TRIFORK.

CQRS with Axon Framework

An introduction to scalable architectures

Allard Buijze – allard.buijze@trifork.nl

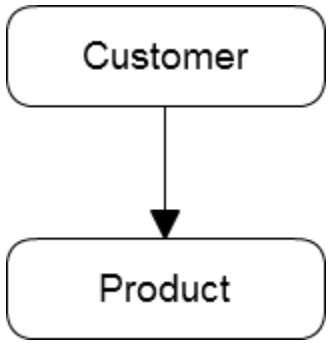
Allard Buijze

- Software Architect at Trifork
  - Formerly known as Orange11 / JTeam
  - Organizers of GOTO Amsterdam

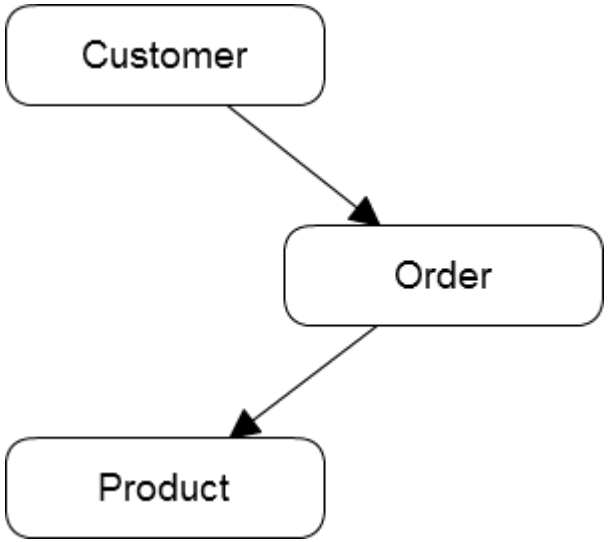
- ~15 years of web development experience
- Strong believer in DDD and CQRS
- Developer and initiator of Axon Framework
  - Java Framework for scalability and performance
  - www.axonframework.org

TRIFORK.

