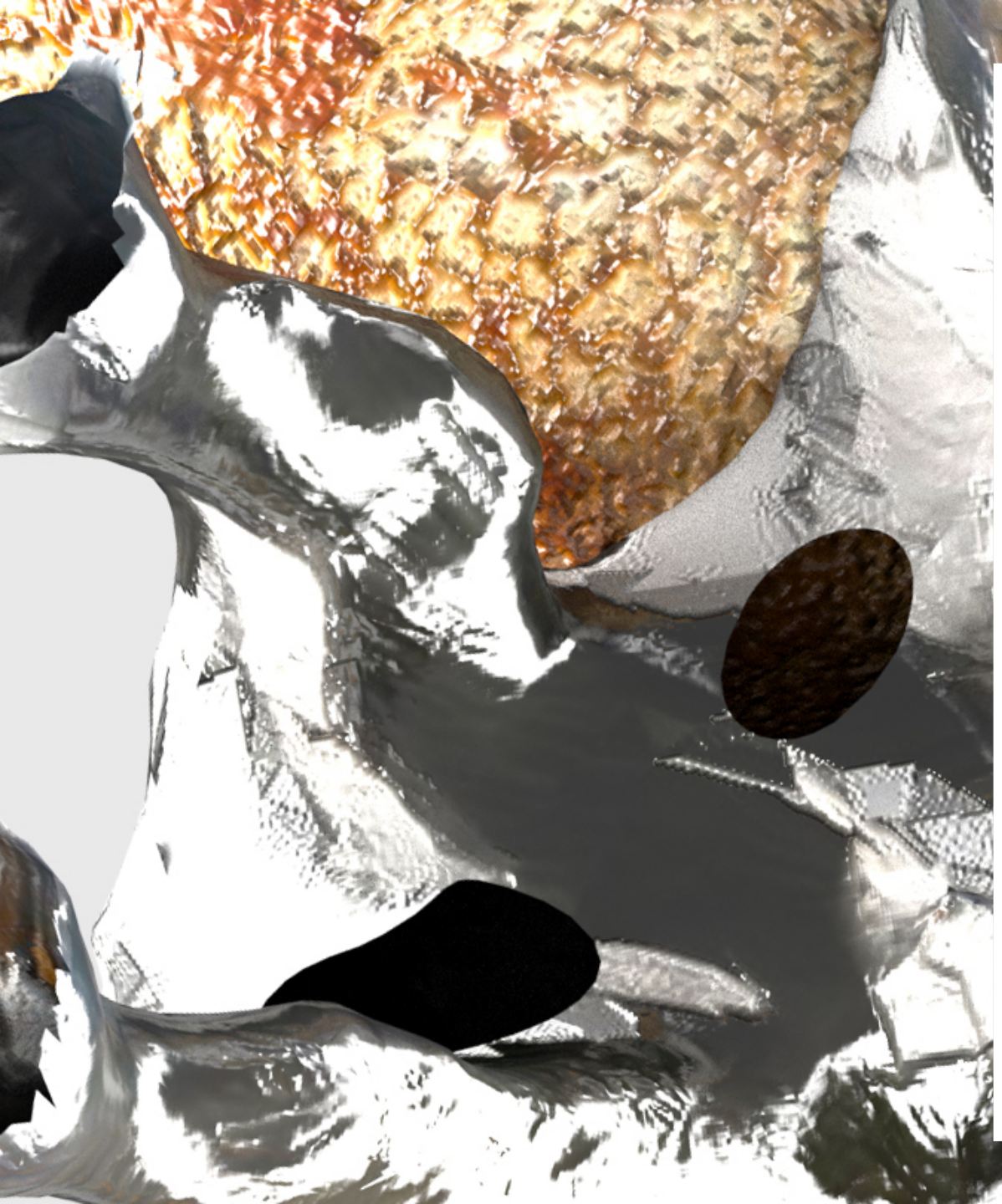


ZÜRICH / 05.03.2020

JAVA USER GROUP SWITZERLAND

EVENT SOURCING UND CQRS

ZWEI KONZEPTE ERKLÄRT UND ENTWIRRT

Christian Stettler

INNOQ Schweiz

INNOQ

iSAQB® ACCREDITED Trainer

CHRISTIAN STETTLER

Senior Consultant / Architekt, INNOQ Schweiz
christian.stettler@innoq.com

Christian ist Senior Consultant und Architekt bei INNOQ in der Schweiz. Seit über 15 Jahren entwickelt er mit Leidenschaft fachlich relevante Systeme mit verteilten, skalierbaren Architekturen. Als DDD-Enthusiast gilt sein Interesse dabei vor allem den Ansätzen von Domain-Driven Design, um ein umfassendes fachliches Verständnis zu erlangen und dieses möglichst verständlich in Code zu giessen. Bei Gelegenheit gibt er seine Erfahrungen in Workshops und Trainings weiter.

