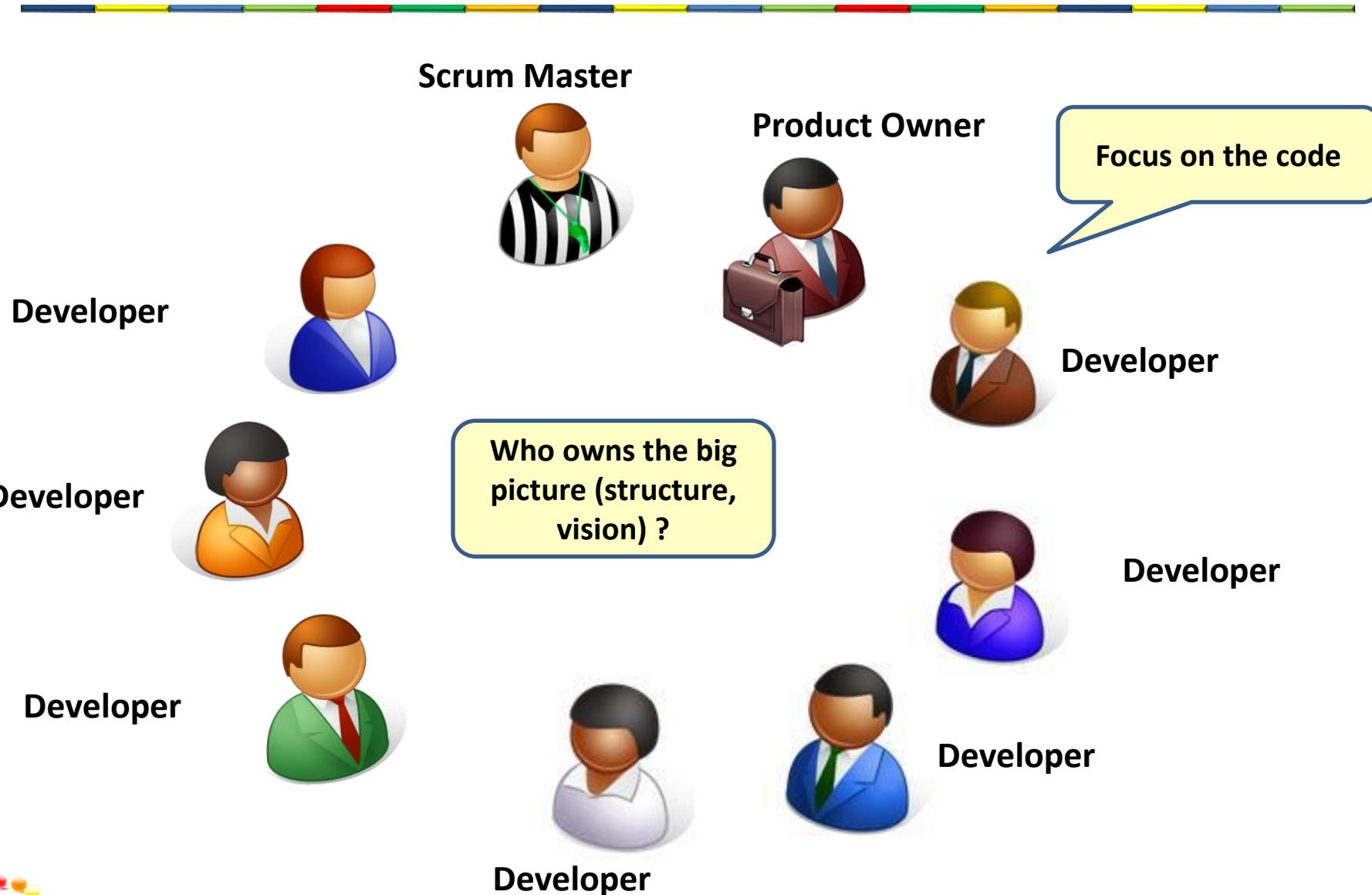


Agile architecture

- Architecture is a **task**, not a role
 - Team is responsible for architecture



Scrum team

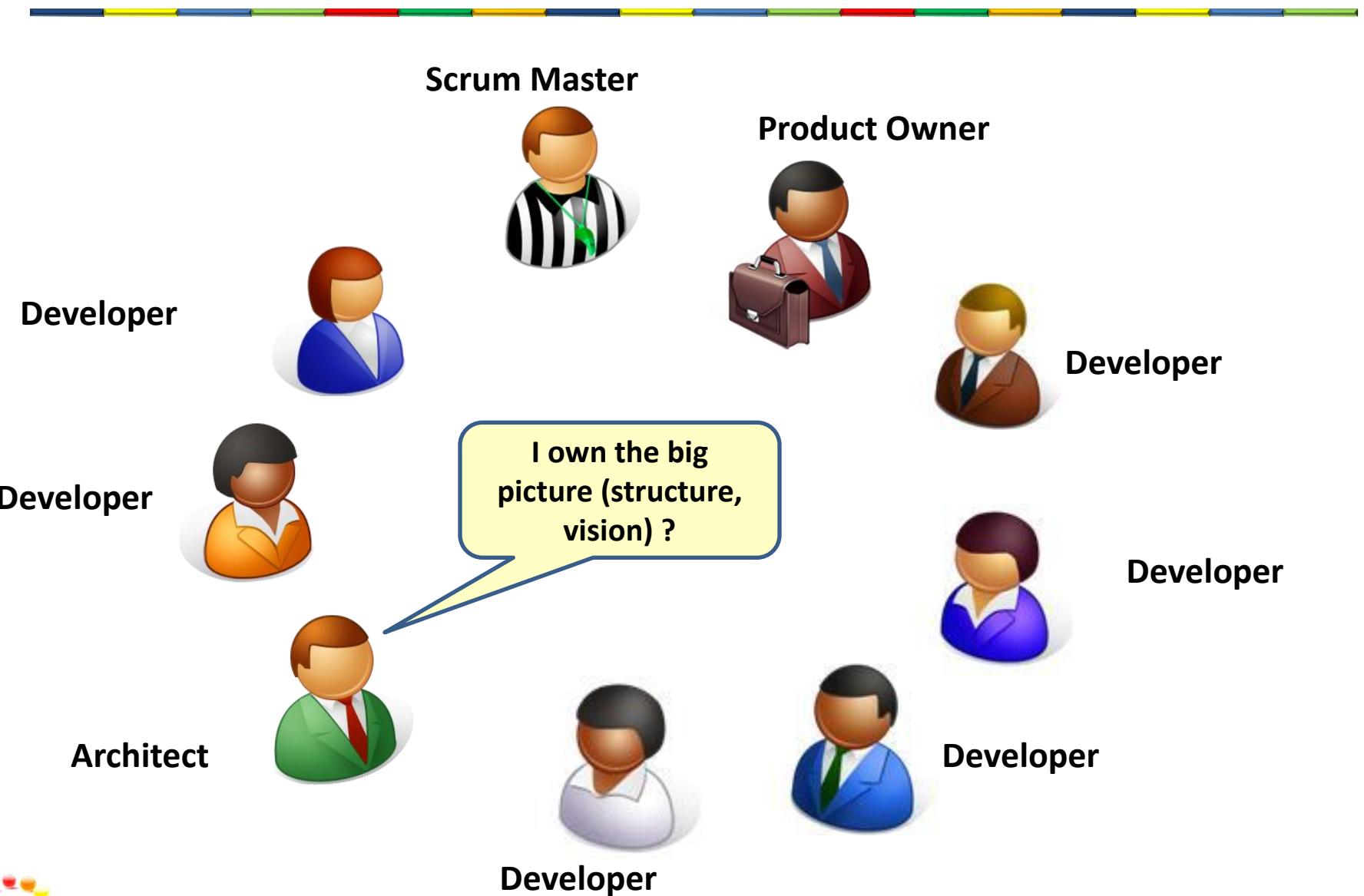


Why software architecture in agile?

- We need a **clear vision** and **roadmap** for the team to follow.
- We need to identifying and mitigating **risk**.
- We have to **communicate** our solution at different levels of abstraction to different audiences.
- We need **technical leadership**
- We have to make sure our architecture is consistent, correct and fits within the context



You need an architect

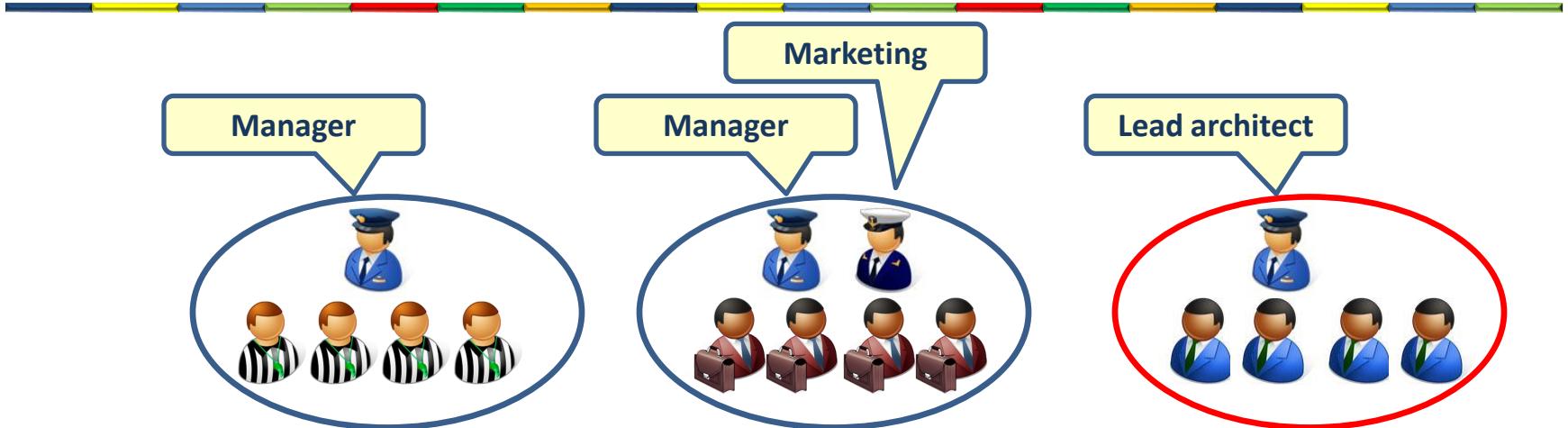


Agile architecture

- Just enough upfront architecture
 - Refine the architecture later
- Keep your architecture flexible
- The architect is available during the whole project
- The architect also writes code
 - But not all the time
 - The architecture is grounded in reality
 - Works together with the developers
 - Architects should be master builders
- Proof the architecture in the first iteration(s)



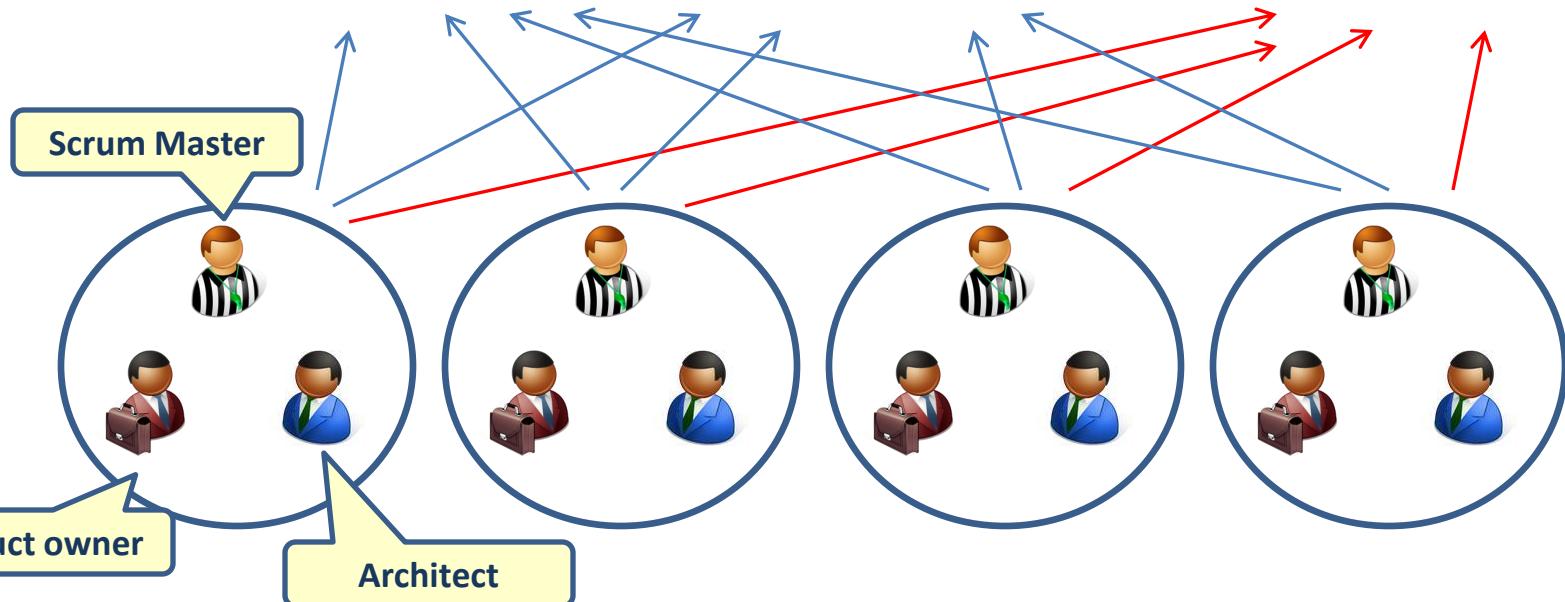
Architect group



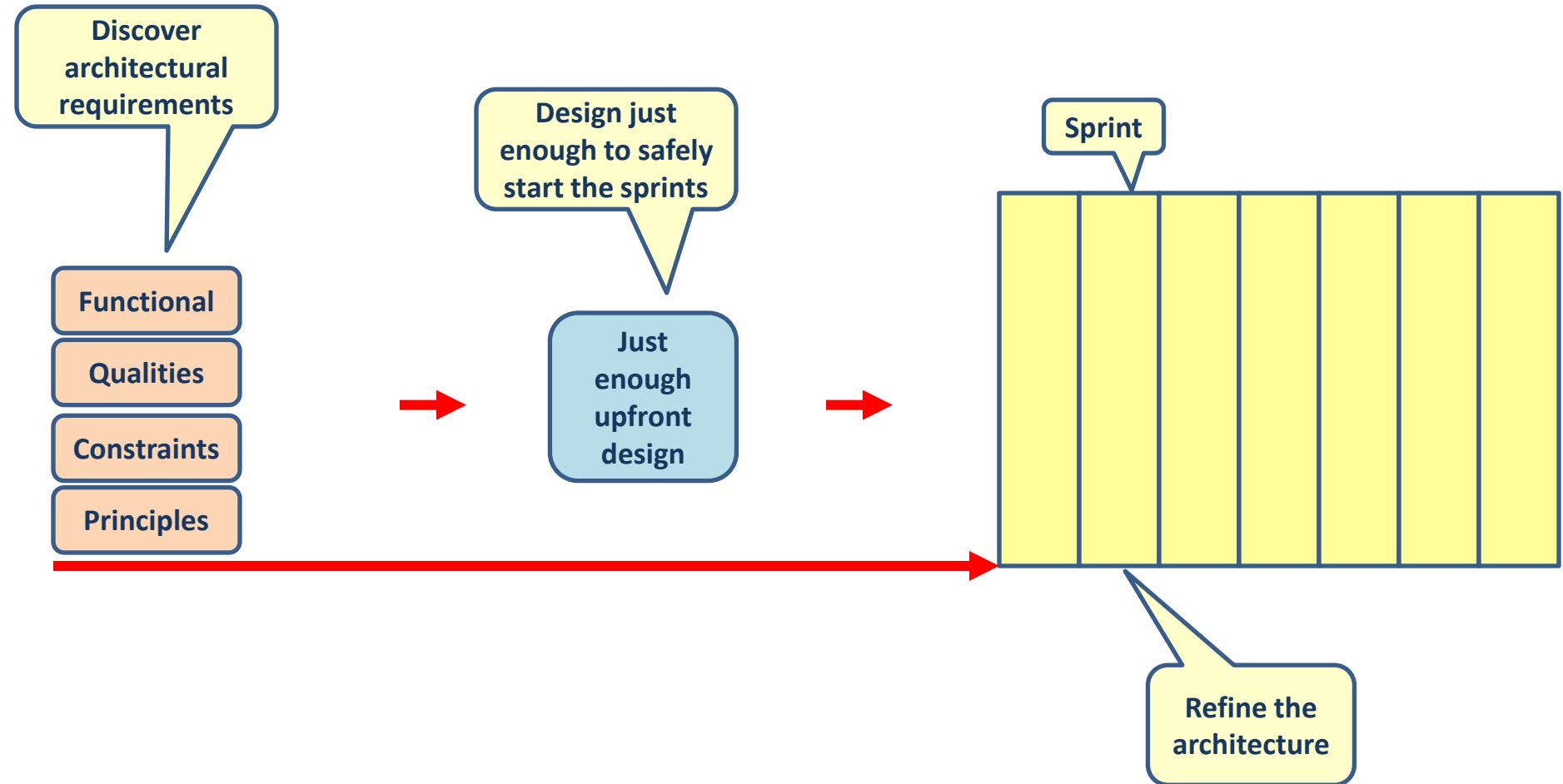
Scrum master group

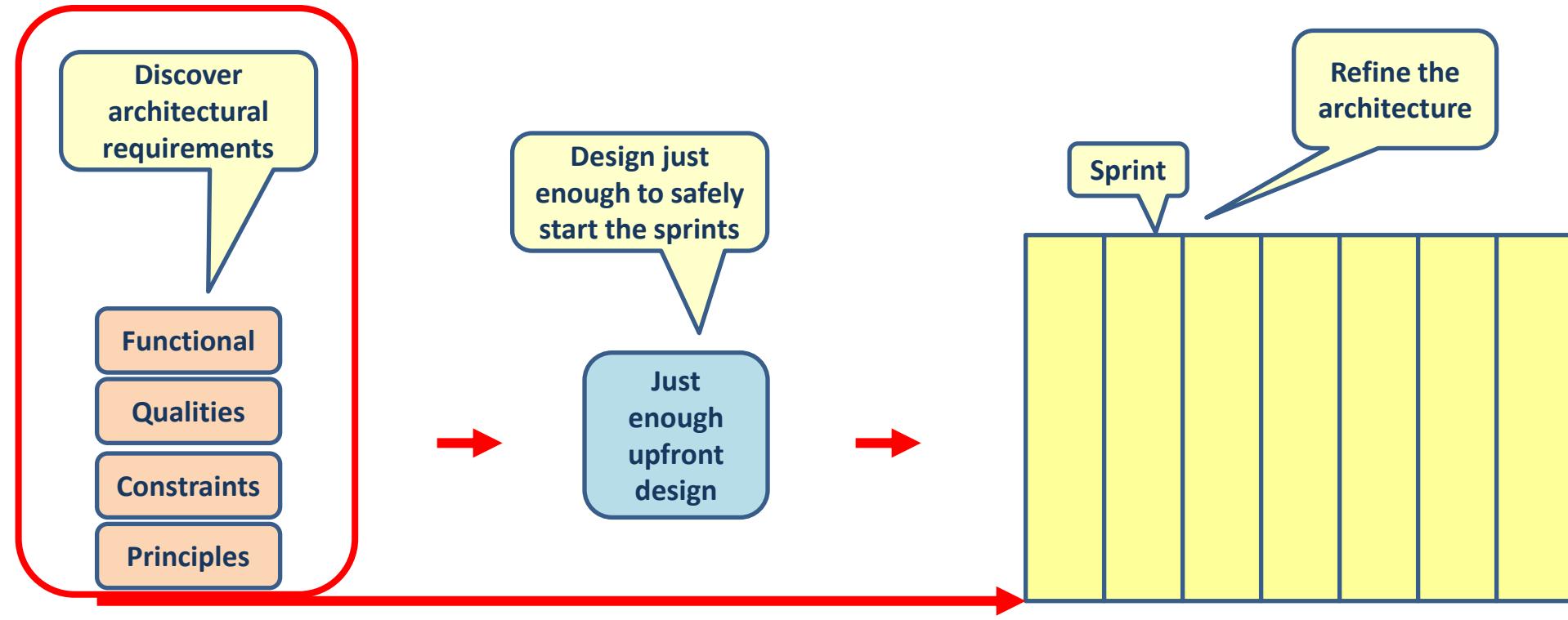
Product owner group

Architect group



Agile architecture





ARCHITECTURAL REQUIREMENTS



Functional requirements

- What should the system functionally do?
 - Use cases
 - User stories

As a customer
I can view my account history
so that I know all transactions on my
account

Functional

Qualities

Constraints

Principles



Architectural constraints

- Constraints from the business (or enterprise architecture
 - We do everything in .Net
 - We always use an Oracle database
 - Our maintenance engineers all know Java
 - All applications talk with the Oracle ESB
- Budget
- Deadlines



SOFTWARE QUALITIES

Functional

Qualities

Constraints

Principles



Wikipedia software qualities

- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composableity [Erl]
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability



ARCHITECTURE PRINCIPLES

Functional

Qualities

Constraints

Principles

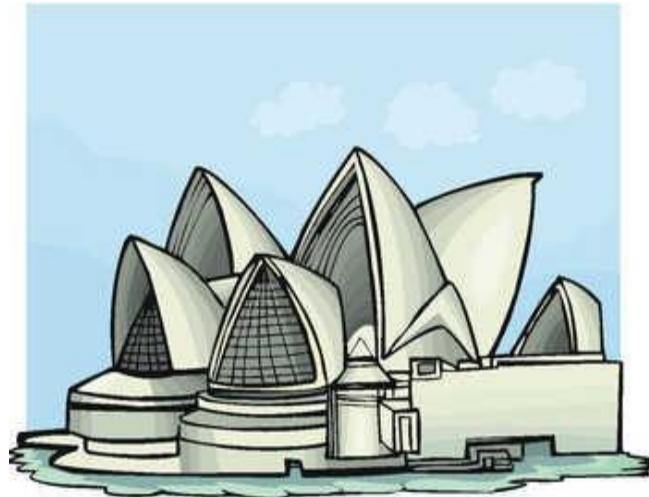


Architecture (design) principles

- Keep it simple
- Keep it flexible
- Loose coupling
 - High cohesion, low coupling
- Separation of concern
- Information hiding
- Principle of modularity
- Open-closed principle



Keep it simple



- The more complexity, the more change on failure
- Simple applications, frameworks, tools, etc. remain to be used
 - Complex ones will be replaced by something simple
- Gold plating

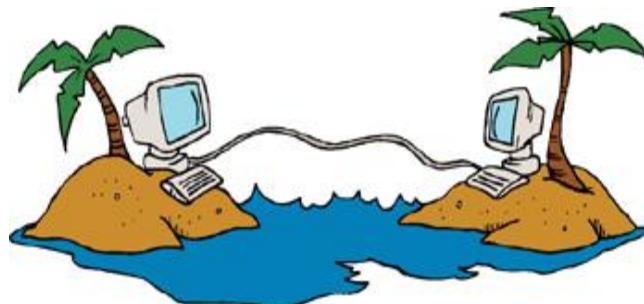
Keep it flexible

- Everthing changes
 - Business
 - Technical
- More flexibility leads to more complexity



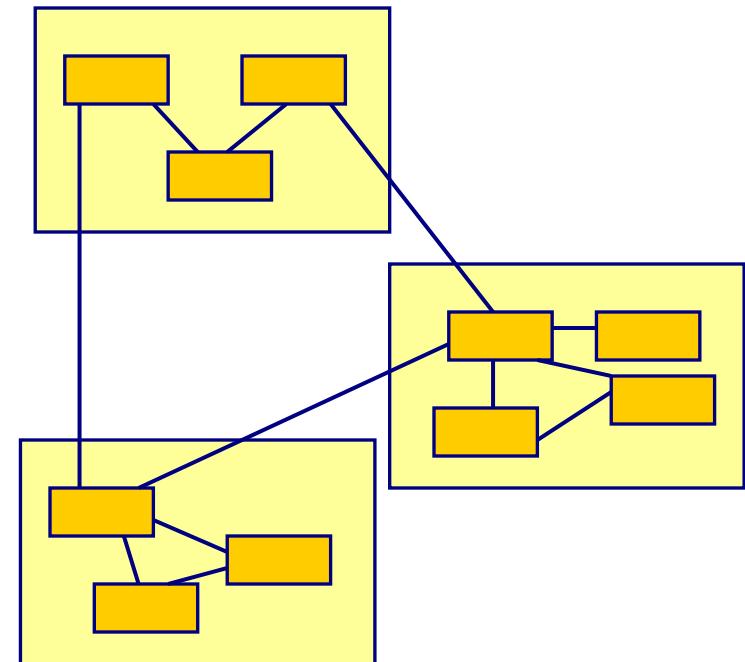
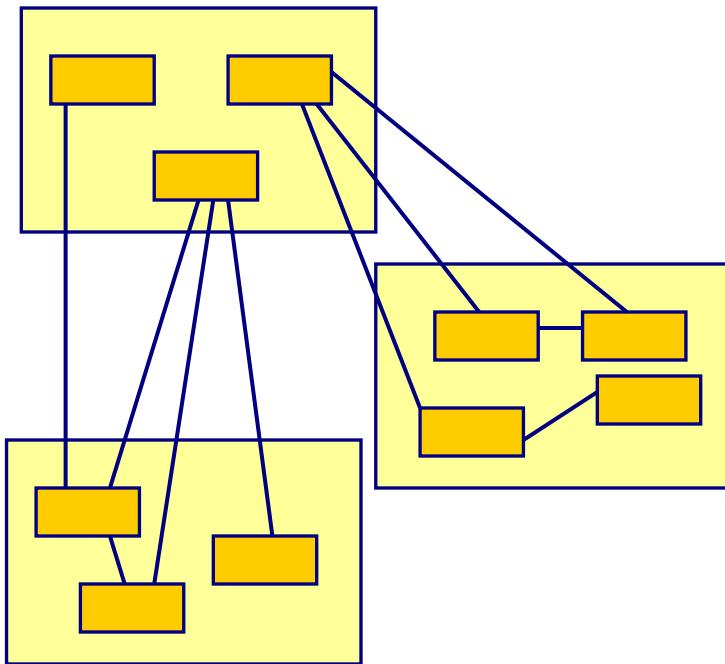
Loose coupling

- Different levels of coupling
 - Technology
 - Time
 - Location
 - Data structure
- You need coupling somewhere
 - Important is the level of coupling



High cohesion, low coupling

- High coupling, low cohesion
- High cohesion, low coupling



Separation of concern

- Separate technology from business
- Separate stable things from changing things
- Separate things that need separate skills
- Separate business process from application logic
- Separate implementation from specification



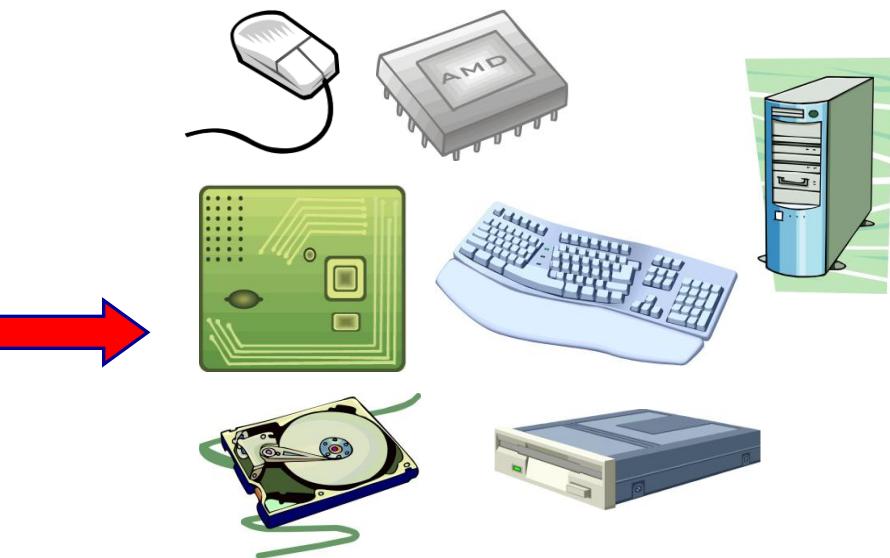
Information hiding

- Black box principle
- Hide implementation behind an interface
- Hide the data structure behind stored procedures
- Hide the data structure behind business logic



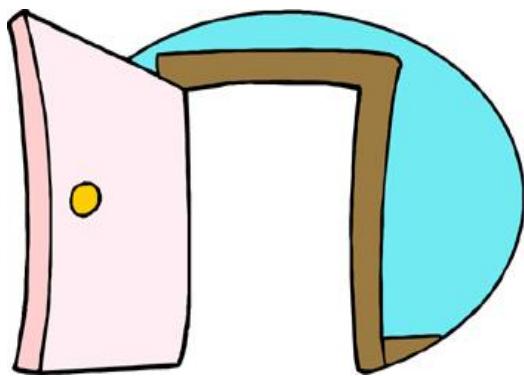
Principle van modulariteit

- Decomposition
- Devide a big complex problem is smaller parts
- Use components that are
 - Better understandable
 - Independent
 - Reusable
- Leads to more flexibility
- Makes finding and solvings bugs easier



Open- closed principle

- The design should be “open” for extension, but “closed” for change.
- You want to add new functionality instead of changing existing, working and tested code.



Most important architecture principles

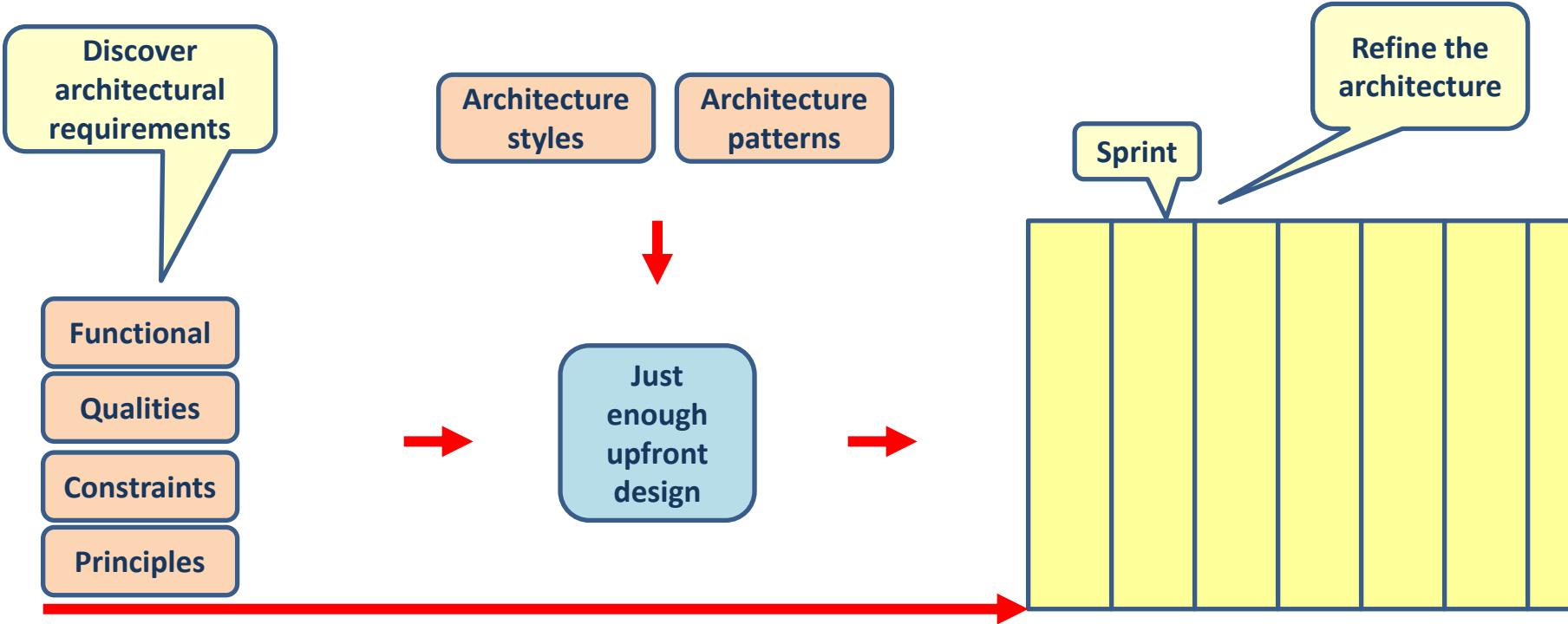
- Keep it simple
- Keep it flexible
- Loose coupling
 - High cohesion, low coupling
- Separation of concern
- Information hiding
- Principle of modularity
- Open-closed principle



Main point

- Software architecture is never ideal. We have to find the right balance between the different software qualities and architecture principles
- Nature always takes the path of least resistance so that the perfection of the unified field can express itself in the relative creation

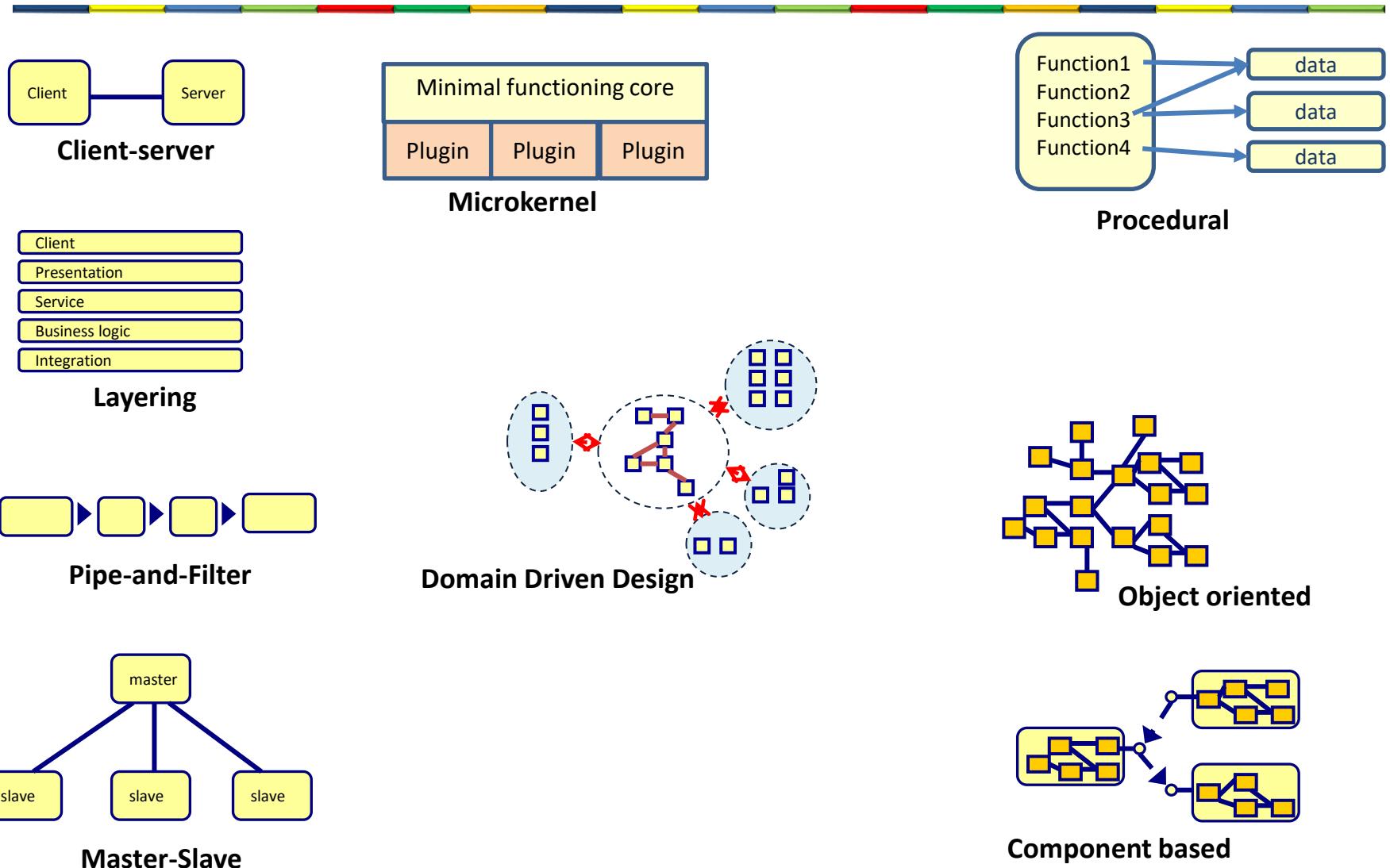




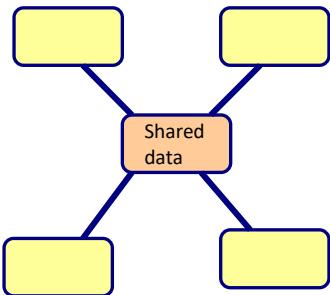
ARCHITECTURAL STYLES



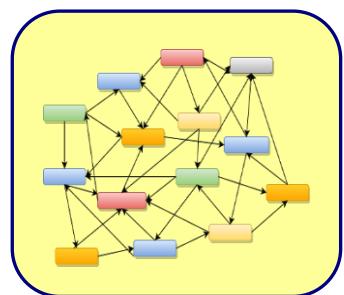
Architecture styles within an application



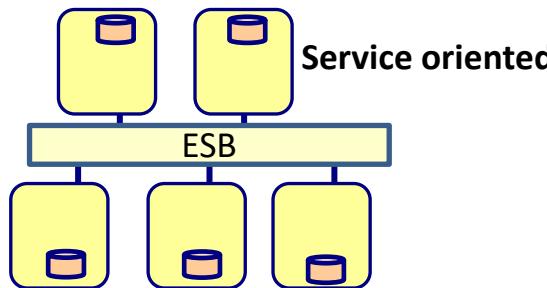
Architecture styles to connect applications



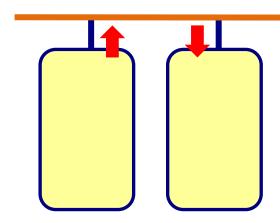
Blackboard



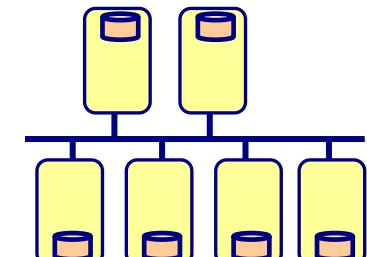
Monolith



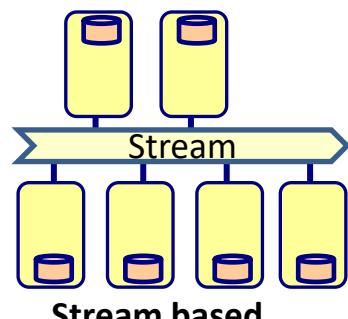
Service oriented



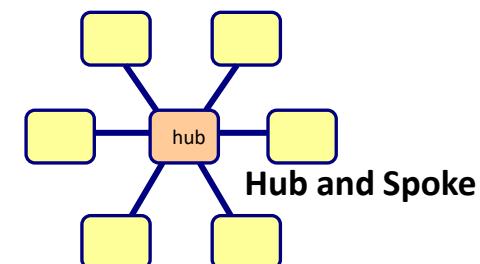
Event Driven



Microservices

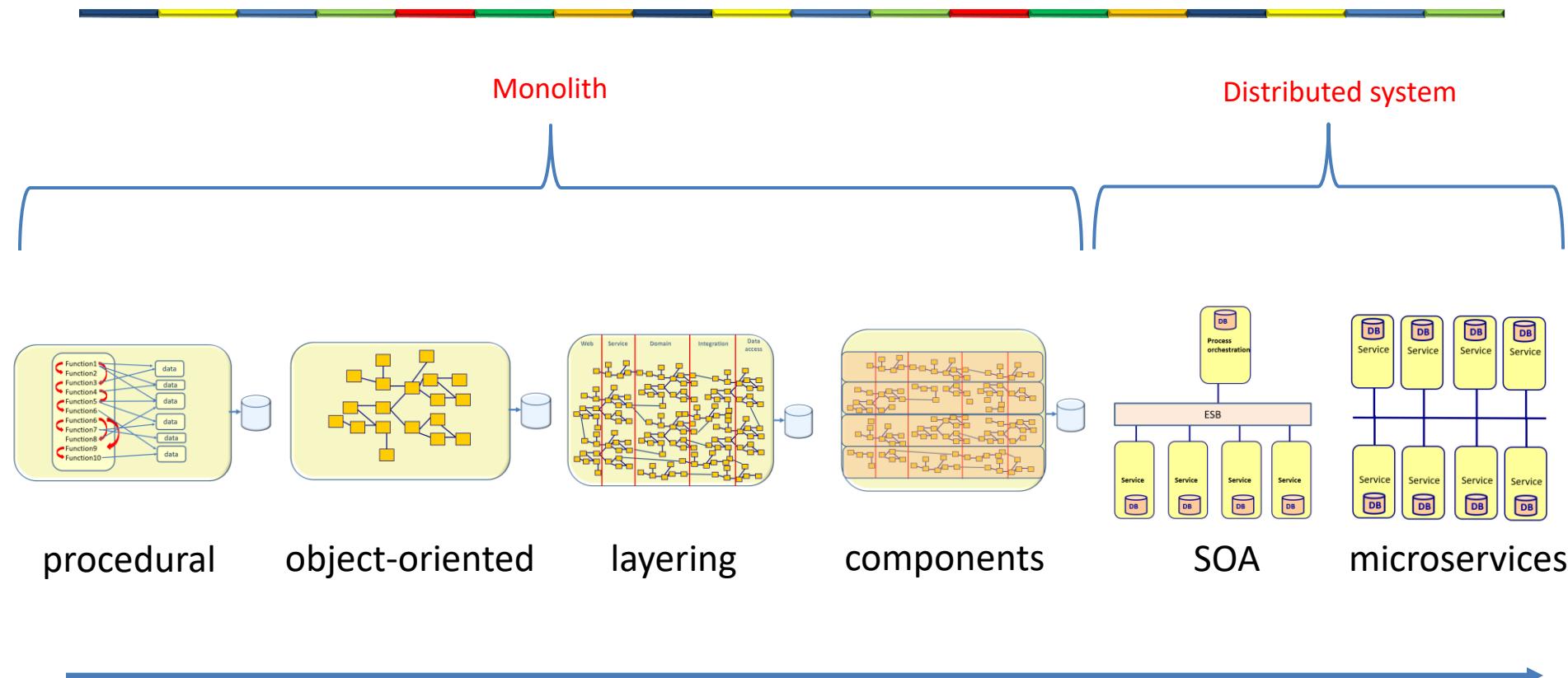


Stream based



Hub and Spoke

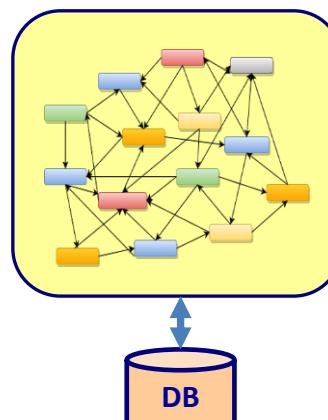
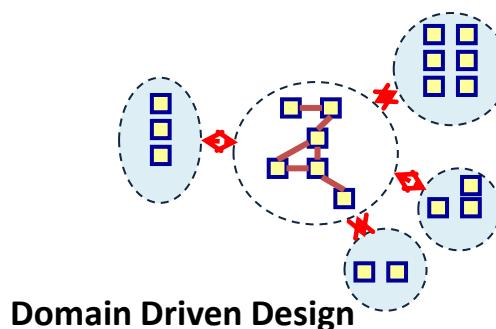
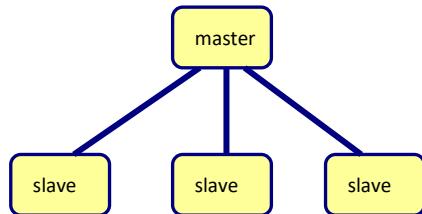
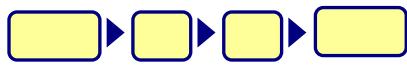
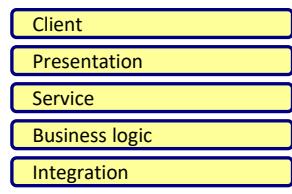
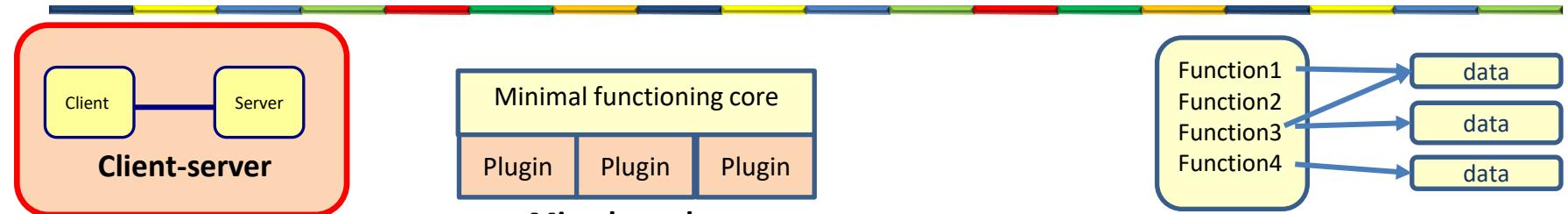
Architecture evolution



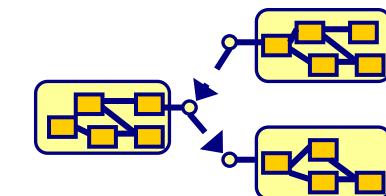
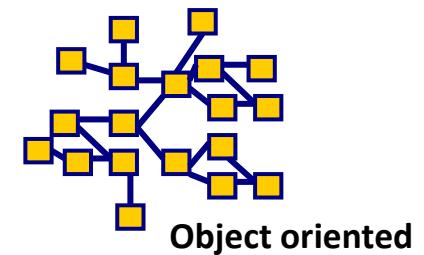
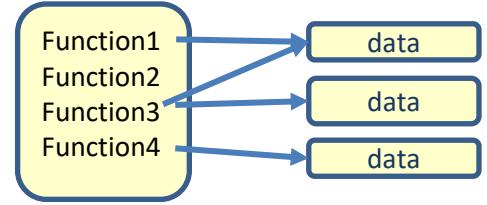
- Smaller and simpler parts
- More separation of concern
- More abstraction
- Less dependencies



Architecture styles within an application

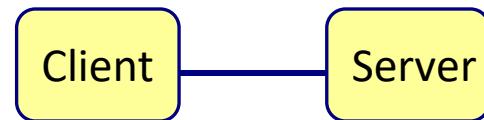


Monolith

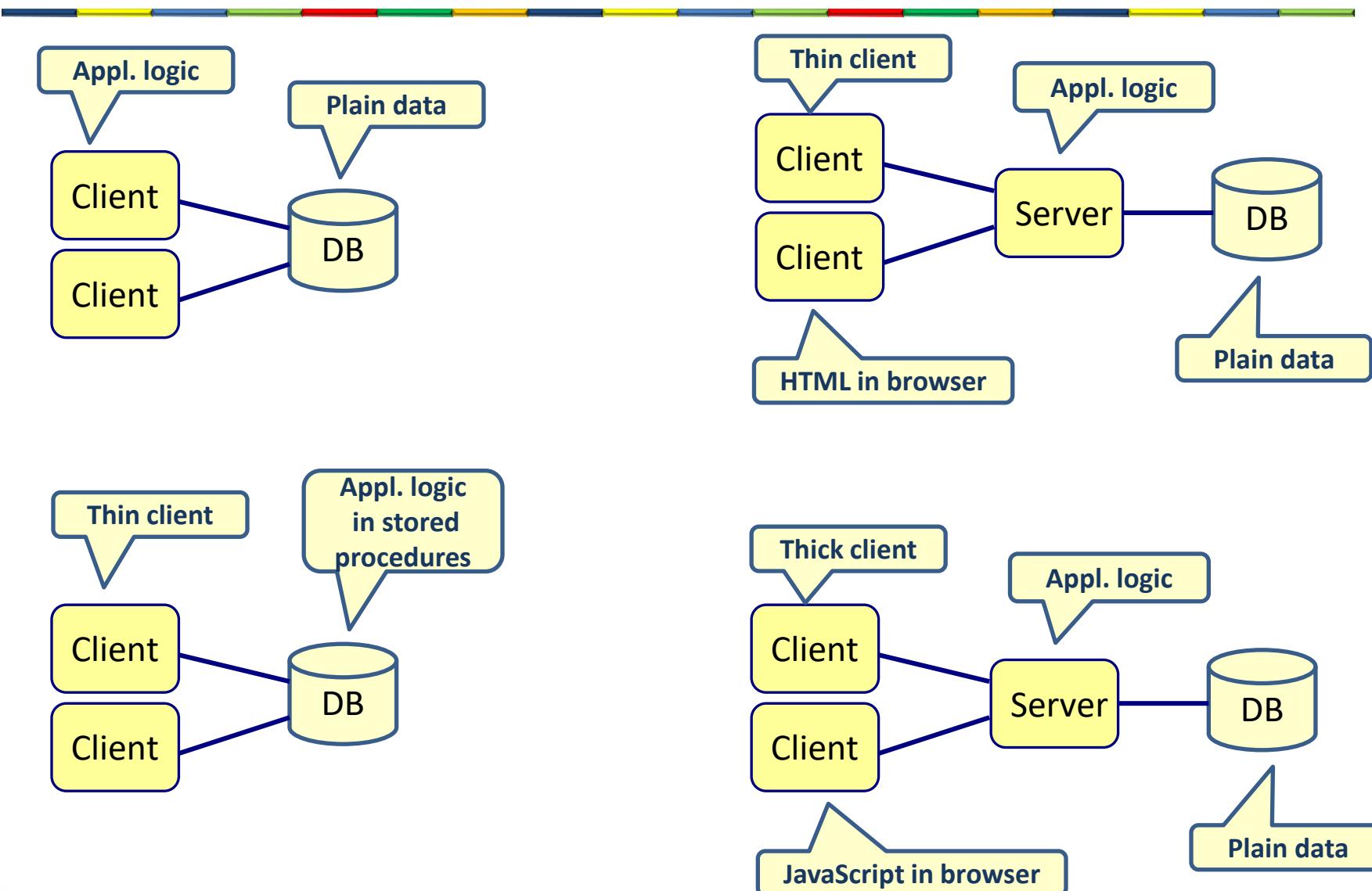


Client-Server

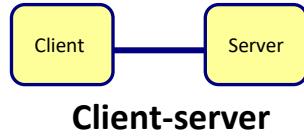
- Multiple clients per server
- Thin-client/Thick client
- Stateless / stateful server
- Multiple tiers
- Requests typically handled in separate threads



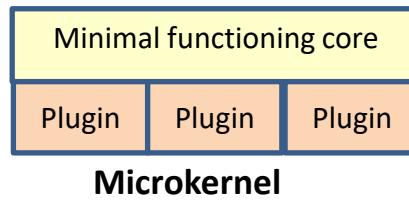
Client-server architectures



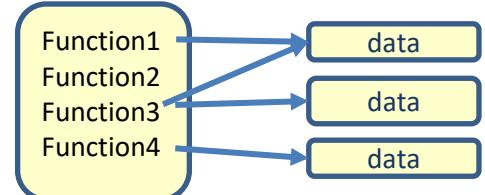
Architecture styles within an application



Client-server



Microkernel



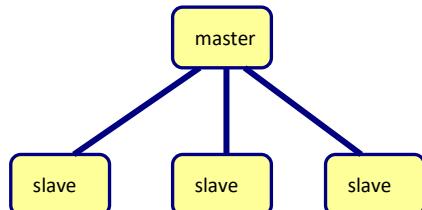
Procedural



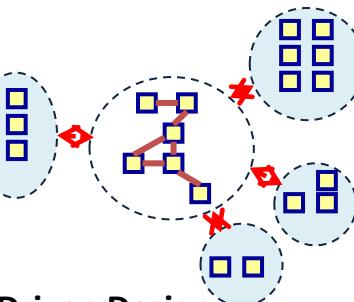
Layering



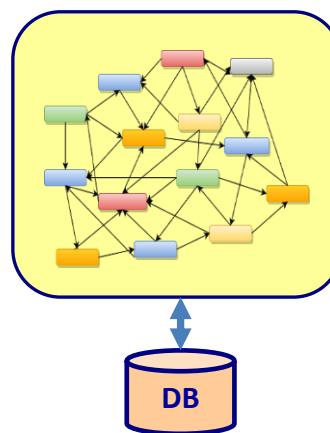
Pipe-and-Filter



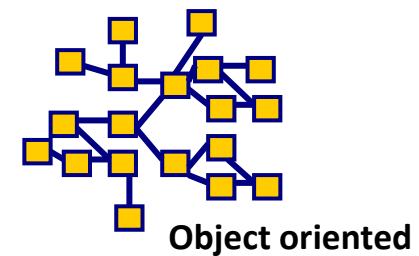
Master-Slave



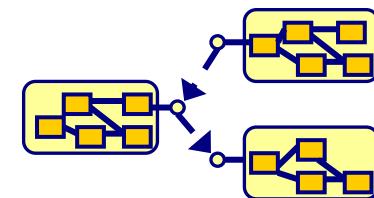
Domain Driven Design



Monolith



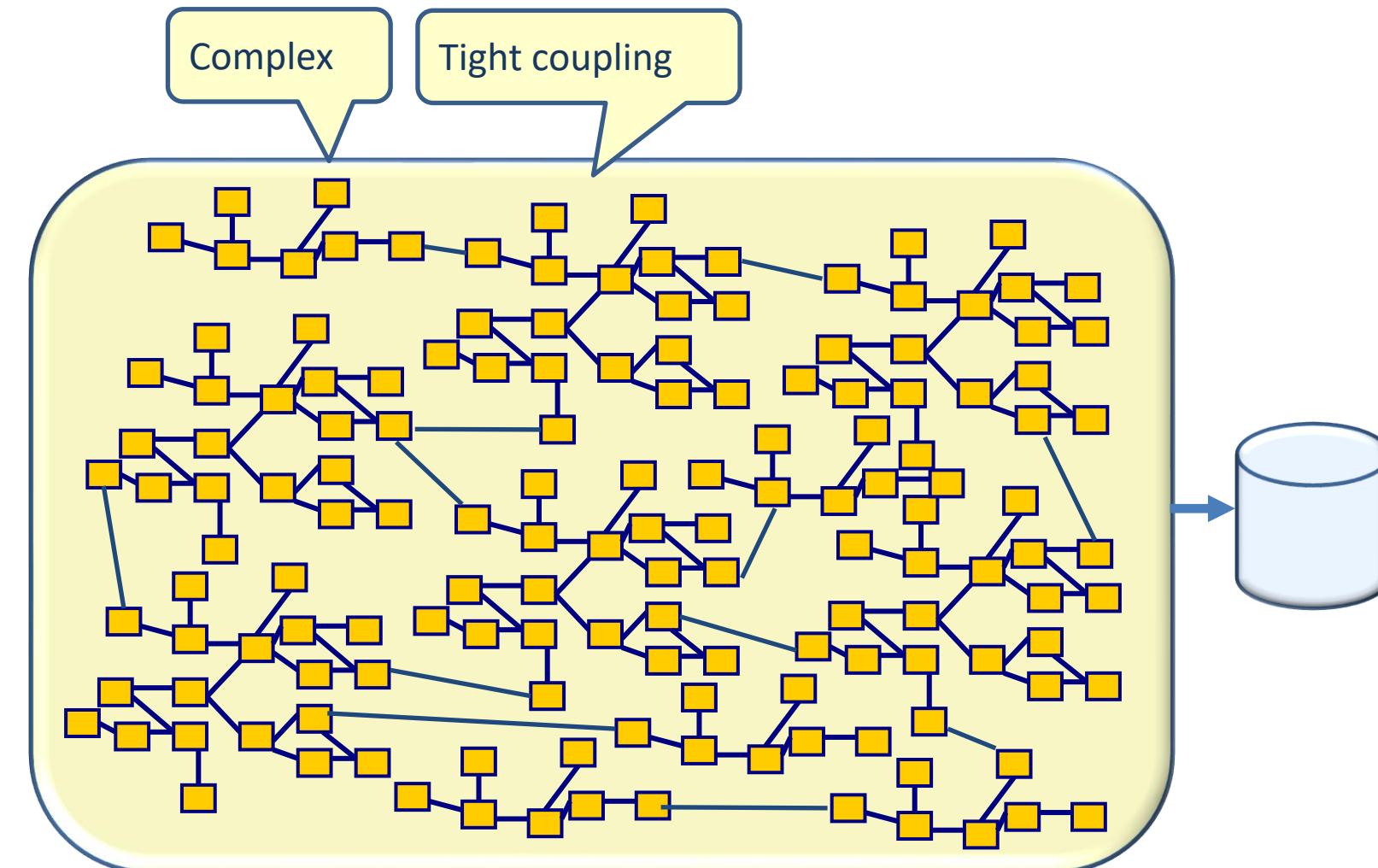
Object oriented



Component based

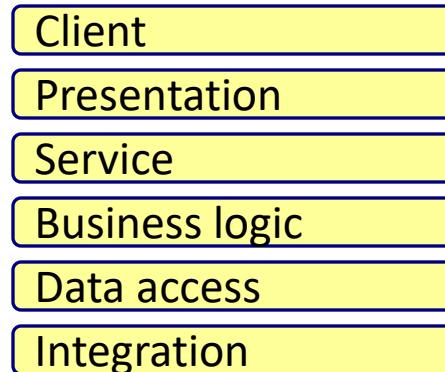


Object orientation

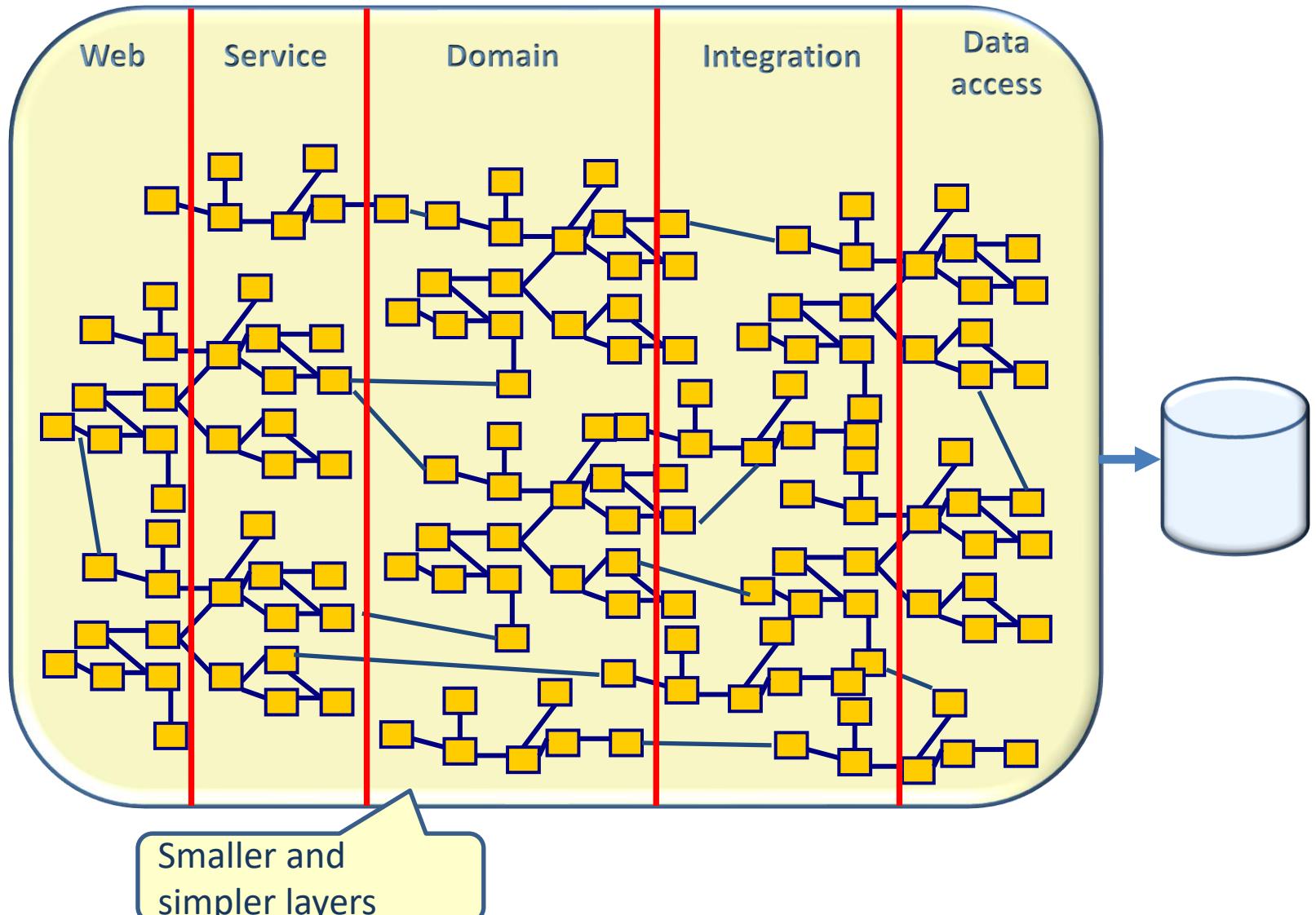


Layering

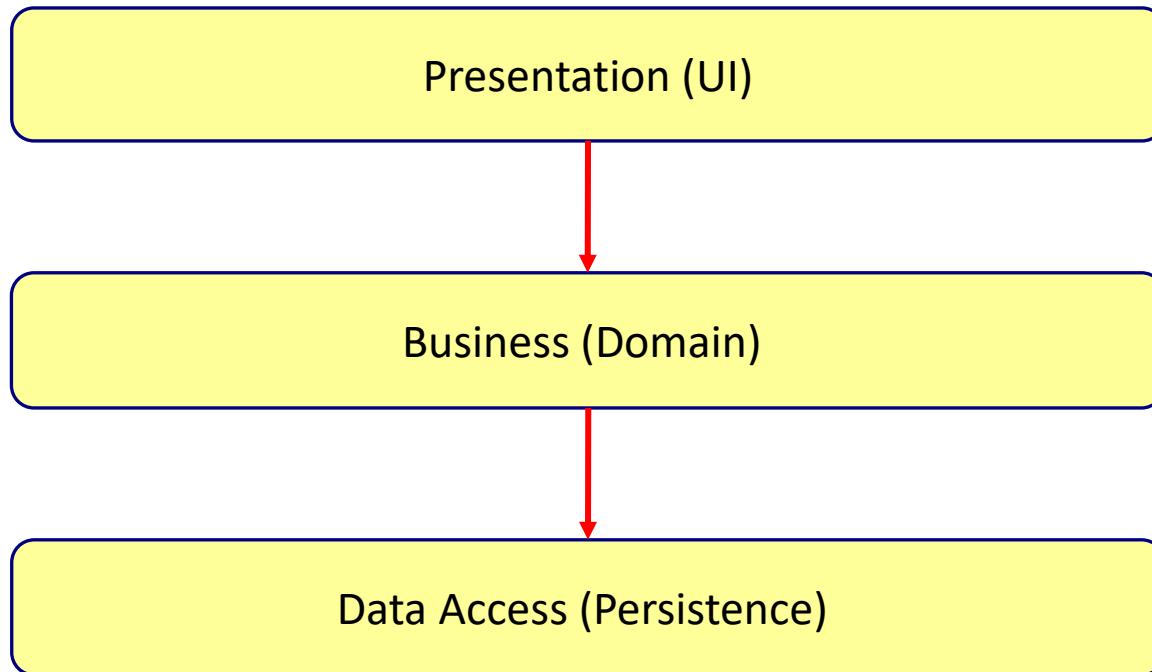
- Separation of concern
- Layers are independent
- Layers can be distributed
- Layers use different techniques



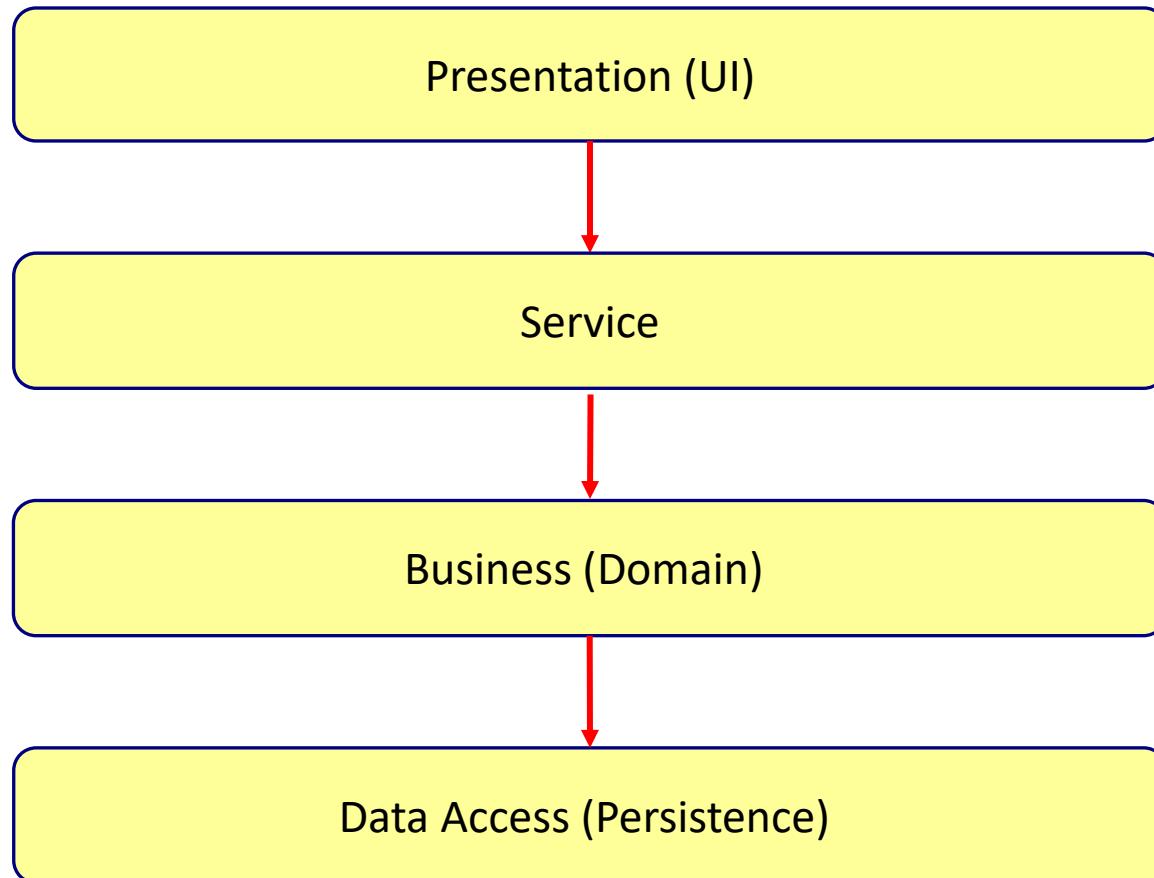
Layering



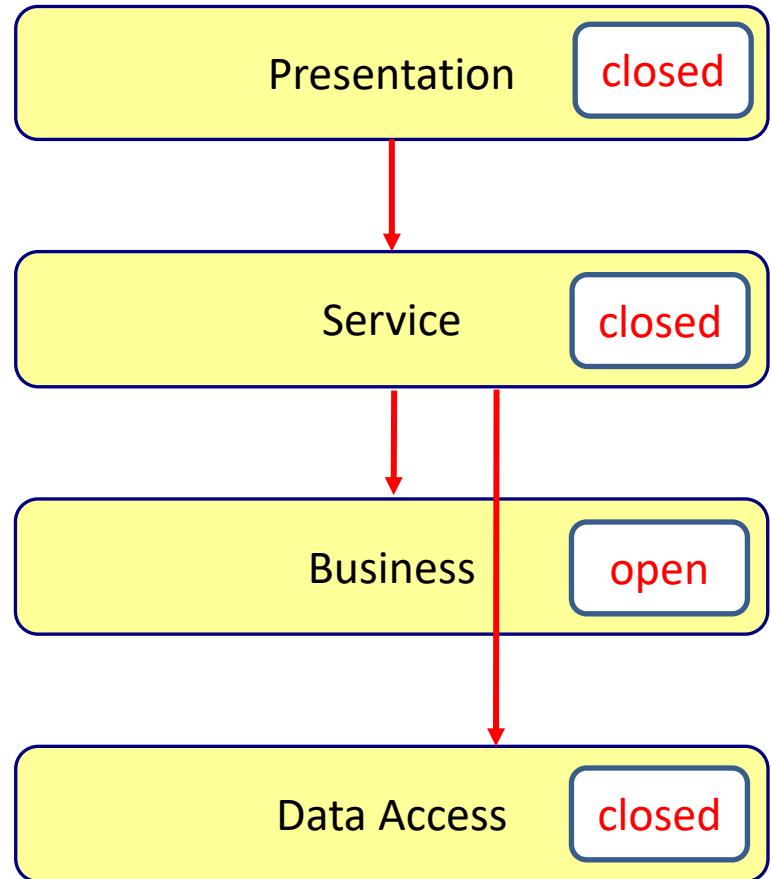
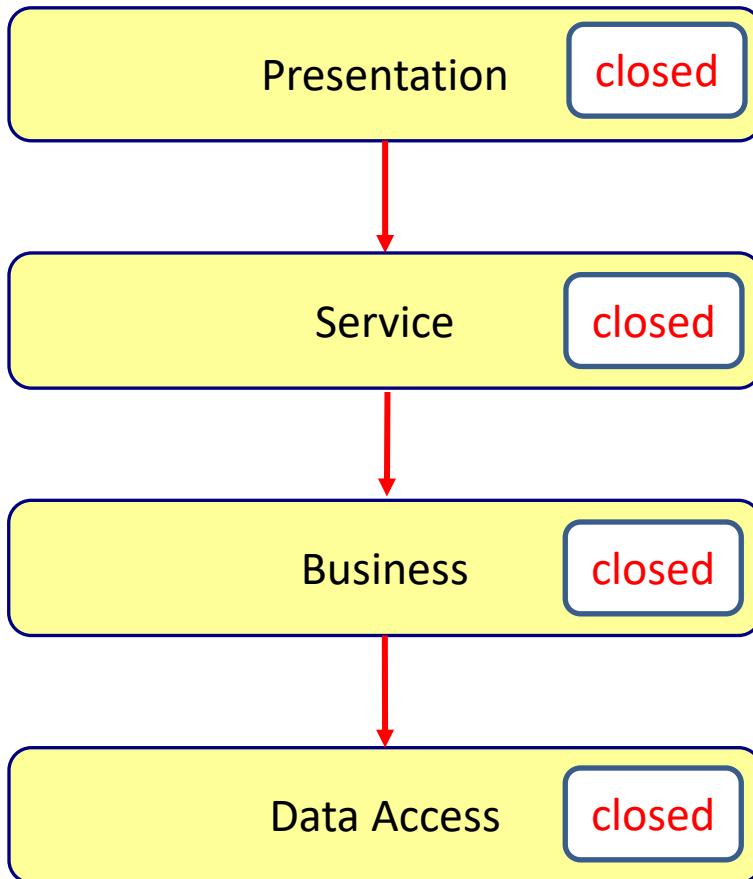
3 layered architecture



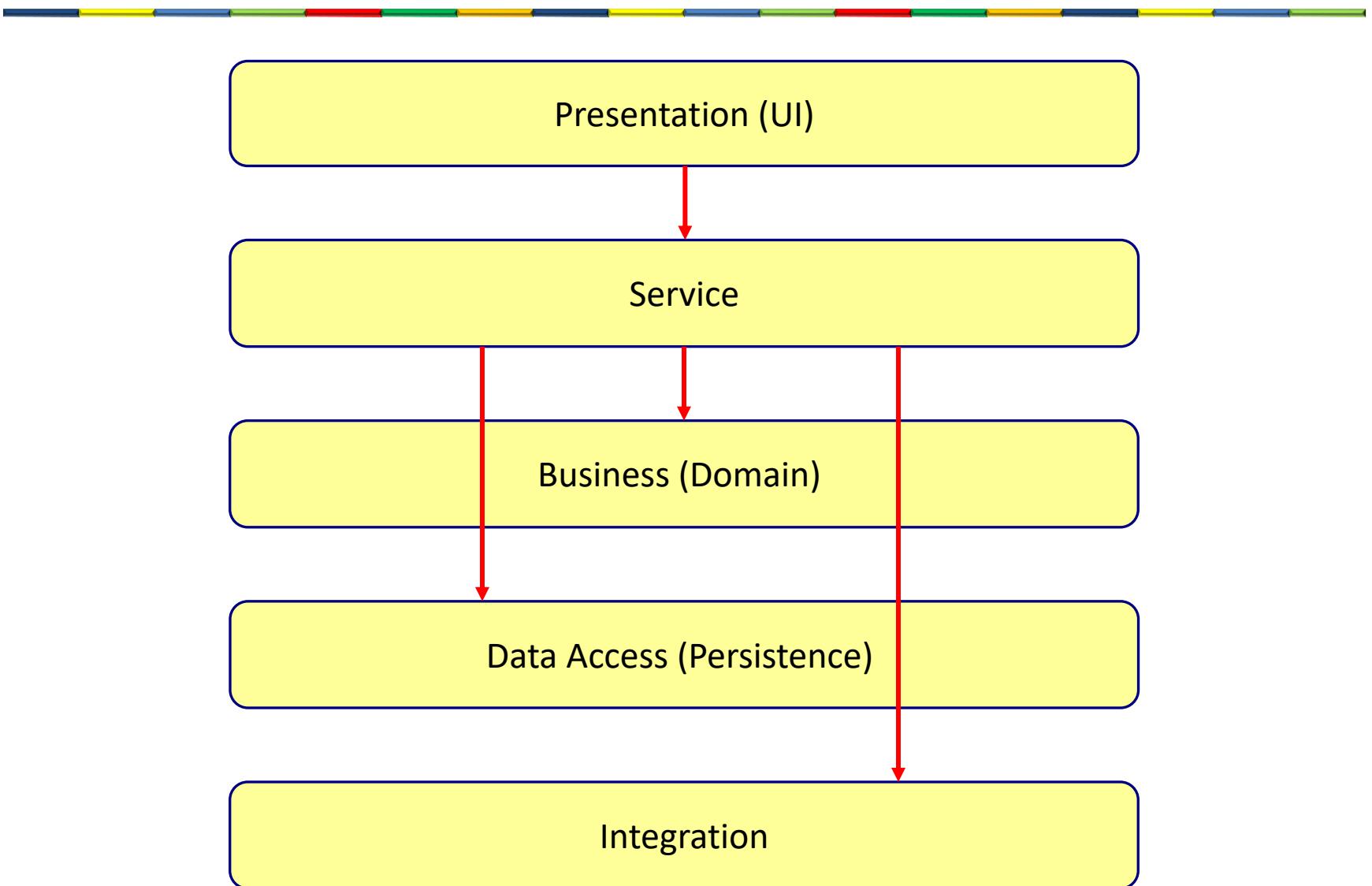
4 layered architecture



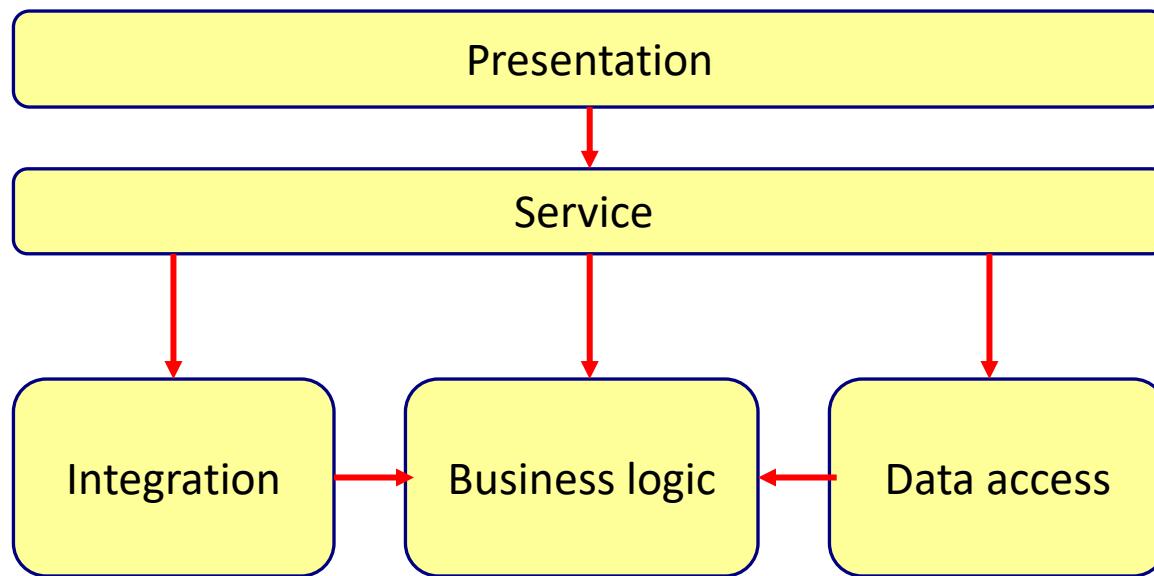
Open and closed layers



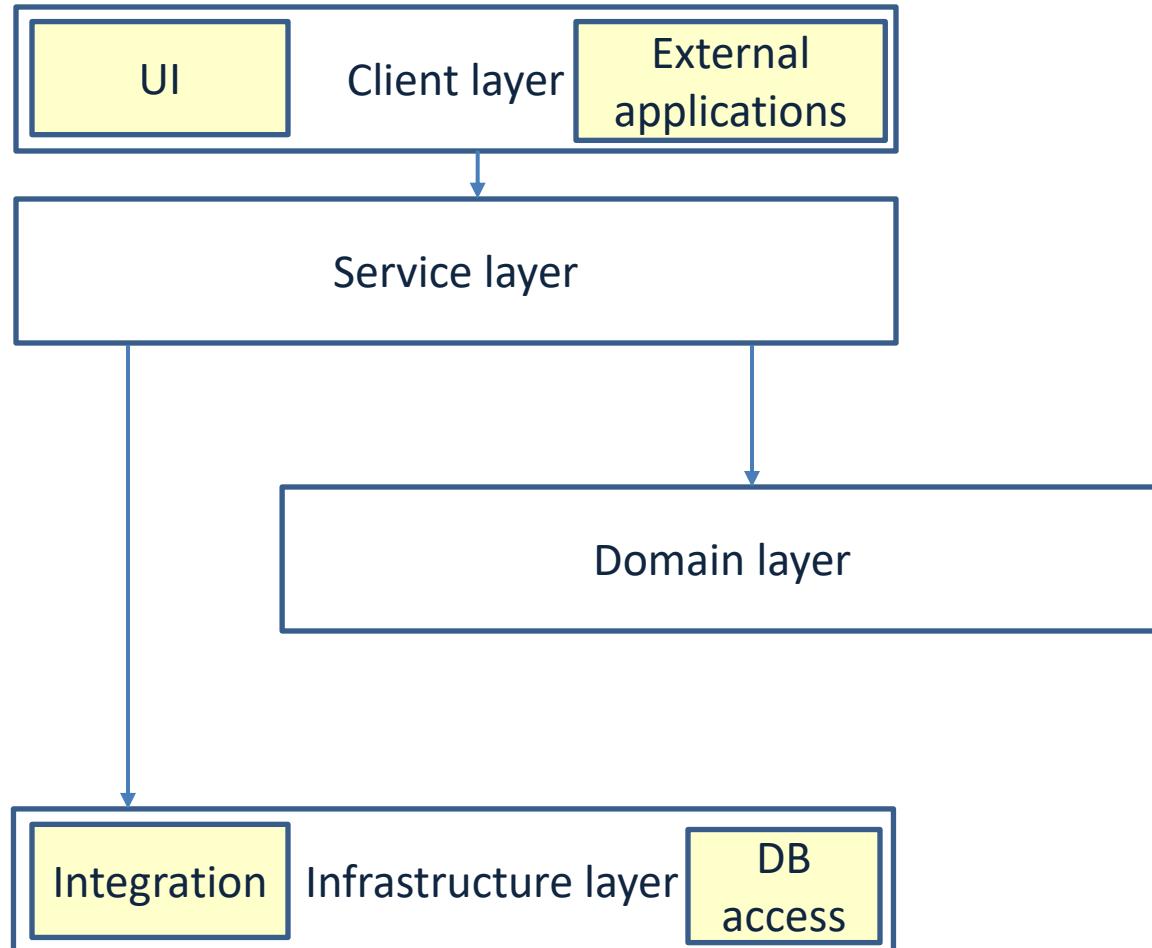
5 layered architecture



Layered architecture



Layered architecture



Layering

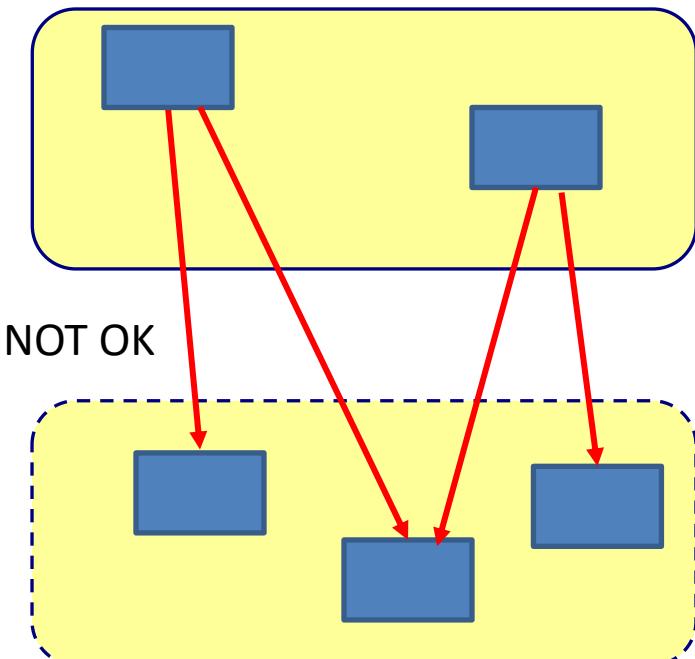
- Benefits
 - Layers can be distributed
 - Separation of concern
 - Different skills required in each layer
 - Easy to modify
 - Easy to test
- Drawbacks
 - Development effort can increase
 - Performance can become an issue



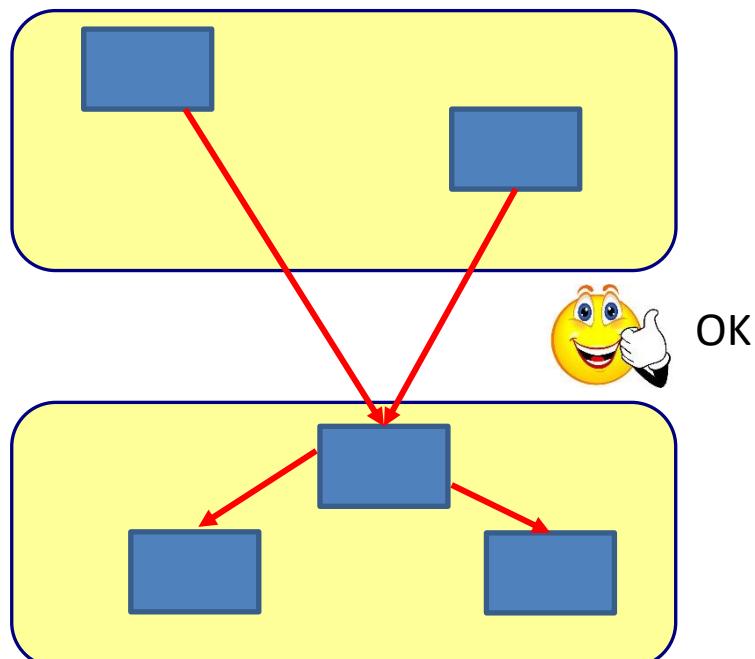
Layering anti patterns

- Too much layers
- No logic in layers
- No encapsulation of layers

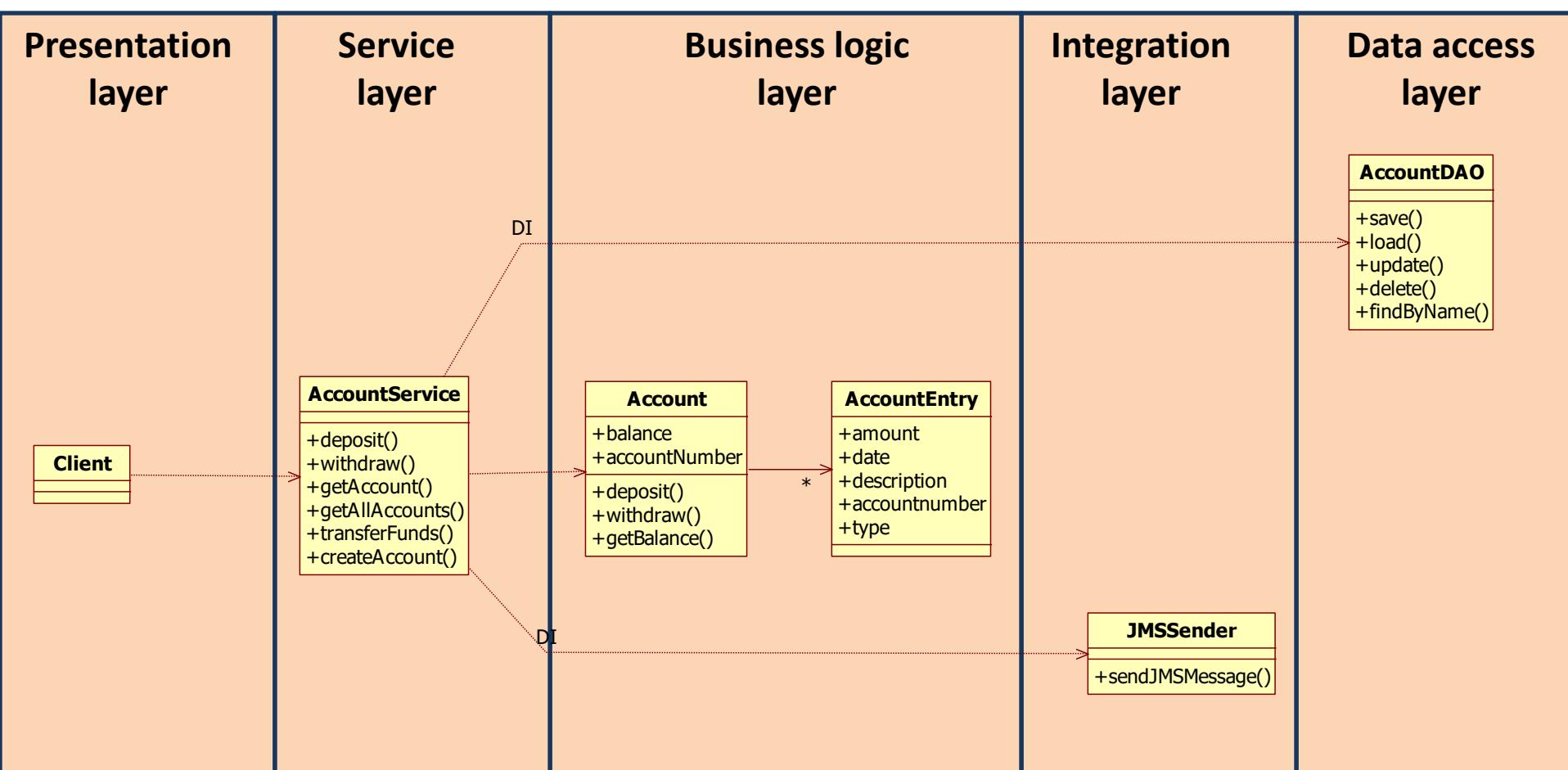
White box layering



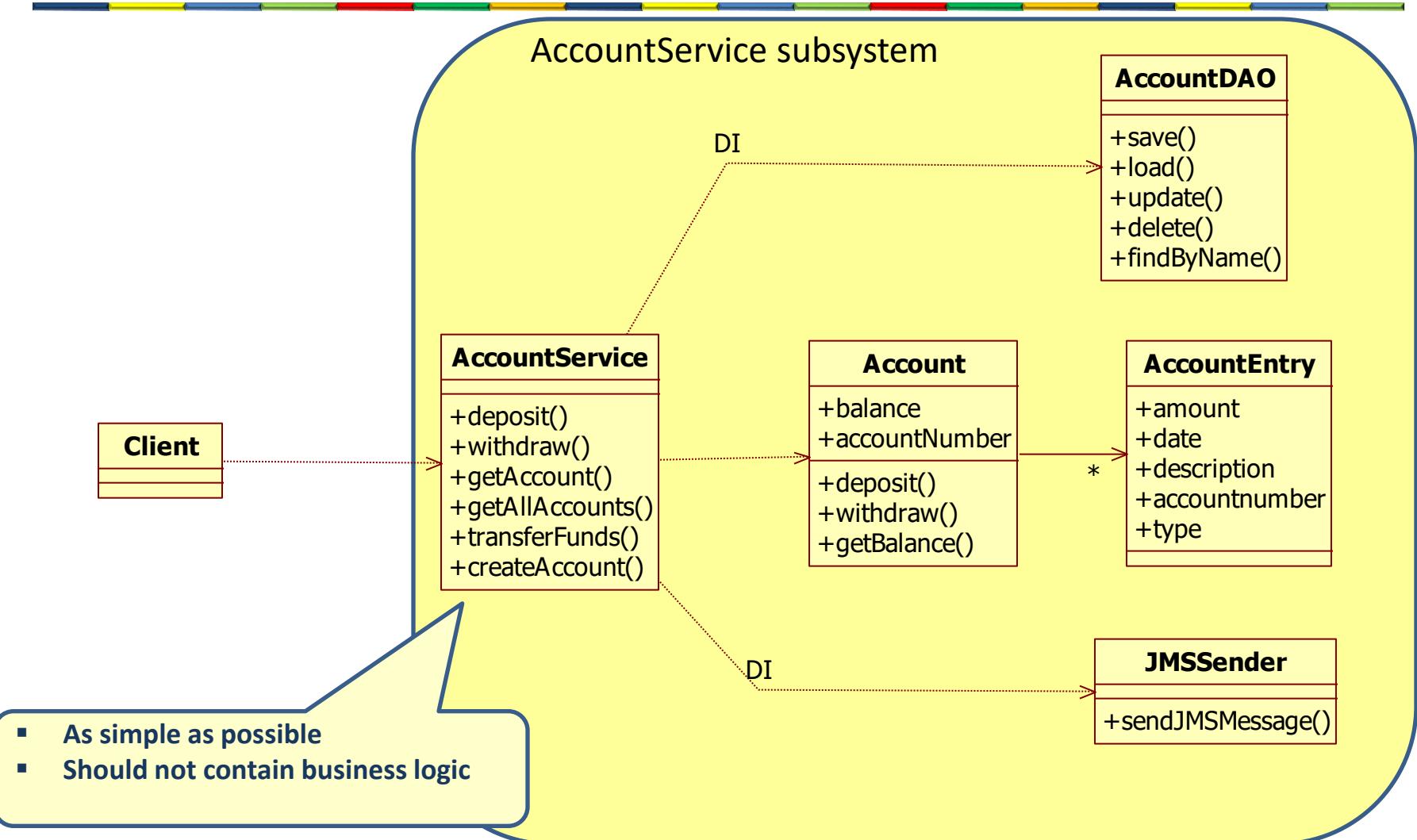
Black box layering



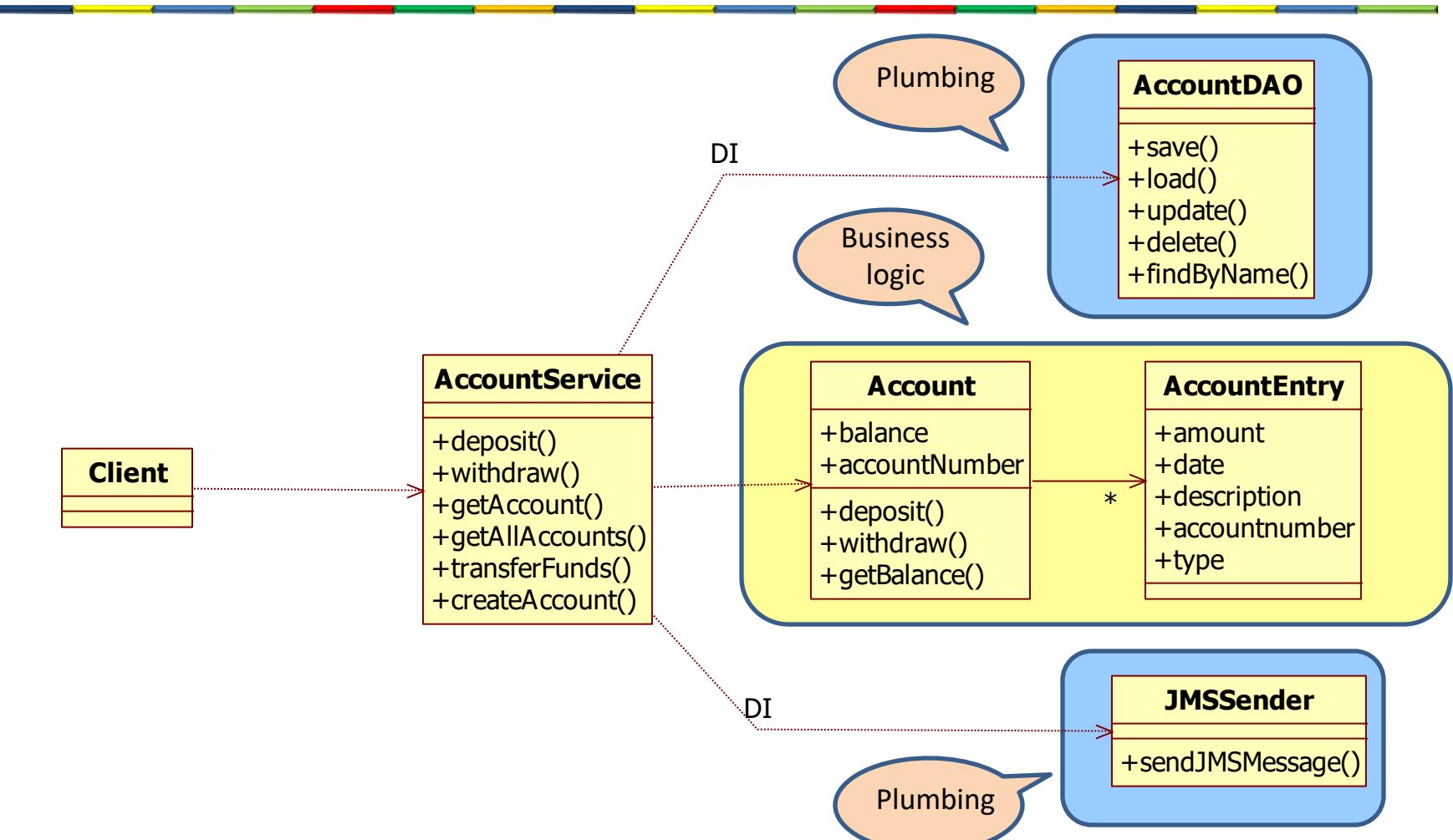
Application layers



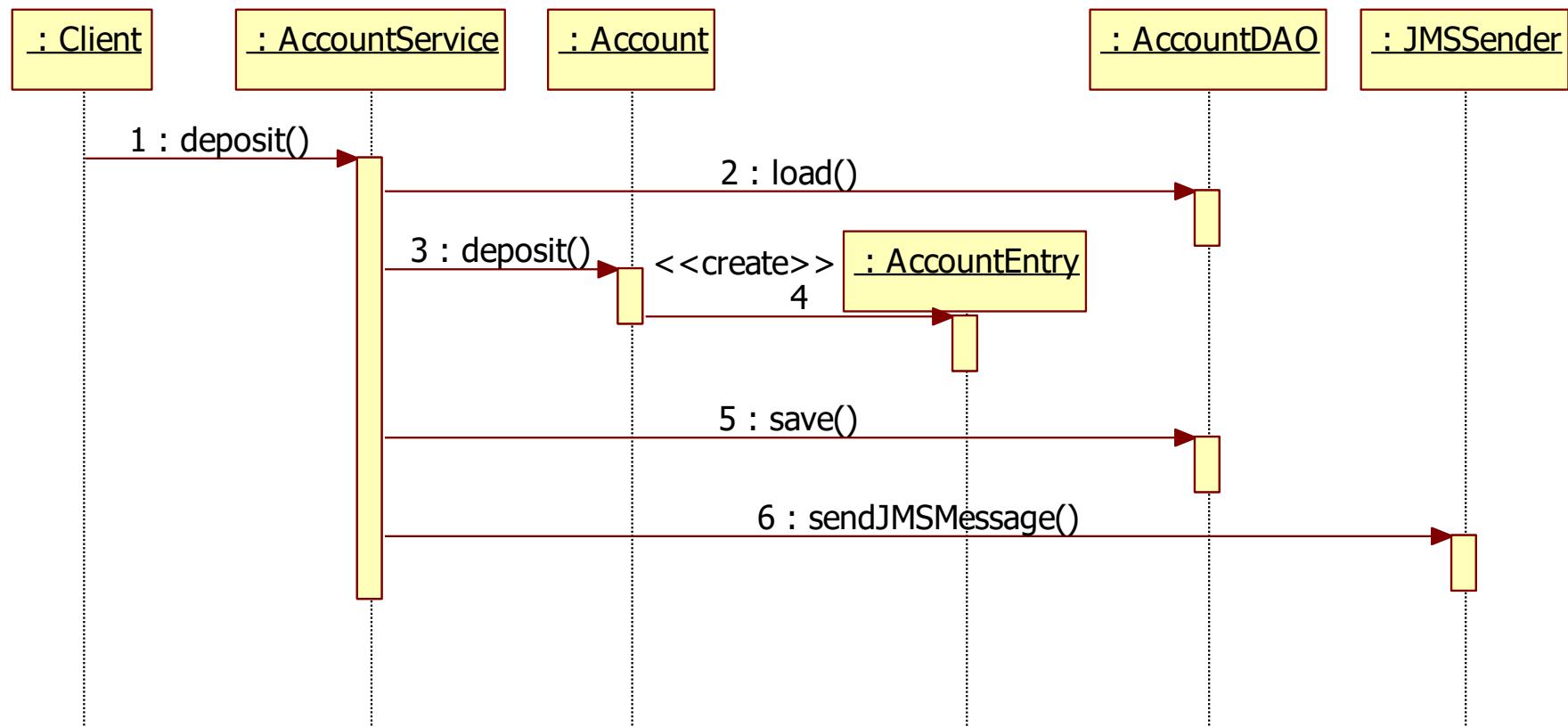
Entry of a complex subsystem



Separation of concern



Service object

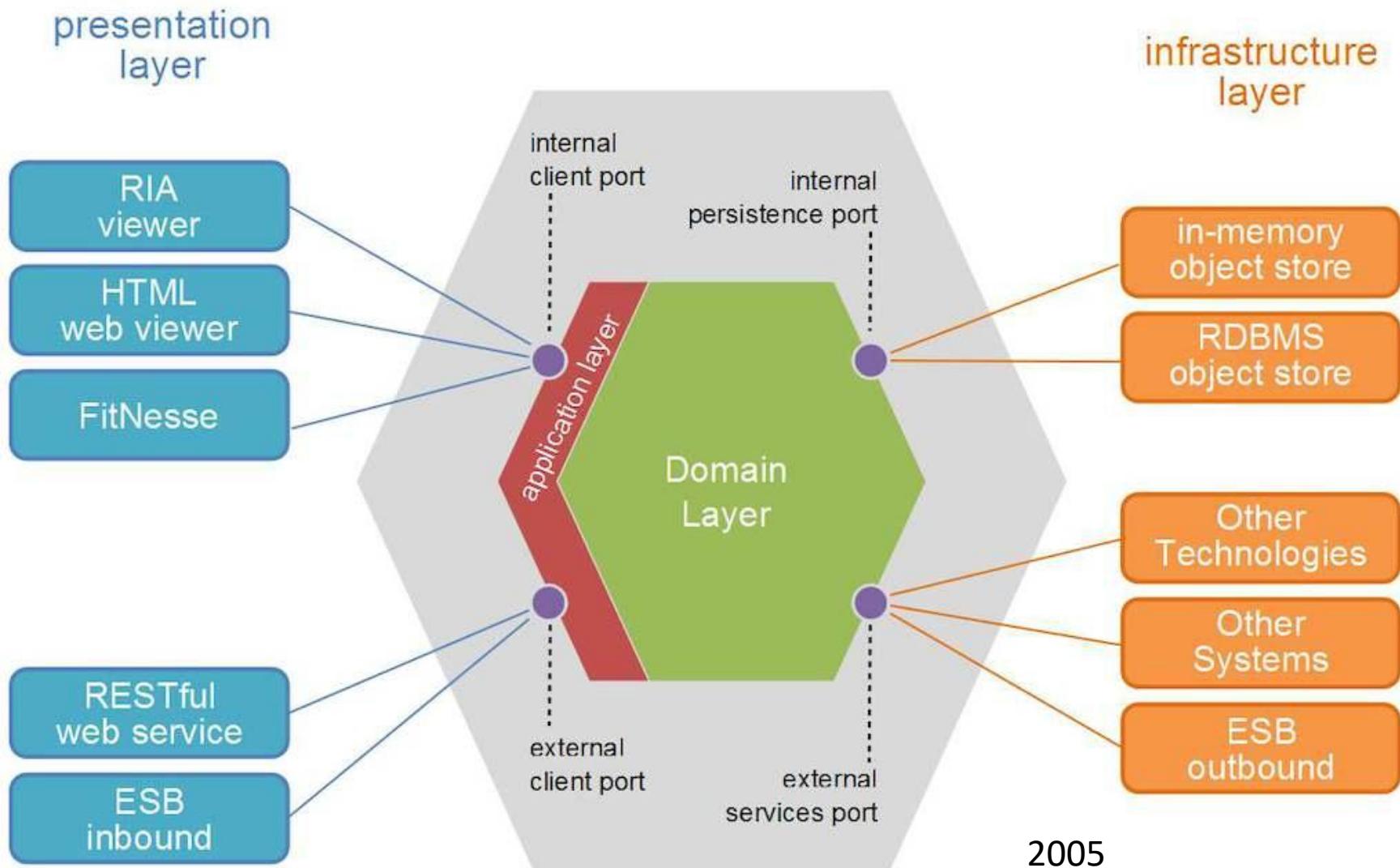


Main point

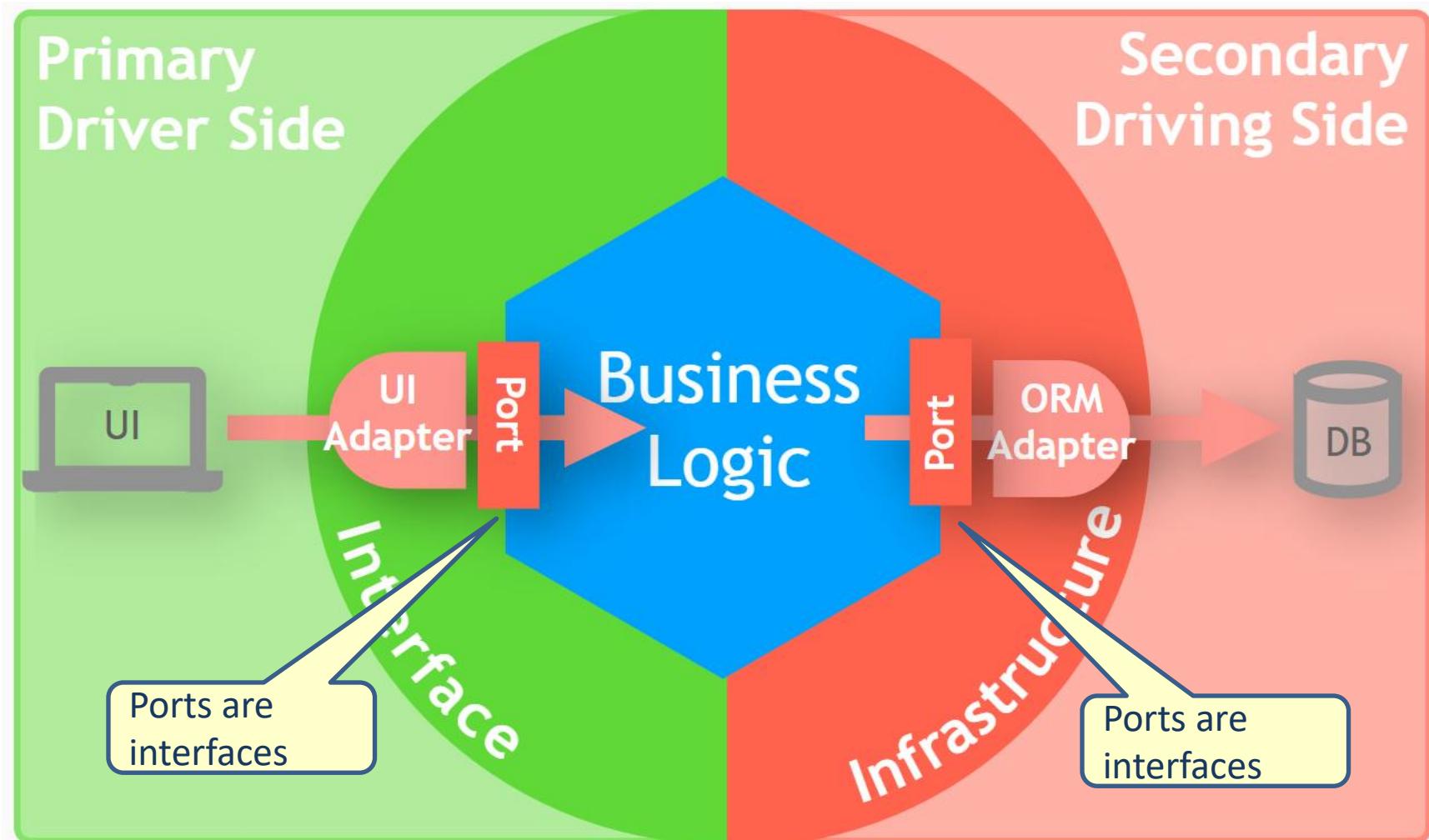
- An enterprise back-end system is typically divided in different layers.
- Life is found in layers



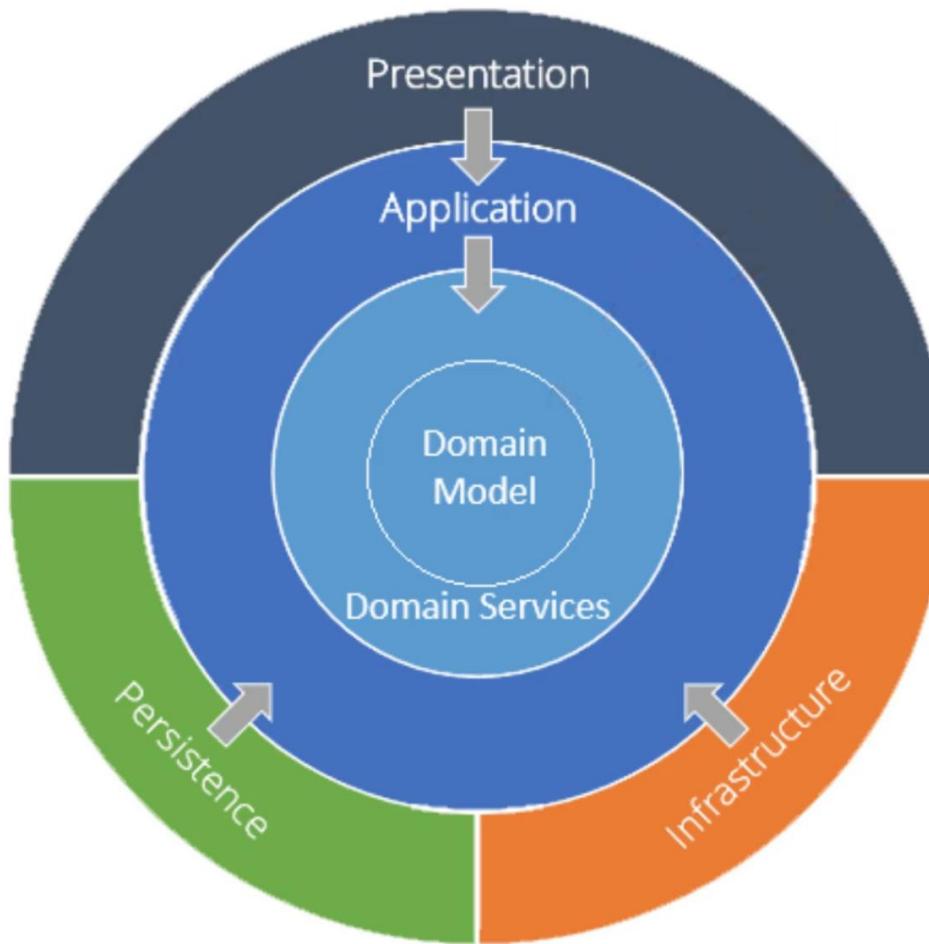
Hexagonal architecture



Port and adapter architecture

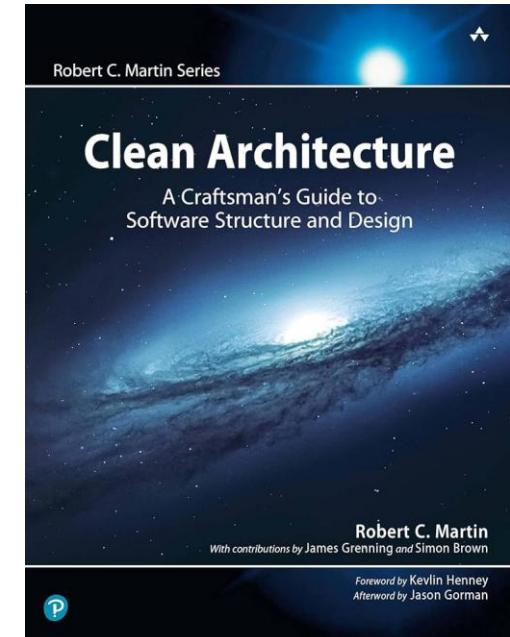
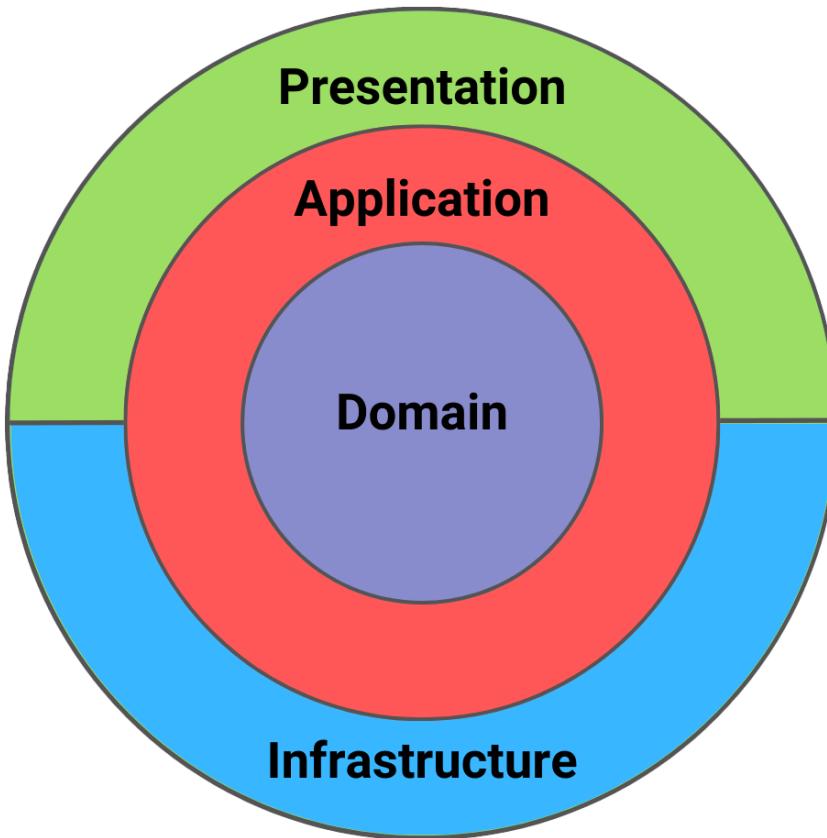


Union architecture



2008

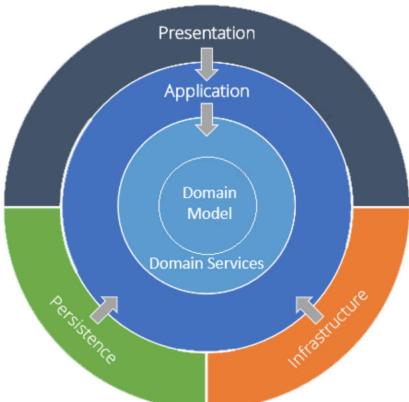
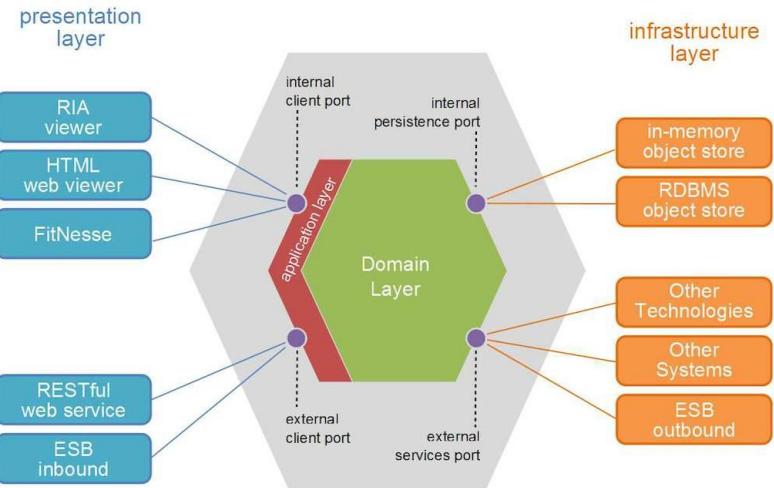
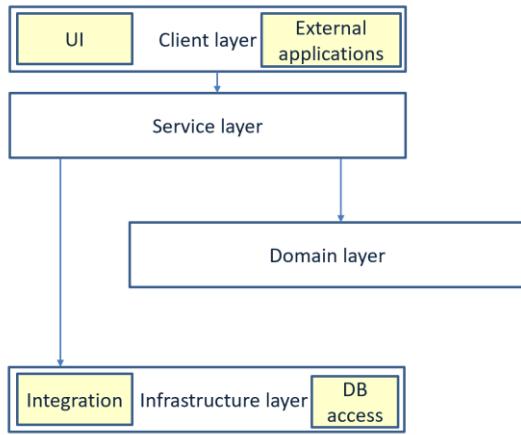
Clean architecture



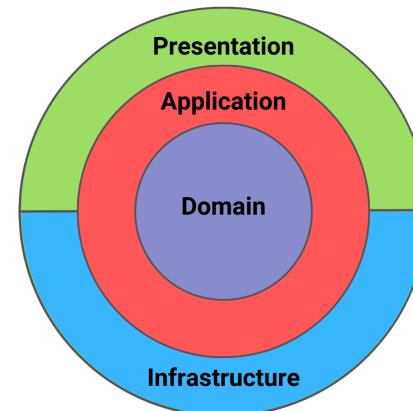
2017



Layered, hexagonal, onion and clean architecture

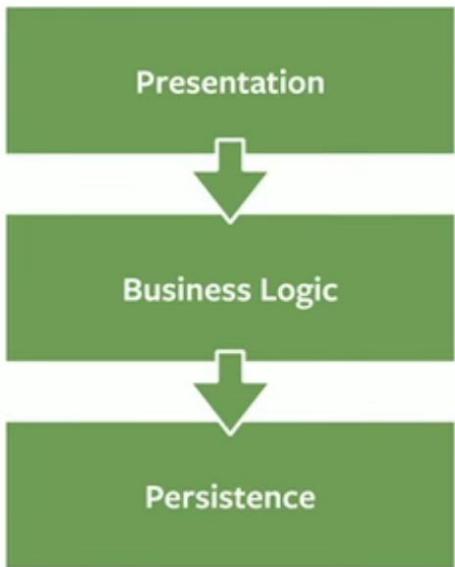


Domain is the core

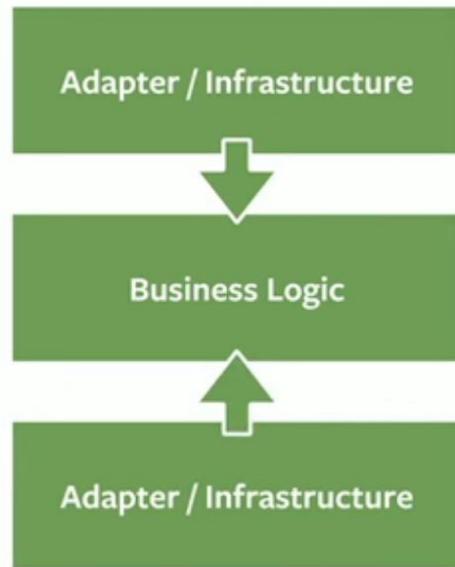


Layered, hexagonal, onion and clean architecture

Layered Architecture



Hexagonal- / Onion-Architecture

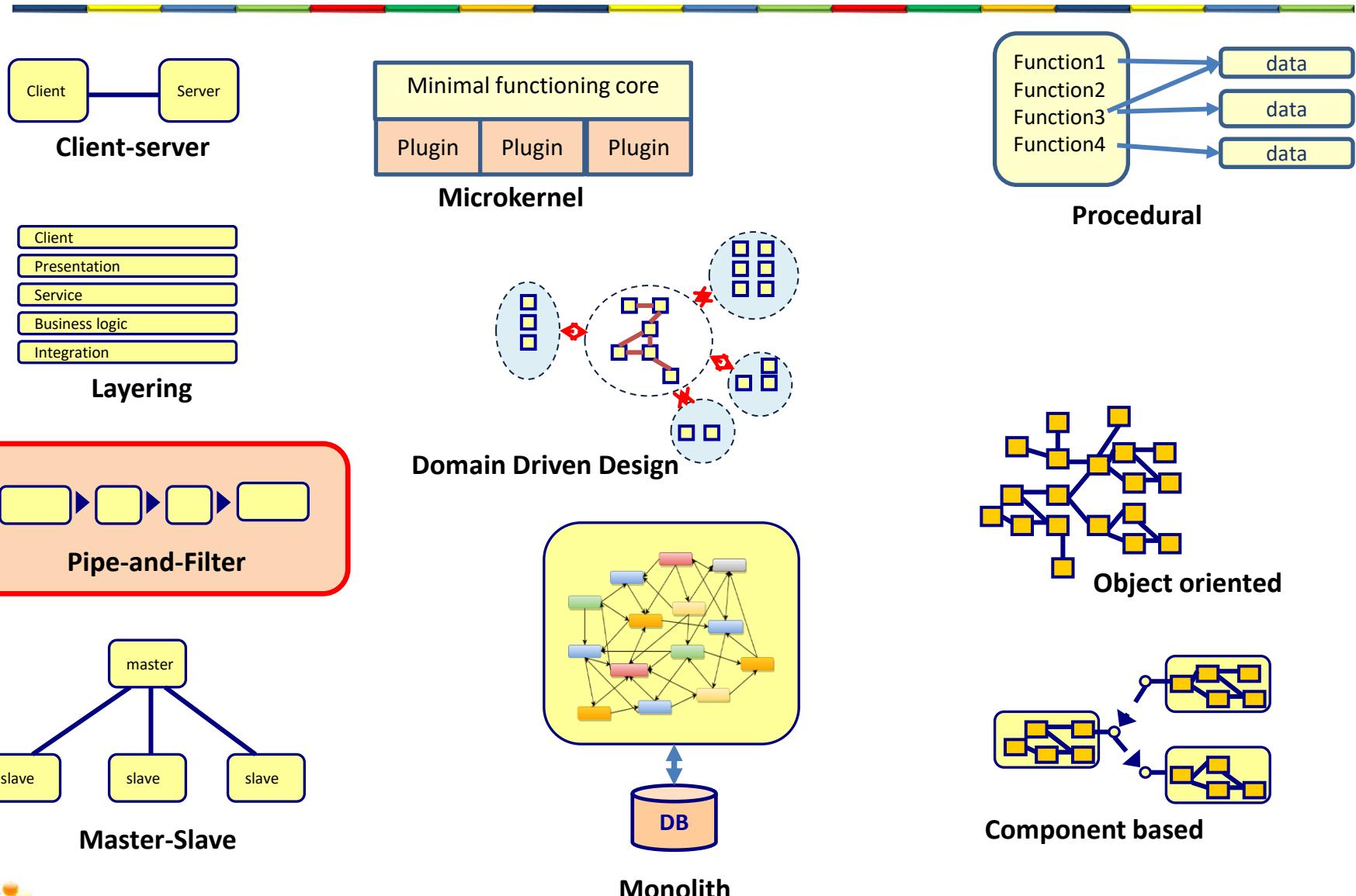


Main point

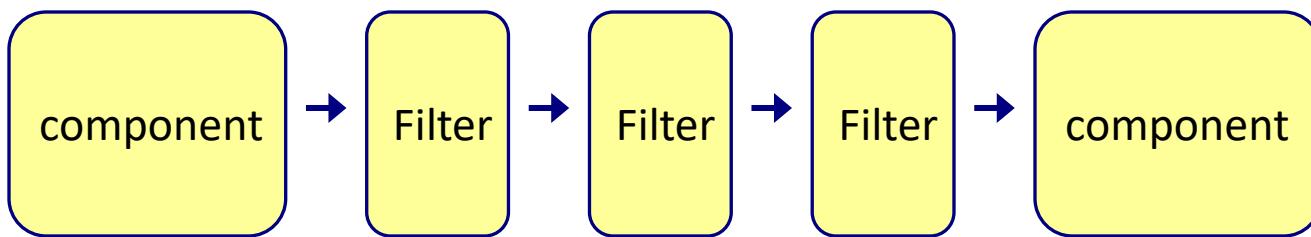
- The domain classes are never aware of technical “plumbing” classes. This gives many different advantages.
- By diving deep into pure consciousness, one gains support of all the laws of nature without needing to know or to be aware of all different details of your life and your world.



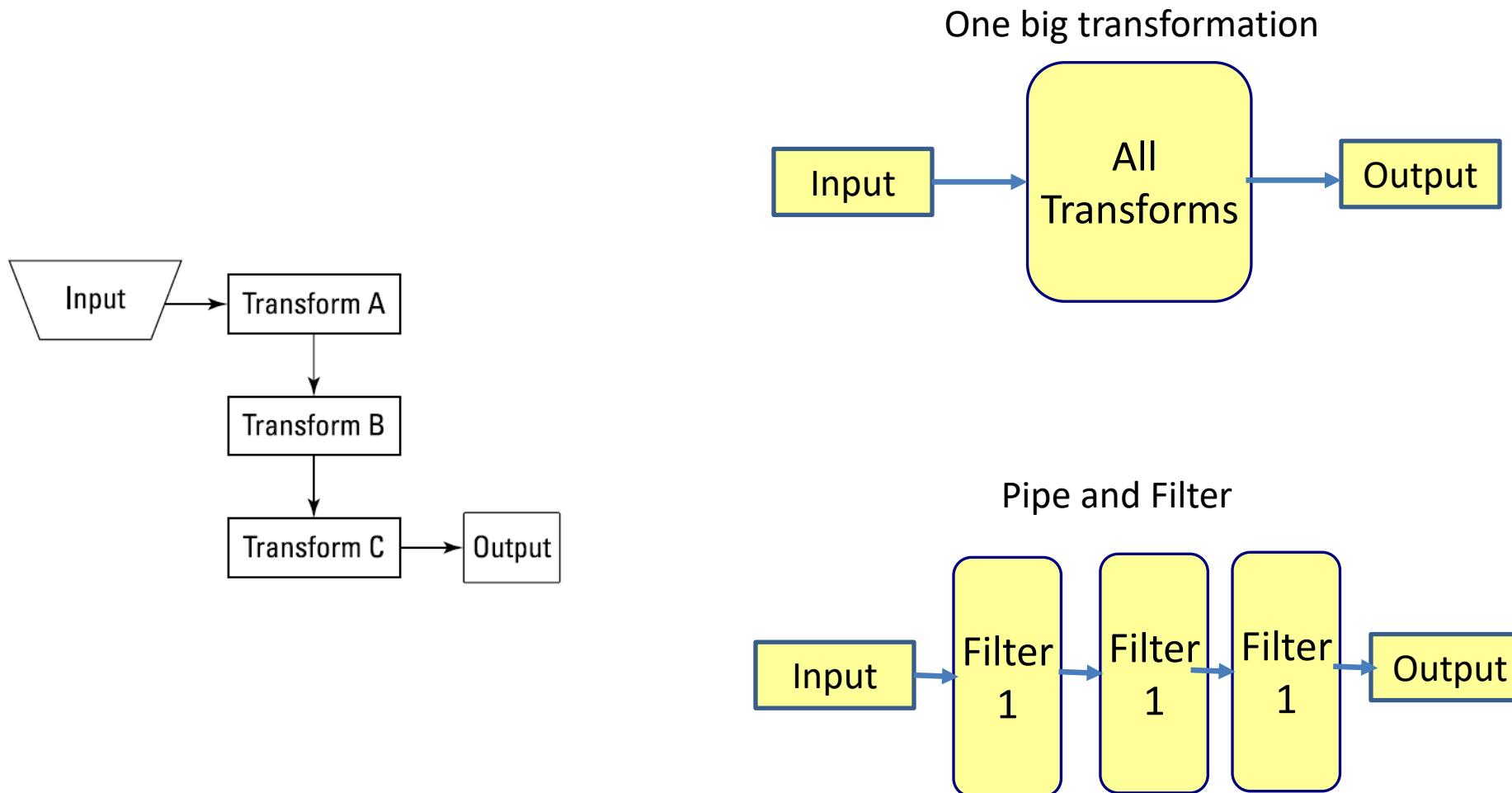
Architecture styles within an application



Pipe and Filter



Analyzing an Image Stream



Pipe and Filter

- Benefits

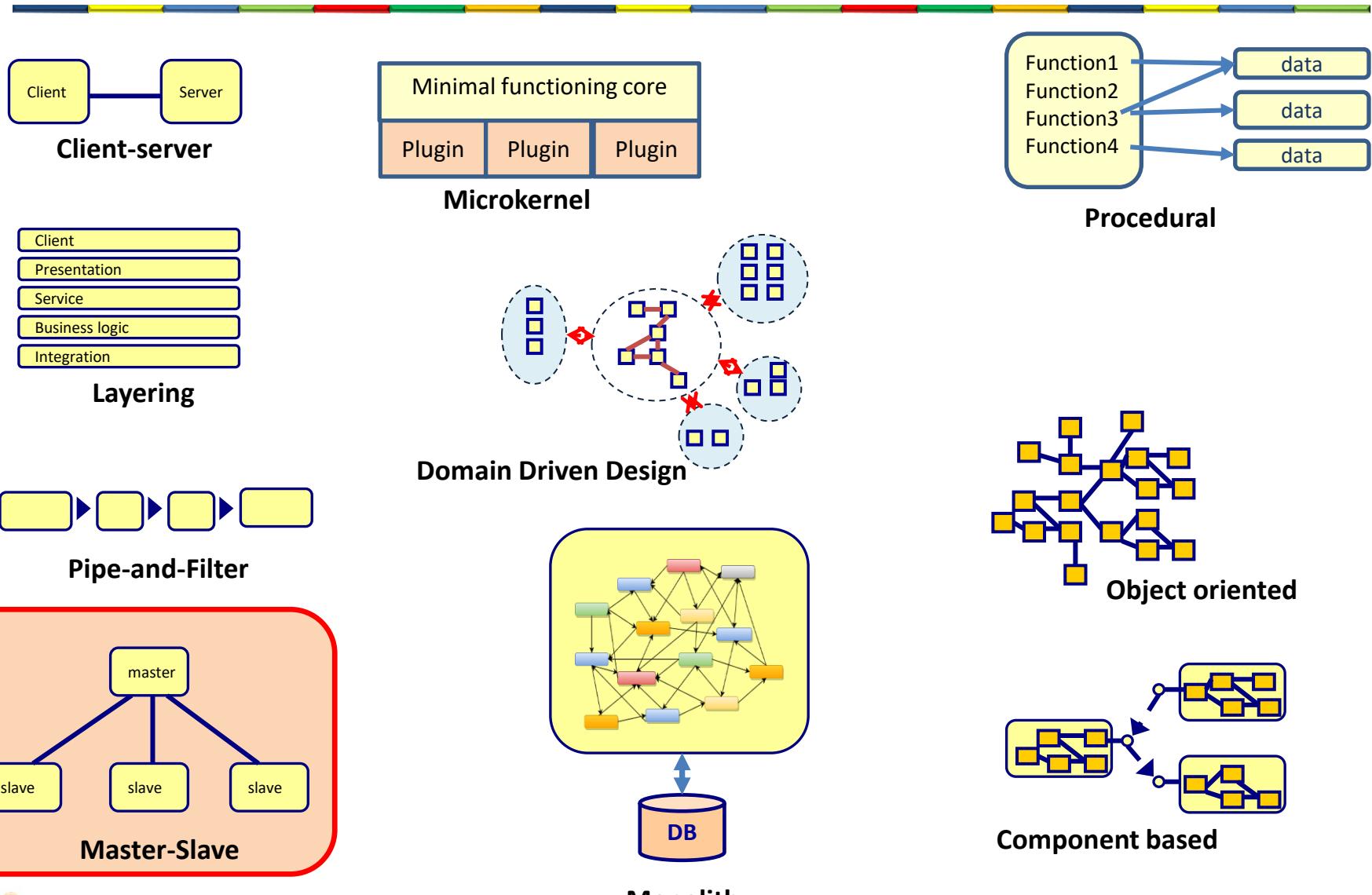
- Filters are independent
- Filters are reusable
- Order of filters can change
- Easy to add new filters
- Filters can work in parallel

- Drawbacks

- Works only for sequential processing
- Sharing state between filters is difficult

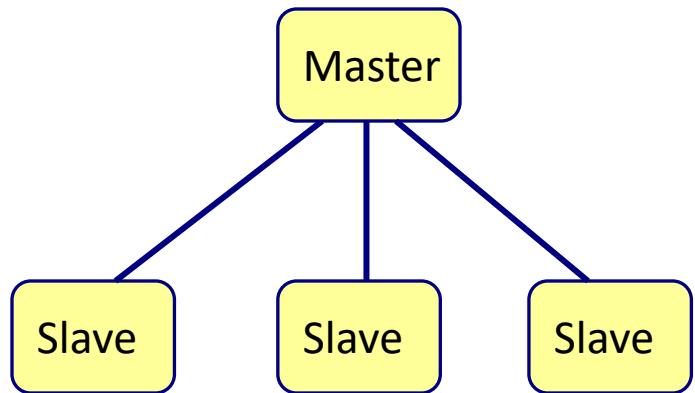


Architecture styles within an application



Master-Slave

- Master organizes work into distinct subtasks
- Subtasks are allocated to isolated slaves
- Slaves report their result to the master
- Master integrates results



Master slave

- Benefits

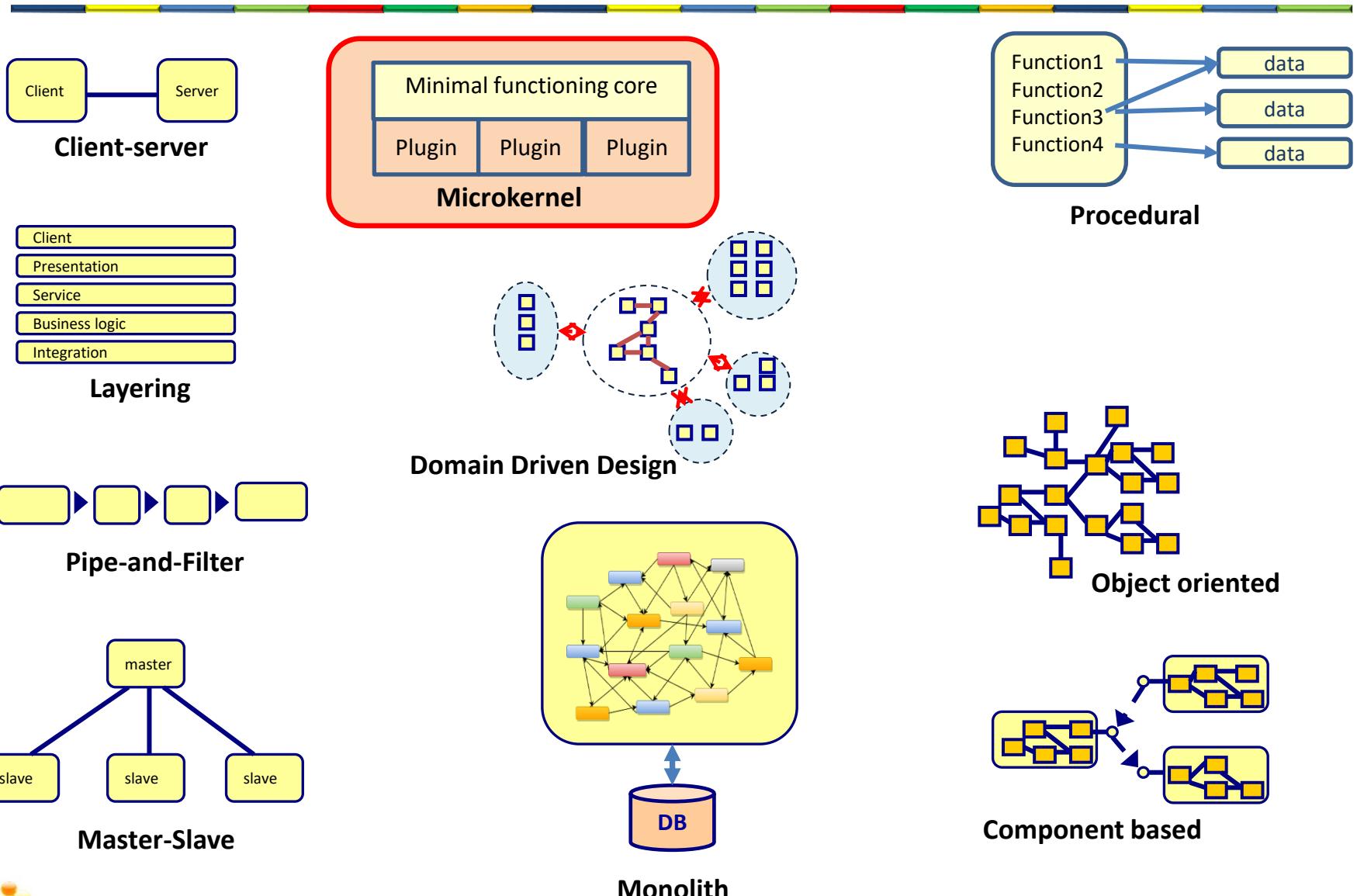
- Separation of coordination and actual work
- Master has complete control
- Slaves are independent
 - No shared state
- Easy to add new slaves
- Slaves can work in parallel
- Slaves can be duplicated for fault tolerance

- Drawbacks

- Problem must be decomposable
- Master is single point of failure

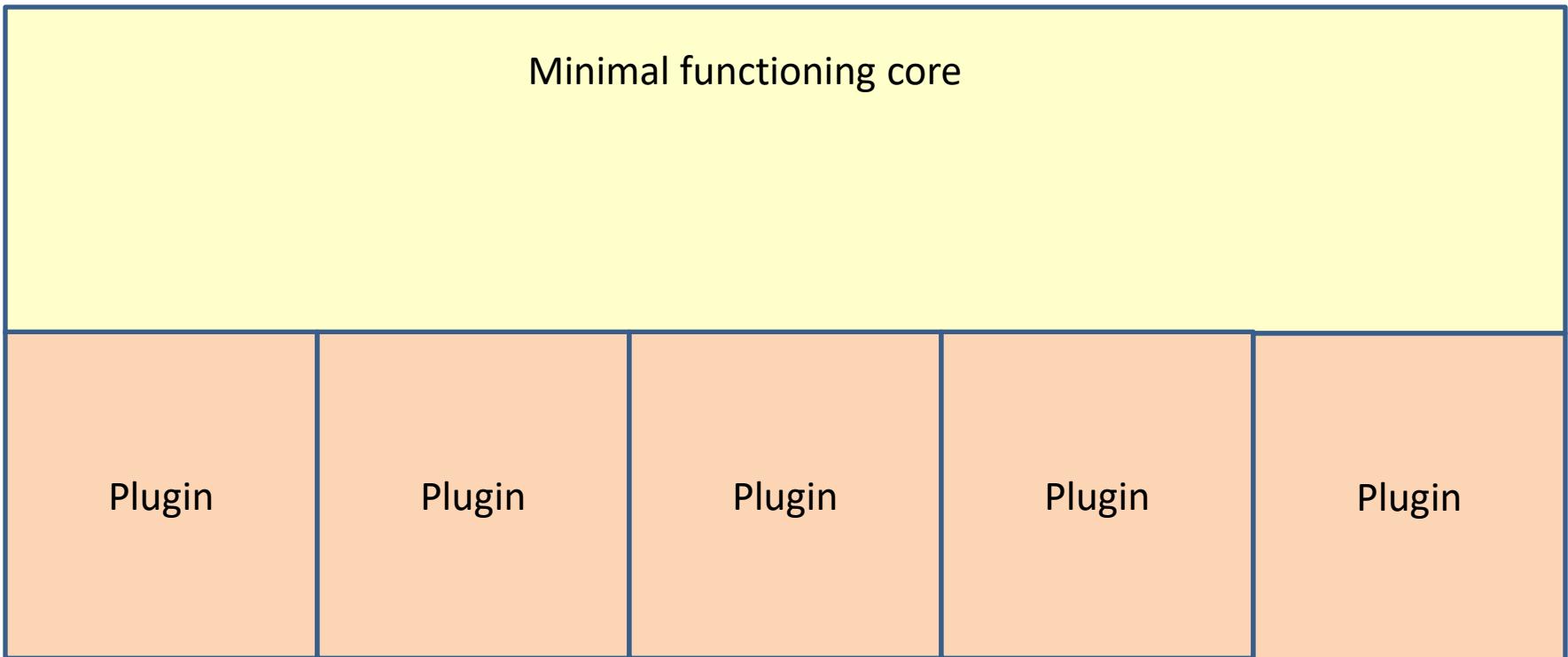


Architecture styles within an application



Microkernel

- Plugin application/framework



Microkernel examples

- Eclipse
 - With plugins



- Operating system
 - With drivers



Microkernel

- Benefits
 - Natural for product based apps
 - Extensibility
 - Flexibility
 - Separation of concern
- Drawbacks
 - Complexity



SUMMARY

What is software architecture?

The important stuff

Whatever that might be

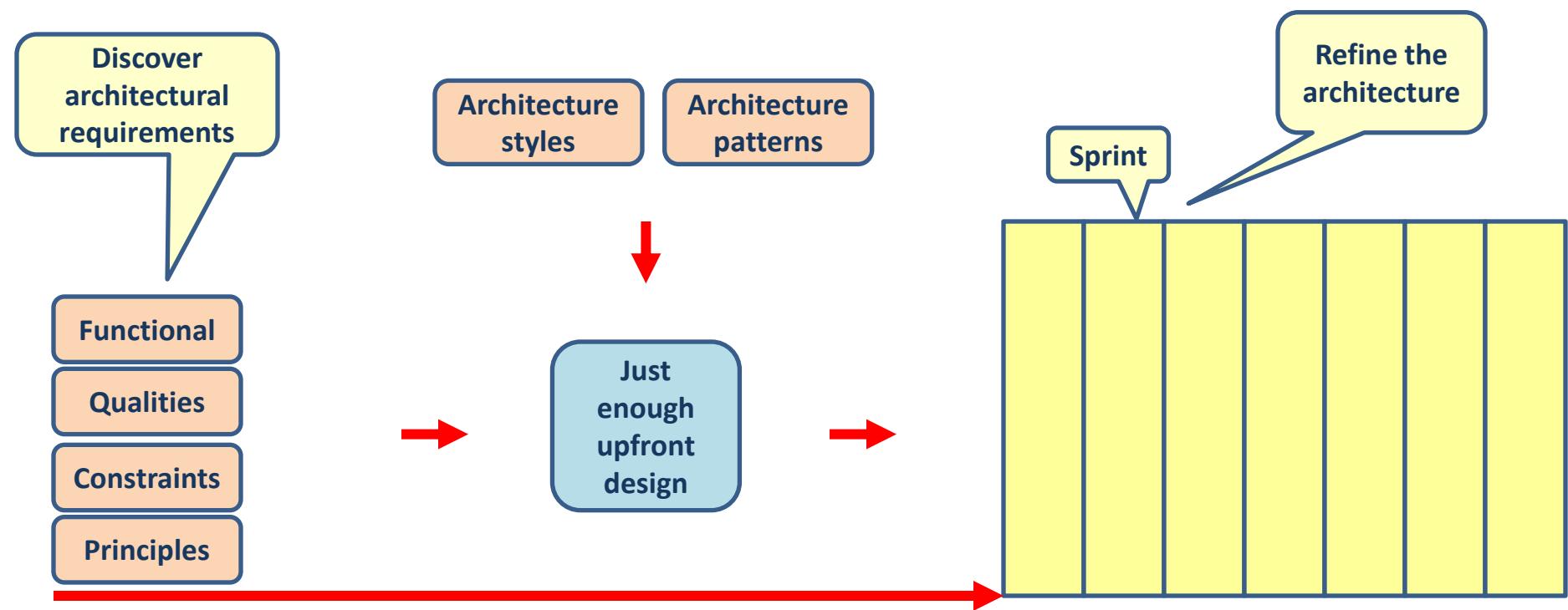


Agile architecture

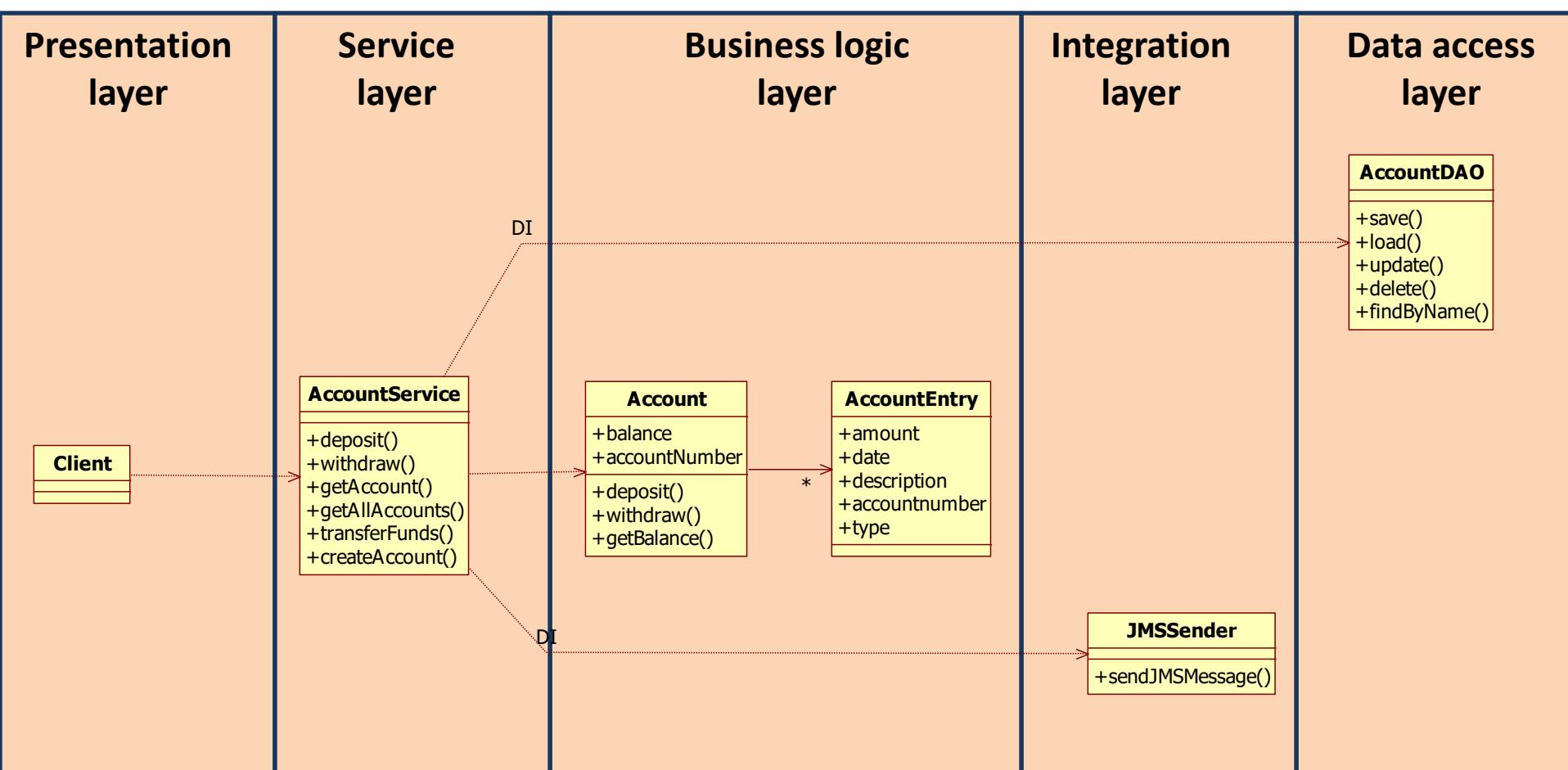
- Just enough upfront architecture
 - Refine the architecture later
- Keep your architecture flexible
- The architect is available during the whole project
- The architect also writes code
 - But not all the time
 - The architecture is grounded in reality
 - Works together with the developers
 - Architects should be master builders
- Proof the architecture in the first iteration(s)



Agile software architecture



Application layers



SOFTWARE ARCHITECTURE KEY PRINCIPLES



Key principle 1

- Software architecture is about making tradeoffs
 - Every decision has advantages and disadvantages
 - There is no silver bullet
 - The architecture is never ideal
 - You cannot read this is a book
 - There is no fixed template you can follow
 - The answer is always: **It depends**



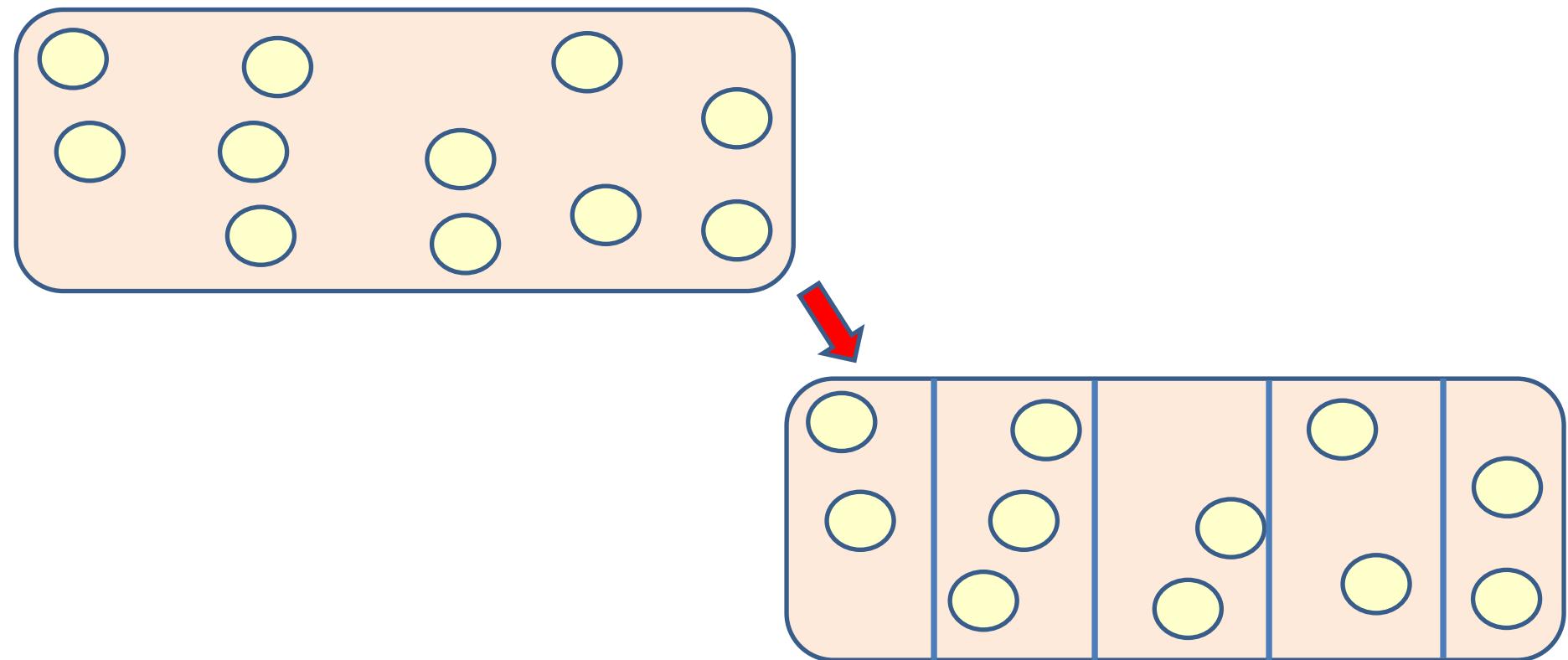
Key principle 2

- Most important architecture principles
 - Keep it simple
 - Keep it flexible
 - Loose coupling
 - High cohesion, low coupling
 - Separation of concern
 - Information hiding
 - Principle of modularity
 - Open-closed principle



Key principle 3

-
- When something becomes too complex, divide it into simpler parts



Connecting the parts of knowledge with the wholeness of knowledge

1. Software architecture defines all important aspects of a system.
 2. The architecture decisions are based on the functional requirements, the qualities, the business constraints and the architecture principles
-

3. **Transcendental consciousness** is the natural experience of pure consciousness, the home of all the laws of nature.
4. **Wholeness moving within itself:** In unity consciousness, one appreciates and enjoys the underlying blissful nature of life even in all the abstract expressions of pure consciousness.



Lesson 3

DOMAIN DRIVEN DESIGN



Building software

- Before you can start writing code you have to understand the domain first

The hardest single part of building a software system is deciding precisely what to build.

Fred Brooks - "No Silver Bullet" 1987

If you don't get the requirements right, it does not matter how well you do anything else.

Karl Wiegers

It is the developer's (mis)understanding, not the expert knowledge, that gets released in production.

Alberto brandolini



Domain

What a business does and the world it does it in.

Banking

Loan

Account

Mortgage

Insurance

Deposit

Trade

Payment

Settlements

Risk mgm.

Hospital

Patient

Doctor

Treatment

Insurance

Payment

Appointment

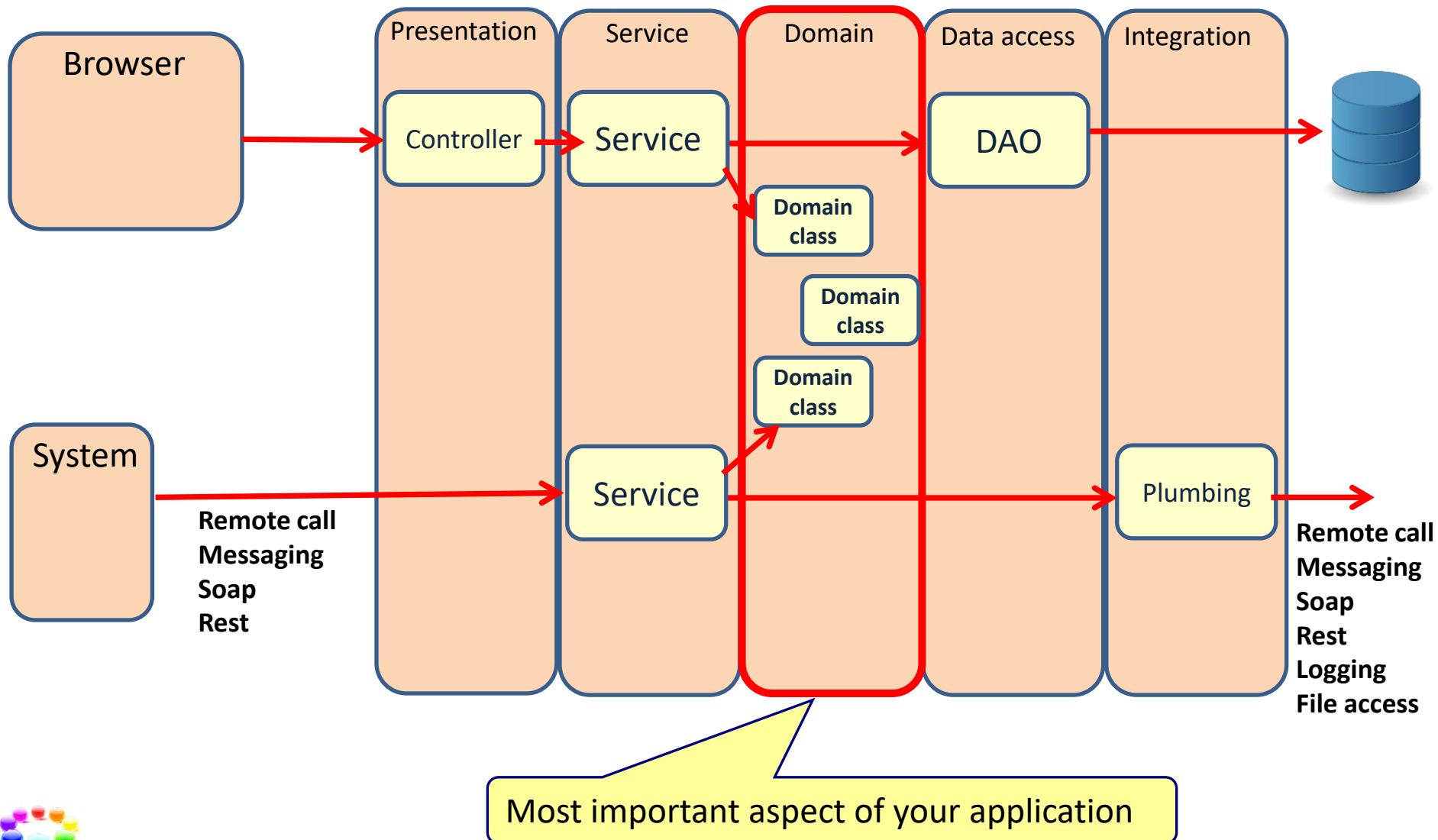
Medication

Illness

Surgery

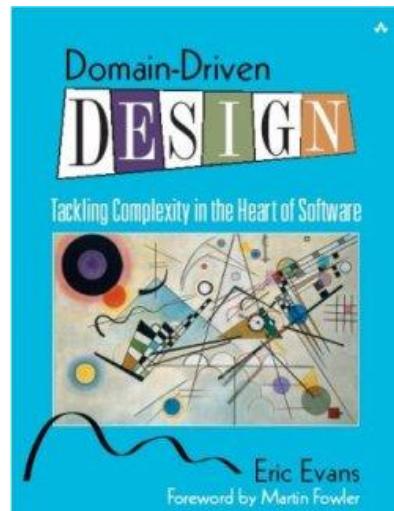


Domain Driven Design



What is Domain Driven Design?

- An approach to software development where the focus is on the core **Domain**.
 - We create a **domain model** to communicate the domain
 - Everything we do (discussions, design, coding, testing, documenting, etc.) is based on the domain model.



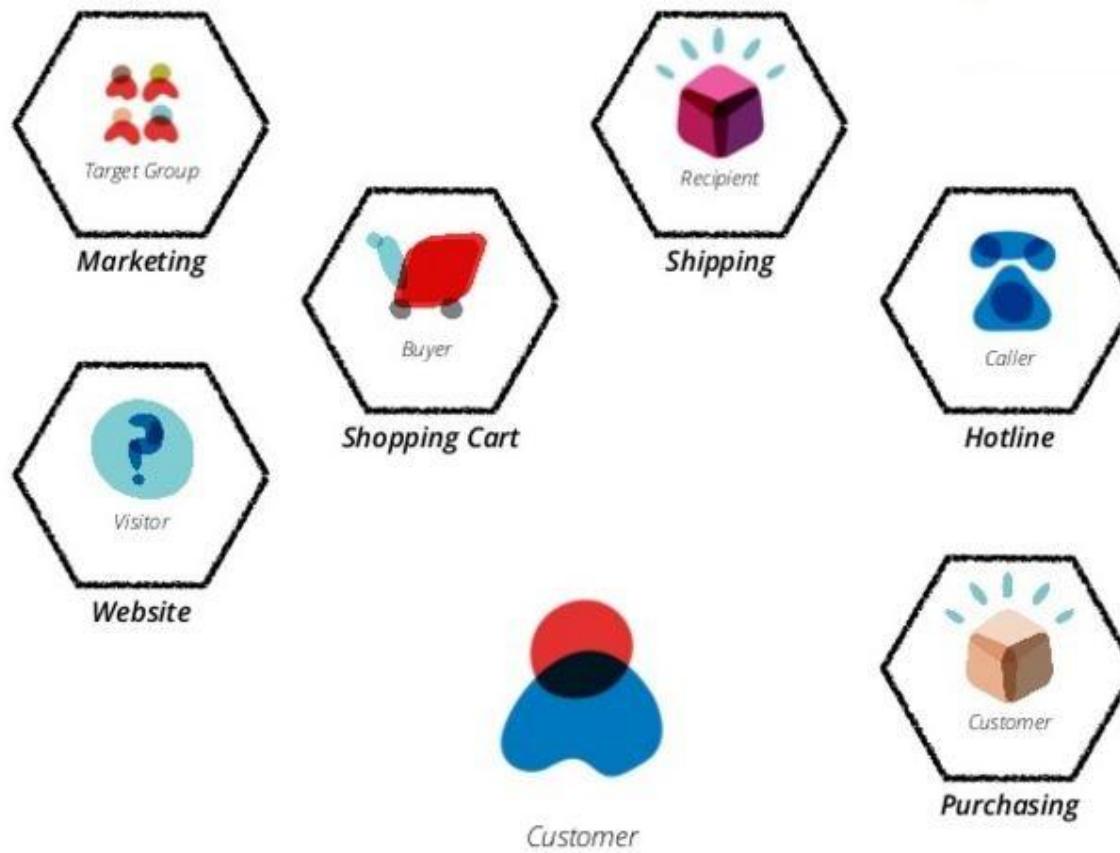
Principles of Domain Driven Design

- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain



Common language

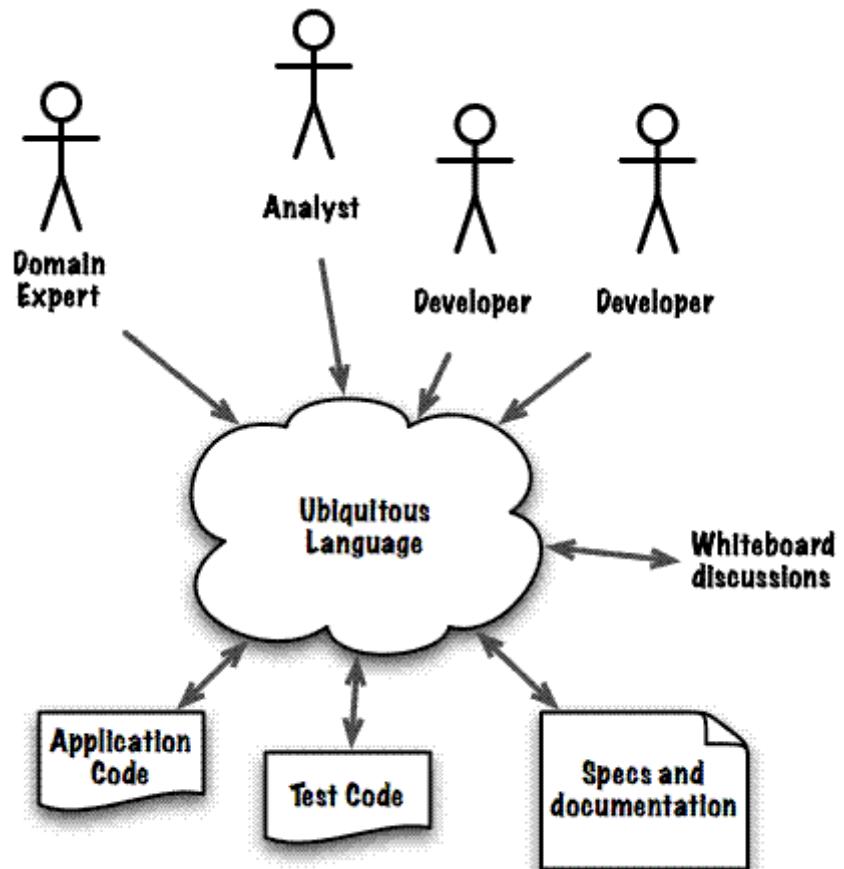
- Different people from the business use different names for the same thing.



We need a common language

Ubiquitous Language

- Language used by the team to capture the concepts and terms of a specific core business domain.
 - Used by the people
 - Used in the code
 - Used everywhere



Principles of Domain Driven Design

- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain



Model



Cottonwoods -Maple House Model



- More complexity -> More modeling
 - Higher level of abstraction
 - Allows for visualization
 - Vehicle of communication



Where is the domain model?

- In diagrams and documentation
- Part of collaboration and discussions
- In code

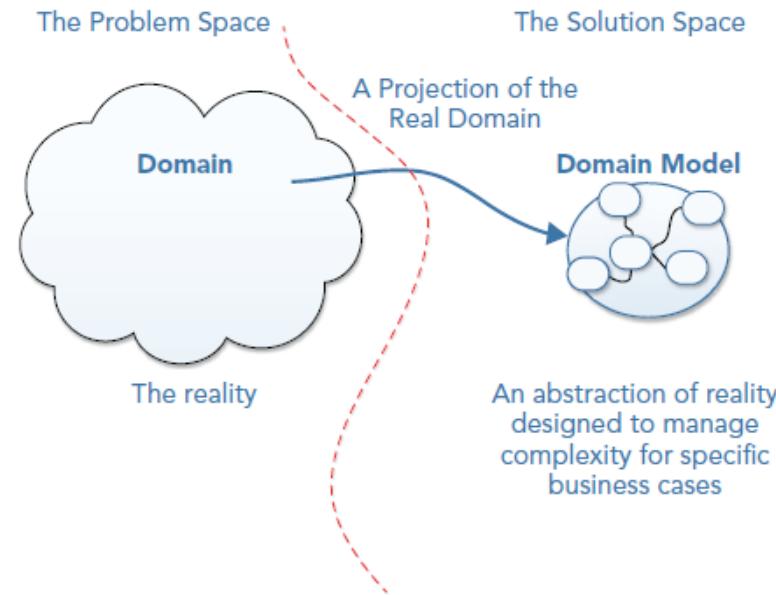
A domain model is not a particular diagram; it is the idea that the diagram is intended to convey.

Eric Evans



Domain model

- Extracts domain **essential** elements
 - Relevant to a specific use
- Layers of **abstractions** representing **selected** aspects of the domain
- Contains **concepts** of importance and their **relationships**



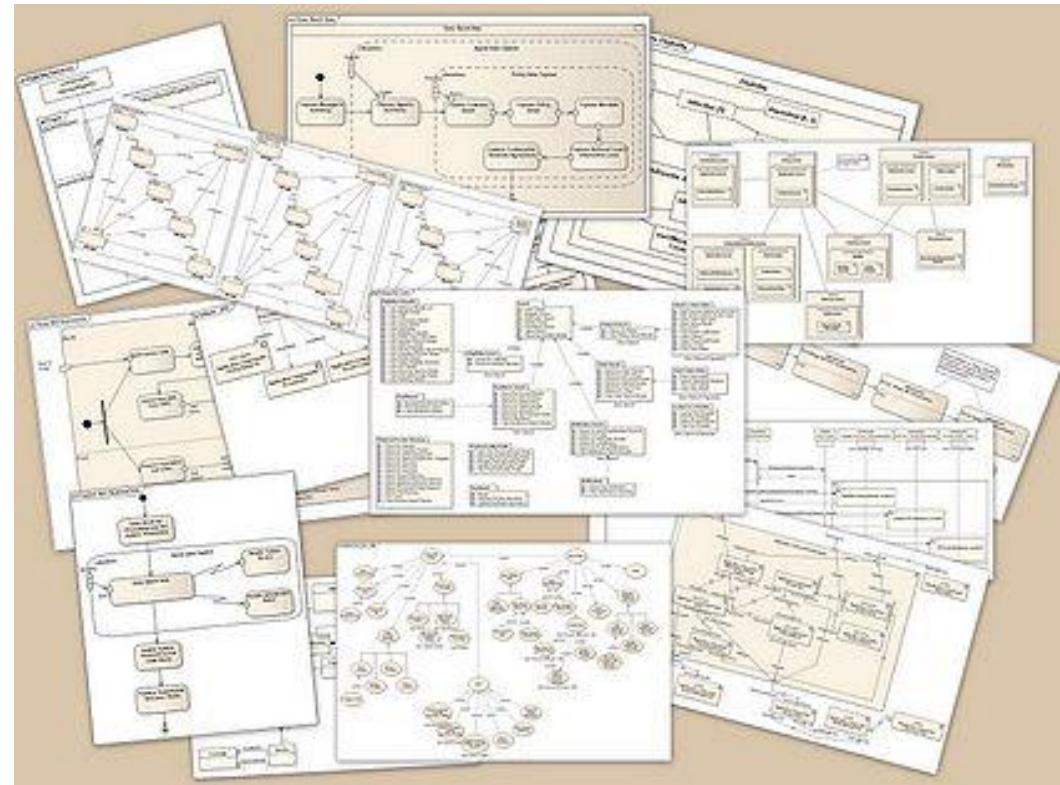
Domain model

- Simplification of reality
- Area of interest

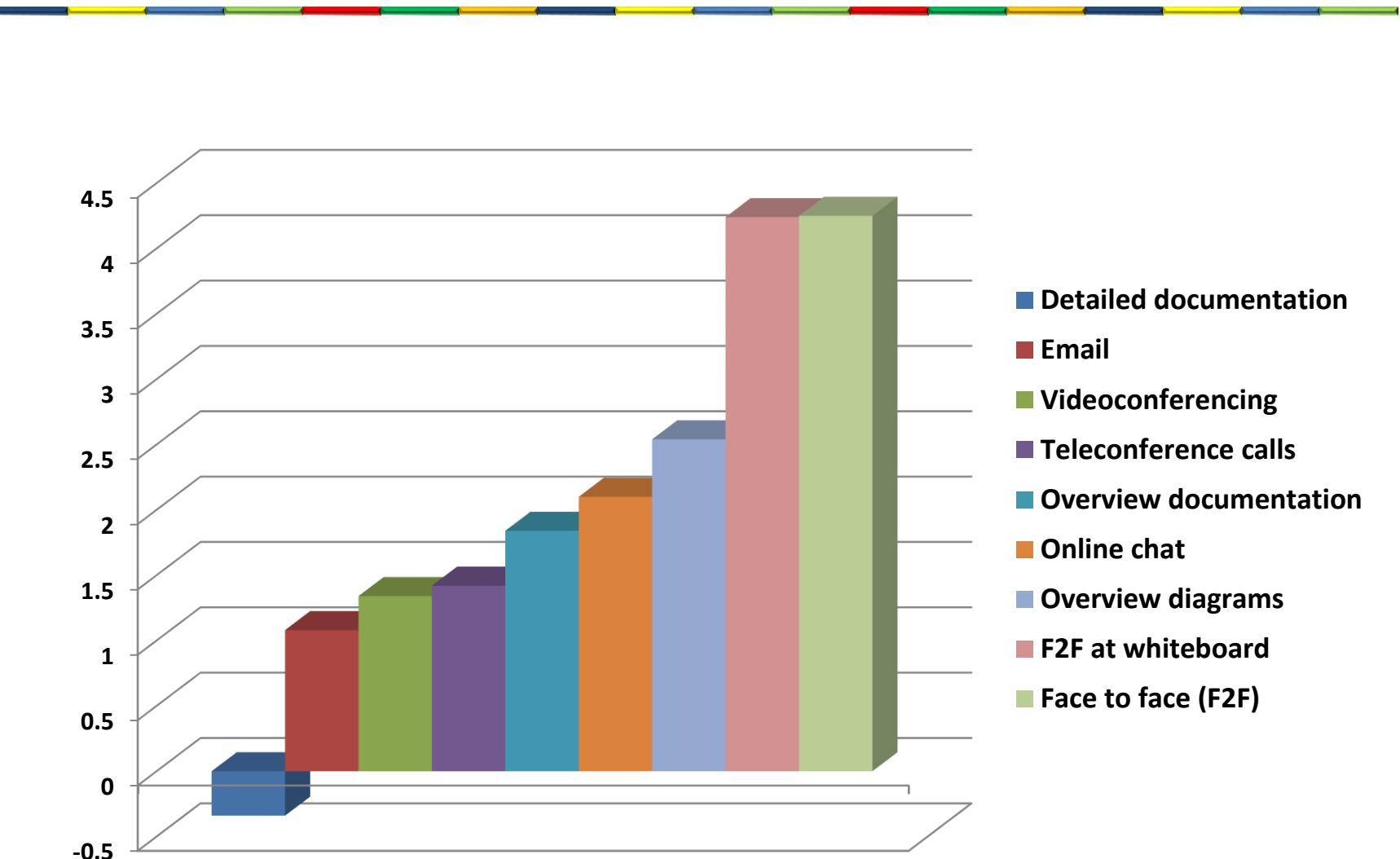


Structure of the domain model

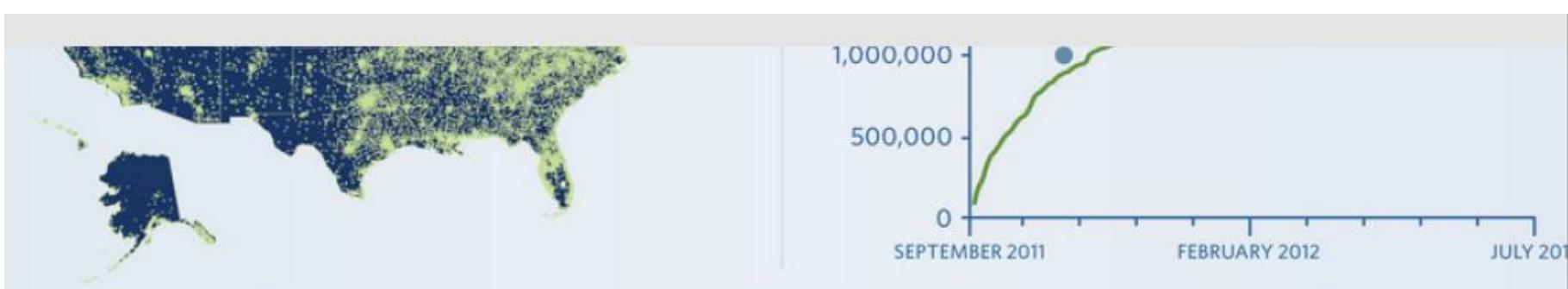
- A domain model is not a particular diagram
- Use the format that communicates the best
 - Diagram
 - Text
 - Code
 - Table
 - Formula



Effectiveness of communication



Effective communication



TOP 5 PETITION CATEGORIES



SIGNATURES

150,945
MOST SIGNATURES
ON A SINGLE PETITION

406
AVERAGE SIGNATURE
PER HOUR

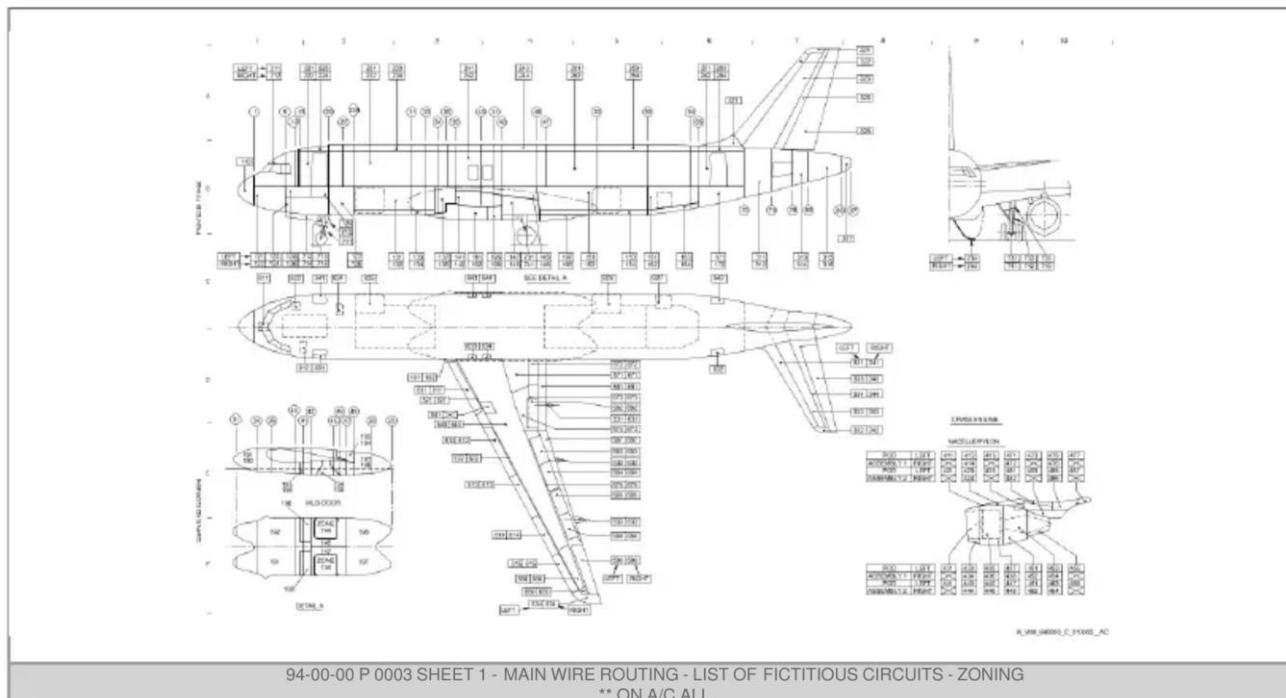
15.2%

People remember 10% of what they hear, 20% of what they read,
but 80% of what they see.**

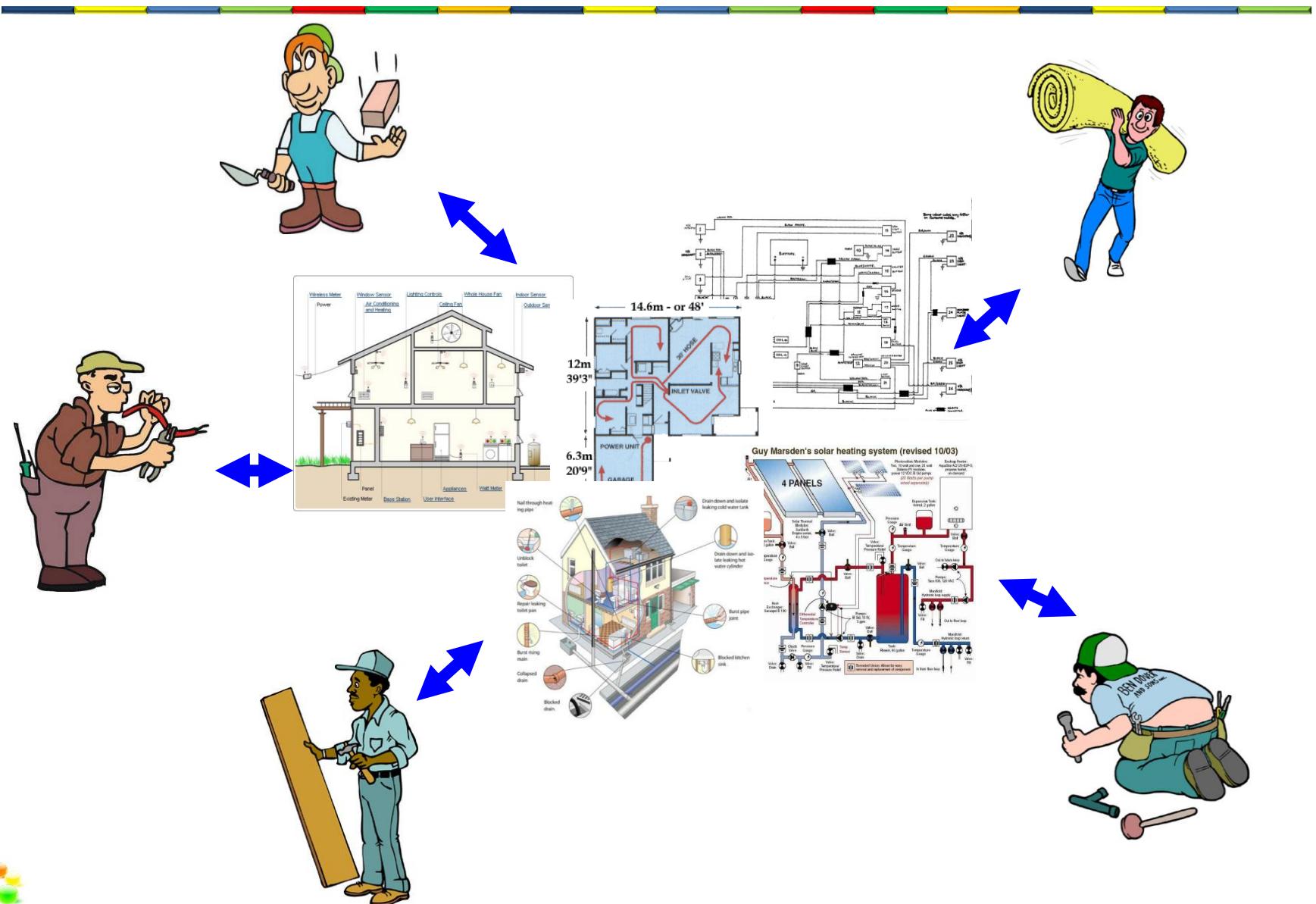


Diagrams

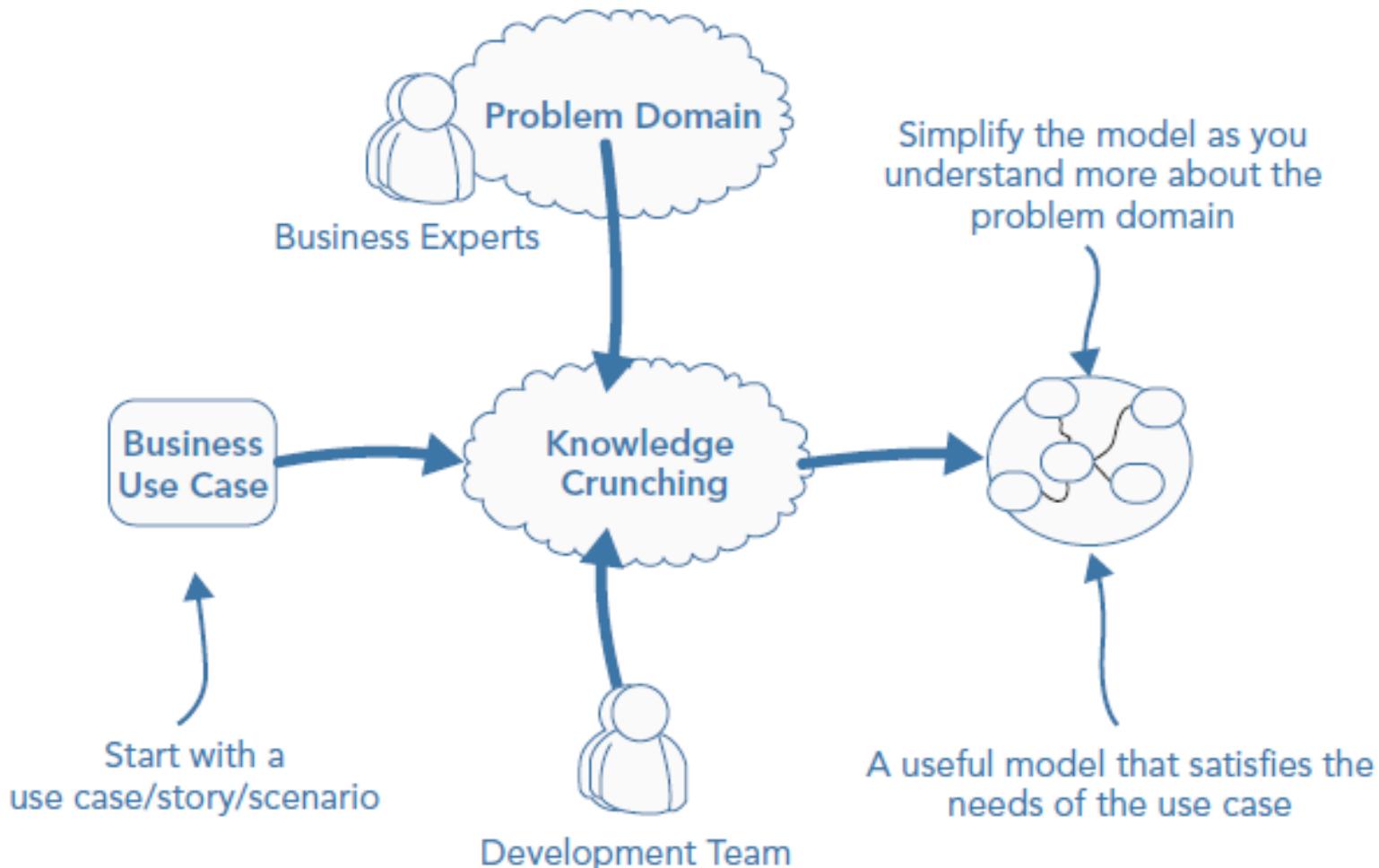
- For a specific Airbus aircraft like the A350, **6,000 physical wiring diagrams** are defined



Model and diagrams



Knowledge crunching



Advantages of a domain model

- Improves understanding
- Validates understanding
- Improves communication
- Shared glossary
- Improves discovery



Principles of Domain Driven Design

- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain

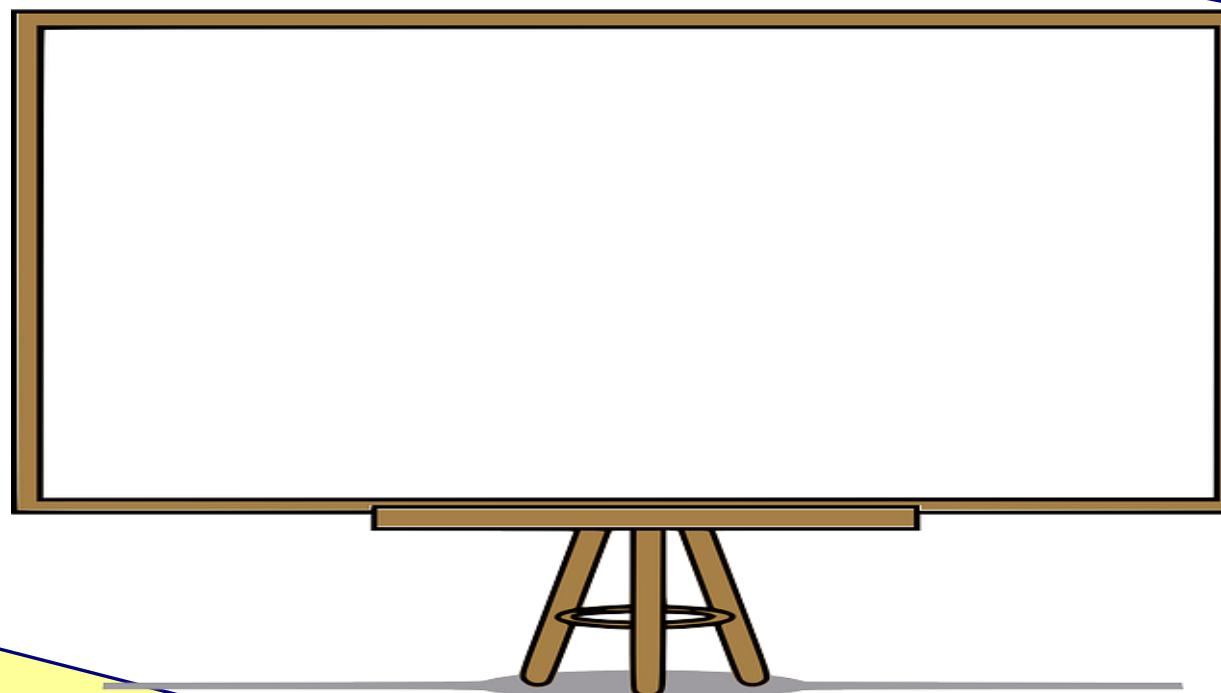


The software is a reflection of the real world

- It is easier to spot inconsistencies, errors, misconceptions.
- The software is easier to understand for
 - Existing developers
 - Testers
 - Business people (with guidance)
 - New developers and testers
- By looking at the code you can learn a lot of domain knowledge
- No translation necessary
- It is easier to write tests
- Easier to maintain the code



Example: Flight control system



We want to monitor air traffic. Where do we start?

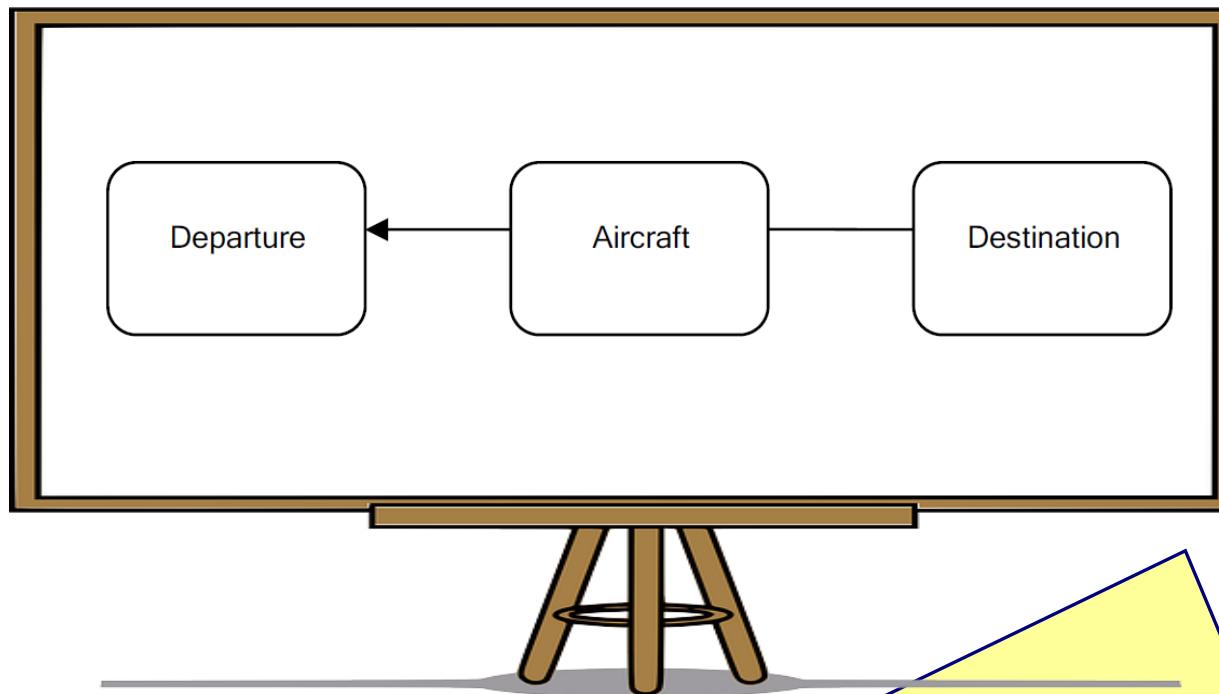
aircraft controller

developer

Let's start with the basics. All this traffic is made up of **planes**. Each plane takes off from a **departure** place, and lands at a **destination** place.

Flight control system

OK, so we get something like this.



aircraft
controller



developer

That's easy. When it flies, the plane can just choose any air path the pilots like? Is it up to them to decide which way they should go, as long as they reach destination?

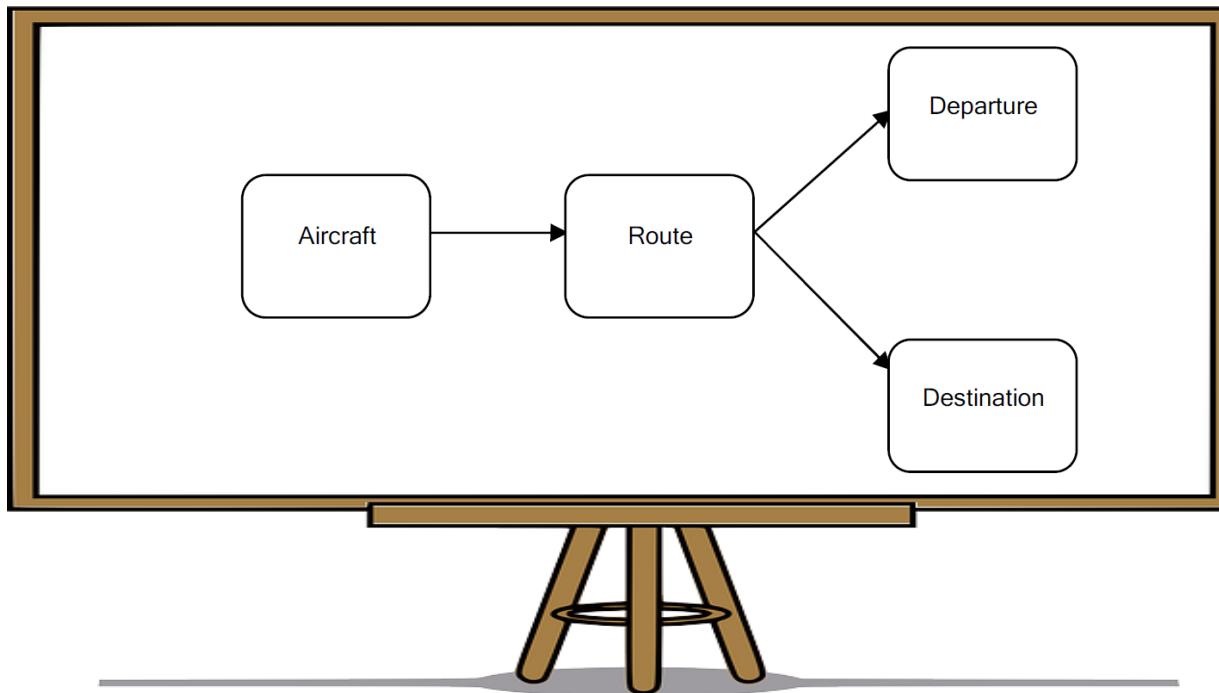


Flight control system

Oh, no. The pilots receive a **route** they must follow.
And they should stay on that route as close as possible.



aircraft
controller



developer

Can you explain routes to me?

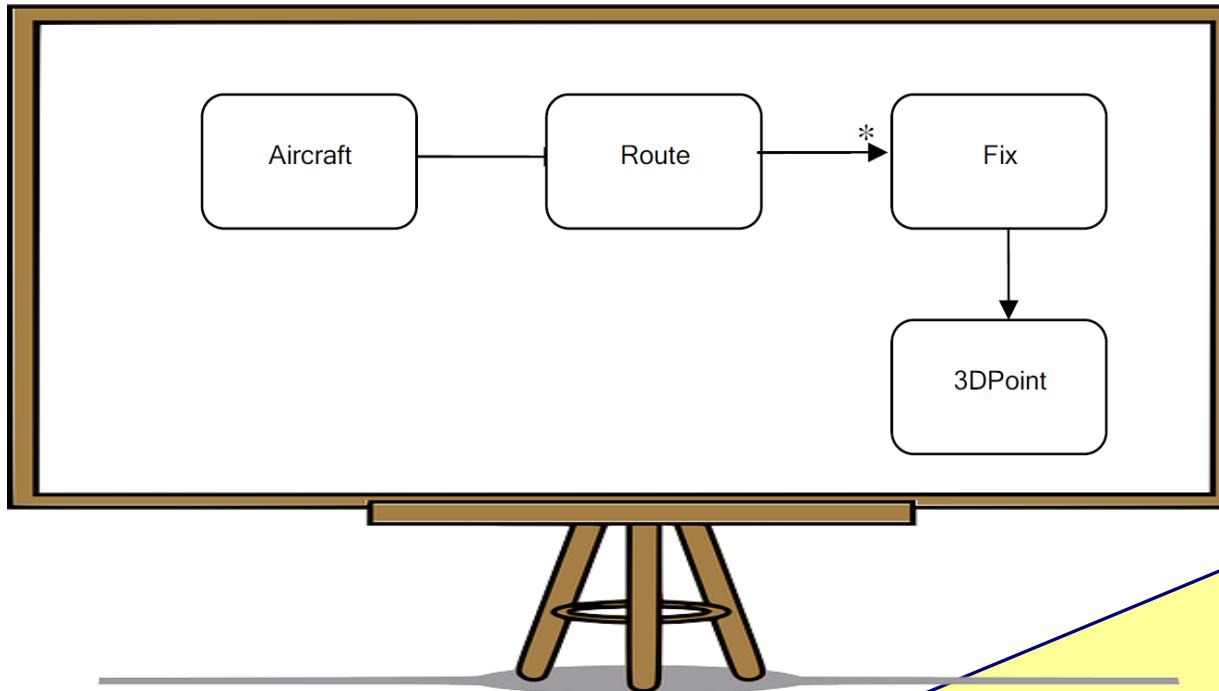


Flight control system

Well, a route is made up of small segments, and each segment has predetermined fixed points



aircraft
controller



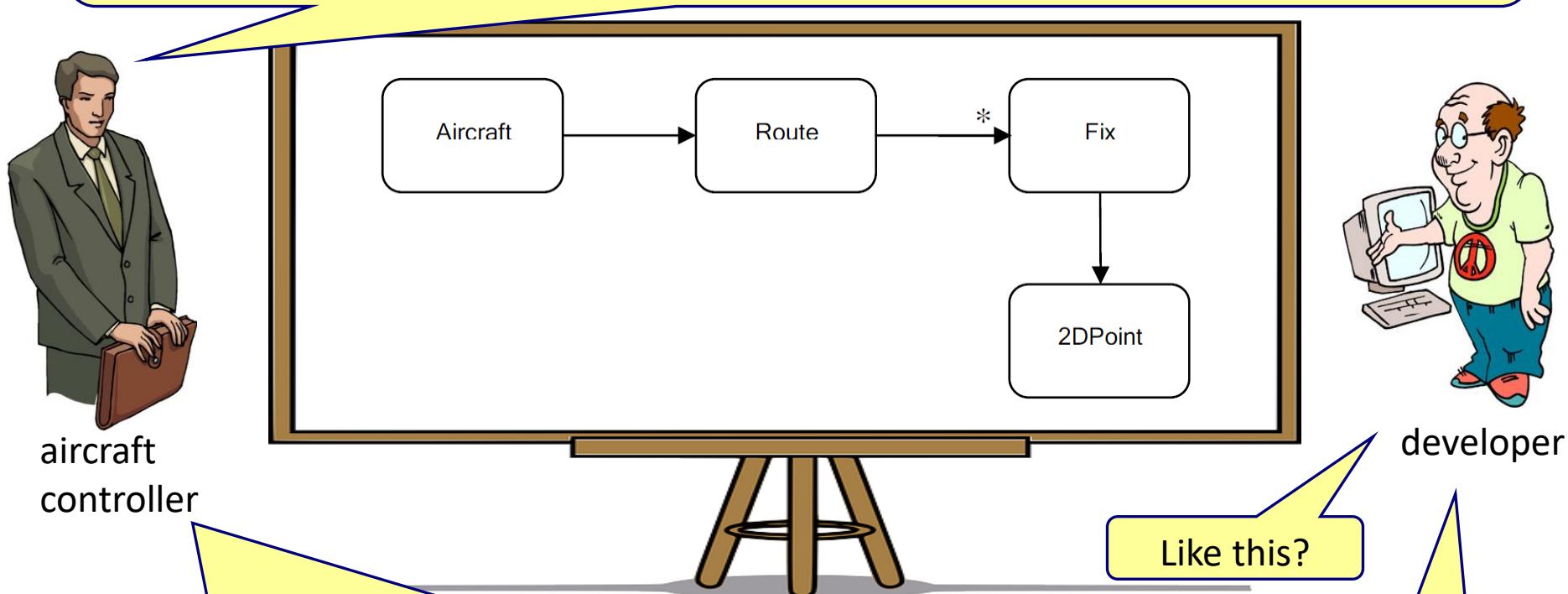
developer

OK, then let's call each of those points a **fix**, because it's a fixed point. And, by the way, the **departure** and **destination** are just **fixes**. I'm thinking of this **route** as a 3D path in the air. If we use a Cartesian system of coordinates, then the **route** is simply a series of 3D points.



Flight control system

I don't think so. We don't see **route** that way. The **route** is actually the projection on the ground of the expected air path of the airplane. The **route** goes through a series of points on the ground determined by their **latitude** and **longitude**.



Yes, the **altitude** that an airplane is to have at a certain moment is also established in the **flight plan**.

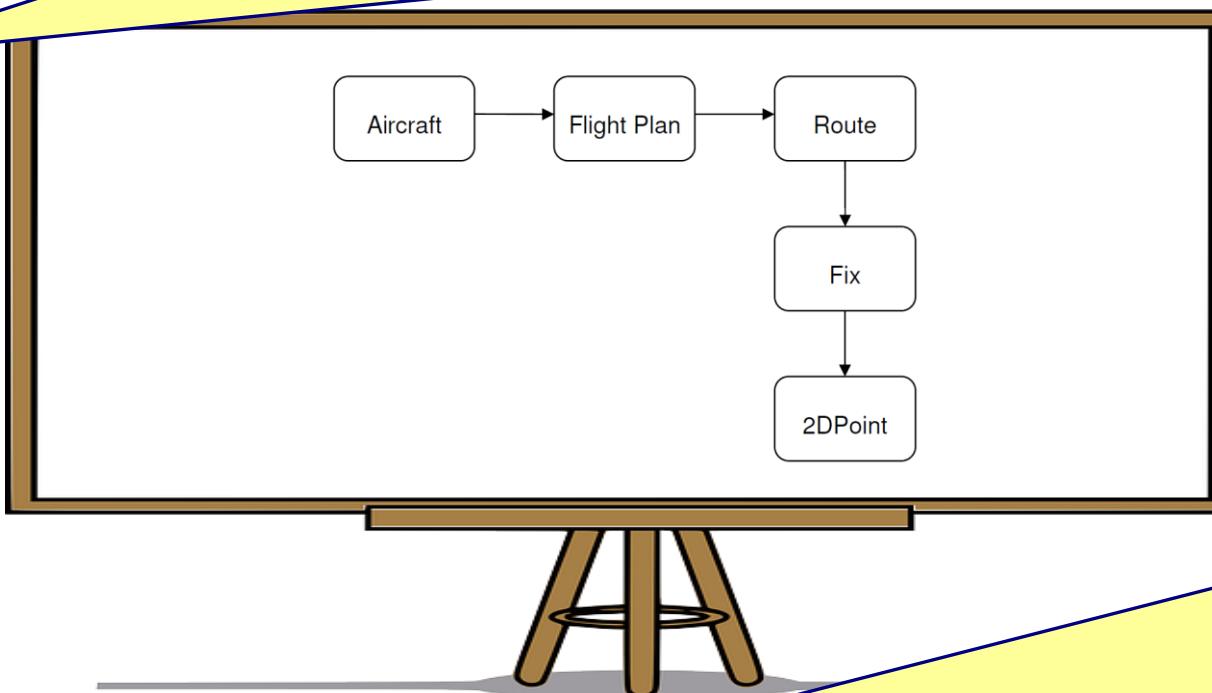
Flight plan? What is that?

Flight control system

Before leaving the airport, the pilots receive a detailed **flight plan** which includes all sorts of information about the **flight**: the **route**, cruise **altitude**, the cruise **speed**, the type of **airplane**, even information about the crew members.



aircraft
controller



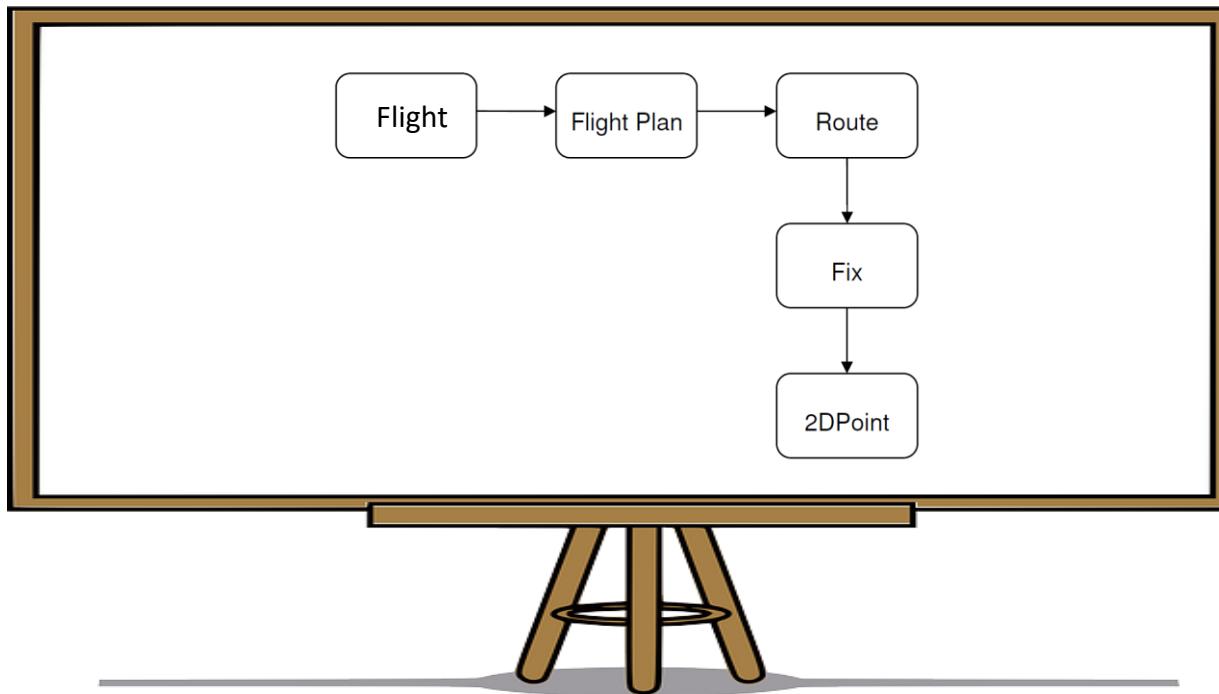
developer

Now that I'm looking at it, I realize something. When we are monitoring air traffic, we are not actually interested in the planes themselves, if they are white or blue, or if they are Boeing or Airbus. We are interested in their **flight**. That's what we are actually tracking and measuring. I think we should change the model a bit in order to be more accurate.

Flight control system



aircraft
controller



developer



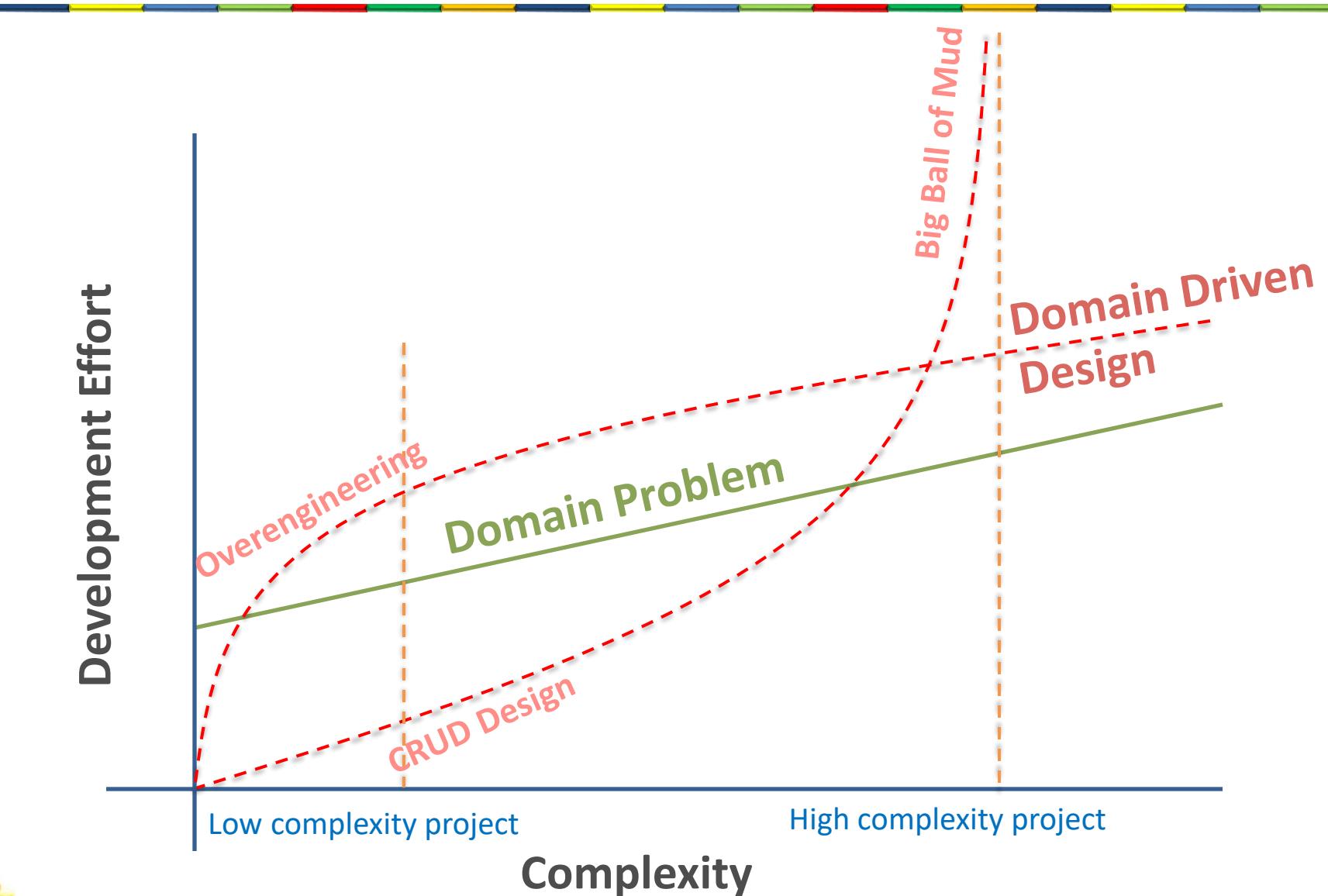
Cost of DDD

- A lot of time is spent on
 - Understanding the business
 - Brainstorming
 - Creating a ubiquitous language
 - Modeling the business
 - Validation
 - Bind the model and the implementation

*This works great in complex **business** domains*



When and why DDD?



