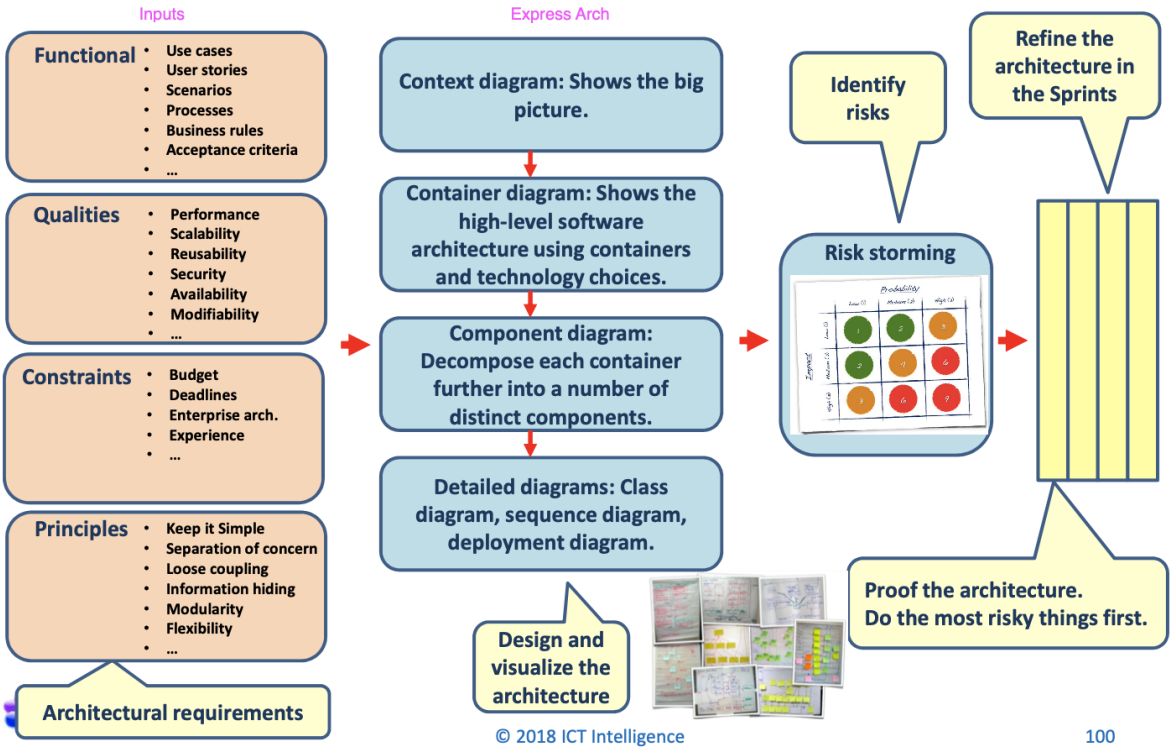
LESSON 1

# Agile architecture



# SEI quality model

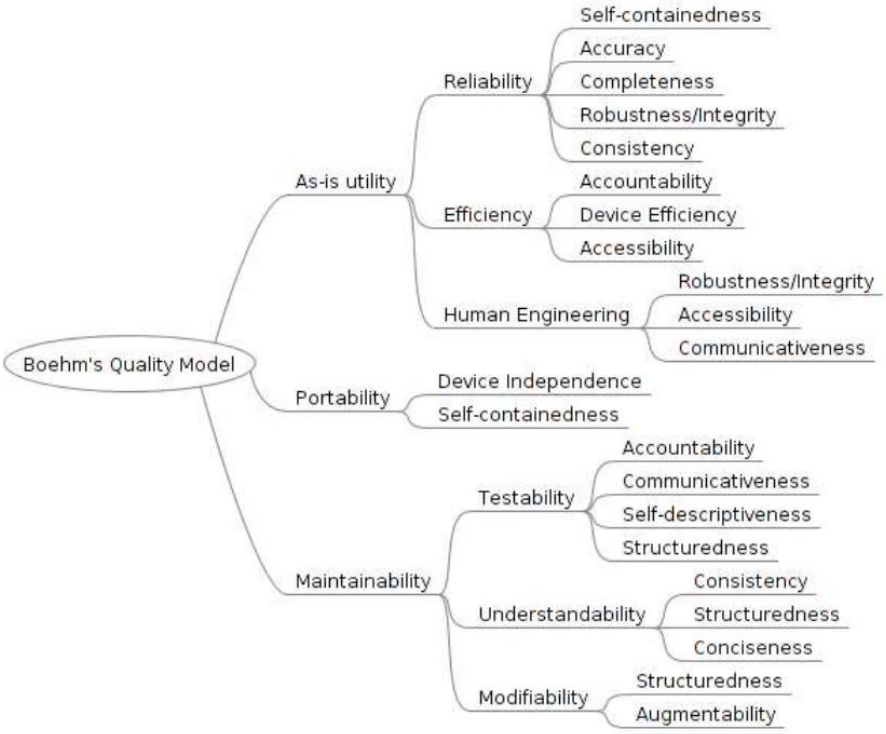
| | | |
|---|---|---|
| Qualities noticeable at runtime | Performance | Responsiveness of the system |
| | Security | Ability to resist unauthorized usage |
| | Availability | Portion of time the system is available |
| | Functionality | Ability to do intended work |
| | Usability | Learnability, efficiency, 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 |

from Rational Unified process

# 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


Boehm's Quality Model
As-is utility
Reliability
Self-containedness
Accuracy
Completeness
Robustness/Integrity
Consistency
Efficiency
Accountability
Device Efficiency
Accessibility
Human Engineering
Robustness/Integrity
Accessibility
Communicativeness
Portability
Device Independence
Self-containedness
Maintainability
Testability
Accountability
Communicativeness
Self-descriptiveness
Structuredness
Understandability
Consistency
Structuredness
Conciseness
Modifiability
Structuredness
Augmentability