Ziele

Event Sourcing und CQRS als Konzepte «sauber» verstehen

Abgrenzung zu Event-* machen können

Konsequenzen beim Einsatz Event Sourcing und CQRS kennen

Event Sourcing

Zustand = ∑ Ereignisse

Event Sourcing in Mesopotamien (7000 BC)

Quelle: Wikipedia - https://upload.wikimedia.org/wikipedia/commons/c/c2/Plimpton_322.jpg

Event vs. Command

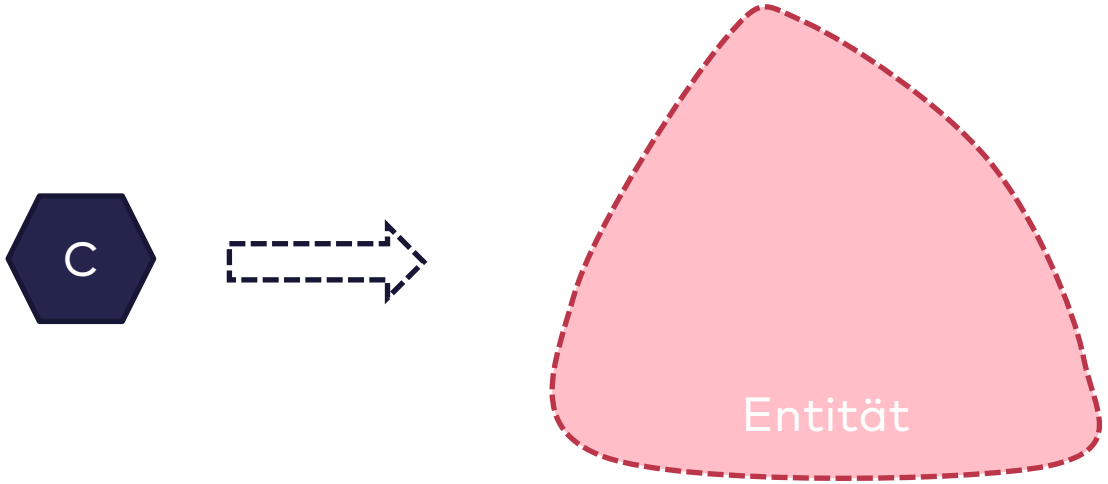
Event

- Repräsentiert geschehene Tatsache
- Hat keine Intention
- Kann nicht mehr «ungeschehen» gemacht werden
- Kann nur kompensiert werden

Command

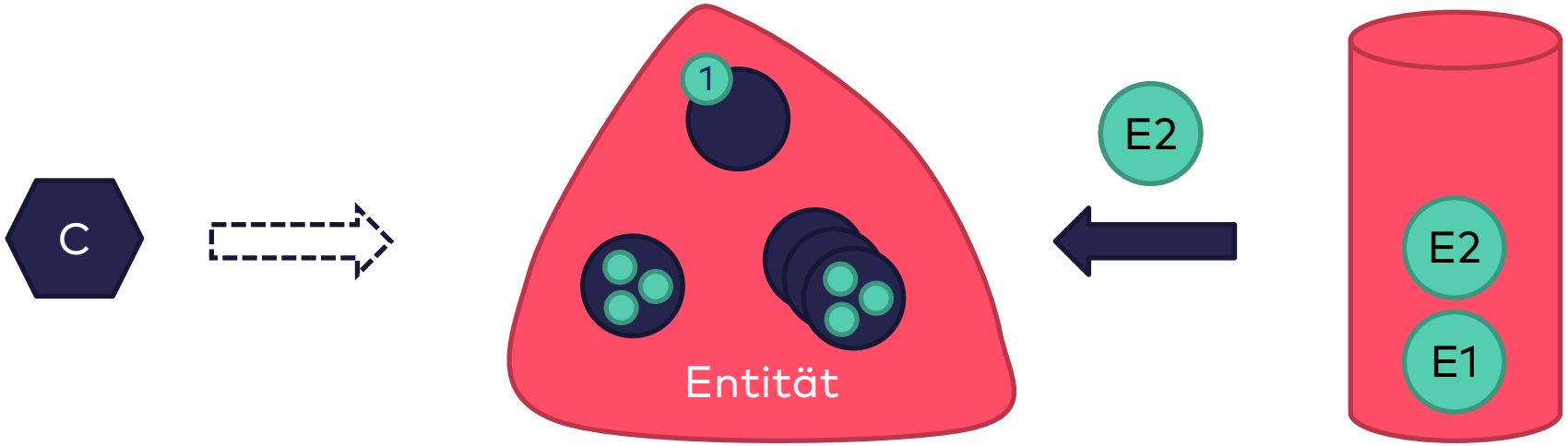
- Stellt Auftrag an Empfänger dar
- Hat eine Intention (mit Erwartung an Resultat)
- Kann fachlich abgelehnt werden
- Kann bei Ausführung fehlschlagen

