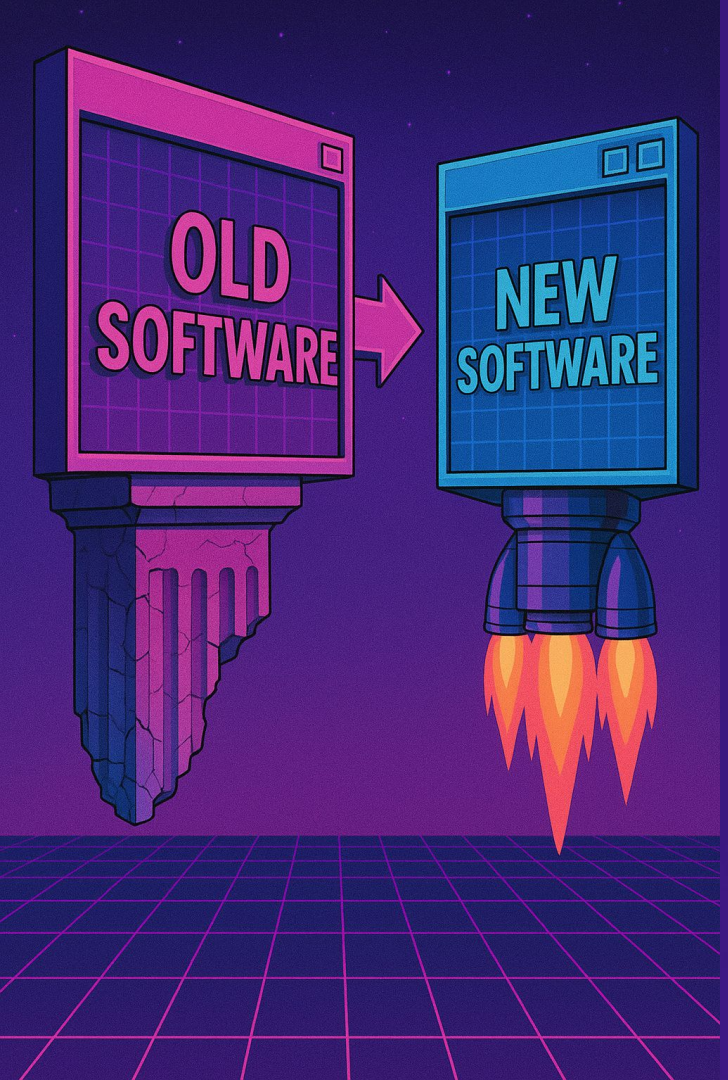


Jenseits von CRUD

Event Sourcing & CQRS als moderne Architektur-Antwort

Einführung und Motivation

Über uns

Stefan Schilling // stefan.schilling@digitalfrontiers.de

Consultants für Cloud native Software

Kersten Kriegbaum // kersten.kriegbaum@digitalfrontiers.de

digital frontiers

Firma

digital frontiers

Digital Frontiers GmbH & Co. KG

Berater mit sehr technischem Fokus

Wir produzieren Software nach Kundenanforderungen

Wir Beraten bei der Digitalisierung von Geschäftsprozessen

Disclaimer:

Hinweise zum Vortrag

Wir = Software produzierende Personen

Wir gehen nur am Rande auf CRUD ein

Methodik zum Identifizieren von Geschäftsereignissen wird nicht behandelt

Beispieldomäne: Bibliothek

Konzeptioneller Vortrag (umfangreich, dafür eher oberflächlich)

Wir sind jederzeit für Nachfragen kontaktierbar

Inhalt

Einführung und Motivation

Event Sourcing - Die Geschichte einer Anwendung verstehen

1. Von Anforderungen zu Events
2. Die Geschichte aufzeichnen
3. Fazit - The Good, the Bad

Pause (10 Minuten)

CQRS - Command Query Responsibility Segregation - Maßgeschneiderte Antworten geben

1. Motivation für CQRS - Warum noch ein Muster
2. Segregation - Das Kernkonzept
3. Architekturelle Implikationen

Zusammenfassung

Q&**A**

Einführung & Motivation

Um was geht es?

Oft wird Software nach **CRUD Prinzip** geschrieben

Heisst: Wir speichern den **aktuellen Zustand** einer Anwendung

Meistens in Relationalen Datenbanken oder NoSQL Datenbanken

Diese bilden dann “**single source of truth**”

Fun Fact: Das D in CRUD wird nur sehr selten genutzt (eher “deleted” Flag)

Einführung & Motivation

Um was geht es?

Software gut nach **Kundenanforderungen** schreiben

Relevant auf vielen Ebenen, von Anforderungserfassung bis Betrieb

Bild vermitteln, wie man von Anfang bis Ende gut vorgeht

Flexibel bleiben!

Technisch eher oberflächlicher, **konzeptioneller Vortrag**

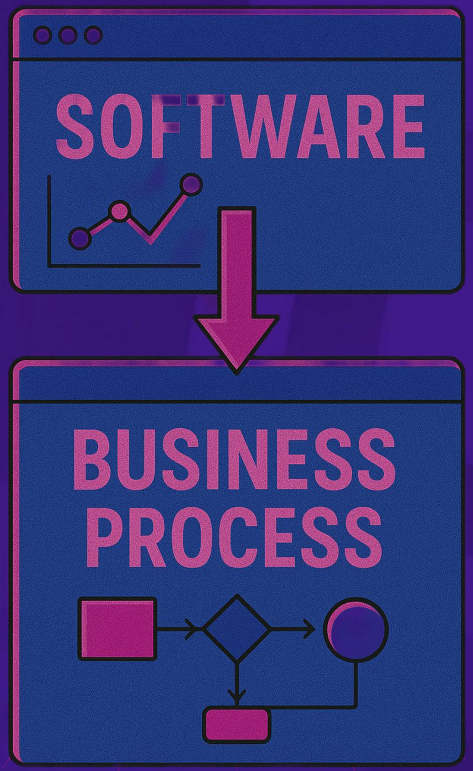
Im Fokus: Anforderungen in Software Umsetzen

Moderne Art **mehrdimensionale Software** zu schreiben

Anforderungen verändern sich während der Lebenszeit der Software (neue hinzu, alte weg)

Software = **Geschäftsprozesse**

Domäne muss verstanden werden



Einführung & Motivation

Beispiel

Kundenwunsch:

“Ich brauche eine Bibliotheks-App”

Was ist gemeint?

Bücher verwalten?

Bücher digital leihen?

Bibliothek simulieren?

Event Sourcing - Die Geschichte einer Anwendung verstehen

1. Von Anforderungen zu Events

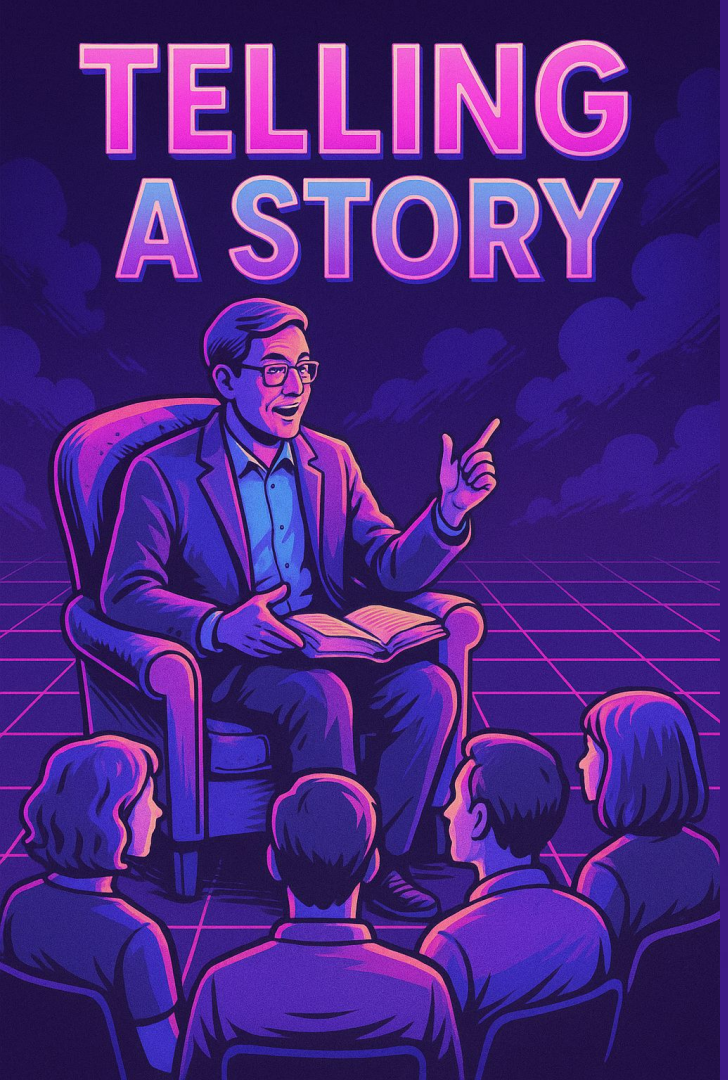
1. Von Anforderungen zu Events

Wir als Softwareproduzenten müssen
Geschäftsprozesse verstehen

- Das tut man am Anfang nicht

Wo fängt man an?

