The Big Friendly Monolith

AxonFramework

About me

Frans van Buul

frans.vanbuul@axoniq.io

+31 6 5068 2984

https://www.linkedin.com/in/frans-van-buul-8030743/

What is AxonIQ?

- AxonIQ was founded in July 2017 as a commercial company with a focus on the open source Axon Framework.

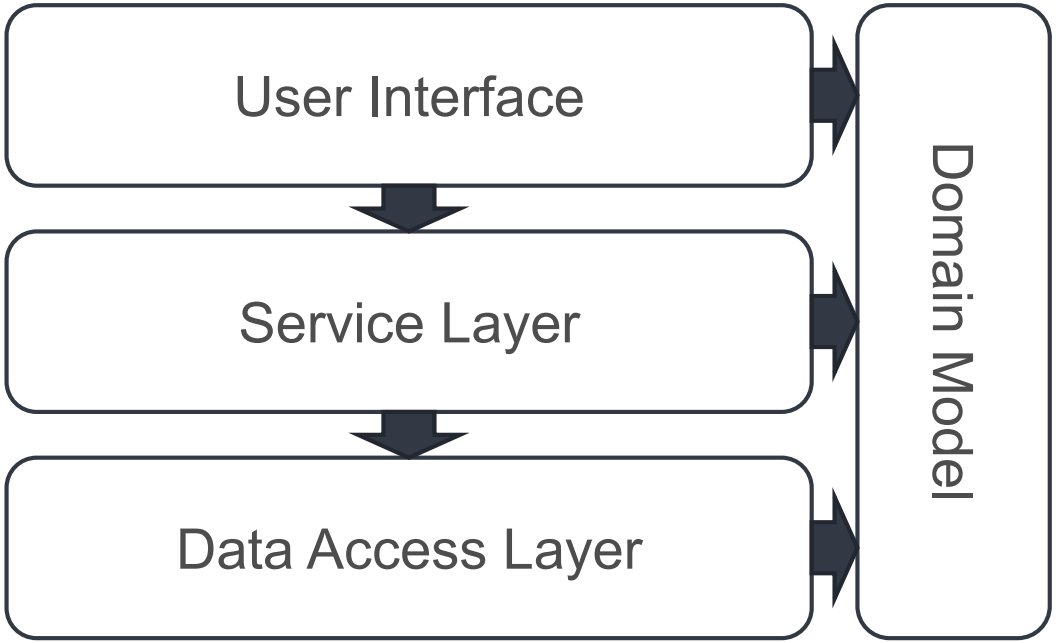
- AxonIQ delivers:
  - Support, training, consultancy for Axon Framework
  - Commercial software products that work in conjunction with Axon Framework.

- Based in Amsterdam, 11 people as of November 2017

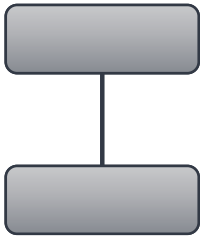
Agenda

- Microservices in the context of layered architecture and DDD
- CQRS, event sourcing and Axon Framework concepts
- Axon Framework code examples

