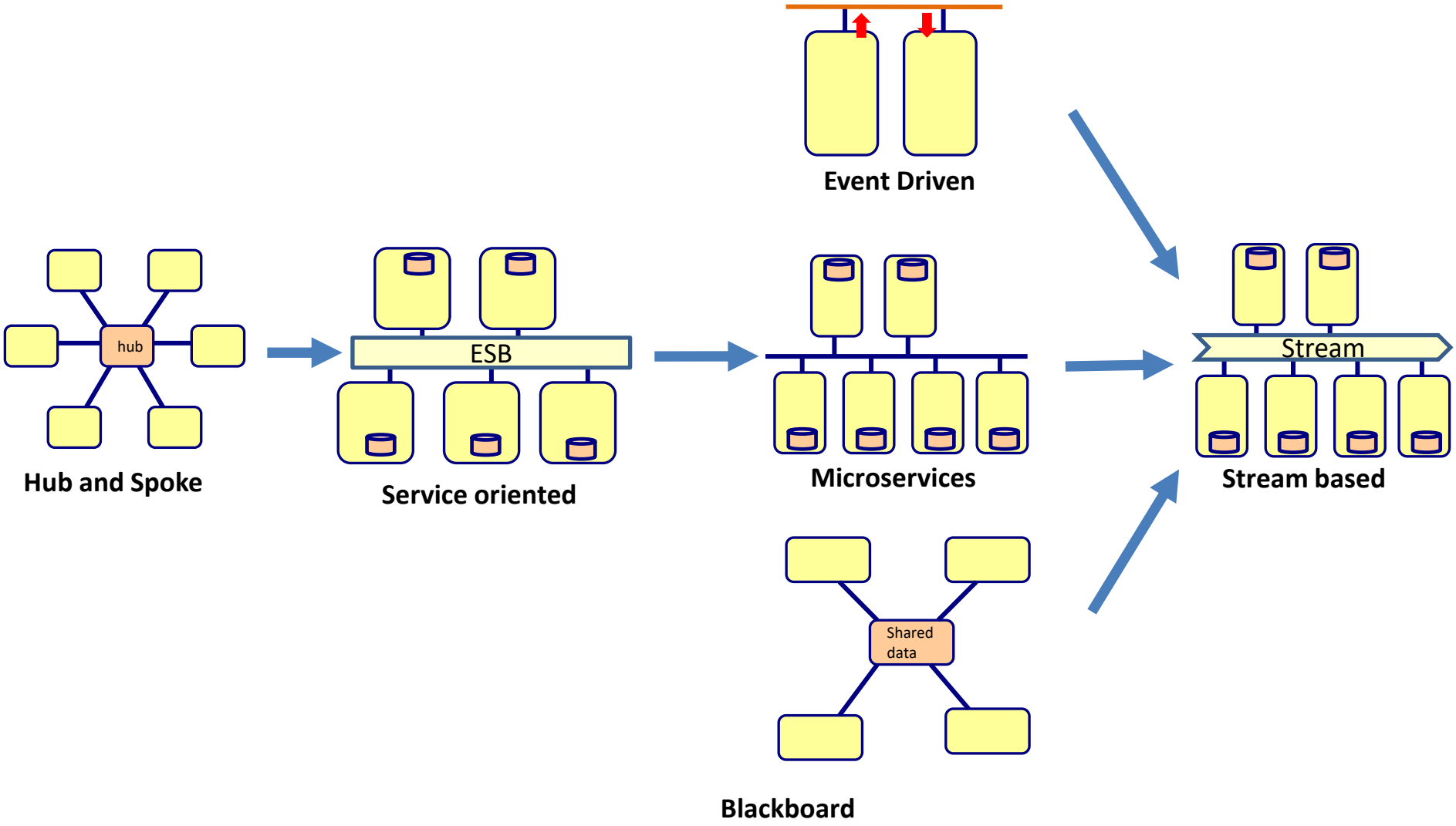
Lesson 12

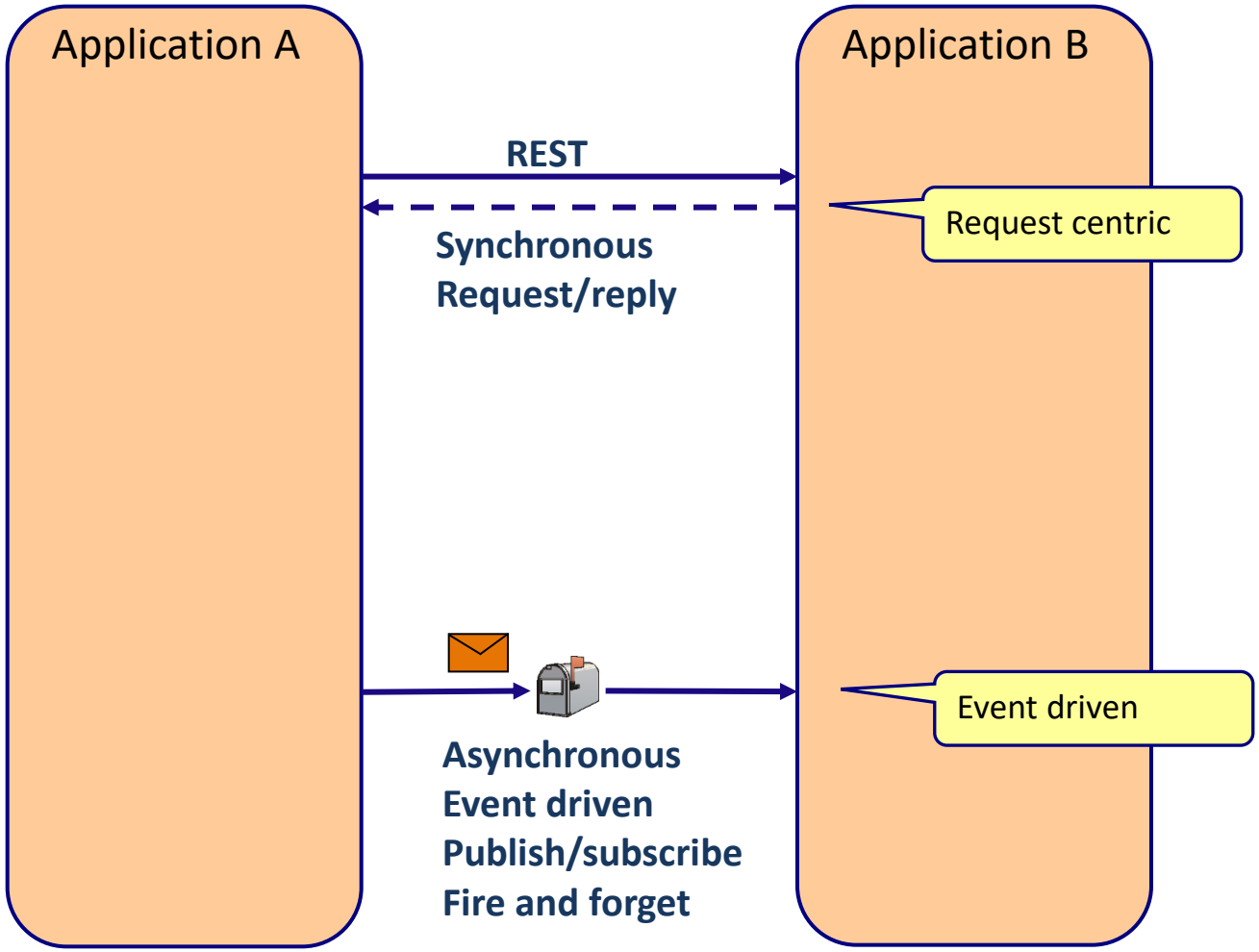
STREAM BASED ARCHITECTURE

Architecture evolution

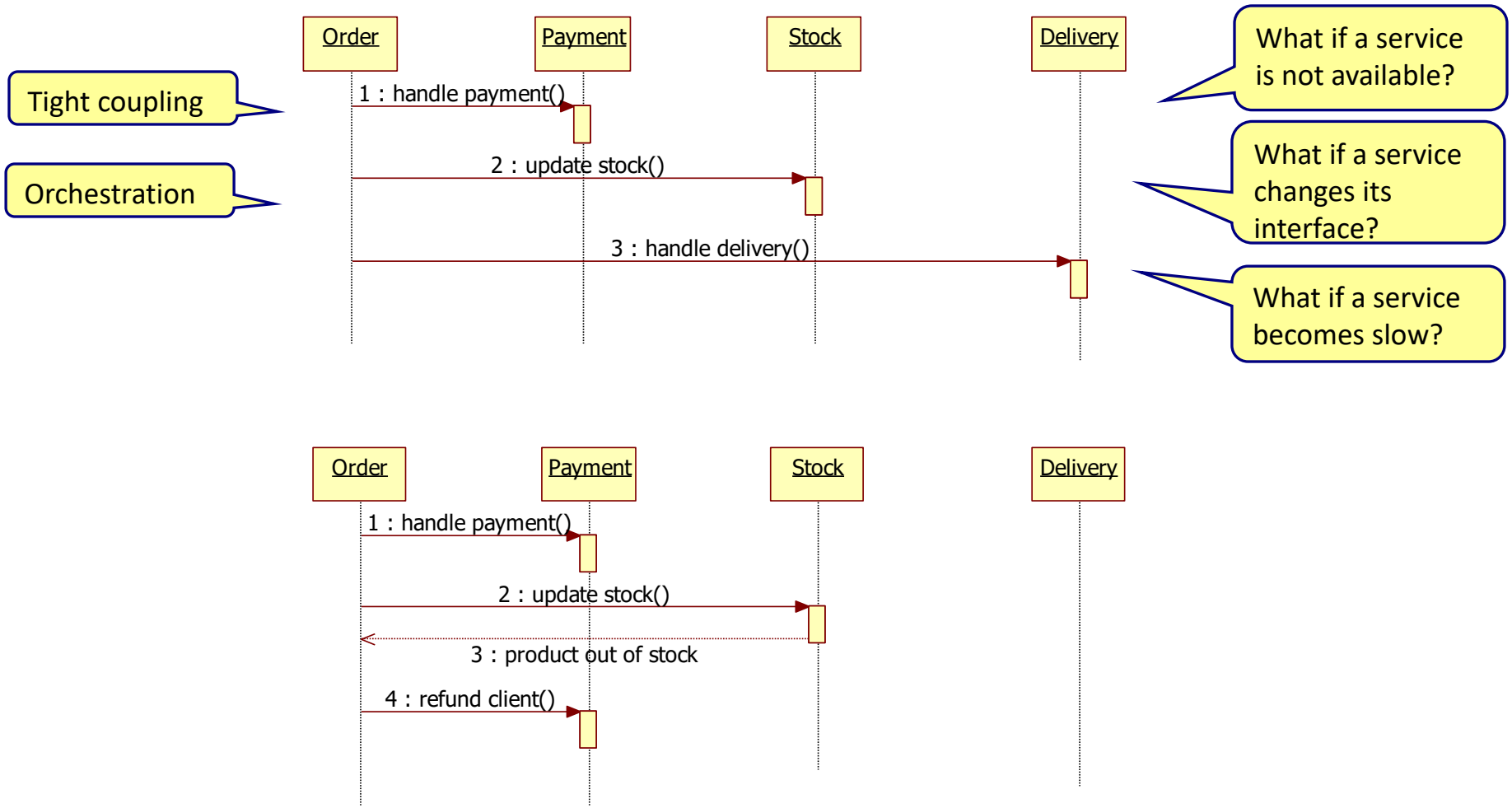


EVENT DRIVEN ARCHITECTURE

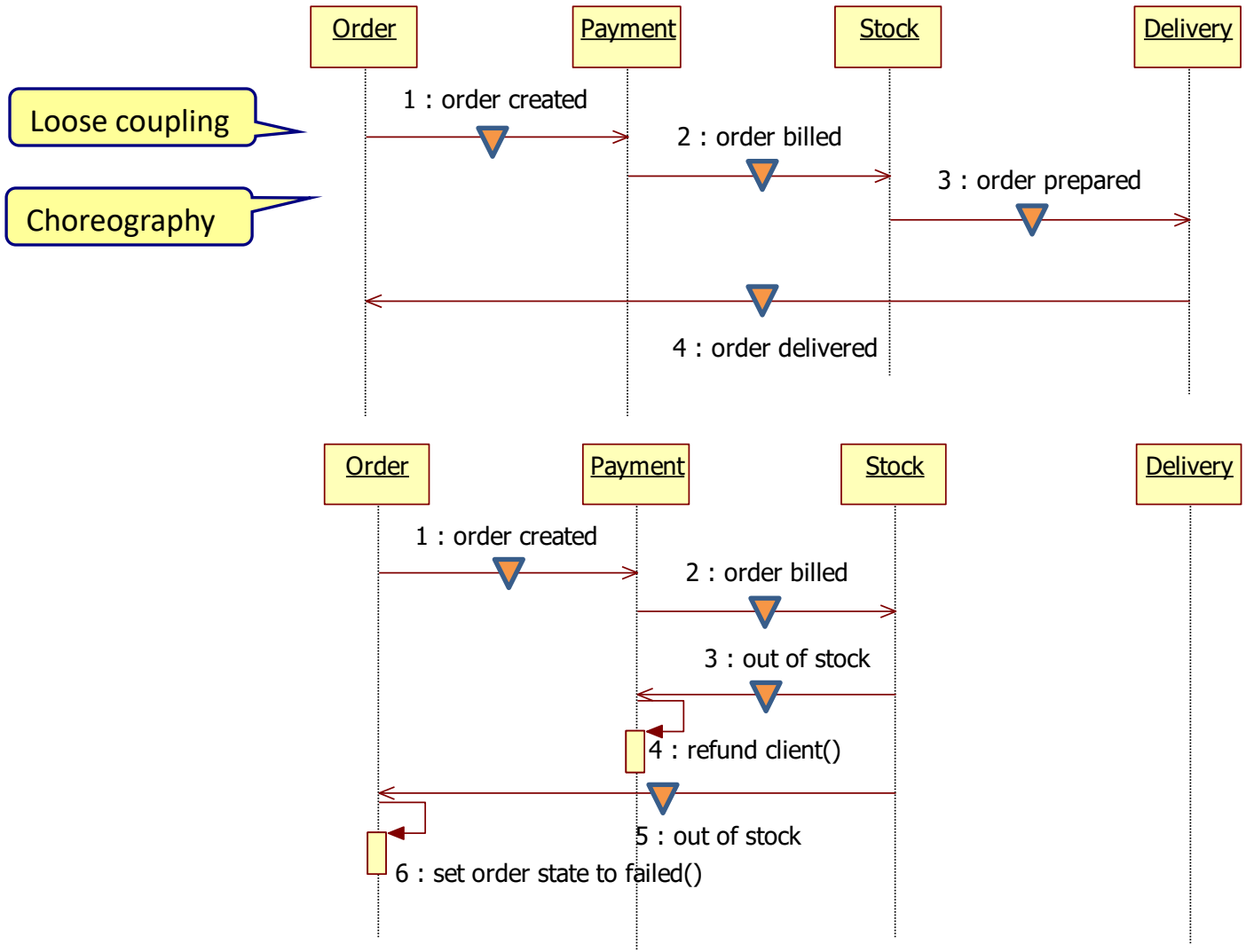
2 ways to communicate



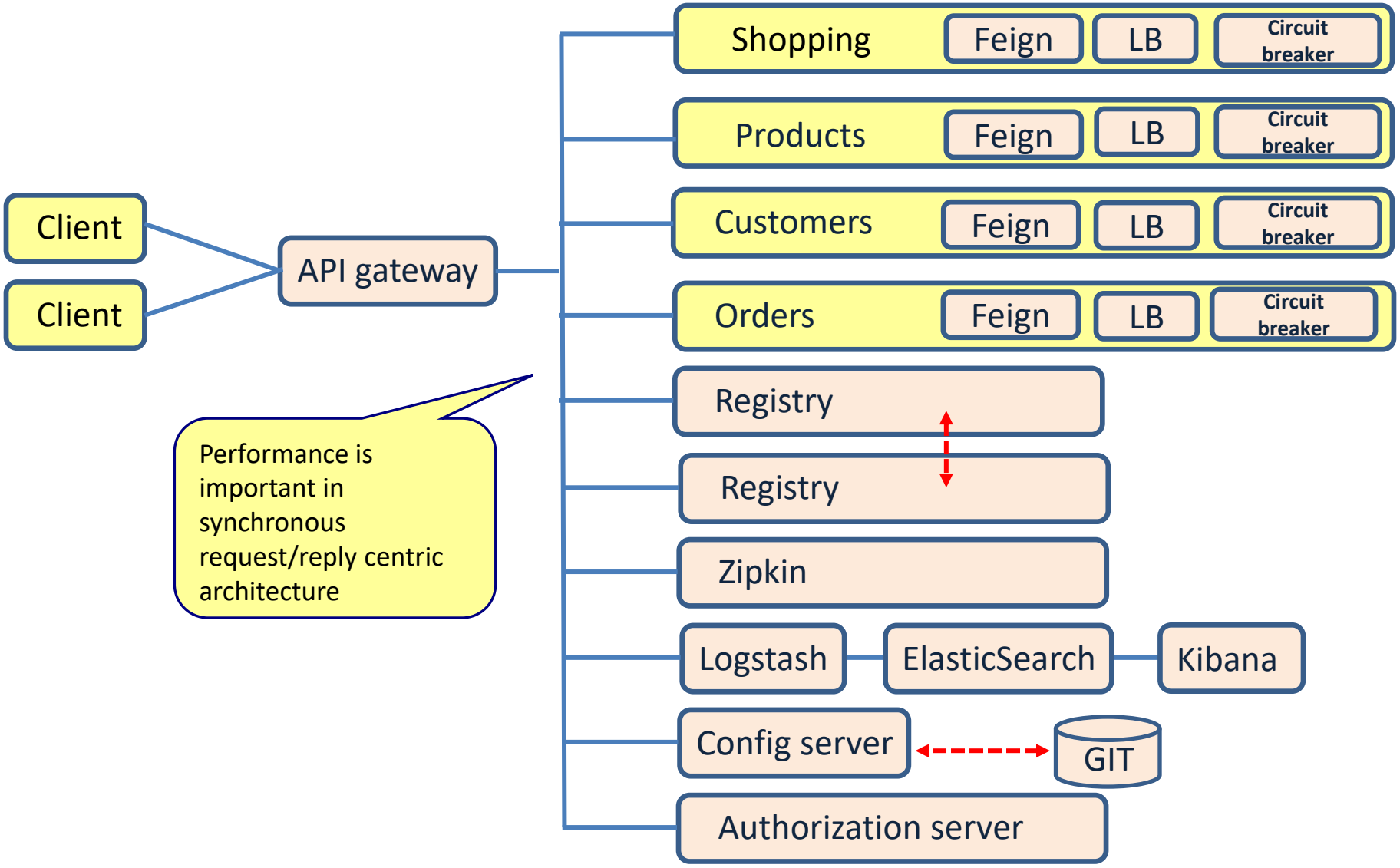
Request centric (REST) calls



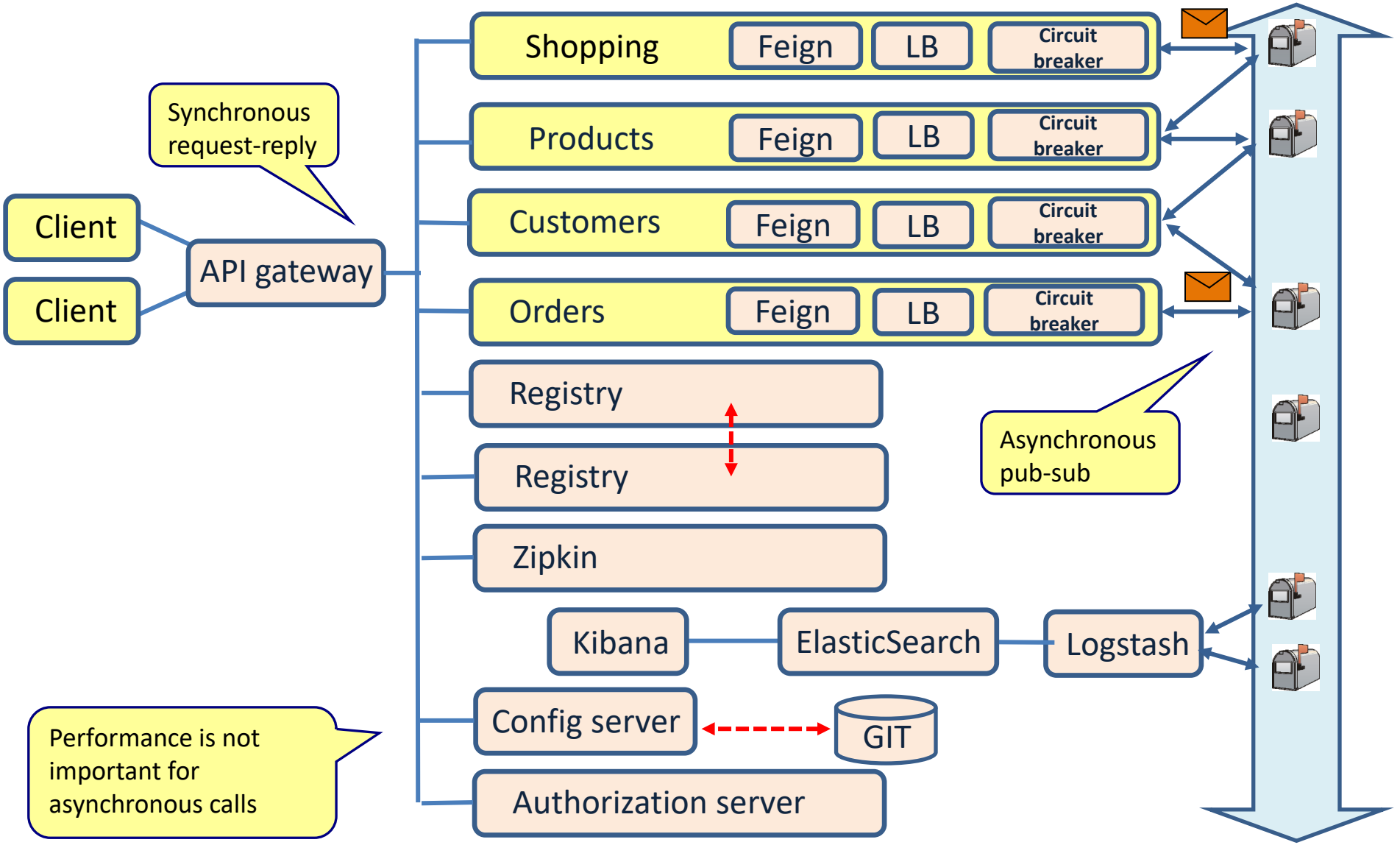
Event driven(messaging)



Implementing microservices



Implementing microservices



Main point

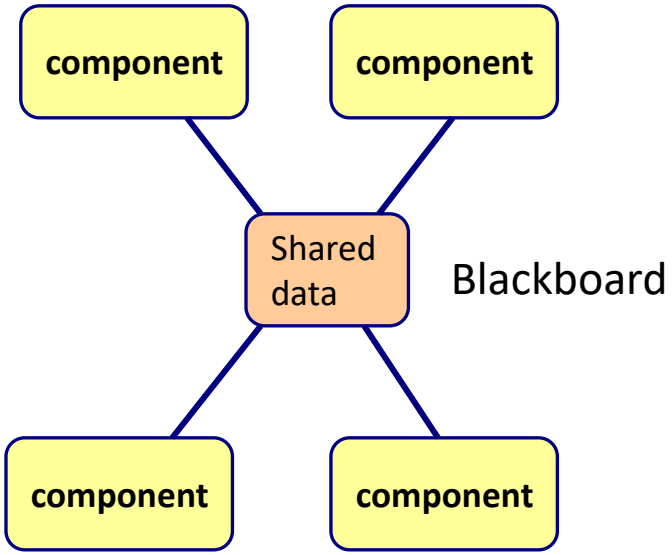
- Messaging gives loose coupling between the sender and the receiver.

  Science of Consciousness: The whole relative creation is an expression of the same one unified field.

BLACKBOARD

