TRIFORK.

High performance and scalable architectures

A practical introduction to CQRS and Axon Framework

Allard Buijze – allard.buijze@trifork.nl

Allard Buijze

- Software Architect at Trifork
  - Organizers of GOTO & QCON

- ~ 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.

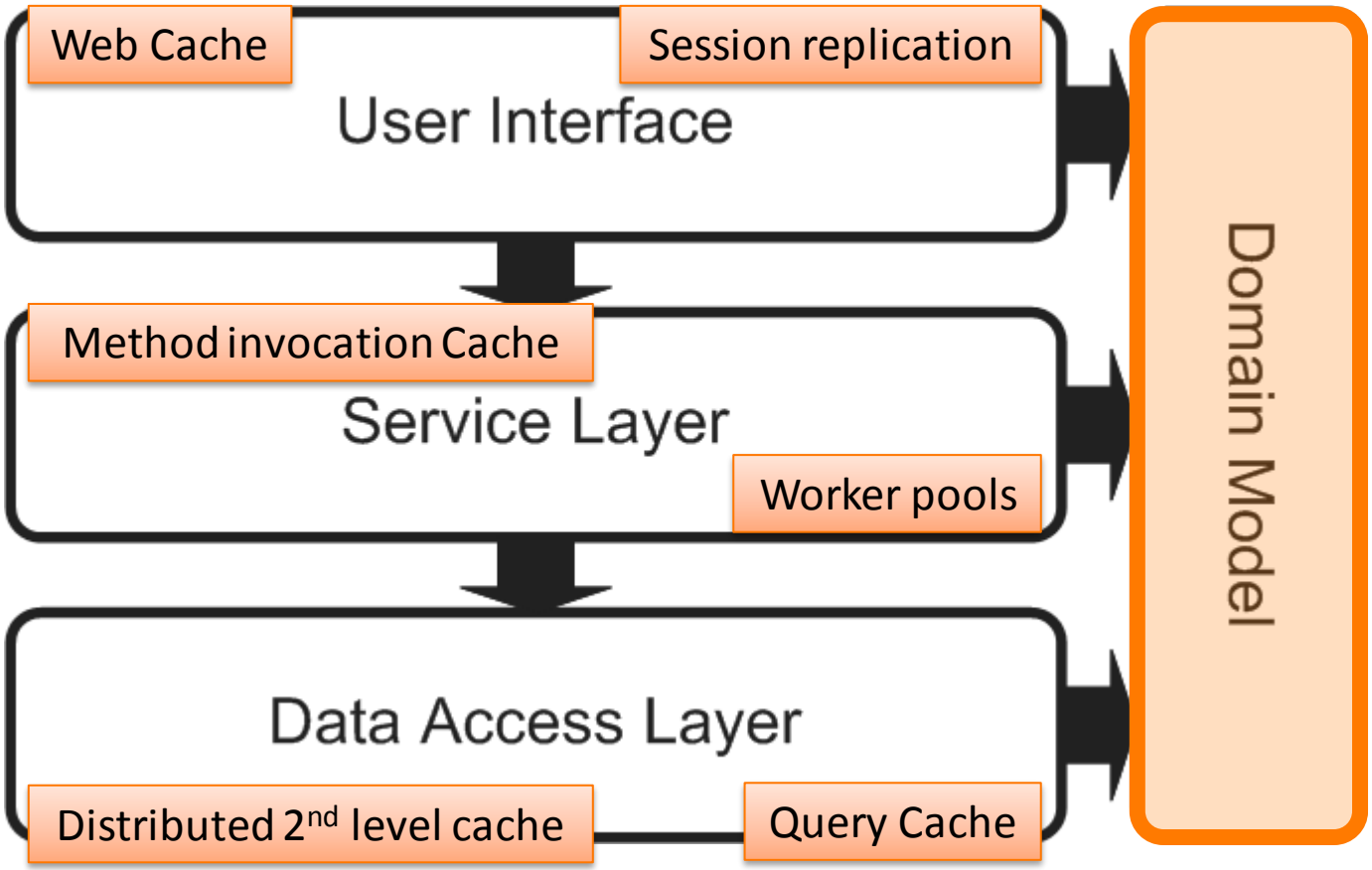
Designed for high performance

TRIFORK.

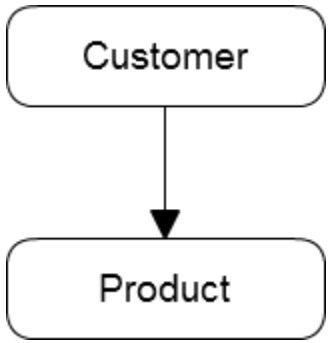
Designed for high performance (?)

TRIFORK.

