Universität St.Gallen

# EDPO – Intermediate Presentation

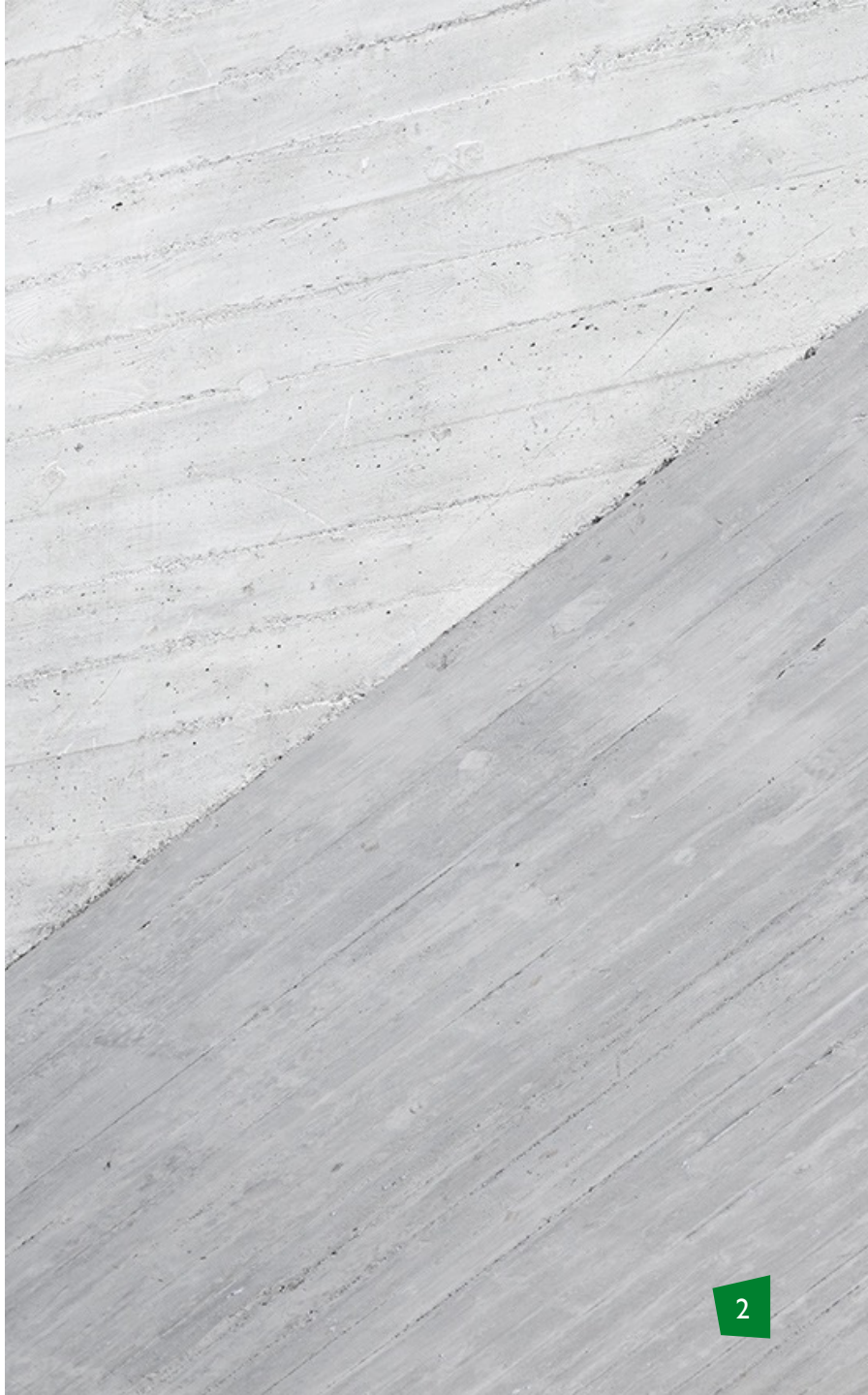
St.Gallen, 18. April. 2024

Karim Ibrahim
Jonas Länzlinger
Kai Schultz

From insight to impact.

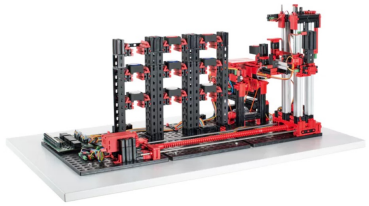
# Agenda

1. Project Domain
2. Holistic View
3. Services
4. Cenceptual Architectural Decisions
5. Discussion of Trade-offs
6. Lessons Learned & Insights
7. Demo

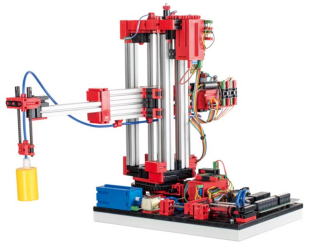


# Project Domain

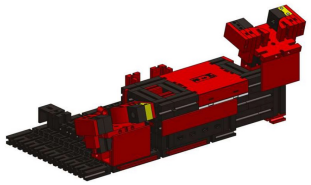
- Fischertechnik Industry 9.0 smart factory
- Modelling a Process Workflow