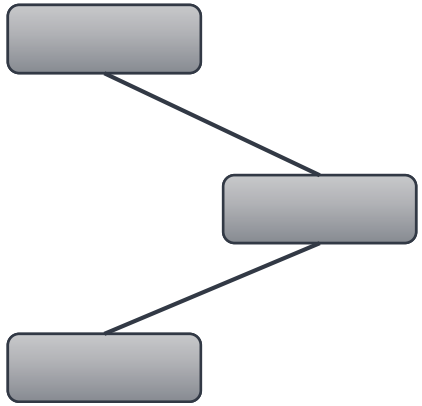
Layered Architecture



Evolution of a Domain Model

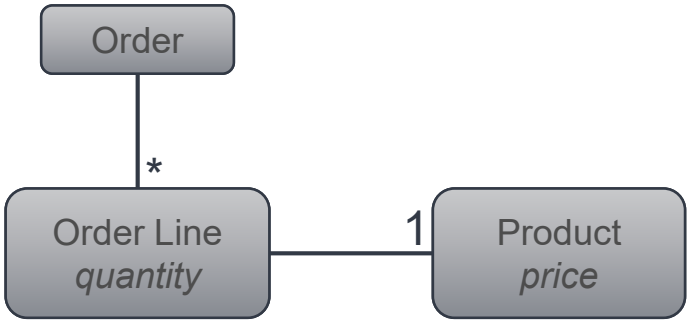


Evolution of a Domain Model



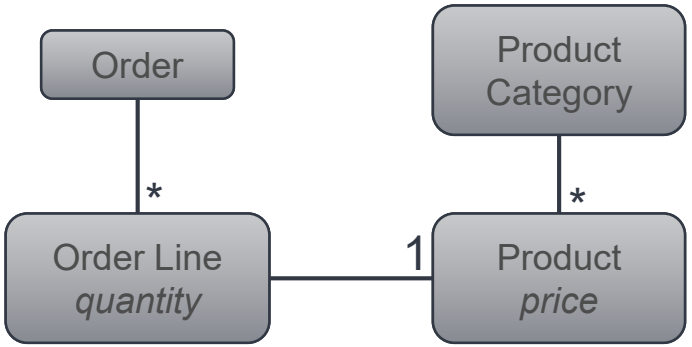
Example

Customer Order



Example

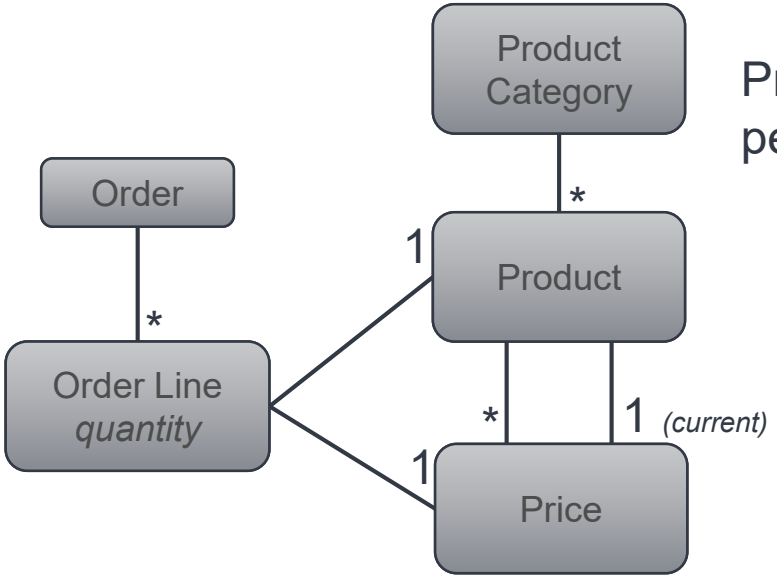
Customer Order perspective



Product catalogue perspective

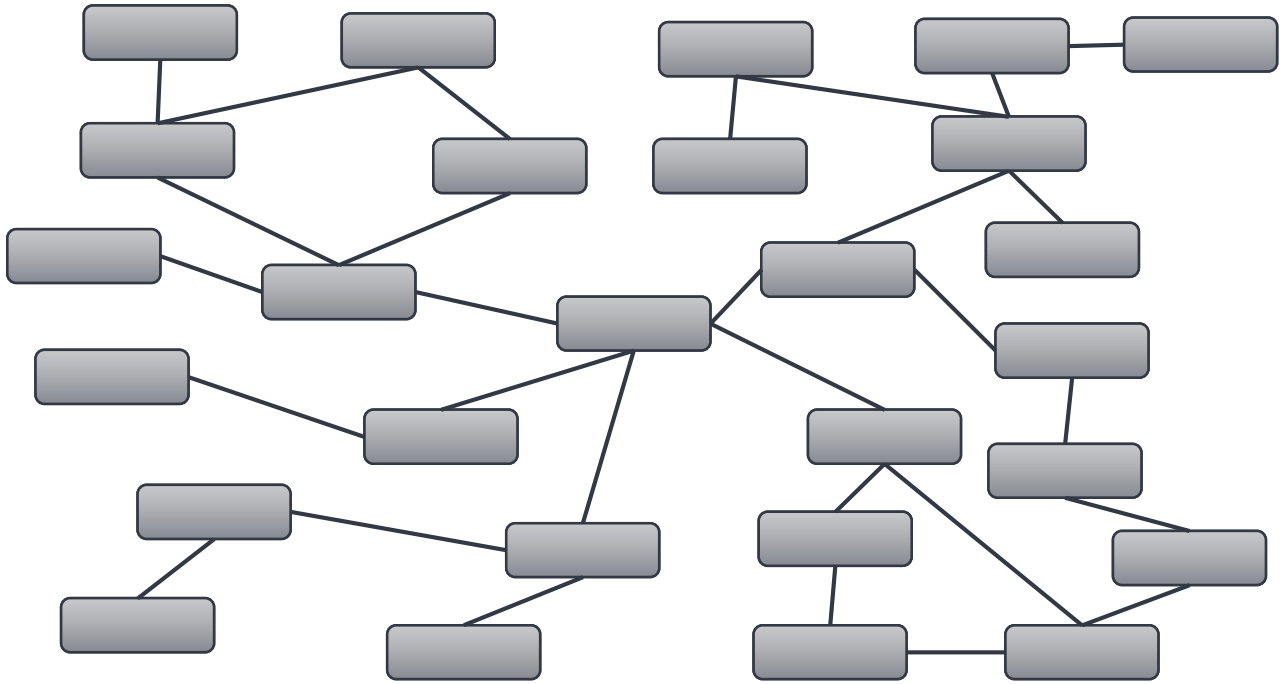
Example

Customer Order perspective



Product catalogue perspective

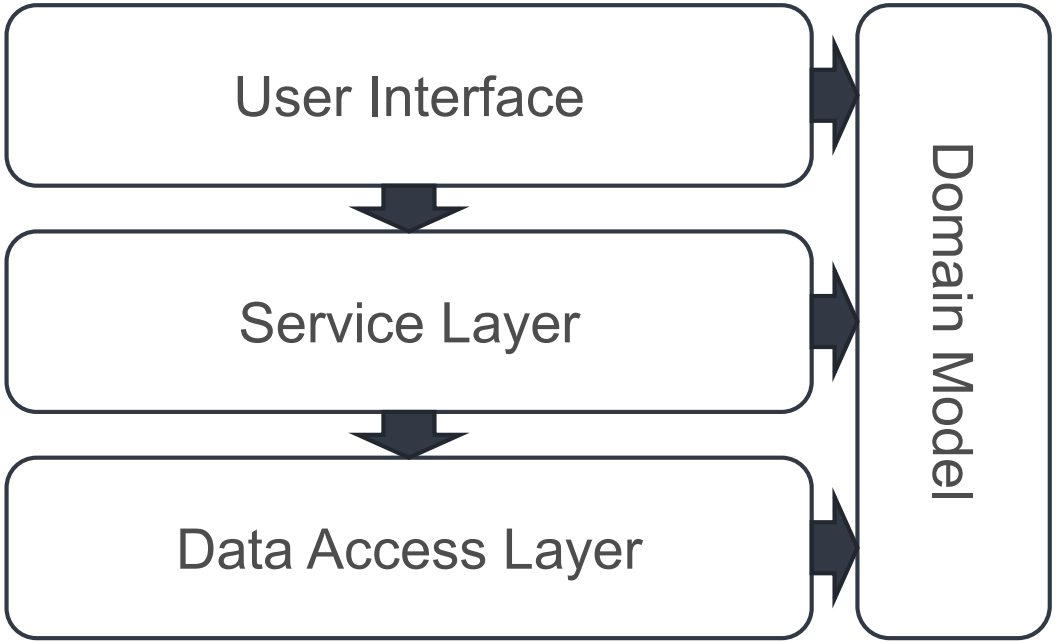
Evolution of a Domain Model



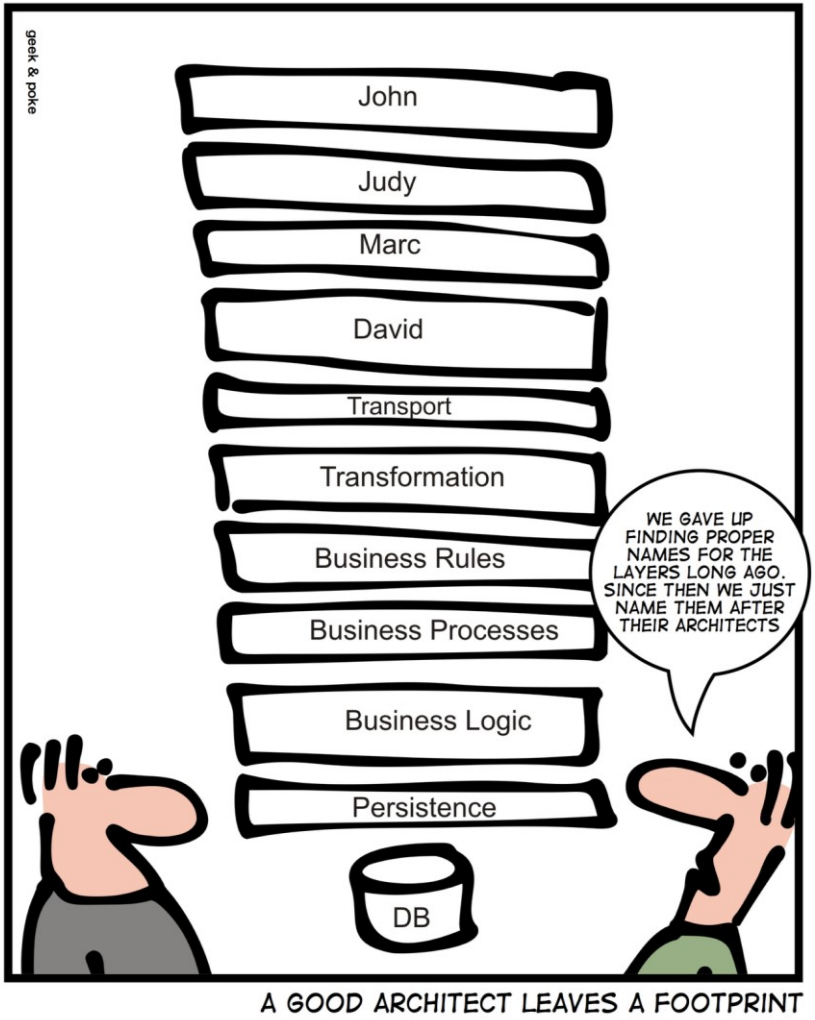
Big Ball of “Mud”

