Big Data
Session 5: Processing Large Data Streams

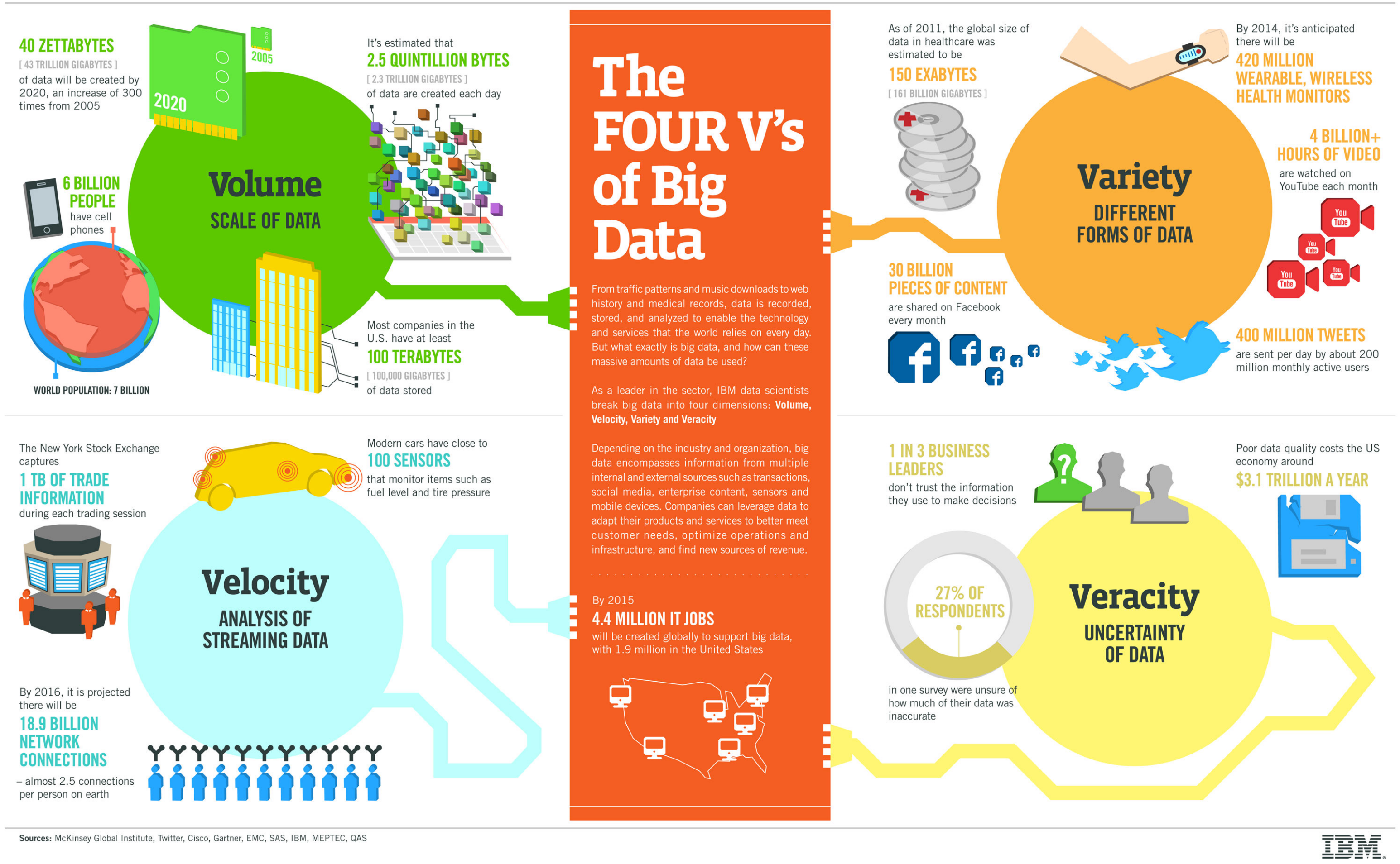
Frank Hopfgartner
Institute for Web Science and Technologies

universität koblenz
weiter:denken

Last week

- Map/Reduce framework
- Querying
  - Spark Core API
  - Pig, Pig Latin
- Machine Learning at Scale
  - Spark MLLib
  - Mahout

Recap: Four V's of Big Data



Source: http://www.ibmbigdatahub.com/sites/default/files/infographic_file/4-Vs-of-big-data.jpg

Motivation

- So far we have really just talked about processing historical, existing big data
  - Sitting on HDFS
  - Sitting in a database
- But how does new data get into your cluster? Especially if it is ‘big data’?
- Streaming lets you publish this data, in real-time, to your cluster
  - And you can even process it in real-time as it comes in.

Intended Learning Outcomes

At the end of this lecture, you will be able to:

- Outline use cases to stream data
- Explain record-at-a-time streaming
- Distinguish between various declarative streaming cases

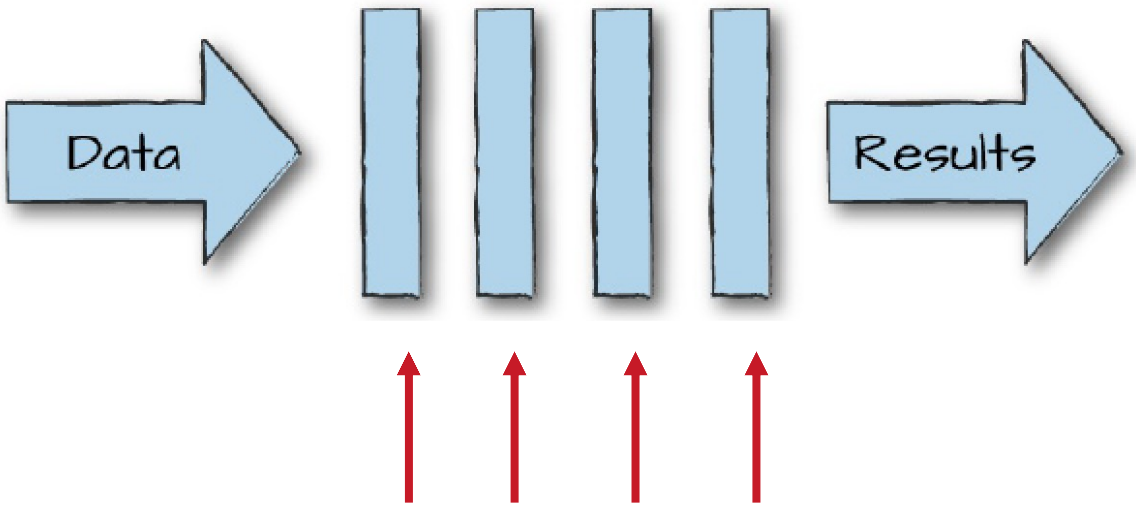
Outline

universität koblenz
weiter:denken

- Introduction
- Record-at-a-time streaming
- Declarative, functional streaming
- Declarative, relational streaming

What is streaming?

