

Modular Monoliths: Introduction



Intro

Monoliths and microservices.

Architectural drivers.

Microservices benefits.

Monolith benefits.

What is Software Architecture?

The design decisions related to the overall system structure and behavior.

It's the decisions that are expensive to change.

Monoliths vs. Microservices

Let's do a high-level comparison between monoliths and microservices.

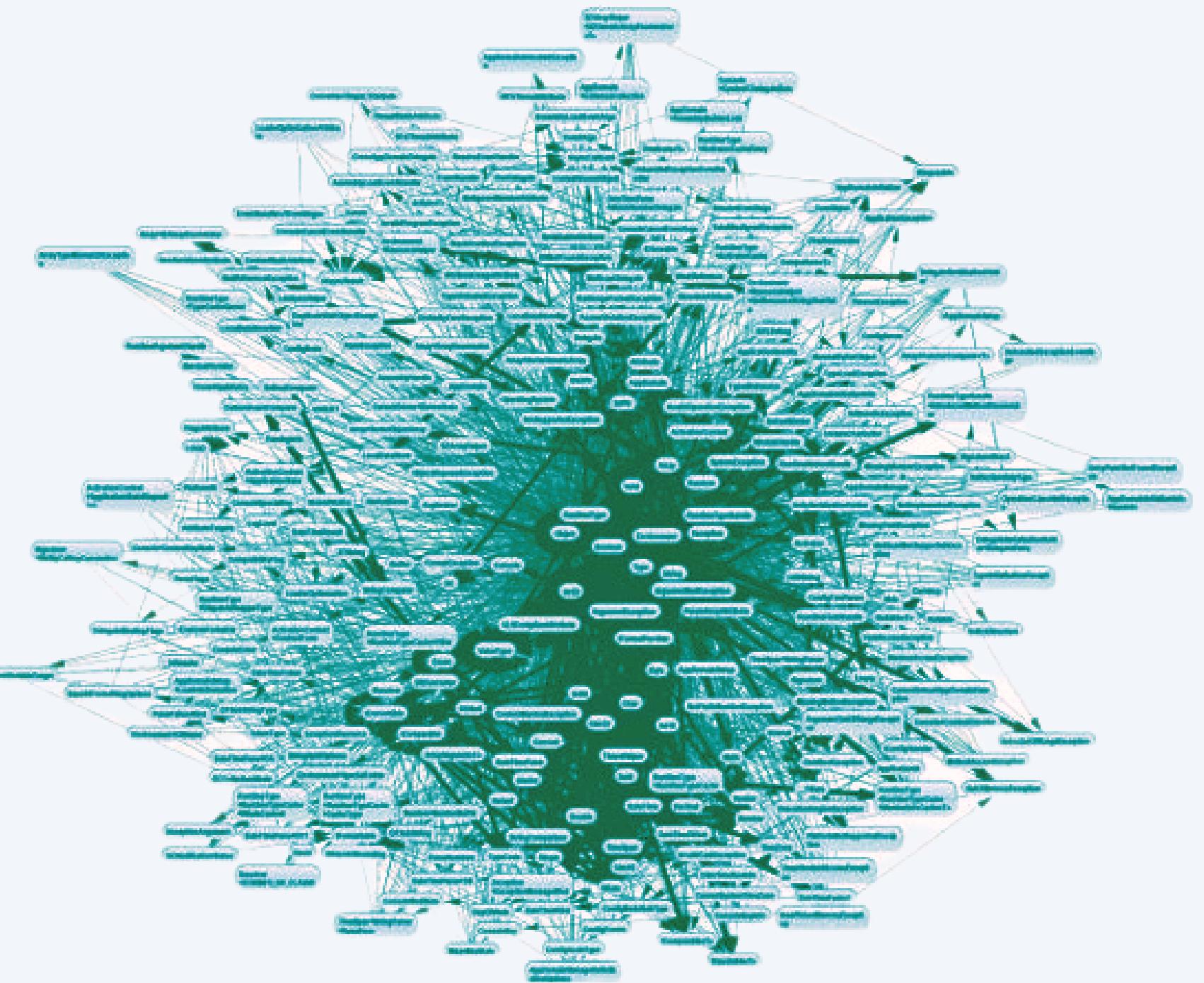
Monoliths

- One deployment unit
- Communication using method calls
- Scalable but less resource-efficient
- Immediate consistency

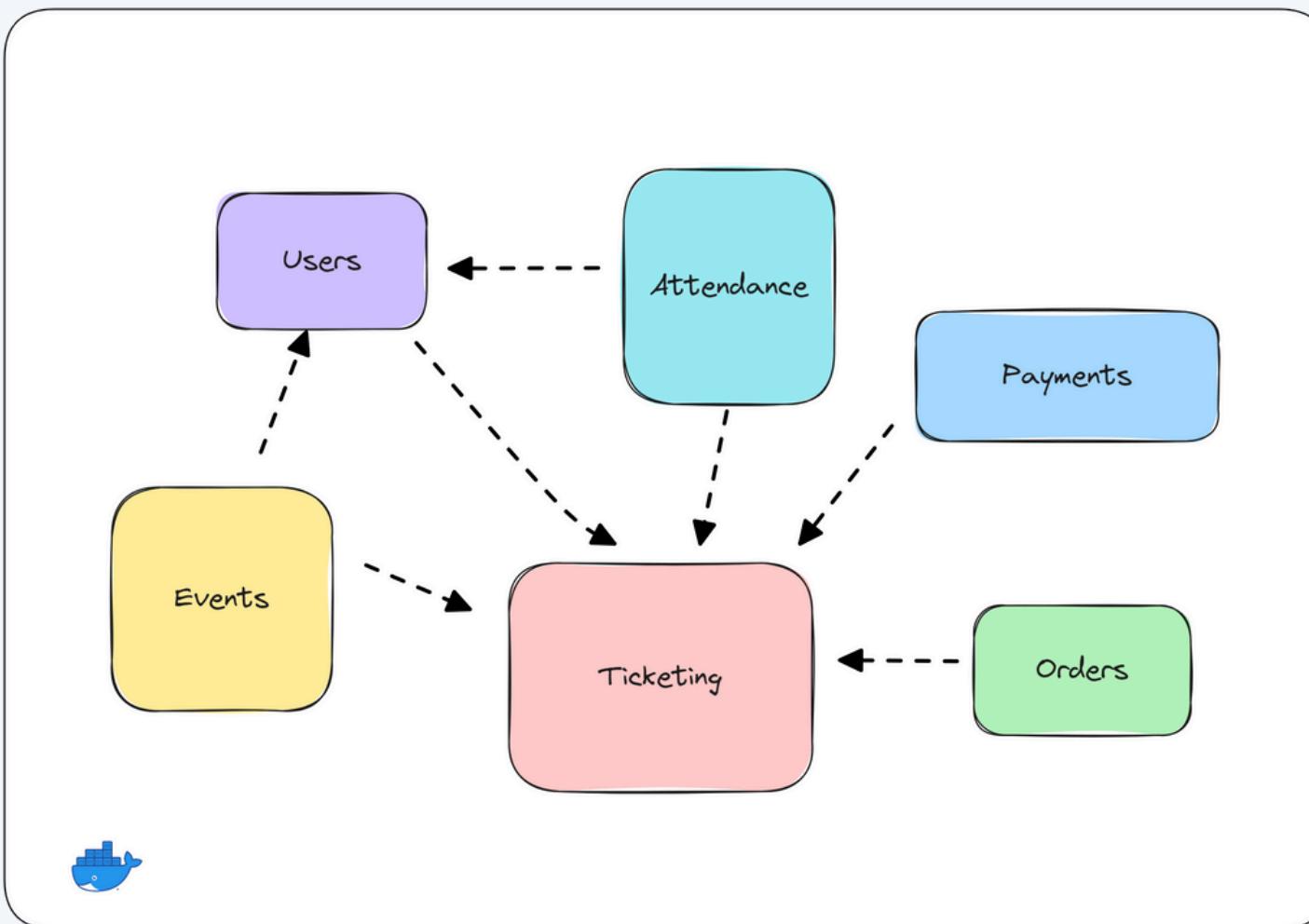


Microservices

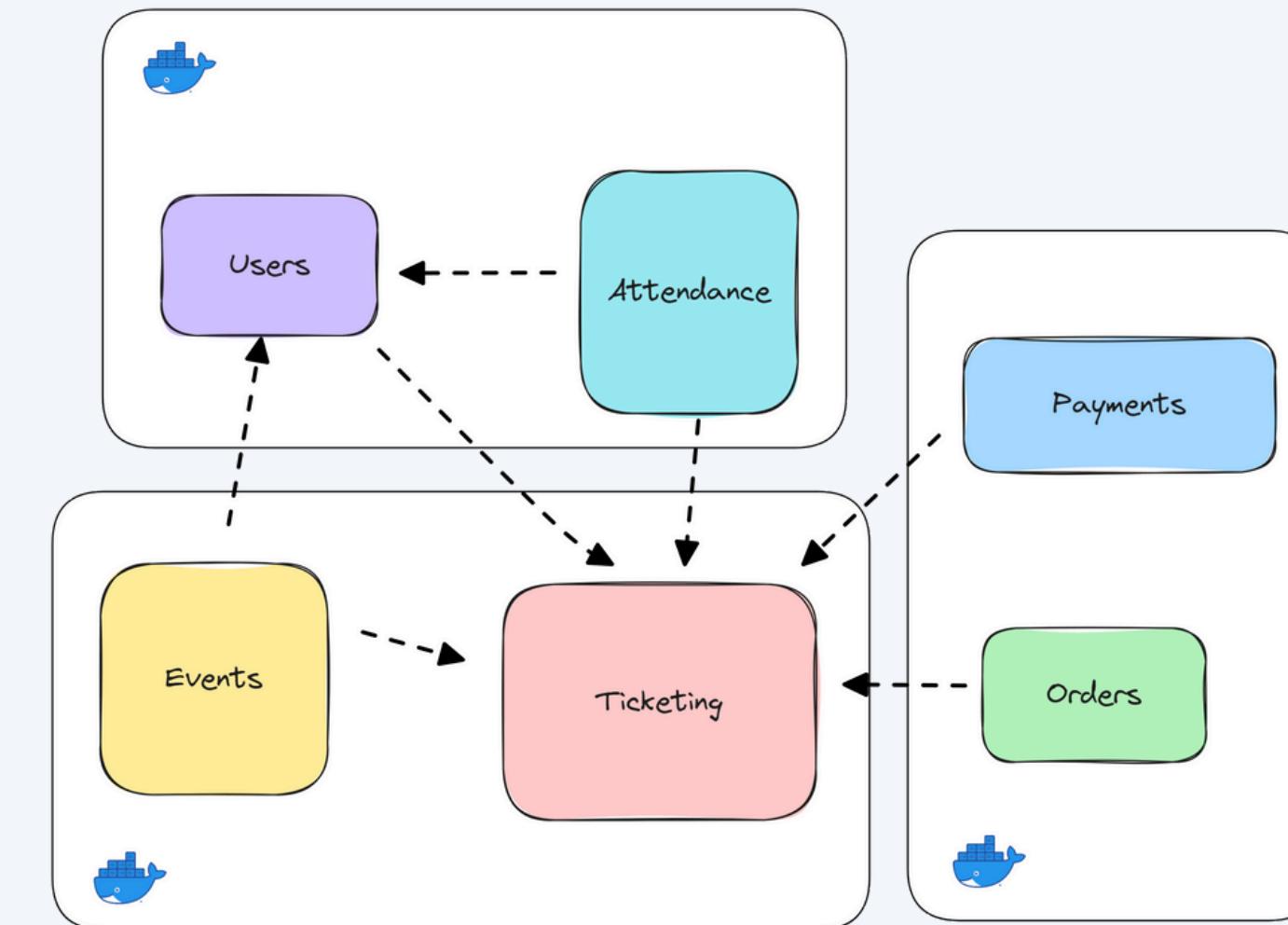
- Many deployment units
- Communication using network calls
- Horizontally and vertically scalable
- Eventual consistency



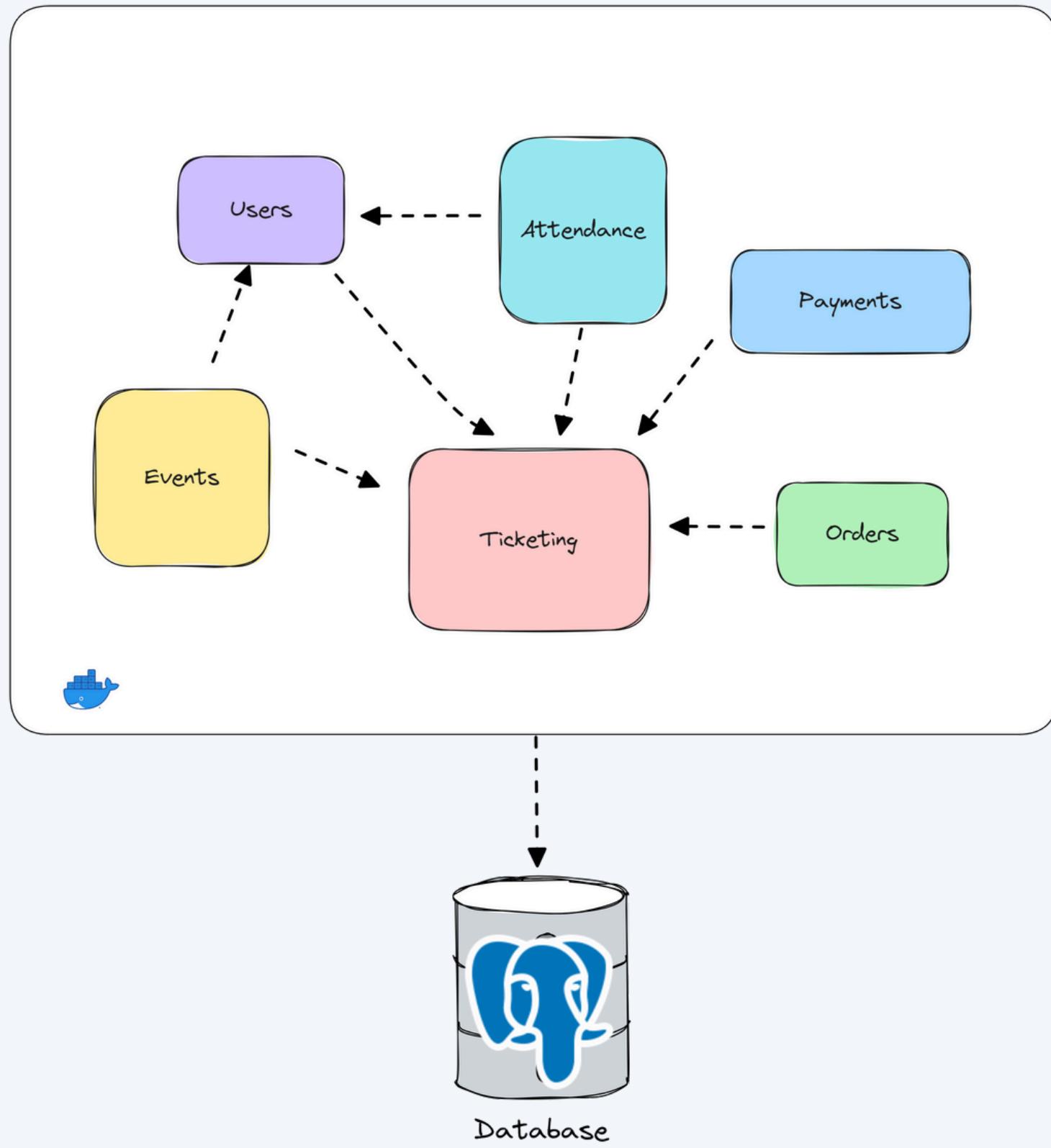
Monoliths



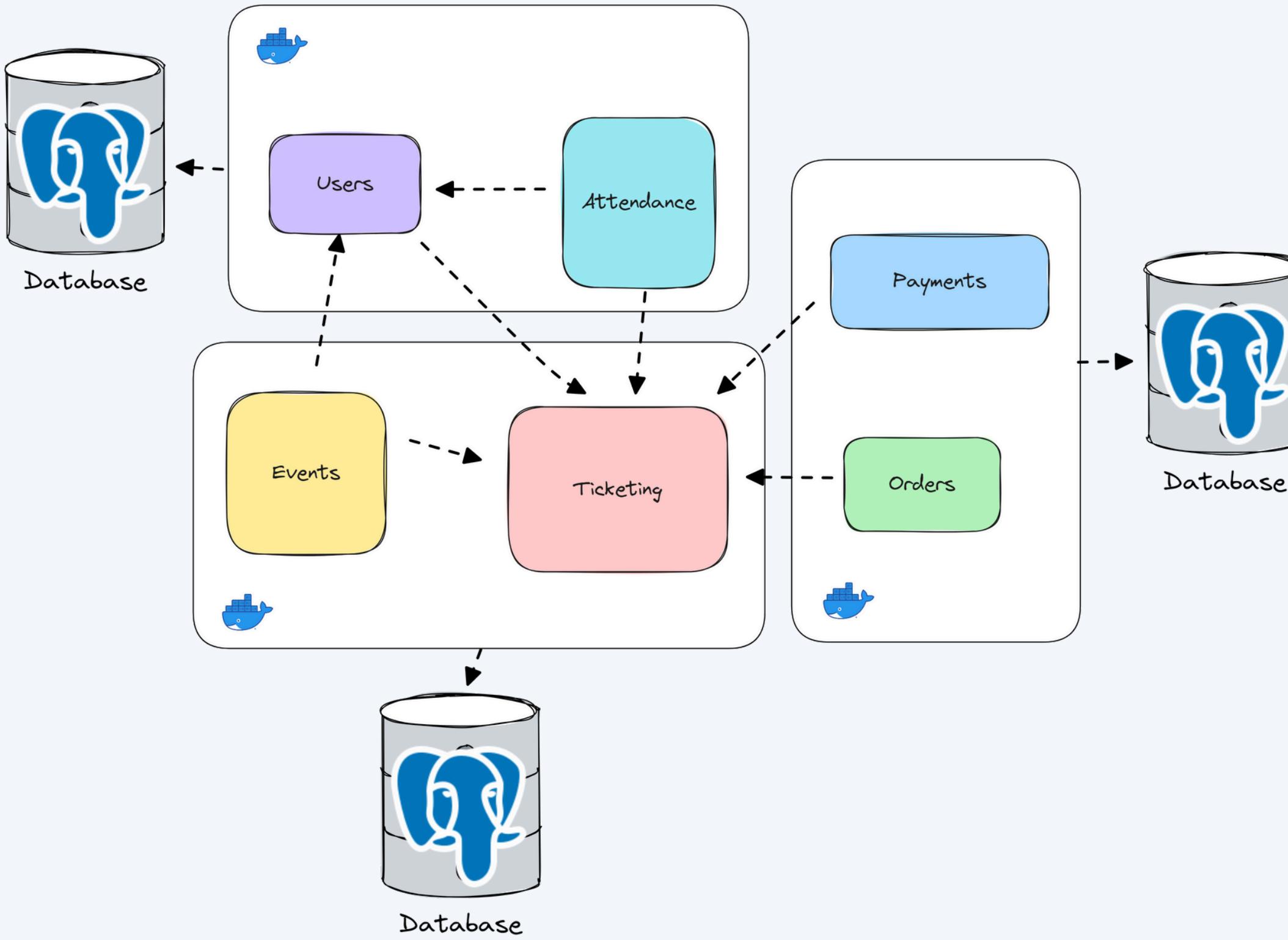
Microservices



Immediate Consistency



Eventual Consistency



Which One Is Better?

We can't just say that (modular) monoliths or microservices are better.

We have to make our decision in a given context.

Architectural Drivers

Architectural drivers are a set of requirements that drive the design process over your architecture.

- Technical Constraints
- Business Constraints
- Functional Requirements
- Quality Attributes

Technical Constraints

Technical constraints are design choices made by the development team that are fixed and hard to reverse in the later stages of the project.

- Programming language
- Use of a specific library or framework

Business Constraints

Business constraints are non-negotiable boundaries affecting the business side of your project.

- Timing (deadlines)
- Budget

Functional Requirements

Functional requirements describe what problems the system solves and how it solves those problems.

Quality Attributes

They're also known as “ilities”.

- Scalability
- Availability
- Testability
- Modularity
- Maintainability

Choosing an Architecture

Architectural drivers should guide which software architecture you choose.

Think about how your decisions impact the business.

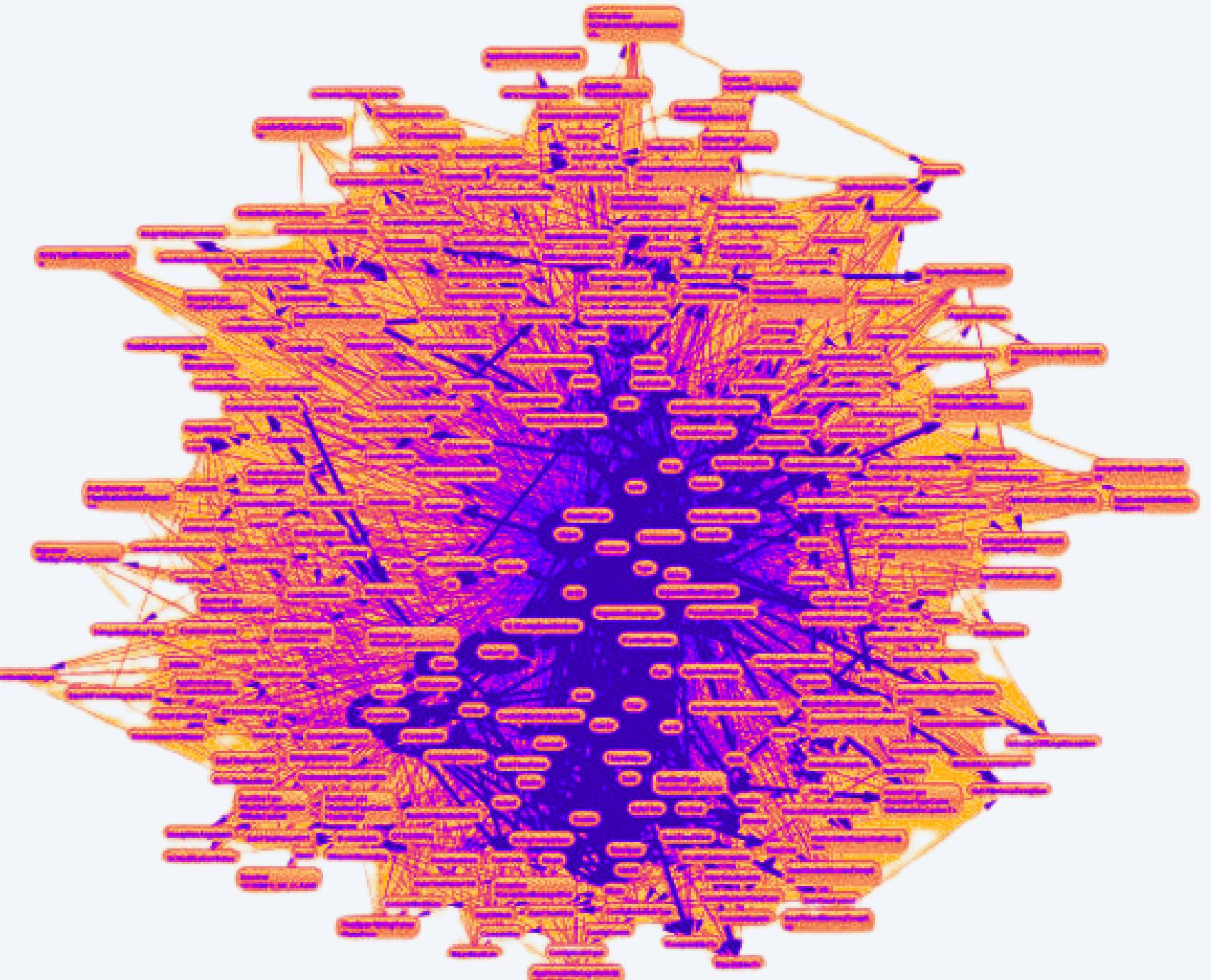
No Free Lunch

Use microservices when you need to.

Microservices **buy** you options, which implies
there is a price to pay.

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



Building distributed systems is hard...

”Choose microservices for the
benefits, not because your
monolithic codebase is a mess.”

- Simon Brown

Microservices Benefits

Good reasons to build microservices:

- Scalability
- Isolation of data
- Organizational autonomy
- Independent deployability
- Zero downtime deployment
- Isolating surface area for failures

The Monolith First movement

”You shouldn't start a new project
with microservices, even if you're
sure your application will be big
enough to make it worthwhile.”

- Martin Fowler

Monolith First

Most successful microservice stories started with a monolith system that got too big and evolved into microservices.

Microservices First?

A system built as a microservice system from scratch will often do worse.

- Service boundaries are difficult to get right
- Boundaries are more challenging to refactor in a distributed system

Who Uses Monoliths?

Many big companies started with monolithic architecture:

- Stack Overflow



Who Uses Monoliths?

Many big companies started with monolithic architecture:

- Stack Overflow
- Twitch



Who Uses Monoliths?

Many big companies started with monolithic architecture:

- Stack Overflow
- Twitch
- Airbnb



Who Uses Monoliths?

Many big companies started with monolithic architecture:

- Stack Overflow
- Twitch
- Airbnb
- Shopify



Who Uses Monoliths?

Many big companies started with (and continue using) monolithic architecture:

- Stack Overflow*
- Twitch
- Airbnb
- Shopify*

*Still running a monolith system.

Advantages of Monoliths

There are many reasons to still build a monolith system:

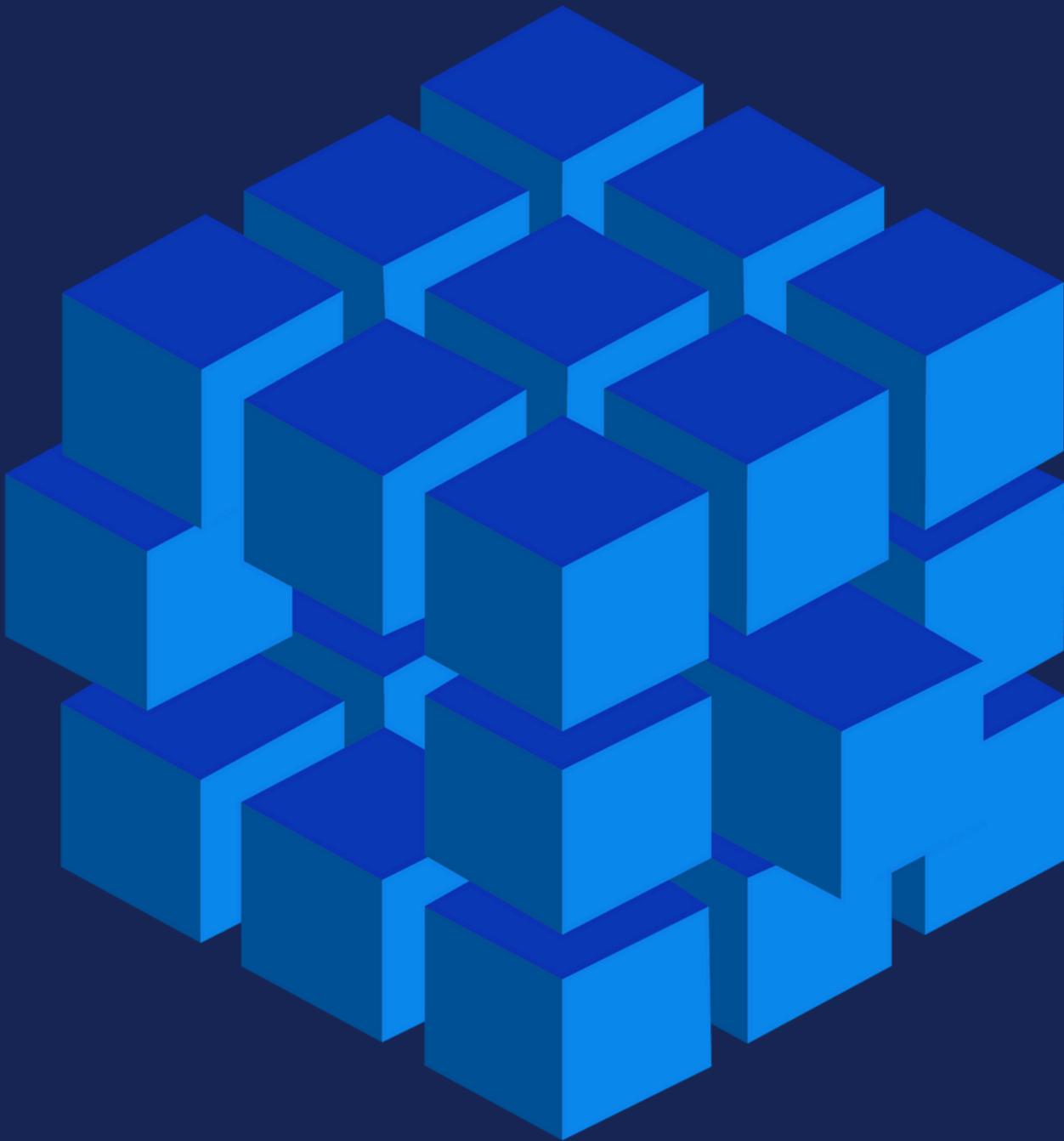
- Simplified deployment
- Easier transaction management
- Reduced complexity compared to microservices

Can We Combine Things?

Get the best of both worlds:

- The **physical architecture** of a monolith
- The **logical architecture** of microservices

Modular Monolith





Next:
Modular Monoliths



Modular Monoliths

Intro

What is a modular monolith?

Modular monolith benefits.

Modular architecture.

What Is a Modular Monolith?

A software design approach in which a monolith is designed with an emphasis on interchangeable (and potentially reusable) modules.

What Is a Modular Monolith?

An explicit name for a monolith system
designed in a modular way.

What Is a Modular Monolith?

Modular (adjective):

- Consisting of separate parts that, when combined, form a complete whole
- Made from a set of separate parts that can be joined together to form a larger object

Simplified Explanation

A modular monolith is an architectural pattern that structures the application into **independent modules** or components with **well-defined boundaries**.

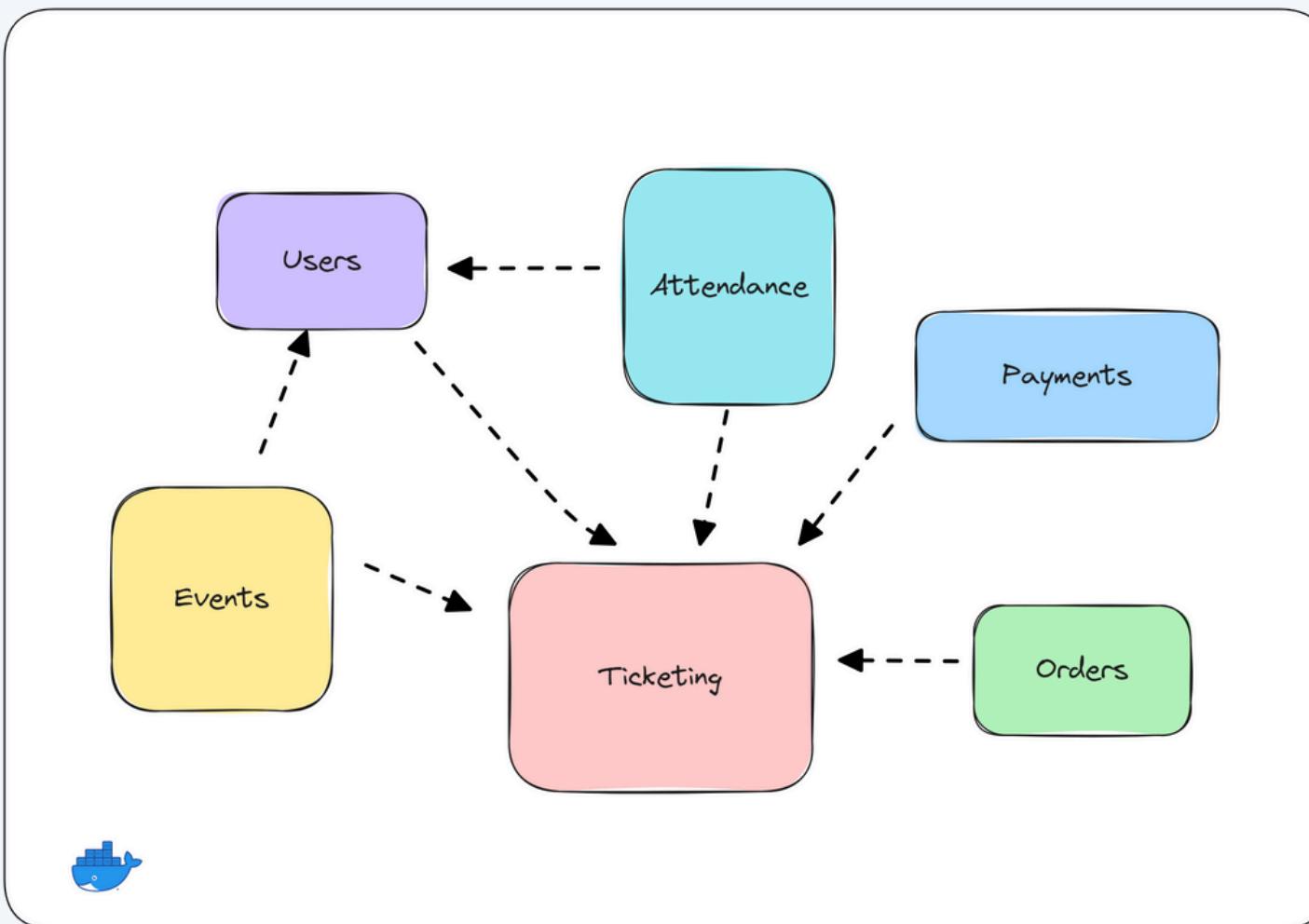
What Is a Module?

Modules represent cohesive sets of functionalities.

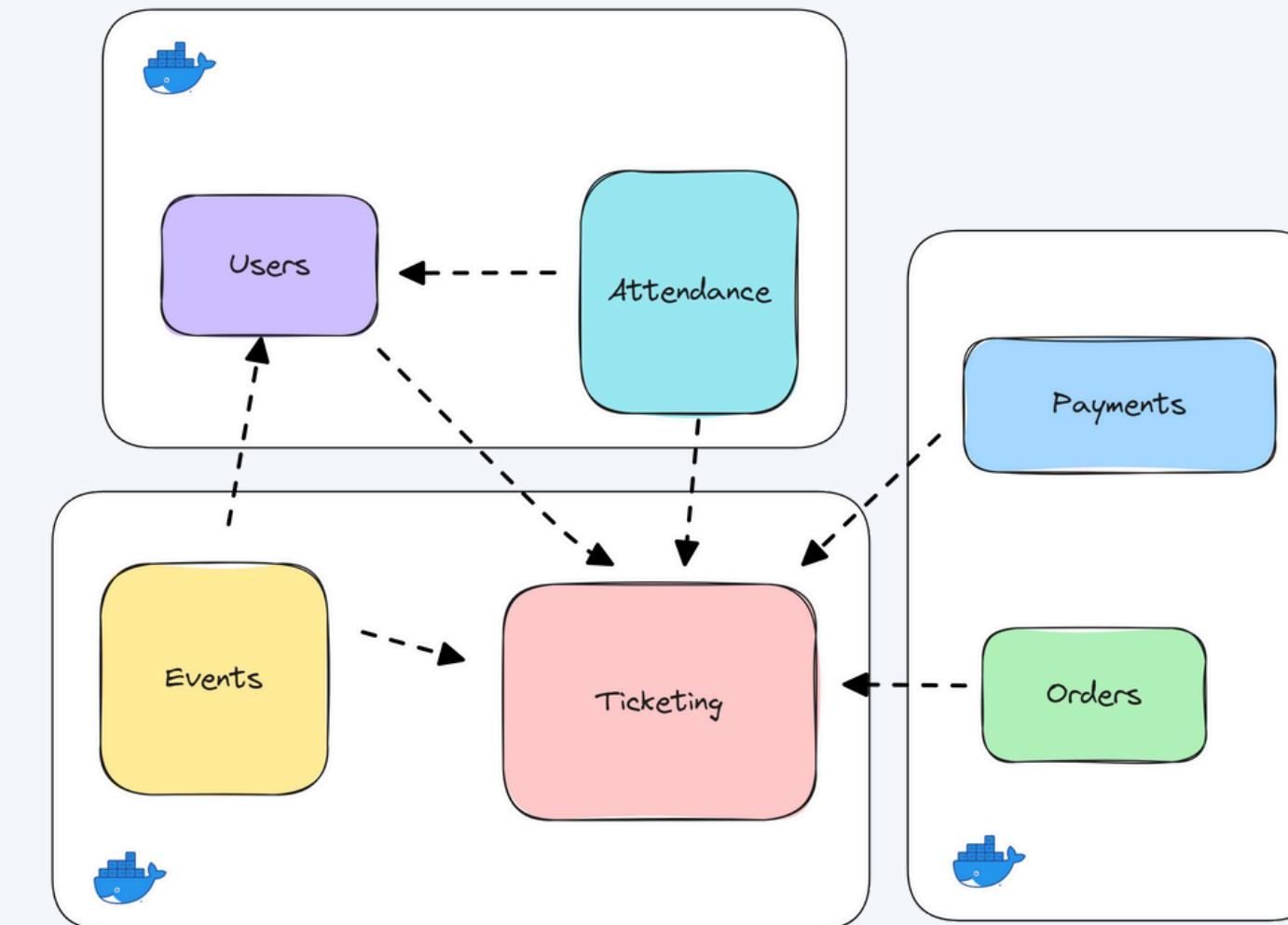
Each module can be treated like a separate *application*.

Modules should be self-contained.

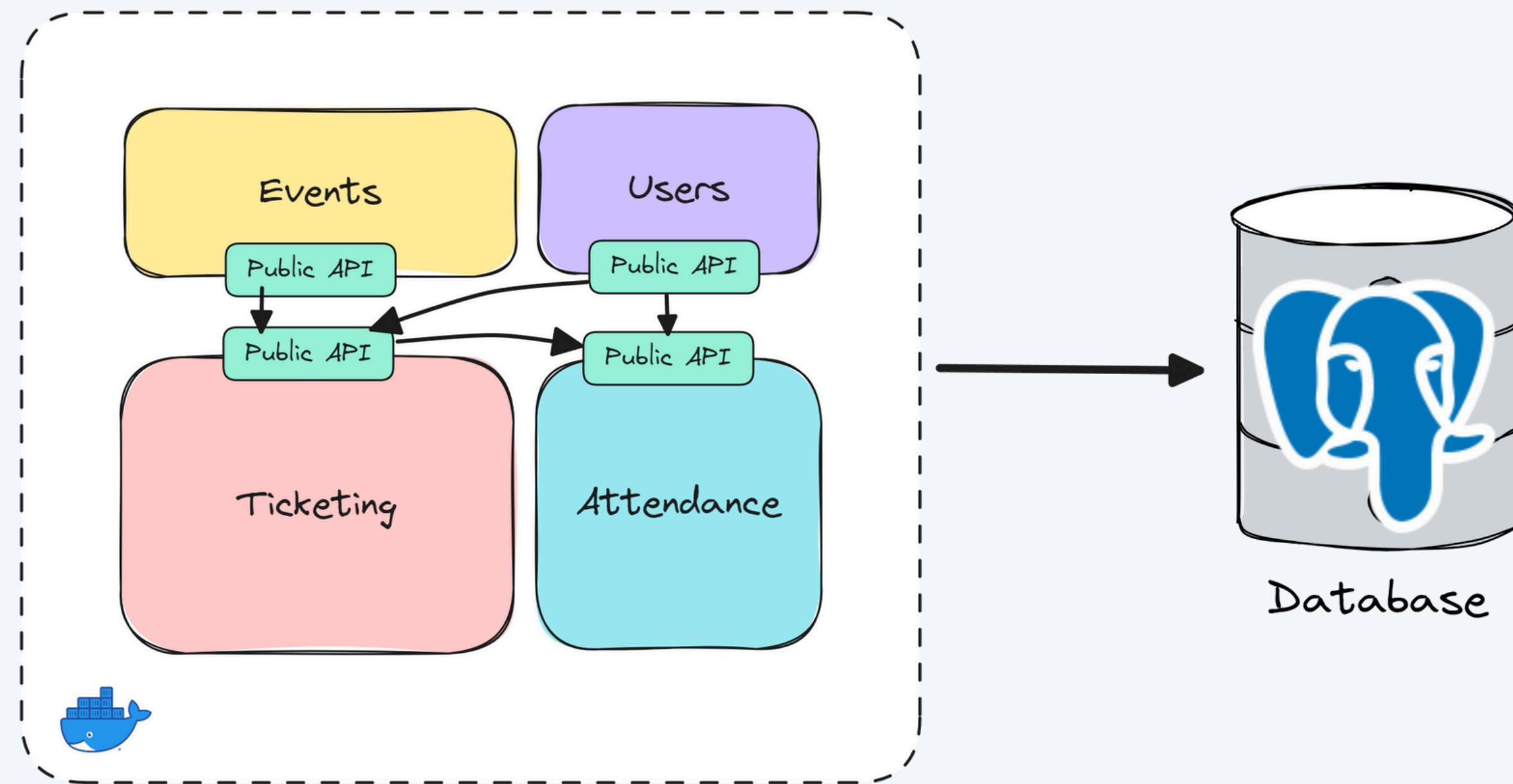
Monoliths



Microservices



Modular Monolith



What are the Benefits?

Modular Monolith Benefits

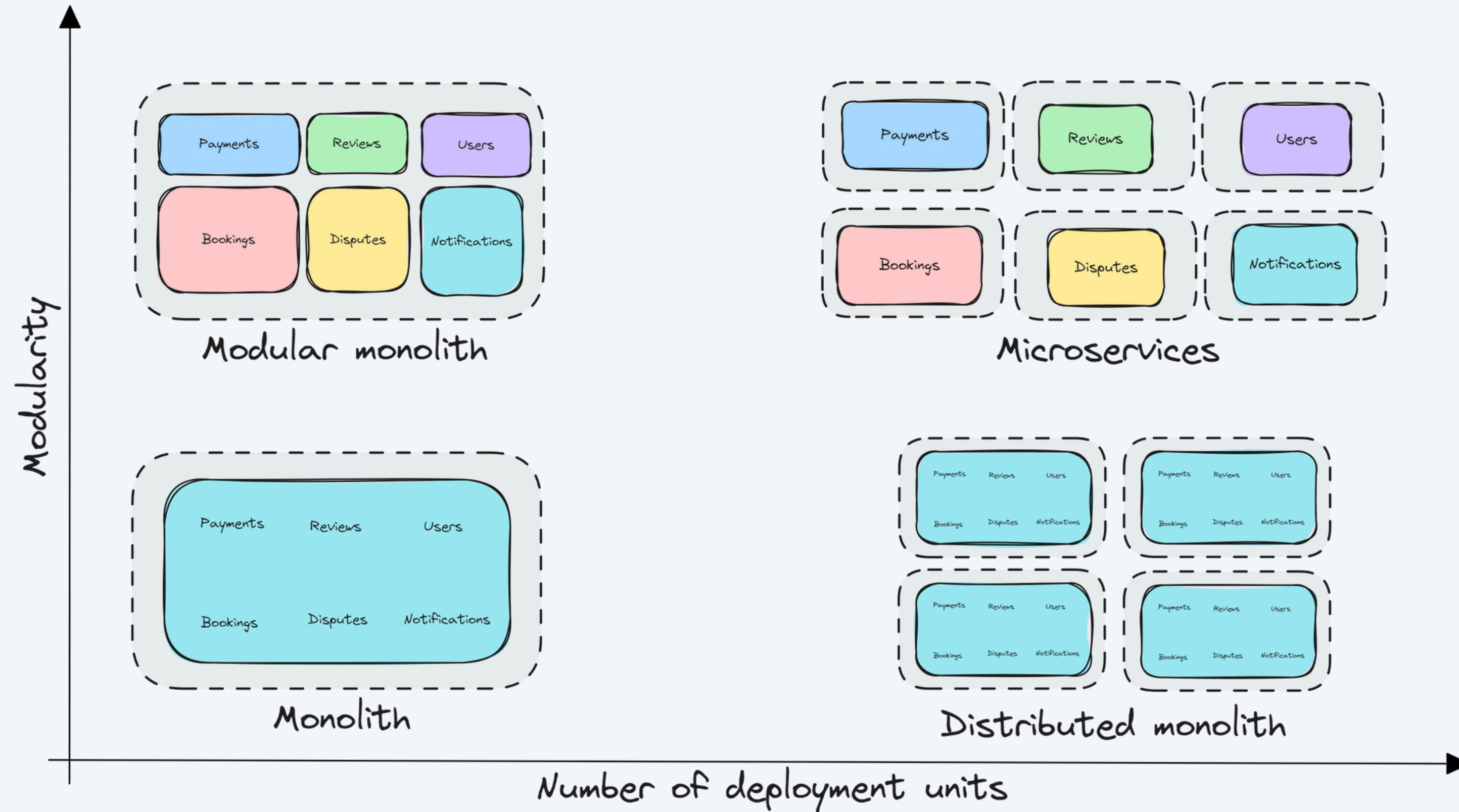
- High cohesion
- Loose coupling
- Simplified deployment
- Improved performance
- Lower operational complexity
- Easier transition to microservices

Okay. But how do we build it?

Modular Architecture

To achieve a modular architecture, the modules:

- Must be independent and interchangeable
- Must be able to provide the required functionality
- Must have a well-defined interface exposed to other modules



Modular Monolith vs. Microservices

Microservices elevate the logical boundaries inside a modular monolith into physical boundaries.

Microservices give you a clear strategy for modularity and decomposing the bounded contexts.

Modular Monolith vs. Microservices

The problem is people end up using microservices to enforce code boundaries.

Instead, you can build a modular monolith to get most of the same benefits.

Evolutionary Architecture

Evolutionary architecture is an approach to building software that's designed to evolve over time as:

- Business priorities change
- Customer demands shift
- New technologies emerge

Modular Monoliths: Evolutionary?

They definitely can be evolutionary.

Well-defined, **in-process components** (modules) can be an excellent stepping stone to **out-of-process components** (services).

Well-defined, in-process components
(modules) can be an excellent
stepping stone to out-of-process
components (services).



Next:
Module Constraints



Module Constraints

Intro

Module constraints.

Cohesion and coupling.

Module communication.

Module data isolation.

Modular Monolith

A modular monolith is an architectural pattern that structures the application into **independent modules** or components with **well-defined boundaries**.