Event Sourcing – Funktionsweise



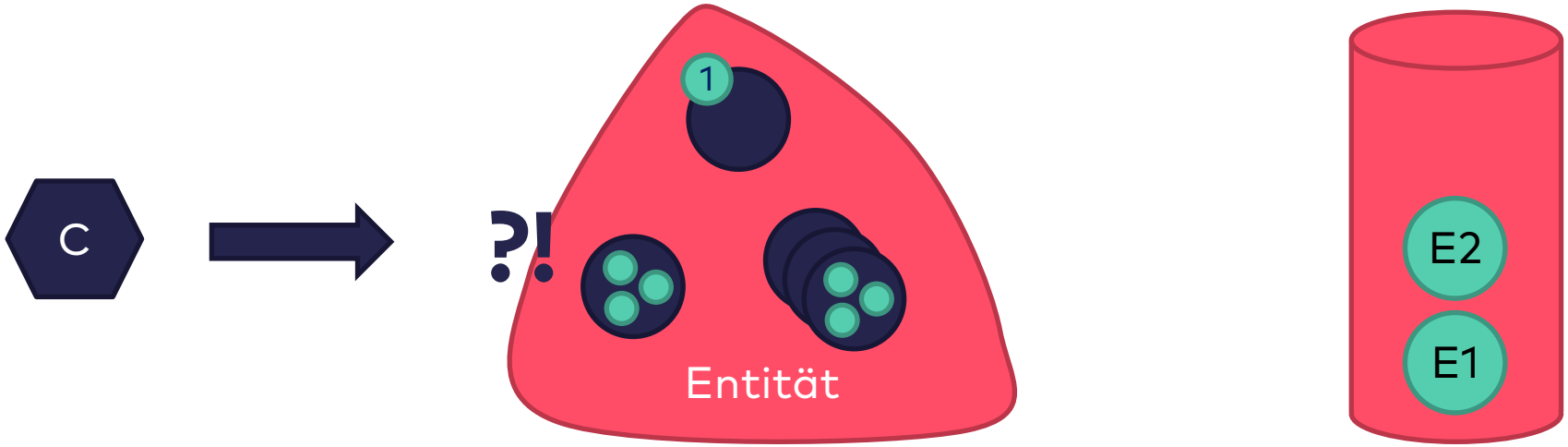
- **Geschäftslogik soll auf einer Entität aufgerufen werden (Command)**

Event Sourcing – Funktionsweise



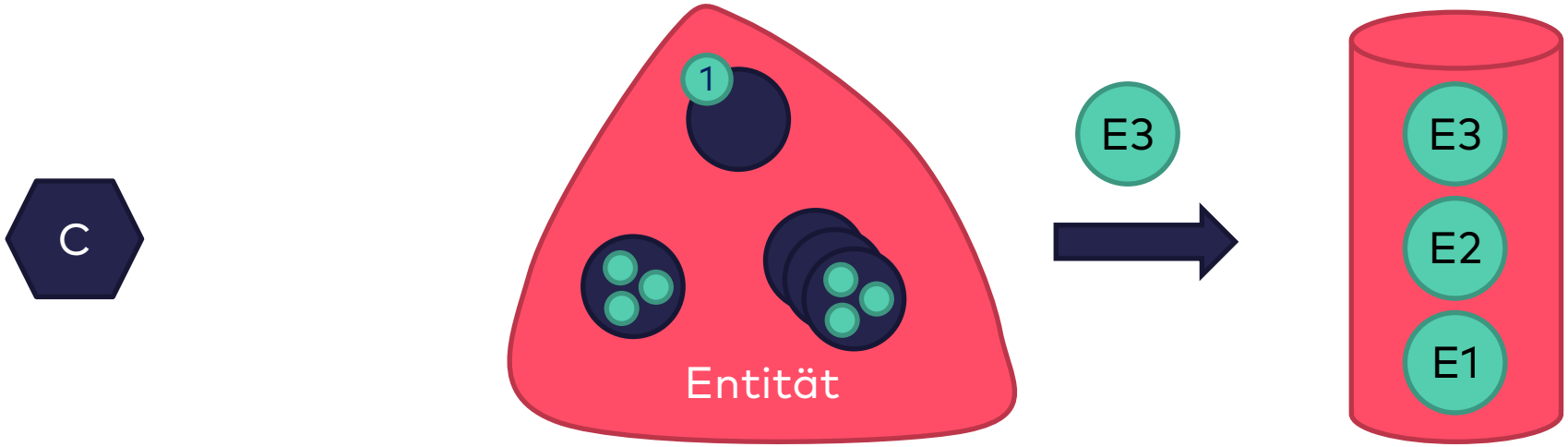
- Eine "leere" Instanz der Entität wird erzeugt
- Der Zustand der Entität wird durch Abspielen der aufgezeichneten Ereignissen wiederhergestellt ("rehydration")
- Es darf dabei keine Geschäftslogik ausgeführt werden