Source: http://www.sabisabi.com/images/DungBeetle-on-dung.JPG

This won't help!

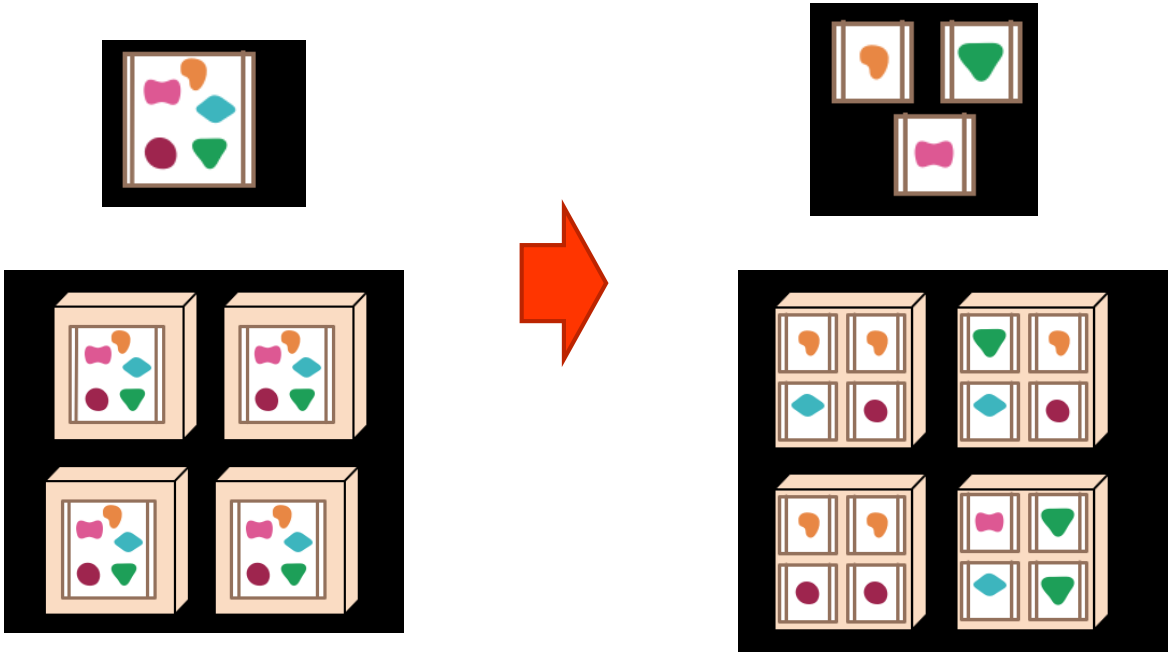


“We gave up finding proper names for the layers long ago. Since then we just name them after their architects.”



Source:
http://geek-and-poke.com/geekandpoke/2013/7/13/foodprints

Microservice Architecture



Source: http://martinfowler.com/articles/microservices.html

Microservices vs Monoliths

Monoliths

Almost all the successful microservice stories have started with a monolith that got too big and was broken up

Microservices system

Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble.

