EDPO – Final Presentation

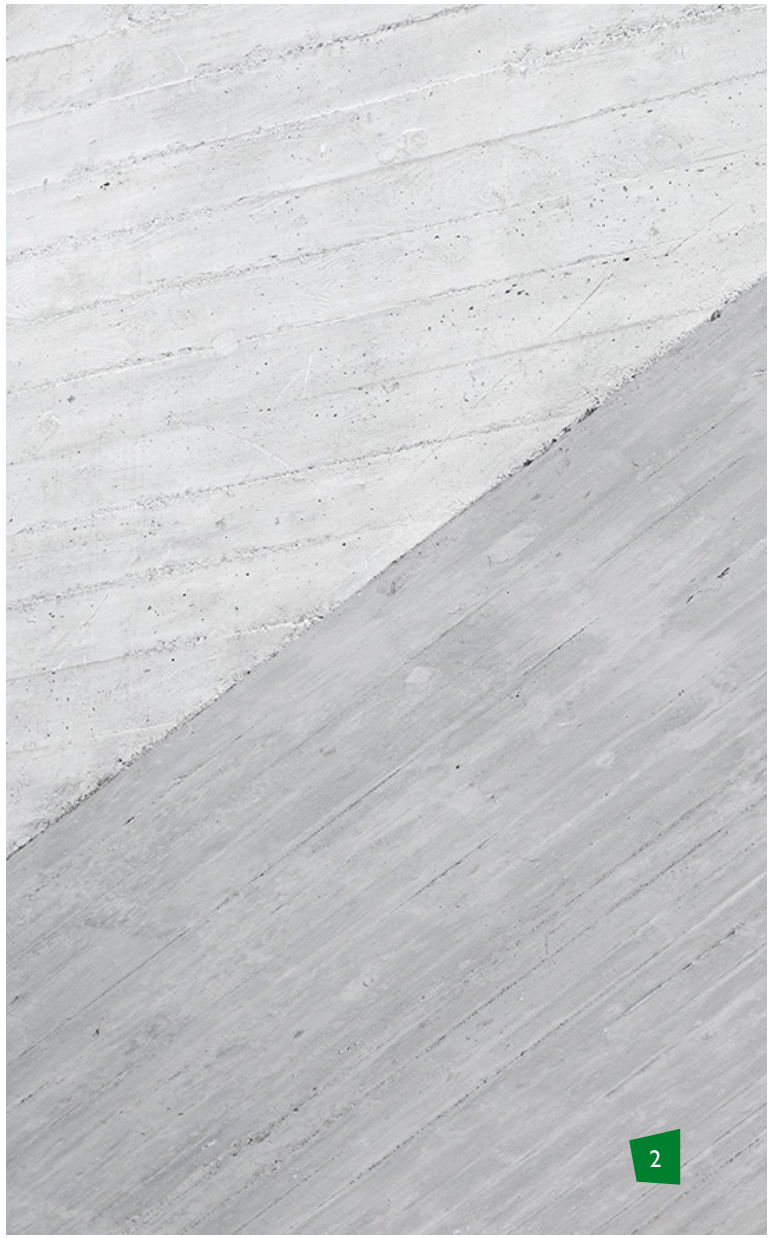
St.Gallen, 23. May 2024

Karim Ibrahim
Jonas Länzlinger
Kai Schultz

From insight to impact.

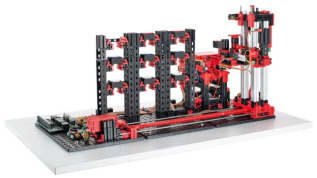
Agenda

1. Recap – Project Domain
2. Topologies
3. Stream Processing
    1. Stateless
    2. Stateful
        1. Time
        2. Join
        3. Windows
4. Demo

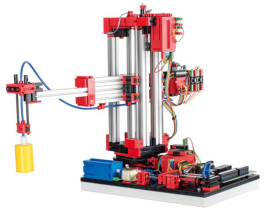


Project Domain

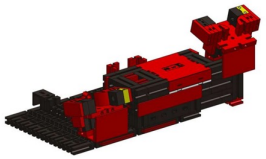
- Fischertechnik Industry 9.0 smart factory
- Modelling a Process Workflow



