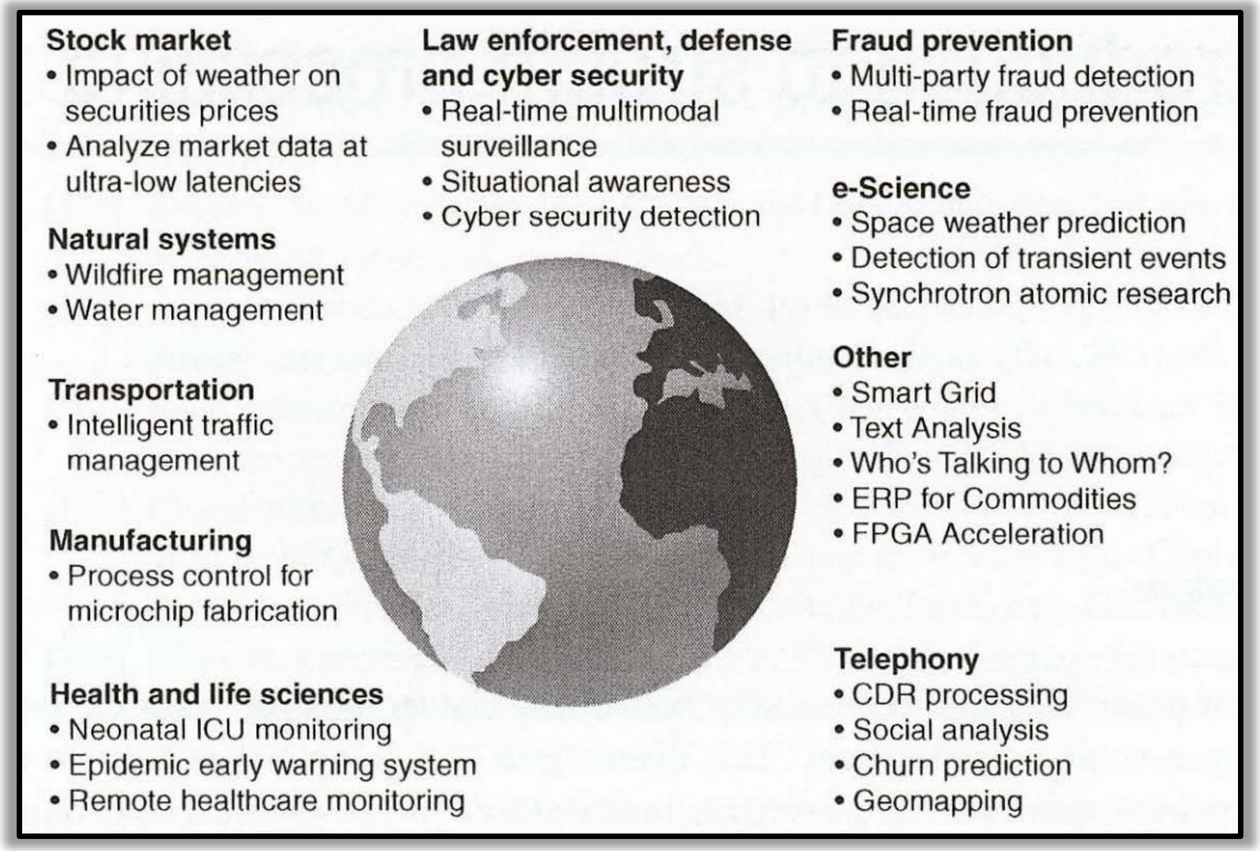
Big Data: Real Time Data Systems

Core Concepts & Principles

Nagiza F. Samatova
Professor, Department of Computer Science
North Carolina State University
and
Senior Scientist, Computer Science & Mathematics Division
Oak Ridge National Laboratory

Use Cases: Real-time Data Systems



Stock market
- Impact of weather on securities prices
- Analyze market data at ultra-low latencies

Natural systems
- Wildfire management
- Water management

Transportation
- Intelligent traffic management

Manufacturing
- Process control for microchip fabrication

Health and life sciences
- Neonatal ICU monitoring
- Epidemic early warning system
- Remote healthcare monitoring

Law enforcement, defense and cyber security
- Real-time multimodal surveillance
- Situational awareness
- Cyber security detection

Fraud prevention
- Multi-party fraud detection
- Real-time fraud prevention

e-Science
- Space weather prediction
- Detection of transient events
- Synchrotron atomic research

Other
- Smart Grid
- Text Analysis
- Who's Talking to Whom?
- ERP for Commodities
- FPGA Acceleration

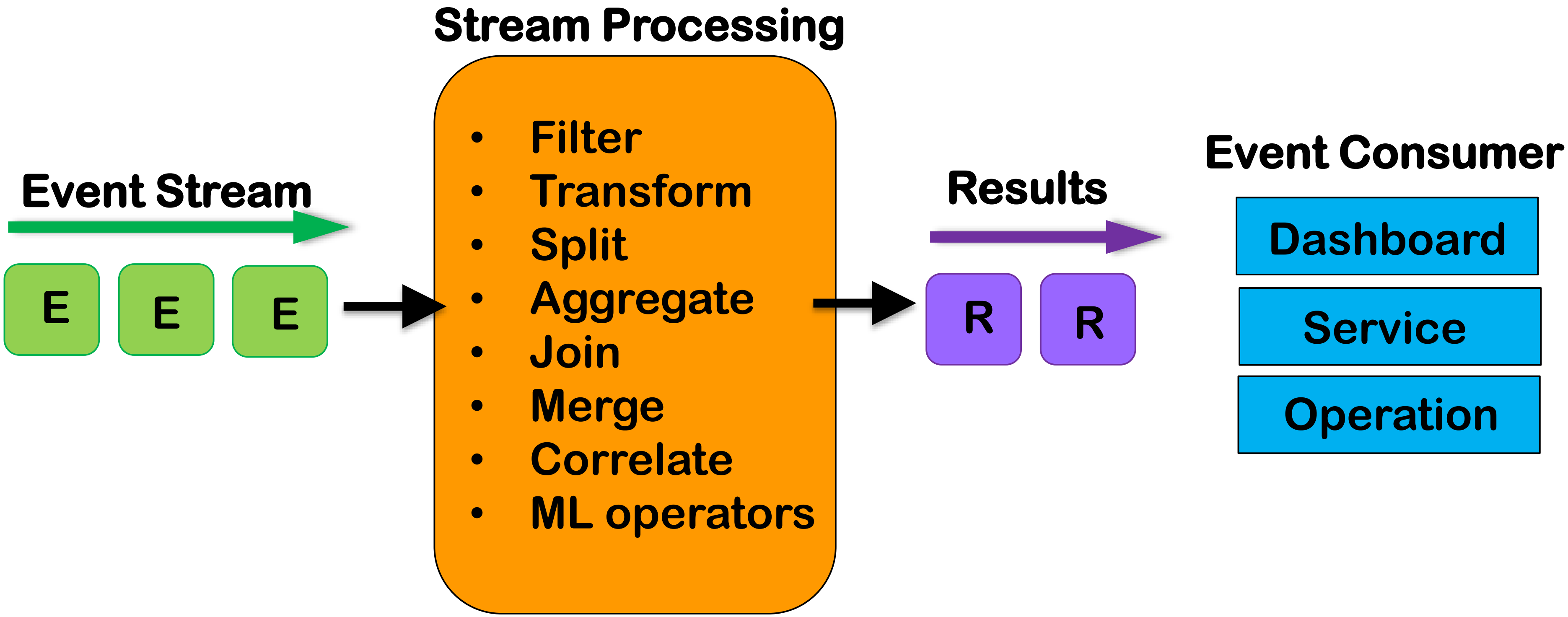
Telephony
- CDR processing
- Social analysis
- Churn prediction
- Geomapping

Src: Fig. 2.1: Fundamentals of stream processing book.

