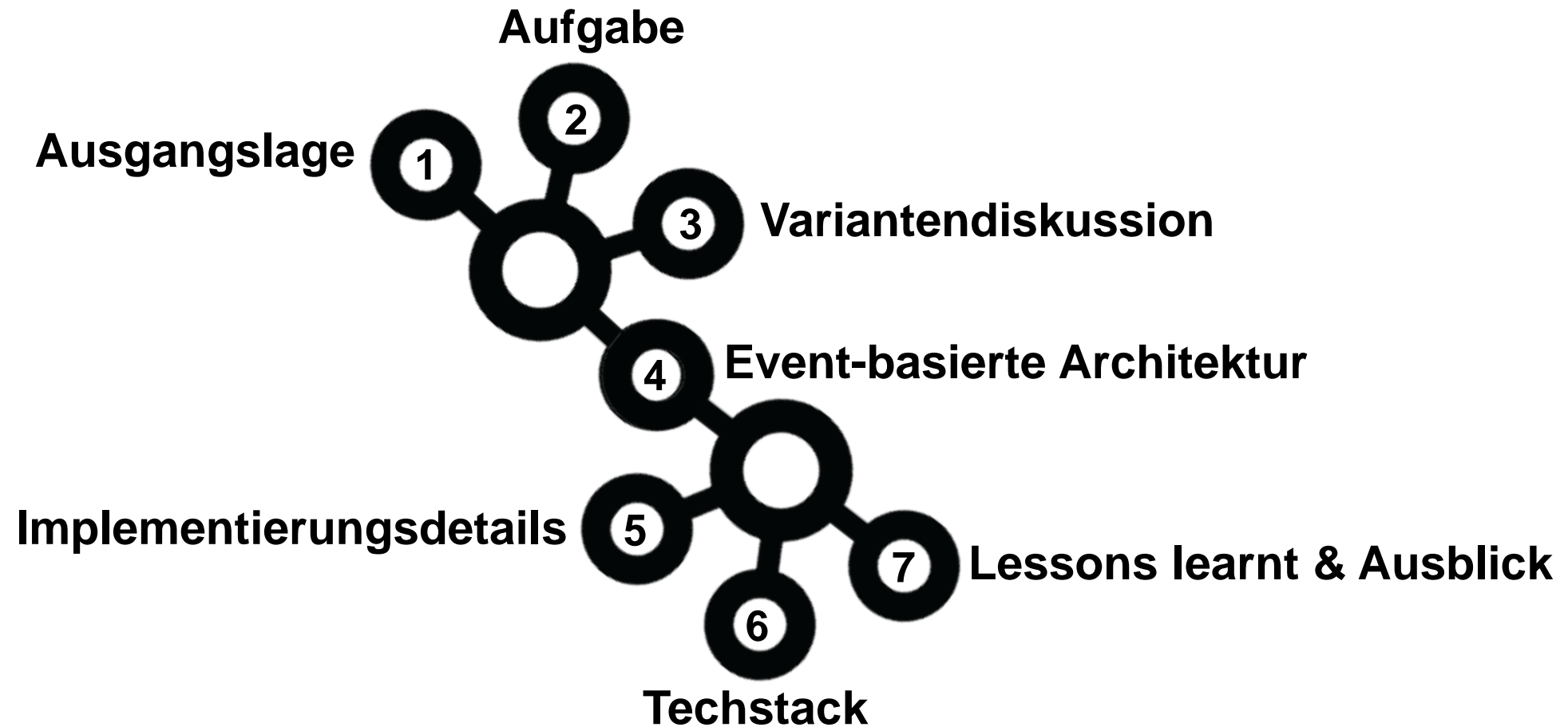


# Event-basierte Architektur mit Apache Kafka

22. Jan. 2019 @jugch  
Korhan Gülseven  
Renato Löffel





# Ausgangslage







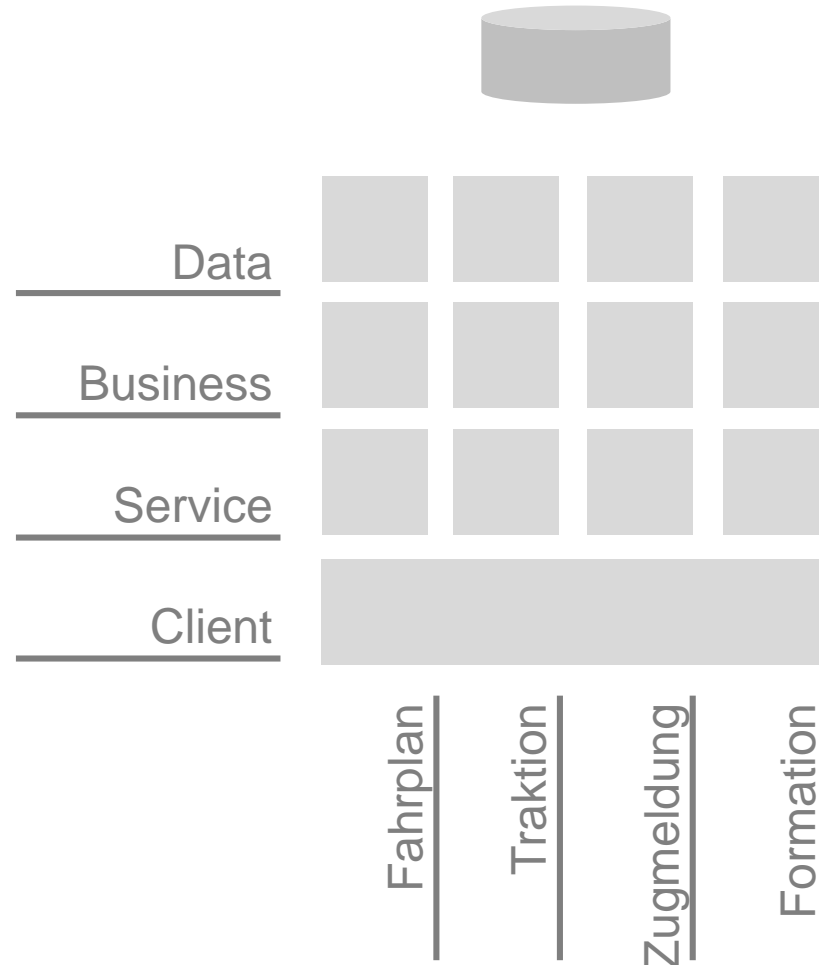
## **Aufgabenstellung**

Ablösung der Produktionsdurchführung CIS Infra

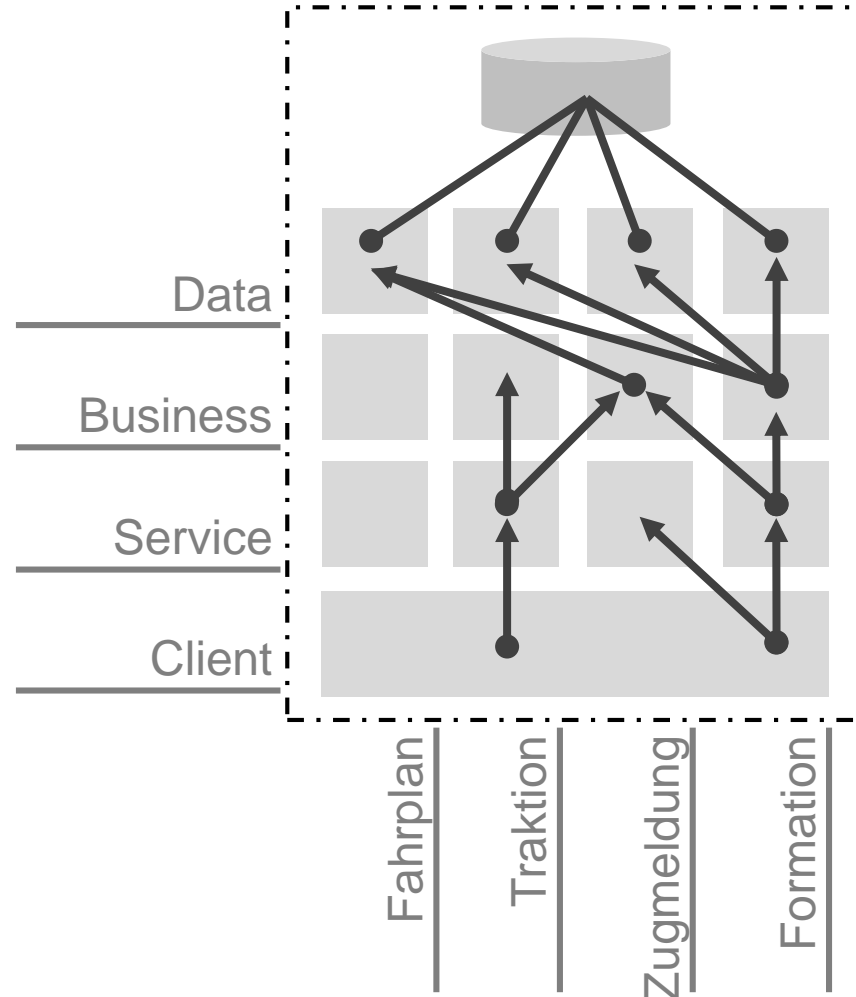
### **Top Qualitätsziele**

1. Änderbarkeit / Wartbarkeit
2. Performance
3. Zuverlässigkeit / Verfügbarkeit

# Monolith



# Monolith



# Monolith Challenging



## Änderbarkeit / Wartbarkeit

- Degeneriert
- Gewünschtes ist schwierig, Ungewünschtes ist einfach
- Trägheit



