Quality Attributes & Software Architecture

Nursultan Askarbekuly

Agenda

Plagiarism.

What we have covered so far.

Quality attributes.

Software Architecture.

Plagiarism

- Copied something from somewhere?
- Mention the source.

- A colleague asks for help?
- Explain how it’s done.
- Do NOT share the solution.

- What is the cheating policy?
- Both sharer and copier get 0 (first time).

What have we learnt so far?

- Understanding the domain.
- Customer collaboration feedback.
- Backlogging.
- Teamwork in Iterations.
- Version control workflow.

The rest of the course:
Let’s improve our engineering.

What are quality attributes?

More generally, what are non-functional requirements?

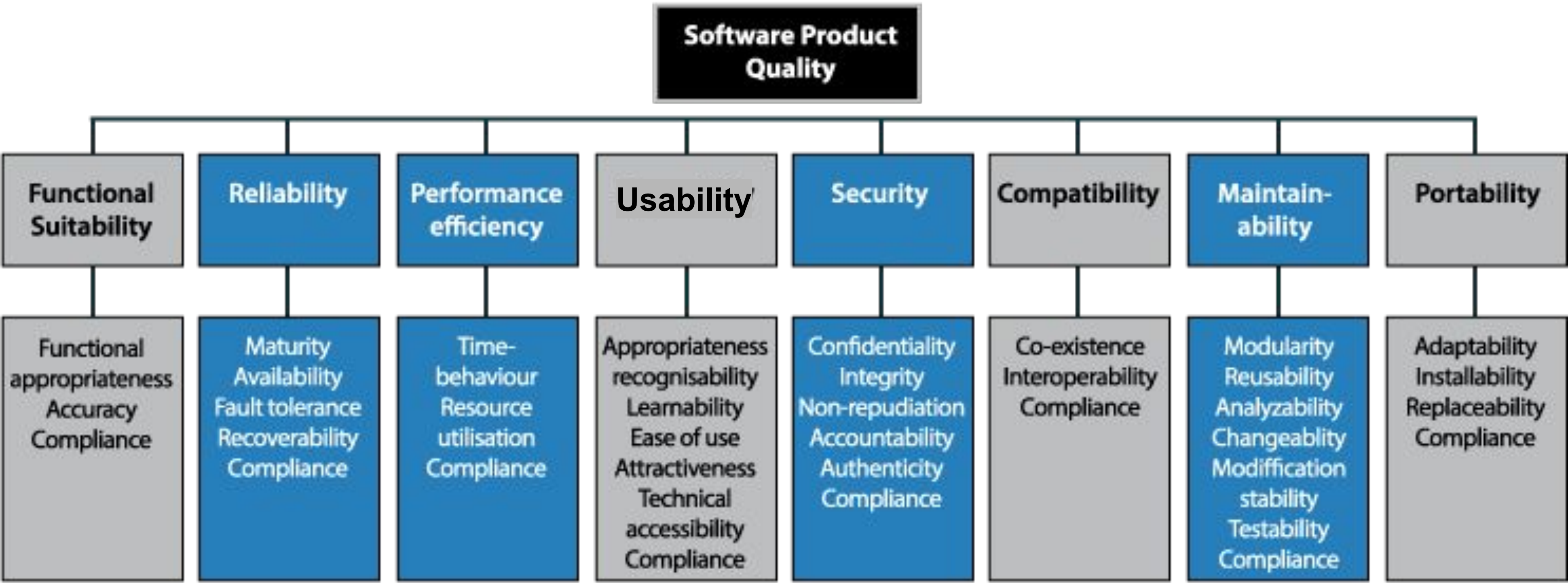
Views on quality

- **Transcendent**: Experiential. Quality can be recognized but not defined or measured
- **Product-based**: Level of attributes & Internal quality
- **User-based:** Fitness for purpose, quality in use
- **Value-based:** Attributes/fitness vs cost
- **Manufacturing:** Conformance to specification, process excellence

Functionality VS Quality Attributes

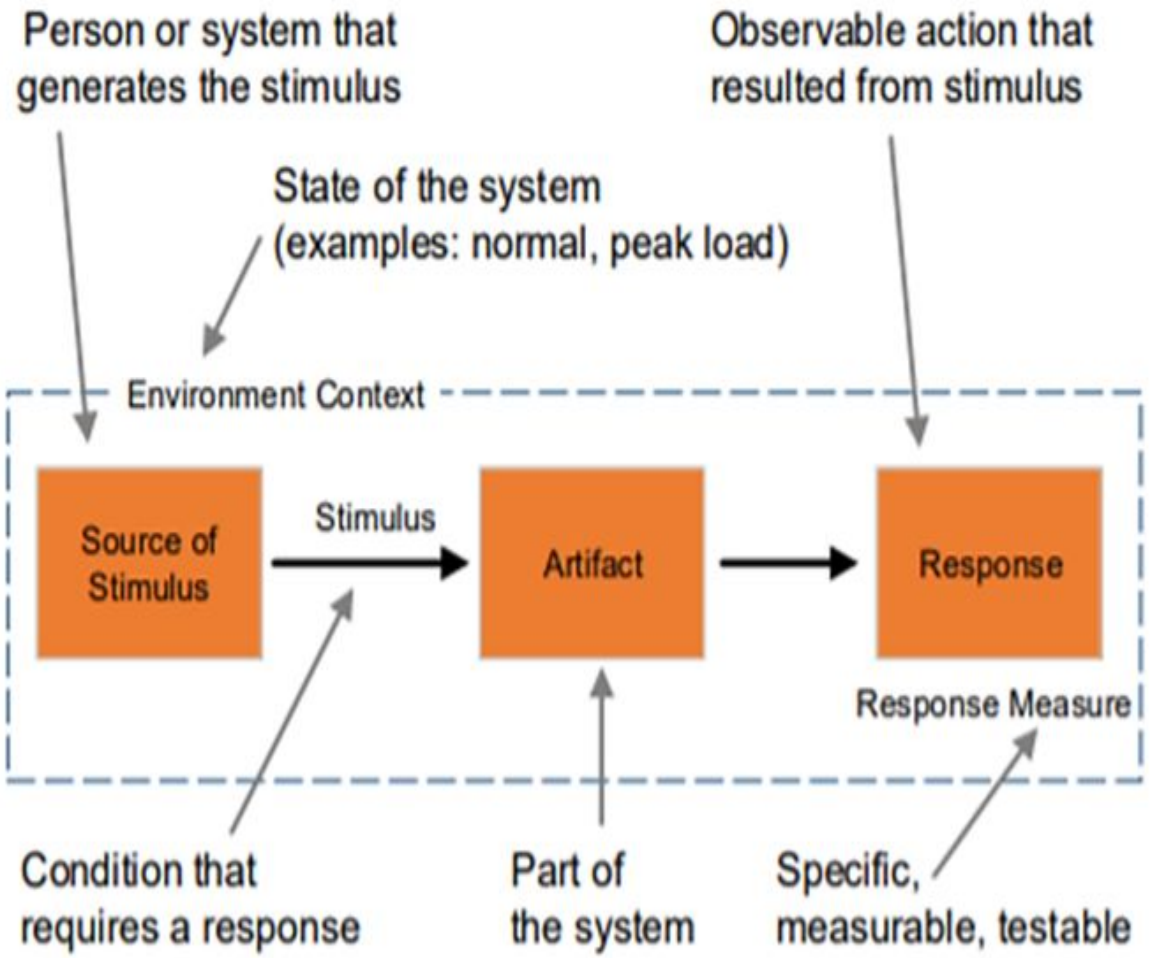
- Maintainability
- Performance
- Availability
- Security
- Interoperability
- Usability

ISO 25010 Quality Model


Software Product Quality
Functional Suitability
Reliability
Performance efficiency
Usability
Security
Compatibility
Maintain-ability
Portability
Functional appropriateness Accuracy Compliance
Maturity Availability Fault tolerance Recoverability Compliance
Time-behaviour Resource utilisation Compliance
Appropriateness recognisability Learnability Ease of use Attractiveness Technical accessibility Compliance
Confidentiality Integrity Non-repudiation Accountability Authenticity Compliance
Co-existence Interoperability Compliance
Modularity Reusability Analyzability Changeablity Modiffication stability Testability Compliance
Adaptability Installability Replaceability Compliance

How to measure quality attributes?

QAs are characterized through scenarios


Person or system that generates the stimulus
Observable action that resulted from stimulus
State of the system (examples: normal, peak load)
Environment Context
Source of Stimulus
Stimulus
Artifact
Response
Response Measure
Condition that requires a response
Part of the system
Specific, measurable, testable

Availability scenario: Requirement & Test

The **heartbeat monitor** determines that **the server is non-responsive** during **normal operations**. The **system informs the operator** and **continues to operate** with **no downtime**.

Sr.No	Part	Values
1.	**Source of stimulus**	Heartbeat Monitor
2.	**Stimulus**	Server Unresponsive
3.	**Environment**	Normal Operation
4.	**Artifact**	System
5.	**Response**	Inform Operator, Continue to Operate
6.	**Response measure**	No Downtime

Example: Face Recognition Service

We need a service that does one-shot face-identification. There are two main use-cases for our product.

