Microservicios y Event Sourcing

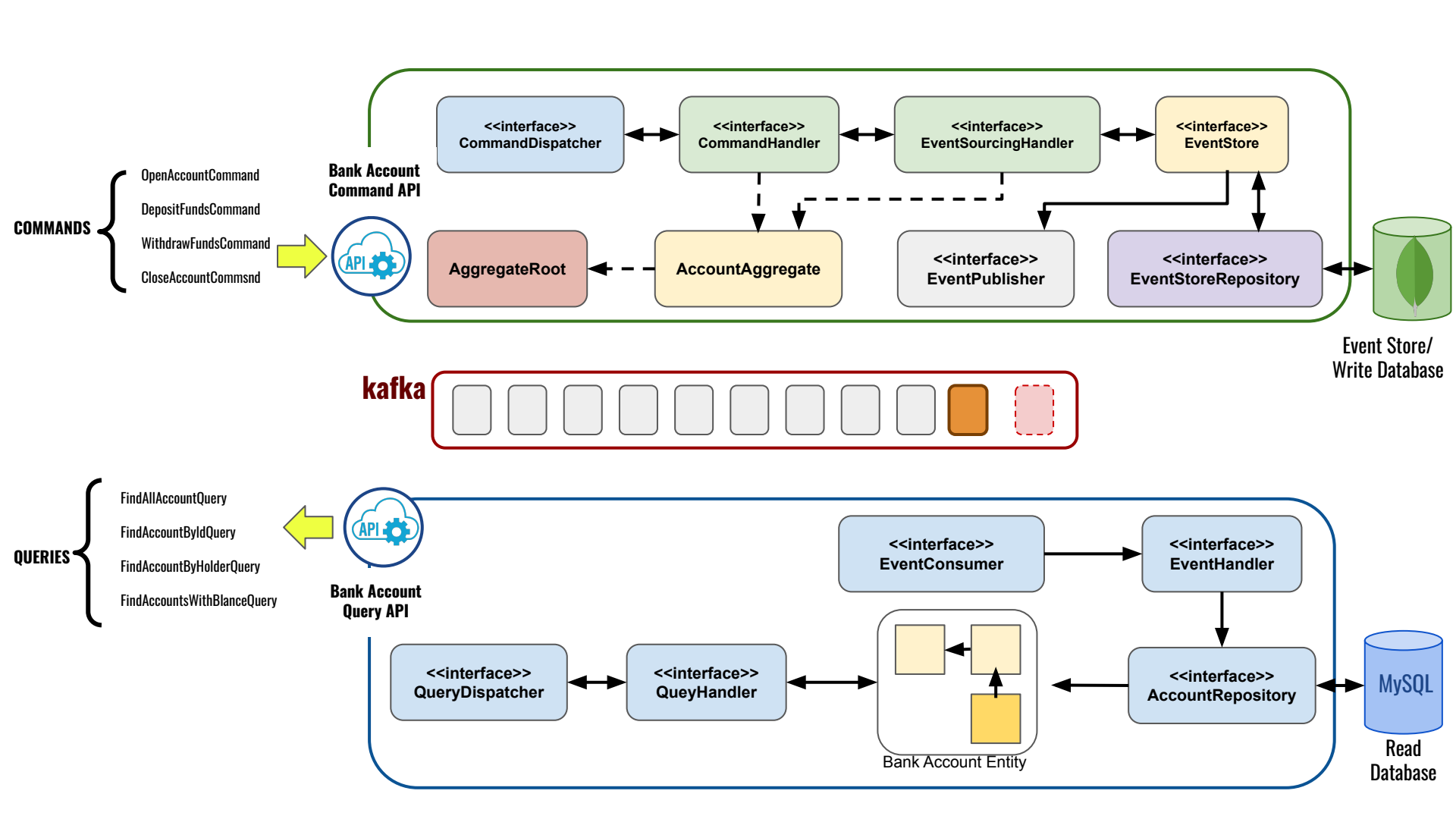
En .NET

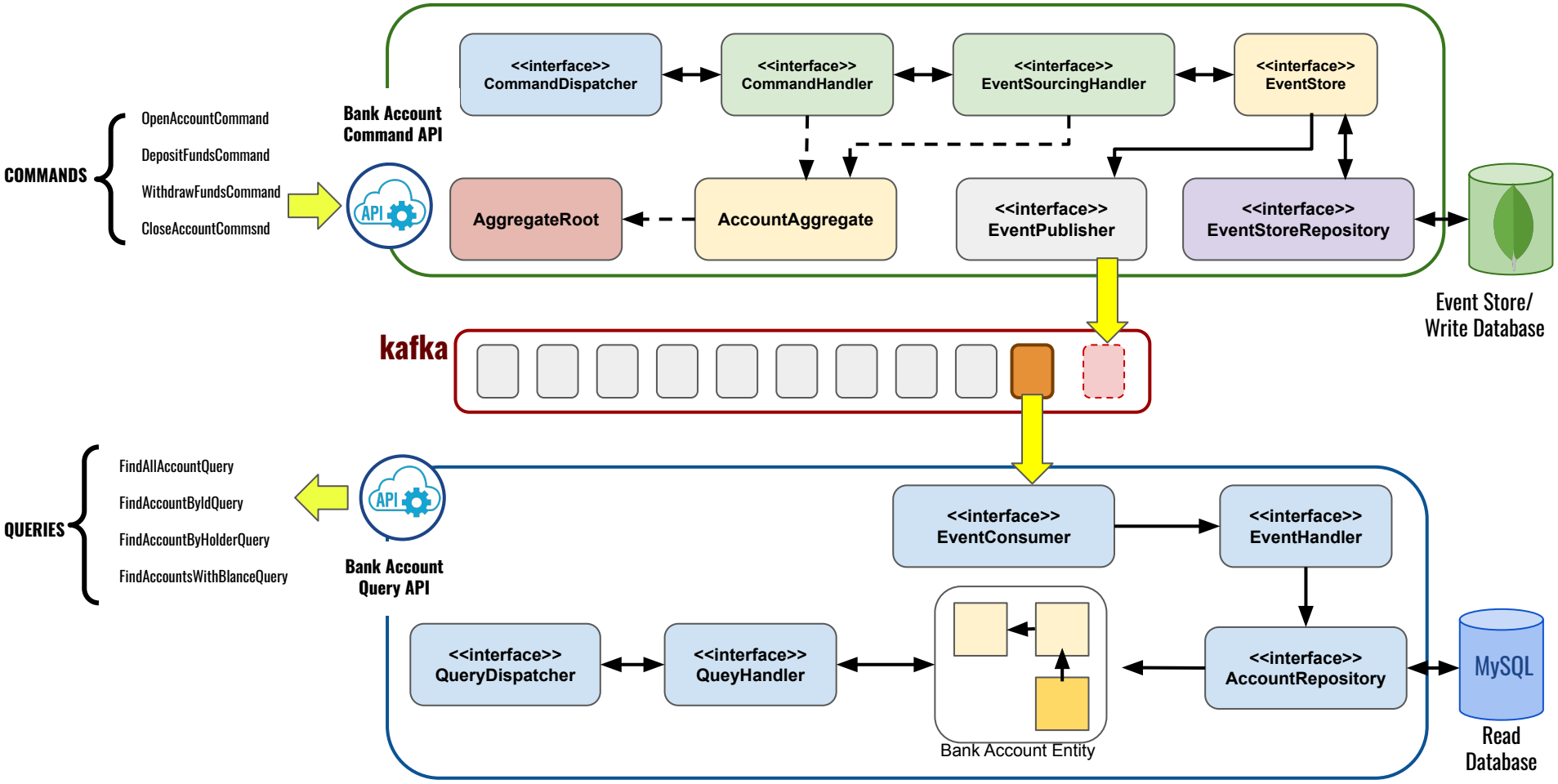
- Manejar comandos y generar eventos.
- Use el patrón de mediador para implementar despachadores de consultas y comandos
- Implementar una base de datos de almacenamiento/escritura de eventos en MongoDB
- Crear una base de datos de lectura en MySQL
- Aplicar control de versiones de eventos
- Implementar control de concurrencia optimista
- Producir eventos para Apache Kafka
- Consumir eventos de Apache Kafka para completar y modificar la base de datos de lectura
- Vuelva a reproducir el almacén de eventos para recrear el estado del agregado
- Inquietudes separadas de lectura y escritura
- Estructure su código utilizando las mejores prácticas de diseño basado en dominios

Definición de un Microservicios

Los microservicios son aplicaciones o servicios pequeños, débilmente acoplados, que pueden fallar independientemente unos de otros.
Cuando falla un microservicio, solo un único proceso o función en el sistema debería dejar de estar disponible, mientras que el resto del sistema no se ve afectado.

- Los microservicios no deben compartir código ni datos
- Evite el acoplamiento innecesario entre servicios y componentes de software
- La independencia y la autonomía son más importantes que la reutilización del código
- Cada microservicio debe ser responsable de una sola función del sistema o proceso
- Los microservicios no deben comunicarse directamente entre sí, deben utilizar un bus de eventos/mensajes para comunicarse entre sí.


COMMANDS
OpenAccountCommand
DepositFundsCommand
WithdrawFundsCommand
CloseAccountCommsnd
Bank Account Command API
API
<<interface>> CommandDispatcher
<<interface>> CommandHandler
