Axon Training

Module 3 – Event Handling & Projections

AxonIQ

Agenda

Week 1

1. DDD and CQRS Fundamentals
2. Command Model
3. **Event Handling & Projections**
4. Sagas and Deadlines

Week 2

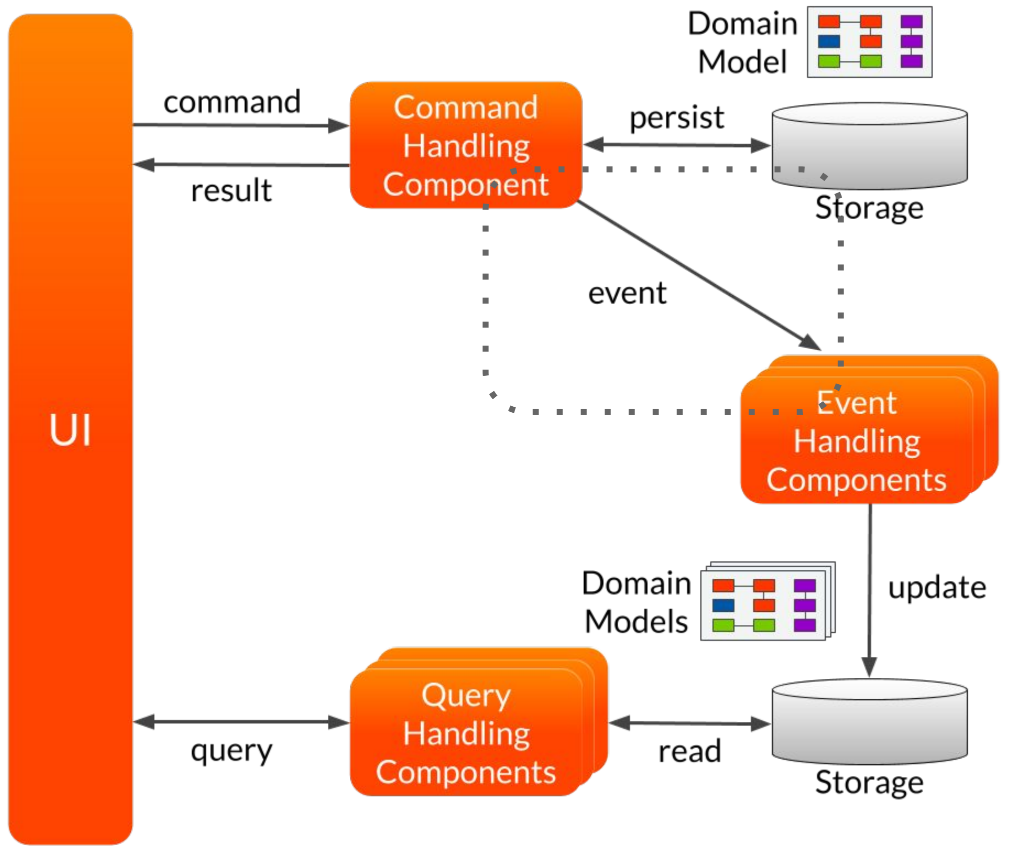
1. Snapshotting and Event Processors
2. Preparing for Production
3. CQRS and Distributed Systems
4. Monitoring, Tracing, Advanced Tuning

Dealing with the consequences…

Event Handling

AxonIQ

Event Handling



AxonIQ

Event Handler

- Handles published events
    - Projections
    - Trigger (external) activities
    - Manage complex transactions (Sagas)

Event Handling in Axon Framework

- Component that is subscribed to the Event Bus to handle specific Events
- `@EventHandler`
  - On (singleton) component

```
@EventHandler
public void on(MyEvent event) {
    ...
}
```

