Software Architecture Styles and Patterns

CS6.401 Software Engieneering

Karthik Vaidhyanathan
https://karthikvaidhyanathan.com

Software Architecture

The Software Architecture is the **earliest model** of the **whole software system** created along the software lifecycle

- A set of components and connectors communicating through interface
- A set of architecture design decisions
- Focus on set of views and viewpoints
- Developed according to **architectural styles**

Architectural Styles

Classic Architecture

Romanesque Architecture

Gothic Architecture

Dravidian Architecture

Software Architecture Styles

Set of **design rules** that identify the **kinds of** components and connectors that may be used to compose a system or subsystem, together with **local or global constraints** on the way the composition is done.

[Shaw and Clements, 1996]

.

Architectural Patterns

1. Collection of design decisions found in practice

2. Has known properties that permit reuse

3. Describes a class of architecture

One does not invent patterns, one discovers them – They are found in practice

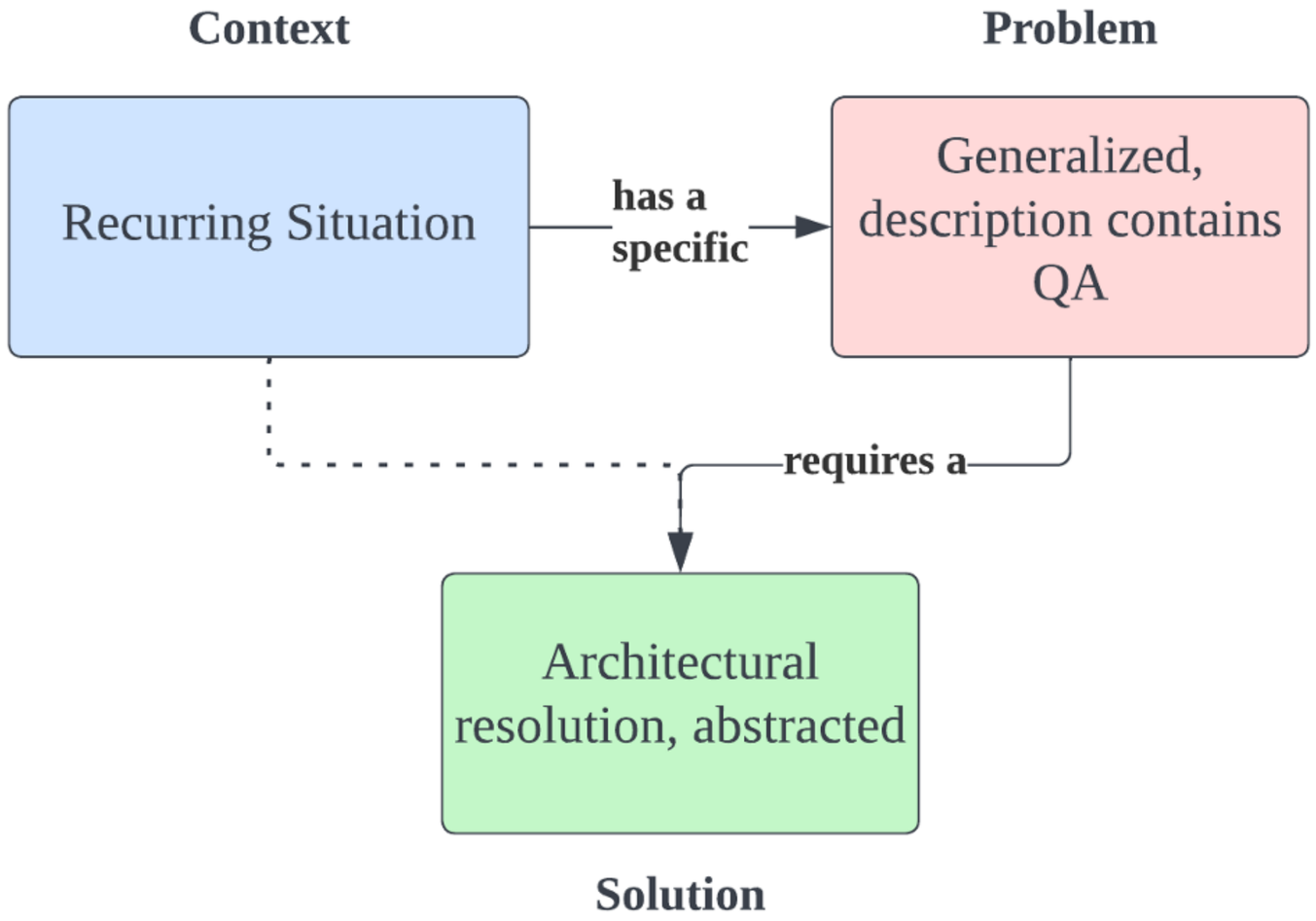
There is never a complete list of patterns

.

Architectural Patterns

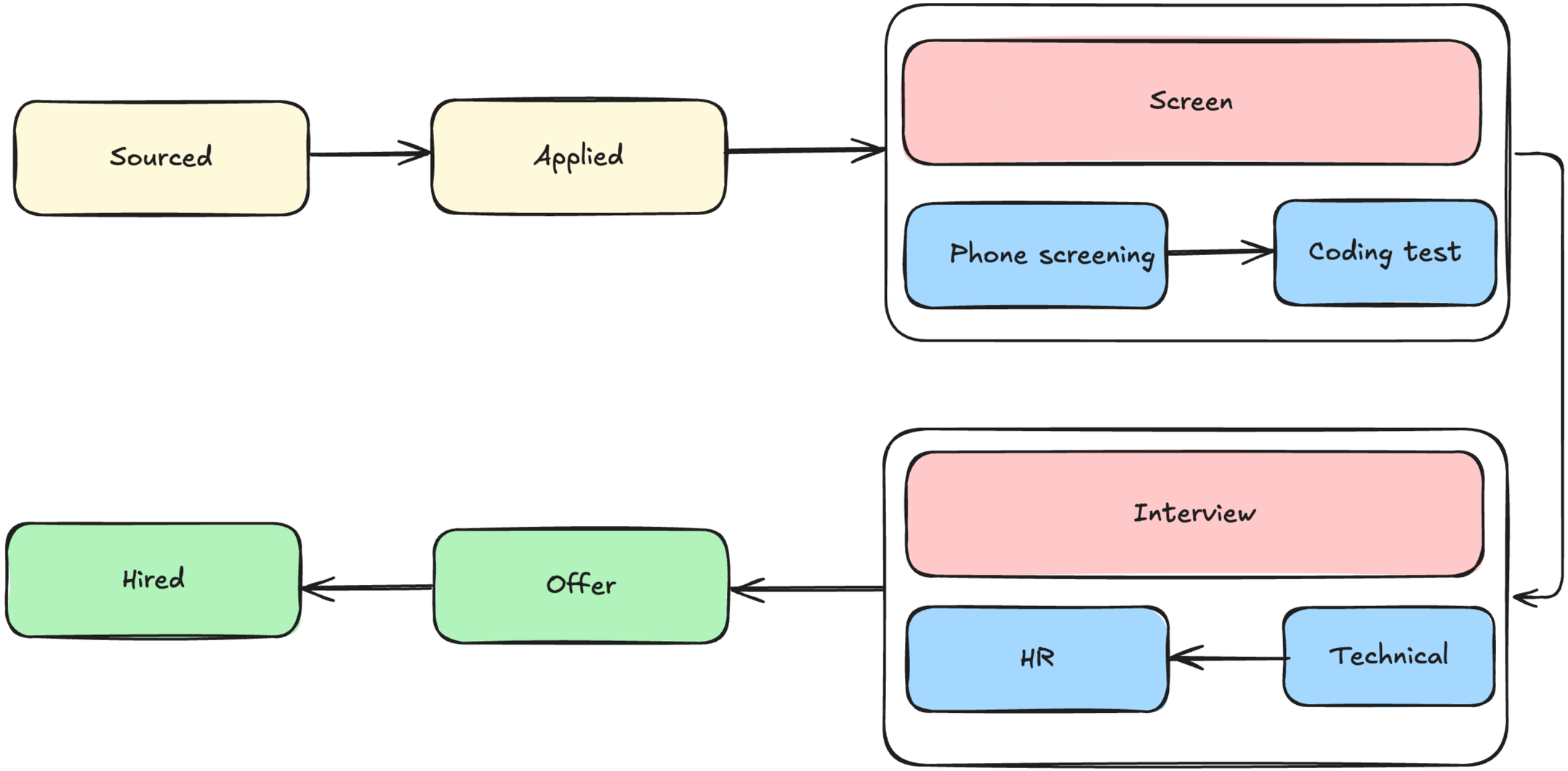
- The architectural patterns is determined and described by:
  - Set of components
  - Set of connectors
  - Topological layout of components
  - A set of semantic constraints on topology, components, their behaviour and interaction mechanisms

