codecentric

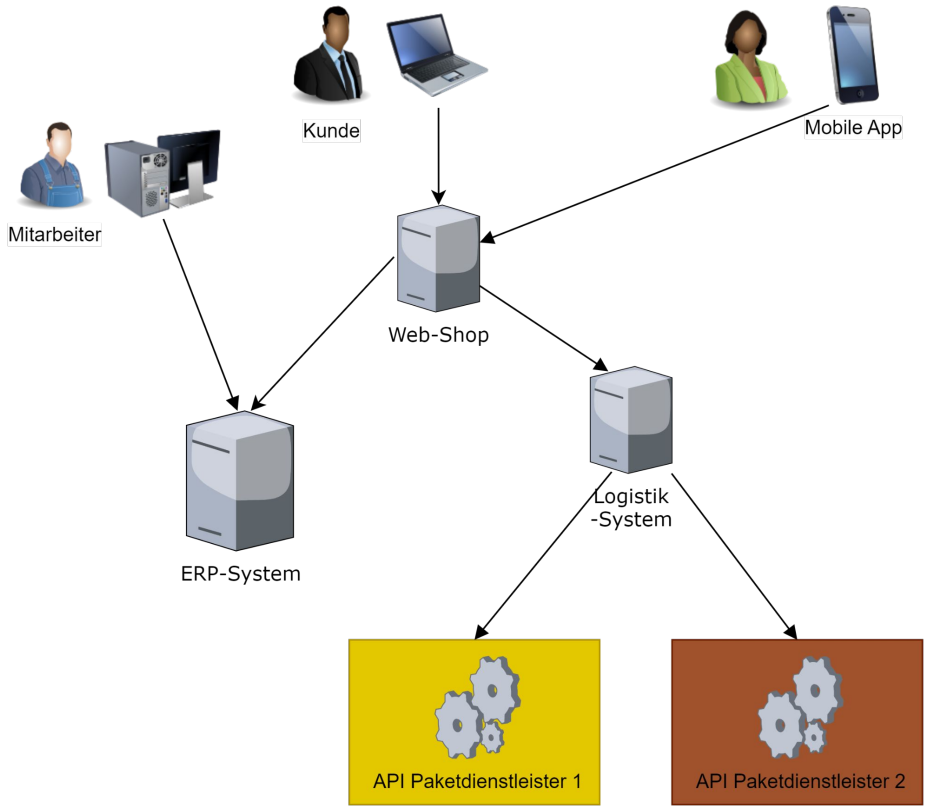
# Kommunikationsmuster für Services

Effizient und zuverlässig

Dr. Roger Butenuth

# IT-Landschaft - Beispiel

Use Case: Web-Shop, Abfrage Versandstatus




**Bisher einfach:**

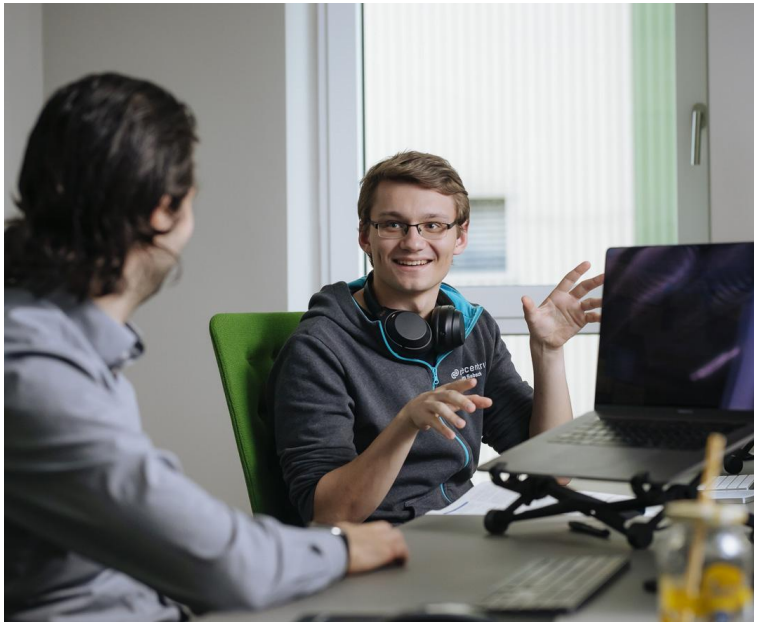
- Systeme mit *eigenen* Daten
- Lesezugriffe
- Ausfälle (fachlich) unkritisch

# Projektalltag…

Fachbereich hat Wünsche:

*“Ich will Versandbenachrichtigung”*

- Email (web shop)
- Push (mobile app)