When to use DDD?

- When the business domain is complex
 - Has initially nothing to do with technical complexity
- When the scope is medium to large
 - Team of > 4 developers
 - Project of > 4 months
- When the application needs to be maintained/evolved for a longer time
- Multiple teams working on the same application



ANEMIC AND RICH DOMAIN MODEL

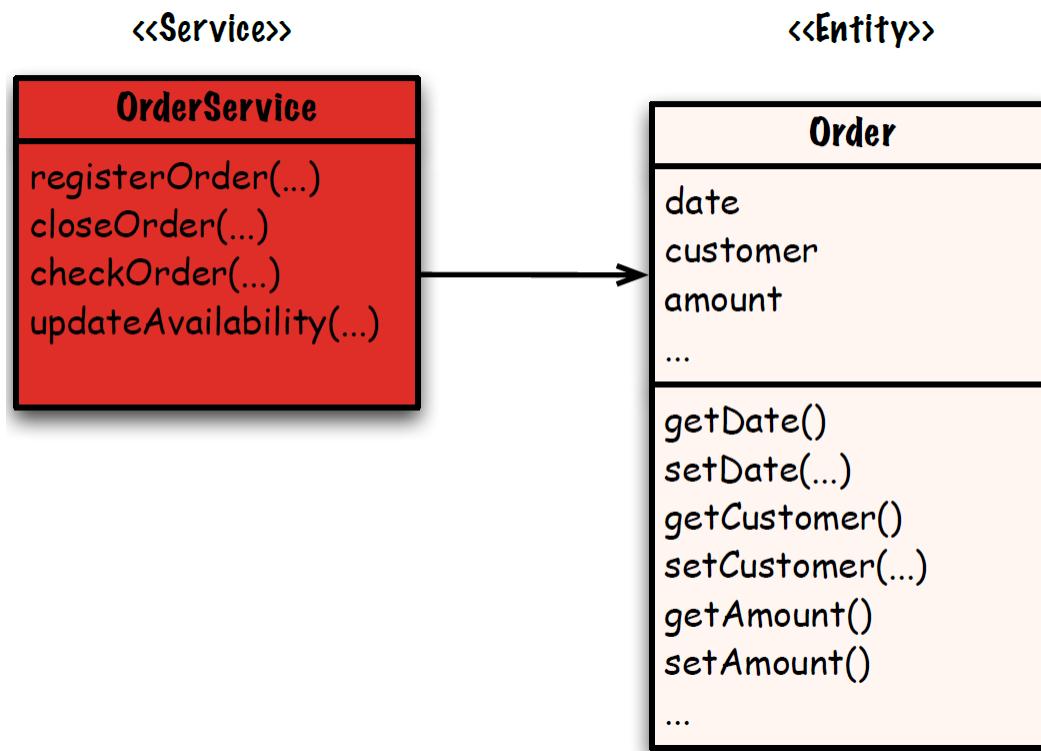


Anemic domain model

- Classes in the model have no business logic



NOT OK



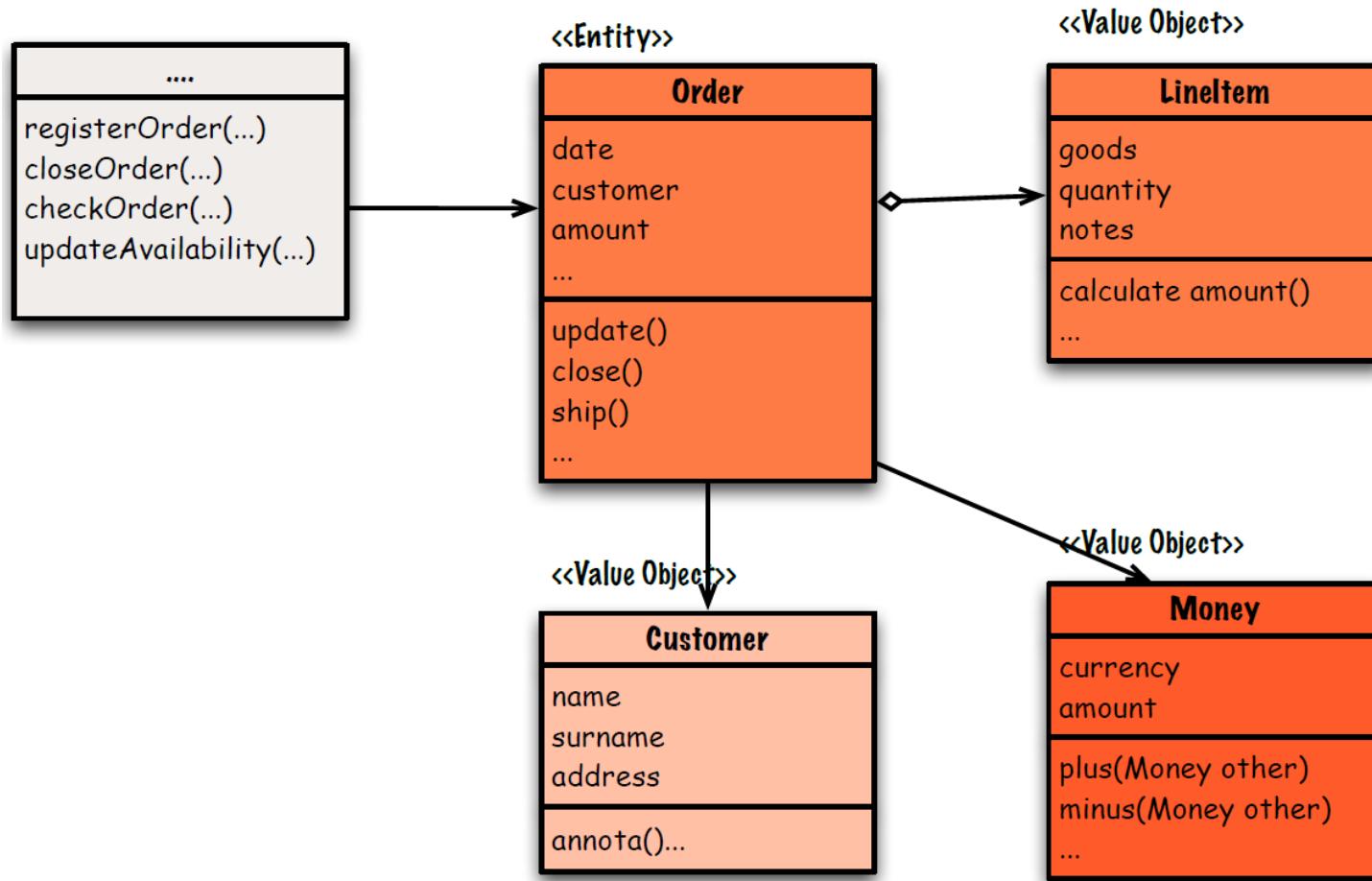
Disadvantages anemic domain model

- You do not use the powerful OO techniques to organize complex logic.
- Business logic (rules) is hard to find, understand, reuse, modify.
- The software reflects the data structure of the business, but not the behavioral organization
- The service classes become too complex
 - No single responsibility
 - No separation of concern

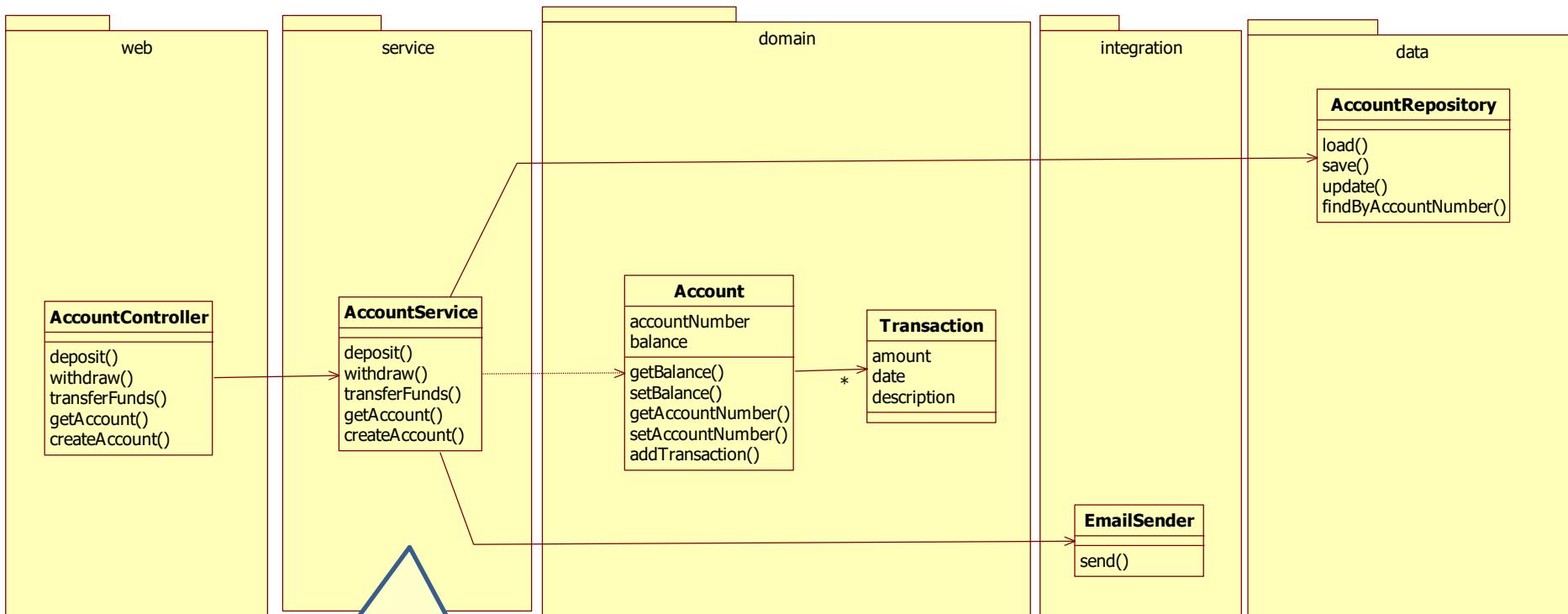


Rich domain model

- Classes with business logic

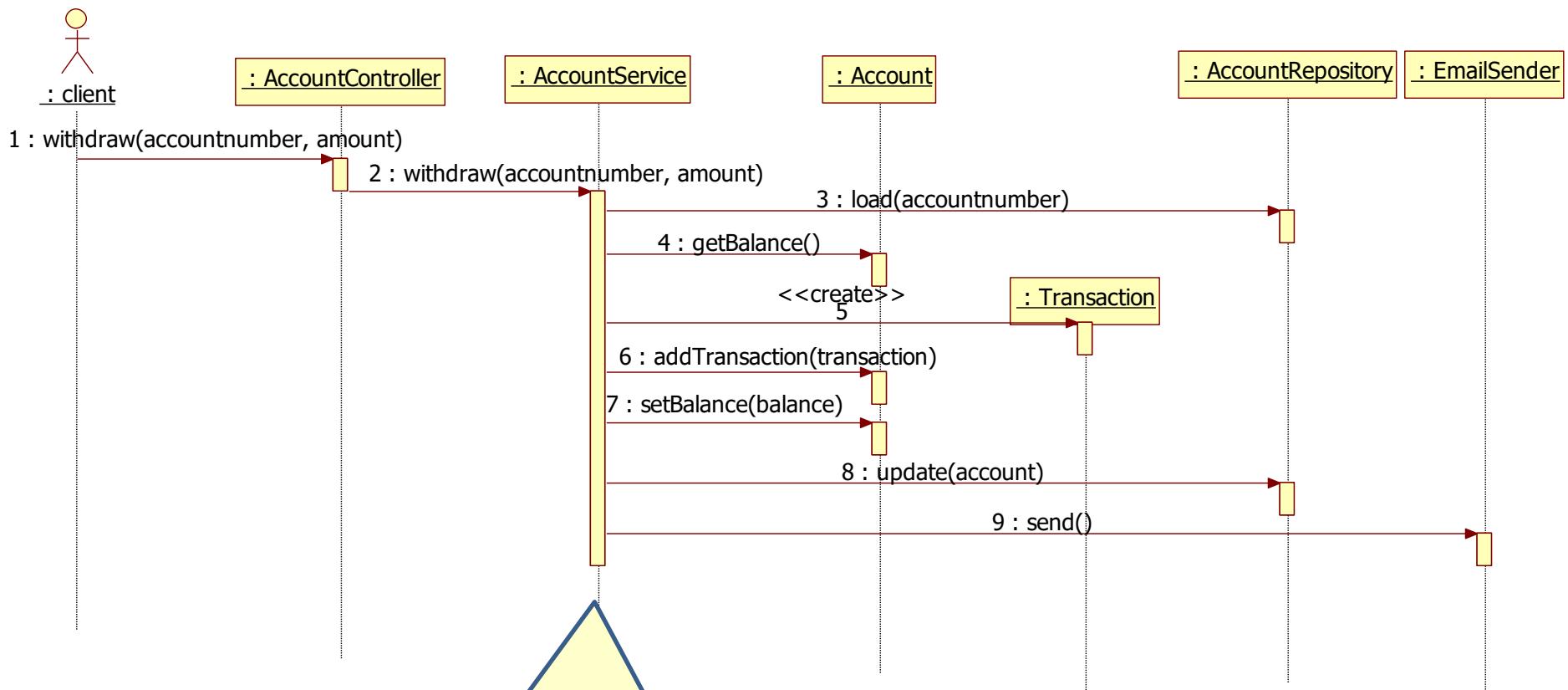


Anemic domain model example



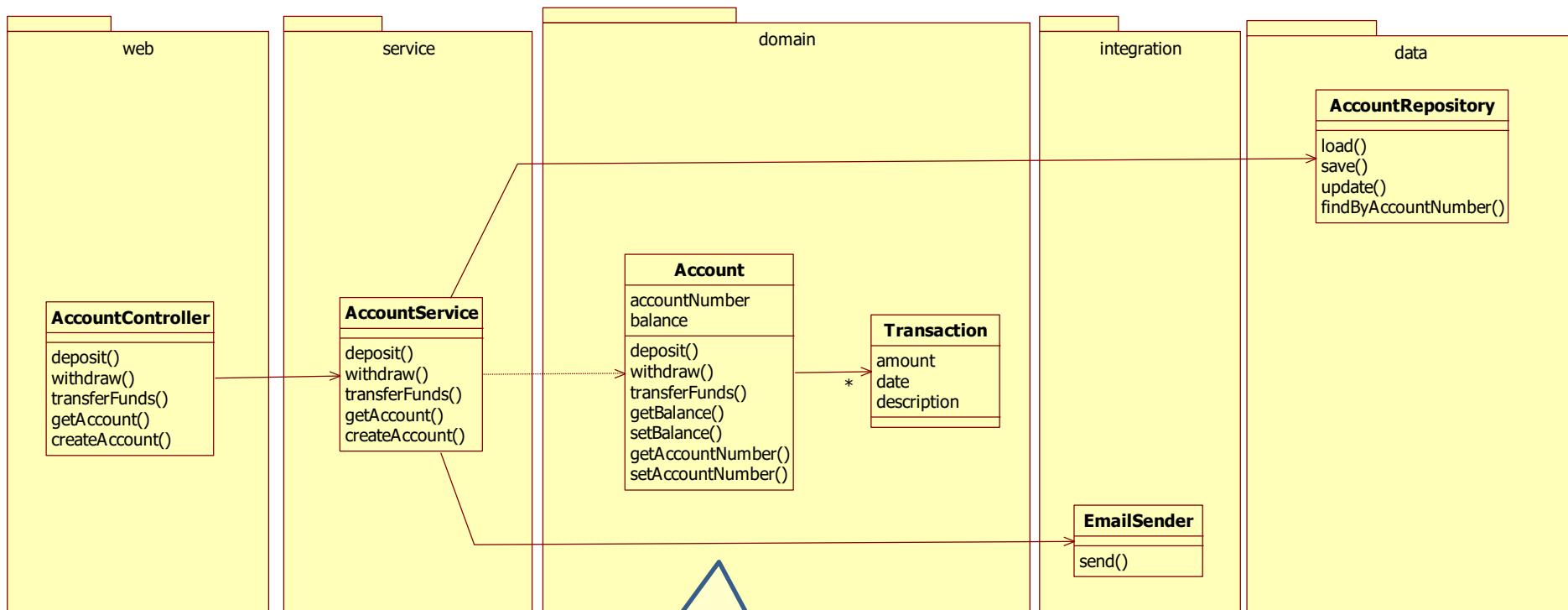
The business logic of withdraw() is done in the AccountService class

Anemic domain model example



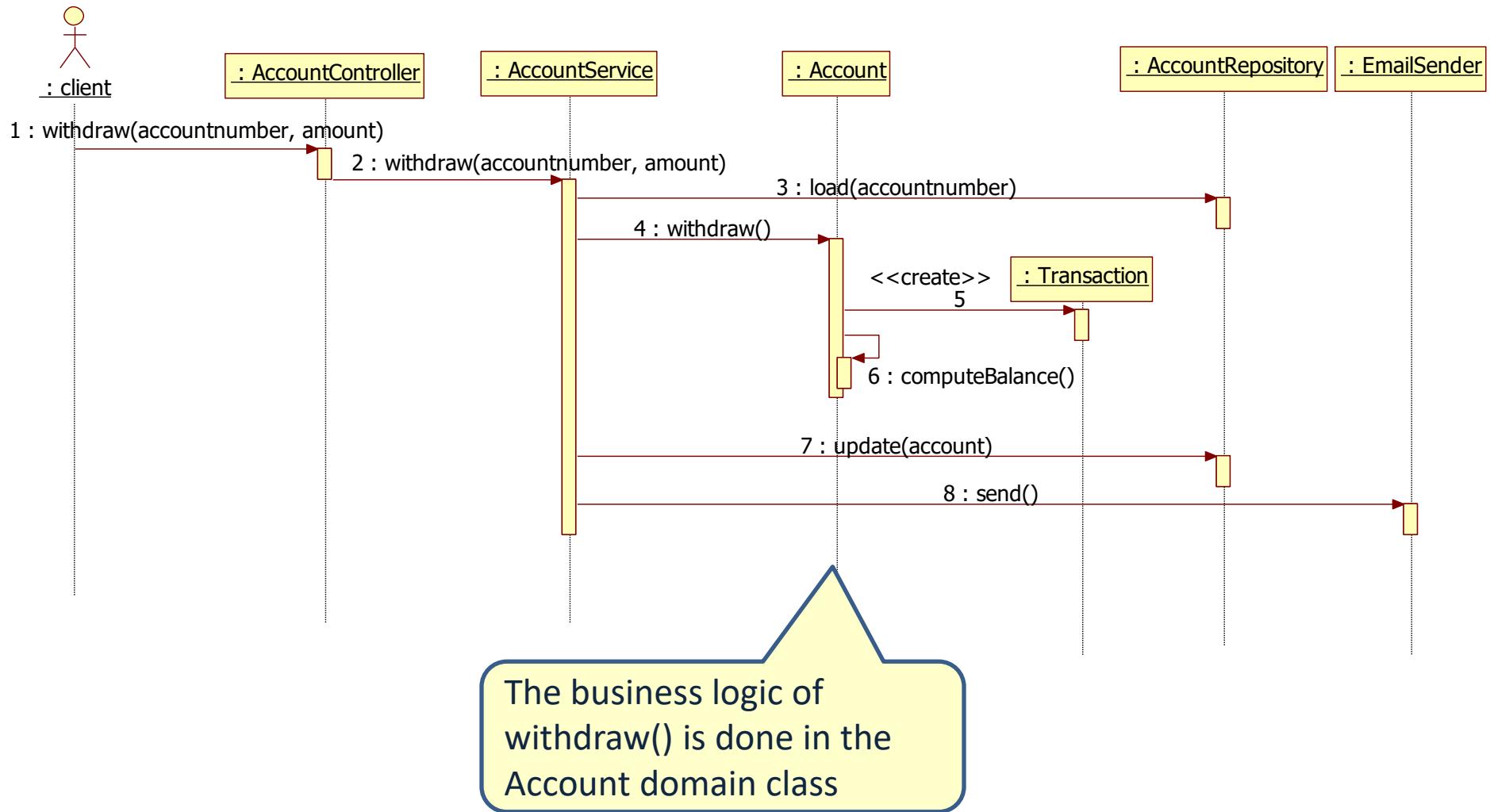
The business logic of
withdraw() is done in the
AccountService class

Rich domain model example



The business logic of
withdraw() is done in the
Account domain class

Rich domain model example



ORCHESTRATION & CHOREOGRAPHY



Orchestration vs. choreography

- **Orchestration**

- One central brain



Easy to follow
the process

Does not work
well in large and or
complex
applications

- **Choreography**

- No central brain



Hard to follow
the process

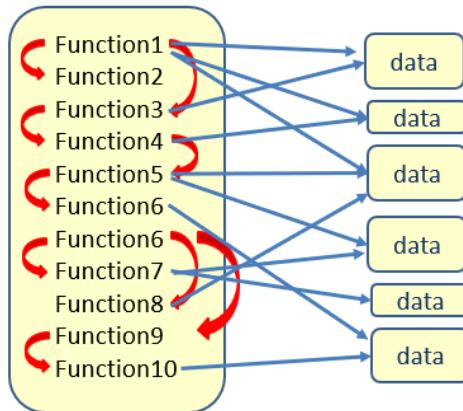
Does work well in
large and or
complex
applications

Orchestration vs. choreography

■ Orchestration

■ One central brain

Procedural programming
(C, Pascal, Algol, Cobol)



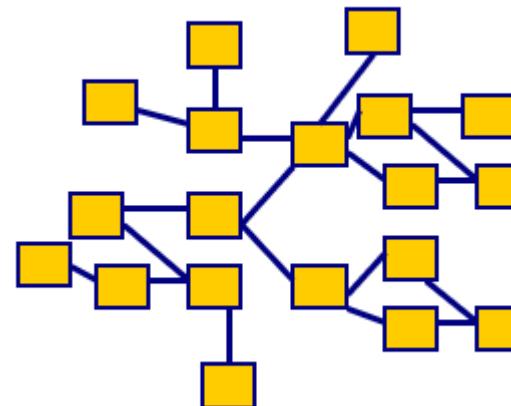
Easy to follow
the process

Does not work
well in large and or
complex
applications

■ Choreography

■ No central brain

Object-Oriented programming
(Java, C#, Python, C++)



Hard to follow
the process

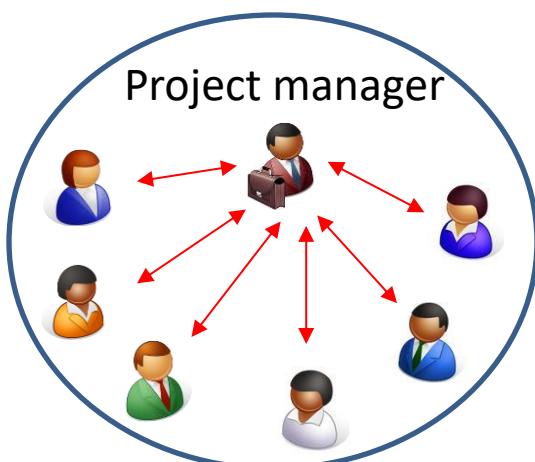
Does work well in
large and or
complex
applications

Orchestration vs. choreography

- Orchestration

- One central brain

Waterfall

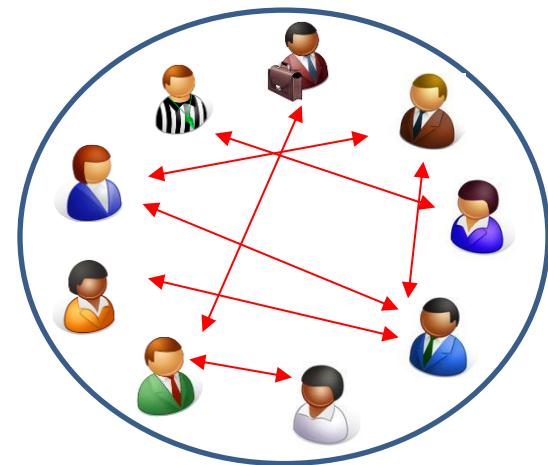


Does not work well
complex projects

- Choreography

- No central brain

Agile



Does work well in
complex projects

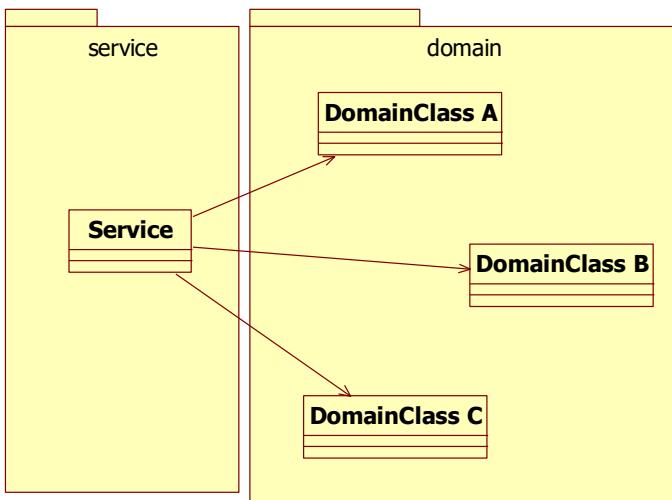


Orchestration vs. choreography

- Orchestration

- One central brain

Anemic domain model



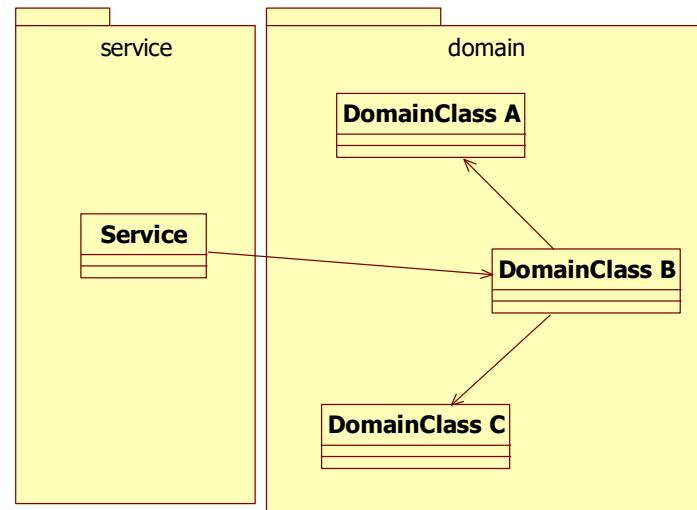
Easy to follow
the process

Does not work
well in large and or
complex
applications

- Choreography

- No central brain

Rich domain model



Hard to follow
the process

Does work well in
large and or
complex
applications



DOMAIN MODEL PATTERNS



Domain Model Patterns

- Entities
- Value objects
- Domain services
- Domain events



ENTITIES

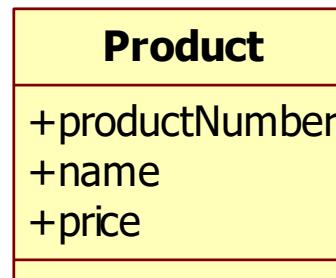
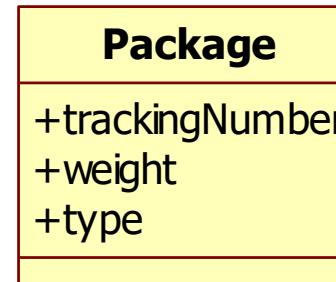
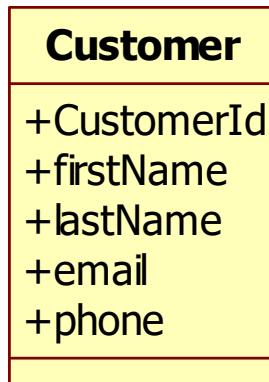


Entities

- A class with identity
- Mutable
 - State may change after instantiation
 - The entity has an lifecycle
 - The order is placed
 - The order is paid
 - The order is fulfilled



Example entity classes



Entities

- Changing attributes doesn't change which one we're talking about
 - Identity remains constant throughout its lifetime



VALUE OBJECTS



Value objects

- Has no identity
 - Identity is based on composition of its values
- Immutable
 - State cannot be changed after instantiation



Example value object classes

Address
-street
-city
-zip
+computeDistance(Address a)
+equals(Address a)

Money
-amount
-currency
+add(Money m)
+subtract(Money m)
+equals(Money m)

Review
-nrOfStars
-description

Weight
-value
-unit
+add(Weight w)
+subtract(Weight w)
+equals(Weight w)

Dimension
-length
-width
-height
+add(Dimension d)
+subtract(Dimension d)
+equals(Dimension d)



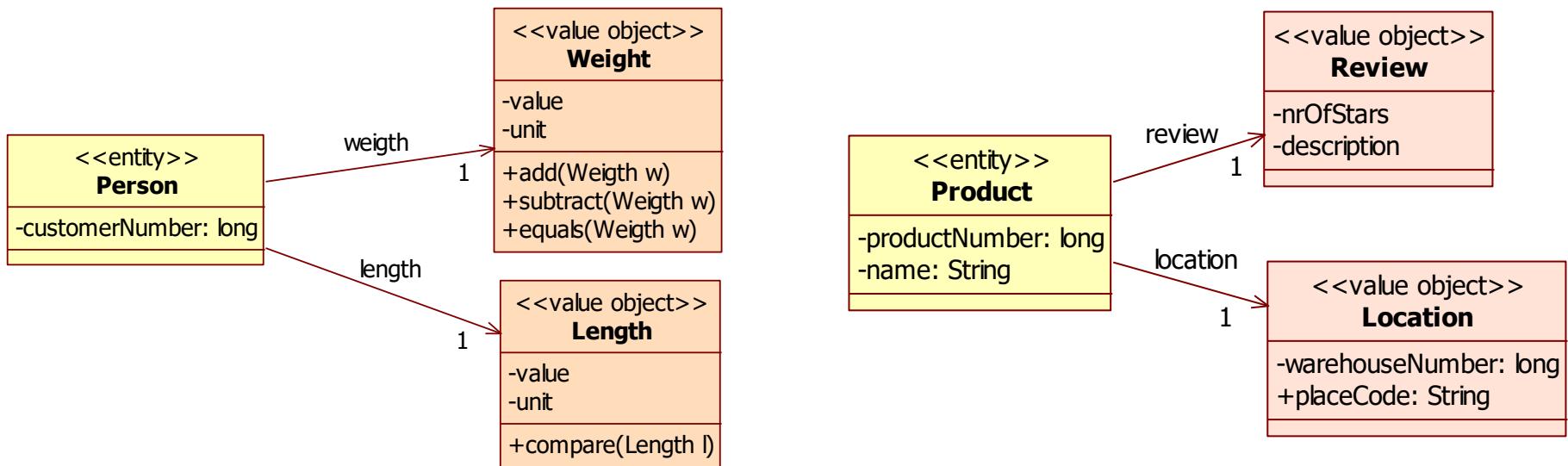
Value object characteristics

- No identity
- Attribute-based equality
- Behavior rich
- Cohesive
- Immutable
- Combinable
- Self-validating
- Testable



No identity

- Value objects tell something about another object



- Technically, value objects may have IDs using some database persistence strategies.
 - But they have no identity in the domain.

Attribute-based equality

- 2 value objects are equal if they have the same attribute values

<<value object>> Address
-street -city -zip
+computeDistance(Address a) +equals(Address a)

<<value object>> Money
-amount -currency
+add(Money m) +subtract(Money m) +equals(Money m)



Behavior rich

- Value objects should expose expressive domain-oriented behavior

<<value object>>
Meters
-value: long
+toYards(): long
+toKilometers(): long
+isLongerThan(Meters m): boolean
+isShorterThan(Meters m): boolean



Cohesive

- Encapsulate cohesive attributes

<<value object>>
Money
-amount
-currency
+add(Money m)
+subtract(Money m)
+equals(Money m)

<<value object>>
Color
-red: int
-green: int
-blue: int
+equals(Color c)



Immutable

- Once created, a value object can never be changed

```
public class Money {  
    private BigDecimal value;  
  
    public Money(BigDecimal value) {  
        this.value = value;  
    }  
  
    public Money add(Money money){  
        return new Money(value.add(money.getValue()));  
    }  
  
    public Money subtract(Money money){  
        return new Money(value.subtract(money.getValue()));  
    }  
  
    public BigDecimal getValue() {  
        return value;  
    }  
}
```

No setter methods

Mutation leads to the creation of new instances



Minimize Mutability

- Reasons to make a class immutable:
 - Less prone to errors
 - Easier to share
 - Thread safe
 - Combinable
 - Self-validating
 - Testable



Combinable

- Can often be combined to create new values

```
public class Money {  
    private BigDecimal value;  
  
    public Money(BigDecimal value) {  
        this.value = value;  
    }  
  
    public Money add(Money money){  
        return new Money(value.add(money.getValue()));  
    }  
  
    public Money subtract(Money money){  
        return new Money(value.subtract(money.getValue()));  
    }  
  
    public BigDecimal getValue() {  
        return value;  
    }  
}
```

Combine 2 Money instances



Self-validating

- Value objects should never be in an invalid state

```
public class Money {  
    private BigDecimal value;  
  
    public Money(BigDecimal value) {  
        validate(value);  
        this.value = value;  
    }  
  
    private void validate(BigDecimal value){  
        if (value.doubleValue() < 0)  
            throw new MoneyCannotBeANegativeValueException();  
    }  
  
    public Money add(Money money){  
        return new Money(value.add(money.getValue()));  
    }  
  
    public BigDecimal getValue() {  
        return value;  
    }  
}
```

Self-validation



Testable

- Value objects are easy to test because of these qualities
 - Immutable
 - We don't need mocks to verify side effects
 - Cohesion
 - We can test the concept in isolation
 - Combinability
 - Allows to express the relationship between 2 value objects



Static factory methods

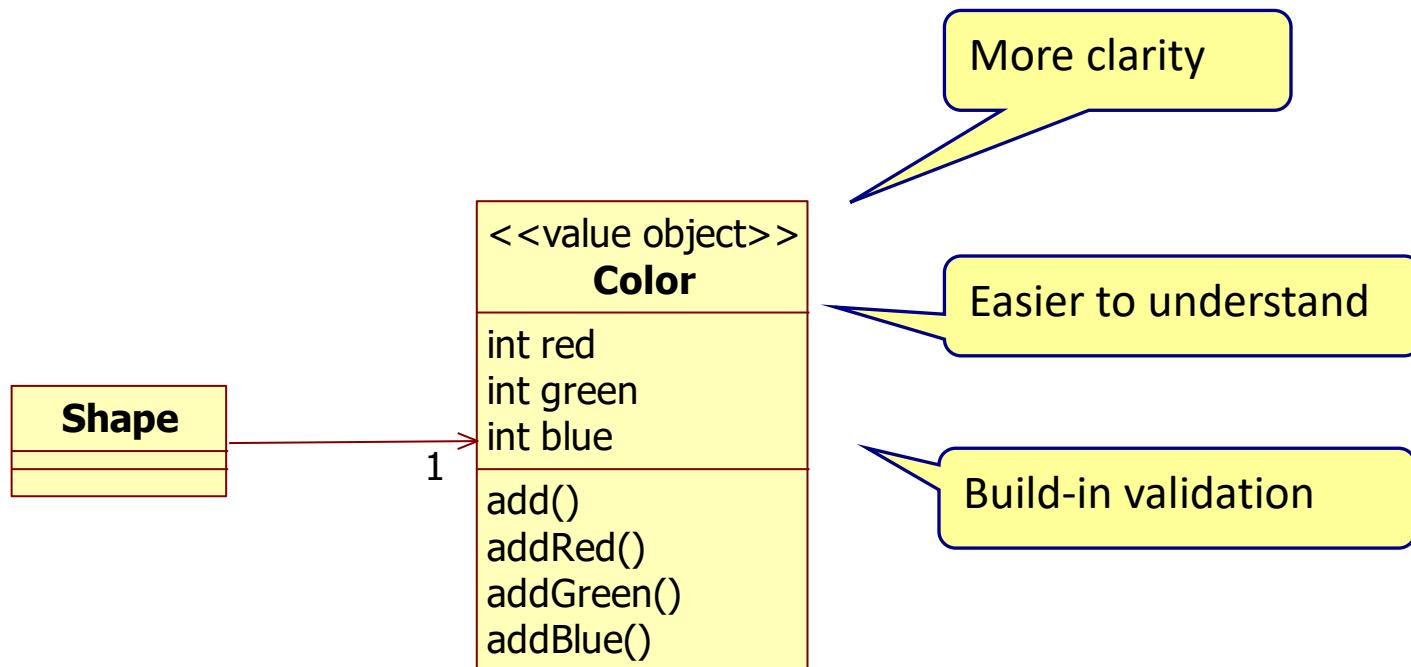
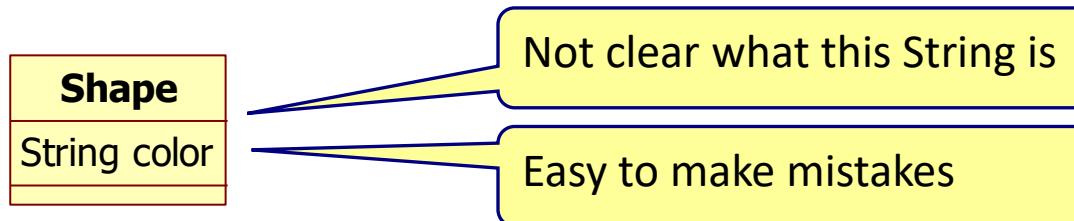
```
public class Height {  
    private enum MeasureUnit {  
        METER,  
        FEET,  
        YARD;  
    }  
  
    private int value;  
    private MeasureUnit unit;  
  
    public Height(int value, MeasureUnit unit) {  
        this.value = value;  
        this.unit = unit;  
    }  
  
    public static Height fromFeet(int value) {  
        return new Height(value, MeasureUnit.FEET);  
    }  
  
    public static Height fromMeters(int value) {  
        return new Height(value, MeasureUnit.METER);  
    }  
}
```

More expressive

Easier for clients to call

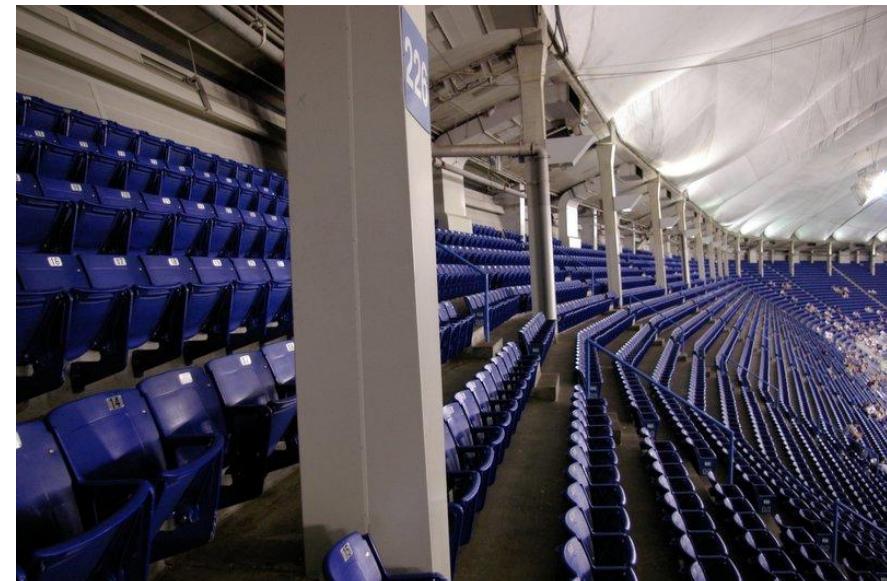
Decouple clients
from MeasureUnit

Enhancing explicitness

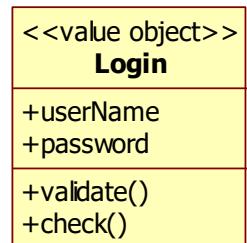
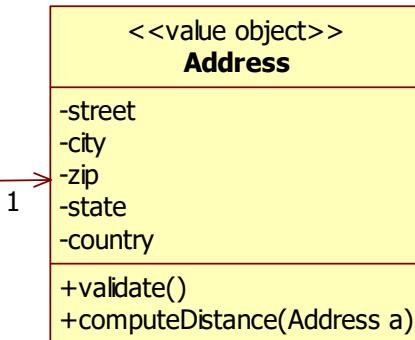
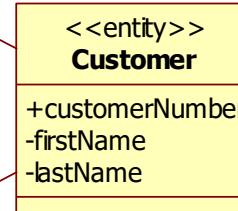
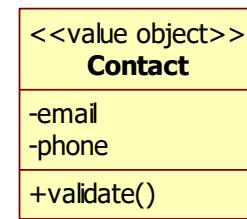


Entity versus value objects

- If visitors can sit wherever they find an empty seat then seat is a...
- If visitors buy a ticket with a seat number on it, then seat is a...



Pushing behavior into value objects



Entities versus Value objects

- Entities have their own intrinsic identity, value objects don't.
- The notion of identity equality refers to entities
 - Two entities are the same if their id's are the same
- The notion of structural equality refers to value objects
 - Two value objects are the same if their data is the same
- Entities have a history; value objects have a zero lifespan.
- A value object should always belong to one or several entities.
 - It can't live by its own.
- Value objects should be immutable; entities are almost always mutable.
 - If you change the data in a value object, create a new object.
- Always prefer value objects over entities in your domain model.



Main point

- Instead of a large entity class, we strive for a small and simple entity class with many value objects
- The Unified Field contains all knowledge in its simplest and most abstract form.



DOMAIN SERVICES



Domain service

- Sometimes behavior does not belong to an entity or value object
 - But it is still an important domain concept
- Use a domain service.

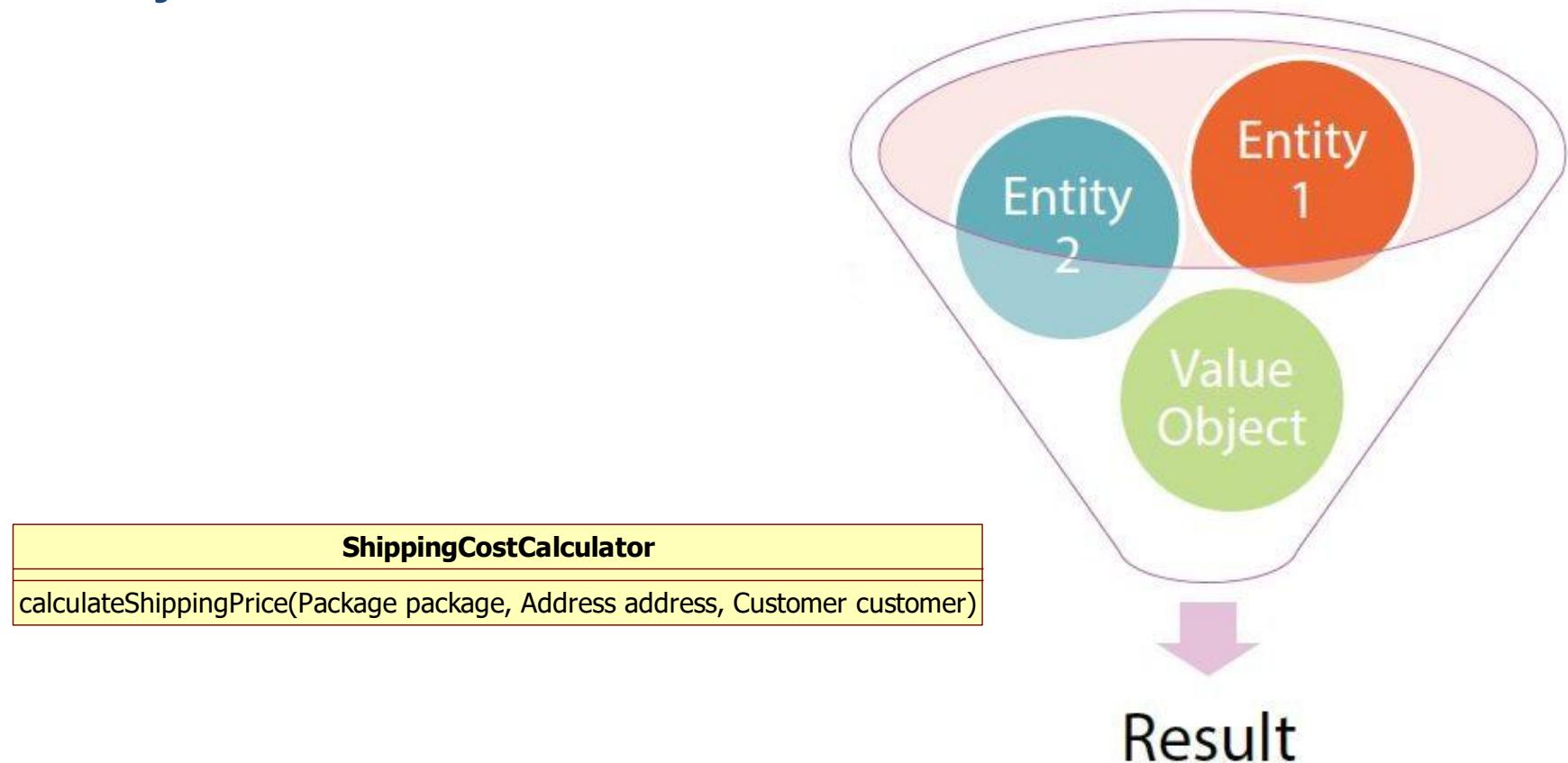
ShippingCostCalculator

calculateShippingPrice(Package package, Address address, Customer customer)



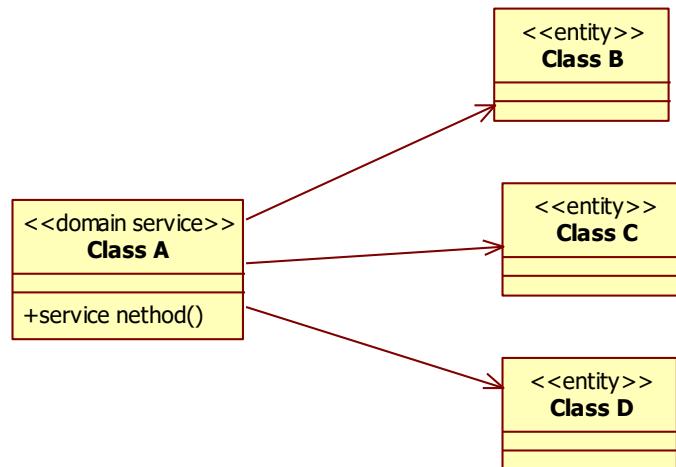
Domain service

- Interface is defined in terms of other domain objects

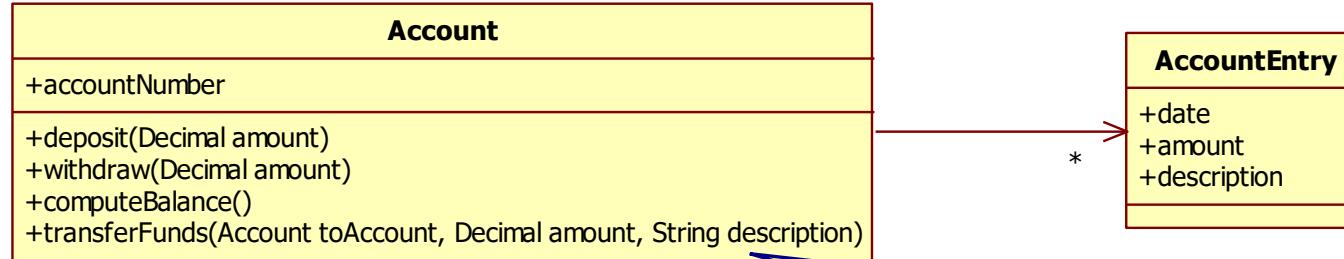


Domain service characteristics

- Stateless
 - Have no attributes
- Represent behavior
 - No identity
- Often orchestrate multiple domain objects



Domain Service example



The Account is responsible for transferring money

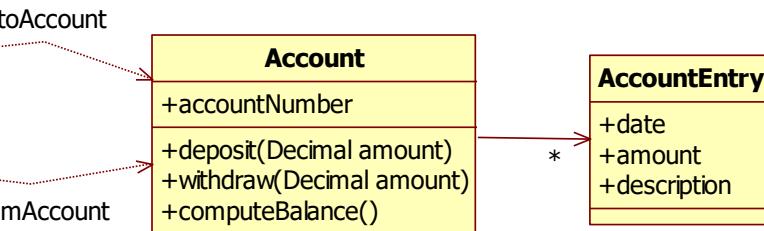
Service for transferring money

TransferFundsService

+transferFunds(Account toAccount, Account fromAccount, Decimal amount, String description)

toAccount

fromAccount

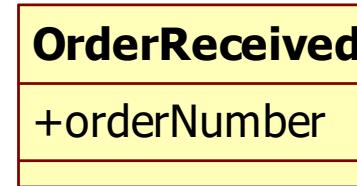
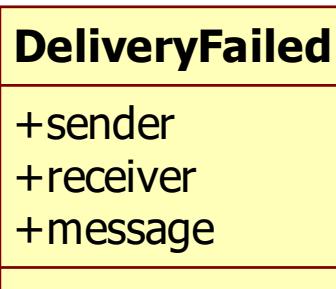


DOMAIN EVENTS



Domain event

- Classes that represent important events in the problem domain that have already happened
 - Immutable

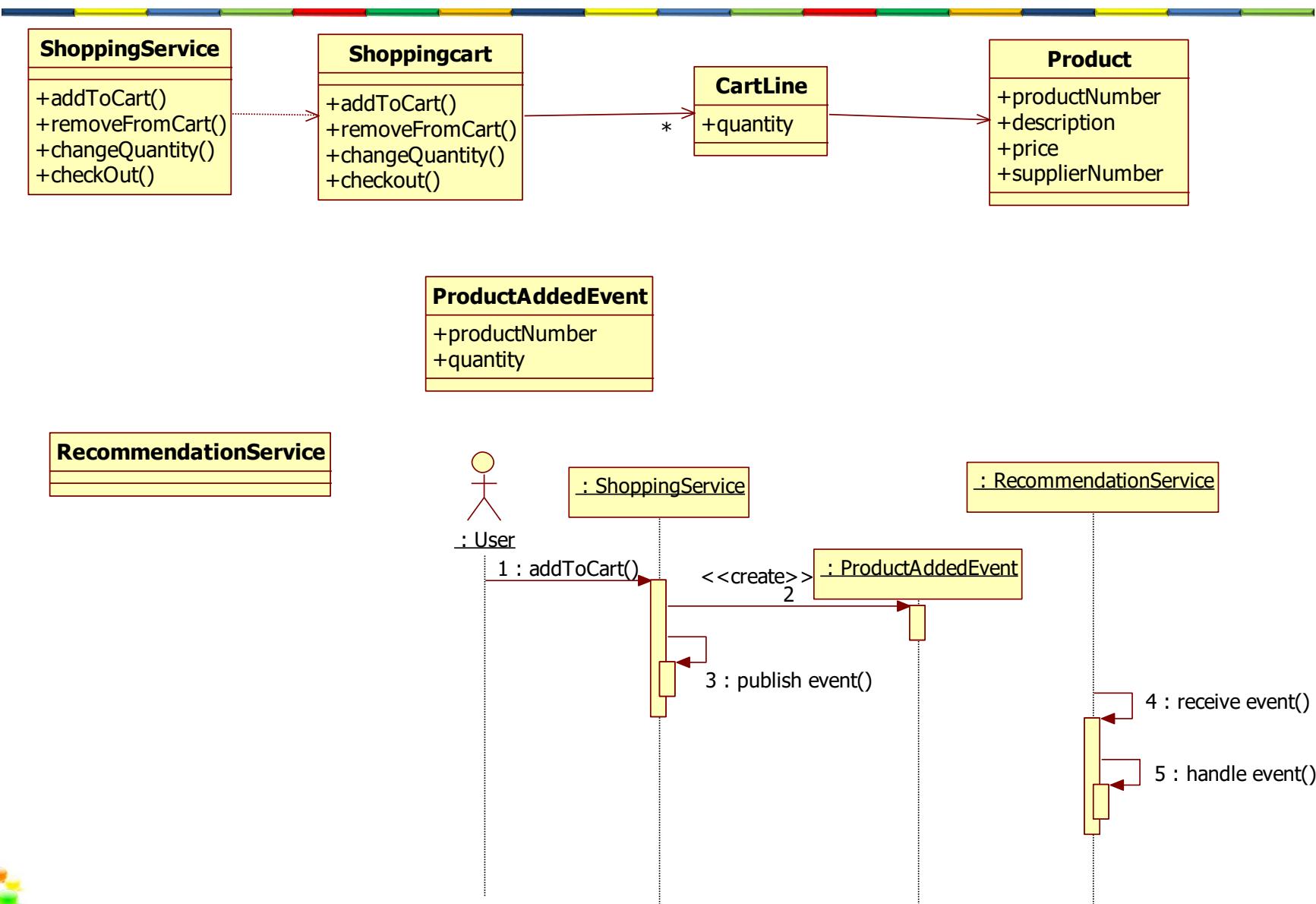


Domain event

- Events are raised and event handlers handle them.
- Some handlers live in the domain, and some live in the service layer.



Domain event example



SUMMARY



Domain Model Patterns

- Entities
- Value objects
- Domain services
- Domain events



Key principle 4

- The hardest and most important aspect of software development is the domain
 - Create a domain model
 - Knowledge crunching between business and IT
 - Place the domain logic in a separate layer
 - Let the domain logic be a reflection of the real world



Key principle 5

- Orchestration works well if the application is simple and/or the scope is small
- Choreography works well if the application is complex and/or the scope is large
- Orchestration
 - One central brain
- Choreography
 - No central brain



Connecting the parts of knowledge with the wholeness of knowledge

1. A rich domain model contains all domain knowledge.
 2. An entity class has identity while a value object has no identity.
-
3. **Transcendental consciousness** is the source of all activity.
 4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that all activity in the universe are expressions from and within one's own silent pure consciousness.
- 



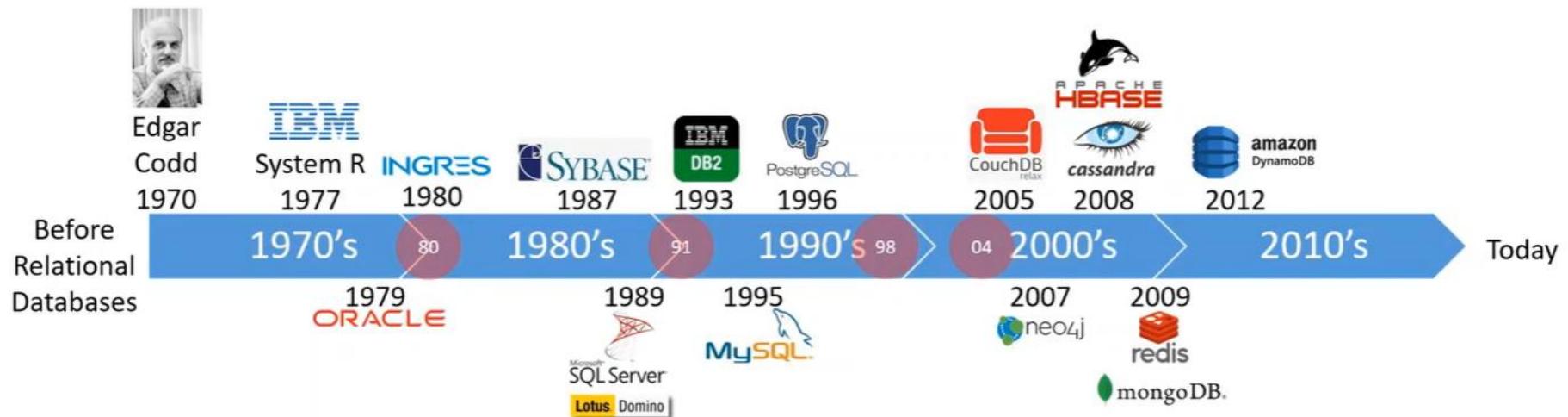
Lesson 3

DATABASES



Today's requirements on databases

- Big data (large datasets)
- Agility
- Unstructured/ semi structured data



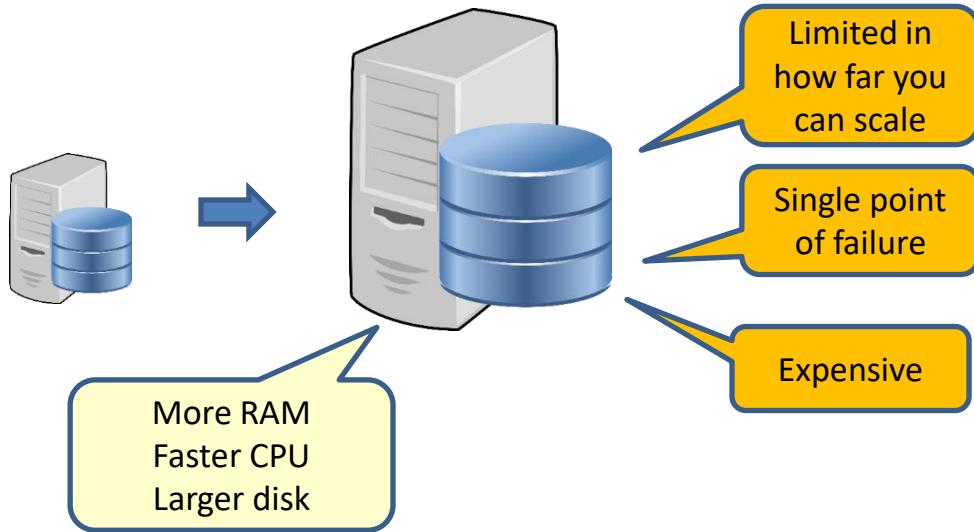
Large datasets

- Too much data
 - The data does not fit anymore on one node

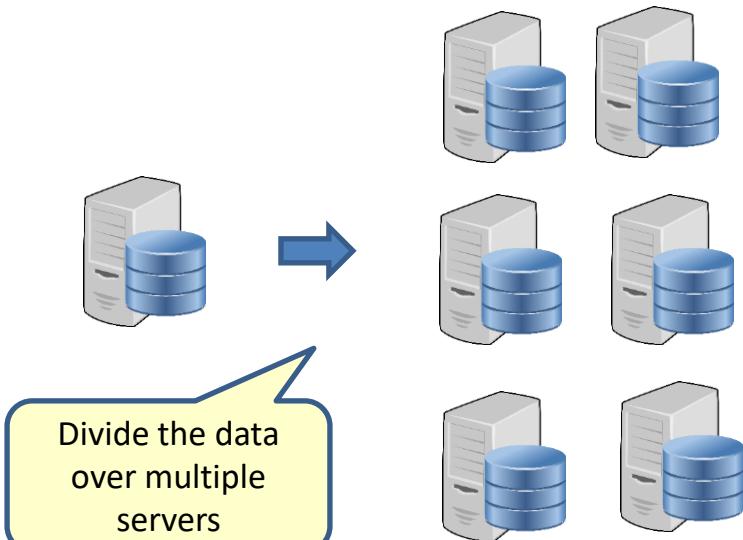


Database Scaling

- Vertical scaling



- Horizontal scaling



Five different types of databases

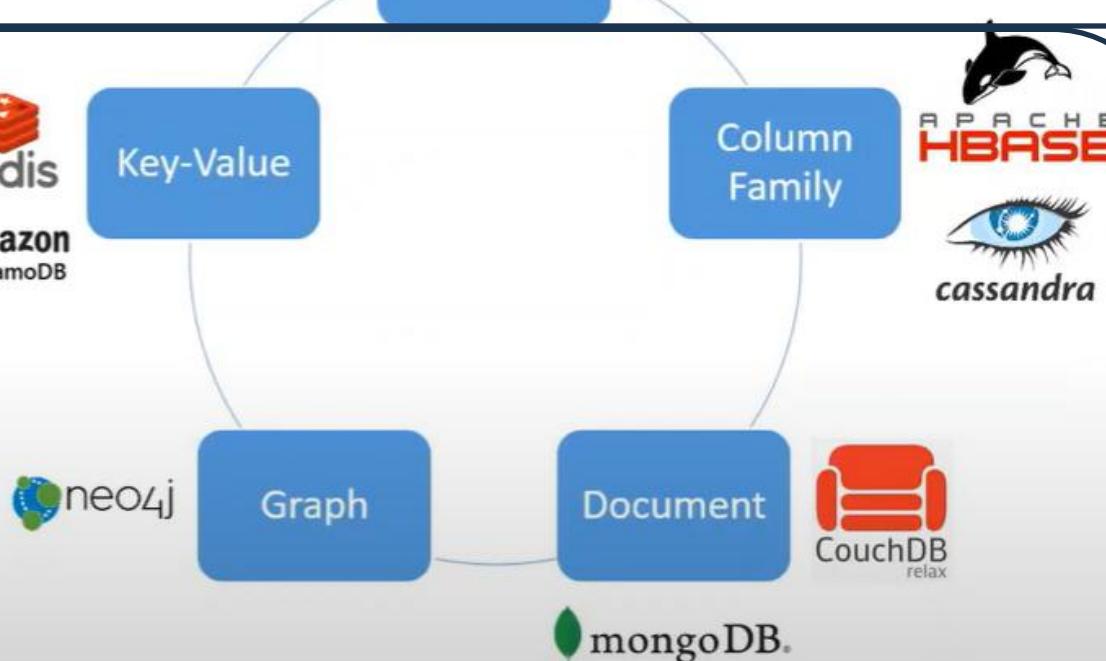
Relational



Non-relational

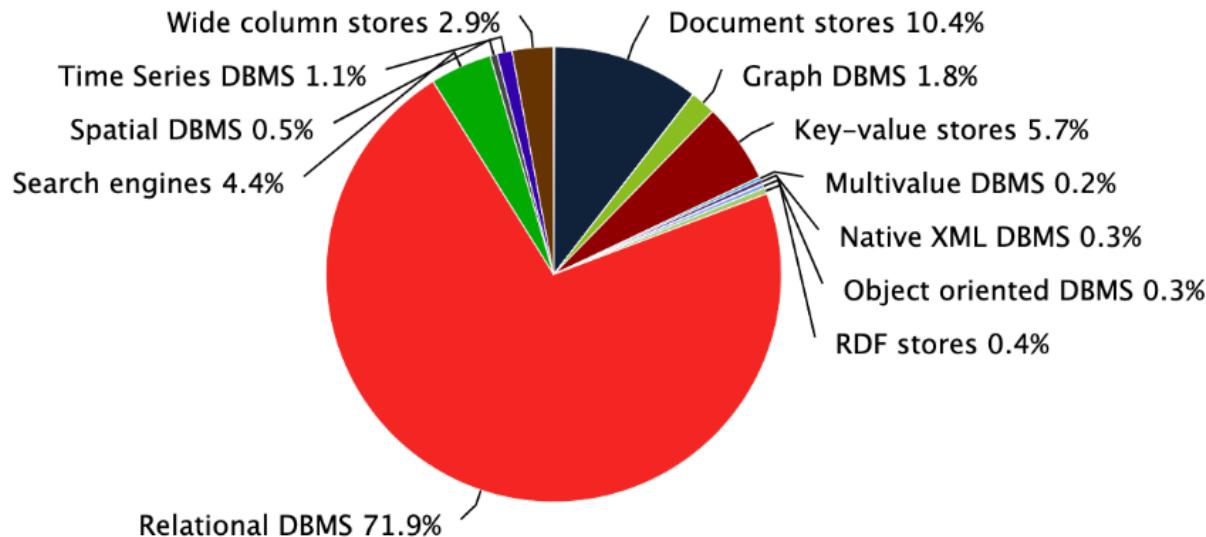


NoSQL
(Not only SQL)



Relational vs non-relational popularity

Ranking scores per category in percent, January 2023



© 2023, DB-Engines.com



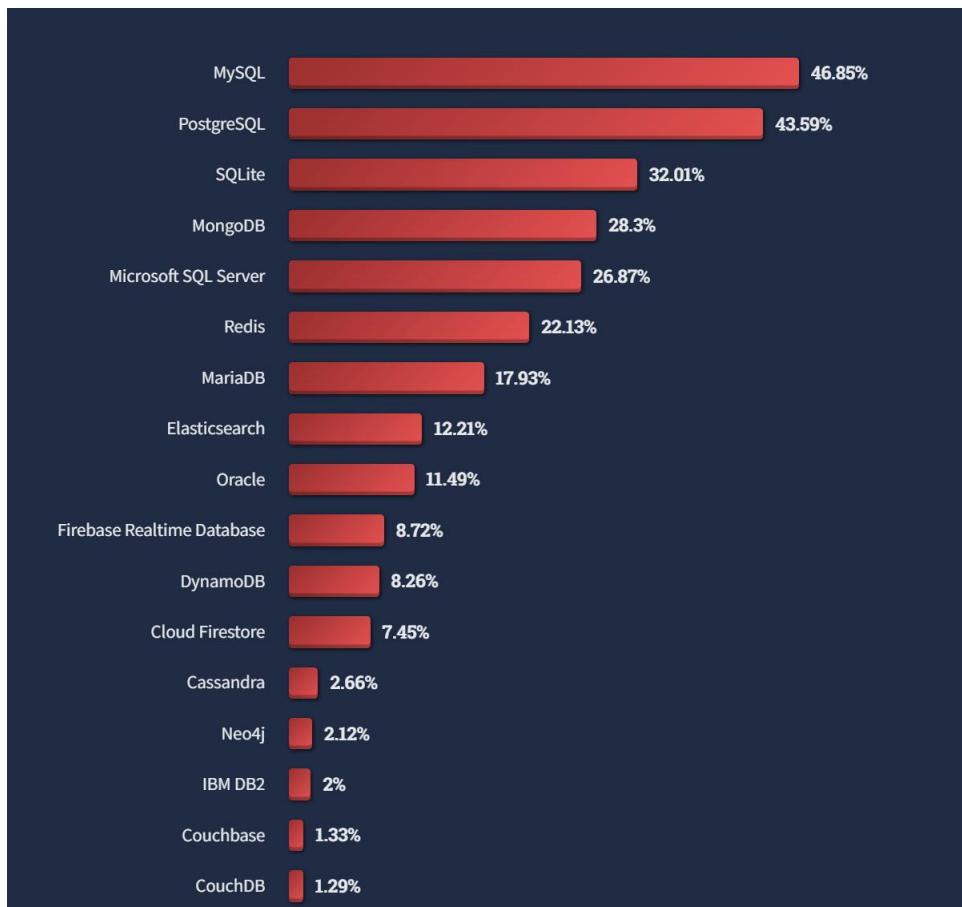
Most popular databases

<https://db-engines.com/en/ranking>

Rank May 2024	Rank Apr. 2024	Rank May 2023	DBMS	Database Model	Score		
					May 2024	Apr. 2024	May 2023
1.	1.	1.	Oracle	Relational, Multi-model	1236.29	+2.02	+3.66
2.	2.	2.	MySQL	Relational, Multi-model	1083.74	-3.99	-88.72
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	824.29	-5.50	-95.80
4.	4.	4.	PostgreSQL	Relational, Multi-model	645.54	+0.49	+27.64
5.	5.	5.	MongoDB	Document, Multi-model	421.65	-2.31	-14.96
6.	6.	6.	Redis	Key-value, Multi-model	157.80	+1.36	-10.33
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	135.35	+0.57	-6.28
8.	8.	↓ 7.	IBM Db2	Relational, Multi-model	128.46	+0.97	-14.56
9.	9.	↑ 11.	Snowflake	Relational	121.33	-1.87	+9.61
10.	10.	↓ 9.	SQLite	Relational	114.32	-1.69	-19.54
11.	11.	↓ 10.	Microsoft Access	Relational	104.92	-0.49	-26.26
12.	12.	12.	Cassandra	Wide column, Multi-model	101.89	-1.97	-9.25
13.	13.	13.	MariaDB	Relational, Multi-model	93.21	-0.60	-3.66
14.	14.	14.	Splunk	Search engine	86.45	-2.26	-0.19
15.	↑ 17.	↑ 18.	Databricks	Multi-model	78.61	+2.28	+14.66
16.	↓ 15.	16.	Microsoft Azure SQL Database	Relational, Multi-model	77.99	-0.41	-1.21
17.	↓ 16.	↓ 15.	Amazon DynamoDB	Multi-model	74.07	-3.50	-7.04
18.	18.	↓ 17.	Hive	Relational	61.17	-1.41	-12.44
19.	19.	↑ 20.	Google BigQuery	Relational	60.38	-1.52	+5.51
20.	20.	↑ 21.	FileMaker	Relational	48.20	-1.53	-3.80
21.	21.	↓ 19.	Teradata	Relational, Multi-model	45.33	-2.52	-17.39
22.	22.	↑ 23.	SAP HANA	Relational, Multi-model	44.69	-1.14	-5.67
23.	23.	↓ 22.	Neo4j	Graph	44.46	-0.01	-6.65

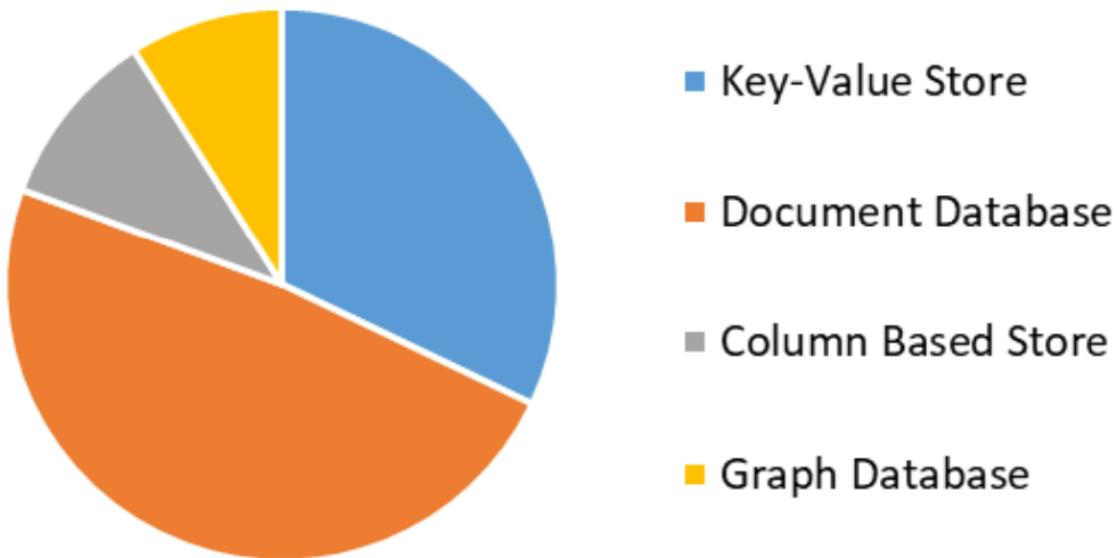
Most popular databases

<https://survey.stackoverflow.co/2022#technology>



Most popular NoSQL databases

NoSQL Database Market, by Type 2022 (%)

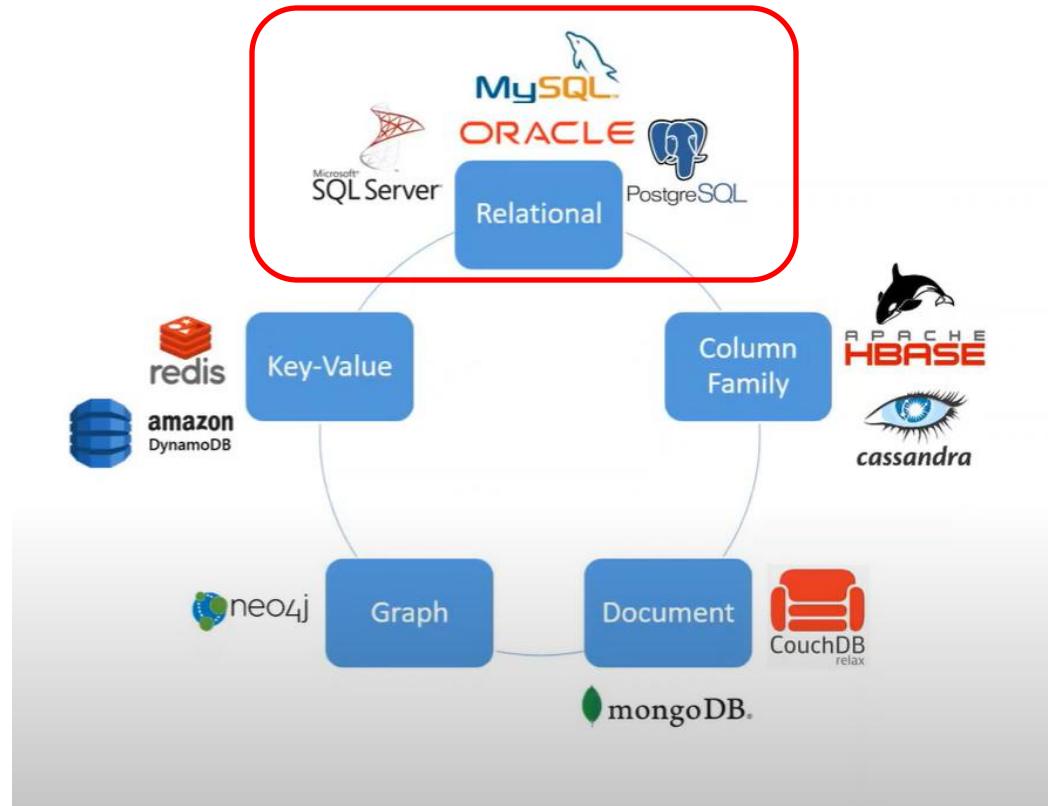


Cloud database

Cloud Database Cheat Sheet

 blog.bytebytego.com

DB Type		aws	Azure	Google Cloud	Open Source / 3rd Party	
Structured	Relational	 RDS	 SQL Database	 Cloud SQL	 Oracle	 PostgreSQL
	Columnar	 Redshift	 Synapse Analytics	 BigQuery	 MySQL	 SQL Server
	Key Value	 DynamoDB	 Cosmos DB	 BigTable	 Snowflake	 Click House
	In-Memory	 ElastiCache	 Azure Cache for Redis	 Memory Store	 Redis	 Scylla
	Wide Column	 Keyspaces	 Cosmos DB	 BigTable	 Redis	 Memcached
	Time Series	 Timestream	 Time Series Insights	 BigTable	 Cassandra	 Scylla
	Immutable Ledger	 Quantum Ledger DB	 Confidential Ledger	 CloudSpanner	 Influx	 OpenTSDB
	Geospatial	 Keyspaces	 Cosmos DB	 BigTable	 Hyper Ledger Fabric	
	Graph	 Neptune	 Cosmos DB	 CloudSpanner	 PostGIS	 geomesa
	Document	 Document DB	 Cosmos DB	 FireStore	 OrientDB	 Dgraph
Semi Structured	Text Search	 OpenSearch	 Cognitive Search	 CloudSearch	 MongoDB	 Couchbase
	Blob	 S3	 Blob Storage	 Cloud Storage	 Elasticsearch	 Elassandra
UnStructured					 Ceph	 OpenIO



RELATIONAL DATABASE

Relational databases

Data is normalized:
No duplication of
data

Key

Relationships are
expressed with
key's, foreign keys
and link tables

Relational - Tables

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	Black	Newark
3	Meagan	White	London
4	Edward	Daniels	Boston

Account Number	Branch ID	Account Type	Customer ID
10	100	Checking	0
11	101	Savings	0
12	101	IRA	0
13	200	Checking	1
14	200	Savings	1
15	201	IRA	2

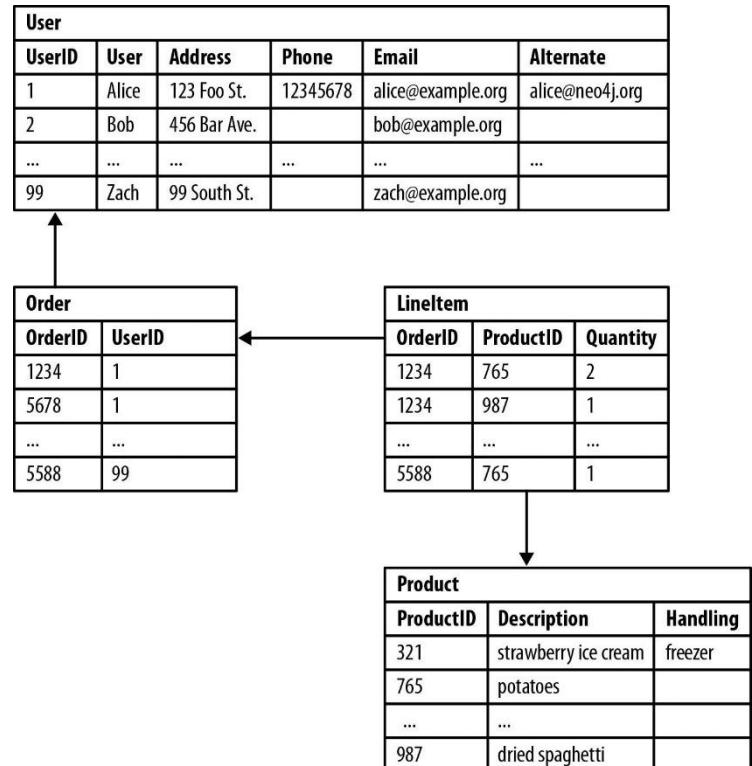
SQL as language to
manage data in
the database

Foreign key

To find related
data you need to
JOIN tables
together

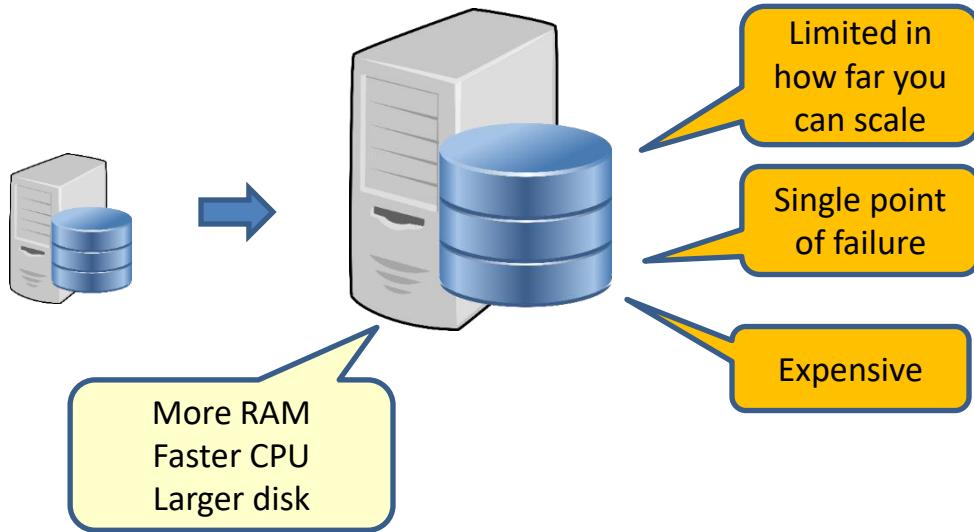
Relational database characteristics

- Structured data
- Normalized
 - No duplication
- ACID transactions
- Fixed schema
- SQL as query language
- Scale vertically
 - Difficult to scale horizontally

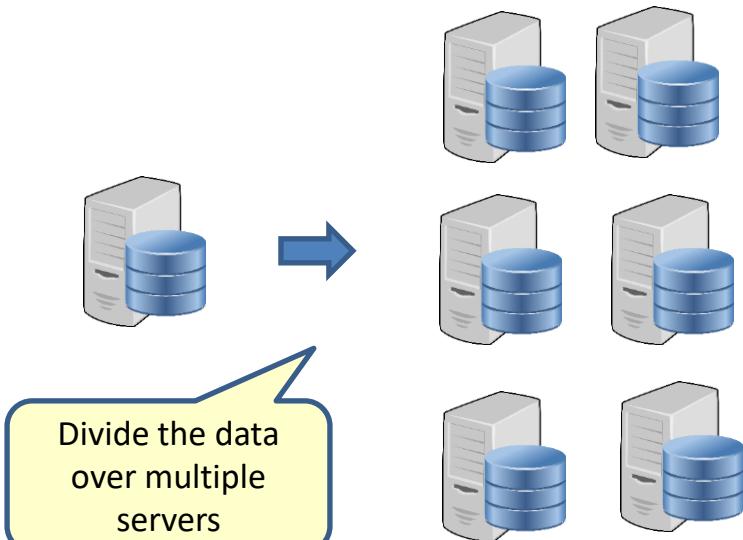


Database Scaling

- Vertical scaling



- Horizontal scaling



Horizontal scaling

- 2 techniques
 1. Sharding/partitioning: Divide the data over the database instances
 2. Replication: make a copy of the data on different database instances



Sharding



Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDÄ	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

HP1

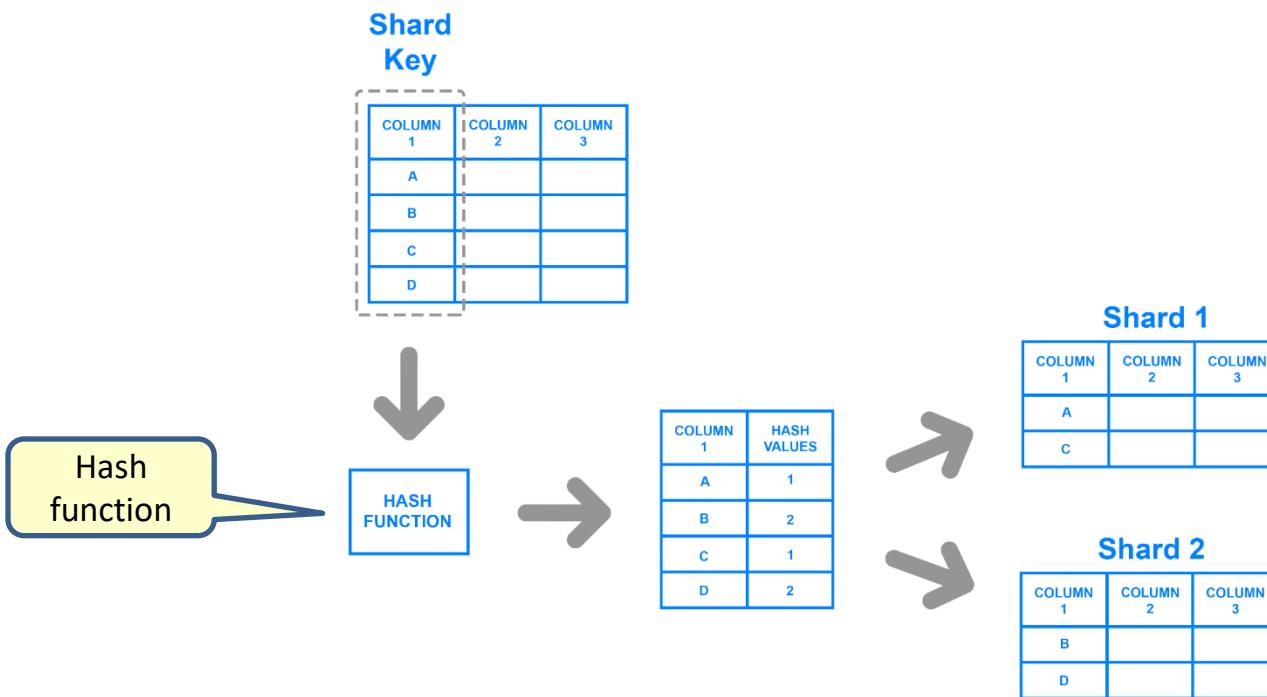
CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDÄ	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE



Key based sharding



Range based sharding



PRODUCT	PRICE
WIDGET	\$118
GIZMO	\$88
TRINKET	\$37
THINGAMAJIG	\$18
DOODAD	\$60
TCHOTCHKE	\$999



(\$0-\$49.99)



(\$50-\$99.99)



(\$100+)

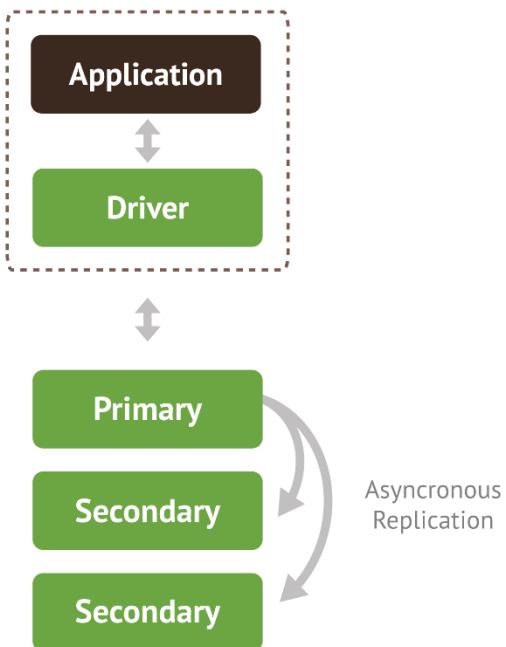
PRODUCT	PRICE
TRINKET	\$37
THINGAMAJIG	\$18

PRODUCT	PRICE
GIZMO	\$88
DOODAD	\$60

PRODUCT	PRICE
WIDGET	\$118
TCHOTCHKE	\$999



Replica Sets



- Replica Set – two or more copies
- Failover
 - “Self-healing” shard
- Addresses many concerns:
 - High Availability
 - Disaster Recovery
 - Maintenance

Relational database advantages

- Data consistency and integrity
 - ACID transactions
- Data accuracy
 - No duplication
- Standard query language (SQL)
- Years of experience
- Structured database design
 - Normalization



Problems with relational databases

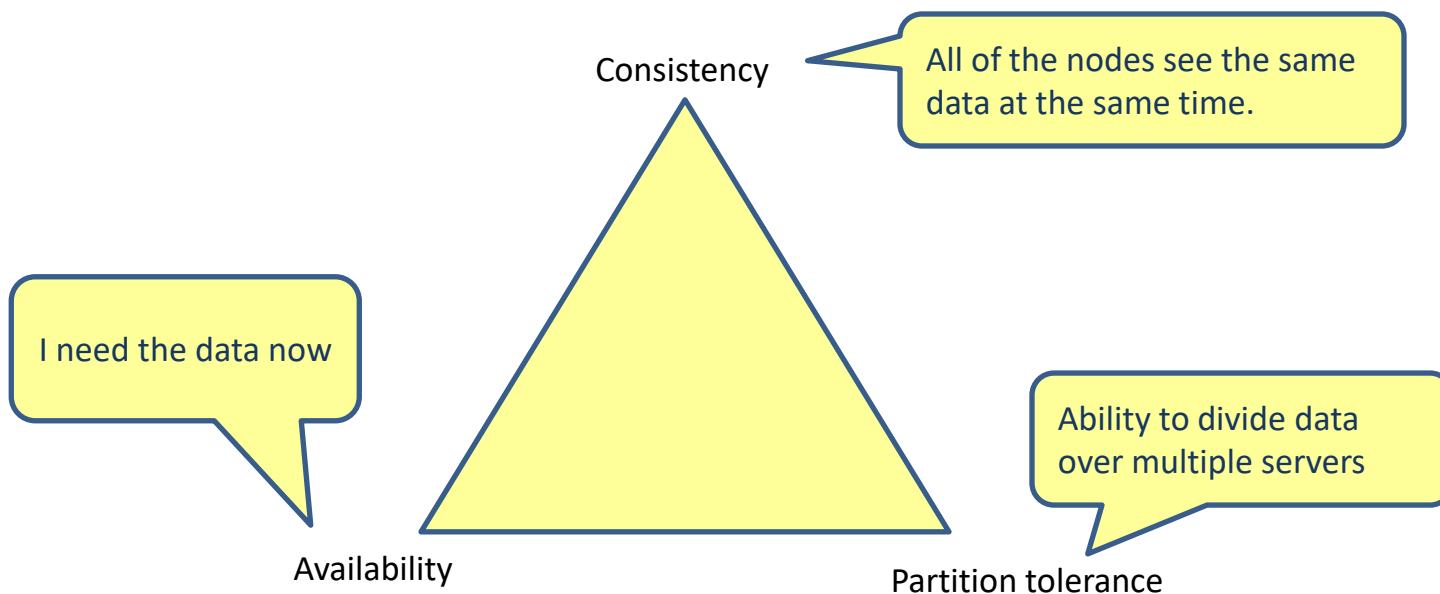
- Scaling writes are very difficult and limited
 - Vertical scaling is limited and is expensive
 - Horizontal scaling is limited and is complex
 - Queries work only within shards
 - Strict consistency and partition tolerance leads to availability problems

A relational database is hard to scale



Brewer's CAP Theorem

- A distributed system can support only two of the following characteristics



Consistency

- Strict consistency
 - The data that I read is always correct
 - You never loose data
- Eventual consistency
 - The data might not be correct
 - But will eventually become correct



Problems with relational databases

- The schema in a database is fixed
- Schema evolution
 - Adding attributes to an object => have to add columns to table
 - You need to do a migration project
 - Application downtime ...

A relational database is hard to change



Problems with relational databases

- Relational schema doesn't easily handle unstructured and semi-structured data
 - Emails
 - Tweets
 - Pictures
 - Audio
 - Movies
 - Text

Unstructured data				Semi-structured data				Structured data					
ID	Name	Age	Degree	ID	Name	Age	Degree	ID	Name	Age	Degree		
1	John	18	B.Sc.	<University>	<Student ID="1">	<Name>John</Name>	<Age>18</Age>	<Degree>B.Sc.</Degree>	</Student>	2	David	31	Ph.D.
2	David	31	Ph.D.	<Student ID="2">	<Name>David</Name>	<Age>31</Age>	<Degree>Ph.D. </Degree>	</Student>	3	Robert	51	Ph.D.	
3	Robert	51	Ph.D.					4	Rick	26	M.Sc.	
4	Rick	26	M.Sc.	</University>					5	Michael	19	B.Sc.	
5	Michael	19	B.Sc.										

A relational database does not handle unstructured and semi structured data very well



Relational database disadvantages

- Schema evolution is difficult
- Horizontal scaling is difficult
- Does not handle unstructured data very well
 - Not good in full text search
- Queries can be slow due to joins



Non relational databases

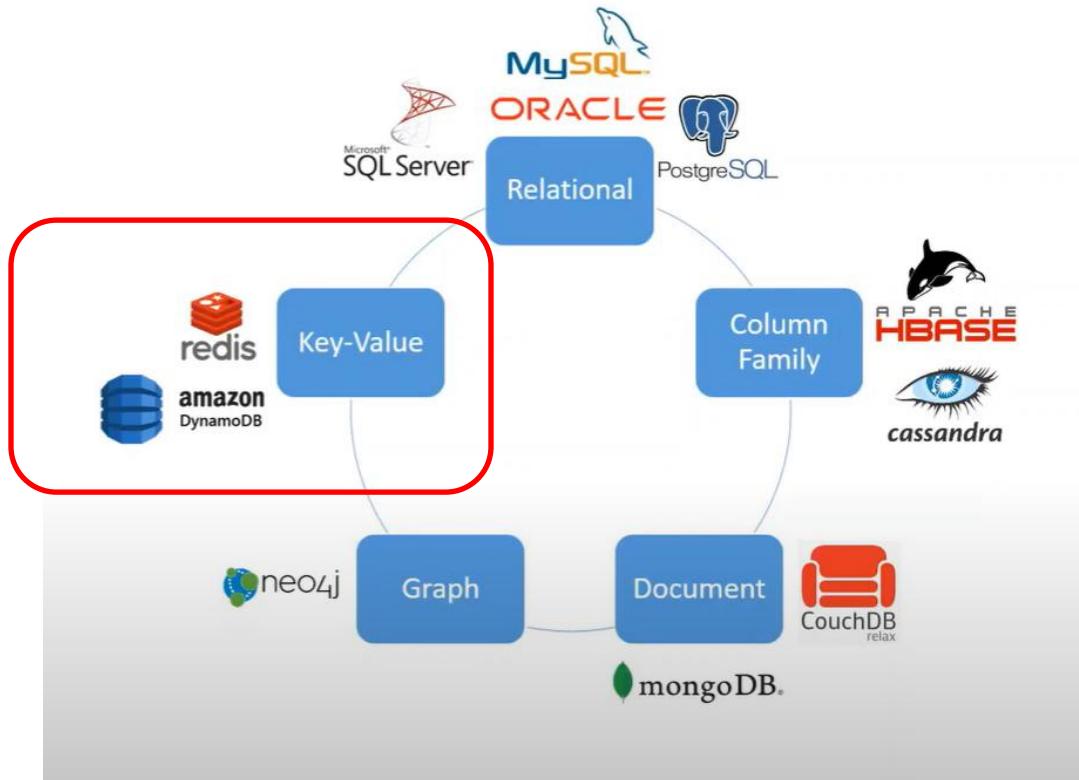
- Schema free
- Distributed
 - Horizontal scalable
 - Replicated
- Some support unstructured or semi-structured data
- BASE



BASE

- Basically available
 - All users can concurrently access the data without having to wait
- Soft state
 - Data can have transient or temporary states that change over time
- Eventual consistency
 - The data will become consistent eventually

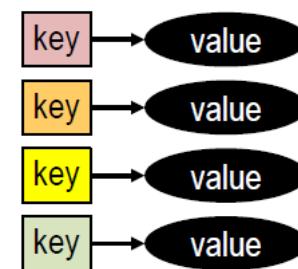




KEY VALUE STORE

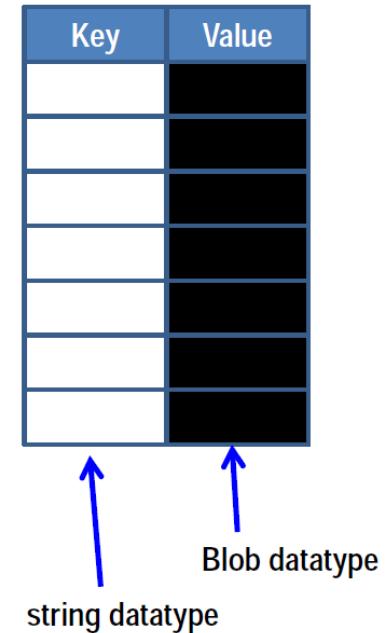
Key-value store

- One key -> one value



Key-value store

- One key -> one value
 - Simple hash table
 - Very fast
- Value is a binary object (BLOB)
 - DB does not understand the content
- Use cases
 - Storing session data
 - User profiles and preferences
 - Shopping cart data



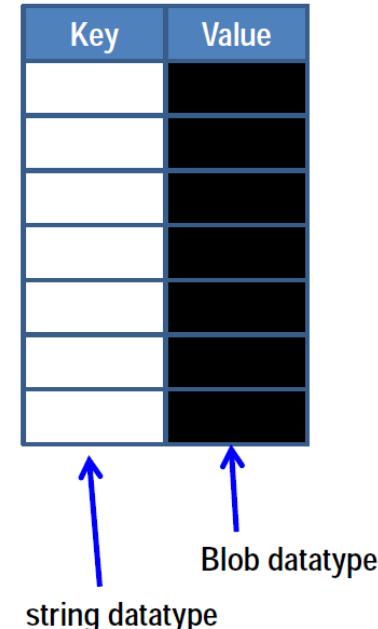
Key-value store

- **Pros:**

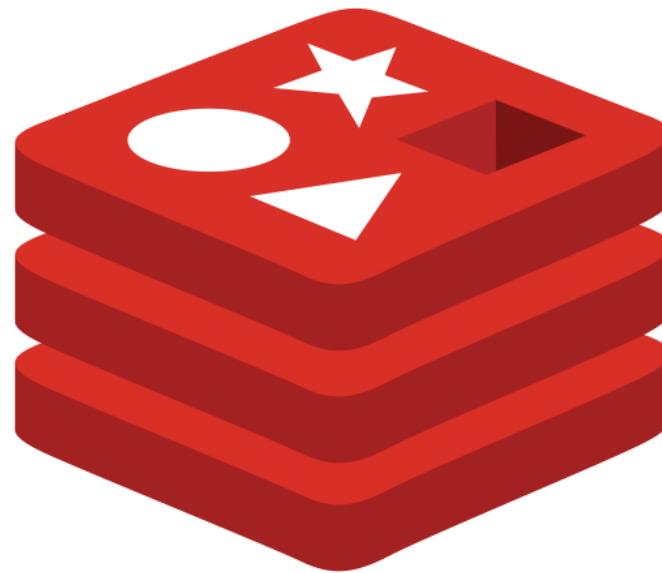
- Very fast
- Scalable
- Simple API
 - Put(key, value)
 - Get(key)
 - Delete(key)

- **Cons:**

- No way to query based on the content of the value



REDIS

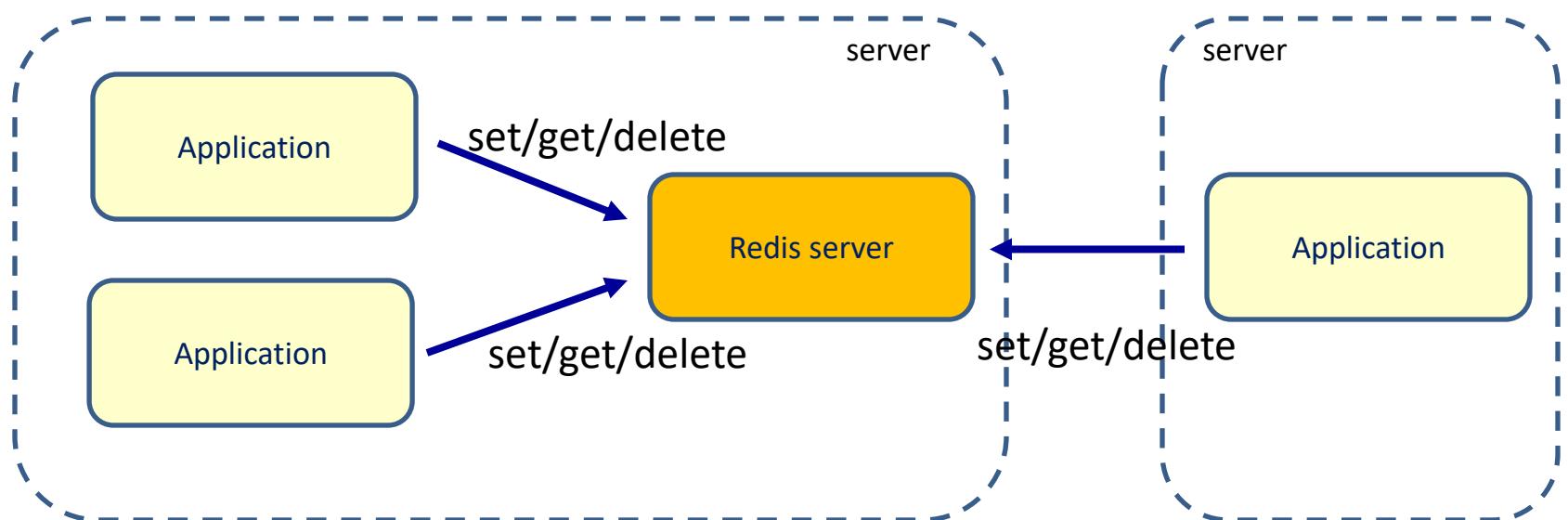


Redis

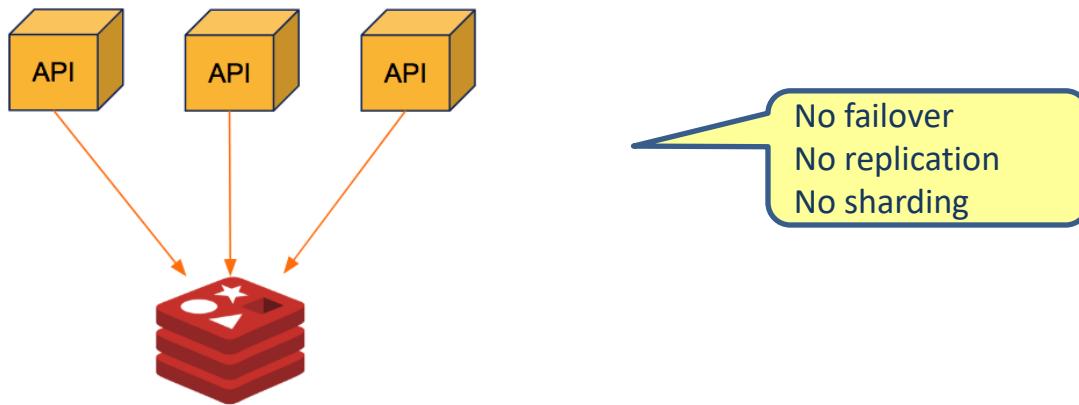
- REmote DIctionary Service
- Key-value store
- Very, very fast
 - Microseconds (not milliseconds)
- “in-memory” database
 - Can periodically write to disk
 - Size of data is limited by amount of memory in the sever



Using redis

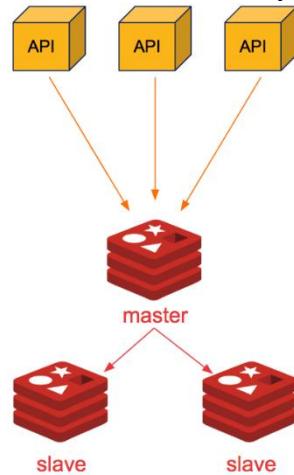


One redis instance



Master slave replication and sentinel

Master slave replication

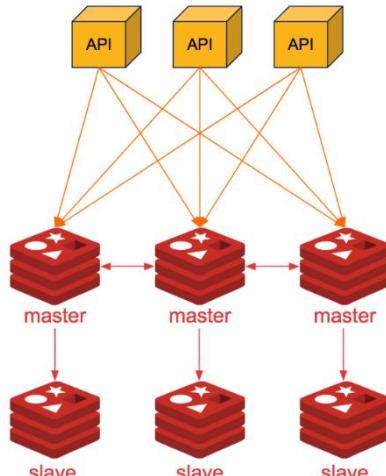


Failover because of replication

Slaves are read only

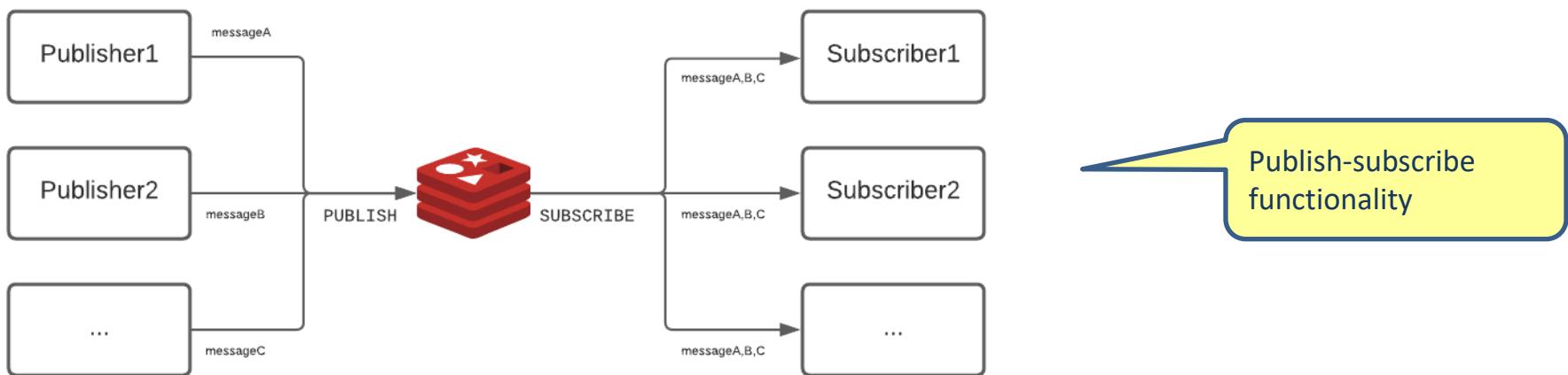
No automatic failover

Redis cluster



Automatic failover
Replication
Key based sharding

Redis streams



Redis advantages & disadvantages

- Advantages
 - Very fast in memory persistence
 - Scalable
 - High available
 - Message functionality
 - Pub-sub
- Disadvantages
 - Not for long term data persistence
 - You cannot query the value
 - No query language
 - Storage is limited to available memory



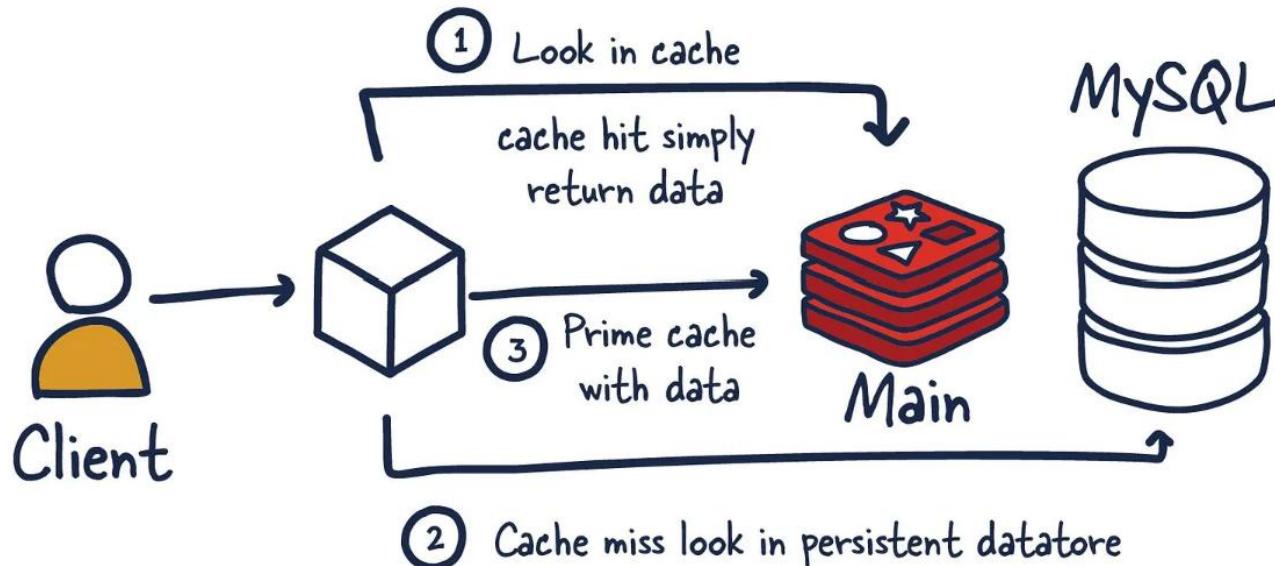
Redis use cases

- In memory cache
 - Data that changes infrequently and is requested often
 - Data that is less mission-critical and is frequently evolving
- Session management
- Queue systems
- Often used in combination with another database for permanent storage



Redis use cases

- Often used in combination with another database for permanent storage



SPRING BOOT REDIS EXAMPLE



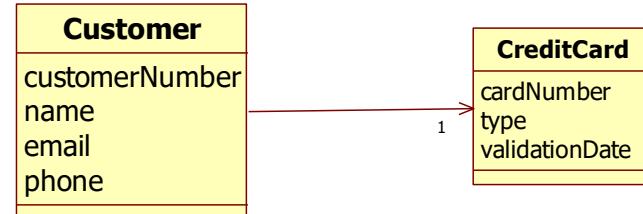
Domain classes

```
@RedisHash("Customer")
```

```
public class Customer {  
    @Id  
    private int customerNumber;  
    private String name;  
    private String email;  
    private String phone;  
    private CreditCard creditCard;
```

```
...  
  
public class CreditCard {  
    private String cardNumber;  
    private String type;  
    private String validationDate;
```

```
...
```



```
Application
```

```
main()  
run()
```

```
CustomerRepository
```

```
save()  
findById()  
findAll()
```

```
DI
```

Repository



```
@Repository  
public interface CustomerRepository extends CrudRepository<Customer, Integer> {}
```



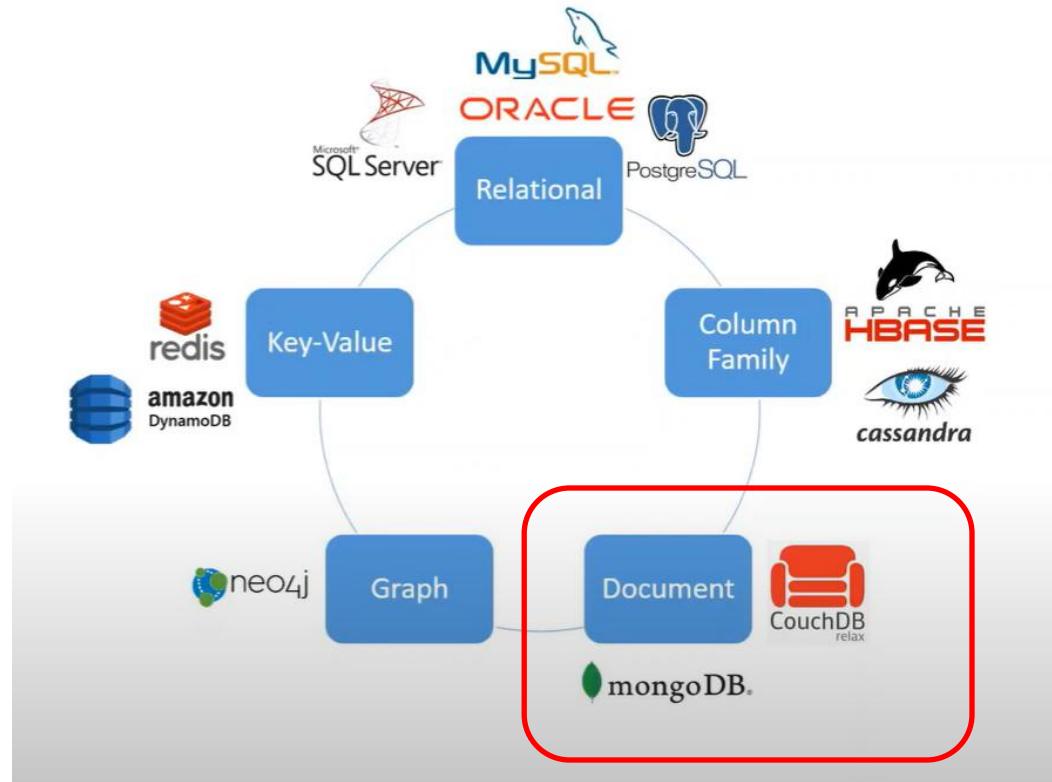
application.properties

```
# Redis configuration.  
spring.redis.host=localhost  
spring.redis.port=6379  
  
logging.level.root=ERROR  
logging.level.org.springframework=ERROR
```



Application

```
public void run(String... args) throws Exception {
    // create customer
    Customer customer = new Customer(101, "John doe", "johnd@acme.com", "0622341678");
    CreditCard creditCard = new CreditCard("12324564321", "Visa", "11/23");
    customer.setCreditCard(creditCard);
    customerRepository.save(customer);
    customer = new Customer(66, "James Johnson", "jj123@acme.com", "068633452");
    creditCard = new CreditCard("99876549876", "MasterCard", "01/24");
    customer.setCreditCard(creditCard);
    customerRepository.save(customer);
    //get customers
    System.out.println(customerRepository.findById(66).get());
    System.out.println(customerRepository.findById(101).get());
    System.out.println("-----All customers -----");
    System.out.println(customerRepository.findAll());
    //update customer
    customer = customerRepository.findById(101).get();
    customer.setEmail("jd@gmail.com");
    customerRepository.save(customer);
    //delete customer
    customerRepository.deleteById(66);
    System.out.println("-----All customers -----");
    System.out.println(customerRepository.findAll());
}
```



DOCUMENT DATABASE

Document database

- Store and retrieve documents
 - Like files in directories
- No schema
- Store documents in JSON



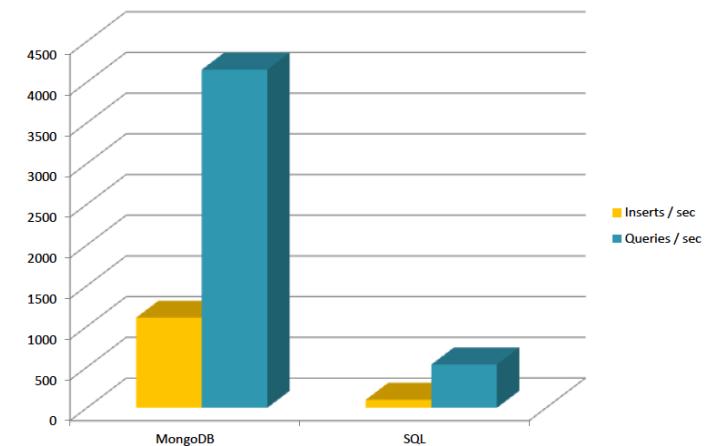


MONGODB



MongoDB

-
- Document database
 - Fast
 - Can handle large datasets



Mongo terminology

RDBMS	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Join	Embedding & Linking



Document data model (JSON)

Relational - Tables

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	Black	Newark
3	Meagan	White	London
4	Edward	Daniels	Boston

Account Number	Branch ID	Account Type	Customer ID
10	100	Checking	0
11	101	Savings	0
12	101	IRA	0
13	200	Checking	1
14	200	Savings	1
15	201	IRA	2

Document - Collections

```
{   customer_id : 1,  
    first_name : "Mark",  
    last_name : "Smith",  
    city : "San Francisco",  
    accounts : [    {  
        account_number : 13,  
        branch_ID : 200,  
        account_type : "Checking"  
    },  
    {   account_number : 14,  
        branch_ID : 200,  
        account_type : "IRA",  
        beneficiaries: [...]  
    } ]  
}
```



Documents are rich data structures

```
{  
    first_name: 'Paul',  
    surname: 'Miller',  
    cell: 447557505611,  
    city: 'London',  
    location: [45.123,47.232],  
    Profession: ['banking', 'finance',  
    'trader'],  
    cars: [  
        { model: 'Bentley',  
         year: 1973,  
         value: 100000, ... },  
        { model: 'Rolls Royce',  
         year: 1965,  
         value: 330000, ... }  
    ]  
}
```

Fields

String
Number
Geo-Coordinates

Typed field values

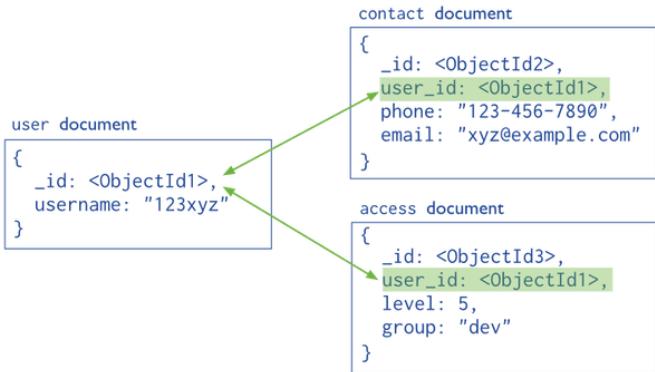
Fields can contain arrays

Fields can contain an array of sub-documents



References vs. embedded data

Reference



Embedded data



Schema free

```
{name: "will",  
eyes: "blue",  
birthplace: "NY",  
aliases: ["bill", "la  
ciacco"],  
gender: "????",  
boss:"ben"}
```

```
{name: "jeff",  
eyes: "blue",  
height: 72,  
boss: "ben"}
```

```
{name: "ben",  
hat:"yes"}
```

```
{name: "brendan",  
aliases: ["el diablo"]}
```

```
{name: "matt",  
pizza: "DiGiorno",  
height: 72,  
boss: 555-555-1212}
```



Find() method

SQL SELECT Statements

`SELECT * FROM users`

`SELECT id, user_id, status FROM users`

`SELECT user_id, status FROM users`

`SELECT * FROM users WHERE status = "A"`

`SELECT user_id, status FROM users WHERE status = "A"`

`SELECT * FROM users WHERE status != "A"`

`SELECT * FROM users WHERE status = "A" AND age = 50`

`SELECT * FROM users WHERE status = "A" OR age = 50`

`SELECT * FROM users WHERE age > 25`

MongoDB find() Statements

`db.users.find()`

`db.users.find({}, { user_id: 1, status: 1 })`

`db.users.find({}, { user_id: 1, status: 1, _id: 0 })`

`db.users.find({ status: "A" })`

`db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })`

`db.users.find({ status: { $ne: "A" } })`

`db.users.find({ status: "A", age: 50 })`

`db.users.find({ $or: [{ status: "A" } , { age: 50 }] })`

`db.users.find({ age: { $gt: 25 } })`



How to structure data in mongo?

To get high performance,
every query should be
done on 1 node only



Data is sharded
over multiple nodes

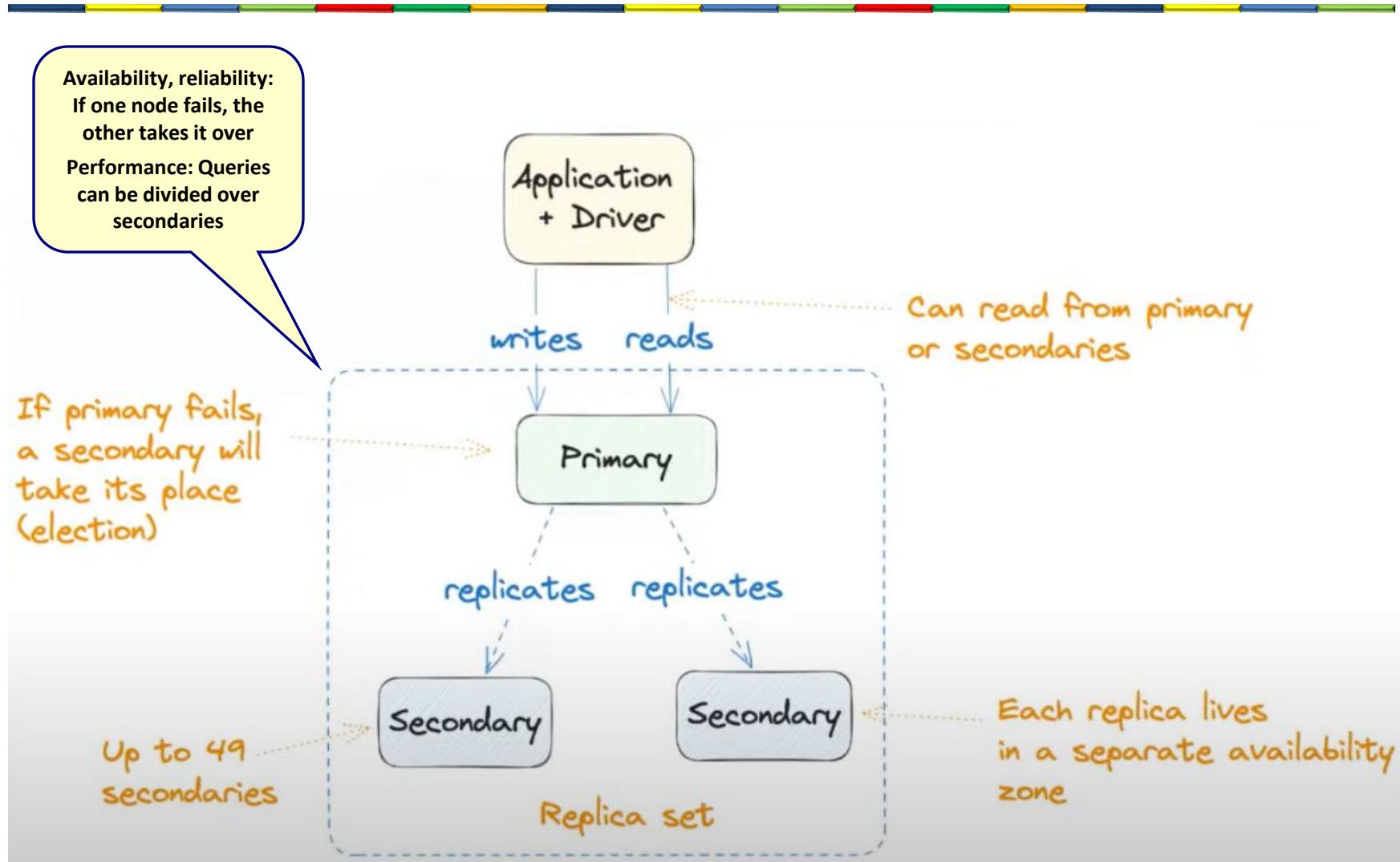


Structure the data
according your
queries

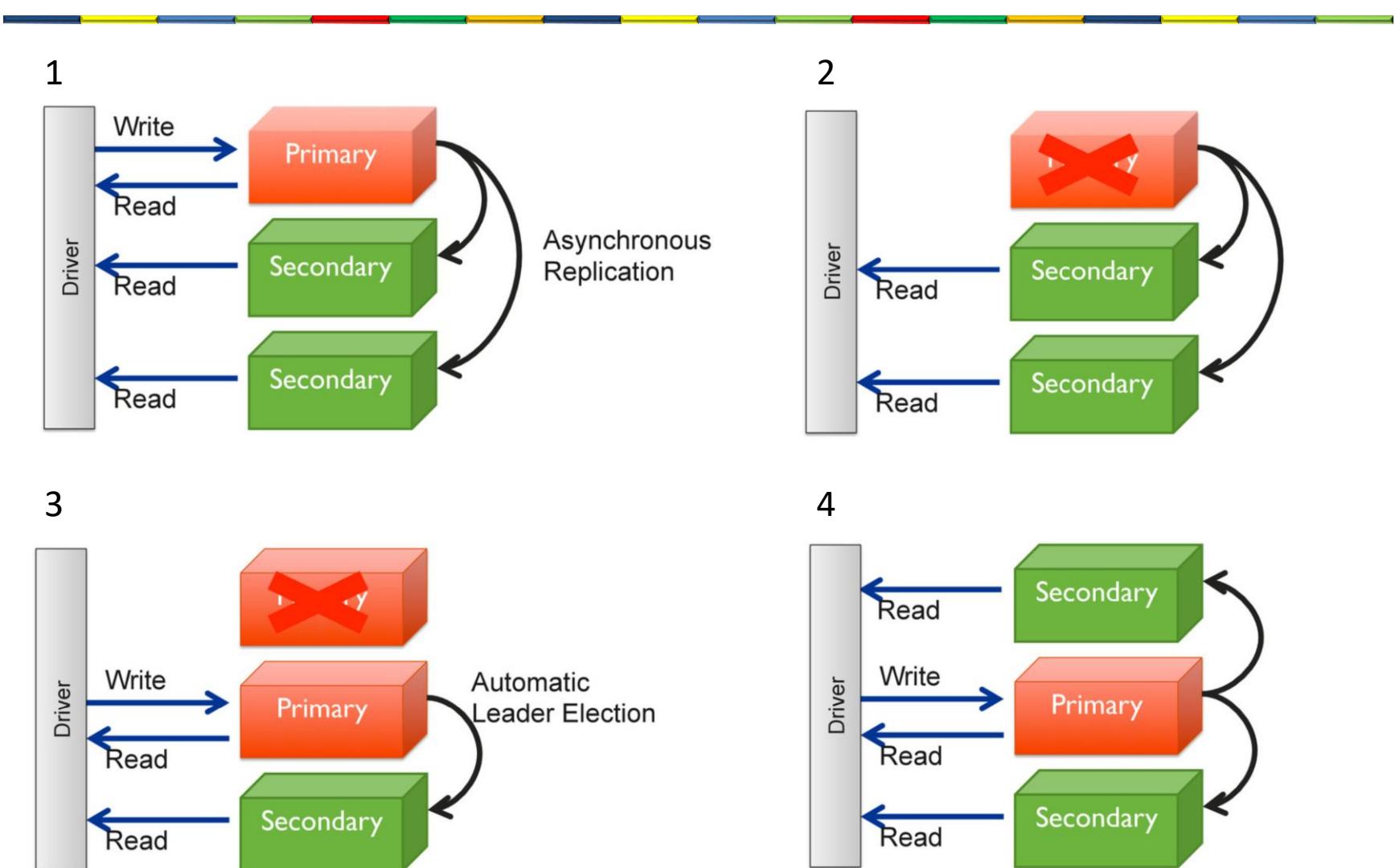
Consequence: data
duplication



MongoDB replication

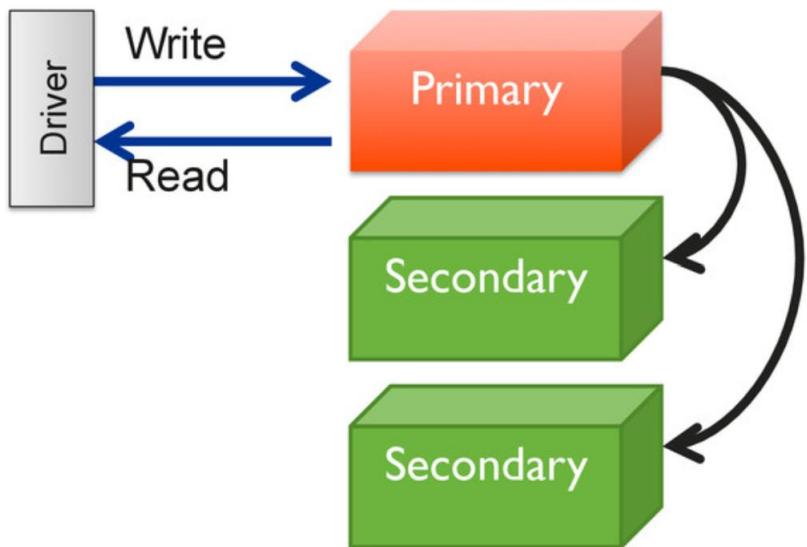


Replica sets

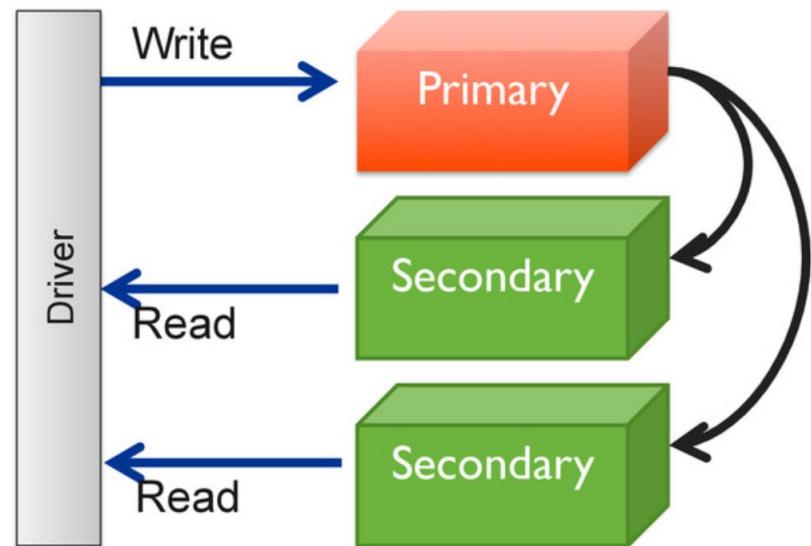


Consistency

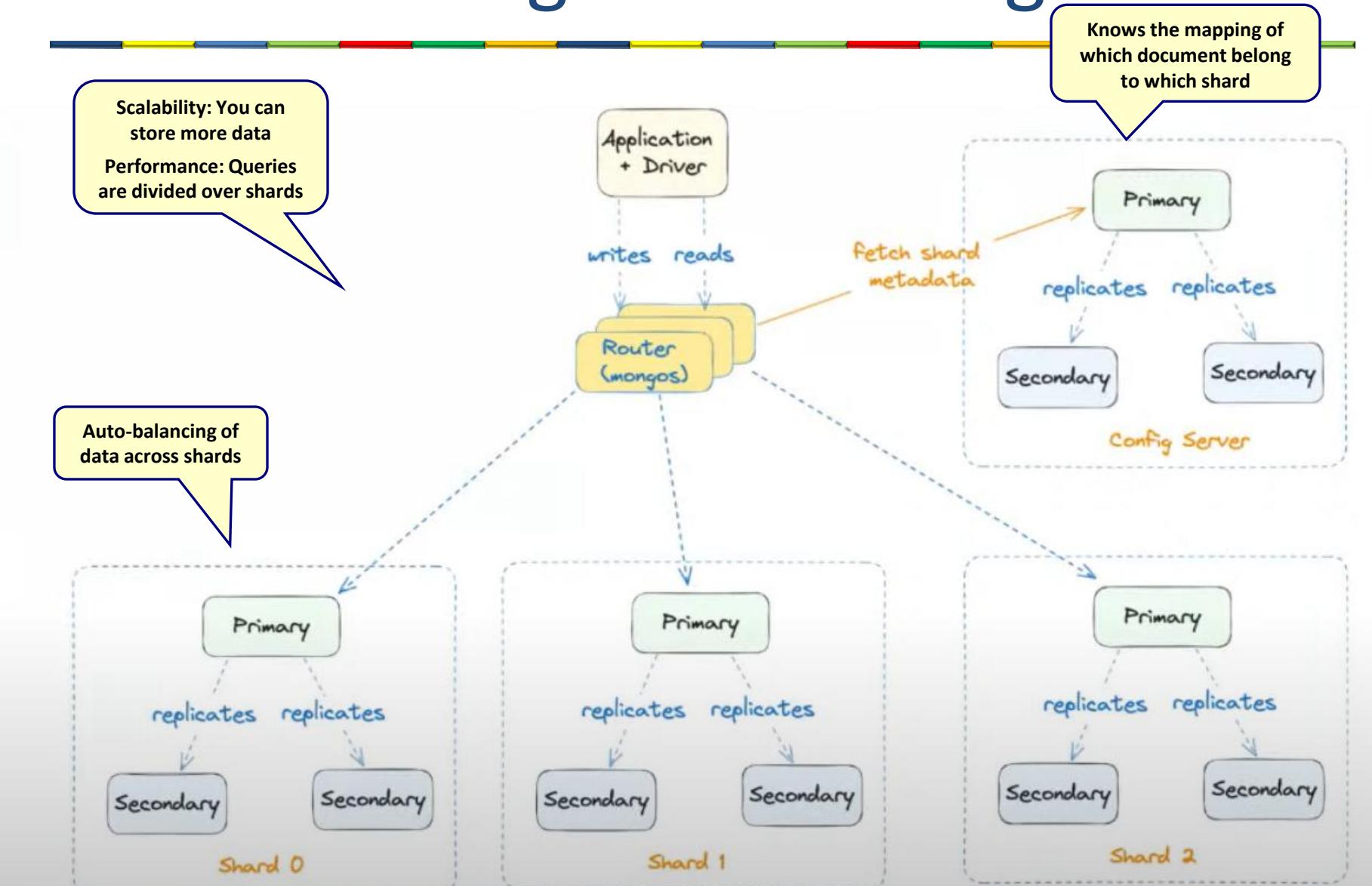
Strong consistency



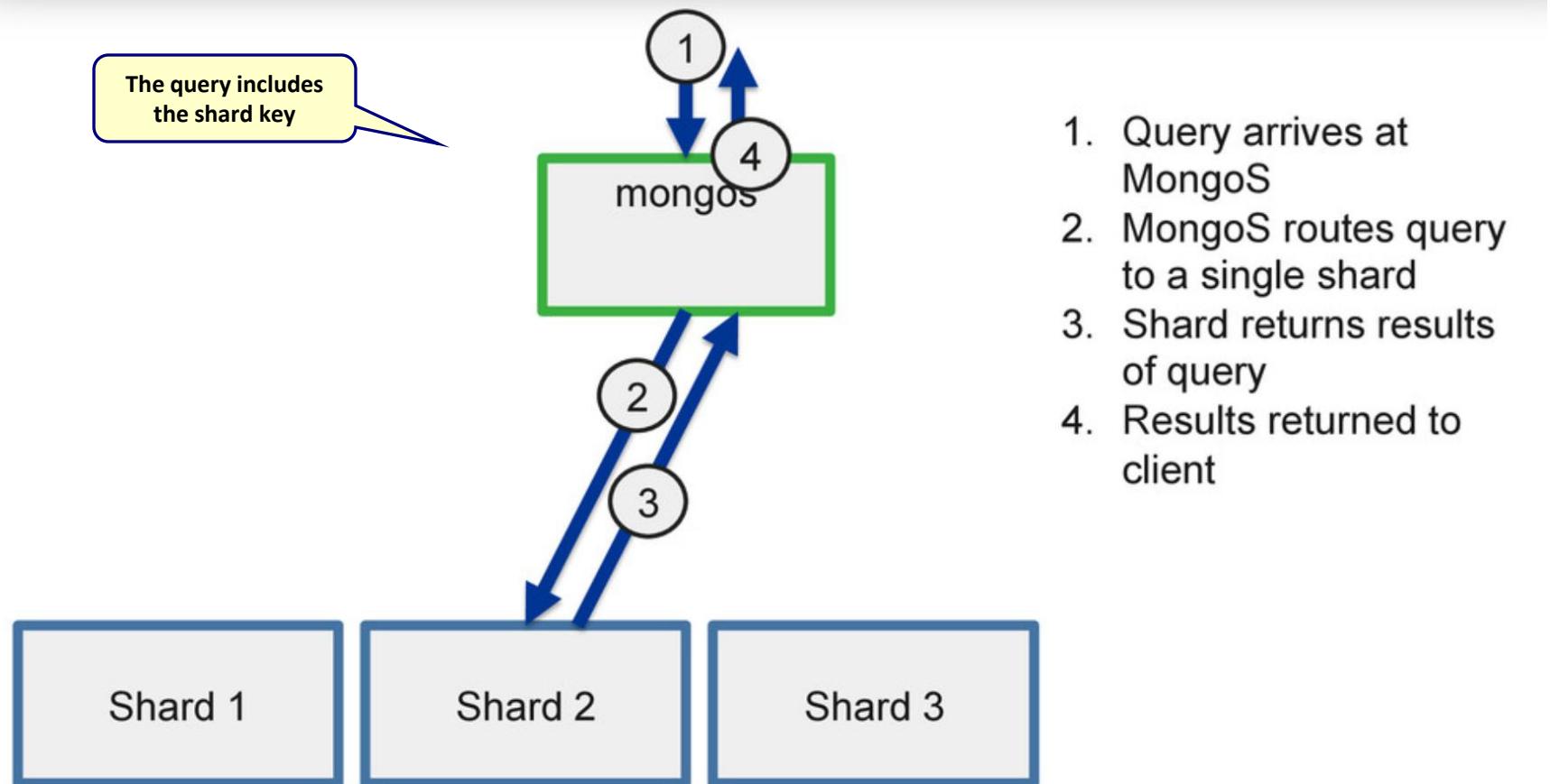
Eventual consistency



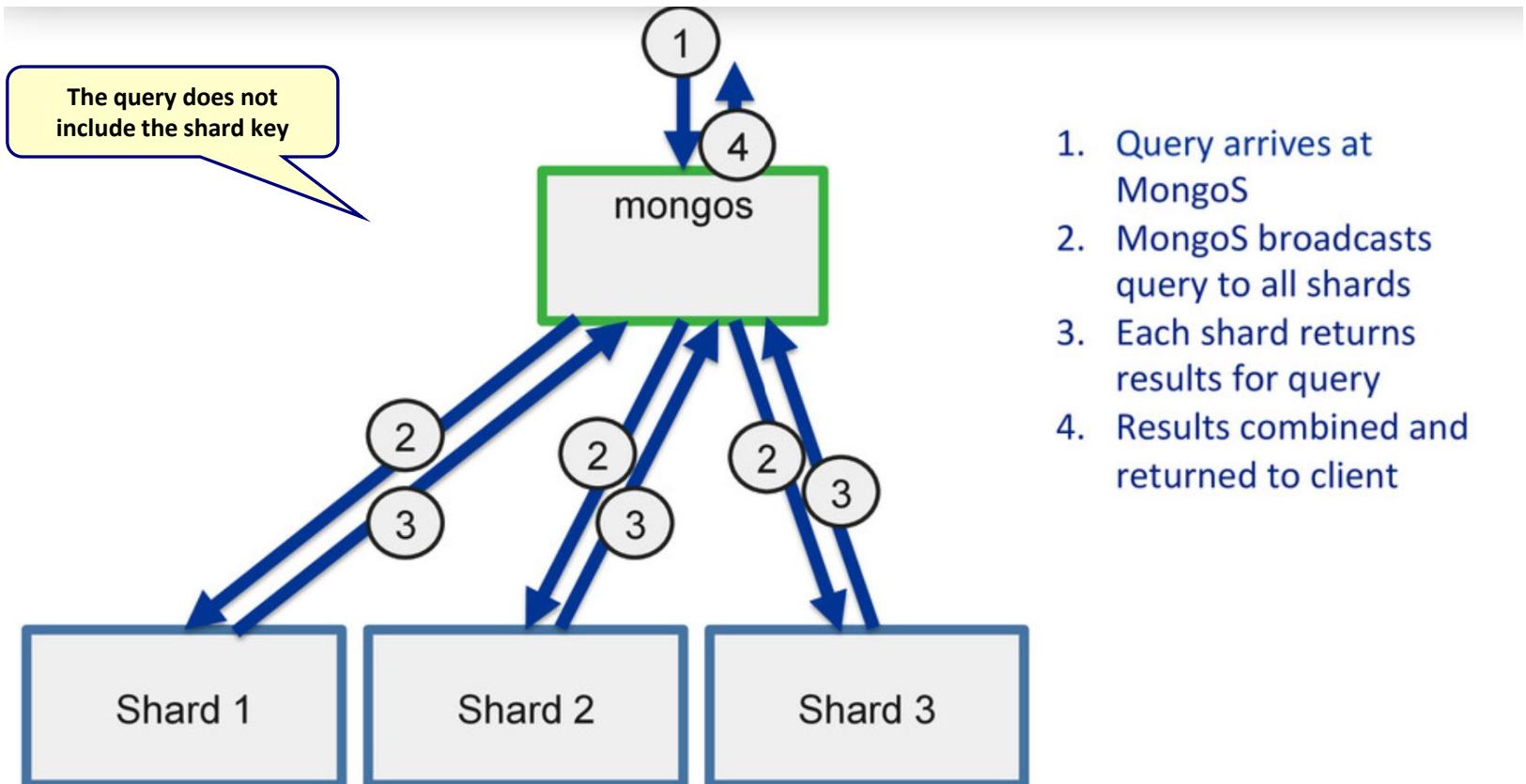
MongoDB sharding



Routed query



Scatter gather



Mongo advantages & disadvantages

- Advantages
 - Very fast
 - Scalable
 - High available
 - Handles unstructured data
 - Rich query language
 - Indexing
 - No schema
 - ACID transaction support
- Disadvantages
 - Limited joins
 - Data redundancy
 - Document size limit (16 MB)
 - Limited nested documents levels (100)



Mongo use cases

- IoT
 - Large growing data sets, flexible schema
- Real-time analytics
 - Large growing data sets, flexible schema
- Content Management
 - Large growing data sets, flexible schema
- Product catalog
 - Large growing data sets, flexible schema



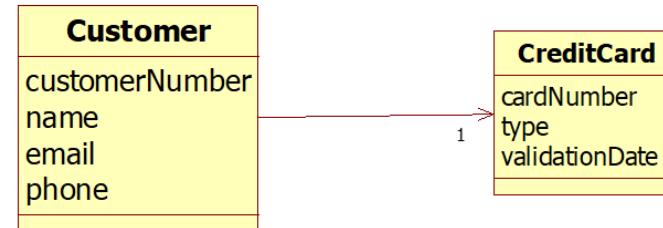
SPRING BOOT MONGO EXAMPLE



Customer

```
@Document  
public class Customer {  
    @Id  
    private int customerNumber;  
    private String name;  
    private String email;  
    private String phone;  
    private CreditCard creditCard;  
    ...}
```

```
public class CreditCard {  
    private String cardNumber;  
    private String type;  
    private String validationDate;  
    ...}
```



Repository

```
@Repository  
public interface CustomerRepository extends MongoRepository<Customer,  
Integer> {  
    Customer findByPhone(String phone);  
    Customer findByName(String name);  
  
    @Query("{email : ?0}")  
    Customer findCustomerWithPhone(String email);  
  
    @Query("{creditCard.type : ?0}")  
    List<Customer> findCustomerWithCreditCardType(String ctype);  
}
```



Application (1/2)

```
public void run(String... args) throws Exception {
    //create customer
    Customer customer = new Customer(101,"John doe", "johnd@acme.com", "0622341678");
    CreditCard creditCard = new CreditCard("12324564321", "Visa", "11/23");
    customer.setCreditCard(creditCard);
    customerRepository.save(customer);
    customer = new Customer(109,"John Jones", "jones@acme.com", "0624321234");
    creditCard = new CreditCard("657483342", "Visa", "09/23");
    customer.setCreditCard(creditCard);
    customerRepository.save(customer);
    customer = new Customer(66,"James Johnson", "jj123@acme.com", "068633452");
    creditCard = new CreditCard("99876549876", "MasterCard", "01/24");
    customer.setCreditCard(creditCard);
    customerRepository.save(customer);
    //get customers
    System.out.println(customerRepository.findById(66).get());
    System.out.println(customerRepository.findById(101).get());
```

Application (2/2)

```
System.out.println("-----All customers -----");
System.out.println(customerRepository.findAll());
//update customer
customer = customerRepository.findById(101).get();
customer.setEmail("jd@gmail.com");
customerRepository.save(customer);
System.out.println("-----find by phone -----");
System.out.println(customerRepository.findByPhone("0622341678"));
System.out.println("-----find by email -----");
System.out.println(customerRepository.findCustomerWithPhone("jj123@acme.com"));
System.out.println("-----find customers with a certain type of creditcard -----");
List<Customer> customers = customerRepository.findCustomerWithCreditCardType("Visa");
for (Customer cust : customers){
    System.out.println(cust);
}

}
```

Mongo collections

The screenshot shows the MongoDB Compass interface connected to the 'testdb.customer' database. The left sidebar displays the 'Local' connection details, including the host 'localhost:27017', cluster 'Standalone', and edition 'MongoDB 3.2.19-14-ge59d00a Community'. The 'testdb' database is selected, and the 'customer' collection is currently viewed. The main pane shows the 'Documents' tab with three document entries. Each document is represented by a JSON object:

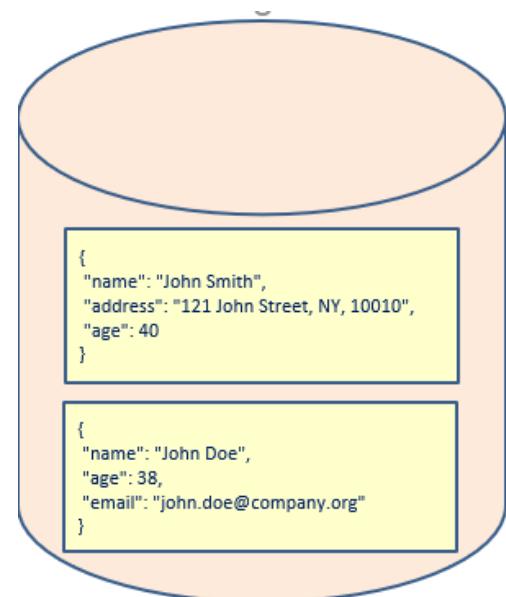
```
_id: 101
name: "John Doe"
email: "j@gmail.com"
phone: "0622341678"
creditCard: Object
  cardNumber: "12324564321"
  type: "Visa"
  validationDate: "11/23"
  _class: "customers.domain.Customer"

_id: 109
name: "John Jones"
email: "jones@acme.com"
phone: "0624321234"
creditCard: Object
  cardNumber: "657483342"
  type: "Visa"
  validationDate: "09/23"
  _class: "customers.domain.Customer"

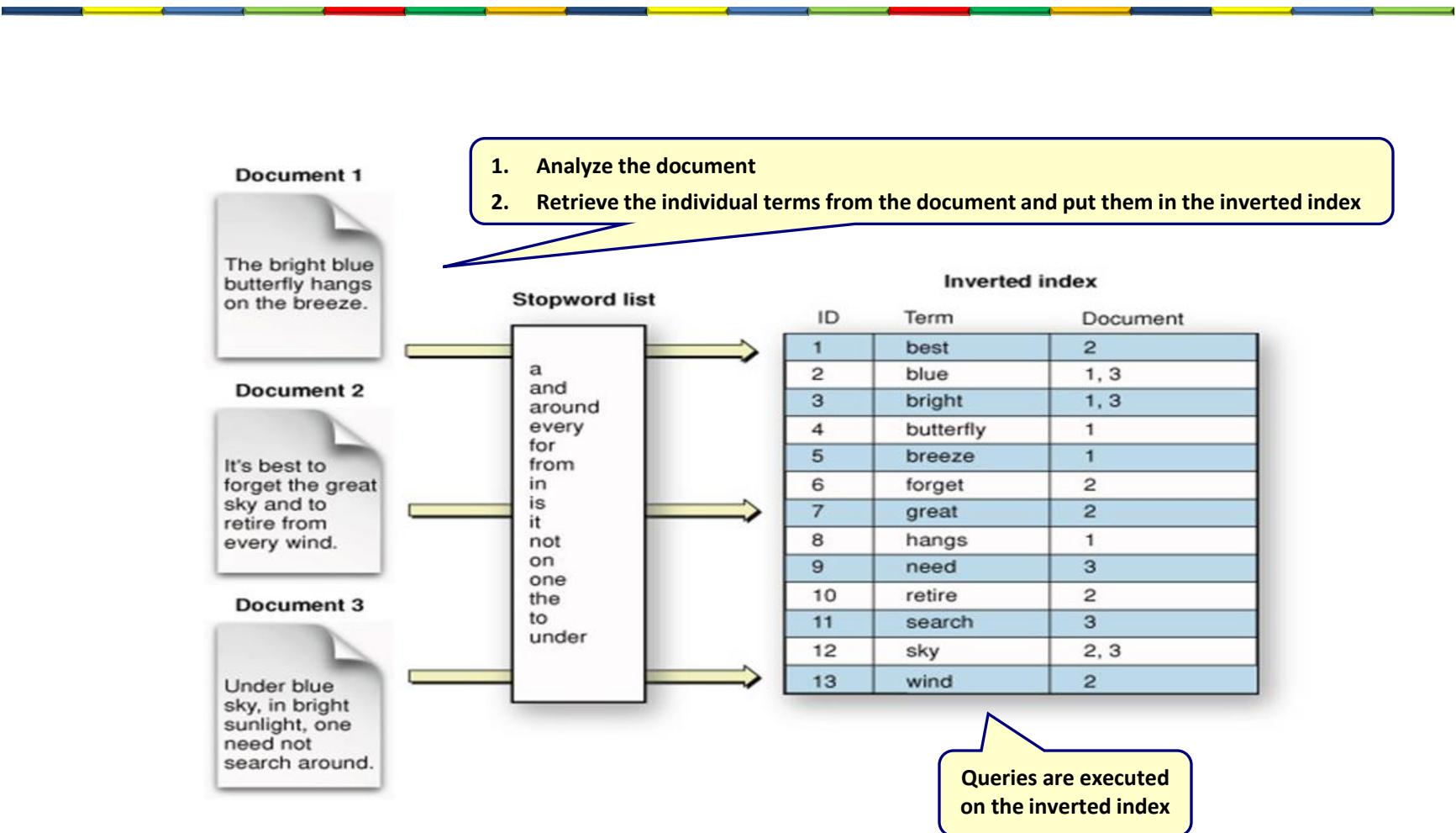
_id: 66
name: "James Johnson"
email: "jj123@acme.com"
phone: "068633452"
creditCard: Object
  cardNumber: "99876549876"
  type: "MasterCard"
  validationDate: "01/24"
  _class: "customers.domain.Customer"
```

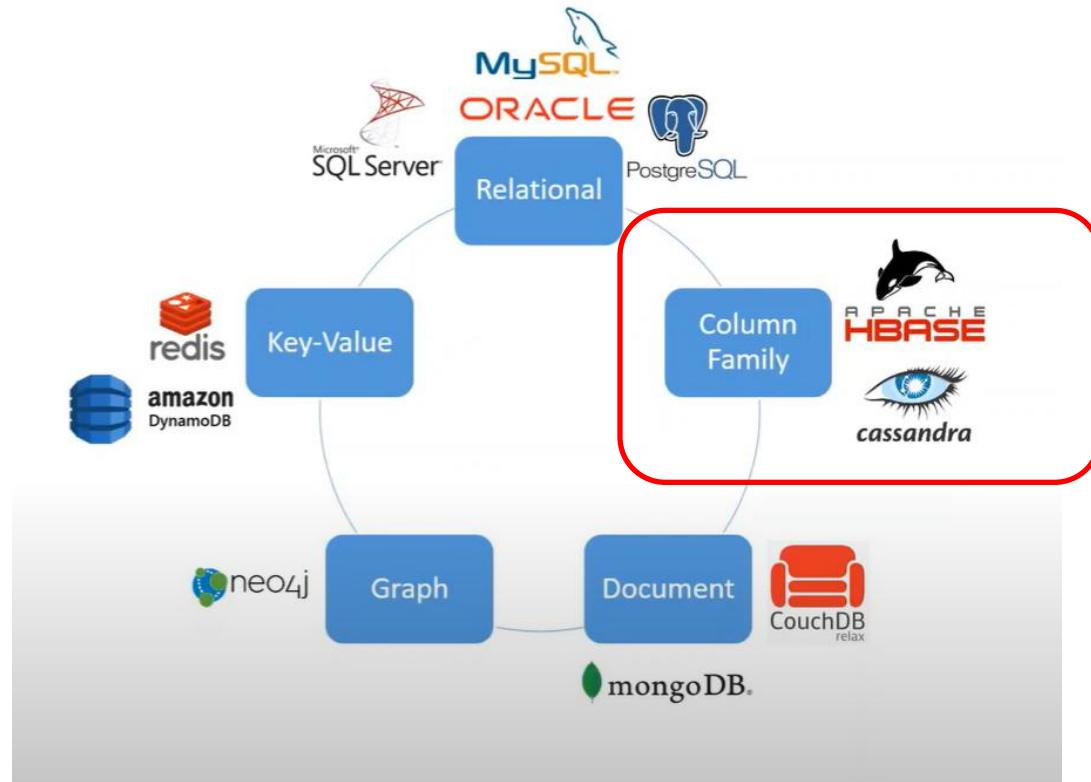
Elasticsearch

- Database
 - Data is stored as documents
 - Data is structured in JSON format
- Full text search engine



Inverted index

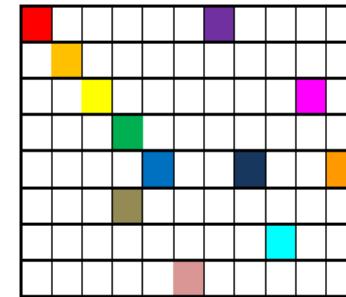




COLUMN FAMILY DATABASE

Column family

- Place data in a certain column
- Ideal for high-variability data sets
- Column families allow to query all columns that have a specific property or properties
- Allow new columns to be inserted without doing an "alter table"

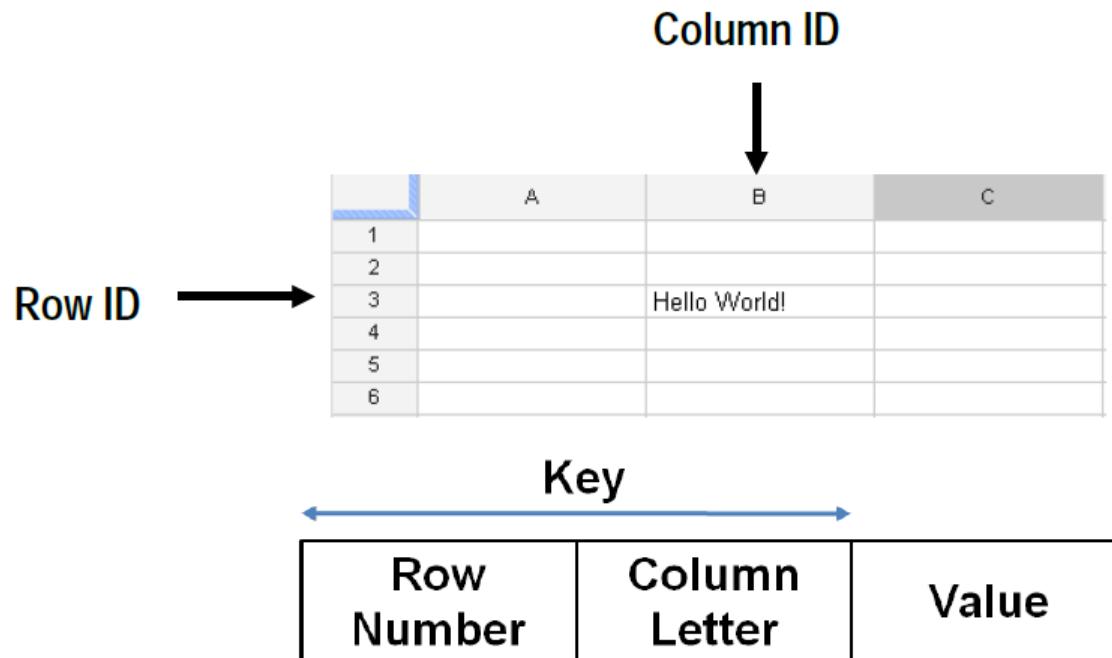


- Players
- Cassandra
- HBase
- Google BigTable

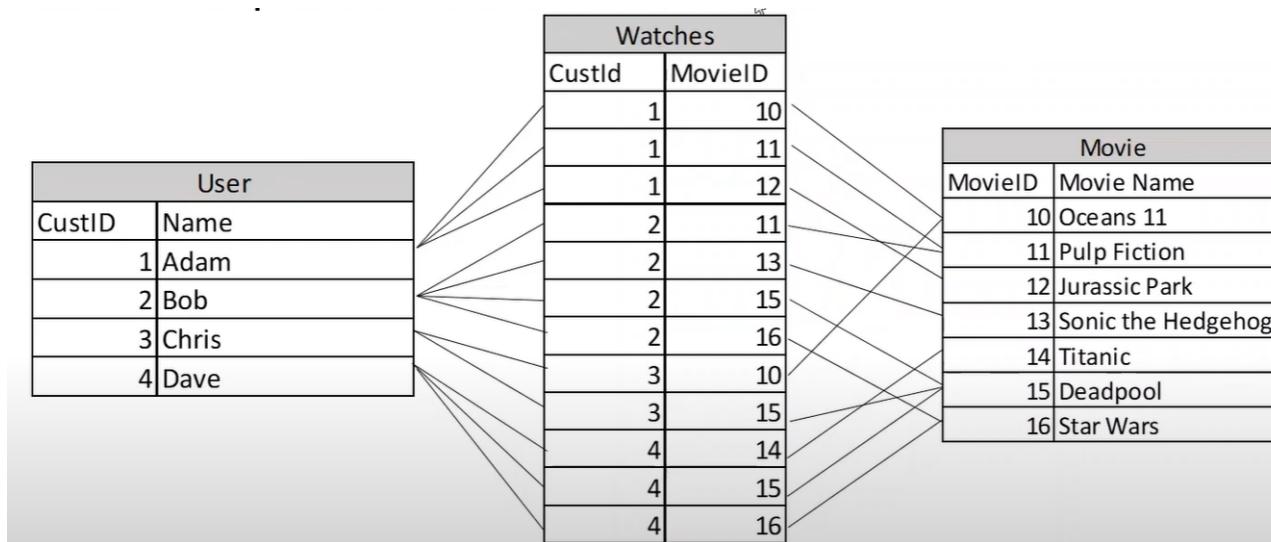


Excel

- The key is the row number and the column letter



Relational



What movies has Adam watched?



Need to join 3 tables (slow)

Who watched Deadpool?



Need to join 3 tables (slow)



Column family

User	Movies			
	Oceans 11	Pulp Fiction	Jurassic Park	
Adam	TRUE	TRUE	TRUE	
	Pulp Fiction	Sonic	Deadpool	Star Wars
Bob	TRUE	TRUE	TRUE	TRUE
	Oceans 11	Deadpool		
Chris	TRUE	TRUE		
	Titanic	Deadpool	Star Wars	
Dave	TRUE	TRUE	TRUE	

Structure the data according your queries

What movies has Adam watched?



Very easy and fast

Who watched Deadpool?



Difficult



CASSANDRA



Cassandra

- Column family NoSQL database
 - Uses tables, primary keys, queries, ...
- Unlimited elastically scalable
- Always available (no downtime)
- High performance
- Fault tolerance



High performance

MySQL Comparision:

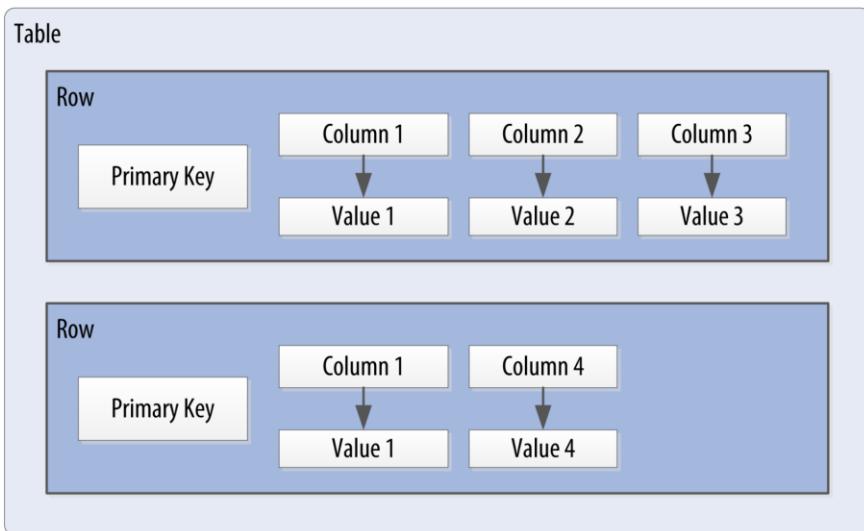


Statistics based on 50 GB Data

	Cassandra	MySQL
Average Write	0.12 ms	~300 ms
Average Read	15 ms	~350 ms



Cassandra data model



UserProfile

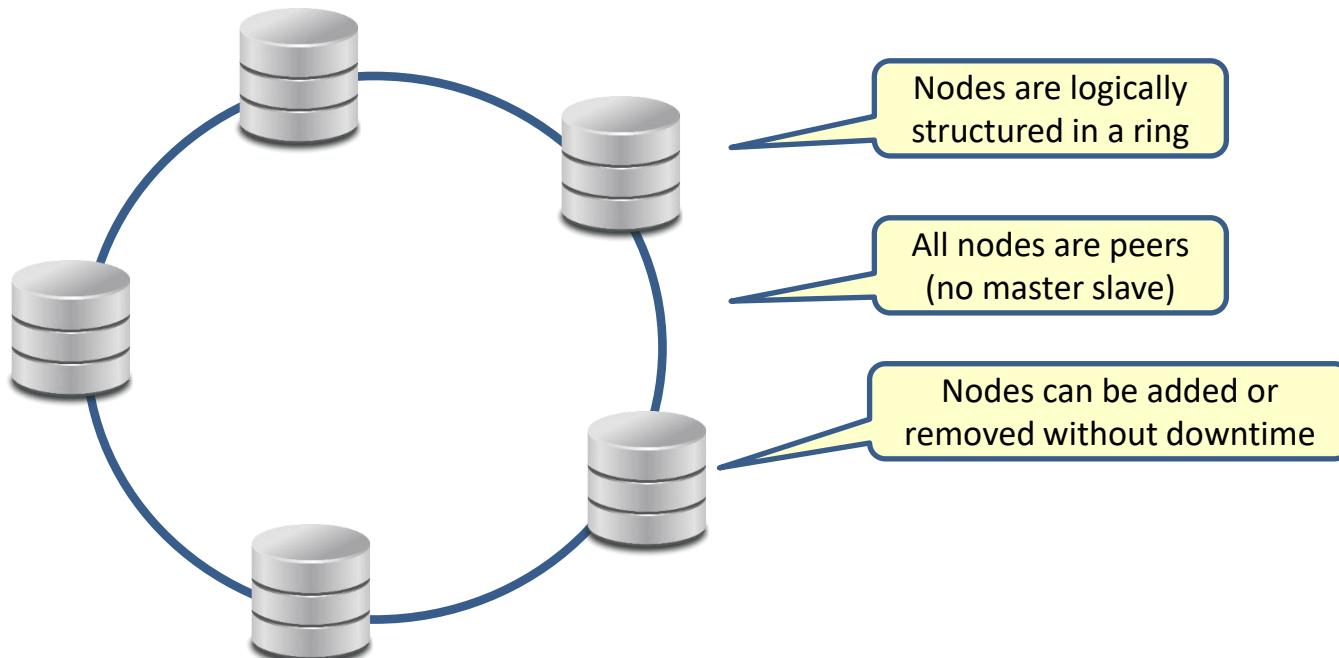
Bob	emailAddress	gender	age
	bob@example.com	male	35
	1465676582	1465676582	1465676582

Britney	emailAddress	gender
	brit@example.com	female
	1465676432	1465676432

Tori	emailAddress	country	hairColor
	tori@example.com	Sweden	Blue
	1435636158	1435636158	1465633654



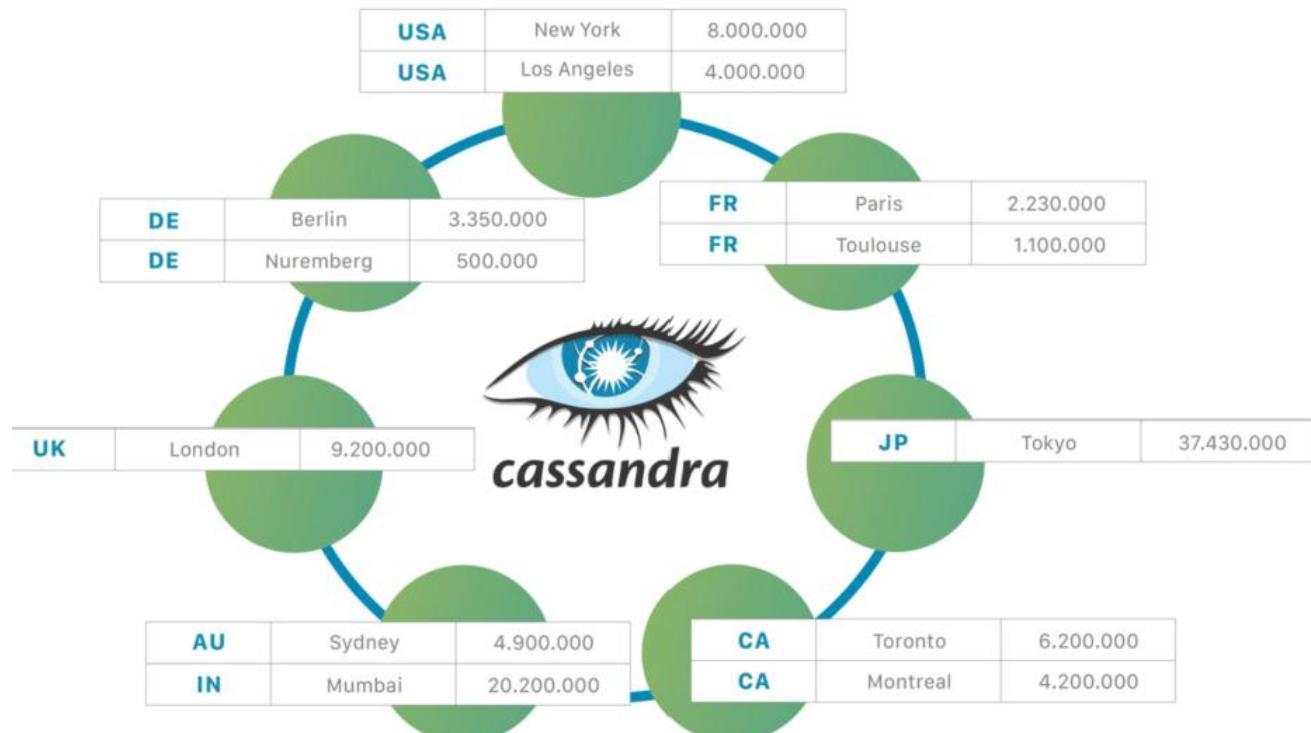
Cassandra cluster of nodes



Cassandra partitioning

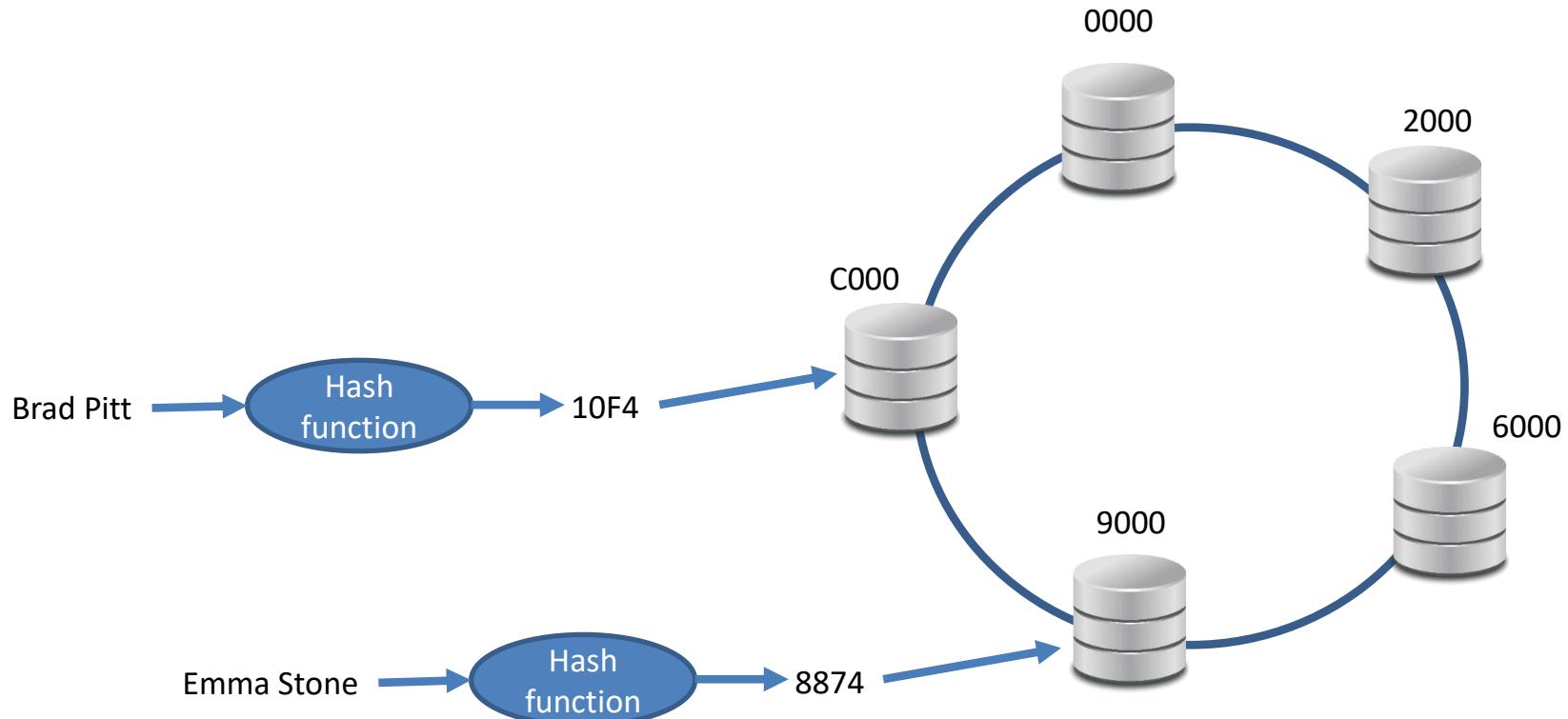
COUNTRY	CITY	POPULATION
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

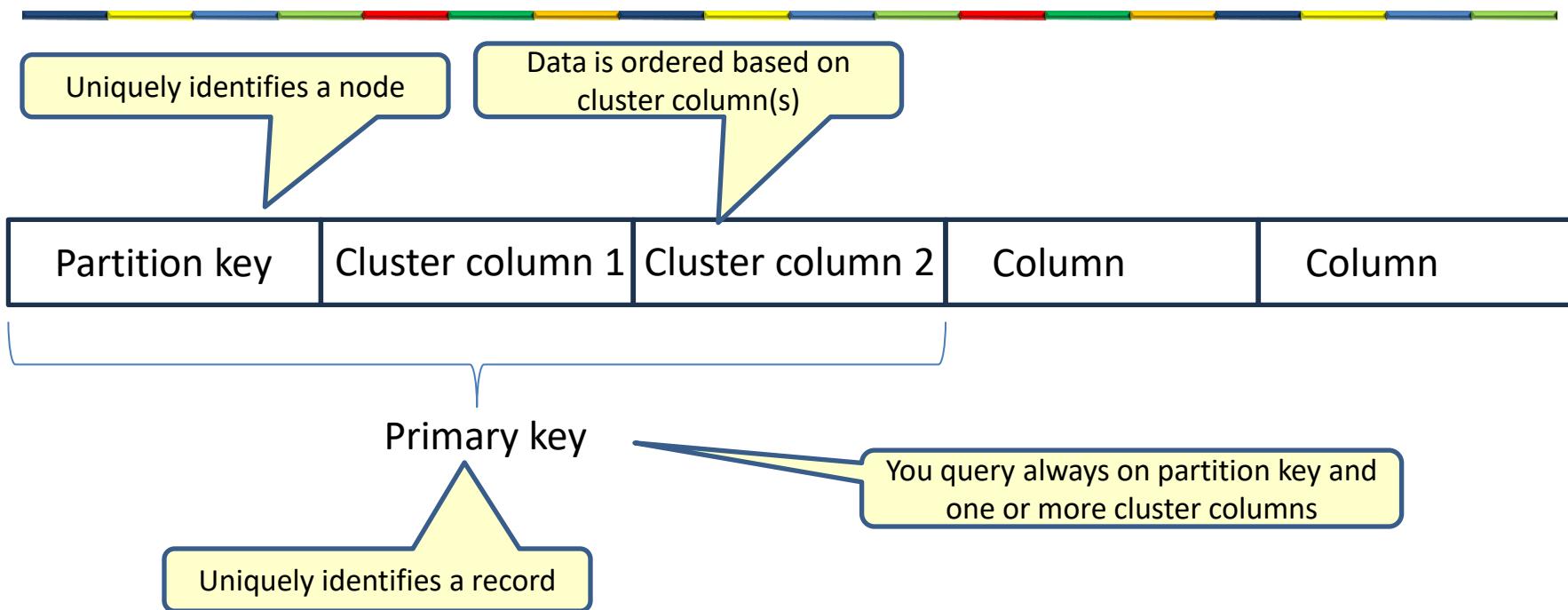


Partition key

- Decides where to place this record



Primary key



```
CREATE TABLE movies_by_actor (
    first_name TEXT,
    last_name TEXT,
    title TEXT,
    year INT,
    PRIMARY KEY ((first_name, last_name), title, year)
); ↵
```

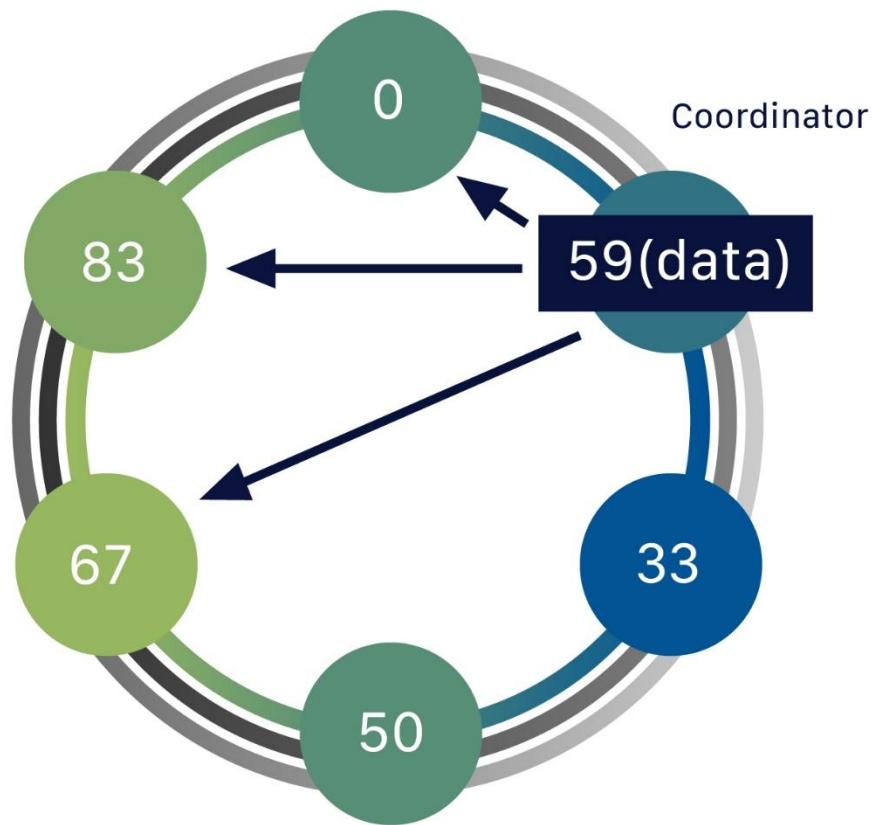
partition
key(s)

clustering
key(s)

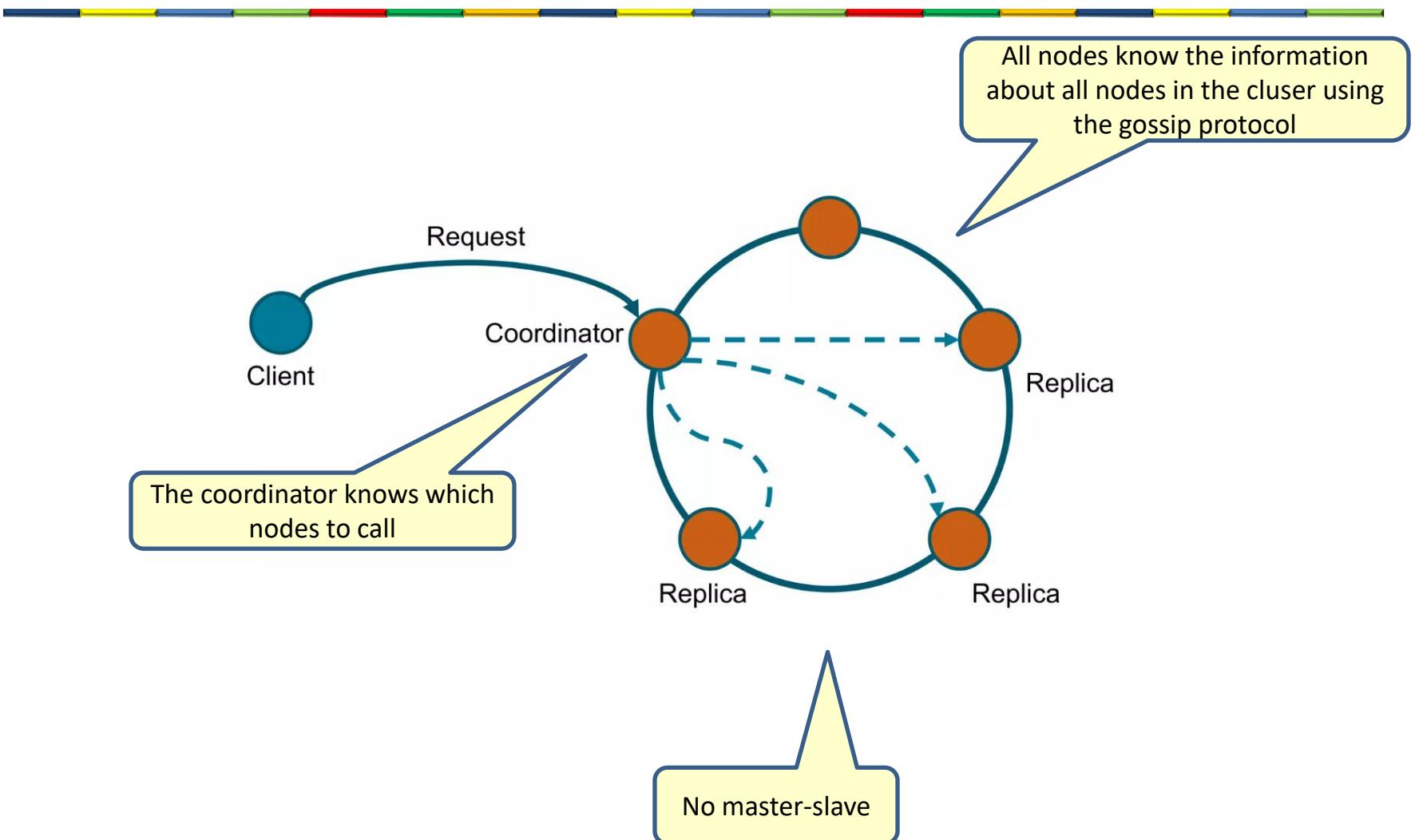
first_name	last_name	title	year
Johnny	Depp	Alice in Wonderland	2010
Johnny	Depp	Edward Scissorhands	1990
Anne	Hathaway	Alice in Wonderland	2010

Cassandra replication

RF = 3

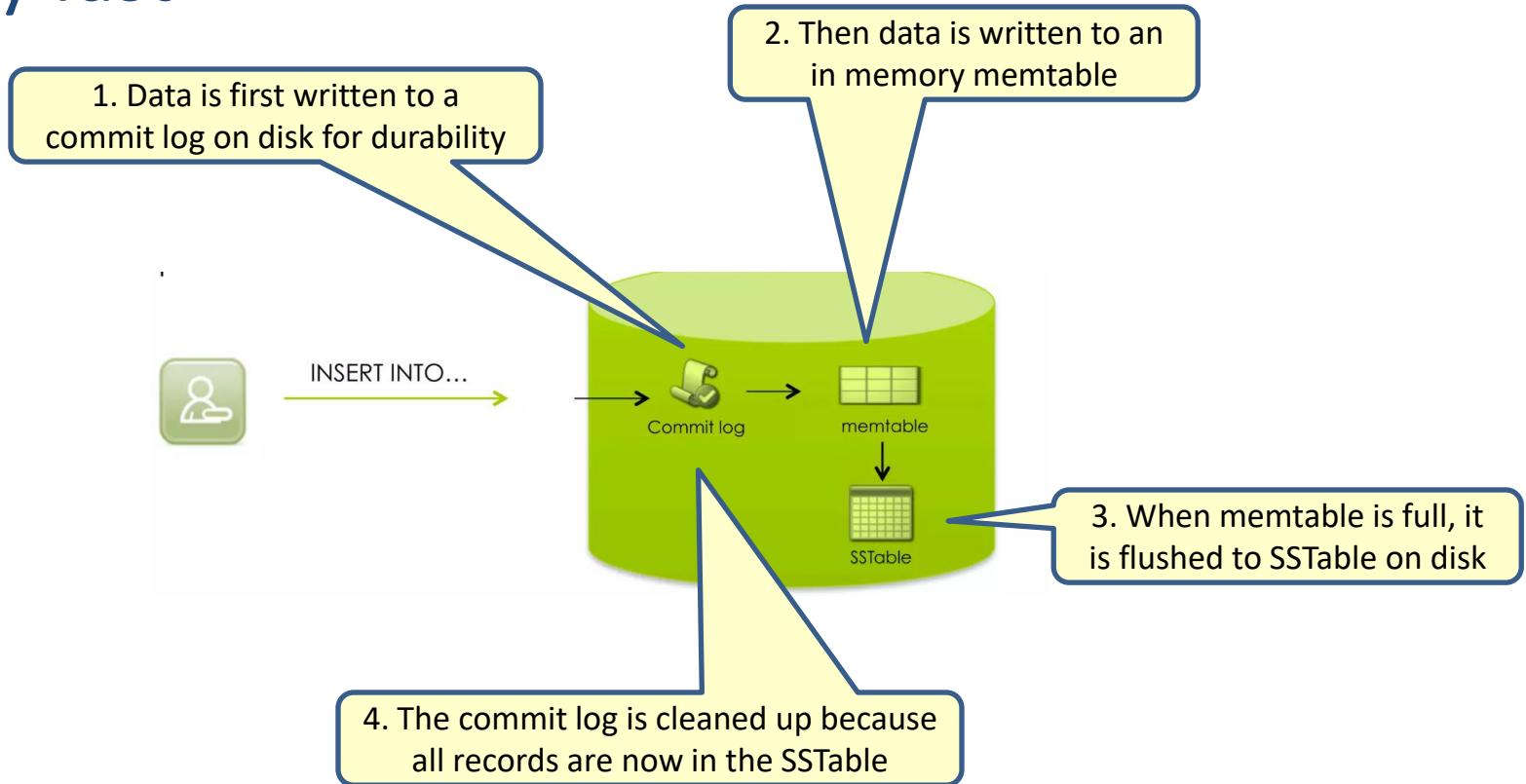


You can read and write to any node



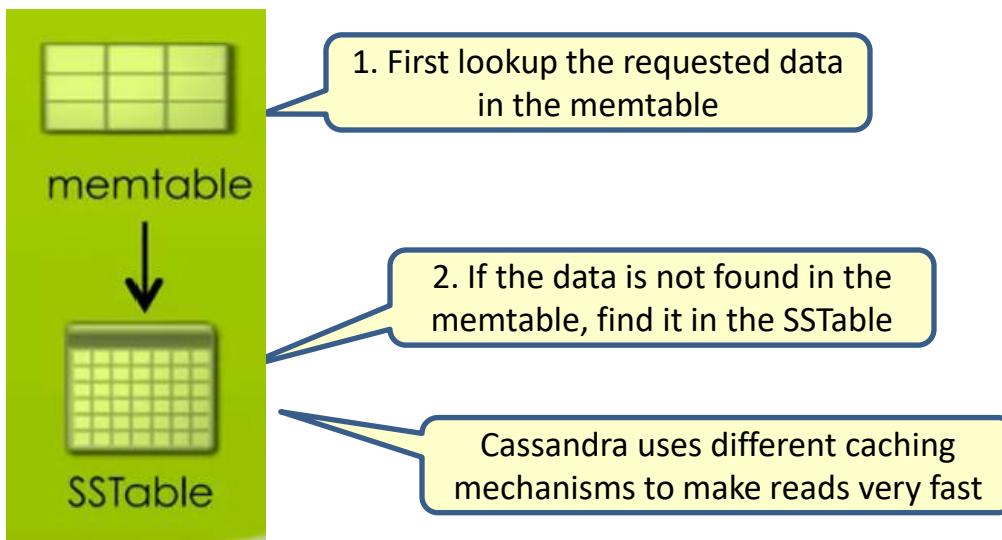
Writes in Cassandra

- Very fast



Reads in Cassandra

- Very fast

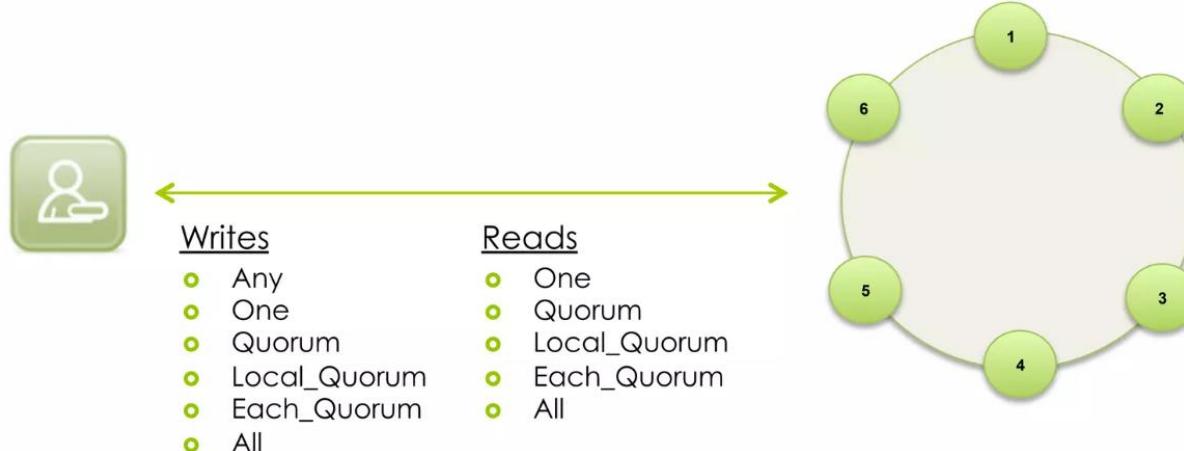


Tunable consistency

- Choose the balance between **data consistency** and **availability** for each read and write operation by setting a consistency level.
- **Write operations:** The client determines how many replicas must acknowledge the write before the operation is reported as successful.
- **Read operations:** The client specifies how many replicas must respond to a read request before data is returned to the client.
- **Key consistency levels**
 - **ONE:** The operation is successful if only one replica responds. This is the fastest but offers the lowest consistency.
 - **QUORUM:** A majority of replicas must respond. For a replication factor of 3, this means 2 out of 3 replicas must acknowledge the operation. This provides a balance between consistency and performance.
 - **ALL:** All replicas must respond. This provides the strongest consistency but is the slowest option.