- Identifikation, Abgrenzung und Definition
- Von fachlichen **Aktionen** & **Ereignisse** im Betrieb



1. Von Anforderungen zu Events

Wir wollen eine **gemeinsame Geschichte** entwickeln!

Alle sollen die **gleiche Geschichte** erzählen können.

Kollaboration ist zentral.

1. Von Anforderungen zu Events

Wir müssen also Geschäftsereignisse identifizieren.

Beispiel:

Status: “Buch verliehen” // “Buch vorhanden”

Geschäftsereignisse:

“Buch wurde katalogisiert”

“Buch wurde zurückgegeben”

“Buch wurde beschädigt”

“Buch wurde ausgehändigt”

“Ausleihe wurde stattgegeben”

“Ausleihe angefragt”

1. Von Anforderungen zu Events

Events sind **Fakten im System**

Events sind **unveränderliche** Wahrheit

Events beschreiben die **Geschichte der Anwendung** aus Business-Sicht

1. Von Anforderungen zu Events

Beispiel Event als JSON

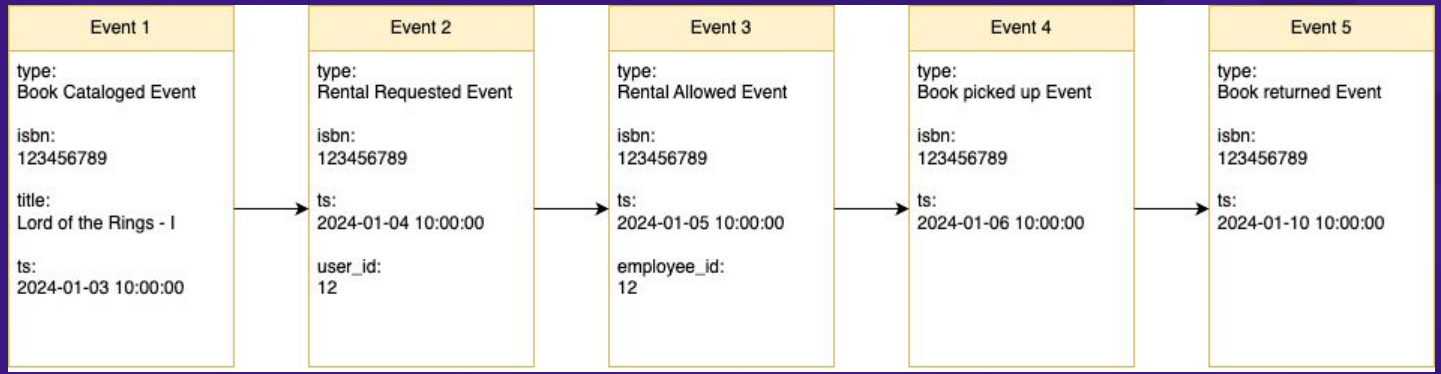
```
{
  "type": "event",
  "payload": {
    "source": "https://library.eventsourcingdb.io",
    "subject": "/books/978-3-608-93981-1",
    "type": "io.eventsourcingdb.library.book-catalogued",
    "specversion": "1.0",
    "id": "0",
    "time": "2025-06-09T15:20:04.566719261Z",
    "datacontenttype": "application/json",
    "predecessorhash": "0000000000000000000000000000000000000000000000000000000000000000",
    "data": {
      "author": "J.R.R. Tolkien",
      "isbn": "978-0756906788",
      "title": "Herr der Ringe 1"
    },
    "hash": "6fad7994961d8a181c0e7bb2075e4186abc0f430e29d96fd51e23635895f81c9"
  }
}
```

2. Event Sourcing - Die Geschichte aufzeichnen

2. Event Sourcing - Die Geschichte aufzeichnen

Grundidee

- Wir speichern alle Veränderungen im System …
- … statt nur dem aktuellen Status



2. Event Sourcing - Die Geschichte aufzeichnen

Kein Ereignis kann je gelöscht werden

Es kann nur ein entgegengesetztes Ereignis stattfinden

Beispiel:

Bei Buchrückgabe wird nicht “Buch ausgeliehen” gelöscht

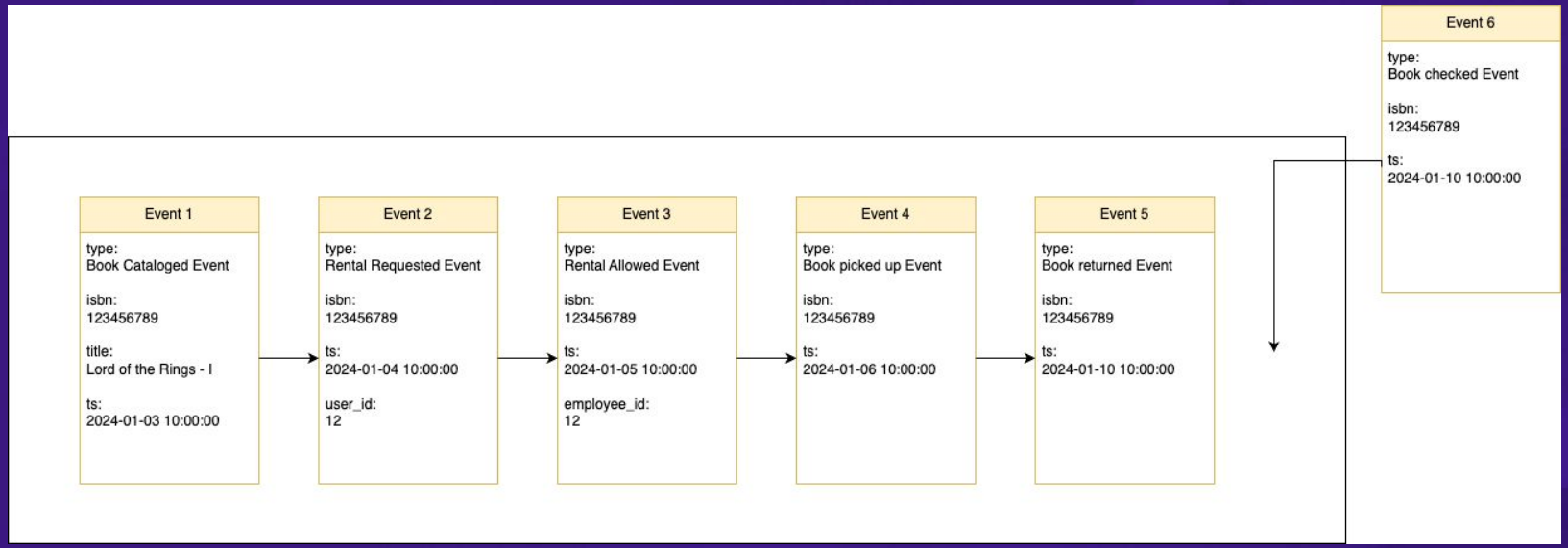
Es wird “Buch zurückgegeben” geschrieben.

Also:

Append only!

