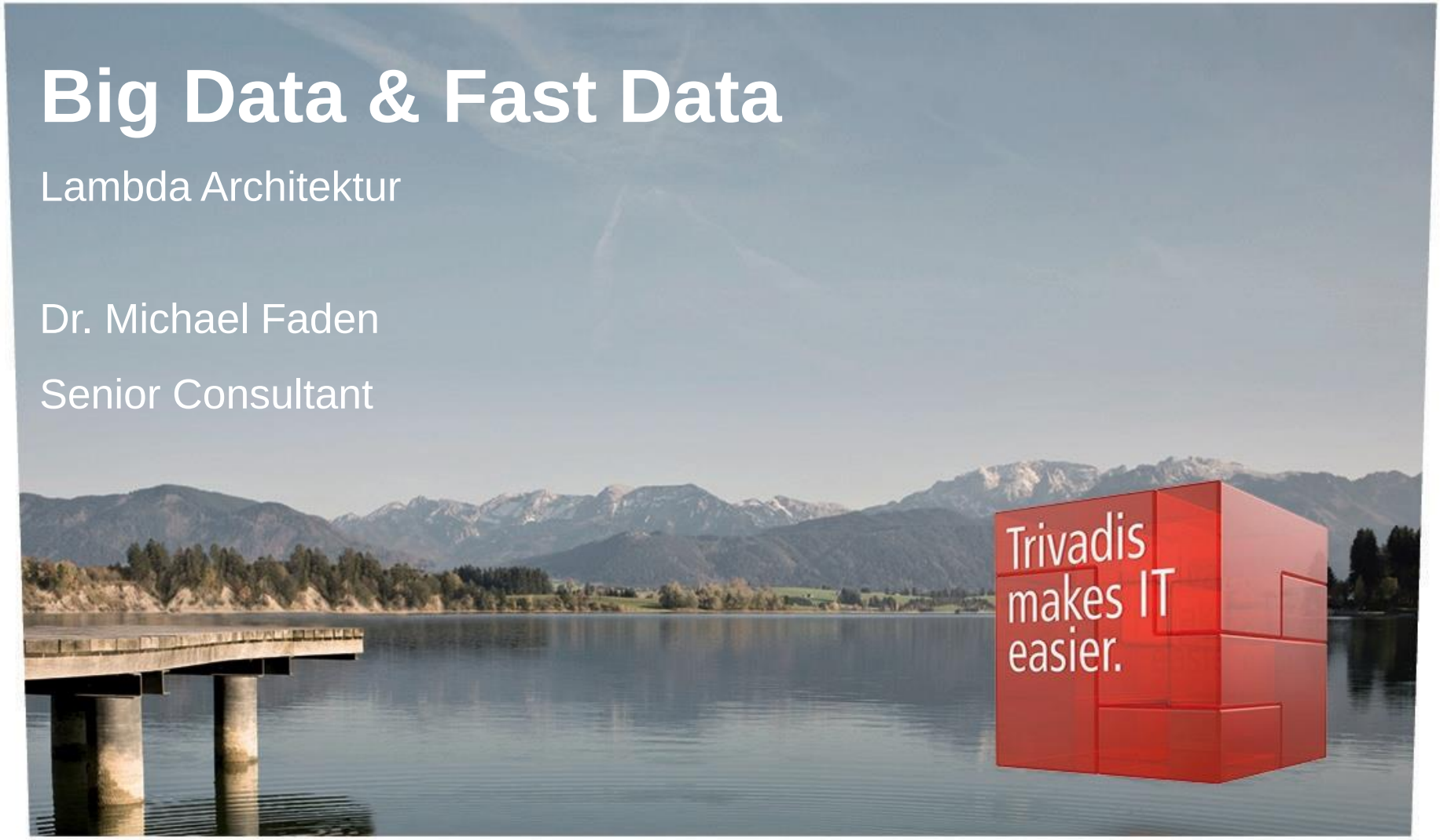
# Big Data & Fast Data

## Lambda Architektur

Dr. Michael Faden

Senior Consultant




BASEL BERN BRUGG LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN



20 JAHRE TRIVADIS We love IT.

trivadis makes IT easier.

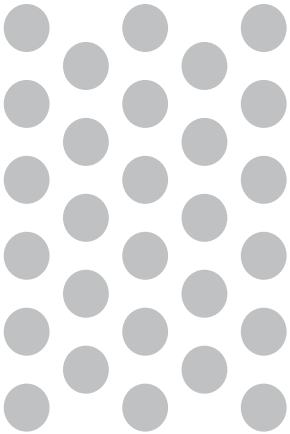
# AGENDA

1. Einleitung
2. Anforderungen an ein System
3. Lösungsansätze
4. Technologien

# Big Data Rekapituliert

## Volume

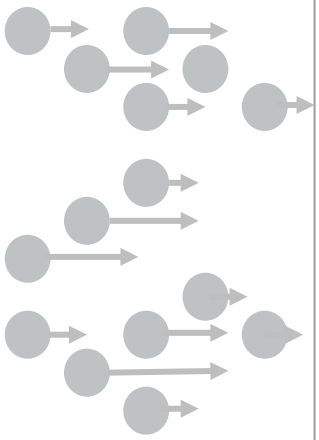
*Daten in Ruhe*



- Tera-, Peta- bis zu Exa-bytes verarbeiten
- Sensor Daten und Daten aus sozialen Netzwerken
- Neue Datenspeicher

## Velocity

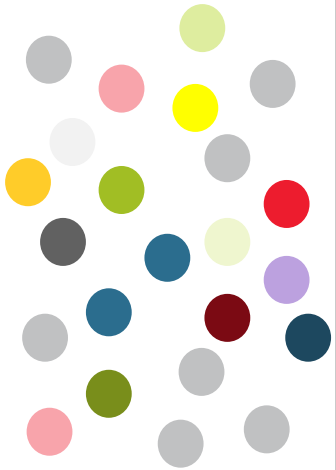
*Daten in Bewegung*



- Streaming Daten
- (Milli)Sekunden bis Minuten als Antwortszeit

## Variety

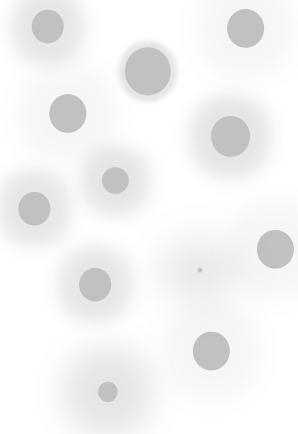
*Daten in versch. Form*



- Strukturiert und unstrukturiert
- Text, Zahlen, und Multimedia
- Viele Datenquellen