Contiuously integrating new, infinitely large data to compute results



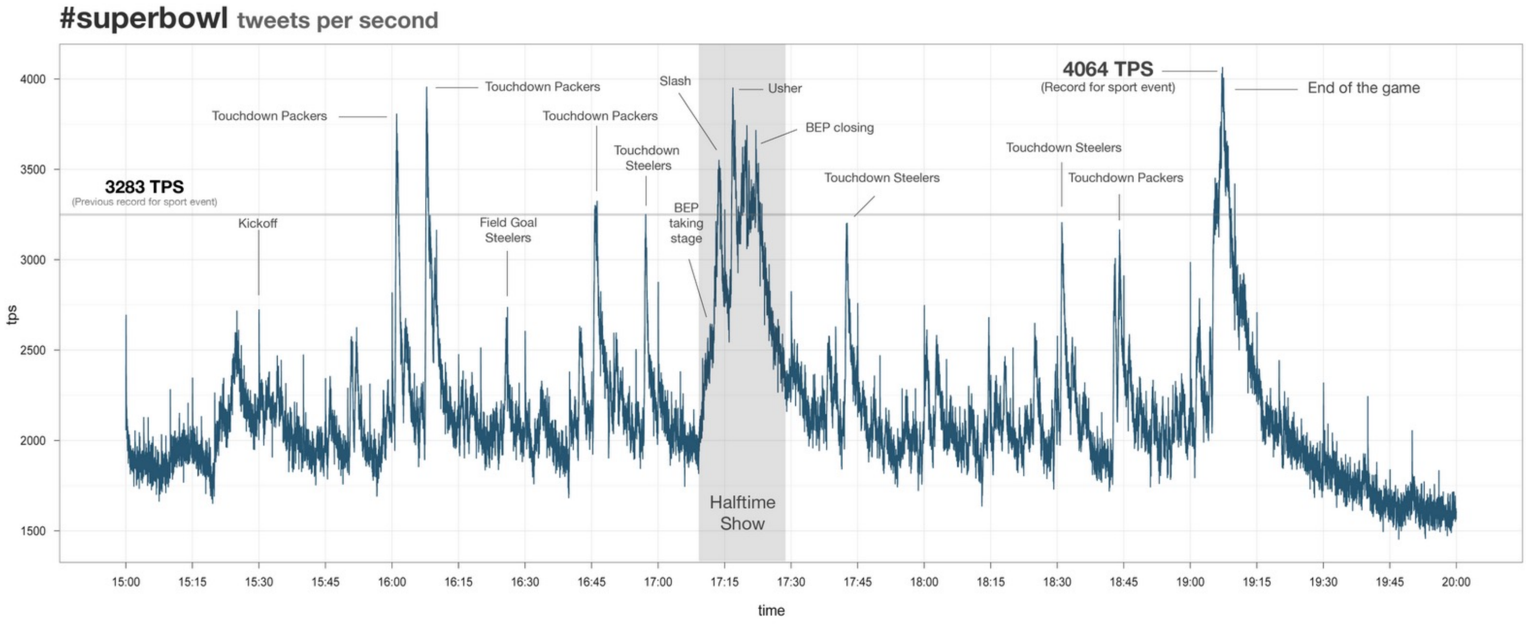
Processing steps in pipeline fashion

Given a stream source

- The amount of data received is unlimited in size

- The volume of data is continuous/variable
  - Typically unpredictable

- Example: Twitter



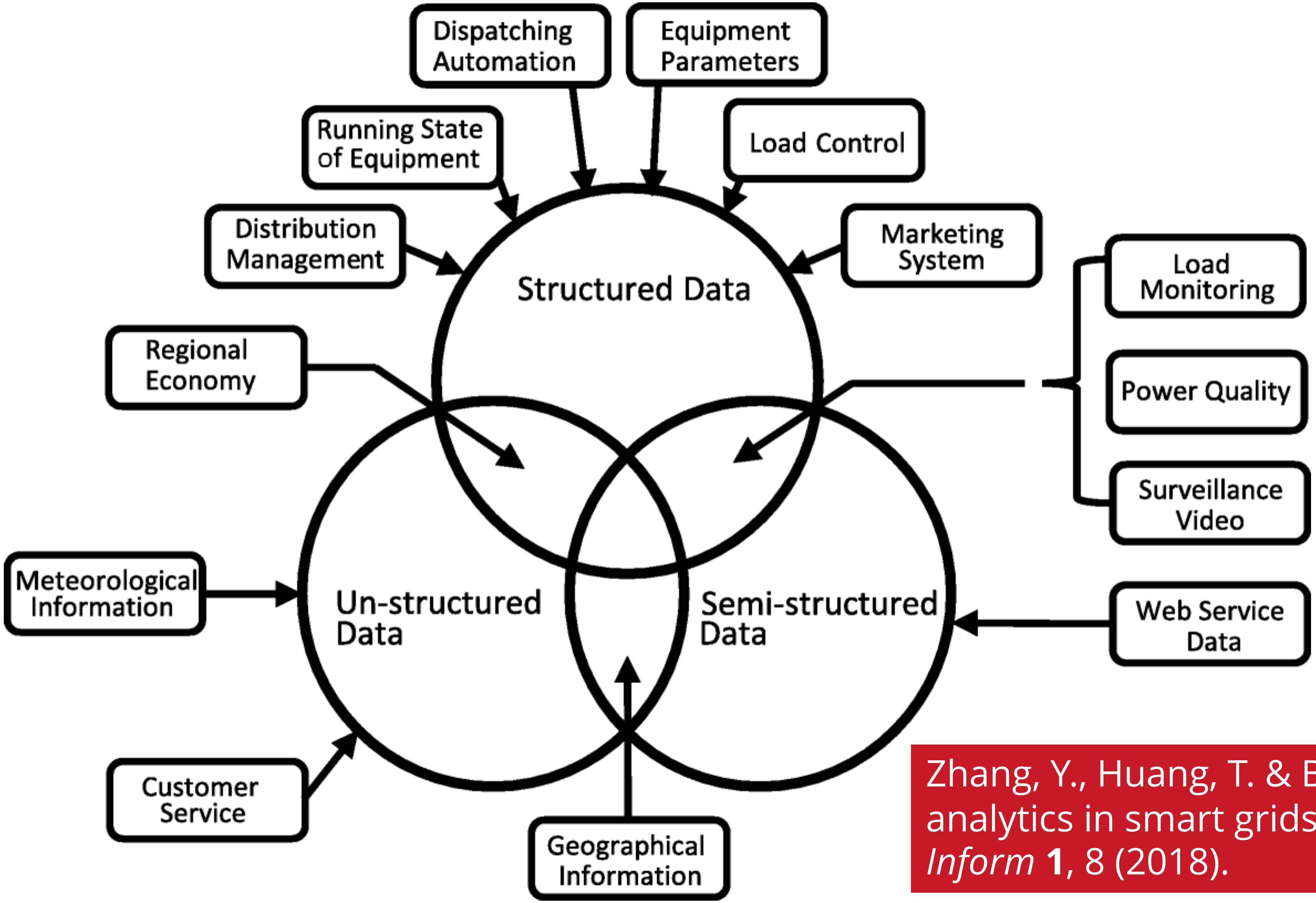
Stream processing use cases

1. Notifications and alerting
   - Redundant failures to log in
2. Real-time reporting
   - Dashboards, e.g. for production/traffic flow, system load, uptime,….
3. Incremental ETL
   - Update the data warehouse
   - Maintain correctness (do not lose data or add data twice)
4. Update data to serve in real time
   - e.g. updating a key-value or relational store with statistics
5. Real-time decision making
   - Analyzing new inputs, e.g. analyzing credit card transactions to discover and prevent fraud
6. Online machine learning
   - e.g. learning to recommend on changing platforms

Example: Thailand's Tsunami warning system

https://www.youtube.com/watch?v=s-9exzIbOrs

Example: Smart Grids



Zhang, Y., Huang, T. & Bompard, E.F. Big data analytics in smart grids: a review. *Energy Inform* **1**, 8 (2018).

Advantages of streaming

universität koblenz
weiter:denken

- Lower latency
  - Timescale: minutes, seconds, milliseconds
  - No need to re-process all previously occurred data
  - Retaining **state of system**

- Higher efficiency for updates
  - Compared to repeated batch jobs

Batch vs. Stream: Same, same, but different

universität koblenz
weiter:denken

Same functionality needed

- Computing account balances
- Computing statistics
- …

Batch processing

- Fixed data during processing
- Query triggers processing

Stream processing

- Continuous arrival of new data
- New data triggers processing (mostly)

Challenges of stream processing

- Batch processing is simpler to understand, troubleshoot and program
- Batch processing allows for much higher throughput
- Data may arrive out of order, e.g.,

```
{value: 1, time: "2017-04-07T00:00:00"}
{value: 2, time: "2017-04-07T01:00:00"}
{value: 5, time: "2017-04-07T02:00:00"}
{value: 10, time: "2017-04-07T01:30:00"}
{value: 7, time: "2017-04-07T03:00:00"}
```

  - Consider a question like:
    - Did 2 – 10 – 5 appear?
    - Did 2 – 10 – 5 **not** appear?
  - Remember events? For how long?
  - Remember states?

