Near Real-Time Stream Processing Architectures

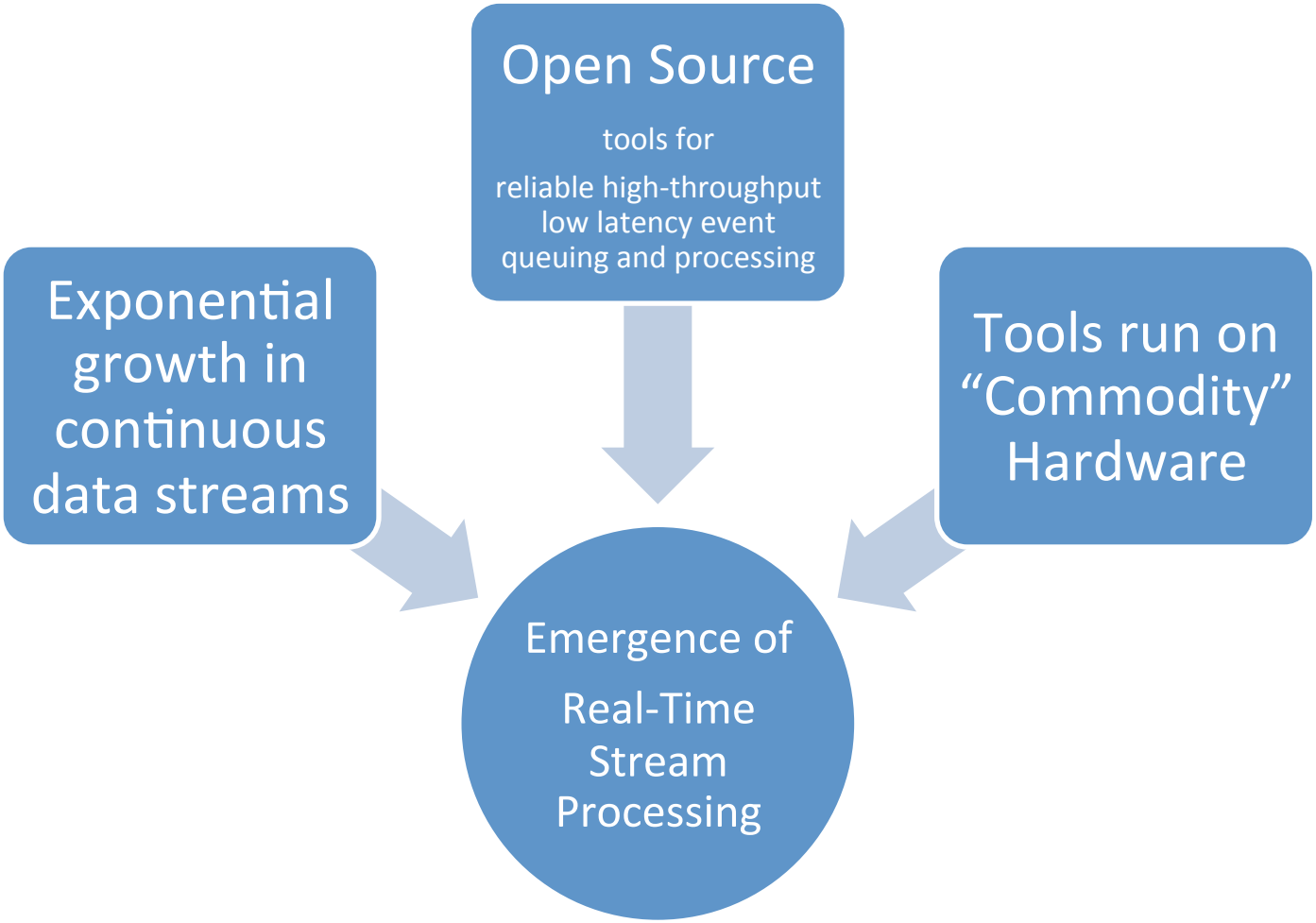
Anand Iyer

Senior Product Manager, Cloudera

aiyer@cloudera.com

Its 2015: Your CTO wants Real-Time

Why now? Complex Event Processing (CEP) is not a new concept.



Use Cases Across Industries

Credit Card & Monetary Transactions

Identify fraudulent transactions as soon as they occur.

Healthcare

Continuously monitor patient vital stats and proactively identify at-risk patients.