# ISO 25010 8 main

# Most important qualities

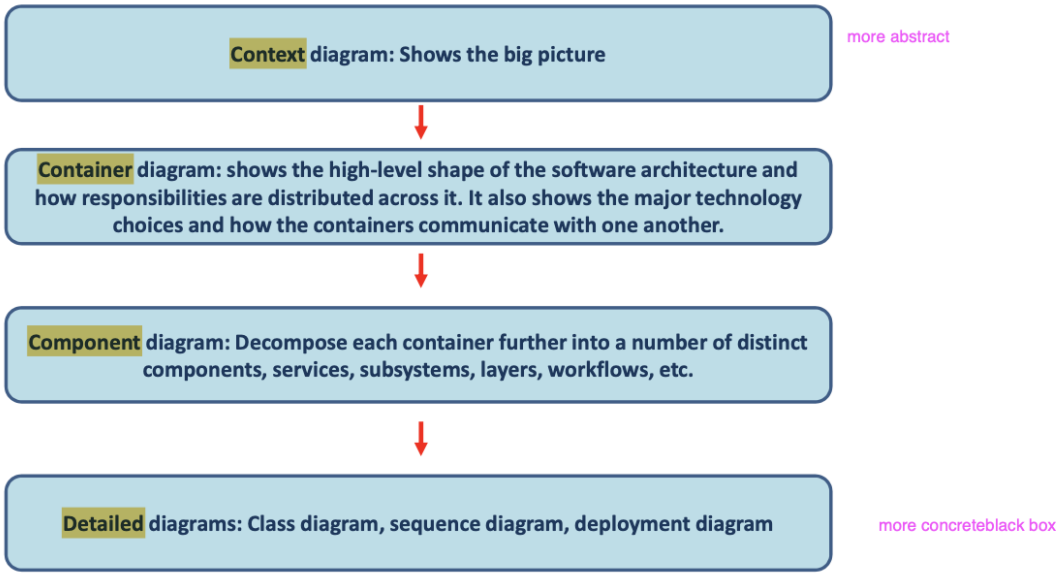
- Performance
- Scalability
- Availability
- Reliability
- Maintainability
- Security
- Interoperability
- Usability

1 Architecture (design) principles

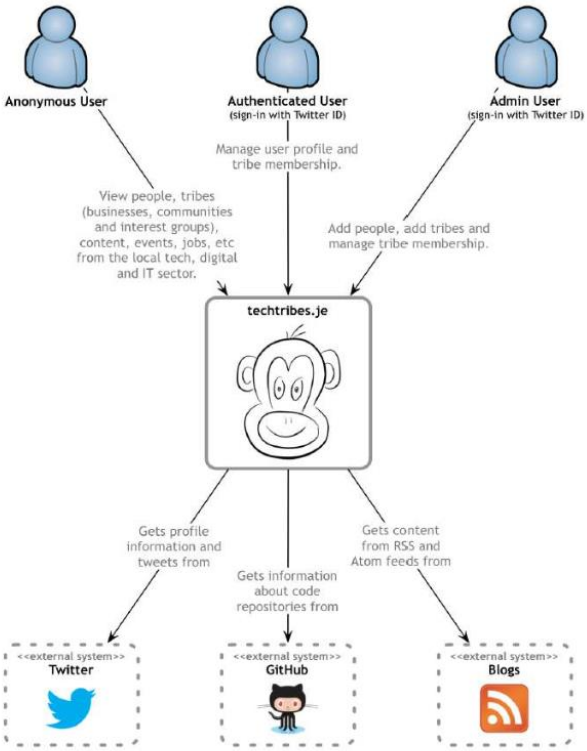
# Architecture (design) principles

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

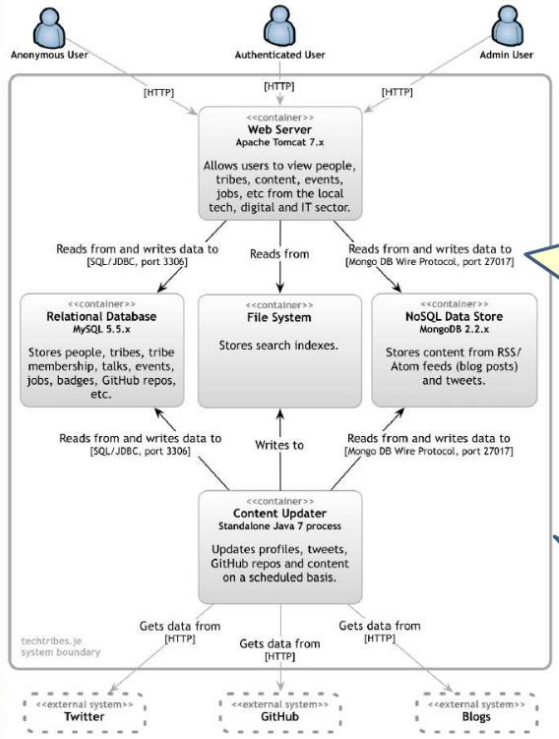
# Structure and Vision



# Techtribes.je context diagram



# Techtribes.je container diagram

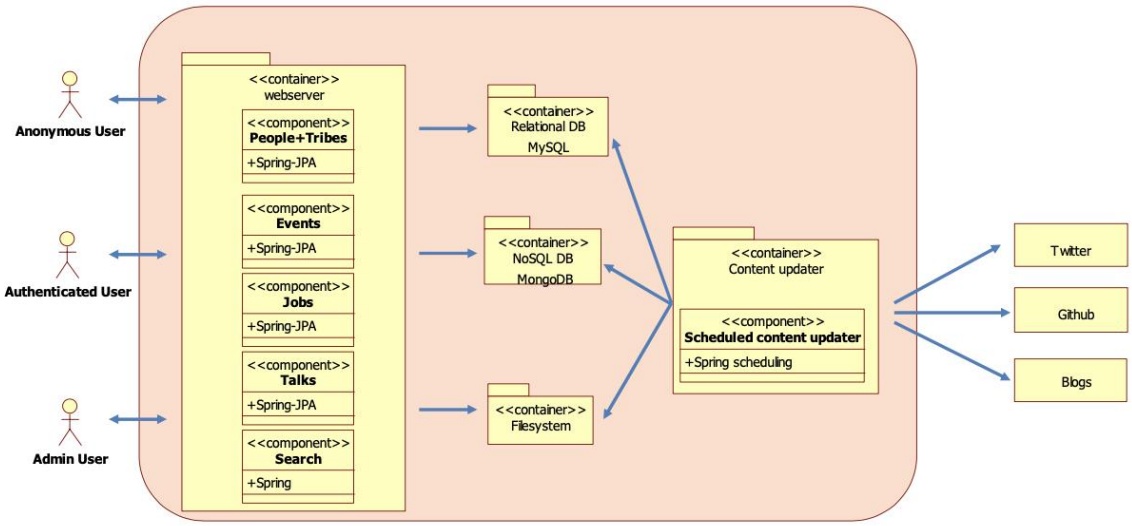



**Describe the interactions :**

- **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**

# Component diagram




for every component , we can specify class diagram, sequence diagram , ....




1 Identify risk: risk storming

# Risk storming

- Quick and fun technique that provides a collaborative way to identify and visualize risks.

# Risk storming

**Step 1. Draw the architecture diagrams**

- on whiteboards or large sheets of flip chart

**Step 2. Identify the risks individually**

- The whole team stand in front of the architecture diagrams and *individually* write down the risks that they can identify, one per sticky note.
- Quantify the risk
- Use different colors of sticky note to represent the different risk priorities.