## Performance

- Kompromiss bei Datenmodell
- Unterschiedliche NFA (Realtime ↔ Reporting)

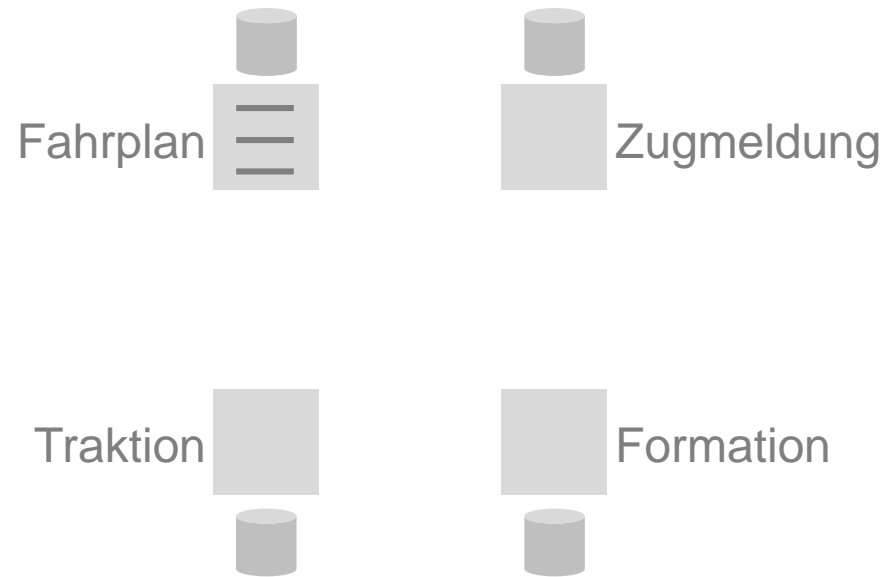


## Zuverlässigkeit / Verfügbarkeit

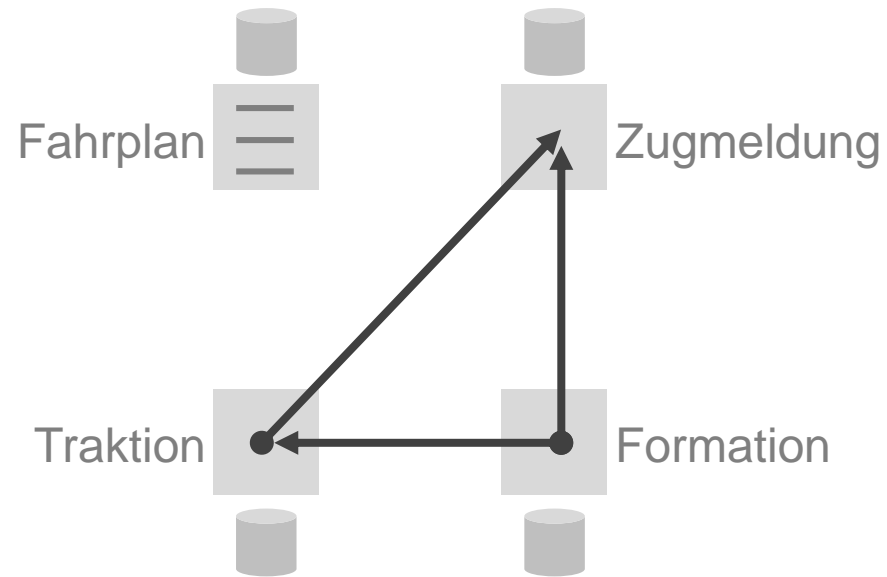
- Lokale Fehler können das gesamte System lahmlegen
- Alles oder Nichts



# Microservice Request/Response



# Microservice Request/Response



# Microservice Request/Response Challenging



## Änderbarkeit / Wartbarkeit

- Starke Isolation
- Kleine Einheiten



## Performance

- Innerhalb eines Moduls optimal
- Modulübergreifend schwach, Response-/Latenzzeiten werden kumuliert