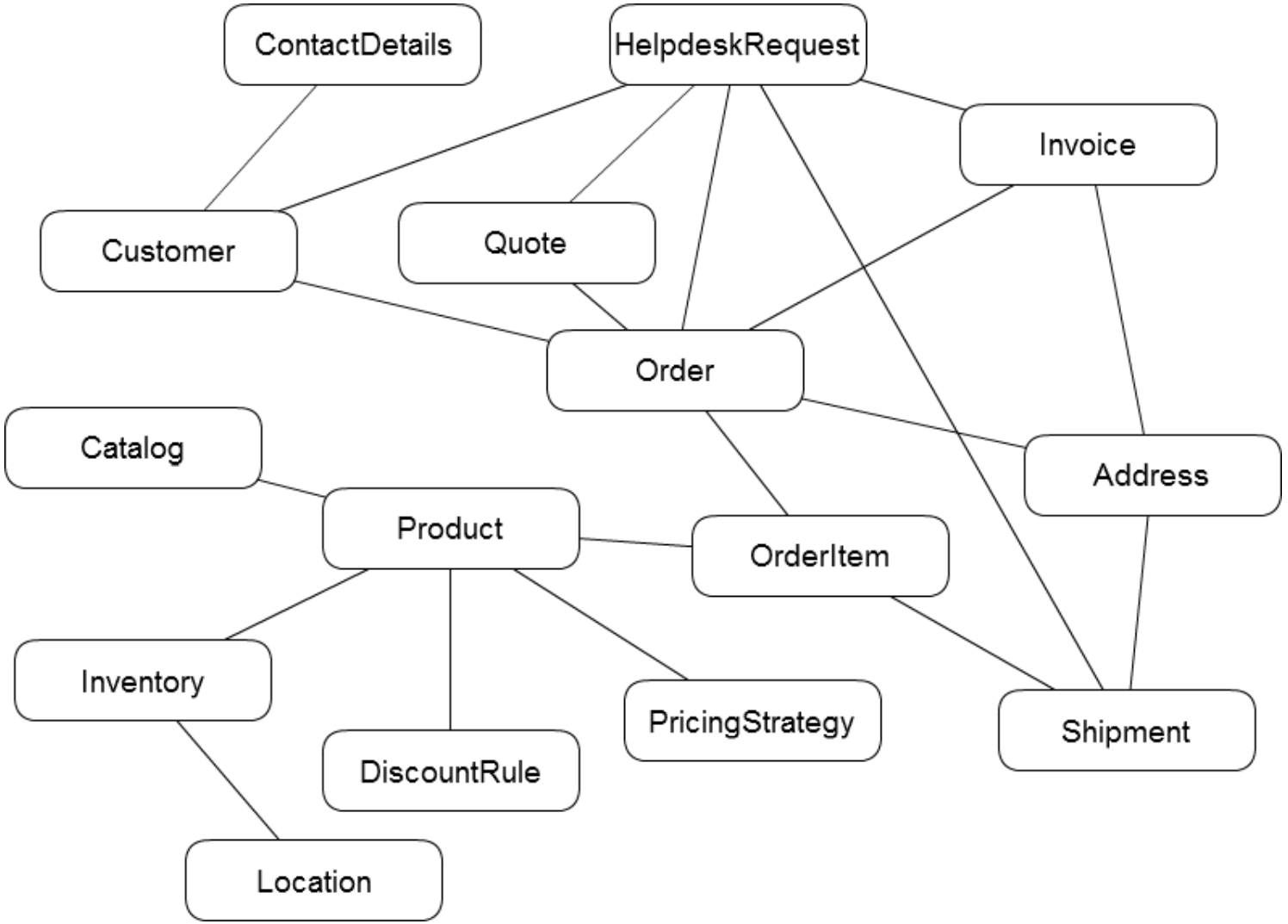
Evolution of Software Complexity



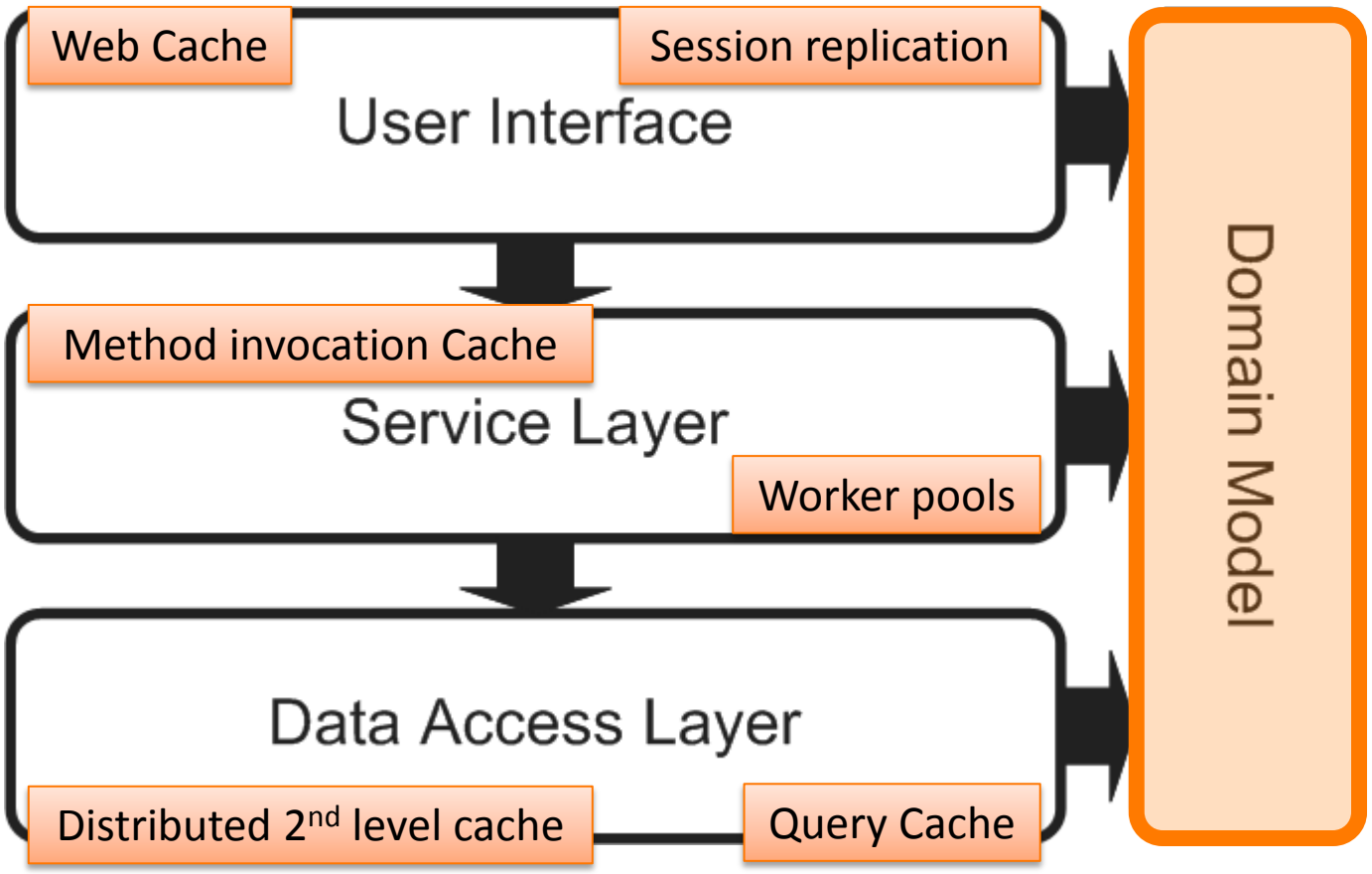
Evolution of Software Complexity



Evolution of Software Complexity



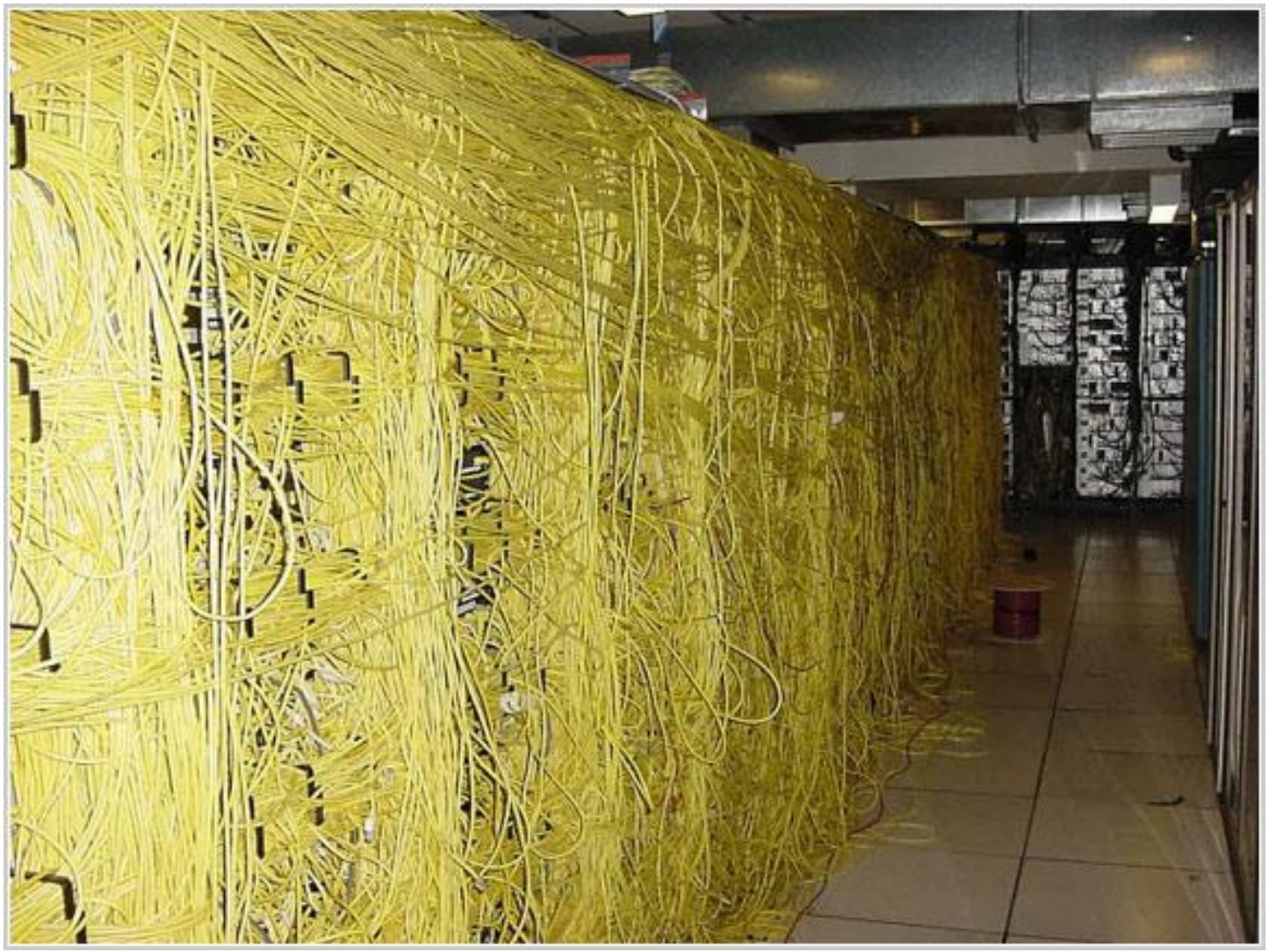
Layered architecture



Designed for … ?

TRIFORK.

Technical complexity

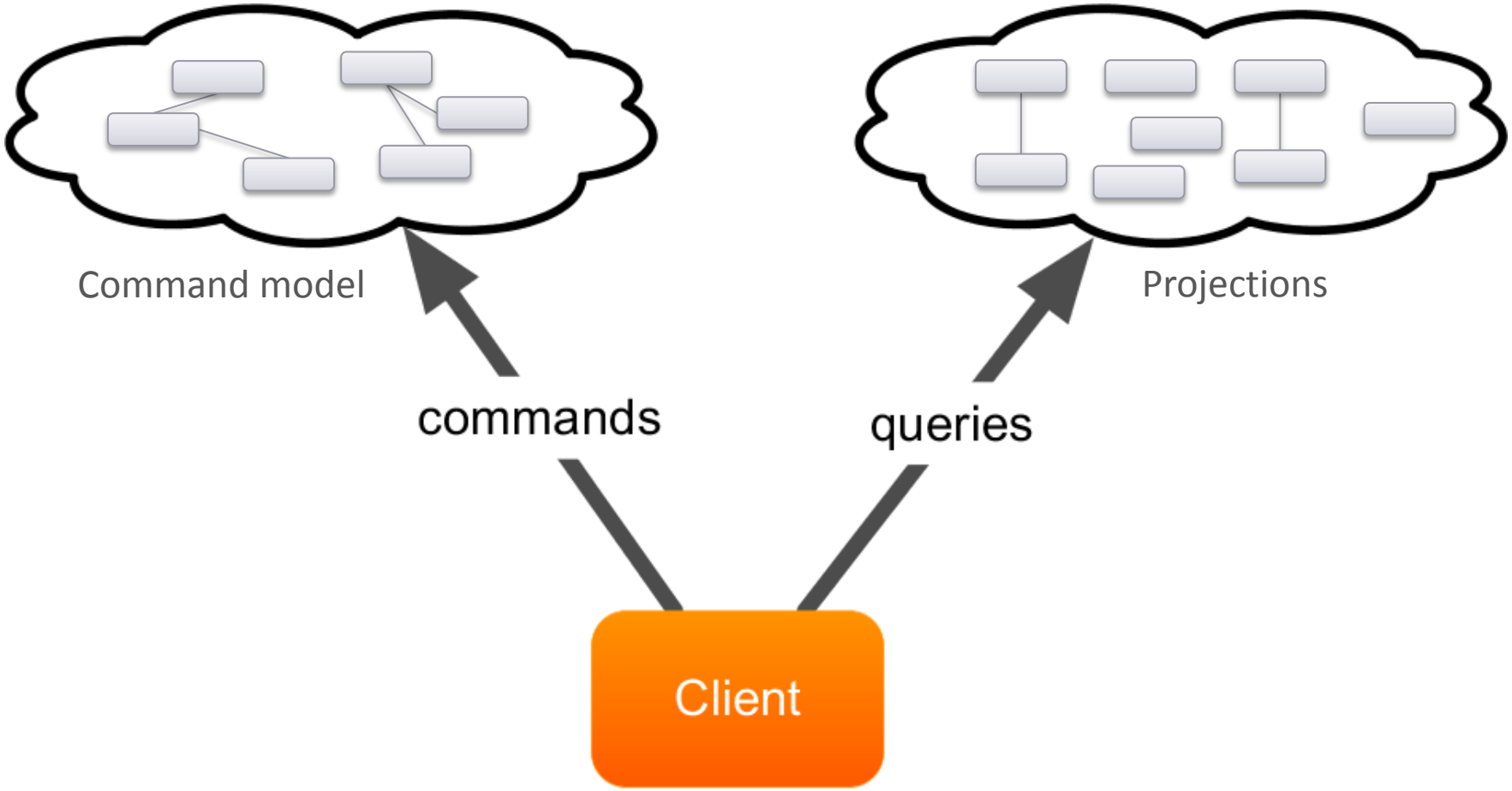


http://royal.pingdom.com/2008/01/09/the-worst-cable-mess-ever/

CQRS – Did you mean cars?