Cassandra tunable consistency



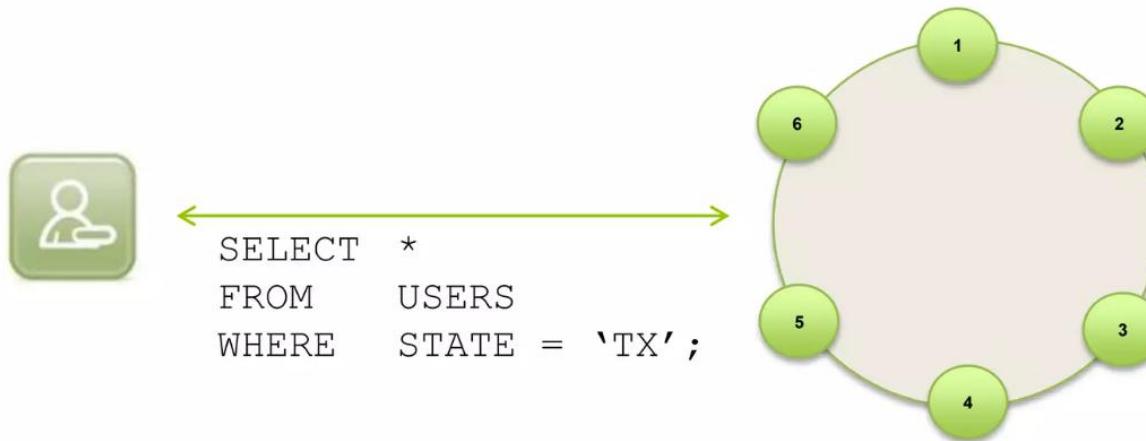
CQL Examples

```
SELECT total_purchases FROM SALES  
USING CONSISTENCY QUORUM  
WHERE customer_id = 5
```

```
UPDATE SALES  
USING CONSISTENCY ONE  
SET total_purchases = 500000  
WHERE customer_id = 4
```

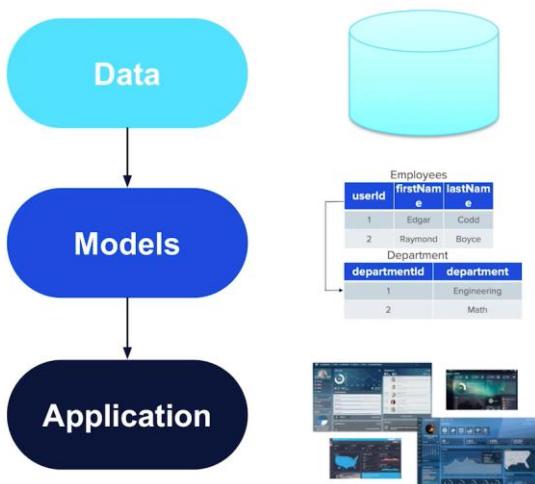
Cassandra Query Language (CQL)

- Very similar to SQL

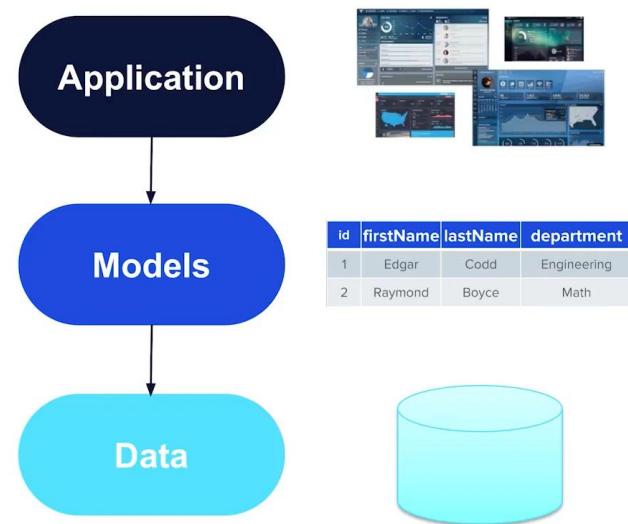


Data modeling

■ RDBMS



• Cassandra



Ratings by movie

title	year	email	rating
Alice in Wonderland	2010	jen@datastax.com	10
Alice in Wonderland	2010	joe@datastax.com	9
Alice in Wonderland	1951	jen@datastax.com	8
Edward Scissorhands	1990	joe@datastax.com	10

```
CREATE TABLE ratings_by_movie (
    title TEXT,
    year INT,
    email TEXT,
    rating INT,
    PRIMARY KEY ((title, year), email)
); ↵
```

```
INSERT INTO ratings_by_movie (title, year, email,
rating)
VALUES ('Alice in Wonderland', 2010,
'jen@datastax.com', 10);
INSERT INTO ratings_by_movie (title, year, email,
rating)
VALUES ('Alice in Wonderland', 2010,
'joe@datastax.com', 9);
INSERT INTO ratings_by_movie (title, year, email,
rating)
VALUES ('Alice in Wonderland', 1951,
'jen@datastax.com', 8);
INSERT INTO ratings_by_movie (title, year, email,
rating)
VALUES ('Edward Scissorhands', 1990,
'joe@datastax.com', 10); ↵
```

Actors by movie



title	year	first_name	last_name
Alice in Wonderland	2010	Anne	Hathaway
Alice in Wonderland	2010	Johnny	Depp
Edward Scissorhands	1990	Johnny	Depp

```
CREATE TABLE actors_by_movie (
    title TEXT,
    year INT,
    first_name TEXT,
    last_name TEXT,
    PRIMARY KEY ((title, year), first_name, last_name)
); ↵
```



Movies by actor

first_name	last_name	title	year
Johnny	Depp	Alice in Wonderland	2010
Johnny	Depp	Edward Scissorhands	1990
Anne	Hathaway	Alice in Wonderland	2010

```
CREATE TABLE movies_by_actor (
    first_name TEXT,
    last_name TEXT,
    title TEXT,
    year INT,
    PRIMARY KEY ((first_name, last_name), title, year)
); ↵
```

Movies by user

email	watched_on	title	year
joe@datastax.com	2020-04-28	Edward Scissorhands	1990
joe@datastax.com	2020-03-08	Alice in Wonderland	2010
joe@datastax.com	2020-02-13	Toy Story 3	2010
joe@datastax.com	2020-01-22	Despicable Me	2010
joe@datastax.com	2019-12-30	Alice in Wonderland	1951
jen@datastax.com	2011-10-01	Alice in Wonderland	2010

```
CREATE TABLE movies_by_user (
    email TEXT,
    title TEXT,
    year INT,
    watched_on DATE,
    PRIMARY KEY ((email), watched_on, title, year)
) WITH CLUSTERING ORDER BY (watched_on DESC); ↵
```



Data duplication



email	title	year	rating
joe@datastax.com	Alice in Wonderland	2010	9
joe@datastax.com	Edward Scissorhands	1990	10
jen@datastax.com	Alice in Wonderland	1951	8
jen@datastax.com	Alice in Wonderland	2010	10

title	year	first_name	last_name
Alice in Wonderland	2010	Anne	Hathaway
Alice in Wonderland	2010	Johnny	Depp
Edward Scissorhands	1990	Johnny	Depp

title	year	email	rating
Alice in Wonderland	2010	jen@datastax.com	10
Alice in Wonderland	2010	joe@datastax.com	9
Alice in Wonderland	1951	jen@datastax.com	8
Edward Scissorhands	1990	joe@datastax.com	10

email	watched_on	title	year
joe@datastax.com	2020-04-28	Edward Scissorhands	1990
joe@datastax.com	2020-03-08	Alice in Wonderland	2010
joe@datastax.com	2020-02-13	Toy Story 3	2010
joe@datastax.com	2020-01-22	Despicable Me	2010
joe@datastax.com	2019-12-30	Alice in Wonderland	1951
jen@datastax.com	2011-10-01	Alice in Wonderland	2010



Cassandra advantages & disadvantages

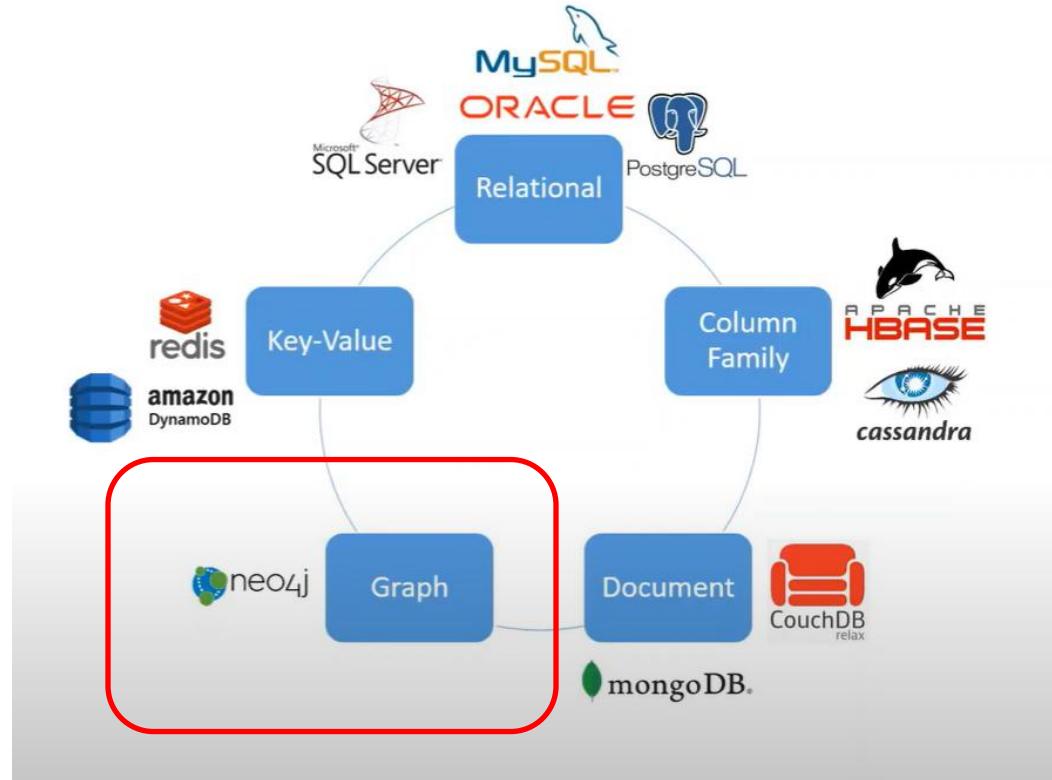
- Advantages
 - Very fast for writes and reads
 - Linear scalability
 - You can write to any node to geographically distributed clusters
 - Tunable consistency
- Disadvantages
 - No joins and aggregates
 - No transactions
 - No strict consistency
 - Data duplication
 - Slow updates and deletes
 - Only structured data



Cassandra use cases

- Real-time analytics pipeline
 - Large data sets, only write and read
- IoT data management
 - Large data sets, only write and read
- Content management systems
 - High write volumes, ensure data availability, and maintain consistency
- Real-time analysis of large datasets
 - Fraud detection and risk management
- Time-series data analysis
 - Large data sets, only write and read





GRAPH DATABASE



NEO4J



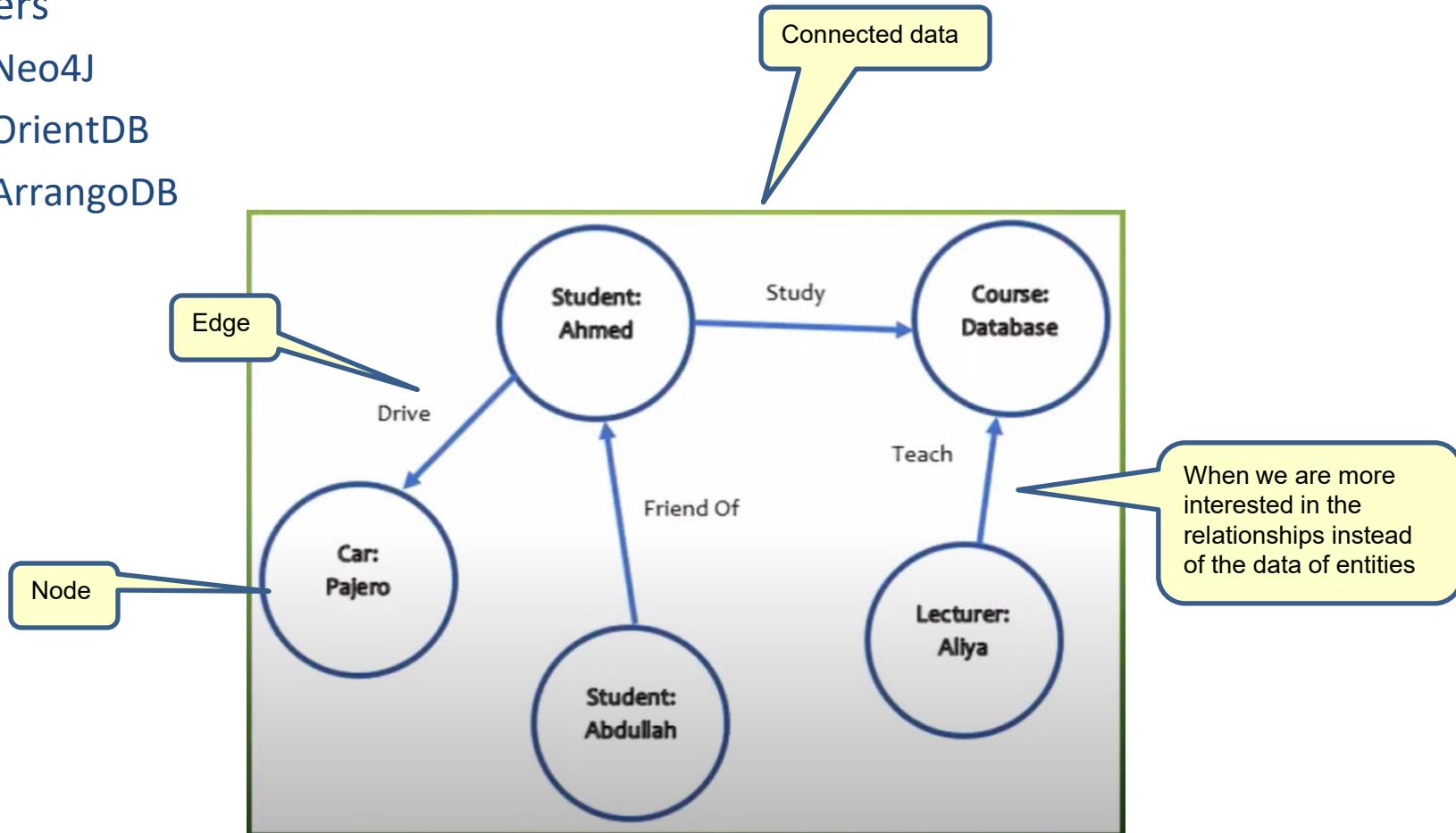
Graph database

- Store data in nodes and relationships
 - No Joins
- Very fast in analyzing data with lots of relationships
- No fixed schema
 - Easy to evolve your dataset



Graph database

- Players
 - Neo4J
 - OrientDB
 - ArangoDB

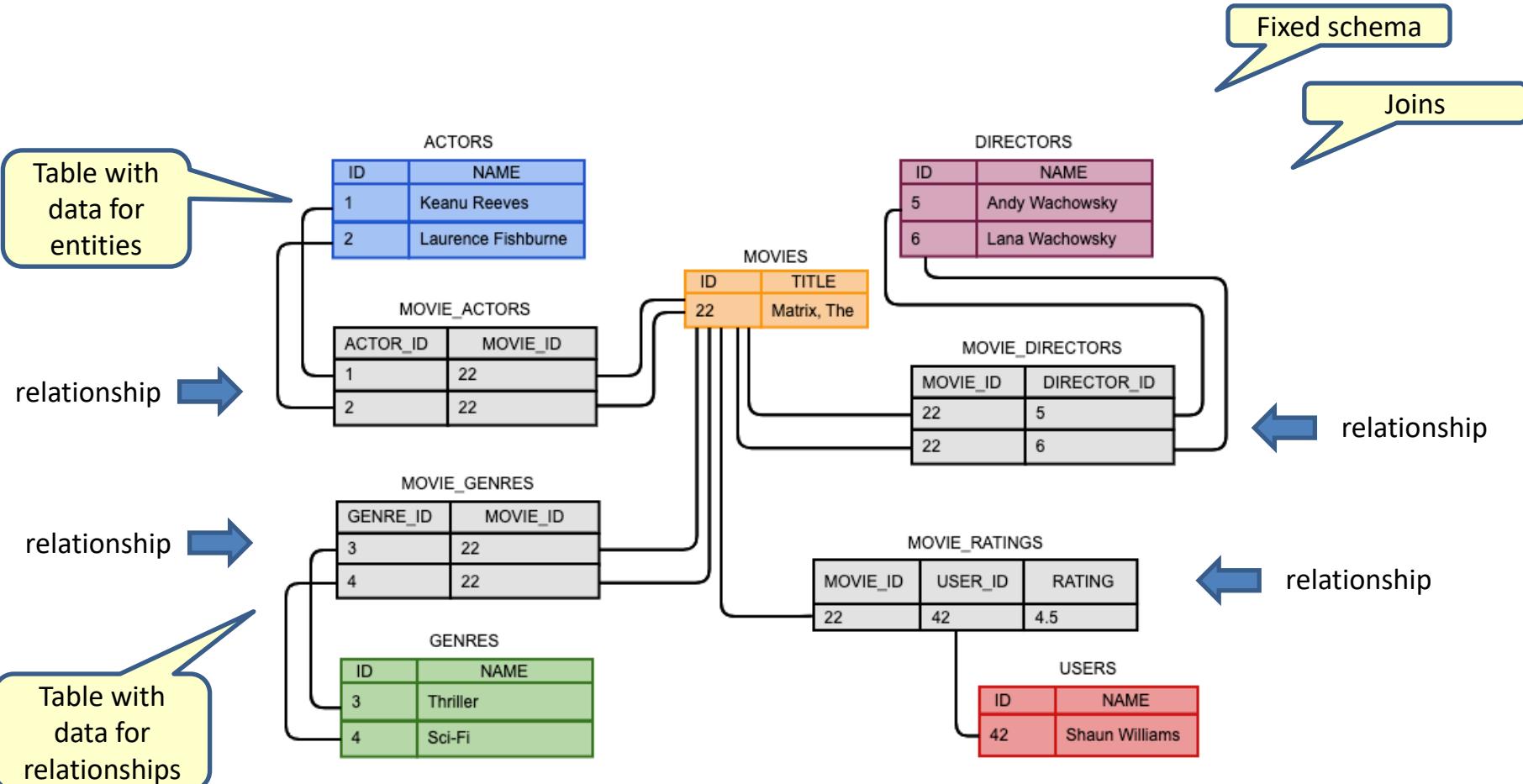


Graph DB use cases

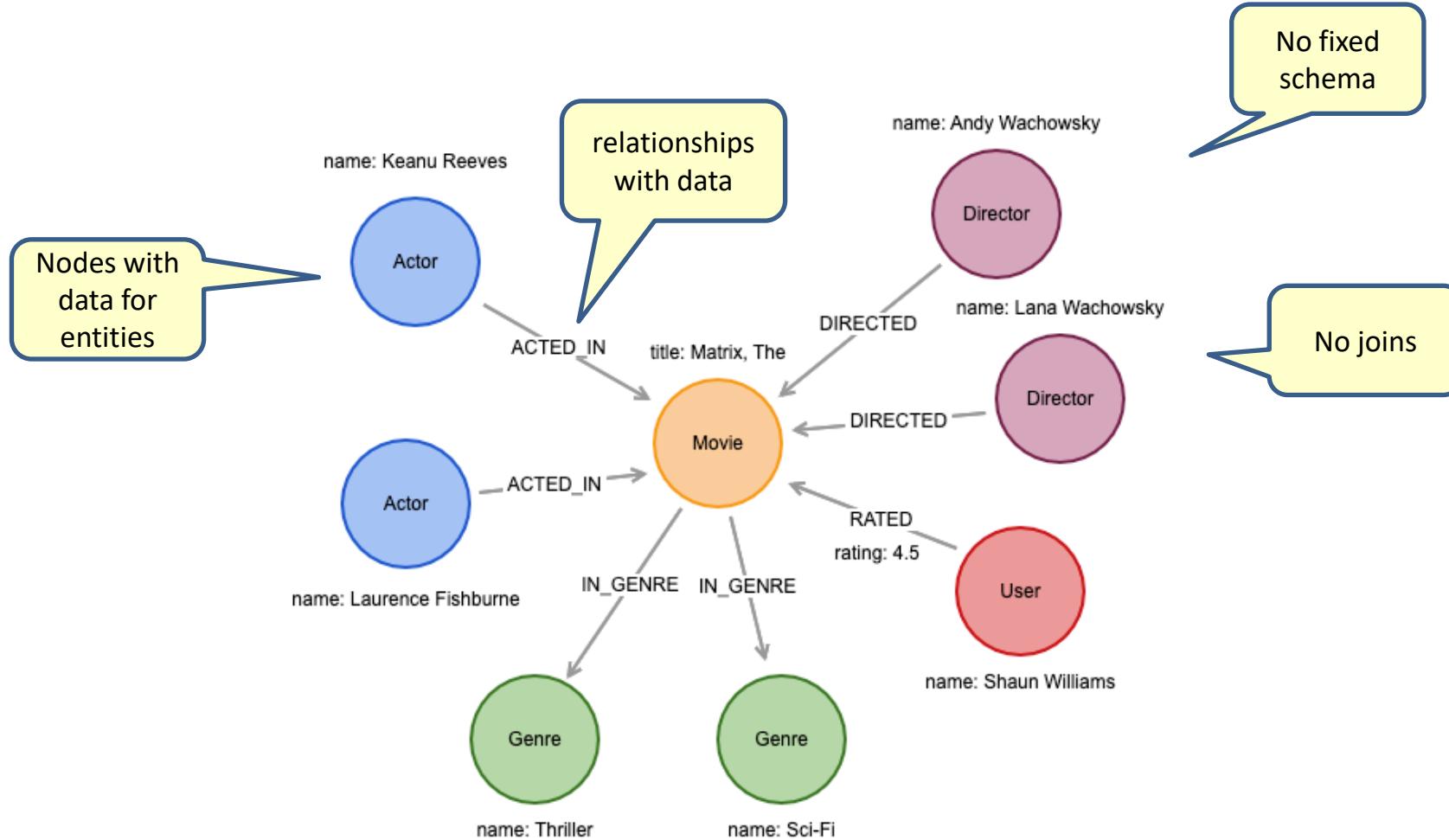
- Social networks
- Knowledge graphs
- Fraud detection
- Contact tracing
- Real time recommendation
- Graph based search
- Network & IT operations
- Identity and access management



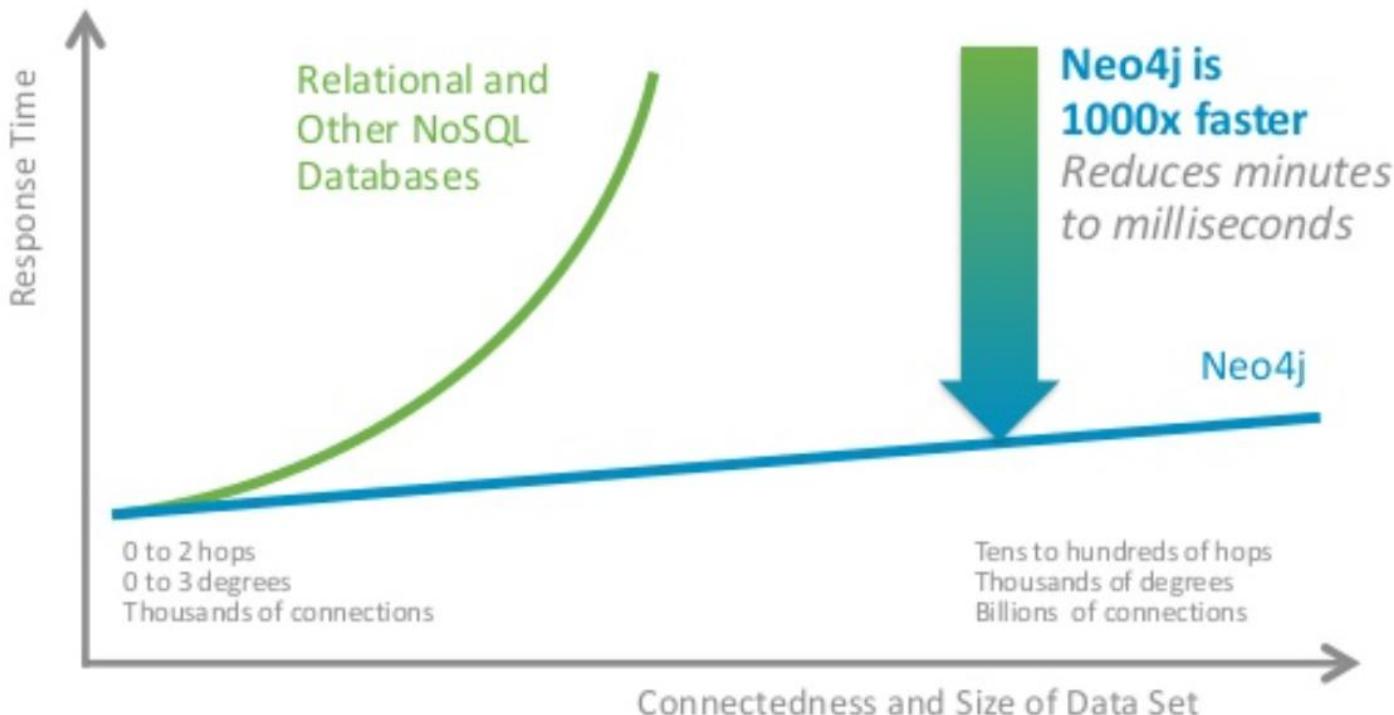
Relational database



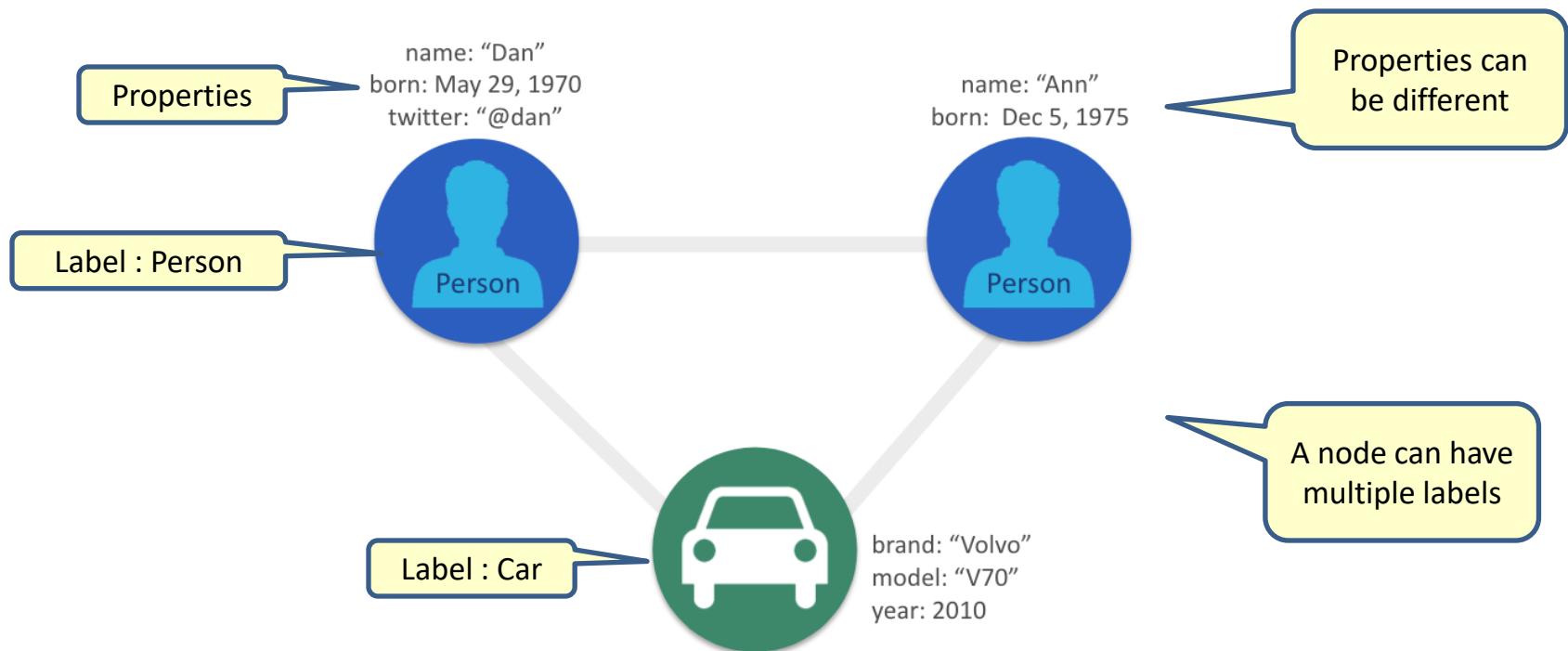
Graph database: NEO4J



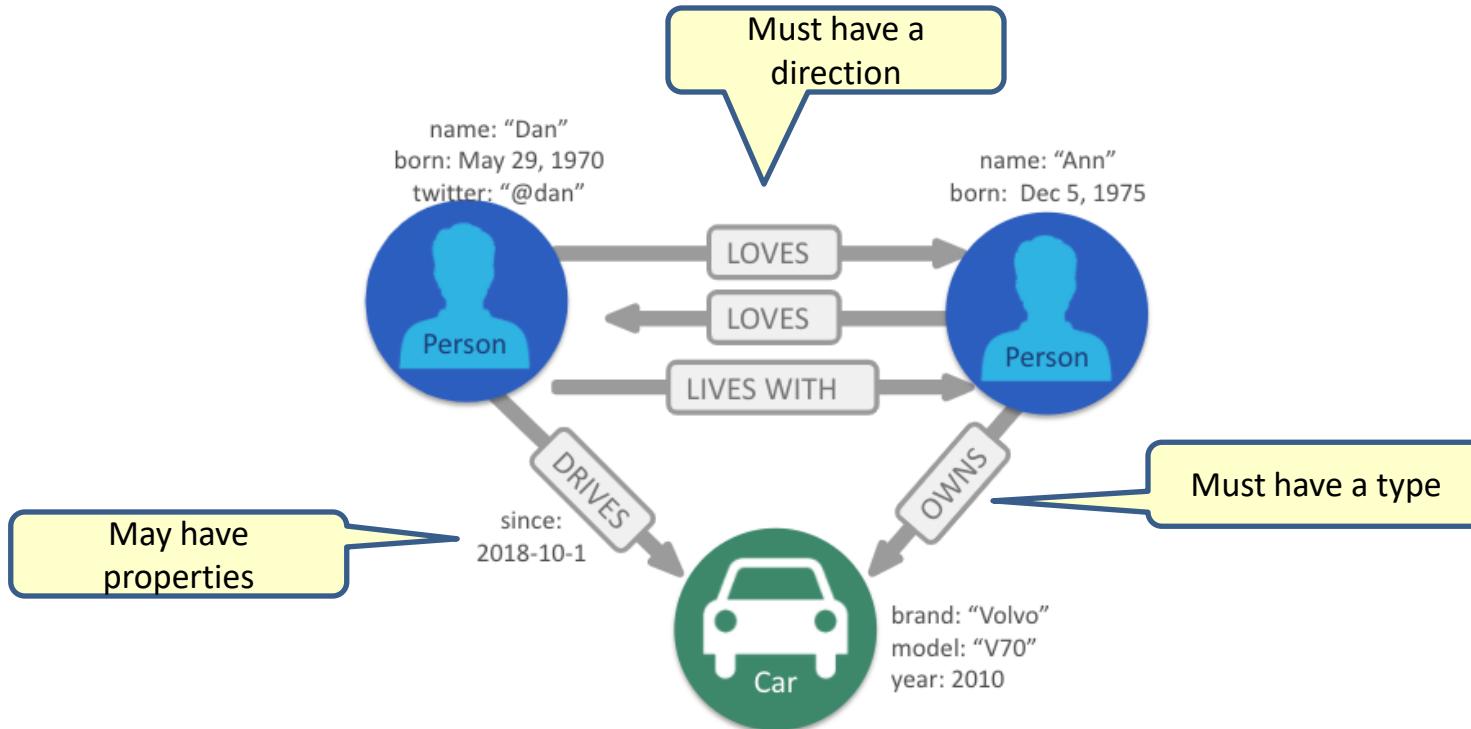
Query performance



Nodes

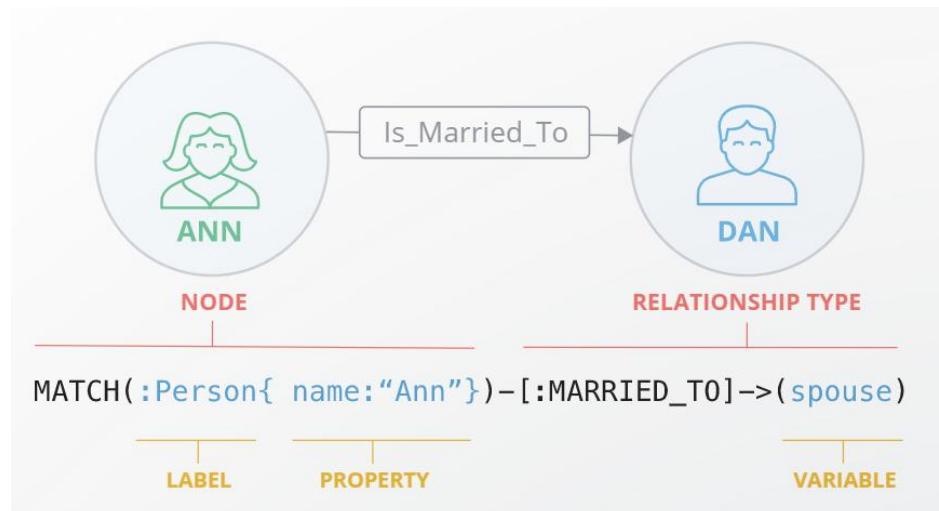


Relationships



Cypher query language

- Declarative
 - Describe what you want
 - Not how to do it



Create a node



```
CREATE (:Movie {title: 'Batman Begins'})
```

```
neo4j$ MATCH (n) RETURN n
```

n
1
{ "identity": 0, "labels": ["Movie"], "properties": { "title": "Batman Begins" } }

```
neo4j$ MATCH (n) RETURN n
```

*	(1)	Movie(1)
Graph		
Table		
A		Batman Begins
Text		
Code		



Create a relationship

```
MATCH (a:Person), (m:Movie)
WHERE a.name = 'Michael Caine' AND m.title = 'Batman Begins'
CREATE (a)-[:ACTED_IN]->(m)
```



```
MATCH (a:Person), (m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
CREATE (a)-[:ACTED_IN {roles: ['Bruce Wayne', 'Batman']}]->(m)
RETURN a, m
```



Queries on nodes

```
MATCH (n)  
RETURN n
```

Retrieve all nodes

```
MATCH (p:Person)  
RETURN p
```

Retrieve all Person nodes

```
MATCH (p:Person {born: 1970})  
RETURN p
```

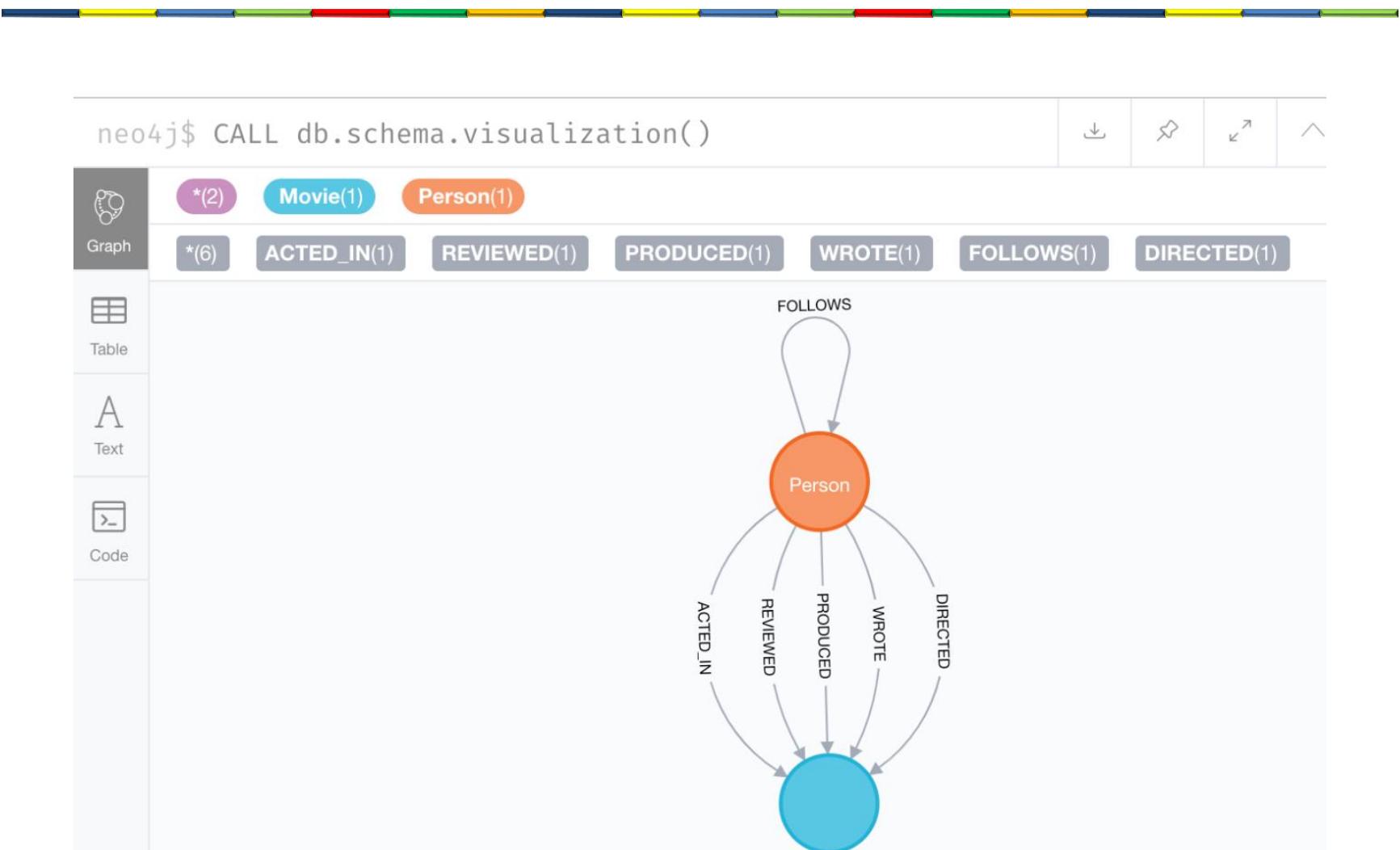
Retrieve all Person nodes born in 1970

```
MATCH (m:Movie {released: 2003, tagline: 'Free your mind'})  
RETURN m
```

Retrieve all Movie nodes released in 2003
with the tagline ‘free your mind’

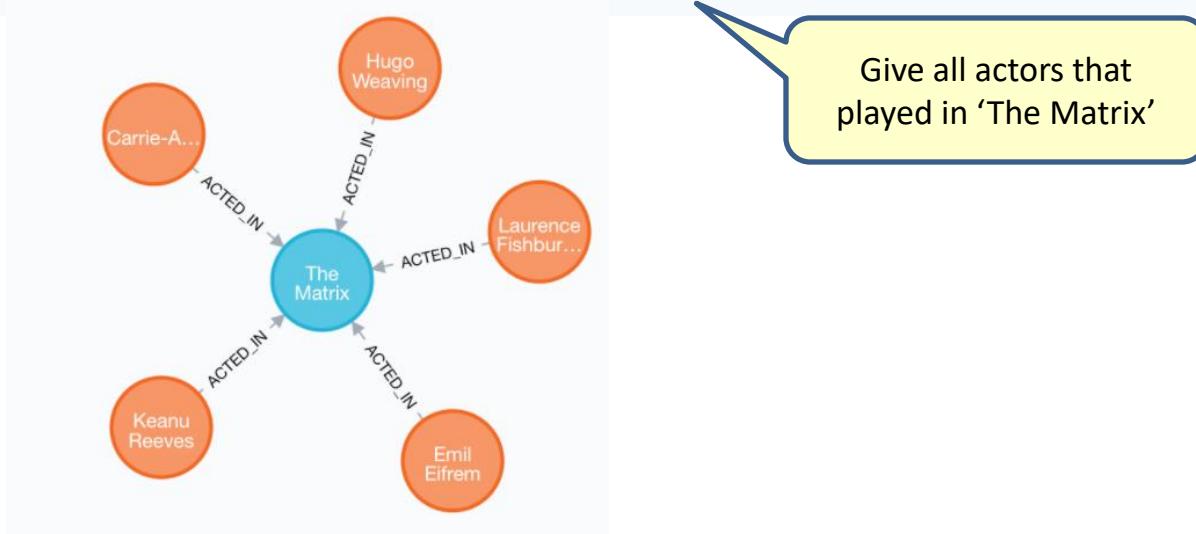


Visualize the schema



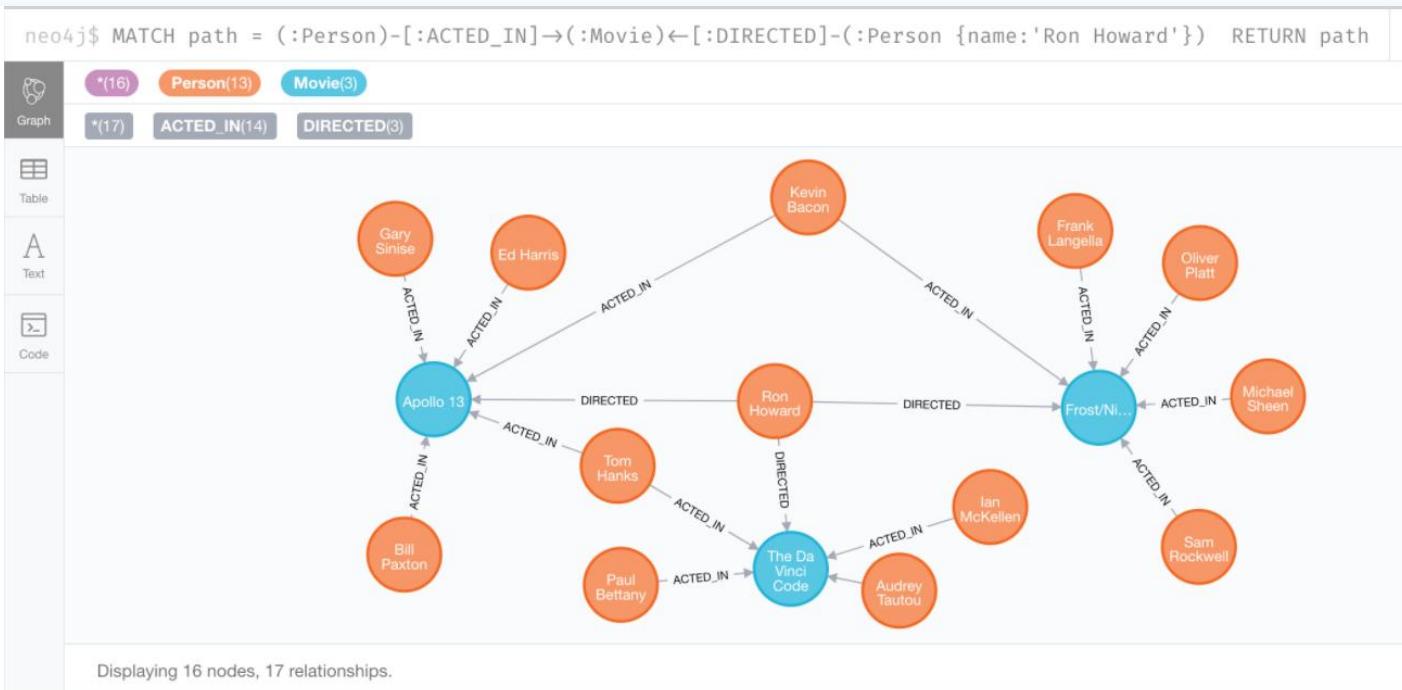
Queries on relationships

```
MATCH (p:Person)-[rel:ACTED_IN]->(m:Movie {title: 'The Matrix'})  
RETURN p, rel, m
```

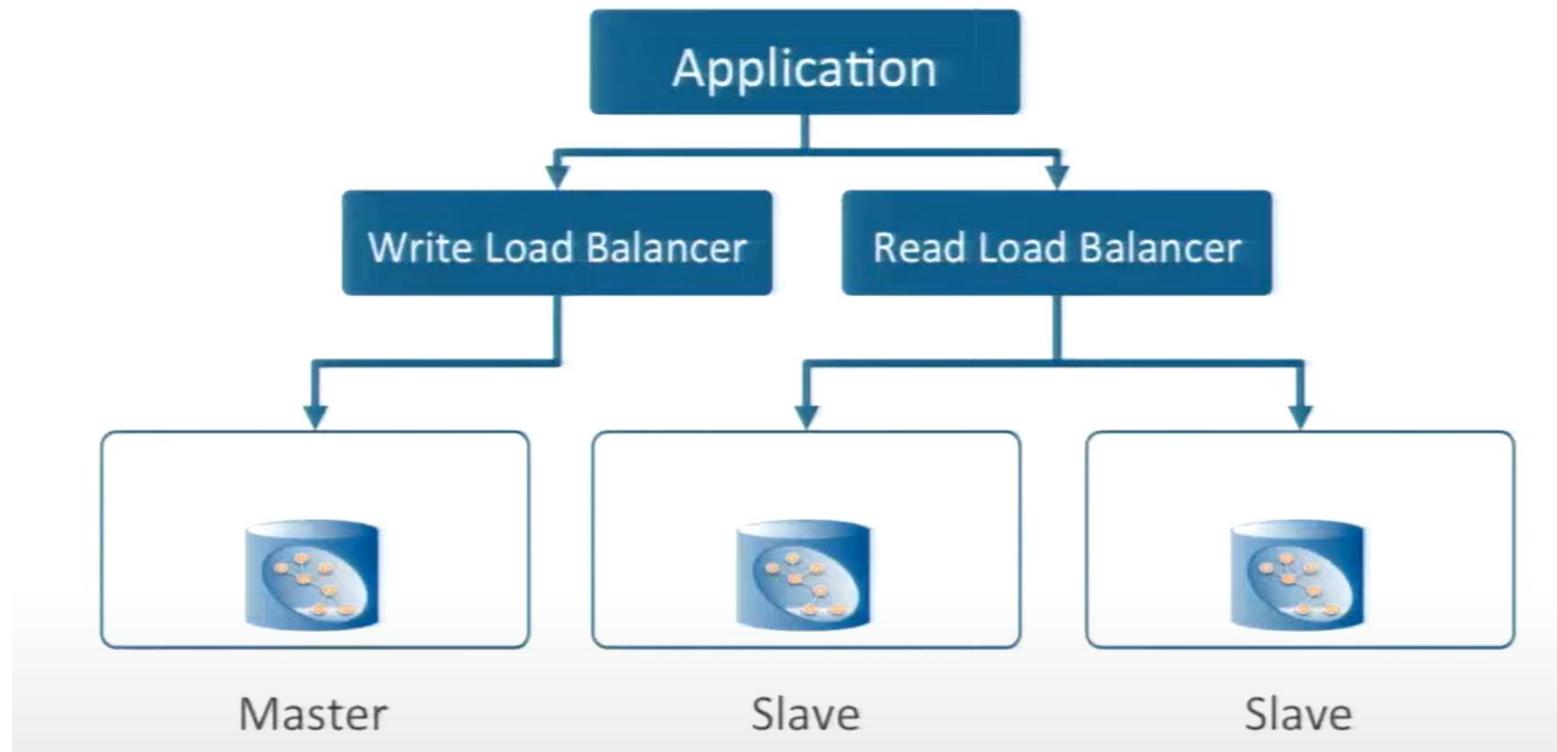


Return multiple paths

```
MATCH path = (:Person)-[:ACTED_IN]->(:Movie)<-[ :DIRECTED] -(:Person {name:'Ron Howard'})  
RETURN path
```



Neo4j High Availability cluster



Neo4j advantages & disadvantages

- Advantages
 - Very fast in querying deeply connected data
 - Database model and domain model are similar
 - Powerful and simple query language
 - Easy to add new nodes, attributes and relationships
 - Cluster (scalability, failover and high availability)
- Disadvantages
 - Not for unconnected data
 - Not for unstructured data



SPRING BOOT NEO4J EXAMPLE



Domain classes + repository

```
@Node  
public class Person {  
  
    @Id @GeneratedValue private Long id;  
  
    private String name;  
  
    @Relationship(type = "TEAMMATE")  
    public Set<Person> teammates= new HashSet<>();
```

```
public interface PersonRepository extends Neo4jRepository<Person, Long> {  
  
    Person findByName(String name);  
    List<Person> findByTeammatesName(String name);  
}
```



Application(1/2)

```
@SpringBootApplication
@EnableNeo4jRepositories
public class PersonApplication implements CommandLineRunner{

    @Autowired
    PersonRepository personRepository;

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



Application(2/2)

```
@Override
public void run(String... args) throws Exception {
    personRepository.deleteAll();

    Person greg = new Person("Greg");
    Person roy = new Person("Roy");
    Person bob = new Person("Bob");
    greg.worksWith(roy);
    greg.worksWith(bob);
    roy.worksWith(bob);

    List<Person> team = Arrays.asList(greg, roy, bob);

    personRepository.save(greg);
    personRepository.save(roy);
    personRepository.save(bob);

    System.out.println("Lookup each person by name...");
    team.stream().forEach(person -> System.out.println(
        personRepository.findByName(person.getName())));
}

List<Person> teammates = personRepository.findByTeammatesName(bob.getName());
System.out.println("The following have Bob as a teammate...");
teammates.stream().forEach(person -> System.out.println( person.getName()));

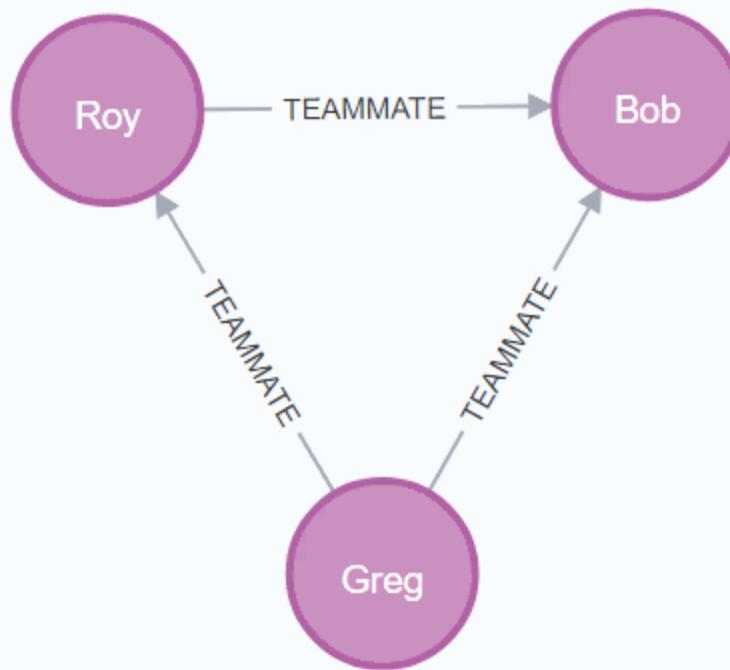
}
```

application.properties

```
spring.neo4j.uri=bolt://localhost:7687  
spring.data.neo4j.username=neo4j  
spring.data.neo4j.password=admin
```



Nodes



SUMMARY

Five different types of databases

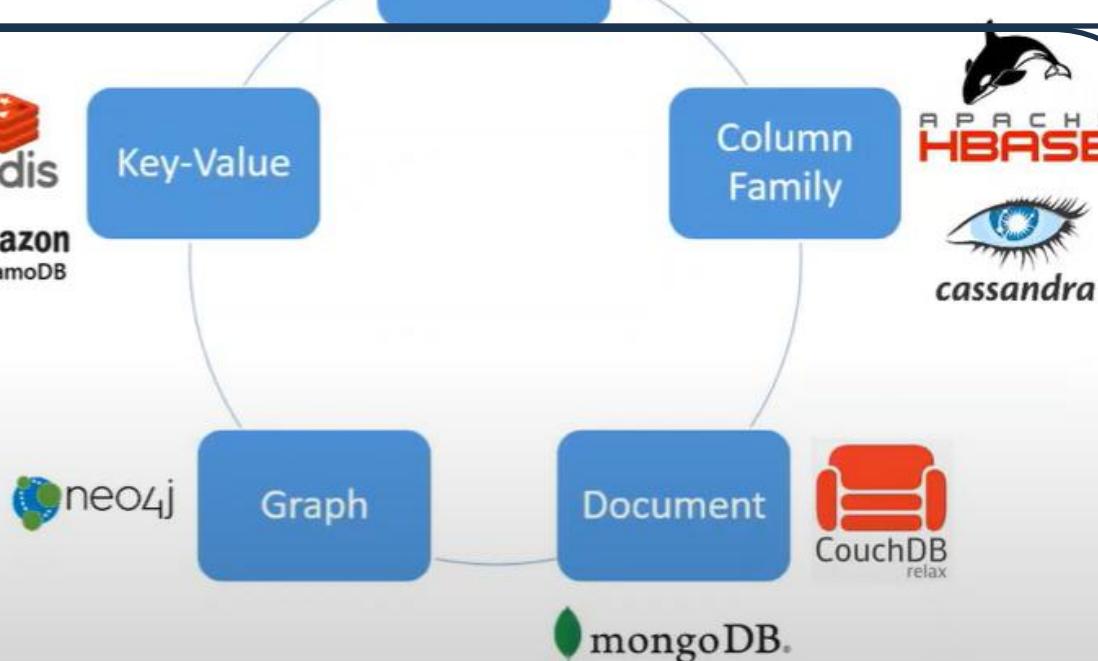
Relational



Non-relational



NoSQL
(Not only SQL)



Relational database

- Does not scale well
- Hard to change
- Does not handle unstructured and semi-structured data



NoSQL databases

- Scales very well
- Easy to change, no fixed schema
- Handles unstructured and semi structured data



Comparing databases

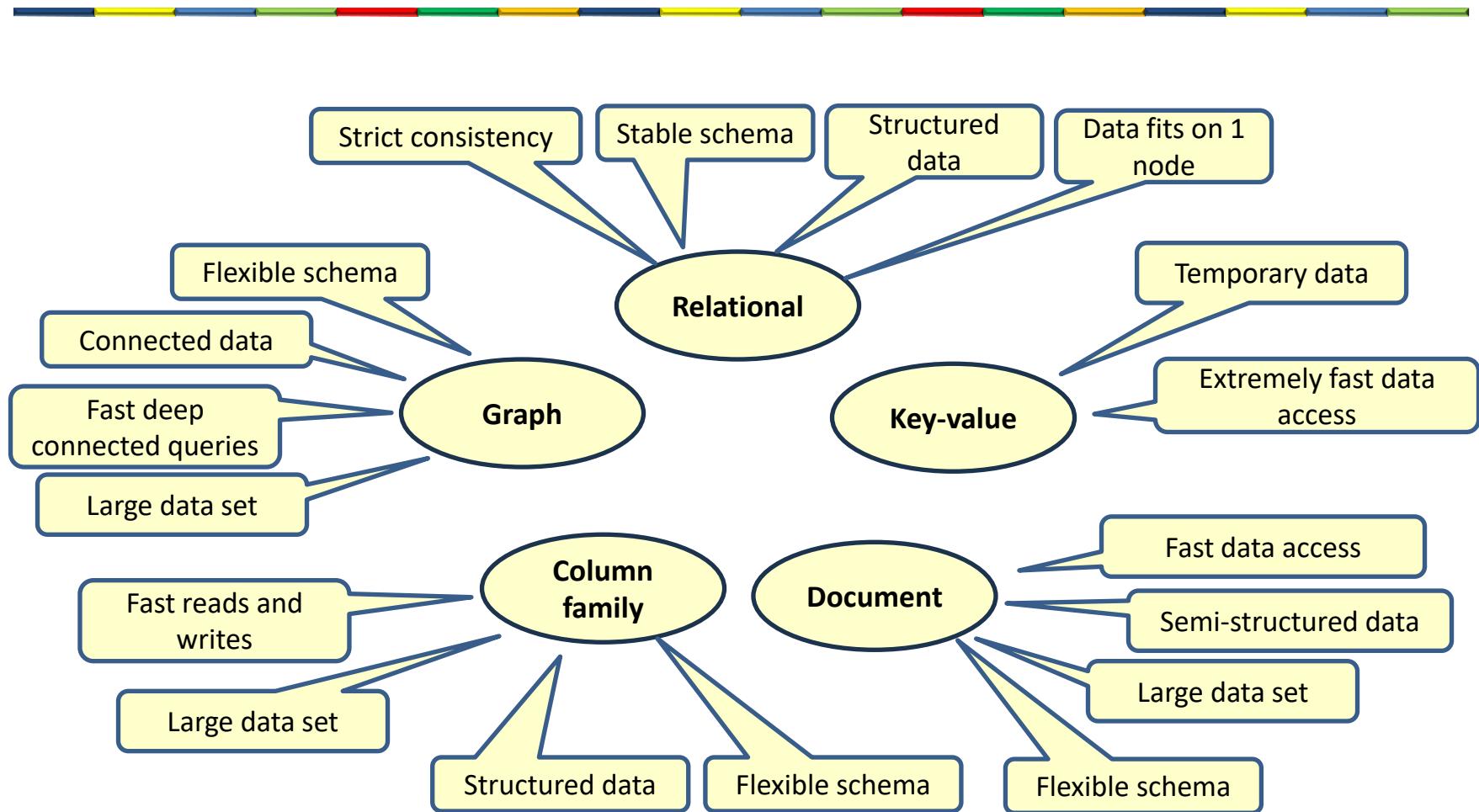
		Strong points	Weak points
Relational	Structured data Wide adaption Strict consistency	Hard to scale Hard to change the schema Not good in unstructured data	
Key-value	Extremely fast in-memory data	Not for permanent data storage No or limited query functionality	
Document	Whole documents Fast reads and writes Unstructured data	Data duplication	
Wide column	Very fast reads and writes Fast queries at scale Linear scalable	Data duplication Slow updates and deletes Not good in unstructured data	
Graph	Connected data Fast queries at scale Flexible data structure	Not good for unconnected data Not good in unstructured data	

Transactions

- Oracle:
 - One node: ACID
 - HA cluster with sharding: ACID
- Mongo, Neo4J:
 - One node: ACID
 - HA cluster with sharding: BASE
- Cassandra:
 - One node: Tunable consistency
 - HA cluster with sharding: Tunable consistency



Which database to use?



Key principle 6

- When your data does not fit on one node you automatically will get
 - Data duplication
- Data duplication over multiple nodes means eventual consistency



Lesson 4

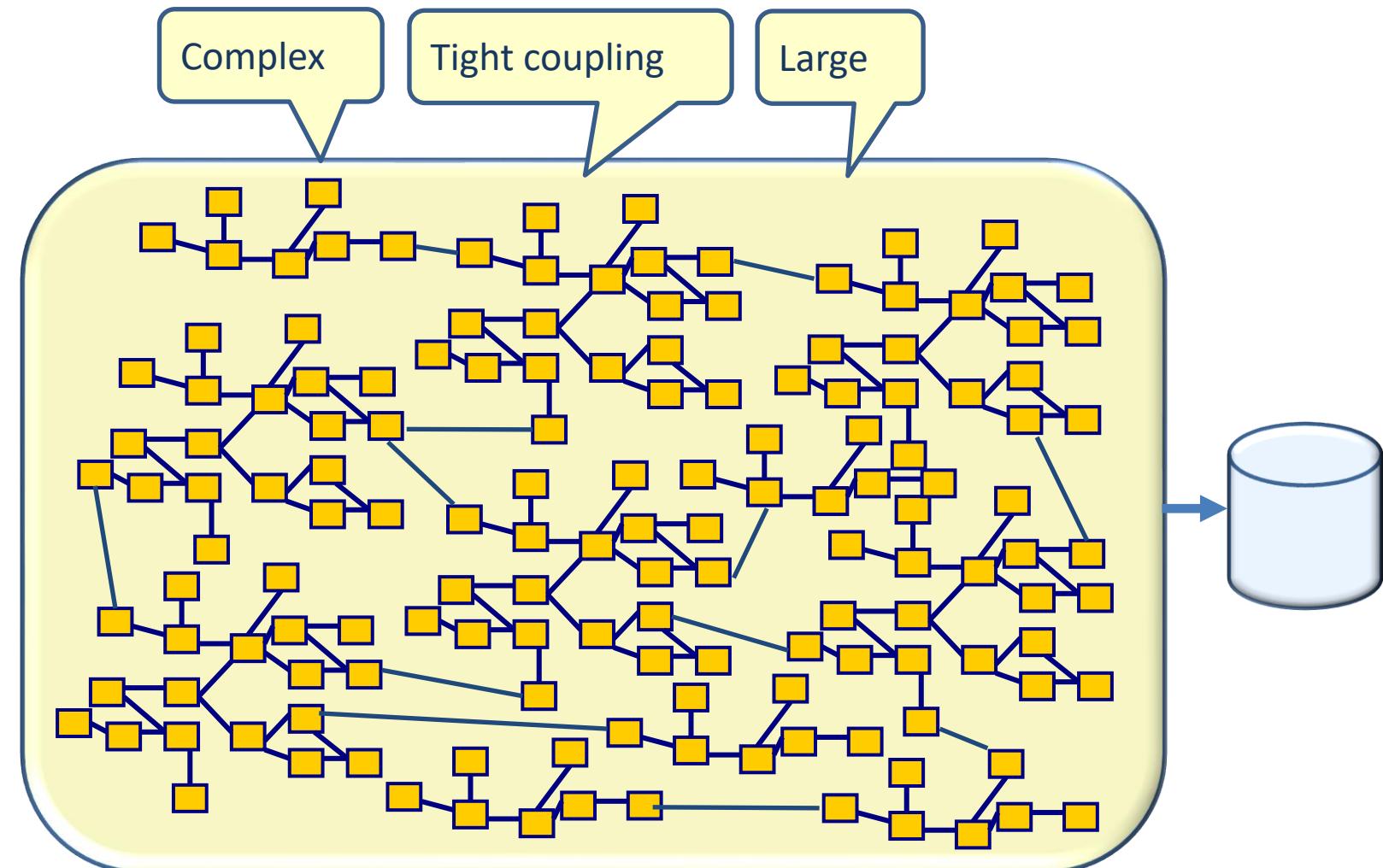
COMPONENT BASED DESIGN



PROBLEM OF 1 LARGE OO APPLICATION

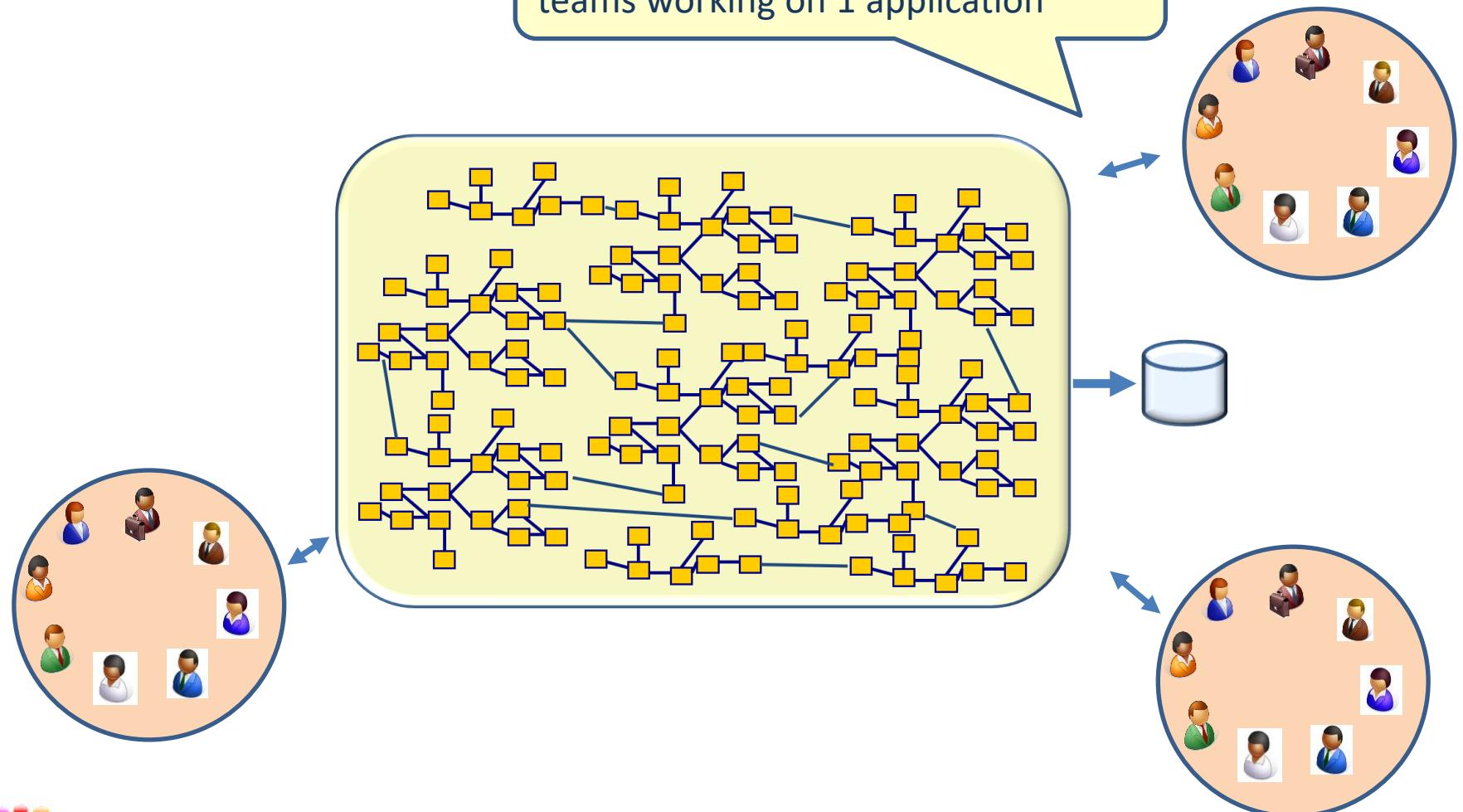


Object orientation

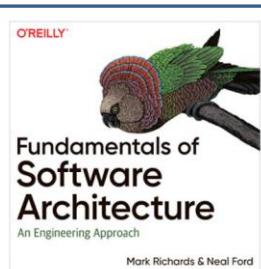


1 large monolith application

Very difficult to have multiple scrum teams working on 1 application



Amazon.com



Fundamentals of Software Architecture: An Engineering Approach Audible Audiobook – Unabridged

Mark Richards (Author), Neal Ford (Author), Benjamin Lange (Narrator), Upfront Books (Publisher)

4.5 out of 5 stars 251 ratings

See all formats and editions

Kindle
\$40.51

Audiobook
\$0.00

Paperback
\$53.41

Read with Our Free App

Free with your Audible trial

16 Used from \$44.10
24 New from \$42.56

Salary surveys worldwide regularly place software architect in the top 10 best jobs, yet no real guide exists to help developers become architects. Until now. This book provides the first comprehensive overview of software architecture's many aspects. Aspiring and existing architects alike will examine architectural characteristics, architectural patterns, component determination, diagramming and presenting architecture, evolutionary architecture, and many other topics.

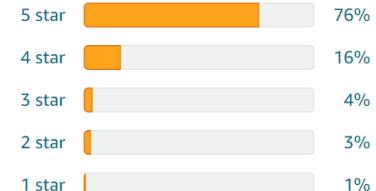
Mark Richards and Neal Ford – hands-on practitioners who have taught software architecture classes professionally for years – focus on
[Read more](#)

Audible Sample

Customer reviews

4.6 out of 5

251 global ratings



[How are ratings calculated?](#)

People who viewed this also viewed



Software Engineering at Google: Lessons Learned from Programming One Thing
Titus Winters
★★★★★ 284
Audible Audiobook
\$0.00 Free with Audible trial



Monolith to Microservices: Evolutionary Patterns...
Sam Newman
★★★★★ 293
Audible Audiobook
\$0.00 Free with Audible trial



Clean Architecture: A Craftsman's Guide to Software Structure & Design
Robert C. Martin
★★★★★ 1443
#1 Best Seller in Industrial Quality Control
Audible Audiobook
\$0.00 Free with Audible trial



Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems
Martin Kleppmann
★★★★★ 1822
#1 Best Seller in Enterprise Data Computing
Audible Audiobook
\$0.00 Free with Audible trial



Clean Agile: Back to Basics
Robert C. Martin
★★★★★ 258
#1 Best Seller in Enterprise Data Computing
Audible Audiobook
\$0.00 Free with Audible trial



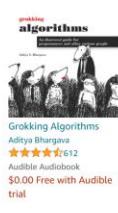
The Clean Coder: A Code of Conduct for Professional Programmers
Robert C. Martin
★★★★★ 844
Audible Audiobook
\$0.00 Free with Audible trial



Clean Code: A Handbook of Agile Software Craftsmanship
Robert C. Martin
★★★★★ 1200
Audible Audiobook
\$0.00 Free with Audible trial

Page 1 of 8

People who bought this also bought



Groking Algorithms: An illustrated guide for programmers and other curious people
Aditya Bhargava
★★★★★ 612
Audible Audiobook
\$0.00 Free with Audible trial



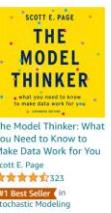
Staff Engineer: Leadership Beyond the Management Track
Bill Larson
★★★★★ 188
Audible Audiobook
\$0.00 Free with Audible trial



Domain-Driven Design: Tackling Complexity in the Heart of Software
Eric Evans
★★★★★ 583
Audible Audiobook
\$0.00 Free with Audible trial



The Kubernetes Book
Scott E. Page
★★★★★ 592
Audible Audiobook
\$0.00 Free with Audible trial



The Model Thinker: What You Need to Know to Make Data Work for You
Niall Ferguson
★★★★★ 523
Audible Audiobook
\$0.00 Free with Audible trial



Microservices Security in Action: Design Secure Network and API
Prashanth Srinivasan
★★★★★ 13
Audible Audiobook
\$0.00 Free with Audible trial



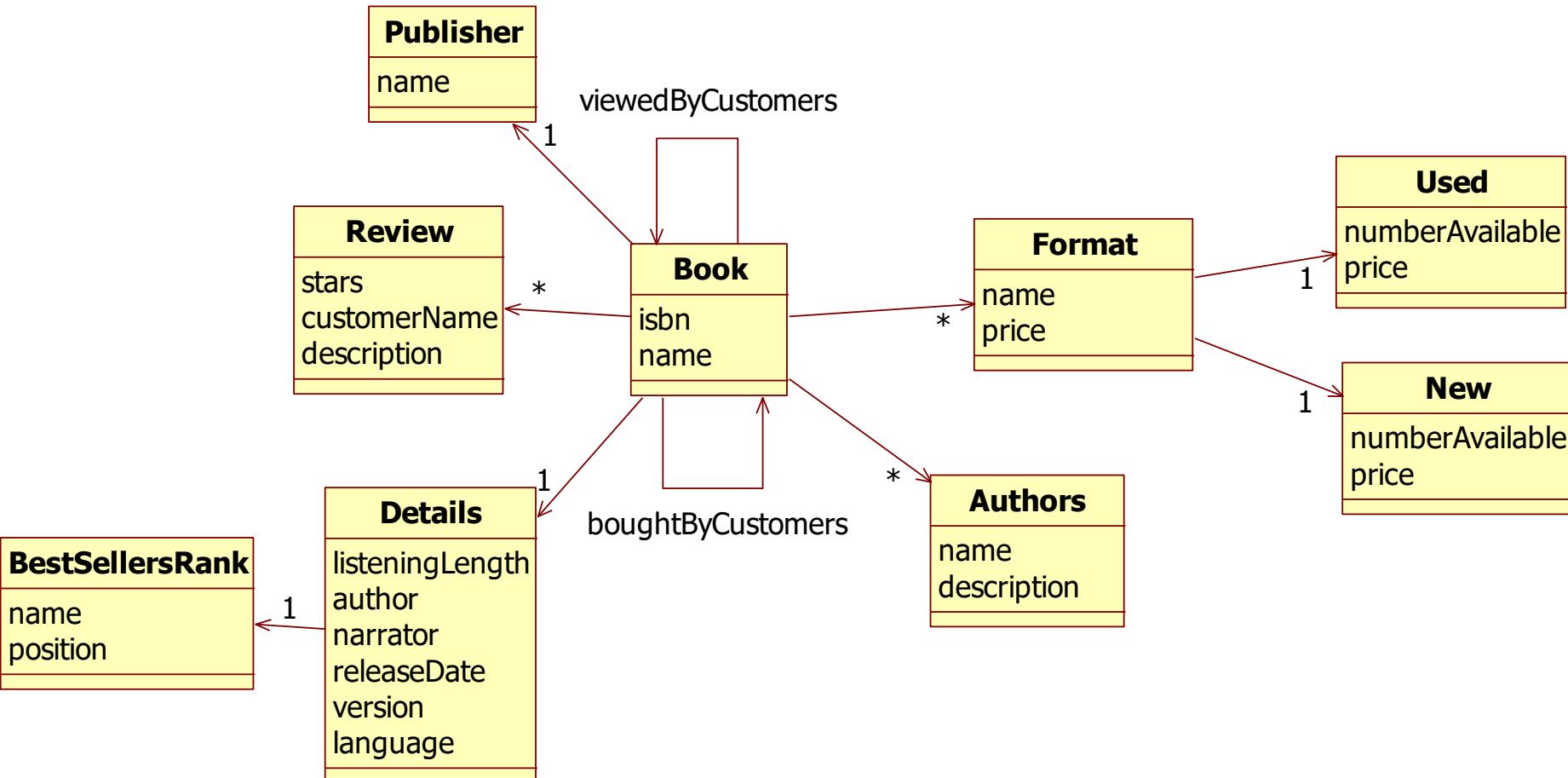
The Dev-Ops Handbook: How to Create World-Class Agility, Reliability, and Efficiency
Gene Kim
★★★★★ 1592
Audible Audiobook
\$0.00 Free with Audible trial

Page 1 of 5

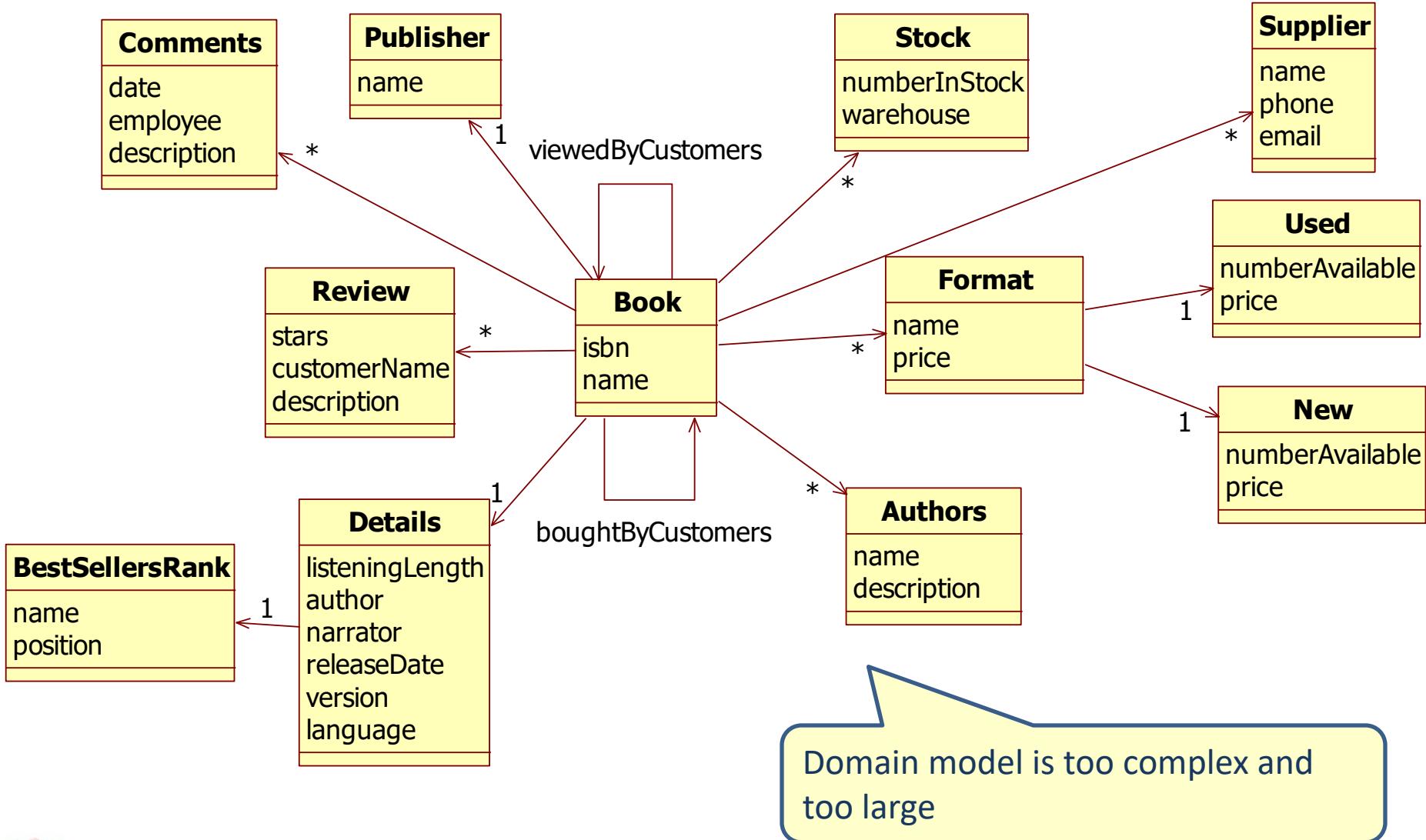
Product details

Listening Length	13 hours and 10 minutes
Author	Mark Richards, Neal Ford
Narrator	Benjamin Lange
Audible.com Release Date	February 27, 2021
Publisher	Upfront Books
Program Type	Audiobook
Version	Unabridged
Language	English
ASIN	B0BX8H15BW
Best Sellers Rank	#5,062 in Audible Books & Originals (See Top 100 in Audible Books & Originals) #2 in Computer Systems Analysis & Design (Books) #3 in Software Design Tools #26 in Computers & Technology (Audible Books & Originals)

Amazon.com book



Amazon.com book

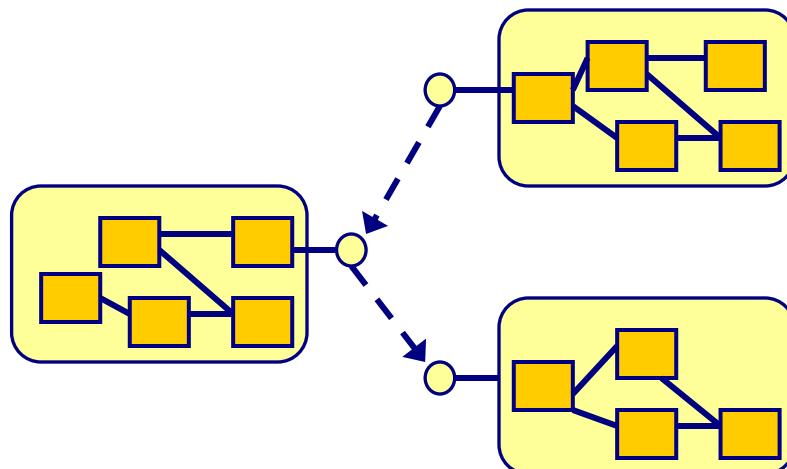


COMPONENT BASED DESIGN



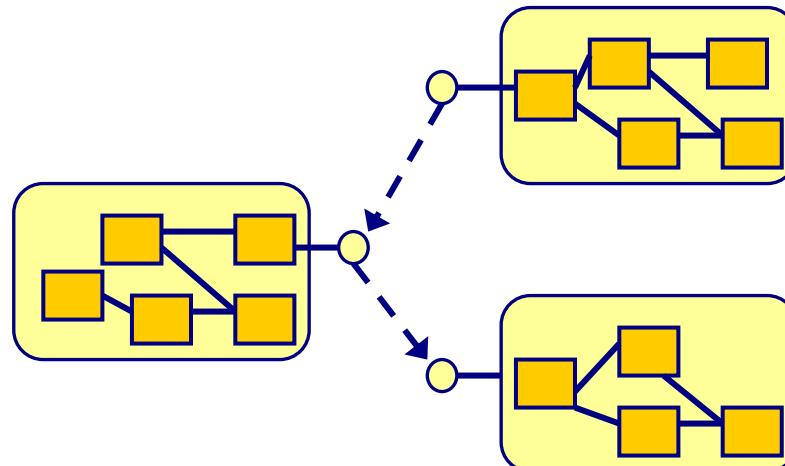
Component Based Development

- Decompose the domain in functional components



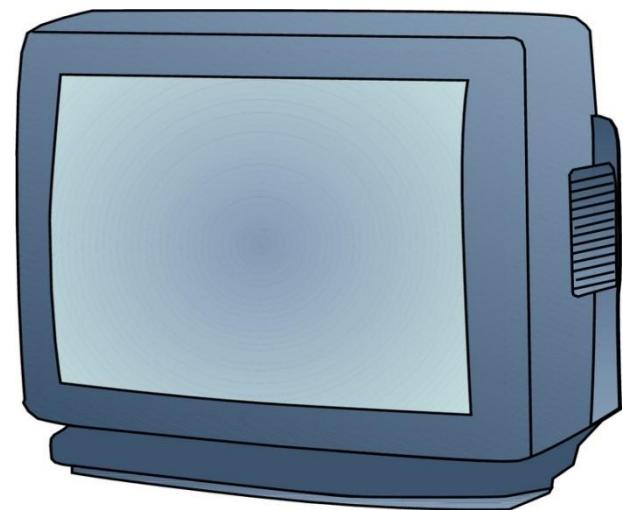
What is a component?

- There is no definition
- What we agree upon:
 1. A component has an **interface**
 2. A component is **encapsulated**
- Plug-and-play
- A component can be a single unit of deployment



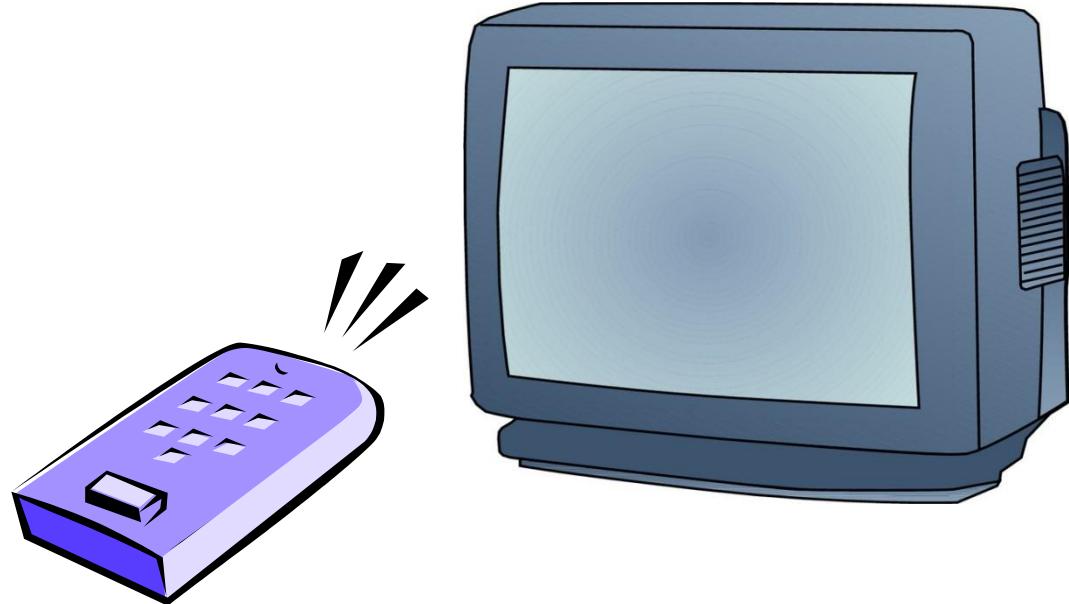
Encapsulation

- The implementation details are hidden



Interface

- The interface tells what you can do (but not how)



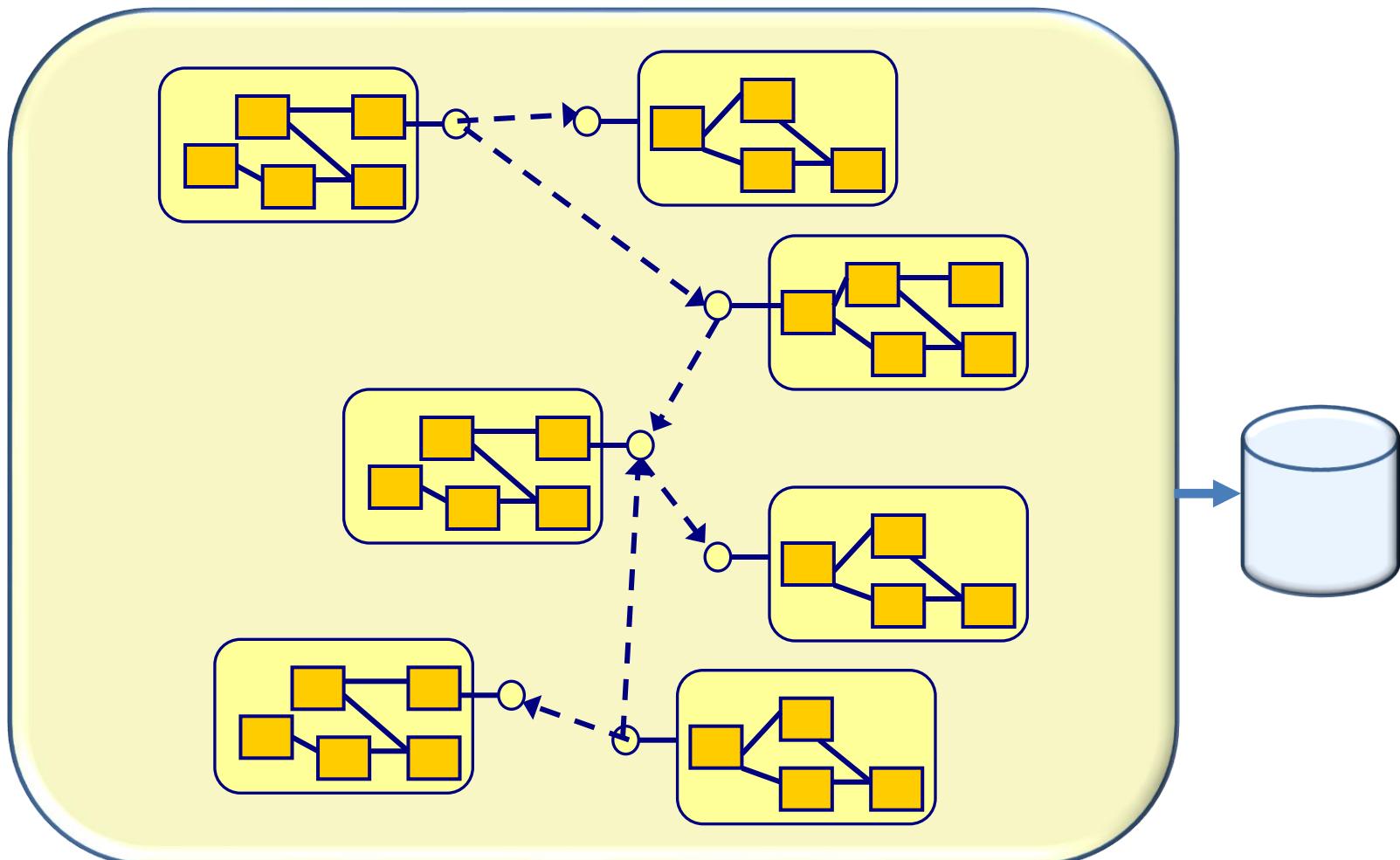
Advantages of components

- High cohesion, low coupling
- Flexibility
- Reuse ?

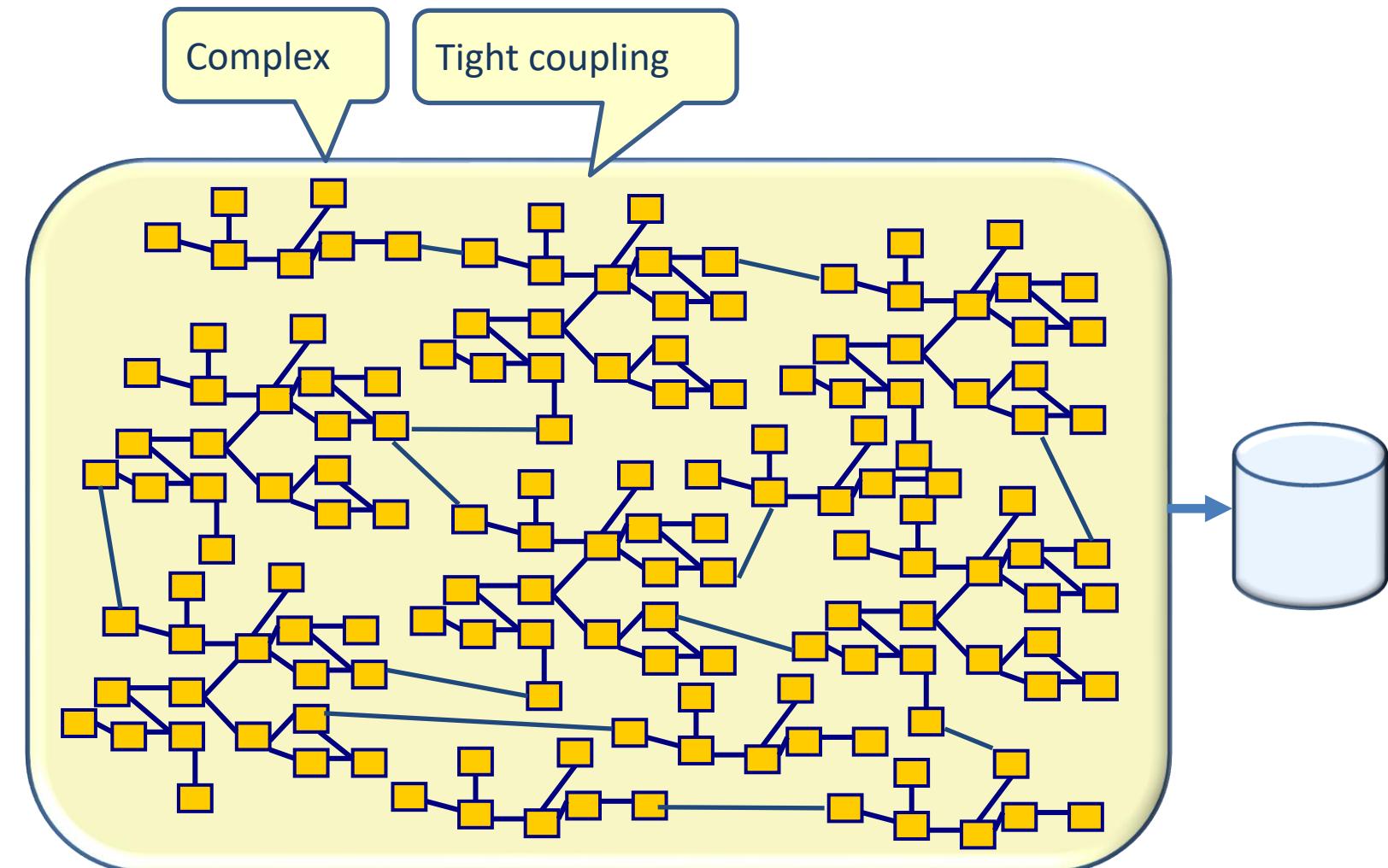


Component Based Development

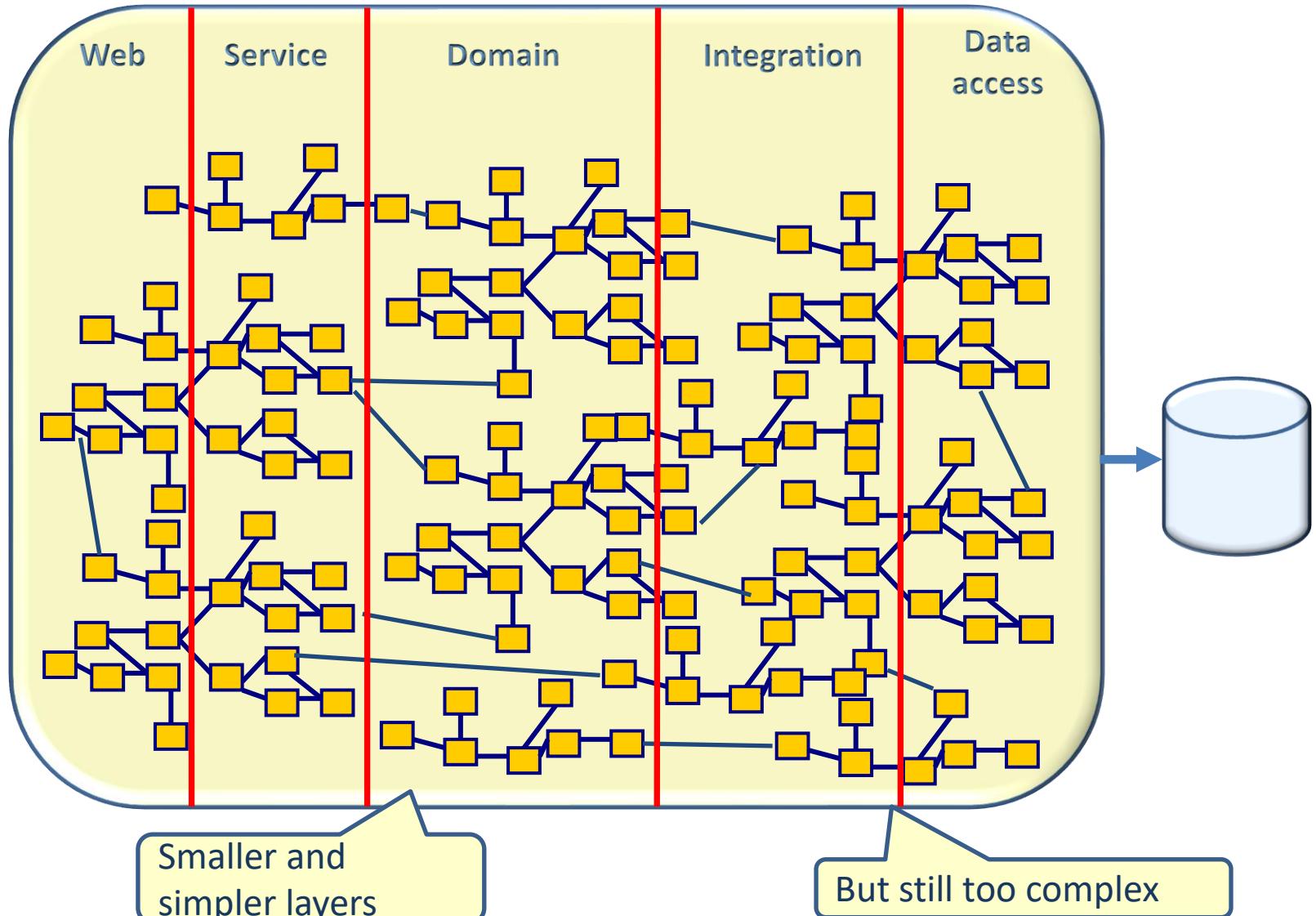
- Decompose the domain in functional components



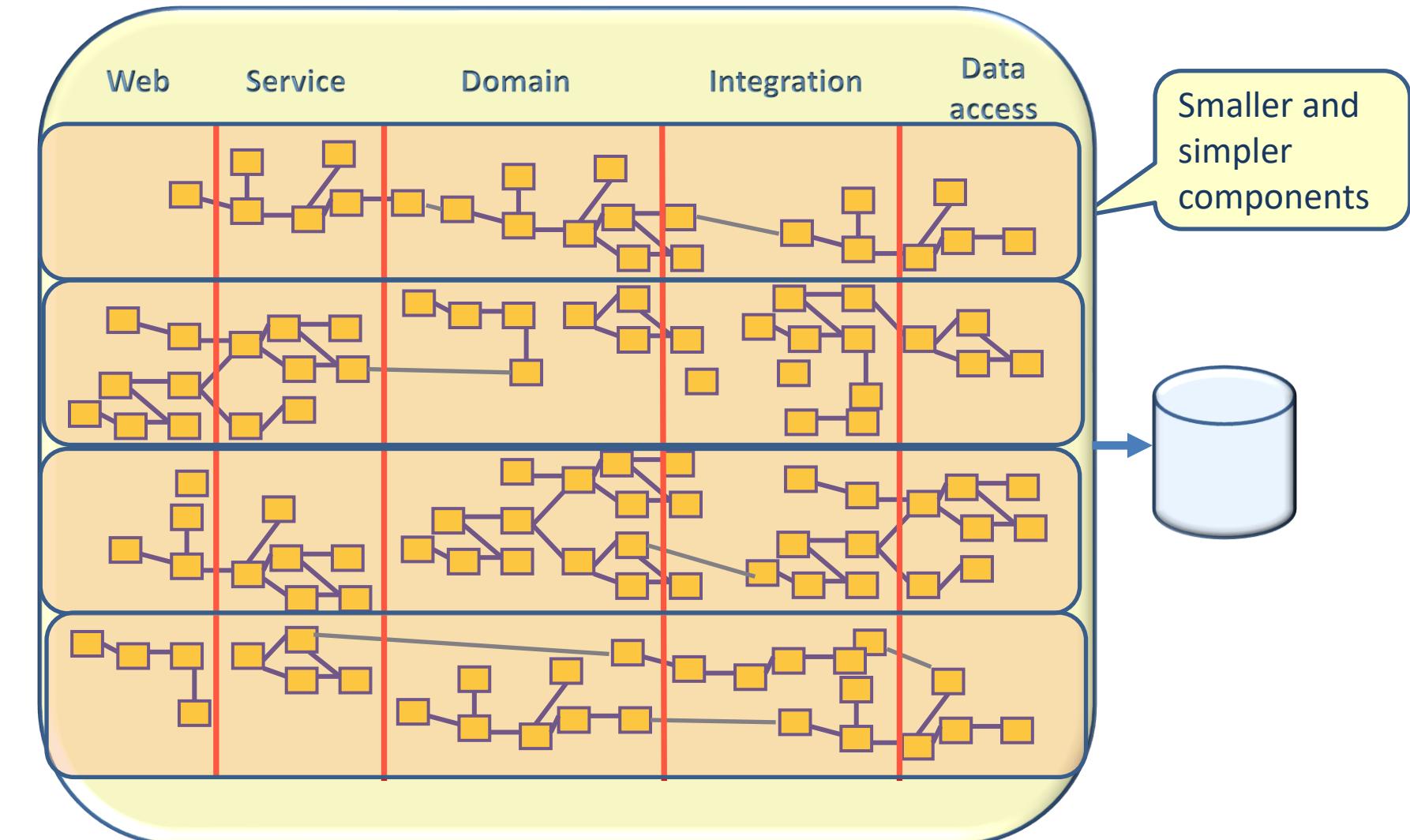
Object orientation



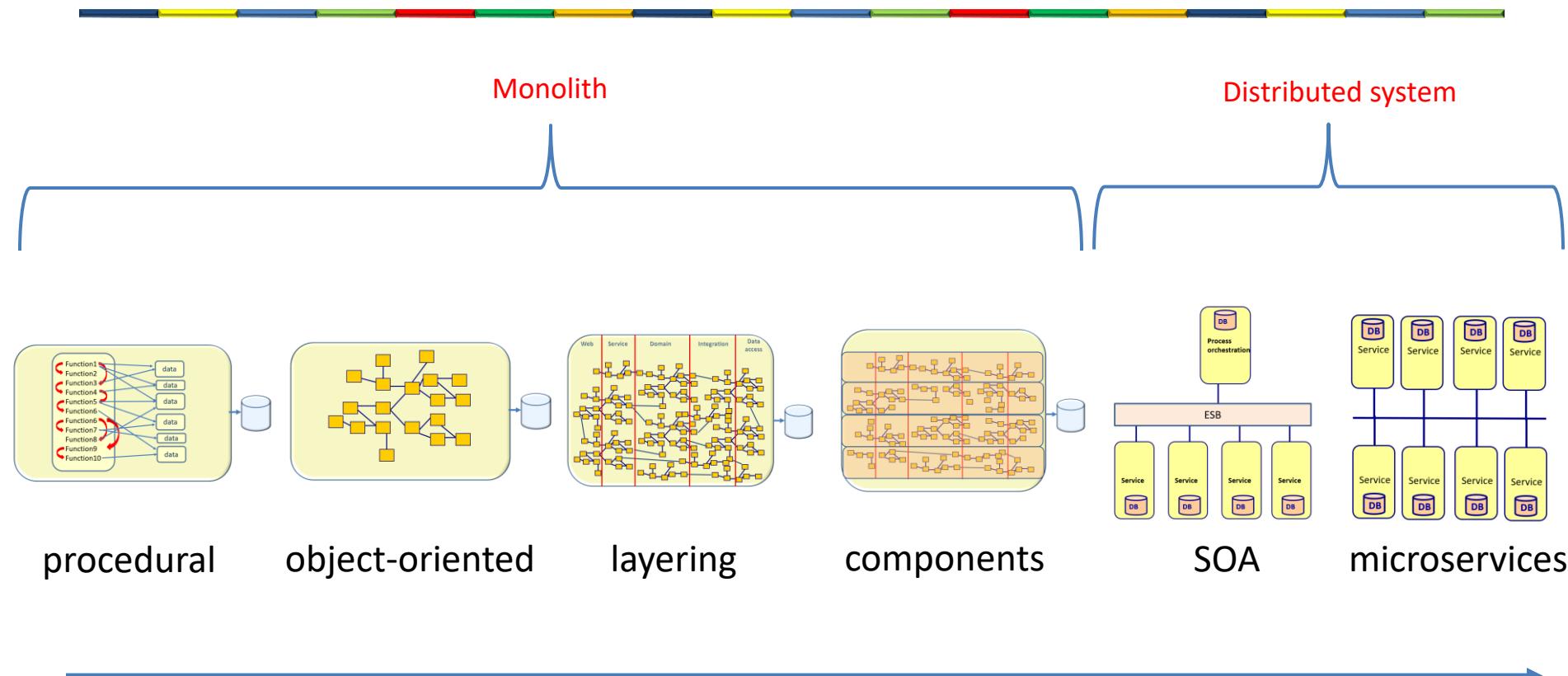
Layering



Components



Architecture evolution



- Smaller and simpler parts
- More separation of concern
- More abstraction
- Less dependencies



Main point

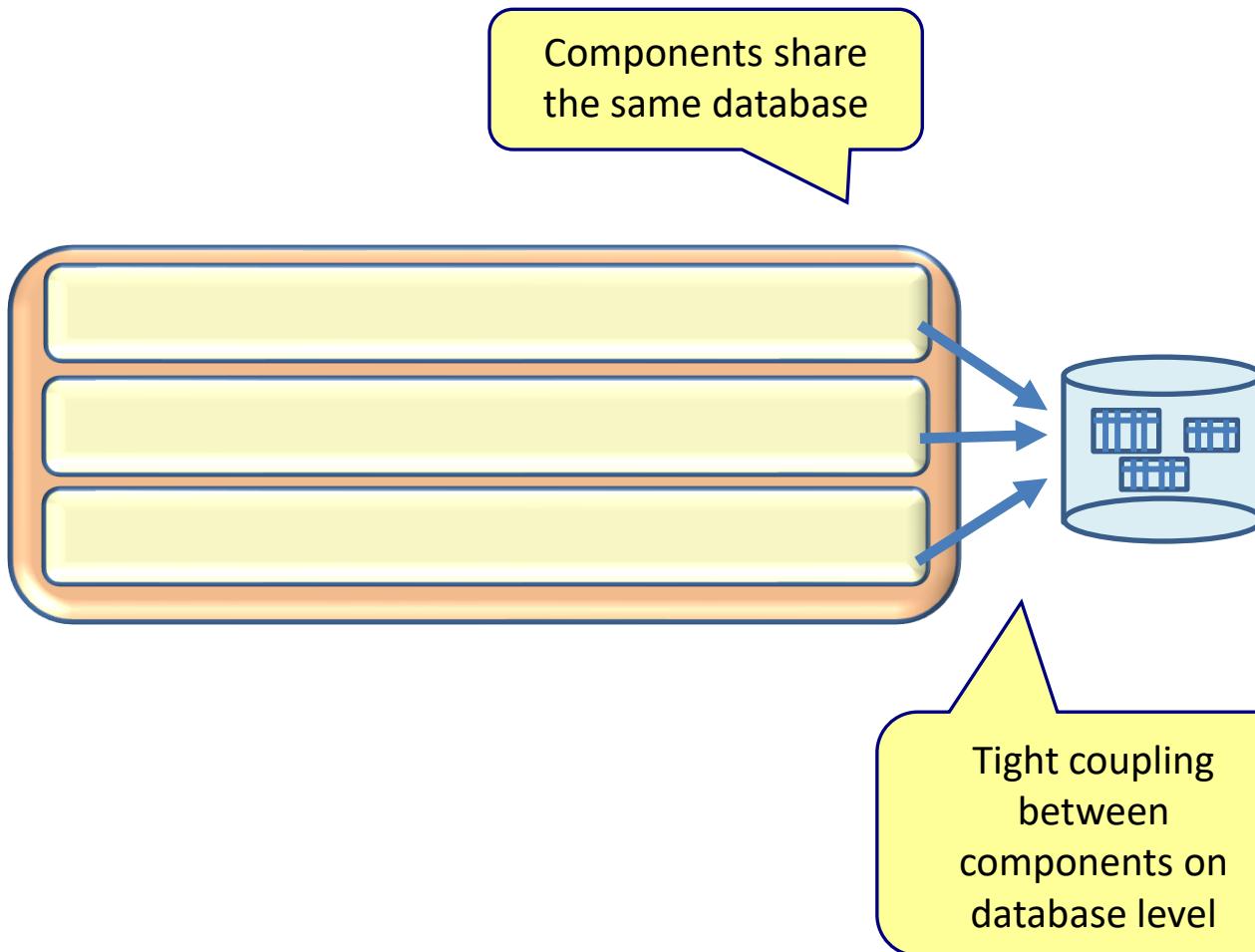
- Components are encapsulated and completely autonomous plug-and-play elements.
- The human nervous system is capable to transcend to that abstract field of pure consciousness which lies at the basis of the whole creation.



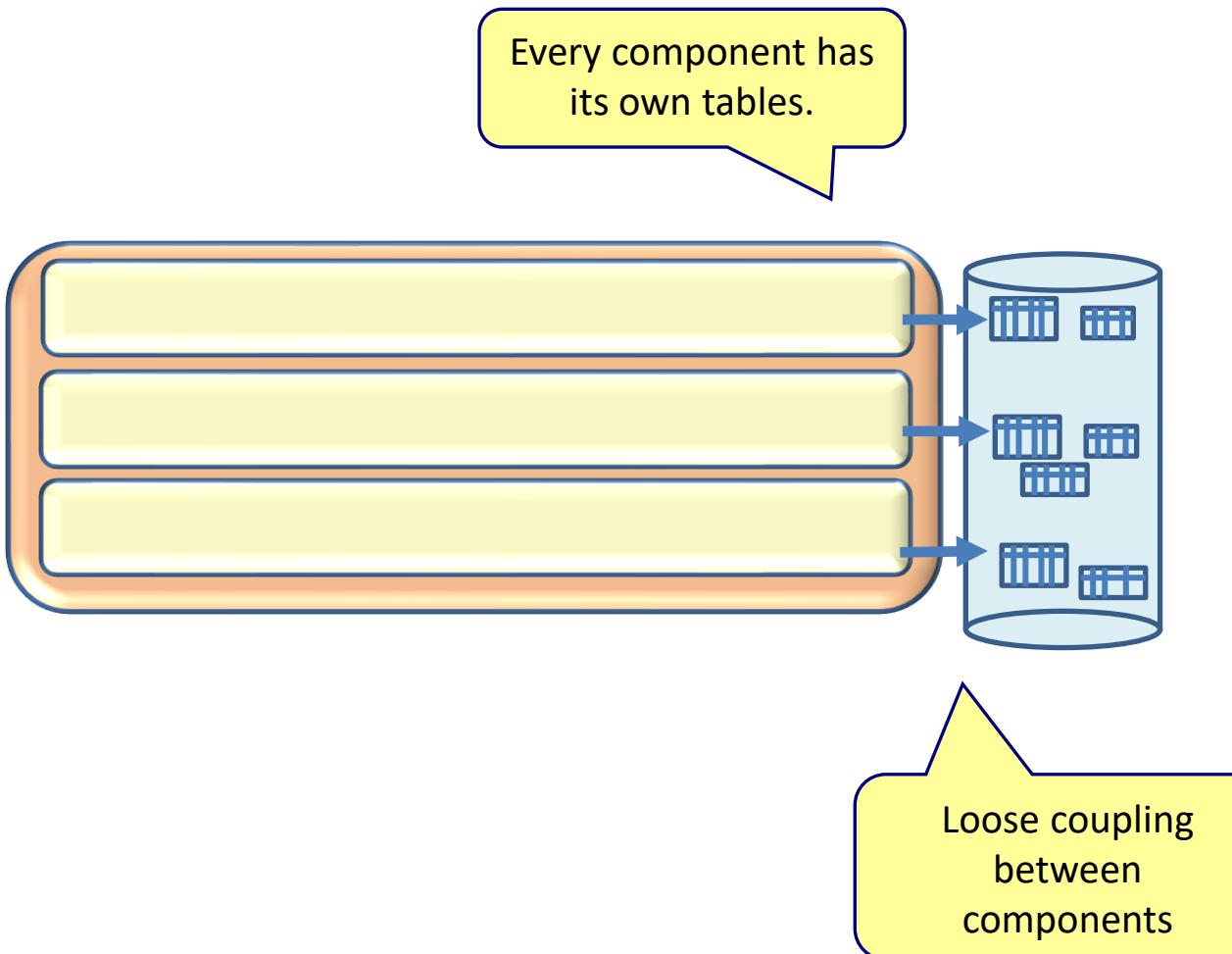
COMPONENT DESIGN



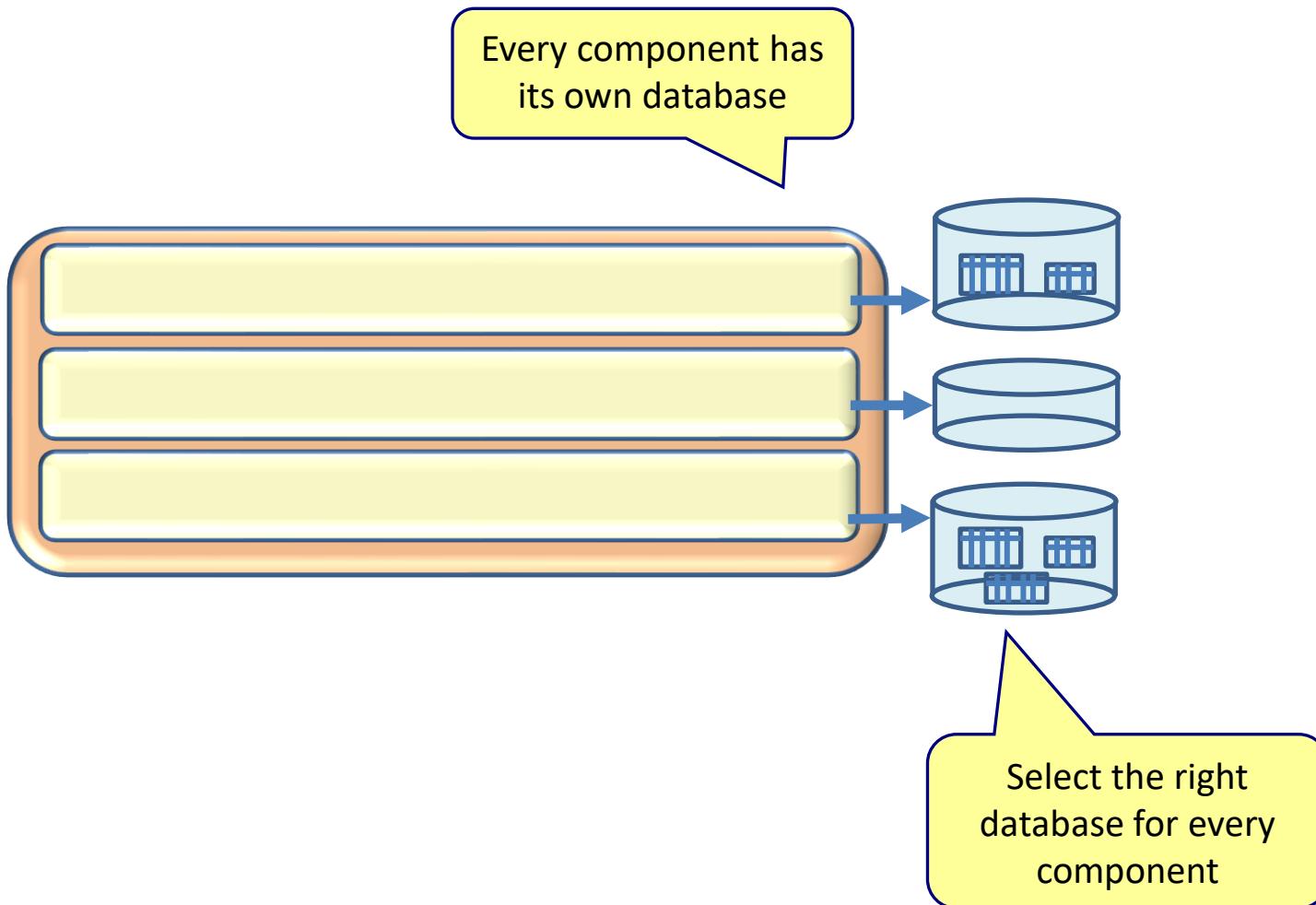
Components and database



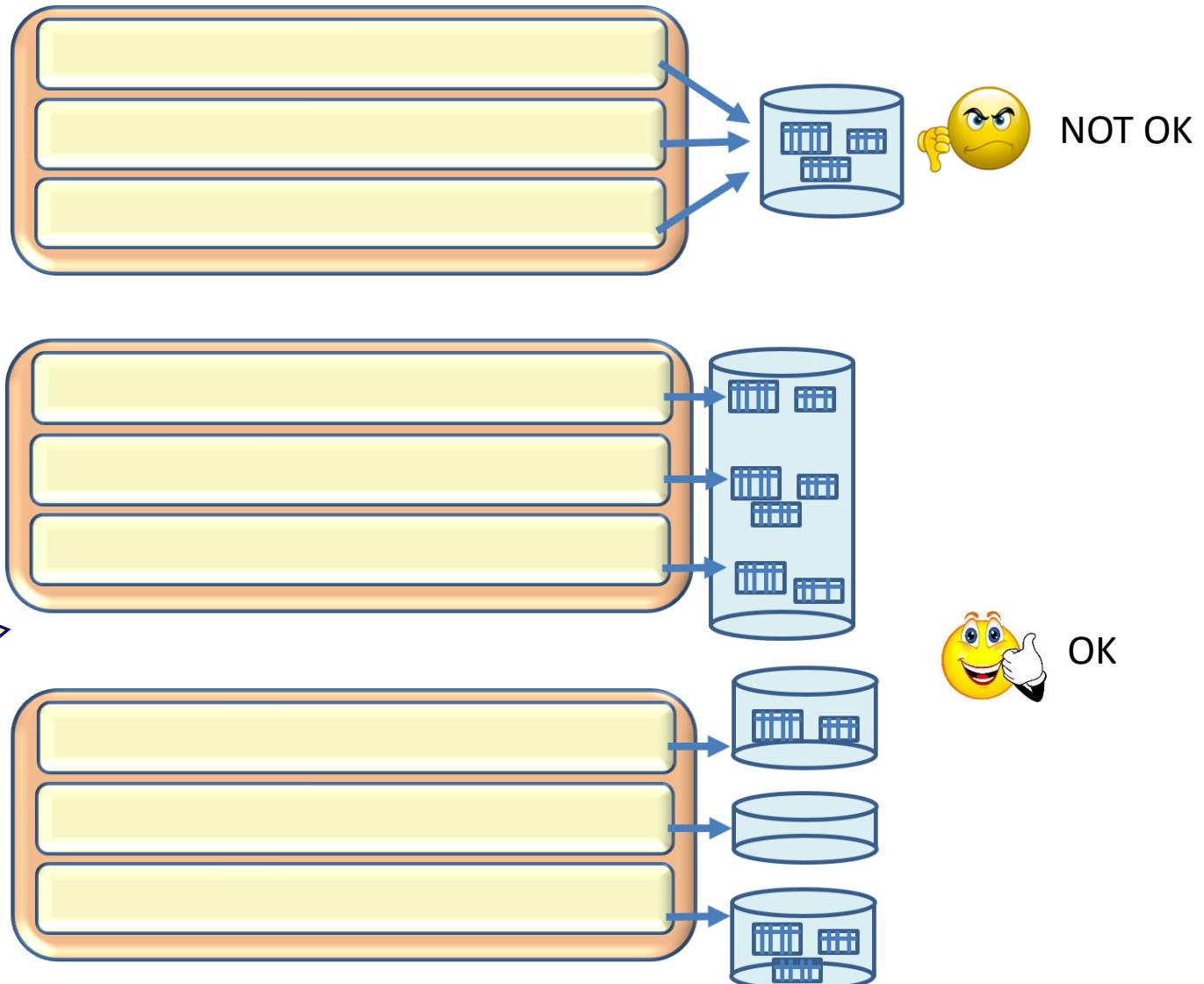
Components and database



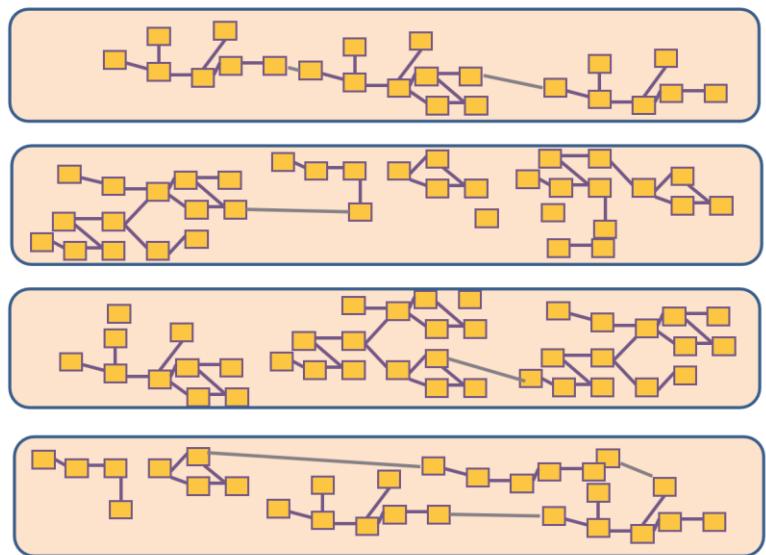
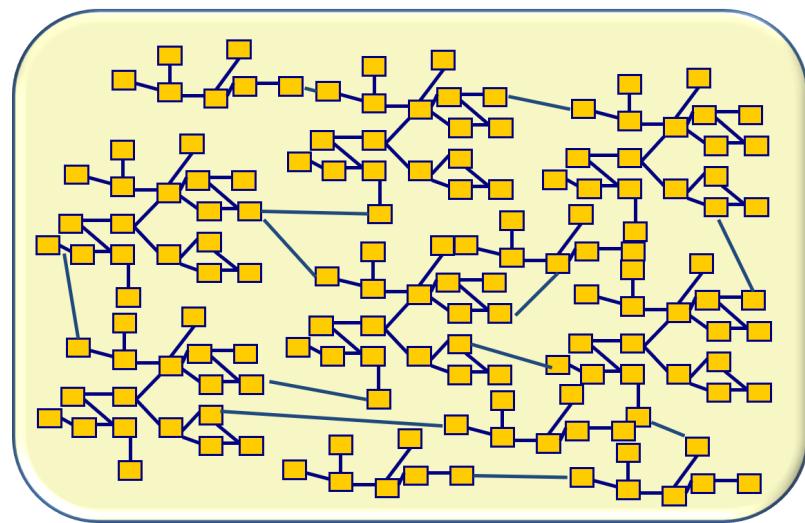
Components and database



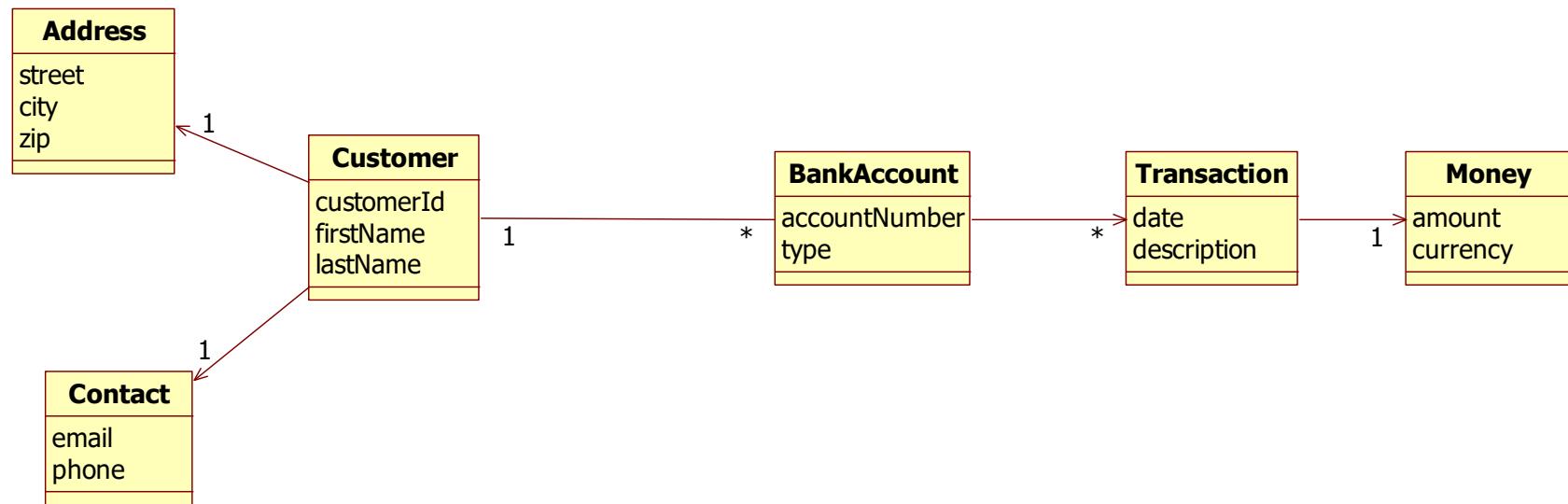
Components and database



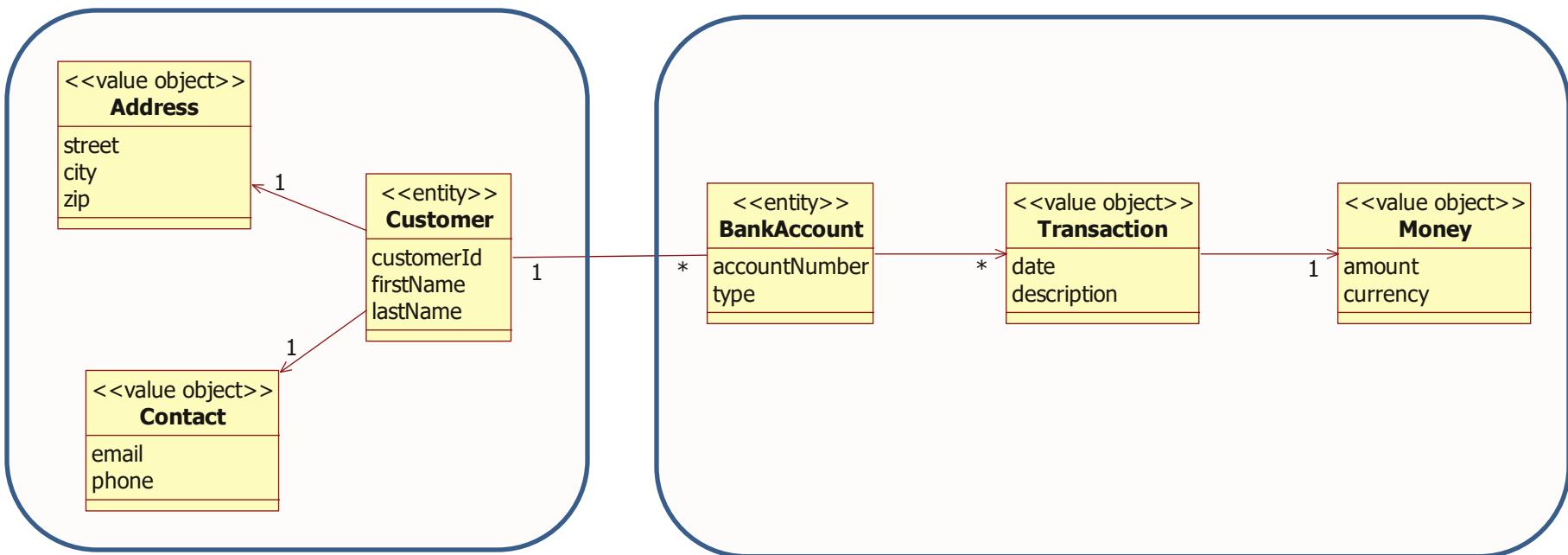
How do we split classes into components?



How do we split classes?

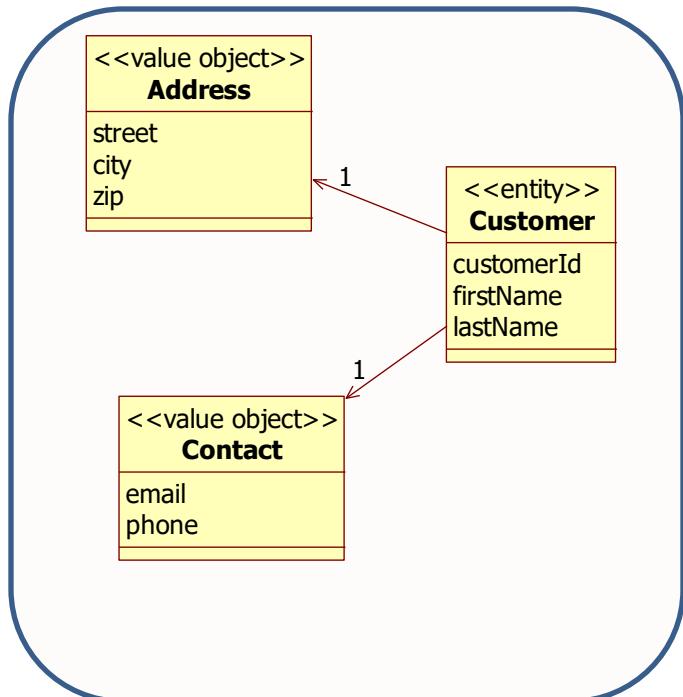


Aggregates

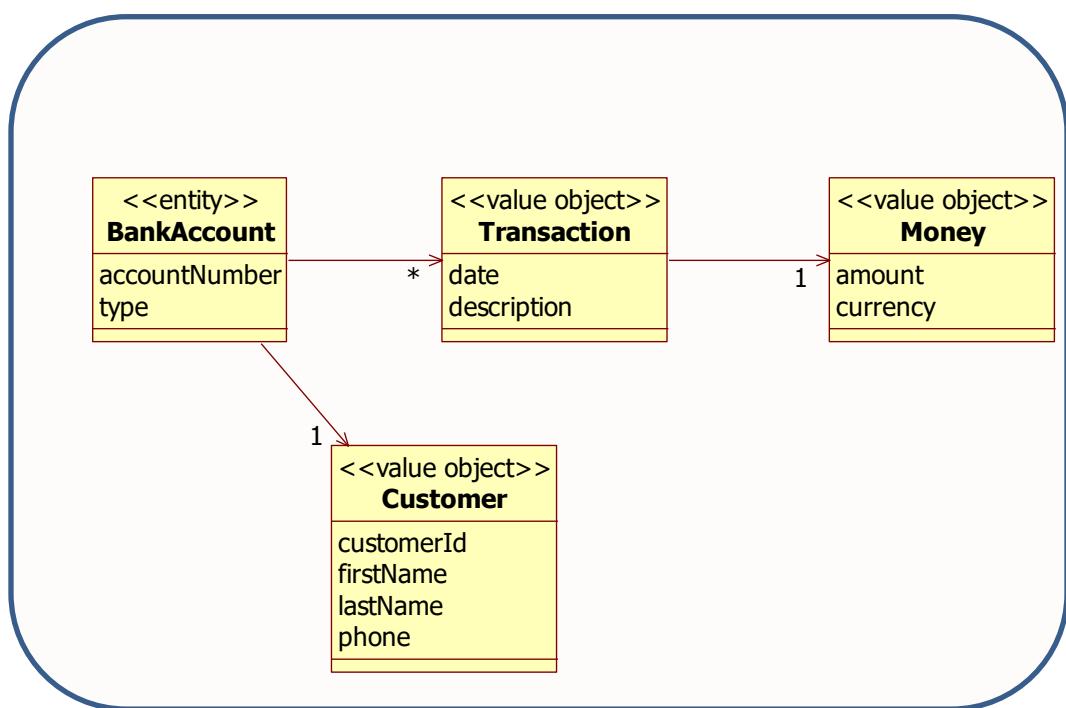


Aggregates

Customer component domain model



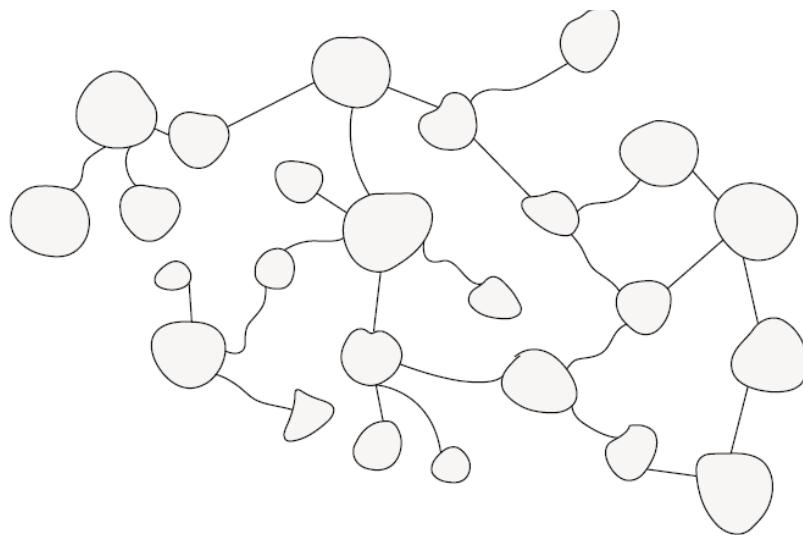
Account component domain model



Simplicity of the domain model



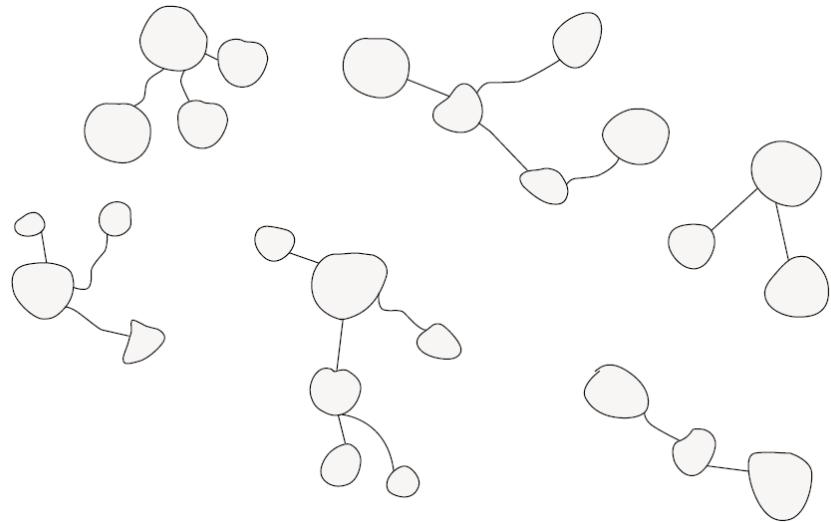
NOT OK



Complex domain model with
many unnecessary associations



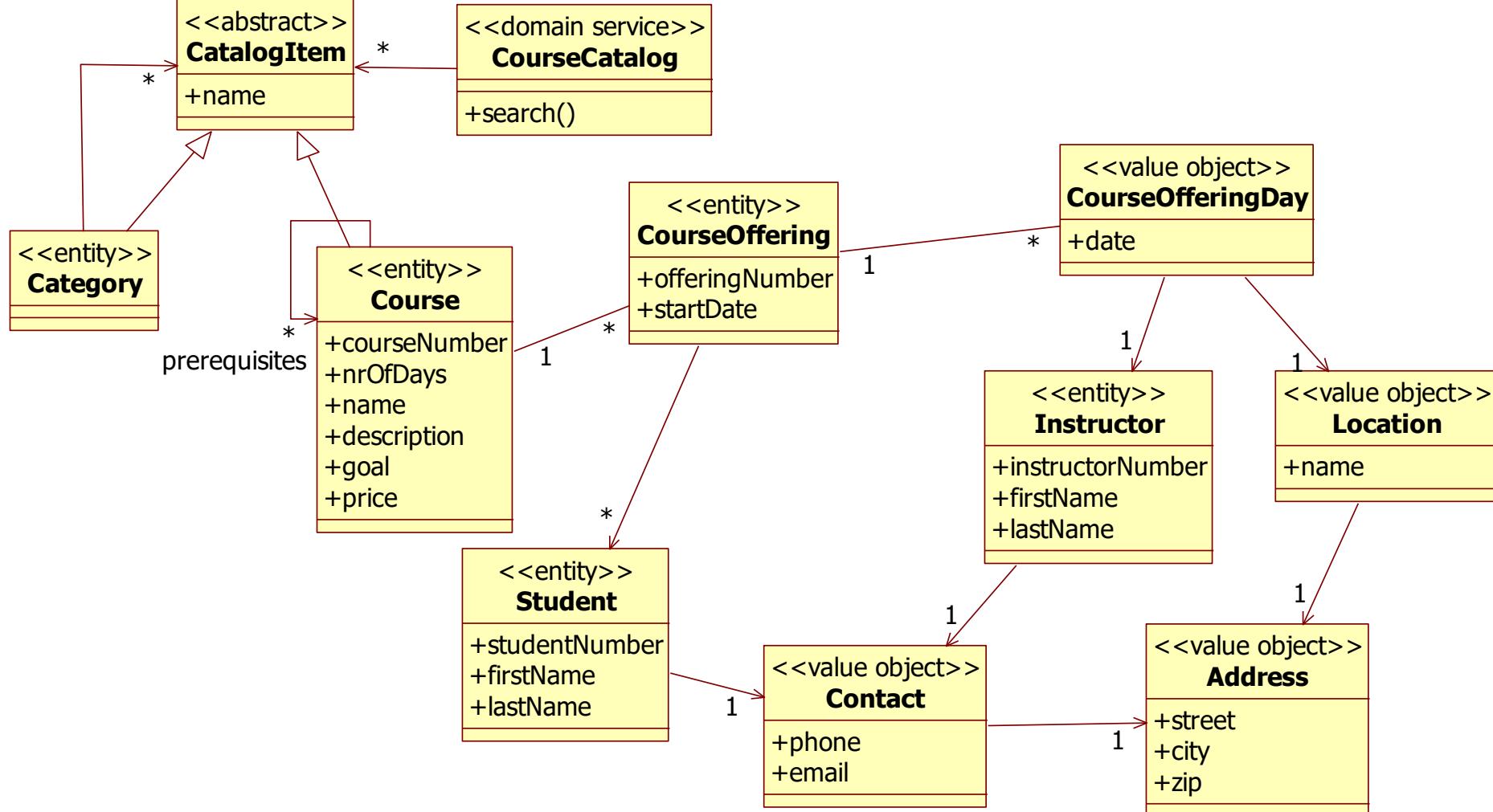
OK



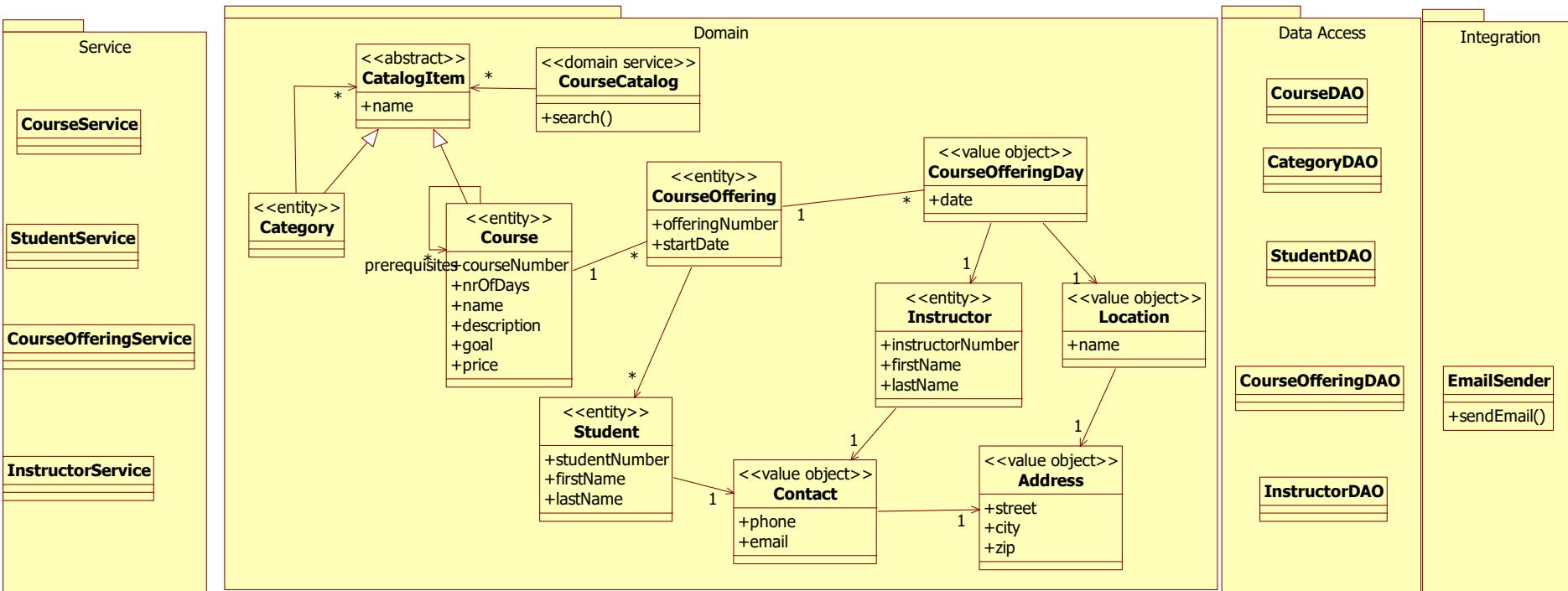
Clearer domain model with only
the essential associations



Course Registration Domain Model



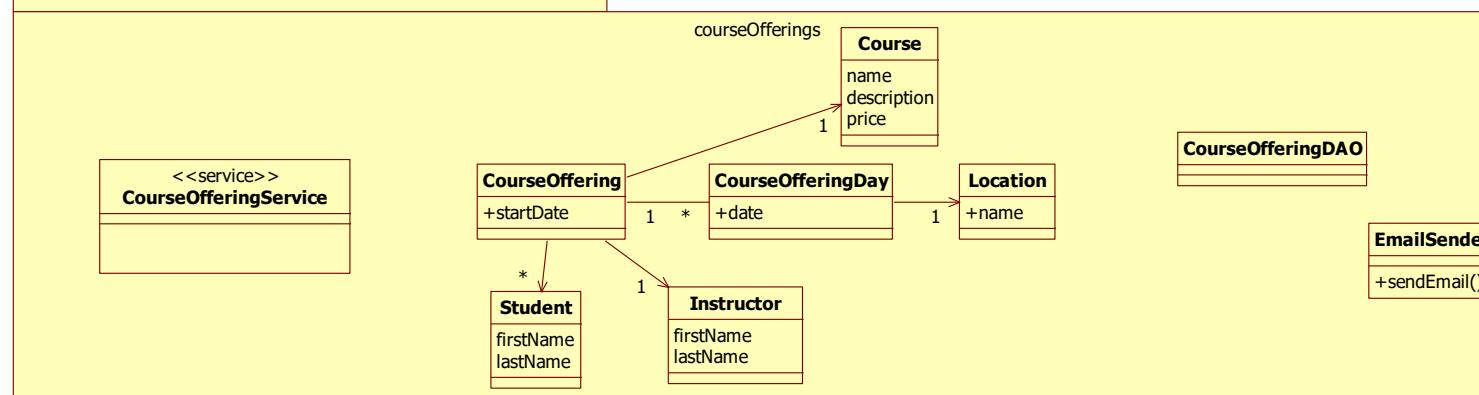
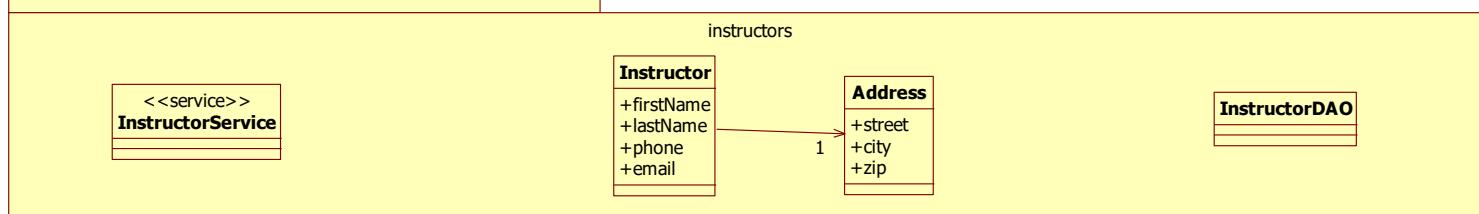
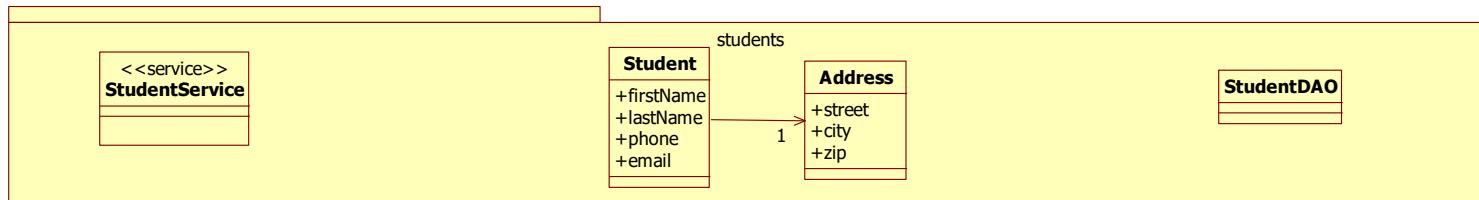
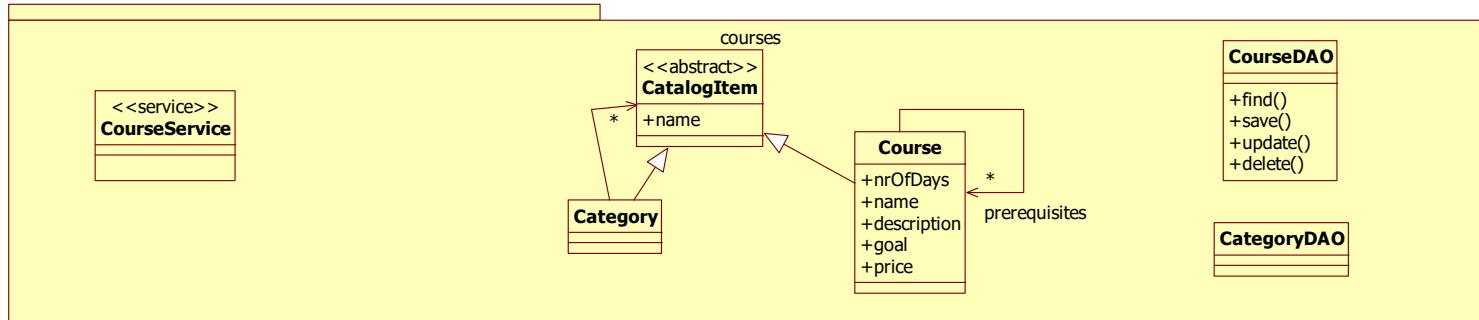
Course Registration Design



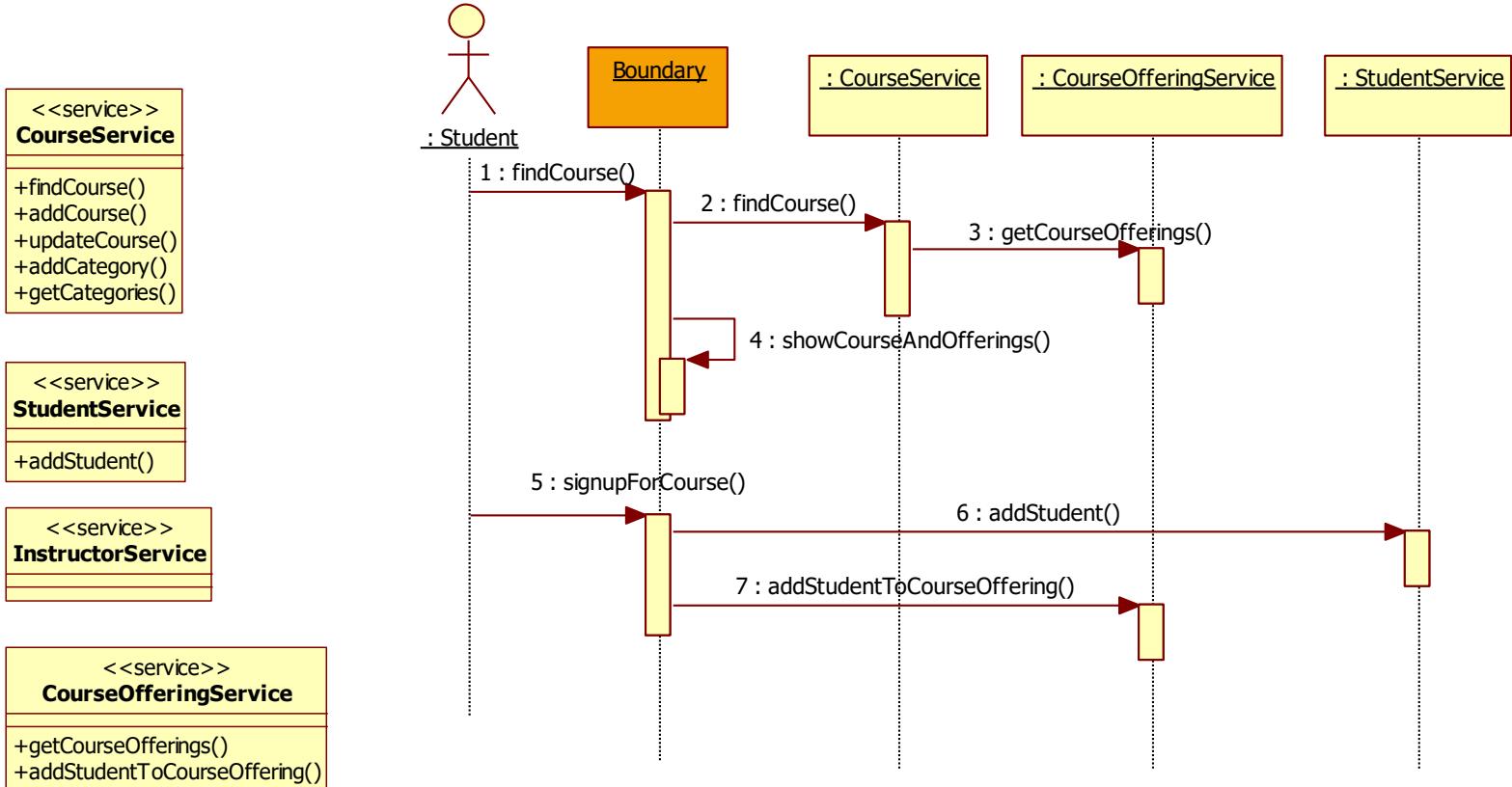
Course Registration with components



Course Registration with components



Course Registration with components

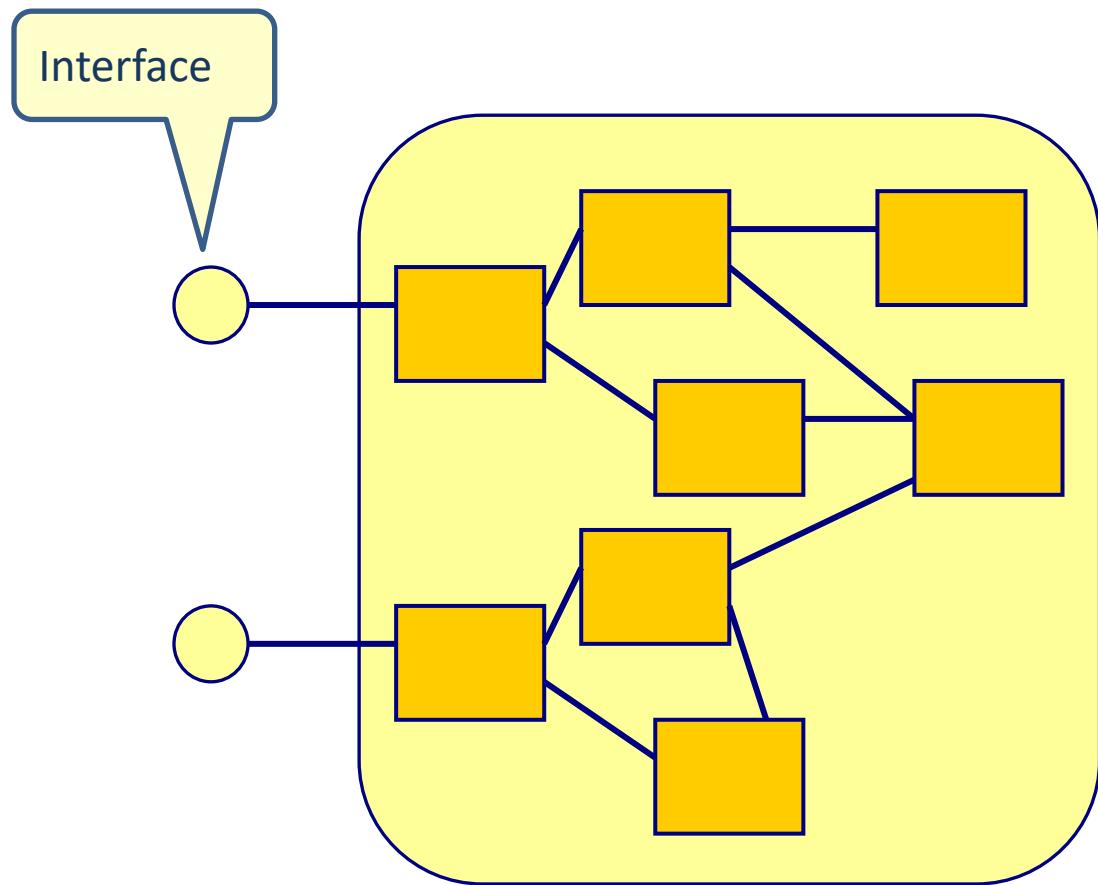


Lesson 5

API DESIGN

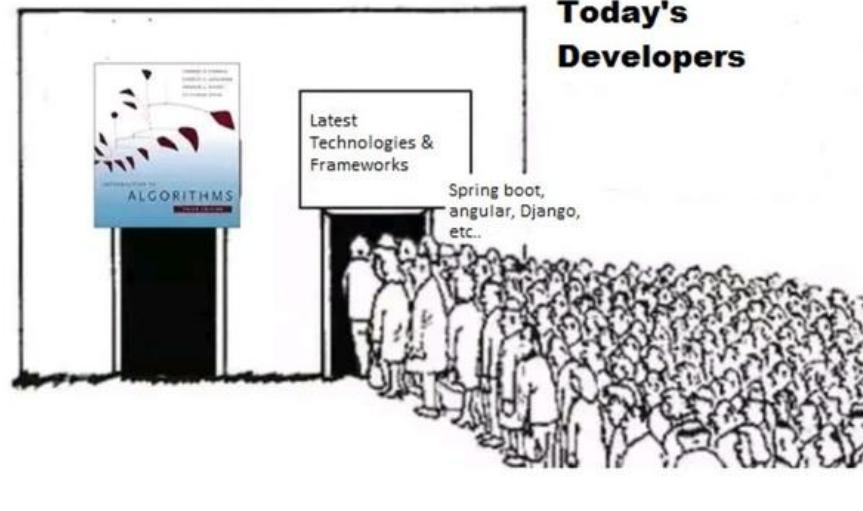


Interface design



Why is API design important?

- APIs play a bigger role in software development than ever before
- Software development has shifted from
 - Designing algorithms and data structures
 - To
 - Choosing, learning, and using an ever-growing set of API's



Why is API design important?

- Good APIs increase the quality of our applications
 - Bad API design infects our complete application
- The API we use shapes our code
 - API authors are responsible for the quality of the client code



Manual versus automatic transmission

Transmission complexity is hidden



Transmission complexity is exposed to the driver



- + Simple for the driver
- + Less chance on mistakes
- + Hard to misuse
- Driver has less control

- More complex for the driver
- Easier to make mistakes
- Easy to misuse
- + Driver has more control

JDBC API client

```
public void update(Employee employee) {  
    Connection conn = null;  
    PreparedStatement prepareUpdateEmployee = null;  
    try {  
        conn = getConnection(); Open connection  
        conn.setAutoCommit(false); Start transaction  
        prepareUpdateEmployee = conn.prepareStatement("UPDATE Employee SET firstname= ?,  
                                                    lastname= ? WHERE employeenumber=?");  
        prepareUpdateEmployee.setString(1, employee.getFirstName());  
        prepareUpdateEmployee.setString(2, employee.getLastName());  
        prepareUpdateEmployee.setLong(3, employee.getEmployeeNumber()); Send the SQL  
        int updateresult = prepareUpdateEmployee.executeUpdate();  
        conn.commit(); Commit transaction  
    } catch (SQLException e) {  
        conn.rollback(); Rollback transaction  
        System.out.println("SQLException in EmployeeDAO update() :" + e);  
    } finally {  
        try {  
            prepareUpdateEmployee.close();  
            closeConnection(conn); Close connection  
        } catch (SQLException e1) {  
            System.out.println("Exception in closing jdbc connection in EmployeeDAO" + e);  
        }  
    }  
}
```

JDBC Template client

```
public void save(Product product) {  
    NamedParameterJdbcTemplate jdbctempl = new NamedParameterJdbcTemplate (dataSource);  
    Map<String, Object> namedParameters = new HashMap<String, Object>();  
    namedParameters.put("productnumber", product.getProductnumber());  
    namedParameters.put("name", product.getName());  
    namedParameters.put("price", product.getPrice());  
    int updateresult = jdbctempl.update("INSERT INTO product VALUES ( :productnumber,  
                                         :name, :price)",namedParameters);  
}
```

The template takes care of connection,
transaction and exception handling



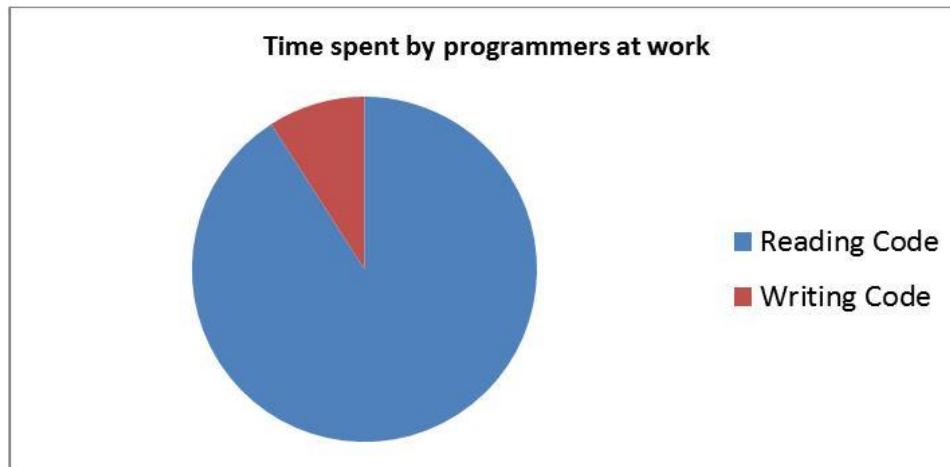
Why is API design important?

- Public APIs are forever
 - One chance to get it right



Why is API design important?

- Usually write-once, read/learn many times by many different people
 - Developers spend most of their time learning APIs, using APIs and debugging code



Does it run? Just leave it alone.



Writing Code that
Nobody Else Can Read

The Definitive Guide

Why is API design important to you?

- If you program, you are an API designer
 - Good code is modular
 - Each module has an API
- Useful modules tend to get reused
 - Once a module has users, can't change API at will

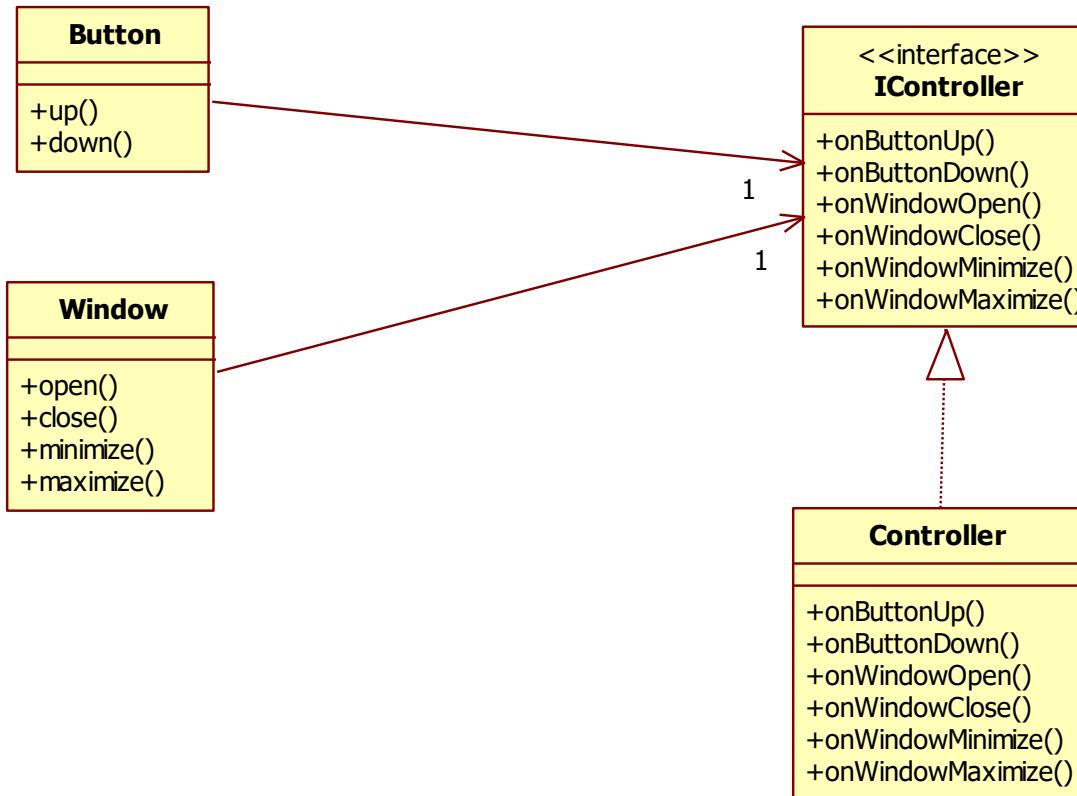


Interface design best practices

- Start with the client first
 - Single responsibility principle
 - Interface segregation principle
- Easy to use
- Easy to learn
- Hide as much as possible
- When in doubt, leave it out

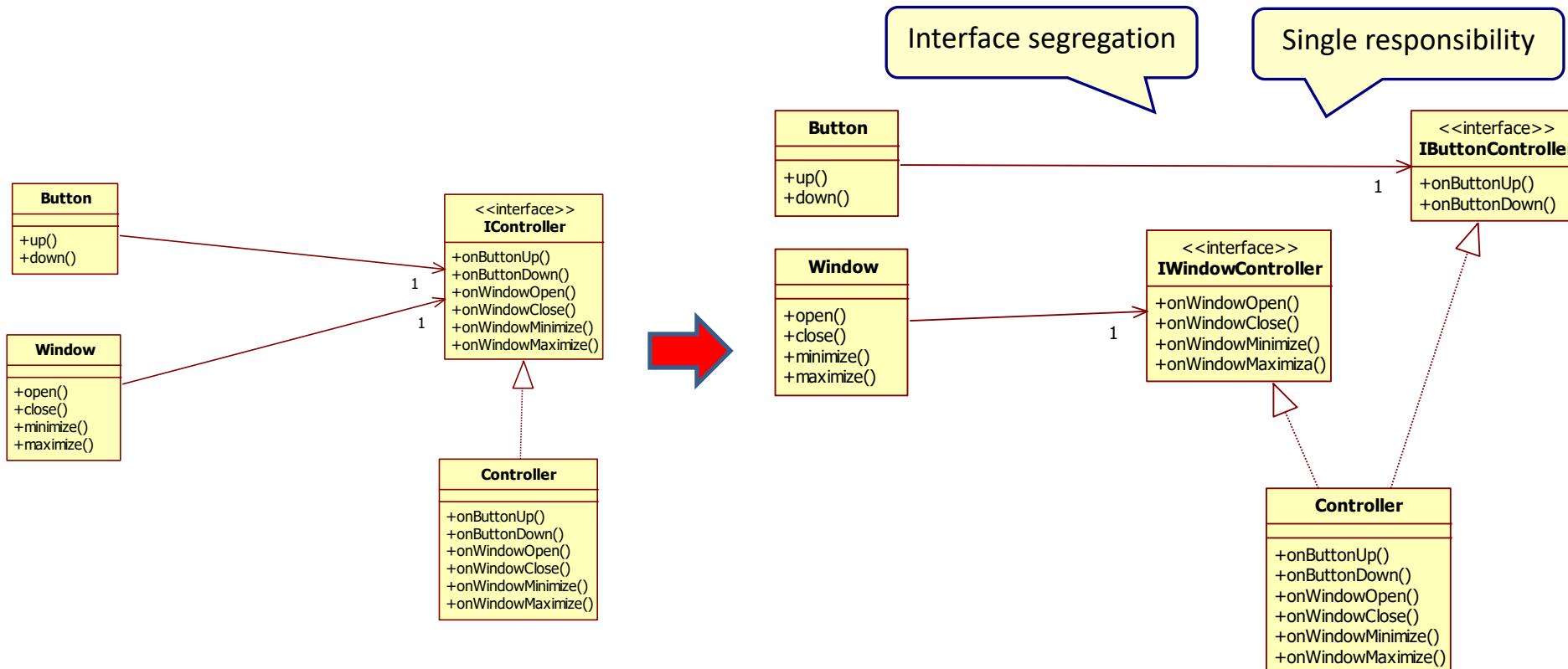


What is wrong with this API?



Interface segregation principle

- Clients should not be forced to depend on methods (and data) they do not use



The interface should be easy to use

- And hard to misuse.
 - Easy to do simple things
 - Possible to do complex things
 - Impossible (or at least difficult) to do wrong things
- The interface should be simple
- No surprising behavior
 - Don't do anything that is relevant to the client and that you cannot derive from the names of the functions and parameters (or comments)

Semantics



Easy to use

```
String findString( String text,  
                  bool search_forward,  
                  bool case_sensitive);
```



NOT OK

```
findString("text", true, false);
```

- Order is clear
- Self-descriptive.

```
enum SearchDirection {  
    FORWARD, BACKWARD  
};
```

```
enum CaseSensitivity {  
    CASE_SENSITIVE, CASE_INSENSITIVE  
};
```

```
String findString( String text,  
                  SearchDirection direction,  
                  CaseSensitivity case_sensitive);
```