# Risk storming

## Step 3. Converge the risks on the diagrams

- Everybody places their sticky notes onto the architecture diagrams, sticking them in close proximity to the area where the risk has been identified.

## Step 4. Prioritize the risks

- Take each sticky note (or cluster of sticky notes) and agree on how you will collectively quantify the risk that has been identified.





1 Clustering and Failover

A **failover cluster** is a group of independent computers that work together to increase the availability and scalability of **clustered** roles (formerly called **clustered** applications and services). ... If one or more of the **cluster** nodes fail, other nodes begin to provide service (a process known as **failover**). Jun 6, 2019

LESSON 2

2 Client-server

# Client-server

- Benefits
  - Easy maintenance
    - Application logic in one place (server)
  - Supports many different clients
- Drawbacks
  - Performance can become an issue

    Due to distributed systems

# Layering

- Benefits
  - Layers can be distributed
  - Separation if concern
    - Different skills required in each layer
    - Easy to modify no need to change anywhere else
    - Easy to test
- Drawbacks
  - Development effort can increase
  - Performance can become an issue




2 Pipe and filter

# 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

They shouldn't share any state
They are not independent any more




2 Master-slave

# 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 => Fast
  - Slaves can be duplicated for fault tolerance
- Drawbacks
  - Problem must be decomposable
  - Master is single point of failure

# Microkernal

- Benefits
  - Natural for product based apps
  - Extensibility
  - Flexibility Add, with out changing functionality
  - Separation of concern
- Drawbacks
  - Complexity





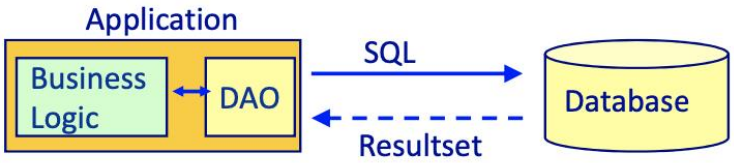
2 DOA/repository class

# Data Access Object (DAO)

- Object that knows how to access the database
- Contains all database related logic
- Also called repository

encapsulated class that has all the logic to talk to db

One DAO for every domain class



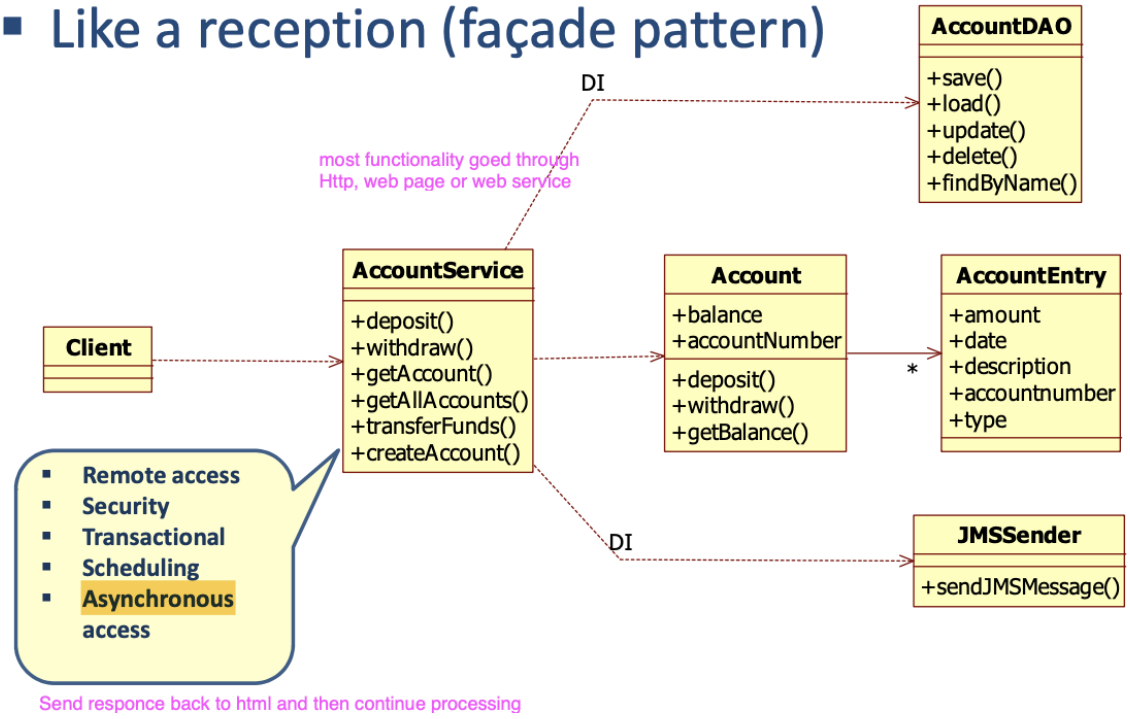




2 Service class

# Service class

- Like a reception (façade pattern)






2 Proxy/Gateway class

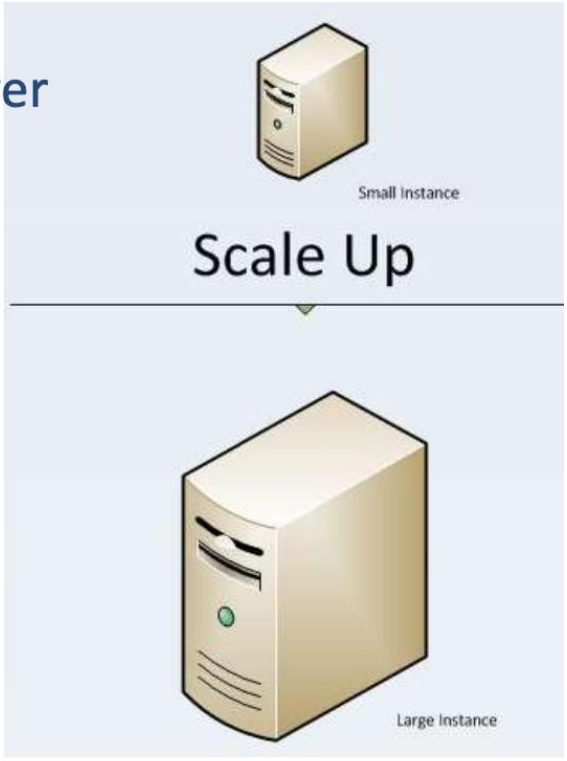
2 Relational database versus NOSQL database