<<interface>> EventSourcingHandler
<<interface>> EventStore
AggregateRoot
AccountAggregate
<<interface>> EventPublisher
<<interface>> EventStoreRepository
Event Store/ Write Database
kafka
QUERIES
FindAllAccountQuery
FindAccountByIdQuery
FindAccountByHolderQuery
FindAccountsWithBlanceQuery
Bank Account Query API
API
<<interface>> EventConsumer
<<interface>> EventHandler
<<interface>> QueryDispatcher
<<interface>> QueyHandler
Bank Account Entity
<<interface>> AccountRepository
MySQL
Read Database


COMMANDS
OpenAccountCommand
DepositFundsCommand
WithdrawFundsCommand
CloseAccountCommsnd
Bank Account Command API
API
<<interface>> CommandDispatcher
<<interface>> CommandHandler
<<interface>> EventSourcingHandler
<<interface>> EventStore
AggregateRoot
AccountAggregate
<<interface>> EventPublisher
<<interface>> EventStoreRepository
Event Store/ Write Database
kafka
QUERIES
FindAllAccountQuery
FindAccountByIdQuery
FindAccountByHolderQuery
FindAccountsWithBlanceQuery
Bank Account Query API
API
<<interface>> EventConsumer
<<interface>> EventHandler
<<interface>> QueryDispatcher
<<interface>> QueyHandler
Bank Account Entity
<<interface>> AccountRepository
MySQL
Read Database

¿Qué es Apache Kafka?

Apache Kafka es una plataforma de transmisión de eventos distribuidos de código abierto que permite el desarrollo de aplicaciones en tiempo real basadas en eventos.

Kafka fue desarrollado por LinkdIn en 2011 como un intermediario de mensajes de alto rendimiento para su propio uso. Luego fue de código abierto y donado a la Fundación Apache.
Hoy en día, Kafka se ha convertido en la plataforma de transmisión más utilizada y es capaz de ingerir y procesar billones de registros por día sin ningún retraso notable en el rendimiento.