```
findString("text",  
          SearchDirection.FORWARD,  
          CaseSensitivity. CASE_INSENSITIVE);
```

Prefer enums over
booleans to improve
code readability.



OK



The interface should be easy to learn

- Well documented
- Names matter (classes, functions, variables)
 - Clear code

`bucket.empty()`



NOT OK

`getCurValue()`
`getCurValue()`
`getCurVal()`

`bucket.isEmpty()`

`bucket.makeEmpty()`



OK

`getCurrentValue()`



Do not expose implementation details

- The client should be independent of implementation details
 - You can change the implementation without changing the client
- Don't let implementation details “leak” into the API
 - An API method that throws a SQL exception
 - An API method that returns a hash table



Minimally complete

- But no smaller
- API should satisfy its requirement
- **When in doubt, leave it out**
 - You can always add
 - You can never remove
- API should do one thing, and do it well
- Do not add extra levels of generality for the future
 - You might never need it
 - When you need it, you have a better understandability of what is needed
 - Its easier to add to a simple API than a complex API

It is tempting to expose all possible functionality you can think of.

If it is not required, then leave it out.

The API should be as small as possible



Use immutable objects on the interface

- Immutable object
 - Object whose internal state cannot be changed after its creation
- Advantages
 - Immutable objects are simple.
 - Immutable objects are inherently thread-safe; they require no synchronization.
 - Immutable objects can be shared freely.
 - Immutable objects make great building blocks for other objects



Don't make the client do anything the module can do

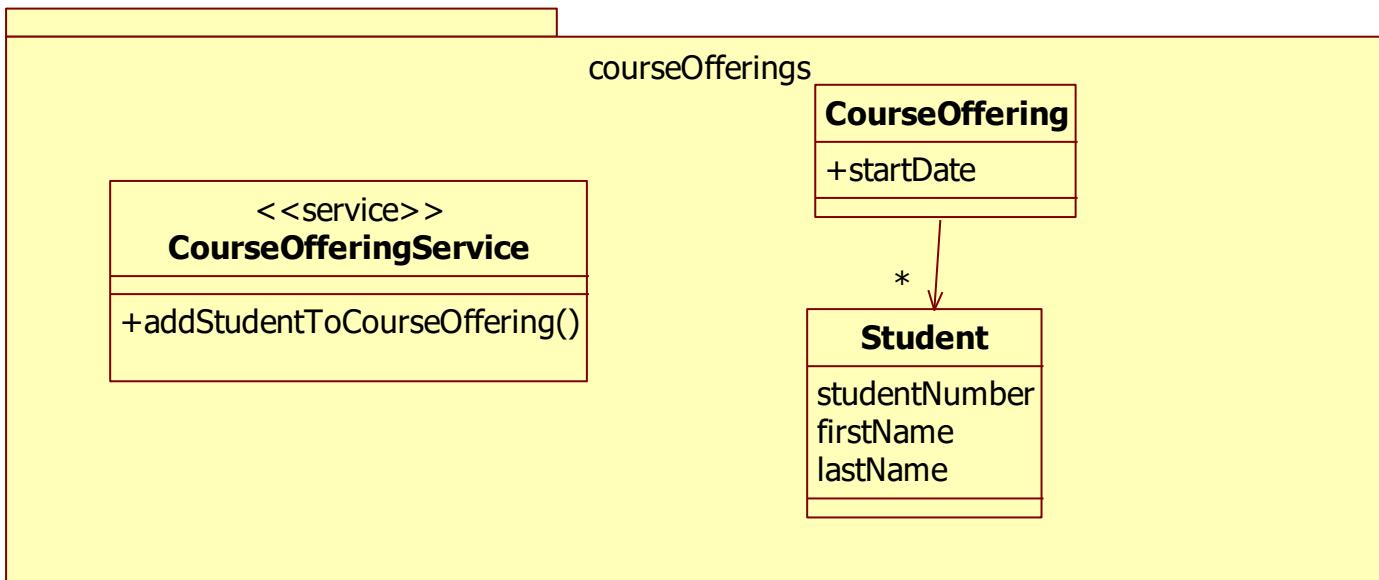
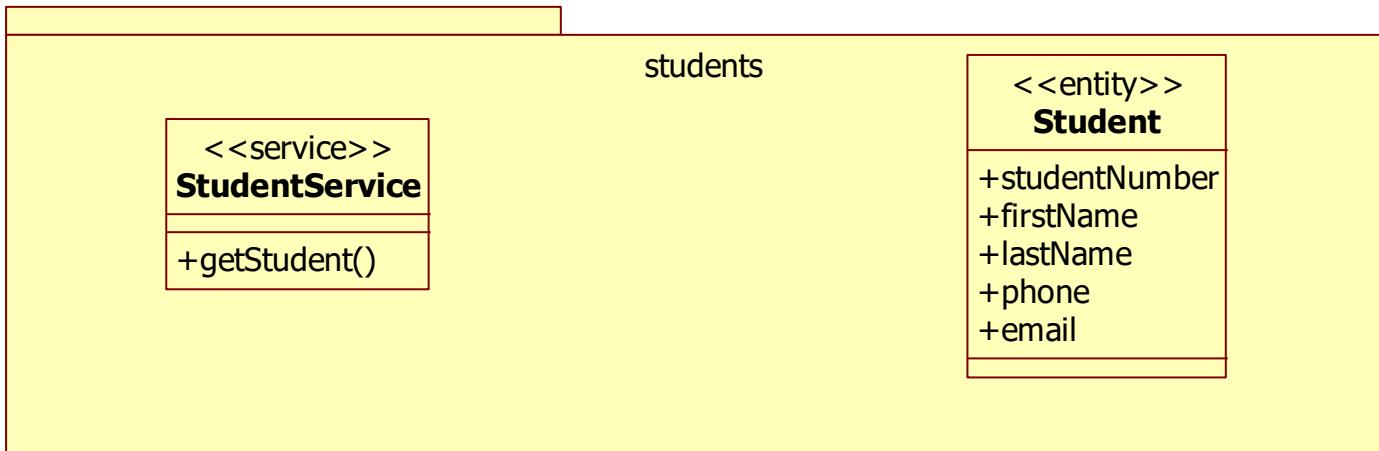
- Reduce the need for boilerplate code
 - Generally done via cut-and-paste
 - Ugly, annoying, and error-prone



DATA TRANSFER OBJECTS (DTO)

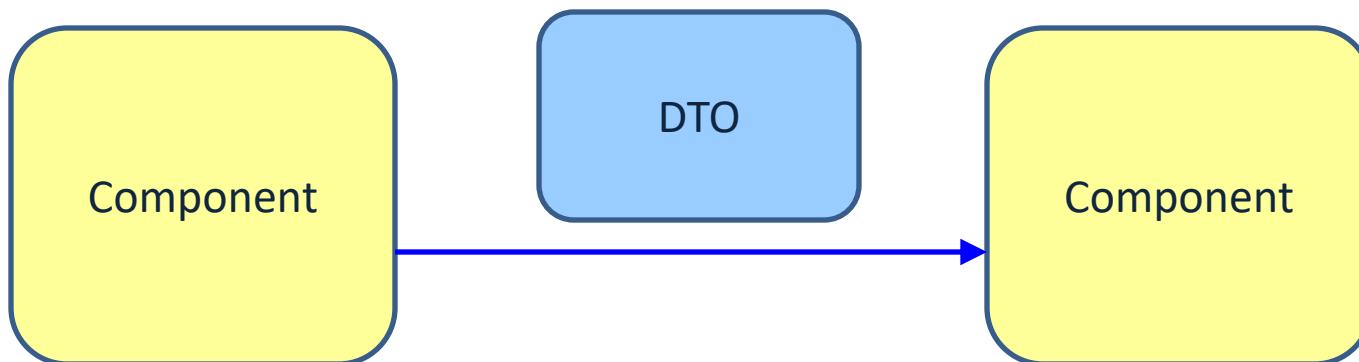


No shared data between components

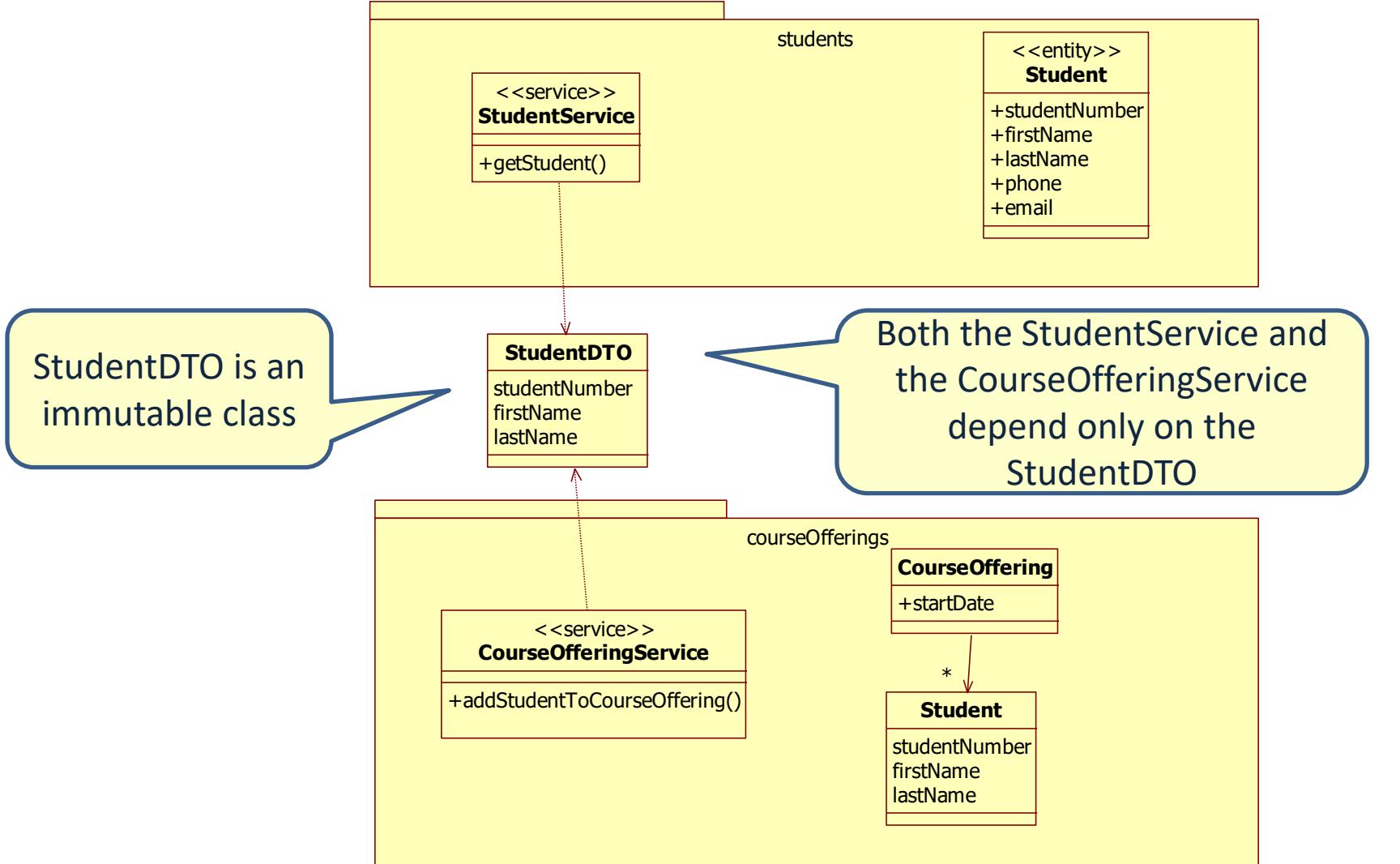


Data Transfer Objects (DTO)

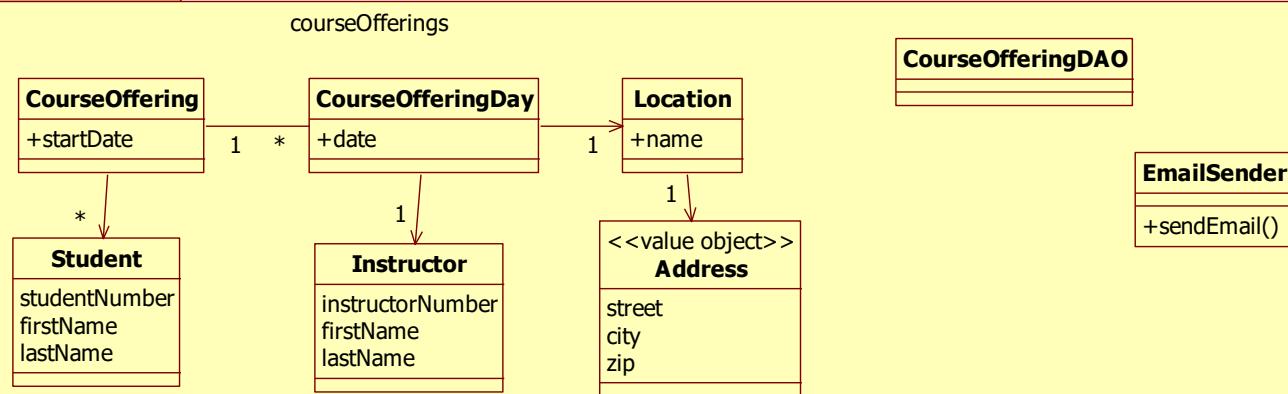
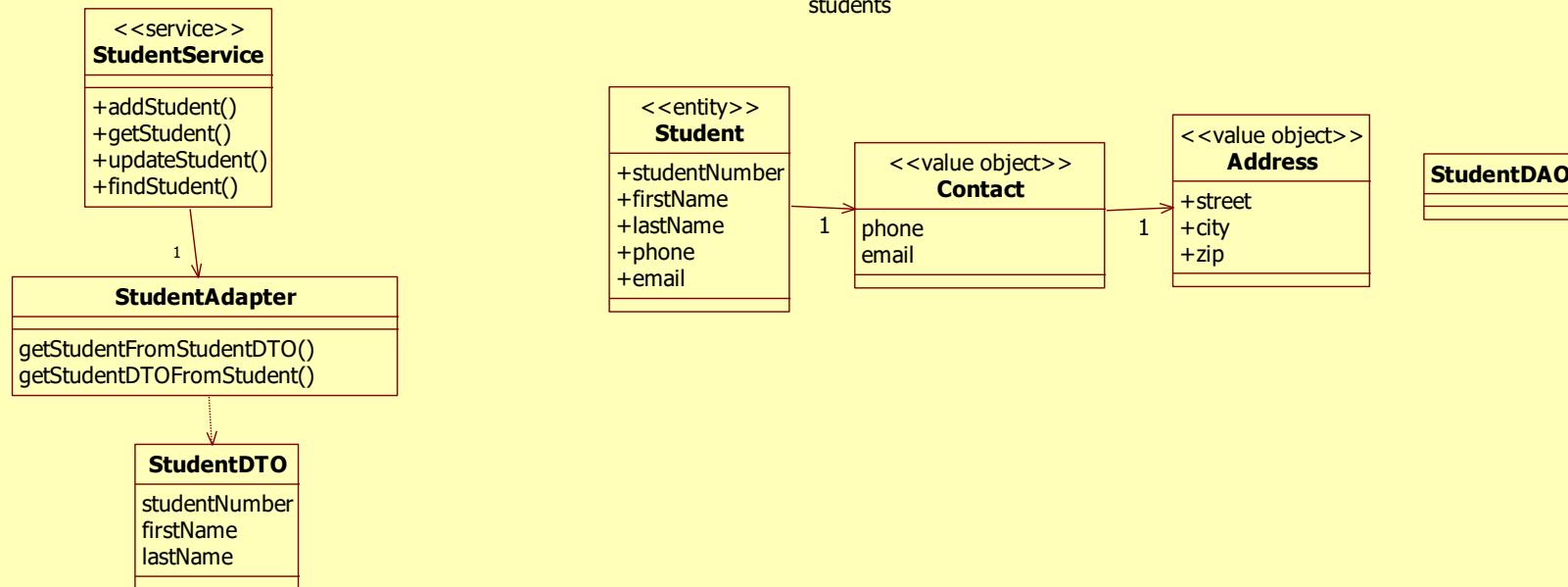
- Object that contains only attributes and getters and setters



Data Transfer Objects (DTO)

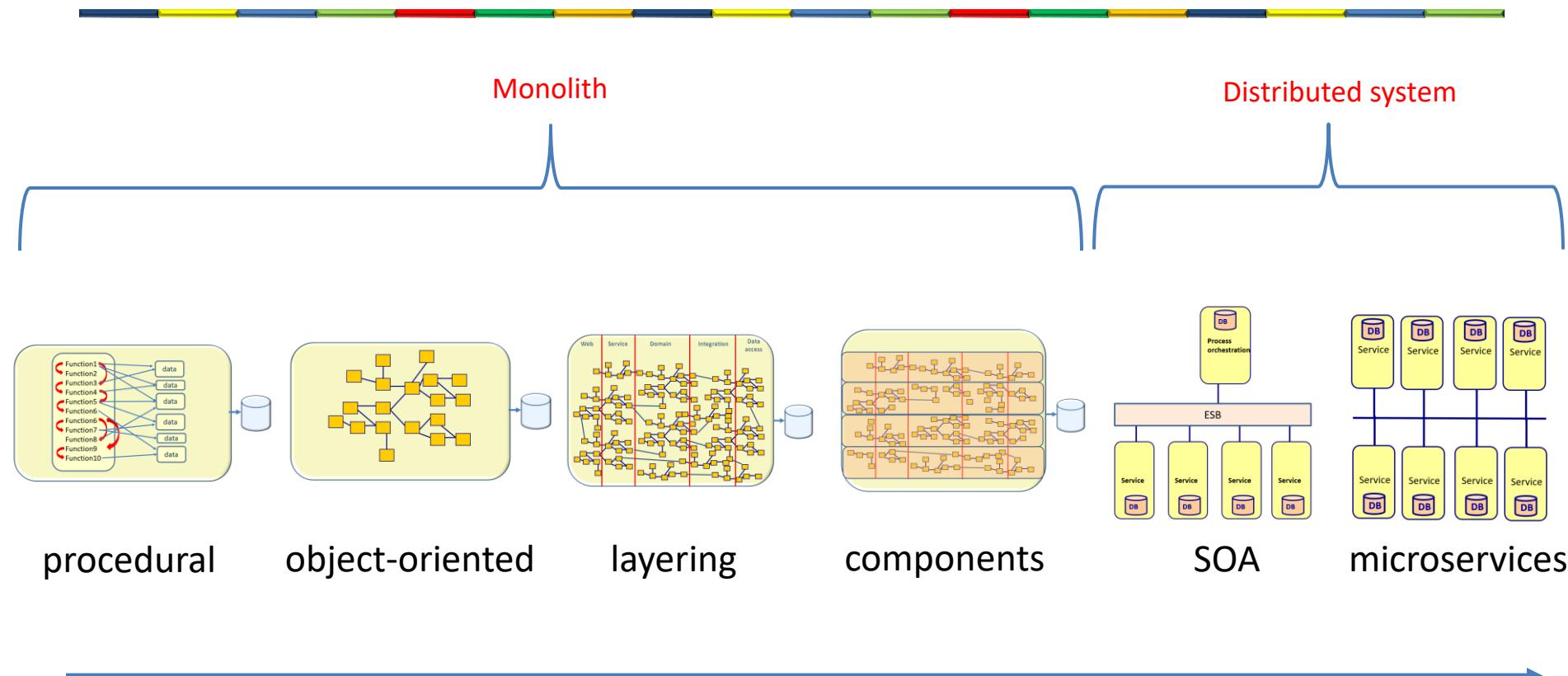


Data Transfer Objects (DTO)



SUMMARY

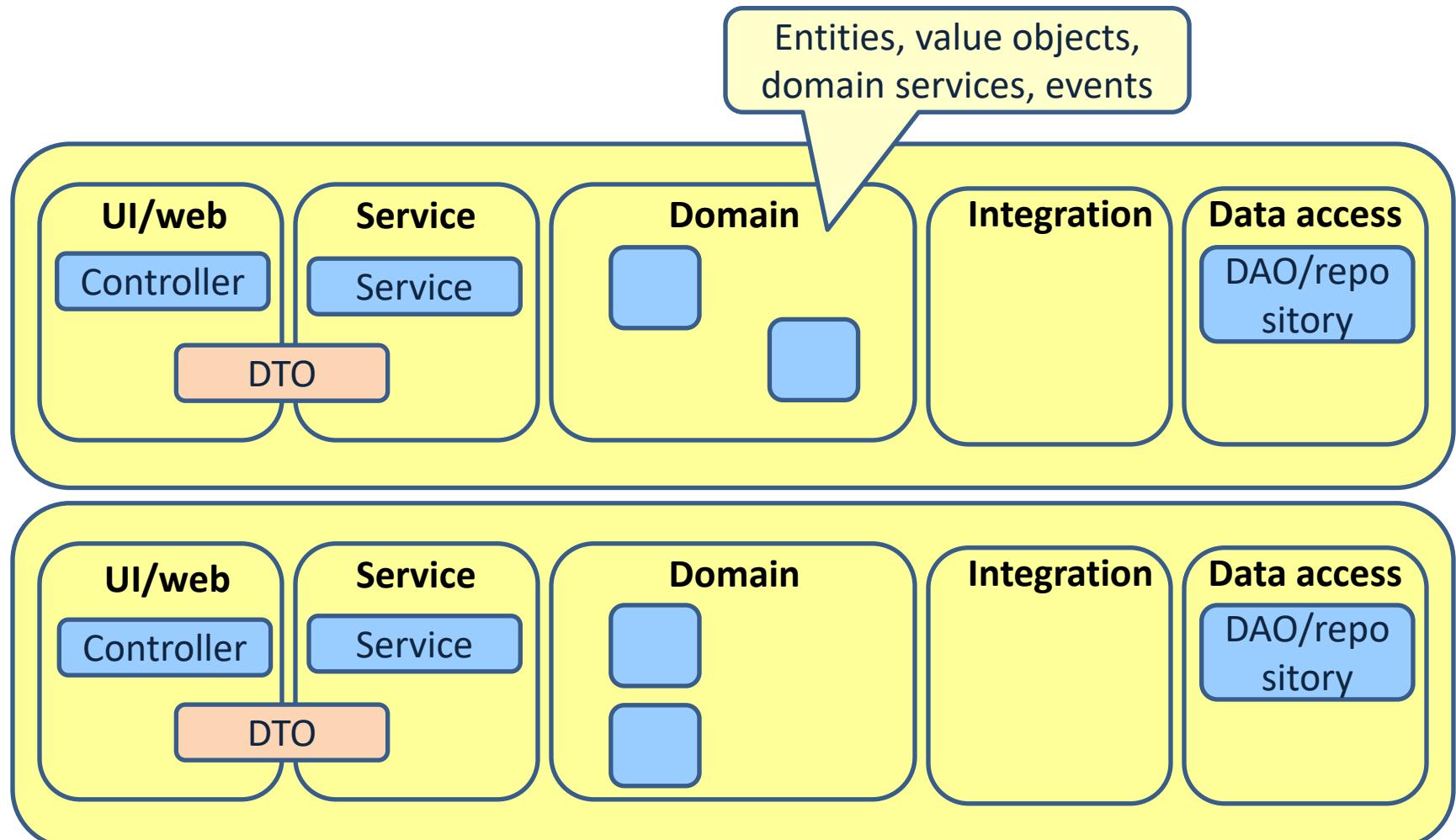
Architecture evolution



- Smaller and simpler parts
- More separation of concern
- More abstraction
- Less dependencies



Component design



Connecting the parts of knowledge with the wholeness of knowledge

1. A component is a loosely coupled module which is encapsulated and has one or more interfaces
 2. The only coupling between components is the interface with Data Transfer Objects
-

3. **Transcendental consciousness** is the source of all intelligence in nature.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that all components in creation are just expressions of ones own Self.

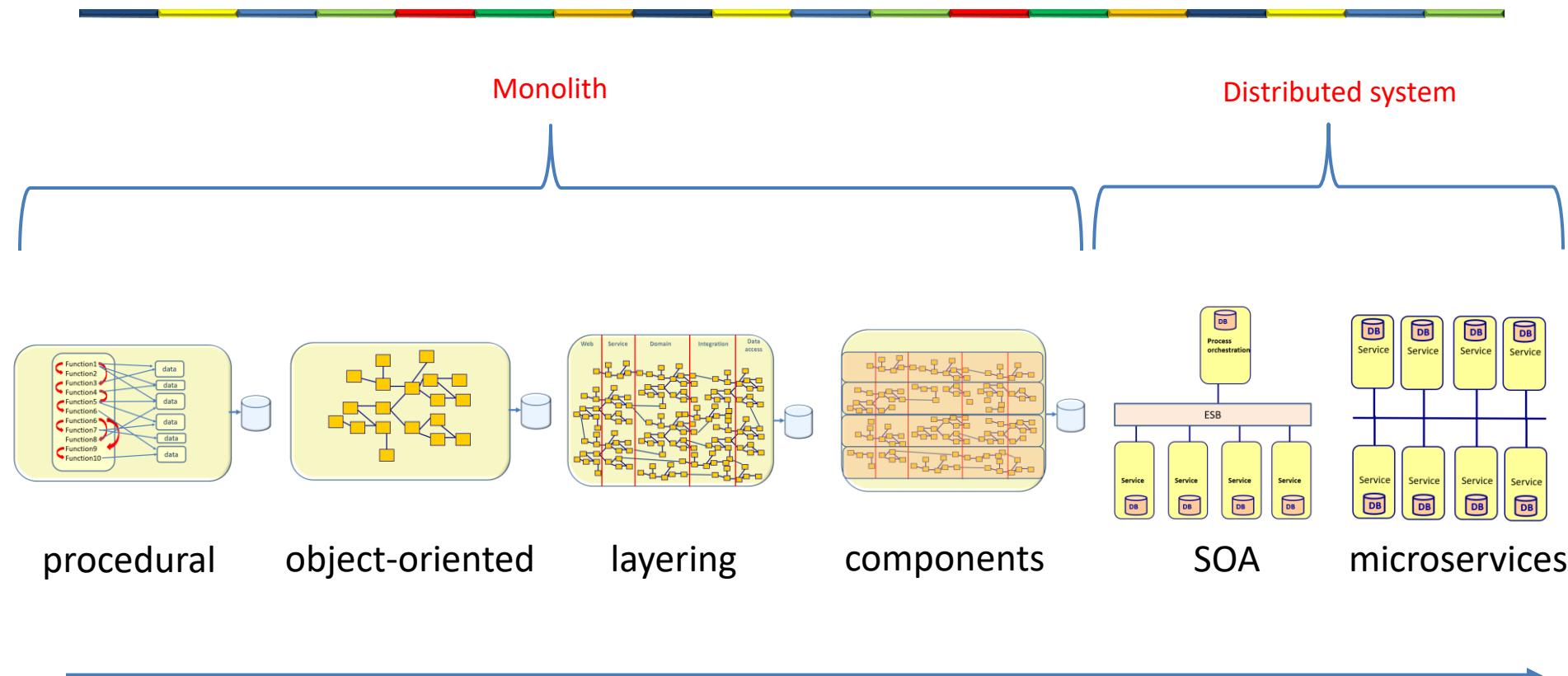


Lesson 5

SERVICE ORIENTED ARCHITECTURE INTEGRATION PATTERNS



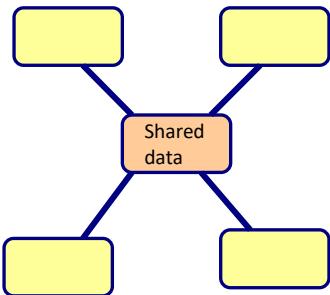
Architecture evolution



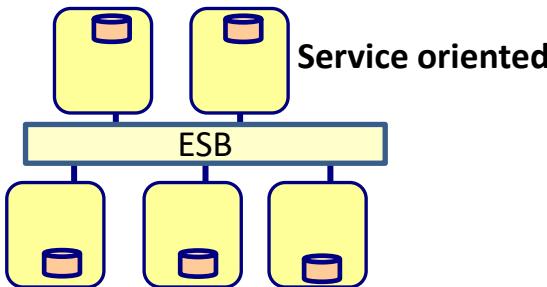
- Smaller and simpler parts
- More separation of concern
- More abstraction
- Less dependencies



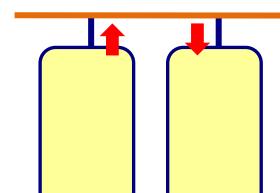
Architecture styles to connect applications



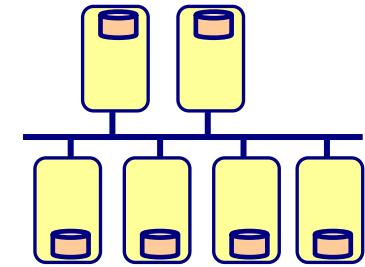
Blackboard



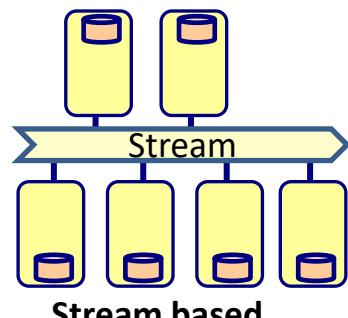
Service oriented



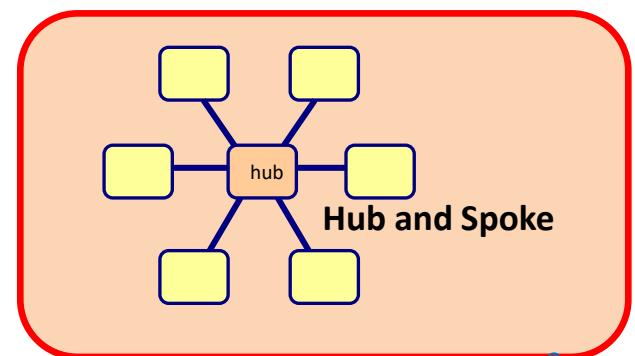
Event Driven



Microservices



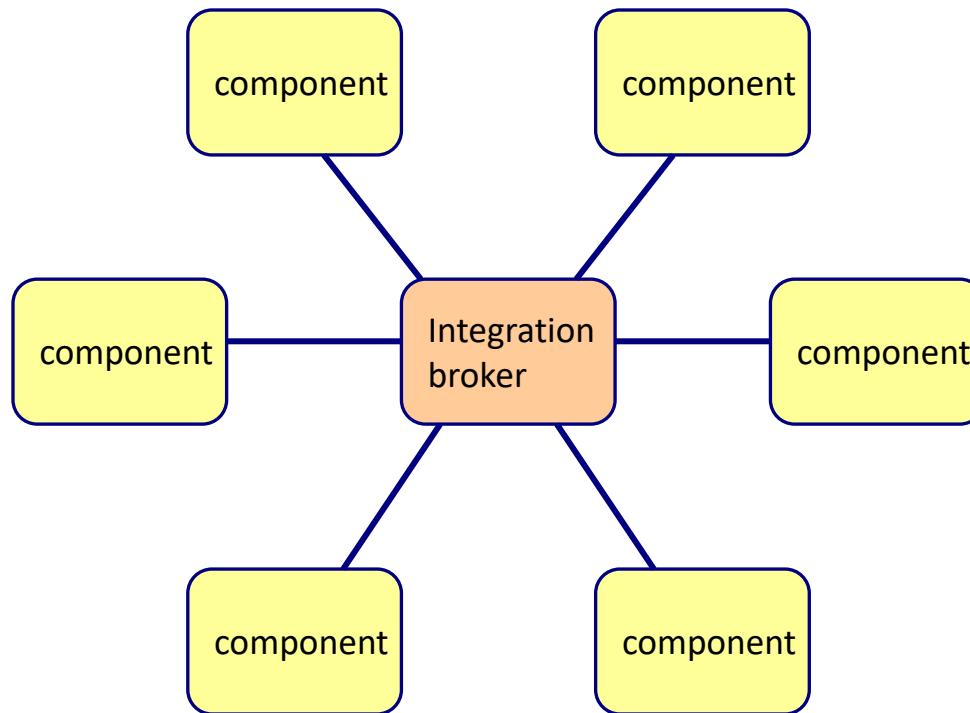
Stream based



Hub and Spoke

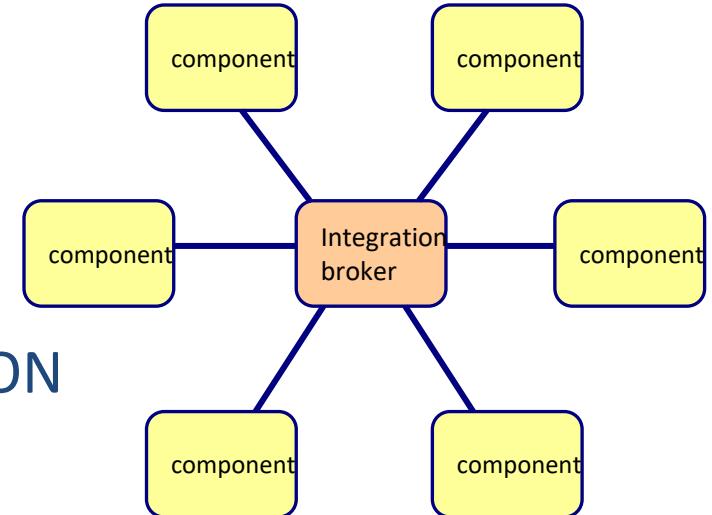
Hub and Spoke

- Integration broker



Hub and Spoke

- Functionality:
 - Transport
 - Transformation
 - For example from XML to JSON
 - Routing
 - Send the message to a component based on certain criteria (content based routing, load balancing, etc.)
 - Orchestration
 - The business process runs within the integration broker

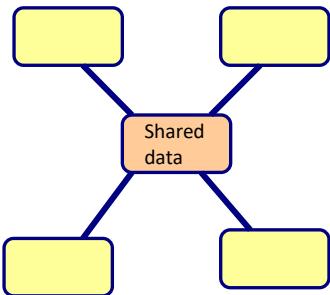


Hub and spoke

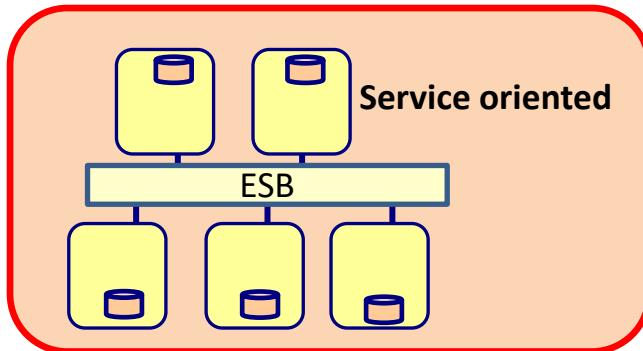
- Benefits
 - Separation of integration logic and application logic
 - Easy to add new components
 - Use adapters to plugin the integration broker
- Drawbacks
 - Single point of failure
 - Integration brokers are complex products
 - Integration broker becomes legacy itself



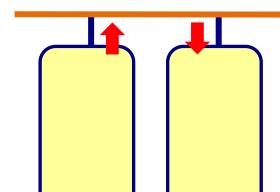
Architecture styles to connect applications



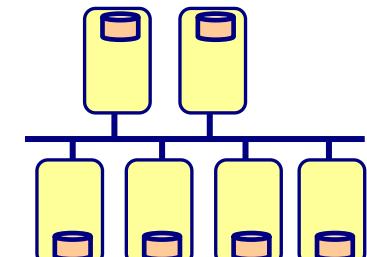
Blackboard



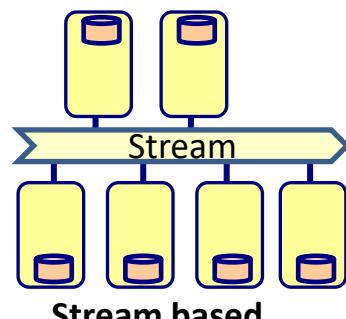
Service oriented



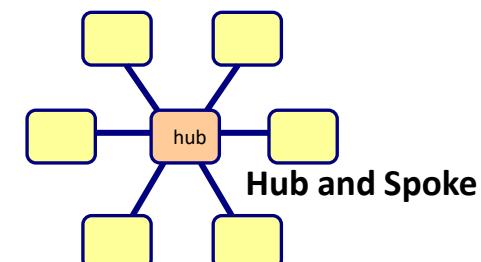
Event Driven



Microservices

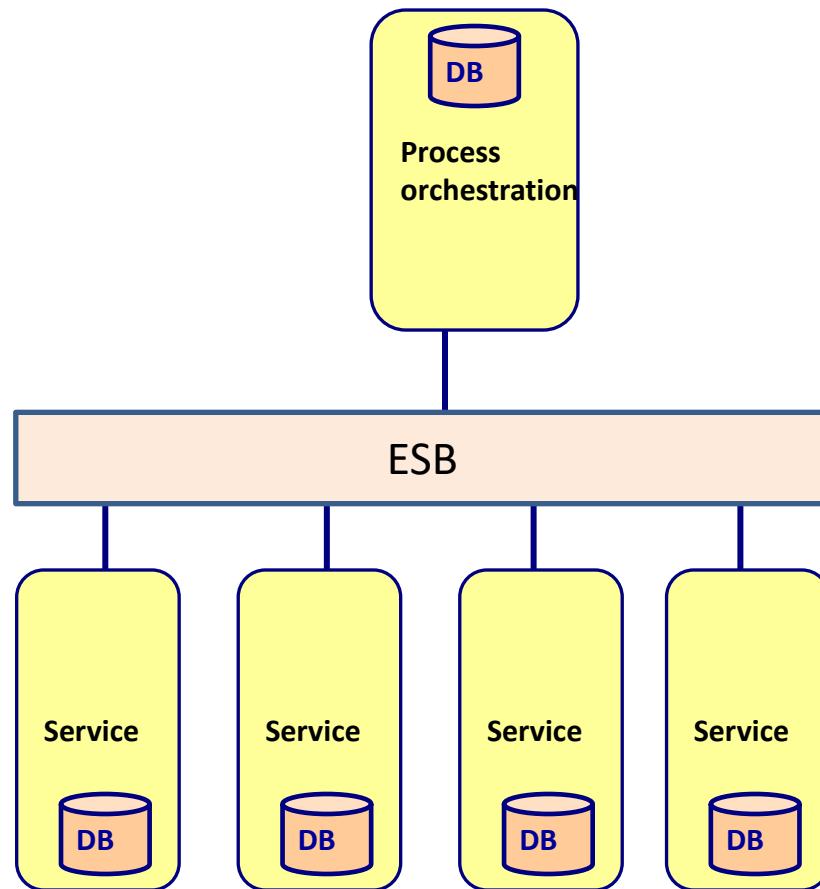


Stream based



Hub and Spoke

Service Oriented Architecture



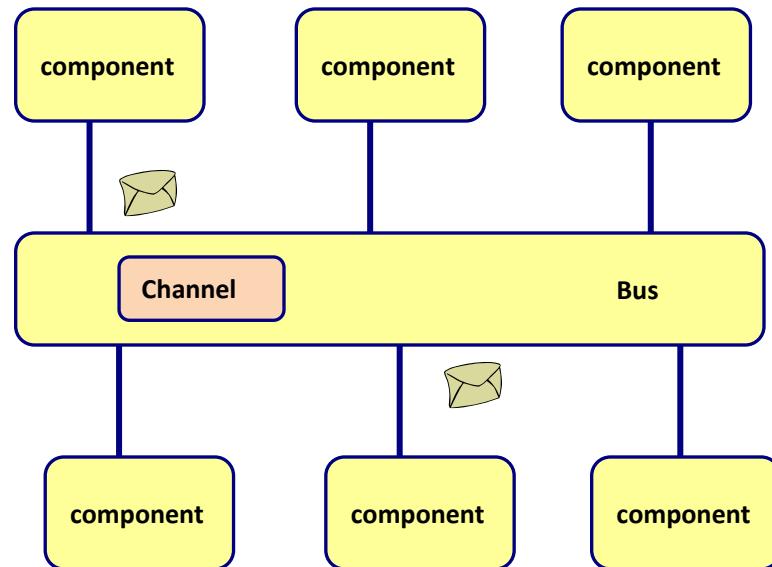
3 different aspects of a SOA

1. Communication through ESB
 - Standard protocols
2. Decompose the business domain in services
 - Often logical services
3. Make the business processes a 1st class citizen
 - Separate the business process from the application logic



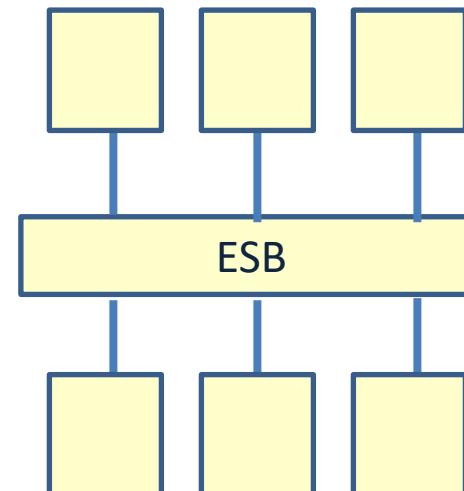
Responsibility of the bus

- Routing
 - Static
 - Content based
 - Rule based
 - Policy based
- Message transformation
- Message enhancing/filtering
- Protocol transformation
 - Input transformation
 - Output transformation
- Service mapping
 - Service name, protocol, binding variables, timeout, etc.
- Message processing
 - Guaranteed delivery
- Process choreography
 - Business process
 - BPEL
- Transaction management
- Security



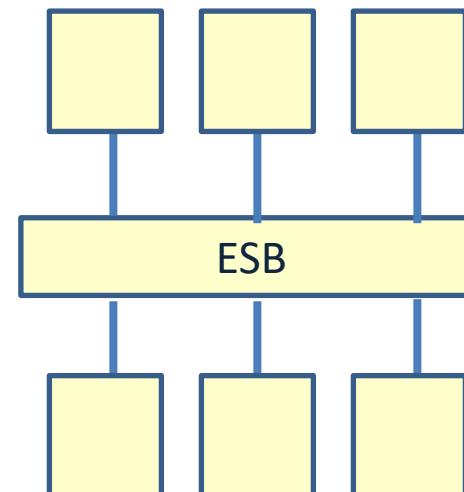
Service Oriented Architecture

- Advantages
 - Independent services
 - Easy to add new services
 - Separation of business processes and service logic
 - Architecture is optimized for the business
 - Reuse of services
 - Architecture flexibility



Service Oriented Architecture

- Disadvantages
 - Complex ESB
 - Changing the business process while business processes are still running is very difficult
 - Most SOA's are build on top of monoliths



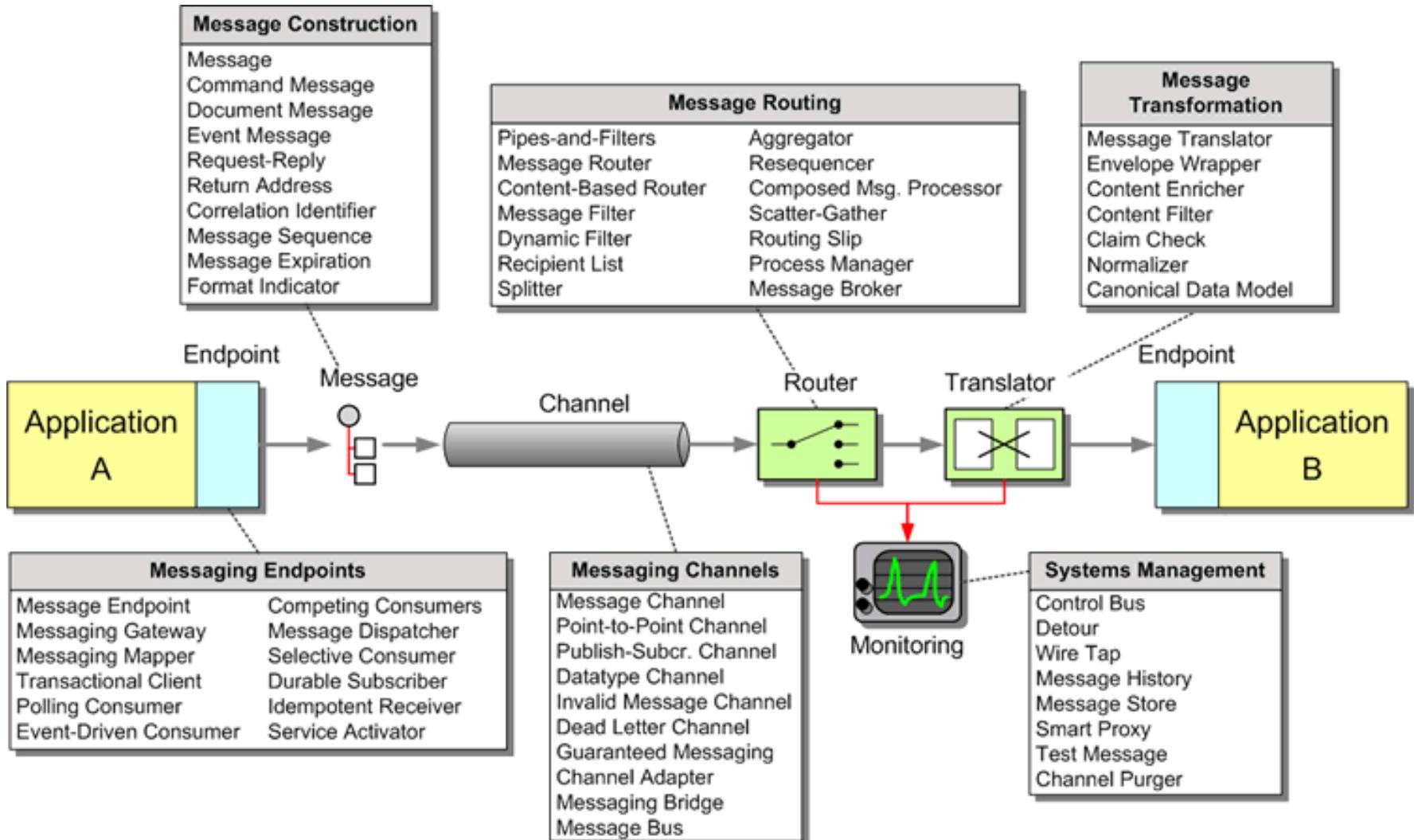
Main point

- In a service oriented architecture the business processes run on the ESB and they orchestrate all activity within the SOA architecture..
- The more you are in tune with the laws of nature, the more support of nature you are able to enjoy.

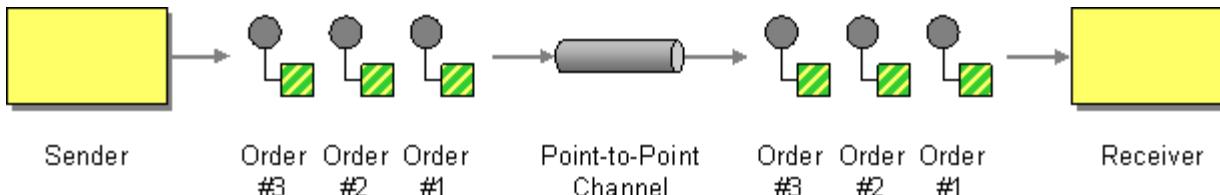


ENTERPRISE INTEGRATION PATTERNS

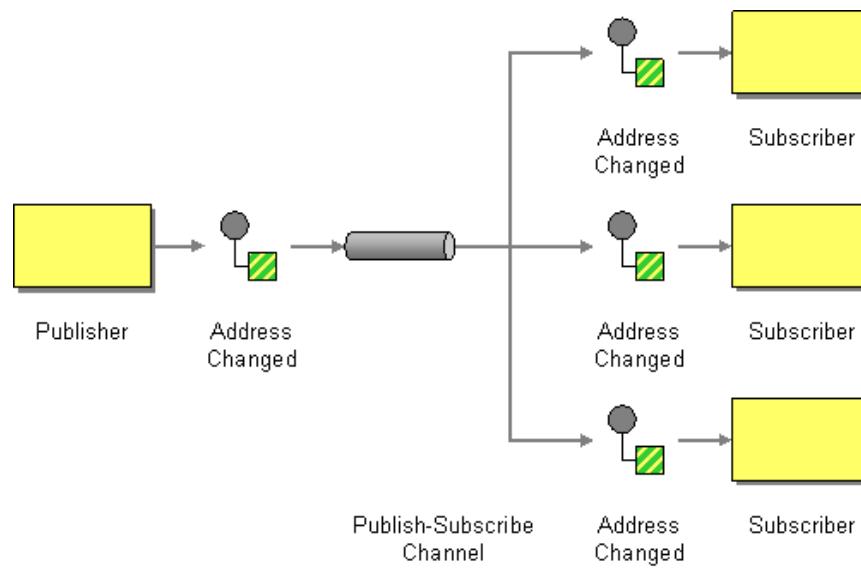
Enterprise Integration Patterns



Messaging channel patterns



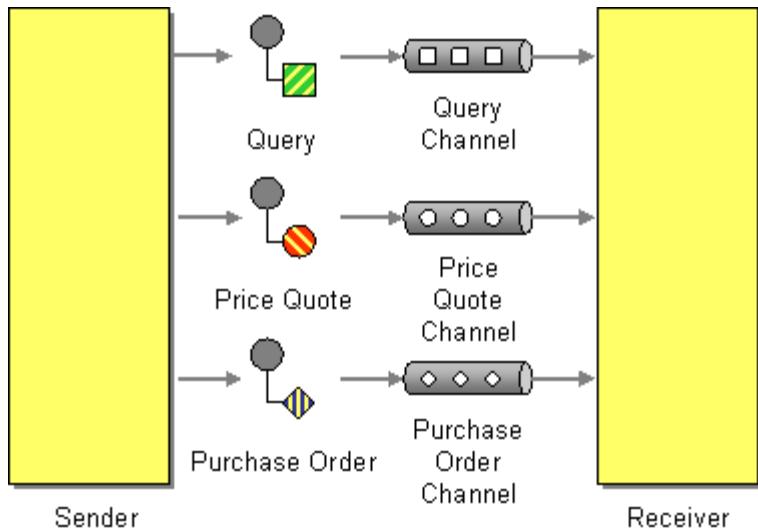
Point-to-point: only one receiver will receive the message



Publish-Subscribe : every subscriber will receive the message



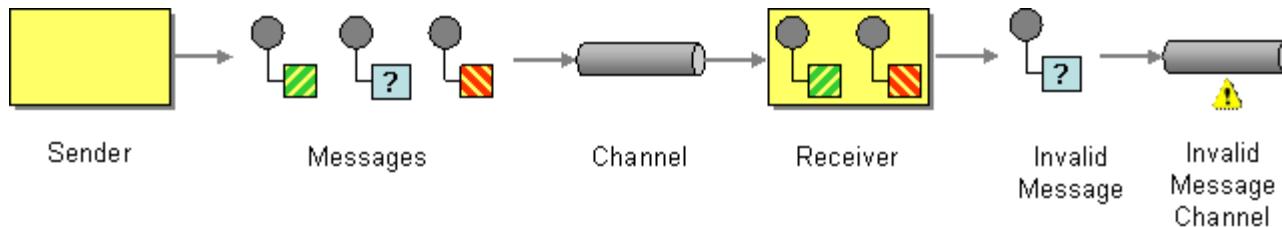
Messaging channel patterns



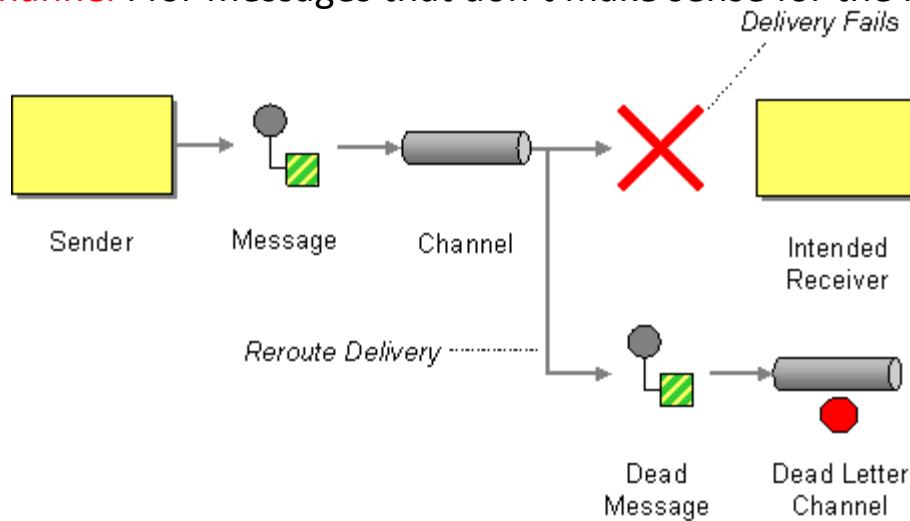
Datatype Channel : use a channel for each data type, so that the receiver know how to process it



Messaging channel patterns



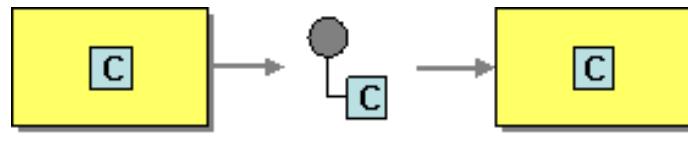
Invalid Message Channel : for messages that don't make sense for the receiver



Dead Letter Channel : for messages that can't be delivered



Message construction patterns



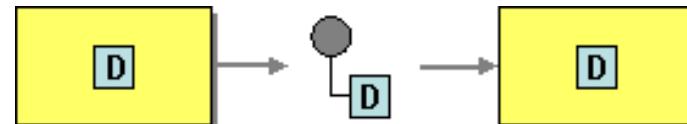
Sender

Command
Message

Receiver

C = getLastTradePrice("DIS");

Command message



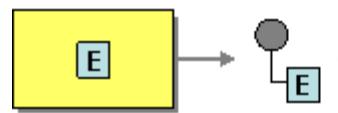
Sender

Document
Message

Receiver

D = aPurchaseOrder **Document message**

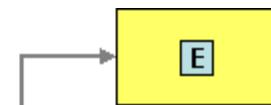
Event message



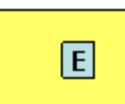
Subject

Event
Message

E = aPriceChangedEvent



Observer

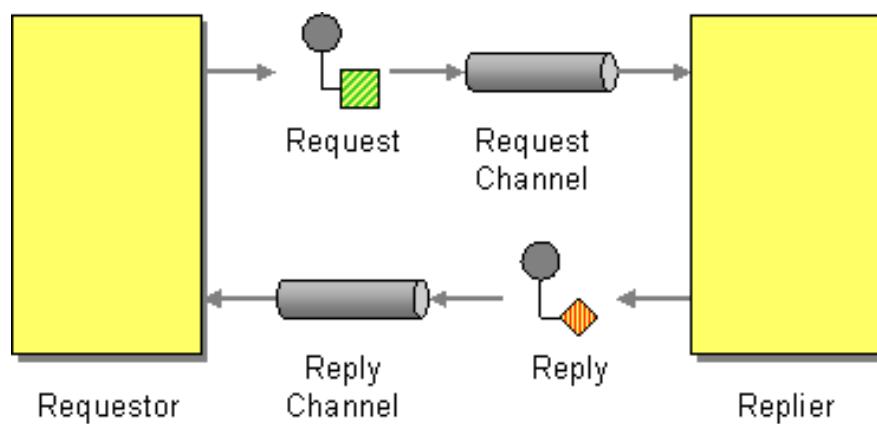


Observer

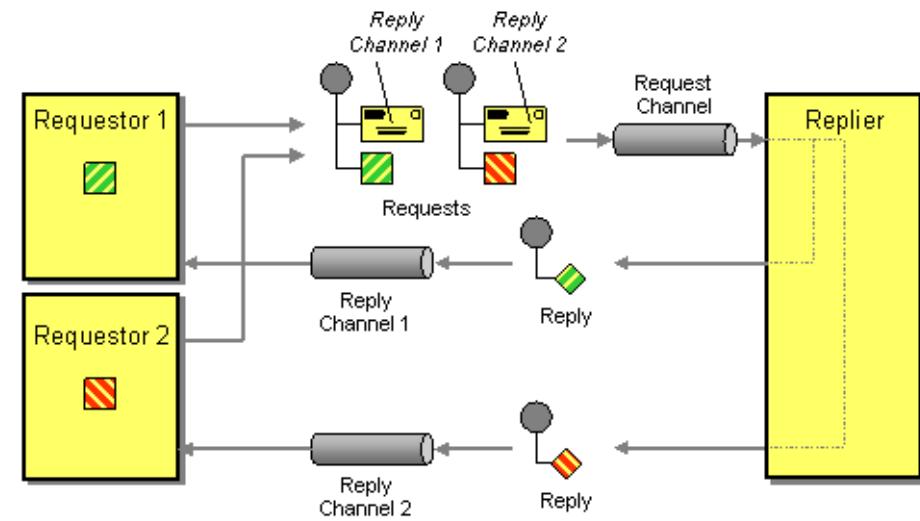


Observer

Message construction patterns



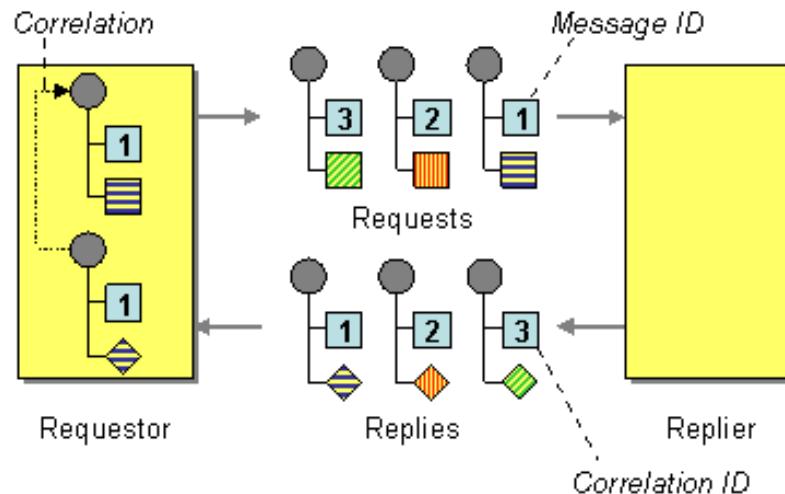
Request-Reply



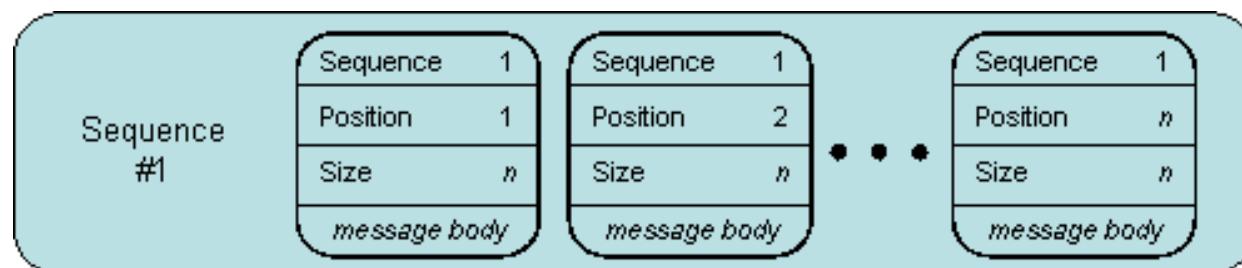
Return address



Message construction patterns

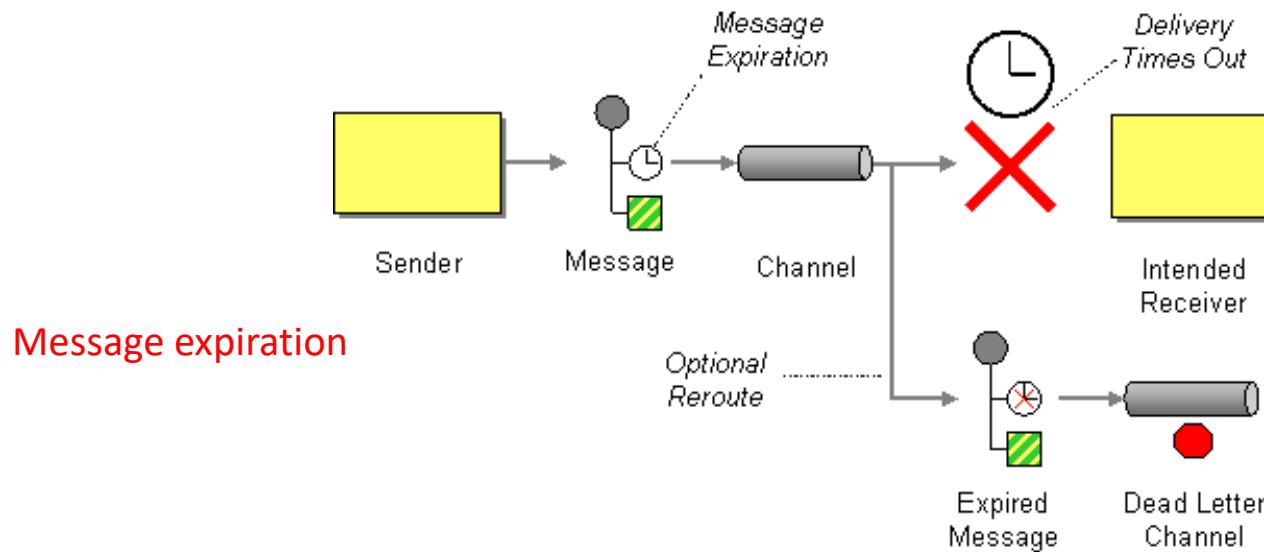


Each reply message should contain a **Correlation Identifier**, a unique identifier that indicates which request message this reply is for



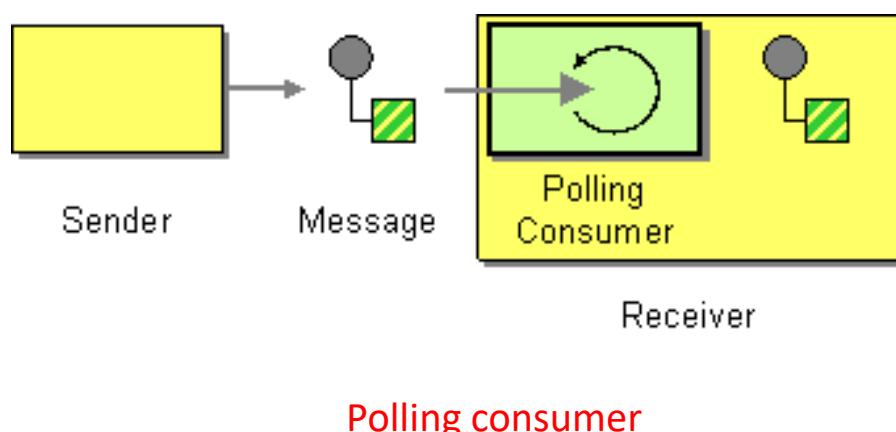
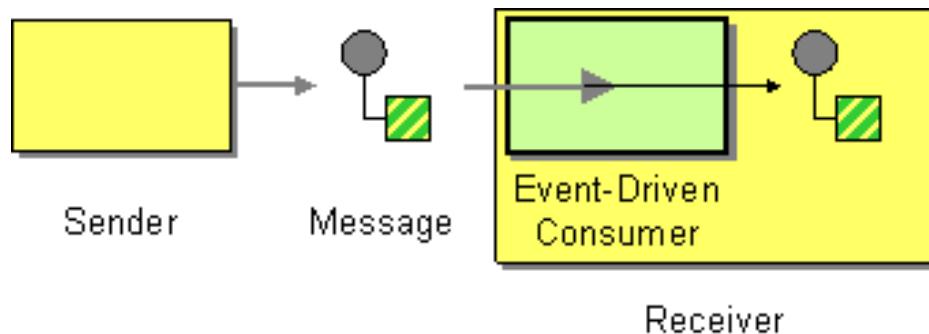
Whenever a large set of data may need to be broken into message-size chunks, send the data as a **Message Sequence** and mark each message with sequence identification fields.

Message construction patterns

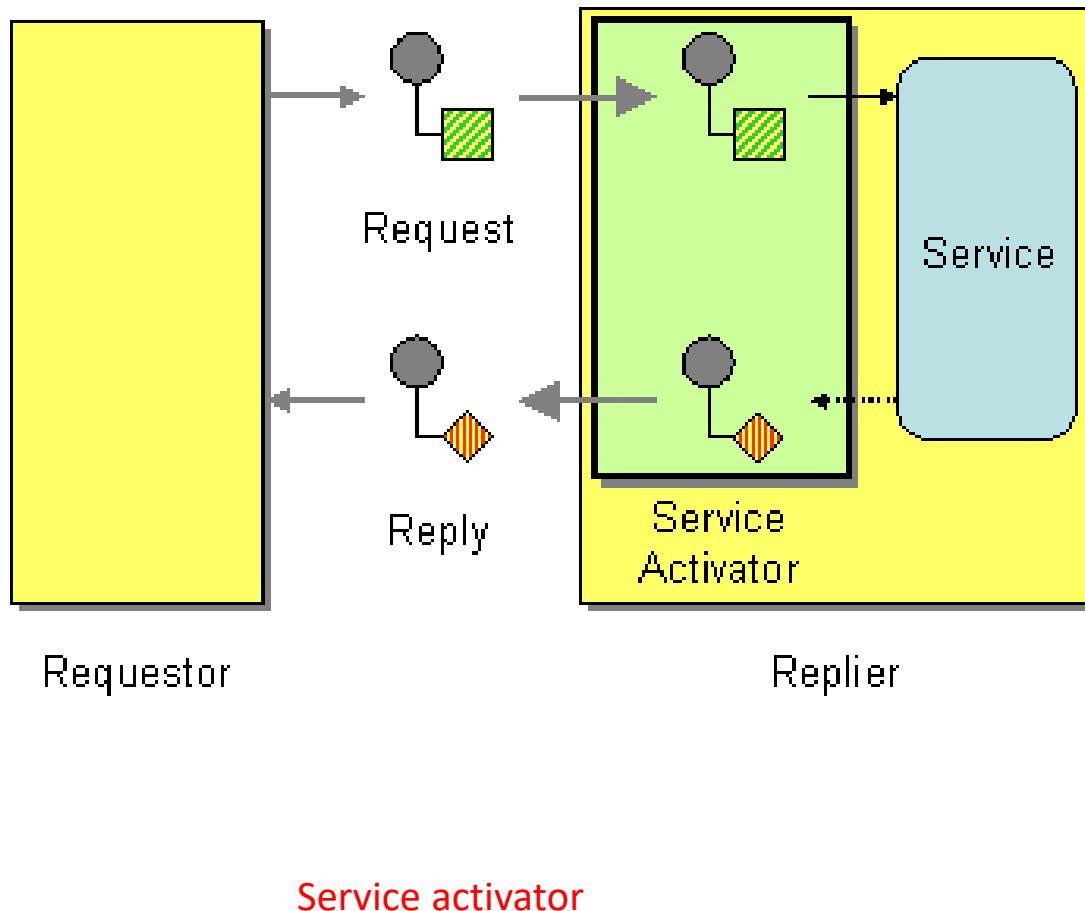


Message expiration

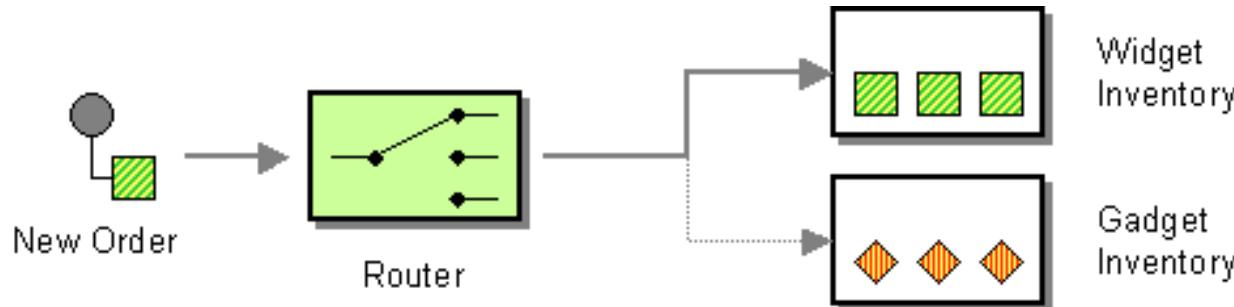
Message Endpoint



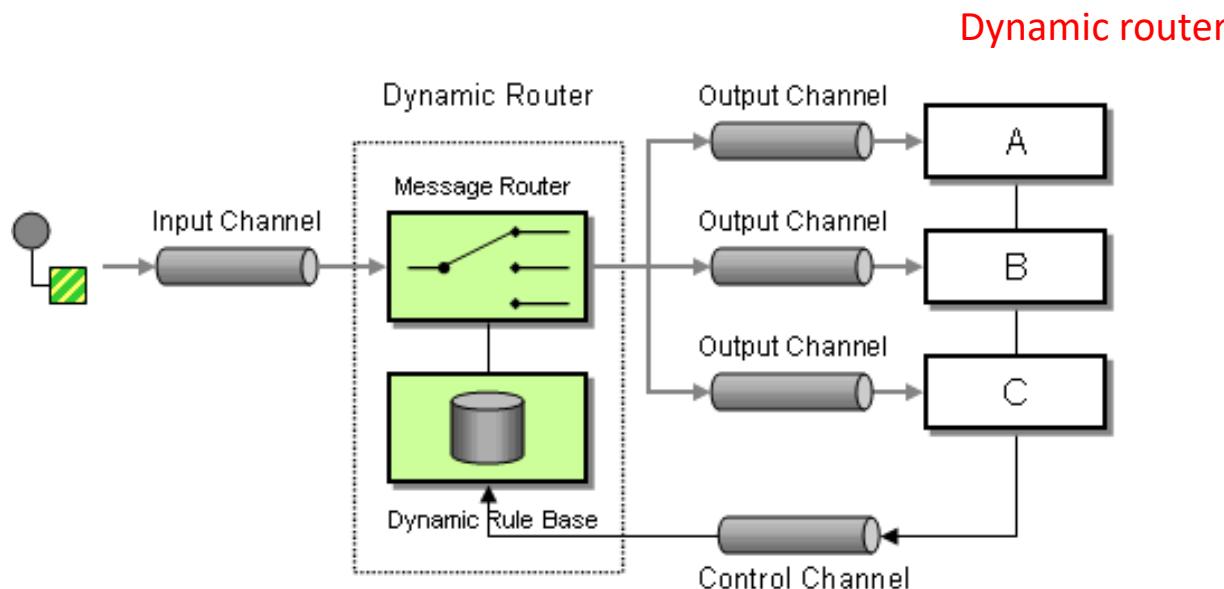
Message Endpoint



Message Routing

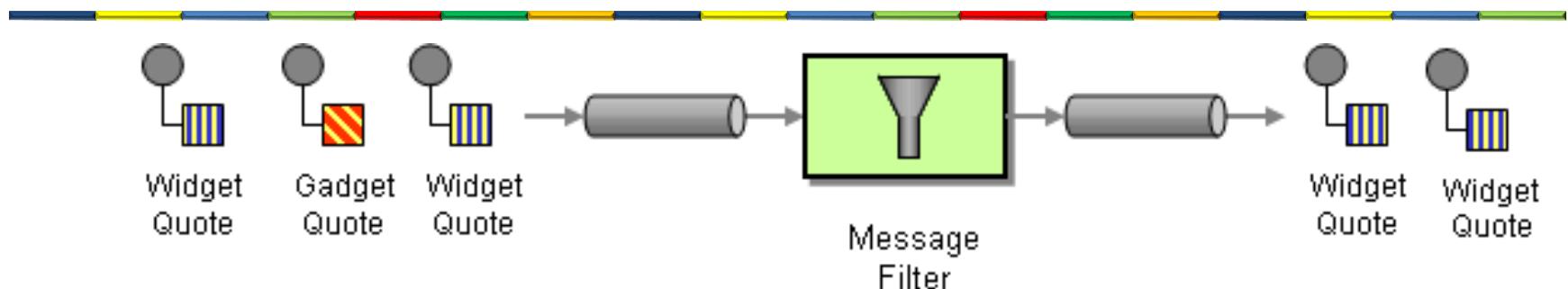


Content based router



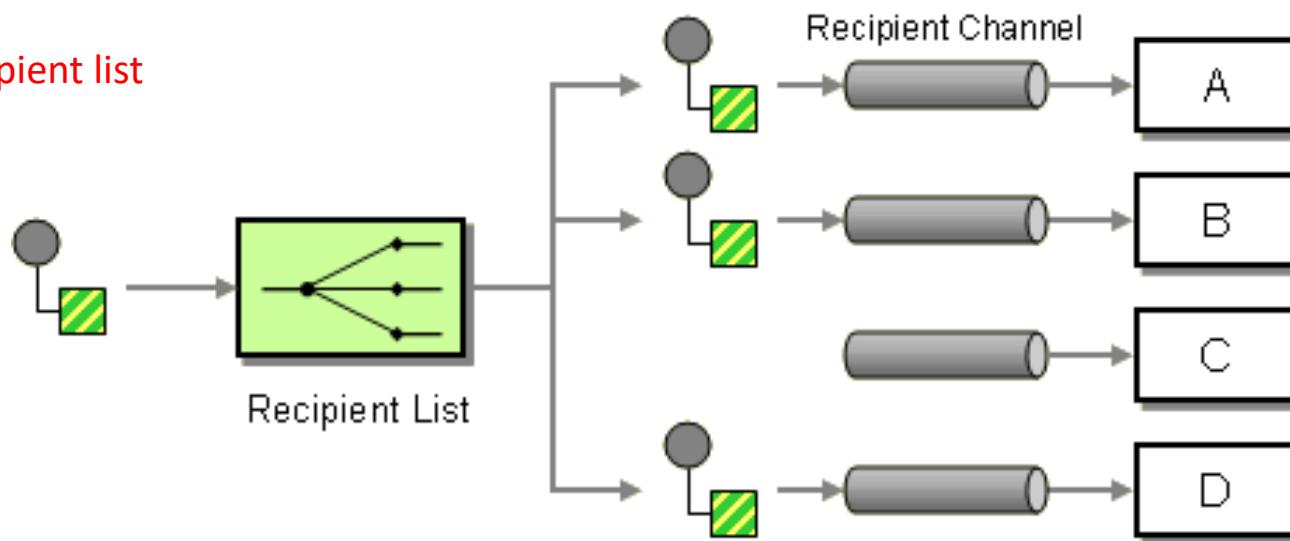
Dynamic router

Message Routing

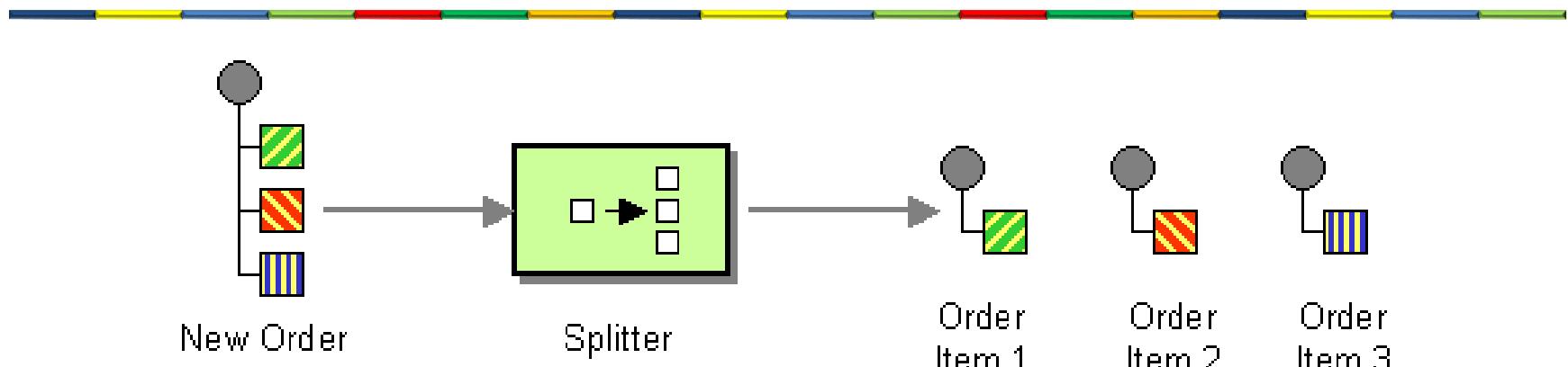


Message filter

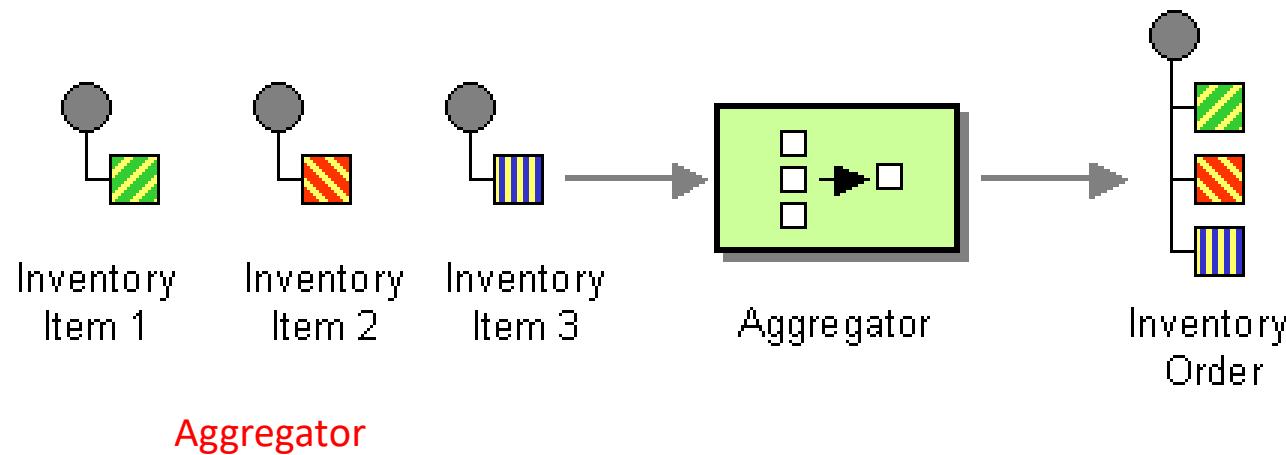
Recipient list



Message Routing



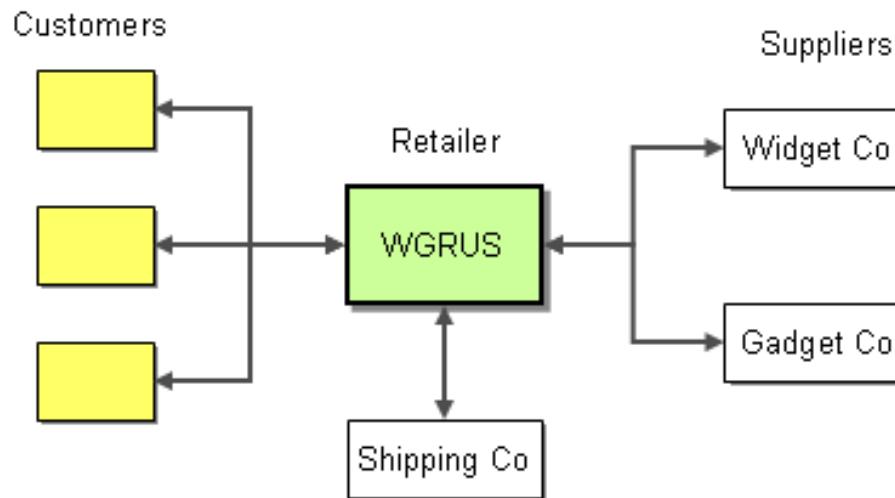
Splitter



Aggregator

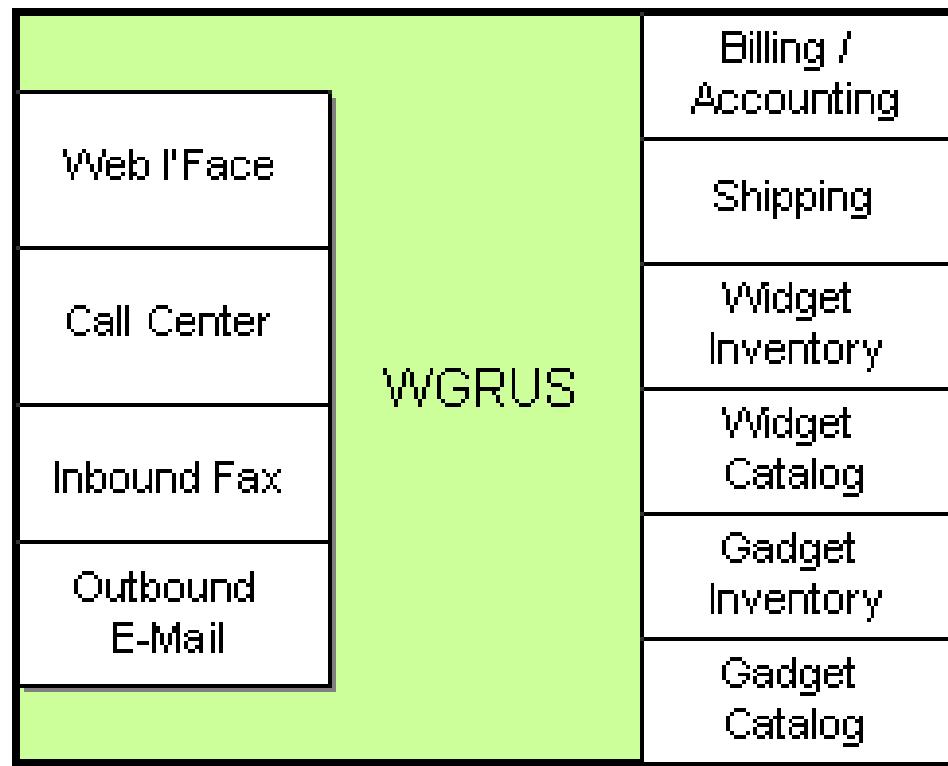


Example: Widgets&Gatchets 'R Us (WGRUS)

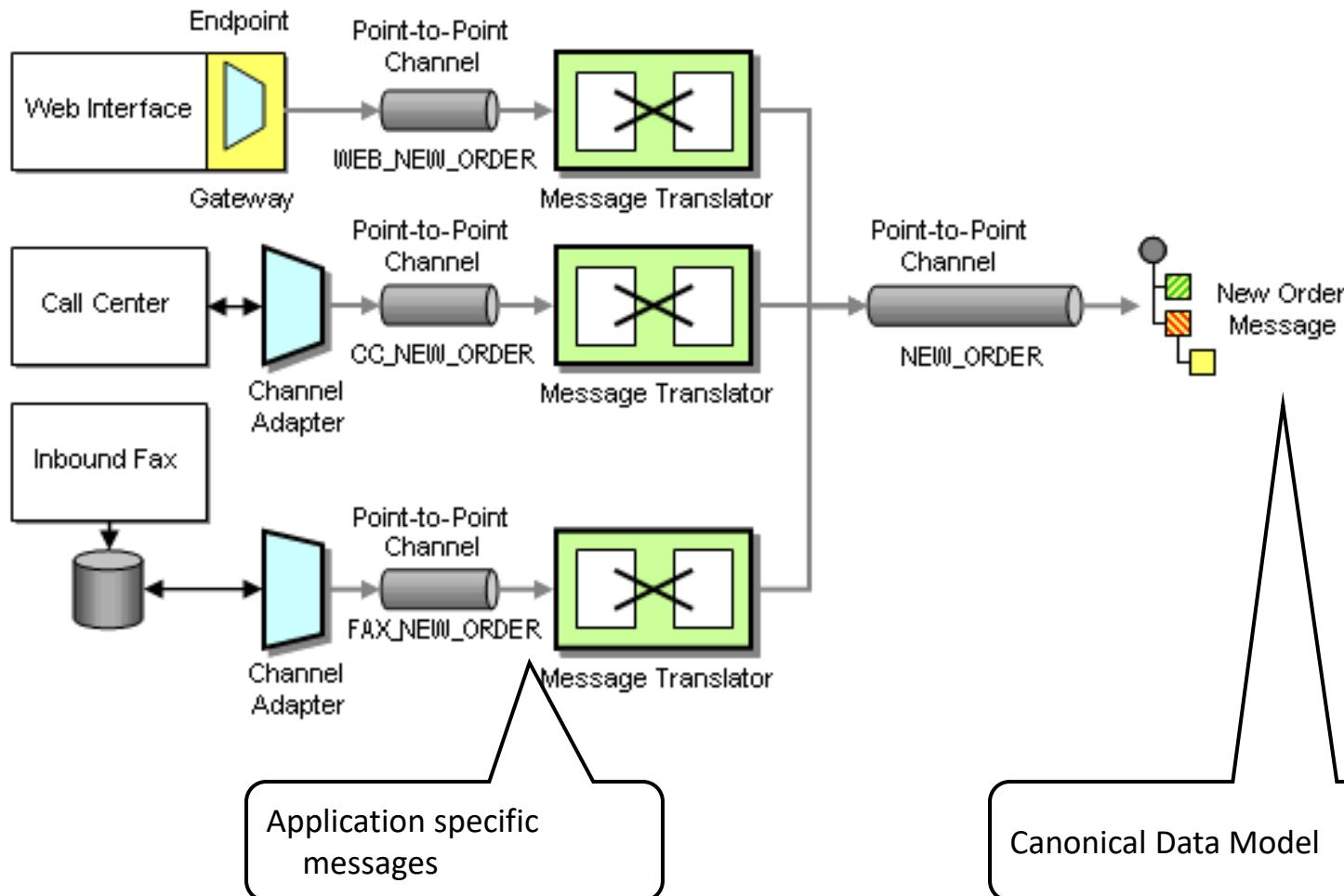


- **Take Orders:** Customers can place orders via Web, phone or fax
- **Process Orders:** Processing an order involves multiple steps, including verifying inventory, shipping the goods and invoicing the customer
- **Check Status:** Customers can check the order status
- **Change Address:** Customers can use a Web front-end to change their billing and shipping address
- **New Catalog:** The suppliers update their catalog periodically. WGRUS needs to update its pricing and availability based in the new catalogs.
- **Announcements:** Customers can subscribe to selective announcements from WGRUS.
- **Testing and Monitoring:** The operations staff needs to be able to monitor all individual components and the message flow between them.

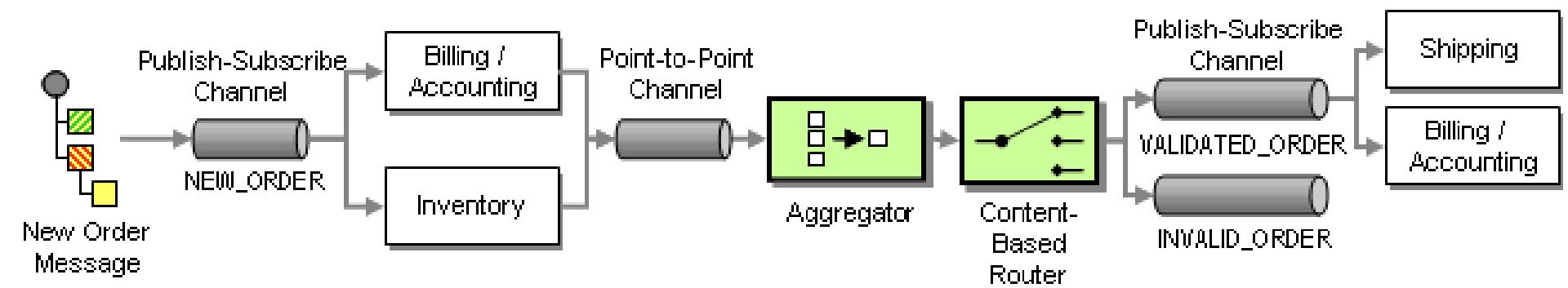
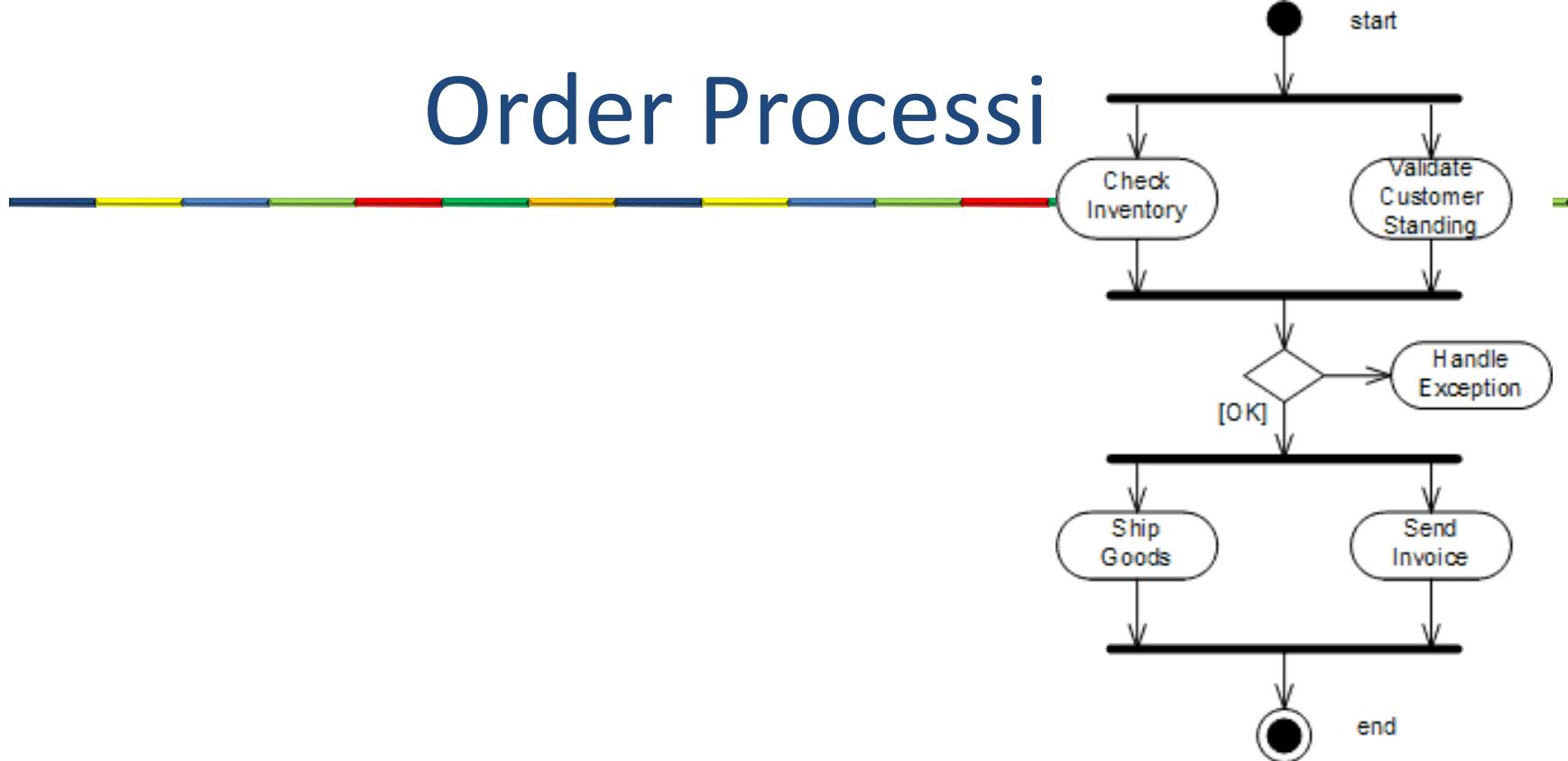
WGRUS internal IT infrastructure



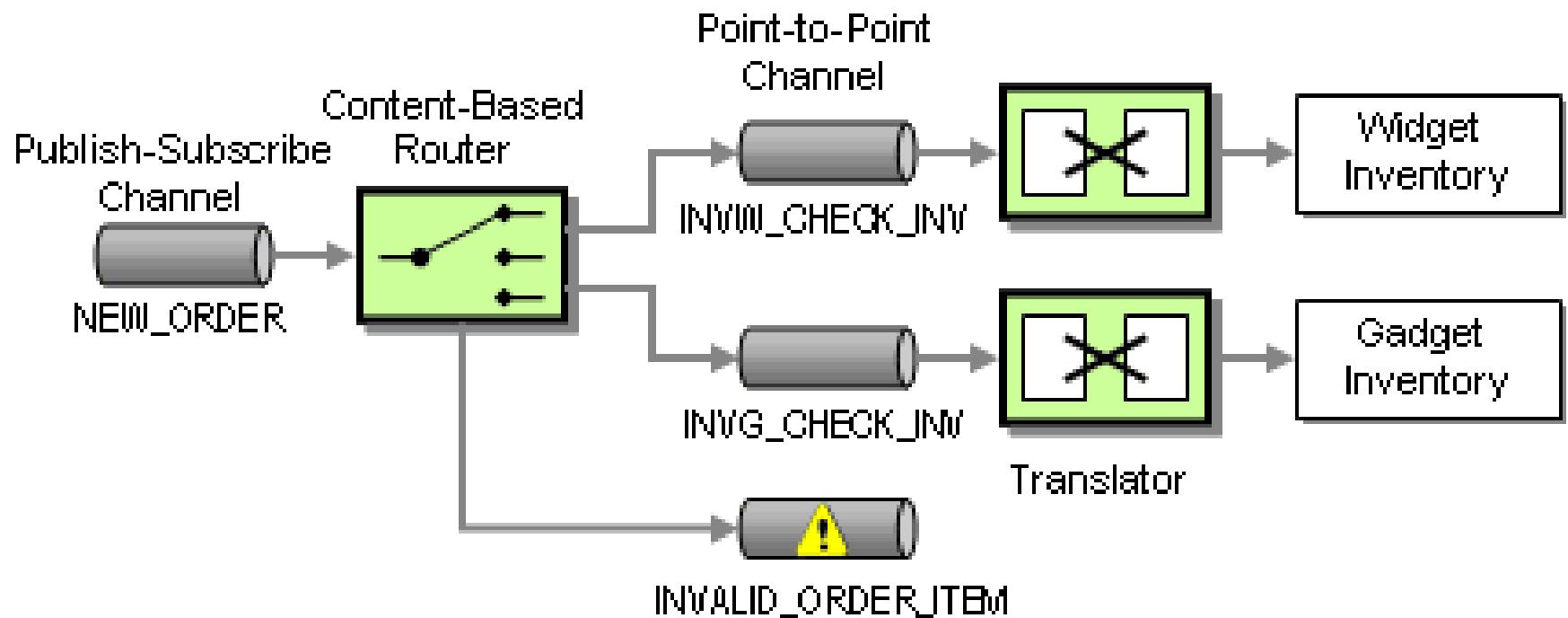
Taking orders from 3 different channels



Order Process

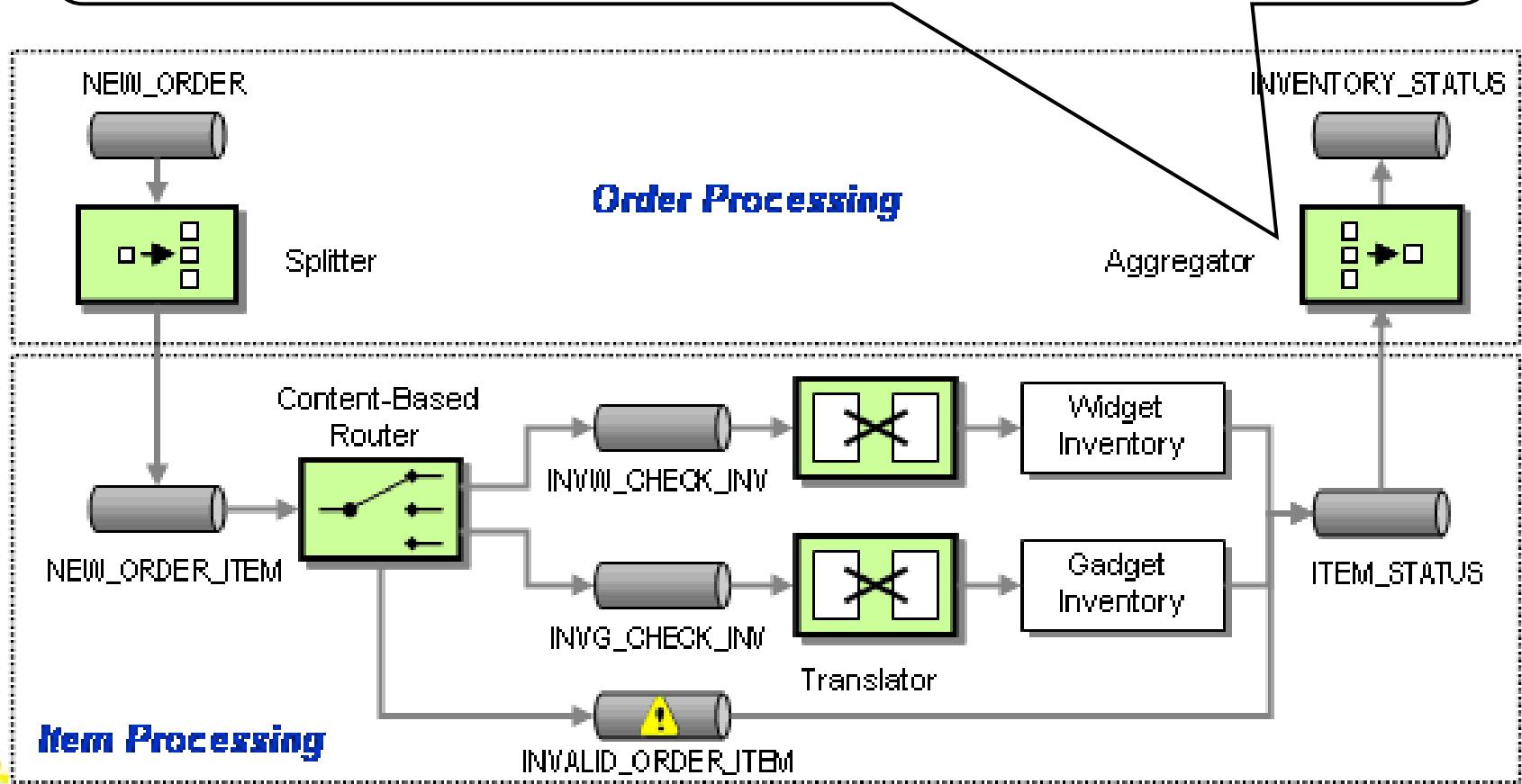


Routing the inventory request

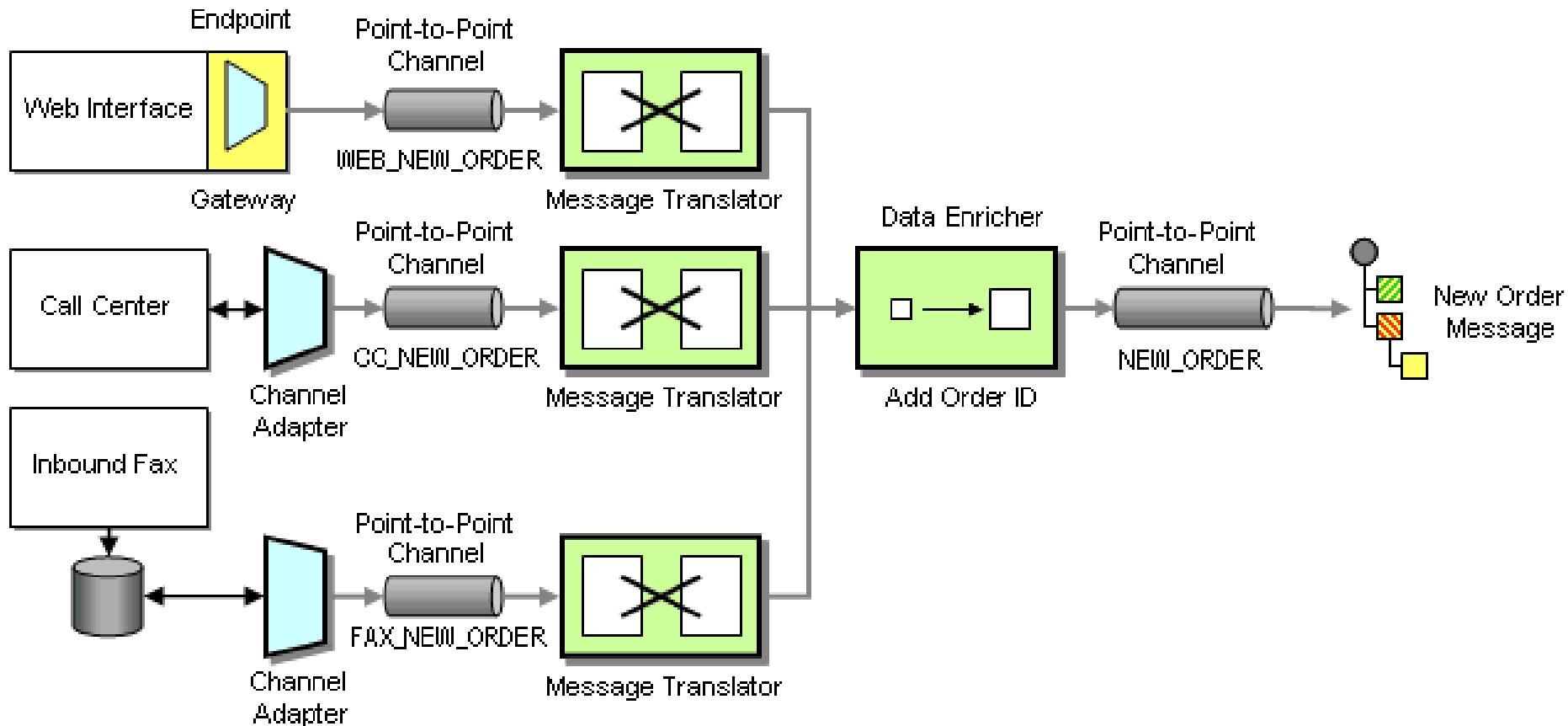


Orders can contain multiple items

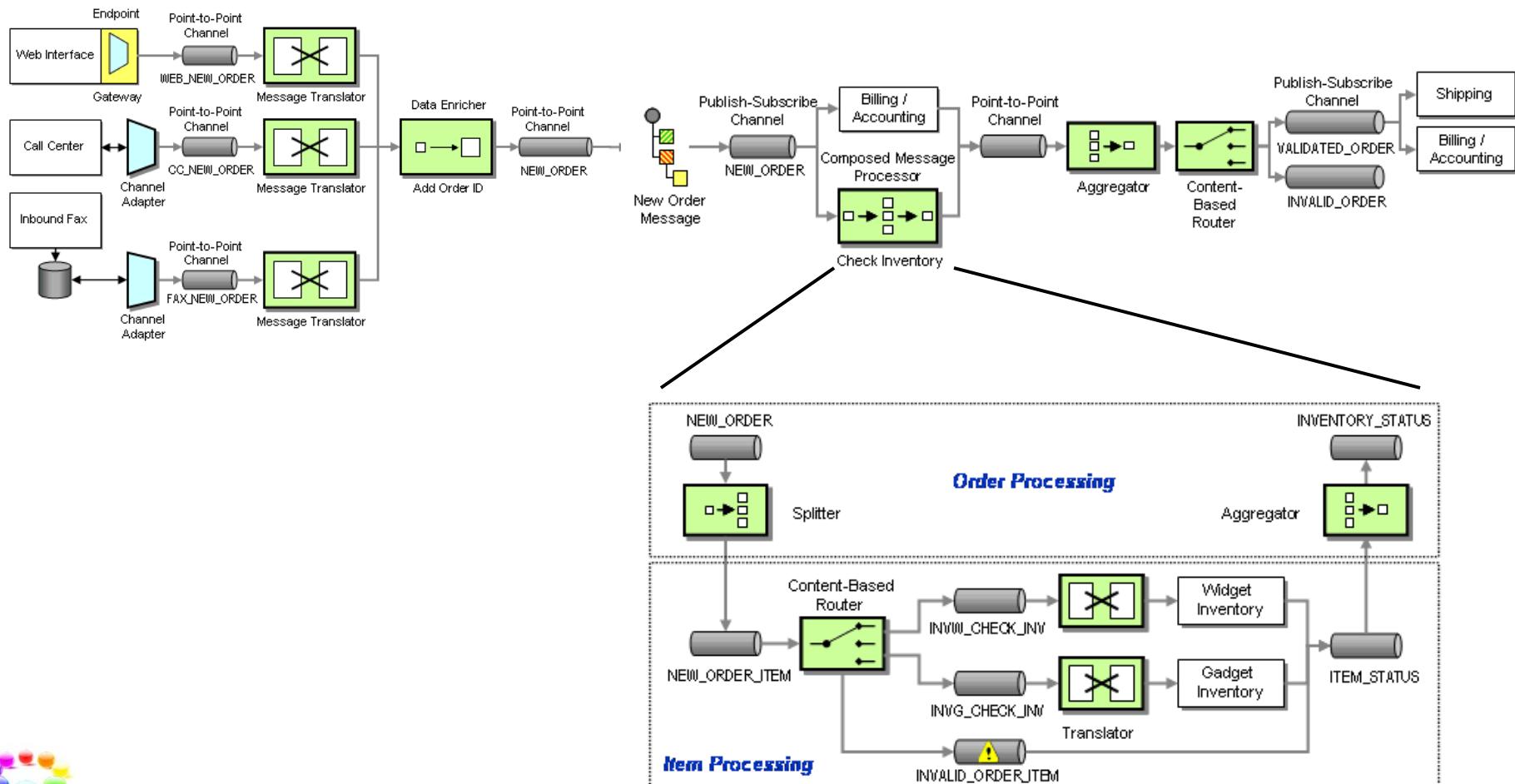
1. Correlation: which messages belong together? **We need an unique order ID**
2. Completeness: how do we know that all messages are received? **Count**
3. Aggregation algorithm: how do we combine the individual messages into one result message?
Append based on order ID



Add an unique order ID



Result so far



Main point

- There are many different integration patterns that you can use to implement the integration logic between components and systems.
- The unified field is the field of all possibilities.

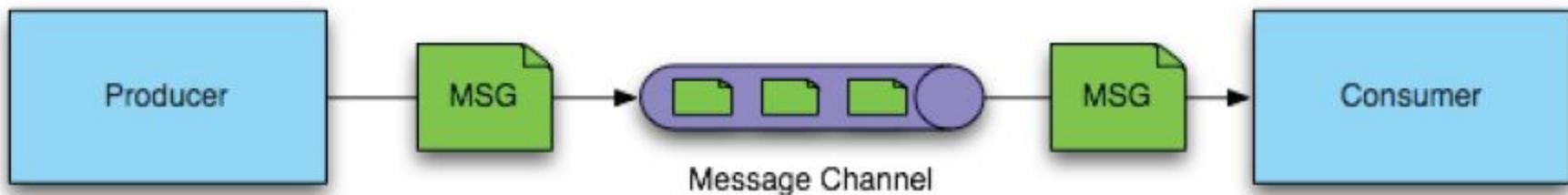


SPRING INTEGRATION

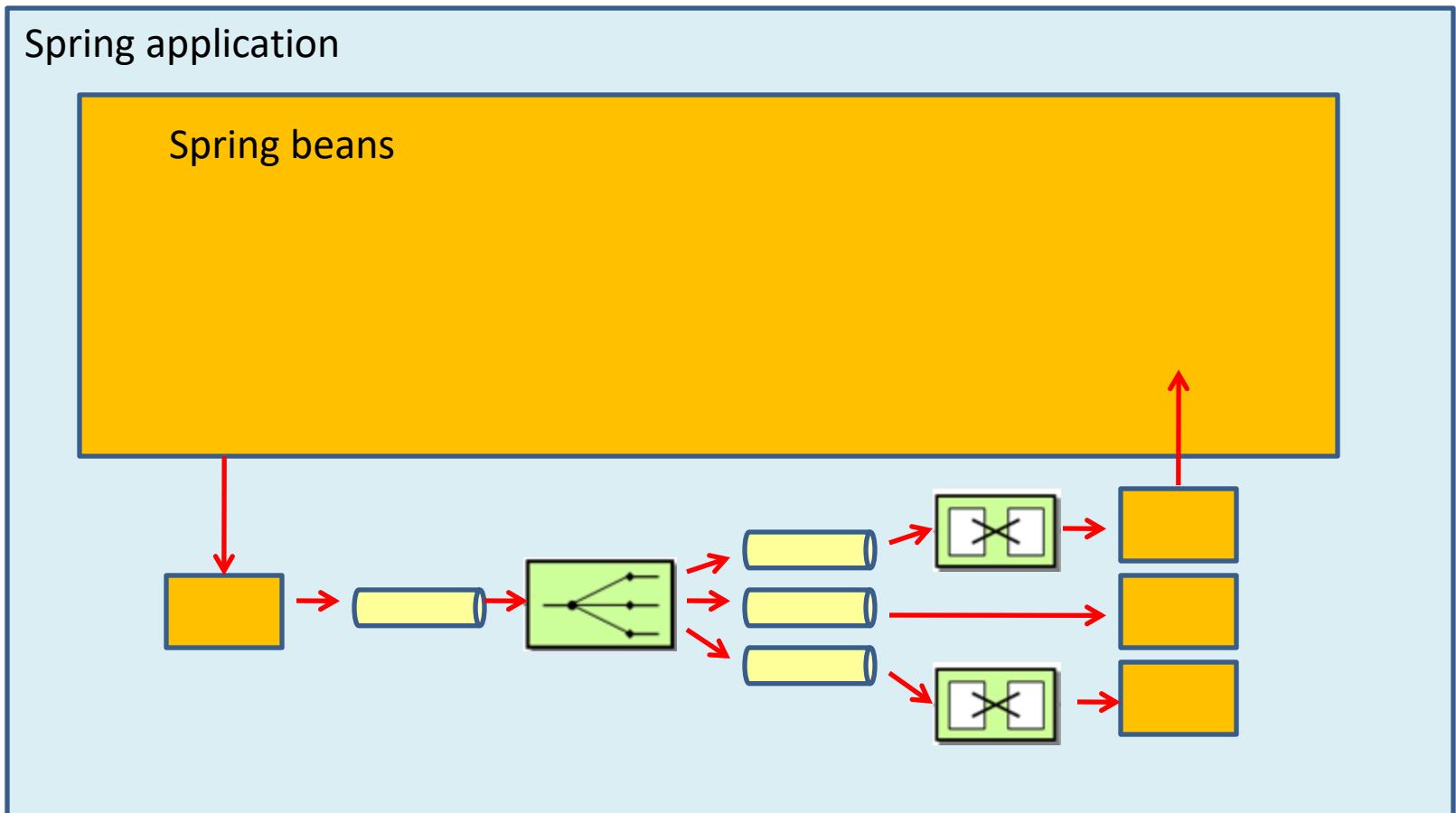


What is Spring Integration?

- Provides a simple model to implement complex enterprise integration solutions
- Facilitate asynchronous, parallel, message-driven behavior within a Spring-based application



Using Spring Integration

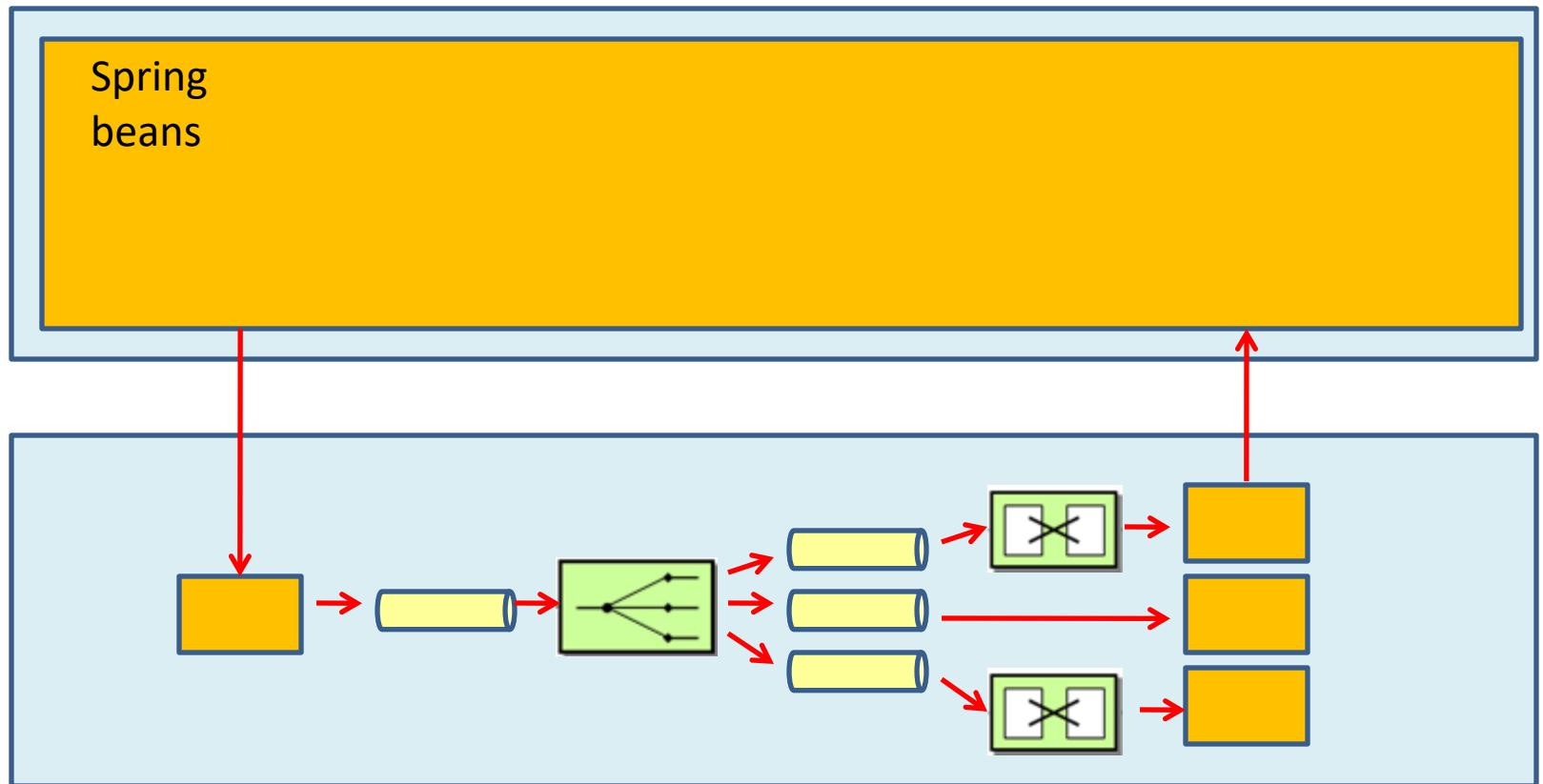


- Use SI inside your application



Using Spring Integration

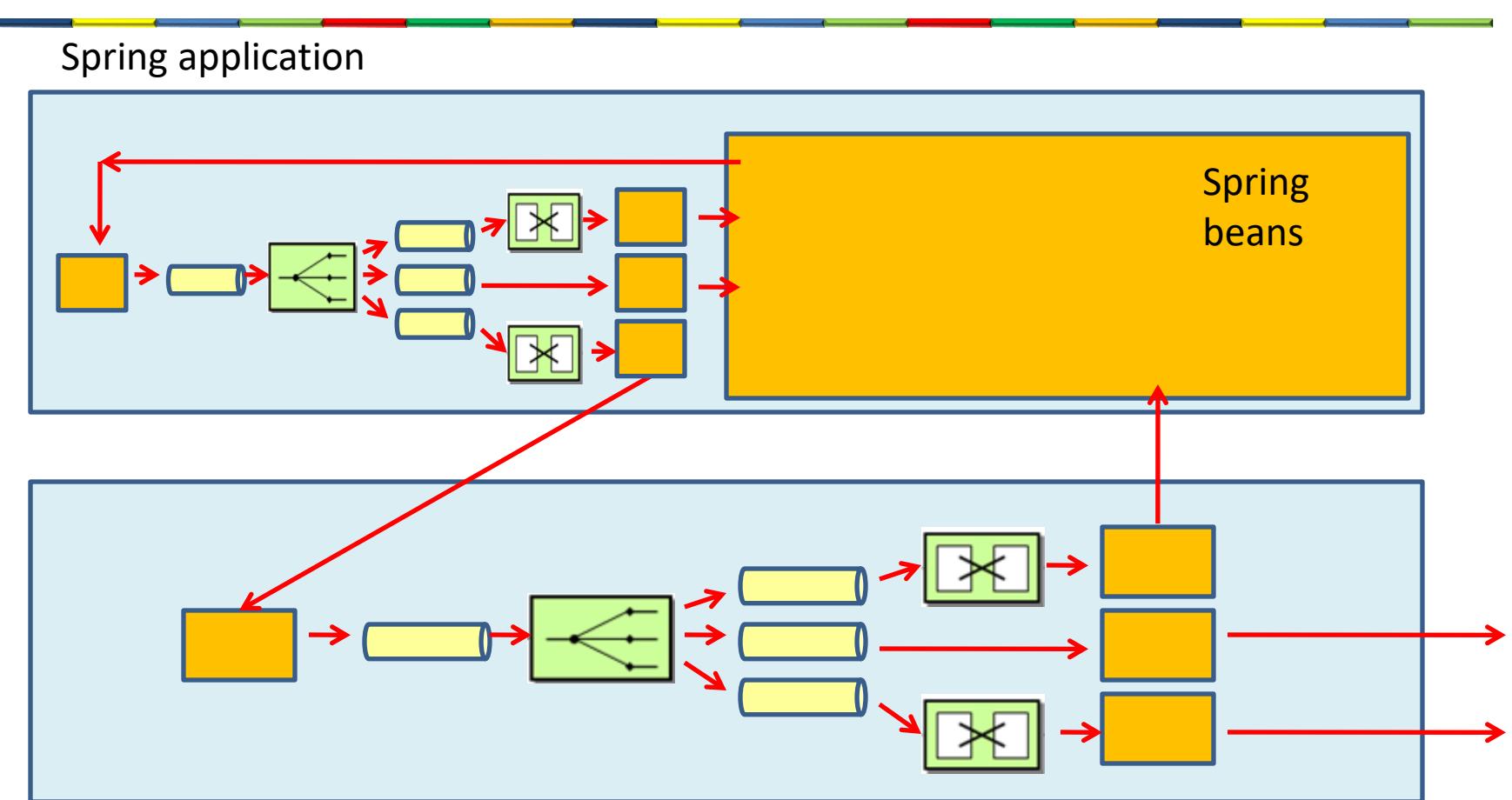
Spring application



- Use SI outside your application



Using Spring Integration



- Use SI inside and outside your application

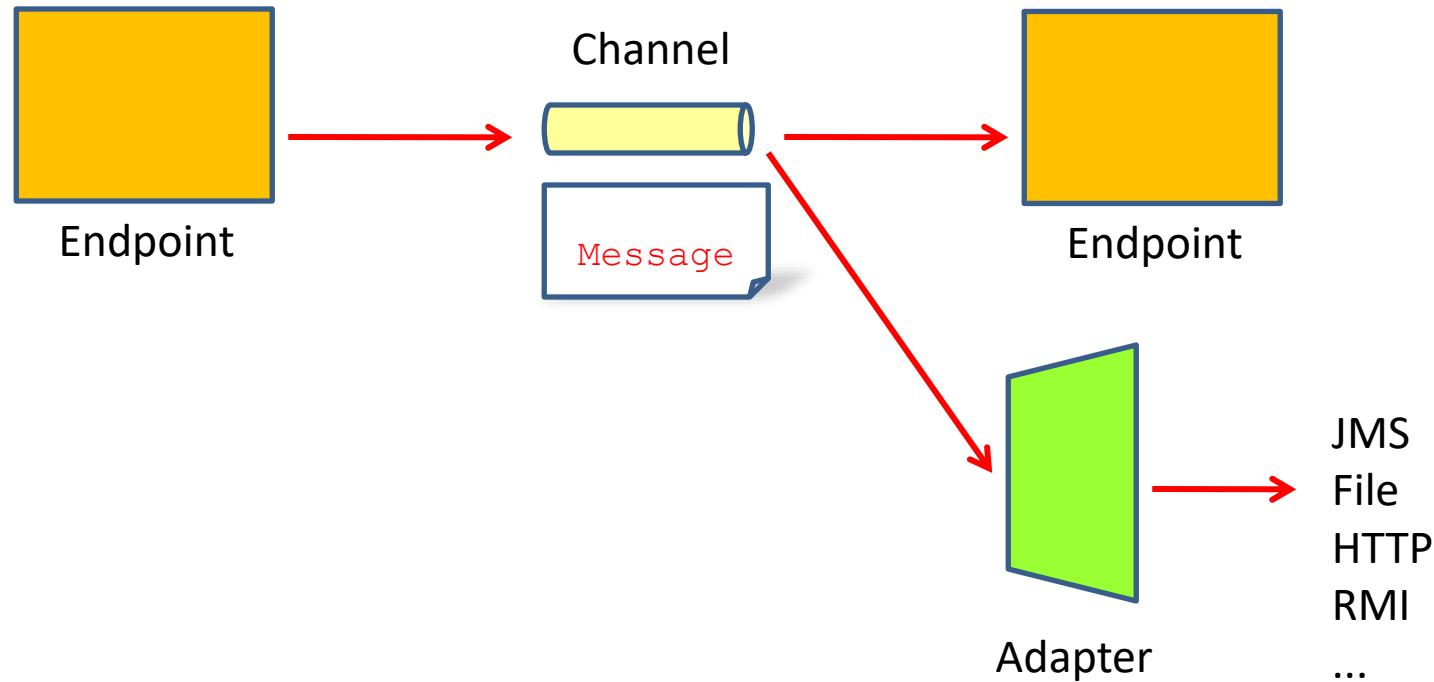


Difference with an ESB

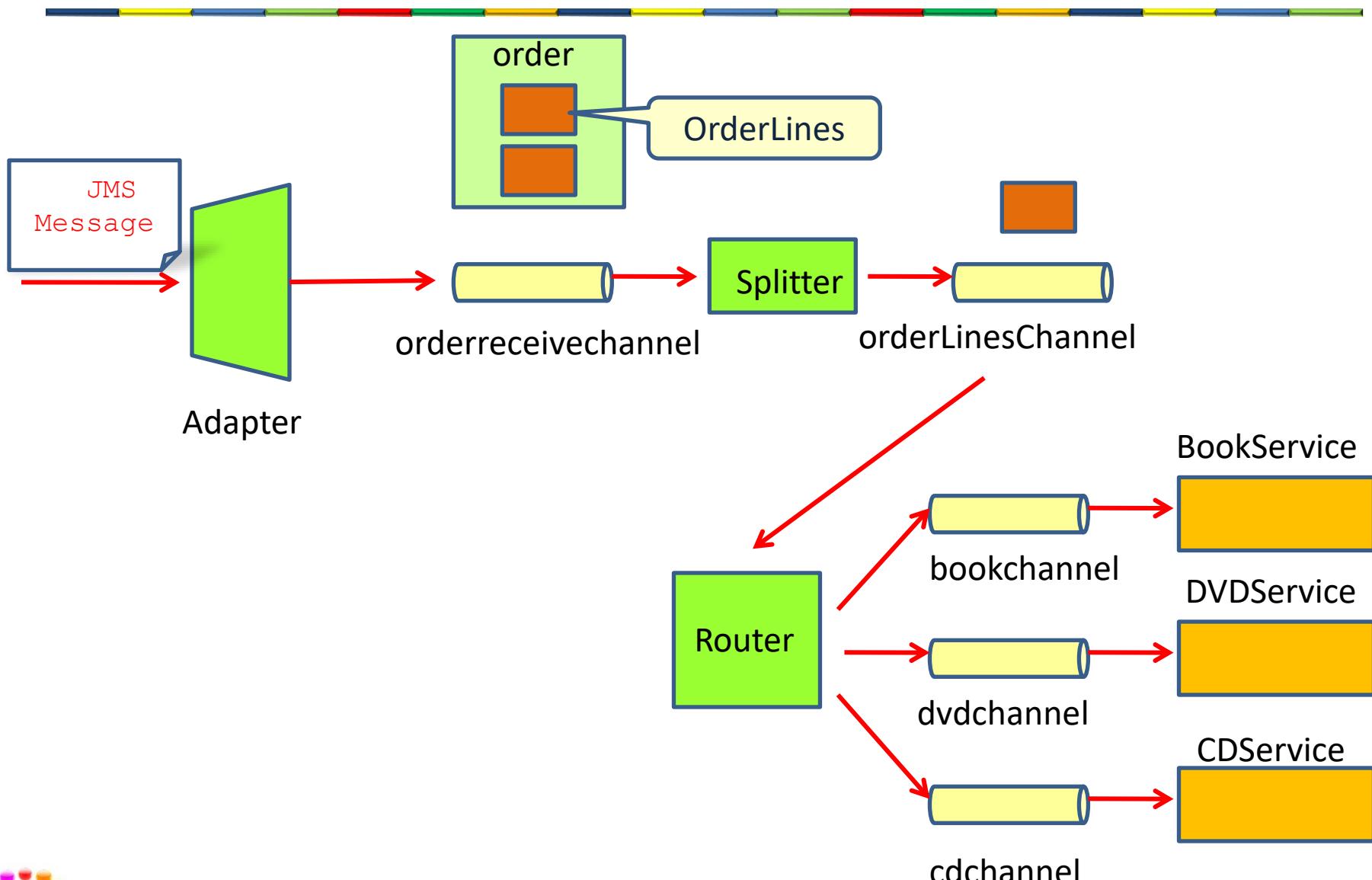
- ESB's run within its own VM
 - Spring Integration can run within an application
- You have to install ESB's
 - Spring integration is a library
- You have to start (and stop) ESB



Basic components



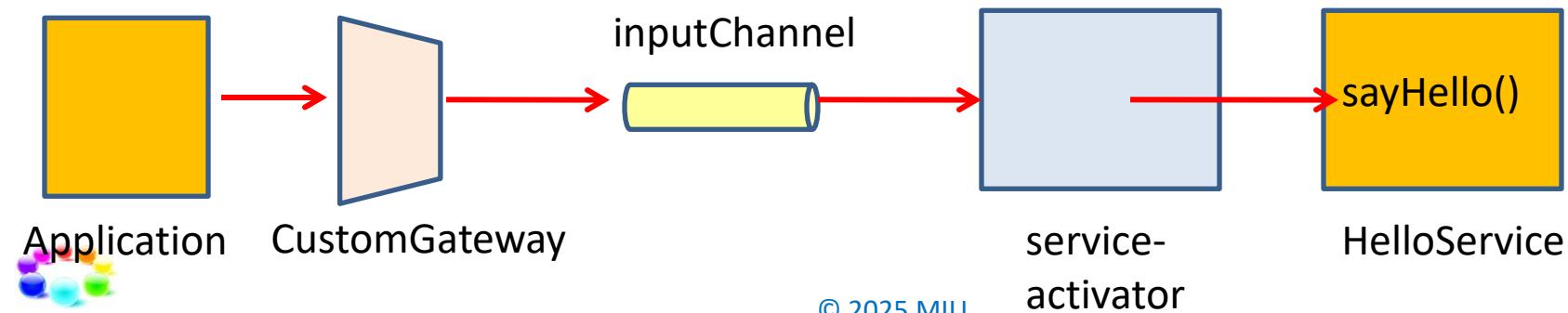
Spring Integration example



Spring integration Hello World

```
public class HelloService {  
    public void sayHello(String name){  
        System.out.println("Hello "+name);  
    }  
}
```

```
public interface CustomGateway {  
  
    public void process(String message);  
}
```



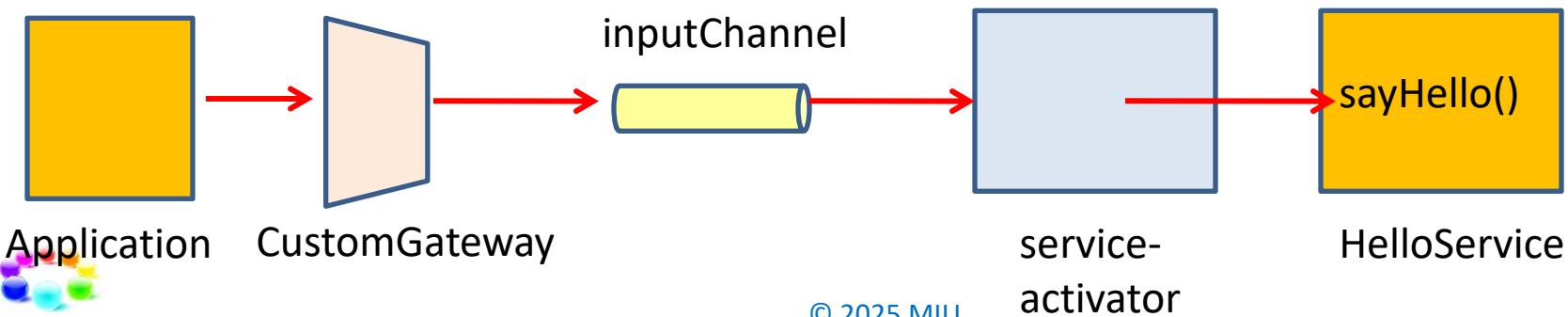
Integration-context.xml

```
<int:gateway service-interface="integration.CustomGateway"
             default-request-channel="inputChannel">
    <int:method name="process" />
</int:gateway>

<int:channel id="inputChannel" />

<int:service-activator
    input-channel="inputChannel" ref="helloService" method="sayHello" />

<bean id="helloService" class="integration.HelloService" />
```



The application

```
@SpringBootApplication
@ImportResource("integration-context.xml")
public class SpringIntegrationProjectApplication implements CommandLineRunner {

    @Autowired
    private CustomGateway gateway;

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

    @Override
    public void run(String... args) throws Exception {
        gateway.process("World");
    }
}
```



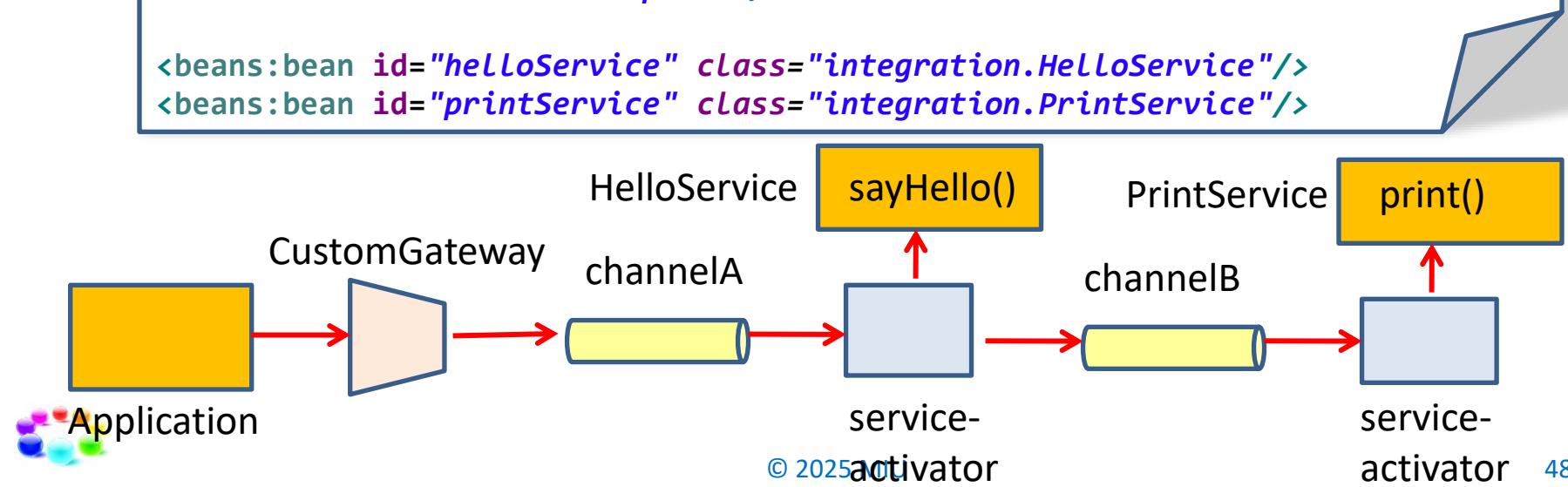
Extending the application

```
<int:gateway service-interface="integration.CustomGateway"
    default-request-channel=" channelA">
    <int:method name="process" />
</int:gateway>

<channel id="channelA"/>
<channel id="channelB"/>

<service-activator input-channel="channelA"
    output-channel="channelB"
    ref="helloService"
    method="sayHello"/>
<service-activator input-channel="channelB"
    ref="printService"
    method="print"/>

<beans:bean id="helloService" class="integration.HelloService"/>
<beans:bean id="printService" class="integration.PrintService"/>
```



Extending the application

```
public class HelloService {  
  
    public String sayHello(String name) {  
        System.out.println("HelloService: receiving name "+name);  
        return "Hello "+ name;  
    }  
}
```

```
public class PrintService {  
  
    public void print(String message) {  
        System.out.println("Printing message: "+ message);  
    }  
}
```



Sending an Order

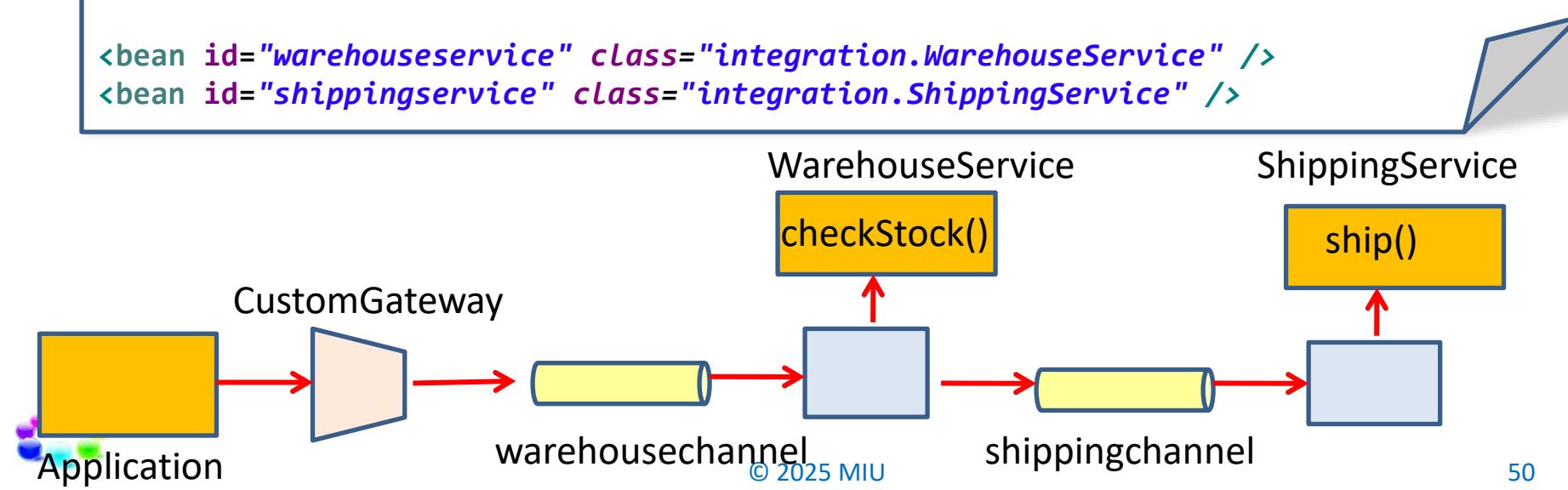
```
<int:gateway service-interface="integration.CustomGateway"
    default-request-channel="warehousechannel">
    <int:method name="process" />
</int:gateway>

<int:channel id="warehousechannel" />
<int:channel id="shippingchannel" />

<int:service-activator
    input-channel="warehousechannel" output-channel="shippingchannel"
    ref="warehouseservice" method="checkStock" />

<int:service-activator
    input-channel="shippingchannel" ref="shippingservice" method="ship" />

<bean id="warehouseservice" class="integration.WarehouseService" />
<bean id="shippingservice" class="integration.ShippingService" />
```



The services

```
public class WarehouseService {  
  
    public Order checkStock(Order order) {  
        System.out.println("WarehouseService: checking order "+order.toString());  
        return order;  
    }  
}
```

```
public class ShippingService {  
    public void ship(Order order) {  
        System.out.println("shipping: "+ order.toString());  
    }  
}
```

```
public class Order {  
    private String orderNumber;  
    private double amount;  
  
    public String toString(){  
        return "order: nr="+orderNumber+" amount="+amount;  
    }  
    ...  
}
```

The application

```
@SpringBootApplication
@ImportResource("integration-context.xml")
public class SpringIntegrationProjectApplication implements CommandLineRunner {

    @Autowired
    private CustomGateway gateway;

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

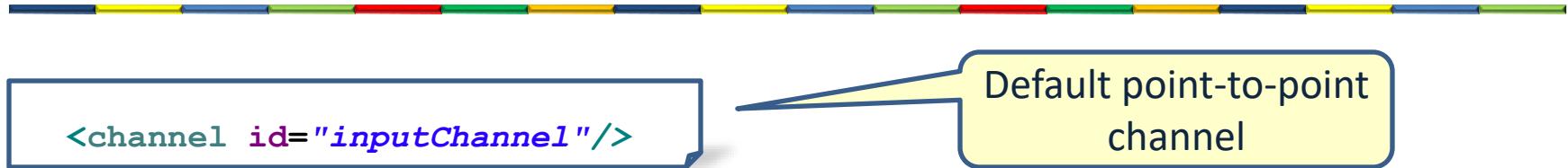
    @Override
    public void run(String... args) throws Exception {
        Order order = new Order("H-234-X56", 1245.75);
        Message<Order> message = MessageBuilder.withPayload(order).build();
        gateway.process(message);
    }
}
```

```
WarehouseService: checking order order: nr=H-234-X56 amount=1245.75
shipping: order: nr=H-234-X56 amount=1245.75
```

MESSAGE CHANNELS



Direct Channel



- Point-to-point
- When there are multiple handlers subscribed to the same channel
 - A “round-robin” loadbalancer balances the messages
 - The loadbalancer will automatically send the message to a subsequent handler if the preceding handler throws an exception (failover)

```
<channel id="failFastChannel">  
    <dispatcher failover="false"/>  
</channel>
```

No failover

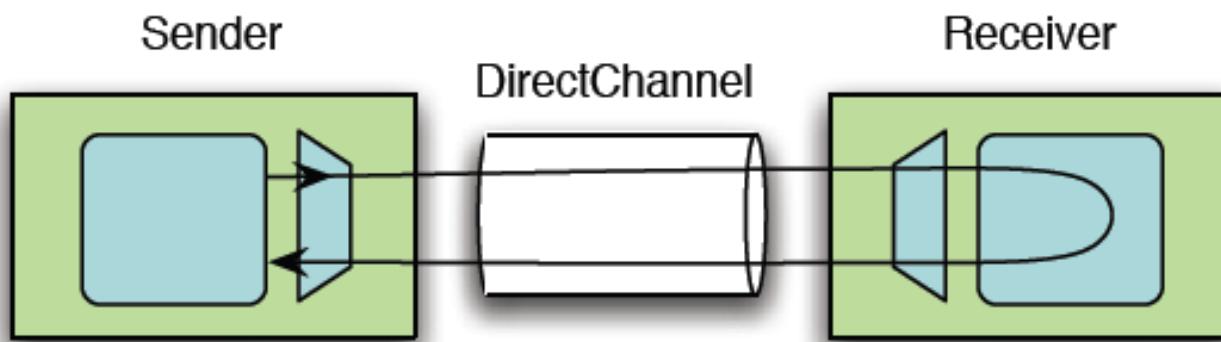
```
<channel id="channelWithFixedOrderSequenceFailover">  
    <dispatcher load-balancer="none"/>  
</channel>
```

No round-robin
balancer



Synchronous

- A direct default channel is synchronous

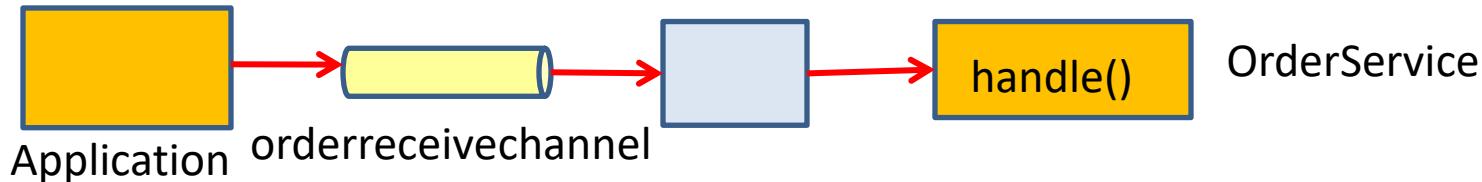


Synchronous

```
<channel id="orderreceivechannel" />  
  
<service-activator input-channel="orderreceivechannel"  
                    ref="orderservice" method="handle" />  
<beans:bean id="orderservice" class="integration.OrderService" />
```

```
public class OrderService {  
    public void handle(Order order) throws Exception {  
        System.out.println("OrderService receiving order: " + order.toString());  
        Thread.sleep(5000);  
    }  
}
```

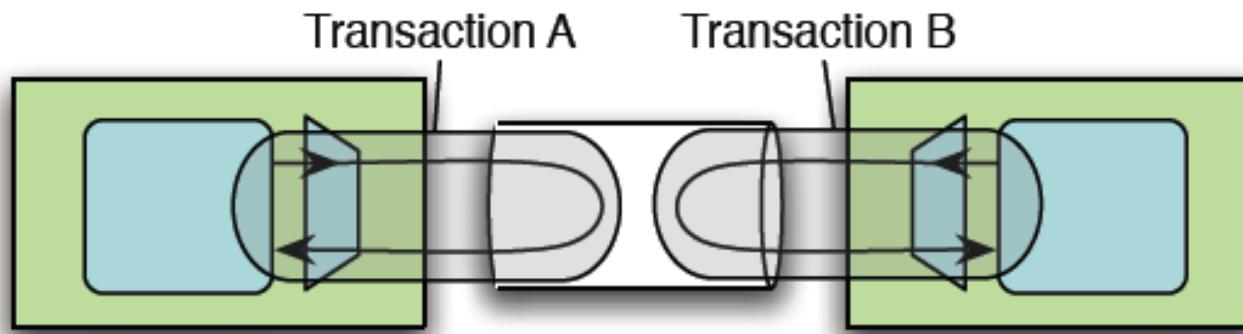
Sleep 5 seconds



```
time before sending message =8:54:15  
OrderService receiving order: order: nr=H-234-X56 amount=1245.75  
time after sending message =8:54:20
```

QueueChannel: Asynchronous

- A queue channel is asynchronous



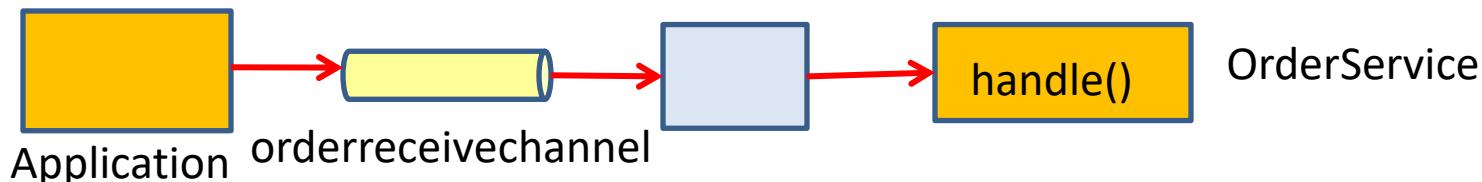
QueueChannel

```
<channel id="orderreceivechannel" >  
  <queue capacity="25"/>  
</channel>  
  
<service-activator input-channel="orderreceivechannel" ref="orderservice"  
  method="handle" >  
  <poller>  
    <interval-trigger interval="200"/>  
  </poller>  
</service-activator>  
  
<beans:bean id="orderservice" class="integration.OrderService" />
```

Add a queue

Now we need a poller

```
time before sending message =9:22:30  
time after sending message =9:22:30  
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
```



Datatype channel

```
<channel id="numberChannel" datatype="java.lang.Number"/>
```

Datatype Channel that only accepts messages containing a certain payload type

```
<channel id="stringOrNumberChannel"  
datatype="java.lang.String,java.lang.Number"/>
```

Accept multiple types



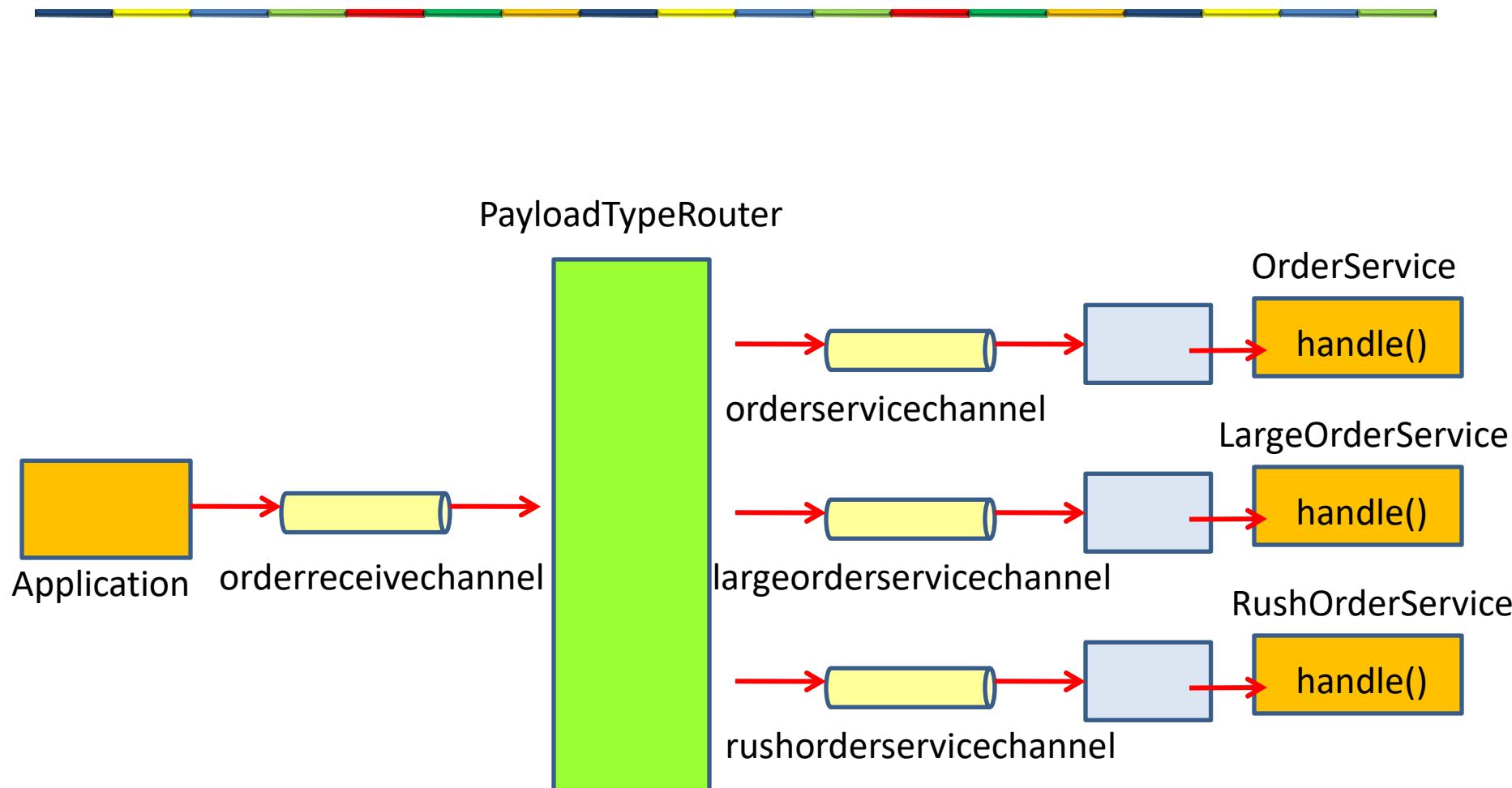
ROUTER

Routers

- Build-in routers
 - PayloadTypeRouter
 - HeaderValueRouter
 - RecipientListRouter
- Custom router



PayloadTypeRouter



PayloadTypeRouter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<payload-type-router input-channel="orderreceivechannel">
    <mapping type="integration.Order" channel="orderservicechannel" />
    <mapping type="integration.RushOrder" channel="rushorderservicechannel" />
    <mapping type="integration.LargeOrder" channel="largeorderservicechannel" />
</payload-type-router>

<service-activator input-channel="orderservicechannel"
                    ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
                    ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
                    ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```

The Payload types

```
public class Order {  
    private String orderNumber;  
    private double amount;  
  
    public String toString(){  
        return "order: nr="+orderNumber+" amount="+amount;  
    }  
    ...  
}
```

```
public class RushOrder extends Order{  
    public RushOrder(String orderNumber, double amount) {  
        super(orderNumber, amount);  
    }  
}
```

```
public class LargeOrder extends Order{  
    public LargeOrder(String orderNumber, double amount) {  
        super(orderNumber, amount);  
    }  
}
```



The services

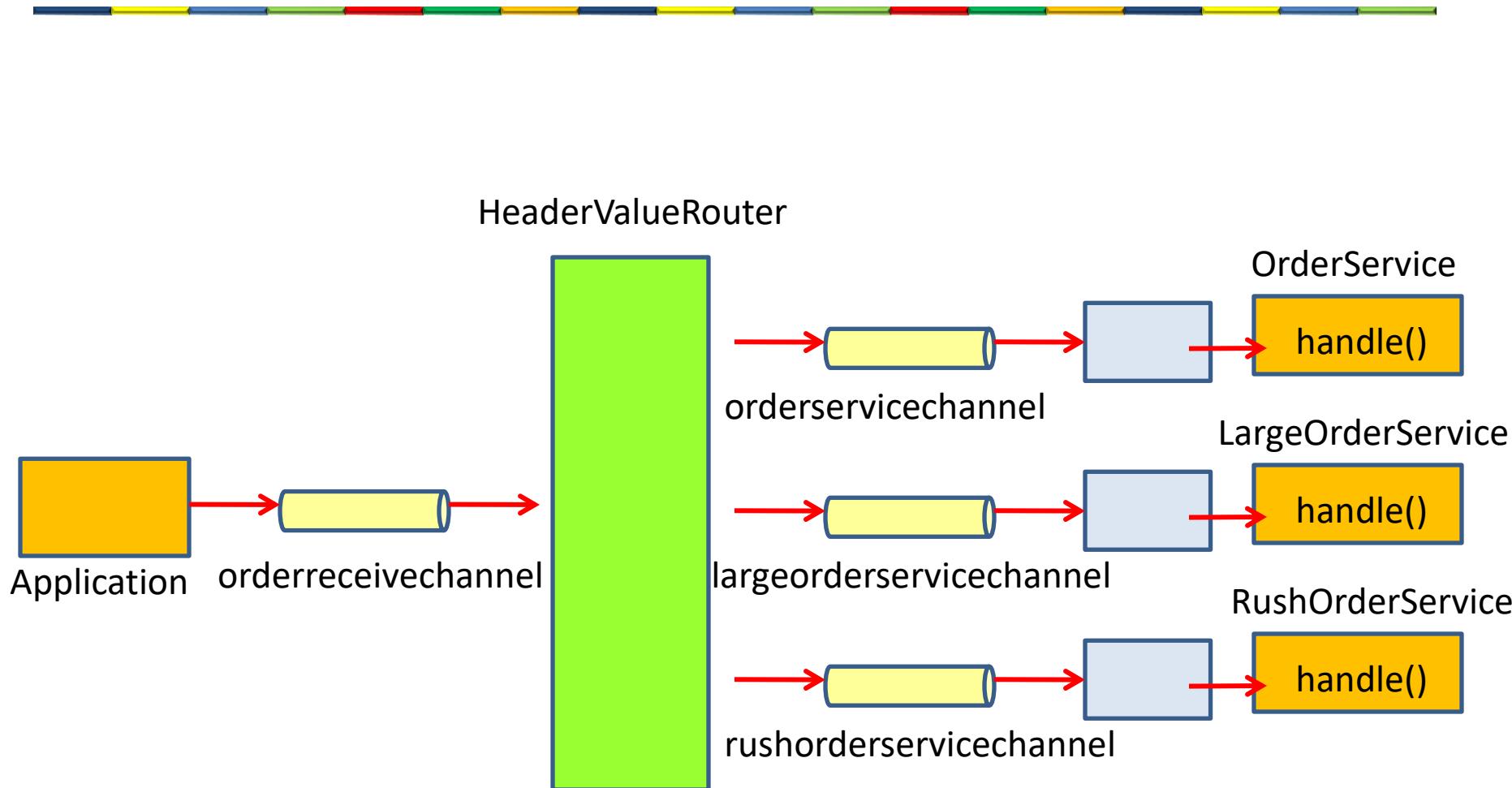
```
public class OrderService {  
    public void handle(Order order) {  
        System.out.println("OrderService receiving order: "+ order.toString());  
    }  
}
```

```
public class LargeOrderService {  
    public void handle(Order order) {  
        System.out.println("LargeOrderService receiving order: "+ order.toString());  
    }  
}
```

```
public class RushOrderService {  
    public void handle(Order order) {  
        System.out.println("RushOrderService receiving order: "+ order.toString());  
    }  
}
```



HeaderValueRouter



HeaderValueRouter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<header-value-router input-channel="orderreceivechannel"
                      header-name="orderType">
    <mapping value="normal" channel="orderservicechannel" />
    <mapping value="rush" channel="rushorderservicechannel" />
    <mapping value="large" channel="largeorderservicechannel" />
</header-value-router>

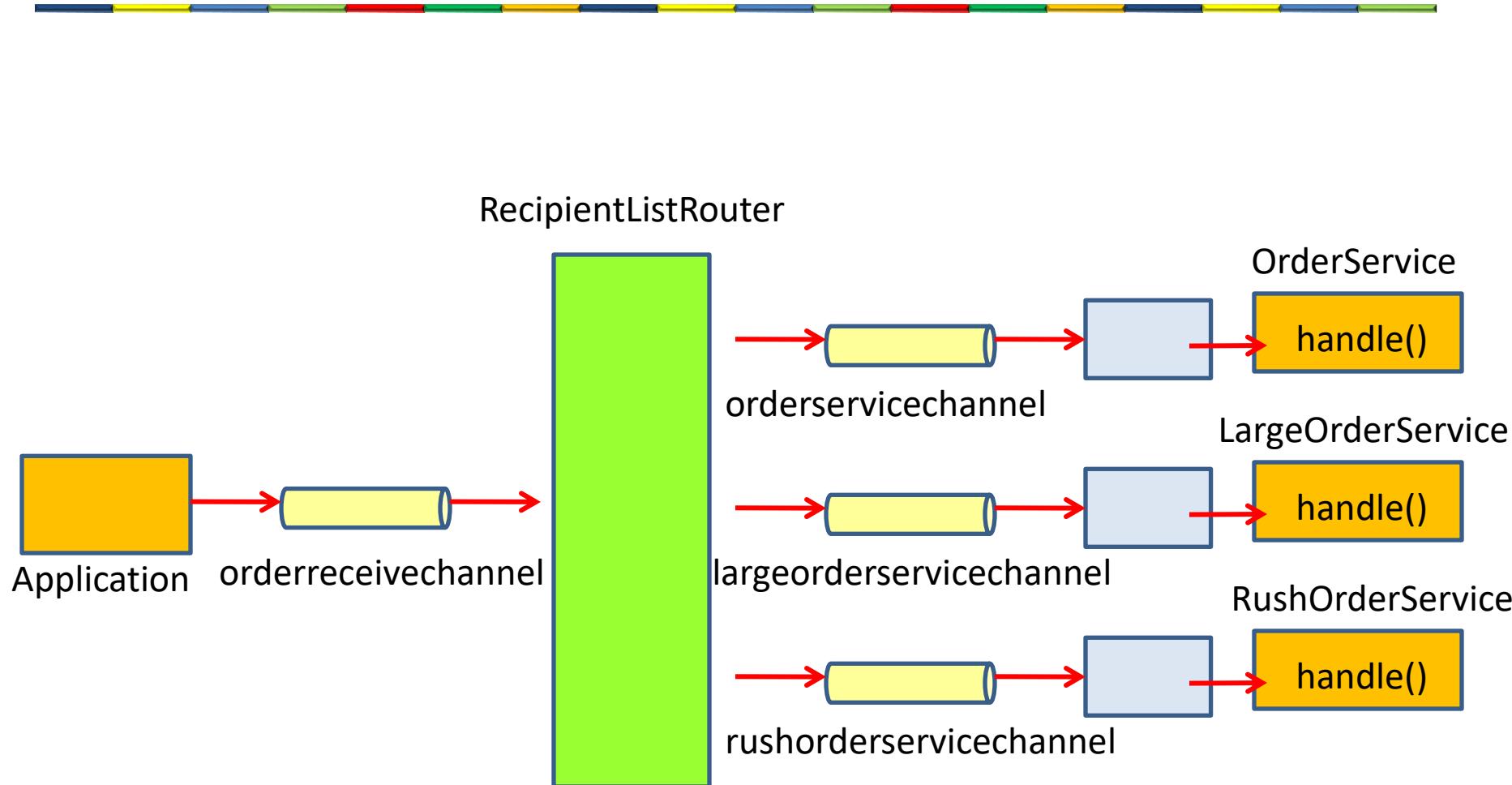
<service-activator input-channel="orderservicechannel"
                     ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
                     ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
                     ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```

RecipientListRouter



RecipientListRouter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<recipient-list-router id="customRouter" input-channel="orderreceivechannel"
                        apply-sequence="true">
    <recipient channel="orderservicechannel" />
    <recipient channel="rushorderservicechannel" />
    <recipient channel="largeorderservicechannel" />
</recipient-list-router>

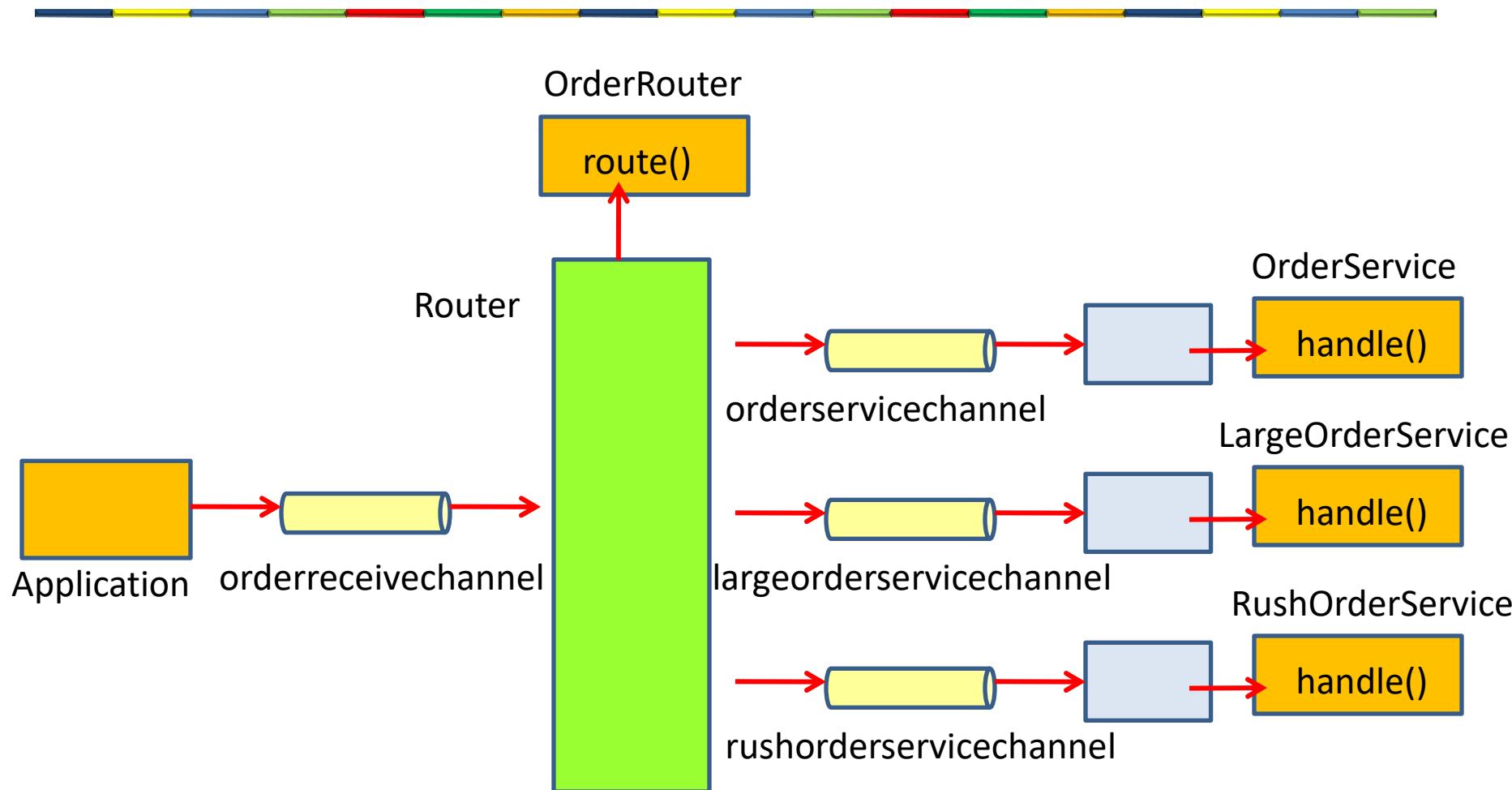
<service-activator input-channel="orderservicechannel"
ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```

Custom Router bean



Custom Router bean

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<router method="route" input-channel="orderreceivechannel">
    <beans:bean class="integration.OrderRouter" />
</router>

<service-activator input-channel="orderservicechannel"
ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```

The router bean

```
public class OrderRouter {  
    public String route(Order order) {  
        String destinationChannel = null;  
        if (order.isRush())  
            destinationChannel = "rushorderservicechannel";  
        else if (order.getAmount() > 20000)  
            destinationChannel = "largeorderservicechannel";  
        else  
            destinationChannel = "orderservicechannel";  
        return destinationChannel;  
    }  
}
```

```
RushOrderService receiving order: order: nr=H-234-X56 amount=1245.75  
OrderService receiving order: order: nr=H-234-X57 amount=600.65  
LargeOrderService receiving order: order: nr=H-234-X58 amount=50600.65
```



The router bean: multiple return values

```
public class OrderRouter {  
    public List<String> route(Order order) {  
        List<String> destinationChannels = new ArrayList<String>();  
        if (order.isRush())  
            destinationChannels.add("rushorderservicechannel");  
        if (order.getAmount() > 20000)  
            destinationChannels.add("largeorderservicechannel");  
        destinationChannels.add("orderservicechannel");  
        return destinationChannels;  
    }  
}
```

```
RushOrderService receiving order: order: nr=H-234-X56 amount=1245.75  
OrderService receiving order: order: nr=H-234-X56 amount=1245.75  
OrderService receiving order: order: nr=H-234-X57 amount=600.65  
RushOrderService receiving order: order: nr=H-234-X58 amount=50600.65  
LargeOrderService receiving order: order: nr=H-234-X58 amount=50600.65  
OrderService receiving order: order: nr=H-234-X58 amount=50600.65
```



FILTER



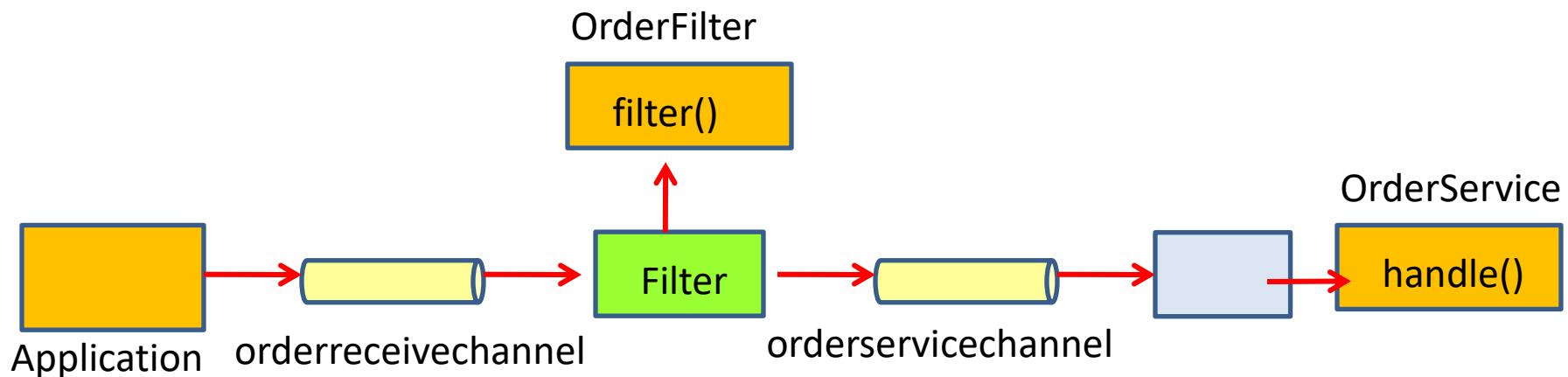
Filter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />

<filter input-channel="orderreceivechannel" output-channel="orderservicechannel"
       ref="orderfilter" method="filter"/>

<service-activator input-channel="orderservicechannel"
                     ref="orderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="orderfilter" class="integration.OrderFilter" />
```



The Filter class

```
public class OrderFilter {  
    public boolean filter(Order order) {  
        if (order.getAmount() > 800)  
            return true;  
        else  
            return false;  
    }  
}
```

What to do with rejected messages?

```
<filter input-channel="orderreceivechannel" output-channel="orderservicechannel"  
ref="orderfilter" method="filter" throw-exception-on-rejection="true"/>
```

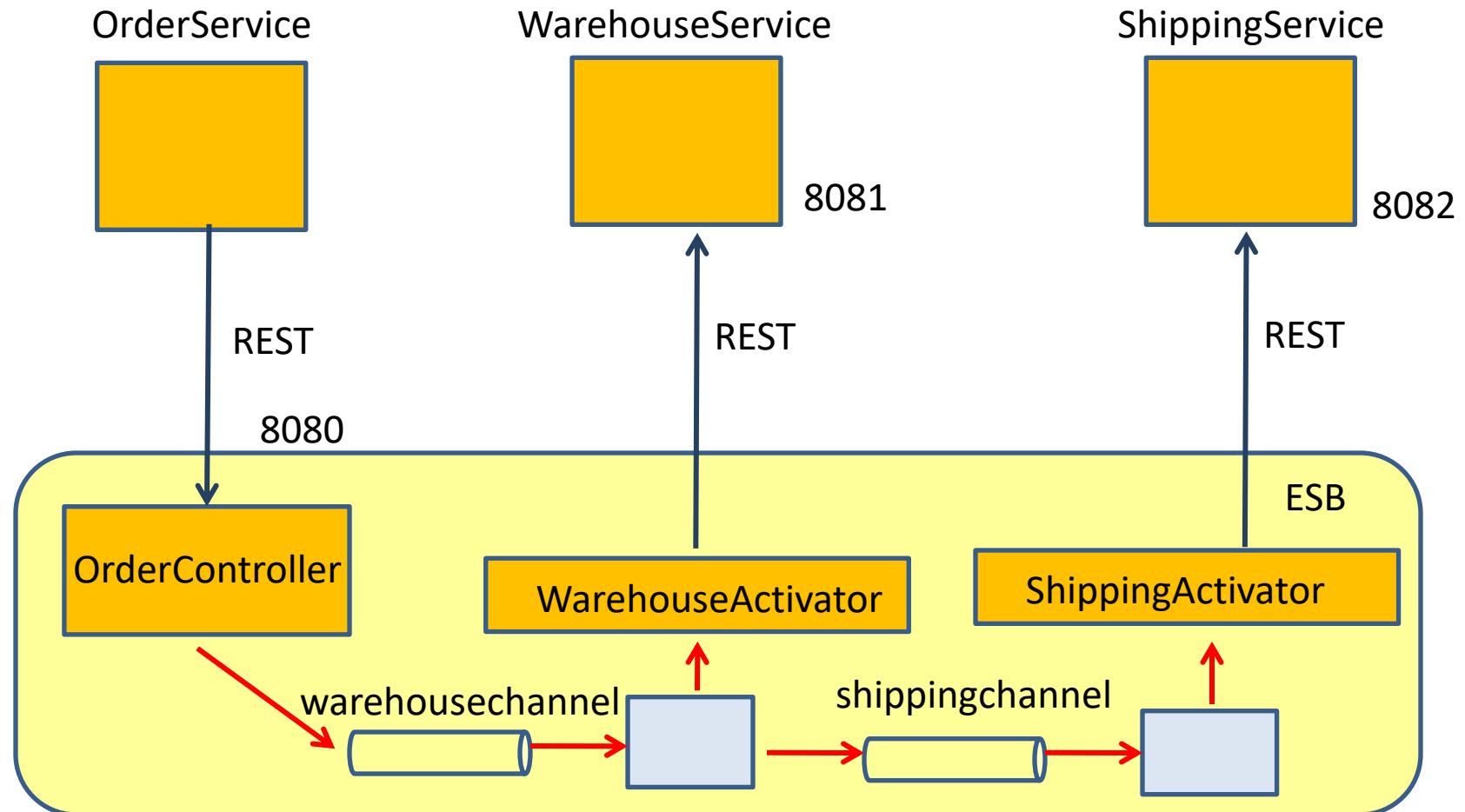
Throw an exception if
a message is rejected

```
<filter input-channel="orderreceivechannel" output-channel="orderservicechannel"  
ref="orderfilter" method="filter" discard-channel="rejectedMessages"/>
```

Send rejected
messages to another
channel



ESB with spring integration



ESB configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/integration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/integration
        http://www.springframework.org/schema/integration/spring-integration.xsd">

    <channel id="wharehousechannel"/>
    <channel id="shippingchannel"/>

    <service-activator input-channel="wharehousechannel"
        output-channel="shippingchannel"
        ref="warehouseservice"
        method="checkStock"/>

    <service-activator input-channel="shippingchannel"
        ref="shippingservice"
        method="ship"/>

    <beans:bean id="warehouseservice" class="esb.WarehouseActivator"/>
    <beans:bean id="shippingservice" class="esb.ShippingActivator"/>

</beans:beans>
```

OrderController

```
@RestController
public class OrderController {
    @Autowired
    @Qualifier("wharehousechannel")
    MessageChannel warehouseChannel;

    @PostMapping("/orders")
    public ResponseEntity<?> receiveOrder(@RequestBody Order order) {
        Message<Order> orderMessage = MessageBuilder.withPayload(order).build();
        warehouseChannel.send(orderMessage);
        return new ResponseEntity<Order>(order, HttpStatus.OK);
    }
}
```



The activator beans

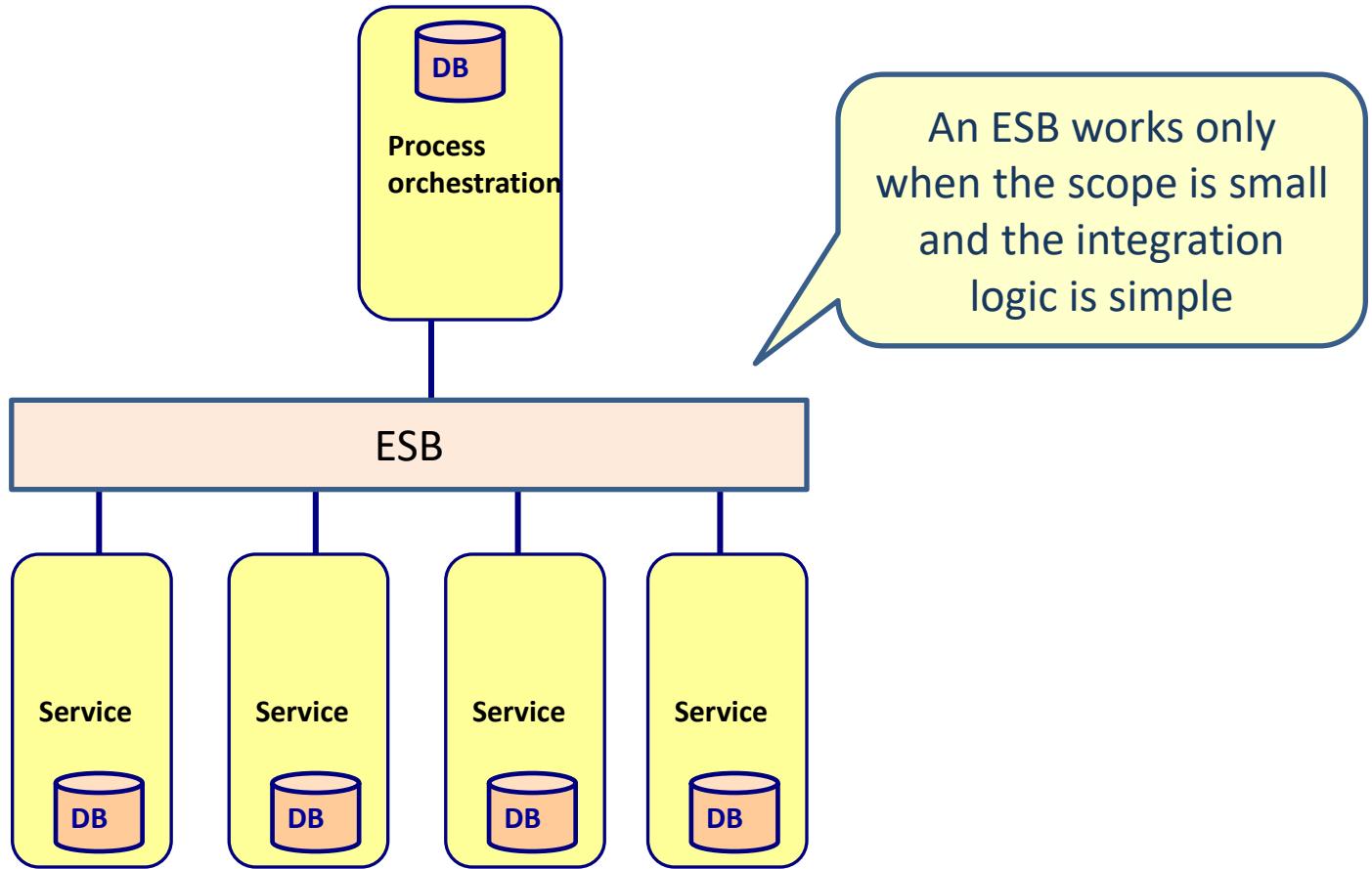
```
public class WarehouseActivator {  
  
    @Autowired  
    RestTemplate restTemplate;  
  
    public Order checkStock(Order order) {  
        System.out.println("WarehouseService: checking order "+order.toString());  
        restTemplate.postForLocation("http://localhost:8082/orders", order);  
        return order;  
    }  
}
```

```
public class ShippingActivator {  
    @Autowired  
    RestTemplate restTemplate;  
  
    public void ship(Order order) {  
        System.out.println("shipping: "+ order.toString());  
        restTemplate.postForLocation("http://localhost:8081/orders", order);  
    }  
}
```

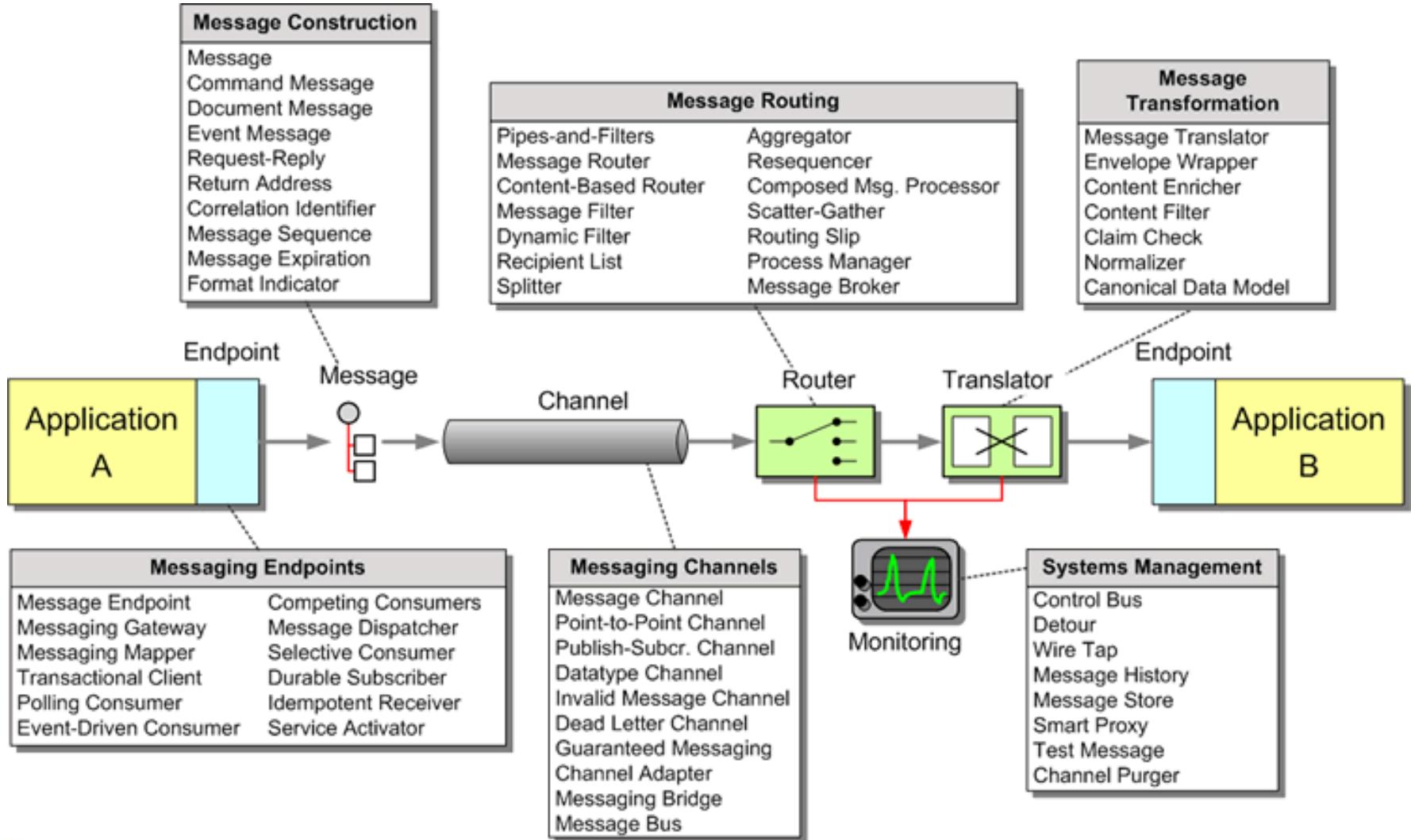


SUMMARY

Service Oriented Architecture



Enterprise Integration Patterns



Connecting the parts of knowledge with the wholeness of knowledge

1. By externalizing integration logic from the application into an ESB, the applications become more loosely coupled.
2. Integration logic can be designed with a basic set of integration patterns.



3. **Transcendental consciousness** is the field that connects everything together.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that everything else in creation is connected at the field of pure consciousness



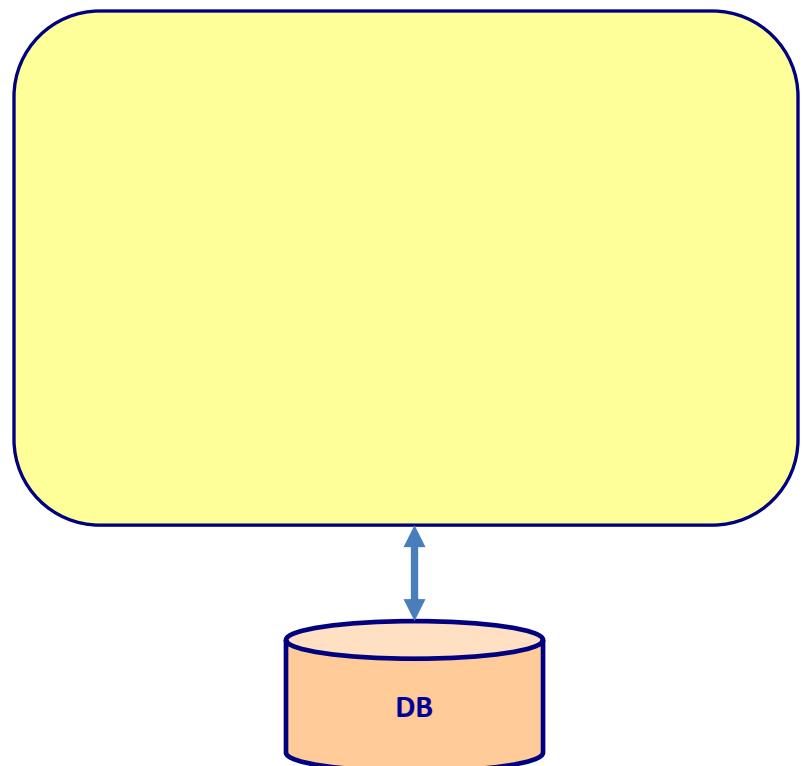
Lesson 7

MICROSERVICES

MONOLITH ARCHITECTURE

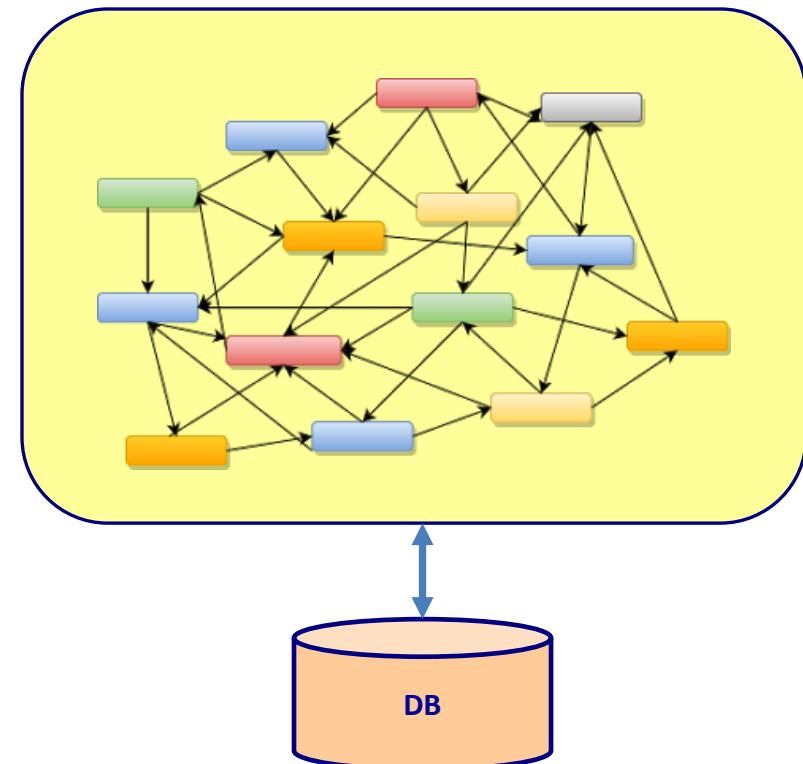
Monolith architecture

- Everything is implemented in one large system



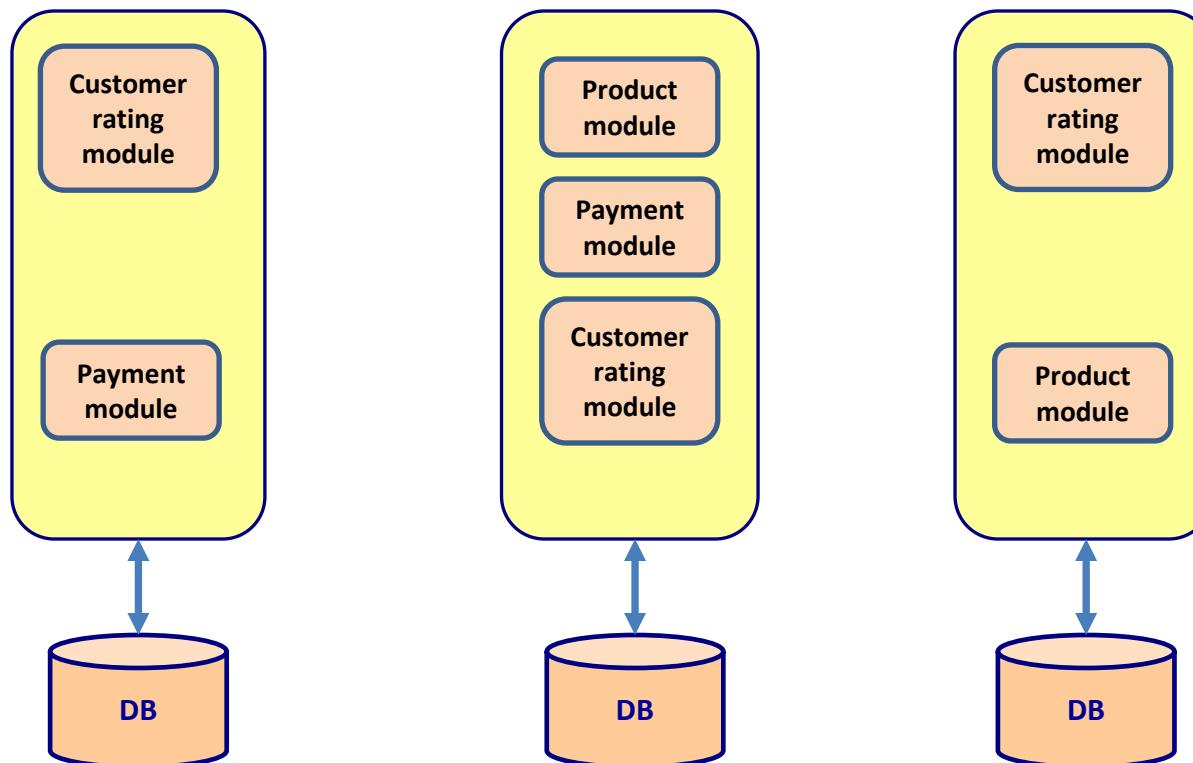
Monolith architecture

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



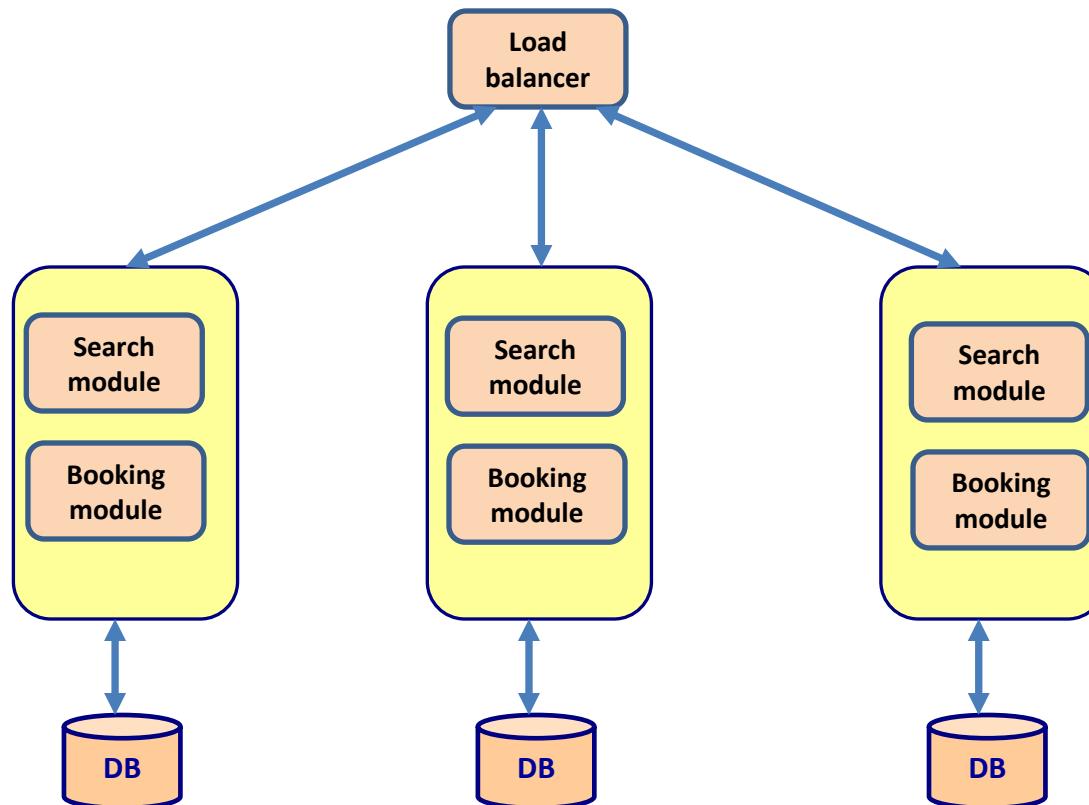
Monolith architecture

- Limited re-use is realized across monolithic applications



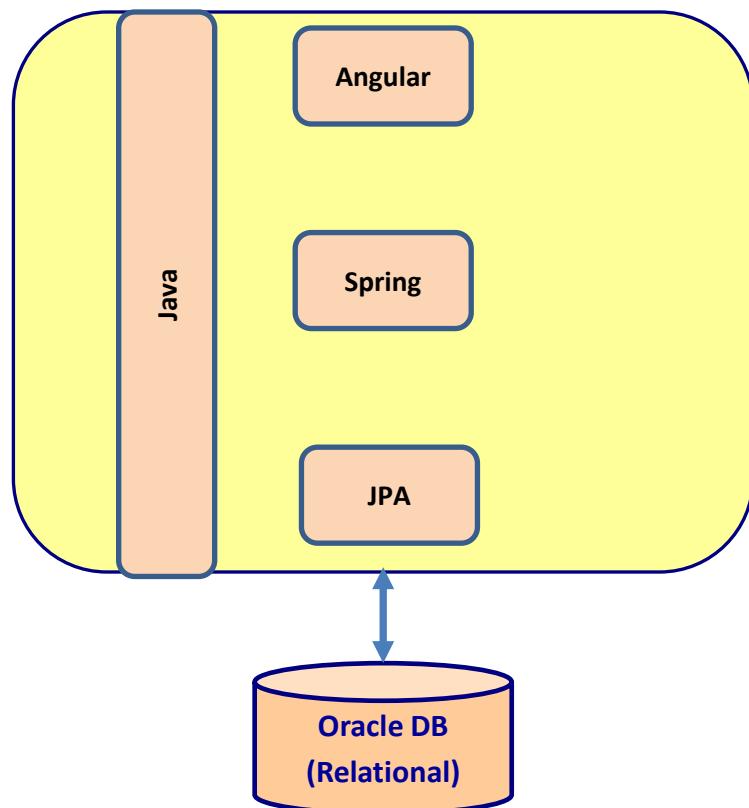
Monolith architecture

- All or nothing scaling
 - Difficult to scale separate parts



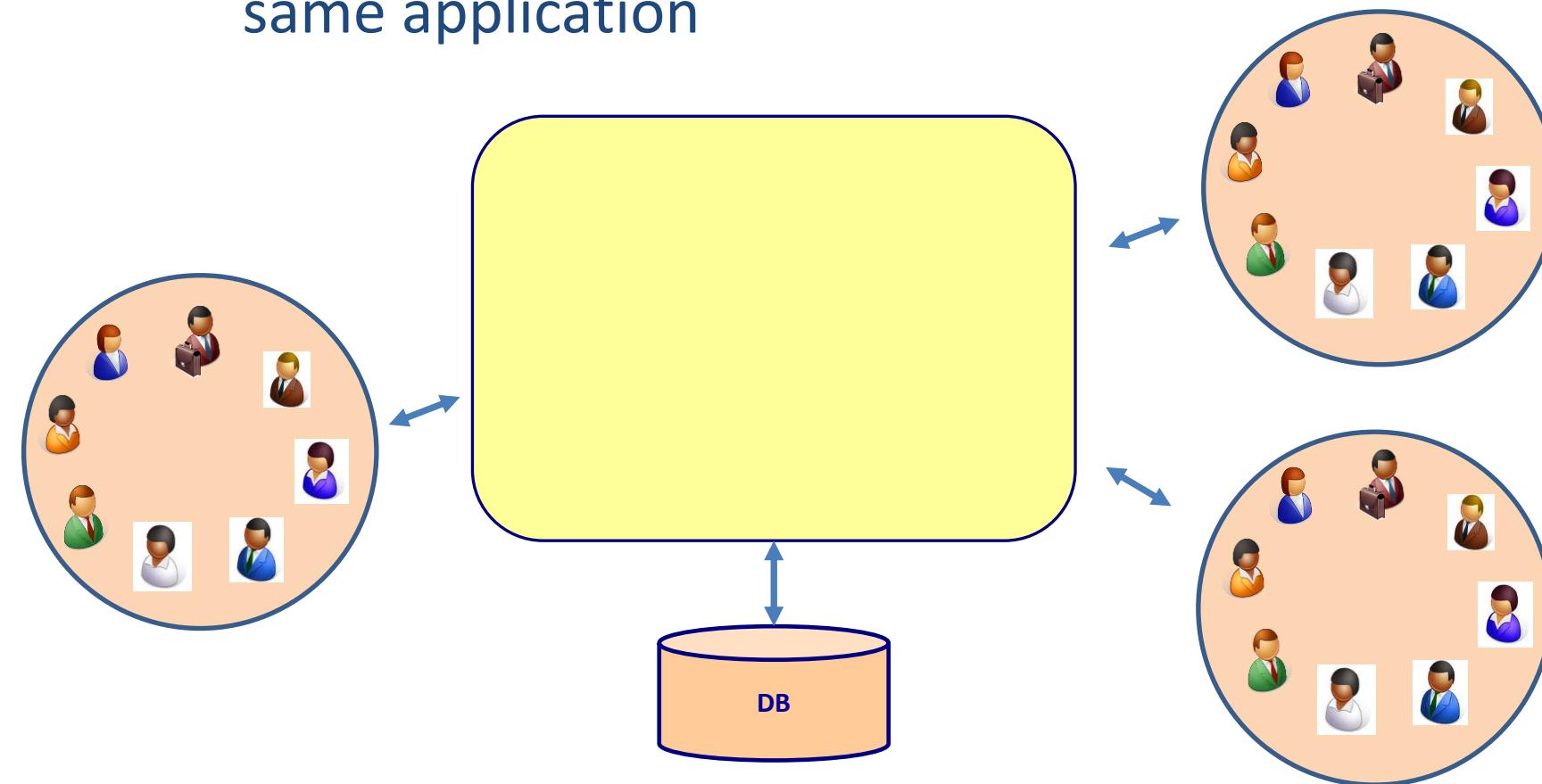
Monolith architecture

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



Monolith architecture

- Does not 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

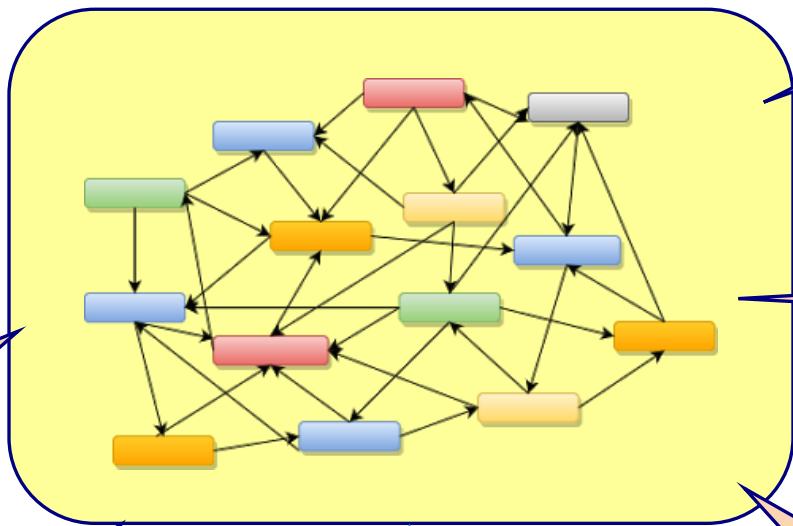
Single development stack

Difficult to work on
with multiple scrum
teams

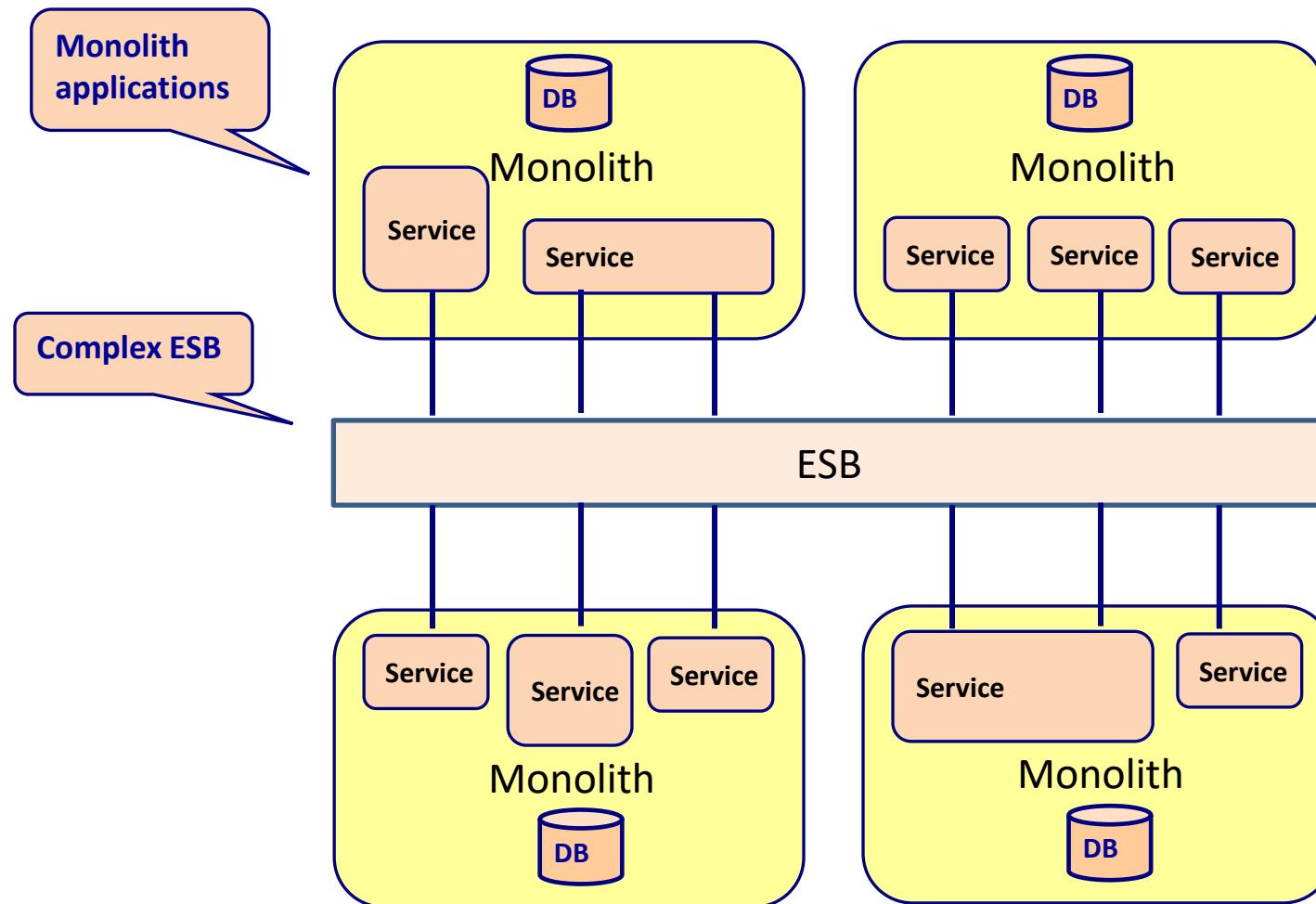
All or nothing scaling

Deployment takes a
lot of ceremony

Not much reuse
between monoliths

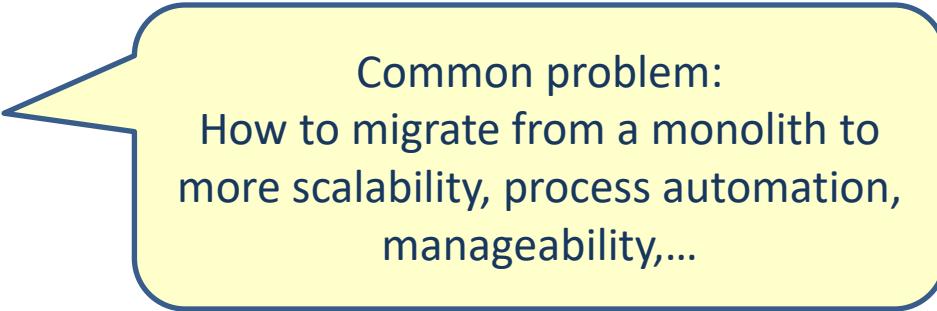


Problems with SOA



Microservice early adopters

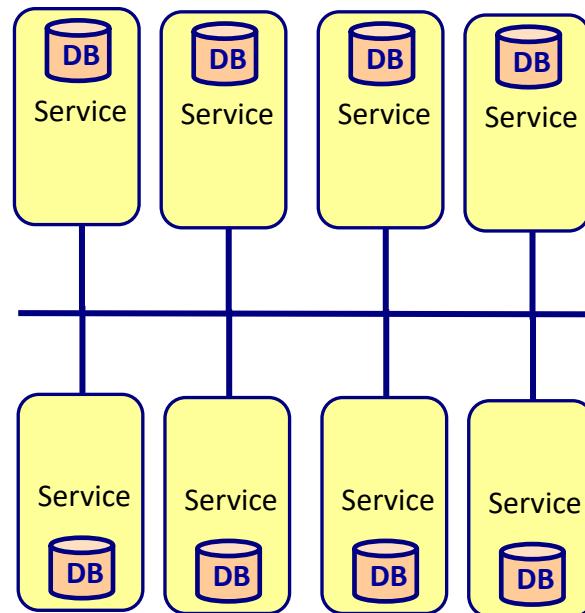
- Netflix
- Uber
- Airbnb
- Orbiz
- eBay
- Amazon
- Twitter
- Nike



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

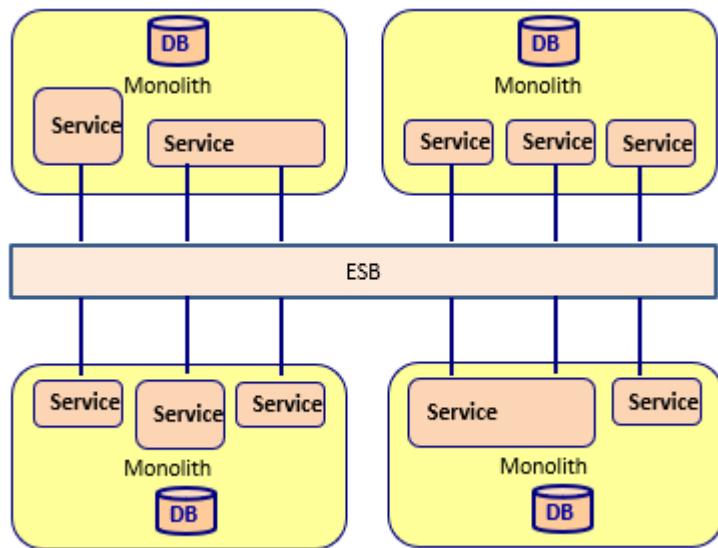
Microservices

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



Orchestration vs. choreography

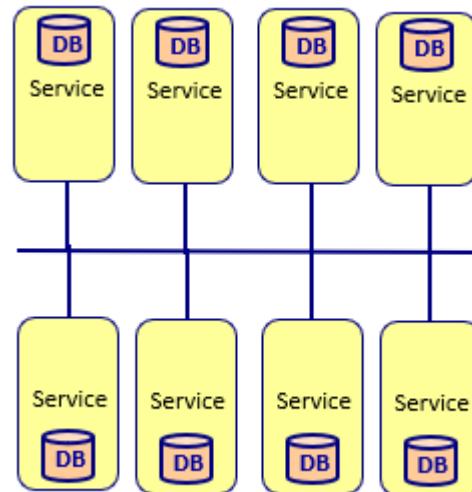
- Orchestration
 - SOA



Easy to follow
the process

Does not work
well in large and or
complex
applications

- Choreography
 - Microservices



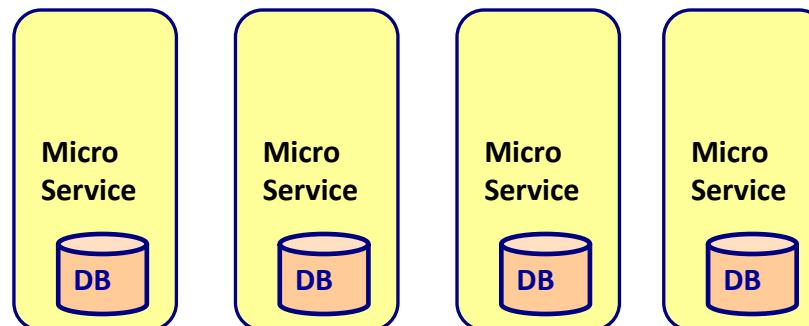
Hard to follow
the process

Does work well in
large and or
complex
applications

CHARACTERISTICS OF A MICROSERVICE

Microservices

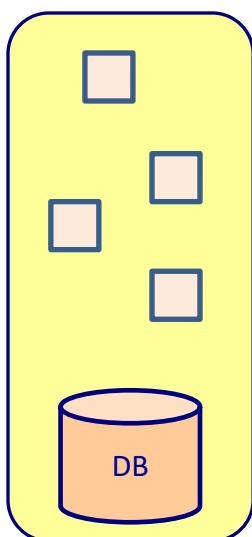
- Small independent services
 - Simple and lightweight
 - Runs in an independent process
 - Technology agnostic
 - Decoupled



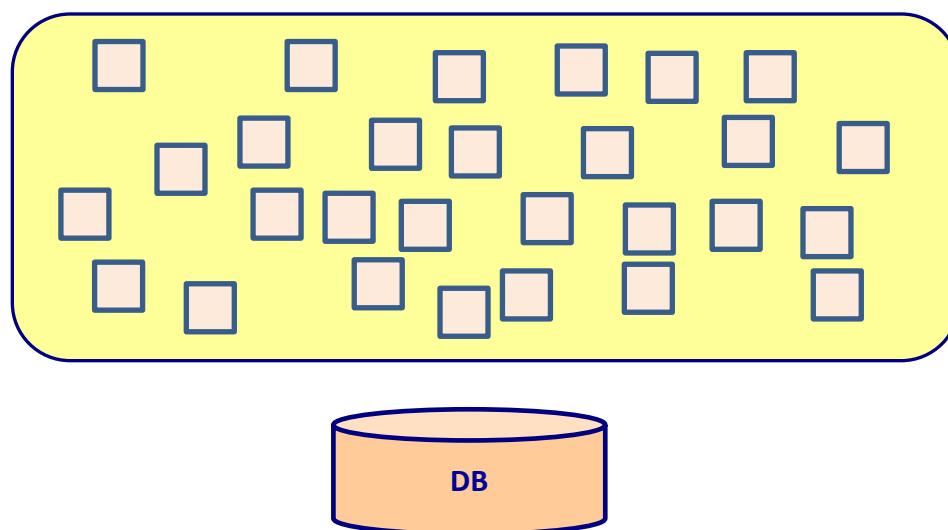
Simple and lightweight

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

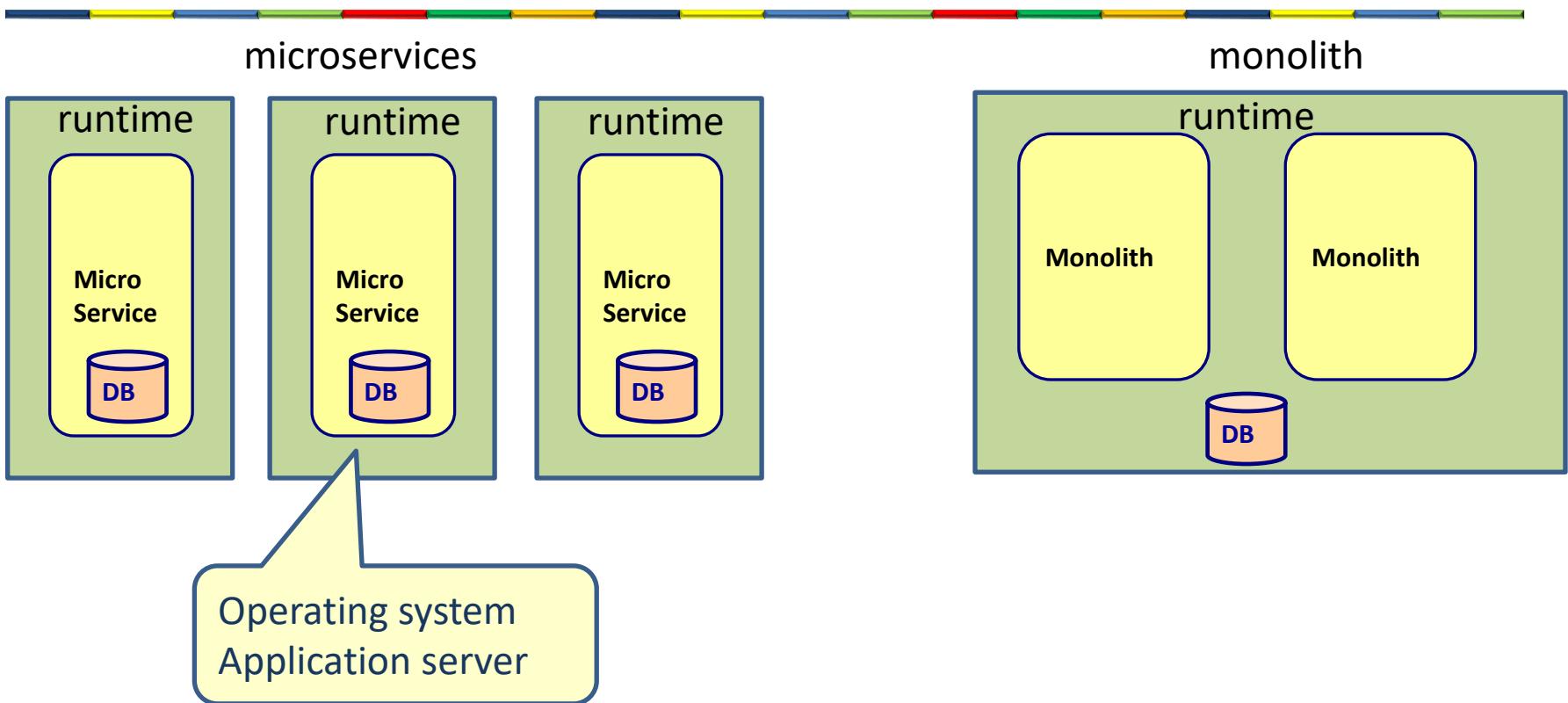
microservices



monolith



Runs in an independent process



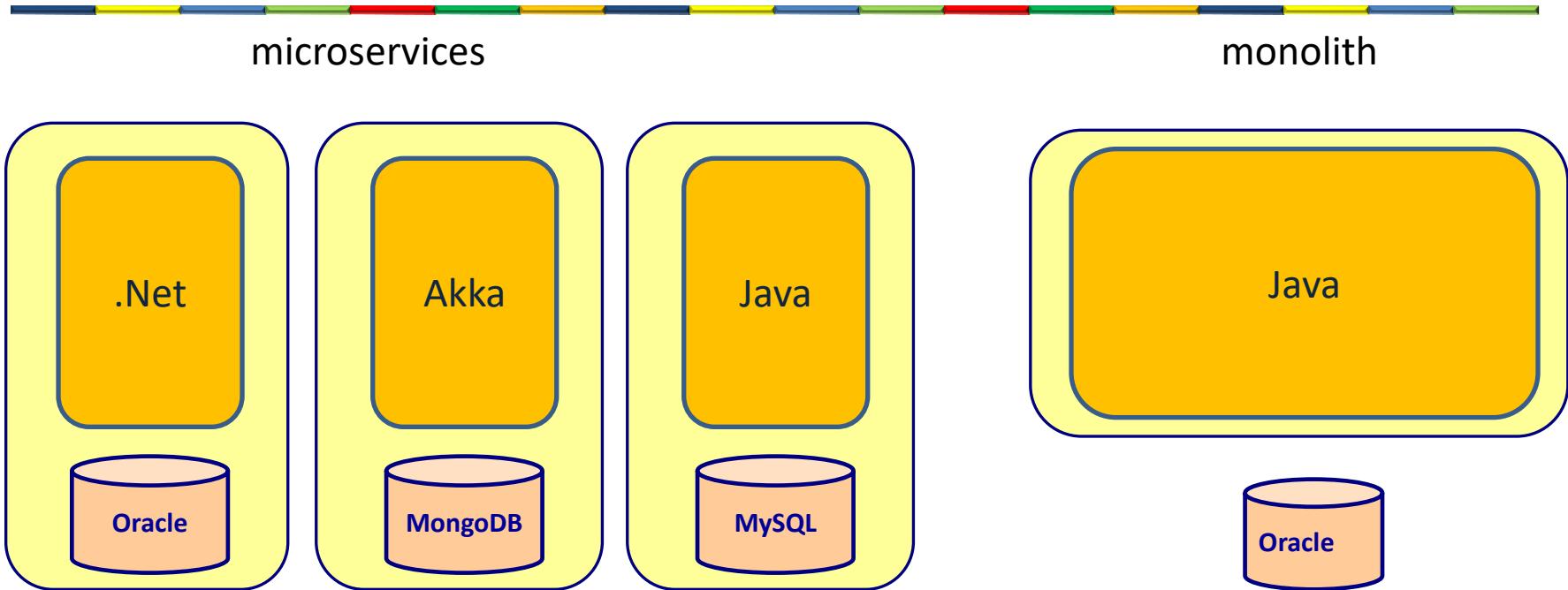
Advantages

- 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

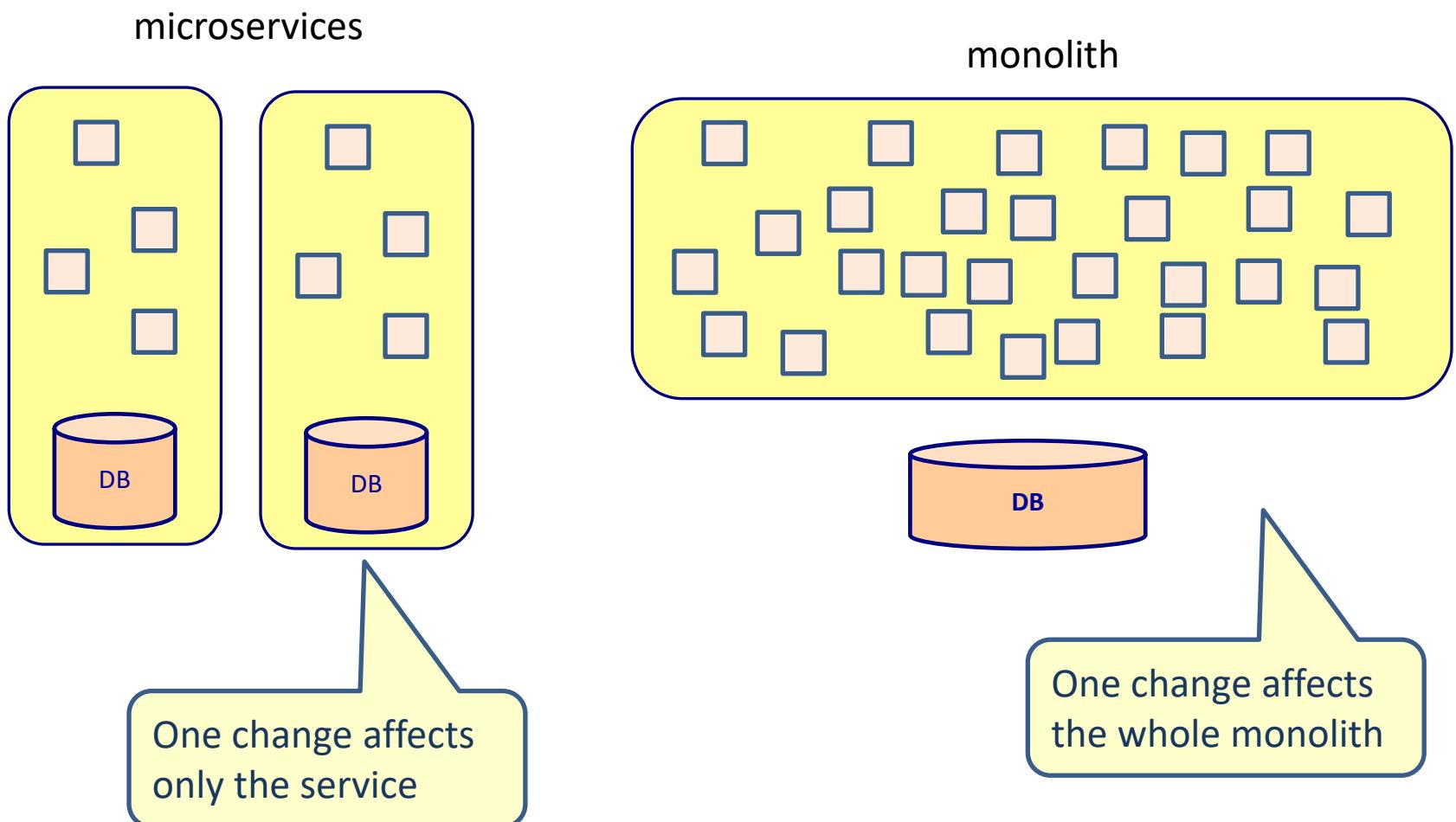
- We need to manage many runtimes

Technology agnostic



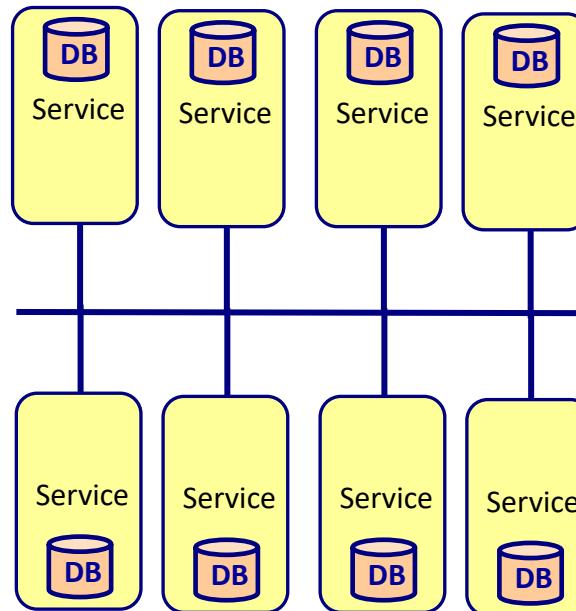
- Use the architecture and technologies that fits the best for this particular microservice

Decoupled



Microservice architecture

Simple microservices
Simpler to understand
Simpler to maintain



Every microservice
has its own
development stack

Every scrum teams
owns one or more
services

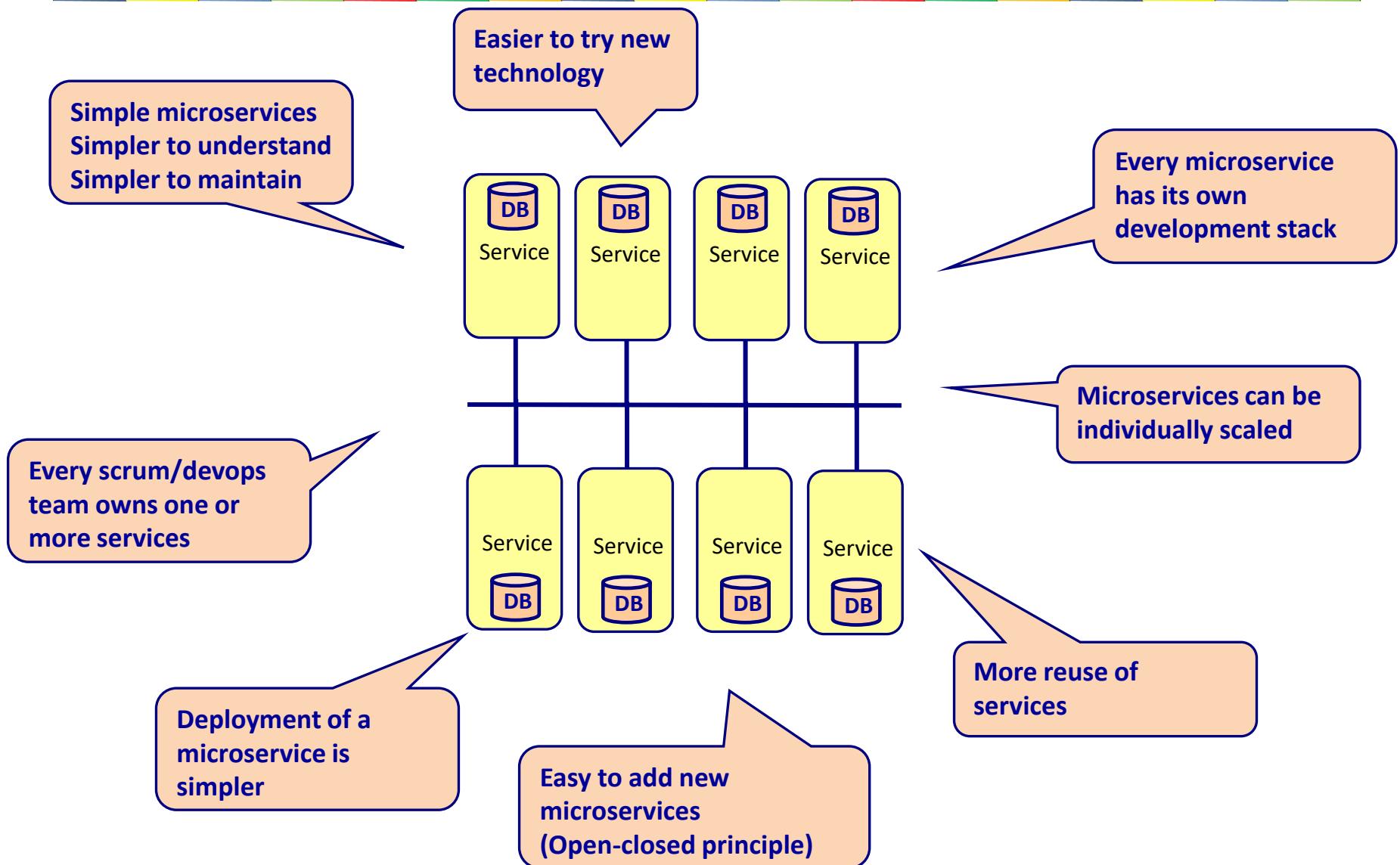
Microservices can be
individually scaled

Deployment of a
microservice is
simpler

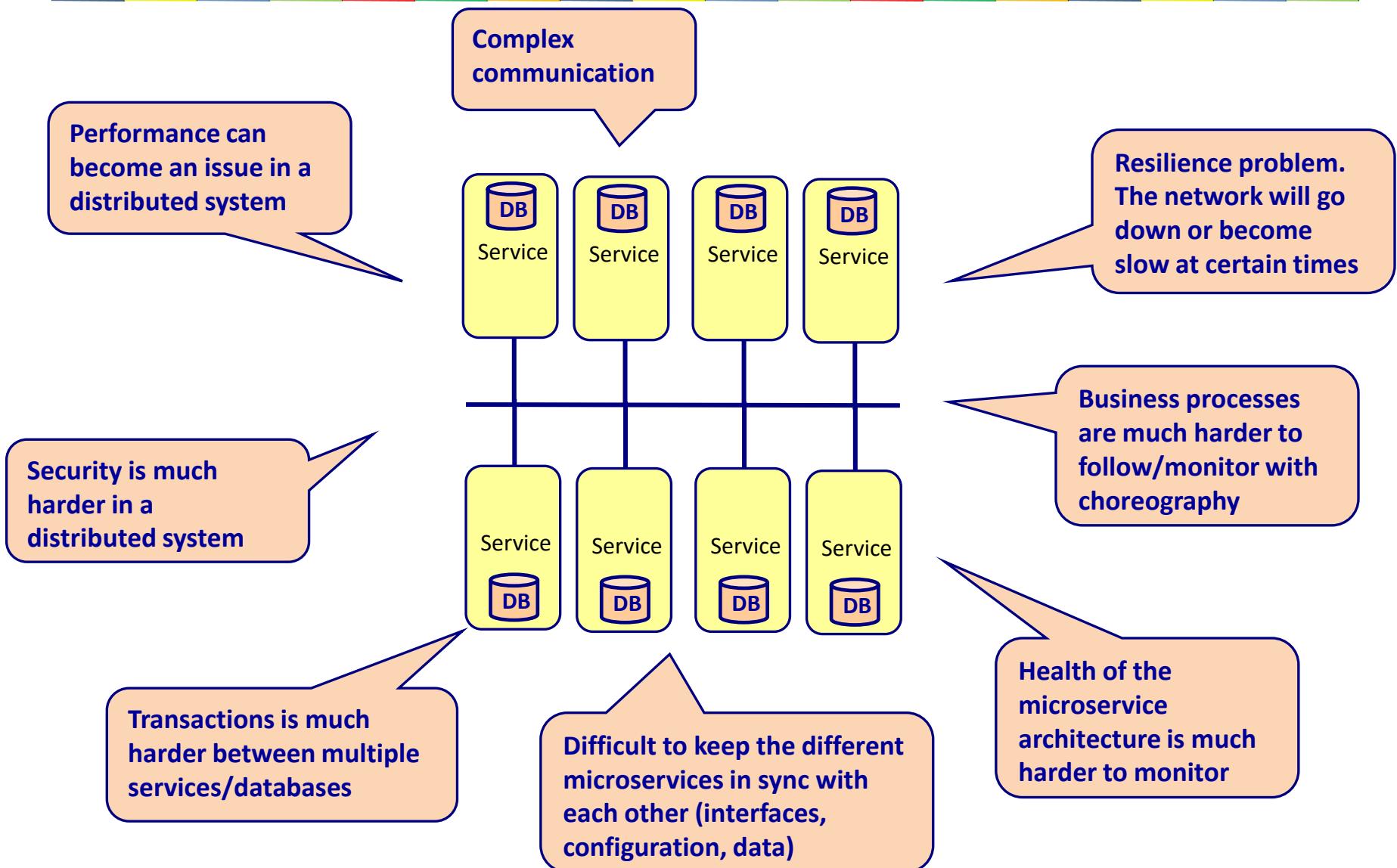
More reuse of
services

ADVANTAGES AND DISADVANTAGES OF A MICROSERVICE ARCHITECTURE

Advantages



Disadvantages

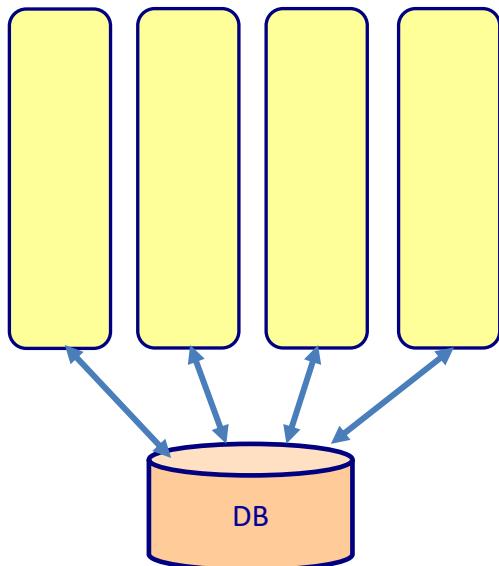


Challenges of a microservice architecture

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	

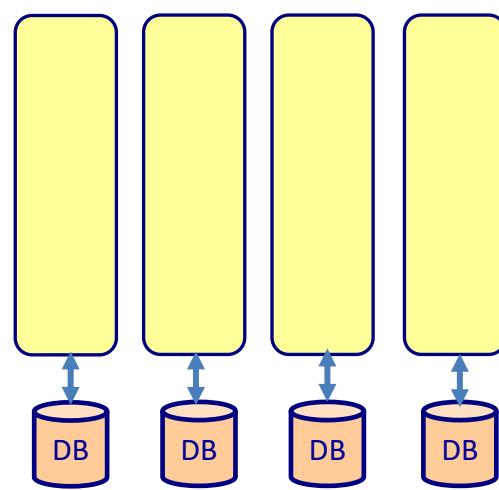
MICROSERVICE AND DATABASES

Every service manages its own data



Tight coupling

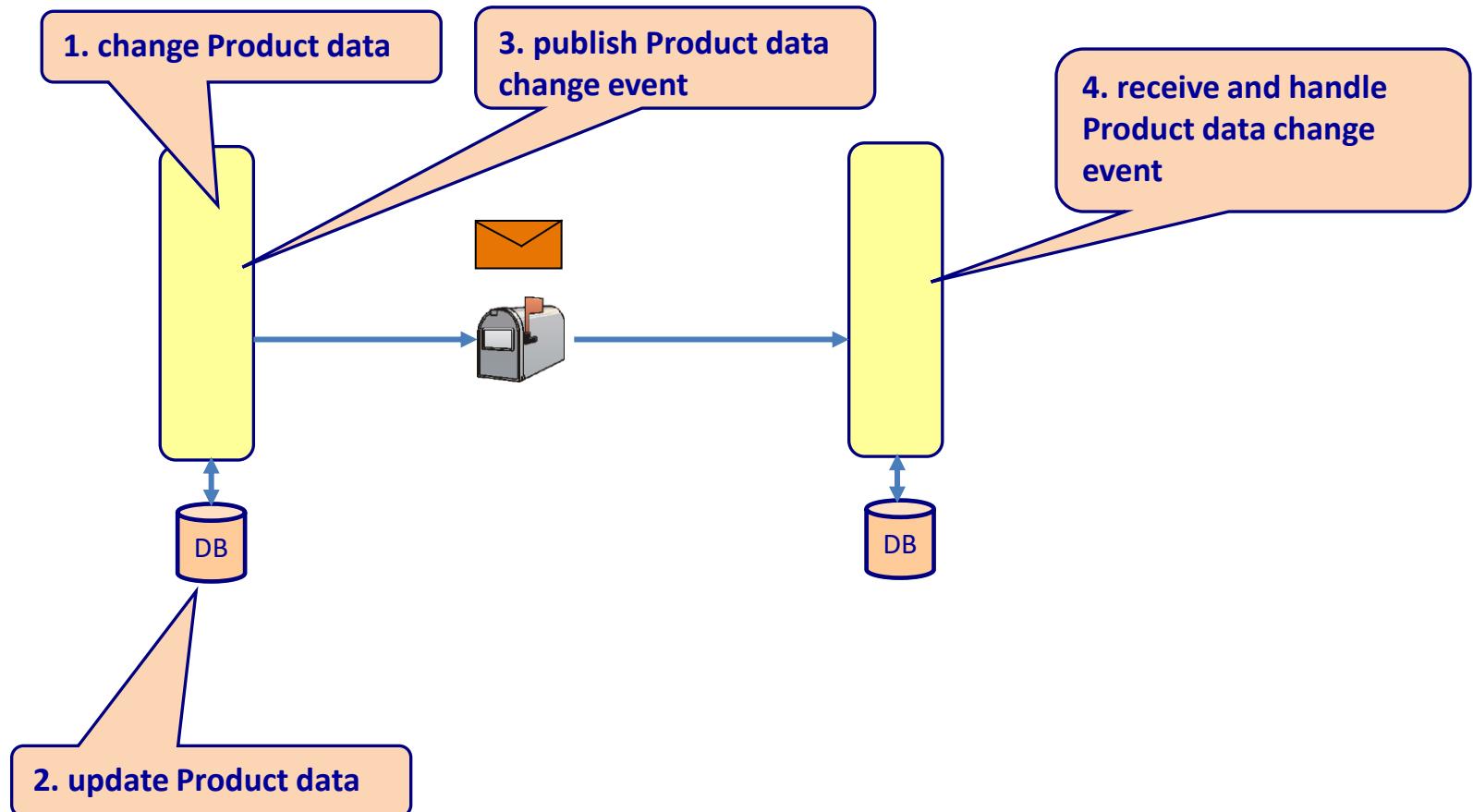
Easy to manage data



Loose coupling

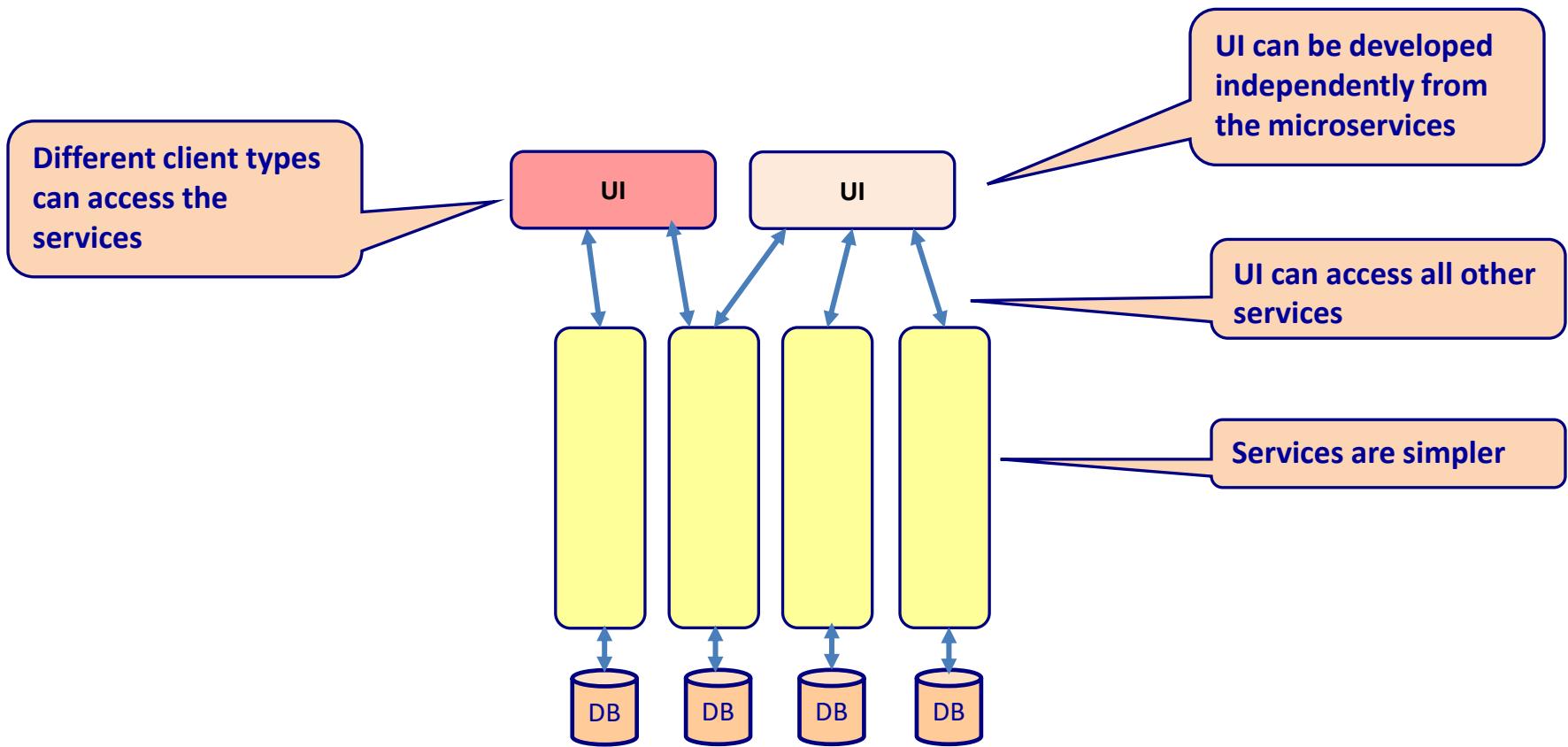
Hard to manage data

Data consistency

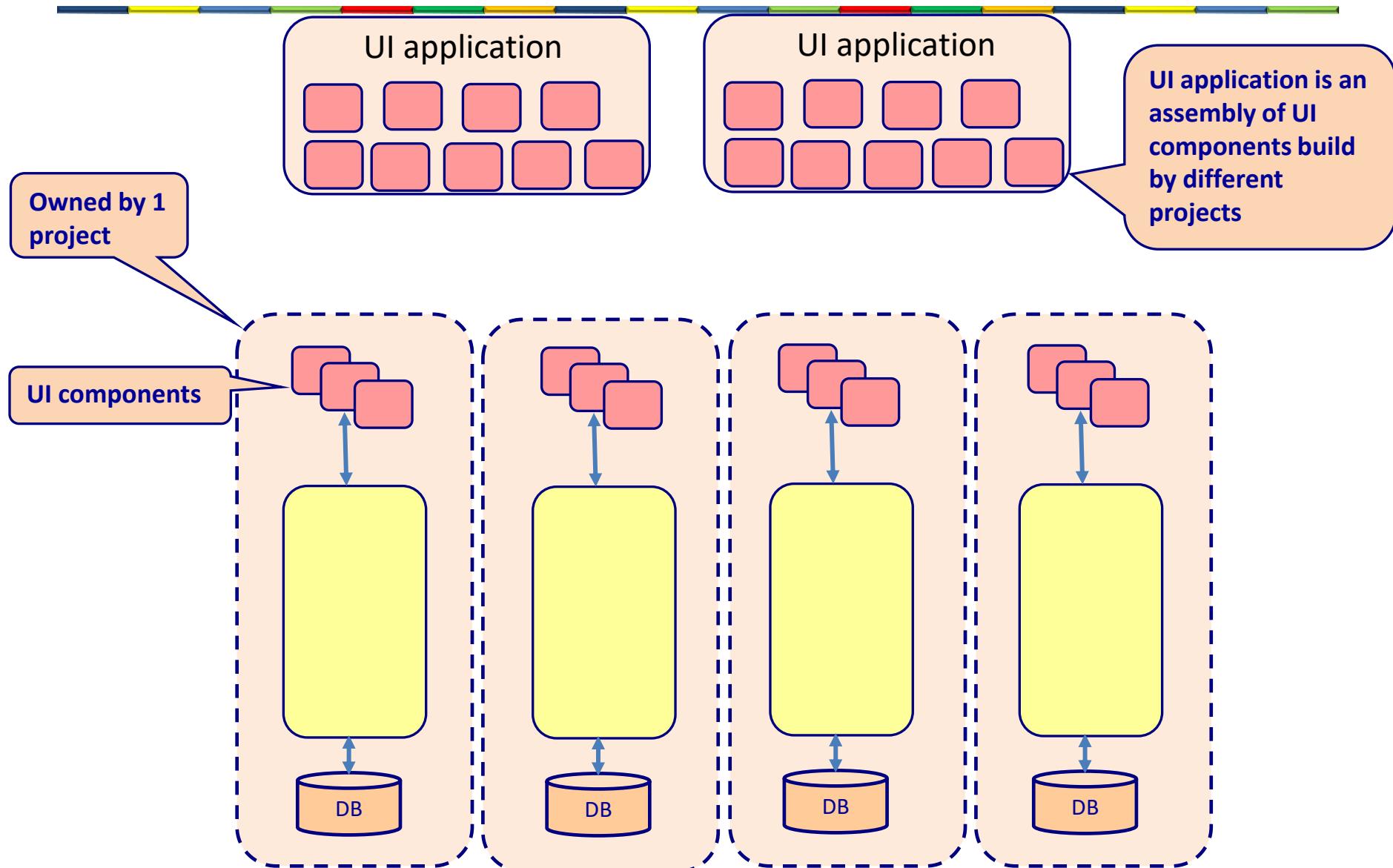


UI AND MICROSERVICE

Split front-end and back-end



Micro frontend



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

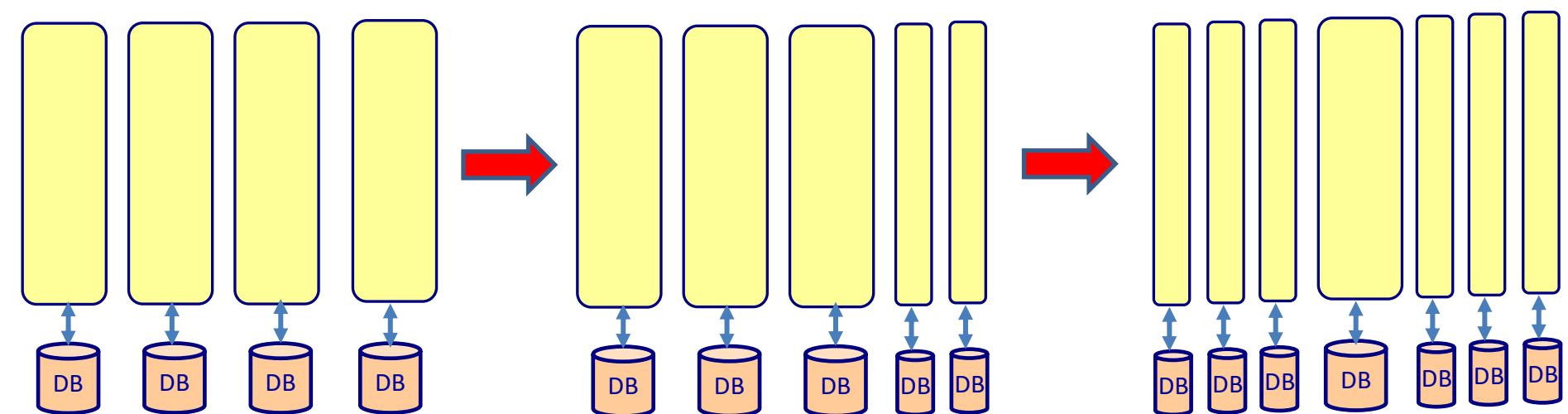
- Replicability or changeability
 - The microservice is easy detachable from the overall system
 - What functionality might evolve in the future?
- Coupling and cohesion
 - Avoid chatty services
 - Too many synchronous request
 - Transaction boundaries within one service

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

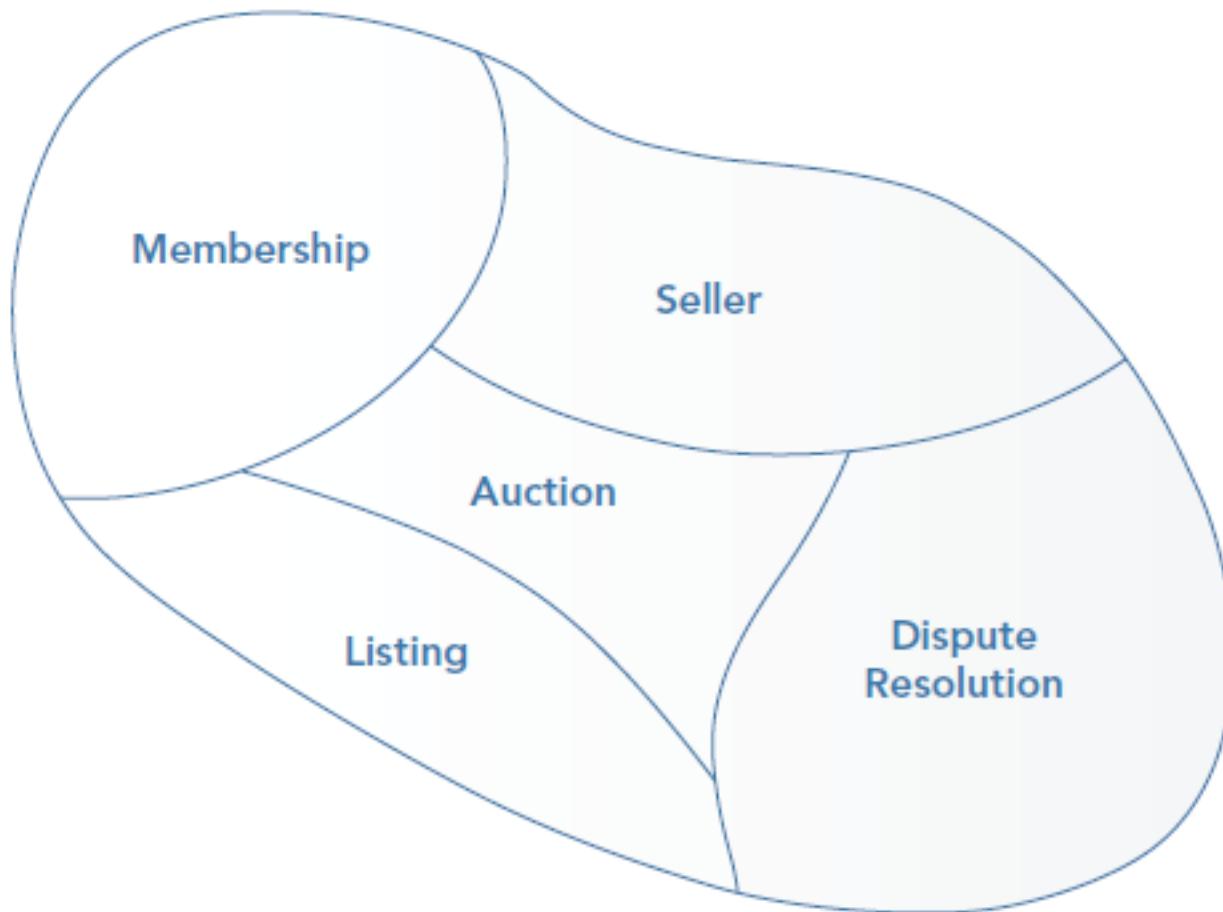
- Core subdomain
 - This is the reason you are writing the software.
- Supporting subdomain
 - Supports the core domain
- Generic subdomain
 - Very generic functionality
 - Email sending service
 - Creating reports service

Distilling the domain

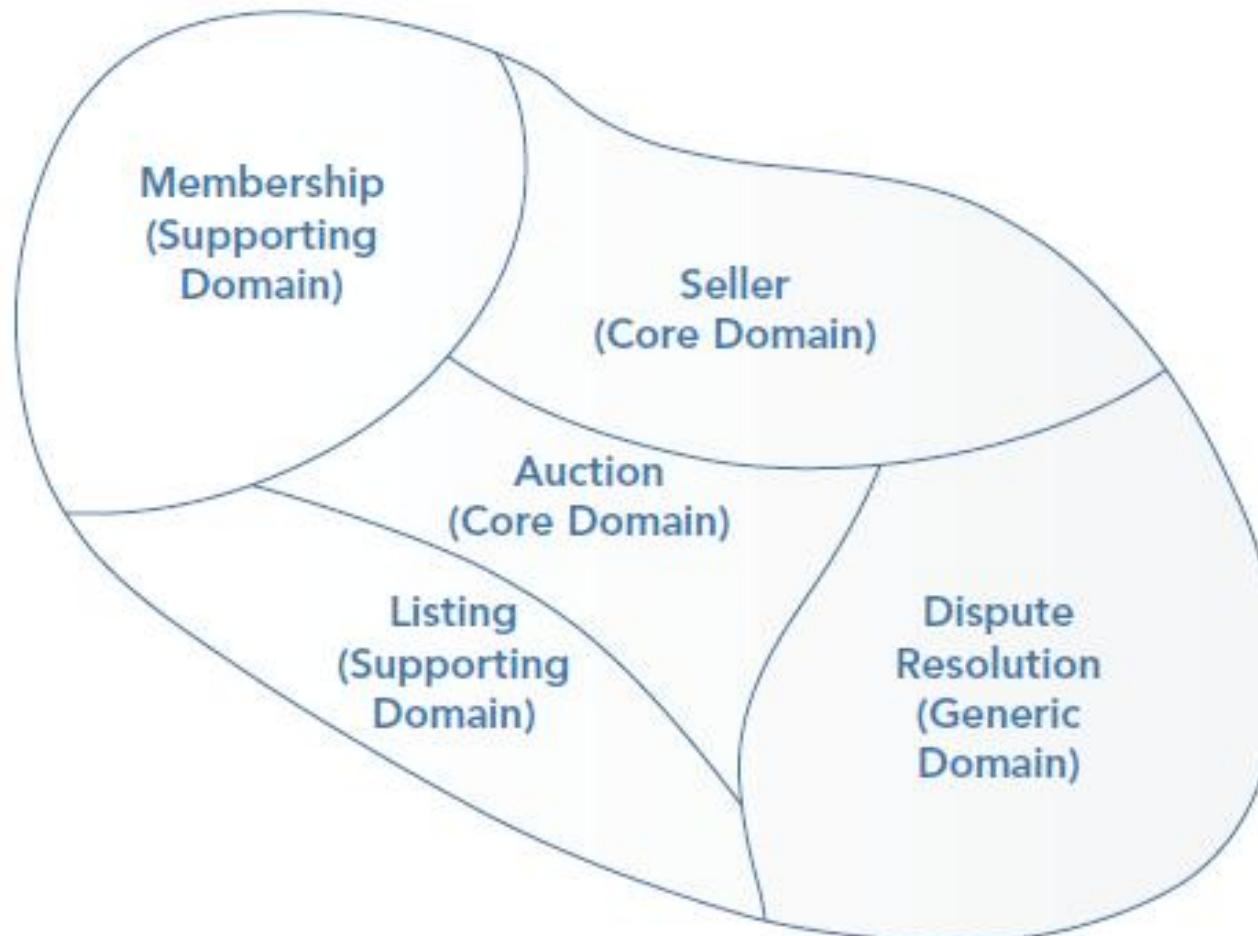
- The large domain of online auction



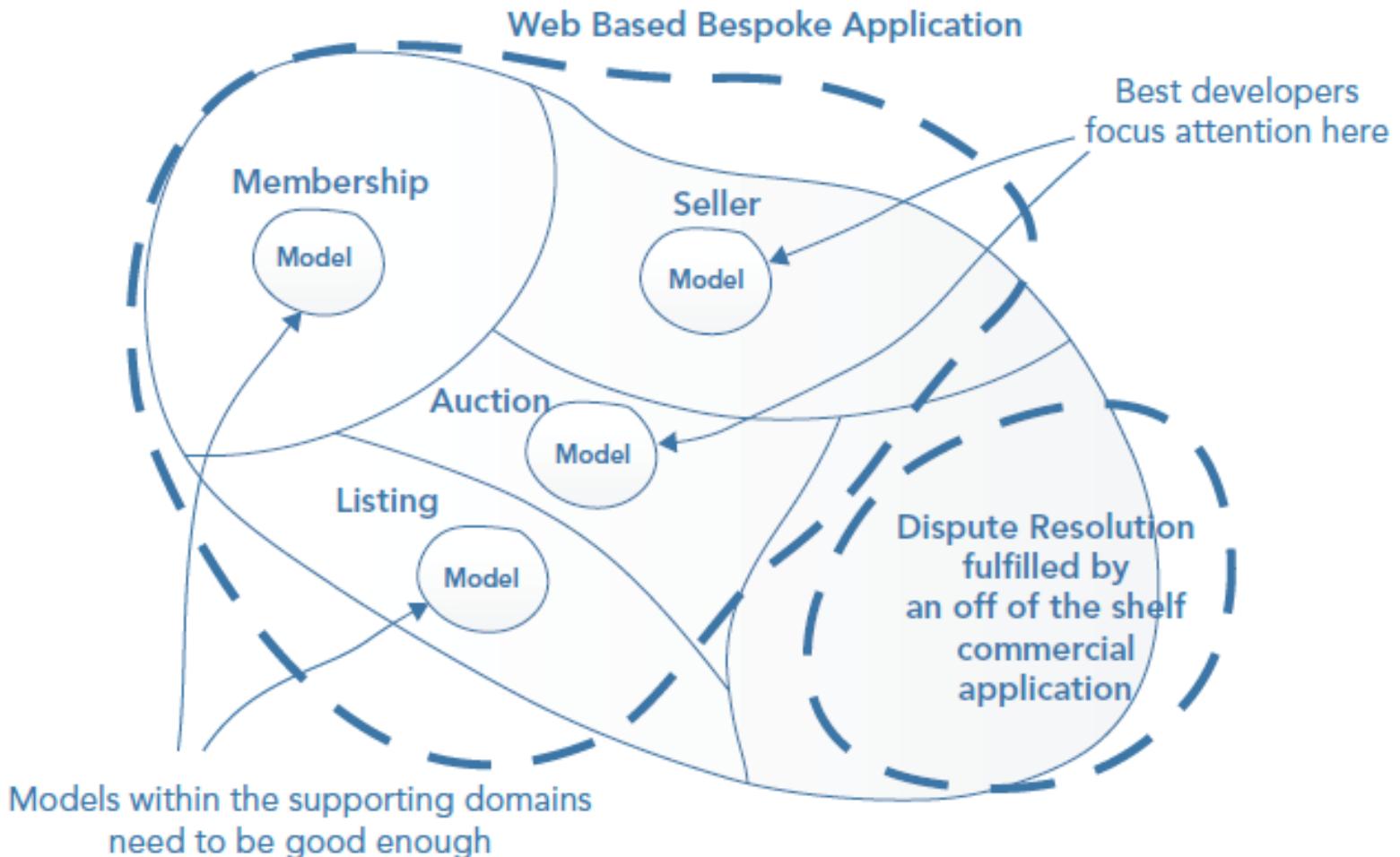
Find the subdomains



Identify the core domain



Subdomains shape the solution

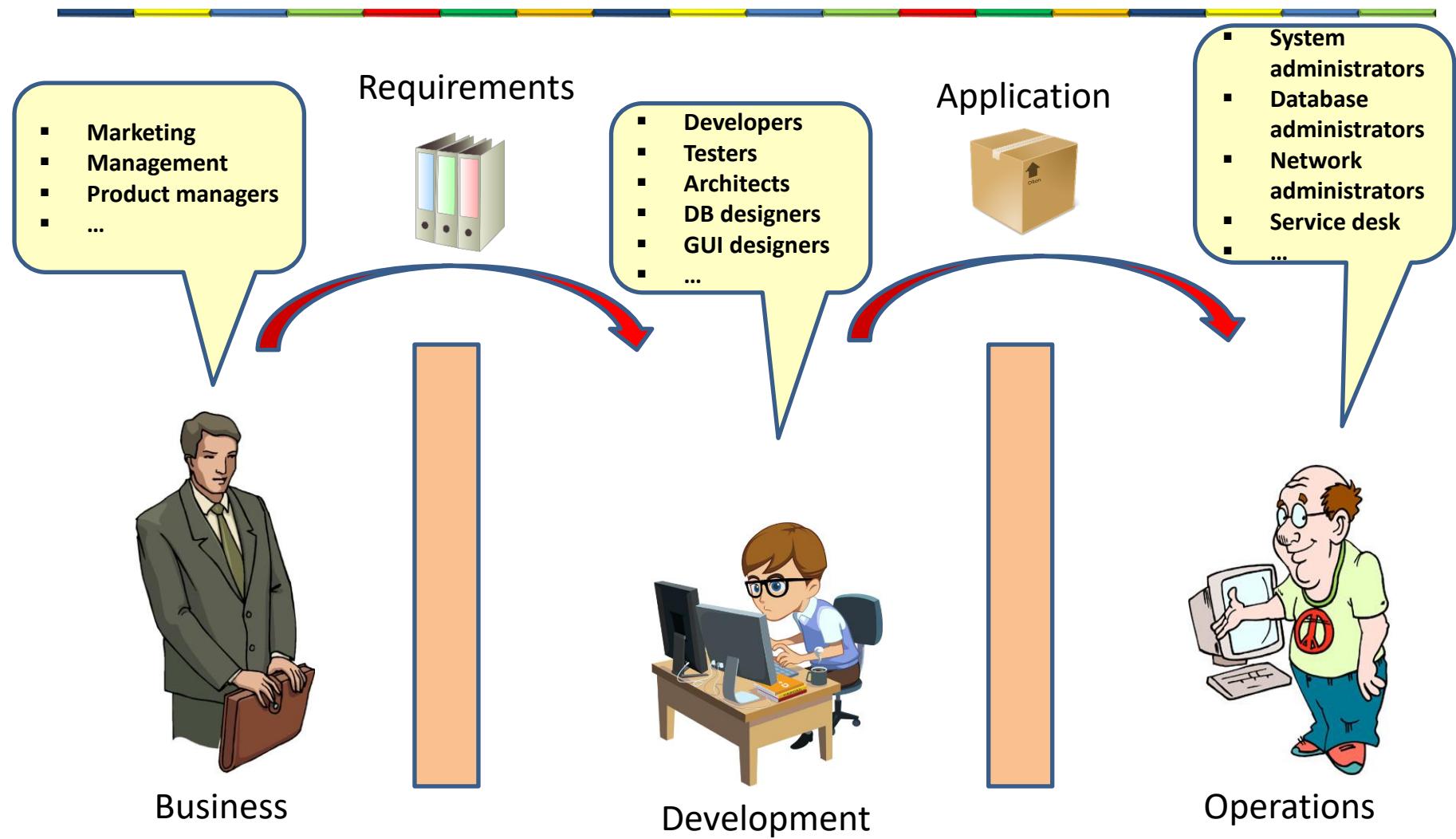


Main point

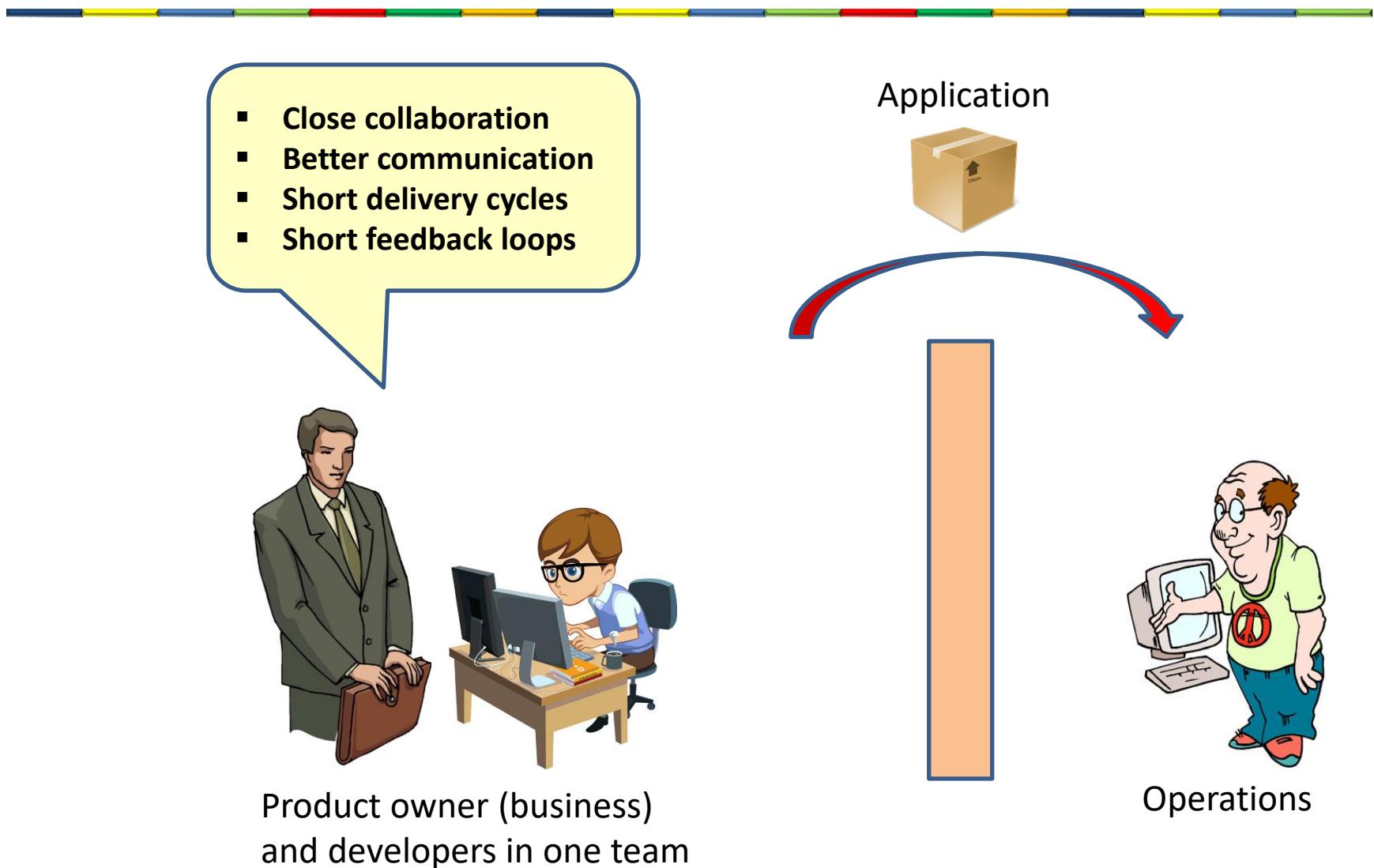
- An ideal architecture does not exist. An microservice architecture has its own advantages and disadvantages. It is almost impossible to transform every application into microservices.
- Water the root and enjoy the fruit. Problems are hard to solve at the level of the problem. It is much easier to solve problems at its root.

MICROSERVICES IN THE ORGANIZATION

Traditional software development



Agile software development: Scrum



DevOps

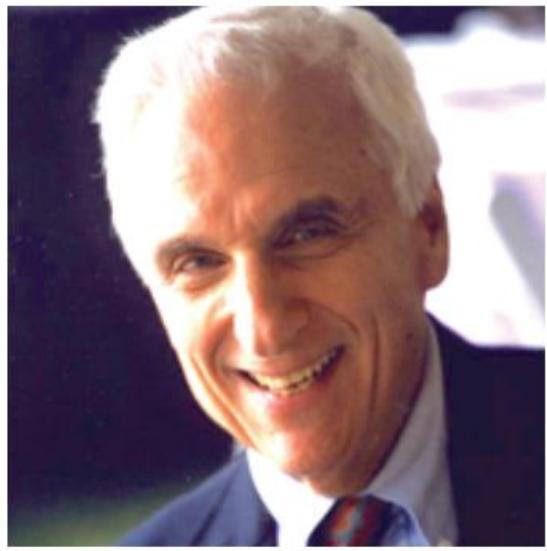
- Close collaboration between developers and operations
- Streamlines the delivery process of software from business requirements to production
- Better communication
- Identical development and production environment
- Shared tools
 - Automate everything
 - Monitor everything



Product owner (business)
and developers in one team

Operations

Conways law



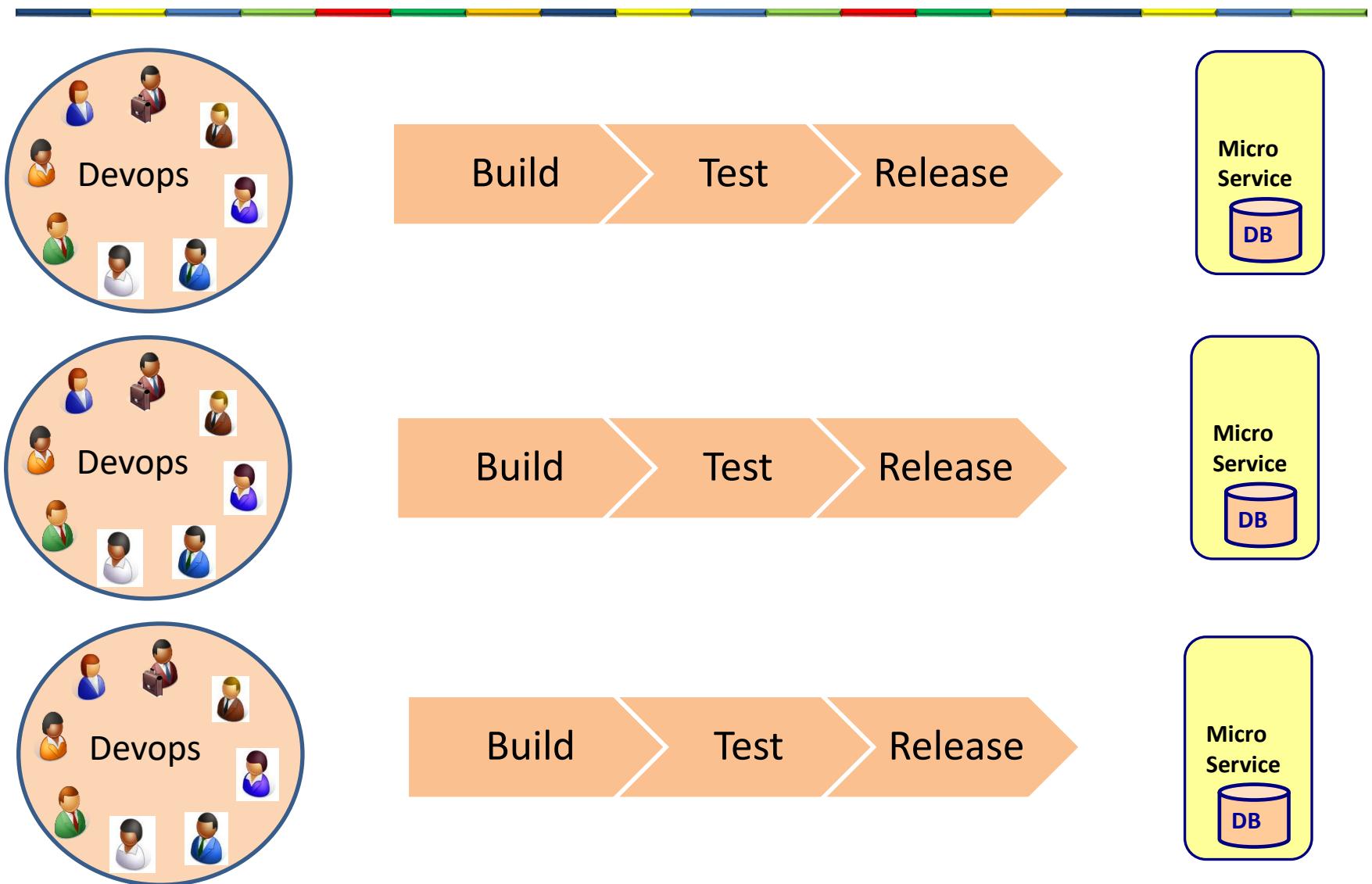
*"If you have four groups
working on a compiler, you'll
get a 4-pass compiler"*

—Eric S Raymond

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

—Melvin Conway

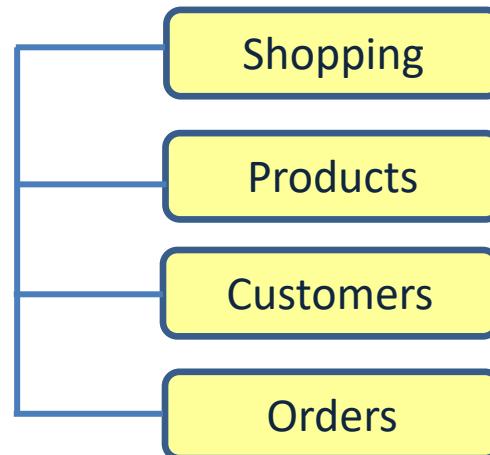
Microservice organization



CALLING ANOTHER MICROSERVICE: FEIGN

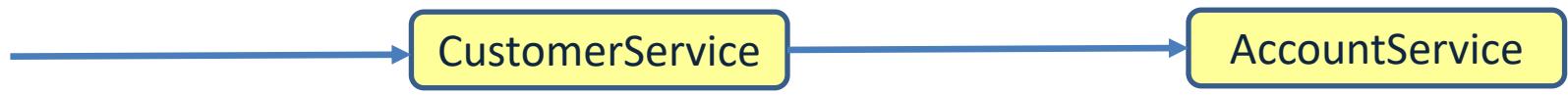
Implementing microservices

Calling another service



Calling another service

GET localhost:8091/customer/1



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

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

Use Feign

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

application.yml

```
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);  
    }  
}
```

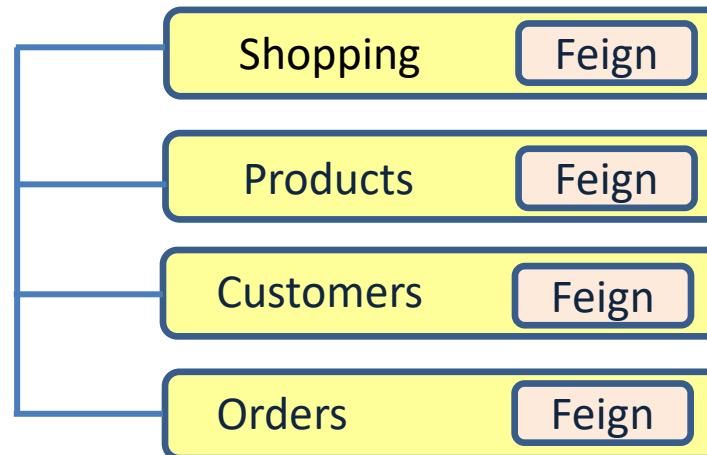
Autowire the client

Remote REST call

Declare the interface, Spring creates the implementation

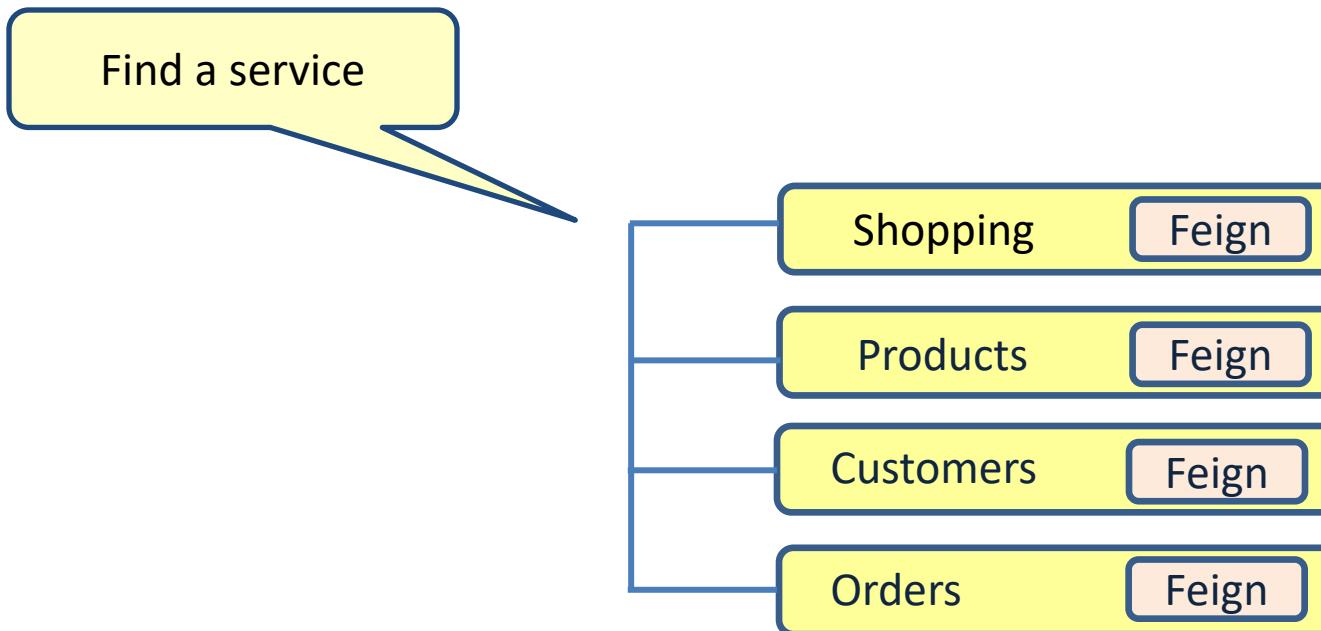
Implementing microservices

Calling another service

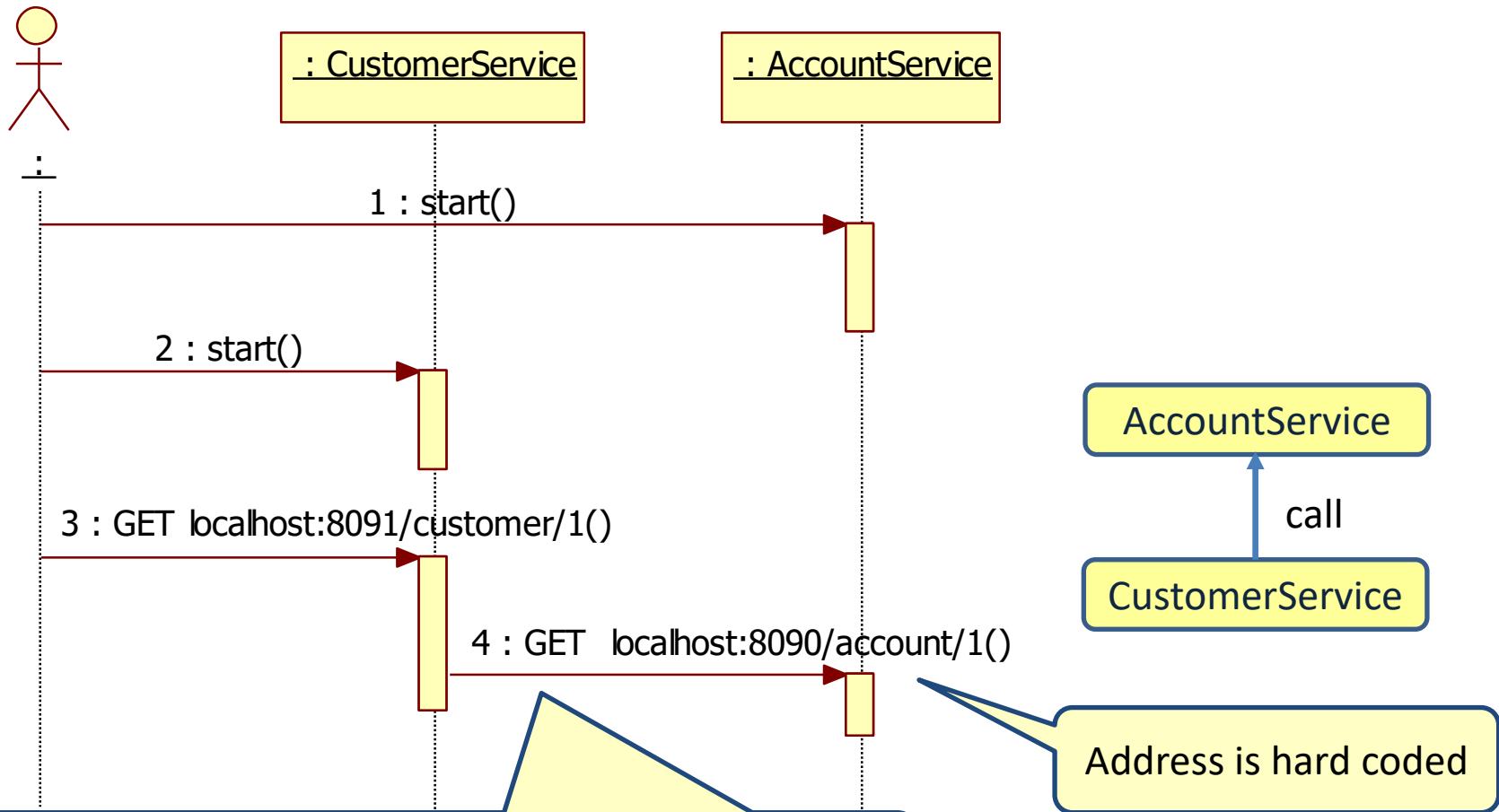


SERVICE REGISTRY: CONSUL

Implementing microservices



One service calling another service

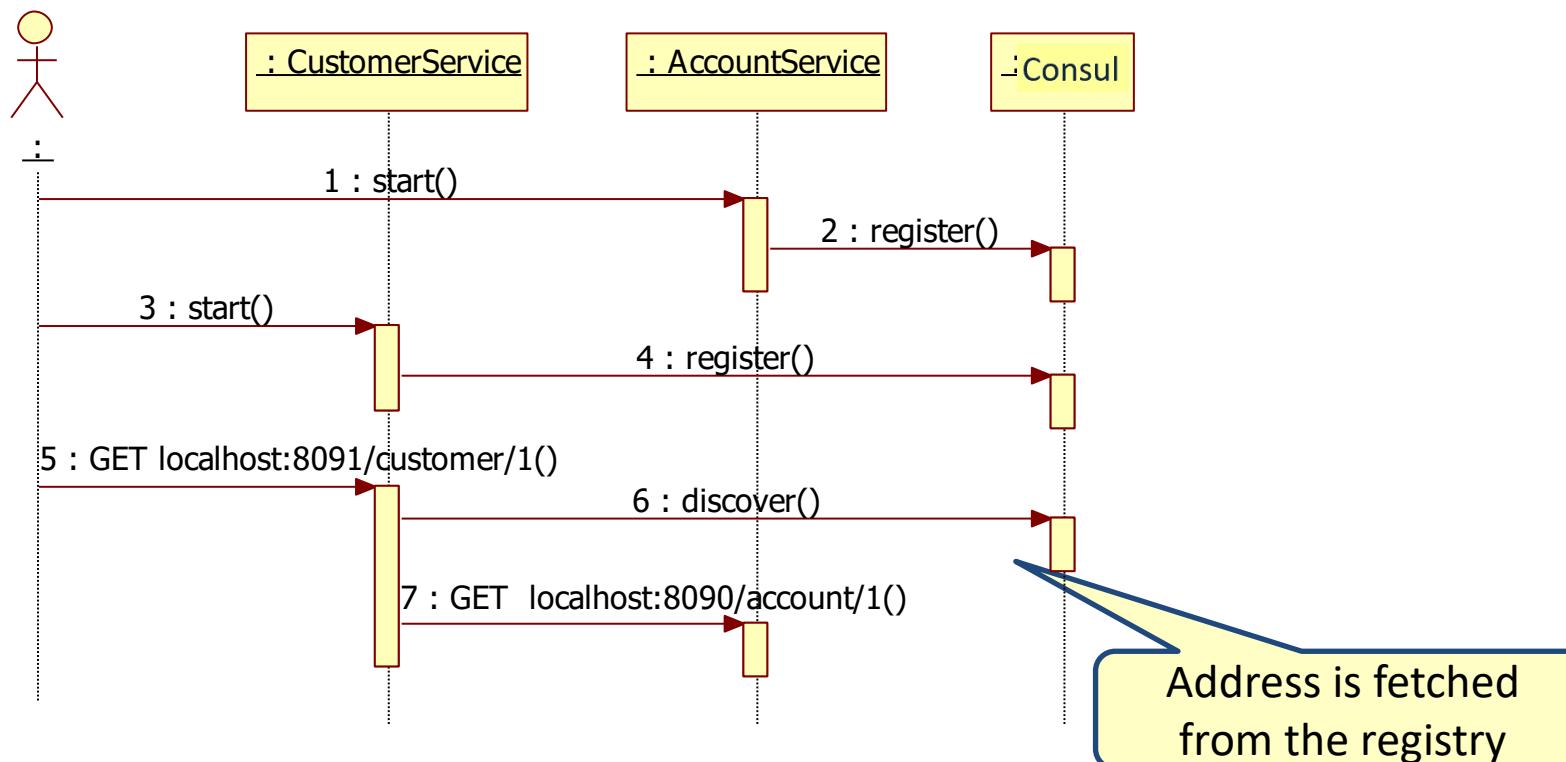
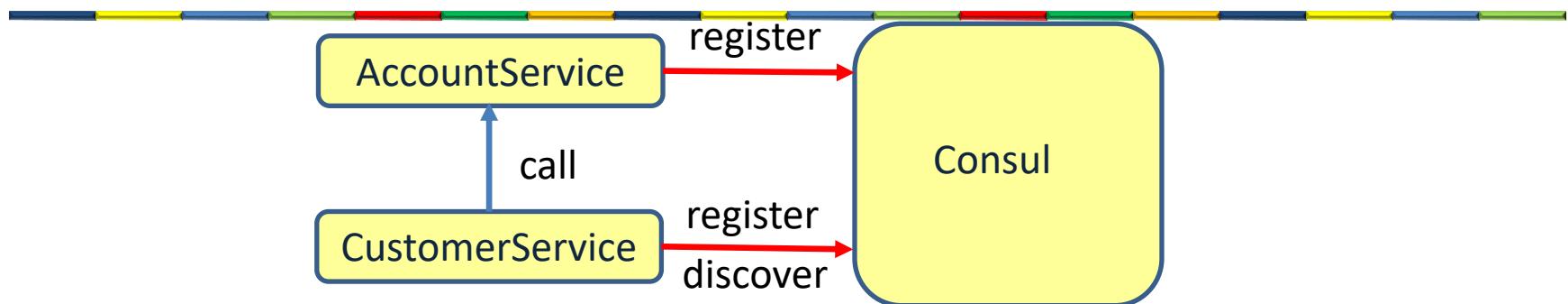


If I change the URL of the AccountService, I need to change the configuration of the CustomerService

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



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

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

The service will register itself in the registry

```
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

The screenshot shows the Consul UI interface. On the left, there's a sidebar with icons for Overview, Services (which is selected), Nodes, Key/Value, and Intentions. The main area has a header with a search bar and filters for Health Status (set to 'consul'), Service Type, and a sorting dropdown ('Unhealthy to Healthy'). Below this, two services are listed: 'consul' (1 instance) and 'Accountservice' (1 instance). Both services are marked with a green checkmark.