Module Constraints

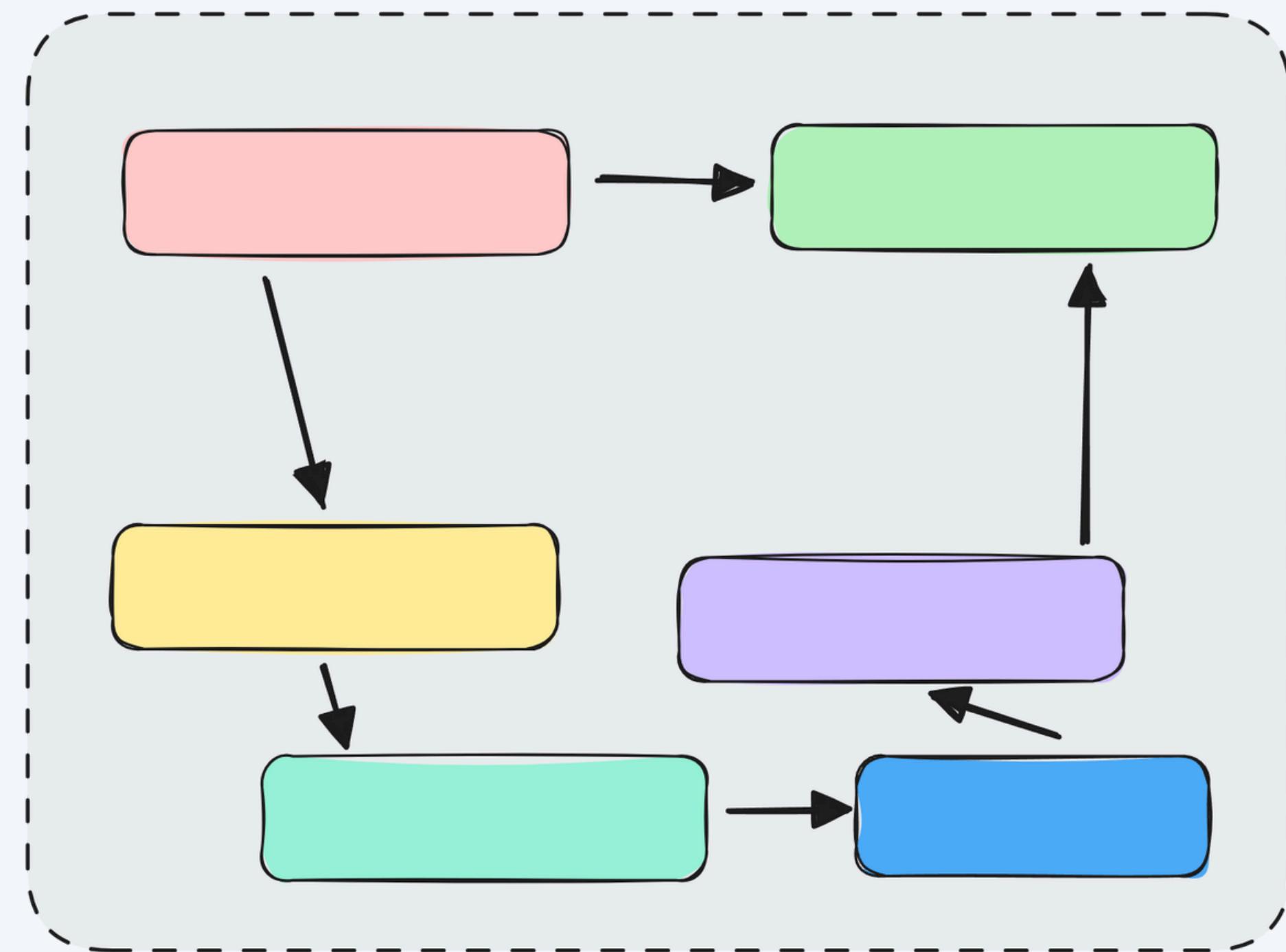
Modules must follow a few constraints:

- Logical isolation between modules
- Modules must talk through a public API
- Module data must be isolated in the database

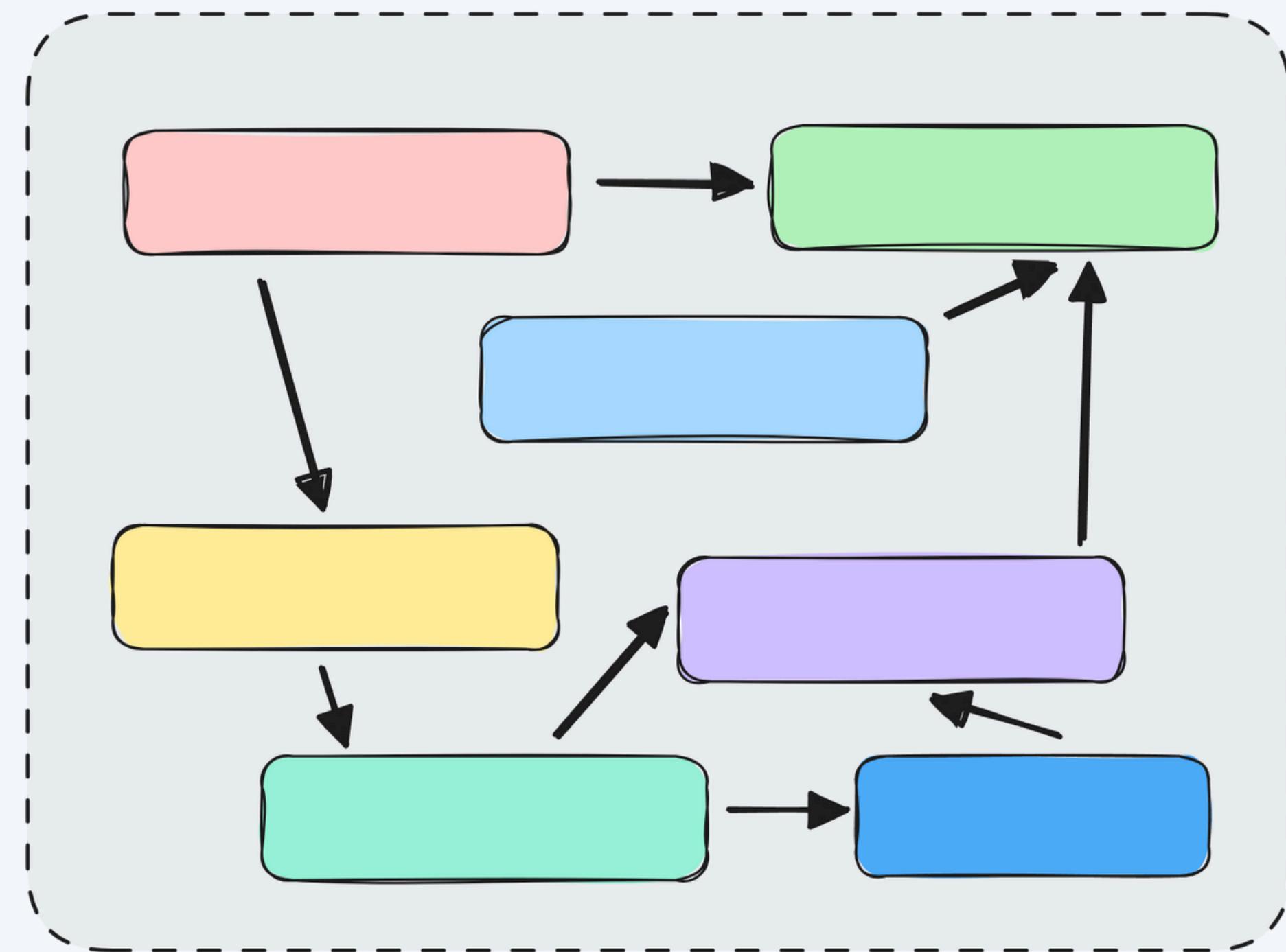
Coupling

Coupling is the degree to which each program module relies on each one of the other modules.

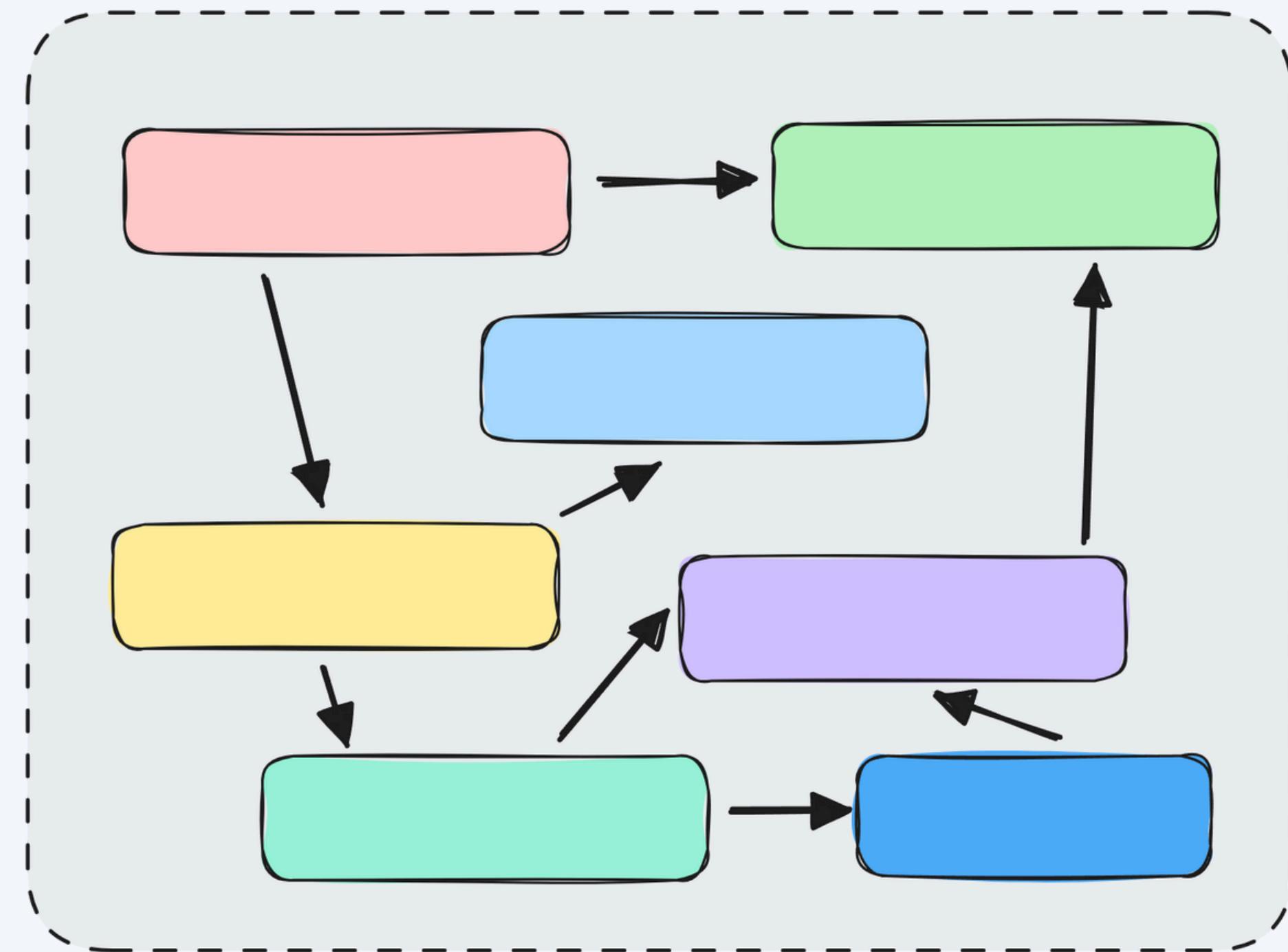
Degree of Coupling



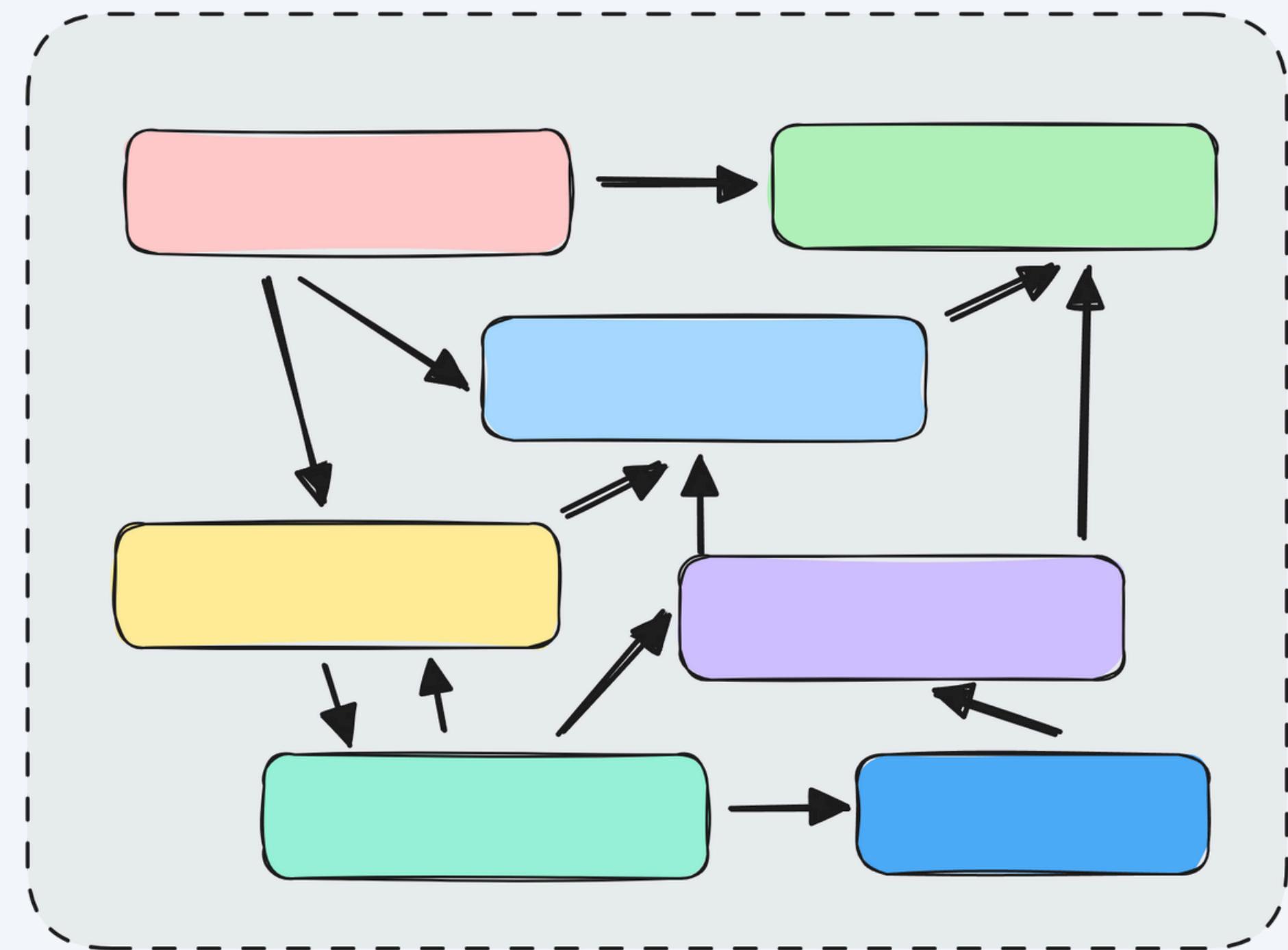
Degree of Coupling



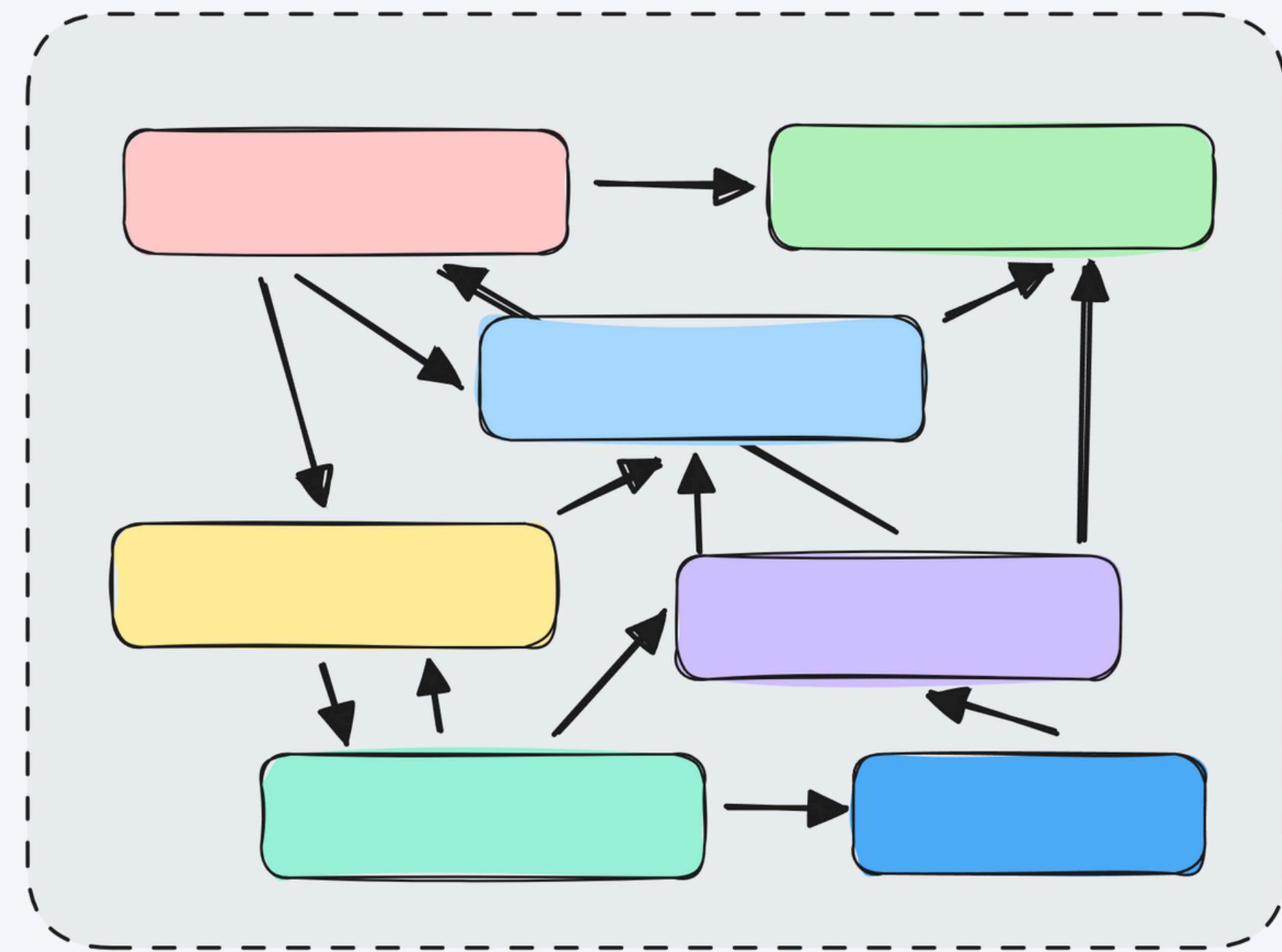
Degree of Coupling



Degree of Coupling



Degree of Coupling



Tight Coupling

Making changes in one part of the system
causes unintended side effects in other parts
of the system.

The system becomes harder to evolve.

Loosely Coupled Modules

We can control the degree of coupling between modules by:

- Communicating through a public API
- Isolating module data in the database

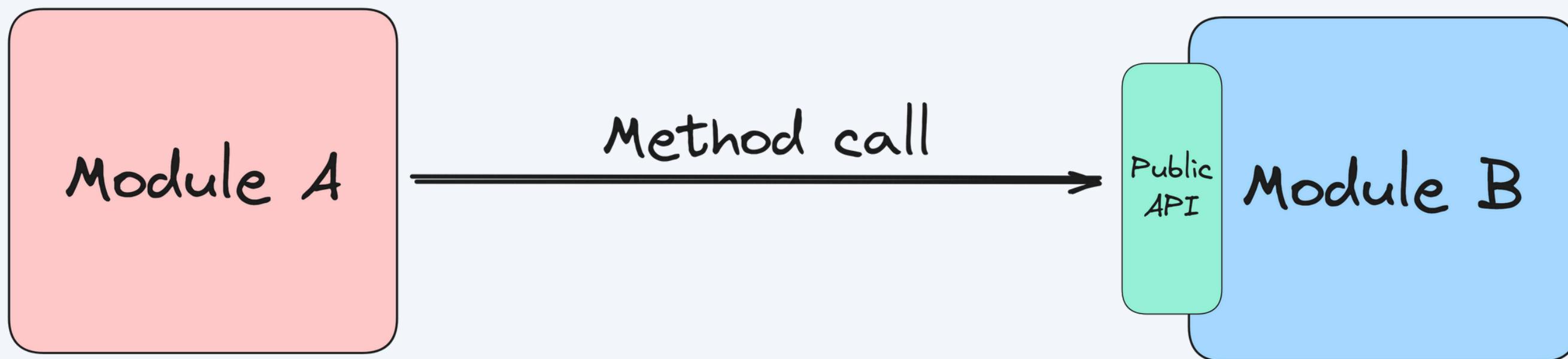
Module Communication

Modular monoliths are loosely coupled by design.

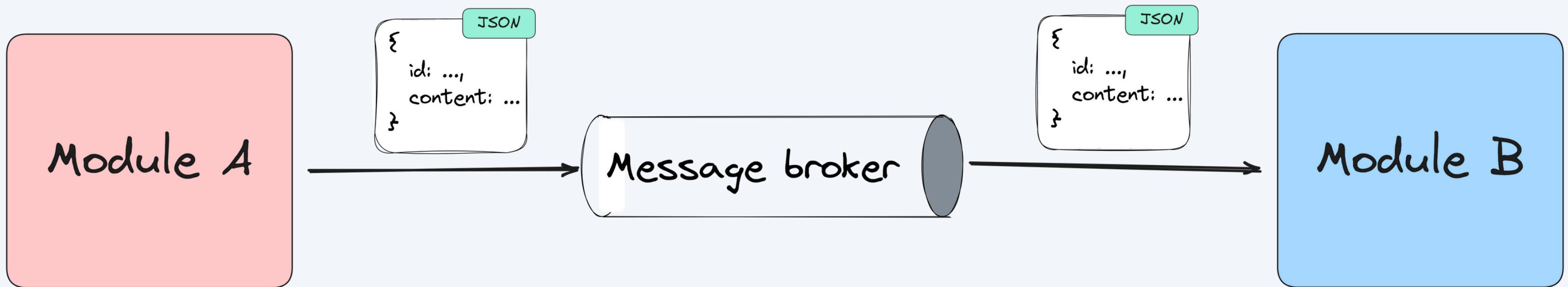
Modules talk through a public API.

- Method calls
- Messaging

Method Calls



Messaging



Module Data Isolation

There are four levels of data isolation:

- Level 1 - Separate tables
- Level 2 - Separate schemas
- Level 3 - Separate databases
- Level 4 - Different databases

Module Data Isolation

There are four levels of data isolation:

- Level 1 - Separate tables
- Level 2 - Separate schemas
- Level 3 - Separate databases
- Level 4 - Different databases

Module Data Isolation

There are four levels of data isolation:

- Level 1 - Separate tables
- **Level 2 - Separate schemas**
- Level 3 - Separate databases
- Level 4 - Different databases

Module Data Isolation

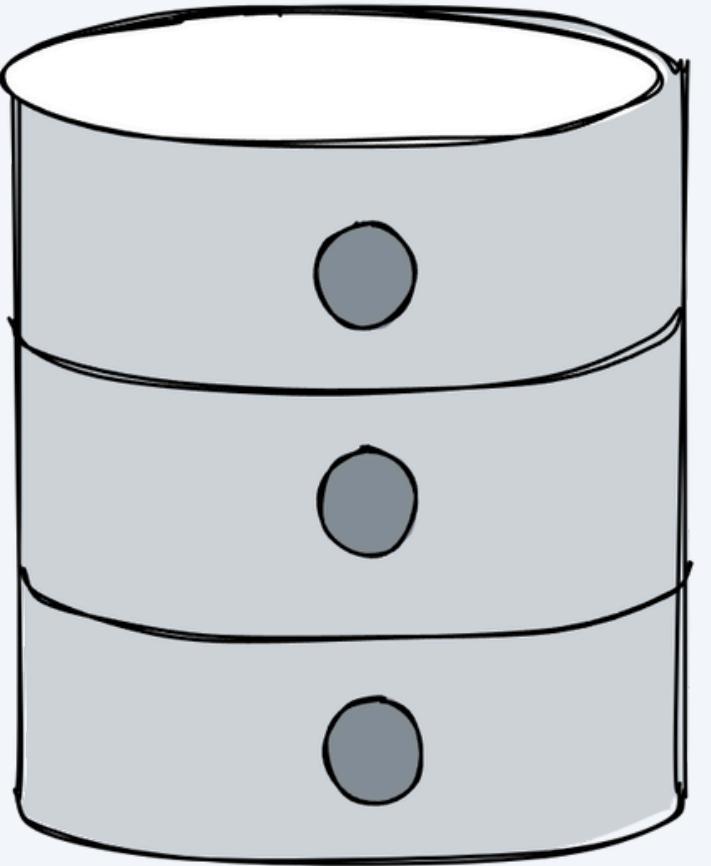
There are four levels of data isolation:

- Level 1 - Separate tables
- Level 2 - Separate schemas
- **Level 3 - Separate databases**
- Level 4 - Different databases

Module Data Isolation

There are four levels of data isolation:

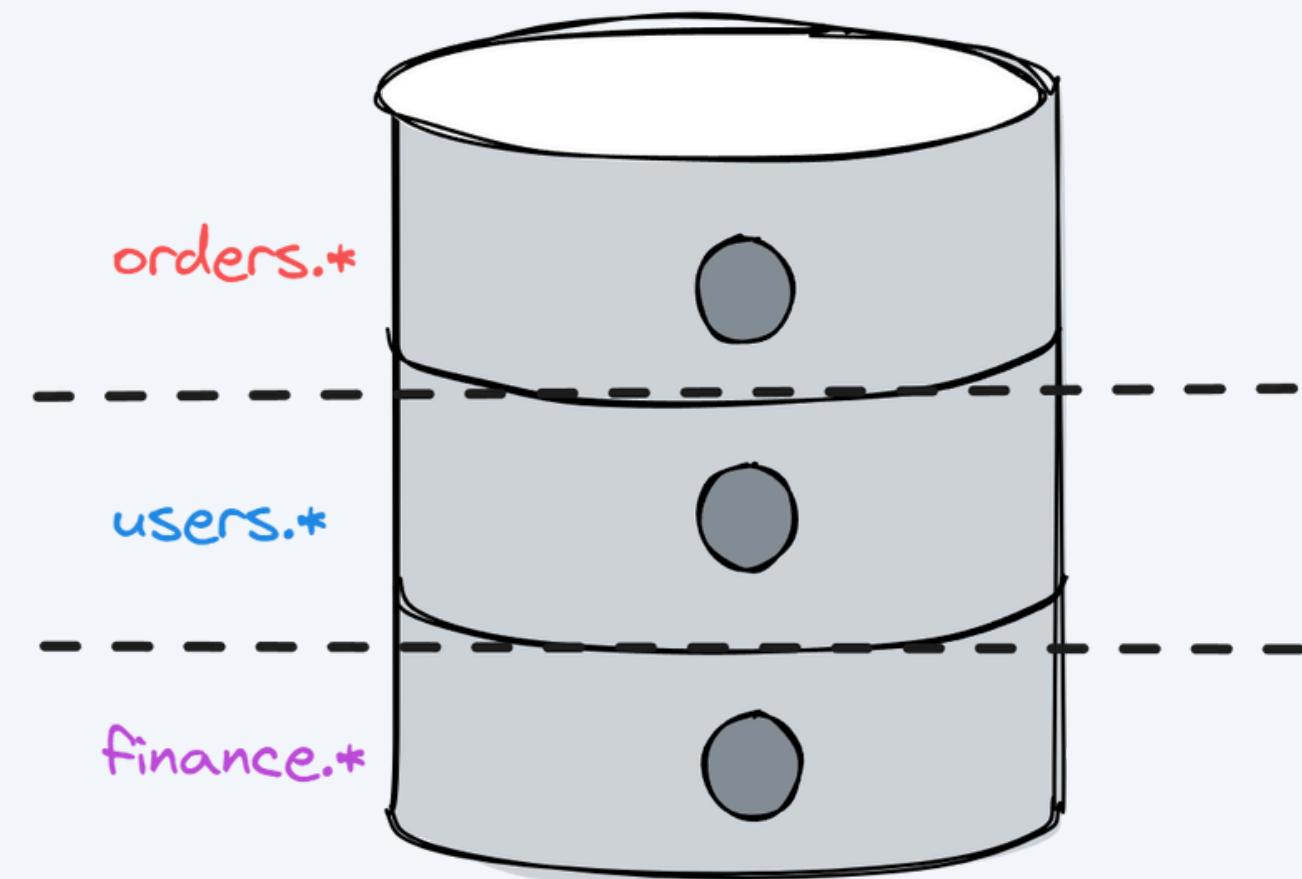
- Level 1 - Separate tables
- Level 2 - Separate schemas
- Level 3 - Separate databases
- Level 4 - Different databases



Relational DB

Separate Table

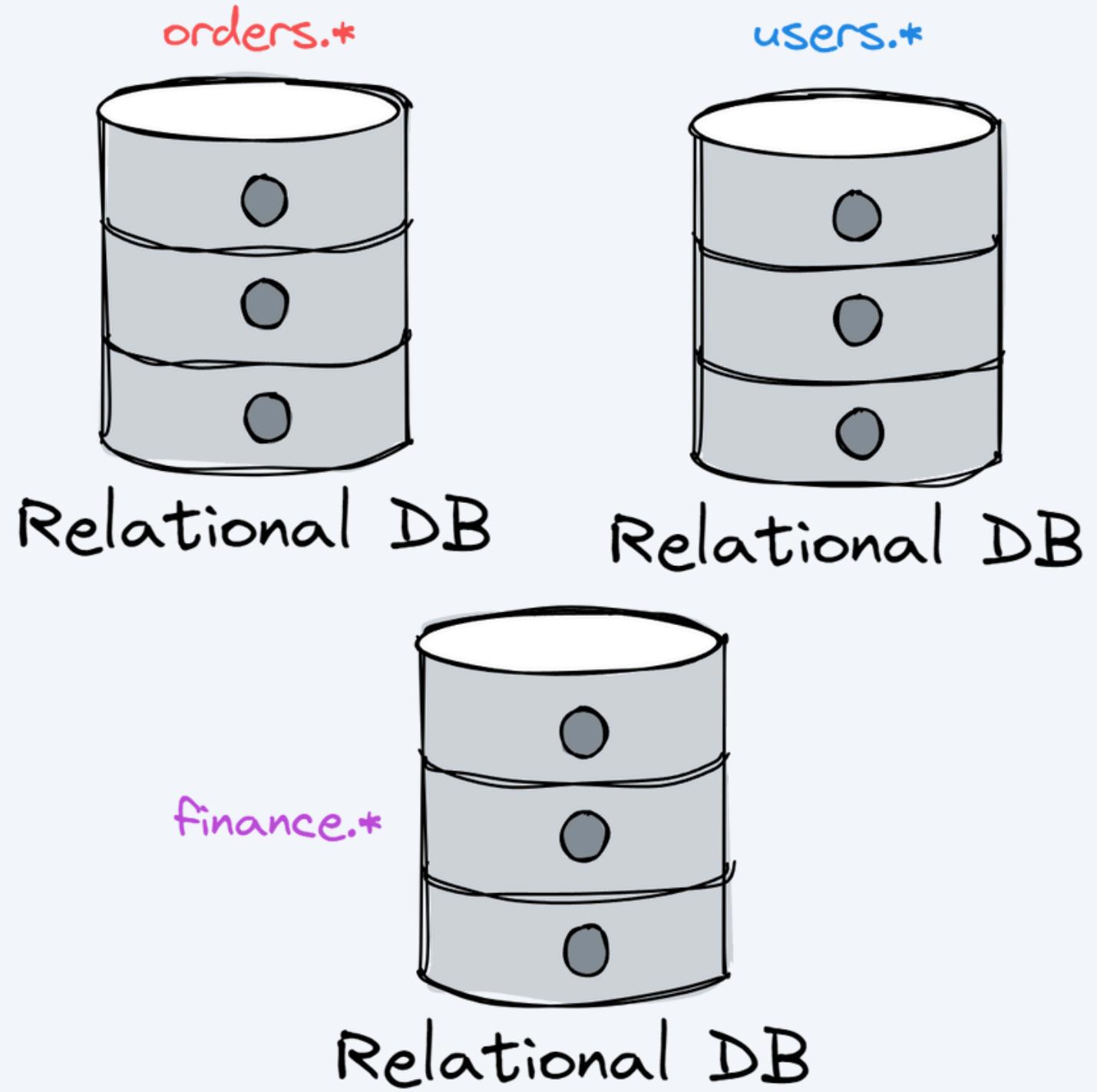
- Same schema
- Same database
- Same DB type



Relational DB

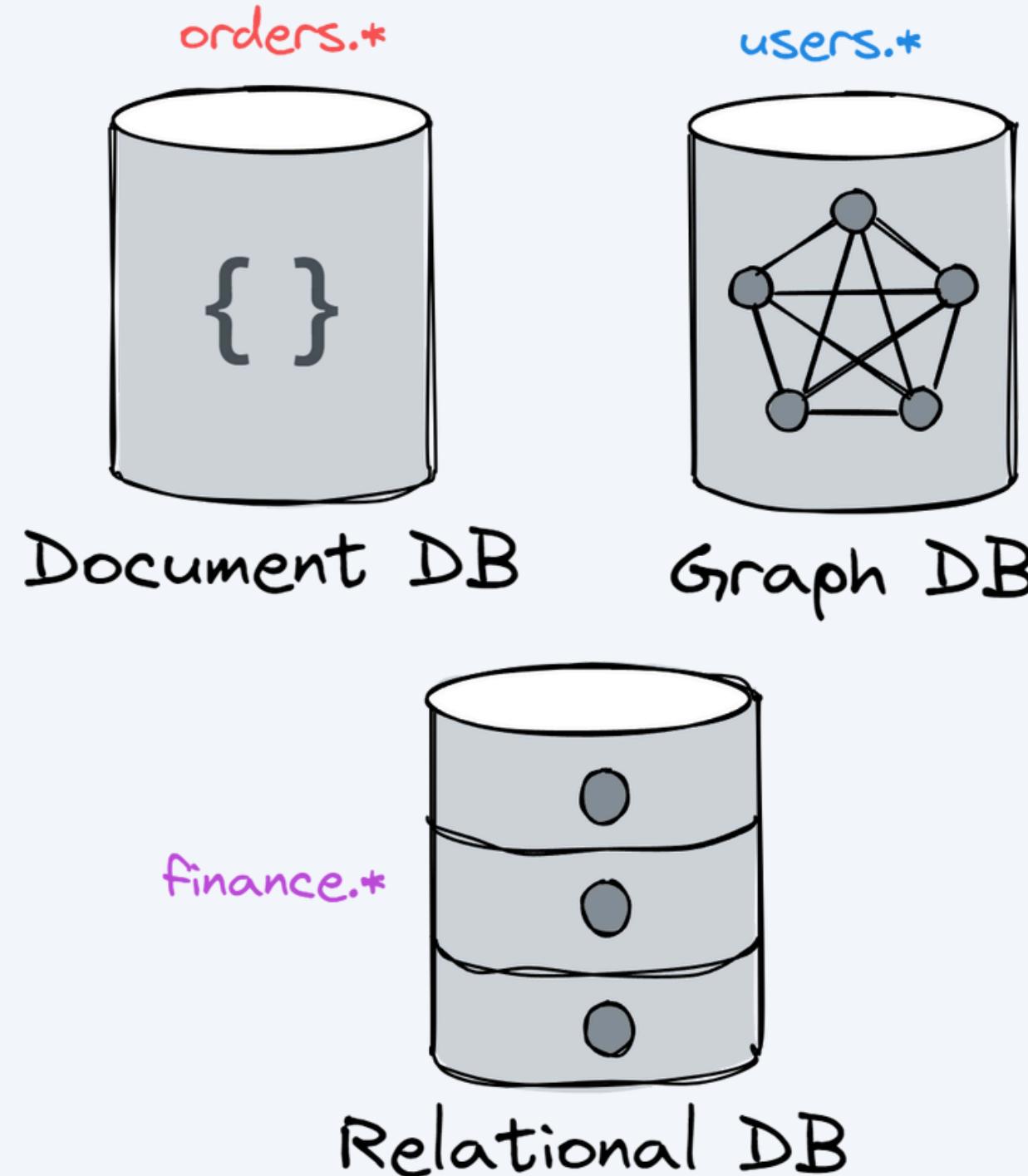
Separate Schema

- Different schema
- Same database
- Same DB type



Separate Database

- Different schema
- Different database
- Same DB type



Different Persistence

- Different schema
- Different database
- Different DB type

Module Data Isolation

My recommendation is to always start with at least **schema-based (logical) data isolation**.

Sharing Data

Never integrate on the database level.

What if I need data from another module?

- Direct call, but using public API
- Messaging (fat events)
- Data copy

Recap: Module Constraints

The modules must follow a few constraints:

- Logical isolation between modules
- Modules must talk through a public API
- Modules must be isolated in the database



Next:
Defining Module
Boundaries

Defining Module Boundaries



Intro

Domain-Driven Design.

Bounded contexts.

Event Storming.

What Is a Module?

Modules represent cohesive sets of functionalities.

Each module can be treated like a separate *application*.

Modules should be self-contained.

Module Constraints

Modules must follow a few constraints:

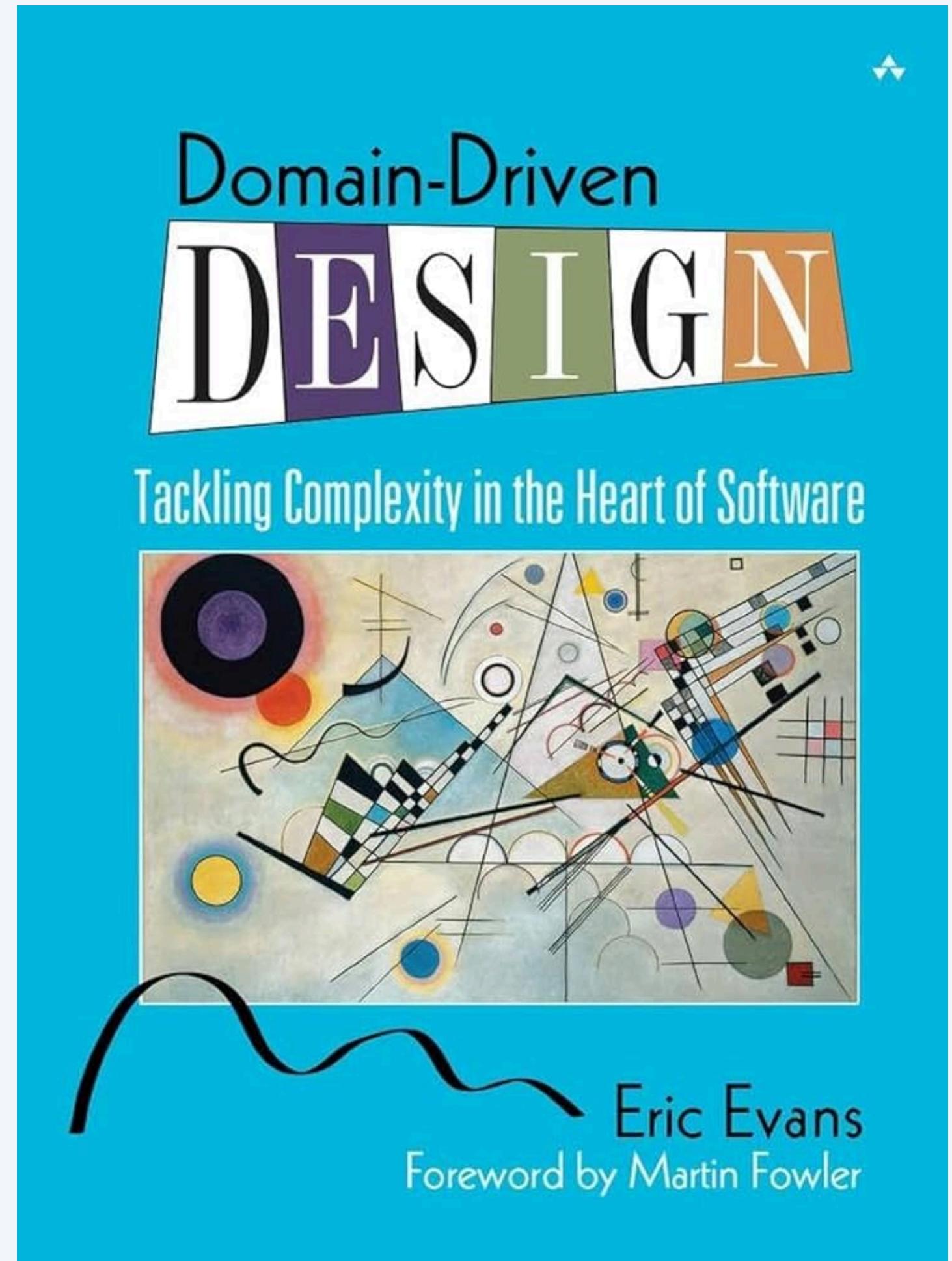
- Logical isolation between modules
- Modules must talk through a public API
- Module data must be isolated in the database

How do we find module boundaries?

Domain-Driven Design

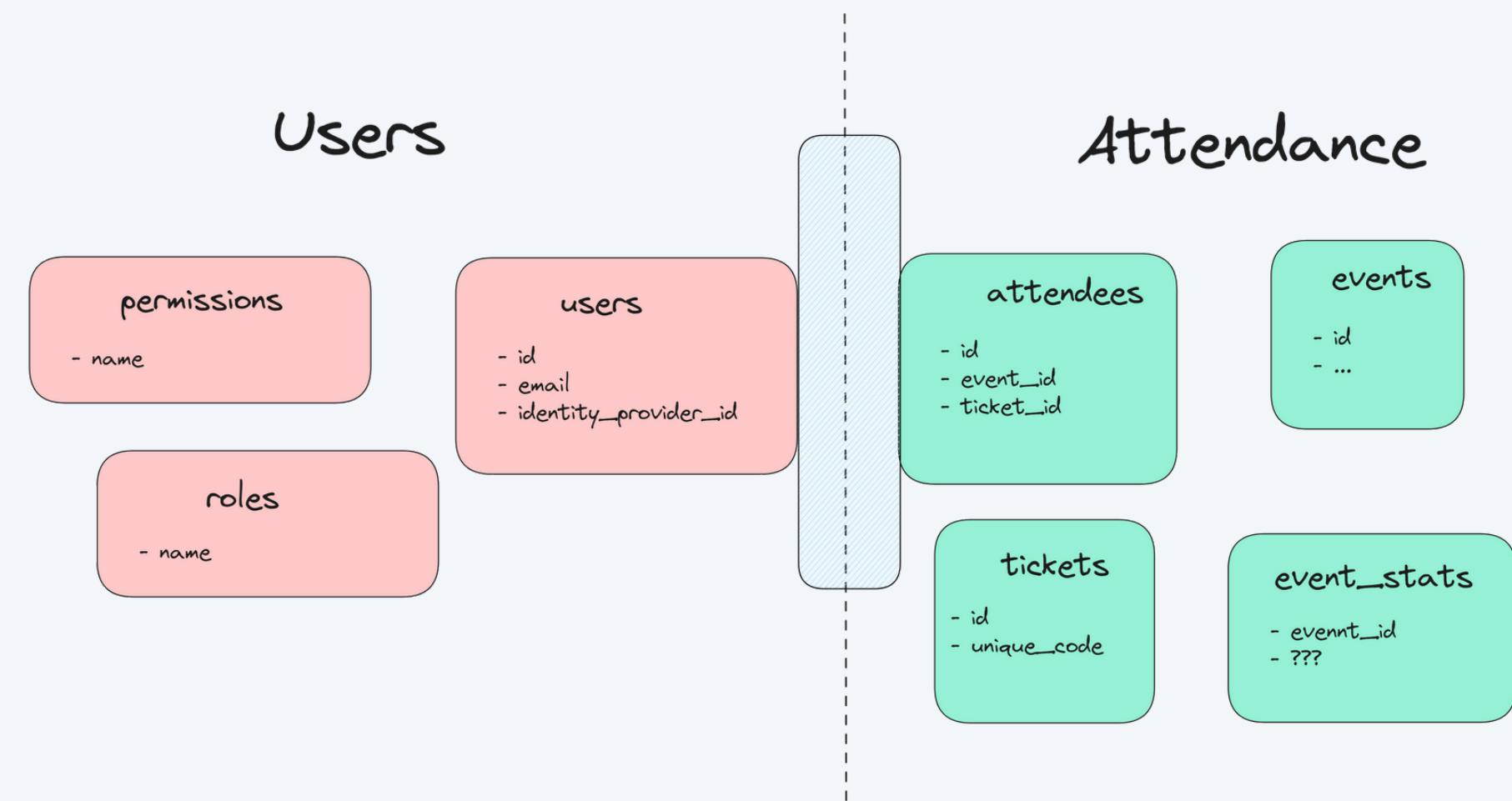
Great modularization concepts:

- Bounded Context
- Aggregate



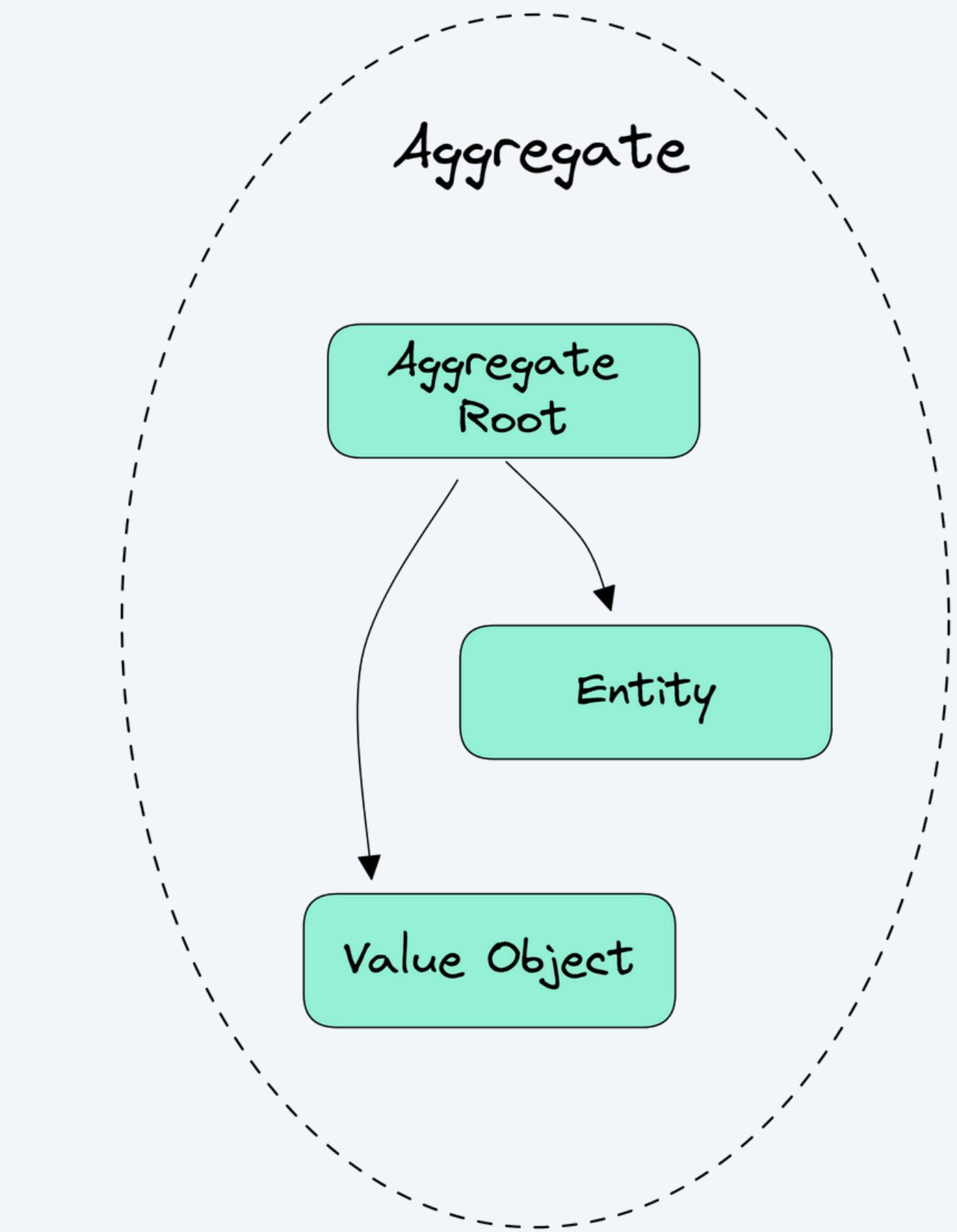
Bounded Context

The boundary within a domain where a particular domain model applies.



Aggregate

A cluster of related objects that form a consistency boundary and are treated as a single unit.



DDD Starter Modelling Process

Discover the domain visually and collaboratively.

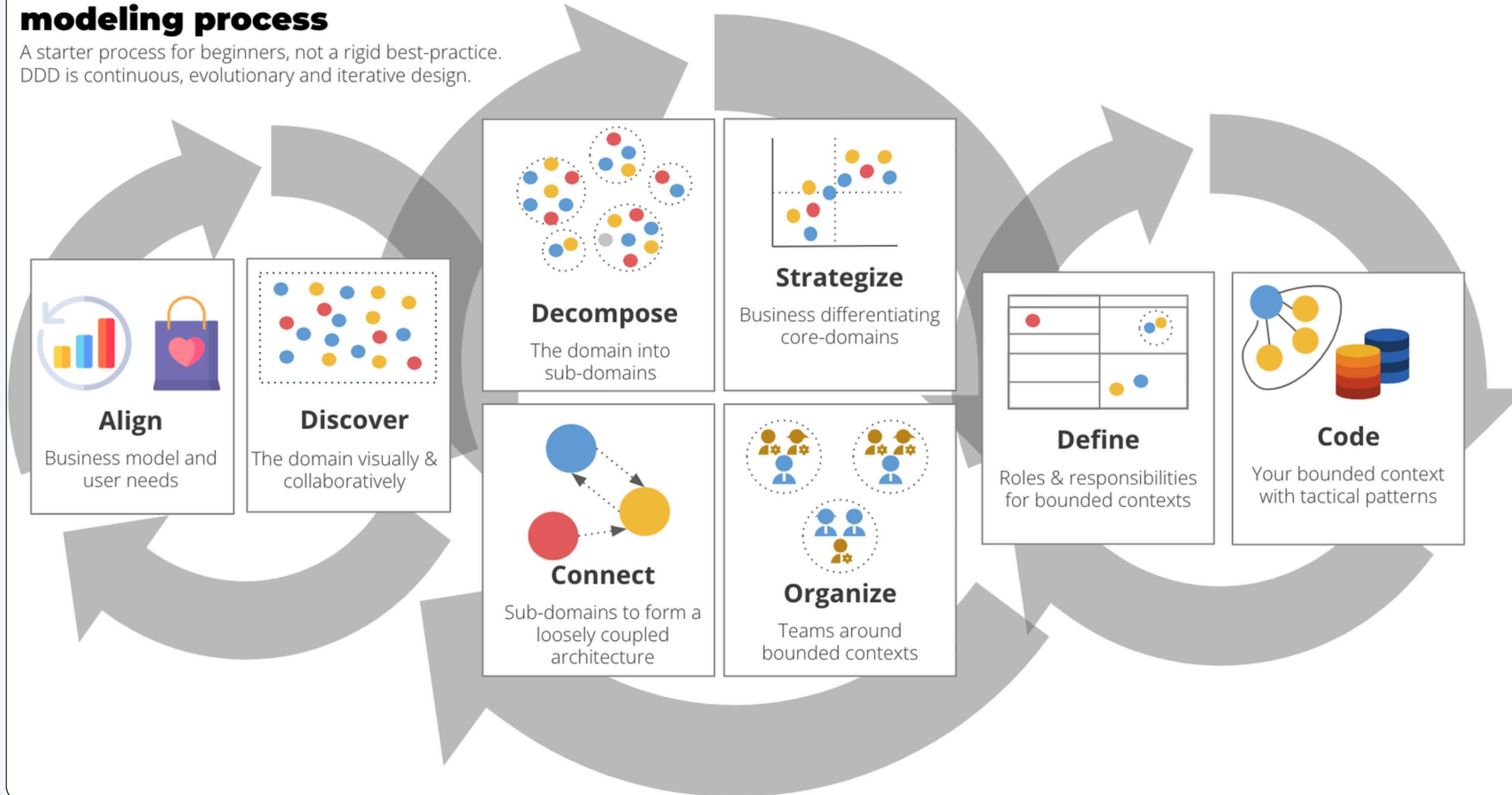
Decompose the domain into sub-domains - loosely coupled parts of the domain.

DDD Starter Modelling Process

Connect the sub-domains into a loosely coupled architecture that fulfills end-to-end business use cases.

