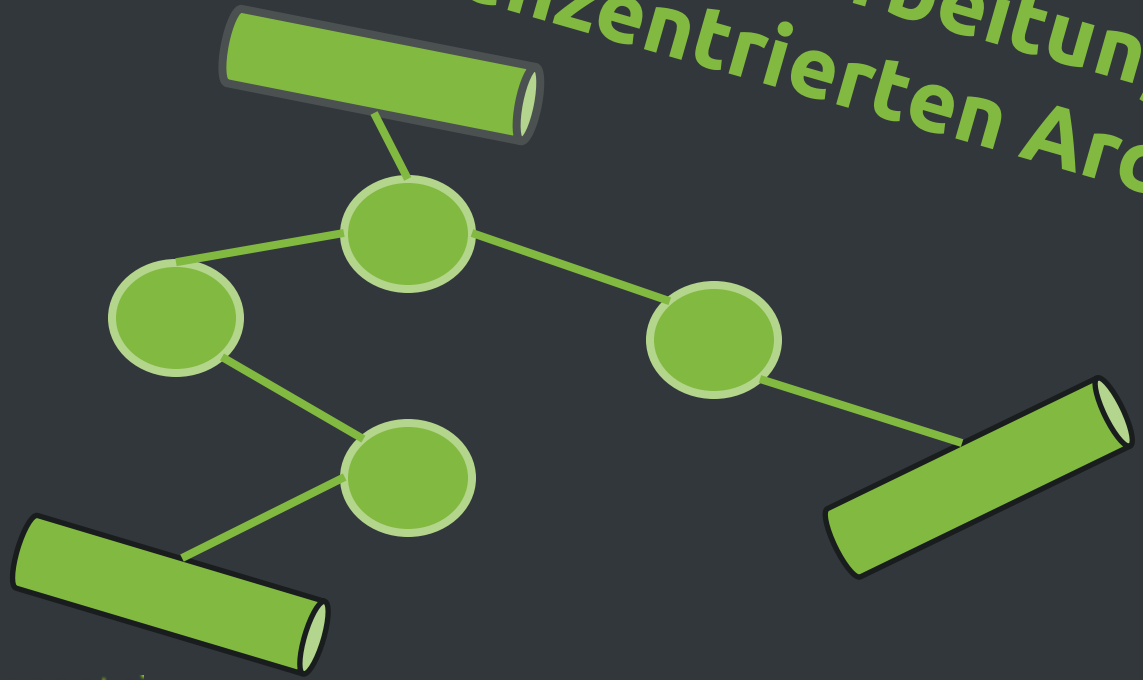
Echtzeitverarbeitung in einer datenzentrierten Architektur



esentri

Thomas Müller

esentri AG
Pricipal Consultant

@thomas-müller-39570099

Code

https://gitlab.com/thm-esentri/vortraege/vortrag-flink-kafka

Die Welt der Daten

Quelle: Christoph Kolumbus Wikipedia

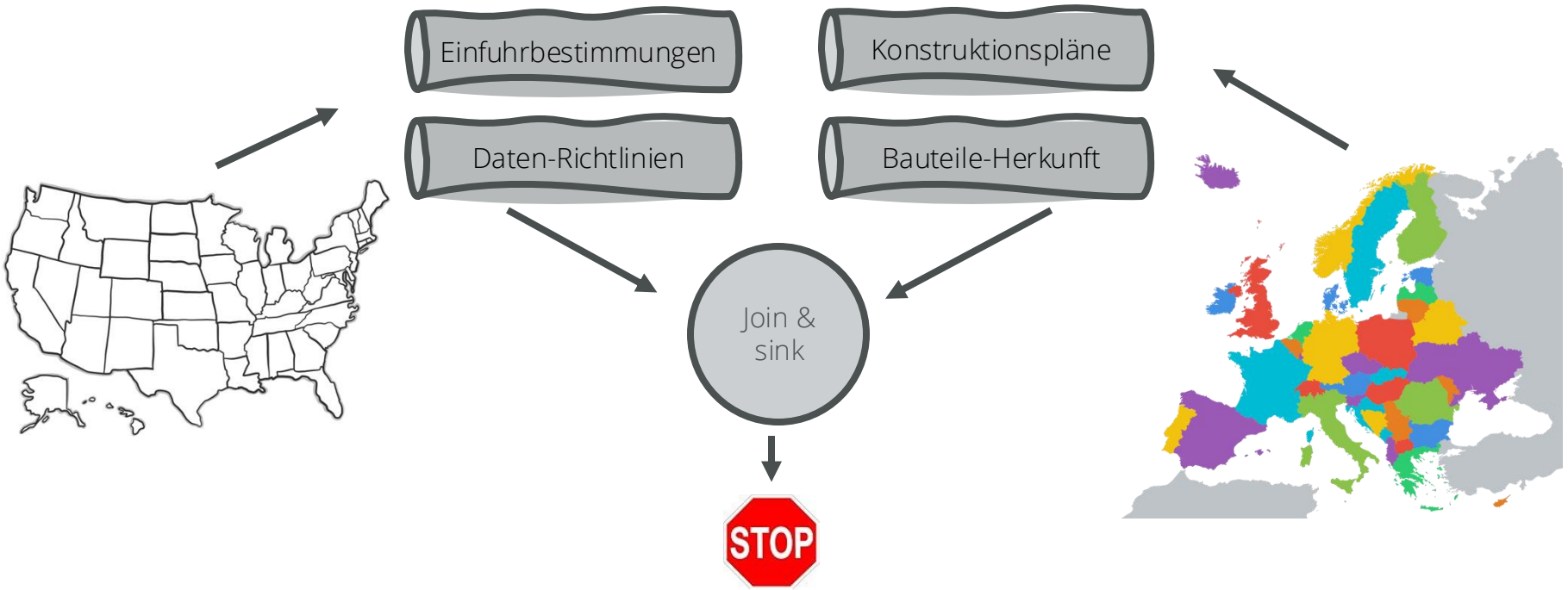
Zuverlässige Informationen sind unbedingt nötig für das Gelingen eines Unternehmens.

Christoph Kolumbus

Datenzentrierte Architektur - Use-Case

Quelle: Datenzentrisch durch den Regulierungsdschungel, Stefan Brock

Datenzentrierte Architektur - Use-Case



Quelle: Datenzentrisch durch den Regulierungsdschungel, Stefan Brock

Was ist Data Streaming?

> Data Streaming ist der kontinuierliche Fluss von Echtzeitinformationen und stellt die Basis des Softwaremodells der **event-gesteuerten** Architektur dar.

Quelle: REDHAT – Was ist Data Streaming?

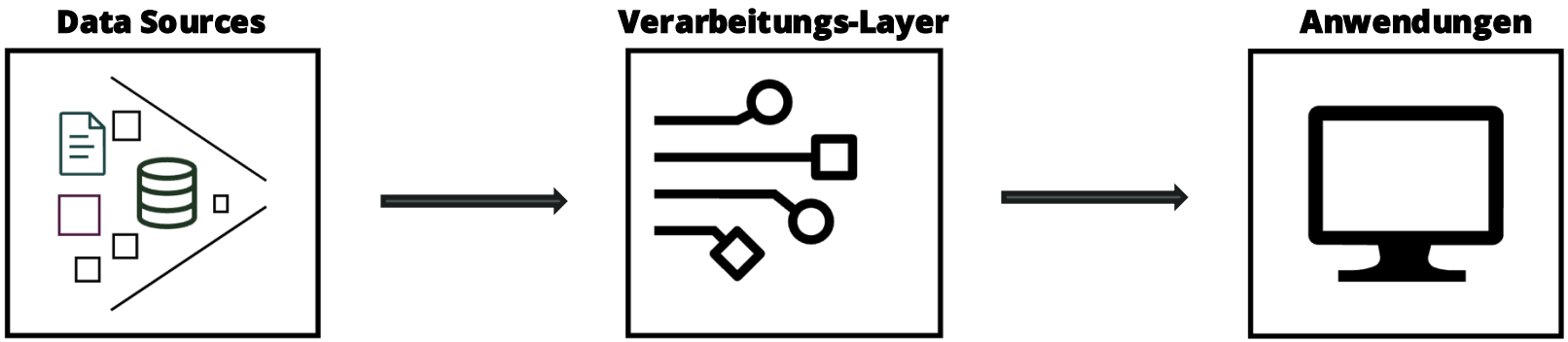
Anwendungsgetriebene Architektur

- Realität: anwendungsgetriebene Architektur
- Kerngeschäft: Zoo voller historisch gewachsenerer Applikationen
- Daten kommen erst an zweiter oder dritter Stelle
- Unternehmen erkenne Mehrwert ihrer Daten
- Datenredundanz durch große, verteilte Datensilos
- Teure Transformations- und Bereinigungs-Prozesse für Neuentwicklungen