Challenges of stream processing (2)

- Processing out-of-order data
  based on application timestamps (*event time*)
- Maintaining large amounts of state
- Supporting high-data throughput
- Processing each event exactly once despite machine failures
- Handling load imbalance and stragglers
- Responding to events at low latency
- Joining with external data in other storage systems
- Determining how to update output sinks as new events arrive
- Writing data transactionally to output systems
- Updating your application‘s business logic at runtime

Streaming approaches

Record-at-a-Time:

- API just hands over one record-at-a-time to application
- Application handles all challenges
- Apache Storm

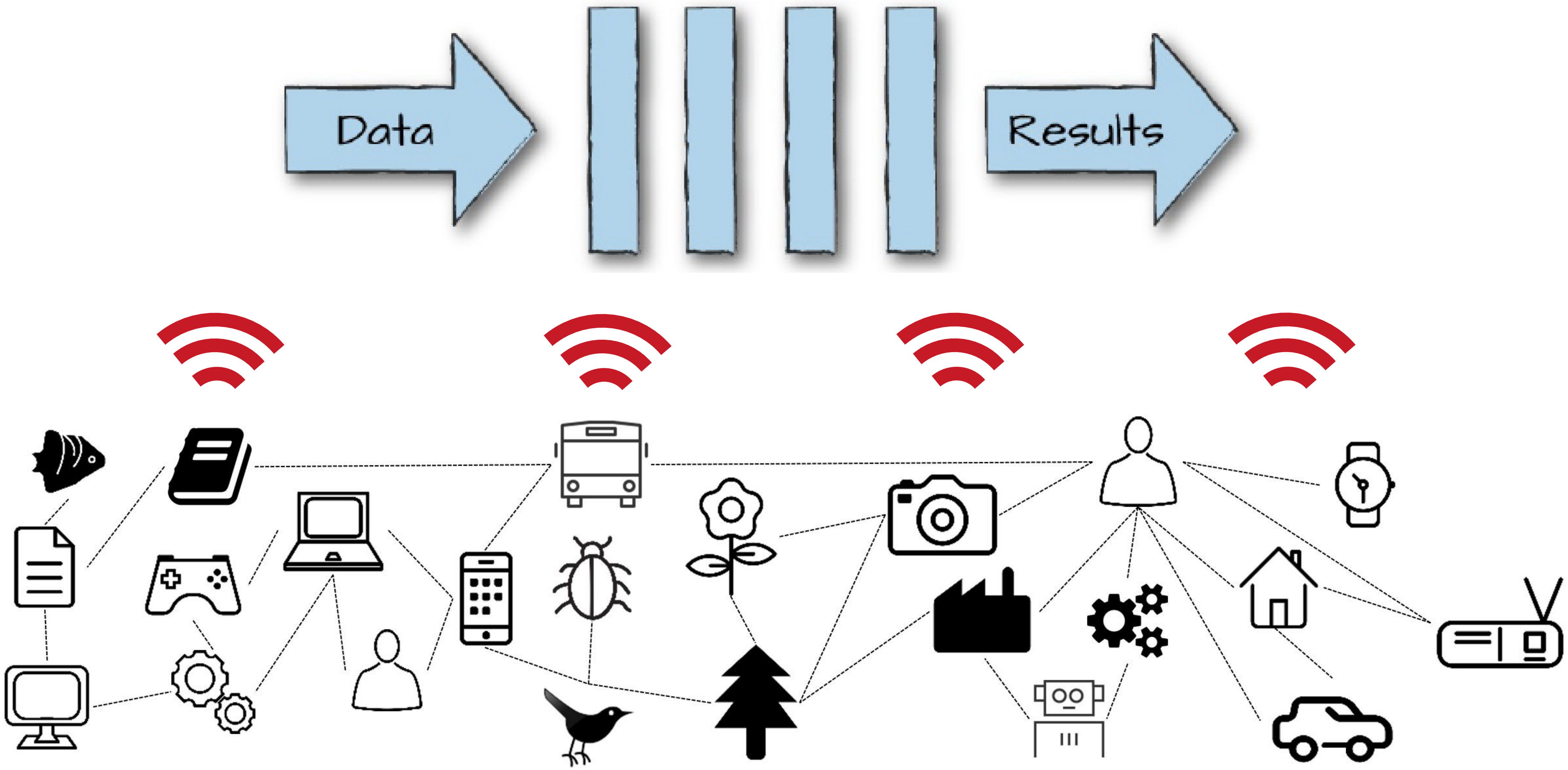
Declarative, functional API:

- Describe what to compute, not how
- Functional: *map*, *reduce*, *filter*
- Dstreams API, Google Dataflow, Apache Kafka

Declarative, relational API:

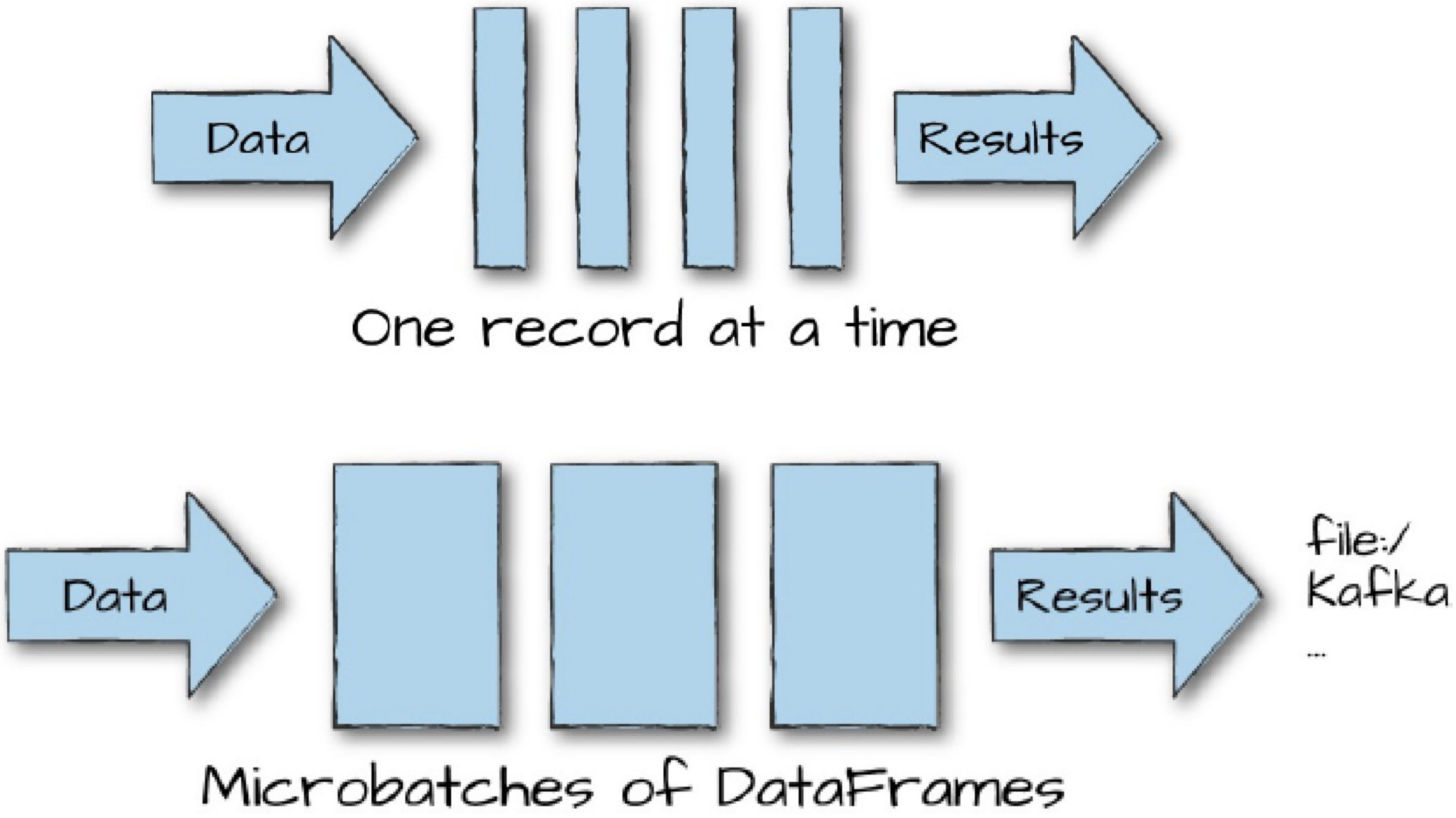
- Rich automatic optimization of execution (beyond functional)
- Spark Structured Streaming, Apache Flink