Architectural Patterns



Pattern documentation template: {*context, problem, solution*}

The Pipe and Filter Pattern: Intuition



Pipe and Filter Pattern

Context

Many systems required to transform discrete stream of data. Occur repeatedly in practice

Problem

Reusable, loosely coupled components, simple and generic interactions

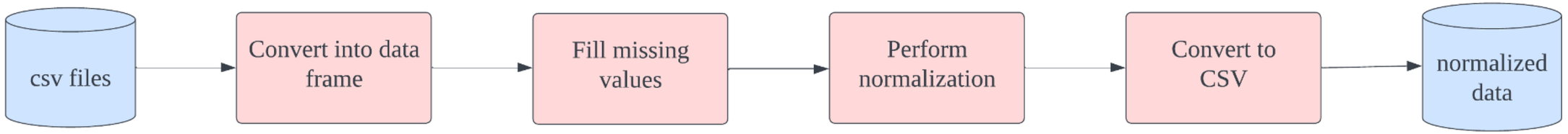
flexibility, resuability

Divide into pipes and filters, pipes transport, filters process/transform

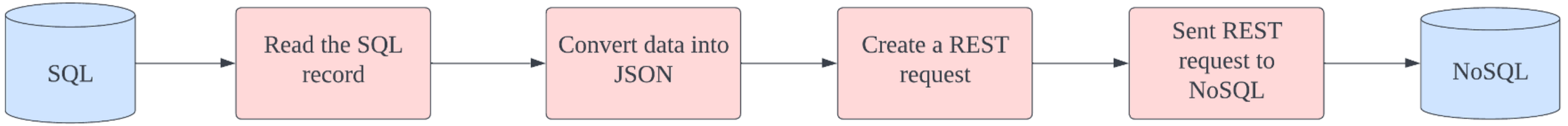
Solution

Pipe and Filter Pattern: Some Use Cases

- Data Preparation for ML



- Data Migration



Pipe and Filter Pattern

Filter (Component)

- Transforms data from input to output
- Can execute concurrently, incrementally transform

Pipe (Connector)

- Single source for input, single target for output
- Does not alter data passing through pipe

The Pipe and Filter Pattern

Constraints

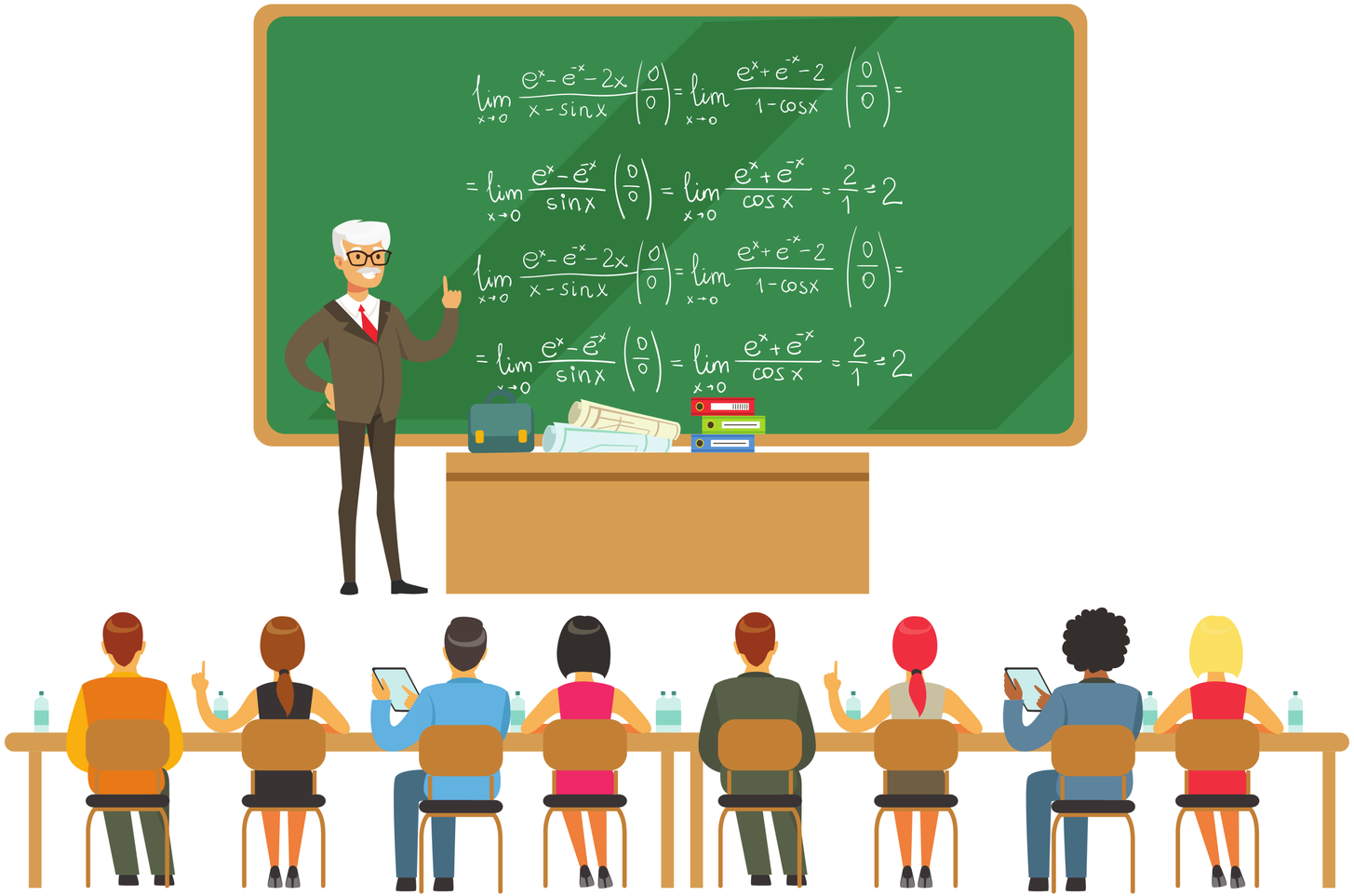
- Pipes connect the filter output to the filter input
- Filters must agree on the type of data being passed from the pipe
- Specialization is more like a linear sequence of actions => Pipelines

Weakness

- Not good for an interactive system
- A large number of filters can add substantial overhead
- Not suitable for long-running jobs

Blackboard Pattern: Intuition

Who wants to solve?



Blackboard Pattern

Context

Open problem domain with various partial solutions

Problem

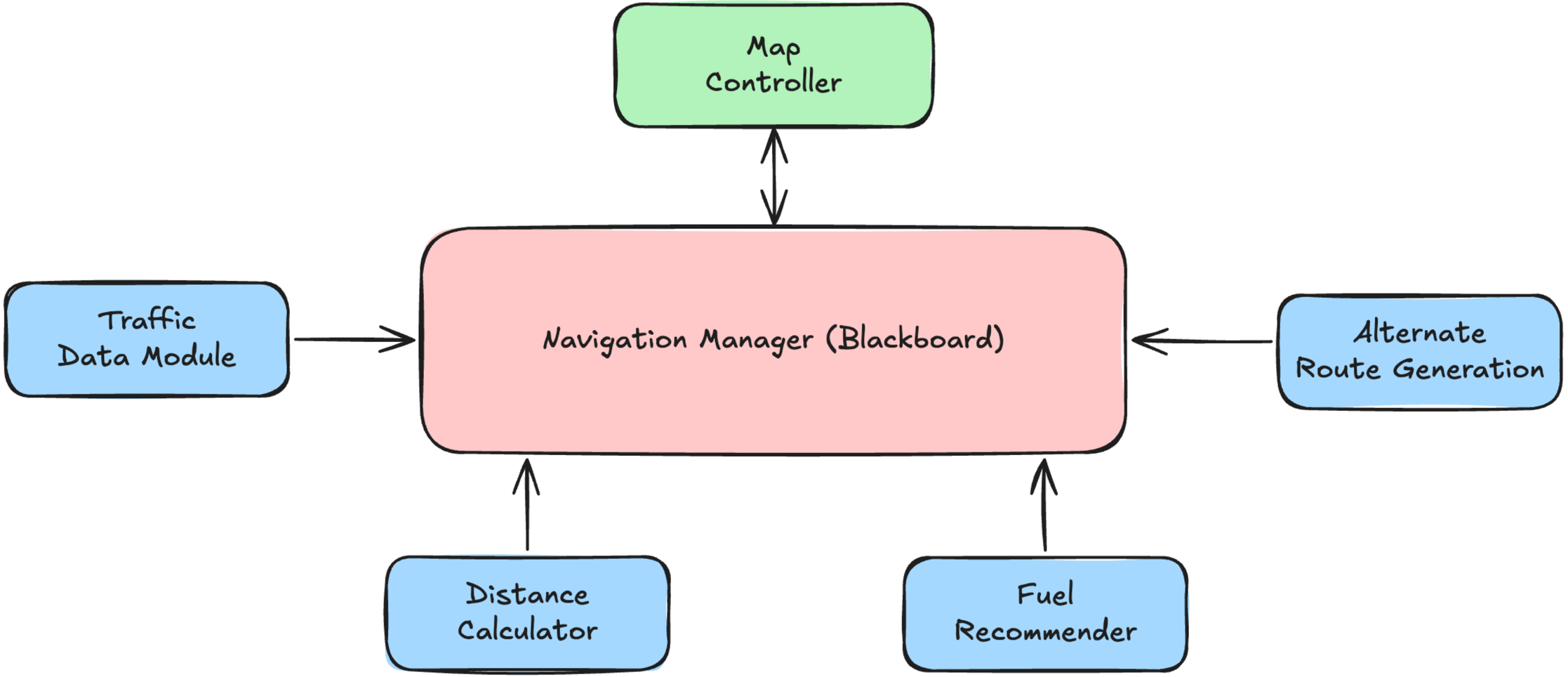
The partial solutions needs to be integrated