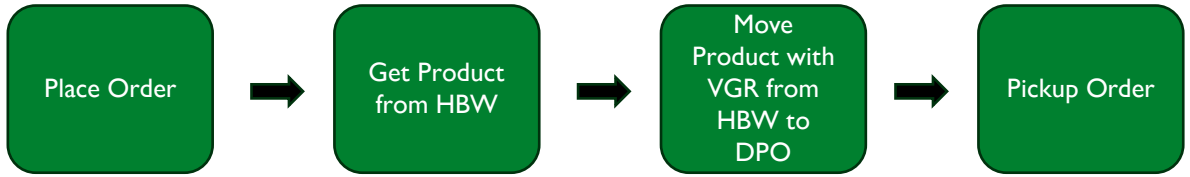
HBW
High-bay Warehouse

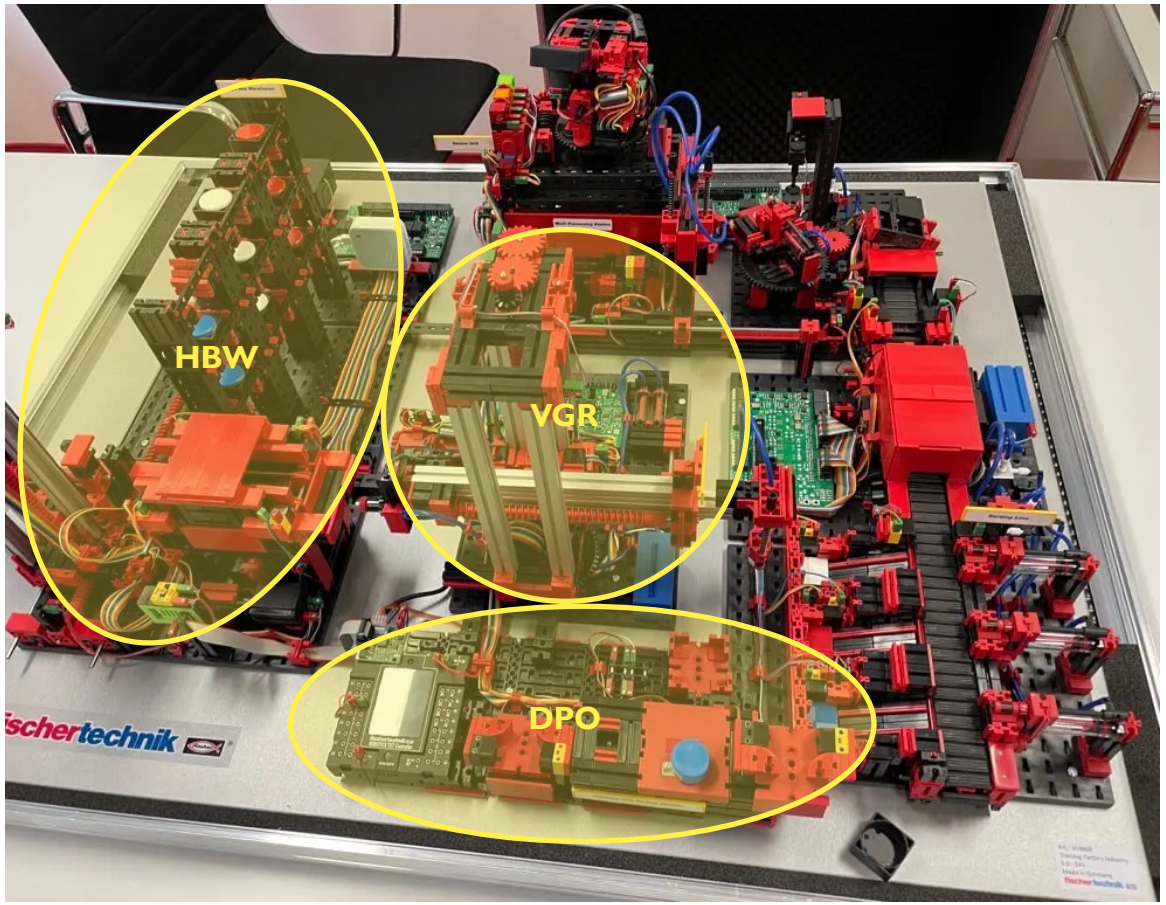


VGR
Vacuum Gripper

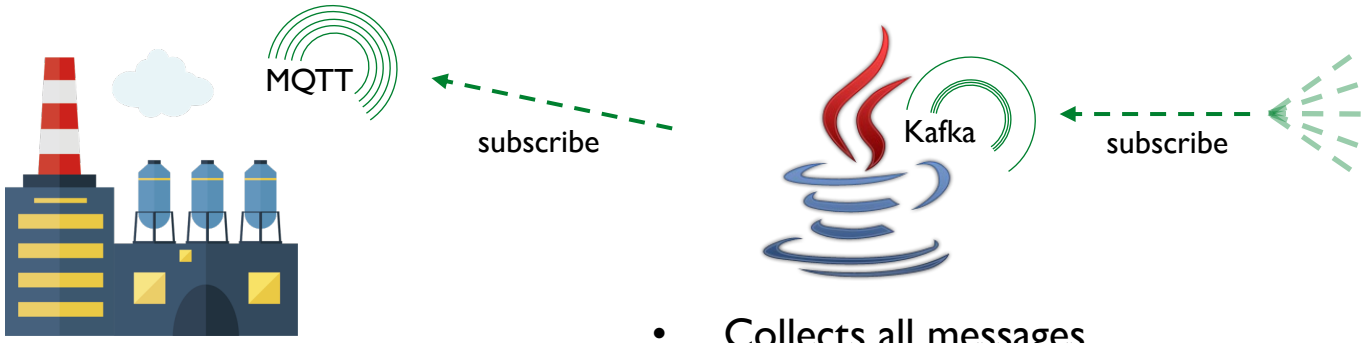


DPO
Delivery & Pickup



# Holistic View – MQTT, Kafka, Camunda




**Simulating a Factory**

- ✓ Produce **MQTT** events like real factory
- ✓ Generate predictable message flows

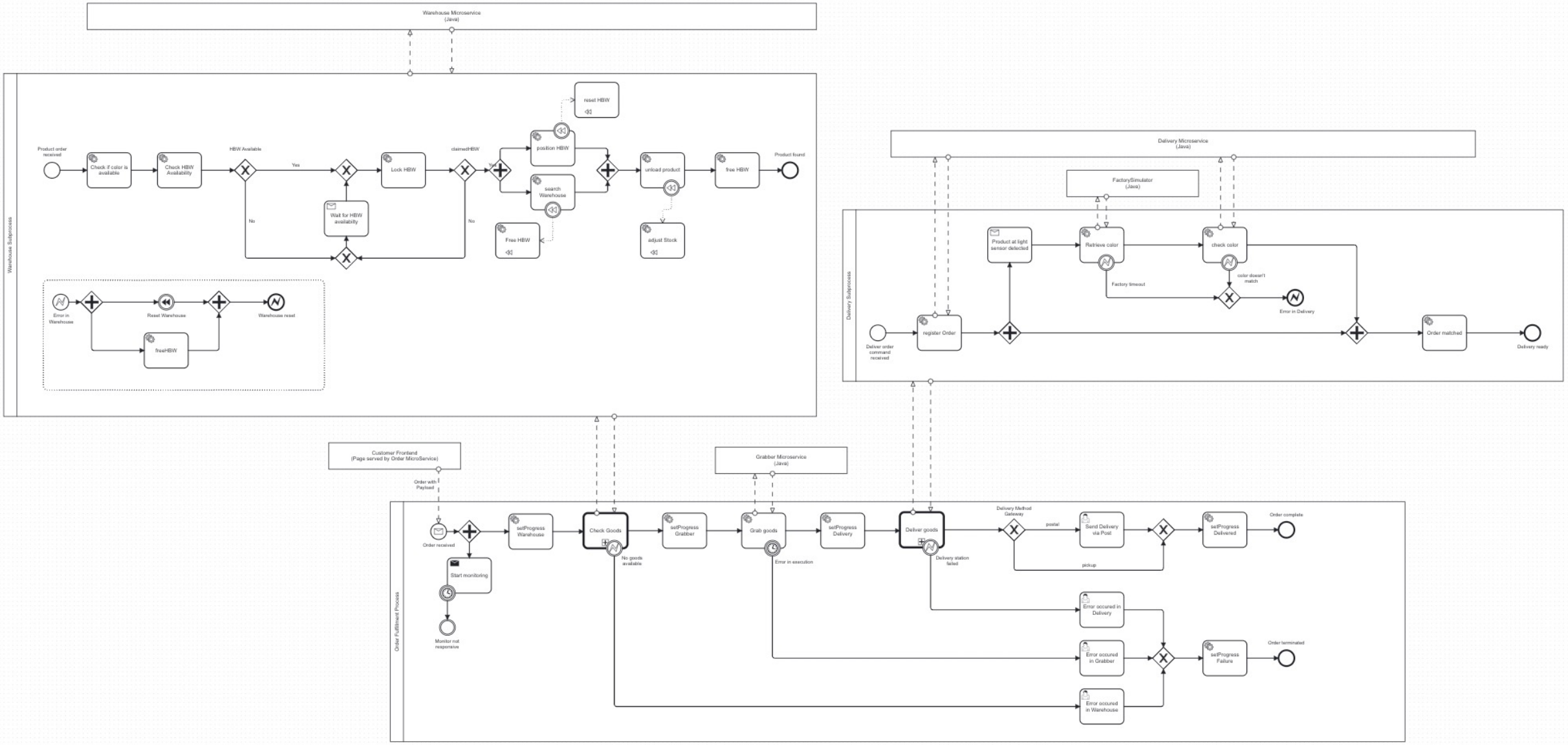
- Collects all messages from Factory
- Dispatches as **Kafka** Messages (1:1)
- See ADR-01

