Fast lane Axon Framework

The quick intro into DDD, CQRS and Event Sourcing with Axon

AxonIQ

Steven van Beelen

@smcvbeelen

Agenda

- Architecture Overview
  - Domain-Driven Design (DDD)
  - Command Query Responsibility Segregation (CQRS)
  - Event Sourcing (ES)

- Axon Framework
  - Command Handling & Aggregate Design
  - Event Handling & Query Models

- Hands-on!

In theory

Architecture Overview

The components that make up the model

Domain-Driven Design

Steven van Beelen @smcvbeelen

Domain-Driven Design

- In DDD, the Domain Model is at the foundation of every application.
- A growth of complexity is often caused by a badly designed model.

Definitions

Domain

A sphere of knowledge, influence, or activity. The subject area to which the user applies a program is the domain of the software.

Model

A system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain.

Domain Model

- Contains the “concepts” used in the client’s domain that are used to solve a specific problem.
- Never contains a concept that the client doesn’t care about.
- Remember, it’s a model!

The Domain

Model


ARCTIC OCEAN
ARCTIC OCEAN
ASIA
PACIFIC
NORTH AMERICA
ATLANTIC
EUROPE
AFRICA
OCEAN
SOUTH AMERICA
OCEAN
INDIAN
OCEANIA
OCEAN

DDD Building blocks

- Entity
- Value Object
- Repository
- Aggregate
- Event
- …

Entity

Objects that are not fundamentally defined by their attributes, but rather by a thread of continuity and identity.

Value Object

Value objects have no conceptual identity, but are fundamentally defined by their attributes.

They describe some characteristic of a thing.

Value Objects are Immutable!

Entity or Value Object?

Human Being

It depends

Repository

A mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects.

Aggregate

- A group of associated objects which are considered as one unit with regard to data changes.
- External references are restricted to one member of the aggregate, designated as the Aggregate Root.
- A set of consistency rules applies within the Aggregate's boundaries.

Event

A notification that something relevant has happened inside the domain.

Event are immutable.

Event handling is conceptually asynchronous.

Model distinction

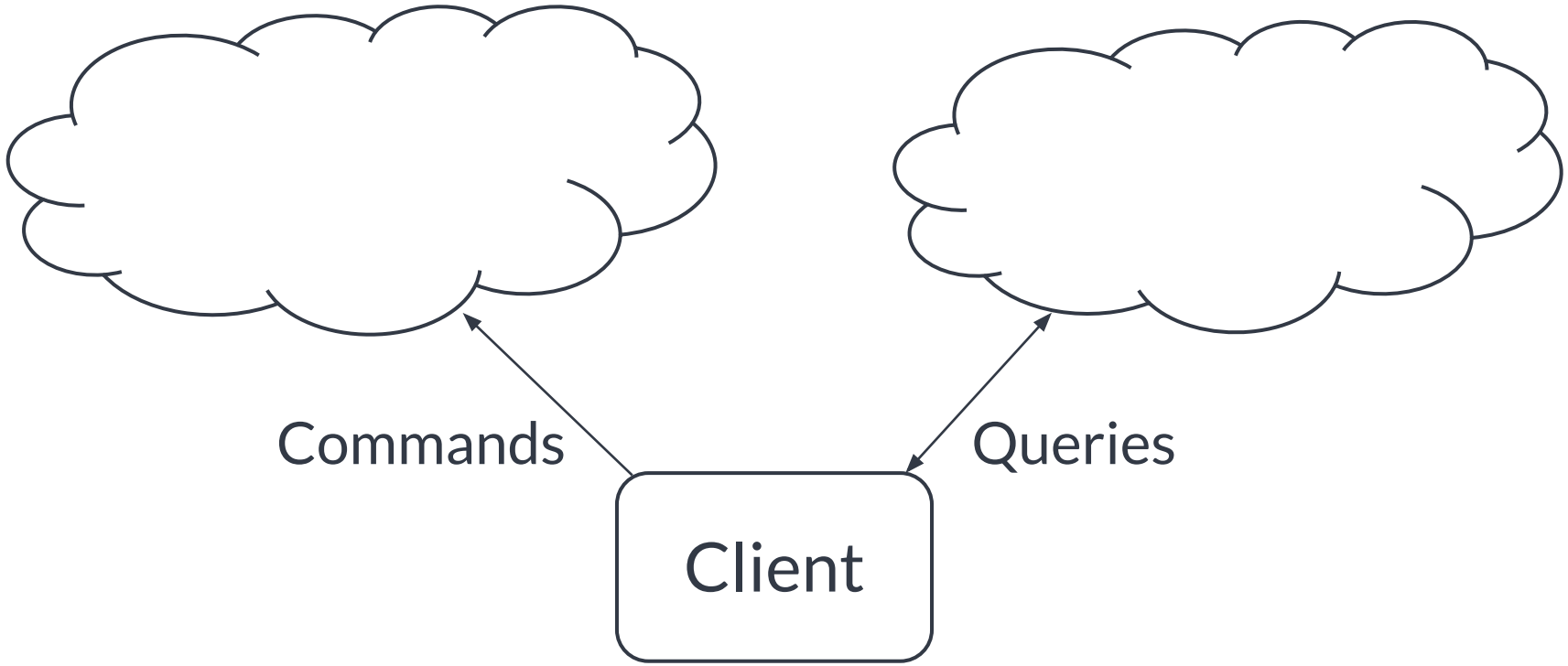
Command Query Responsibility Segregation

CQRS - Definition

Command Query Responsibility Segregation is an architectural pattern that distinguishes between two parts of an application:

- one with the responsibility to process commands,
- another that provides information (queries).