Service	Status	Instances
consul	Healthy	1 instance
Accountservice	Healthy	1 instance

CustomerService

```
@SpringBootApplication  
@EnableDiscoveryClient  
@EnableFeignClients
```

Use Feign and the Registry

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

```
<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

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

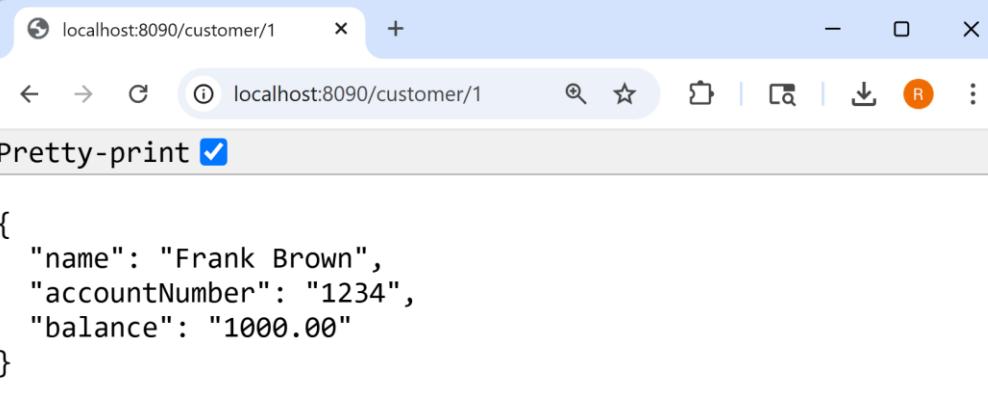
By default, Consul will do a health check every 10 seconds

```
server:  
  port: 8090
```

Running the CustomerService

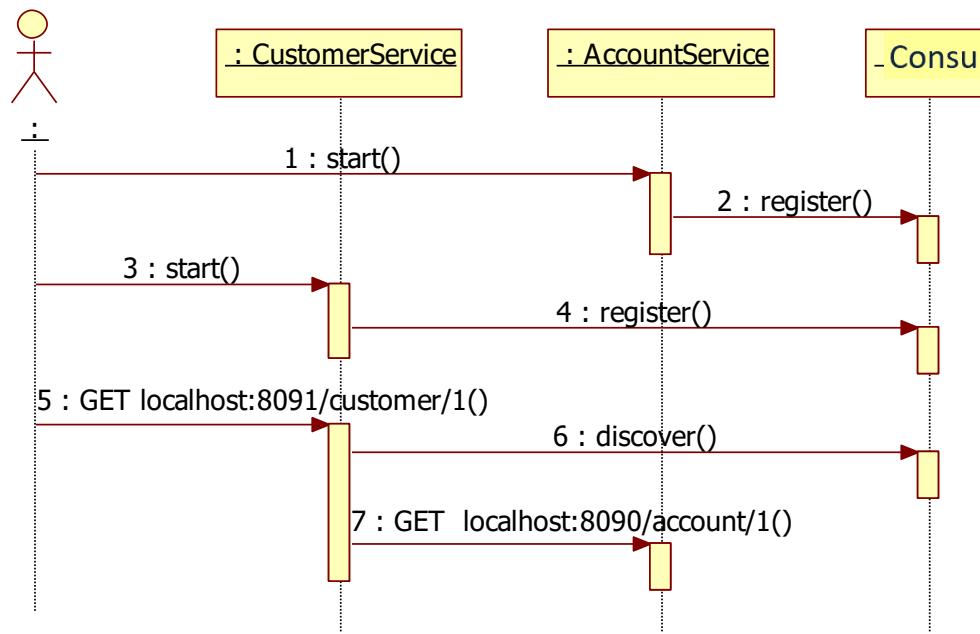
The screenshot shows the Consul UI interface. On the left, there's a sidebar with navigation links: Overview, Services (which is selected and highlighted in grey), Nodes, Key/Value, Intentions, Access Controls, and Tokens. The main area is titled "Services 3 total". It includes a search bar and filters for Health Status (set to "consul"), Service Type (dropdown), and a sorting option "Unhealthy to Healthy" (dropdown). Below the filters, three services are listed: "consul" (1 instance, green checkmark), "Accountservice" (1 instance, green checkmark), and "CustomerService" (1 instance, green checkmark).

Calling the CustomerService



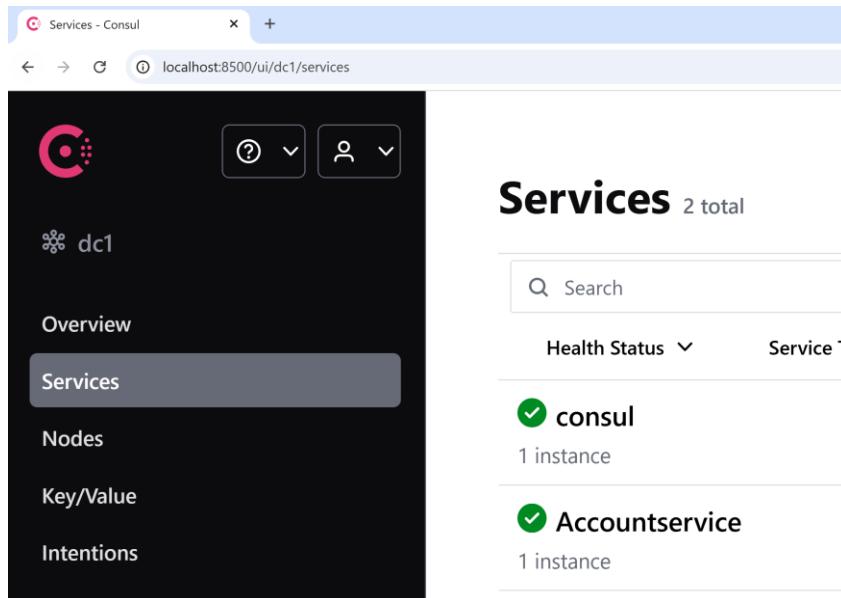
A screenshot of a web browser window. The address bar shows "localhost:8090/customer/1". The page content is a JSON object:

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



Stopping the CustomerService

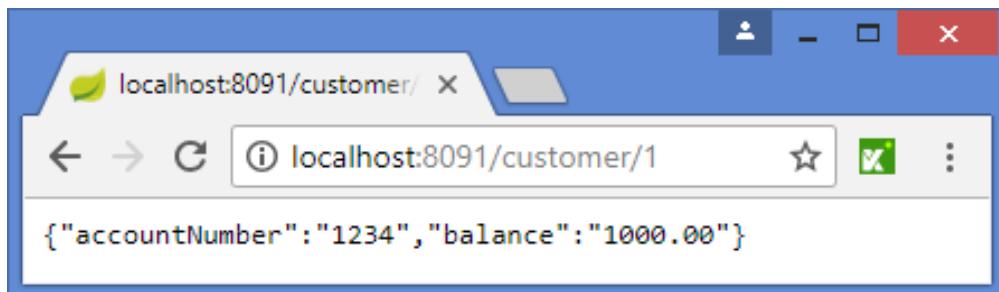
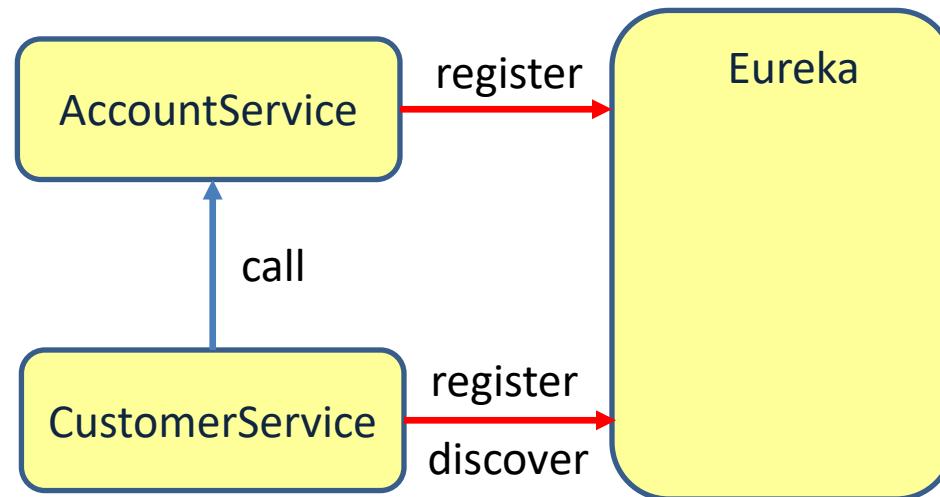
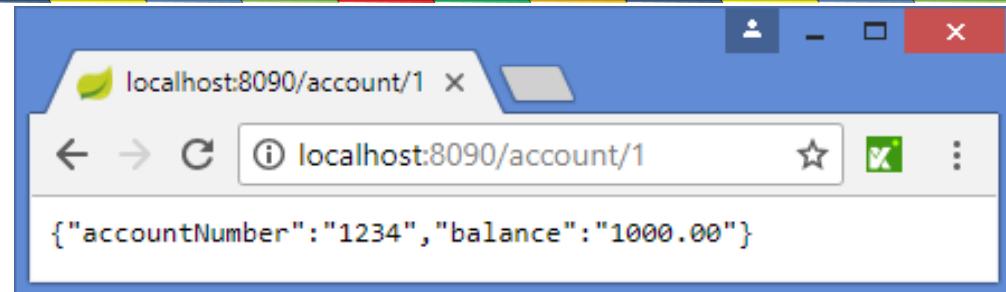
- Consul monitors the health of registered services.
- If we stop the CustomerService, Consul will notice that automatically



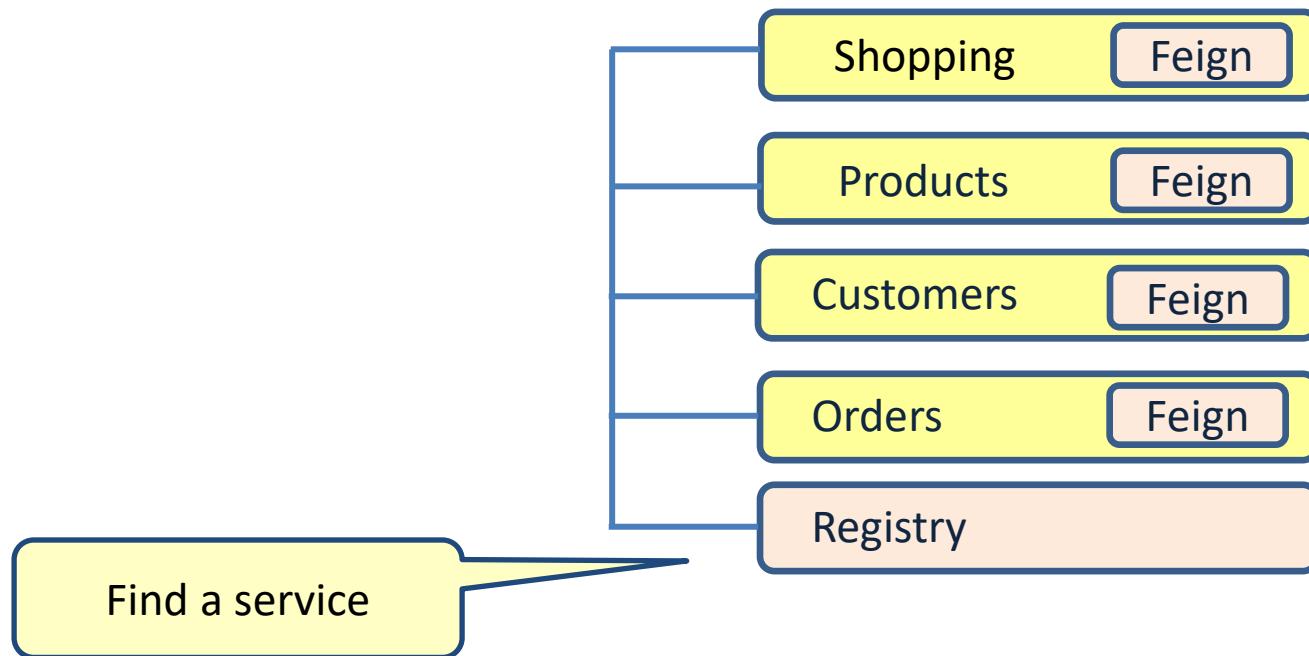
The screenshot shows the Consul UI interface. On the left, there's a sidebar with navigation options: Overview, Services (which is selected and highlighted in grey), Nodes, Key/Value, and Intentions. The main area is titled "Services 2 total". It includes a search bar and filters for "Health Status" and "Service T". Below the title, there are two entries: "consul" and "Accountservice", each with a green checkmark indicating they are healthy. Both entries show "1 instance".

Service	Status	Instances
consul	Healthy	1 instance
Accountservice	Healthy	1 instance

Using Eureka



Implementing microservices



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

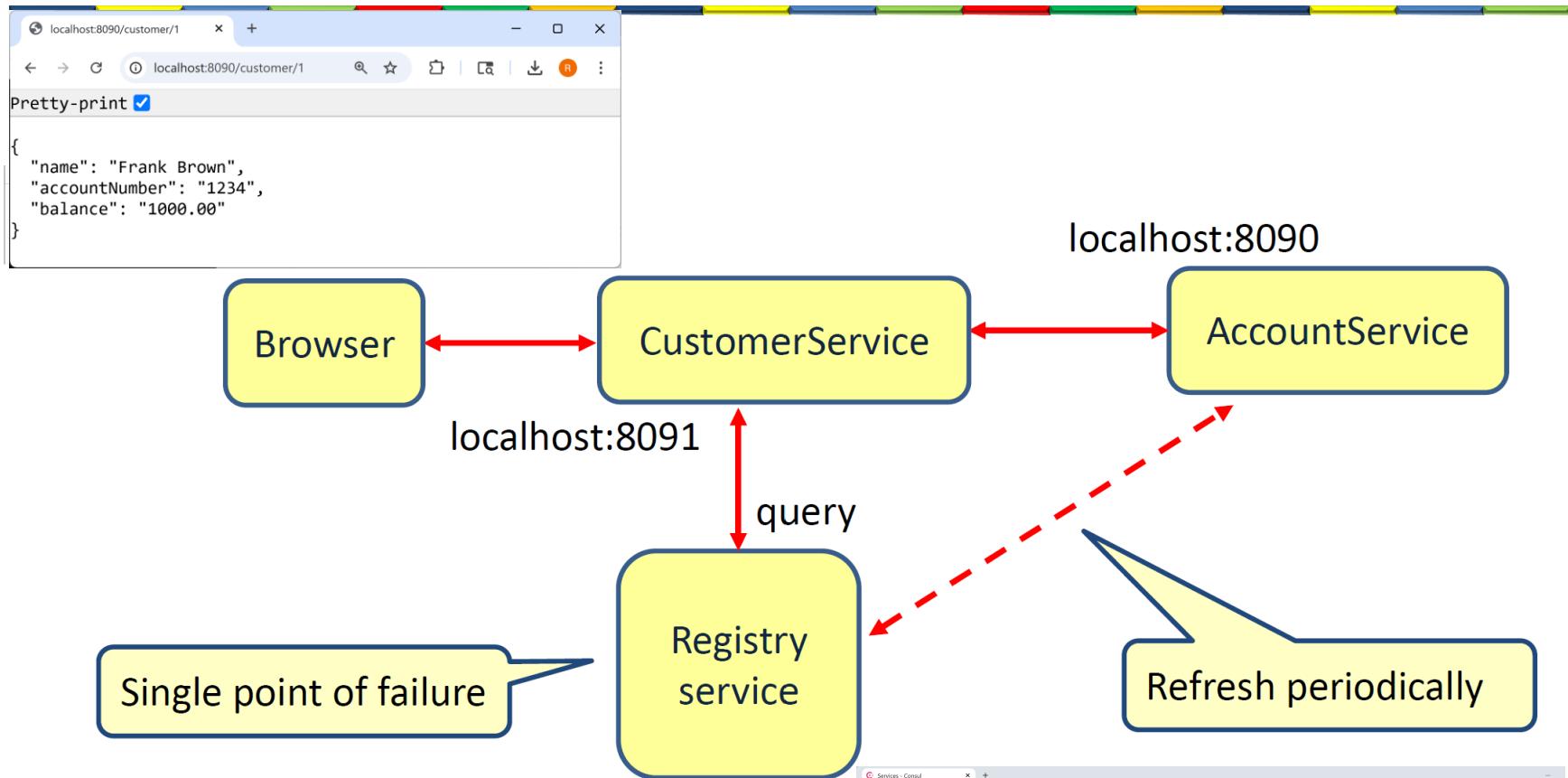
1. A microservice is an autonomous application owned by 1 team.
2. A microservice architecture is a distributed architecture which is complex by nature.
3. **Transcendental consciousness** is the source from which the whole complex world is created.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that all distributed components in creation are just expressions of ones own Self.

Lesson 7

MICROSERVICES

SERVICE REGISTRY: CONSUL

Registry



Consul cluster



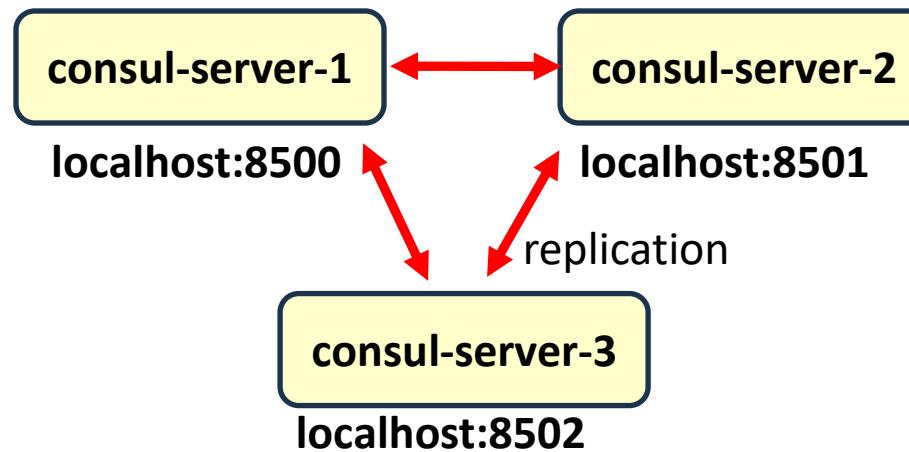
```
version: '3.8'
services:
  consul-server-1:
    image: hashicorp/consul:latest
    container_name: consul-server-1
    command: "agent -server -bootstrap-expect=3 -client=0.0.0.0 -ui -retry-join=consul-server-2 -retry-join=consul-server-3"
    ports:
      - "8500:8500" # UI and HTTP API
      - "8600:8600/udp" # DNS
    networks:
      - consul-net

  consul-server-2:
    image: hashicorp/consul:latest
    container_name: consul-server-2
    command: "agent -server -client=0.0.0.0 -ui -retry-join=consul-server-1 -retry-join=consul-server-3"
    ports:
      - "8501:8500" # UI and HTTP API (different host port to avoid conflict)
    networks:
      - consul-net

  consul-server-3:
    image: hashicorp/consul:latest
    container_name: consul-server-3
    command: "agent -server -client=0.0.0.0 -ui -retry-join=consul-server-1 -retry-join=consul-server-2"
    ports:
      - "8502:8500" # UI and HTTP API (different host port to avoid conflict)
    networks:
      - consul-net

networks:
  consul-net:
    driver: bridge
```

Consul cluster



AccountService

application.yml

```
spring:  
  application:  
    name: Accountservice  
  cloud:  
    consul:  
      host: localhost  
      port: 8501  
  
  server:  
    port: 8091
```

Register in Consul on port 8501

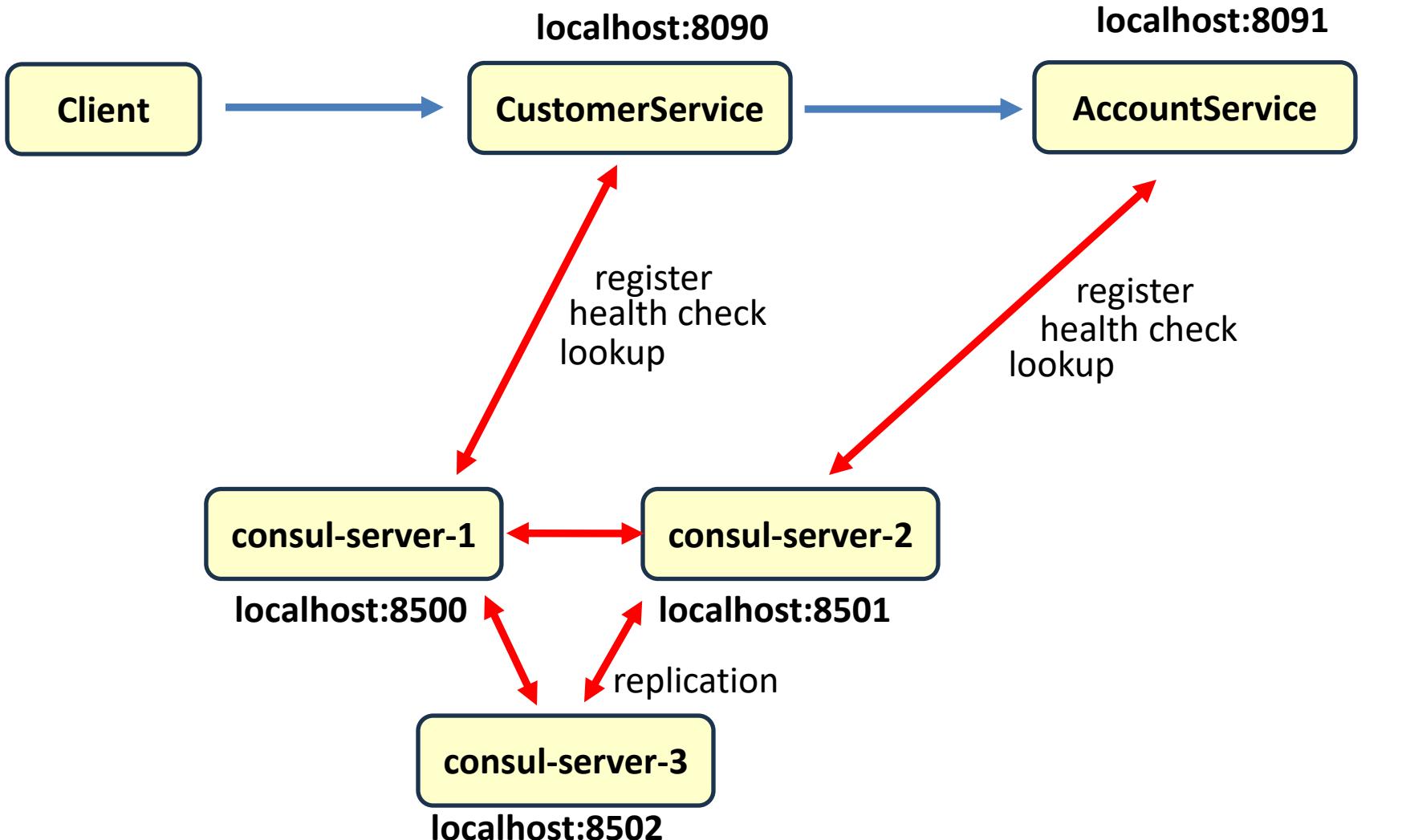
CustomerService configuration

application.yml

```
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
```

Register in Consul on port 8500

Consul cluster and replication



Replication

The image displays three separate browser windows, each showing the Consul UI for a data center named 'dc1'. Each window shows a list of services: 'Accountservice' (1 instance), 'Customerservice' (1 instance), and 'consul' (3 instances). The 'Services' tab is selected in all three windows.

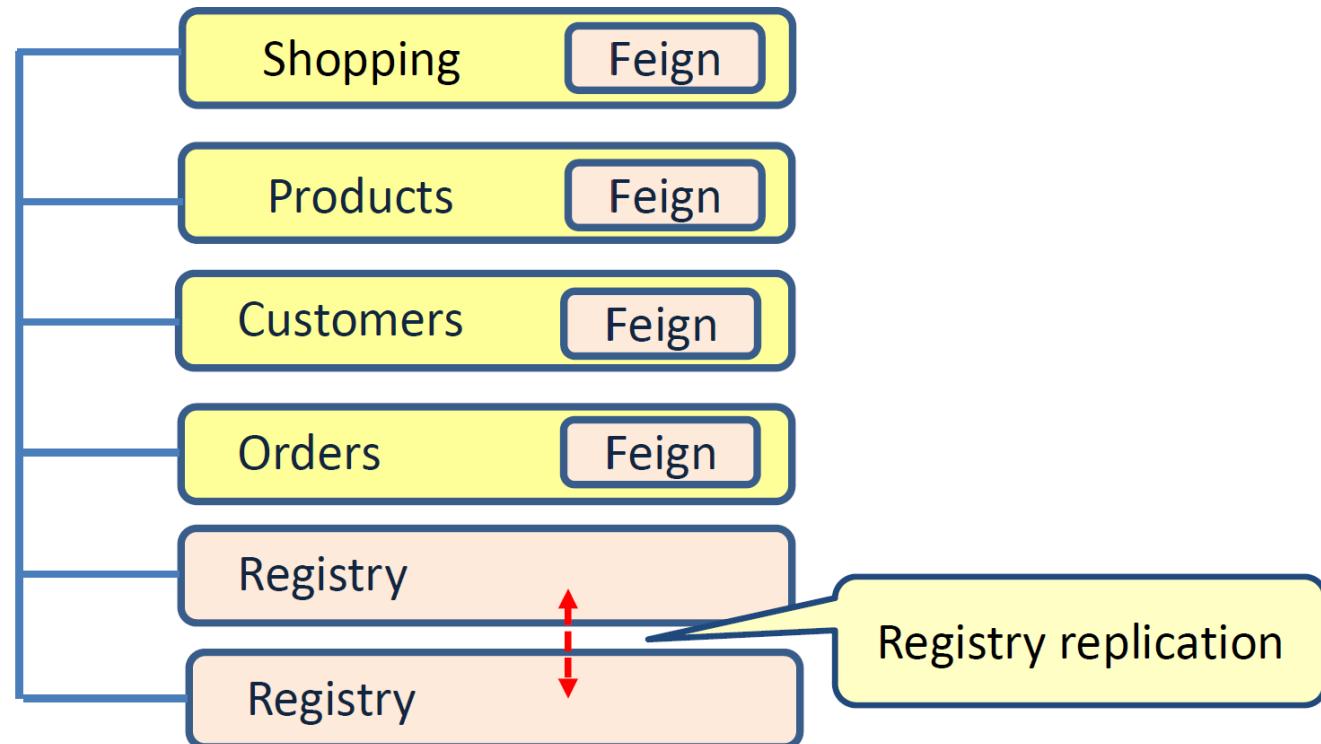
- Top Left Window:** Shows 'Services' 3 total. The 'consul' service has 3 instances listed.
- Top Right Window:** Shows 'Services' 3 total. The 'consul' service has 3 instances listed. A 'Saved to this PC.' button is visible in the top right corner.
- Bottom Window:** Shows 'Services' 3 total. The 'consul' service has 3 instances listed.

Running the CustomerService

The screenshot shows the Consul UI interface running at `localhost:8500/ui/dc1/services`. The left sidebar has tabs for Overview, Services (which is selected), Nodes, Key/Value, Intentions, Access Controls, and Tokens. The main area displays the services page with the following details:

Service	Health Status	Service Type	Instances
consul	Green (Healthy)	Consul	1 instance
Accountservice	Green (Healthy)	Accountservice	1 instance
CustomerService	Green (Healthy)	CustomerService	1 instance

Implementing microservices

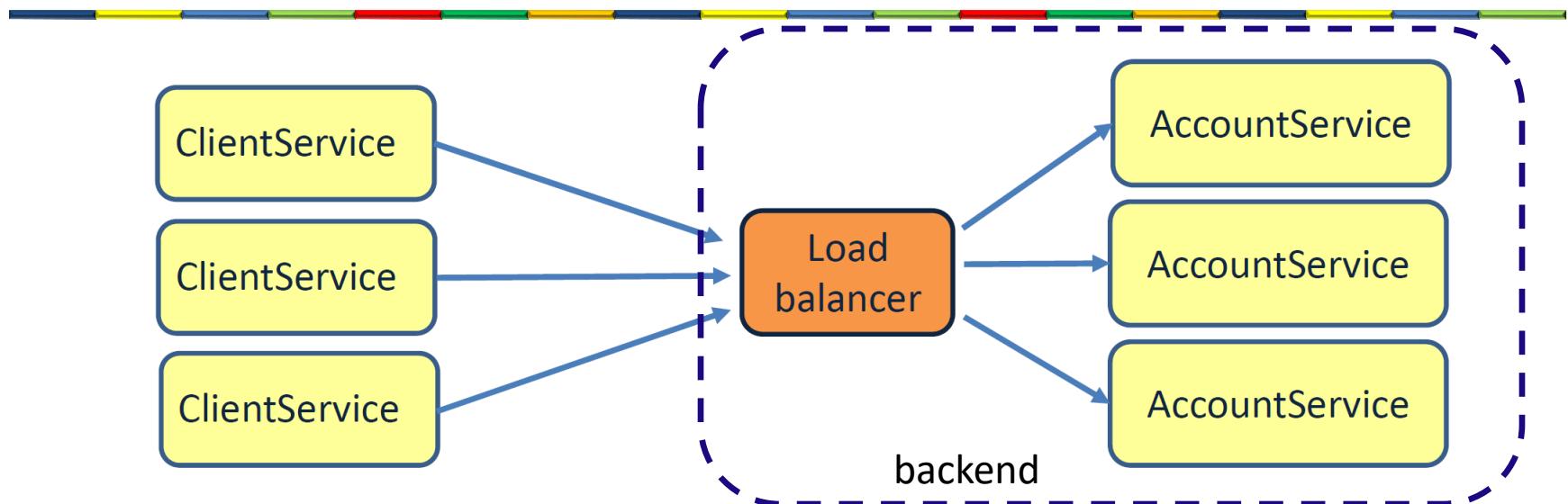


Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry
Performance	
Resilience	Registry and replicas
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	

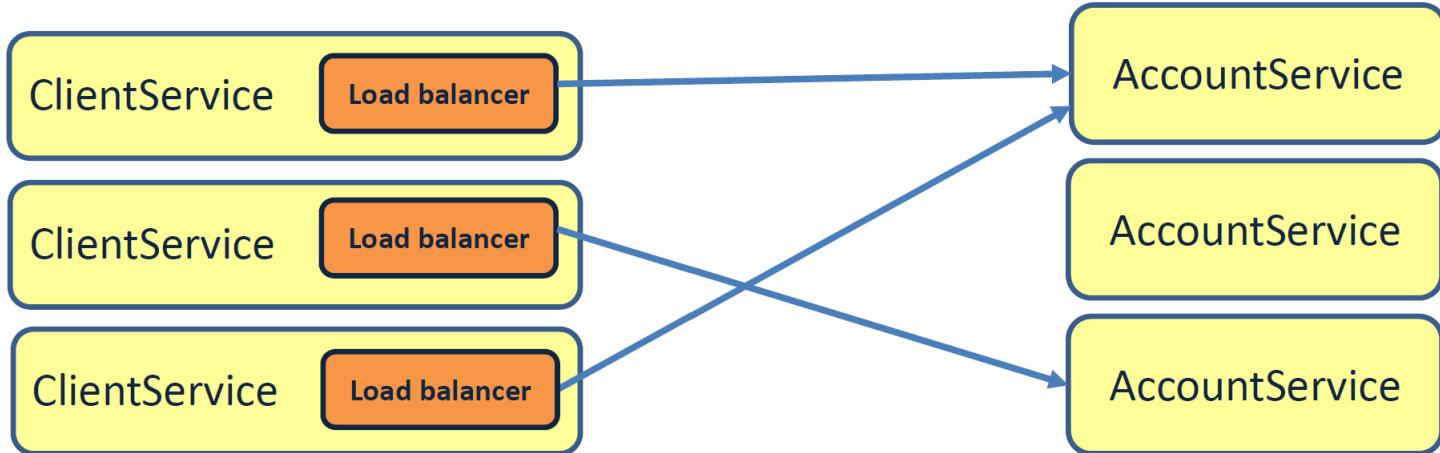
LOAD BALANCING

Server side load balancing



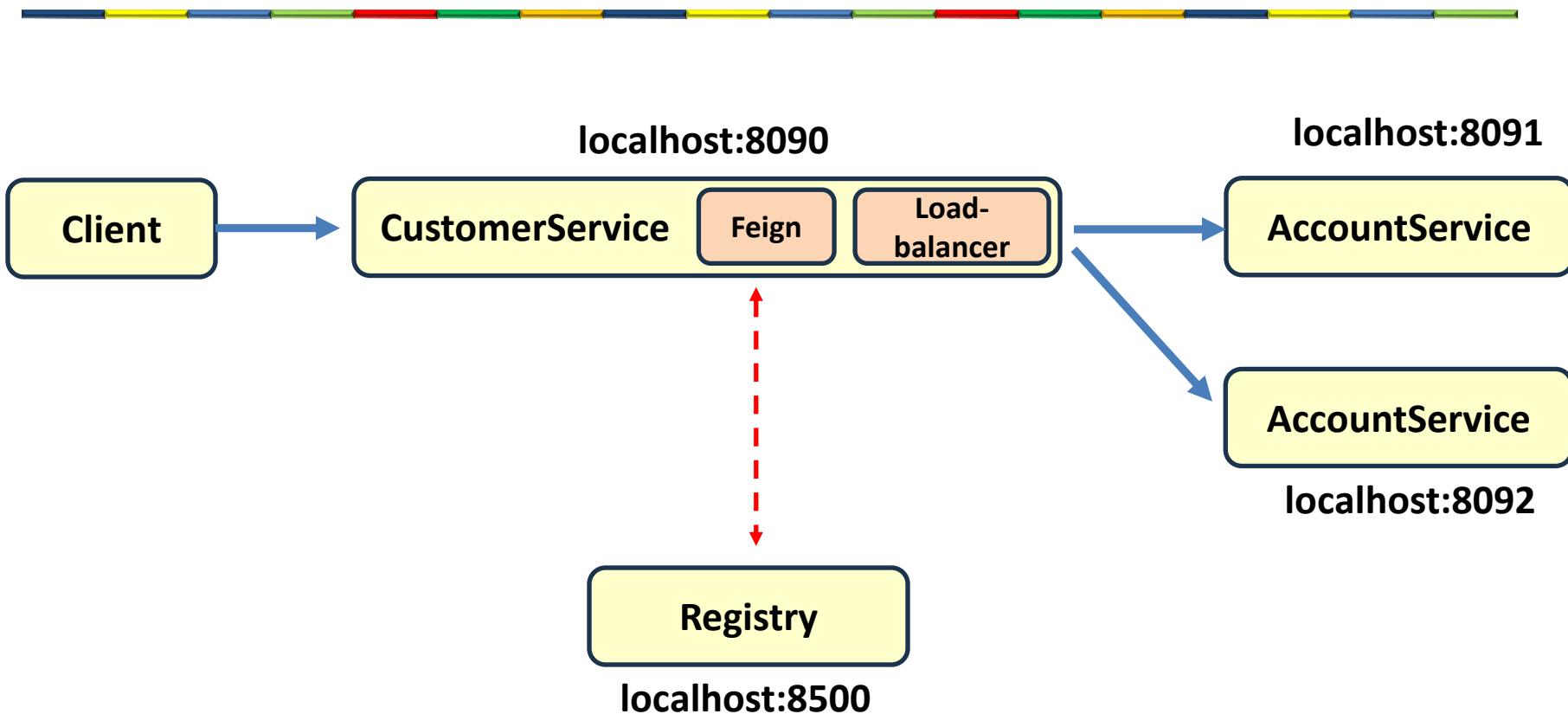
- Single point of failure
- If we add a new instance of AccountService, we need to reconfigure the load balancer
- Extra hop (performance)
- Every microservice needs its own load balancer
- Same load balance algorithm for every client
- Scaling limitation, load balance can handle only a certain number of requests

Client side load balancing



- No single point of failure
- Simplifies service management
- Only one hop (performance)
- Auto discovery with registry based lookup (flexibility)
- Every client can use its own load balancing algorithm
- Unlimited scalable

Spring cloud load balancer



Accountservice 1

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

1000.00

```
spring:  
application:  
    name: Accountservice  
cloud:  
    consul:  
        host: localhost  
        port: 8501
```

```
server:  
port: 8091
```

Same name, different port

Accountservice 2

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

2000.00

```
spring:  
application:  
    name: Accountservice  
cloud:  
    consul:  
        host: localhost  
        port: 8501
```

```
server:  
port: 8092
```

Same name, different port

Two account services

The screenshot shows the Consul UI interface. On the left, there is a sidebar with the following options:

- dc1 (selected)
- Overview
- Services (selected)
- Nodes
- Key/Value
- Intentions

The main area is titled "Services" and shows "3 total". It lists the following services:

- Customerservice**: 1 instance
- Accountservice**: 2 instances
- consul**: 3 instances

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);
    }
}
```

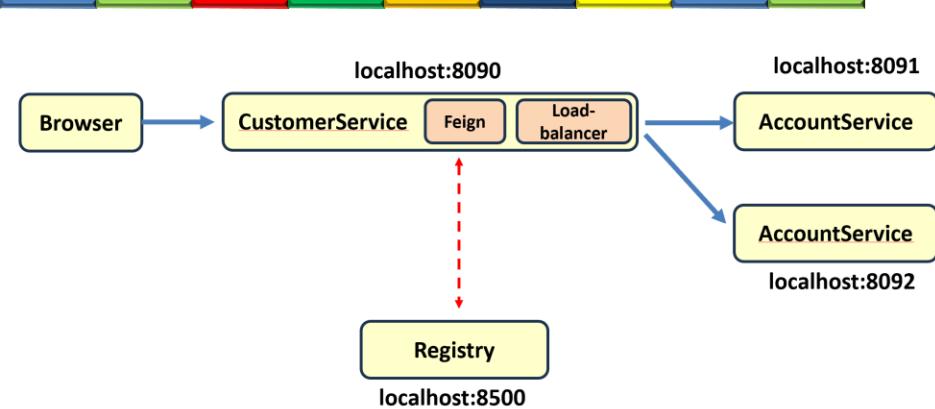
Feign automatically uses Spring cloud load balancer together with the Registry

Round robin load balancing

```
localhost:8090/customer/1
Pretty-print
{"name": "Frank Brown", "accountNumber": "1234", "balance": "1000.00"}
```

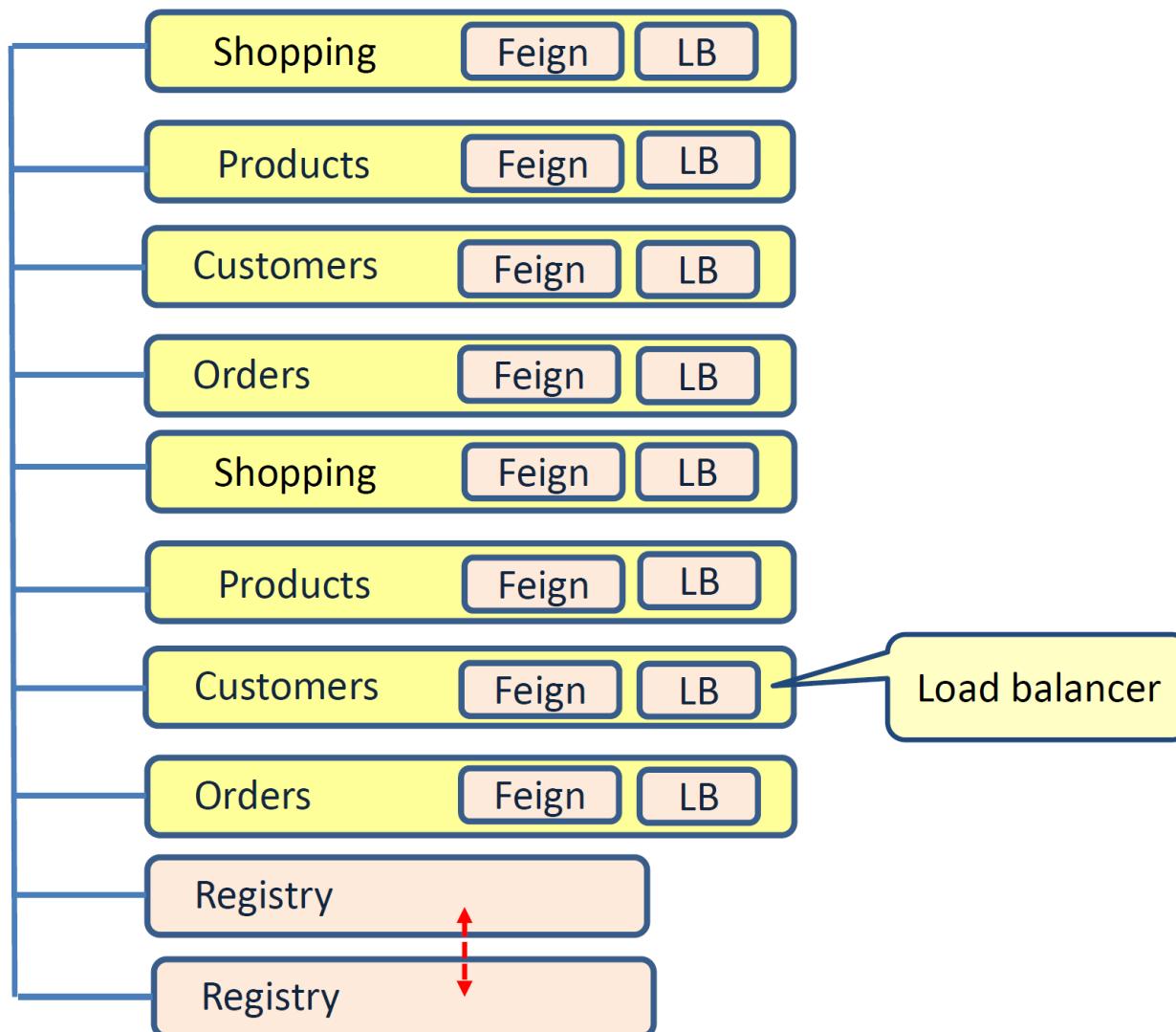
```
localhost:8090/customer/1
Pretty-print
{"name": "Frank Brown", "accountNumber": "1234", "balance": "2000.00"}
```

```
localhost:8090/customer/1
Pretty-print
{"name": "Frank Brown", "accountNumber": "1234", "balance": "1000.00"}
```



Feign does automatically load balance the calls using the registry

Implementing microservices

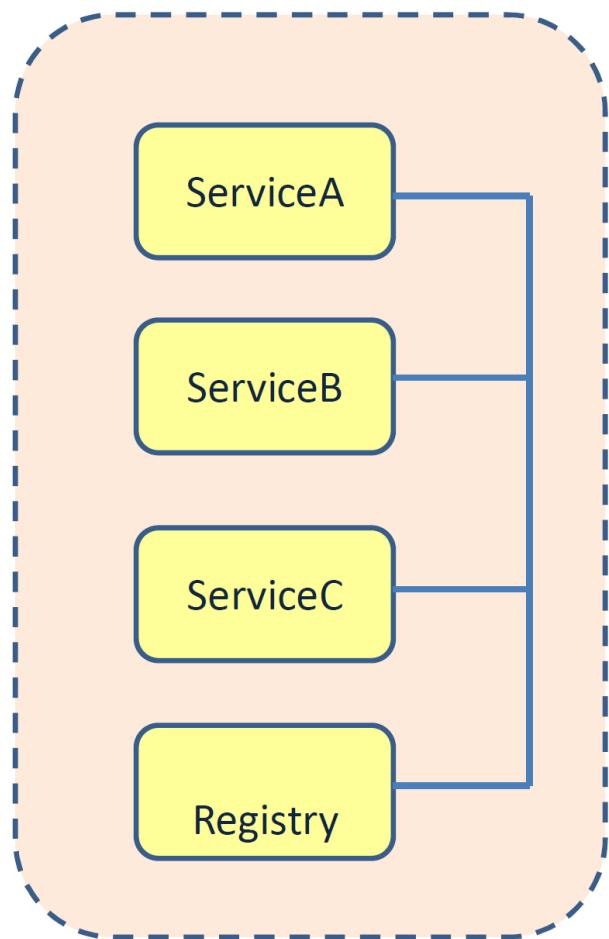


Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry
Performance	
Resilience	Registry and replicas Load balancing between multiple service instances
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	

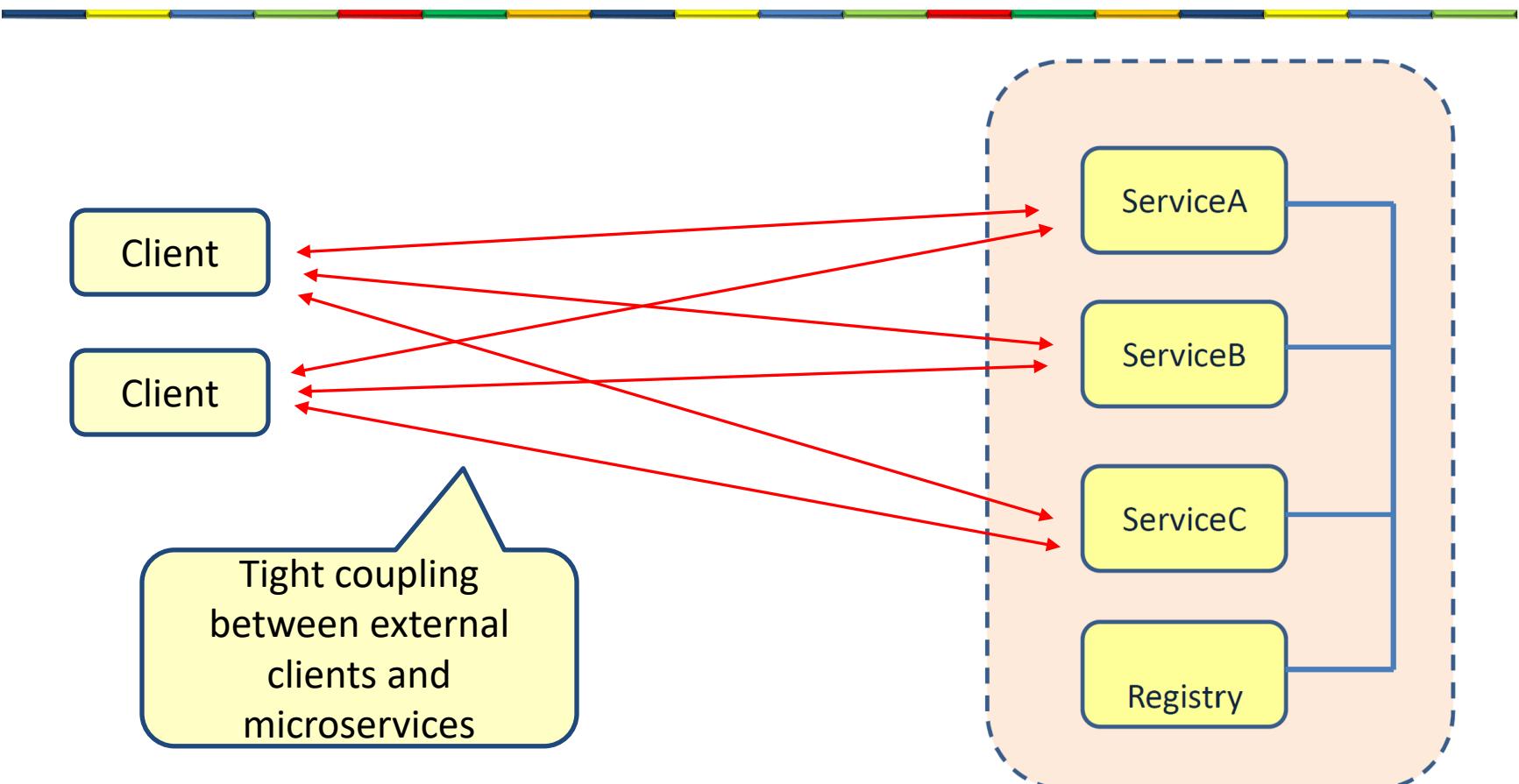
API GATEWAY

Microservice architecture

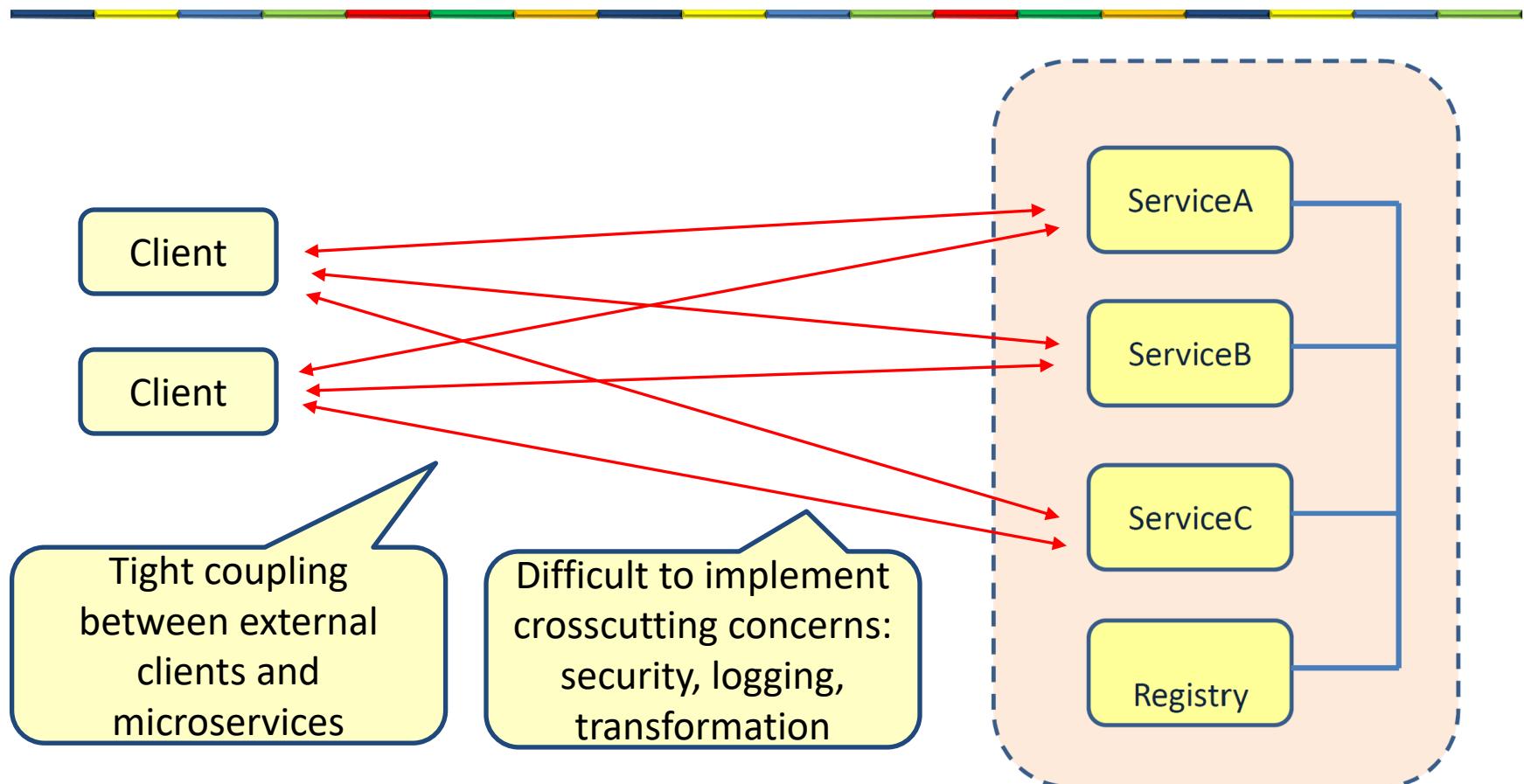


Services talk to each other using
the registry

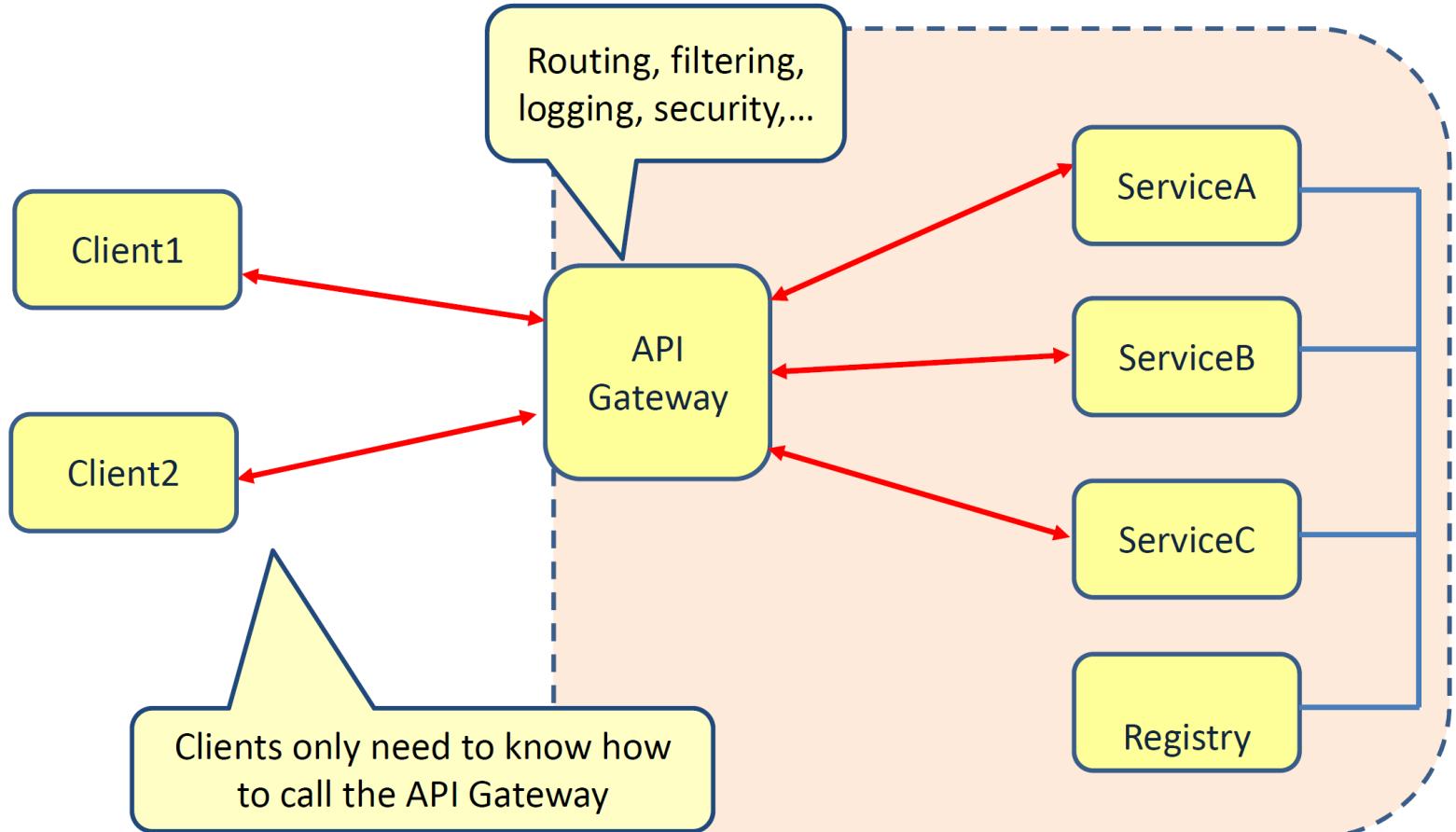
Adding external clients



Adding external clients



API Gateway



StudentService

```
@RestController  
public class StudentController {  
    @RequestMapping("/students/{studentid}")  
    public String getStudent(@PathVariable("studentid") String studentId) {  
        return "Frank Brown";  
    }  
}
```

```
spring:  
    application:  
        name: Studentservice  
    cloud:  
        consul:  
            host: localhost  
            port: 8500  
        discovery:  
            enabled: true  
            prefer-ip-address: true  
            instance-id: ${spring.application.name}:${random.value}  
  
    server:  
        port: 8095
```

GradingService

```
@RestController  
public class GradingController {  
    @RequestMapping("/grades/{studentid}/{courseid}")  
    public String getGrade(@PathVariable("studentid") String studentId,  
                          @PathVariable("courseid") String courseid) {  
        return "A+";  
    }  
}
```

```
spring:  
application:  
  name: GradingService  
cloud:  
consul:  
  host: localhost  
  port: 8500  
discovery:  
  enabled: true  
  prefer-ip-address: true  
  instance-id: ${spring.application.name}:${random.value}  
  
server:  
  port: 8096
```

API Gateway

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

```
<dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter-gateway-server-  
    webflux</artifactId>  
</dependency>
```

API Gateway

```
spring:  
  application:  
    name: api-gateway  
  cloud:  
    consul:  
      host: localhost  
      port: 8500  
    gateway:  
      server:  
        webflux:  
          routes:  
            - id: studentModule  
              uri: lb://Studentservice  
              predicates:  
                - Path=/students/**  
            - id: gradingModule  
              uri: lb://GradingService  
              predicates:  
                - Path=/grades/**  
  server:  
    port: 8080
```

Configure name of the service, so registry is used

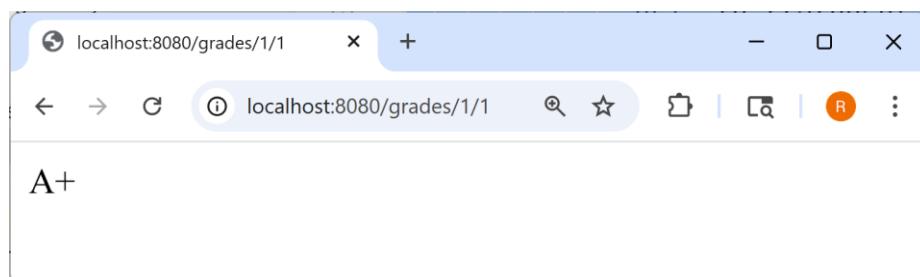
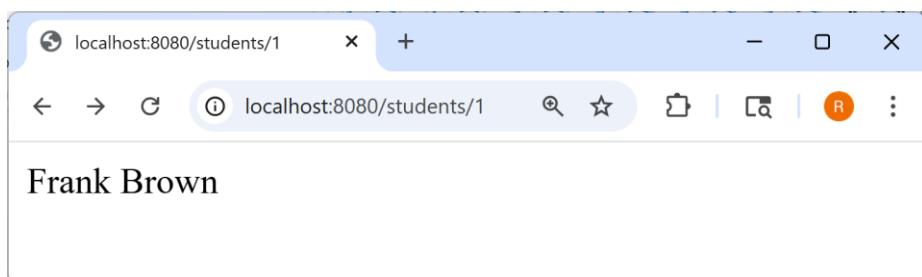
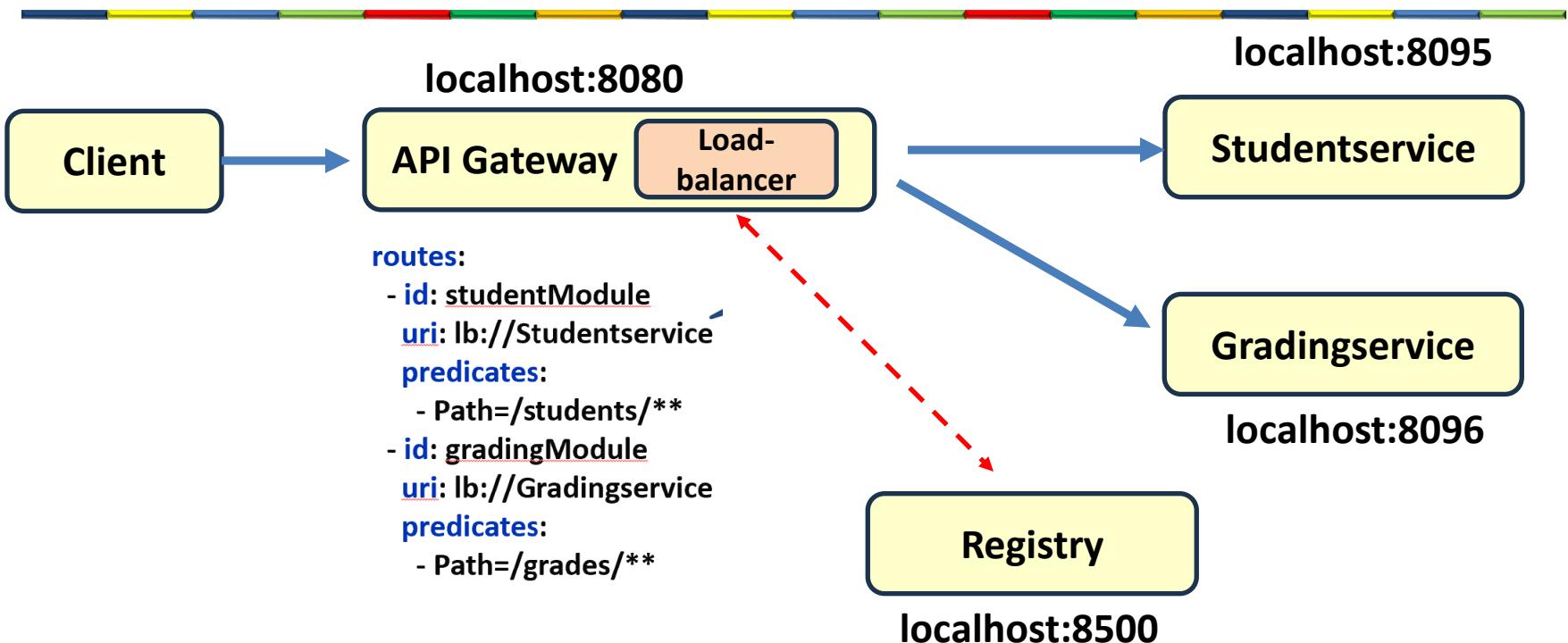
lb is load balancing

API Gateway

The screenshot shows the Consul UI interface for the 'dc1' data center. The left sidebar has a dark theme with white text and icons. It includes links for Overview, Services (which is highlighted in a grey bar), Nodes, Key/Value, Intentions, and Access Controls. The main content area has a light background. At the top, it says 'Services' and '4 total'. Below this, there is a list of four services, each with a green checkmark icon:

- Gradingservice**: 1 instance
- Studentservice**: 1 instance
- api-gateway**: 1 instance
- consul**: 3 instances

API Gateway



Java based config

```
@Configuration  
public class BeanConfig {  
    @Bean  
    public RouteLocator gatewayRoutes(RouteLocatorBuilder builder) {  
        return builder.routes()  
            .route(r -> r.path("/students/**")  
                  .uri("lb://StudentService"))  
            .route(r -> r.path("/grades/**")  
                  .uri("lb://GradingService"))  
            .build();  
    }  
}
```

Build-in predicates

Name	Description	Example
After Route	It takes a date-time parameter and matches requests that happen after it	After=2017-11-20T...
Before Route	It takes a date-time parameter and matches requests that happen before it	Before=2017-11-20T...
Between Route	It takes two date-time parameters and matches requests that happen between those dates	Between=2017-11-20T..., 2017-11-21T...
Cookie Route	It takes a cookie name and regular expression parameters, finds the cookie in the HTTP request's header, and matches its value with the provided expression	Cookie=SessionID, abc.
Header Route	It takes the header name and regular expression parameters, finds a specific header in the HTTP request's header, and matches its value with the provided expression	Header=X-Request-Id, \d+
Host Route	It takes a hostname ANT style pattern with the . separator as a parameter and matches it with the Host header	Host=*.example.org
Method Route	It takes an HTTP method to match as a parameter	Method=GET
Path Route	It takes a pattern of request context path as a parameter	Path=/account/{id}
Query Route	It takes two parameters—a required param and an optional regexp and matches them with query parameters	Query=accountId, 1.
RemoteAddr Route	It takes a list of IP addresses in CIDR notation, like 192.168.0.1/16, and matches it with the remote address of a request	RemoteAddr=192.168.0.1/16

Custom Global filter

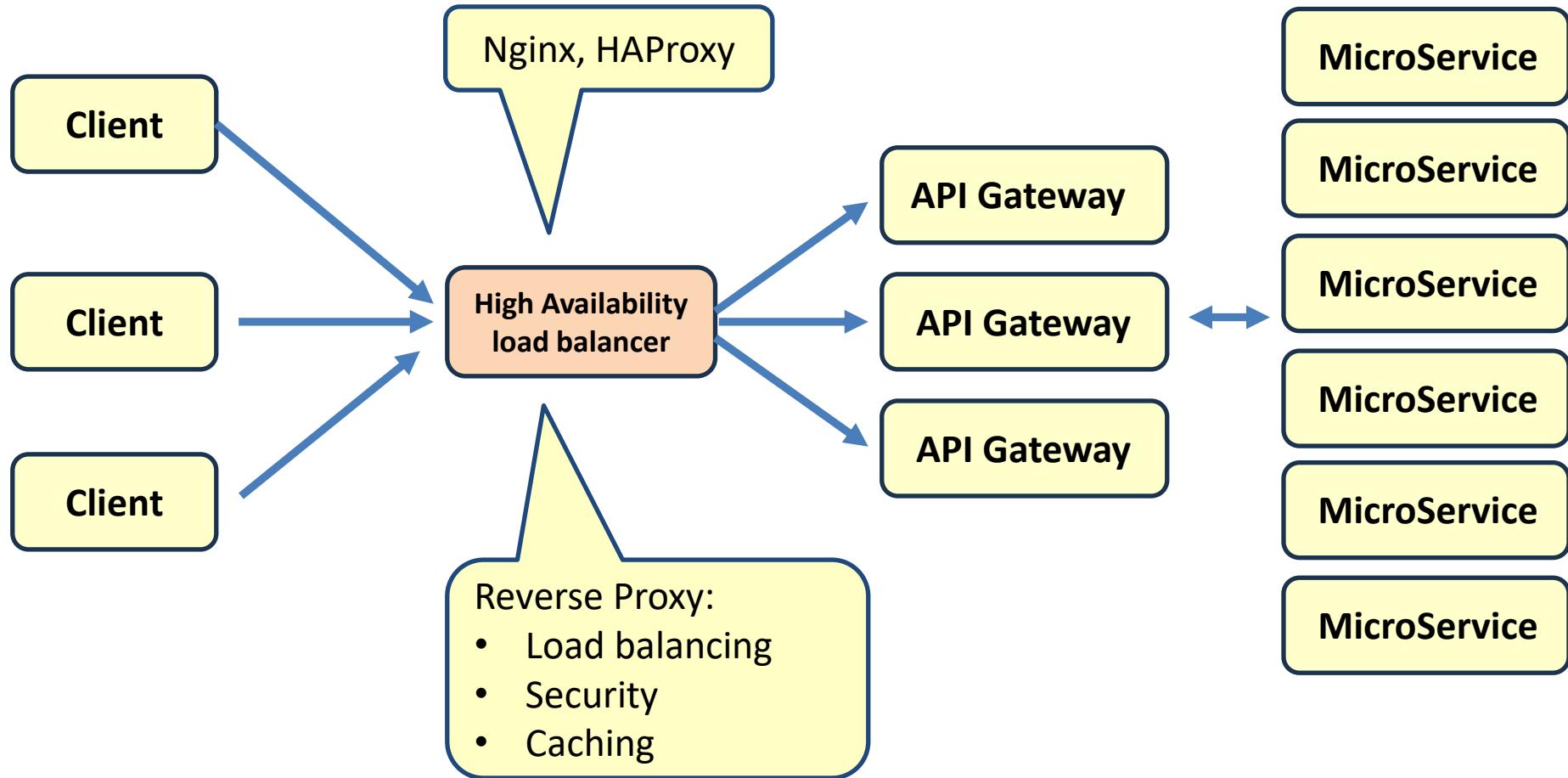
- Global filter is applied to all routes

```
@Component
public class PreLastPostGlobalFilter
    implements GlobalFilter {

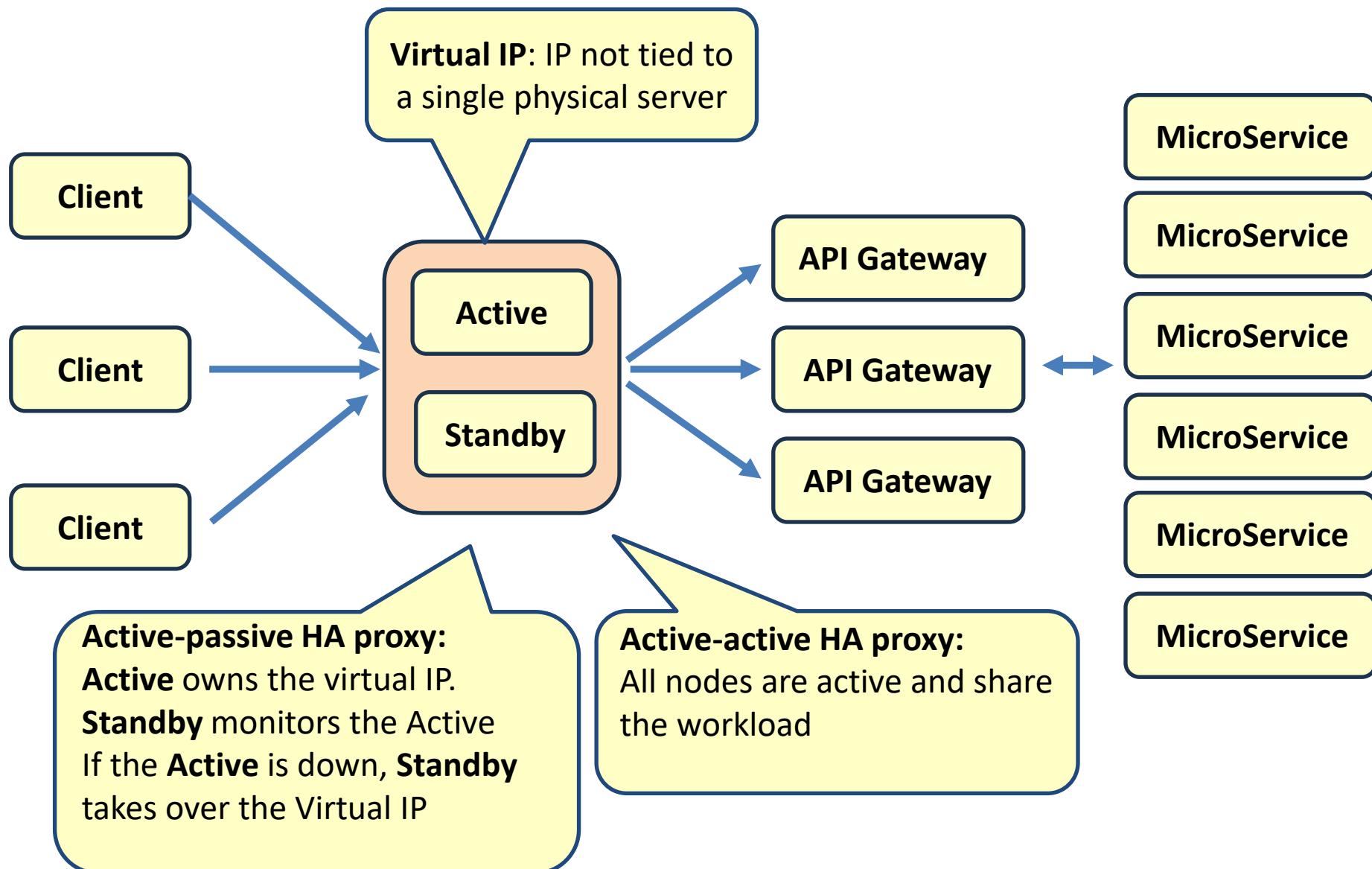
    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
        GatewayFilterChain chain) {
        System.out.println("Pre Global Filter "+exchange.getRequest().getURI());
        return chain.filter(exchange)
            .then(Mono.fromRunnable(() -> {
                System.out.println("Post Global Filter "+exchange.getResponse().getStatusCode();
            }));
    }
}
```

Pre Global Filter <http://localhost:8080/students/1>
Post Global Filter 200 OK

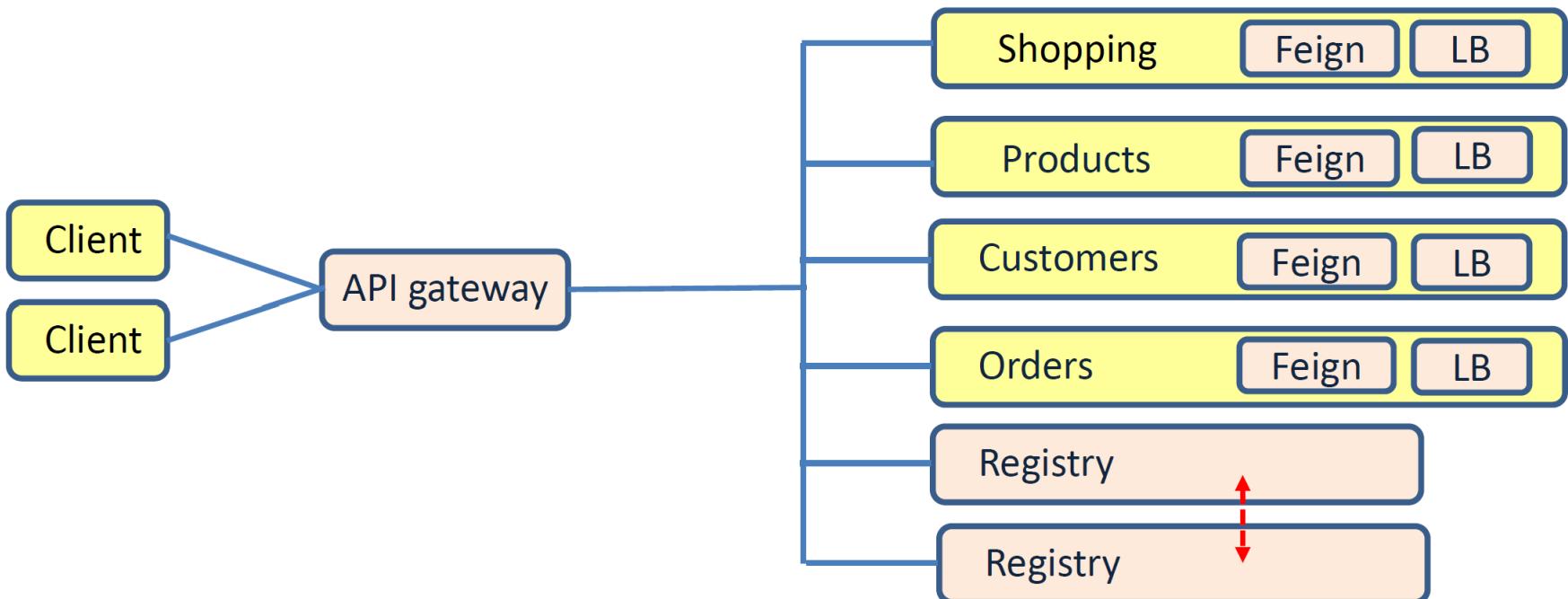
Avoid single point of failure



Avoid single point of failure



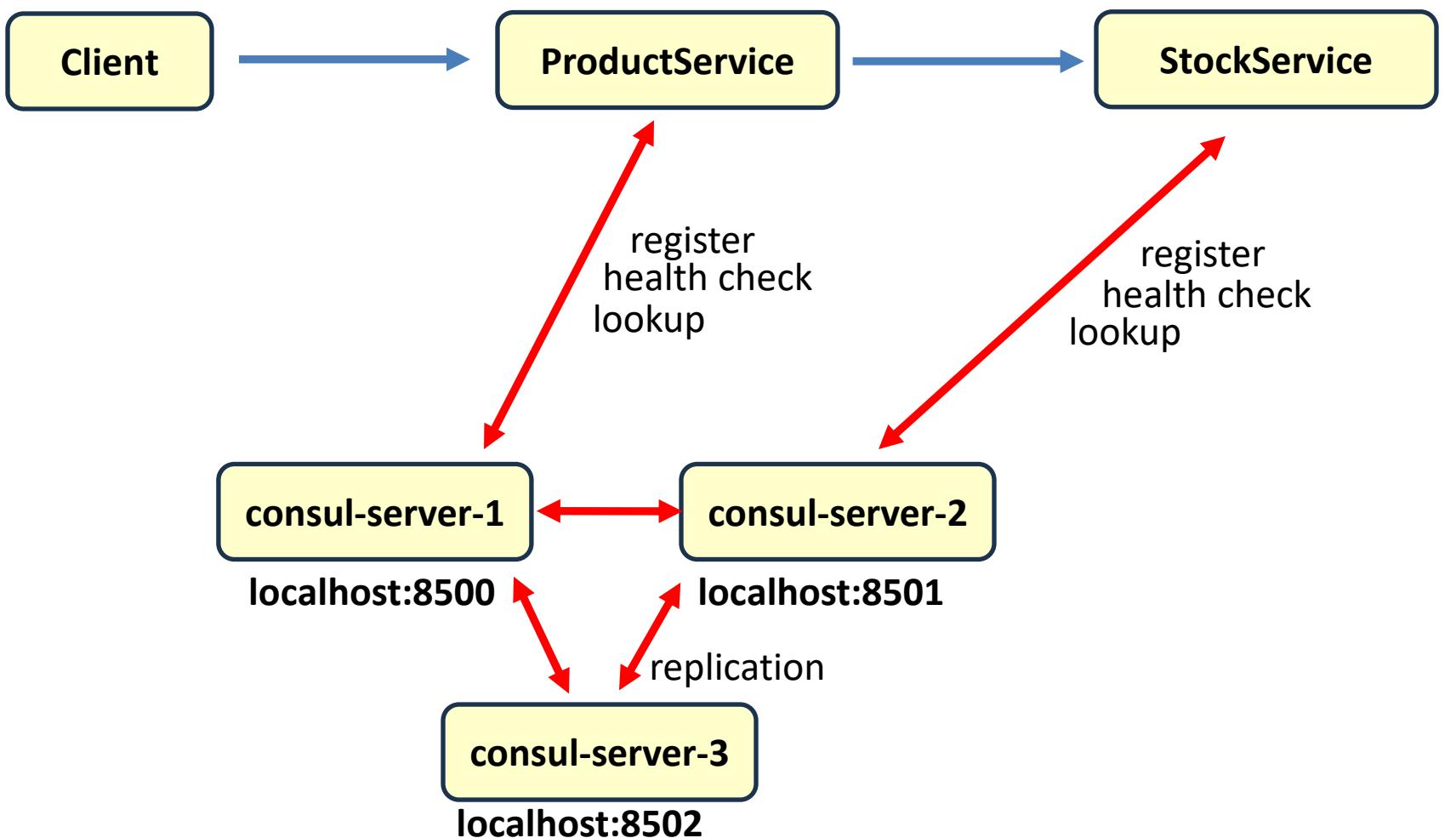
Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API Gateway
Performance	
Resilience	Registry and replicas Load balancing between multiple service instances
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	

Consul cluster and replication



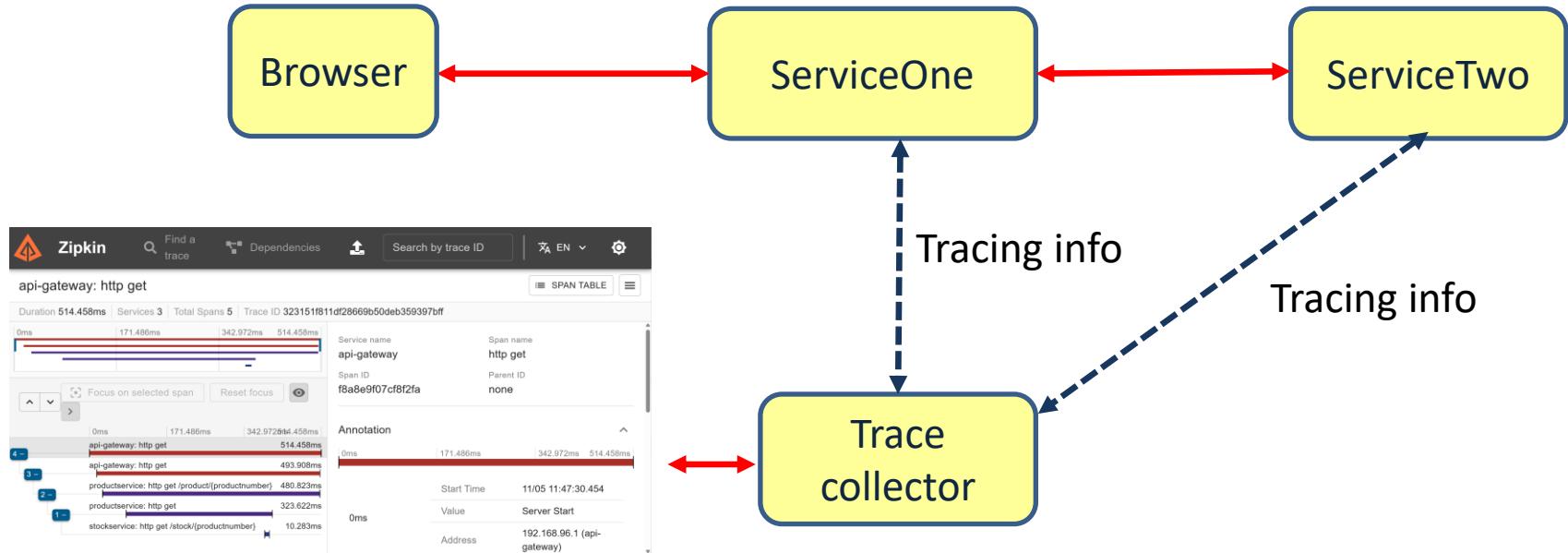
Lesson 8

MICROSERVICES

DISTRIBUTED TRACING: ZIPKIN

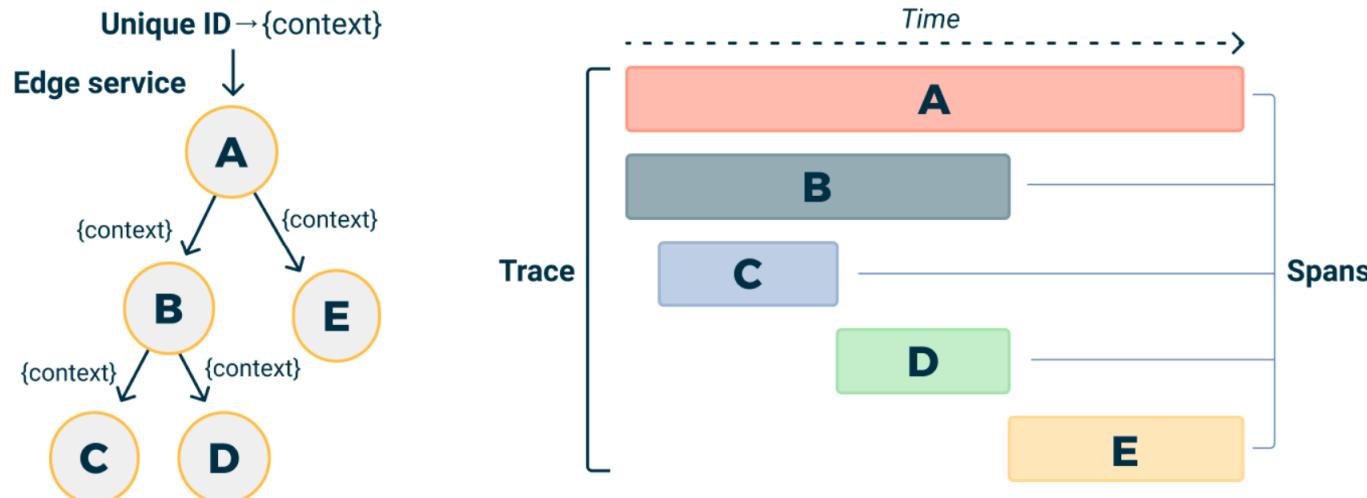
Distributed Tracing

- One central place where one can see the end-to-end tracing of all communication between services



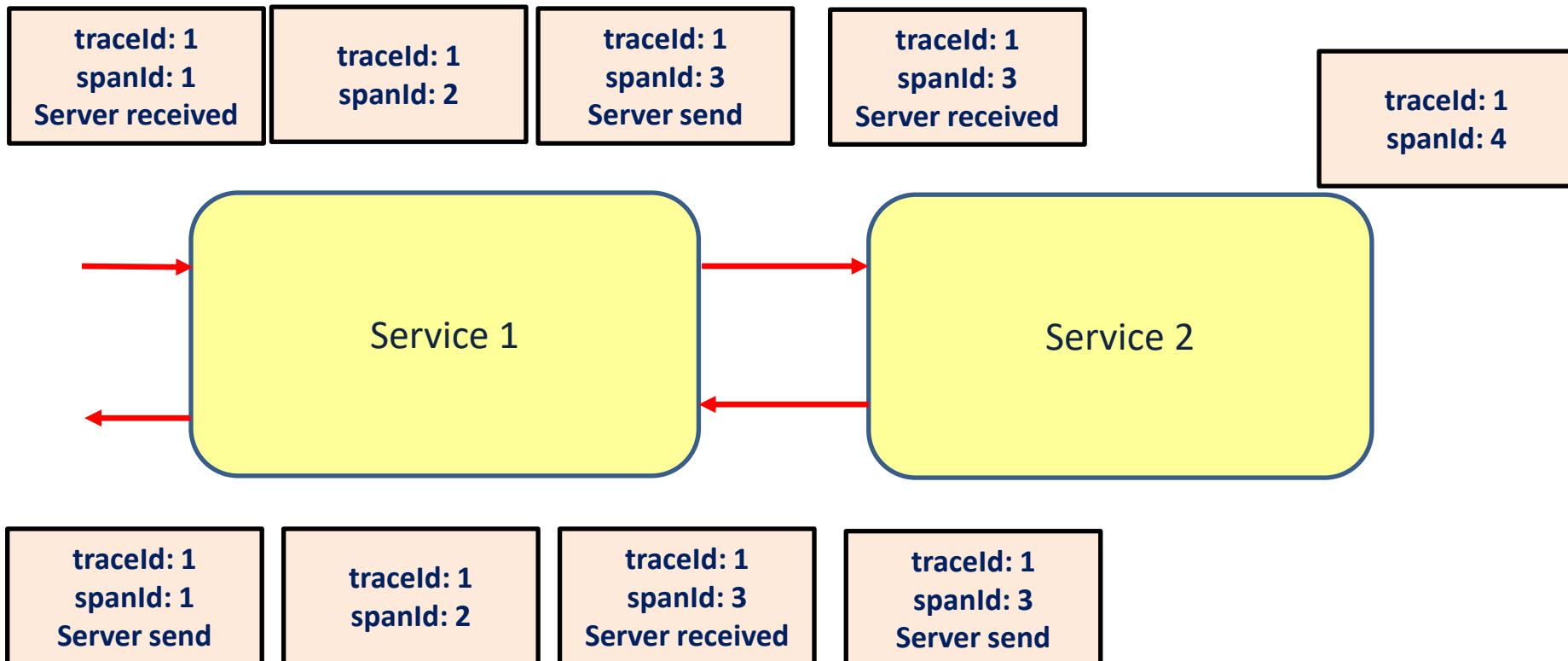
Micrometer

- Adds unique id's to a request so we can trace the request
 - Span id: id for an individual operation
 - Trace id: id for a set of spans



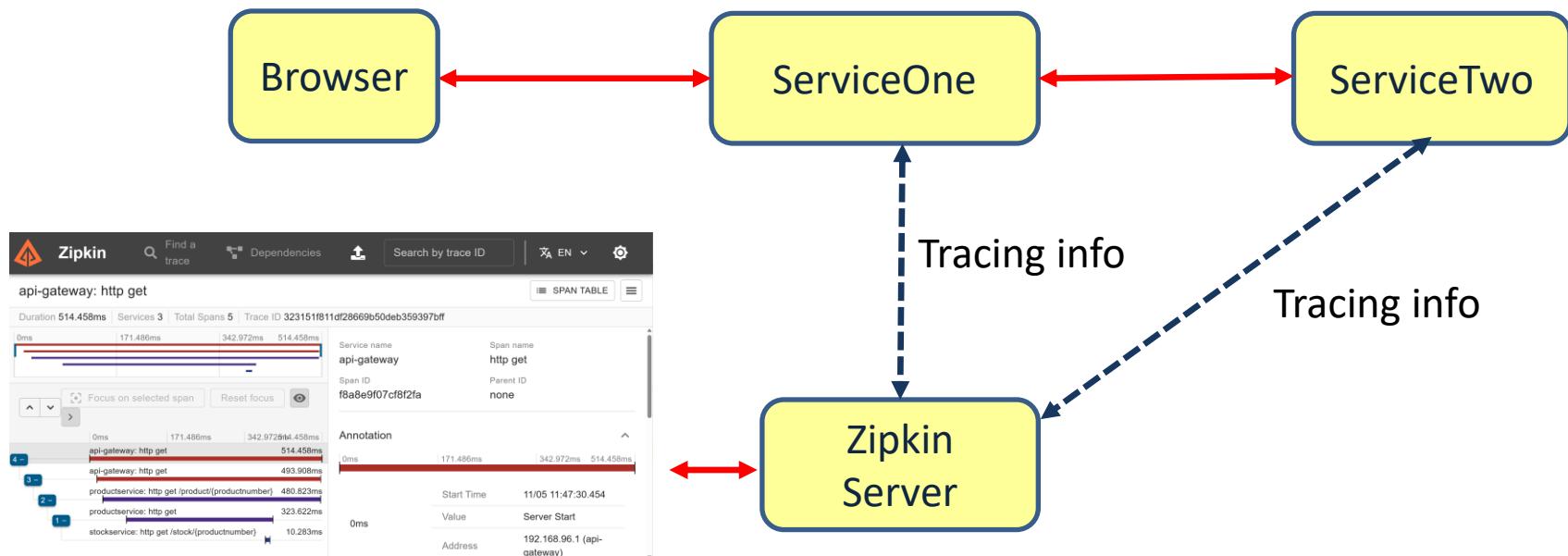
Span and Trace id's

- Span: an individual operation
- Trace: a set of spans



Zipkin

- Centralized tracing server
 - Collects tracing information
- Zipkin console shows the data



ServiceOne

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

ServiceOne

```
@RestController
public class ServiceOneController {

    @Autowired
    ServiceTwoClient serviceTwoClient;

    @RequestMapping("/text")
    public String getText() {
        String service2Text = serviceTwoClient.getText();
        return "Hello " + service2Text;
    }

    @FeignClient("ServiceTwo")
    interface ServiceTwoClient {
        @RequestMapping("/text")
        public String getText();
    }
}
```

ServiceOne

```
server:  
  port: 9093
```

application.yml

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

```
otlp:  
  tracing:  
    endpoint: http://localhost:9411/api/v2/spans # Zipkin endpoint
```

```
management:  
  tracing:  
    sampling:  
      probability: 1.0
```

probability =1.0 means
send tracing info for
every call

Typically set probability to 0.1 which
sends trace info only for 10% of the calls

ServiceOne

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <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>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-otel</artifactId>
  </dependency>
  <dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-micrometer</artifactId>
    <version>13.6</version>
  </dependency>
  <dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-zipkin</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
  </dependency>
</dependencies>
```

pom.xml

ServiceTwo

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

```
@RestController  
public class ServiceTwoController {  
    @RequestMapping("/text")  
    public String getText() {  
        return "World";  
    }  
}
```

ServiceTwo



```
server:  
  port: 9091  
  
spring:  
  application:  
    name: ServiceTwo  
  cloud:  
    consul:  
      host: localhost  
      port: 8500  
      discovery:  
        enabled: true  
        prefer-ip-address: true  
      instance-id: ${spring.application.name}:${random.value}  
  
otlp:  
  tracing:  
    endpoint: http://localhost:9411/api/v2/spans # Zipkin endpoint  
  
management:  
  tracing:  
    sampling:  
      probability: 1.0
```

application.yml

Zipkin console

The screenshot shows the Zipkin console interface with the following details:

Header: Zipkin logo, Find a trace search bar, Dependencies icon, Search by trace ID input field, XA EN dropdown, and settings gear icon.

Trace Summary: serviceone: http get /text, Duration 625.216ms, Services 2, Total Spans 3, Trace ID 34d9eb2cccd37ce3ee8ffedf1860761cb.

Timeline: A horizontal timeline showing three spans. The first span (highlighted in green) is labeled "serviceone: http get /text" and has a duration of 625.216ms. The second span is labeled "serviceone: http get" and has a duration of 531.506ms. The third span is labeled "servicetwo: http get /text" and has a duration of 23.517ms. The timeline is marked at 0ms, 208.405ms, 416.811ms, and 625.216ms.

Focus: Focus on selected span button, Reset focus button, and a zoom-in icon.

Annotations: An annotation for the first span is shown with the following details:

Annotation	Start Time	Value
0ms	11/05 19:52:36.493	Server Start
Address	192.168.96.1 (serviceone)	

Zipkin console

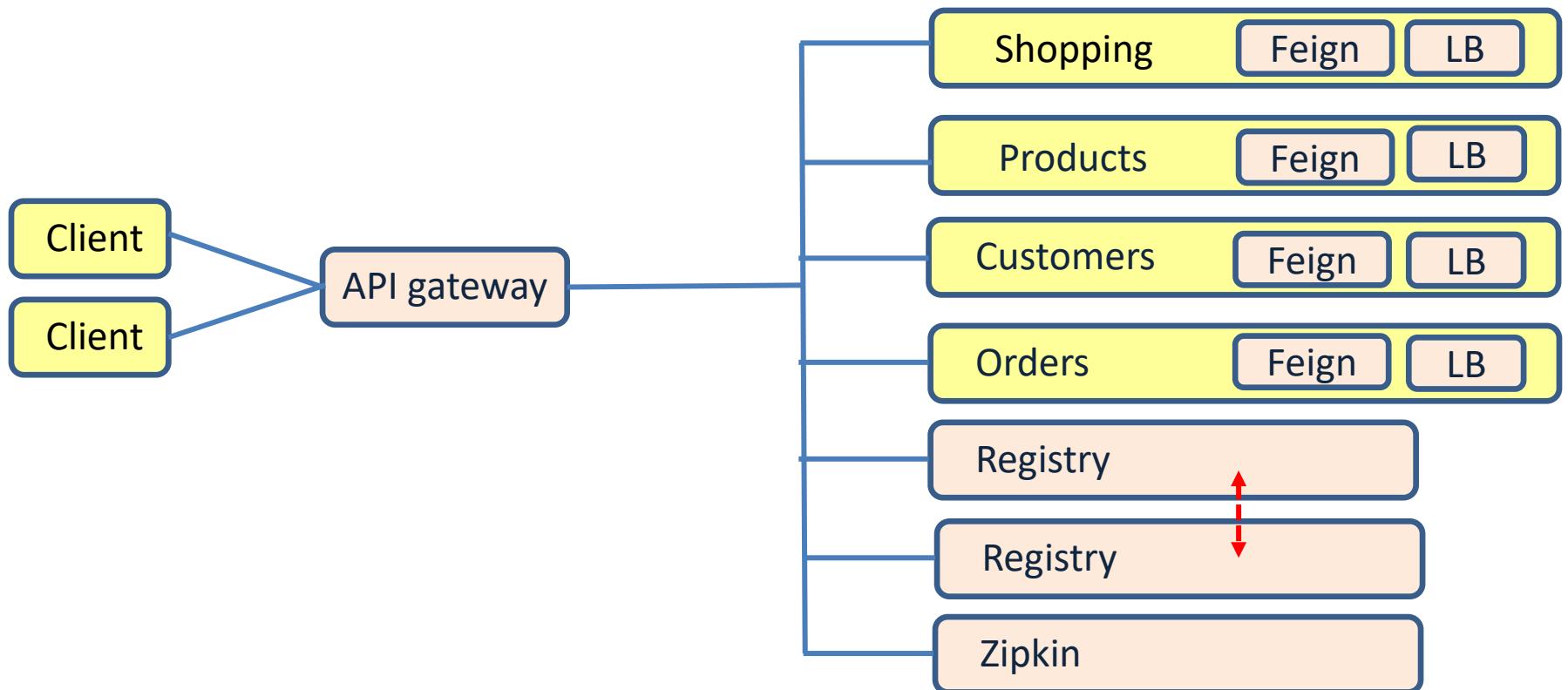
The screenshot shows the Zipkin console interface. At the top, there is a navigation bar with the Zipkin logo, a search bar labeled "Find a trace", a "Dependencies" button, a "Search by trace ID" input field, language selection ("EN"), and a settings gear icon.

Below the navigation bar, there are date range inputs for "Start Time" (11/04/2025 19:53:36) and "End Time" (11/05/2025 19:53:36), followed by a "RUN QUERY" button.

A "Filter" dropdown menu is located on the left side.

The main area displays a horizontal timeline with colored segments representing different spans. A specific trace is highlighted with blue segments. Two service nodes, "serviceone" and "servicetwo", are shown as blue circles connected by a line, indicating their interaction. The entire screenshot is framed by a thin gray border.

Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances
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	Zipkin

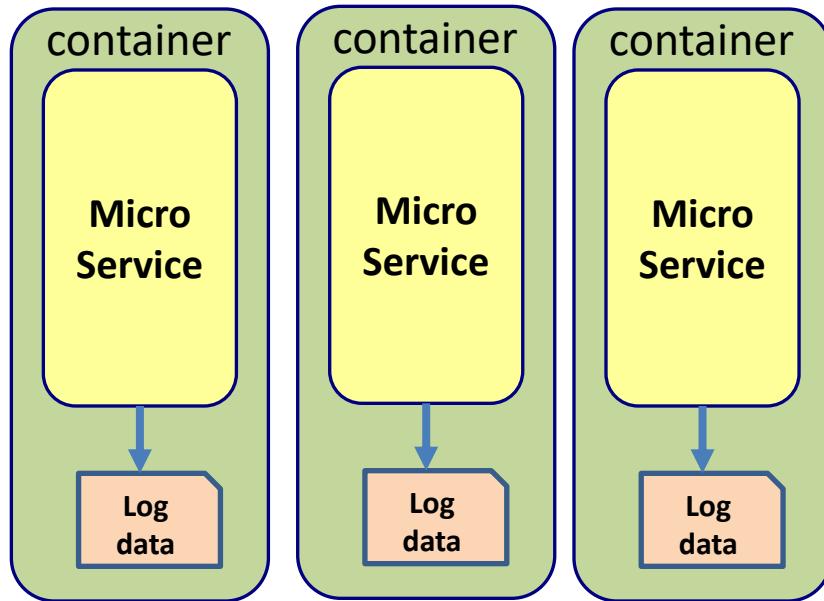
Main point

- We need zipkin in order to monitor and debug service-to-service communication
- The Unified Field is the field of perfection

DISTRIBUTED LOGGING

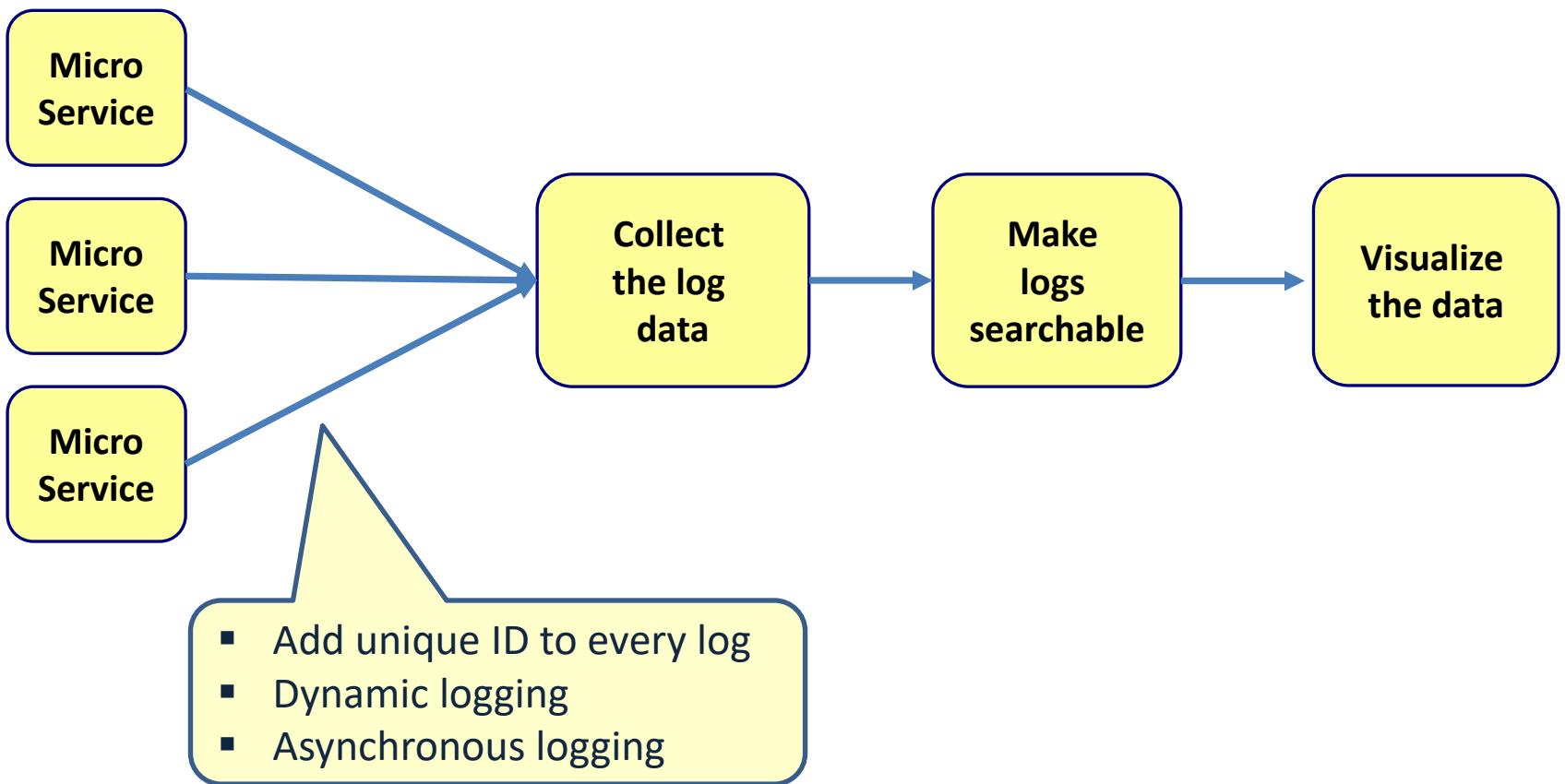
ELK STACK

The need for centralized logging

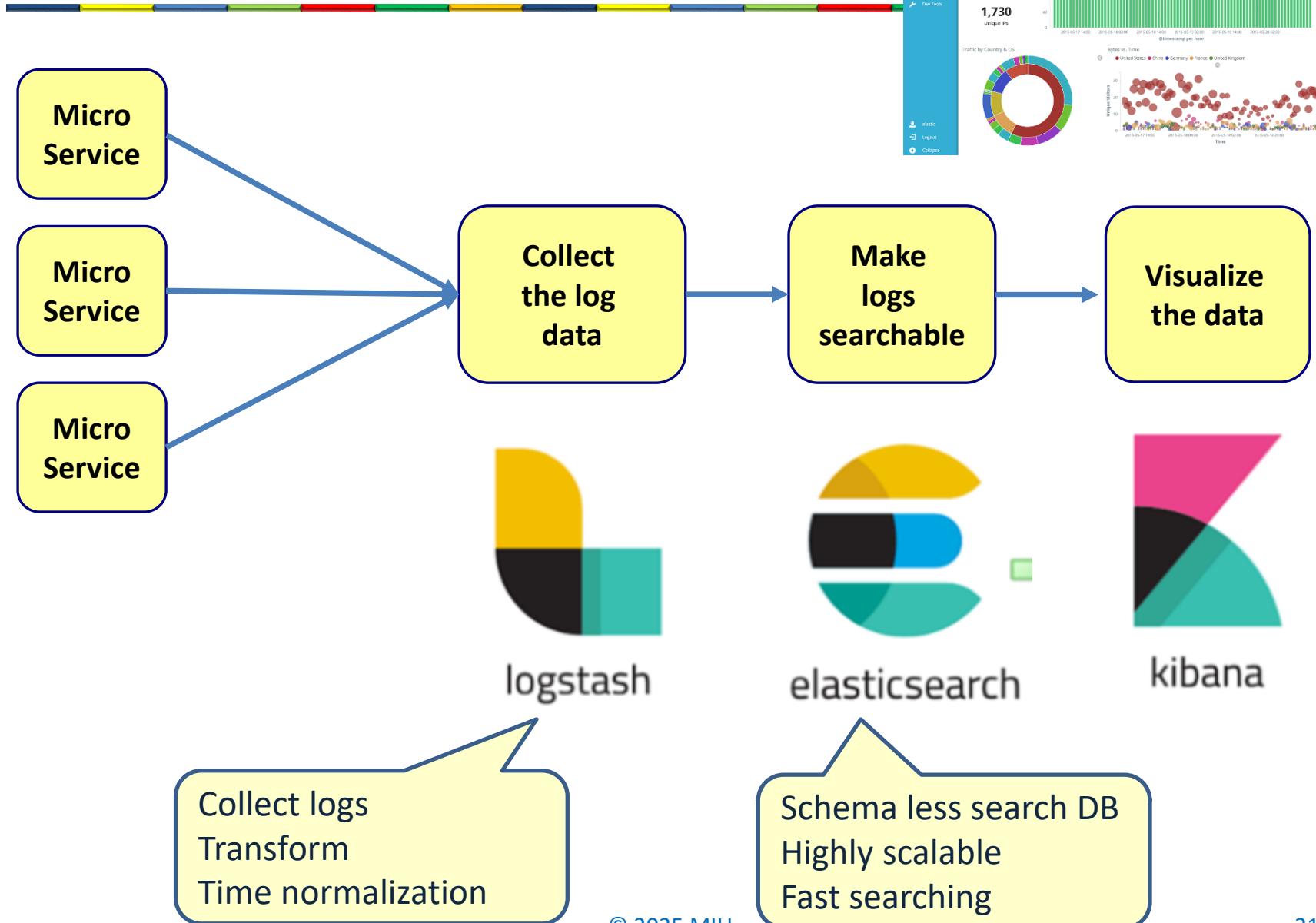


- Local logging does not work
 - Containers come and go
 - Containers have no fixed address

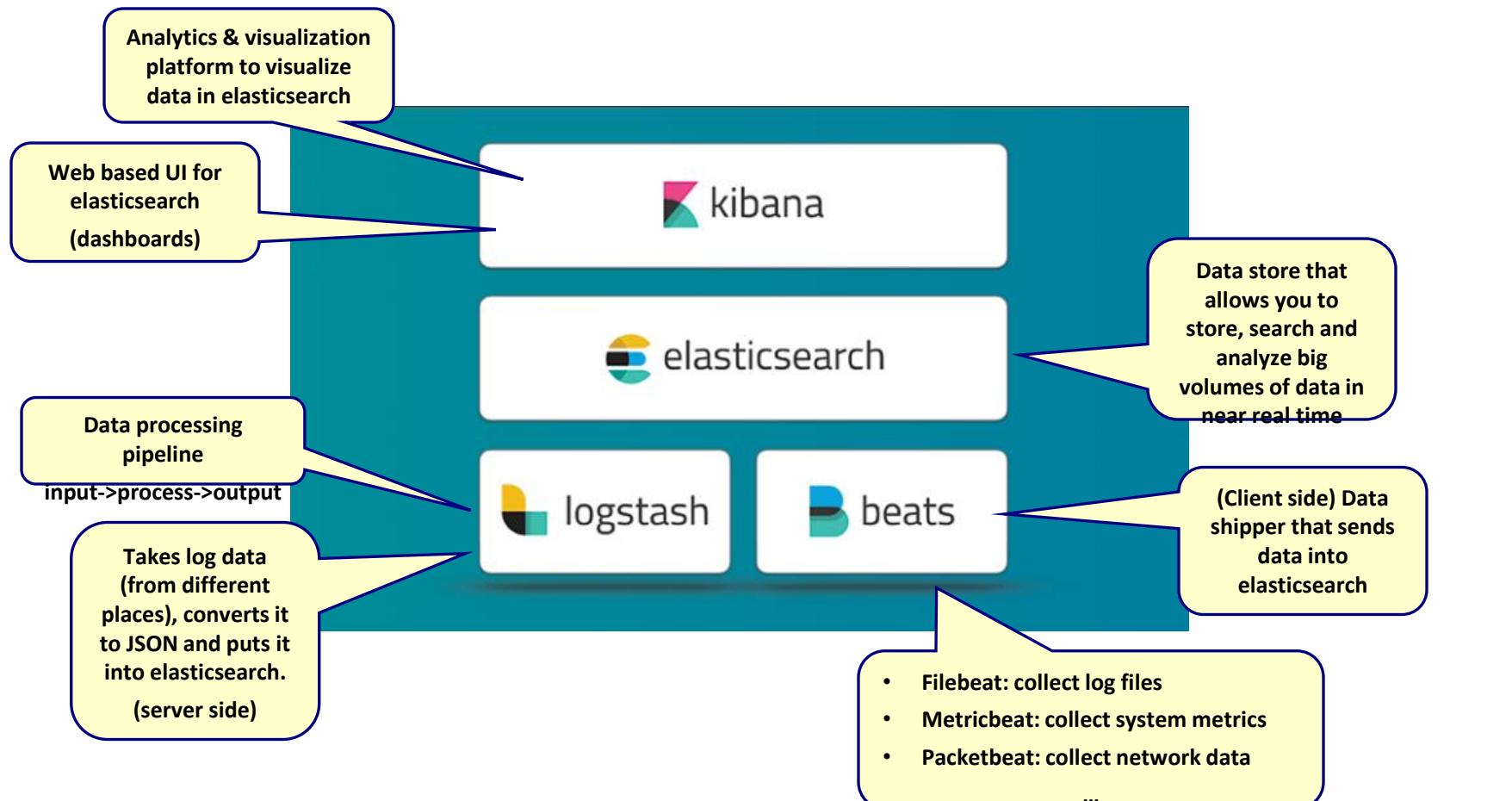
Microservice logging architecture



ELK stack

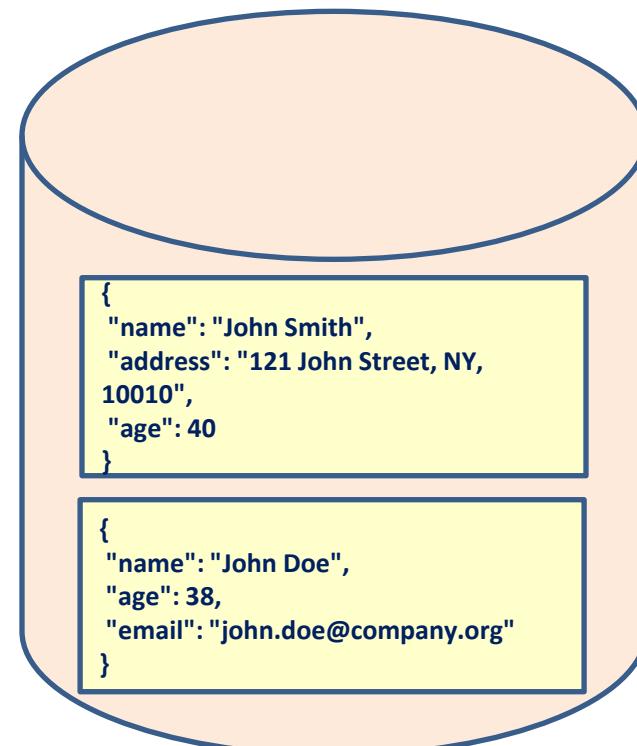


Elastic stack components



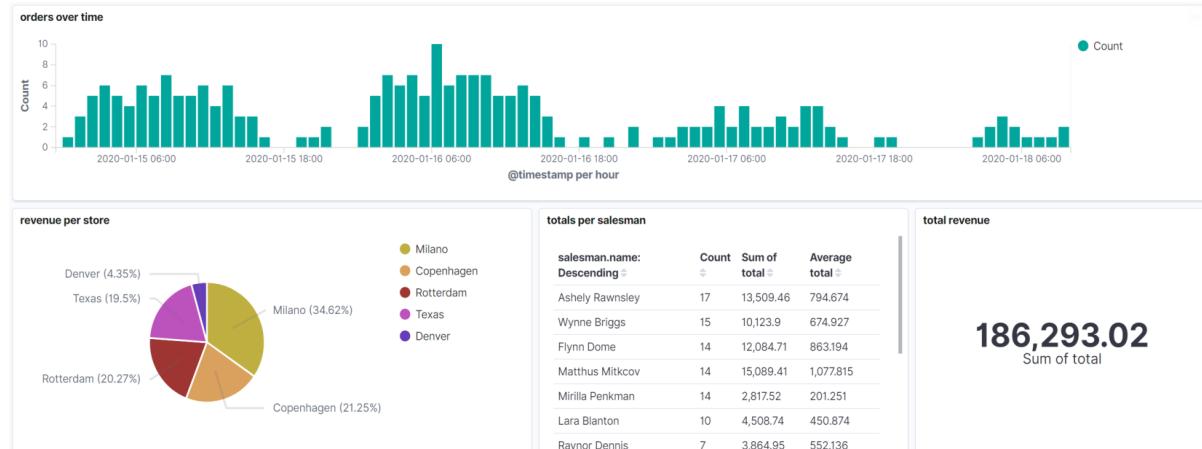
What is Elasticsearch?

- Database
 - Data is stored as JSON documents
- Full text search engine



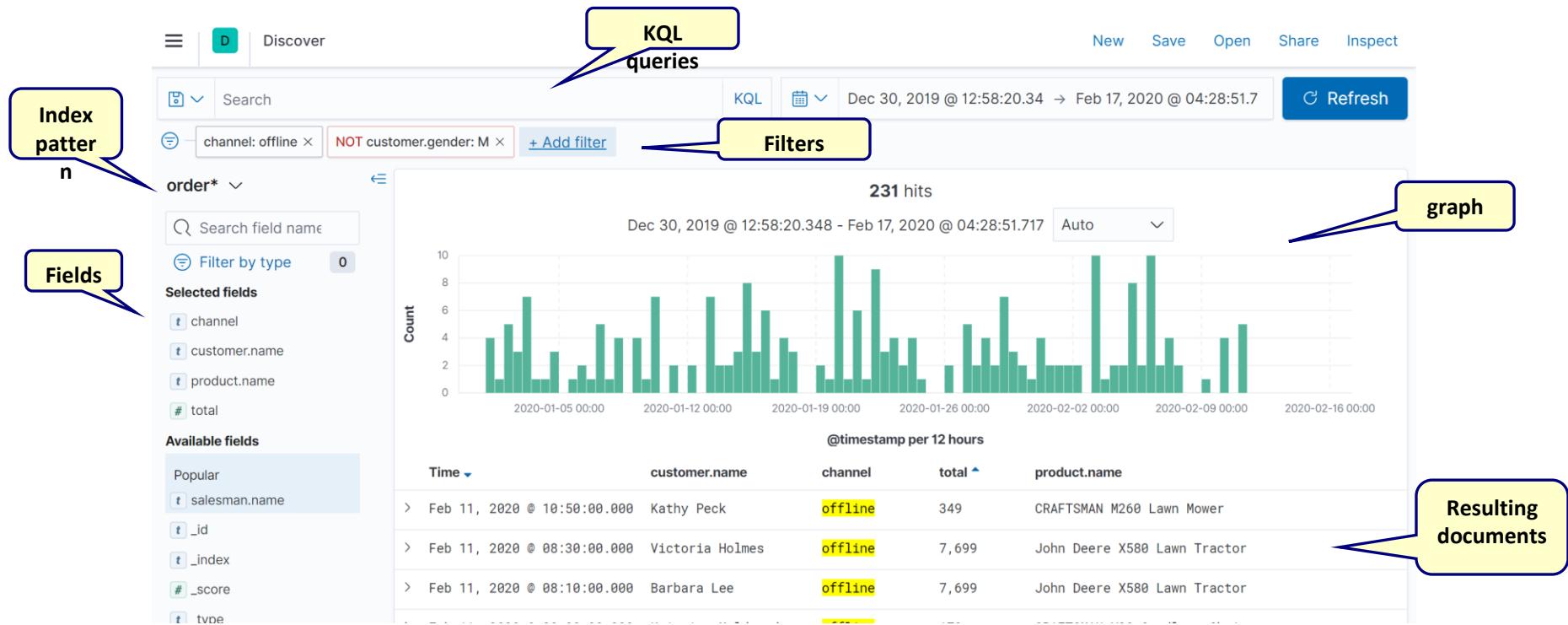
Kibana

- Web UI on top of elasticsearch
- Has its own Kibana query language (KQL)
- Objects (Queries, visualizations, dashboards, etc.) are saved in elasticsearch

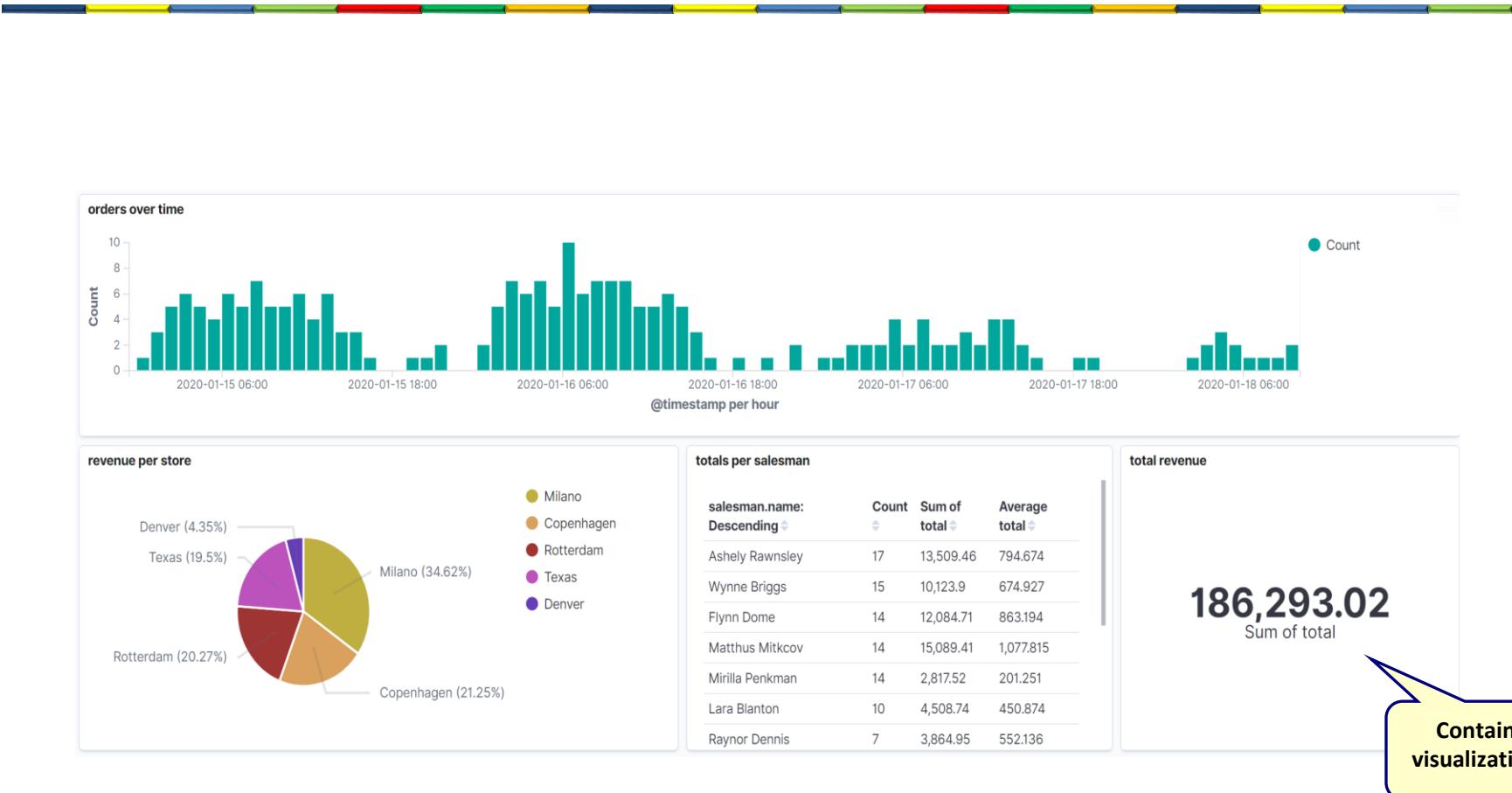


Discover app

- Good for exploring and analyzing data

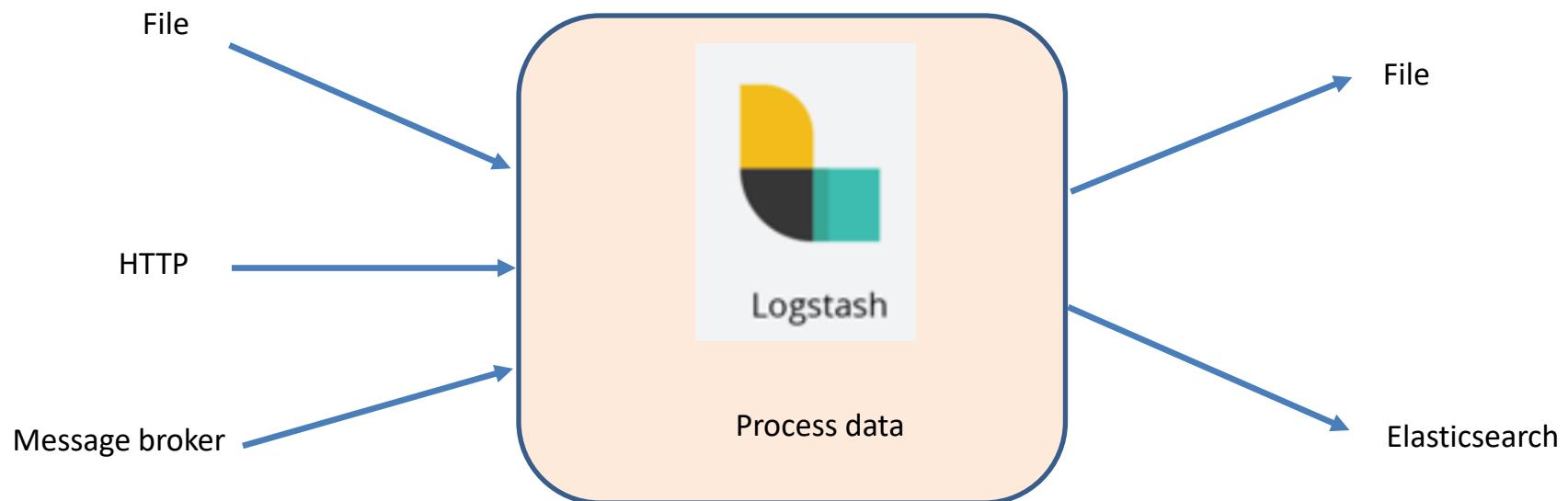


Dashboard

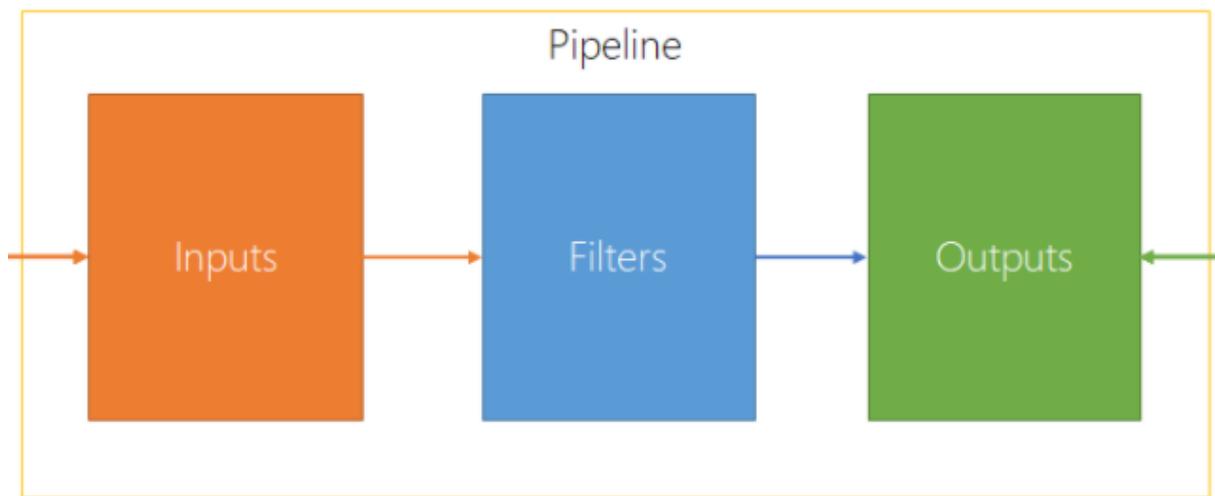


Logstash

- Event processing engine



How does Logstash work?



Logstash configuration

```
input {  
    ...  
}
```

- **file**: This streams log events from a file
- **redis**: This streams events from a redis instance
- **stdin**: This streams events from standard input
- **syslog**: This streams syslog messages over the network
- **ganglia**: This streams ganglia packets over the network via udp
- **lumberjack**: This receives events using the lumberjack protocol
- **eventlog**: This receives events from Windows event log
- **s3**: This streams events from a file from an s3 bucket
- **elasticsearch**: This reads from the Elasticsearch cluster based on results of a search query

```
filter {  
    ...  
}
```

- **date**: This is used to parse date fields from incoming events, and use that as Logstash timestamp fields, which can be later used for analytics
- **drop**: This drops everything from incoming events that matches the filter condition
- **grok**: This is the most powerful filter to parse unstructured data from logs or events to a structured format
- **multiline**: This helps parse multiple lines from a single source as one Logstash event
- **dns**: This filter will resolve an IP address from any fields specified
- **mutate**: This helps rename, remove, modify, and replace fields in events
- **geoip**: This adds geographic information based on IP addresses that are retrieved from [Maxmind](#) database

```
output {  
    ...  
}
```

- **file**: This writes events to a file on disk
- **e-mail**: This sends an e-mail based on some conditions whenever it receives an output
- **elasticsearch**: This stores output to the Elasticsearch cluster, the most common and recommended output for Logstash
- **stdout**: This writes events to standard output
- **redis**: This writes events to redis queue and is used as a broker for many ELK implementations
- **mongodb**: This writes output to mongodb
- **kafka**: This writes events to Kafka topic

Logstash configuration

input.txt

Hello world

pipeline.conf

```
input {  
  file {  
    path => "C:/elasticsearchtraining/temp/input.txt"  
    start_position => "beginning"  
  }  
}  
  
output {  
  stdout {  
    codec => rubydebug  
  }  
  file {  
    path => "C:/elasticsearchtraining/temp/output.txt"  
  }  
}
```

output.txt

```
{  
  "host": "DESKTOP-BVHRK6K",  
  "@version": "1",  
  "path": "C:/elasticsearchtraining/temp/input.txt",  
  "message": "Hello world\r",  
  "@timestamp": "2021-01-16T13:52:32.726Z"
```

Write the output to the console

Anytime this file changes, read from this file

Write the output to the specified file

Logstash configuration

input.txt

Hi there

pipeline.conf

```
input {
  file {
    path => "C:/elasticsearchtraining/temp/input.txt"
      start_position => "beginning"
  }
}

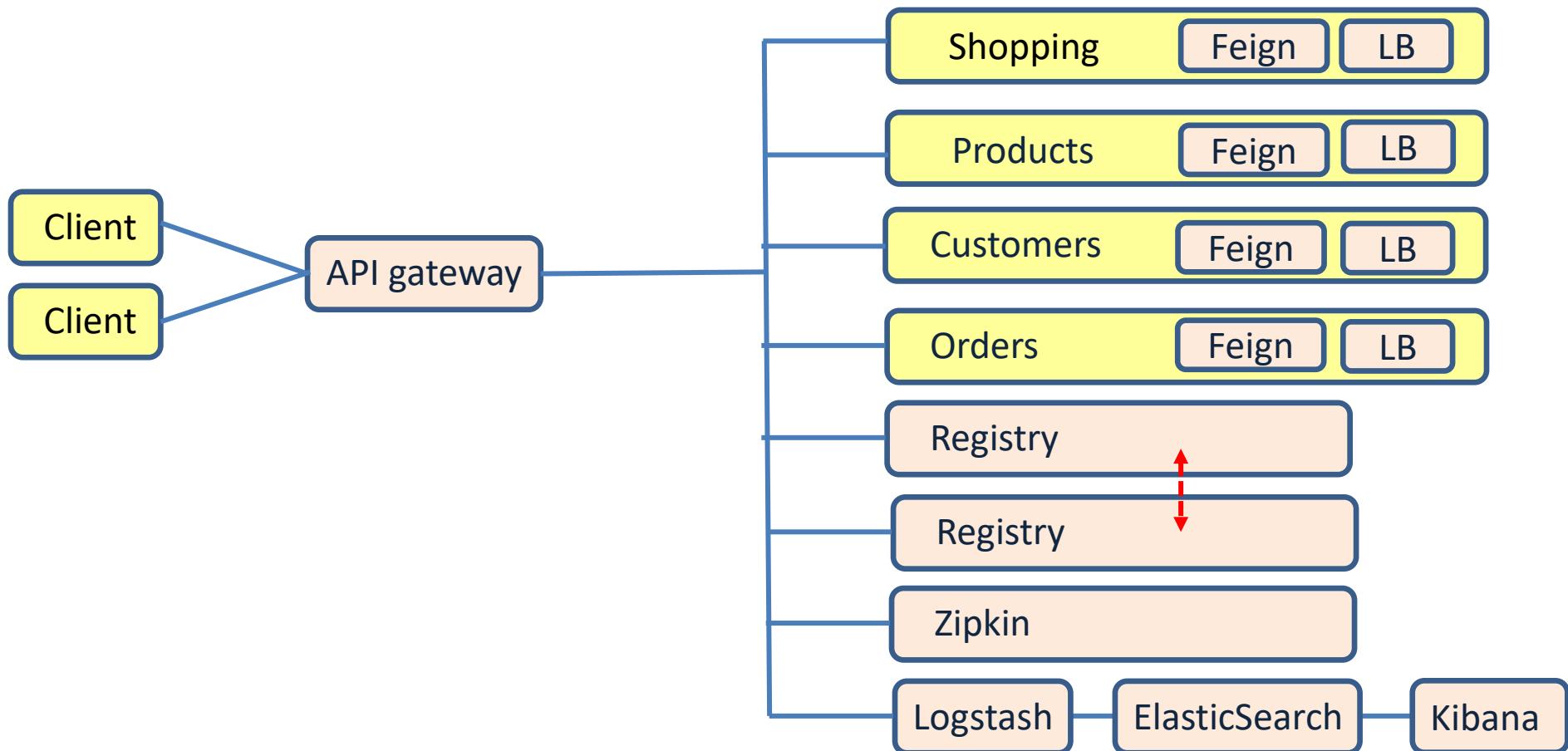
filter {
  mutate {
    uppercase => ["message"]
  }
}

output {
  stdout {
    codec => rubydebug
  }
  file {
    path => "C:/elasticsearchtraining/temp/output.txt"
  }
}
```

output.txt

```
{
  "path": "C:/elasticsearchtraining/temp/input.txt",
  "message": "HI THERE\r",
  "host": "DESKTOP-BVHRK6K",
  "@version": "1",
  "@timestamp": "2021-01-16T14:17:10.537Z"
}
```

Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK

RESILIENCE

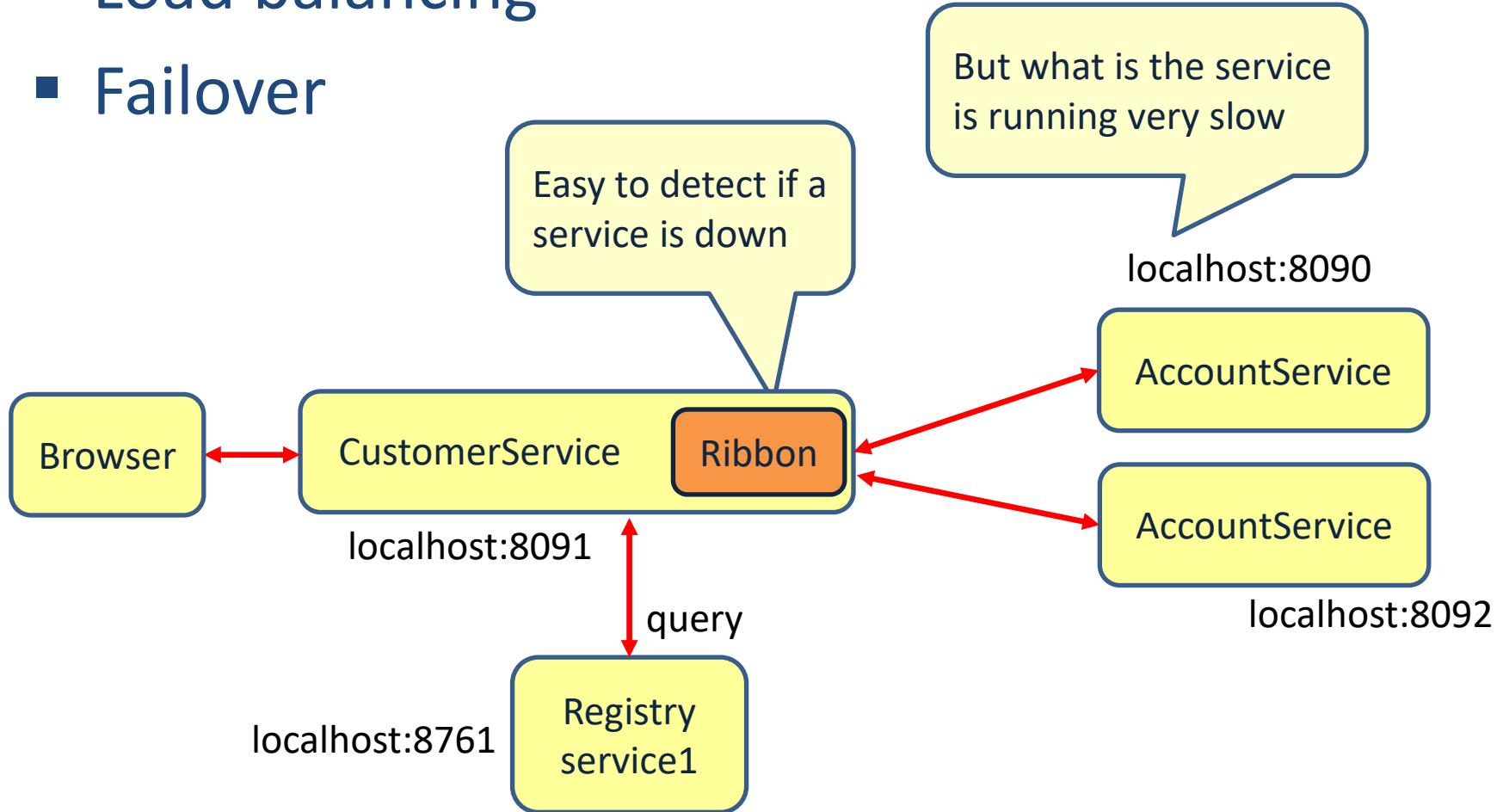
The ability to recover from failures

Fallacies of distributed computing

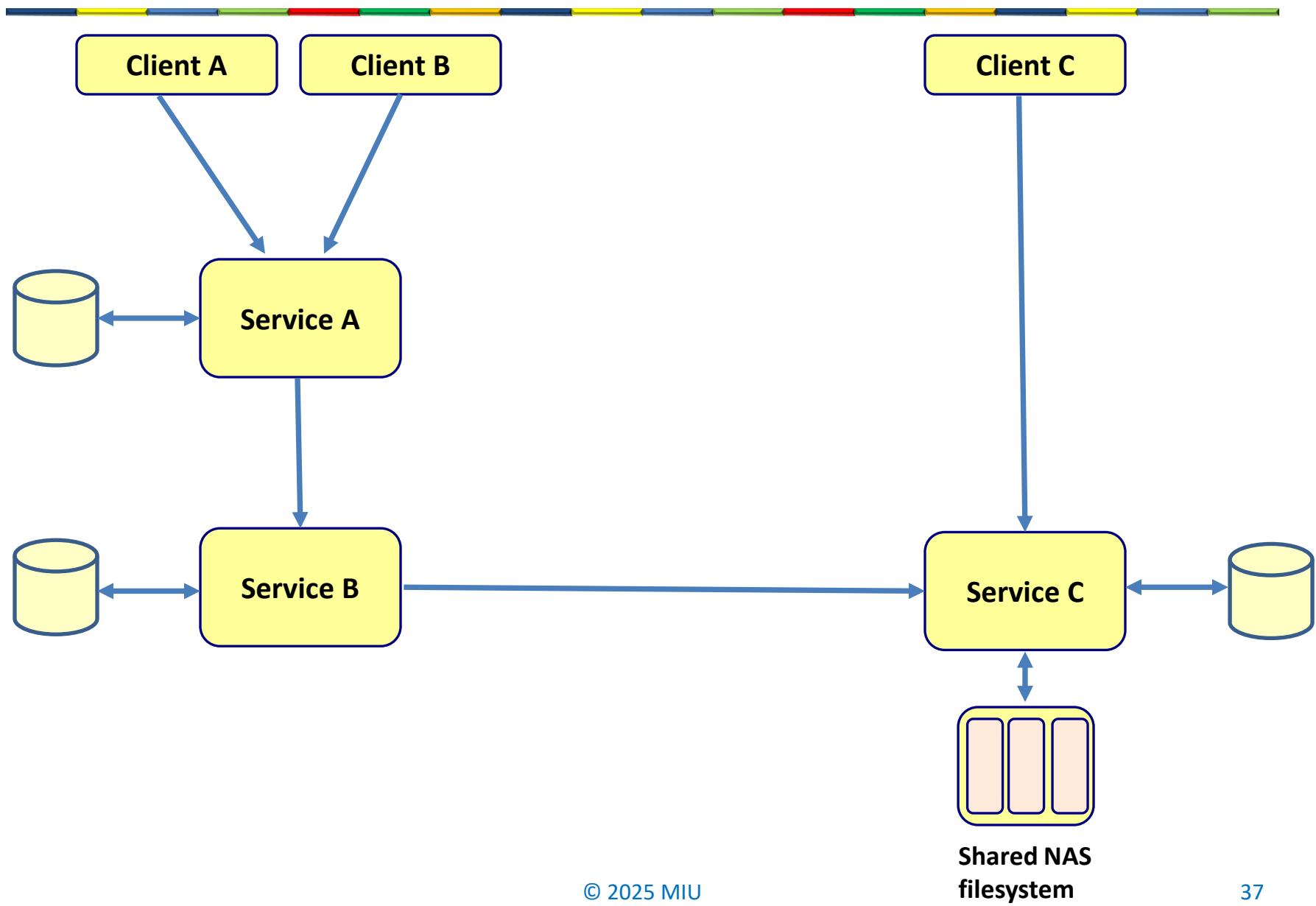
- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

Clustering

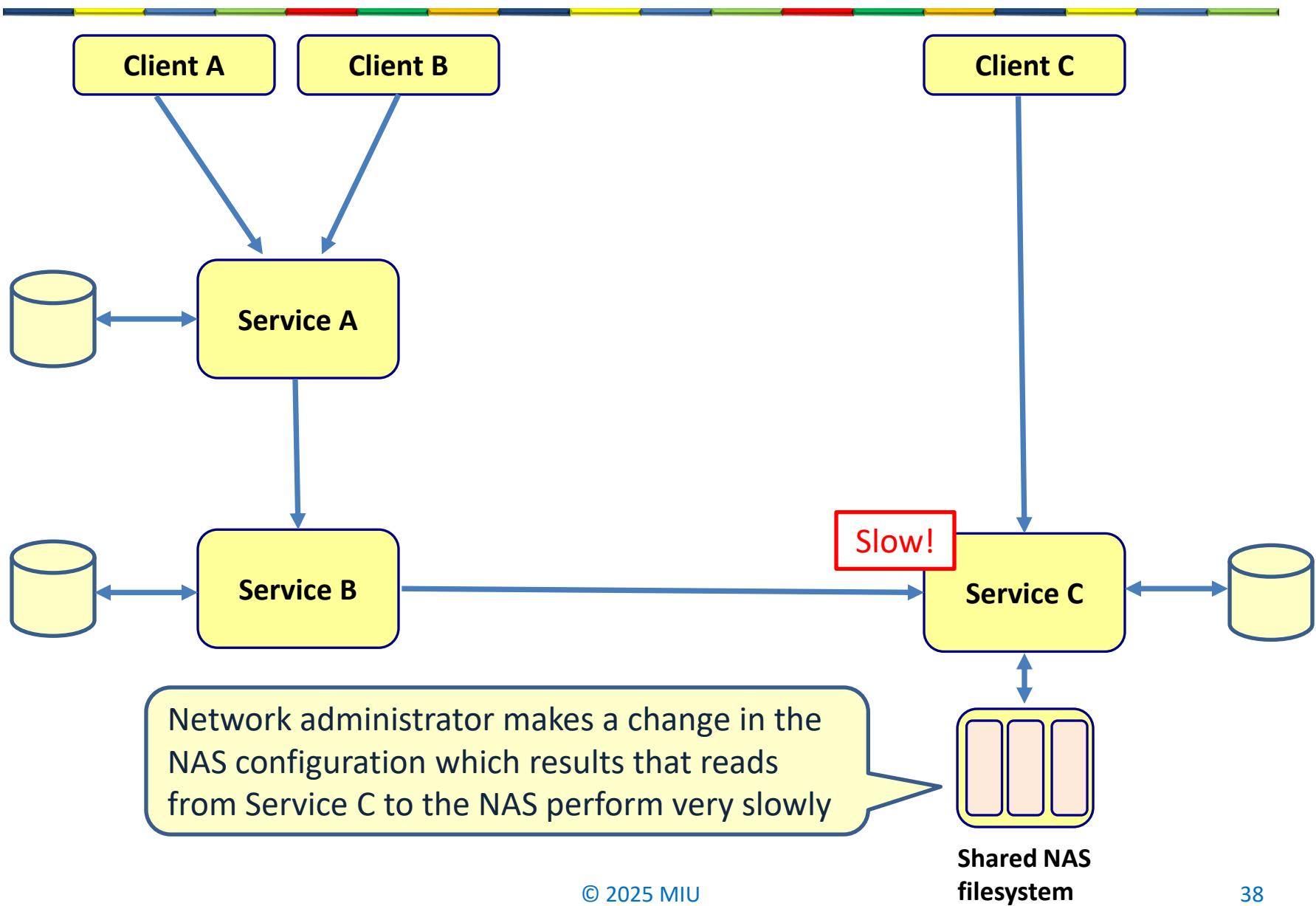
- Load balancing
- Failover



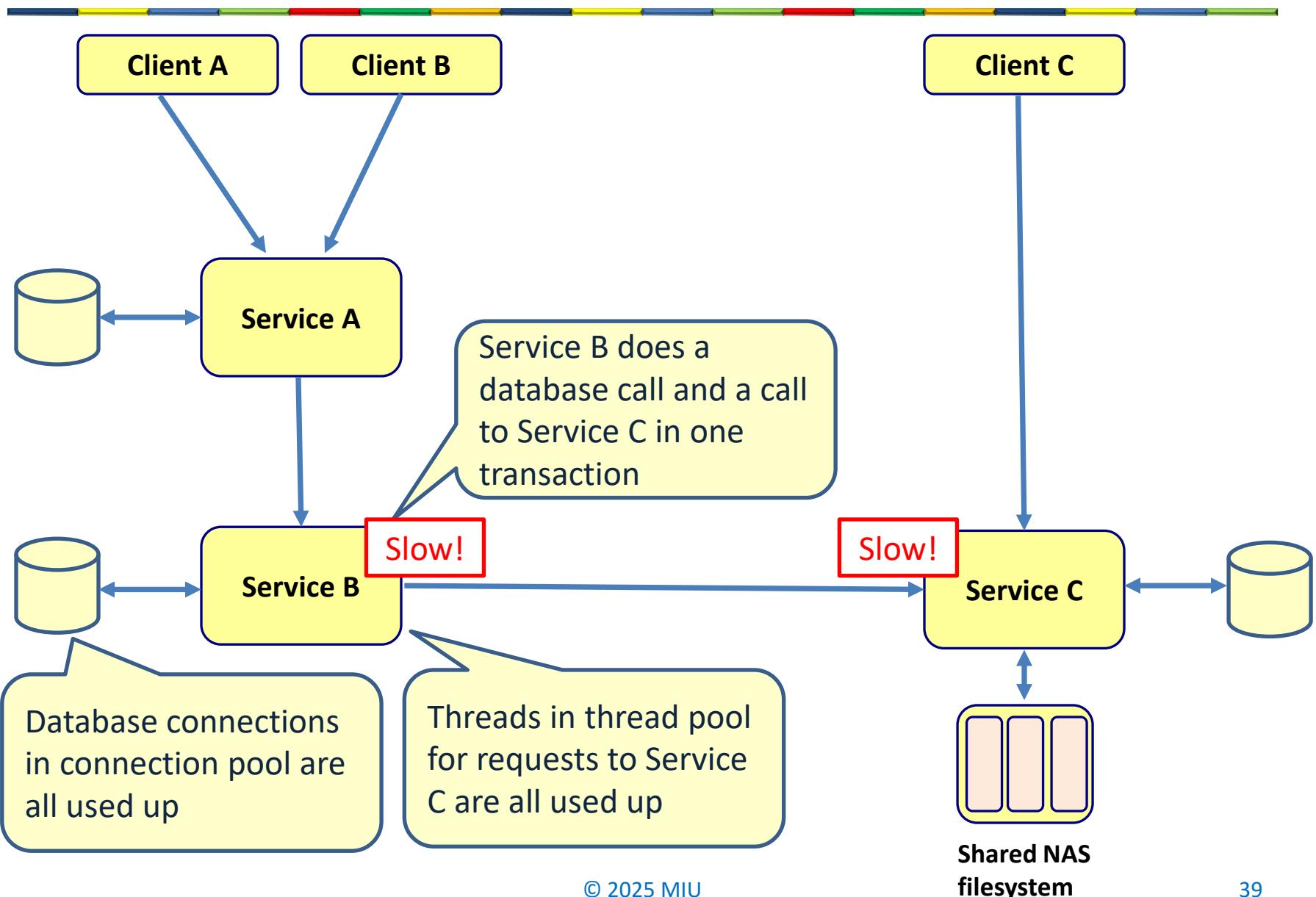
Example



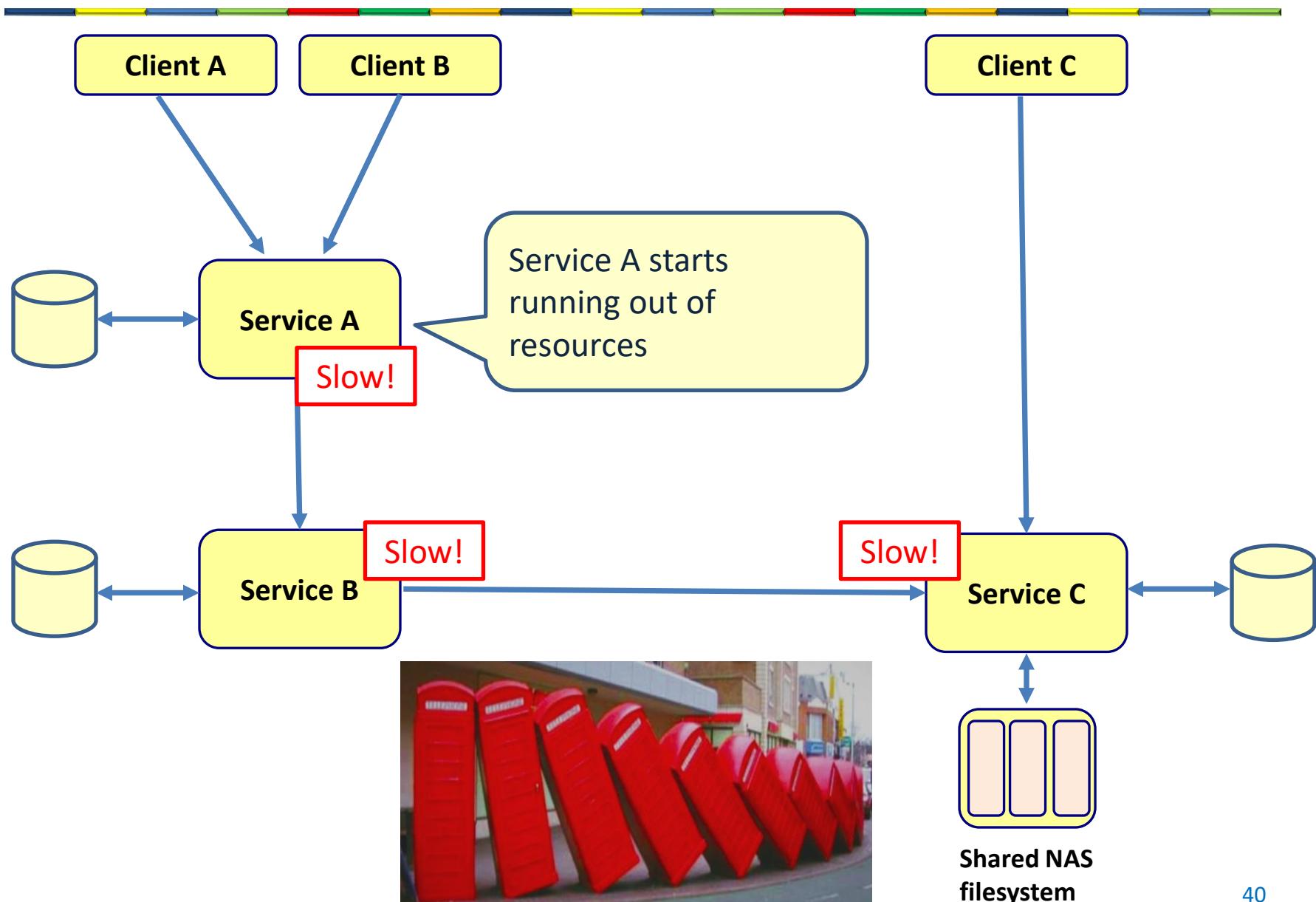
Example



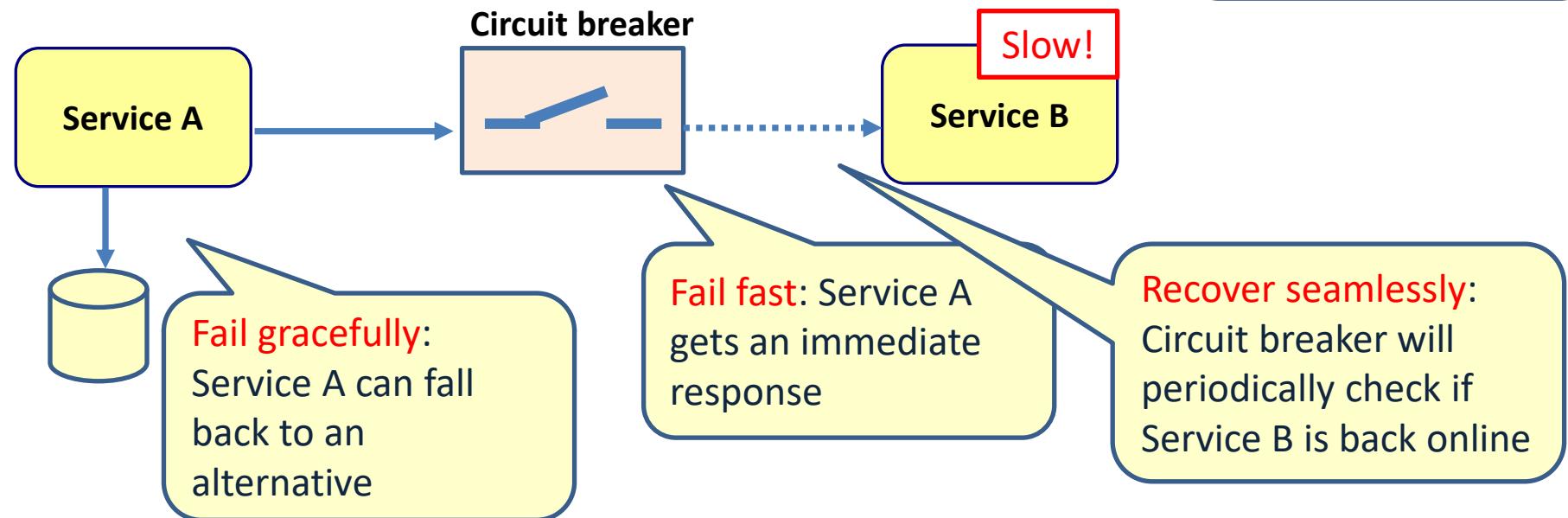
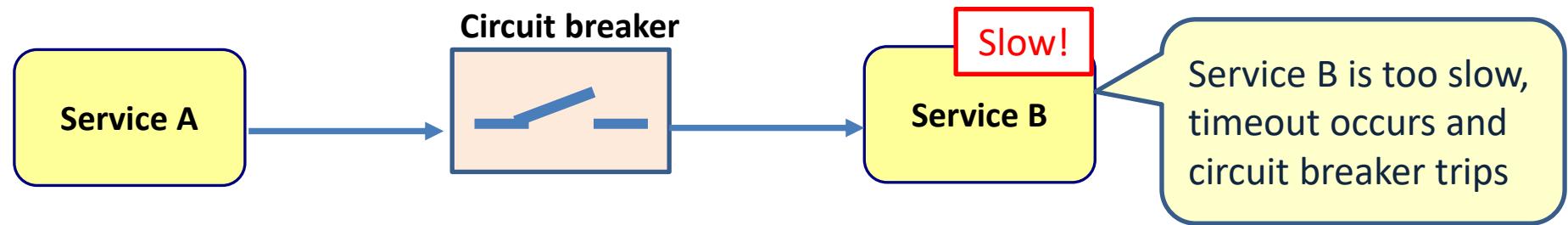
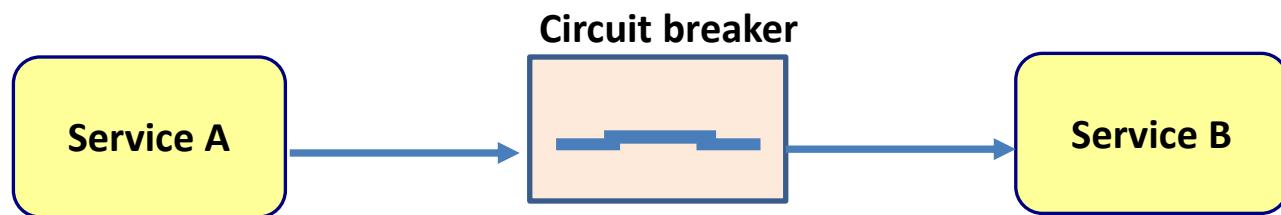
Example



Example



Circuit breaker

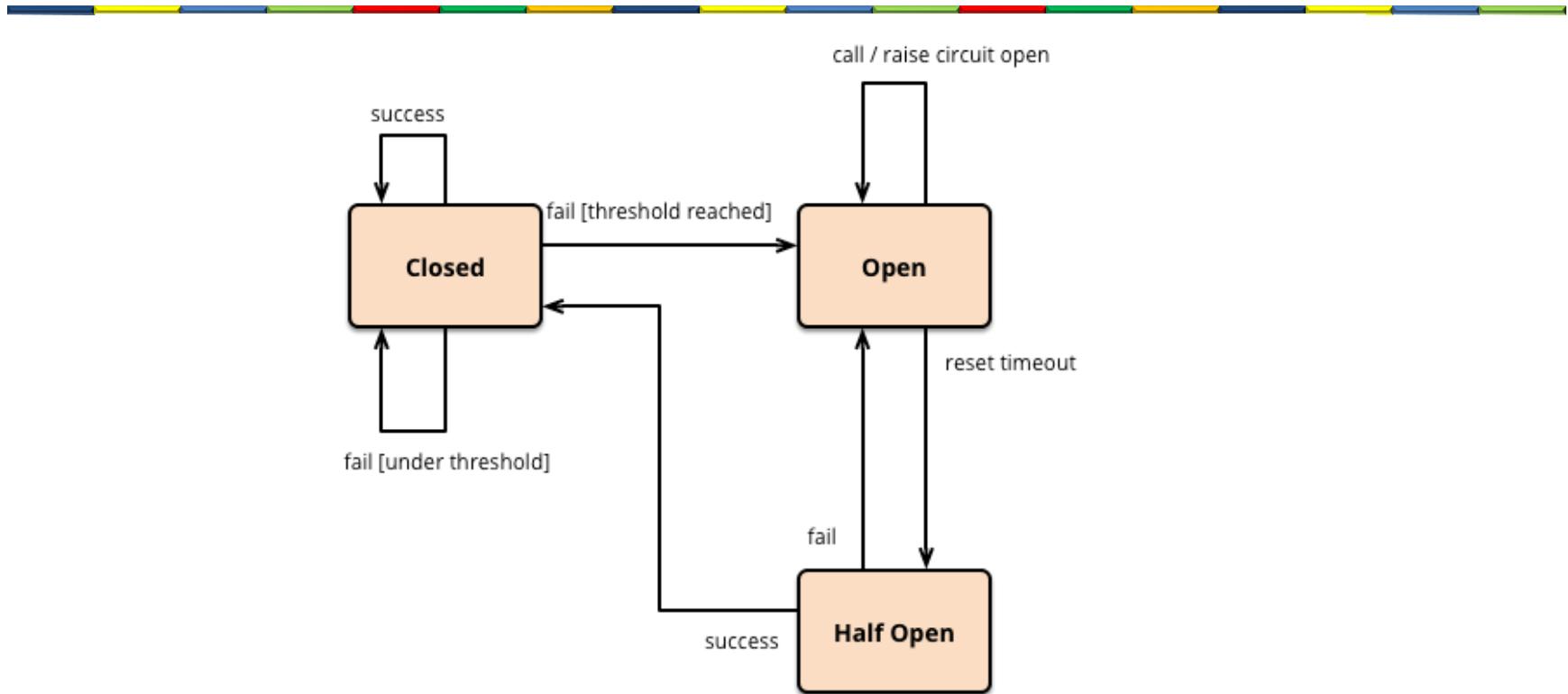


Main point

- A circuit breaker takes care that not the whole microservice architecture gets slow when one service becomes slow.
- Every relative part of creation is connected at the level of pure consciousness.

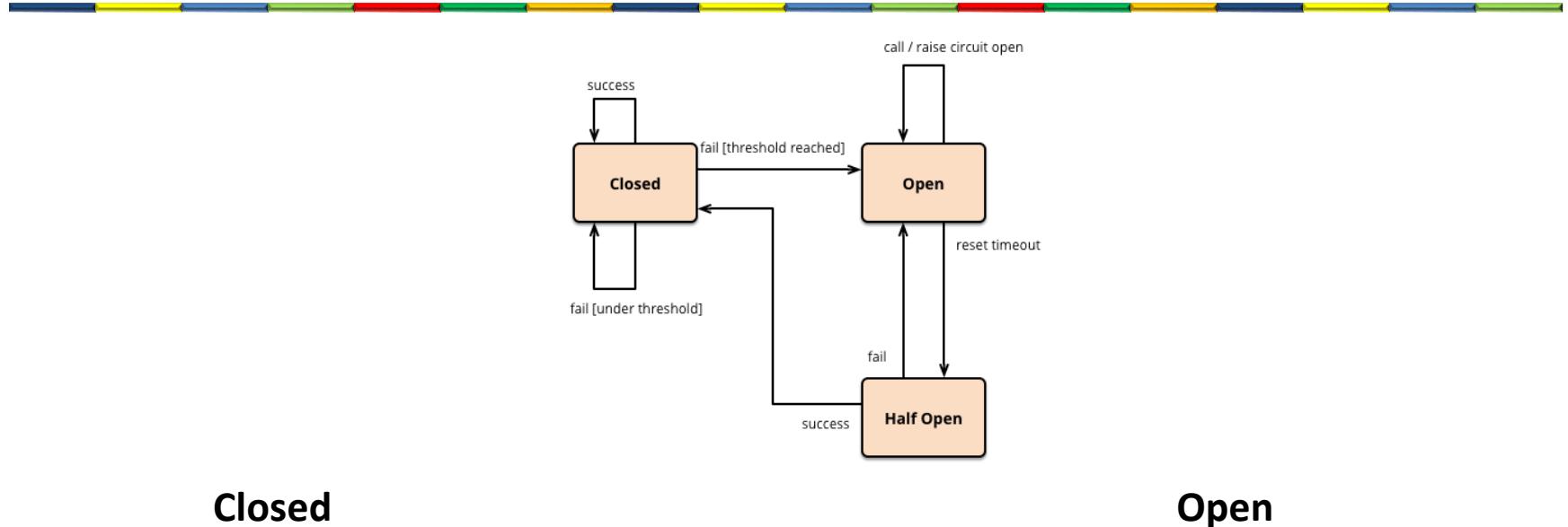
RESILIENCE: RESILIENCE4J

Resilience4J



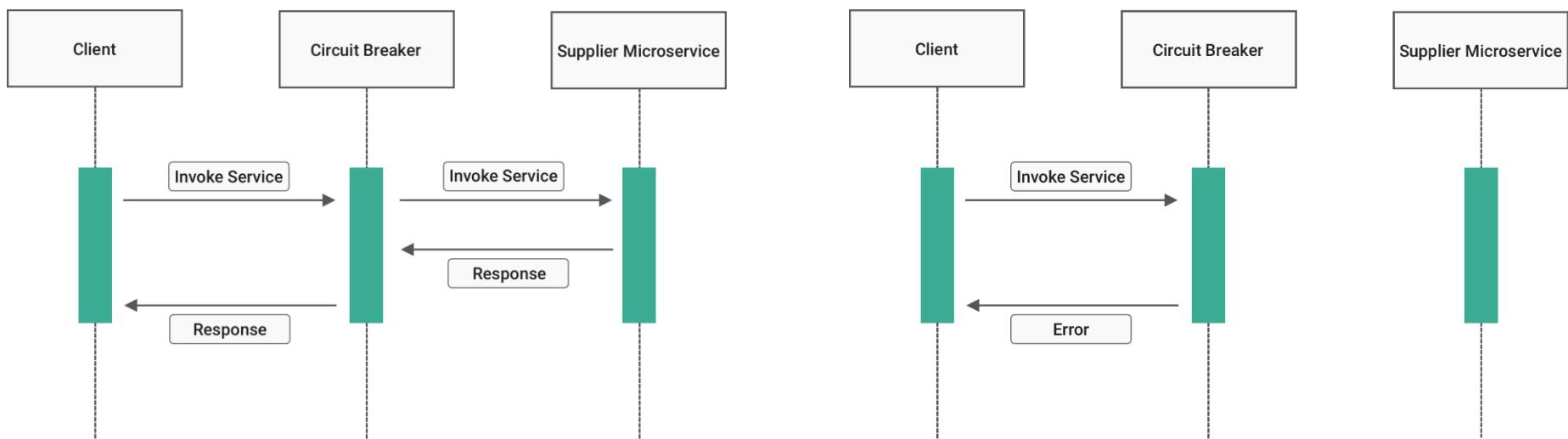
CircuitBreaker State	Method Executed?	Fallback Called?
CLOSED	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> if method throws
OPEN	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> always
HALF-OPEN	<input checked="" type="checkbox"/> Some allowed	<input checked="" type="checkbox"/> if failure

Resilience4J



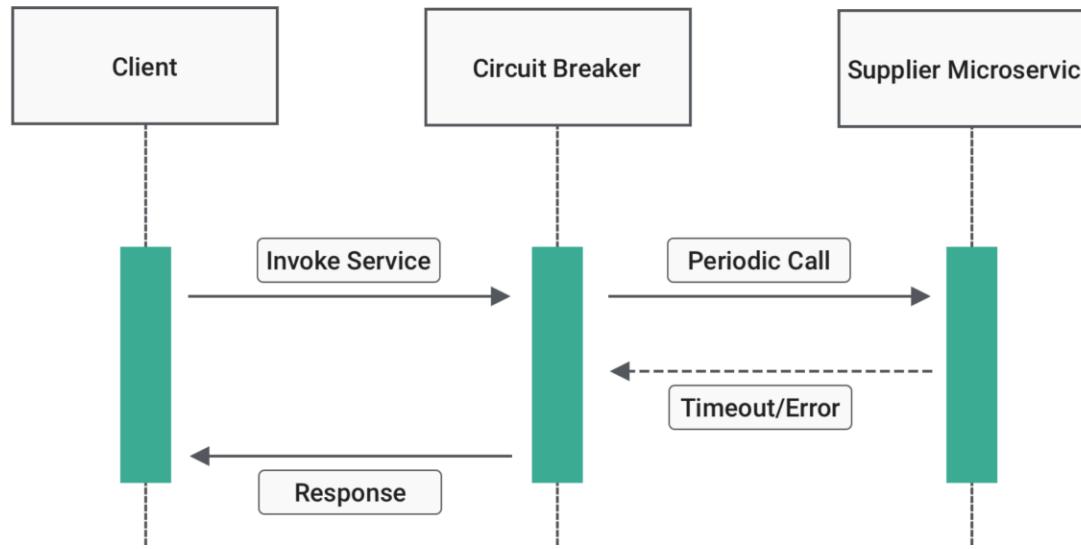
Closed

Open



Resilience4J

Half-Open



In the OPEN state and after a wait time duration has elapsed, it makes state transition from OPEN to HALF_OPEN and allows only a configurable number of calls. If the failure rate or slow call rate is greater than or equal to the configured threshold, the state changes back to OPEN. If the failure rate and slow call rate is below the threshold, the state changes back to CLOSED.

ServiceTwo

```
@RestController
public class ServiceTwoController {
    private static final Logger logger =
LoggerFactory.getLogger(ServiceTwoController.class.getName());

    @RequestMapping("/text")
    public String getText() {
        return "World";
    }
}
```

ServiceOne

```
@RestController
public class ServiceOneController {
    @Autowired
    private ServiceTwoClient serviceTwoClient;

    @CircuitBreaker(name = "demoCircuitBreaker", fallbackMethod = "fallbackMethod")
    @RequestMapping("/text")
    public String getText() {
        String service2Text = serviceTwoClient.getText();
        return "Hello "+ service2Text;
    }
    private String fallbackMethod(Throwable throwable) {
        return "Hello World from fallbackMethod";
    }

    @FeignClient("ServiceTwo")
    interface ServiceTwoClient {
        @RequestMapping("/text")
        public String getText();
    }
}
```

Fallback method

configuration



```
server:                                application.yml
  port: 9093

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

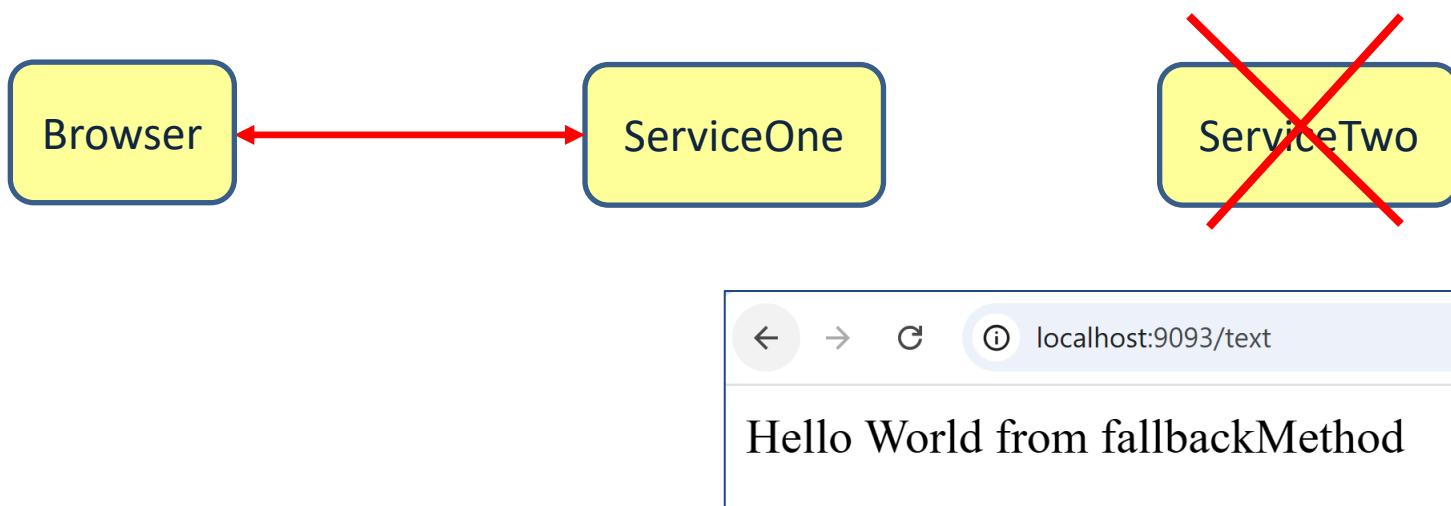
resilience4j:
  circuitbreaker:
    instances:
      demoCircuitBreaker:
        slidingWindowSize: 5
        minimumNumberOfCalls: 3
        failureRateThreshold: 50
        waitDurationInOpenState: 5s
```

dependencies

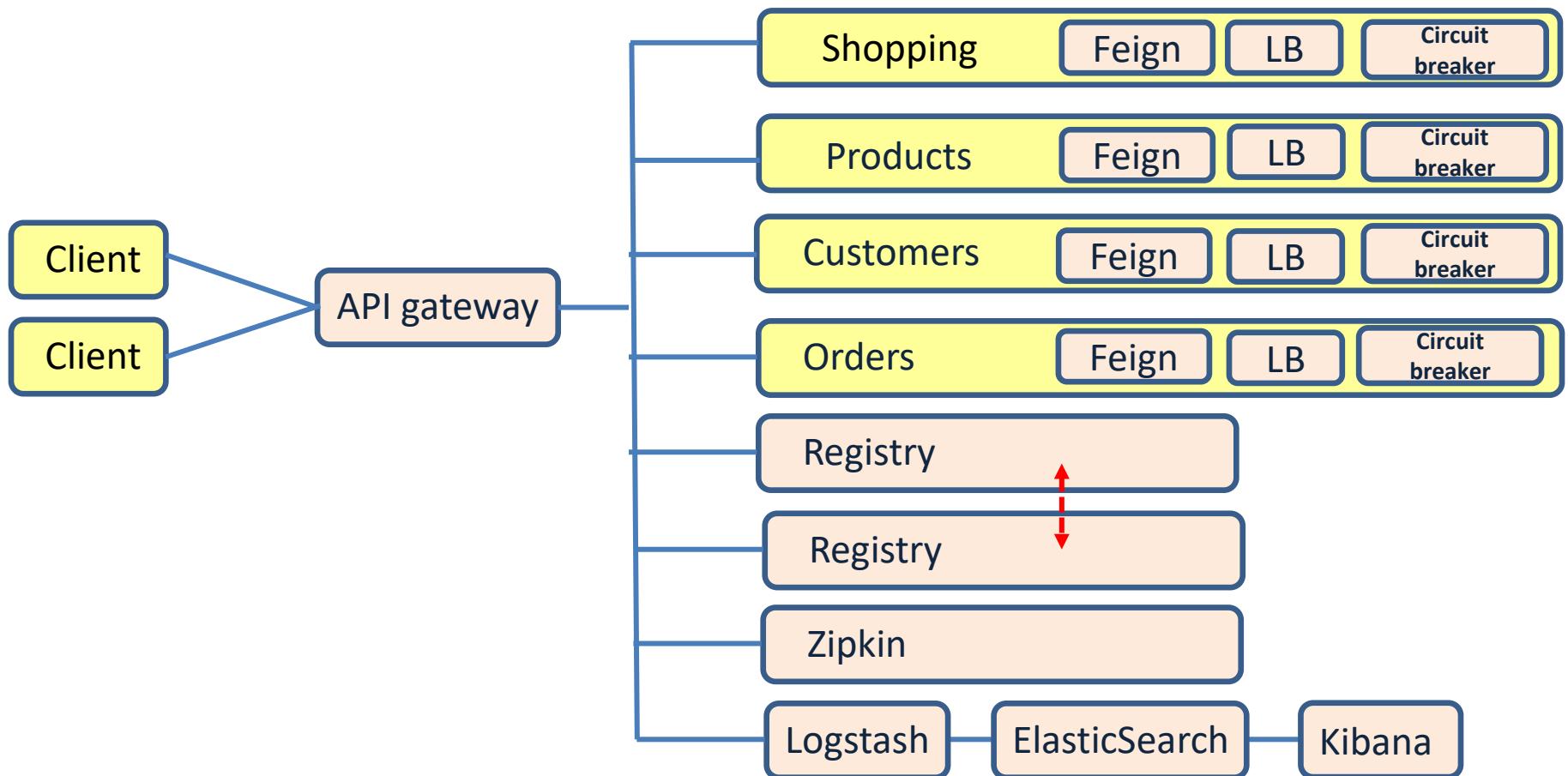
```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-consul-discovery</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
  </dependency>
</dependencies>
```

pom.xml

Using Resilience4J



Implementing microservices

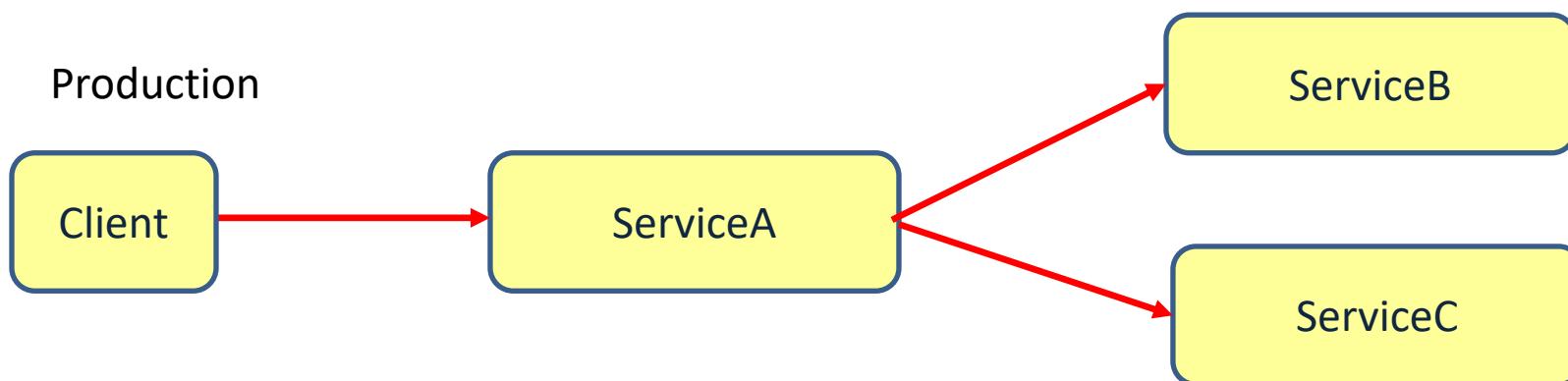


Challenges of a microservice architecture

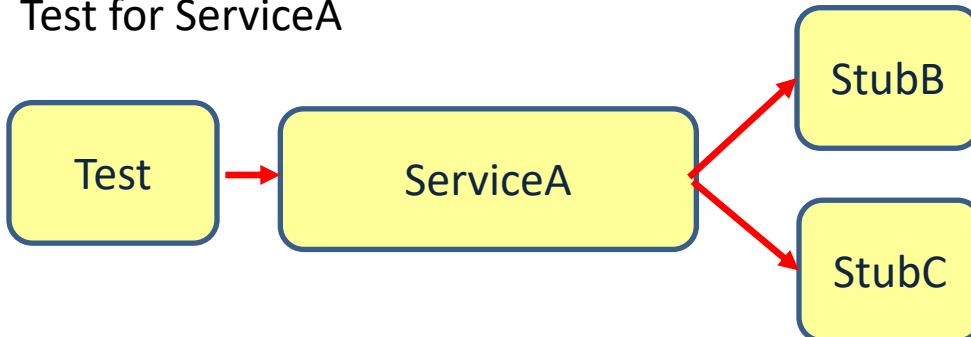
Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	
Keep configuration in sync	
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK

SPRING CLOUD CONTRACT

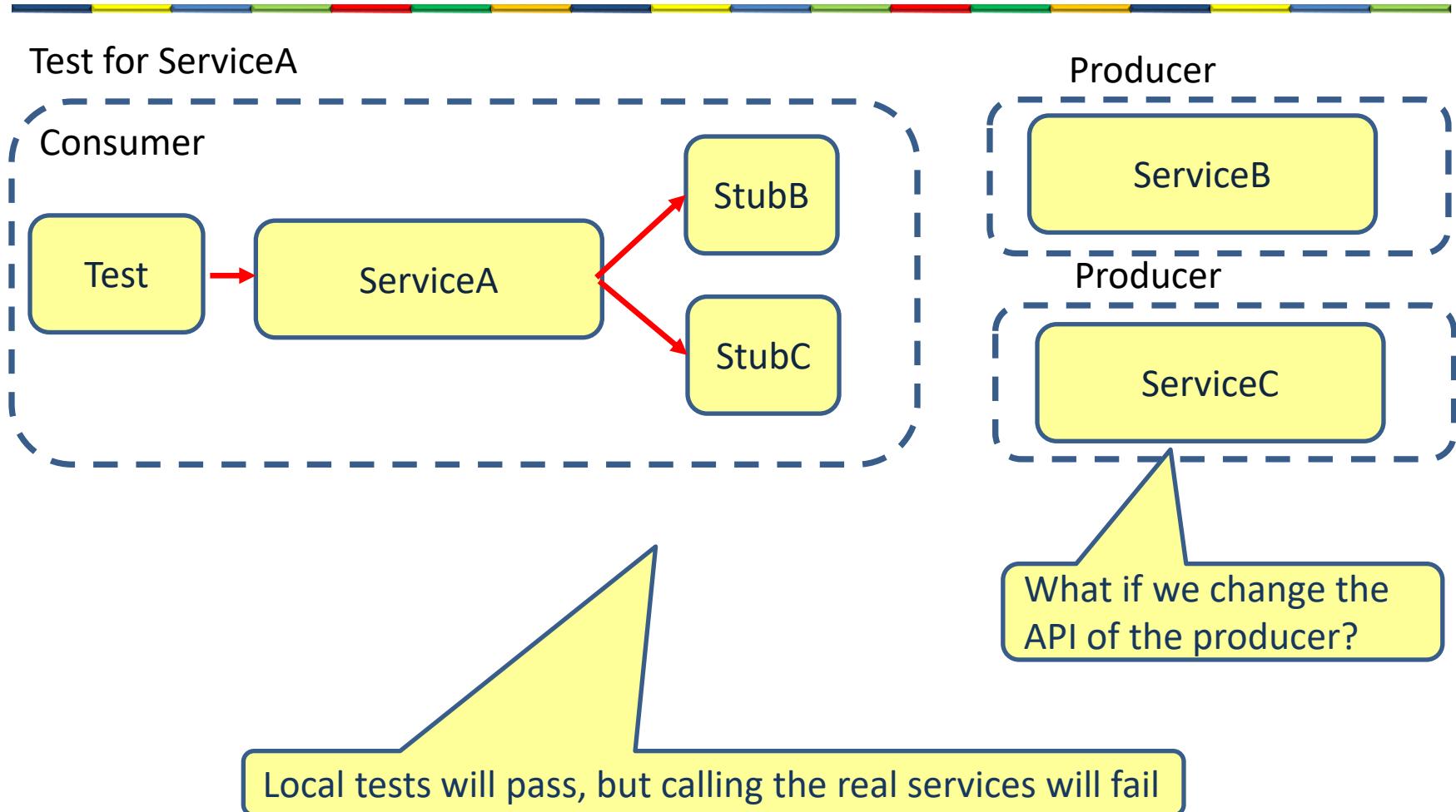
How to test microservices



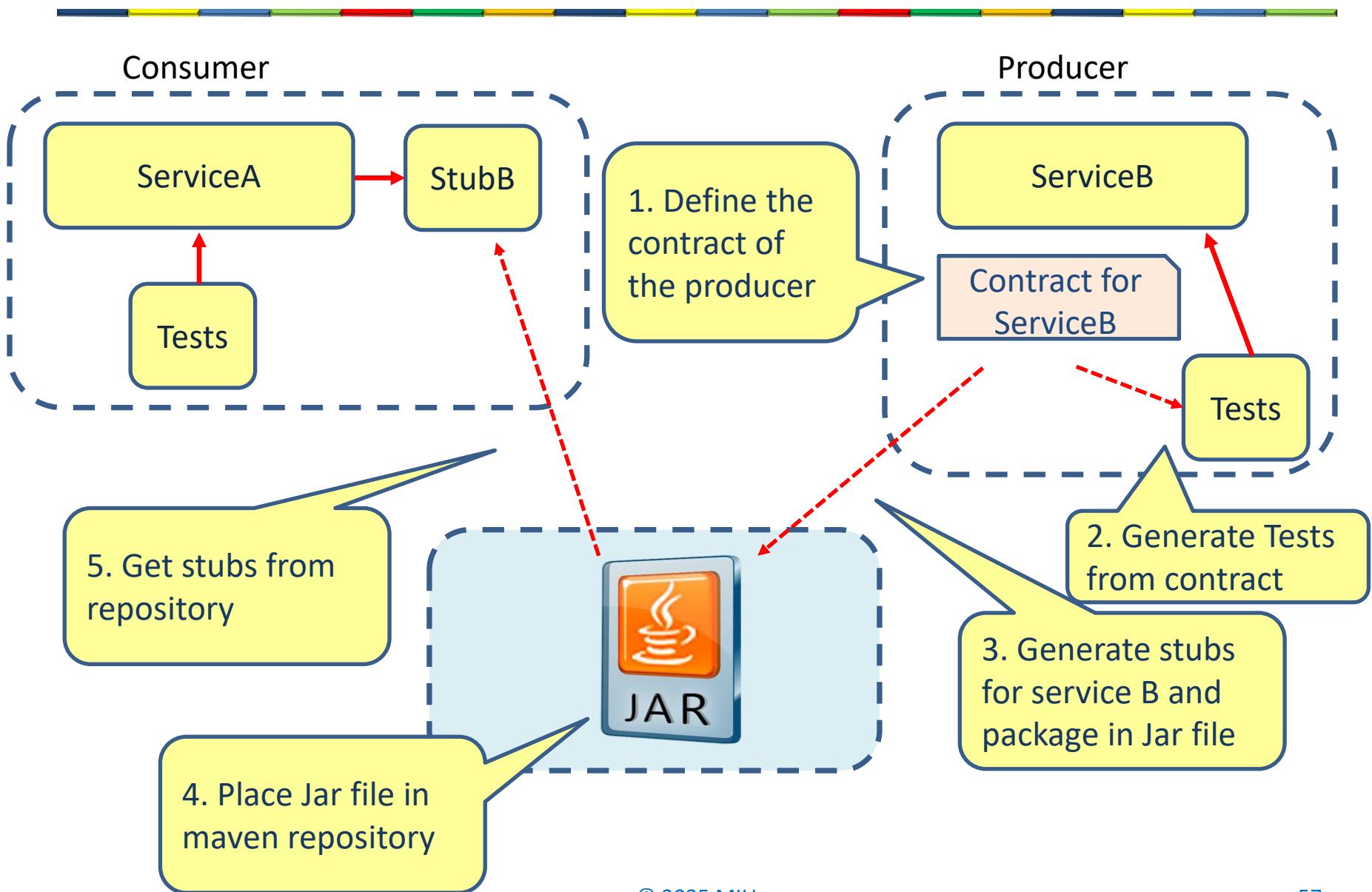
Test for ServiceA



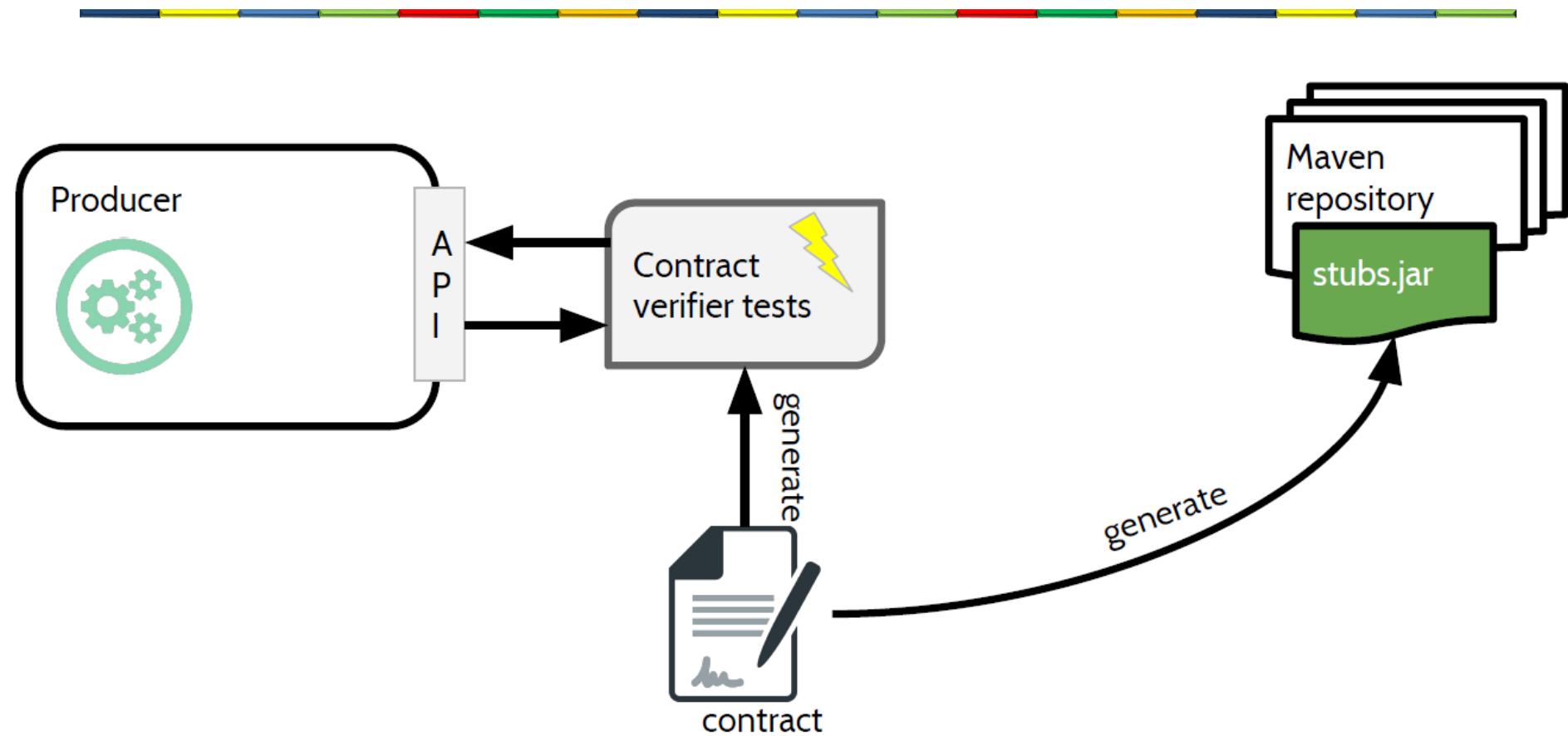
Stubs live at the consumer



Spring cloud contracts



Producer



Producer maven configuration

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-contract-verifier</artifactId>
  <scope>test</scope>
</dependency>

<plugin>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-contract-maven-plugin</artifactId>
  <version>2.2.2.RELEASE</version>
  <extensions>true</extensions>
  <configuration>
    <baseClassForTests>service.BaseTestClass</baseClassForTests>
    <testFramework>JUNIT5</testFramework>
  </configuration>
</plugin>
```

Producer

```
@RestController
public class EvenOddController {

    @GetMapping("/validate")
    public String evenOrOdd(@RequestParam("number") Integer number) {
        return number % 2 == 0 ? "Even" : "Odd";
    }
}
```

```
@SpringBootApplication
public class EvenoddServiceApplication {

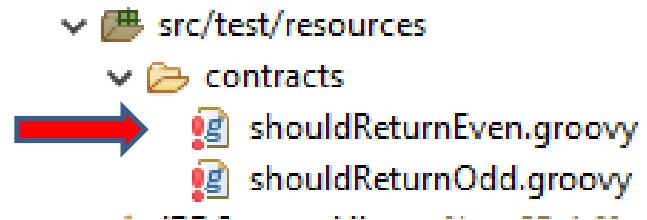
    public static void main(String[] args) {
        SpringApplication.run(EvenoddServiceApplication.class, args);
    }
}
```

Producer contract 1

```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
    description "should return even when number input is even"
    request{
        method GET()
        url("/validate") {
            queryParameters {
                parameter("number", "2")
            }
        }
    }
    response {
        body("Even")
        status 200
    }
}
```

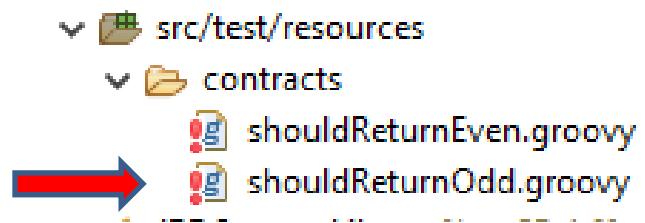
Contract in groovy



Producer contract 2

```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
    description "should return odd when number input is odd"
    request {
        method GET()
        url("/validate") {
            queryParameters {
                parameter("number", "1")
            }
        }
    }
    response {
        body("Odd")
        status 200
    }
}
```



Producer: base test class

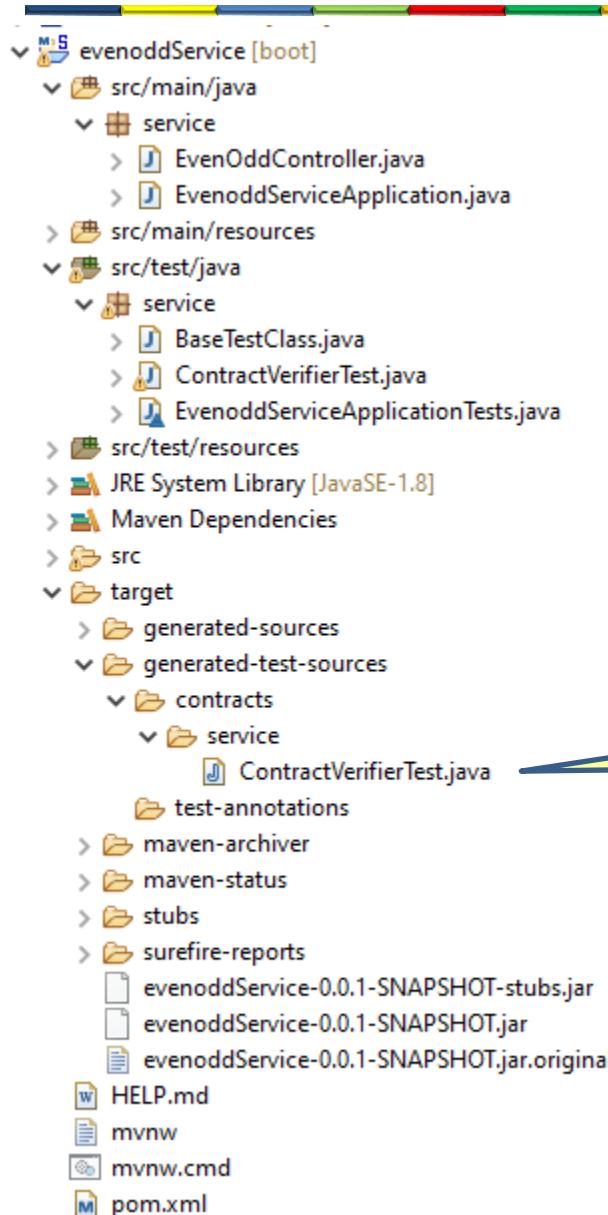
```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@DirtiesContext
@Autowired
public class BaseTestClass {

    @Autowired
    private EvenOddController evenOddController;

    @BeforeEach
    public void setup() {
        StandaloneMockMvcBuilder standaloneMockMvcBuilder
            = MockMvcBuilders.standaloneSetup(evenOddController);
        RestAssuredMockMvc.standaloneSetup(standaloneMockMvcBuilder);
    }
}
```

This is the base class for all to be generated test classes

After running maven install



Generated test class based on the contract

Generated stub classes to be used by the consumer. This jar will be placed in the local maven repository

Generated tests

```
@SuppressWarnings("rawtypes")
public class ContractVerifierTest extends BaseTestClass {

    @Test
    public void validate_shouldReturnEven() throws Exception {
        // given:
        MockMvcRequestSpecification request = given();

        // when:
        ResponseOptions response = given().spec(request)
            .queryParam("number","2")
            .get("/validate");
        // then:
        assertThat(response.statusCode()).isEqualTo(200);
        // and:
        String responseBody = response.getBody().asString();
        assertThat(responseBody).isEqualTo("Even");
    }
}
```

Generated tests

```
@Test
public void validate_shouldReturnOdd() throws Exception {
    // given:
    MockMvcRequestSpecification request = given();

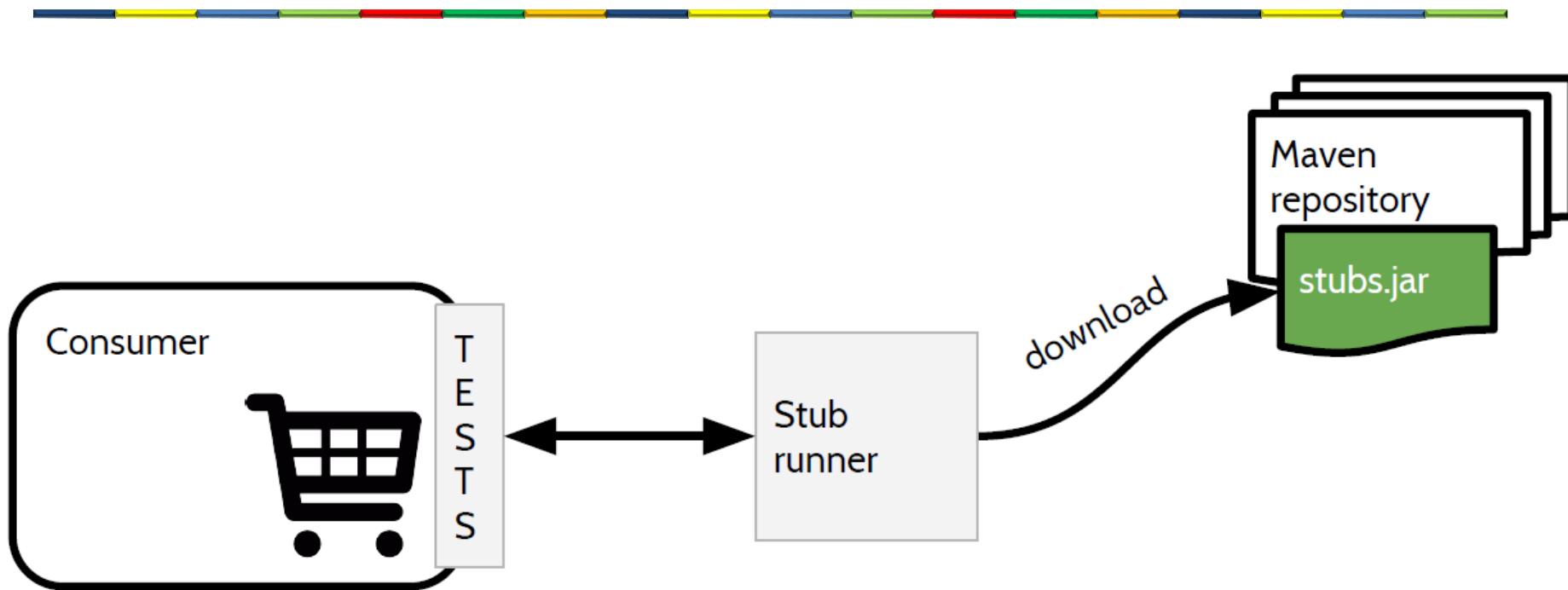
    // when:
    ResponseOptions response = given().spec(request)
        .queryParam("number","1")
        .get("/validate");
    // then:
    assertThat(response.statusCode()).isEqualTo(200);
    // and:
    String responseBody = response.getBody().asString();
    assertThat(responseBody).isEqualTo("Odd");
}
```

✓ ContractVerifierTest (service)	1 sec 307 ms
✓ validate_shouldReturnOdd()	1 sec 296 ms
✓ validate_shouldReturnEven()	11 ms

Spring cloud contract DSL

```
import org.springframework.cloud.contract.spec.Contract
Contract.make {
    description("GET employee with id=1")
    request {
        method 'GET'
        url '/employee/1'
    }
    response {
        status 200
        body("""
        {
            "id": "1",
            "fname": "Jane",
            "lname": "Doe",
            "salary": "123000.00",
            "gender": "M"
        }
        """)
        headers {
            contentType(applicationJson())
        }
    }
}
```

Consumer



```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-contract-stub-runner</artifactId>
  <version>2.2.2.RELEASE</version>
  <scope>test</scope>
</dependency>
```

Consumer

```
@RestController
public class MathController {

    private RestTemplate restTemplate = new RestTemplate();

    @GetMapping("/calculate")
    public String checkOddAndEven(@RequestParam("number") Integer number) {
        HttpHeaders httpHeaders = new HttpHeaders();
        httpHeaders.add("Content-Type", "application/json");

        ResponseEntity<String> responseEntity = restTemplate.exchange(
            "http://localhost:8090/validate?number=" + number,
            HttpMethod.GET,
            new HttpEntity<>(httpHeaders),
            String.class);

        return responseEntity.getBody();
    }
}
```

Consumer

```
@SpringBootApplication
public class MathServiceApplication {

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

}
```

Consumer test

Get the stubs from the local repository

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@AutoConfigureJsonTesters
@AutoConfigureStubRunner(stubsMode = StubRunnerProperties.StubsMode.LOCAL,
    ids = "com.acme:evenoddService:+:stubs:8090")
public class MathControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void given_WhenPassEvenNumberInQueryParam_ThenReturnEven() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=2")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(content().string("Even"));
    }

    @Test
    public void given_WhenPassOddNumberInQueryParam_ThenReturnOdd() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=1")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(content().string("Odd"));
    }
}
```

Consumer test

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@AutoConfigureJsonTesters
@AutoConfigureStubRunner(stubsMode = StubRunnerProperties.StubsMode.LOCAL,
    ids = "com.acme:evenoddService:+:stubs:8090")
public class MathControllerIntegrationTest {
```

Group id

Artifact id

Version + means latest version

stubs

Port number to run the stubs on

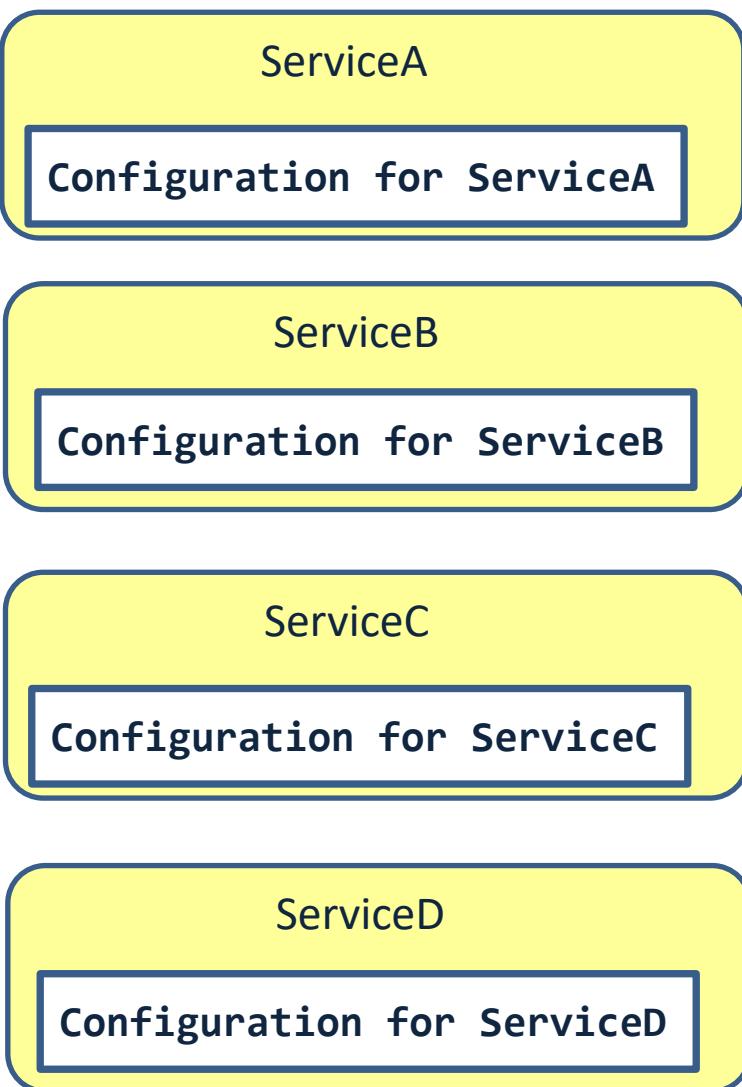
✓ ✓ MathControllerIntegrationTest (service)	792 ms
✓ given_WhenPassOddNumberInQueryParam_ThenReturnOdd()	782 ms
✓ given_WhenPassEvenNumberInQueryParam_ThenReturnEven()	10 ms

Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	
Following the process	
Keep data in sync	
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK

CENTRALIZED CONFIGURATION SERVICE

Local configuration



Local configuration challenges

- When we change the configuration we need to rebuild and redeploy the application
- Configuration may contain sensitive information
- Some of the properties are the same among services: lots of duplication

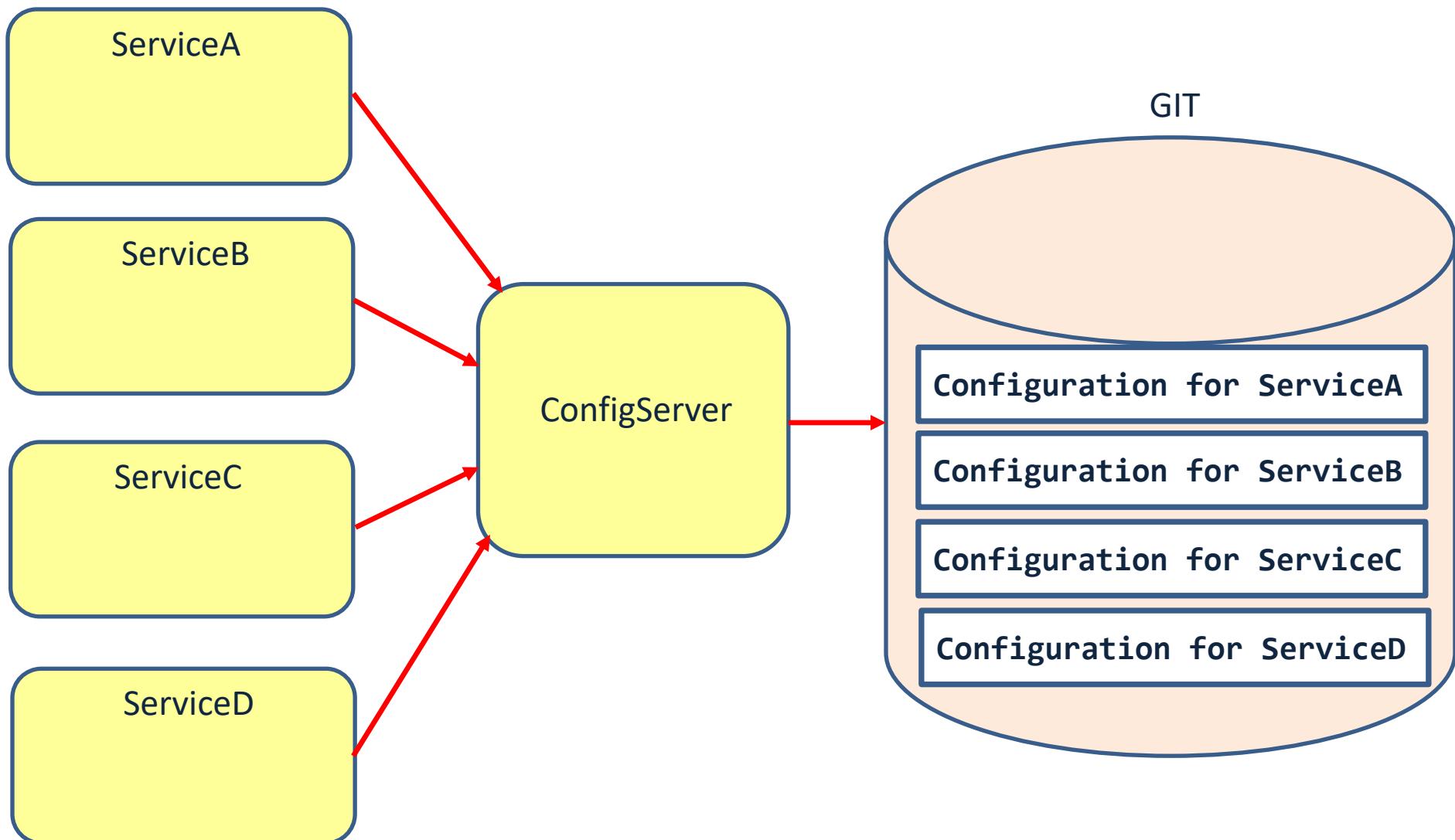
ServiceA

Configuration for ServiceA

ServiceB

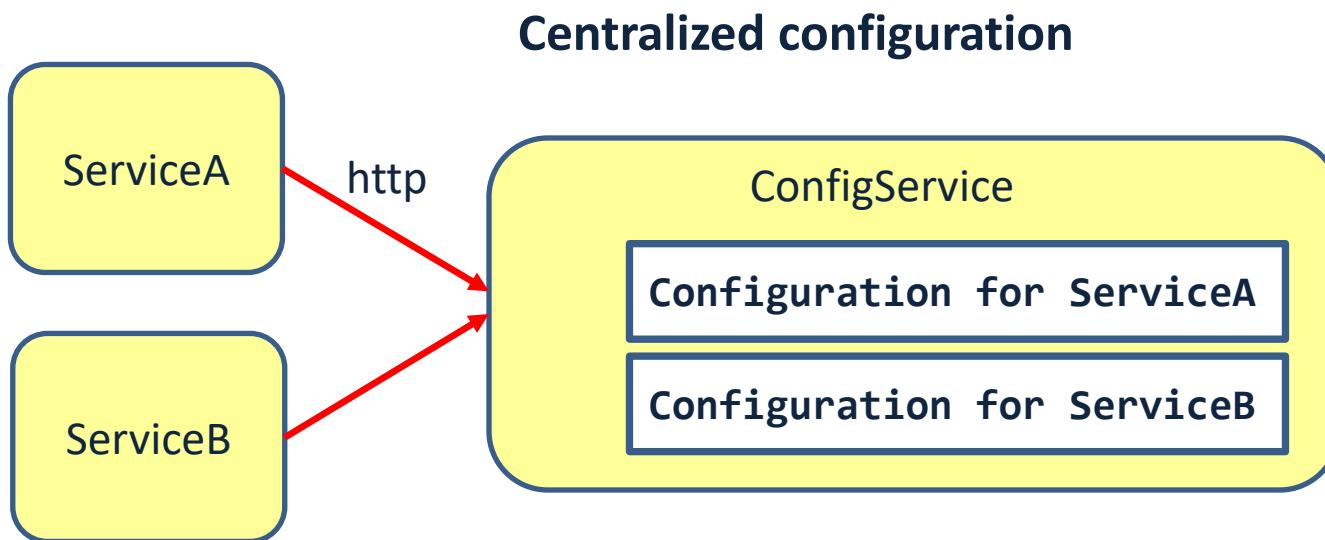
Configuration for ServiceB

Spring cloud config server



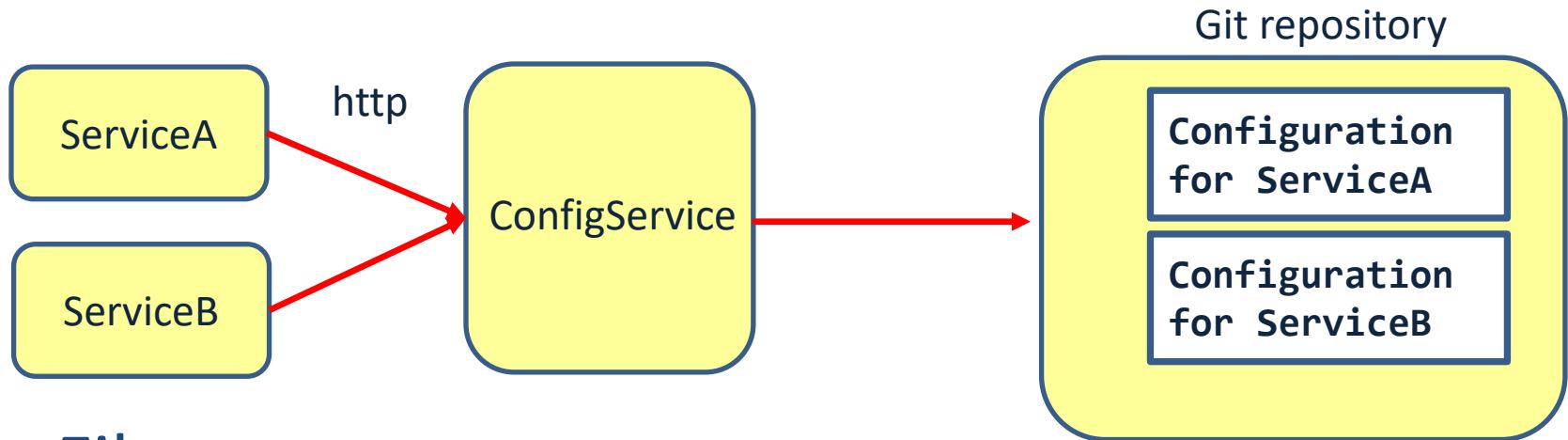
Spring cloud config

- HTTP access to centralized configuration



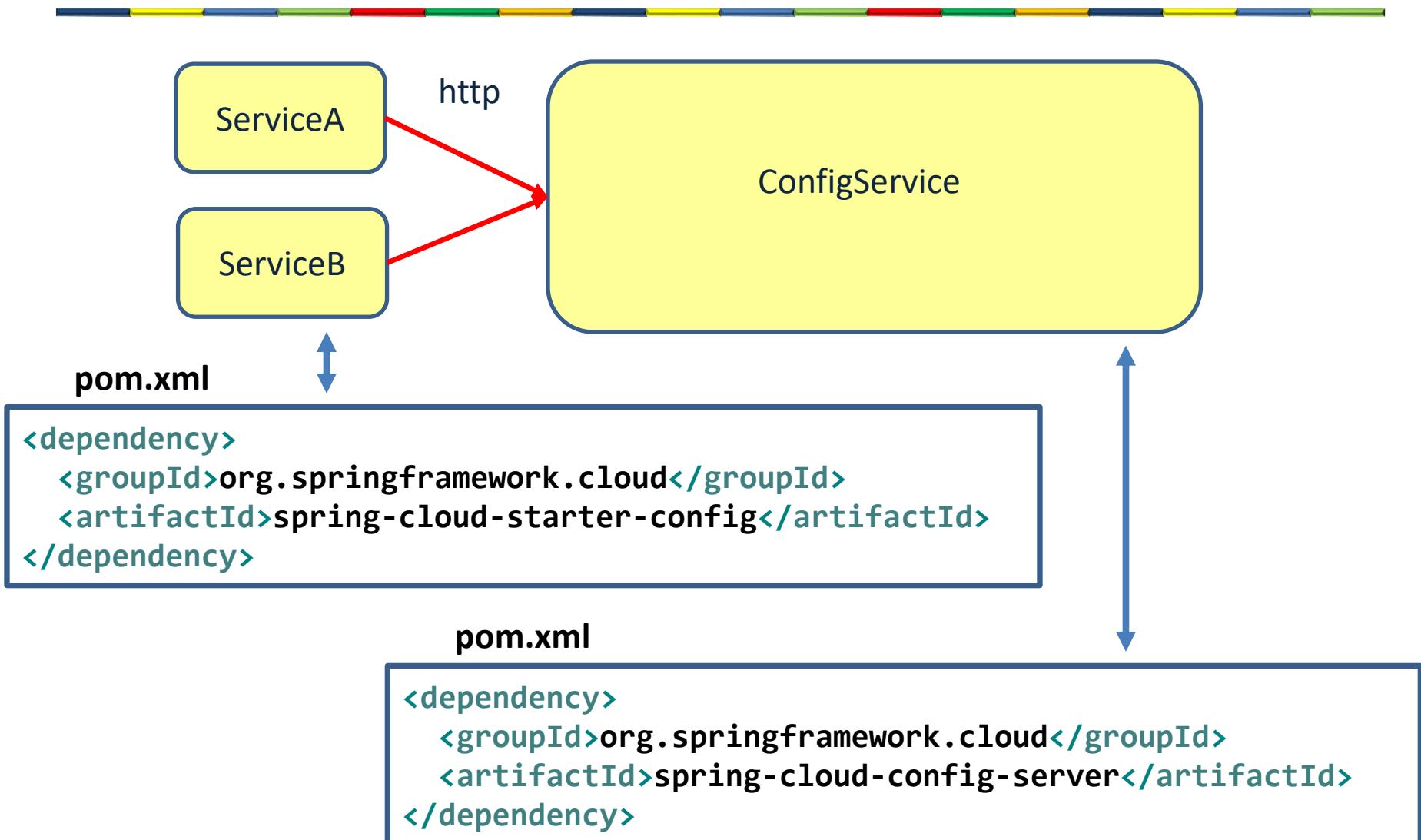
Where to store the config files?