- Churn prediction
- Financial markets
- Healthcare sensor monitoring
- Manufacturing
- Traffic systems
- Infrastructure monitoring
- Security
- Water management
- Multi-model surveillance for law enforcement
- Fraud detection and prevention for e-Commerce

What is Stream Processing Infrastructure?

Infrastructure for **continuous (∞)** processing of data streams—**Event Processing/Complex Event Processing (CEP)**—producing **high-throughput** results at **low latency**.



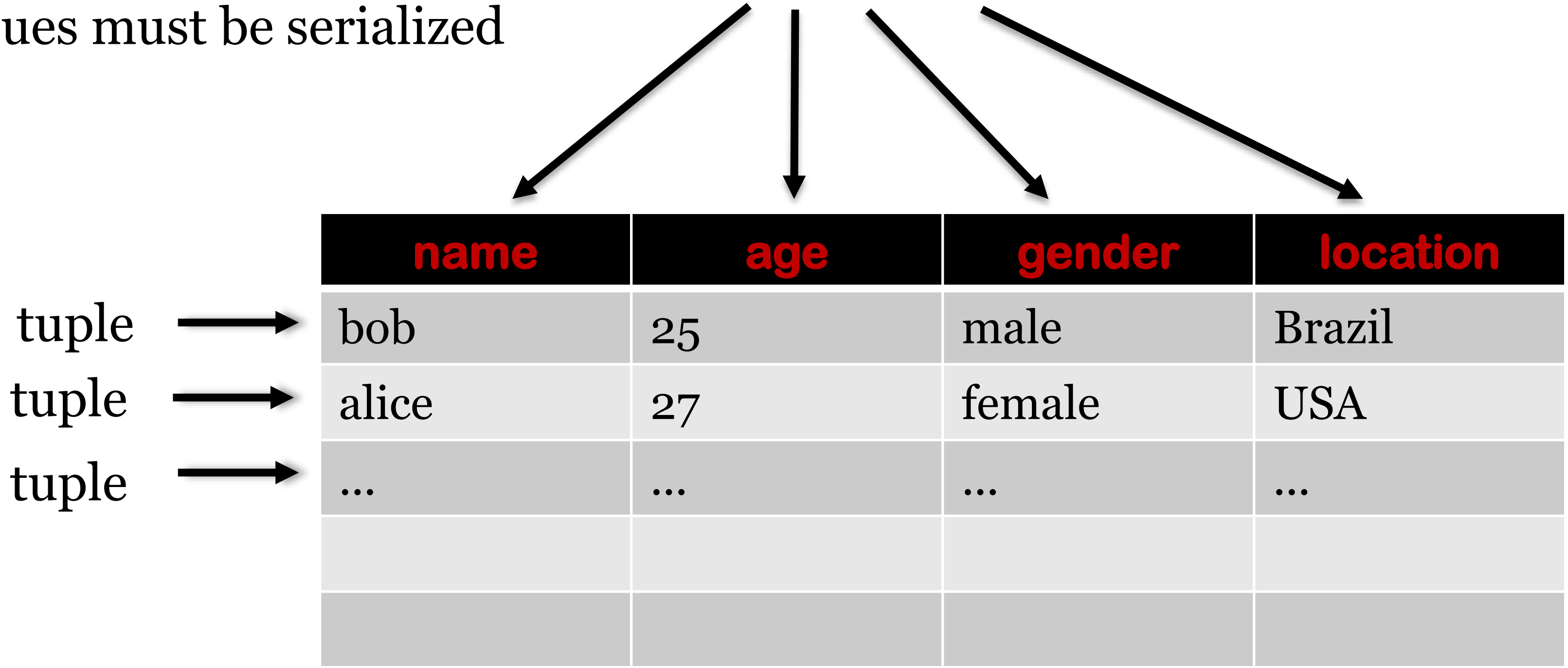
- Filter
- Transform
- Split
- Aggregate
- Join
- Merge
- Correlate
- ML operators

Tuple: Event with Named Fields

tuples ≡ events

Tuple

- Core data structure in stream processing middleware (e.g., Storm, Spark)
- Immutable set of key/value pairs
- Includes an arbitrary number of **named fields**
- Values must be serialized



	name	age	gender	location
tuple	bob	25	male	Brazil
tuple	alice	27	female	USA
tuple	…	…	…	…

Stream: Infinite Sequence of Tuples

Stream ≡ ∞ sequence of events/tuples

Stream

sequence vs. set

- An unbounded sequence of tuples
- Each stream is given an ID
- Computational units (i.e., Bolts) consume tuples from these streams on the basis of their ID
- Each stream has a schema of the tuples that flow through the stream

name	age	gender	location
bob	25	male	Brazil
alice	27	female	USA
…	…	…	…

Processing Model

Batch, Micro-Batch, Event-Stream

Batch

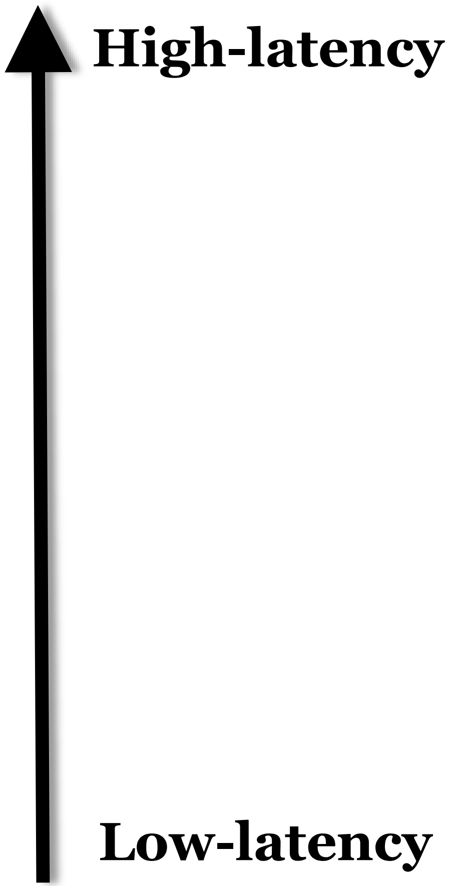
- Process data en mass
- Incurs high latency

Micro-Batch

- Mix of batching and streaming
- At cost of latency
- Offers stateful computation
- Windowed tasks become possible

Event-Stream

- Sub-second latency
- A datum/event/tuple at a time
- Stateful computation is expensive



High-latency

Low-latency

Processing Semantics: Delivery Guarantee

At Most Once, At Least Once, Exactly Once

At Most Once [0,1]

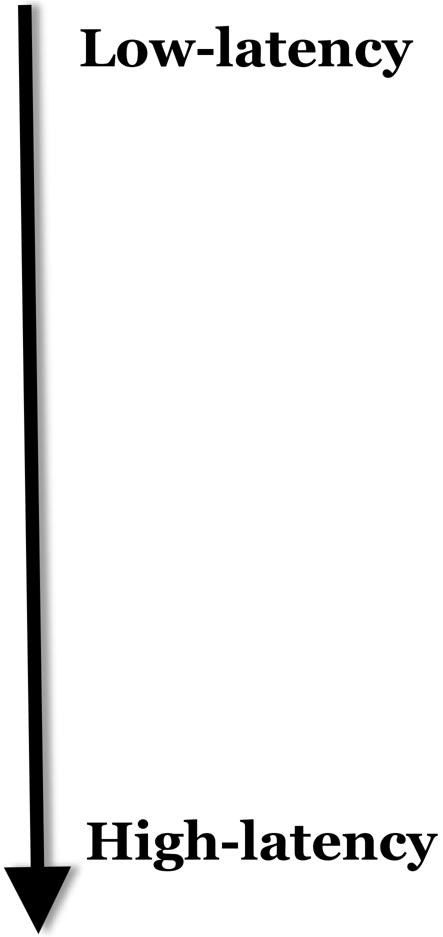
- Tuples may be lost
- Tuples never redelivered