**Connected to Camunda 8 Cloud Cluster**

- Subscribe to respective FactoryListener Kafka Topics
- Track, manage, and control behavior of respective factory stations
- Choreographic communication through Kafka

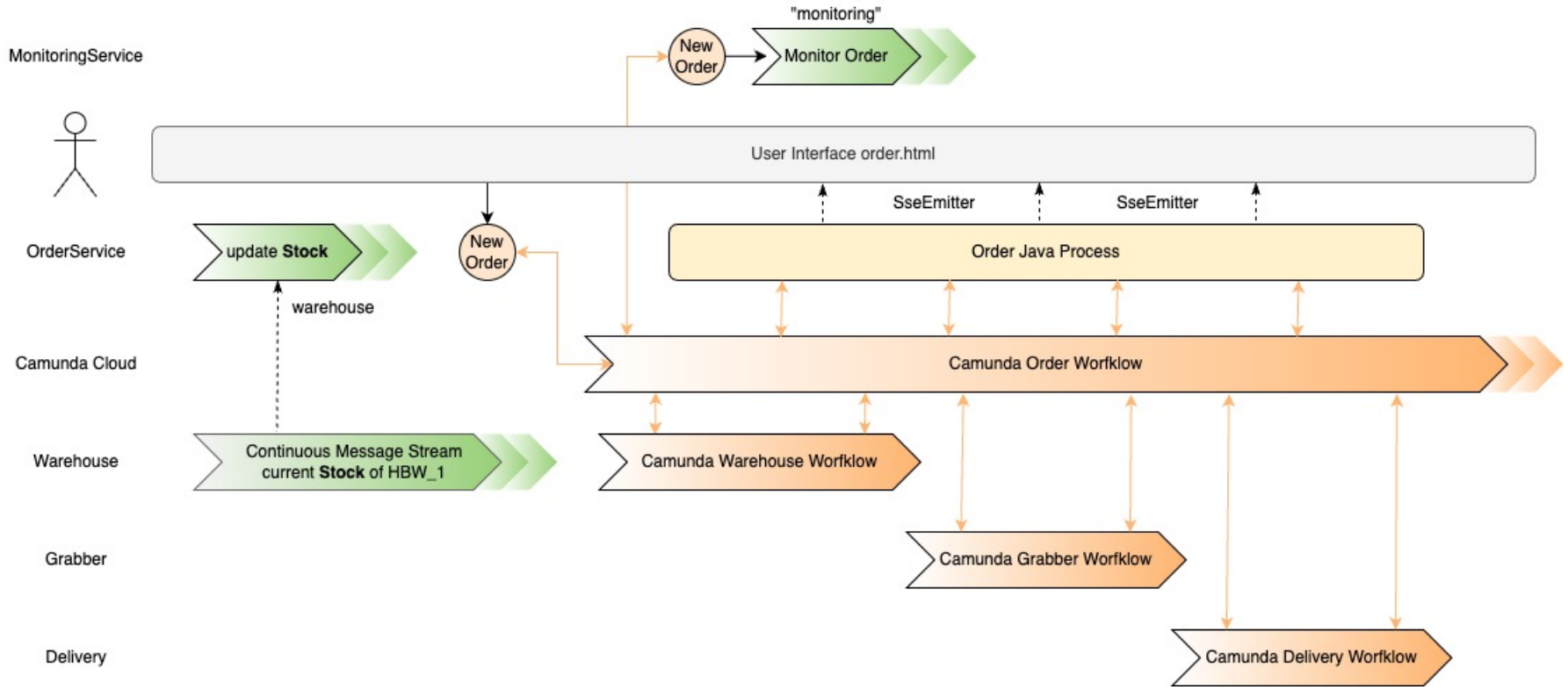


"Factory" von Unbekannter Autor ist lizenziert gemäß CC BY-NC
"Java icon" von Unbekannter Autor ist lizenziert gemäß CC BY-NC


Warehouse Microservice (Java)
Warehouse Subprocess
Product order received
Check if color is available
Check HBW Availability
HBW Available
Yes
No
Lock HBW
claimedHBW
reset HBW
position HBW
search Warehouse
Free HBW
unload product
free HBW
Product found
Wait for HBW availability
adjust Stock
Error in Warehouse
Reset Warehouse
Warehouse reset
freeHBW
Delivery Microservice (Java)
FactorySimulator (Java)
Delivery Subprocess
Product at light sensor detected
Retrieve color
check color
Factory timeout
color doesn't match
Error in Delivery
Deliver order command received
register Order
Order matched
Delivery ready
Customer Frontend (Page served by Order MicroService)
Grabber Microservice (Java)
Order with Payload
Order Fulfillment Process
Order received
Start monitoring
Monitor not responsive
setProgress Warehouse
Check Goods
No goods available
setProgress Grabber
Grab goods
Error in execution
setProgress Delivery
Deliver goods
Delivery station failed
Delivery Method Gateway
postal
pickup
Send Delivery via Post
setProgress Delivered
Order complete
Error occured in Delivery
Error occured in Grabber
Error occured in Warehouse
setProgress Failure
Order terminated