Command – Query Responsibility Segregation

And then there was CQRS

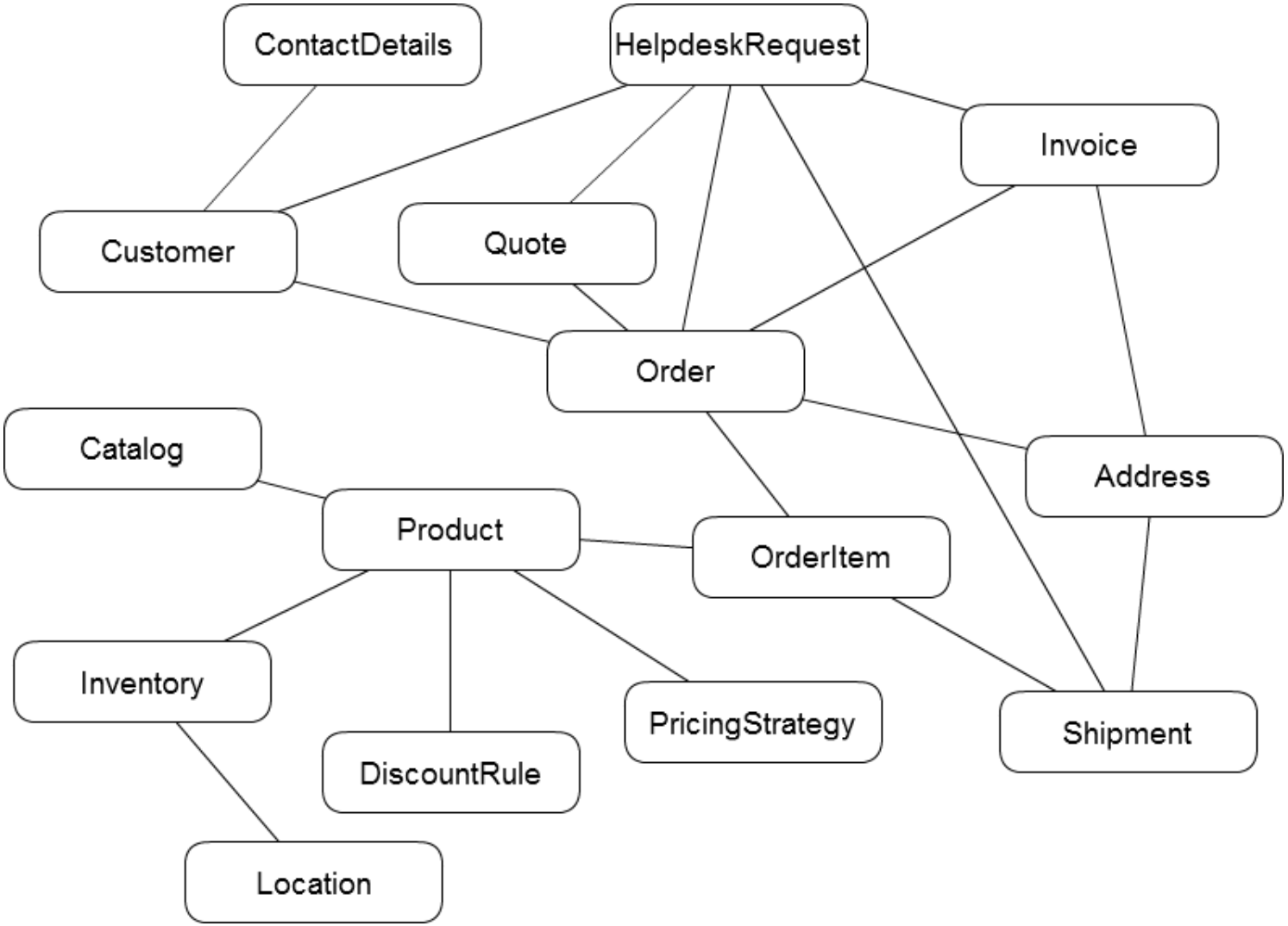


Complexity

CQRS as a weapon in the battle against complexity

TRIFORK.

Our domain model



Find products ordered by customer

```
SELECT …

FROM Customer

    LEFT JOIN Order …

    LEFT JOIN OrderLine …

    LEFT JOIN Product …

WHERE Customer.id = :customerId
```

WHOOPS…

```
    AND Order.status = “Accepted”
```

Update a user’s address

UPDATE Customer

SET Address = :newAddress

WHERE Customer.id = :customerId

WHOOPS…

What about the invoices and orders?

Let’s update a product price

```
UPDATE Product

SET Price = :newPrice

WHERE Product.id = :productId
```

WHOOPS…

What about the invoices and orders?

The source of complexity

- Order Pickers
  - Which products do I need to fetch from the warehouse?
- Finance
  - Which invoices are overdue?
- Management
  - What is our revenue this year?
- Customer
  - Where is my order?

Real life example

```
private static final String PLAYER_COCKPIT_WATERFALL_ITEMS_QUERY =
        "(" +
            "select id, " + EntityType.NEWS_ITEM.ordinal() + " as entity_type, publish_date as sort_date " +
            "from news_item " +
            "where active = true and (" +
                "poster_player_id = :playerId " +
                "or poster_player_id in (" +
                  "select destination_friend_id from friendship where origin_friend_id = :playerId " +
                ") " +
                "or project_id in (" +
                  "select distinct project_id " +
                  "from donation " +
                  "where donor_participant_id = :playerId and status = 'OK'" +
                ")" +
                "or partner_id in (" +
                  "select project_id from ambassador_project where player_id = :playerId " +
                "))" +
        ") union all (" +
            "select id, " + EntityType.DONATION.ordinal() + " as entity_type, approval_date as sort_date " +
            "from donation " +
            "where status = 'OK' and (" +
                "donor_participant_id = :playerId " +
                "or donor_participant_id in (" +
                  "select destination_friend_id from friendship where origin_friend_id = :playerId" +
                ")" +
                "or raised_via_player_id = :playerId " +
                "or raised_via_player_id in (" +
                  "select destination_friend_id from friendship where origin_friend_id = :playerId" +
                ") " +
            ") " +
        ") union all ( " +
            "select id, " + EntityType.FRIENDSHIP.ordinal() + " as entity_type, created as sort_date " +
            "from friendship " +
            "where origin_friend_id = :playerId or (origin_friend_id in ( " +
                "select destination_friend_id from friendship where origin_friend_id = :playerId " +
            ") and destination_friend_id <> :playerId)" +
        ") ";
```

SELECT

NEWS_ITEM

or project in (…

status = ‘OK’

or project in (…

UNION ALL

DONATION

status = ‘OK’

or raised_via_player in (…

UNION ALL

FRIENDSHIP

```
SELECT *
FROM waterfall_items
WHERE relevant_to = :user_id
ORDER BY timestamp DESC
LIMIT :num_items
```

TRIFORK.

Decoupling domain models

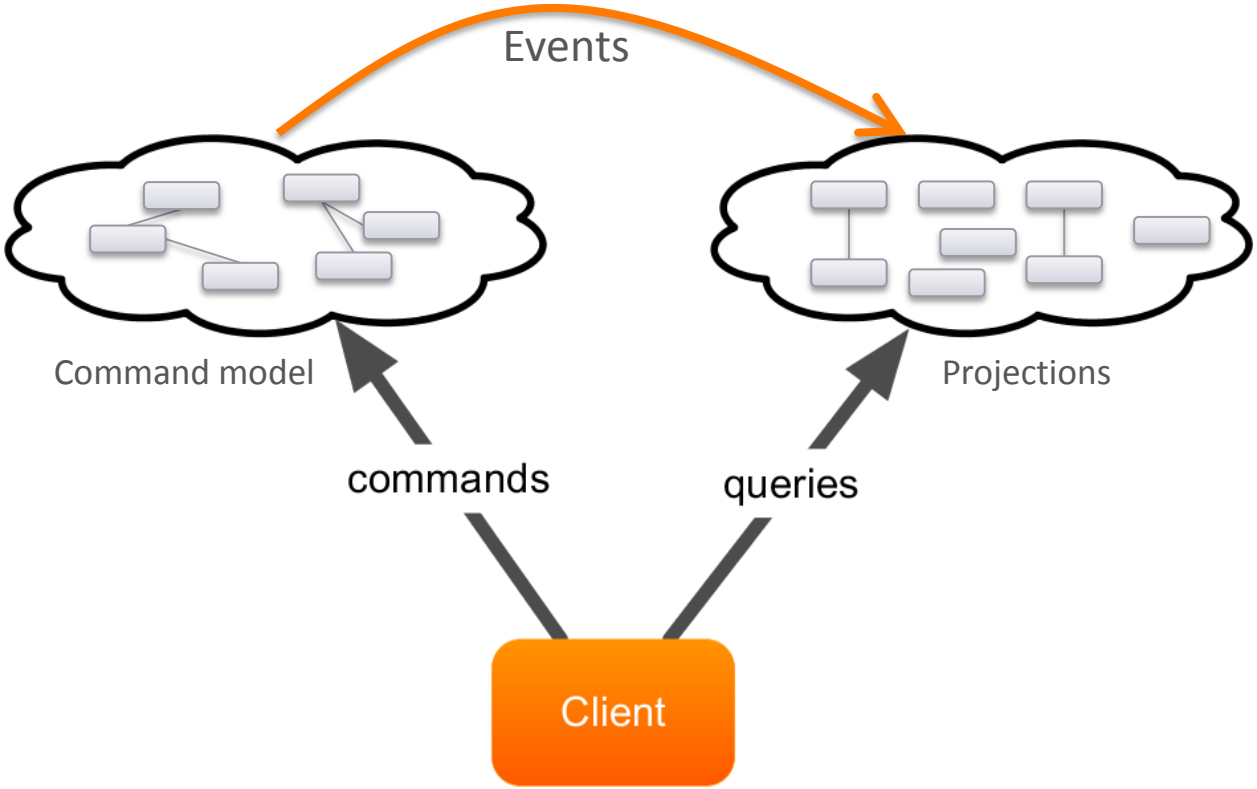
- Optimize each model for your exact needs
  - Updating
  - Operational information
  - Management information
- Clearly defined API
  - Commands
  - Events
- Keep the models in sync using Events