Event time vs. processing time



Web of Things

Continuous vs. batch processing



Trade-offs

Continuous processing

- Processes data immediately
- Lower base latency

Microbatch processing

- Waits for some amount of data to arrive before processing
- Latencies starting at 100ms to 1s
- Improved throughput
- Preferred by distributed streaming
  - If scalability is an issue, throughput must be optimized

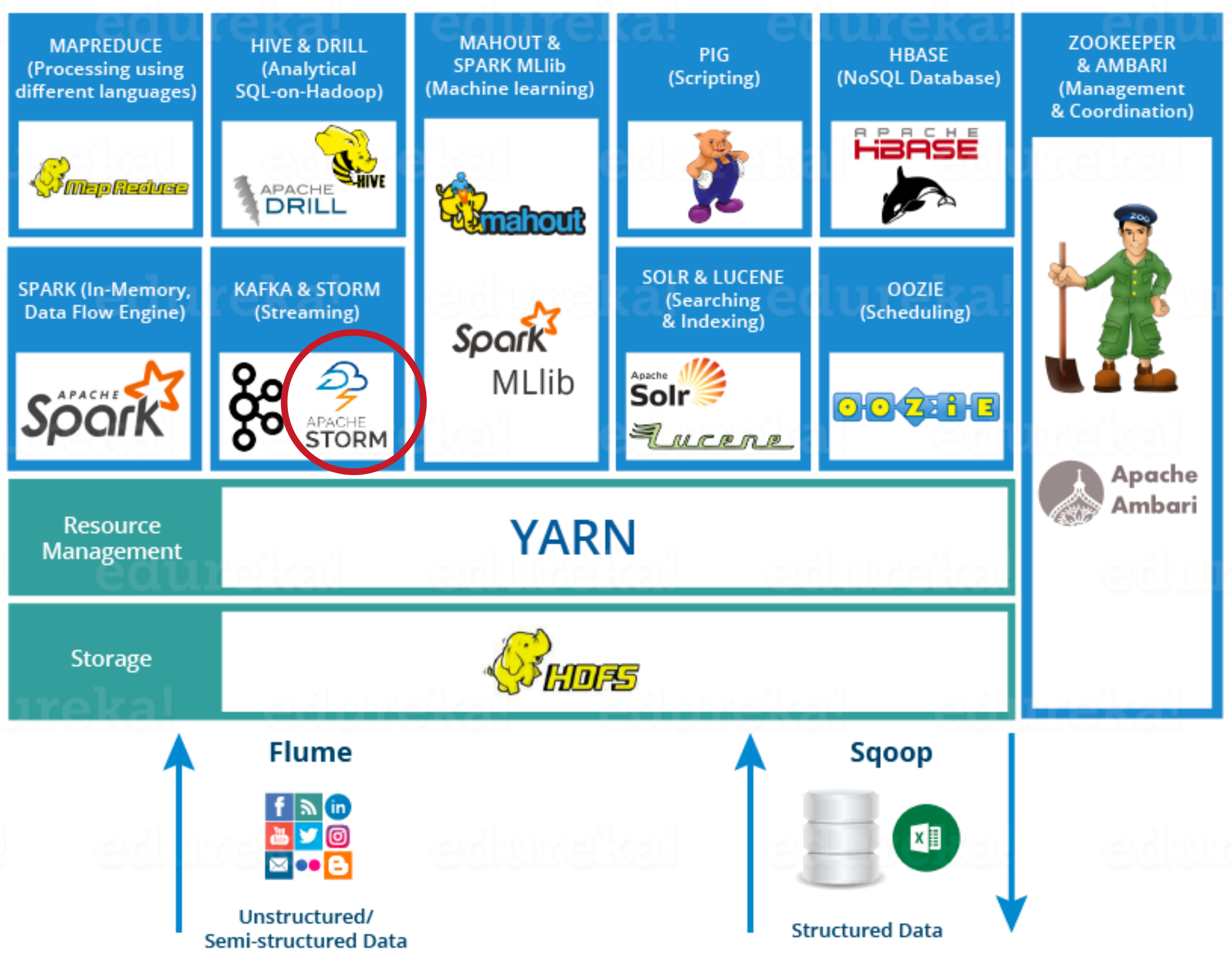
Practical, continuous applications

- reacts to data in real time
- mixes
  - Streaming jobs
  - Batch jobs
  - Joins between streaming and offline data
  - Interactive ad-hoc queries

Outline

- Introduction
- **Record-at-a-time streaming**
  - **Apache Storm**
- Declarative, functional streaming
- Declarative, relational streaming

Hadoop ecosystem



Apache Storm

universität koblenz
weiter:denken

- Distributed and fault-tolerant real-time computation system for processing limitless streaming data
- Built at Twitter
- Real-time analytics
  - Not batch data processing like Hadoop
- Define a topology: graph of computation
  - Consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams as needed

https://storm.apache.org/

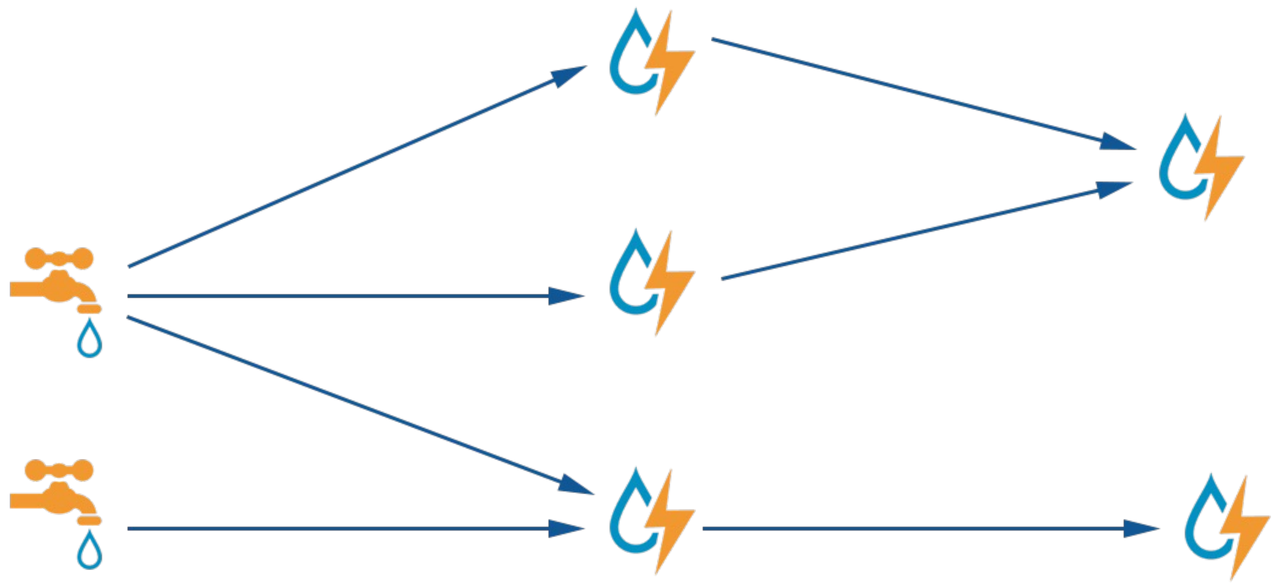
Storm vs Hadoop

Storm	Hadoop
Real-time streams of data	Batch data processing
Stateless	Stateful (data stored on HDFS)
Zookeeper coordination	Zookeeper coordination
1K msg / sec processed	TB/PB processed in minutes/hours
Topology runs as more data arrives	M/R jobs completed and results written on HDFS