# Services - Order

SSE Communication

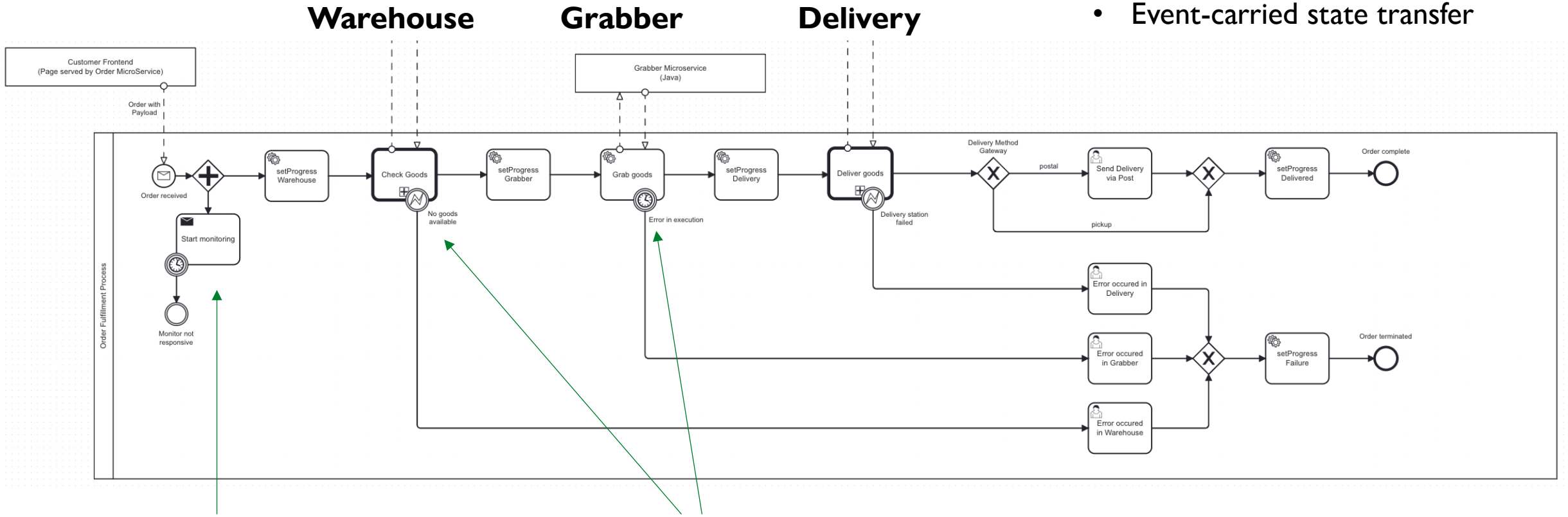



## Server-sent Events

- Enables to send messages from Order service to Frontend
- server push technology
- via HTTP connection
- customized update interval

# Services - Order

- “Process Model Ownership”
- calls subprocesses
- Event-carried state transfer



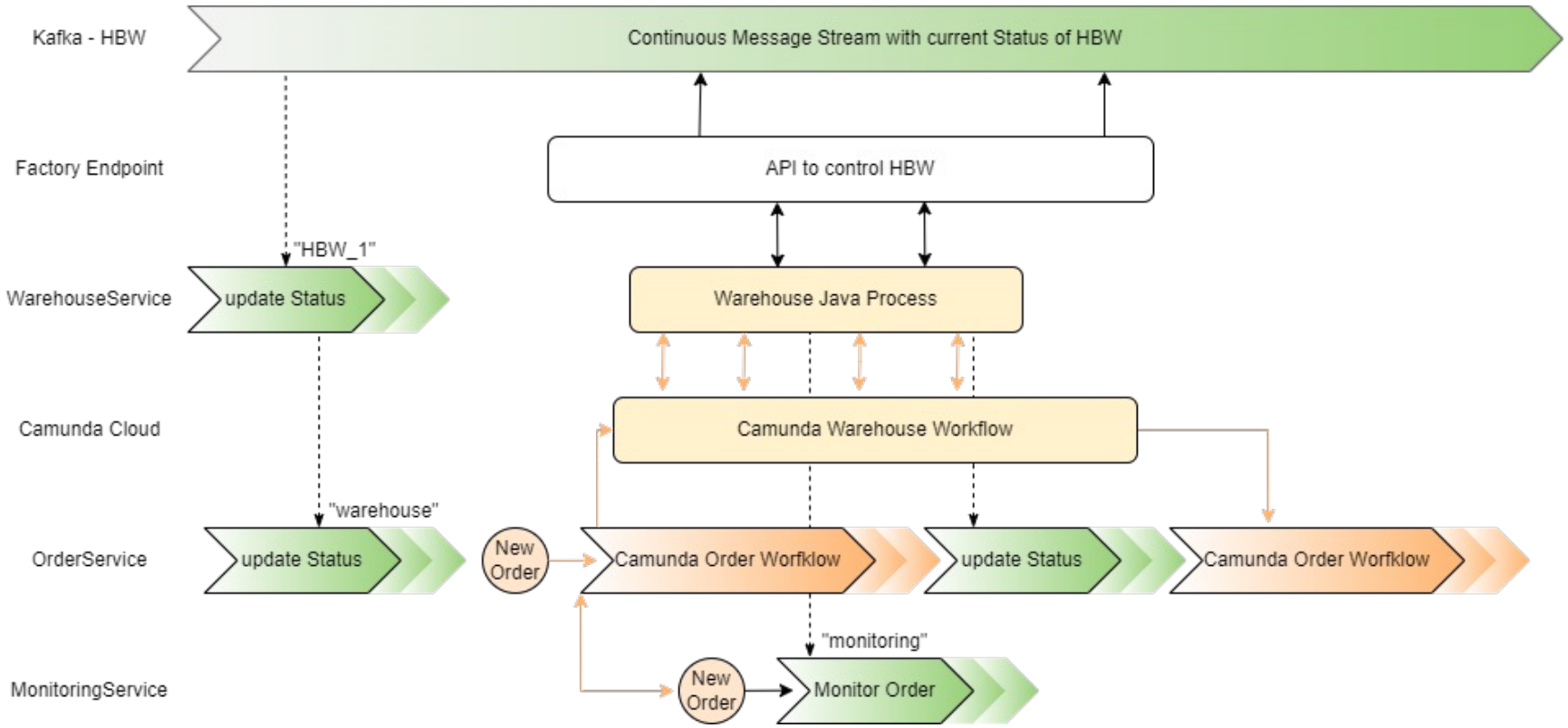

**Stateful Resilience Pattern**

Start monitoring the Order upon receiving an event on the “monitoring” Kafka topic.