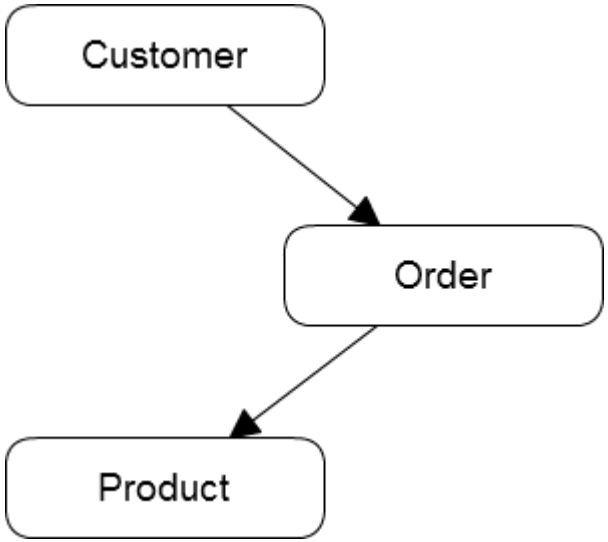
Layered architecture



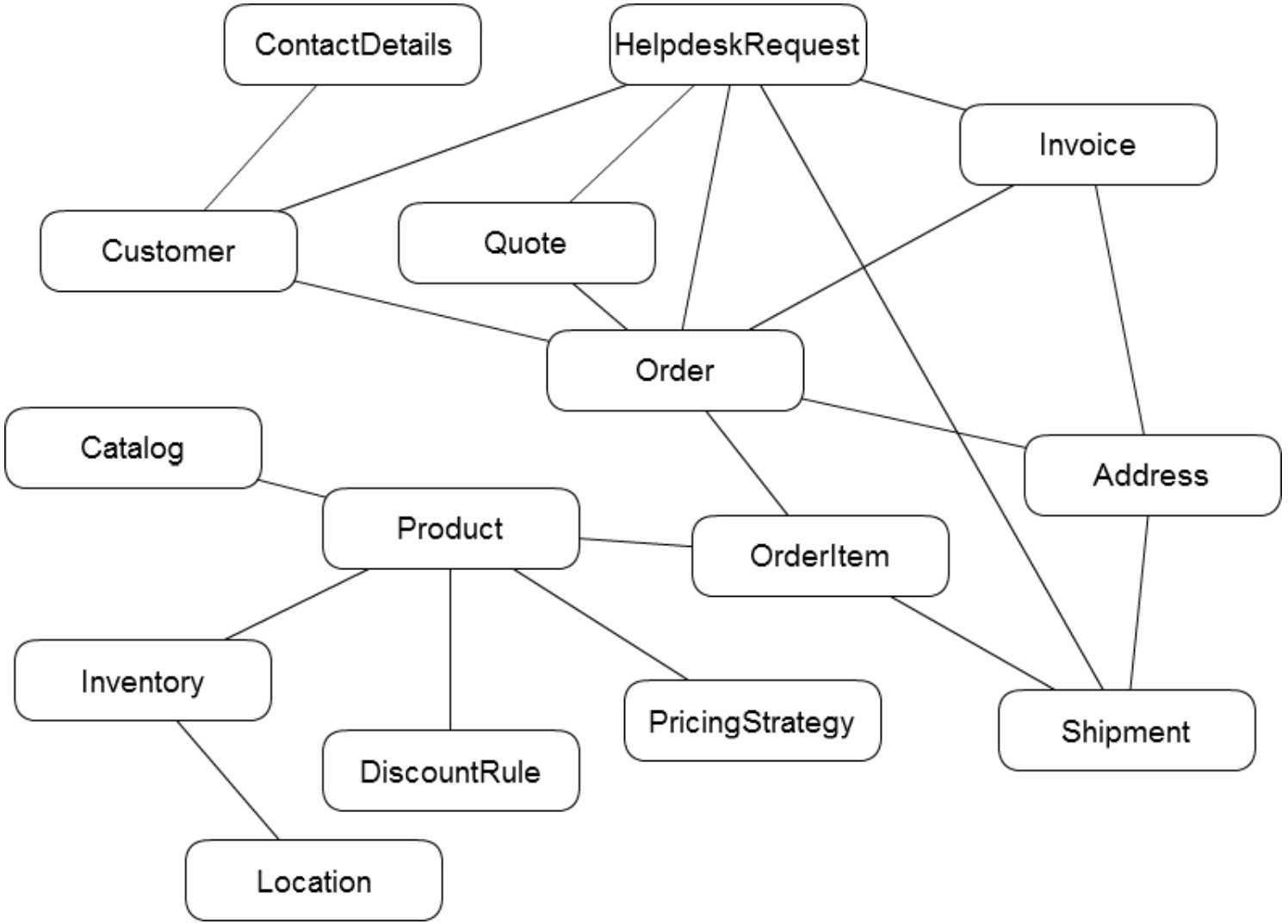
Evolution of a Domain Model



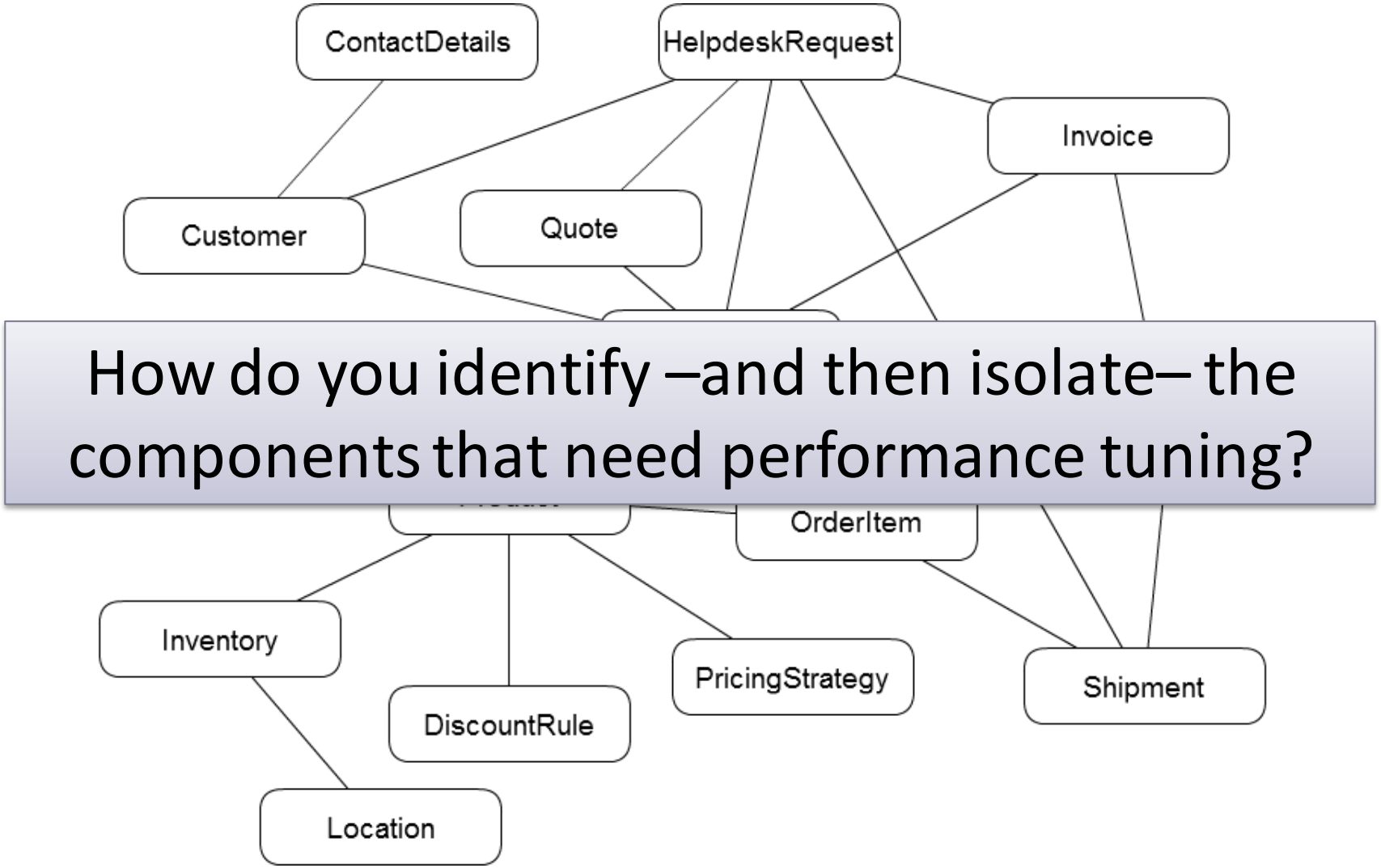
Evolution of a Domain Model



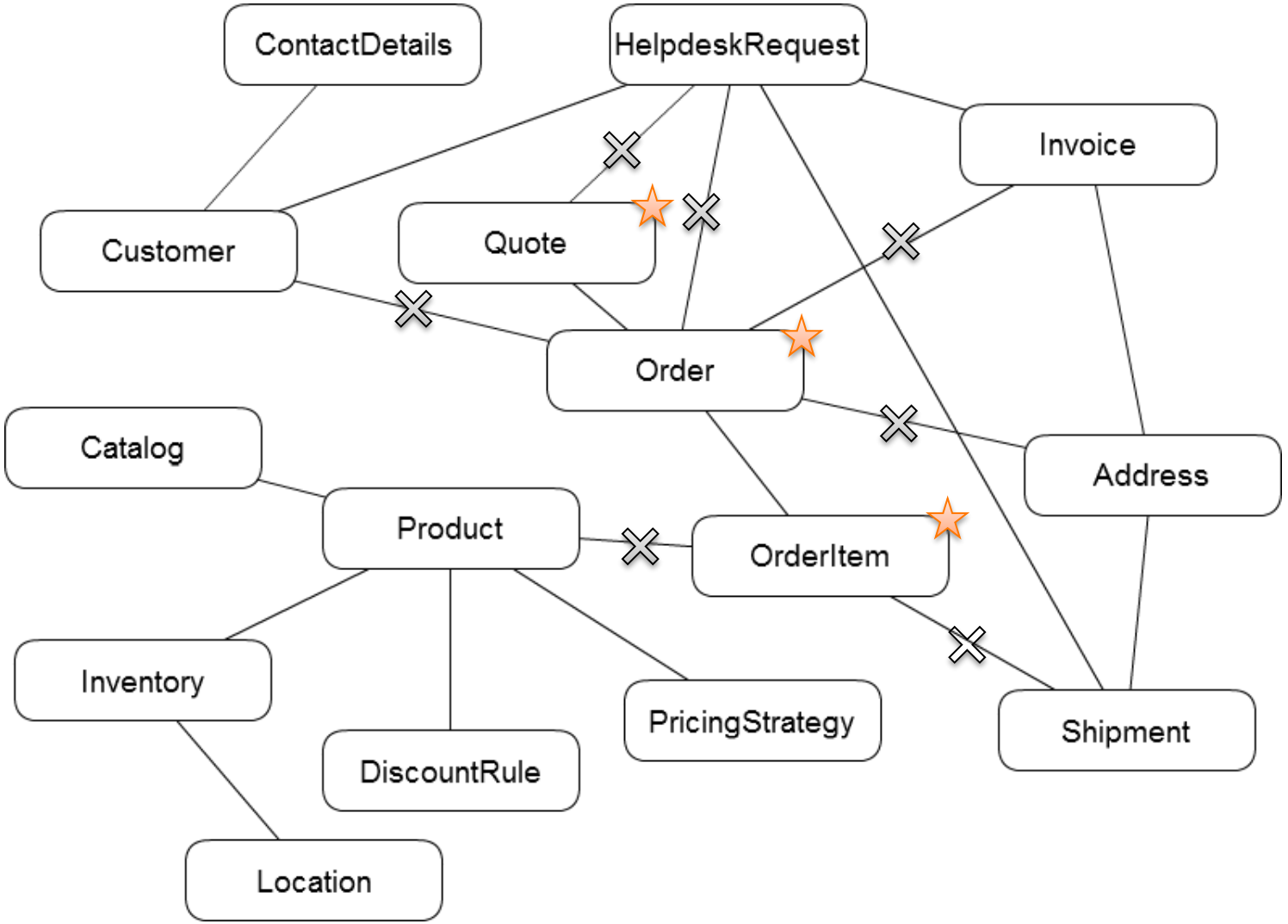