Flexibility, Maintainability,

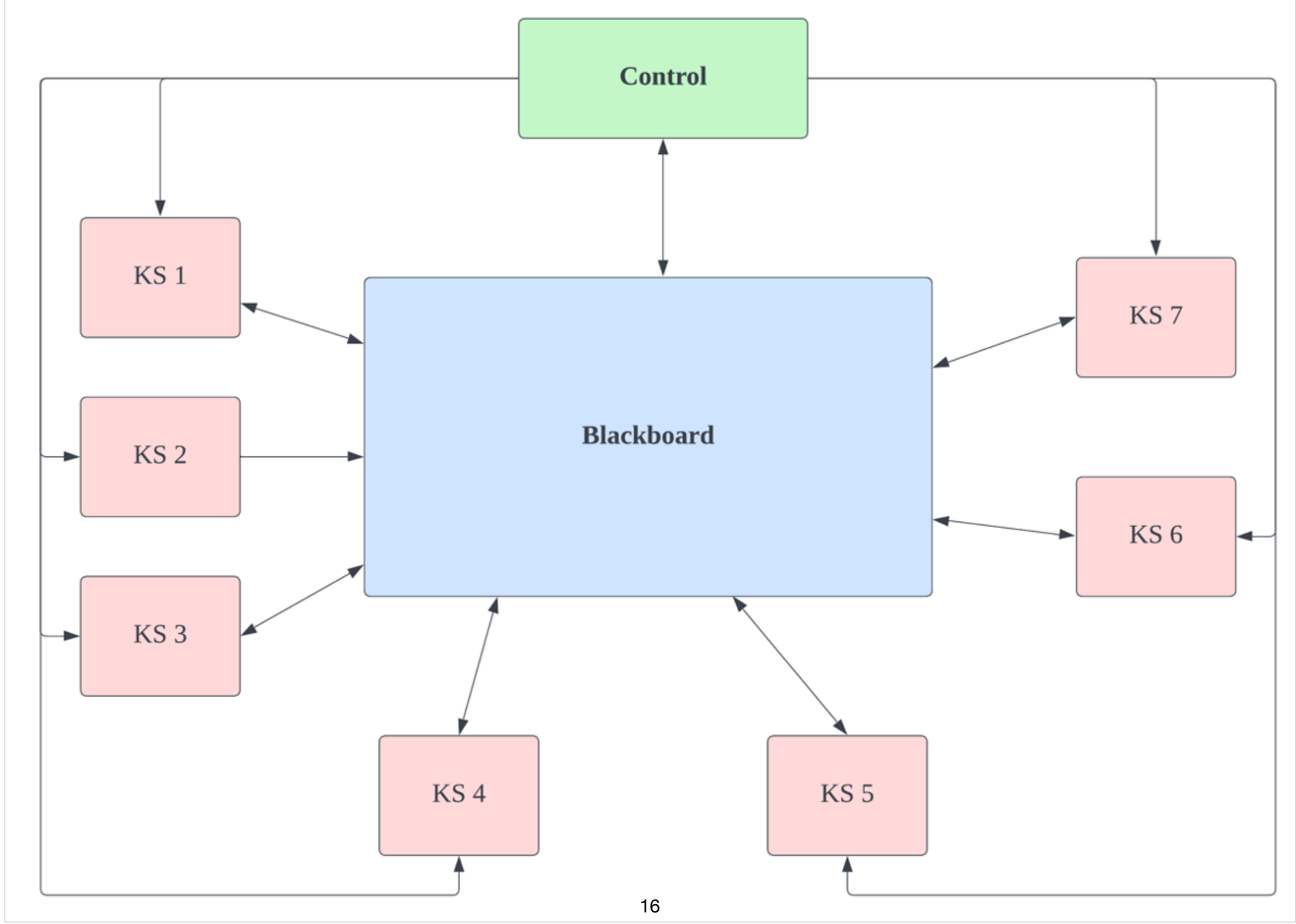
Decompose the software into blackboard, knowledge source and control

Solution

Blackboard Pattern: Use case



Blackboard Pattern



The Blackboard Pattern

Architectural Elements

Blackboard

- Global repository containing input data and partial solutions

Knowledge Sources (KS)

- Separate and independent components
- Contains the knowledge required to solve the problem

Controller

- Component managing course of problem solving (eg: manage KS)

Relation: Attachment relation (KS's attached to the blackboard)

The Blackboard Pattern

Constraints

- No direct communication among the KS
- Any interaction happens via the blackboard

Weakness

- Blackboard can become a bottleneck (too many KS)
- Difficult to determine the partitioning of knowledge
- Control can be very complex

.

Publish-Subscribe Pattern Intuition

Youtube Subscription

Newspaper Subscription

Publish-Subscribe Pattern

Context

Number of independant producers and consumers that must interact. The number or nature of data is not fixed

Problem

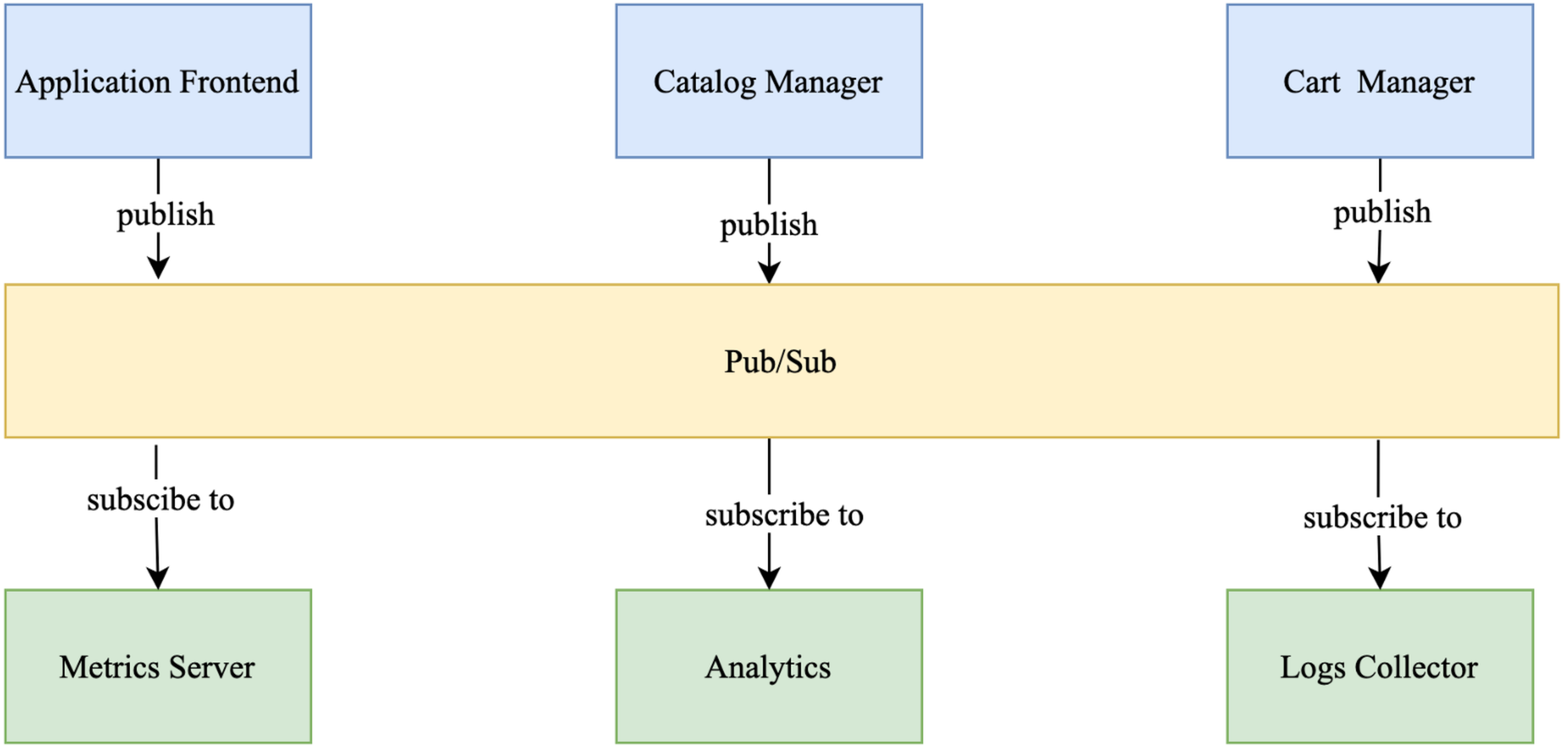
How to create integration mechanisms that support transmission without coupling