# NoSQL or RDBMS

| NoSQL | RDBMS |
| --- | --- |
| ▪ Schema-free | ▪ Relational schema |
| ▪ Scalable writes/reads | ▪ Scalable reads |
| ▪ Auto high-availability | ▪ Custom high-availability |
| ▪ Eventual consistency | ▪ Strict consistency |

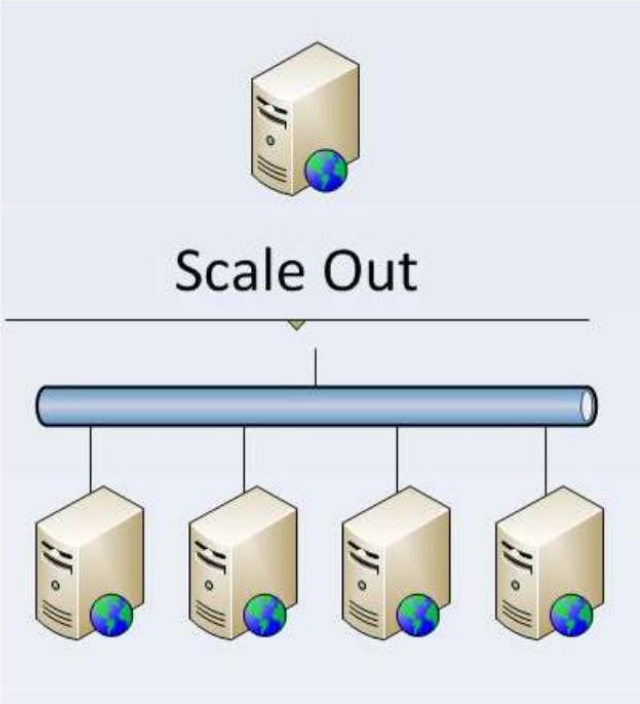
# Vertical Scaling

- Scale up
- Use a more powerful server
- Single point of failure
- Upgrading results in downtime
- Limitations
  - Cost
  - Software does not use all resources
  - Hardware
- Vendor lock-in




# Horizontal scaling

- Scale out
- Divide the data over multiple servers
- Easy to add more servers
  - Without downtime







# Horizontal scaling

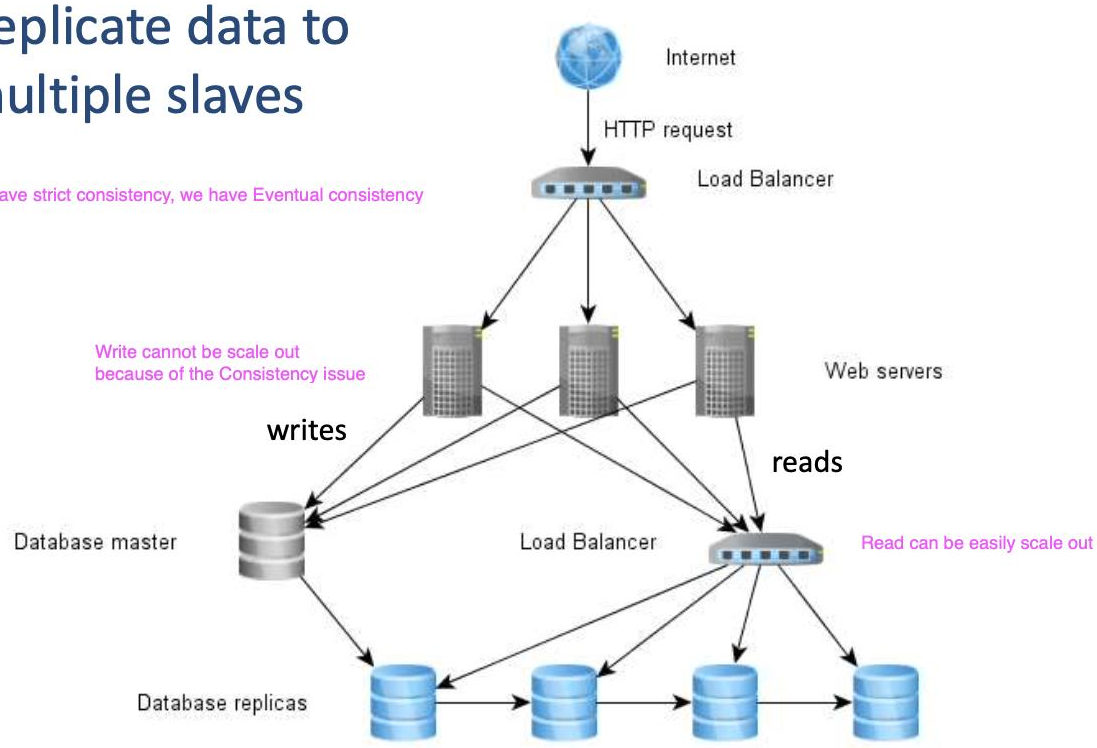
- Replication
- Partitioning
- Sharding

Different ways to do H scaling

# Replication

- Replicate data to multiple slaves

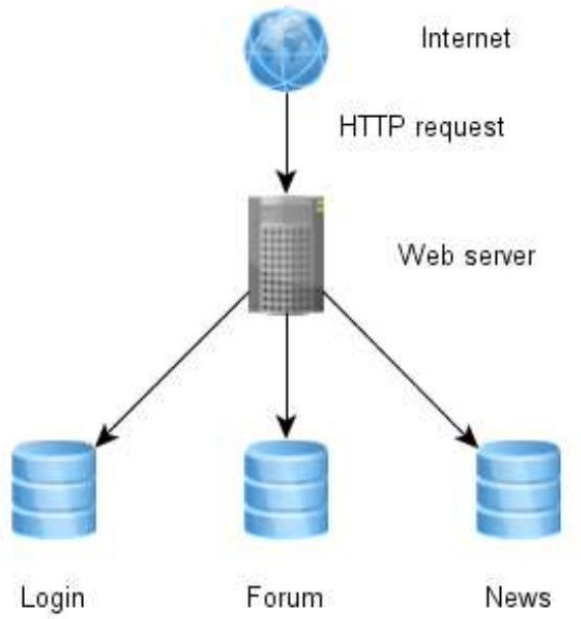
write we don't have strict consistency, we have Eventual consistency