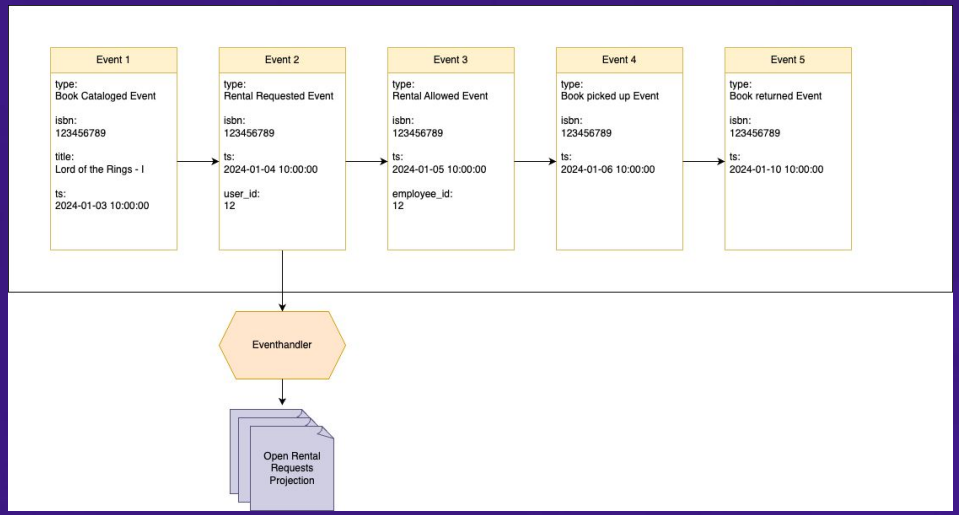
2. Event Sourcing - Die Geschichte aufzeichnen

Append only



2. Event Sourcing - Die Geschichte aufzeichnen

- Aktuellen Zustand bekommt man Inklusive (Ergebnis aller Ereignisse)
- **Zustand bei Bedarf** jederzeit konstruierbar
- (hier Beispielhaft Zustand nach zweitem Event)



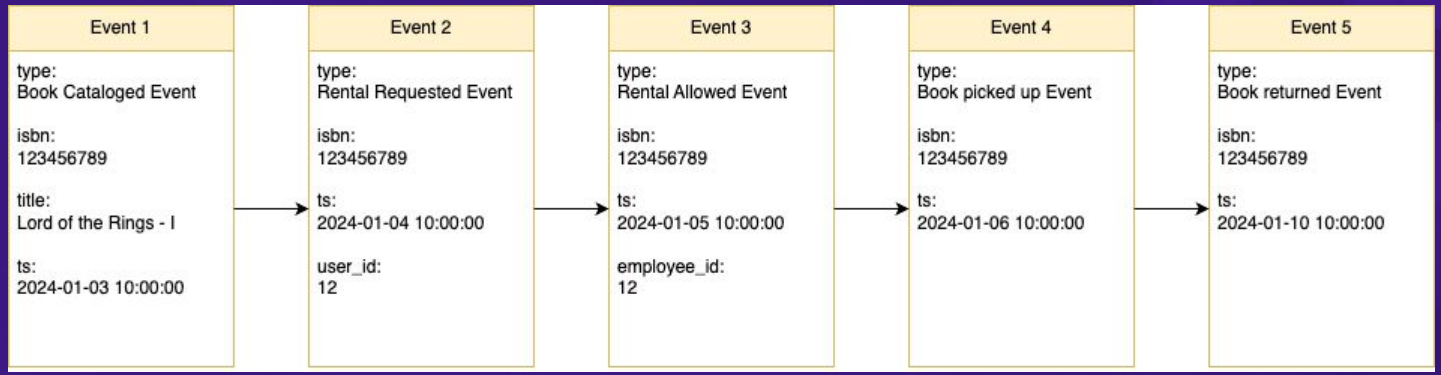
2. Event Sourcing - Die Geschichte aufzeichnen

Bedeutet auch:

Jeder alte Zustand kann wiederhergestellt werden!

Zeitreisen im System werden möglich

Nachvollziehbar, was geschehen ist



2. Event Sourcing - Die Geschichte aufzeichnen

Kernkonzepte zusammengefasst:

- Wir speichern die **Sequenz** aller Geschäftsereignisse
- Diese Sequenz der Ereignisse ist die **single source of truth**
- Speicher ist append only

2. Event Sourcing - Die Geschichte aufzeichnen

Speicher ist heute **sehr billig**

Events sind in der Regel **sehr klein** (einige Byte bis wenige kb)

Wachsender Event Store ist (technisch) **kein Problem**

… Aber **nicht technisch schon** (wir werden darauf zurückkommen)

2. Event Sourcing - Die Geschichte aufzeichnen

Beispiel:

Eventgröße:
- 2.0 KB (~2000 Zeichen)

Events pro Tag:
- 100.000 Events am Tag

Zeitraum:
- 10 Jahre

Speicherplatz:

- 930 GB

1 TB Google Cloud Standard Storage: **~24$ pro Monat**

Quelle: https://www.thenativeweb.io/products/eventsourcingdb

3. Event Sourcing - Fazit - The Good, the Bad

4. Event Sourcing - Fazit - The Good, the Bad

Vorteile:

- Wir **verstehen** unsere **Anwender**
- Daher können wir gut **über Änderungen sprechen**

4. Event Sourcing - Fazit - The Good, the Bad

Vorteile:

- Vollständige **Historie**
- **Audit Trails** sehr einfach möglich (mit Hashes absicherbar)
- Ideal für **Debugging** (Was ist wieso passiert)
- Direkt erkennbar, welcher Verlauf zu Problematischem Verhalten geführt hat

4. Event Sourcing - Fazit - The Good, the Bad

Vorteile:

- Analysen zu belieben Zeitpunkten oder Zeiträumen möglich
- Es gehen keine Informationen verloren

Beispiel:

- “Wo war das Buch vor 14 Tagen?”

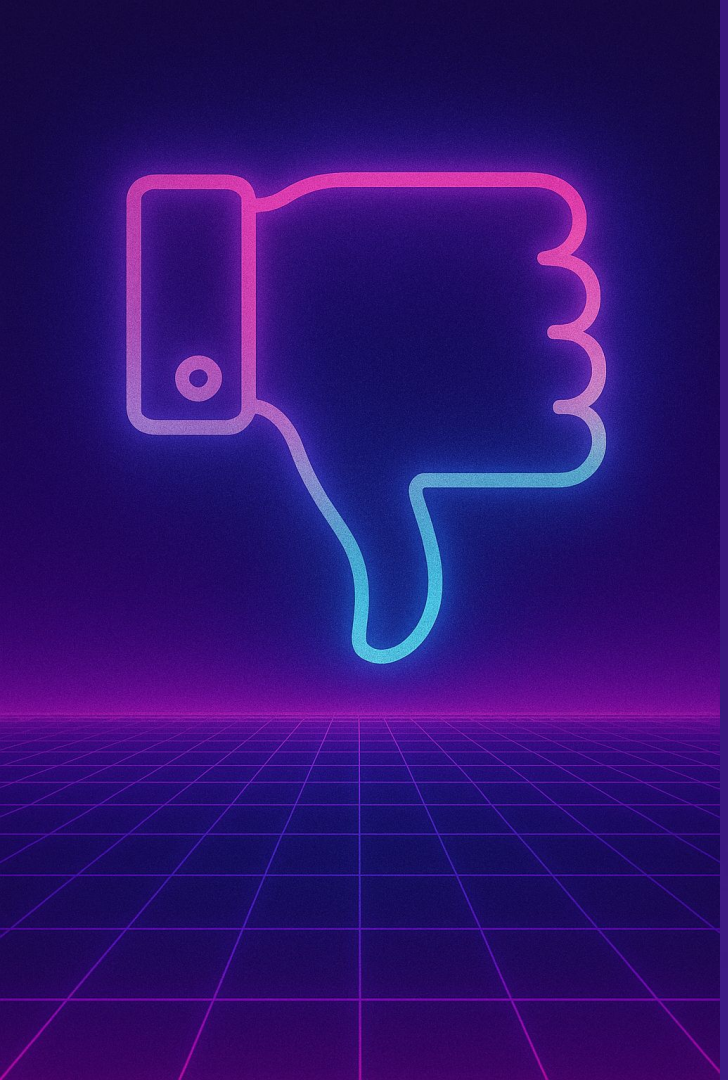
4. Event Sourcing - Fazit - The Good, the Bad

Vorteile:

- Flexibles System
- Wir können zu späterem Zeitpunkt beliebige Analysen/Sichten erzeugen …
- … von denen wir bei Beginn nicht wussten, dass sie gewollt werden

Beispiel:

- “Wie viele Leihanfragen hätte ich nicht ablehnen müssen, wenn ich statt 10 20 Exemplare Herr der Ringe Band 1 bestellt hätte”



4. Event Sourcing - Fazit - The Good, the Bad

Nachteile:

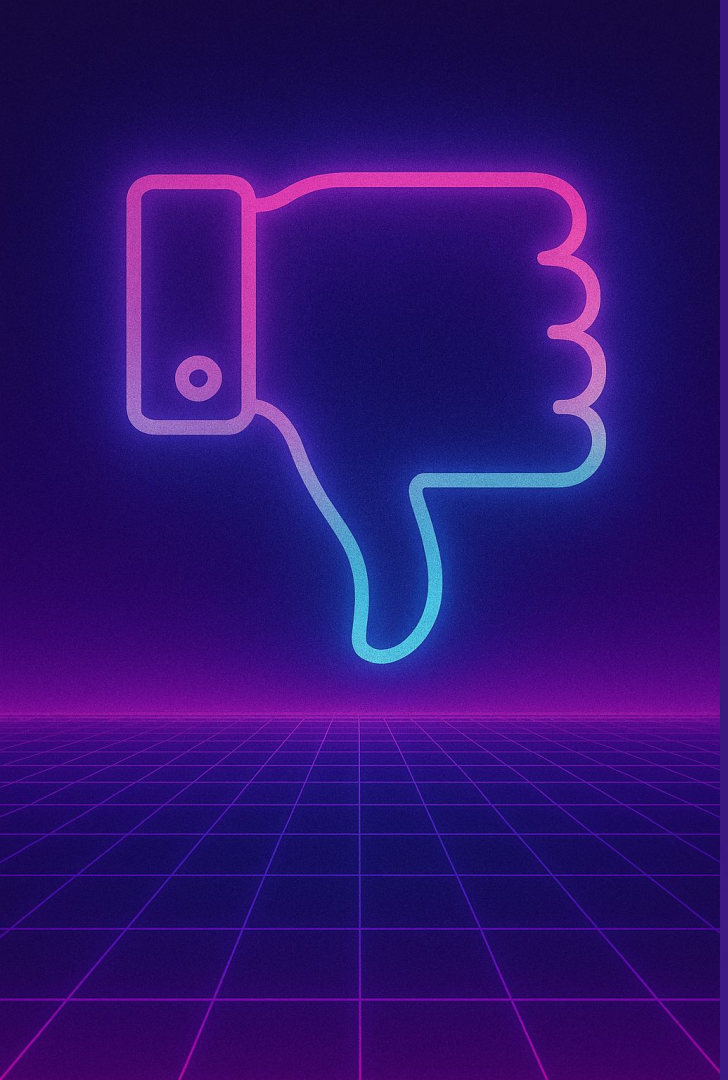
- Den Zustand zu berechnen kann sehr teuer werden bei vielen Events
- Es erfordert ein Umdenken, was herausfordernd sein kann …
- … gerade wenn man schon lange CRUD Software schreibt

Pause

CQRS - Command Query Responsibility Segregation

Maßgeschneiderte Antworten geben

1. CQRS - Motivation für CQRS - Warum noch ein Muster



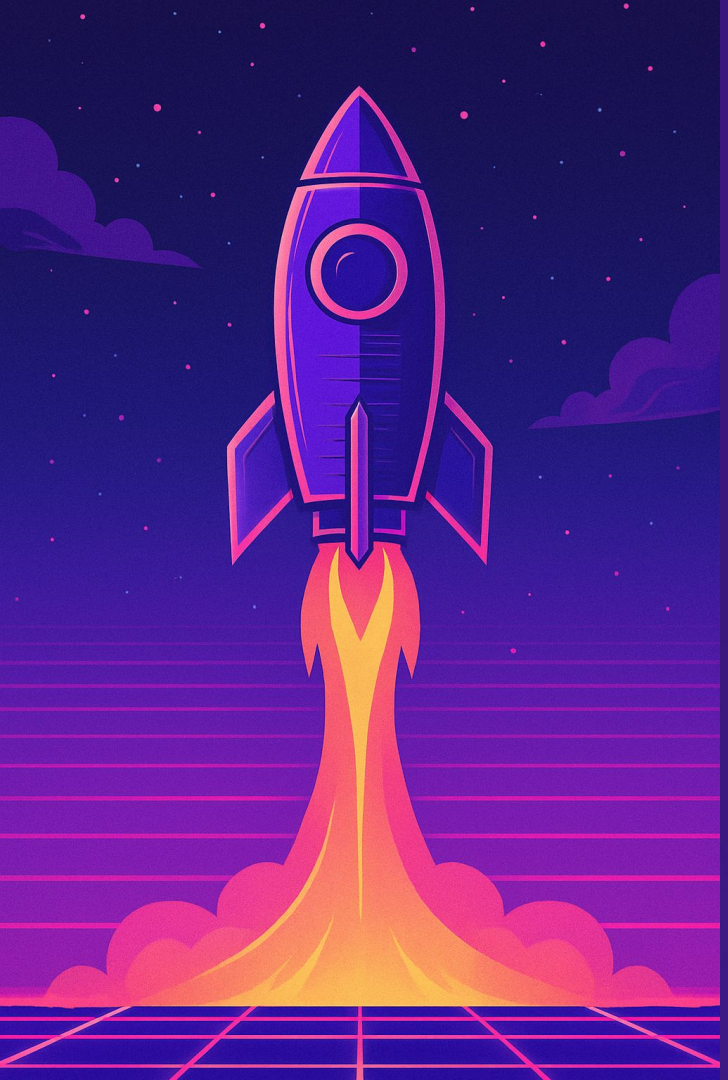
1. Motivation für CQRS

Rückblick Nachteile ES:

- Antworten geben kann **teuer werden**
- Meistens will man aber nur **schnell eine** bestimmte **Sache wissen**

Beispiel:

- Welche Anfragen zur Ausleihe sind gerade offen?



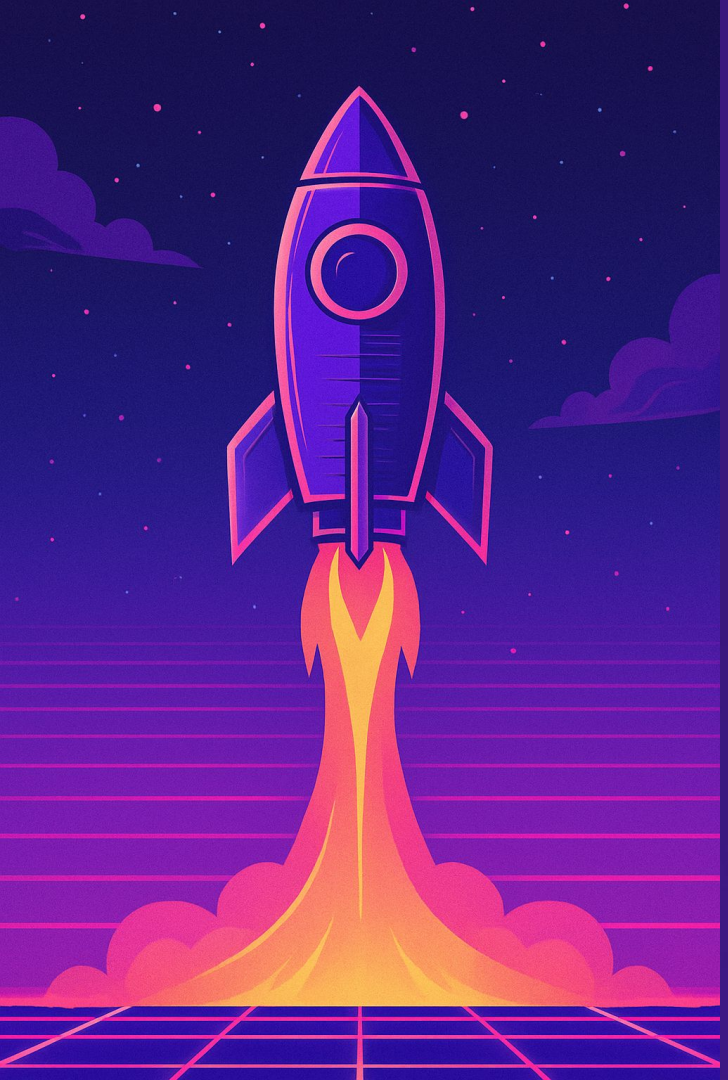
1. Motivation für CQRS

Lesen VS Schreiben:

- Wir wissen jetzt, wie wir unsere D**aten Schreiben** (ES)

Schreiben

- Meistens kritisch (Rechte)
- Konsistenz
- Validierung (Business-Regeln)