Retail

- Real-time in-store Offers and Recommendations.
- Email and marketing campaigns based on real-time social trends

Digital Advertising & Marketing

Optimize and personalize digital ads based on real-time information.

Consumer Internet, Mobile & E-Commerce

Optimize user engagement based on user's current behavior. Deliver recommendations relevant "in the moment"

Manufacturing

- Identify equipment failures and react instantly
- Perform proactive maintenance.
- Identify product quality defects immediately to prevent resource wastage.

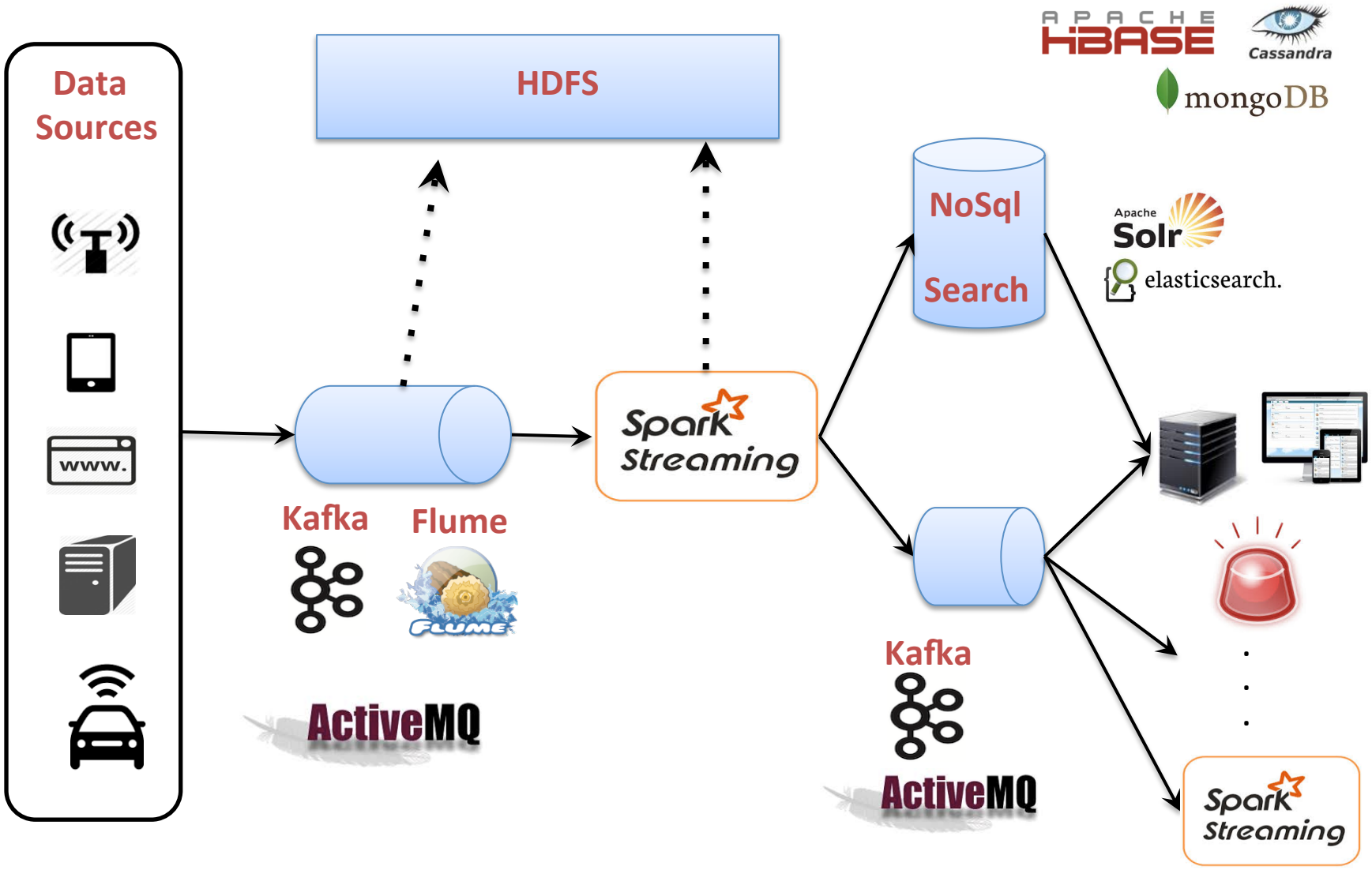
Security & Surveillance

Identify threats and intrusions, both digital and physical, in real-time.