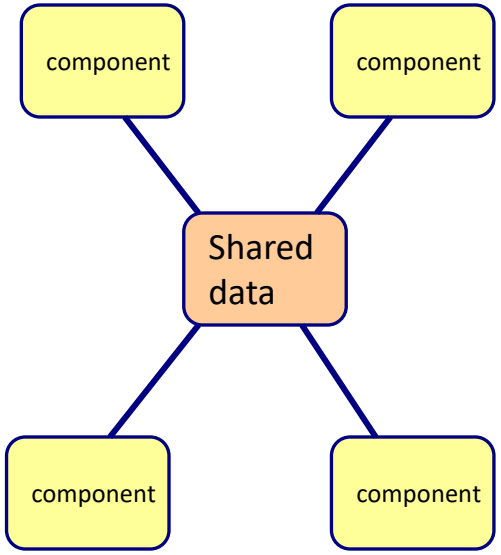
Blackboard

- 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

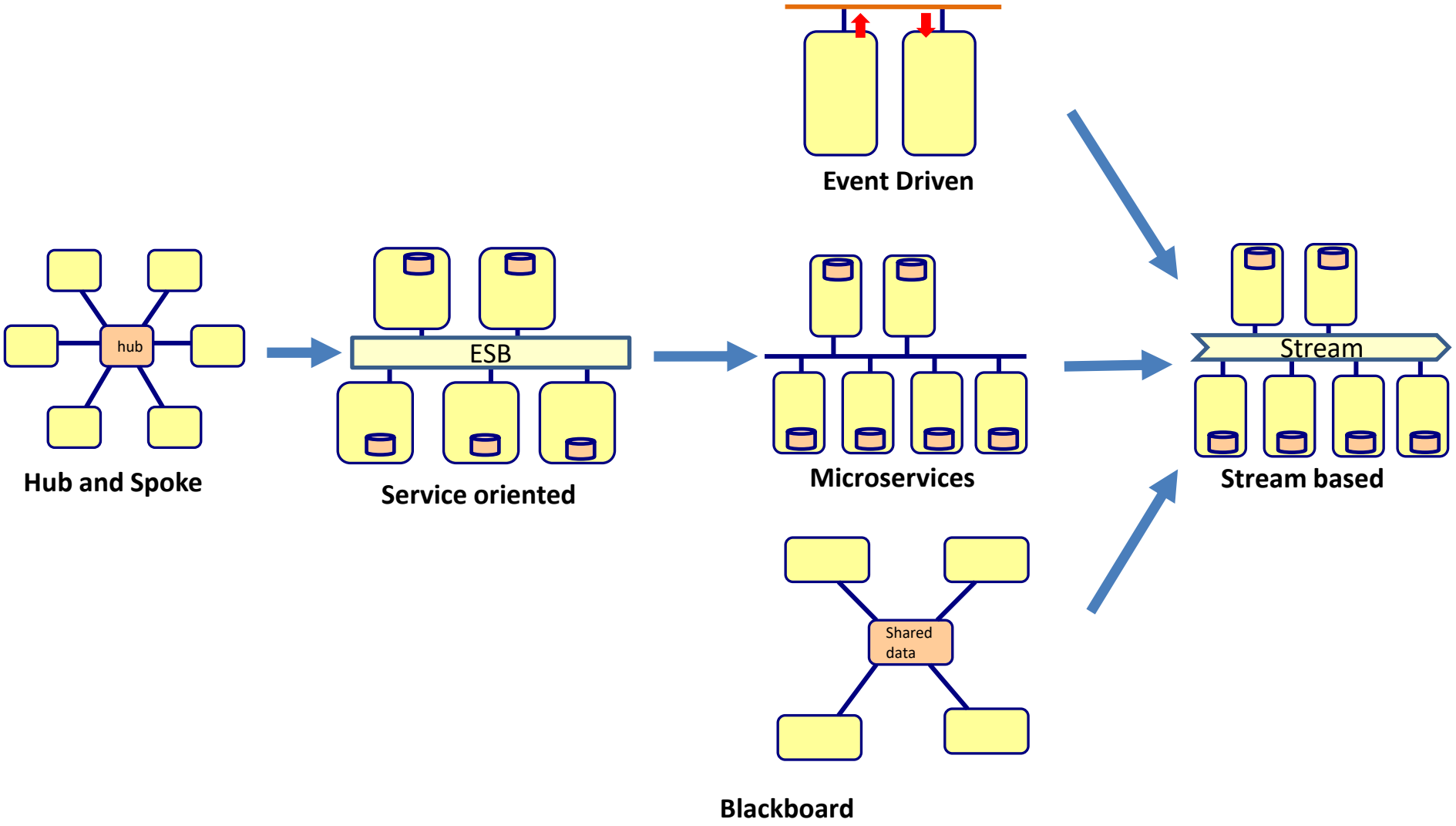
- Benefits
  - Easy to add new components
  - Components are independent of each other
  - Components can work in parallel
- Drawbacks
  - Data structure is hard to change
    - All components share the same data structure
  - Synchronization issues

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

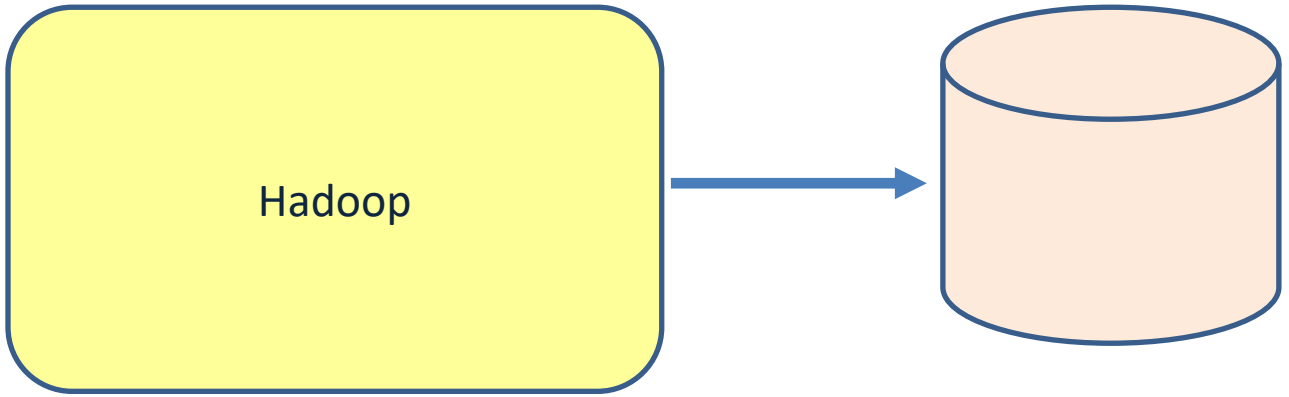
- Continuous stream of data
  - Stock market systems
  - Social networking systems
  - Internet of Things (IoT)systems
  - Systems that handle sensor data
  - System that handle logfiles
  - Systems that monitor user clicks
  - Car navigator software