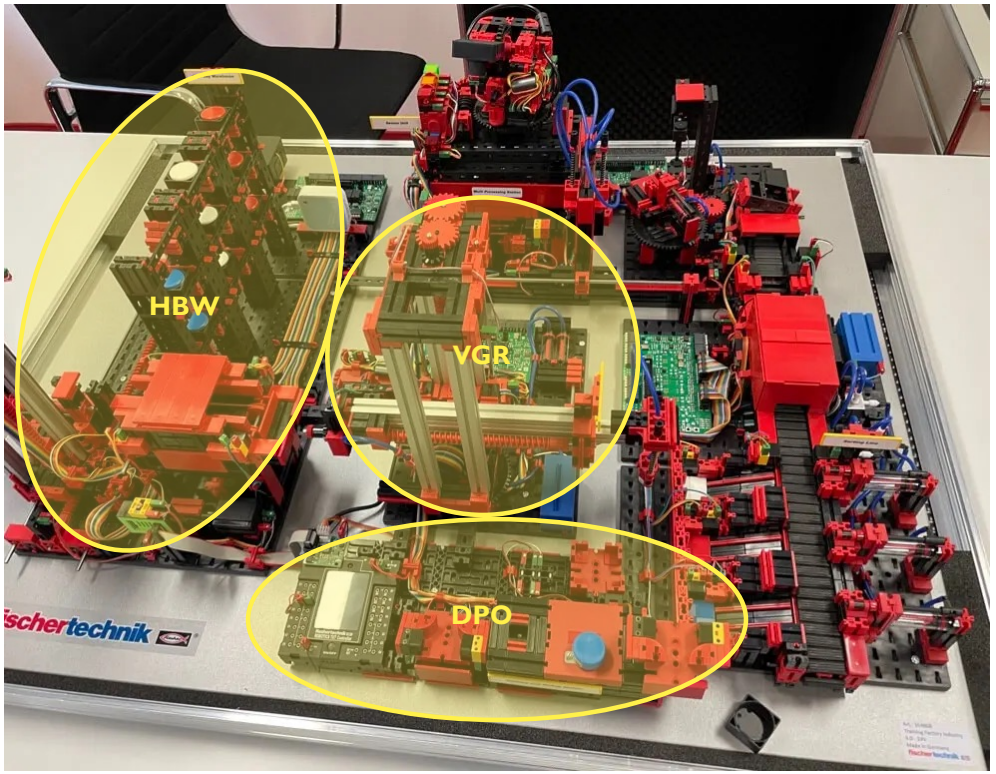
HBW
High-bay Warehouse

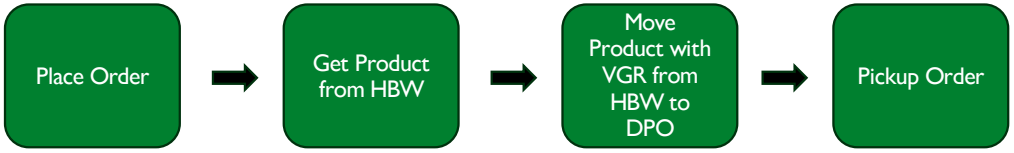


VGR
Vacuum Gripper

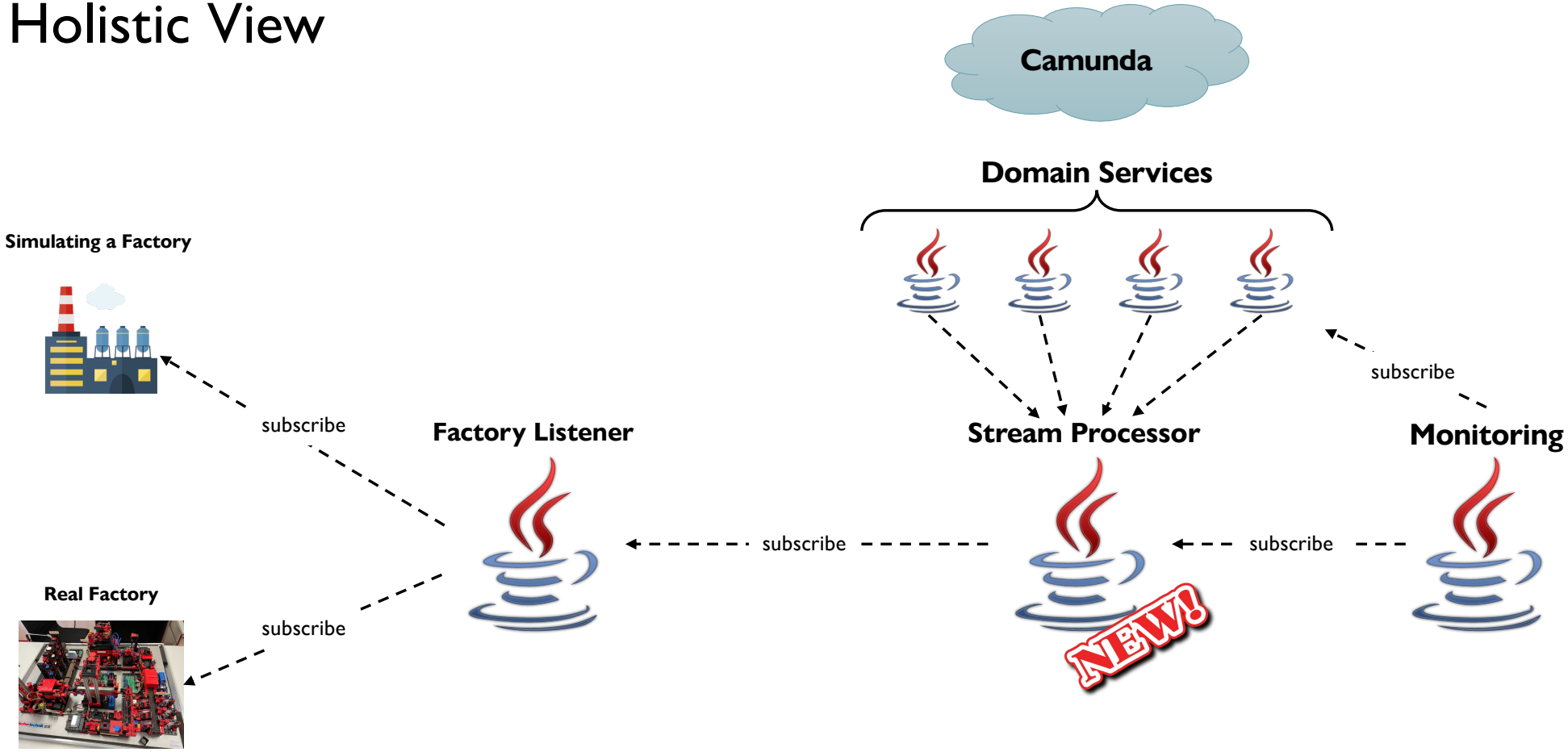


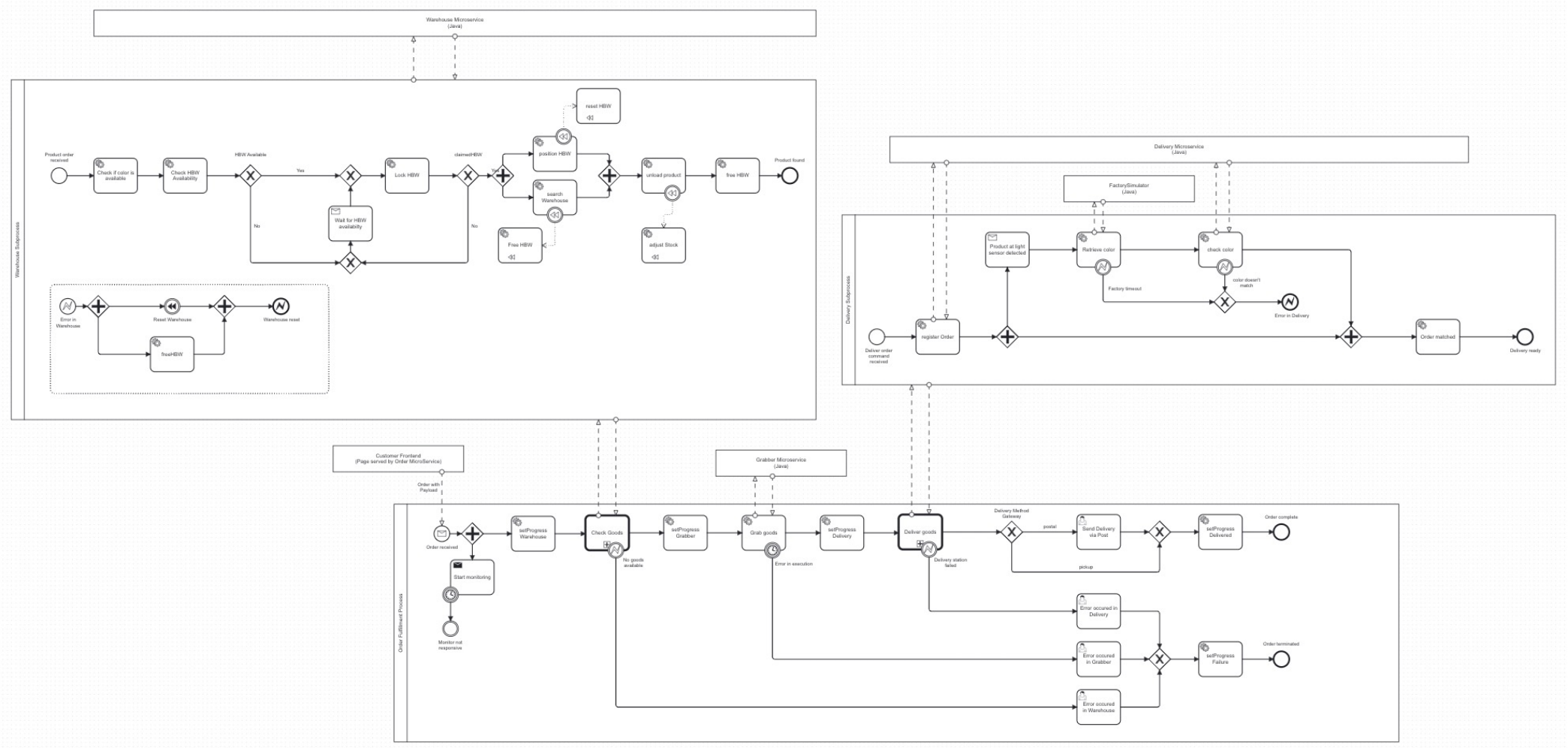
DPO
Delivery & Pickup



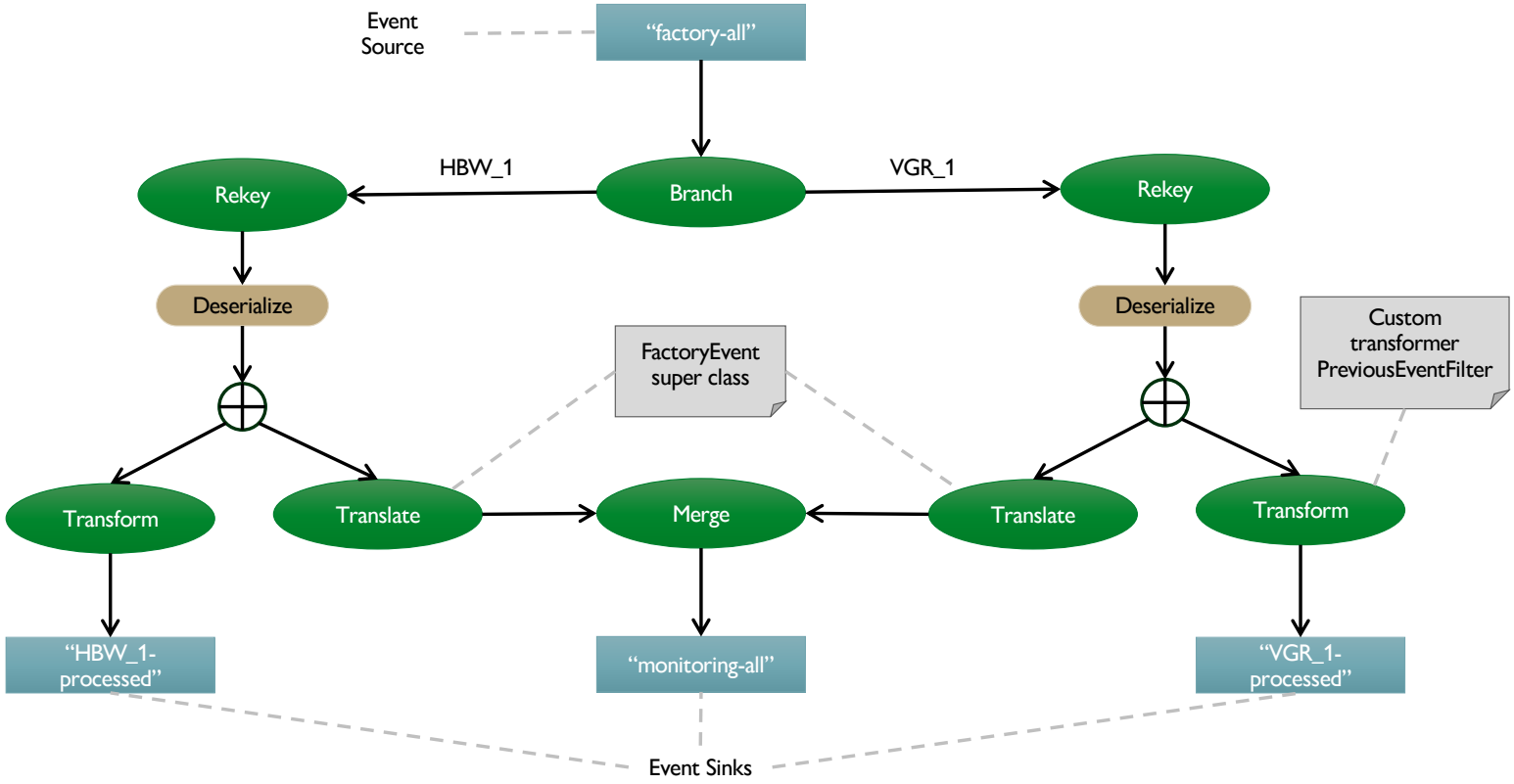


Holistic View

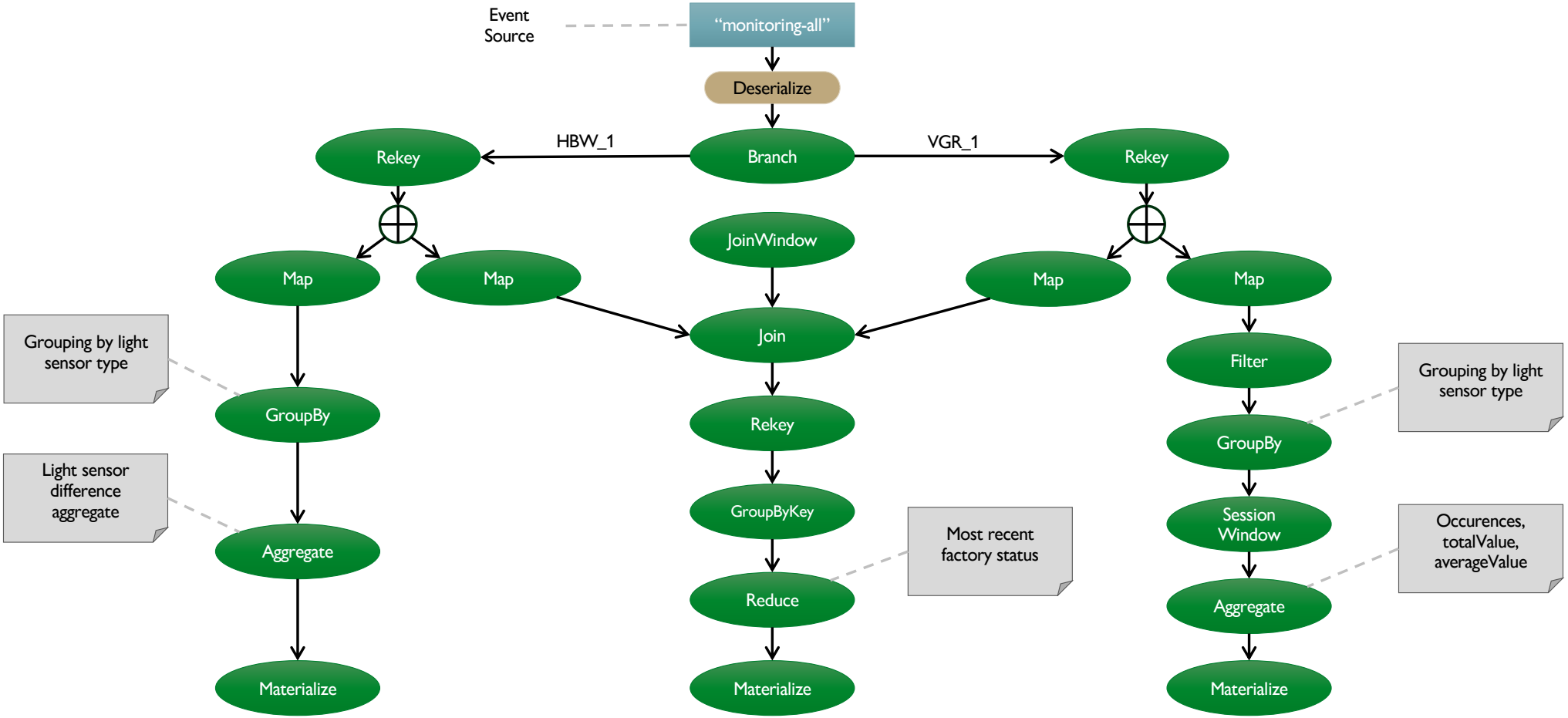



Warehouse Microservice (Java)
Warehouse Subprocess
Product order received
Check if color is available
Check HBW Availability
HBW Available
Yes
No
Wait for HBW availability
Lock HBW
claimedHBW
Yes
No
position HBW
search Warehouse
Free HBW
reset HBW
unload product
adjust Stock
free HBW
Product found
Error 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 Fulfillment Process
Order with Payload
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

Stream Processor – Topology

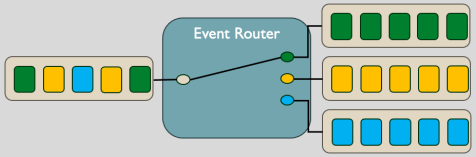


Monitoring – Topology



Stateless Processing

Routing

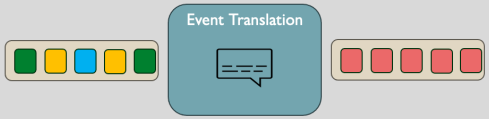


- Routing combined stream of all stations of the factory into dedicated event streams per station (branching)

Filtering

- Filtering on specific event values (i.e to only get events where light barrier is broken)

Translation



- Translation of specific events to generic events and back dependent on service specific requirements