CQRS & Event Sourcing with Kafka

¿Qué es un comando?
Un comando es una combinación de intenciones expresadas.
En otras palabras, describe algo que quieres que se haga.
También contiene la información requerida para emprender acciones basadas en esa intención. Los comandos se nombran con un verbo en modo imperativo.

Que es un evento?

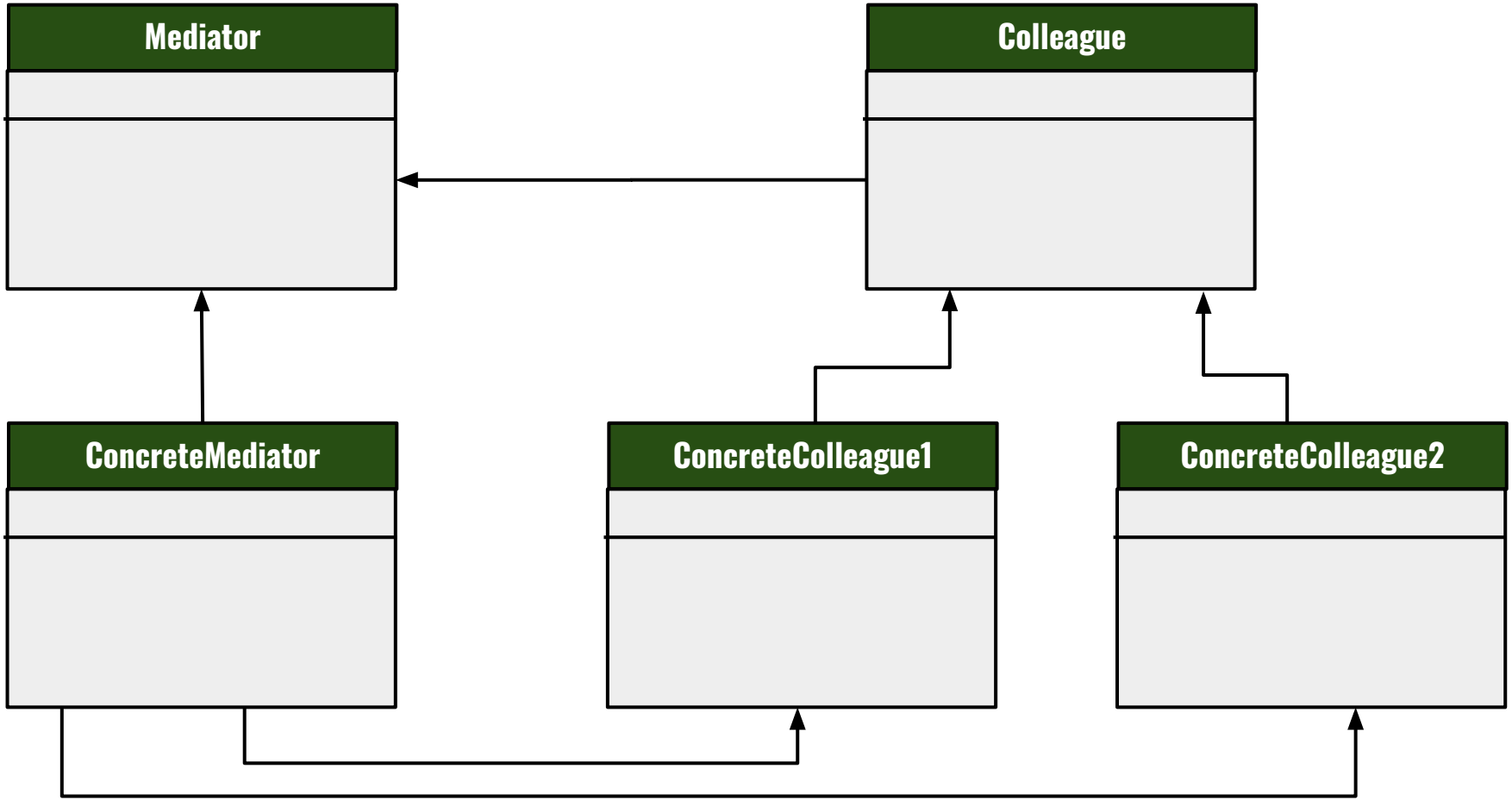
Los eventos son objetos que describen algo que ha ocurrido en la aplicación. Una fuente típica de eventos es el agregado. Cuando algo importante ha ocurrido dentro del agregado, generará un evento