Domain-Driven Design starter modeling process

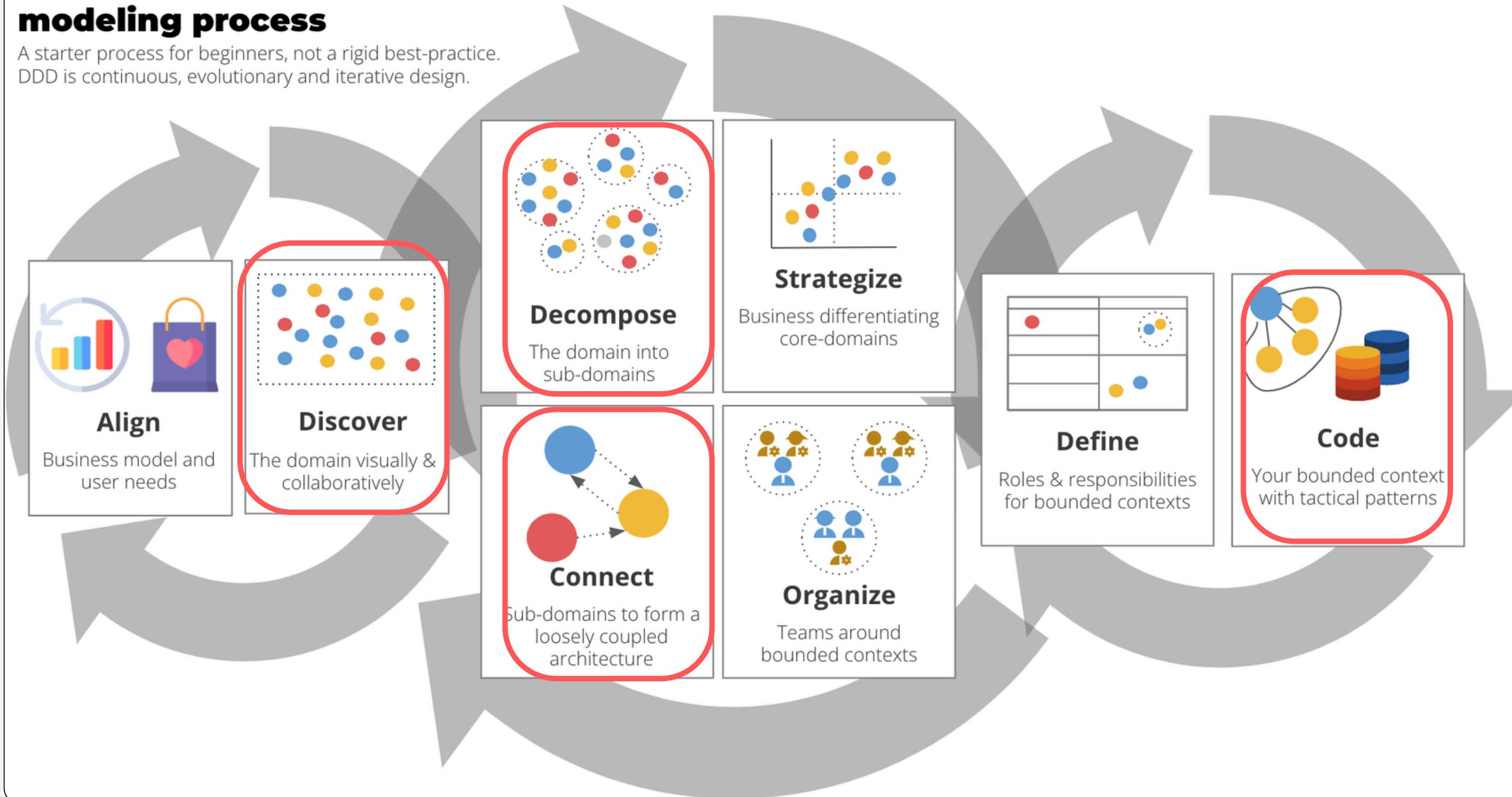
A starter process for beginners, not a rigid best-practice.
DDD is continuous, evolutionary and iterative design.



<https://github.com/ddd-crew/ddd-starter-modelling-process>

Domain-Driven Design starter modeling process

A starter process for beginners, not a rigid best-practice.
DDD is continuous, evolutionary and iterative design.



<https://github.com/ddd-crew/ddd-starter-modelling-process>

Event Storming

Map out the domain and identify domain boundaries.

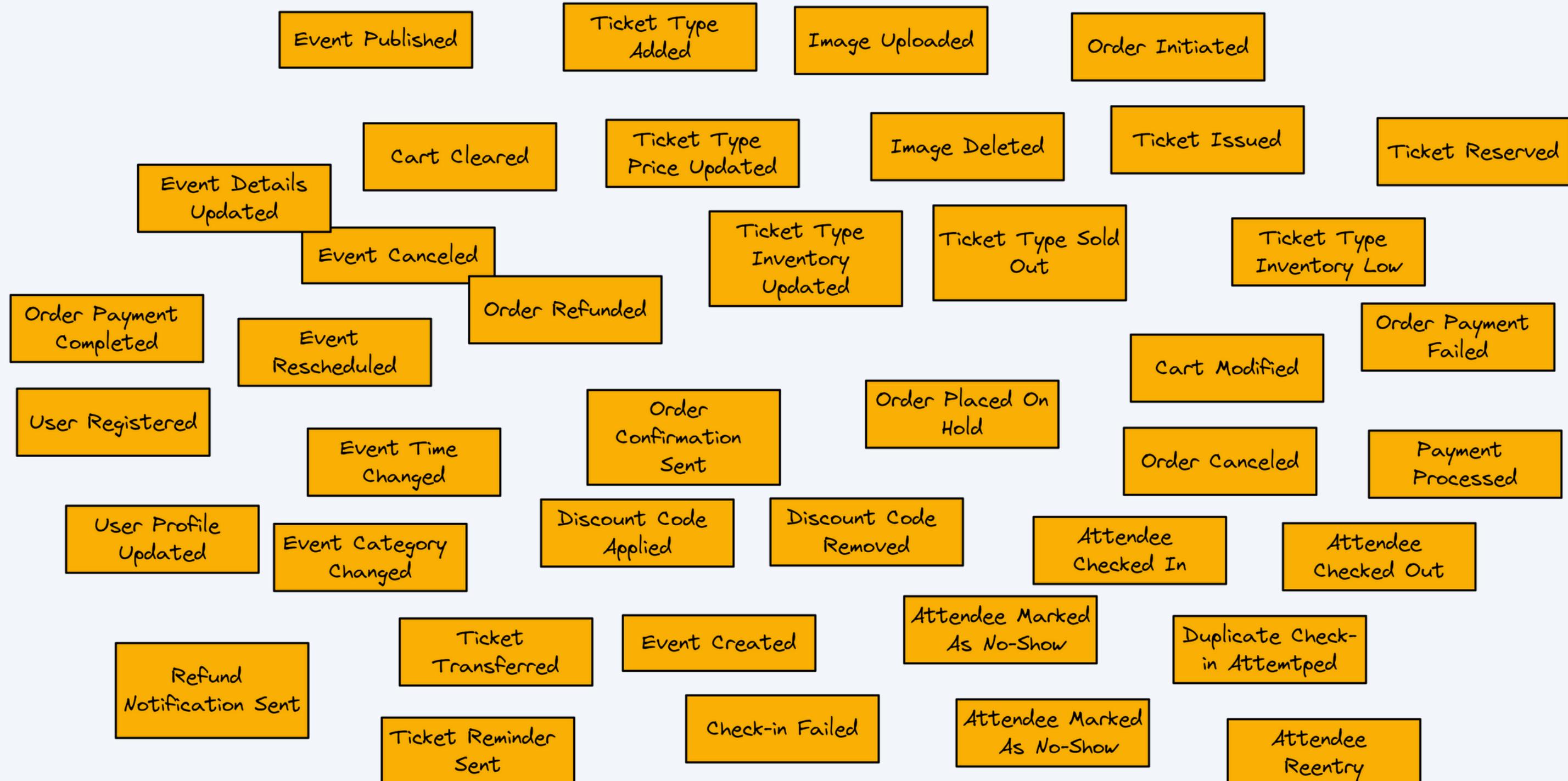
Write up the domain events using sticky notes.

Behavior and events that occur in our domain are very important.

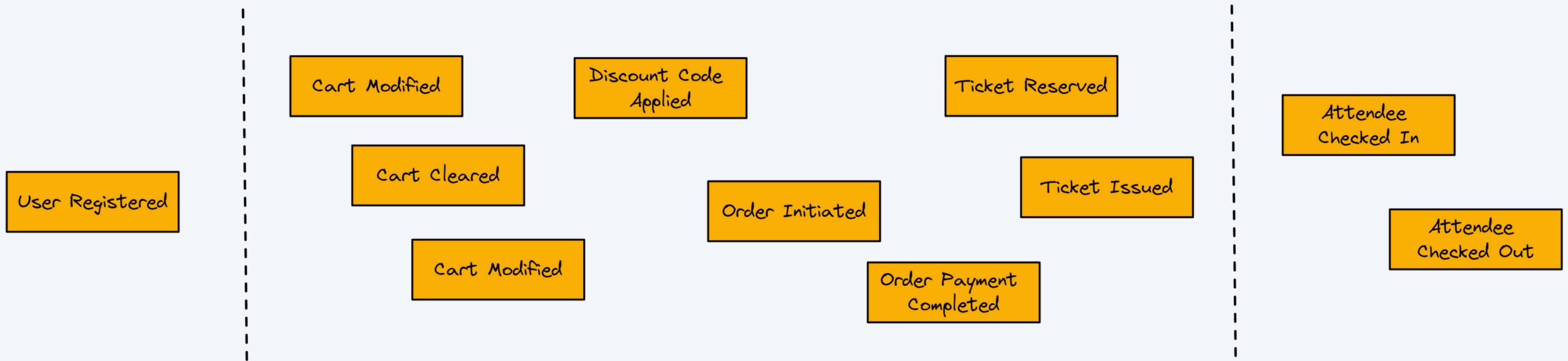


<https://github.com/ddd-crew/eventstorming-glossary-cheat-sheet>

A Tangled Mess?



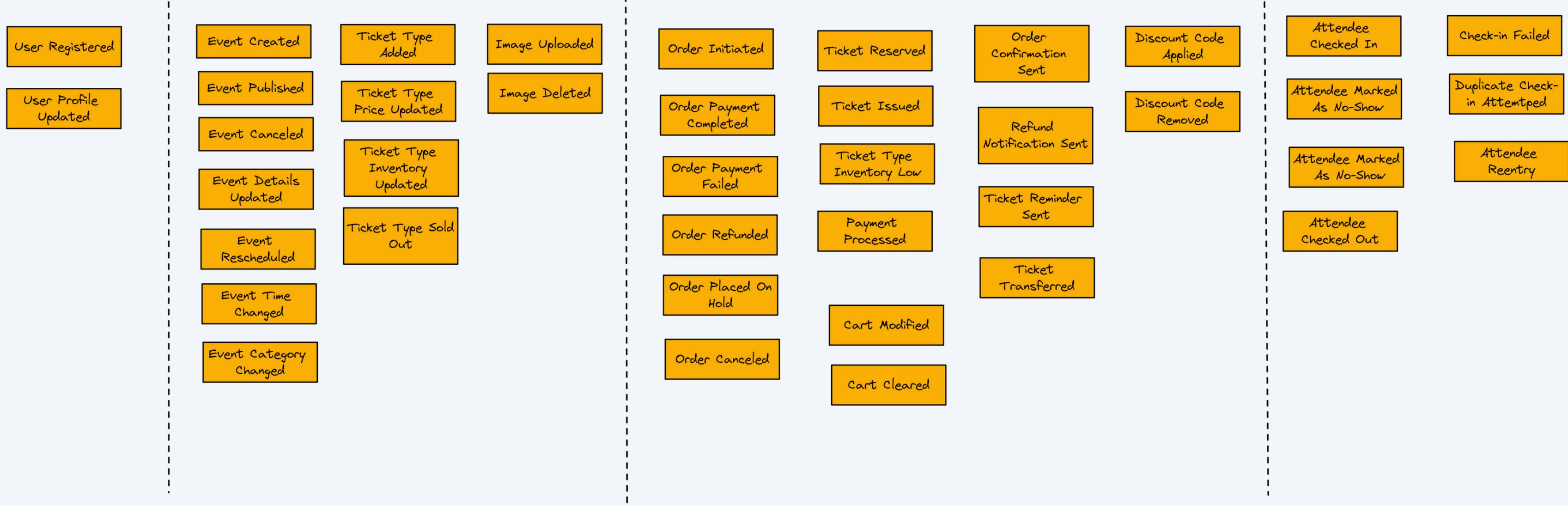
Enforcing the Timeline

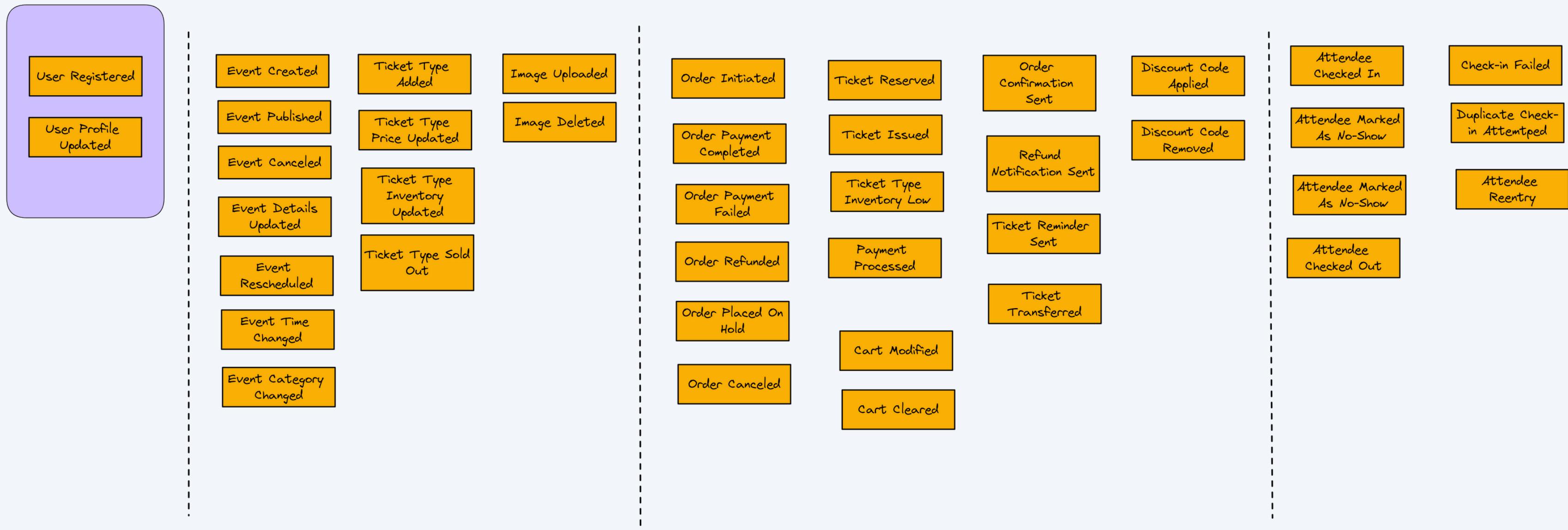


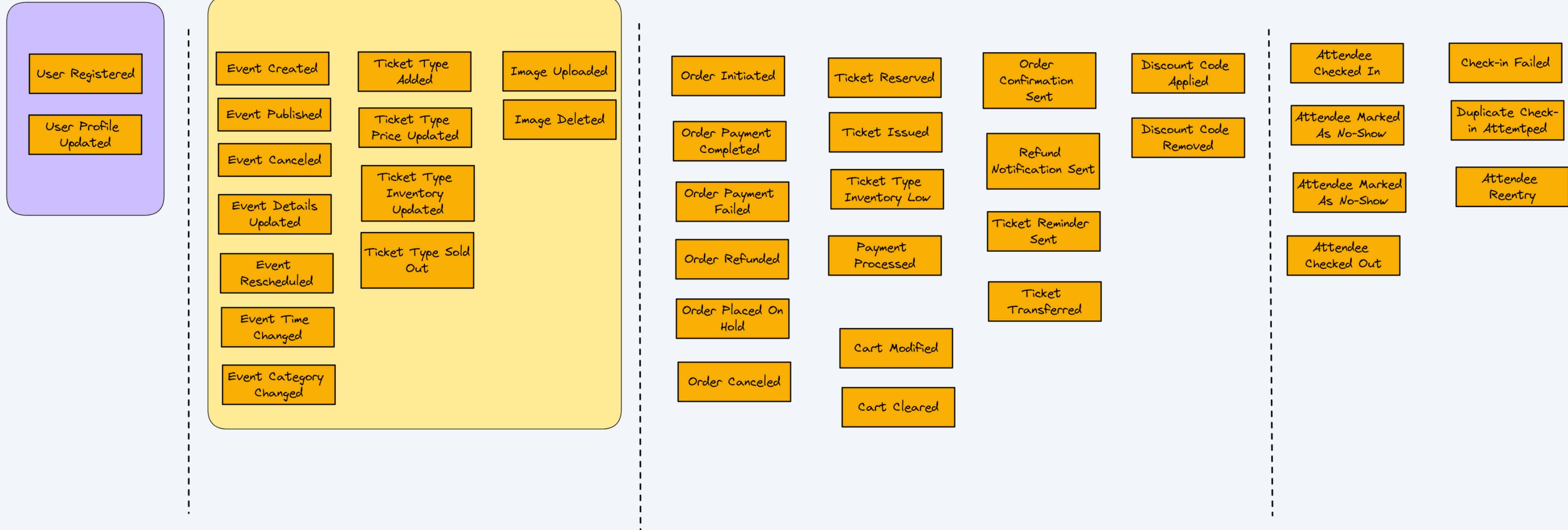
Event Storming - Continued

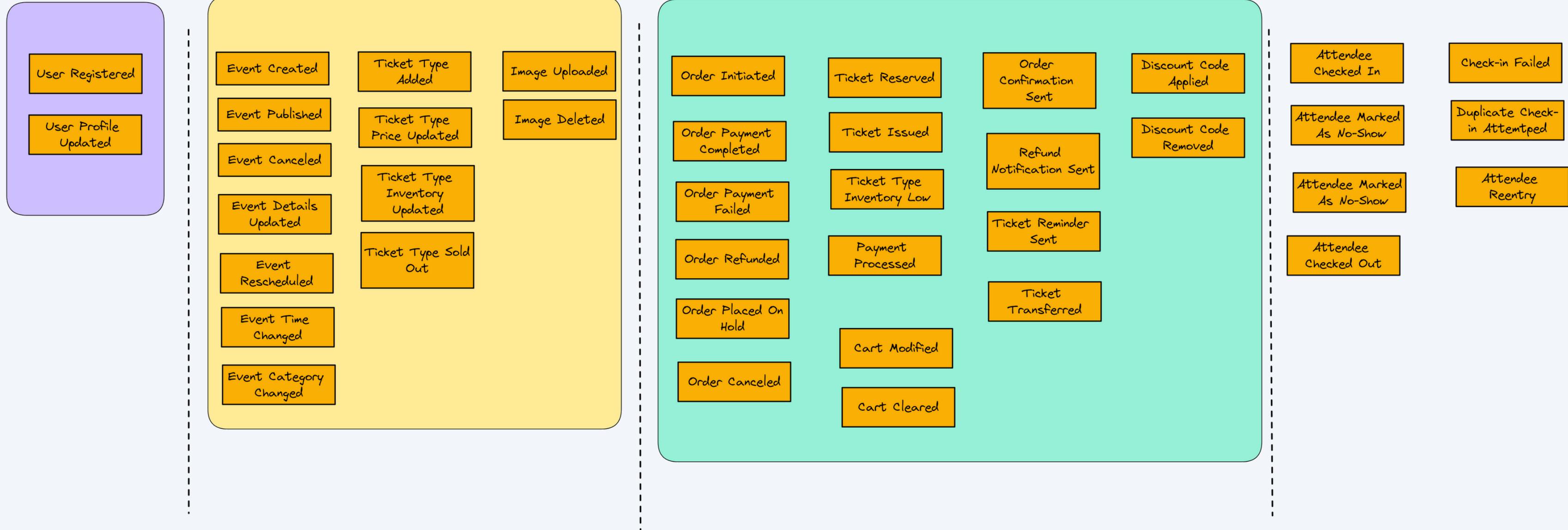
Organize the domain events on a timeline.

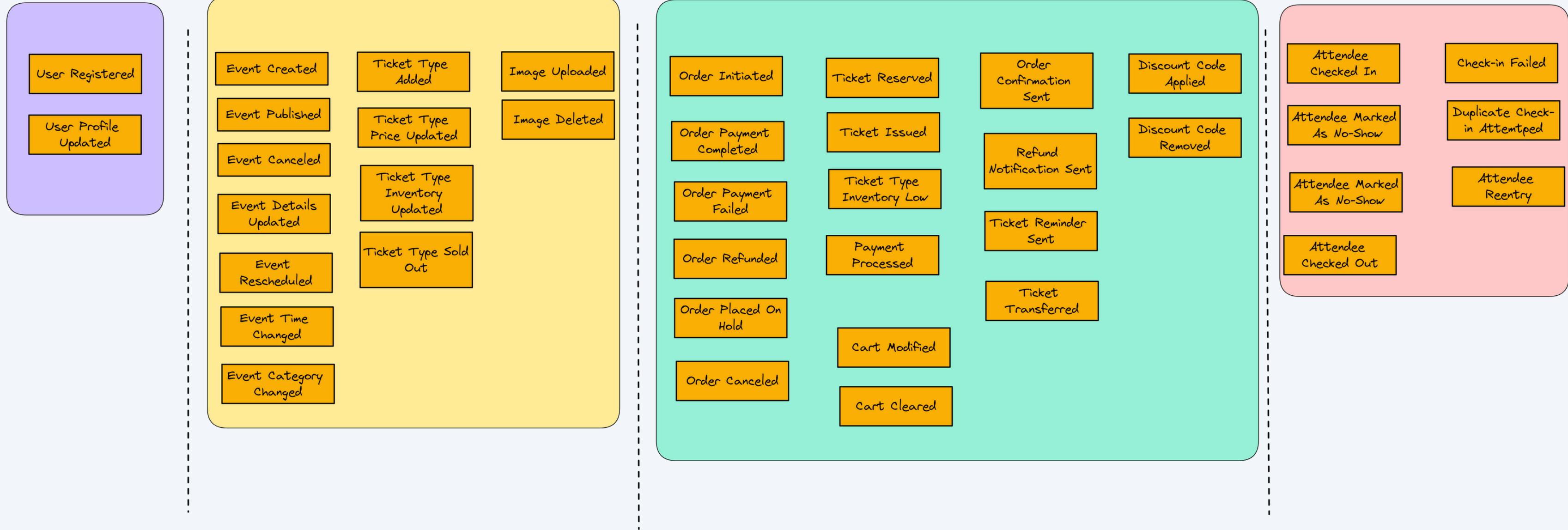
Explore potential domain boundaries.











Refinement

Defining domain boundaries is a continual process.

It's unlikely you will get it “right” from the start.

Bounded Contexts

The boundary within a domain where a particular domain model applies.

Users

users

- id
- email
- identity_provider_id

Users

users

- id
- email
- identity_provider_id

Ticketing

customers

- id
- email

Users

users

- id
- email
- identity_provider_id

Ticketing

customers

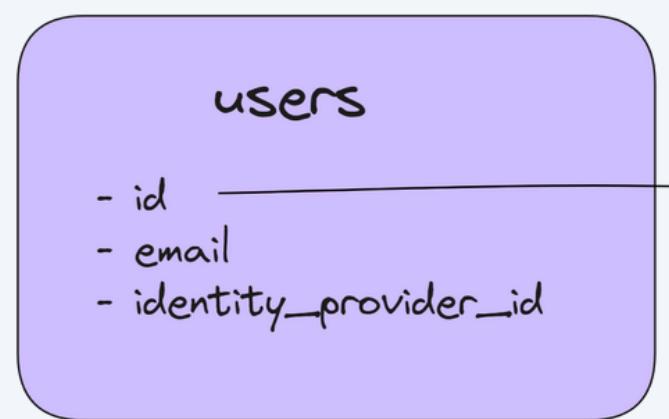
- id
- email

Attendance

attendees

- id
- event_id
- ticket_id

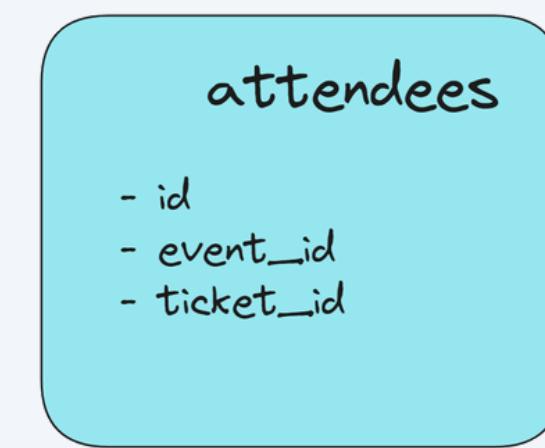
Users



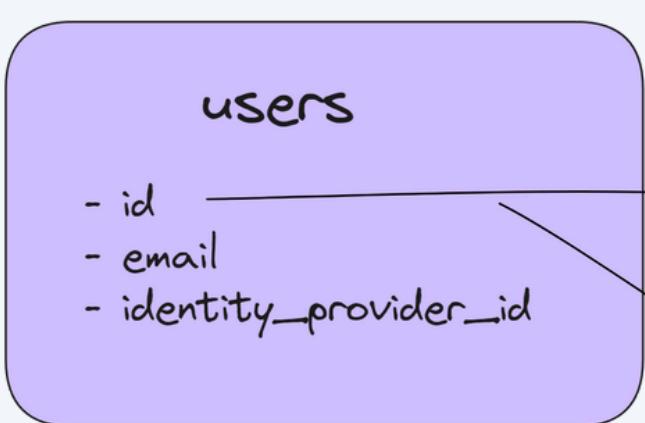
Ticketing



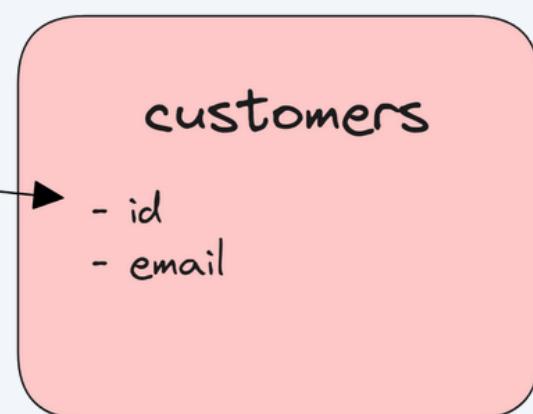
Attendance



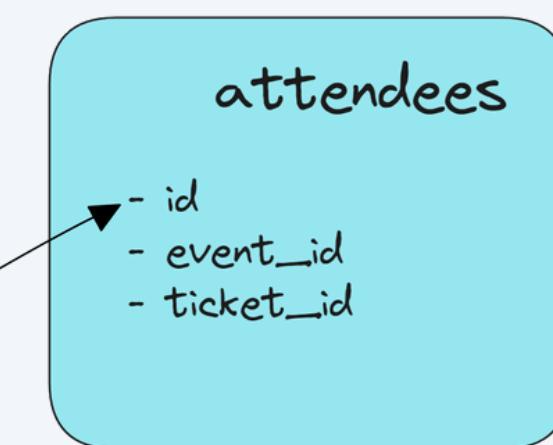
Users



Ticketing



Attendance



Users

permissions

- name

roles

- name

users

- id
- email
- identity_provider_id

Attendance

attendees

- id
- event_id
- ticket_id

tickets

- id
- unique_code

events

- id
- ...

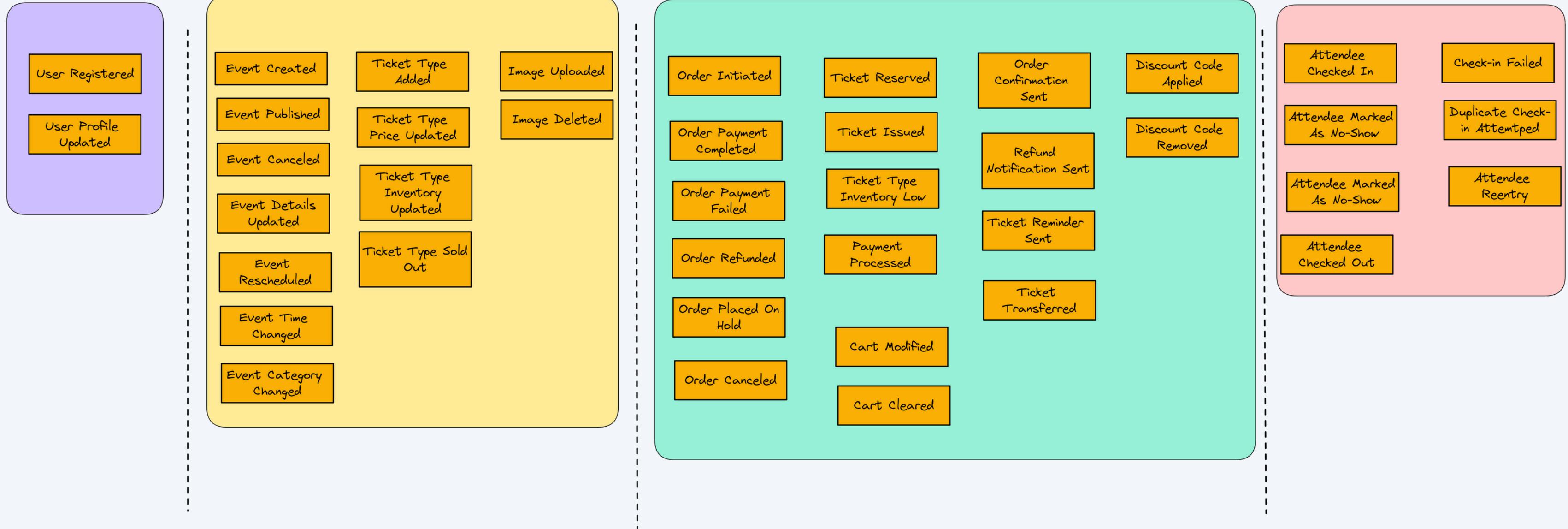
event_stats

- evennt_id
- ???

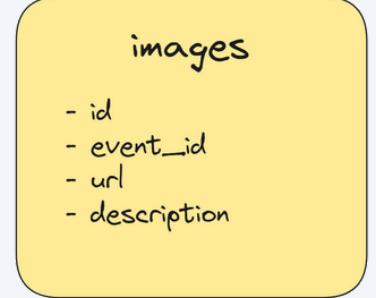
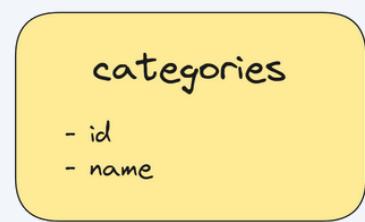
Defining Modules

Modules represent cohesive sets of functionalities.

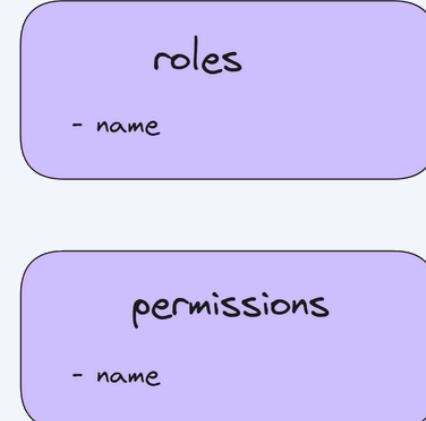
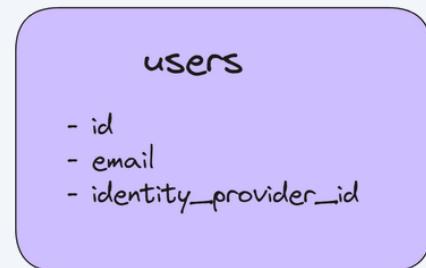
Use Bounded contexts to identify potential modules.



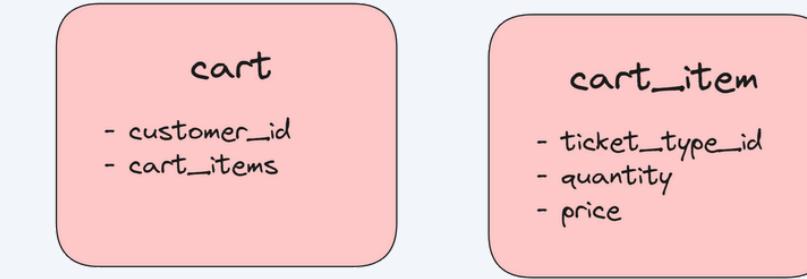
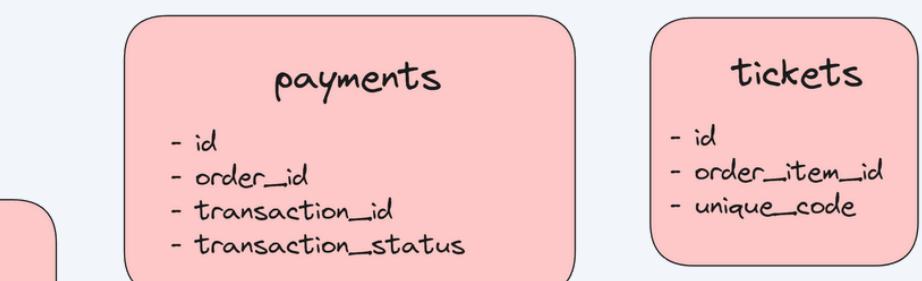
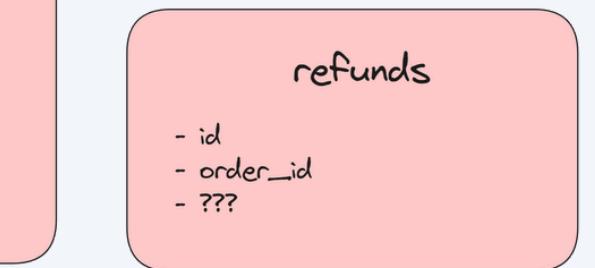
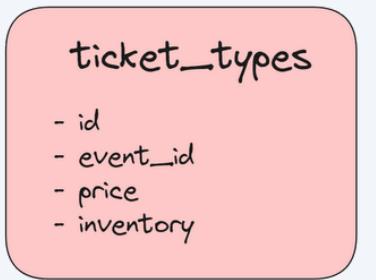
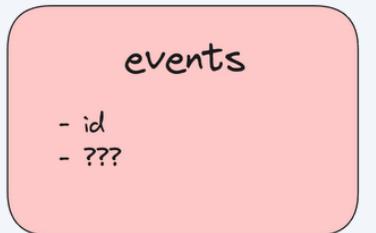
Events



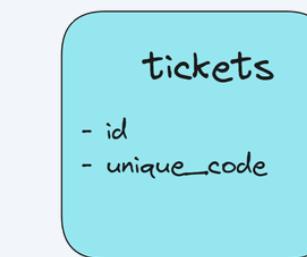
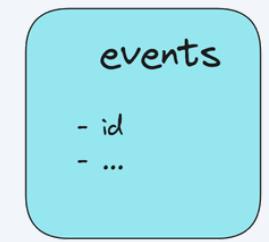
Users



Ticketing



Attendance



Considerations

Getting the module boundaries “right” is difficult but important.

- How often will modules communicate?
- How will you share data between modules?
- How will you deal with eventual consistency?

Considerations

Spend more time defining module boundaries

- it will pay dividends later.

Consider merging “chatty” modules.



Next:
Introducing Evently
Sample Application



Introducing Evently Sample Application

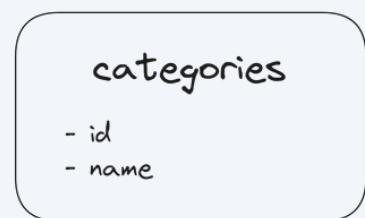
Event Management System

The modules:

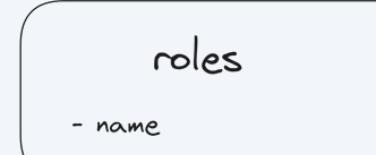
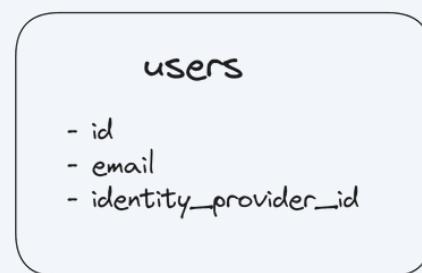
- Users
- Events
- Ticketing
- Attendance

* This application and its features are entirely fictional. Any resemblance to existing software is purely coincidental.

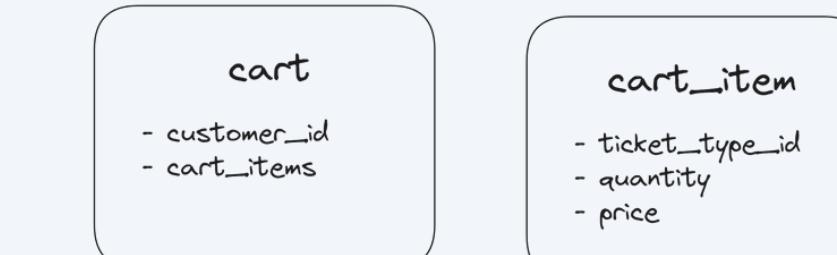
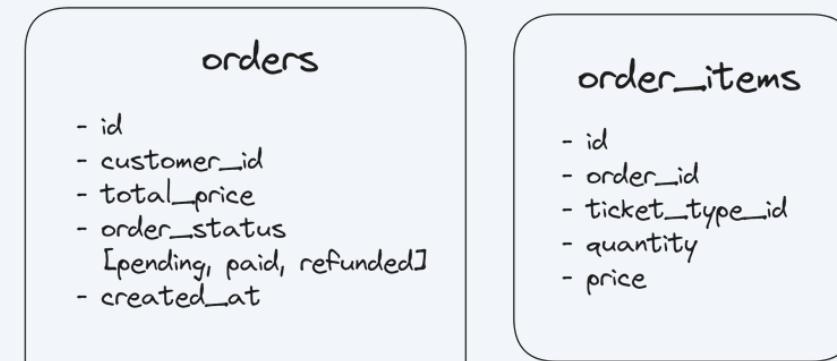
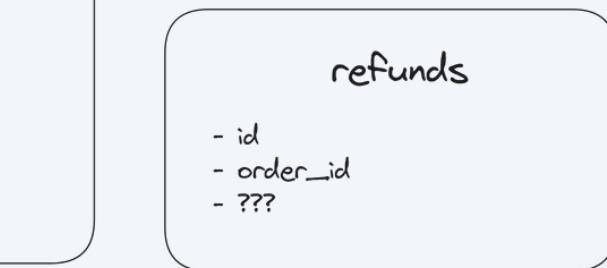
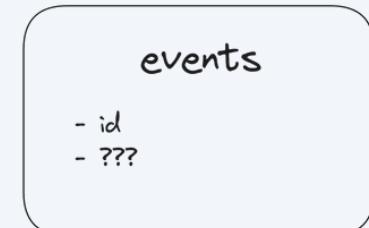
Events



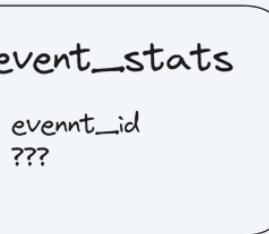
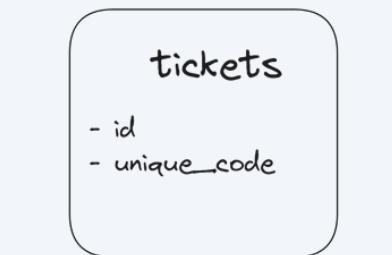
Users



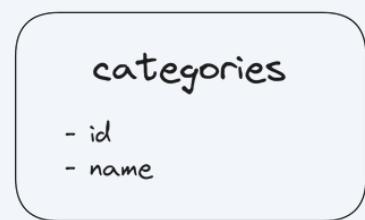
Ticketing



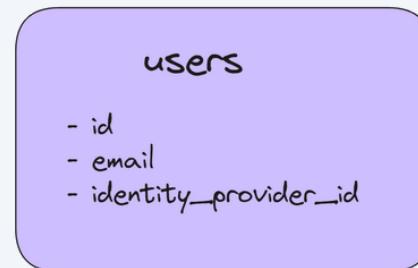
Attendance



Events



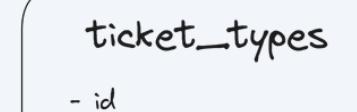
Users



Ticketing



- id
- ???



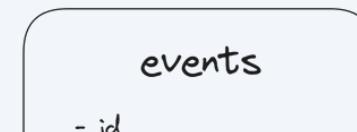
- id
- event_id
- price
- inventory



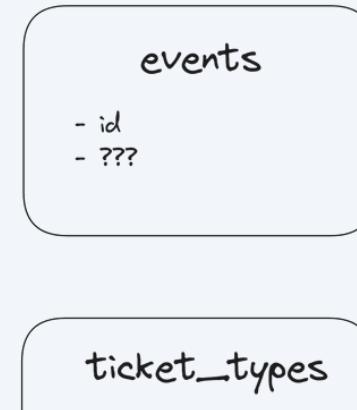
- id
- code
- event_id
- discount_type
- discount_value
- expiration_date
- usage_limit



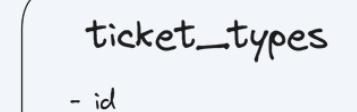
- id
- email



- id
- ???



- id
- customer_id
- total_price
- order_status [pending, paid, refunded]
- created_at



- id
- order_id
- ticket_type_id
- quantity
- price



- id
- order_id
- transaction_id
- transaction_status



- id
- order_id
- ???

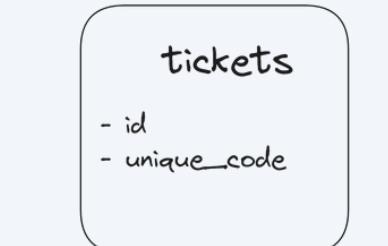


- customer_id
- cart_items

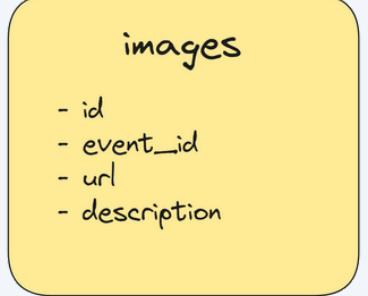


- ticket_type_id
- quantity
- price

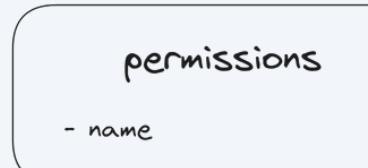
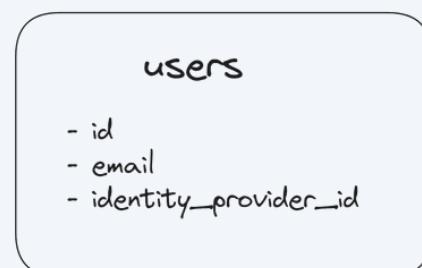
Attendance



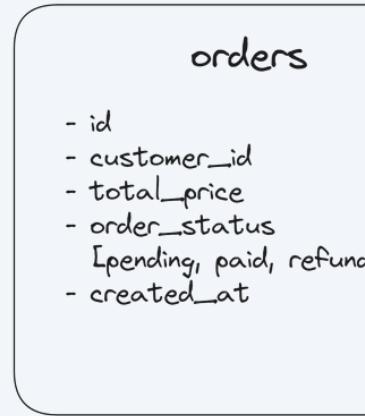
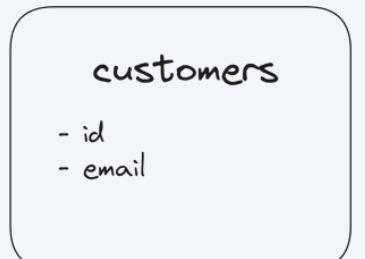
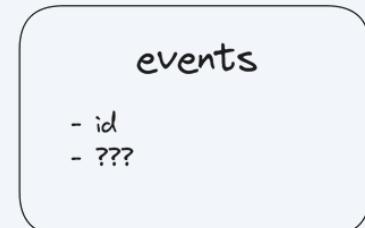
Events



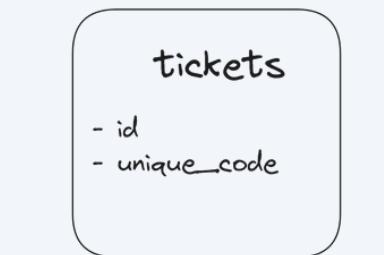
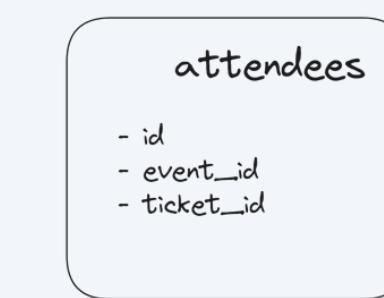
Users



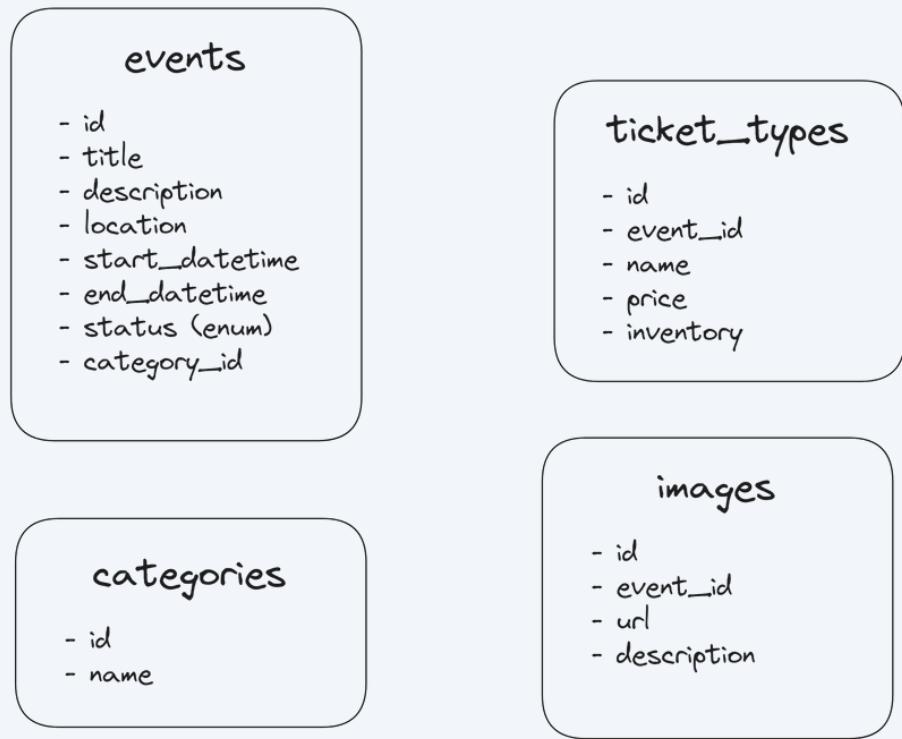
Ticketing



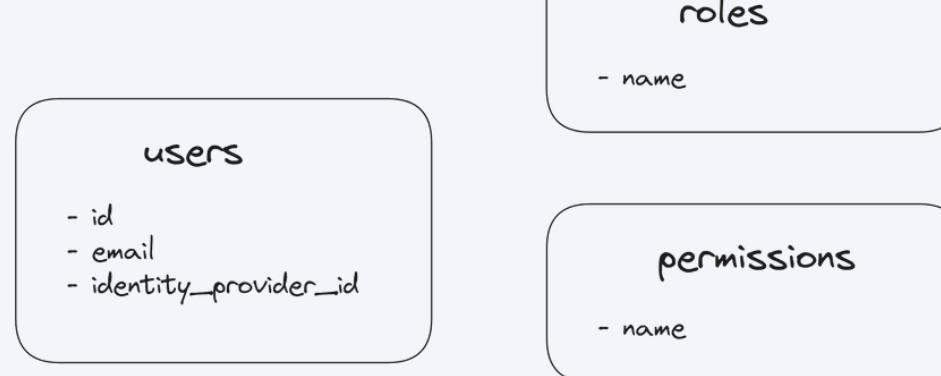
Attendance



Events



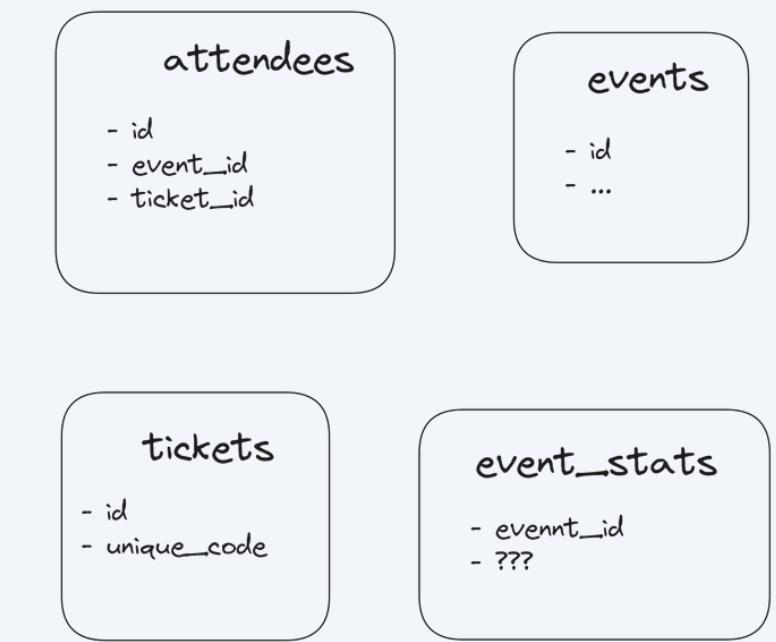
Users



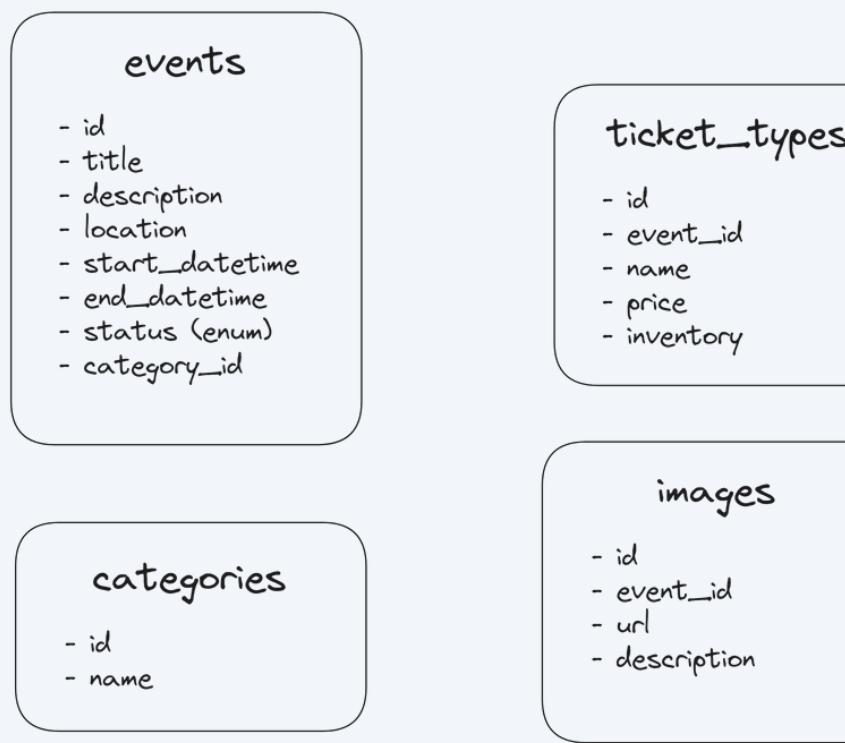
Ticketing



Attendance



Events



Ticketing



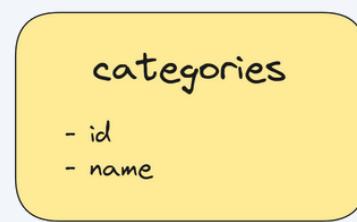
Attendance



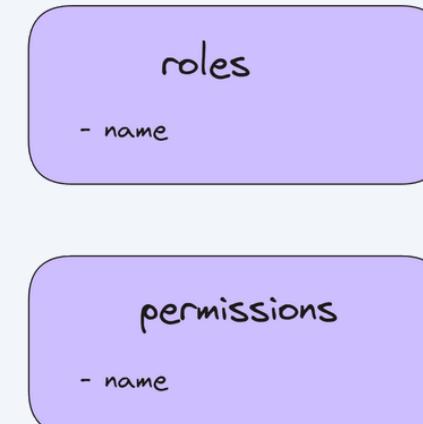
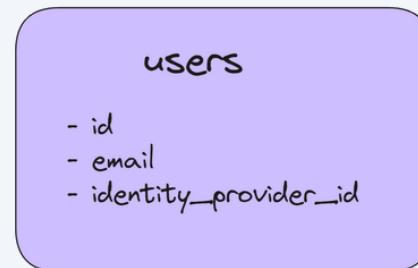
Users