Event:

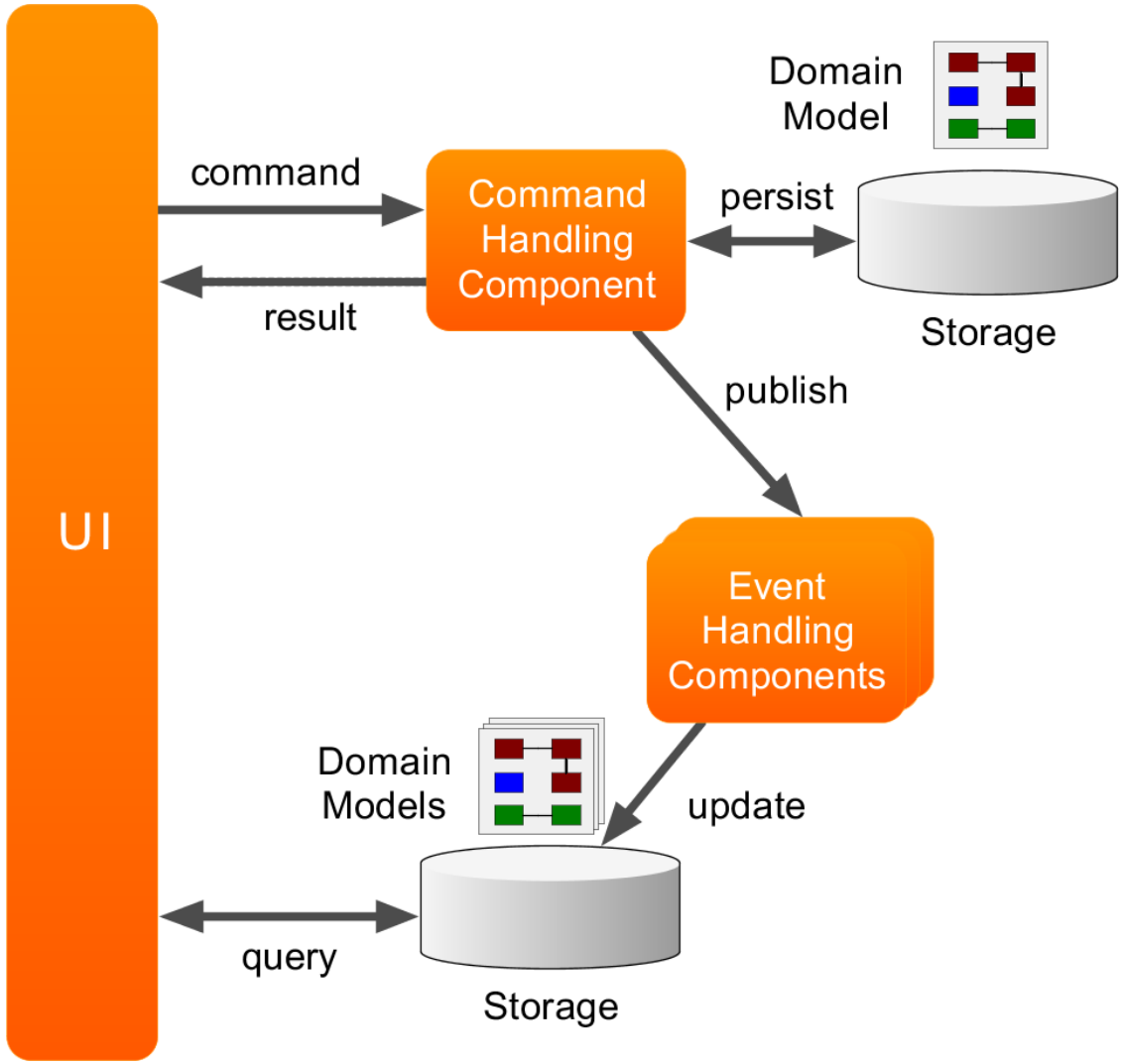
Notification that something relevant has happened within the domain

CQRS and EDA – Basics

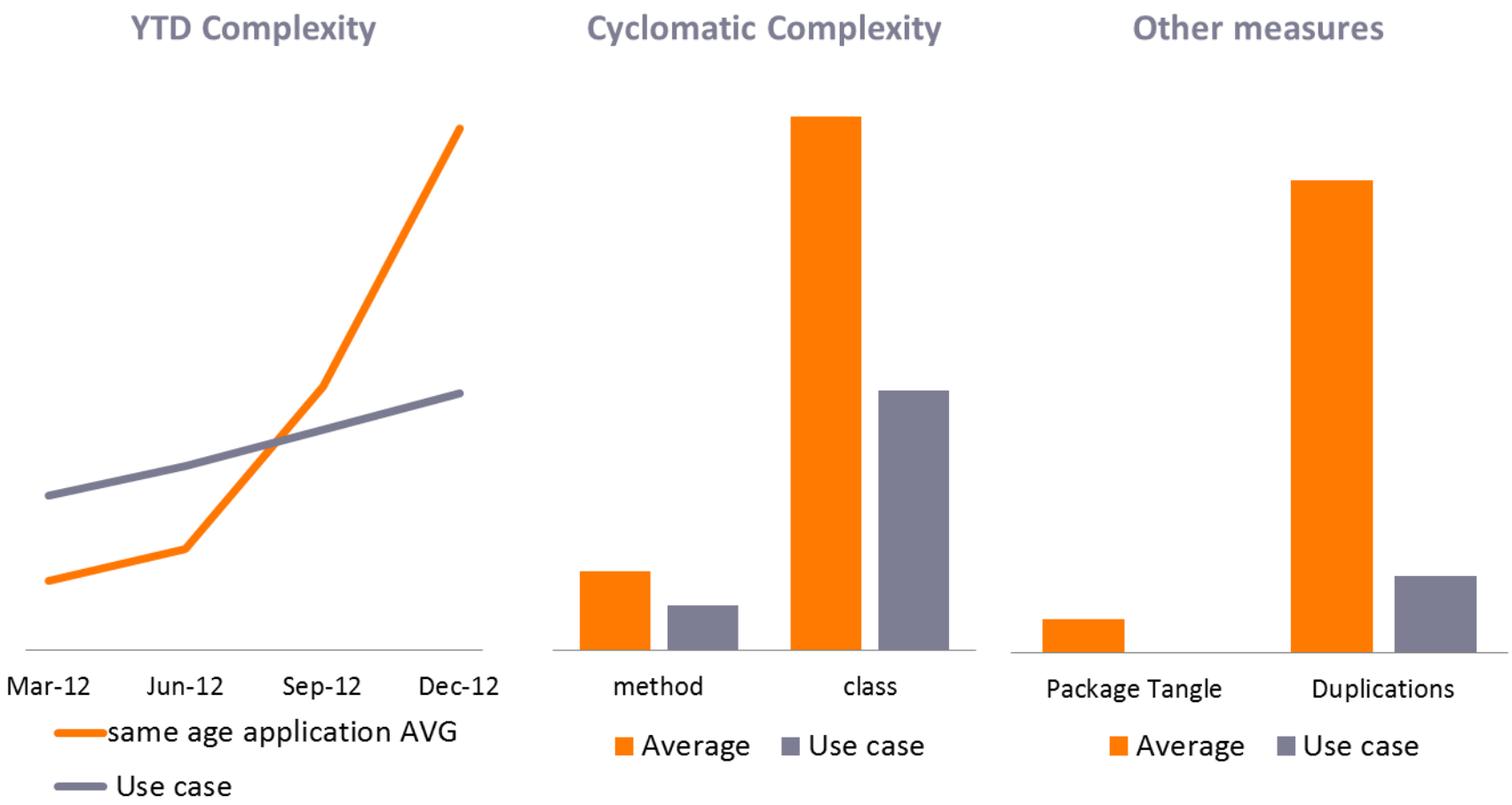
- EDA: Emit an event when something important has happened
- CQRS + EDA: Raise an event for each change in the command model