## Veracity

*Daten in Zweifel*



Zweifel durch

- Inkonsistenz
- Unvollständigkeit
- Widersprüchlichkeit
- Betrug
- Modell Approximationen

# Big Data Rekapituliert

## Volume

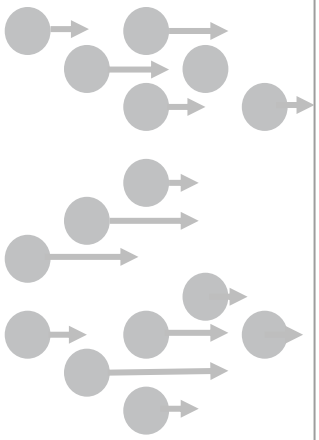
*Daten in Ruhe*



- Tera-, Peta- bis zu Exa-bytes verarbeiten
- Sensor Daten und Daten aus sozialen Netzwerken
- Neue Datenspeicher

## Velocity

*Daten in Bewegung*



- Streaming Daten
- (Milli)Sekunden bis Minuten als Antwortszeit

## Variety

*Daten in versch. Form*

- Strukturiert und unstrukturiert
- Text, Zahlen, und Multimedia
- Viele Datenquellen

## Veracity

*Daten in Zweifel*



Zweifel durch

- Inkonsistenz
- Unvollständigkeit
- Widersprüchlichkeit
- Betrug
- Modell Approximationen

# Beispiele

- Recommendation Engine
- Predictive Analytics
- Marketing Campaign Analysis
- Customer Retention and Churn Analysis
- Social Graph Analysis
- Capital Markets Analysis
- Risk Management
- Rogue Trading
- Fraud Detection
- Retail Banking
- Network Monitoring
- Research and Development
- …….

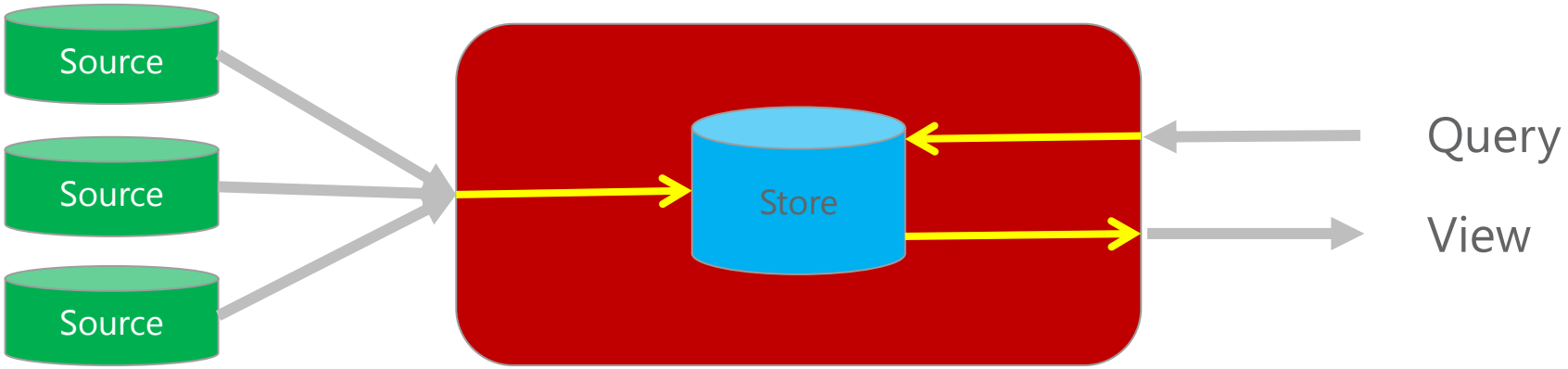
# Funktionale Anforderungen

- Das System erfasst die anfallenden Daten.
- Das System speichert die erfassten Daten permanent.
- Das System liefert die Daten für  beliebige Abfragen.

# Nicht-funktionale Anforderungen

- Robust
- Fehlertolerant
- Create/Read mit kurzer Latenzzeit
- Skalierbar
- Erweiterbar
- Allgemeingültig
- Minimaler Wartungsaufwand

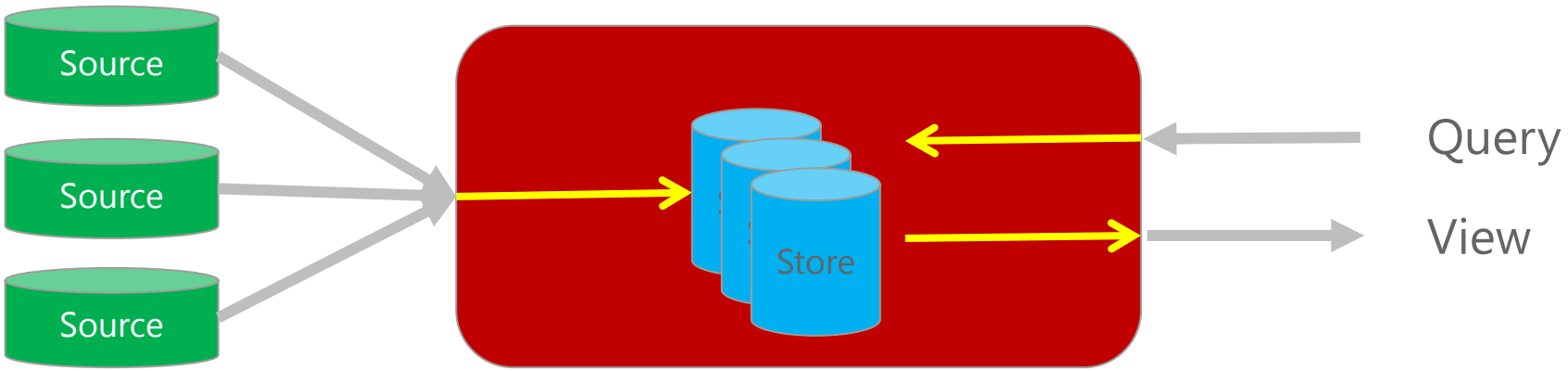
# Ansatz



# Nicht-funktionale Anforderungen

- Robust
- Fehlertolerant
- Create/Read mit kurzer Latenzzeit
- Skalierbar
- Erweiterbar
- Allgemeingültig
- Minimaler Wartungsaufwand