Event Sourcing – Funktionsweise



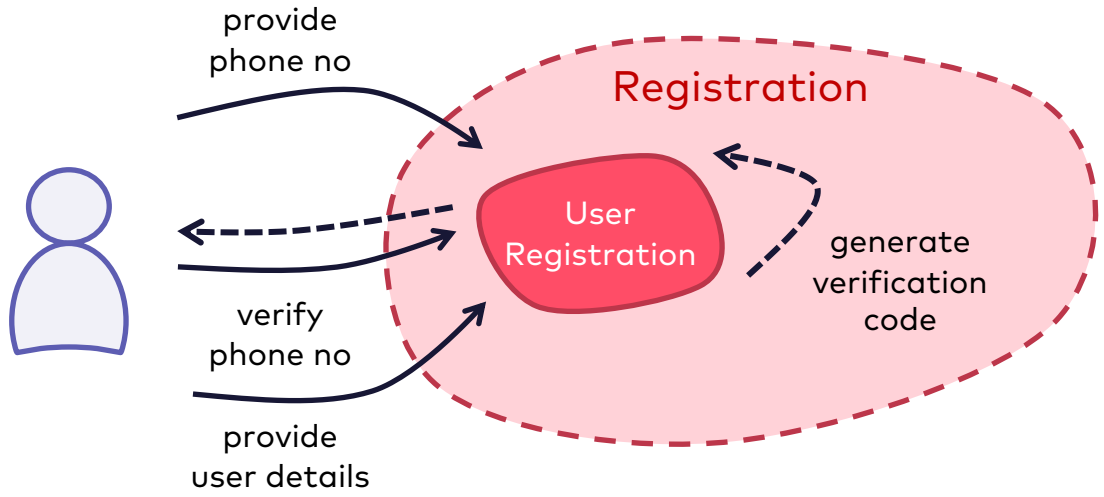
- Der Aufruf wird von Geschäftslogik gegen den Zustand der Entität validiert
- Der Aufruf wird abgelehnt, falls Invarianten der Entität verletzt würden
- Der aktuelle Zustand der Entität wird nicht verändert

Event Sourcing – Funktionsweise



- Ein neues Ereignis zur Darstellung der fachlichen Konsequenzen wird erzeugt
- Das Ereignis wird gespeichert und ist so beim nächsten Aufruf verfügbar
- Das Ereignis muss “nur” alle Semantik enthalten, um später den Zustand der Entität korrekt wiederherstellen zu können

Event Sourcing – Beispiel "User Registration"



Ereignisse

- UserRegistrationCompleted (–)
- UserDetailsProvided (FullName, Address)
- MobileNumberVerified (–)
- VerificationCodeGenerated (VerificationCode)
- MobileNumberProvided (MobileNumber)

Zustand

ID: 001	
MobileNumber	+41 79 123 45 67
VerificationCode	123456
FullName	Peter Meier
Address	8000 Zürich
Status	**COMPLETED**

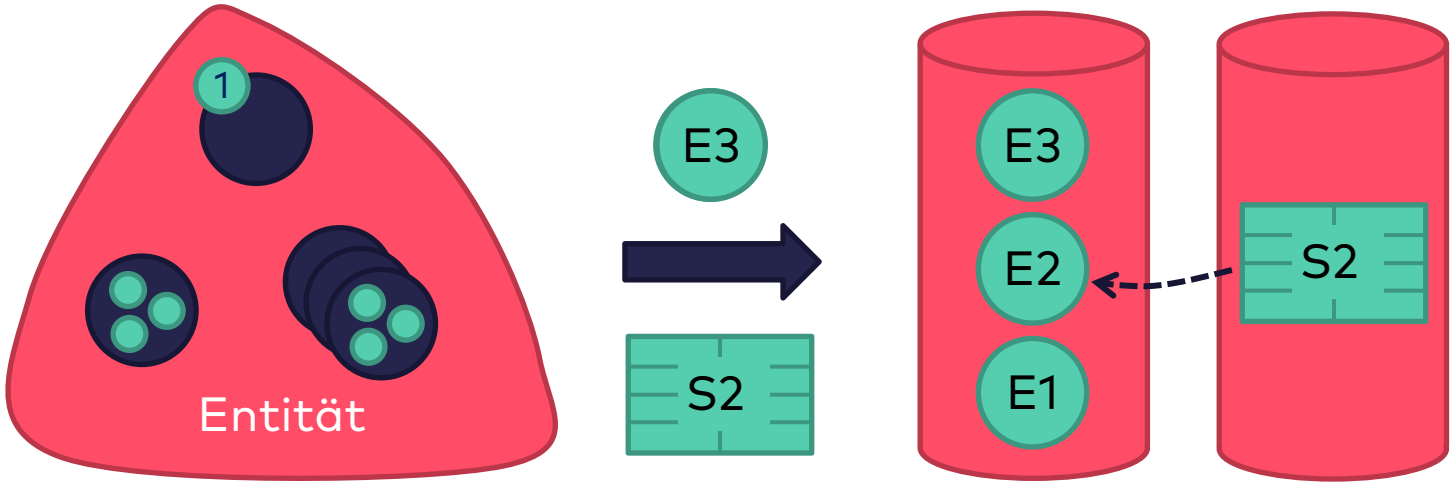
Event Sourcing – Vorteile

- **Erlaubt Herleitung, wie aktueller Zustand entstanden ist**
  - Daten werden nie unwiderruflich überschrieben, sondern nur “überblendet”
  - “Gratis Audit-Trail” (?)
- **Erlaubt Zugriff auf historischen Zustand**
  - Zeitsensitive Anwendungen, Referenzen auf historischen Zustand, Bi-Temporalität
- **Erlaubt Korrektur von fehlerhaftem Zustand**
  - Nachvollziehbare Kompensation von falschen Ereignissen
  - Retro-aktive Events