CQRS and EDA – Architecture



Complexity measures



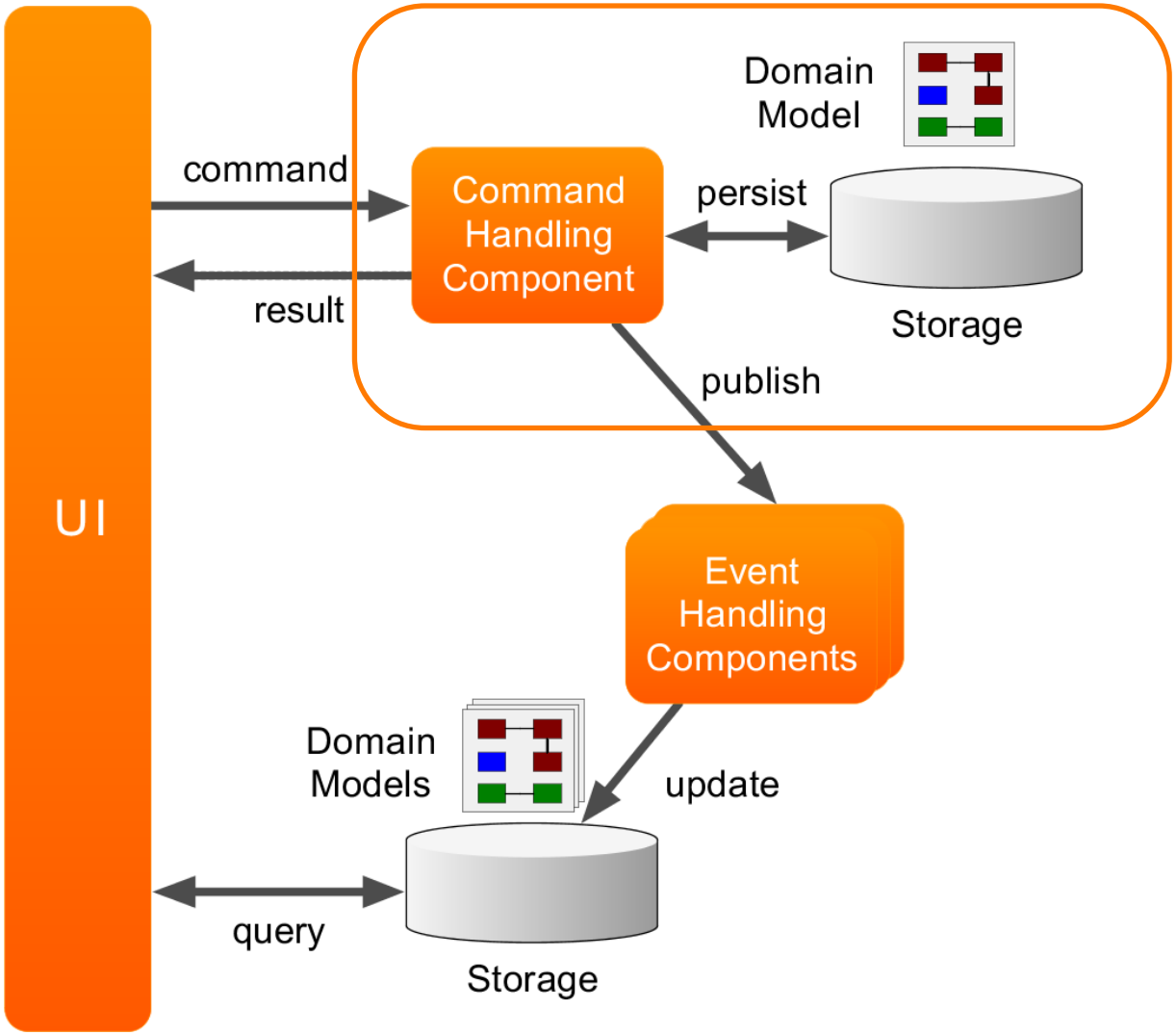
Source: Axon case study presentation by Aktive Reply srl

Event Sourcing

The business value of history

TRIFORK.

Event Sourcing



Event Sourcing

- Don’t store state, store history

- State Storage

Order

- id: 123
- items
  - 1x Deluxe Chair - € 399
- status: return shipment rcvd

- Event Sourcing

1. OrderCreatedEvent
   - id: 123
2. ItemAddedEvent
   - 2x Deluxe Chair - € 399
3. ItemRemovedEvent
   - 1x Deluxe Chair - € 399
4. OrderConfirmed
5. OrderShipped
6. OrderCancelledByUserEvent
7. ReturnShipmentReceived

Event Sourcing or not…

- Data staleness…
  - *User makes a change. A big one.*
  - *System:* Sorry, someone else has made a change too. Try again…

With Event Sourcing, the system knows what the other user did,

and can try to merge the changes

Event Sourcing

- Reporting…
  - *Manager:* I need to know how long it takes us to process an incoming order
  - *Developer:* we're not recording that right now. We'll build it now, deploy it in 2 months, and you'll have reliable reports 3 months after.

Event Sourcing: Data from day 1 is there. Simply analyze past events…

Event Sourcing as Business Case

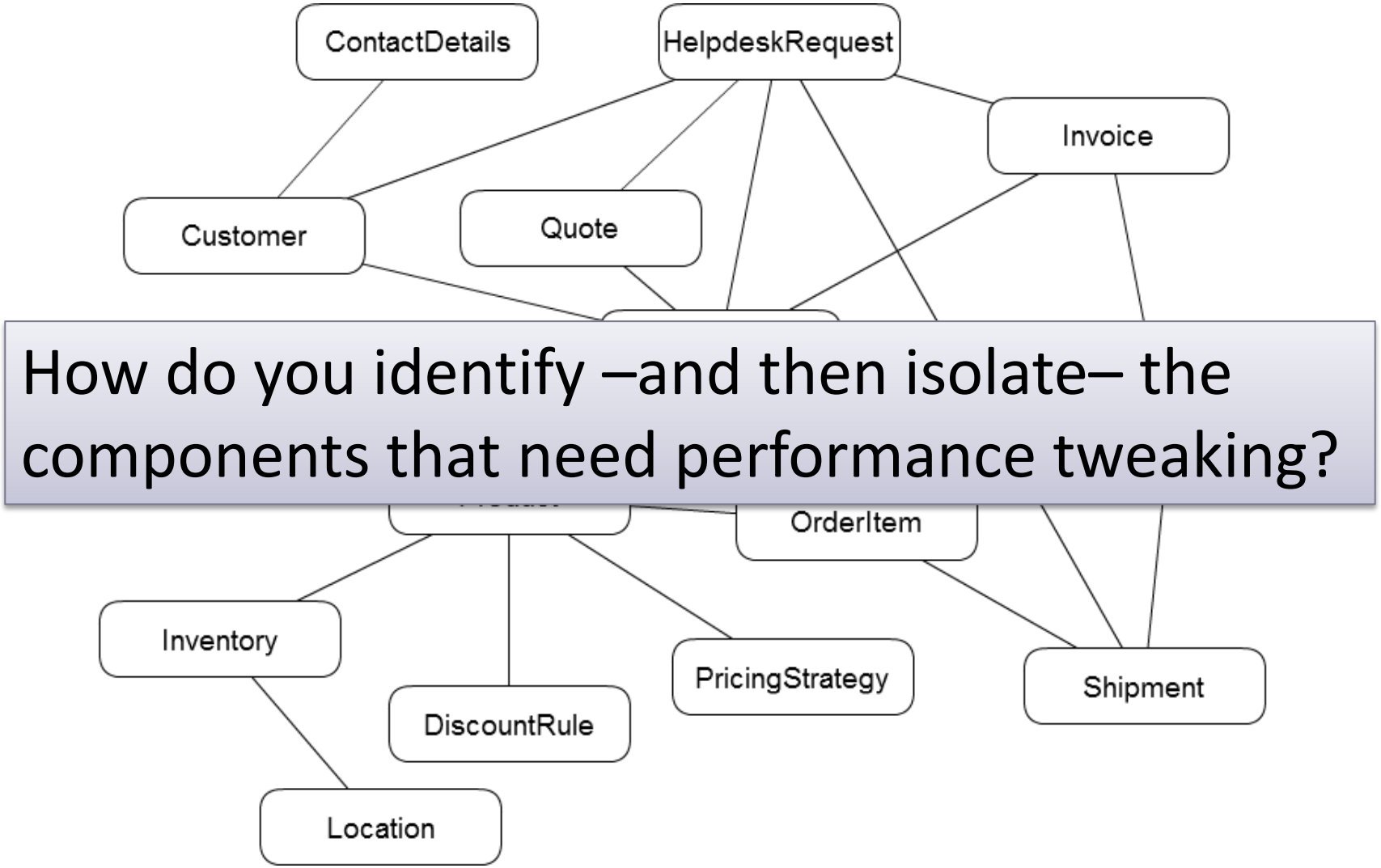
- Event Sourcing has less information loss
  - Event Store contains information that can be used in different ways in the future
- Event Store is a reliable audit log
  - Not only state, but also how it is reached.
- Event Sourcing increases performance
  - Only deltas need to be stored.
  - Caches prevent reads.