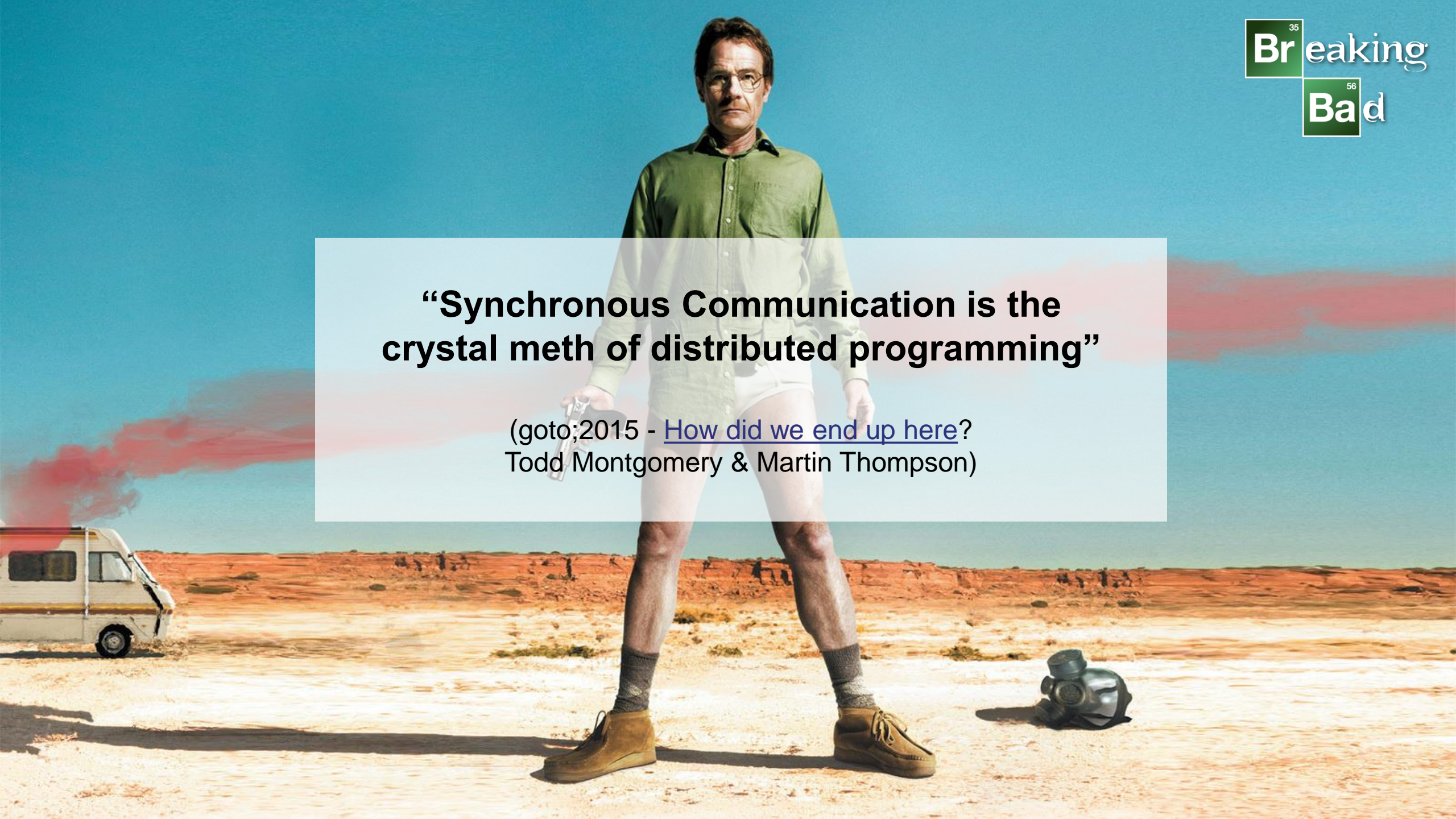


## Zuverlässigkeit / Verfügbarkeit

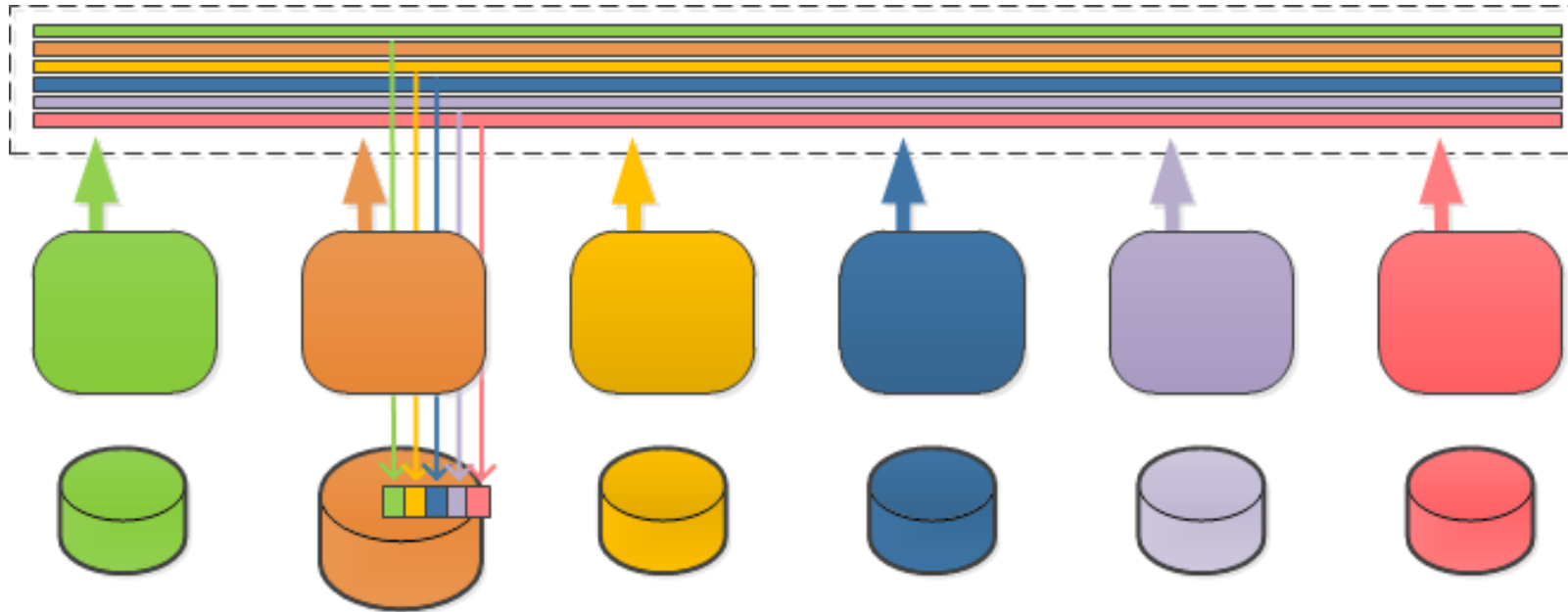
- Verfügbarkeit eines Moduls ist direkt von anderen abhängig
- Daten sind nicht da, wo sie benötigt werden.