Martin Fowler

Source: http://martinfowler.com/bliki/MonolithFirst.html

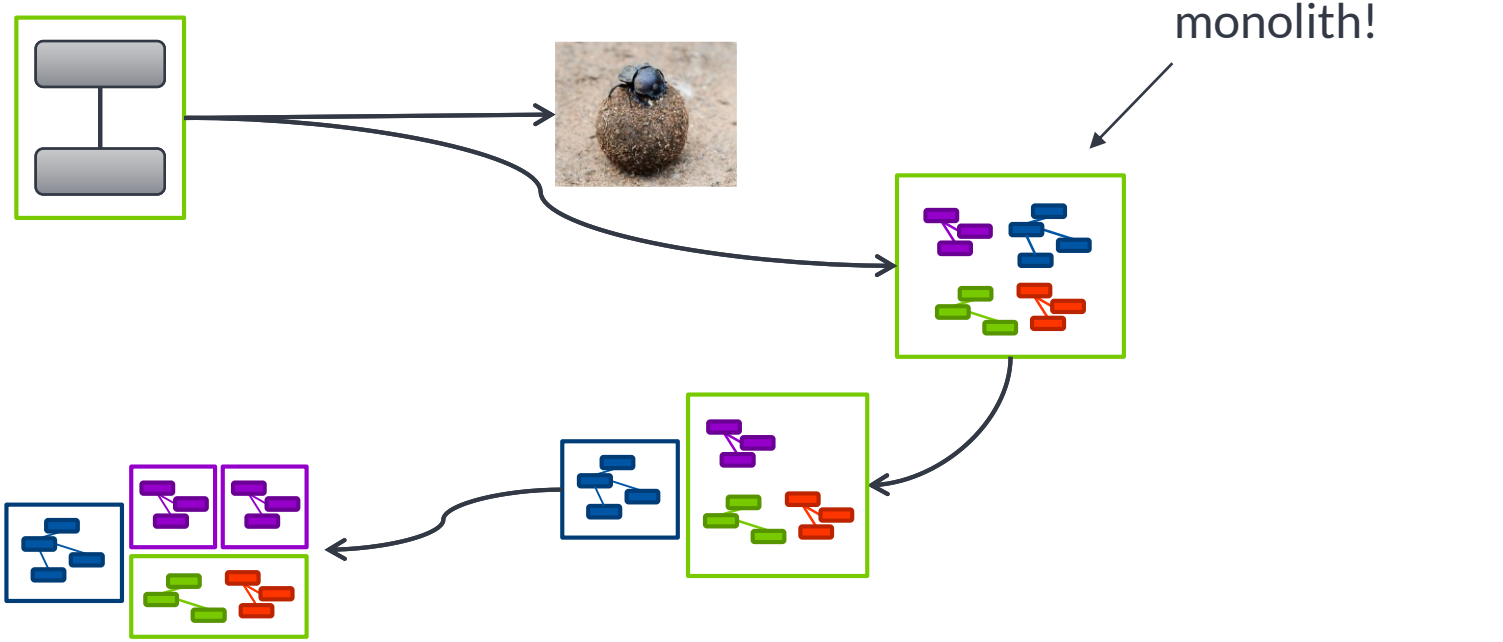
Microservices vs Monoliths

Going from monoliths to microservices

It's really difficult to break up our legacy monolith into microservices!

Pretty much everybody these days

Microservice journey



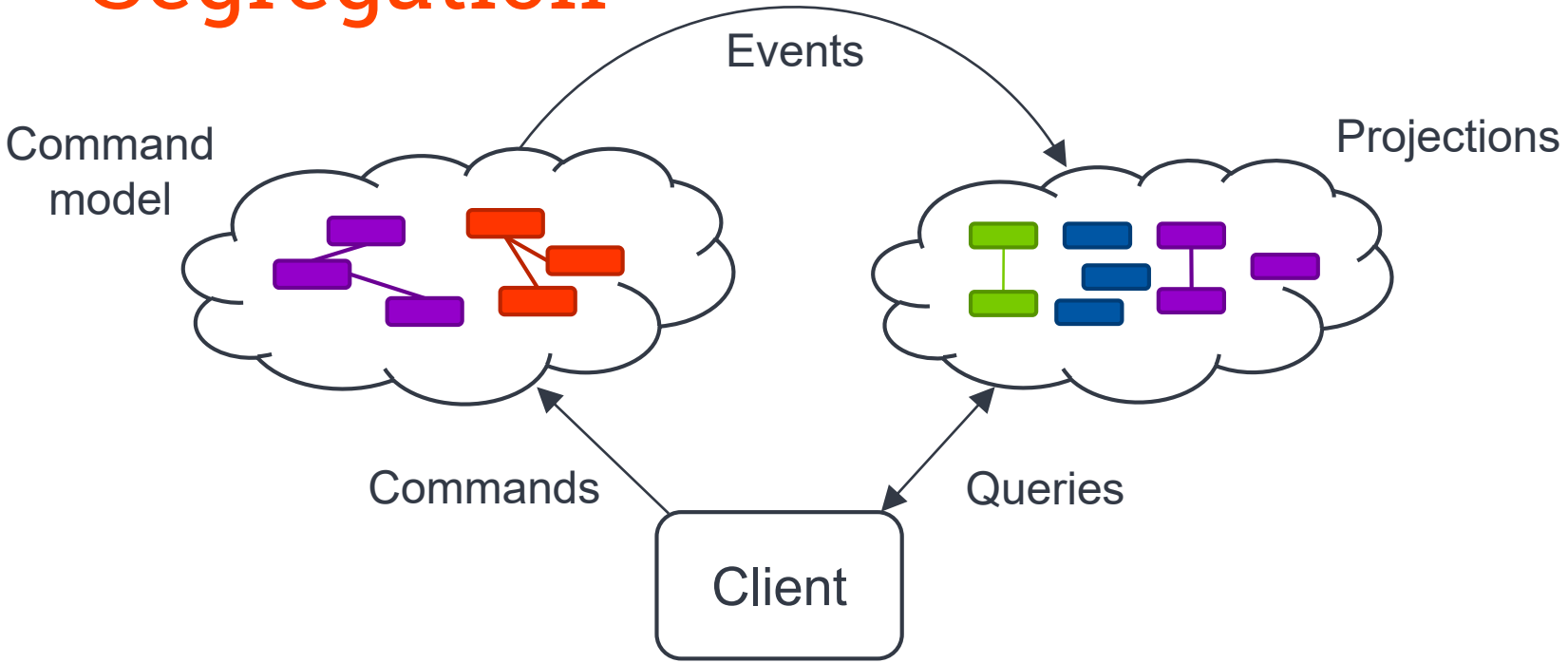
Axon Framework

- “CQRS Framework” for Java
  - Open source (Apache 2 License)
- Simplify CQRS based applications
  - Building blocks common in CQRS-based architectures
- More information: http://www.axonframework.org

Core Principles

- Message oriented
  - Events
  - Commands
  - Queries
- Location transparency
  - Separate infrastructure from business logic
- Customizable
  - Configure to match your infrastructure, not vice versa

Command Query Responsibility Segregation



Source: https://giphy.com

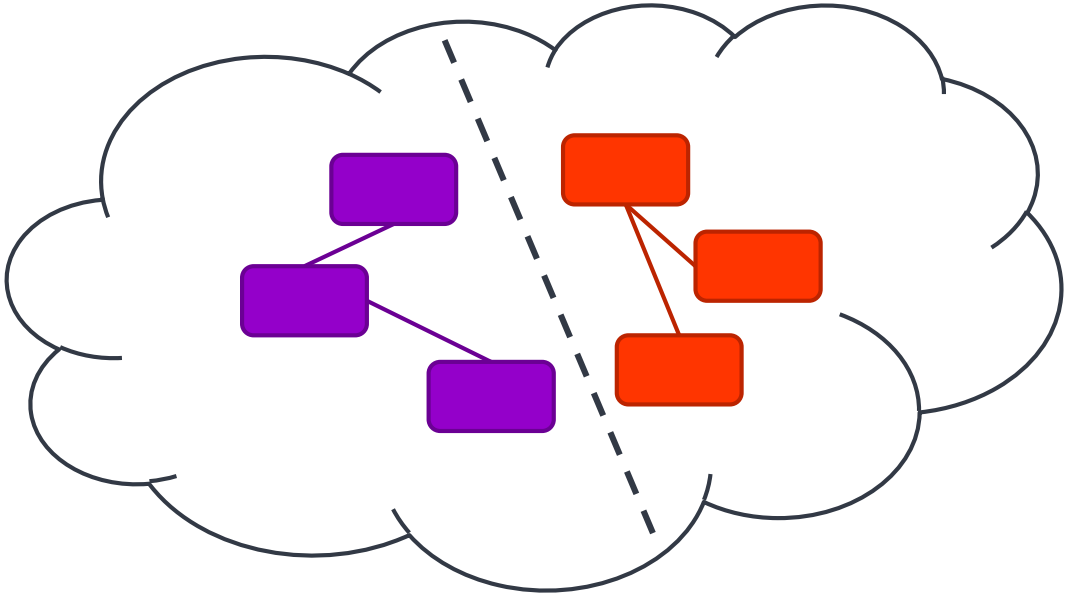
Why?