Evolution of a Domain Model



Isolating performance bottlenecks



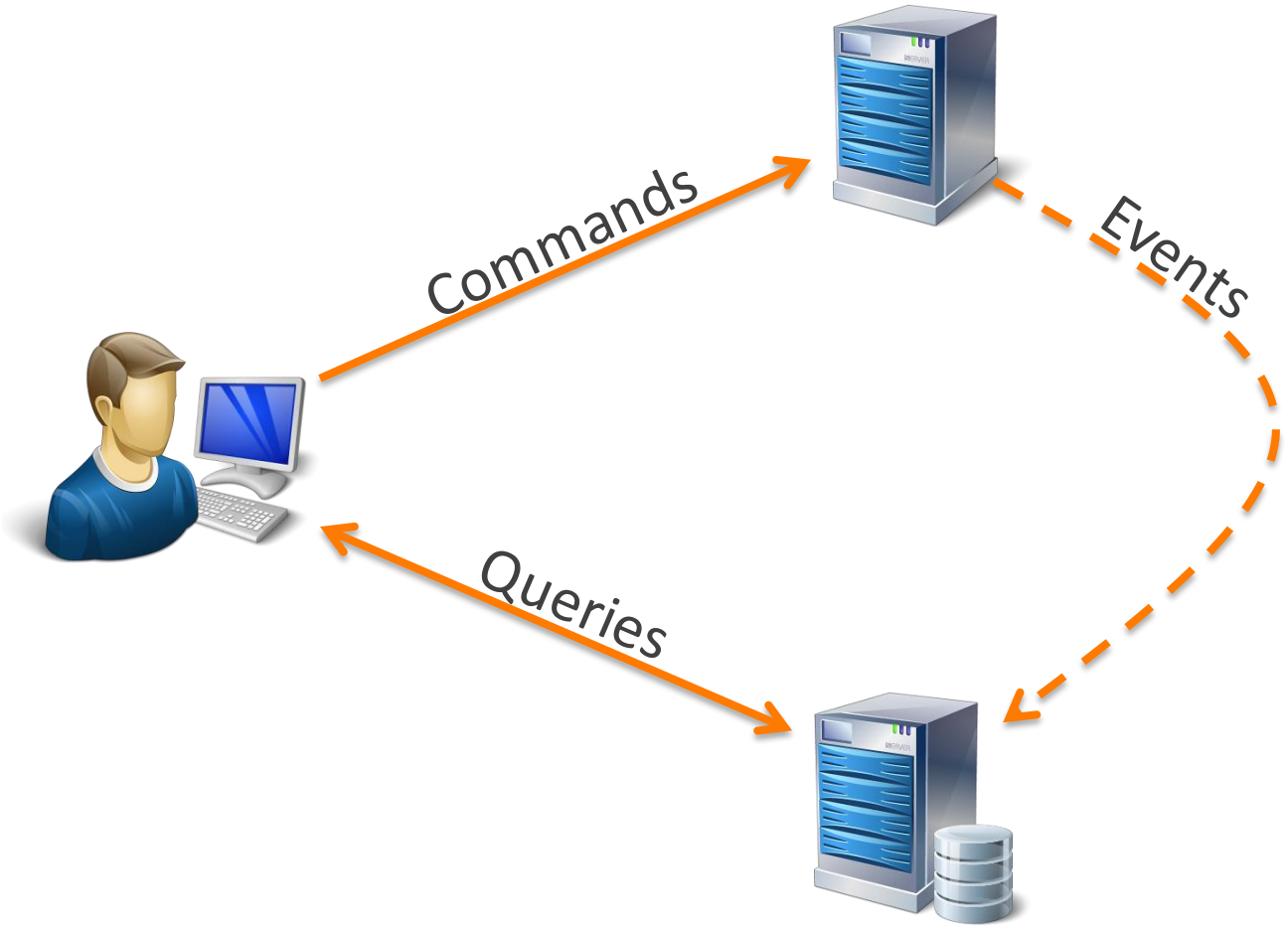
Isolating performance bottlenecks…



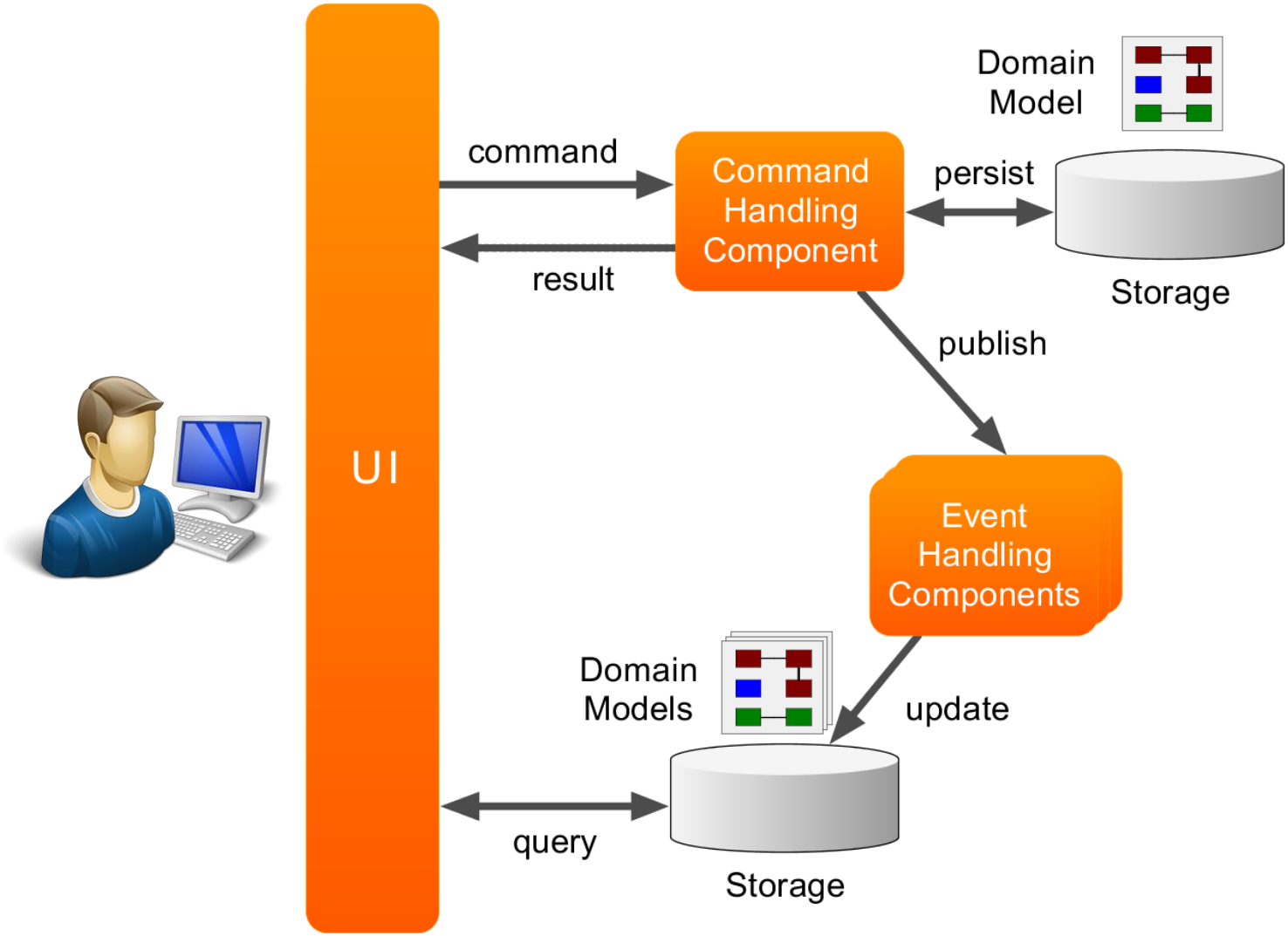
Principles of CQRS

- Use different models for different purposes
  - Commands
  - Queries
- Define clear consistency boundaries
  - Aggregates
- Address non-functional requirements
  - Response times / latency
  - Usage / throughput
  - Staleness
  - Consistency