**“Synchronous Communication is the  
crystal meth of distributed programming”**

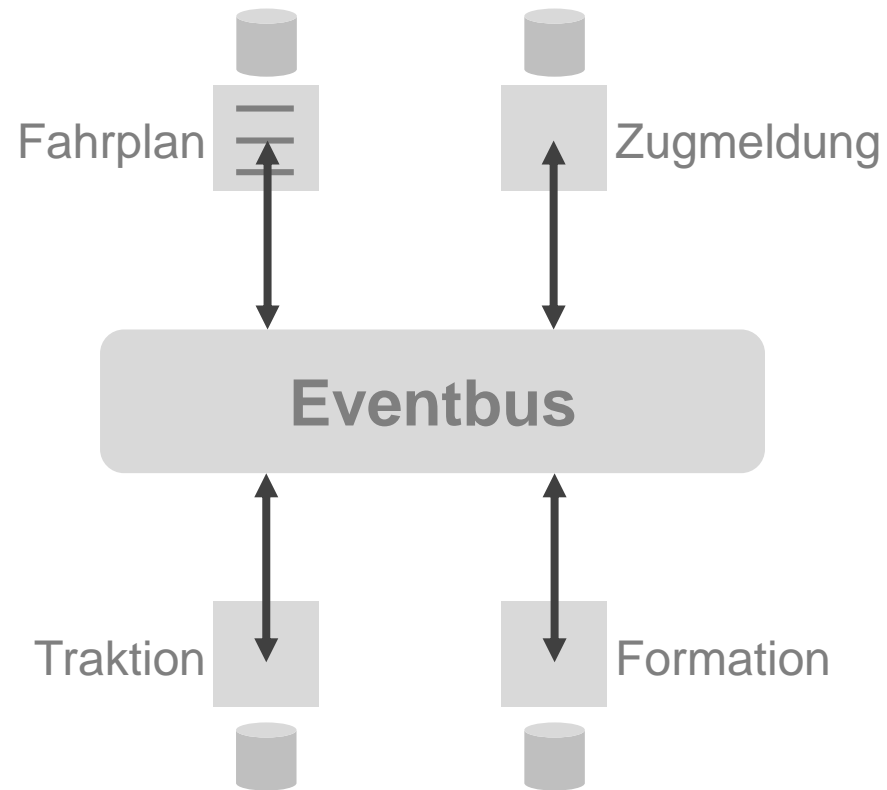
(goto;2015 - [How did we end up here?](#)  
Todd Montgomery & Martin Thompson)



# Exkurs: event-basierte Integration



# Microservice event-basiert



# Microservice event-basiert Challenging



## Änderbarkeit / Wartbarkeit

- Analog zu Microservice Request/Response



## Performance

- Realtime-Event-Processing
- Laufende Datenverteilung



## Zuverlässigkeit / Verfügbarkeit

- Maximale Isolierung in der Laufzeit

# Event-basierte Architektur

Maximale Entkopplung (Raum und Zeit)



# Event-basierte Architektur

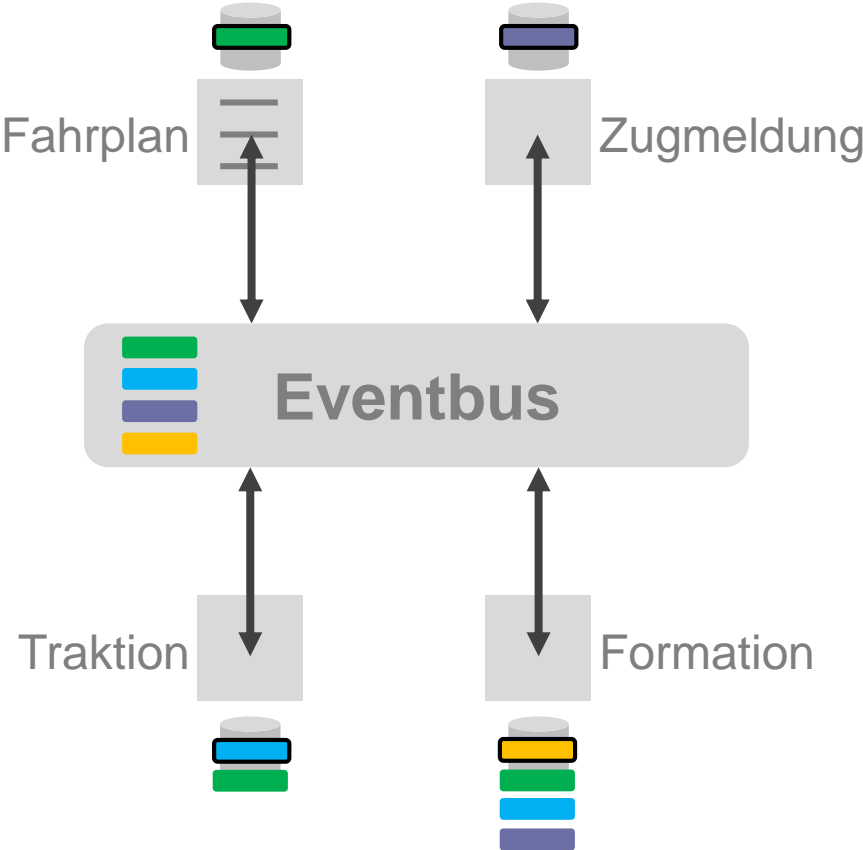
## **Verteiltes System**


- CAP
- Keine verteilten Transaktionen
- Keine Garantien (Reihenfolge, Duplikat)
- Versionierung