# Replikation



# Nicht-funktionale Anforderungen

- Robust
- Fehlertolerant
- Create/Read mit kurzer Latenzzeit
- Skalierbar
- Erweiterbar
- Allgemeingültig
- Minimaler Wartungsaufwand

# Datenmodifikation

- **U** und **D** in CRUD
- Aktualisiert den jeweiligen Zustand

➔Gefahr von Datenverlust oder -verfälschung

| Name | Wohnort |
|---|---|
| Gustav | Köln |
| Karl | Bonn |

| Name | Wohnort |
|---|---|
| Gustav | Aachen |
| Karl | Bonn |

# Unveränderliche Daten (Master Data)

- Speichert historische Records von Events
- Ein Event ereignet sich zu einer bestimmten Zeit und ist immer wahr

➔ Es gibt keine Duplikate

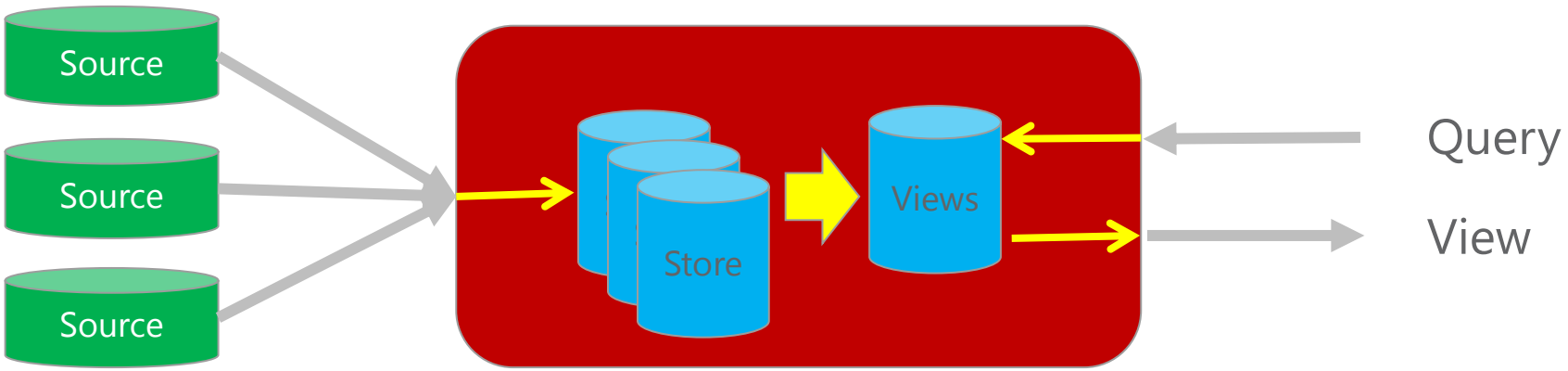
| Name | Wohnort | Zeitstempel |
|---|---|---|
| Gustav | Köln | 1.1.2001 |
| Karl | Bonn | 12.7.2003 |

| Name | Wohnort | Zeitstempel |
|---|---|---|
| Gustav | Köln | 1.1.2001 |
| Karl | Bonn | 12.7.2003 |
| Gustav | Aachen | 23.9.2013 |

# Nicht-funktionale Anforderungen

- Robust
- Fehlertolerant
- Create/Read mit kurzer Latenzzeit
- Skalierbar
- Erweiterbar
- Allgemeingültig
- Minimaler Wartungsaufwand

# Separation von Store und (pre-computed) Views



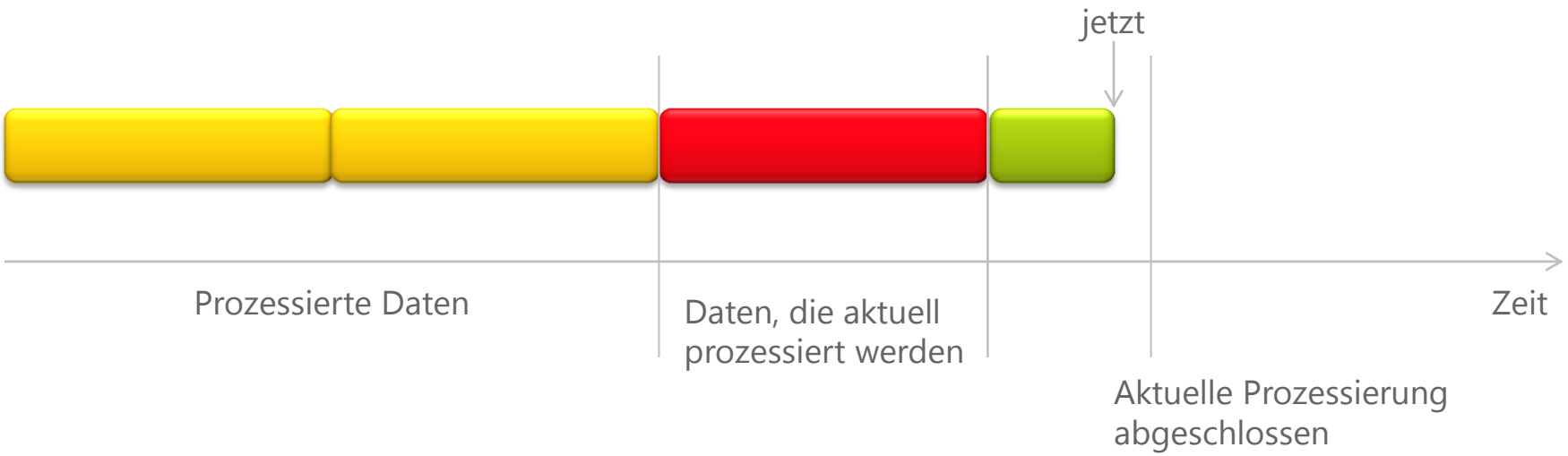

Command Query Responsibility Segregation (CQRS)