CQRS Based Architecture



CQRS Based Architecture



Axon Framework

- "CQRS Framework" for Java
- Simplify CQRS based applications
  - Provides building blocks for CQRS applications
- Current version*: 2.0.6
- More information: www.AxonFramework.org

* On November 20th, 2013

Axon Framework

- Provide necessary abstractions
  - EventBus
  - CommandBus
  - Saga, Aggregate, Event
- Provide infrastructure building blocks
  - Local JVM (Simple.....)
  - High performance (DisruptorCommandBus)
  - Distributed (DistributedCommandBus, AMQPTerminal)
- Other building blocks
  - Given-when-then test fixtures
  - Event Store

The Case: BridgeBig.com

- On-line Bridge (card game) platform
- 100% Browser based
- Play tournaments for money prizes

TRIFORK.

BridgeBig – A CQRS based architecture

- Divide the application in logical functional components / bounded contexts
  - User account
  - Tournament
  - Game
- Separate the commands from the queries
- Main frameworks, libraries & tools
  - Axon Framework
  - Spring
  - GWT
  - Hibernate / JPA
  - RabbitMQ
  - MySQL

Main application components

Front-end

- Display game state
- Catch user actions

Game engine

- Keep track of game state
- Enforces Bridge rules
- Process commands

Tournament engine

- Game coordination
- Player ranking
- Process commands

Query component(s)

- Pushes events to clients
- Executes queries

Event Store

- Stores events
- Source of engine state

Relational Store

- Stores Query Models

TRIFORK.

Aggregates & Bounded Contexts

- Game and Tournament
  - Clearly separated
  - Each has a separate “core API”

- Improves maintainability
  - Easy to implement new tournament types
  - Seamless refactoring for performance improvements

- Aggregates are “synchronized” using Sagas
  - Triggered by Events
  - Dispatch Commands

Event Sourcing

- Storage option for command model
- Past events contain invaluable data
- Appending small delta's is faster
- Build new features
  - Concept of "Credits" is added later
  - Management reports based on data from day 1
- Gameplay analysis
- Fraud detection a posteriori
- Tests described functionally

Event Sourcing – The code

```
@CommandHandler
public void handle(SeatPlayerCommand command) {
    Participant participant = command.getParticipant();
    if (!getGameState().mayTakeSeat(command.getParticipant())) {
        logInvalidCommand(command);
        return;
    }
    apply(new PlayerSeatedEvent(gameId, getGameState().getDirection(participant)));
    if (getGameState().areAllPlayersSeated()) {
        apply(new RegularGameStartedEvent(gameId, getGameState().getGameDefinition()));
        applyTurnChange();
    }
}

@EventHandler
public void handle(PlayerSeatedEvent event) {
    // update seating state
}
```

```
@EventHandler
public void handle(CardPlayedEvent event) {
    // update "cards on table" state
}
```

Event Sourcing - Testing

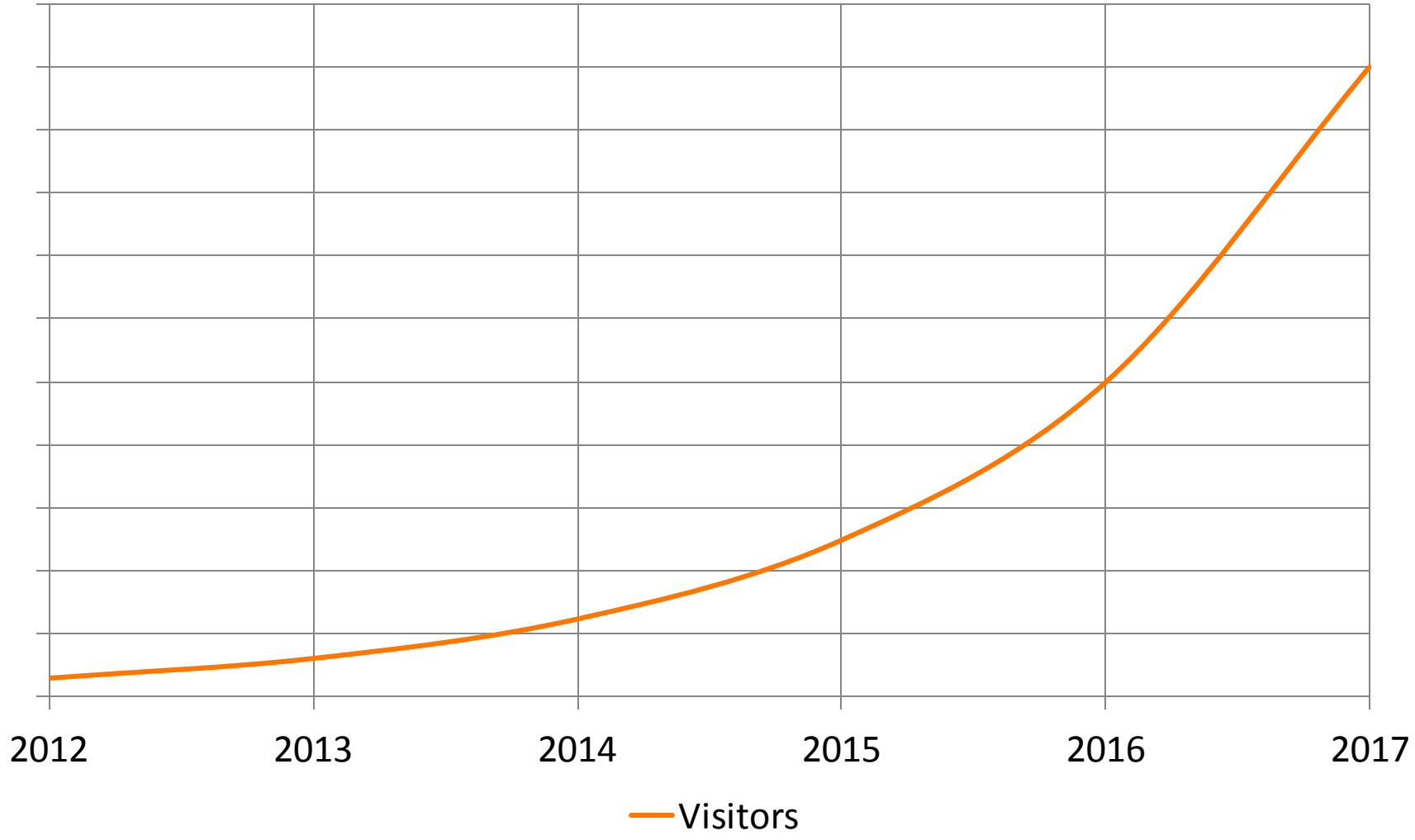
- Given-when-then fixtures
  - Given some past events
  - When I apply a new Command
  - Expect these new Events

```
fixture.given(gameStarted())
       .when(callCommand)
       .expectEvents(new CallMadeEvent(…),
                     new TurnChangedEvent(…));
```