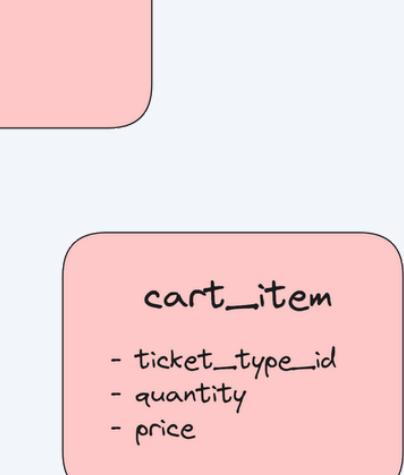
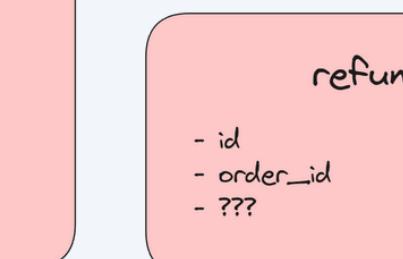
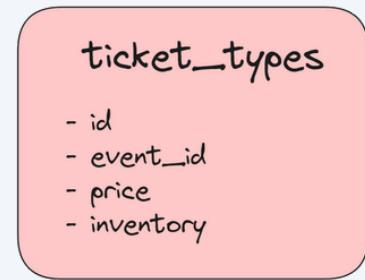
Events



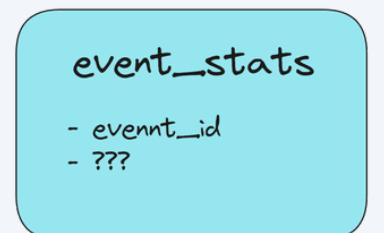
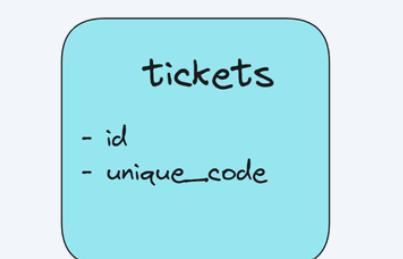
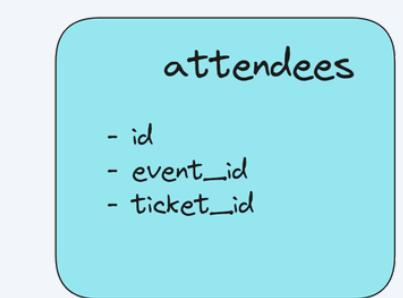
Users

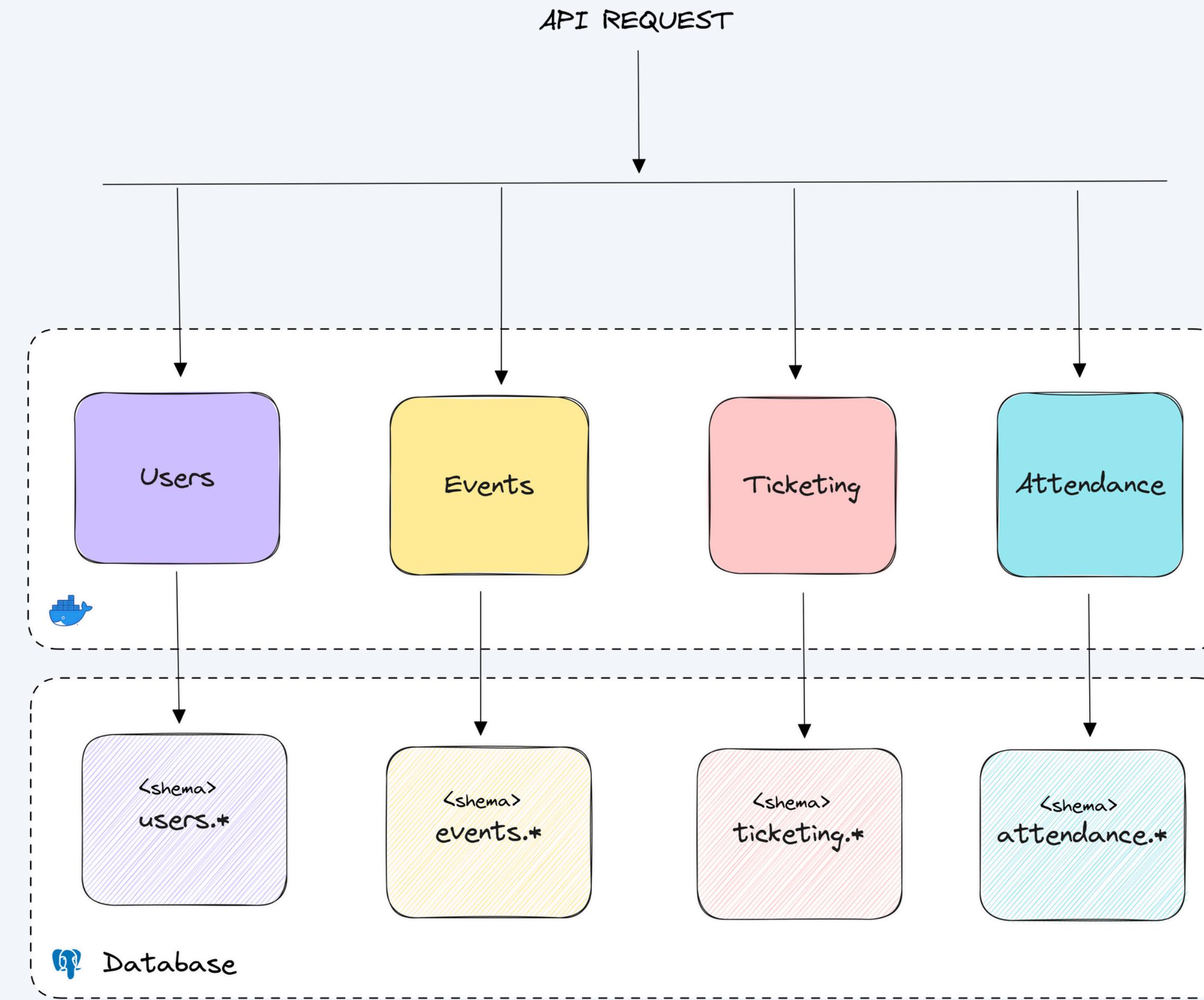


Ticketing



Attendance





Module Architecture

You can treat each module as a separate application.

Use the architectural style you think is the best fit.

Module Architecture

- Vertical Slice Architecture
- Hexagonal Architecture
- Clean Architecture

Module A

Presentation

Application

Domain

Infrastructure

Module B

Presentation

Application

Domain

Infrastructure

Module C

Presentation

Application

Domain

Infrastructure

Module A

Presentation

Application

Domain

Infrastructure

Module B

Api

Core

Module C

Application

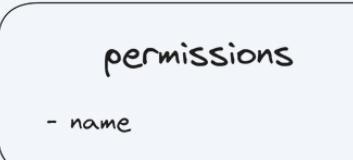
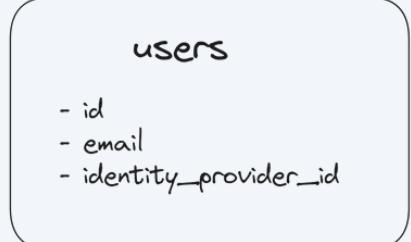
Business

Data

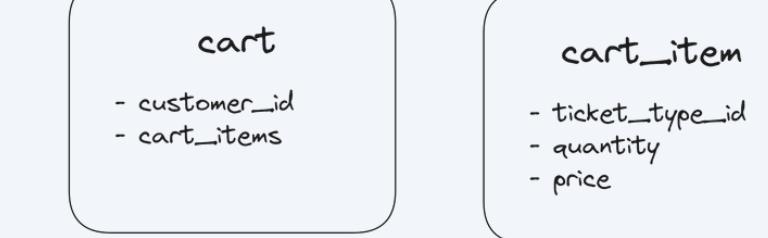
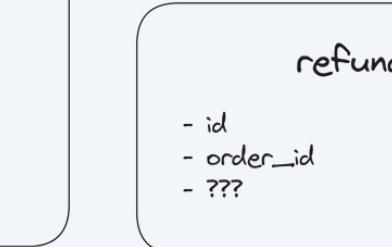
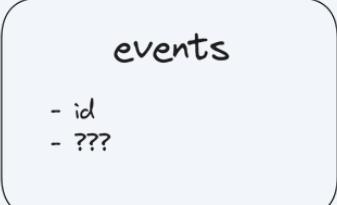
Events



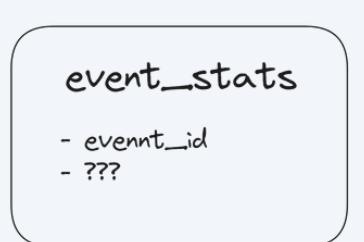
Users



Ticketing



Attendance





Next:
Building the First
Module



Building the First Module

Events



events

- id
- title
- description
- location
- start_datetime
- end_datetime
- status (enum)
- category_id

ticket_types

- id
- event_id
- name
- price
- inventory

categories

- id
- name

images

- id
- event_id
- url
- description



Next:
Refactoring to
Clean Architecture



Refactoring to Clean Architecture

Module A

Presentation

Application

Domain

Infrastructure

Module B

Api

Core

Module C

Application

Business

Data

Module A

Presentation

Application

Domain

Infrastructure

Module B

Presentation

Application

Domain

Infrastructure

Module C

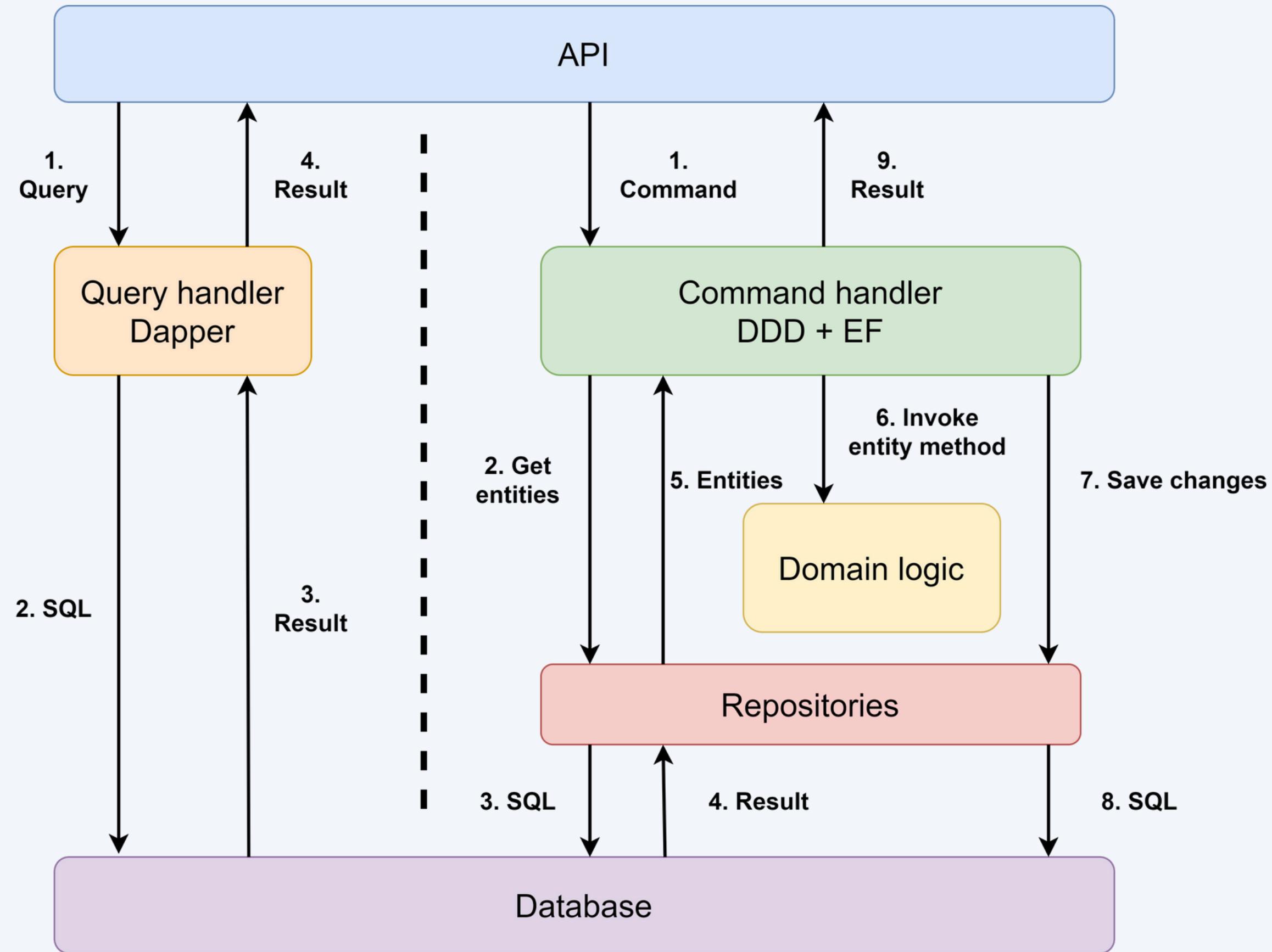
Presentation

Application

Domain

Infrastructure

CQRS





Next:
Events Module Review



Events Module Review



Next:
Module Cross-Cutting
Concerns

Module Cross-Cutting Concerns



Intro

Motivation for solving cross-cutting concerns.

Sharing code between modules.

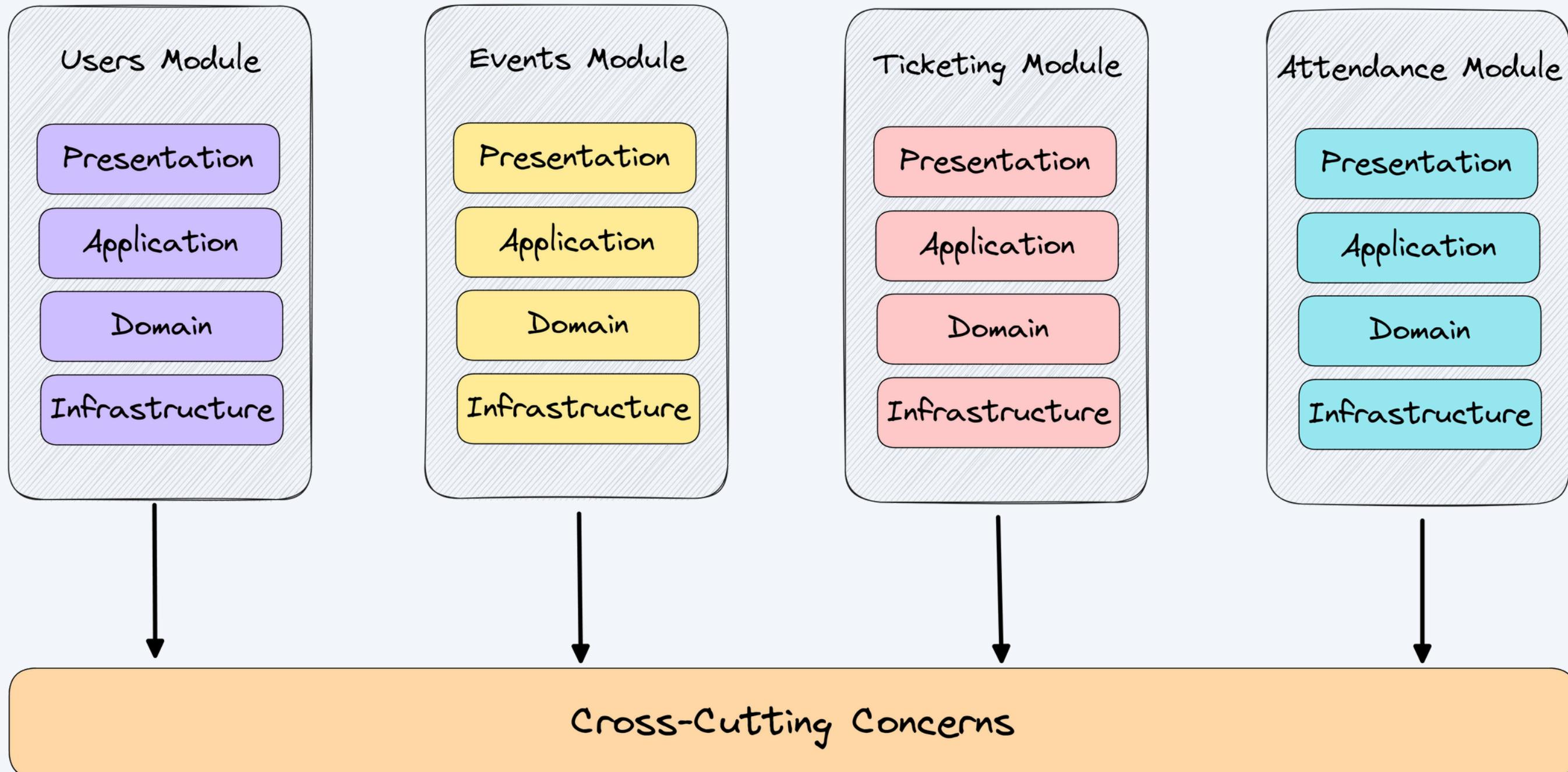
Cross-cutting concerns.

Motivation

Modules should be self-contained.

But we still want to avoid code duplication.

Make it faster to introduce new modules.



Sharing Code

What should we share between modules?

Duplicating code is sometimes necessary.

NuGet packages.

Cross-Cutting Concerns

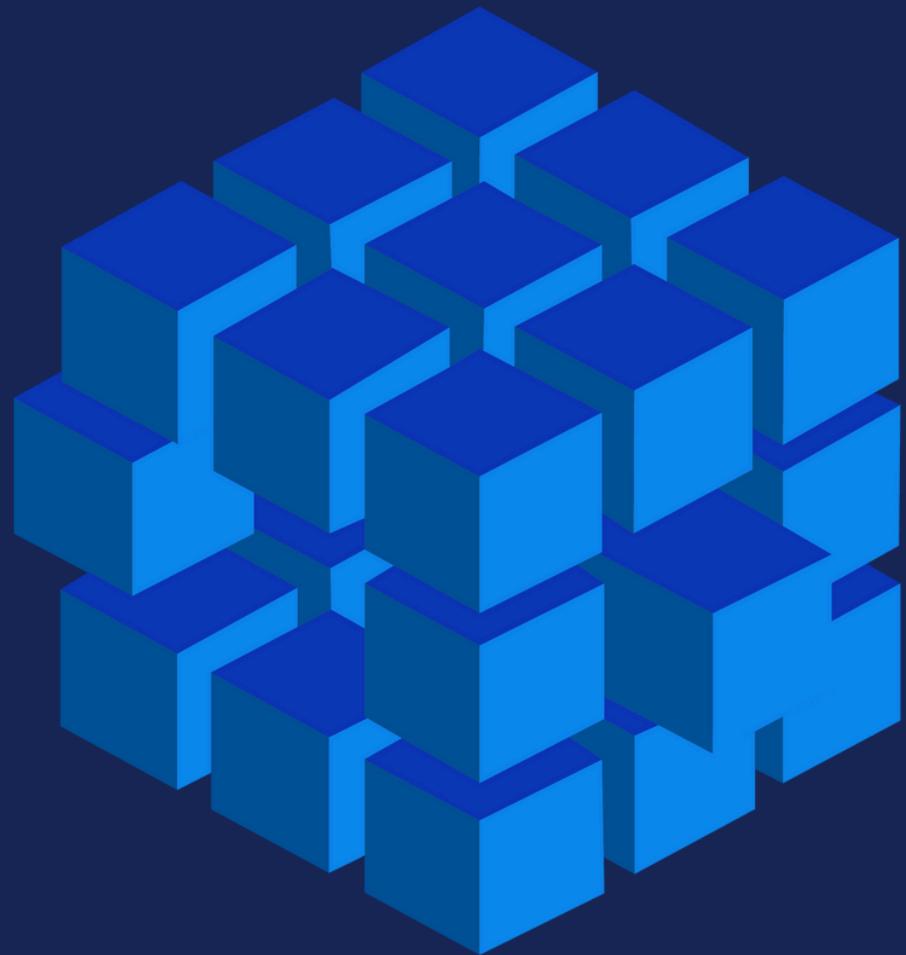
- Dependency injection
- Configuration
- Structured logging
- Exception handling
- Validation
- Caching
- Health checks
- And much more...



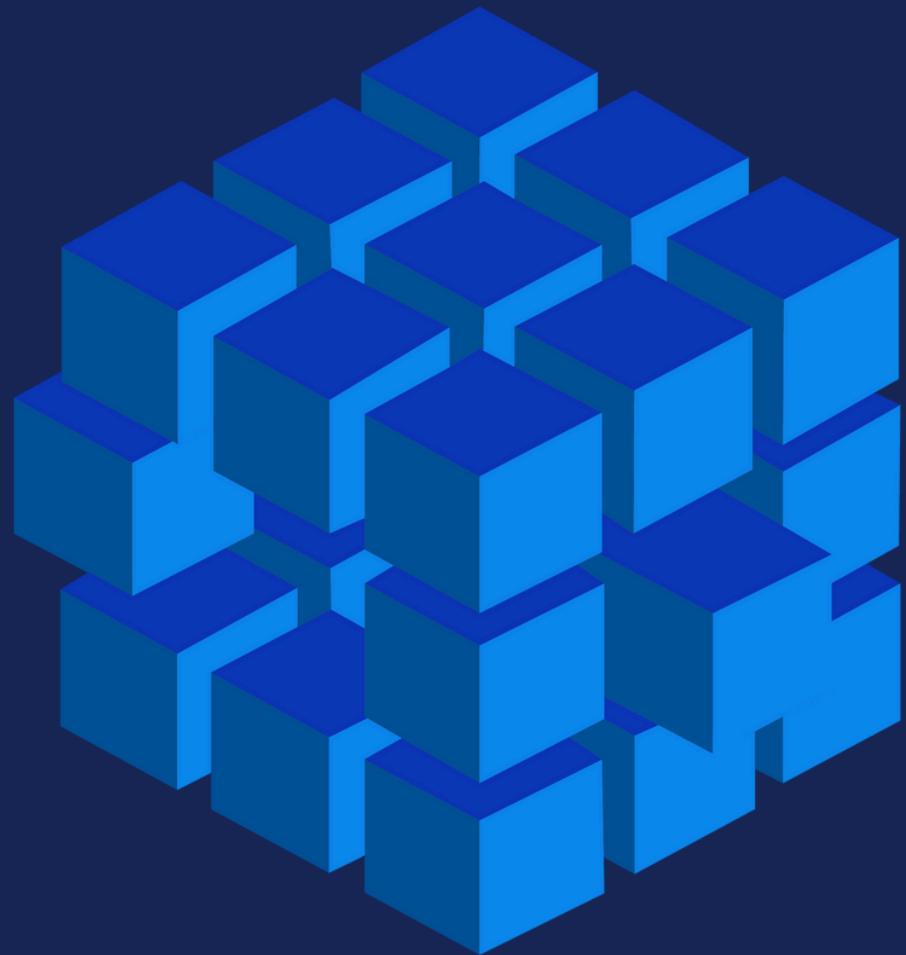
Next:
Dependency Injection



Dependency Injection



**Next:
Configuration**



Configuration



Next:
Structured Logging,
Serilog, Seq



Structured Logging, Serilog, Seq



Next:
Exception Handling



Exception Handling



**Next:
Validation**



Validation



Next:
Distributed Caching,
Redis



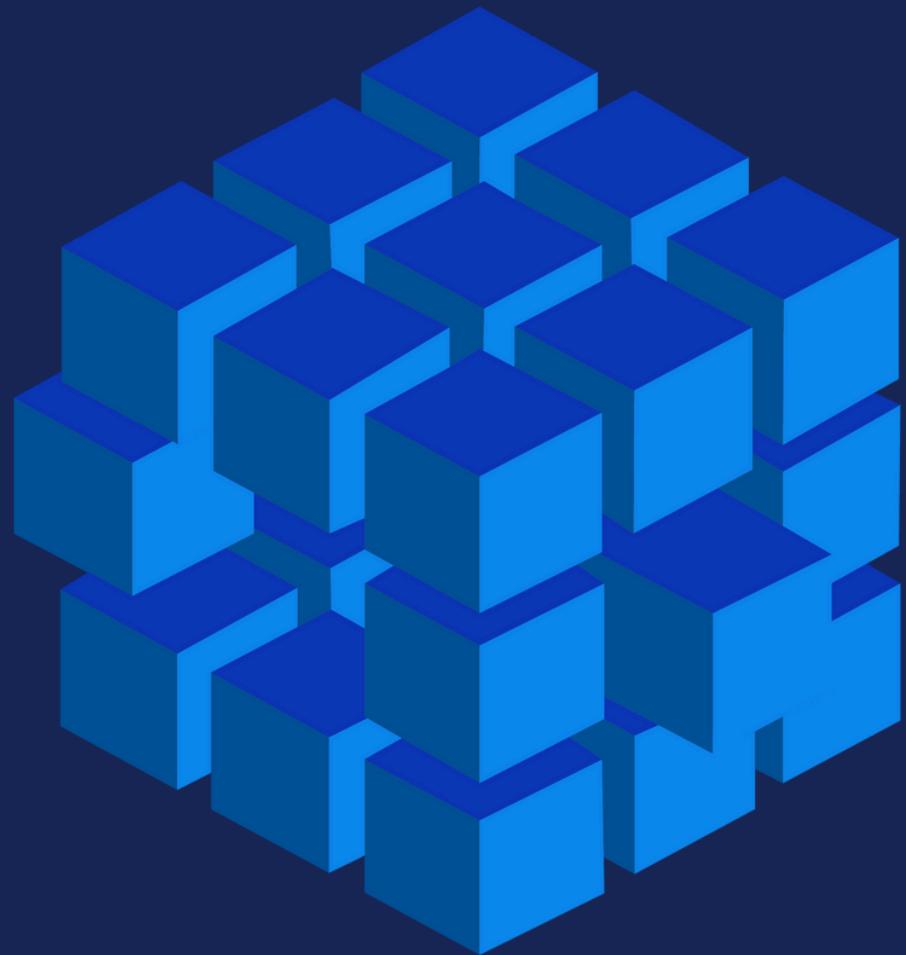
Distributed Caching, Redis



**Next:
Health Checks**



Health Checks



Next:
Automatically
Registering Endpoints

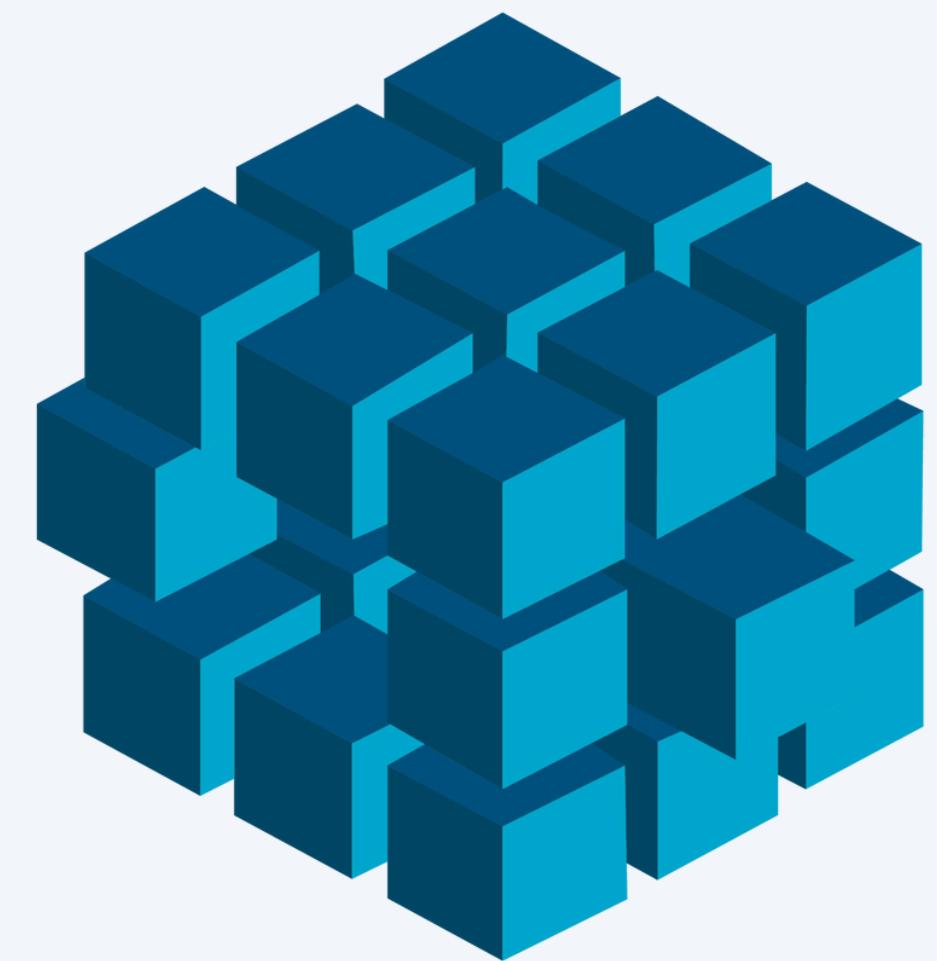


Automatically Registering Endpoints



**Next:
Module
Communication**

Module Communication



Intro

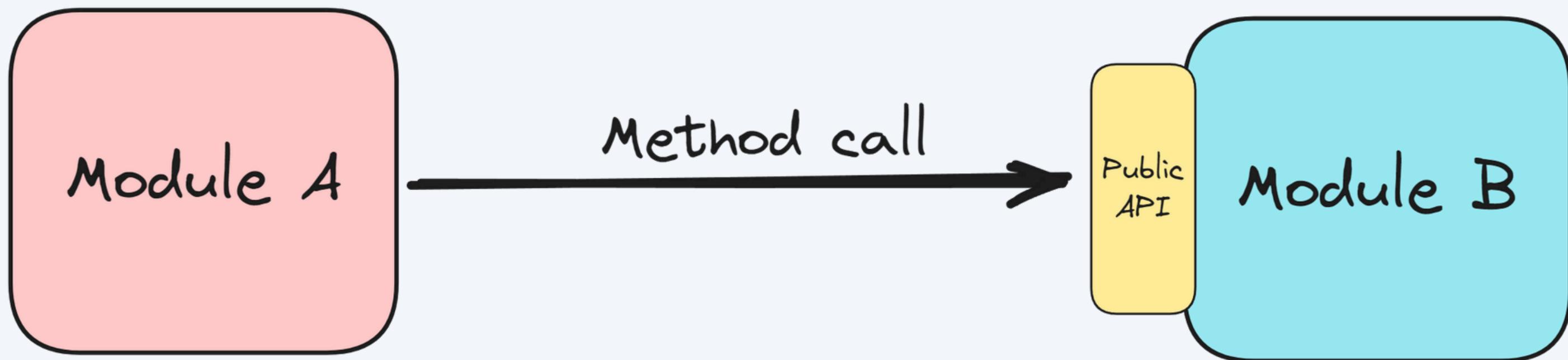
Synchronous communication.

Asynchronous communication.

Eventual consistency and reliable messaging.

Synchronous Communication

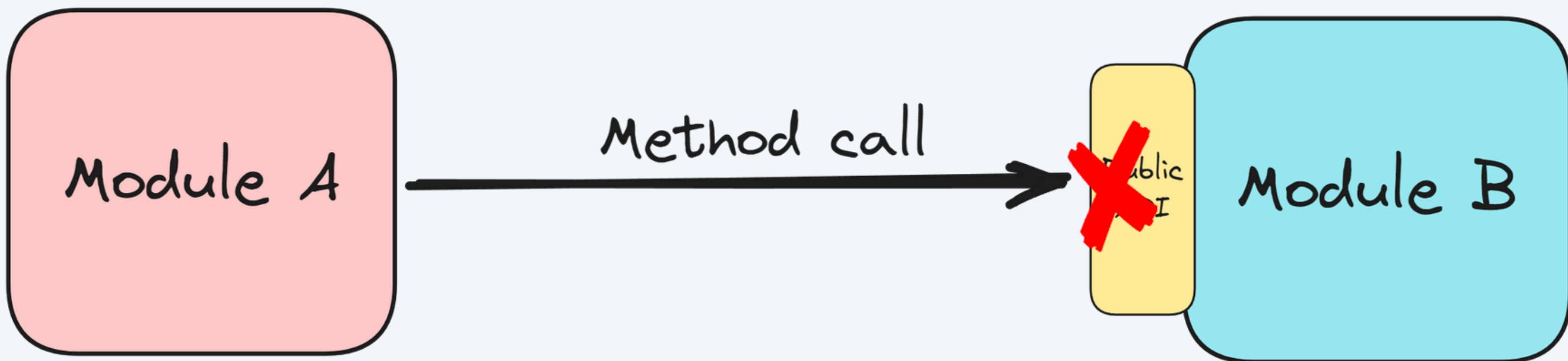
Method Calls



Method Calls

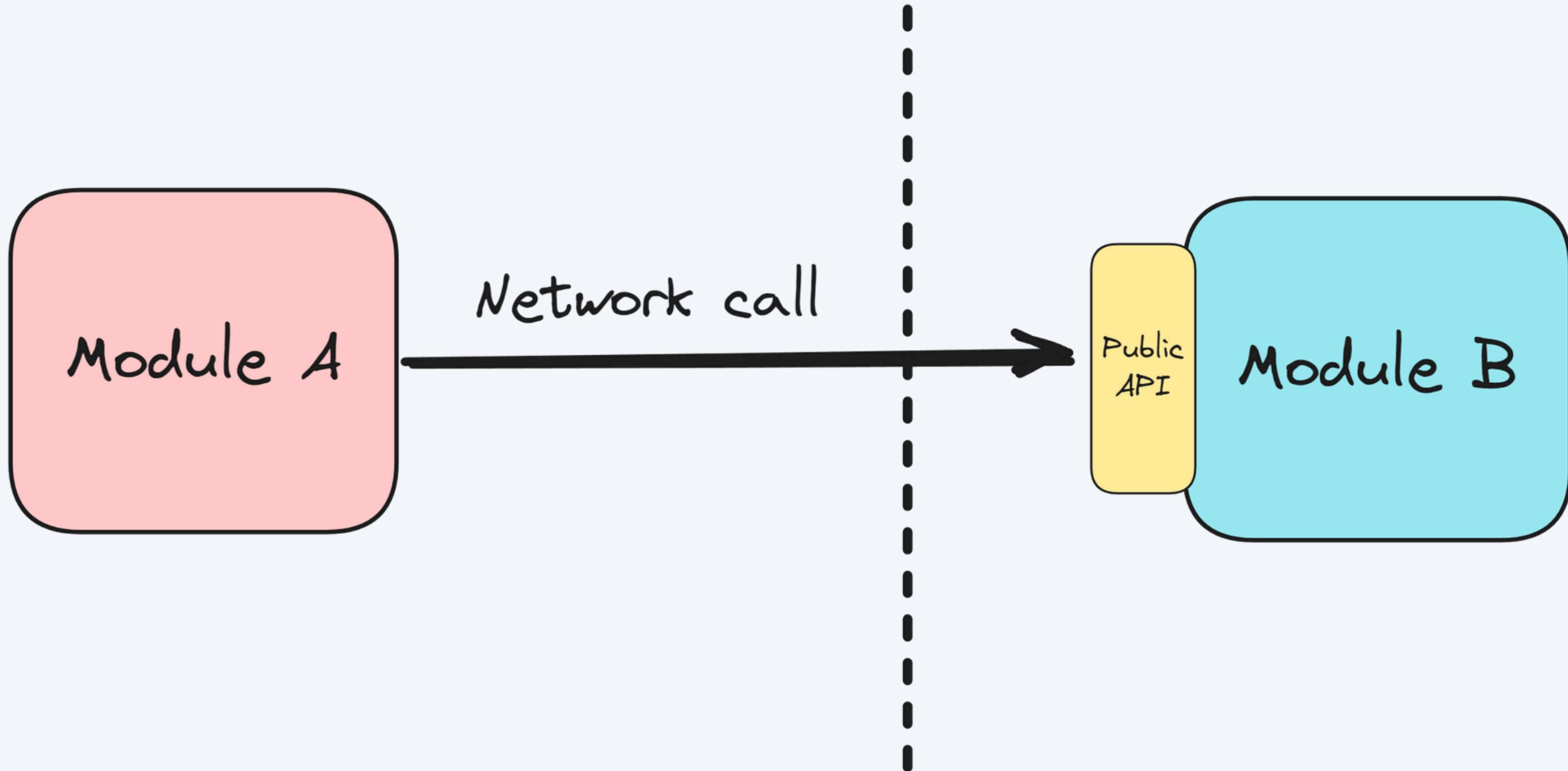
- Interface for public API
- Fast, in-memory calls
- Temporal coupling

Temporal Coupling



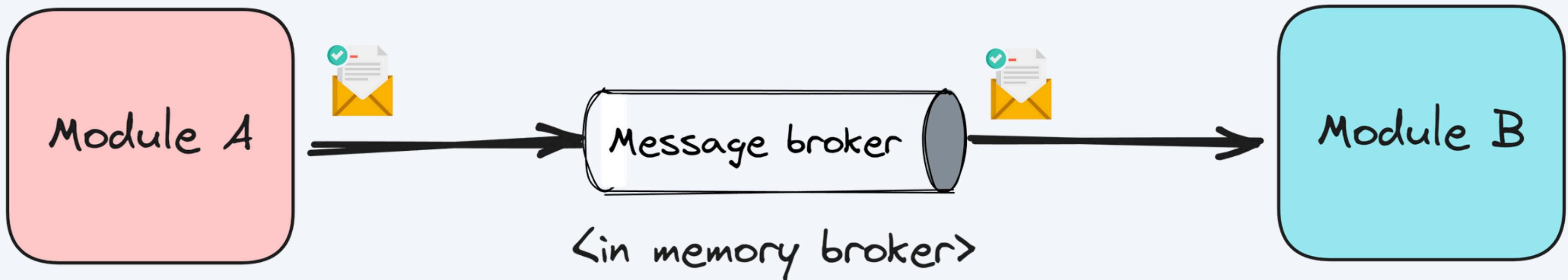
What if we are in a
distributed system?

Microservices



Asynchronous Communication

Messaging

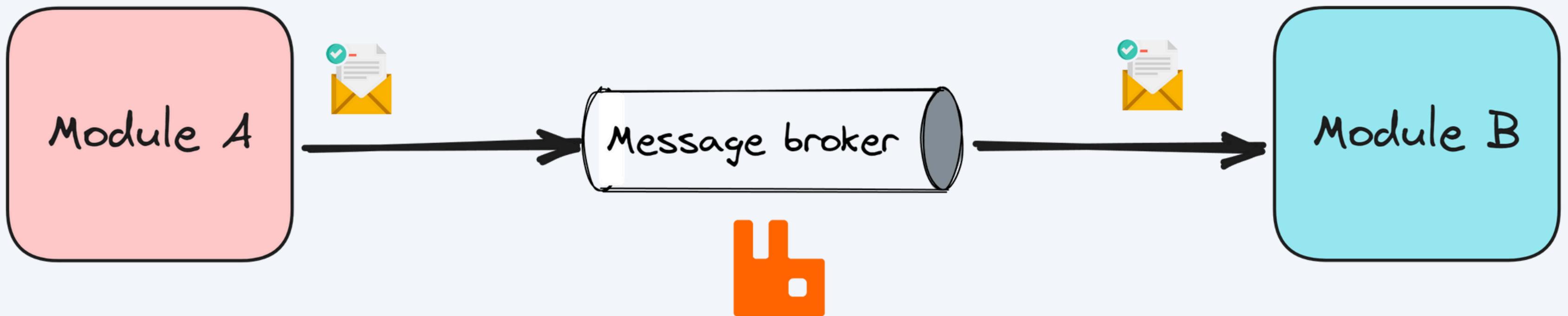


Messaging

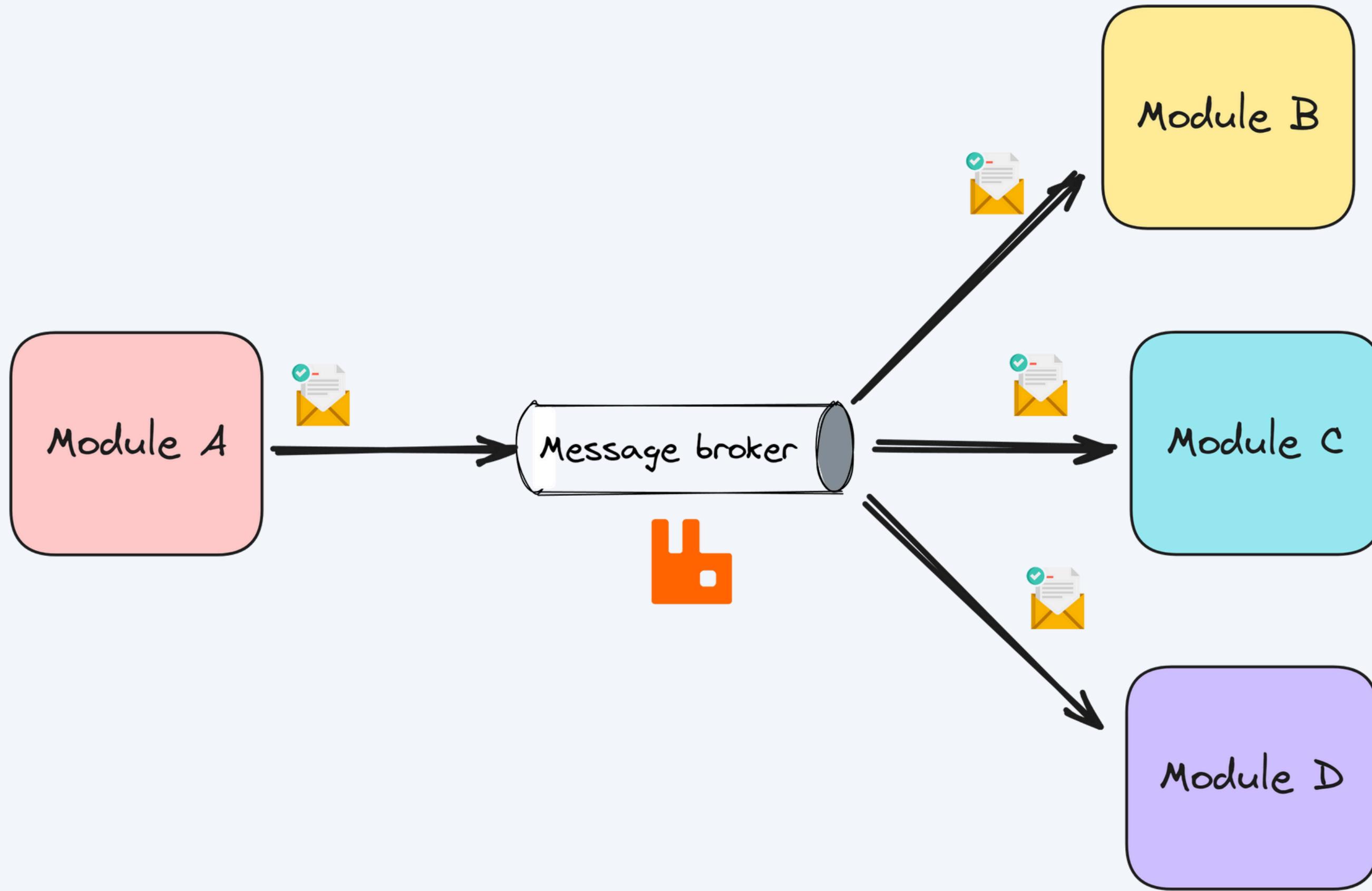
- Message contracts are the public API
- Asynchronous
- Loosely coupled

What if we are in a
distributed system?

Messaging

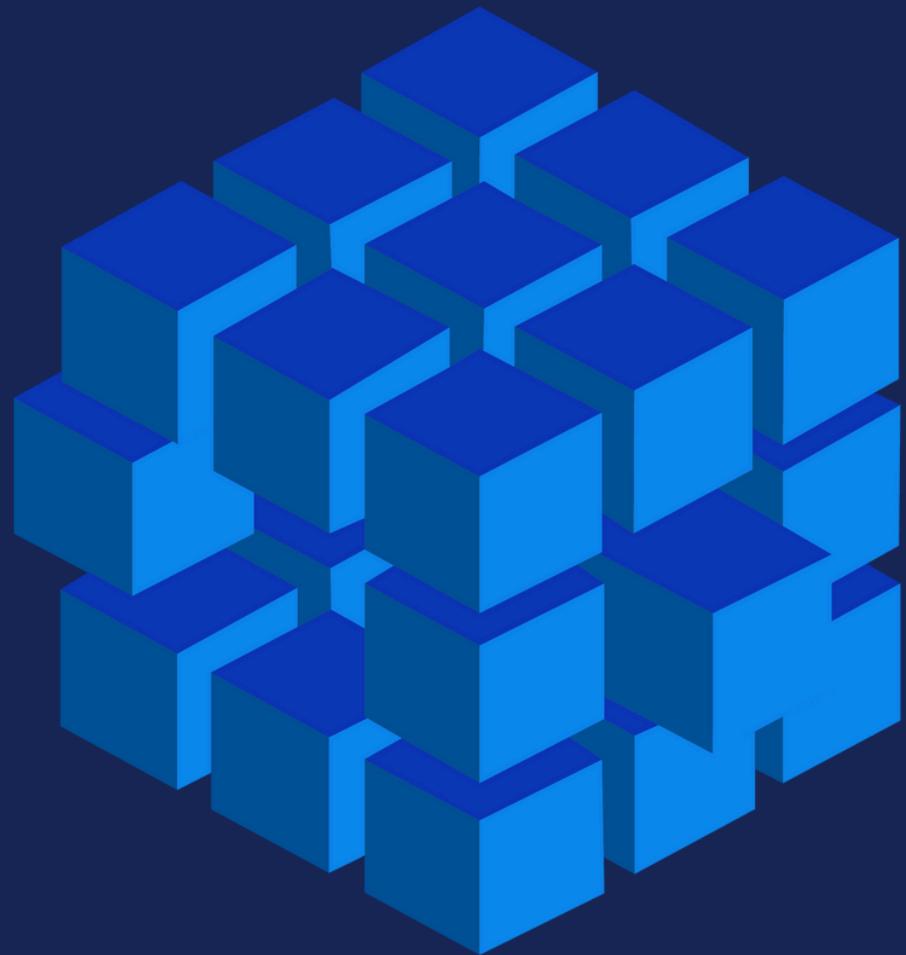


What if we want to add
more message consumers?



Eventual Consistency

Dealing with eventual consistency.