# Lösungsvorschlag

## Naive Idee: Polling

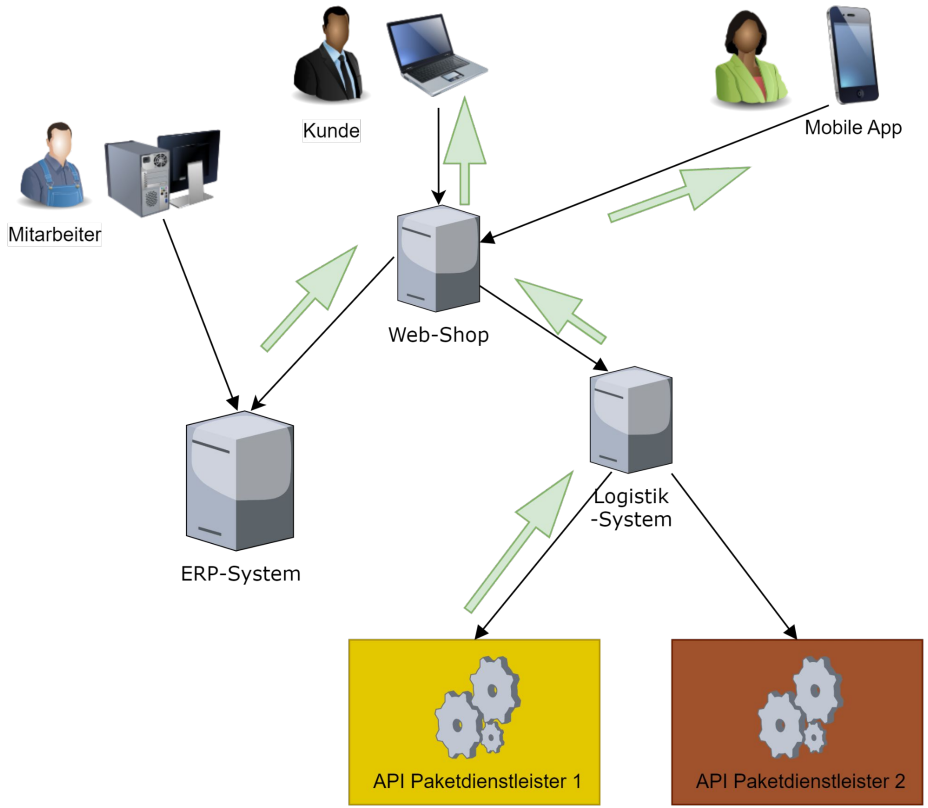
- Im Web-Shop
- In der App

## Probleme

- Rechenzeit
- Bandbreite in App
- Rate-Limit Paketdienst API
- Nie wirklich aktuell