Mediator Pattern

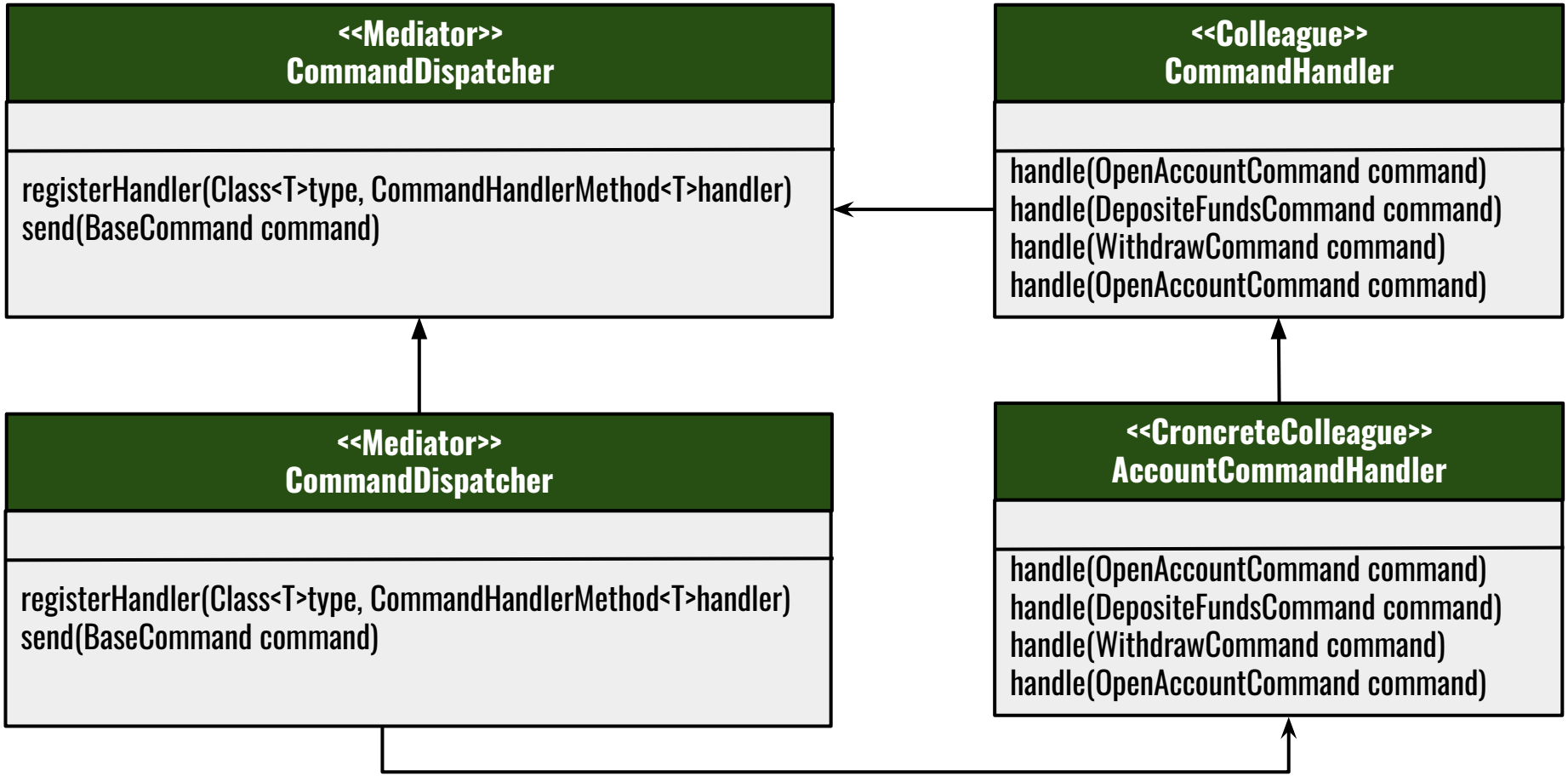
Mediator Pattern

- Patrón de diseño conductual
- Promueve el acoplamiento flexible al evitar que los objetos se refieran entre sí explícitamente
- Simplifica la comunicación entre objetos mediante la introducción de un solo objeto conocido como el mediador que gestiona la distribución de mensajes entre otros objetos.