Event Handling Component (with Spring)

```
@Component
public class EventHandlingComponent {

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

Organizing Event Handlers

- Event Processor
  - Responsible for managing the technical aspect of processing an Event
  - Starts and Commits Unit of Work
  - Invokes handler methods

- Each handler is assigned to a single Processor
  - @ProcessingGroup on Event Handler class
  - Assignment rules in EventProcessingConfigurer
    (part of Configuration API)

Event Processors

- SubscribingEventProcessor
  - Receives messages as they are published, in the thread that publishes the messages
  - Requires a subscribable message source

- TrackingEventProcessor (default *)
  - Uses its own thread(s) to read EventMessages from a Stream
  - Requires a streamable message source
  - Records progress using TrackingToken

Event Handler parameters

Supported parameter types

- First parameter (if none of below) resolves to `Message` payload
- `Message` → Resolves to entire message
- `EventMessage` → Resolves to `EventMessage`
- `UnitOfWork` → Resolves to the current Unit of Work
- `MetaData` → Resolves to the `MetaData` of the `Message`
- `@MetaDataValue (“name”)` … → Resolves to a Meta Data value of the `Message`
- `@Timestamp` → Resolves the `Instant` on which the `EventMessage` was created
- Any Spring bean or component registered using Configuration API

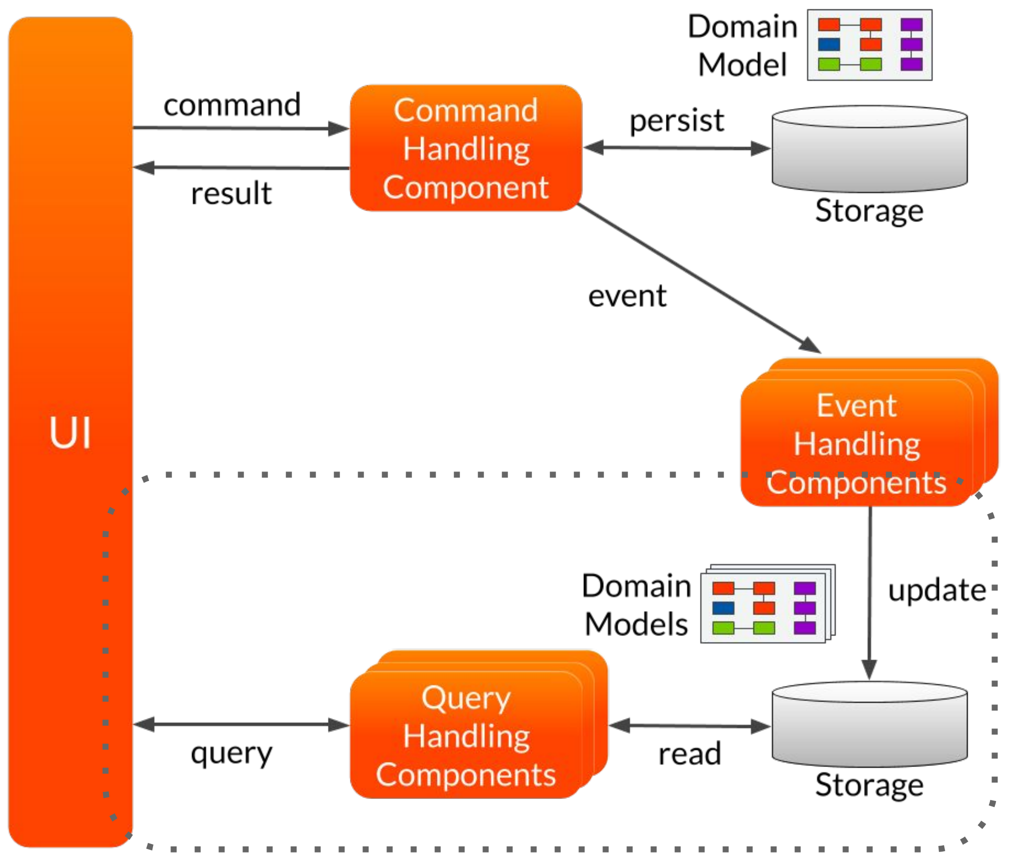
Custom values using `ParameterResolverFactory`

Projecting events into useful information

Projections

AxonIQ

Projections



AxonIQ

Query Model

- Model optimized to answer queries
  - Focused on data
  - Denormalized to suit information needs (e.g. table per view)
  - Updated by Event Handling component

- Consciously optimize for
  - Performance
  - Storage
  - Flexibility

Example - Query Model

- Relevant information for Passengers

FlightTimes Table

PNR	Origin	Dest.	Departure	Arrival	Layover
BGTR4	AMS	SLC	8:12	10:50	2:48
BGTR4	SLC	LAX	10:30	12:50	<null>
HGYT2	JKF	GRU	8:12	15:50	<null>

Example - Query Model

- Relevant information for Pilots

FlightTimes Table

Origin	Dest.	Departure	Arrival	Captain	Aircraft
AMS	SLC	10:50	8:12	C. Lindbergh	B747
SLC	LAX	10:30	12:50	J. Yeager	ES80
JKF	GRU	8:12	15:50	T. Cruise	A380

Example - Query Model

- Optimized for full-data retrieval based on ID
  - Give full Itinerary overview for PNR BGTR4

Itinerary Table

PNR	ItineraryData
BGTR4	{“passengerName”:”John Doe”, “legs” : [{“origin”: “AMS”, “dest…
YTFE4	{“passengerName”:”Mary Joe”, “legs” : [{“origin”: “SLC”, “dest…
POGH2	{“passengerName”:”Steven May”, “legs” : [{“origin”: “GRU”, “d…

Storage technology selection

- Use the storage that fits the method of access
    - Generic Query → Relational DataBase
    - Relationships → Graph Database
    - Full-text search → Search Engine
    - Etc.
- Do not create a single model that can answer all queries.
  It will answer none efficiently.
- Do not fear (data) duplication

Query Handler

- Responsible for handling queries
- Exposes relevant information from query models

Query Handling in Axon Framework

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

`@QueryHandler`

- On (singleton) component