Logging

```
[monitoring-all]: [B@5dc28a32, FactoryEvent [id=4f82c595-33fb-4ab8-b07a-1b72c3d521b4, source=FactoryListener, time=2024-05-22T16:50:50.571116060Z, data=VGR_1
[monitoring-all]: [B@3c118180, FactoryEvent [id=8c2c4529-dbfb-4e6e-9235-23d9b9d079cf, source=FactoryListener, time=2024-05-22T16:50:50.673982596Z, data=HBW_1
[monitoring-all]: [B@4a55d3cb, FactoryEvent [id=d50e0105-0492-47b3-bcad-bd984bb39eb7, source=FactoryListener, time=2024-05-22T16:50:51.185888082Z, data=VGR_1
```

- Logging of..
  - ..events to console (useful for debugging)
  - ..interesting events (i.e. light sensor broken)

Stateful Processing - Send Updates only if value changed

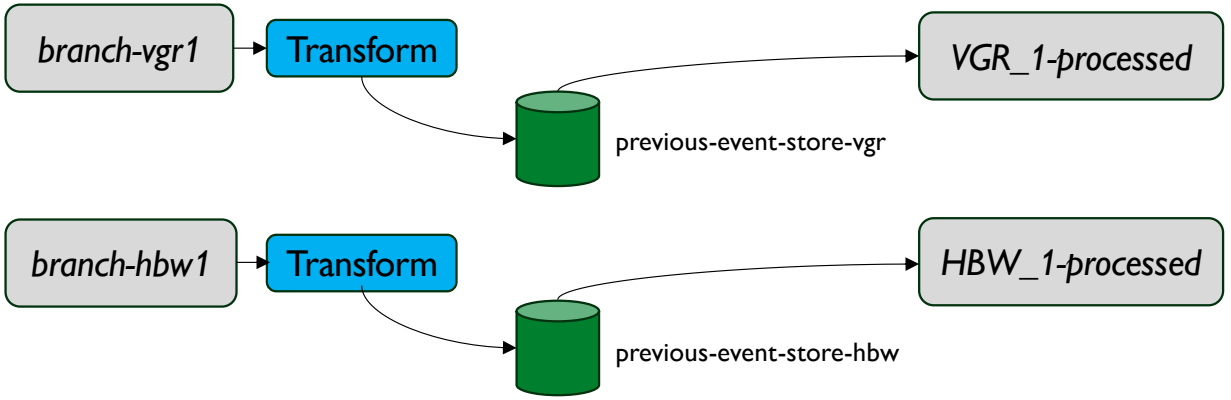
Problem:
Send only event updates that are relevant for specific services based on their needs

Approach:
Custom Transformer that compares previous event with the current event and updates it only if…

- a change in the stock or light barrier has been detected (Warehouse)
- a change in the color sensor or light barrier has been detected (Gripper)

```
// custom filtering logic to only let events through that differs from the previous one
KStream<byte[], VgrEvent>  vgrTypedFilteredStream =  vgrTypedStream
        .transform(PreviousEventFilterVGR::new, ...strings: "previous-event-store-vgr");

KStream<byte[], HbwEvent>  hbwTypedFilteredStream =  hbwTypedStream
        .transform(PreviousEventFilterHBW::new, ...strings: "previous-event-store-hbw");
```



Stateful Processing - Average Color Value

Problem:
For each Color we want to compute the average value based on measured SensorValues

Approach:
Projection of original stream to new stream that contains only relevant fields (Color, SensorValue) (stateless)

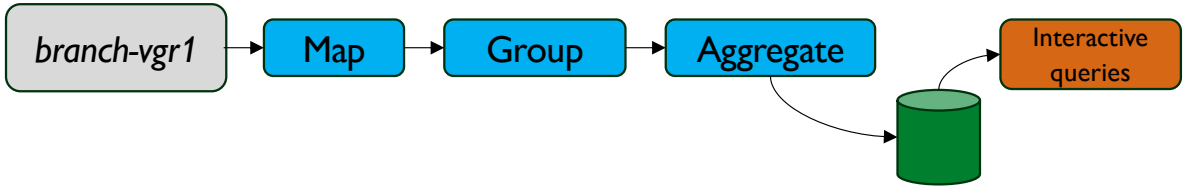
Aggregation of the (Color, SensorValue) events and in memory storage of average value per color

Interactive querying by the monitoring UI

```
// Create Stream of Color, ColorValues
KStream<String, Double> colorSensorStream = vgrTypedStream.map((key, vgrEvent) ->
        new KeyValue<>(vgrEvent.getData().getColor(), vgrEvent.getData().getI8_color_sensor())
);

// KTable that exposes the ColorStats for each color
colorSensorStream
        .groupBy((key, value) -> key, Grouped.with(Serdes.String(), Serdes.Double()))
        .aggregate(
                ColorStats::new,
                ColorStats::aggregate,
                Materialized.<>
                                as( storeName: "colorStats")
                        .withKeySerde(Serdes.String())
                        .withValueSerde(colorStatsSerde));
```