# Bessere Lösung

Datenreplikation, Aktualisierung per Event




## Vorteile

- Aktuelle Daten
- Weniger Traffic
- Weniger Abhängigkeiten

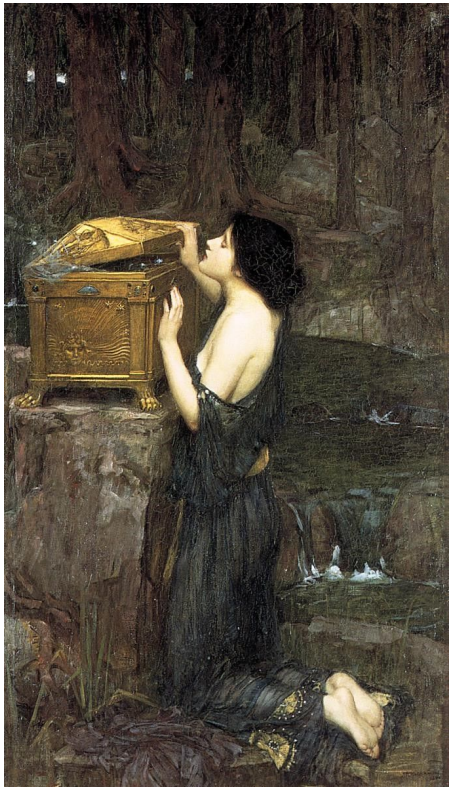
# Verteilte Datenhaltung

## Büchse der Pandora

1. Einmaligkeit von Ereignissen
2. Reihenfolgetreue
3. Konsistenz

## Aber auch Hoffnung

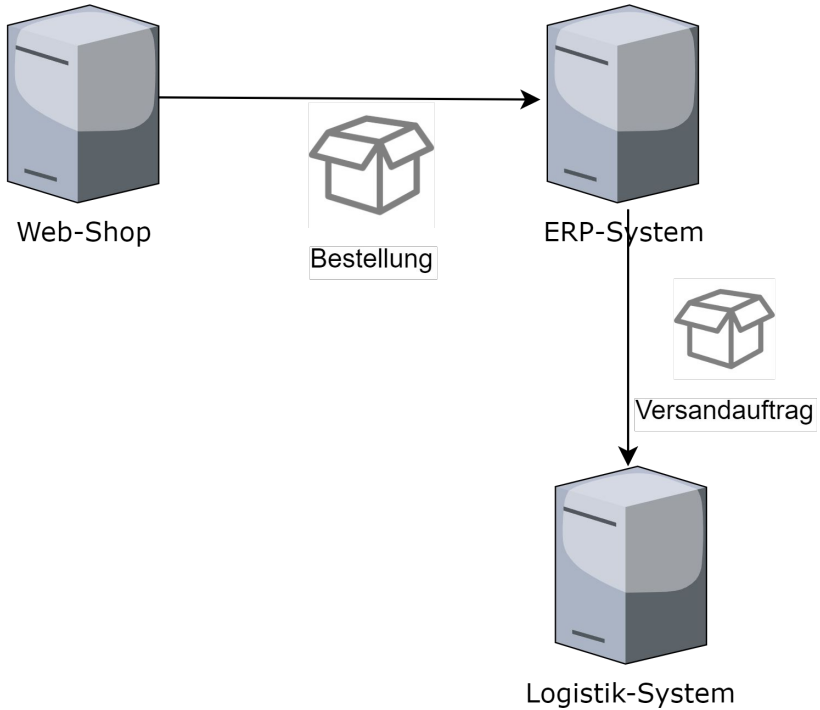
1. Niedrige Latenz
2. Hohe Leistung
3. Ausfallsicherheit



https://en.wikipedia.org/wiki/Pandora

# Ähnliche Problemstellung (I)

Aufträge




**Duplikate:**

Kunde freut sich ;-)

# Ähnliche Problemstellung (II)

Redundante (Stamm-)Daten


**Einfacher Fall:**

Ein System führend

# Wie sieht die Welt aus?

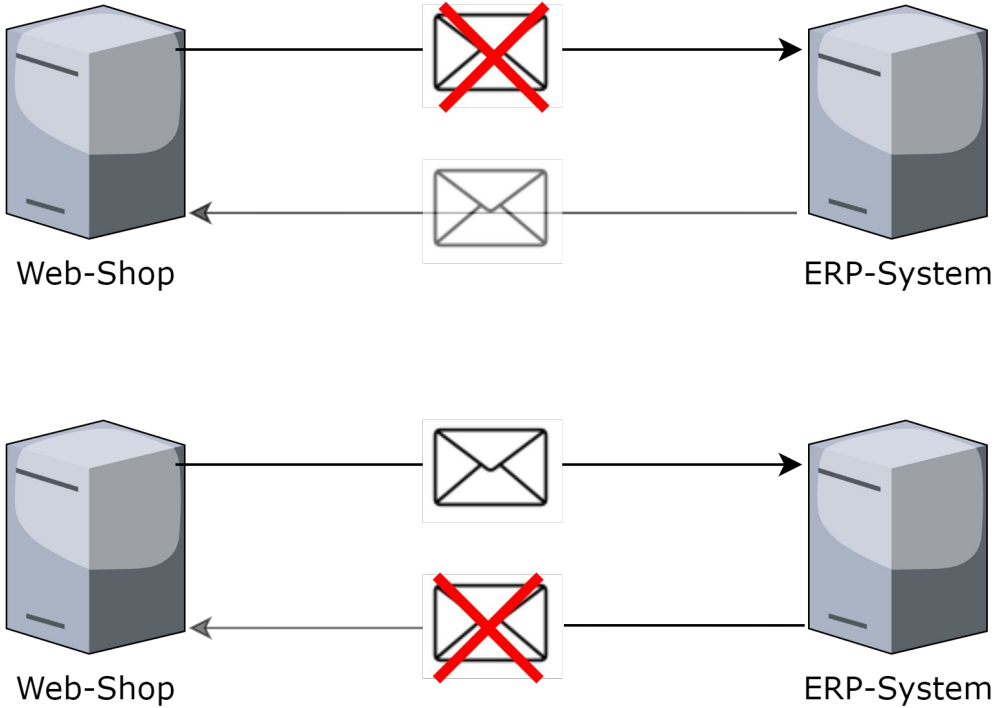
- Systeme mit eigener Datenhaltung
  ➔ kein SOA mit “Datenservices”
- Duplizierte Daten
- Gekaufte, “fertige” Systeme

## Randbedingungen

- Parallelität
- Netzwerkfehler
- Keine Reihenfolgetreue

# Exactly Once?

Auf Event-Ebene nicht möglich!

