Scalable Stream Processing

Surveying Storm, Samza, Spark & Flink

Wolfram Wingerath
ww@baqend.com
September 12, techcamp 2018, Hamburg

BaQend

@baqendcom

About me
Wolfram Wingerath

PhD Thesis & Research

Distributed Systems Engineer

Research:
- Real-Time Databases
- Stream Processing
- NoSQL & Cloud Databases
- …

Practice:
- Backend-as-a-Service
- Web Caching
- Real-Time Database
- …

Universität Hamburg

www.baqend.com

Outline

Introduction
Big Data in Motion

System Survey
Big Data + Low Latency

Wrap-Up
Summary & Discussion

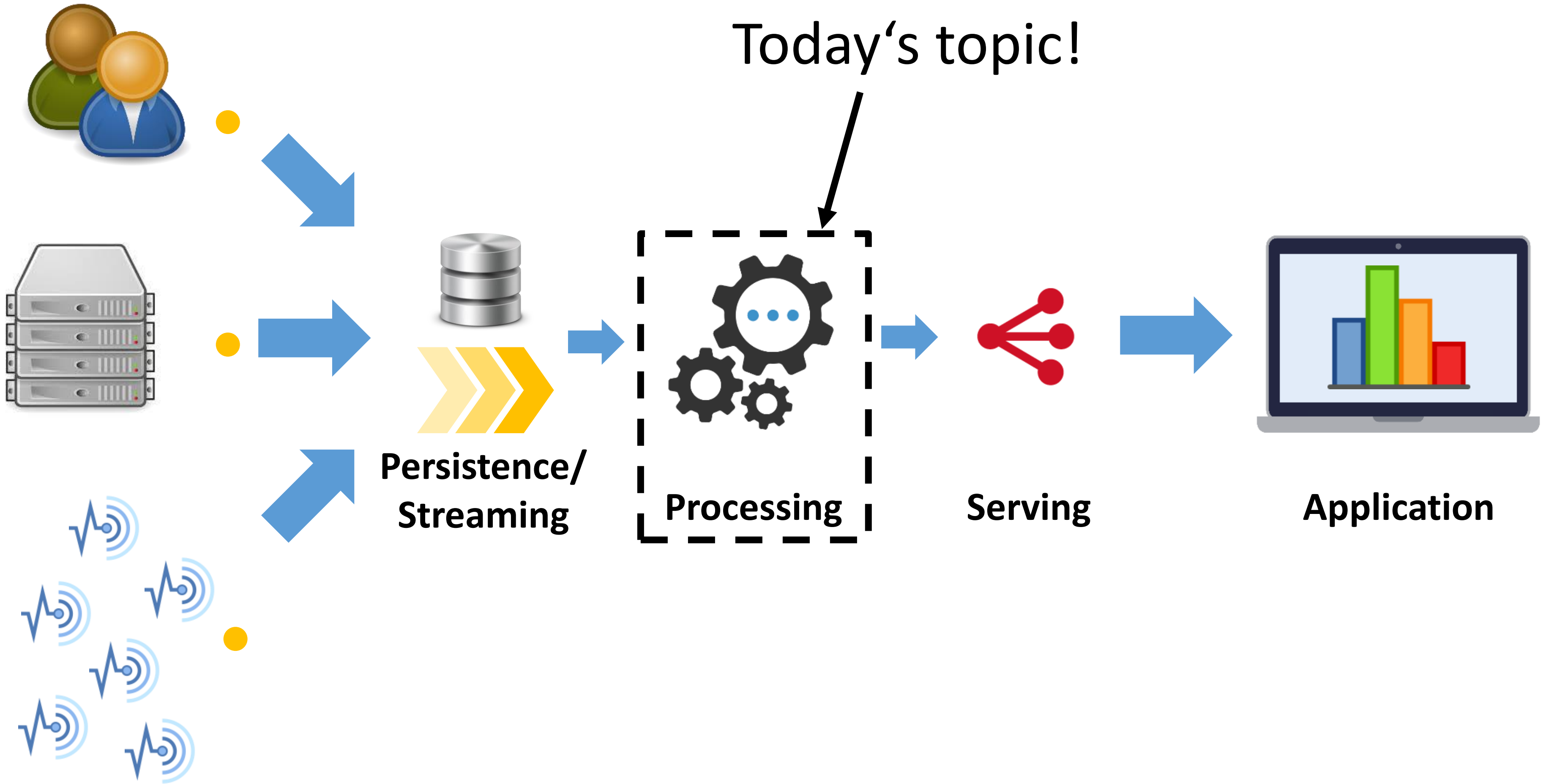
Future Directions
Real-Time Databases

- **Big Picture:**
  - A Typical Data Pipeline
  - Processing Frameworks
- **Processing Models:**
  - Batch Processing
  - Stream Processing

IN PRACTICE

Scalable Data Processing

A Data Processing Pipeline

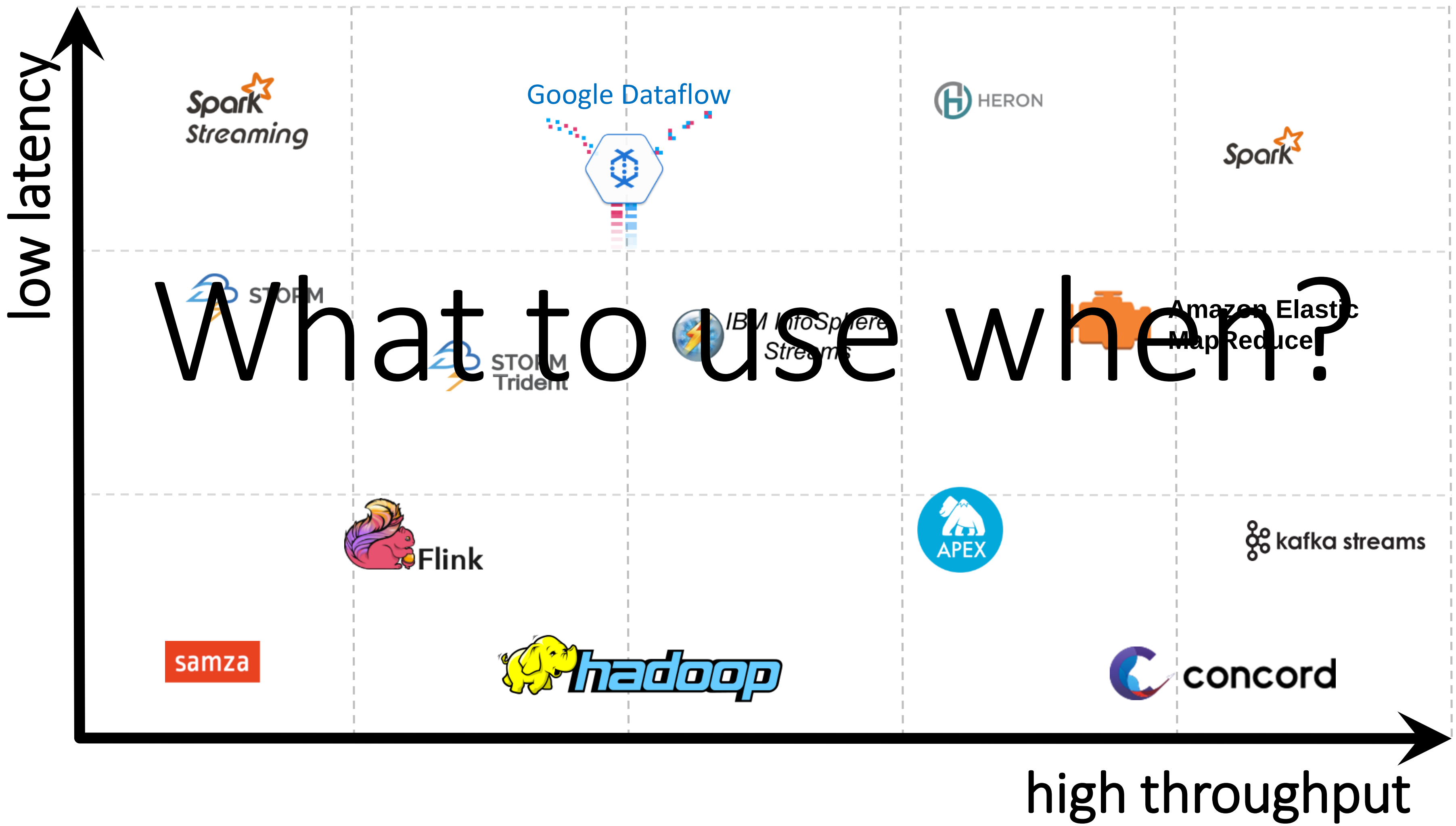


INTRODUCTION

Batch vs Stream Processing

Big Data Processing Frameworks

What are your options?



What to use when?

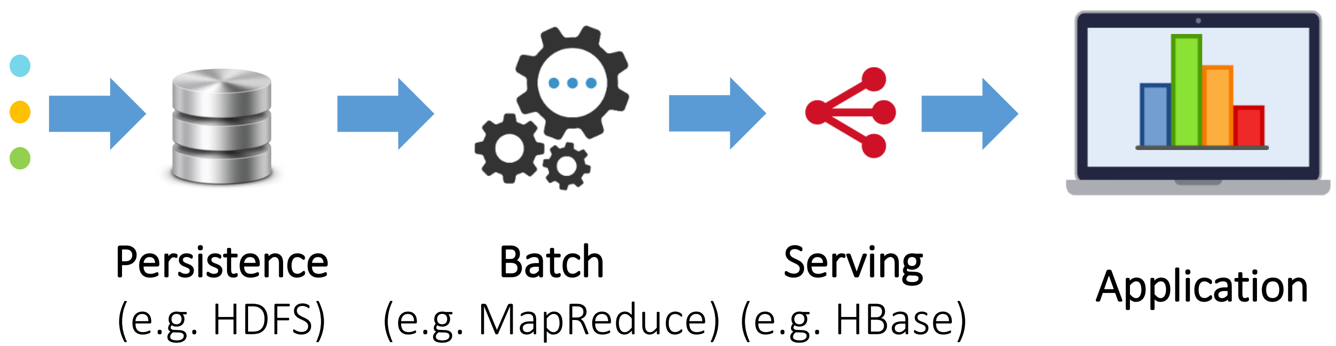
Batch Processing

„Volume“

- **Cost-effective** & Efficient
- **Easy to reason about**: operating on complete data

But:

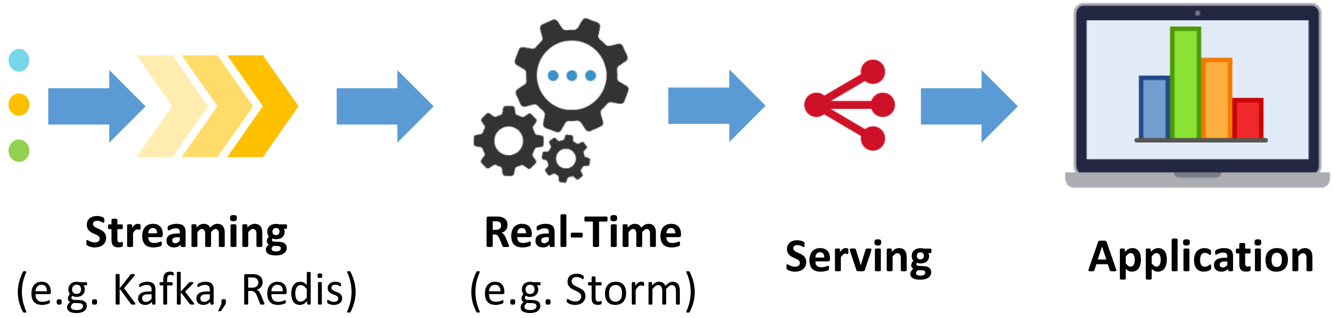
- High latency: jobs periodic (e.g. during night times)



Stream Processing

„Velocity“

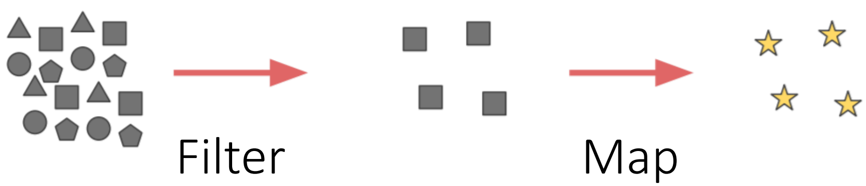
- **Low end-to-end latency**
- Challenges:
  - **Long-running jobs** - no downtime allowed
  - **Asynchronism** - data may arrive delayed or out-of-order
  - **Incomplete input** - algorithms operate on partial data
  - More: fault-tolerance, state management, guarantees, ...



Typical Stream Operators

Examples

Filter & Transform

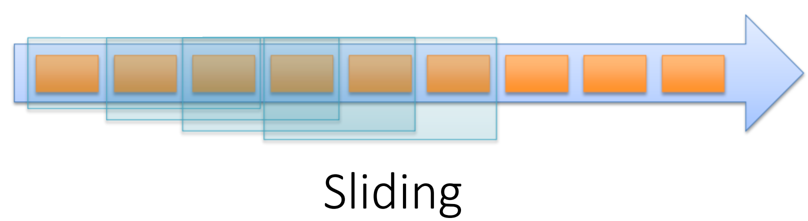


Group

Aggregates

SUM()
COUNT()

Windows



https://www.infoq.com/presentations/stream-processors-databases

https://www.infoq.com/presentations/stream-processing-apache-flink

Typical Use Case

Example from Yahoo!

Input

- **Read** Ad tracking data **from Kafka**

Filter

- **Discard** useless data

Project

- **Extract** relevant fields

Group

- By Ad **campaign**

Window

- Ad views per **10-min-window**

https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at

Wrap-up
Data Processing

- Processing frameworks abstract from **scaling issues**

Batch processing
- easy to reason about
- extremely efficient
- huge input-output latency

Stream processing
- quick results
- purely incremental
- potentially complex to handle

Outline

Introduction
Big Data in Motion

System Survey
Big Data + Low Latency

Wrap-Up
Summary & Discussion

Future Directions
Real-Time Databases

- **System Survey:**
  - Processing Model Overview
  - Storm/Trident
  - Samza
  - Spark Streaming
  - Flink

SURVEY

Popular Stream Processing Systems

Processing Models

Batch vs. Micro-Batch vs. Stream

stream	micro-batch	batch
Flink		hadoop
STORM Trident		Spark Streaming
samza		Amazon Elastic MapReduce

low latency ⟷ high throughput

Storm
„Hadoop of real-time“

STORM

Overview

- **First** production-ready, well-adopted stream processor
- **Compatible**: native Java API, Thrift, distributed RPC
- **Low-level**: no primitives for joins or aggregations
- **Native stream processor**: latency < 50 ms feasible
- **Big users**: Twitter, Yahoo!, Spotify, Baidu, Alibaba, …

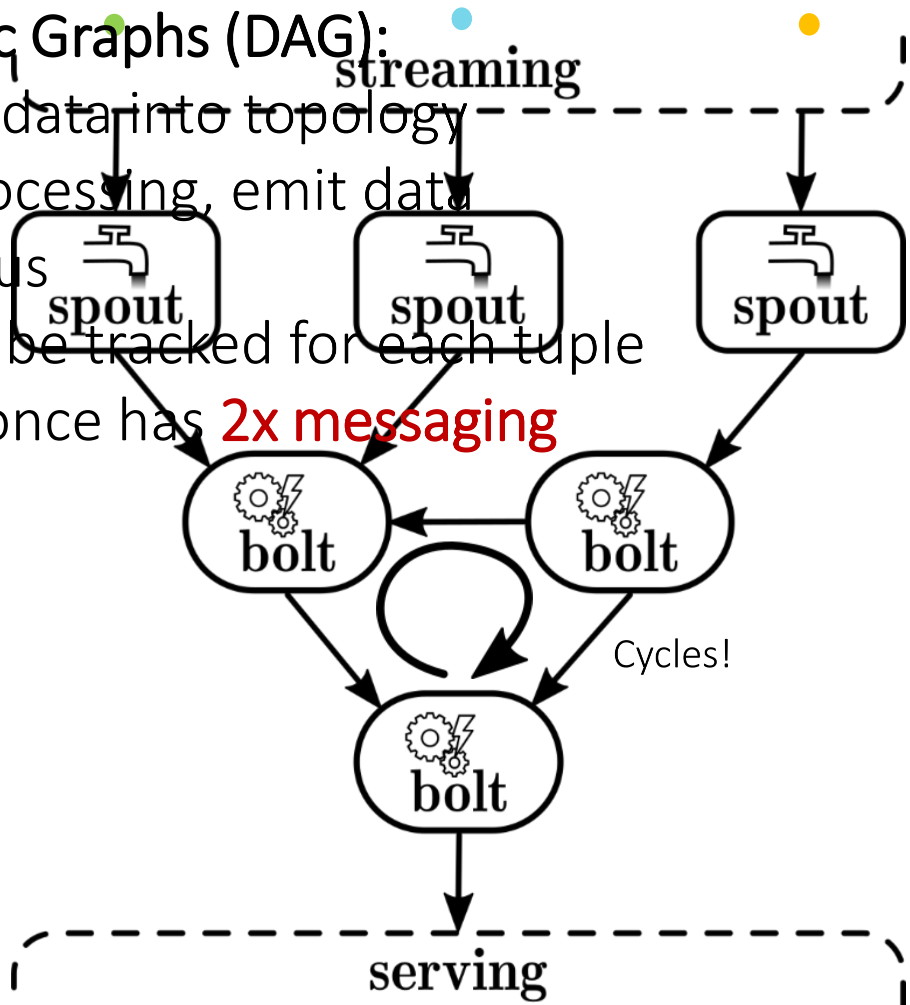
History

- **2010**: developed at BackType (acquired by Twitter)
- **2011**: open-sourced
- **2014**: Apache top-level project

Dataflow

Directed Acyclic Graphs (DAG):

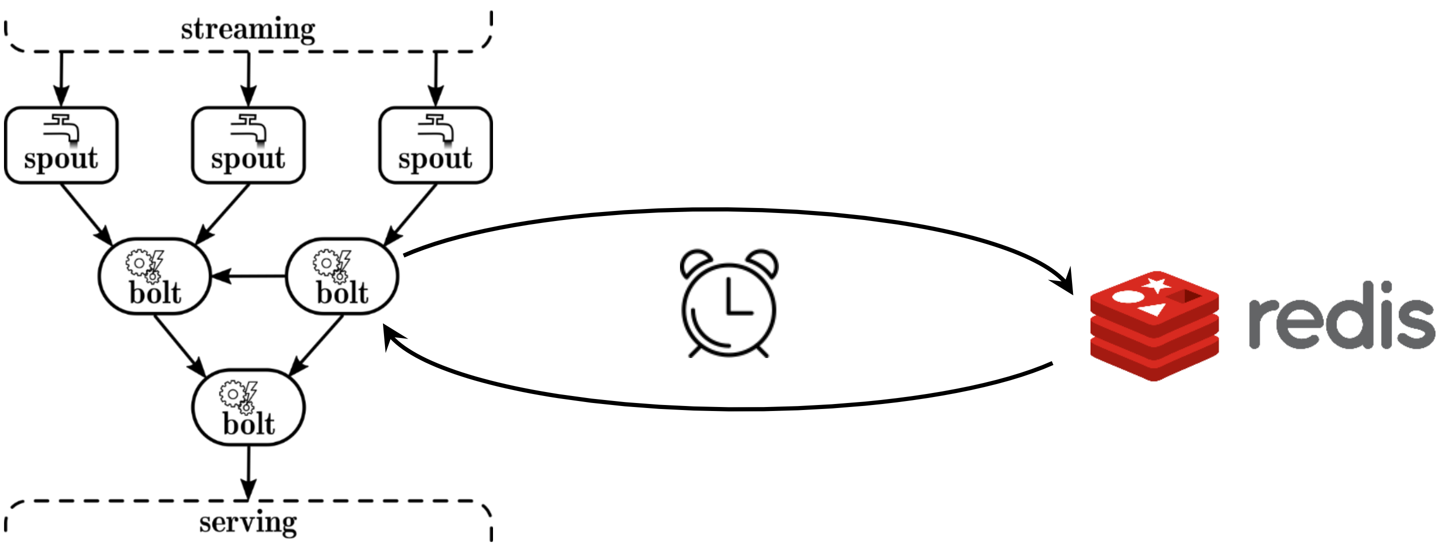
- **Spouts**: pull data into topology
- **Bolts**: do processing, emit data
- Asynchronous
- Lineage can be tracked for each tuple
  → At-least-once has 2x messaging overhead



State Management

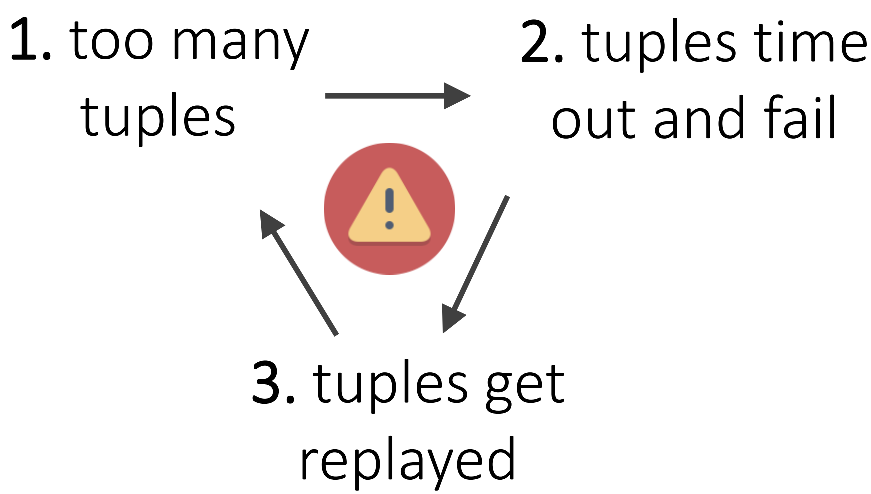
Recover State on Failure

- **In-memory** or **Redis**-backed reliable state
- *Synchronous state communication* on the critical path
  → infeasible for large state



Back Pressure

Throttling Ingestion on Overload



Approach: monitoring bolts‘ inbound buffer

1. Exceeding **high watermark** → throttle!
2. Falling below **low watermark** → full power!

Trident

Stateful Stream Joining on Storm

Overview:

- Abstraction layer on top of Storm
- Released in 2012 (Storm 0.8.0)
- **Micro-batching**
- **New features**:
  - High-level API: aggregations & joins
  - Strong ordering
  - Stateful exactly-once processing

    → Performance penalty

Trident

Partitioned Micro-Batching

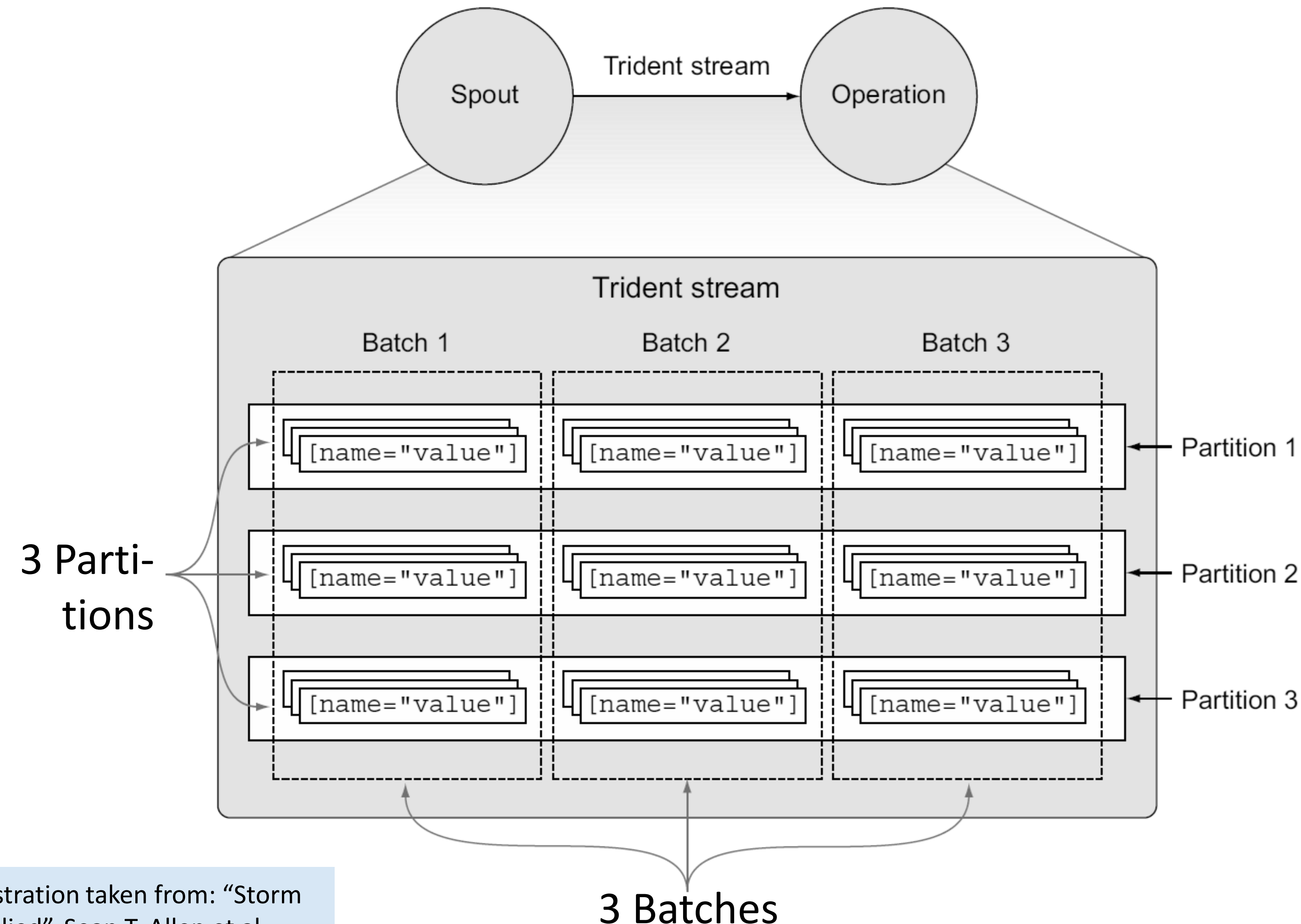
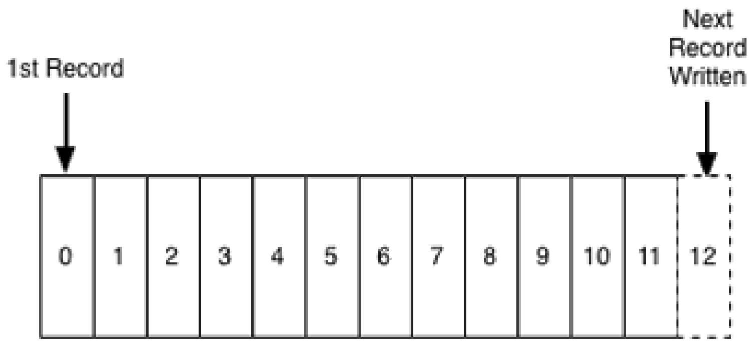


Illustration taken from: “Storm applied”, Sean T. Allen et al.

Samza

Real-Time on Top of Kafka



Overview

- Co-developed with **Kafka**
  → **Kappa Architecture**
- **Simple**: only single-step jobs
- Local state
- Native stream processor: low latency
- **Users**: LinkedIn, Uber, Netflix, TripAdvisor, Optimizely, ...

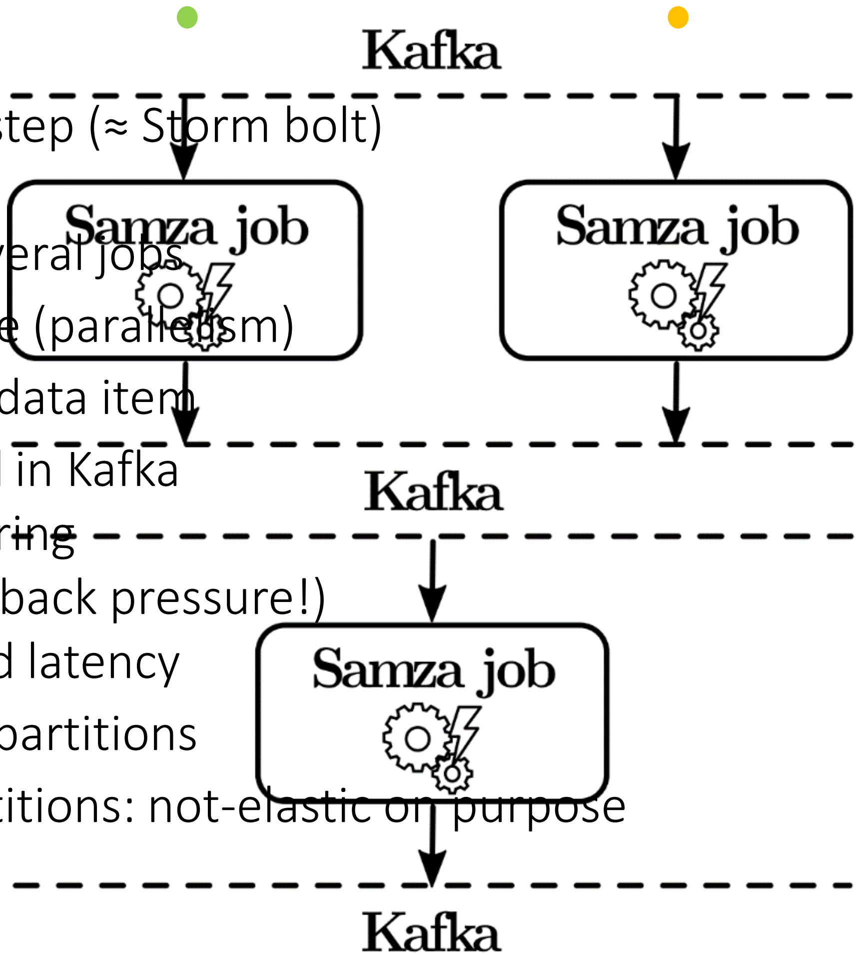
History

- Developed at **LinkedIn**
- **2013**: open-source (Apache Incubator)
- **2015**: Apache top-level project

Illustration taken from: Jay Kreps, *Questioning the Lambda Architecture* (2014)
https://www.oreilly.com/ideas/questioning-the-lambda-architecture (2017-03-02)

Dataflow

Simple By Design



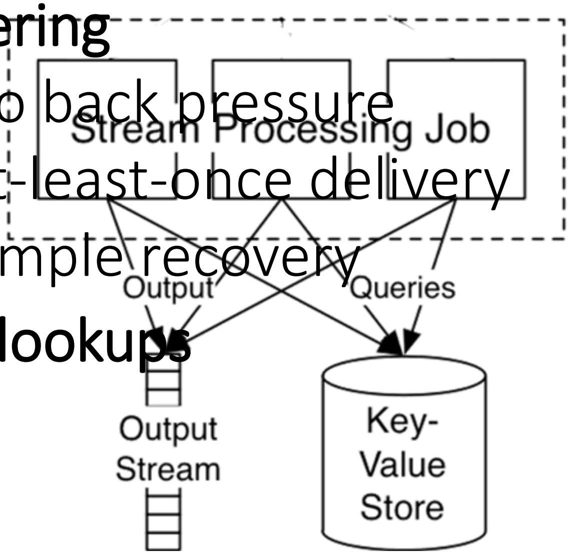
- **Job**: processing step (≈ Storm bolt)
  → Robust
  → But: often several jobs
- **Task**: job instance (parallelism)
- **Message**: single data item
- **Output persisted** in Kafka
  → Easy data sharing
  → Buffering (no back pressure!)
  → But: Increased latency
- **Ordering** within partitions
- Task = Kafka partitions: not-elastic on purpose

Martin Kleppmann, *Turning the database inside-out with Apache Samza* (2015)
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/ (2017-02-23)

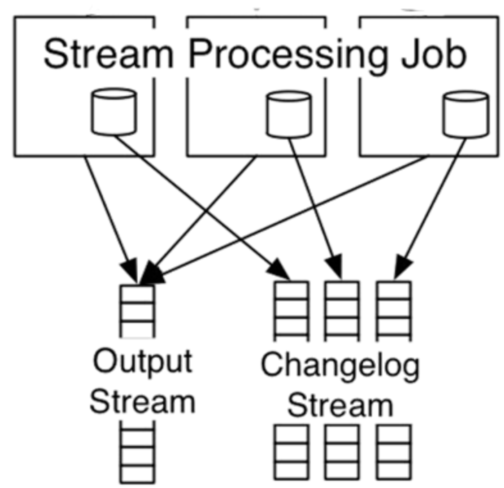
Samza
Local State

Advantages of local state:

- **Buffering**
  → No back pressure
  → At-least-once delivery
  → Simple recovery
- **Fast lookups**



Remote State



Local State

Illustrations taken from: Jay Kreps, *Why local state is a fundamental primitive in stream processing* (2014)
https://www.oreilly.com/ideas/why-local-state-is-a-fundamental-primitive-in-stream-processing (2017-02-26)

Dataflow

Example: Enriching a Clickstream

Example: the *enriched clickstream* is available to every team within the organization

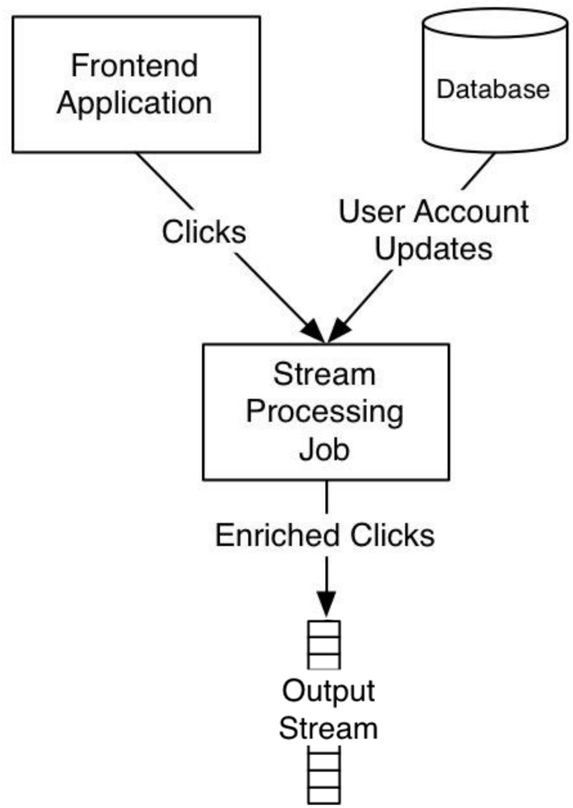


Illustration taken from: Jay Kreps, *Why local state is a fundamental primitive in stream processing* (2014)
https://www.oreilly.com/ideas/why-local-state-is-a-fundamental-primitive-in-stream-processing (2017-02-26)

State Management

Straightforward Recovery

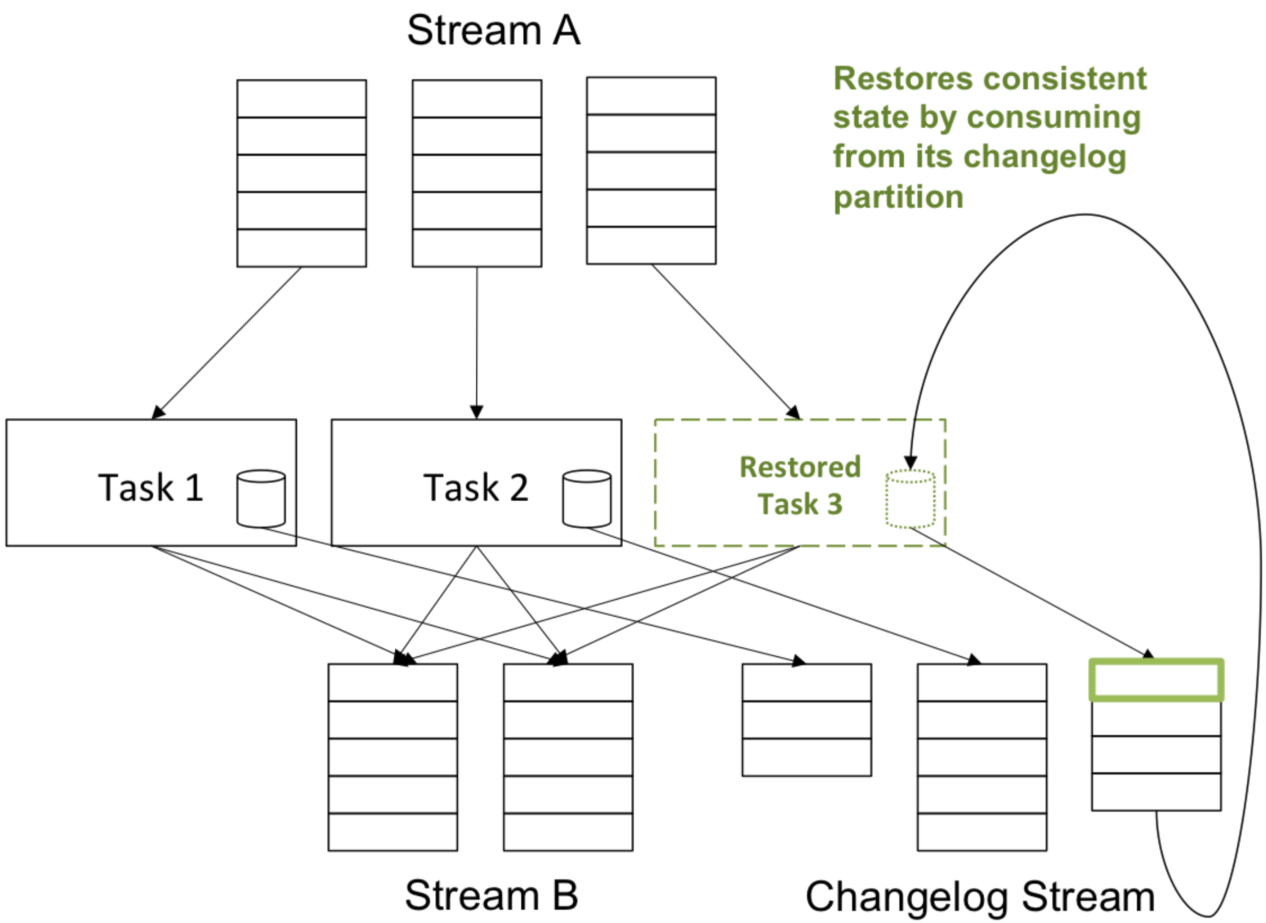


Illustration taken from: Navina Ramesh, *Apache Samza, LinkedIn's Framework for Stream Processing* (2015) https://thenewstack.io/apache-samza-linkedins-framework-for-stream-processing (2017-02-26)

Spark

„MapReduce successor“

Overview

- **High-level API**: immutable collections (RDDs)

Core	SQL	MLlib	GraphX	Spark Streaming

- **Community**: 1000+ contributors in 2015
- **Big users**: Amazon, eBay, Yahoo!, IBM, Baidu, ...

History

- **2009**: developed at UC Berkeley
- **2010**: open-sourced
- **2014**: Apache top-level project

Spark Streaming

Overview

- **High-level API**: DStreams (~Java 8 Streams)
- **Micro-Batching**: seconds of latency
- **Rich features**: stateful, exactly-once, elastic

History

- **2011**: start of development
- **2013**: Spark Streaming becomes part of Spark Core

Spark Streaming

Core Abstraction: DStream

Resilient Distributed Data set (RDD)

- **Immutable** collection & **deterministic** operations
- **Lineage** tracking:
  → state can be reproduced
  → periodic checkpoints reduce recovery time

DStream: Discretized RDD

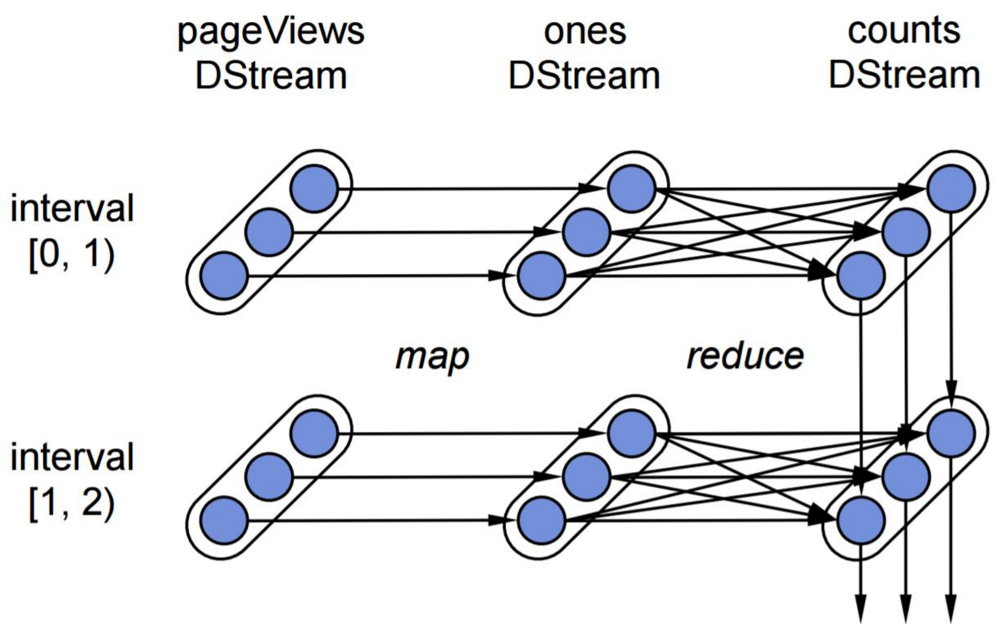
- **RDDs are processed in order**: no ordering within RDD
- RDD scheduling ~50 ms → latency >100ms

Illustration taken from:
http://spark.apache.org/docs/latest/streaming-programming-guide.html#overview (2017-02-26)

Example

Counting Page Views

```
pageViews = readStream("http://...", "1s")
ones = pageViews.map(event => (event.url, 1))
counts = ones.runningReduce((a, b) => a + b)
```



Zaharia, Matei, et al. "Discretized streams: Fault-tolerant streaming computation at scale." *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013.

Flink

Overview

- **Native stream processor:** Latency <100ms feasible
- **Abstract API** for stream and batch processing, stateful, exactly-once delivery
- **Many libraries**: Table and SQL, CEP, Machine Learning , Gelly…
- **Users**: Alibaba, Ericsson, Otto Group, ResearchGate, Zalando…

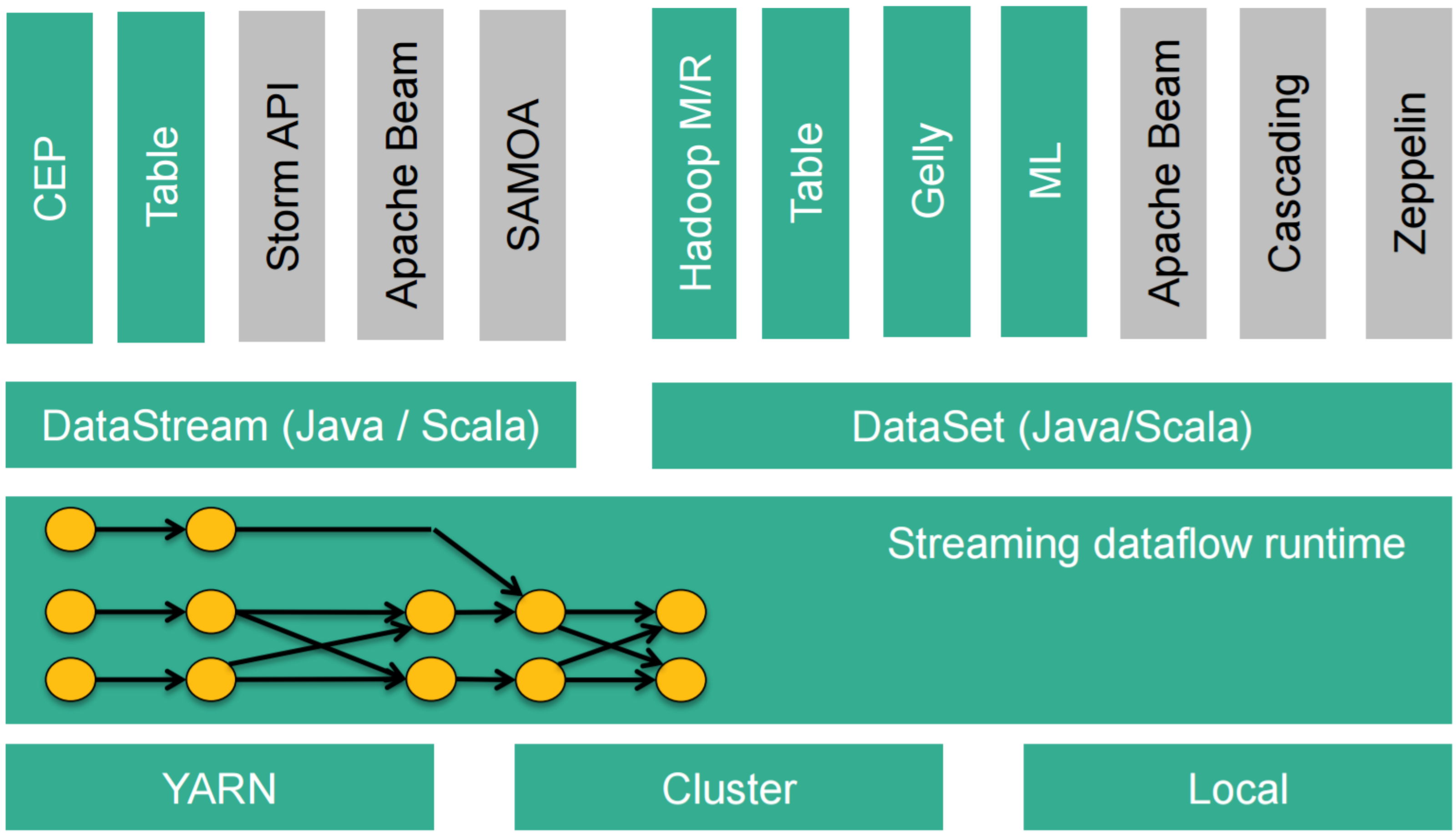
History

- **2010**: start as **Stratosphere** at TU Berlin, HU Berlin, and HPI Potsdam
- **2014**: Apache Incubator, project renamed to Flink
- **2015**: Apache top-level project

Architecture

Streaming + Batch

Flink

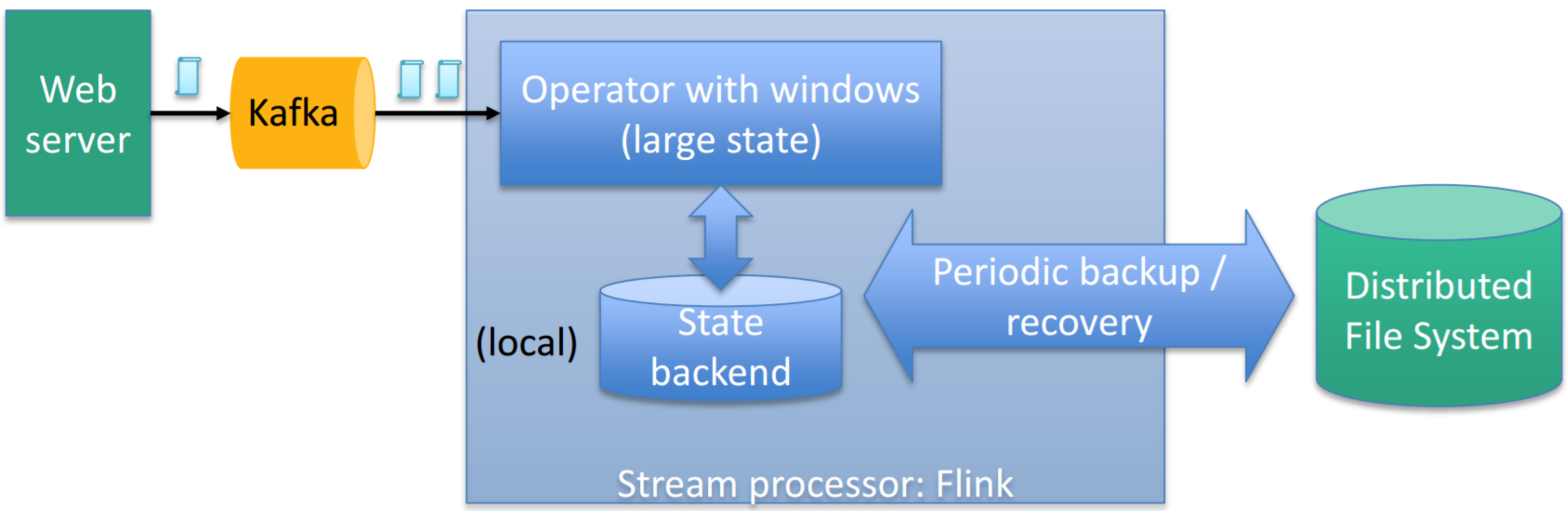


https://www.infoq.com/presentations/stream-processing-apache-flink

Managed State

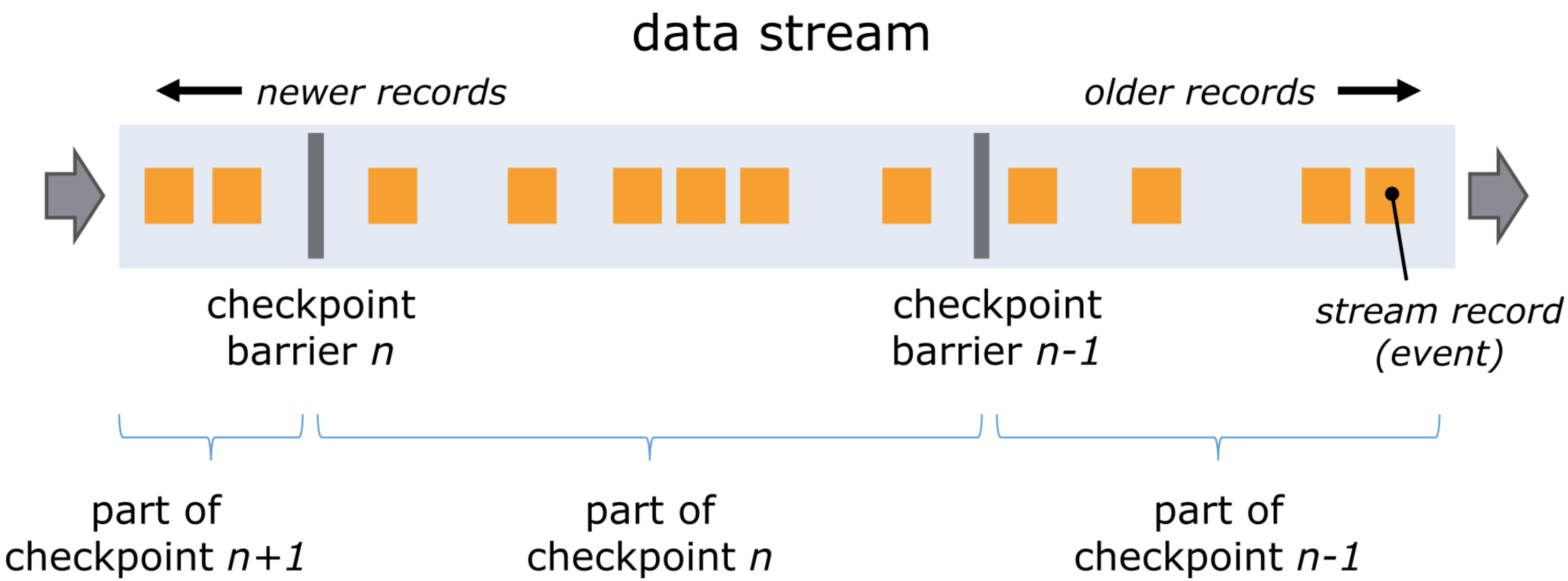
Streaming + Batch

- Automatic **Backups** of local state
- Stored in **RocksDB,** Savepoints written to **HDFS**



https://www.infoq.com/presentations/stream-processing-apache-flink

Highlight: Fault Tolerance
Distributed Snapshots



- **Ordering** within stream partitions
- Periodic **checkpoints**
- **Recovery**:
  1. *reset state* to checkpoint
  2. *replay data* from there

→ **Exactly-once**

Illustration taken from: https://ci.apache.org/projects/flink/flink-docs-release-1.2/internals/stream_checkpointing.html (2017-02-26)

Outline

Introduction
Big Data in Motion

System Survey
Big Data + Low Latency

Wrap-Up
Summary & Discussion

Future Directions
Real-Time Databases

- **Discussion:**
  - Comparison Matrix
  - Other Systems
- **Takeaway**

WRAP UP

Side-by-side comparison

Comparison

	Storm	Trident	Samza	Spark Streaming	Flink (streaming)
Strictest Guarantee	at-least-once	exactly-once	at-least-once	exactly-once	exactly-once
Achievable Latency	≪100 ms	<100 ms	<100 ms	<1 second	<100 ms
State Management	○ (small state)	○ (small state)	✓	✓	✓
Processing Model	one-at-a-time	micro-batch	one-at-a-time	micro-batch	one-at-a-time
Backpressure	✓	✓	no (buffering)	✓	✓
Ordering	✗	between batches	within partitions	between batches	within partitions
Elasticity	✓	✓	✗	✓	✓

Performance
Yahoo! Benchmark

- Based on **real use case**:
  - Filter and count ad impressions
  - 10 minute windows

"**Storm** […] and **Flink** […] show **sub-second latencies** at relatively high throughputs with **Storm** having the **lowest 99th percentile** latency. **Spark** streaming […] supports high **throughputs**, but at a relatively **higher latency**."

From https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at

Other Systems

Heron

Apex

Dataflow

Beam

Kafka Streams

IBM InfoSphere Streams

And even more: Kinesis, Gearpump, MillWheel, Muppet, S4, Photon, …

Outline

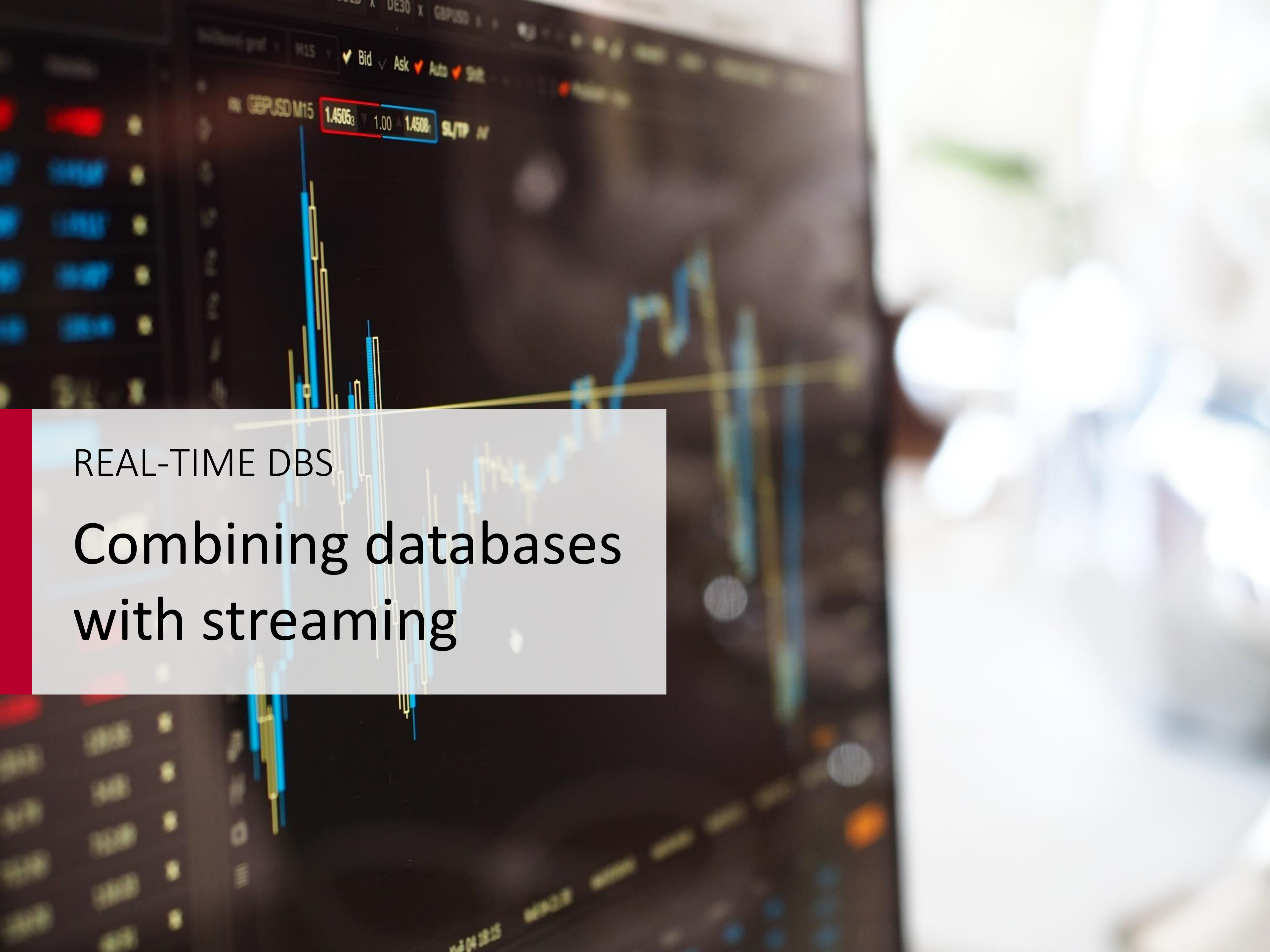
Introduction
Big Data in Motion

System Survey
Big Data + Low Latency

Wrap-Up
Summary & Discussion

Future Directions
Real-Time Databases

- **Real-Time Databases:**
  - **Why** Push-Based Database Queries?
  - **Where** Do Real-Time Databases Fit in?
- **Comparison Matrix:**
  - Meteor
  - RethinkDB
  - Parse
  - Firebase
  - Baqend

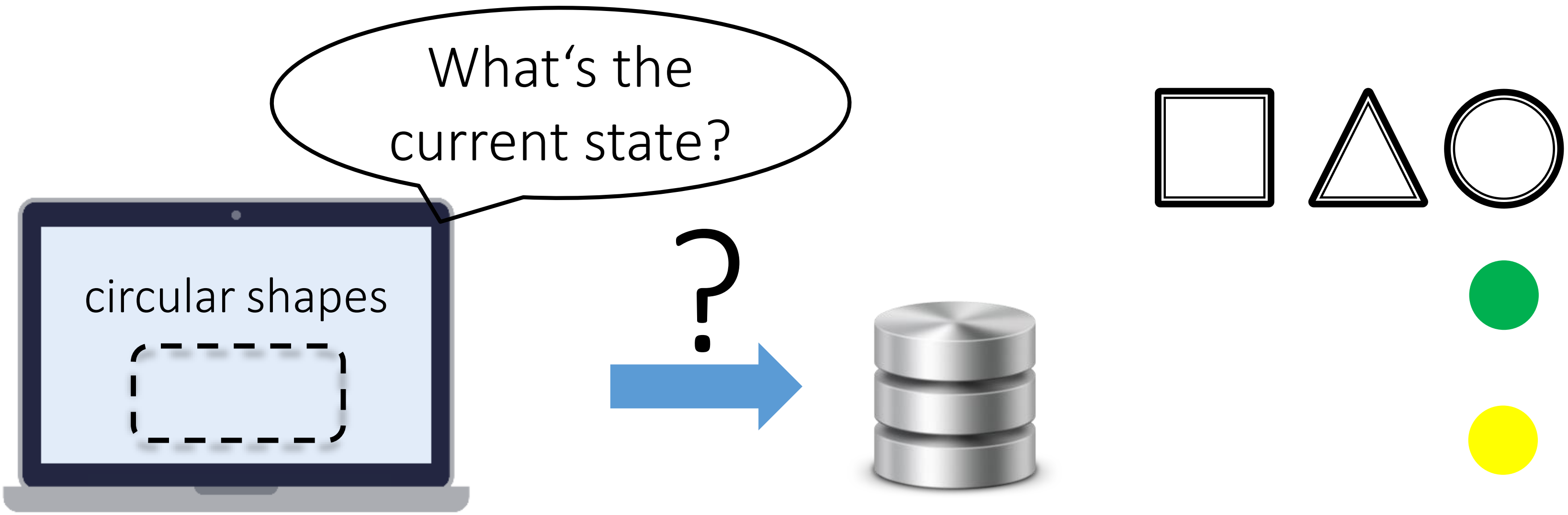


REAL-TIME DBS

Combining databases with streaming

Traditional Databases

No Request? No Data!



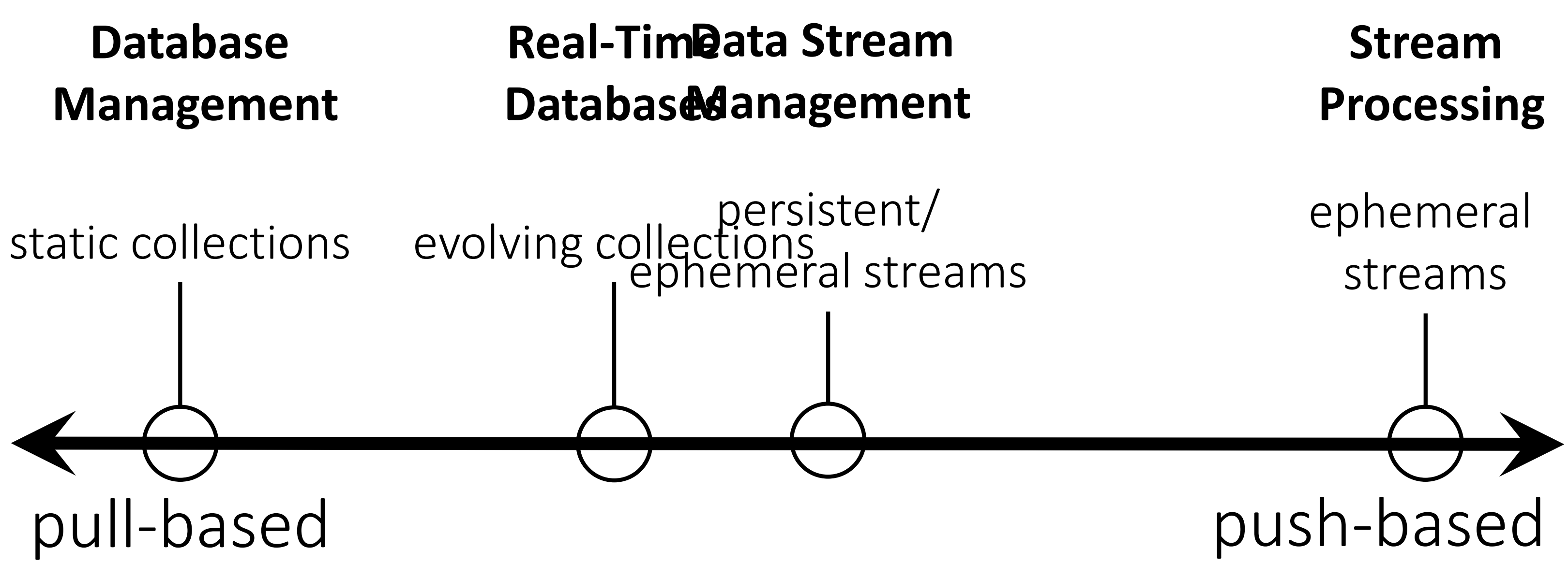
Query maintenance: periodic polling

→ Inefficient

→ Slow

Quick Comparison

DBMS vs. RT DB vs. DSMS vs. Stream Processing



Real-Time Databases

In a Nutshell

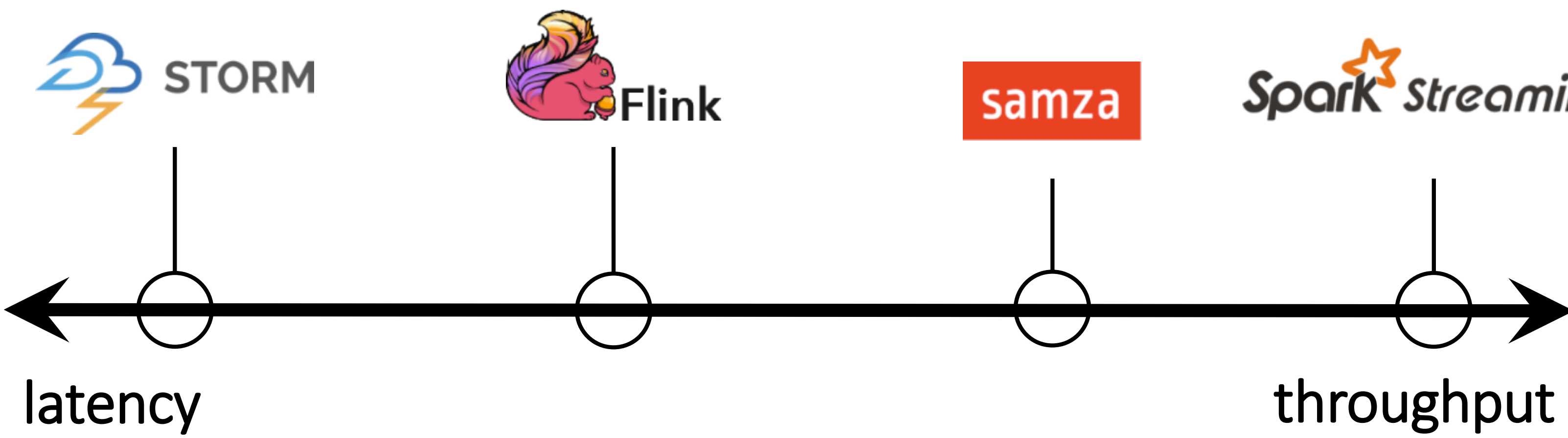
	Meteor Poll-and-Diff	Meteor Oplog Tailing	RethinkDB	Parse	Firebase
Scales with write TP	✓	✗	✗	✗	✗
Scales with no. of queries	✗	✓	✓	✓	? (100k connections)
Composite queries (`AND`/`OR`)	✓	✓	✓	✓	○ (`AND` In Firestore)
Sorted queries	✓	✓	✓	✗	○ (single attribute)
Limit	✓	✓	✓	✗	✓
Offset	✓	✓	✗	✗	○ (value-based)

TAKEAWAY

Trade-Offs in Stream Processing

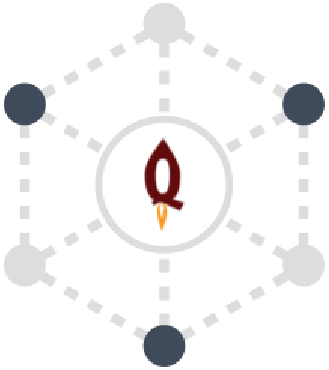
Summary

- **Stream Processors:**



- **Real-Time Databases** integerate Storage & Streaming
- **Learn more**: slides.baqend.com

Who We Are



Our Product

Speed Kit:

- Accelerates *Any* Website
- Pluggable
- Easy Setup

test.speed-kit.com

Our Services

- Web & Data Management Workshops
- Performance Auditing
- Implementation Services

consulting@baqend.com

Baqend

Wolfram Wingerath

ww@baqend.com

Distributed Systems Engineer