Event Sourcing – Herausforderungen

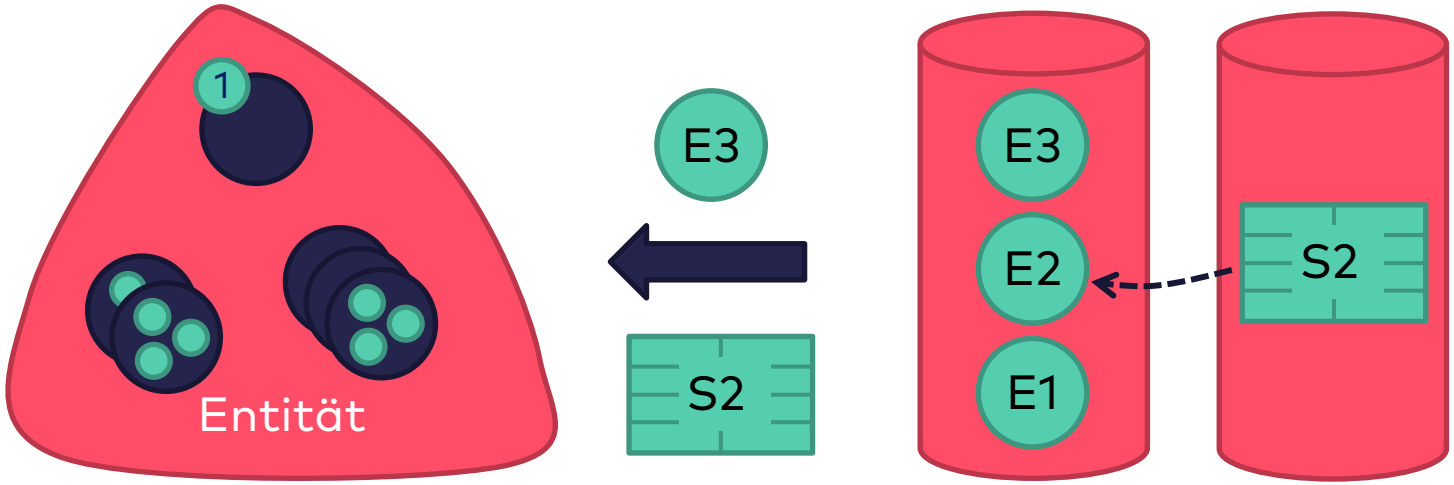
- **Bestehende Events müssen unbegrenzt aufbewahrt und verstanden werden**
  - Syntaktische und semantische “Legacy” für die Zukunft
  - Upcasting, Migration
- **Zustand kann nicht direkt effizient befragt werden**
  - Keine Queries auf bestimmte Eigenschaften von Entitäten möglich
- **Events können nicht einfach so gelöscht oder verändert werden**
  - DSGVO-Probleme ?!
- **Ungenügende Performance bei langer Historie mit vielen Events**
  - Rehydration kann lange dauern → Snapshots

Event Sourcing – Snapshots erstellen



- Nach Ablage jedes n-ten neuen Events wird zusätzlich aktueller Zustand der Entity als Snapshot persistiert
- Snapshot repräsentiert Zustand nach einem bestimmten Event
- Erstellung des Snapshots kann auch asynchron erfolgen

Event Sourcing – Snapshots verwenden



- Bei der Rehydration des Zustands wird zuerst der aktuellste Snaphot geladen
- Dann werden nur noch Ereignisse abgespielt, welche seither aufgetreten sind

Event Sourcing – Umsetzung

- **Persistenz von Events: es muss nicht Kafka sein**
  - Ablage als Dokument (kein ORM) in Datenbank
  - Möglichkeit zur Migration / Korrektur von Events
  - Skalierung über Sharding / Partitionierung auf Entitätsebene
- **Transaktionen**
  - Transaktionsklammer: alle Events aus einem Command
  - Events = einzige Wahrheit → keine verteilten Transaktionen
- **Optimistic Locking mit Sequenz-Nummer des Events**
  - “Optimistic Locking ++”: Konfliktlösung durch Analyse der fachlichen Bedeutung
  - automatischer Retry unter Verwendung der Invarianten der Entität

Event Sourcing vs. Domain Events

Event Sourcing Events

- Ziel: Wiederaufbau des internen Zustands → Strategie für Implementation der Persistenz
- Feingranular mit notwendigem Zustand für Interpretation in Entität
- Lokale Entscheidung für eine Entität

≠

Domain Events

- Ziel: Kommunikation eines fachlichen Ereignisses an “Externe”
- Grobgranular, mit sinnvollem Zustand für Interpretation durch “Externe”
- Architekturmuster für Integration zwischen Komponenten