Apache Storm

- Two kinds of nodes: spouts and bolts
  - **Spout**: source of data streams
  - **Bolt**: process input stream and outputs new stream

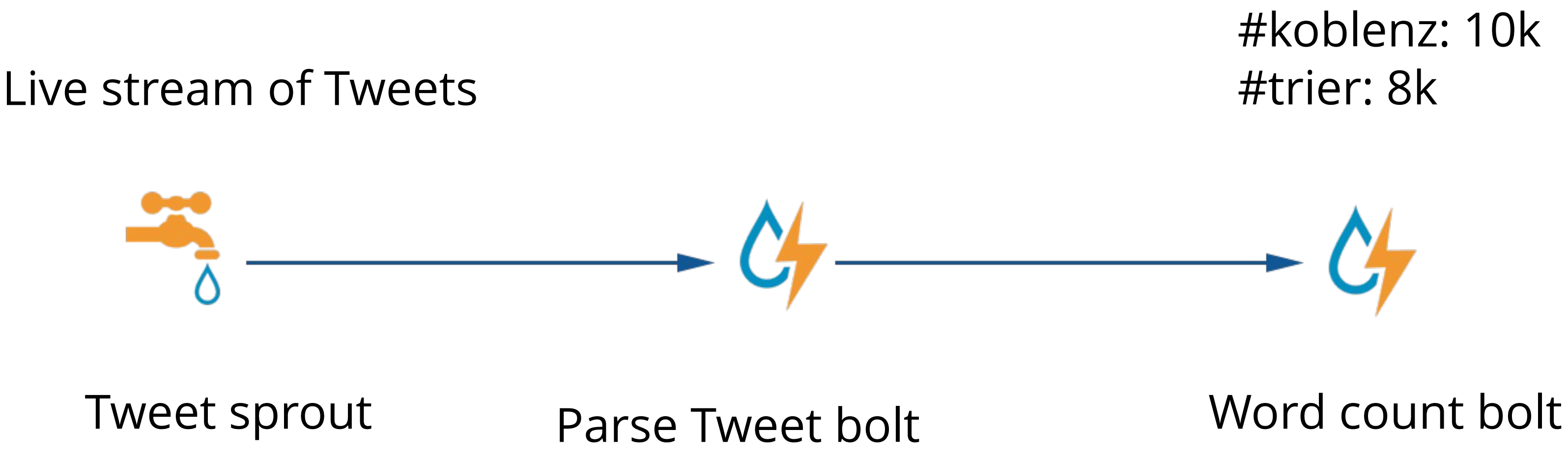
- Nodes execute in parallel



Apache Storm

- Spout
  - Data sources like Twitter Streaming API
  - Kafka queue (see later)
  - Read from datasources
- Bolt
  - Filtering, aggregation, joining operations
  - Interact with other datasources, e.g., databases
- Topology
  - Directed graph where vertices are computation and edges are stream of data
  - Distributed over multiple worker nodes running all the time and waiting for jobs to process
  - Multiple nodes can execute one bolt and take a share of the data
  - Topology is run by the master node (called Nimbus) assigning tasks to nodes

Example: Twitter word count



Storm – Use cases

	"Prevent" Use Cases	"Optimize" Use Cases
Financial Services	Securities fraud Operational risks & compliance violations	Order routing Pricing
Telecom	Security breaches Network outages	Bandwidth allocation Customer service
Retail	Shrinkage Stock outs	Offers Pricing
Manufacturing	Preventative maintenance Quality assurance	Supply chain optimization Reduced plant downtime
Transportation	Driver monitoring Predictive maintenance	Routes Pricing

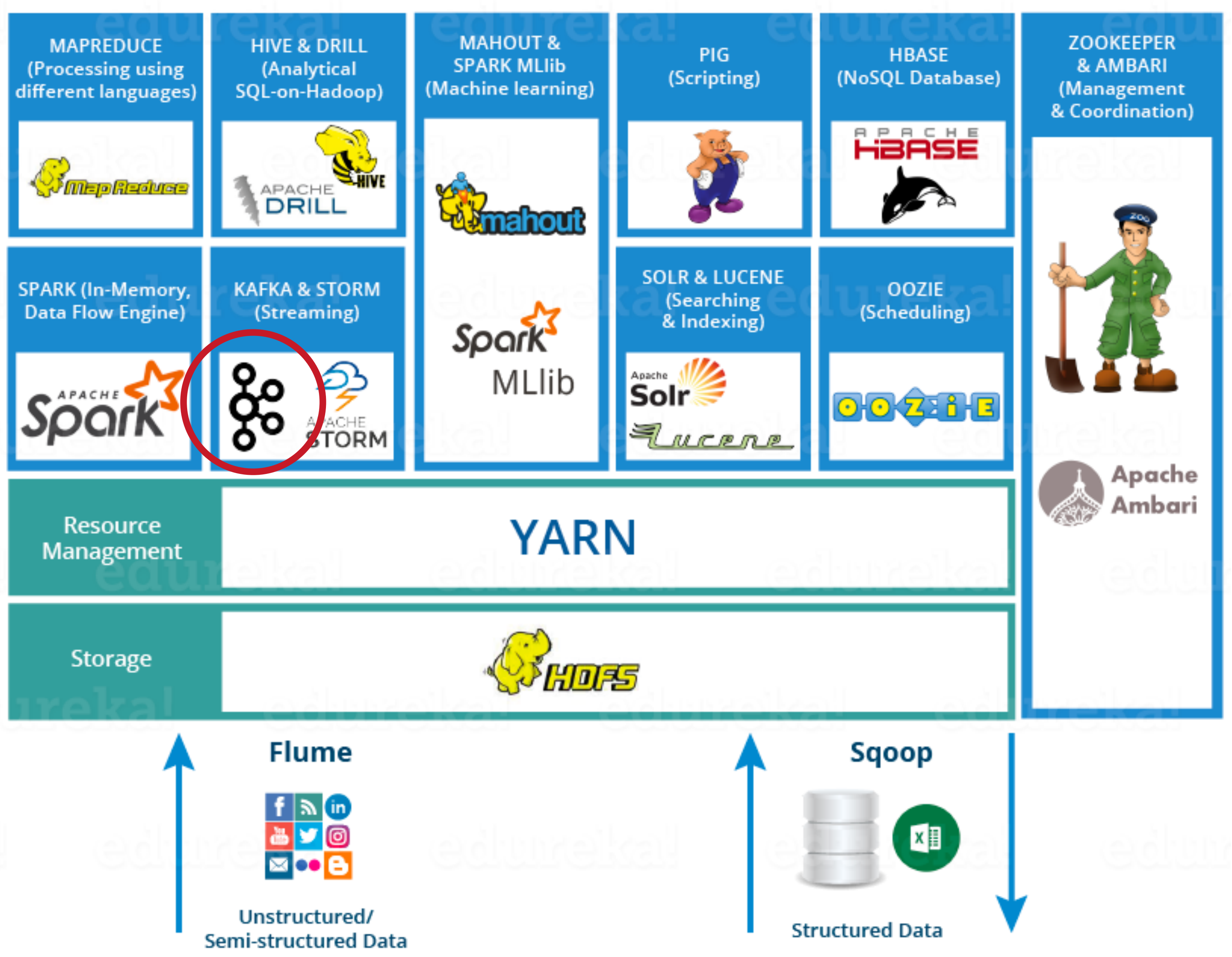
https://hortonworks.com/apache/storm/

Outline

universität koblenz
weiter:denken

- Introduction
- Record-at-a-time streaming
- **Declarative, functional streaming**
  - **Apache Kafka**
  - **DStreams API**
- Declarative, relational streaming