scalability, manageability

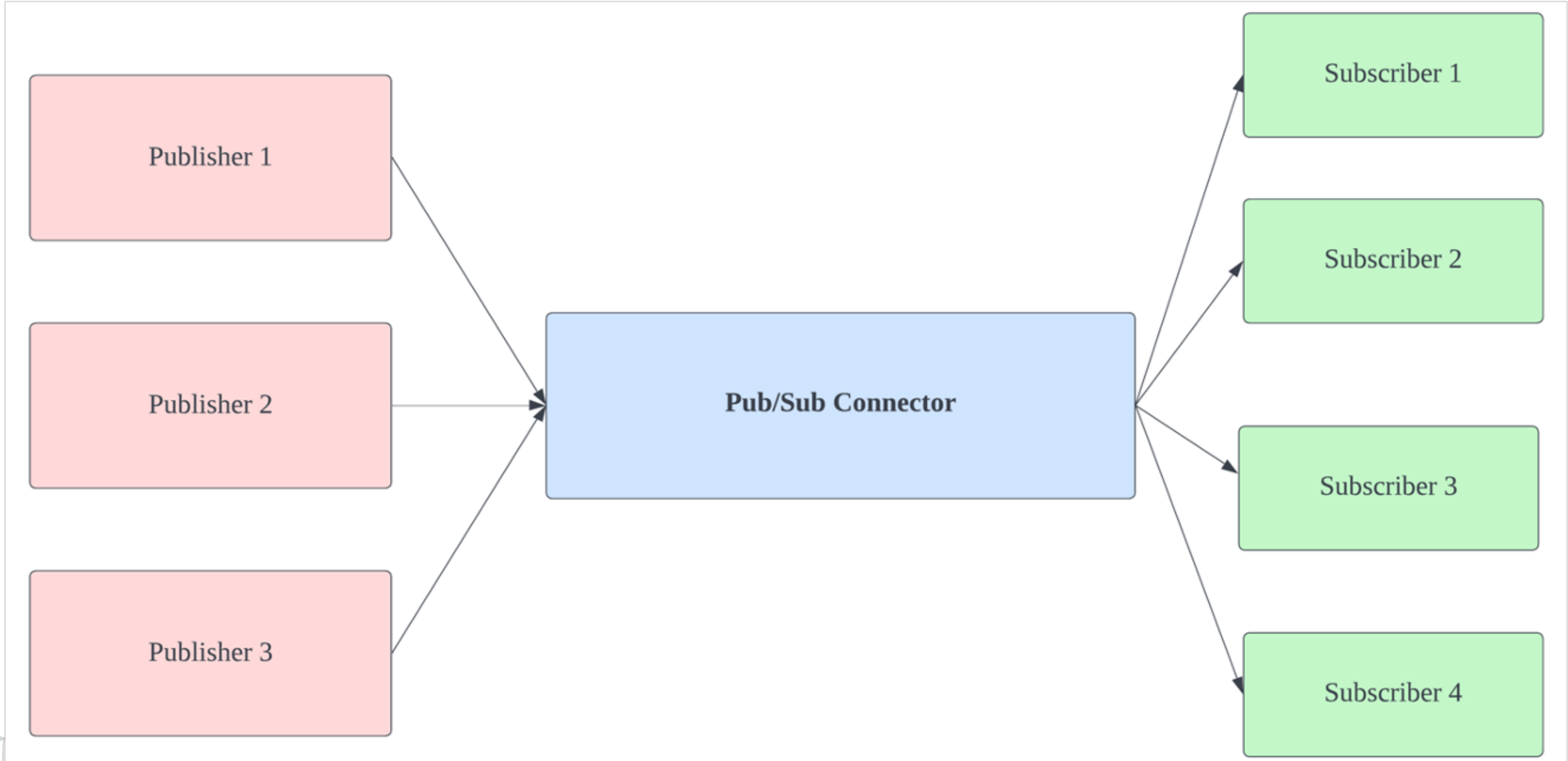
Publishers publish information which can be subscribed to by the subscribers. Have connectors to manage

Solution

Publish Subscribe: Use Case



Publish-Subscribe Pattern



Publish-Subscribe Pattern

Architectural Elements

- Publisher

Components that produces messages/events

- Subscriber

Components that consume the messages/events produced by publisher

- Pub-Sub Connector

Component that has *announce and listen* roles for publishers and subscribers

Relation: Attachment relation associates pub/sub components with the connectors

Publish-Subscribe Pattern

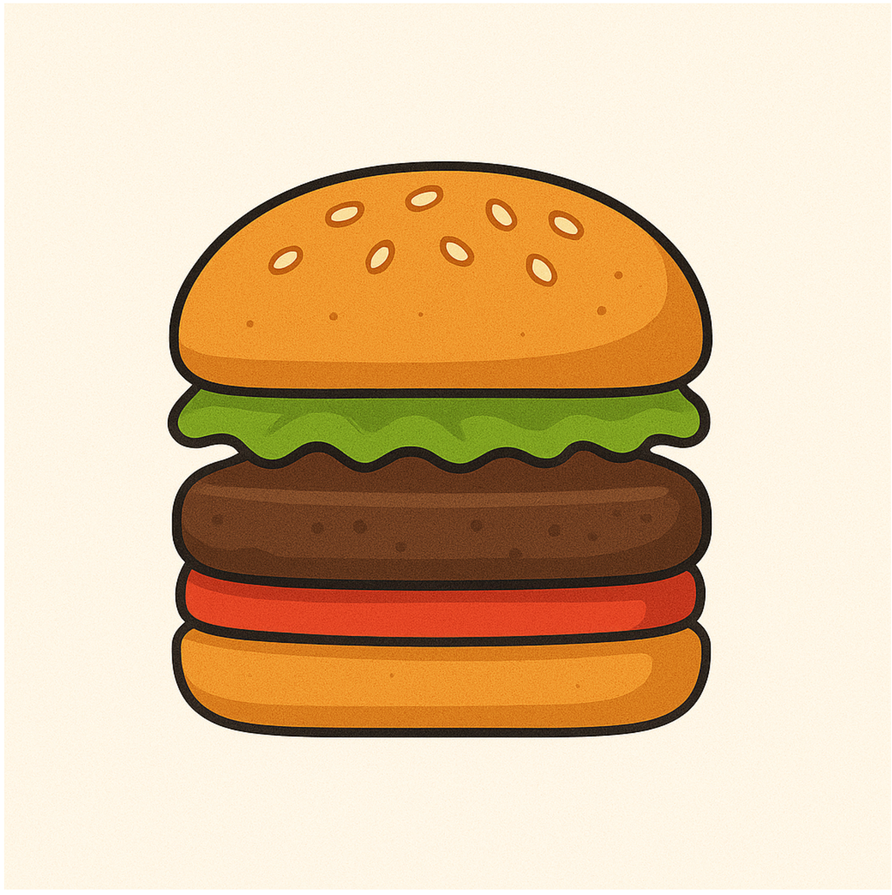
Constraints

- All components are connected to a connector (bus or a component)
- Restrictions on which component can listen to what
- A component may be both a publisher and a subscriber

Weakness

- May increase latency
- Can have a negative impact on the predictability of message delivery time
- Less control on the ordering of messages
- Delivery of the message is not guaranteed

Layered Architectural Pattern: Intuition



Layered Architectural Pattern

Context

Develop and evolove portions of systems independently. Promote seperation of concerns.