Mediator Pattern



Command Dispatching

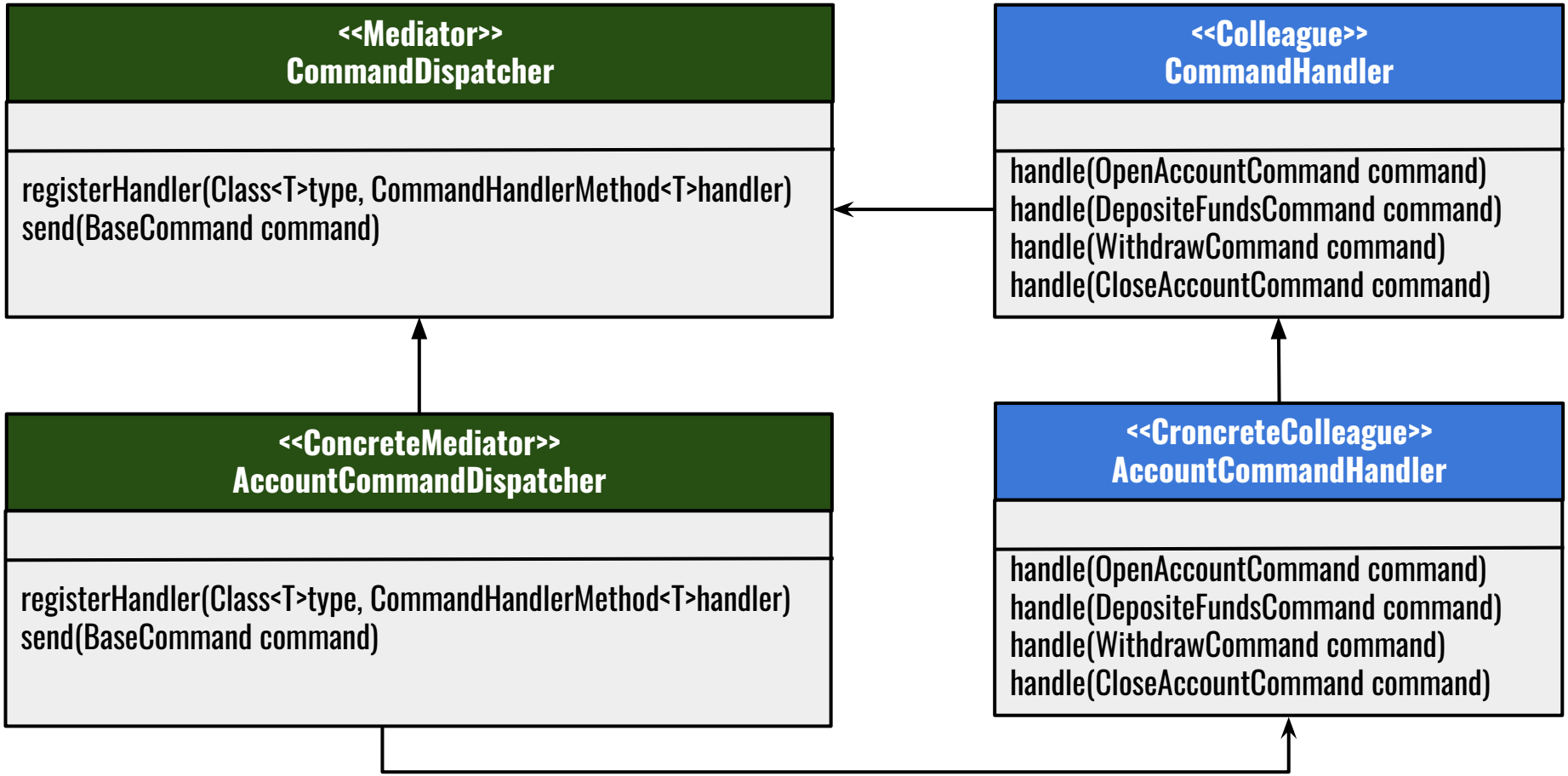


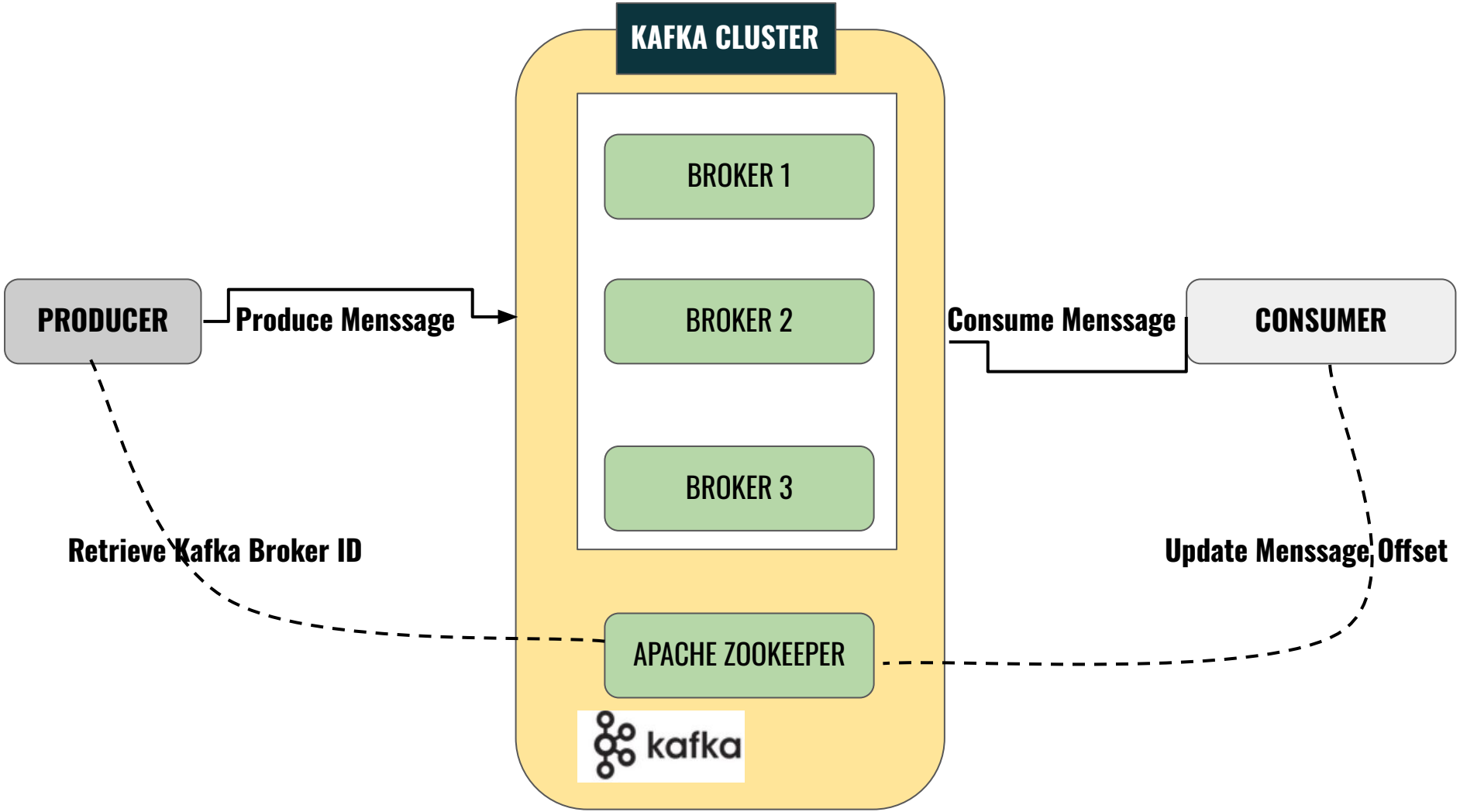
Tienda de eventos: consideraciones clave