Hadoop ecosystem



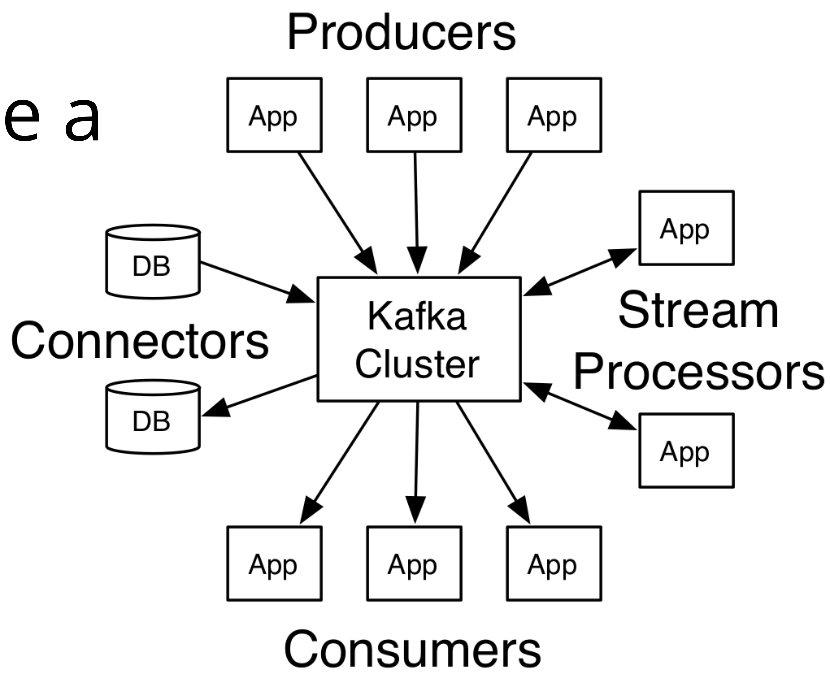
Apache Kafka

- Originally built by LinkedIn (open-sourced in 2011)
- Kafka is a general-purpose **publish/subscribe messaging system**
- Kafka servers store all incoming messages from *publishers* for some period of time, and publishes them to a stream of data called a *topic.*
- Kafka *consumers* subscribe to one or more topics, and receive data as it is published
- A stream/topic can have many different consumers, all with their own position in the stream maintained
- It’s not just for Hadoop.

https://kafka.apache.org/intro.html

Publish/subscribe model

- Broadcast data to multiple processes
- Producers: publish a stream of data
- Consumers: subscribe to a stream
- Stream processors: consume and produce a new stream

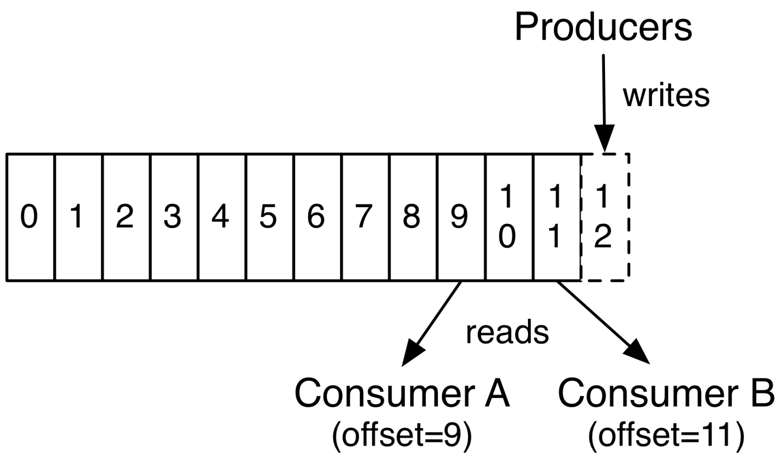


How Kafka operates

- Kafka itself may be distributed among many processes on many servers
  - Will distribute the storage of stream data as well
- Consumers may also be distributed
  - Consumers of the same group will have messages distributed amongst themselves
  - Consumers of different groups will get their own copy of each message

Processing

- Producers write into a sequence of records that is continually appended
  - Sequences are partitioned and distributed in the cluster to scale
  - Partitions are replicated for fault tolerance
  - Order only maintained within a partition!
- Kafka cluster retains all published records for a predefined retention period (e.g., 2 days)
- Consumers read content using offset information



Example Kafka applications

- A retail application
  - takes in input streams of **sales** and **shipment** data
  - outputs a stream of **reorders** and **price adjustments** based on this data
- Usage at LinkedIn
  - https://engineering.linkedin.com/kafka/kafka-linkedin-current-and-future
    - Page views, clicks
  - https://engineering.linkedin.com/kafka/running-kafka-scale
    - Multiple datacenters
    - Mirroring data across Kafka clusters

APIs in Spark

universität koblenz
weiter:denken

Batch APIs

- RDD
  - Low level
- Data Frame
  - High level
  - Several means for self-optimization available

Streaming APIs

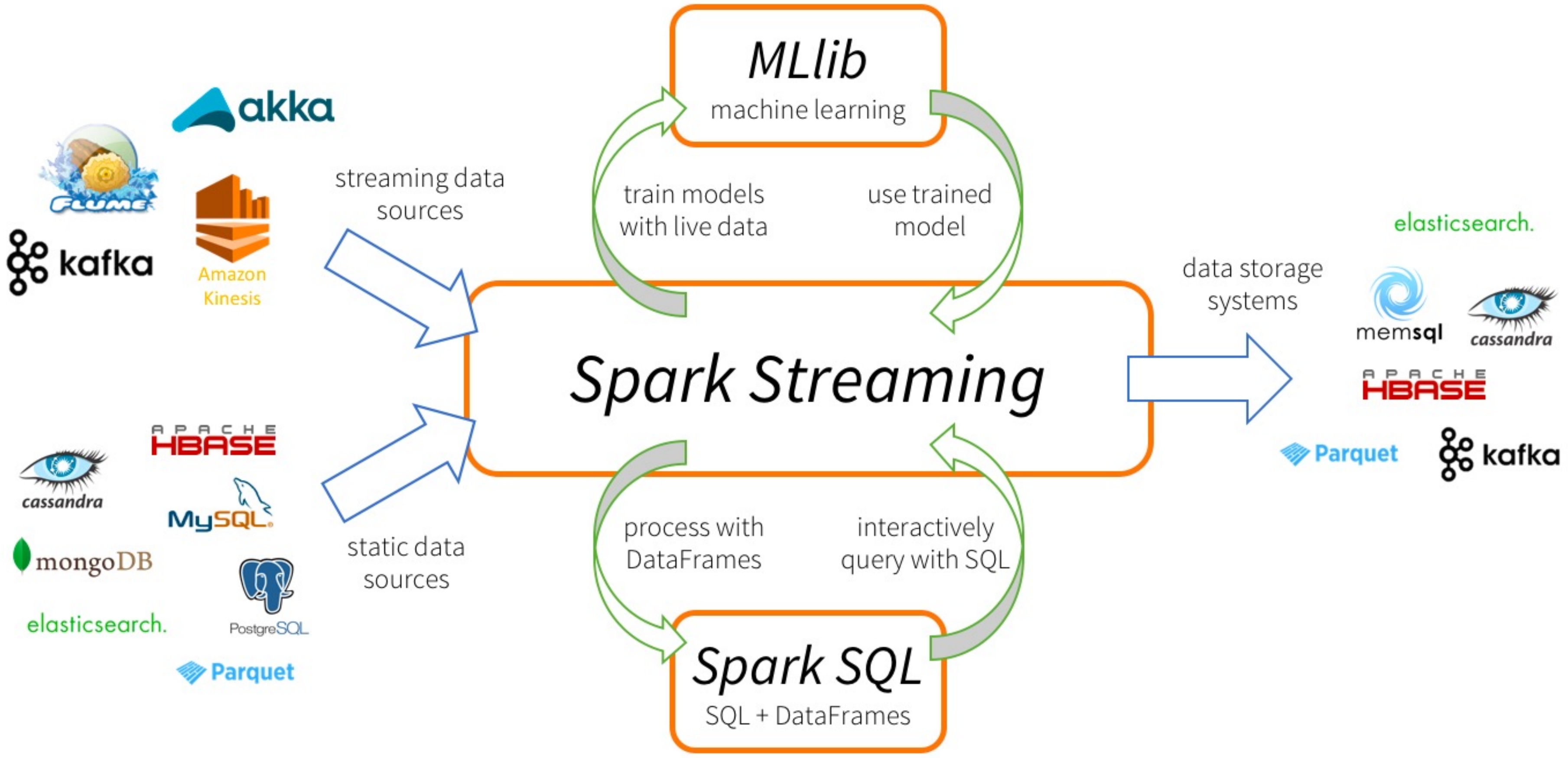
- Spark Streaming / DStreams API
  - Low level
- Structured Streaming API
  - High level
  - Several means for self-optimization available