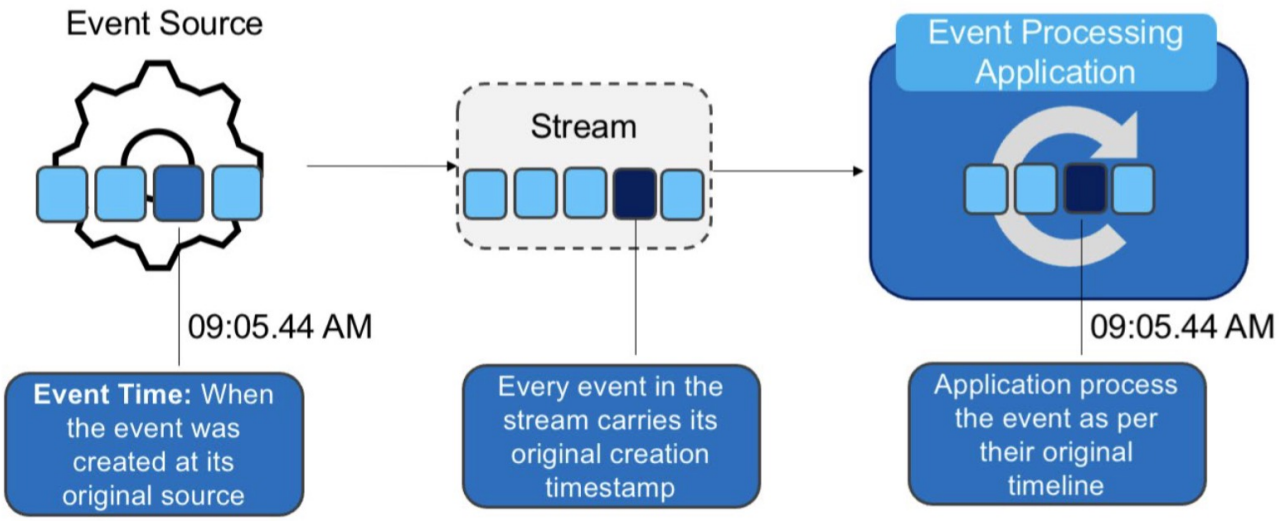
Red	Blue	White	None
Total Readings: 528	Total Readings: 64	Total Readings: 0	Total Readings: 4
Total Color Values: 753233	Total Color Values: 96959	Total Color Values: 0.0	Total Color Values: 6242
Average Color Value: 1426.58	Average Color Value: 1514.98	Average Color Value: 0.0	Average Color Value: 1560.50



Stateful Processing – Time

Time semantics

- Event time used for joining individual station streams
- Utilized custom timestamp extractors



Stateful Processing – Time

Timestamp Extractor – Example 1

StreamProcessor Service

- Consumes raw data from factory
- timestamp format conformity:
  - read raw data
  - extract timestamp with regex
  - parse timestamp to type «Instant»

```java
public class CustomTimestampExtractor implements TimestampExtractor {

    private static final Pattern PATTERN = Pattern.compile( regex: "timestamp=(.*?),");
    private static final DateTimeFormatter FORMATTER = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SS");

    @Override
    public long extract(ConsumerRecord<Object, Object> record, long previousTimestamp) {
        Matcher matcher = PATTERN.matcher(record.value().toString());
        if (matcher.find()) {
            String timestampStr = matcher.group(1);
            if (!timestampStr.isEmpty()) {
                LocalDateTime dateTime = LocalDateTime.parse(timestampStr, FORMATTER);
                Instant timestamp = dateTime.toInstant(ZoneOffset.UTC);
                return timestamp.toEpochMilli();
            }
        }
        return System.currentTimeMillis();
    }
}
```

Stateful Processing – Time

Timestamp Extractor – Example 2

Monitoring Service

- Consumes typed serialized data from StreamProcessor
- Multiple possible event types
- Deserialize depending on type
- Timestamp Extractor casts event accordingly to access the event time

```java
@Override
public Deserializer<Object> deserializer() {
    return (topic, data) -> {
        if (data == null) {
            return null;
        }

        String json = new String(data);
        if (json.contains("VGR_1")) {
            return vgrEventDeserializer.deserialize(topic, data);
        } else if (json.contains("HBW_1")) {
            return hbwEventDeserializer.deserialize(topic, data);
        } else {
            throw new IllegalArgumentException("Unknown event type: " + json);
        }
    };
}
```

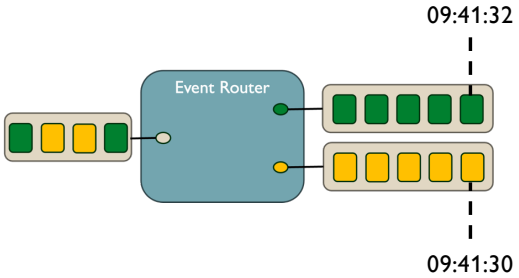
```java
@Override
public long extract(ConsumerRecord<Object, Object> record, long previousTimestamp) {
    Object event = record.value();

    if (event instanceof VgrEvent vgrEvent) {
        return vgrEvent.getData().getTimestamp().toEpochMilli();
    } else if (event instanceof HbwEvent hbwEvent) {
        return hbwEvent.getData().getTimestamp().toEpochMilli();
    } else {
        throw new IllegalArgumentException("Unknown event type: " + event.getClass());
    }
}
```

Stateful Processing – Join

Goal

- Get status of the complete factory
- Join events from Warehouse and Gripper service
- Based on timestamps



1. Define key of event in steps of 10 seconds

```
KStream<String, VgrEvent> vgrTimestampKey = vgrTypedStream.map((key, value) -> {
        long epochSecond = value.getData().getTimestamp().getEpochSecond();
        long keySecond = epochSecond - epochSecond % 10;
        return new KeyValue<>(Long.toString(keySecond), value);
});
```

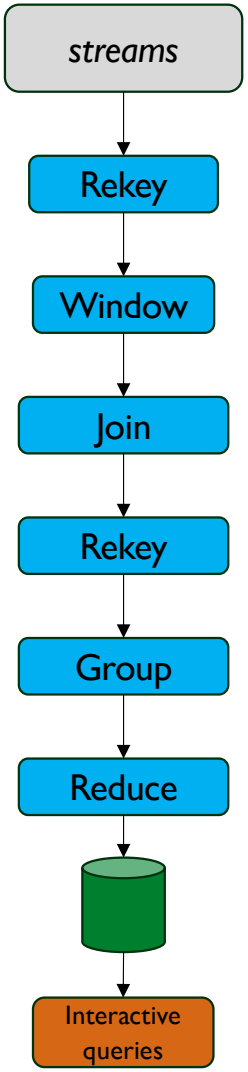
2. Specify JoinParams + JoinWindow

3. When joining, create new FactoryStats instance and rekey to “factoryStats”

```
KStream<String, FactoryStats> factoryStatsStream =
        vgrTimestampKey.join(
                hbwTimestampKey,
                FactoryStats::new,
                joinWindows,
                joinParams
        ).selectKey((k, v) -> "factoryStats");
```

4. Select most recent record; materialize in KeyValueStore

```
factoryStatsStream
        .groupByKey(Grouped.with(Serdes.String(), factoryStatsSerde))
        .reduce((aggValue, newValue) -> newValue,
                Materialized.<String, FactoryStats, KeyValueStore<Bytes, byte[]>>
                        as( storeName: "factoryStats")
                        .withKeySerde(Serdes.String())
                        .withValueSerde(factoryStatsSerde));
```



Stateful Processing – Windows

Problem:
To optimize the process of retrieving a product, we need to know where the product might spend an extended period of time.