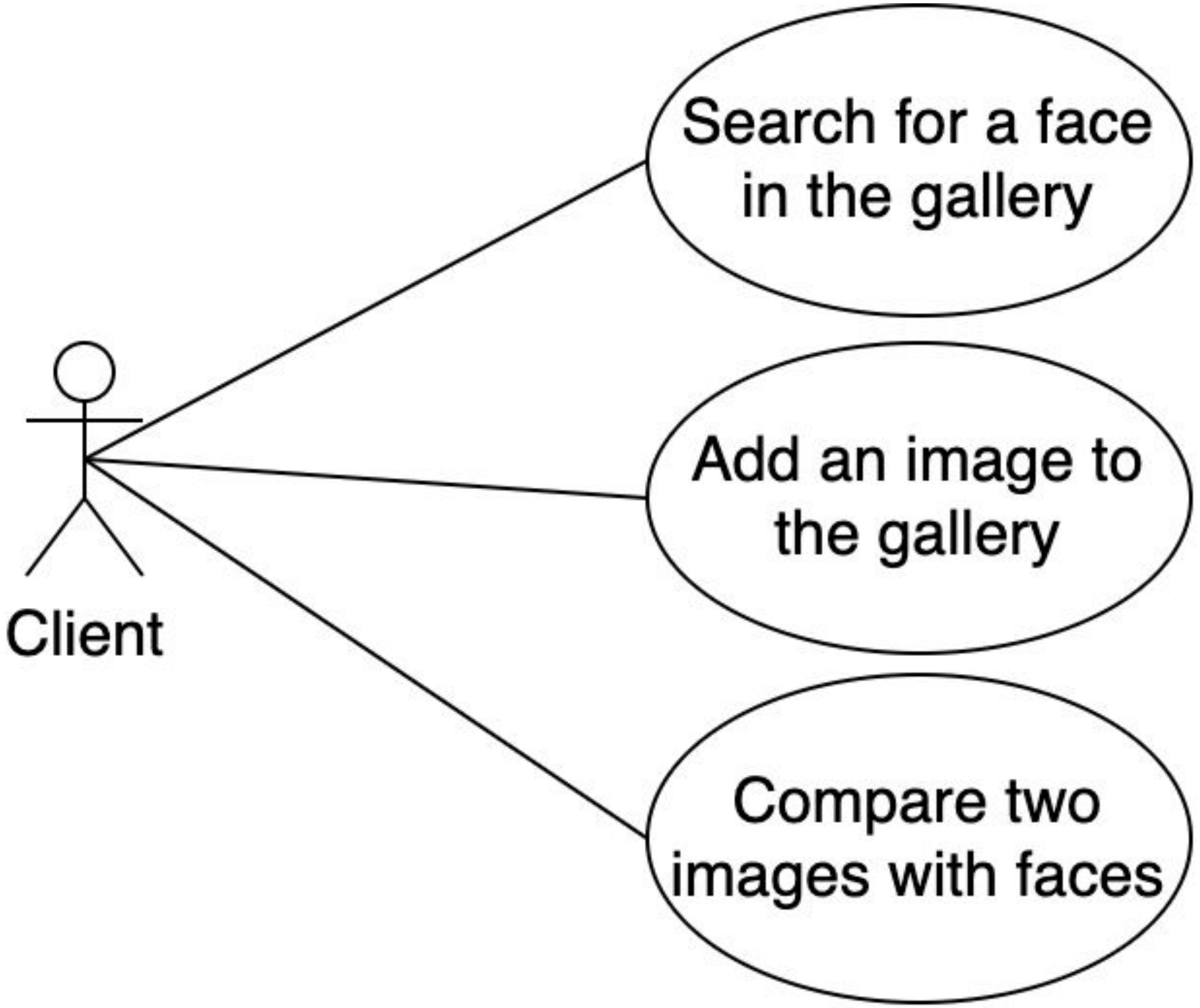
1:N Face Search: Check whether a face belongs to a person from our database, i.e. identify. We take a face and then search for a similar face in the gallery of faces.

1:1 Face Comparison: Given two photos check if they have the same faces.

The demand for our service might vary over time (increase or decrease) and the system should adjust to it.

We should also be able to easily change the face identifier module, and also where we store our data and images.

Face recognition service


Search for a face in the gallery
Add an image to the gallery
Compare two images with faces
Client

Face recognition service

Functionality	**Ideas how to achieve**
F1. Add an image to the gallery	It will be a service to send requests to. We will process the requests using some ML image processing model. We need some persistence like Image Storage and Database.
F2. Search for a face in the gallery	
F3.Compare two images with faces	

Quality Attributes

Q1. Modifiability	Developer should be able to change the ML model while application is running. Modifications are made with no side effects.
Q2. Performance Efficiency (Scalability)	Client requests are increasing / decreasing. Free or block resources based on the load (amount requests).

https://docs.google.com/document/d/1q0IVw7tTnqPYRaOnc2PiXY_CZYp3duAheSWlBiuQhrM/edit?usp=sharing

Q1. **Attribute:** Modifiability

Scenario: Developer intends to change / replace the face identifier, database or image storage in design time.

The module is changed and unit tested in a business day without any side effects to the core business component of the system.

Q2. **Attribute:** Performance Efficiency (Scalability)

Scenario: In production time, the customer demand (# of requests) goes up/down.

The system is able to adjust to the demand and increase/decrease the resources with no downtime and no delays in response time.

How to achieve quality attributes?

How to achieve quality attributes?

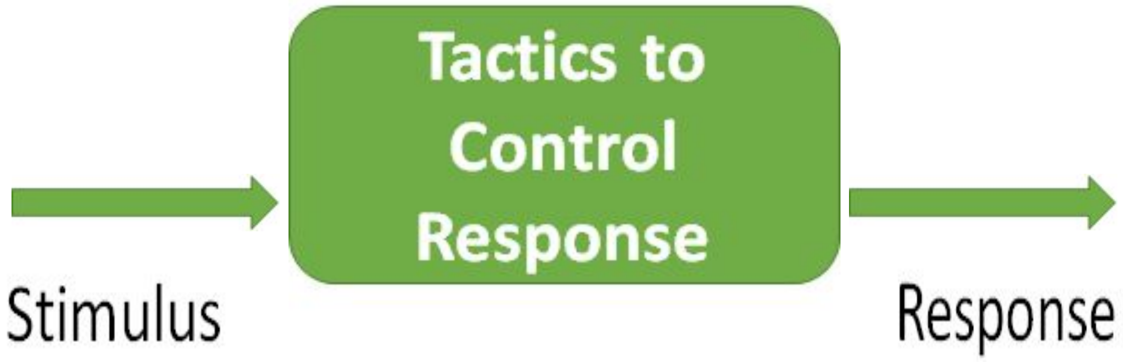
Software Architecture

- A high level view on your system
- Documented through diagrams and commentary
- Describe are common perspectives:
  - structural (static)
  - behavioural (dynamic)
- Directly corresponds to quality attributes
- Has a set of commonly used practices (tactics & patterns)

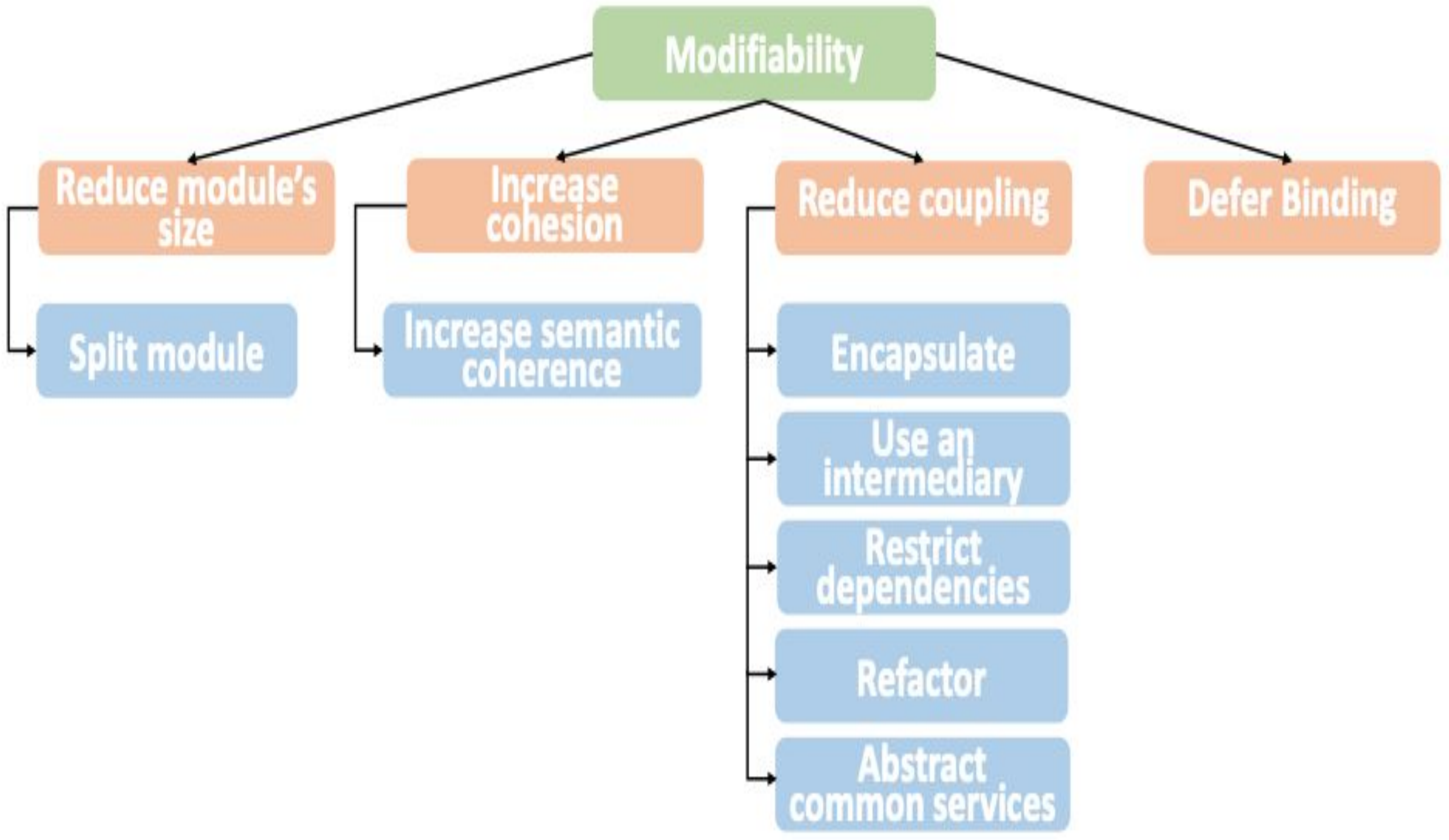
Quality Attribute	Ideas how to achieve
Q1. **Attribute:** Modifiability **Scenario:** Developer intends to change / replace the face identifier, database or image storage in design time. The module is changed and unit tested in a business day without any side effects to the core business component of the system.	Move the identifier into a separate module and specify an interface for it. Do the same with the functionality dealing with the database and image storage instances.