At Least Once [1…n]

- Tuples are never lost
- Tuples maybe redelivered
  (ok to process multiple times)

Exactly Once [1]

- Tuples are never lost
- Tuples are never redelivered
- Perfect delivery
- Incurs high latency for transactional semantics



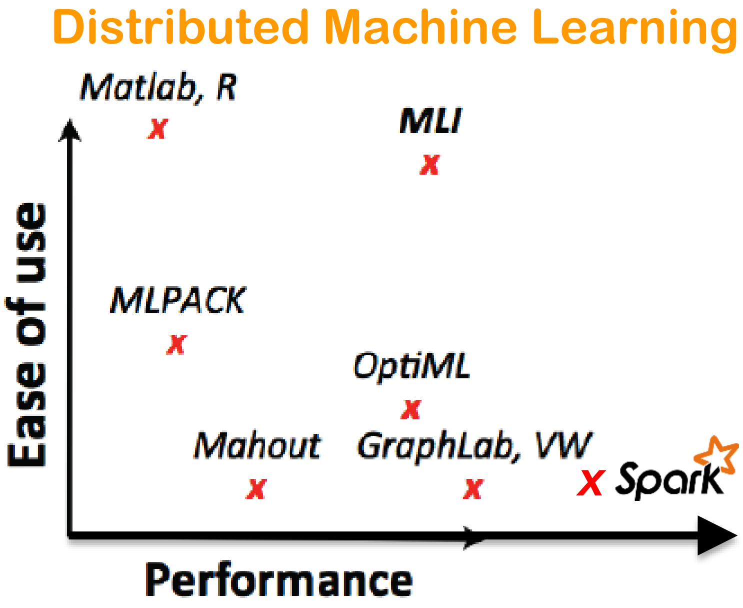
Low-latency

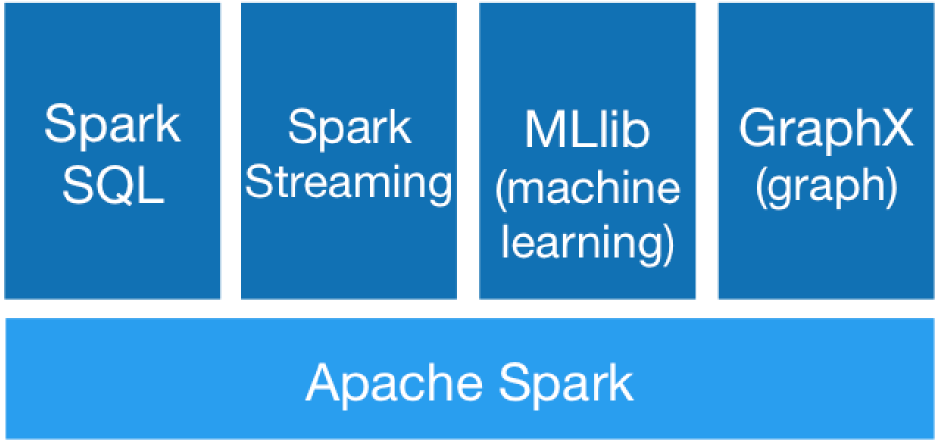
High-latency

What is Apache Spark?

An **in-memory, batch & stream, parallel & distributed** large-scale data processing infrastructure

- Originated at UC Berkeley AMPLabOpen (2009)
- Open Sourced: 2010
- Part of Apache Incubator: since February 2014
- Written in Java, Scala
- Languages: Java, Closure, Python, Scala, Rubi
- Scalable
- Fault-tolerant
- Simplifies working queues & workers
- Complementary to Hadoop, batch processing


Distributed Machine Learning



In addition to simple *map* and *reduce* operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box.

Better yet, combine these capabilities seamlessly into one integrated workflow…

What is Apache Storm?

APACHE STORM™
Distributed • Resilient • Real-time

A stream processing infrastructure for **highly distributed**, **real-time**, **data stream** processing and analysis.

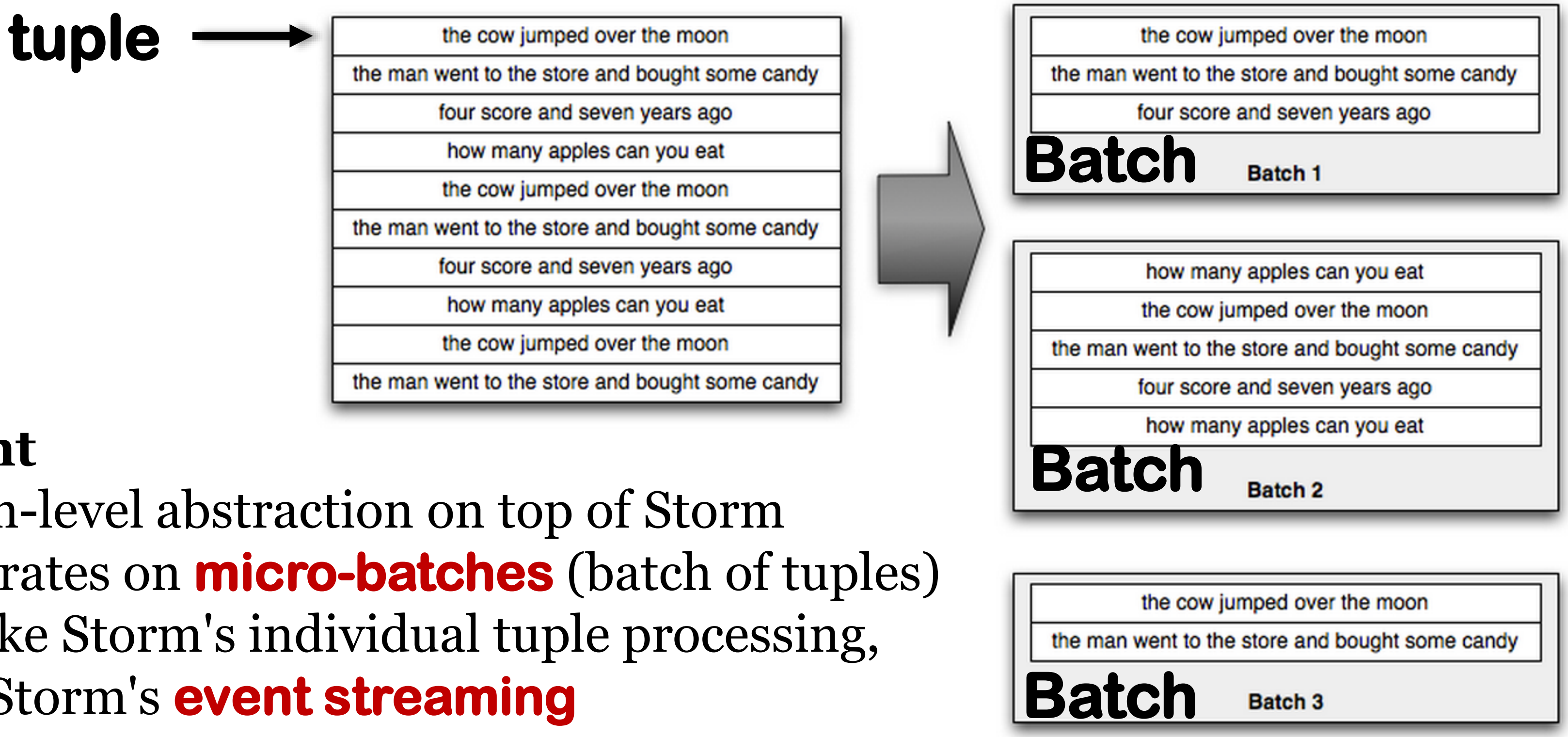
- Originated at Backtype, acquired by Twitter in 2011
- Open Sourced: late 2011
- Part of Apache Incubator: since September 2013

- Written in Closure
- Multi-lingual support: Java, Closure, Python, Scala, Rubi, others

- Scalable
- Fault-tolerant
- Simplifies working queues & workers
- Complementary to Hadoop, batch processing system

What is Storm **Trident**?