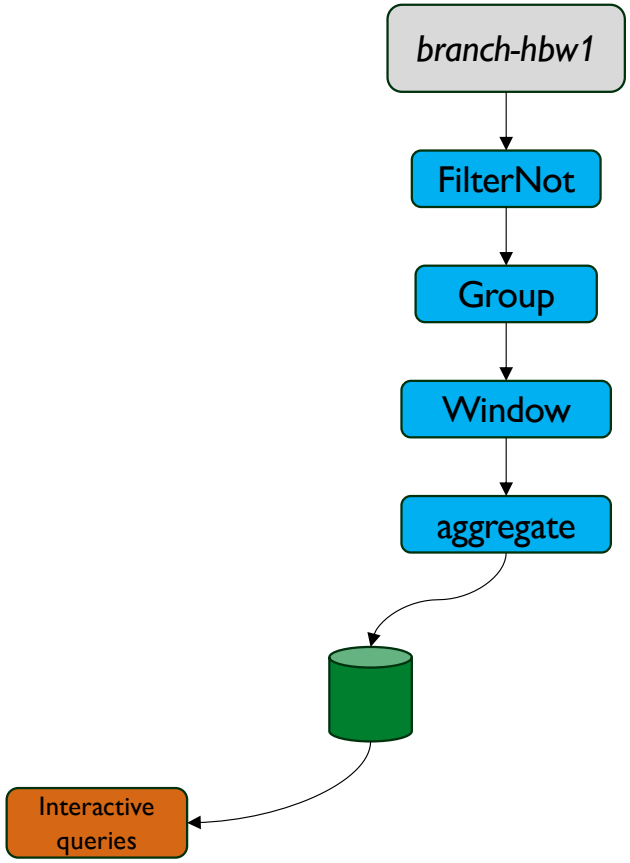
Approach:
Filter a new stream that only hold events with broken light barriers.

Create session windows based on stream (in-) activity.

Group by the broken light sensor.

Aggregation into custom data type that holds start and end time of light sensor disconnect

Interactive querying by the monitoring UI



…
i2_light_barrier=false,
i3_light_barrier=false,
i4_light_barrier=true,
i5_pos_switch=false,
i6_pos_switch=true,
i7_pos_switch=false,
i8_pos_switch=false,
…

Stateful Processing – Windows

- Not all light barriers are relevant for our use case

```
// Stream of events with broken light barriers
KStream<String, HbwEvent> lightBarrierBrokenHBW = hbwTypedStream.filterNot((k, v) ->
        v.getData().isI1_light_barrier() && v.getData().isI4_light_barrier());
```

- Convenient logging for immediate feedback
- Appropriate gap and grace durations

```
// grouped by light barrier and windowed by session
SessionWindowedKStream<String, HbwEvent> sessionizedHbwEvent = lightBarrierBrokenHBW
        .groupBy((k, v) -> {
            String sensorKey = "unknown";
            if (!v.getData().isI4_light_barrier()) {
                sensorKey = "i4_light_sensor";
            } else if (!v.getData().isI1_light_barrier()) {
                sensorKey = "i1_light_sensor";
            }
            System.out.println("LightSensor broken: " + sensorKey);
            return sensorKey;
        }, Grouped.with(Serdes.String(), hbwEventSerdes))
        .windowedBy(SessionWindows.ofInactivityGapAndGrace(Duration.ofSeconds(2),Duration.ofMillis(500)));
```

Stateful Processing – Windows

```
// aggregated by timedifference of each session
sessionizedHbwEvent.aggregate(
        TimeDifferenceAggregation::new,
        TimeDifferenceAggregation::add,
        TimeDifferenceAggregation::merge,
        Materialized.<~>as( storeName: "lightSensor")
                .withKeySerde(Serdes.String())
                .withValueSerde(timeDifferenceAggregationSerde)
                .withCachingEnabled()
        ).suppress(Suppressed.untilWindowCloses(Suppressed.BufferConfig.unbounded().shutDownWhenFull()));
```

- **Custom Aggregation Class** to enable aggregation of events into single Object that holds start and end time for a session
- **Supression of result** since we want to know the actual times spent and no intermediary results
- **Is queriable through REST API and displayed in frontend**