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
- …

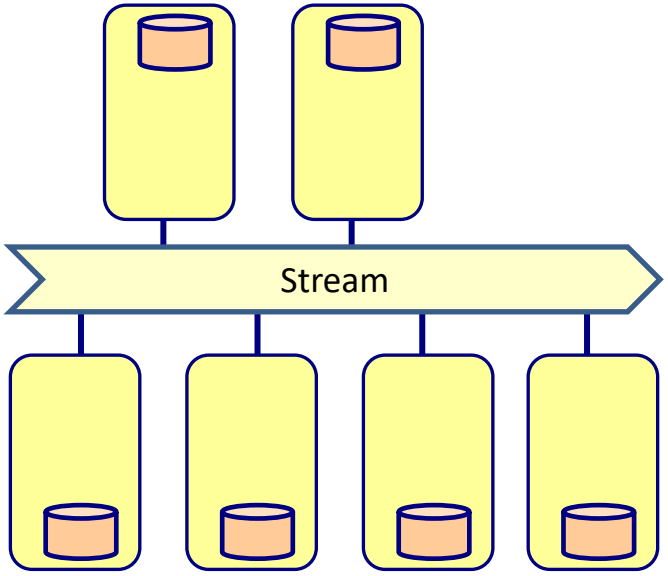
Batch processing

- 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

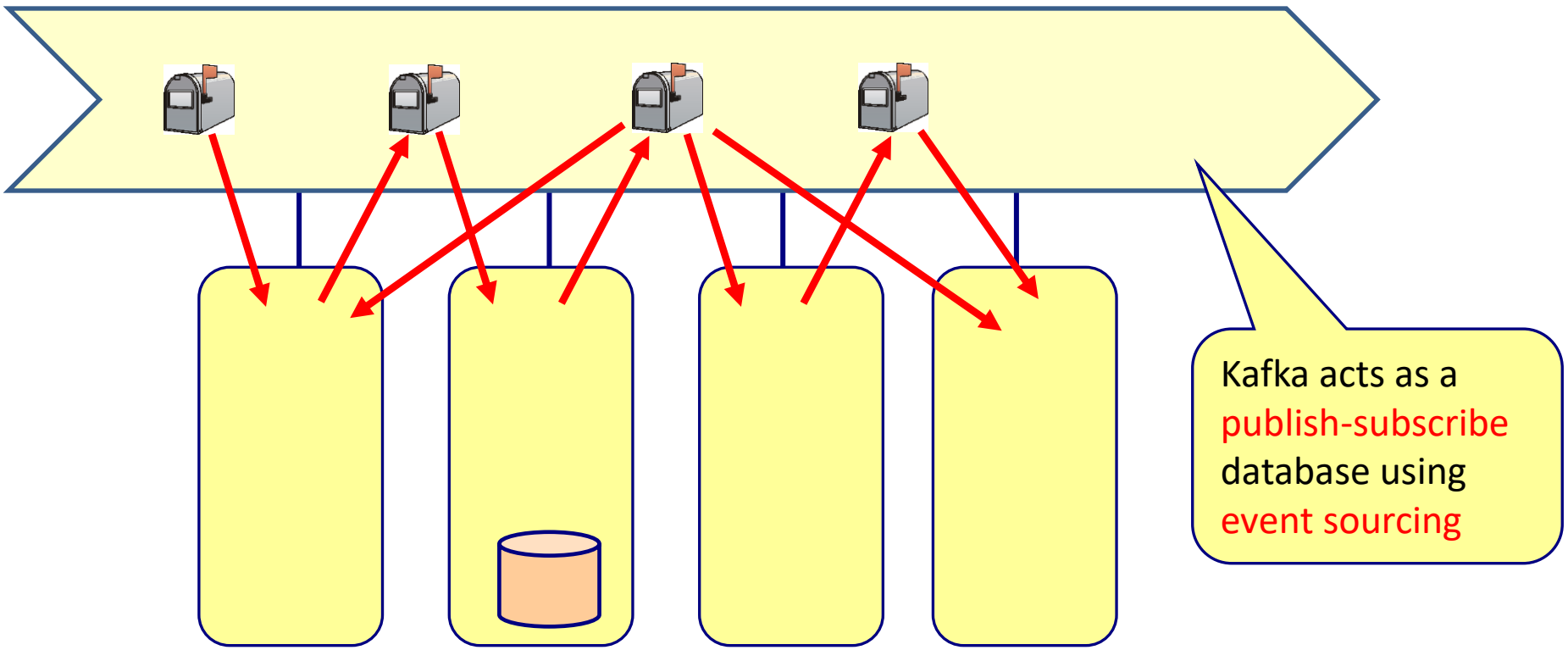


Stream processing

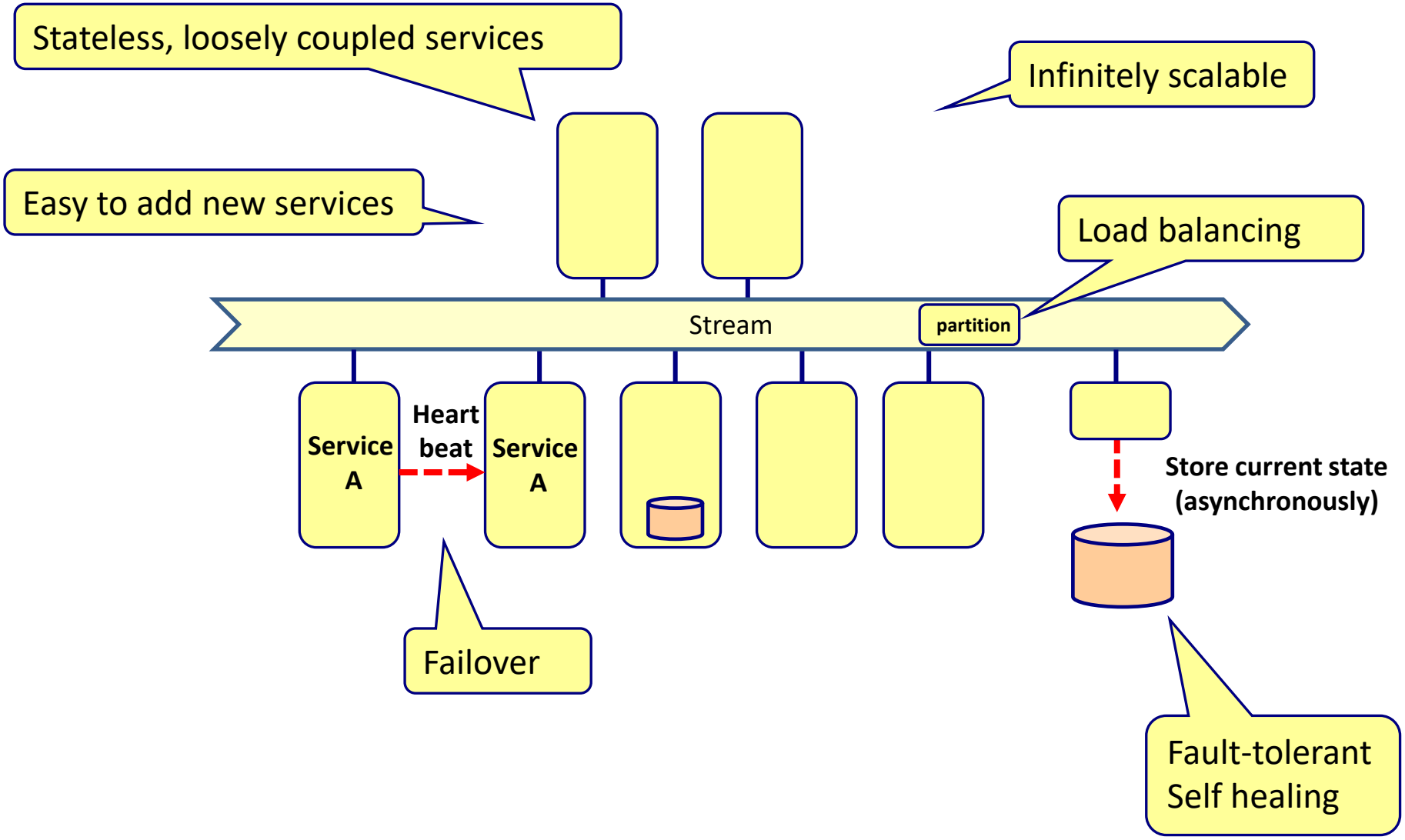
- Handle the data when it arrives
- Handle event (small data) by event
- Latencies in seconds or milliseconds



Publish-subscribe and event sourcing

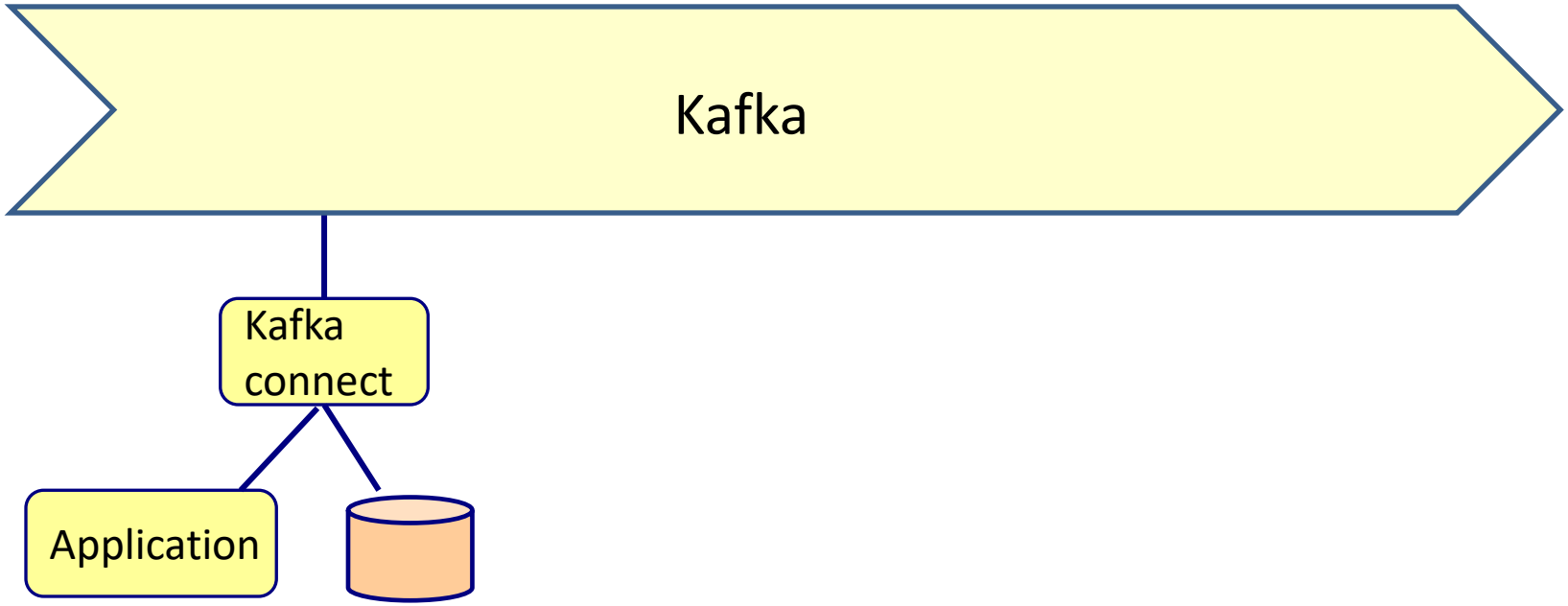


Stream based architecture

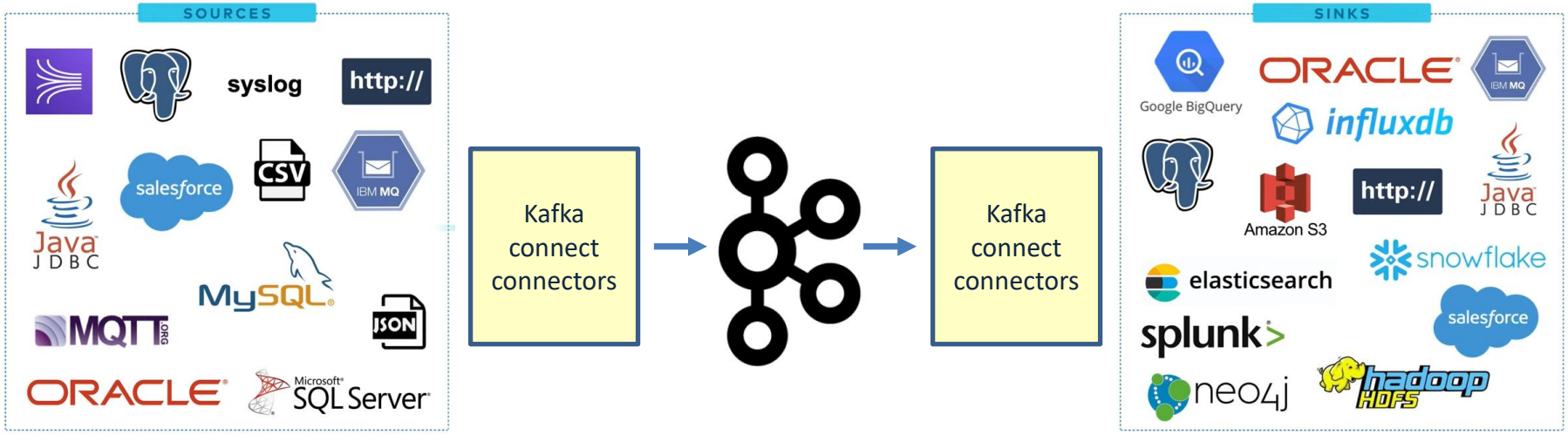


KAFKA ECOSYSTEM

Kafka ecosystem: Kafka connect

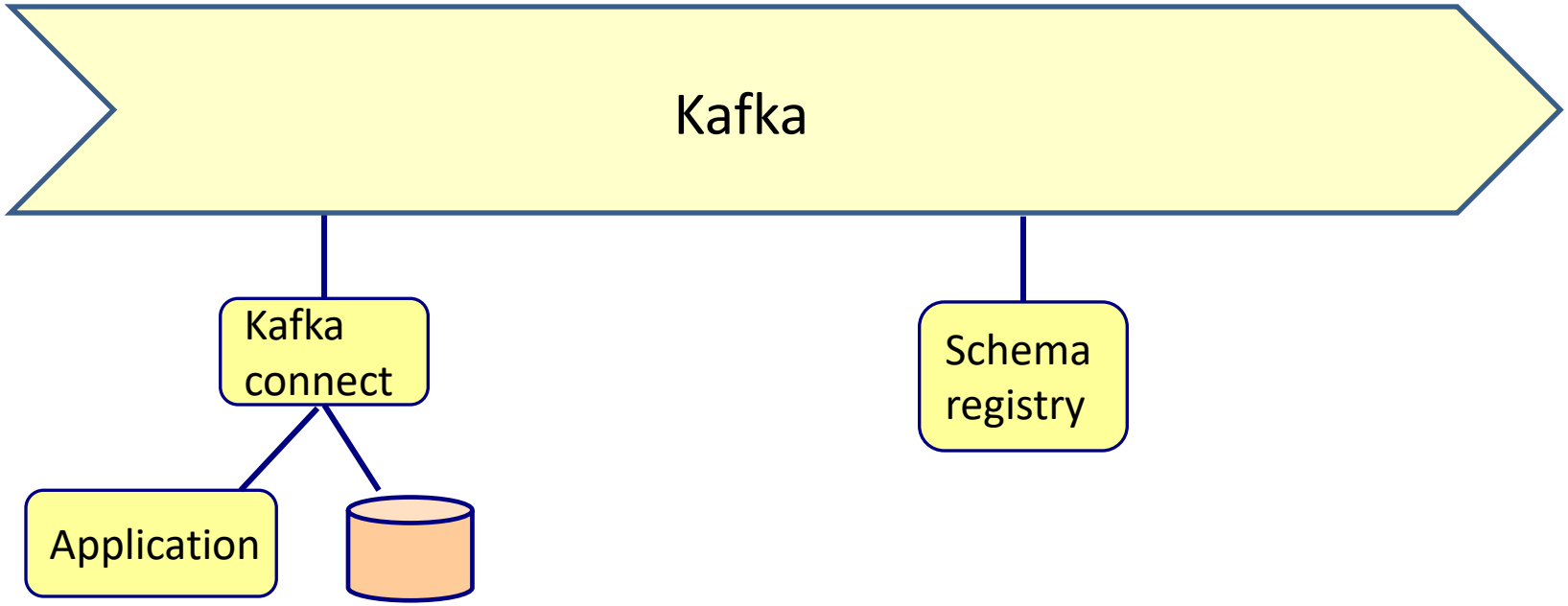


Kafka connect



Kafka

Kafka ecosystem: Schema registry



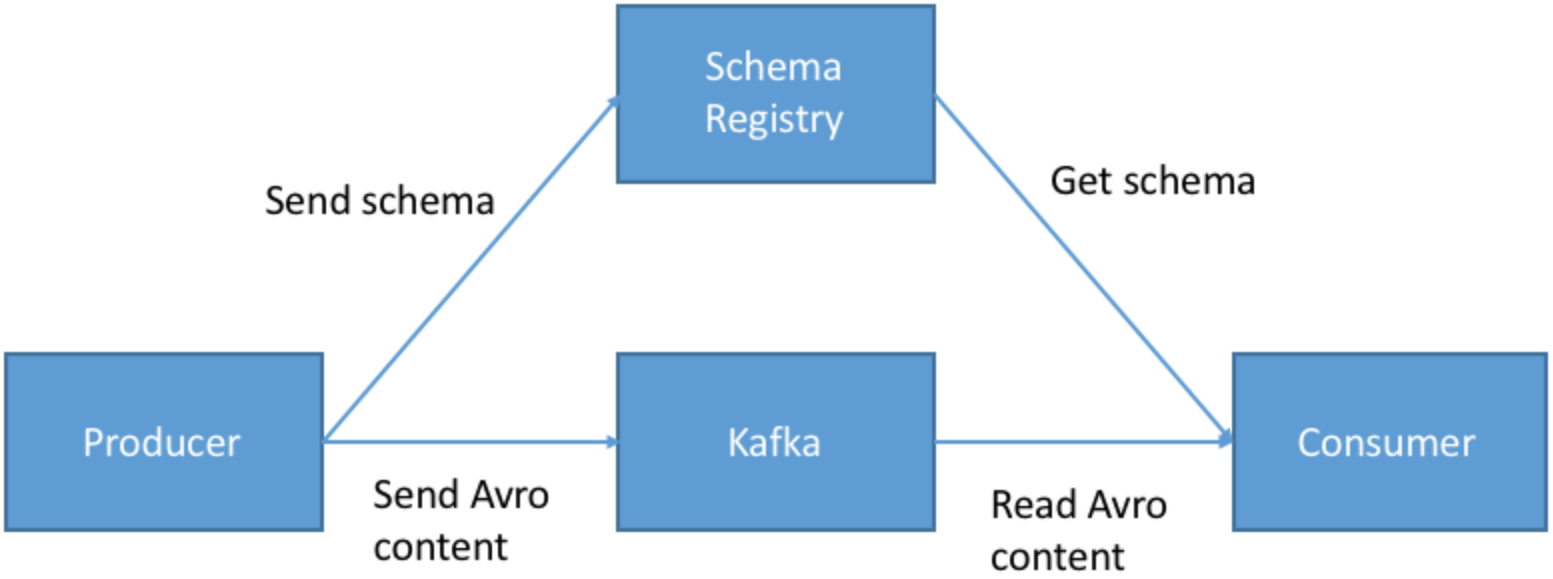
Need for a schema registry

- What if the producer sends bad data?
- What if a field gets renamed?
- What if the data format changes ?