Performance

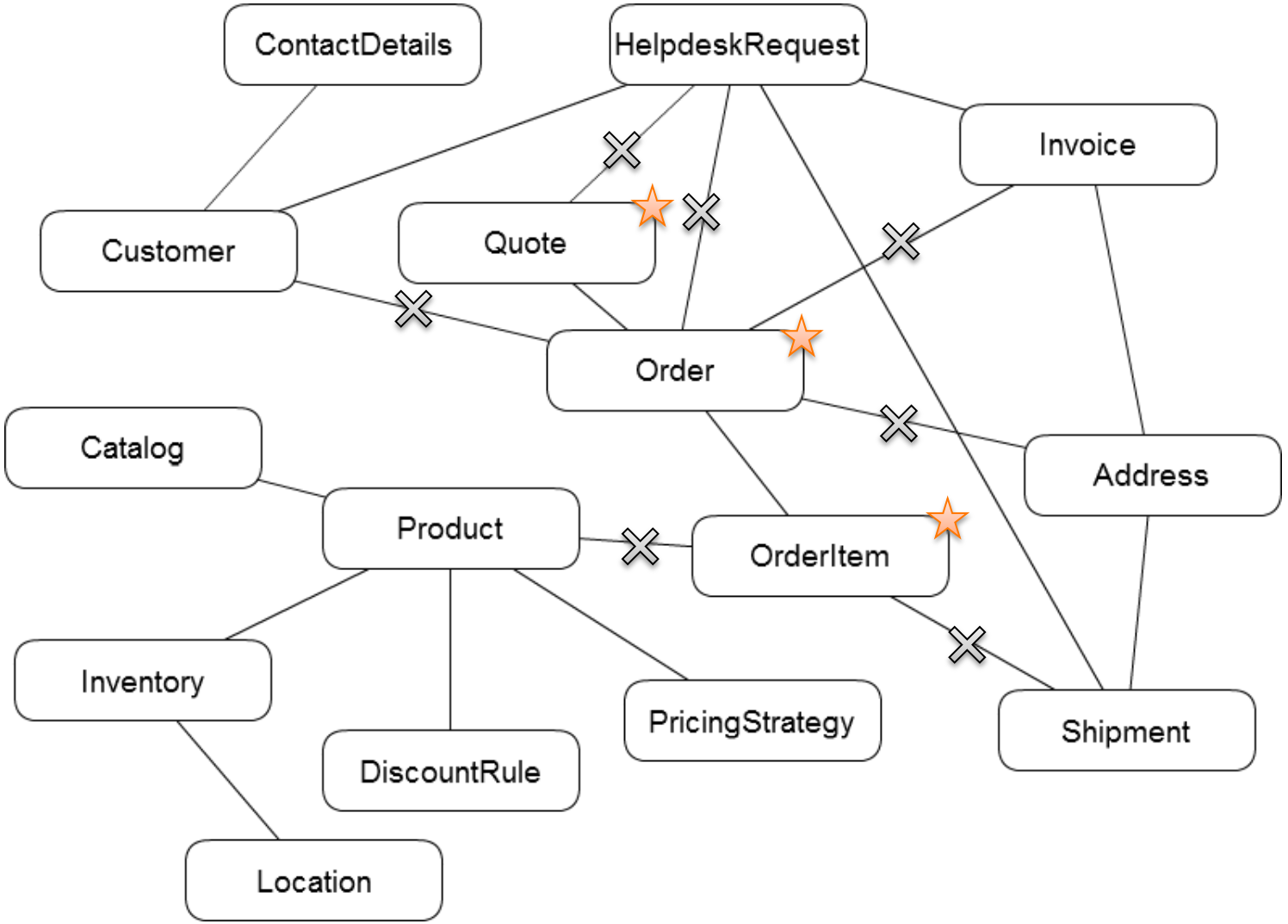
Speed through simplicity

TRIFORK.

Isolating performance bottlenecks



Isolating performance bottlenecks…



Addressing Performance

- Separation of command and query components
  - Focus on the performance where it's needed
  - Driven by non-functional requirements

- Non-functional requirements (example)
  - Queries: 100.000 /s, max 100ms
  - Updates: 100 /s, max 500ms

Command performance

- Optimize model to make quick decisions
  - Keep required data in-memory
  - Local cache
- Use Event Sourcing
  - Appending small amounts of data to Event Store
  - Execute Processing and Event Storage in parallel

Query Performance

- Store data as required
  - Simple queries run much faster
- Cache for often-accessed data
  - Use events to invalidate (or update) caches

Scalability

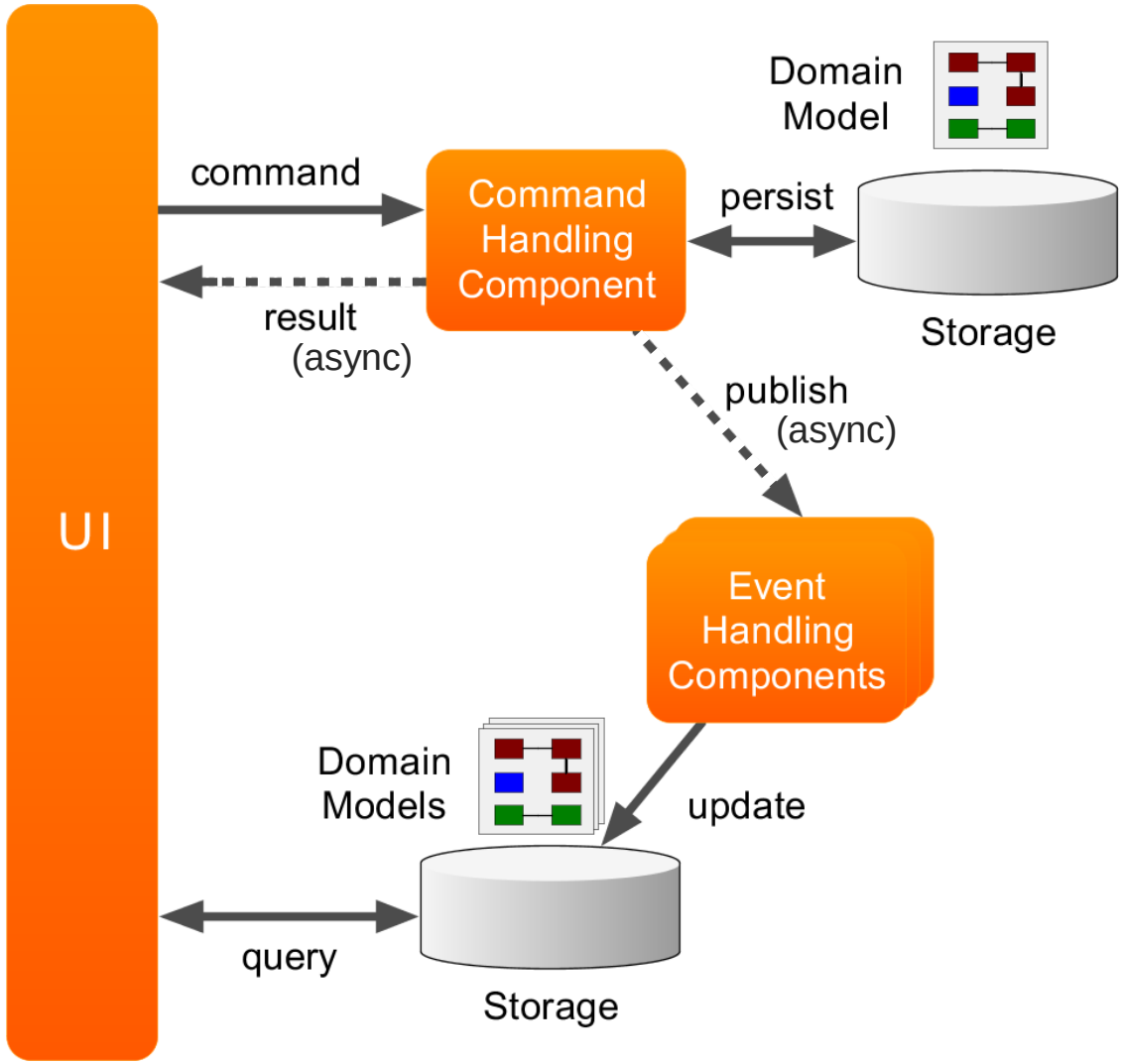
When requirements go beyond a single box