Architecture Tactics

- Tactics are techniques that an architect can use to achieve the required quality attributes
- The focus of a tactic is on a single quality attribute response



Modifiability Tactics



Example

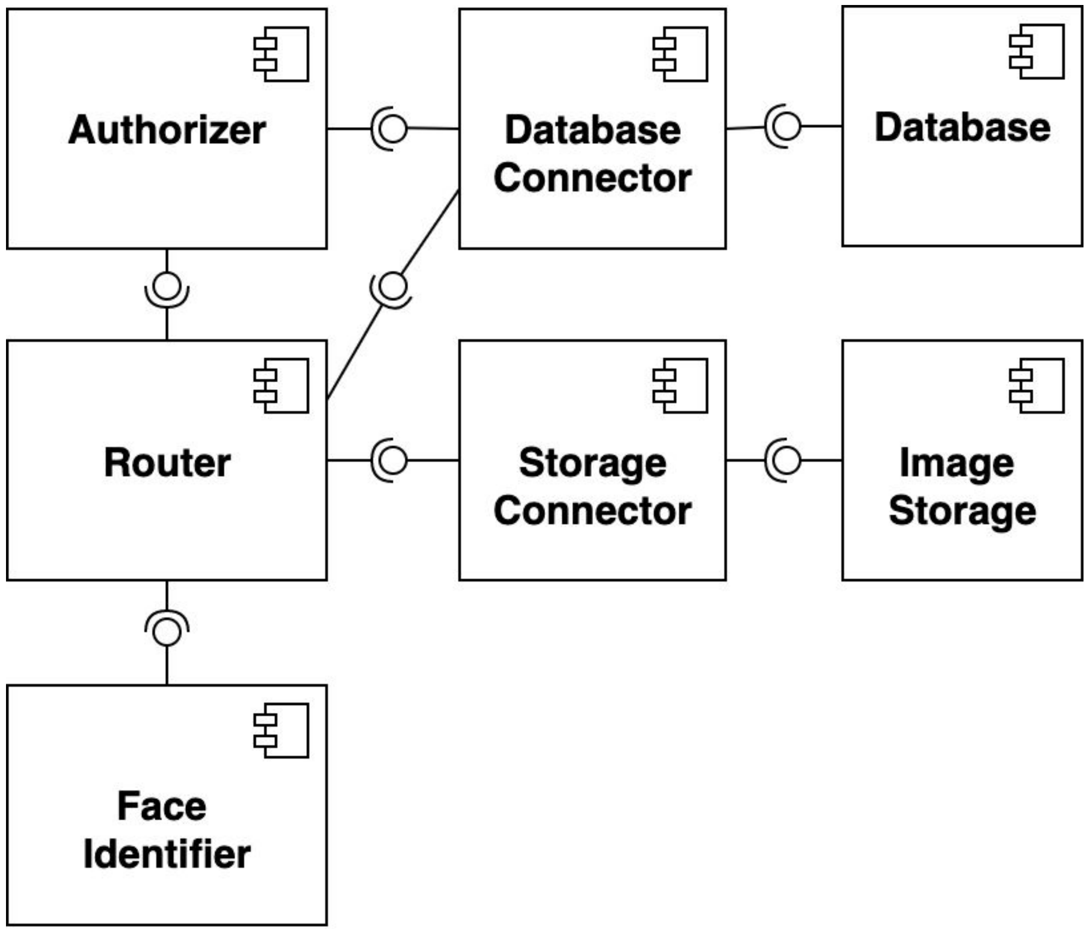
Try to achieve:

Modular design

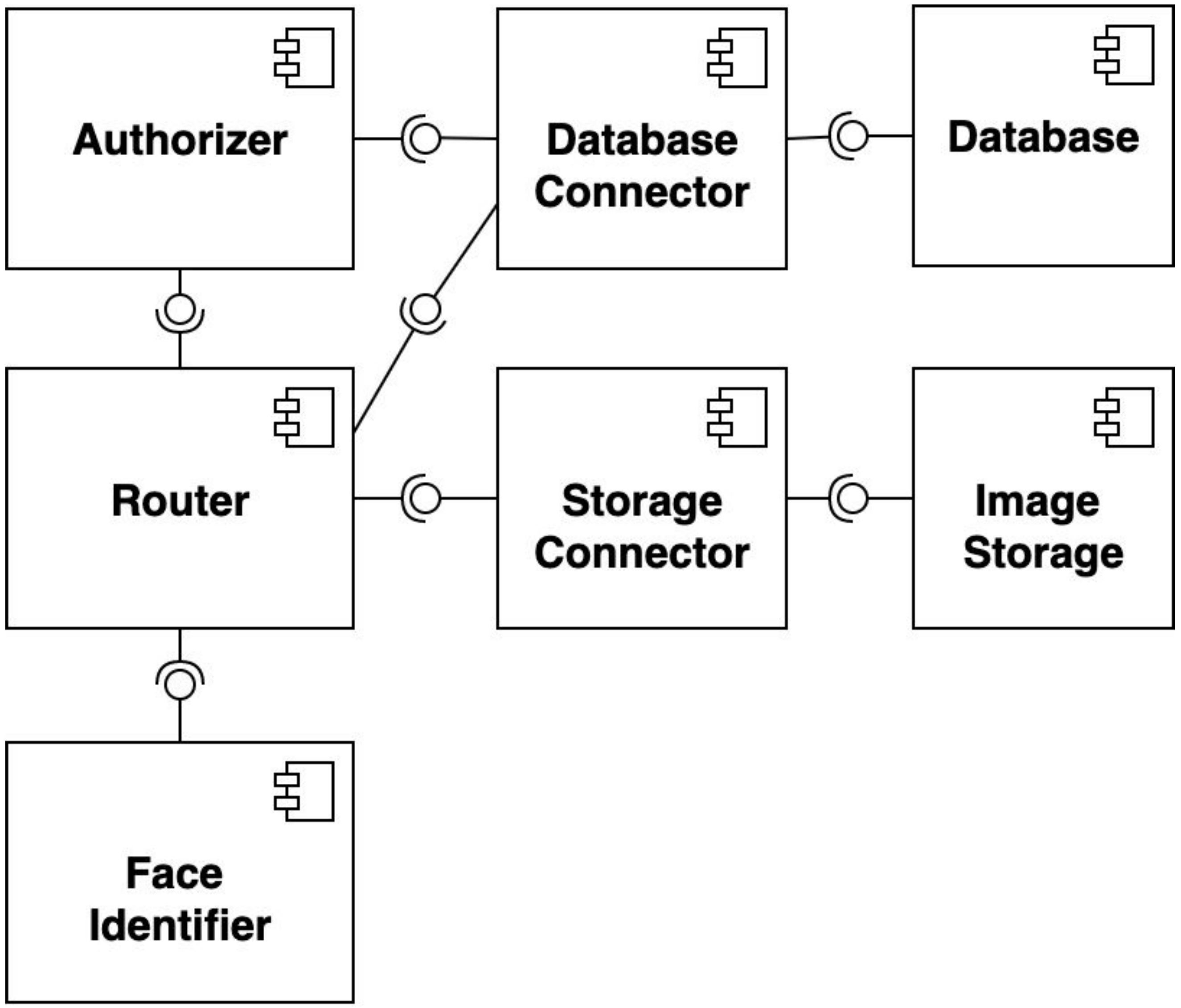
Loose coupling: Few "use-relations"

High cohesion: Sensible distribution of responsibilities

Replaceable modules



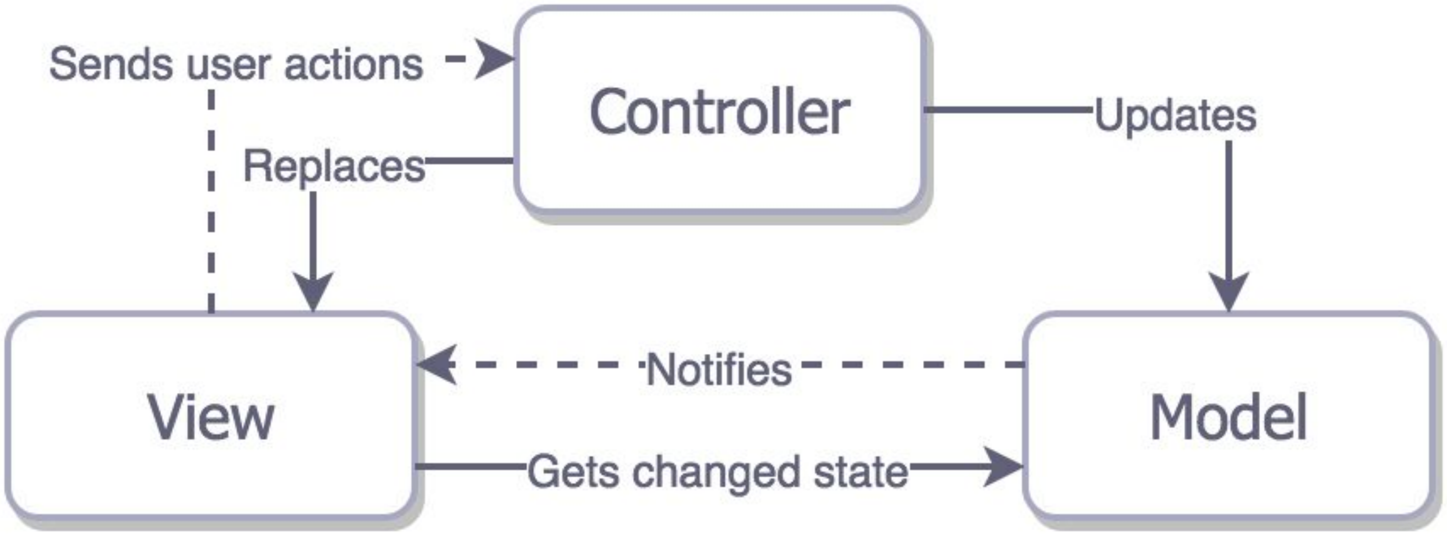
Face ID service

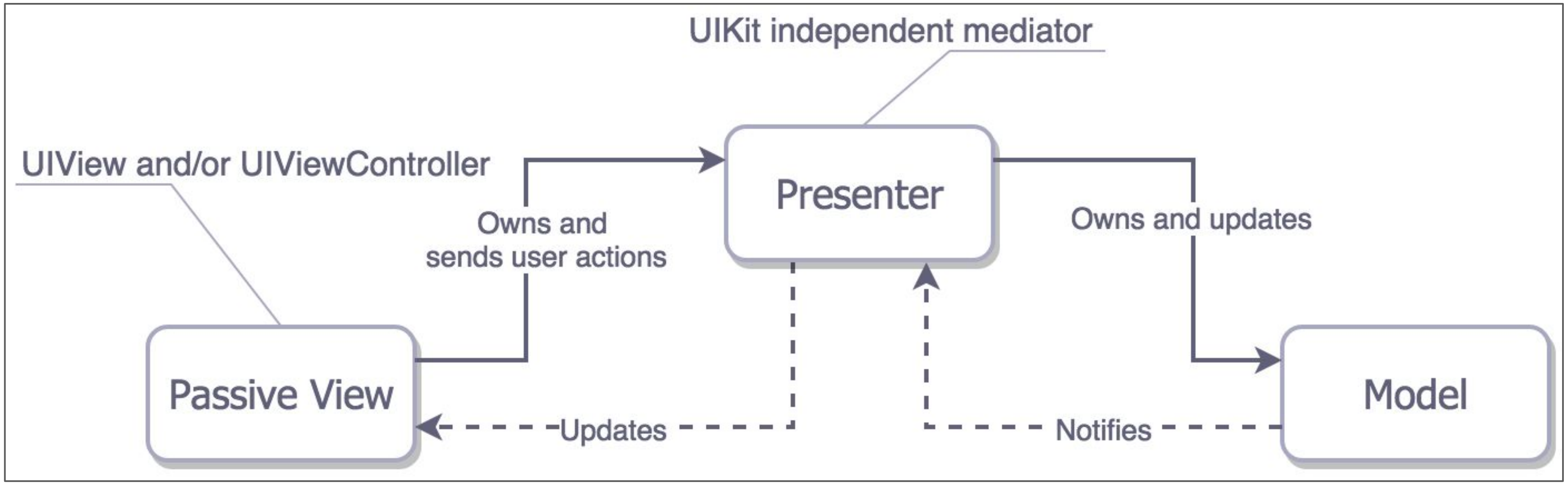


Architecture has patterns, which are collections of tactics.

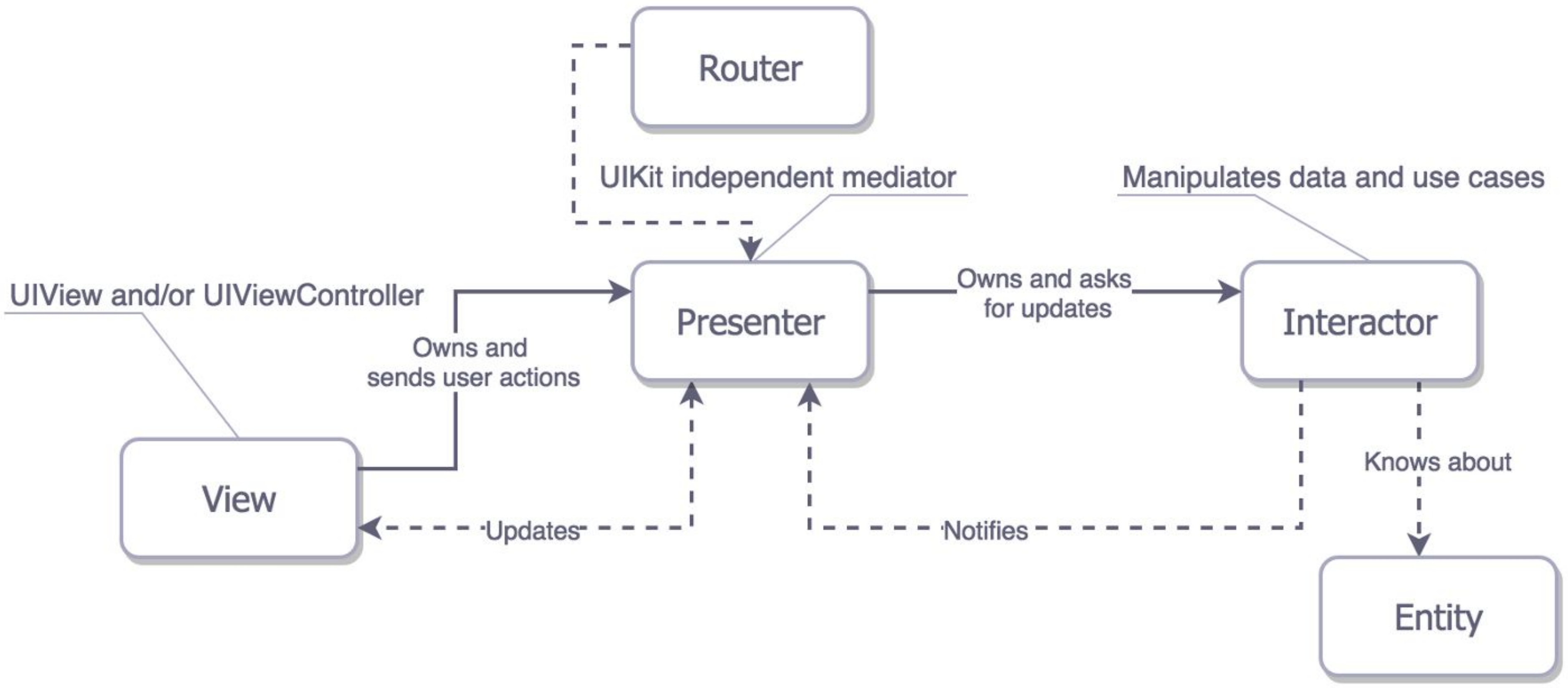
They come with trade-offs.

Architectural Patterns in iOS





Architectural Patterns in iOS


Router
UIKit independent mediator
Manipulates data and use cases
UIView and/or UIViewController
Presenter
Owns and asks for updates
Interactor
Owns and sends user actions
View
Updates
Notifies
Knows about
Entity

Improving maintainability: Layered Pattern

Strict Layered Pattern

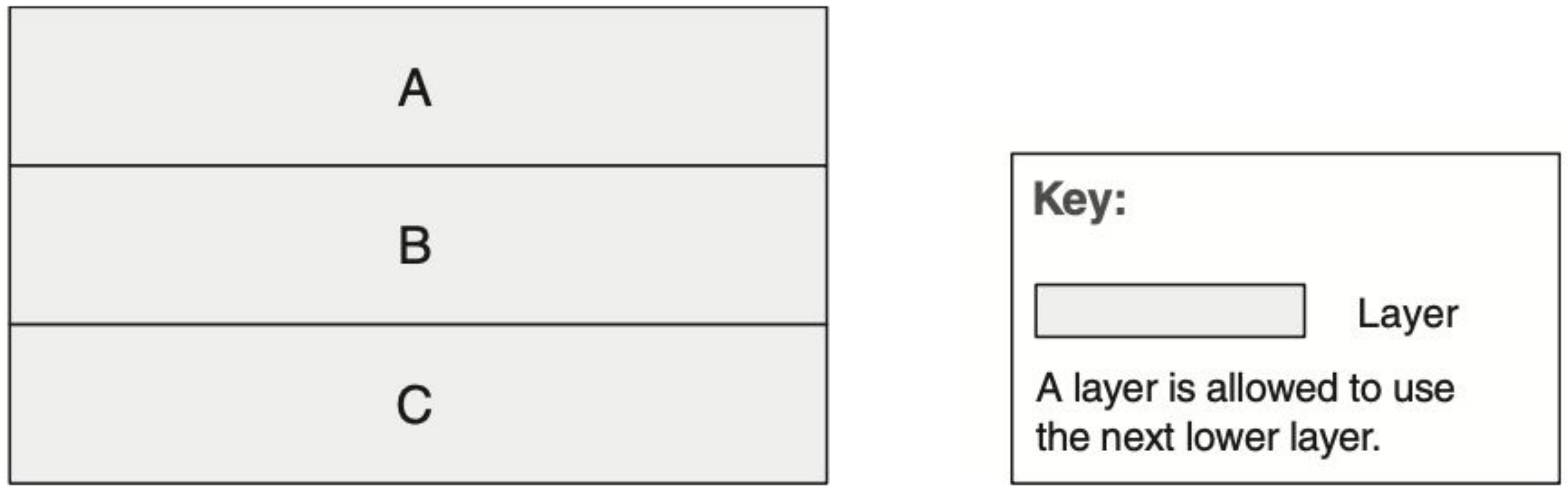


FIGURE 13.1 Stack-of-boxes notation for layered designs

Variations of the Layered Pattern

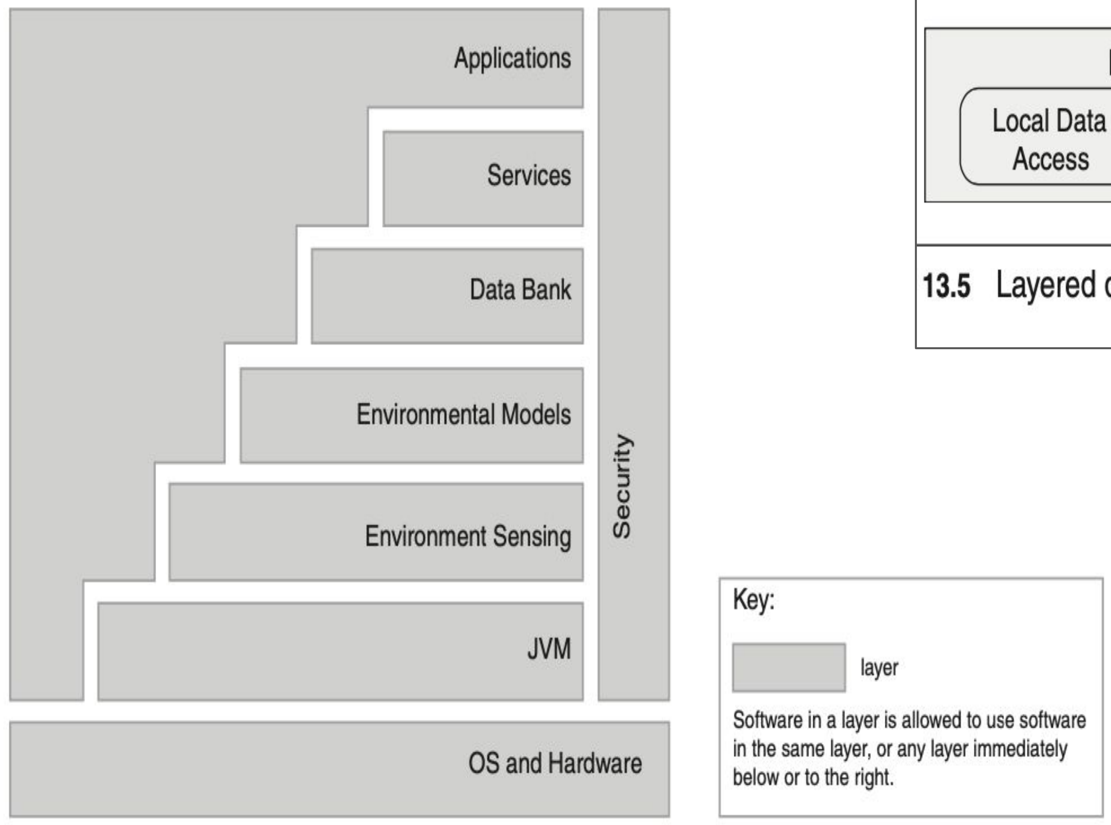
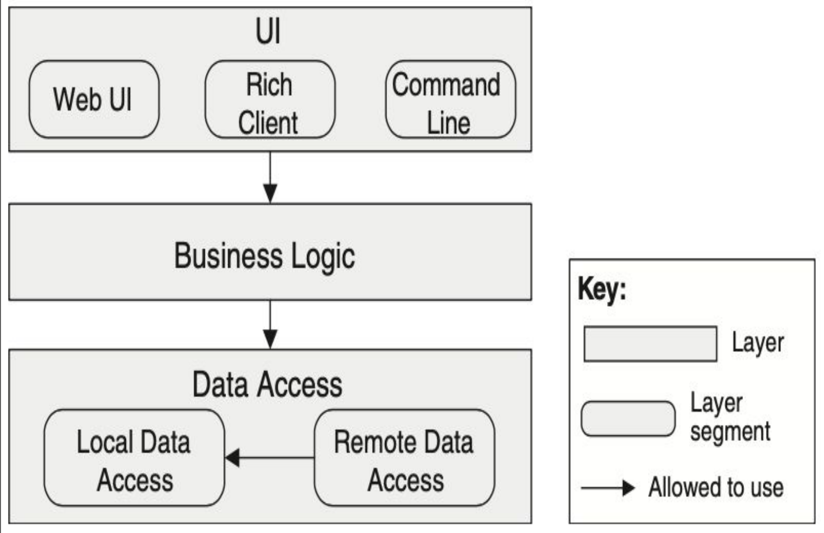
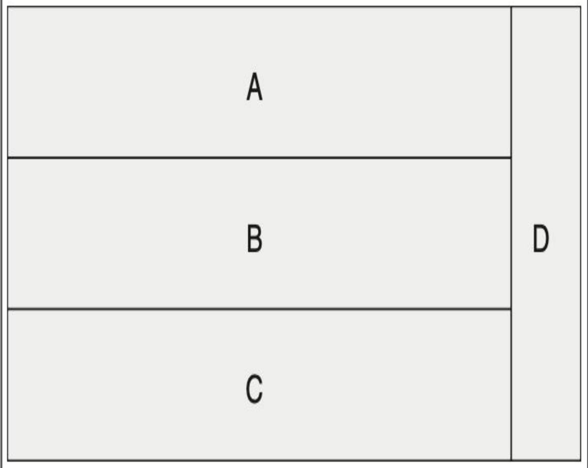

Applications
Services
Data Bank
Environmental Models
Environment Sensing
JVM
OS and Hardware
Security
Key:
layer
Software in a layer is allowed to use software in the same layer, or any layer immediately below or to the right.

FIGURE 13.2 A simple layer diagram, with a simple key answering the uses question

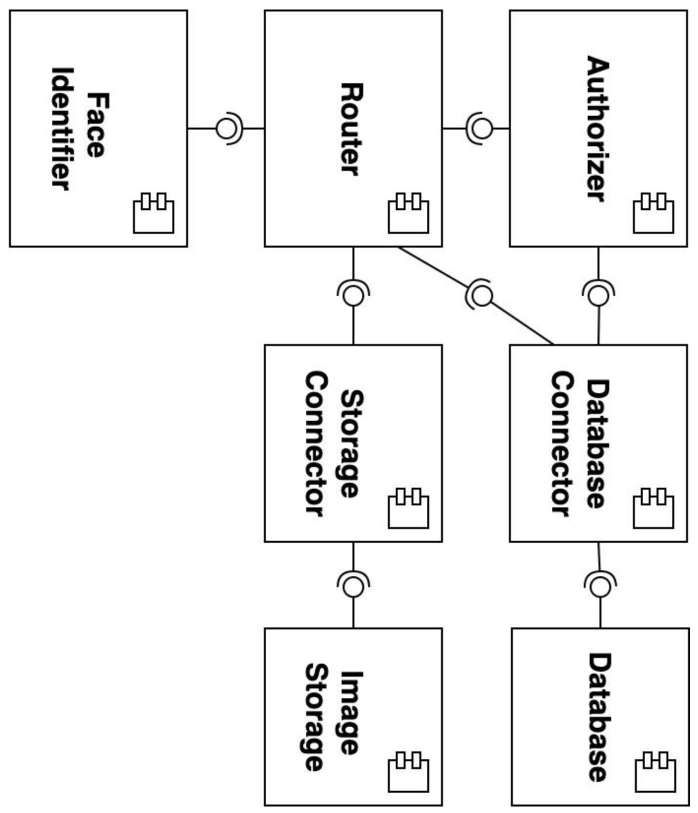


13.5 Layered design with segmented layers

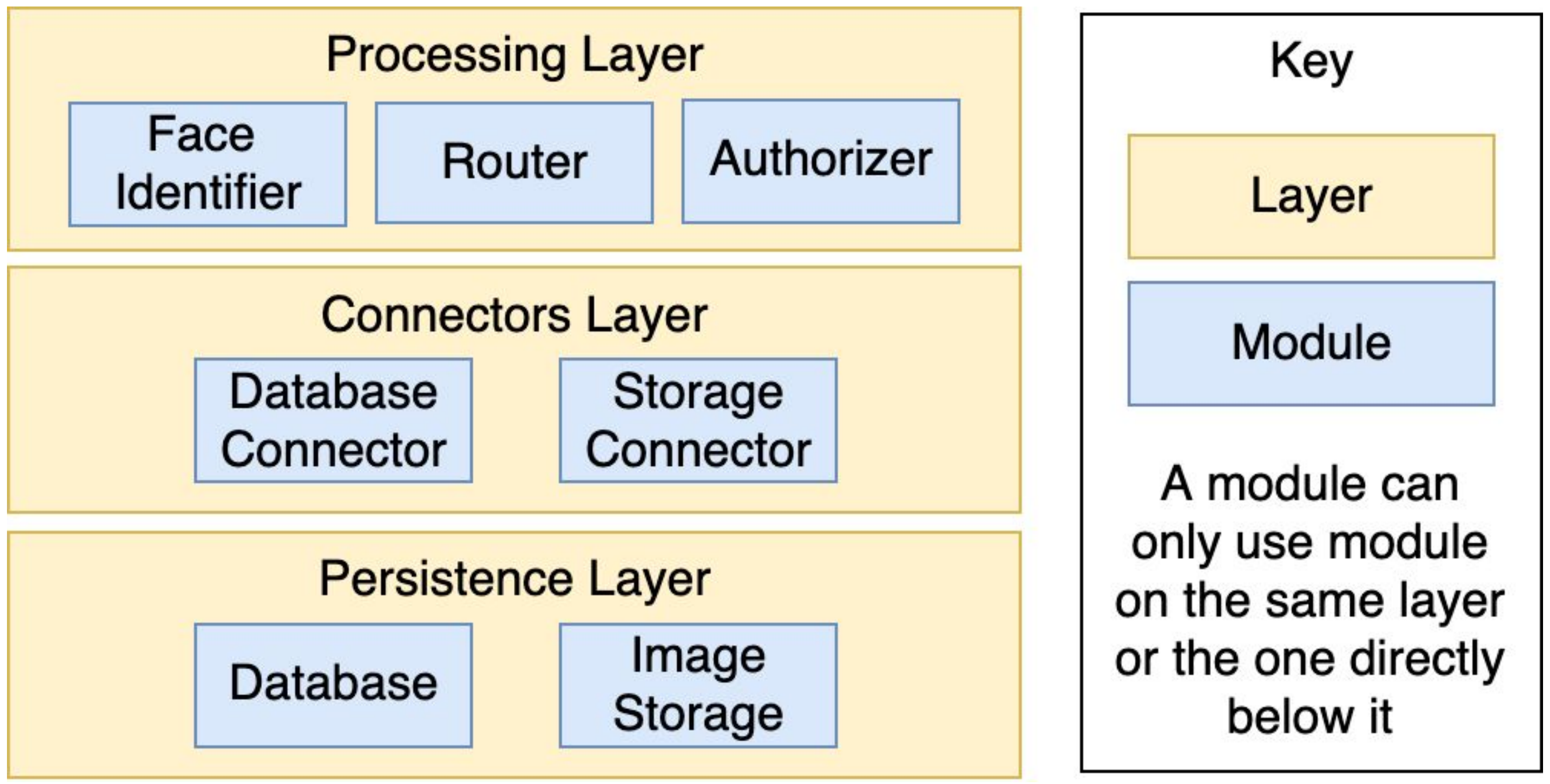


Layers with a "sidecar"

Can we convert to layered?



Converting into Layered Pattern

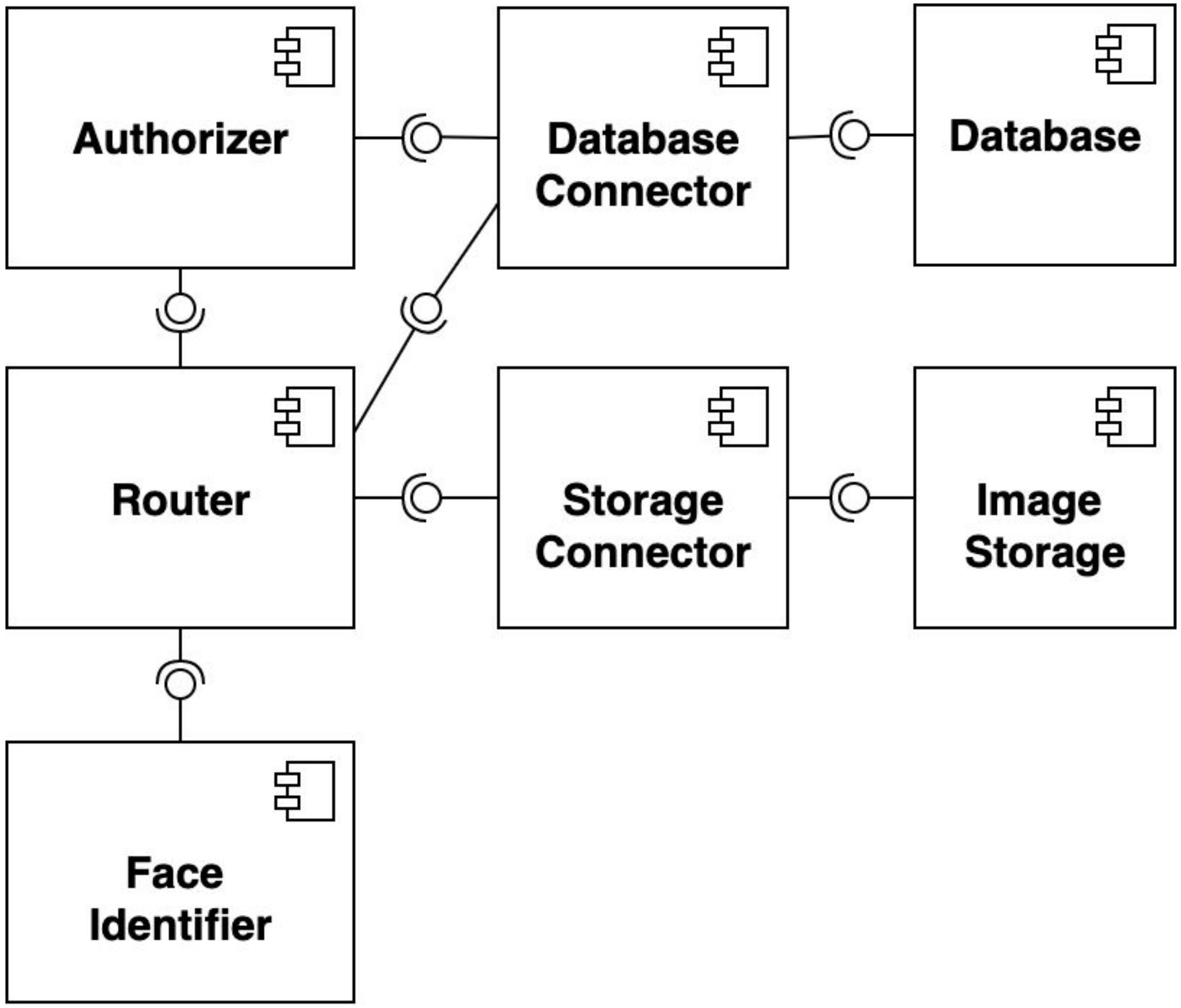


Maintainability is an attribute we reason about in the static view.