- Git based (Github)



- File system
- HashiCorp Vault
- Database
- Cloud Secrets/Config Stores

Spring cloud config example



Configuration server

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class ConfigServiceApplication {

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

application.yml

```
server:
  port: 8888
spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/renespring/springcloud.git
```

Config files in github

The image shows two screenshots of a GitHub repository interface. Both screenshots have a header bar with the GitHub logo, user name 'renespring', repository name 'springcloud', and navigation links: Code, Issues, Pull requests, Actions, Projects, Wiki, and Security. The 'Code' link is underlined.

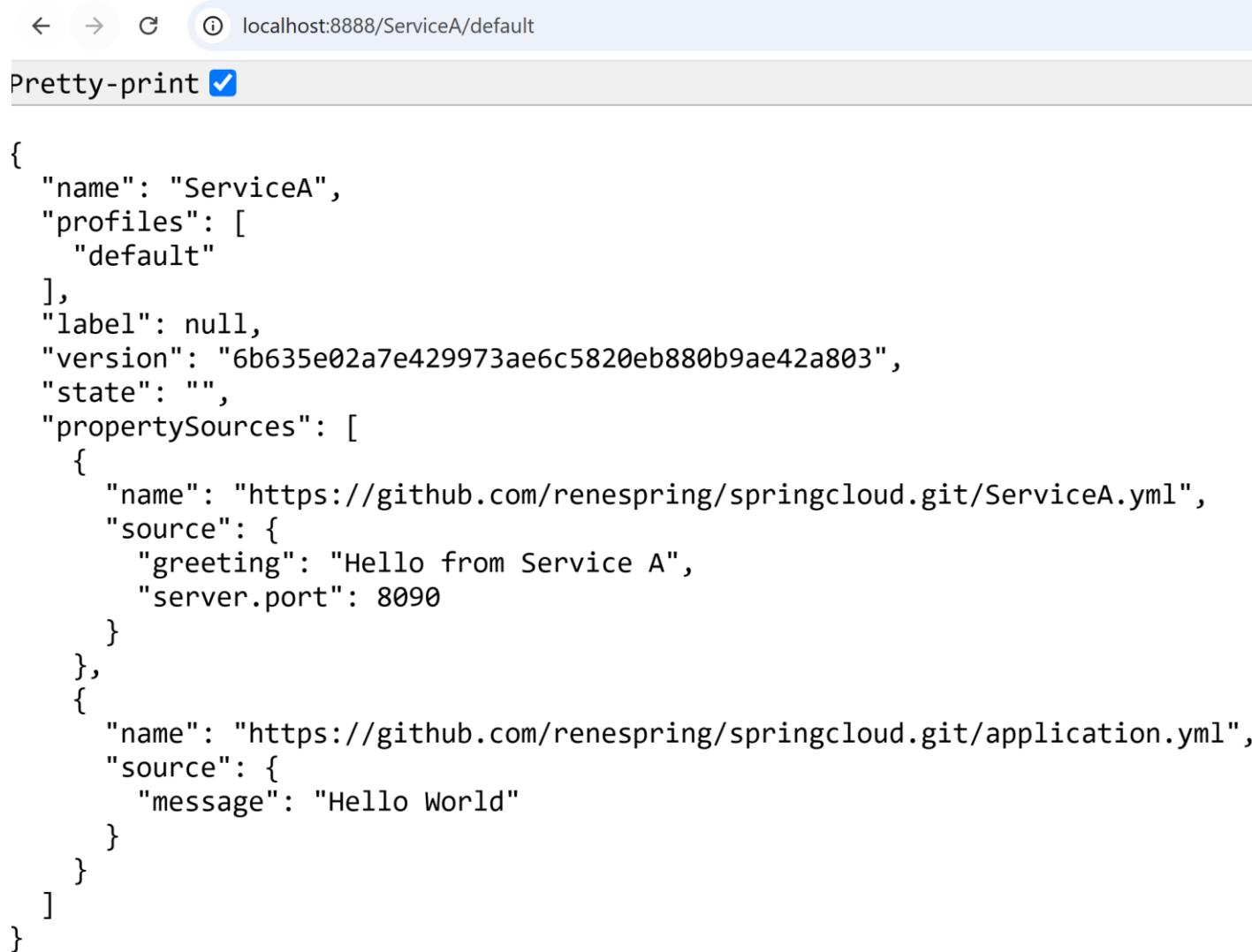
Screenshot 1 (Top): This screenshot shows the 'ServiceA.yml' file. The left sidebar lists files: README.md, ServiceA.yml (selected), ServiceB.yml, and application.yml. The right panel displays the contents of 'ServiceA.yml':

```
greeting: Hello from Service A
server:
  port: 8090
```

Screenshot 2 (Bottom): This screenshot shows the 'ServiceB.yml' file. The left sidebar lists files: README.md, ServiceA.yml, ServiceB.yml (selected), and application.yml. The right panel displays the contents of 'ServiceB.yml':

```
greeting: Hello from Service B
server:
  port: 8091
```

Configuration server



A screenshot of a web browser window displaying the configuration for ServiceA. The URL bar shows "localhost:8888/ServiceA/default". Below the URL bar, there is a "Pretty-print" checkbox which is checked. The main content area displays a JSON object representing the configuration:

```
{  
  "name": "ServiceA",  
  "profiles": [  
    "default"  
  ],  
  "label": null,  
  "version": "6b635e02a7e429973ae6c5820eb880b9ae42a803",  
  "state": "",  
  "propertySources": [  
    {  
      "name": "https://github.com/renespring/springcloud.git/ServiceA.yml",  
      "source": {  
        "greeting": "Hello from Service A",  
        "server.port": 8090  
      }  
    },  
    {  
      "name": "https://github.com/renespring/springcloud.git/application.yml",  
      "source": {  
        "message": "Hello World"  
      }  
    }  
  ]  
}
```

Configuration client: ServiceA

```
@SpringBootApplication
public class ServiceAApplication {

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

```
@RestController
public class ServiceAController {
    @Value("${greeting}")
    private String message;

    @RequestMapping("/")
    public String getName() {
        return message;
    }
}
```

application.yml

```
spring:
  application:
    name: ServiceA
  config:
    import: configserver:http://localhost:8888
```

Configuration client: ServiceB

```
@SpringBootApplication
public class ServiceBApplication {

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

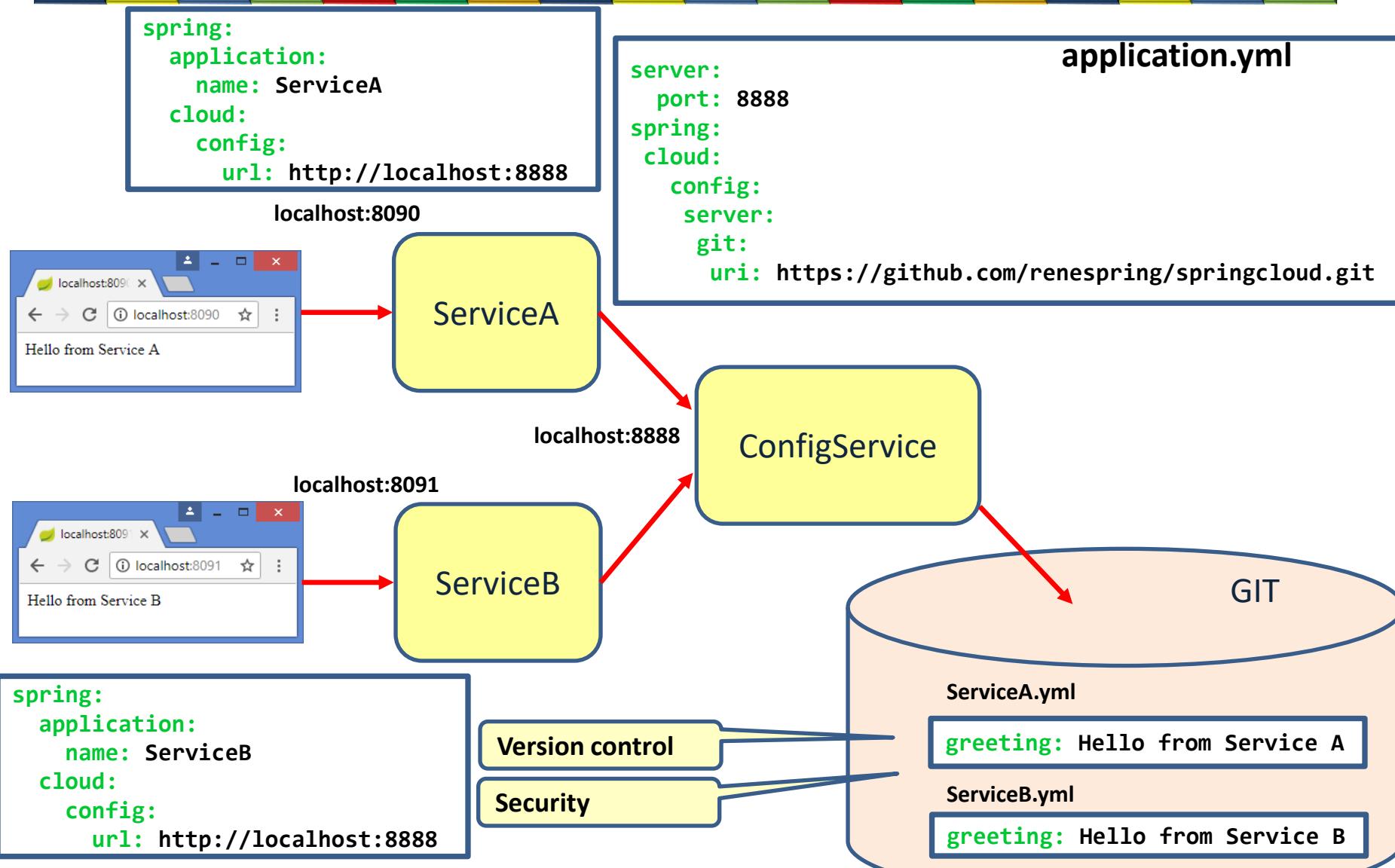
```
@RestController
public class ServiceBController {
    @Value("${greeting}")
    private String message;

    @RequestMapping("/")
    public String getName() {
        return message;
    }
}
```

application.yml

```
spring:
  application:
    name: ServiceB
  config:
    import: configserver:http://localhost:8888
```

Use of the Config Server



Shared configuration

The screenshot shows a GitHub repository interface. On the left, the 'Files' sidebar lists files: README.md, ServiceA.yml, ServiceB.yml, and application.yml, with application.yml currently selected. The main area displays the contents of the application.yml file:

```
springcloud / application.yml
```

renespring Update application.yml

Code Blame 1 lines (1 loc) • 15 Bytes

```
1 message: CS590
```

A yellow callout bubble points to the line 'message: CS590' with the text: 'Place shared configuration in application.yml'.

Shared configuration

```
@RestController  
public class ServiceAController {  
    @Value("${greeting}")  
    private String greeting;  
  
    @Value("${message}")  
    private String message;  
  
    @RequestMapping("/")  
    public String getName() {  
        return greeting +" from "+ message;  
    }  
}
```

localhost:8090

Hello from Service A from CS590

springcloud / application.yml

renespring Update application.yml

Code Blame 1 lines (1 loc) · 15 Bytes

1 message: CS590

springcloud / ServiceA.yml

renespring Update ServiceA.yml

Code Blame 5 lines (3 loc) · 54 Bytes

1 greeting: Hello from Service A
2
3 server:
4 port: 8090

Refreshing configuration

- Use `@RefreshScope` and “/actuator/refresh” event

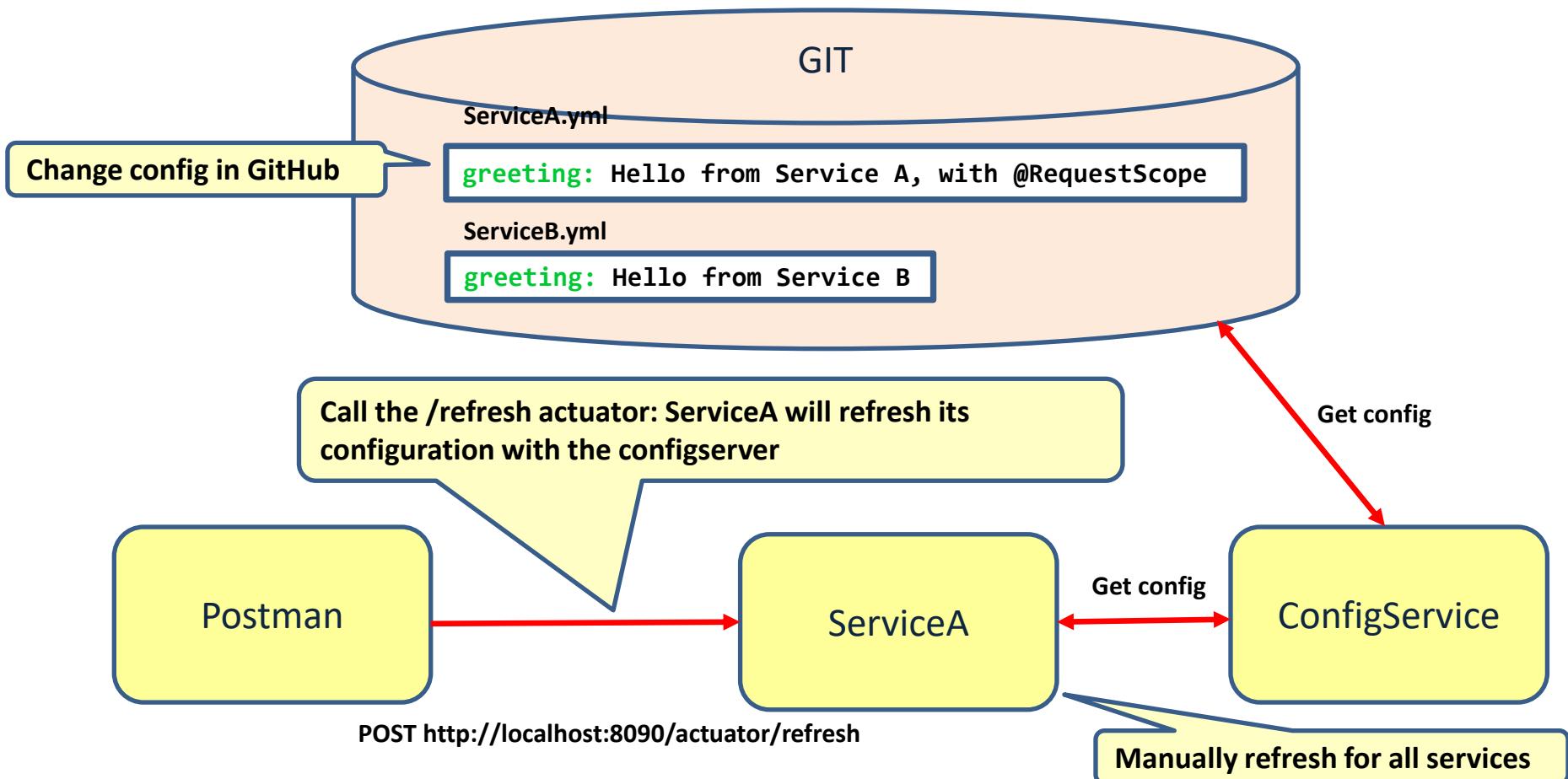
```
@RestController  
@RefreshScope  
public class ServiceAController {  
    @Value("${greeting}")  
    private String message;  
  
    @RequestMapping("/")  
    public String getName() {  
        return message;  
    }  
}
```

```
server:  
port: 8888  
  
spring:  
cloud:  
config:  
server:  
git:  
uri:  
https://github.com/renespring/springcloud.git  
  
management:  
endpoints:  
web:  
exposure:  
include: refresh
```

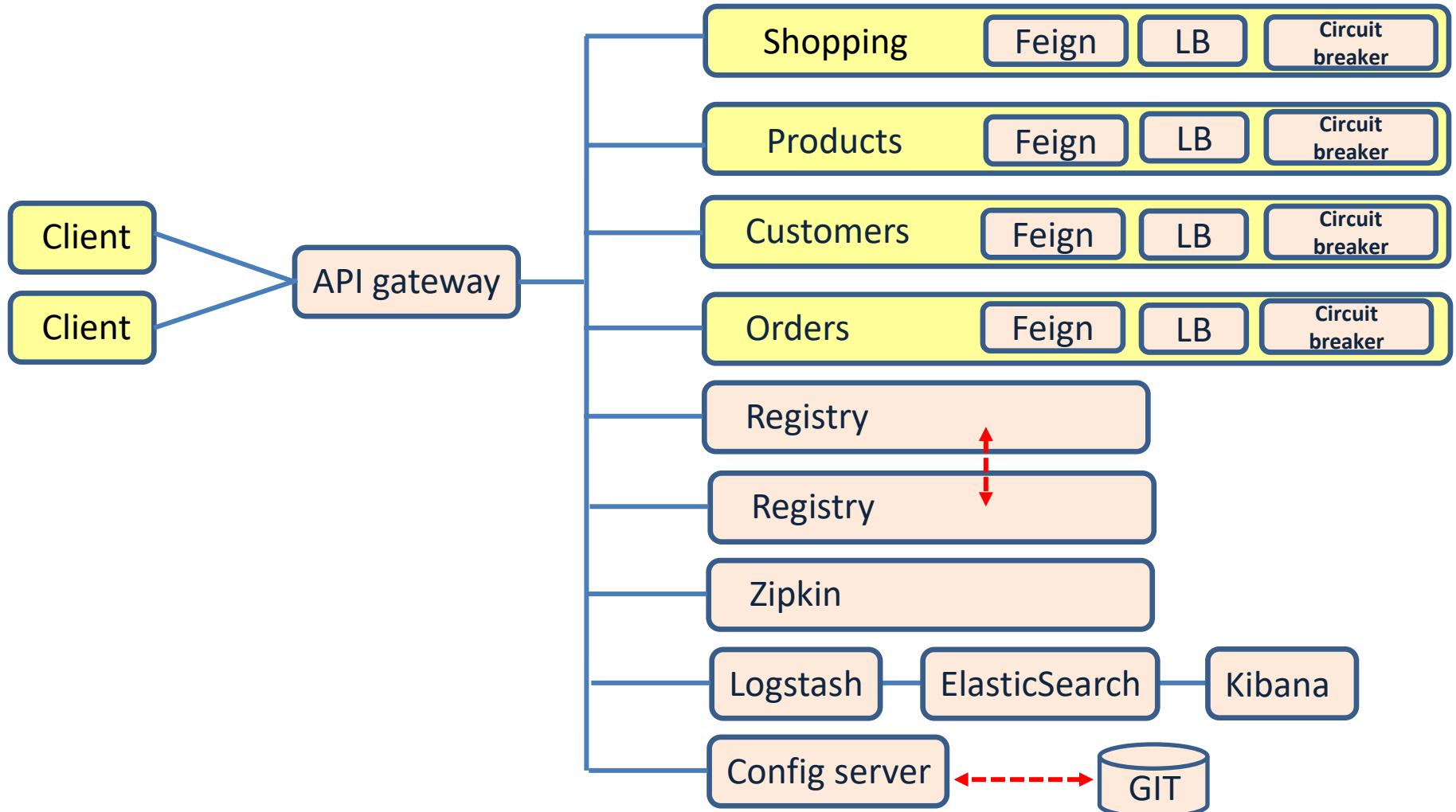
Expose the /refresh actuator

Refreshing configuration

- Use `@RefreshScope` and “`/actuator/refresh`” event



Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	
Keep data in sync	
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	Zipkin, ELK
Follow/monitor business processes	Zipkin, ELK

Lesson 9

MICROSERVICES

SERVICE DEPLOYMENT

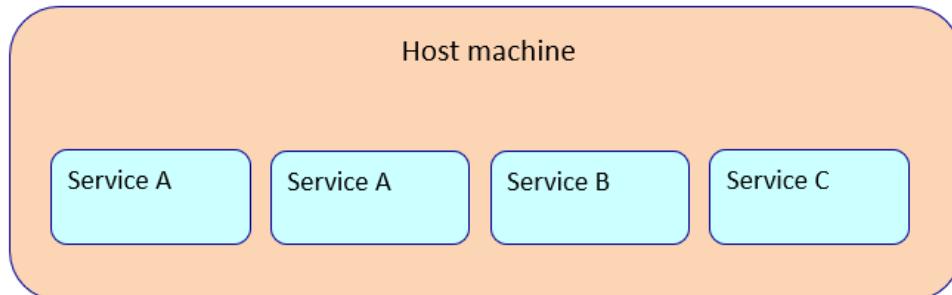
Service deployment

- Service are written using different languages, frameworks, framework versions
- Run multiple service instances of a service for throughput and availability
- Building and deploying should be fast
- Instances need to be isolated
- Constrain the resources a service may consume (CPU, memory, etc.)
- Deployment should be reliable

Multiple service instances per host

- Benefits

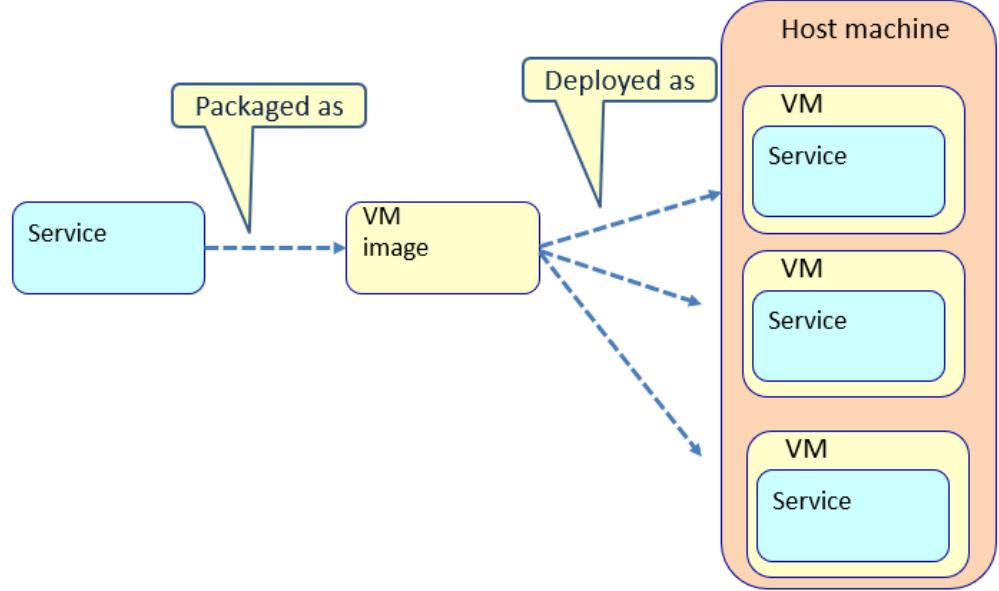
- Efficient resource utilization
- Fast deployment



- Drawbacks

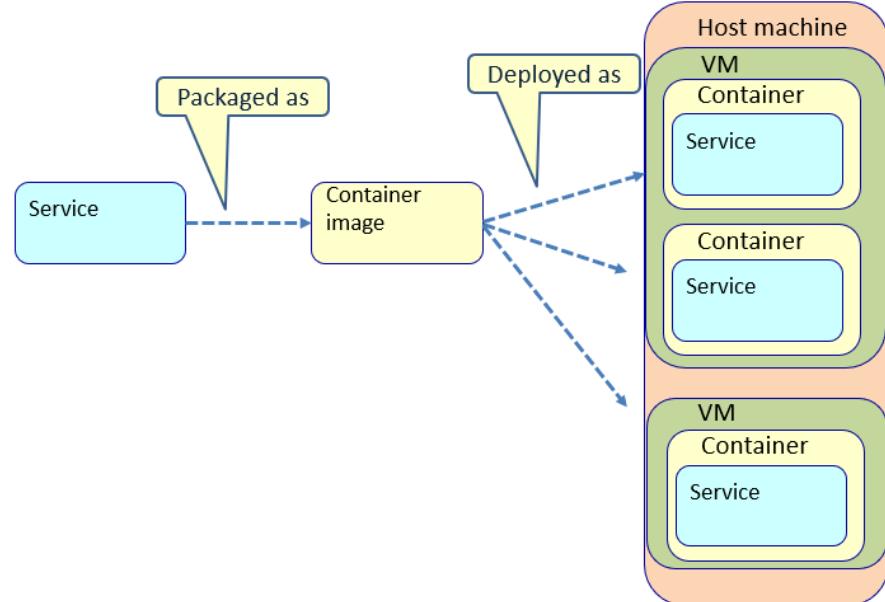
- Poor isolation
- Poor visibility of resource utilization
- Difficult to constrain resource utilization
- Risk of dependency version conflicts
- Poor encapsulation of implementation technology

Service per VM

-
- Benefits
 - Great isolation
 - Great manageability
 - VM encapsulates implementation technology
 - Leverage cloud infrastructure for auto scaling/load balancing
 - Drawbacks
 - Less efficient resource utilization
 - Slow deployment
- 

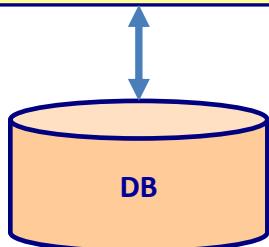
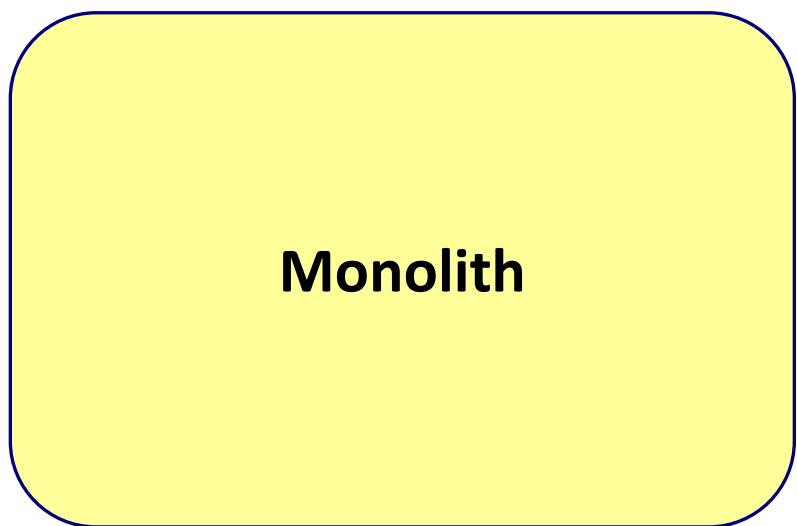
Service per container

- Benefits
 - Great isolation
 - Great manageability
 - Container encapsulates implementation technology
 - Efficient resource utilization
 - Fast deployment
- Drawbacks
 - Technology is not as mature as VM's
 - Containers are not as secure as VM's



CONTAINERS

Monolith

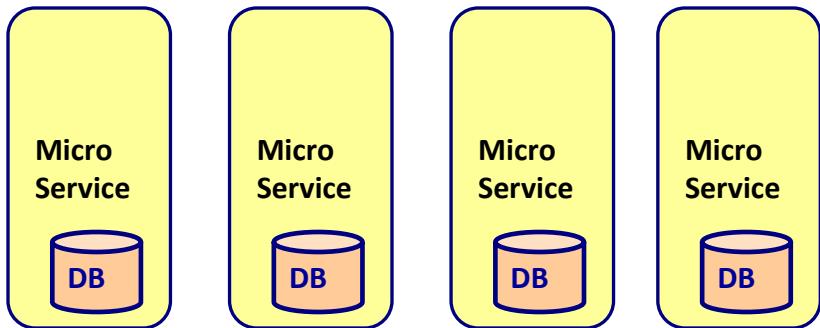


Monolith

Uses a well defined
stack
(OS, runtime,
middleware)

Runs on our private
server

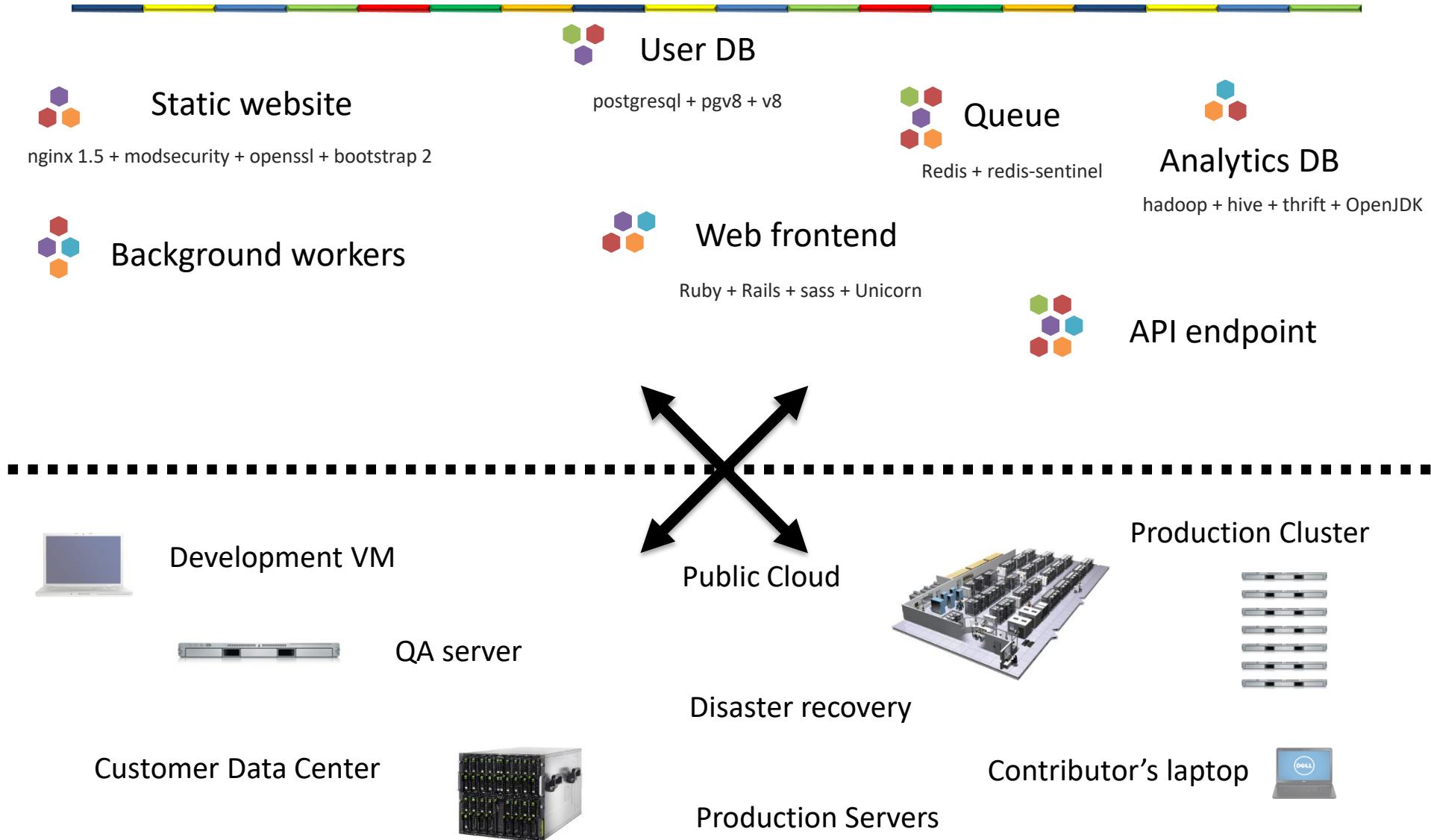
Microservice



Microservices use different stacks. (OS, runtime, middleware)

Runs anywhere (private, cloud)

The challenge



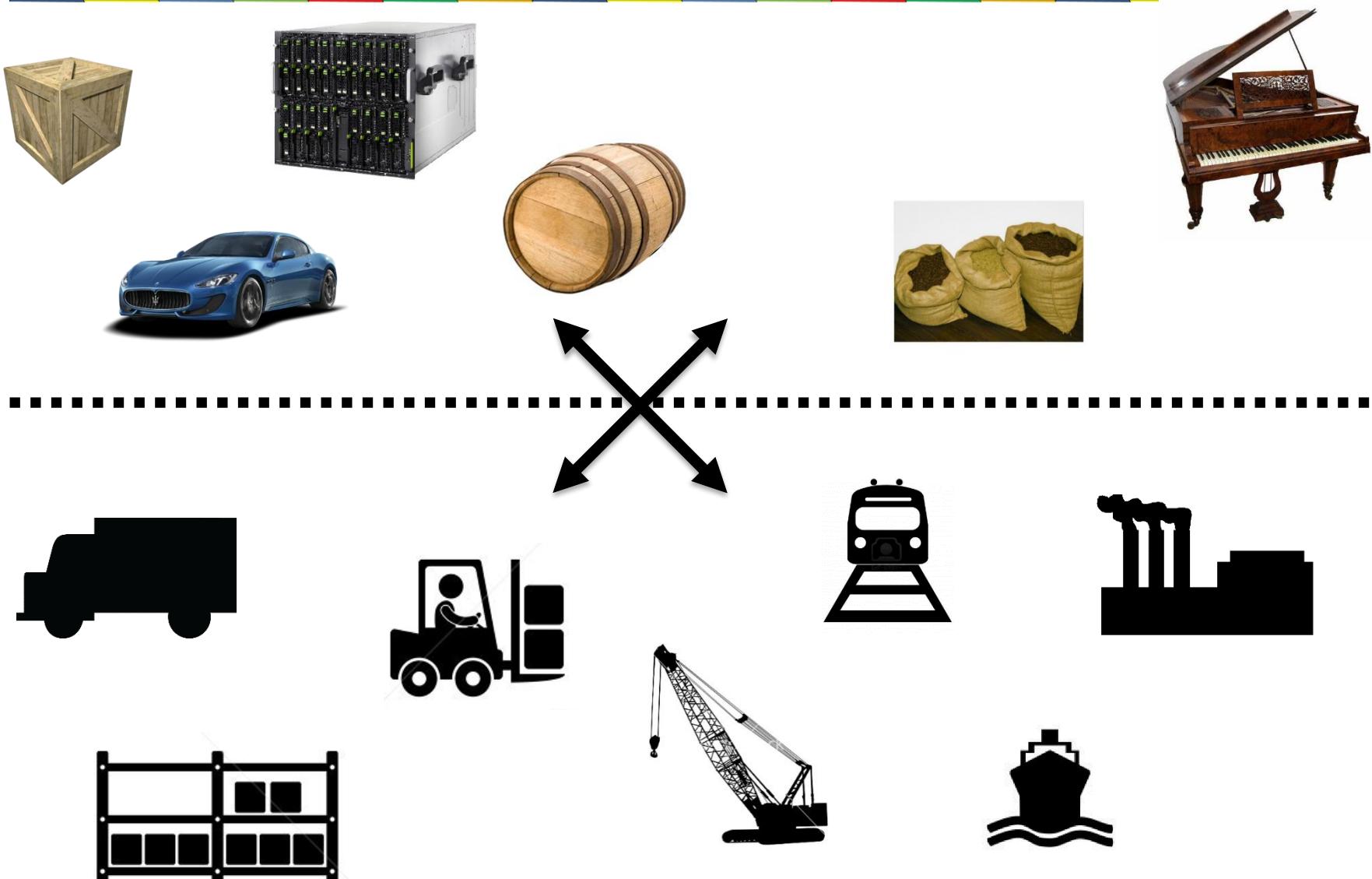
Matrix from hell



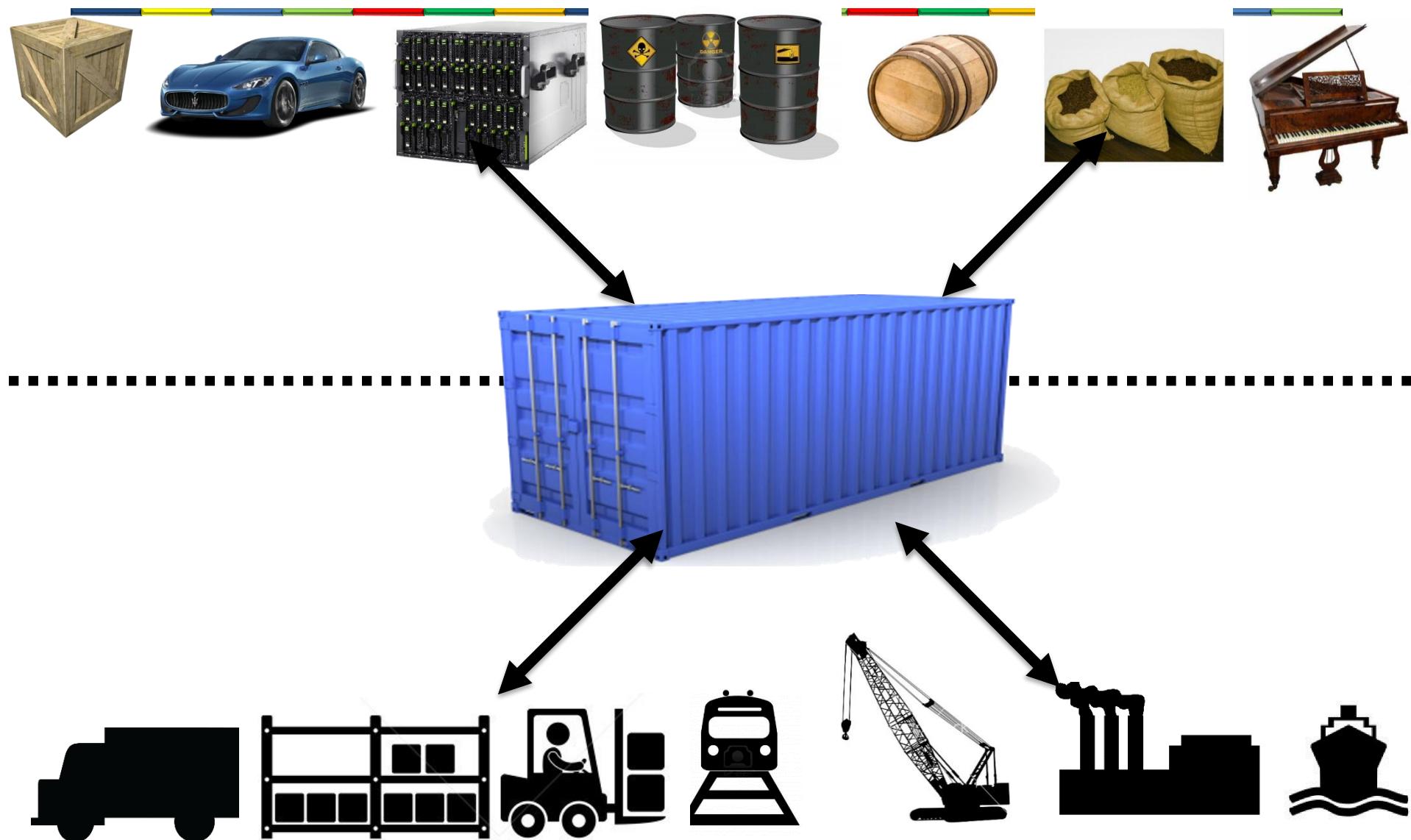
 Static website	?	?	?	?	?	?	?	?
 Web frontend	?	?	?	?	?	?	?	?
 Background workers	?	?	?	?	?	?	?	?
 User DB	?	?	?	?	?	?	?	?
 Analytics DB	?	?	?	?	?	?	?	?
 Queue	?	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers	



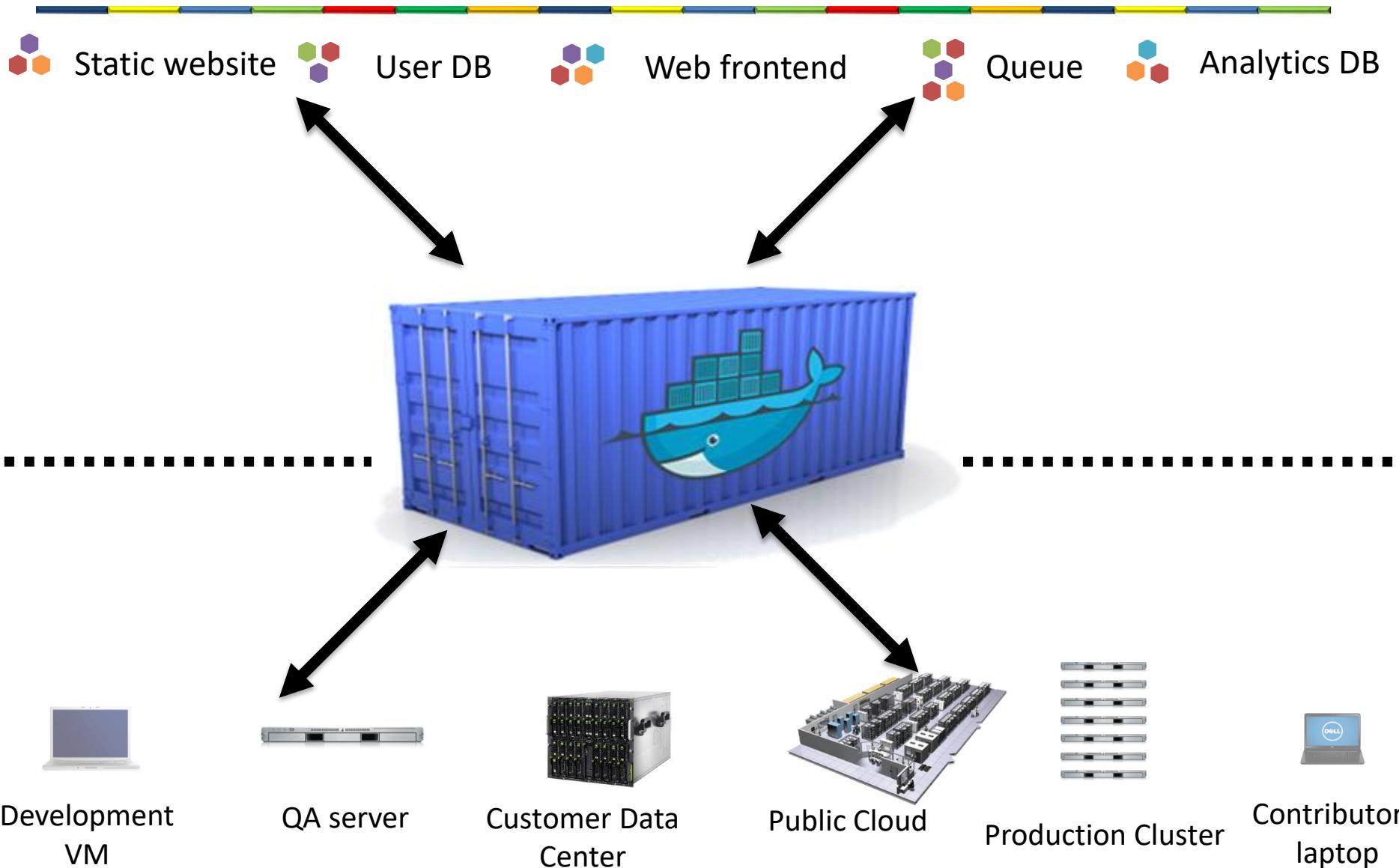
Cargo Transport Pre-1960



The solution



Docker is a shipping container for code

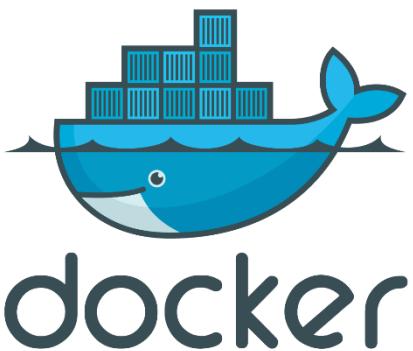


Separation of concern

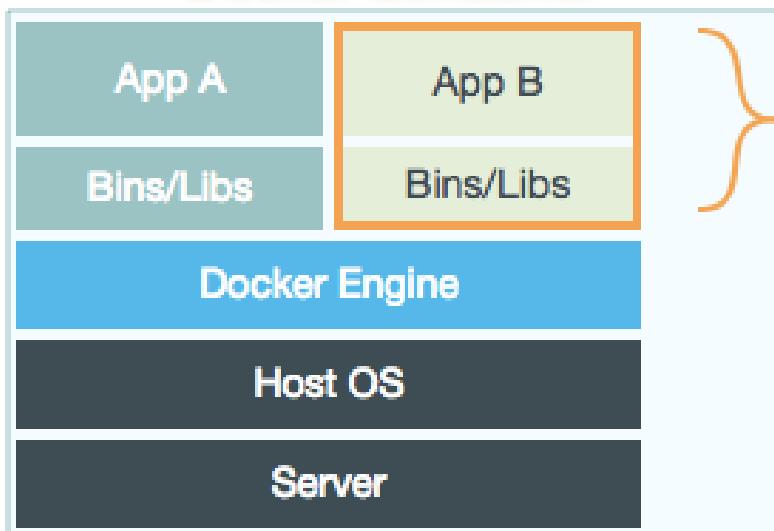
- Dan the Developer
 - Worries about what's "inside" the container
 - His code
 - His Libraries
 - His Package Manager
 - His Apps
 - His Data
- Oscar the Ops Guy
 - Worries about what's "outside" the container
 - Logging
 - Remote access
 - Monitoring
 - Network config
 - All containers start, stop, copy, attach, migrate, etc. the same way

Containers

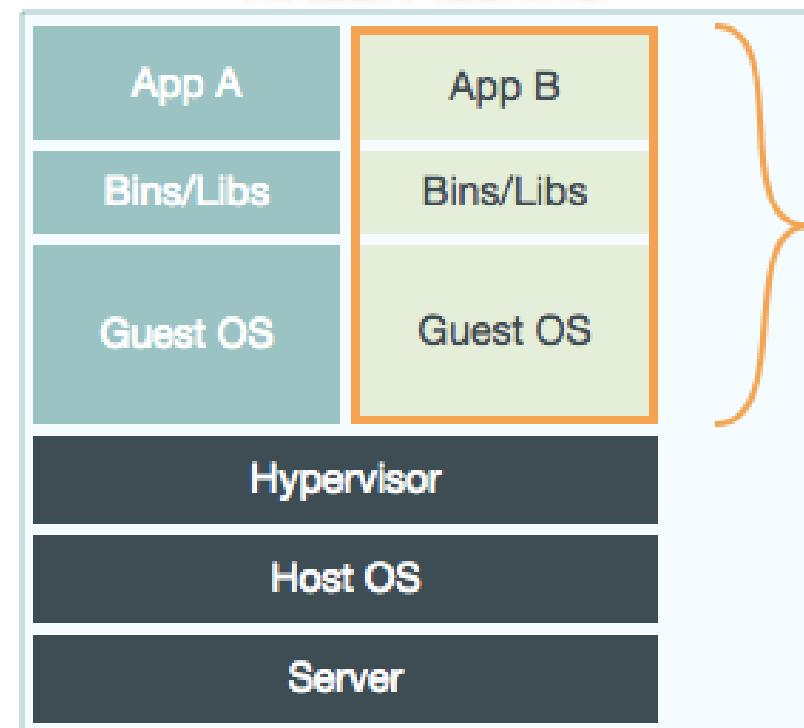
- Docker



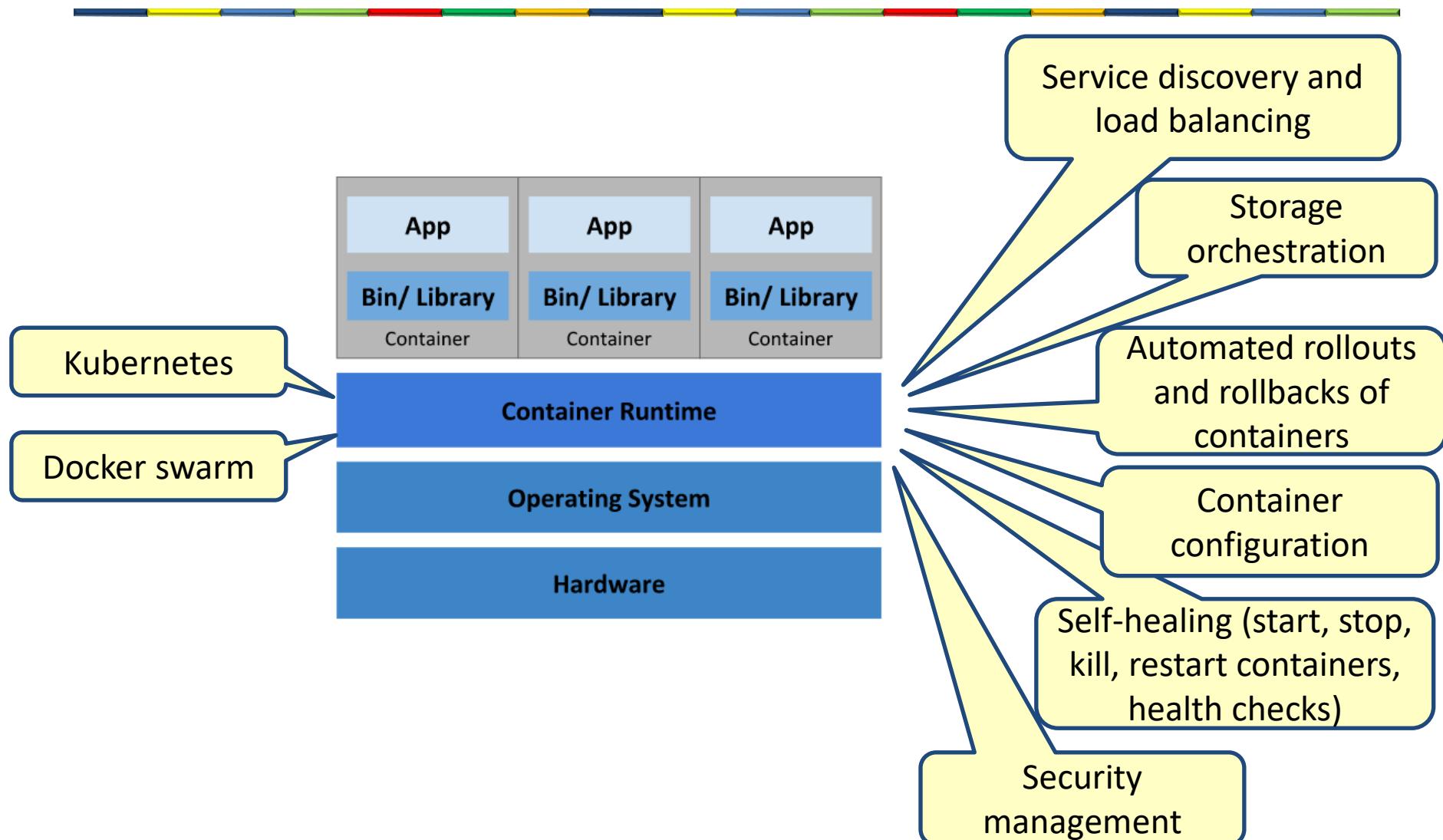
Docker Container



Virtual Machine



Container management



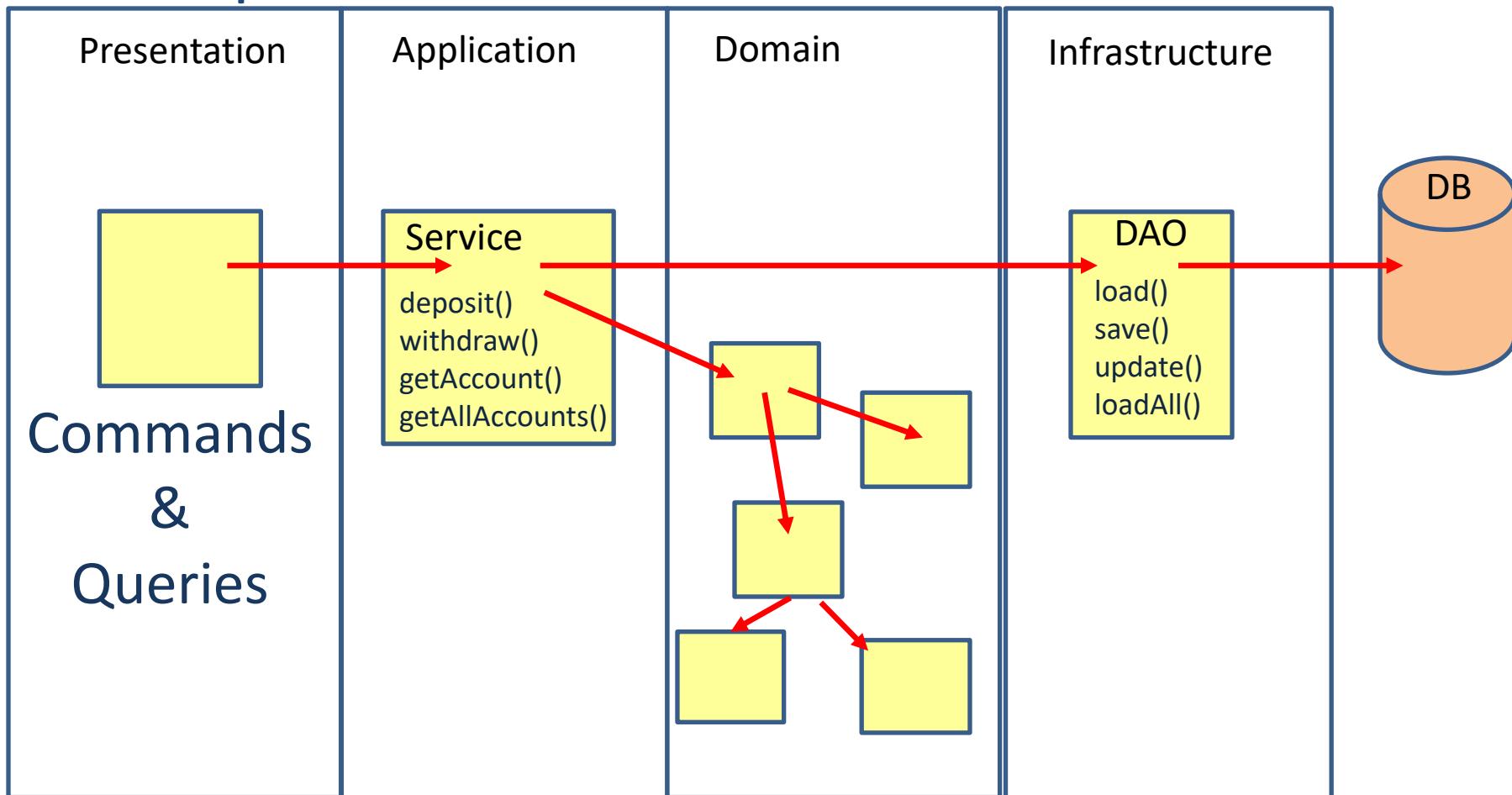
CQRS

Command Query Responsibility Segregation (CQRS)

- Separates the querying from command processing by providing two models instead of one.
 - One model is built to handle and process commands
 - One model is built for presentation needs (queries)

Typical architecture

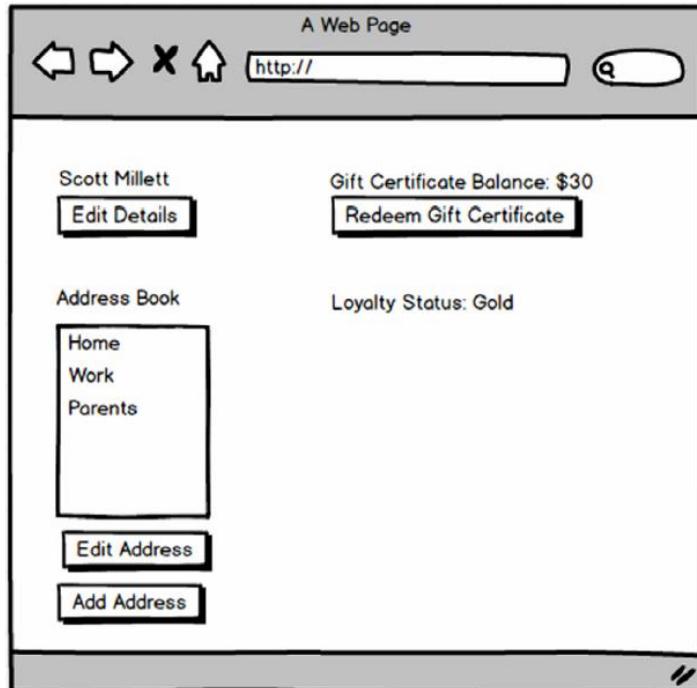
- One domain model that is used for commands and queries



One model for both commands and queries

- To support complex views and reporting
 - Required domain model becomes complex
 - Internal state needs to be exposed
 - Aggregates are merged for view requirements
 - Repositories often contain many extra methods to support presentation needs such as paging, querying, and free text searching
- Result: single model that is full of compromises

Example of complex aggregates



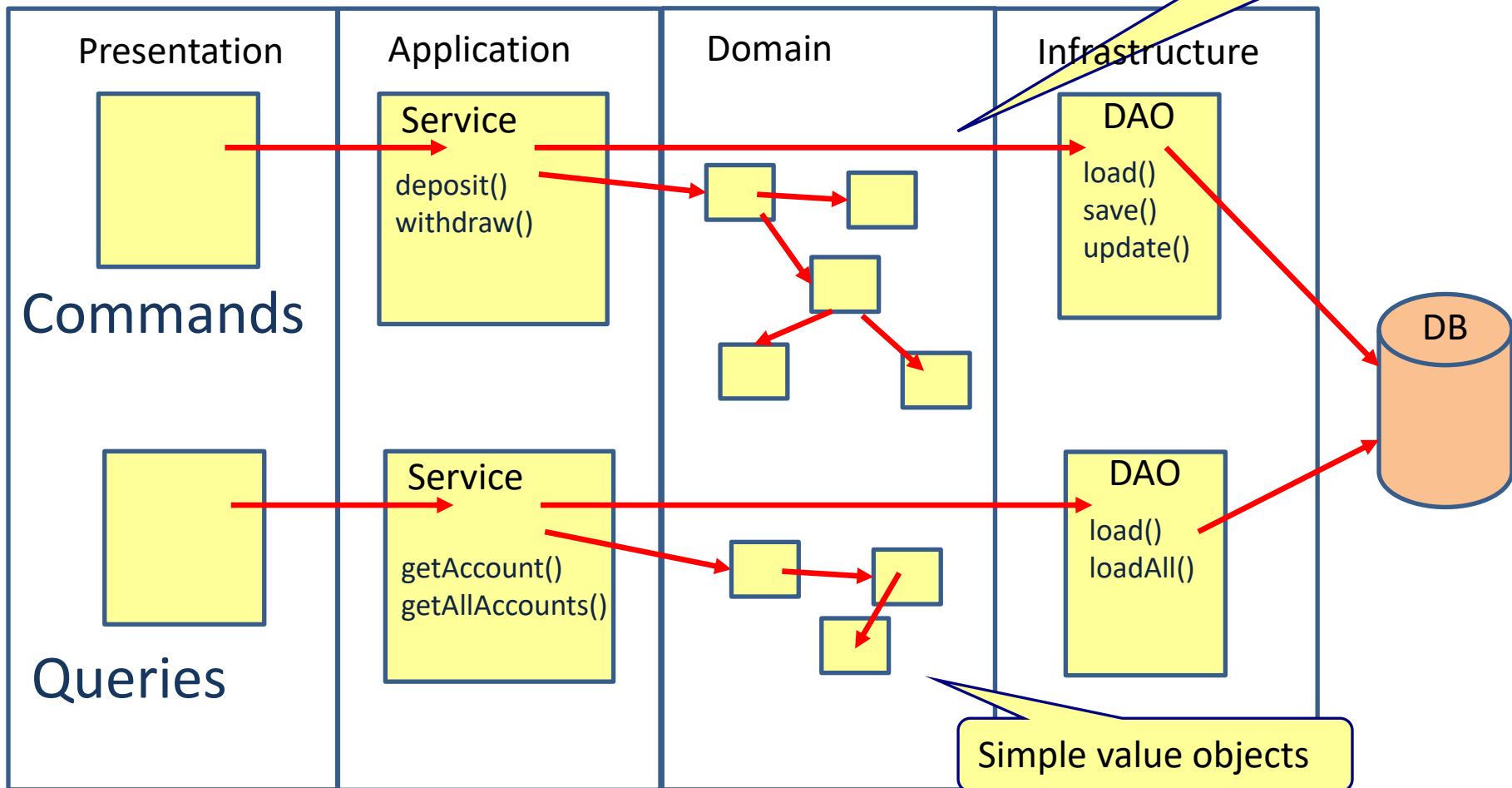
Complex aggregate
because of UI needs

```
public class Customer
{
    // ...
    public ContactDetails ContactDetails { get; private set; }
    public LoyaltyStatus LoyaltyStatus { get; private set; }
    public Money GiftCertBalance { get; private set; }
    public IEnumerable<Address> AddressBook { get; private set; }
}
```

We don't need this
complexity for commands

CQRS

- Domain model is used for commands
- View model is used for queries



2 services instead of one

Traditional service

CustomerService

```
void MakeCustomerPreferred(CustomerId)  
Customer GetCustomer(CustomerId)  
CustomerSet GetCustomersWithName(Name)  
CustomerSet GetPreferredCustomers()  
void ChangeCustomerLocale(CustomerId, NewLocale)  
void CreateCustomer(Customer)  
void EditCustomerDetails(CustomerDetails)
```



Service with CQRS

CustomerWriteService

```
void MakeCustomerPreferred(CustomerId)  
void ChangeCustomerLocale(CustomerId, NewLocale)  
void CreateCustomer(Customer)  
void EditCustomerDetails(CustomerDetails)
```

CustomerReadService

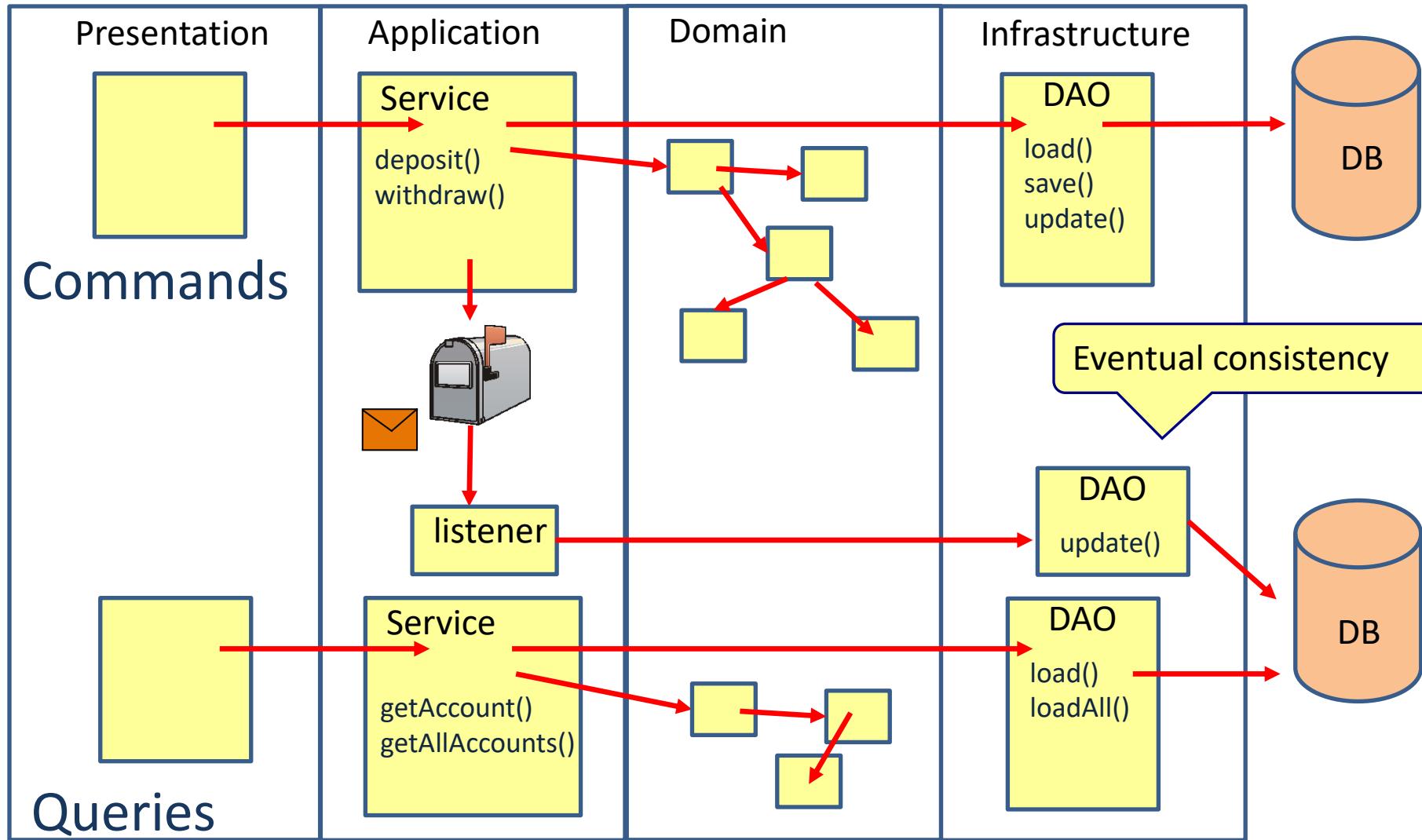
```
Customer GetCustomer(CustomerId)  
CustomerSet GetCustomersWithName(Name)  
CustomerSet GetPreferredCustomers()
```

Architectural properties

- Command and query side have different architectural properties
- Consistency
 - Command: needs consistency
 - Query: eventual consistency is mostly OK
- Data storage
 - Command: you want a normalized schema (3rd NF)
 - Query: denormalized (1st NF) is good for performance (no joins)
- Scalability
 - Command: commands happens not that often. Scalability is often not important.
 - Query: queries happen very often, scalability is important

Eventual consistency

- Views will become eventual consistent



CQRS advantages

- The query side is very simple
 - No transactions
- The query side can be optimized for performance
 - Fast noSQL database
 - Caching is easy
- Queries and commands can be scaled independently

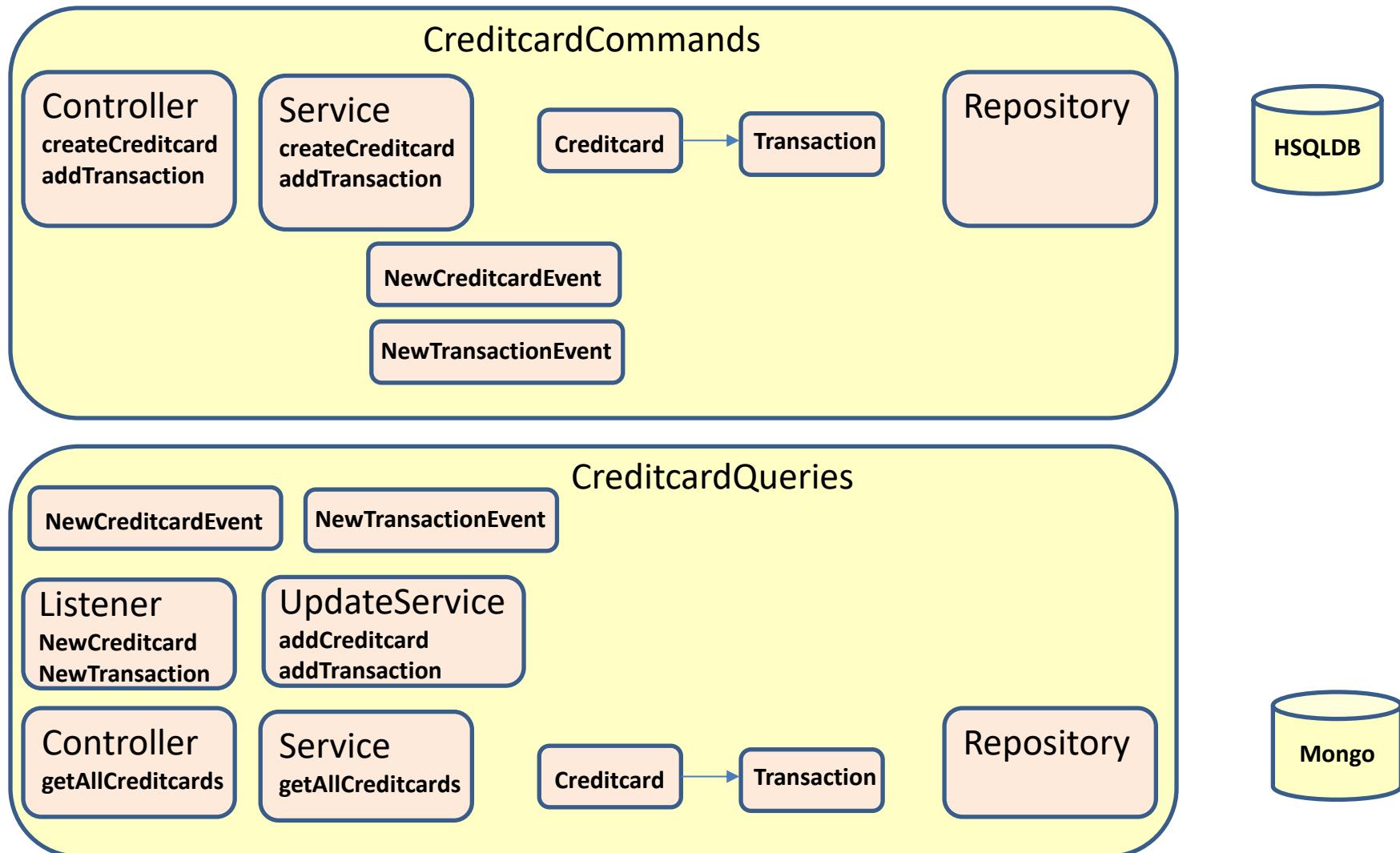
When to use CQRS?

- When queries and commands have different scaling requirements
- When read performance is critical
- When your screens start to look very different than your tables
- When you apply event sourcing

When not to use CQRS

- Systems with simple CRUD functionality
- When you need strict consistency (instead of eventual consistency)

CQRS example



Main point

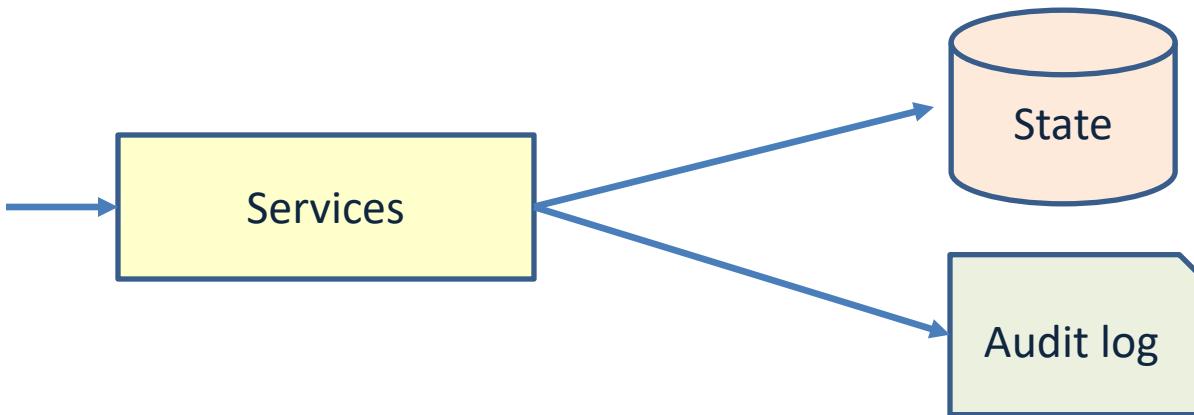
- In CQRS we separate the queries from the commands so that both can be designed, build, optimized and scaled separately.
- Every relative part of creation is connected at the level of the Unified Field.

EVENT SOURCING

State based persistance

- You capture where you are, but not how you got there
- You don't know what happened in the past
- You cannot fix bad state because of bugs in the code

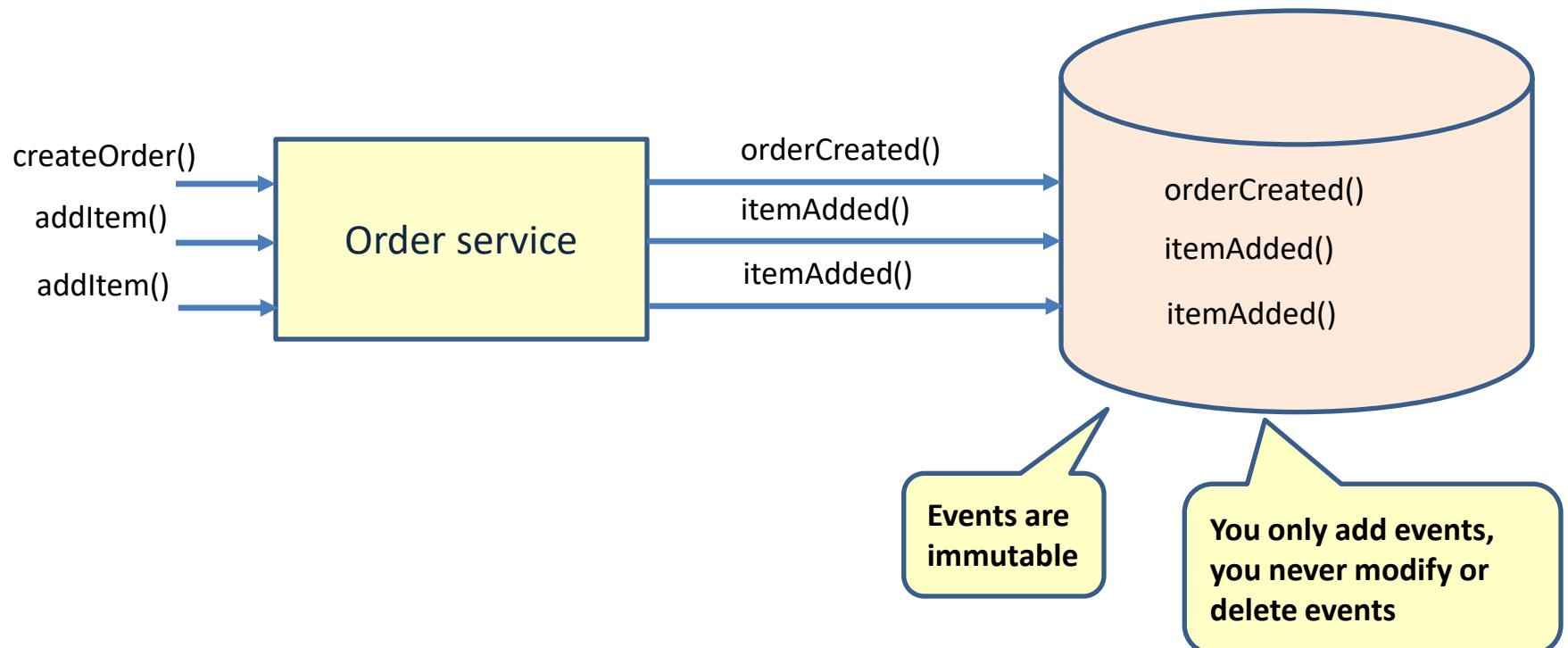
Audit log



- Now we know how we got there
- What if the audit log gets out of sync with the state due to a bug?
- Which is the source of truth? (Audit log)

Event sourcing

- Only capture how we got there



Event sourcing

- Bank account
 - Store all deposits and withdrawals
- Phone account
 - Store all phone calls
- Version control systems
 - Each commit or change is an event
- DBMS
 - Databases keep a transaction log of all inserts, deletes and updates

Storing state and storing events

- Store state

ID	status	data...
101	accepted	...

Store entity data

- Event sourcing

Entity ID	Entity type	Event ID	Event type	Event data...
101	Order	901	OrderCreated	...
101	Order	902	OrderApproved	...
101	Order	903	OrderShipped	...

Store state changing events

Store the state of a system



Structural representation

List of ordered goods

Payment information

Shipping information

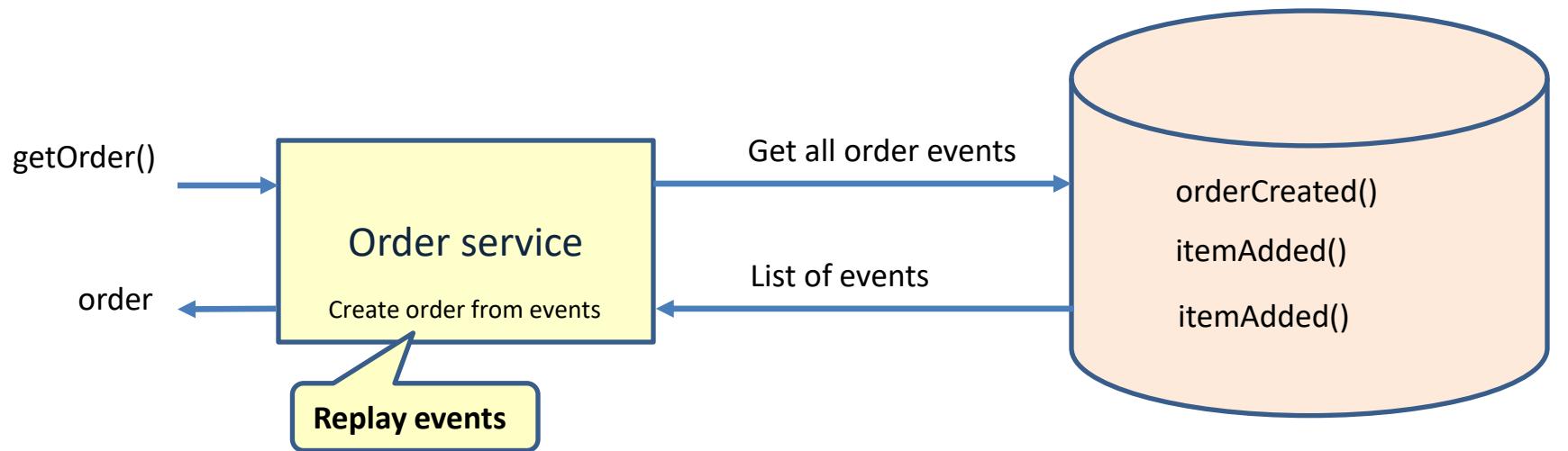
Store the events of a system



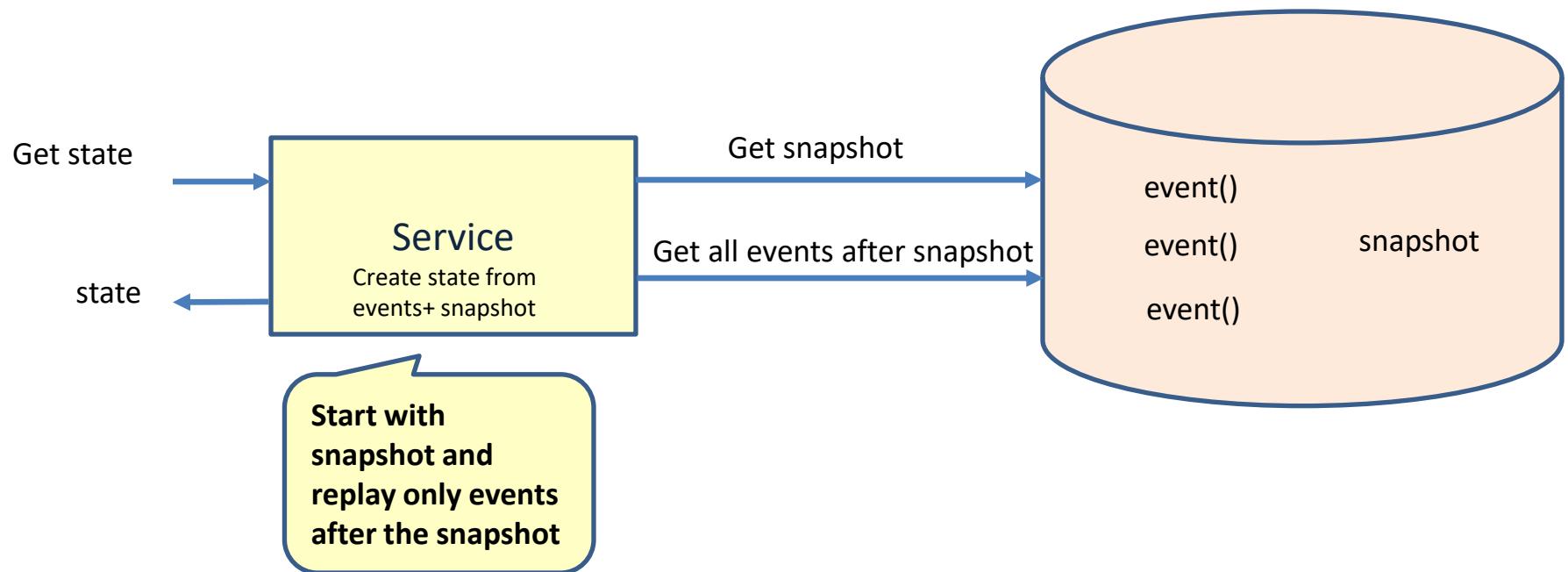
Event representation

- Add item #1
- Add item #2
- Add payment info
- Update item #2
- Remove item #1
- Add shipping info

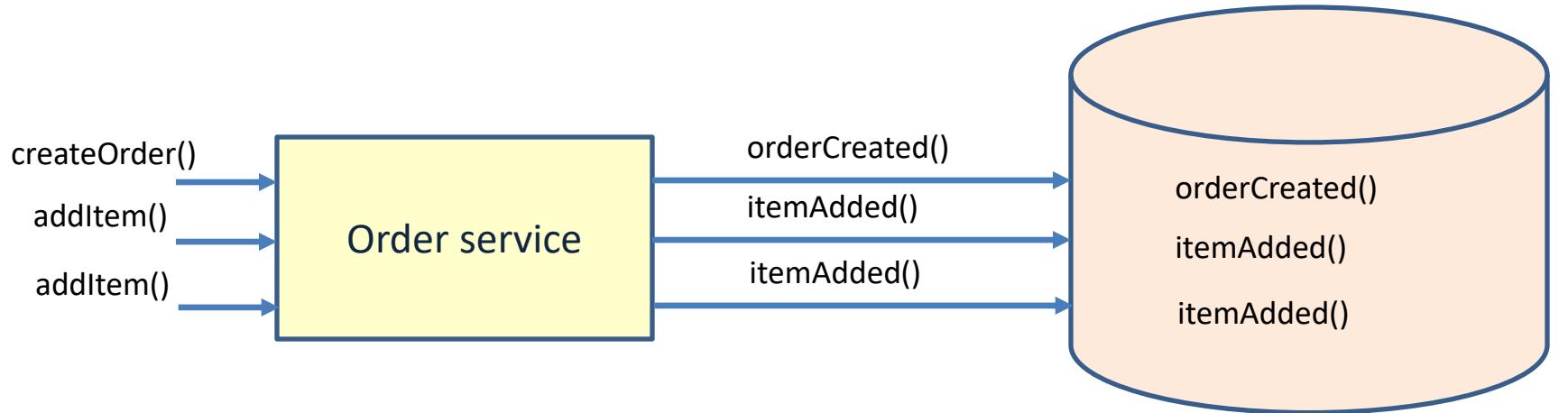
How do we get the state?



Snapshots

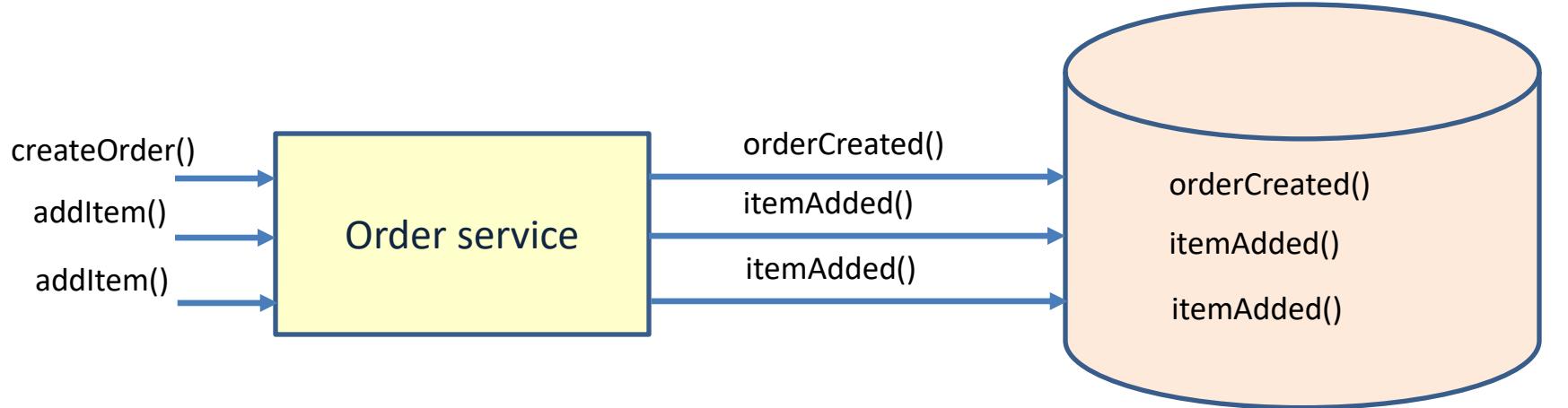


Event sourcing advantages



- You don't miss a thing
 - Business can analyze history of events
 - Bugs can be solved easier
- Creates a built-in audit log
- Allows for rewinding or undo changes
- Append is usually more efficient in databases
- No transactions needed

Event sourcing disadvantages



- You cannot easily see what your state is
 - You always have to replay the events first (lower read performance)
 - Always use CQRS when you apply event sourcing
- You need more storage
- What if events have to change?
 - We need event versioning
 - The logic needs to support all versions
- Queries over multiple event logs can be complex
 - Use CQRS

When to consider event sourcing?

- When you need an audit log
- When you want to derive additional business value from the event history
- When you need fast performance on the command side

Main point

- In event sourcing we store the events instead of the current state.
- Pure consciousness is the field of all possibilities where one has access to all intelligence of creation.

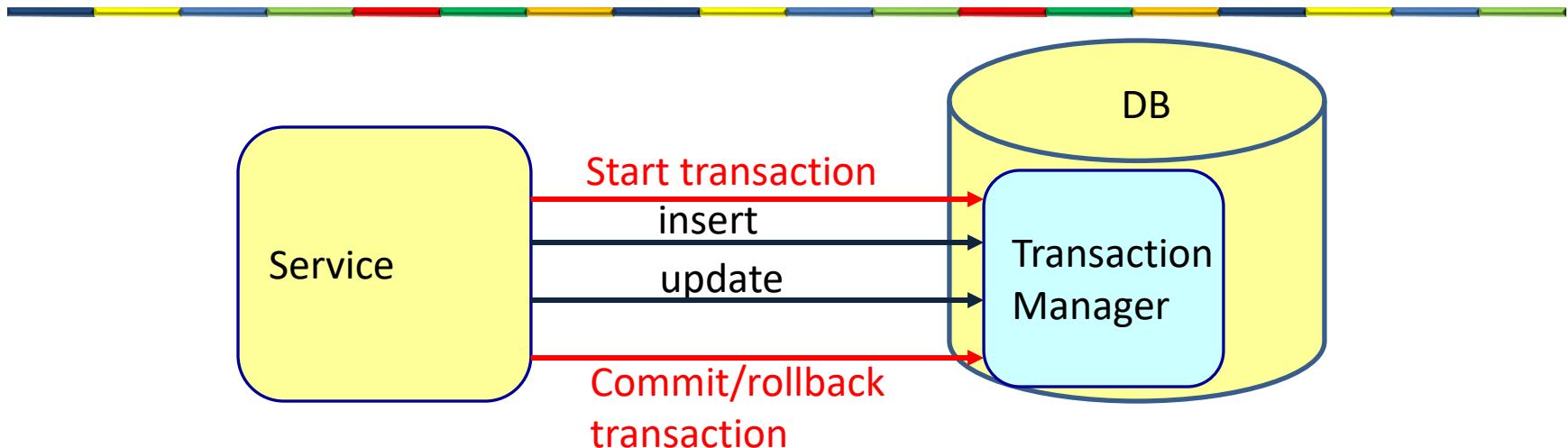
TRANSACTIONS

Transactions

- A Transaction is a unit of work that is:
 - **ATOMIC:** The transaction is considered a single unit, either the entire transaction completes, or the entire transaction fails.
 - **CONSISTENT:** A transaction transforms the database from one consistent state to another consistent state
 - **ISOLATED:** Data inside a transaction can not be changed by another concurrent processes until the transaction has been committed
 - **DURABLE:** Once committed, the changes made by a transaction are persistent

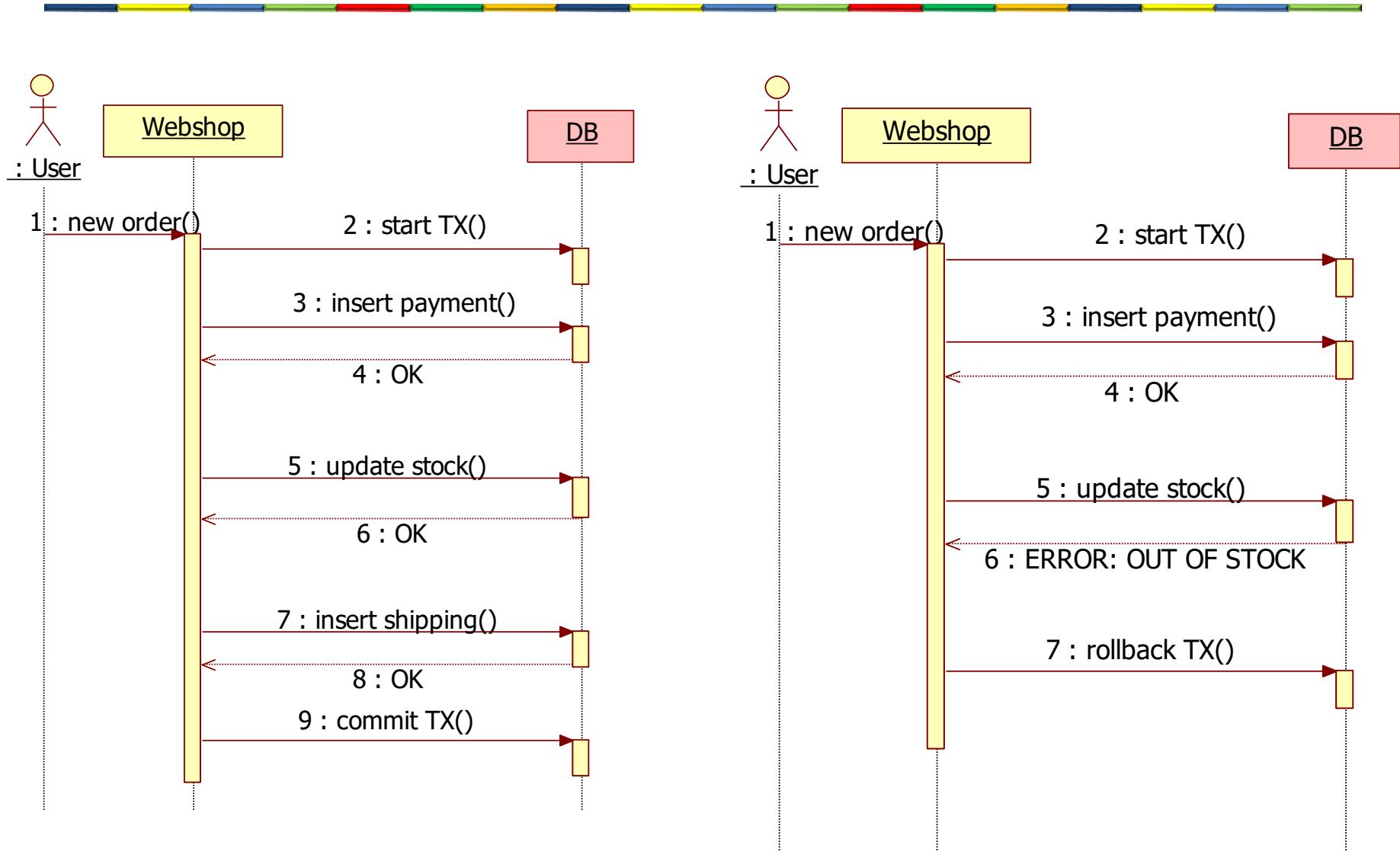


Local transaction

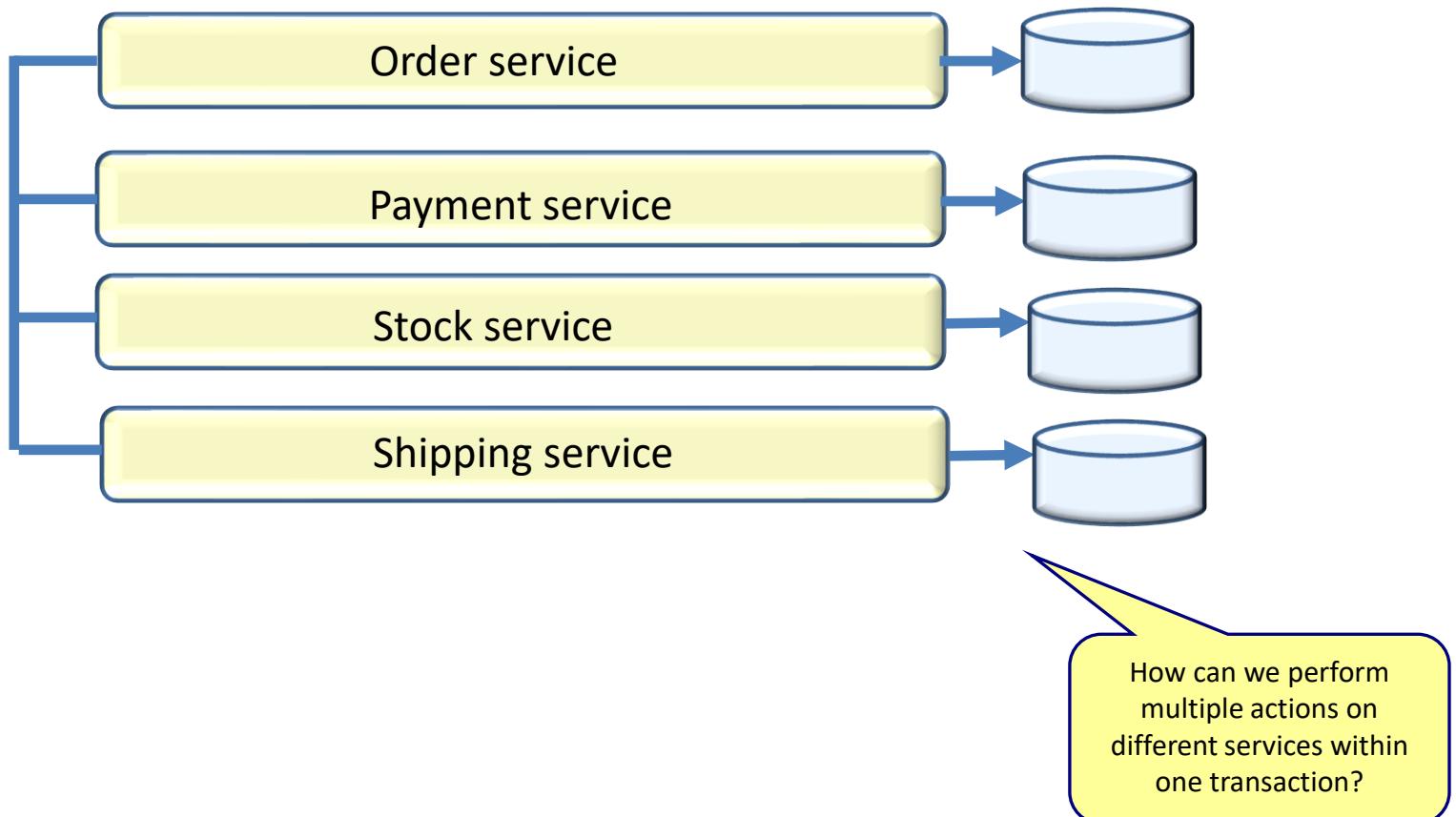


- The transaction is managed by the database
 - Simple
 - Fast
- Always try to keep transaction boundaries within a service

Local transaction

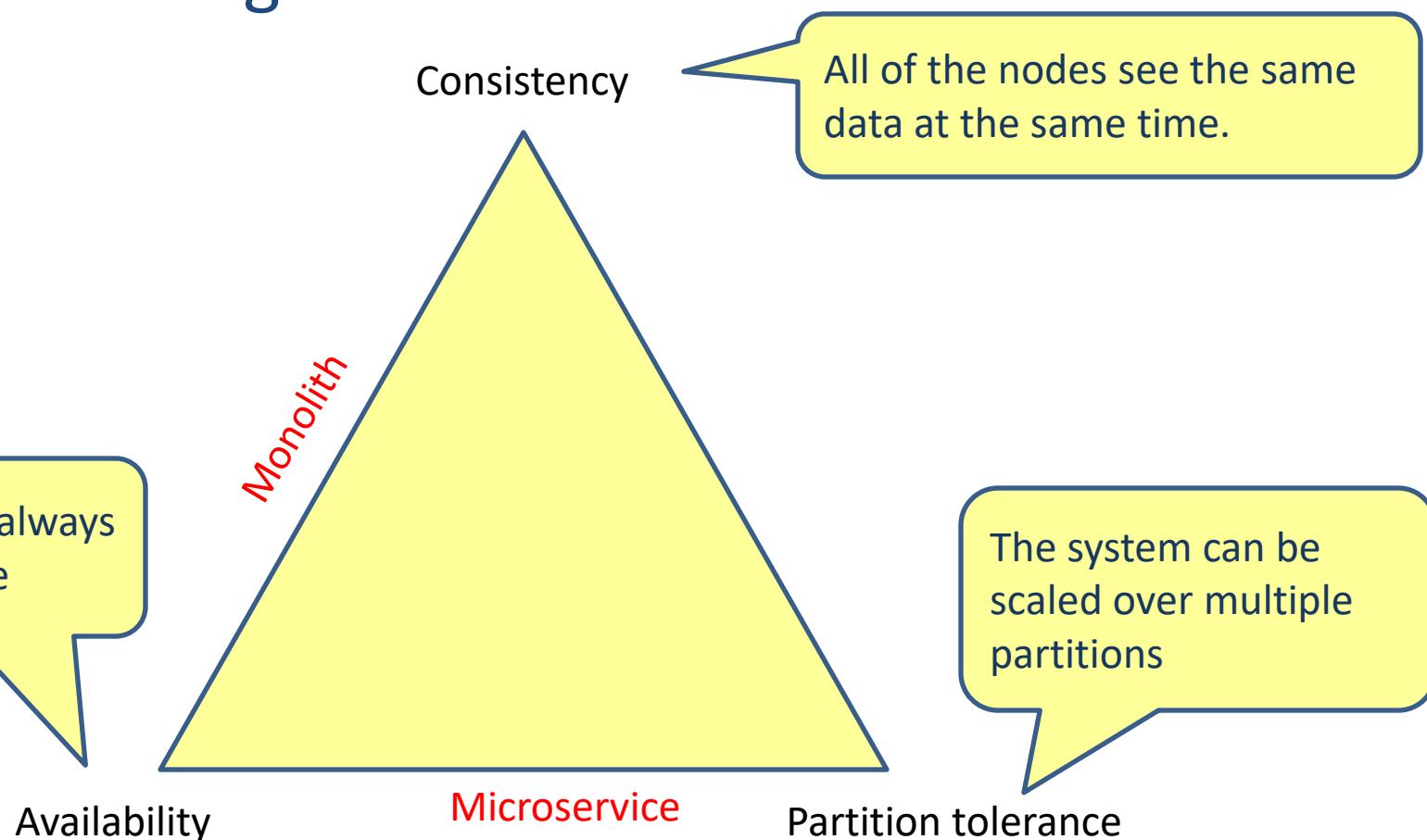


Distributed transactions

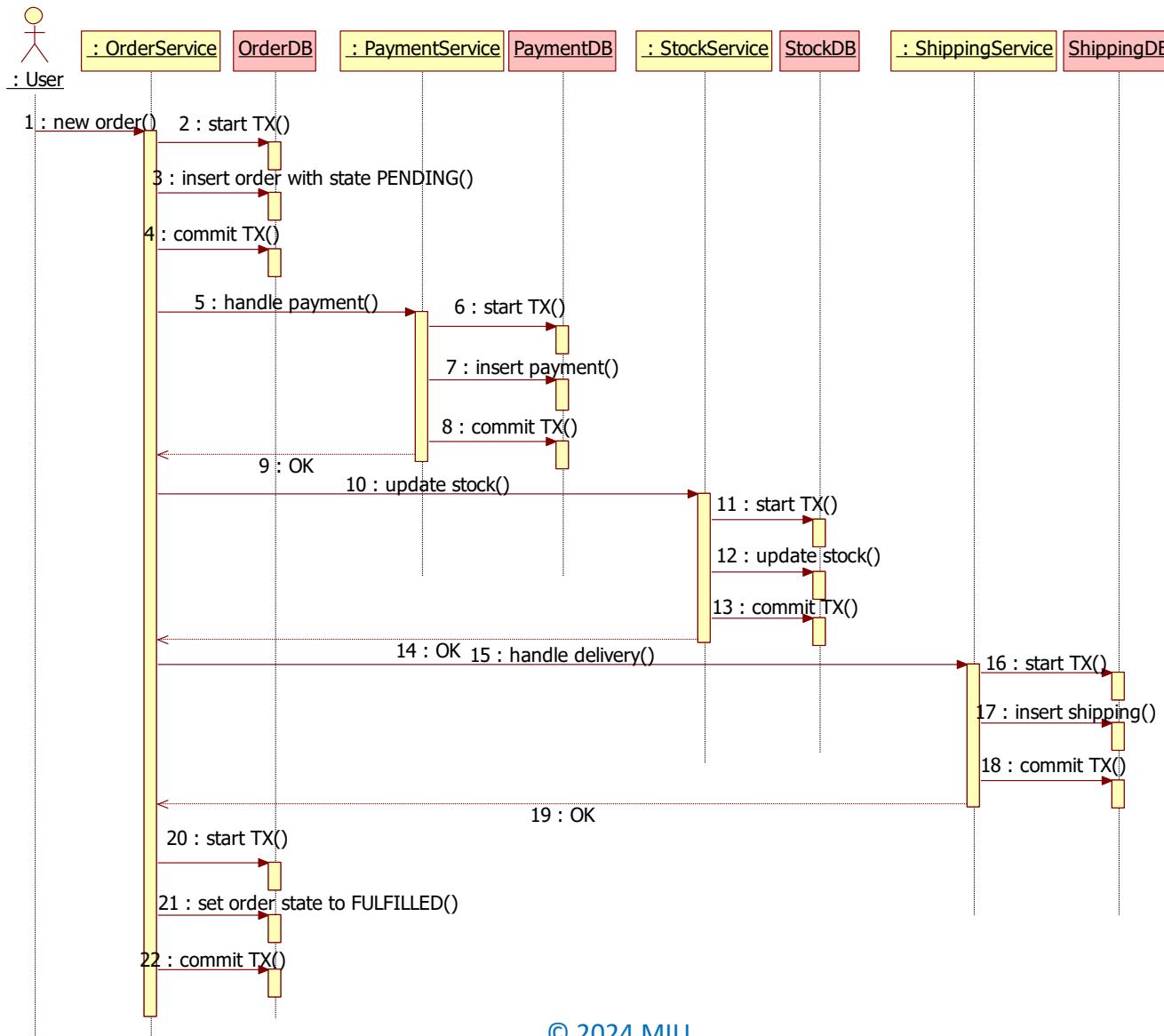


Brewer's CAP Theorem

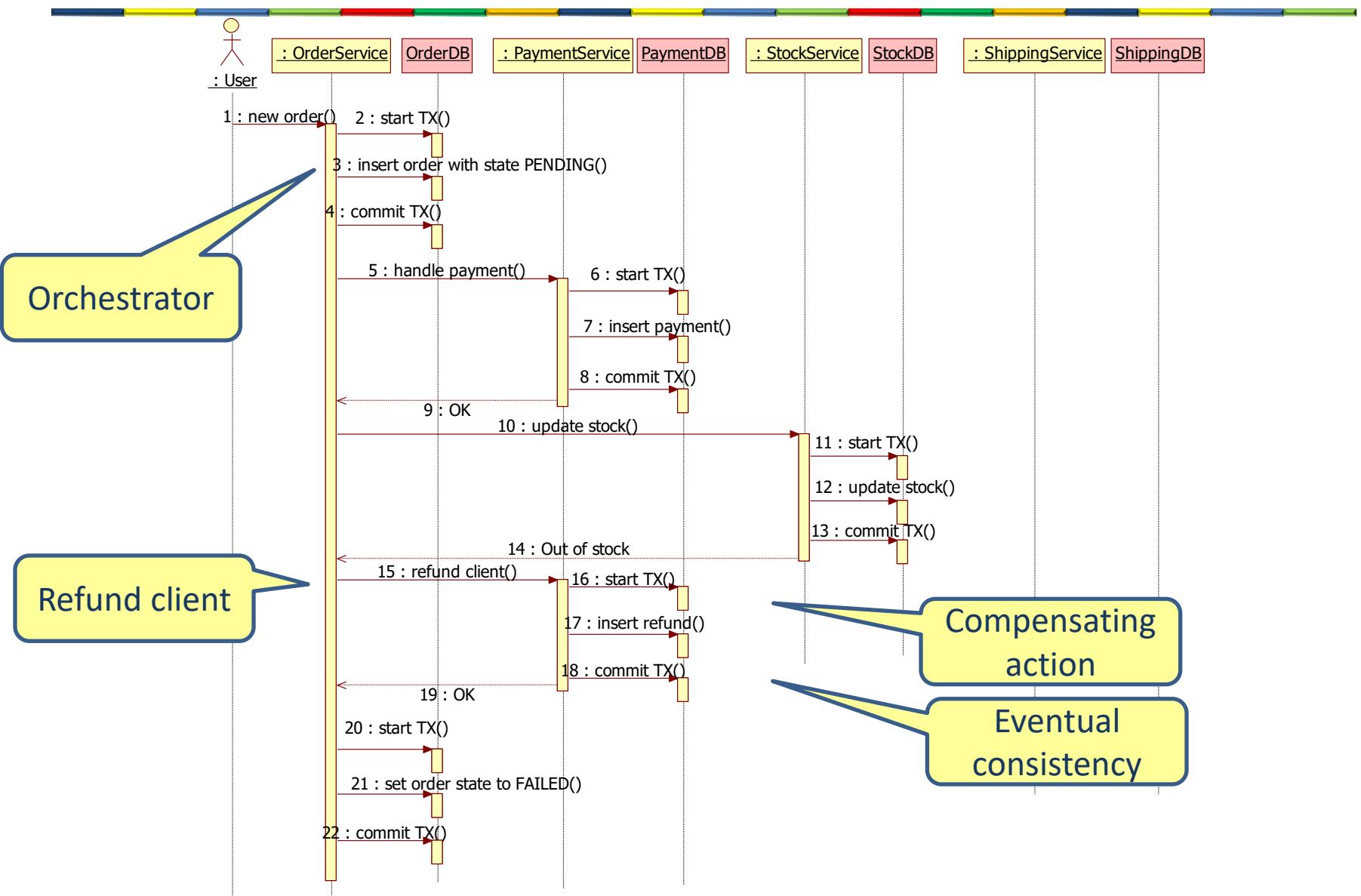
- A distributed system can support only two of the following characteristics



Distributed transaction (SAGA)

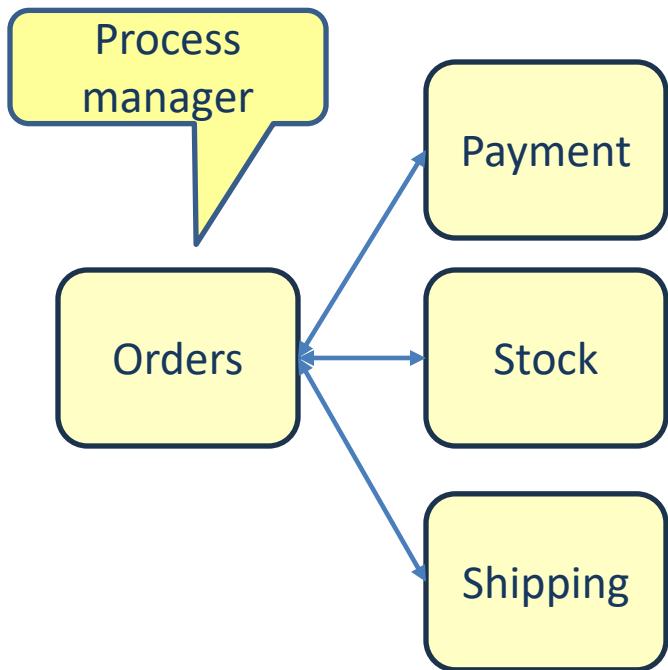


Distributed transaction (SAGA)

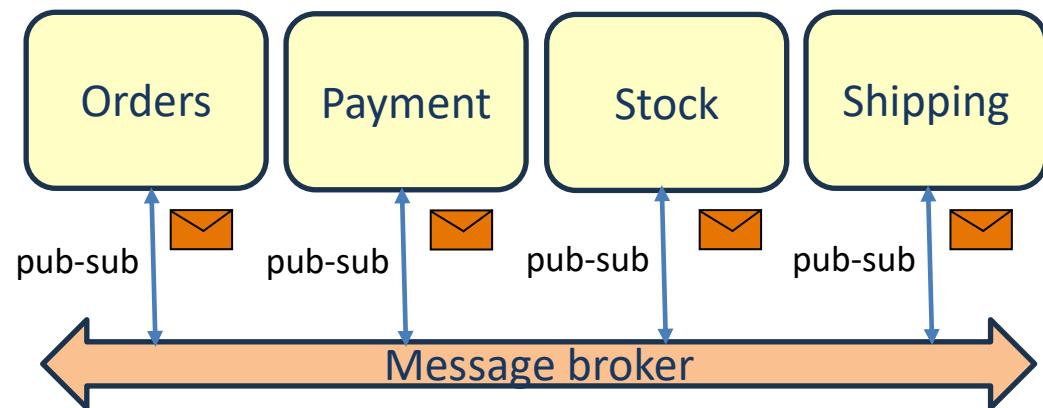


Orchestration & choreography

- Orchestration



- Choreography



Challenges of a microservice architecture

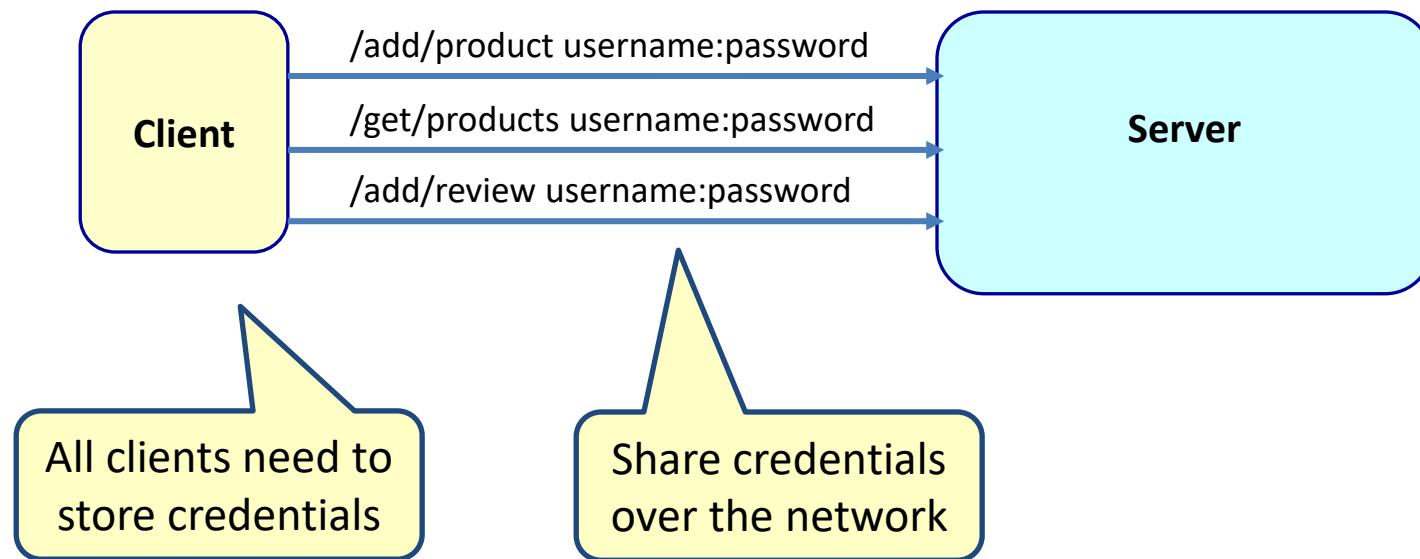
Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	
Transactions	Compensating transactions Eventual consistency
Keep data in sync	Publish-subscribe data change event
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	ELK + beats
Follow/monitor business processes	Zipkin ELK

Lesson 11

MICROSERVICES SECURITY

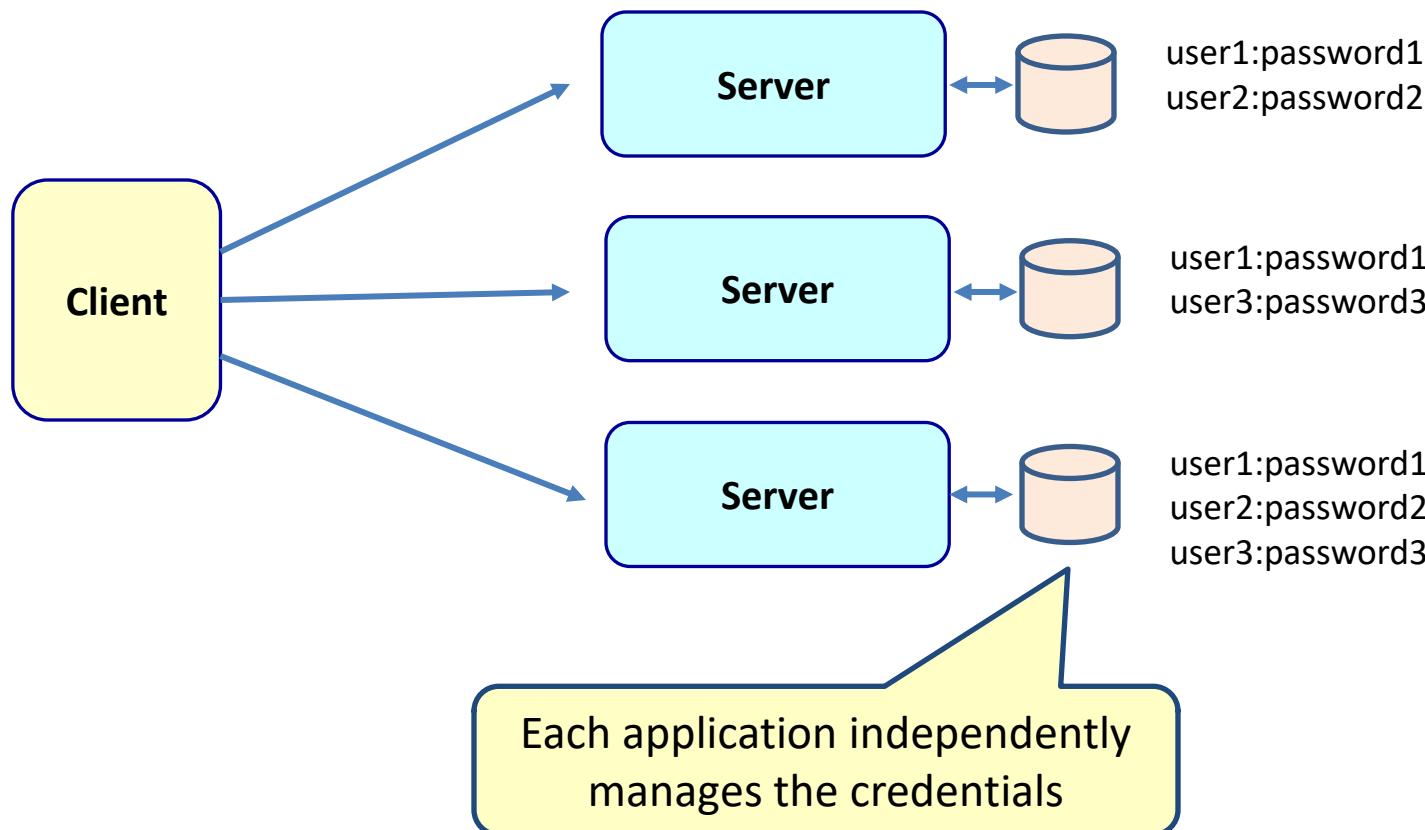
Problems with HTTP basic authentication

- We have to send credentials for every request



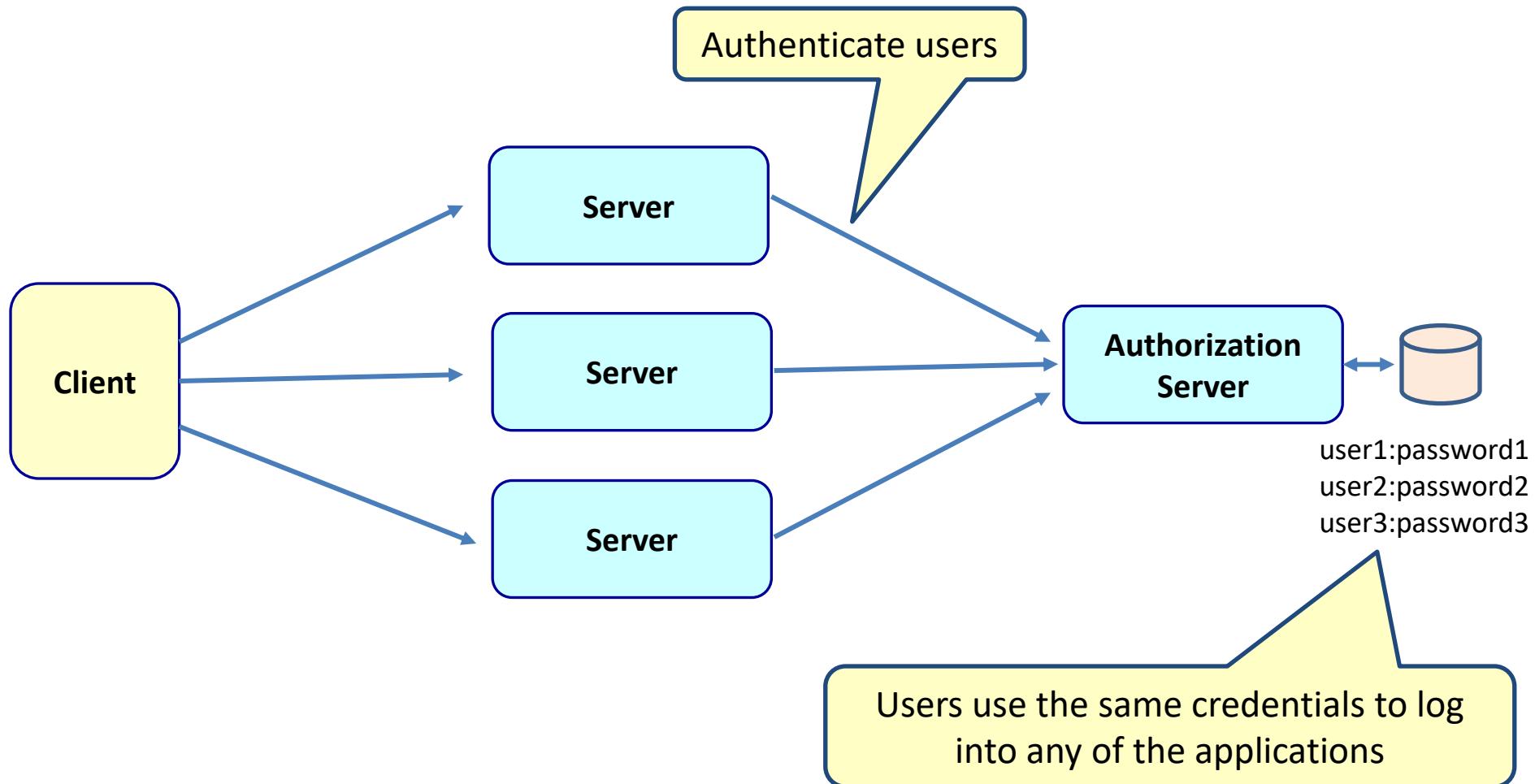
Problems with HTTP basic authentication

- Every system needs to manage these credentials

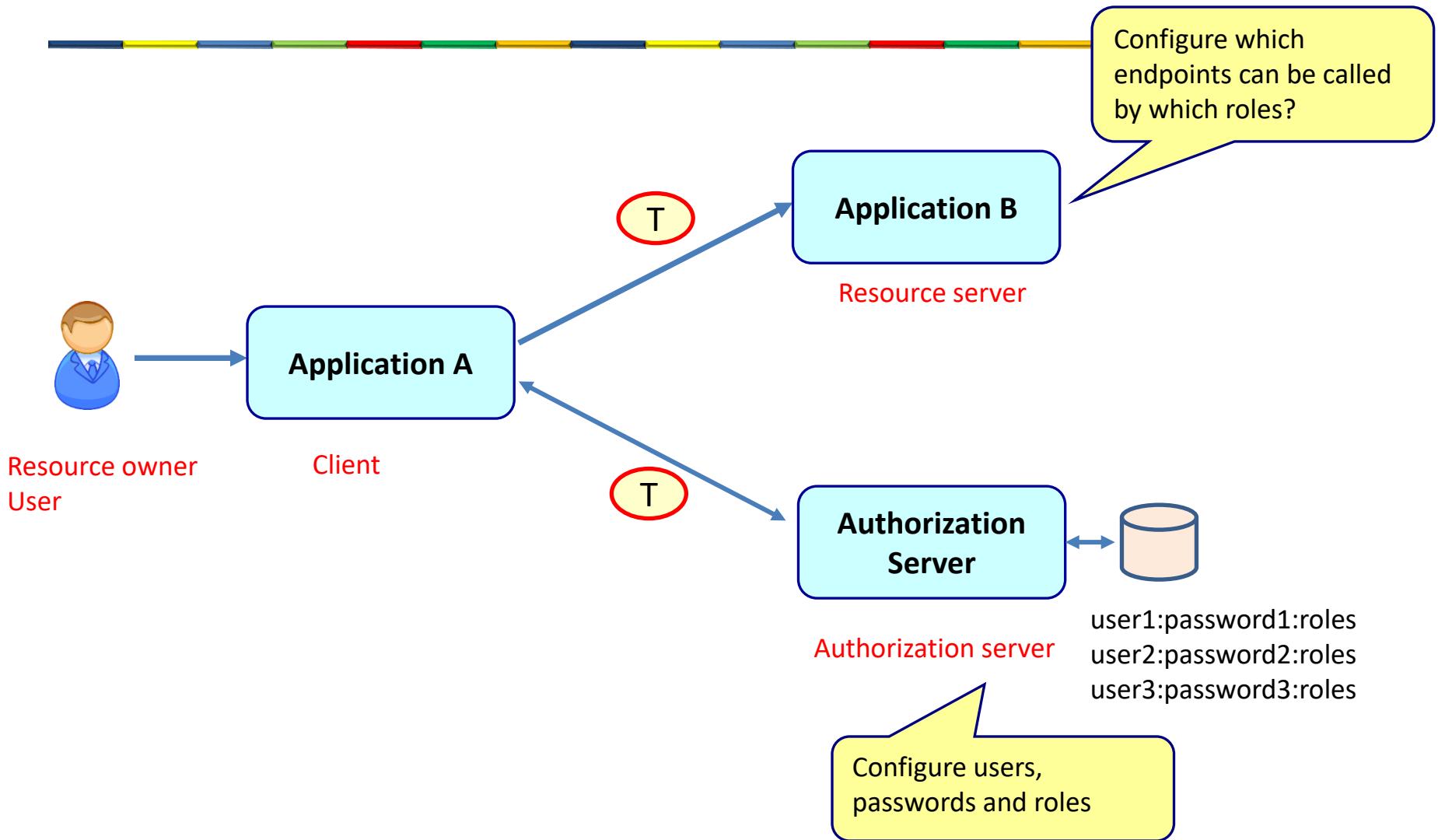


Better solution

- Authorization server



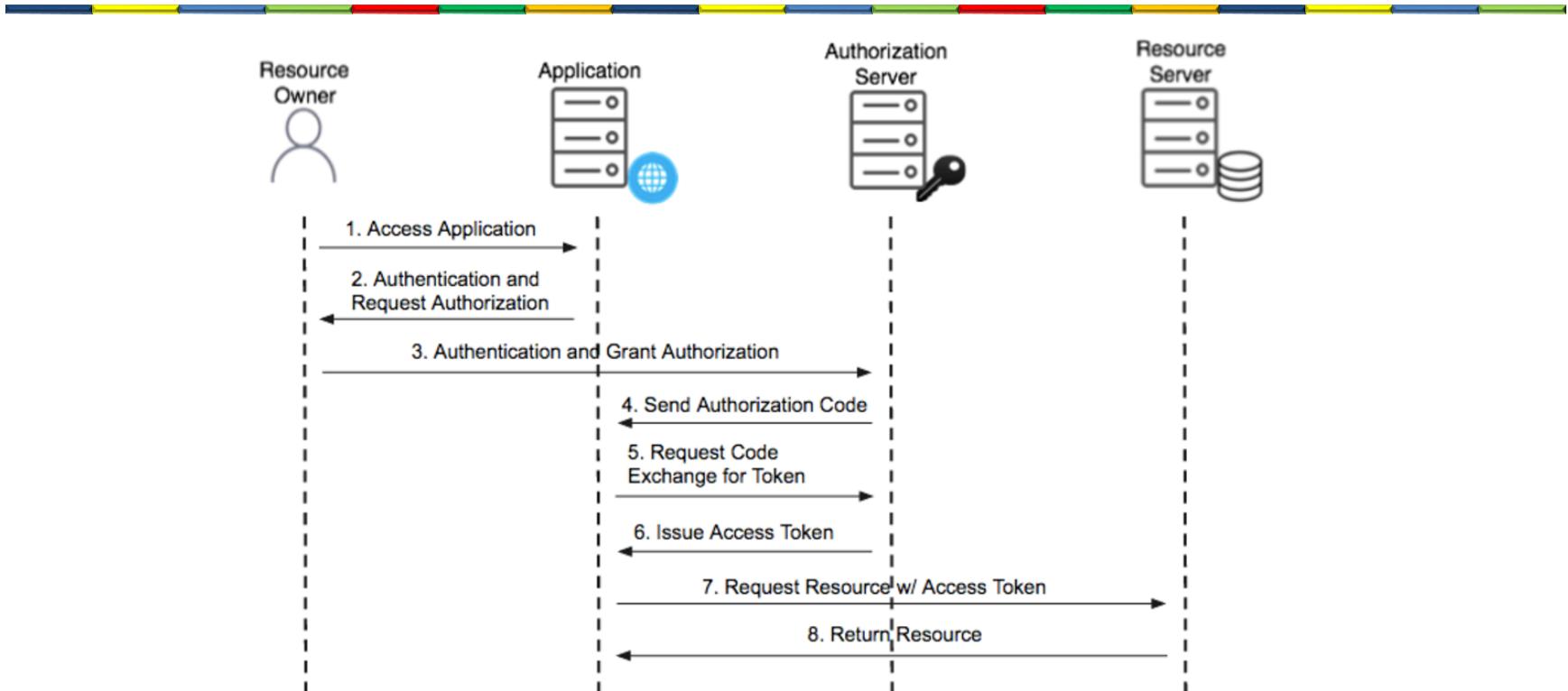
Parts of OAuth 2



OAuth 2 grants

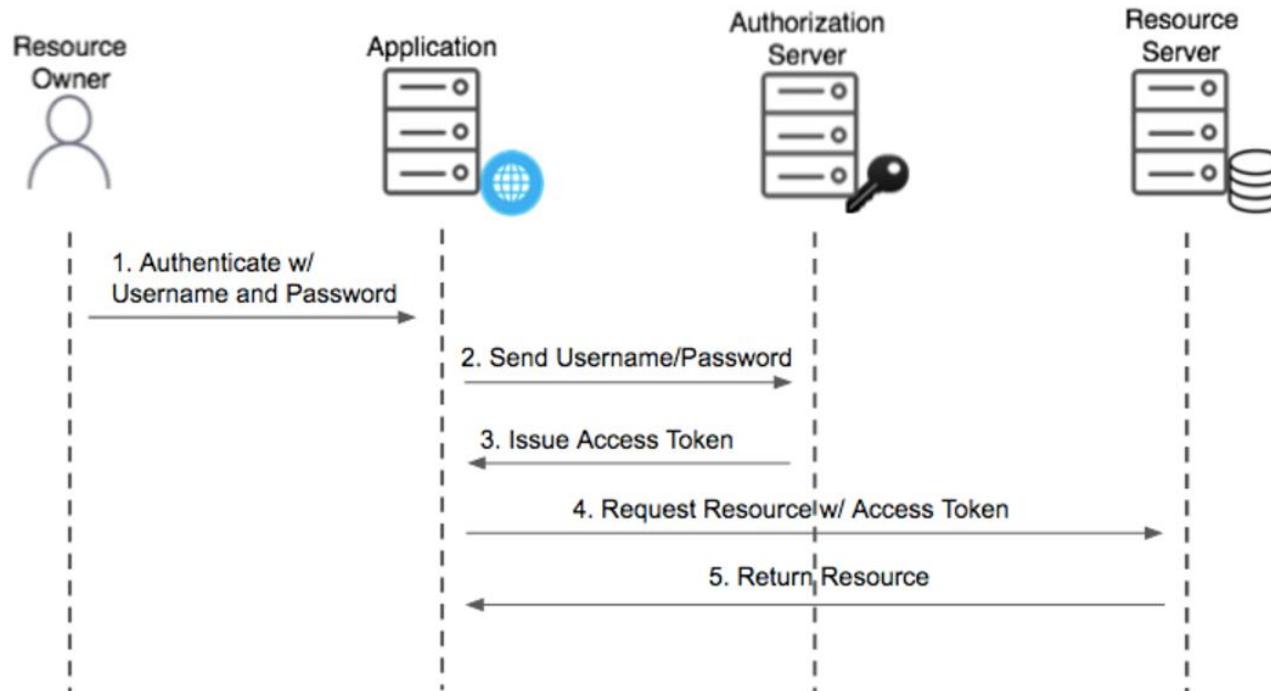
- Authorization code
- Password
- Client credentials
- Refresh token

Authorization code grant



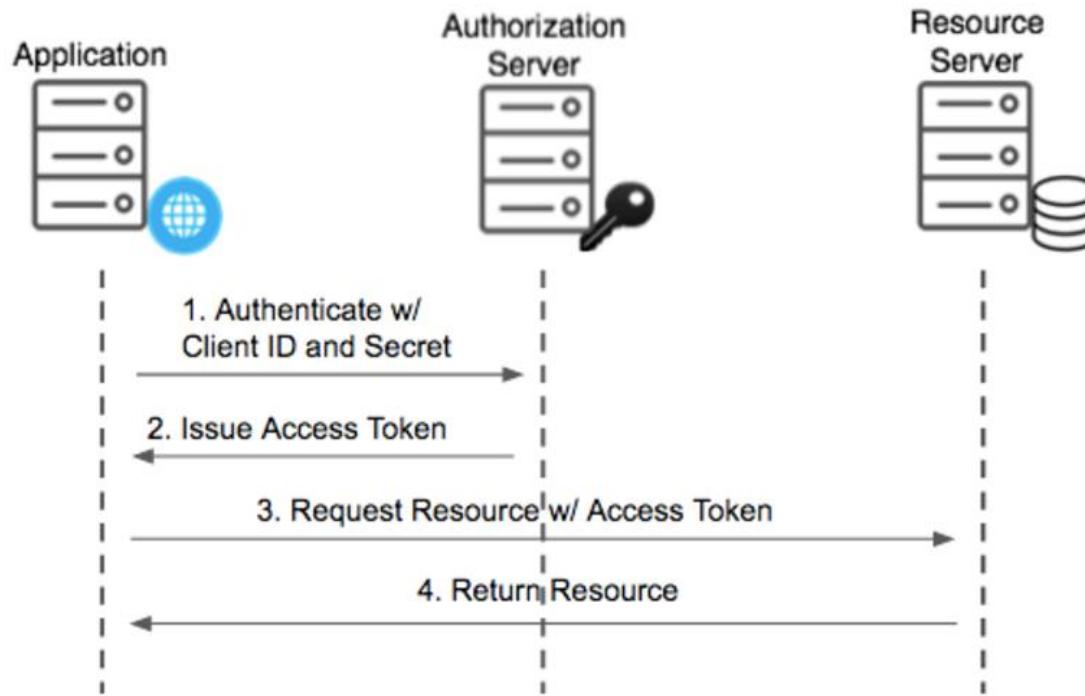
- This flow is widely used for its robust security.
- **Pros:** High security as authorization codes and access tokens are separate; ideal for server-side applications that can securely store client secrets.
- **Cons:** Involves multiple redirects between the client, authorization server, and resource server

Password grant



- This method directly uses user credentials (username and password).
- **Pros:** Simple implementation, suitable for trusted first-party applications.
- **Cons:** Lower security due to direct exposure of user credentials; not recommended for public clients.

Client credentials grant



- This grant is specifically for scenarios where applications act on their behalf rather than on behalf of a user.
- **Pros:** Effective for machine-to-machine interactions that do not require user involvement.
- **Cons:** Access permissions are granted to the application itself, requiring careful consideration of the permissions allowed.

Token

- Tokens can expire
 - Avoid tokens that don't expire
- Why refresh token?
 - User does not have to login again after token is expired
 - Client does not need to store user credentials

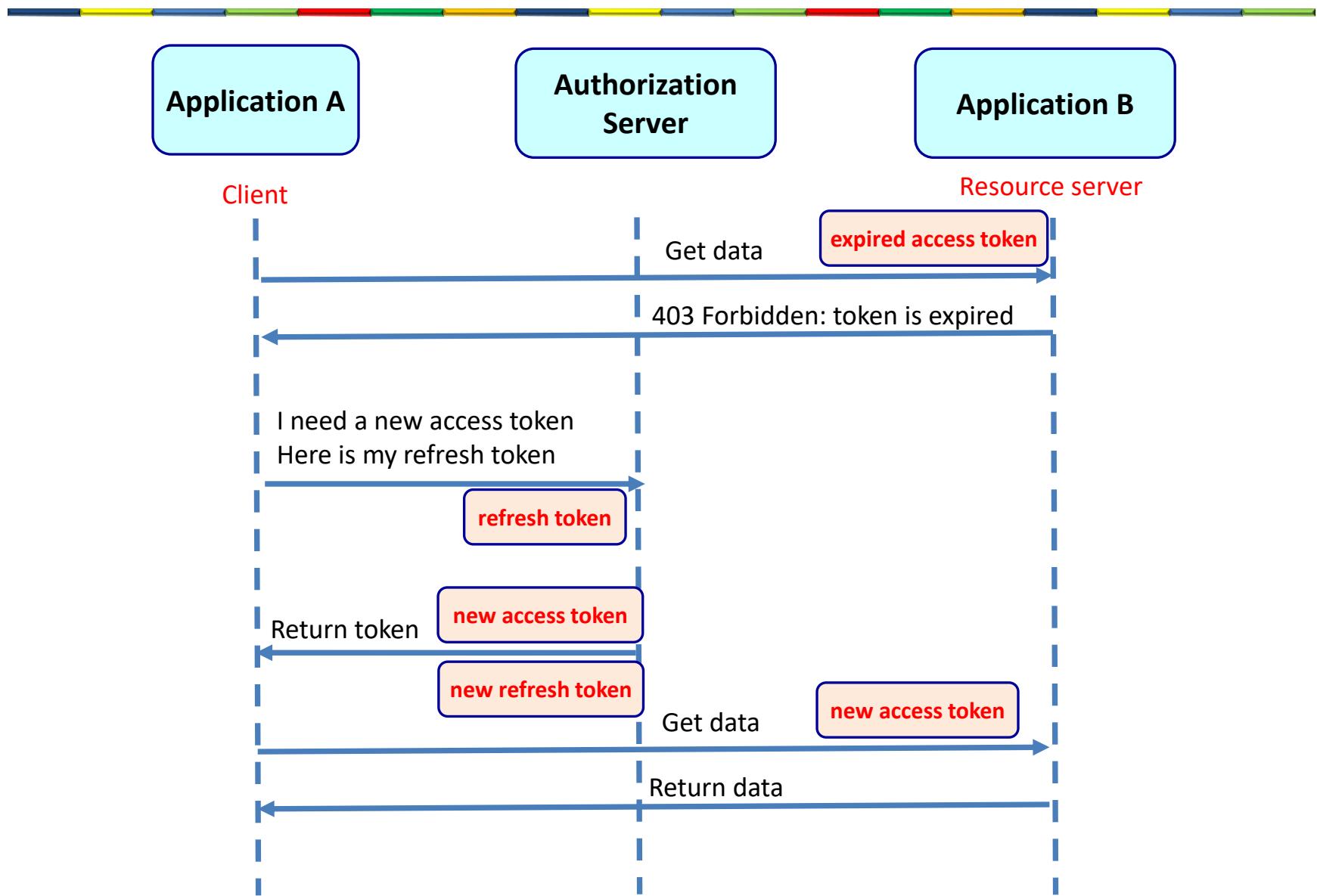
The screenshot shows a REST API tool interface with the following details:

- Headers (8)**: The Headers tab is selected.
- JSON**: The response is displayed in JSON format.
- Body**: The JSON response body is shown:

```
1 {  
2   "access_token": "b56f8dc4-4a19-411c-86f9-96c1c502f9a7",  
3   "token_type": "bearer",  
4   "refresh_token": "9618262c-0a5a-458d-afce-c45a544b5aad",  
5   "expires_in": 43199,  
6   "scope": "webclient"  
7 }
```

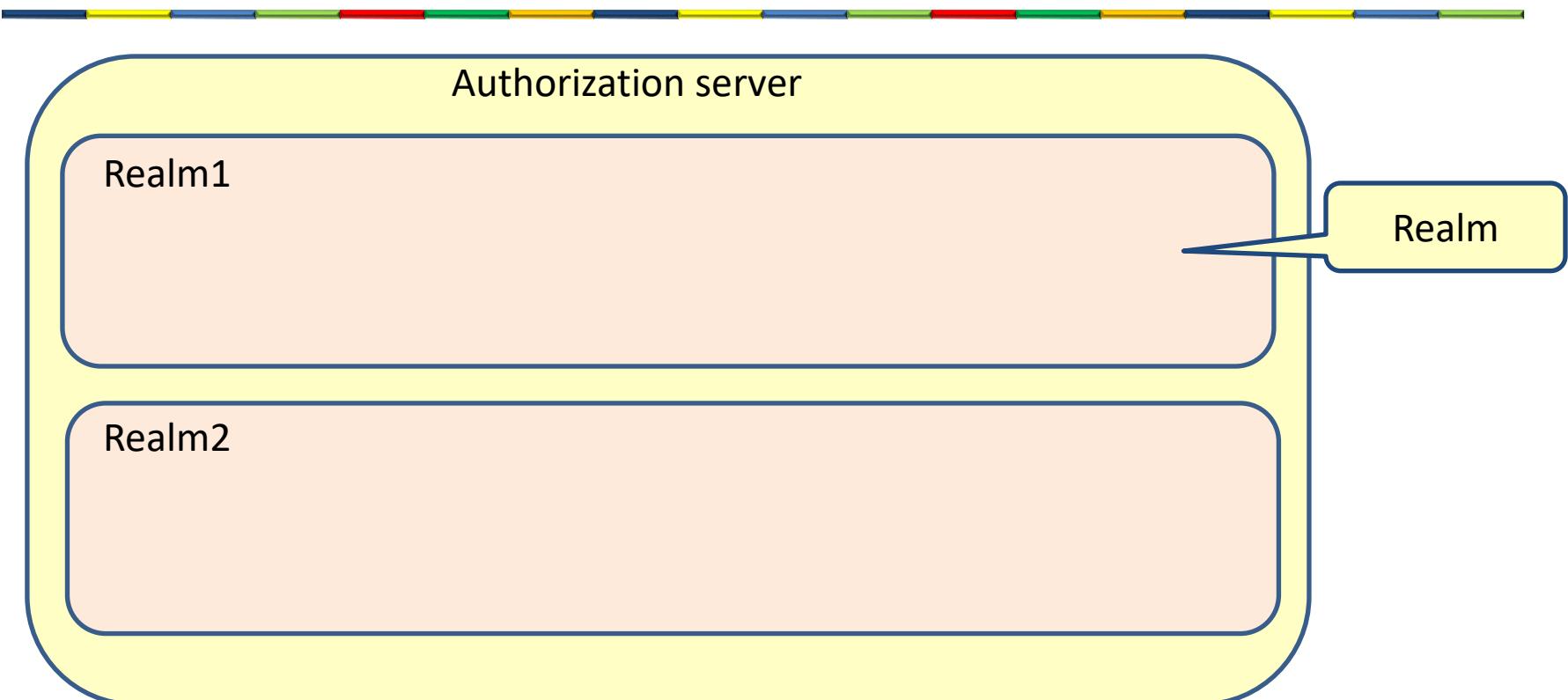
- Annotations:**
 - A yellow callout points to the `access_token` field with the text: "Access token is presented with each call".
 - A yellow callout points to the `refresh_token` field with the text: "Refresh token".
 - A yellow callout points to the `expires_in` field with the text: "Number of seconds before the token expires in seconds".

Refresh token grant



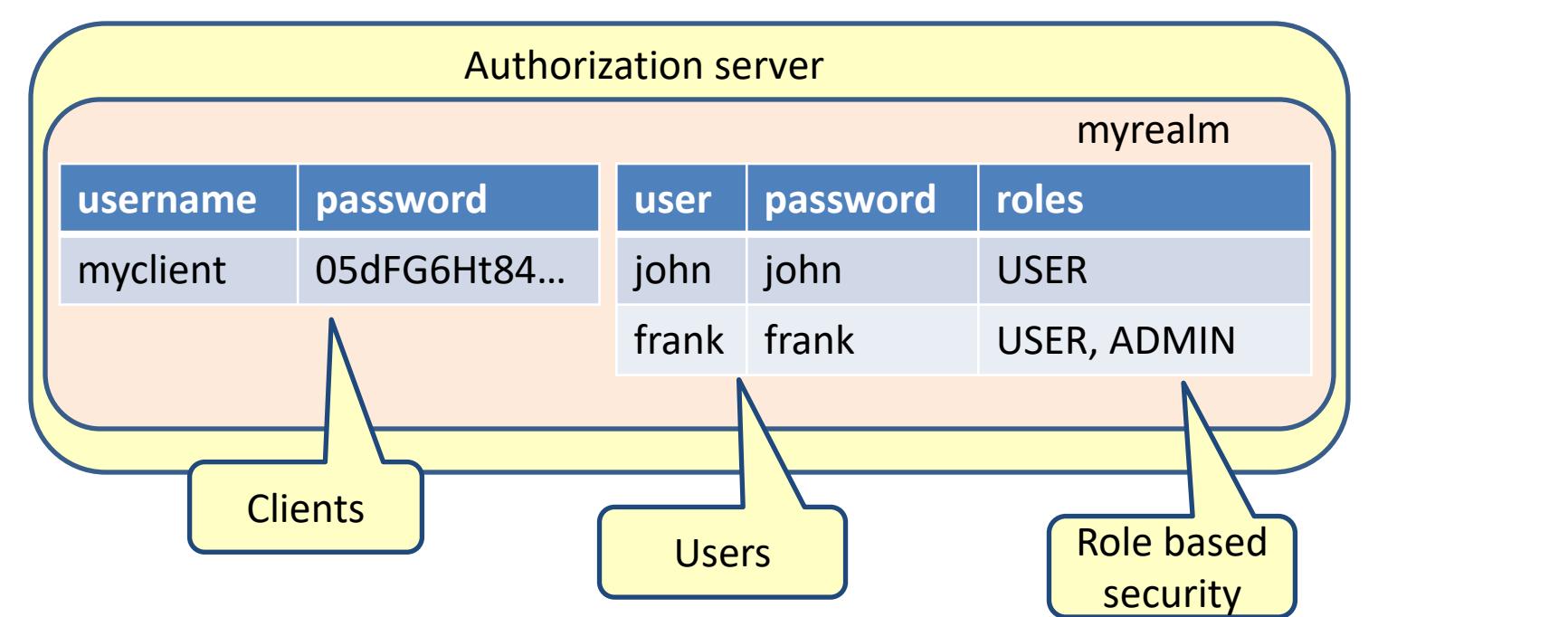
OAUTH2 AUTHORIZATION SERVICE

OAuth2 authorization server



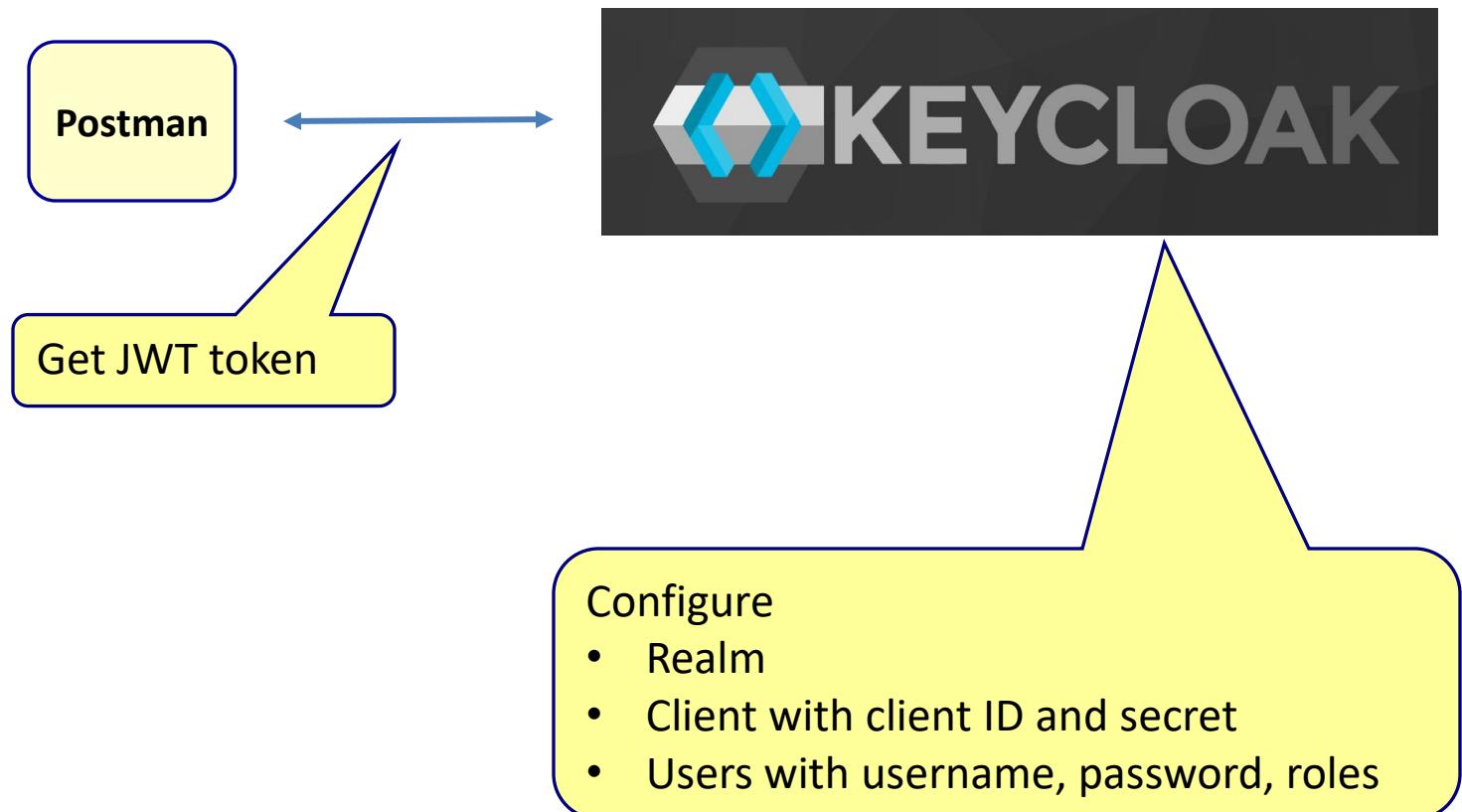
- A **realm** is a **logical isolation boundary** — basically, a **container** that holds **users, clients, roles, and configurations**.

OAuth2 authorization server



- A **client** is an application that wants security information
- A **user** is a human that want to access a secure application (client)

Keycloak authorization server



Keycloak URL's



localhost:8090/realms/myrealm/.well-known/openid-configuration

Pretty-print

```
{  
  "issuer": "http://localhost:8090/realms/myrealm",  
  "authorization_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/auth",  
  "token_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/token",  
  "introspection_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/token/introspect",  
  "userinfo_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/userinfo",  
  "end_session_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/logout",  
  "frontchannel_logout_session_supported": true,  
  "frontchannel_logout_supported": true,  
  "jwks_uri": "http://localhost:8090/realms/myrealm/protocol/openid-connect/certs",  
  "check_session_iframe": "http://localhost:8090/realms/myrealm/protocol/openid-connect/login-status-iframe.html",  
  "grant_types_supported": [  
    "authorization_code",  
    "client_credentials",  
    "implicit",  
    "password",  
    "refresh_token",  
    "urn:ietf:params:oauth:grant-type:device_code",  
    "urn:ietf:params:oauth:grant-type:token-exchange",  
    "urn:ietf:params:oauth:grant-type:uma-ticket",  
    "urn:openid:params:grant-type:ciba"  
,  
  "acr_values_supported": [  
    "0",  
    "1"  
]
```

Retrieve a token

The screenshot shows the Postman interface for a POST request to `http://localhost:8090/realm/myrealm/protocol/openid-connect/token`. The 'Auth' tab is selected, set to 'Basic Auth'. The 'Username' field contains 'myclient' and the 'Password' field contains a redacted password. A note on the left explains that the authorization header will be generated automatically.

The screenshot shows the Postman interface for a POST request to `http://localhost:8090/realm/myrealm/protocol/openid-connect/token`. The 'Body' tab is selected, set to 'x-www-form-urlencoded'. The table lists the following parameters:

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> grant_type	password		
<input checked="" type="checkbox"/> username	john		
<input checked="" type="checkbox"/> password	john		
<input checked="" type="checkbox"/> scope	openid roles		
<input type="checkbox"/>			

Returned payload

The diagram illustrates the structure of a JSON payload returned by a service. The payload is enclosed in a large blue bracket at the top, which is divided into colored segments: dark blue, yellow, green, red, blue, yellow, green, red, blue, yellow, green. Below this, the JSON structure is shown with numbered callouts:

- Access_token** (Callout 1): Points to the value of the "access_token" key, which is a long string of characters.
- Number of seconds before the token expires** (Callout 2): Points to the "expires_in": 3000 field.
- Refresh token** (Callout 3): Points to the "refresh_token" key, which is another long string.

```
"access_token":  
  "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJyeHBmMDJqMWF0UlgsS012S  
  HdLTnZXVFoxcmdzOhdDYlh5RENka11DamtFIn0.  
  eyJleHAiOjE3Njc3NjYsImlhdCI6MTc2MjU2NDc2NiwianRpIjoib25ydHJv0jE40WE5MT  
  k2LThkYWEtMzg3Zi0yZmQ2LTAYzE2ZWNiMTI4MiIsImlzcyI6Imh0dHA6Ly9sb2Nhbgvhc3Q6  
  ODA5MC9yZWFBsbXMvbXlyZWFBsISimF1ZCI6ImFjY291bnQiLCJzdWIi0iJlMTE1NGFkYy0wYj  
  BiLTQwNTQtYWJ1Mi05ZjczNjNkM2FiNjUiLCJ0eXAi0iJCZWfyzXiiLCJhenAi0iJteWNsaWVu  
  dCIsInNpZCI6Ijg4NmFhOTAyLTZhMTEtM2Y4Ny0wNWExLTQ2MTFiMzU1MjU0ZCIsImFjciI6Ij  
  EiLCJhbGxvd2VklW9yaWdpbnMi0lsilYoiXSwicmVhbG1fYWNgZXNzIjp7InJvbGVzIjpbImR1  
  ZmF1bhQtcm9sZXMtBxlyZWFBsISim9mZmxpbmVfYWNgZXNzIwidW1hX2F1dGhvcm16YXRpb2  
  4iXX0sInJlc291cmN1X2FjY2VzcyI6eyJteWNsaWvudCI6eyJyb2xlcyI6WyJVU0VSI119LCJh  
  Y2NvdW50Ijp7InJvbGVzIjpbIm1hbmfnsZs1hY2NvdW50IiwibWFyWd1LWFjY291bnQtbGlua3  
  MiLCJ2aWV3LXByb2ZpbGUIXX19LCJzY29wZSI6Im9wZW5pZCB1bWFpbCBwcm9maWxlIwiZW1h  
  aWxfdmVyaWZpZWQjOnRydWUsIm5hbWUi0iJqb2huIGRvZSIisInByZWZlcnJlZF91c2VybmrTzS  
  I6ImpvaG4iLCJnaXZ1b19uYW11Ijoiam9obiIsImZhbWlseV9uYW11IjoizG91IiwiZW1haWwi  
  OiJqb2huQGdtYWlsLmNvbsJ9.  
  nkt1Xac0sCziWLWdgV2PzSFaW-aTBjkFvFvrxynbiSoGmUyKD1AljoWUtLqd_hLx6kWtTaMzc  
  o-MCXZj3U5QyWWtXEsxCA-d4eDUTJE1jEG06yQIN2d3NiZcbLrYyP6B71Nd33oh7UEchBeXW0e  
  F-b1HUYs78IxysYwFJE0cxXhD5fu6wtrDWB5DKY91W3z5gJXujsNhi9VQ6g0evEaYWBD_iKWmZ0  
  sefj_IuxiWQj5jNCnFFVj6nrhuF0VoqXAonf5a3PsUQtqy5quk2yg60MUssSeGB_0FUb9leV1sN  
  zXNmuWd1SAoFo--9gizhG4anrFAAA2PsYt0zSgRBW8wqfA",  
  "expires_in": 3000,  
  "refresh_expires_in": 1800,  
  "refresh_token":  
    "eyJhbGciOiJIUzUxMiIsInR5cCIgOiAiSldUIiwia2lkIiA6IC12YWU4ZjNkOS1hYmN1LTQ0Y  
    zUtYmQxYy0yMzdjMzMwY2MxY2IifQ.  
    eyJleHAiOjE3Njc3NjY1NjYsImlhdCI6MTc2MjU2NDc2NiwianRpIjoizGM0ZjBiMTQtOGUwMy  
    04NjI1LTU1MmEtNDQ1ZWZ1YzcxZDdiIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDkwL3J1  
    YWxtcy9teXJ1YWxtIiwiYXVkiIjoiaHR0cDovL2xvY2FsaG9zdDo4MDkwL3J1YWxtcy9teXJ1YW  
    xtIiwiC3ViIjoizTExNTRhZGMtMGIwYi00MDU0LWFfZTIt0WY3MzYzZDNhYjY1IiwidHlwIjoi  
    UmVmcmVzaCIsImF6cCI6Im15Y2xpZW50IiwiC2lkIioi0Dg2YWE5MDItNmExMS0zZig3LTA1YT
```

Get user information

The screenshot shows a Postman API request to retrieve user information. The URL is `http://localhost:8090/realm/myrealm/protocol/openid-connect/userinfo`. The request method is GET. The 'Auth' tab is selected, showing 'Bearer Token' as the type. A yellow callout points to the token field, which contains a redacted string and icons for copy, eye, and lock. The 'Body' tab shows a JSON response with user details. A yellow callout points to the response body, which is labeled 'User information from john'. The response status is 200 OK with 97 ms latency and 432 B size.

GET http://localhost:8090/realm/myrealm/protocol/openid-connect/userinfo Send

Params Auth Headers (9) Body Scripts Tests Settings Cookies

Auth Type

Bearer Token

The authorization header will be automatically generated when you send the request.

Learn more about [Bearer Token](#) authorization.

Token

.....

200 OK • 97 ms • 432 B •

{ } JSON Preview Visualize

1 {
2 "sub": "e1154adc-0b0b-4054-abe2-9f7363d3ab65",
3 "email_verified": true,
4 "name": "john doe",
5 "preferred_username": "john",
6 "given_name": "john",
7 "family_name": "doe",
8 "email": "john@gmail.com"
9 }

User information from john

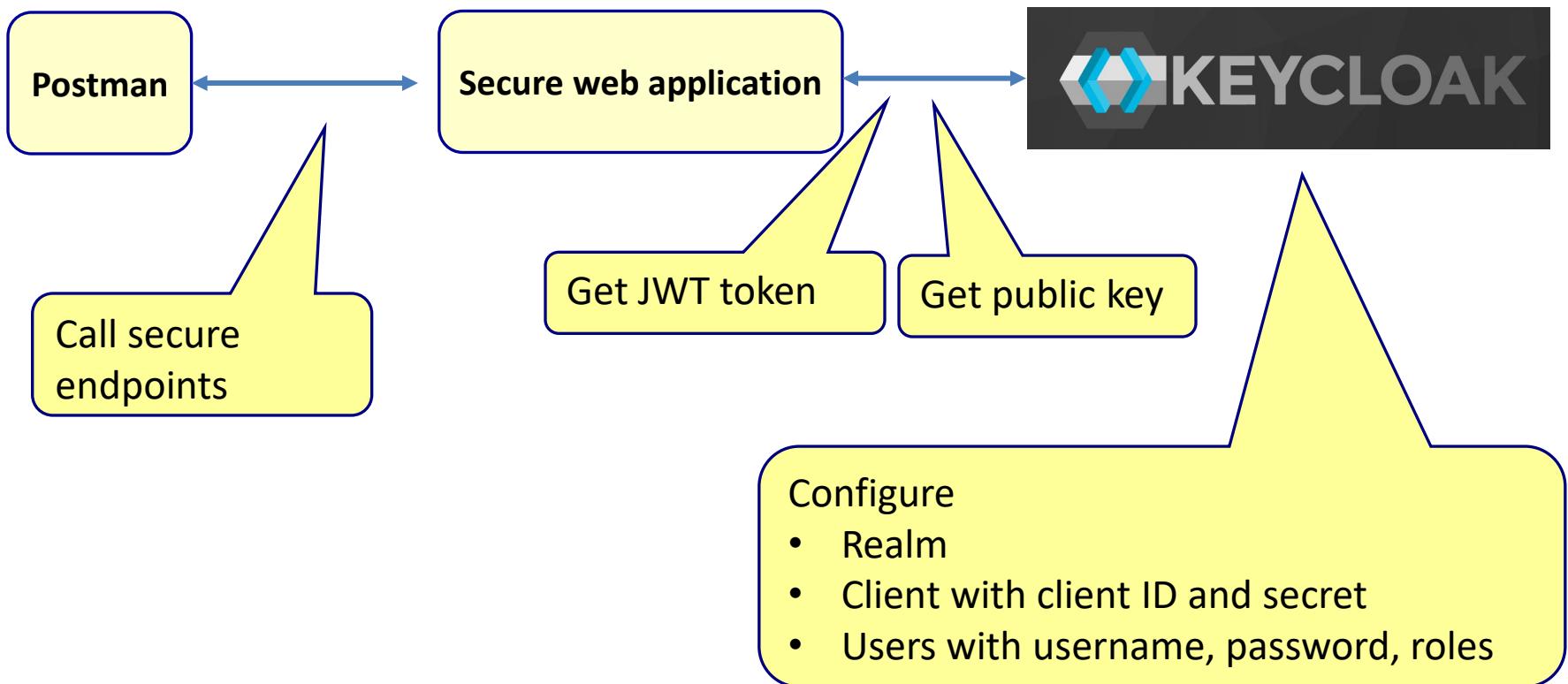
See token content

The screenshot shows a browser window with the URL jwt.io. The page title is "JWT Debugger". The main content area displays a JSON object representing a JWT payload. The payload includes sections for realm access, resource access, account, and scope, along with user metadata like name and email.

```
realm_access
"roles": [
    "default-roles-myrealm",
    "offline_access",
    "uma_authorization"
],
"resource_access": {
    "myclient": {
        "roles": [
            "USER"
        ]
    },
    "account": {
        "roles": [
            "manage-account",
            "manage-account-links",
            "view-profile"
        ]
    }
},
"scope": "openid email profile",
"email_verified": true,
"name": "john doe",
"preferred_username": "john",
"given_name": "john",
"family_name": "doe",
"email": "john@gmail.com"
}
```

A SECURE APPLICATION

Secure application



A secure application



Authentication server

username	password
myclient	4H0uT54...

user	password	roles
john	john	USER
frank	frank	USER, ADMIN

Secure application

URL	Roles
/phone	USER
/salary	ADMIN

A secure application

```
@RestController  
public class EmployeeController {  
    @GetMapping("/name")  
    public String getName() {  
        return "Frank Brown";  
    }  
  
    @PreAuthorize("hasRole('ADMIN')")  
    @GetMapping("/salary")  
    public String getGetSalary() {  
        return "95.000";  
    }  
  
    @PreAuthorize("hasRole('USER')")  
    @GetMapping("/phone")  
    public String getPhone() {  
        return "645322899";  
    }  
}
```

Method based security

Security configuration

```
@Configuration  
@EnableWebSecurity  
@EnableMethodSecurity  
public class SecurityConfig {  
    @Bean  
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
        http.httpBasic(Customizer.withDefaults());  
  
        http  
            .oauth2ResourceServer(oauth2 -> oauth2  
                .jwt(jwt -> jwt.jwtAuthenticationConverter(jwtAuthenticationConverter())));  
  
        http  
            .sessionManagement(session -> session  
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
            );  
  
        return http.build();  
    }  
}
```

Method based security

Security configuration

```
@Bean
public JwtAuthenticationConverter jwtAuthenticationConverter() {
    JwtAuthenticationConverter converter = new JwtAuthenticationConverter();
    converter.setJwtGrantedAuthoritiesConverter(jwt -> {
        Set<String> roles = new HashSet<>();

        // Realm roles
        Map<String, Object> realmAccess = jwt.getClaimAsMap("realm_access");
        if (realmAccess != null && realmAccess.get("roles") instanceof Collection<?> realmRoles) {
            roles.addAll(realmRoles.stream().map(Object::toString).toList());
        }

        // Client roles (replace "myclient" with your client ID)
        Map<String, Object> resourceAccess = jwt.getClaimAsMap("resource_access");
        if (resourceAccess != null && resourceAccess.get("myclient") instanceof Map<?, ?> clientAccess) {
            Object clientRoles = ((Map<?, ?>) clientAccess).get("roles");
            if (clientRoles instanceof Collection<?> clientRolesList) {
                roles.addAll(clientRolesList.stream().map(Object::toString).toList());
            }
        }

        // Convert to SimpleGrantedAuthority with ROLE_ prefix
        return roles.stream()
            .map(role -> new SimpleGrantedAuthority("ROLE_" + role))
            .collect(Collectors.toSet());
    });
    return converter;
}
```

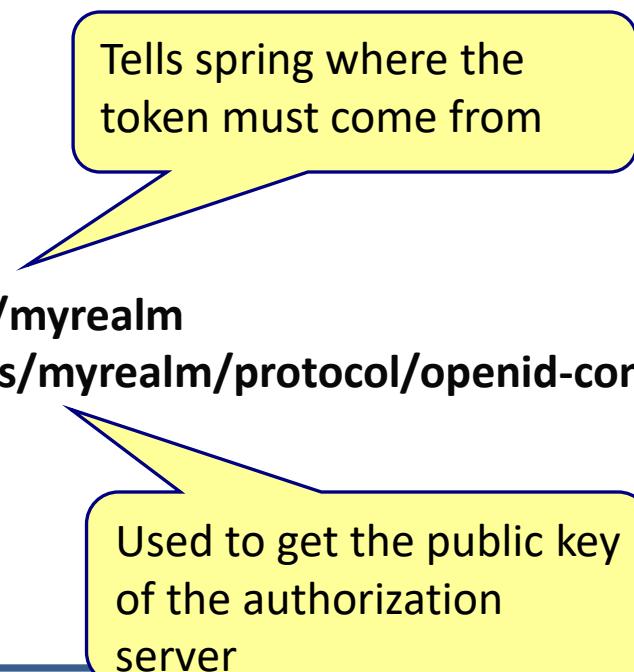
Make sure we get both the realm and the client roles

Spring expects “ROLE_USER” instead of “USER”

The configuration

application.yml

```
spring:  
  security:  
    oauth2:  
      resourceserver:  
        jwt:  
          issuer-uri: http://localhost:8090/realm/myrealm  
          jwk-set-uri: http://localhost:8090/realm/myrealm/protocol/openid-connect/certs  
  
      server:  
        port: 8081
```



Tells spring where the token must come from

Used to get the public key of the authorization server

Get the user name

The screenshot shows the POSTMAN application interface. At the top, there is a header bar with a color gradient. Below it, the main interface has a search bar with "GET" and "http://localhost:8081/name". To the right of the search bar is a "Send" button. Below the search bar, there are tabs for "Params", "Auth", "Headers (8)", "Body", "Scripts", "Tests", and "Settings". The "Auth" tab is currently selected and underlined. A dropdown menu labeled "No Auth" is open. On the right side of the interface, there is a section titled "No Auth" with the subtext "This request does not use any authorization." Below this, there is a "Body" section with a "Raw" tab selected, showing the response "200 OK" and the content "Frank Brown". There are also "Preview" and "Visualize" tabs. At the bottom right, there is a yellow callout bubble with the text "Everyone can get the user name without authorization".

Get the phone number

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** http://localhost:8081/phone
- Auth Type:** No Auth
- Headers (8):** (This section is visible but empty)
- Body:** (This section is visible but empty)
- Tests:** (This section is visible but empty)
- Scripts:** (This section is visible but empty)
- Settings:** (This section is visible but empty)
- Cookies:** (This section is visible but empty)
- No Auth:** A note stating "This request does not use any authorization." with an info icon.
- Response Status:** 401 Unauthorized
- Response Time:** 13 ms
- Response Size:** 342 B
- Global:** (checkbox)
- More Options:** (ellipsis)
- Body Content:** Raw (selected), Preview, Visualize
- Tools:** (copy, search, refresh, link)

A yellow callout bubble with a black border and blue outline points from the bottom right towards the "No Auth" note, containing the text: "You cannot get the phone number without access token".

Get the phone number

A screenshot of the Postman application interface. At the top, a navigation bar shows 'GET' and the URL 'http://localhost:8081/phone'. Below the URL is a 'Send' button. Underneath the URL, there are tabs for 'Params', 'Auth' (which is currently selected), 'Headers (9)', 'Body', 'Scripts', 'Tests', and 'Settings'. The 'Auth' tab has a dropdown set to 'Bearer Token'. A yellow box highlights the 'Token' field, which contains a redacted token. To the left of the token field, a note says: 'The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.' In the bottom right corner of the main window, there is a status bar showing '200 OK', '13 ms', '342 B', and other metrics.

John can get the phone number

A second screenshot of the Postman application interface, similar to the first one. It shows a GET request to 'http://localhost:8081/phone'. The 'Auth' tab is selected, showing 'Bearer Token' in the dropdown. A yellow box highlights the 'Token' field, which contains a redacted token. To the left of the token field, a note says: 'The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.' The bottom right corner shows a status bar with '200 OK', '17 ms', '342 B', and other metrics.

Frank can get the phone number

Get the salary

The authorization header will be automatically generated when you send the request.
Learn more about [Bearer Token](#) authorization.