- Throughput characteristics
- Mixing technologies (SQL, NoSQL)
- Query complexity
- Event sourcing

- **Simplification on the long run**

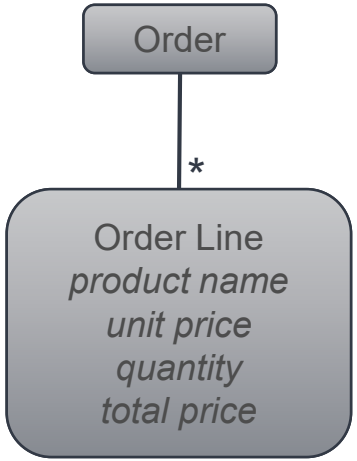
Aggregates

Command model

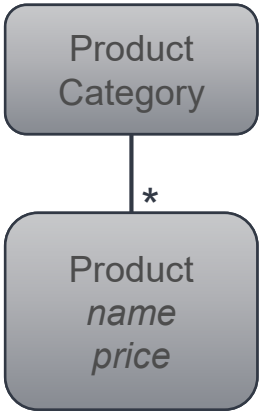


Revisiting the example

Order aggregate



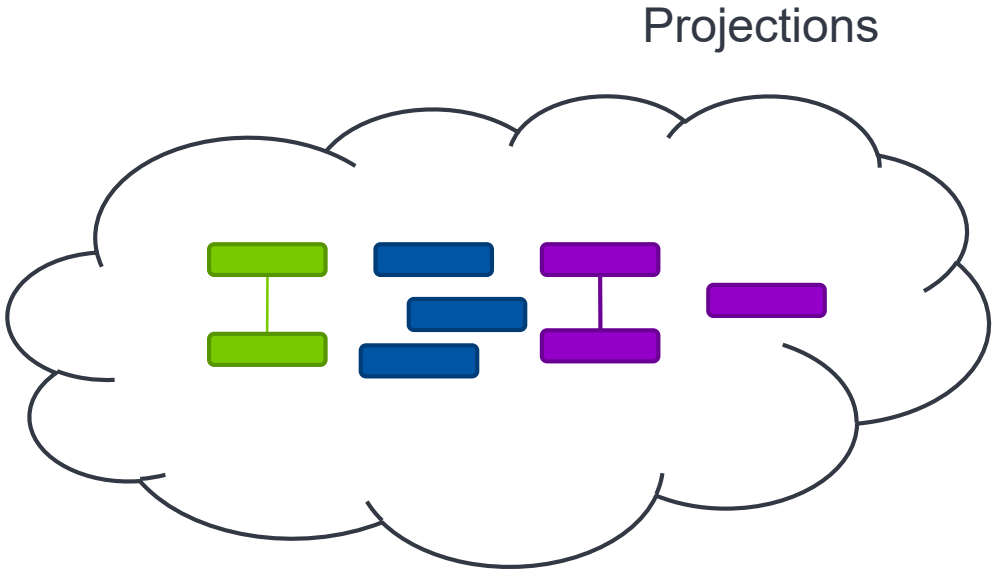
Product aggregate



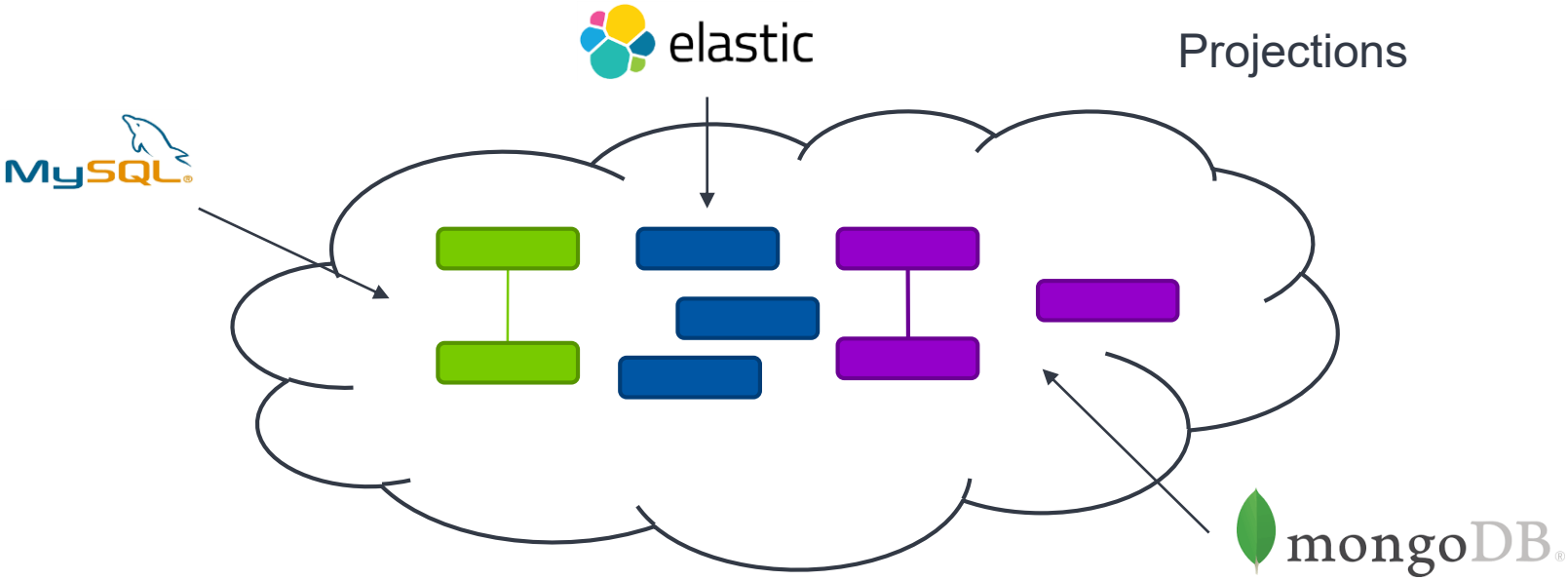
Read models

Optimized for the specific read use-cases (e.g. screens, API methods)

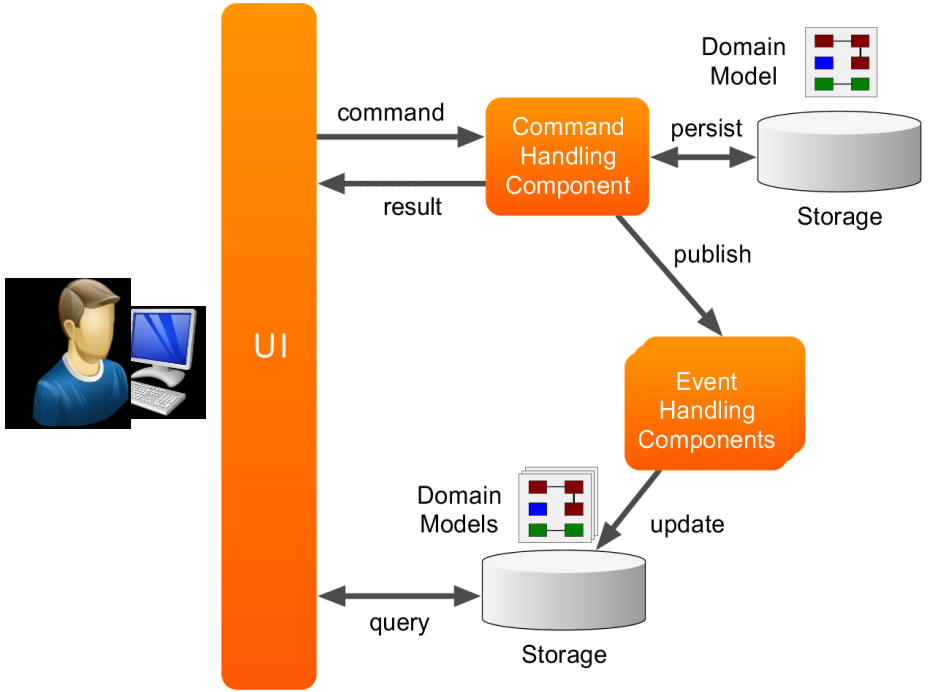
Many separated ones instead of one big one.



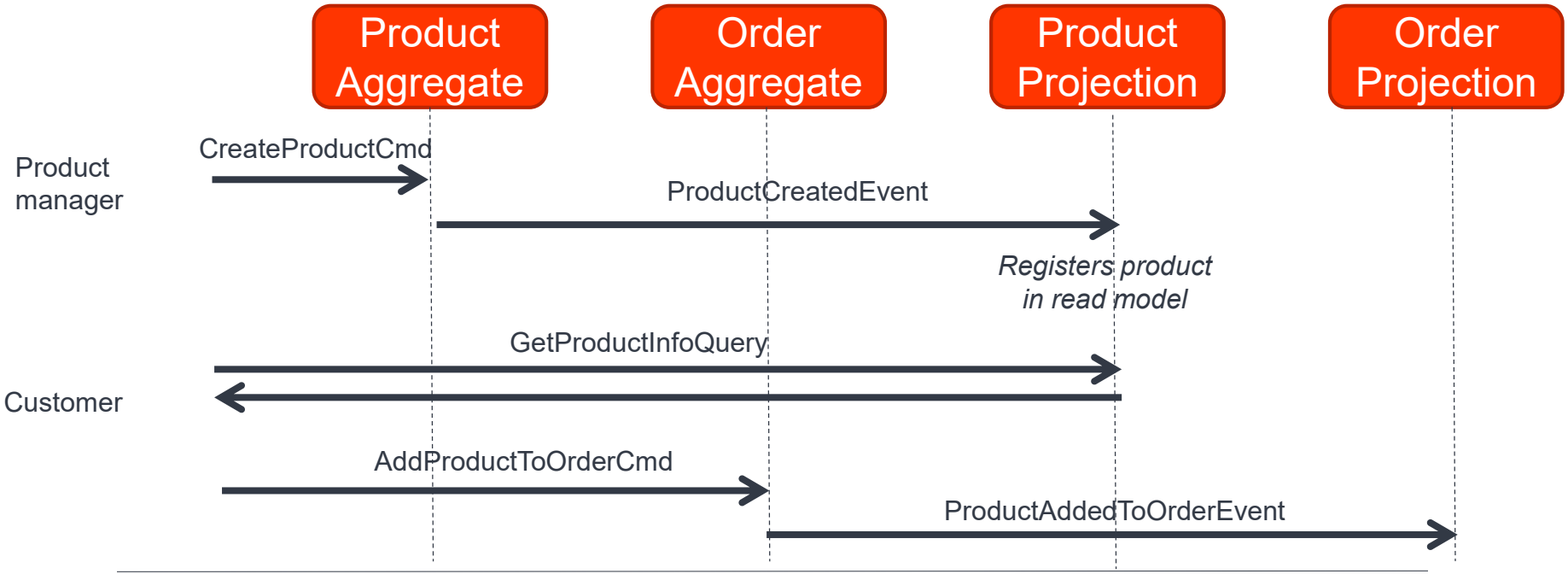
Read models



CQRS Based Architecture



All together



AxonIQ

About persisting aggregates

Traditional method

- Store the current state directly to database (e.g. by using JPA).

About persisting aggregates

Traditional method

- Store the current state directly to database (e.g. by using JPA).

Event Sourcing

- Aggregates only change state through events.
- Events are distributed *and* persisted to an event store.
- To read an aggregate, read all events belonging to that aggregate and *replay* them.

Why use event sourcing?

Business reasons

- Auditing / compliance / transparency
- Data mining, analytics: value from data

Technical reasons

- *Guaranteed completeness of raised events*
- *Single source of truth*
- *Concurrency / conflict resolution*
- *Facilitates debugging*
- *Replay into new read models (CQRS)*
- *Easily capturing intent*

Example: interesting history

Traditional

Order:

2 bananas
1 peach

Example: interesting history

Traditional

Order:

2 bananas
1 peach

Event Sourcing

Order:

3 apples added
2 bananas added
3 apples removed
1 peach added

Example: capturing intent

Traditional

UPDATE
customer_address
SET
line1 = “10 Random St”

Example: capturing intent

Traditional

UPDATE
customer_address

SET
line1 = “10 Random St”

Event Sourcing

WrongAddressCorrectedEvent
(newLine1 = “10 Random St”)

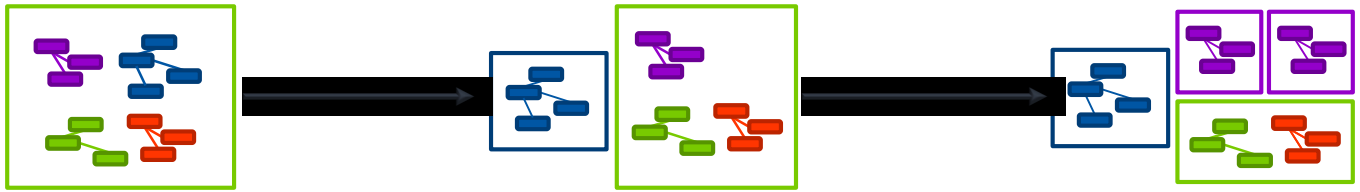
Or

CustomerRelocatedEvent
(newLine1 = “10 Random St”)

CQRS (+ optionally event sourcing):

1 ingredient for the big friendly monolith

Location transparency



A component should neither be aware of nor make any assumptions about the location of components it interacts with.

Location transparency starts with good API design
(but doesn't end there)

Location transparency in Axon

Components

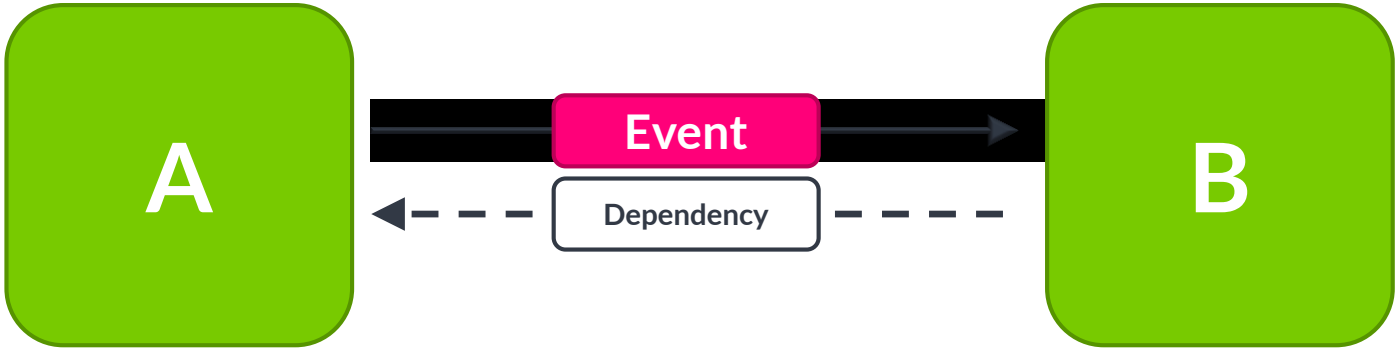
- All communications between components take place through a bus API.
- Components don't know where the others are.

Infrastructure

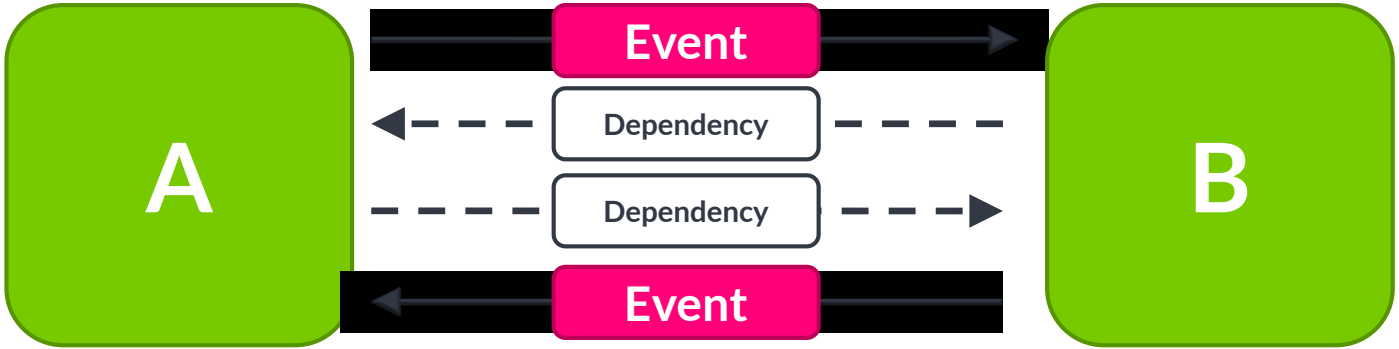
- Implementations of those buses can be switched without changing the business logic.

It’s NOT just events!

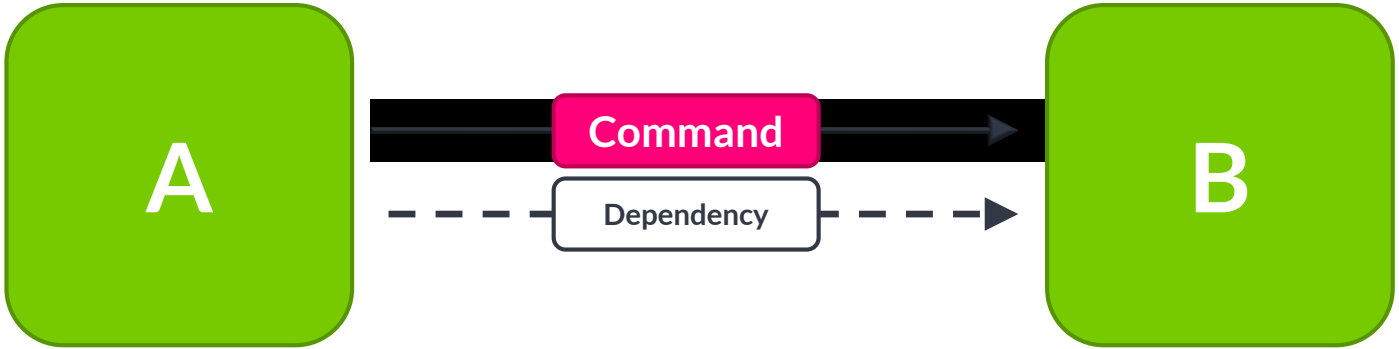
It’s NOT just events!



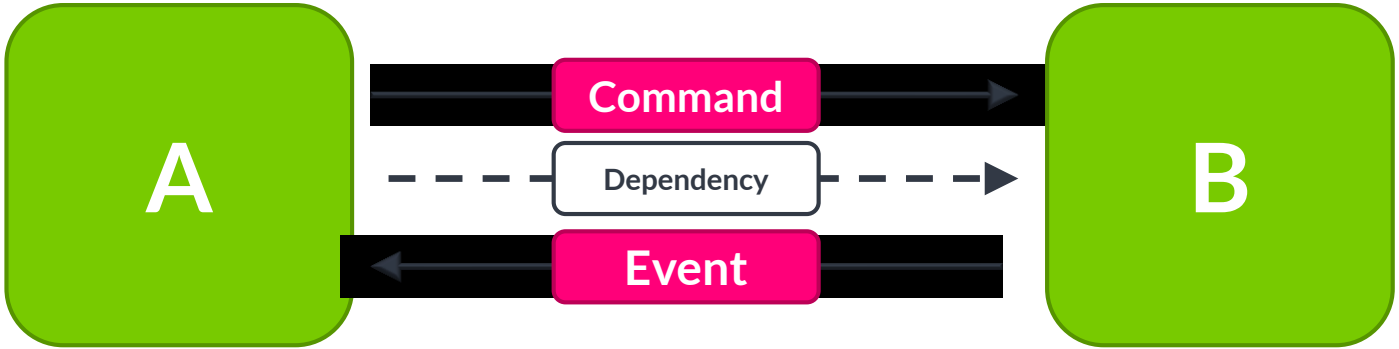
It’s NOT just events!



It’s NOT just events!



It’s NOT just events!



Messaging patterns

Command

- Handled once
- Confirmed
- Consistent routing

Event

- Sent to everyone
- Unconfirmed
- Order consistency

Query

- Gets an answer
- Usually handled once
- May have scatter/gather characteristics

Code examples

Command and Events

```
data class CreateProductCommand(
        val productId: UUID,
        val name: String,
        val price: BigDecimal)

data class CreateNewOrderCommand(
        val orderId: UUID)

data class AddOrderLineCommand(
        @TargetAggregateIdentifier val orderId: UUID,
        val orderLineId: UUID,
        val productId: UUID,
        val productName: String,
        val quantity: Int,
        val unitPrice: BigDecimal)
```

Sending commands

```
@Component
public class OrderService {

    @Autowired
    private final CommandGateway commandGateway;

    public UUID createNewOrder() {
        UUID id = UUID.randomUUID();
        commandGateway.send(new CreateNewOrderCommand(id));
        return id;
    }

}
```

AxonIQ

Aggregates

```
@Aggregate
public class Order {

    @AggregateIdentifier UUID orderId;

    @CommandHandler
    void handle(AddOrderLineCommand cmd) {
        if(cmd.getQuantity() < 1) throw new IllegalArgumentException("quantity must be >= 1");
        apply(new OrderLineAddedEvent(cmd.getOrderId(), cmd.getOrderLineId(), cmd.getProductId(),
                   cmd.getProductName(), cmd.getQuantity(), cmd.getUnitPrice()));
    }

    @EventHandler
    void handle(OrderLineAddedEvent evt) {
    }
}
```

Aggregates

```
@Aggregate
public class Order {

    @AggregateIdentifier
    UUID orderId;
    BigDecimal totalOrderValue;

    @CommandHandler
    void handle(AddOrderLineCommand cmd) {
        if(cmd.getQuantity() < 1) throw new IllegalArgumentException("quantity must be >= 1");
        if(totalOrderValue.add(cmd.getUnitPrice().multiply(BigDecimal.valueOf(cmd.getQuantity())))
                .compareTo(BigDecimal.valueOf(10000L)) > 0) {
            throw new IllegalStateException("Total order value must be <= 10000");
        }
        apply(new OrderLineAddedEvent(cmd.getOrderId(), cmd.getOrderLineId(), cmd.getProductId(),
           cmd.getProductName(), cmd.getQuantity(), cmd.getUnitPrice()));

    }

    @EventHandler
    void handle(OrderLineAddedEvent evt) {
        totalOrderValue = totalOrderValue.add(
                evt.getUnitPrice().multiply(BigDecimal.valueOf(evt.getQuantity())));
    }

}
```

Read models

```
@Component
public class AllOrdersProjection {

    @Autowired
    EntityManager entityManager;

    @EventHandler
    void handle(OrderLineAddedEvent evt) {
        /* Find Order JPA entity, add the line so it gets persisted. */
    }

    public List<OrderRecord> findAllOrders() {
        return entityManager.createQuery("select e from OrderRecord e",
                OrderRecord.class).getResultList();
    }

}
```

AxonIQ

Setting up Axon with default command gateway, commandbus, eventbus, event sourcing repositories, and serialization mechanism

This page intentionally left blank.

Making the command bus async

```
/* The default with Axon Spring Boot */
@Bean
public CommandBus commandBus() {
    return new SimpleCommandBus();
}

/* To make this async */
@Bean
public CommandBus commandBus() {
    return new AsynchronousCommandBus();
}
```

Making the command bus distributed with JGroups

Put JGroups and Axon distributed command bus on the classpath.

Set property:

```
axon.distributed.enabled=true
```

Making the command bus distributed with Spring Cloud

```
/* To make this distributed */
@Bean
public CommandRouter springCloudCommandRouter(DiscoveryClient discoveryClient) {
    return new SpringCloudCommandRouter(discoveryClient, new AnnotationRoutingStrategy());
}
@Bean
public CommandBusConnector springHttpCommandBusConnector(@Qualifier("localSegment")
            CommandBus localSegment, RestOperations restOperations, Serializer serializer)
{
    return new SpringHttpCommandBusConnector(localSegment, restOperations, serializer);
}
@Primary // to make sure this CommandBus implementation is used for autowiring
@Bean
public DistributedCommandBus springCloudDistributedCommandBus(CommandRouter commandRouter,
                                                    CommandBusConnector commandBusConnector) {
    return new DistributedCommandBus(commandRouter, commandBusConnector);
}
```

AxonIQ

To summarize

- CQRS, DDD and location transparency are the key ingredients to a big friendly monolith that enables evolutionary microservices.
- CQRS enables event sourcing, which has a important set of benefits in and by itself.
- Axon Framework enables you to implement this as easily as possible in Java applications.