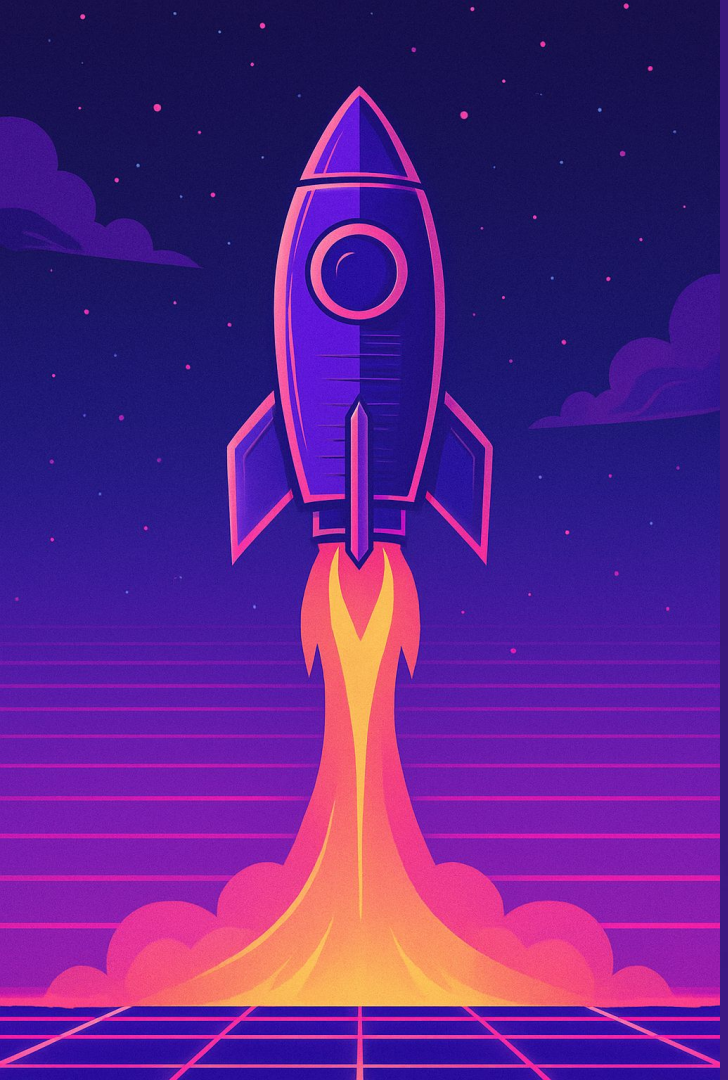


1. Motivation für CQRS

Lesen VS Schreiben:

Lesen

- Schnell/Performance
- Meistens wird **viel öfter gelesen** als geschrieben
- Im Frontend filtern ist zu spät (bei vielen Daten/mobile) und unsicher



1. Motivation für CQRS

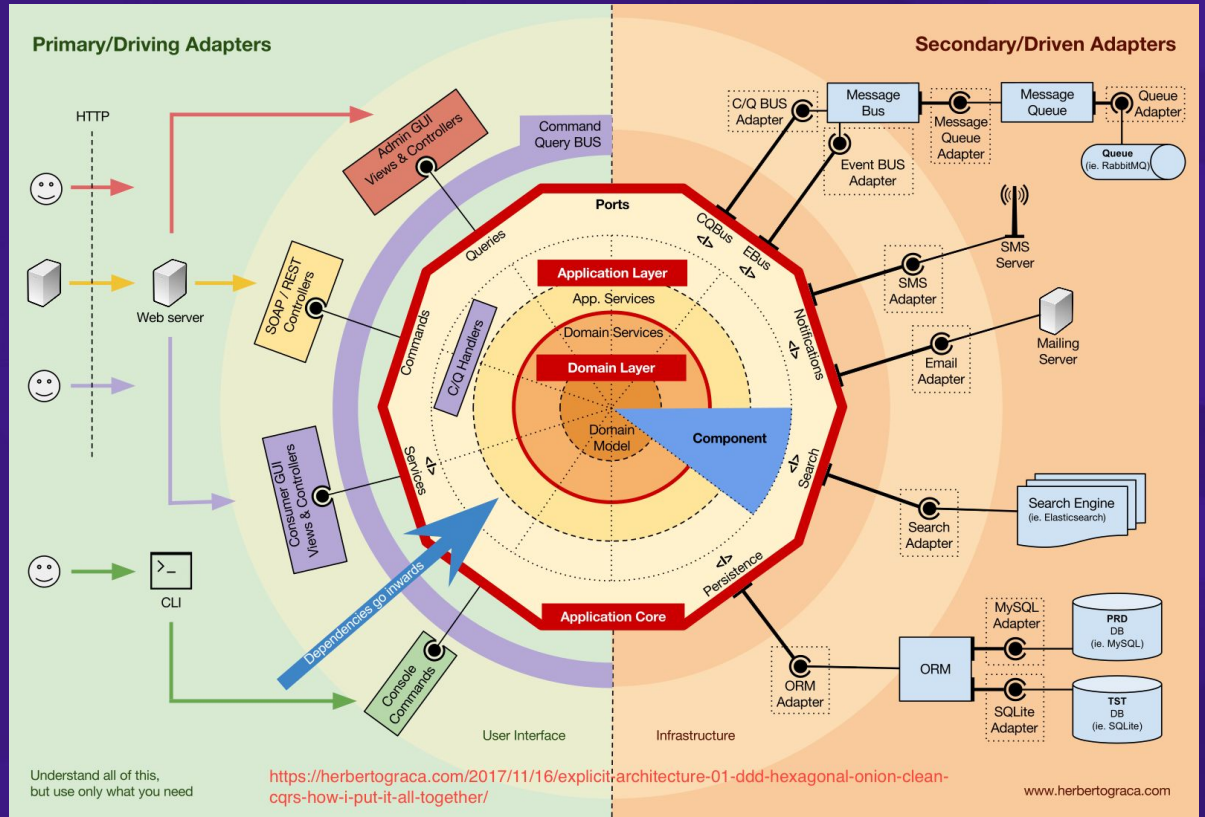
Lesen VS Schreiben:

- **Lesen** hat also oft ganz **andere Anforderungen** als Schreiben
- Siehe: Normalisierung Relationale Datenbanken. In der Praxis oft 3, wieso?
- Kompromiss aus gut schreibbar und gut lesbar (Hohe NF = gut schreiben, schlecht lesen da viele Joins)

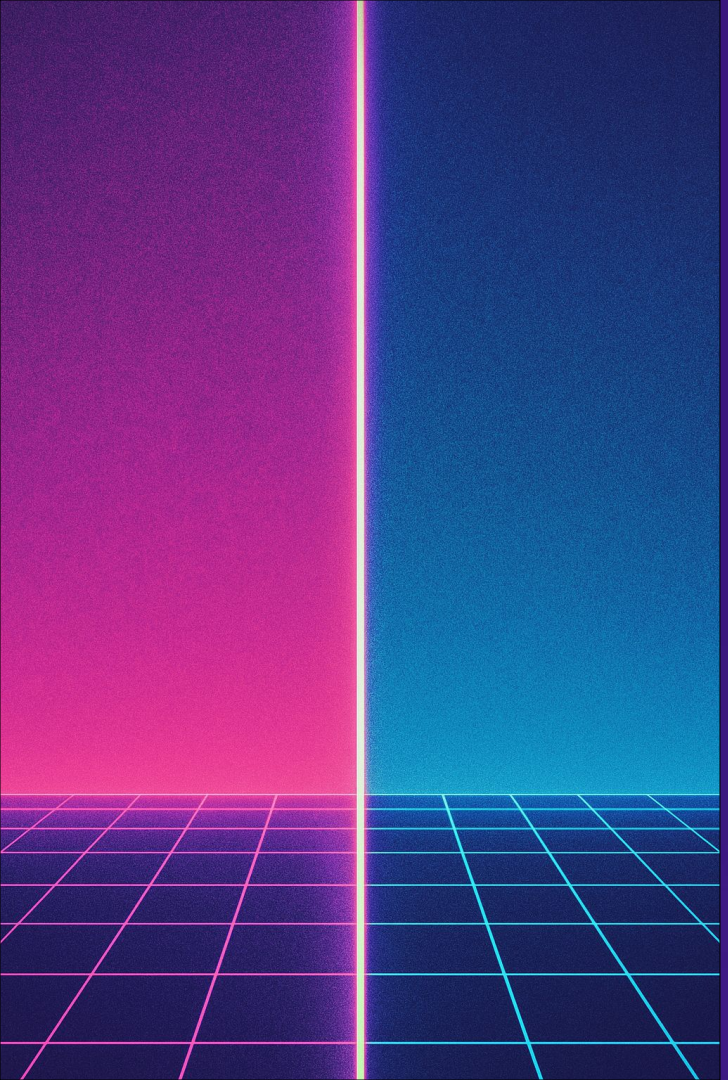
2. CQRS - Segregation - Das Kernkonzept

2. CQRS - Segregation - Das Kernkonzept

Bitte nicht erschrecken!