Transportation & Logistics

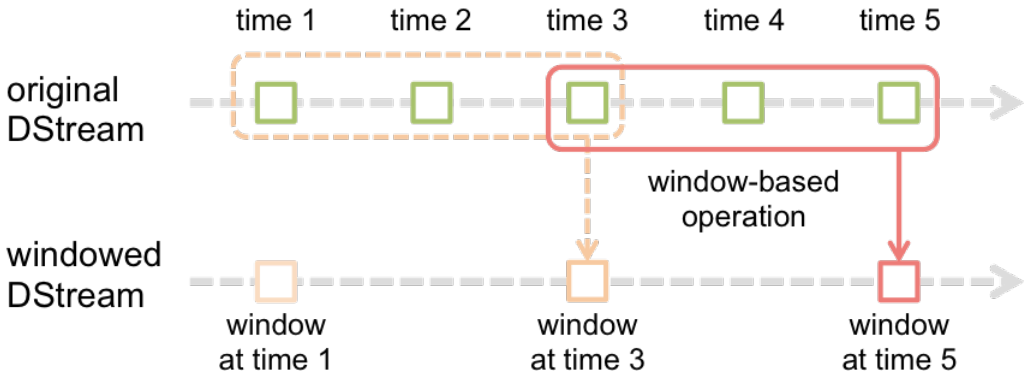
- Real-time traffic conditions
- Tracking fleet and cargo locations and dynamic re-routing to meet SLAs

Canonical Stream Processing Architecture



Operations on Sliding Windows

Easily define operations over a sliding window of data



Specify:

- Window length as multiple of micro-batch size
- Sliding step size

NOTE: Provide adequate memory to hold sliding window worth of data.

Maintain and update arbitrary state

updateStateByKey(...)

- Define initial state
- Provide state update function
- Continuously update with new information

Examples:

- Running count of words seen in text stream
- Per user session state from activity stream

Note:
Requires periodic check-pointing to fault-tolerant storage.

OSS options for Stream Processing

	Spark Streaming	Storm	Trident (built on Storm)	Samza
Architecture	micro-batch	one-at-a-time	micro-batch	one-at-a-time
Language Support	Scala, Java, Python	Java, Scala, Python, Ruby, Clojure…	Java, Clojure, Scala	Scala, Java
Resource Managers	YARN, Mesos, Standalone	YARN, Mesos	YARN, Mesos	YARN
Latency	~0.5 seconds	~100ms	~0.5 seconds	~100ms
Throughput	****	**	***	**
Age	2+	3.5+	1.5+	1+
Known Production Instances	50+ Multi-Vendor Support	50+ Multi-Vendor Support	??? Multi-Vendor Support	Outside LinkedIn only a handful. No vendors.

OSS options for Stream Processing

	Spark Streaming	Storm	Trident (built on Storm)	Samza
Exactly Once Processing*	Yes	No	Yes	No
Functions on Sliding Windows	Yes	No	Yes	No
Higher Order Functions (Aggregations, Joins, etc)	Yes. From Spark.	No.	Yes	No
State-full Operations	Yes.	No. Roll your own.	Yes.	Yes. But Node Local. Embedded RocksDb. Great if state exceeds memory, and volume of state updates is very high.

The Spark Streaming Advantage

- Automatically inherit developments in Spark
  - DataFrames
  - Mllib
  - Dynamic Resource Allocation
  - Vast ecosystem of “packages”

- Same framework for batch and streaming
  - Operational ease
  - Lambda Architectures are easy to implement

Exactly Once Processing

Should you care?

- In a cluster, machine failure is frequent
- "Double Counting" leads to False Positives:
  Alerting, predictive analytics, etc will have too many false positives when you double count data. You will end up "loosening" your thresholds

Thus, not a trivial consideration.

Exactly Once in Spark Streaming

Receiving Data:

- Use Kafka Direct receiver
- If offsets are fixed, can re-Create micro-batch RDD identically
- What if producer put dupes in Kafka? Generate UUID per event and dedupe.

Process Data:

- Deterministic DAG of operations

Output Processed Data:

- Failures can happen when only part of the output data is written
- Each micro-batch has a unique identifier: Batch-Time
- Use batch-time as key to perform “transactional writes”

Thank You!