# Functional partitioning

- Split up data in functional areas

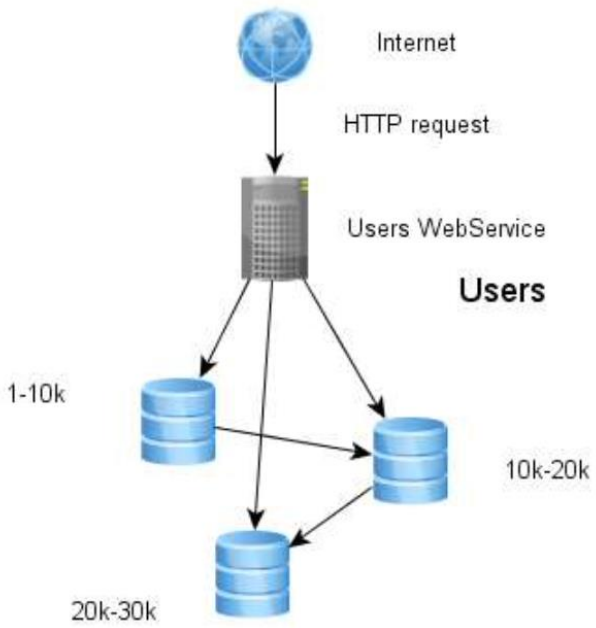
Problem is for the time that we wanna do the Join




# Sharding

- Split the data into pieces(shards) and store them on different nodes

The problem is Join and queries for getting all the data



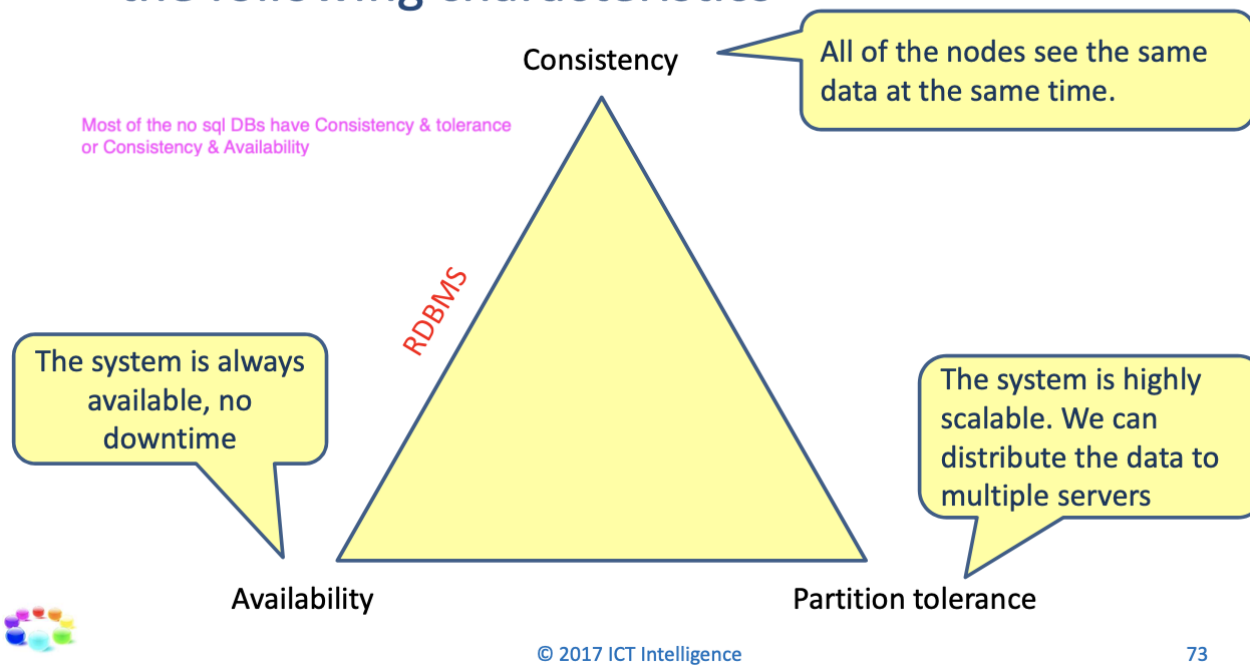




2 Brewers cap theorem

# Brewer's CAP Theorem

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







2 Strict consistency – eventual consistency

- Strict Consistency
  - All read operations must return data from the latest completed write operation, regardless of which replica the operations went to
- Eventual Consistency
  - Readers will see writes, as time goes on: "In a steady state, the system will eventually return the last written value".

3 Dependency injection

# Main point

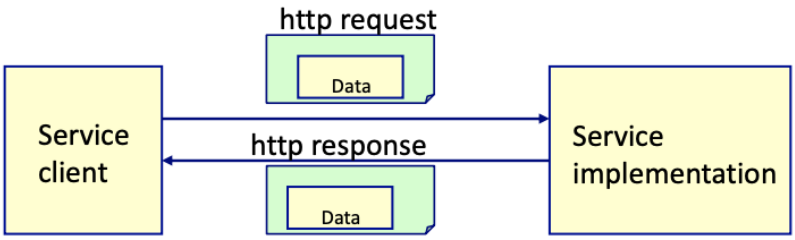
- Dependency injection is a flexible technique to connect objects together by configuration.
- Everything in creation is connected with everything else in its source, the Unified Field, the home of all the laws of nature.





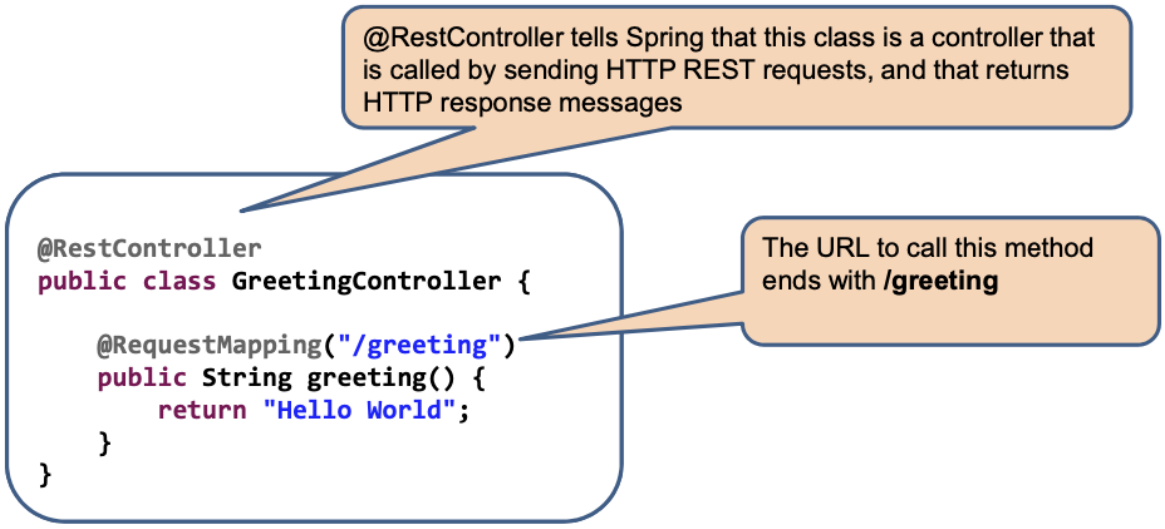
3 REST

# RESTful Web Services




- The URL specifies the resource to act on
- Not bound to a specific data format
- Data in HTTP messages In service we should create a mapping between httpmethods and service methods
  - GET message for retrieving data
  - POST message for creating data
  - PUT message for updating data
  - DELETE message for deleting data