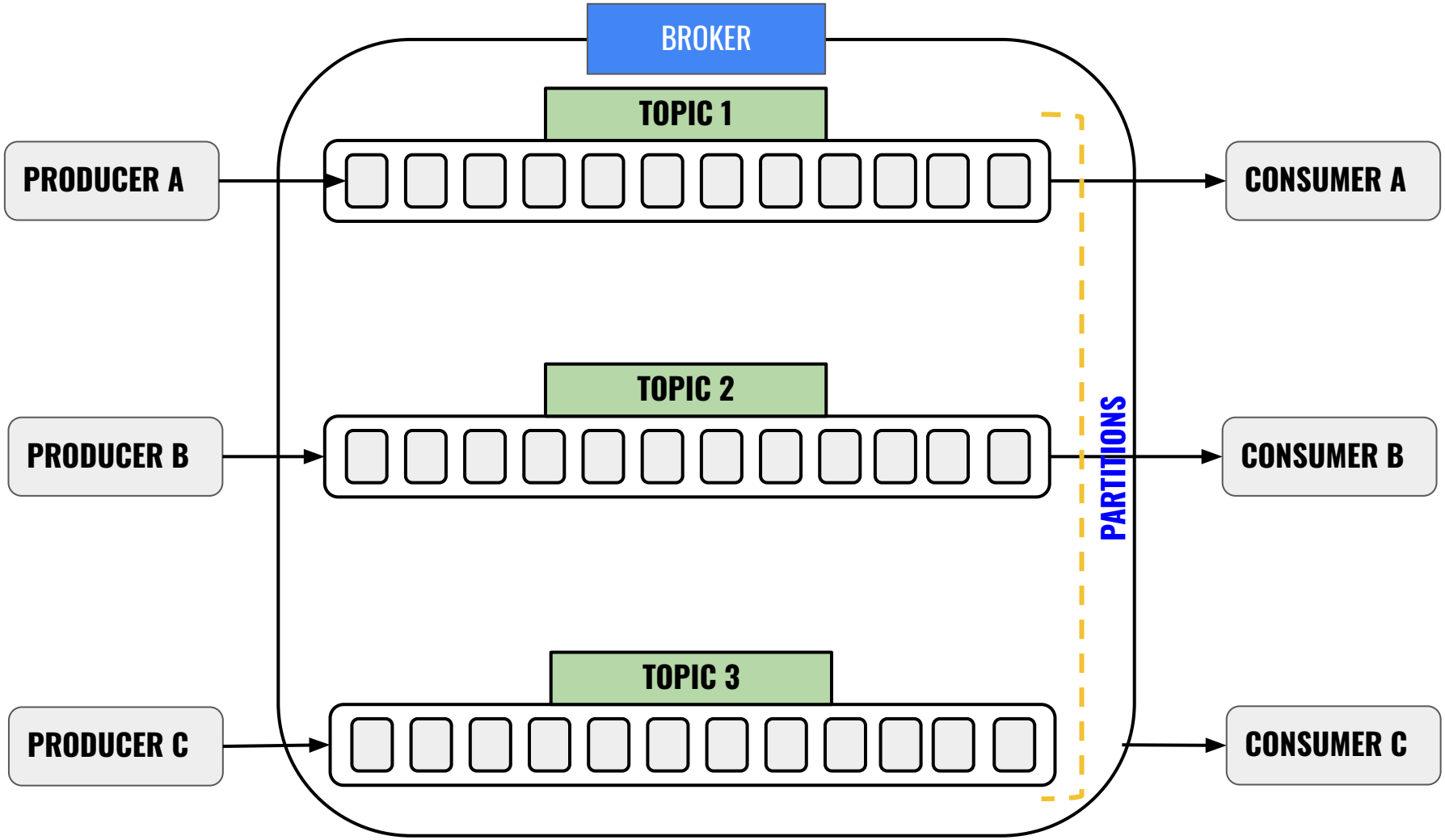
- Una tienda de eventos debe ser una tienda de solo anexar, no se deben permitir operaciones de actualización o eliminación
- Cada evento que se guarda debe representar la versión o el estado de un agregado en un momento dado.
- Los eventos deben almacenarse en orden cronológico y los nuevos eventos deben agregarse al evento anterior
- El estado del agregado debe poder recrearse reproduciendo el almacén de eventos
- Implementar control de concurrencia optimista

Mediator Pattern (Continuación)

Command Dispatching




KAFKA CLUSTER
BROKER 1
BROKER 2
BROKER 3
PRODUCER
Produce Menssage
Consume Menssage
CONSUMER
Retrieve Kafka Broker ID
Update Menssage Offset
APACHE ZOOKEEPER
kafka


BROKER
TOPIC 1
TOPIC 2
TOPIC 3
PRODUCER A
PRODUCER B
PRODUCER C
CONSUMER A
CONSUMER B
CONSUMER C
PARTITIONS

Domain Driven Design (DDD)

- Creado en 2003
- Estructurar y modelar
- Trabaja con el Core del business logic
- Problemas = Dominio
- Bounded Context