Command Query Responsibility Segregation



Two Models

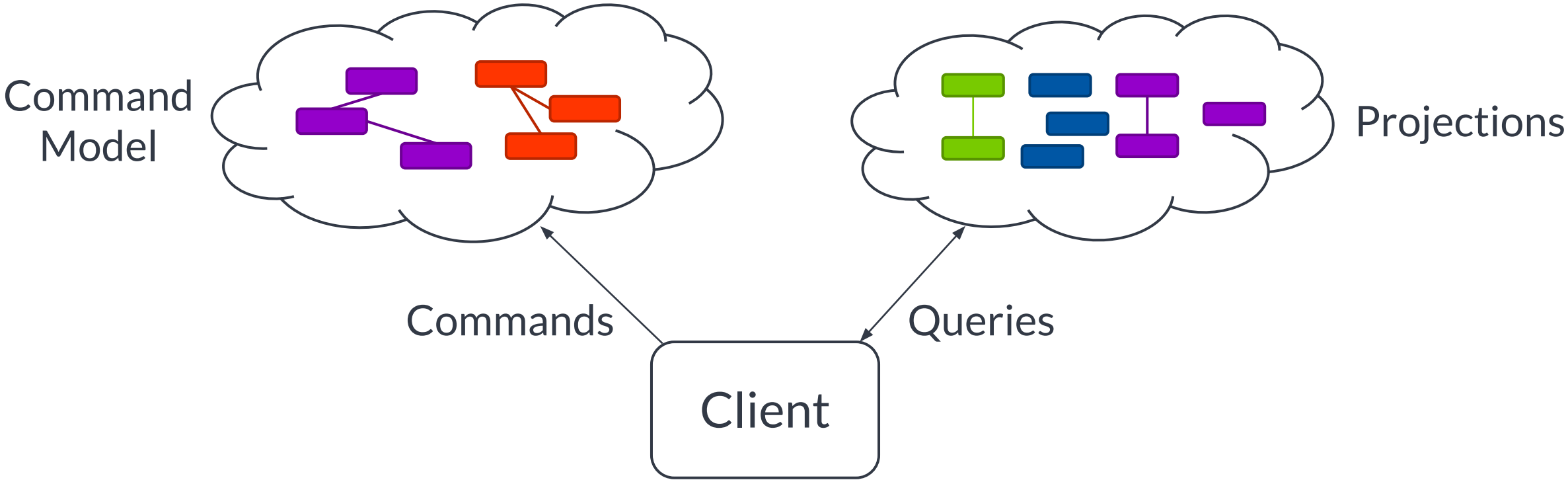
Command Model

- Focused on executing tasks.
- Primarily expressed in operations.
- Only contains data necessary for task execution and decision making.

Query Model / Projections

- Focused on delivering information.
- Data is stored the way it is used.
- Denormalized / “table-per-view”

Command Query Responsibility Segregation



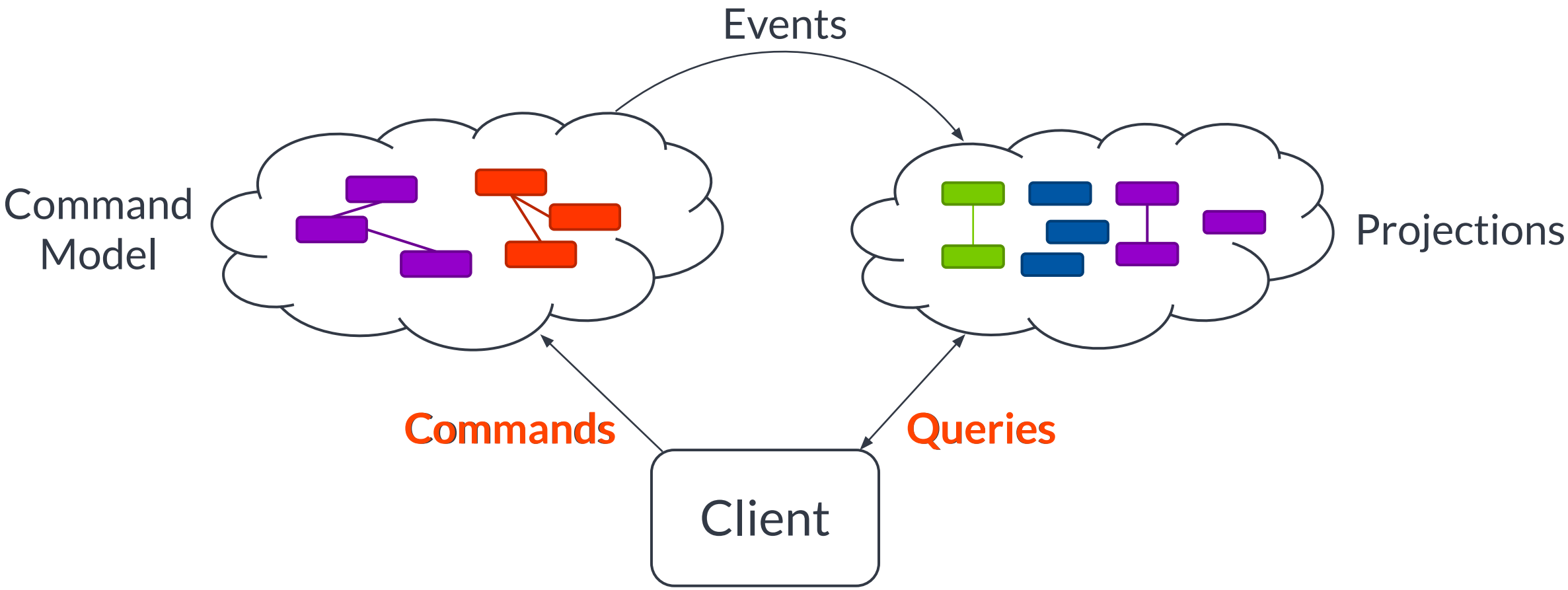
Synchronization of Models

Changes in the Command Model should (eventually) be visible in the Query Model.