# Simple Rest Example: the controller







3 Mongo

# Spring Mongo libraries

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

# The Mongo Documents

```
@Document   = @Entity for relational DB
public class Student {

    @Id
    private String studentId;
    private String name;
    List<Course> courses = new ArrayList();

    public void addCourse(Course course) {
      courses.add(course);
    }

    public void print() {
      System.out.println( "Student{" + "studentId=" + studentId
      + ", name=" + name + ", [" );
      for (Course course : courses) {
        course.print();
        System.out.print(" ,");
      }
      System.out.println( "]}");
    }
    ...
}
```

```
@Document
public class Course {
  @Id
  private String courseId;
  private String name;

  public void print() {
    System.out.println( "Course{" +
    "courseId=" + courseId + ", name="
    + name + "}" );
  }
    ...
}
```



# The repository = Data Access Object (DAO)

```
public interface StudentRepository extends MongoRepository<Student, String> {

}
```

application.properties

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=testdb
```

4 Entity

Different type of class in DDD

# Domain Model Patterns

- Entities — Classes that stores in DB <<Entity>>
- Value objects — A Value Object is an immutable type that is distinguishable only by the state of its properties. That is, unlike an Entity, which has a unique identifier and remains distinct even if its properties are otherwise identical
- Domain services — is used to perform domain operations and business rules.
- Domain events — Classes that represent important events in the problem domain that have already happened
- Aggregates — group of obj that belongs together

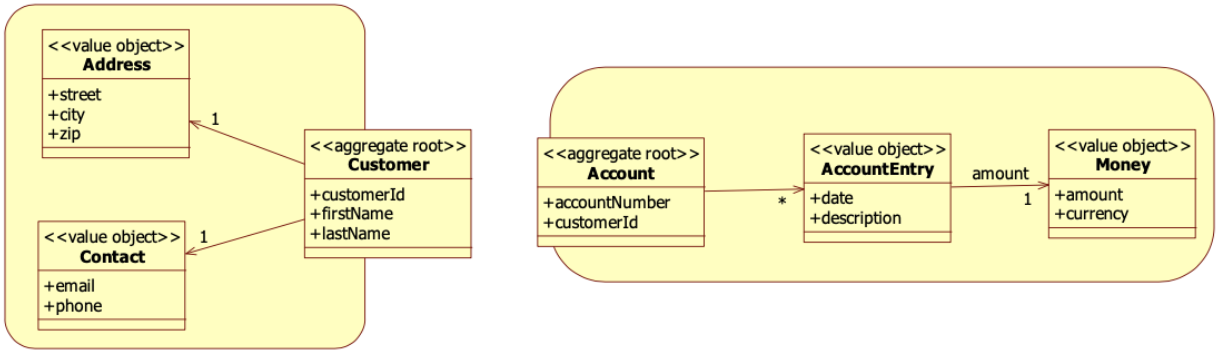
aggregates are basically small application by its own

# 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.
- If you can safely replace an instance of a class with another one which has the same set of attributes, that's a good sign this concept is a value object
- Value objects don't need their own tables in the database.
  - The data can be embedded into the entity table
- Always prefer value objects over entities in your domain model.

# Referencing other aggregates

- Using an Id





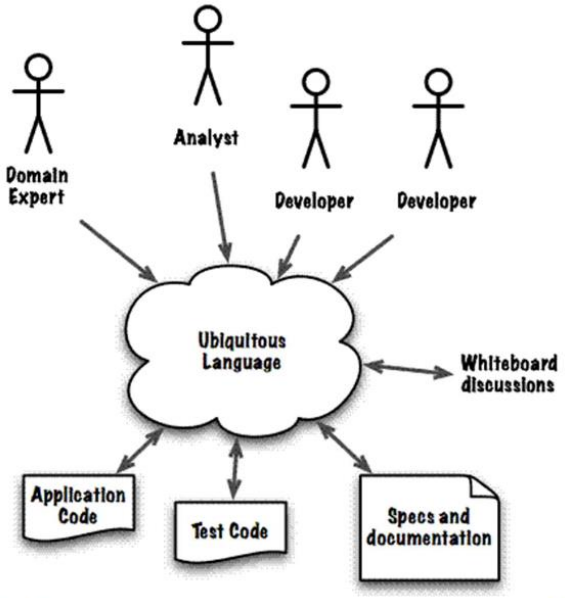


LESSON 5

5 Ubiquitous 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



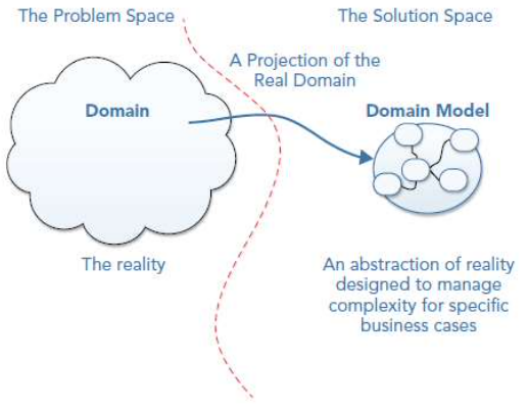




5 Rich domain model

# 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




# Advantages of a domain model

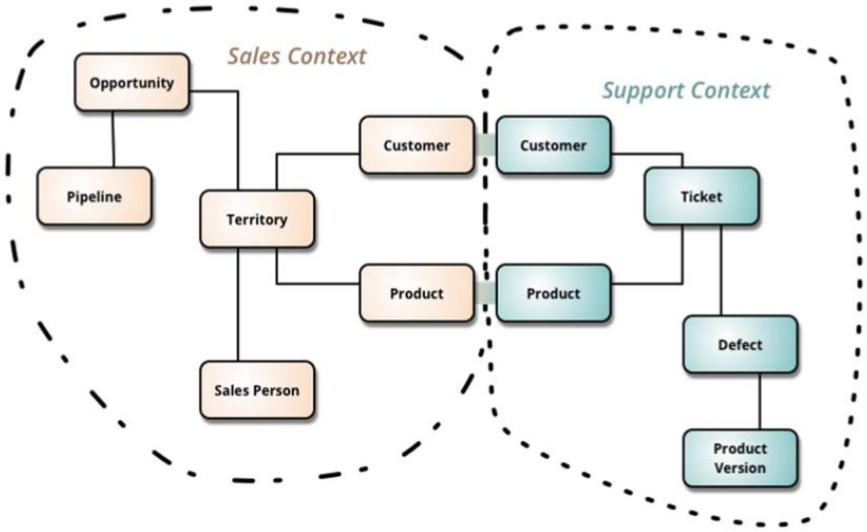
- Improves understanding
- Validates understanding
- Improves communication no documentation
- Shared glossary
- Improves discovery




5 Bounded context

# Context

- A specific domain term may have a different definition in a different context




# Bounded context

it is a context with its own language, with its own rules, it is also called subdomain or sub systems

- Create explicit boundaries in terms of
  - Team organization
  - Usage of the system
  - Physical manifestation (code, database)
- Create a different domain model per bounded context
  - A model is only valid within the scope of the bounded context



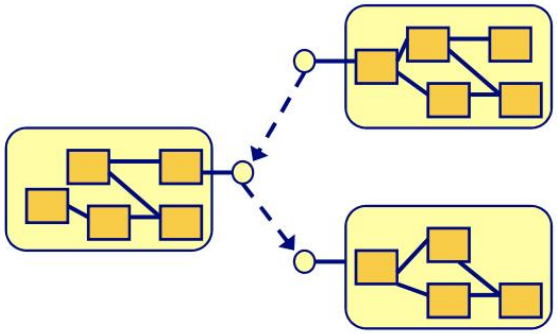


5 Components

# Component Based Development

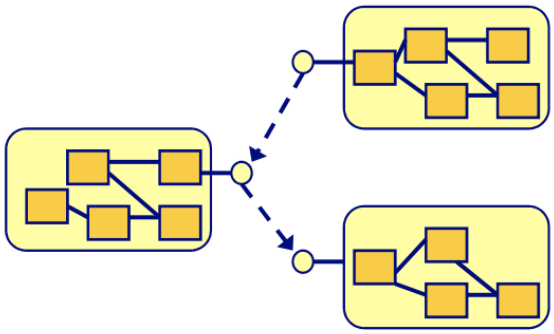
- Decompose the domain in functional components

component is some classes together, an aggregate part of the system



# 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 if we remove a component and replace with another, it will work if it uses the same interface
- A component can be a single unit of deployment



# Interface design

- Start with the client first
- Single responsibility principle
- Interface segregation principle
- Easy to use
- Easy to learn




5 DTO's

# Data Transfer Objects (DTO)

- Object that contains only attributes and getters and setters


# Events