Problem

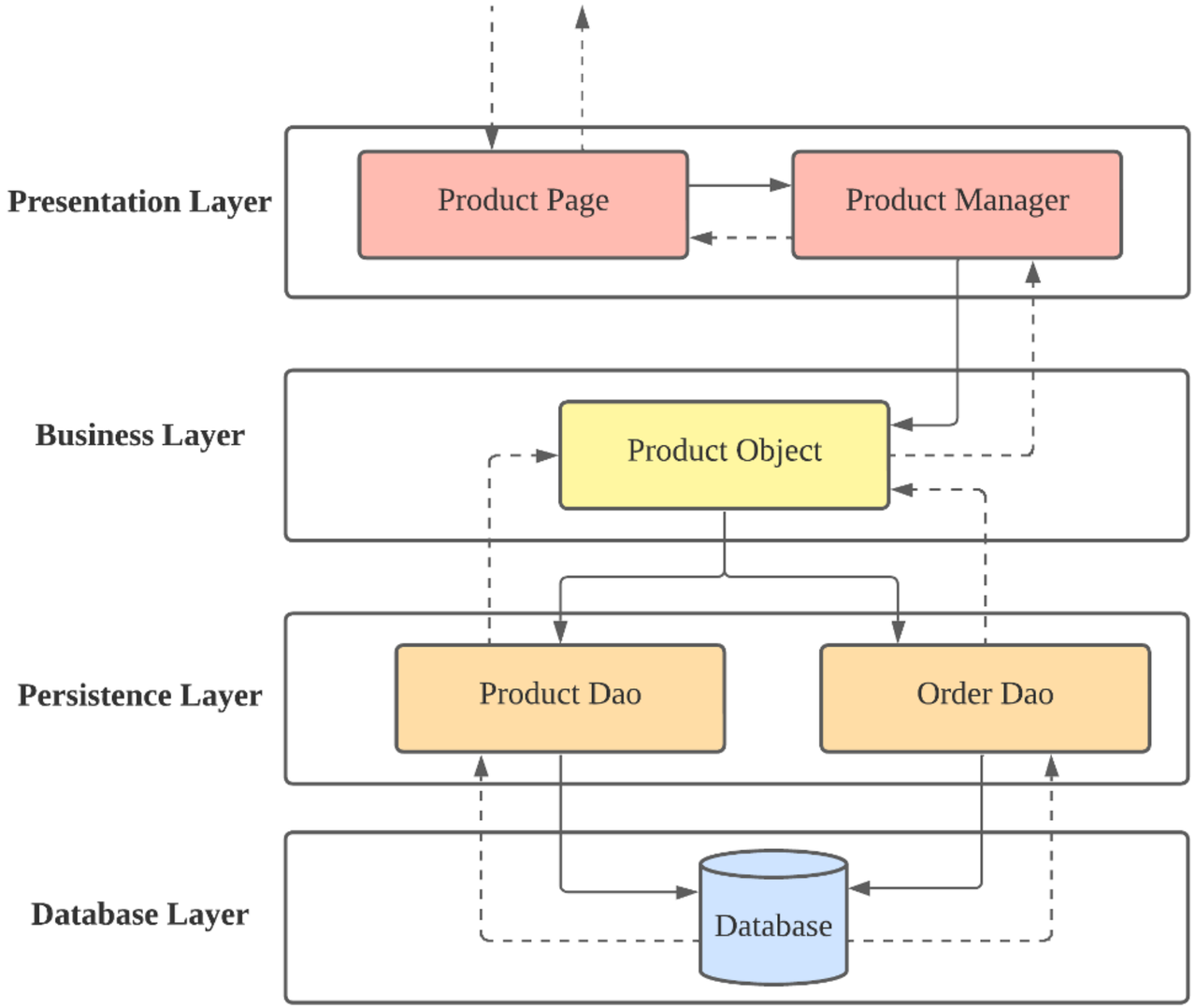
Modules can be developed and evolved seperately with little interaction

modifiability, portability, reuse

Divide the software into units called layers. Each layer is a grouping of modules

Solution

Layered Architectural Pattern: Use Case



Layered Architectural Pattern

Architectural Elements

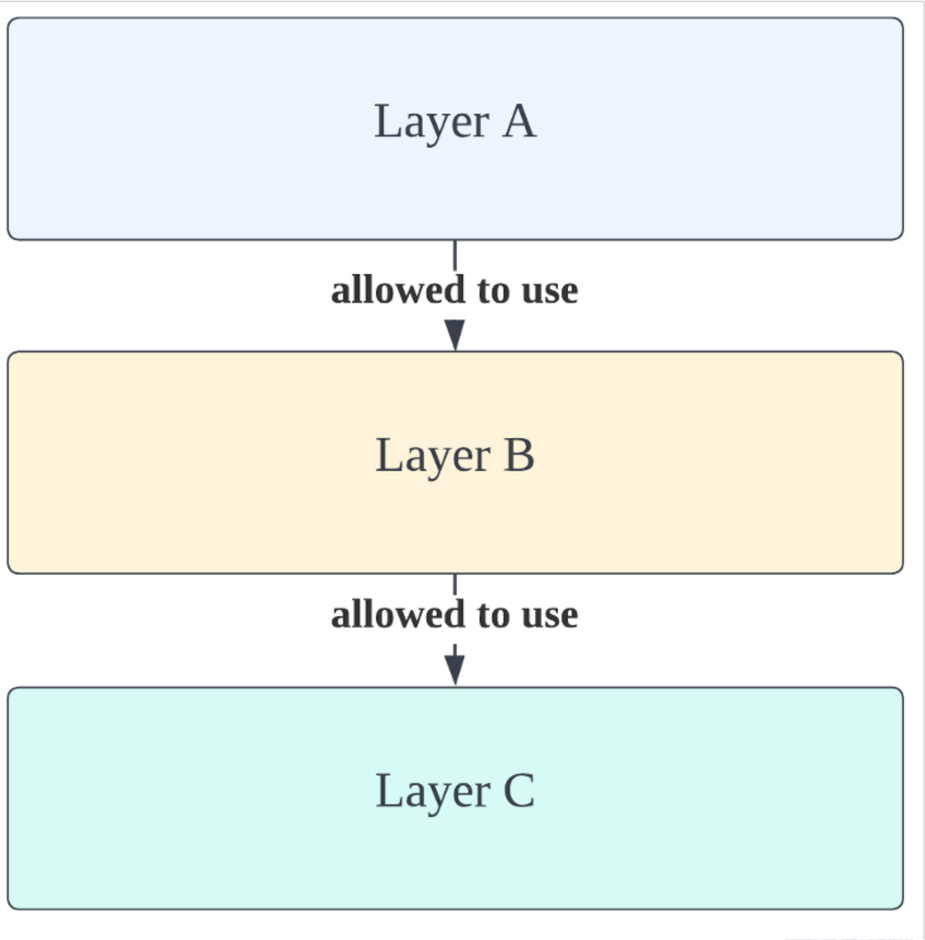
Layer

- Kind of a module
- Description should define the what modules it can contain

Relation

- Allowed to use

The design should always define theusage rules



Layered Architectural Pattern

Constraints

- Every piece of software is exactly allocated to one layer
- There are at least two layers (often more!)
- Allowed to use relations should be acyclic

Weakness

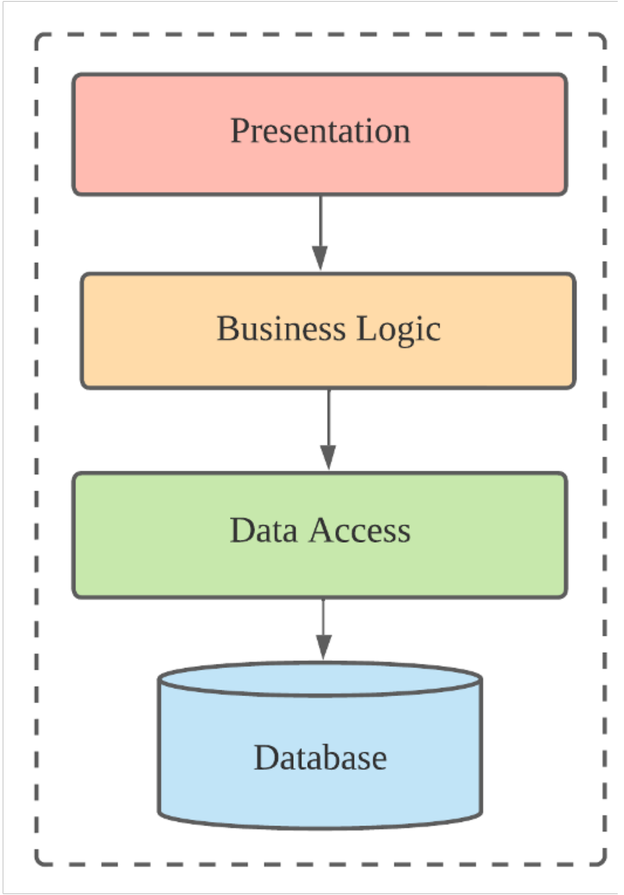
- The addition of layers adds up-front cost and complexity
- Performance bottlenecks

Layered Architectural Pattern - Some Issues

One of the most commonly used patterns – still people get it wrong!