**Different Boundary Error Events**

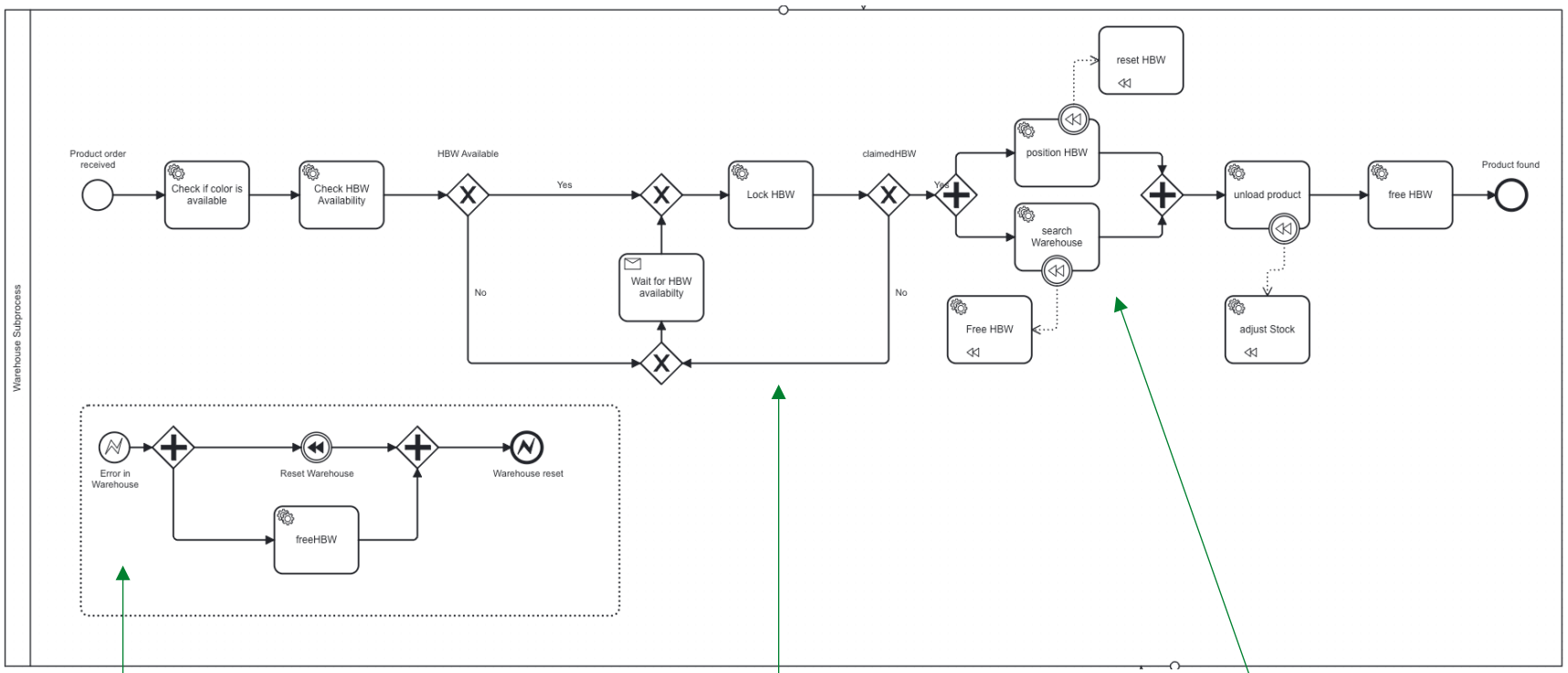
- Timer Boundary Event
- Error Boundary Event

# Services - Warehouse




- Orchestrated Communication via Camunda Process
- Choreography via Kafka
- Event-carried state transfer for error prevention