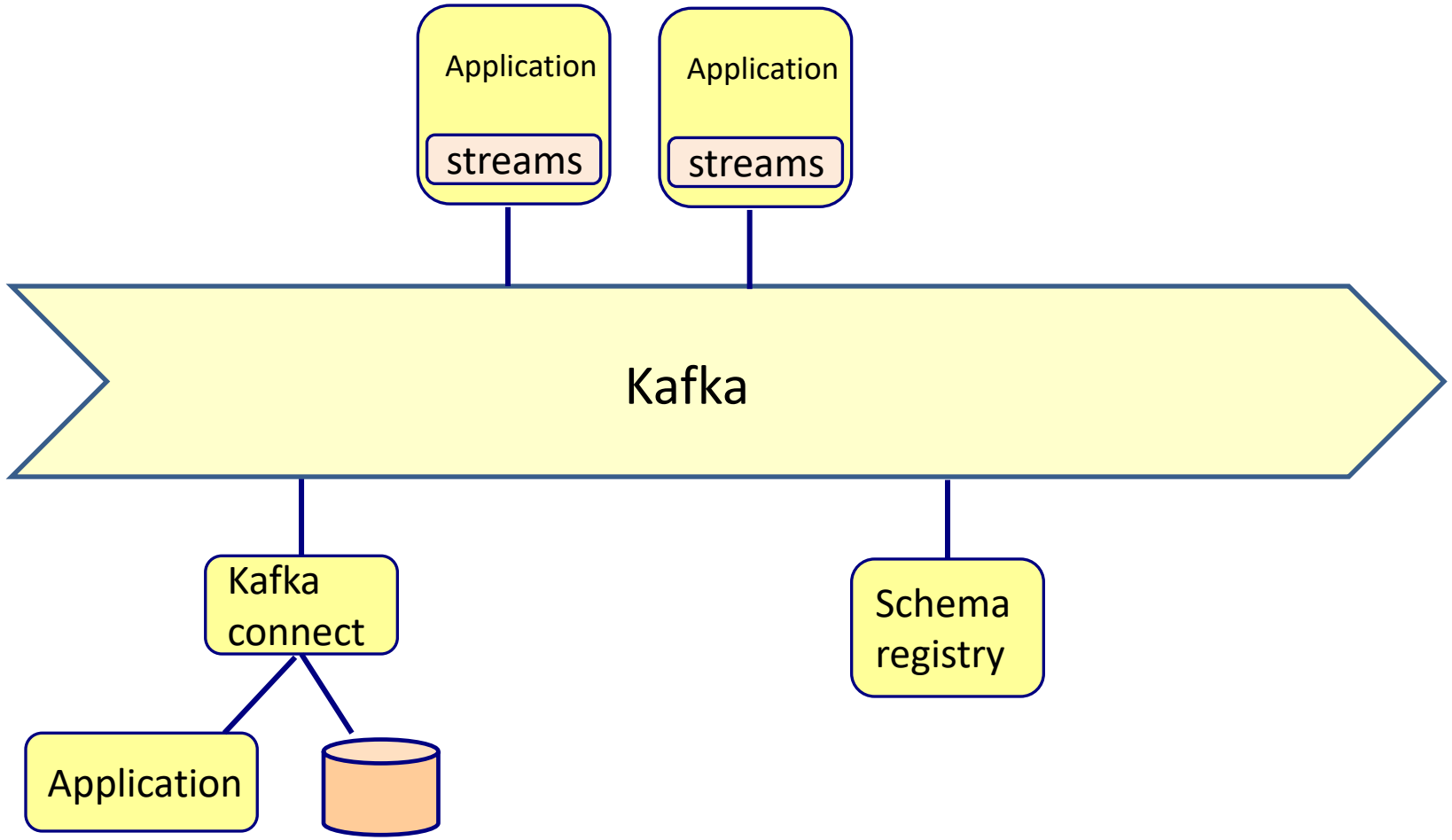
- The consumer breaks

Kafka does not verify the message

- Schema registry is a separate component (server)
- Maintains a database of schema's



Kafka ecosystem: Kafka streams



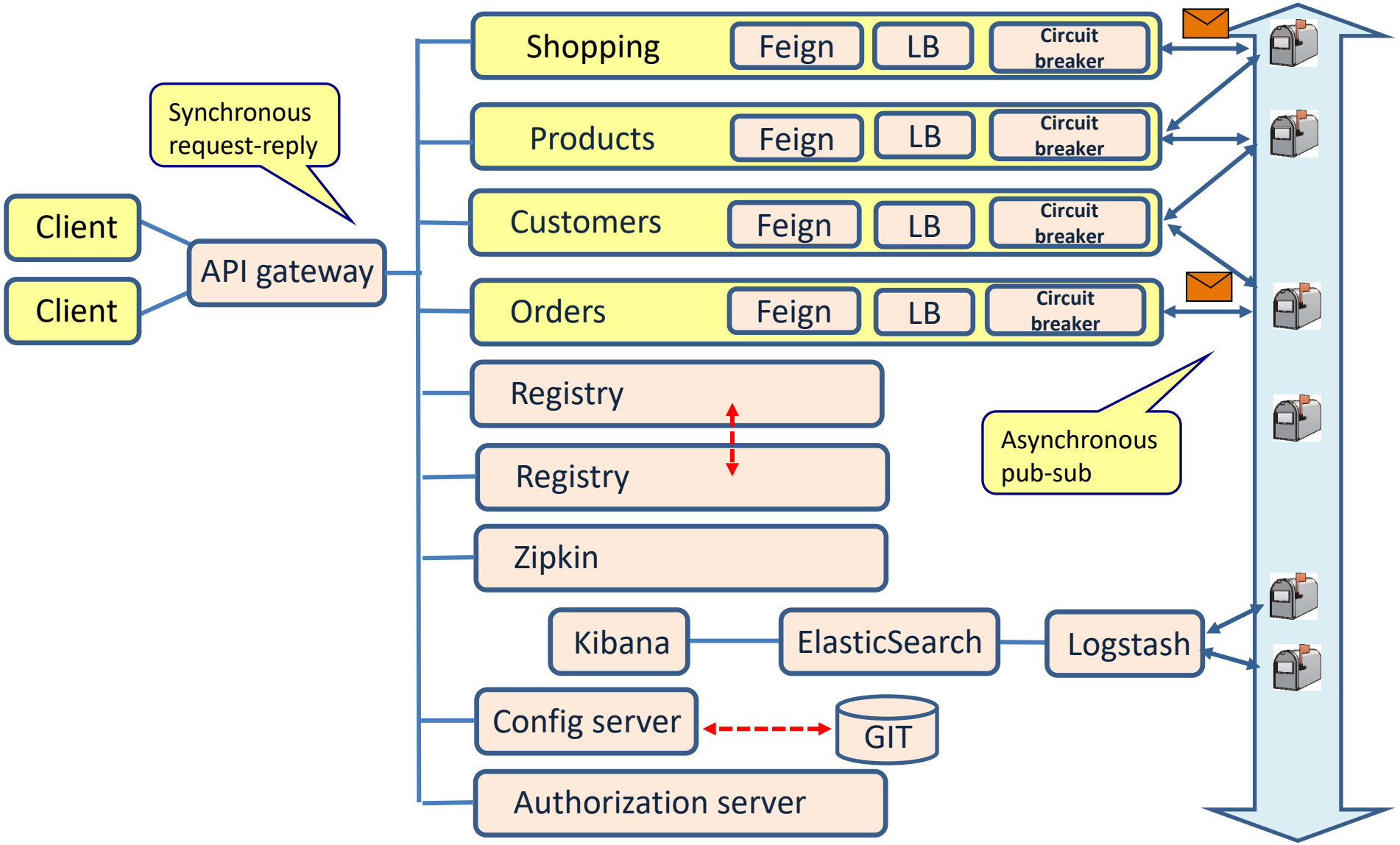
Kafka streams

- 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



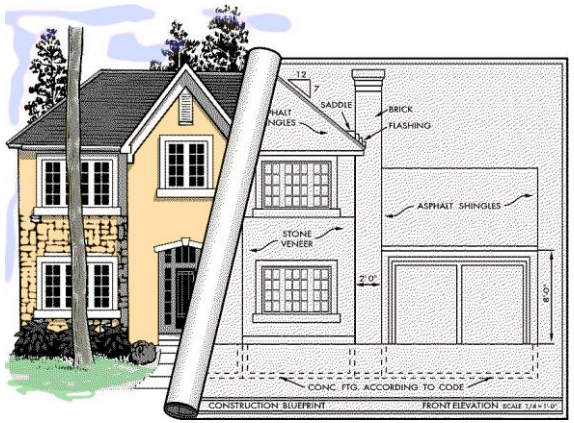
Challenges of a microservice architecture

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

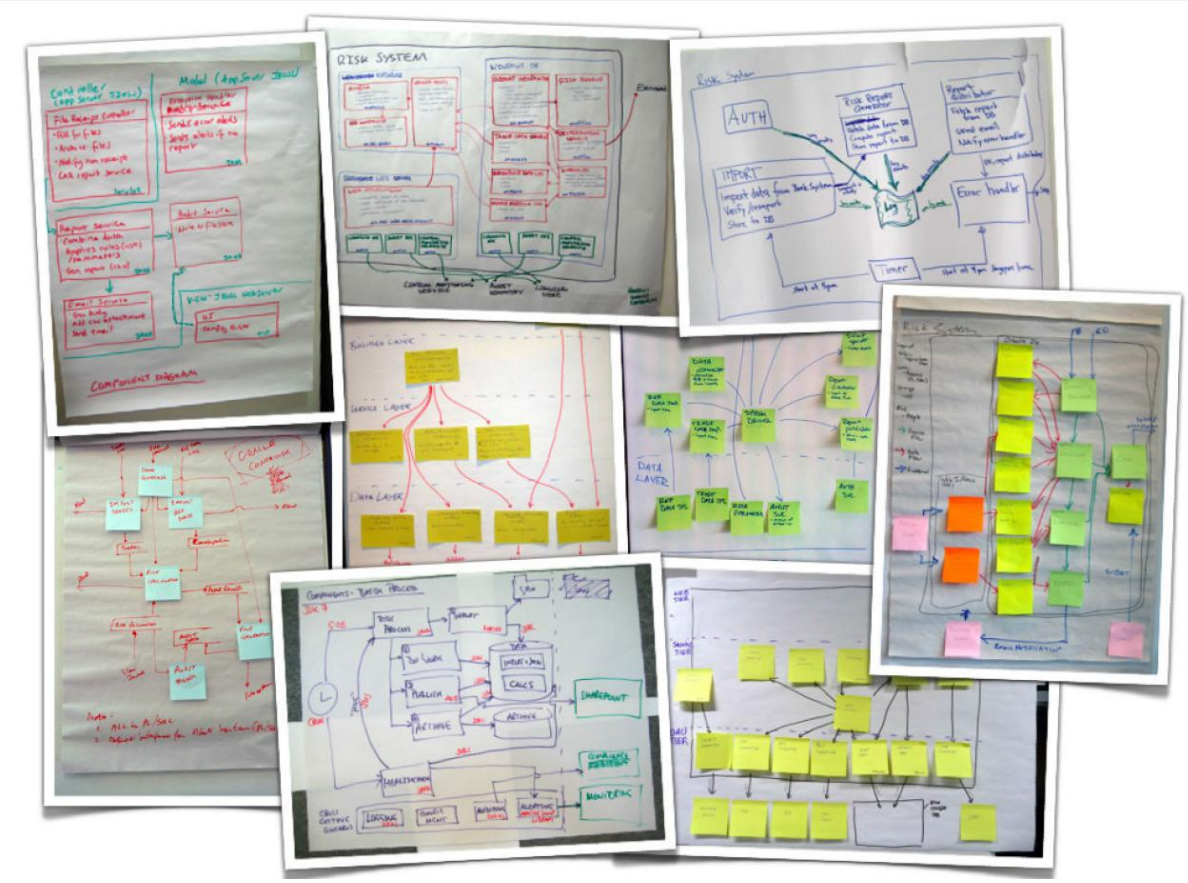
Architecture diagrams

- 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

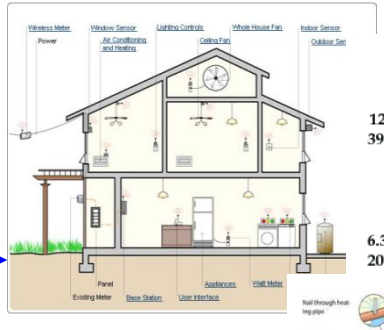
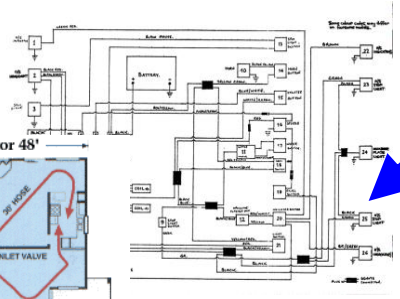
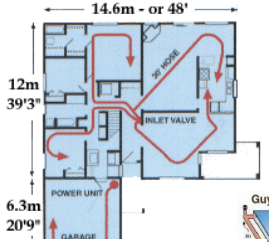
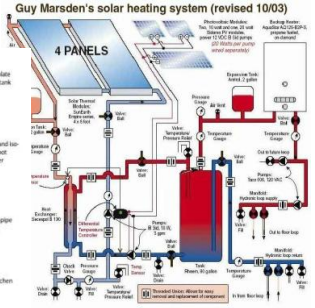
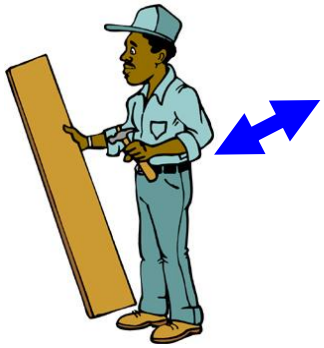
- Make the architecture visible with diagrams
  - Information radiator



Documenting architecture

- Use different views

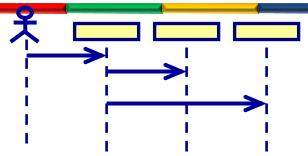


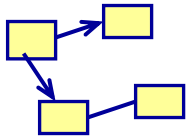

Which diagrams?