Trident ≡ Micro-batching of Storm



the cow jumped over the moon
the man went to the store and bought some candy
four score and seven years ago
how many apples can you eat
the cow jumped over the moon
the man went to the store and bought some candy
four score and seven years ago
how many apples can you eat
the cow jumped over the moon
the man went to the store and bought some candy

Batch — Batch 1

the cow jumped over the moon
the man went to the store and bought some candy
four score and seven years ago

Batch — Batch 2

how many apples can you eat
the cow jumped over the moon
the man went to the store and bought some candy
four score and seven years ago
how many apples can you eat

Batch — Batch 3

the cow jumped over the moon
the man went to the store and bought some candy

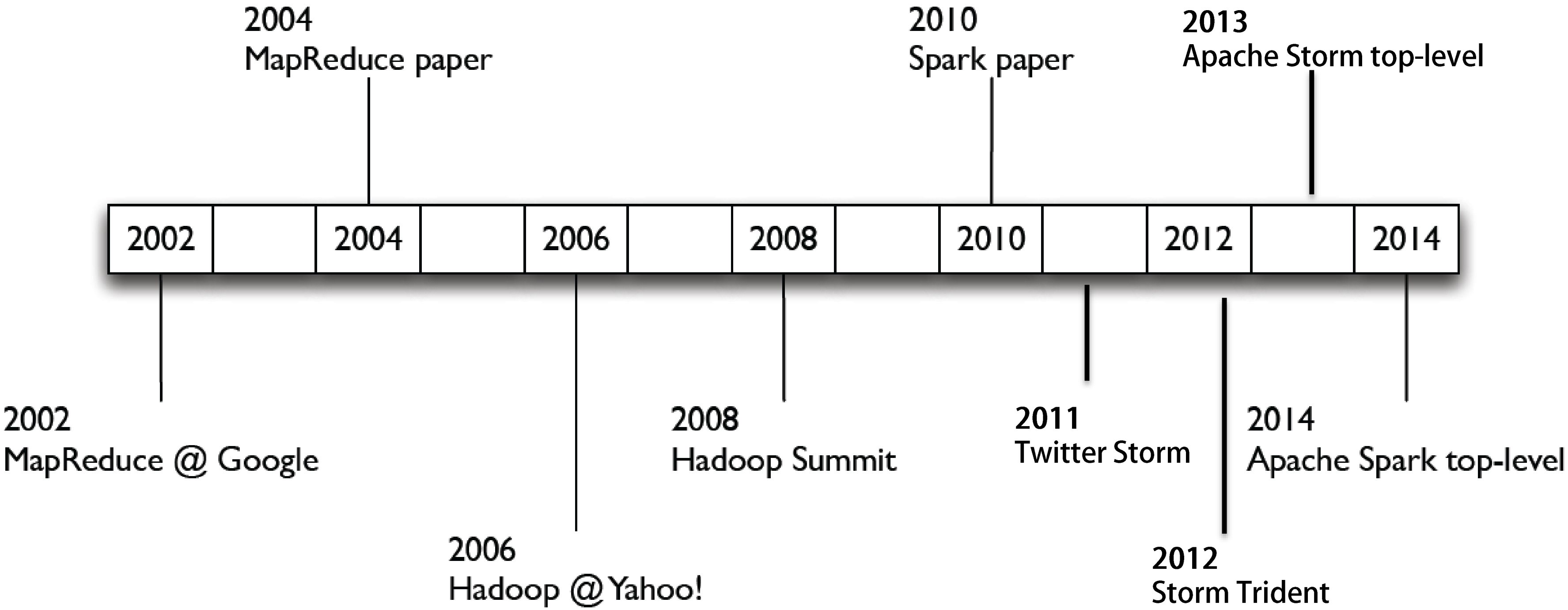
Trident

- High-level abstraction on top of Storm
- Operates on **micro-batches** (batch of tuples) unlike Storm's individual tuple processing, i.e. Storm's **event streaming**

Src: https://storm.apache.org/documentation/Trident-tutorial.html

Functional Programming for Big Data

Brief History



Spark Streaming vs Storm vs Trident

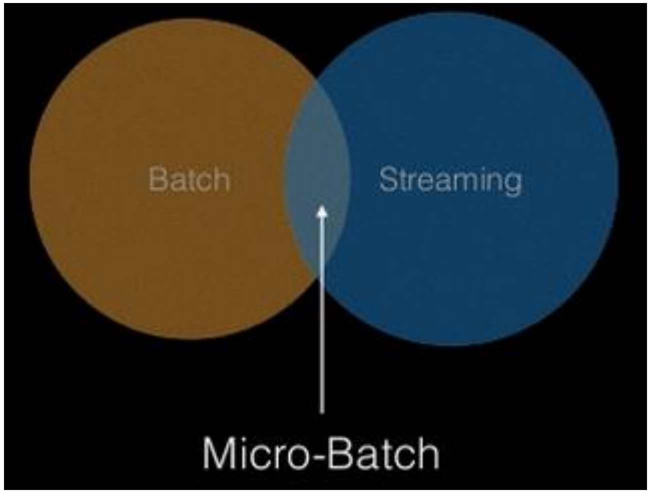
Choose Your Own Weapon!

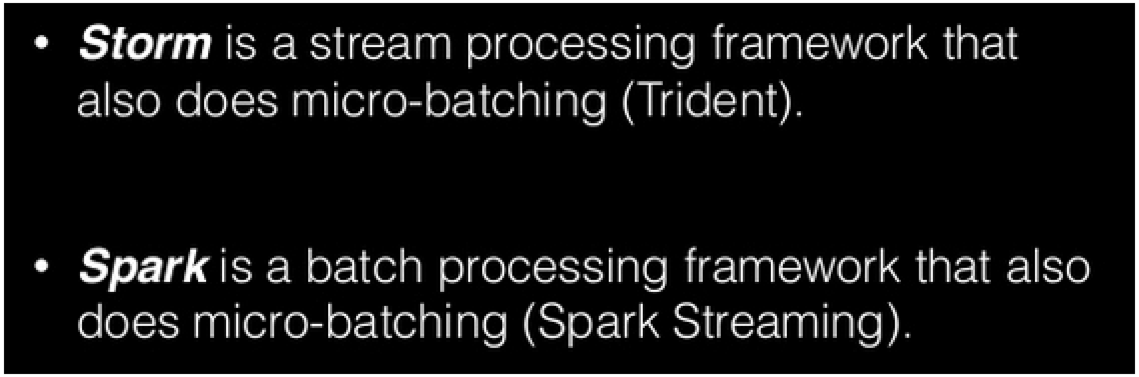
Feature	Spark Streaming	Storm Core	Storm Trident
Processing Model	Micro-Batching (Batch: Spark Core)	Event-streaming	Micro-batching
Delivery Semantics	Exactly once	At most once / At least once	Exactly Once
Latency	seconds	sub-seconds	seconds
Stream Source	HDFS, network (eg, Kafka)	Spout	Spout, Trident Spout
Computation	Transformations, Actions, Window	Bolt	Filters, Aggregations, Functions, Joins
Stateful Ops	RDD resilience w/ lineage)	No	Yes
Fault Tolerance (FT)	Master FT w/ checkpoint file in HDFS; worker FT w/ lineage	Worker FT w/ checkpointing	Worker FT w/ checkpointing
Throughput	++++	++	++++
Multi-language support	Scala, Java, Python, R	Java, Closure, Scala, Python, Rubi	Java, Closure, Scala