Next:
Publishing
Domain Events



Publishing Domain Events



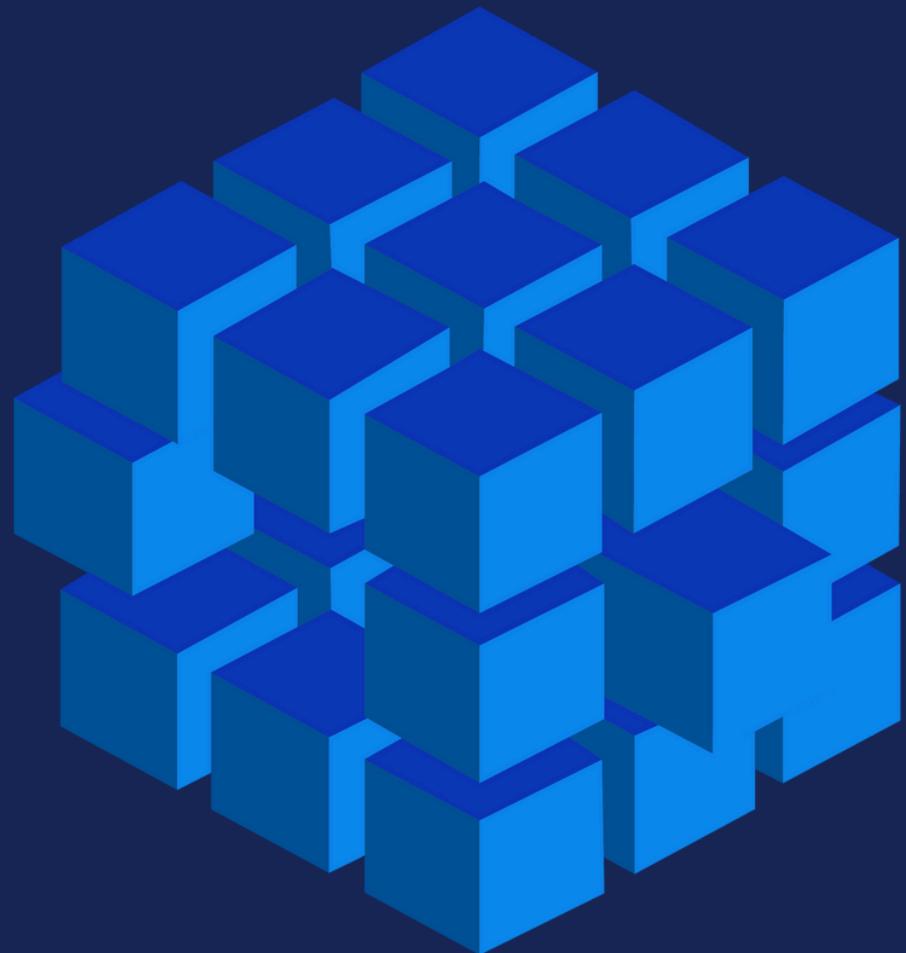
Next:
Introducing the
Users Module



Introducing the Users Module

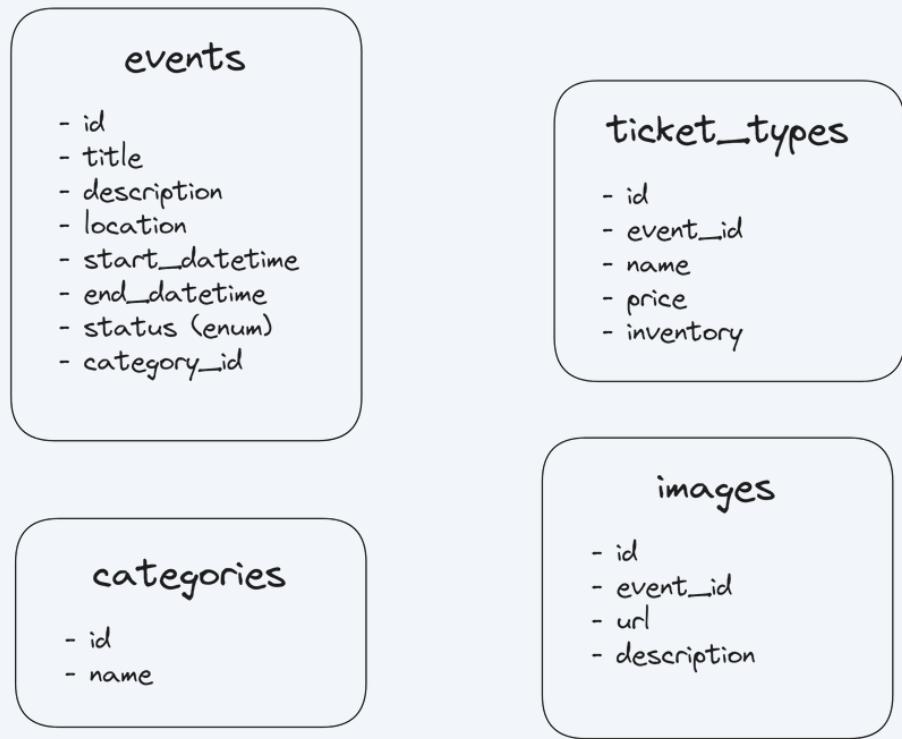


Next:
Scaffolding the
Ticketing Module

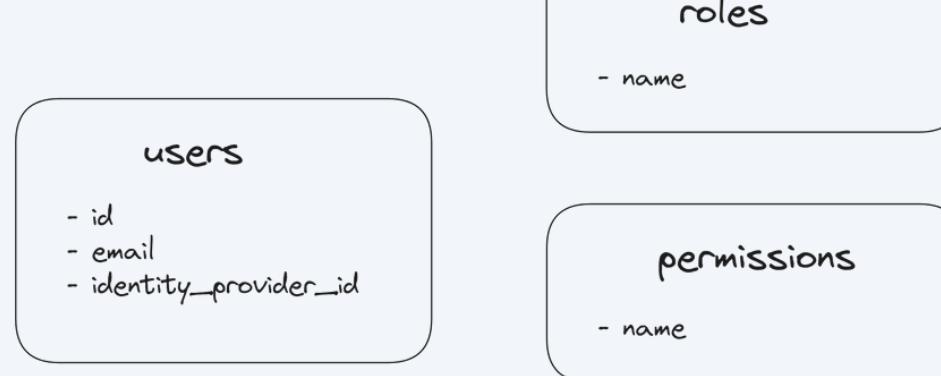


Scaffolding the Ticketing Module

Events



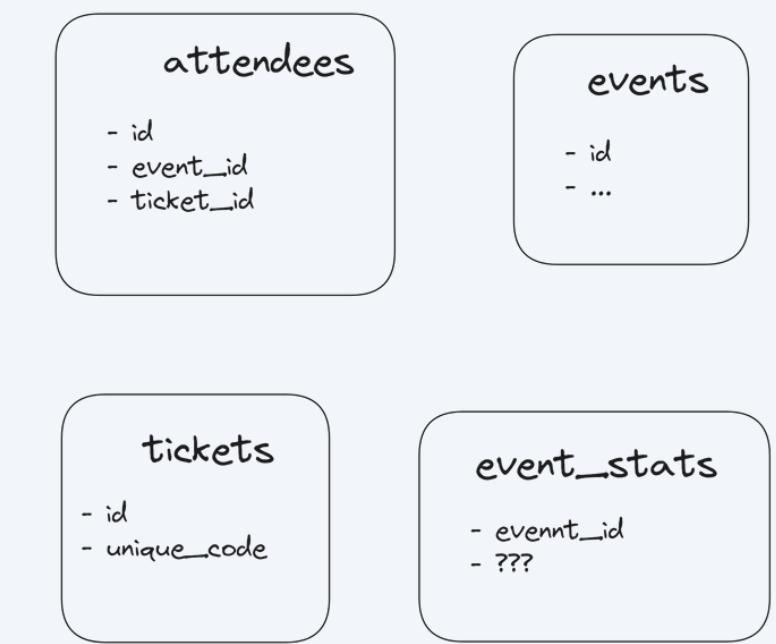
Users



Ticketing



Attendance





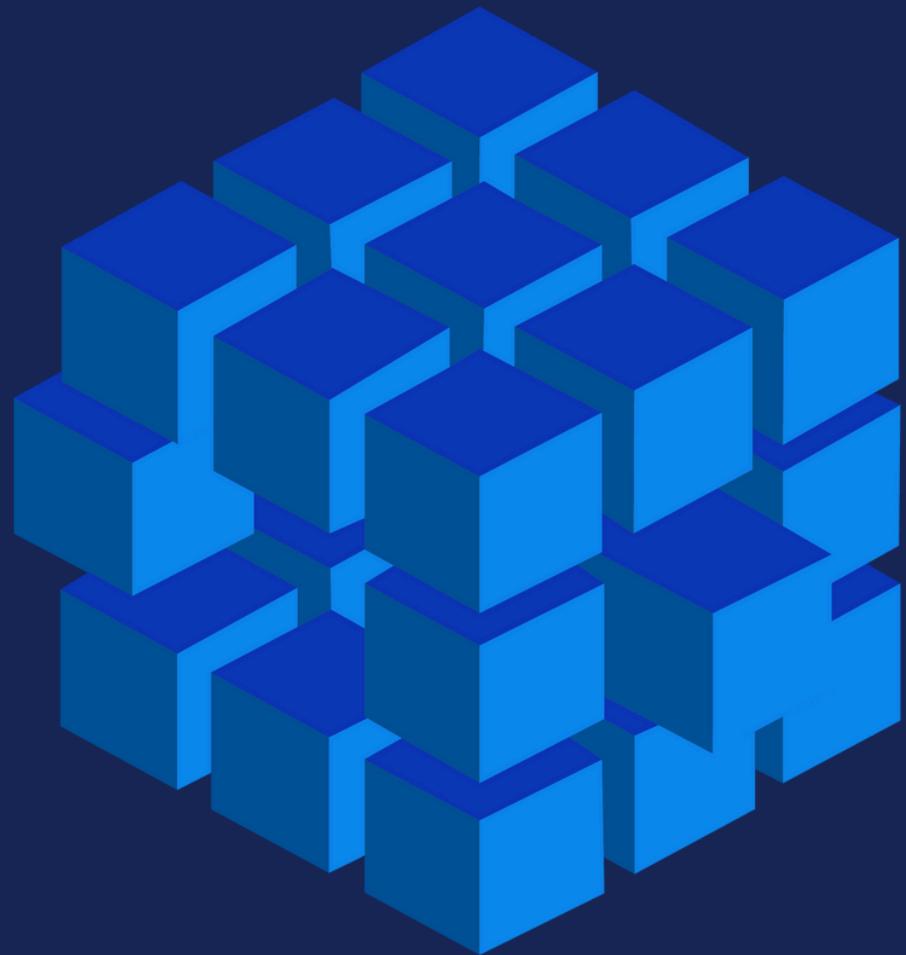
Next:
Synchronous
Communication



Synchronous Communication

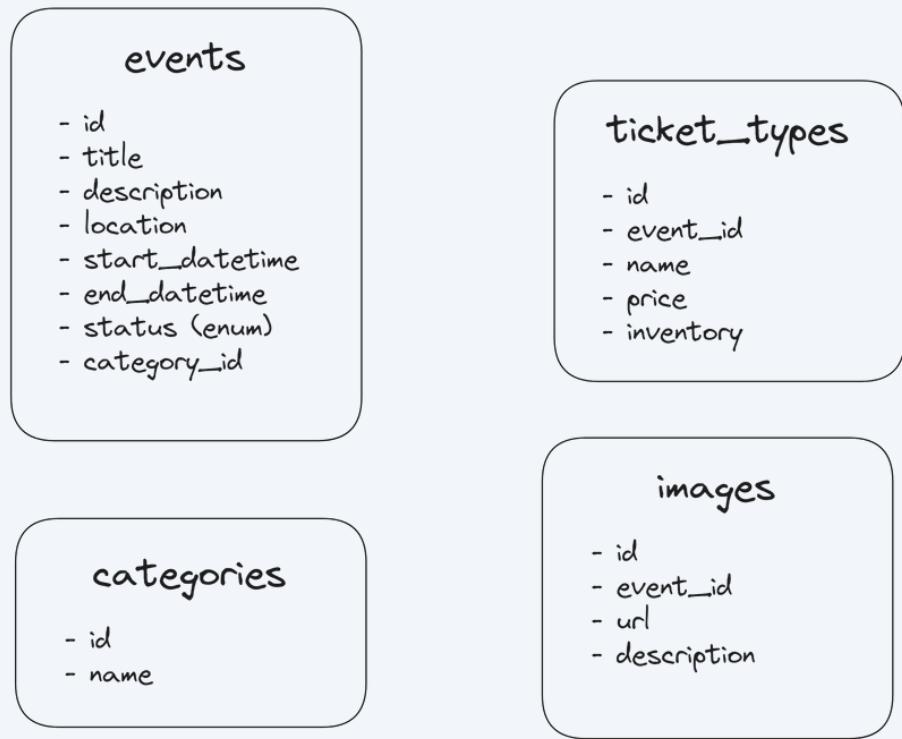


Next:
Duplicating Data
Between Modules

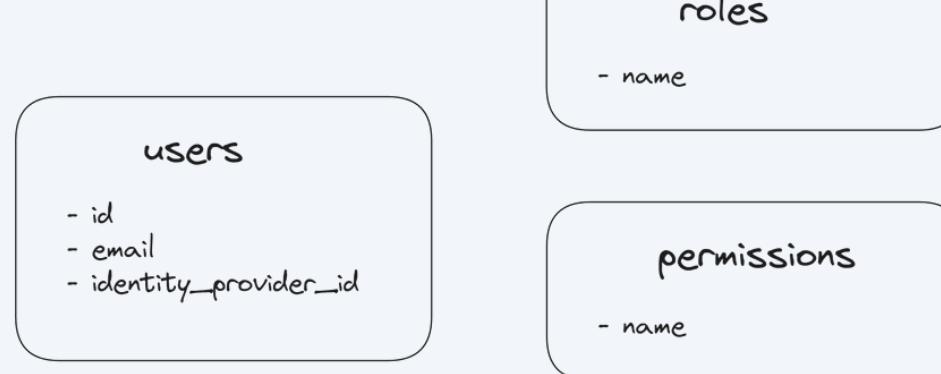


Duplicating Data Between Modules

Events



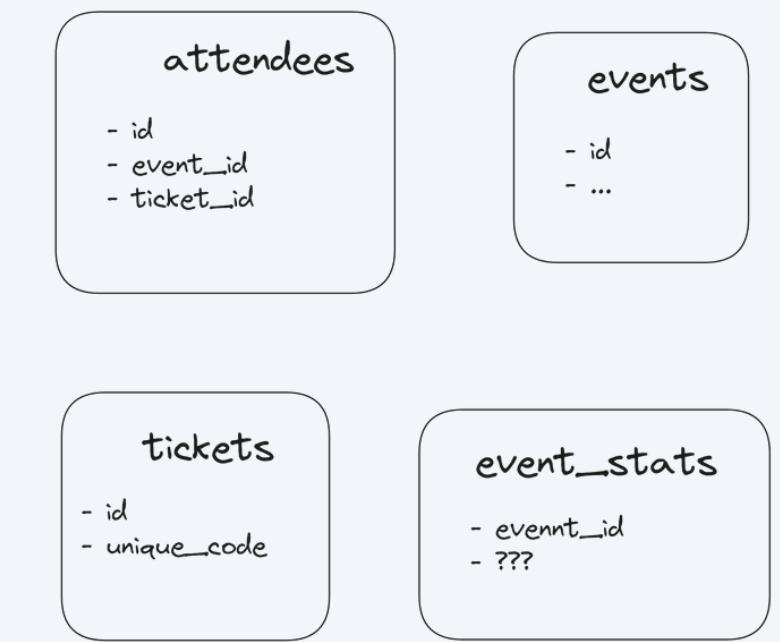
Users



Ticketing



Attendance



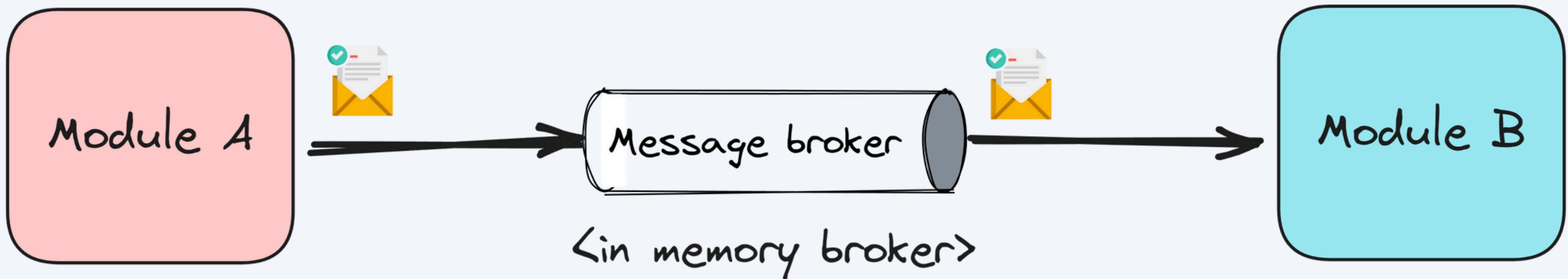


Next:
Asynchronous
Communication



Asynchronous Communication

Messaging





Next:
Eventual Consistency:
Introduction



Eventual Consistency: Introduction

Intro

Dealing with eventual consistency.

Eventual consistency between modules.

Maintaining consistency within a module.



Next:
Authentication &
Authorization

Authentication & Authorization



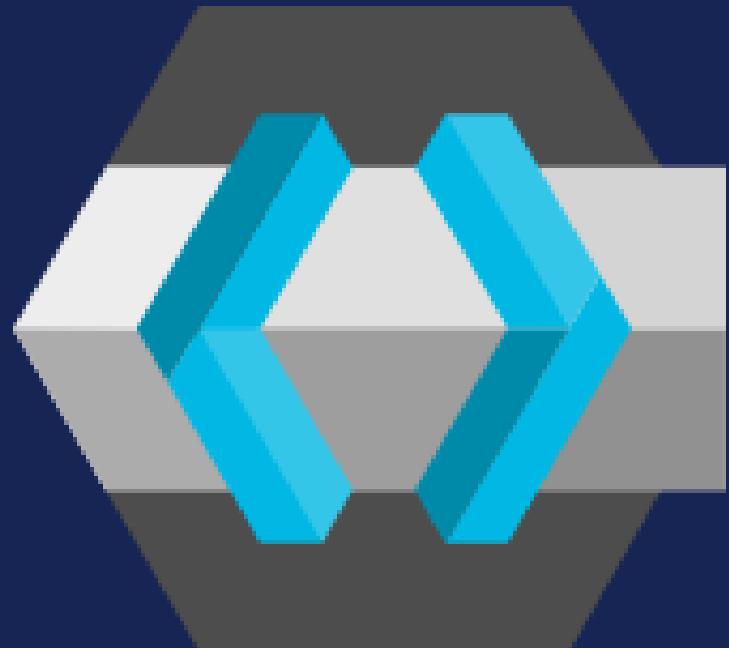
Intro

KeyCloak external identity provider.

Authentication in ASP.NET Core with KeyCloak.

RBAC (Role-based Access Control) authorization.

Why use an external Identity Provider?



External IDP Benefits

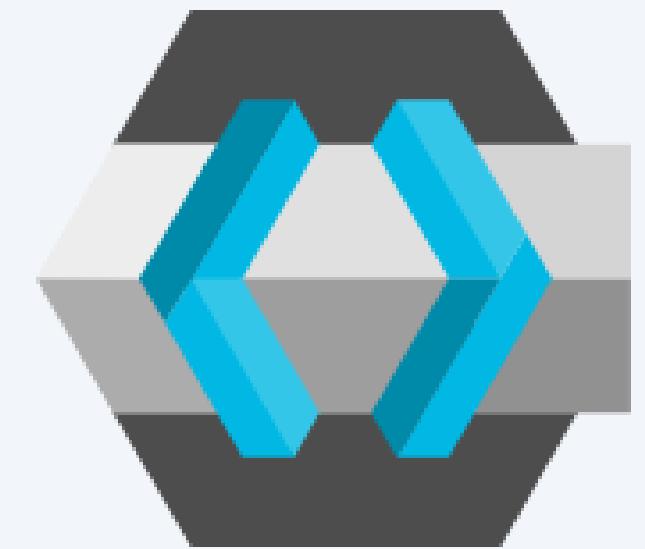
Established IDPs have robust security protocols in place.

Social Sign-in, Single Sign-On (SSO).

Reduced development overhead.

OAuth 2.0

Industry-standard protocol for authorization.

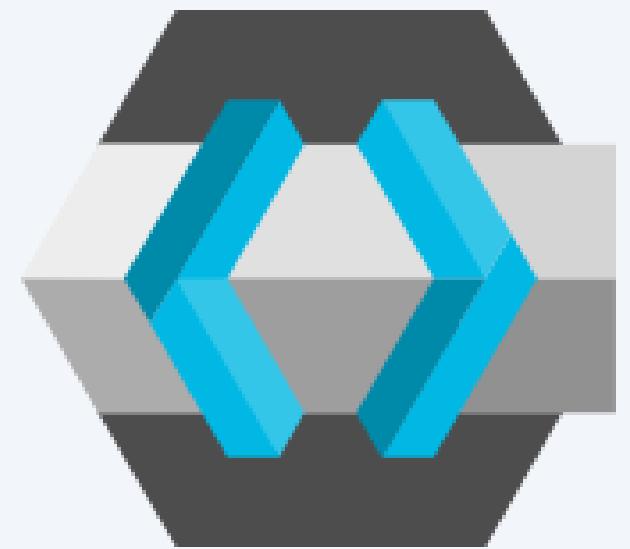


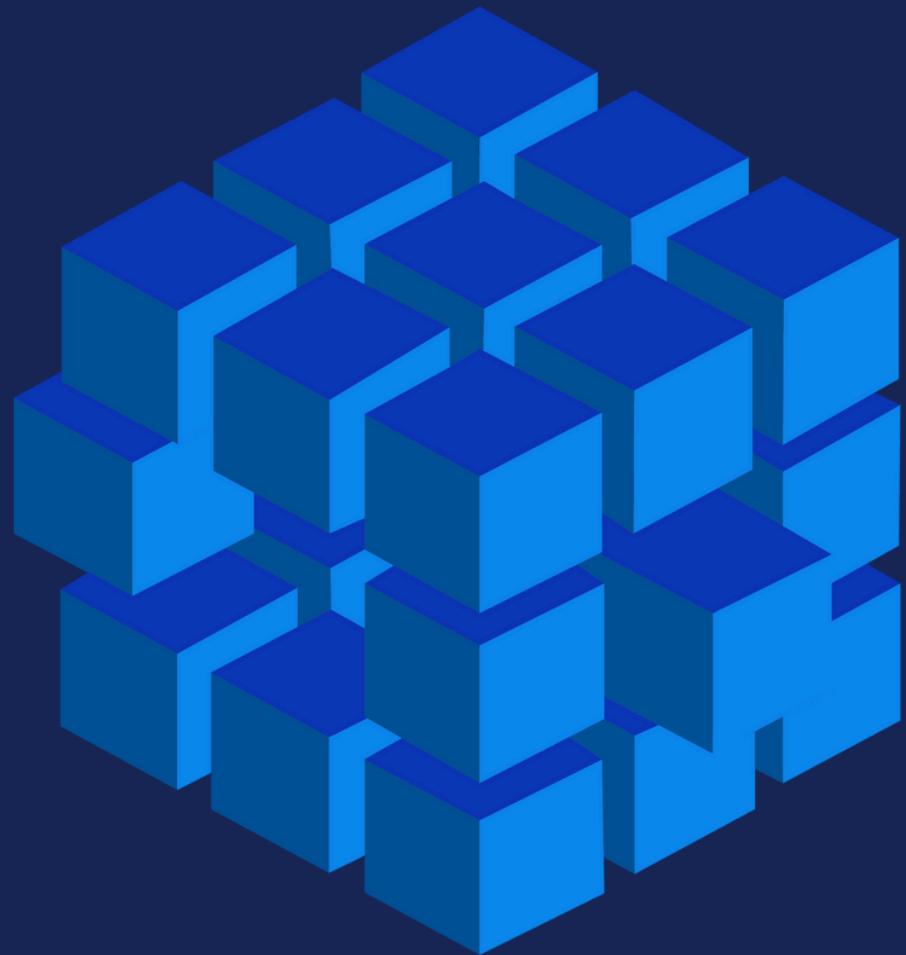
OAuth 2.0

Industry-standard protocol for authorization.

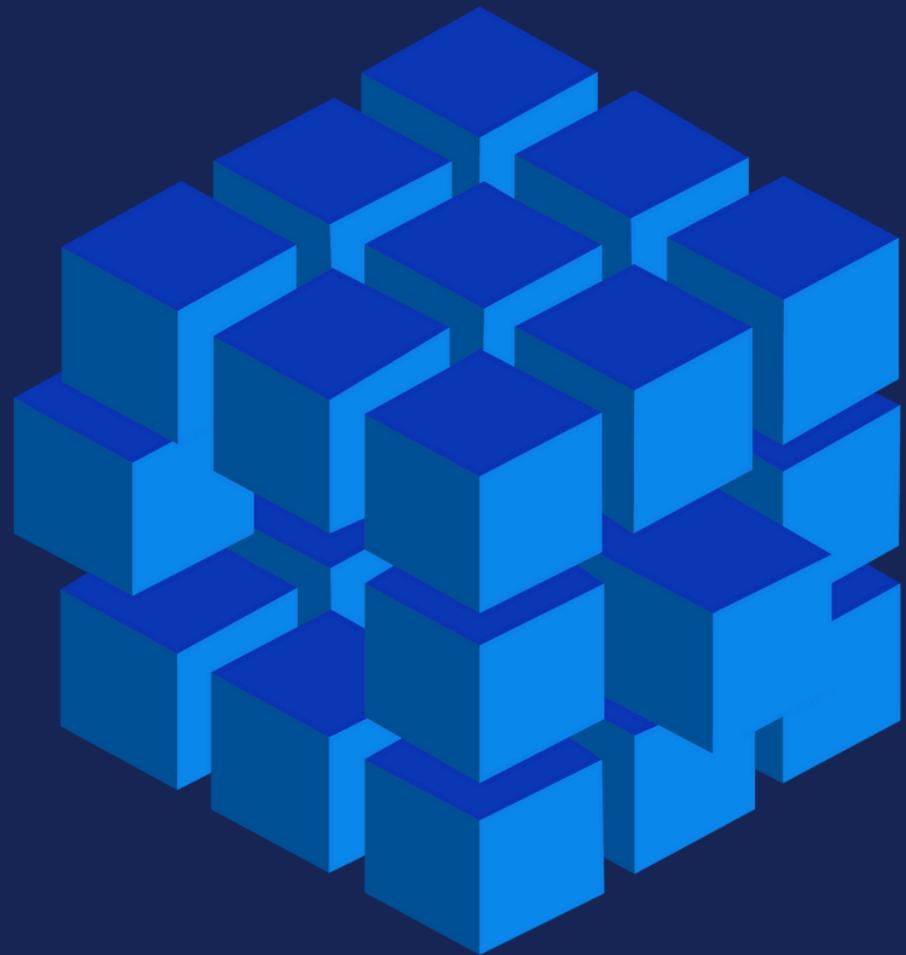
Authorization Code Grant.

Client Credentials Grant.





Next:
Keycloak Identity
Provider Setup



Keycloak Identity Provider Setup

Intro

KeyCloak setup with Docker.

Configuring public and confidential clients.

Exporting (and importing) the KeyCloak realm.

Troubleshooting guide.



Next:
Token-Based
Authentication

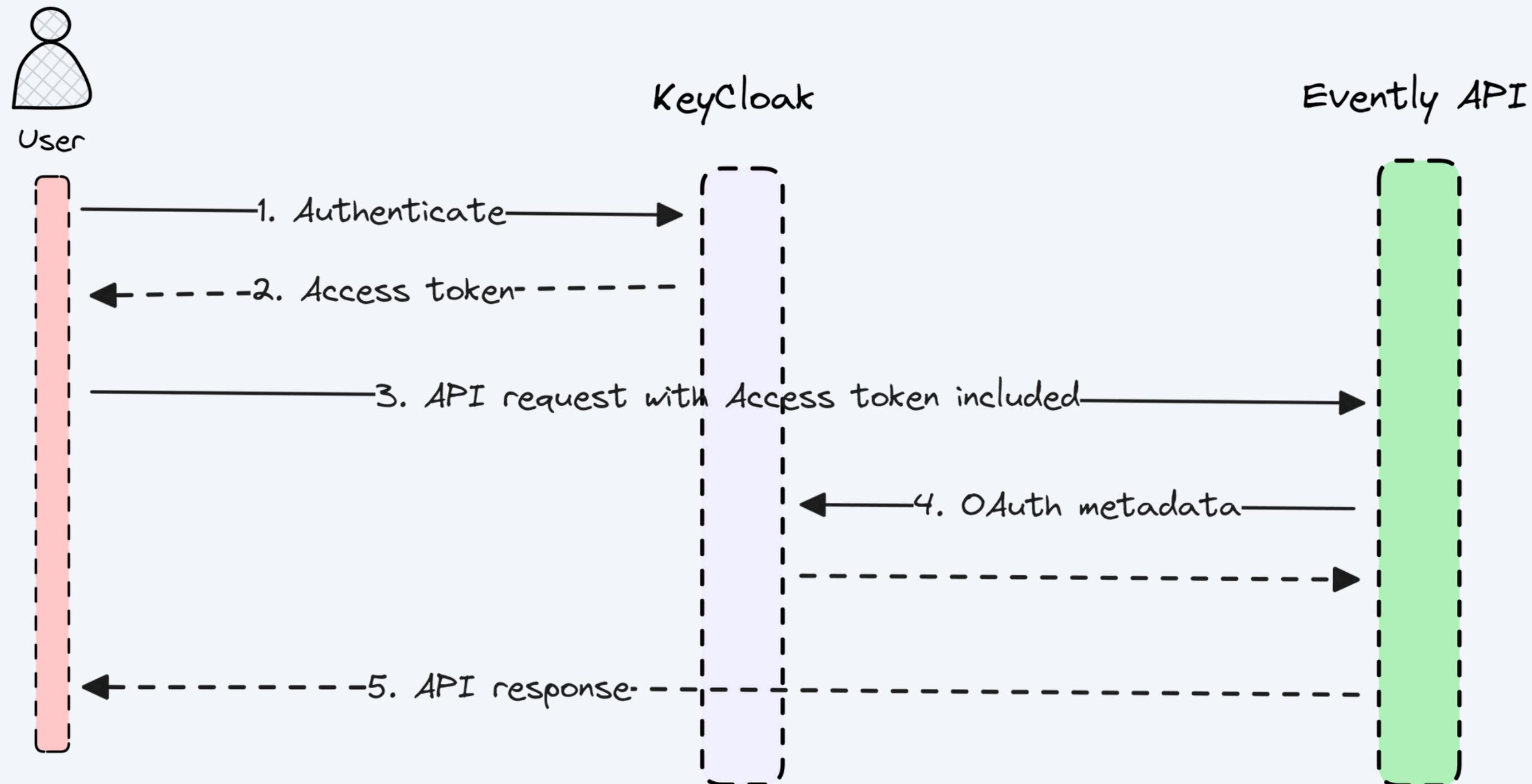


Token-Based Authentication

JSON Web Tokens

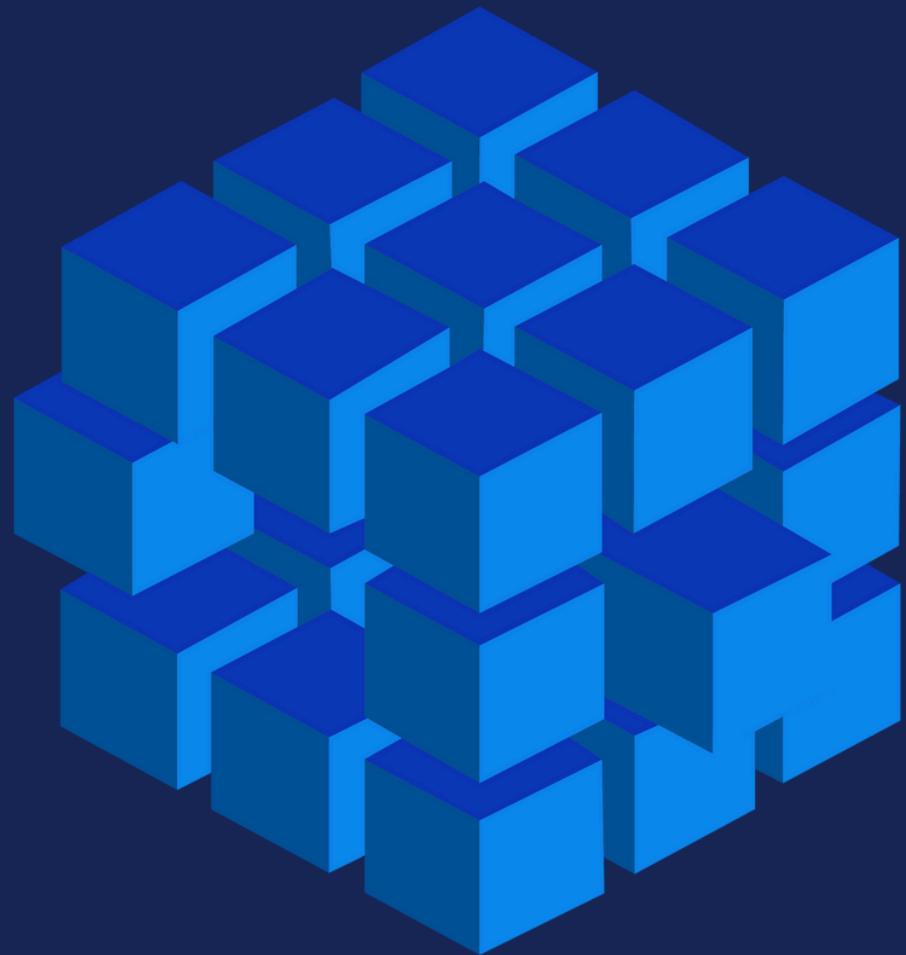
Stateless authentication approach.

Easy to integrate in ASP.NET Core.

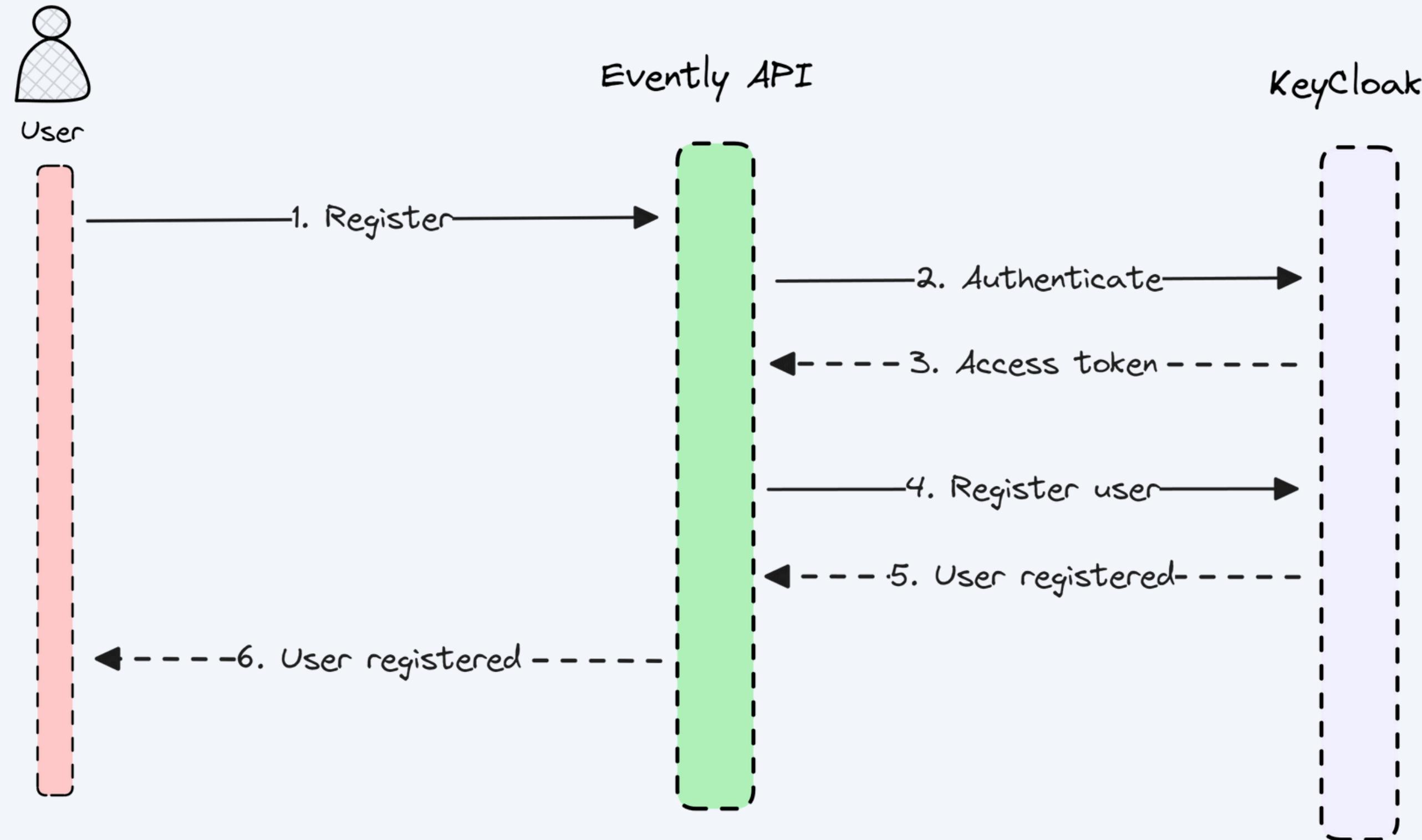




Next:
User Registration
With Keycloak



User Registration With Keycloak



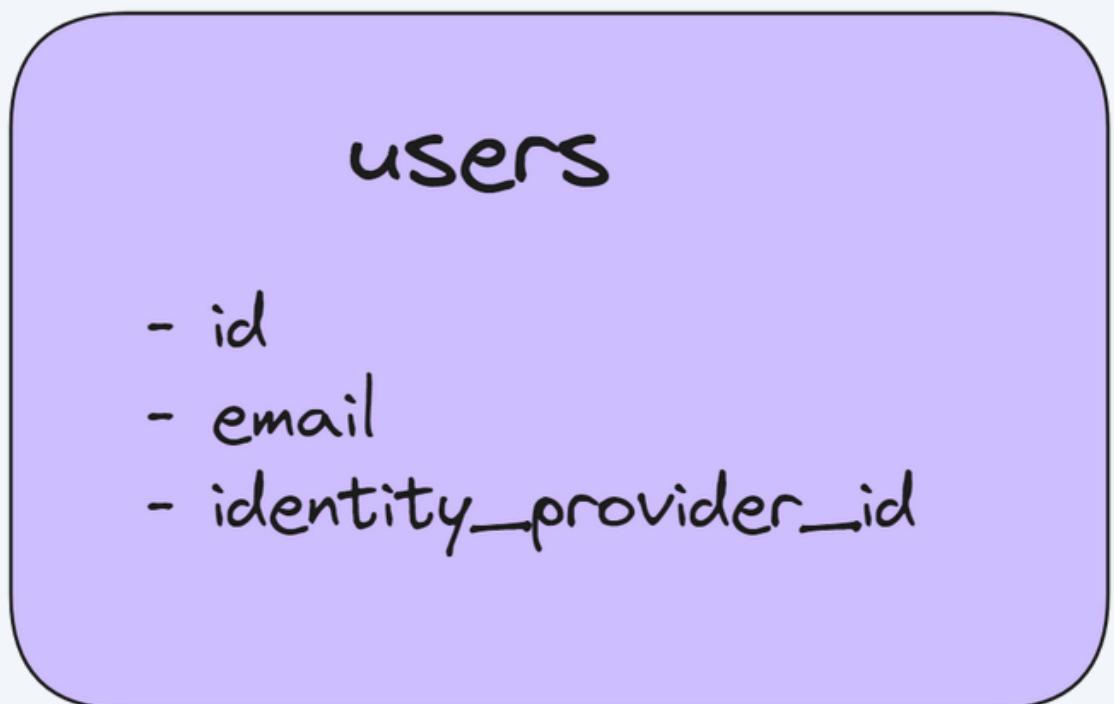


Next:
RBAC Authorization



RBAC Authorization

Users

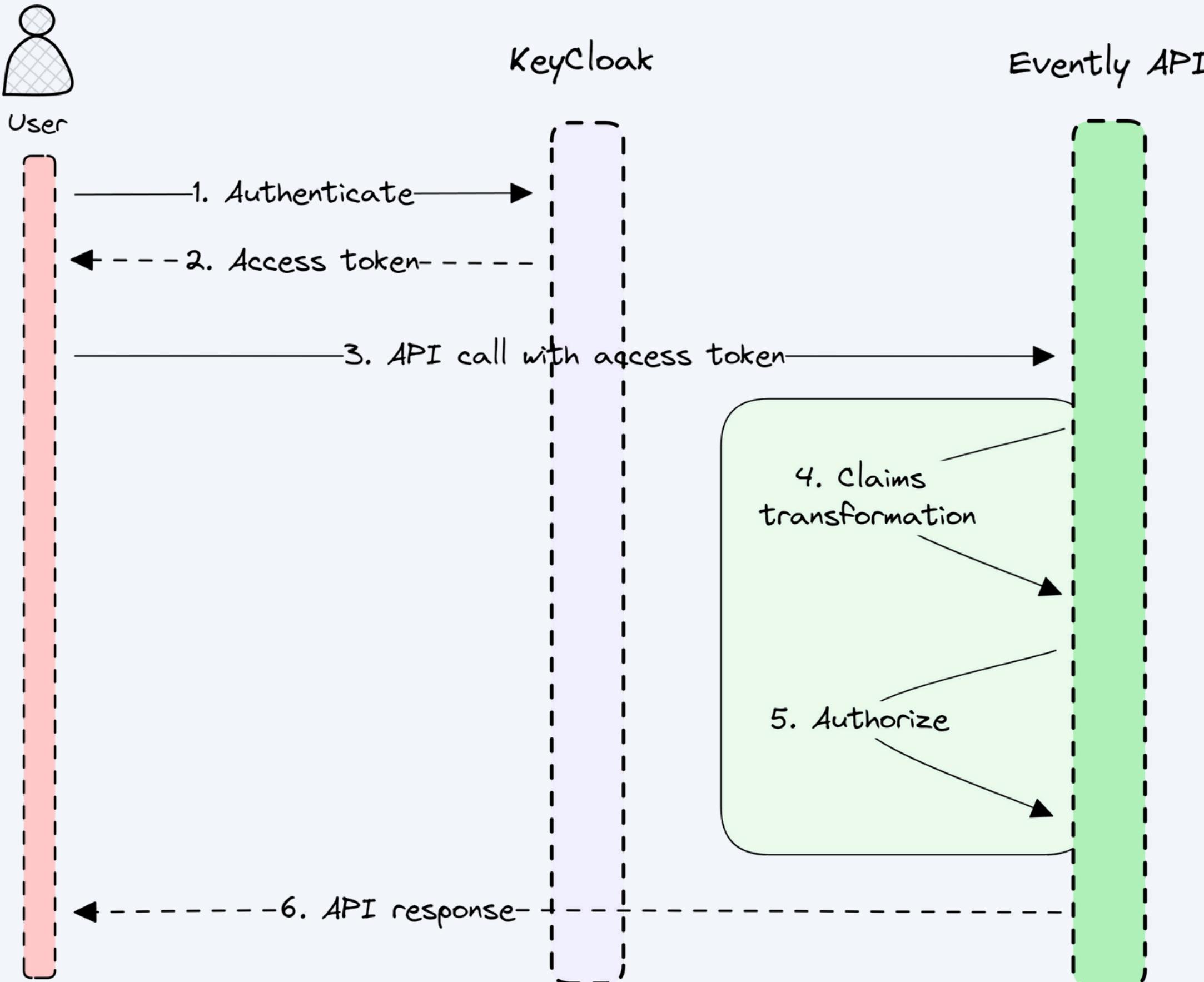


RBAC in ASP.NET Core

Use Keycloak for authentication.

Fetch authorization data from the Users module database.

Fine-grained control.





Next:
Architecture
Enforcement

Architecture Enforcement

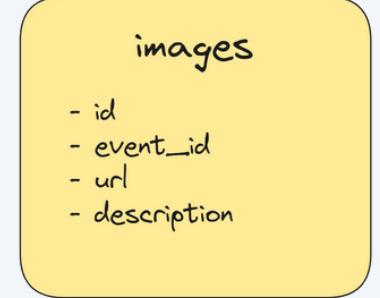


Intro

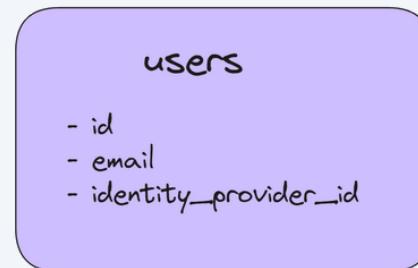
Three ways of enforcing architecture.

Cost vs. benefit of enforcing architecture.

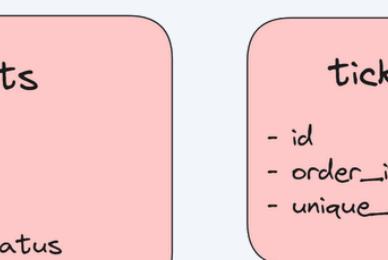
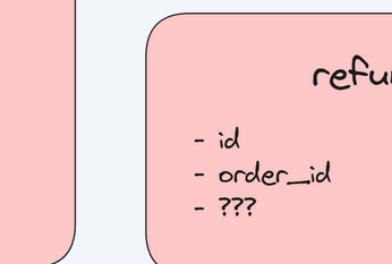
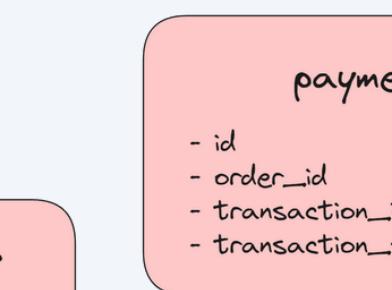
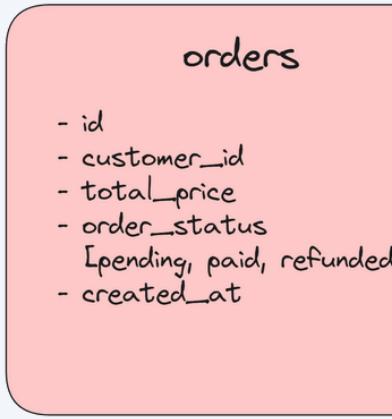
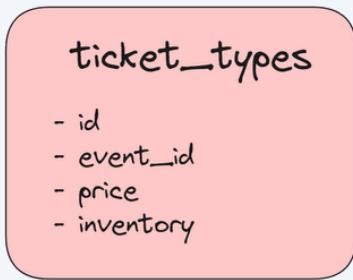
Events



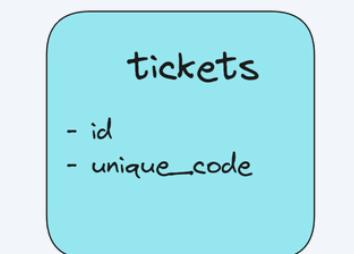
Users

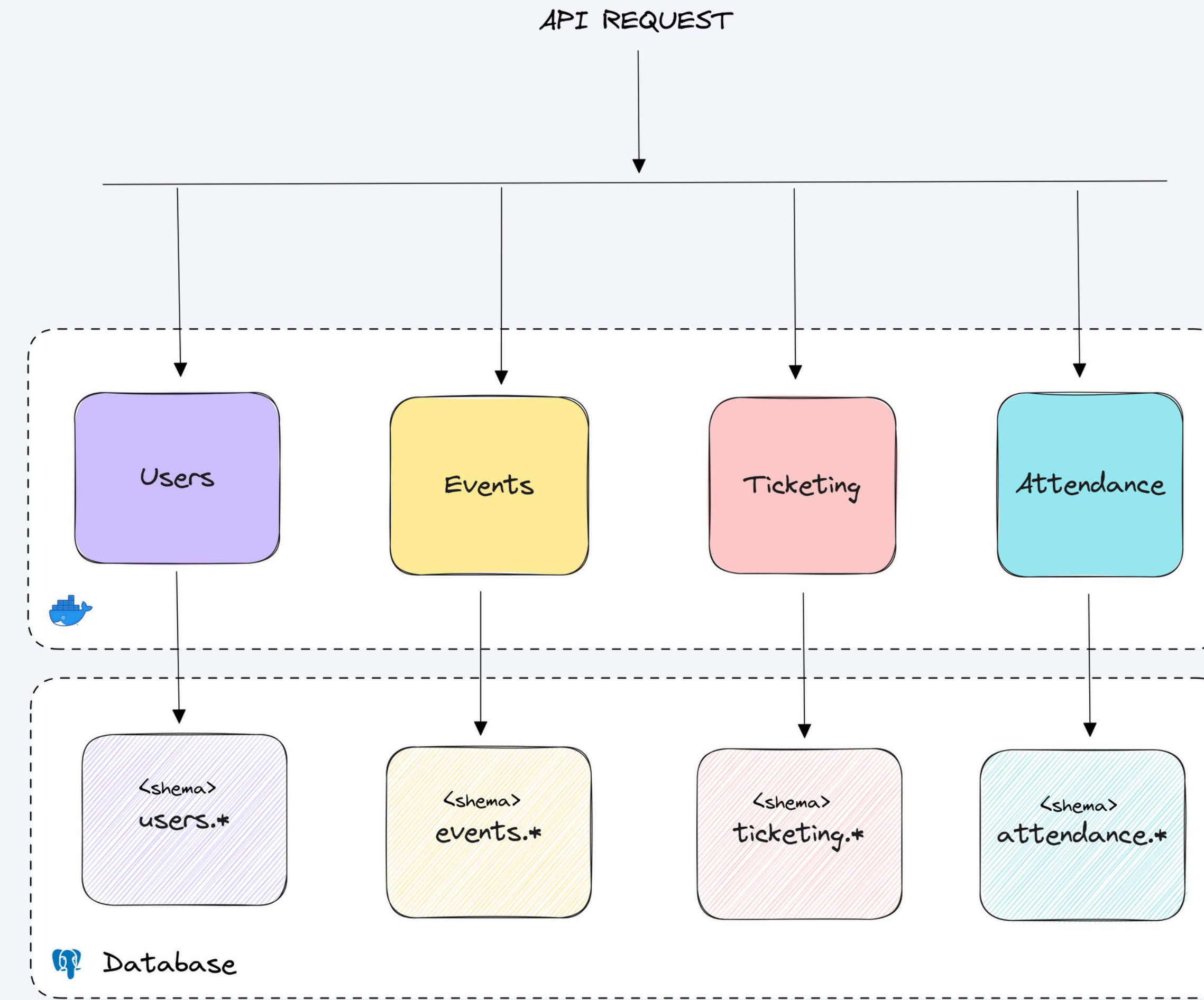


Ticketing

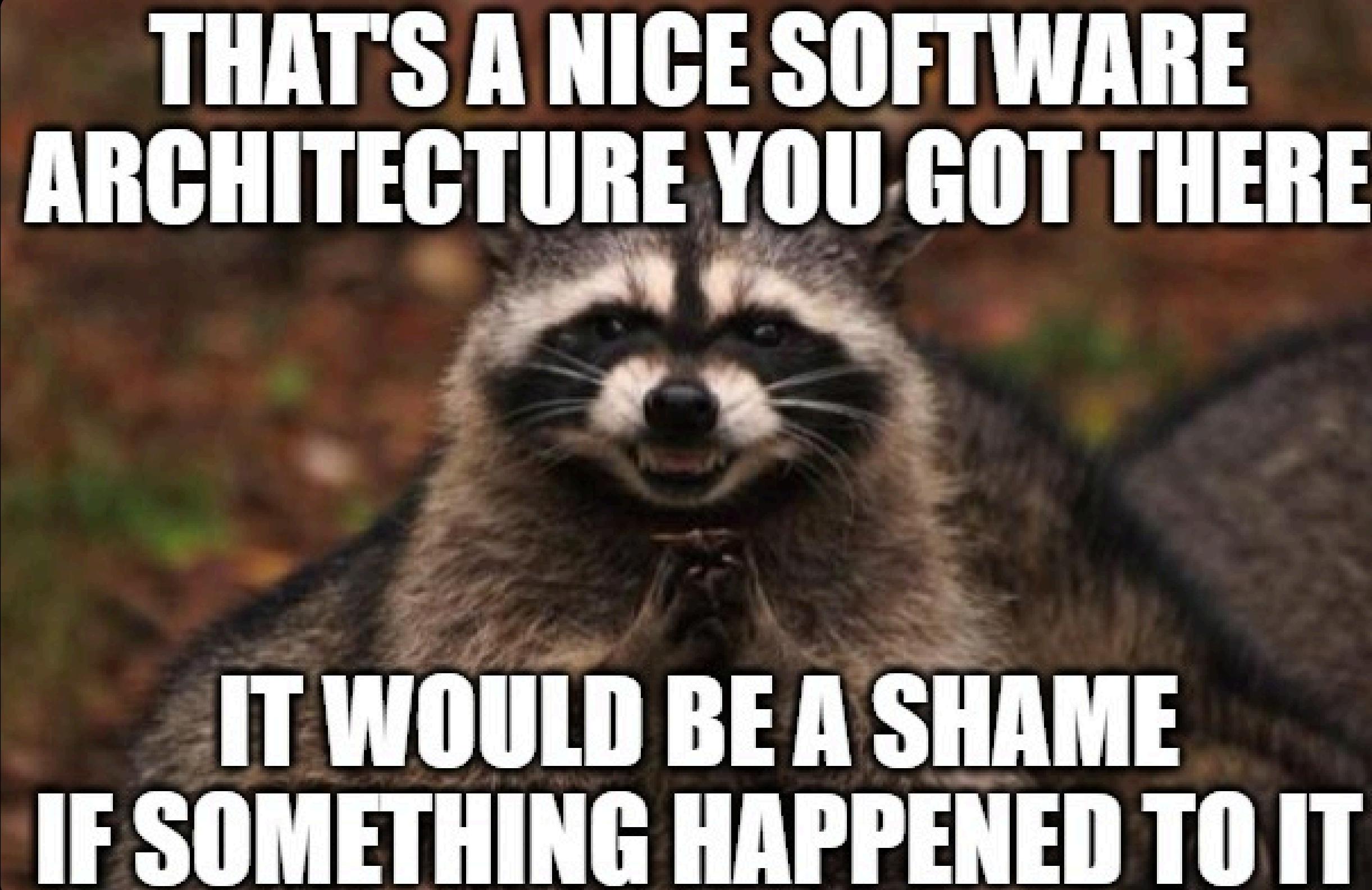


Attendance





**THAT'S A NICE SOFTWARE
ARCHITECTURE YOU GOT THERE**



**IT WOULD BE A SHAME
IF SOMETHING HAPPENED TO IT**

Architecture Enforcement

- Compiler, Static code analysis
- Architecture testing
- Code reviews...