Maintainability is an attribute we reason about in the static view.

Limitations of Static View

- What meanings does the static view convey?
- What you cannot describe using it?

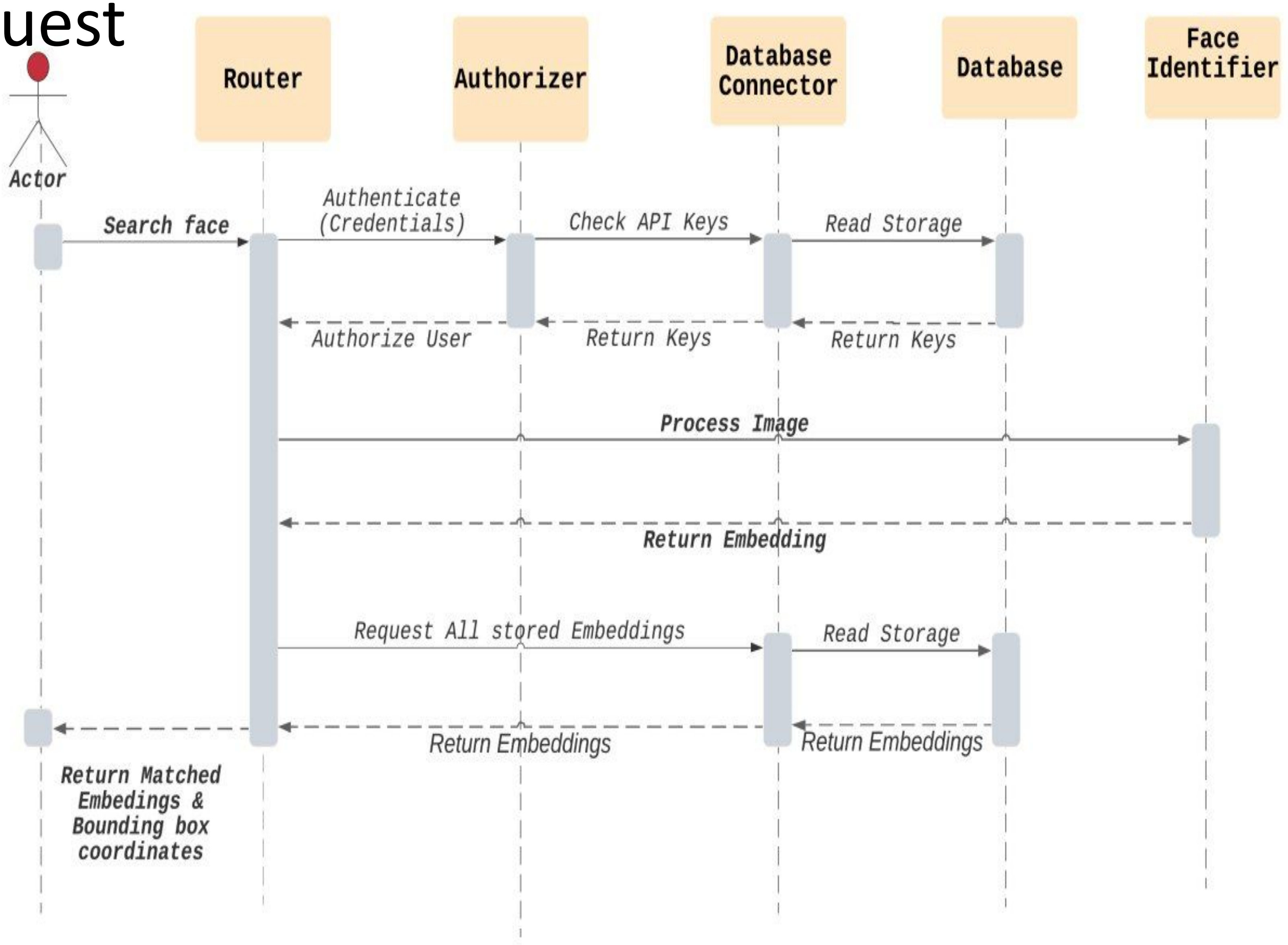


Quality: Scalability

From what perspectives/views can we reason about it?

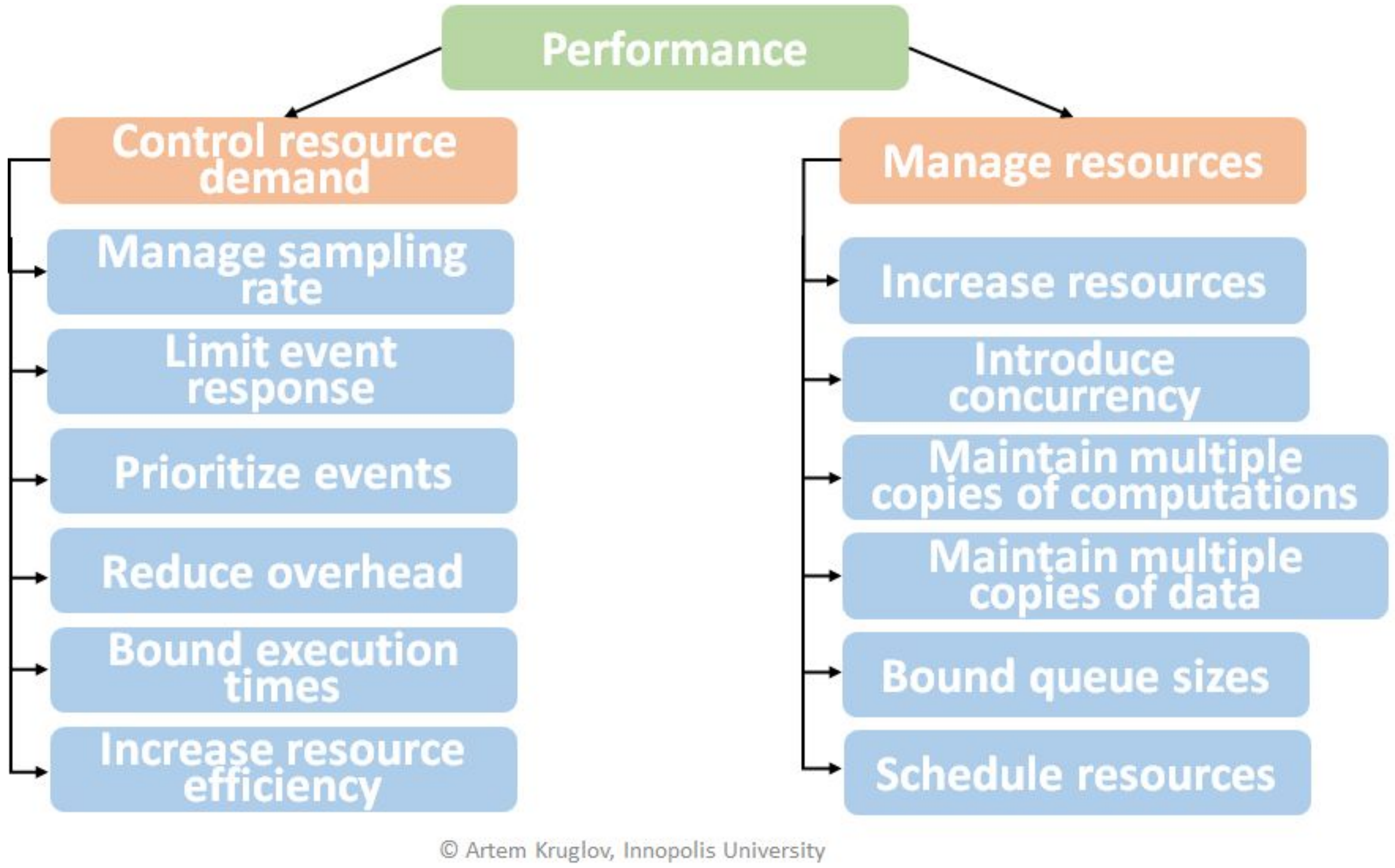
Dynamic Perspective:
UML Sequence Diagram