- Define proper relations with key (which layer can use what)
- Stack of boxes lined up does not belong to layered
- A layer isn't allowed to use any layer above it.

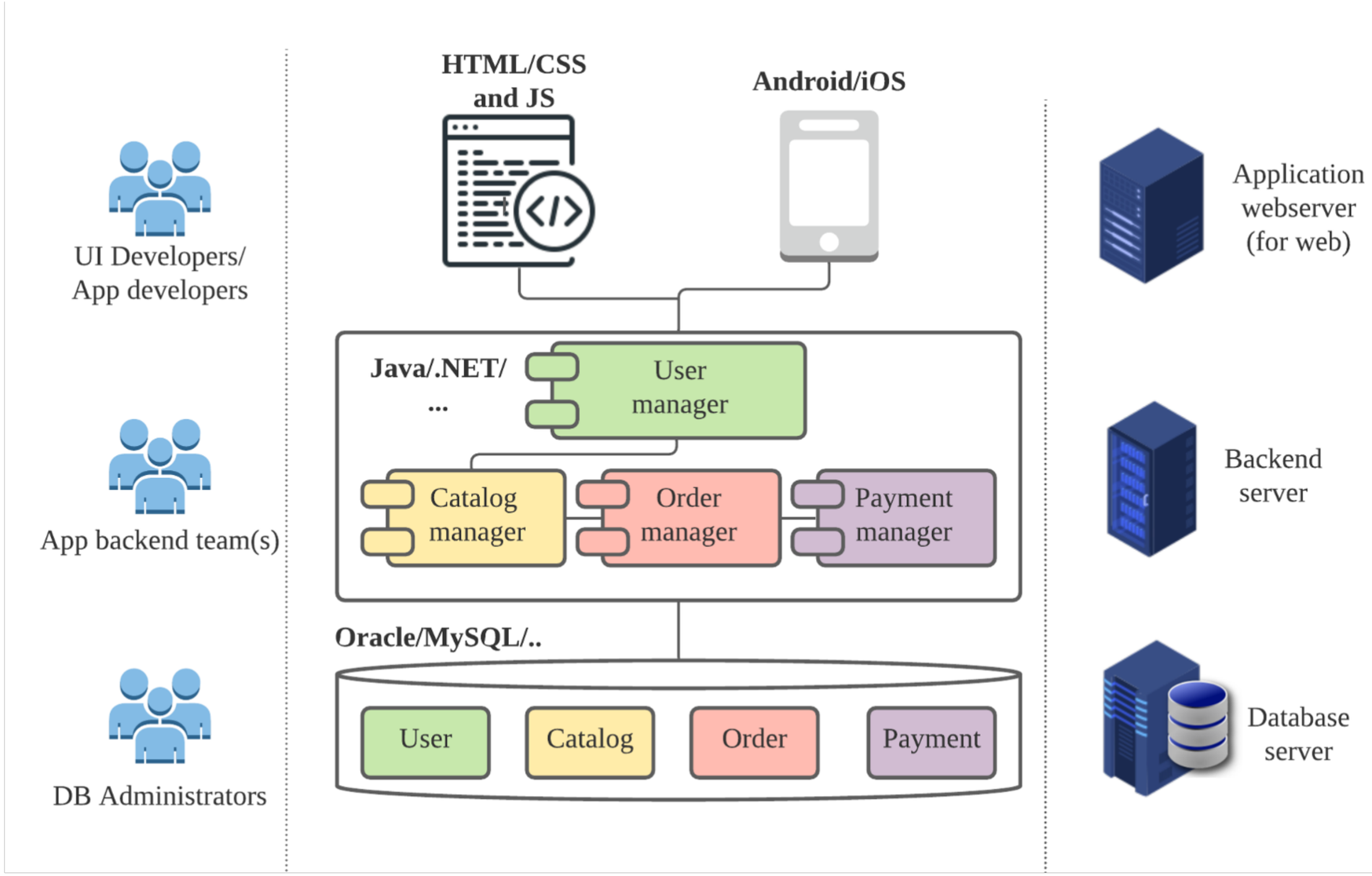
.



Brief Look into Monoliths

One .py or .java file for the entire system

Monolithic Approach to E-Commerce



Monolithic Approach: Some Pitfalls

- High degree of **coupling** - everyone needs to know everything !!!
- Change cycle and bug fix can take weeks - **Modifiability** and time to market
- Adding a new feature can be challenging - **Extensibility**
- Separation of concerns via components with inherent coupling - **Modularity**
- Scaling system implies scaling the whole stack - **Scalability**
- Limited by the language of choice - eg: add recommendation feature to e-commerce (Java or Python ?)
- The database is centralized - addition or modification is a costly process

Monolith has its own advantages, too!

Service-Oriented Architecture

SOA

Context

A number of services offered by service providers and consumed by service consumers. Service consumer should be able to use services without knowing detailed implementation

Problem

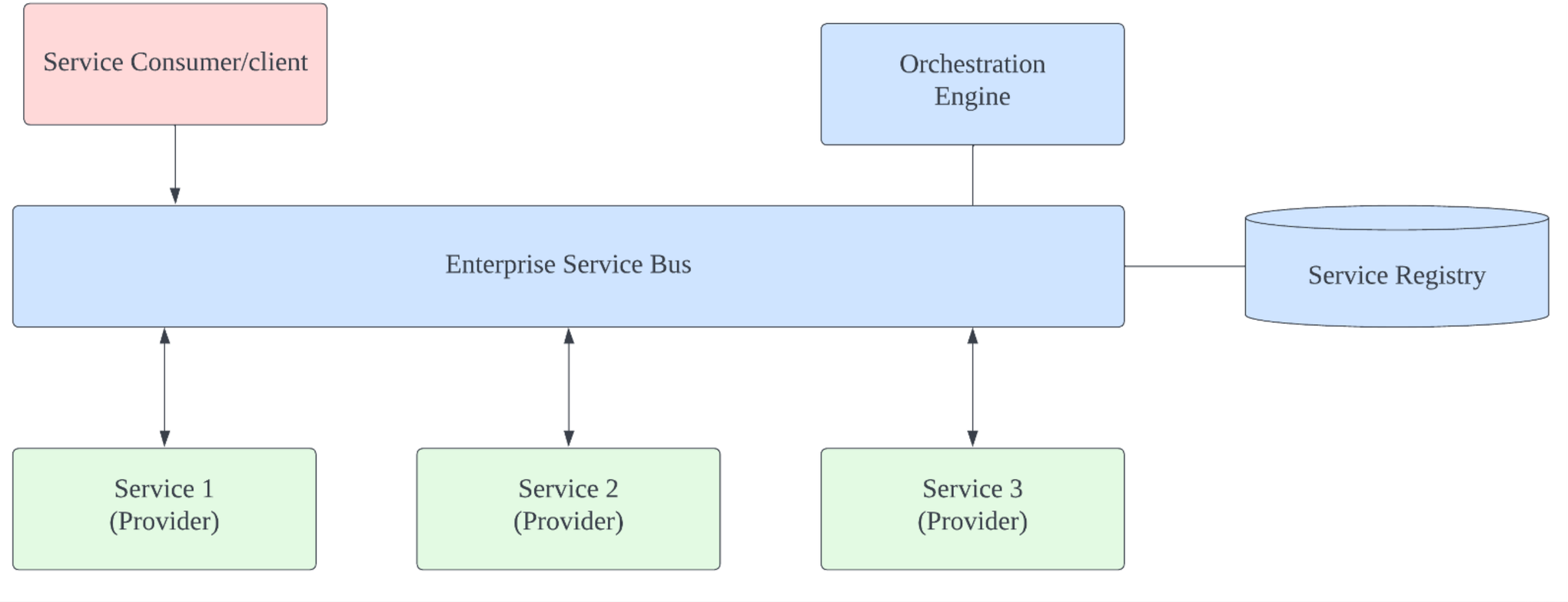
How to provide support for interoperability among different components running in different platforms implemented in different languages

availability, performance, security

Collection of loosely coupled services with clearly defined interfaces. Can be implemented in different languages. Supports commmunication between and to/from services

Solution

SOA



SOA - Architectural Elements

Components

Service Providers: Components that provide 1 or more services through defined interfaces

Service Consumers: Invoke services directly or through intermediary

ESB: Intermediary component that can route and transform messages

Service Registry: Providers can register services, consumers can discvoer services

Orchestration Server: Coordinates interaction between consumers and providers based on languages

.