Frank can get the salary (role=ADMIN)

John cannot get the salary (role = USER)

200 OK • 14 ms • 339 B

1 95.000

403 Forbidden • 9 ms • 509 B

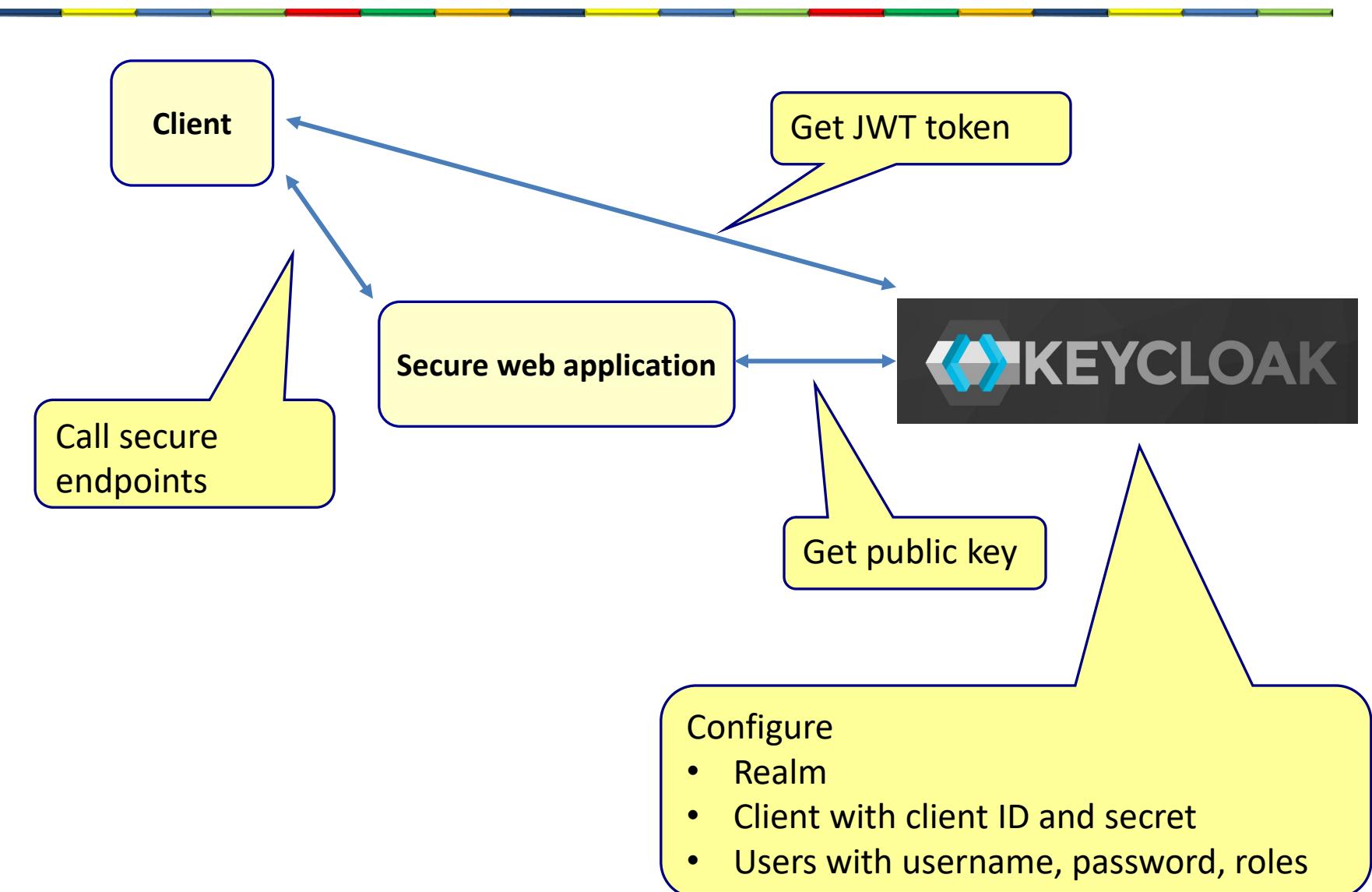
1

Main point

- To implement security in a microservice architecture we use token based security (OAuth2). *The human nervous system is able to access that most basic field of pure consciousness which is the source of all the laws of Nature.*

CLIENT FOR THE SECURE APPLICATION

Client for the secure application



Client for the secure application

```
@SpringBootApplication
public class Client implements CommandLineRunner {

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

    @Override
    public void run(String... args) throws Exception {
        callRemoteService("http://localhost:8081/name", "nobody", "nobody");
        callRemoteService("http://localhost:8081/phone", "john", "john");
        callRemoteService("http://localhost:8081/phone", "frank", "frank");
        callRemoteService("http://localhost:8081/salary", "john", "john");
        callRemoteService("http://localhost:8081/salary", "frank", "frank");
    }
}
```

No token for user nobody

Call to http://localhost:8081/name for user nobody gave the response Frank Brown

Call to http://localhost:8081/phone for user john gave the response 645322899

Call to http://localhost:8081/phone for user frank gave the response 645322899

Forbidden: insufficient roles http://localhost:8081/salary with user john

Call to http://localhost:8081/salary for user john gave the response null

Call to http://localhost:8081/salary for user frank gave the response 95.000

Client of the secure application

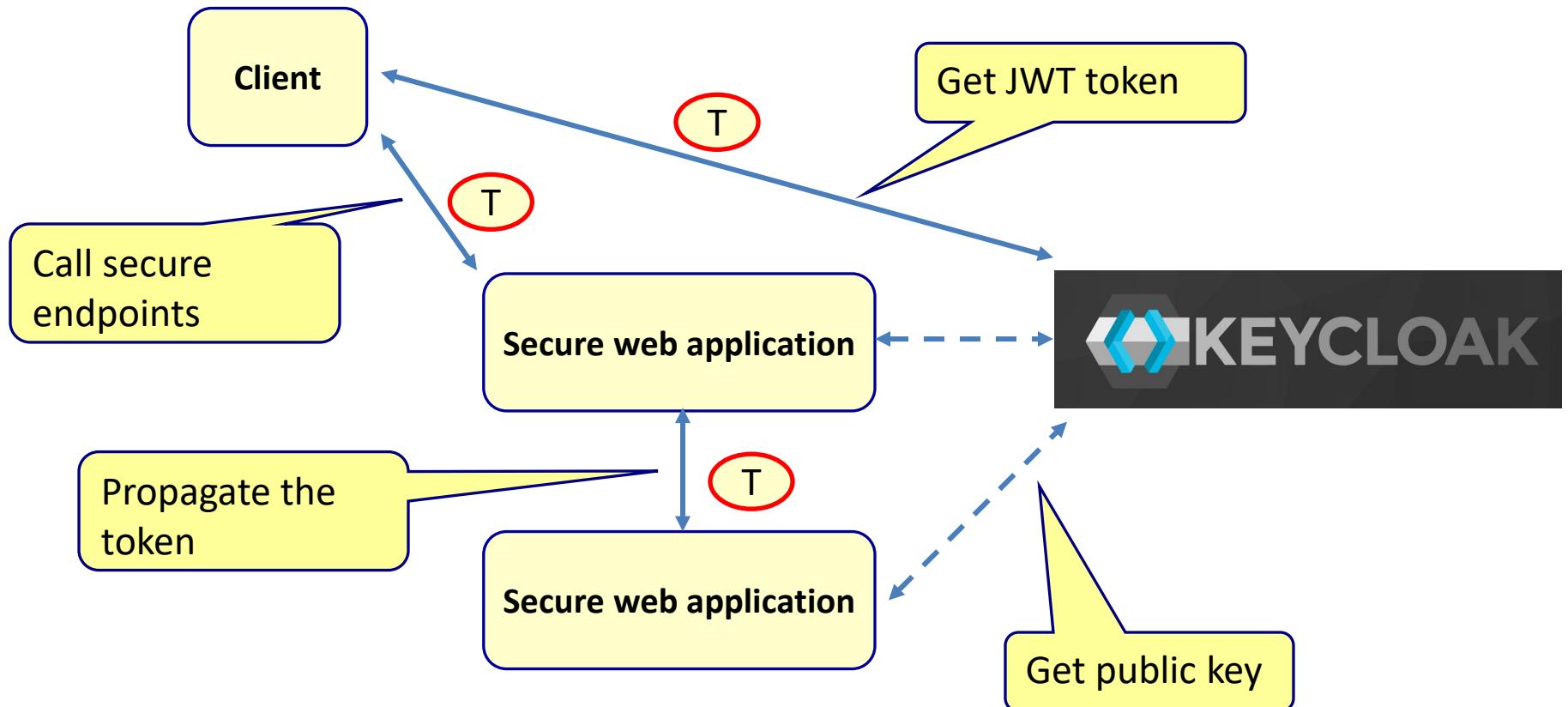
```
public void callRemoteService(String url, String username, String password){  
    String accessToken = getAccessToken(username, password);  
    // Call resource server  
    String response = WebClient.create(url)  
        .get()  
        .headers(h ->{  
            if (accessToken != null)  
                h.setBearerAuth(accessToken);  
        })  
        .retrieve()  
        .onStatus(status -> status.value() == 401, clientResponse -> {  
            System.out.println("Error: Unauthorized - token missing or expired for call "+url+" with user "+username);  
            return Mono.empty(); // Do NOT throw  
        })  
        .onStatus(status -> status.value() == 403, clientResponse -> {  
            System.out.println("Forbidden: insufficient roles "+url+" with user "+username);  
            return Mono.empty(); // Do NOT throw  
        })  
        .bodyToMono(String.class)  
        .block();  
  
    System.out.println("Call to " +url+ " for user "+username+" gave the response "+ response);  
}
```

Client of the secure application

```
public String getAccessToken(String username, String password){  
    WebClient webClient = WebClient.builder()  
        .baseUrl("http://localhost:8090/realm/myrealm/protocol/openid-connect/token")  
        .build();  
    // Get access token  
    Map<String, Object> tokenResponse = webClient.post()  
        .header("Content-Type", "application/x-www-form-urlencoded")  
        .body(BodyInserters.fromFormData("grant_type", "password")  
            .with("username", username)  
            .with("password", password)  
            .with("scope", "openid roles")  
            .with("client_id", "myclient")  
            .with("client_secret", "0maxLeON2d9gwyF1ilv52YBxgB5AoXGL"))  
        .retrieve()  
        .onStatus(status -> status.value() == 401, clientResponse -> {  
            System.out.println("No token for user "+username);  
            return Mono.empty(); // Do NOT throw  
        })  
        .bodyToMono(Map.class)  
        .block();  
  
    String accessToken = (String) tokenResponse.get("access_token");  
    return accessToken;  
}
```

PROPAGATING THE TOKEN

Client for the secure application



Downstream secure application

```
@RestController  
public class PrivateDataController {  
    @PreAuthorize("hasRole('ADMIN')")  
    @GetMapping("/private")  
    public String getPrivateData() {  
        return "This is very private data";  
    }  
}
```

application.yml

```
spring:  
  security:  
    oauth2:  
      resourceserver:  
        jwt:  
          issuer-uri: http://localhost:8090/realm/myrealm  
          jwk-set-uri: http://localhost:8090/realm/myrealm/protocol/openid-connect/certs  
  
  server:  
    port: 8082
```

Upstream secure application

```
@PreAuthorize("hasRole('ADMIN')")
@GetMapping("/sensitive")
public String getPrivateData(@AuthenticationPrincipal Jwt jwt) {
    String token = jwt.getTokenValue();
    WebClient webClient = WebClient.builder()
        .baseUrl("http://localhost:8082")
        .build();
    // Call the downstream application with the same token
    String response = webClient.get()
        .uri("/private")
        .headers(headers -> headers.setBearerAuth(token))
        .retrieve()
        .bodyToMono(String.class)
        .block();

    return response;
}
```

Get JWT token

Propagate the token

Change the Client

```
@Override  
public void run(String... args) throws Exception {  
    callRemoteService("http://localhost:8081/name", "nobody", "nobody");  
    callRemoteService("http://localhost:8081/phone", "john", "john");  
    callRemoteService("http://localhost:8081/phone", "frank", "frank");  
    callRemoteService("http://localhost:8081/salary", "john", "john");  
    callRemoteService("http://localhost:8081/salary", "frank", "frank");  
    callRemoteService("http://localhost:8081/sensitive", "frank", "frank");  
}
```

No token for user nobody

Call to http://localhost:8081/name for user nobody gave the response Frank Brown

Call to http://localhost:8081/phone for user john gave the response 645322899

Call to http://localhost:8081/phone for user frank gave the response 645322899

Forbidden: insufficient roles http://localhost:8081/salary with user john

Call to http://localhost:8081/salary for user john gave the response null

Call to http://localhost:8081/salary for user frank gave the response 95.000

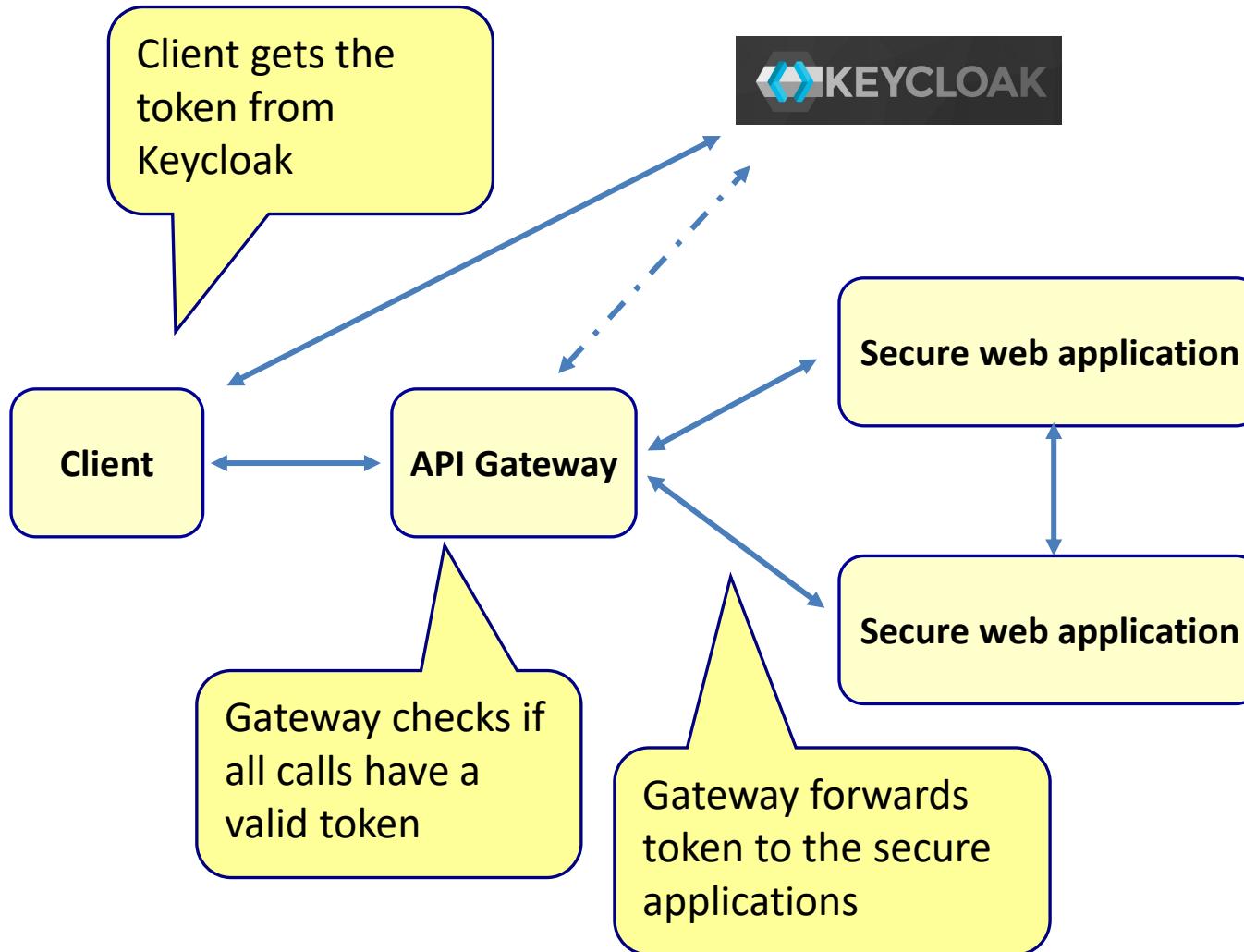
Call to http://localhost:8081/sensitive for user frank gave the response This is very private data

Main point

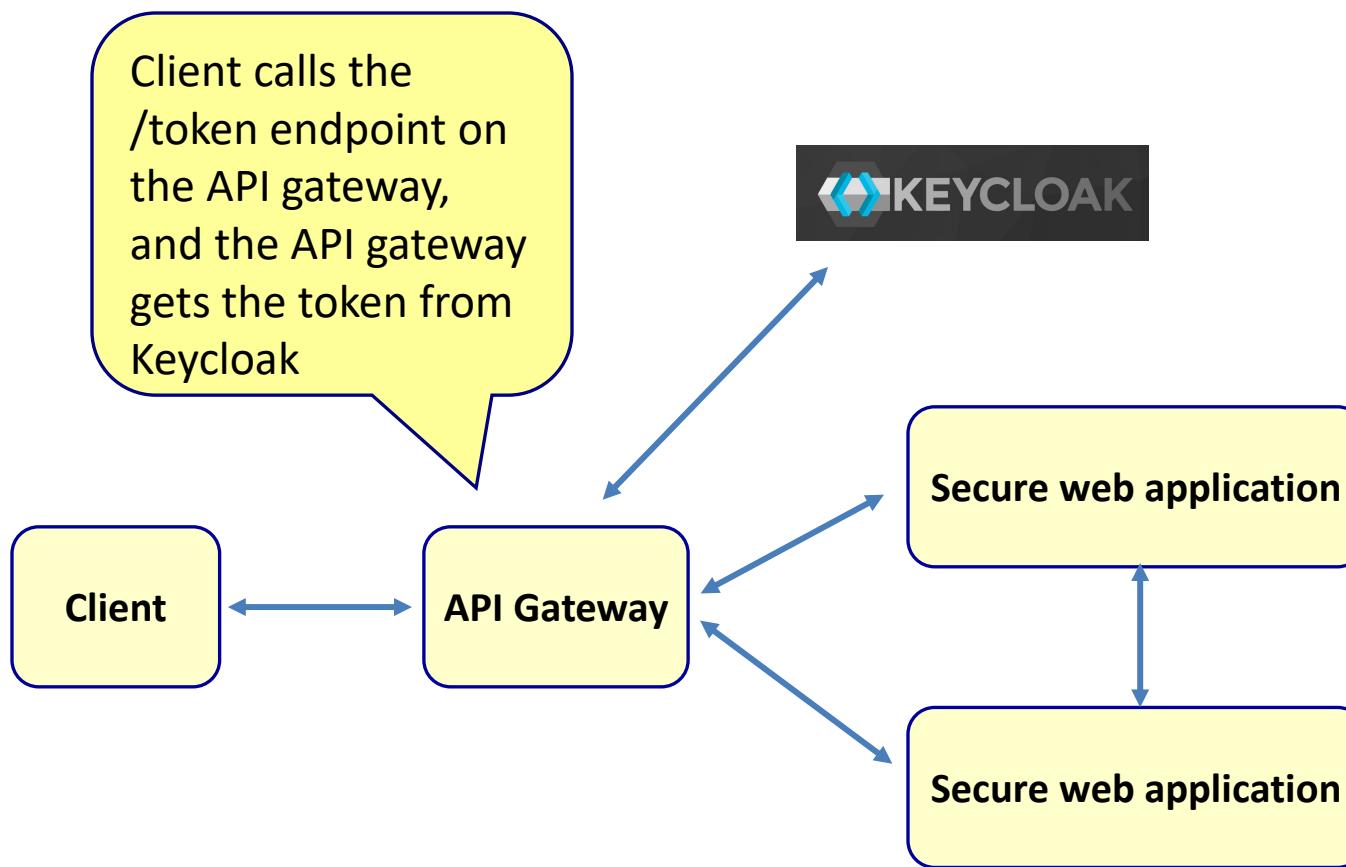
- With JWT a token receiver can verify the correctness of the token using the signature within the token. *The TM technique is the key to transcend and access pure consciousness.*

LETS ADD THE API GATEWAY

API Gateway option 1



API Gateway option 2



API Gateway security

```
spring:  
  application:  
    name: api-gateway  
  security:  
    oauth2:  
      resourceserver:  
        jwt:  
          issuer-uri: http://localhost:8090/realm/myrealm  
          jwk-set-uri: http://localhost:8090/realm/myrealm/protocol/openid-connect/certs  
  cloud:  
    gateway:  
      server:  
        webflux:  
          routes:  
            - id: upstream  
              uri: http://localhost:8081  
              predicates:  
                - Path=/name, /phone, /salary, /sensitive  
  
            - id: downstream  
              uri: http://localhost:8082  
              predicates:  
                - Path=/private  
  server:  
    port: 8085
```

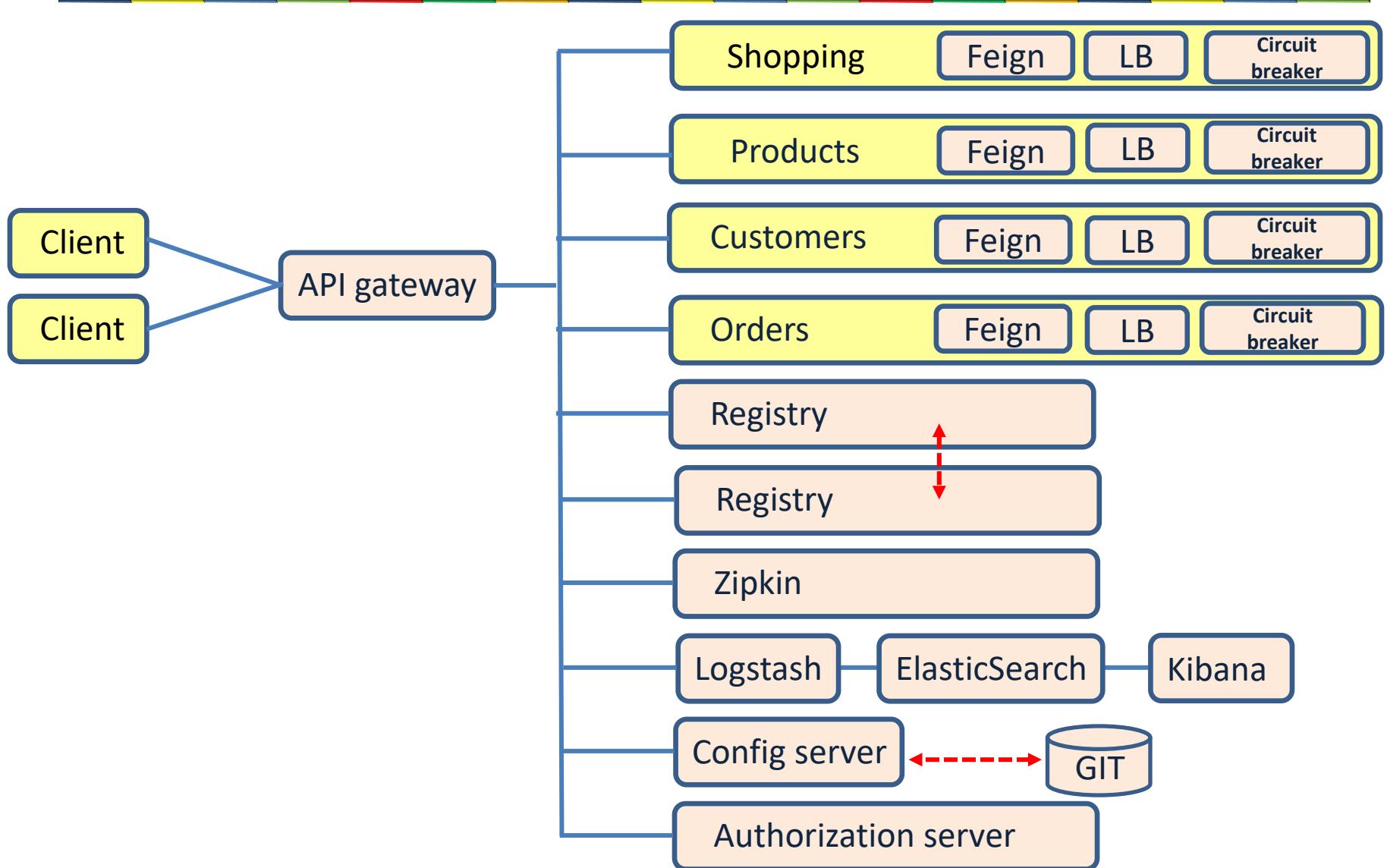
API Gateway security

```
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {
        http
            .csrf(ServerHttpSecurity.CsrfSpec::disable)

            .authorizeExchange(exchanges -> exchanges
                .pathMatchers("/name").permitAll()      // <- exclude /name from authentication
                .anyExchange().authenticated()         // all other routes require valid JWT
            )
            .oauth2ResourceServer(oauth2 -> oauth2
                .jwt(jwt -> {})
            );

        return http.build();
    }
}
```

Implementing microservices



Challenges of a microservice architecture

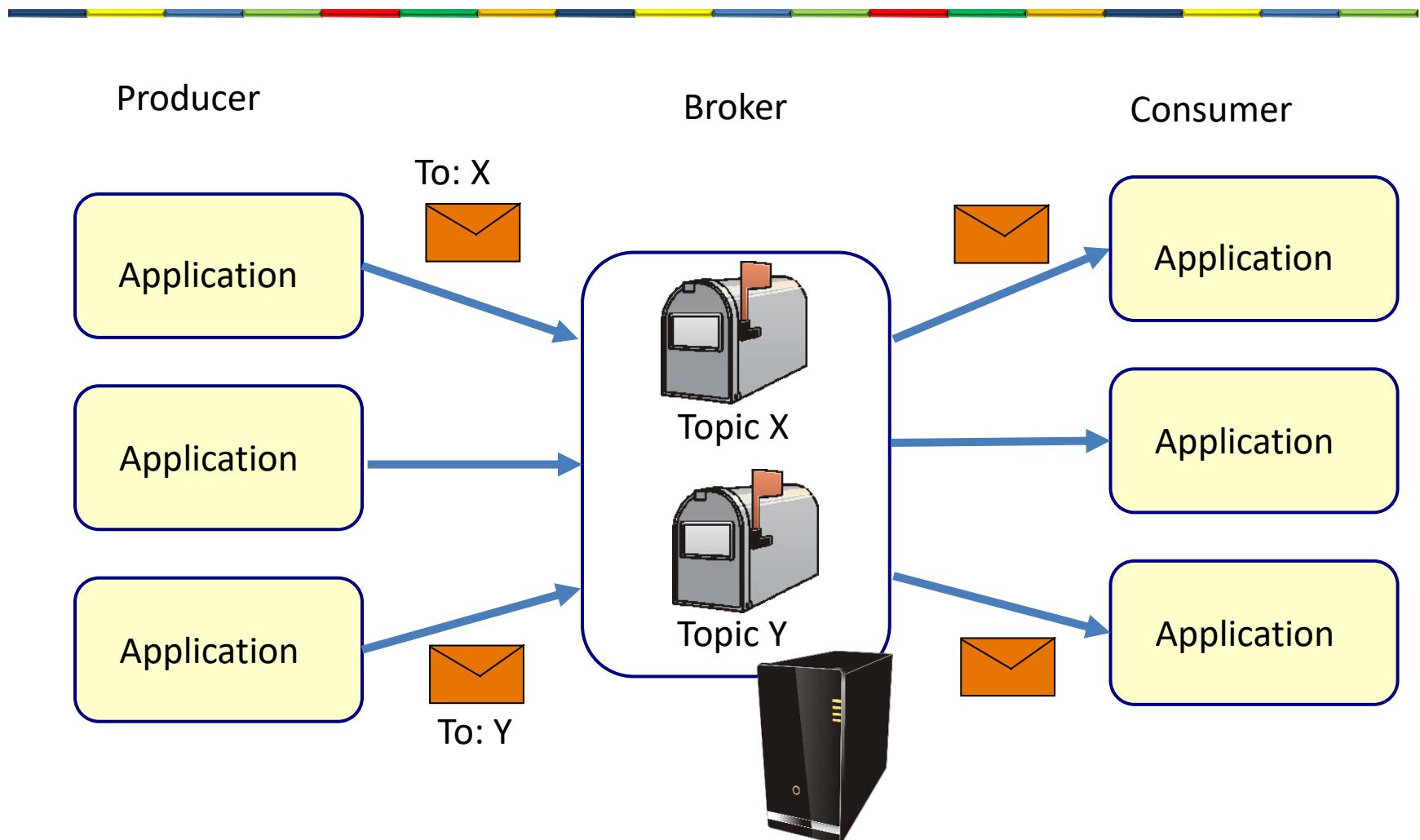
Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	CQRS
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	Token based security (OAuth2) Digitally signed (JWT) tokens
Transactions	Compensating transactions Eventual consistency
Keep data in sync	Publish-subscribe data change event
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	ELK + beats
Follow/monitor business processes	Zipkin ELK

Lesson 11

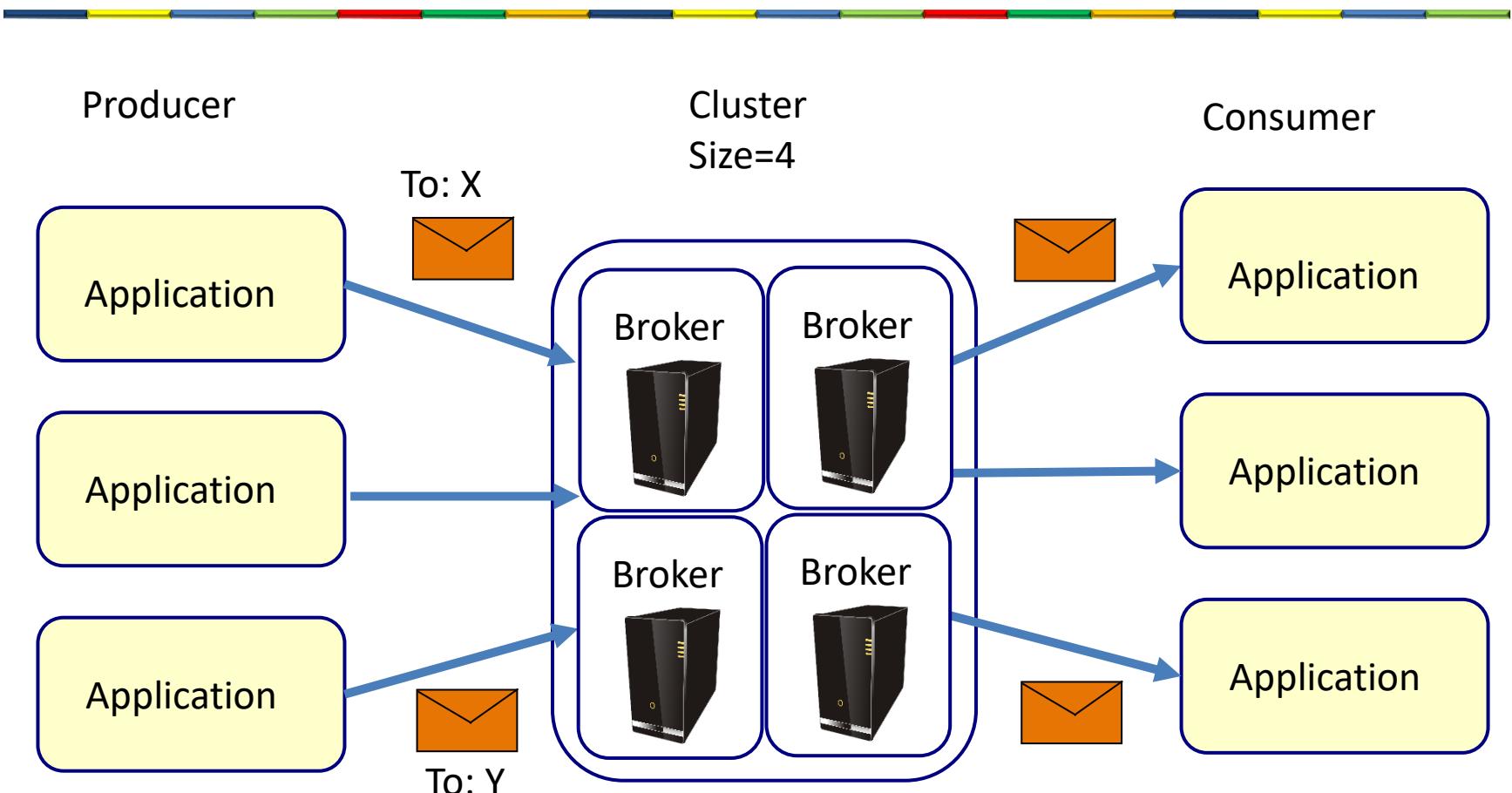
KAFKA

KAFKA BASICS

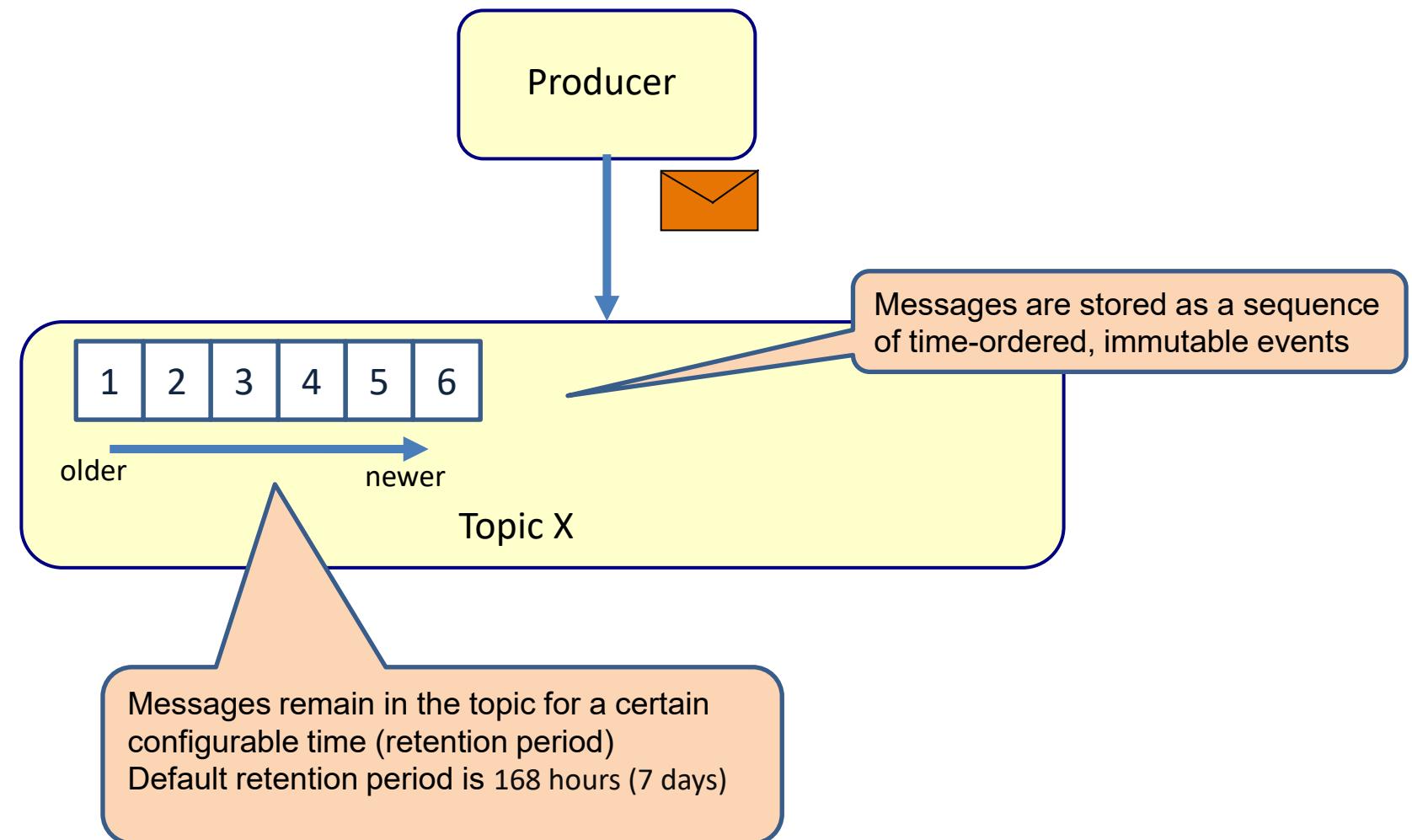
Kafka



Cluster of Brokers



Event sourcing



Message

```
{
```

```
  "Timestamp": "2025-09-02T09:36:14.532+00:00",
```

Set when Kafka receives the message

```
  "Topic": "topicZ",
```

```
  "Partition": 0,
```

```
  "Offset": 1,
```

```
  "SchemaId": null,
```

```
  "SchemaType": null,
```

```
  "Key": [
```

```
    107,
```

Key (optional)

```
    101,
```

```
    121
```

```
  ],
```

Headers(optional)

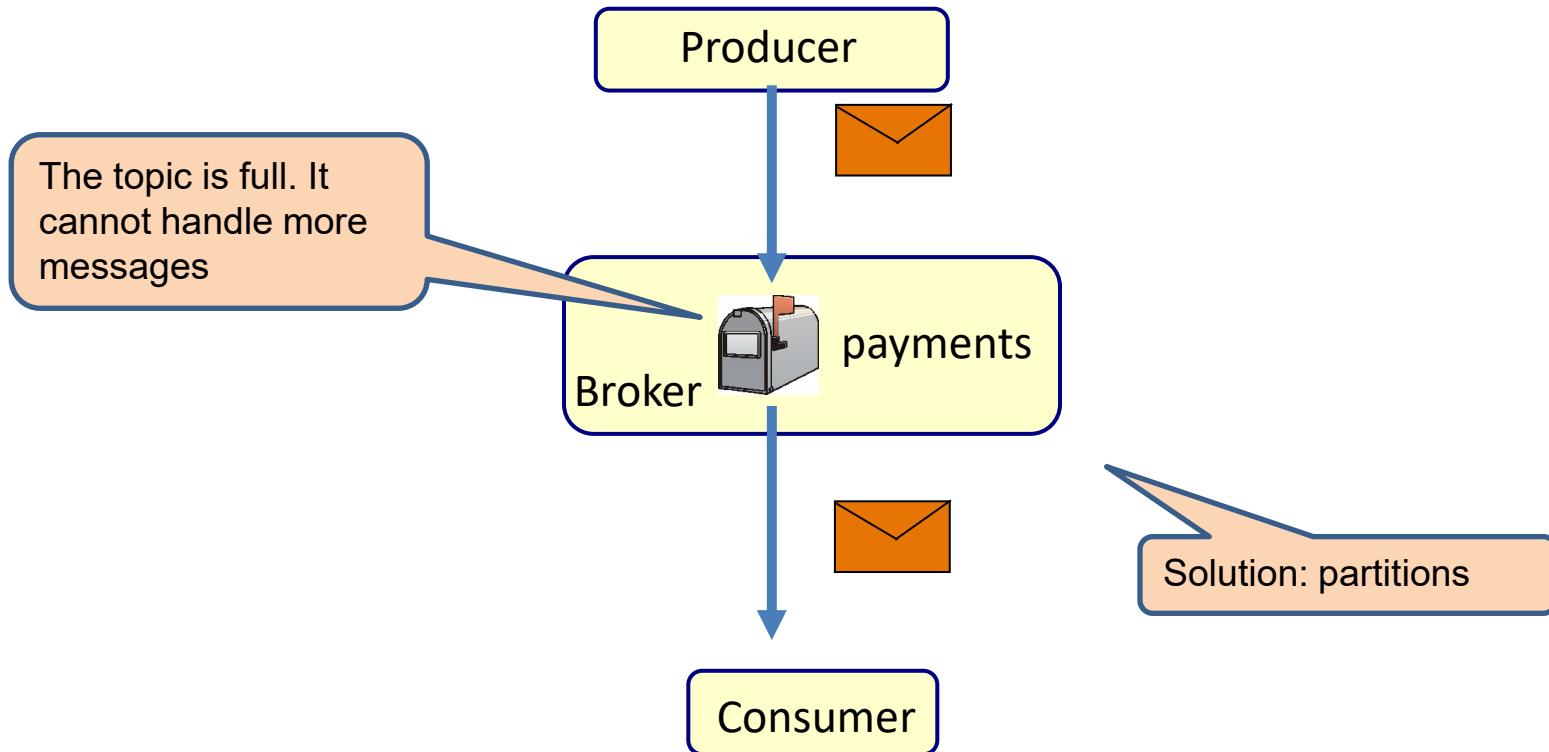
```
  "Headers": null,
```

```
  "Message": "Hello World"
```

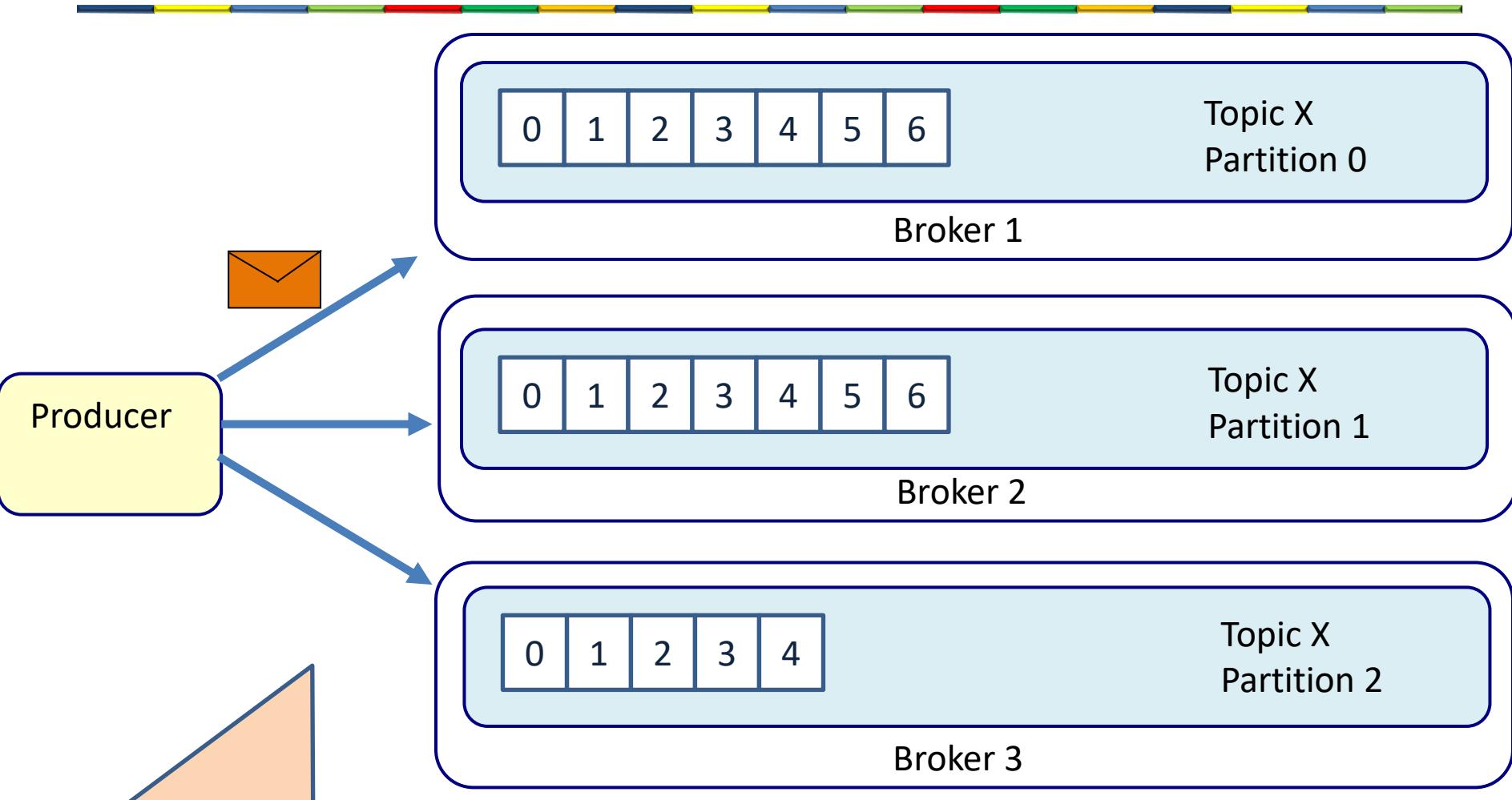
Content of the message

```
}
```

What if the topic gets too full?



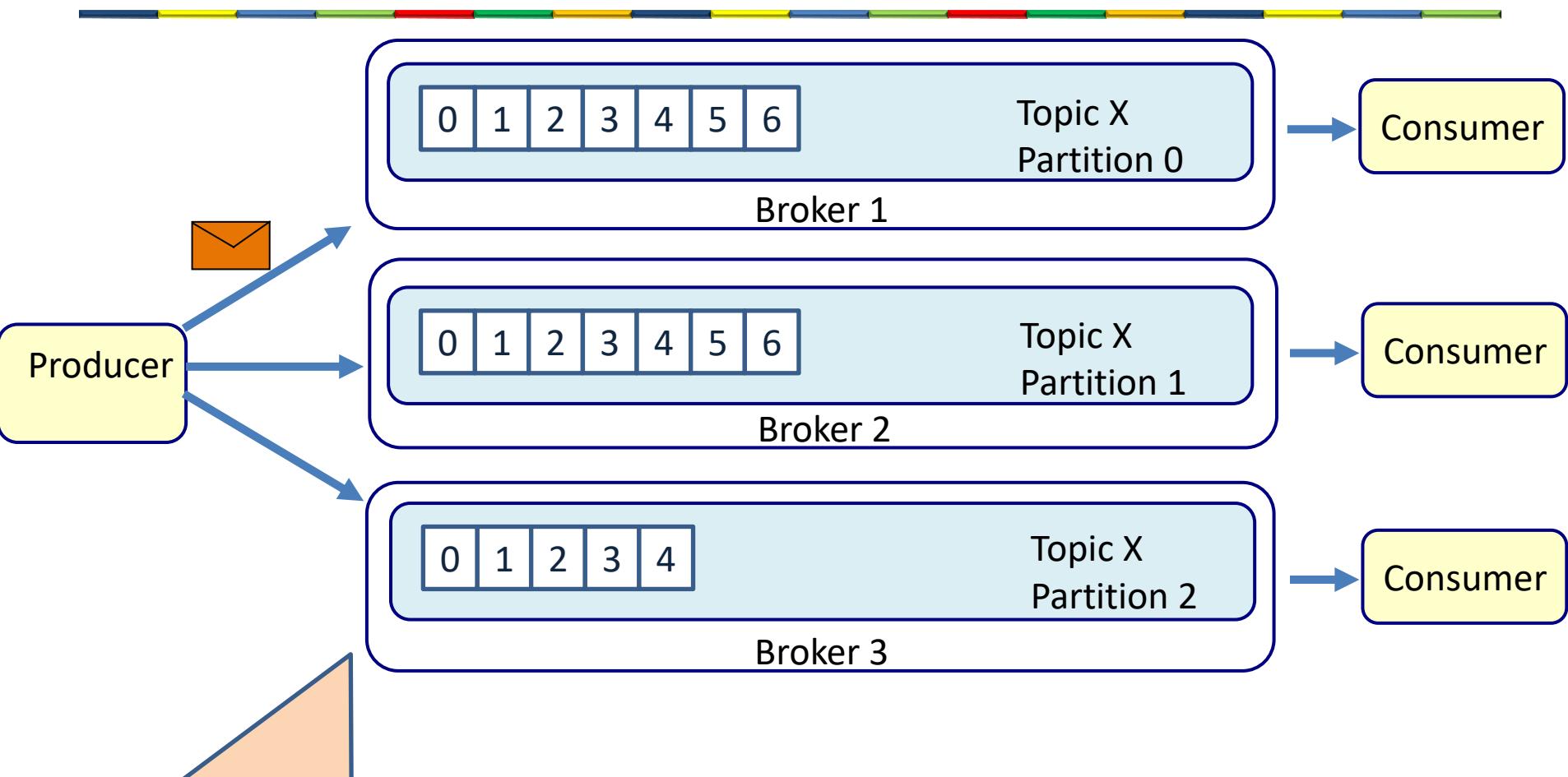
Scale out partitions



Increases

- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

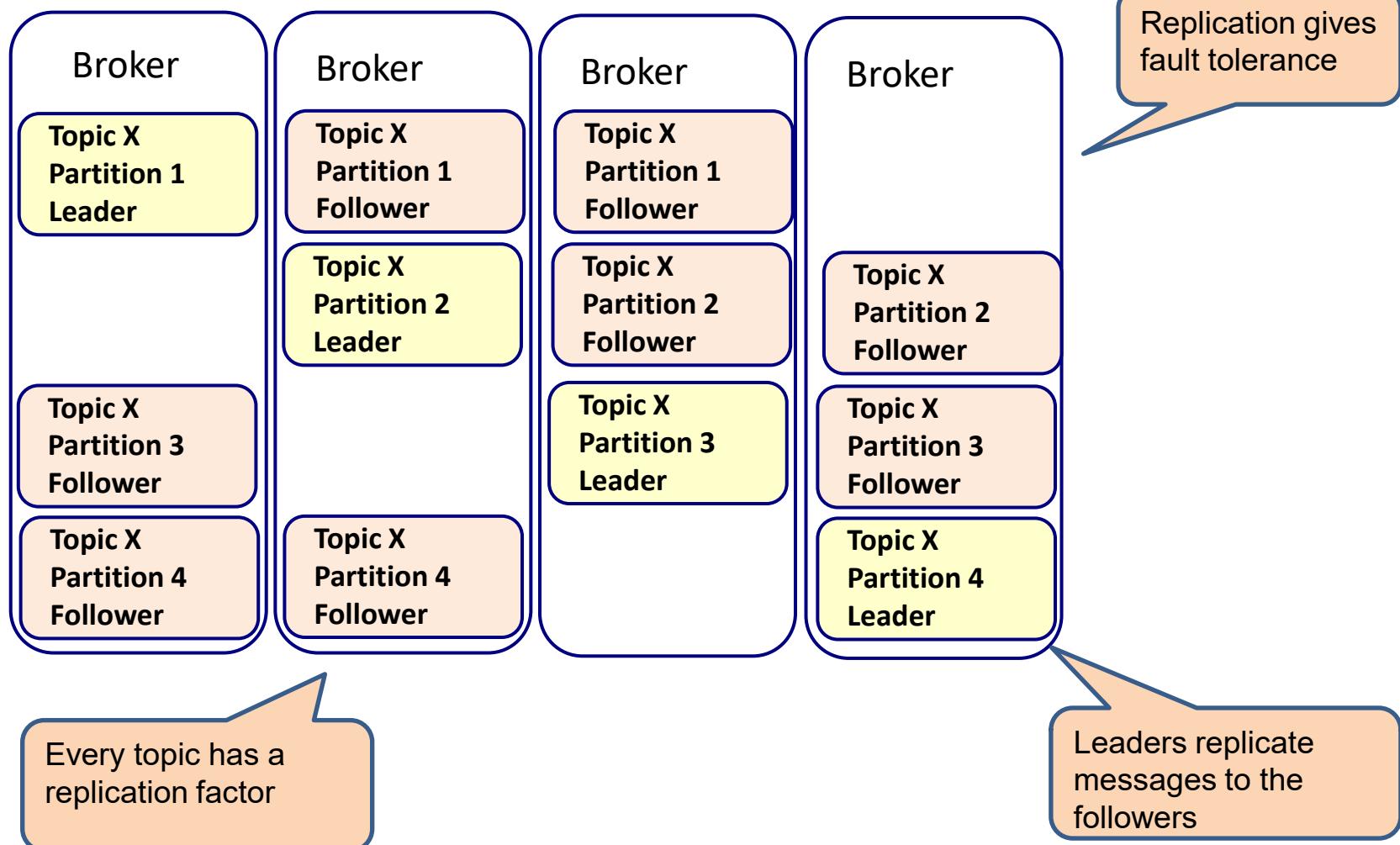
Scale out partitions



Increases

- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

Replication



SPRING BOOT KAFKA

Kafka producer

```
@Service  
public class KafkaProducer {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() {  
        kafkaTemplate.send("topic1", "Hello World");  
    }  
}
```

The topic will automatically be created if it does not exist

Spring makes a KafkaTemplate with the following defaults:
Server = localhost:9092
Offset = latest

Kafka producer

```
@SpringBootApplication
public class KafkaProducerApplication implements CommandLineRunner {
    @Autowired
    KafkaProducer kafkaProducer;

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

    @Override
    public void run(String... args) throws Exception {
        kafkaProducer.sendMessage();
    }
}
```

application.properties

```
1 spring.application.name=KafkaProducer
2
```

Kafka consumer

```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(String message) {  
        System.out.println(message);  
    }  
}
```

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

application.properties

1 spring.application.name=KafkaConsumer

2

Set the Kafka server

application.properties

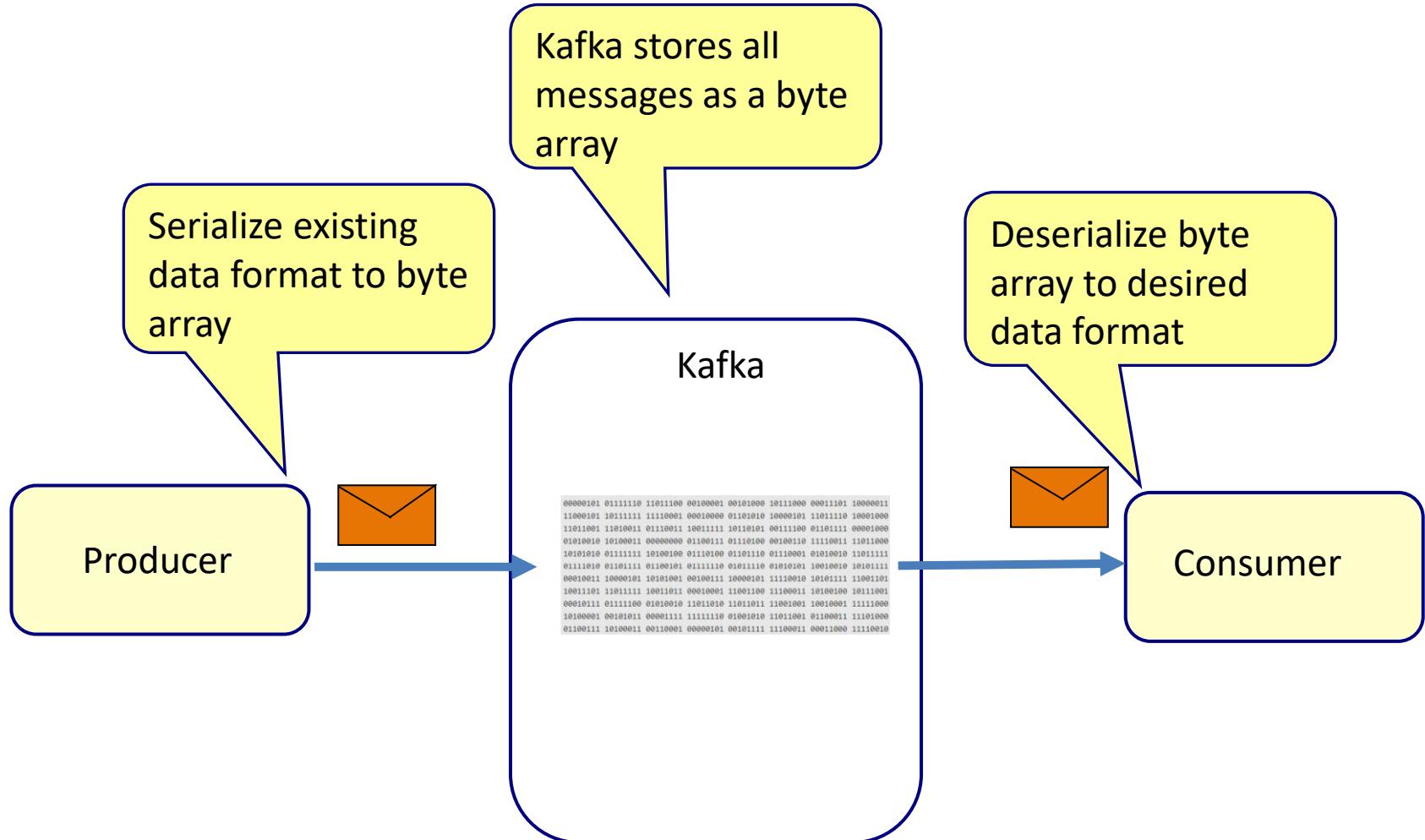
```
1 spring.application.name=KafkaProducer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 |
```

application.properties

```
1 spring.application.name=KafkaConsumer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 |
```

SERIALIZATION AND DESERIALIZATION

Serialization/deserialization



Sending an Object

```
package producer;
```

Package producer

```
public class Product {  
    private String productNumber;  
    private String name;  
    private double price;
```

```
@Service
```

```
public class KafkaProducer {
```

```
@Autowired
```

```
private KafkaTemplate<String, Object> kafkaTemplate;
```

Object

```
public void sendMessage() {
```

```
    Product product = new Product("A158", "iPhone13", 180.0);
```

```
    kafkaTemplate.send("topic1", product);
```

```
}
```

```
}
```

Sending an Object

application.properties

```
1 spring.application.name=KafkaProducer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
```

```
{  
    "Timestamp": "2025-08-23T10:42:199+00:00",  
    "Topic": "topic1",  
    "Partition": 0,  
    "Offset": 0,  
    "SchemaId": null,  
    "SchemaType": null,  
    "Key": null,  
    "Headers": {  
        "__TypeId__": "producer.Product"  
    },  
    "Message": {  
        "productNumber": "A159",  
        "name": "iPhone13",  
        "price": 180  
    }  
}
```

Kafka consumer

JsonDeserializer

application.properties

```
1 spring.application.name=KafkaConsumer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer  
5 spring.kafka.consumer.properties.spring.json.trusted.packages=*
```

Only classes in trusted packages can be used to deserialize the received JSON to an object

Kafka consumer

```
package consumer;
```

Package consumer

```
public class Product {  
    private String productNumber;  
    private String name;  
    private double price;
```

```
@Service
```

```
public class KafkaConsumer {
```

```
    @KafkaListener(topics = "topic1", groupId = "gid1")
```

```
    public void consume(Product product) {
```

```
        System.out.println(product);
```

```
}
```

```
}
```

The consume() method is not called because we receive a **producer.Product**

Kafka consumer

```
package consumer;
```

Package consumer

```
public class Product {  
    private String productNumber;  
    private String name;  
    private double price;
```

```
{  
    "Timestamp": "2025-08-23T10:42:42.199+00:00",  
    "Topic": "topic1",  
    "Partition": 0,  
    "Offset": 0,  
    "SchemaId": null,  
    "SchemaType": null,  
    "Key": null,  
    "Headers": {  
        "__TypeId__": "producer.Product"  
    },  
    "Message": {  
        "productNumber": "A159",  
        "name": "iPhone13",  
        "price": 180  
    }  
}
```

We receive a **producer.Product** but we have only a **consumer.Product**. This means Spring will not call the listener method.

Possible solution: Producer

application.properties

```
1 spring.application.name=KafkaProducer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer  
5 spring.kafka.producer.properties[spring.json.add.type.headers]=false  
6
```

```
{  
    "Timestamp": "2025-08-23T10:57:35.607+00:00",  
    "Topic": "topic1",  
    "Partition": 0,  
    "Offset": 1,  
    "SchemaId": null,  
    "SchemaType": null,  
    "Key": null,  
    "Headers": null,  
    "Message": {  
        "productNumber": "A159",  
        "name": "iPhone13",  
        "price": 180  
    }  
}
```

Do not add the class type in the message.

No class type in the message.

Possible solution: Consumer

```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic1", groupId = "gid1", properties =  
        {"spring.json.value.default.type=consumer.Product"})  
    public void consume(Product product) {  
        System.out.println(product);  
    }  
}
```

Specify the type Spring needs to deserialize the received JSON to

```
application.properties  
spring.application.name=KafkaConsumer  
spring.kafka.bootstrap-servers=localhost:9092  
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer  
spring.kafka.consumer.properties.spring.json.trusted.packages=*
```

Better solution: Producer

```
@Service  
public class KafkaProducer {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() throws JsonProcessingException {  
        Product product = new Product("A159", "iPhone13", 180.0);  
        ObjectMapper objectMapper = new ObjectMapper();  
        String productAsString = objectMapper.writeValueAsString(product);  
        kafkaTemplate.send("topic1", productAsString);  
    }  
}
```

Always send a String

application.properties

```
1 spring.application.name=KafkaProducer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092
```

Convert the object
to a JSON string

Better solution: Consumer

```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(String productAsString) {  
        ObjectMapper objectMapper = new ObjectMapper();  
        try {  
            Product product = objectMapper.readValue(productAsString, Product.class);  
            System.out.println("Kafka receiver received message:" + product);  
        } catch (IOException e) {  
            System.out.println("Kafka receiver: Cannot convert : " + productAsString+ " to a  
Product object");  
        }  
    }  
}
```

Always receive a String

Convert the JSON string to an object

```
application.properties  
1 spring.application.name=KafkaConsumer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092
```

JAVA CONFIG

Producer configuration

```
@Configuration
public class KafkaProducerConfig {

    @Bean
    public ProducerFactory<String, String> producerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                        StringSerializer.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

Configure Kafka in Java instead of applications.properties

Consumer configuration

```
@Configuration  
public class KafkaConsumerConfig {  
  
    @Bean  
    public ConsumerFactory<String, String> consumerFactory() {  
        Map<String, Object> configProps = new HashMap<>();  
        configProps.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");  
        configProps.put(ConsumerConfig.GROUP_ID_CONFIG, "group_id");  
        configProps.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);  
        configProps.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);  
        return new DefaultKafkaConsumerFactory<>(configProps);  
    }  
  
    @Bean  
    public ConcurrentKafkaListenerContainerFactory<String, String> kafkaListenerContainerFactory() {  
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new  
            ConcurrentKafkaListenerContainerFactory<>();  
        factory.setConsumerFactory(consumerFactory());  
        return factory;  
    }  
}
```

Configure Kafka in Java instead of applications.properties

HEADERS

Custom headers: producer

```
public class KafkaProducer {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() {  
        Message<String> message = MessageBuilder.withPayload("Hello World")  
            .setHeader(KafkaHeaders.TOPIC, "topic1")  
            .setHeader("MyCustomHeader", "HeaderValue")  
            .build();  
        kafkaTemplate.send(message);  
    }  
}
```

Add a custom header

Custom header

```
{  
    "Timestamp": "2025-08-25T08:58:34.829+00:00",  
    "Topic": "topic1",  
    "Partition": 0,  
    "Offset": 9,  
    "SchemaId": null,  
    "SchemaType": null,  
    "Key": null,  
    "Headers": {  
        "MyCustomHeader": "HeaderValue",  
        "spring_json_header_types": "{\"MyCustomHeader\":\"java.lang.String\""}  
    },  
    "Message": "Hello World"  
}
```

Custom header.

Custom headers: consumer

```
@Service
public class KafkaConsumer {

    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(@Payload String message, @Header(name =
        "MyCustomHeader", required = false) String customHeaderValue) {
        System.out.println("Message header MyCustomHeader = "+customHeaderValue);
        System.out.println("Message payload = "+message);
    }
}
```

Message header MyCustomHeader = HeaderValue
Message payload = Hello World

Receiving a Message

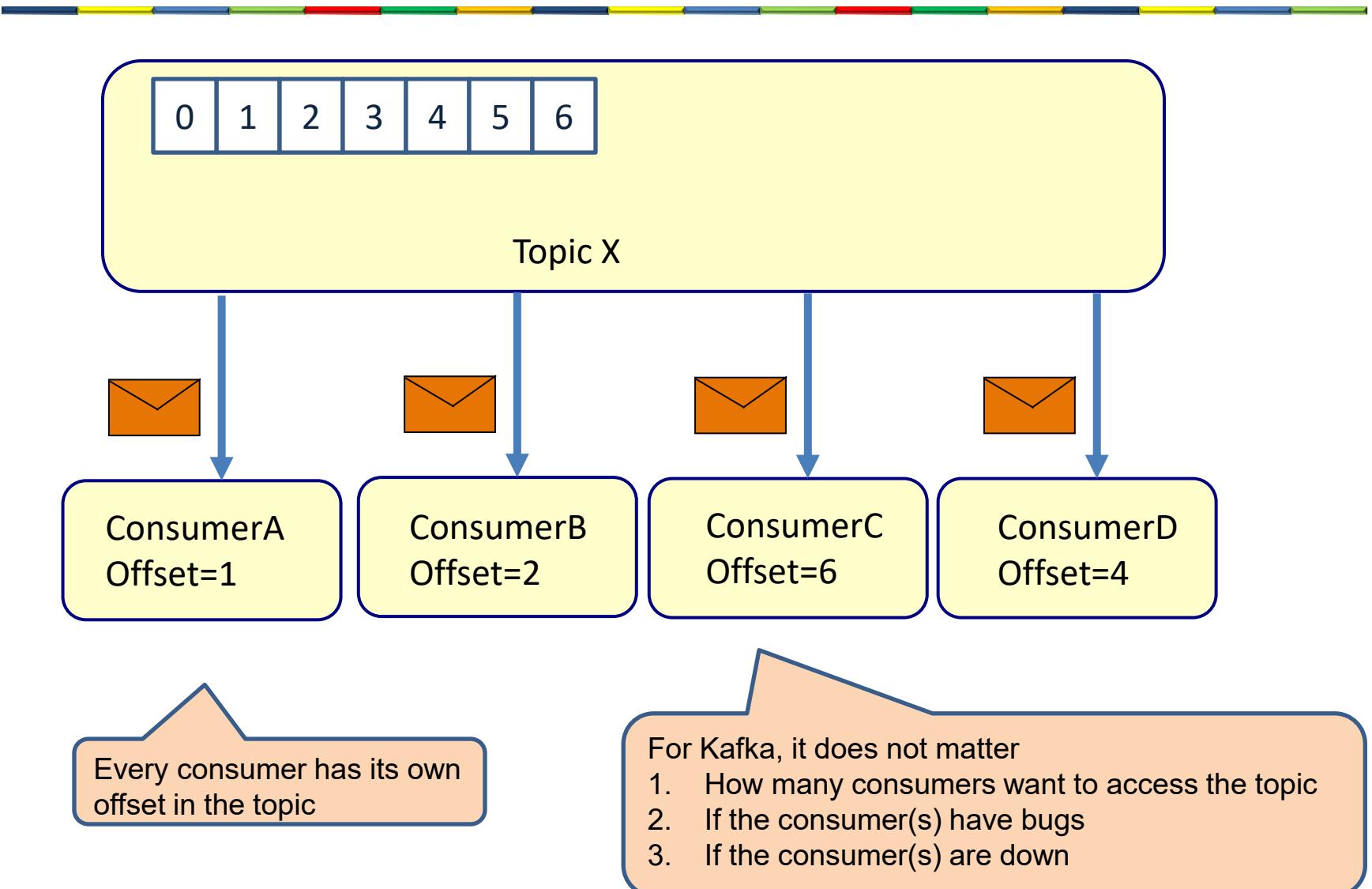
```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(Message<String> message) {  
        MessageHeaders headers = message.getHeaders();  
        System.out.println("Message header MyCustomHeader = "  
                           +headers.get("MyCustomHeader"));  
        System.out.println("Message payload = "+message.getPayload());  
    }  
}
```

Receive a Message

Message header MyCustomHeader = HeaderValue
Message payload = Hello World

OFFSET

Offset



Offset

```
{  
    "Timestamp": "2025-08-25T09:34:11.608+00:00",  
    "Topic": "topic1",  
    "Partition": 0,  
    "Offset": 13, The offset header  
    "SchemaId": null,  
    "SchemaType": null,  
    "Key": null,  
    "Headers": null,  
    "Message": "Hello World"  
},  
{  
    "Timestamp": "2025-08-25T09:35:21.386+00:00",  
    "Topic": "topic1",  
    "Partition": 0,  
    "Offset": 14, The offset header  
    "SchemaId": null,  
    "SchemaType": null,  
    "Key": null,  
    "Headers": null,  
    "Message": "Hello World"  
}
```

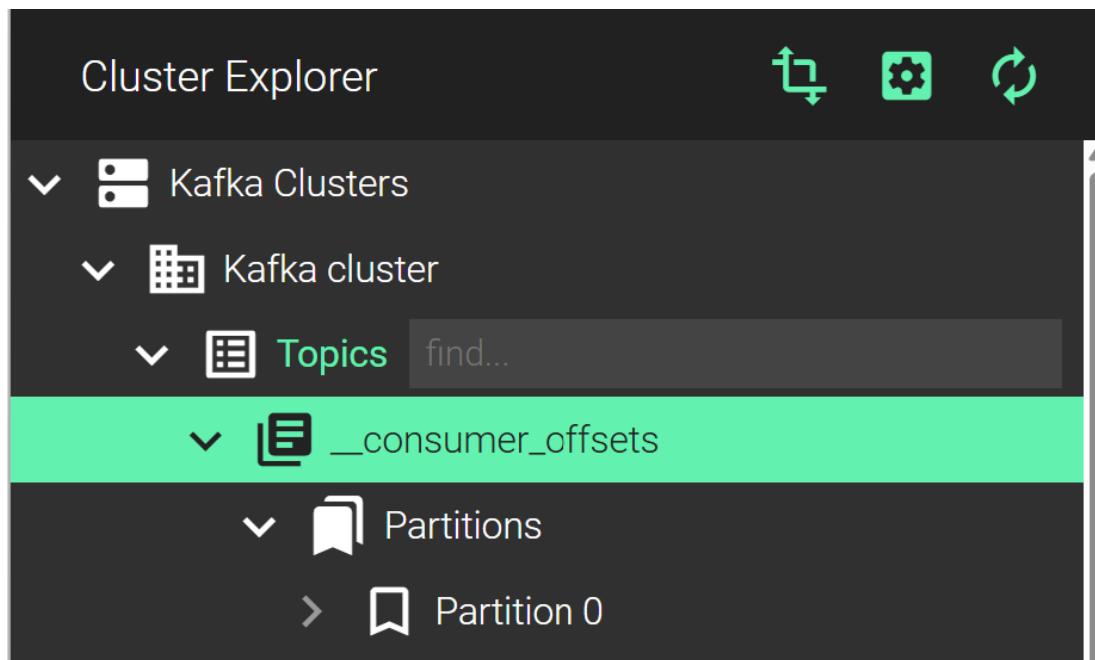
Offset

```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(@Payload String message,  
                        @Header(KafkaHeaders.OFFSET) long offset) {  
        System.out.println("Message offset = "+offset);  
        System.out.println("Message payload = "+message);  
    }  
}
```

Get the offset header

```
Message offset = 13  
Message payload = Hello World  
Message offset = 14  
Message payload = Hello World
```

Offset is stored within Kafka



Offset: latest message

@Service

```
public class KafkaConsumer {
```

```
    @KafkaListener(topics = "topic1", groupId = "gid1", properties = {"auto.offset.reset:latest"})  
    public void consume(String message, @Header(KafkaHeaders.OFFSET) long offset) {  
        System.out.println("Message received = " + message + ", offset= " + offset);  
    }
```

```
}
```

```
Message received = Hello World, offset= 16  
Message received = Hello World, offset= 17
```

Get the latest message

Kafka broker

Topic partition



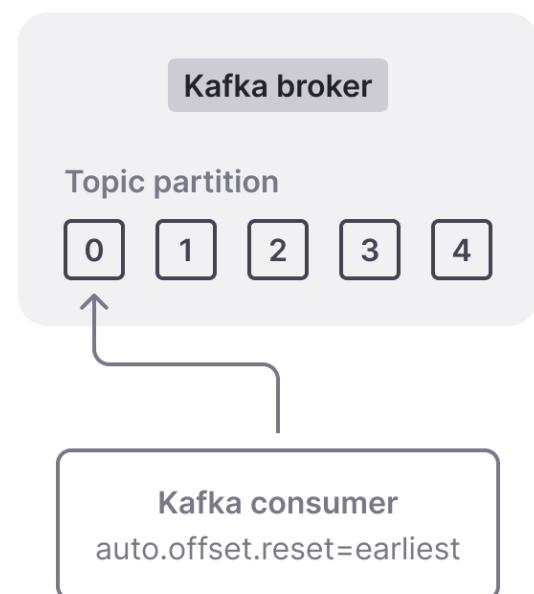
Kafka consumer
auto.offset.reset=latest

Offset: earliest message

Get the earliest message

```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic1", groupId = "gid2", properties = {"auto.offset.reset:earliest"})  
    public void consume(String message, @Header(KafkaHeaders.OFFSET) long offset) {  
        System.out.println("Message received = "+message+", offset= "+offset);  
    }  
}
```

Message received = Hello World, offset= 1
Message received = Hello World, offset= 2



Set auto.offset.reset in application.properties



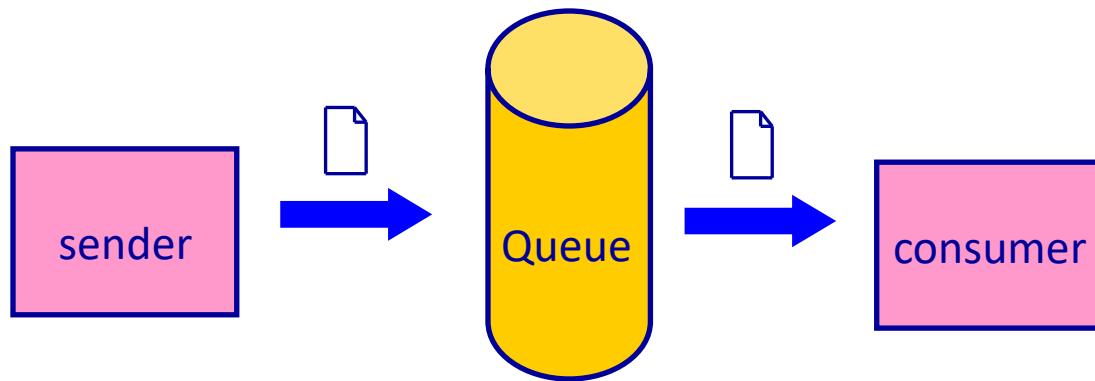
application.properties

```
spring.kafka.bootstrap-servers=localhost:9092  
spring.kafka.consumer.group-id= gid  
spring.kafka.consumer.auto-offset-reset= earliest
```

GROUPID

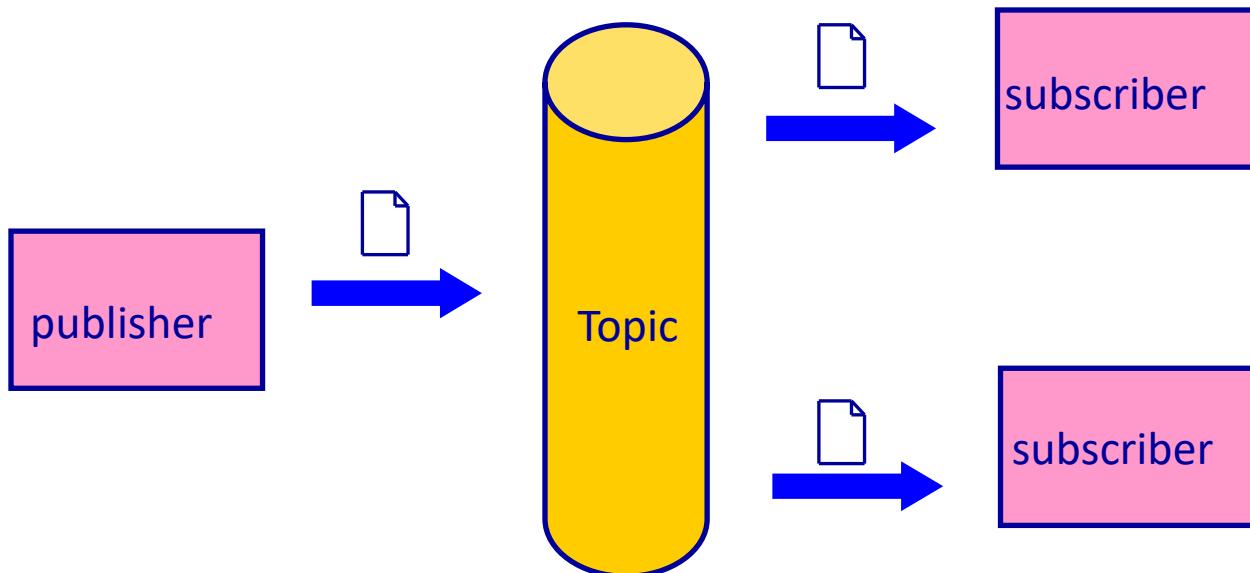
Point-To-Point (PTP)

- A dedicated consumer per Queue message



Publish-Subscribe (Pub-Sub)

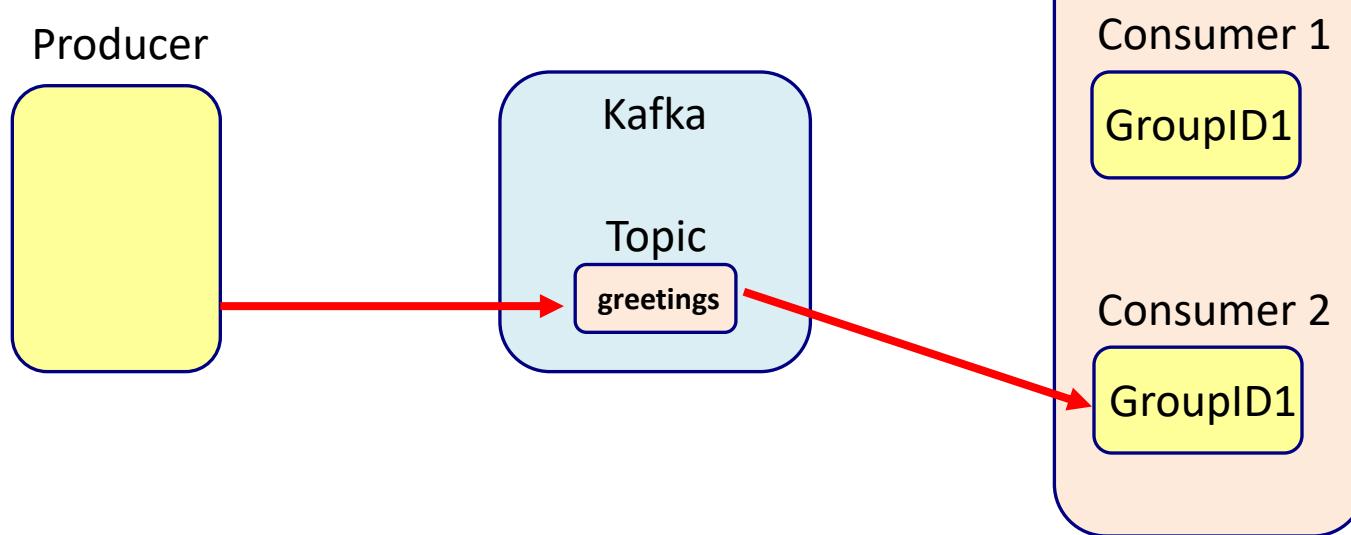
- A message channel can have more than one '*consumer*'
 - Ideal for broadcasting



Point to point

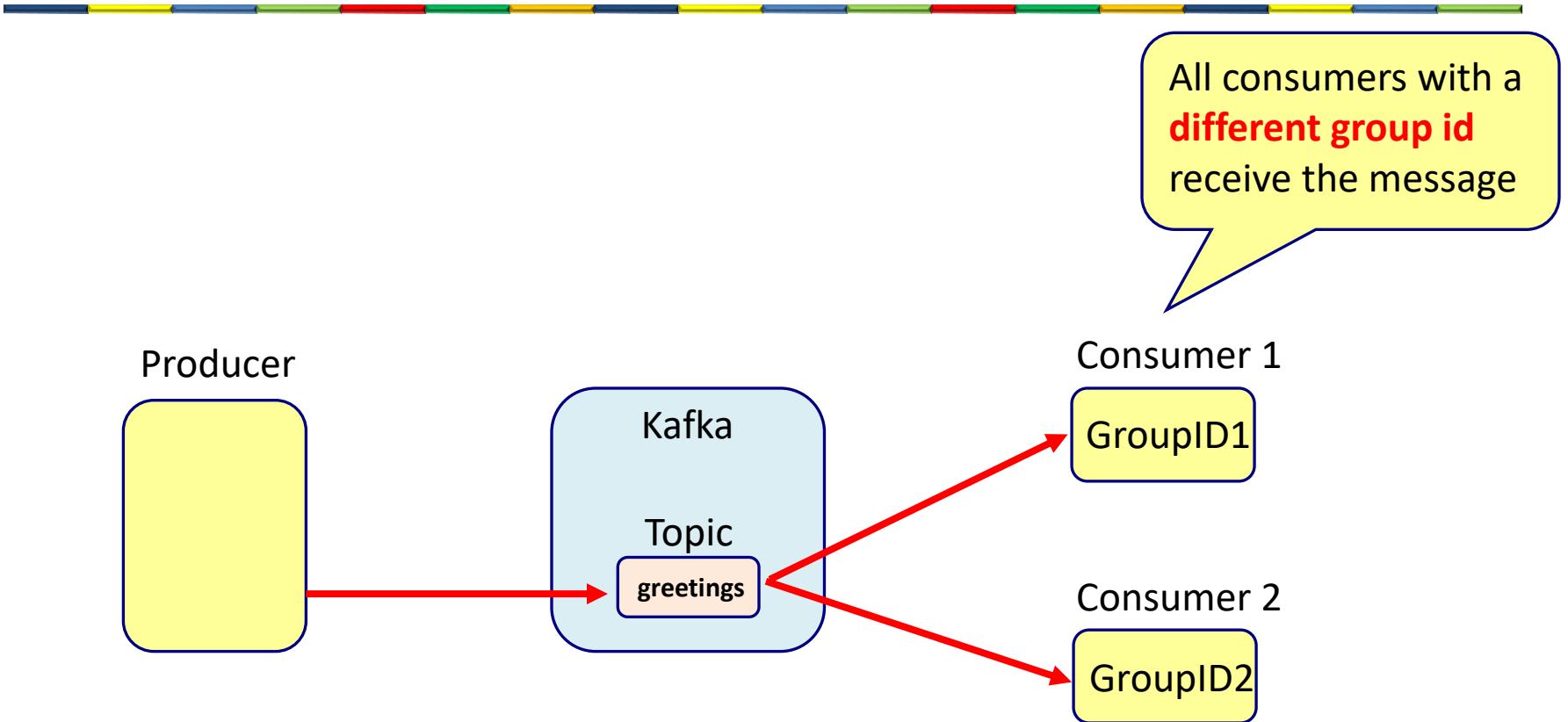
- Competing consumers

Consumer group: all consumers have the **same group id**



- Only one consumers receives the message

Publish-Subscribe



Multiple listeners

```
@Service
```

```
public class KafkaConsumer {
```

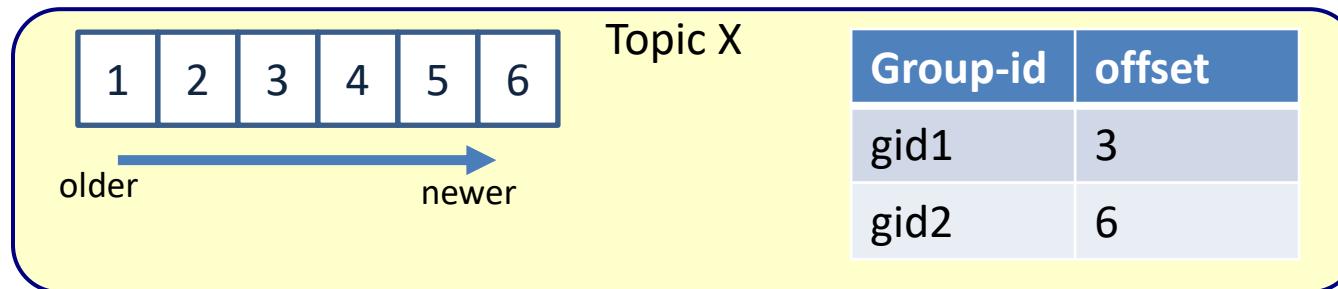
```
    @KafkaListener(topics = "topic1", groupId = "gid1", properties = {"auto.offset.reset:earliest"})
    public void consume(String message, @Header(KafkaHeaders.OFFSET) long offset) {
        System.out.println("Message received 1= "+message+", offset= "+offset);
    }
```

```
    @KafkaListener(topics = "topic1", groupId = "gid1", properties = {"auto.offset.reset:earliest"})
    public void consume2(String message, @Header(KafkaHeaders.OFFSET) long offset) {
        System.out.println("Message received 2= "+message+", offset= "+offset);
    }
```

```
    @KafkaListener(topics = "topic1", groupId = "gid5", properties = {"auto.offset.reset:earliest"})
    public void consume3(String message, @Header(KafkaHeaders.OFFSET) long offset) {
        System.out.println("Message received 3= "+message+", offset= "+offset);
    }
```

Offset and group-id

- Offset is stored per group-id



A consumer connects to topic X in Kafka

Group-id	auto.offset.reset	Message received
gid1	earliest	4
gid1	latest	7
gid2	earliest	7
gid2	latest	7
gid3	earliest	1
gid3	latest	7

MESSAGES

Sending a message with KafkaTemplate

```
CompletableFuture<SendResult<K, V>> sendDefault(V data);

CompletableFuture<SendResult<K, V>> sendDefault(K key, V data);

CompletableFuture<SendResult<K, V>> sendDefault(Integer partition, K key, V data);

CompletableFuture<SendResult<K, V>> sendDefault(Integer partition, Long timestamp, K key, V data);

CompletableFuture<SendResult<K, V>> send(String topic, V data);

CompletableFuture<SendResult<K, V>> send(String topic, K key, V data);

CompletableFuture<SendResult<K, V>> send(String topic, Integer partition, K key, V data);

CompletableFuture<SendResult<K, V>> send(String topic, Integer partition, Long timestamp, K key, V data);

CompletableFuture<SendResult<K, V>> send(ProducerRecord<K, V> record);

CompletableFuture<SendResult<K, V>> send(Message<?> message);
```

Sending a message

```
public void sendMessage() {  
    Product product = new Product("A158", "iPhone13", 180.0);  
    kafkaTemplate.send("product_topic", product);  
}
```

Sending an object

```
public void sendMessage() {  
    ProducerRecord<String, String> producerRecord = new ProducerRecord<>("topic1", "key",  
        "Hello World");  
    producerRecord.headers().add("MyCustomHeader", "HeaderValue".getBytes());  
    kafkaTemplate.send(producerRecord);  
}
```

Sending a ProducerRecord

```
public void sendMessage() {  
    Product product = new Product("A158", "iPhone13", 180.0);  
    Message<Product> message = MessageBuilder.withPayload(product)  
        .setHeader("kafka_topic", "product_topic") // Spring uses this header for the topic  
        .build();  
    kafkaTemplate.send(message);  
}
```

Sending a Message

Receiving a message

```
@KafkaListener(topics = "product_topic", groupId = "gid1")
public void consume(Product product) {
    System.out.println(product);
}
```

Consume an object

```
@KafkaListener(topics = "topic1", groupId = "gid1")
public void consume(ConsumerRecord<String, String> consumerRecord) {
    System.out.println("Message received = "+consumerRecord.value()+ ", offset= "
        +consumerRecord.offset());
    Headers consumedHeaders = consumerRecord.headers();
    for (Header header : consumedHeaders) {
        System.out.println("Header key="+header.key()+", value = "+new String(header.value()))
    }
}
```

Consume a
ConsumerRecord

Receiving a message

```
@KafkaListener(topics = "topic1", groupId = "gid1")
public void consume(Message<String> message) {
    System.out.println("Message received = "+message.getPayload());
    MessageHeaders consumedHeaders = message.getHeaders();
    System.out.println("offset="+consumedHeaders.get("kafka_offset"));
    System.out.println("group id="+consumedHeaders.get("kafka_groupId"));
}
```

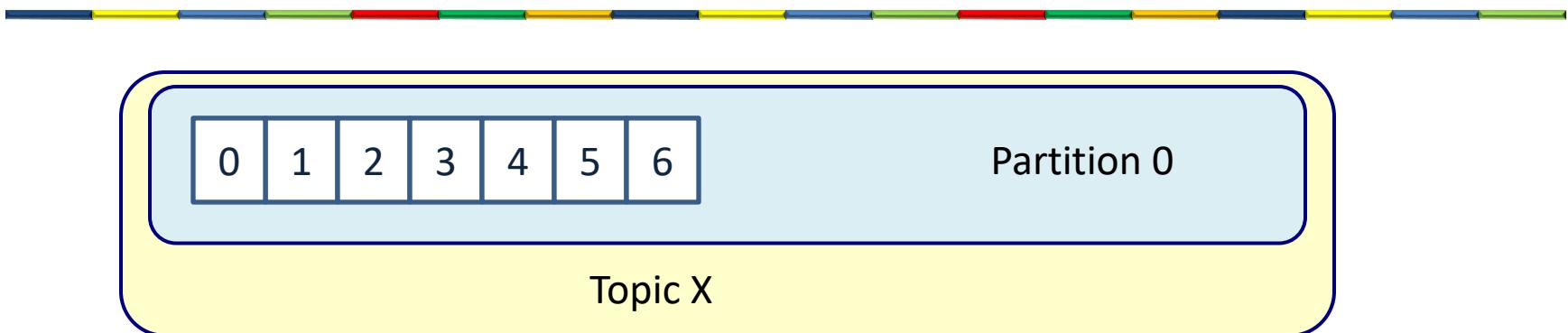
Consume a Message

PARTITIONS

Partition

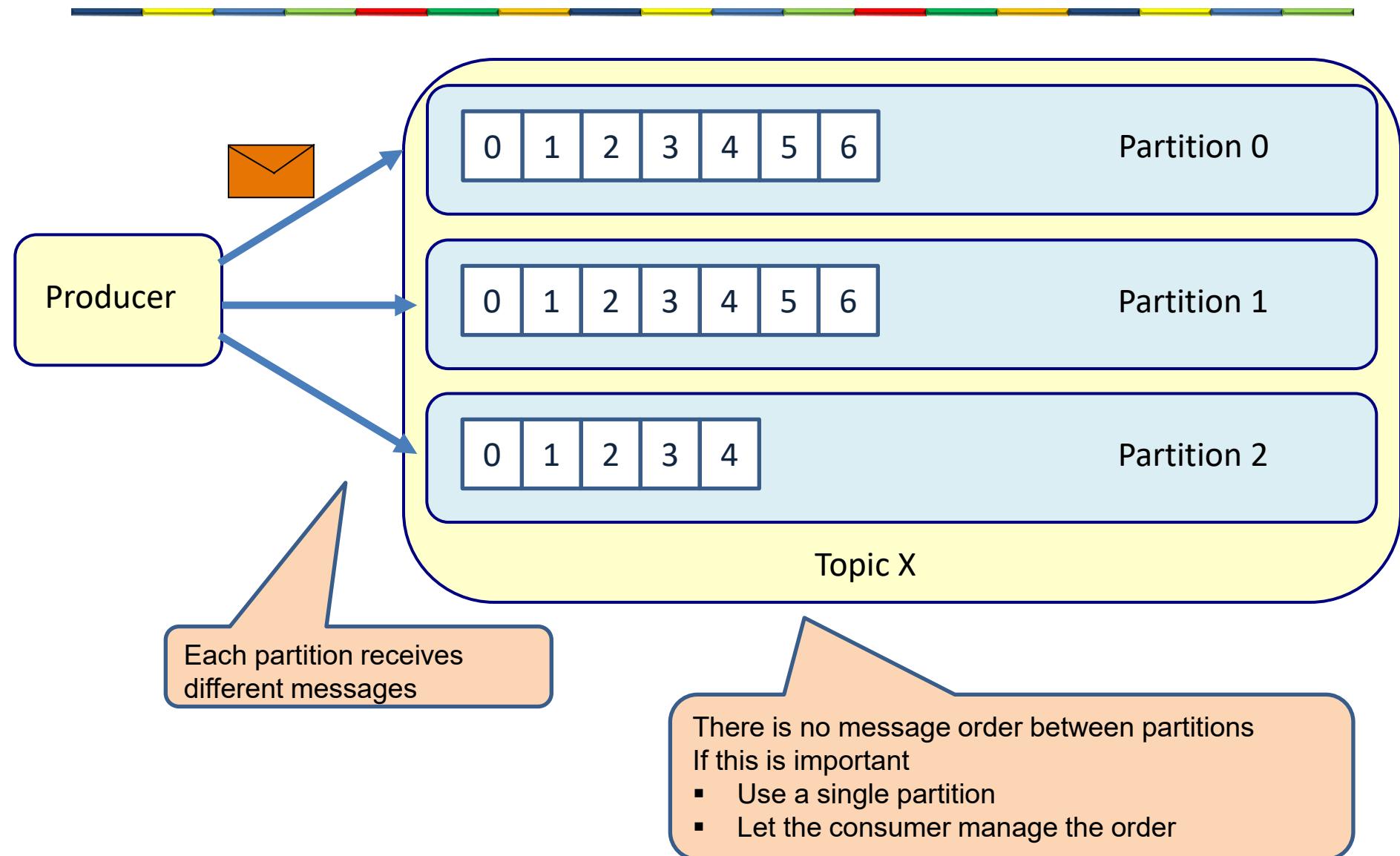
- Each topic has one or more partitions
 - This is configurable
- Each partition is maintained on 1 or more brokers

1 partition

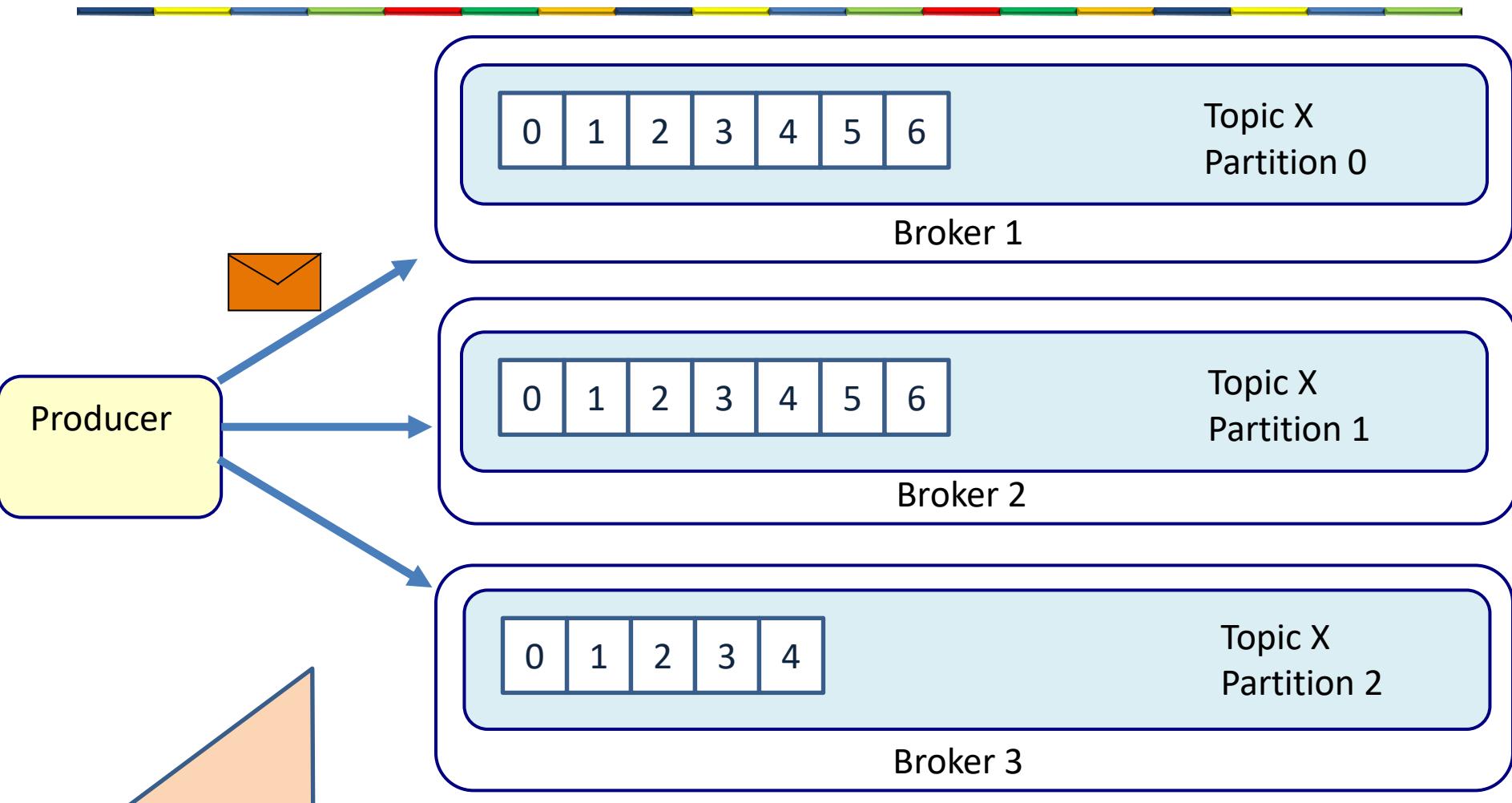


- Each partition must fit on 1 broker

3 partitions



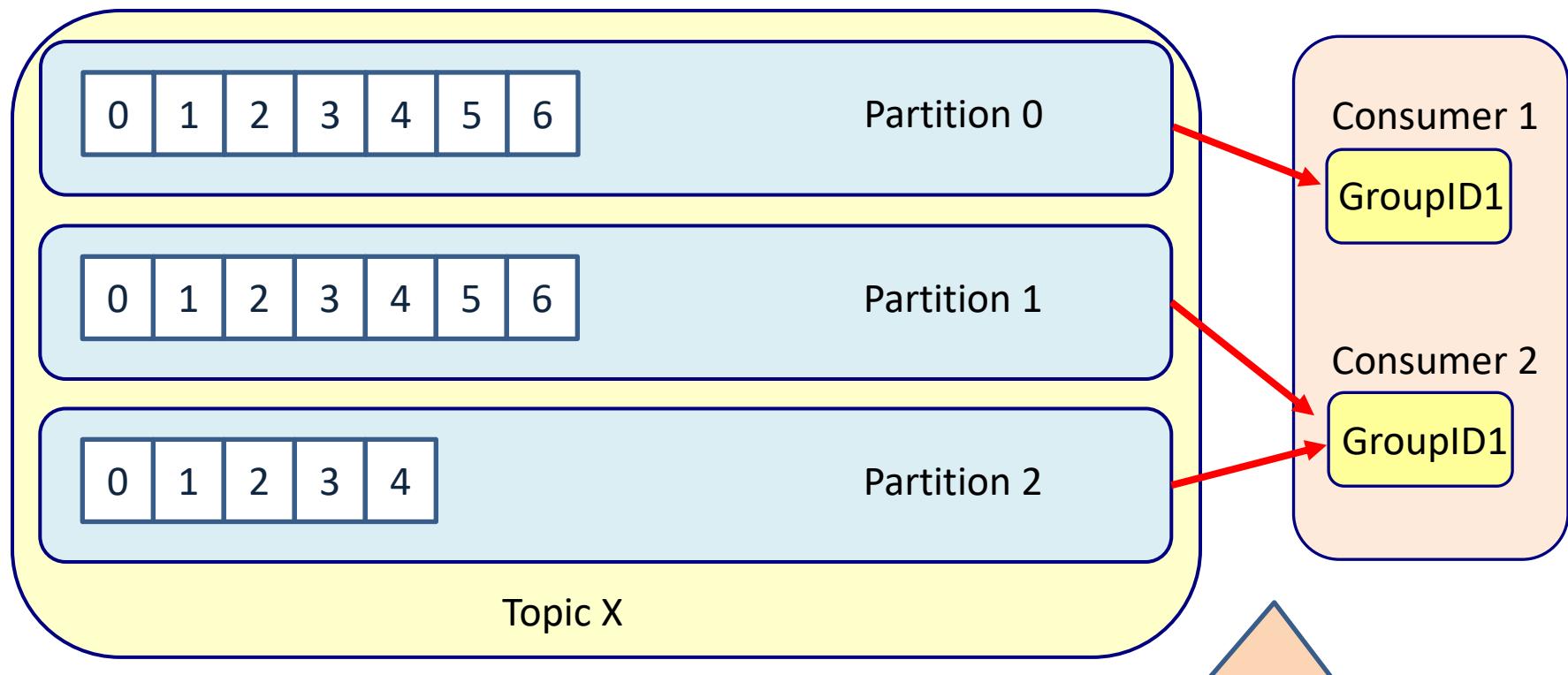
Scale out partitions



Increases

- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

Consumer groups



Create a topic with 3 partitions

```
@SpringBootApplication
public class KafkaProducerApplication implements CommandLineRunner {
    @Autowired
    KafkaProducer kafkaProducer;

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

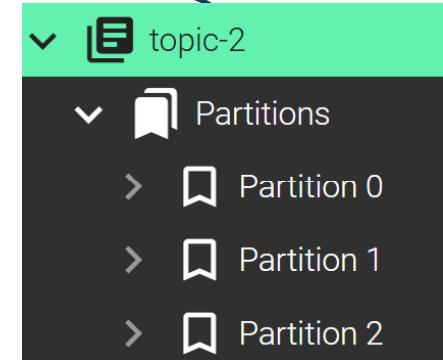
    @Override
    public void run(String... args) throws Exception {
        kafkaProducer.sendMessage();
    }

    @Bean
    public NewTopic createtopic2() {
        return TopicBuilder.name("topic-2").partitions(3).build();
    }
}
```

Create a topic with 3 partitions

Producer

```
public class KafkaProducer {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() {  
        for (int x=1; x<11 ; x++){  
            kafkaTemplate.send("topic-2", "Message-"+x);  
        }  
    }  
}
```



All messages go to 1 partition

```
[{"Timestamp": "2025-08-25T13:38:46.093+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 0, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 1, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 2, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 3, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 4, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 5, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 6, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 7, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 8, "Sche  
{"Timestamp": "2025-08-25T13:38:46.103+00:00", "Topic": "topic-2", "Partition": 1, "Offset": 9, "Sche
```

Consumer

```
@Service
public class KafkaConsumer {

    @KafkaListener(topics = "topic-2", groupId = "gid1", properties =
        {"auto.offset.reset:earliest"})
    public void consume(String message,
        @Header(KafkaHeaders.OFFSET) long offset,
        @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
        System.out.println("Message received 1= "+message+", offset= "+offset+", partition= "+partition);
    }
}
```

Consumer is connected to all 3 partitions

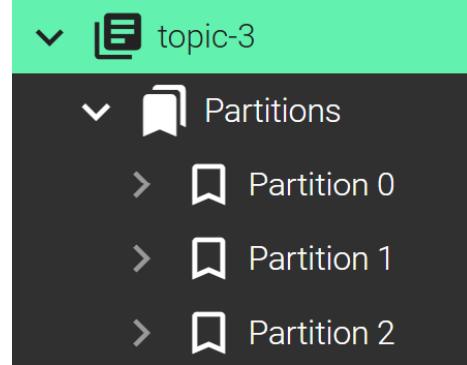
Message received 1= Message-1, offset= 12, partition= 1
Message received 1= Message-2, offset= 13, partition= 1
Message received 1= Message-3, offset= 14, partition= 1
Message received 1= Message-4, offset= 15, partition= 1
Message received 1= Message-5, offset= 16, partition= 1
Message received 1= Message-6, offset= 17, partition= 1
Message received 1= Message-7, offset= 18, partition= 1
Message received 1= Message-8, offset= 19, partition= 1
Message received 1= Message-9, offset= 20, partition= 1
Message received 1= Message-10, offset= 21, partition= 1

All messages come from the same partition

Producer

```
public class KafkaProducer {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() {  
        for (int x=1; x<11 ; x++){  
            kafkaTemplate.send("topic-3","key-"+x , "Message-"+x);  
        }  
    }  
}
```

Every message has an unique key



Messages are distributed over the 3 partitions

```
[{"Timestamp": "2025-08-25T13:49:36.463+00:00", "Topic": "topic-3", "Partition": 0, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 49], "Headers": null, "Message": "Message-1"}, {"Timestamp": "2025-08-25T13:49:36.482+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 50], "Headers": null, "Message": "Message-2"}, {"Timestamp": "2025-08-25T13:49:36.482+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 51], "Headers": null, "Message": "Message-3"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 1, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 51], "Headers": null, "Message": "Message-7"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 0, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 55], "Headers": null, "Message": "Message-3"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 0, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 52], "Headers": null, "Message": "Message-4"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 1, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 56], "Headers": null, "Message": "Message-8"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 0, "Offset": 2, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 57], "Headers": null, "Message": "Message-9"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 2, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 53], "Headers": null, "Message": "Message-5"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 3, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 54], "Headers": null, "Message": "Message-6"}, {"Timestamp": "2025-08-25T13:49:36.48300:00", "Topic": "topic-3", "Partition": 2, "Offset": 4, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 49, 48], "Headers": null, "Message": "Message-10"}]
```

All messages have an unique key

- Partition 0

```
[{"Timestamp": "2025-08-25T13:49:36.463+00:00", "Topic": "topic-3", "Partition": 0, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 49], "Headers": null, "Message": "Message-1"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 0, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 52], "Headers": null, "Message": "Message-4"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 0, "Offset": 2, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 57], "Headers": null, "Message": "Message-9"}]
```

- Partition 1

```
[{"Timestamp": "2025-08-25T13:49:36.482+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 50], "Headers": null, "Message": "Message-2"}, {"Timestamp": "2025-08-25T13:49:36.482+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 51], "Headers": null, "Message": "Message-3"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 2, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 53], "Headers": null, "Message": "Message-5"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 2, "Offset": 3, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 54], "Headers": null, "Message": "Message-6"}, {"Timestamp": "2025-08-25T13:49:36.48300:00", "Topic": "topic-3", "Partition": 2, "Offset": 4, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 49, 48], "Headers": null, "Message": "Message-10"}]
```

- Partition 2

```
[{"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 1, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 55], "Headers": null, "Message": "Message-7"}, {"Timestamp": "2025-08-25T13:49:36.483+00:00", "Topic": "topic-3", "Partition": 1, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121, 45, 56], "Headers": null, "Message": "Message-8"}]
```

Consumer

```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic-3", groupId = "gid1", properties =  
        {"auto.offset.reset:latest"})  
  
    public void consume(String message,  
        @Header(KafkaHeaders.OFFSET) long offset,  
        @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {  
        System.out.println("Message received 1= "+message+", offset= "+offset+", partition= "+partition);  
    }  
}
```

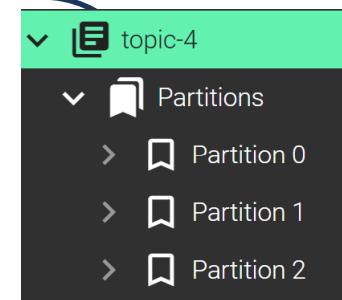
Consumer is connected to all 3 partitions

```
Message received 1= Message-1, offset= 5, partition= 2  
Message received 1= Message-2, offset= 6, partition= 2  
Message received 1= Message-9, offset= 7, partition= 2  
Message received 1= Message-3, offset= 2, partition= 1  
Message received 1= Message-5, offset= 3, partition= 1  
Message received 1= Message-8, offset= 4, partition= 1  
Message received 1= Message-4, offset= 3, partition= 0  
Message received 1= Message-6, offset= 4, partition= 0  
Message received 1= Message-7, offset= 5, partition= 0  
Message received 1= Message-10, offset= 6, partition= 0
```

All messages come from different partition

Producer

```
public class KafkaProducer {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() {  
        for (int x=1; x<11 ; x++){  
            kafkaTemplate.send("topic-4","key" , "Message-"+x);  
        }  
    }  
}
```



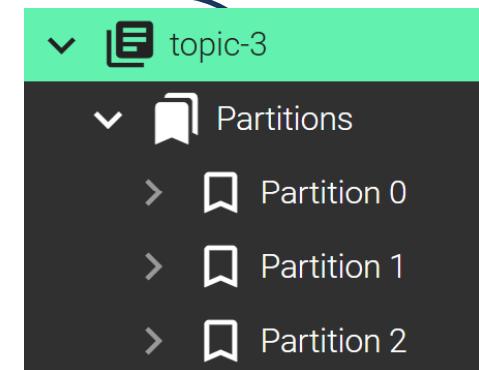
All messages have
the same key

All messages go to 1
partition

```
[{"Timestamp": "2025-08-25T14:02:59.22+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-1"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 1, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-2"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 2, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-3"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 3, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-4"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 4, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-5"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 5, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-6"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 6, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-7"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 7, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-8"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 8, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-9"}, {"Timestamp": "2025-08-25T14:02:59.232+00:00", "Topic": "topic-4", "Partition": 1, "Offset": 9, "SchemaId": null, "SchemaType": null, "Key": [107, 101, 121], "Headers": null, "Message": "Message-10"}]
```

Producer

```
public class KafkaProducer {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() {  
        for (int x=1; x<11 ; x++){  
            kafkaTemplate.send("topic-3","key-"+x , "Message-"+x);  
        }  
    }  
}
```



Every message has
an unique key

Connect listener to different partitions

```
@Service
```

```
public class KafkaConsumer {
```

```
    @KafkaListener(groupId = "gid1", properties = {"auto.offset.reset:latest"}, topicPartitions =  
        { @TopicPartition(topic = "topic-3", partitions = { "0" }) })
```

```
    public void consume(String message,
```

```
        @Header(KafkaHeaders.OFFSET) long offset,
```

```
        @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
```

```
        System.out.println("Listener 1: Message received = "+message+ ", offset= "+offset+ ",  
partition= "+partition);
```

```
}
```

```
    @KafkaListener(groupId = "gid1", properties = {"auto.offset.reset:latest"}, topicPartitions =  
        { @TopicPartition(topic = "topic-3", partitions = { "1" }) })
```

```
    public void consume2(String message,
```

```
        @Header(KafkaHeaders.OFFSET) long offset,
```

```
        @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
```

```
        System.out.println("Listener 2: Message received = "+message+ ", offset= "+offset+ ",  
partition= "+partition);
```

```
}
```

Connect listener to different partitions

```
@KafkaListener(groupId = "gid1", properties = {"auto.offset.reset:latest"}, topicPartitions =  
    { @TopicPartition(topic = "topic-3", partitions = { "2" }) })  
public void consume3(String message,  
    @Header(KafkaHeaders.OFFSET) long offset,  
    @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {  
    System.out.println("Listener 3: Message received = "+message+", offset= "+offset+",  
partition= "+partition);  
}  
}
```

Consumer

Listener 1: Message received = Message-4, offset= 15, partition= 0
Listener 3: Message received = Message-1, offset= 14, partition= 2
Listener 2: Message received = Message-3, offset= 11, partition= 1
Listener 2: Message received = Message-5, offset= 12, partition= 1
Listener 1: Message received = Message-6, offset= 16, partition= 0
Listener 2: Message received = Message-8, offset= 13, partition= 1
Listener 3: Message received = Message-2, offset= 15, partition= 2
Listener 1: Message received = Message-7, offset= 17, partition= 0
Listener 3: Message received = Message-9, offset= 16, partition= 2
Listener 1: Message received = Message-10, offset= 18, partition= 0

Different listener for every partition

concurrency

```
@Service  
public class KafkaConsumer {  
  
    @KafkaListener(topics = "topic-3", groupId = "gid1", properties =  
    {"auto.offset.reset.latest"}, concurrency = "3")  
    public void consume(String message,  
        @Header(KafkaHeaders.OFFSET) long offset,  
        @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {  
        System.out.println(Thread.currentThread().getName()+"Message received =  
        "+message+", offset= "+offset+", partition= "+partition);  
    }  
}
```

concurrency = 3. Create 3 listeners, one for every partition

concurrency



```
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-4,  
offset= 19, partition= 0  
org.springframework.kafka.KafkaListenerEndpointContainer#0-2-C-1Message received = Message-1,  
offset= 17, partition= 2  
org.springframework.kafka.KafkaListenerEndpointContainer#0-1-C-1Message received = Message-3,  
offset= 14, partition= 1  
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-6,  
offset= 20, partition= 0  
org.springframework.kafka.KafkaListenerEndpointContainer#0-1-C-1Message received = Message-5,  
offset= 15, partition= 1  
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-7,  
offset= 21, partition= 0  
org.springframework.kafka.KafkaListenerEndpointContainer#0-2-C-1Message received = Message-2,  
offset= 18, partition= 2  
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-10,  
offset= 22, partition= 0  
org.springframework.kafka.KafkaListenerEndpointContainer#0-1-C-1Message received = Message-8,  
offset= 16, partition= 1  
org.springframework.kafka.KafkaListenerEndpointContainer#0-2-C-1Message received = Message-9,  
offset= 19, partition= 2
```

3 different listeners, one for every partition

ERROR HANDLING

Producer errors

```
@Service
```

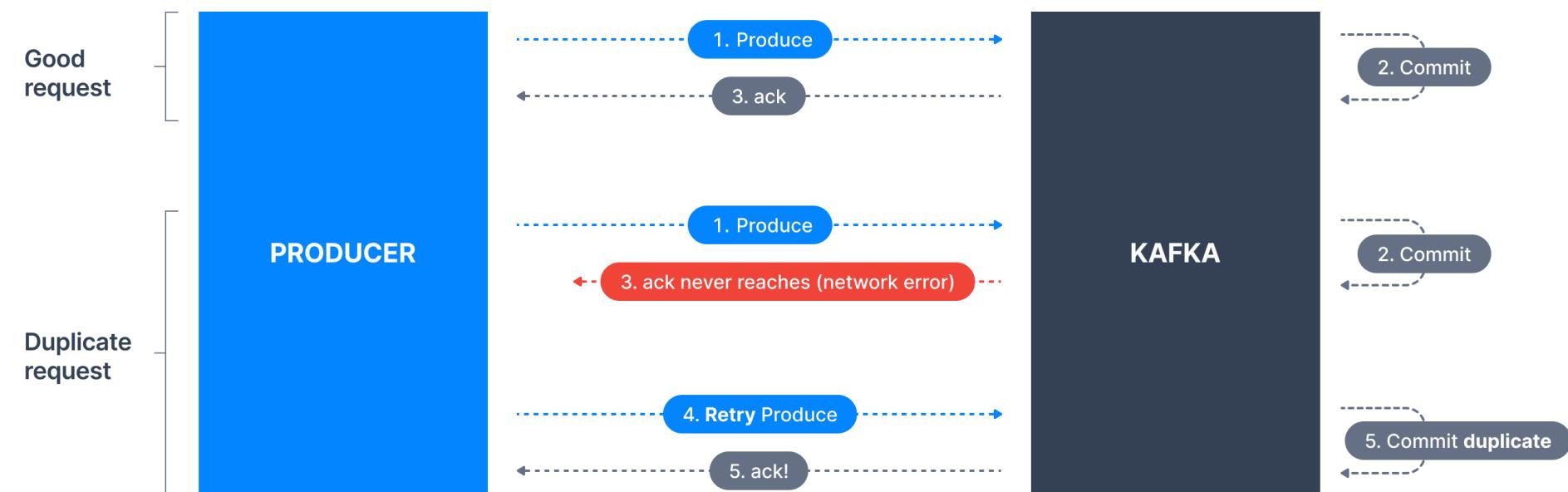
```
public class KafkaProducer {  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() {  
  
        CompletableFuture<SendResult<String, String>> future2 = kafkaTemplate.send("topic1",  
            "Hello");  
        future2.whenComplete((result, sendException) -> {  
            if (sendException == null) {  
                System.out.println("Sent Message= Hello to topic= topic1 with offset= " +  
                    result.getRecordMetadata().offset());  
            } else {  
                System.out.println("Unable to send Message= Hello World due to " +  
                    sendException.getMessage());  
            }  
        });  
    }  
}
```

Check CompletableFuture if
there was an exception

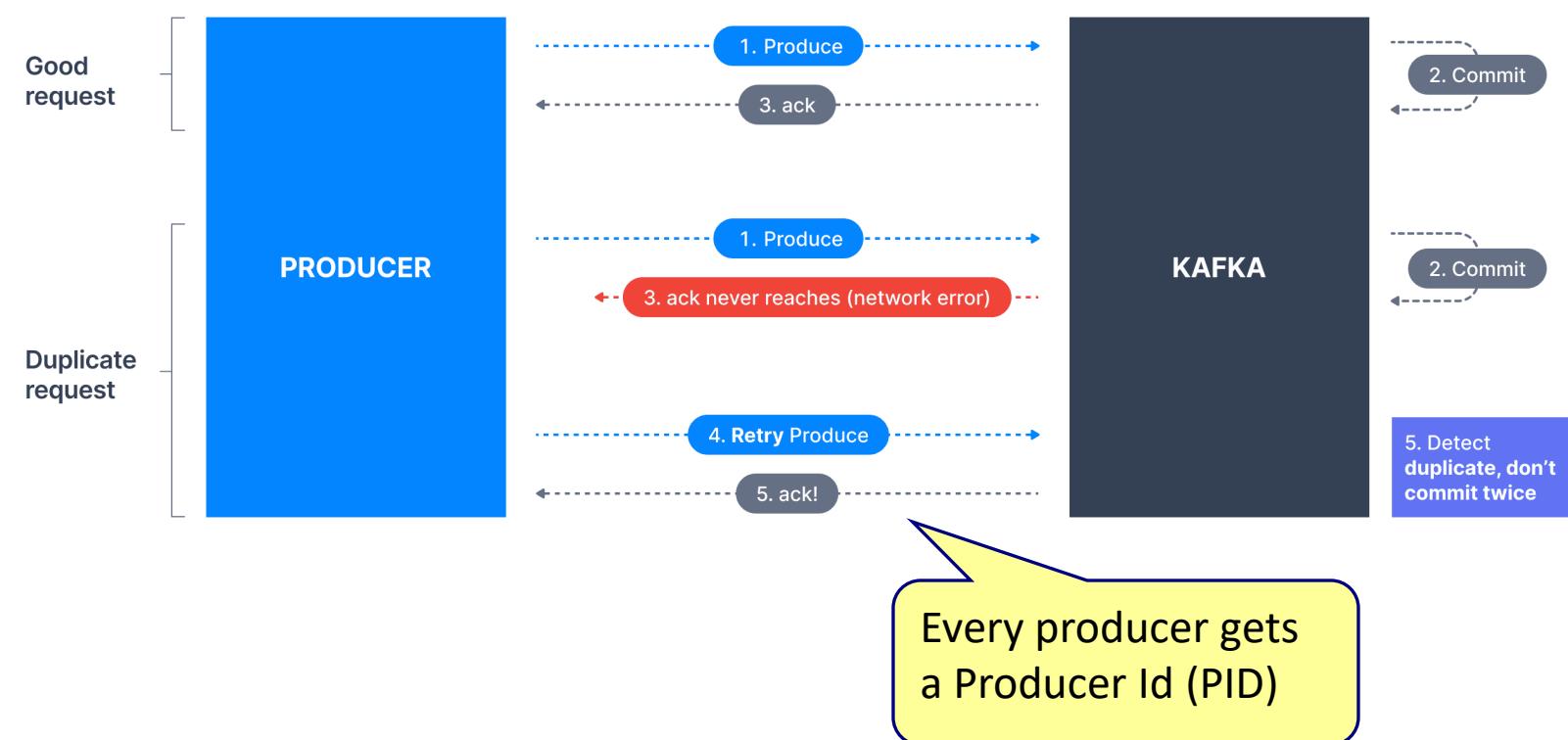
Producer retry

- If producer cannot send the message to Kafka, it retries multiple times and ultimately times out after 2 minutes.
- Default values
 - *retries* (defaults to `Integer.MAX_VALUE`): the maximum number of attempts to publish the message
 - *delivery.timeout.ms* (defaults to 120.000): the maximum time to wait for a message to be acknowledged before considering it failed
 - *retry.backoff.ms* (defaults to 100): the time to wait before retrying

Problem with producer retries



Idempotent producer



Idempotent producer

application.properties

```
1 spring.application.name=KafkaProducer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 spring.kafka.producer.enable-idempotence=true
```

```
: [Producer clientId=KafkaProducer-producer-1] Instantiated an idempotent producer.  
: Kafka version: 3.9.1  
: Kafka commitId: f745dfdce2b9851  
: Kafka startTimeMs: 1756476098345  
: [Producer clientId=KafkaProducer-producer-1] Cluster ID: 5L6g3nShT-eMCTK--X86sw  
: [Producer clientId=KafkaProducer-producer-1] ProducerId set to 1048 with epoch 0  
: [Producer clientId=KafkaProducer-producer-1] Closing the Kafka producer with timeoutMillis = 30000
```

Consumer

- At most once



- At least once



Consumer needs
to be idempotent

- Exactly once

- You need a cache or database to store the messages you have already processed

At most once



- This is the default for Spring Boot Kafka
- The message is automatically acknowledged the moment it is received by the consumer
- Problem: what if we get an error during processing the message

At least once



- Retry when there is a processing error
- What if it still fails after a few retries?
 - Send the message to the Dead Letter Topic (DLT)

Simple producer

```
@Service
public class KafkaProducer2 {
    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage() {
        kafkaTemplate.send("topic1", "Hello World");
        kafkaTemplate.send("topic1", "Hello");
        kafkaTemplate.send("topic1", "GoodBy");
    }
}
```

Handle error in consumer

```
@Service
public class KafkaConsumer2 {
    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
        topic, @Header(KafkaHeaders.OFFSET) long offset) {
        try {
            System.out.println("Listener: Message received = " + message + ", topic= " + topic + ","
                + offset);
            if (message.equals("Hello")) {
                throw new RuntimeException("Invalid message received");
            }
        } catch (Exception exception) {
            System.out.println("Exception in listener: exception = " + exception);
        }
    }
}
```

Exception in
message handler

Listener: Message received = Hello World, topic= topic1, offset= 15

Listener: Message received = Hello, topic= topic1, offset= 16

Exception in listener: exception = java.lang.RuntimeException: Invalid message received

Listener: Message received = GoodBy, topic= topic1, offset= 17

Consumer error handling options

- **Blocking retry**

- Do retry when retriable exceptions occur during consuming a message, and **block** the next message.
- **DefaultErrorHandler**



Consumer thread is blocked during retry
Order is guaranteed

- **Non-blocking** retry

- Send the message to another retry topic, when the message exceeds the retry max attempts limit.
- **@RetryableTopic**

Consumer thread is not blocked during retry
Order is not guaranteed

- Dead letter queue and handler

- Send the message to a dead letter topic.

Global exception handler (DefaultErrorHandler)

```
@Configuration
public class KafkaConfig {
    @Bean
    public DefaultErrorHandler errorHandler() {
        FixedBackOff noRetry = new FixedBackOff(0L, 0L);

        BiConsumer<ConsumerRecord<?, ?>, Exception> globalHandler = (record, ex) -> {
            System.err.println("Kafka error occurred:");
            System.err.println("Topic: " + record.topic());
            System.err.println("Value: " + record.value());
            System.err.println("Exception: " + ex.getMessage());
        };
        DefaultErrorHandler errorHandler = new DefaultErrorHandler(
            (record, exception) -> globalHandler.accept(record, exception),
            noRetry
        );
        return errorHandler;
    }
}
```

Global exception handler config

```
@Bean  
public ConcurrentKafkaListenerContainerFactory<String, String>  
    kafkaListenerContainerFactory(  
        ConsumerFactory<String, String> consumerFactory,  
        DefaultErrorHandler errorHandler) {  
  
    ConcurrentKafkaListenerContainerFactory<String, String> factory =  
        new ConcurrentKafkaListenerContainerFactory<>();  
    factory.setConsumerFactory(consumerFactory);  
    factory.setCommonErrorHandler(errorHandler);  
    return factory;  
}
```

Add global error
handler to the
ListernerContainer

Kafka listener with global exception handler

```
public class KafkaConsumer2 {  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String  
        topic, @Header(KafkaHeaders.OFFSET) long offset) {  
  
        System.out.println("Listener: Message received = " + message + ", topic= " + topic + ",  
            offset= " + offset);  
        if (message.equals("Hello")) {  
            throw new RuntimeException("Invalid message received");  
        }  
    }  
}
```

No exception handling in the Listener

Kafka listener with global exception handler

Listener: Message received = Hello World, topic= topic1, offset= 21

Listener: Message received = Hello, topic= topic1, offset= 22

Kafka error occurred:

Topic: topic1

Value: Hello

Exception: Listener method 'public void

consumer.KafkaConsumer2.consume(java.lang.String,java.lang.String,long)' threw exception

Listener: Message received = GoodBy, topic= topic1, offset= 23

Global error handler with retries

```
@Configuration
public class KafkaConfig2 {
    @Bean
    public DefaultErrorHandler errorHandler() {
        // Retry 2 times, wait 1 second between retries
        FixedBackOff fixedBackOff = new FixedBackOff(1000L, 2L);
        2 retries
        DefaultErrorHandler errorHandler = new DefaultErrorHandler(
            (ConsumerRecord<?, ?> record, Exception ex) -> {
                // Custom recovery logic after retries are exhausted
                System.err.println("Error after retries for record: " + record.value()
                    + ", exception: " + ex.getMessage());
            },
            fixedBackOff
        );
        Blocking retry
        return errorHandler;
    }
}
```

Global error handler with retries and Dead Letter Topic

```
@Bean
```

```
public DefaultErrorHandler errorHandler(KafkaTemplate<Object, Object> kafkaTemplate) {  
    // Sends failed messages to "<topic>.DLT"  
    DeadLetterPublishingRecoverer recoverer = new  
        DeadLetterPublishingRecoverer(kafkaTemplate,  
            (record, exception) -> {  
                return new TopicPartition(record.topic() + ".DLT", record.partition());  
            });  
  
    // Retry twice with 1 second interval, then send to DLT  
    FixedBackOff backOff = new FixedBackOff(1000L, 2L);  
  
    DefaultErrorHandler errorHandler = new DefaultErrorHandler(recoverer, backOff);  
    return errorHandler;  
}
```

Send failed messages to DLT

2 retries

Global error handler with retries and Dead Letter Topic

```
@Service
```

```
public class KafkaConsumer2 {
```

```
    @KafkaListener(topics = "topic1", groupId = "gid1")
```

```
    public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String topic, @Header(KafkaHeaders.OFFSET) long offset) {
```

```
        System.out.println("Listener: Message received = " + message + ", topic= " + topic + ",  
        offset= " + offset);
```

```
        if (message.equals("Hello")) {
```

```
            throw new RuntimeException("Invalid message received");
```

```
}
```

```
}
```

```
    @KafkaListener(topics = "topic1.DLT", groupId = "dlt-group")
```

```
    public void handleDltMessage(String message) {
```

```
        System.err.println("Received from DLT: " + message);
```

```
}
```

```
}
```

Listen on DLT

Global error handler with retries

Listener: Message received = Hello World, topic= topic1, offset= 33

Listener: Message received = Hello, topic= topic1, offset= 34

...

Listener: Message received = Hello, topic= topic1, offset= 34

...

Listener: Message received = Hello, topic= topic1, offset= 34

...

Received from DLT: Hello

...

Listener: Message received = GoodBy, topic= topic1, offset= 35

Retry 1

Retry 2

Send to DLT

Received from DLT

Consumer error handling options

- Blocking retry
 - Do retry when retriable exceptions occur during consuming a message, and **block** the next message.
 - **DefaultErrorHandler**
- Non-blocking retry
 - Send the message to another retry topic, when the message exceeds the blocking retry max attempts limit.
 - **@RetryableTopic** 
- Dead letter queue and handler
 - Send the message to another dead letter topic.

Non blocking retries with DLT

```
@Service  
public class KafkaConsumer {  
  
    @RetryableTopic(attempts = "2")  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String  
        topic, @Header(KafkaHeaders.OFFSET) long offset){  
  
        System.out.println("Listener: Message received = "+message+", topic= "+topic+",  
            offset= "+offset);  
        if (message.equals("Hello")){  
            throw new RuntimeException("Invalid message received");  
        }  
    }  
}
```

1 retry on exception

Non-blocking retry

Non blocking retries with DLT

```
@Service  
public class KafkaConsumer {
```

After the retries failed, Spring will automatically place this message in the Dead Letter Topic (DLT)

```
    @RetryableTopic(attempts = "2")  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String topic, @Header(KafkaHeaders.OFFSET) long offset) {  
        ...  
    }
```

Listener for DLT topic

```
    @DltHandler  
    public void listenDLT(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String topic, @Header(KafkaHeaders.OFFSET) long offset) {  
        System.out.println("DLT Received = "+message+", topic= "+topic+", offset= "+offset);  
    }  
}
```

Non blocking retries with DLT

Listener: Message received = Hello World, topic= topic1, offset= 36

Listener: Message received = Hello, topic= topic1, offset= 37

...

Listener: Message received = GoodBy, topic= topic1, offset= 38

Listener: Message received = Hello, topic= topic1-retry, offset= 0

...

DLT Received = Hello, topic= topic1-dlt, offset= 6

1 retry on topic1-retry

Non blocking retries with DLT

```
@Service
public class KafkaConsumer {
    @RetryableTopic(attempts = "3")
    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
    ...
}

@DltHandler
public void listenDLT(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
    System.out.println("DLT Received = "+message+", topic= "+topic+", offset= "+offset);
}
}
```

2 retries

Non blocking retries with DLT

Listener: Message received = Hello World, topic= topic1, offset= 39

Listener: Message received = Hello, topic= topic1, offset= 40

Listener: Message received = GoodBy, topic= topic1, offset= 41

...

Listener: Message received = Hello, topic= topic1-retry, offset= 1

...

Listener: Message received = Hello, topic= topic1-retry, offset= 2

...

DLT Received = Hello, topic= topic1-dlt, offset= 7

2 retries on topic1-retry

Non blocking retries with DLT

```
@Service
public class KafkaConsumer {
    @RetryableTopic(attempts = "3",
        backoff = @Backoff(delay = 1000, multiplier = 2))
    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
    ...
}

@DLtHandler
public void listenDLT(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
    System.out.println("DLT Received = "+message+", topic= "+topic+", offset= "+offset);
}
}
```

Backoff algorithm

Non blocking retries with DLT

Listener: Message received = Hello World, topic= topic1, offset= 42

Listener: Message received = Hello, topic= topic1, offset= 43

...

Listener: Message received = GoodBy, topic= topic1, offset= 44

Listener: Message received = Hello, topic= topic1-retry-1000, offset= 0

..

Listener: Message received = Hello, topic= topic1-retry-2000, offset= 0

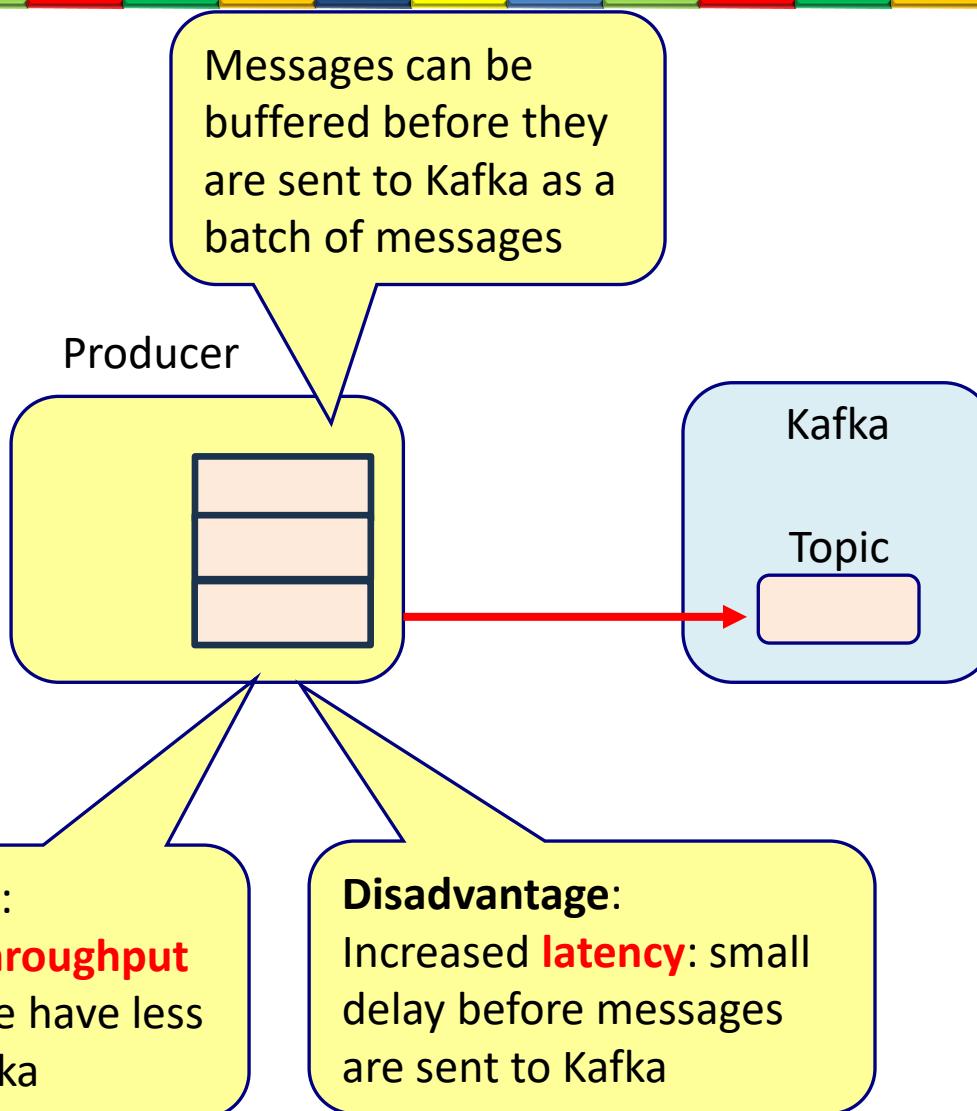
..

DLT Received = Hello, topic= topic1-dlt, offset= 8

Send message to new topic
every retry

BATCH MESSAGES

Producer batching



Producer batch settings

- **linger.ms**
 - Number of milliseconds a producer is willing to wait before sending a batch out.
 - Default value is 0, which means "send the messages right away".
- **batch.size**
 - Maximum number of bytes that will be included in a batch
 - Default value is 16KB

Producer

```
@Service  
public class KafkaProducer2 {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() throws InterruptedException {  
        for (int x=1; x<13 ; x++){  
            kafkaTemplate.send("topic1","Message-"+x);  
            System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());  
            Thread.sleep(1000);  
        }  
    }  
}
```

Send message every second

application.properties

```
1 spring.application.name=KafkaProducer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 spring.kafka.producer.properties.linger.ms=4000
```

Batch all messages
for 4 seconds

Producer

```
public void sendMessage() throws InterruptedException {  
    for (int x=1; x<13 ; x++){  
        kafkaTemplate.send("topic1" , "Message-"+x);  
        System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());  
        Thread.sleep(1000);  
    }  
}
```

sending message-1 at 29
sending message-2 at 30
sending message-3 at 31
sending message-4 at 32
sending message-5 at 33
sending message-6 at 34
sending message-7 at 35
sending message-8 at 36
sending message-9 at 37
sending message-10 at 38
sending message-11 at 39
sending message-12 at 40

Sending a message
every second

Consumer

```
@Service  
public class KafkaConsumer2 {  
  
    @KafkaListener(topics = "topic1", groupId = "gid1")  
    public void consume(Message<String> message) {  
        System.out.println("Receiving message = "+message.getPayload()+" at "+  
            LocalTime.now().getSecond());  
    }  
}
```

```
Receiving message = Message-1 at 33  
Receiving message = Message-2 at 33  
Receiving message = Message-3 at 33  
Receiving message = Message-4 at 33  
Receiving message = Message-5 at 37  
Receiving message = Message-6 at 37  
Receiving message = Message-7 at 37  
Receiving message = Message-8 at 37  
Receiving message = Message-9 at 41  
Receiving message = Message-10 at 41  
Receiving message = Message-11 at 41  
Receiving message = Message-12 at 41
```

4 messages every 4 seconds

Consumer batch settings

- **Max.poll.records**

- The maximum number of messages your consumer will receive in a single poll.
- Default value is 500

- **fetch.min.bytes**

- The broker will wait until at least this many bytes of messages are available before responding.
- Default value is 1

- **fetch.max.wait.ms**

- The maximum time the broker will wait if fetch.min.bytes isn't satisfied.
- Default value is 500 ms

Producer

```
@Service  
public class KafkaProducer2 {  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage() throws InterruptedException {  
        for (int x=1; x<13 ; x++){  
            kafkaTemplate.send("topic1", "Message-"+x);  
            System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());  
            Thread.sleep(1000);  
        }  
    }  
}
```

Send message every second

application.properties

```
1  spring.application.name=KafkaProducer  
2  
3  spring.kafka.bootstrap-servers=localhost:9092  
4
```

No batching

Producer

```
public void sendMessage() throws InterruptedException {  
    for (int x=1; x<13 ; x++){  
        kafkaTemplate.send("topic1" , "Message-"+x);  
        System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());  
        Thread.sleep(1000);  
    }  
}
```

sending message-1 at 29
sending message-2 at 30
sending message-3 at 31
sending message-4 at 32
sending message-5 at 33
sending message-6 at 34
sending message-7 at 35
sending message-8 at 36
sending message-9 at 37
sending message-10 at 38
sending message-11 at 39
sending message-12 at 40

Sending a message
every second

Consumer

```
@Service  
public class KafkaConsumer2 {
```

```
@KafkaListener(topics = "topic1", groupId = "gid1")  
public void consume(List<Message<String>> messages) {  
    for(Message message: messages) {  
        System.out.println("Receiving message = " + message.getPayload() + " at " +  
            LocalTime.now().getSecond());  
    }  
}
```

List of messages

```
}
```

application.properties

```
1 spring.application.name=KafkaConsumer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4  
5 spring.kafka.listener.type=batch  
6 spring.kafka.consumer.properties.fetch.min.bytes=1048576  
7 spring.kafka.consumer.properties.fetch.max.wait.ms=5000
```

Wait for a maximum of 5 seconds

Consumer

```
Receiving message = Message-1 at 19  
Receiving message = Message-2 at 19  
Receiving message = Message-3 at 19  
Receiving message = Message-4 at 19  
Receiving message = Message-5 at 24  
Receiving message = Message-6 at 24  
Receiving message = Message-7 at 24  
Receiving message = Message-8 at 24  
Receiving message = Message-9 at 24  
Receiving message = Message-10 at 29  
Receiving message = Message-11 at 29  
Receiving message = Message-12 at 29
```

Receive messages in batches

TESTING

Testing Kafka applications

- Using the embedded Kafka broker
- Using Testcontainers

Test using embedded Kafka

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka-test</artifactId>
    <scope>test</scope>
</dependency>
```

Producer

```
@Service
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage(String message) {
        kafkaTemplate.send("topic1", message);
    }
}
```

Consumer

```
@Service
public class KafkaConsumer {
    private String receivedMessage="";

    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(String message) {
        System.out.println("Receiving message = " + message);
        receivedMessage = message;
    }

    public String getReceivedMessage() {
        return receivedMessage;
    }
}
```

Test using embedded Kafka

```
@SpringBootTest  
@EmbeddedKafka(topics = {"topic1"})  
public class KafkaIntegrationTest {  
    @Autowired  
    private KafkaConsumer consumer;  
  
    @Autowired  
    private KafkaProducer producer;  
  
    @Test  
    void testKafka() {  
        producer.sendMessage("Hello World");  
        assertThat(consumer.getReceivedMessage()).equals("Hello World"));  
    }  
}
```

Create the Spring context

Use the embedded kafka broker

Test using TestContainer

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>kafka</artifactId>
    <scope>test</scope>
</dependency>
```

Test using TestContainer(1/2)

```
@SpringBootTest  
@Testcontainers  
public class KafkaIntegrationTest {  
    @Container  
    static final KafkaContainer kafka = new KafkaContainer(  
        DockerImageName.parse("apache/kafka:latest")  
    );  
  
    @DynamicPropertySource  
    static void overrideProperties(DynamicPropertyRegistry registry) {  
        registry.add("spring.kafka.bootstrap-servers", kafka::getBootstrapServers);  
    }  
}
```

@Testcontainers

Create a test container based on a Docker image

Test using TestContainer(2/2)

```
@Autowired  
private KafkaConsumer consumer;  
  
@Autowired  
private KafkaProducer producer;  
  
@Test  
void testKafka() {  
    producer.sendMessage("Hello World");  
    assertThat(consumer.getReceivedMessage()).equals("Hello World"));  
}  
}
```

TRANSACTIONS

Producer

```
@Service
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @Transactional
    public void generateAndSendPackage()
        throws InterruptedException, TransactionException {
        for (long i = 0; i < 10; i++) {
            kafkaTemplate.send("topic8", "Message"+i+" Time =" + LocalTime.now());
            System.out.println("sending "+ "Message" + i + " Time created=" + LocalTime.now());
            Thread.sleep(1000);
        }
    }
}
```

@Transactional

Producer

application.properties ×

```
1 spring.application.name=KafkaProducer
2
3 spring.kafka.bootstrap-servers=localhost:9092
4 spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
5 spring.kafka.producer.key-serializer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
6 spring.kafka.producer.transaction-id-prefix=tx-
```

Transaction-id-prefix

Instantiated a transactional producer.

Invoking InitProducerId for the first time in order to acquire a producer ID

Discovered transaction coordinator localhost:9092 (id: 1 rack: null)

ProducerId set to 3000 with epoch 4 sending Message0 Time

created=08:22:08.007622300 sending Message1 Time

created=08:22:09.023629900 sending Message2 Time

...

created=08:22:15.075196200 sending Message8 Time

created=08:22:16.077350600 sending Message9 Time

created=08:22:17.090879600 Closing the Kafka producer with timeoutMillis = 30000 ms.

Consumer

```
@Service  
public class KafkaConsumer {  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
    @Transactional  
    @KafkaListener(topics = "topic8", groupId = "gid1")  
    public void consume(String message) {  
        System.out.println("Consumer receiving message = " + message+ " Time  
        received = "+ LocalTime.now());  
    }  
}
```

@Transactional

application.properties

```
1 spring.application.name=KafkaConsumer  
2  
3 spring.kafka.bootstrap-servers=localhost:9092  
4 spring.kafka.consumer.key-serializer=org.apache.kafka.common.serialization.StringSerializer  
5 spring.kafka.consumer.value-serializer=org.springframework.kafka.support.serializer.StringSerializer  
6 spring.kafka.consumer.properties.isolation.level=read_committed
```

Default is read_uncommitted

Consumer



Consumer receiving message = Message0 Time =08:33:05.944749900 Time received = 08:33:16.101232800

Consumer receiving message = Message1 Time =08:33:07.011881300 Time received = 08:33:16.104627700

Consumer receiving message = Message2 Time =08:33:08.018925500 Time received = 08:33:16.104627700

...

Consumer receiving message = Message8 Time =08:33:14.054309500 Time received = 08:33:16.106232900

Consumer receiving message = Message9 Time =08:33:15.060405900 Time received = 08:33:16.106232900

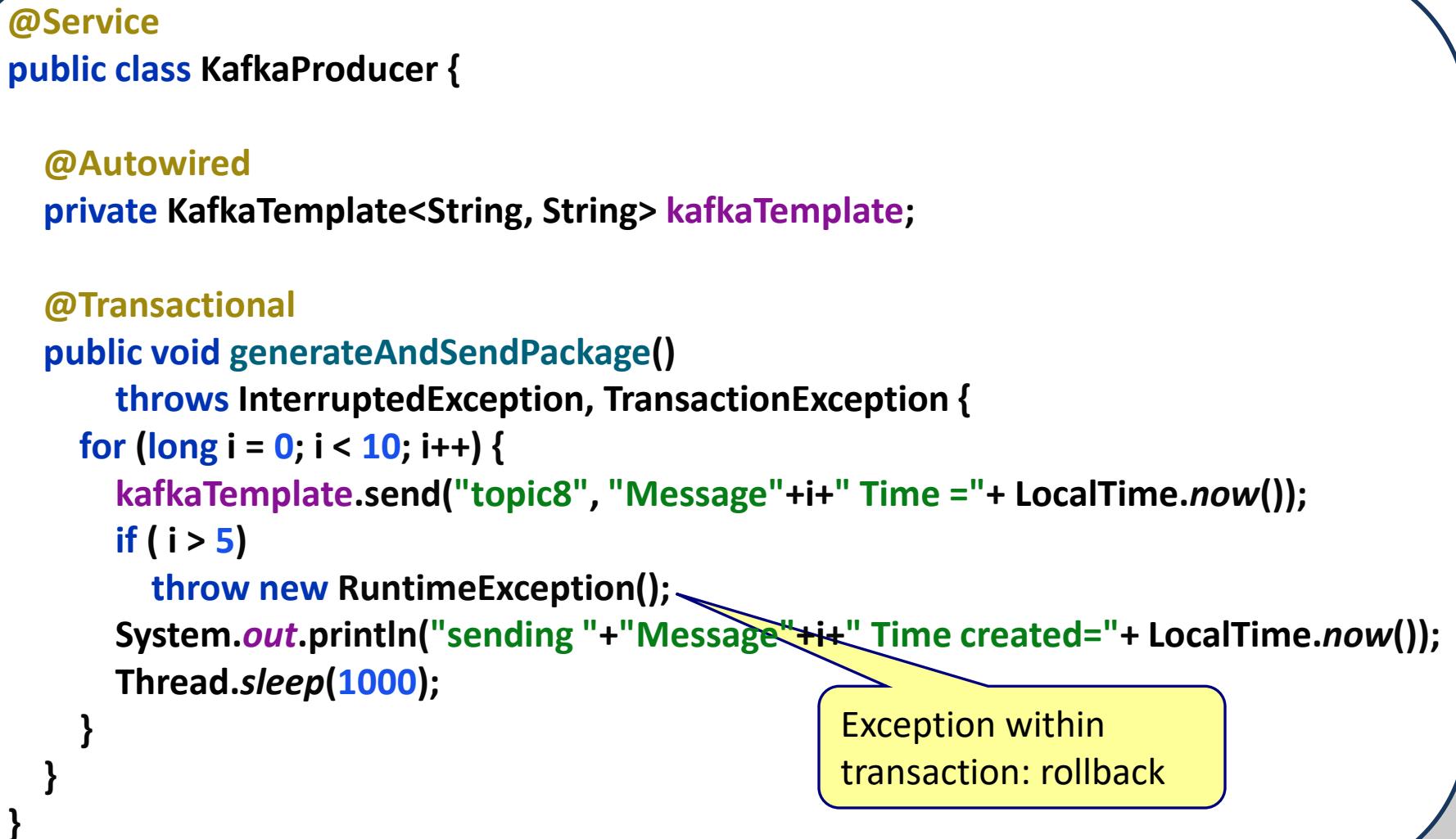
All messages received at the same time (when the sender committed the messages)

Producer