```
@QueryHandler
public Itinerary handle(FindItinerary 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
    }
}
```

Query Handler parameters

Supported parameter types

- First parameter (if none of below) resolves to Message payload
- `Message` → Resolves to entire message
- `QueryMessage` → Resolves to `QueryMessage`
- `UnitOfWork` → Resolves to the current Unit of Work
- `MetaData` → Resolves to the `MetaData` of the `Message`
- `@MetaDataValue (“name”) …` → Resolves to a Meta Data value of the Message
- Any Spring bean or component registered using Configuration API

Custom values using ParameterResolverFactory

Dispatching Queries

- Directly on QueryBus:

```
QueryBus queryBus;
queryBus.query(new GenericQueryMessage<>(
    myQueryPayload, ResponseTypes.instanceOf(ResponseType.class)
));
```

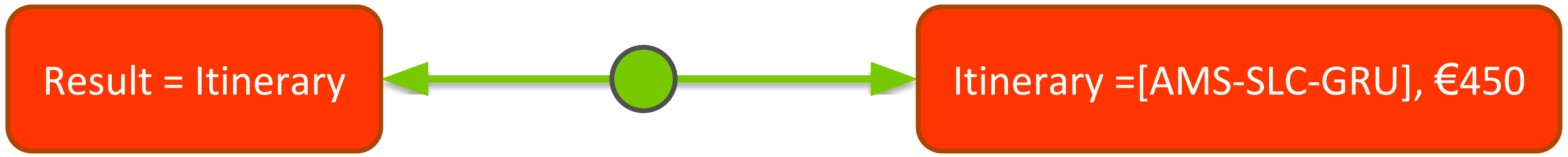
- Using Query Gateway