- 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

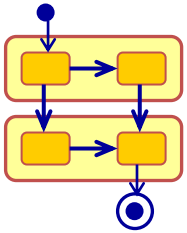
UML Diagrams



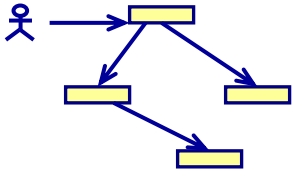
sequence diagram



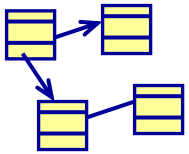
object diagram



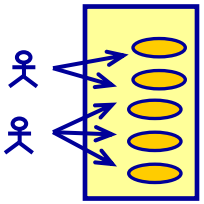
state machine diagram



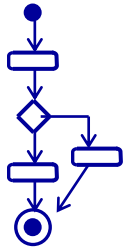
communication diagram



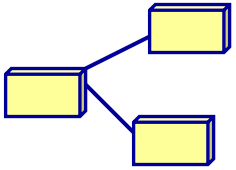
class diagram



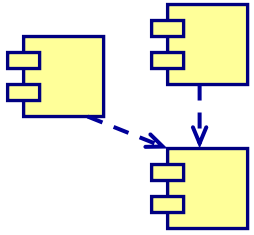
use case diagram



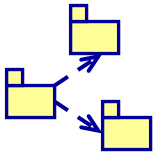
activity diagram



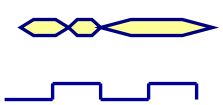
deployment diagram



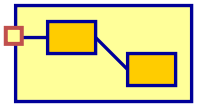
component diagram



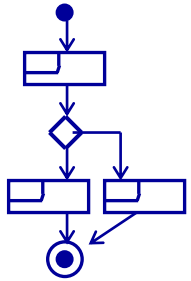
package diagram



timing diagram

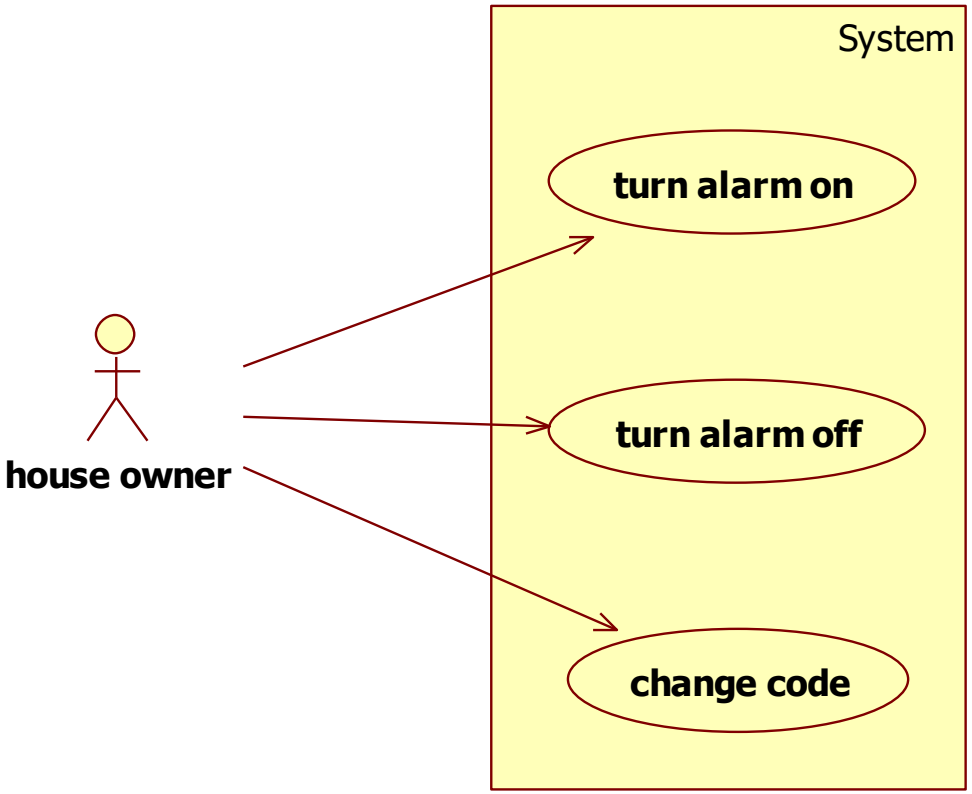


composite structure diagram

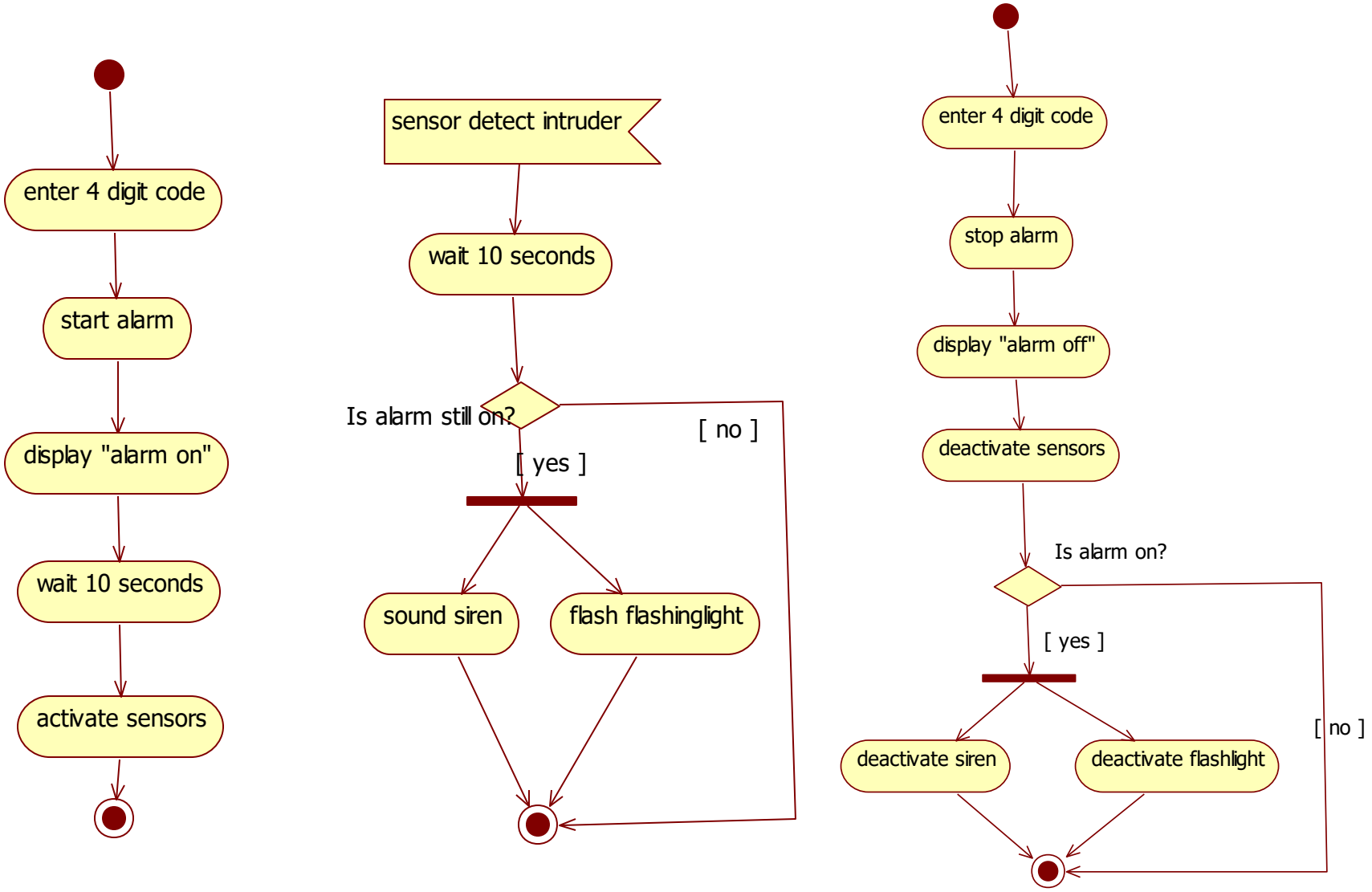