```
public class AddCustomerEvent {
  private String message;

  public AddCustomerEvent(String message) {
    this.message = message;
  }

  public String getMessage() {
    return message;
  }
}
```

A simple event class

Immutable

data should be in the event to sent

# Event publisher and listener

```
@Service
public class CustomerServiceImpl implements CustomerService {
  @Autowired
  private ApplicationEventPublisher publisher;

  public void addCustomer() {
    publisher.publishEvent(new AddCustomerEvent("New customer is added"));
  }
}
```

Inject a publisher

```
@Service
public class Listener {

  @EventListener
  public void onEvent(AddCustomerEvent event) {
    System.out.println("received event :" + event.getMessage());;
  }
}
```

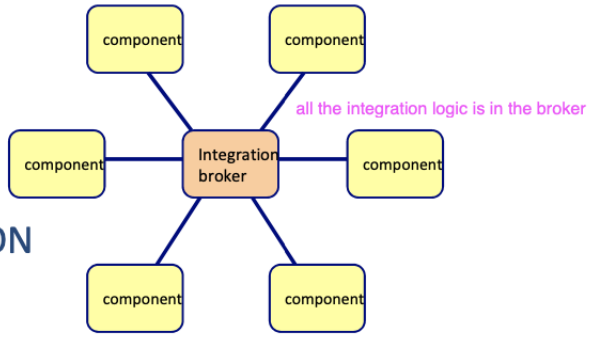
Listen to AddCustomer events

we do not put the whole data in event, only some data in the DTO

# Hub and Spoke

connect all together

- 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

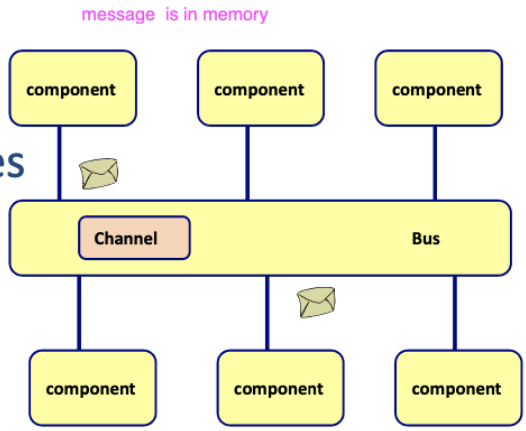
- Benefits
  - Separation of integration logic and application logic
  - Easy to add new components
  - Use adapters to plugin the integration broker
- Drawbacks
  - Single point of failure
  - Integration brokers are complex products
  - Integration broker becomes legacy itself leghacy means use, not old

# 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

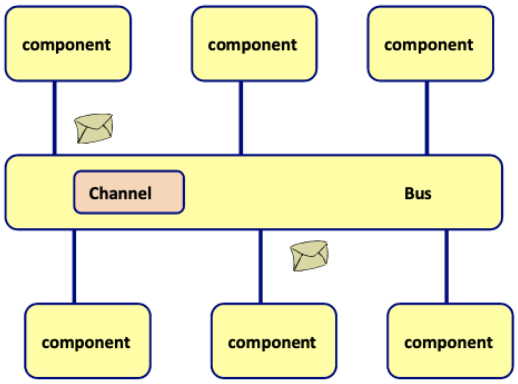
# Event bus

- Publish-subscribe
  - Subscribers subscribe to particular channels
  - Publishers publish messages to particular channel
  - Subscribers receive messages from channels



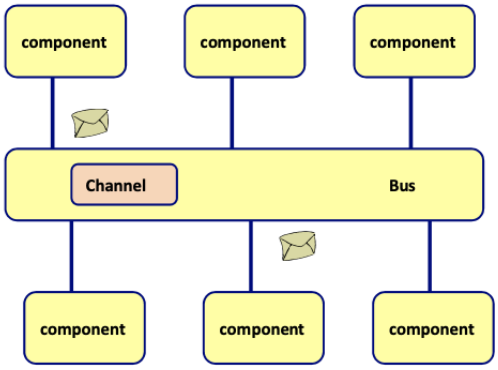

# Event bus characteristics

- Asynchronous
- Loose coupling
- Easy to add new components
- Separation of integration logic and 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







# ESB

- Benefits
  - Separation of integration logic and application logic
  - Easy to add new components
- Drawbacks
  - ESB's are complex products
  - Performance can be an issue

- Messaging channel patterns
  - Point-to-point: only one receiver will receive the message