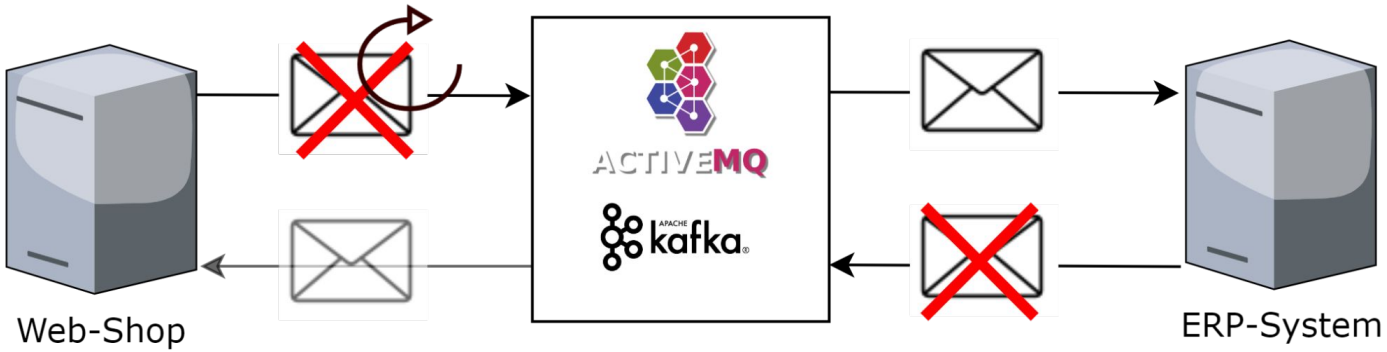



Fire and Forget:
At most once

Retry until Ack:
At least once

# Aber mit Event Broker?

Auch nicht…




- Andere Fehlermöglichkeiten
- Verschiebung des Problems
- Empfänger kontrolliert Last
- Retry beim Sender auch hier notwendig!

# Es gibt doch RAFT

Ja, aber…

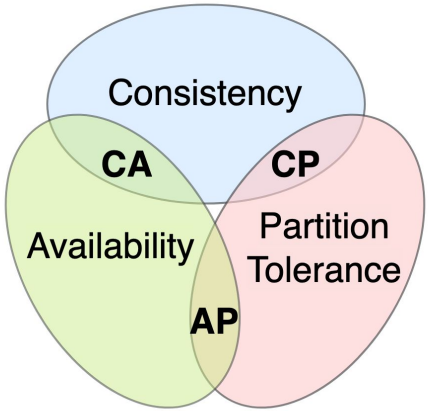
## Beispiel: RAFT in etcd (k8s)

- Key/Value store
- Request Größe < 1,5 MByte (Standard)
- Größe der Datenbank:
  - 2 GByte (Standard)
  - 8 GByte (empfohlenes Maximum)

➔ etcd: **CP**

➔ Shop, ERP, …: **AP** (mit eventual consistency)