interaction overview diagram

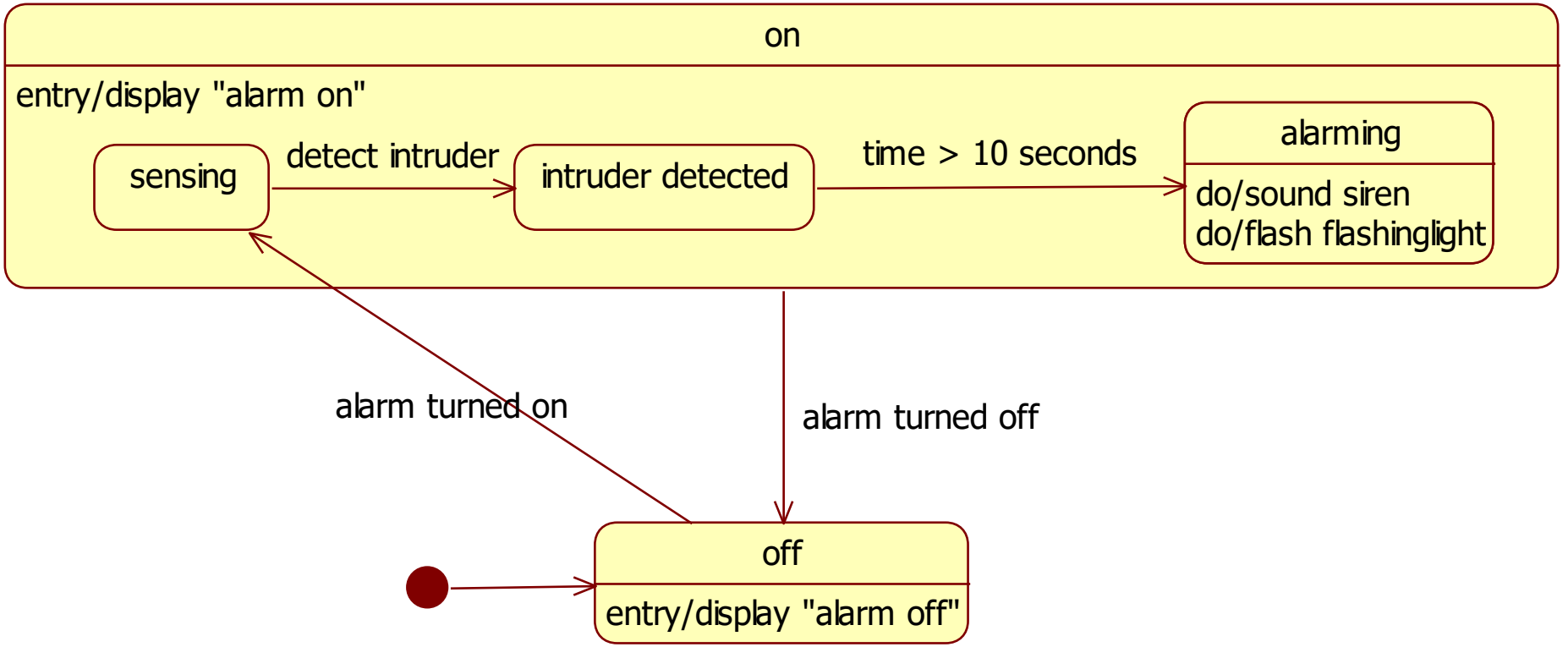
Home alarm system use case diagram



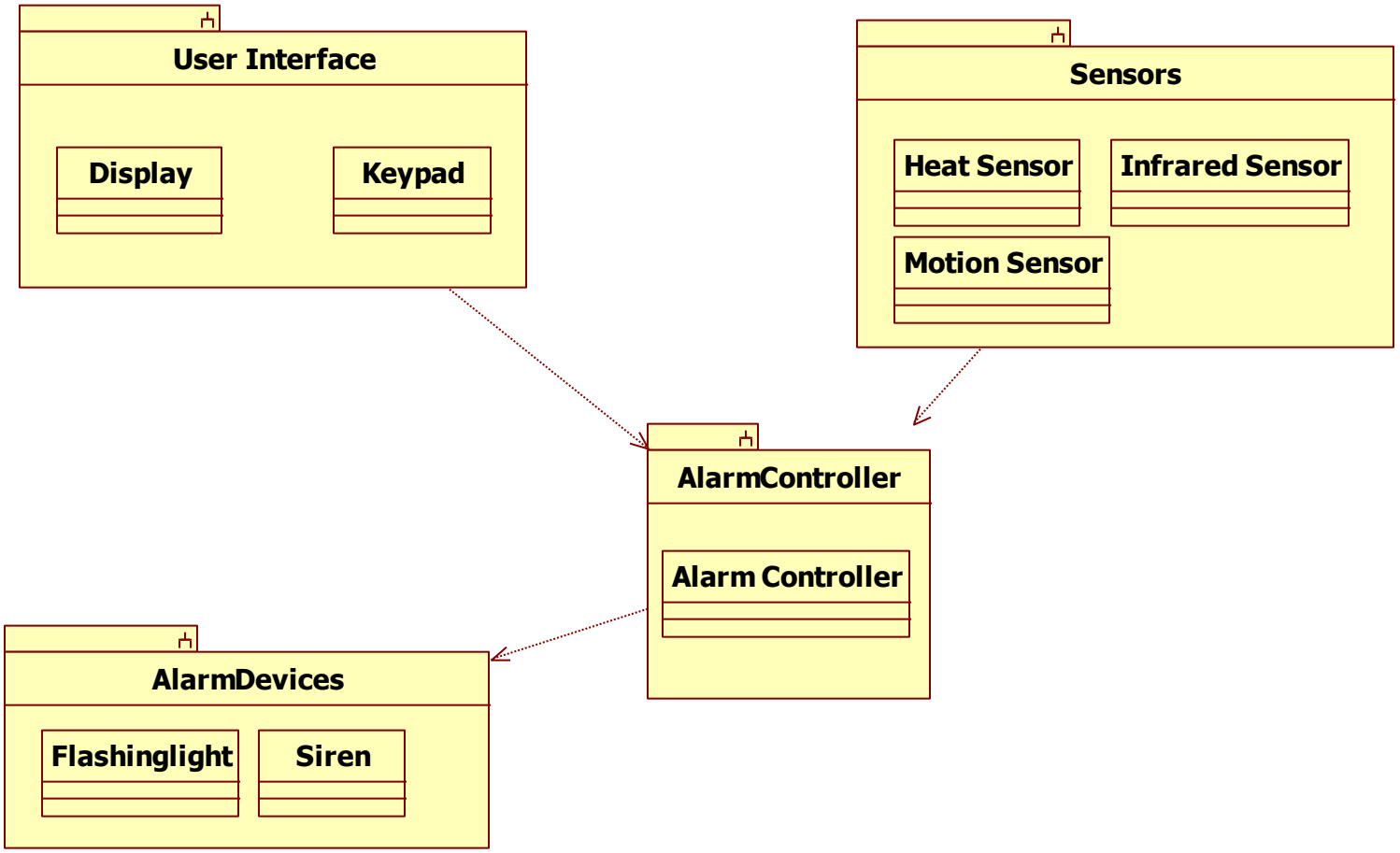
Home alarm system



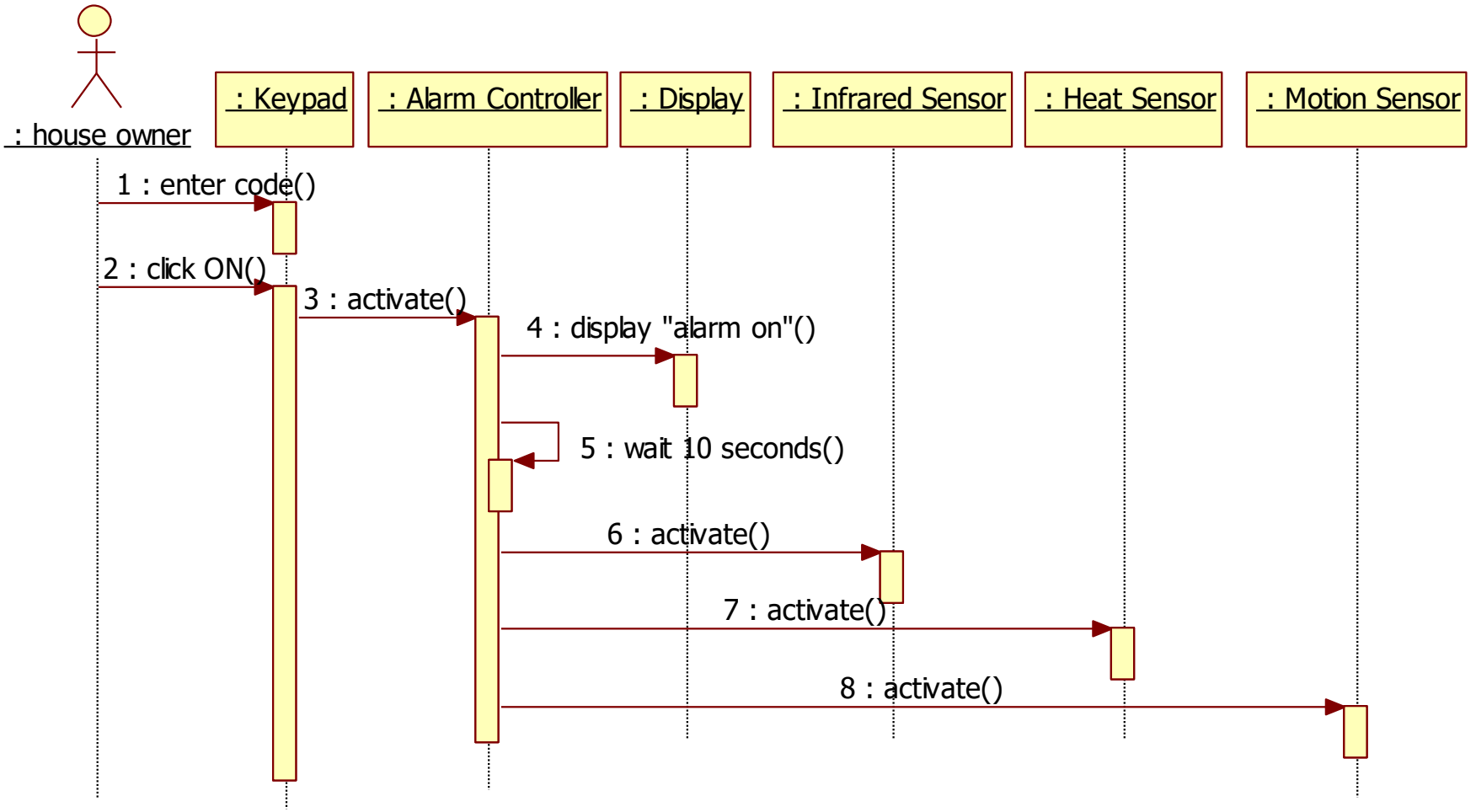
Home alarm system



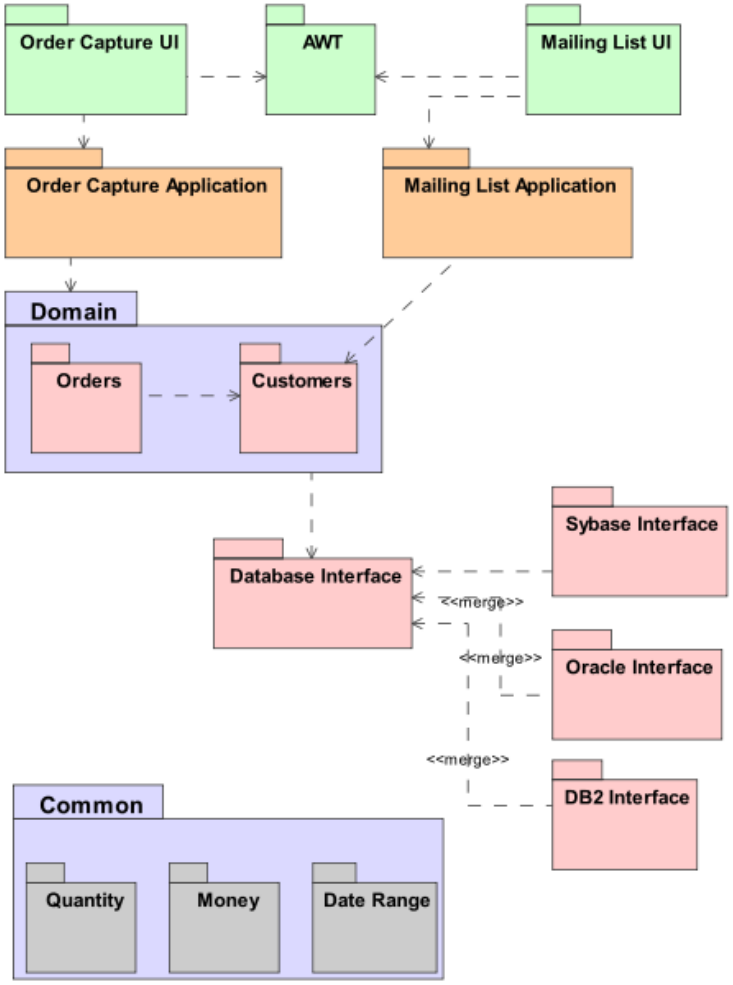
Home alarm system



Activate the alarm system

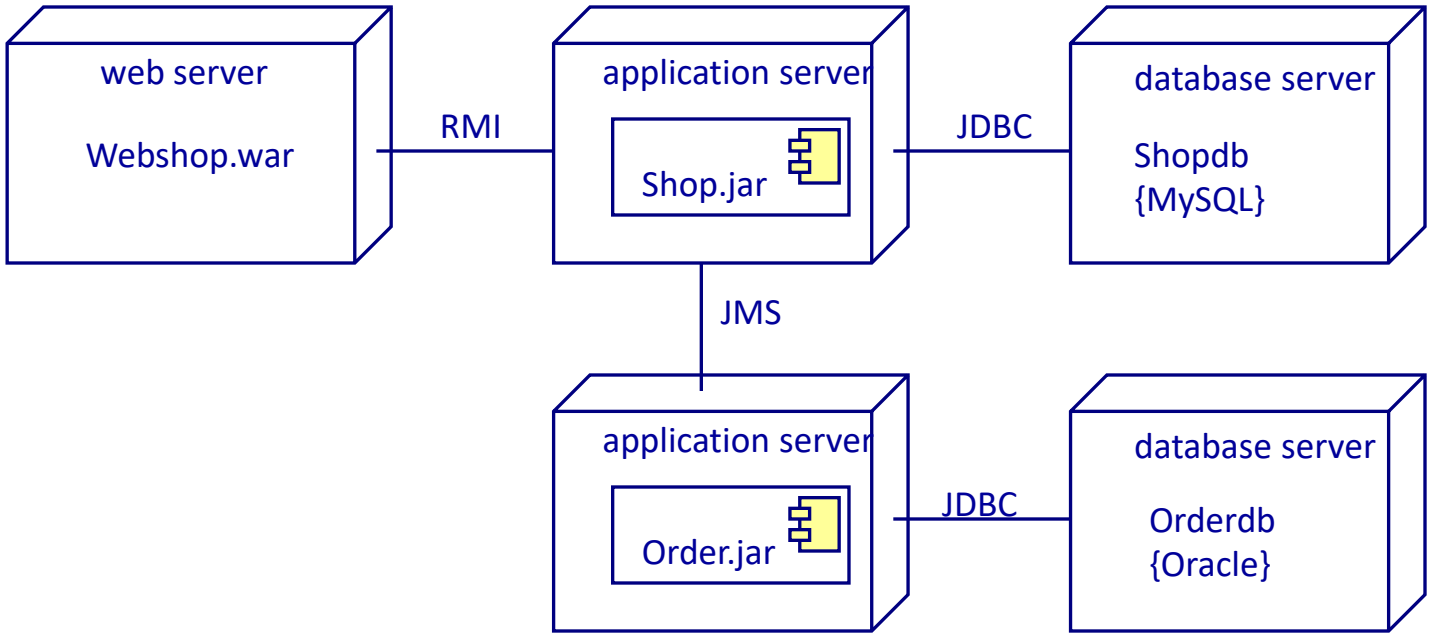


Package diagram



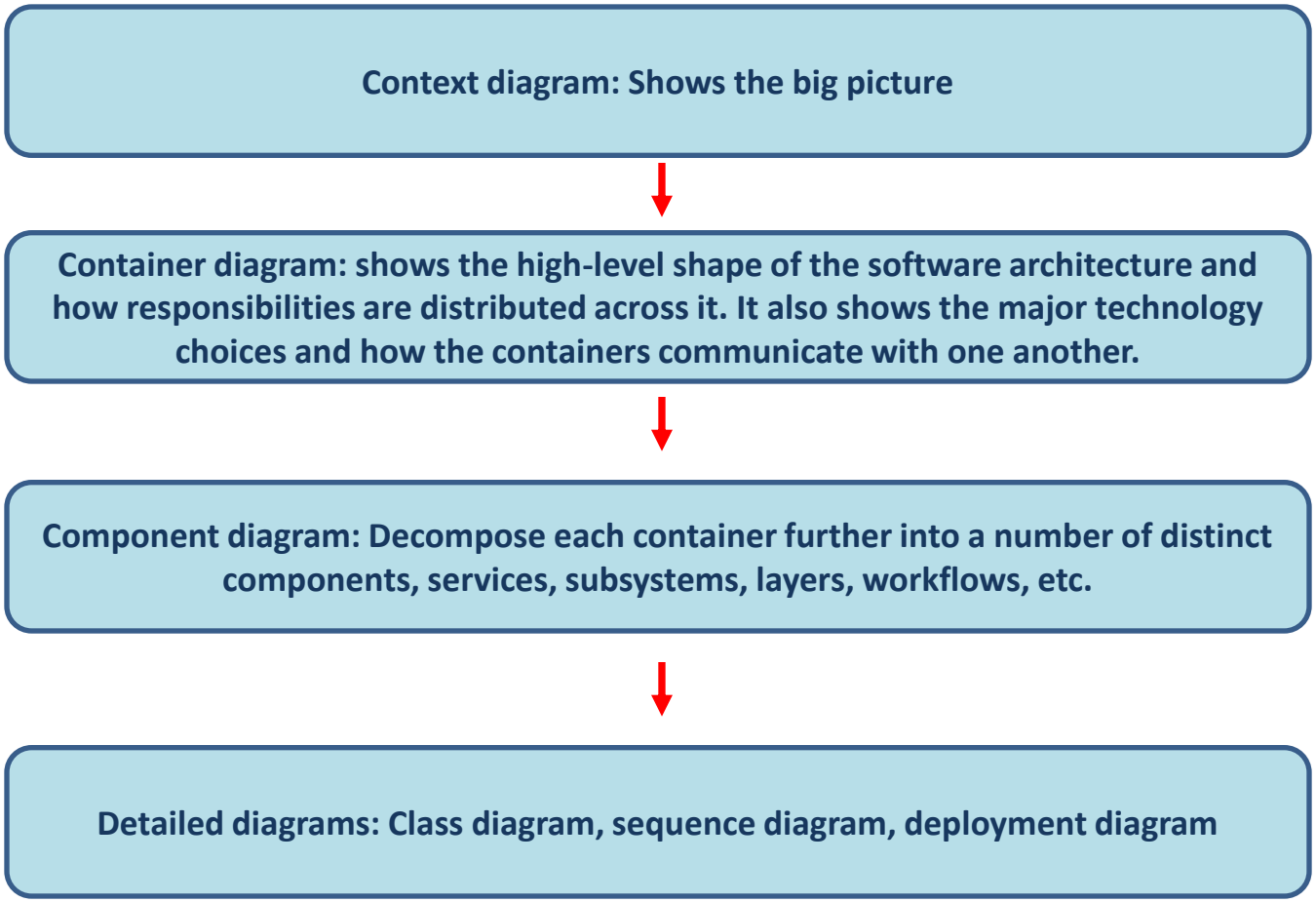
Describes how a system is split up into logical groupings and the dependencies among these groupings.

Deployment diagram



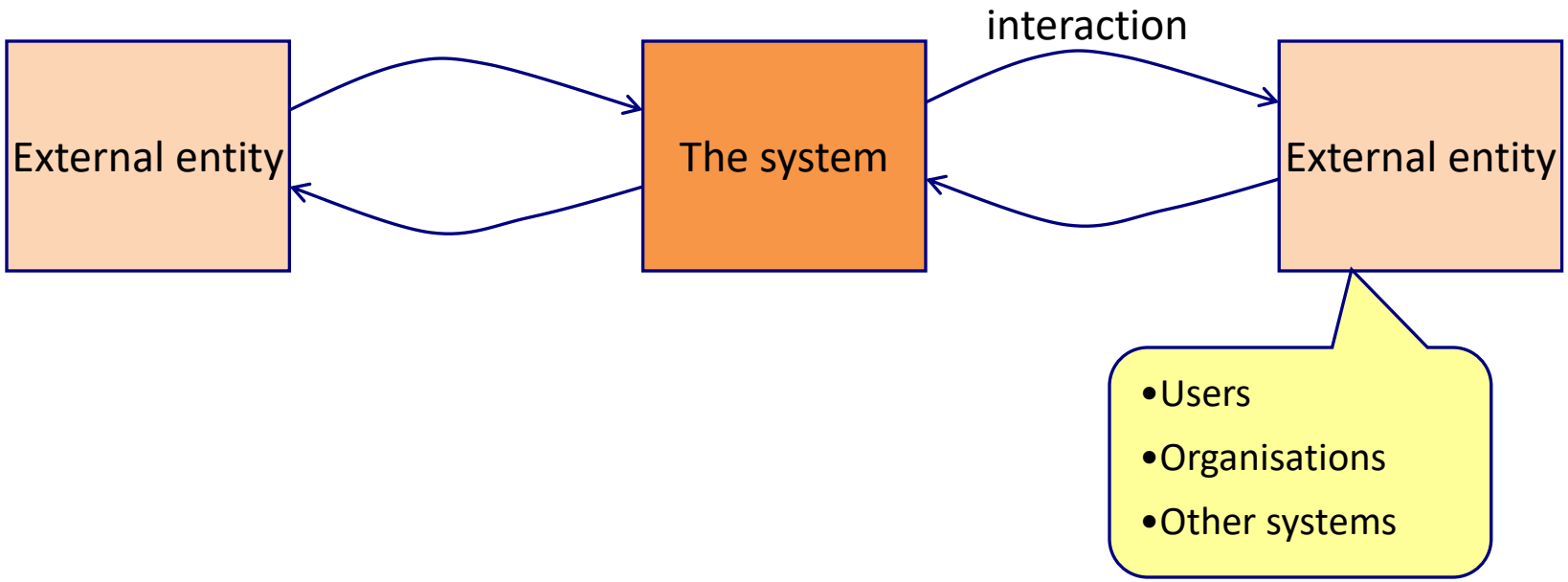
Describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.

4C model



Context diagram

- Shows the big picture
  - 10 km view



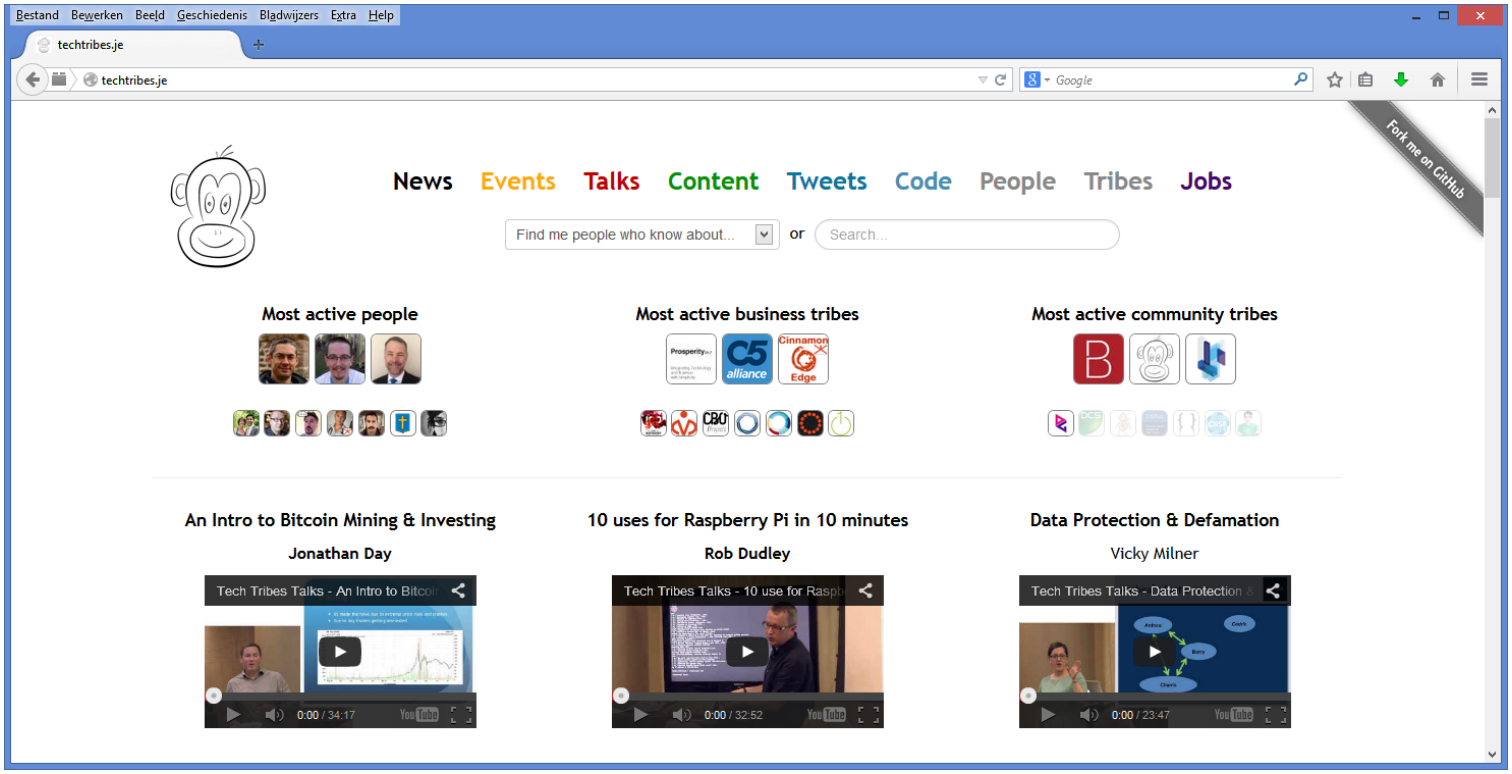
Context diagram

- 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.