## **Zustand**

- Redundanz

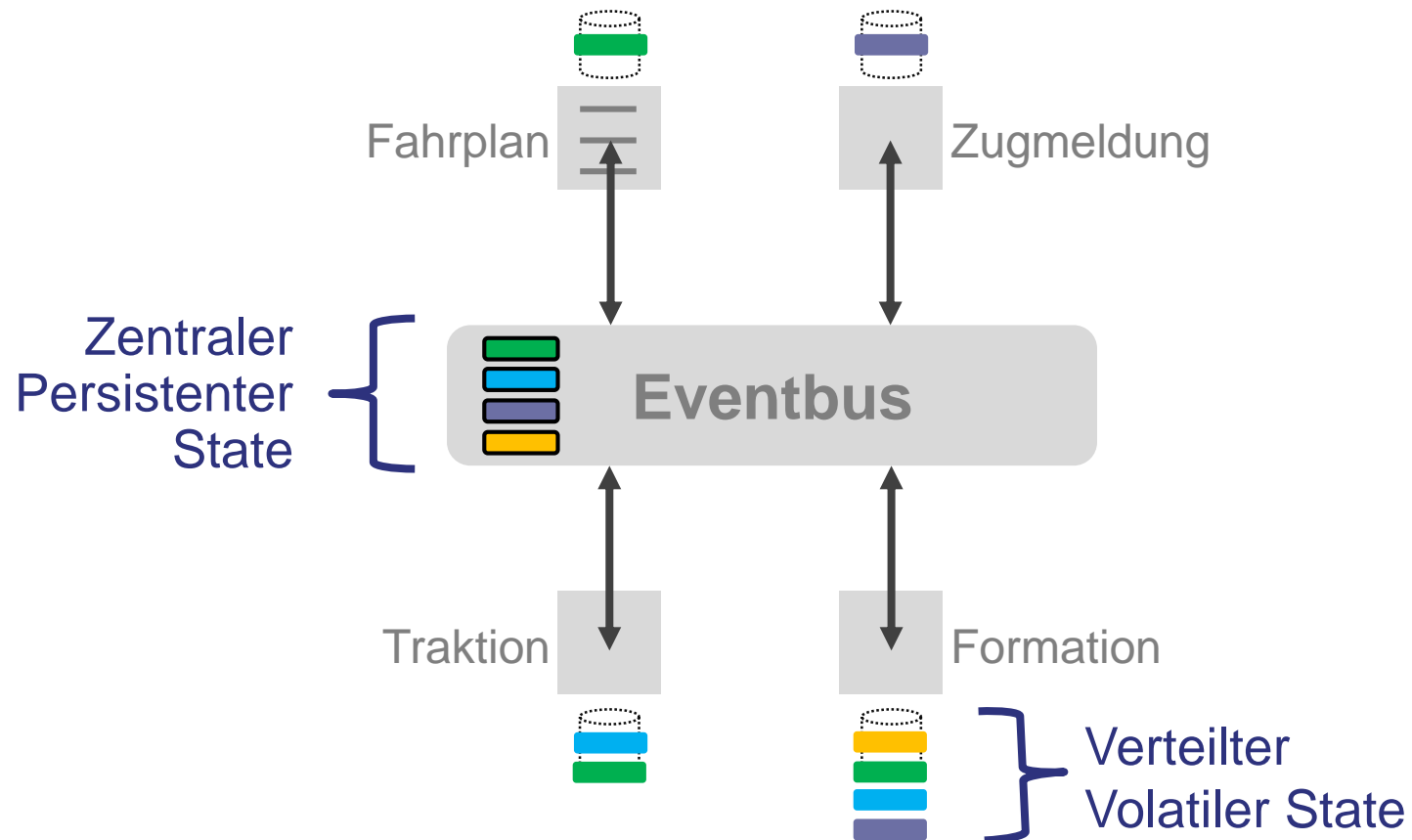
# Verteilter State



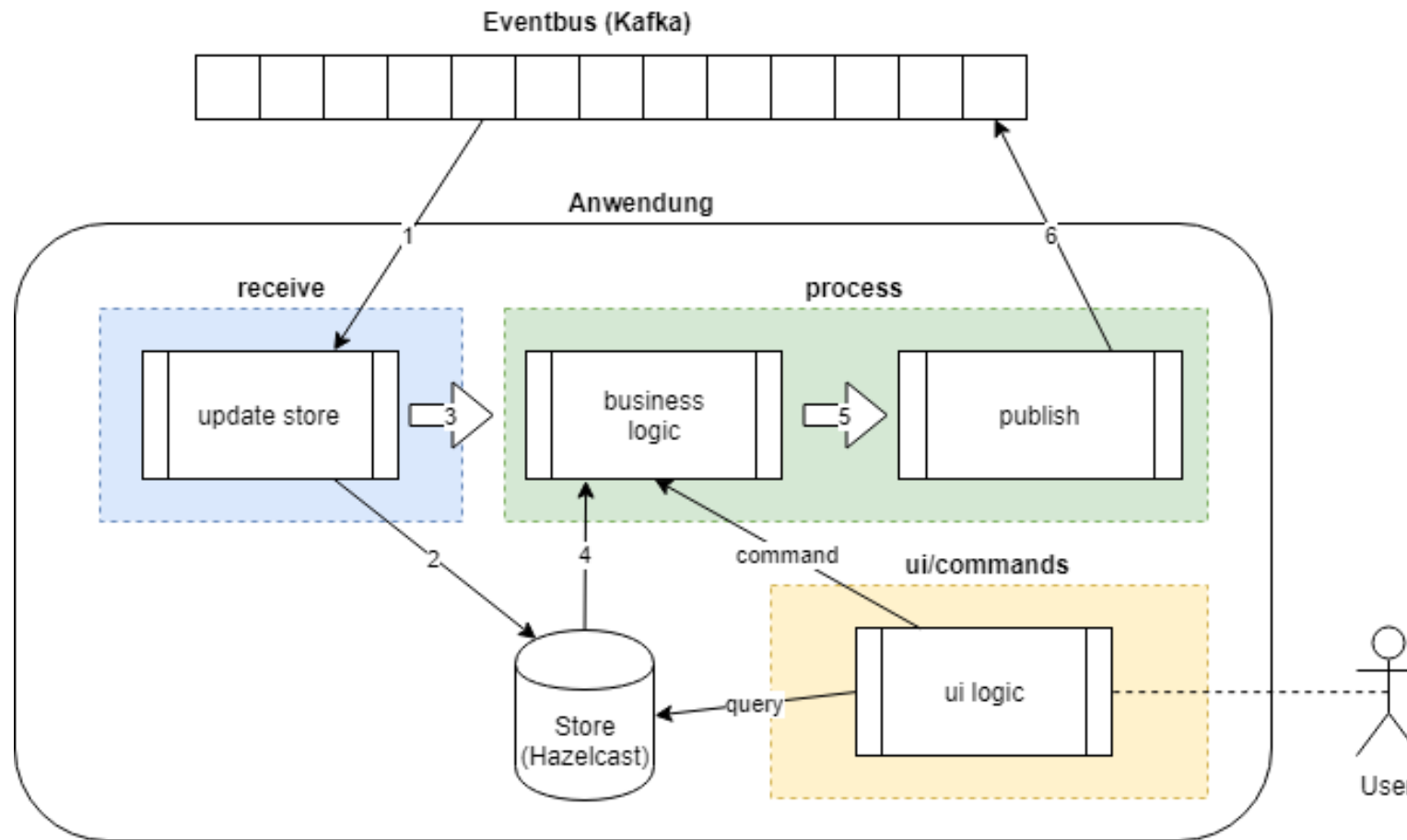
 Primary  
↓  
 Replica

 Änderung im Primary muss in allen Replicas nachgeführt werden!