Spark streaming

- An extension of the core Spark
- Process real-time data from different data sources
- Stream of data divided into small batches (Discretized Stream – DStream)
  - Build on RDDs
- Can seemlessly integrate with other Spark components
- Scalable, high-throughput, fault-tolerant stream processing of live data streams

https://spark.apache.org/docs/latest/streaming-programming-guide.html

Spark streaming ecosystem



Spark Streaming – Data Sources

universität koblenz
weiter:denken

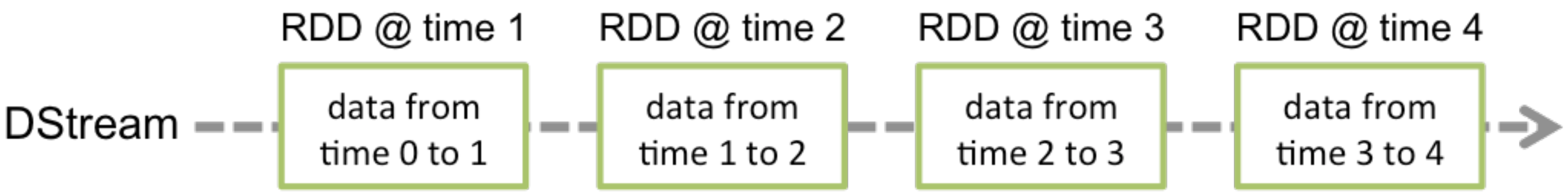
- Kafka
  - Most popular (and older)
  - High throughput (20k msg/sec)
- Apache Flume
  - Streaming event log data (web page visits, clicks) from a web server
  - Distributed / high availability
- Kinesis
  - Amazon AWS solution
- Streams are represented as a sequence of **RDDs**.

Spark – Stream processing

- Series of batch computations on small time intervals (windows over the stream)
- Spark Streaming receives live input data streams
- Divides the data into batches
- Spark engine processes batches

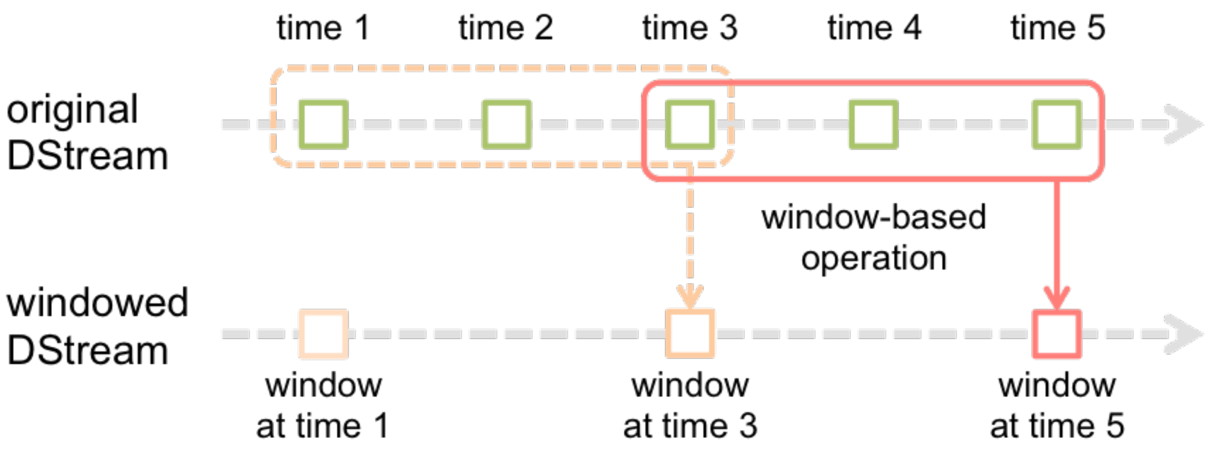
Discretised Streams (DStreams)

- Continuous stream of data
  - From source
  - Transforming an input file
- DStream is represented by a continuous series of RDDs
  - Each RDD has data from a certain interval
- Resilient Distributed Datasets (RDDs)
  - Keep data in memory
  - Can recover it without replication (track the lineage graph of operations that were used to build it)



Window operations

- Apply **transformations** (map, flatMap, etc) over a sliding window of data
- RDDs that fall within the window are combined and operated upon
  - Parameters: *window length, sliding interval*
  - Custom window-based transformations



Spark Streaming Fault-tolerance

universität koblenz
weiter:denken

- Streams arrive 24/7
- Storage able to recover from failures (HDFS)
  - Store computation metadata
  - Store data from streams
- When a node fails, each node in the cluster works to recompute part of the lost node's RDDs
- Batch interval needs to be set such that the expected data rate in production can be sustained

Spark Streaming – Use Cases

- Uber
  - Data from mobile users
  - Kafka as data source
  - Event data to structured data into HDFS
  - Analytics as M/R
- Pinterest
  - Real-time user interaction analysis
  - Use this for recommendations (products to buy, places to visit)