# Services - Warehouse

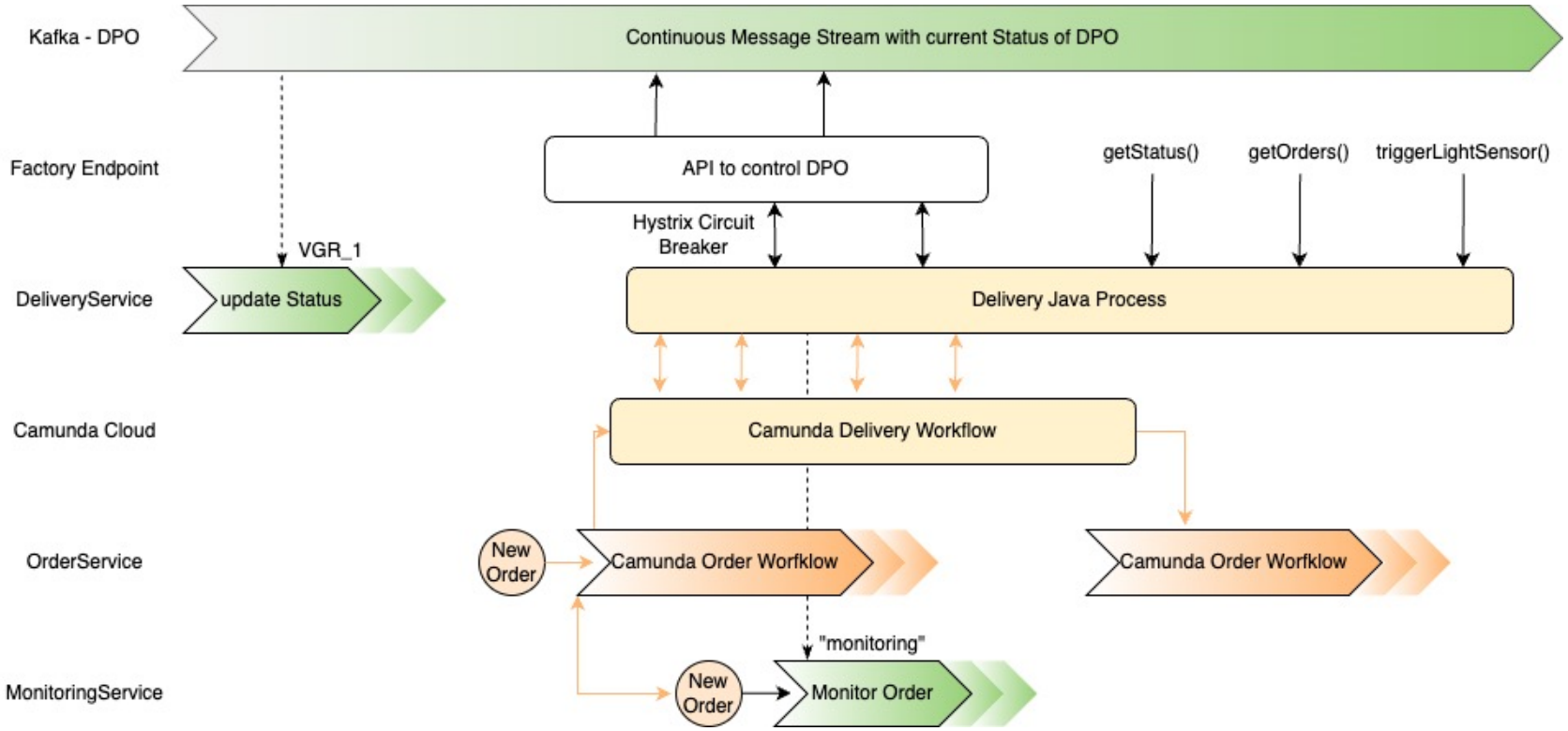



**Saga Pattern**
Physical Warehouse
System needs to be reset in case of failure

**Limited Resources**
Only one HBW process can access the system at a time

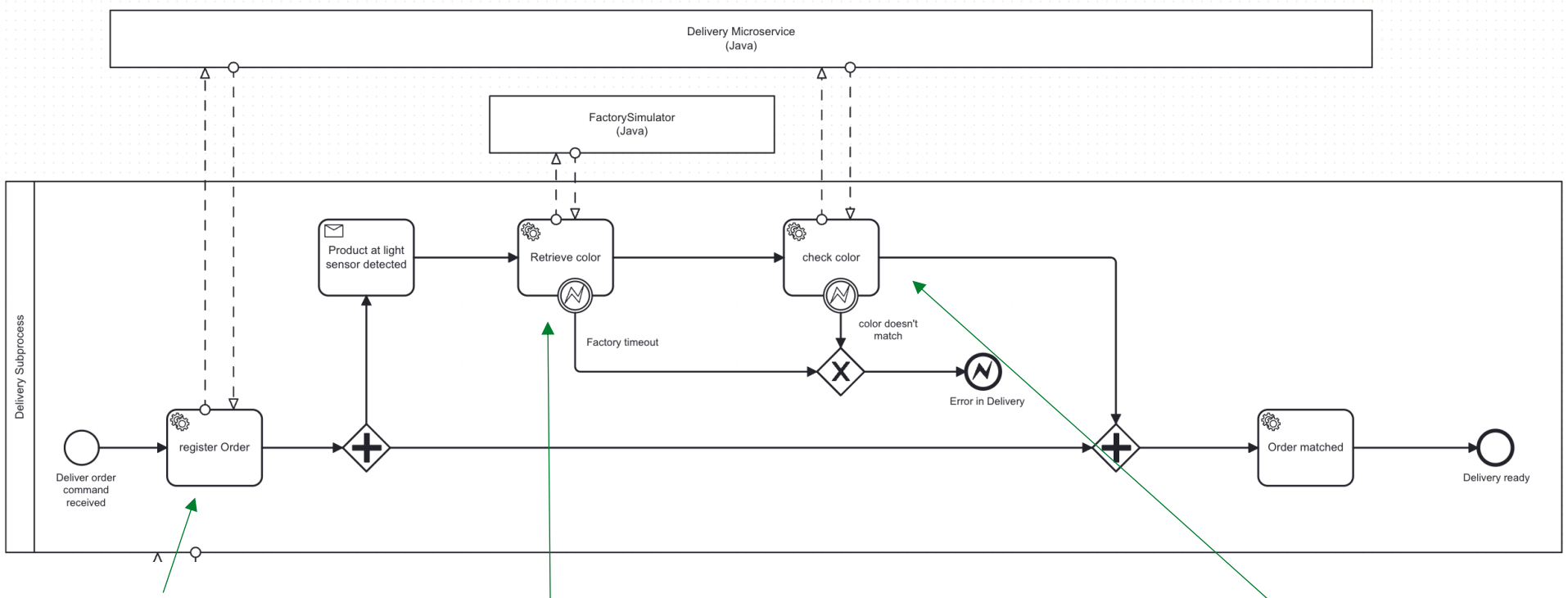
**Aggregating Messages**
Process depends on success of both physical and digital tasks

# Services - Delivery




- Orchestrated Communication via Camunda Process
- Choreography via Kafka
- Hystrix Circuit breaker
- Rest endpoints for easier maintainability and testability

# Services - Delivery

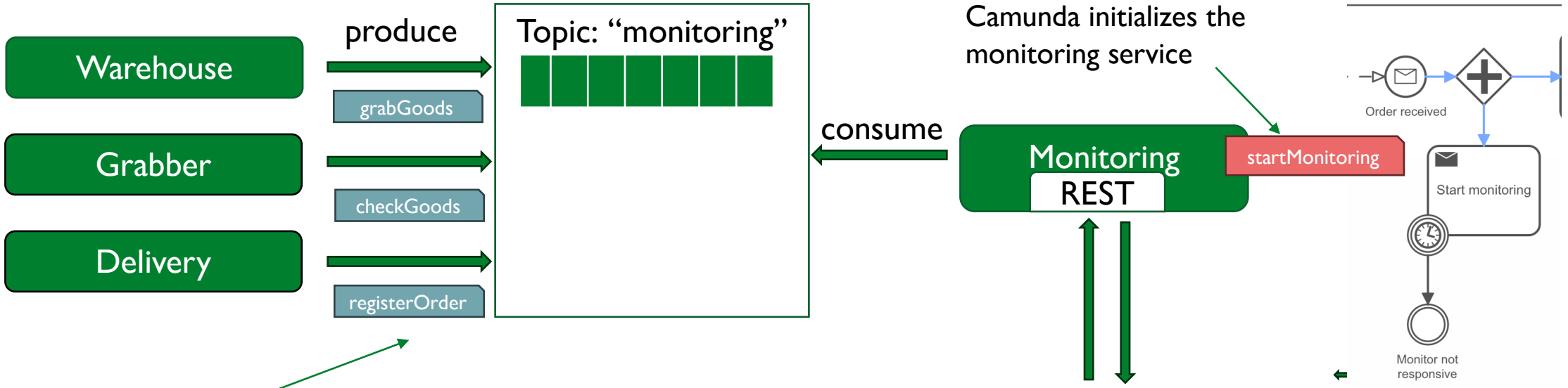



**Consistency**
Put Order in a FIFO queue upon arrival of a new Order token at the Delivery station.

**Fallback**
Hystrix Circuit breaker in place.
Waiting for response.
If Factory does not respond, user needs to remove the product manually.

**Color Matching**
Order can only be processed if it is in the queue.
Order can only be processed if Factory detects a matching color workpiece at the light sensor.

# Services - Monitoring




**Events**
Services produce dedicated monitoring events that are consumed by the monitoring service

**HTML UI**
Minimal UI that communicates over HTTP to display monitoring events

**Monitoring**

2ee2902c-6c9f-4859-a4ac-972ba76bc1dc - Service: warehouse Method: checkGoodsAvailable Status: [success]
2ee2902c-6c9f-4859-a4ac-972ba76bc1dc - Service: warehouse Method: checkHBW Status: [success]
2ee2902c-6c9f-4859-a4ac-972ba76bc1dc - Service: warehouse Method: lockHBW Status: [success]
2ee2902c-6c9f-4859-a4ac-972ba76bc1dc - Service: warehouse Method: checkGoods Status: [success]
2ee2902c-6c9f-4859-a4ac-972ba76bc1dc - Service: grabber Method: grabGoods Status: [success]
2ee2902c-6c9f-4859-a4ac-972ba76bc1dc - Service: delivery Method: registerOrder Status: [success]

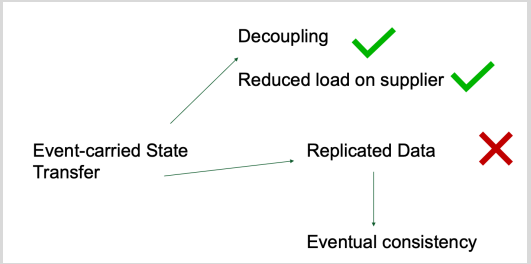
# Conceptual Architectural E1/E2

## Kafka

- Every Service publishes to its own topic (ADR1)
- Usage of consumer groups (ADR5)

## Event-carried State Transfer (warehouse)

- Order Service that serves an UI should be informed about the current stock of the warehouse => **Event-based synchronization**