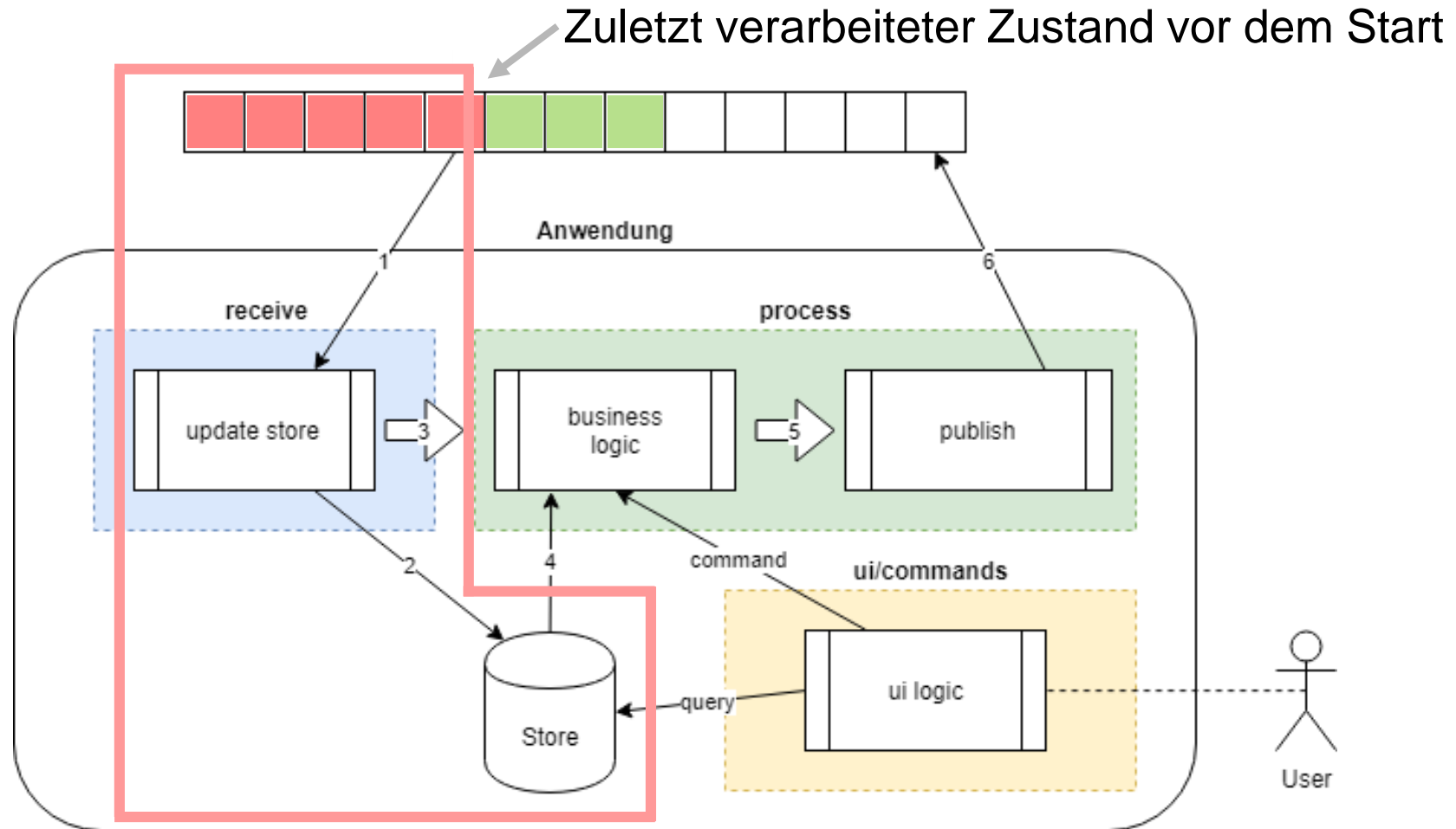
# Zentraler State – Single Source of Truth



# Verarbeitungsmodell

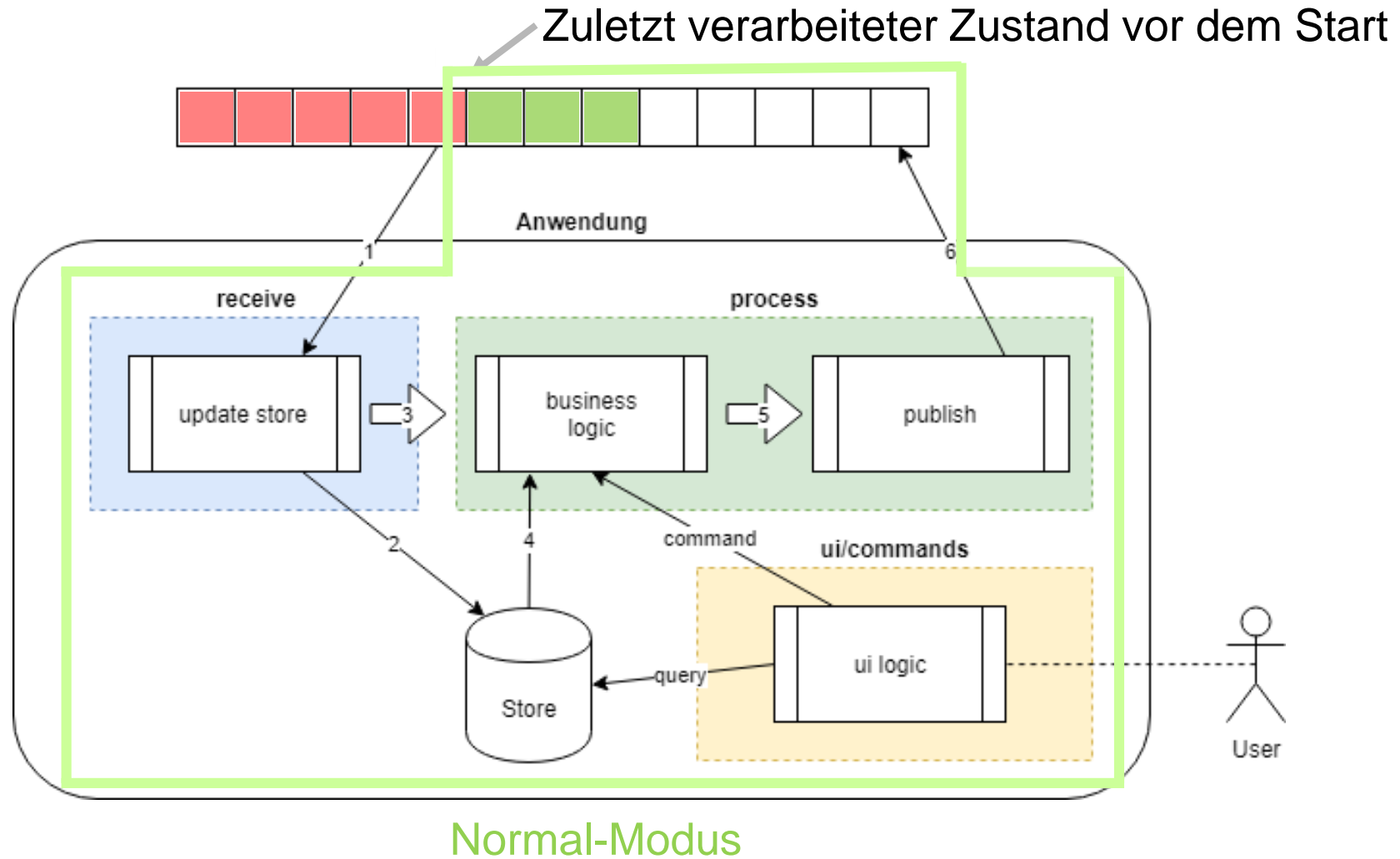


# Verarbeitungsmodell: State Recovery nach einem Cold-Start



Recovery-Modus

# Verarbeitungsmodell: State Recovery nach einem Cold-Start



Boilerplate «Kafka/Event-Sourcing» in ein Framework extrahiert.