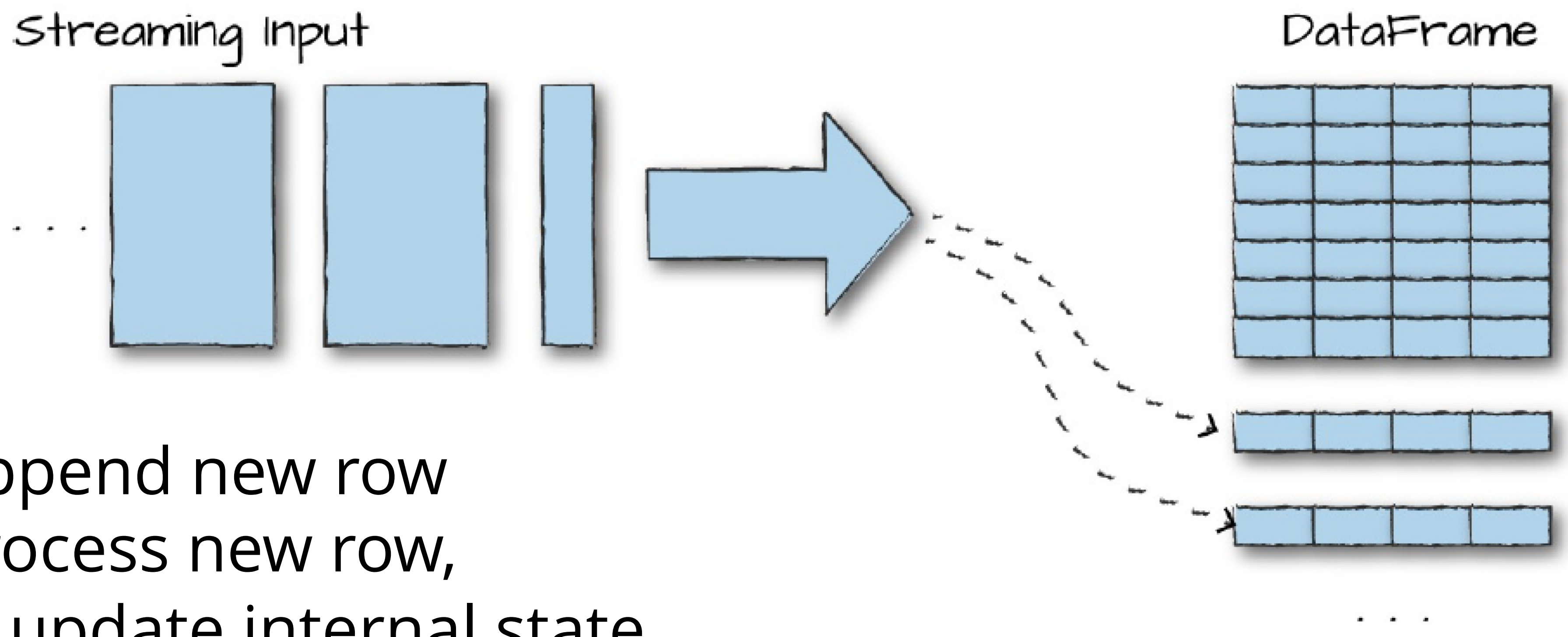
Outline

- Introduction
- Record-at-a-time streaming
- Declarative, functional streaming
- **Declarative, relational streaming**
  - **Spark Structured Streaming API**
  - **Apache Flink**

Spark Structured Streaming

- More recent
- More types of optimization (compared to Dstreams)
- Native support for event time
- Microbatches
- Write DataFrame or SQL computation!
  Integrates well with batch functionality!
- Output to Parquet (for downstream usage)

Structured Streaming Processing



1. Append new row
2. Process new row,
   - update internal state
   - update result

(3.) Evict oldest rows

Structured Streaming In/Out

universität koblenz
weiter:denken

Input from

- Apache Kafka
- Files
  - HDFS
  - S3
- Socket

Output to

- Apache Kafka
- File
- Foreach sink
- Console sink (for testing)
- Memory sink (for debugging)

Apache Flink

universität koblenz
weiter:denken

- Developed at TU Berlin, Apache project since 2014
- Big data processing engine: distributed and scalable streaming dataflow engine
- Exploits data streaming and in-memory processing and iteration operators to improve performance
- Seen as 4th Generation IFP system because:

  Iterative algorithms

  Internal optimization mechanisms

  Uses Lambda architecture model

  Hybrid programming architecture: allows simultaneous batch and real-time runs.

Who uses Apache Flink?

- **Alibaba** uses a Flink-based system to optimize search rankings in real-time
- **Ericsson** used Flink to build a real-time anomaly detector over large infrastructures using machine learning
- **Huawei** Cloud offers a product called "CloudStream", based on Apache Flink
- **Netflix, Uber, Zalando, …**

See: https://cwiki.apache.org/confluence/display/FLINK/Powered+by+Flink

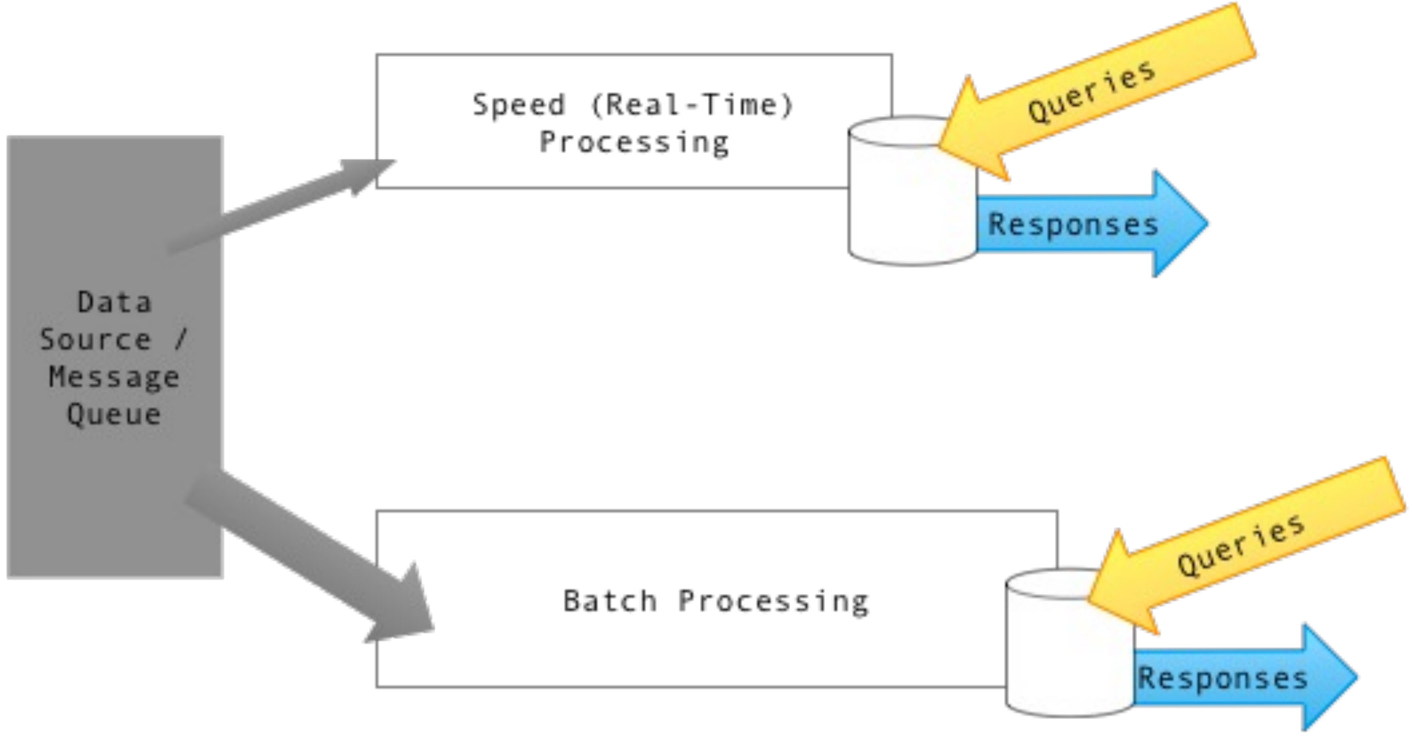
Apache Flink Framework

- Several APIs in Java/Python/Scala
  - Dataset API – Batch processing
  - DataStream API – Real-time streaming analytics
  - Table API – Relational Queries
- Domain specific libraries
  - FlinkML: Machine Learning Library for Flink
  - Gelly: Graph library for Flink
- Shell for interactive data analysis

Lambda architecture

- Batch layer: stores ALL the incoming data in an immutable master dataset and pre-computes batch views on historic data.
- Serving layer: indexes views on the master dataset.
- Real-time processing layer: requests data views depending on incoming queries.

Lambda architecture



Stream Processing Frameworks

universität koblenz
weiter:denken

G. van Dongen and D. Van den Poel, "Evaluation of Stream Processing Frameworks," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1845-1858, 1 Aug. 2020, doi: 10.1109/TPDS.2020.2978480

Summary

universität koblenz
weiter:denken

- Introduction
- Record-at-a-time streaming
- Declarative, functional streaming
- Declarative, relational streaming

What's next – Processing Graph Data

- Network Theory (Briefly)
- Data representation
- Graph Processing Examples
- Distributed Systems for Graph Processing