```
QueryGateway gateway = DefaultQueryGateway.builder().queryBus(queryBus).build();
// non-blocking, returns CompletableFuture<>
gateway.query(myQuery, ResponseTypes.instanceOf(ResponseType.class));
```

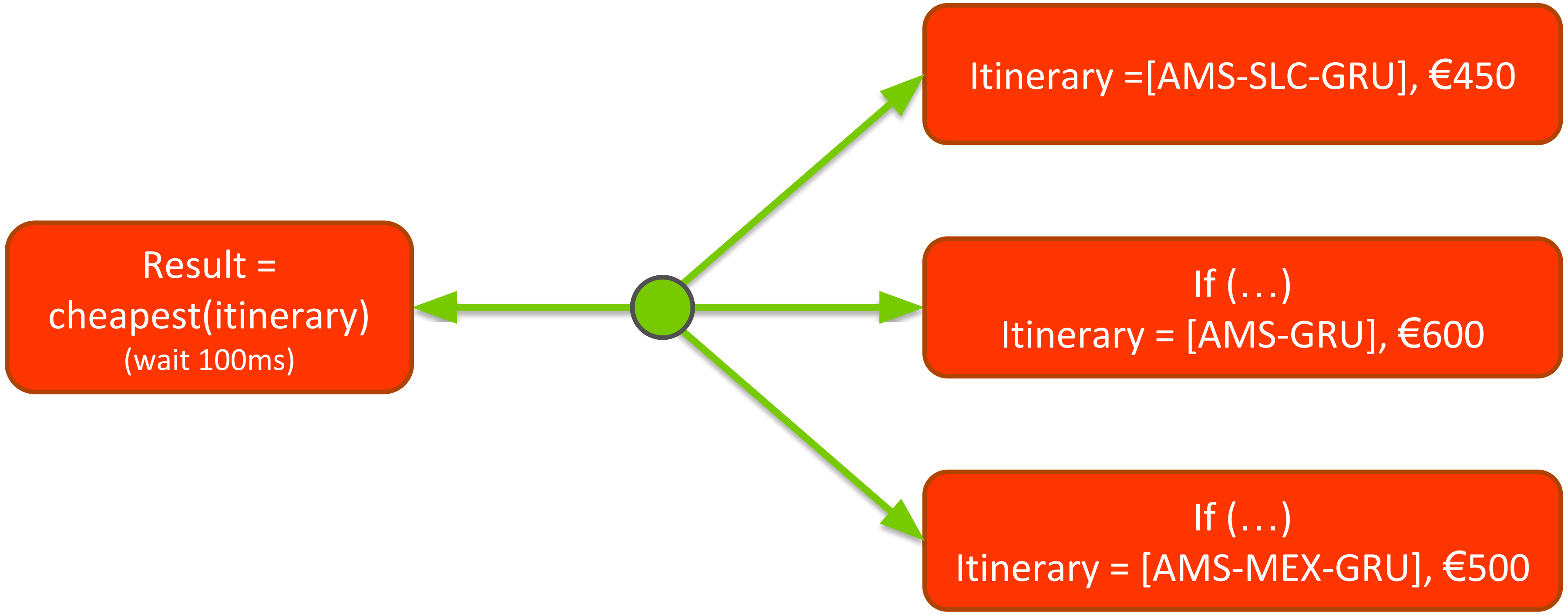
Types of Queries

- Direct query
  - Single destination, single reply
- Scatter Gather
  - Published to all relevant destinations, multiple replies
  - Reduce function to reduce replies to single response
- Subscription
  - Single destination for initial result
  - Real-time updates

Query – Direct query

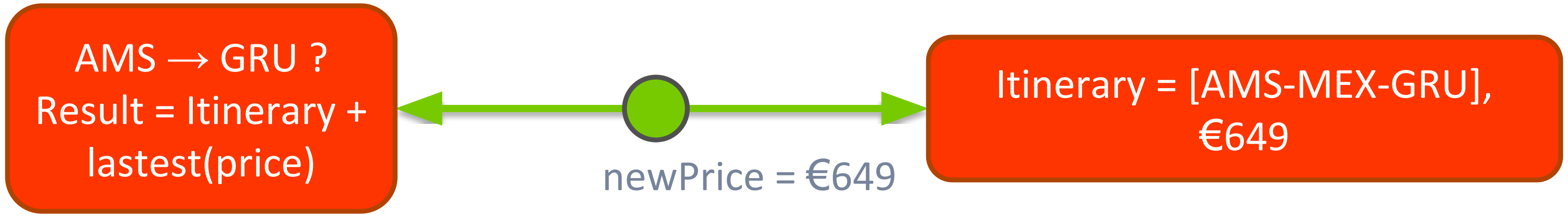


Query – Scatter-Gather


Itinerary =[AMS-SLC-GRU], €450
Result = cheapest(itinerary) (wait 100ms)
If (…) Itinerary = [AMS-GRU], €600
If (…) Itinerary = [AMS-MEX-GRU], €500

AxonIQ

Query – Subscription


AMS → GRU ?
Result = Itinerary + lastest(price)
newPrice = €649
Itinerary = [AMS-MEX-GRU], €649

Subscription queries

```
@Component
public class FlightStatusProjection {

  private QueryUpdateEmitter emitter;

  @QueryHandler
  public FlightStatus handle(GetFlightStatus query) {
    // find that data and return it
  }

  @EventHandler
  public void handle(FlightStatusUpdated event) {
    // update the projection
    emitter.emit(GetFlightStatus.class,
                 query -> event.flightId().equals(query.flightId()),
                 new FlightStatusUpdate());
  }
}
```

Whatever else you wanted to know…

Questions

AxonIQ