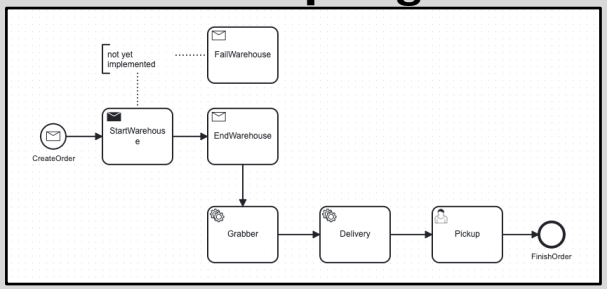

Source: Lecure Slides Week 2; Slide 31; 02-EDPO 2024-02-29.pdf

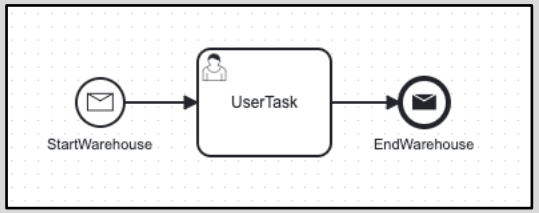
## Abstraction of the factory

- Decision to use the low-level data set
- Testset contains MQTT Messages => Since we use kafka we decided to implement a **factorysimulator** that reads from a file and publishes to MQTT (ADR3)
- Unclear in the beginning what fields of the testsets messages we really need => Implementation of a **factorylistener** that transforms MQTT to Kafka messages as second service (ADR3)
- Abstraction of factory stations to services => 1:1 (ADR4)

# Conceptual Architectural E3/E4

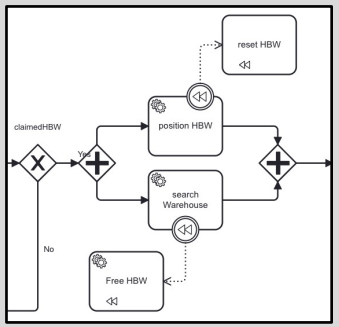
## BPNM with Spring & Camunda

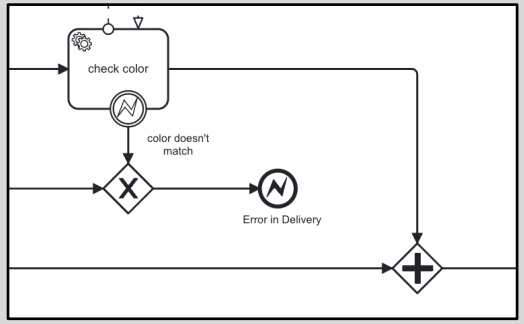



⇒ Initial modelling of BPNM
⇒ Initial usage of Camunda 7 (later Camunda 8)

## Process Elements



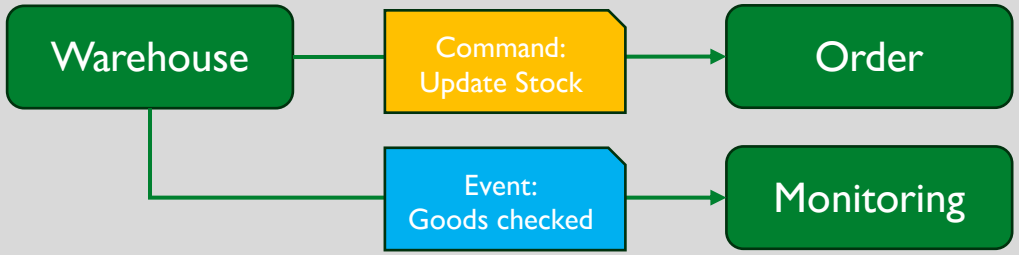

⇒ Extension of processes with gateways & subprocesses

## Orchestration vs Choreography (ADR7)

Core processes modelled in Camunda => Orchestration

Future Plans: Introduce more choreography where it makes sense (stream processing)

## Events/Commands




⇒ Commands to indicate future actions
⇒ Events to indicate past events

# Conceptual Architectural E5

- Camunda 8 and Zeebe

| | Advantages | Disadvantages |
|---|---|---|
| **Architecture** | Reduction of operational overhead | Less control over physical deployment and infrastructure |
| **Scalability** | Scales without the need for manual intervention | |
| **Development** | Simple deployment process, as it is a SaaS solution; versioning | Less flexibility in terms of deep integration compared to Camunda 7 |
| **Features** | Zeebe, Operate and Tasklist | Not as mature as Camunda 7 |
| **Ease of Use** | Very user-friendly, as it is a SaaS solution | |

- Decoupling of BPMN Engine and worker applications
- Intuitive API

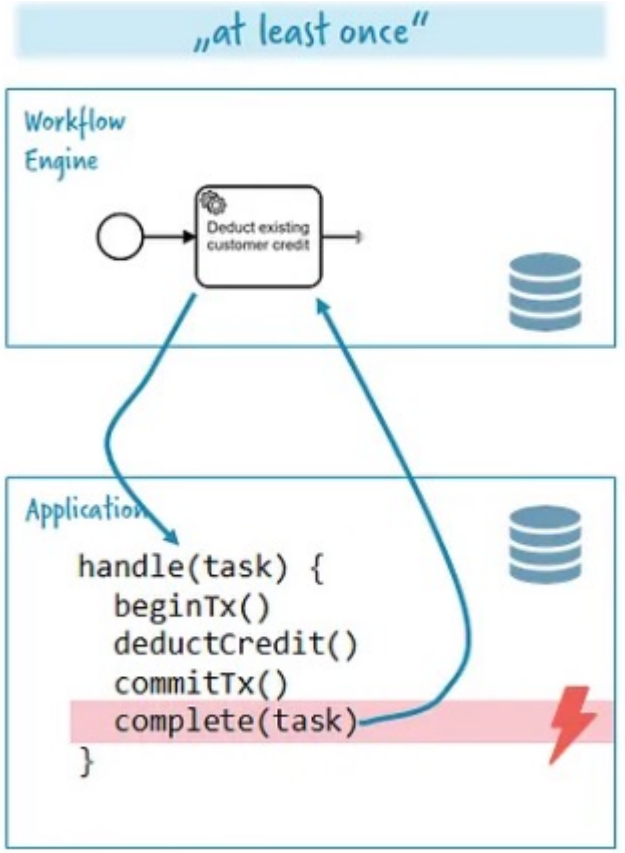


Source: https://blog.bernd-ruecker.com/zeebe-io-a-horizontally-scalable-distributed-workflow-engine-45788a90d549

# Conceptual Architectural E5

- Camunda 8 and Zeebe

| | Advantages | Disadvantages |
|---|---|---|
| **Architecture** | Reduction of operational overhead | Less control over physical deployment and infrastructure |
| **Scalability** | Scales without the need for manual intervention | |
| **Development** | Simple deployment process, as it is a SaaS solution; versioning | Less flexibility in terms of deep integration compared to Camunda 7 |
| **Features** | Zeebe, Operate and Tasklist | Not as mature as Camunda 7 |
| **Ease of Use** | Very user-friendly, as it is a SaaS solution | |



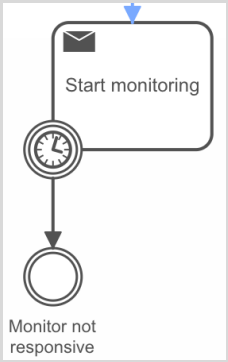

- Decoupling of BPMN Engine and worker applications
- Intuitive API
- Following At-least Once semantics (idempotency)