→ Event Sourcing Events sind Interna einer Entität

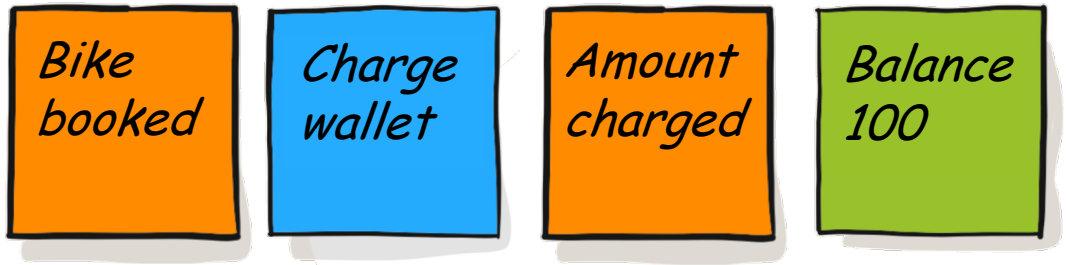
https://www.innoq.com/de/blog/domain-events-vs-event-sourcing/

Event-Sourced vs. Event-Driven

- **Event-Sourced**
  - Ansatz für Persistenz von Zustand
  - Aktion → Event → Zustand

- **Event-Driven**
  - Ansatz für Integration von Systemen
  - Event → Aktion → Zustand

- **Event-Driven + Event-Sourced**
  - Event → Aktion → Event → Zustand



CQRS - Command Query Responsibility Segregation

Schreibmodell ≠ Lesemodell(e)

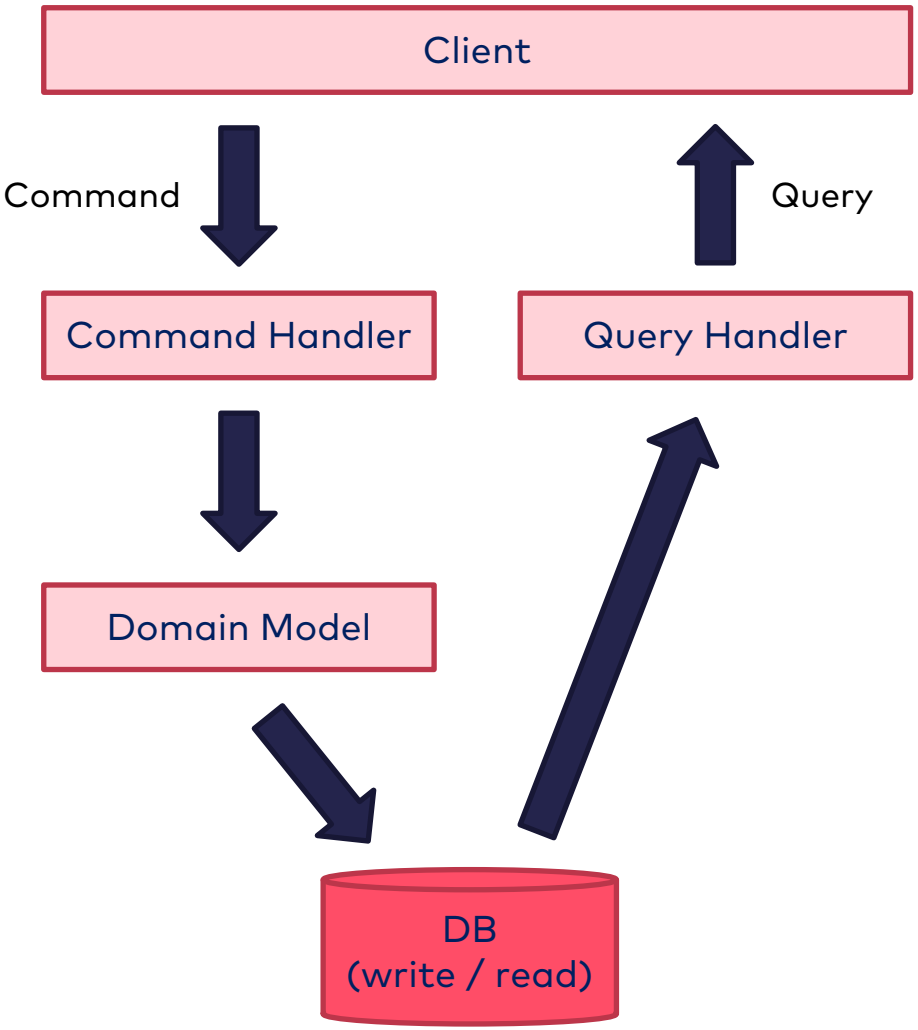
CQRS – Command Query Responsibility Segregation