UML Sequence Diagram: “Search gallery” request



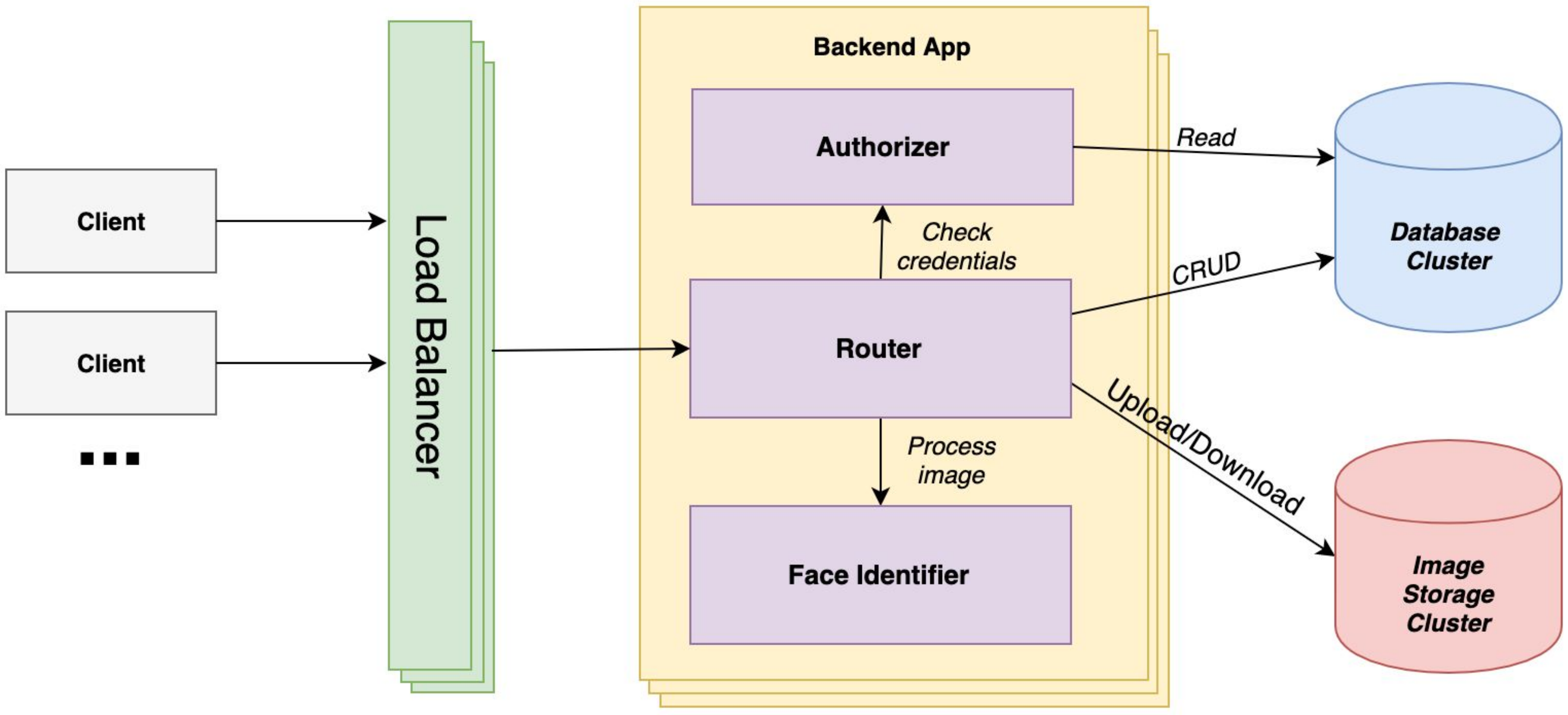
Quality Attribute	**Ideas how to achieve**
Q2. **Attribute:** Efficiency (Scalability) **Scenario:** In production time, the customer demand (# of requests) goes up/down. The system is able to adjust to the demand and increase/decrease the resources with no downtime and no delays in response time.	Put the service into a docker and use some automatically scalable platform to deploy it on (such as Google Cloud Run or AWS). For database and storage use an automatically scalable solution (such as Firestore, AWS, or Google Cloud Storage)

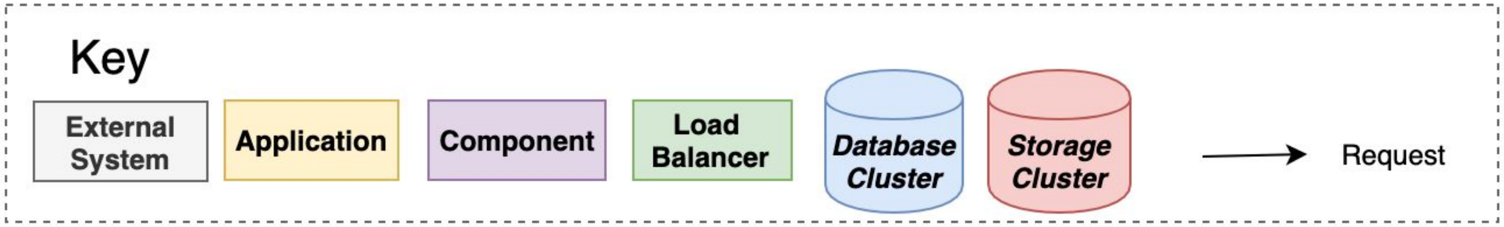
Performance Tactics



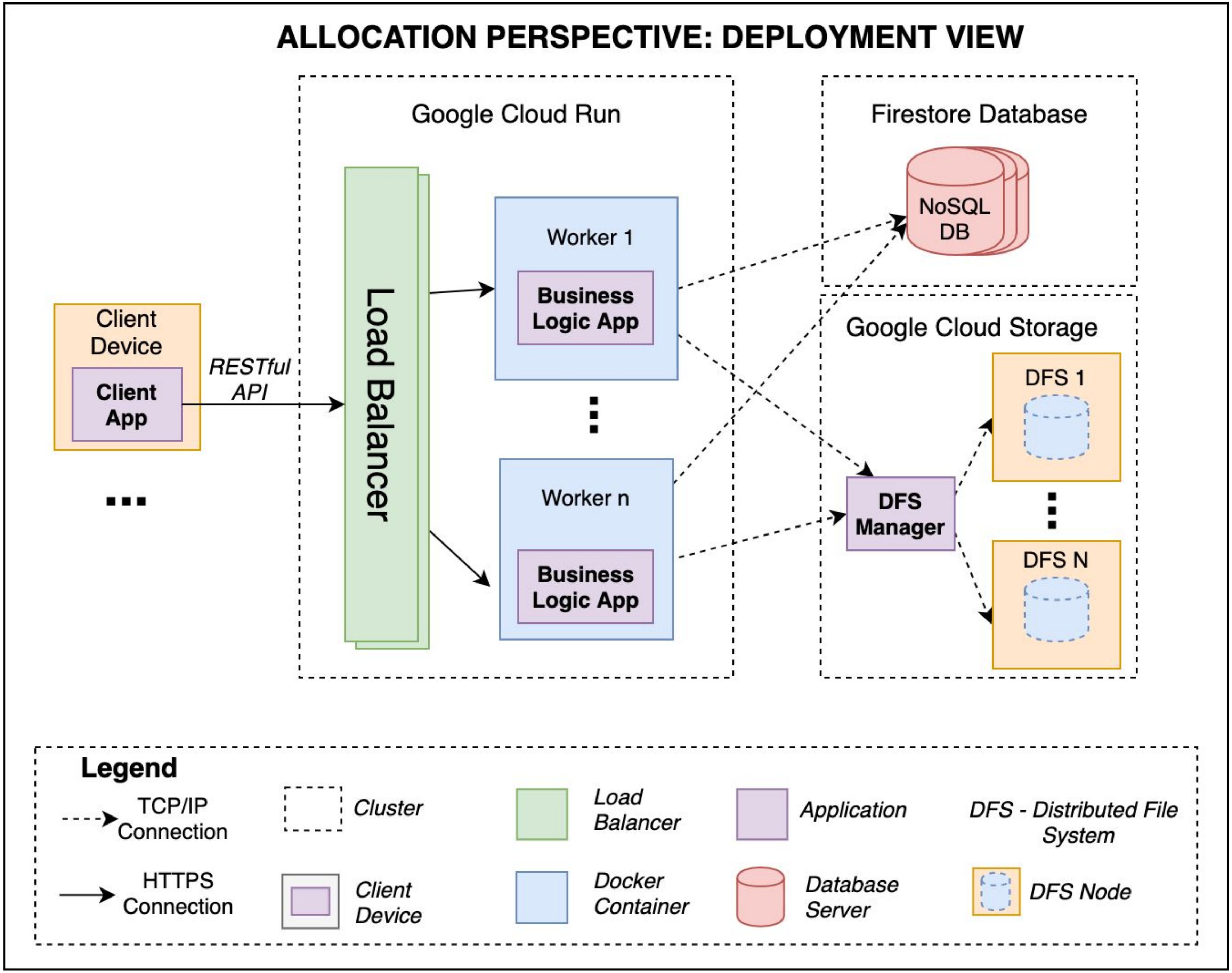
Multi-tiered Pattern

DYNAMIC PERSPECTIVE





Allocation Perspective: Deployment View


ALLOCATION PERSPECTIVE: DEPLOYMENT VIEW
Google Cloud Run
Firestore Database
NoSQL DB
Google Cloud Storage
Client Device
Client App
RESTful API
Load Balancer
Worker 1
Business Logic App
Worker n
Business Logic App
DFS Manager
DFS 1
DFS N
Legend
TCP/IP Connection
HTTPS Connection
Cluster
Client Device
Load Balancer
Docker Container
Application
Database Server
DFS - Distributed File System
DFS Node

Documenting architecture

What we can use:

UML - conventional.

Custom charts - provide legend.

Both require prose commentary.

Why we document:

To design, reason and make decisions.

To communicate.

Documenting architecture

What we can use:

UML - conventional.

Custom charts - provide legend.