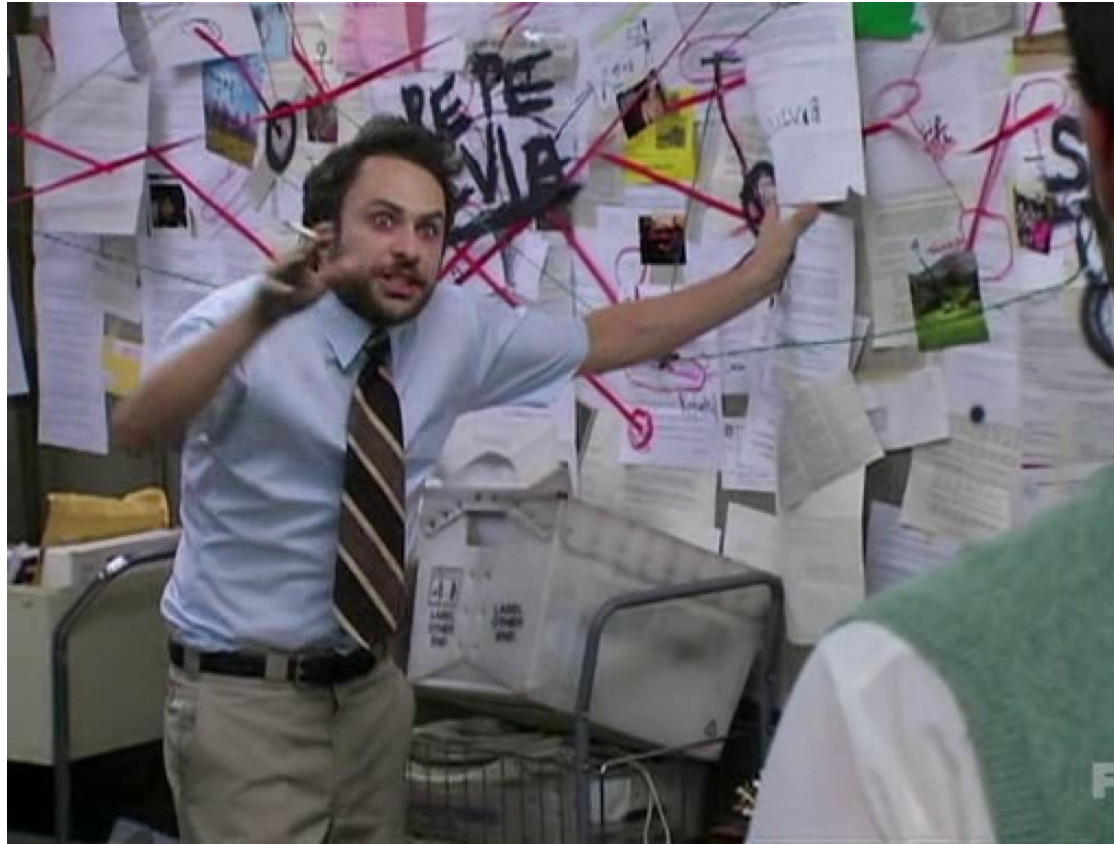
- Recovery Detection
- Orchestrierung receive/process
- Duplicate Check
- Concurrency
  - ~~Optimistic Locking~~
  - ~~Pessimistic Locking~~ } Kafka!
- Fehlerhandling → DeadLetter Topic
- Monitoring/Tracing