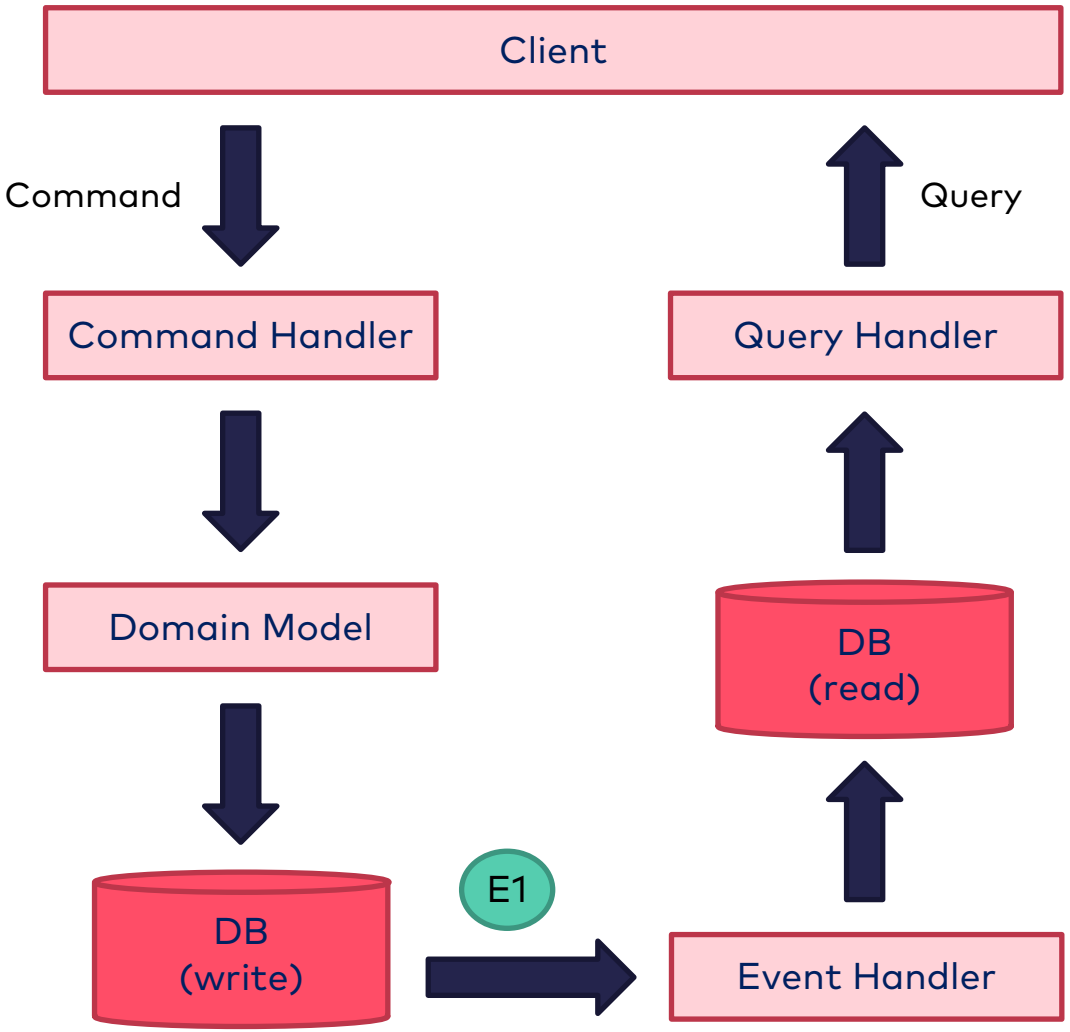
- **Separate Modelle für Schreiben ("Command") und Lesen ("Query")**
  - Separierung über Objekt-Modell, Datenbank, Prozesse, Hardware
  - Vermeidet Komplexität eines gemeinsamen Modells ("One Size Fits Never")

- **Erlaubt unterschiedliche Darstellungen einer Information**
  - Schreibmodell: fachliche Kohäsion, Kapselung von Wissen
  - Lesemodell: Integration von Daten, Aggregation von Werten, optimierter Zugriff

- **Erlaubt "Separation of Concerns" im Umgang mit Daten**
  - Schreibmodell: Invarianten, Konsistenz, Transaktionen, Locking
  - Lesemodell: Anfragen, Suchen, unterschiedliche Konsumenten (Sichten)

CQRS – Funktionsweise





CQRS - Motivation

- **Deutlich unterschiedliche Sichten beim Schreiben vs. Lesen**
  - 1 Sicht beim Schreiben, n unterschiedliche Sichten beim Lesen → Optimierung
  - Fachlich “saubere” Modellierung der Sichten → Reduktion der Komplexität, Separation of Concerns
- **Unabhängige Skalierung von Lesen und Schreiben**
  - Deutlich mehr Reads als Writes
  - n Replikate für Lesezugriff
- **Event Sourcing ;-)**
  - Effiziente Umsetzung von Anfragen auf Entitäten