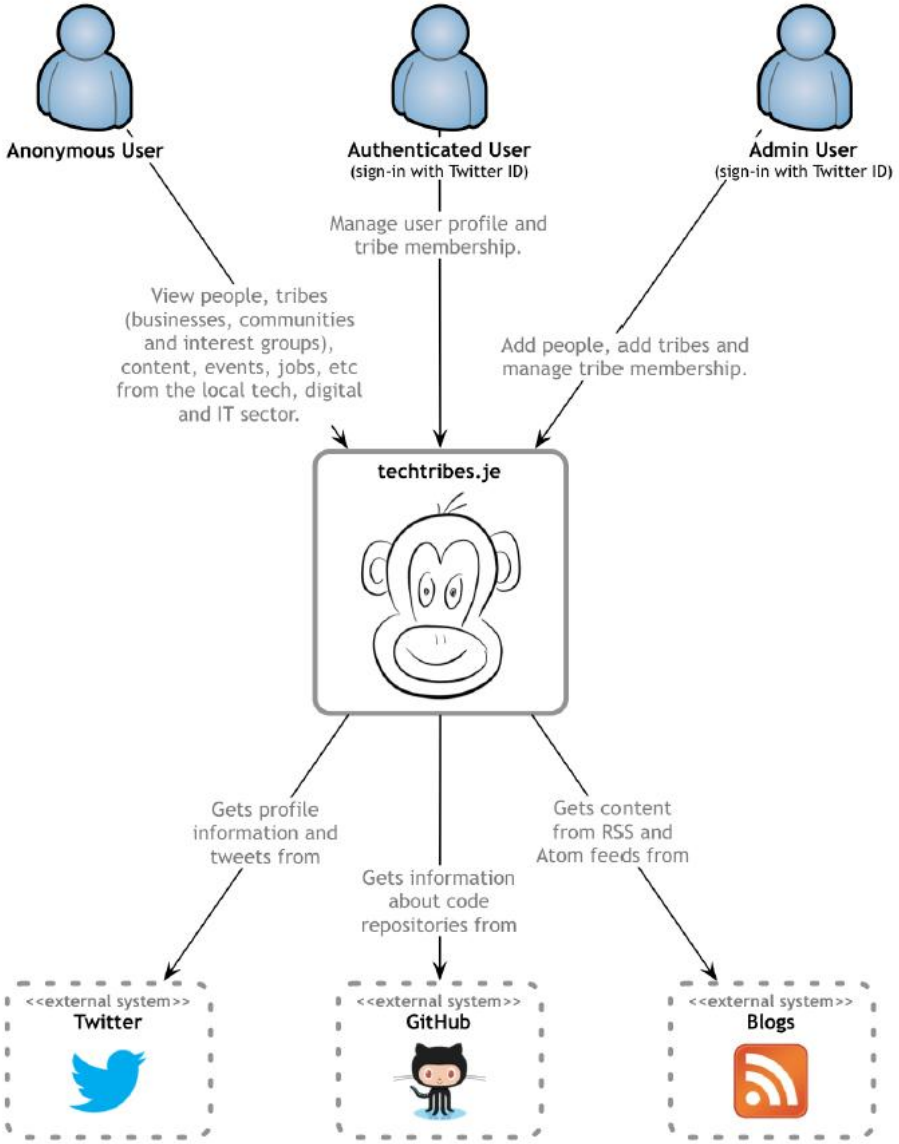
Should only take a couple of minutes to draw, so there really is no excuse not to do it

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.



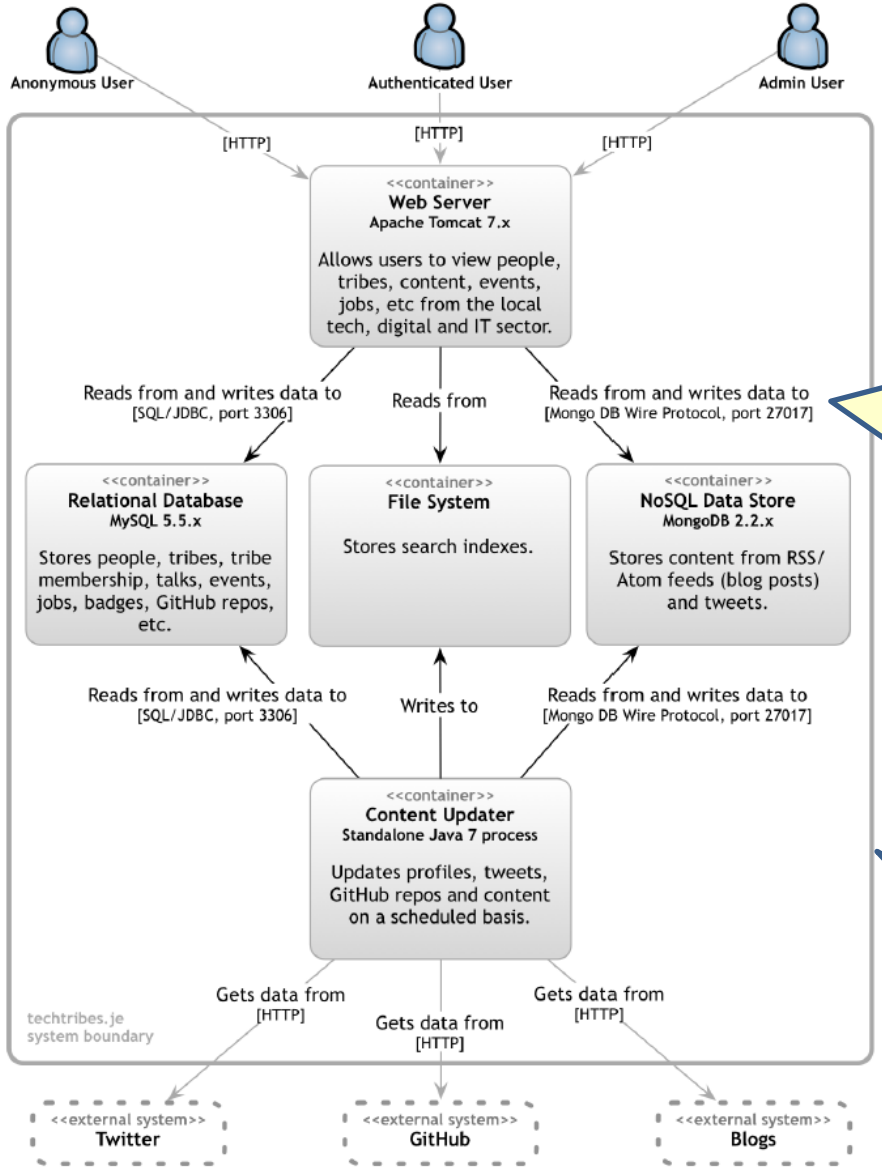
Techtribes.je context diagram



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

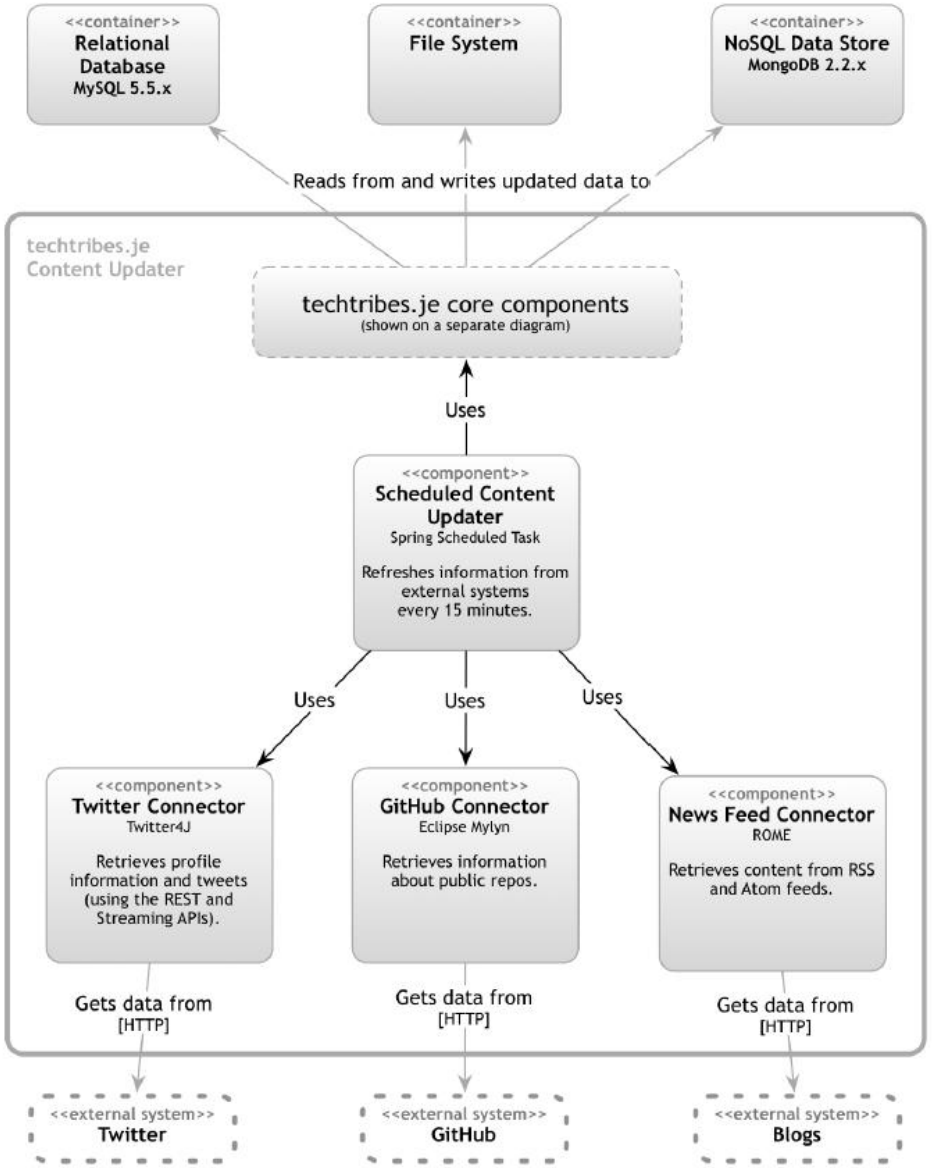
Container diagram

- 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.

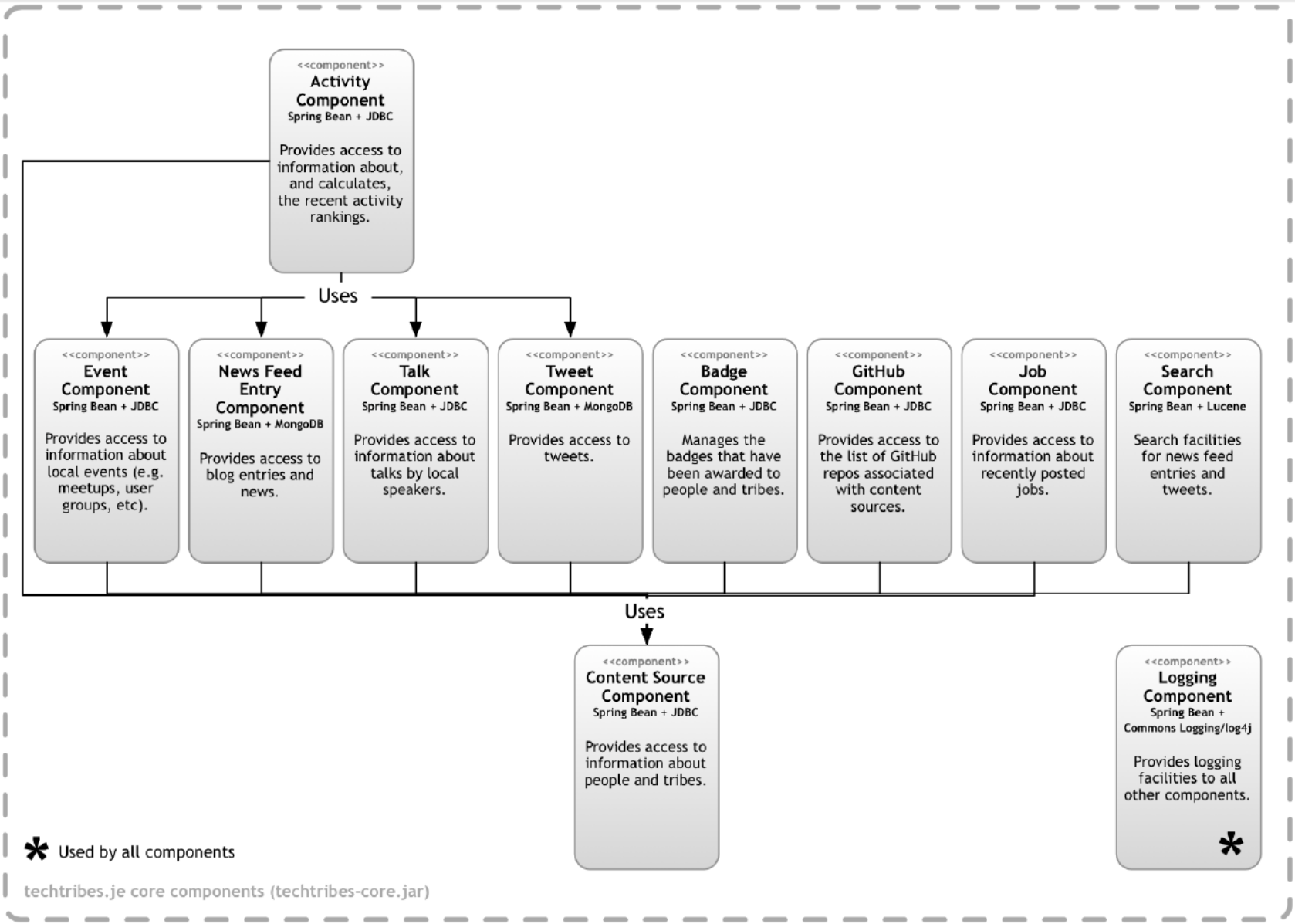
Component diagram

- 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



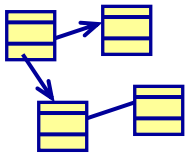
Core components



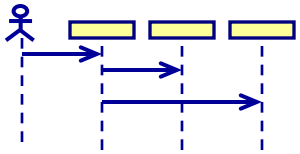
Component diagram

- 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.