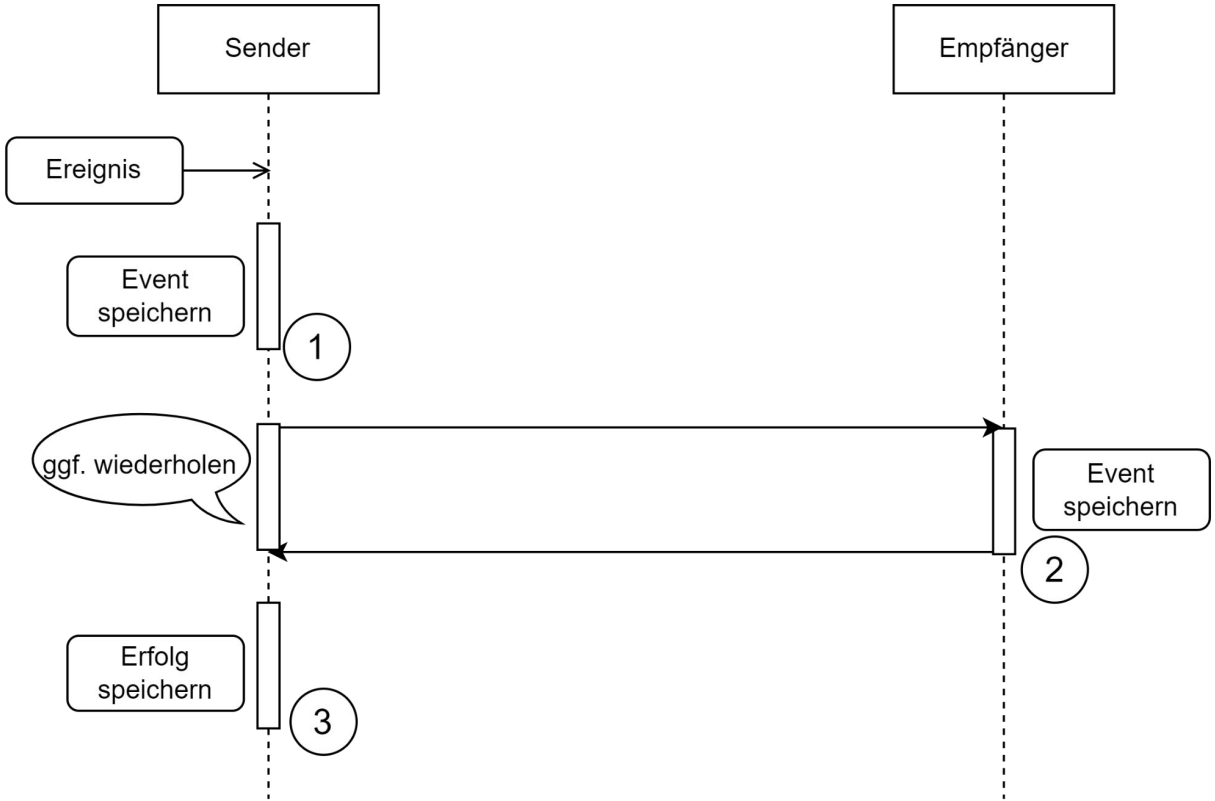
https://en.wikipedia.org/wiki/Raft_(algorithm)




https://en.wikipedia.org/wiki/CAP_theorem

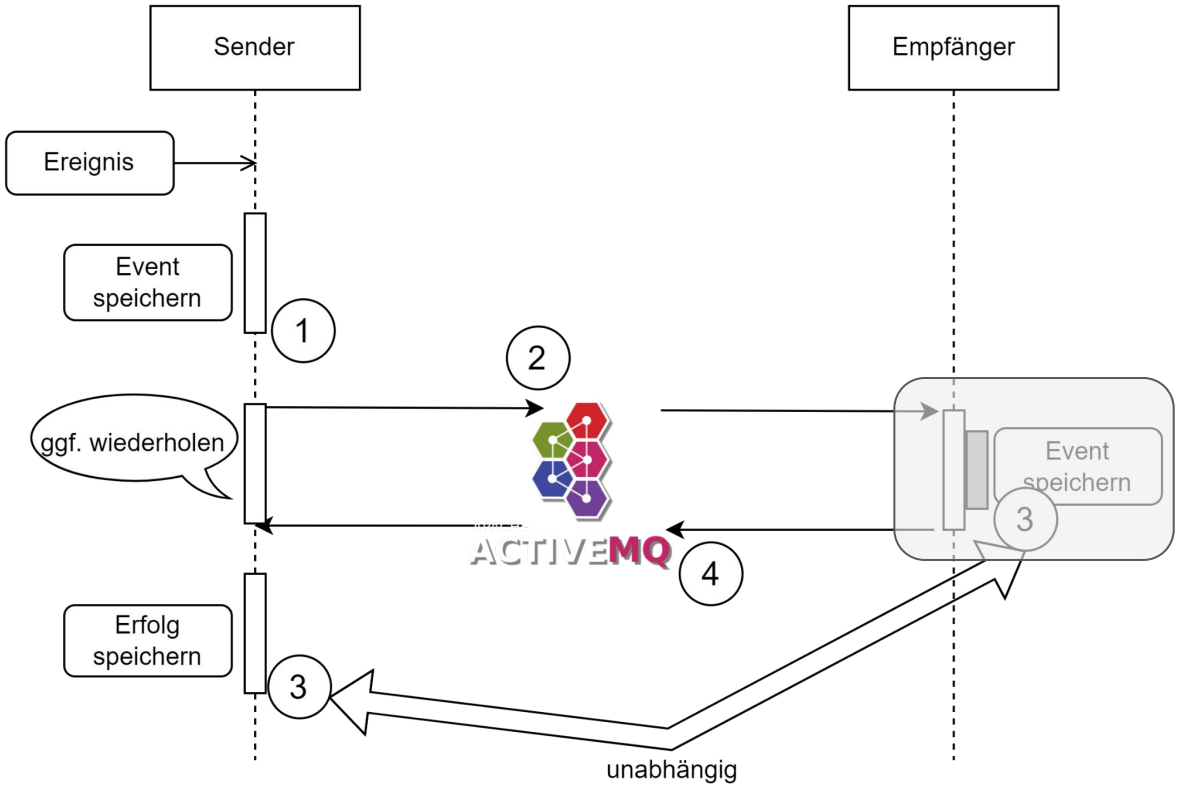
# Muster für “at least once”