Spark Streaming vs Storm vs Trident

Feature	Spark Streaming	Storm Core	Storm Trident
Stream primitive	DStream	Tuple	Tuple, Tuple Batch, Partition
Output Persistence	foreachRDD	Bolts	State, MapState
Resource Manager	Yarn, Mesos	Yarn, Mesos	Yarn, Mesos
Distributed Remote Procedure Call	No	Yes	Yes
Community	++++	++	++

Batch vs. Streaming





- ***Storm*** is a stream processing framework that also does micro-batching (Trident).

- ***Spark*** is a batch processing framework that also does micro-batching (Spark Streaming).

Input Data Stream Sources

Stream Source

- Source of tuples; emits tuples from the data stream source
- Can be run in reliable or unreliable modes
- Data system may/may not handle multiple data stream sources concurrently

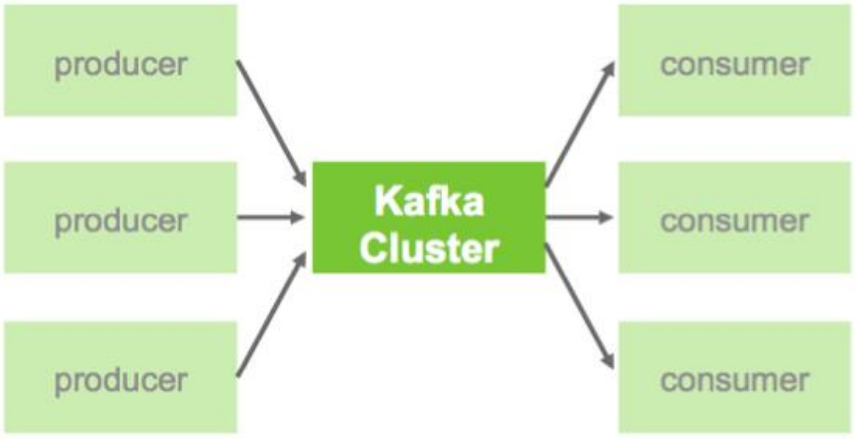
Type	Definition	Exemplars
Unreliable	No means to replay a previously received events	Twitter, MongoDB, Scribe
Reliable	Supports replay of previous events if processing fails at any point	Amazon Kinesis, Amazon SQS, Kestrel, JMS, AMQP
Durable	Supports replay of any events/set of events for the selection criteria	**Apache Kafka**

Apache Kafka

Apache Kafka
A high-throughput distributed messaging system.

Apache Kafka is a distributed replicated, multi-topic publish subscribe message system (http://kafka.apache.org/):

- Designed for processing of **high-volume**, **real-time** activity stream data (logs, metrics collection, social media streams, etc.)
- Developed at LinkedIn, now part of Apache
- Maintains feeds of messages in **topics**
- **Low-latency** message delivery
- **Fault-tolerance** guarantee in the presence of machine failure
- Download: http://apache.mirrors.pair.com/kafka/0.8.2.0/kafka_2.10-0.8.2.0.tgz
- Extract. Refer to http://kafka.apache.org/documentation.html#quickstart



APACHE STORM™
Distributed • Resilient • Real-time

Zookeeper

Zookeeper is a dependency for Kafka:

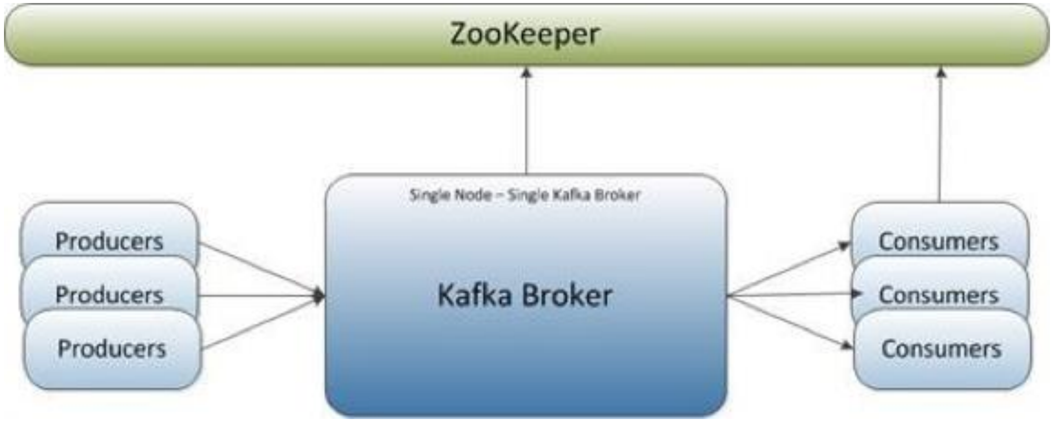
- Download: http://apache.osuosl.org/zookeeper/zookeeper-3.4.6/zookeeper-3.4.6.tar.gz
- Extract. Refer to http://zookeeper.apache.org/doc/trunk/zookeeperStarted.html
- Start zookeeper (to configure, see conf/zoo.cfg or create from conf/zoo_sample.cfg)

```
$ bin/zkServer.sh start
$ top zookeeper
$ bin/zkServer.sh stop
```

- Kafka comes with built-in zookeeper. Start zookeeper as:

```
$  bin/zookeeper-server-start.sh config/zookeeper.properties
```

Tune config to avoid a JVM error for insufficient memory: http://stackoverflow.com/questions/21448907/kafka-8-and-memory-there-is-insufficient-memory-for-the-java-runtime-environme



Kafka Operations: http://kafka.apache.org

- **To start Kafka:**

```
$ bin/kafka-server-start.sh config/server.properties
```

- **List Kafka topics:**

```
$ bin/kafka-topics.sh --list --zookeeper localhost:2181
```