# Pre-computed Views

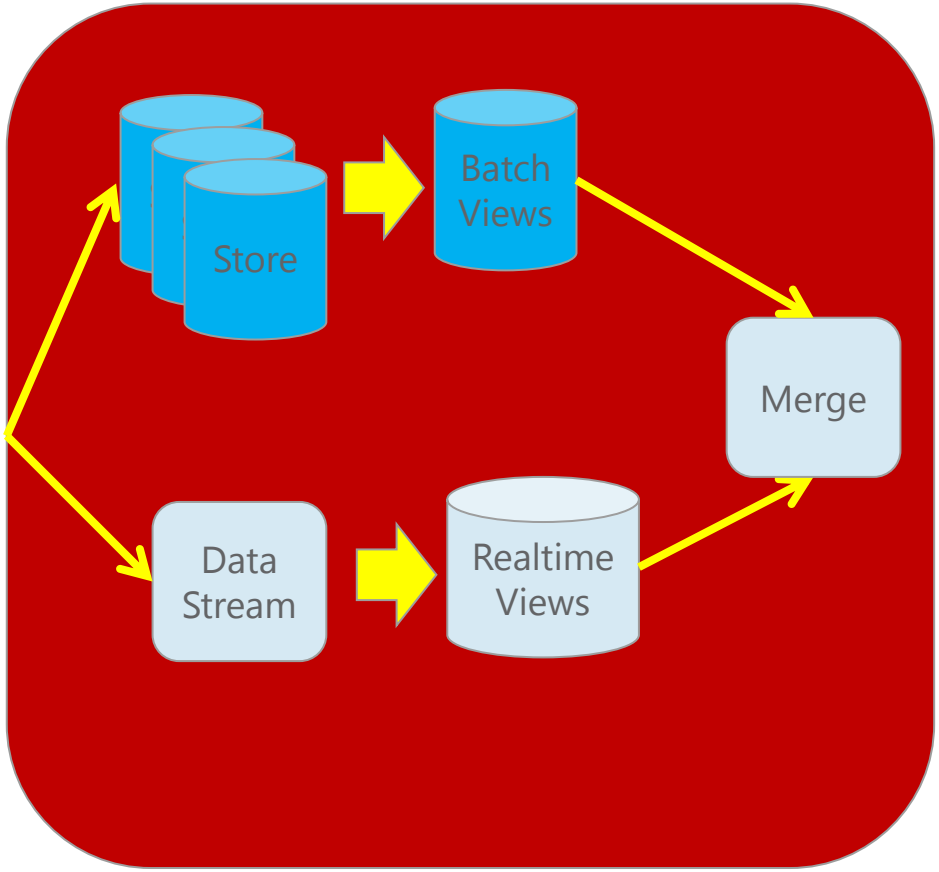
Query = f (all data)

- Vollständige Neuberechnung der Views
- Inkrementelle Aktualisiererung der Views