HTTP POST, gRPC, …



# Muster für “at least once”

Mit Event-Broker




**XA?**

- empfange Event
- Event speichern

**Vorsicht:**

- Komplex
- Teuer

# Da war noch was: Reihenfolgetreue

Leider Illusion…

**Na und?**

- Egal: Lieferschein drucken
- Kritisch: Update (mit alten) Daten

**Ursachen**

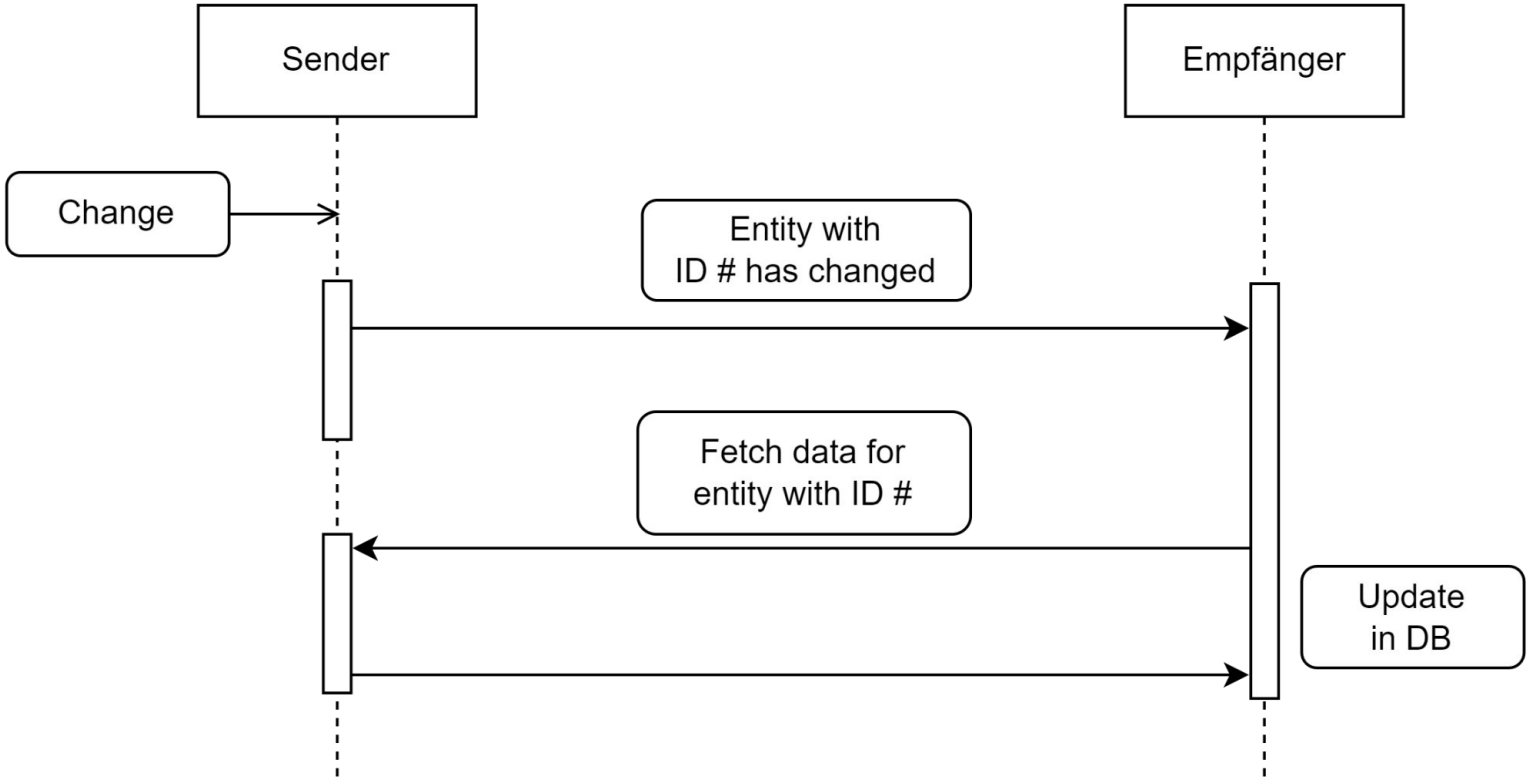
- Mehrere Threads (Sender / Empfänger)
- Broker nicht reihenfolgetreu
- Error-Queues
- Mehrere Queue-Consumer

**Erzwingbar**

(Jegliche) Parallelität verhindern (teuer)

# Lösung bei Datensynchronisation (I)

“ID only” Event



# Lösung bei Datensynchronisation (II)

Versionsnummern

**In Zielsystem**

- Schlüssel aus Quellsystem
- Version aus Quellsystem

**In Nachricht**

- Schlüssel Quellsystem
- Version Quellsystem
- (letzte bekannte Version Zielsystem)

| Some Table... | | | | |
|---|---|---|---|---|
| **PK** | **VS** | **FK** | **FVS** | **Data** |
| | | | | Row 1 |
| | | | | Row 2 |
| | | | | Row 3 |

- Ggf. eigene Synchronisationstabelle (speziell bei > 2 Systemen)
- Konflikterkennung möglich

# At least once: Duplikate

Deduplizierung oder Idempotenz

Idempotenz: Mehrfach ist egal

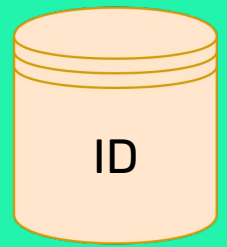
z.B. Aktualisierung in der Datenbank

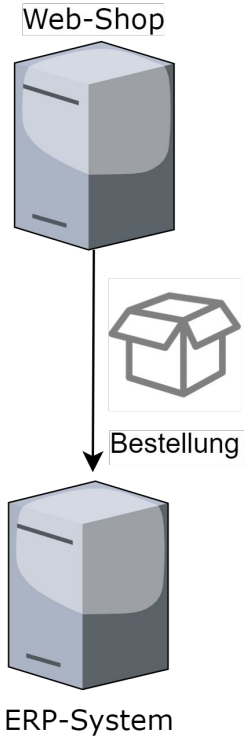
Deduplizierung: Bei Aktionen wichtig

z.B. Lieferschein drucken, Paket verschicken

Strategie:

- Eindeutige Event-ID
- Persistent speichern
- Duplikatcheck vor Folgeaktion



# Duplikaterkennung

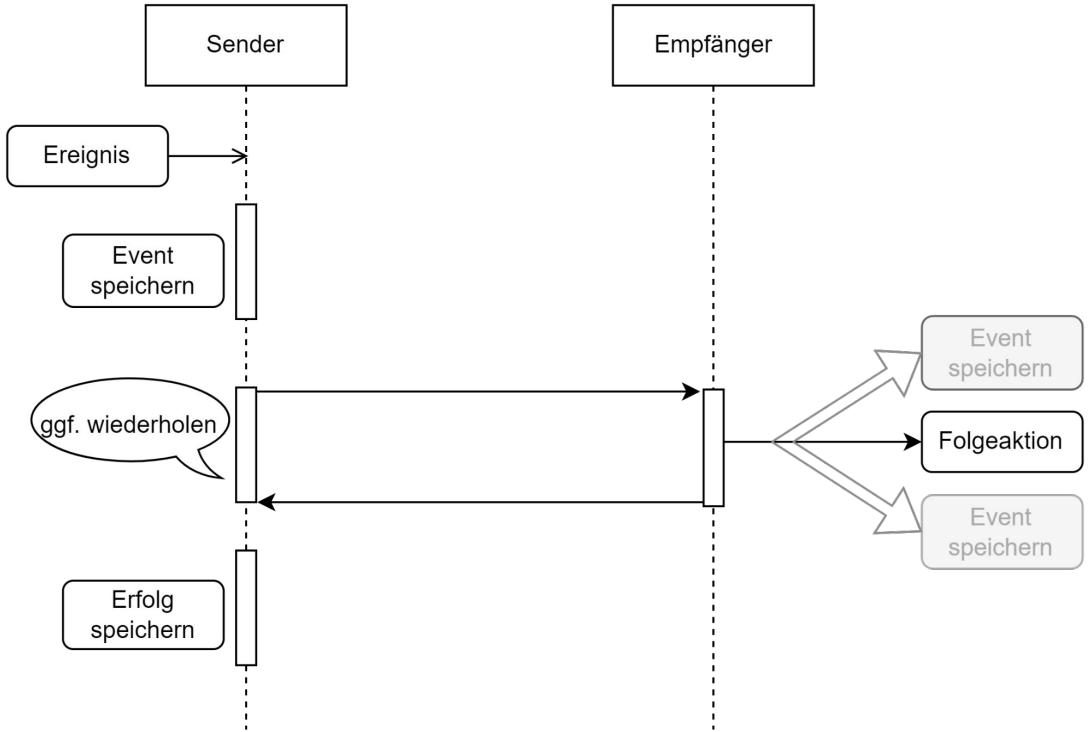
Leider nicht perfekt…

**Und wieder**

- At most once
- At least once

**Strategie**

- Bei Wiederholung warnen
- Prüfung
  - “nächste Instanz”
  - Zielsystem

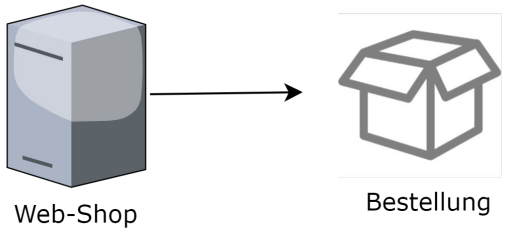


# Praxis

Wie implementiert man das?

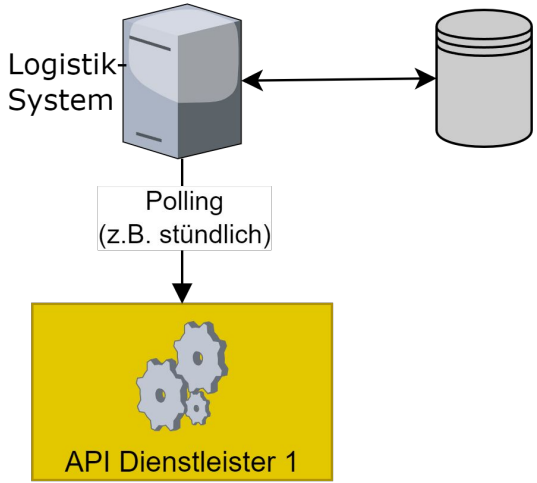
**Ideal**

- Event “serienmäßig”
- bei “eigener” Software nachrüstbar
- ggf. per “customizing”




**Polling**

- Zeitstempel (Änderung)
- Delta-Ermittlung
- Erkennung gelöschter Elemente?

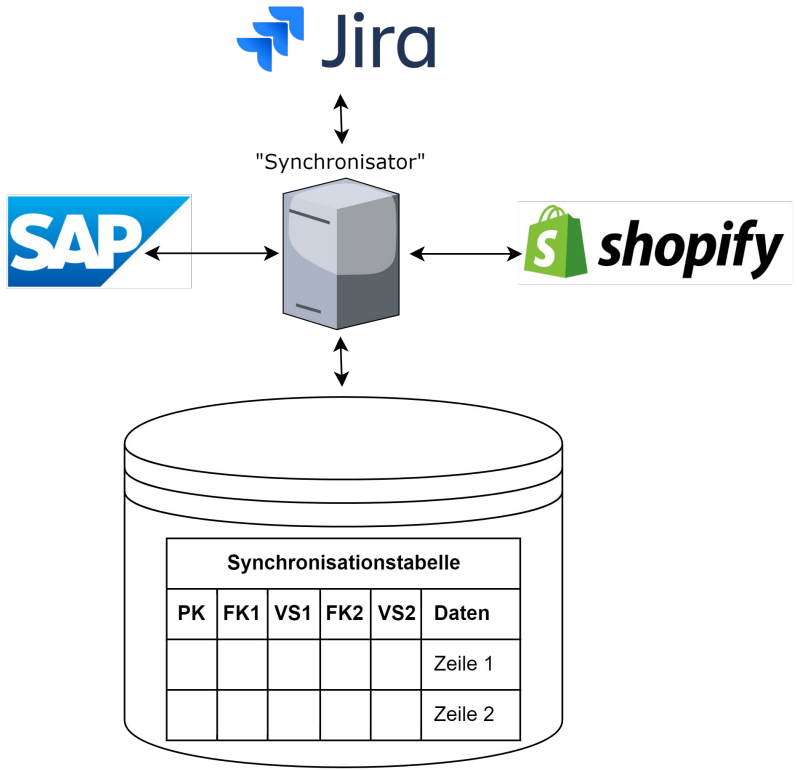


# Standardsoftware

Wenn man nicht ändern kann (oder will)

**System in der Mitte**

- Events generieren (ggf.)
- Daten verteilen
- Eigene Persistenz
- Formate umwandeln
- Konflikte melden



# Zusammenfassung

Es bleibt schwierig…

- Alles geht nicht, “richtigen” Kompromiss wählen
  - CP: konsistent, langsam
  - AP: verfügbar, schneller
- Das Framework regelt nicht alles
- Automatismen / Komplexität begrenzen

Danke!

# Fragen jetzt oder später beim Kaffee


codecentric

roger.butenuth@codecentric.de
www.codecentric.de

* Nordwijk Aan Zee, eigenes Foto