# Concurrency: Optimistic Locking

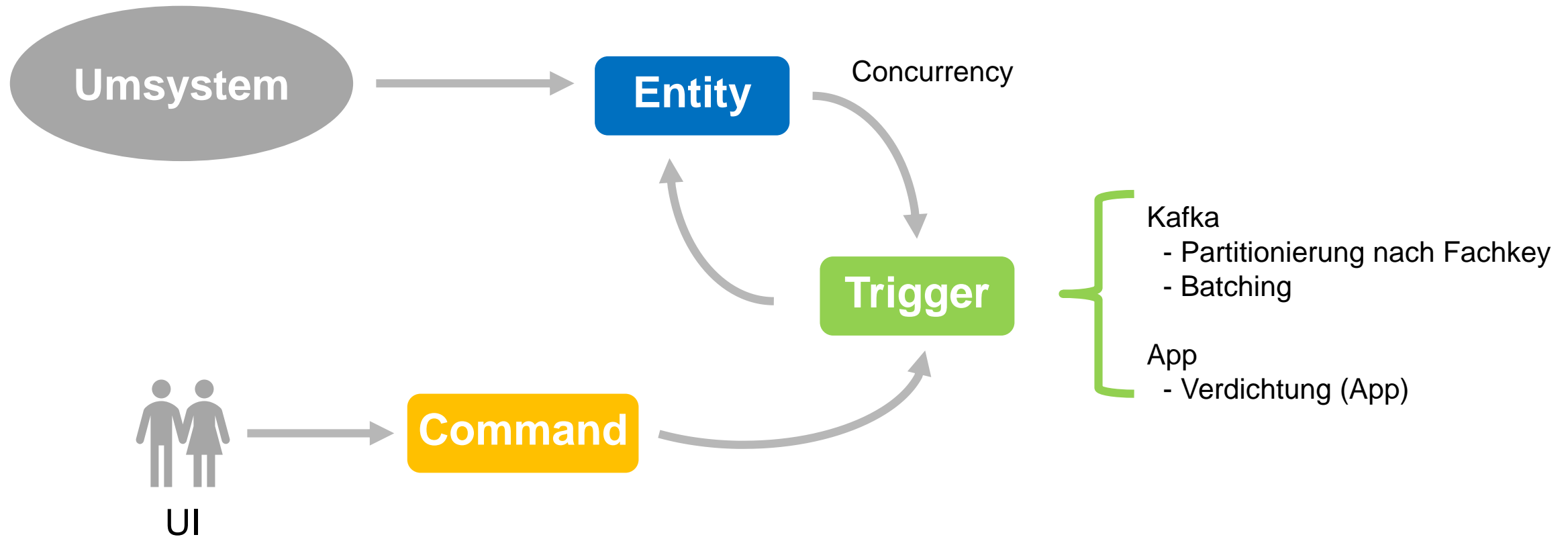




## Concurrency: Pessimistic Locking



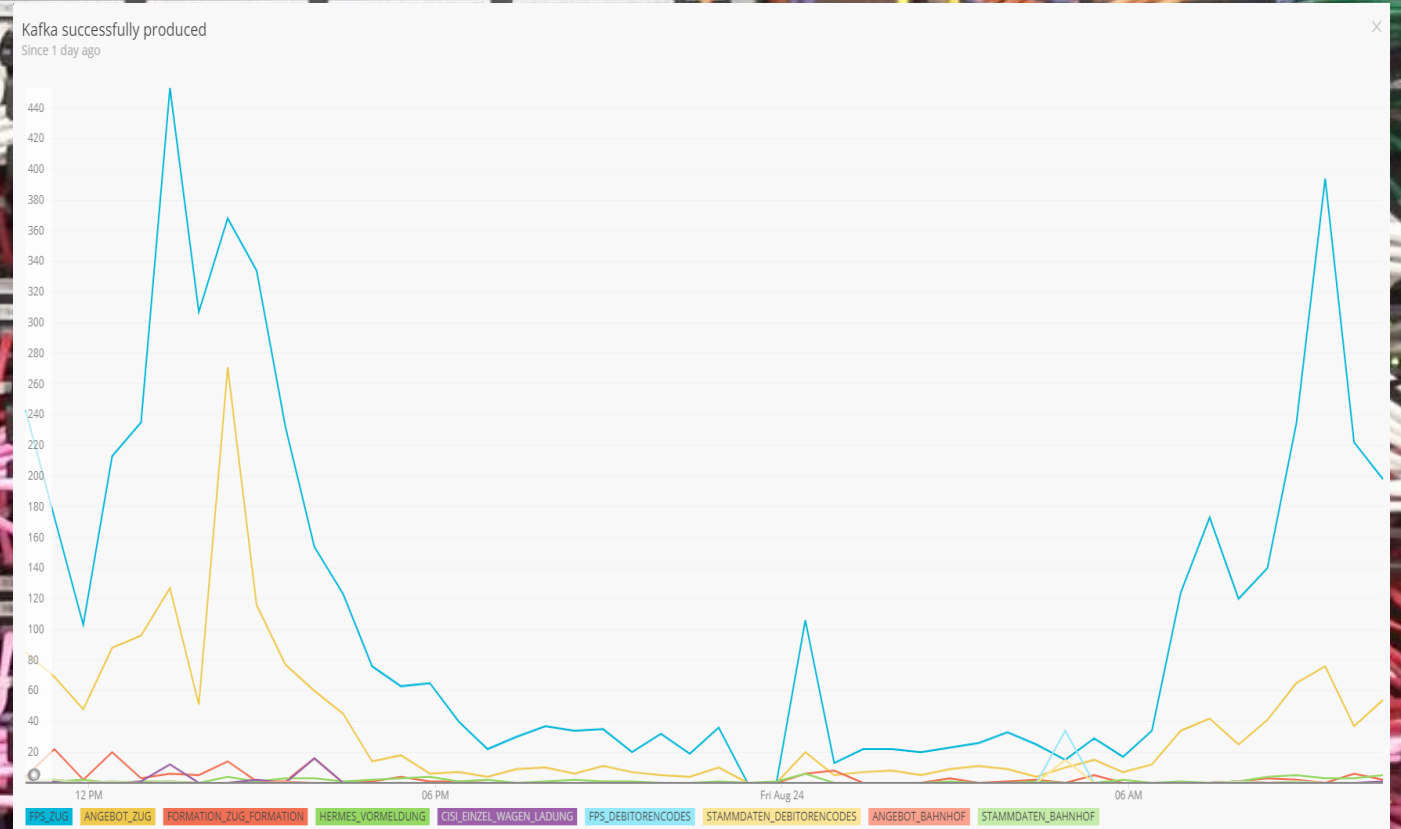
# Concurrency: Kafka Partitioning



## Techstack

- Kafka Cluster auf OpenShift
- Hazelcast Embedded/Clustered
- Spring Kafka
- New Relic
- Zipkin

- fps	.	011.846ms
- angebot	.	.453.914m
- angebot	.	.57.784ms
- angebot	.	08.647ms
formation	.	.972.188n





## Techstack: Benefits

### Kafka

- Persistenz & Messaging
- Skalierung mittels Partitionen
- Streaming Plattform
- Keine Datentransformation
- Technisches vs Fachliches Log (Retention/ Compaction)

### Hazelcast

- Schnell (In Memory)
- Keine Datentransformation
- Natürlicher Partner mit Kafka (Key, Value)
- Reduzierte Query Möglichkeiten

### Spring Kafka

- Kafka Consumer/Producer the spring way

# Lessons Learnt & Ausblick



- Kafka Streaming & Interactive Queries
- Aufbau & Betrieb Kafka (Projektrisiko)
- Hazelcast Query Tool?



- Agilität der event-basierten Architektur
- Stabilität & Ökosystem & Toolunterstützung
- Performance
- Produktive Daten

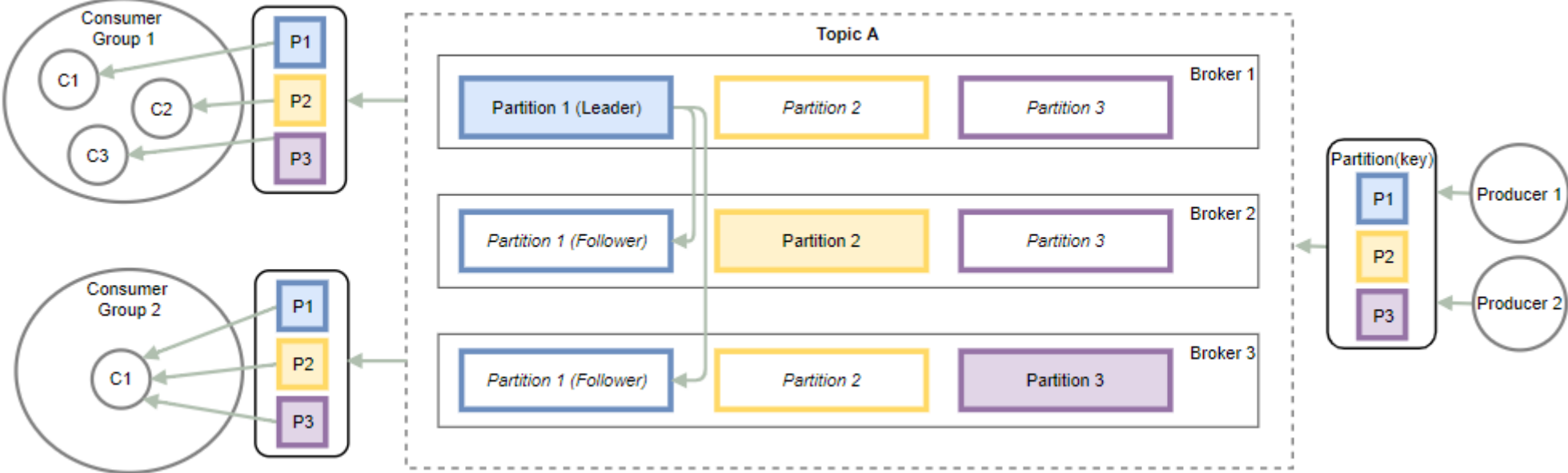
# Fragen

→ Kontakt:

- [korhan.guelseven@sbb.ch](mailto:korhan.guelseven@sbb.ch)
- [renato.loeffel@adesso.ch](mailto:renato.loeffel@adesso.ch)

# Backup

# Kafka: Skalierung & Failover





# Concurrency & Optimistic Locking