Bounded Context

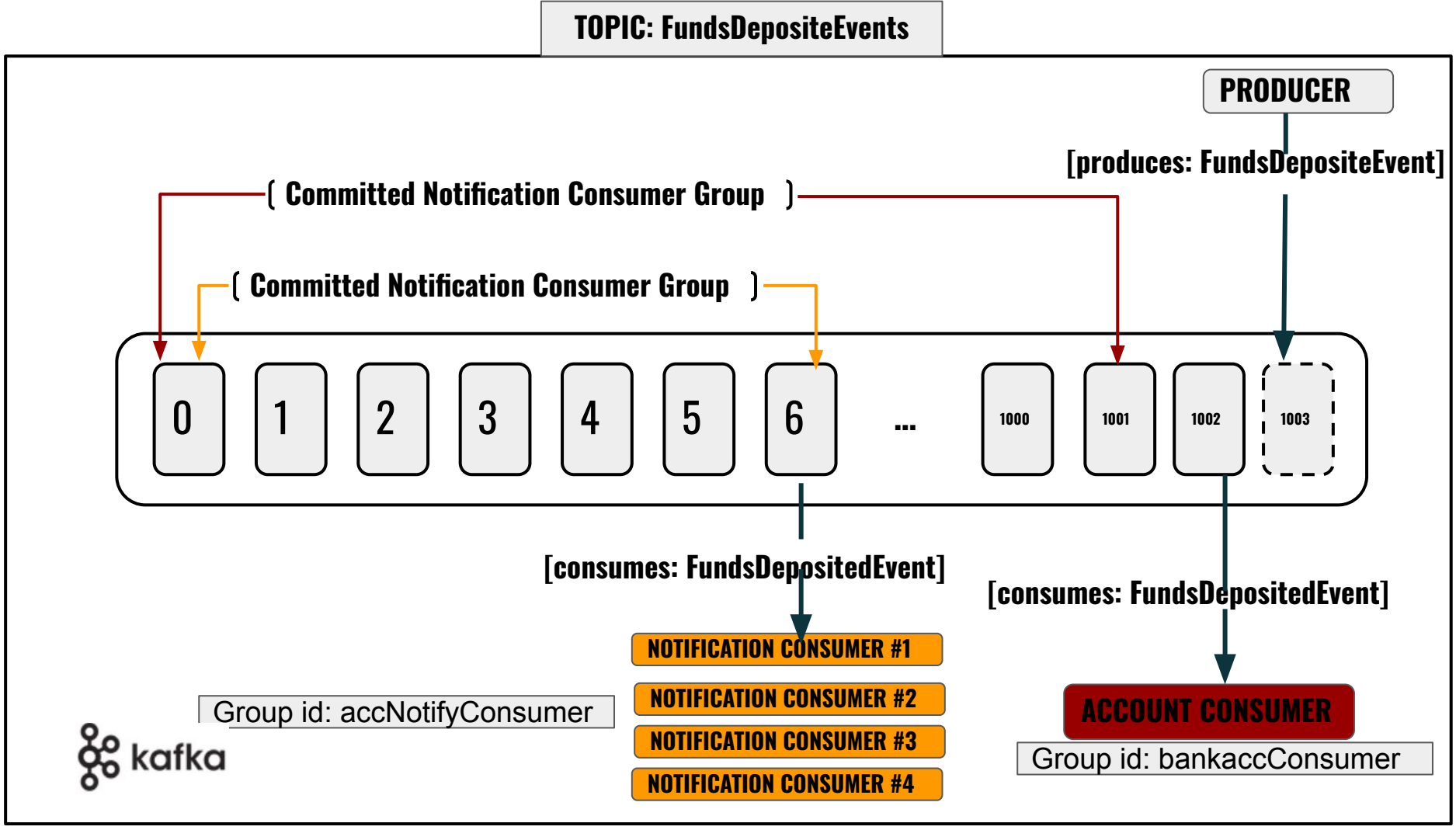
- Es un área de problema independiente.
- Traza limites entre problemas
- Cada Bounded Context tendra una solucion -> Microservice

BankAccount Entity

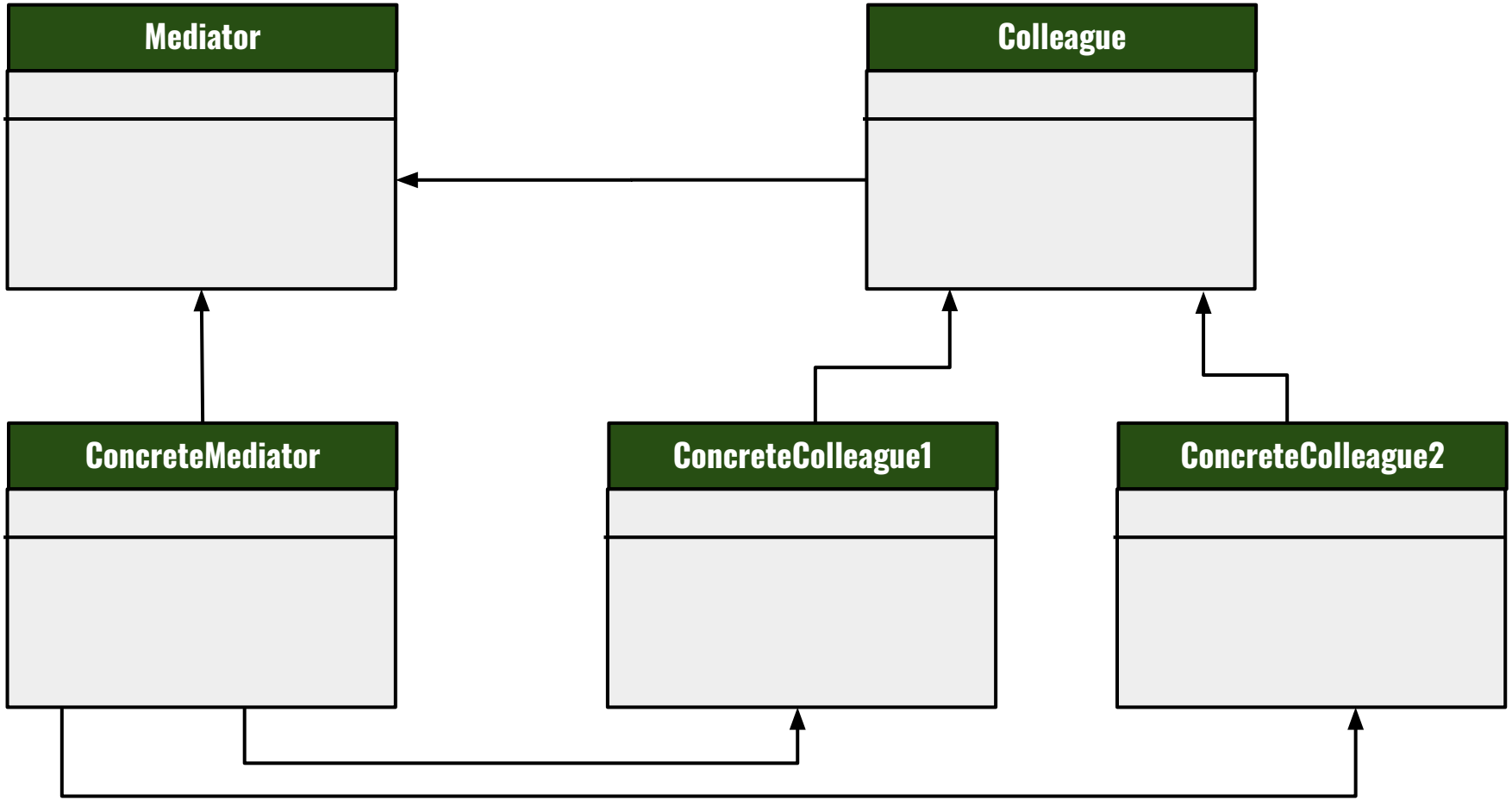
```java
@Entity
public class BankAccount {
    @Id
    private String id;
    private String accountHolder;
    private Date creationDate;
    private AccountType accountType;
    private double balance;
}
```

bank_account	
PK	id
	accountHolder
	creationDate
	accountType
	balance

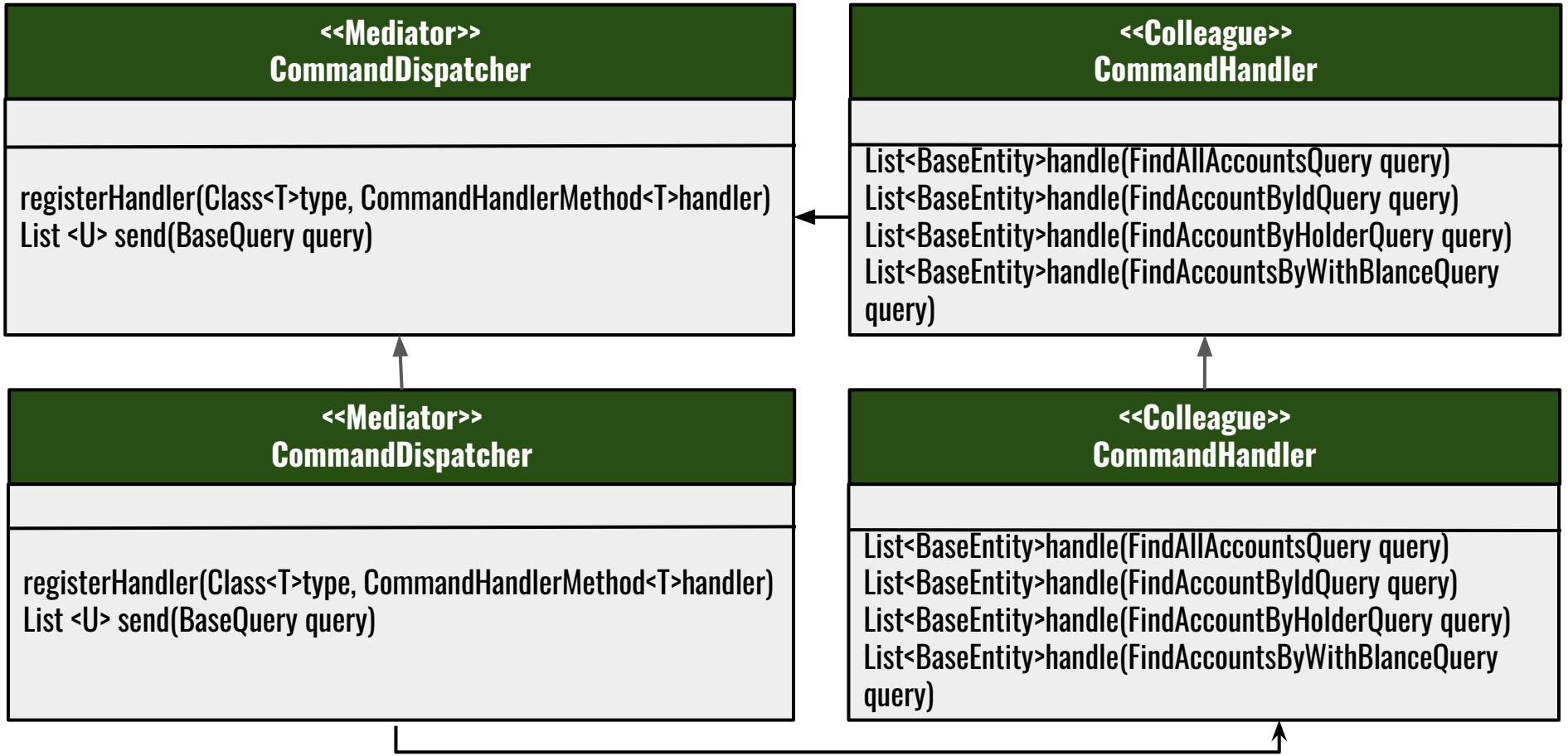
id	accountHolder	creationDate	accountType	balance
0faca428-01ab-4eb6-af8b-986bd3dd7268	John Doe	2021-05-27 21:17:31.467000	1	1 500 000.00
3e246fbc-c7dd-47c3-beb5-1e2463fa2461	Jane Doe	2021-06-01 11:47:01.200000	2	1 000 000.00


TOPIC: FundsDepositeEvents
PRODUCER
[produces: FundsDepositeEvent]
{ Committed Notification Consumer Group }
{ Committed Notification Consumer Group }
0
1
2
3
4
5
6
...
1000
1001
1002
1003
[consumes: FundsDepositedEvent]
[consumes: FundsDepositedEvent]
NOTIFICATION CONSUMER #1
NOTIFICATION CONSUMER #2
NOTIFICATION CONSUMER #3
NOTIFICATION CONSUMER #4
Group id: accNotifyConsumer
ACCOUNT CONSUMER
Group id: bankaccConsumer
kafka

Mediator Pattern



Query Dispatching



Query Dispatching