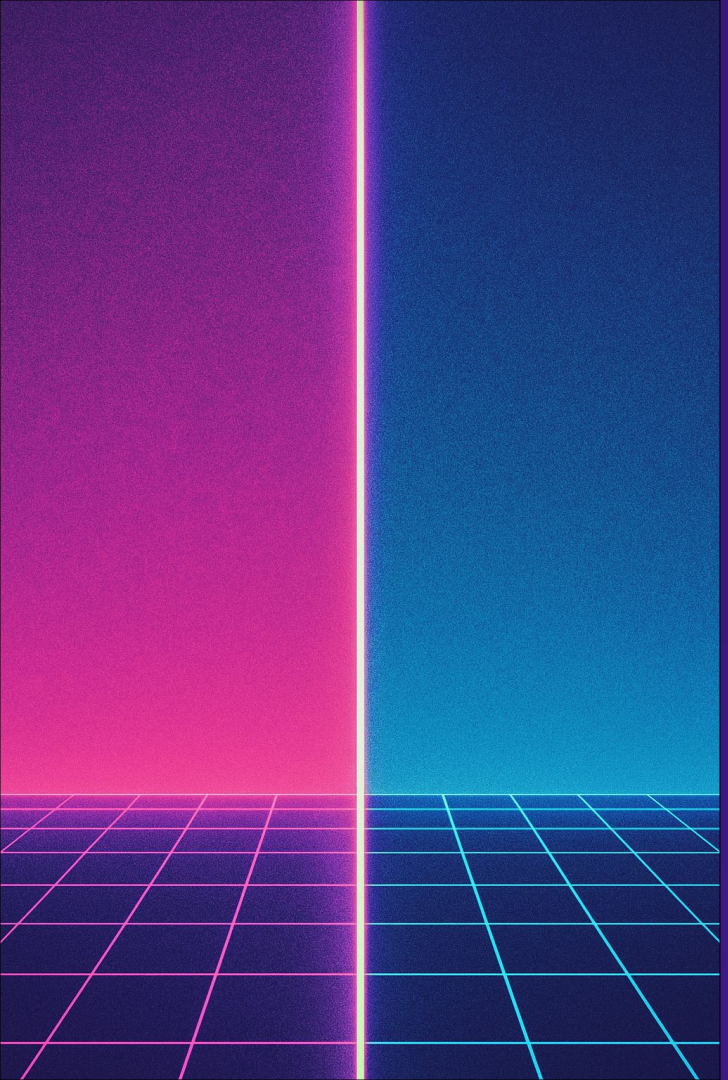
digital frontiers



2. CQRS - Segregation - Das Kernkonzept

Welche Funktionen von CRUD brauchen wir mit einem Eventstore noch?

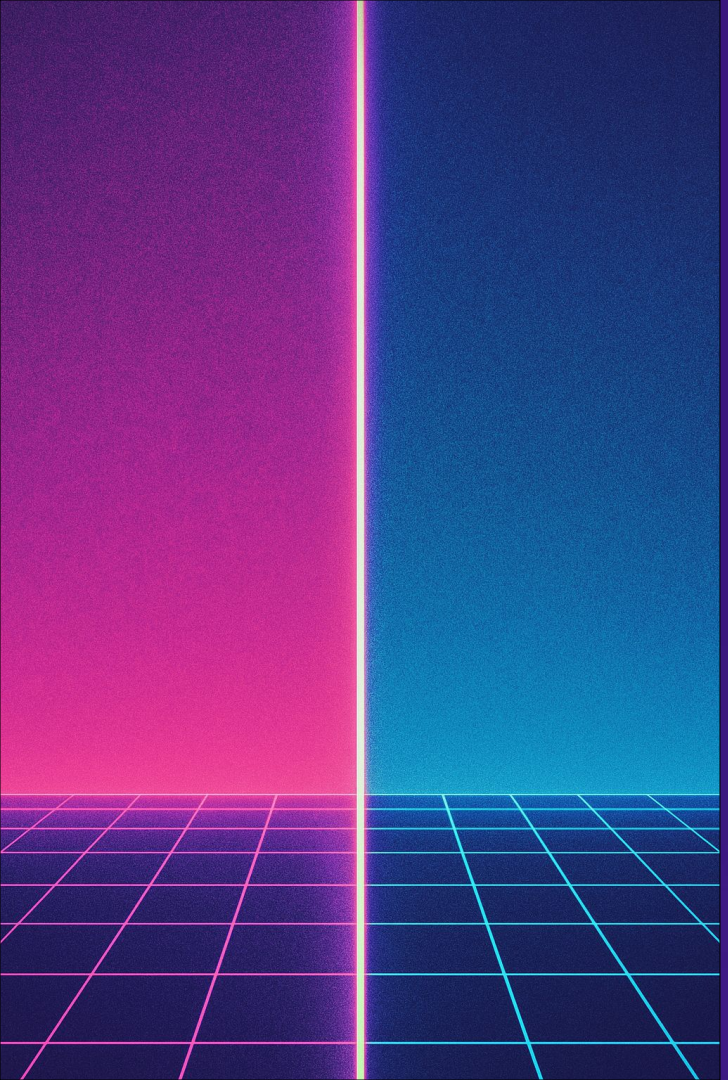
- CRUD: Create, Read, Update, Delete
- Erinnerung: Events sind **immutable** und Store ist **append only**
- Also: **Create und Read!**



2. CQRS - Segregation - Das Kernkonzept

Grundidee:

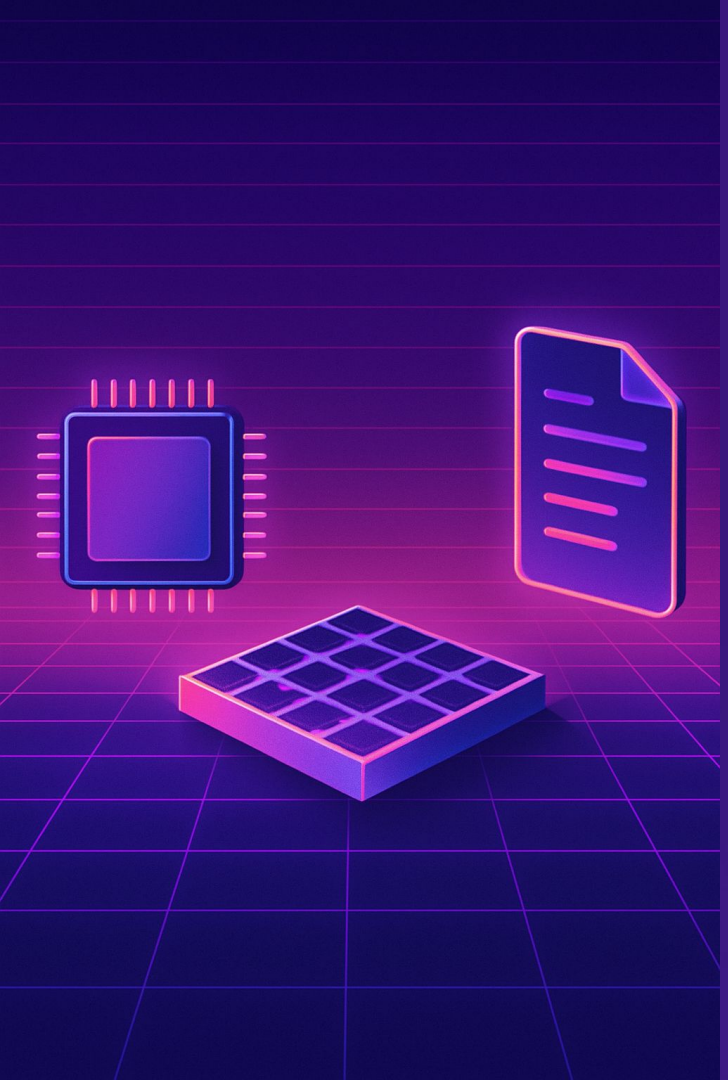
- **Trennen von Verantwortung**
- **Trennen von Lese- und Schreibvorgängen** (Schreibmodell muss nicht Lesemodell sein)



2. CQRS - Segregation - Das Kernkonzept

Außerdem:

- **Trennen von Lesemodellen**
- Oft **verschiedene Modelle** (Suchindex, Relational, NoSQL)
- **Maßgeschneiderte Datenstrukturen** für verschiedene Ansprüche
- Viele “Datenschnipsel” für **spezifische Anfragen**



2. CQRS - Segregation - Das Kernkonzept

Was sind Lesemodelle?