Code Reviews

- Most expensive way to enforce architecture
- Error-prone



Next:
Modular Architecture



Modular Architecture

Compiler

Limited capability to enforce architecture rules.

Assemblies, internal keyword.

Encapsulation.

Static Code Analysis

Enforce code style rules for your code.

- StyleCop
- SonarAnalyzer
- Custom Roslyn rules

Static Code Analysis

There are some more robust tools:

- SonarQube
- NDepend
- NsDepCop

NsDepCop

“A static code analysis tool that helps you to enforce namespace dependency rules in C# projects.”

NuGet: NsDepCop

```
<NsDepCopConfig IsEnabled="true" ChildCanDependOnParentImplicitly="true">

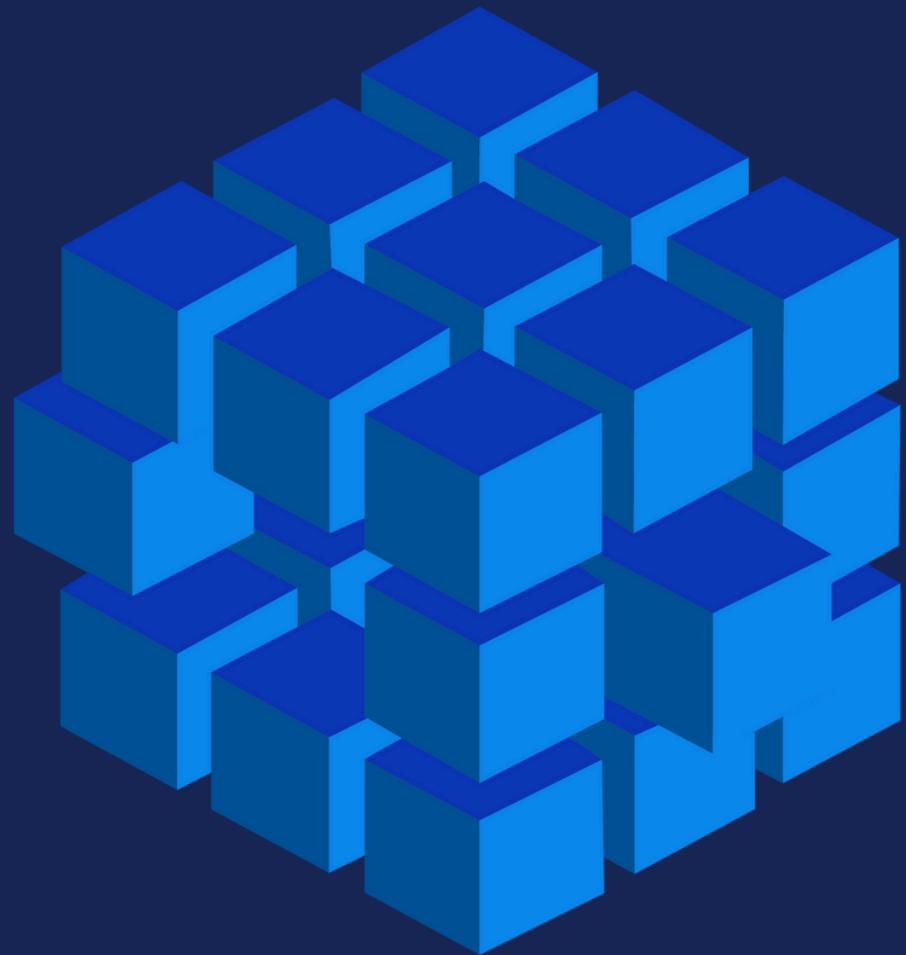
    <Allowed From="*" To="System.*" />

    <Allowed From="Evently.Application.*" To="Evently.Domain.*" />
    <Allowed From="Evently.Presentation.*" To="Evently.Application.*" />
    <Allowed From="Evently.Infrastructure.*" To="Evently.Application.*" />
    <Allowed From="Evently.Infrastructure.*" To="Evently.Presentation.*" />

</NsDepCopConfig>
```



Next:
Architecture Testing



Architecture Testing

Architecture Testing

Automated tests that verify the structure and design of your code.

- NetArchTest
- ArchUnitNET

```
string[] otherModules = [EventsNamespace, TicketingNamespace, AttendanceNamespace];
string[] integrationEventsModules = [
    EventsIntegrationEventsNamespace,
    TicketingIntegrationEventsNamespace,
    AttendanceIntegrationEventsNamespace];

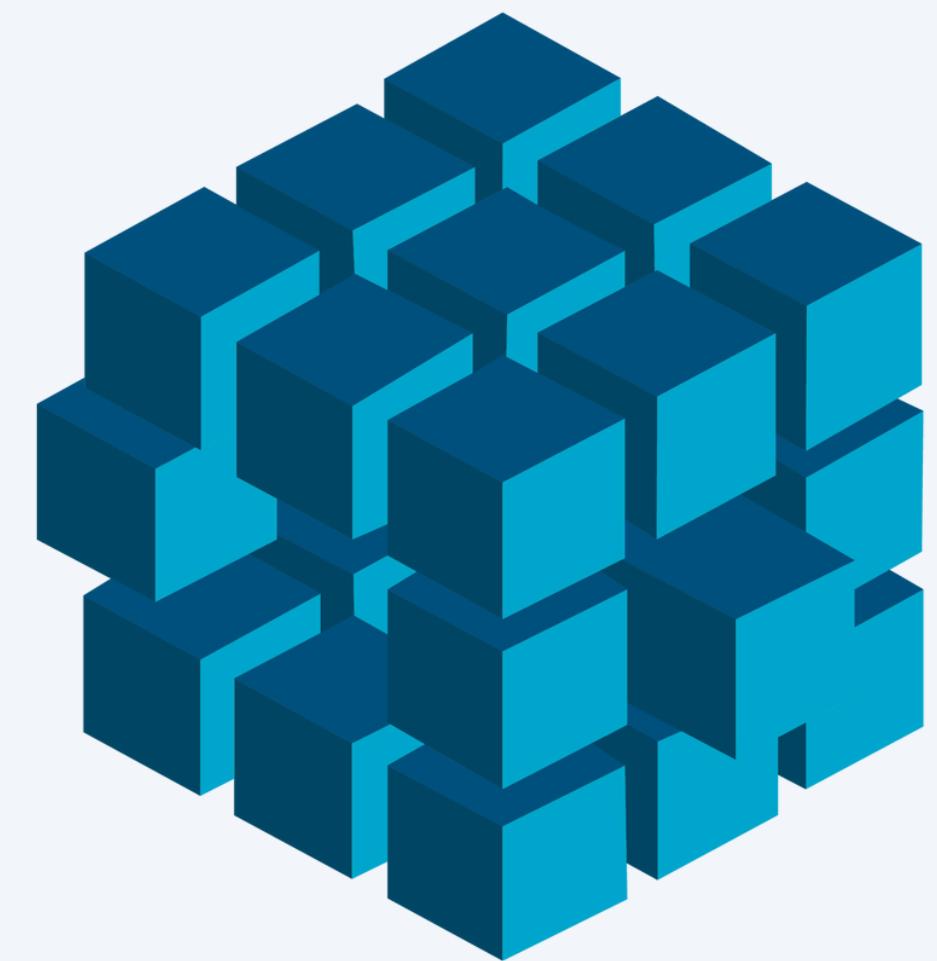
List<Assembly> usersAssemblies =
[
    typeof(User).Assembly,
    Modules.Users.Application.AssemblyReference.Assembly,
    Modules.Users.Presentation.AssemblyReference.Assembly,
    typeof(UsersModule).Assembly
];

Types.InAssemblies(usersAssemblies)
    .That()
    .DoNotImplementInterface(typeof(IIntegrationEventHandler<>))
    .And()
    .DoNotHaveDependencyOnAny(integrationEventsModules)
    .Should()
    .NotHaveDependencyOnAny(otherModules)
    .GetResult()
    .ShouldBeSuccessful();
```



Next:
Reliable Messaging:
Outbox and Inbox

Reliable Messaging: Outbox and Inbox



Intro

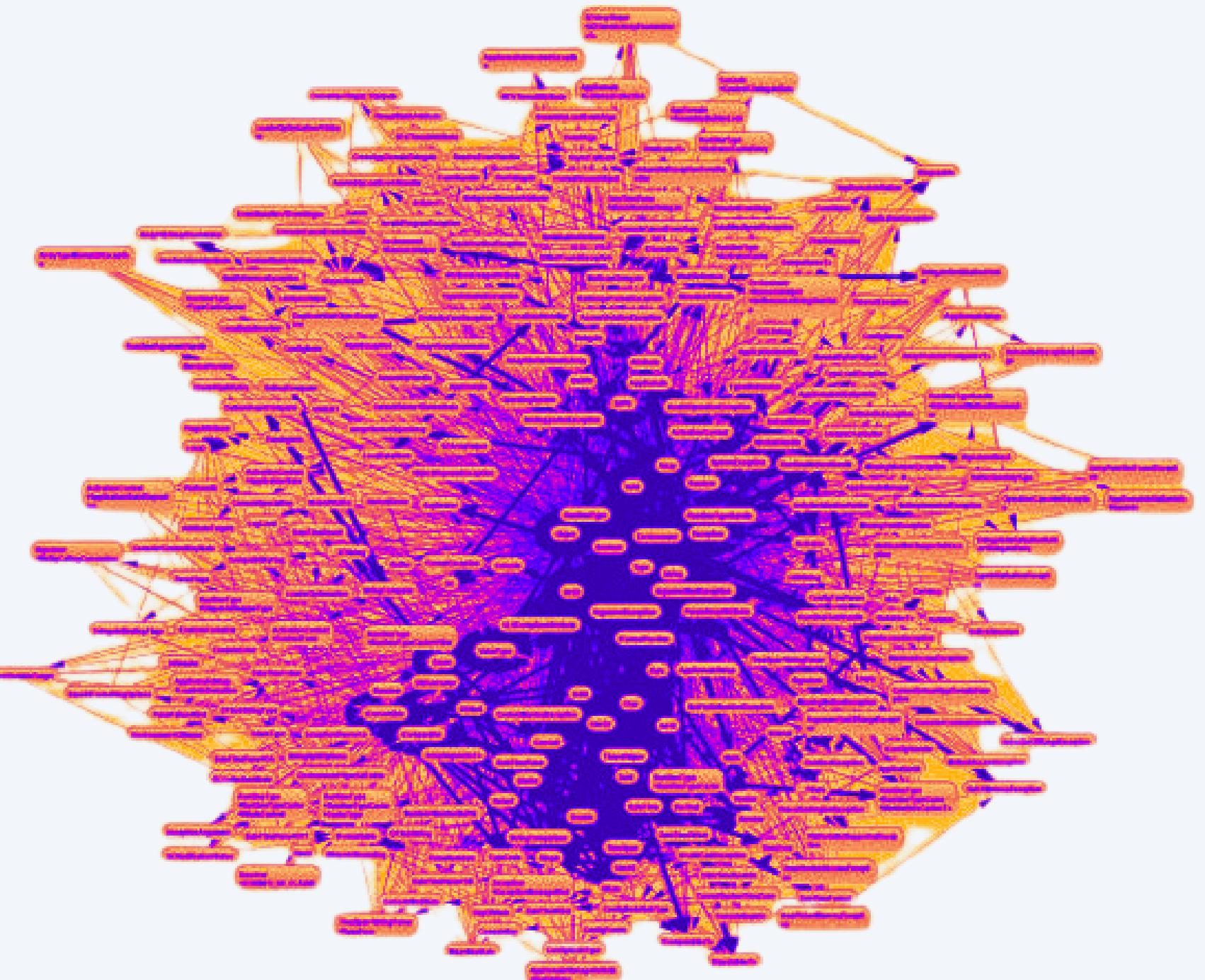
Asynchronous messaging guarantees.

Outbox and Inbox pattern.

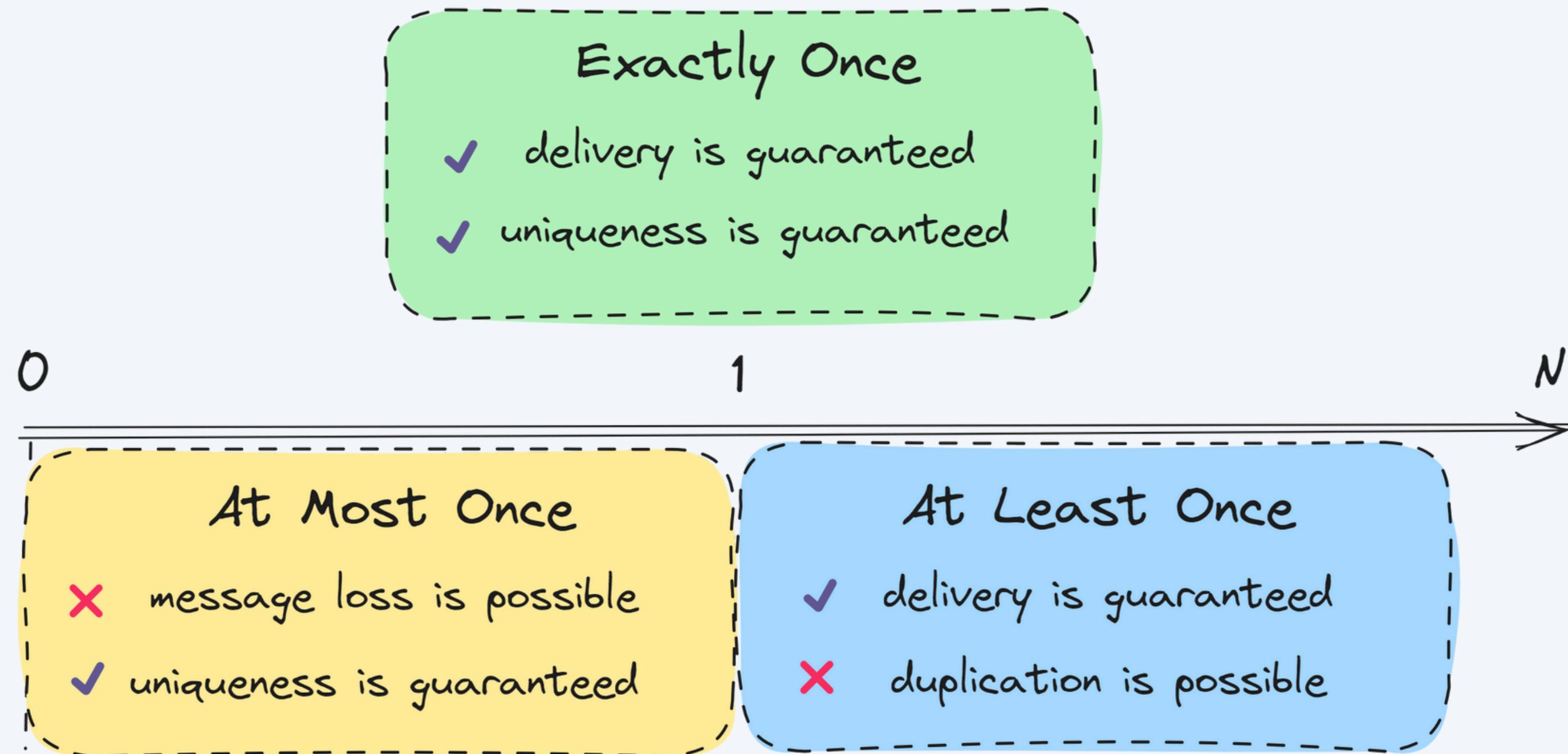
Idempotency.

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



Messaging Guarantees





Next:
Outbox Pattern



Outbox Pattern

The Problem (Simplified)

```
// Start the transaction -----  
  
var user = User.Create(request.Email, request.FirstName, request.LastName, result.Value);  
  
userRepository.Insert(user);  
  
await unitOfWork.SaveChangesAsync(cancellationToken);  
  
// Complete the transaction -----  
  
await eventBus.PublishAsync(  
    new UserRegisteredIntegrationEvent(user.Id, user.Email, user.FirstName, user.LastName),  
    cancellationToken);
```

The Solution?

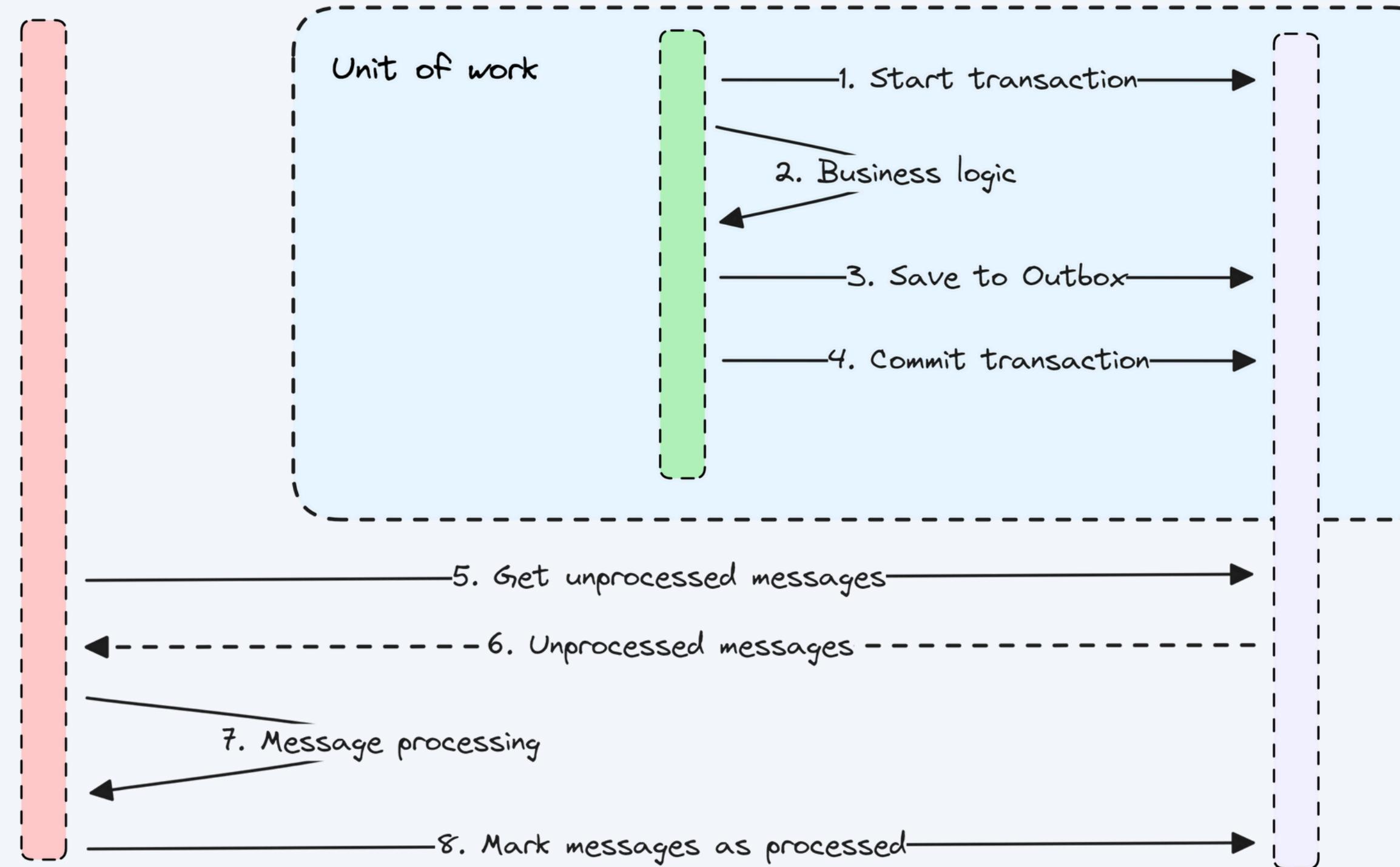
Decouple the primary transaction from the secondary side effects.

Ensure the primary transaction is atomic.

Outbox Processor

Command Handler

Database





Next:
Outbox: Idempotent
Consumers



Outbox: Idempotent Consumers

Outbox Pattern

At Least Once delivery.

What about duplicates?

Idempotency:

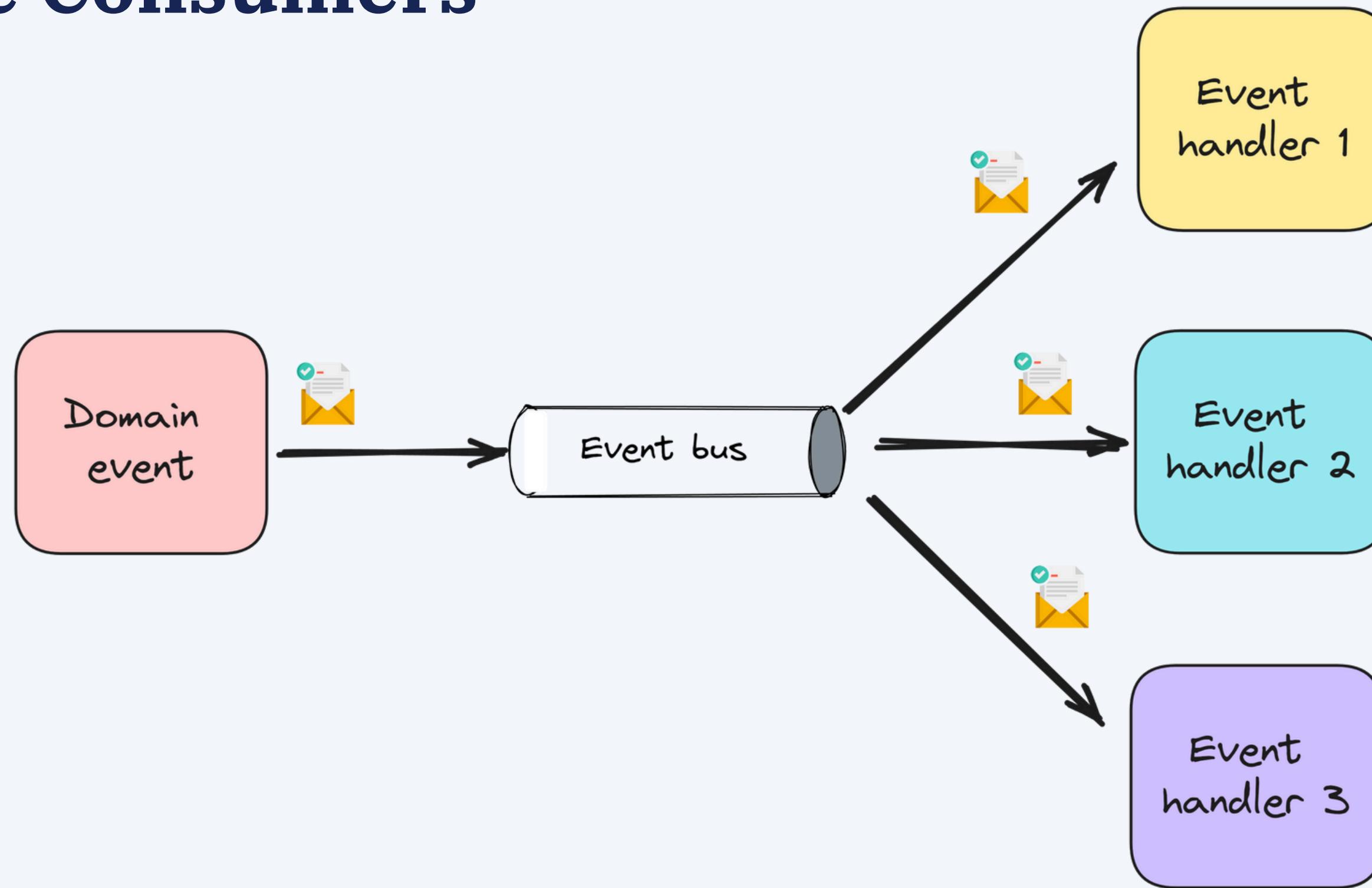
The ability of a system to produce the same outcome,
even if the same event or message is received more
than once.

Idempotent Consumers

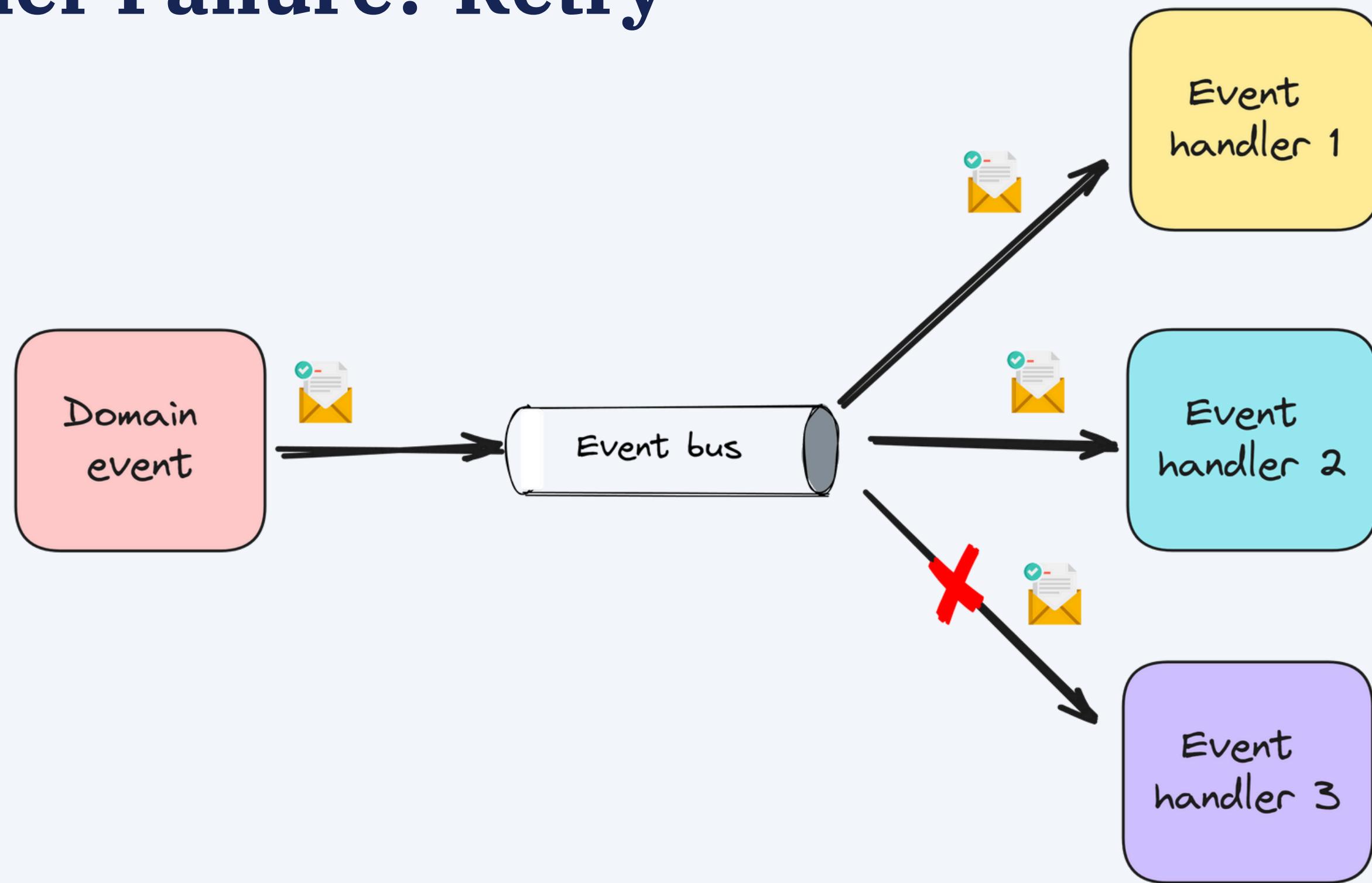
Two ways we can achieve idempotency:

- Implement the consumers so they are “idempotent” by design.
- Implement explicit de-duplication logic to ensure idempotency.

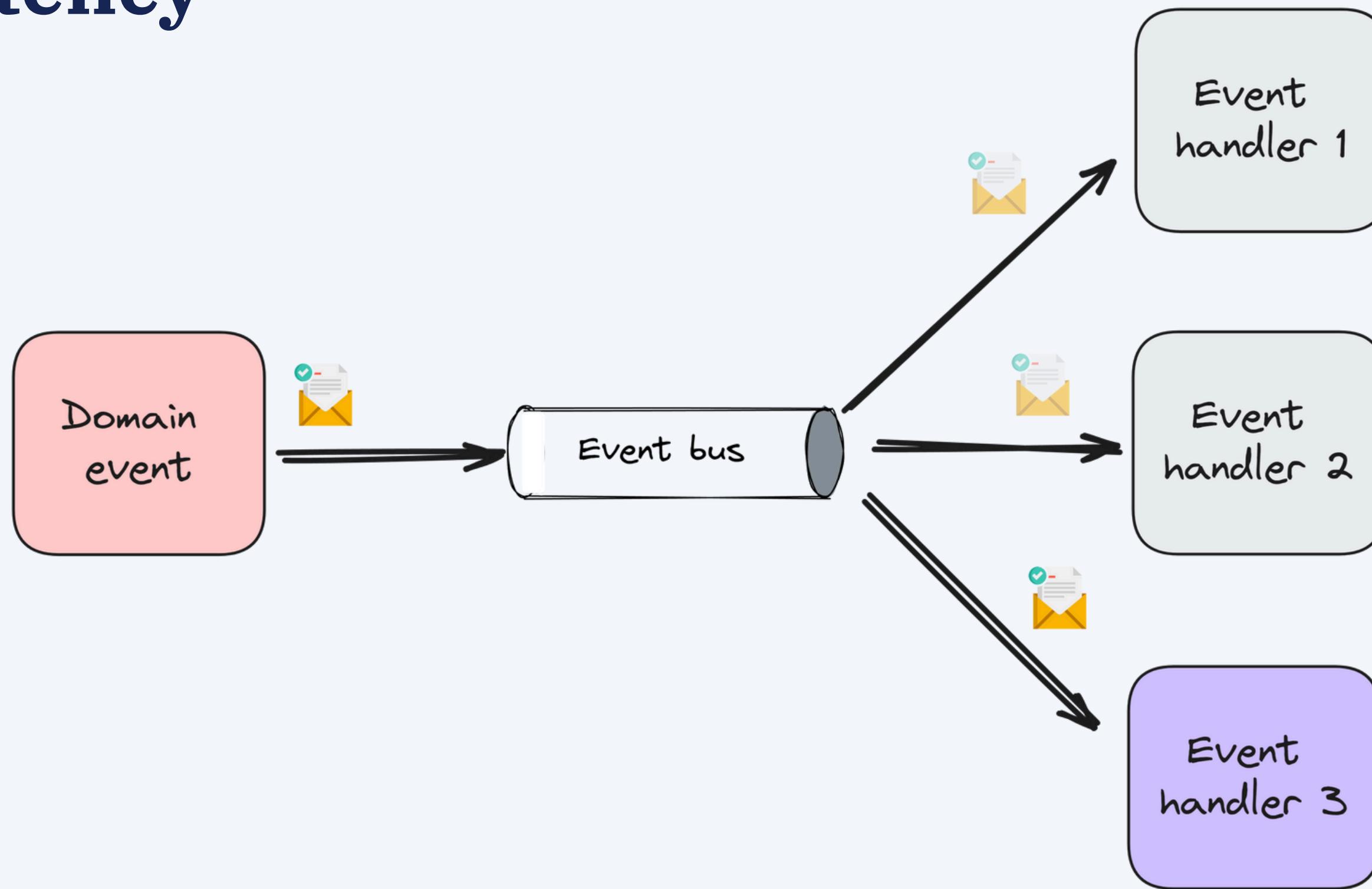
Multiple Consumers



Consumer Failure? Retry



Idempotency





Next:
Inbox Pattern



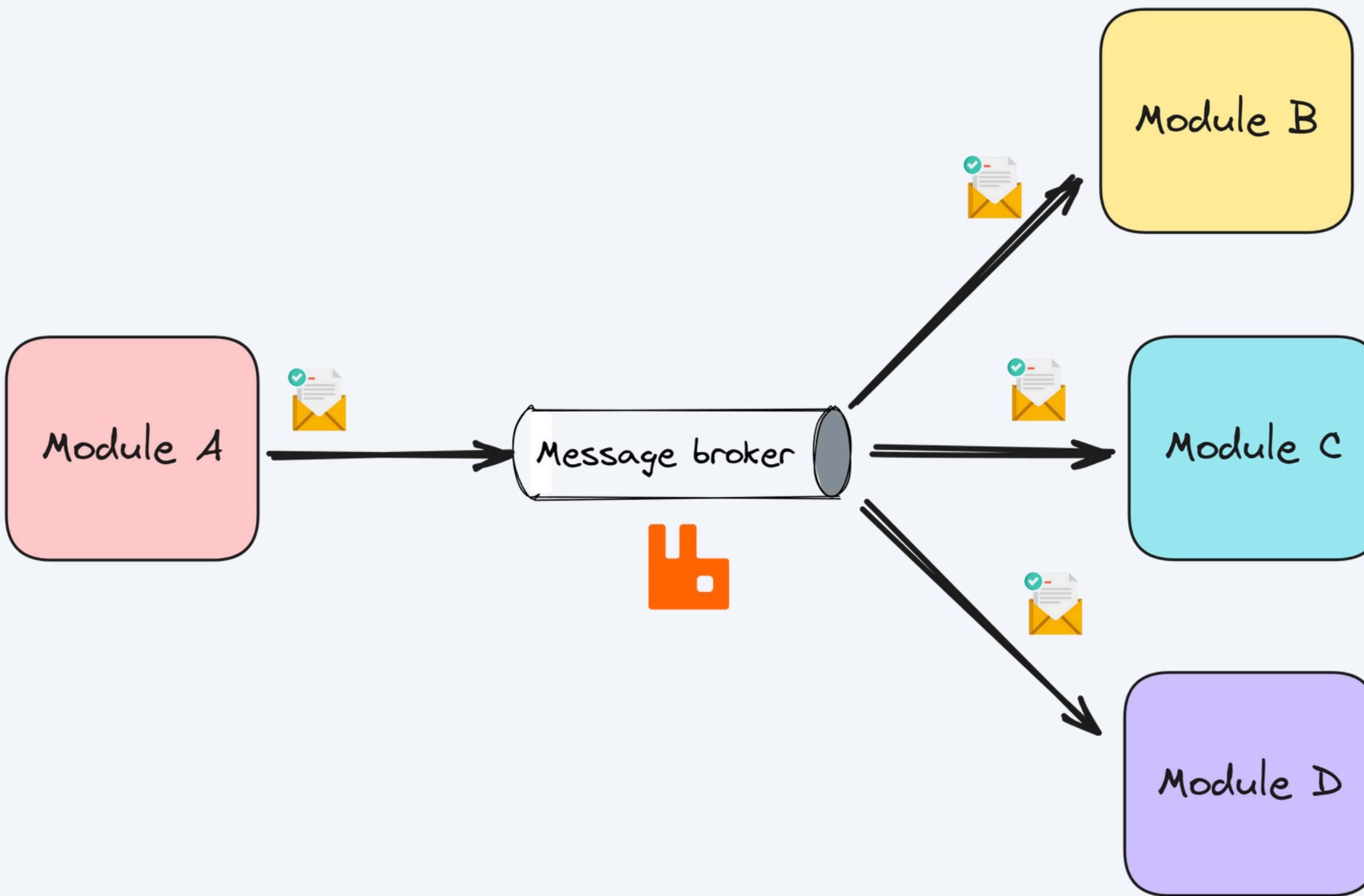
Inbox Pattern

Intro

Inbox pattern for integration events.

At Least Once processing.

Idempotent consumers.





**Next:
Event-Driven
Architecture**

Event-Driven Architecture



Intro

Event-driven architecture.

Benefits of being event-driven.

Orchestration, distributed transactions.

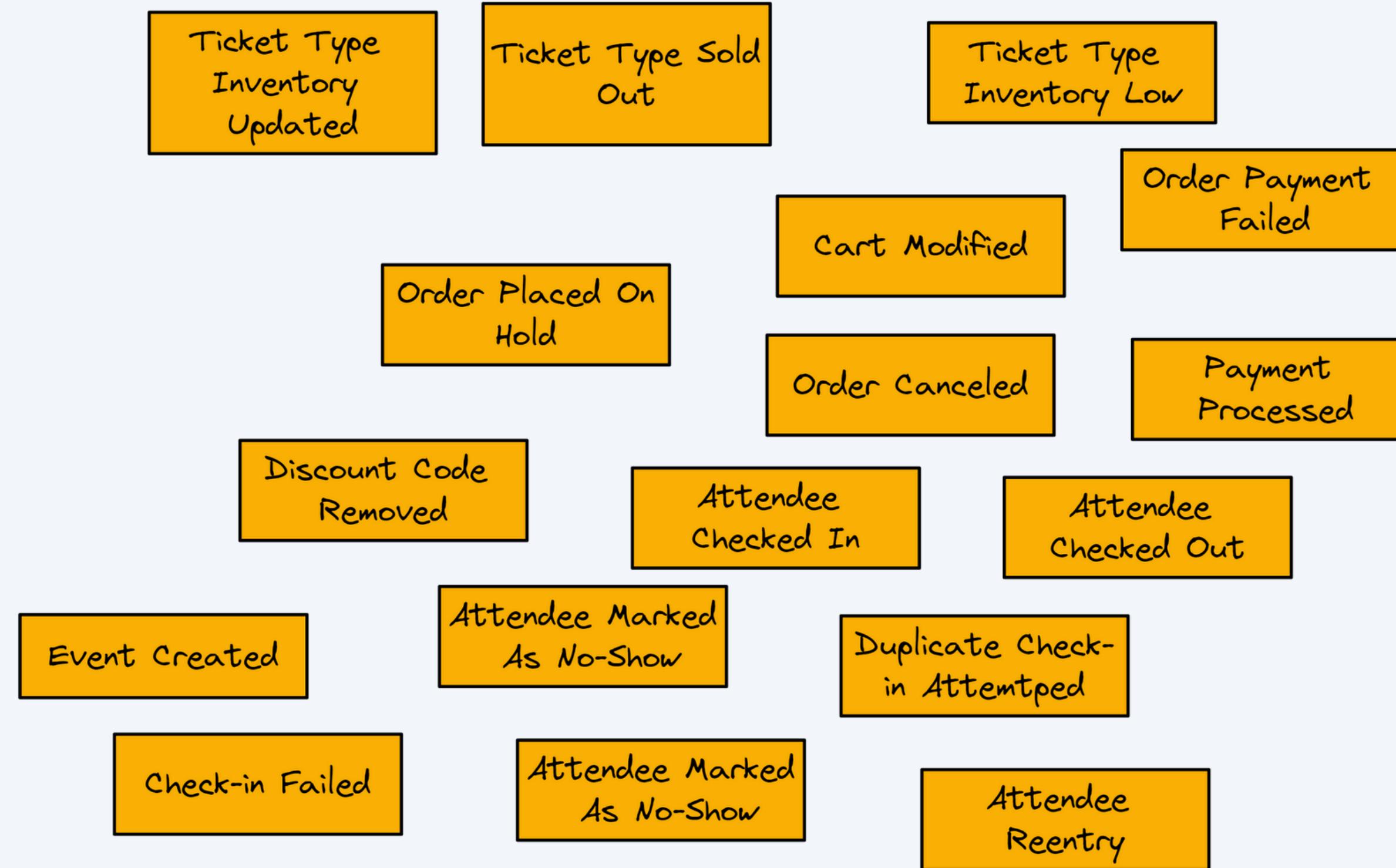
What is an Event?

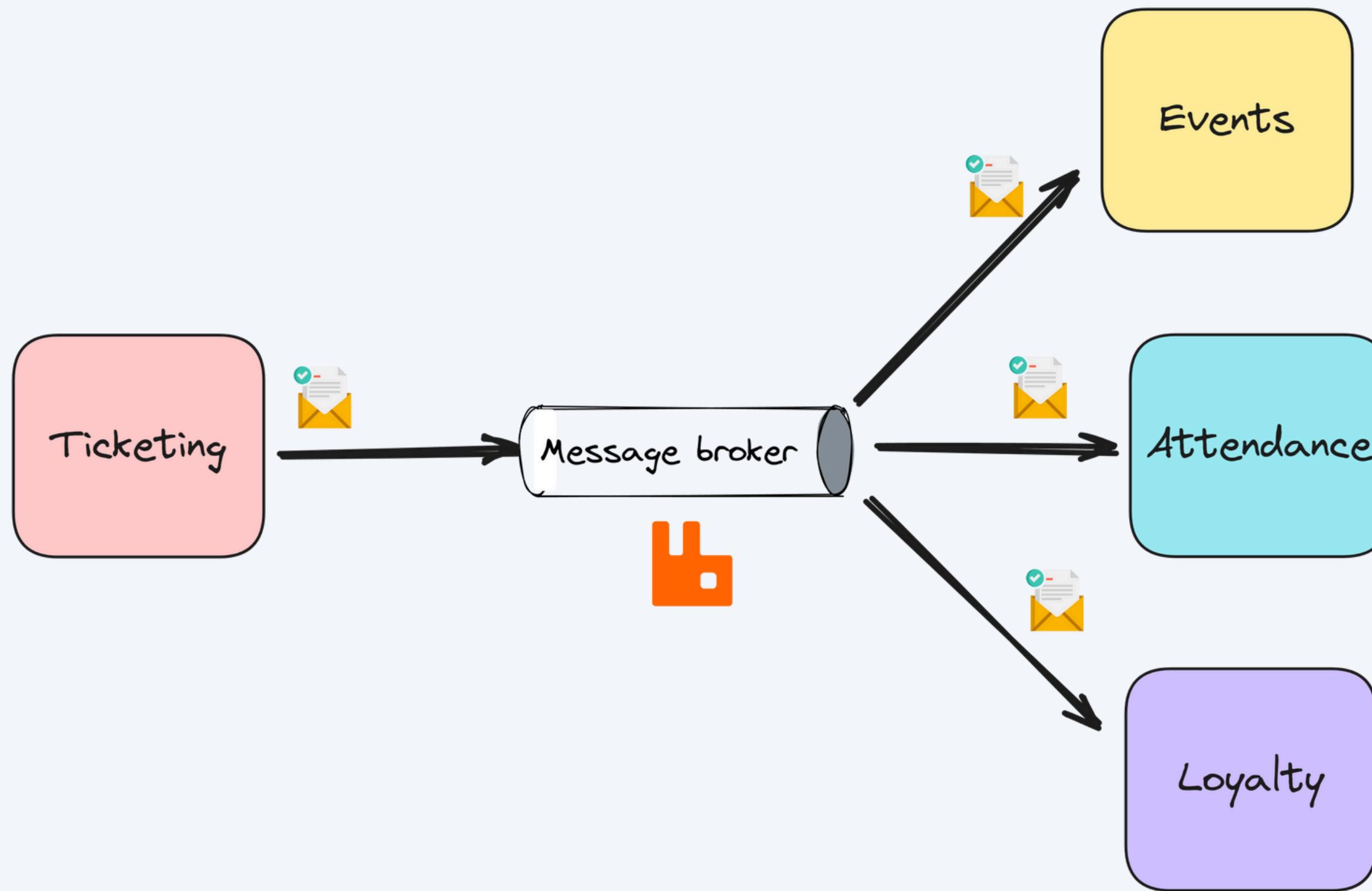
An event is an immutable *fact*, something that has happened in the past and cannot be changed.

Event-driven

What does it mean to be event-driven?

Driven by business events.





Patterns of EDA

You are *event-driven* if you are using at least one of these patterns.