What if the platform becomes a success?

BridgeBig's plan for scaling out

BridgeBig – The success story?



TRIFORK.

Scalability

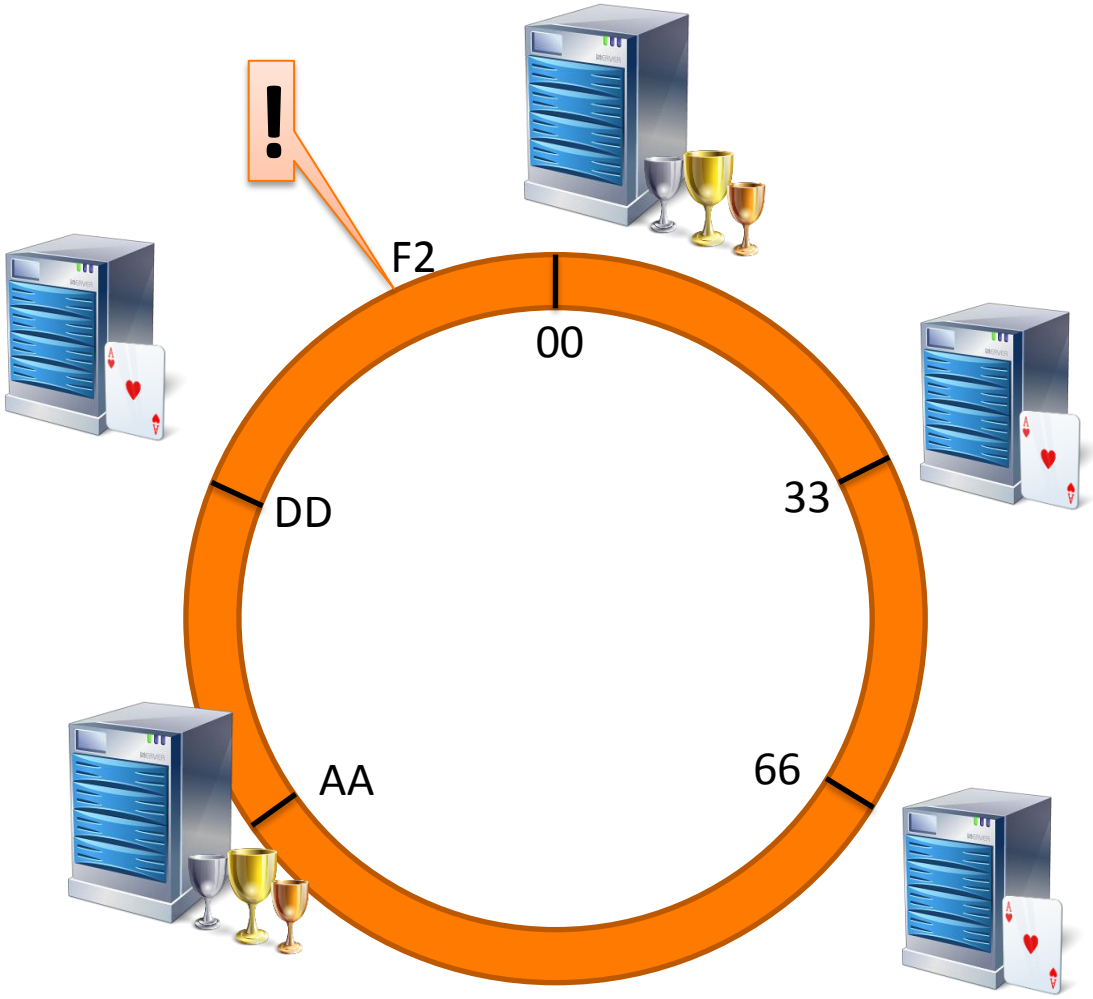
- Scaling out is straightforward
  - No need to change architectural features
  - No need to change application logic
  - No caches “to the rescue”

- Step 1: Each context on a different machine
  - Publish events over a message broker (e.g. RabbitMQ)
- Step 2: Duplicate a context
  - Route commands based on targeted aggregate identifier
  - Consistent hashing
  - Standard component in Axon 2

Demo

See some scalability in action

Routing commands – Consistent Hashing



Routing commands – Node Membership

- Axon Framework
  - DistributedCommandBus
  - JGroupsConnector
- Jgroups
  - “Toolkit for reliable multicast messaging”
  - Automatic detection and management of “members”
  - Multicast, Fixed IP list, Gossip
  - (Limited) State sharing
  - Messaging

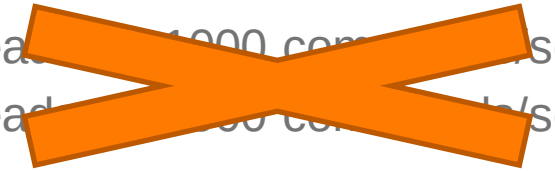
Boosting performance

Making the most of existing CPU power

TRIFORK.