Team Contributions

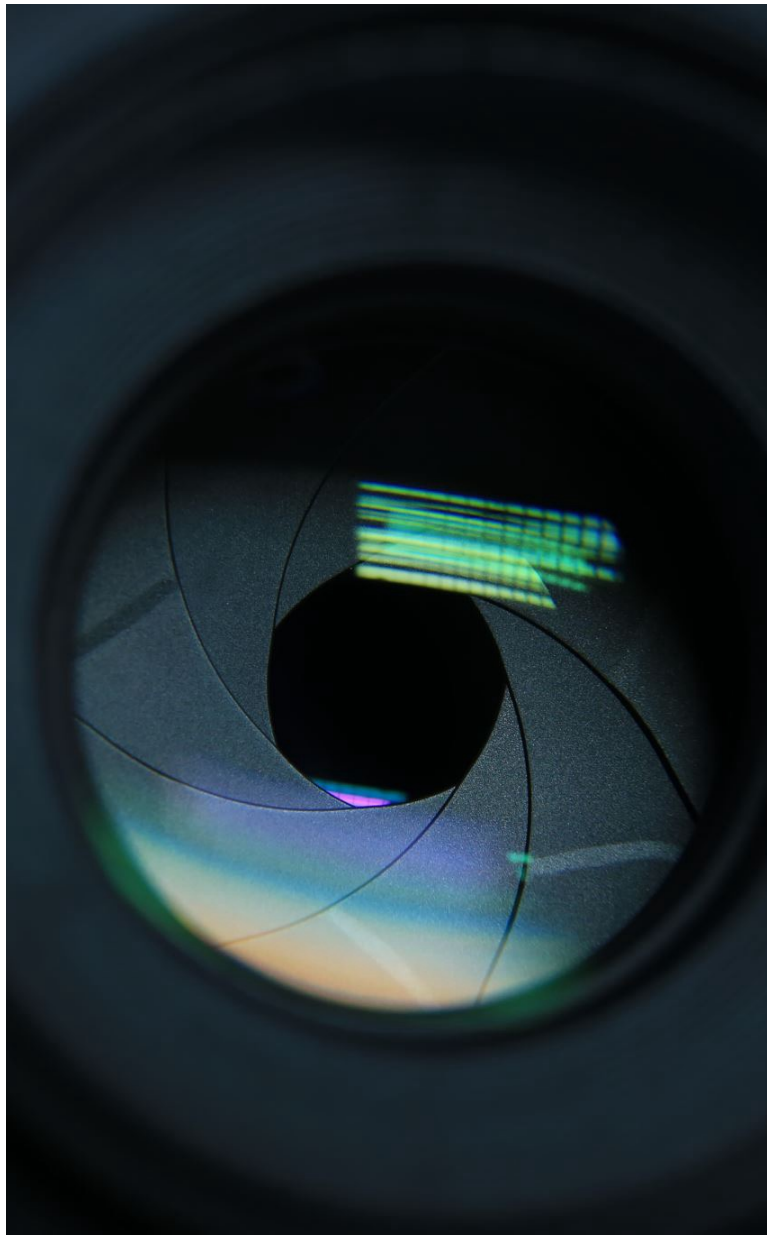
All tasks for this project have been done collaboratively and have been equally distributed among all team members.

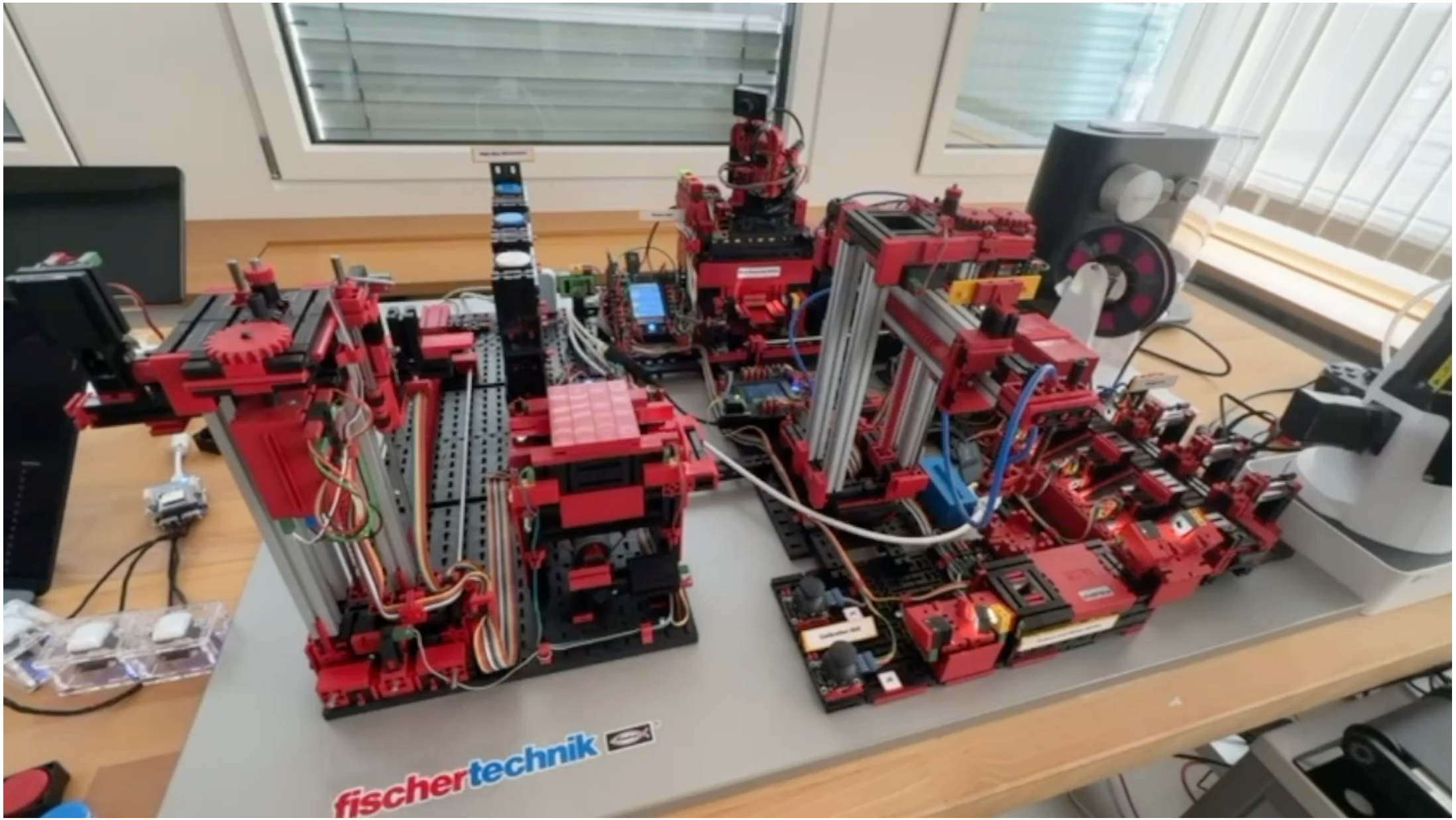
Universität St.Gallen

Demo

Stream Processing

From insight to impact.




fischertechnik