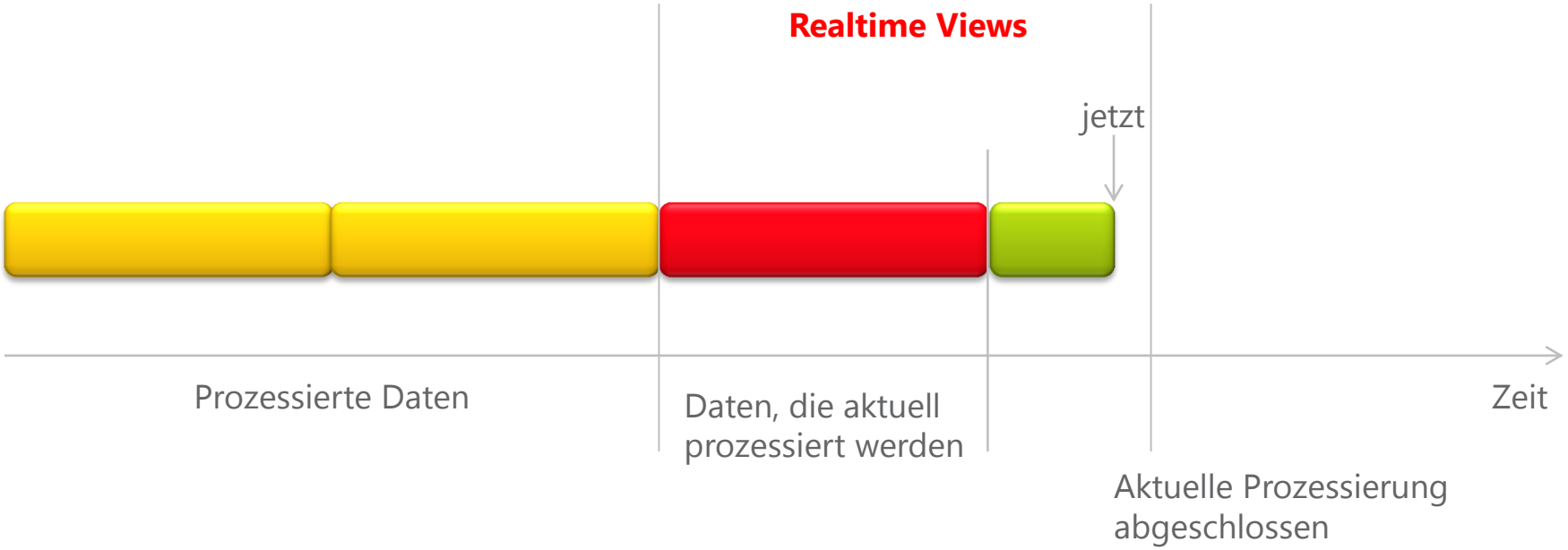
# Prozessierungslücke



# Realtime Processing



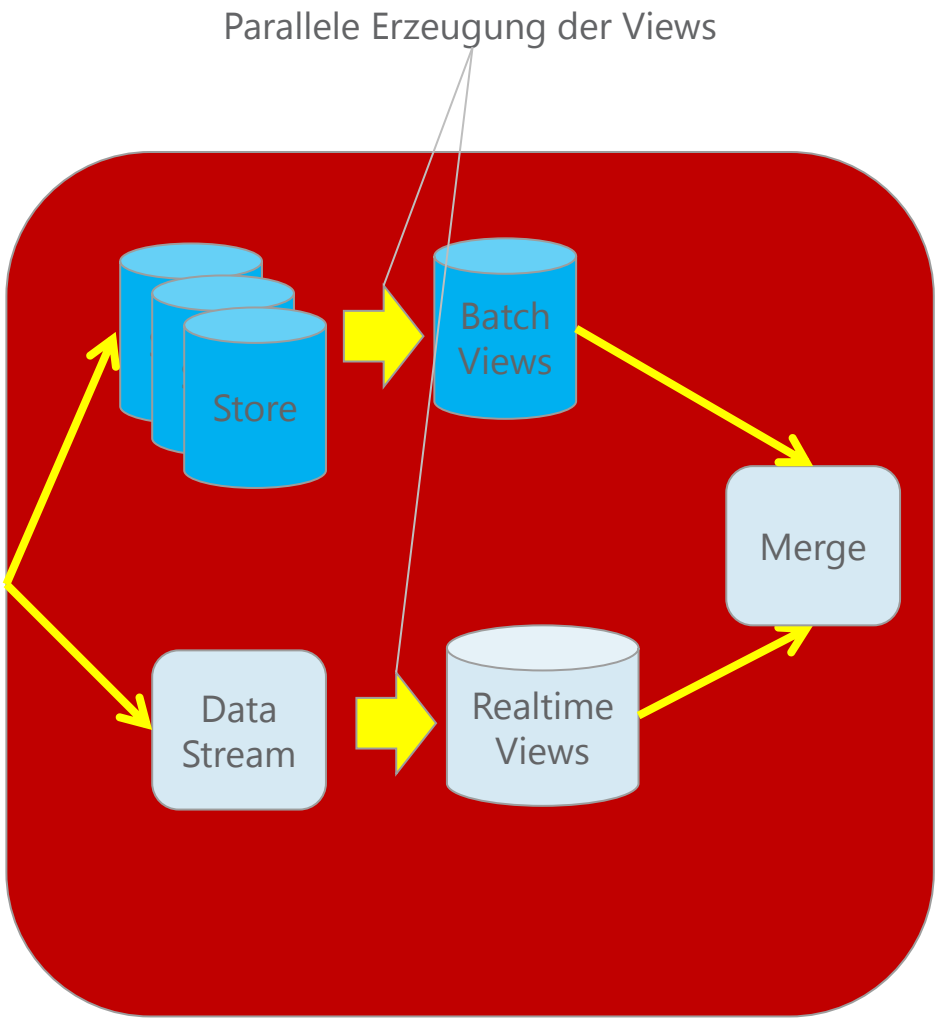
# Schließen der Prozessierungslücke



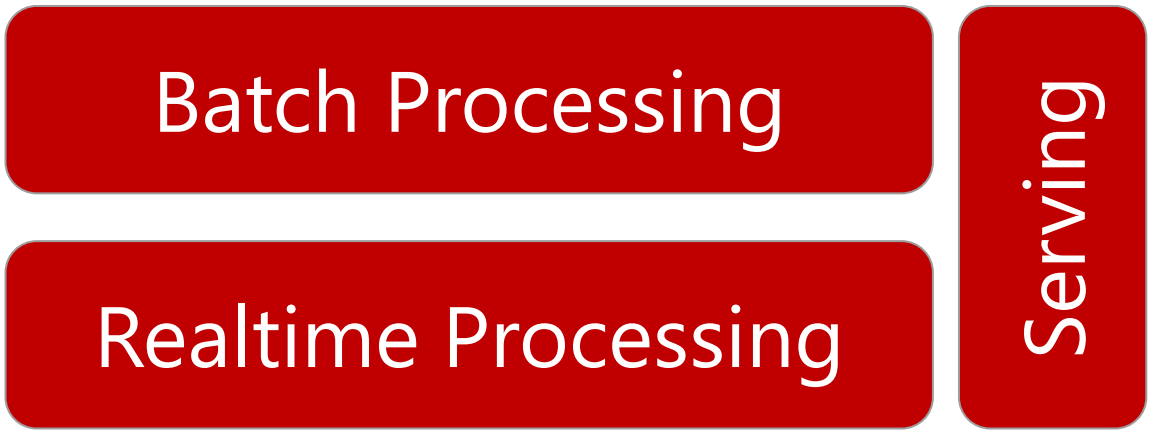
# Nicht-funktionale Anforderungen

- Robust
- Fehlertolerant
- Create/Read mit kurzer Latenzzeit
- Skalierbar
- Erweiterbar
- Allgemeingültig
- Minimaler Wartungsaufwand