Tackling performance bottlenecks

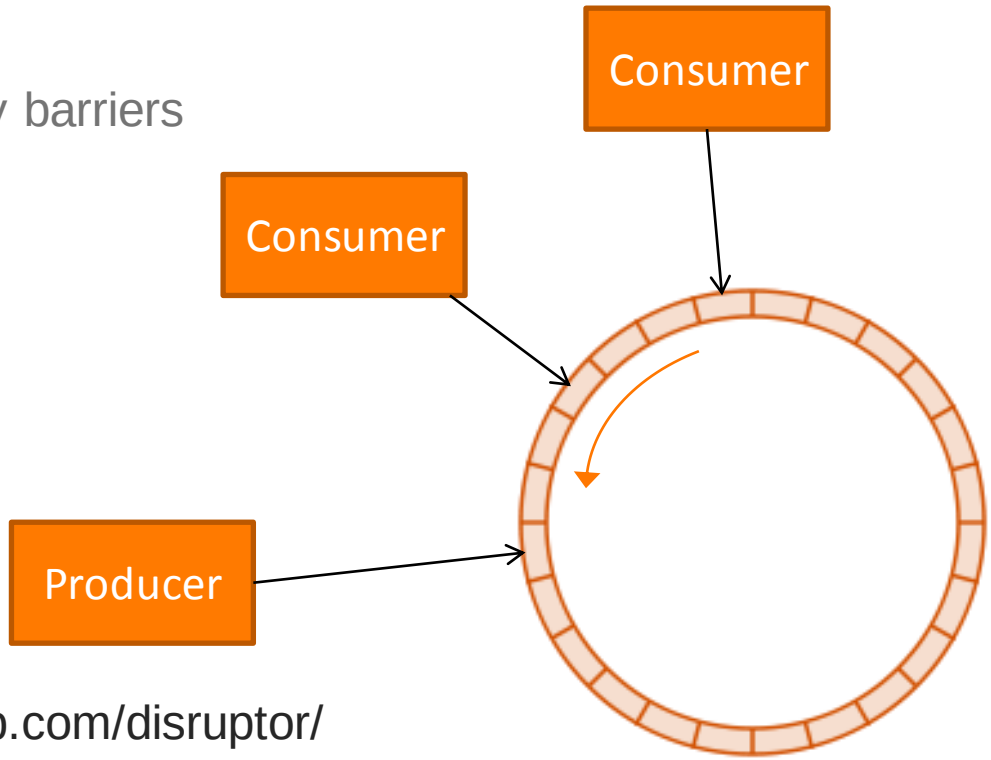
- All functional components are split into separate modules
  - Divide, measure and conquer
- Game Engine has biggest impact on overall performance
- Beware of false assumptions
  - 2 threads, 1000 components/second
  - 20 threads, 1000 components/second



TRIFORK.

High performance Command Processing

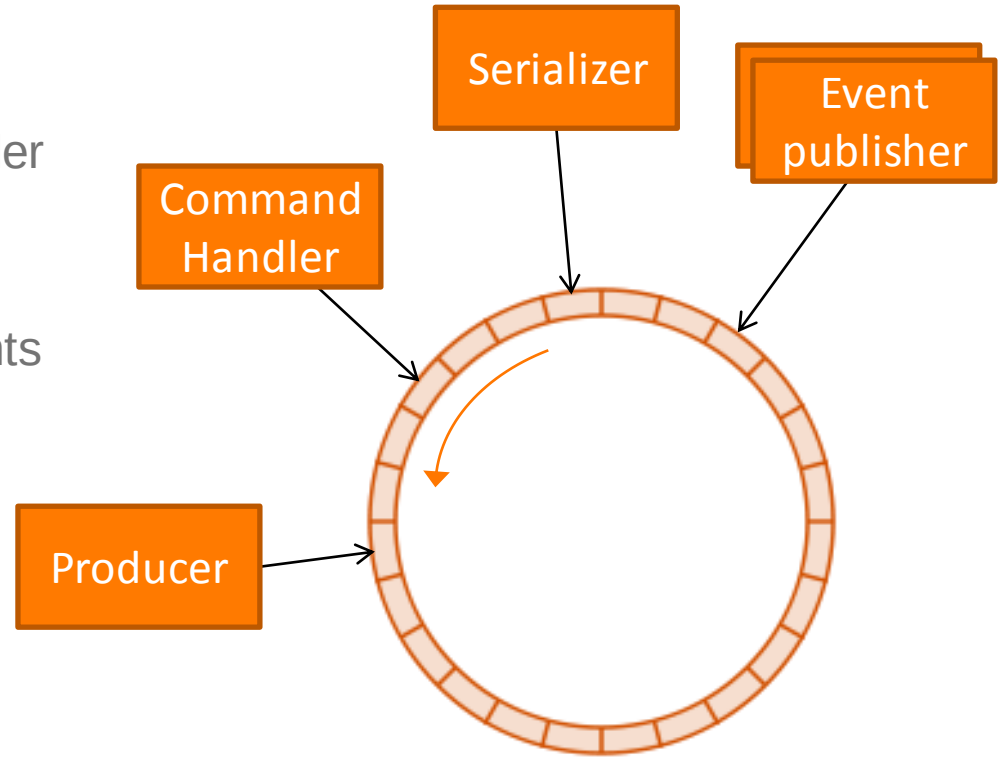
- Disruptor
  - “High performance inter-thread messaging”
  - Alternative to queues
  - Less locks and memory barriers
  - Mechanical sympathy



- http://lmax-exchange.github.com/disruptor/
- http://www.parleys.com/#st=5&id=2772

Disruptor Command Bus

- Producer
  - Prepare command for execution
- Command Handler
  - Invoke the command handler
- Serializer (Optional)
  - Serialize the resulting events
- Event Publisher
  - Publish resulting events



Battle of the Buses in Axon Framework

- SimpleCommandBus
  - Command executed on dispatching thread
  - 96 842 commands per second (JDK 6, pool = 2)
  - 140 753 commands per second (JDK 6, pool = 4)
  - 120 980 commands per second (JDK 6, pool = 8)
  - 276 671 commands per second (JDK 6, no pool)
  - 298 311 commands per second (JDK 7, no pool)
- DisruptorCommandBus
  - Each command processed by 2 threads
  - 961 168 commands per second (JDK 6)
  - 1 010 979 commands per second (JDK 7)

Intel Core i7 2.80Ghz – 2 cores (4 threads) – 6 GB RAM

User Interface Performance Tuning

A little white lie never hurts an end-user

TRIFORK.