SOA - Architectural Elements

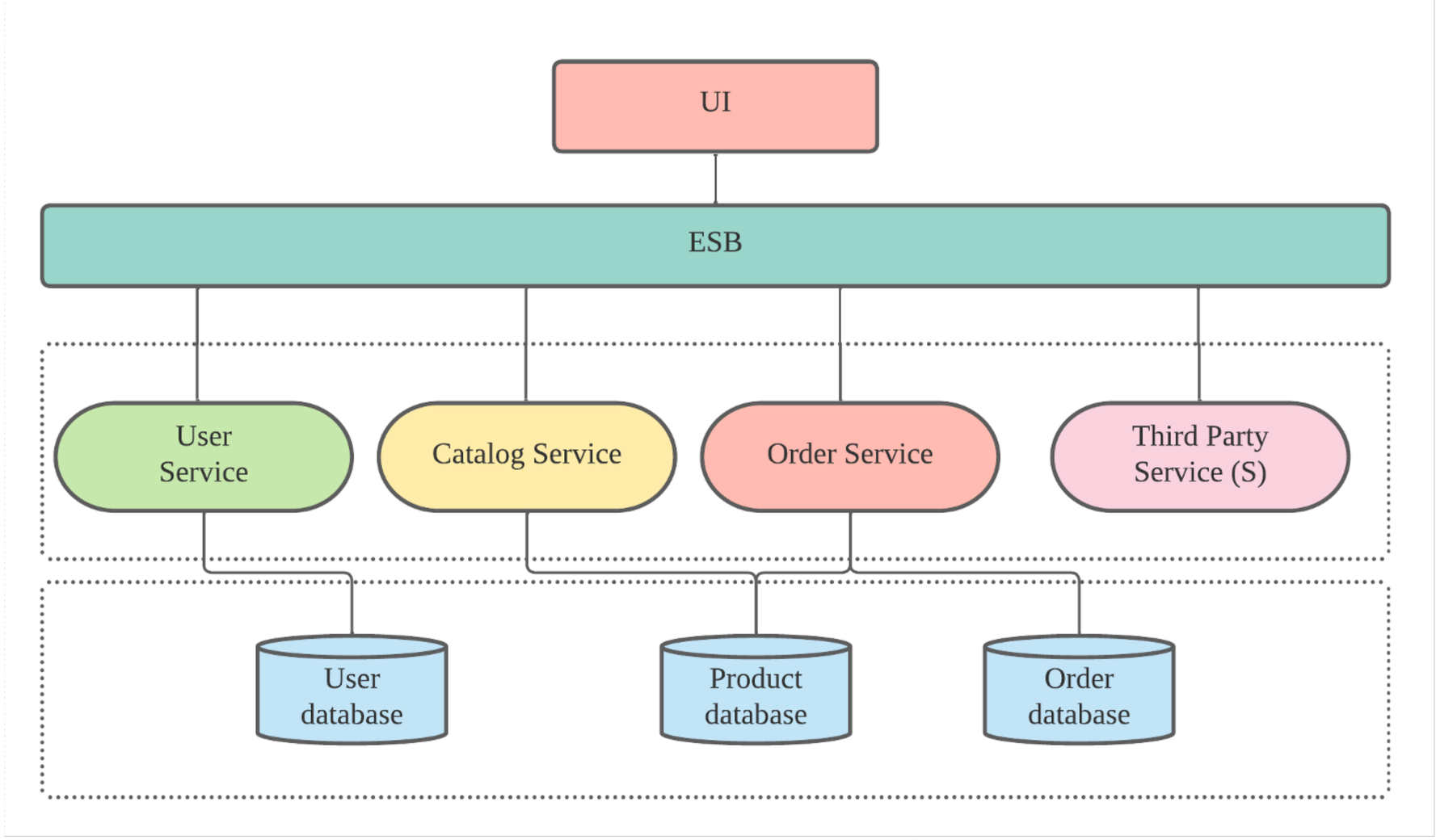
Connectors

SOAP Connector: SOAP Protocol for synchronous communication over HTTP

REST Connector: Relies on request/response operations over HTTP

Asynchronous messaging connector: For point-to-point asynchronous message exchanges or pub-sub exchanges

SOA Applied to E-Commerce



ESB can become bottleneck!

SOA Pattern

Relations

Attachments of different components to available connectors

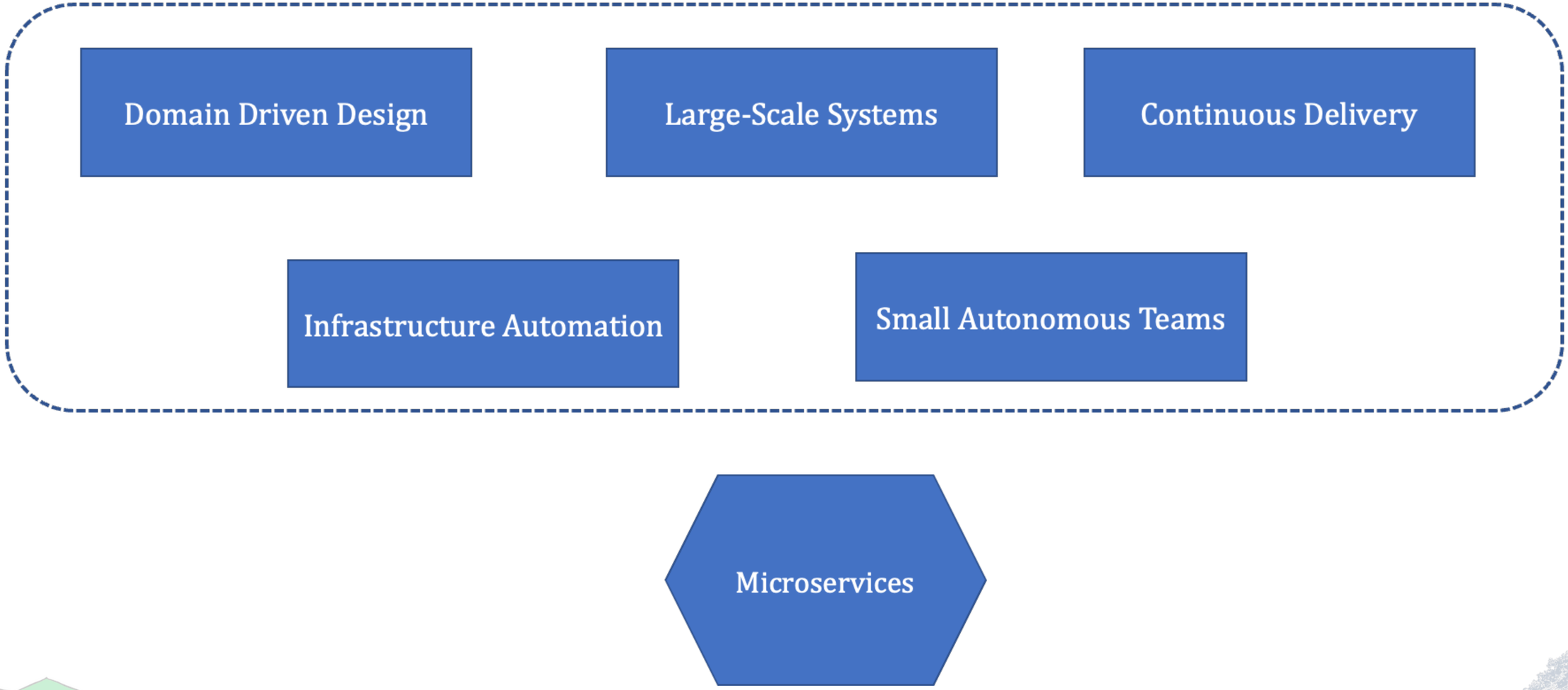
Constraints

- Service consumers are connected to providers (ESBs or other intermediary component may be used)

Weakness

- Complex to build
- Performance bottlenecks due to middleware
- Performance guarantees are usually not met