CQRS + Event Sourcing

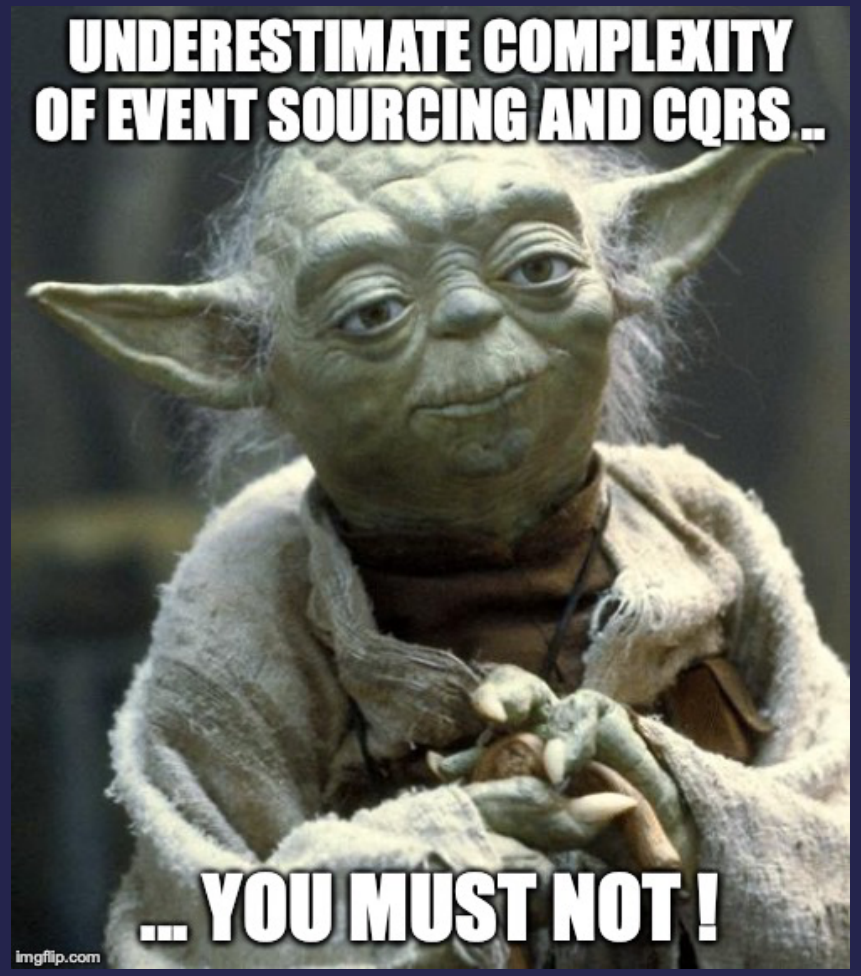
- **Schreibmodell verwendet Event Sourcing**
  - Schreibmodell speichert nur Events
- **Events aus Event Sourcing als Mittel zur Aktualisierung der Lesemodelle**
  - Events werden via Event Bus publiziert und an Lesemodell übertragen
- **Lesemodelle verwenden Events zum Aufbau einer Projektion**
  - Projektionen bilden aktuellen Zustand ab
  - Projektionen können später erneut erzeugt oder komplett neu erstellt werden

→ Engere Kopplung von Lesemodellen an Event Sourcing Events !!!

CQRS – Konsequenzen

- **Eventual Consistency bei asynchroner Aktualisierung des Lesemodells**
  - Lesemodell ist nicht streng konsistenz mit Schreibmodell
  - “Read Your Own Write” nicht mehr gegeben
- **Redundanz der Daten**
  - Einhaltung der “Data Ownership” ist kritisch
  - Kopplung zwischen Lesemodell und Schreibmodell (direkt oder indirekt)
- **Höhere Komplexität des Gesamtsystems**
  - Meist nicht lohnenswert für CRUD-Anwendungen
  - Mehr Nutzen bei Command- / Task-basierten Anwendungen

Fazit

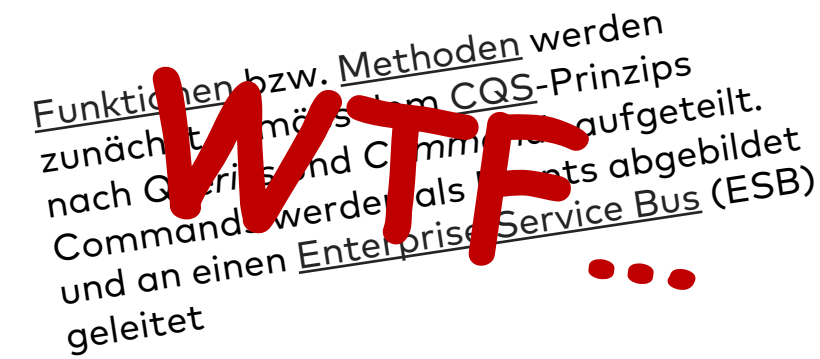


Quelle: https://imgflip.com/memegenerator

Links

- **Event Sourcing (Fowler)**
  https://martinfowler.com/eaaDev/EventSourcing.html

- **Event Sourcing (Wikipedia)**
  https://de.wikipedia.org/wiki/Event_Sourcing



- **Event Sourcing vs. Event-Driven Architecture**
  https://dev.to/olibutzki/why-event-sourcing-is-a-microservice-anti-pattern-3mcj

EVENT SOURCING UND CQRS

ZWEI KONZEPTE ERKLÄRT UND ENTWIRRT

Merci !!!

Christian Stettler
christian.stettler@innoq.com