- **Create a Kafka topic:**

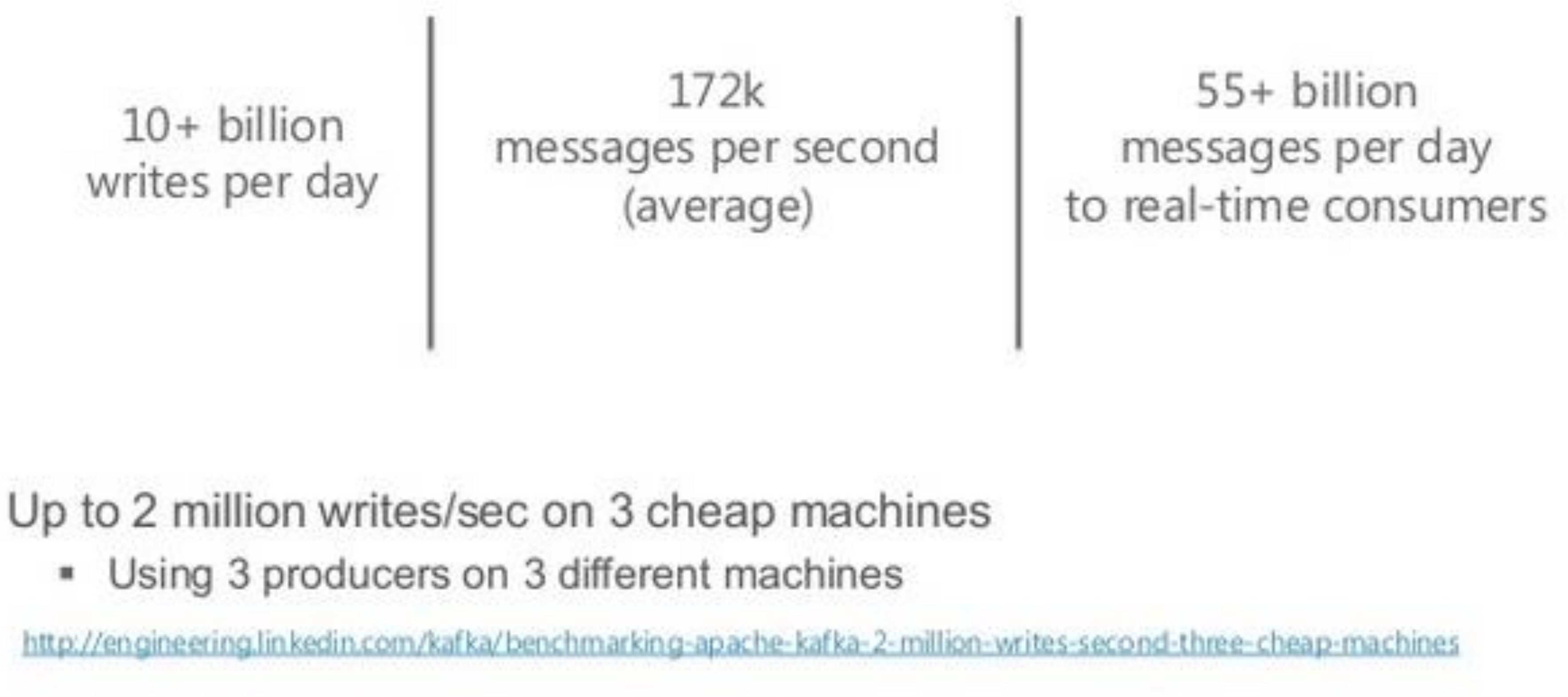
```
$ bin/kafka-topics.sh --topic <topicname> --create --zookeeper
localhost:2181 --partitions 1 --replication-factor 1
```

- **Delete a Kafka topic:**

```
$ bin/kafka-topics.sh --topic <topicname> --delete --zookeeper
localhost:2181
```

Note: This is not a clean delete. You cannot create a new topic with the same name after it is deleted.

Kafka Performance



10+ billion
writes per day

172k
messages per second
(average)

55+ billion
messages per day
to real-time consumers

Up to 2 million writes/sec on 3 cheap machines

- Using 3 producers on 3 different machines

http://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

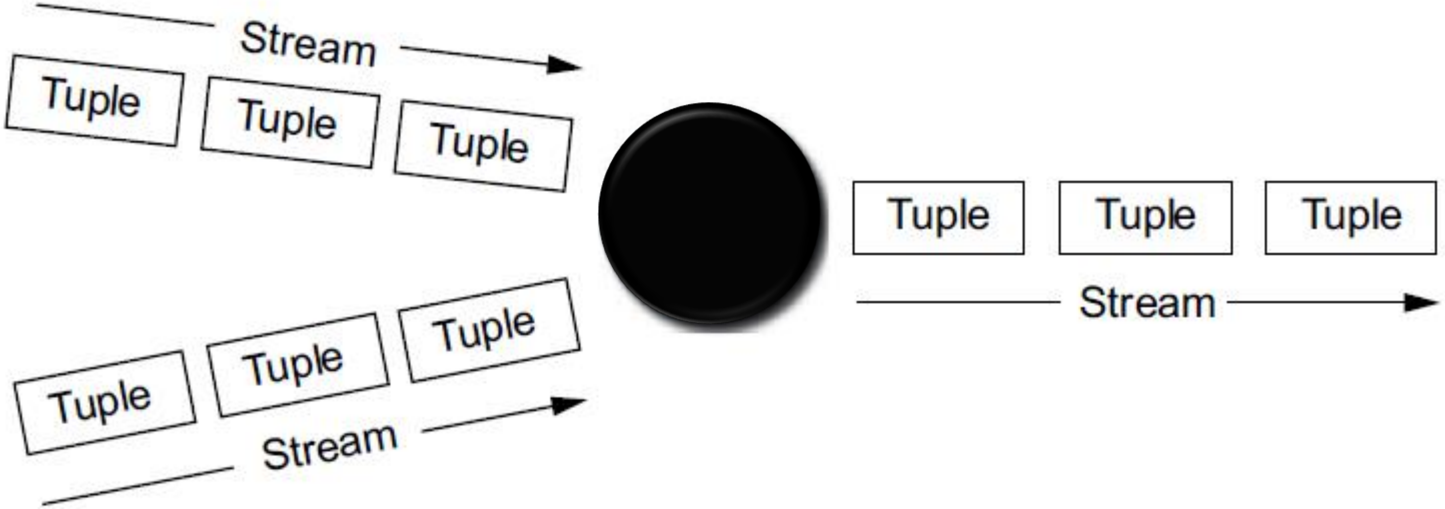
Output Data Stream Sink

- Cassandra
- HBase
- HDFS
- Kafka
- Redis
- Memcached
- R
- JMS
- MongoDB
- RDBMS

Operators over Data Stream(s)

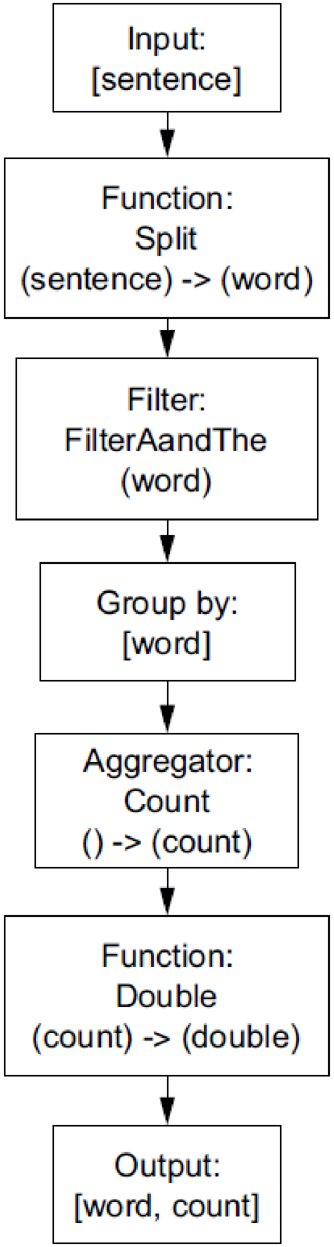
Operators: Transformations, Actions

- Consumes 1+ streams and possibly produces new streams
- Operations: Calculate, filter, aggregate, join, interface w/ a database



Pipe Diagram: Stream Process Abstraction

Pipe Diagram ≡ High-level abstraction of processing stream tuples



Pipe Diagram

- **Defines a sequence of operations** in terms of
  - **Functions**
  - **Filters**
  - **Aggregators**
  - **Joins**
  - **Merges**
- **Specifies** which **tuple fields** are consumed (input), produced (output), or discarded as the tuples flow through the various steps of the pipe diagram