Into Microservices



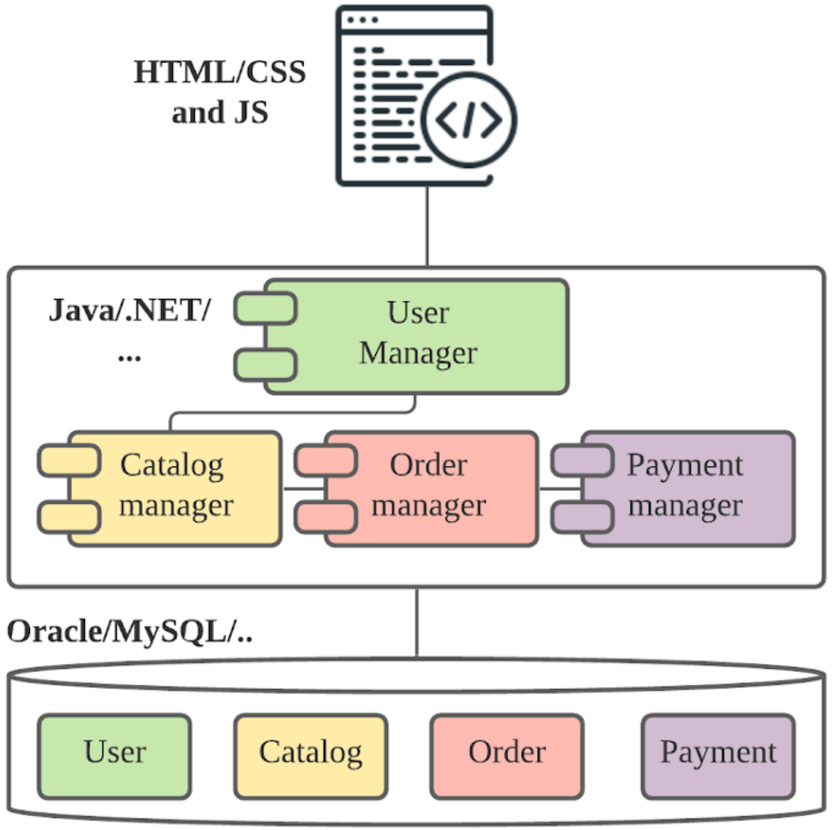
Microservices: What does it mean?

"*Small autonomous services that work together*" -- Sam Newman

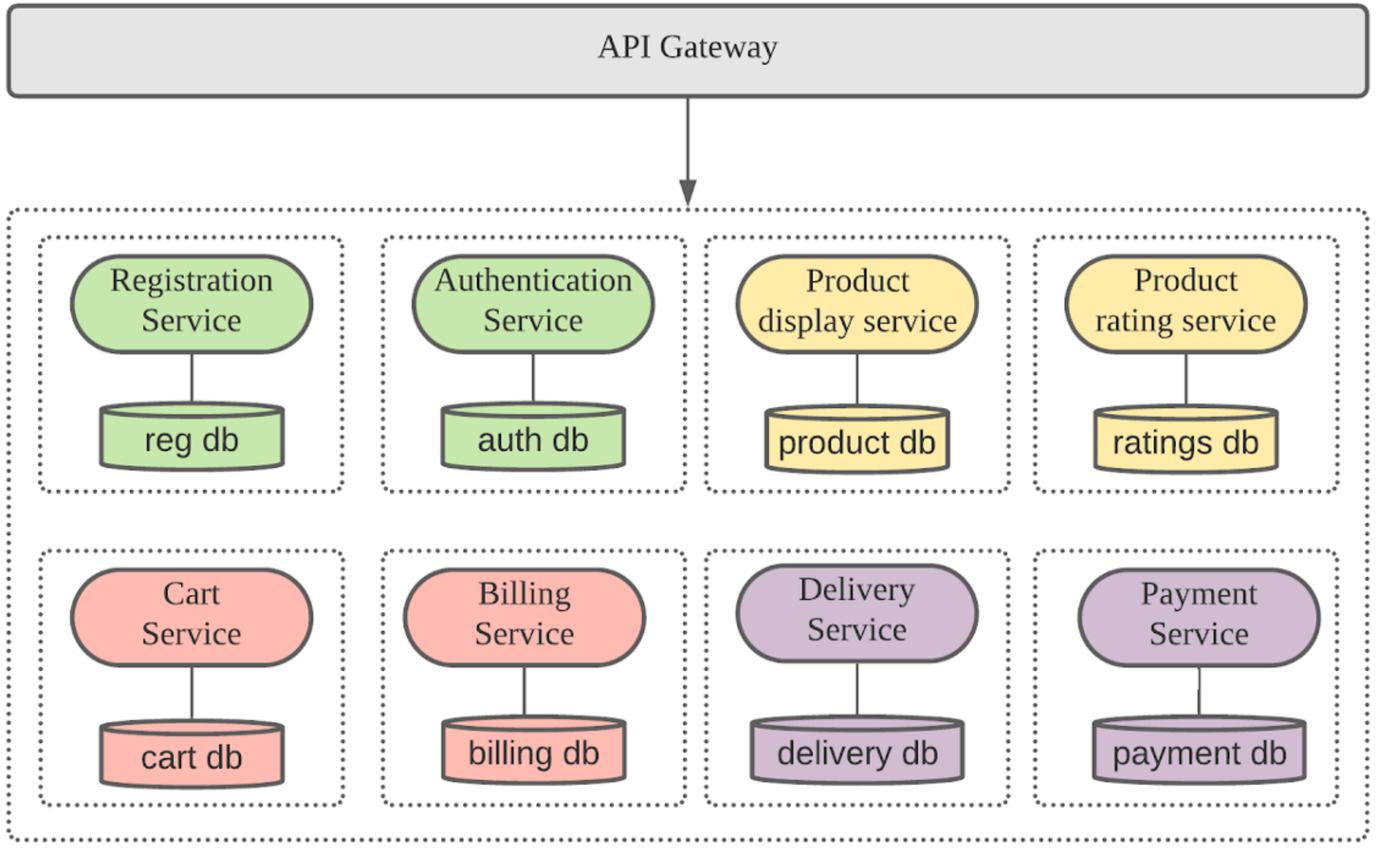
"It is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API" -- Martin Fowler

Microservices: What does it mean?

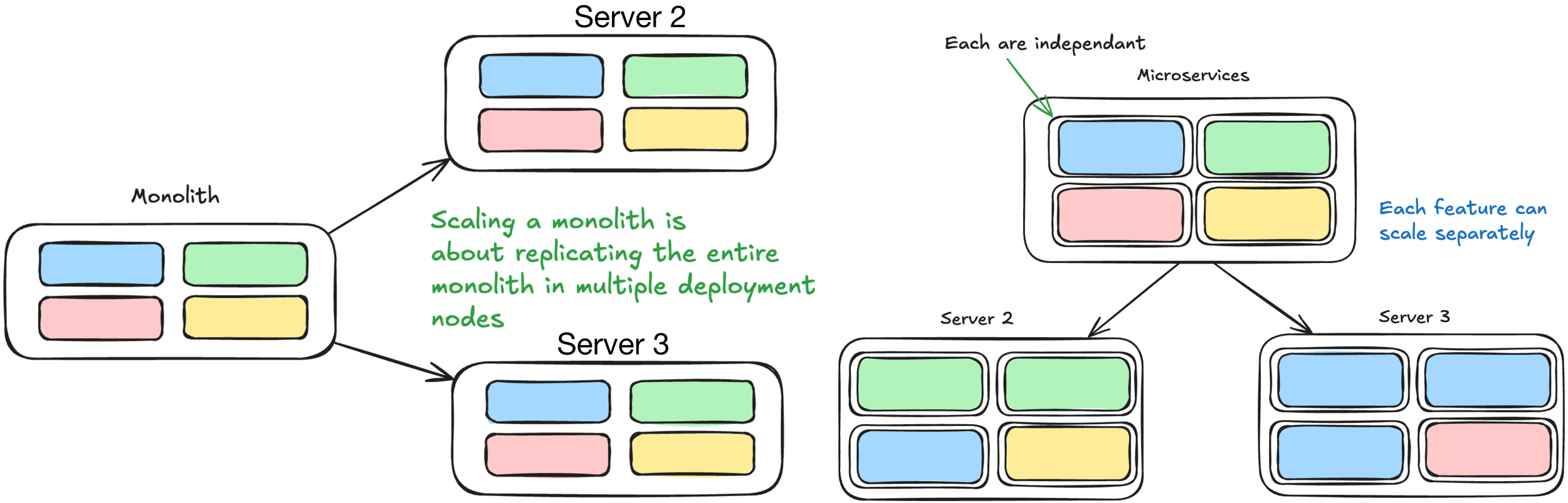
Monolithic Version



Microservices Version



Microservices: What does it mean?



Assume each colour denotes a feature

Thank you

Email: karthik.vaidhyanathan@iiit.ac.in
Twitter: @karthi_ishere