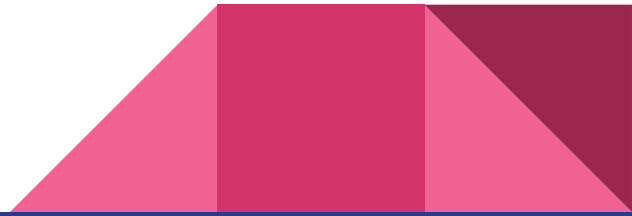
Data Processing Architectures

(Real-time stream processing for Big Data)

Dirk Löwenstrom
Grundseminar WS 2018/19
HAW Hamburg

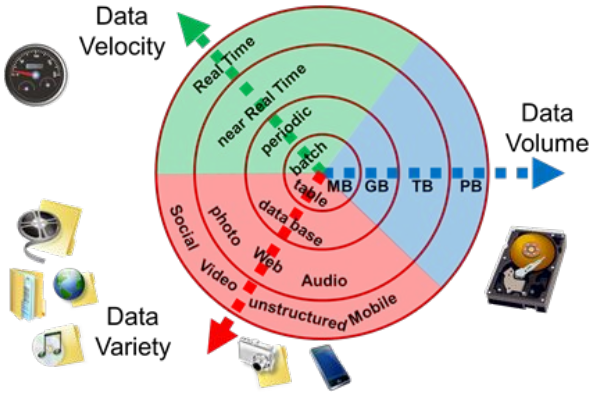
Outline

- Motivation
- Ansätze & State of the Art
  - “Stream Processing”
- Forschung & Konferenzen
- Course of Action



Motivation

Problem Big Data



[Abb. 1]

Volume = massive Datensätze (Terabyte, Petabyte)

Velocity = hohe Datenrate (Events/sek)

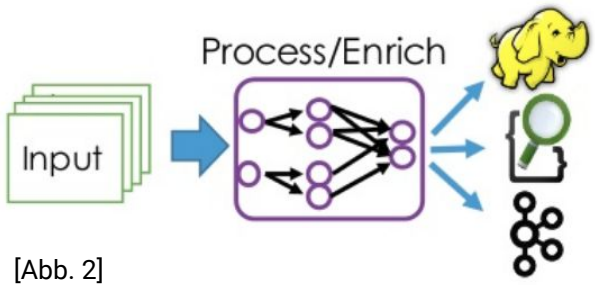
Variety = unstrukturierte Daten (IoT-Sensors, logs, clickstreams)
Bereinigung notwendig

Ziel: schnell Wert aus Daten ziehen & reagieren

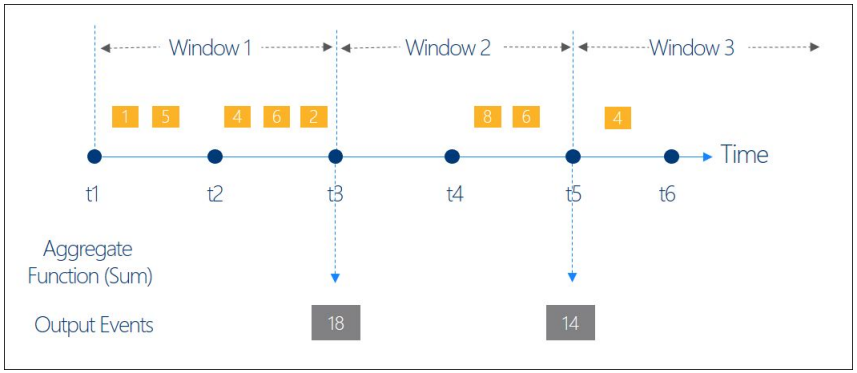
Anwendungen Streaming-System

⇒ ETL-Operationen (Filtern, Analysieren), hoher Durchsatz

⇒ **Data Analytics** → **Real-Time Processing**
⇒ **Window Aggregation** auf **Events** (User Session, Trending Tweets, Pattern Recognition (Fraud))
⇒ **Complex Event Processing**



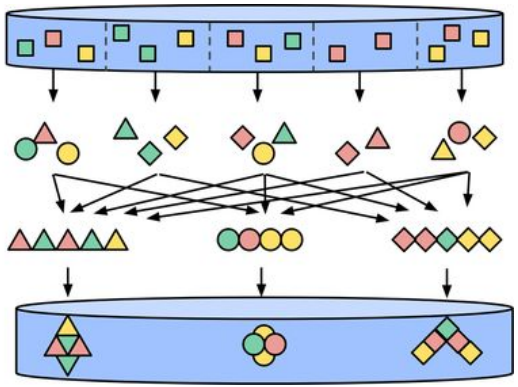
[Abb. 2]



[Abb. 3]

Batch Processing

[Abb. 4]



[1]

Alle Daten in Hadoop (**data lake**)

Skalierbares, fehlertolerantes, verteiltes Dateisystem (HDFS)
Batch-Jobs Ausführung **Minuten bis Stunden**

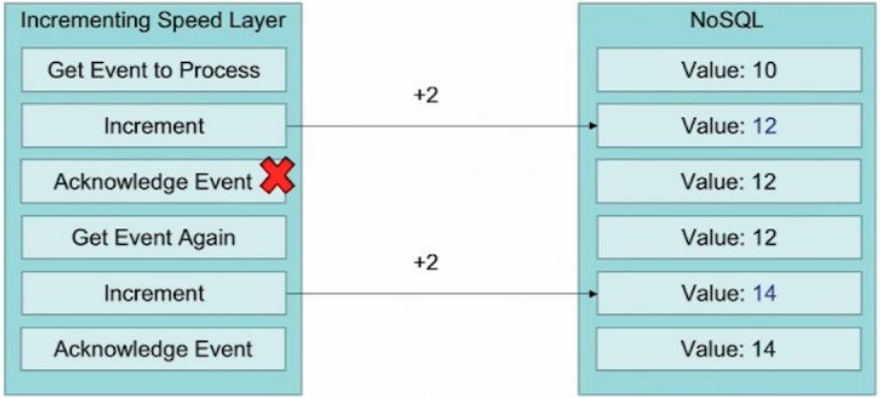
Verarbeitung großer Datenmengen mit parallelen, verteilten Algorithmen in einem Cluster

langsame, aber korrekte Ergebnisse

Stream Processing

Zu beachten:

- kein **Double Counting** (interner State)
- **Exactly-Once Processing** (at-least-once & at-most-once) im State-Handling (nicht User Logik)
- Fehlertoleranz (starke Konsistenz, State)

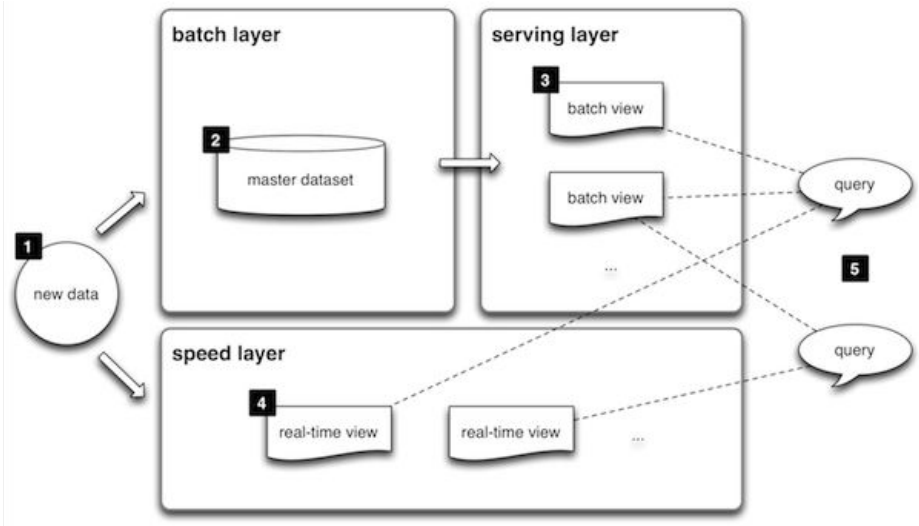


[Abb. 5]

schnelle, aber nicht notwendigerweise korrekte Ergebnisse*

Ergebnisse mit niedriger Latenz & (eventuelle) Korrektheit?

Lambda Architecture



[Abb. 6]

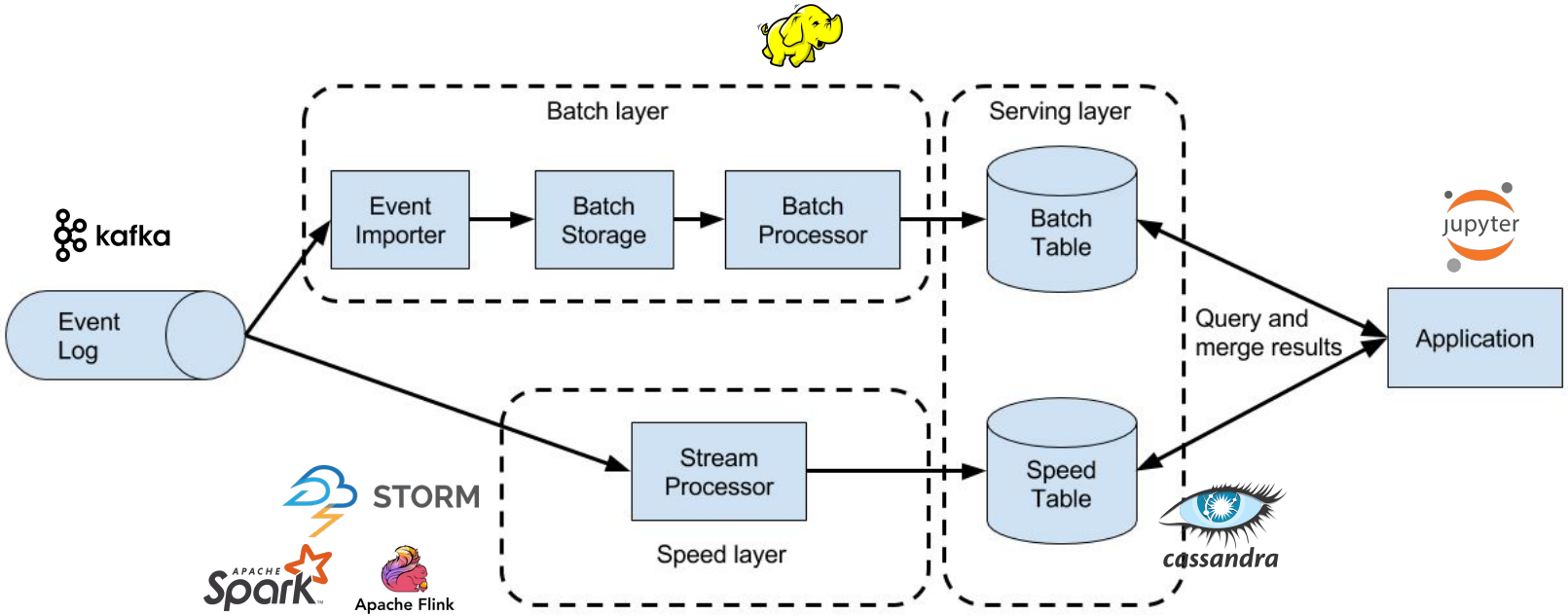
Batch-Ebene
- speichert Rohdaten
- immutable, append-only
- berechnet periodisch **batch-views**

Speed-Ebene
- real-time Datenanalyse
- geringe Latenz → kleinere Genauigkeit
  → nur aktuelle Daten

Serving-Ebene
- indiziert batch-views für schnellen Zugriff
- Query → Kombination beider Ebenen

[3]

Lambda Architecture



langsame Batch-Verarbeitung + Echtzeitverarbeitung (**Velocity, Volume**)

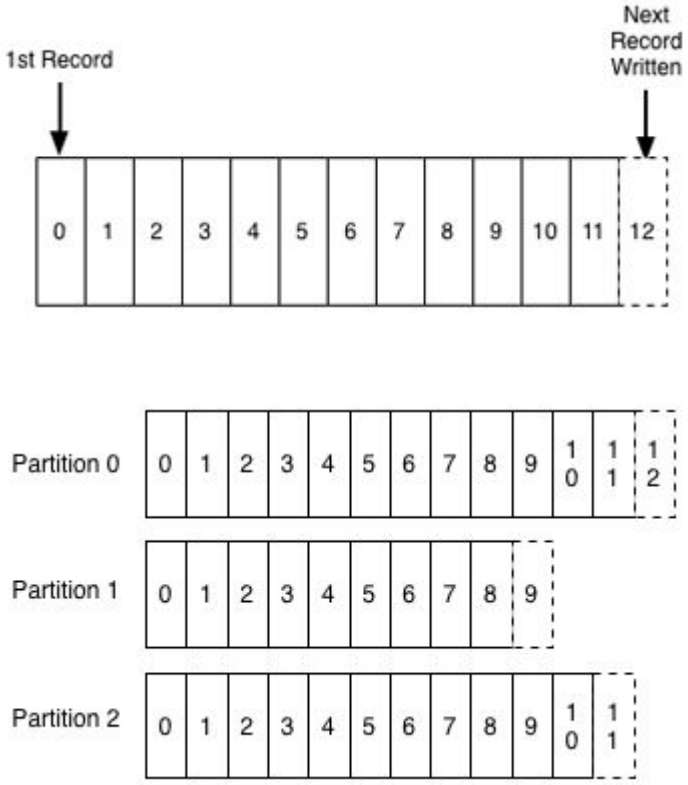
[Abb. 7]

Kafka

- Grundlage für massives Stream Processing
- geordnete, partitionierte, fehlertolerante Streams (keine verlorenen Events) → ***replayable***

- Scale-Out, Buffering

Wiedereinspielen von Events ist Grundlage für Korrektheit von Streaming Systemen



[Abb. 8]

Lambda Architecture

Pro:

- skalierbar
- Fehlertolerant (HDFS)
- Echtzeitanalysen (Genauigkeit)

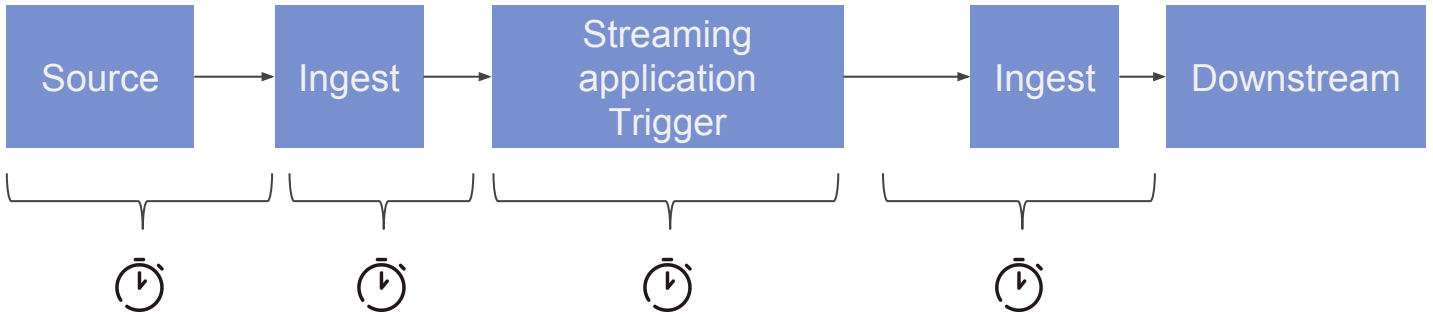
Con:

- Komplexität
- zwei Systeme/Versionen der Datenpipeline
- Duplizierte Logik
- Merge für Query

Stream Processing

Ein kurzer Gedanke zur Latenz

Ende-zu-Ende Latenz (Netzwerk, Streaming-Engine, Downstream Databases/System)
⇒ Latenz Eintritt->Verarbeitung



Streaming Landscape (Open-Source)

und noch viele mehr...

Vergleich

	STORM	Spark	samza	
Streaming model	Native	Micro-batching	Native	Native
API	Compositional	Declarative	Compositional	Declarative
Fault tolerance	Record ACKs	RDD-based	Log-based	Checkpoints
Guarantee	At-least-once	Exactly-once	At-least-once	Exactly-once
State	Only in Trident	State as DStream	Stateful operators	Stateful operators
Windowing	Not built-in	Time based	Not built-in	Policy based
Latency	Very-Low	Medium	Low	Low
Throughput	Medium	High	High	High

[Abb. 9]

	Storm	**Trident**	**Samza**	**Spark Streaming**
strictest guarantee	at-least-once	exactly-once	at-least-once	exactly-once
achievable latency	$\ll 100$ ms	< 100 ms	< 100 ms	< 1 s
state management	yes	yes (small state)	yes	yes
processing model	one-at-a-time	micro-batch	one-at-a-time	micro-batch
backpressure mechanism	yes	yes	not required (buffering)	yes
ordering guarantees	no	between batches	within stream partitions	between batches
elasticity	yes	yes	no	yes

[4]

Grundlagen

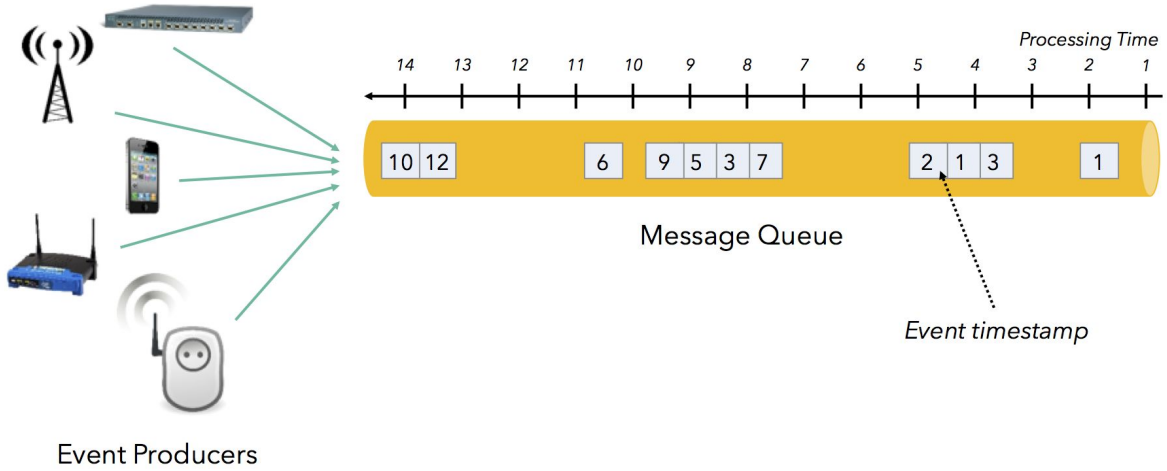
State of the art: Inspiration

The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak,
Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills,
Frances Perry, Eric Schmidt, Sam Whittle
Google

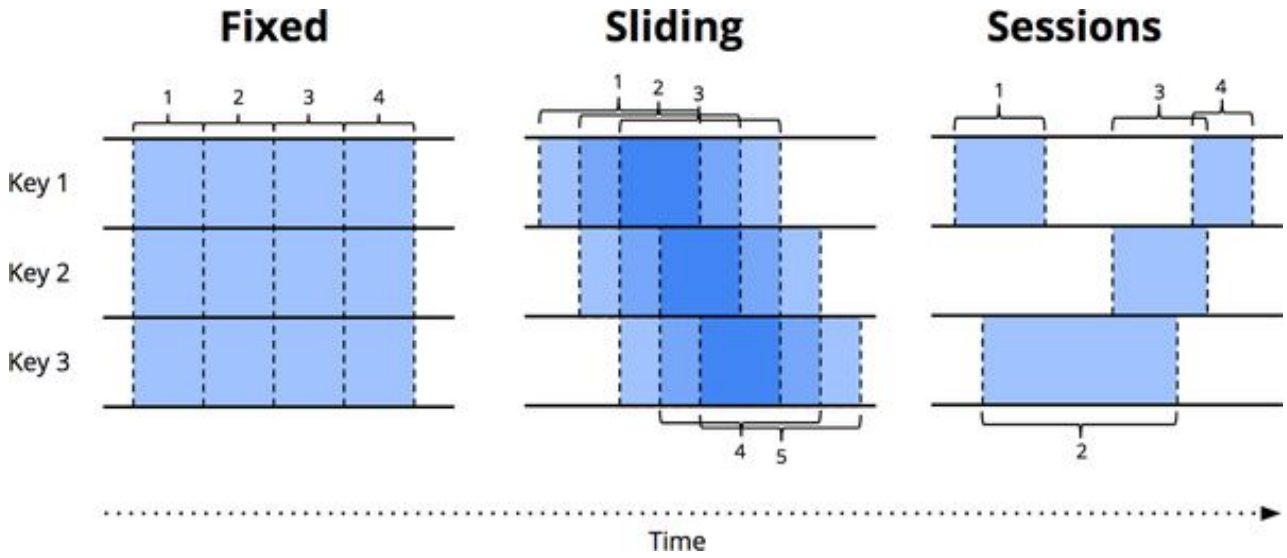
[5]

Event time vs. Processing Time



- Event-Time ist unabhängig von der Processing Time
- Streaming-Engine → einen Weg um den **Fortschritt der Event-Time** zu messen
  ⇒ benötigt für **Window Operatoren**, die auf Event-Time operieren
  ⇒ verhindern von Events im falschen Window

Windowing



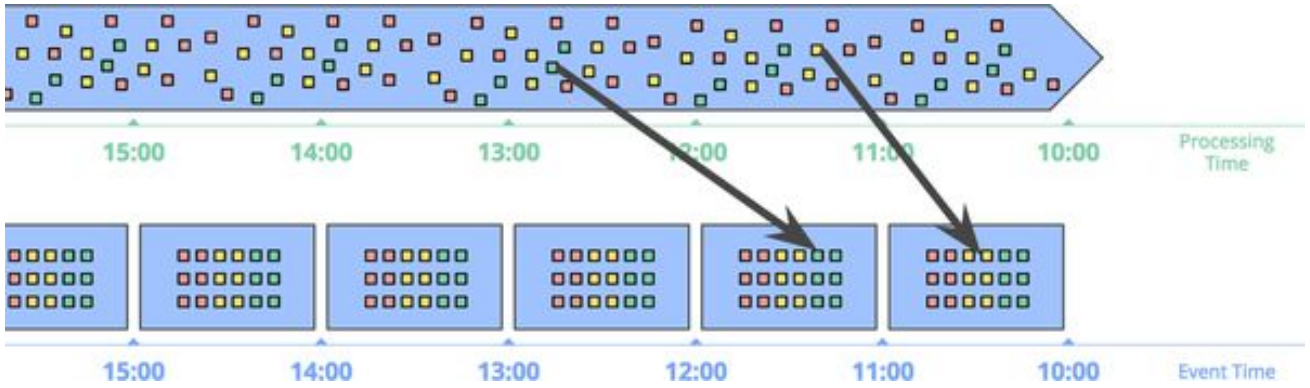
[2]

- Datenquelle wird entlang zeitlicher Grenze in einzelne Teile zerlegt

Windowing by even-time

Quellen sind nicht immer verfügbar (Latenz, Netzwerkpartition, Mobile Devices)
⇒ korrektes Bearbeiten “später” Events (richtige Reihenfolge)

(fixed window)

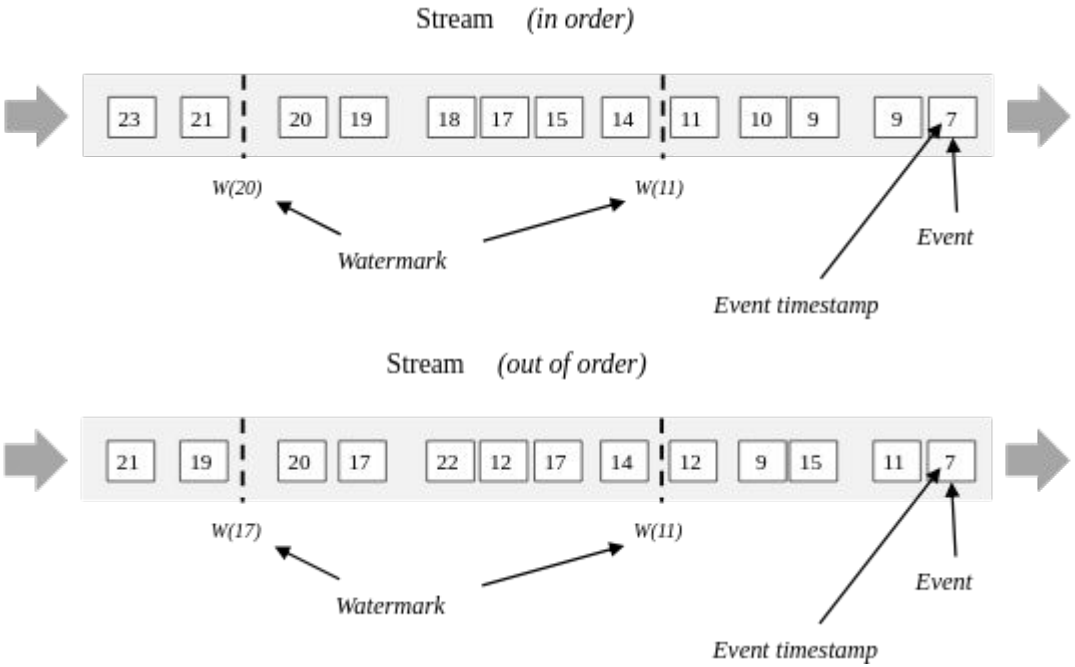


[2]

⇒ Grundidee: erhöhte **Window** Lebenszeit → längeres Buffering, State Handling
⇒ Vollständigkeitsfrage → fixe/heuristische **Watermarks** (keine 100% Garantie)

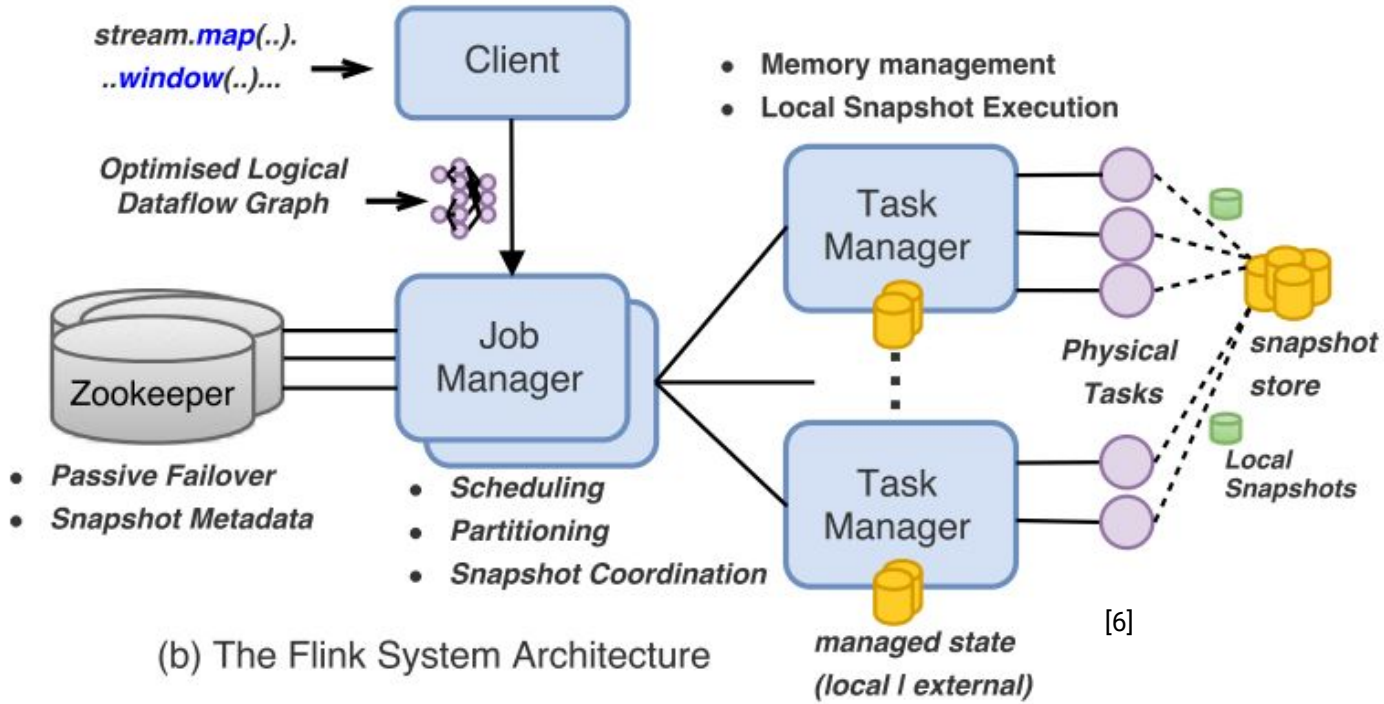
Watermarks (in-order vs out-of-order)

- Watermark nimmt an, dass alle früheren Ereignisse (wahrscheinlich) eingetroffen sind
- “späte Events ins richtige Window kriegen”



[Abb. 11]

State of the art: Apache Flink



(b) The Flink System Architecture

[6]

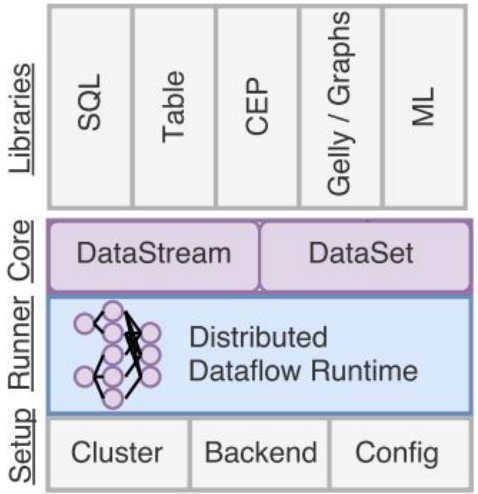
State of the art: Apache Flink

Kernkonzept:
DataStream (unbegrenzt) & **DataSet** (begrenzt) - Optimierer

- **Exactly-Once** (periodischer lokaler Checkpoint-State in dauerhaften Speicher) → fehlertolerant

- **State-Handling** Wiederherstellung State/Operator Fortschritt
- **(Event-Time) Windowing, out-of-order (watermarking)**

- Modi:
  - natives Event-Processing (geringe Latenz)
  - **Stateful Stream Processing**
    (starke Konsistenz **Savepoints & Snapshots**)
    ⇒ zu Kosten der Performanz

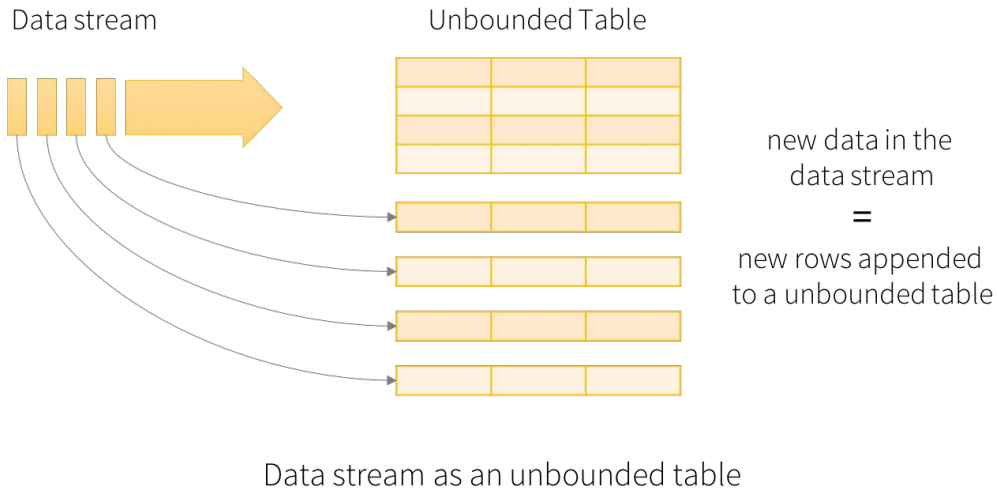


[6]

(a) The Flink Software Stack

StreamingSQL
CEP (Mustererkennung)

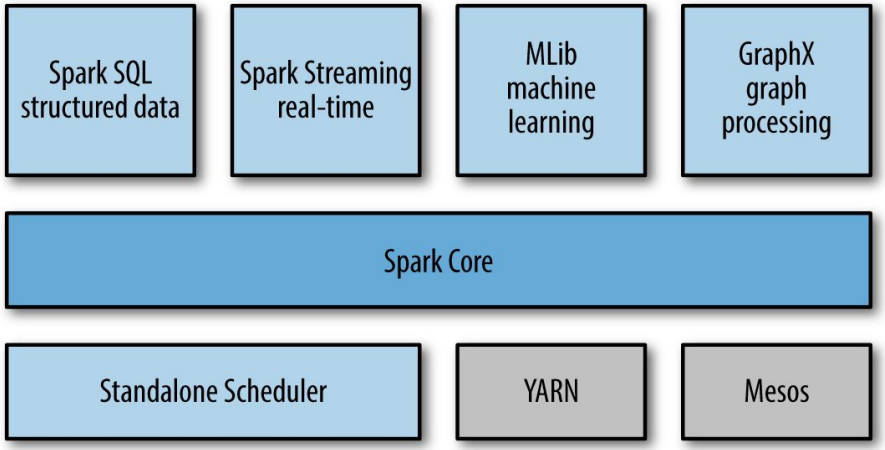
Streams & Tables



[Abb. 12]

- Live-Datastream ⇒ unbegrenzte Tabelle an die kontinuierlich angefügt wird
- Inkrementelle SQL-Queries
- **Trigger**, **Evictor**

State of the art: Apache Spark

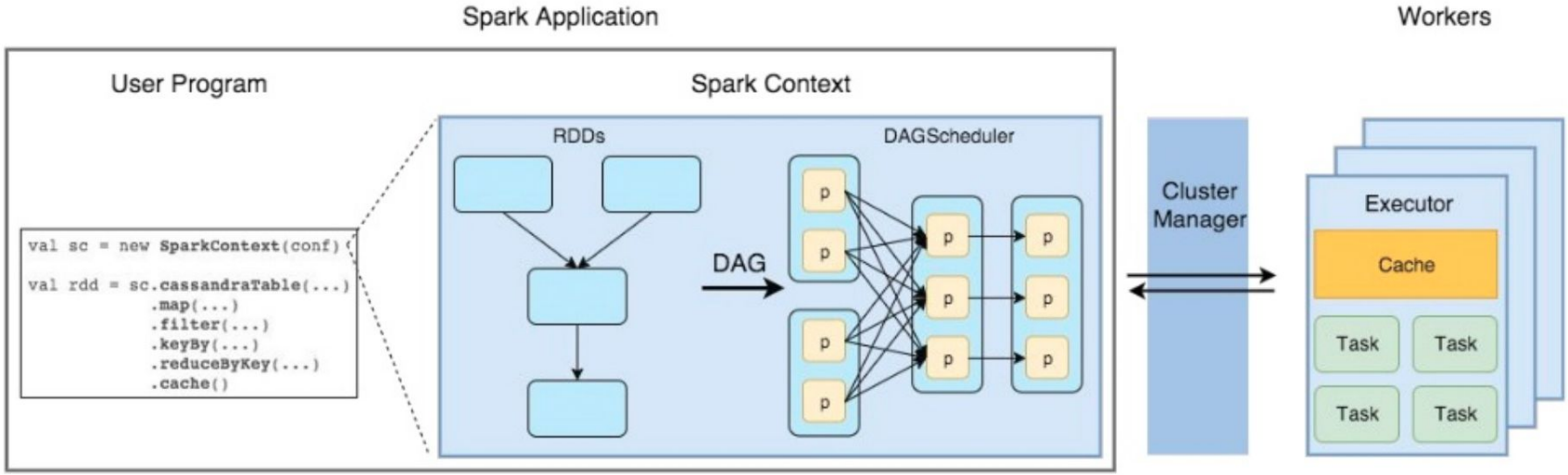


[Abb. 13]

⇒ starke Konsistenz durch Ausführung von **Batch-Jobs in Memory**

⇒ mehrere Batch-Jobs hintereinander in kurzen Intervallen*? ⇒ **Micro-Batching**

State of the art: Apache Spark

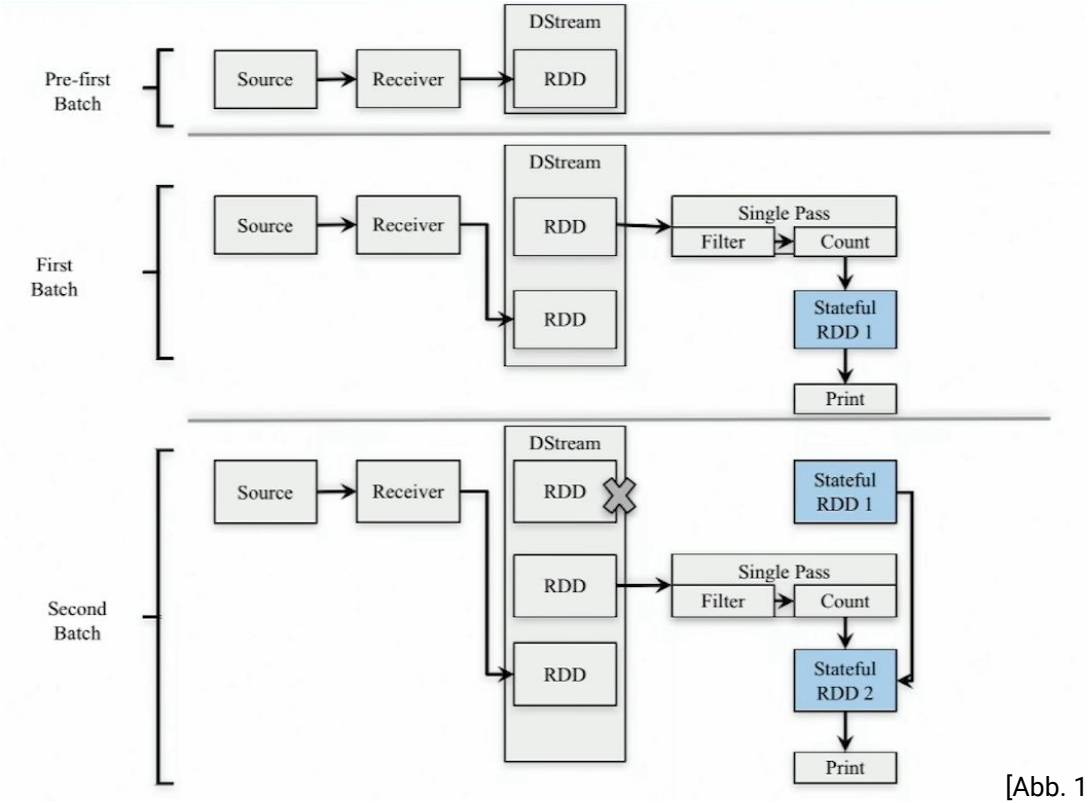


[Abb. 14]

RDD = fehlertolerante, immutable, parallele Datenstruktur
(Optimierung: hält Daten im Speicher für Wiederverwendung nachfolgender Stufen)
DStream = unbegrenzter Stream von Daten, repräsentiert Reihe von RDDs

State of the art: Apache Spark

- replayability durch Kafka
- stateful RDD
- Verarbeitungs-garantien
- immernoch Micro-Batch



[Abb. 15]

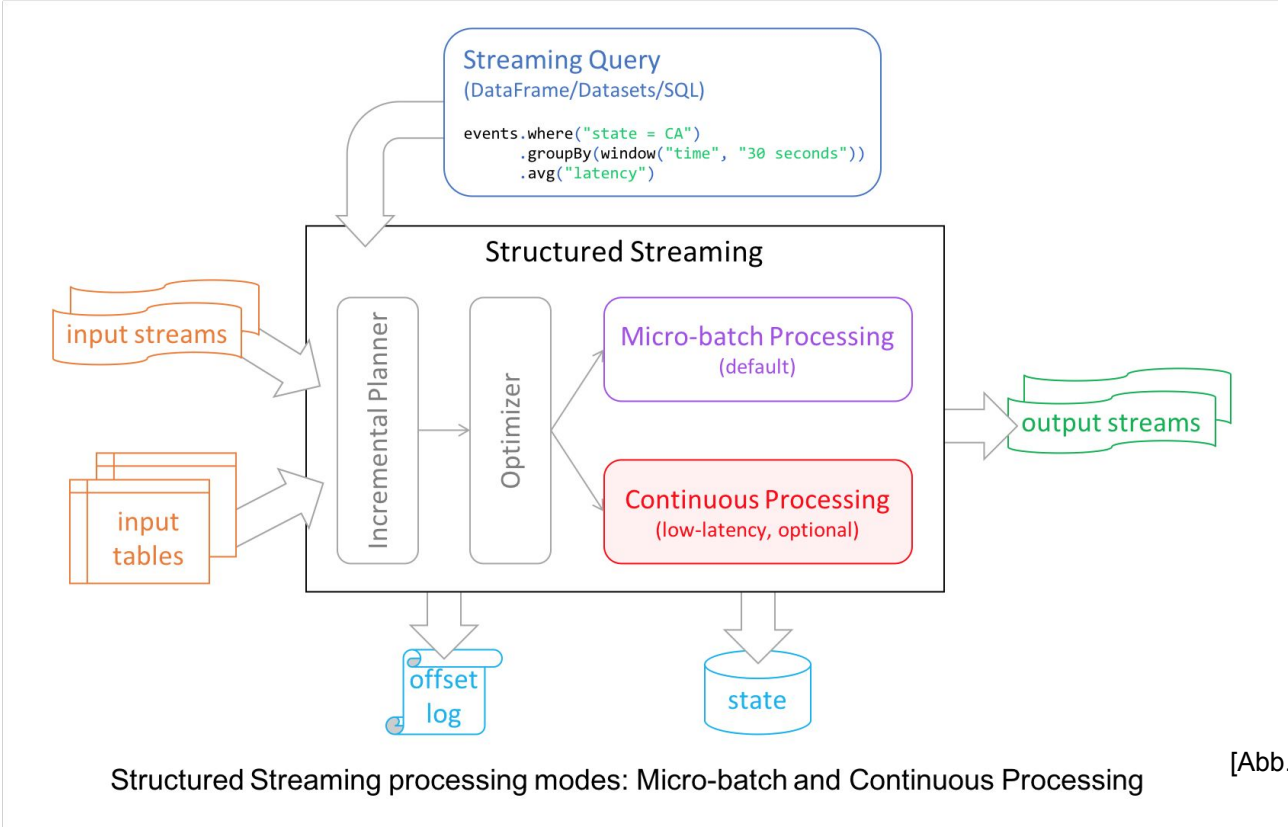
State of the art: Apache Spark 2.x

Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark

Michael Armbrust[†], Tathagata Das[†], Joseph Torres[†], Burak Yavuz[†], Shixiong Zhu[†], Reynold Xin[†], Ali Ghodsi[†], Ion Stoica[†], Matei Zaharia[†‡]
[†]Databricks Inc., [‡]Stanford University

- höhere API → Streaming & Batching
- basiert auf Spark SQL-Engine
- Streaming Aggregation, Event-Time Windows, Stream-to-Batch Joins,..
- End-to-End Exactly-Once Processing
- Fehlertoleranz, Garantien durch Checkpointing & Write-Ahead Logs
- **Continuous Processing** → bessere Performance (geringere Latenz)

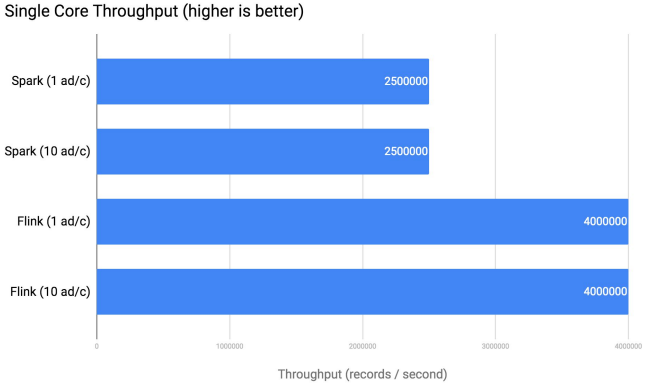
State of the art: Apache Spark 2.x

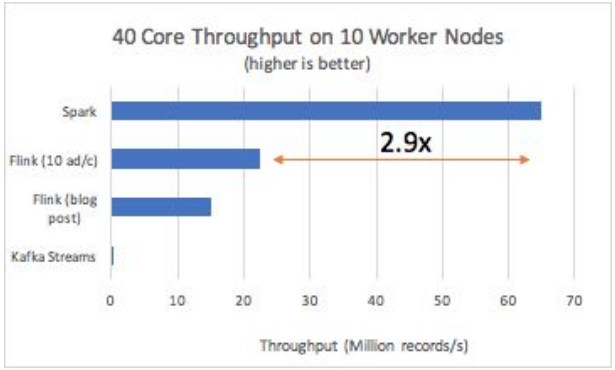


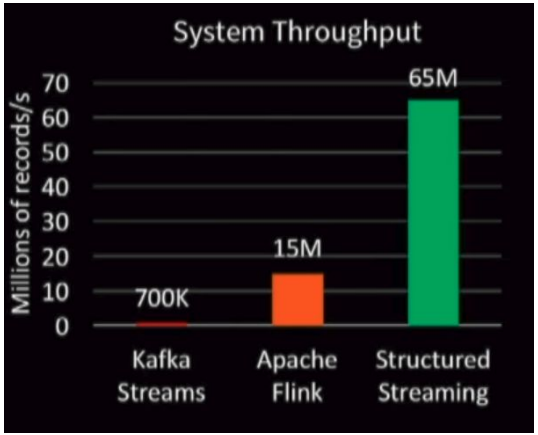
Structured Streaming processing modes: Micro-batch and Continuous Processing

[Abb. 16]

In Zahlen...







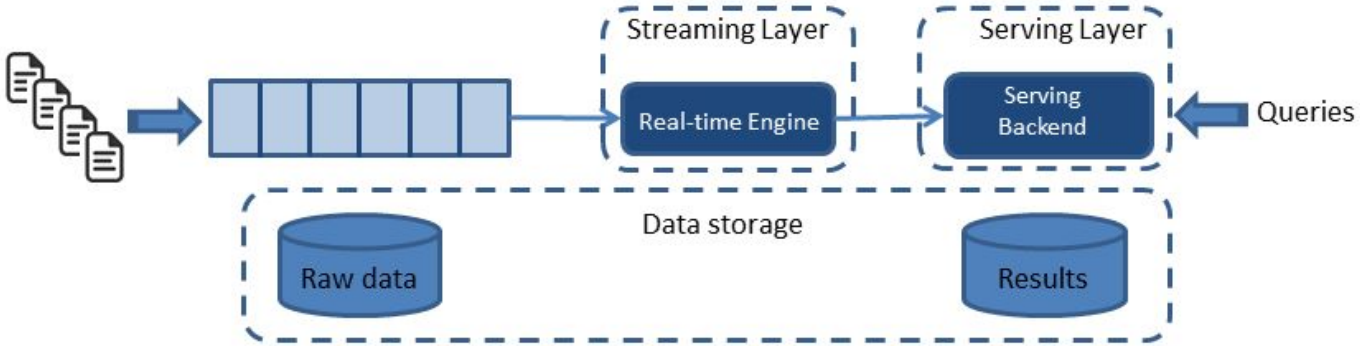
mittels Yahoo! Benchmark für Streaming Systeme

Benchmark Problem

Fazit: Flink & Spark

- Spark & Flink erlauben sowohl Batch & Stream Processing
- Flink als Innovator für
  - **Event-time Windowing**, **out-of-order Streams**, and **exactly-once** Semantik
  - **Checkpointing** & **Replayability** → Consistency
- Adoption einer höheren API, die Batch & Stream transparent macht
- Performanz abhängig vom Anwendungsfall → **Benchmarks** schwer vergleichbar

Kappa Architecture [7]



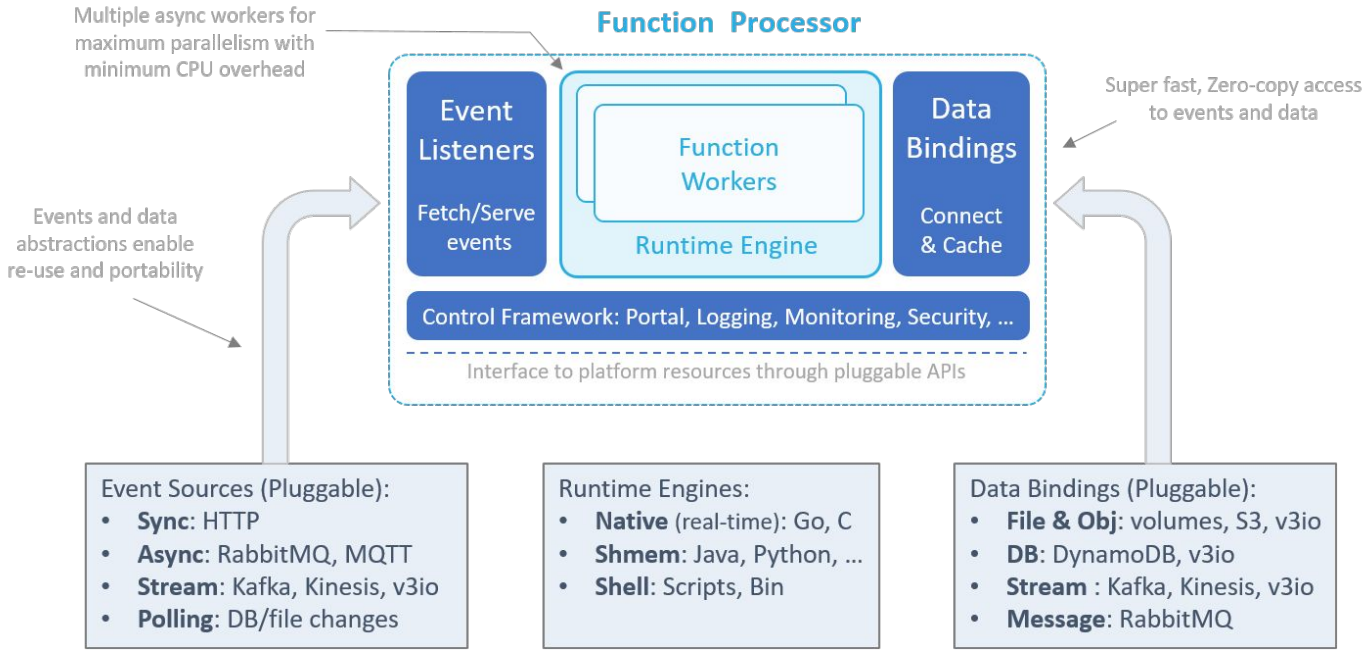
[Abb. 17]

- eine **zentrale Datenpipeline** (für Batch und Streaming → Flink/Spark)
- immutable, append-only **Event-Store (Kafka)** → long-term HDFS/S3
- alle Operationen im Streaming-Layer → **geänderte Logik** → **Replay**

- Aufwand für die Wiedergabe der gesamten Historie steigt linear mit dem Datenvolumen
  - spezieller Use Case

Was ist mit diesem  Lambda ?

Use Case: Nuclio



- E2E Nutzen
- Serverless Visual Analytics?

[Abb. 18]

Research & Conferences

Konferenzen

IEEE International Conference on Big Data (10-13 Dezember 2018)

ACM International Conference on Distributed and Event-based Systems (25-29 June 2018)

IEEE International Conference on Data Engineering (16-19 April 2018)

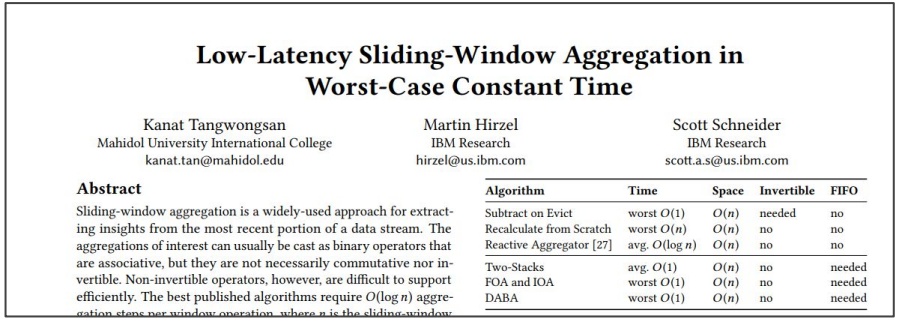
ACM SIGMOD/PODS International Conference on Management of Data (10-15 June 2018)

Andere:

Strata Data conference

Forschung

- Algorithmen & Mechanismen verbessern
- Performance
- Benchmarking & Improvement Detection
- Sicherheit & Governance
- Anwendungsbereich Distributed Machine Learning (nähere Untersuchung notwendig)



Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time

Kanat Tangwongsan
Mahidol University International College
kanat.tan@mahidol.edu

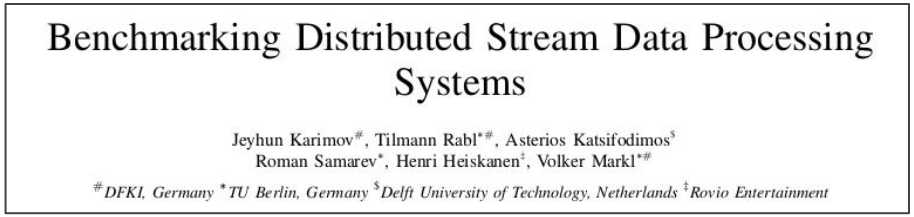
Martin Hirzel
IBM Research
hirzel@us.ibm.com

Scott Schneider
IBM Research
scott.a.s@us.ibm.com

Abstract

Sliding-window aggregation is a widely-used approach for extracting insights from the most recent portion of a data stream. The aggregations of interest can usually be cast as binary operators that are associative, but they are not necessarily commutative nor invertible. Non-invertible operators, however, are difficult to support efficiently. The best published algorithms require $O(\log n)$ aggregation steps per window operation, where n is the sliding-window

Algorithm	Time	Space	Invertible	FIFO
Subtract on Evict	worst $O(1)$	$O(n)$	needed	no
Recalculate from Scratch	worst $O(n)$	$O(n)$	no	no
Reactive Aggregator [27]	avg. $O(\log n)$	$O(n)$	no	no
Two-Stacks	avg. $O(1)$	$O(n)$	no	needed
FOA and IOA	worst $O(1)$	$O(n)$	no	needed
DABA	worst $O(1)$	$O(n)$	no	needed

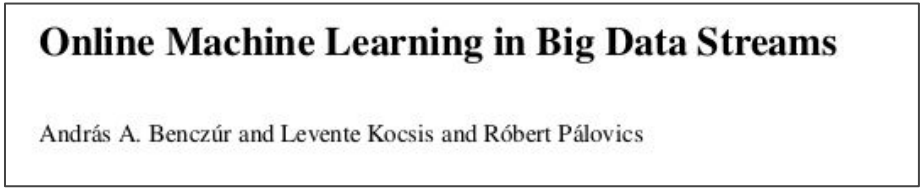


Benchmarking Distributed Stream Data Processing Systems

Jeyhun Karimov[#], Tilmann Rabl[*#], Asterios Katsifodimos[$]
Roman Samarev[*], Henri Heiskanen[‡], Volker Markl[*#]

[#] *DFKI, Germany* [*] *TU Berlin, Germany* [$] *Delft University of Technology, Netherlands* [‡] *Rovio Entertainment*

Exploring the Impact of Processing Guarantees on Performance of Stream Data Processing



Online Machine Learning in Big Data Streams

András A. Benczúr and Levente Kocsis and Róbert Pálovics

Riffle: Optimized Shuffle Service for Large-Scale Data Analytics

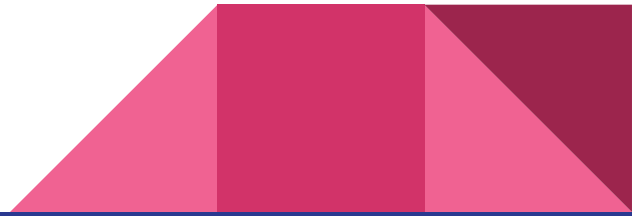
Haoyu Zhang[⋆], Brian Cho[†], Ergin Seyfe[†], Avery Ching[†], Michael J. Freedman[⋆]

[⋆] *Princeton University* [†] *Facebook, Inc.*

Course of action

Cours of action

- tieferes Verständnis der Streaming Anwendungen/Design
- Architekturansätze vergleichen
- eigene Architektur aufbauen & testen
- Vergleich/Benchmarks
- tieferes Verständnis der anwendbaren Algorithmen
- Use Case: Machine/Deep Learning näher untersuchen

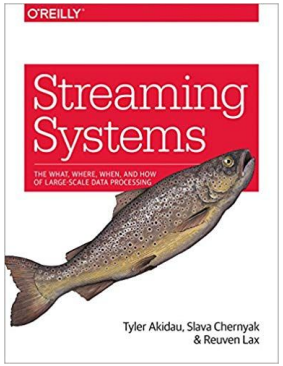


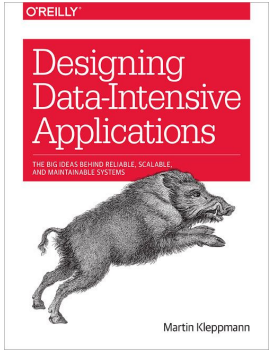
Referenzen

1. Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113
2. Akidau, T., Chernyak, S. & Lax, R. (2018). *Streaming systems : the what, where, when, and how of large-scale data processing*. Sebastopol, CA: O'Reilly.
3. Marz, N. (2011, October 13). Http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html. Retrieved November 19, 2018, from http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html
4. Wingerath, Wolfram, et al. "Real-time stream processing for Big Data." it-Information Technology 58.4 (2016): 186-194
5. Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. 2015. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc. VLDB Endow. 8, 12 (August 2015), 1792-1803.
6. Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. 2017. State management in Apache Flink®: consistent stateful distributed stream processing. Proc. VLDB Endow. 10, 12 (August 2017), 1718-1729
7. Kreps, J. (2014, July 2). Questioning the Lambda Architecture. Retrieved November 19, 2018, from https://www.oreilly.com/ideas/questioning-the-lambda-architecture

Bilderverzeichnis

Hilfreiches Material





Gesamtüberblick

https://flink.apache.org/ + Papers

https://spark.apache.org/ + Papers

Streaming 101/ 102 Blog Posts

Fragen
oder
Anmerkungen ?