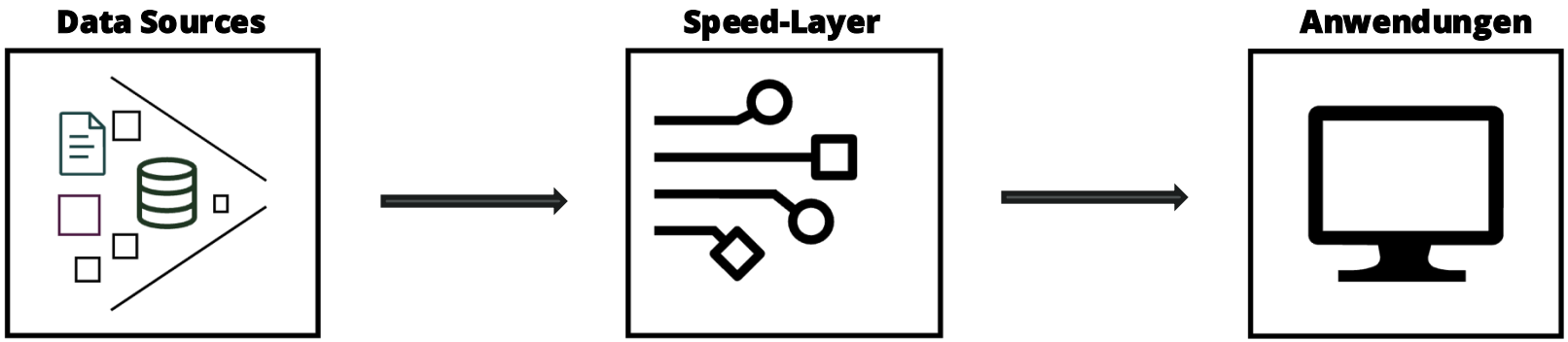
Datenzentrierte Architektur

- weg vom anwendungsorientierten Ansatz des Datenmanagements, hin zum datenorientierten Ansatz
- Daten stellen den Fokus
- Daten werden in Echtzeit in Datenhub geschrieben
- Daten stehen allen Konsumenten zur Verfügung
- Anwendungen werden anhand von Datenanforderungen entworfen

Datenzentrierte Architektur



Kappa-Architektur



Real Time System

A real-time system is one in which the correctness of the computations not only depends on their logical correctness, but also on the time at which the result is produced.

In other words, a late answer is a wrong answer.

Dave Stewart

Real-Time

None Real-Time

- User Interface
- Computer Simulation

Soft/Near Real-Time

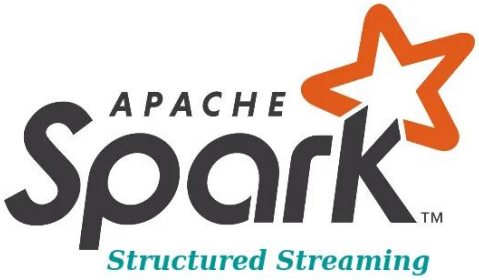
- Messaging-Queue
- Analytics & Reporting
- Aktienhandel
- ETL / Machine Learning

Hard Real-Time

- Motor-Steuerung
- Prozessregler
- Herzschrittmacher

Generationen des Stream Processings

Apache Pig



APACHE STORM

Confluent Cloud

Google Dataflow

Apache Flink

Erste Generation - Batch Processing

Merkmale:

- Daten werden in festen Zeitintervallen oder Stapeln (Batches) verarbeitet
- Ergebnisse werden nach Abschluss der Verarbeitung geliefert

Verwendung:

- Historische Datenanalyse
- Periodische Datenverarbeitung

Zweite Generation – Micro-Batch Processing

Merkmale:

- Daten werden in kleinen Stapeln (Micro-Batches) gesammelt und verarbeitet
- Verzögerung zwischen Dateninput und Ergebnissen wird reduziert

Verwendung:

- Schnelleres Datenstreaming mit geringerer Latenz
- „Echtzeit“-Dashboards

APACHE
STORM™

Apache Flink

Dritte Generation – Event-basierte Echtzeitverarbeitung

Merkmale:

- Daten werden sofort verarbeitet sobald sie eintreffen
- Jedes Event wird individuell verarbeitet
- Echtzeitverarbeitung mit minimaler Latenz

Verwendung:

- Reaktive Systeme
- Echtzeitdatenanalyse
- Kontinuierliche Datenverarbeitung

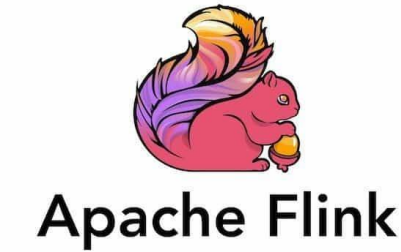
Vierte Generation – Hybrid Streaming & Batch Processing

Merkmale:

- Streaming- und Batchverarbeitung werden in einer Plattform integriert
- Historische- wie Echtzeitdaten können oft mit gleicher Architektur verarbeitet werden

Verwendung:

- Plattformen die ohne Architekturwechsel Batch- und Streaming-Analysen durchführen



Fünfte Generation – Serverless & Auto-Scaling Processing

Merkmale:

- Streaming-Jobs werden serverless und automatisch skalierbar ausgeführt
- Infrastruktur pass sich automatisch an Datenlast an

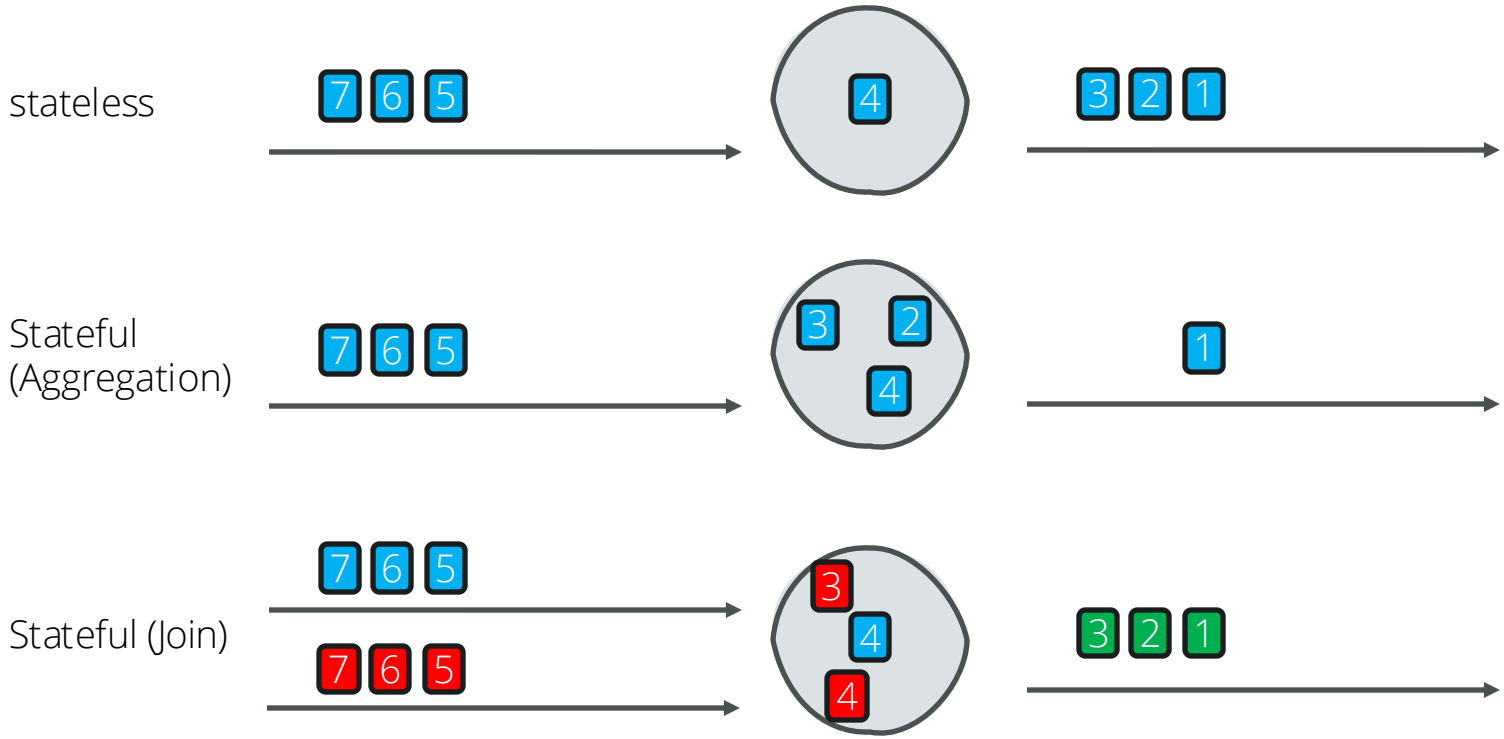
Verwendung:

- Skalierbare, cloud-native Echtzeitverarbeitung ohne Infrastrukturverwaltung

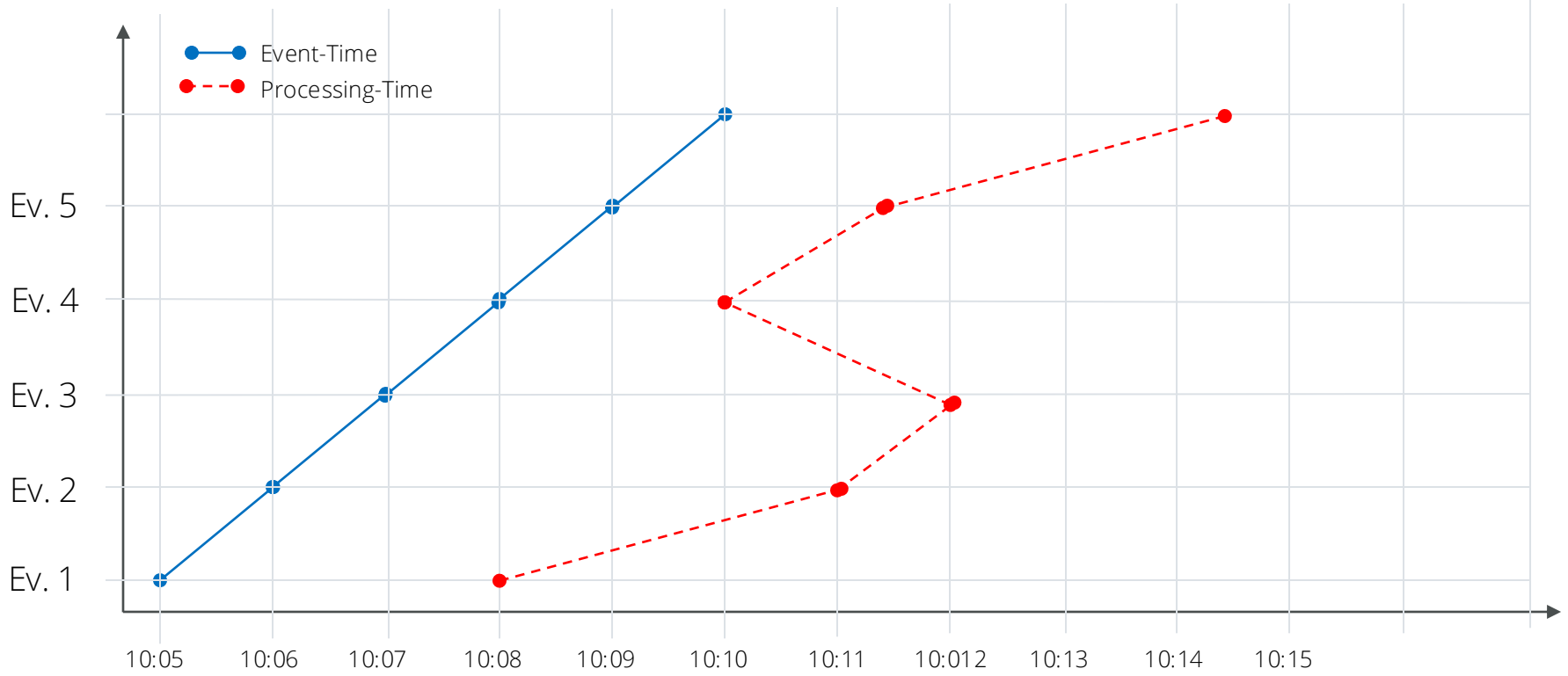
Google Dataflow

Confluent Cloud

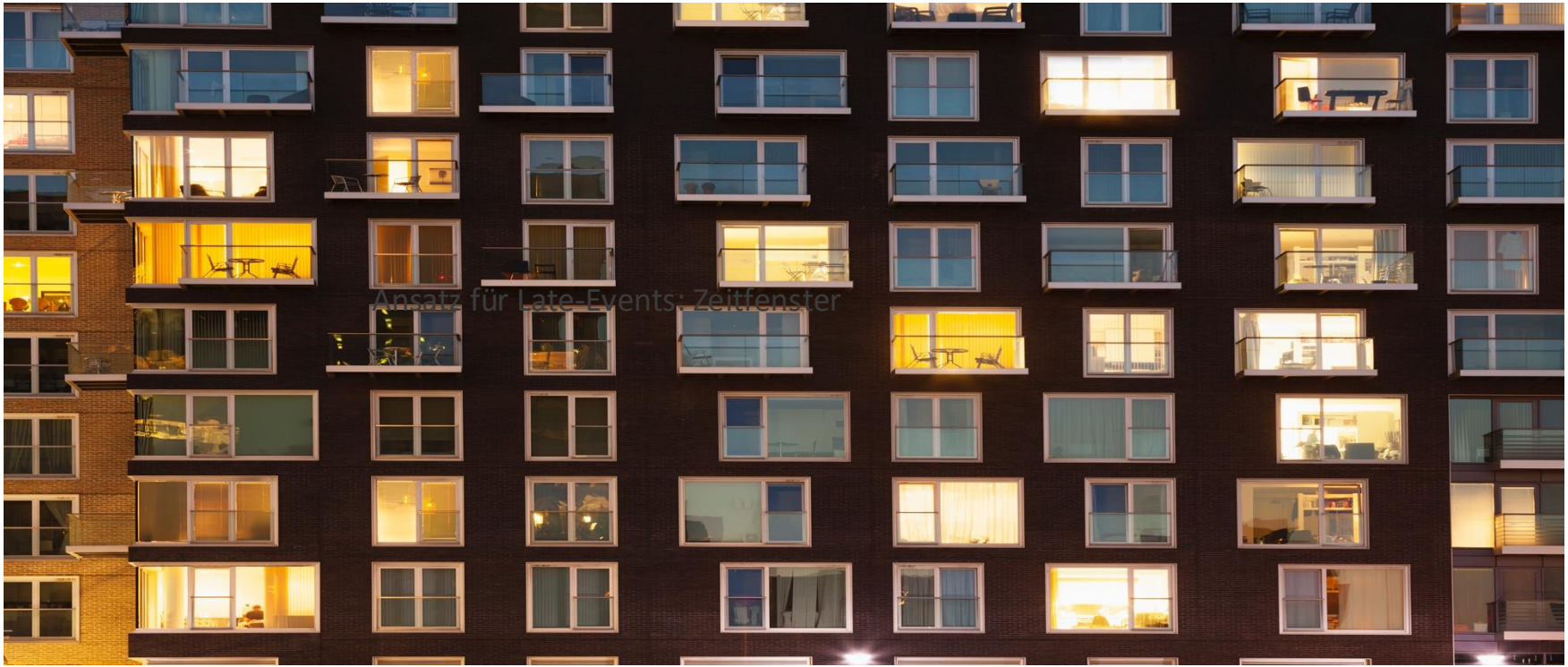
Stateless vs. Stateful Processing



Event Time vs. Processing Time



Zeitfenster



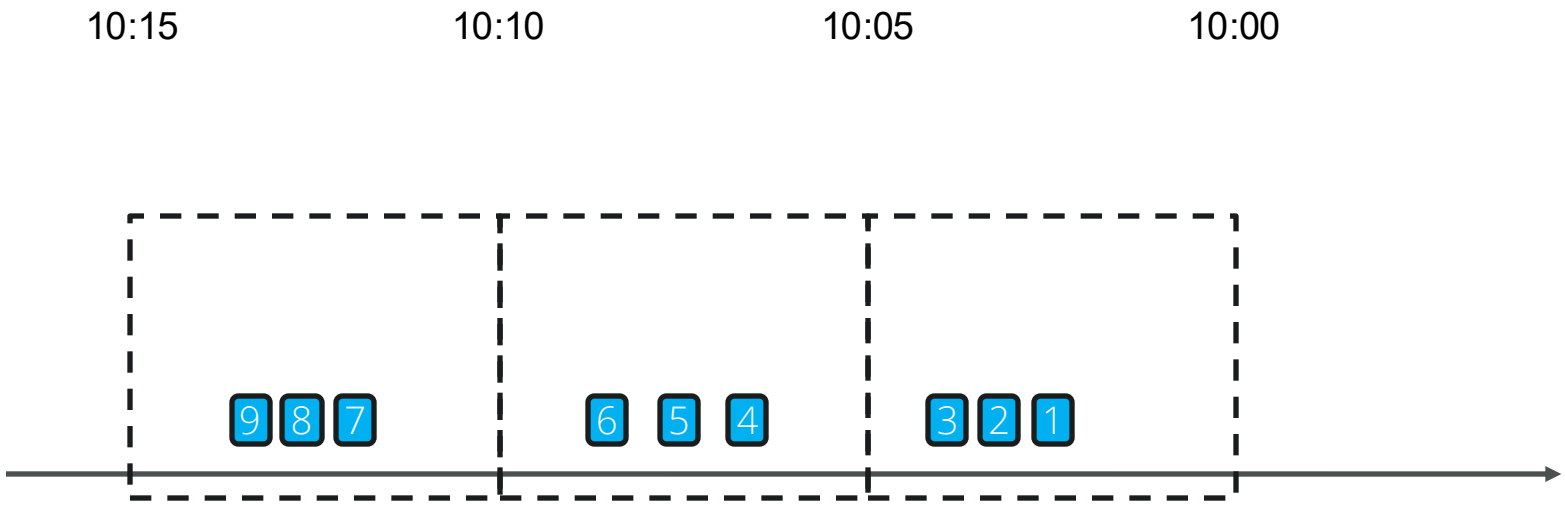
Zeitfenster

Fenster-Typ	Beschreibung
Tumbling-Window	Feste, nicht überlappende Zeitfenster
Hopping-Window	Feste Fenster mit einem konfigurierbaren Sprung (kann sich überlappen)
Sliding-Window	Fenster, die sich dynamisch anhand von Events mit einer Zeitdifferenz bilden
Session-Window	Variable Fenster, die basierend auf einer Inaktivitätslücke enden
Global-Window	Ein einzelnes Fenster für alle eingehenden Events

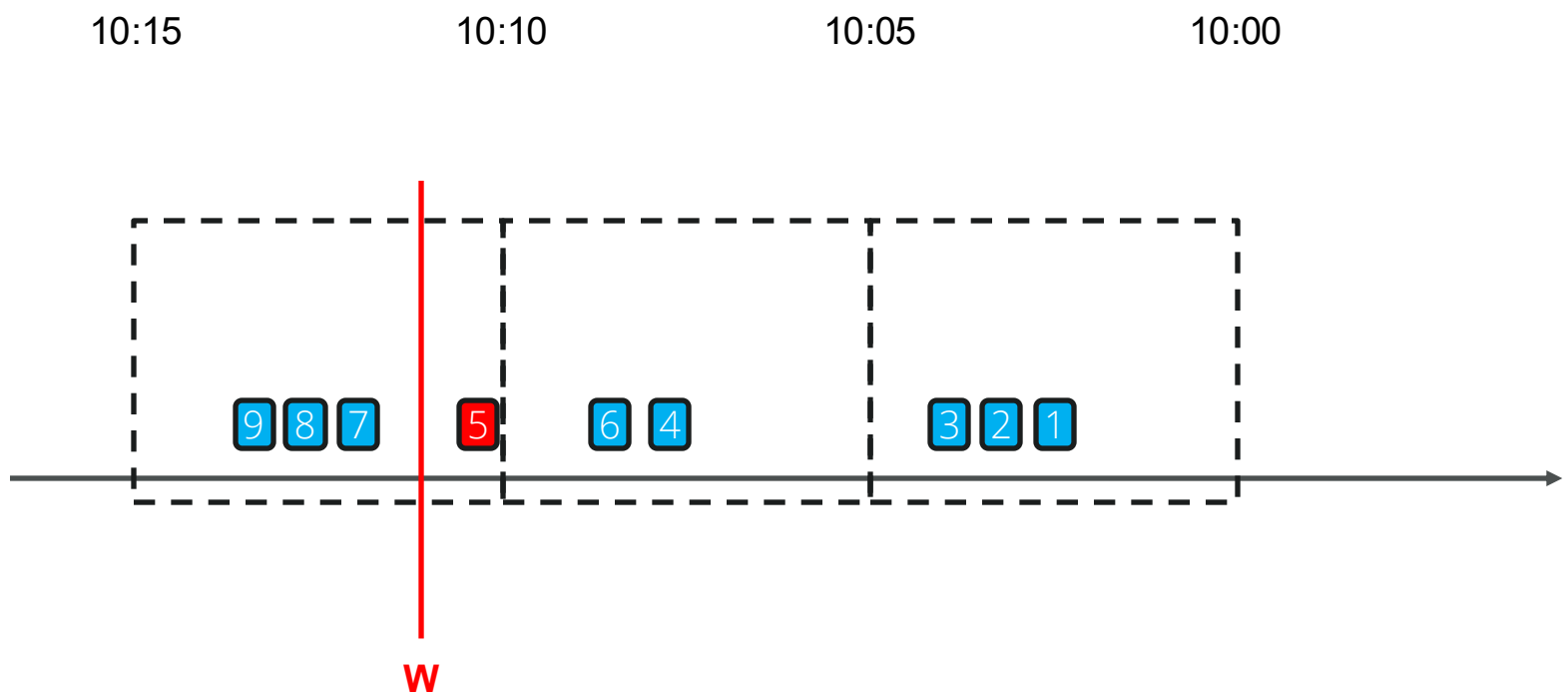
Watermarks

- Watermark = Fortschrittsanzeiger für Event-Zeit
- Bis zu diesem Zeitpunkt sind (fast) alle Events eingetroffen
- Fenster schließen, sobald Watermark ≥ Fenster-Ende
- Ermöglicht korrekte, zeitbasierte Aggregationen trotz Verzögerungen
- Spät eintreffende Daten → Late Events (Verwerfen oder Sonderbehandlung)
- Strategien: z. B. „max Timestamp – erlaubte Verzögerung“

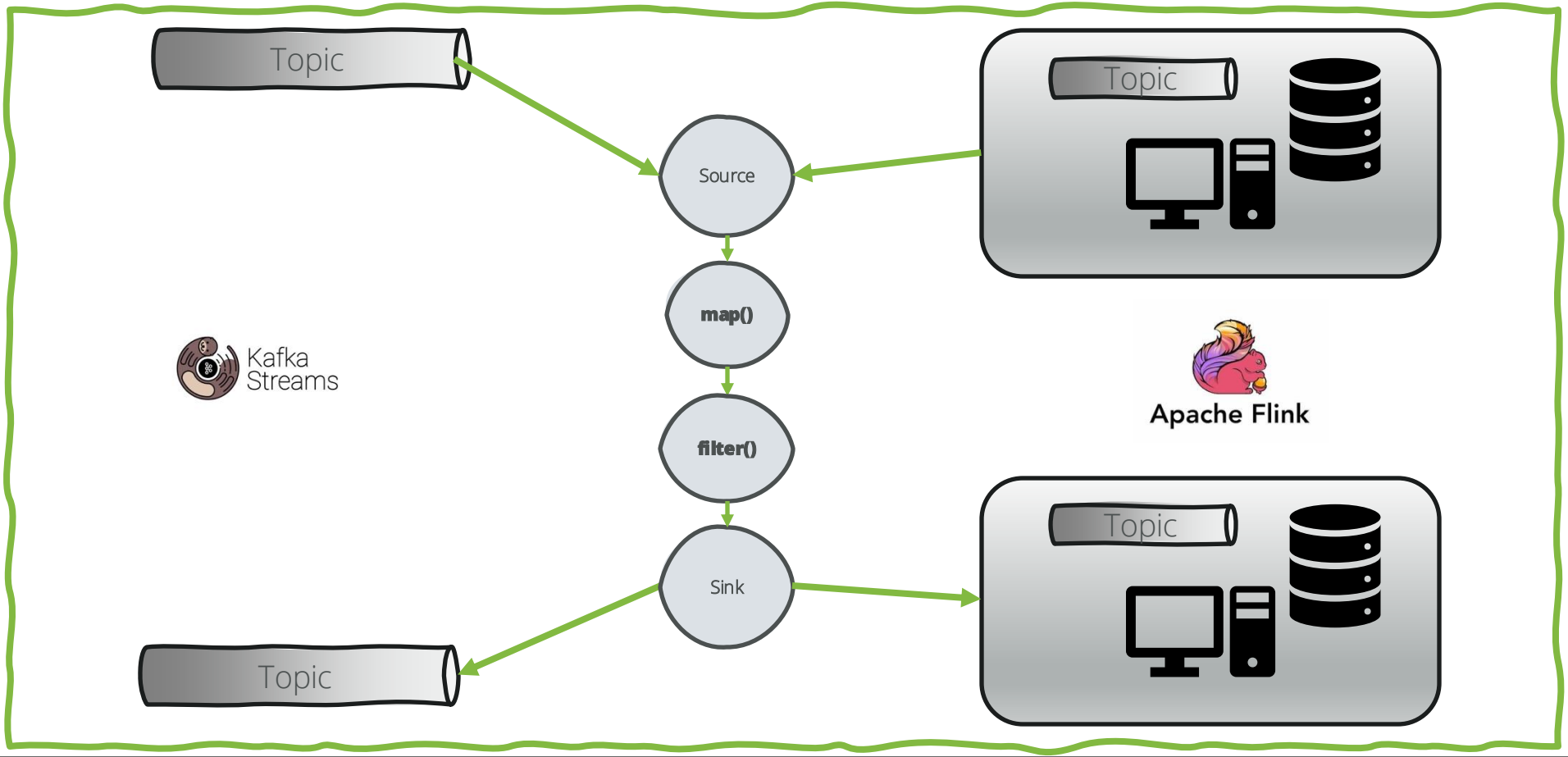
Tumbling-Window & Watermarks



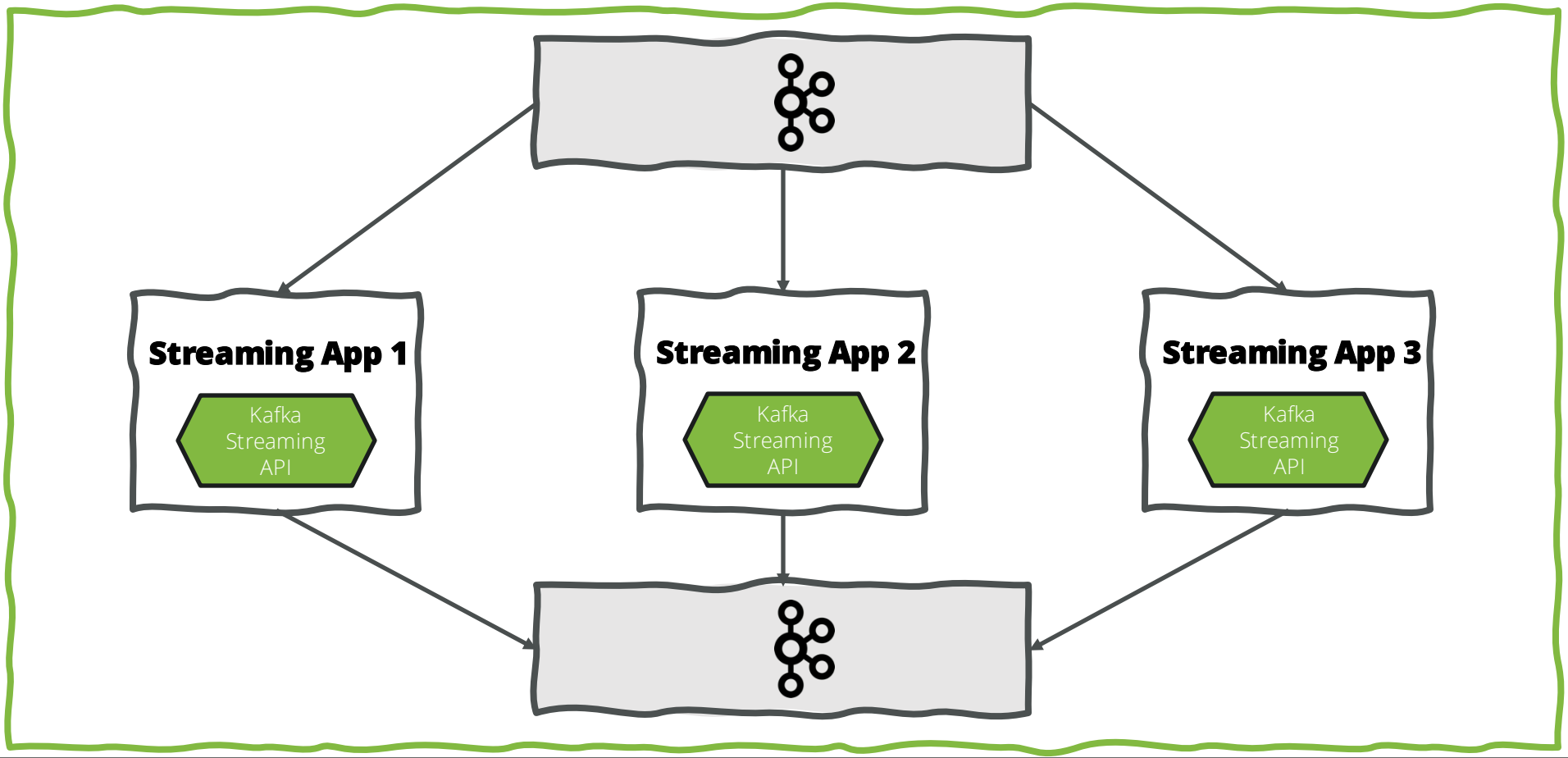
Tumbling-Window & Watermarks



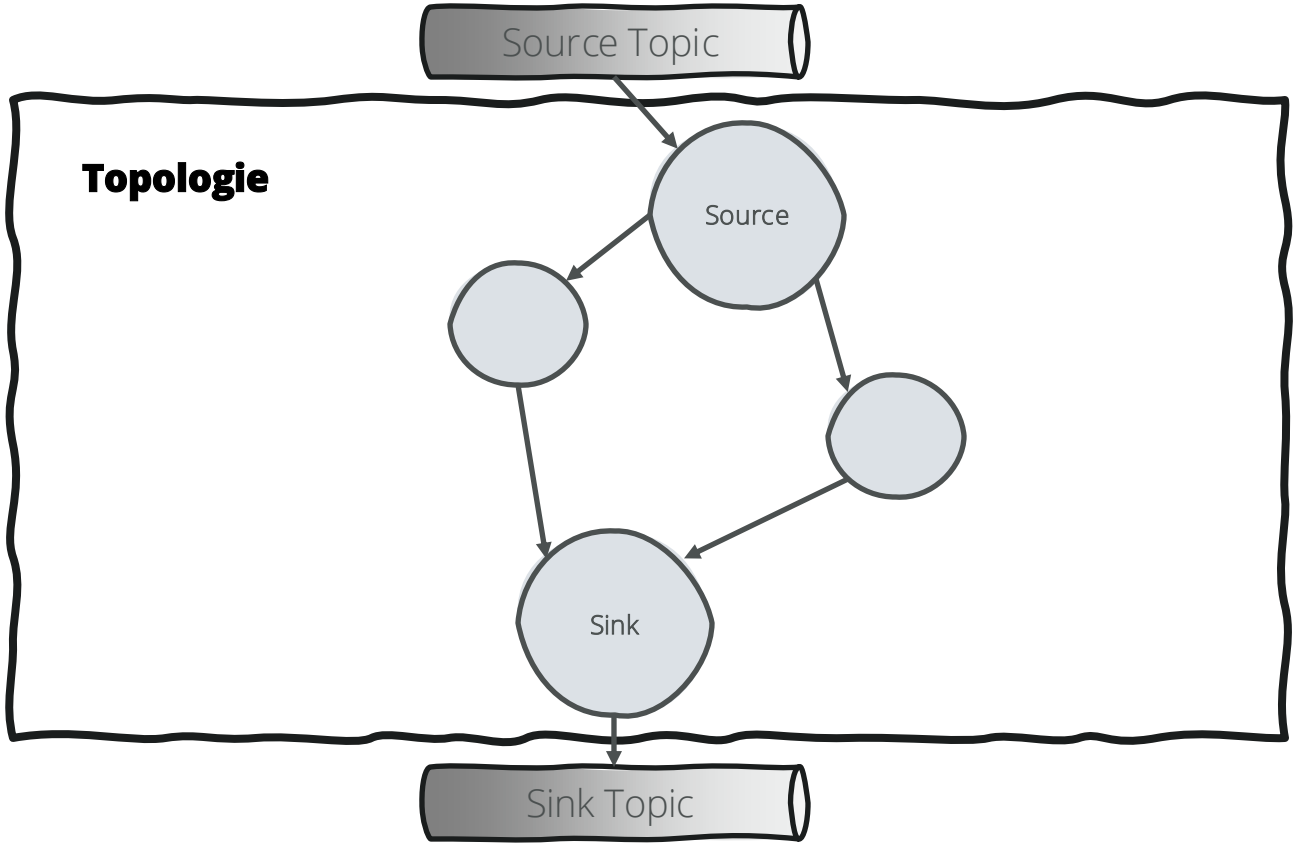
Datastreaming



Kafka Streaming



Topologien



Kafka Streaming API

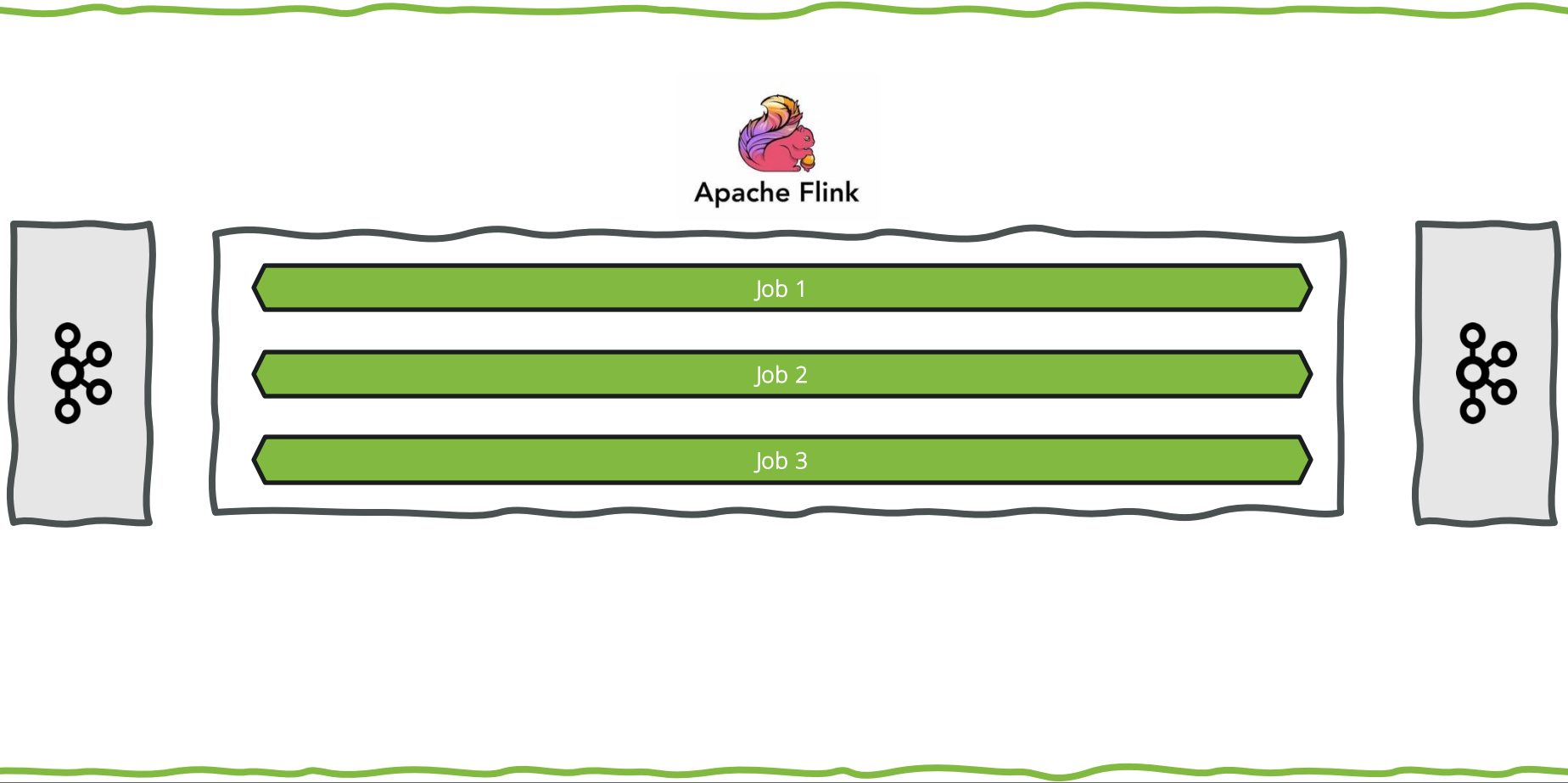
Kafka Stream DSL

- Fluent-API, ähnlich wie Java Stream DSL
- Kapselt die Komplexität der Kafka Processor API
- Mit wenig Code können große Teile von Business-Logik abgebildet werden

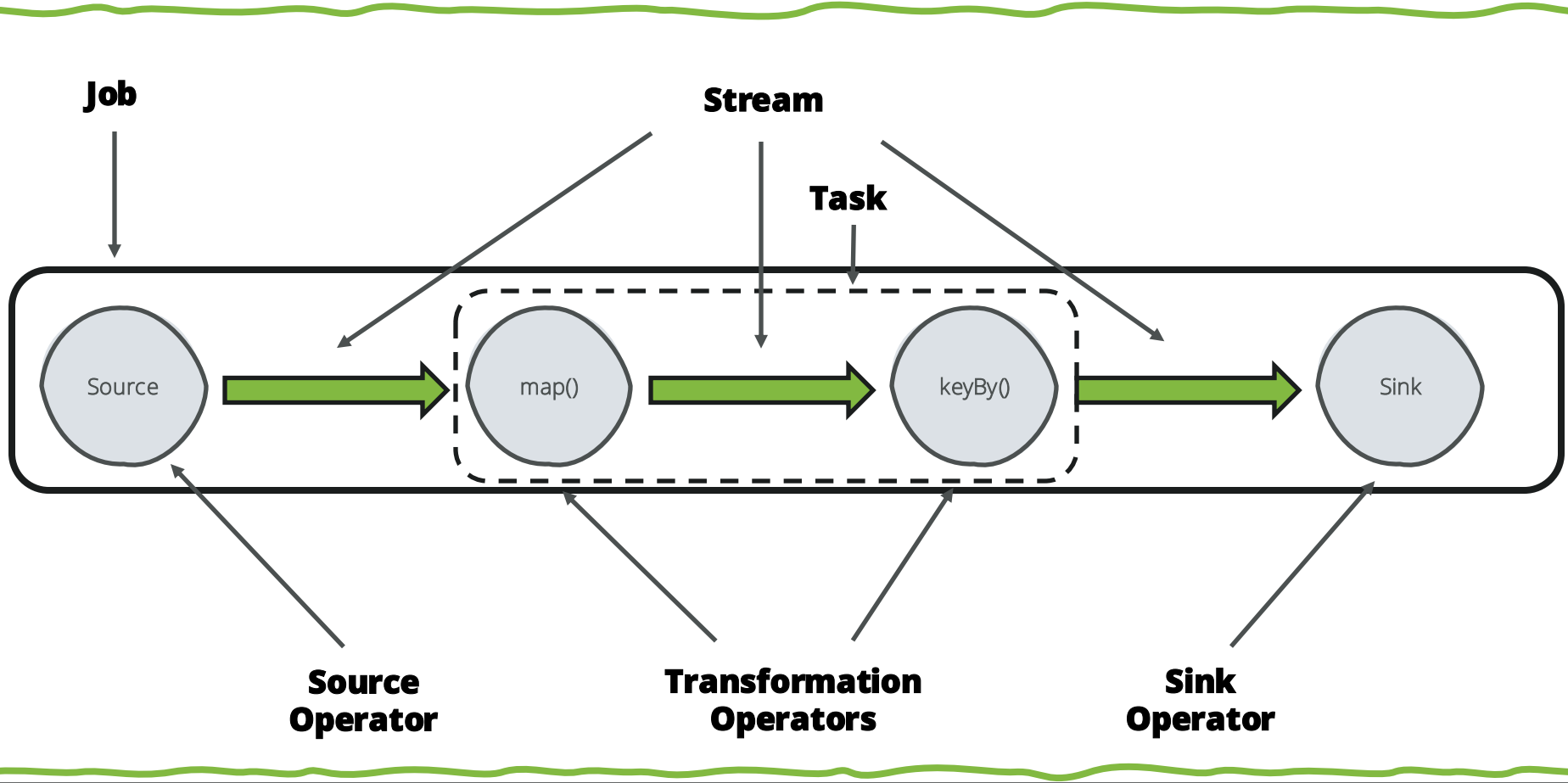
Kafka Processor API

- Aufbau der einzelnen Prozessoren mittels API
- Verdrahtung der Up- und Downstreams
- Zugriff auf Header-Werte der Events
- Errorhandling

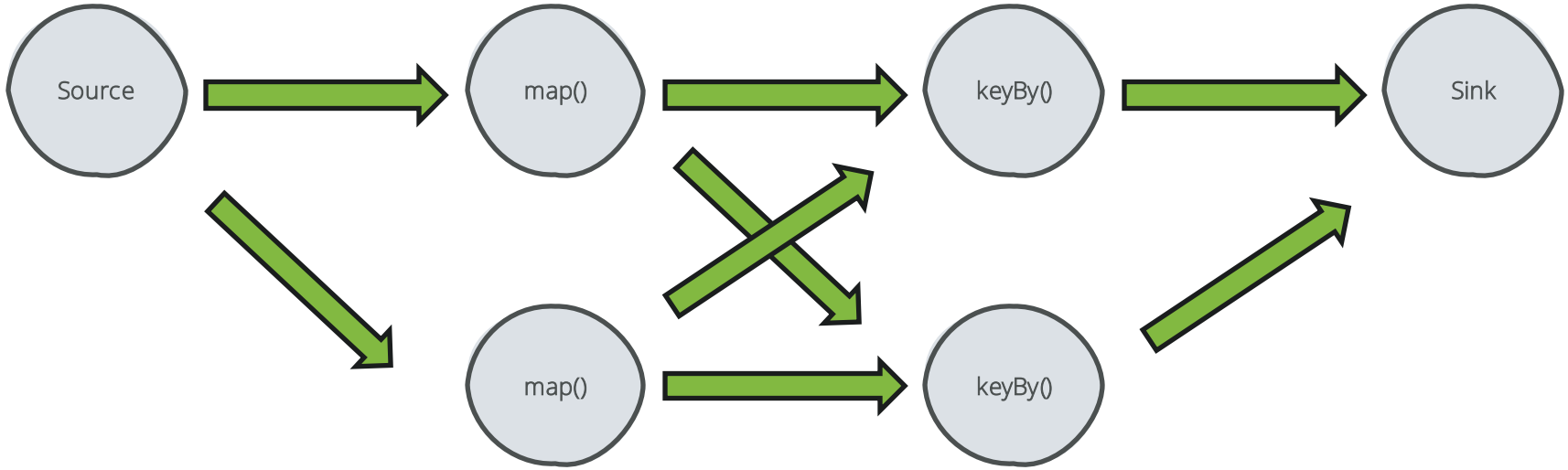
Apache Flink Streaming



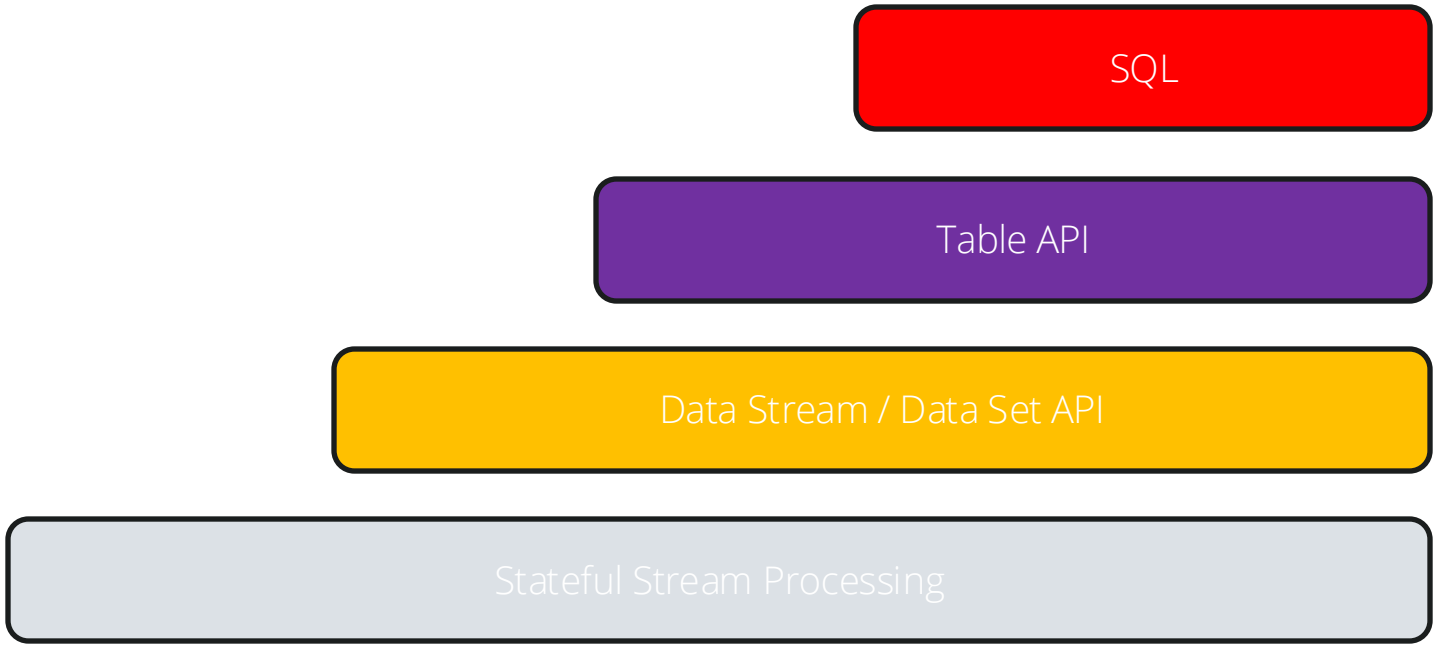
Dataflow Programming



Dataflow Programming



Apache Flink Streaming API



Kafka Streams oder Apache Flink?

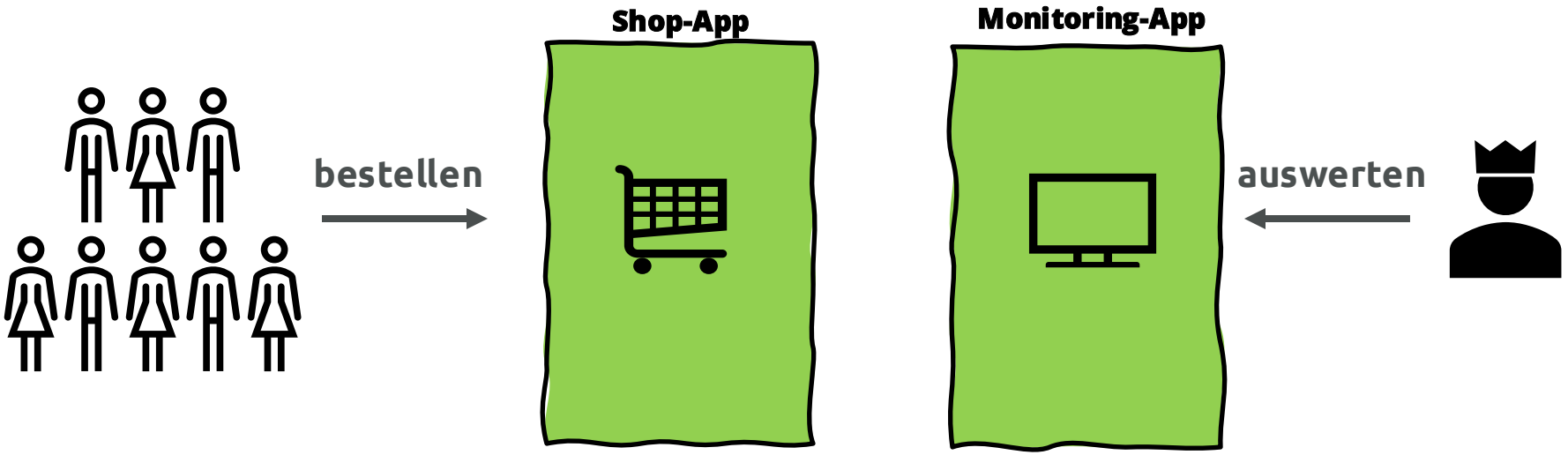
	Kafka Streams	Apache Flink
Infrastruktur	Keine eigene Cluster-Umgebung notwendig	Benötigt eigenes Cluster
Integration mit Kafka	Sehr eng integriert	Unterstützt Kafka, aber auch viele andere Quellen
Skalierbarkeit	Durch Kafka-Partitionierung	Hohe Skalierbarkeit über verteilte Cluster
Stateful Processing	Ja, über Kafka Topics	Ja, mit leistungsfähigem State-Management
Fault-Tolerance	Kafka-Log als Persistenz	Checkpointing und State Snapshots
Komplexität	Einfach zu verwenden	Komplexer, aber leistungsfähiger
Latenz	Niedrig, aber nicht für sub-ms Echtzeit geeignet	Sehr niedrige Latenz, optimiert für Echtzeitanalysen
Batch-Verarbeitung	Nicht nativ	Unterstützt Batch- und Stream-Verarbeitung

Was meint die KI dazu?

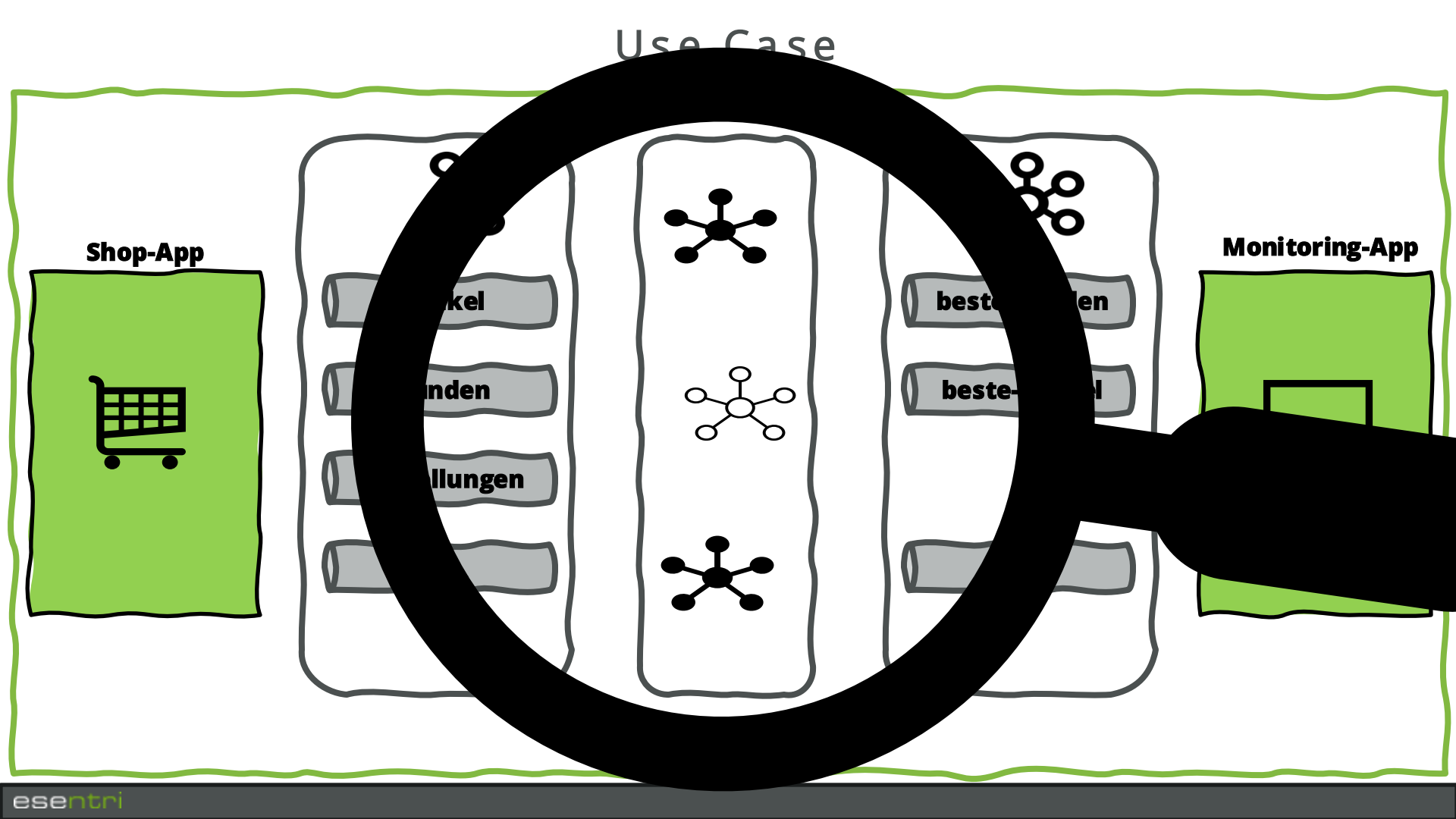
Nutze Kafka Streams, wenn du bereits Kafka verwendest und leichtgewichtiges Stream-Processing in deine bestehenden Anwendungen einbauen willst.

Nutze Apache Flink, wenn du hochkomplexe, verteilte, echtzeitkritische und skalierbare Stream-Processing-Aufgaben mit mehr Flexibilität und Performance bewältigen musst.

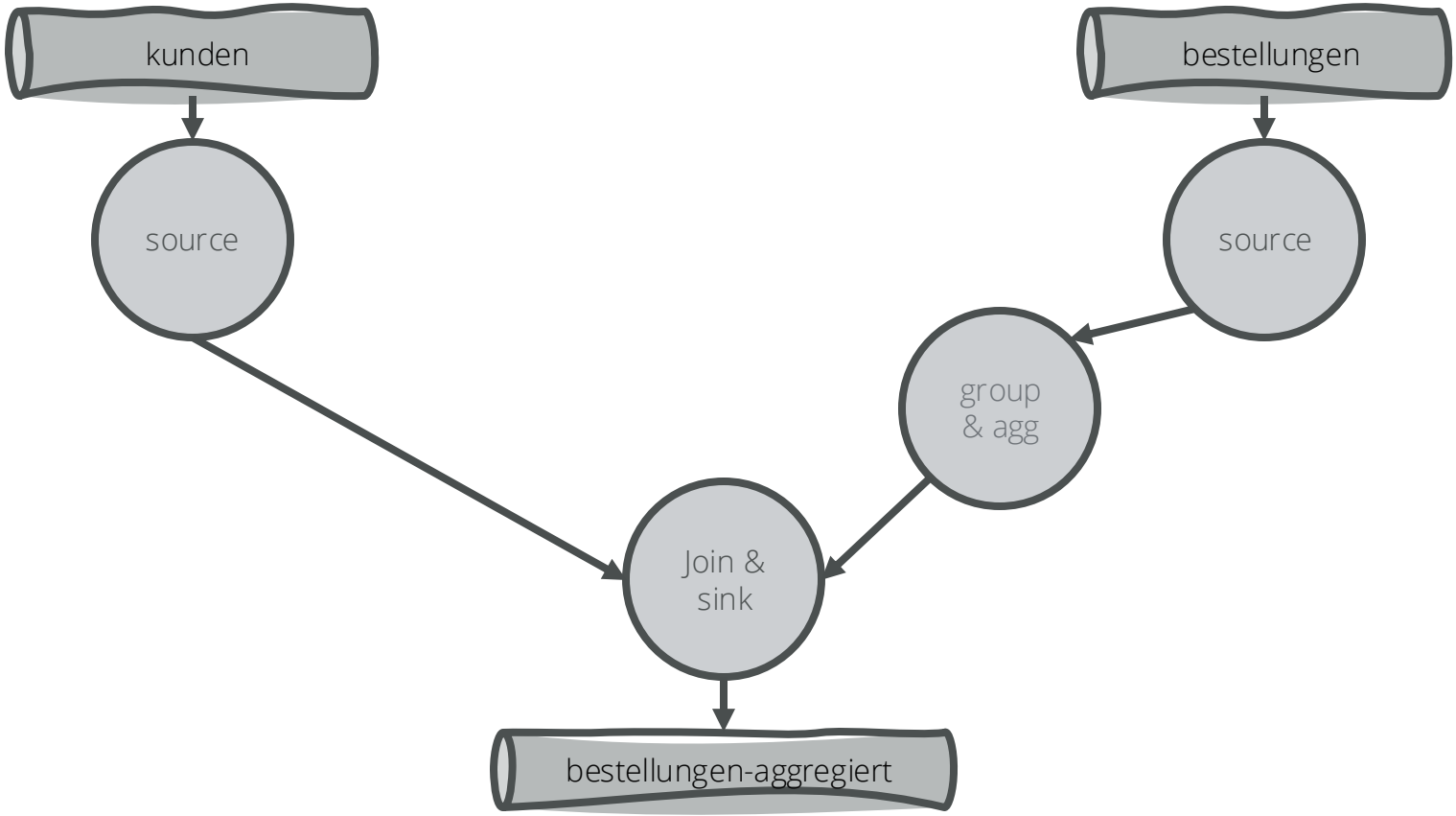
Use Case

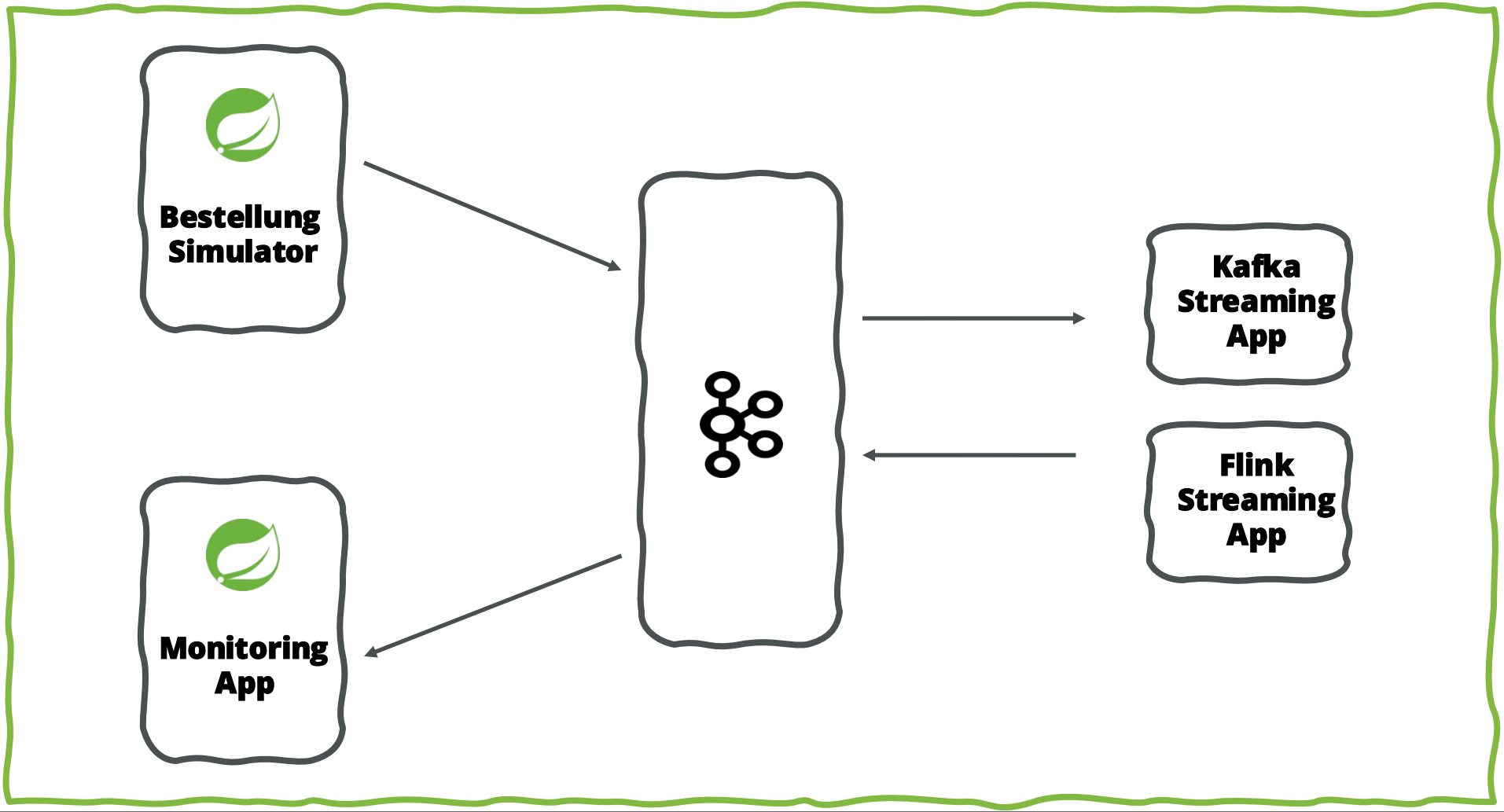


Use Case



Aggregierte Kundenbestellungen

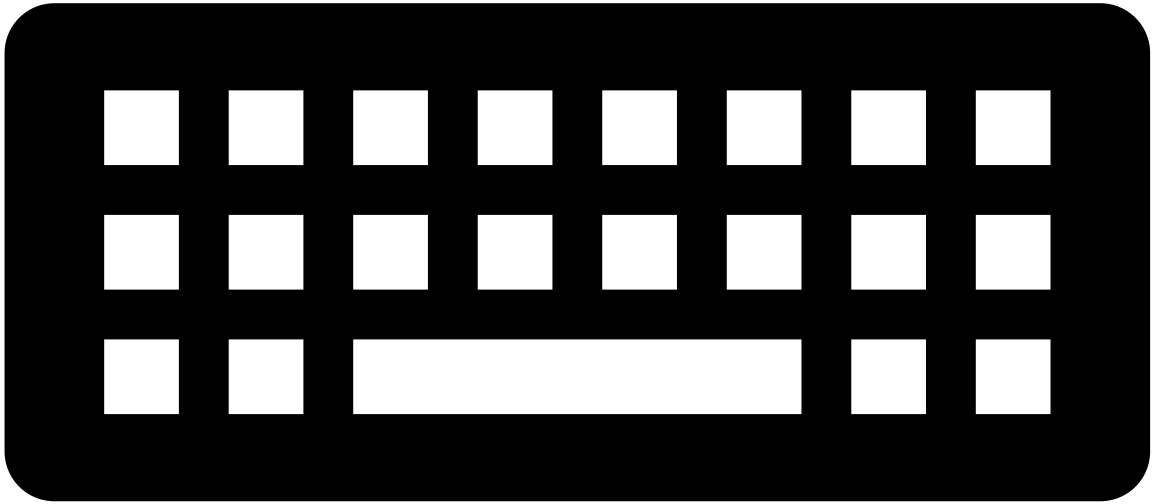



Bestellung Simulator
Monitoring App
Kafka Streaming App
Flink Streaming App

Code

https://gitlab.com/thm-esentri/vortraege/vortrag-flink-kafka

Let's Start Coding



VIELEN DANK!