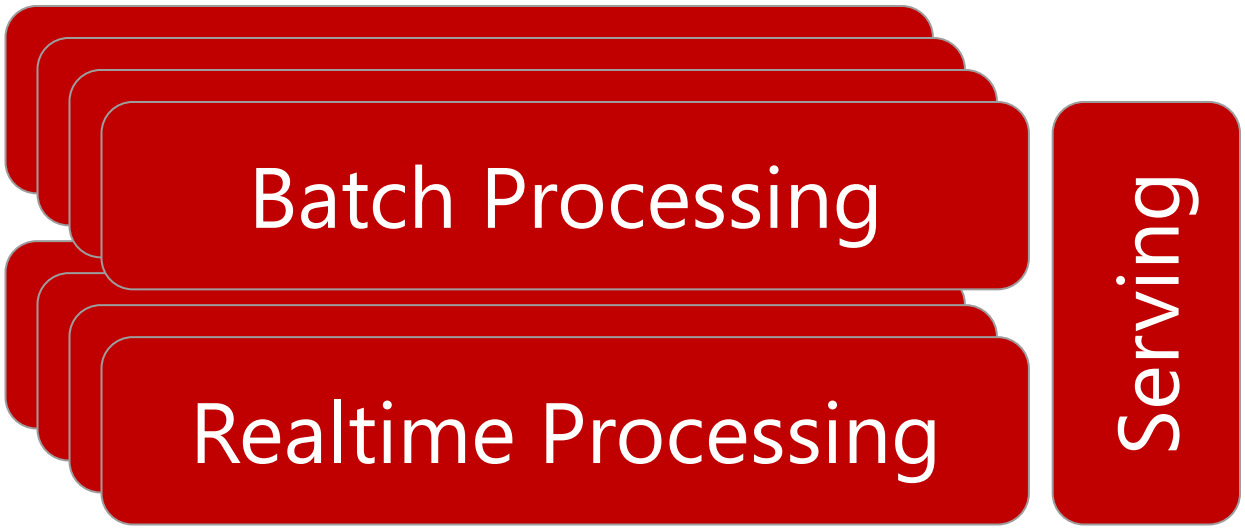
# Skalierbarkeit I



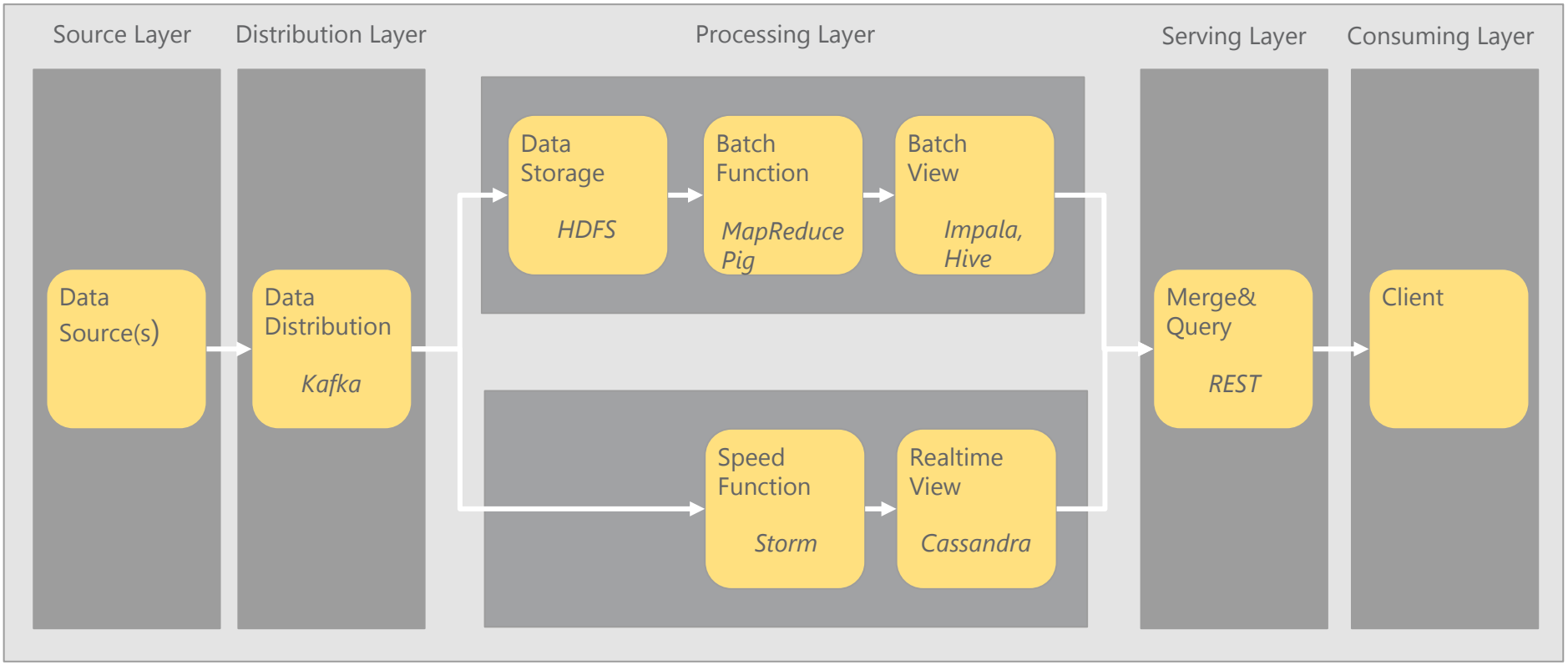
# Skalierbarkeit II



# Skalierbarkeit II

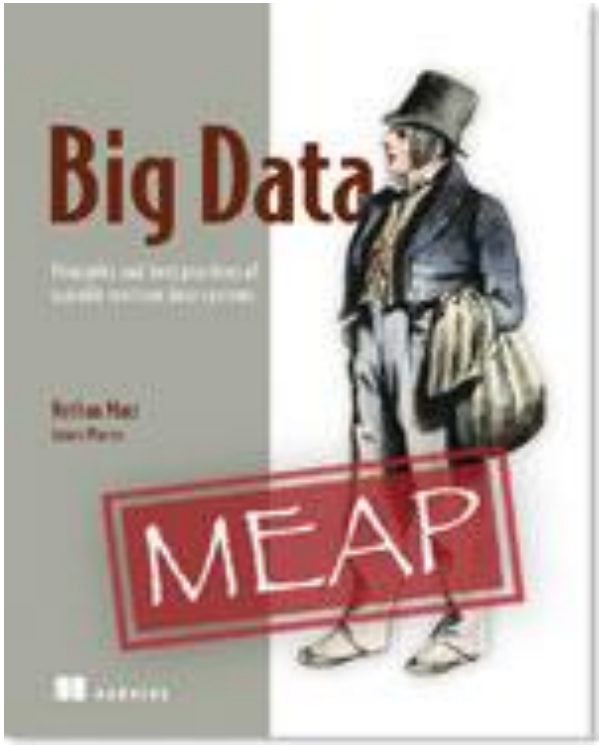


# Lambda Architektur



# Lambda Architektur

Geprägt von Jonathan Marz



# Zwischenbemerkung

Funktionale Programmierung (λ–Kalkül):

- Jeder Ausdruck und Wert wird als ausführbare Funktion betrachtet
- Query = f(allData)
- Kein Zugriff auf einzelne Daten, sondern nur Queries