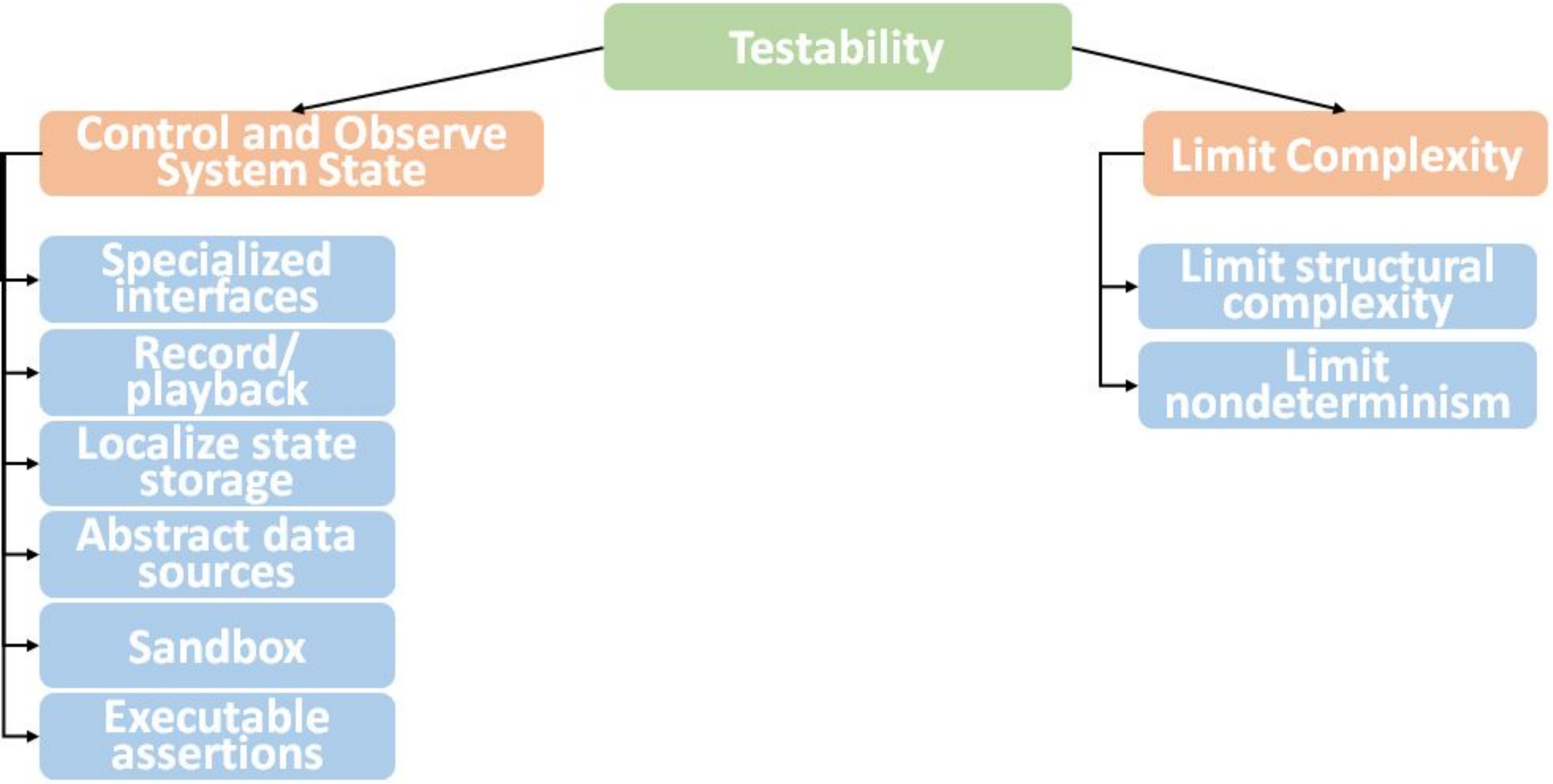
Both require prose commentary.

Why we document:

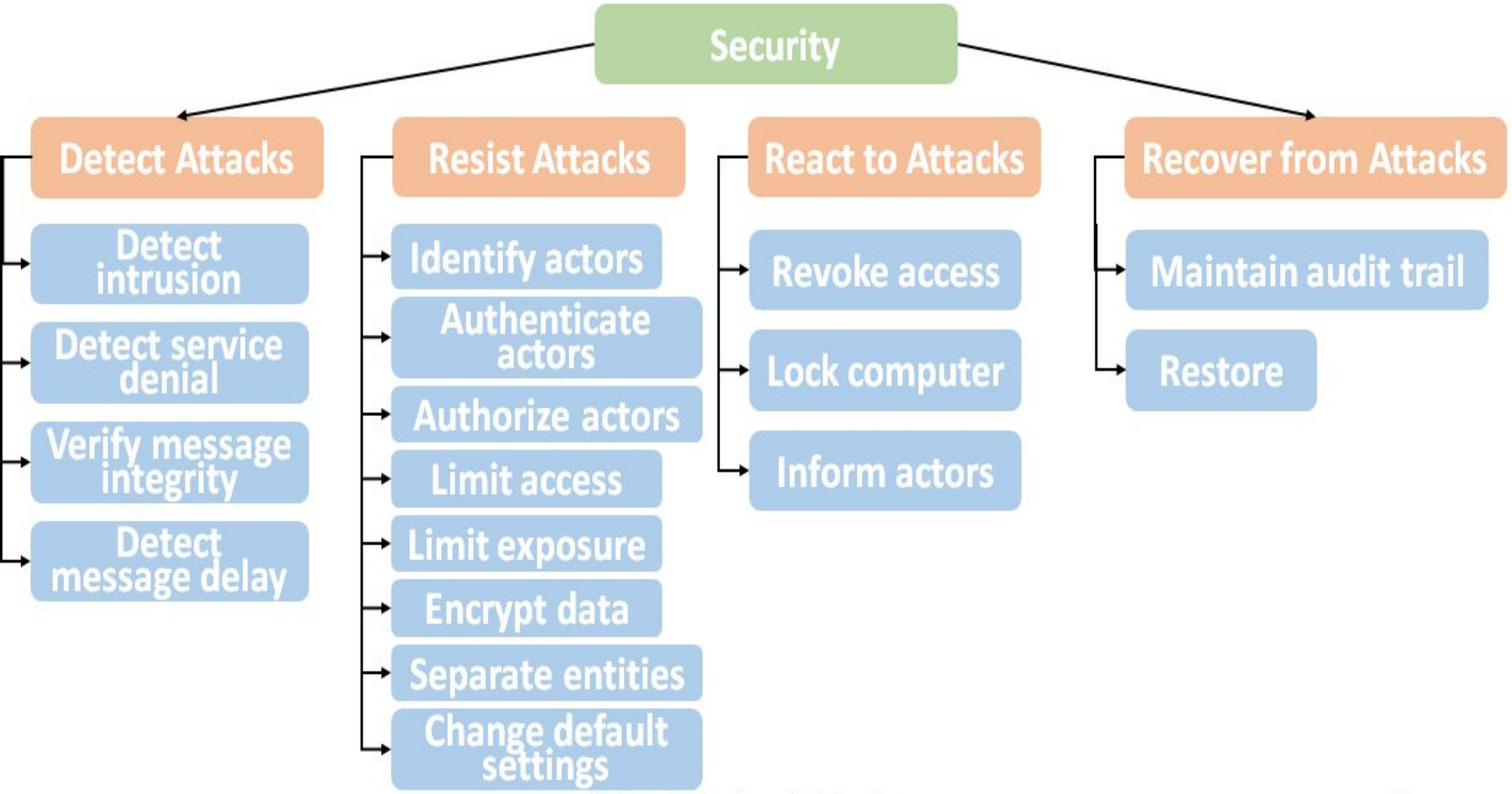
To design, reason and make decisions.

To communicate.

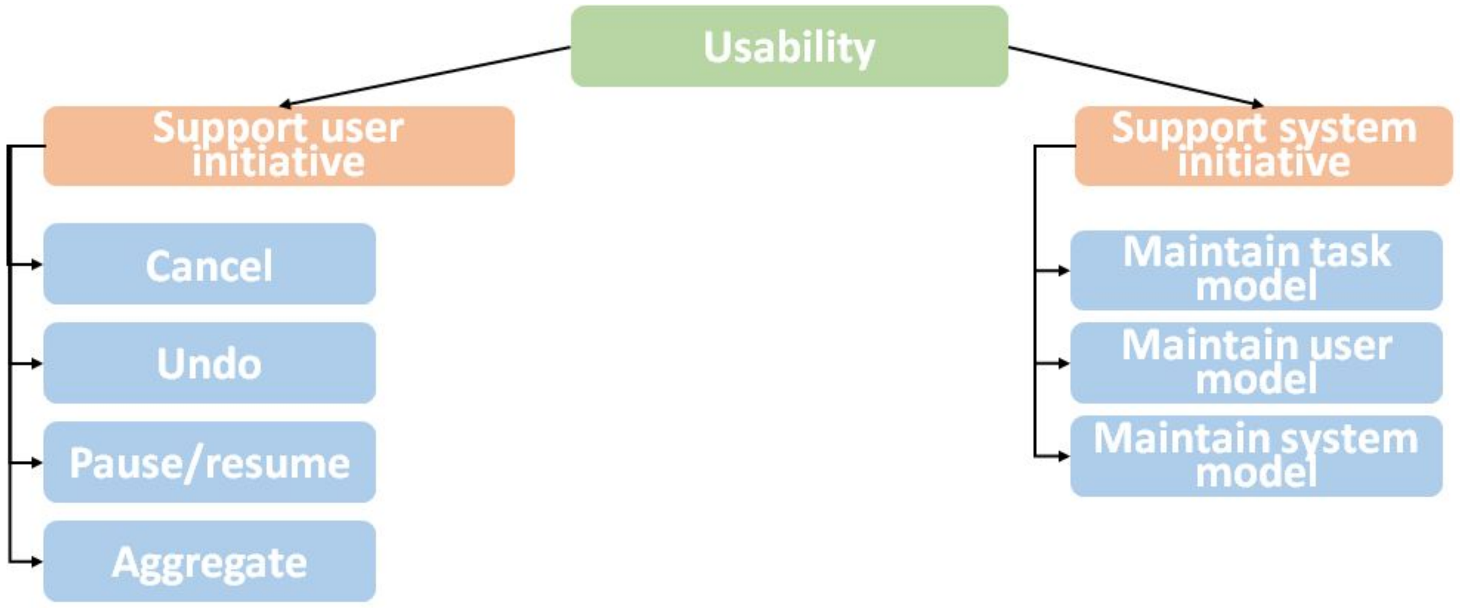
Testability Tactics



Security Tactics

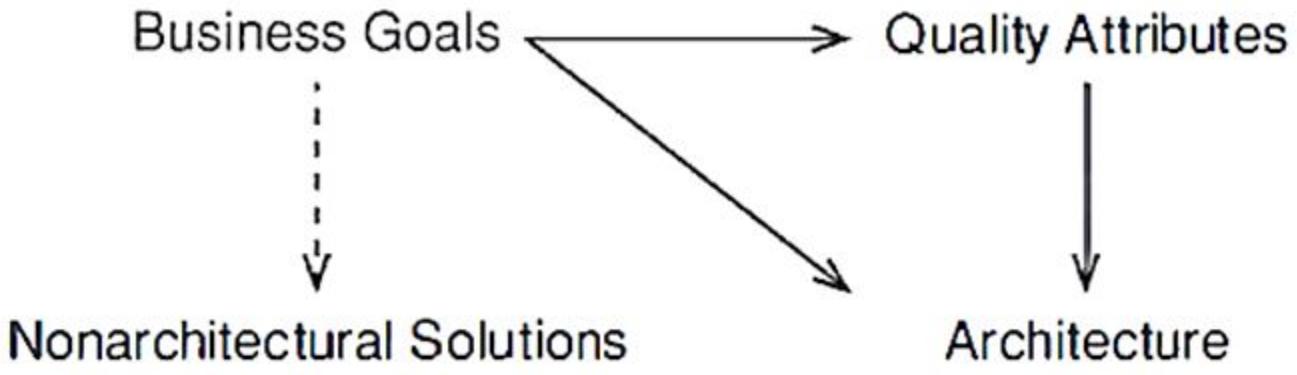


Usability Tactics



Business Goals and Quality Attributes

Business goals and their implied quality concerns play a significant role in the creation of an architecture for a given system



What have we learned today?