- Event sourcing
- Event notifications
- Event-carried state transfer
- CQRS

Event Sourcing



Event Notifications

```
9 references
public sealed class OrderCreatedDomainEvent(Guid orderId) : DomainEvent
{
    | 4 references | 0/1 passing
    | public Guid OrderId { get; init; } = orderId;
}
```

Event-Carried State Transfer

```
7 references
public sealed class EventPublishedIntegrationEvent : IIntegrationEvent
{
    5 references
    public Guid Id { get; init; }

    3 references
    public Guid EventId { get; init; }

    2 references
    public string Title { get; init; }

    2 references
    public string Description { get; init; }

    2 references
    public string Location { get; init; }

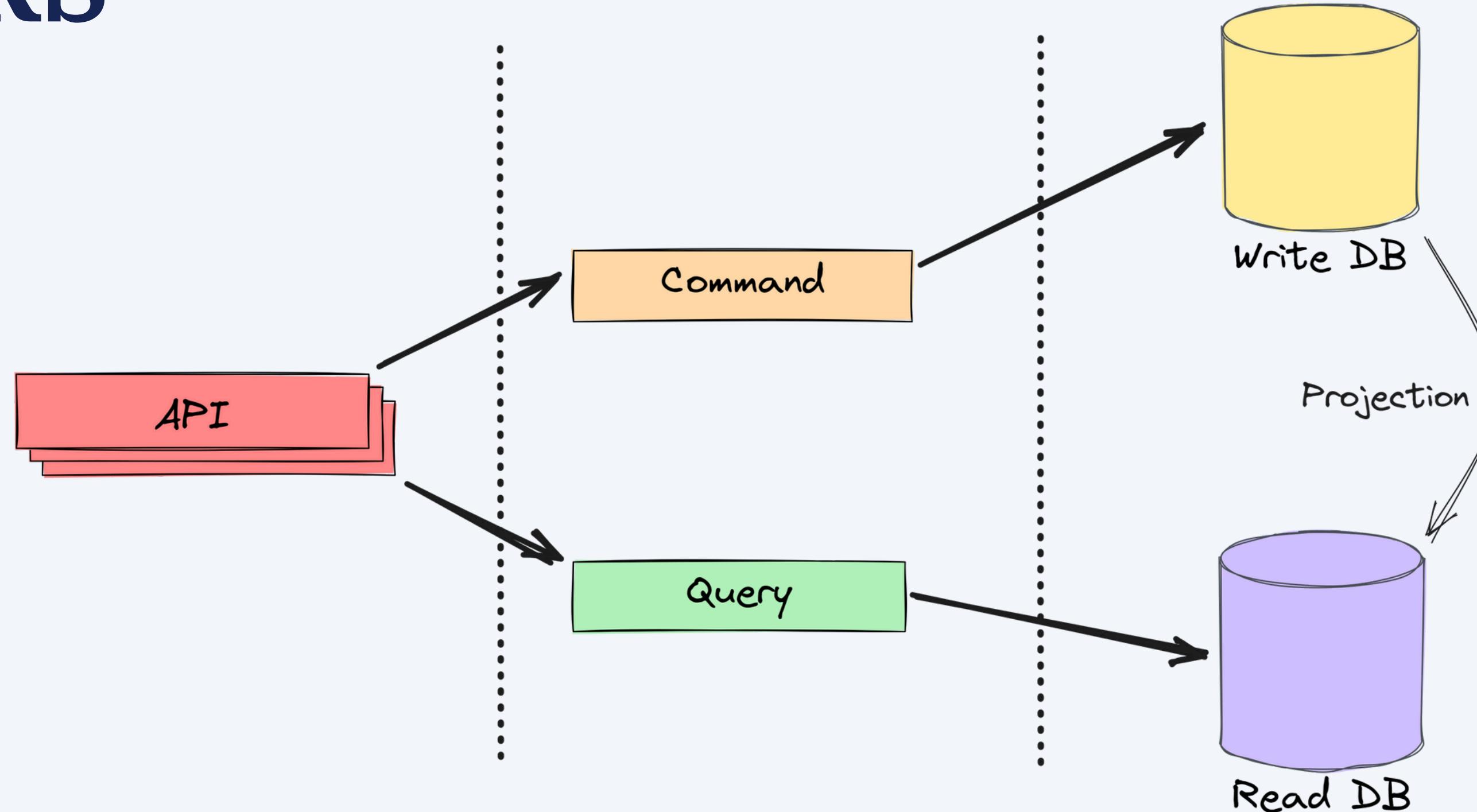
    2 references
    public DateTime StartsAtUtc { get; init; }

    2 references
    public DateTime? EndsAtUtc { get; init; }

    1 reference
    public List<TicketTypeModel> TicketTypes { get; init; }

    1 reference
    public DateTime OccurredOnUtc { get; init; }
}
```

CQRS



Domain Event and Integration Event

Inside Event and Outside Event

Internal Event and Public Event

Domain Event and Integration Event (my preference)

Where Is EDA Useful?

- Large business applications
- Complex enterprise systems

EDA Use Cases

- External integration
- Temporal decoupling
- State transfer
- Workflows

Temporal Decoupling

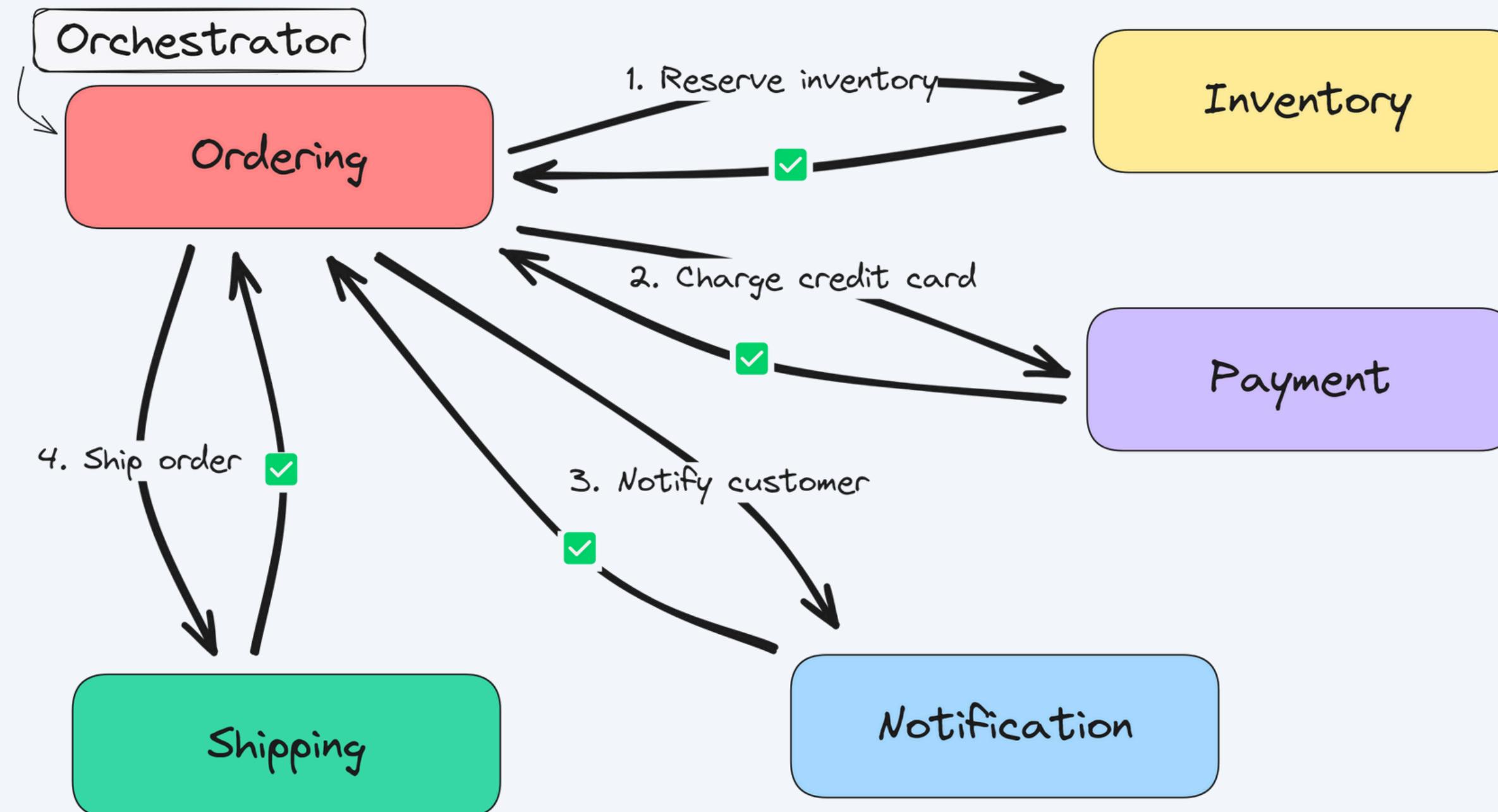
```
// Placing an order -----
orderRepository.Insert(order);

// Order payment -----
PaymentResponse paymentResponse = await paymentService.ChargeAsync(order.TotalPrice, order.Currency);

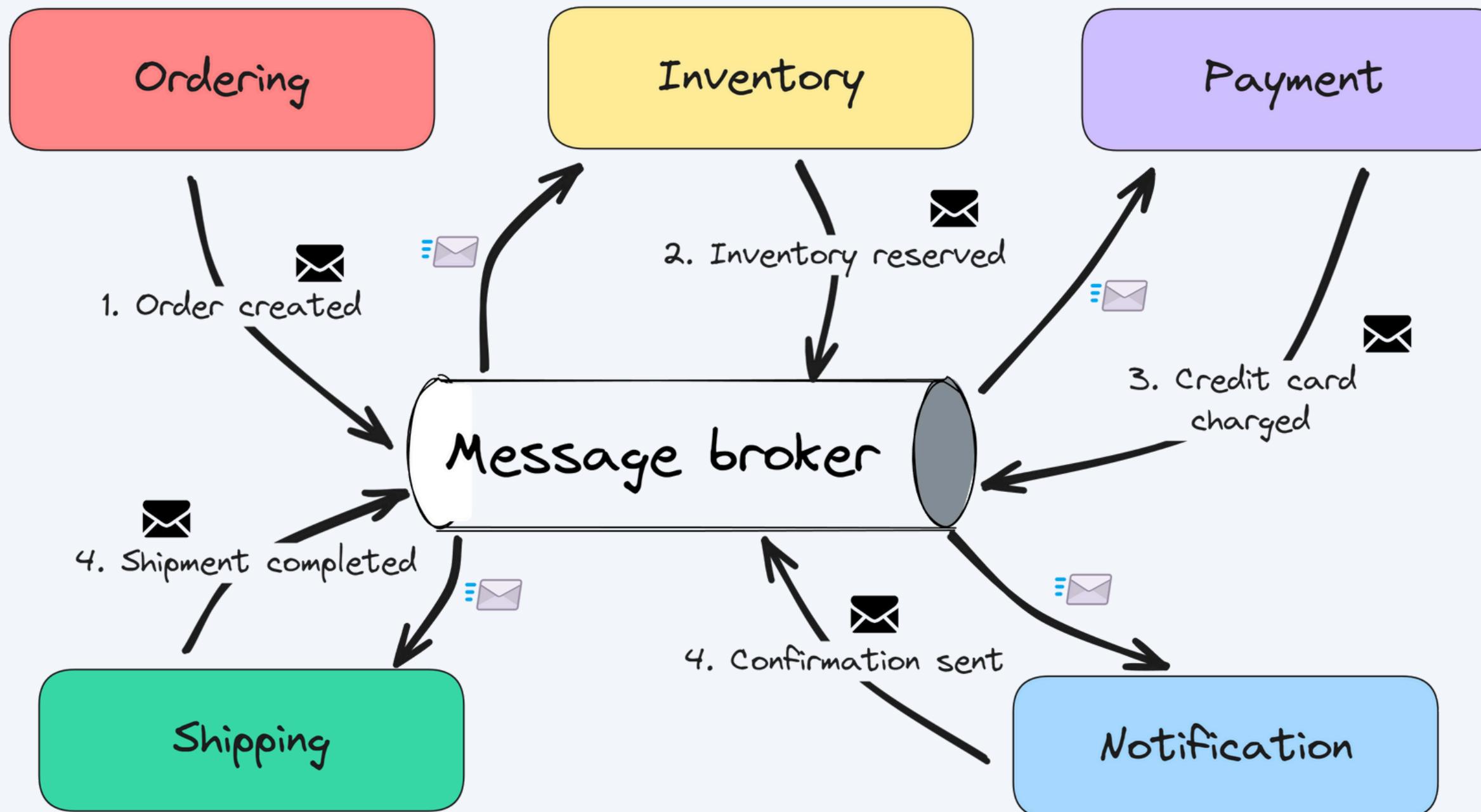
paymentRepository.Insert(
    Payment.Create(order, paymentResponse.TransactionId, paymentResponse.Amount, paymentResponse.Currency));

// Complete transaction -----
await unitOfWork.SaveChangesAsync(cancellationToken);
```

Orchestration



Choreography



Embrace eventual consistency



Next:
Event Notifications



Event Notifications



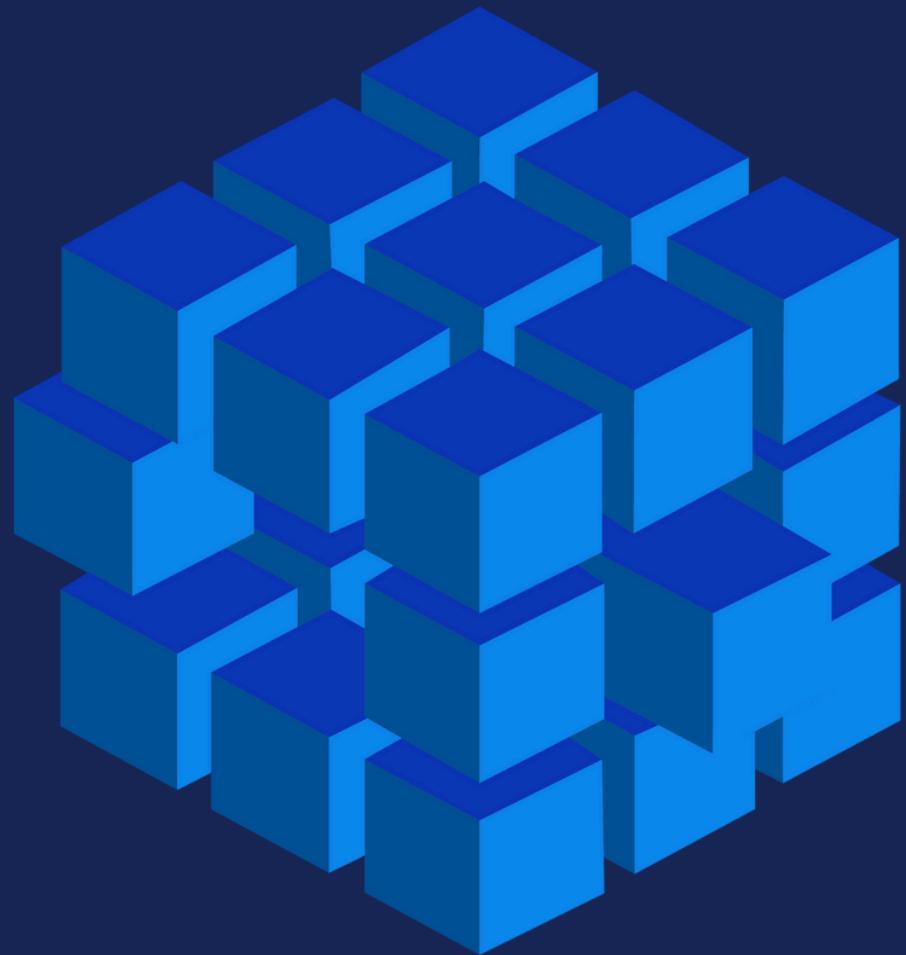
Next:
Event-Carried State
Transfer



Event-Carried State Transfer



Next:
Materialized Views
and CQRS



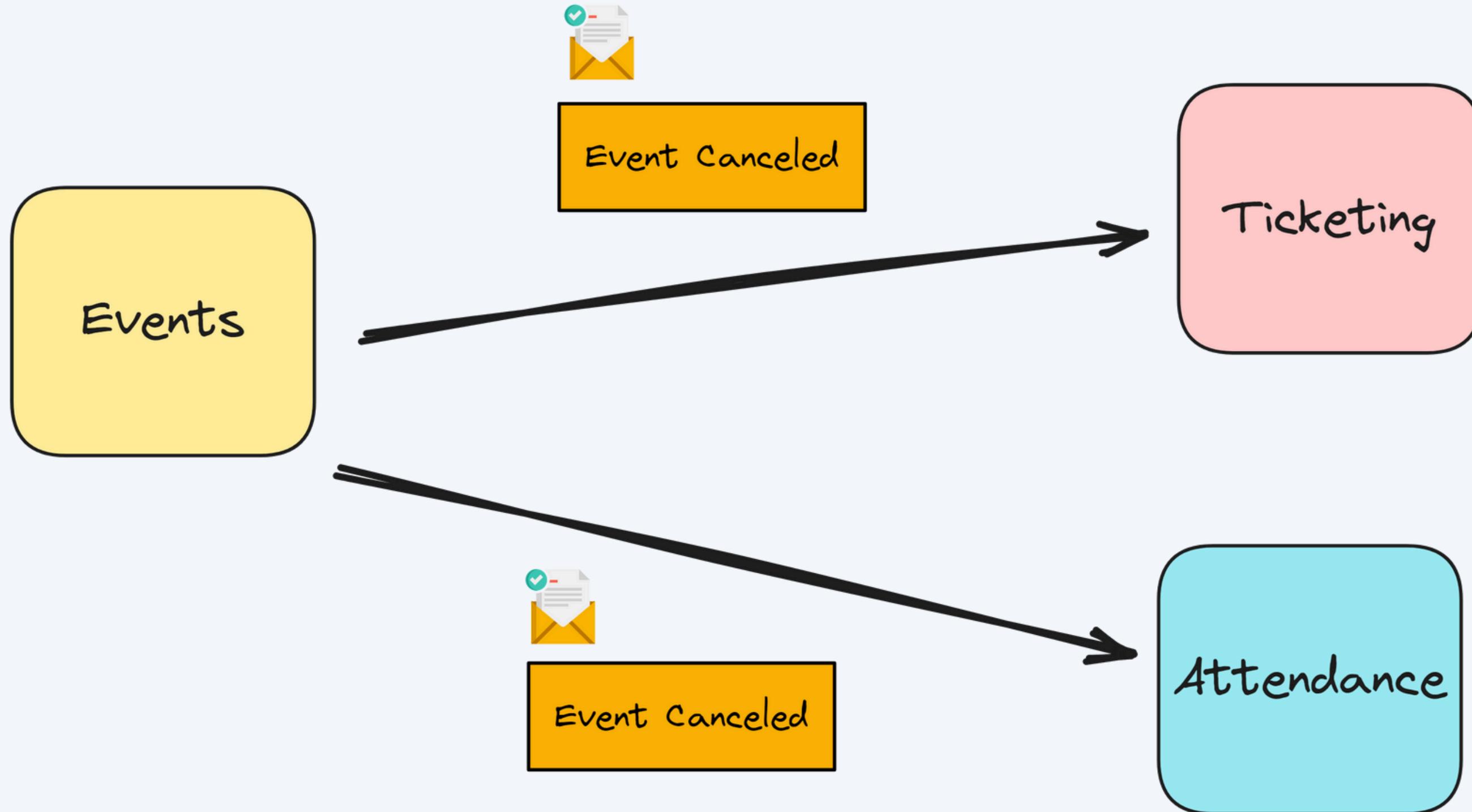
Materialized Views and CQRS

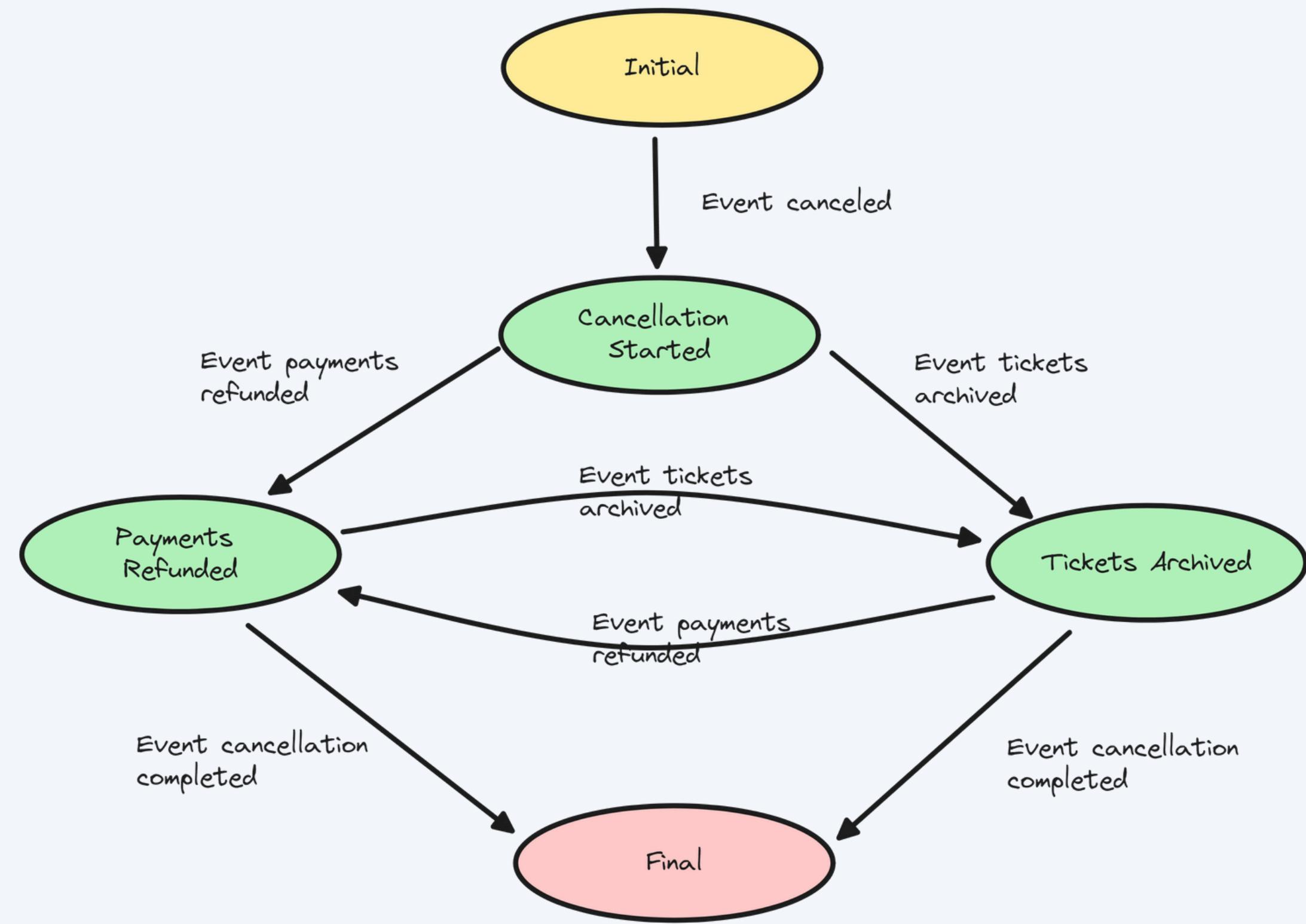


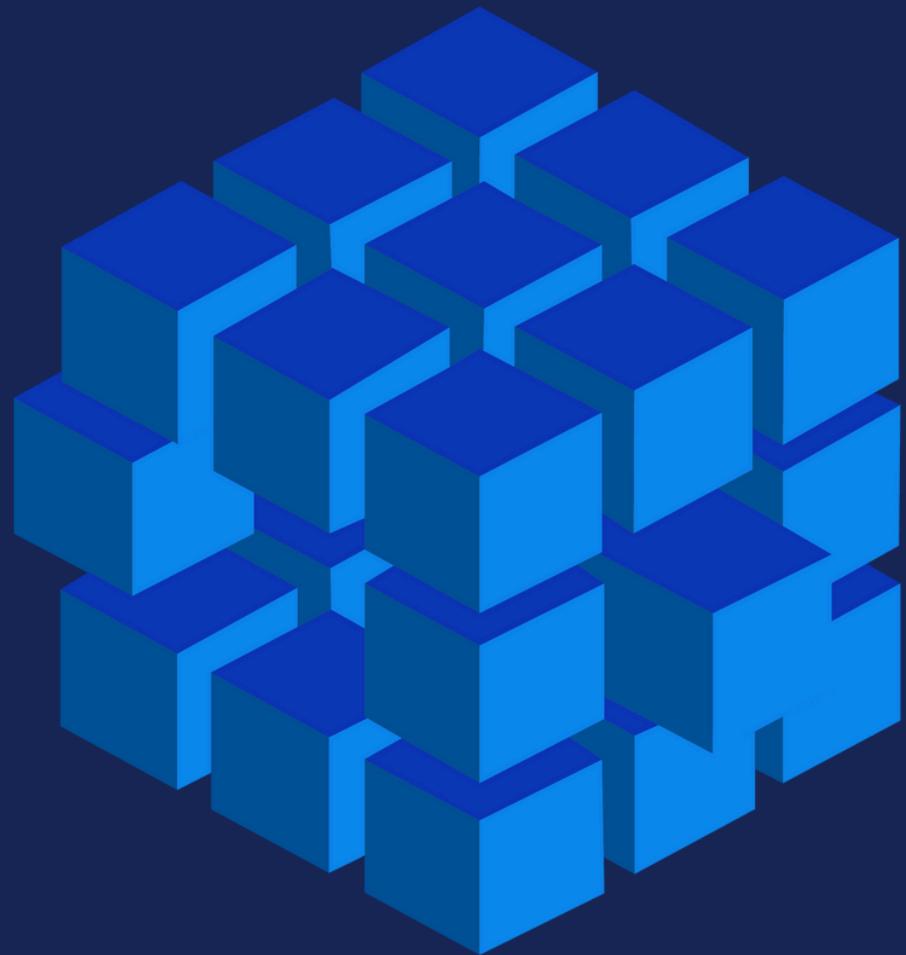
Next:
Saga Pattern,
Orchestration



Saga Pattern, Orchestration







Next:
Testing Modular
Monoliths

Testing Modular Monoliths

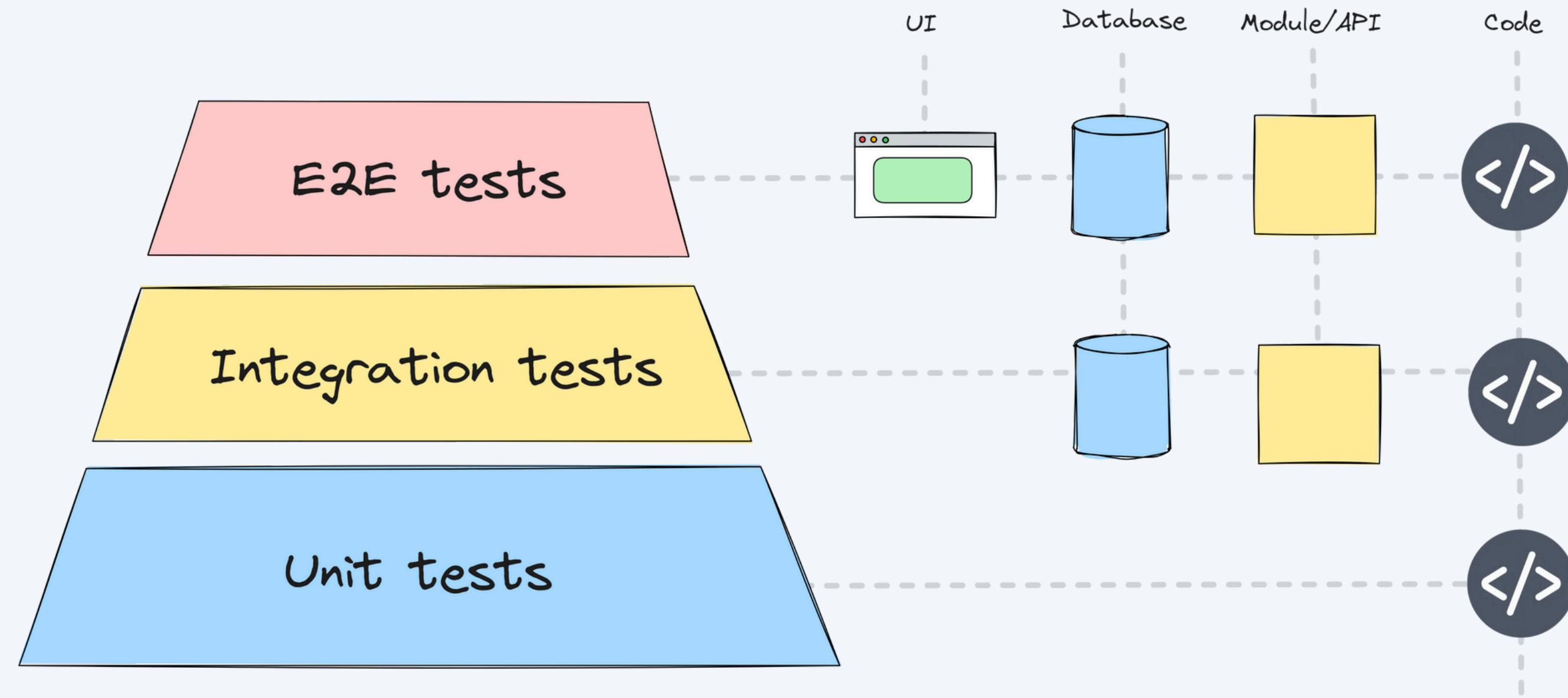


Intro

What should we test?

Code coverage as a quality metric.

Unit testing, Integration testing.



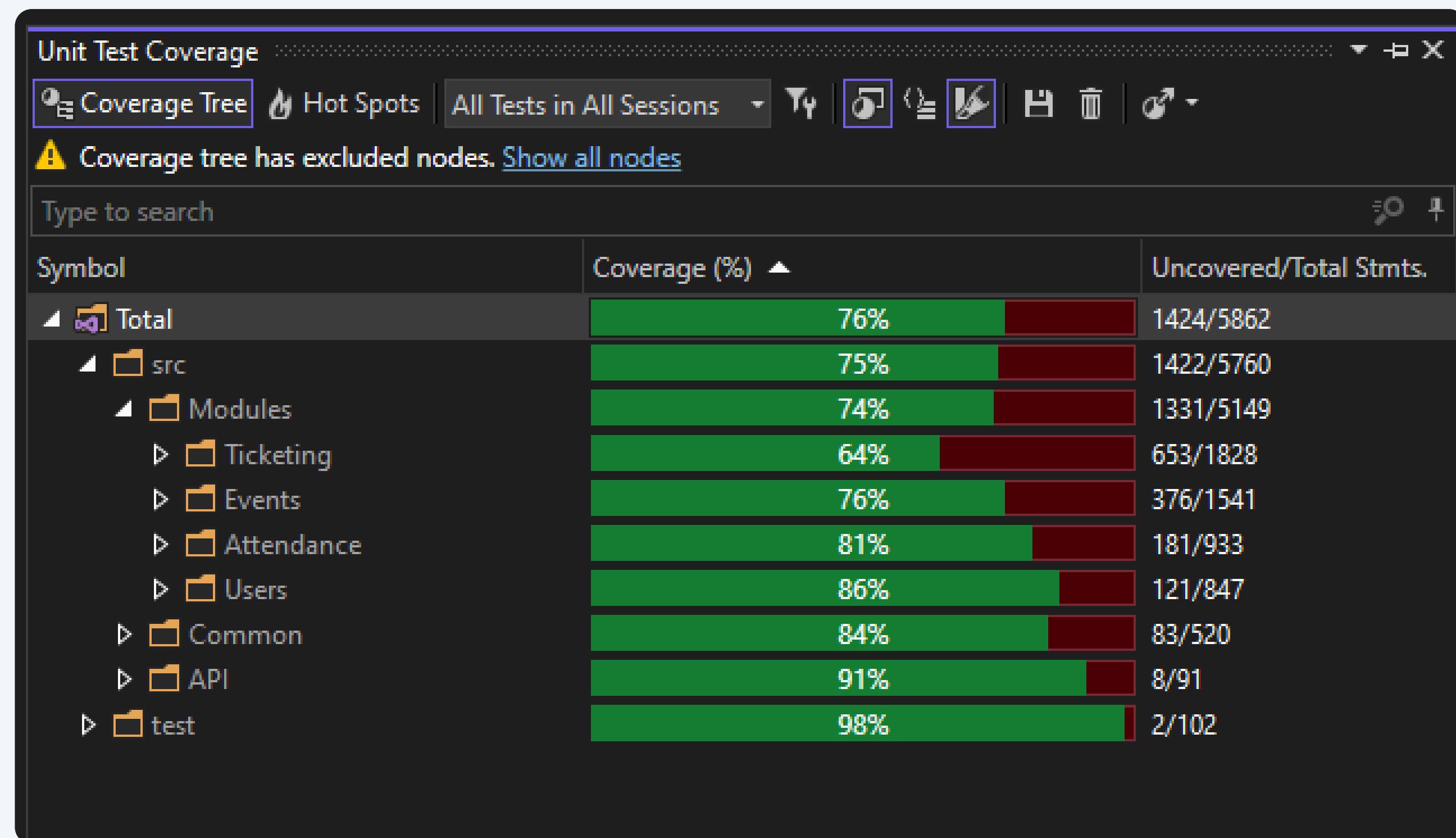
Test Explorer

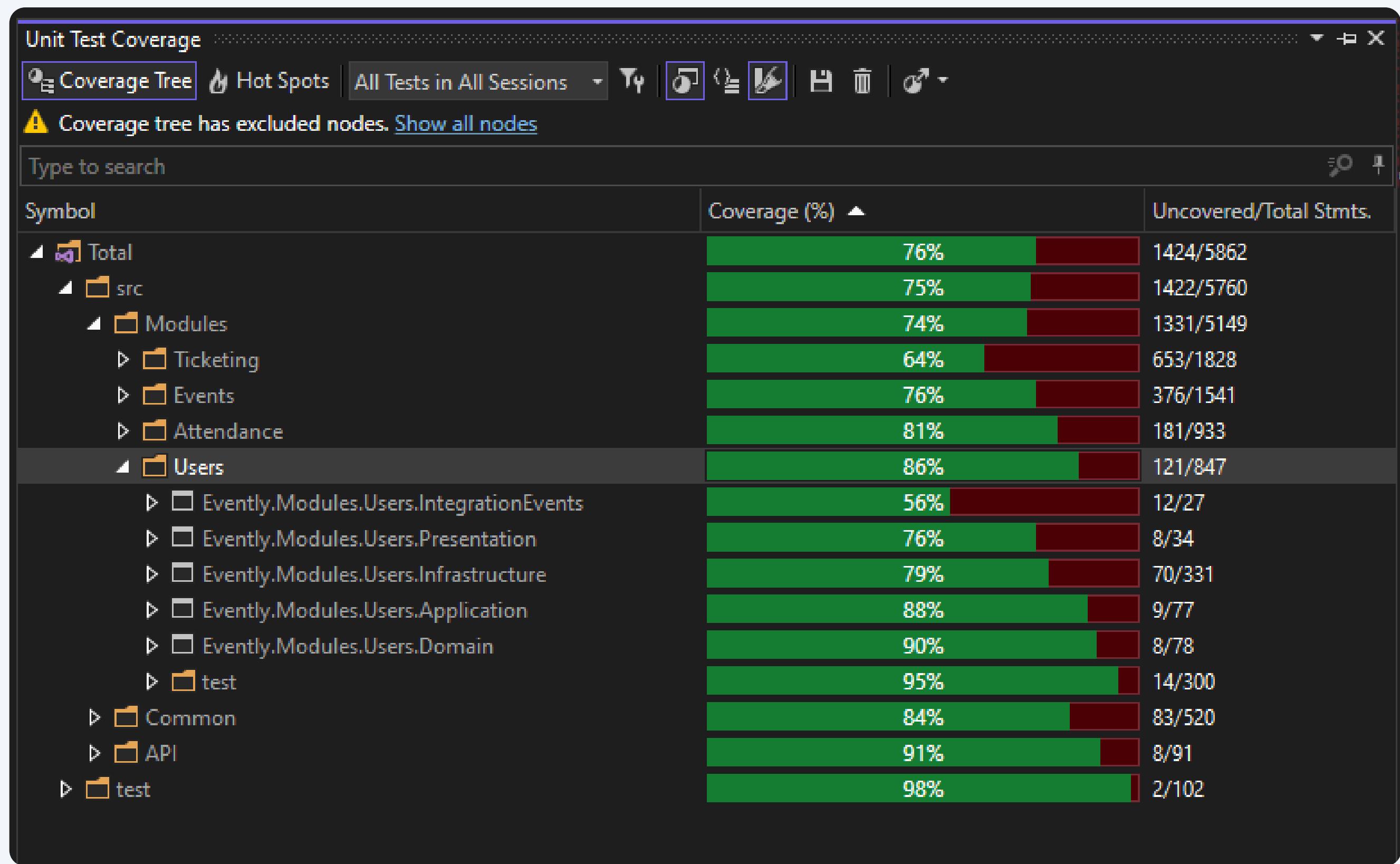
247 247 0

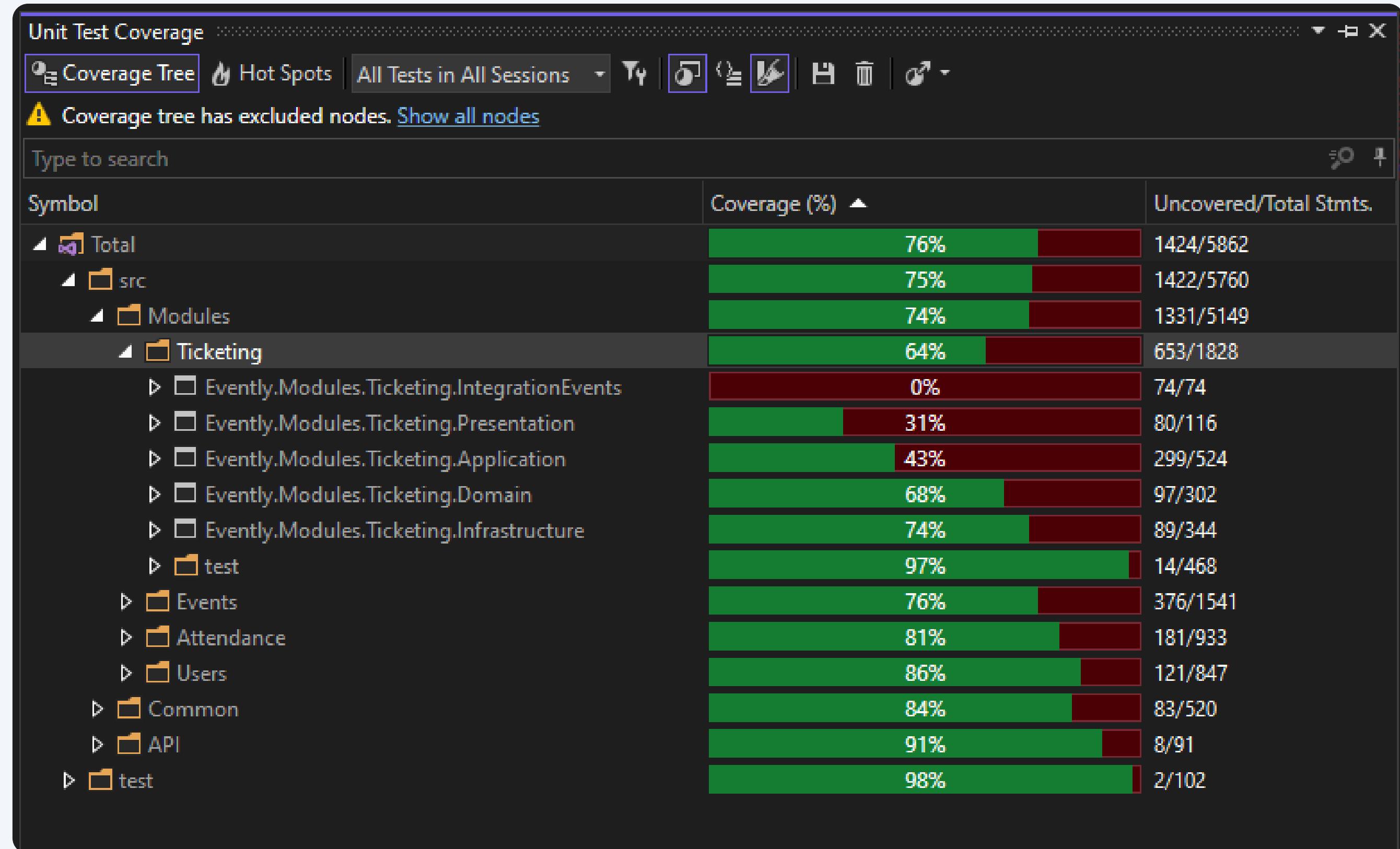
Search (Ctrl+I)

Test run finished: 254 Tests (254 Passed, 0 Failed, 0 Warnings, 0 Errors)

Test	Duration	Trait
Evently.ArchitectureTests (4)	229 ms	
Evently.IntegrationTests (1)	12.1 sec	
Evently.Modules.Attendance.ArchitectureTests (..)	524 ms	
Evently.Modules.Attendance.IntegrationTests (...)	2.2 sec	
Evently.Modules.Attendance.UnitTests (6)	234 ms	
Evently.Modules.Events.ArchitectureTests (28)	642 ms	
Evently.Modules.Events.IntegrationTests (38)	2.9 sec	
Evently.Modules.Events.UnitTests (13)	182 ms	
Evently.Modules.Ticketing.ArchitectureTests (28)	579 ms	
Evently.Modules.Ticketing.IntegrationTests (23)	2.7 sec	
Evently.Modules.Ticketing.UnitTests (16)	269 ms	
Evently.Modules.Users.ArchitectureTests (28)	410 ms	
Evently.Modules.Users.IntegrationTests (16)	3.9 sec	
Evently.Modules.Users.UnitTests (5)	50 ms	









Next:
Unit Testing



Unit Testing



Next:
Integration Testing



Integration Testing



**Next:
System Integration
Testing**



System Integration Testing

```

[Fact]
● 0 references
public async Task RegisterUser_Should_PropagateToTicketingModule2()
{
    // Register user
    var command = new RegisterUserCommand(
        Faker.Internet.Email(),
        Faker.Internet.Password(6),
        Faker.Name.FirstName(),
        Faker.Name.LastName());

    Result<Guid> userResult = await Sender.Send(command);

    userResult.IsSuccess.Should().BeTrue();

    Result<CustomerResponse> customerResult = await Poll(
        TimeSpan.FromSeconds(15),
        async () =>
    {
        var query = new GetCustomerQuery(userResult.Value);

        Result<CustomerResponse> customerResult = await Sender.Send(query);

        return customerResult;
    });

    // Get customer
    customerResult.IsSuccess.Should().BeTrue();
    customerResult.Value.Should().NotBeNull();
}

1 reference | ● 1/1 passing
private static async Task<Result<T>> Poll<T>(TimeSpan timeout, Func<Task<Result<T>>> action)
{
    using var timer = new PeriodicTimer(TimeSpan.FromSeconds(1));
    DateTime endTime = DateTime.UtcNow.Add(timeout);
    while (DateTime.UtcNow < endTime && await timer.WaitForNextTickAsync())
    {
        Result<T> result = await action();

        if (result.IsSuccess)
        {
            return result;
        }
    }

    return Result.Failure<T>(Error.NullValue);
}

```



Next:
Automated Testing in
CI/CD Pipelines



Automated Testing in CI/CD Pipelines

Intro

The value of automated testing in CI/CD.

GitHub Actions.

Value of Automated Testing

Immediate feedback.

Regression testing.

CI/CD using GitHub Actions



Workflow

```
1  name: Build
2
3  on:
4    workflow_dispatch:
5    push:
6      branches:
7        - main
8
9  env:
10   DOTNET_VERSION: "8.x"
11
12 jobs:
13   build:
14     runs-on: ubuntu-latest
15
16   defaults:
17     run:
18       working-directory: Evently
19
20   steps:
21     - uses: actions/checkout@v4
22
23     - name: Setup .NET
24       uses: actions/setup-dotnet@v4
25       with:
26         dotnet-version: ${{ env.DOTNET_VERSION }}
27
28     - name: Restore
29       run: dotnet restore Evently.sln
30
31     - name: Build
32       run: dotnet build Evently.sln --configuration Release --no-restore
33
34     - name: Test
35       run: dotnet test Evently.sln --configuration Release --no-restore --no-build
36
37     - name: Publish
38       run: dotnet publish Evently.sln --configuration Release --no-restore --no-build
```

← Build

✓ feat(test): Ticketing integration tests #104

Re-run all jobs

...

Summary

Jobs

✓ build

Run details

⌚ Usage

⤷ Workflow file

Triggered via push 3 hours ago	Status	Total duration	Billable time	Artifacts
 m-jovanovic pushed -o 7655720 main	Success	4m 2s	4m	-

build.yml	on: push
 build	3m 54s

⤷ - +

← Build

✓ **feat(test): Ticketing integration tests #104**

Re-run all jobs

...

Summary

Jobs

✓ build

Run details

⌚ Usage

⤷ Workflow file

build
succeeded 2 hours ago in 3m 54s

Beta Give feedback Search logs

- > ✓ Set up job 1s
- > ✓ Run actions/checkout@v4 1s
- > ✓ Setup .NET 1s
- > ✓ Restore 14s
- > ✓ Build 58s
- > ✓ Test 2m 35s
- > ✓ Post Setup .NET 0s
- > ✓ Post Run actions/checkout@v4 1s
- > ✓ Complete job 0s



Next:
Microservices:
Extracting Modules

Microservices: Extracting Modules



Intro

Distributed tracing, OpenTelemetry.

Distributed messaging, RabbitMQ.

API Gateways, introducing YARP.

Extracting modules to microservices.

Why move to Microservices?

Microservices Benefits

Independent deployability.

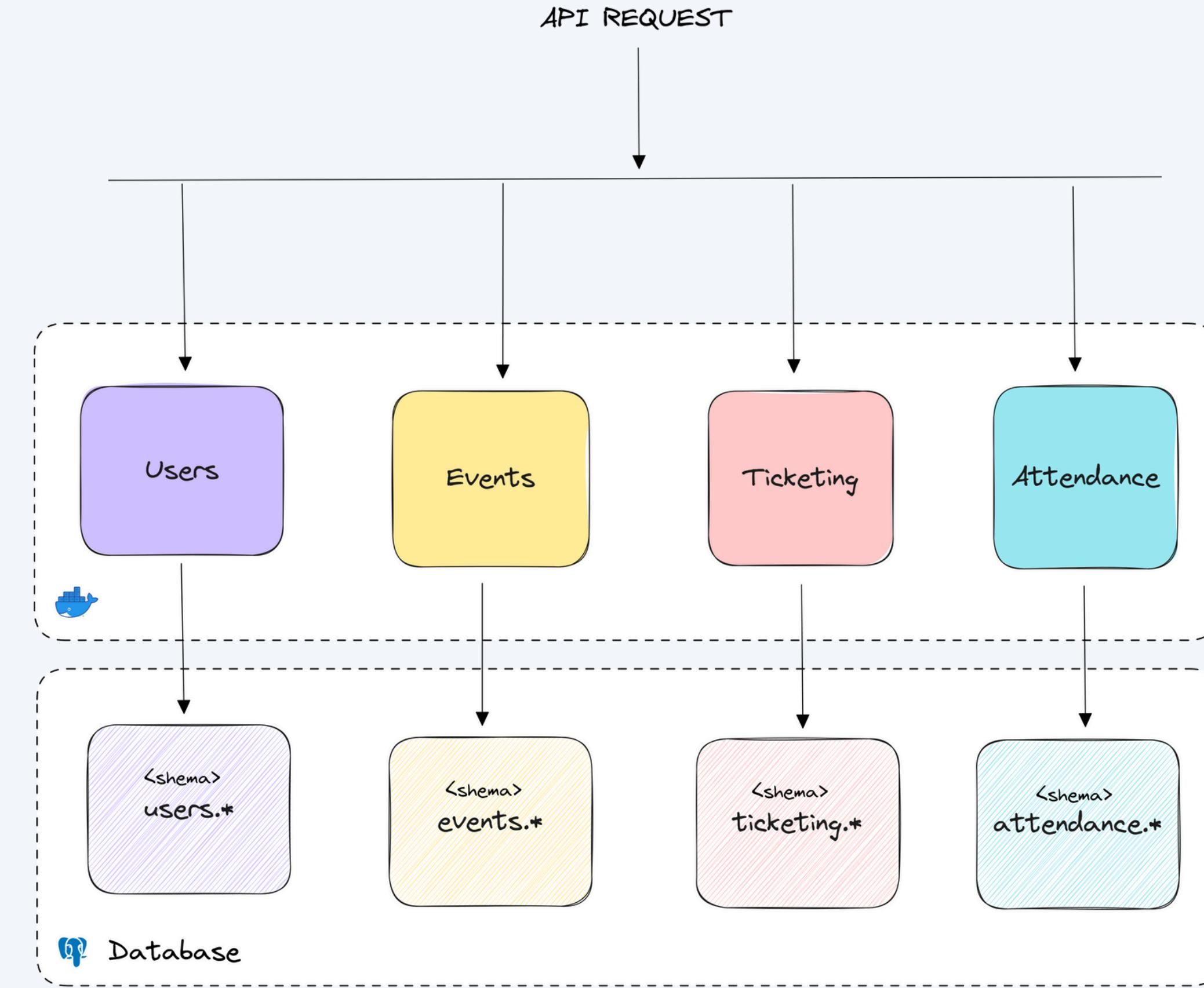
Organizational autonomy.

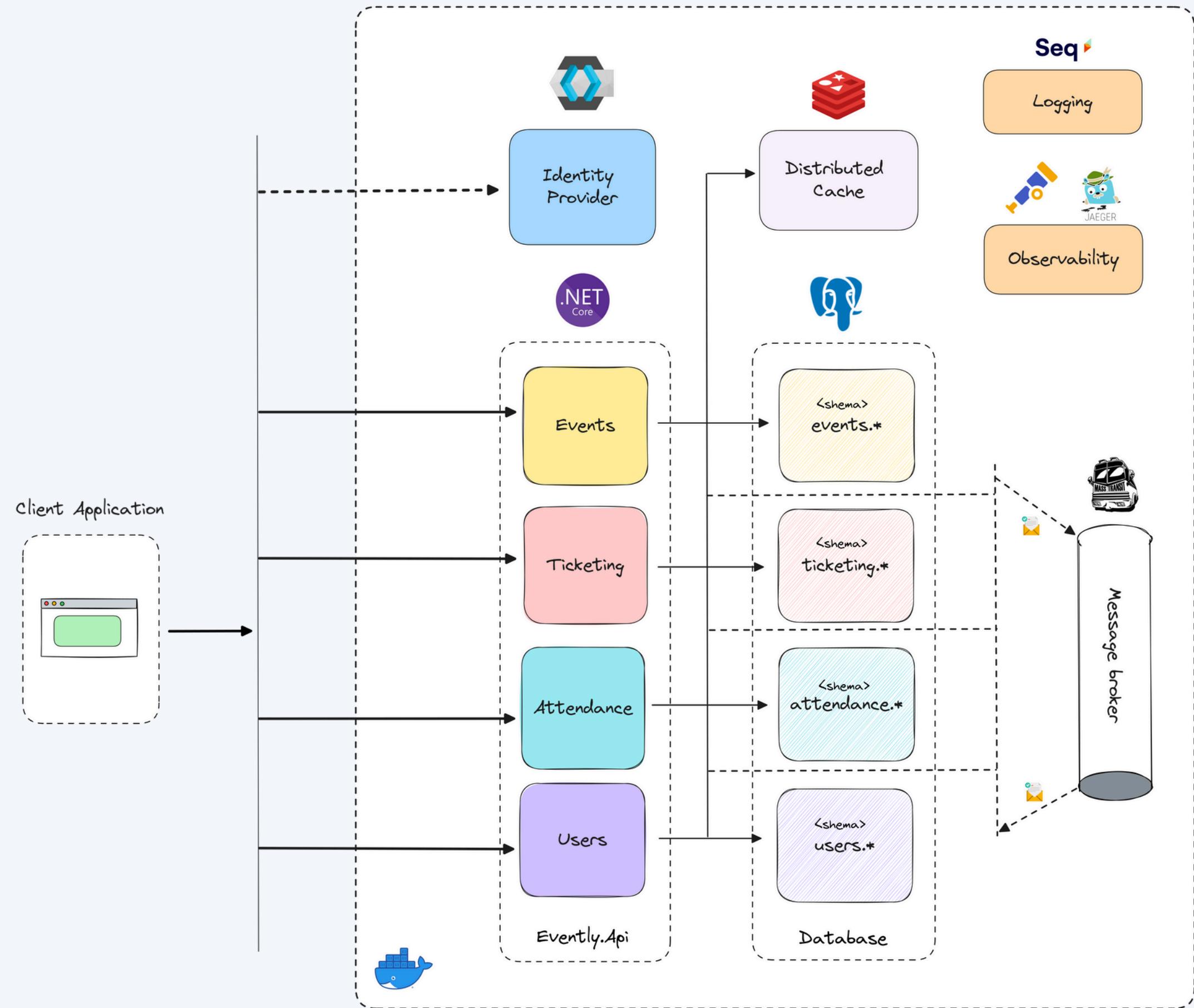
Fault isolation.