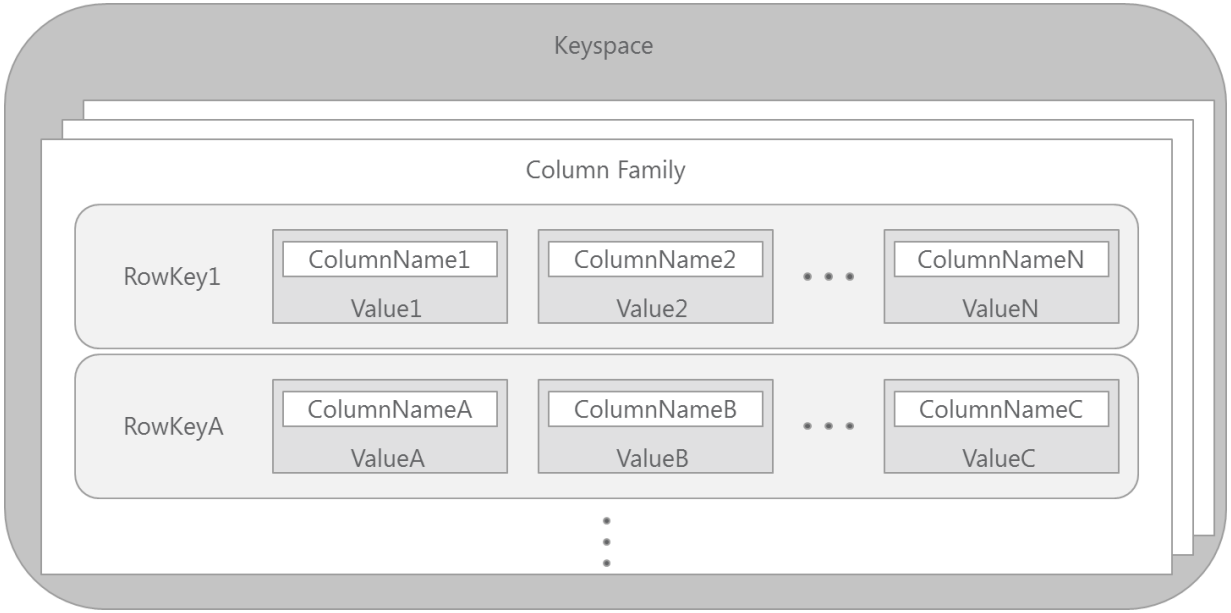
# Storm

- Storm (Apache Projekt) ist eine verteilte, fehlertolerante Realtime-Plattform
- Schlüsselkonzepte
  - **Tuple**: geordnete Liste von Elementen
  - **Streams**: unbegrenzte Sequenz von Tupeln
  - **Spouts**: Quellen von Streams
  - **Bolts**: Prozessieren Tuples und erzeugen neue Streams
  - **Topologien**: Gerichteter Graph von Spouts und Bolts

# Cassandra

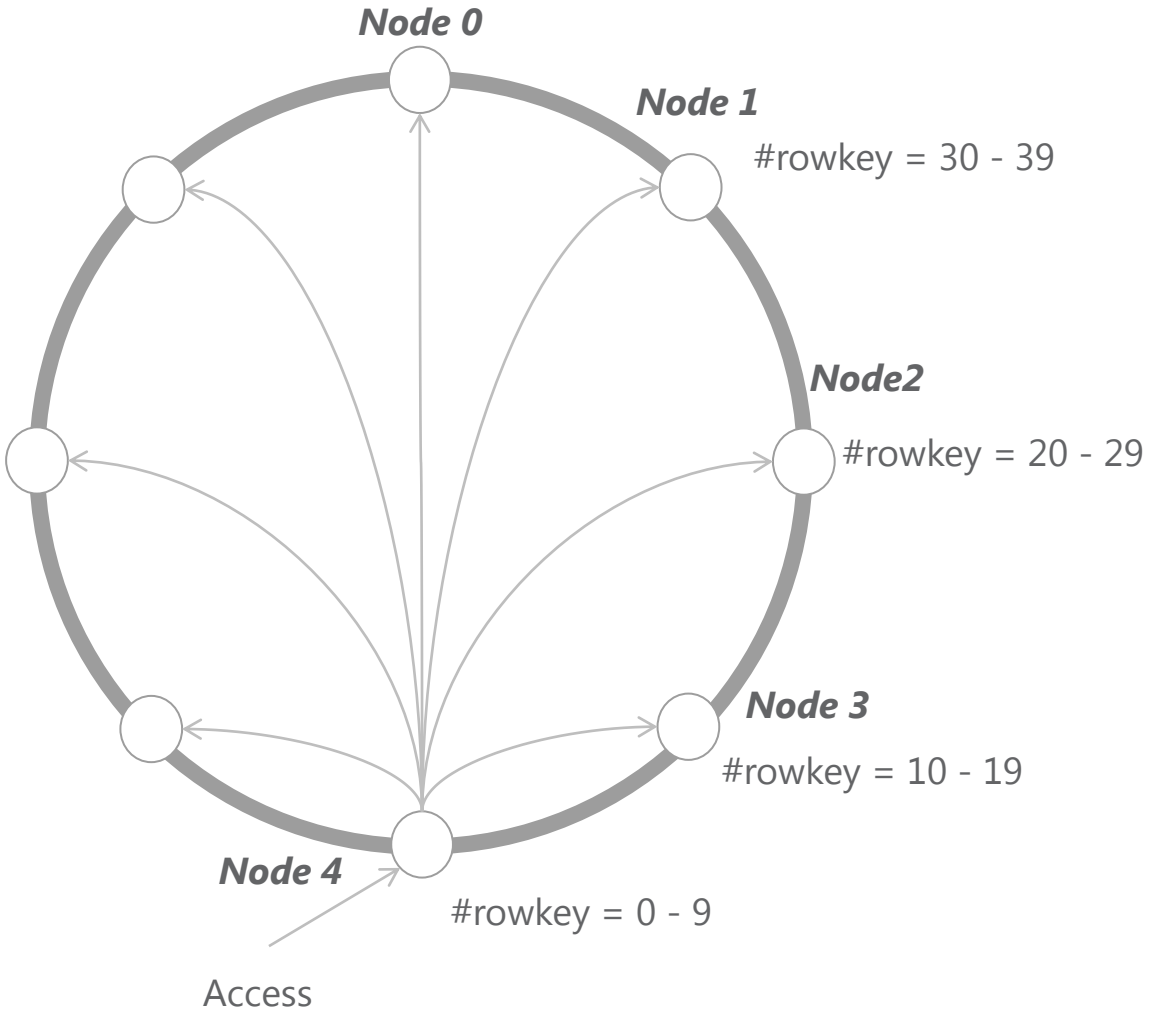

- Cassandra ist ein *Wide Column Store*
- Skaliert auf einer beliebigen Zahl von Knoten
- *Eventualy consistent*
- Entwicklung begonnen bei Facebook, jetzt ein Apache Toplevel Project
- Erweiterungen und Support durch Datastax