  - Publish-Subscribe: every subscriber will receive the message
  - Datatype Channel: use a channel for each data type, so that the receiver knows how to process it
  - Invalid Message Channel: for messages that don't make sense for the receiver
  - Dead Letter Channel: for messages that can't be delivered
  - Channel adapter: connect the application to the messaging system
  - Message bridge: connect 2 messaging systems

- Message construction patterns
  - Command message.- Send command as message succh as last trade price
  - Document message.- Doc with data as a message such as Order
  - Event message.- Event as message such as Price change event. Event =something that happened in the past, Cmmd => something need to happen

- Request-Reply
- Return address
- Message expiration

- Message Endpoint
  - Event driven consumer
  - Polling consumer
  - Competing consumers
  - Message dispatcher
  - Durable and Non-Durable subscribers
  - Service activator

- Message Routing
  - Content based router
  - Dynamic router.- Dynamic router has some rules, knows when gets some thing , where should send to
  - Message filter.- It filter gadgets => just widget can continue
  - Recipient list.- contains the logic to where the message should be sent to the whole list of receivers
  - Splitter.- Split up the msg, you sent the order and get the order lines, most of the time you need to write logic
  - Aggregator
  - Sequencer
  - Composite Message Processor
  - Routing Slip .- specifying the sequence of processing steps.

- Message transformation
  - Envelope Wrapper
  - Claim Check
  - Normalizer
  - Canonical Data Model

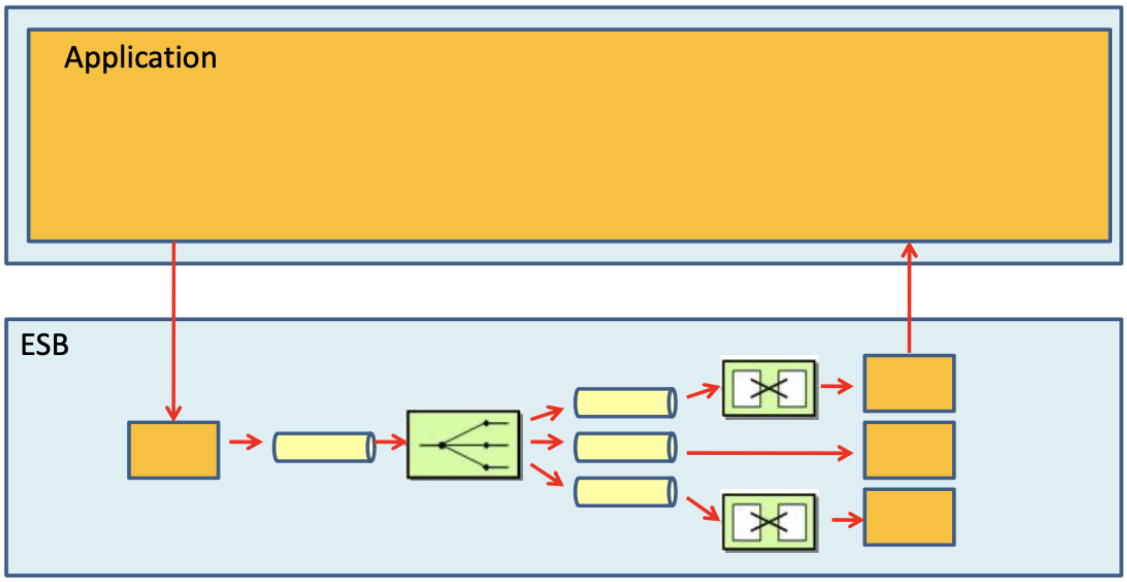
- Management
  - control Bus to manage an enterprise integration system.
  - Deteur
  - Wire tap
  - Message history
  - Message store
  - Test message

LESSON 7

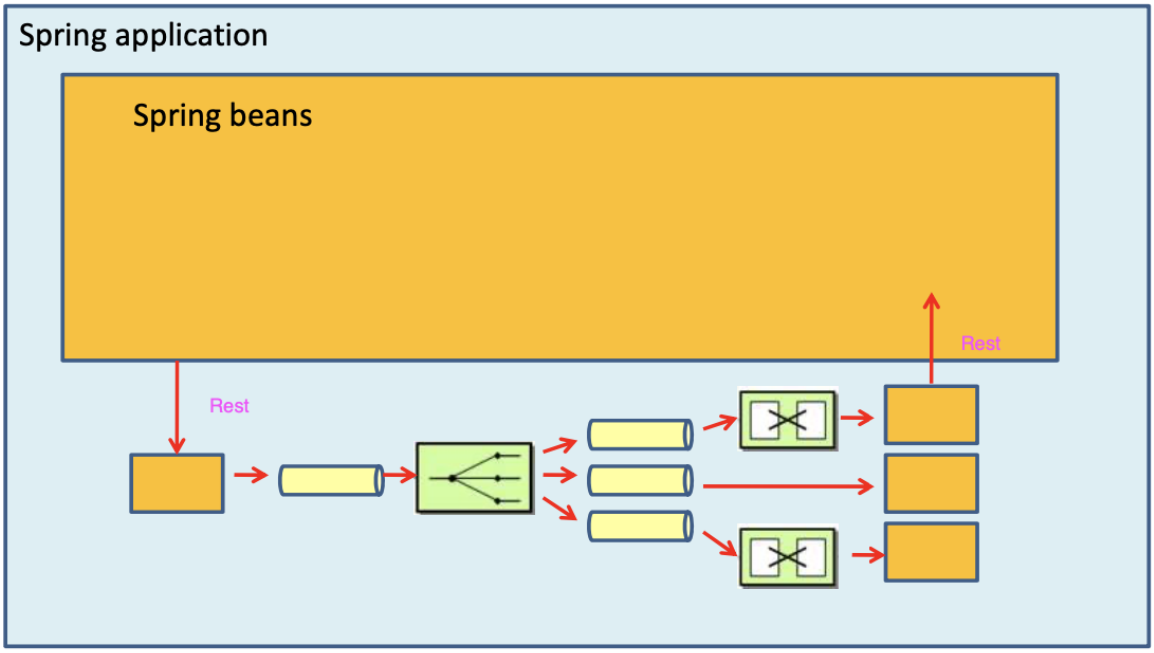
7 ESB versus Integration Framework

# ESB

- Runs outside the application
  - Needs to be installed, started, stopped, monitored.



# Using Spring Integration




- Use SI inside your application