- **Speichern verschiedene Status**
- **Sollen Antworten auf bestimmte Fragen geben können**
- Meistens optimiert für **schnelles Lesen**
- Werden oft **durch die Events** aus dem ES **aktualisiert** (quasi als "Reaktion" auf das, was passiert ist)

Beispiel:

- Liste aller offenen Leihanfragen

2. CQRS - Segregation - Das Kernkonzept

Trennung in Komponenten (vereinfachte Darstellung):

Queries: Frage nach Information

Queryhandler: Validierung

Beispiel:
- Zeige mir alle offenen Leihanfragen

Commands: Absicht, etwas zu ändern

Commandhandler: Validierung

Beispiel:
- Ich möchte Herr der Ringe Band 1 ausleihen

2. CQRS - Segregation - Das Kernkonzept

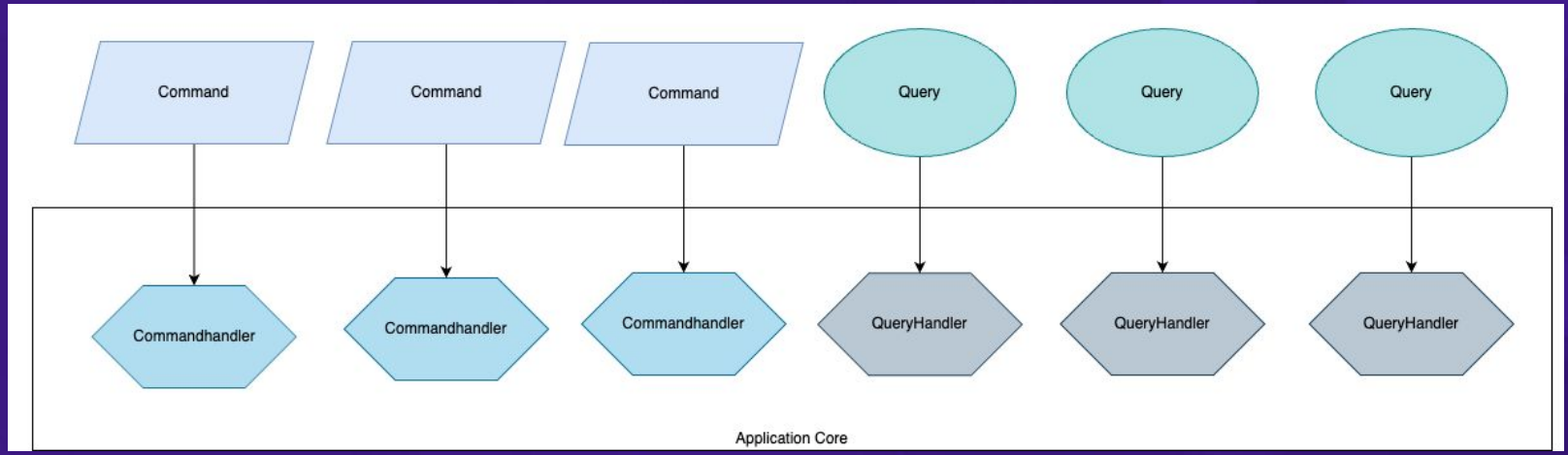
Komponenten (vereinfachte Darstellung):

Queries: Frage nach Information

Queryhandler: Validierung

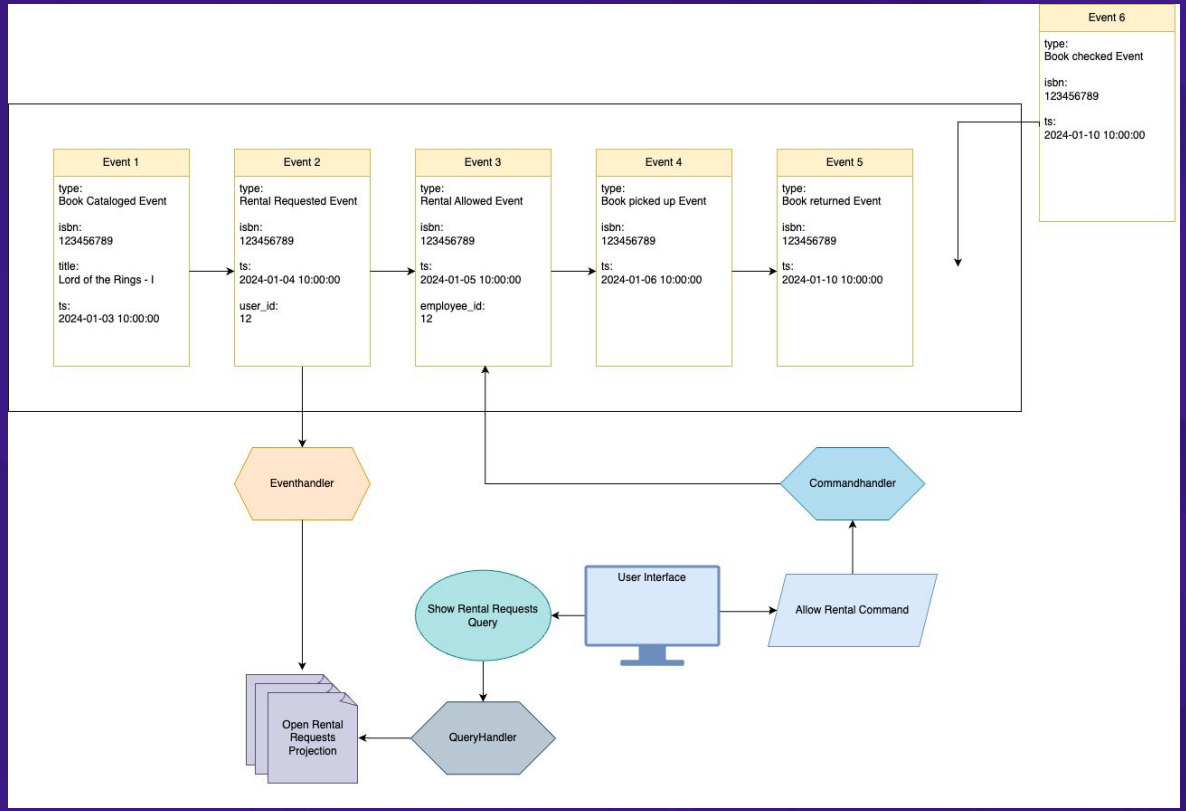
Commands: Absicht, etwas zu ändern

Commandhandler: Validierung



2. CQRS - Segregation - Das Kernkonzept

In Kombination mit einem EventStore



digital frontiers

3. CQRS - Architekturelle Implikationen

3. Architekturelle Implikationen

Zusammenspiel ES & CQRS:

Müssen nicht zusammen verwendet werden

Auch **alleine daseinsberechtigt**

3. Architekturelle Implikationen

Zusammenspiel ES & CQRS:

Aber sind **sehr gute Partner**!

ES liefert Events auf der Schreibseite

Events sind **idealer Input** zum **Aufbauen** Verschiedener **Readmodels**

Readmodels werden oft **asynchron geupdated** (Eventual consistency)

3. Architekturelle Implikationen

Technologiefreiheit bei unterschiedlichen Lesemodellen

Optimierte Datenbank für Events (Write-Optimized Store)

Andere, **passende** Datenbanken für **verschiedene Read Models**

Zum Beispiel:

- relationale DB für Listen
- Suchindex für Suche
- Graph-DB für Beziehungen

3. Architekturelle Implikationen

Mobile & UI - Erweiterbarkeit

Android Datenmodell

iOS Datenmodell

Maßgeschneiderte Sichten:
Erstelle **genau das Read Model,**
das die **Mobile** App braucht

schlank, schnell, passend für kleine Bildschirme

3. Architekturelle Implikationen

Security

Commands als klar definierter Ort für **Berechtigungsprüfungen** (Rückgriff: Commands sind Intention! > Berechtigungsprüfung)

Übersetzungsschicht als Möglichkeit klar zu definieren, **was wer darf** (Fachlichkeit)

3. Architekturelle Implikationen

Security

Unterschiedliche Projektionen für verschiedene Nutzerrollen

Beispiel:

- Der Admin alles
- Der Kunde nur seine Daten
- Der Analyst nur anonymisierte Daten

3. Architekturelle Implikationen

Security

- **Audit Trail** (Events) Bleibt als **Sicherheitsnetz** (wer hat was getan?)
- Sehr gut bei **Debugging**
- Sehr gut bei **Schadensanalyse**
  - Security Breaches nachvollziehbar, es wird ja nichts vergessen

3. Architekturelle Implikationen

```
@Test
fun `sollte ein Buch reservieren, wenn die Ausleihe genehmigt wird`() {
    fixture.given(
        BookCataloguedEvent(bookId),
        RentalRequestedEvent(bookId, renter)
    )
        .`when`(
            RentalAllowedCommand(bookId)
        )
        .expectEvents(
            BookReservedEvent(bookId, renter)
        )
}
```

Testing

- Klare Trennung erleichtert Tests:
- Command Handler
  (Command > Output: Events?)
  given().when().then() Syntax
- Query Handler
  (Query > Output: Read Model Data?)

Zusammengefasst: Führt Aktion X zu erwartetem Verhalten Y?

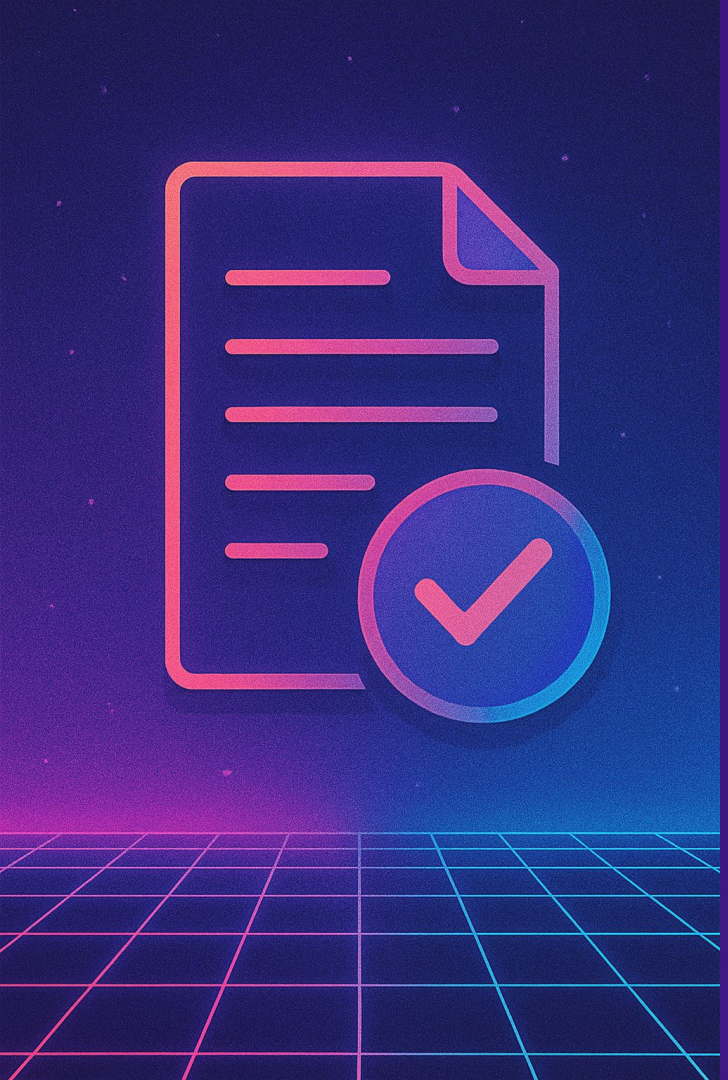
3. Architekturelle Implikationen

Skalierbarkeit

- Lese- und Schreib-Pfade können **unabhängig skaliert werden**

Beispiel:
Mehr Server für die Produktsuche als
für den Bestellprozess

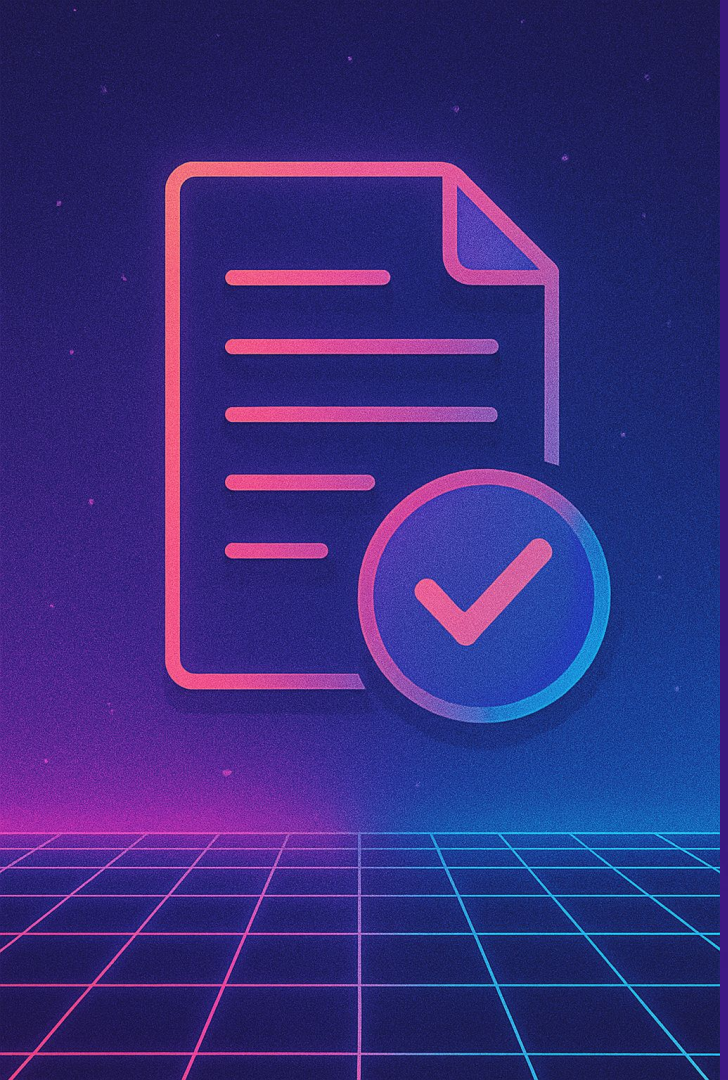
Zusammenfassung



Zusammenfassung

Event Sourcing:

- Notwendigkeit, die Sprache des Anwenders und die Geschäftsereignisse zu verstehen
- Wir speichern diese Ereignisse: Wir halten die **Geschichte** fest
- Events als **einzige Wahrheit**
- Append only
- Events immutable

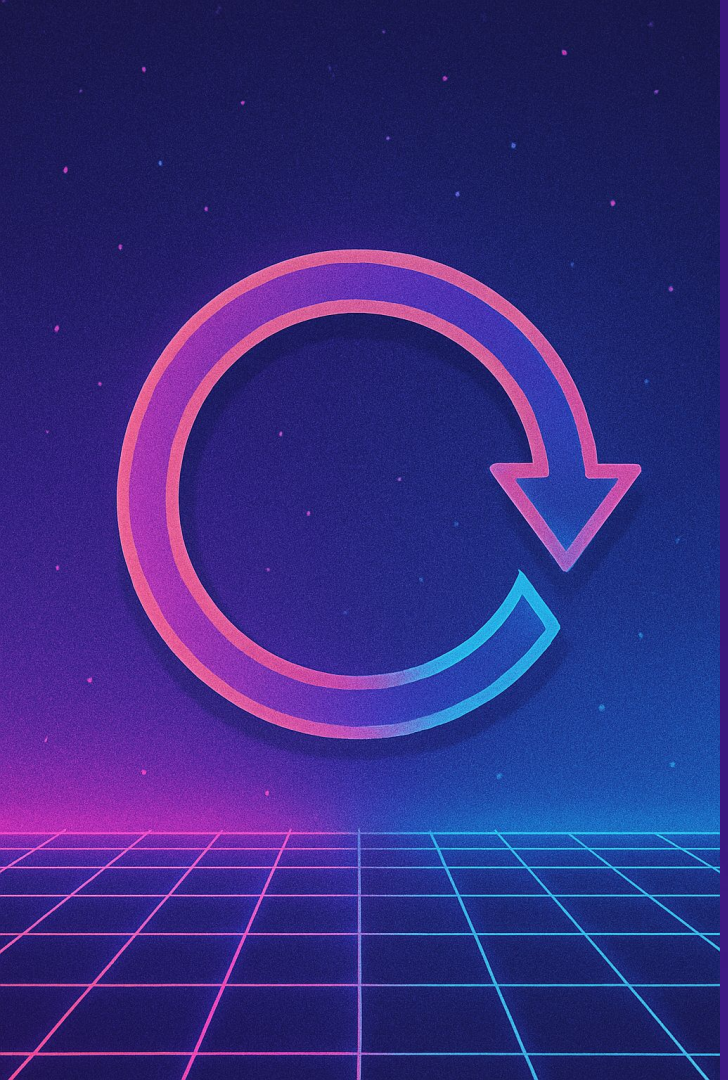


Zusammenfassung

CQRS:

Trennung von Verantwortlichkeit

- Lesen/Schreiben
- Lesen/Lesen
- Maßgeschneiderte Antworten geben



Zusammenfassung

- Mächtiger je **komplexer Domänen**
- Gut wenn Bedarf an **Historie/Audit**
- Kleiner Hinweis: **Kafka ist kein Event Store**

Weiterführende Quellen

OpenCQRS Framework:
https://github.com/open-cqrs/opencqrs

Event Sourcing DB:
https://eventsourcingdb.io/

Eric Evans - Domain Driven Design:
https://hds.hebis.de/hda/Record/HEB291420311

Vaughn Vernon - Implementing Domain Driven Design:
https://hds.hebis.de/hda/Record/HEB326245251

Microsoft Event Sourcing Learning Page:
https://learn.microsoft.com/de-de/azure/architecture/patterns/event-sourcing

OpenCQRS Live Coding Session:
https://www.youtube.com/watch?v=pkh7otw3F2M

Q&A

Stefan Schilling // stefan.schilling@digitalfrontiers.de

Jenseits von CRUD

Event Sourcing & CQRS als moderne Architektur-Antwort

Kersten Kriegbaum // kersten.kriegbaum@digitalfrontiers.de