# Cassandra: Datenmodell



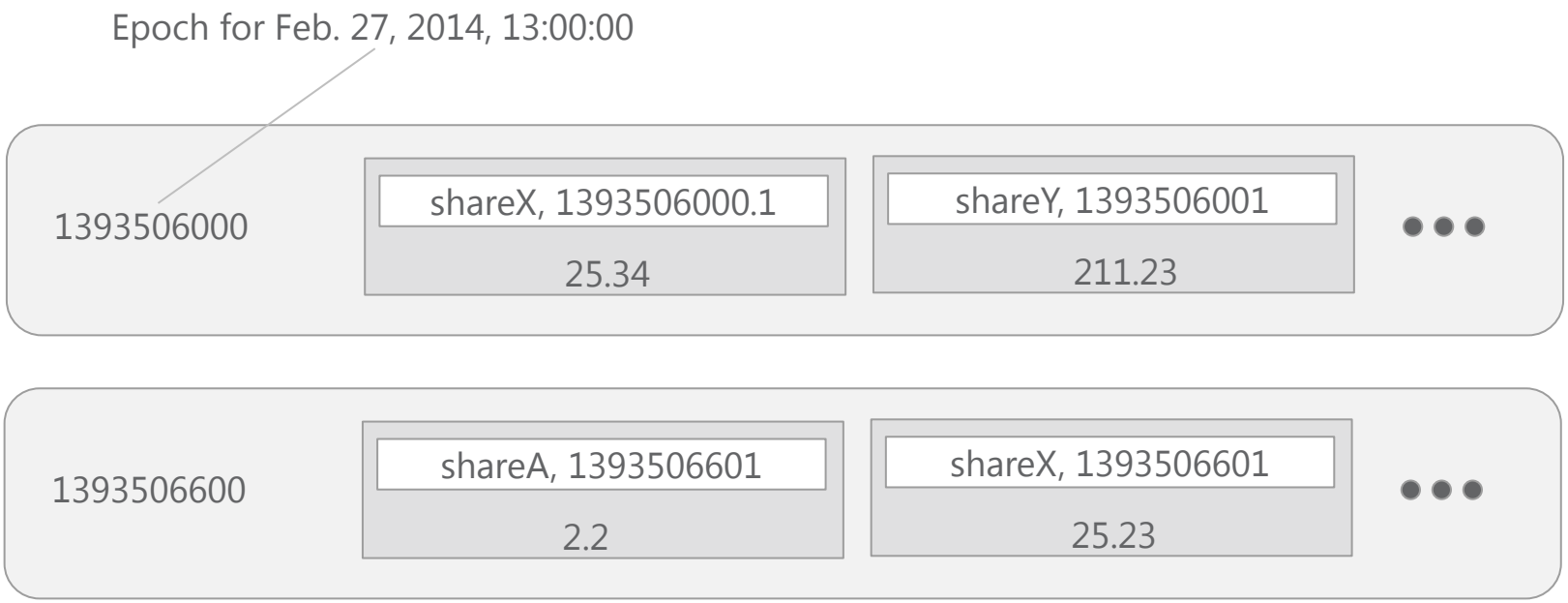

- Jede Zeile wird auf einem Knoten gespeichert
- *Column* ist ein Schlüssel: kann für jede Spalte anders sein
- *Composite Columns* z.B.: *{Timestamp, SensorID, Location}*

# Cassandra: Ring Topologie (example)

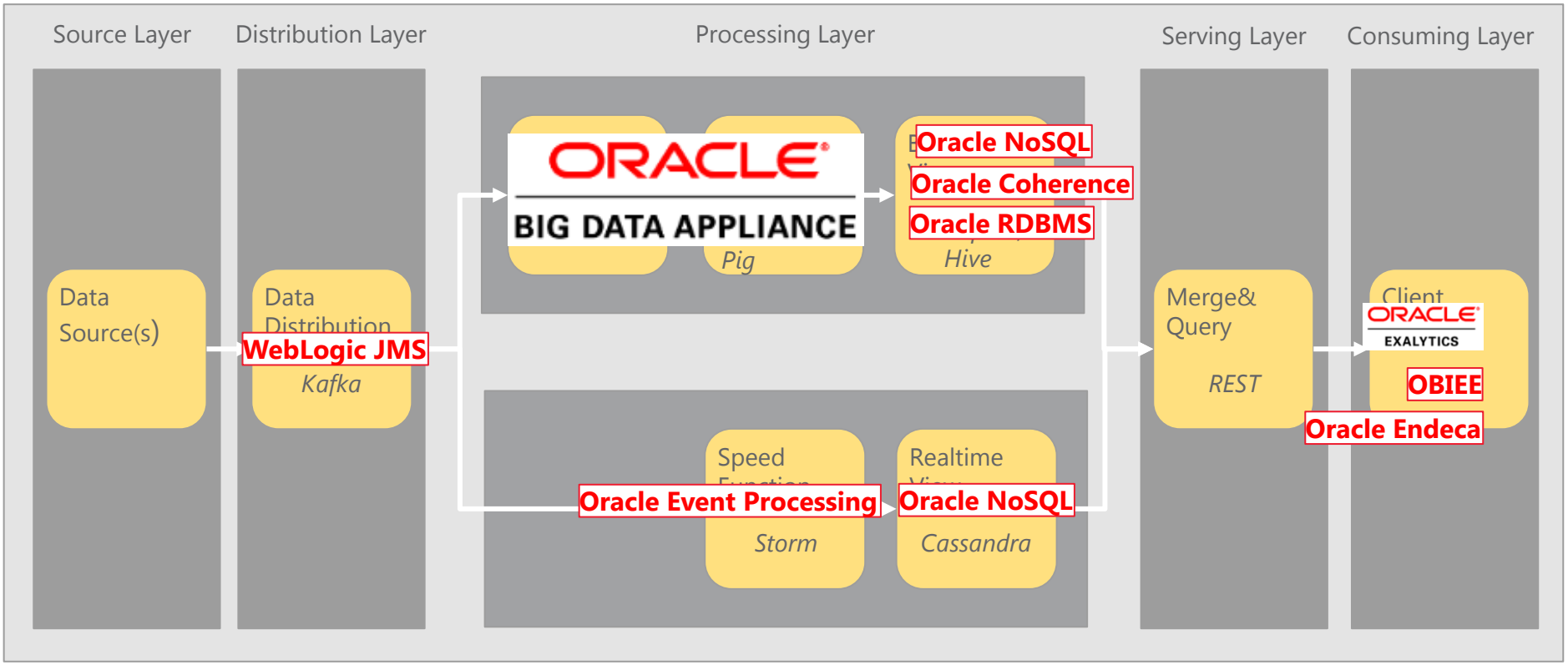



- Symmetrische Knoten
- Automatische Replikation
- Konfigurierbare Lesekonistenz
- Dynamische Änderungen von Knoten (Fehler, Erweiterungen..)
- Peer-to-Peer Kommunikation

# Cassandra: Time Series (Bucketing)



# Lambda Architektur



# Zusammenfassung

- Lambda Architektur ist eine Kombination von *CQRS* und *Tee-Pipe-and-Filters*.
- Verteilt, skalierbar, kein Single-Point-of-Failure
- Flexibel anpassbar
- Lässt sich mit verschiedenen Technologien realisieren.

# Fragen und Antworten...

Dr. Michael Faden

Senior Consultant

+49 -211- 58 666 47 06

michael.faden@trivadis.com


BASEL BERN BRUGG LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN