Big Data **Analytics** at Scale

Designing Complex Data Pipelines for Continuous Insights

Invited Talk at Kristu Jayanti College

March 01, 2024

Wolle

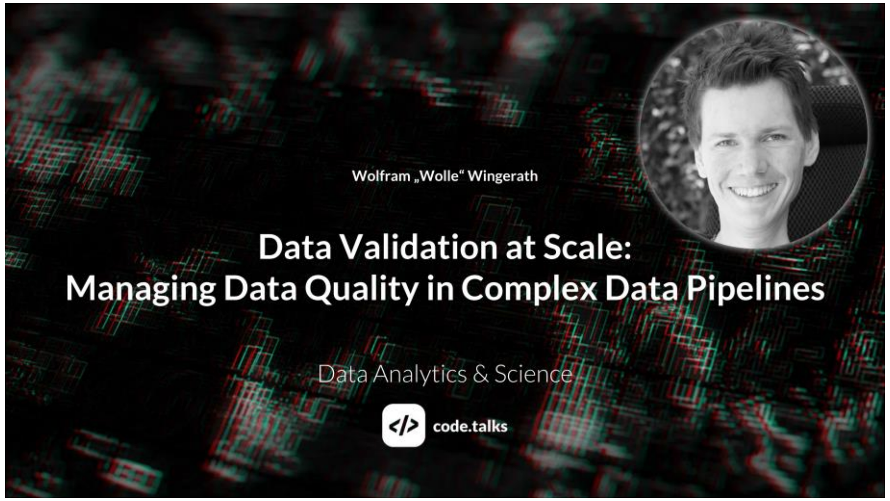
Material Available at https://wolle.science

A Presentation in 2 Parts

Part 1: Big Data Analytics Frameworks
(Stream Processing Systems)

Wolfram „Wolle" Wingerath. Real-Time Processing Explained: A Survey of Storm, Samza, Spark & Flink, code.talks (2018)

Part 2: Data Validation at Scale
(Real-World Example)



Wolfram „Wolle" Wingerath. Data Validation at Scale: Managing Data Quality in Complex Data Pipelines, code.talks (2023)

Note: Not all material is included in the live presentation, but the slides and the above talks contain all the details!

I Am Wolle

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

+

Practice:
- Web Caching
- Big Data Analytics
- Anger Management
- …

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Carl von Ossietzky Universität Oldenburg

Baqend

Part 1: Big Data Analytics Frameworks

1 **Processing Paradigms**

Where is the difference between batch & stream processing and what are their resp. benefits?

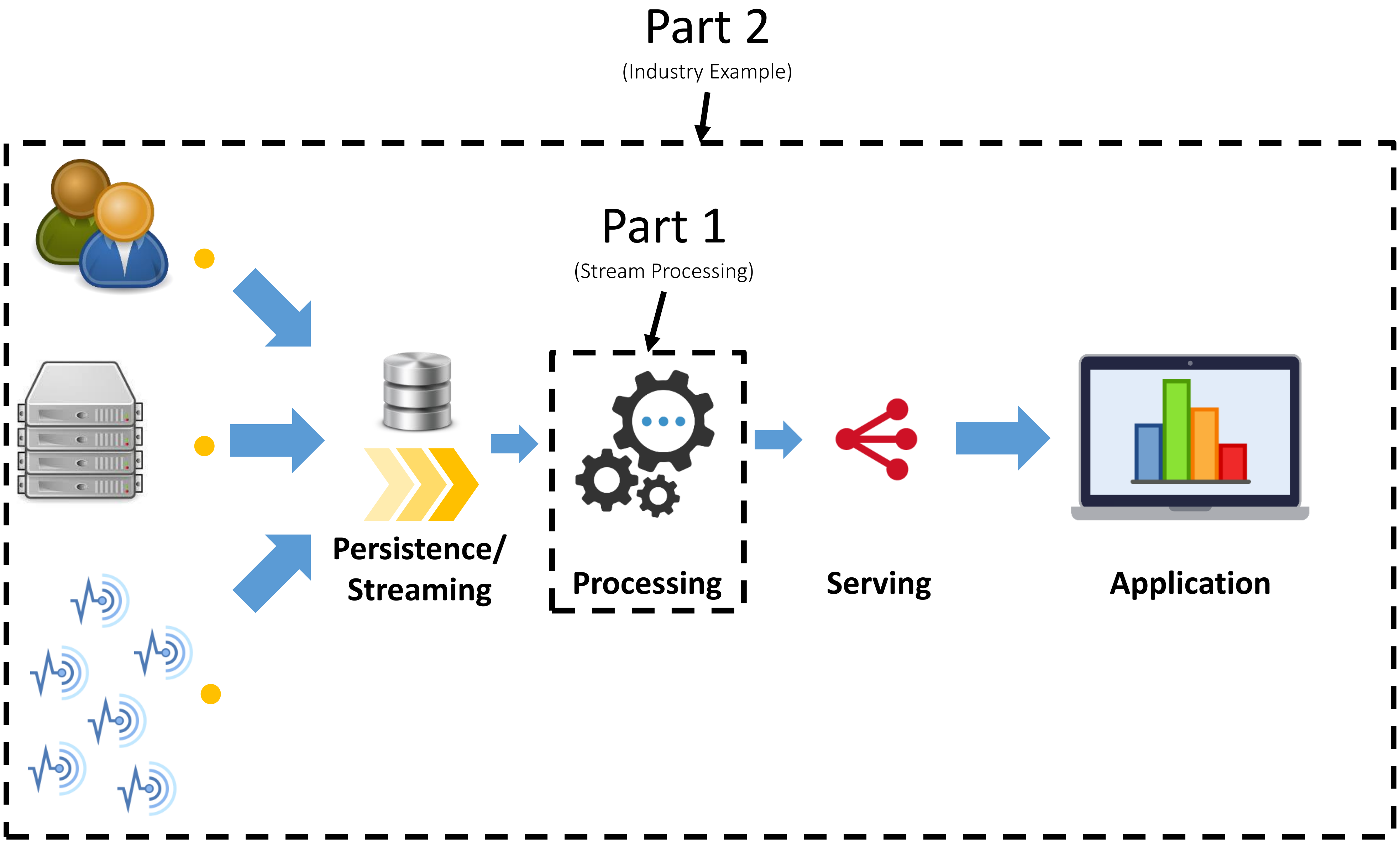
2 **Stream Processing Systems**

How did the stream processing system space evolve and what are the landmark systems?

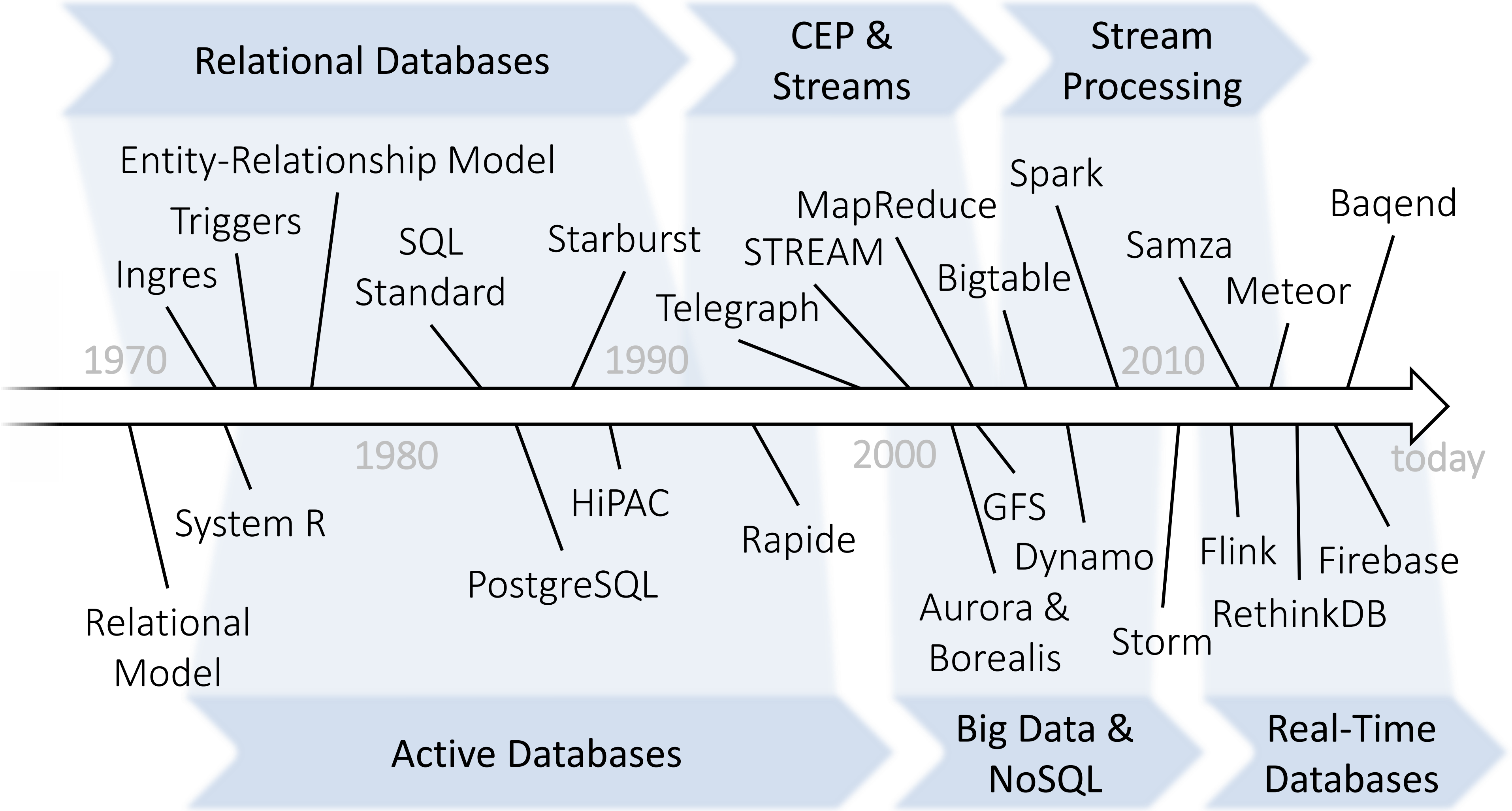
3 **System Comparison & Trade-Offs**

What are fundamental design decisions and how do they reflect in the individual systems?

A Data Processing Pipeline



Data Management Through the Ages: Historical Context



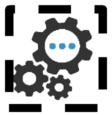
Data Processing Frameworks: Scale-Out Made Feasible

Data processing frameworks **hide complexities of scaling**, e.g.:

- **Deployment -** code distribution, starting/stopping work
- **Monitoring -** health checks, application stats
- **Scheduling -** assigning work, rebalancing
- **Fault-tolerance** - restarting workers, rescheduling failed work

Running in cluster

Running on single node



Scaling out

Big Data Processing Frameworks: What are your options?

What to use when?

Processing Models: Batch vs. Micro-Batch vs. Stream

stream | **micro-batch** | **batch**

Flink

STORM Trident

samza

hadoop

Spark Streaming

Amazon Elastic MapReduce

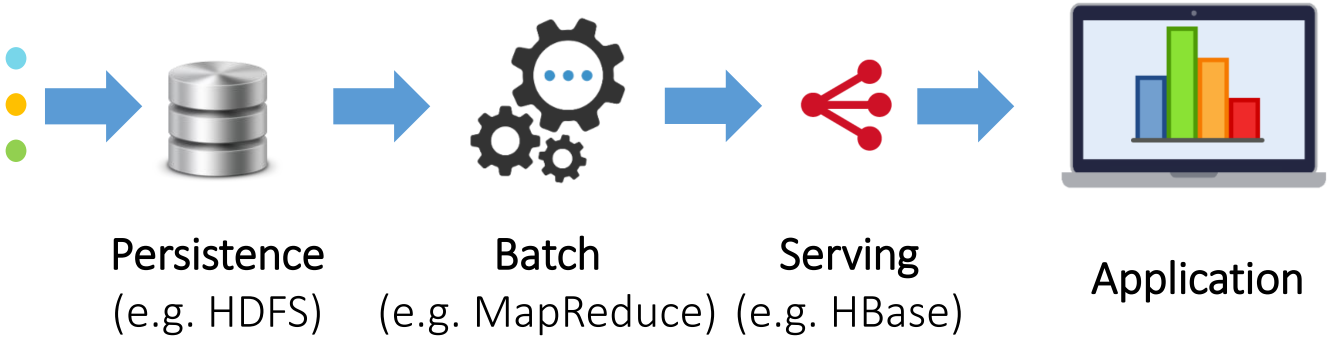
low latency ⟷ high throughput

Batch Processing: „Volume“

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

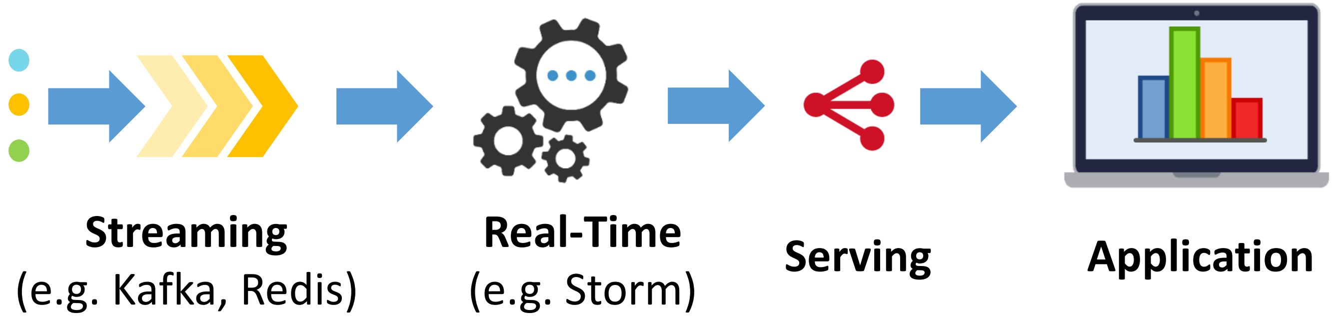
But:

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



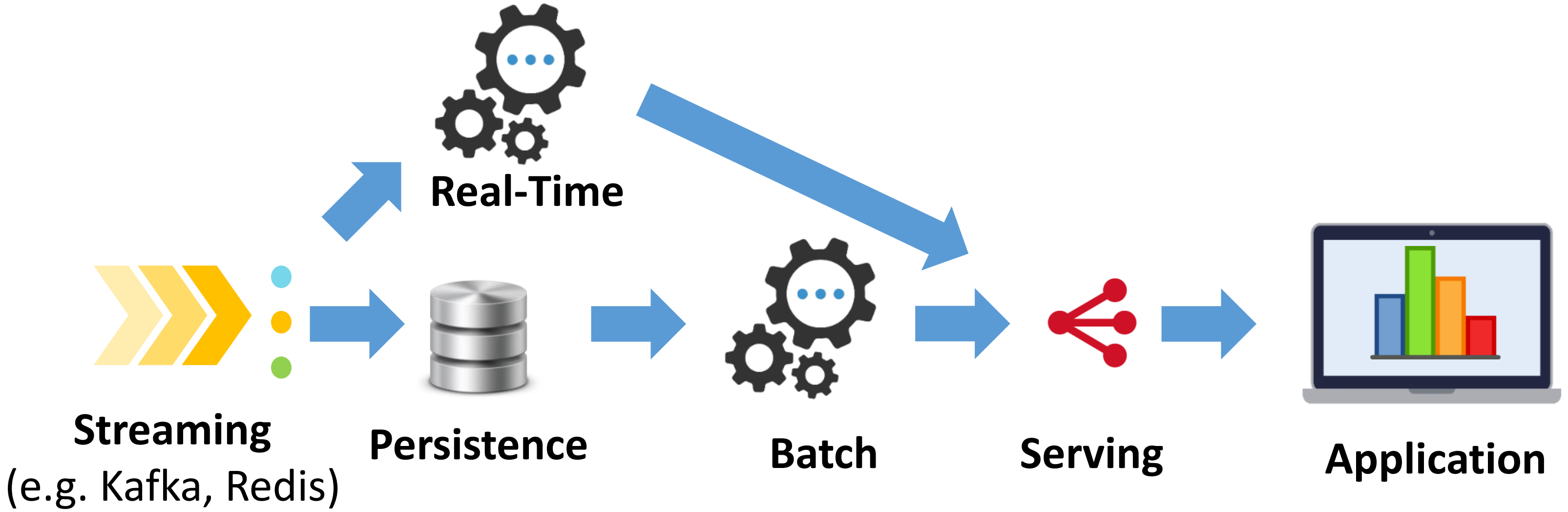
Stream Processing: „Velocity“

- 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, ...



Lambda Architecture: $Batch(D_{old}) + Stream(D_{\Delta now}) \approx Batch(D_{all})$

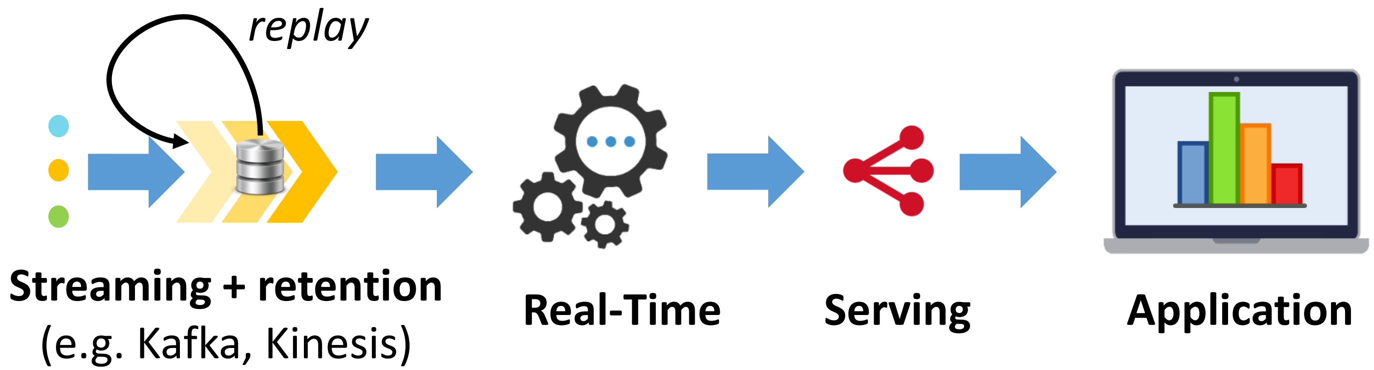
- **Fast** output (real-time)
- Data retention + reprocessing (batch)
  → **„eventually accurate"** merged views of real-time & batch
  Typical setups: Hadoop + Storm (→ Summingbird), Spark, Flink
- High complexity 2 code bases & 2 deployments



Nathan Marz: How to Beat the CAP Theorem, Thoughts From the Red Planet (2011)

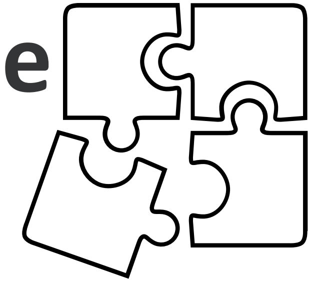
Kappa Architecture: Stream(D_{all}) = Batch(D_{all})

- **Simpler** than Lambda architecture
- **Data retention** for history
- Reasons against Kappa:
  - Existing **legacy batch system**
  - **Special tools** only for a particular batch processor
  - Only **incremental** algorithms



Jay Kreps: Questioning the Lambda Architecture, O'Reilly Blog (2014)

Intermediate Wrapup: Data Processing at Scale



- 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

- **Lambda Architecture**: batch + stream processing
- **Kappa Architecture**: stream-only processing

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

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, ...

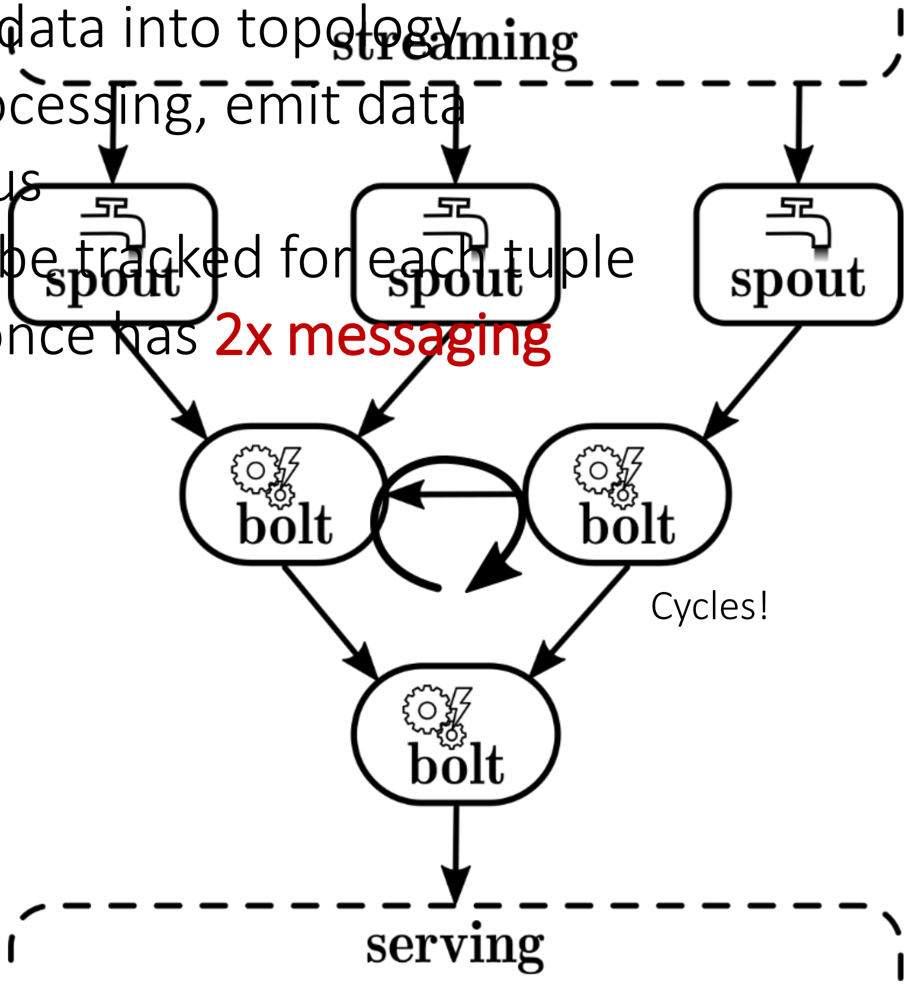
History

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

Data Flow

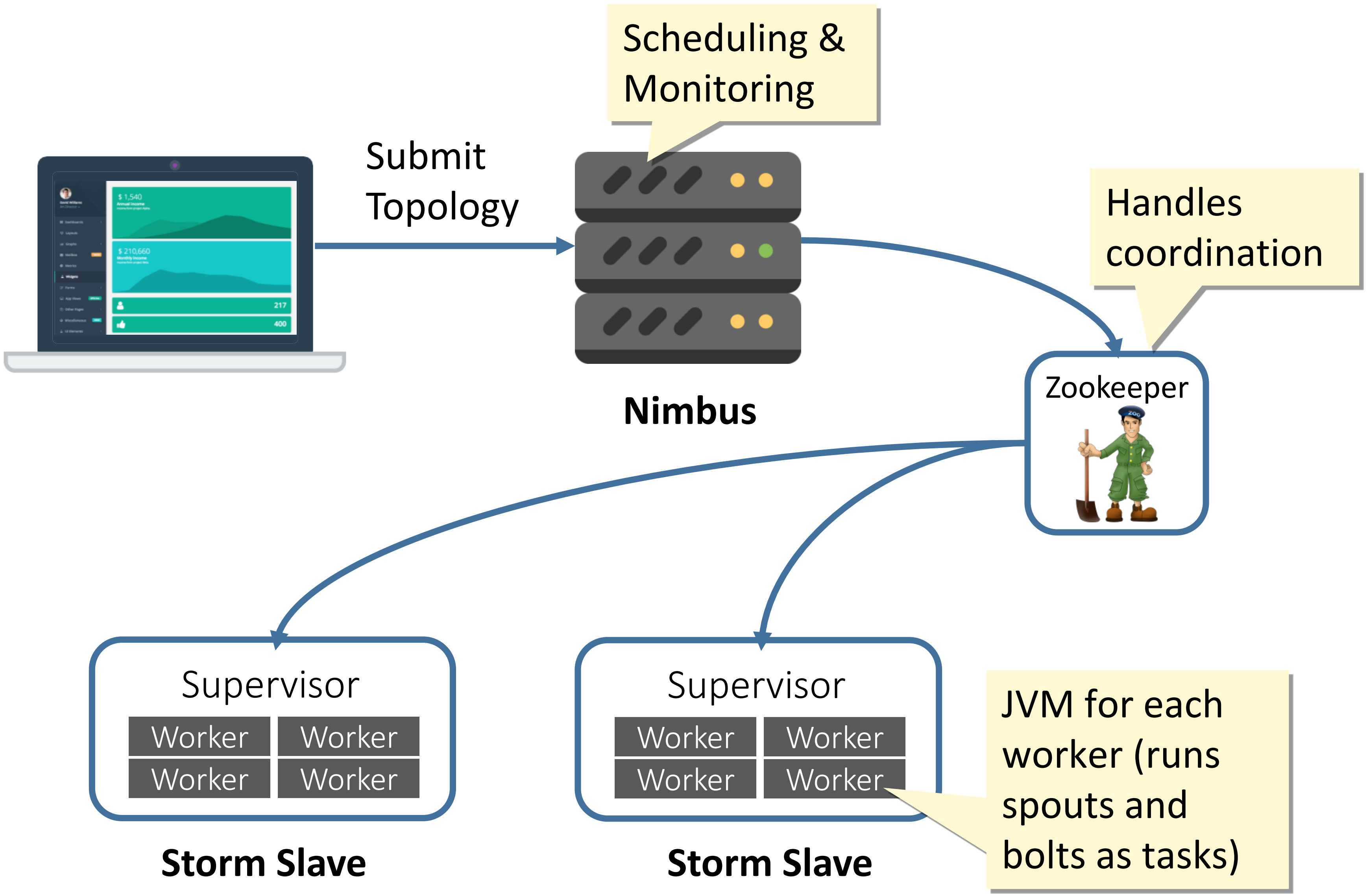
Directed Acyclic Graphs (DAG):

- **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**



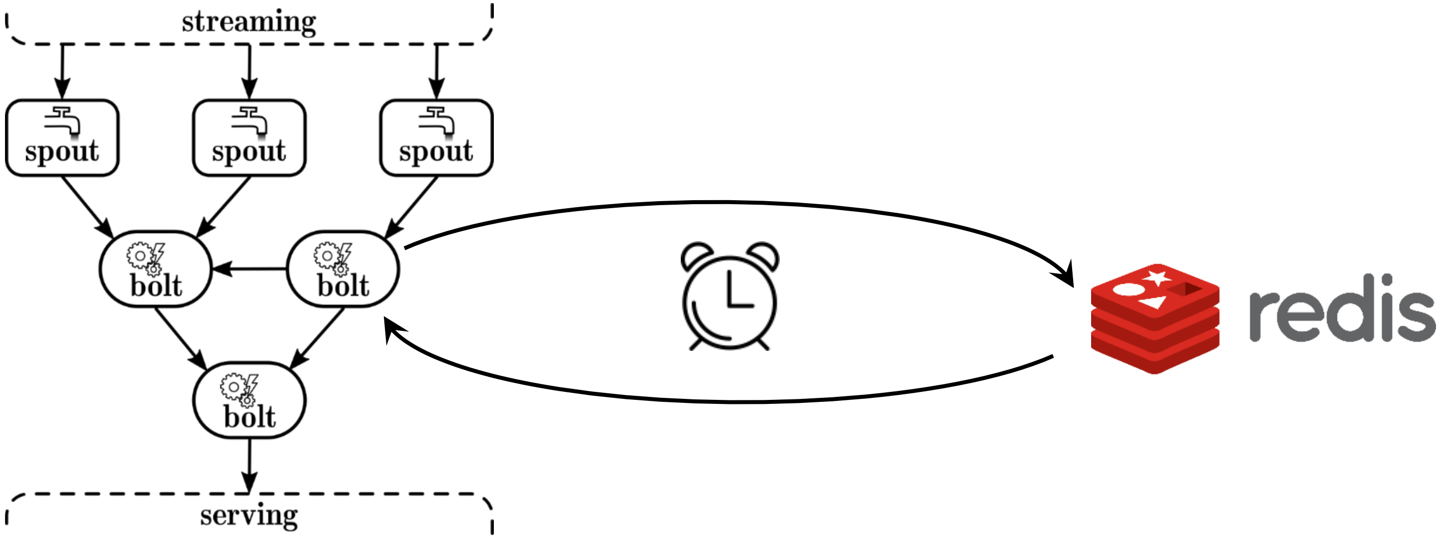
Cycles!

Cluster Architecture: How Storm Scales



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

1. too many tuples → **2.** tuples time out and fail → **3.** tuples get replayed → **1.**

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 & scalability bottleneck

Trident: Partitioned Micro-Batching

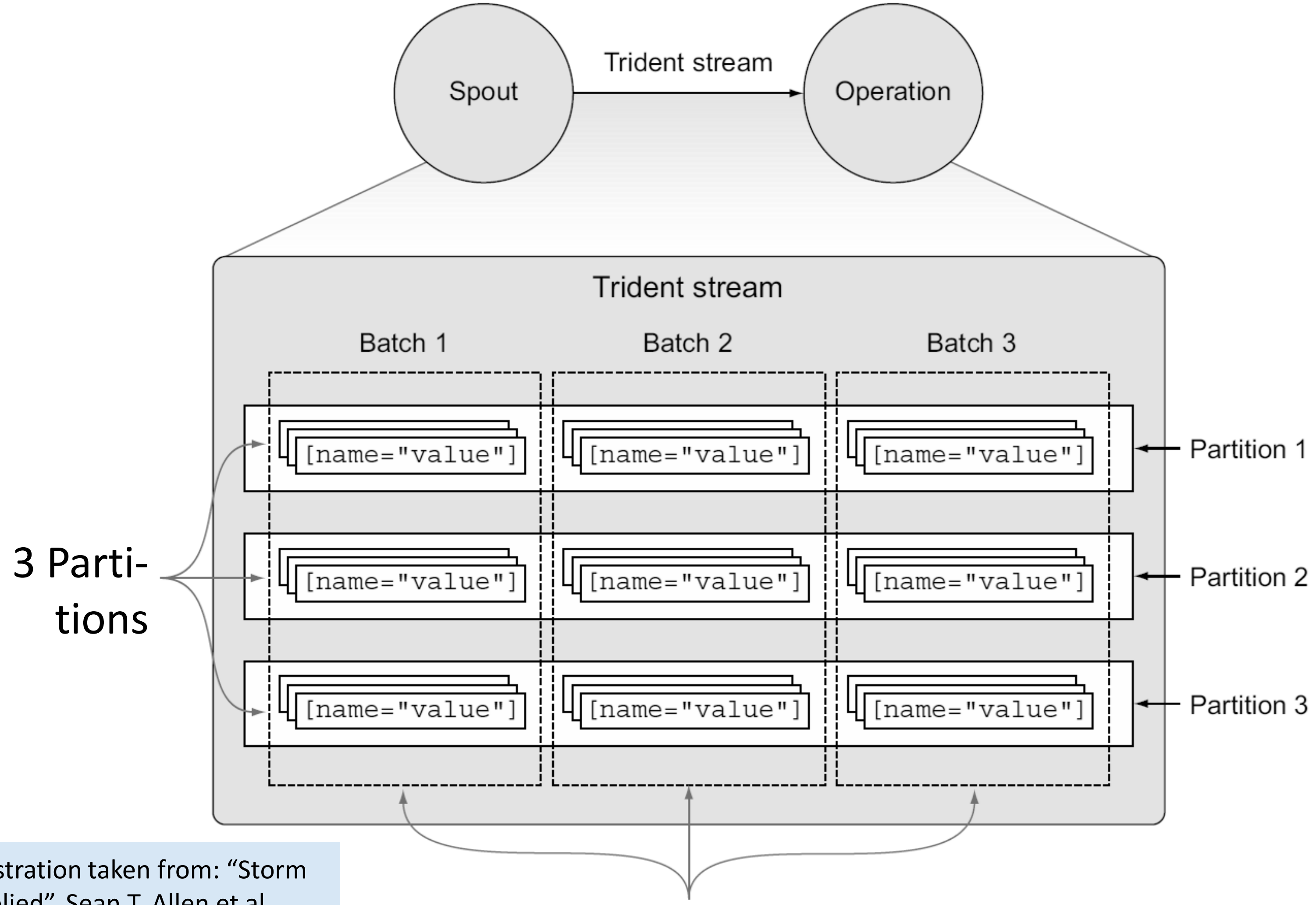


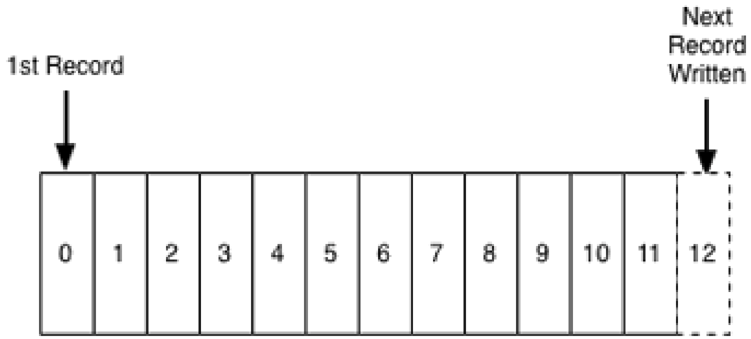
Illustration taken from: "Storm applied", Sean T. Allen et al.

Samza: Real-Time on Top of Kafka

samza

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

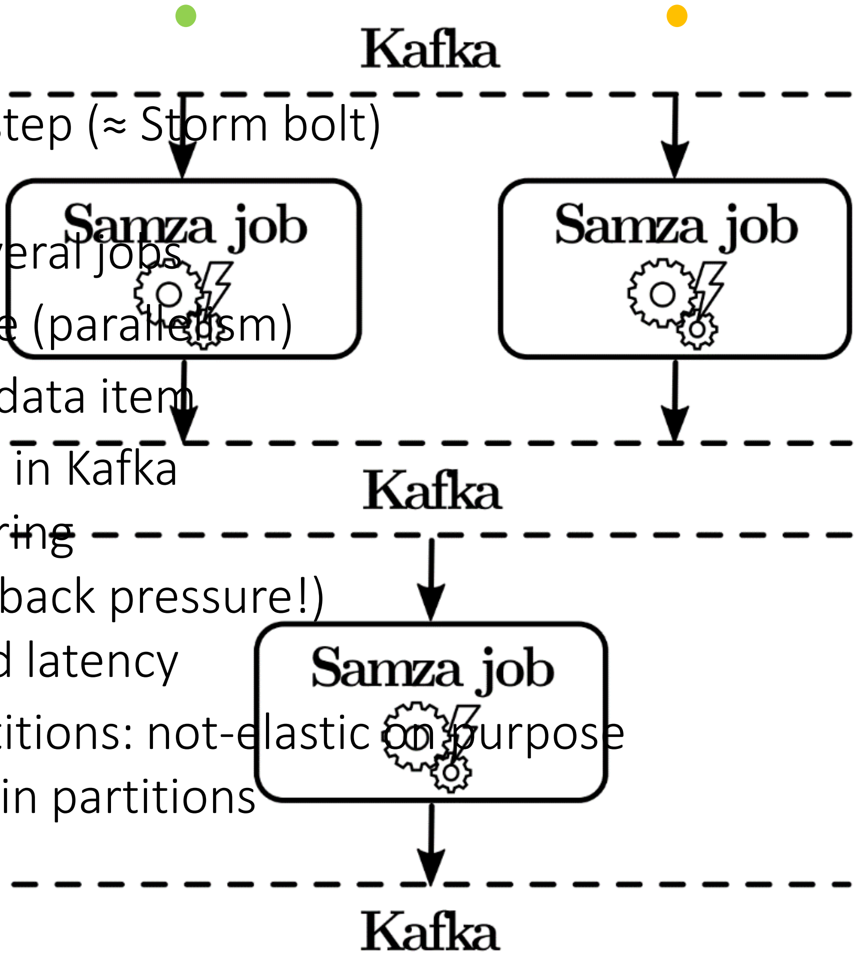
- Developed at **LinkedIn**
- **2013**: open-source (Apache Incubator)
- **2015**: Apache top-level project

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

Data Flow: Simple By Design

samza

- **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
- Task = Kafka partitions: not-elastic on purpose
  → **Ordering** within partitions



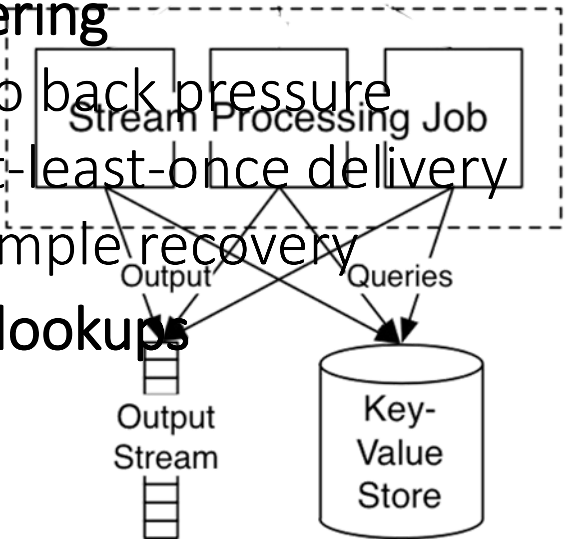
Martin Kleppmann, *Turning the database inside-out with Apache Samza* (2015)
https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/ (2017-02-23)

Samza: Local State

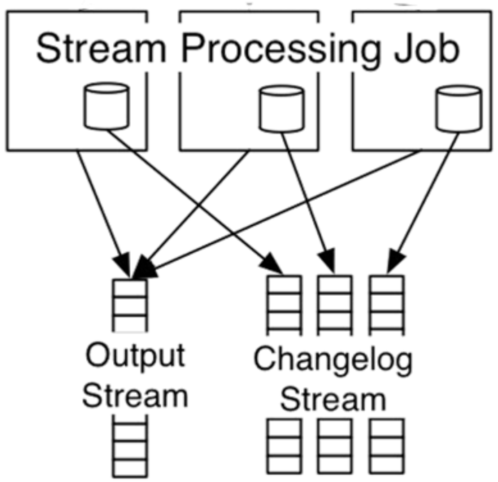
samza

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)

Data Flow Example: Enriching a Clickstream

samza

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

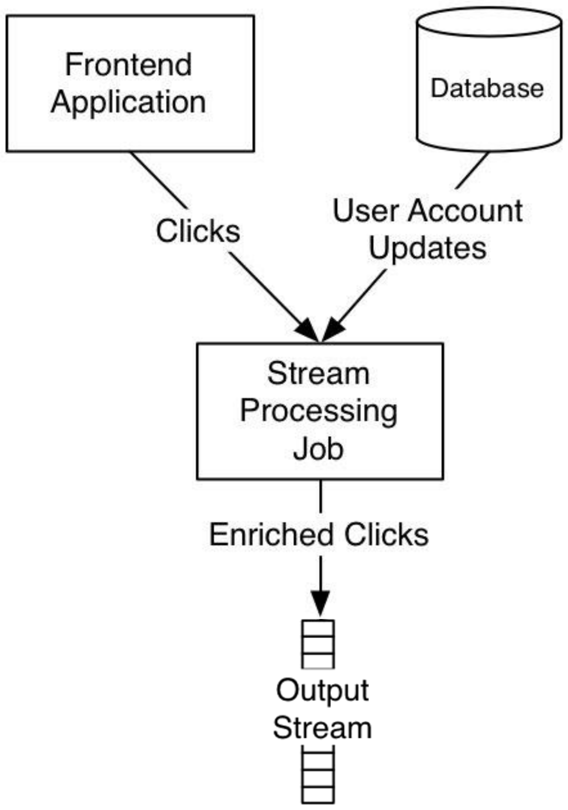


Illustration taken from: Jay Kreps, *Why local state is a fundamental primitive in stream processing* (2014)
https://www.oreilly.com/ideas/why-local-state-is-a-fundamental-primitive-in-stream-processing (2017-02-26)

State Management: Straightforward Recovery

samza

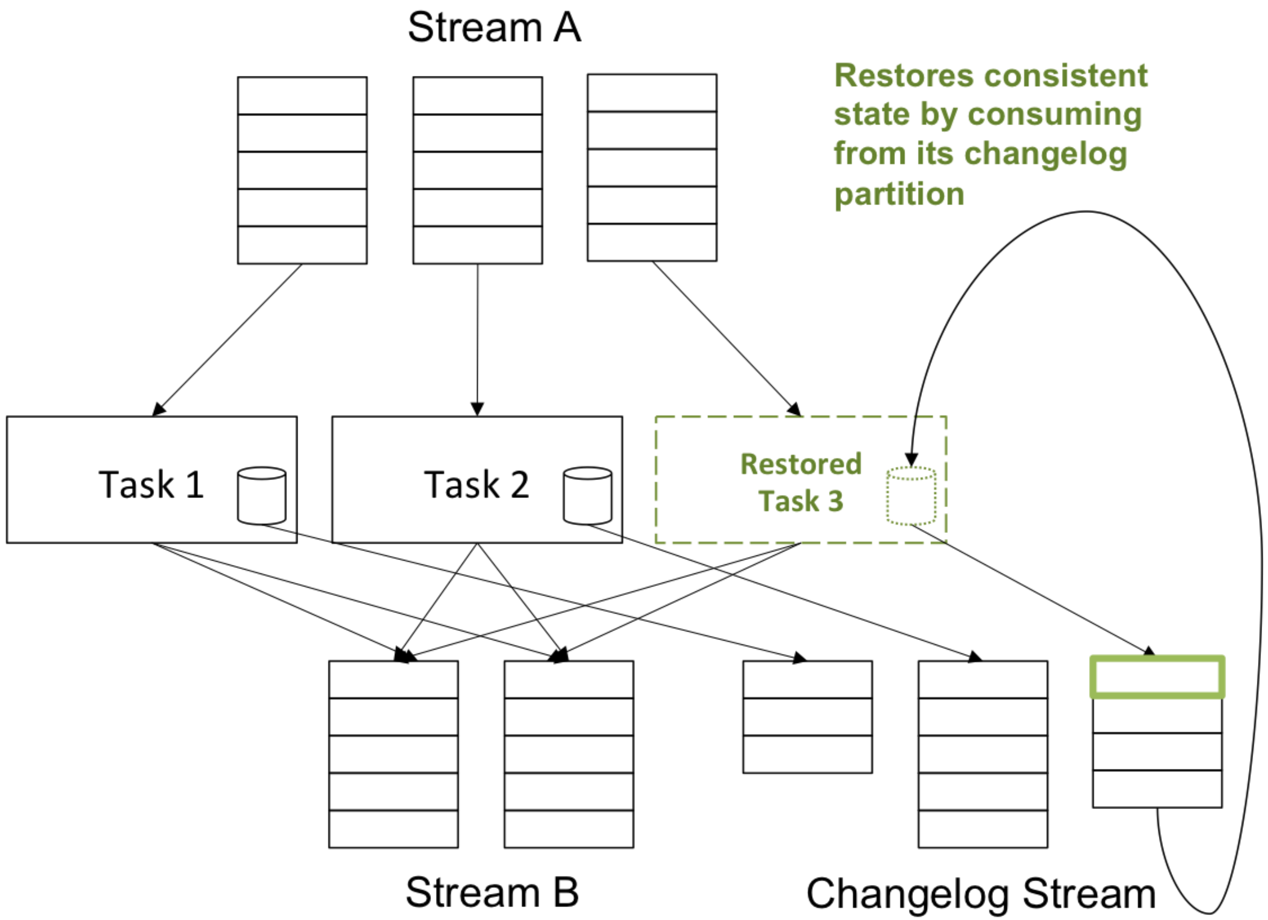


Illustration taken from: Navina Ramesh, *Apache Samza, LinkedIn's Framework for Stream Processing* (2015)
https://thenewstack.io/apache-samza-linkedins-framework-for-stream-processing (2017-02-26)

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

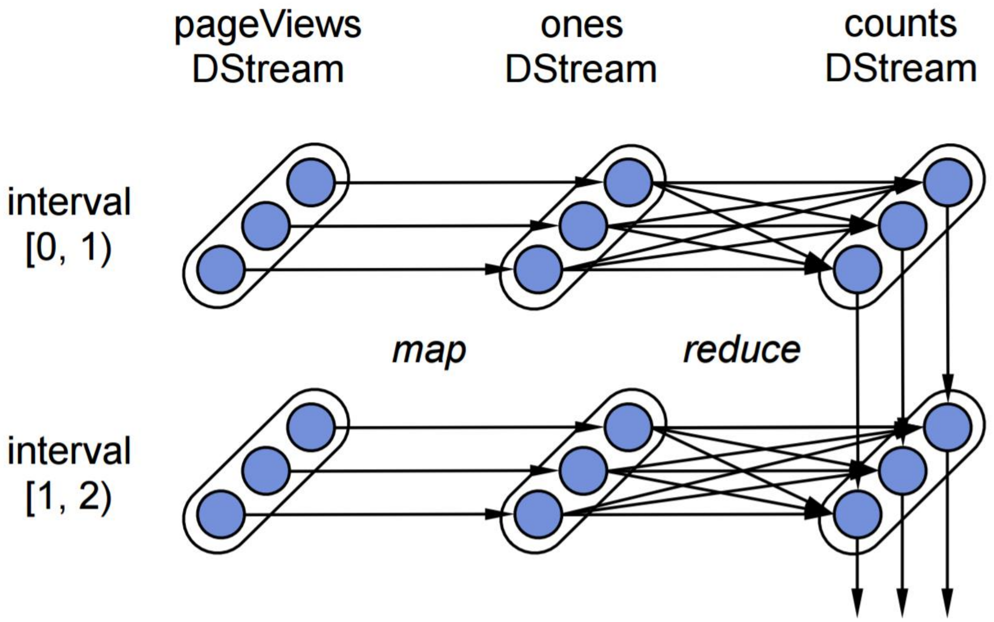
DStream: Discretized RDD

- **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

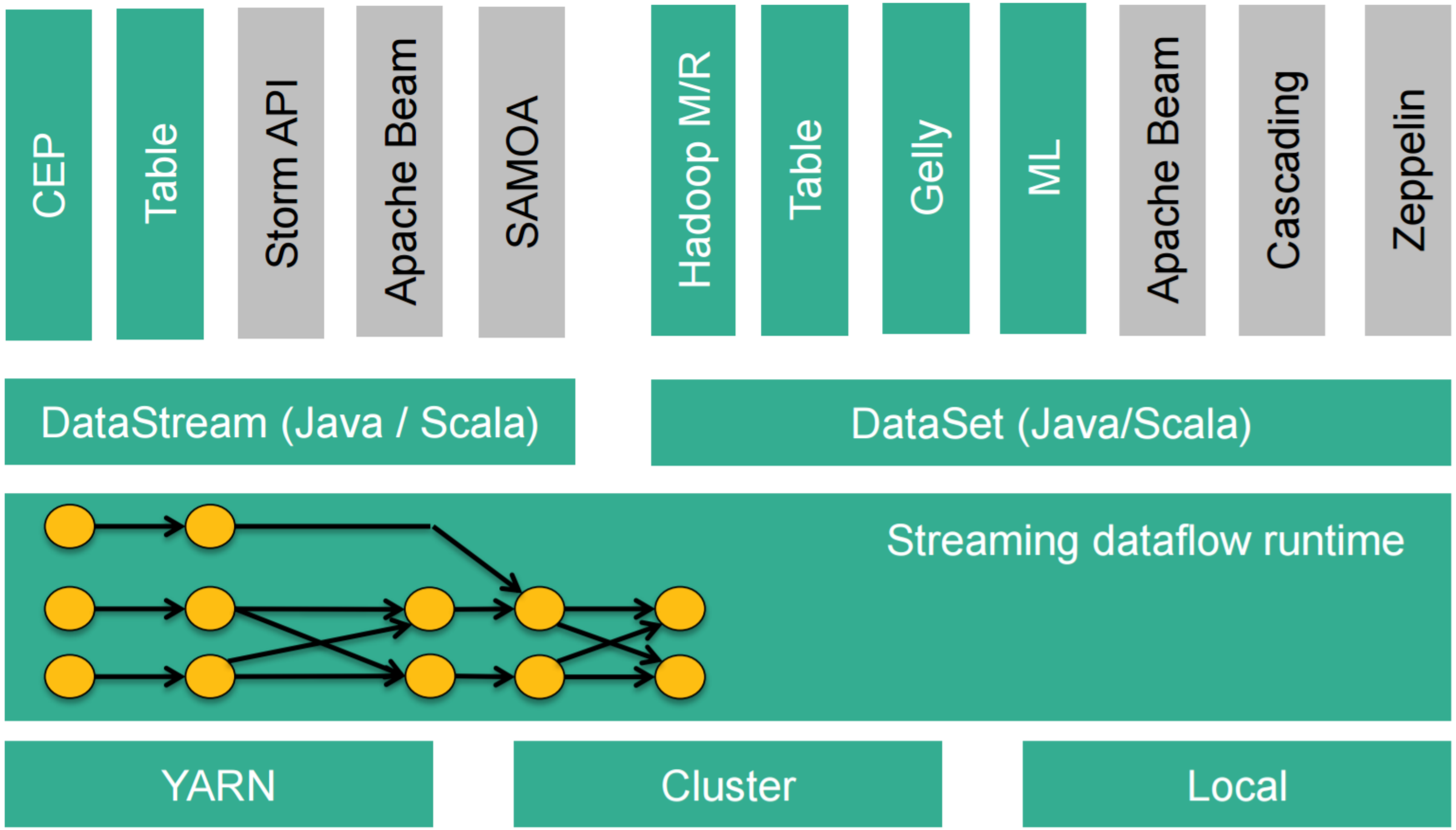
Overview

- **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

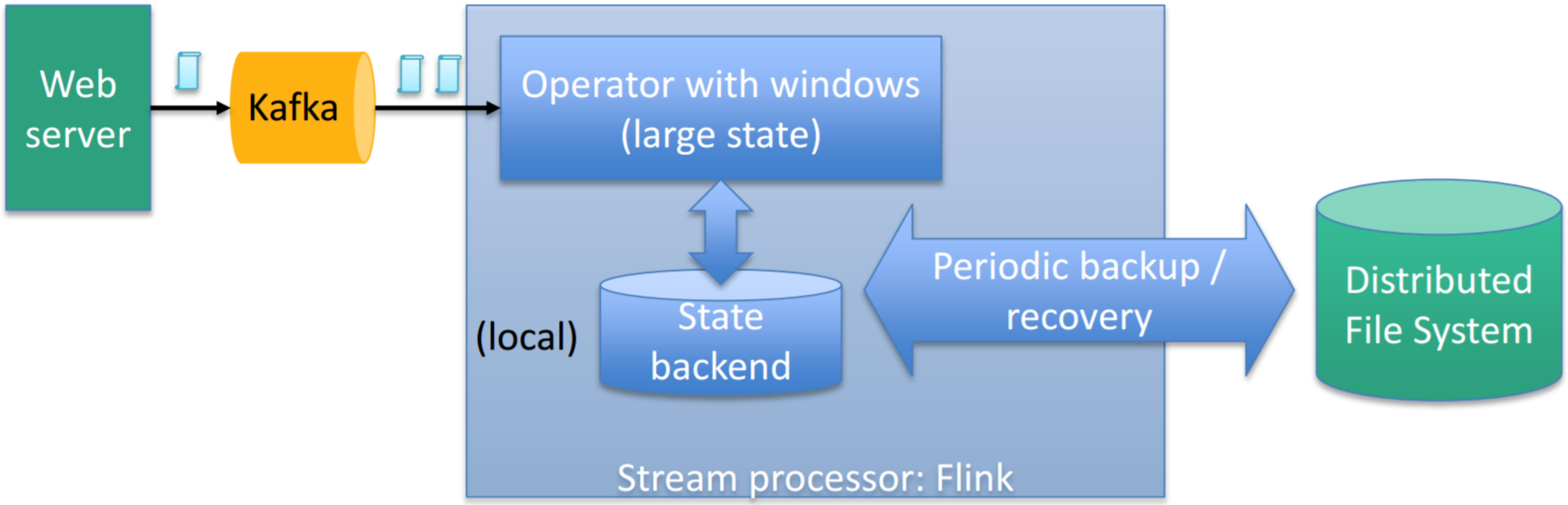
Architecture: Streaming + Batch



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

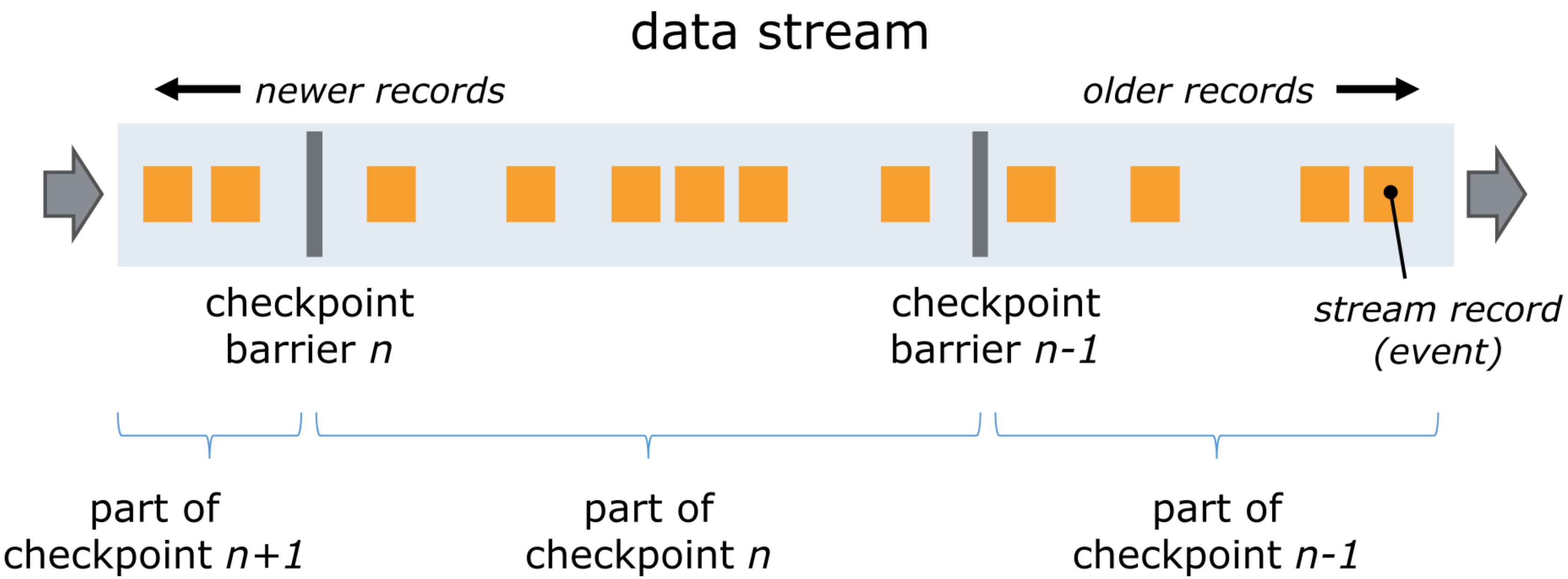
State Management

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



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

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)

System 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

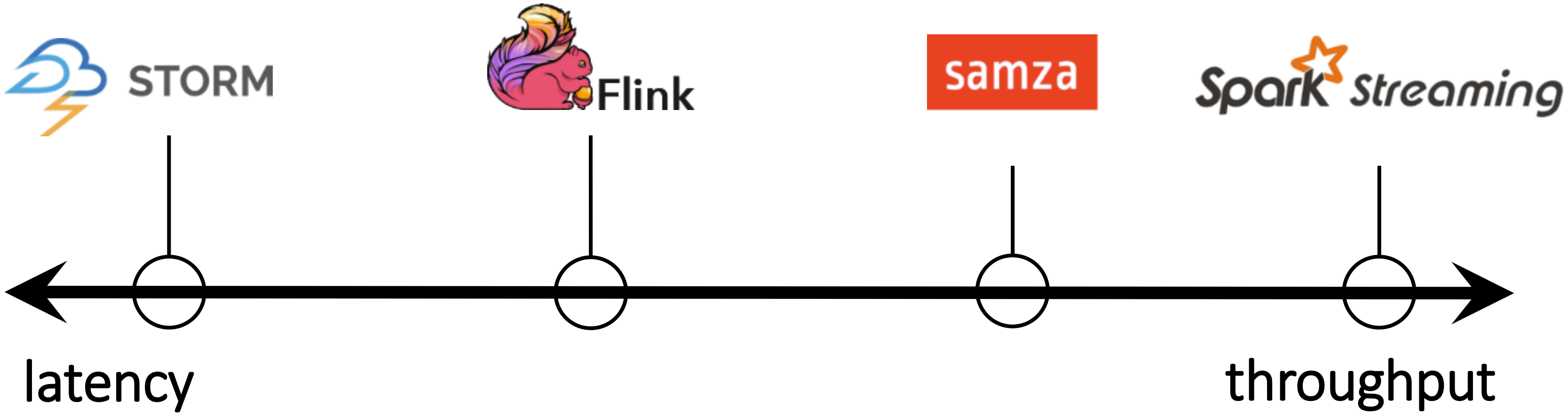
Kafka Streams

IBM InfoSphere Streams

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

Wrapup: Continuous Data Processing at Scale

- Stream Processors:



- **Many Dimensions of Interest**: consistency guarantees, state management, backpressure, ordering, elasticity, …

Part 2: Data Validation at Scale

1. **The Importance of Data Validation**

   Where is data validation integrated into data science pipelines and what is its impact?

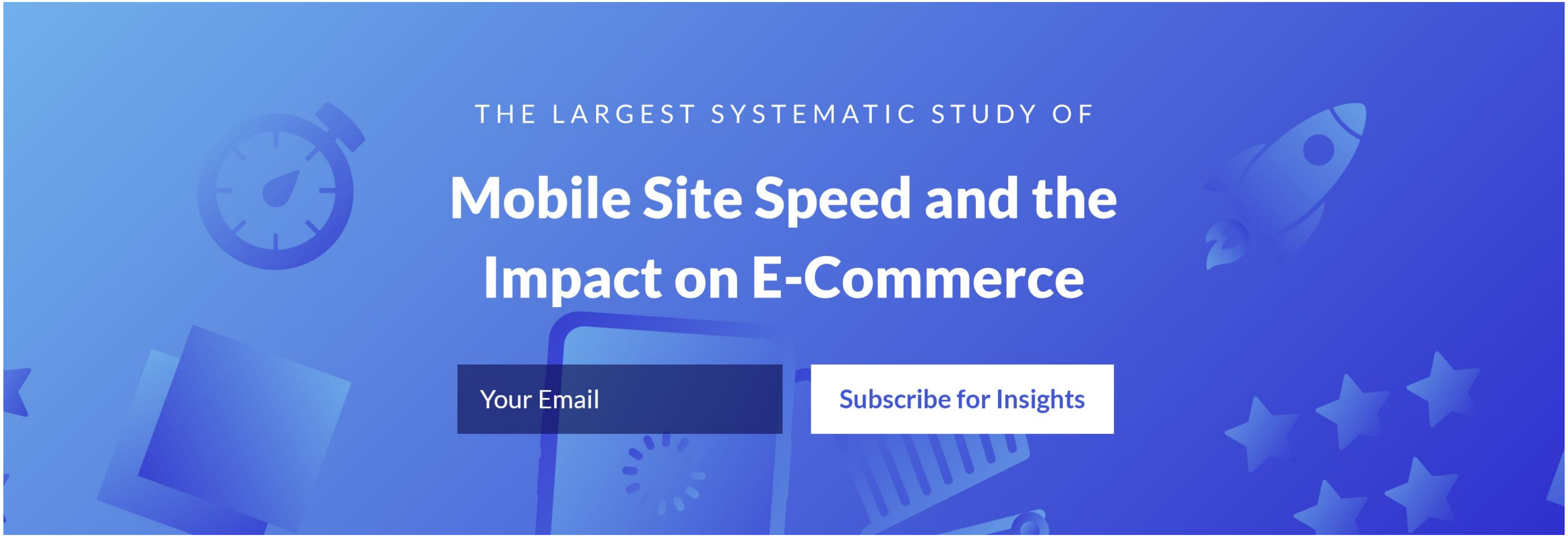
2. **Data Quality & Constraints**

   What dimensions of data quality are there and how can they be ensured?

3. **Scalability-Related Challenges**

   Why is data validation difficult in data-intensive domains?

How and Why Acceleration: speedhub.org



Google

Baqend

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

W. Wingerath, B. Wollmer, F. Gessert, S. Succo, N. Ritter. Going for Speed: Full-Stack Performance Engineering in Modern Web-Based Applications, WWW 2021 (Tutorial)

Split Testing for Web Performance

Speed Kit Users

vs.

Normal Users

Tracking →

Tracking ←

- Speed Kit enabled

- **Measurable uplift**:
  + Performance
  + User engagement
  + Revenue
  …

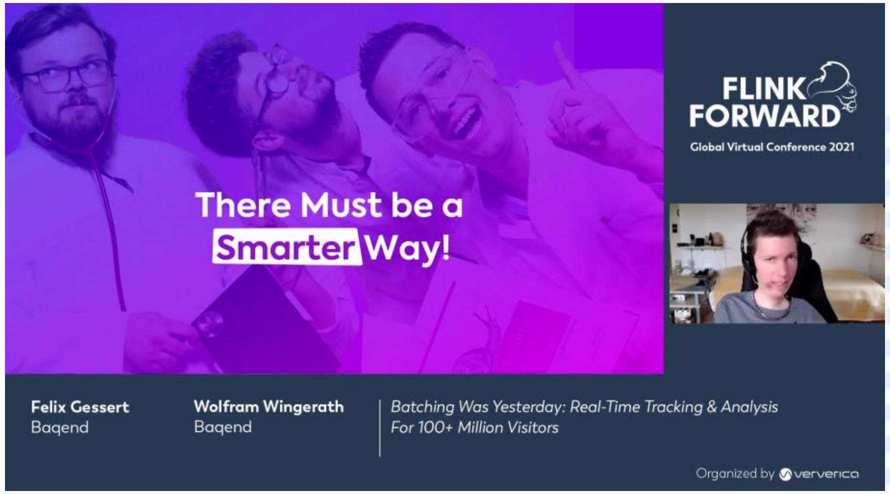
- Speed Kit disabled (no acceleration)

W. Wingerath, F. Gessert, E. Witt, H. Kuhlmann, F. Bücklers, B. Wollmer, N. Ritter. Speed Kit: A Polyglot & GDPR-Compliant Approach For Caching Personalized Content, ICDE 2020

Running Example: Web Performance Analysis Pipeline

Collection | Ingestion | Analytics | Reporting

Tracking (RUM)

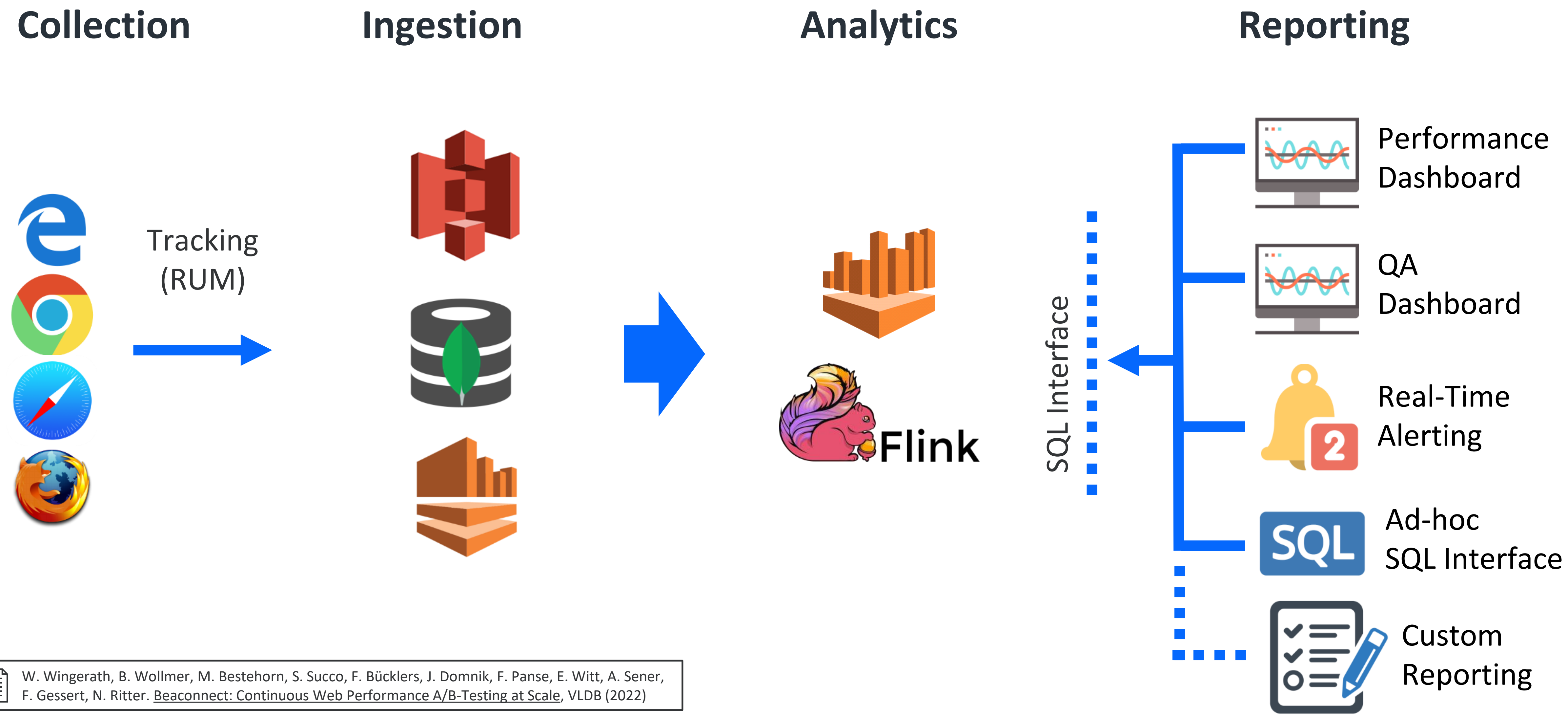


F. Gessert, W. Wingerath. Batching Was Yesterday: Real-Time Tracking & Analysis For 100+ Million Visitors, Flink Forward (2021)

- Performance Dashboard
- QA Dashboard
- Real-Time Alerting
- Ad-hoc SQL Interface
- Custom Reporting

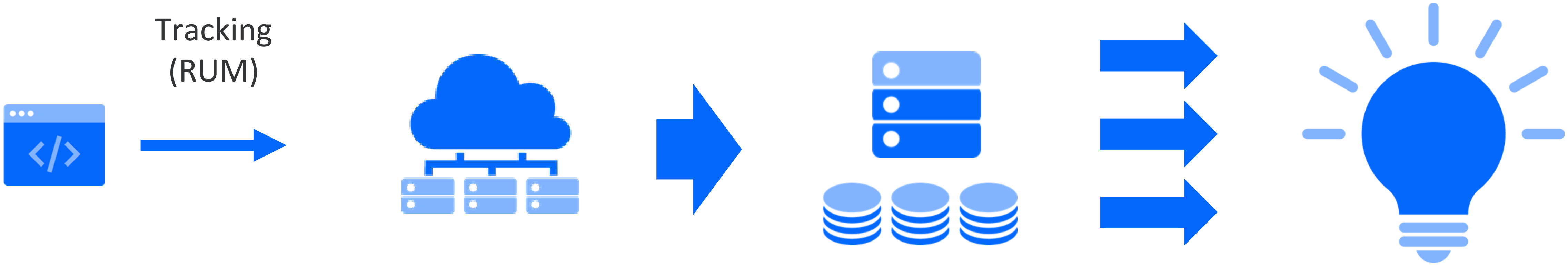
W. Wingerath, B. Wollmer, M. Bestehorn, S. Succo, F. Bücklers, J. Domnik, F. Panse, E. Witt, A. Sener, F. Gessert, N. Ritter. Beaconnect: Continuous Web Performance A/B-Testing at Scale, VLDB (2022)

Running Example: Web Performance Analysis Pipeline

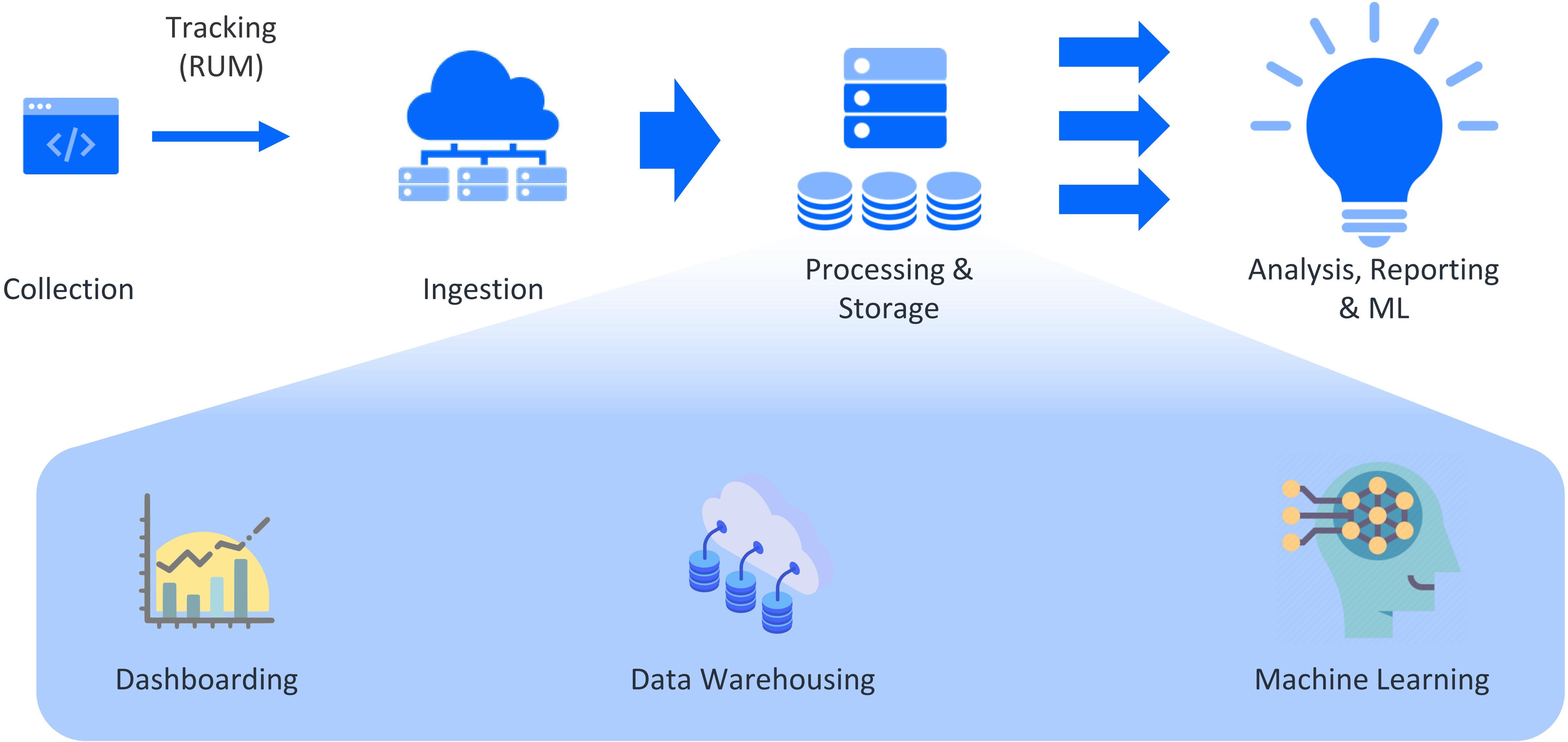


W. Wingerath, B. Wollmer, M. Bestehorn, S. Succo, F. Bücklers, J. Domnik, F. Panse, E. Witt, A. Sener, F. Gessert, N. Ritter. Beaconnect: Continuous Web Performance A/B-Testing at Scale, VLDB (2022)

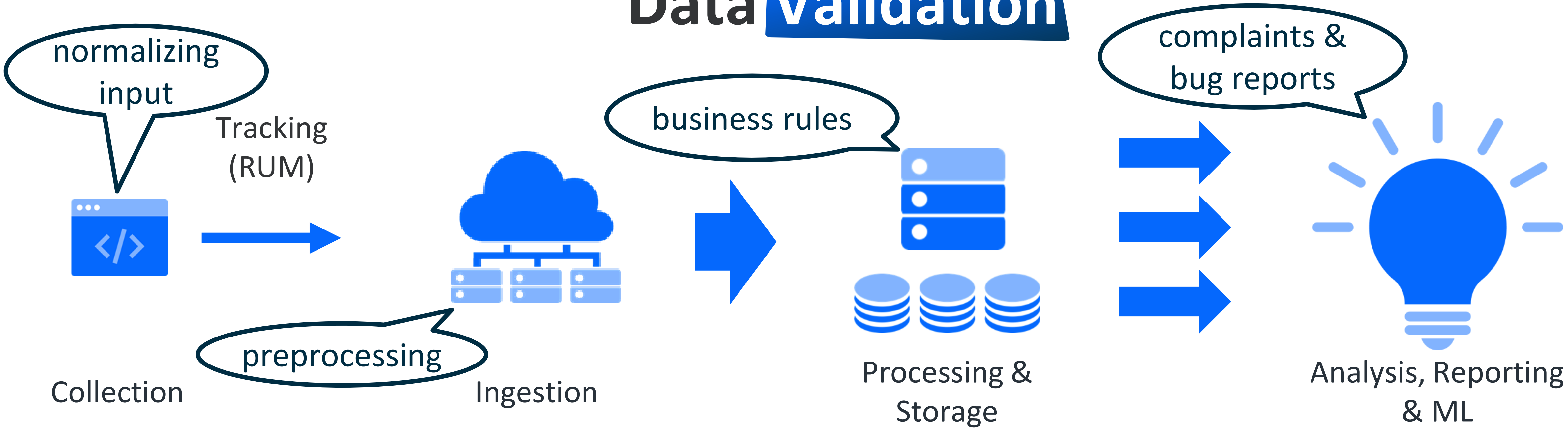
Running Example: Web Performance Analysis Pipeline



Running Example: Web Performance Analysis Pipeline

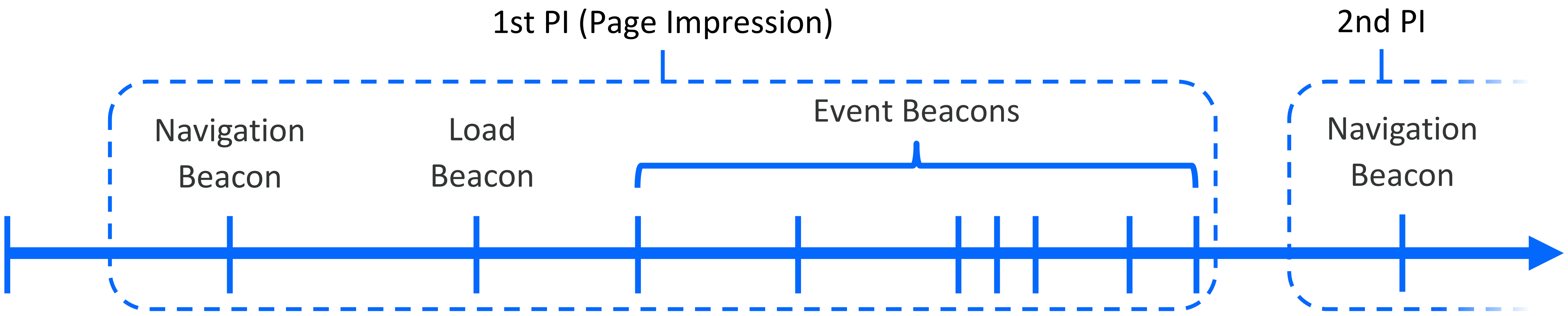


Data Validation



- **Goal**: verify that data in the pipeline is in an acceptable state for downstream processing, e.g.
  - External reporting (statistics, visualizations & dashboarding)
  - Internal reporting (debugging, product optimizations)
  - Decision-making (analytics, machine learning)
- Data validation can be integrated **in and between all stages**

Dimensions of Data Quality: Completeness



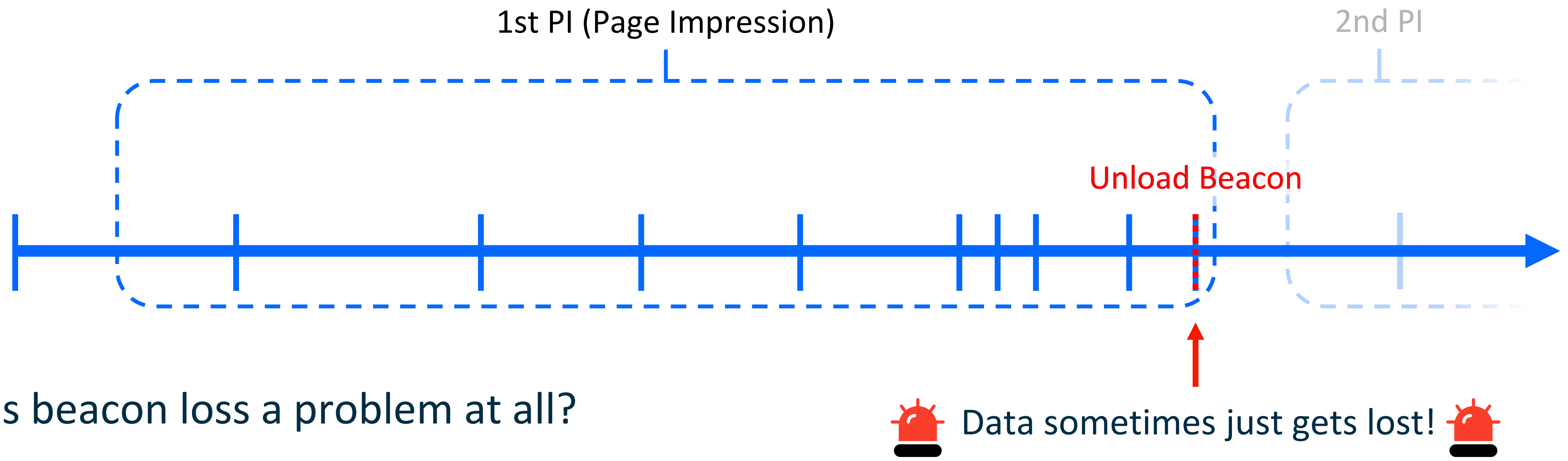
- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

Dimensions of Data Quality: Completeness

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	"670ms"	123	null
13:26:48.139	9cddf7	Firefox	692 654	456	abc.de/red
13:28:23.857	0b577a	Firefox	0.256	456	abc.de/blue
13:29:17.468	faf55e	Edge	1.598	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

Unload Beacon **Reliability** as an Example Challenge



- Is beacon loss a problem at all?
- **When** is beacon loss a problem?

  → For which beacon types? For which beacon strategies?
- **Where** is beacon loss a problem?

  → For which browsers? For which device types?

Unload Beacon **Reliability** by Strategy

```
addEventListener("unload", (event) => {
  navigator.sendBeacon(url, JSON.stringify(data));
});
```

```
addEventListener("beforeunload", (event) => {
  navigator.sendBeacon(url, JSON.stringify(data));
});
```

```
addEventListener("pagehide", (event) => {
  navigator.sendBeacon(url, JSON.stringify(data));
});
```

```
addEventListener("visibilitychange", (event) => {
  if (document.visibilityState === 'hidden') {
    navigator.sendBeacon(url, JSON.stringify(data));
  }
});
```

available on all platforms

experimental feature

(only available as origin trial in Chrome)

```
var beacon = new window.PendingPostBeacon(
  url,
  {
    timeout: 60000,
    backgroundTimeout: 0
  });
beacon.setData(JSON.stringify(data));
```

Erik Witt. Unload Beacon Reliability: Benchmarking Strategies for Minimal Data Loss, Speed Kit Tech Blog (2023)

Unload Beacon Reliability by Strategy

```
addEventListener("unload", (event) => {
  navigator.sendBeacon(url, JSON.stringify(data));
});
```

Available on all platforms

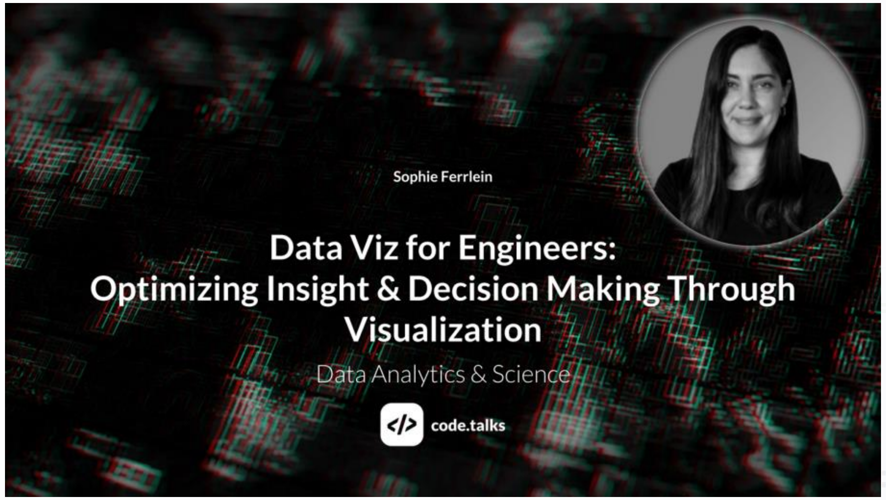
```
addEventListener("befor
  navigator.sendBeacon(
});
```

```
addEventListener("pageh
  navigator.sendBeacon(
});
```

```
addEventListener("visibilityc
  if (document.visibilityStat
    navigator.sendBeacon(url, JSON.stringify(data));
  }
});
```

ntly only available as origin trial in Chrome

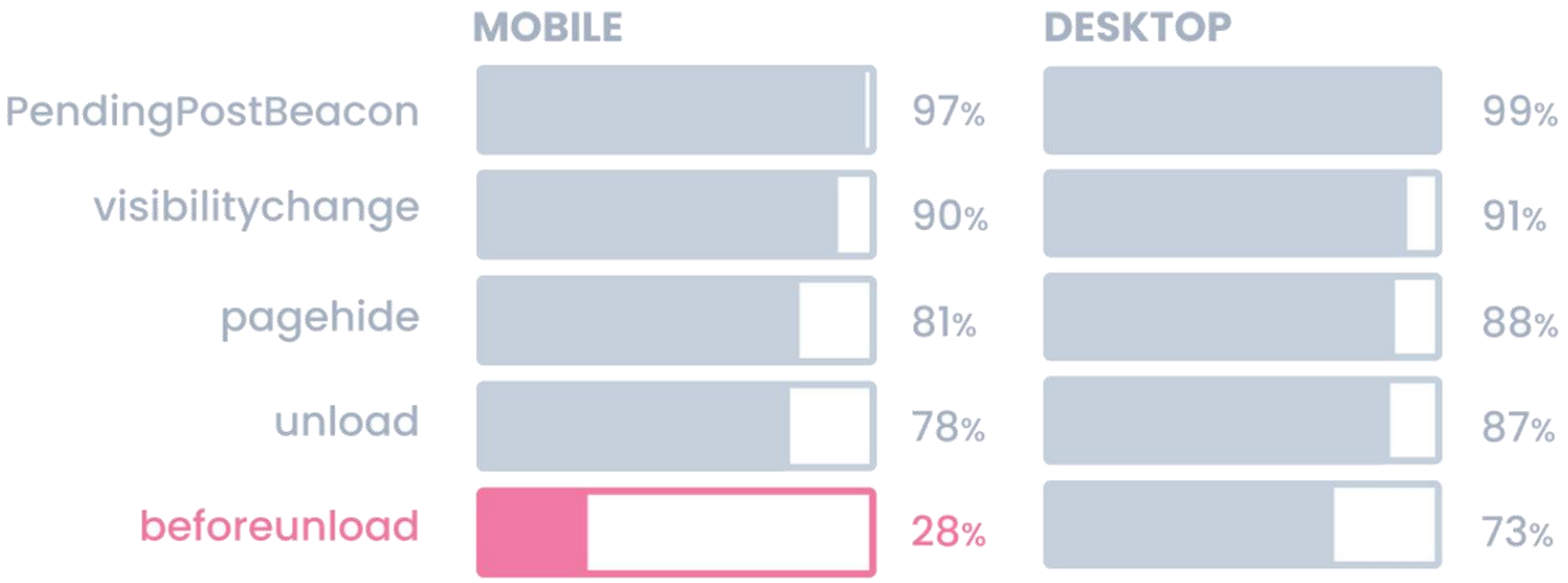
```
= new window.PendingPostBeacon(

    60000,
  ndTimeout: 0
  });
beacon.setData(JSON.stringify(data));
```



Sophie Ferrlein. Data Viz for Engineers: Optimizing Insight & Decision Making Through Visualization, code.talks (2023)

Erik Witt. Unload Beacon Reliability: Benchmarking Strategies for Minimal Data Loss, Speed Kit Tech Blog (2023)

Unload Beacon Reliability by Strategy & Device

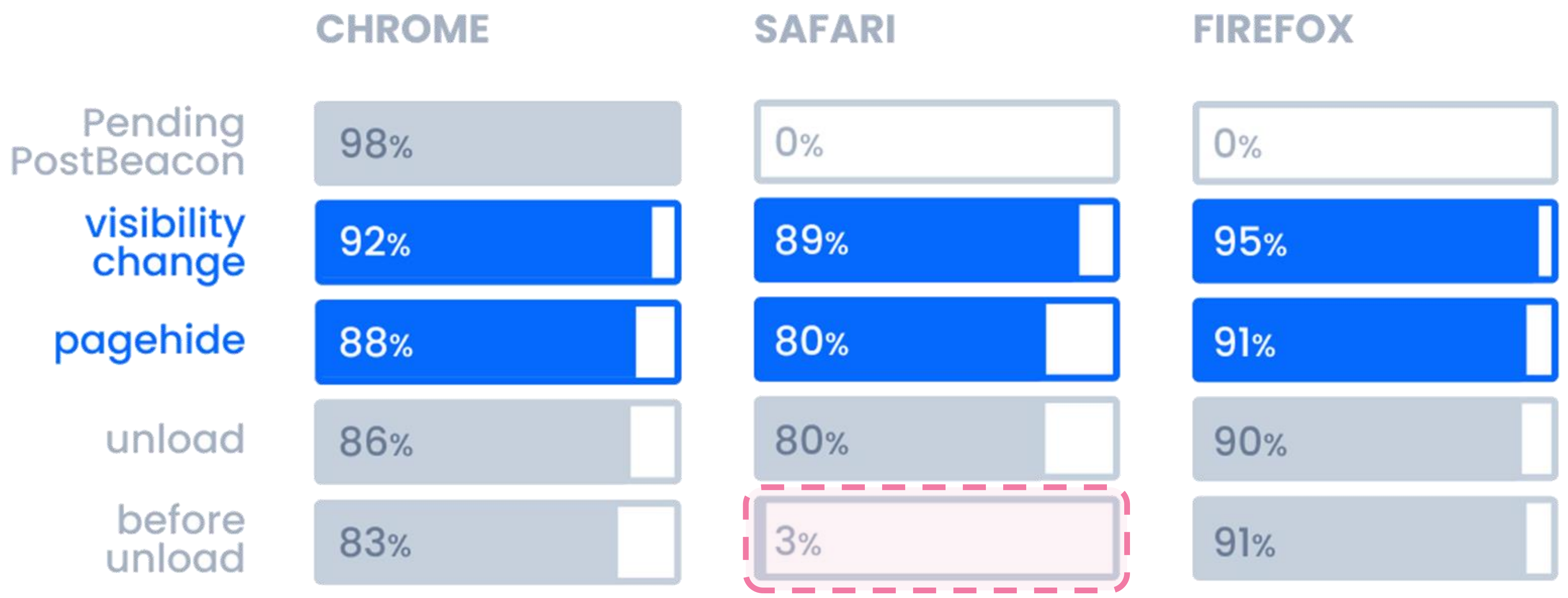


Based on 52 million page views on a globally operating e-commerce site measured by Speed Kits real user-monitoring | August 2023

Desktop data may be overrepresented!

Erik Witt. Unload Beacon Reliability: Benchmarking Strategies for Minimal Data Loss, Speed Kit Tech Blog (2023)

Unload Beacon Reliability by Strategy & Browser



Based on 52 million page views on a globally operating e-commerce site measured by Speed Kits real user-monitoring | August 2023

Erik Witt. Unload Beacon Reliability: Benchmarking Strategies for Minimal Data Loss, Speed Kit Tech Blog (2023)

Unload Beacon Reliability: The Ideal Combo Strategy

Can be used with Backward-Forward Cache?

Based on 52 million page views on a globally operating e-commerce site measured by Speed Kits real user-monitoring | August 2023

Erik Witt. Unload Beacon Reliability: Benchmarking Strategies for Minimal Data Loss, Speed Kit Tech Blog (2023)

Dimensions of Data Quality: Consistency

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	"670ms"	123	null
13:26:48.139	9cddf7	Firefox	692 654	456	abc.de/red
13:28:23.857	0b577a	Firefox	0.256	456	abc.de/blue
13:29:17.468	faf55e	Edge	1.598	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

Same browser

Same session

Note: Browser values should be unified for all records in the same session!

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

Dimensions of Data Quality: Consistency

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	"670ms"	123	null
13:26:48.139	9cddf7	Firefox	692 654	456	abc.de/red
13:28:23.857	0b577a	Firefox	0.256	456	abc.de/blue
13:29:17.468	faf55e	**Firefox**	1.598	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

Note: Browser values should be unified for all records in the same session!
→ value may be replaced with majority vote (another reasonable option: replace old values with latest one)

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

Dimensions of Data Quality: Consistency

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	"670ms"	123	null
13:26:48.139	9cddf7	Firefox	692 654	456	abc.de/red
13:28:23.857	0b577a	Firefox	0.256	456	abc.de/blue
13:29:17.468	faf55e	**Firefox**	1.598	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

Note: All values represent milliseconds, but formats differ depending on browser.

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - Consistency: Does data have a valid format and does it comply with business semantics?
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

Dimensions of Data Quality: Consistency

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	**670**	123	null
13:26:48.139	9cddf7	Firefox	**692 654**	456	abc.de/red
13:28:23.857	0b577a	Firefox	**256**	456	abc.de/blue
13:29:17.468	faf55e	**Firefox**	**1598**	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

Note: All values represent milliseconds, but formats differ depending on browser.
→ values may be converted to integer

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - Consistency: Does data have a valid format and does it comply with business semantics?
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

Dimensions of Data Quality: Accuracy

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	**670**	123	null
13:26:48.139	9cddf7	Firefox	**692 654**	456	abc.de/red
13:28:23.857	0b577a	Firefox	**256**	456	abc.de/blue
13:29:17.468	faf55e	**Firefox**	**1598**	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

Note: Despite being in the right format, one value does not represent a reasonable timer value.

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - Accuracy: Do data items represent their corresponding real-world entities well?
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

Dimensions of Data Quality: Accuracy

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	**670**	123	null
13:26:48.139	9cddf7	Firefox	**null**	456	abc.de/red
13:28:23.857	0b577a	Firefox	**256**	456	abc.de/blue
13:29:17.468	faf55e	**Firefox**	**1598**	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

Note: Despite being in the right format, one value does not represent a reasonable timer value.
→ broken value may be removed

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - Accuracy: Do data items represent their corresponding real-world entities well?
  - *Uniqueness: Are duplicate records known and are all unique attributes actually distinct?*

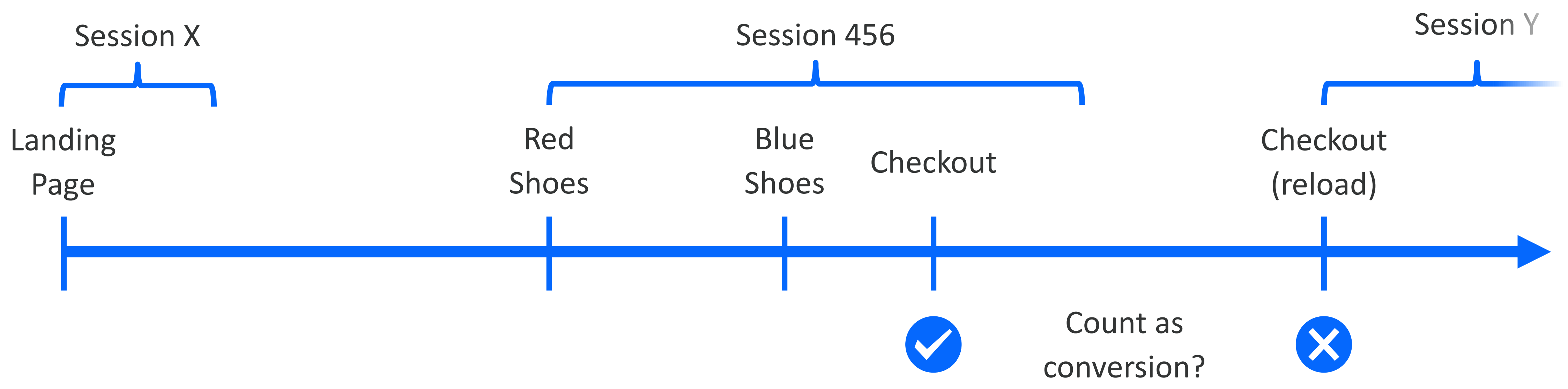
Dimensions of Data Quality: Uniqueness

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	**670**	123	null
13:26:48.139	9cddf7	Firefox	**null**	456	abc.de/red
13:28:23.857	0b577a	Firefox	**256**	456	abc.de/blue
13:29:17.468	faf55e	**Firefox**	**1598**	456	abc.de/sold
20:45:38.941	faf55e	null	null	null	abc.de/sold

Note: The ID field should be unique, but two different records share the same value!

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - Uniqueness: Are duplicate records known and are all unique attributes actually distinct?

Dimensions of Data Quality: Uniqueness



- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - Uniqueness: Are duplicate records known and are all unique attributes actually distinct?

Dimensions of Data Quality: Uniqueness

Timestamp.	Pageload ID	Browser	LCP (Performance)	Session ID	URL
09:05:04.578	37ab08	Edge	**670**	123	null
13:26:48.139	9cddf7	Firefox	**null**	456	abc.de/red
13:28:23.857	0b577a	Firefox	**256**	456	abc.de/blue
13:29:17.468	faf55e	**Firefox**	**1598**	456	abc.de/sold
~~20:45:38.941~~	~~faf55e~~	~~null~~	~~null~~	~~null~~	~~abc.de/sold~~

Note: The ID field should be unique, but two different records share the same value!
→ merge duplicates into a single record

- There are different **dimensions of data quality**, especially:
  - *Completeness: Do we have all the data we need to assess page load performance?*
  - *Consistency: Does data have a valid format and does it comply with business semantics?*
  - *Accuracy: Do data items represent their corresponding real-world entities well?*
  - Uniqueness: Are duplicate records known and are all unique attributes actually distinct?

Maintaining Data Quality With Constraints

- **Constraints** are rules, conditions, or limits that data must adhere to
- **Type Checks** represent expectations on the data format, e.g.
  - *Value range* for numerical data (e.g. [0,MAX_INTEGER) for load timers)
  - *Format or pattern* for string-valued data (e.g. ISO 8601 for timestamps)
  - *Structure* for complex attributes (e.g. required keys for JSON objects)
- **Complex conditions** can further describe complex semantics such as
  - Cross-field or cross-record relationships (e.g. same browser within sessions)
  - Referential integrity between records in different collections
  - Custom constraints for domain semantics

Example: Declarative Constraints With Pandera (1/3)

```python
import pandas as pd
import pandera as pa
from pandera import Column, DataFrameSchema, Check

# Define the schema
schema = DataFrameSchema(
    {
        "Timestamp": Column(pa.DateTime),
        "Pageload ID": Column(pa.String,
            Check(lambda x: x.str.len() == 8)),
        "Browser": Column(pa.String,
            Check(lambda x: x.isin(["Chrome", "Edge", "Firefox"]))),
        "LCP": Column(pa.Int,
            Check(lambda x: (x >= 0) & (x <= 600000))),
        "Session ID": Column(pa.Int),
    }
)
```

Example: Declarative Constraints With Pandera (2/3)

```python
# valid data item
good = {
    "Timestamp": [pd.Timestamp("2023-05-10 13:26:48.139")],
    "Pageload ID": ["9cddf7"],
    "Browser": ["Firefox"],
    "LCP": [256],
    "Session ID": [456],
}

# invalid data item
bad = {
    "Timestamp": [pd.Timestamp("2023-05-10 09:05:04.578")],
    "Pageload ID": ["37ab08"],
    "Browser": ["Edge"],
    "LCP": [692654], # timer value out of bounds
    "Session ID": [123],
}
```

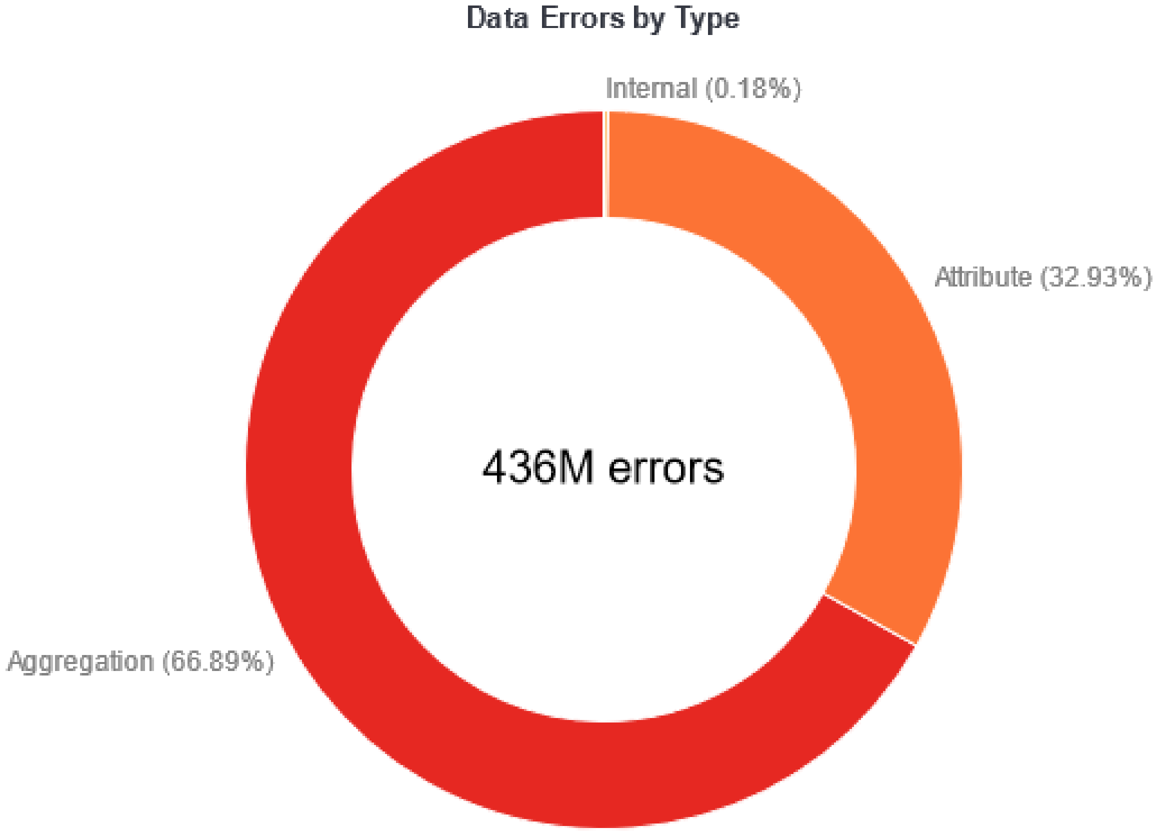
Example: Declarative Constraints With Pandera (3/3)

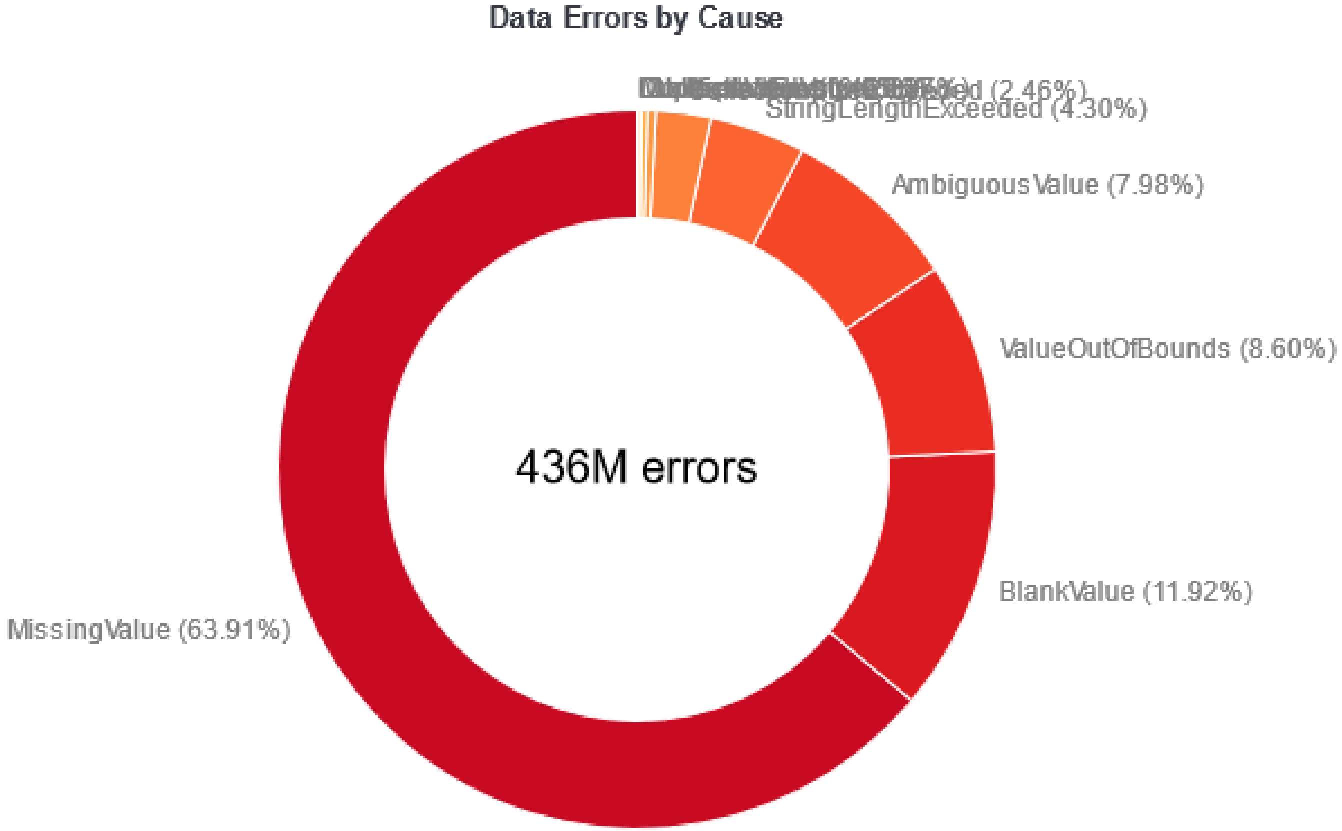
```python
for record in [good, bad]:
    record_id = record['Pageload ID'][0]
    try:
        validated = schema(pd.DataFrame(record))
        print(f"\nValidation passed for record {record_id}!")
    except pa.errors.SchemaError as e:
        print(f"\nValidation FAILED for record {record_id}:")
        print(e)
```

```
Validation passed for record 9cddf7!

Validation FAILED for record 37ab08
<Schema Column(name=LCP, type=DataType(int64))> failed element-wise
validator 0:
<Check <lambda>>
failure cases:
   index  failure_case
0      0        692654
```

Fundamental Challenge: Scalability

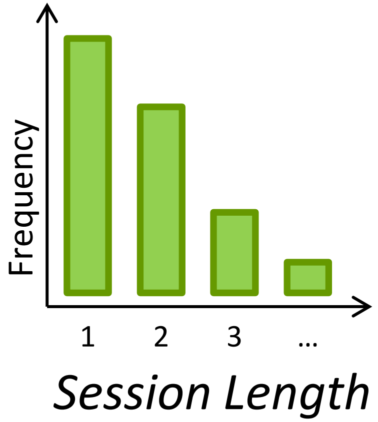
One Month in Data Errors at Baqend: April 2023





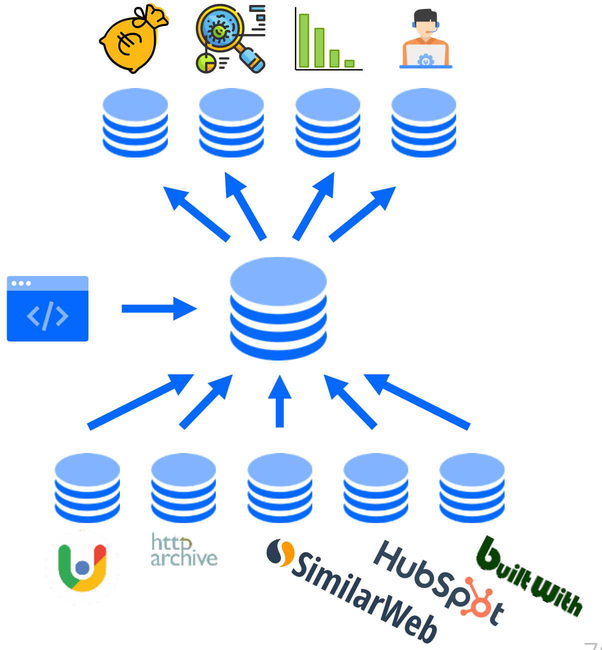
Challenging at Scale: Complexity

Timestamp	Pageload ID	Browser	LCP (Performance)	Session ID	URL	...	Session Length	Conversion
09:05:04.578	37ab08	Edge	**670**	123	null	...	1	0
13:26:48.139	9cddf7	Firefox	**null**	456	abc.de/red	...		
13:28:23.857	0b577a	Firefox	**256**	456	abc.de/blue	...	3	1
13:29:17.468	faf55e	**Firefox**	**1598**	456	abc.de/sold	...		
~~20:45:38.941~~	~~faf55e~~	~~null~~	~~null~~	~~null~~	~~abc.de/sold~~	...		

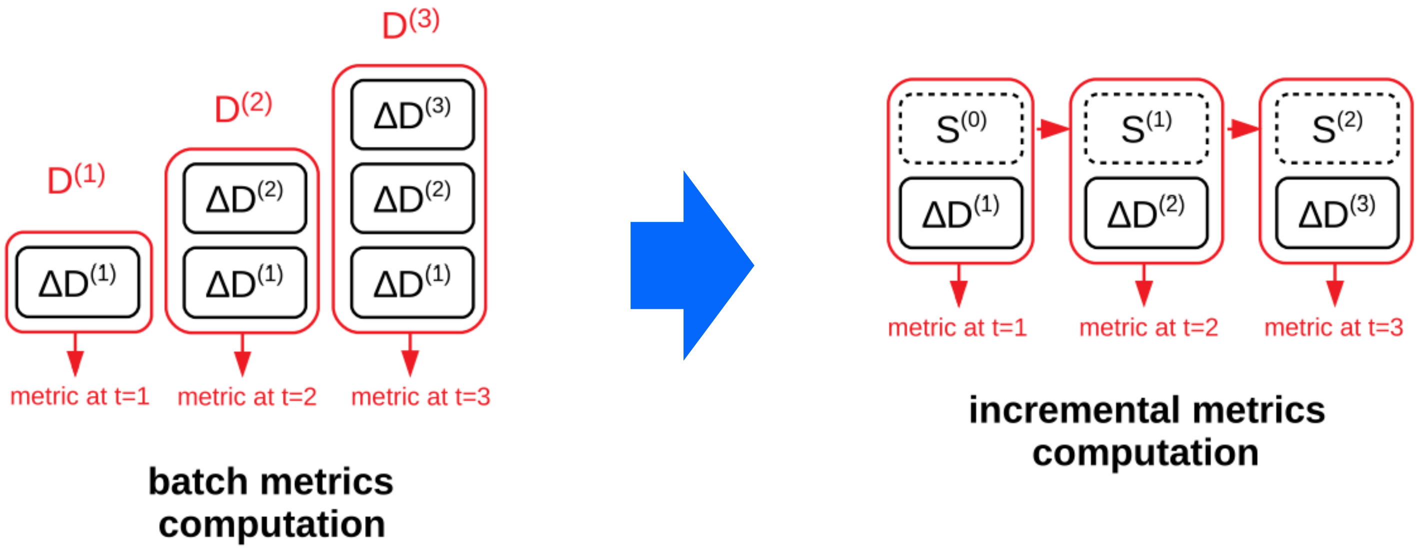


- Manual constraint definition is often infeasible, because of …
  - … inherent data complexity (often hundreds of attributes)
  - … aggregation, derived storage, and evolving schemas
  - … a plethora of other data stores to integrate!

→ **Automation** is necessary!



Challenging at Scale: Continuity



- Computing validation metrics from scratch periodically can be infeasible, because of …
  - … strict timing requirements
  - … efficiency or cost reasons
  - … data privacy reasons

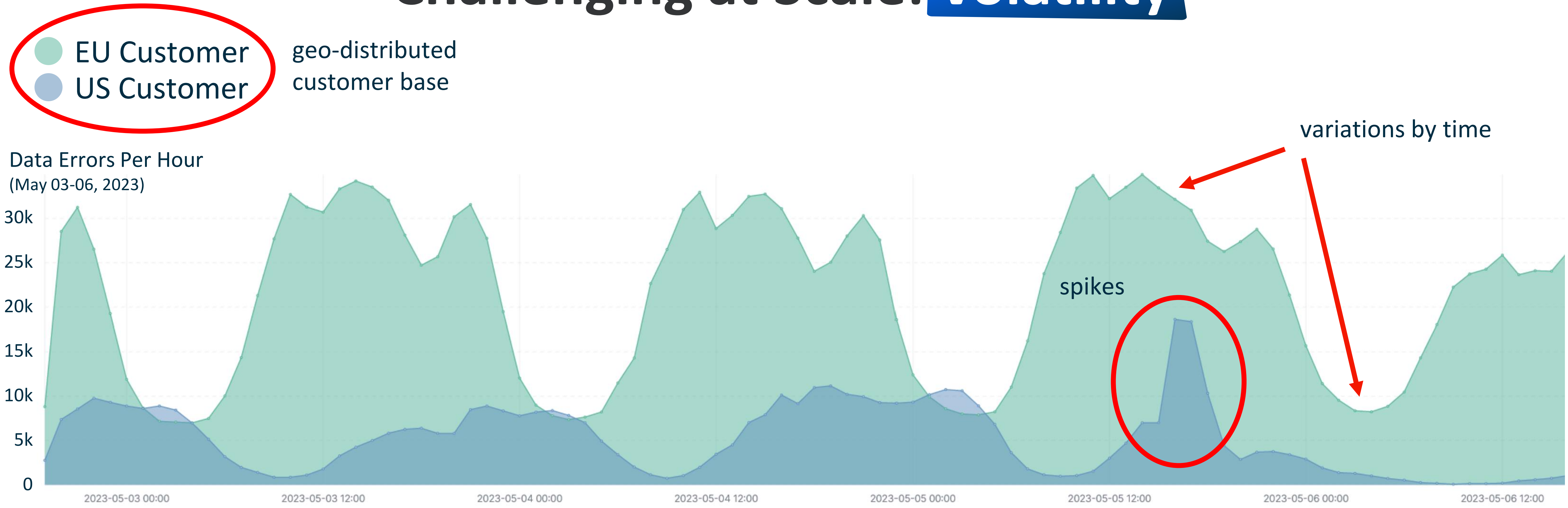
Details in the above-mentioned Flink Forward Presentation!

→ **Incremental computation** can be the only option!

W. Wingerath, B. Wollmer, M. Bestehorn, S. Succo, F. Bücklers, J. Domnik, F. Panse, E. Witt, A. Sener, F. Gessert, N. Ritter. Beaconnect: Continuous Web Performance A/B-Testing at Scale, VLDB (2022)

Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann: Automating Large-Scale Data Quality Verification, VLDB 2018.

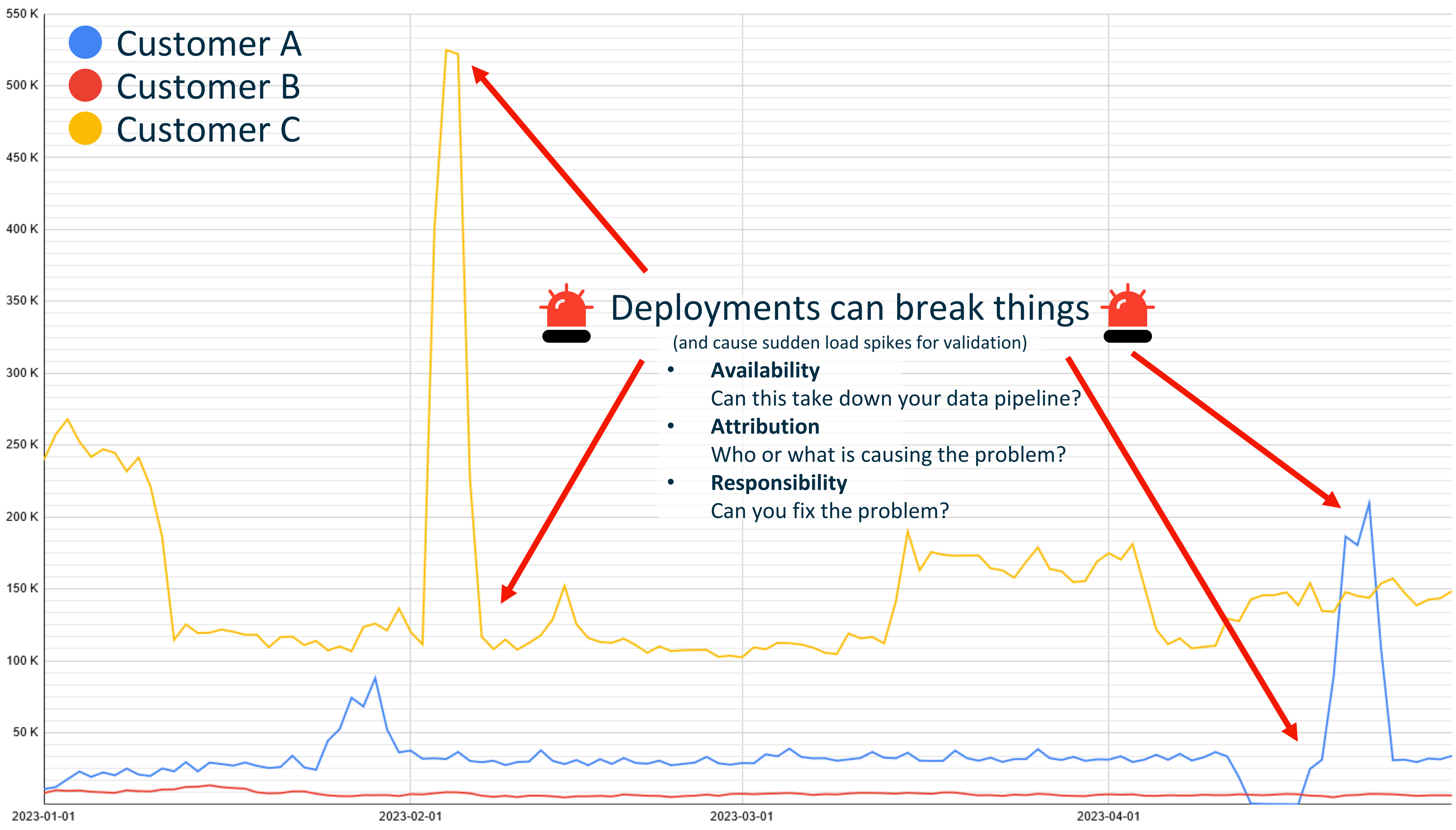
Challenging at Scale: Volatility



- Specifying generalized constraints can be difficult in large deployments, because of …
  - … temporal fluctuations (e.g. throughout the day, on black Friday, or during holidays)
  - … multi-tenancy (e.g. different data patterns by customer timezone or domain)

→ **Elasticity & Multi-Tenancy** requirements can be challenging!

Challenging at Scale: „Continility“ (Continuity + Volatility)



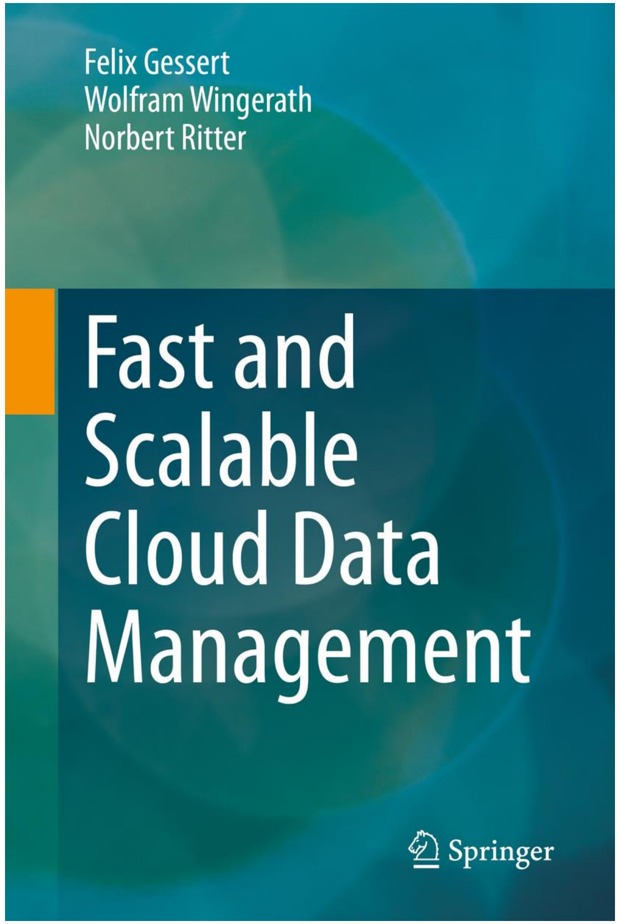
So How Do You Handle All This?

- **Advanced Techniques**
  - Inferring constraints
  - Adapting to schema changes
  - Incremental computation of complex measures
- **Tooling & Frameworks**
  - Validation libraries such as Great Expectations, Pandera, TFDV, or Deequ
  - Preprocessing and validation with Apache Spark and Apache Flink
- **Further Challenges**
  - Handling distribution (validation per partitioning, avoiding skew, ...)
  - Efficiency and performance (load distribution, approximation, ...)
  - Operational challenges (anomaly detection, fixing, load shedding, ...)

Data Validation at Scale: Summary

- **Data Quality** can be measured along dimensions such as completeness, consistency, accuracy, and uniqueness
- **Constraints** specify expectations about the data and can be used to enforce them
- **Data Validation** is the process of ensuring high data quality for processes like analysis, modeling, and decision-making
- Data Validation **Challenges at Scale** include
  - *Complexity*: schemas are often too complex to define constraints manually
  - *Volatility*: data varies throughout the day, by season, or by customer
  - *Continuity*: incremental processing is required when computation from scratch is infeasible

More on the Topic




K.-U. Sattler et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2021), Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 423

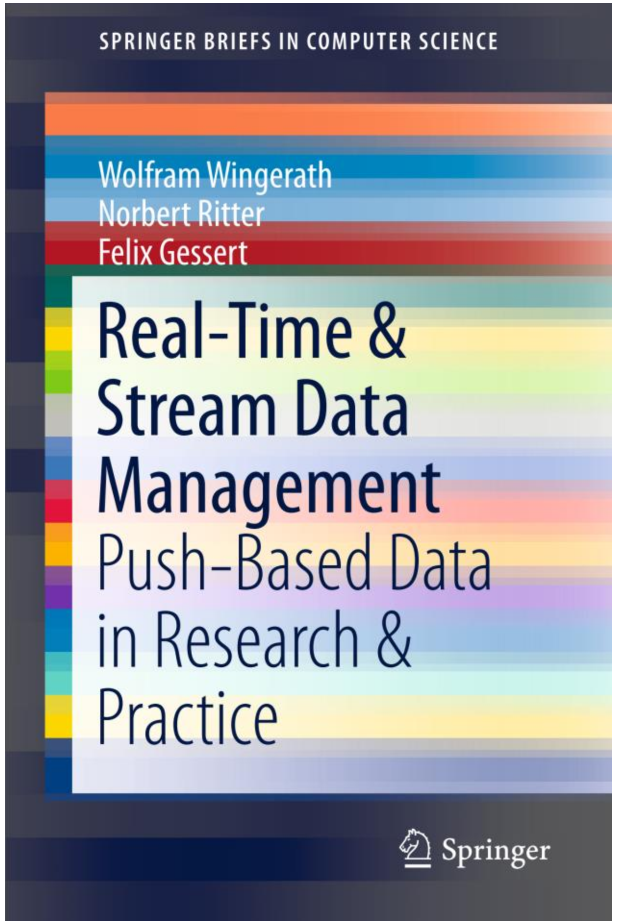
Data Management in Multi-Agent Simulation Systems

From Challenges to First Solutions

Daniel Glake[1] Fabian Panse[1] Norbert Ritter[1] Thomas Clemen[2] Ulfia Lenfers[2]

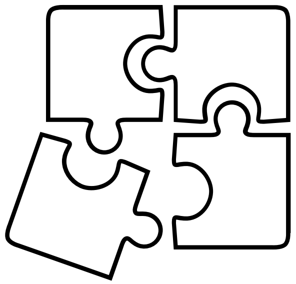
Abstract: Multi-agent simulations are an upcoming trend to deal with the urgent need to predict complex situations as they arise in many real-life areas, such as disaster or traffic management. Such simulations require large amounts of heterogeneous data ranging from spatio-temporal to standard object properties. This and the increasing demand for large scale and real-time simulations pose many challenges for data management. In this paper, we present the architecture of a typical agent-based simulation system, describe several data management challenges that arise in such a data ecosystem, and discuss their current solutions within our multi-agent simulation system MARS.

Keywords: Multi-agent simulations, Spatio-temporal data, Polyglot data management



For videos & books, visit
https://wolle.science!

Wrapup: Working With Big Data



- The **classic challenges** of Big Data management are also known as the „3 Vs of Big Data“:
  - *Volume*: How to reliably process huge amounts of data?
  - *Velocity*: How to keep throughput up and latency down?
  - *Variety*: How to handle all kinds of structured & unstructured data?
- But there are **additional challenges**:
  - *Veracity*: Is your data (source) trustworthy / meaningful?
  - *Visualization*: How to communicate? insights & knowledge?
  - *Value*: How to use data for (machine) learning, optimization, ...?
  - (Volatility, Vulnerability, Validity, ...)

More than just data management & engineering

Thanks! Questions ?

Material Available at https://wolle.science