Options:

- Shared data source
- Stored procedures
- Event Driven Architecture

Command Query Responsibility Segregation



CQRS Building Blocks

Command
An expression of intent to trigger an action in the domain.

Query
A request for information or state.

Reliable, executable audit trails

Event Sourcing

Steven van Beelen

@smcvbeelen

Event Sourcing

Storage method for the Command Model.

- Only persist changes.
- The generated events can be used.

To load an aggregate:

- Replay all the past events on an “empty” instance.

Event Store

Responsible for storage of events.

Primary focus: writing (append).

Axon Server is a built-for-purpose Event Store.

- Axon also has implementations that support JPA, JDBC and MongoDB

Event Sourcing as Business Case

Event Sourcing has less information loss.

- Event Store contains information that can be used in different ways in the future.

Event Store is a reliable audit log.

- Not only state, but also how it is reached.

Event Sourcing increases performance.

- Only deltas need to be stored. Caches prevent reads.

Event Sourcing

State Storage

id: 123

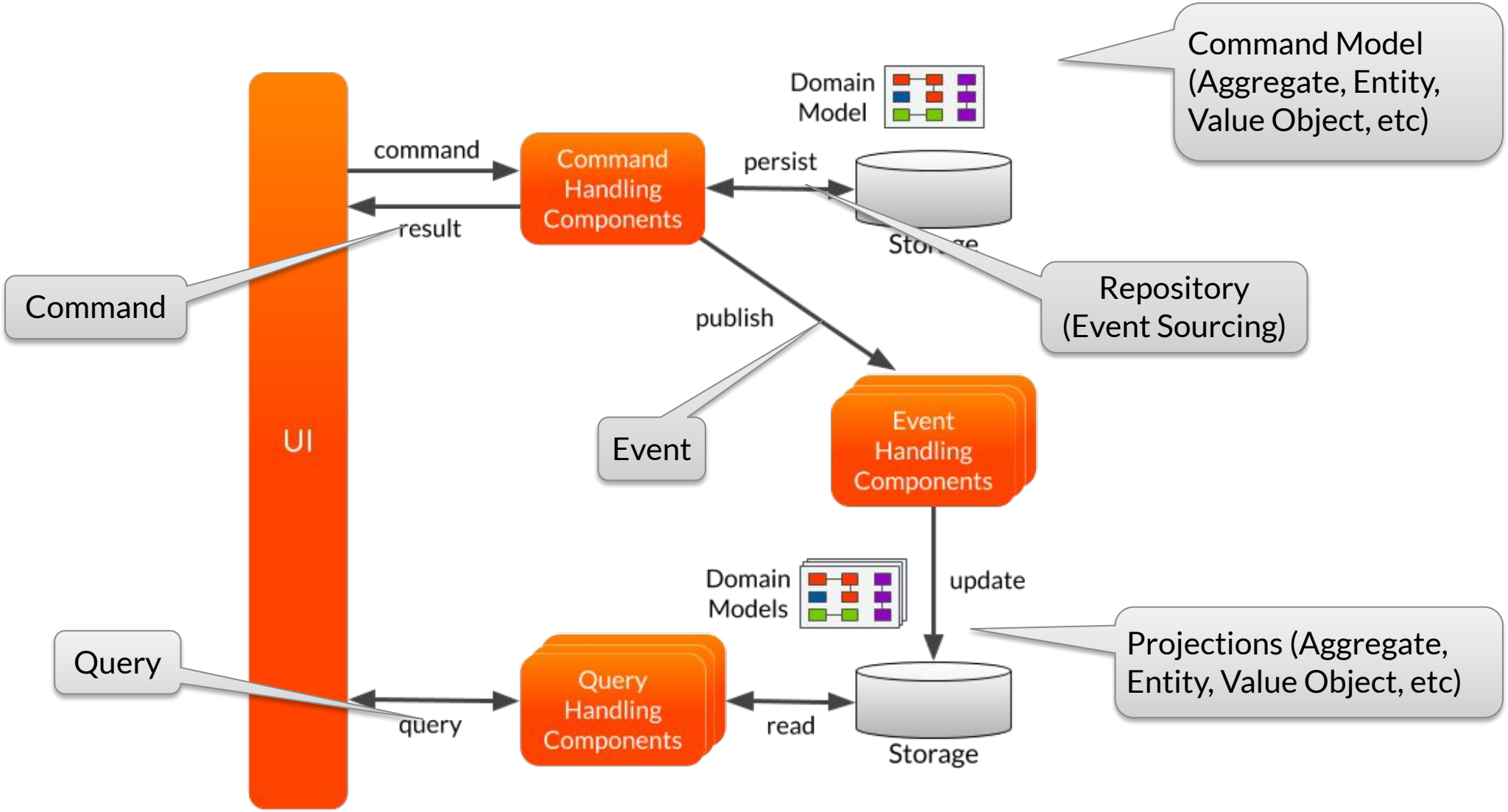
Items:

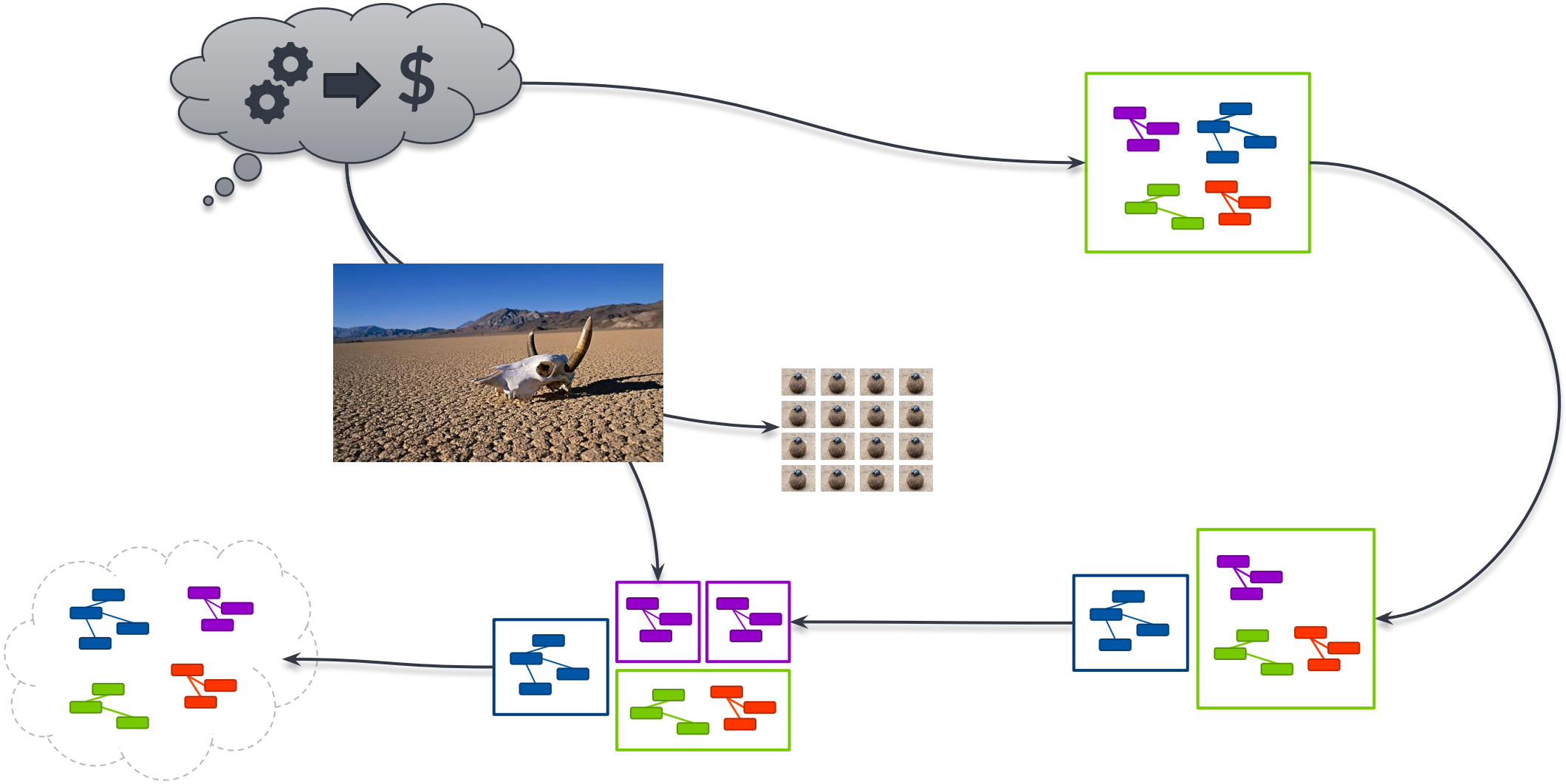
- 1x Deluxe Chair - € 399

status: return shipment rcvd

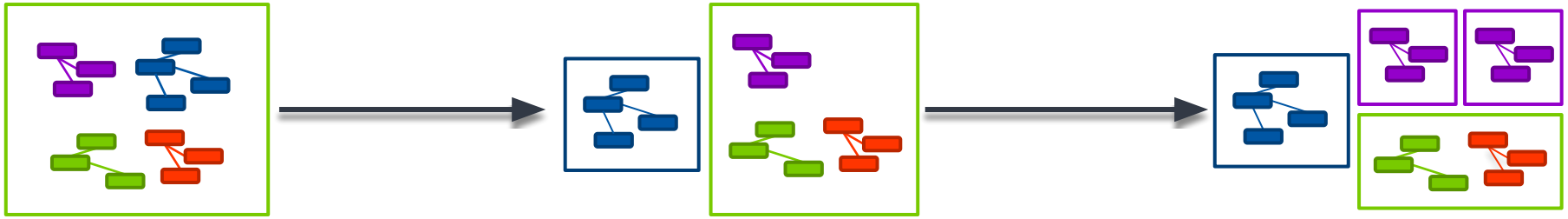
Event Sourcing

- OrderCreated *(id: 123)*
- ItemAdded *(2x Deluxe Chair, €399)*
- ItemRemoved *(1x Deluxe Chair, €399)*
- OrderConfirmed
- OrderShipped
- OrderCancelledByUser
- ReturnShipmentReceived


Command Model (Aggregate, Entity, Value Object, etc)
Domain Model
command
Command Handling Components
persist
result
Command
Repository (Event Sourcing)
publish
Event
UI
Event Handling Components
Domain Models
update
Query
Projections (Aggregate, Entity, Value Object, etc)
Query Handling Components
query
read
Storage



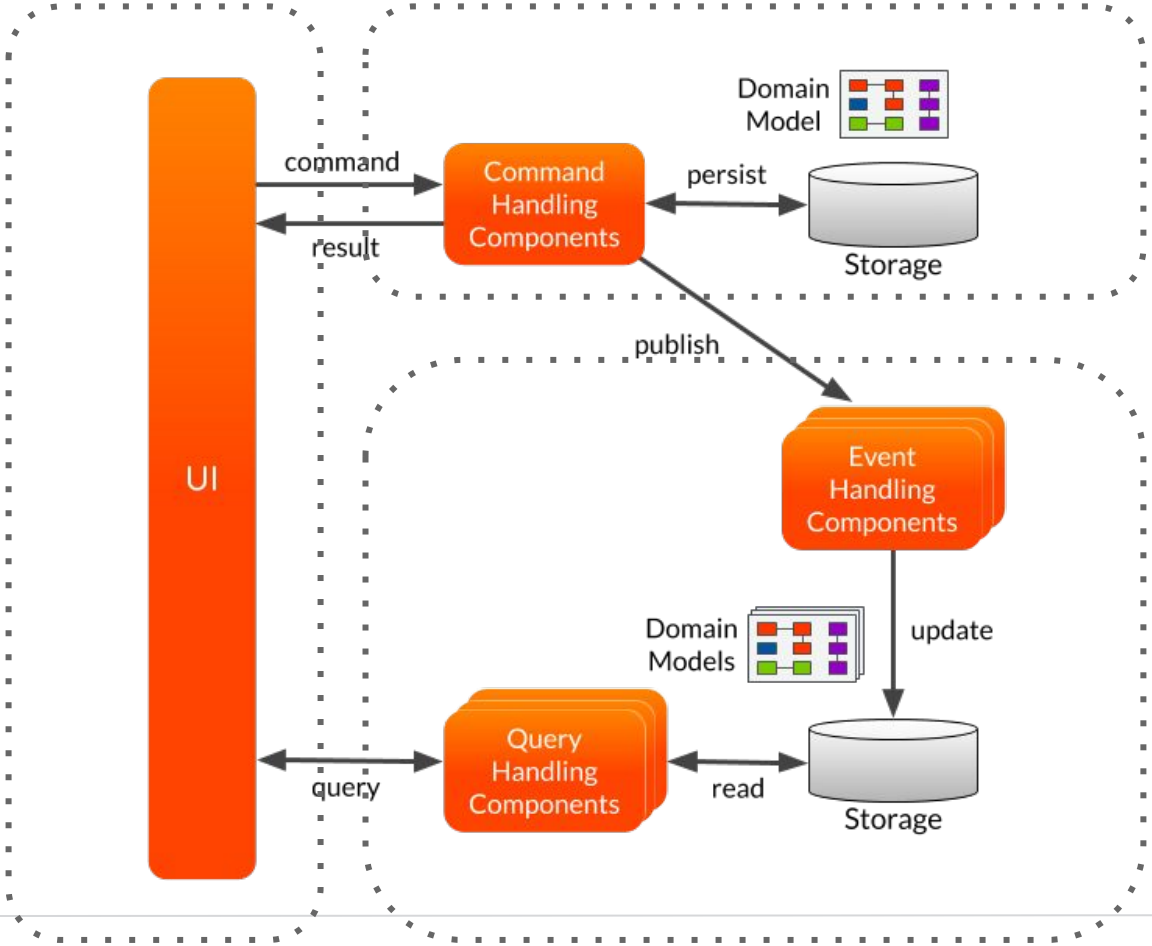
Location transparency



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 boundaries



In practice

Axon Framework

Steven van Beelen

@smcvbeelen

The Command Model through Axon

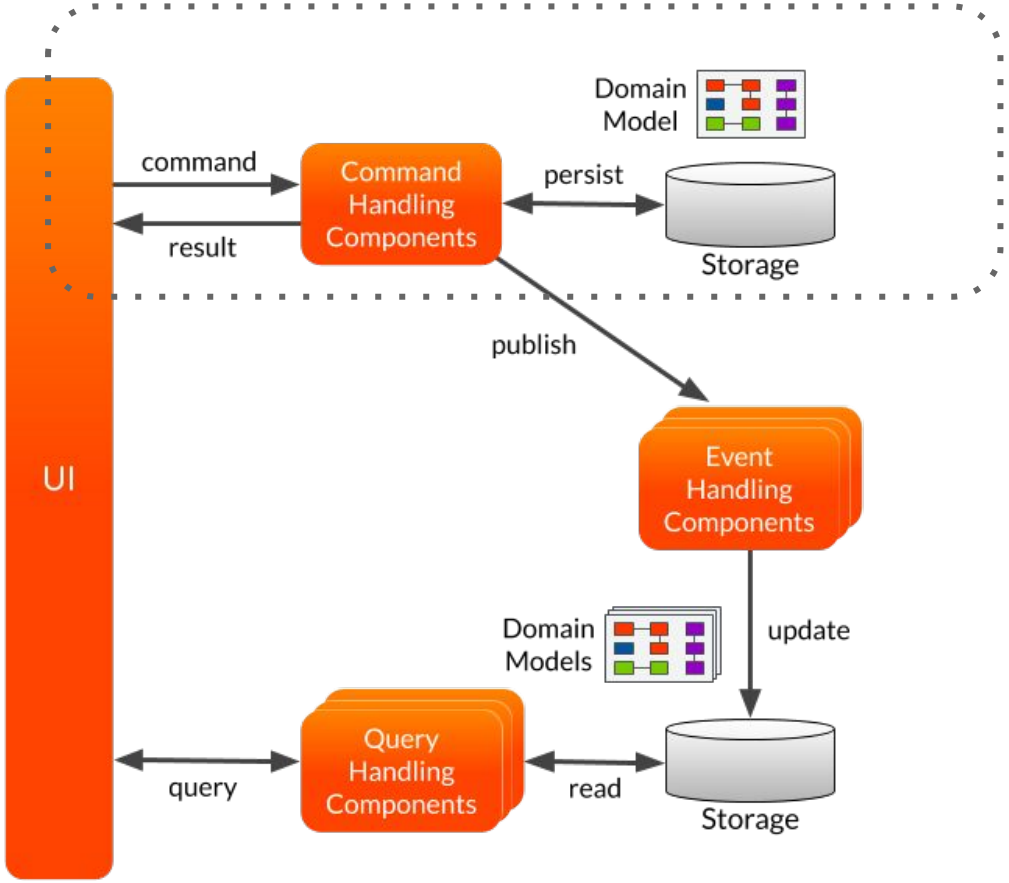
Command Handling & Aggregate Design

AxonIQ

Steven van Beelen

@smcvbeelen

Command Handling & Aggregate Design



Command Handler

- Accepts incoming commands
- Consults the command model and publishes events
- Command model only contains data necessary for task execution and decision making

Command Handling in Axon

A component that is subscribed to the Command Bus to process specific Commands.

`@CommandHandler`

- On (singleton) component
- Directly on Command Model

```
@CommandHandler
public void handle(JoinRoomCommand command) {
    …
}
```

Annotated Command Model (with Spring)

```
import static org.axonframework.modelling.command.AggregateLifecycle.apply

@Aggregate
public class ChatRoom {

  @AggregateIdentifier
  private String roomId;

  @CommandHandler
  public void handle(JoinRoomCommand cmd) {
    apply(new RoomJoinedEvent(roomId, …));
  }
}
```

Tells Axon Spring Auto configuration to set up necessary infrastructure.

Indicates which of the fields is the identifier.

Registers this method as a Command Handler for "`JoinRoomCommand`".

Publishes a "`RoomJoinedEvent`" via the event bus registered with the Repository that manages this instance's lifecycle.

Event Sourcing the Command Model

```
@Aggregate
public class ChatRoom {
  // Some decision driving state …

  @CommandHandler
  public ChatRoom(CreateRoomCommand cmd) {
    apply(new RoomCreatedEvent(cmd.getRoomId(), …));
  }

  @EventSourcingHandler
  public void on(RoomCreatedEvent event) {
    roomId = event.getRoomId();
  }
}
```

Decision making

State changes

Event sourcing handlers **only** handle events from *this* aggregate instance.

Command Message Routing

```
public class CreateRoomCommand {
  @TargetAggregateIdentifier
  private final String roomId;
  // Other state
  public CreateRoomCommand(String roomId, …) {
    this.roomId = roomId;
  }
  // Getters
}

// Or in Kotlin:
data class CreateRoomCommand(@TargetAggregateIdentifier val roomId: String, …)
data class RoomCreatedEvent(val roomId: String, …)
```

Marks the field that contains the value to use to load an Aggregate

Tip: Kotlin allows one-liner definitions of messages. You can also group many of them in a single file.

Dispatching Commands

Directly on **CommandBus**:

```
CommandBus commandBus;
commandBus.dispatch(asCommandMessage(new CreateRoomCommand(…)));
```

Or using **CommandGateway**:

```
CommandGateway gateway =
    DefaultCommandGateway.builder().commandBus(commandBus).build();
// Non-blocking
gateway.send(new CreateRoomCommand(…)); // Returns CompletableFuture<>
// Blocking
gateway.sendAndWait(new CreateRoomCommand(…));
gateway.sendAndWait(new CreateRoomCommand(…), 1, TimeUnit.SECONDS);
```

The Query Models through Axon

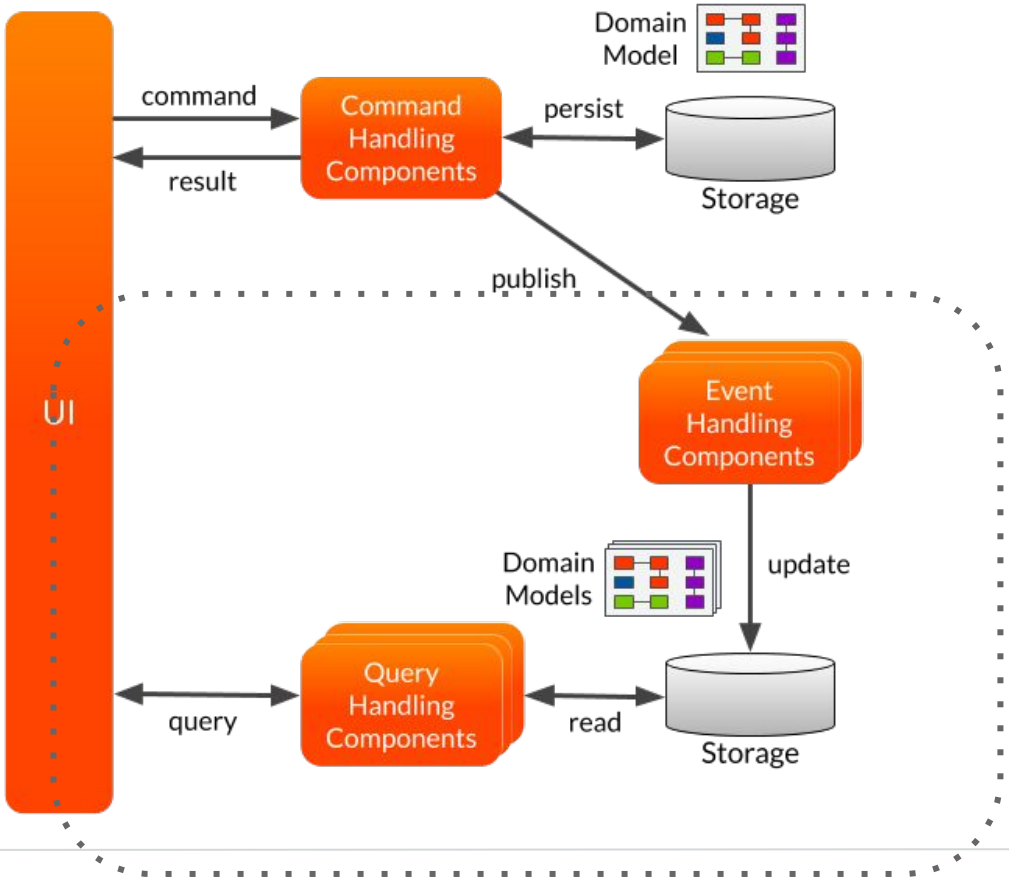
Event Handling & Query Models

AxonIQ

Steven van Beelen

@smcvbeelen

Event Handling & Query Models



Event Handler

Handles published events to:

- Update Projections
- Trigger (external) activities
- Manage complex transactions (Sagas)

Event Handling in Axon

A component that is subscribed to the Event Bus to handle specific Events.

`@EventHandler`

- On (singleton) component

```
@EventHandler
public void on(RoomCreatedEvent event) {
    …
}
```

Event Handling Component (with Spring)

```
@Component
public class EventHandlingComponent {

  @EventHandler
  public void on(RoomCreatedEvent event) {
    // Do what you need to do
  }
}
```

Query Model

Model optimized to answer queries.

- Focused on delivering information.
- Denormalized to suit information needs (e.g. “table-per-view”).
- Updated by Event Handling Component.

Consciously optimize for

- Performance
- Storage
- Flexibility

Query Database Denormalization

Optimize the query database (i.e. the query model) for your UI

OrderHeader Table

CustId	OrderId	CustomerName	Address	Total amount
12	56	John Doe	Amsterdam	€ 38,00
12	57	John Doe	Amsterdam	€ 85,00
13	58	Sjonnie	Den Haag	€ 12,00

Optimized for specific Use Case

Optimized for full-data retrieval based on ID

- Give all order details for Order '123'

OrderDetails Table

OrderId	OrderData
56	{"customer":"John Doe", "orderItems" : [{"itemId": 123, "item…
57	{"customer":"John Doe", "orderItems" : [{"itemId": 456, "item…
58	{"customer":"Sjonnie", "orderItems" : [{"itemId": 789, "itemN…

Query Handling in Axon

A component that is subscribed to the Query Bus to handle specific Queries, returning specific Query Responses.

`@QueryHandler`

- On (singleton) component

```
@QueryHandler
public List<String> handle(RoomParticipantsQuery query) {
    …
}
```

Query Handling Component (with Spring)

```
@Component
public class QueryHandlingComponent {

  @QueryHandler
  public SomeResponse handle(SomeQuery query) {
    // Find that data and return it.
  }

  @QueryHandler
  public List<SomeListResponse> handle(SomeListQuery query) {
    // Find that data and return it.
  }
}
```

Dispatching Queries

Directly on **QueryBus**:

```
QueryBus queryBus;
queryBus.query(new GenericQueryMessage<>(
    new FindRoomQuery(…),
    ResponseTypes.instanceOf(RoomSummary.class)
));
```

Or using **QueryGateway**:

```
QueryGateway gateway =
    DefaultQueryGateway.builder().queryBus(queryBus).build();
// Non-blocking, returns CompletableFuture<>
gateway.query(
    new RoomParticipantsQuery(…),
    ResponseTypes.multipleInstancesOf(String.class)
);
```

Types of Queries

Point-to-point query

- Single destination

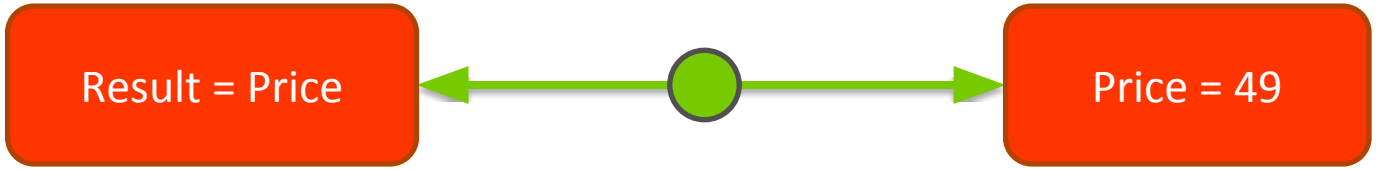
Scatter-gather query

- Published, multiple responses

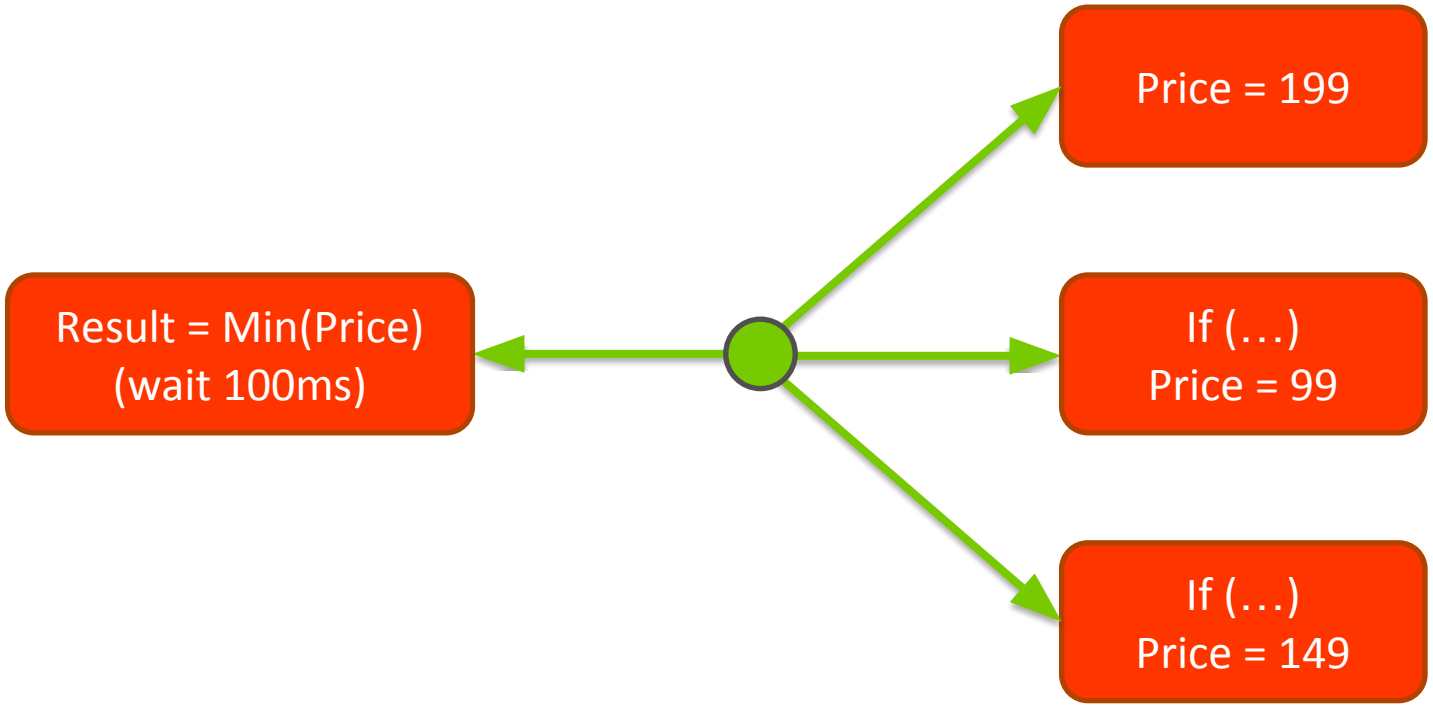
Subscription query

- Single destination for initial result
- Real-time updates

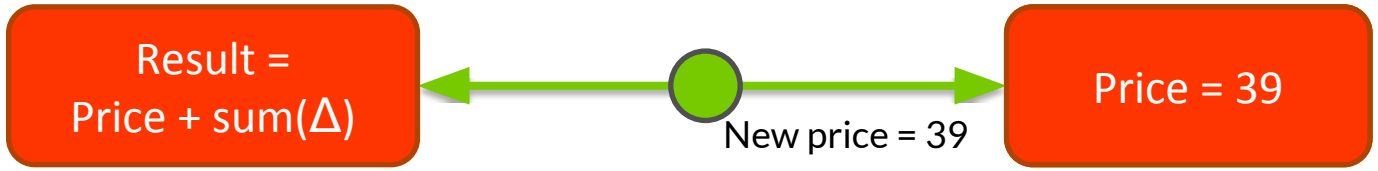
Query – Point to Point



Query – Scatter-gather



Query – Subscription



Query - Subscription - Emitting Updates

```
@Component
public class ChatMessageProjection {
  private QueryUpdateEmitter emitter;

  @QueryHandler
  public List<ChatMessage> handle(RoomMessagesQuery query) {
    // Find that data and return it
  }

  @EventHandler
  public void on(MessagePostedEvent event) {
    ChatMessage chatMessage = new ChatMessage(...)
    emitter.emit(
        RoomMessagesQuery.class, query -> /* query selection */, chatMessage
    );
  }
}
```

Summary - Architecture Overview

- DDD
  - Domain Model
  - Entity / Value Object
  - Aggregate
- CQRS
  - Command Model
  - Query Model
  - Message Driven Architecture
- Event Sourcing
- Location Transparency

Summary
Command Handling & Aggregate Design

- `@CommandHandler`
  - Singleton Component
  - Command Model
- `@Aggregate`
  - `@AggregateIdentifier`
  - `AggregateLifecycle#apply`
  - `@EventSourcingHandler`
- Command Dispatching
  - `@TargetAggregateIdentifier`
  - CommandBus
  - CommandGateway

Summary
Event Handling & Query Models

- `@EventHandler`
  - Singleton Component
  - Update Projections / Query Models
- `@QueryHandler`
- Query Dispatching
  - `ResponseType`
  - QueryBus
  - QueryGateway
- Query Types
  - Point-to-Point
  - Scatter-Gather
  - Subscription -> `QueryUpdateEmitter`

Hands-on!

- Clone https://github.com/AxonIQ/axon-quick-start

- Run Axon Server in Docker:

```
docker run -d -p 8024:8024 -p 8124:8124
    --name axonserver axoniq/axonserver
```

- Or download Axon Server here:

  http://download.axoniq.io/training/AxonServer.zip

  And run: `java -jar axonserver.jar`

- Select *chat-getting-started* from the main README.md

- Follow the description and exercises from the *chat-getting-started*.

Thanks for attending!

- You can find the survey here:
  https://surveys.hotjar.com/s?siteId=1684500&surveyId=155045