HTTP Performance Overhead

- Users want to see real time data on screen
  - Or at least, think they do
- HTTP protocol overhead is immense
  - > 60% of data is overhead
- Don’t underestimate cost of Socket.accept()
  - Use keep-alive when possible
  - Tune your web server

I want to know what's happening on BridgeBig

```
POST http://localhost:8080/gametable/gamePolling HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
X-GWT-Permutation: HostedMode
X-GWT-Module-Base: http://localhost:8080/gametable/
Content-Type: text/x-gwt-rpc; charset=utf-8
Referer: http://localhost:8080/game?tournamentId=88271c61-e3d0-495d-b51e-c38024696d61&gwt.codesvr=192.168.56.1:9997
Content-Length: 383
Cookie: POWER_USER="rene|2012-12-07T06:52:02.541Z|e30a3b211952461fc13434db7e83574a7fcd3492";
JSESSIONID=48A7A3BF0292F039639F5756DA84FA7F; backdoor=1
Pragma: no-cache
Cache-Control: no-cache

7|0|10|http://localhost:8080/gametable/|AC956FD5F7AA72CEE499AAAF712C4081|com.bridgebig.web.game.shared.service.GamePollingService|poll|
com.bridgebig.api.ui.game.common.GamePollRequest/433853789|com.bridgebig.api.common.Participant/3784909309|2b989bb5-a005-464a-b084-
da1c66cb3500|88271c61-e3d0-495d-b51e-c38024696d61|rene|6338f72c-81b3-42b9-8c49-552ed3f910e4|1|2|3|4|1|5|5|6|7|8|9|10|
```

TRIFORK.

Here's what happened

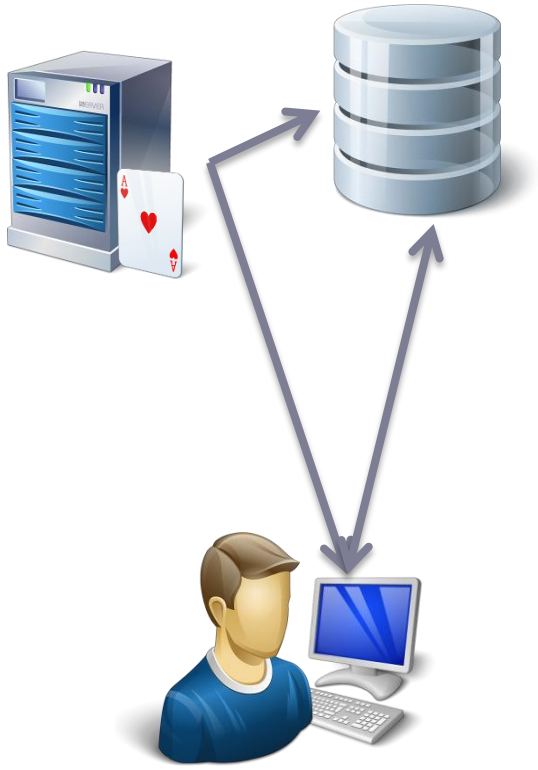
```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Cache-control: no-cache, no-store, max-age=0
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Disposition: attachment
Content-Type: application/json;charset=utf-8
Content-Length: 470
Date: Tue, 16 Oct 2012 11:37:32 GMT

//OK['TppXUli',8,'TppXUi7',0,2,0,7,0,6,2,5,4,2,2,3,1,2,1,["com.bridgebig.api.ui.game.common.GamePollResponse/1198729754","java.util.ArrayList/4159755760","com.bridgebig.api.ui.game.uievent.UICallExplanationsFetchedEvent/3304683639","com.bridgebig.api.game.common.Bid/4116320222","com.bridgebig.api.game.common.Level/2593678207","com.bridgebig.api.game.common.Suit/2748266156","com.bridgebig.api.game.common.Bidding/984765157","18fadec5-34b7-4a65-b995-5ae97cb9508c"],0,7]
```

TRIFORK.

Real time UI feedback

- There is an Event for every change
- Stream those events (directly) to the UI
  - WebSockets
  - Fallback to Long Polling
    - Servlet 3 Async
- Boosts perceived performance



WebSockets

- Effectively: Full Duplex TCP connection
  - With SSL/TLS Support
- HTTP 1.1 compatible
  - Supported by Chrome, Firefox, IE10, etc.
- Protocol overhead (excl. handshake): 2-14 bytes per message

Result prediction

- Commonly seen procedure:
  - Server.loadUserDetails()
  - User modifies data and clicks “save”
  - Server.updateUserDetails()
  - Server.loadUserDetails()
- Why would you want to “read your writes”?
- Update the User Interface *while* sending the command to the server
  - Only act on errors

Summarizing

Technology overview

Game engine

- Distributed CommandBus
- Disruptor CommandBus
- Cache (short lived)

Event Store

- Optimized for appending

Tournament engine

- Distributed CommandBus
- Disruptor CommandBus
- Cache (longer lived)
- Sagas

Front-end

- Browser as app. platform
- Instant feedback
- WebSockets

Relational Store

- Denormalized data

Query component(s)

- Push events to clients
- Straightforward queries

TRIFORK.

High performance and scalable architectures

- Divide the application in logical functional components / bounded contexts
- Separate the commands from the queries
- Profiling and performance testing
- Improve performance or scale out key components
- Trick users into believing it's fast
- Use caching thoughtfully
  - Beware of arbirary time-to-live settings
  - Prefer fact-based eviction (e.g. Events)

Questions?

More information:

http://www.axonframework.org
http://www.jgroups.org
http://lmax-exchange.github.com/disruptor/
http://www.parleys.com/#st=5&id=2772

allard.buijze@trifork.nl

TRIFORK.