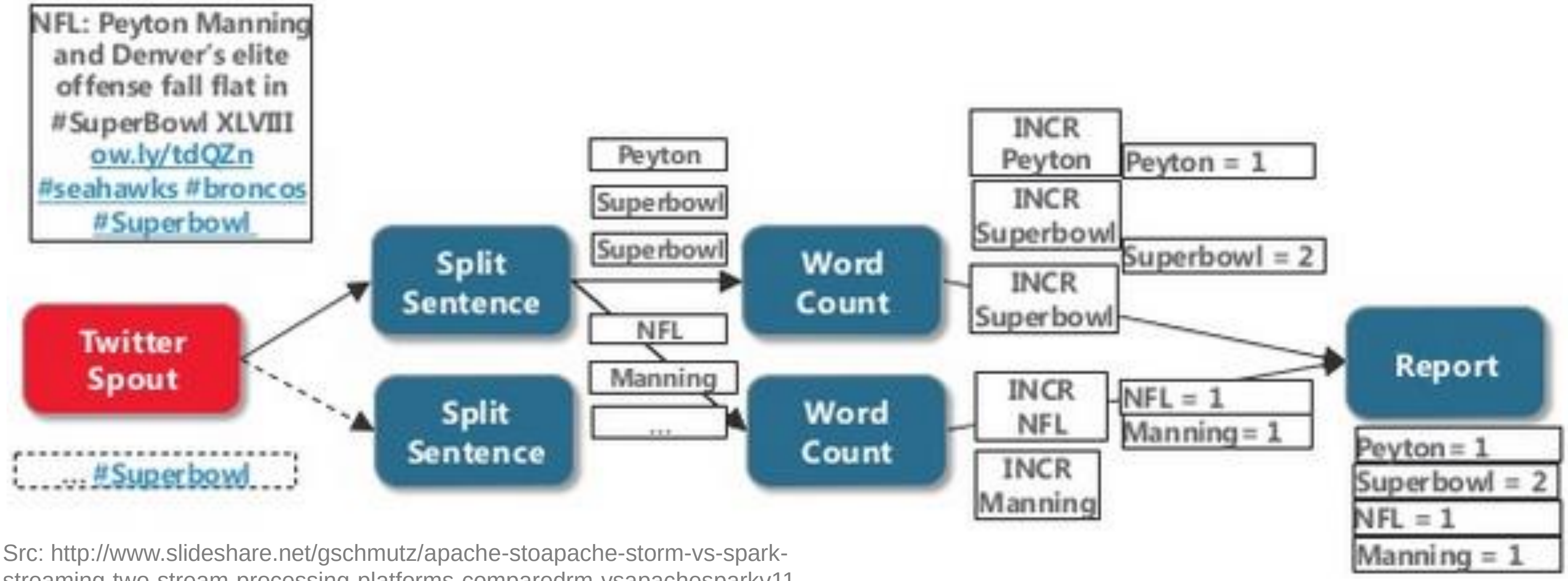
Fig 6.15 in Nathan Marz' Big Data Book

Analytic Workflow (AW): DAG

AW ≡ Graph of computation over data stream(s)

DAG/Topology/Lineage Graph

- Wires data stream sources and operations (filters, aggregators, functions, joins, merges) via a DAG (directed acyclic graph)
- A connected network of pipe diagrams
- Executes on many machines; each node runs multiple instances in parallel

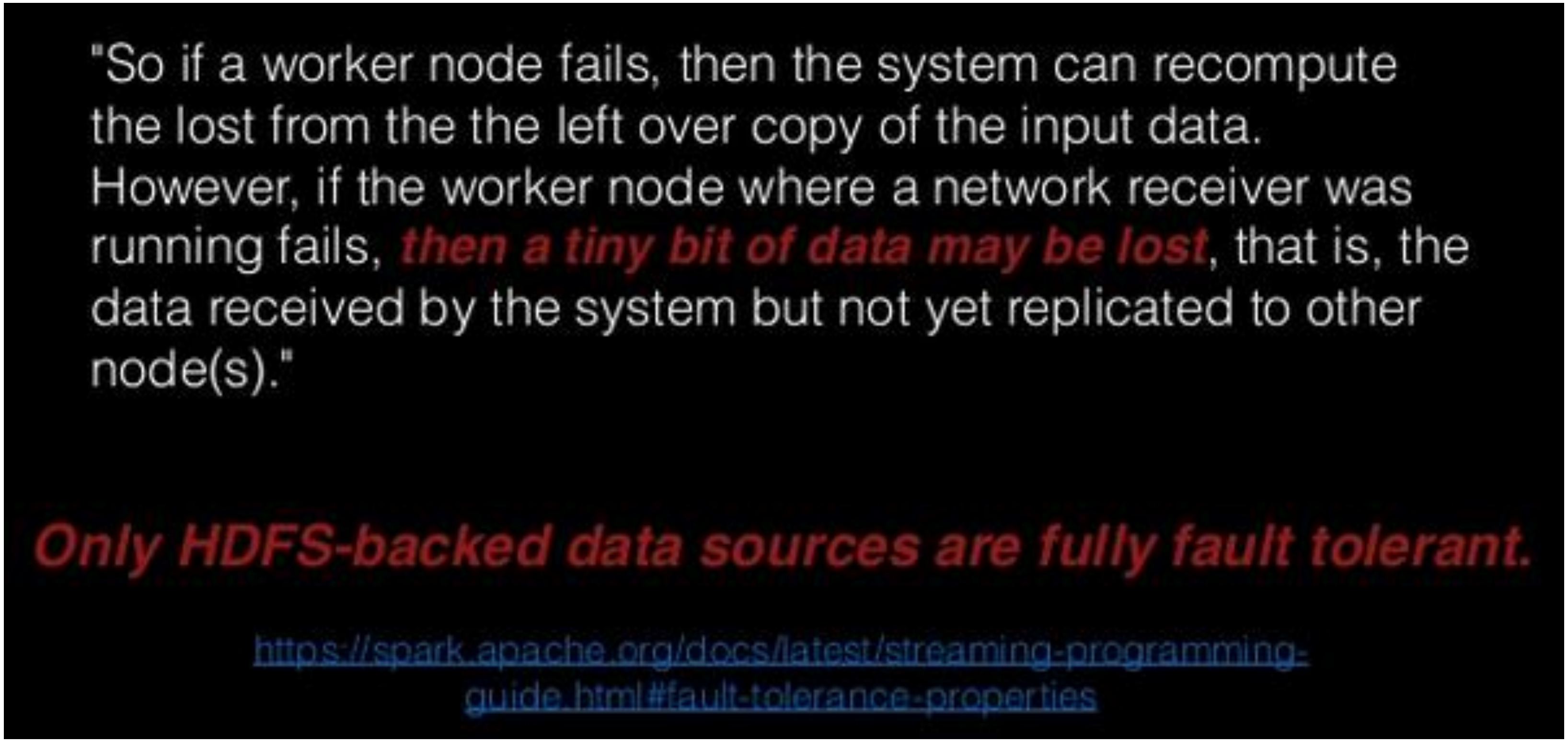


Src: http://www.slideshare.net/gschmutz/apache-stoapache-storm-vs-spark-streaming-two-stream-processing-platforms-comparedrm-vsapachesparkv11

Fault Tolerance

- If a node fails, who will reassign that node's tasks to other nodes and how task completion is being tracked
- What happens with any tuples/batches sent to a failed node (e.g., will time out and be replayed)
- Delivery guarantees are only dependent on a reliable data source