Before you scale…

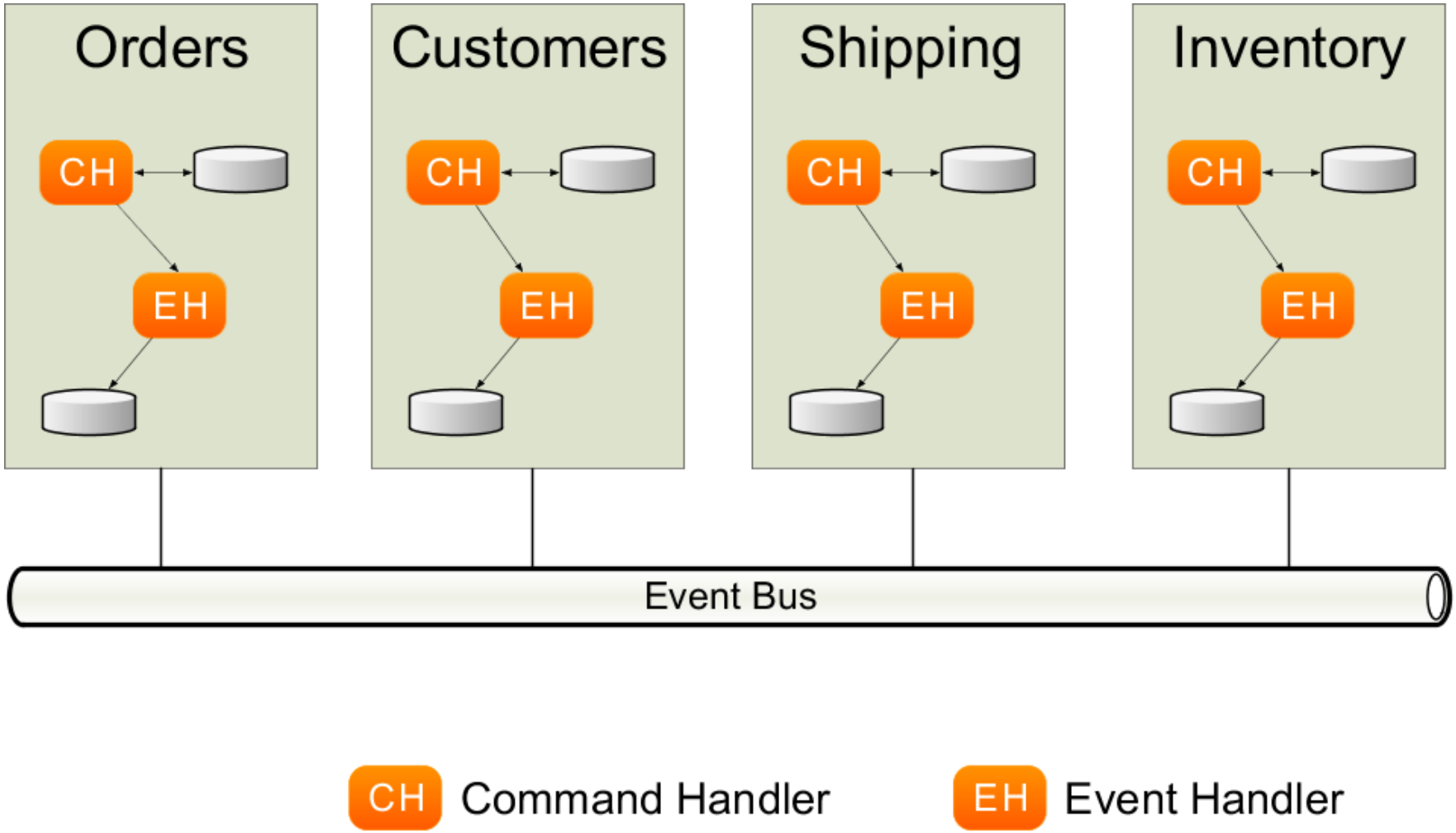
- Rule 1 of distributed systems: Don’t !!