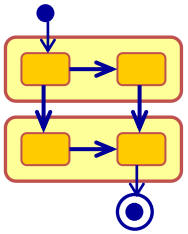
Detailed diagrams



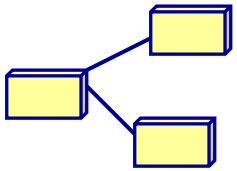
class diagram



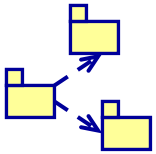
sequence diagram



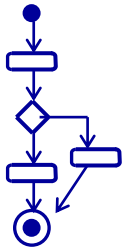
state machine diagram



deployment diagram

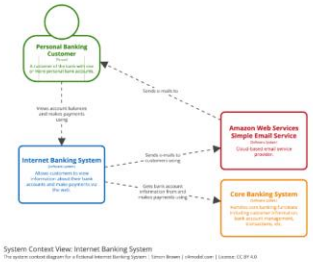


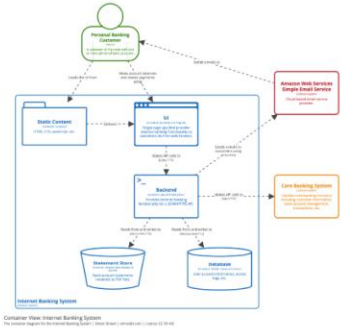
package diagram

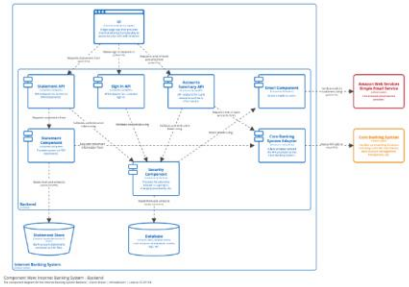


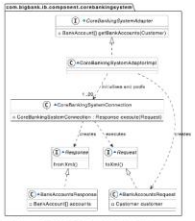
activity diagram

C4









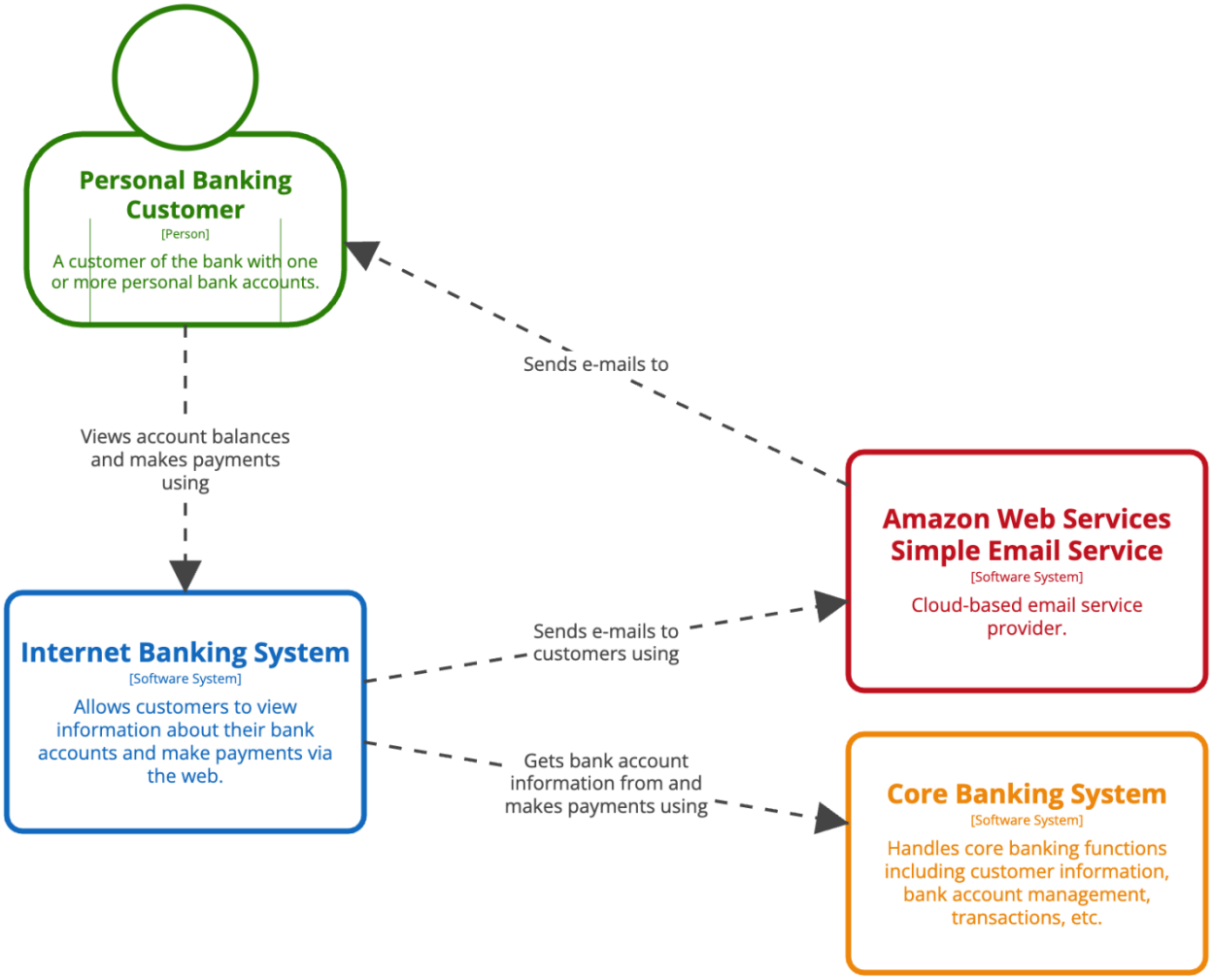
A system context diagram provides a starting point, showing how the software system in scope fits into the world around it.

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.

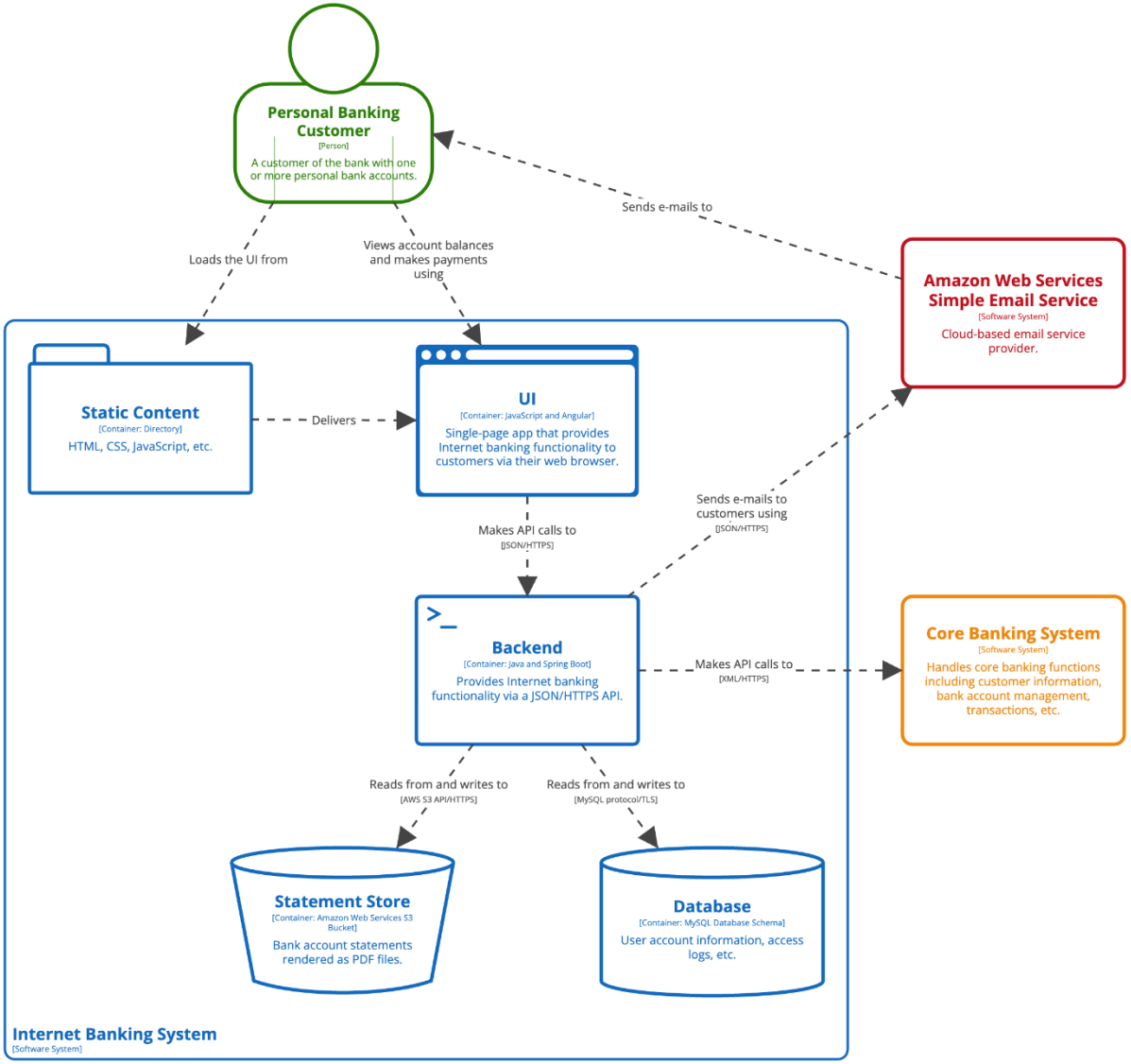
Context diagram



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

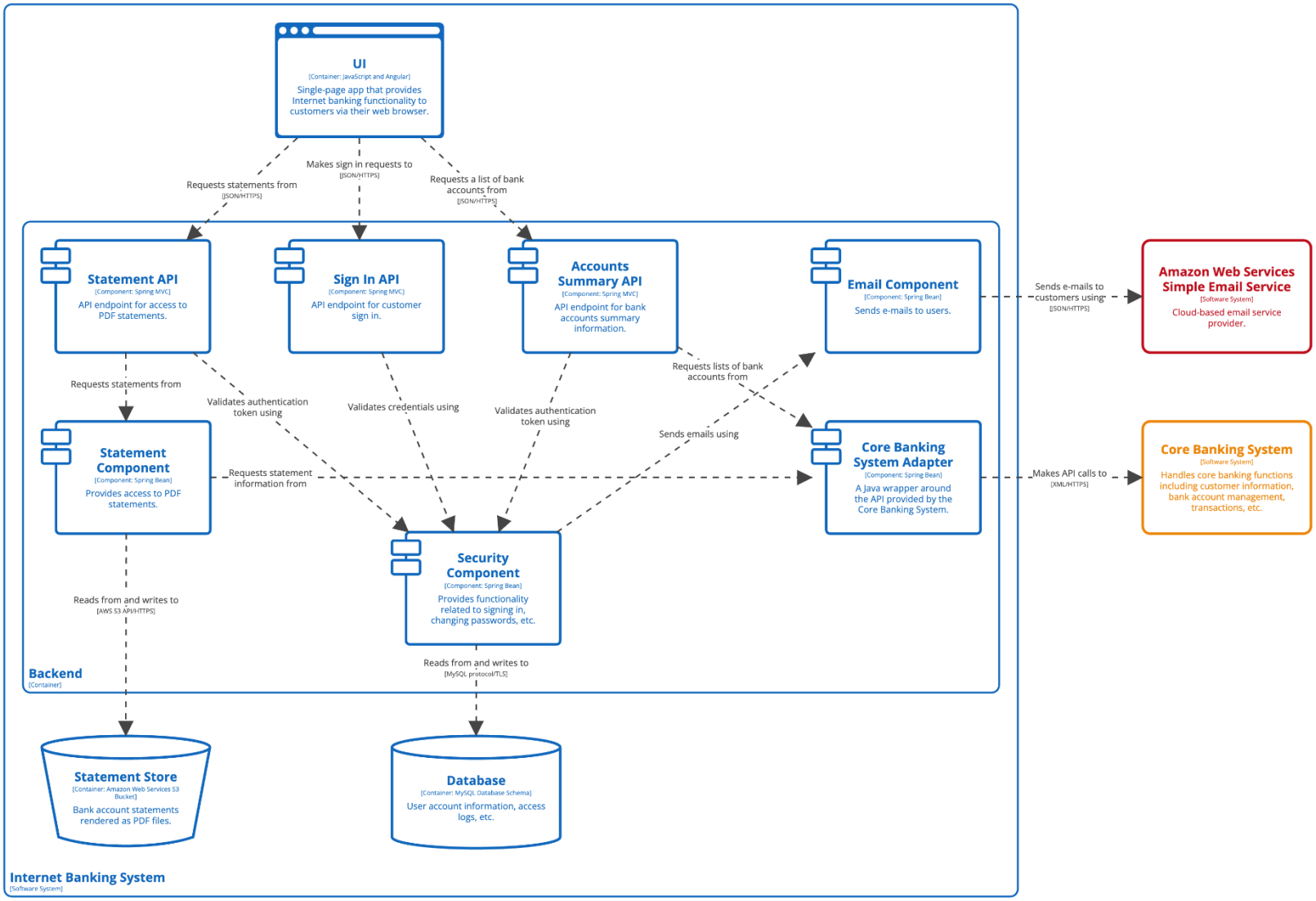
Container diagram



Container View: Internet Banking System

The container diagram for the Internet Banking System | Simon Brown | c4model.com | License: CC BY 4.0

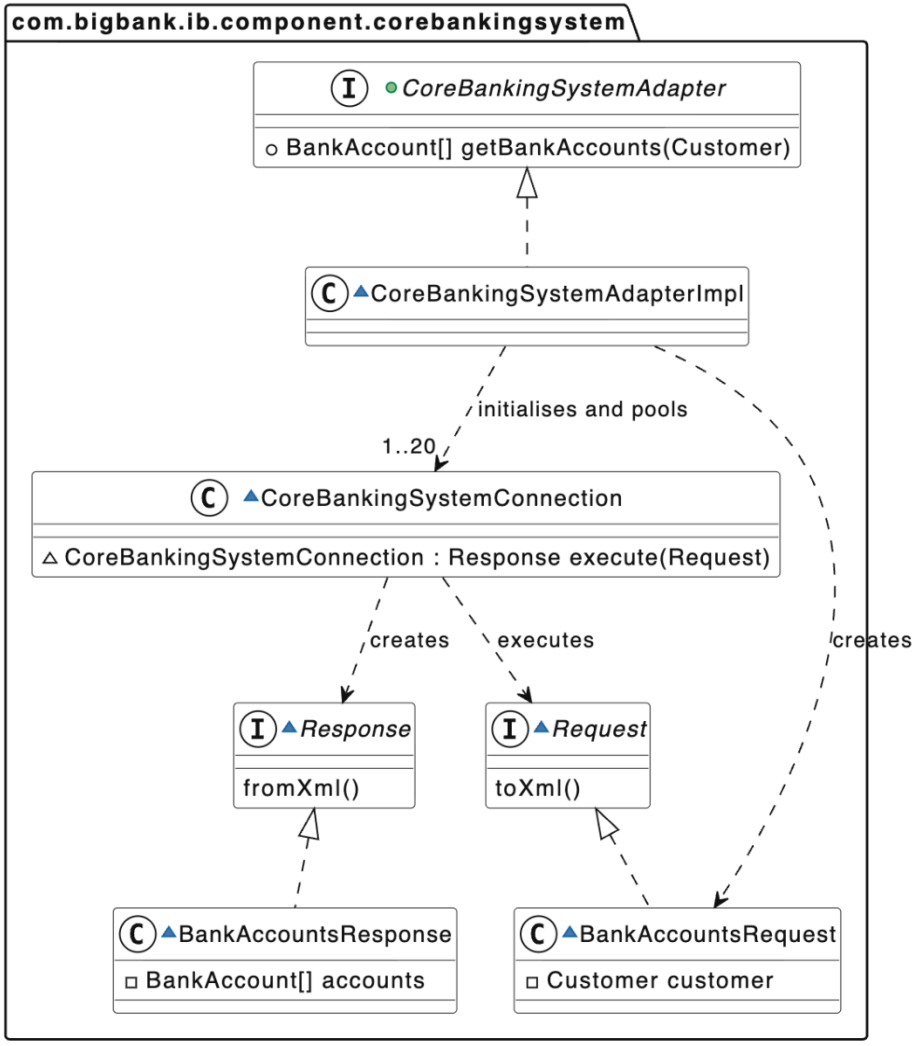
Component diagram



Component View: Internet Banking System - Backend

The component diagram for the Internet Banking System Backend | Simon Brown | c4model.com | License: CC BY 4.0

Code diagram (Class diagram)



Code View: Internet Banking System - Backend - Core Banking System Adapter

A summary of the implementation details for the Core Banking System Adapter component | Simon Brown | c4model.com | License: CC BY 4.0

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.