Spark: Fault Tolerance

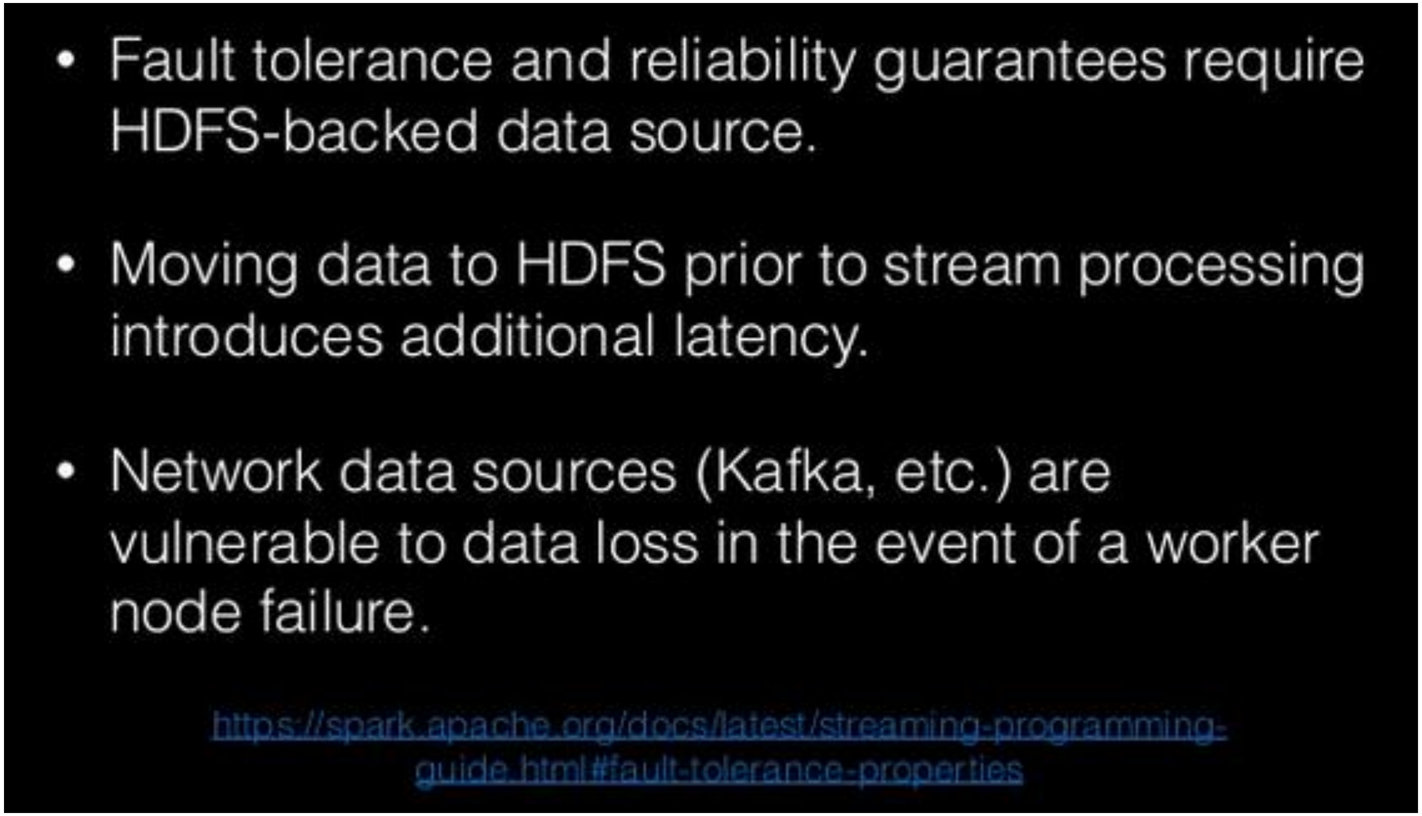


"So if a worker node fails, then the system can recompute the lost from the the left over copy of the input data. However, if the worker node where a network receiver was running fails, ***then a tiny bit of data may be lost***, that is, the data received by the system but not yet replicated to other node(s)."

Only HDFS-backed data sources are fully fault tolerant.

https://spark.apache.org/docs/latest/streaming-programming-guide.html#fault-tolerance-properties

Spark Streaming: Reliability Limitations



- Fault tolerance and reliability guarantees require HDFS-backed data source.
- Moving data to HDFS prior to stream processing introduces additional latency.
- Network data sources (Kafka, etc.) are vulnerable to data loss in the event of a worker node failure.

https://spark.apache.org/docs/latest/streaming-programming-guide.html#fault-tolerance-properties

Performance Benchmarks: What matters?

Latency

Is performance of streaming application paramount?

Delivery Guarantees

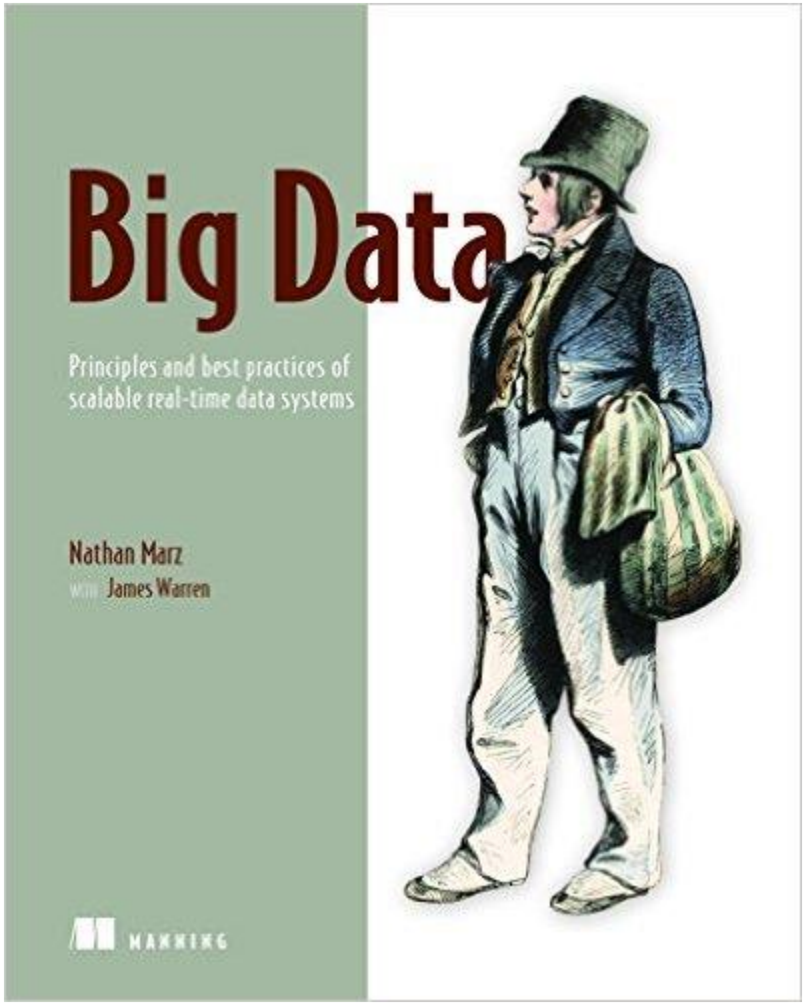
How important to process every single datum?
Is normal amount of data loss acceptable?

Fault Tolerance

Is high availability of primary concern?

Lambda Architecture

Read Bonus Chapters (1.7 and 18) from the book by Nathan Marz


Big Data
Principles and best practices of scalable real-time data systems
Nathan Marz
with James Warren
MANNING