```
@Service
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @Transactional
    public void generateAndSendPackage()
        throws InterruptedException, TransactionException {
        for (long i = 0; i < 10; i++) {
            kafkaTemplate.send("topic8", "Message"+i+" Time =" + LocalTime.now());
            if (i > 5)
                throw new RuntimeException();
            System.out.println("sending "+ "Message" + i + " Time created=" + LocalTime.now());
            Thread.sleep(1000);
        }
    }
}
```



Exception within transaction: rollback

Producer

Instantiated a transactional producer

....

Discovered transaction coordinator localhost:9092 (id: 1 rack: null)

ProducerId set to 3000 with epoch 6

sending Message0 Time created=08:37:25.019418100 sending Message1 Time

created=08:37:26.025871200 sending Message2 Time

created=08:37:27.035123100 sending Message3 Time

created=08:37:28.037490100 sending Message4 Time

created=08:37:29.047949800 sending Message5 Time

created=08:37:30.054130400 Aborting incomplete transaction

...

Closing the Kafka producer with timeoutMillis = 30000 ms.

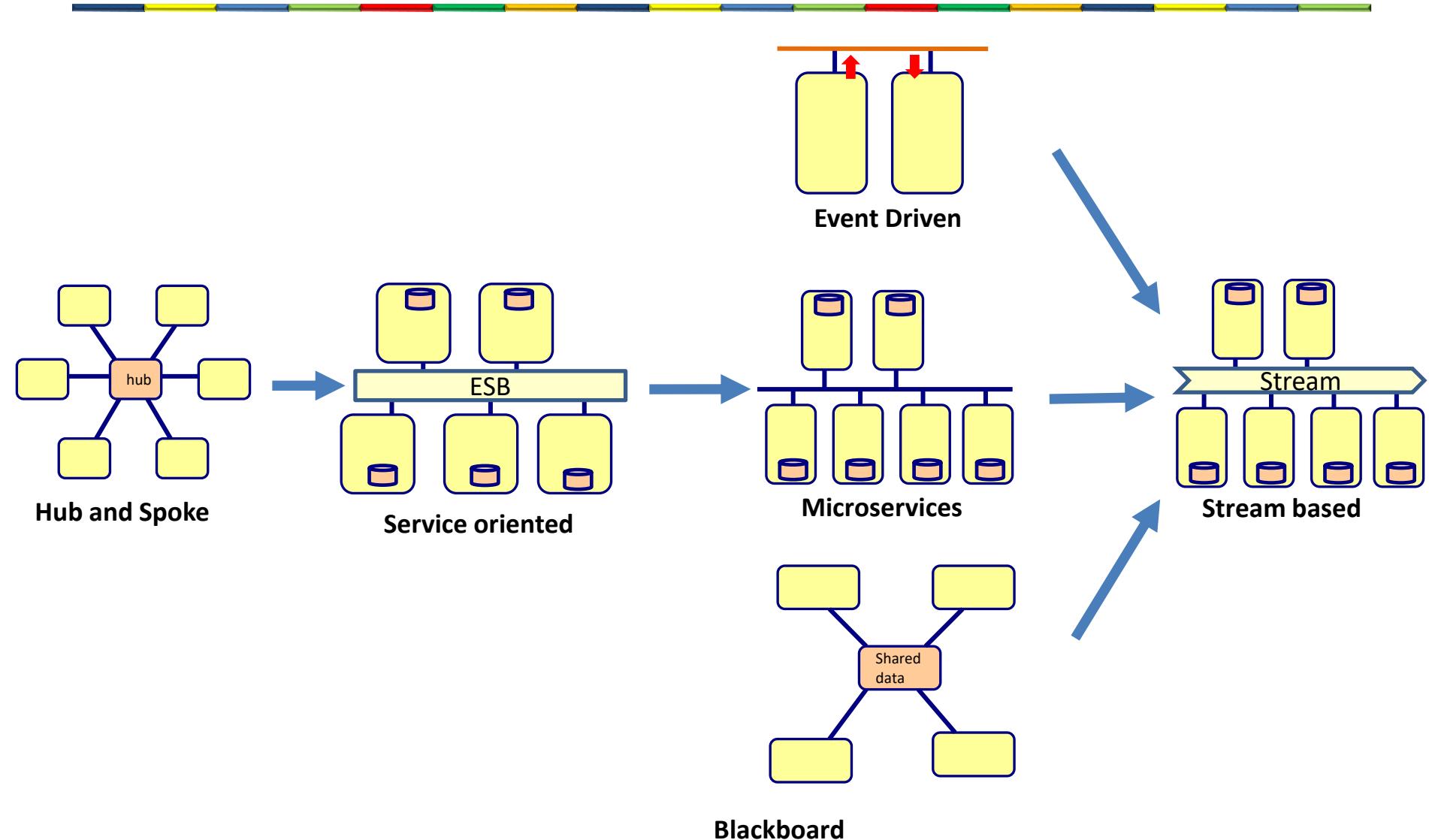
These messages are removed from Kafka

The consumer does not receive any message

Lesson 12

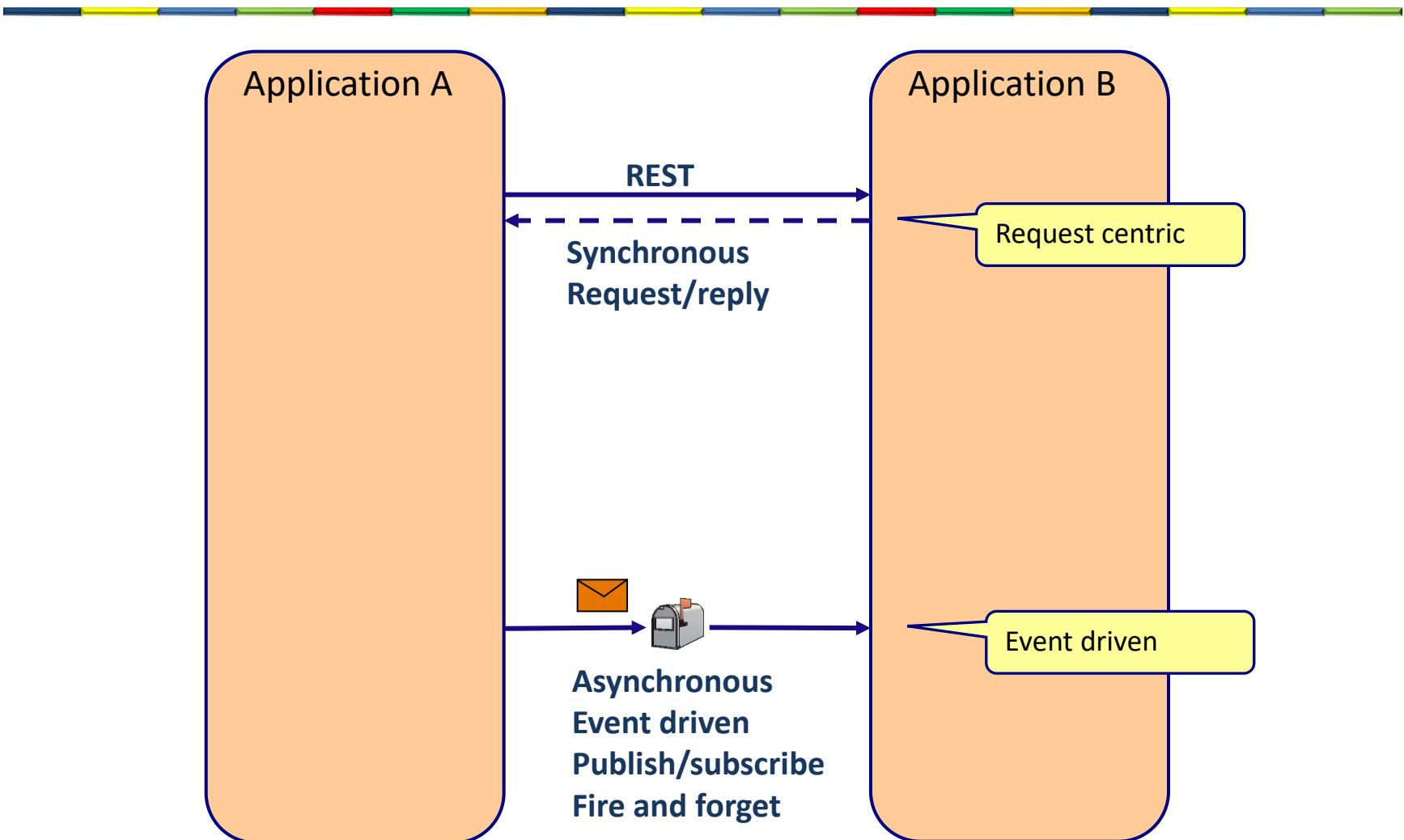
STREAM BASED ARCHITECTURE

Architecture evolution

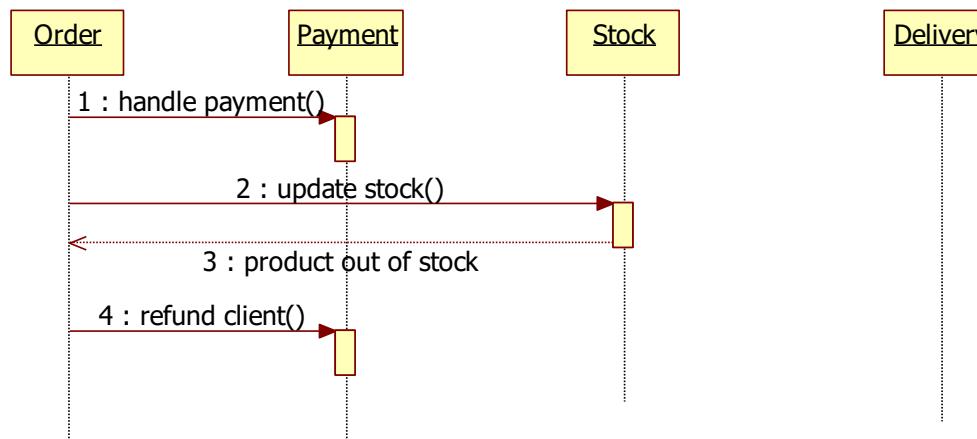
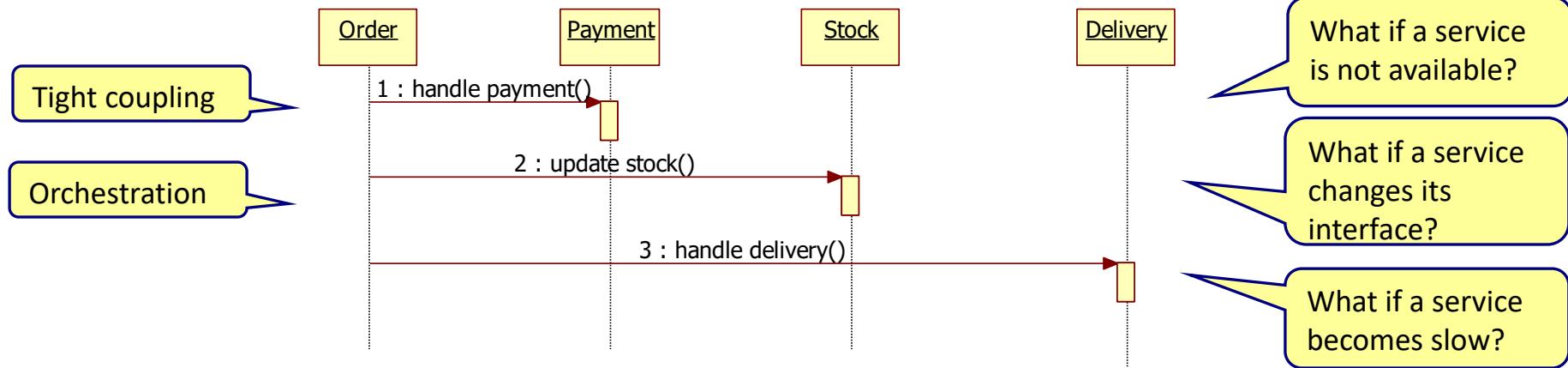


EVENT DRIVEN ARCHITECTURE

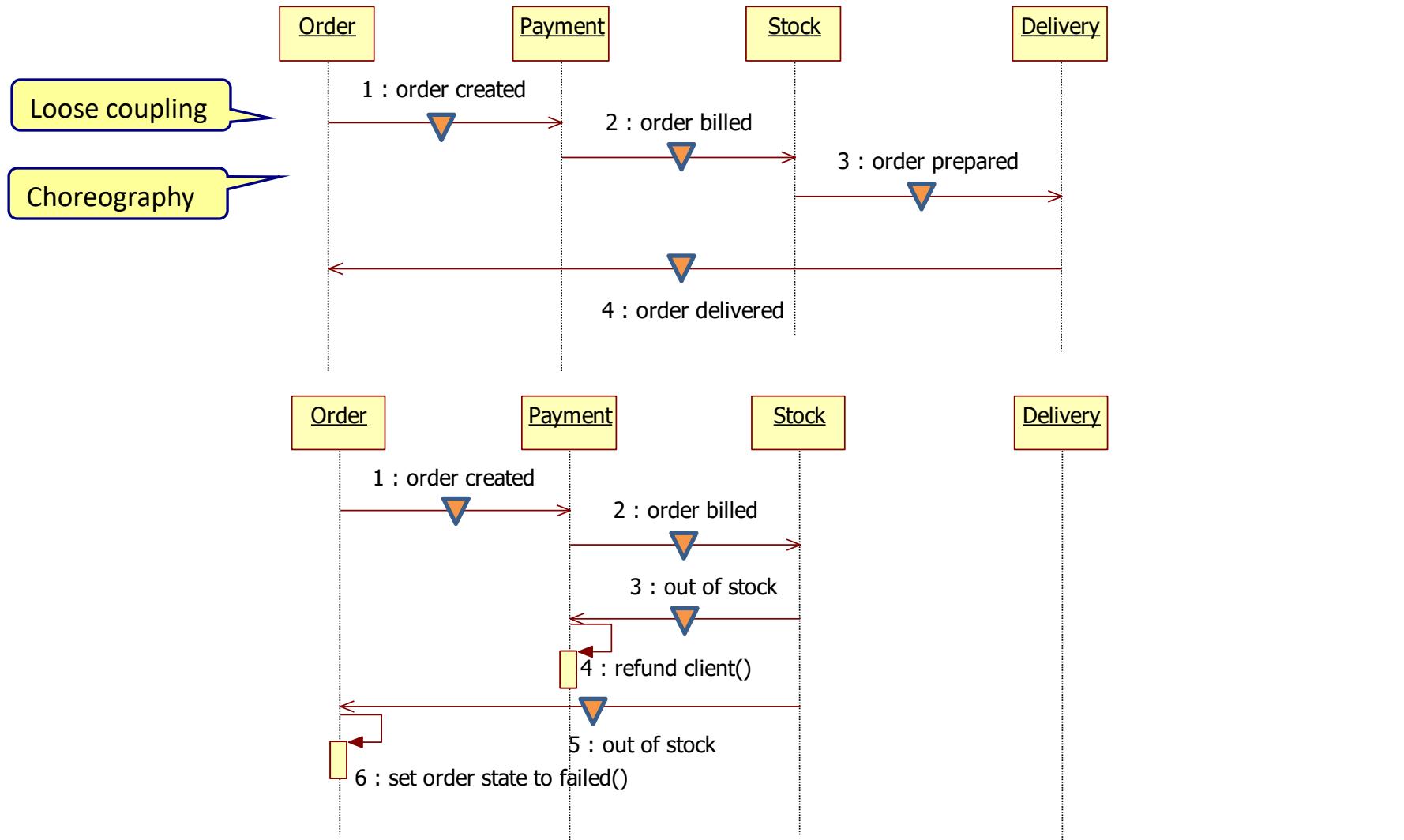
2 ways to communicate



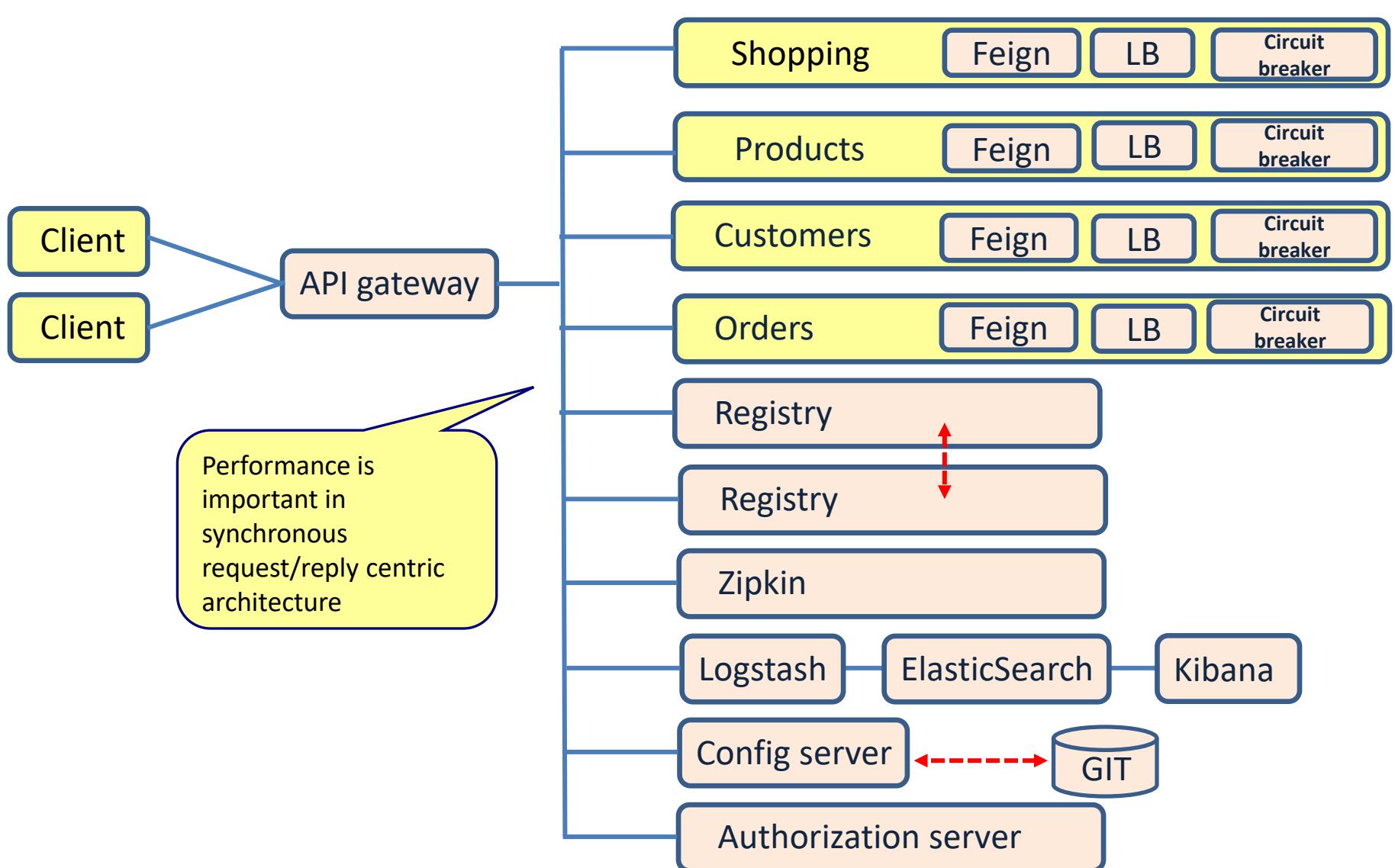
Request centric (REST) calls



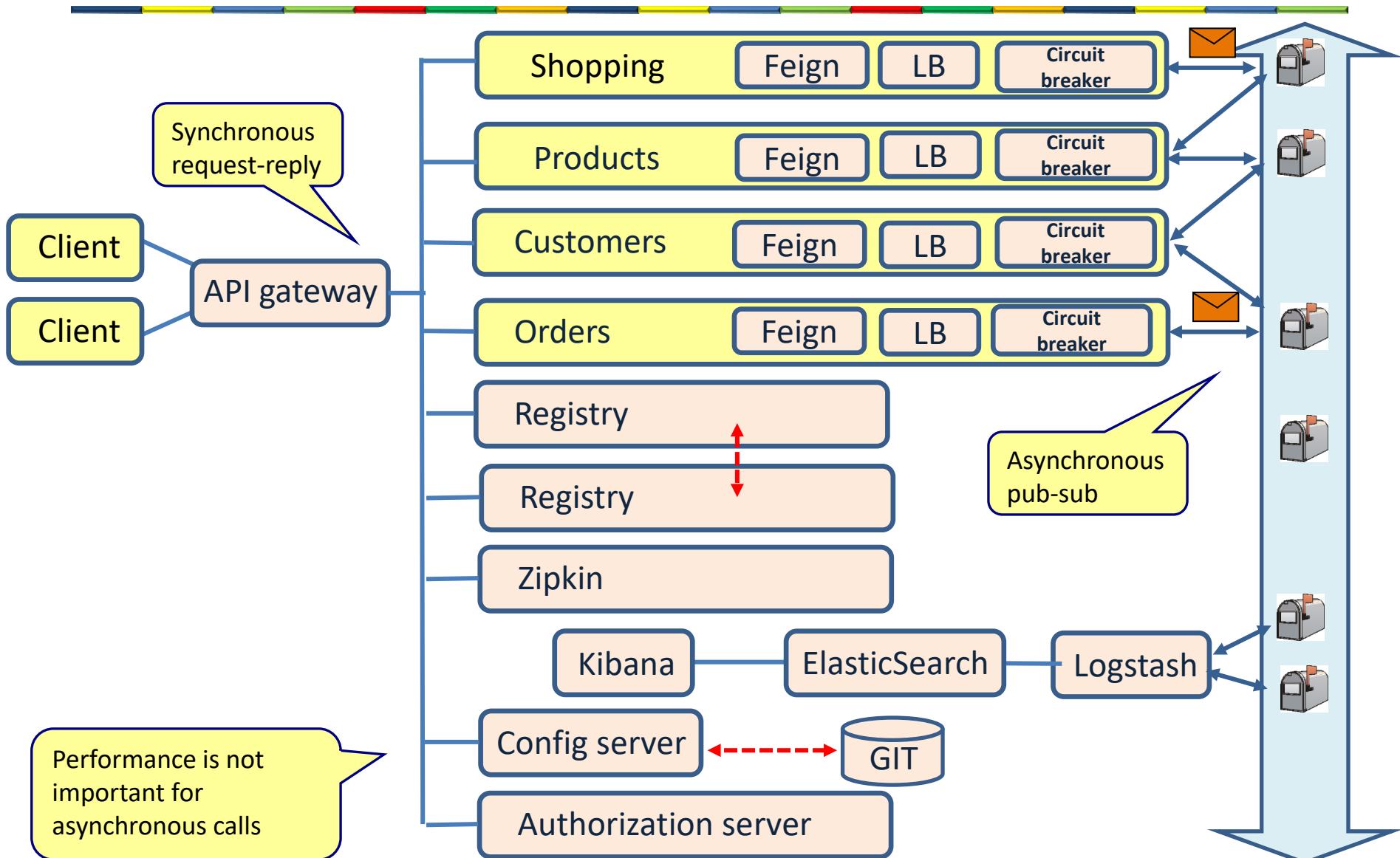
Event driven(messaging)



Implementing microservices



Implementing microservices



Main point

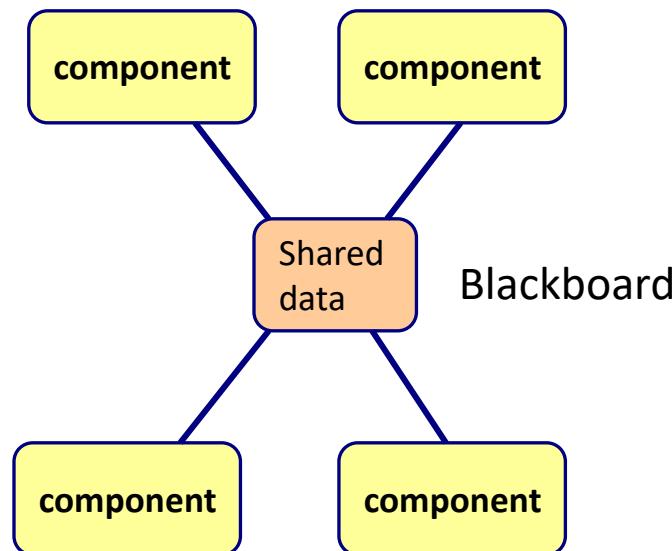
- Messaging gives loose coupling between the sender and the receiver.

Science of Consciousness: The whole relative creation is an expression of the same one unified field.

BLACKBOARD

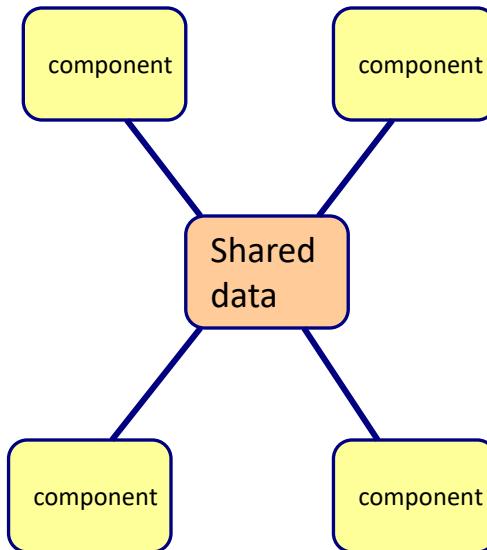
Blackboard

- Used for non deterministic problems
 - There is no fixed straight-line solution to a problem
- Every component adds her information on the blackboard
 - The shared data is **enriched** by the components



Blackboard

- Common data structure
 - Extension is no problem
 - Change is difficult
- Easy to add new components
- Tight coupling for data structure
- Loose coupling for
 - Location
 - Time
 - Technology(?)
- Synchronisation issues



Blackboard

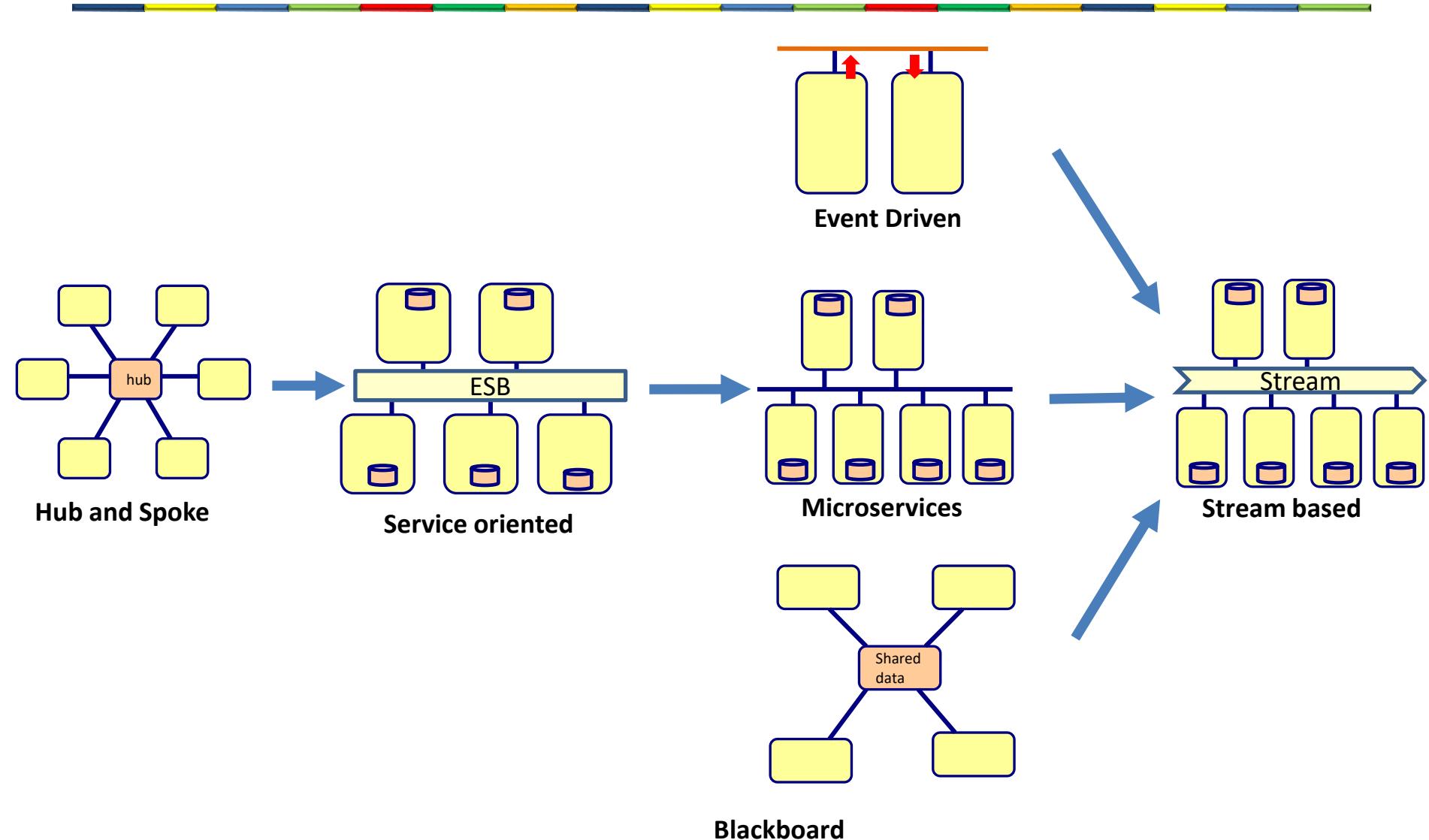
- Benefits
 - Easy to add new components
 - Components are independent of each other
 - Components can work in parallel
- Drawbacks
 - Data structure is hard to change
 - All components share the same data structure
 - Synchronization issues

Main point

- In the blackboard architecture, the different components enhance the data on the blackboard.

Science of Consciousness: The quality of your life gets enhanced when you first water the root before you enjoy the fruit.

Architecture evolution



STREAM BASED ARCHITECTURE

Stream based systems

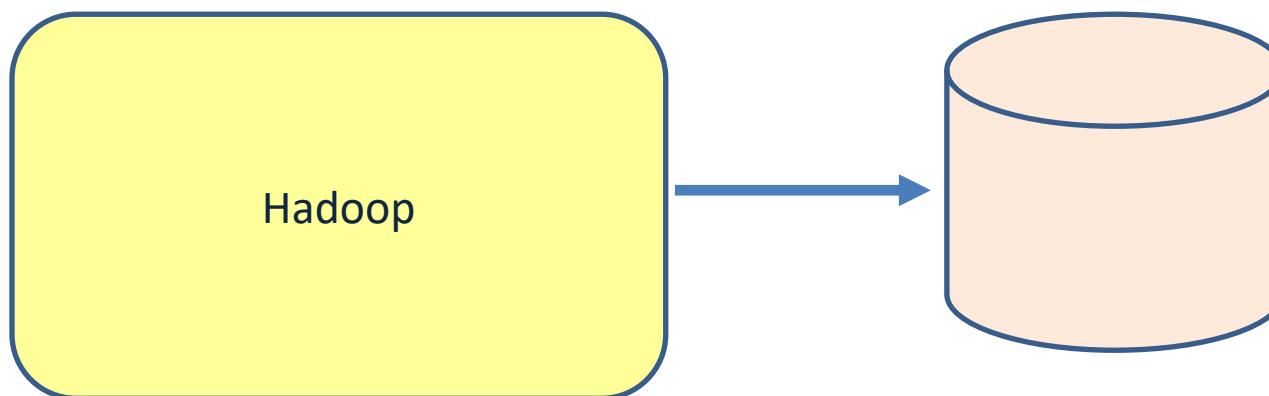
- Continuous stream of data
 - Stock market systems
 - Social networking systems
 - Internet of Things (IoT)systems
 - Systems that handle sensor data
 - System that handle logfiles
 - Systems that monitor user clicks
 - Car navigator software

But also

- Stream of purchases in web shop
- Stream of transactions in a bank
- Stream of actions in a multi user game
- Stream of bookings in a hotel booking system
- Stream of user actions on a web application
- ...

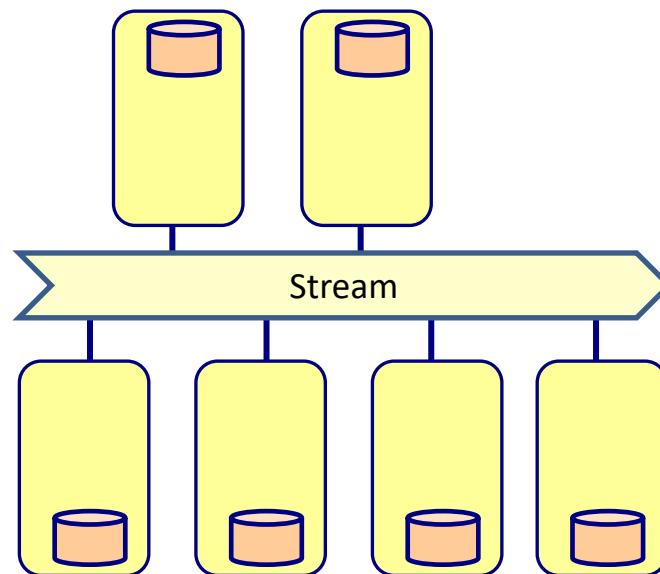
Batch processing

- First store the data in the database
- Then do queries (map-reduce) on the data
- Queries over all or most of the data in the dataset.
- Latencies in minutes to hours

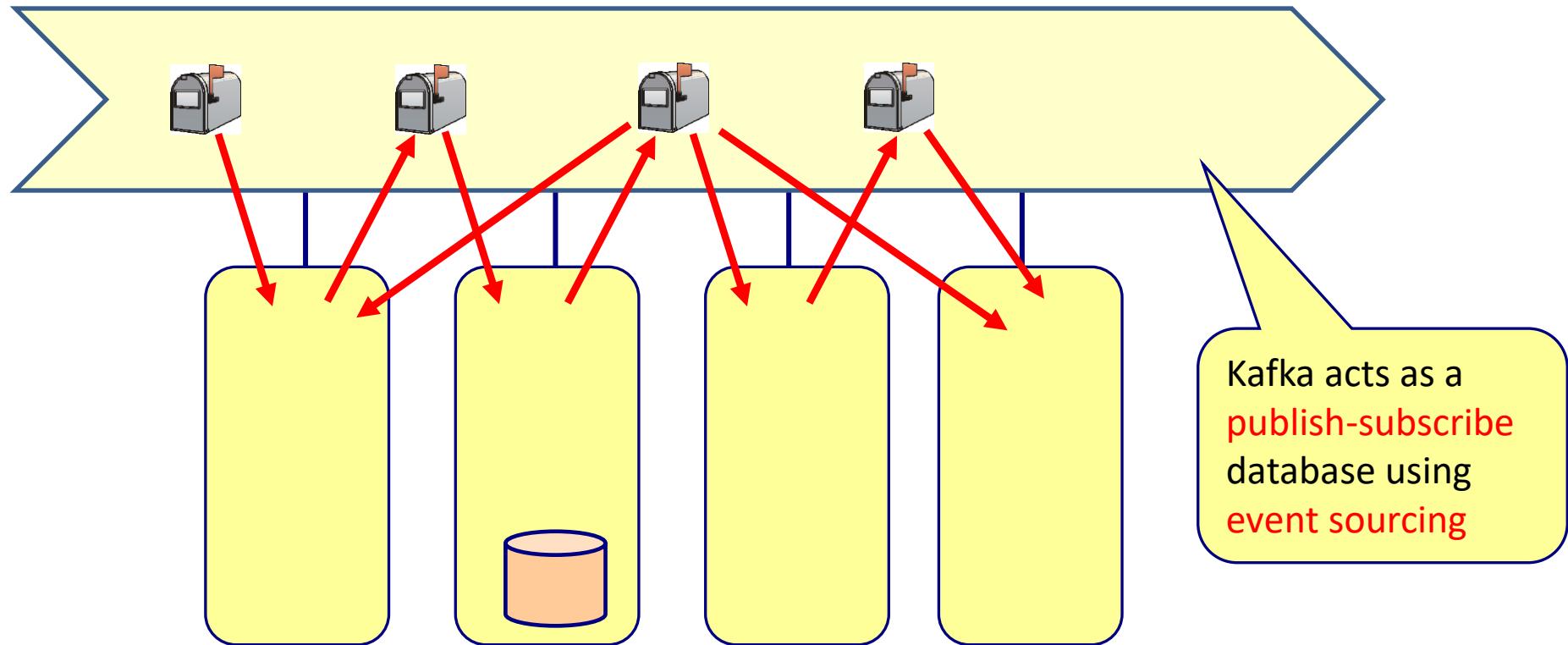


Stream processing

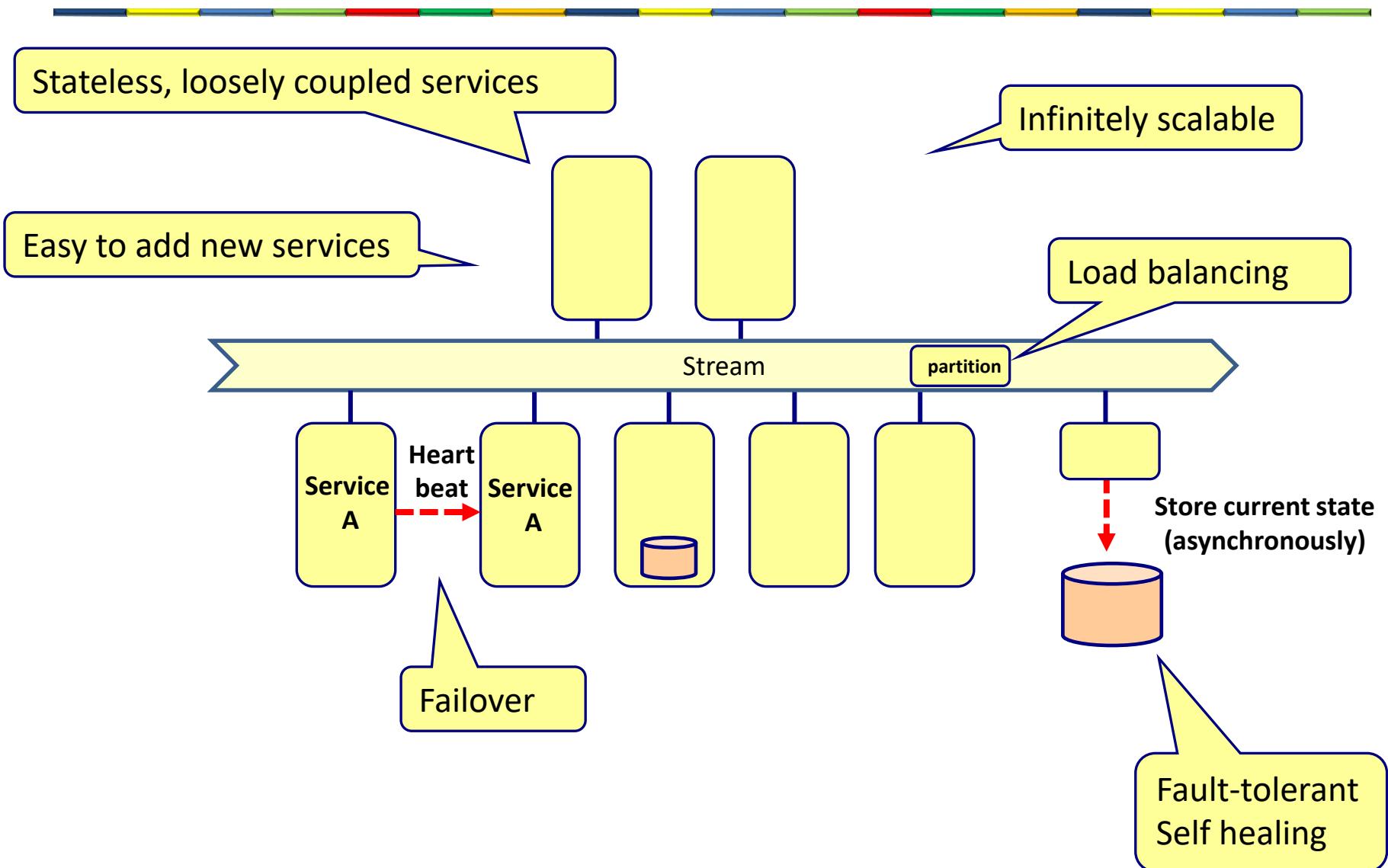
- Handle the data when it arrives
- Handle event (small data) by event
- Latencies in seconds or milliseconds



Publish-subscribe and event sourcing

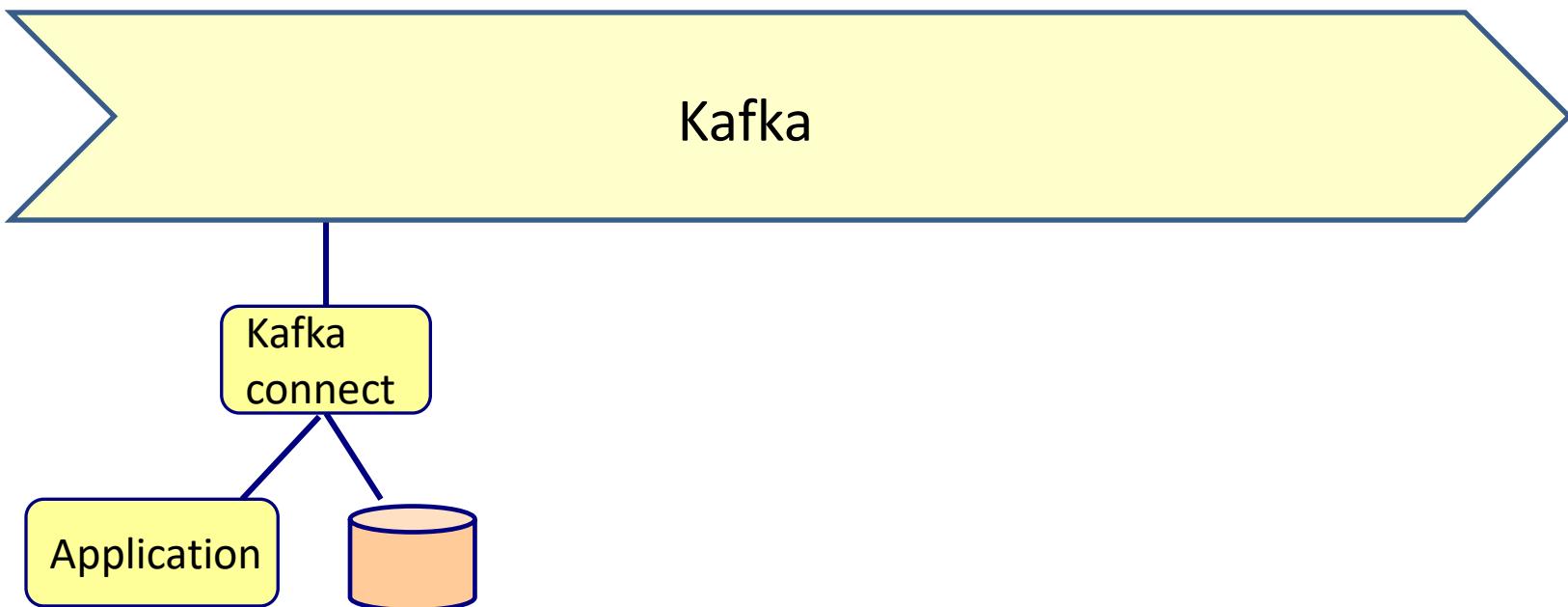


Stream based architecture

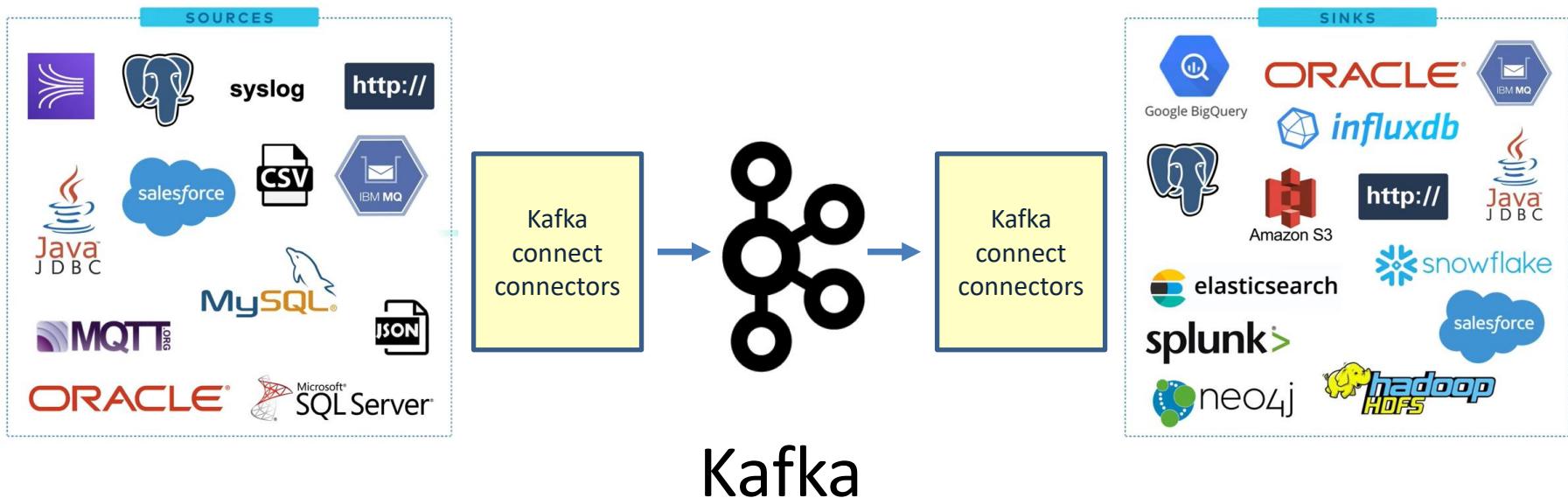


KAFKA ECOSYSTEM

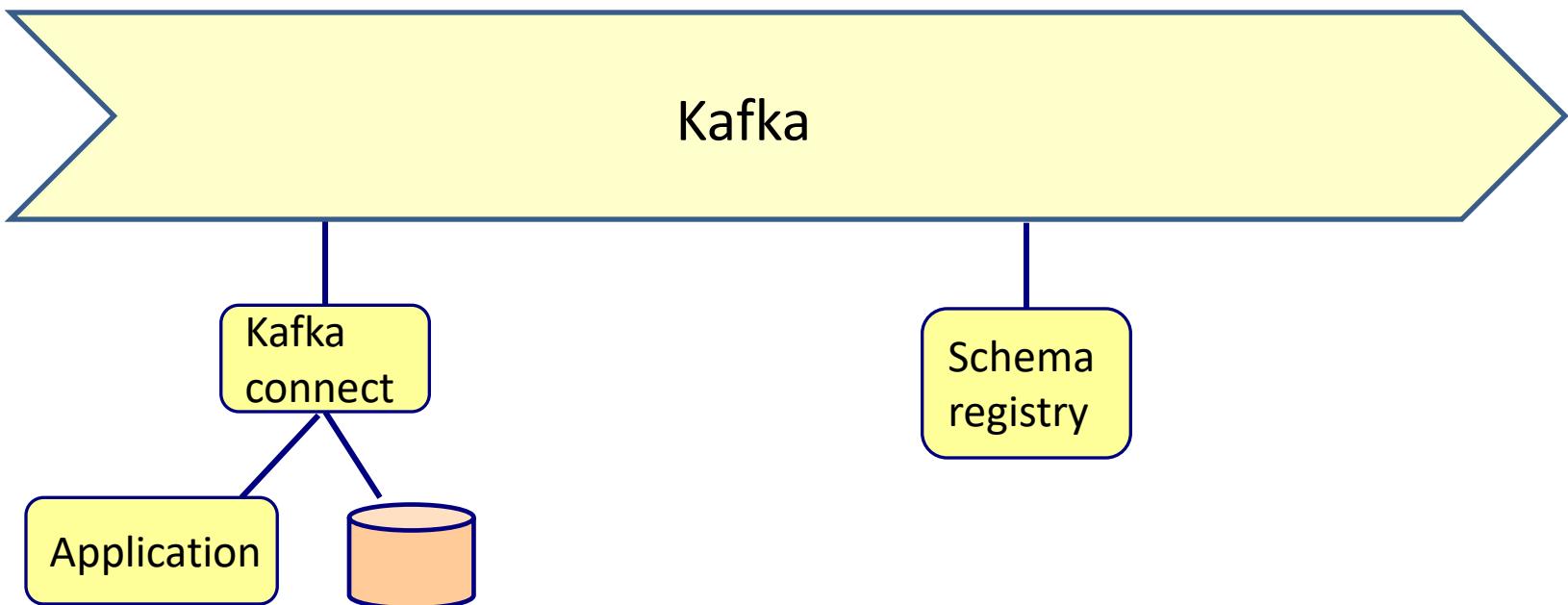
Kafka ecosystem: Kafka connect



Kafka connect



Kafka ecosystem: Schema registry

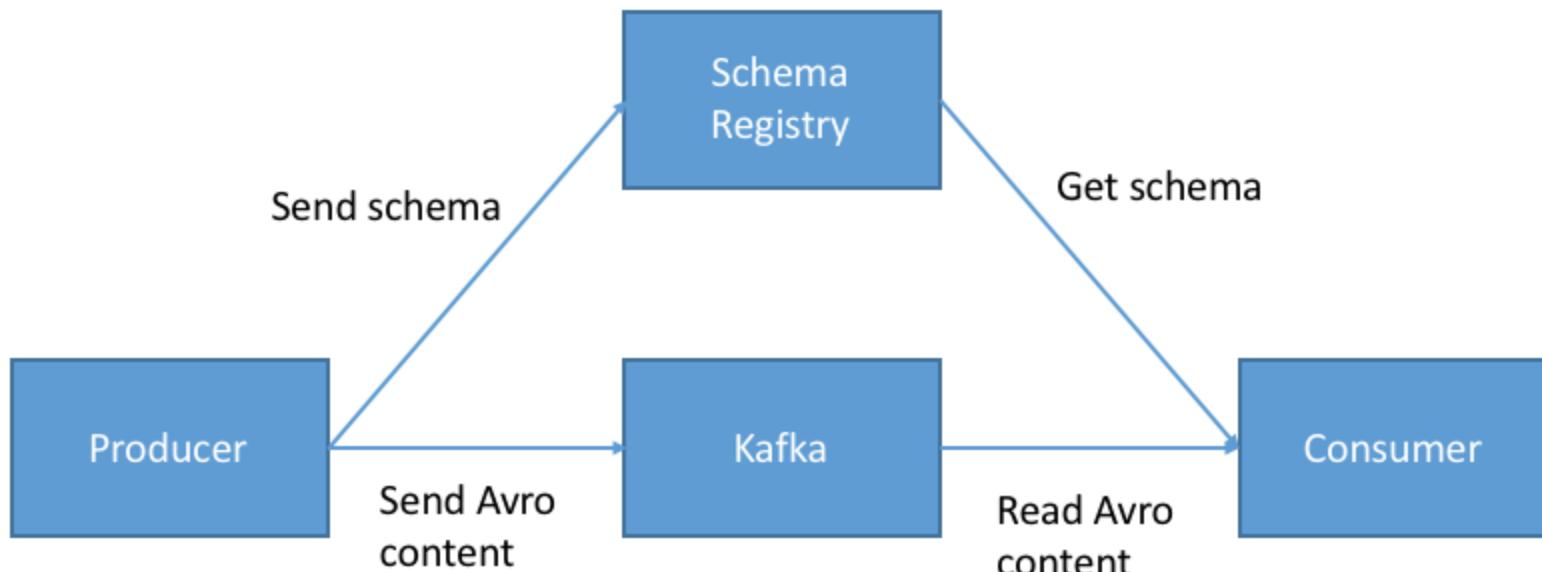


Need for a schema registry

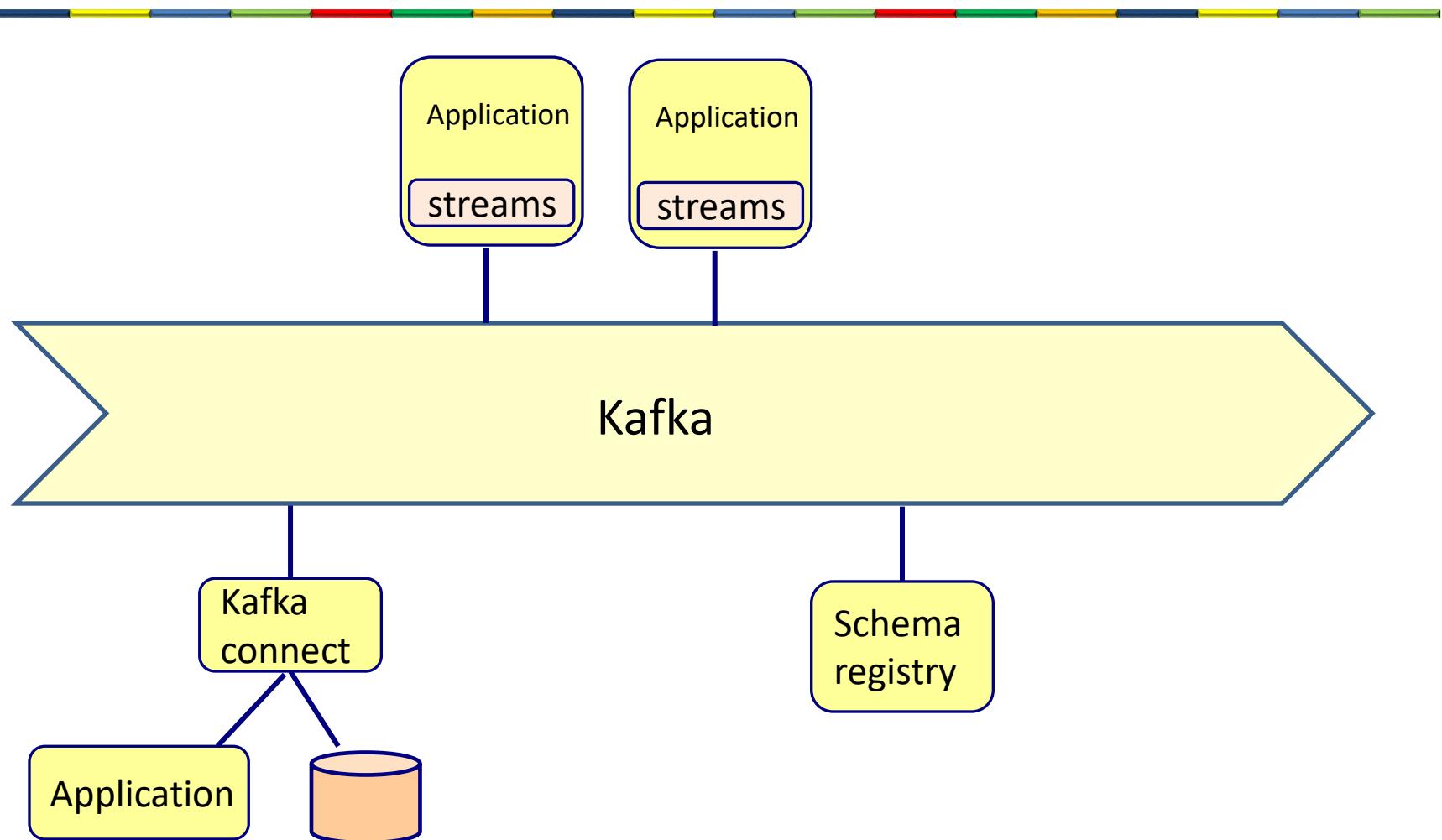
- What if the producer sends bad data?
 - What if a field gets renamed?
 - What if the data format changes ?
-
- The consumer breaks

Kafka does not verify the message

- Schema registry is a separate component (server)
- Maintains a database of schema's



Kafka ecosystem: Kafka streams



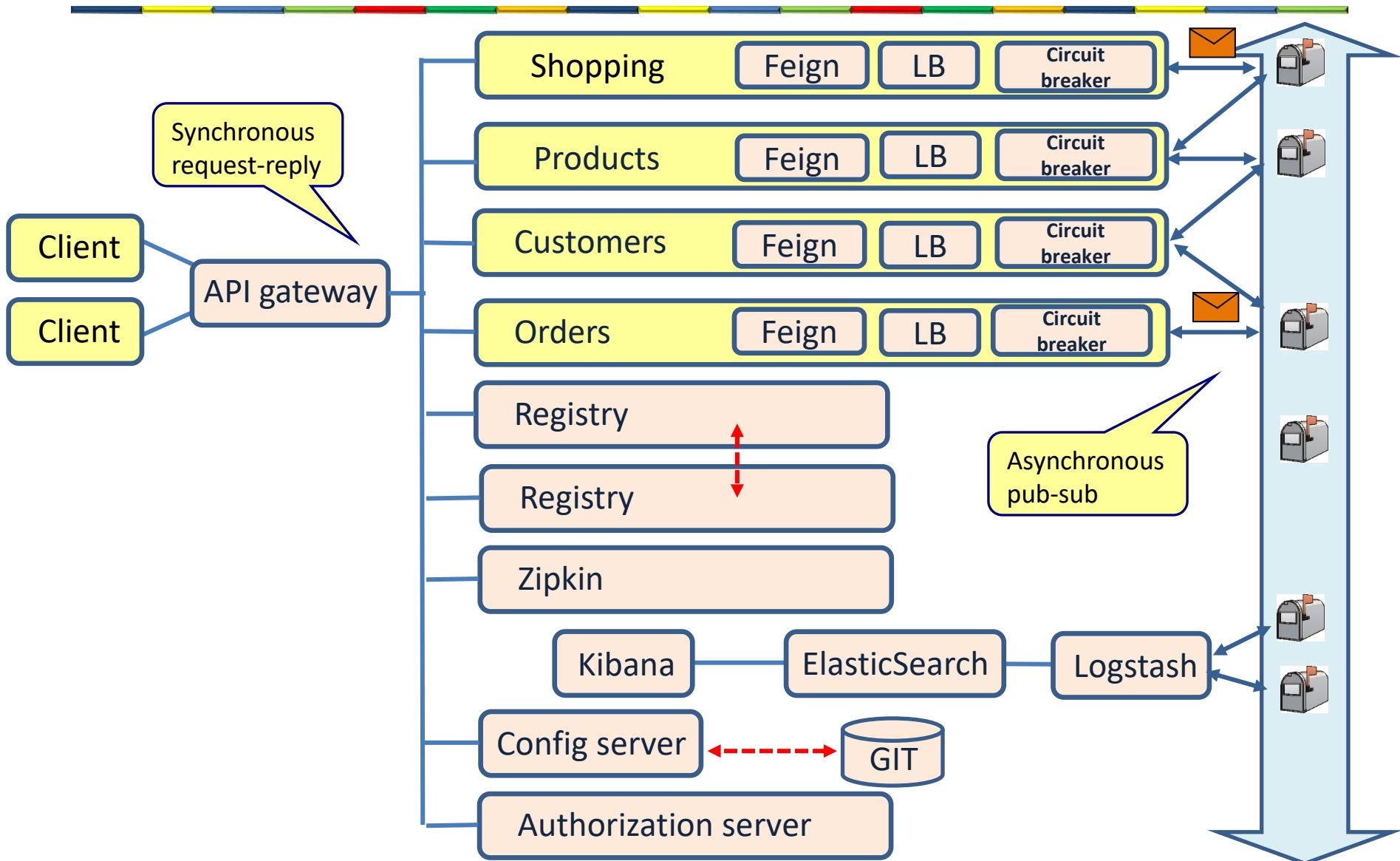
Kafka streams

- Java library for making stream processing simpler
 - Simple concise code
 - Threading and parallelism
 - Stream DSL (map, filter, aggregations, joins,...)

Kafka security

- Authentication
 - Are you allowed to access kafka?
 - SSL & SASL
 - Using certificates
- Authorization
 - Who is allowed to publish or consume which topic?
 - Access Control Lists (ACL)
- Encryption
 - Data sent is not readable by others
 - SSL
 - Only inflight security

Implementing microservices



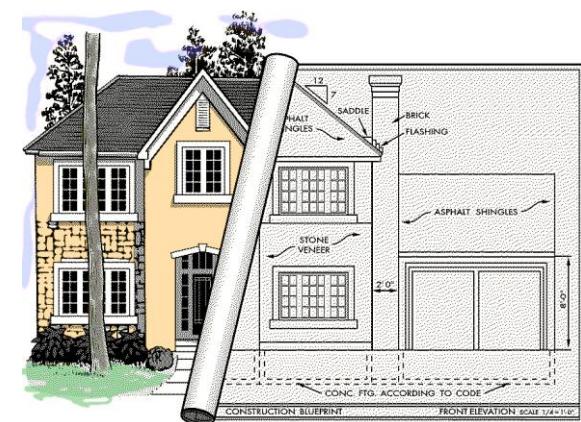
Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	Event Driven Architecture (EDA) CQRS
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	Token based security (OAuth2) Digitally signed (JWT) tokens
Transactions	Compensating transactions Eventual consistency
Keep data in sync	Publish-subscribe data change event
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	ELK + beats
Follow/monitor business processes	Zipkin ELK

ARCHITECTURE DIAGRAMS

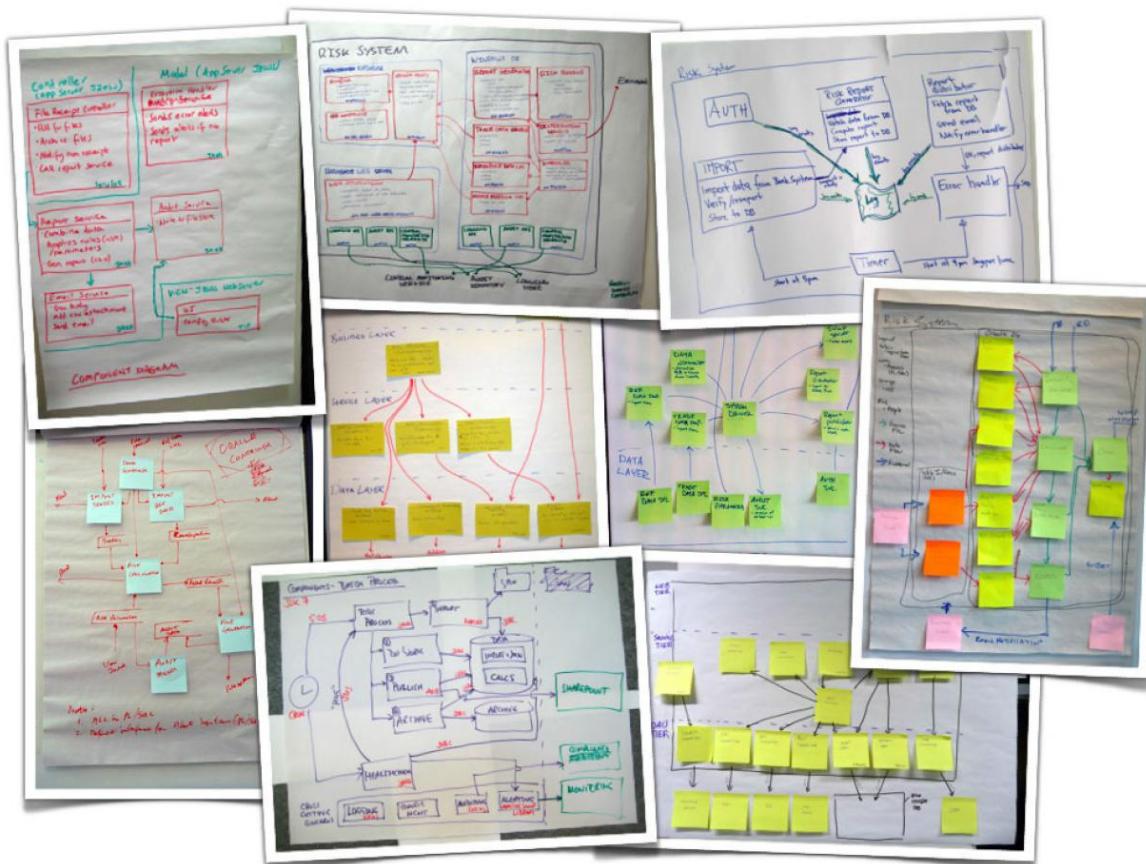
Architecture diagrams

- A picture says more than 1000 words
- Diagrams are meant to communicate the architectural important points
 - Diagrams are never ideal
 - Do not try to draw the ideal diagram
- Diagrams help us to
 - Convey the architecture
 - Evaluate the architecture
 - Discovery of other solutions



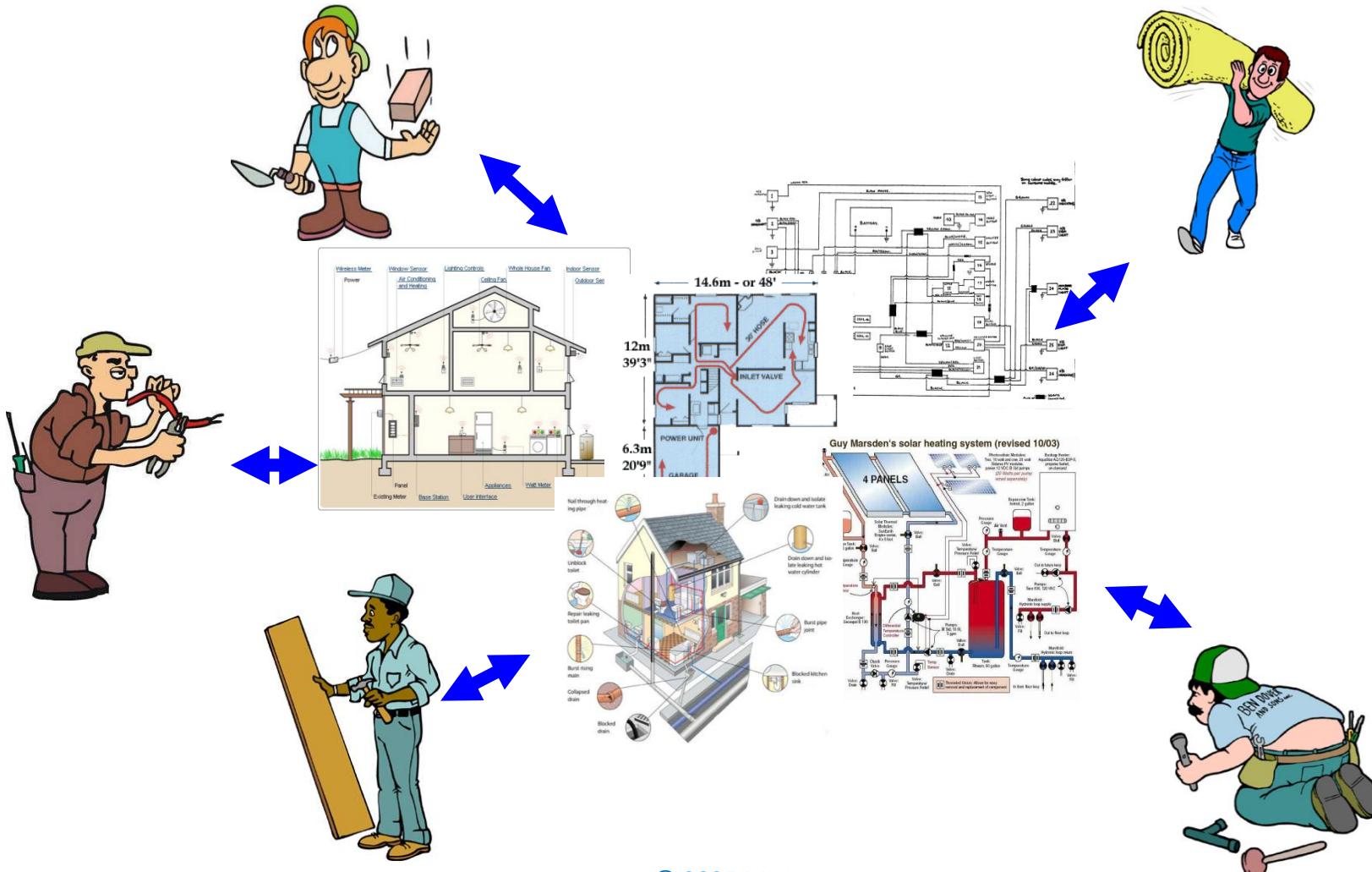
Architecture visibility

- Make the architecture visible with diagrams
 - Information radiator



Documenting architecture

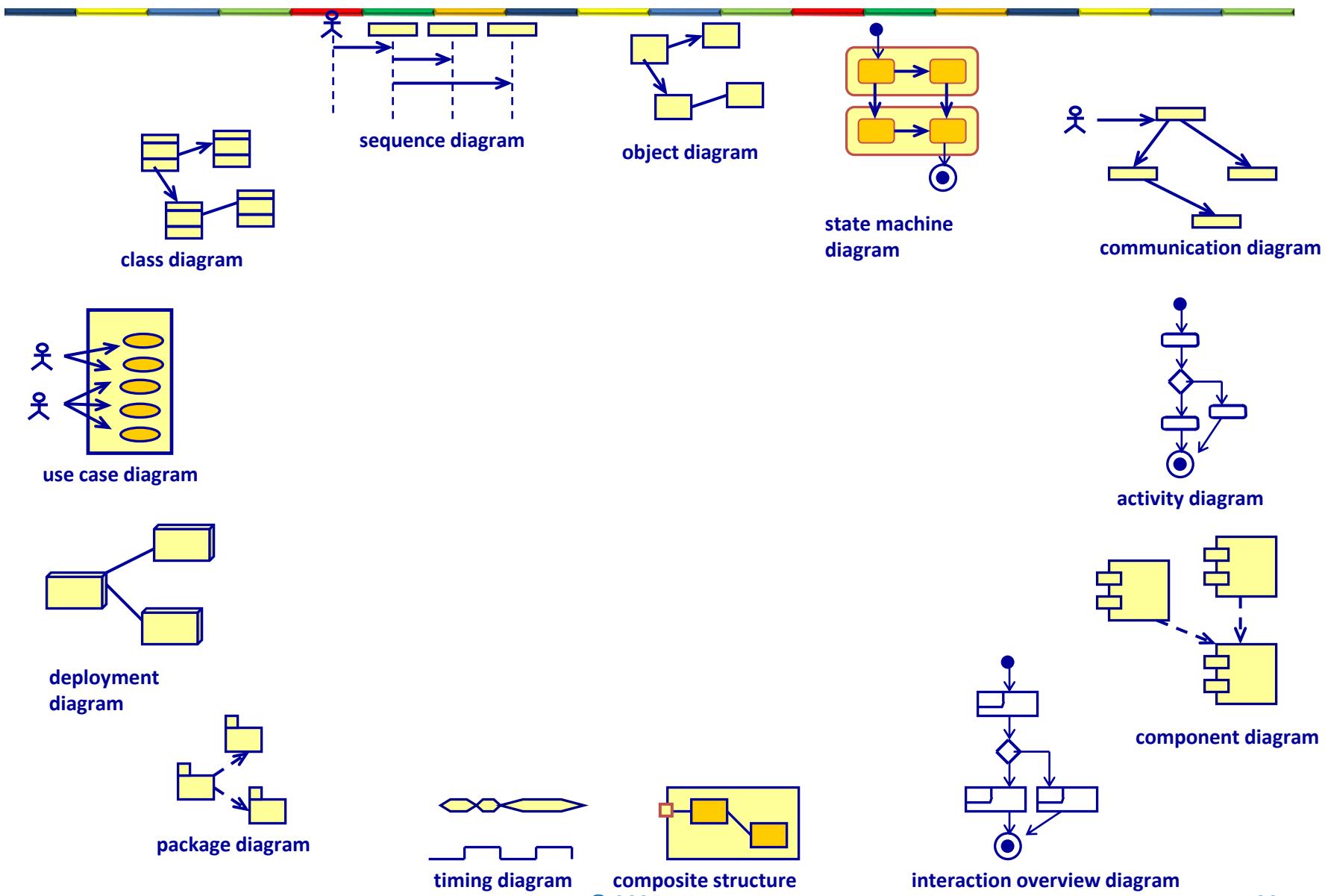
- Use different views



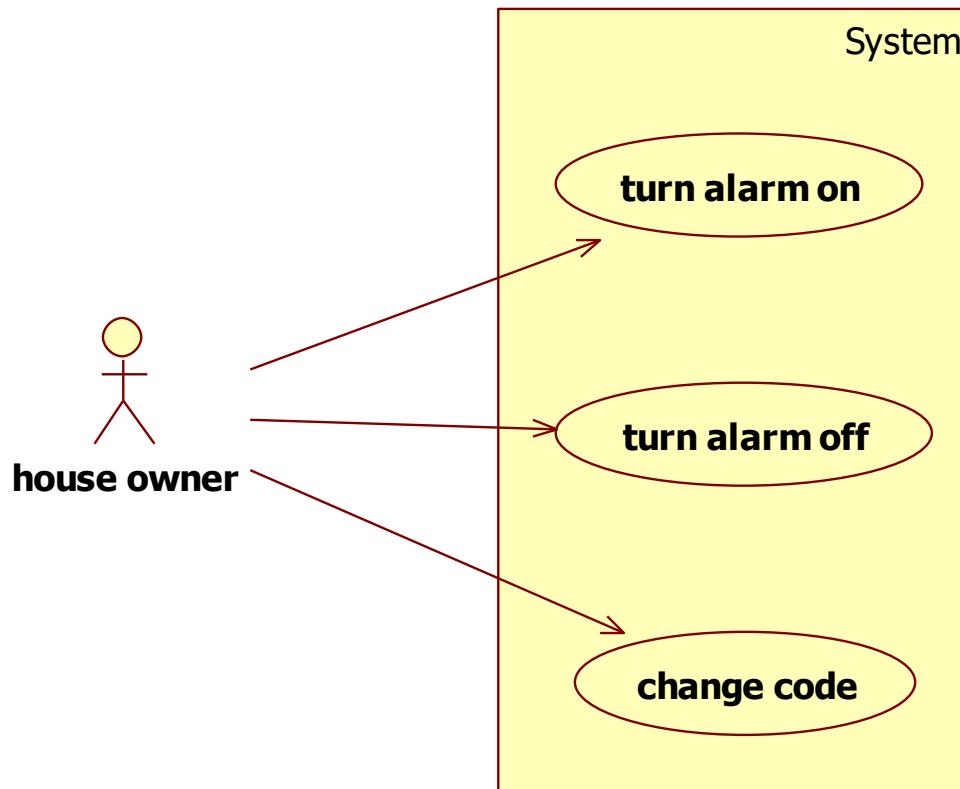
Which diagrams?

- Different diagram standards
 - UML
 - ArchiMate
 - C4 model
- They all have their own restrictions
- Better use an ad-hoc diagram using a ledger
- Different diagrams
 - Structural
 - Behavioral

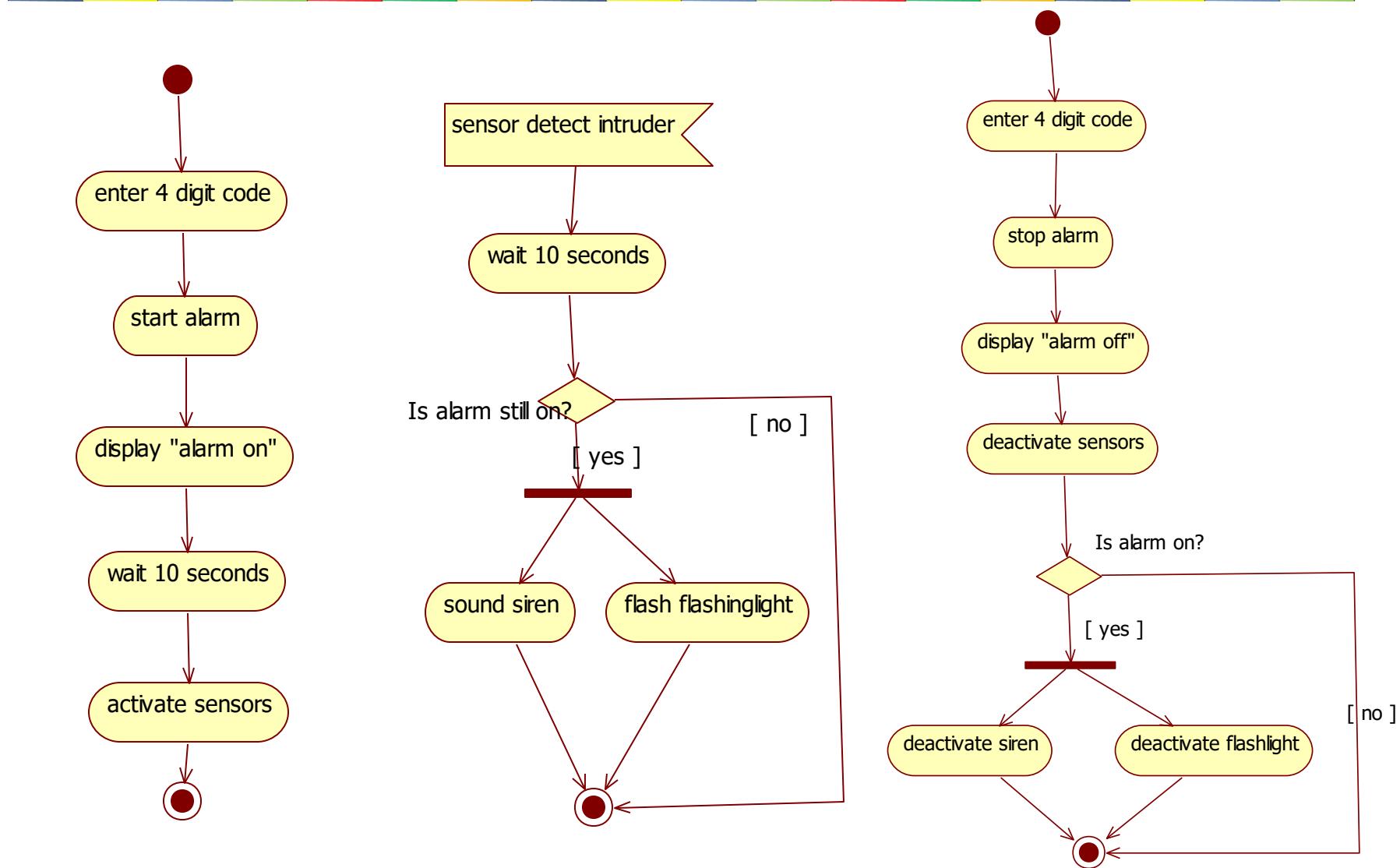
UML Diagrams



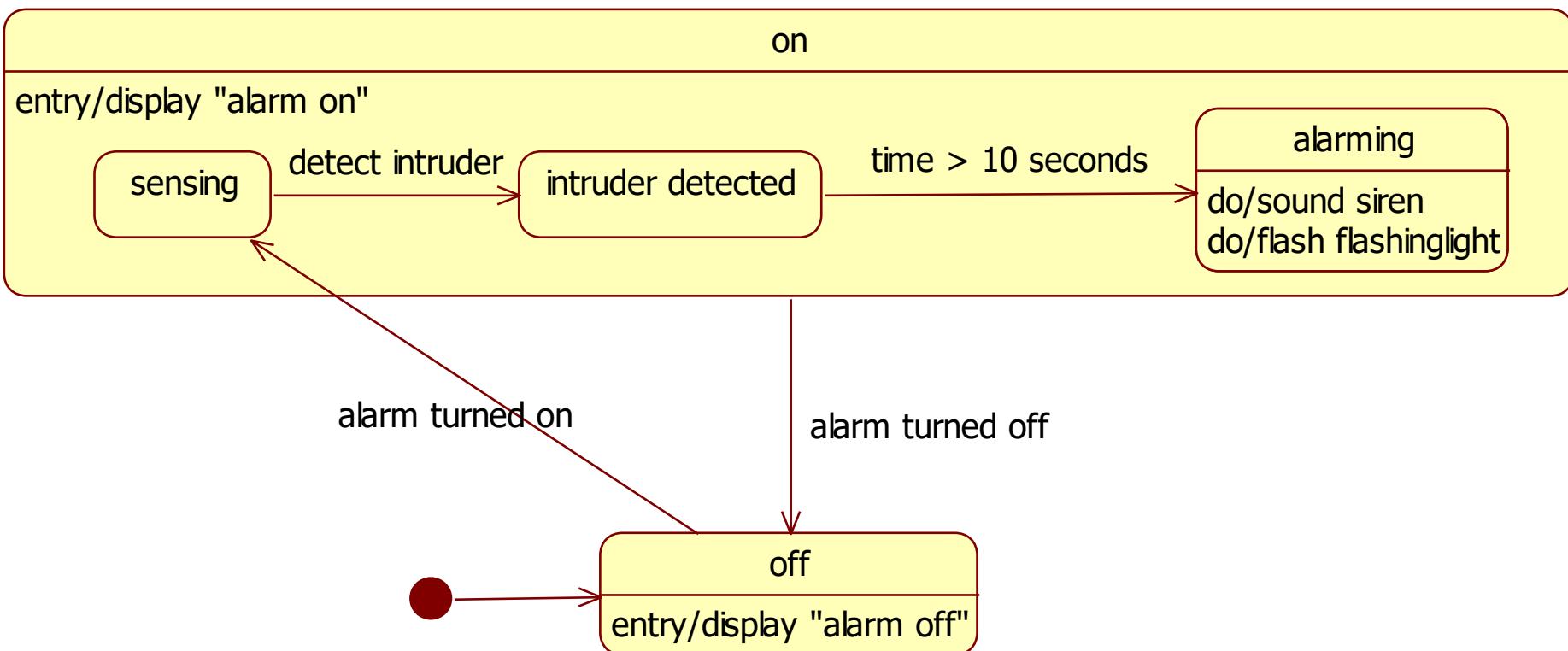
Home alarm system use case diagram



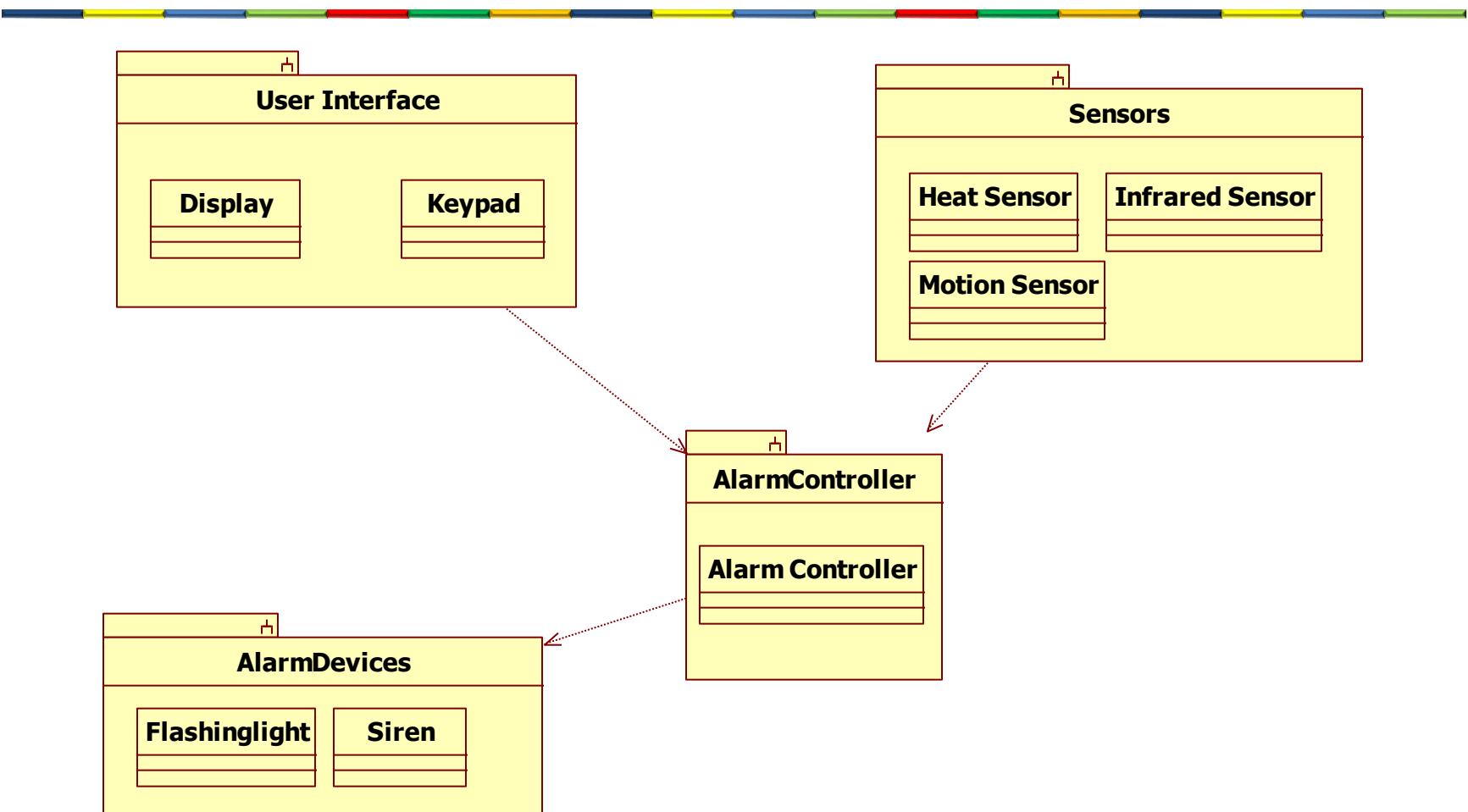
Home alarm system



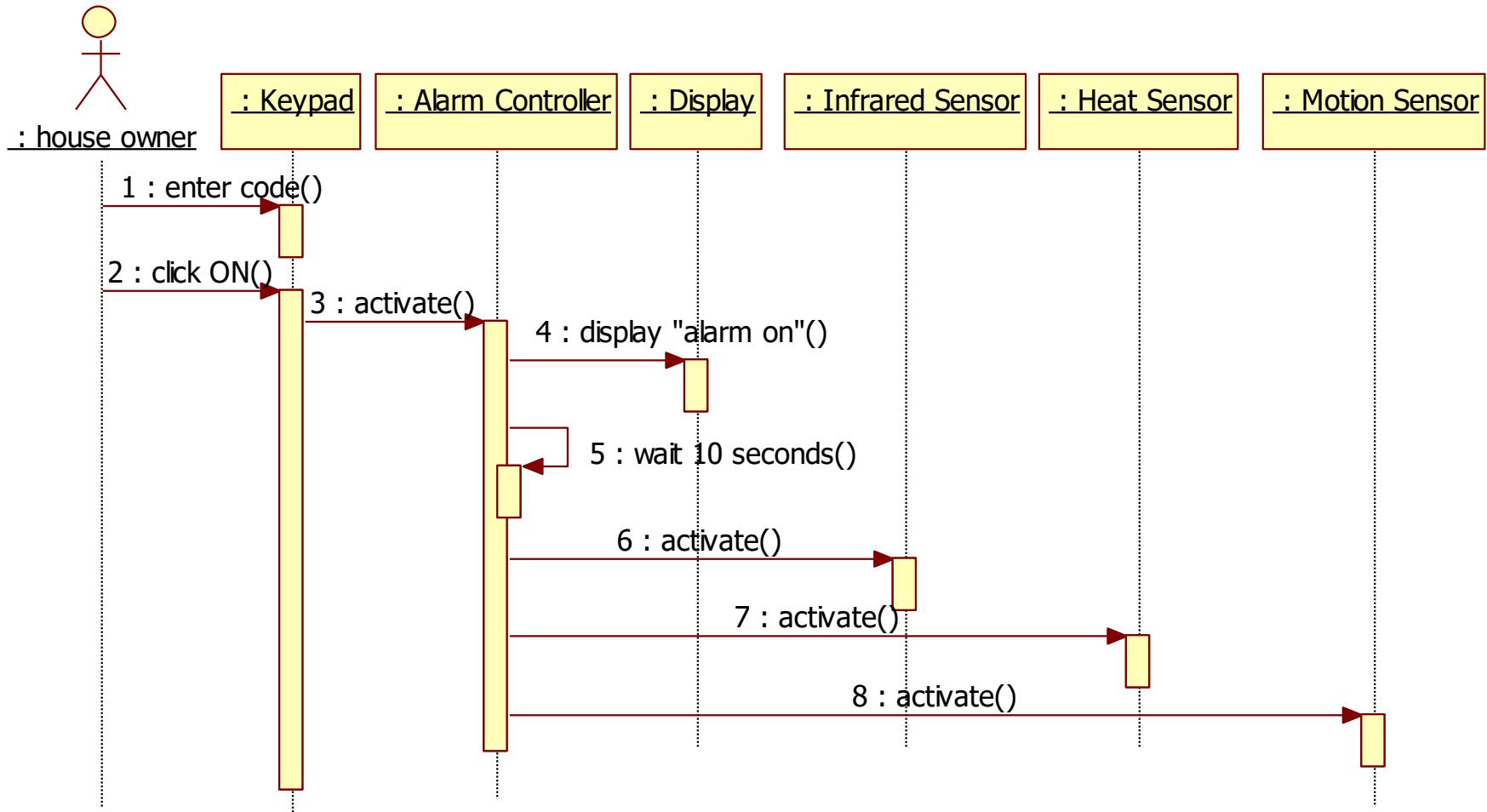
Home alarm system



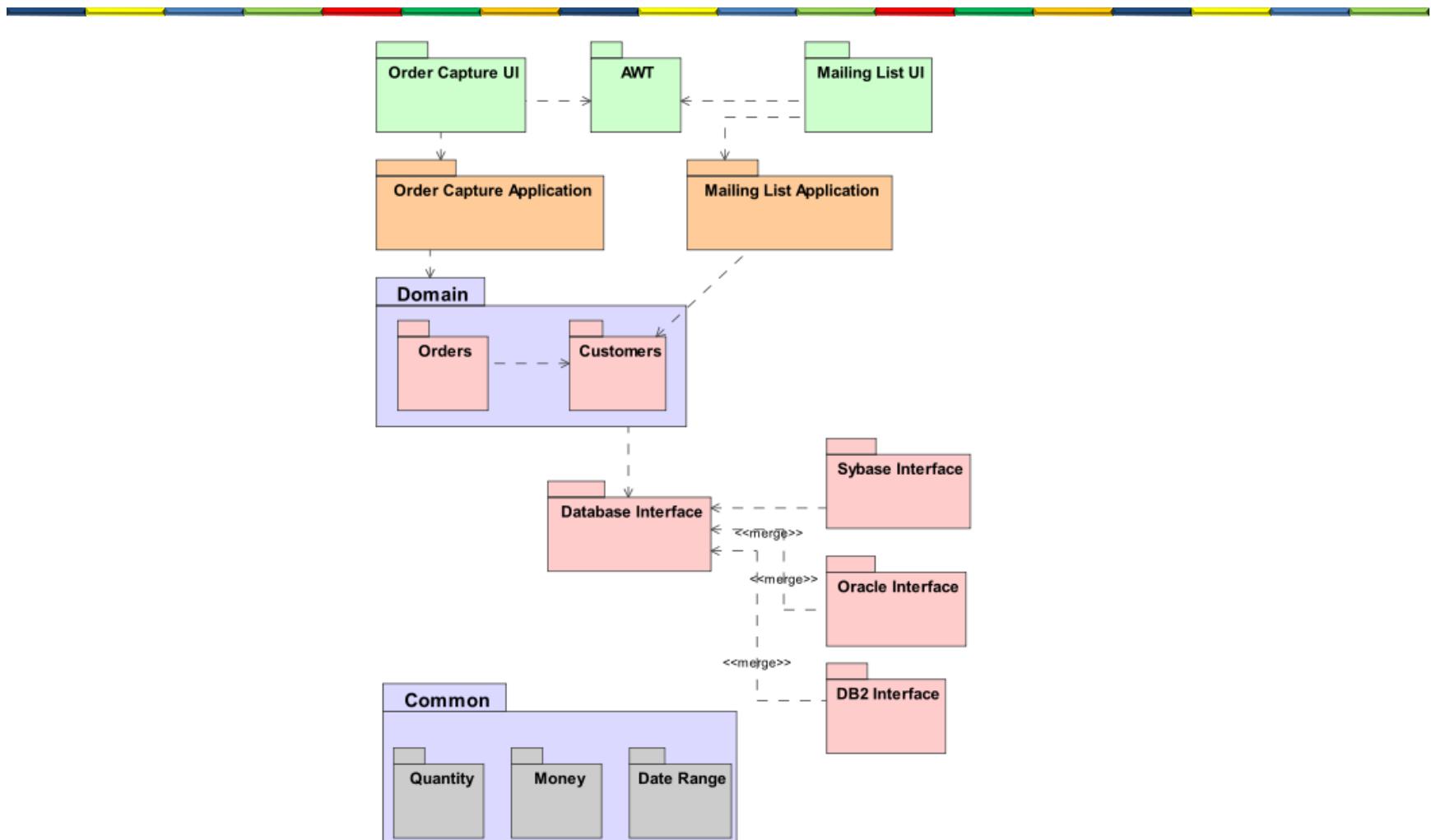
Home alarm system



Activate the alarm system

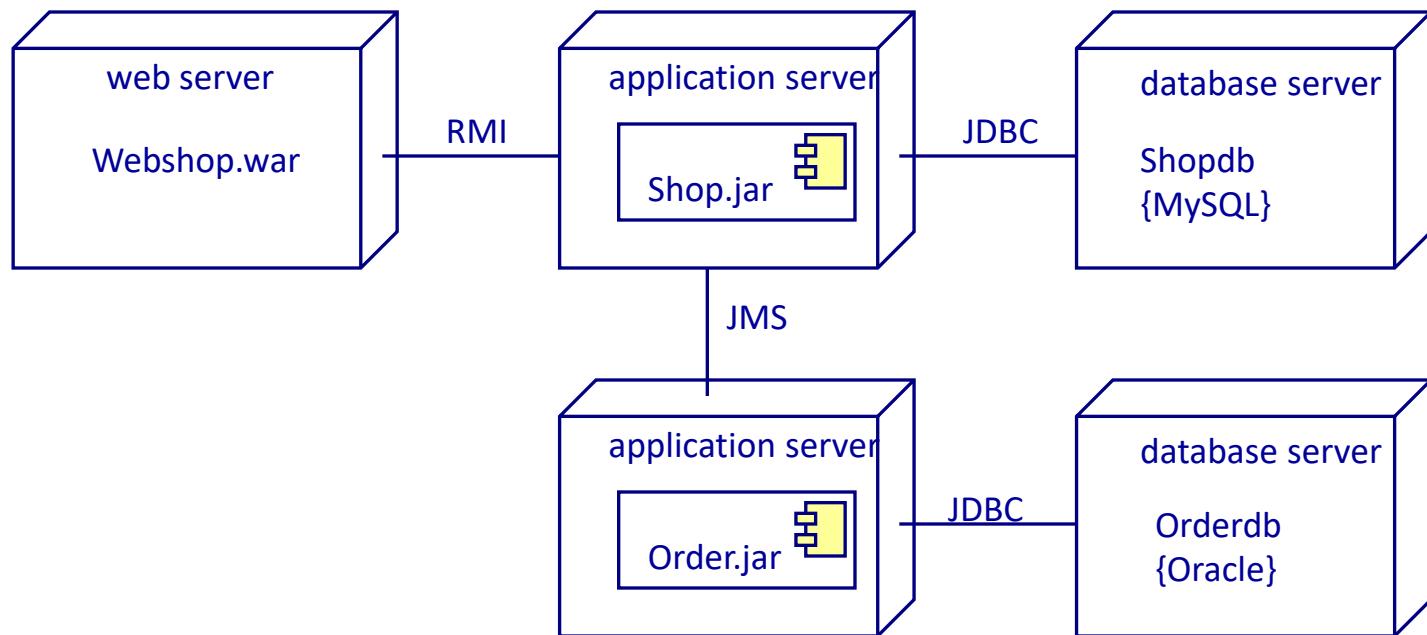


Package diagram



Describes how a system is split up into logical groupings and the dependencies among these groupings.

Deployment diagram



Describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.

4C model

Context diagram: Shows the big picture



Container diagram: shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.



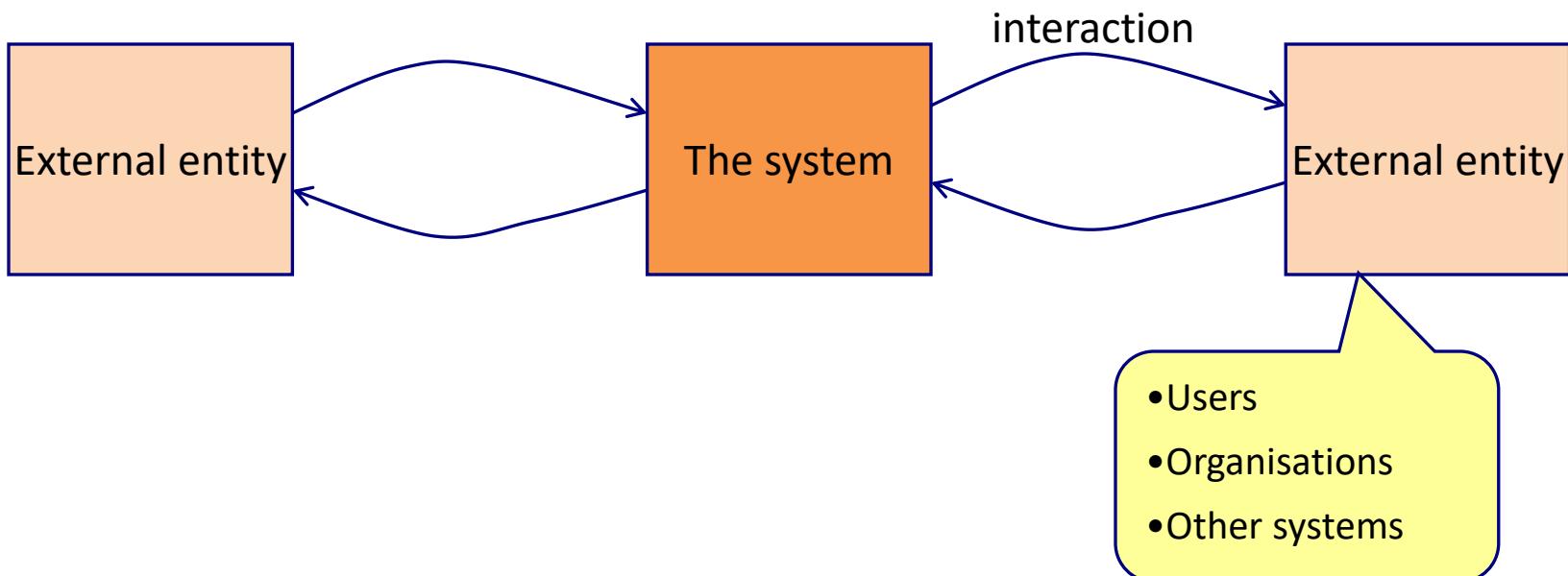
Component diagram: Decompose each container further into a number of distinct components, services, subsystems, layers, workflows, etc.



Detailed diagrams: Class diagram, sequence diagram, deployment diagram

Context diagram

- Shows the big picture
 - 10 km view



Context diagram

- Why?
 - It makes the context explicit so that there are no assumptions.
 - It shows what is being added (from a high-level) to an existing IT environment.
 - It's a high-level diagram that technical and non-technical people can use as a starting point for discussions.
 - It provides a starting point for identifying who you potentially need to go and talk to as far as understanding inter-system interfaces is concerned.

Should only take a couple of minutes to draw, so there really is no excuse not to do it

Example: Techtribes.je

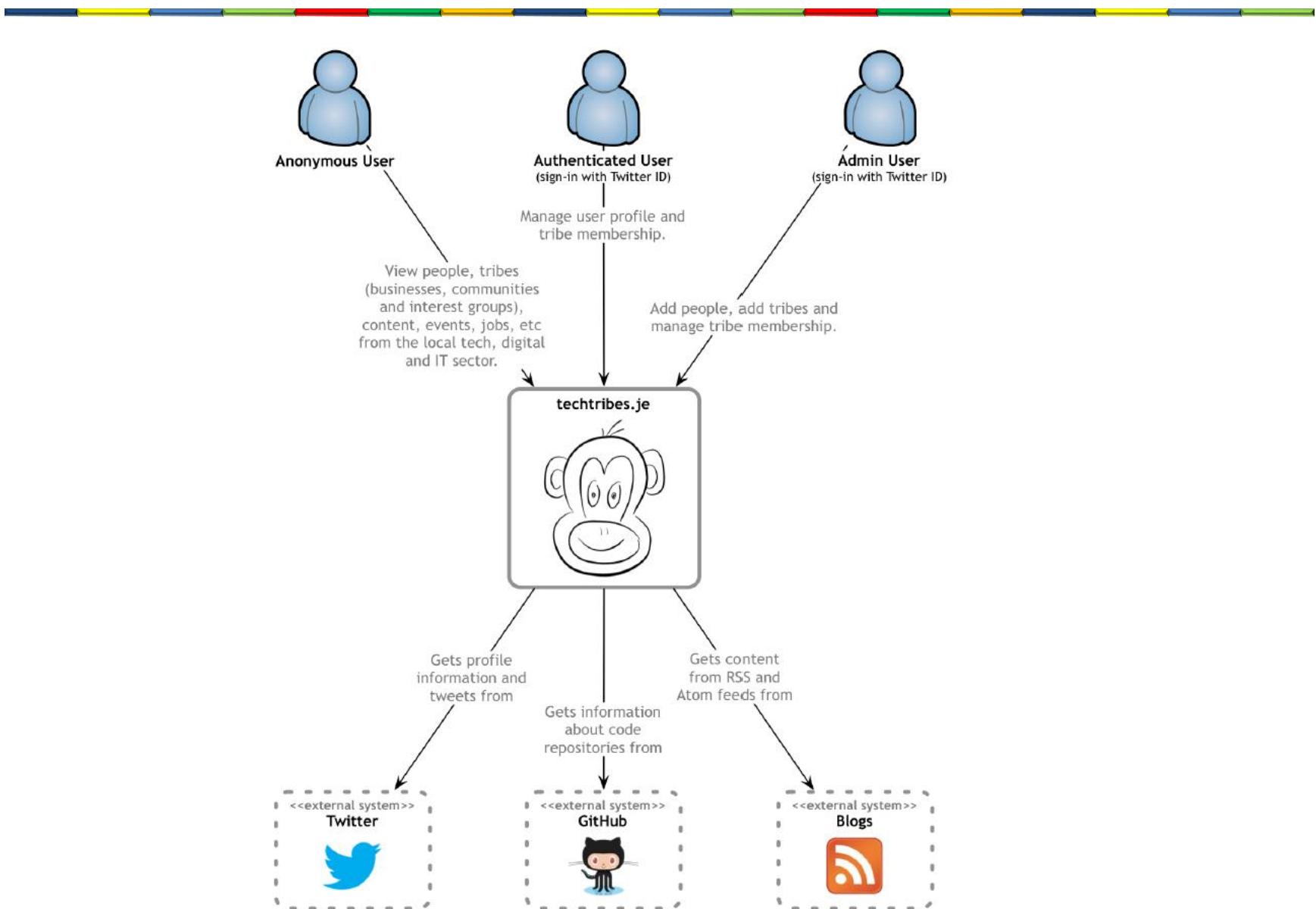
Website that provides a way to find people, tribes (businesses, communities, interest groups, etc) and content related to the tech, IT and digital sector in Jersey and Guernsey. At the most basic level, it's a content aggregator for local tweets, news, blog posts, events, talks, jobs and more.

The screenshot shows the homepage of the Techtribes.je website. At the top, there is a navigation bar with links for Bestand, Bewerken, Beeld, Geschiedenis, Bladwijzers, Extra, and Help. Below the navigation is a search bar with the placeholder "Find me people who know about..." and a "Search..." button. A "Fork me on GitHub" badge is visible on the right side of the header.

The main content area features several sections:

- Most active people:** Shows three profile pictures and a grid of smaller profile pictures below.
- Most active business tribes:** Shows icons for Prosperity+, C5 alliance, Cinnamon Edge, and others.
- Most active community tribes:** Shows icons for B, a stylized head, and other community symbols.
- News:** A list of news items including "An Intro to Bitcoin Mining & Investing" by Jonathan Day, "10 uses for Raspberry Pi in 10 minutes" by Rob Dudley, and "Data Protection & Defamation" by Vicky Milner. Each news item includes a thumbnail image and a YouTube video player.

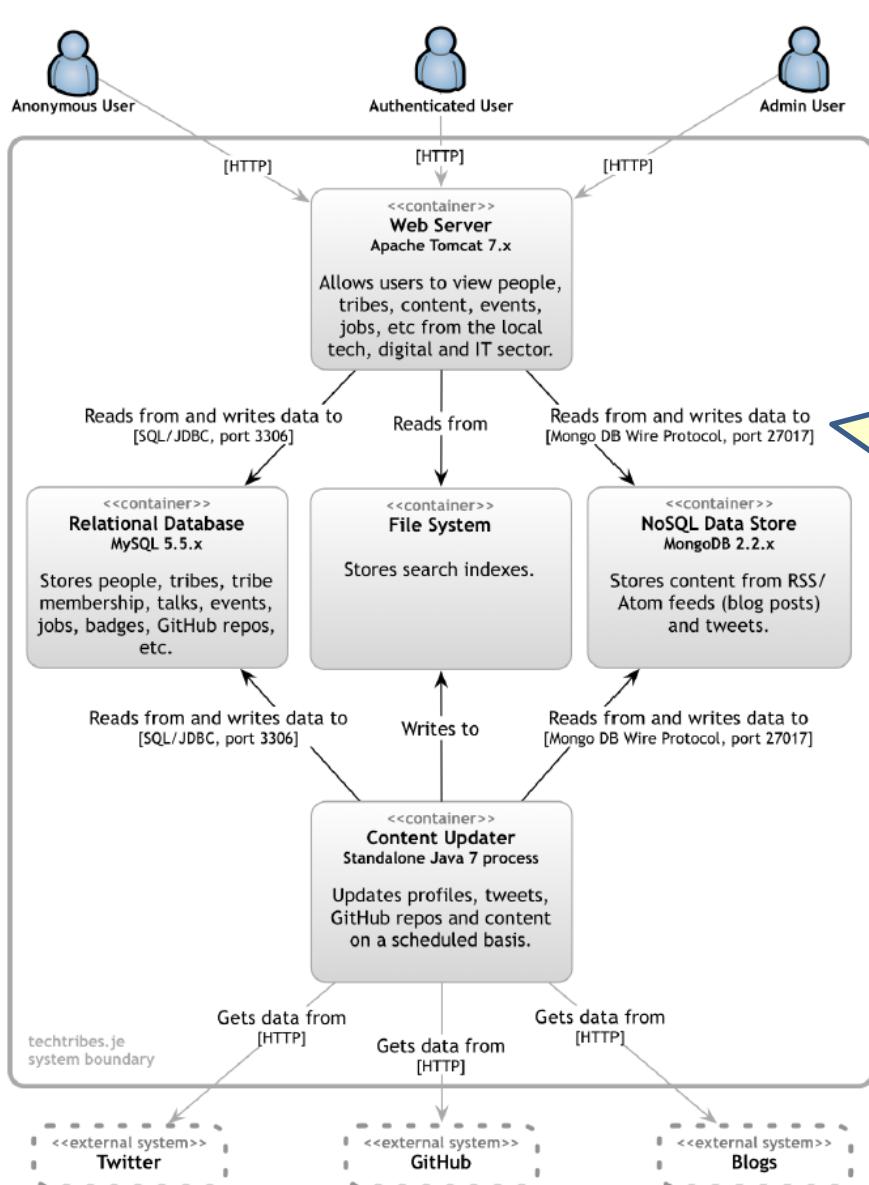
Techtribes.je context diagram



Container diagram

- Shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.
- Containers are the *logical* executables or processes that make up your software system
 - Web servers
 - Application servers
 - ESBs
 - Databases
 - Other storage systems
 - File systems
 - Windows services
 - Standalone/console applications
 - Web browsers
- For each container specify:
 - **Name:** “Web server”, “Database”, ...)
 - **Technology:** (e.g. Apache Tomcat 7, Oracle 11g, ...)
 - **Responsibilities:** A very high-level statement or list of the container’s responsibilities.
- Everything on a container diagram should potentially deployable separately

Techtribes.je container diagram



Describe the interactions :

- Purpose (“reads/writes data from”, “sends reports to”).
- Communication method (Web Services, REST, Java RMI, JMS).
- Communication style (synchronous, asynchronous, batched, two-phase commit)
- Protocols and port numbers (HTTP, HTTPS, SOAP/HTTP, SMTP, FTP).

Show the boundary
of what you are
building

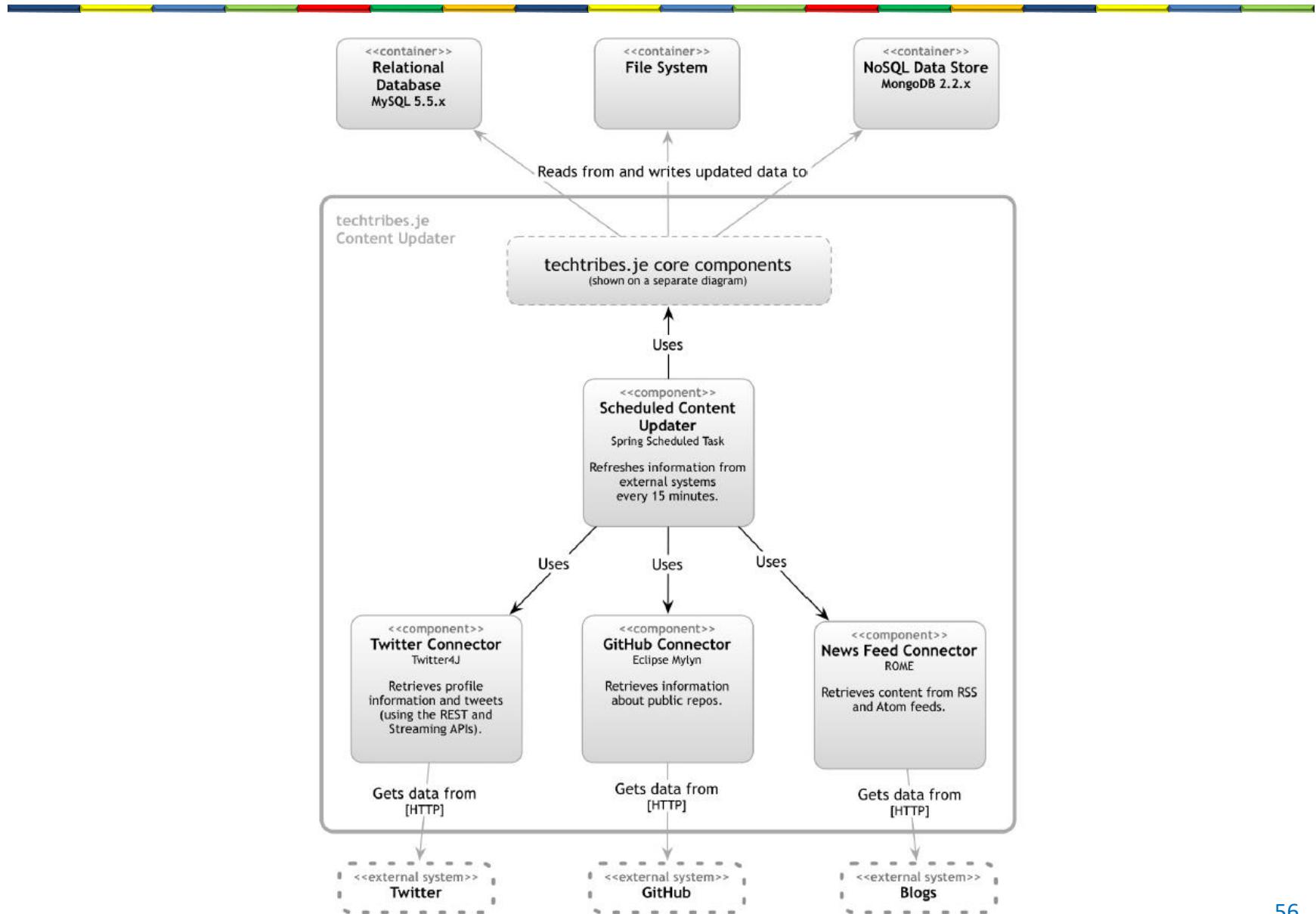
Container diagram

- Why?
 - It makes the high-level technology choices explicit.
 - It shows where there are relationships between containers and how they communicate.
 - It provides a framework in which to place components (i.e. so that all components have a home).
 - It provides the often missing link between a very high-level context diagram and (what is usually) a very cluttered component diagram showing all of the logical components that make up the entire software system.

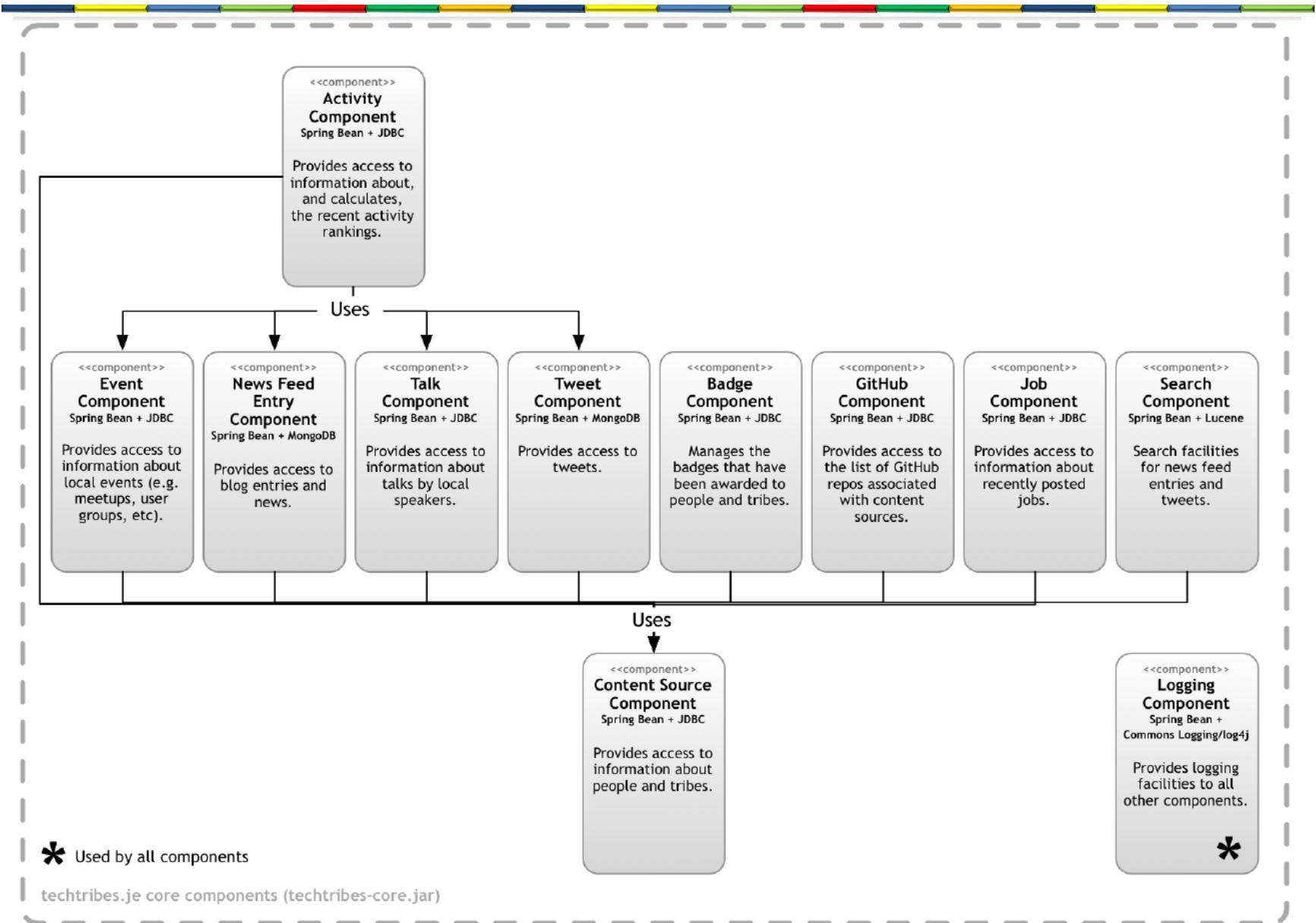
Component diagram

- Decompose each container further into a number of distinct components, services, subsystems, layers, workflows, etc.
- For each of the components drawn on the diagram, you could specify:
 - **Name:** The name of the component (“Risk calculator”, “Audit component”, etc).
 - **Technology:** The technology choice for the component (Plain Old Java Object, Enterprise JavaBean, etc).
 - **Responsibilities:** A very high-level statement of the component’s responsibilities (either important operation names or a brief sentence describing the responsibilities).

Component diagram for Content Updater



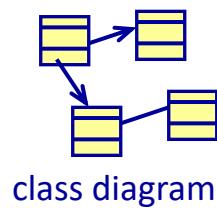
Core components



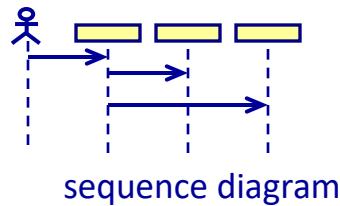
Component diagram

- Why?
 - It shows the high-level decomposition of your software system into components with distinct responsibilities.
 - It shows where there are relationships and dependencies between components.
 - It provides a framework for high-level software development estimates and how the delivery can be broken down.

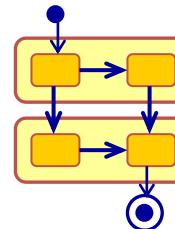
Detailed diagrams



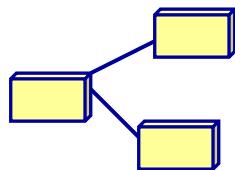
class diagram



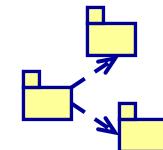
sequence diagram



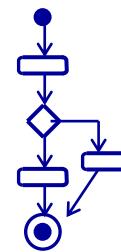
state machine
diagram



deployment
diagram

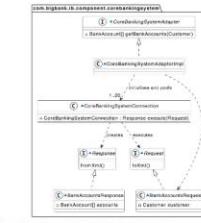
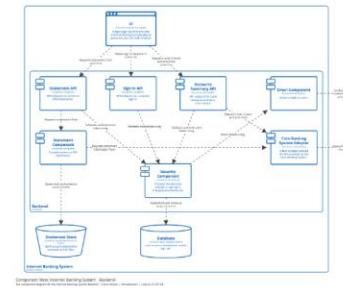
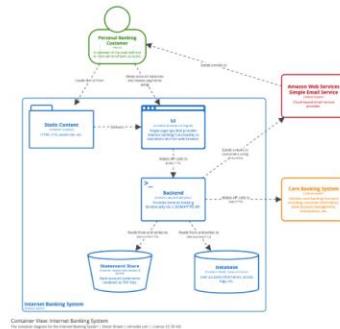
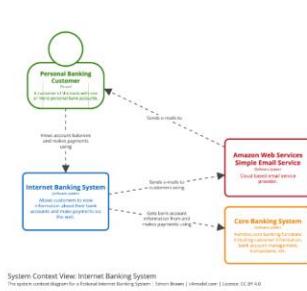


package
diagram



activity diagram

C4



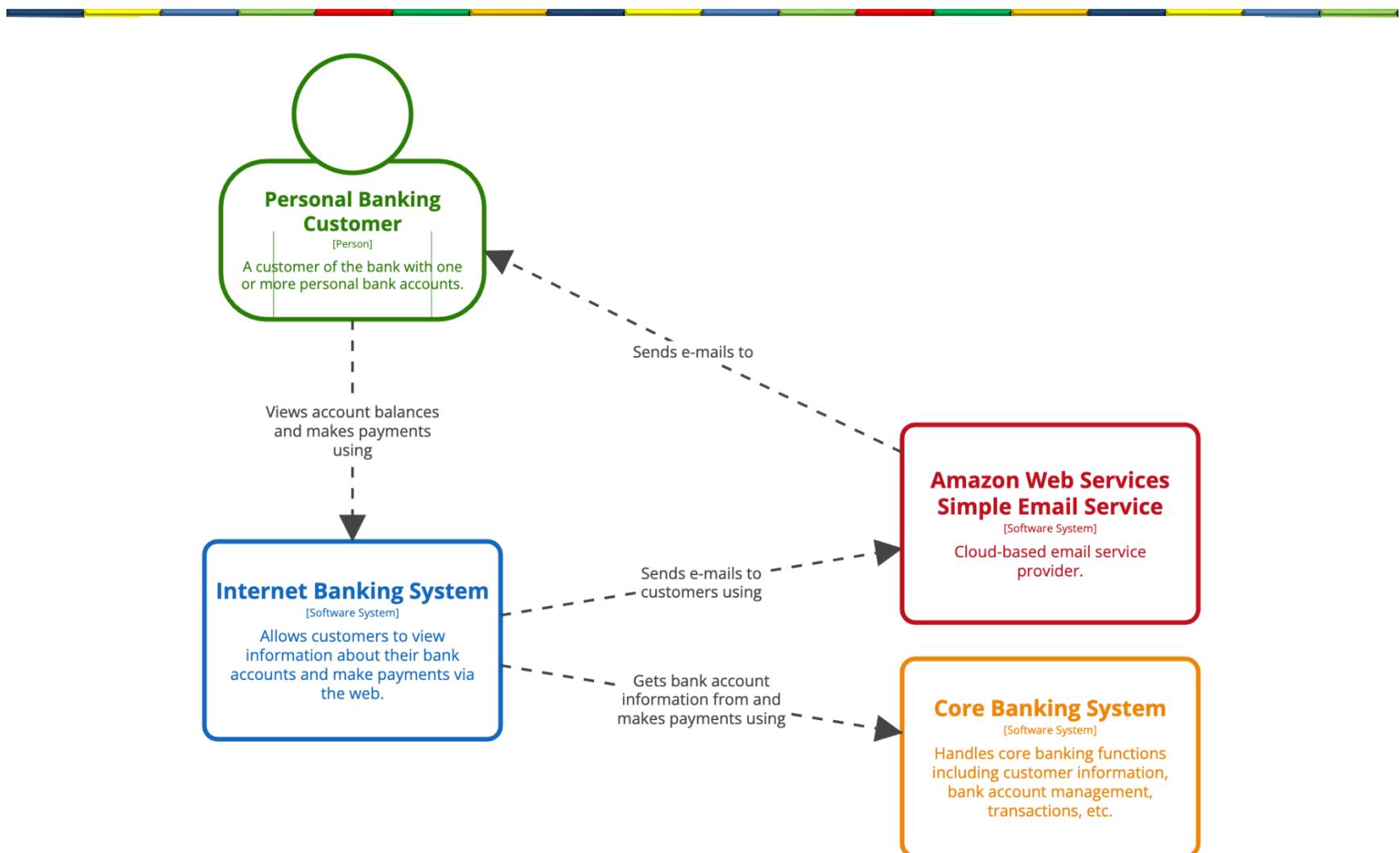
A system context diagram provides a starting point, showing how the software system in scope fits into the world around it.

A container diagram zooms into the software system in scope, showing the applications and data stores inside it.

A component diagram zooms into an individual container, showing the components inside it.

A code diagram (e.g. UML class) can be used to zoom into an individual component, showing how that component is implemented at the code level.

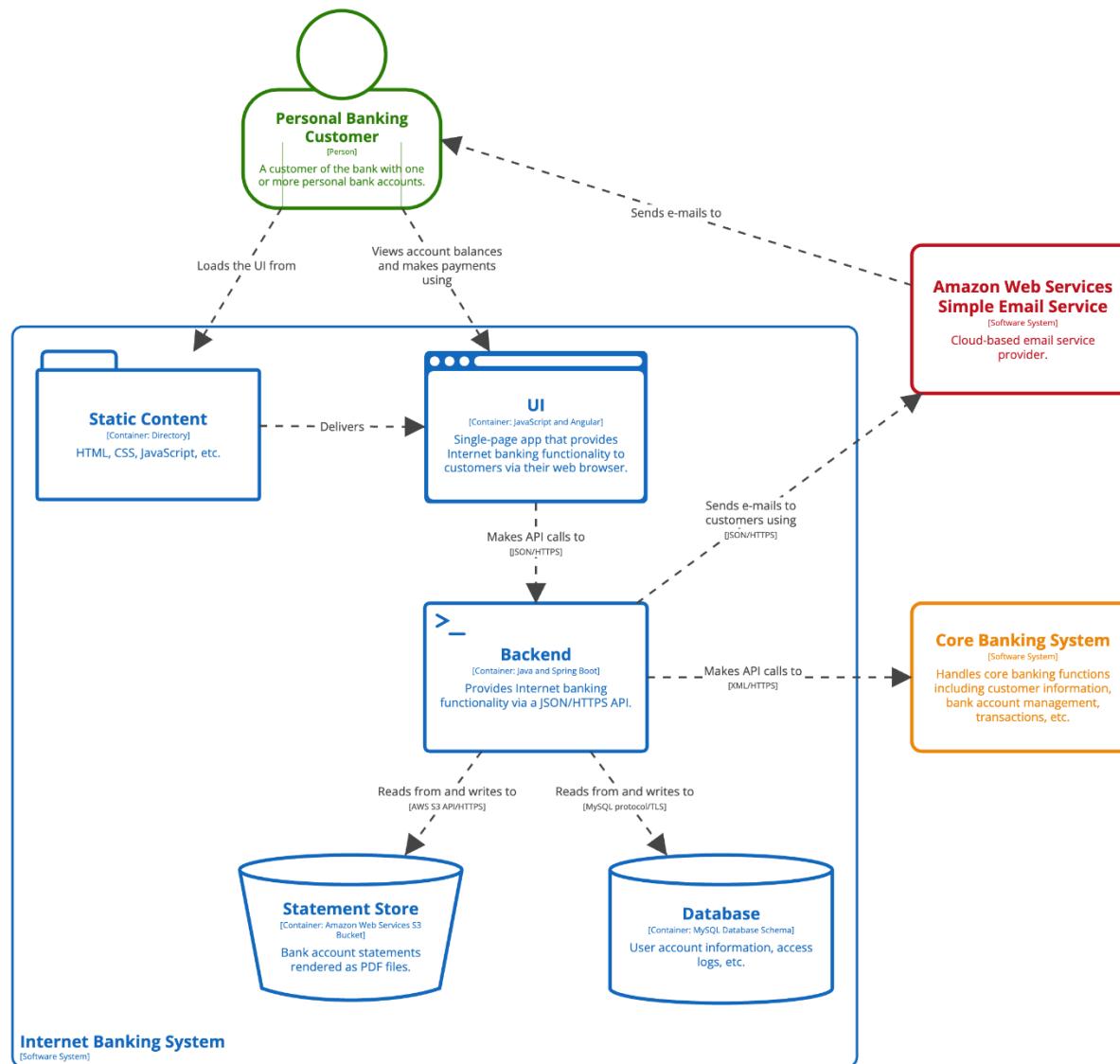
Context diagram



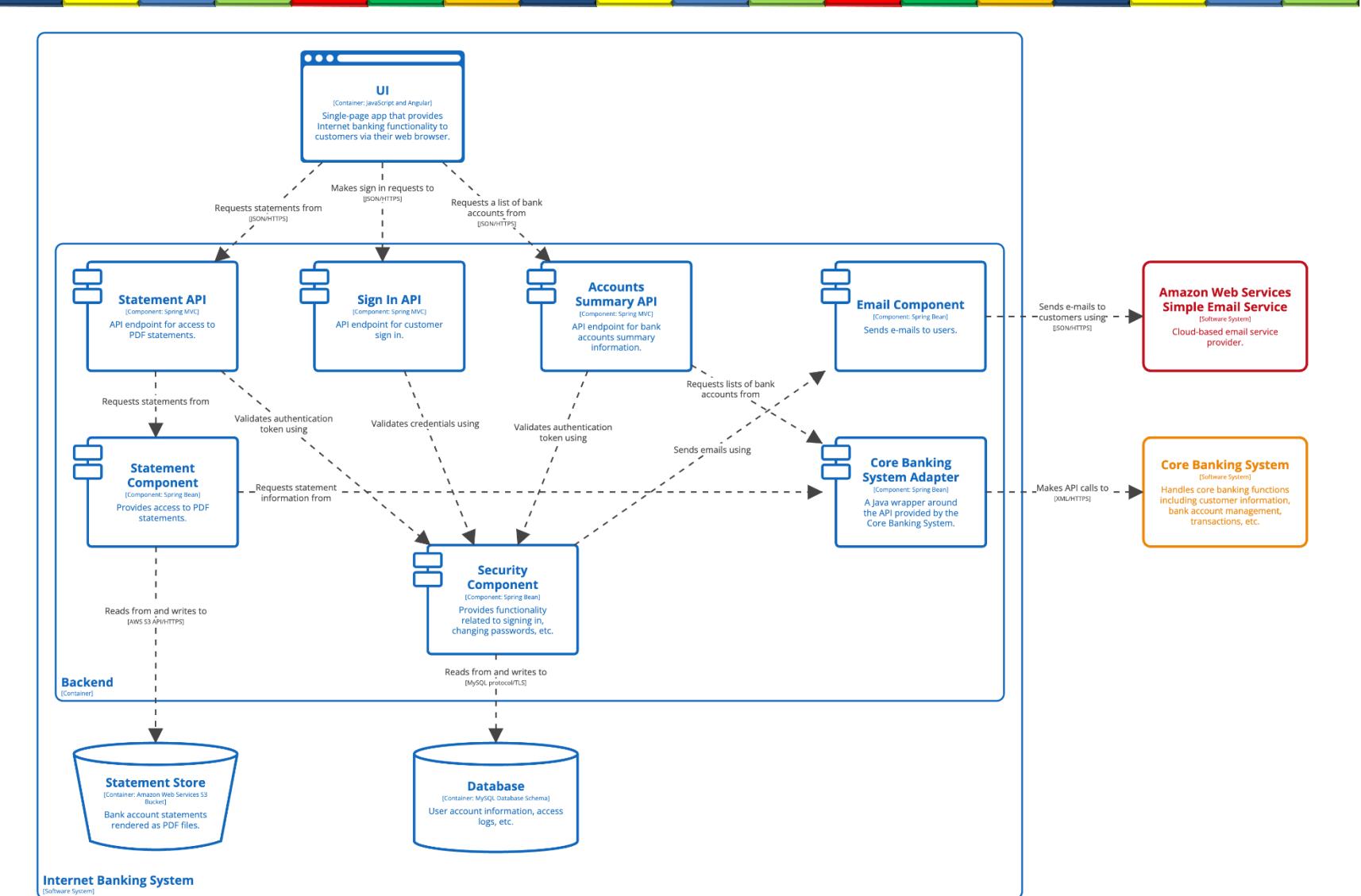
System Context View: Internet Banking System

The system context diagram for a fictional Internet Banking System | Simon Brown | c4model.com | License: CC BY 4.0

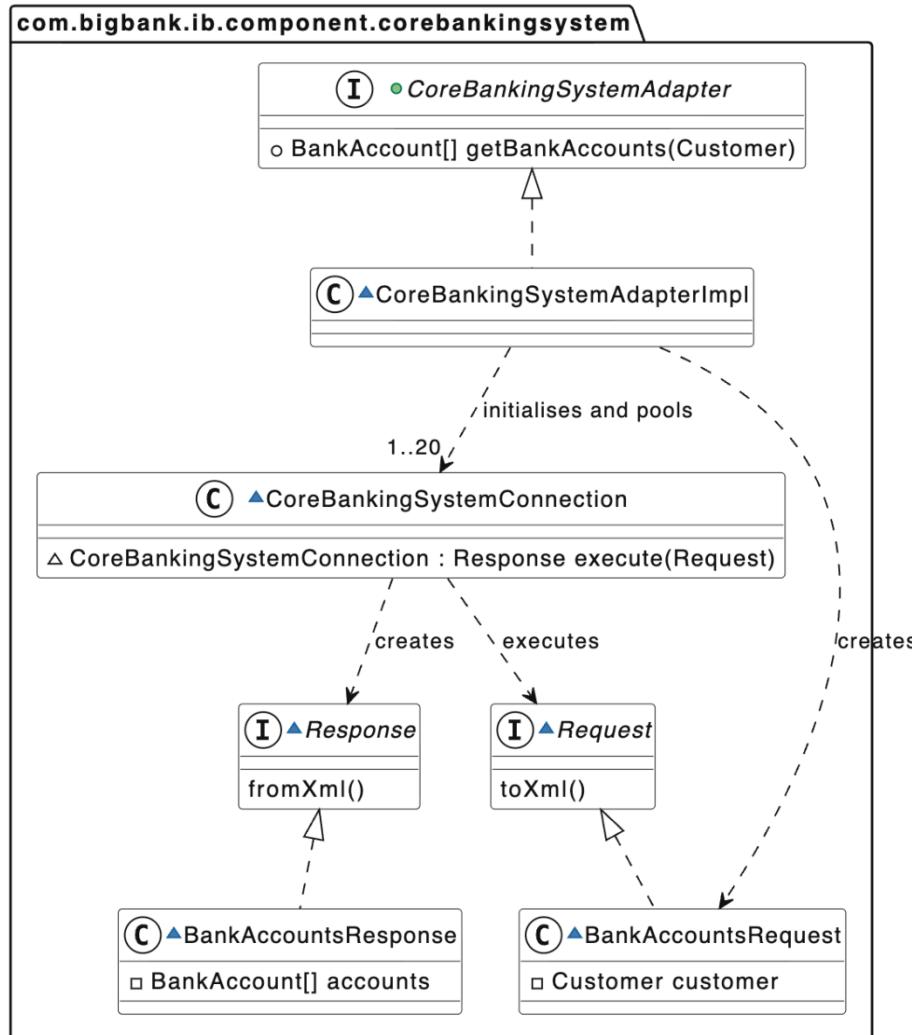
Container diagram



Component diagram



Code diagram (Class diagram)



Main point

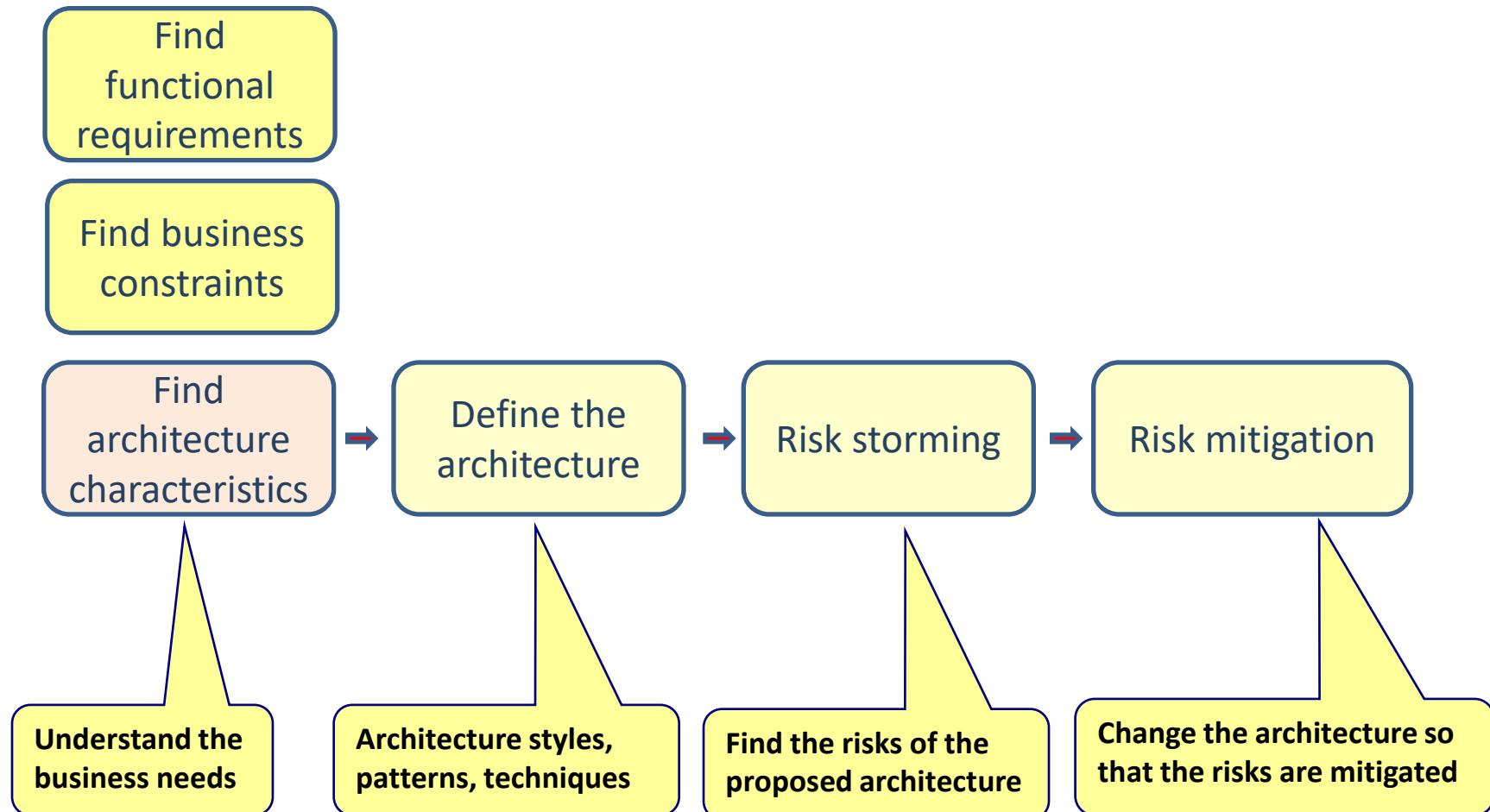
- The 4C model helps us to communicate architecture at different levels of abstraction.

Science of Consciousness: Knowledge if different in different states of consciousness.

Lesson 13

FINDING THE RIGHT ARCHITECTURE

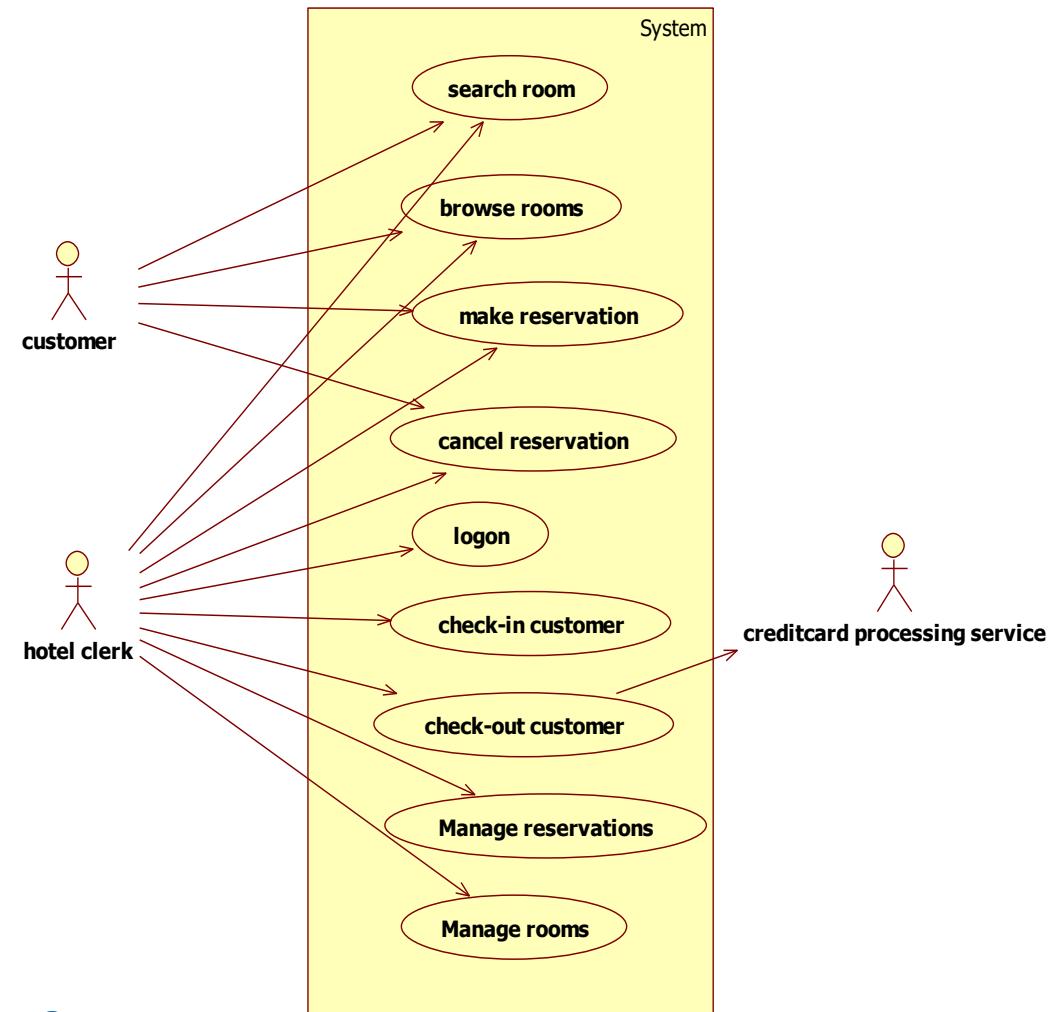
Finding the right architecture



Functional requirements

- What should the system functionally do?
 - Use cases
 - User stories

As a customer
I can view my account history
so that I know all transactions on my
account



Business constraints

- Constraints from the business (or enterprise architecture
 - We do everything in .Net
 - We always use an Oracle database
 - Our maintenance engineers all know Java
 - All applications talk with the Oracle ESB
- Budget
- Deadlines

ARCHITECTURAL CHARACTERISTICS

Architectural characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

Wikipedia software qualities



- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composableity [Erl]
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

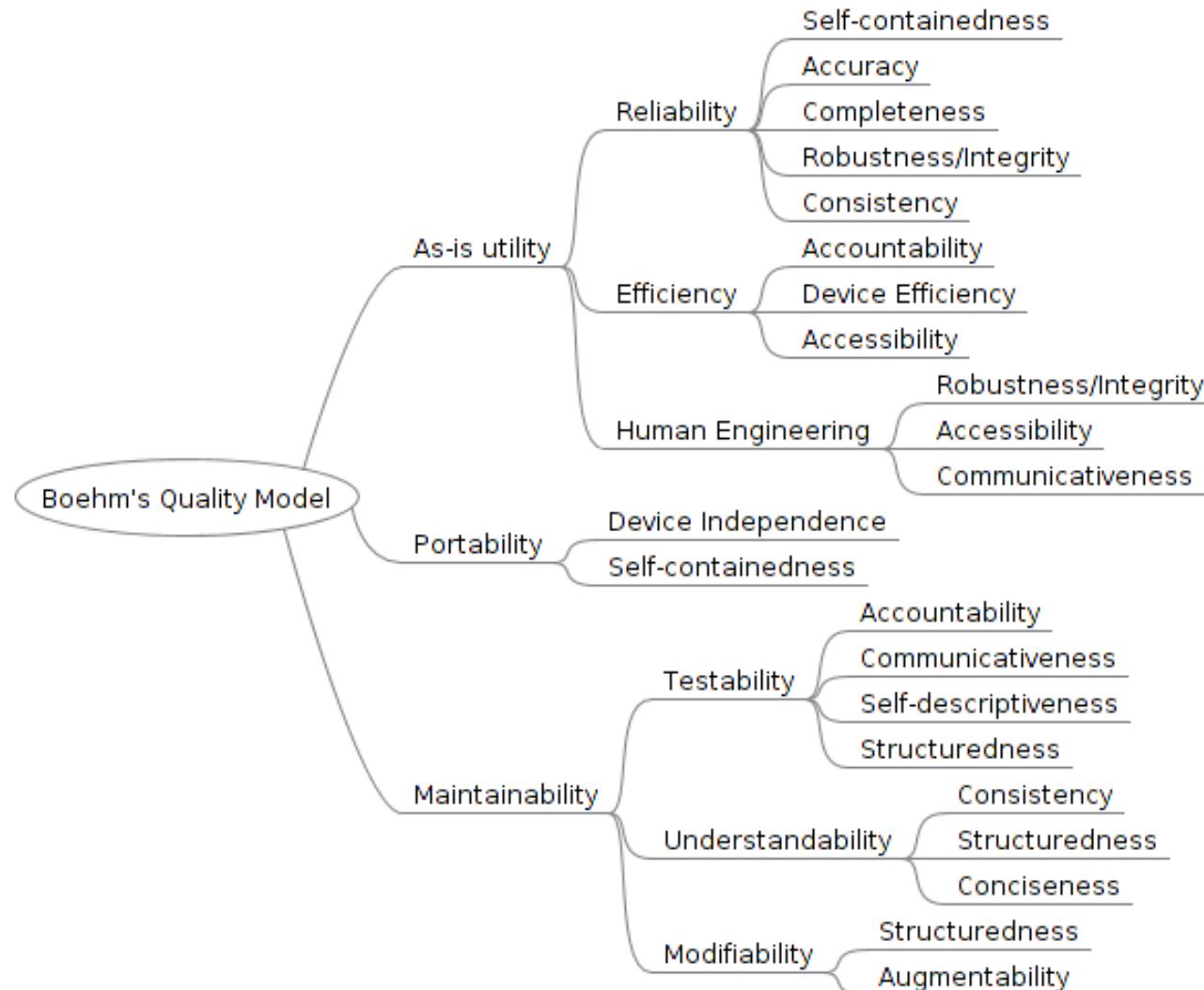
SEI quality model

Qualities noticeable at runtime	Performance Responsiveness of the system
	Security Ability to resist unauthorized usage
	Availability Portion of time the system is available
	Functionality Ability to do intended work
	Usability Learnability, efficiency, satisfaction, error handling, error avoidance
Qualities not noticeable at runtime	Modifiability Cost of introducing change
	Portability Ability to operate in different computing environments
	Reusability Ability to reuse components in different applications
	Integrability Ability that components work correctly together
	Testability Ability to systematic testing to discover defects

FURPS model

- **Functionality** - evaluate the feature set and capabilities of the program, the generality of the functions delivered and the security of the overall system
- **Usability** - consider human factors, overall aesthetics, consistency, and documentation
- **Reliability** - measure the frequency and severity of failure, the accuracy of outputs, the ability to recover from failure, and the predictability
- **Performance** - measure the processing speed, response time, resource consumption, throughput and efficiency
- **Supportability** - measure the maintainability, testability, configurability and ease of installation

Boehm



ISO 25010



Organize architectural characteristics

Software development process

- **Modifiability** – Ease with which the system can be changed or extended.
- **Testability** – How easily the system's behavior can be tested and verified.
- **Reusability** – Degree to which components can be reused in other systems.
- **Maintainability** – Effort required to identify and fix issues or improve the system.
- **Deployability** – How easily, reliably, and frequently the system can be released and installed into target environments.

Structure of the system

- **Modularity** – Division of the system into independent, cohesive components.
- **Portability** – How easily the system can operate in different environments with minimal changes.
- **Extensibility** – How easily new features can be added to the system without affecting existing functionality.

Operation of the system

- **Performance** – How quickly the system responds and processes data.
- **Availability** – Degree to which the system is operational and accessible when needed.
- **Reliability** – Ability of the system to function correctly over time without failure.
- **Scalability** – Capacity to handle increased load or usage without degradation.
- **Recoverability** – How quickly the system can restore normal operations after a failure or disruption.
- **Robustness** – How well the system continues to function correctly under stress, invalid inputs, or unexpected conditions.

Crosscutting

- **Security** – Protection of the system and its data from unauthorized access, misuse, or attacks.
- **Accessibility** – Ease with which people of all abilities can use and interact with the system.
- **Usability** – How easily and efficiently users can learn, understand, and operate the system.
- **Privacy** – Safeguarding of personal or sensitive information from unauthorized collection, use, or disclosure.

Architectural characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

Balance the qualities

- More security through encryption lowers performance
- More scalability through clustering lowers performance
- More scalability through clustering increases the cost



Find the top 5(+/- 2) qualities

Architectural characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

Quality scenario's

- A quality on itself has little meaning
- Create scenario's for the top qualities
- Make scenario's measurable
 - The system should be able to scale to 1000 concurrent users
 - The system should be available 24/7
 - All user actions should give a response within 3 seconds.
- Prioritize the scenario's
- Write acceptance tests for NFR scenario's



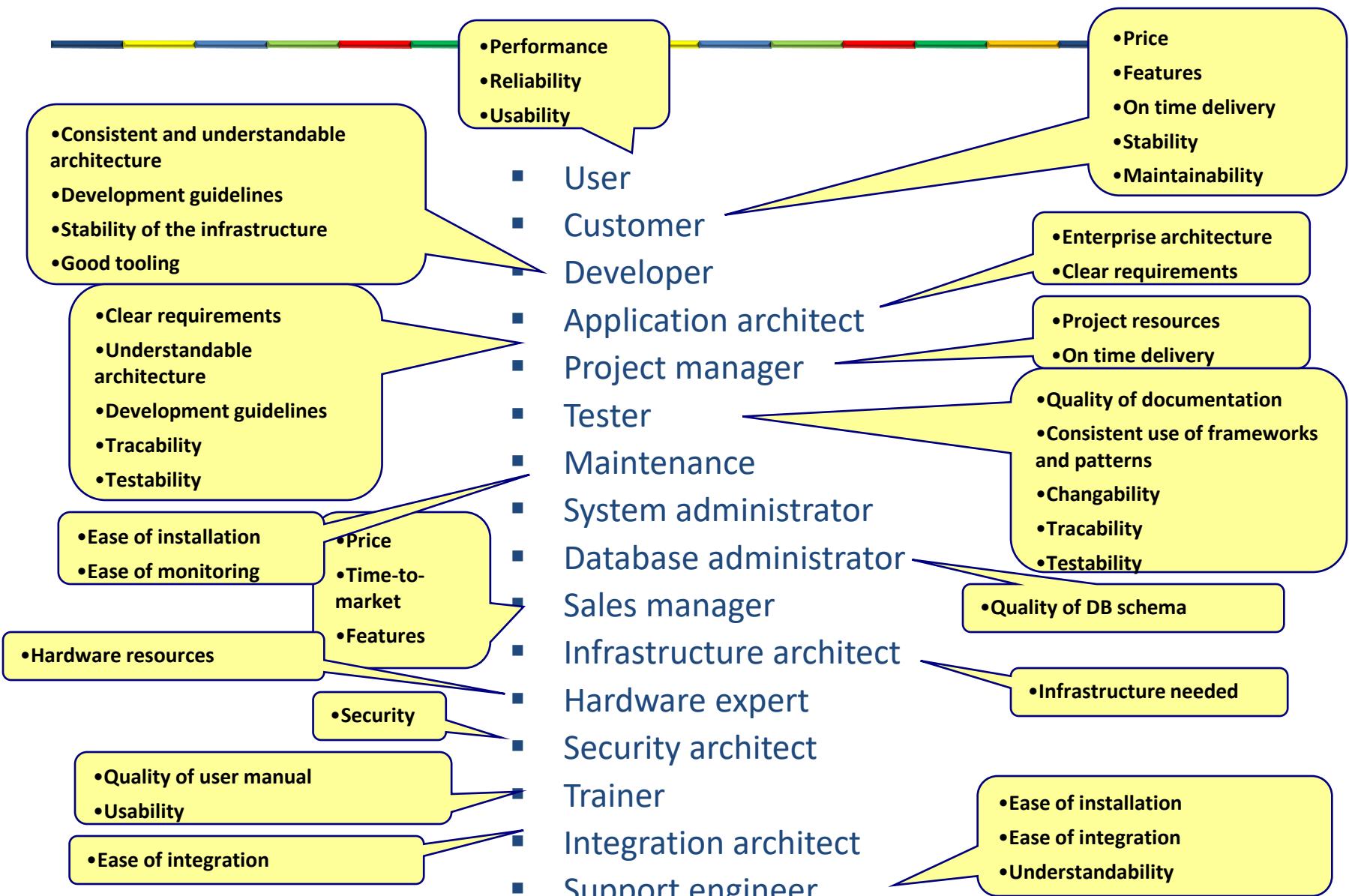
Architectural characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

Stakeholders

-
- User
 - Customer
 - Developer
 - Application architect
 - Project manager
 - Tester
 - Maintenance
 - System administrator
 - Database administrator
 - Sales
 - Infrastructure architect
 - Hardware expert
 - Security architect
 - Trainer
 - Integration architect
 - Support engineer

Stakeholders and their interest

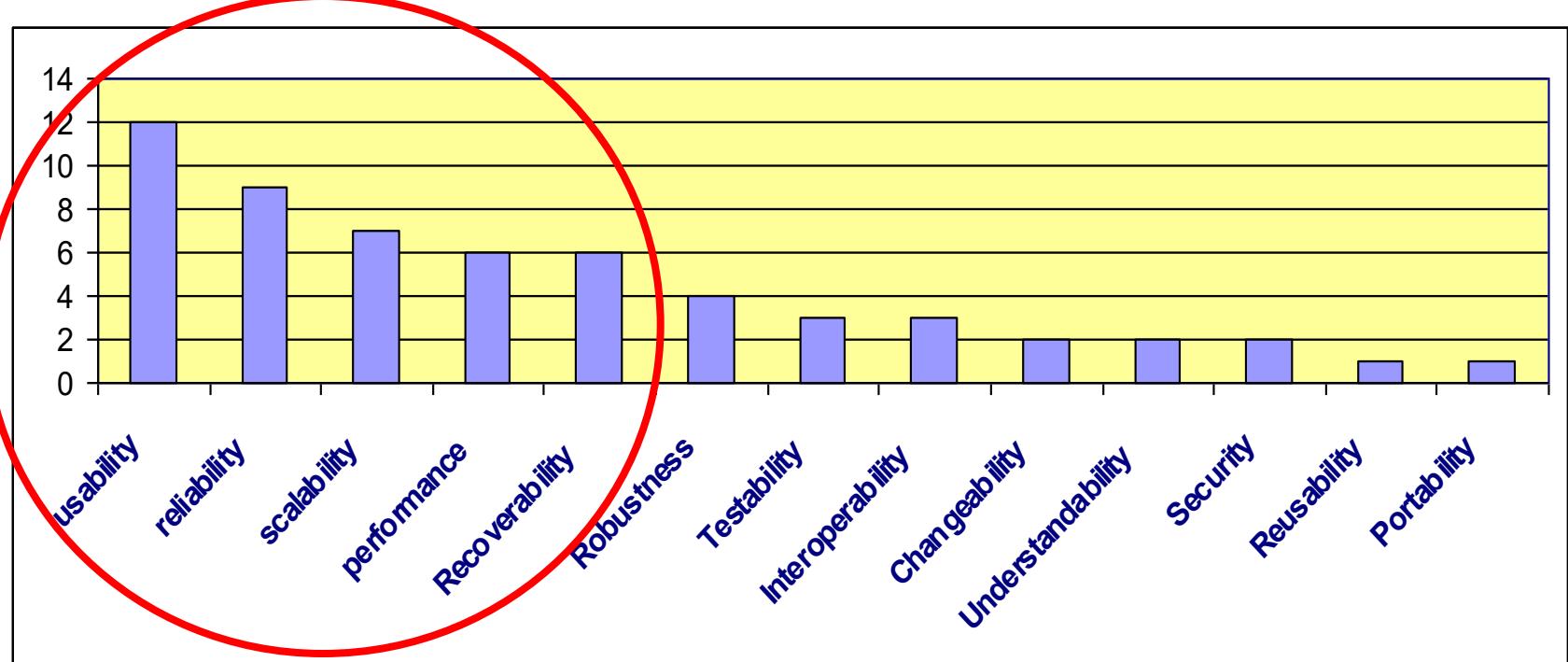


Quality workshop

- Goal:
 - Find the prioritized scenario's of the top qualities.
 - Communicate the qualities between stakeholders
- Workshop agenda
 - Explain the vision of the system
 - Explain the different qualities
 - Everyone votes (everyone gets \$10 to divide)
 - Discuss the result
 - Vote again
 - Create scenario's for the top qualities
 - Prioritize the scenario's (vote)



Quality workshop result



Some important architectural characteristics

- Performance
- Scalability
- Availability
- Recoverability
- Adaptability
- Maintainability
- Security
- Testability
- Fault-tolerance
- Deployability

Performance

- The responsiveness of the application to perform specific actions in a given time span.
- Scenario's
 - All actions must respond in 3 seconds
 - Complex actions must respond in 5 seconds

Availability

- The probability that the system is operating properly when it is requested for use
 - Calculated as uptime/total time
- Scenario's
 - The critical part of the system should be available 99.5% of the time
 - The non critical part of the system should be available 98.3% of the time

Availability

Availability %	Downtime per year ^[note 1]	Downtime per month	Downtime per week	Downtime per day
55.5555555% ("nine fives")	162.33 days	13.53 days	74.92 hours	10.67 hours
90% ("one nine")	36.53 days	73.05 hours	16.80 hours	2.40 hours
95% ("one and a half nines")	18.26 days	36.53 hours	8.40 hours	1.20 hours
97%	10.96 days	21.92 hours	5.04 hours	43.20 minutes
98%	7.31 days	14.61 hours	3.36 hours	28.80 minutes
99% ("two nines")	3.65 days	7.31 hours	1.68 hours	14.40 minutes
99.5% ("two and a half nines")	1.83 days	3.65 hours	50.40 minutes	7.20 minutes
99.8%	17.53 hours	87.66 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	8.77 hours	43.83 minutes	10.08 minutes	1.44 minutes
99.95% ("three and a half nines")	4.38 hours	21.92 minutes	5.04 minutes	43.20 seconds
99.99% ("four nines")	52.60 minutes	4.38 minutes	1.01 minutes	8.64 seconds
99.995% ("four and a half nines")	26.30 minutes	2.19 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	5.26 minutes	26.30 seconds	6.05 seconds	864.00 milliseconds
99.9999% ("six nines")	31.56 seconds	2.63 seconds	604.80 milliseconds	86.40 milliseconds
99.99999% ("seven nines")	3.16 seconds	262.98 milliseconds	60.48 milliseconds	8.64 milliseconds
99.999999% ("eight nines")	315.58 milliseconds	26.30 milliseconds	6.05 milliseconds	864.00 microseconds
99.9999999% ("nine nines")	31.56 microseconds	2.63 microseconds	604.80 microseconds	86.40 microseconds

Recoverability

- How easy can the system recover from failure
- Scenario's
 - Mean Time To Recovery = 10 hours

Scalability

- The ability to handle an increase in the workload without impacting the performance
- Scenario's
 - The system should scale up to 100.000 users in a year
 - The system should be able to handle 500 concurrent users

Adaptability

- Easy of which a system can be changed
- Scenario's
 - It should be easy to change the database
 - It should be easy to support different clients (web, mobile apps, etc)

Maintainability

- The ability of any application to go through modifications and updates with a degree of ease.
- Scenario's
 - The system should be highly configurable
 - The system should have 80% test coverage
 - All errors should be logged

Security

- System's ability to resist unauthorized usage while still providing its services to legitimate users
- 3 aspects
 - Authentication: are you who you say you are?
 - Authorization: what are you allowed to do?
 - Secrecy: encrypt the data
- Scenario's
 - Only authorized users may access the system
 - We always use 2 factor authentication
 - All secret data that is sent to other systems should be encrypted

Testability

- The ease with which software can be made to demonstrate its faults through testing.
- Scenario's
 - We use automated tests
 - We need 80% test coverage

Fault-tolerance

- Ability to continue to operate properly in the event of failure
- Scenario's
 - The system should be fault tolerant for database failure
 - The system should be fault tolerant for network failure

Deployability

- Ease of installing the software
- Scenario's
 - The system should run on-premise and in the cloud with minimal changes in the configuration
 - There is an automatic installation script

Other important architectural characteristics

- Performance
- Scalability
- Availability
- Recoverability
- Elasticity
- Adaptability
- Maintainability
- Security
- Testability
- Fault-tolerance
- Deployability
- Cost
- Time to market
- Simplicity
- Number of agile teams
- Try out new technology

Auction system

- An auction company wants to take their auctions online to a nationwide scale. Customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.
- **Users:** scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements:**
 - bidders can see a live video stream of the auction and see all bids as they occur
 - auctions must be as real-time as possible
 - both online and live bids must be received in the order in which they are placed
 - bidders register with credit card; system automatically charges card if bidder wins
- • **Additional Context:**
 - auction company is expanding aggressively by merging with smaller competitors
 - if nationwide auction is a success, replicate the model overseas
 - budget is not constrained--this is a strategic direction
 - company just exited a lawsuit where they settled a suit alleging fraud

Auction system

- An auction company wants to take their auctions online to a nationwide scale. Customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.
- **Users:** scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements:**
 - bidders can see a live video stream of the auction and see all bids as they occur
 - **auctions must be as real-time as possible**
 - both online and live bids must be received in the order in which they are placed
 - bidders register with credit card; system automatically charges card if bidder wins
- • **Additional Context:**
 - auction company is expanding aggressively by merging with smaller competitors
 - if nationwide auction is a success, replicate the model overseas
 - budget is not constrained--this is a strategic direction
 - company just exited a lawsuit where they settled a suit alleging fraud

availability performance

Auction system

- An auction company wants to take their auctions online to a **nationwide scale**. Customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.
- **Users:** **scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible**
- **Requirements:**
 - bidders can see a live video stream of the auction and see all bids as they occur
 - auctions must be as real-time as possible
 - both online and live bids must be received in the order in which they are placed
 - bidders register with credit card; system automatically charges card if bidder wins
- • **Additional Context:**
 - auction company is expanding aggressively by merging with smaller competitors
 - **if nationwide auction is a success, replicate the model overseas**
 - budget is not constrained--this is a strategic direction
 - company just exited a lawsuit where they settled a suit alleging fraud

availability performance scalability

Auction system

- An auction company wants to take their auctions online to a **nationwide scale**. Customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.
- **Users:** **scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible**
- **Requirements:**
 - bidders can see a live video stream of the auction and see all bids as they occur
 - auctions must be as real-time as possible
 - both online and live bids must be received in the order in which they are placed
 - bidders register with credit card; system automatically charges card if bidder wins
- • **Additional Context:**
 - auction company is expanding aggressively by merging with smaller competitors
 - if nationwide auction is a success, replicate the model overseas
 - budget is not constrained--this is a strategic direction
 - company just exited a lawsuit where they settled a suit alleging fraud

availability performance scalability elasticity

Auction system

- An auction company wants to take their auctions online to a nationwide scale. Customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.
- **Users:** scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements:**
 - bidders can see a live video stream of the auction and see all bids as they occur
 - auctions must be as real-time as possible
 - both online and live bids must be received in the order in which they are placed
 - bidders register with credit card; system automatically charges card if bidder wins
- • **Additional Context:**
 - auction company is expanding aggressively by merging with smaller competitors
 - if nationwide auction is a success, replicate the model overseas
 - budget is not constrained--this is a strategic direction
 - company just exited a lawsuit where they settled a suit alleging fraud

Availability performance scalability elasticity security

Do not choose more than 7 architectural characteristics

Auction system

System must be up
during an auction
session

availability

Bids are received
within 3 seconds

performance

Scale to thousands
of participants

scalability

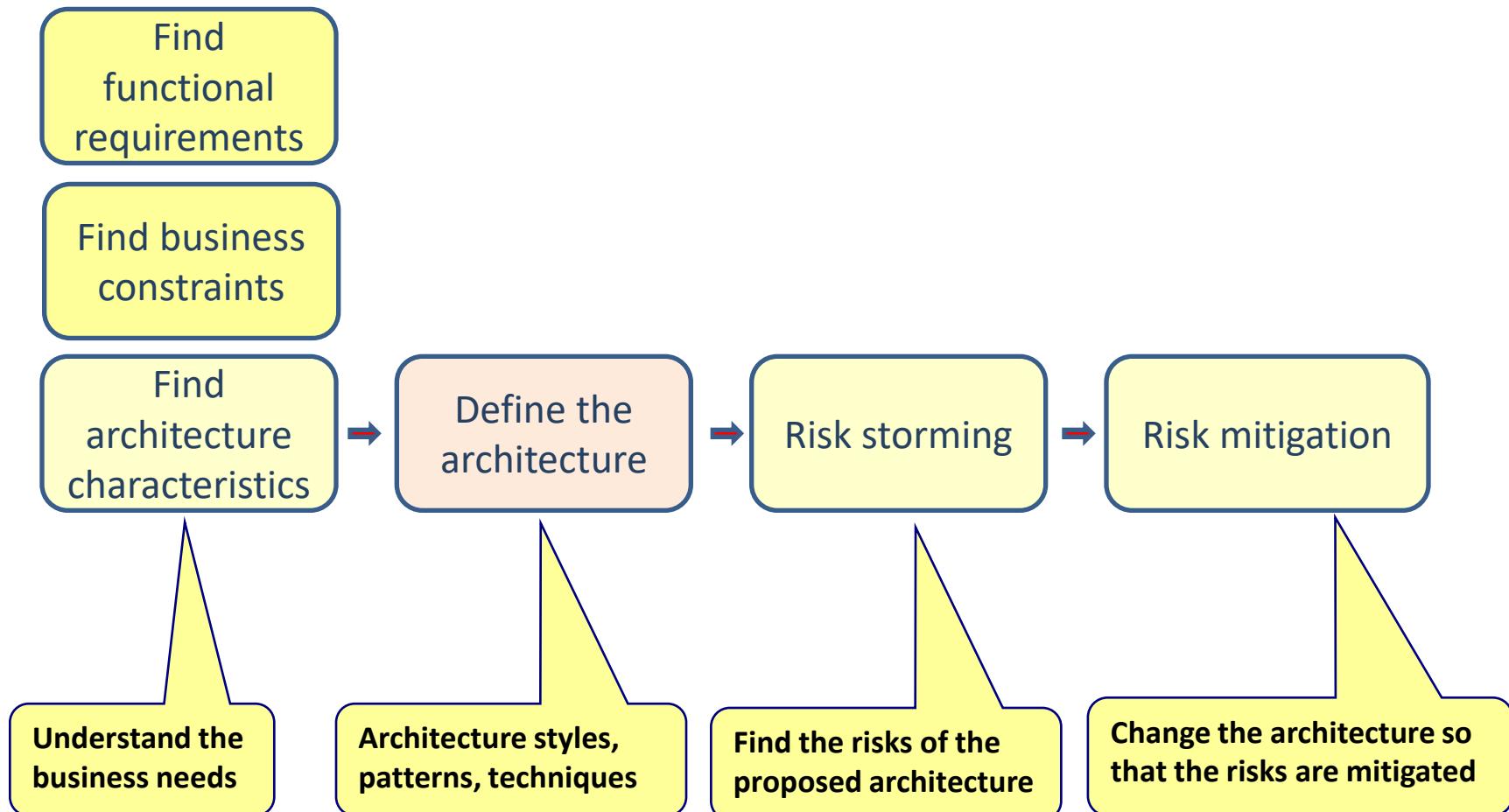
Be elastic between
popular & less
popular auctions

elasticity

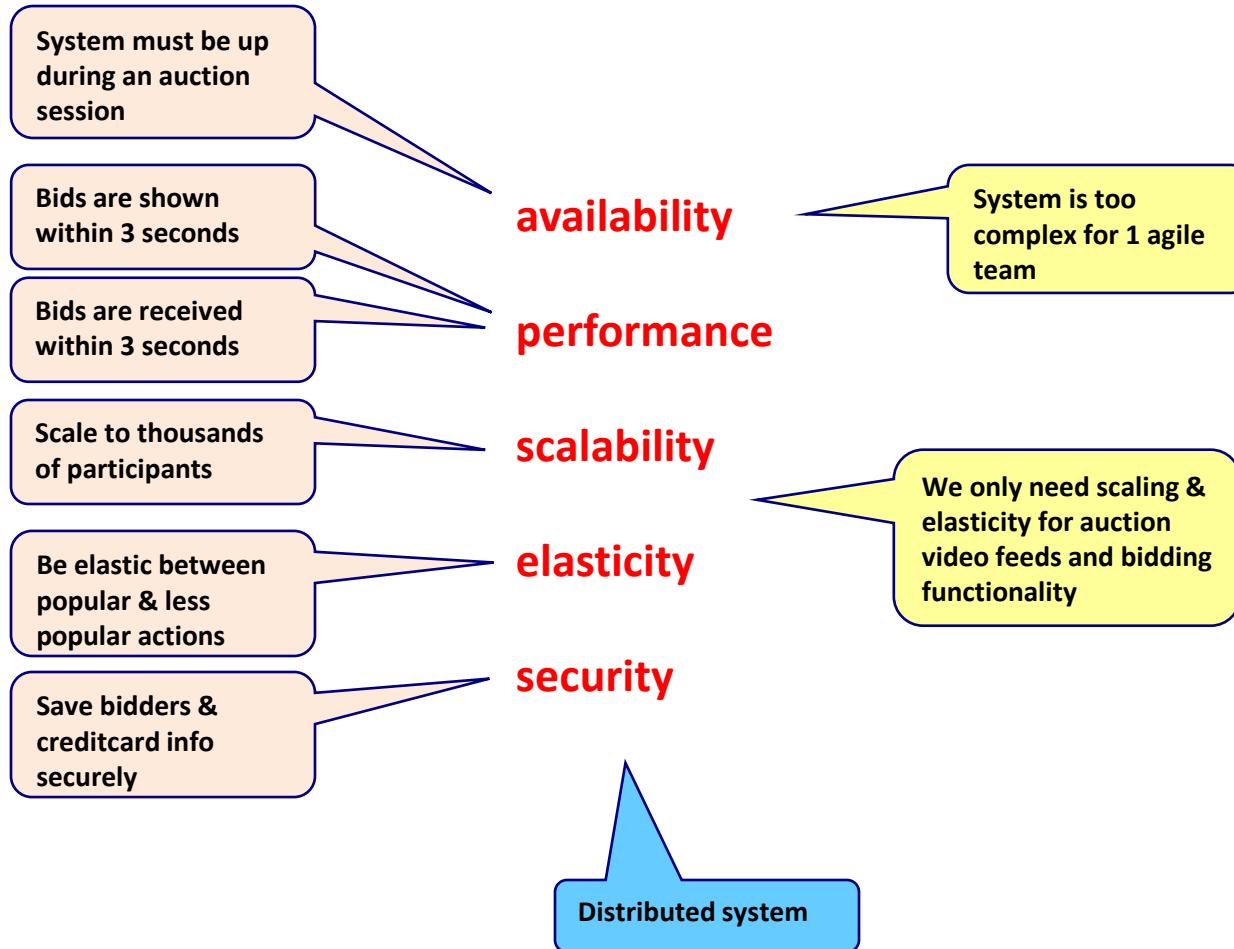
Save bidders &
creditcard info
securely

security

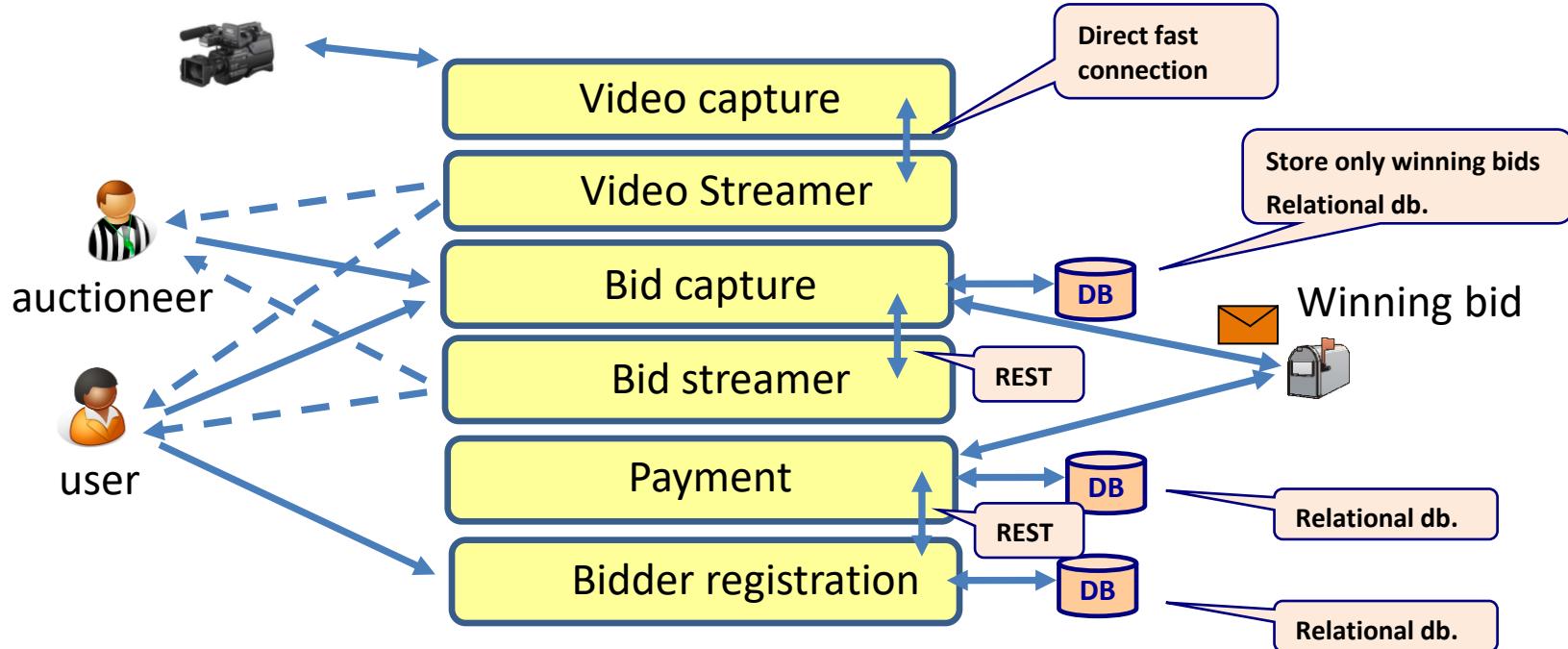
Finding the right architecture



Auction system

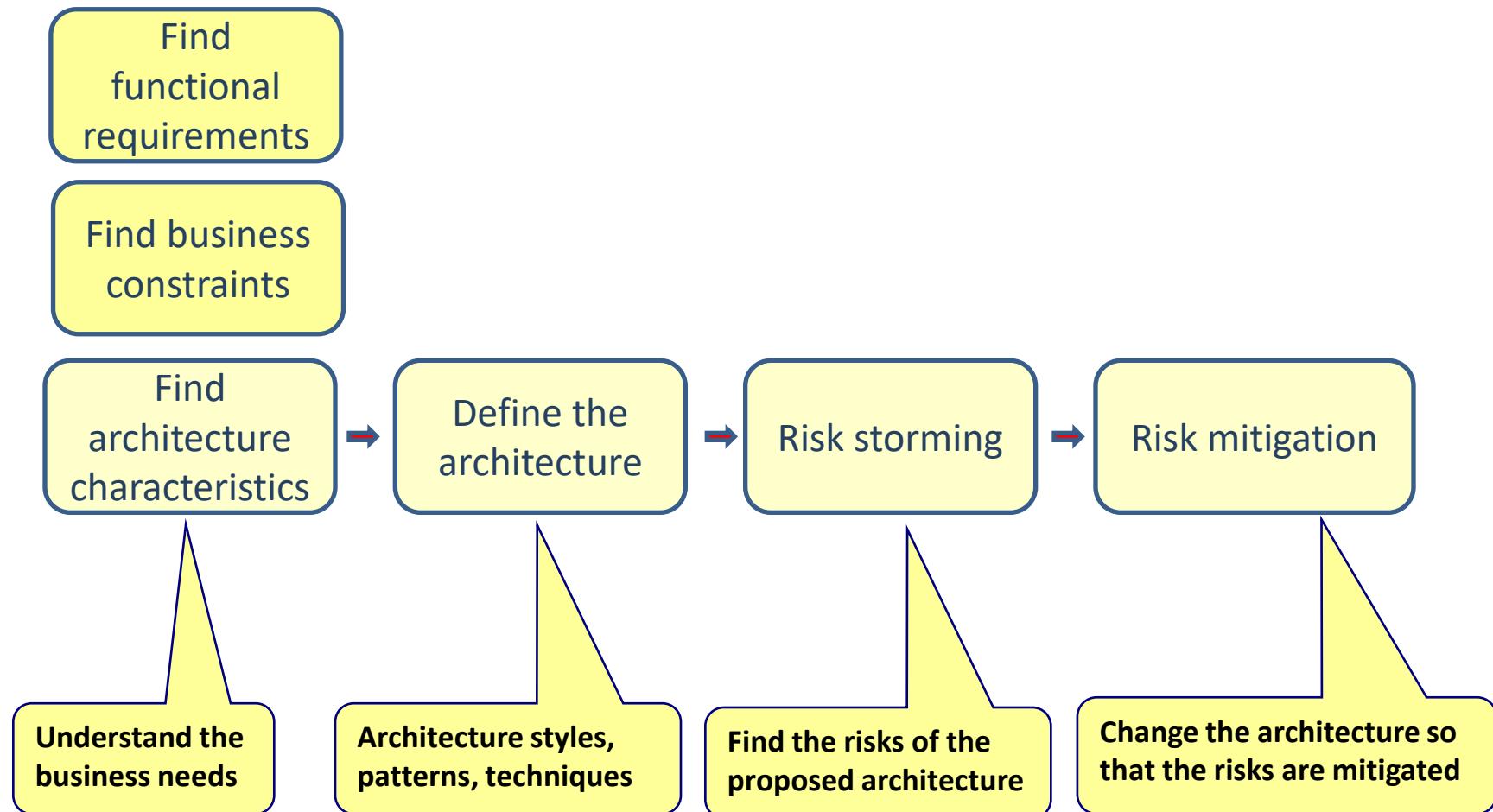


Auction system



Availability reliability performance scalability elasticity security

Software architecture

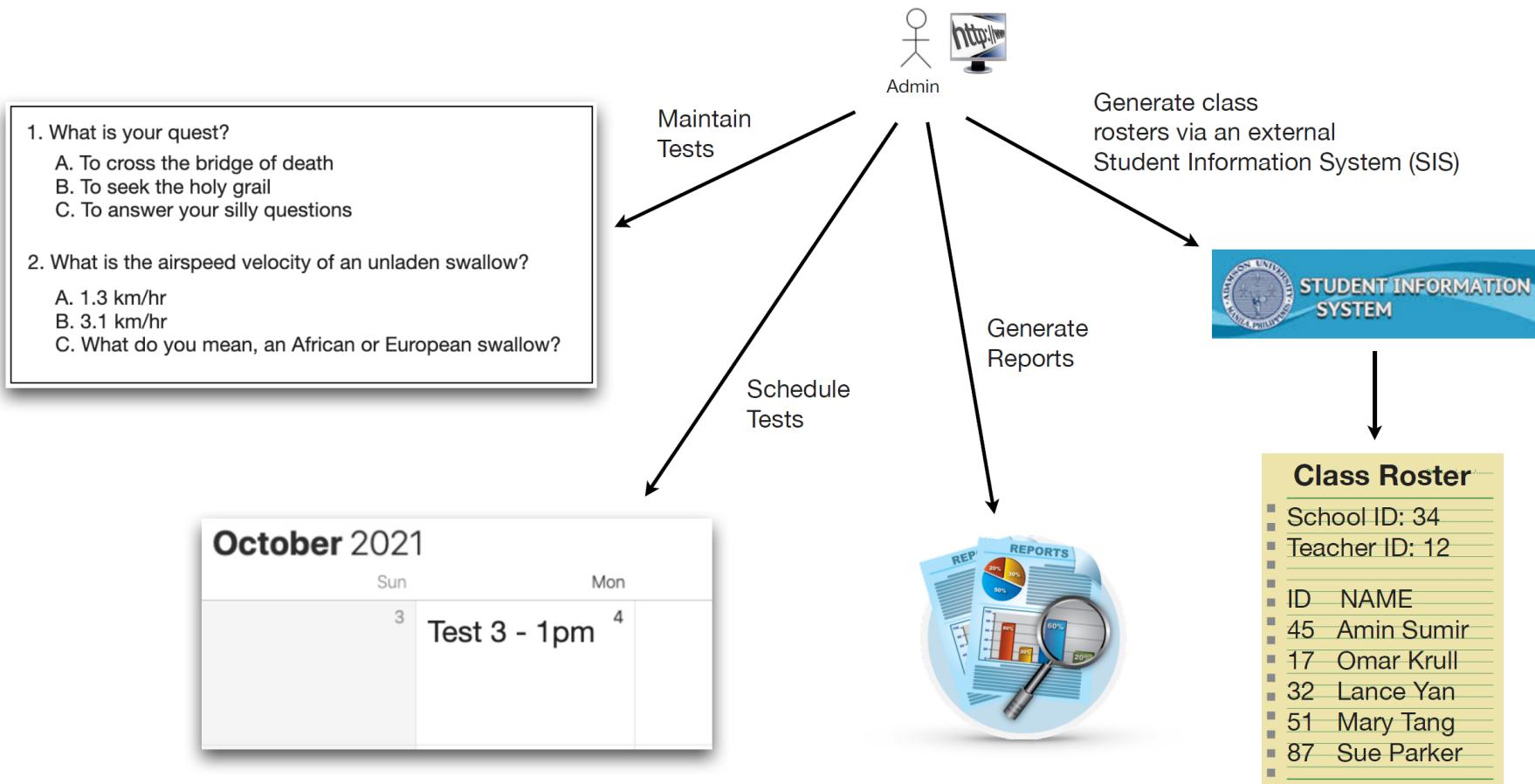


Standardized testing system

Your state (or country) would like a new system to support standardized testing for all public and private schools, grades 3-12.

- Requirements:
 - Students will use a new web-based (and backend) testing application to take the test from their school based on their assigned teacher. Questions are presented to the student, they submit the answer (true/false or multiple choice), and get the next question. The student answers the question and the corresponding grade for each question (right or wrong) are eventually consolidated into a single test answer database representing all of the test scores for all students.
 - Administrative staff will use the system to create the test and corresponding answer keys, schedule the tests, and generate the sign-in roster from an existing external Student Information System (SIS). After the testing is complete, the admin staff will use the new system to generate student, teacher, school, and question validation reports.
 - Users: 800,000+ students (up to 120,000 at a time), 2 administrators (test creation, test scheduling, etc.)
- Additional info from the education department:
 - “It is ***absolutely imperative*** that no student answers are lost in the event of a system crash.”
 - “We need the new system before the start of the next term (6 months).”
 - “It is vital that the test answers be protected from prying eyes.” “We need to make sure a student doesn’t take the test multiple times.”
 - “Somehow we need to make sure a student doesn’t take a test for someone else.”

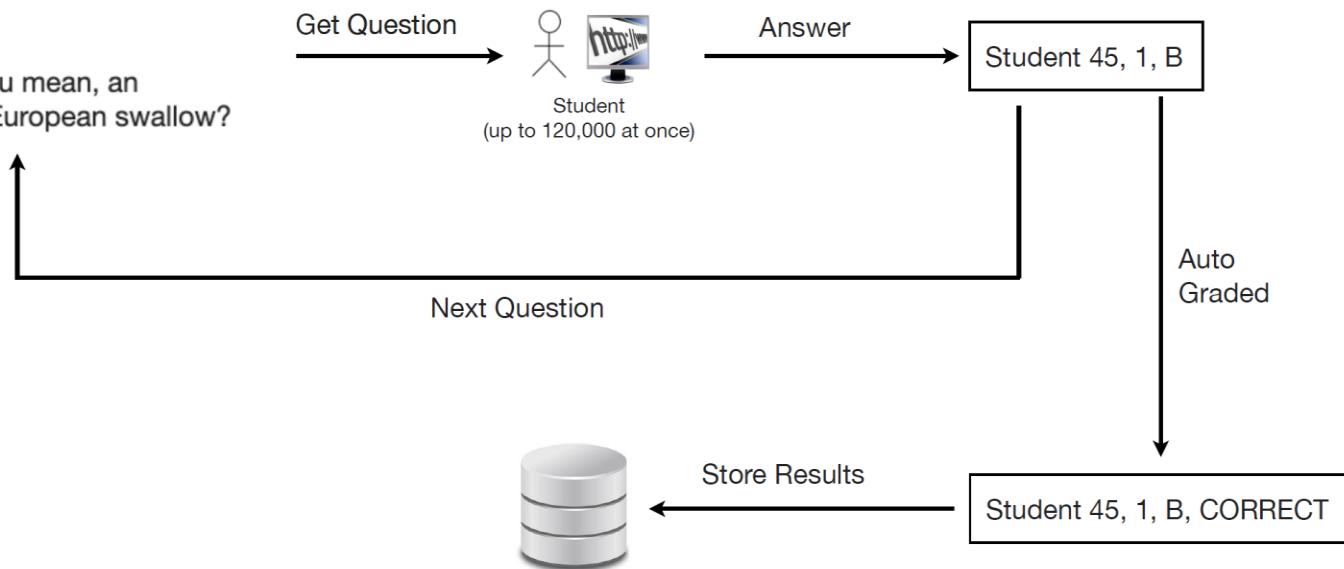
Admin user



Student user

2. What is the airspeed velocity of an unladen swallow?

- A. 1.3 km/hr
- B. 3.1 km/hr
- C. What do you mean, an African or European swallow?



Architectural characteristics

Your state (or country) would like a new system to support standardized testing for all public and private schools, grades 3-12.

- Requirements:

- Students will use a new web-based (and backend) testing application to take the test from their school based on their assigned teacher. Questions are presented to the student, they submit the answer (true/false or multiple choice), and get the next question. The student answers the question and the corresponding grade for each question (right or wrong) are eventually consolidated into a single test answer database representing all of the test scores for all students.
- Administrative staff will use the system to create the test and corresponding answer keys, schedule the tests, and generate the sign-in roster from an existing external Student Information System (SIS). After the testing is complete, the admin staff will use the new system to generate student, teacher, school, and question validation reports.
- Users: 800,000+ students (up to 120,000 at a time)
- 2 administrators (test creation, test scheduling, etc.)

Performance Availability Reliability

- Additional info from the education department:

- “It is **absolutely imperative** that no student answers are lost in the event of a system crash.”
- “We need the new system before the start of the next term (6 months).”
- “It is vital that the test answers be protected from prying eyes.” “We need to make sure a student doesn’t take the test multiple times.”
- “Somehow we need to make sure a student doesn’t take a test for someone else.”

Data integrity

Security

Feasibility(cost/time)

Architectural characteristics

Security

Students should not be able to “hack into the system” and change grades or see test answers

Data integrity

Only students who are allowed to take the test, can take the test. Nobody can take a test for someone else

Performance

A student should not have to wait for more than 4 seconds until the next question is shown after answering a question

Elasticity

The system should be able to handle 120.000 students at the same time

Availability

The system should be available during test times (9:00 AM – 5:00 PM)

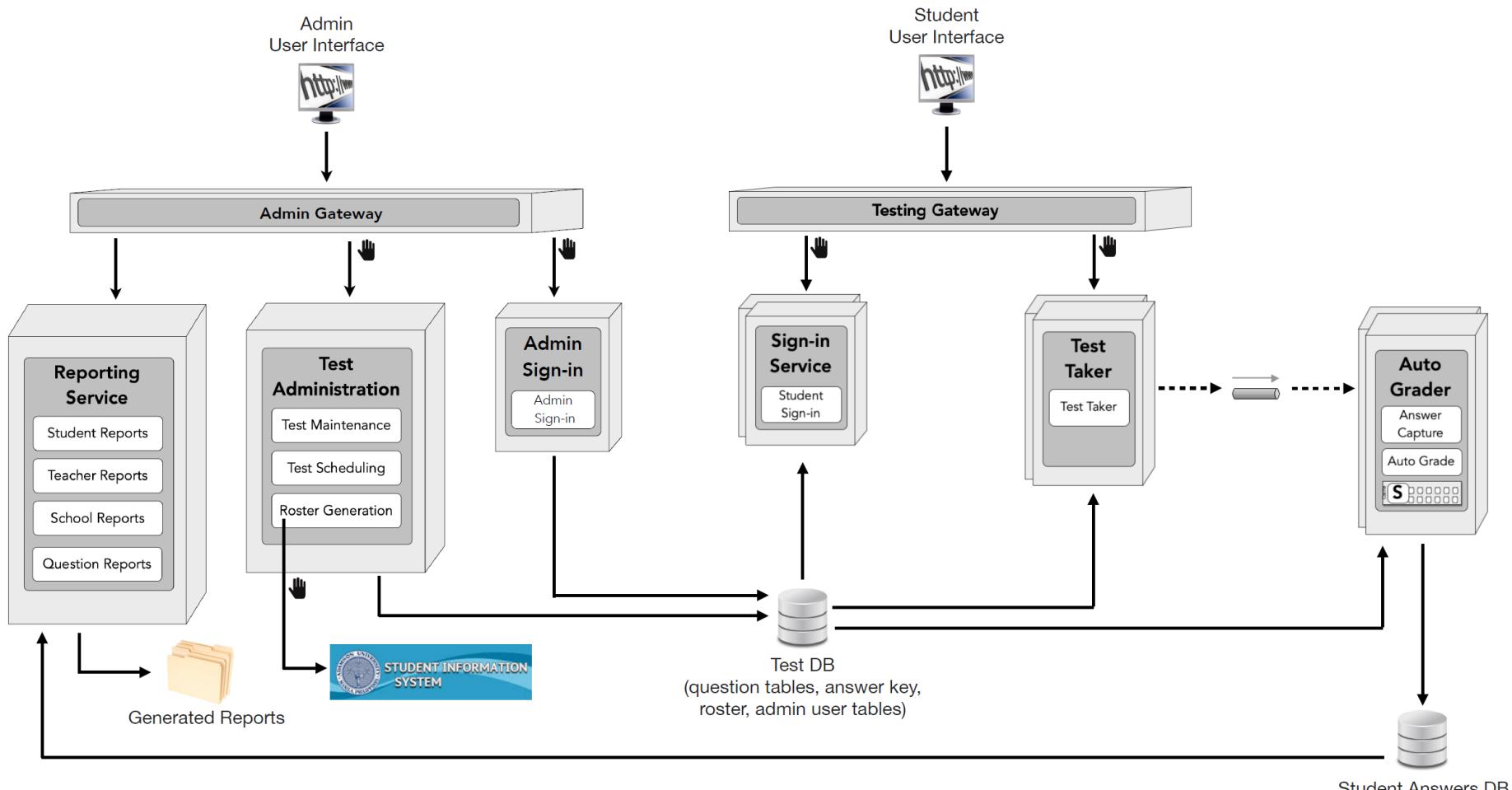
Reliability

All tests and corresponding grading are always correct

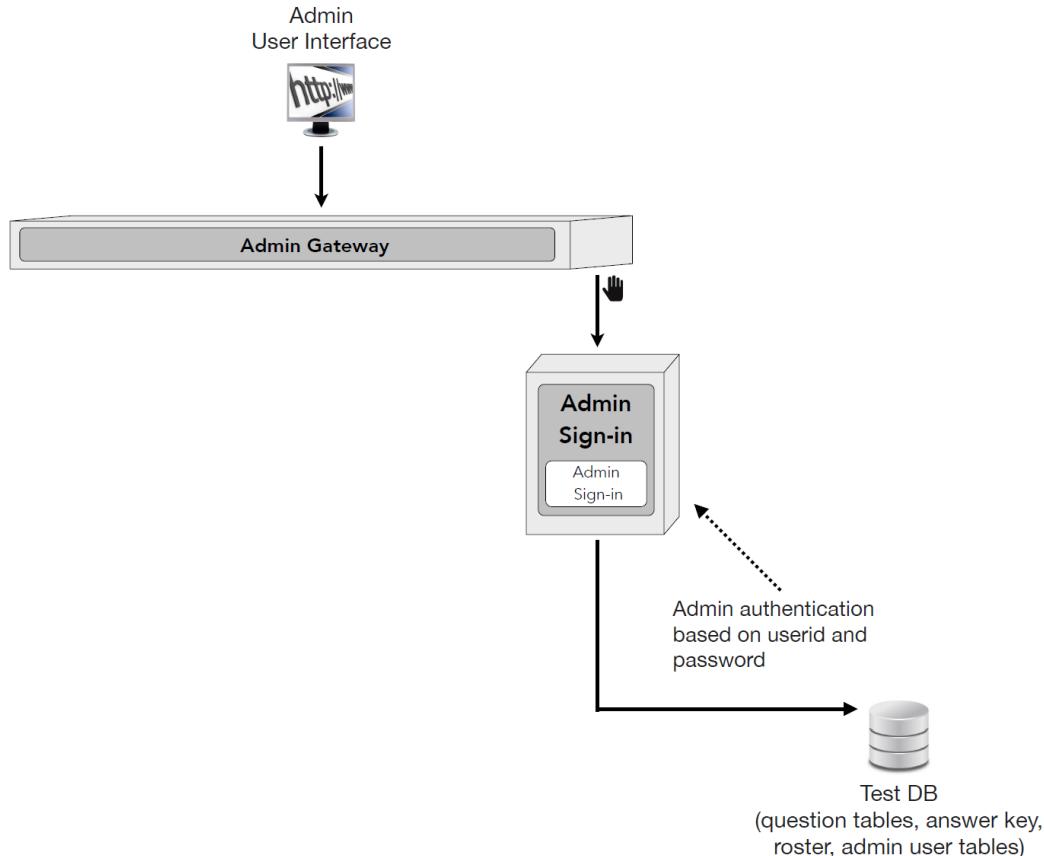
Feasibility(cost/time)

The system needs to be ready in 6 months

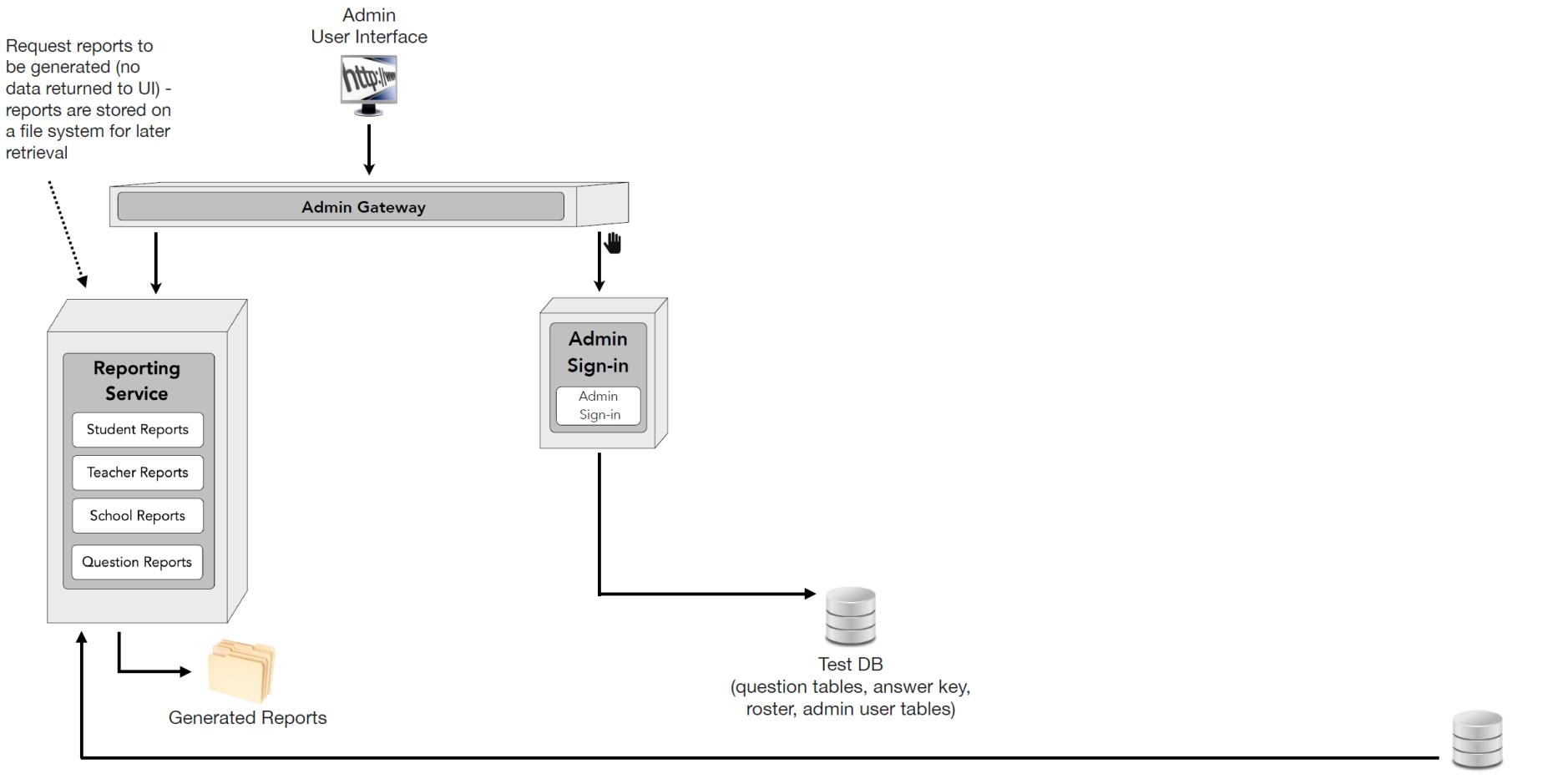
Proposed architecture



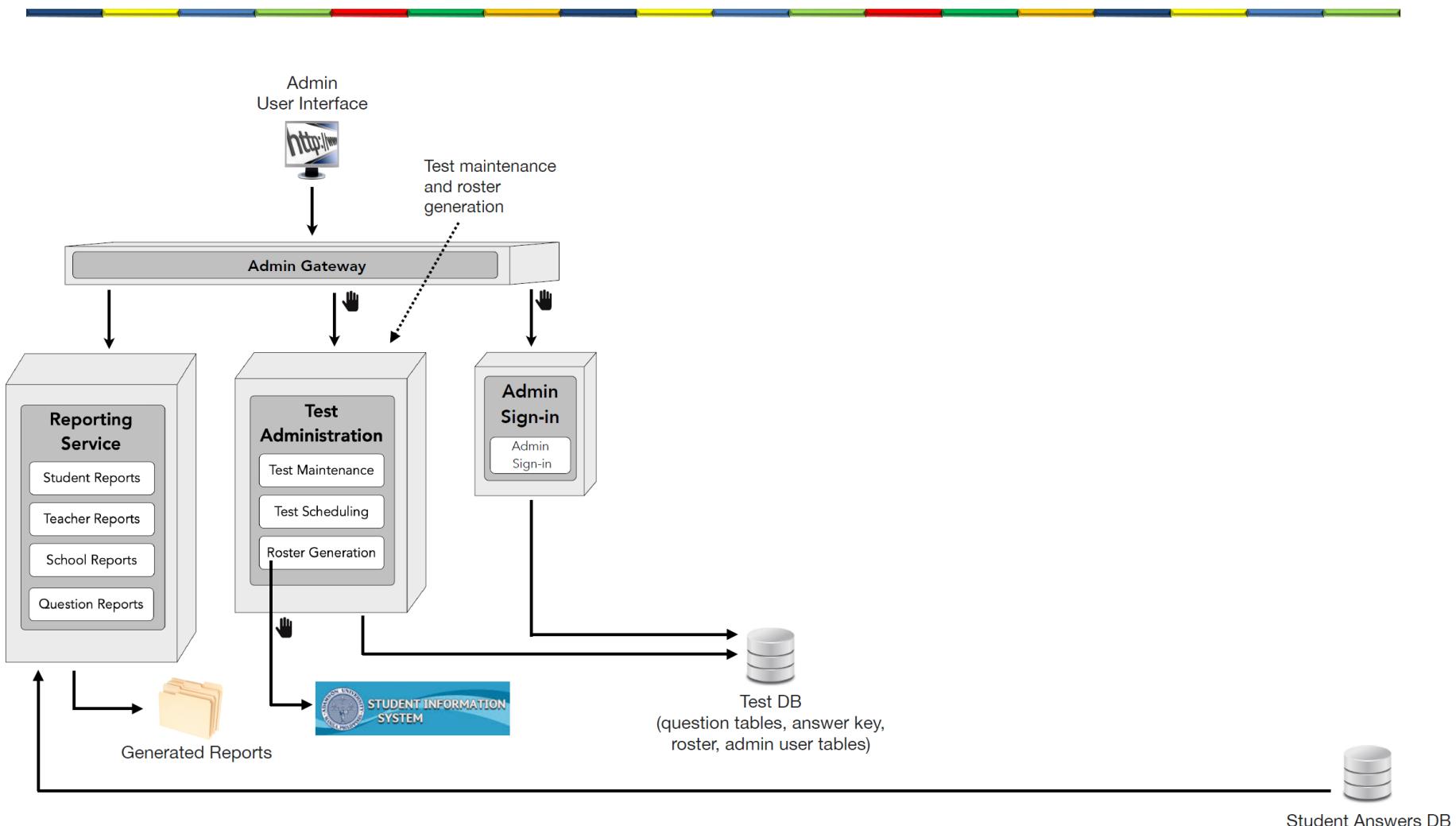
Admin sign-in



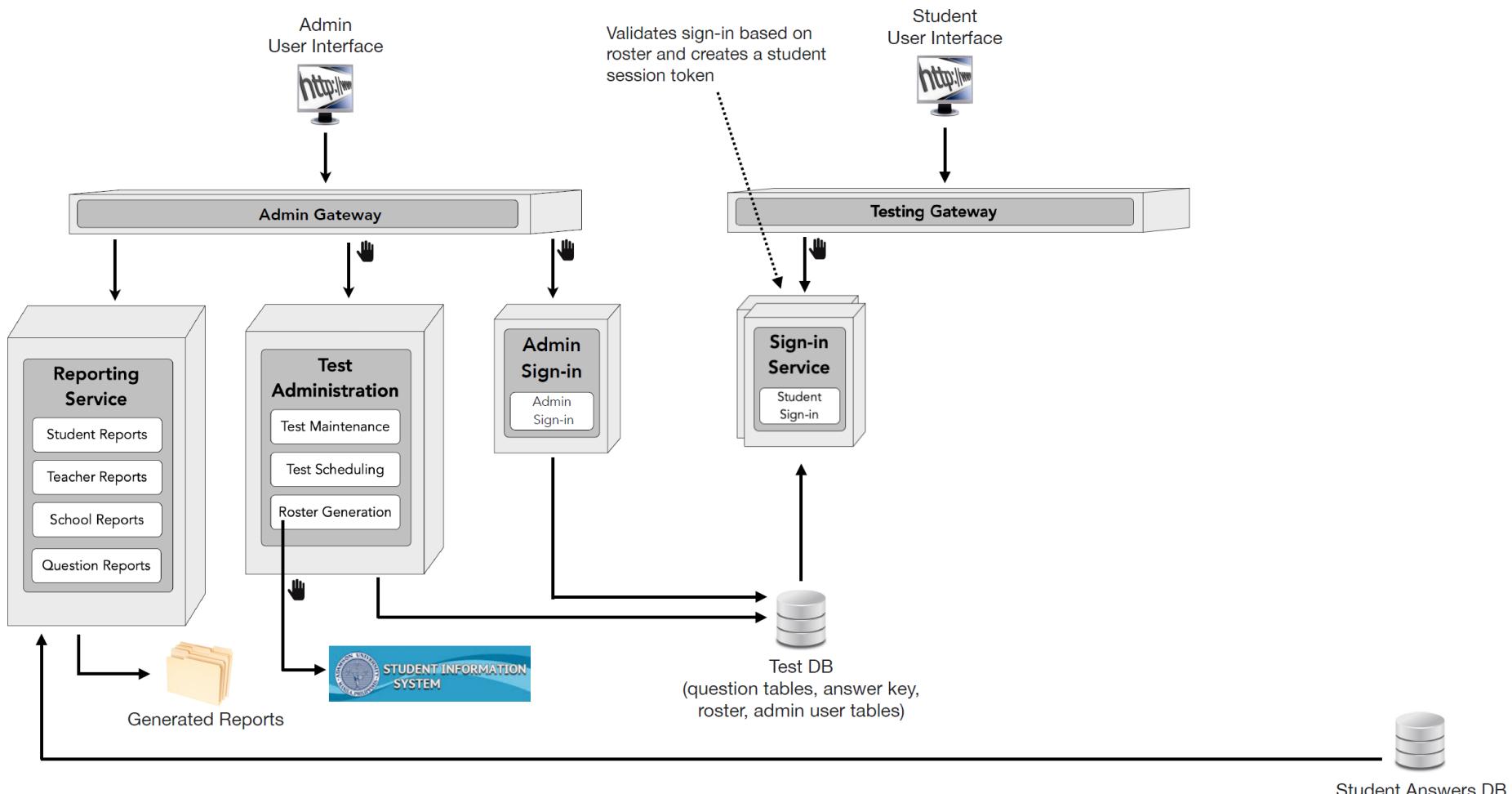
Reporting



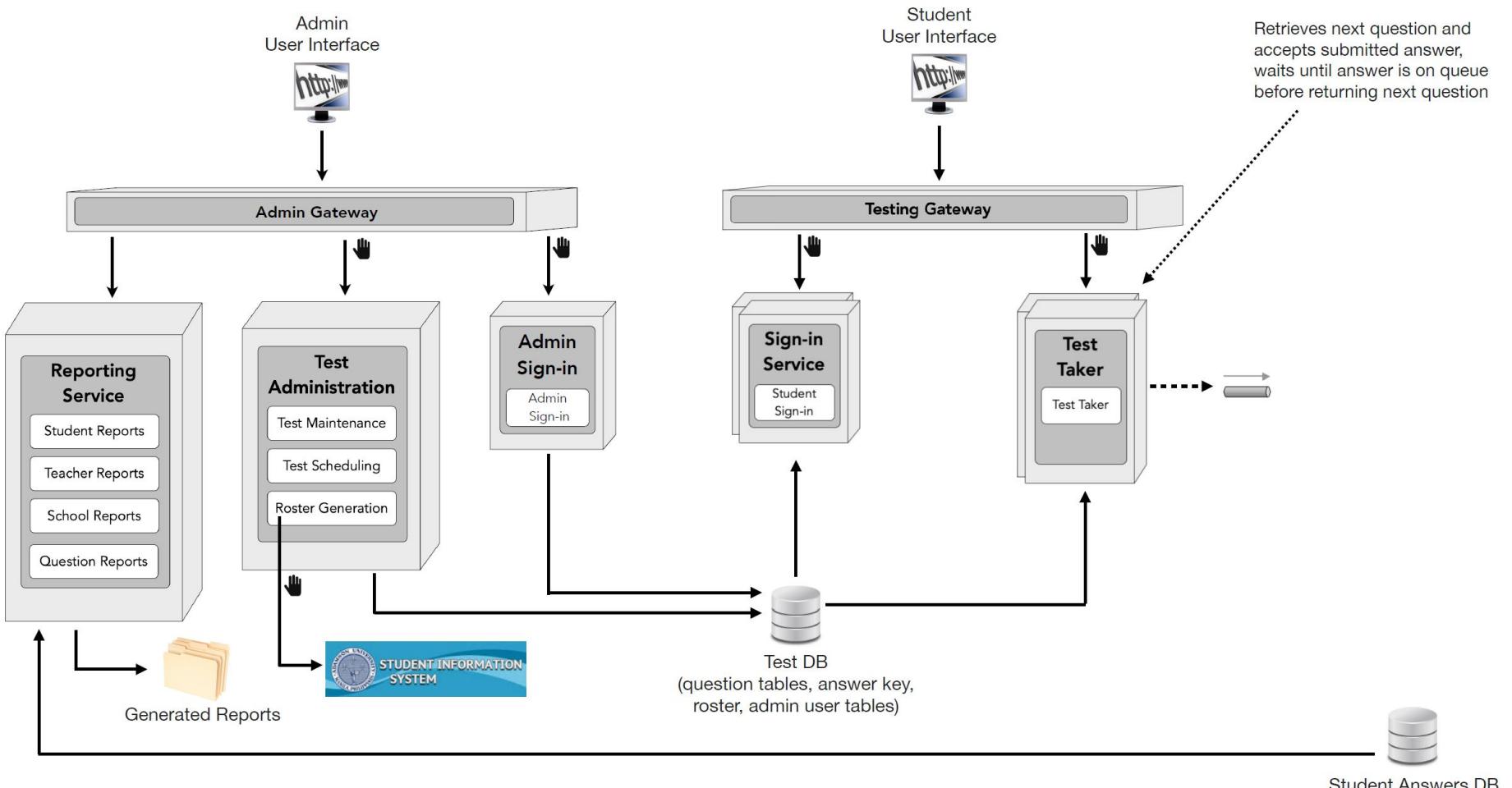
Test administration



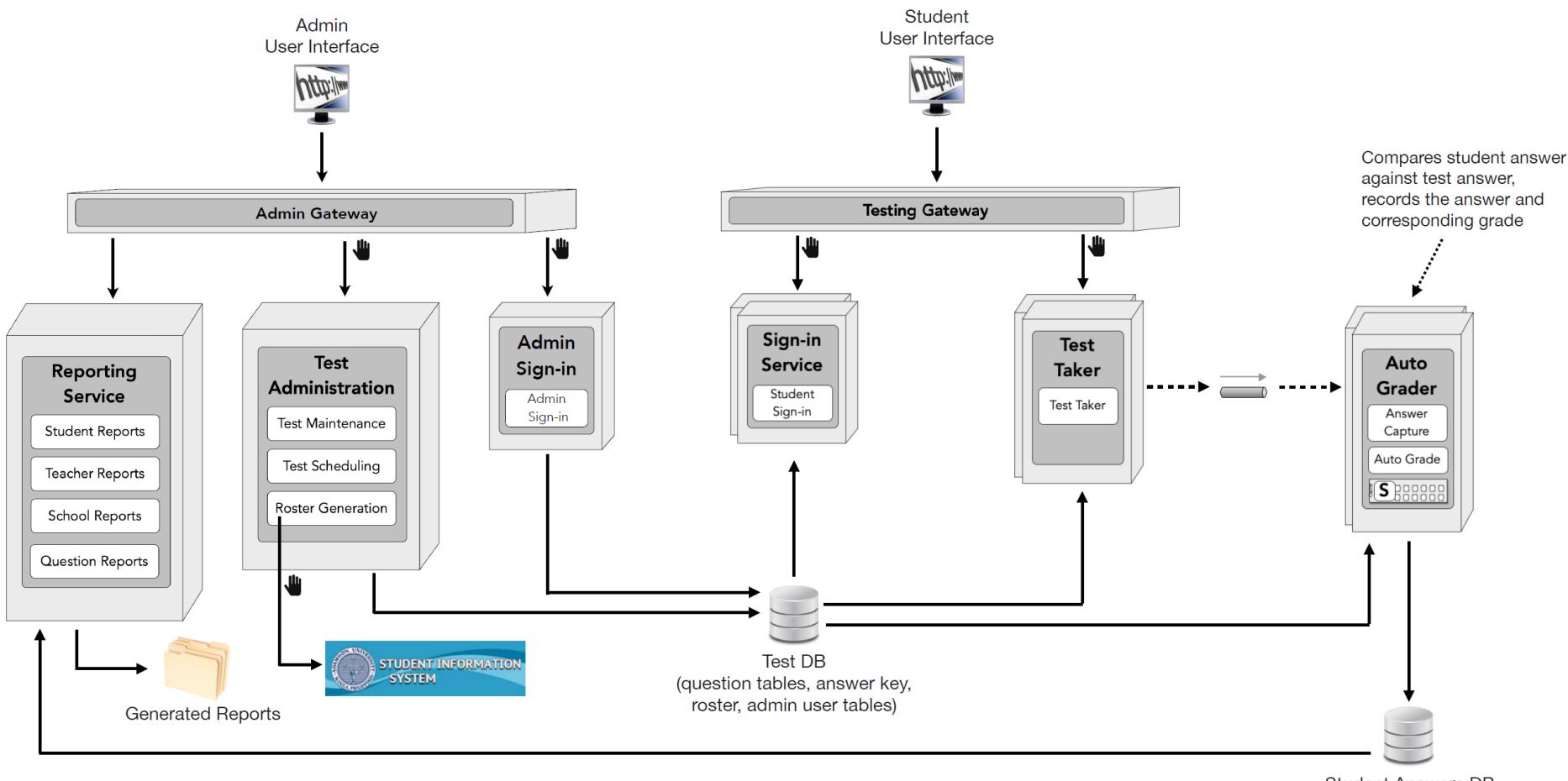
Student sign-in



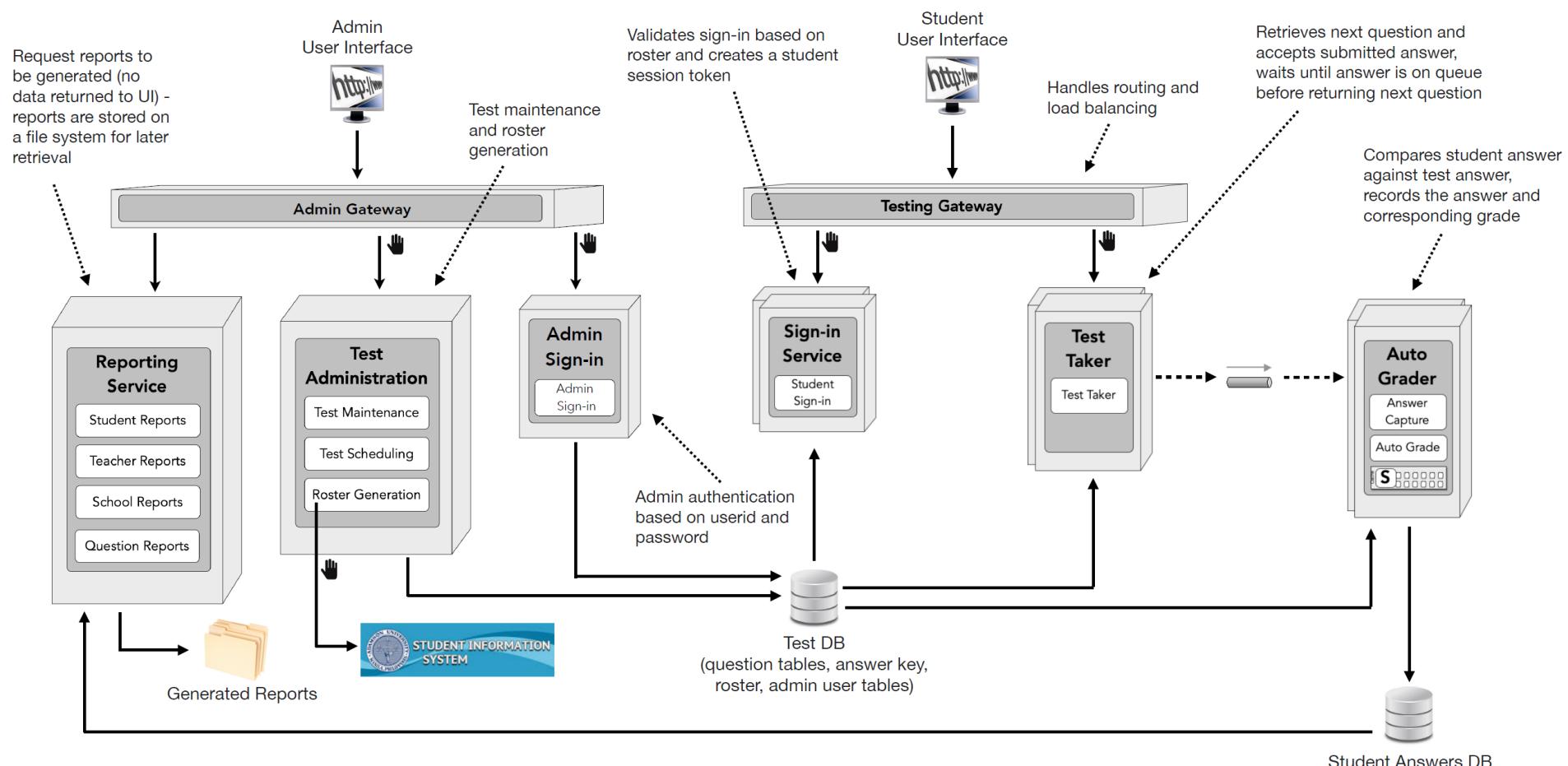
Students take the test



Automated grading

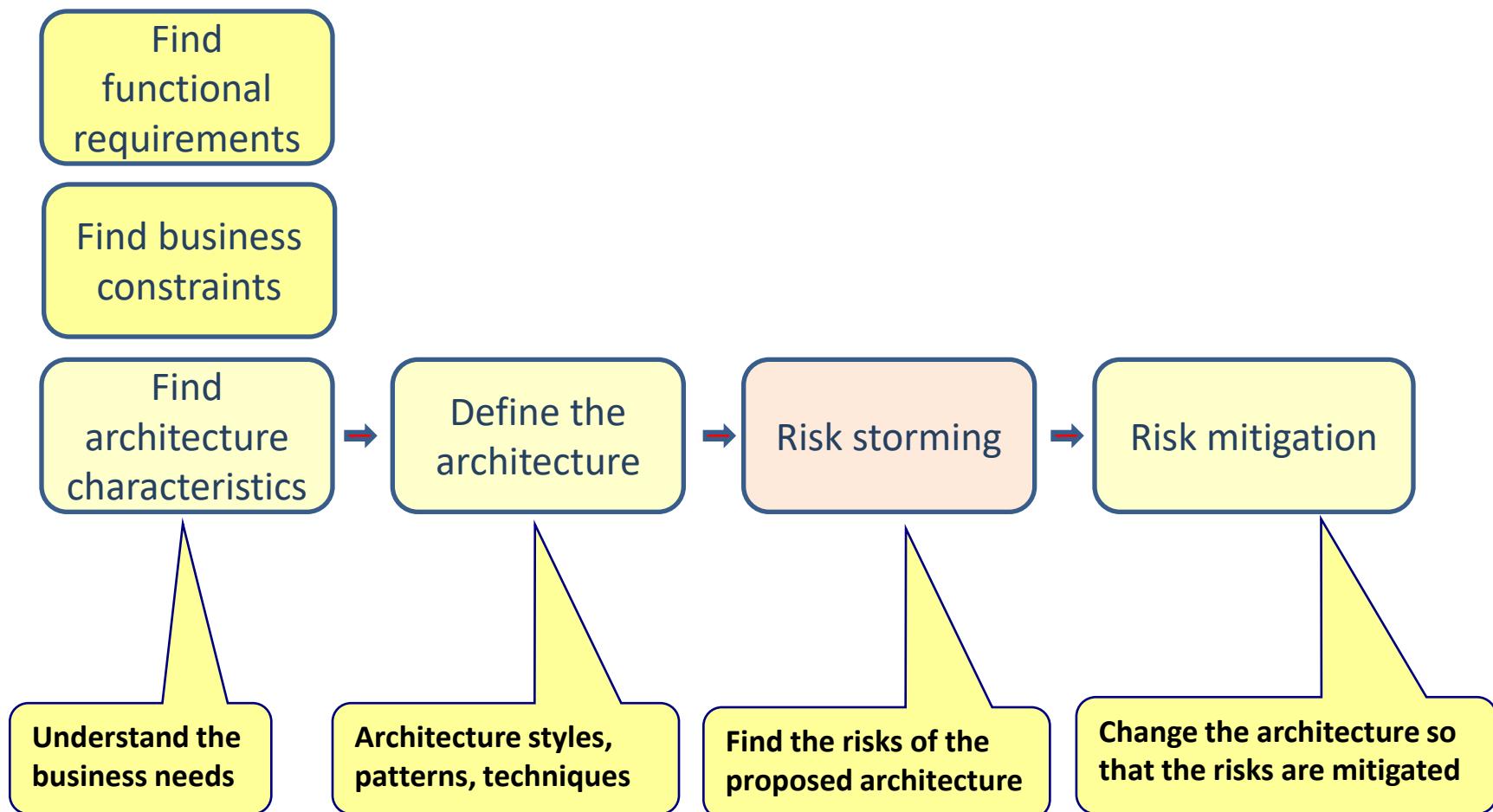


Proposed architecture



RISK STORMING

Software architecture



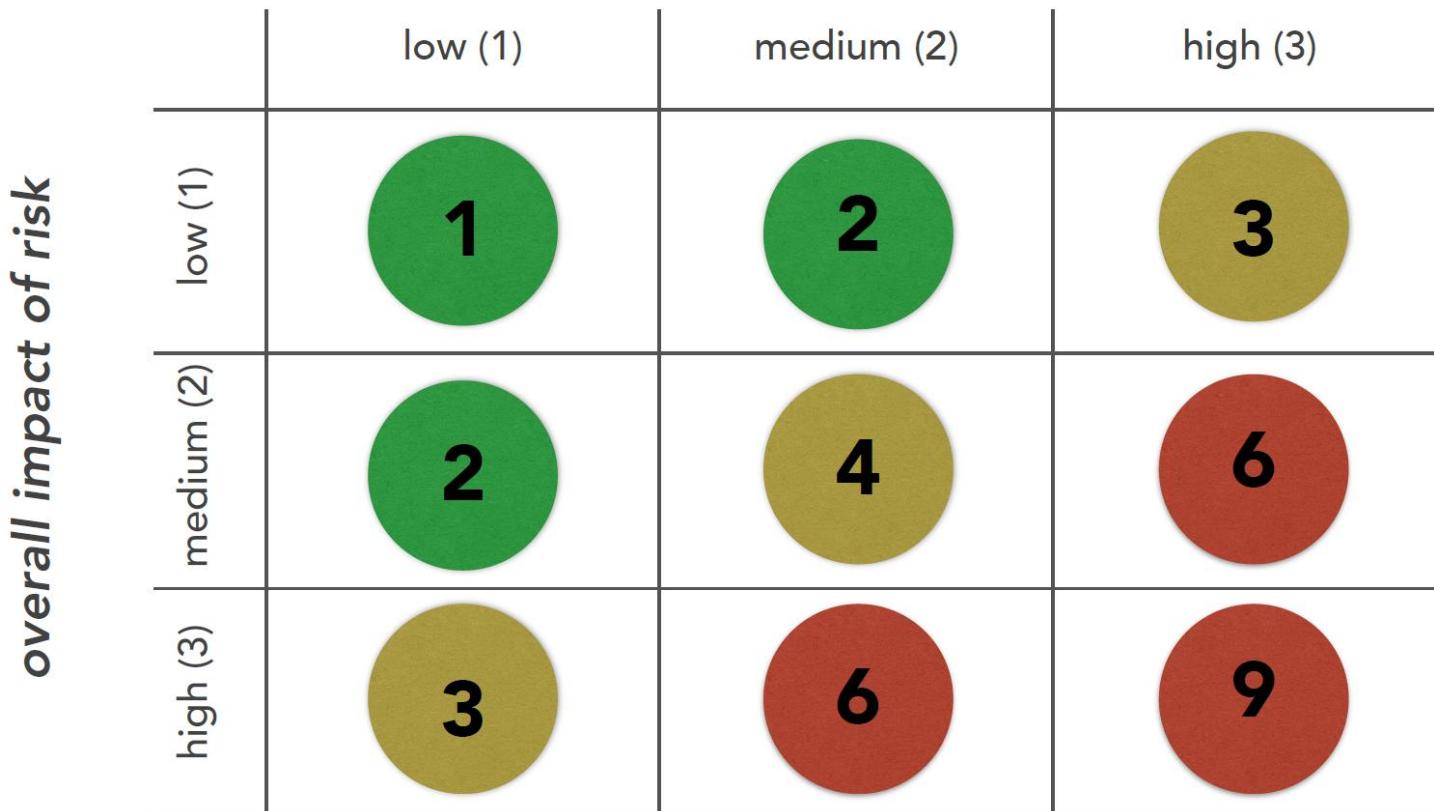
Risk storming

- Quick and fun technique that provides a collaborative way to identify and visualize risks.



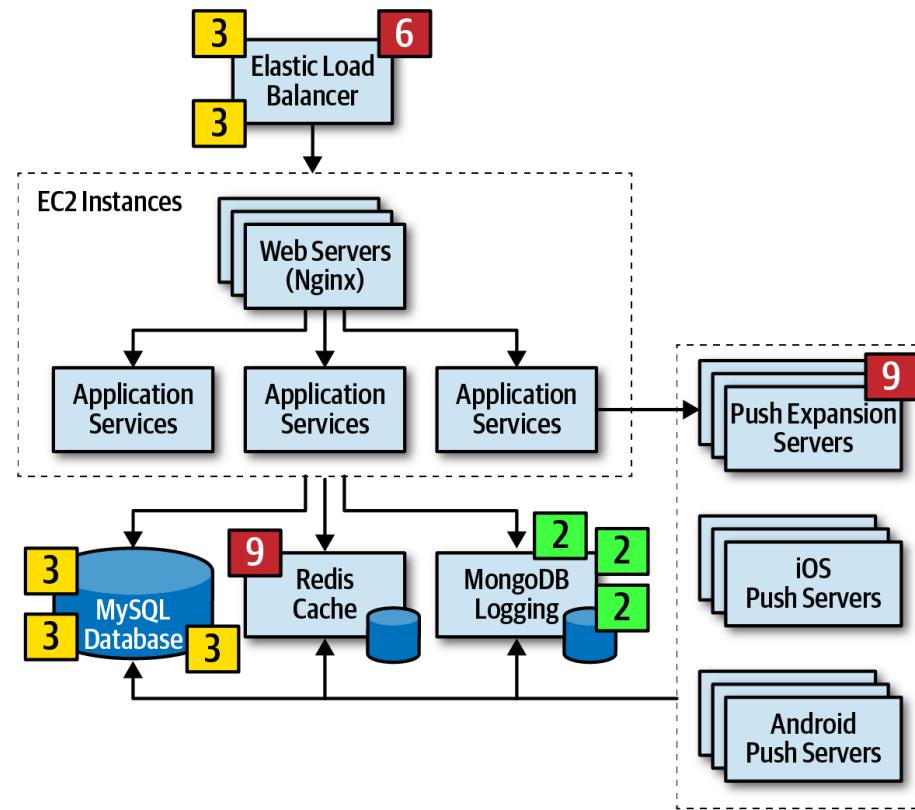
Quantify risk

likelihood of risk occurring



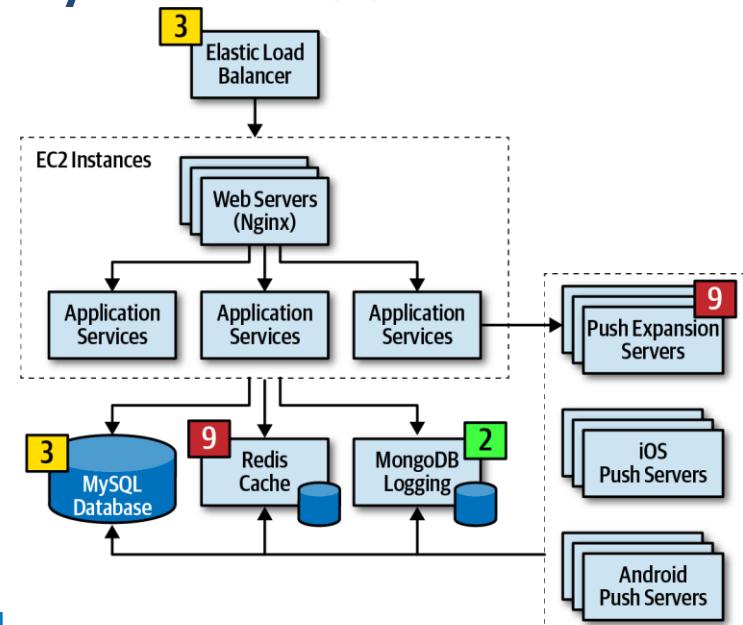
Risk storming

1. Draw the architectural diagrams
2. Team members individually identify risks of the architecture by placing post-it's on the diagrams



Risk storming

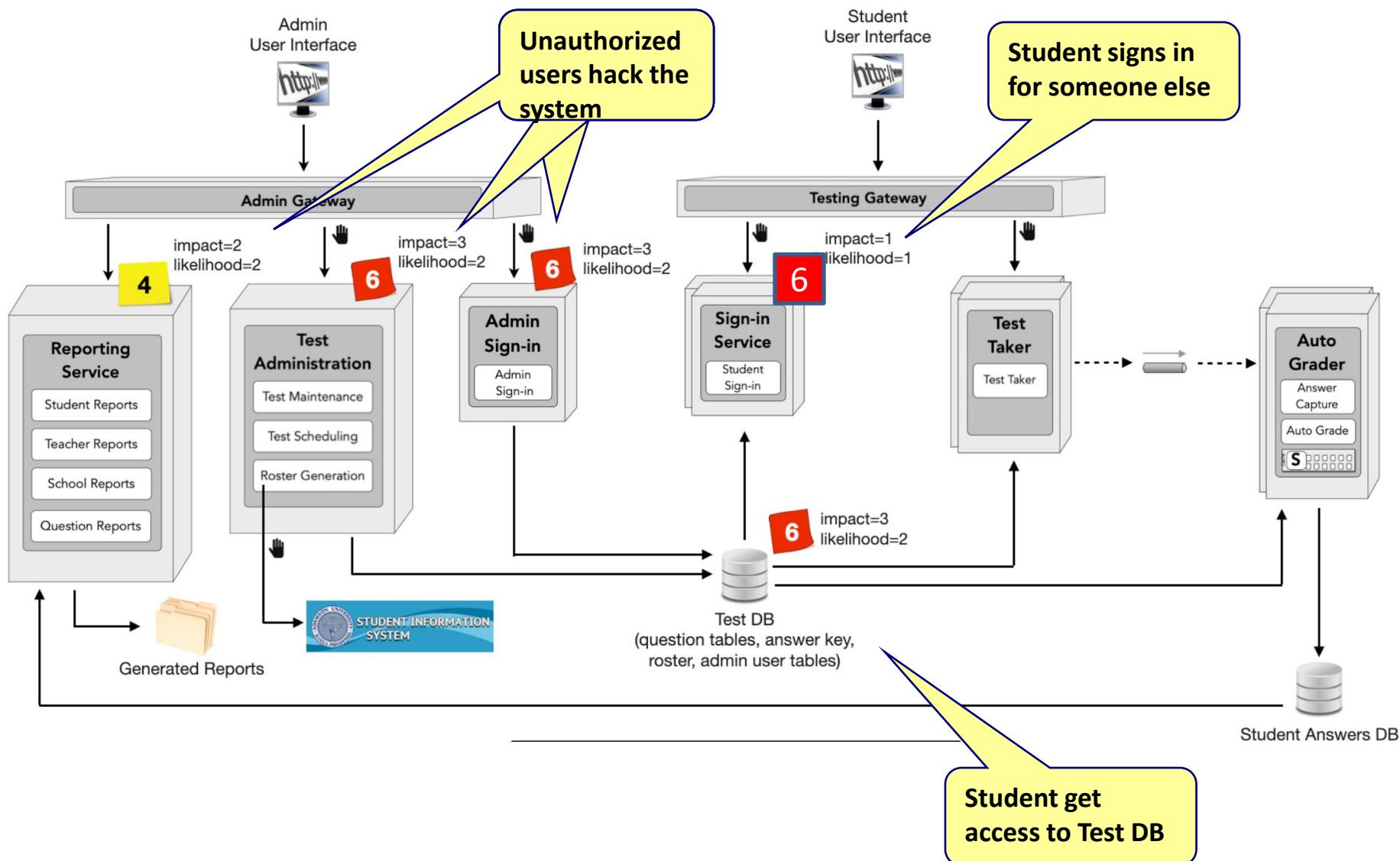
1. Draw the architectural diagrams
2. Team members individually identify risks of the architecture by placing post-it's on the diagrams
3. Discuss result and identify the most important risks



Risk storming

1. Draw the architectural diagrams
2. Team members individually identify risks of the architecture by placing post-it's on the diagrams
3. Discuss result and identify the most important risks
4. Mitigate the risks

Security risk

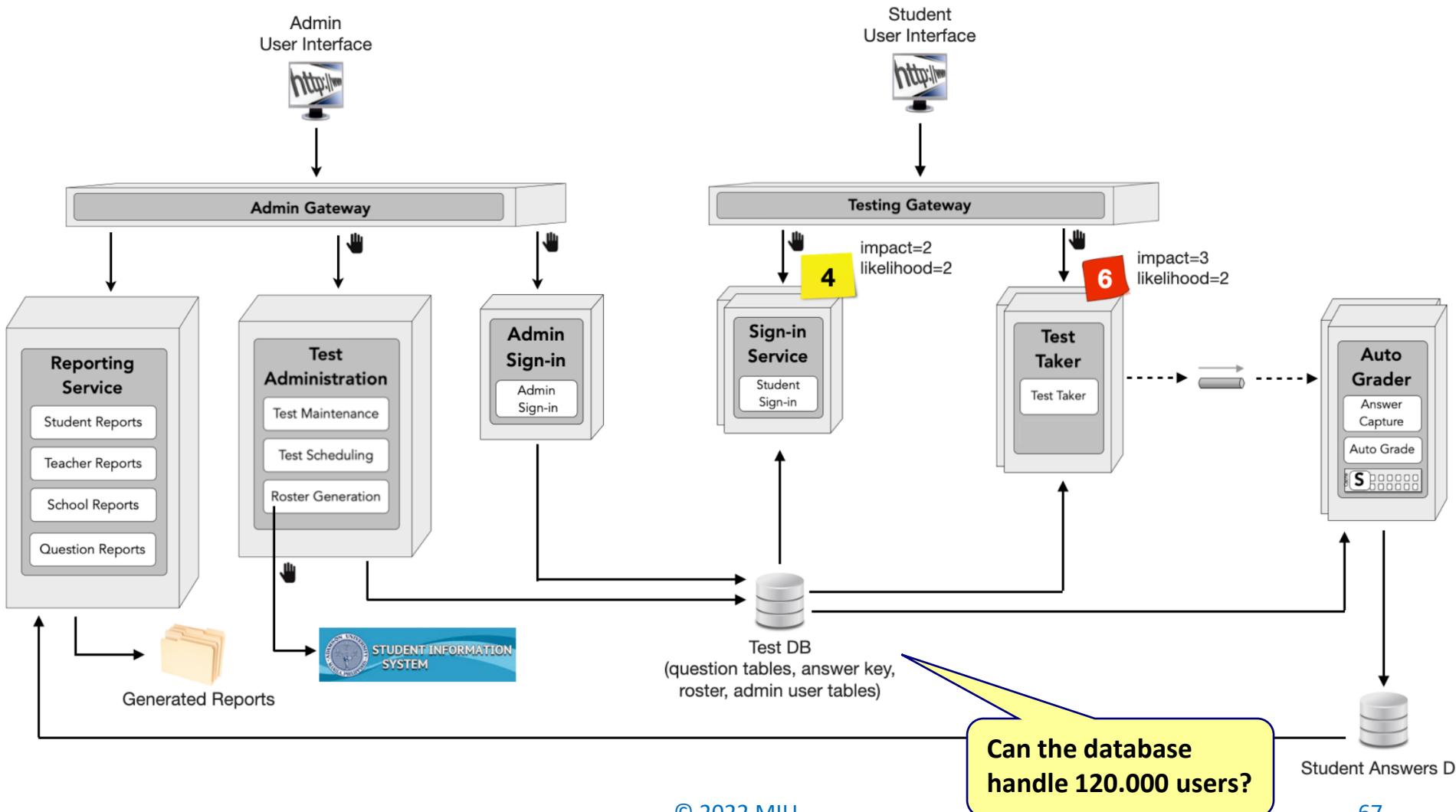


Security risk



Architecture Risk Assessment	Test Administration	Test Taking	Grading
Data Integrity			
Responsiveness			
Elasticity			
Security	6	6	1
Availability			
Feasibility (cost/time)			
Reliability			

Responsiveness risk

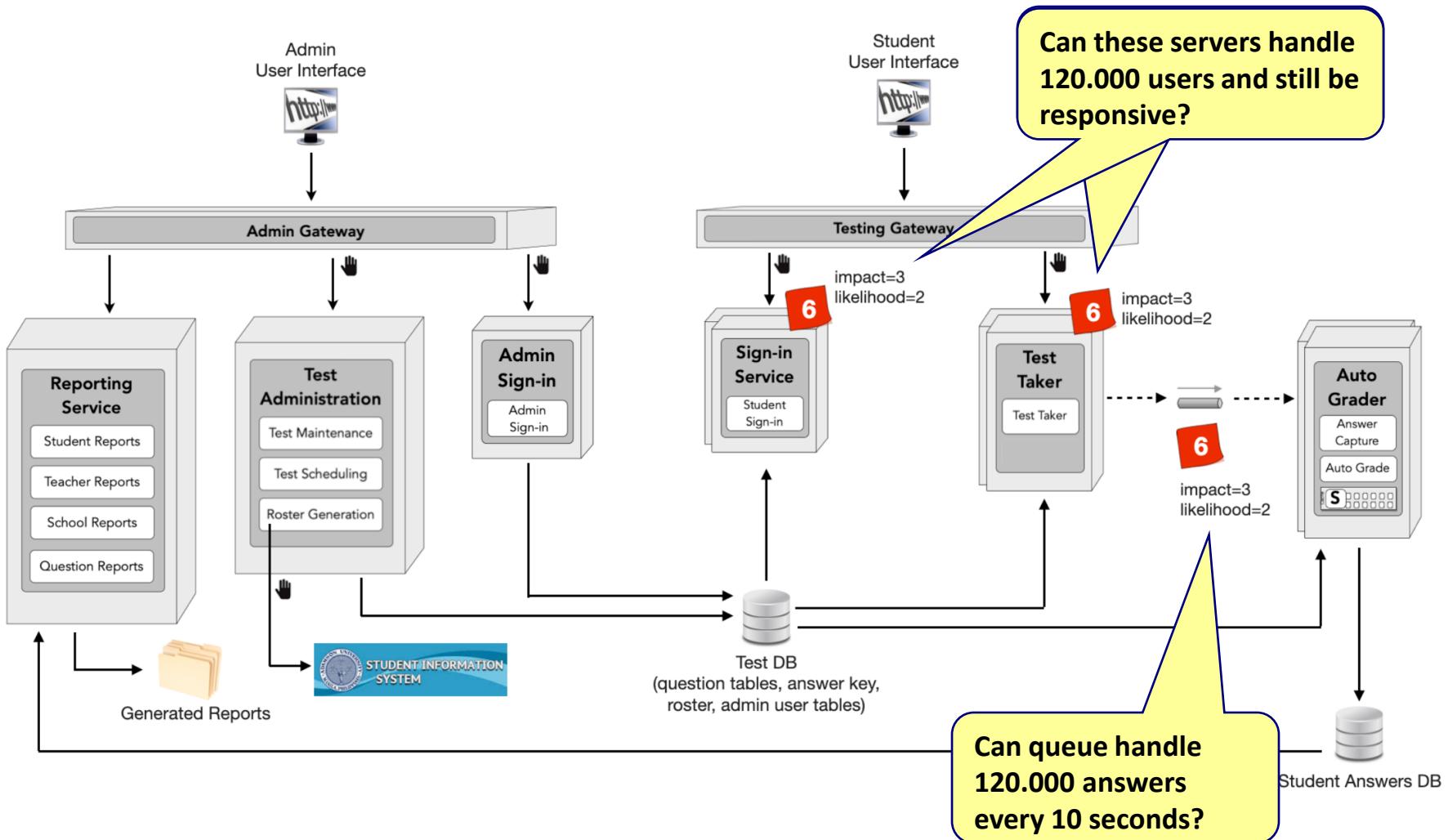


Responsiveness risk



Architecture Risk Assessment	Test Administration	Test Taking	Grading
Data Integrity			
Responsiveness	1	6	1
Elasticity			
Security	6	6	1
Availability			
Feasibility (cost/time)			
Reliability			

Elasticity risk

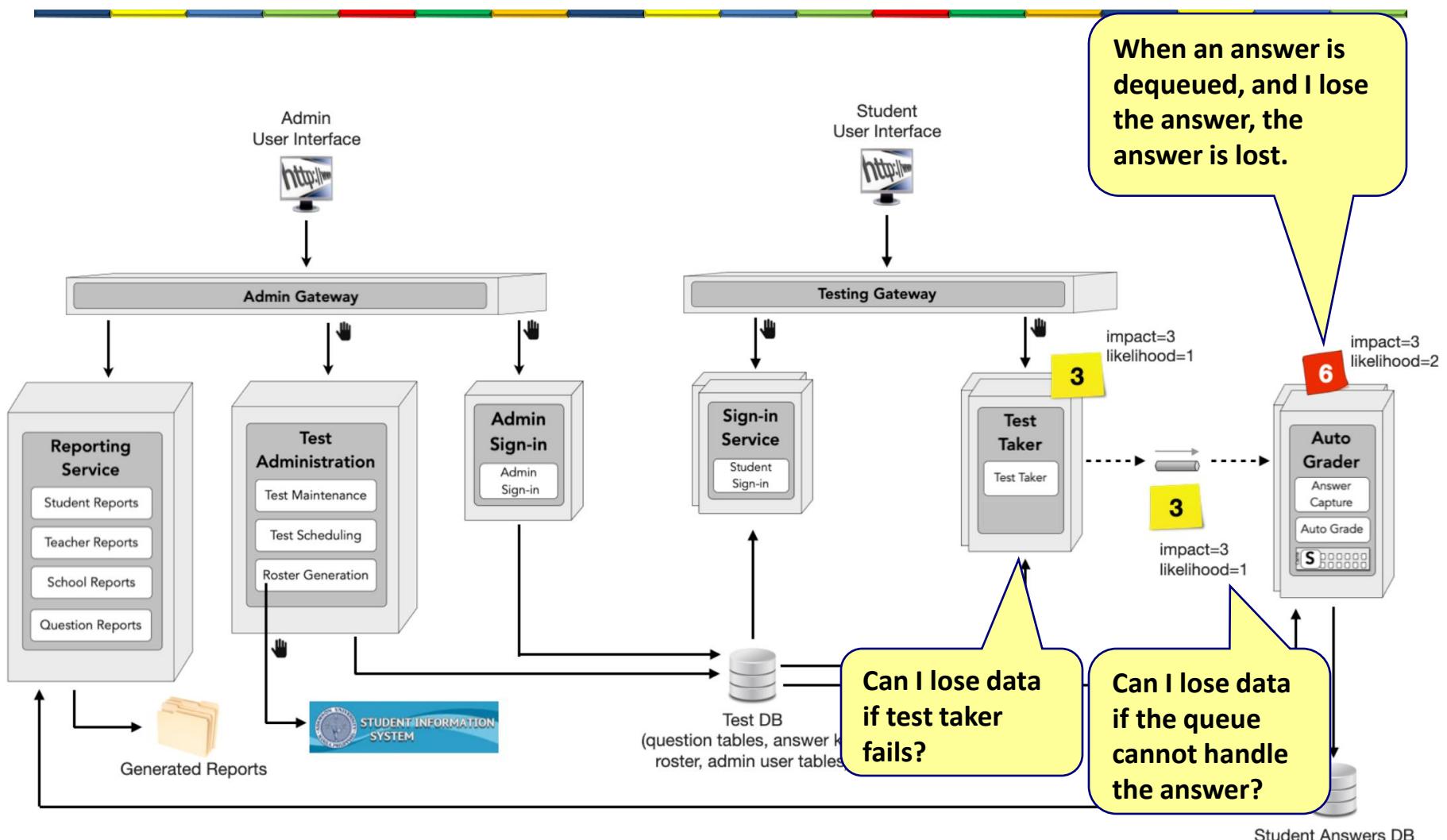


Elasticity risk



Architecture Risk Assessment	Test Administration	Test Taking	Grading
Data Integrity			
Responsiveness	1	6	1
Elasticity	1	6	1
Security	6	6	1
Availability			
Feasibility (cost/time)			
Reliability			

Data integrity risk



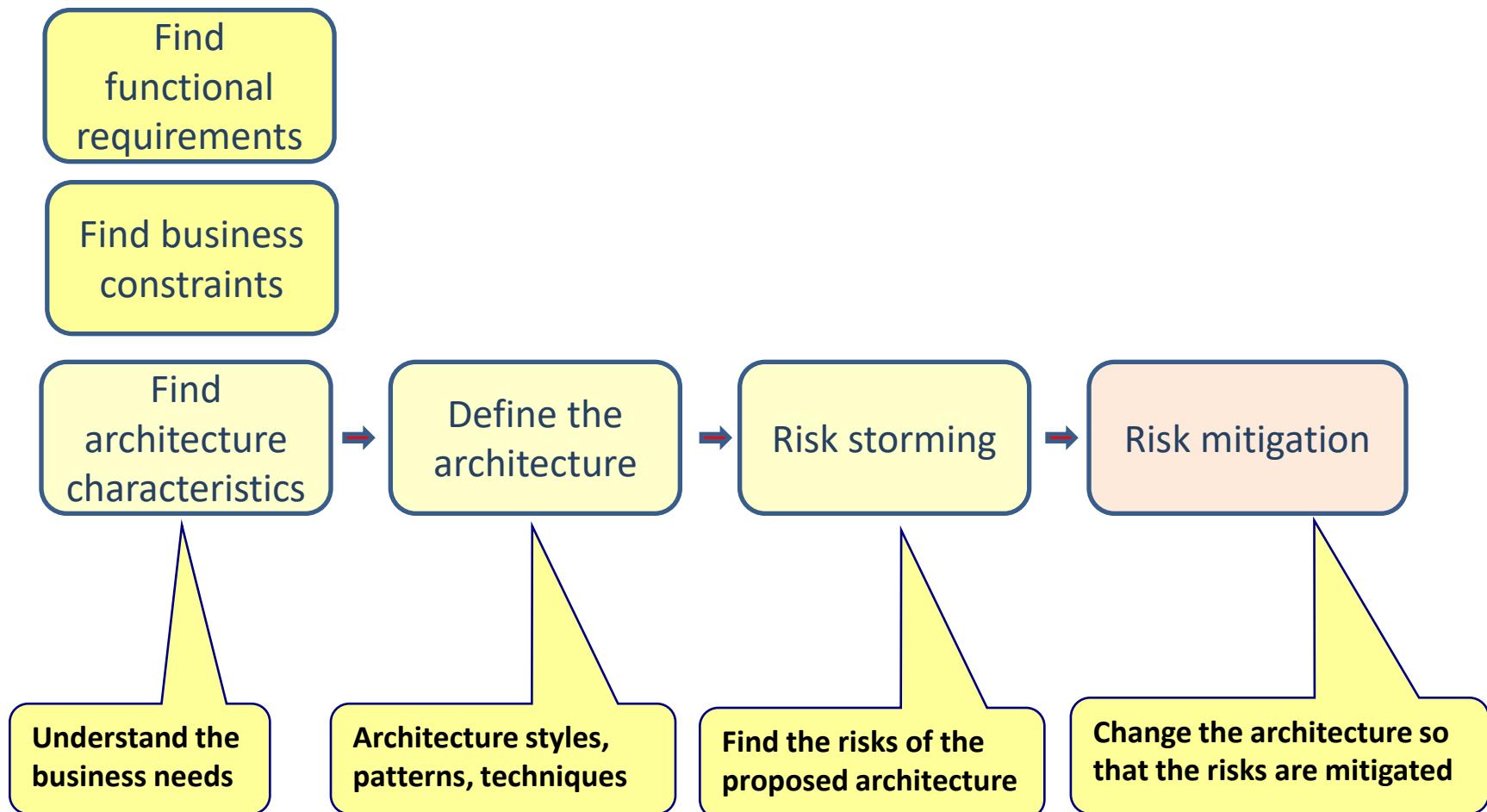
Risk storming result



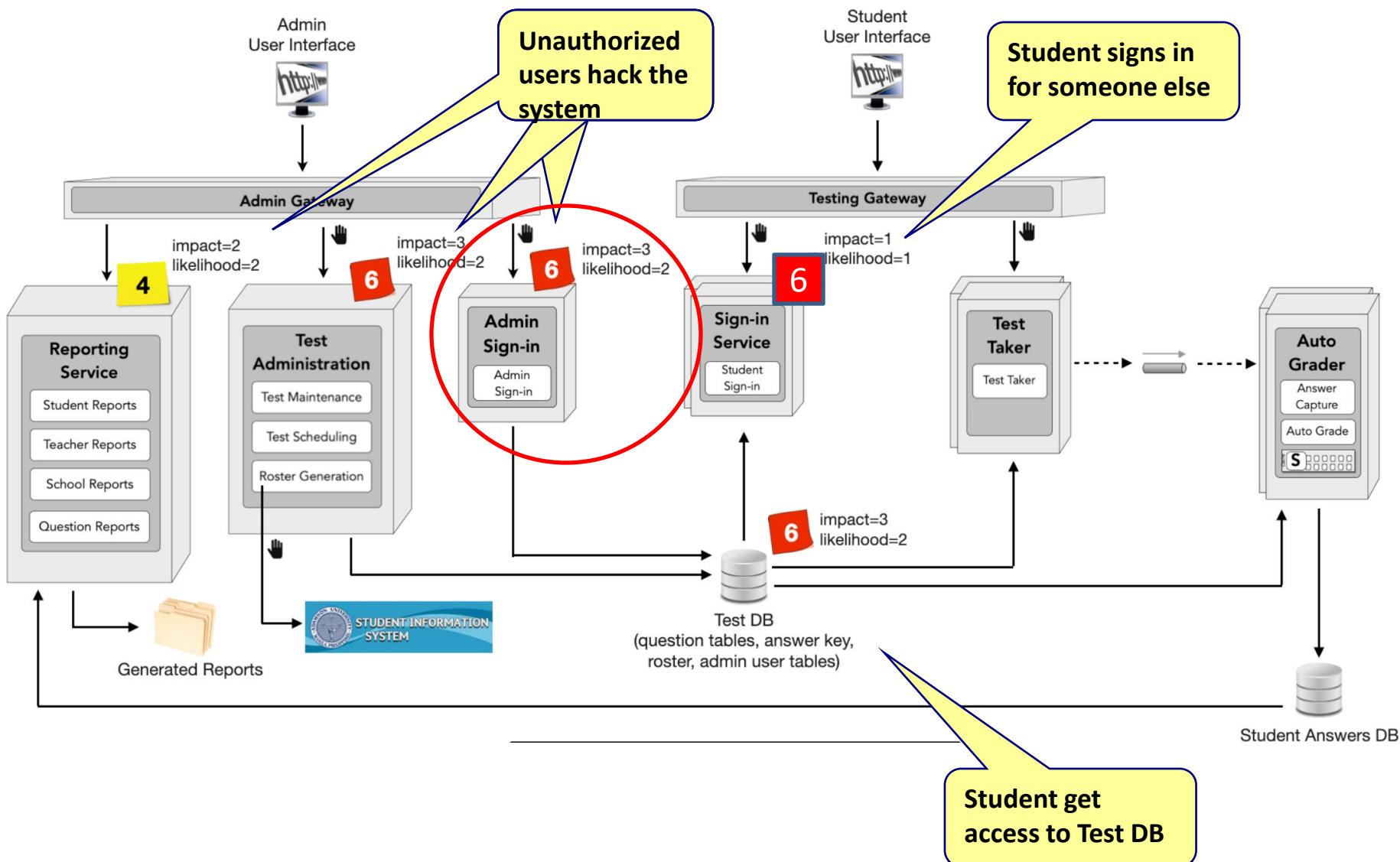
Architecture Risk Assessment	Test Administration	Test Taking	Grading
Data Integrity	1	3	6
Responsiveness	1	6	1
Elasticity	1	6	1
Security	6	6	1
Availability			
Feasibility (cost/time)			
Reliability			

MITIGATING RISK

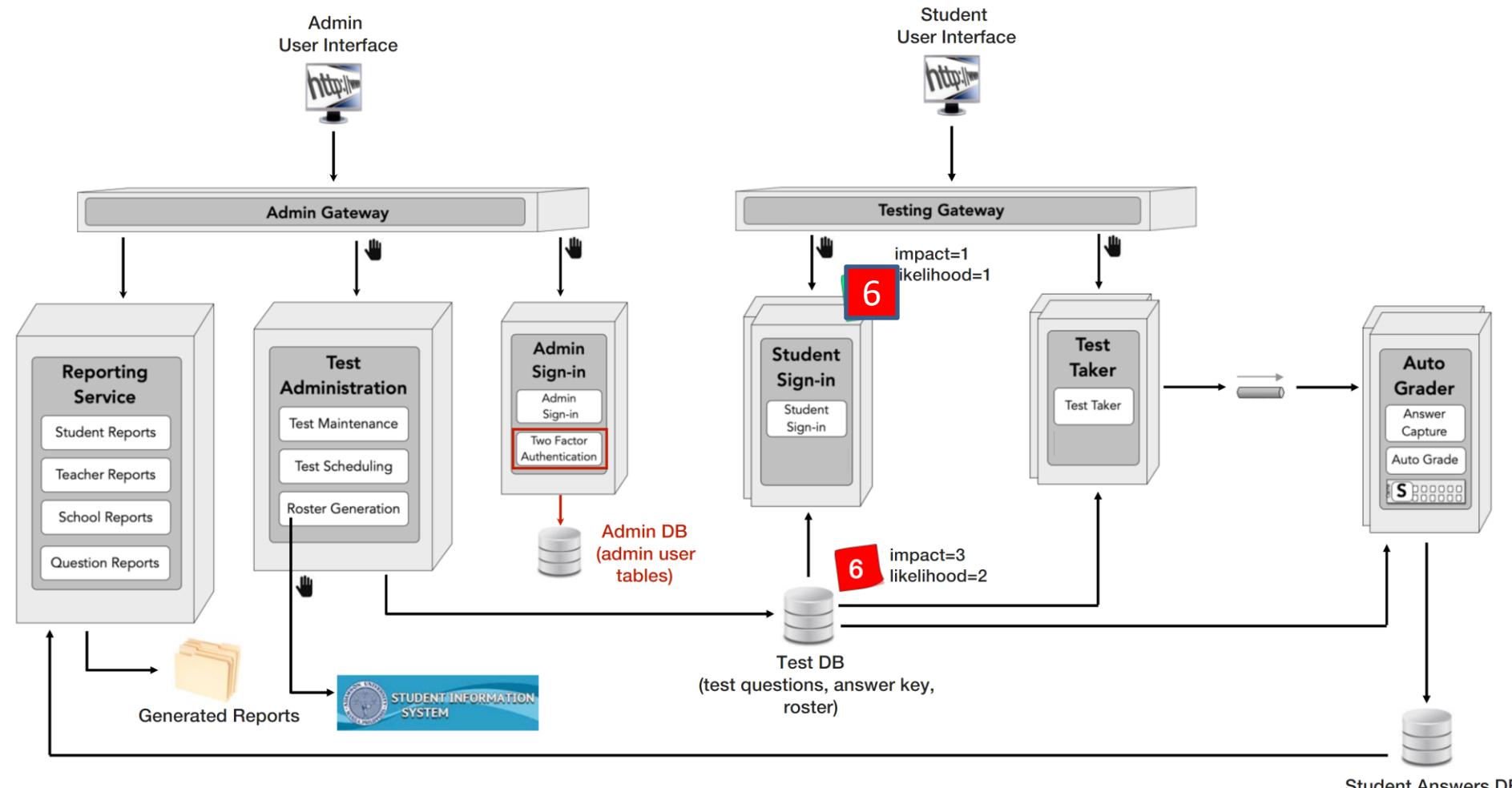
Risk mitigation



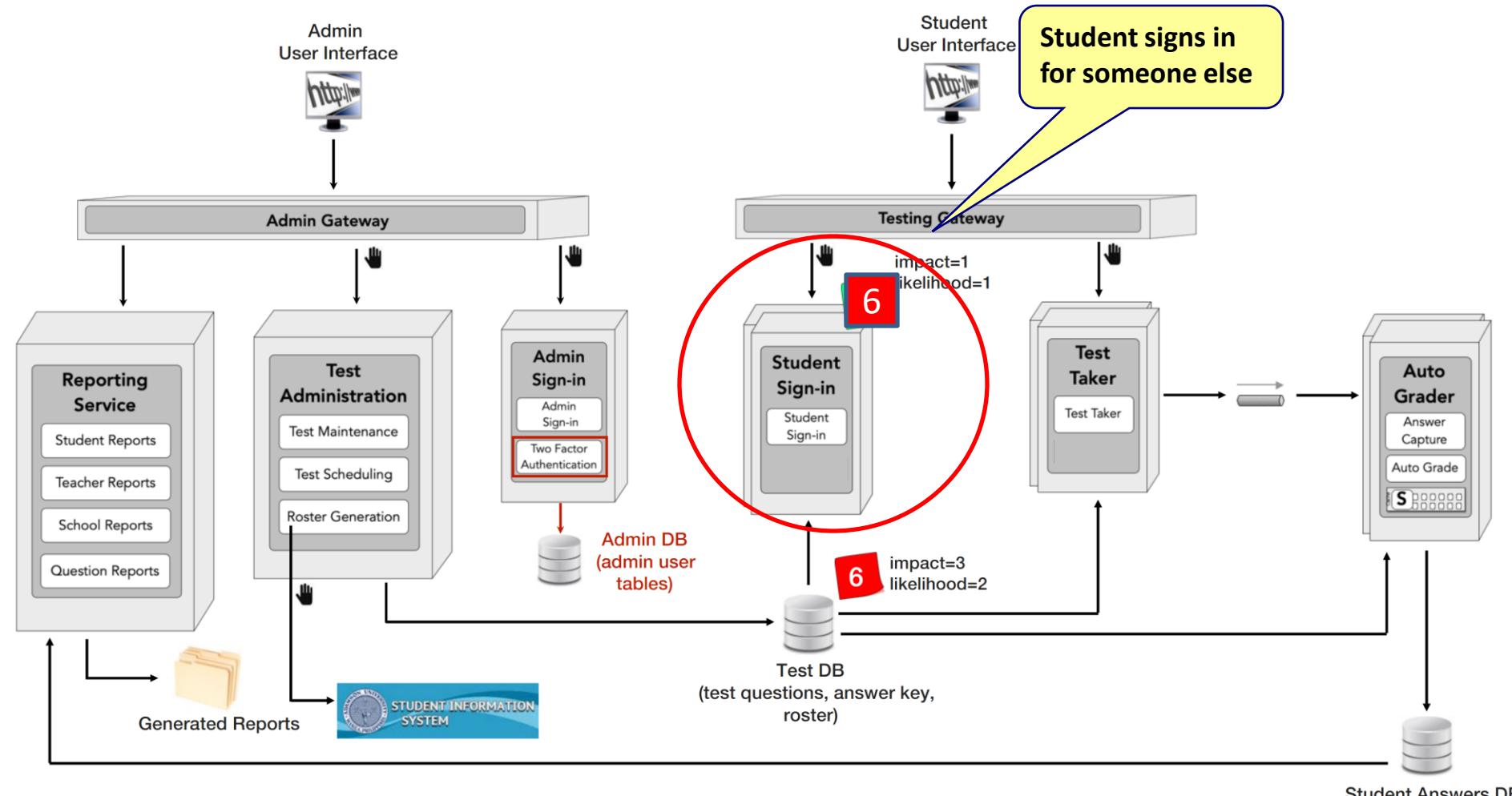
Security risk



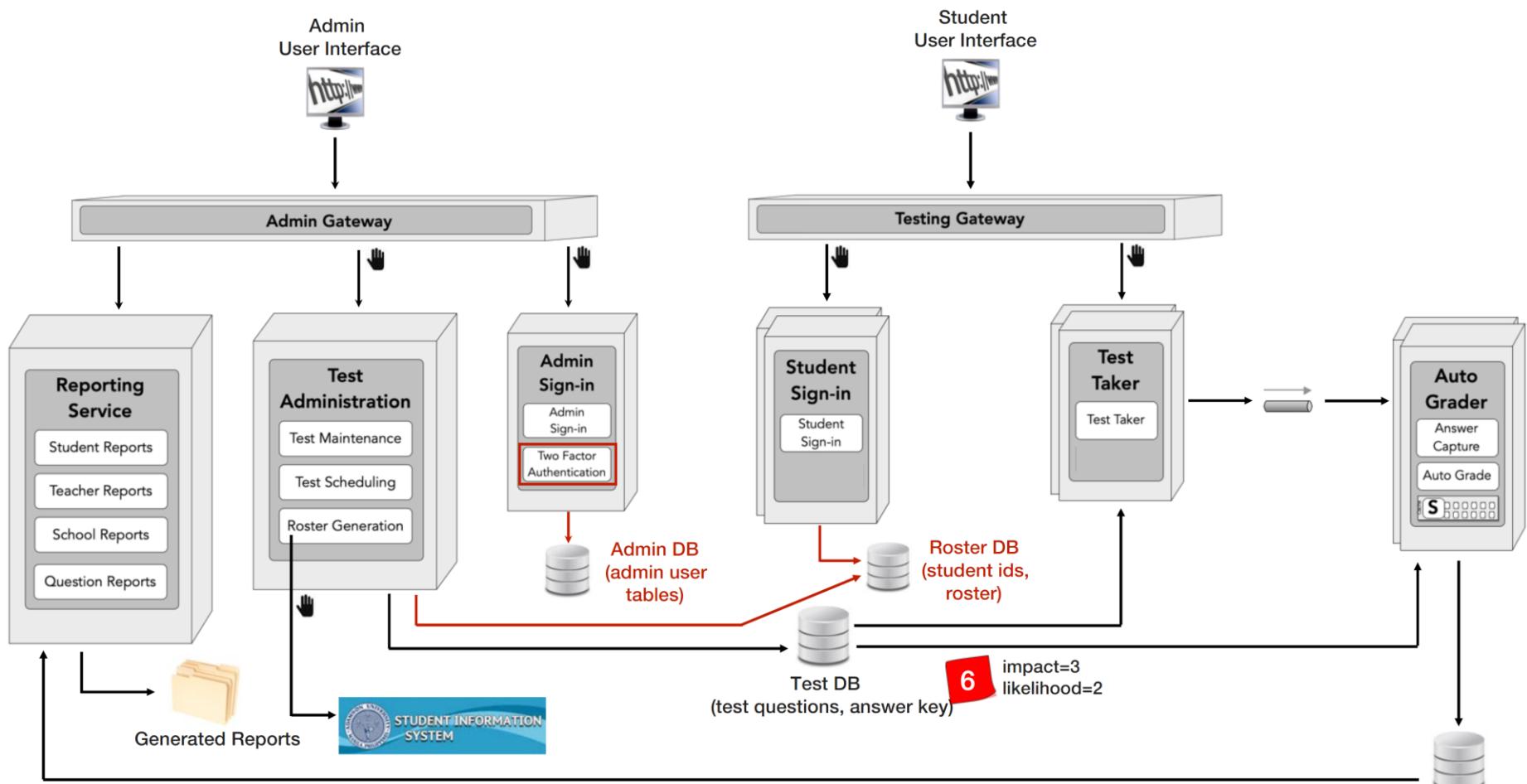
Security risk mitigation



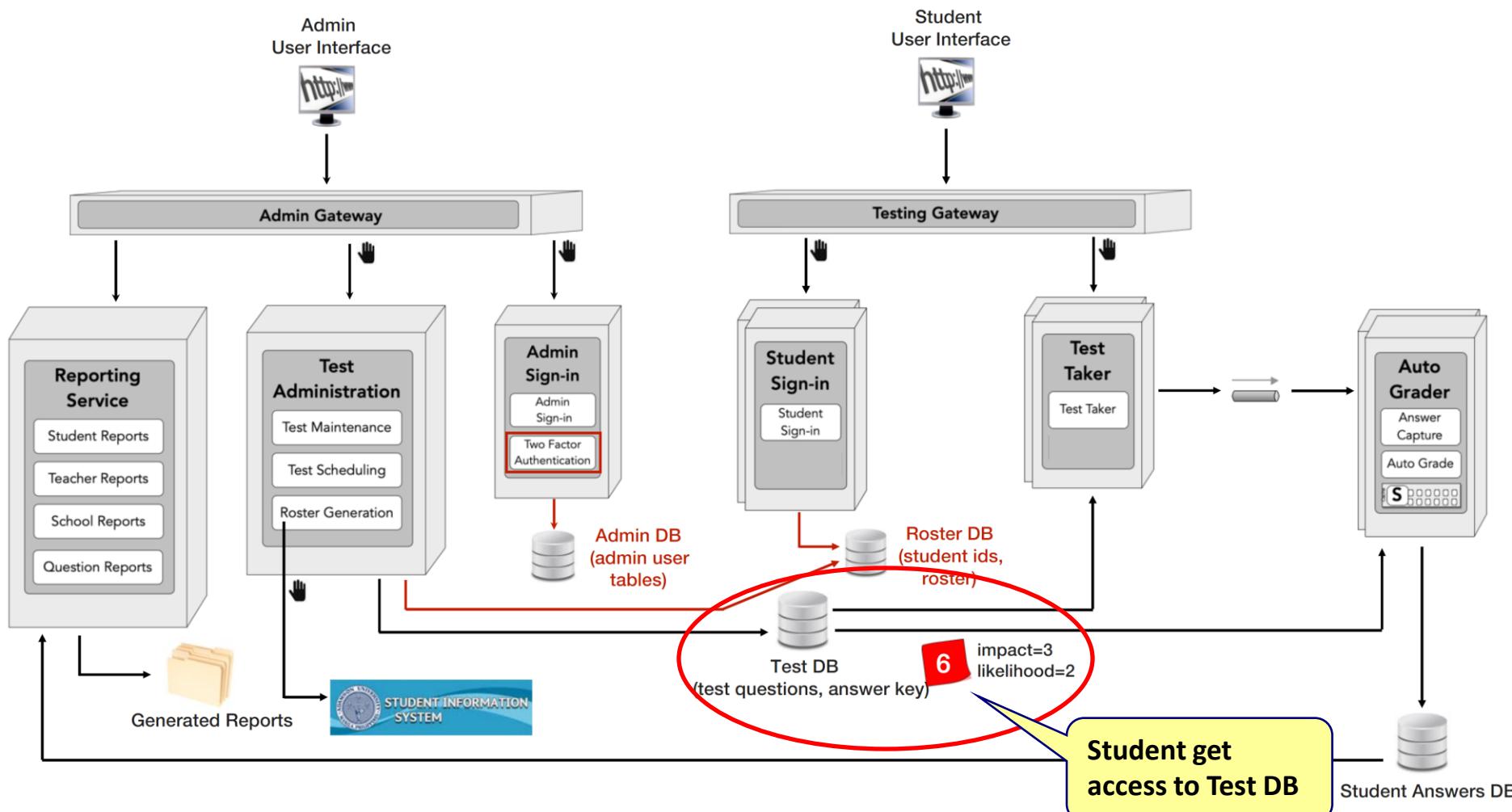
Security risk mitigation



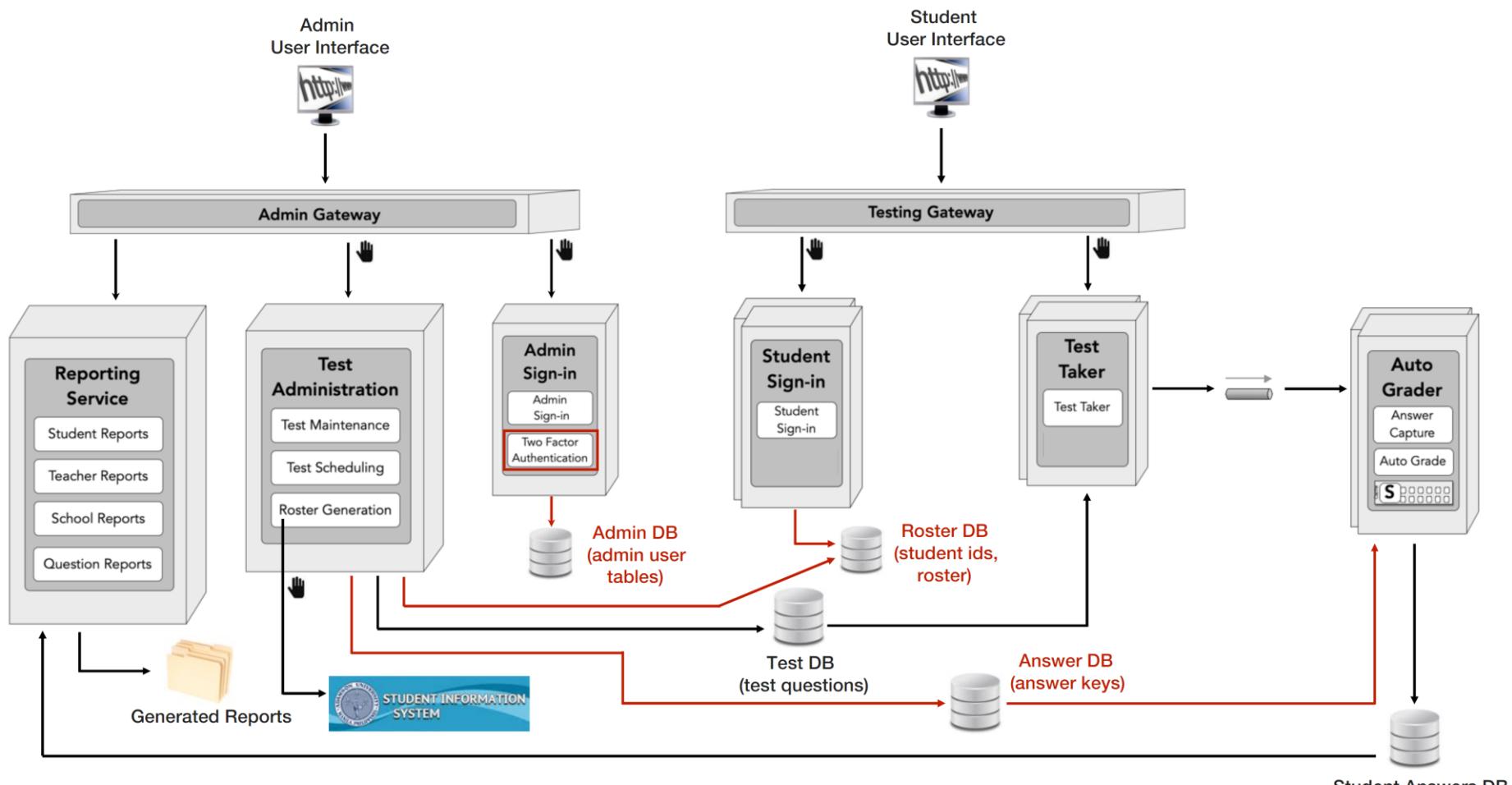
Security risk mitigation



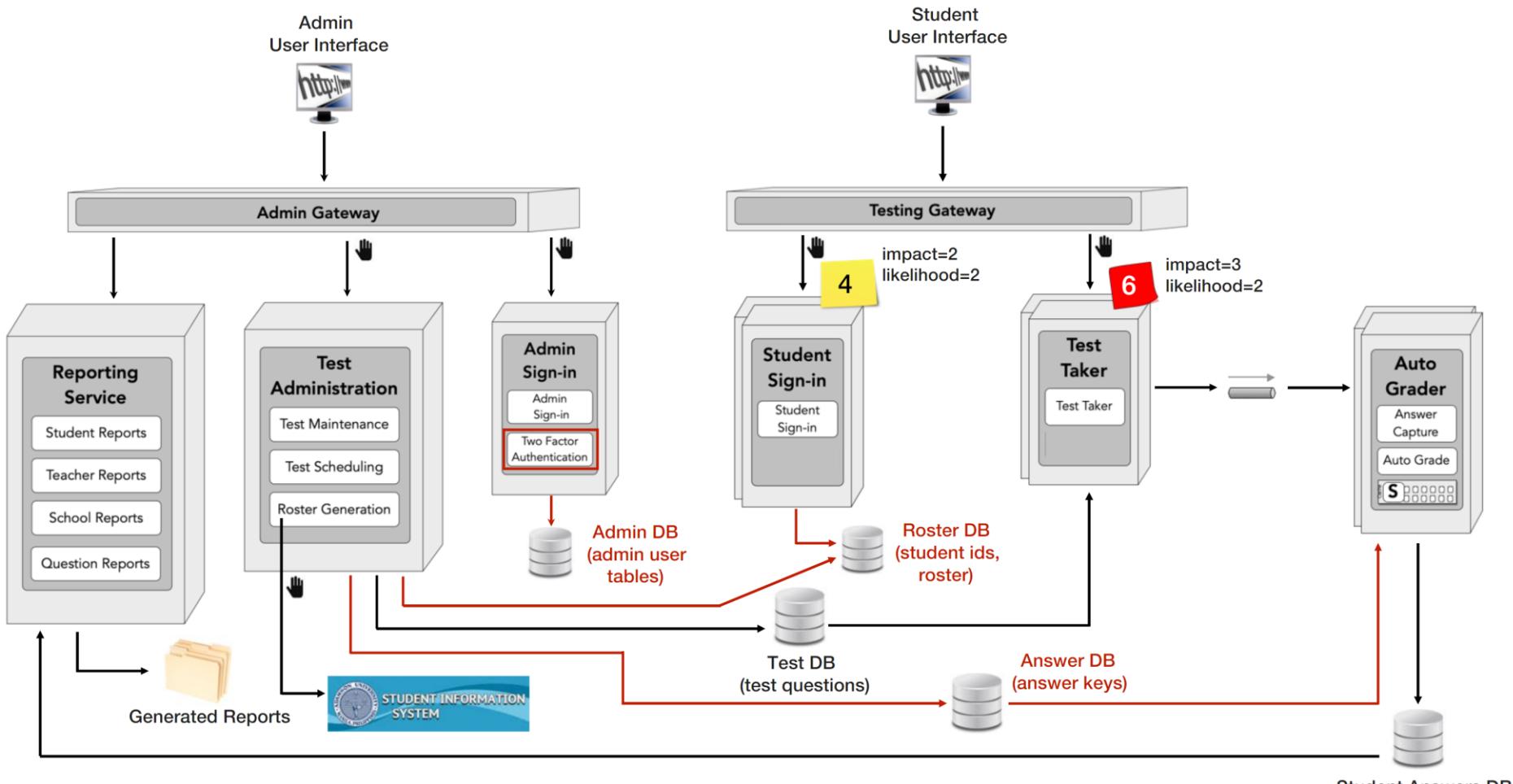
Security risk mitigation



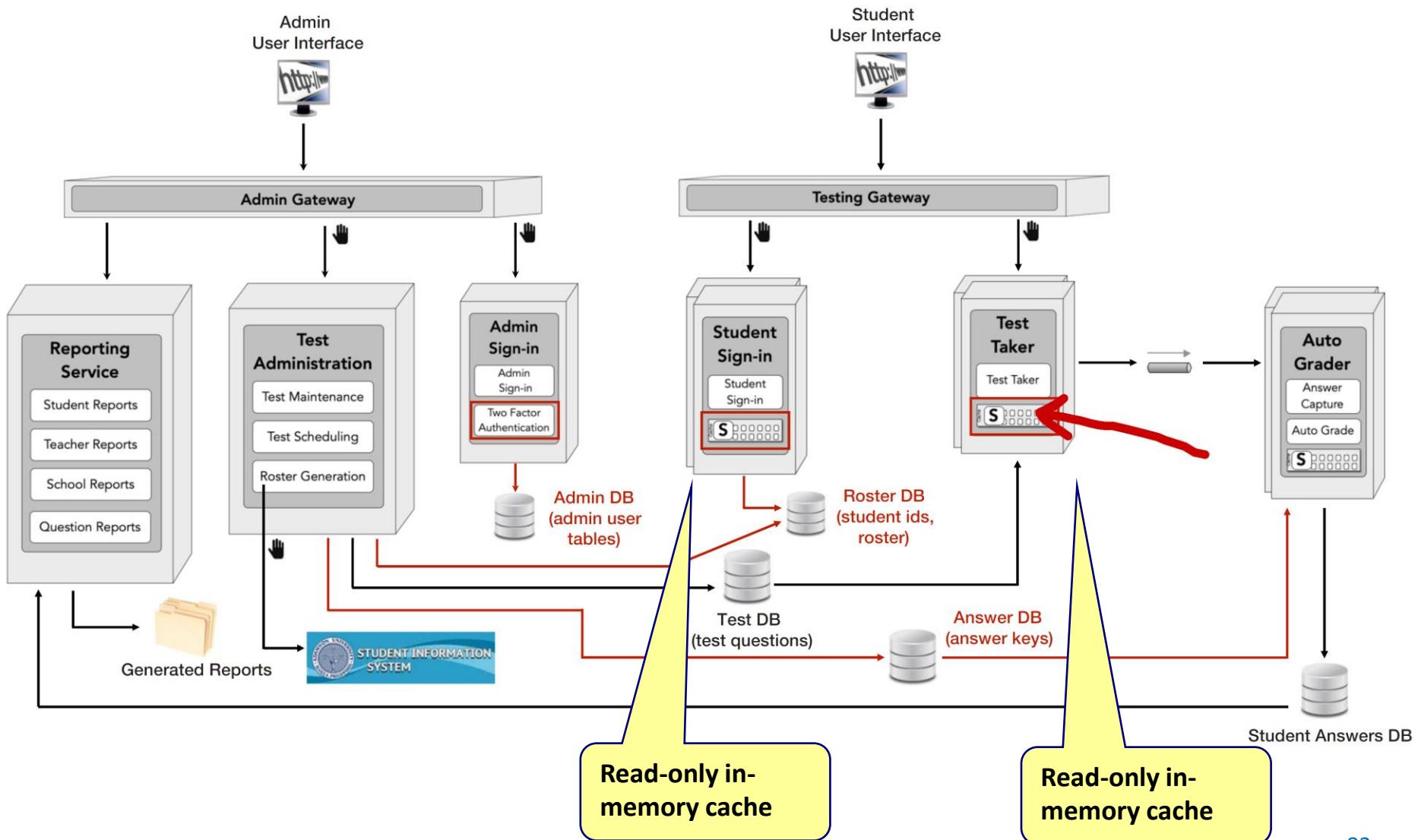
Security risk mitigation



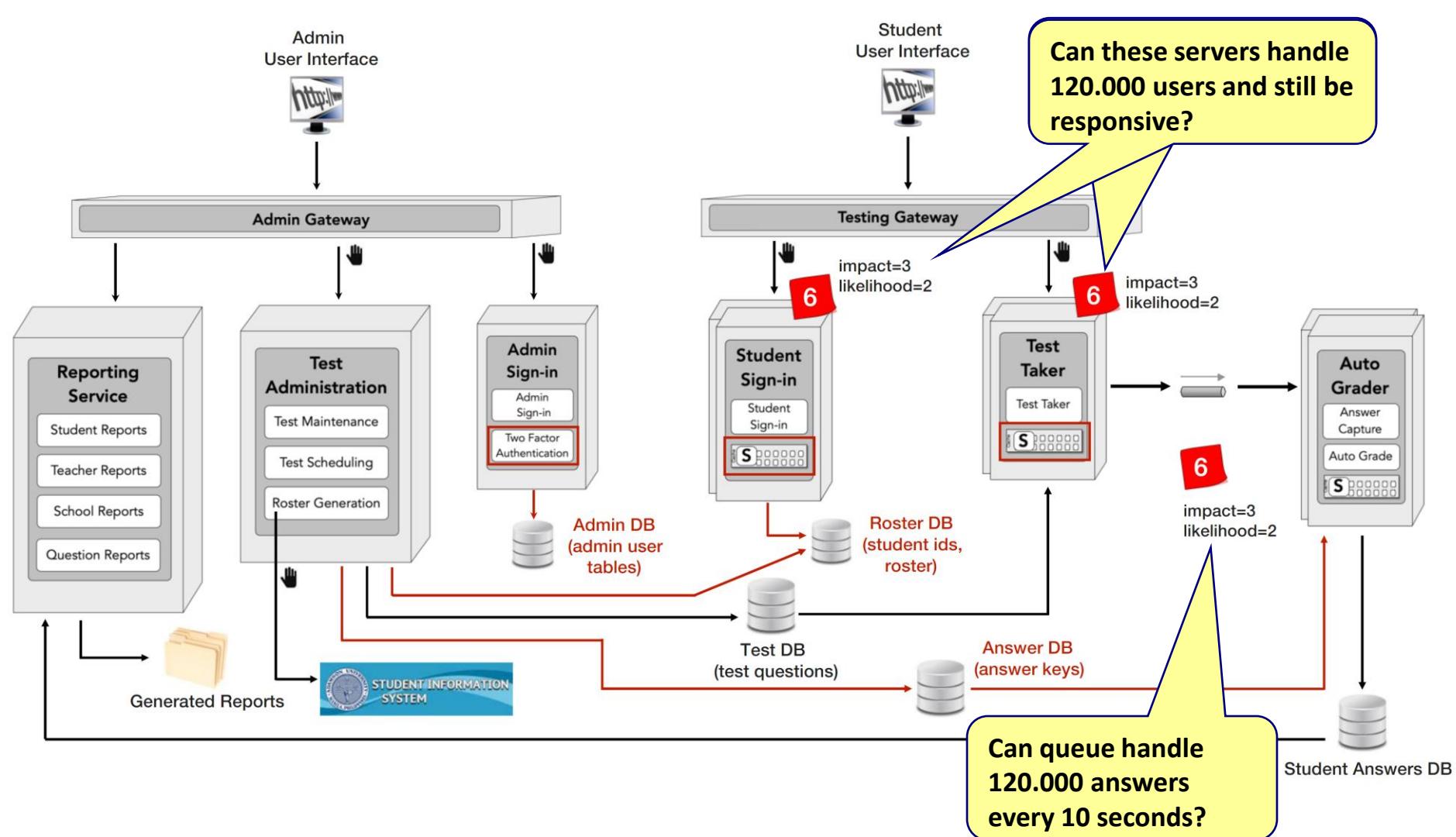
Responsiveness



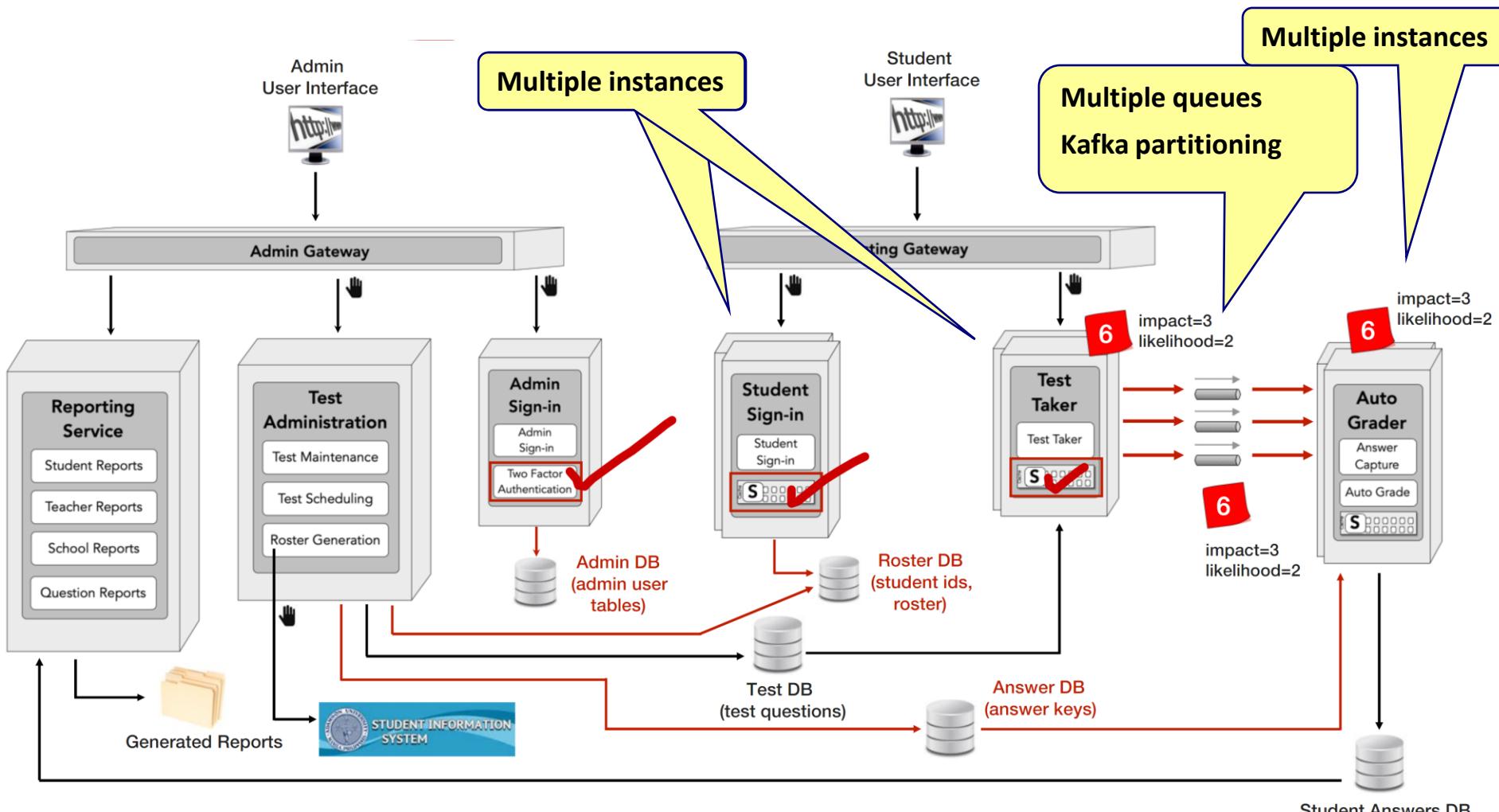
Responsiveness risk mitigation



Elasticity



Elasticity risk migration



Software architecture

- You do this every time the architecture requirements and/or the architecture changes.