Source: https://blog.bernd-ruecker.com/zeebe-io-a-horizontally-scalable-distributed-workflow-engine-45788a90d549

# Conceptual Architectural E6

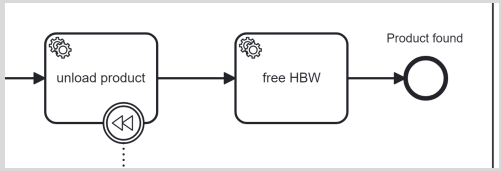
## Stateful Resilience Patterns




⇒ Stateful Retry - Monitoring service
⇒ Default of 3 retries

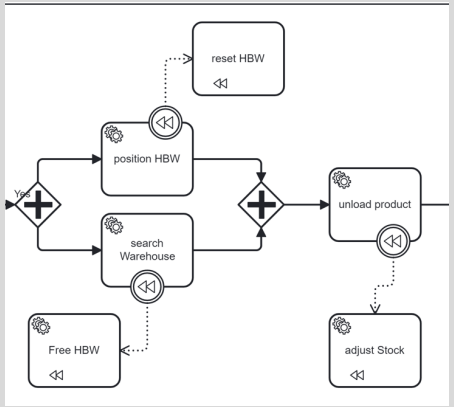
⇒ Circuit Breaker in Communication with Factory

## Outbox Pattern



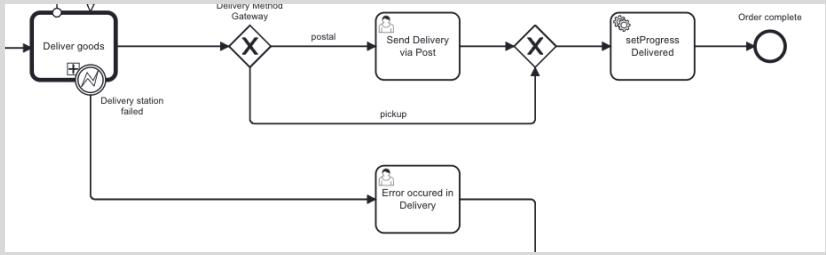

⇒ Execute the Business logic, i.e.: Unload the product
⇒ Only if succeeded publish event to free HBW and start next process

## Saga Pattern



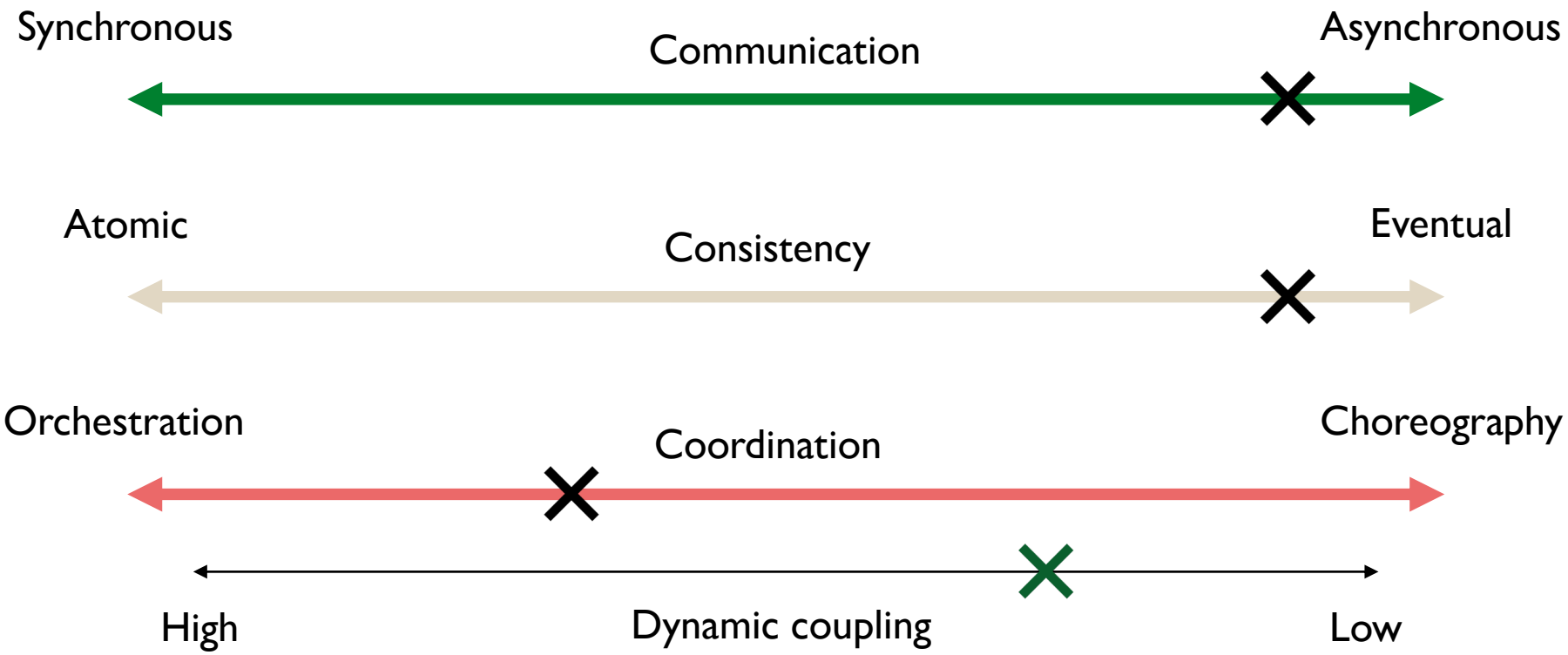

⇒ State of Machine needs to be carefully reset to initial state
⇒ Possible errors in Stock need to be factored in

## Human Intervention




⇒ If the product gets stuck use human intervention to succeed with task

# Discussion of Trade-offs



# Lessons Learned & Insights

## Messaging

- Event notification is the quick and easy solution
- Can apply multiple patterns in a system
- EDA increases and reduces complexity at the same time
- Use graphical representations of messaging flow to keep a mental model

## Kafka

- Robust message broker
- Use out-of-the box configuration
- Easy cluster management with Zookeeper
- Consumer lag may lead to retransmissions of messages -> carefully configure retention-, and cleanup policy
- Offset misconfigurations may lead to data loss
- Multiple Kafka instances gives redundancy

## Camunda

- Camunda 8 cloud platform offers a much more polished and developer friendly experience over Camunda 7
- The Zeebe client also has a very straight forward API
- Having an additional UUID independent from Camunda enables for easier correlation, especially for subprocesses
- Once a variable is in the process all subsequent tasks have access to it without specifying inputs

## General

- Consequent and meaningful naming
- Overview of variables available in process
- Reusability of code is important
- Orchestration with Camunda got very helpful as our workflow became more complex

# Team Contributions

*All tasks for this project have been done collaboratively and have been equally distributed among all team member.*

Universität St.Gallen

# Demo

From insight to impact.