- Scalability != Scaling
  - Scalability is about the ability to scale

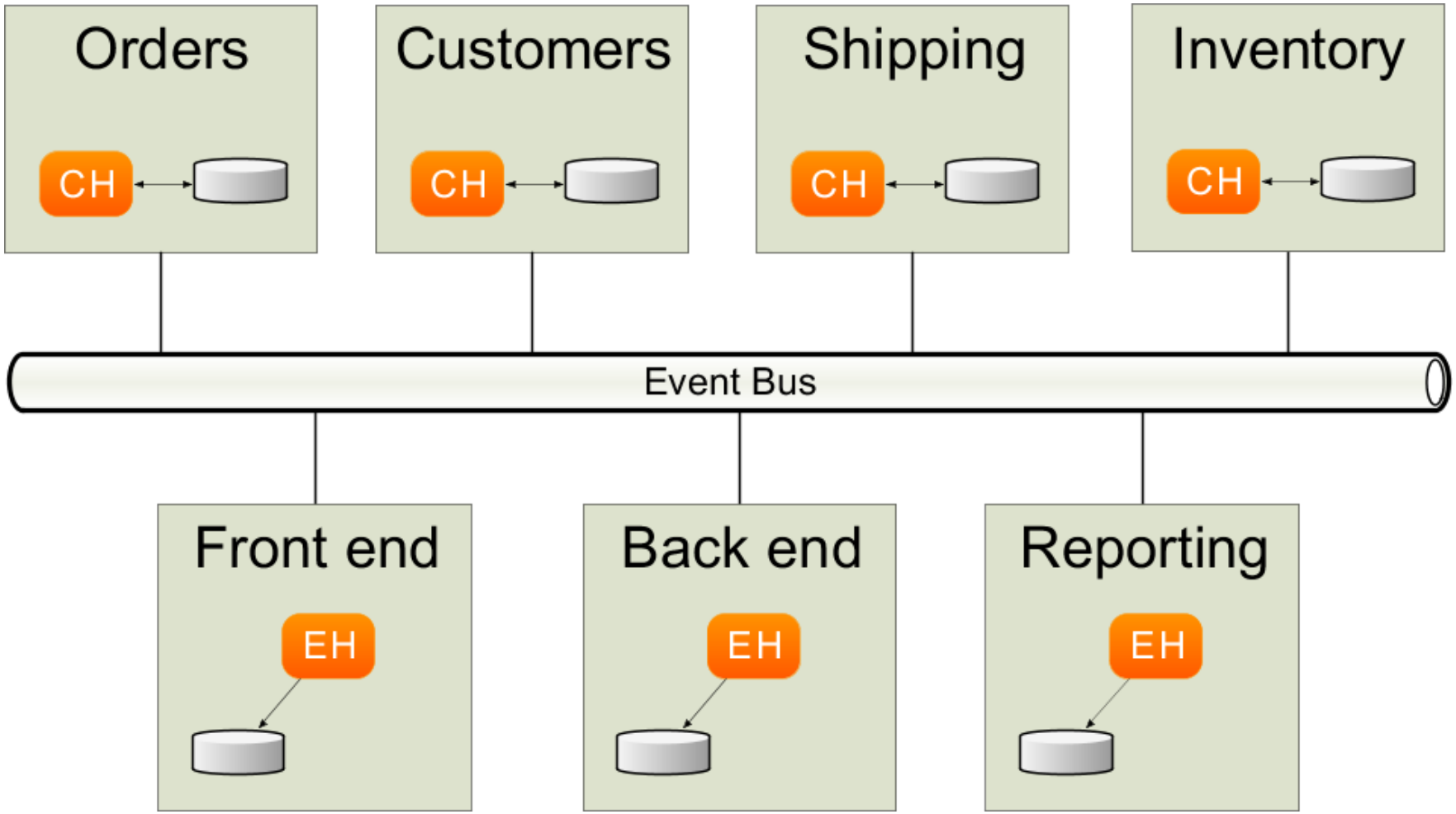
Scalability



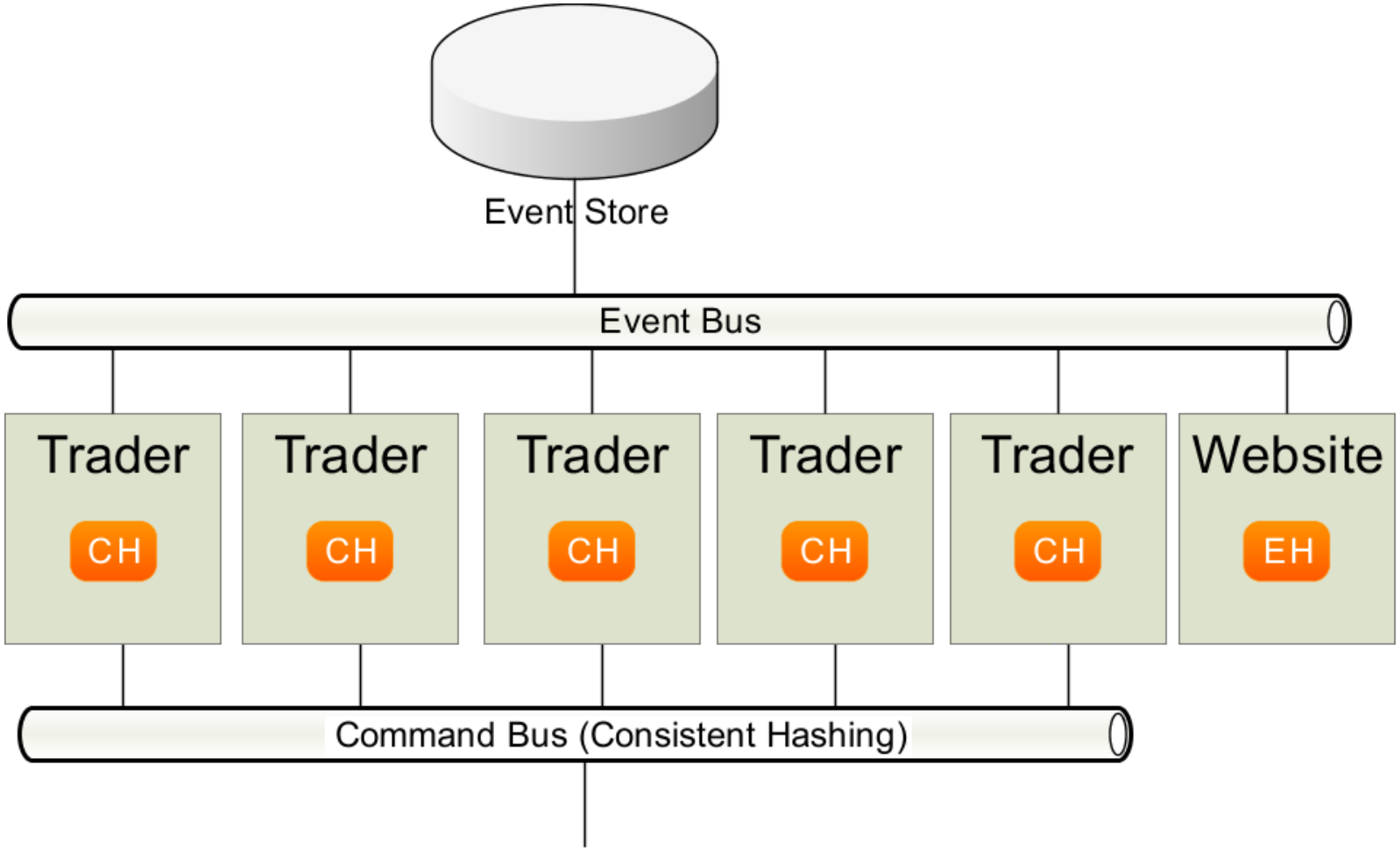
Scalability – Context Based



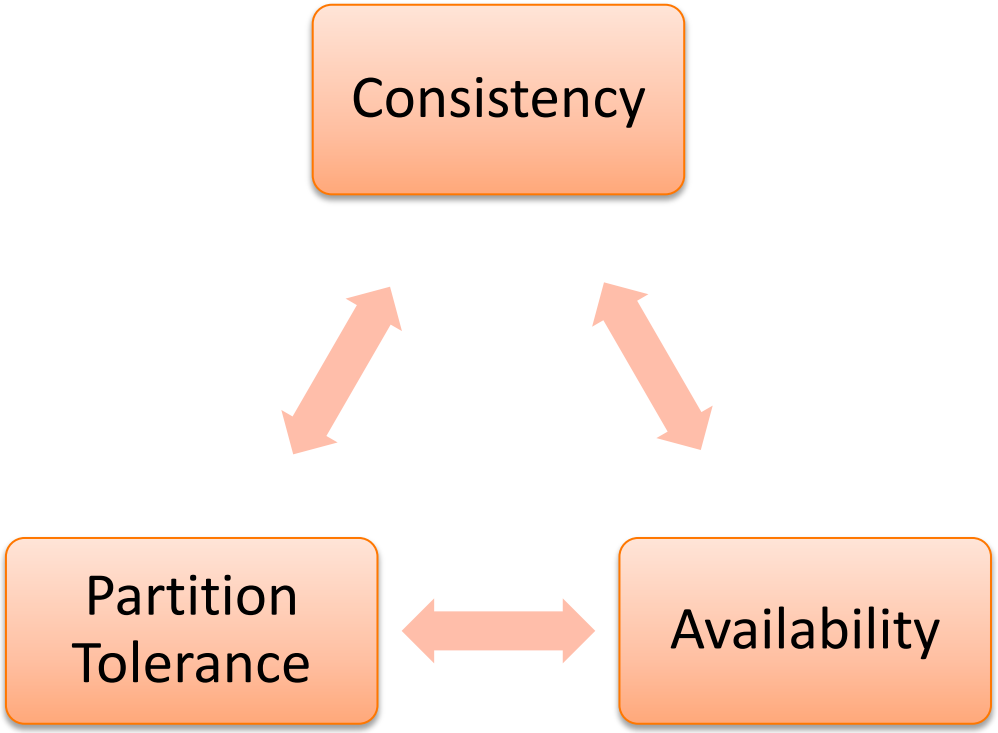
Scalability – Context and Audience Based



Scalability – High Volume Processing



Transactions in distributed systems – CAP Theorem



Transactions in CQRS

- Full consistency within context
  - Aggregate boundaries
  - Guarded by transactional execution of command
- Eventual consistency between contexts
  - Process managed by Sagas
  - Compensating actions on failure

Axon Framework

The fast track into CQRS

TRIFORK.

Proof of Concept – Mini webshop

- ❑ Browse products
- ❑ Place orders
- ❑ Update inventory
- ❑ Sales report
- ☑ Generic Infrastructure

32 days of development

1 year of development

2 days of development

0 Lines of Business logic code

Axon Framework

- Java framework for CQRS
  - Puts the CQRS theory to practice
  - Commercially backed Open Source project
- Provides the common CQRS building blocks
  - Command and Event Buses
  - Complex Transaction Management (Sagas)
  - Simple configuration of Event Listeners
- No need to worry about infrastructure components
  - Clear separation of business logic and infrastructure
  - Infrastructure through configuration

CQRS Components

- Command Bus
  - SimpleCommandBus → In-Process
  - AsyncCommandBus → In-Process Async
  - DisruptorCommandBus → High performance async
- Event Bus
  - SimpleEventBus → In-Process – Sequential
  - ClusteringEventBus → Configurable behavior

Event Sourcing in Axon Framework

Decision making

```
@CommandHandler
public void changeCreditLimit(UpdateCustomerCreditLimitCommand cmd) {
    if (cmd.getNewLimit() > currentLimit) {
        apply(new CreditLimitRaisedEvent(id, cmd.getNewLimit()));
    } else if (cmd.getNewLimit() < currentLimit) {
        apply(new CreditLimitLoweredEvent(id, cmd.getNewLimit()));
    }
}
```

State changes

```
@EventHandler
protected void onCreditLimitUpdated(CreditLimitUpdatedEvent event) {
    this.currentLimit = event.getNewLimit();
}
```

```
@EventHandler
protected void onCustomerCreatedEvent(CustomerCreatedEvent event) {
    this.id = event.getCustomerId();
    this.currentLimit = event.getInitialCreditLimit();
}
```

Creating an Event Handler

```
<axon:event-bus id="eventBus"/>

<axon:annotation-config/>
```

```
public class ToDoEventHandler {

    @EventHandler
    public void handle(ToDoItemCreatedEvent event, @Timestamp DateTime time) {
        System.out.println(String.format("We've got something to do: %s (%s, cr
                                         event.getDescription(),
                                         event.getTodoId(),
                                         time.toString("d-M-y H:m")));
    }

    @EventHandler
    public void handle(ToDoItemCompletedEvent event) {
        System.out.println(String.format("We've completed the task with id %s",
    }
}
```

TRIFORK.

Sagas

```
public class CustomerSaga {

    /* State required to process events */

    @StartSaga
    @SagaEventHandler(associationProperty = "customerId")
    public void handleCustomerCreated(CustomerCreatedEvent event) {
        /* some activity */
    }

    @SagaEventHandler(associationProperty = "customerId")
    public void handleLimitRaised(CreditLimitRaisedEvent event) {
        /* some activity */
    }

    /* ... */

    @EndSaga
    @SagaEventHandler(associationProperty = "customerId")
    public void handleCustomerRemoved(CustomerRemovedEvent event) {
        // this ends the life of this saga
    }
}
```

Declarative Testing

Given: a set of historic events

When: I send a command

Then: expect certain events

```
@Test
public void testCreditLimitRaised() {
    fixture.given(new CustomerCreatedEvent("myId", "John", 1000))
           .when(new UpdateCustomerCreditLimitCommand("myId", 0, 1100))
           .expectEvents(new CreditLimitRaisedEvent("myId", 1100));
}
```

Axon in production…

- Finance
  - Process automation in a top 50 bank
  - Trading engine for ETF (index trackers) trading
  - Pension fund calculations at a large bank
  - On-line payment processing
- Gaming
  - On-line bridge platform (bridgebig.com)
  - On-line casino (casumo.com)
- Healthcare
  - Electronic Medical Record for the Geriatric Healthcare
  - Tracking and Tracing of equipment for dental implants
- etc…

Using Axon in your project

- Download full package from website:
  - www.axonframework.org/download
- Maven

```
<dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-core</artifactId>
    <version>2.0</version>
</dependency>
```

- Build from sources
  - www.axonframework.org/sources/

The future of Axon

- New features
  - API Simplification
  - Point-to-point Command- and Event Busses
  - Distributed Saga Manager
  - Custom DSL for Event and Command declaration
  - More Event Store implementations
  - Exactly once command processing
  - Set validation
- Operations Tooling
  - Management and Monitoring

Questions?

More information:

http://www.axonframework.org

abu@trifork.com

TRIFORK.