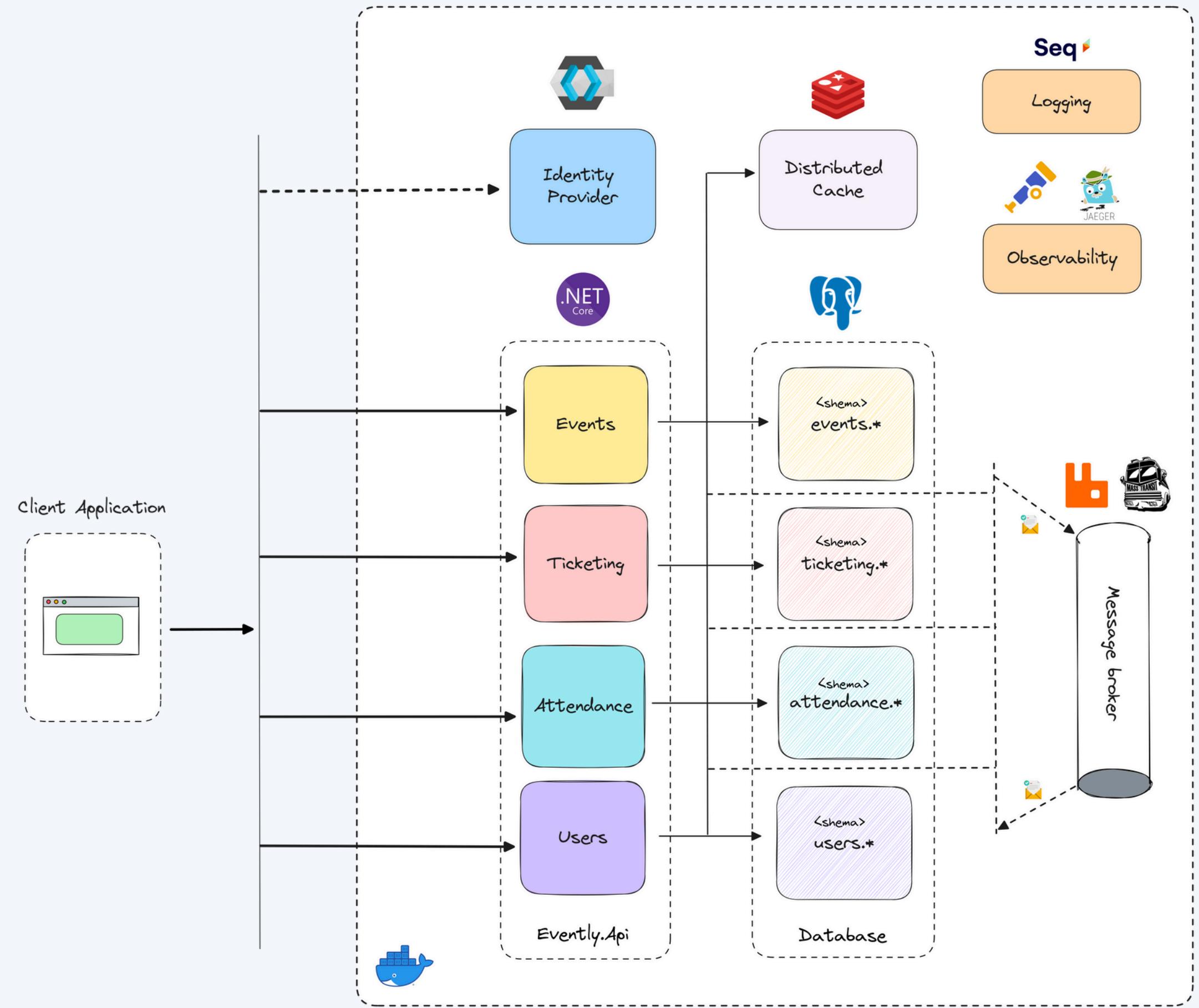
Scalability.

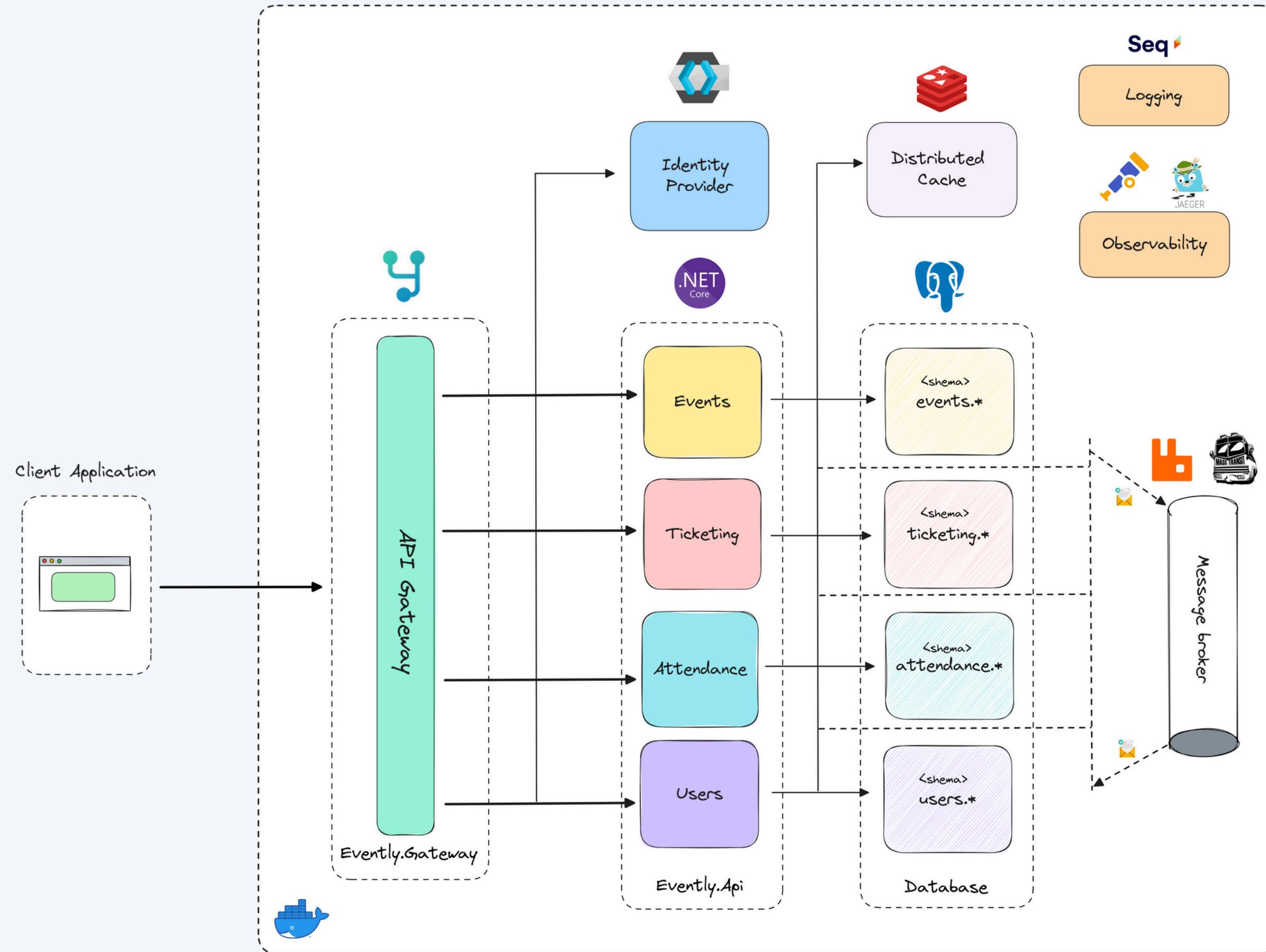
Modular monoliths make the
transition straightforward.

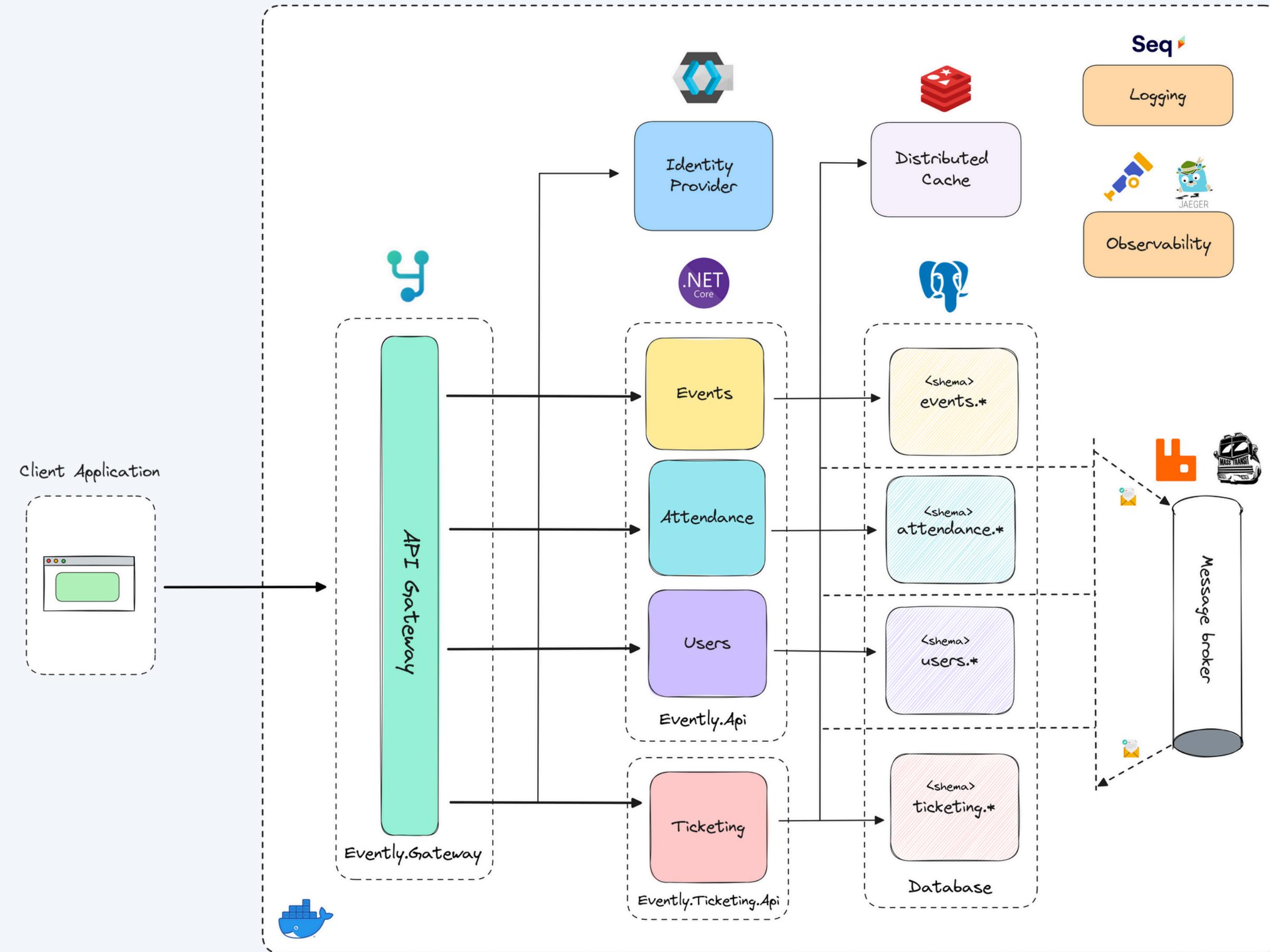
Well-defined, in-process components
(modules) can be an excellent
stepping stone to out-of-process
components (services).

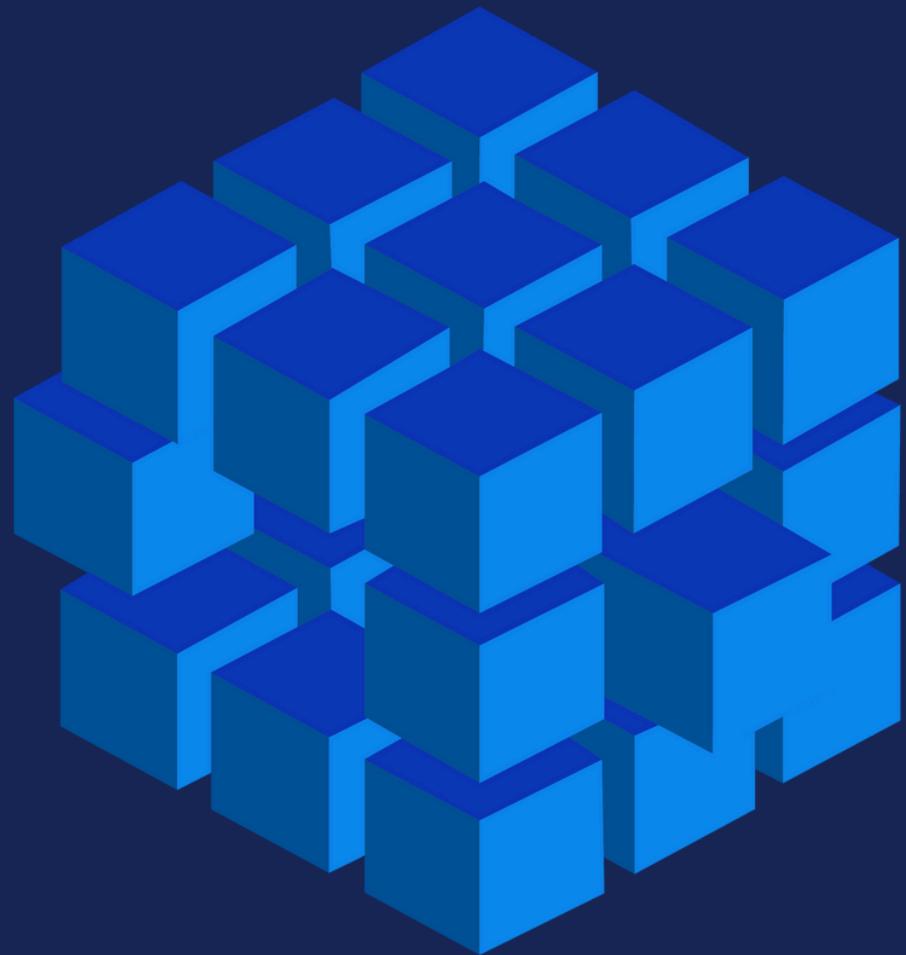












Next:
Distributed Tracing,
OpenTelemetry



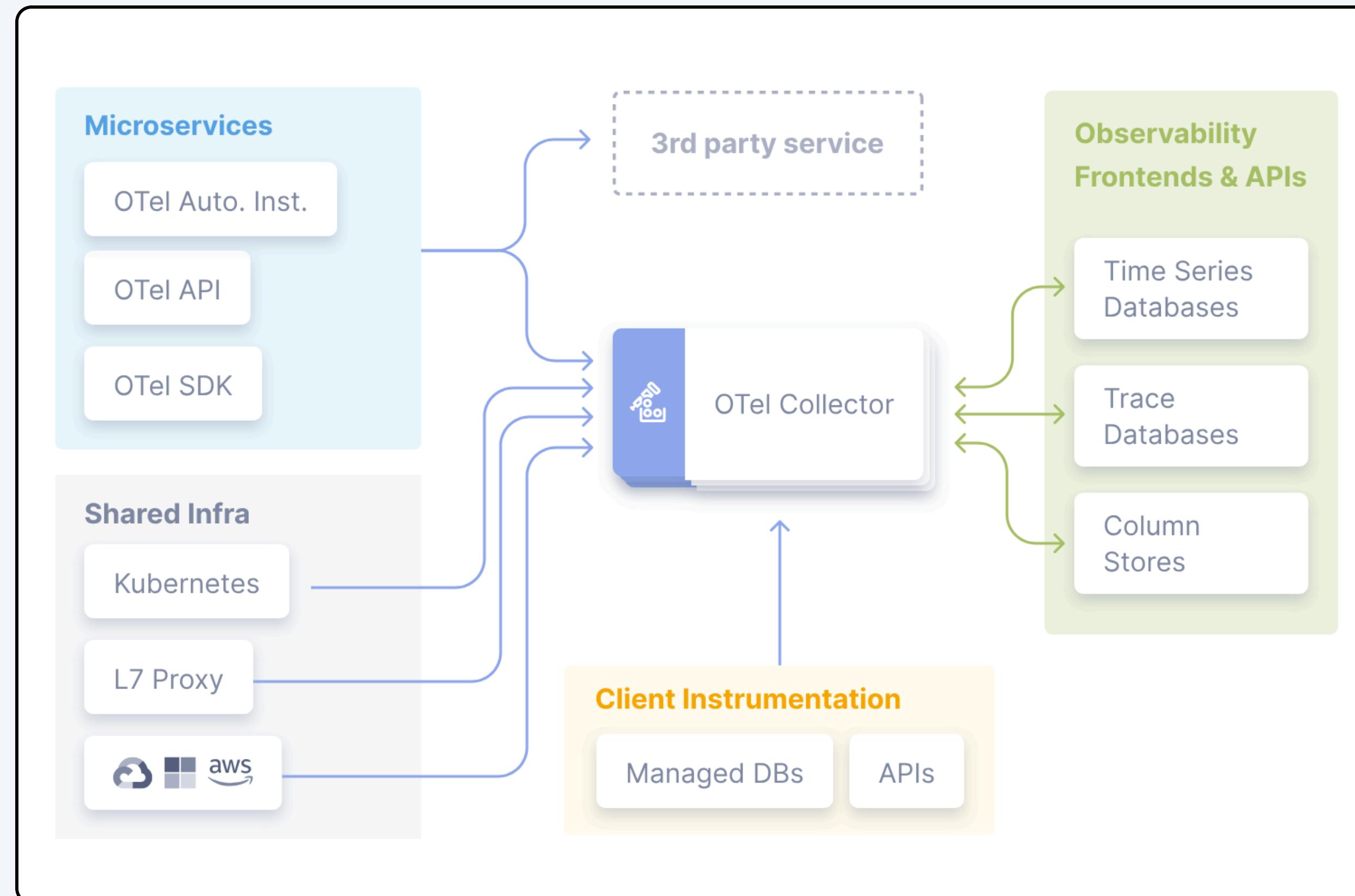
Distributed Tracing, OpenTelemetry

What is OpenTelemetry?

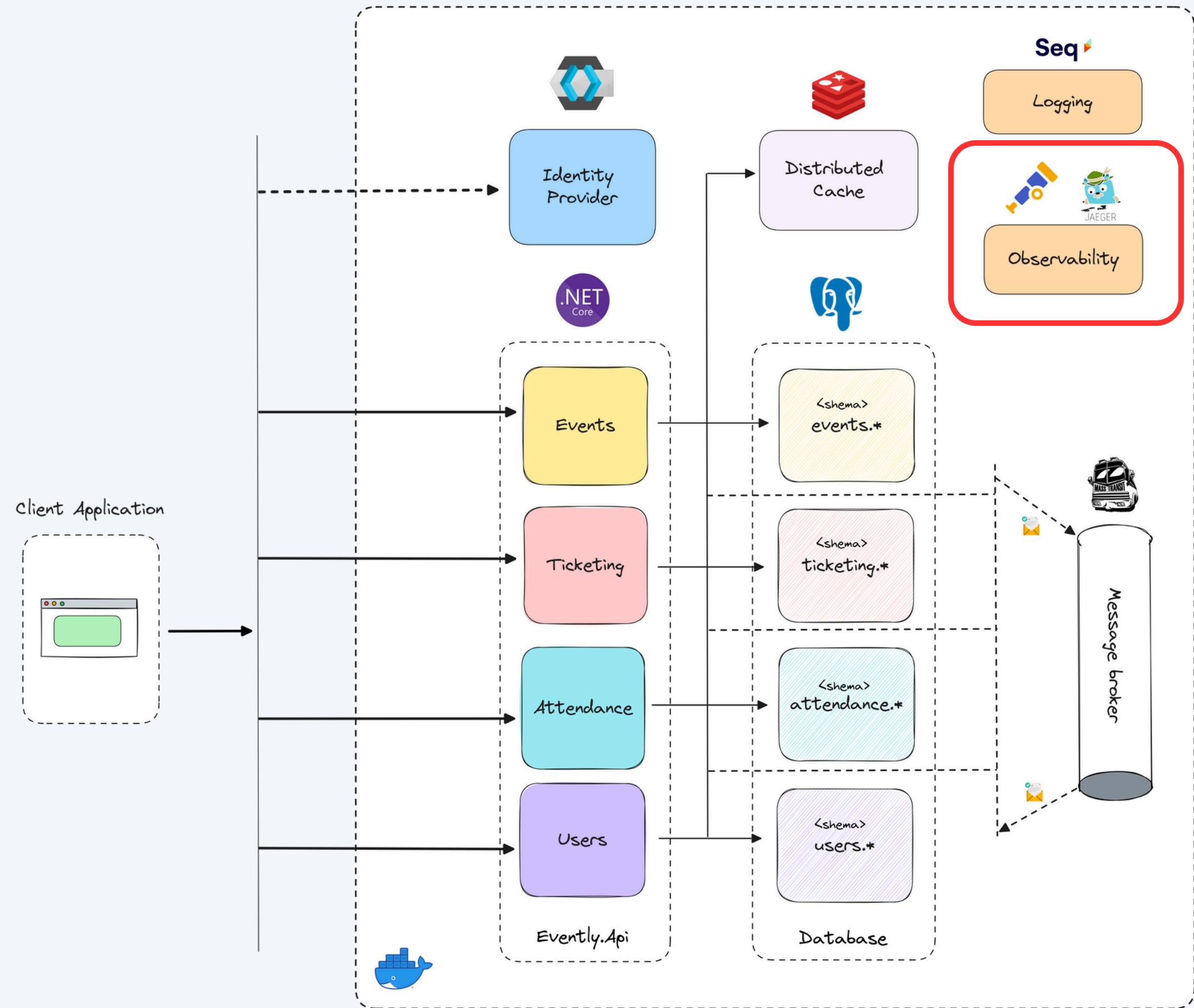
Open-source standard for instrumenting applications.

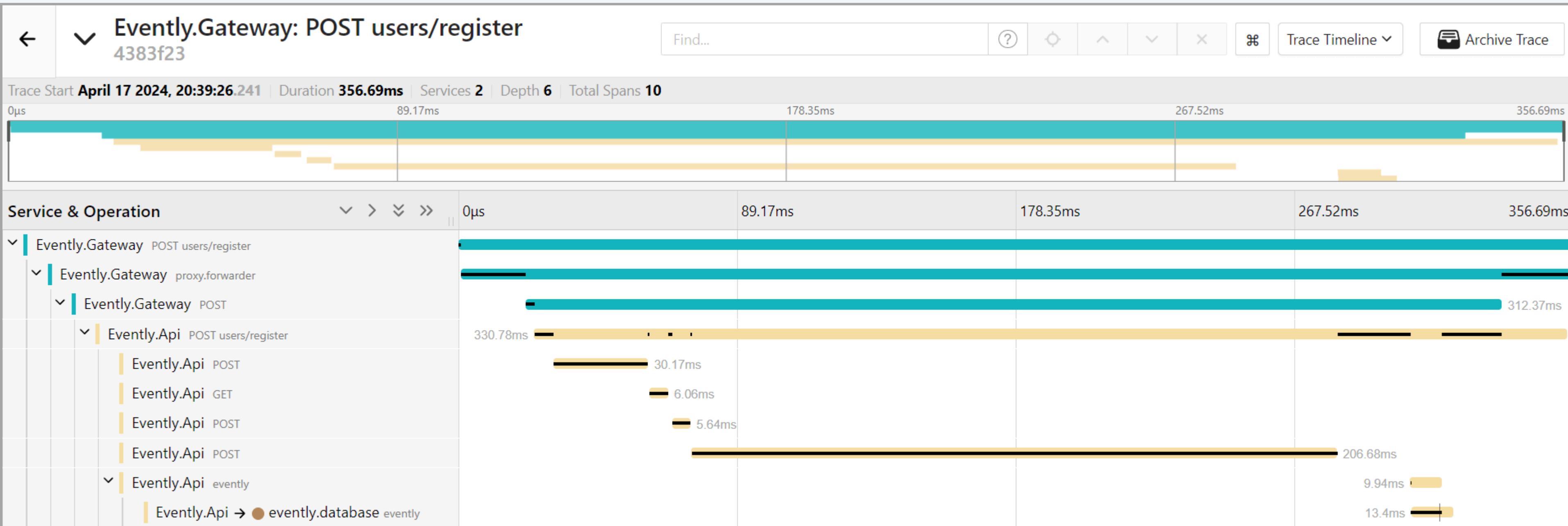
Telemetry data: traces, metrics, logs.





Source: <https://opentelemetry.io/docs/>

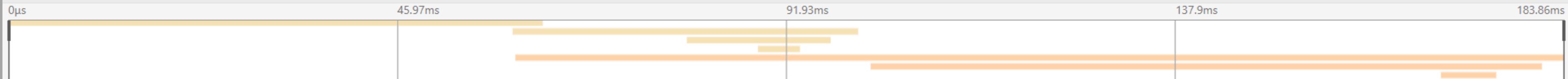




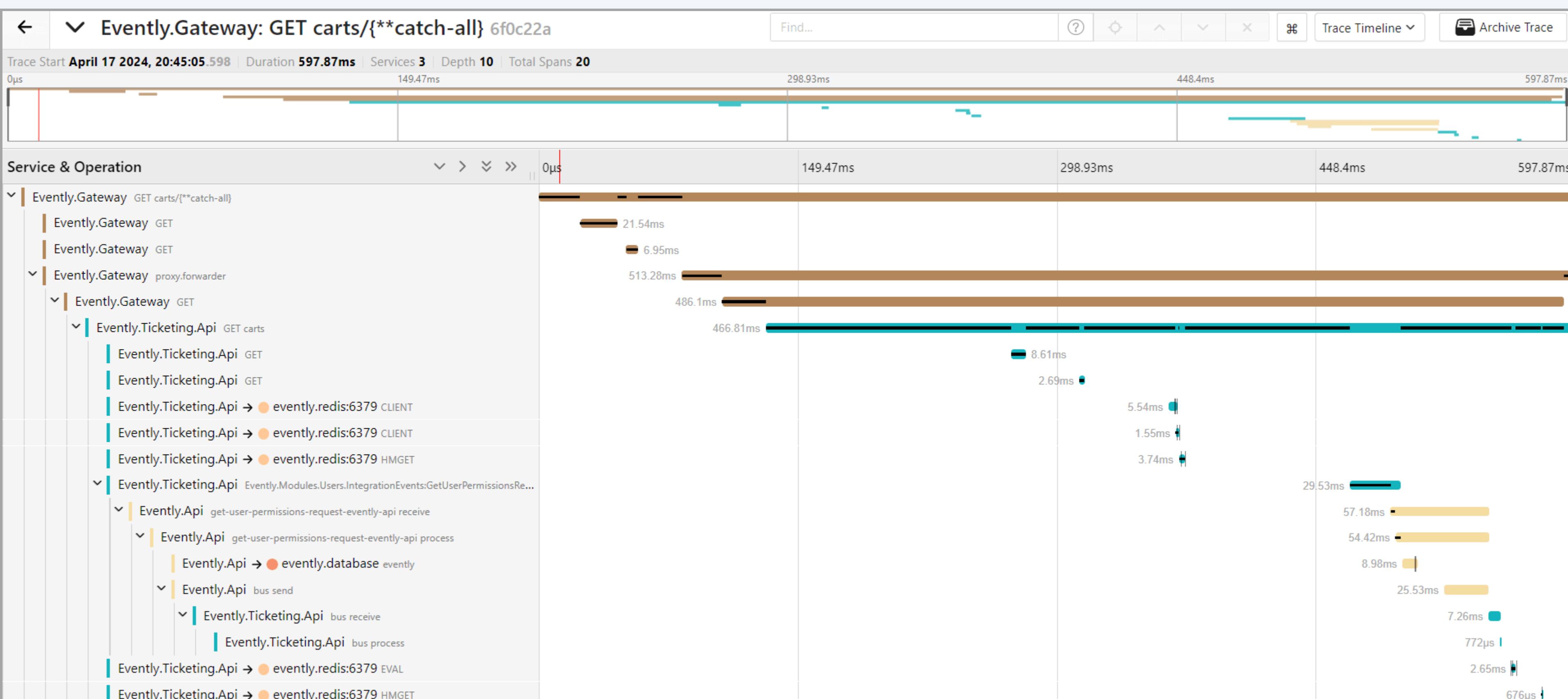
← Evently.Api: Evently.Modules.Users.
IntegrationEvents:
UserRegisteredIntegrationEvent send
8be33a0

Find... ? ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋ ⌓ ⌔ Trace Timeline ⌕ Archive Trace

Trace Start April 17 2024, 20:39:28.707 | Duration 183.86ms | Services 2 | Depth 4 | Total Spans 7



Service & Operation	<	>	<<	>>	0μs	45.97ms	91.93ms	137.9ms	183.86ms
Evently.Api Evently.Modules.Users.IntegrationEvents:UserRegisteredIntegrationEvent send	▼	▶	◀	▶▶	0μs	45.97ms	91.93ms	137.9ms	183.86ms
Evently.Api user-registered-integration-event-evently-api receive	▼	▶							
Evently.Api user-registered-integration-event-evently-api process	▼	▶							
Evently.Api → evently.database evently	▼	▶							
Evently.Ticketing.Api user-registered-integration-event-evently-ticketing-api receive	▼	▶							
Evently.Ticketing.Api user-registered-integration-event-evently-ticketing-api process	▼	▶							
Evently.Ticketing.Api → evently.database evently	▼	▶							





Next:
Distributed Messaging
With RabbitMQ



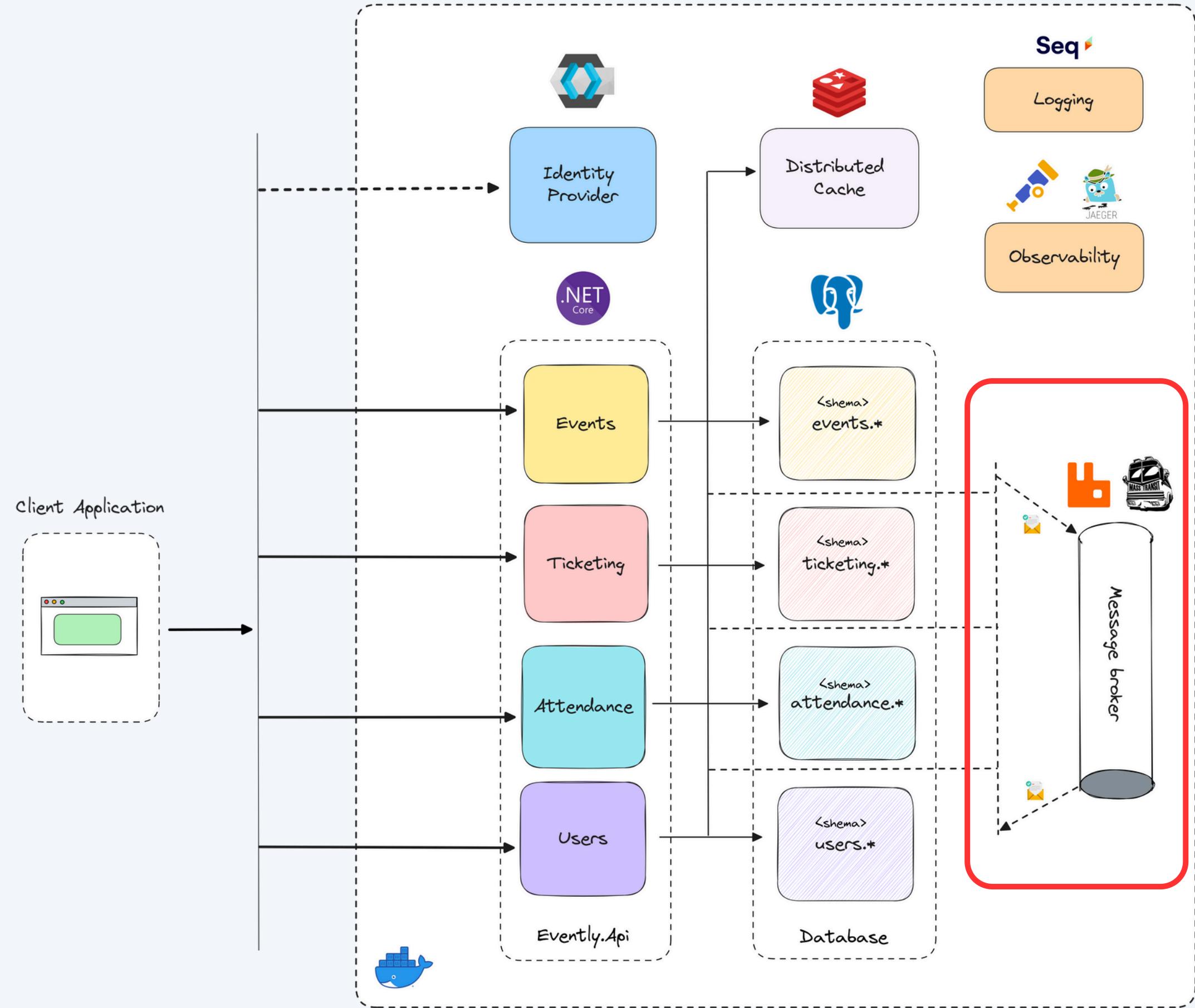
Distributed Messaging With RabbitMQ

RabbitMQ

Reliable and popular message broker.

Supported by MassTransit.







Next:
API Gateways,
Introducing YARP



API Gateways, Introducing YARP

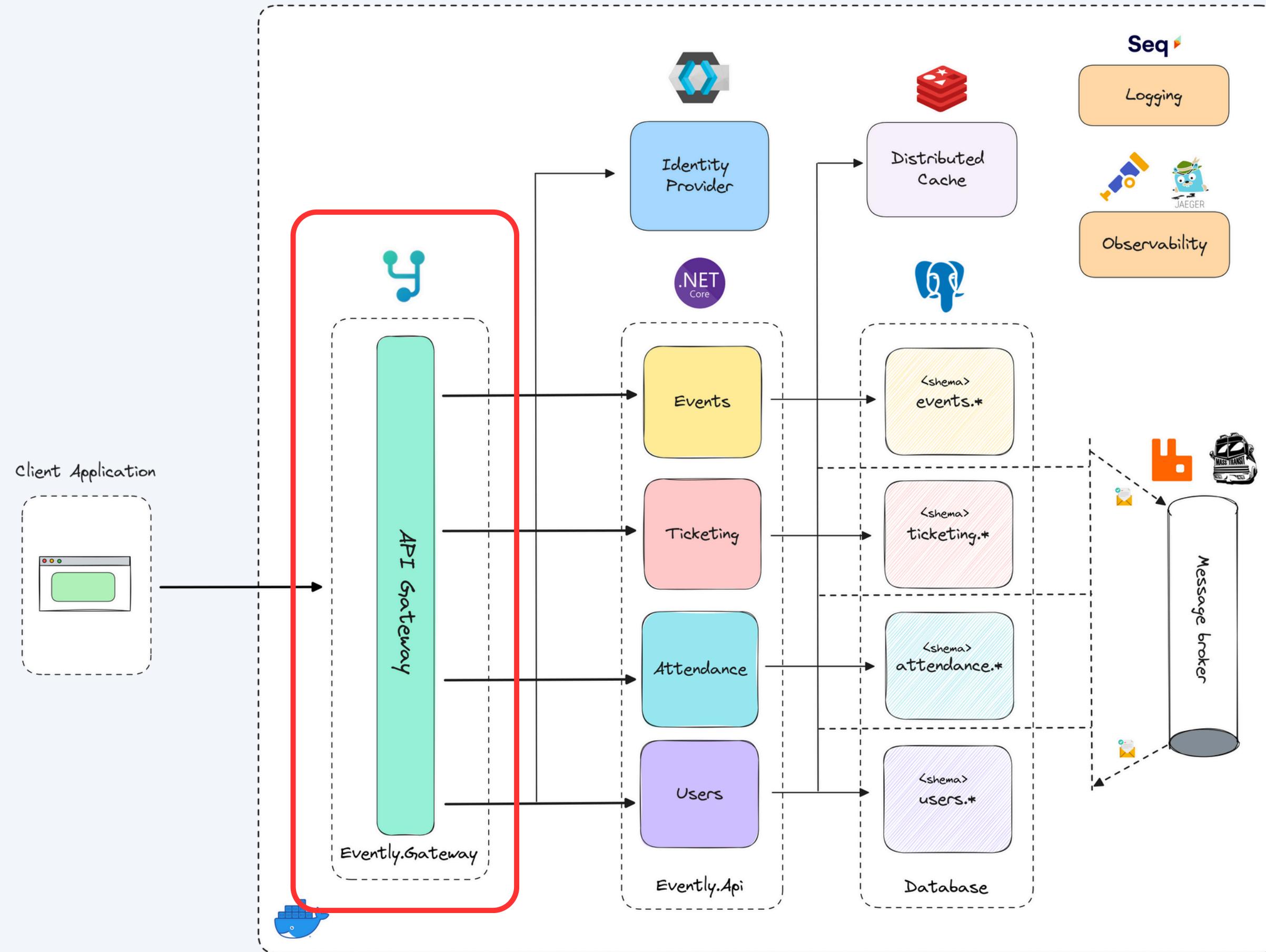
API Gateway

Acts as a reverse proxy for backend services.

Request routing and composition.

Authentication and authorization.

Load balancing, rate limiting, caching.





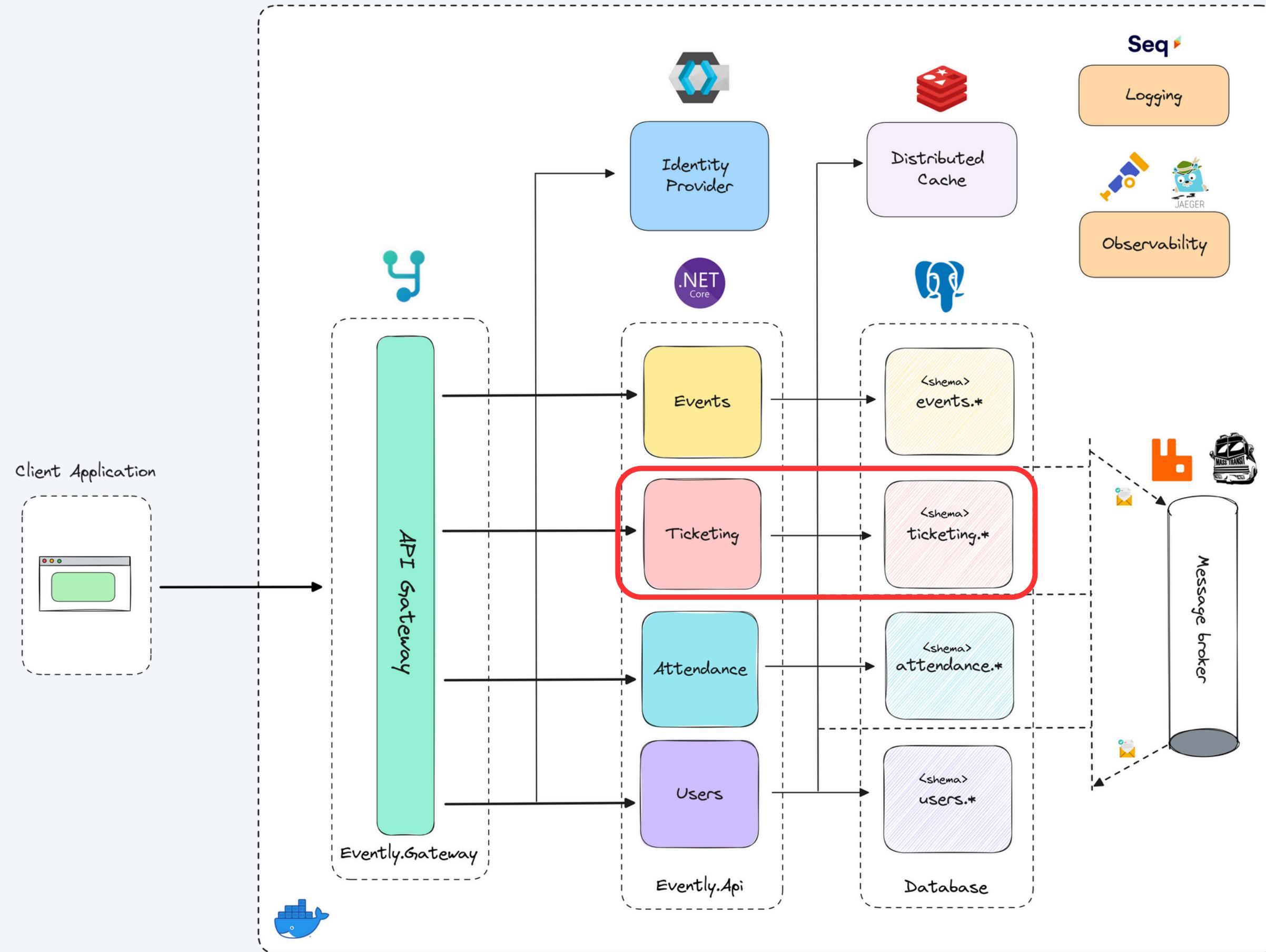
Next:
Extracting Modules to
Microservices

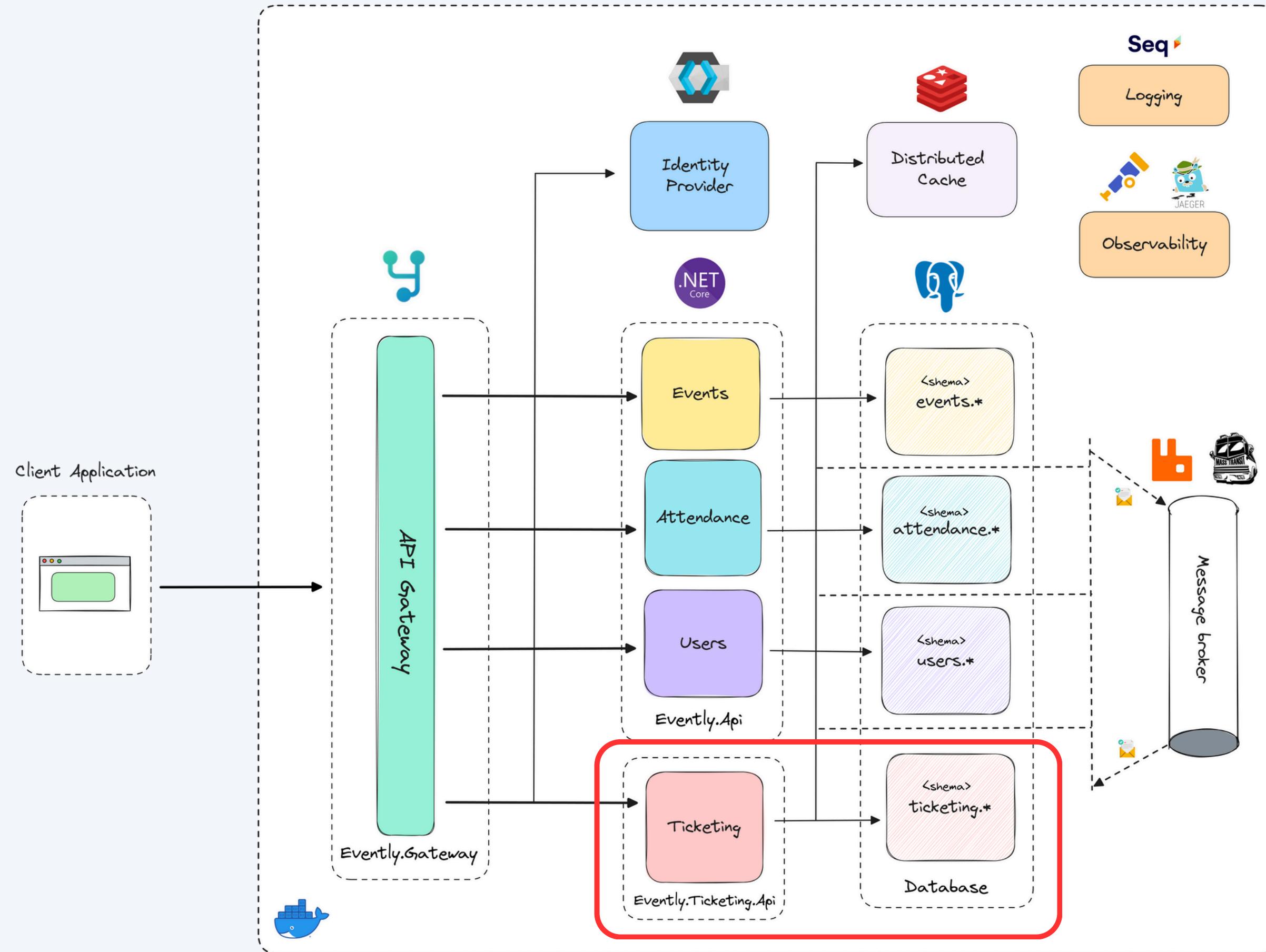


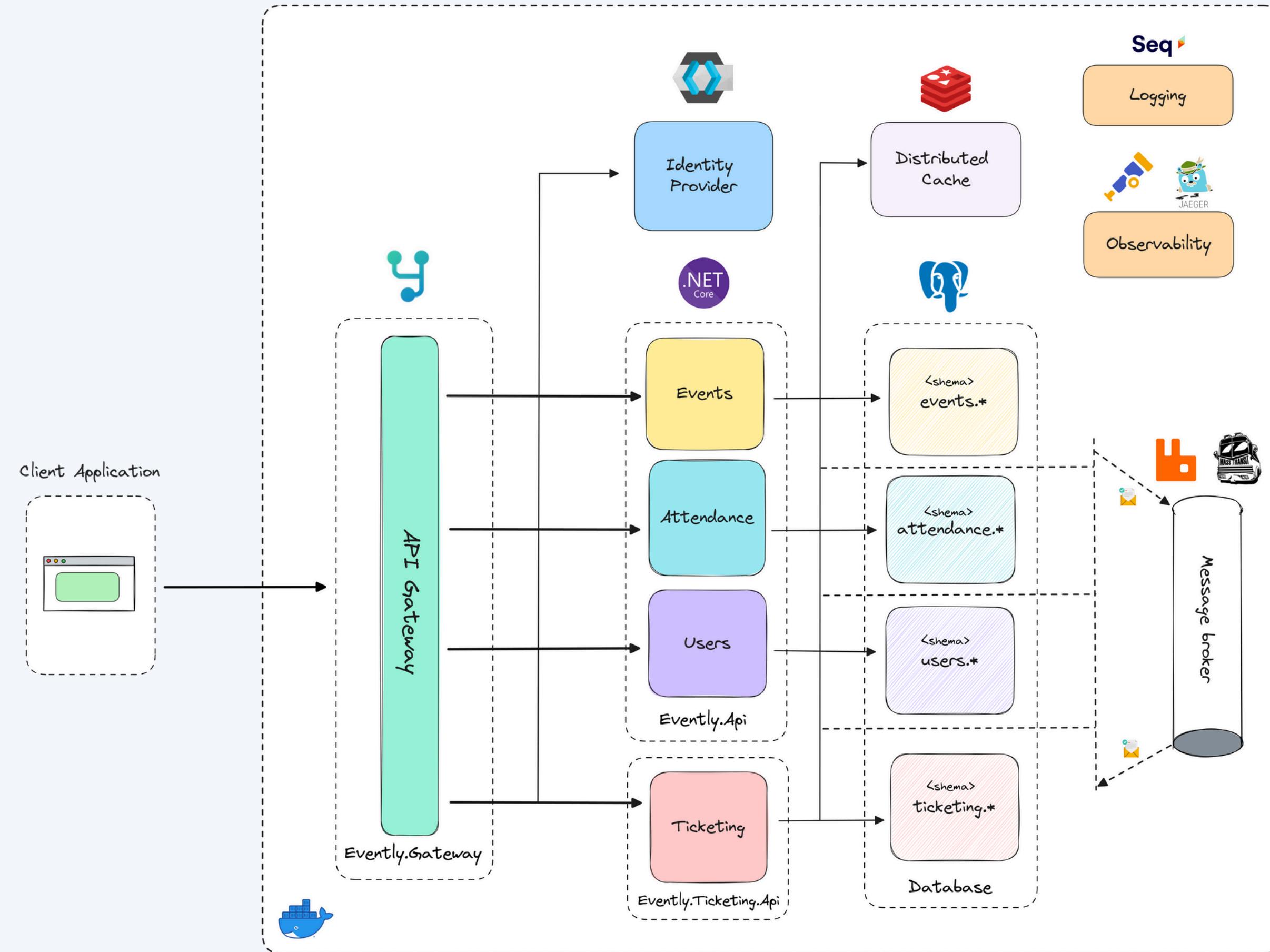
Extracting Modules to Microservices

Extracting Modules

Microservices elevate the logical boundaries in
a modular monolith to physical boundaries.







RECAP

Microservices: Extracting Modules

