

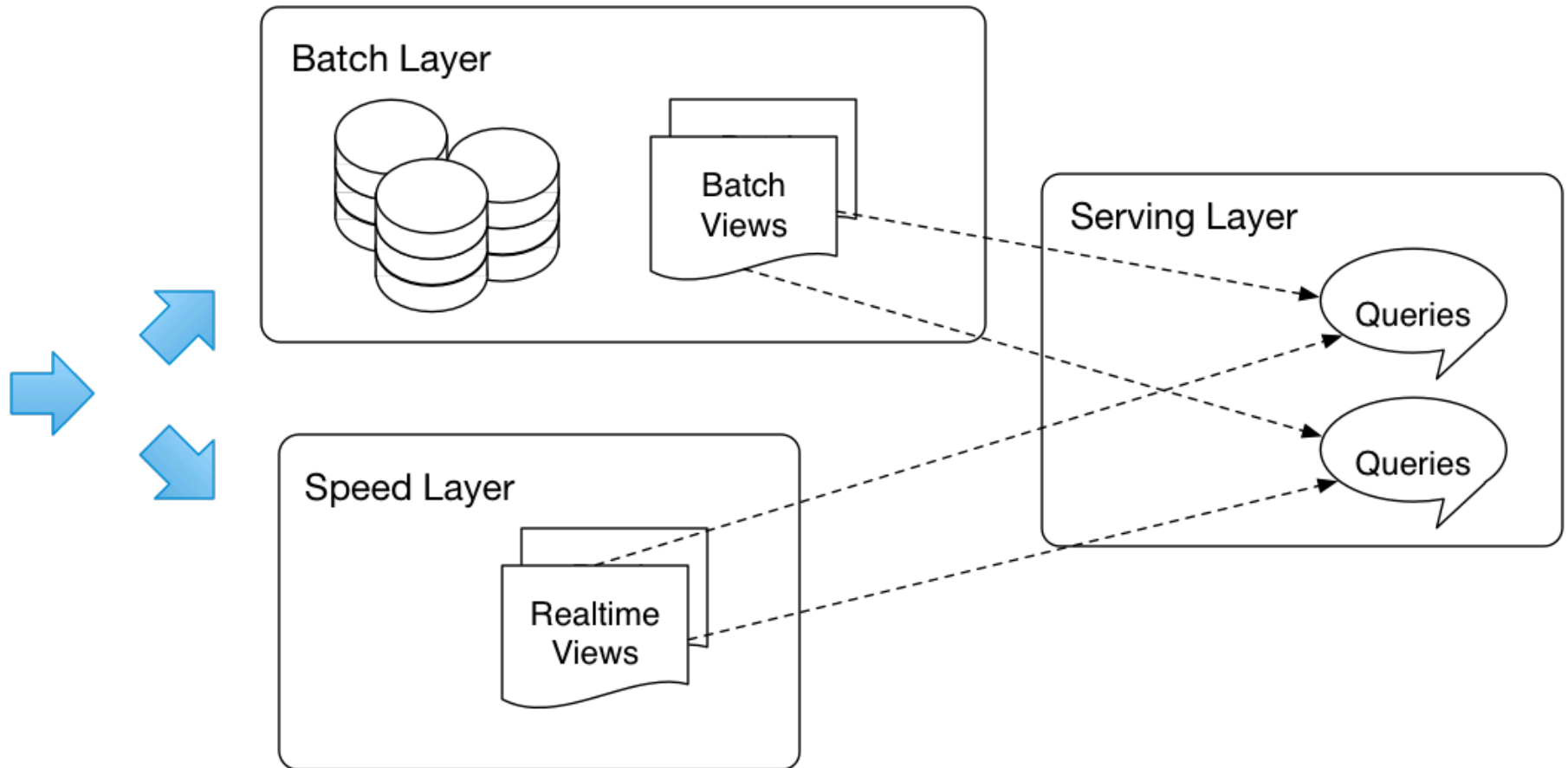
# Cloud Computing and Big Data

## Realtime Big Data

Oxford University  
Software Engineering  
Programme  
July 2017



# Recap on the Lambda Architecture



# Streaming

- Continuous data flow
  - “Unbounded streams of data”
- Usually uses a message distribution system
  - JMS
  - Apache Kafka
  - MQTT
  - Etc
- An unbounded set of events with time
  - $\langle t_1, E_1 \rangle, \langle t_2, E_2 \rangle, \dots, \langle t_n, E_n \rangle, \dots$

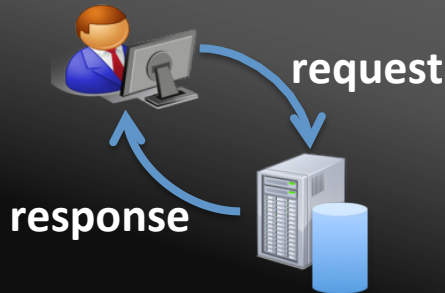
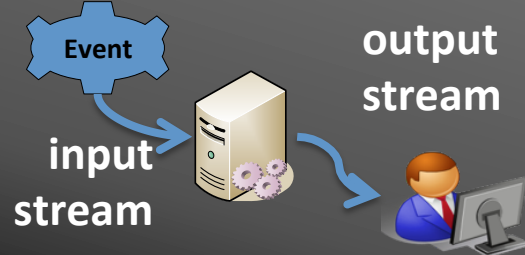


# Stream processing categorization

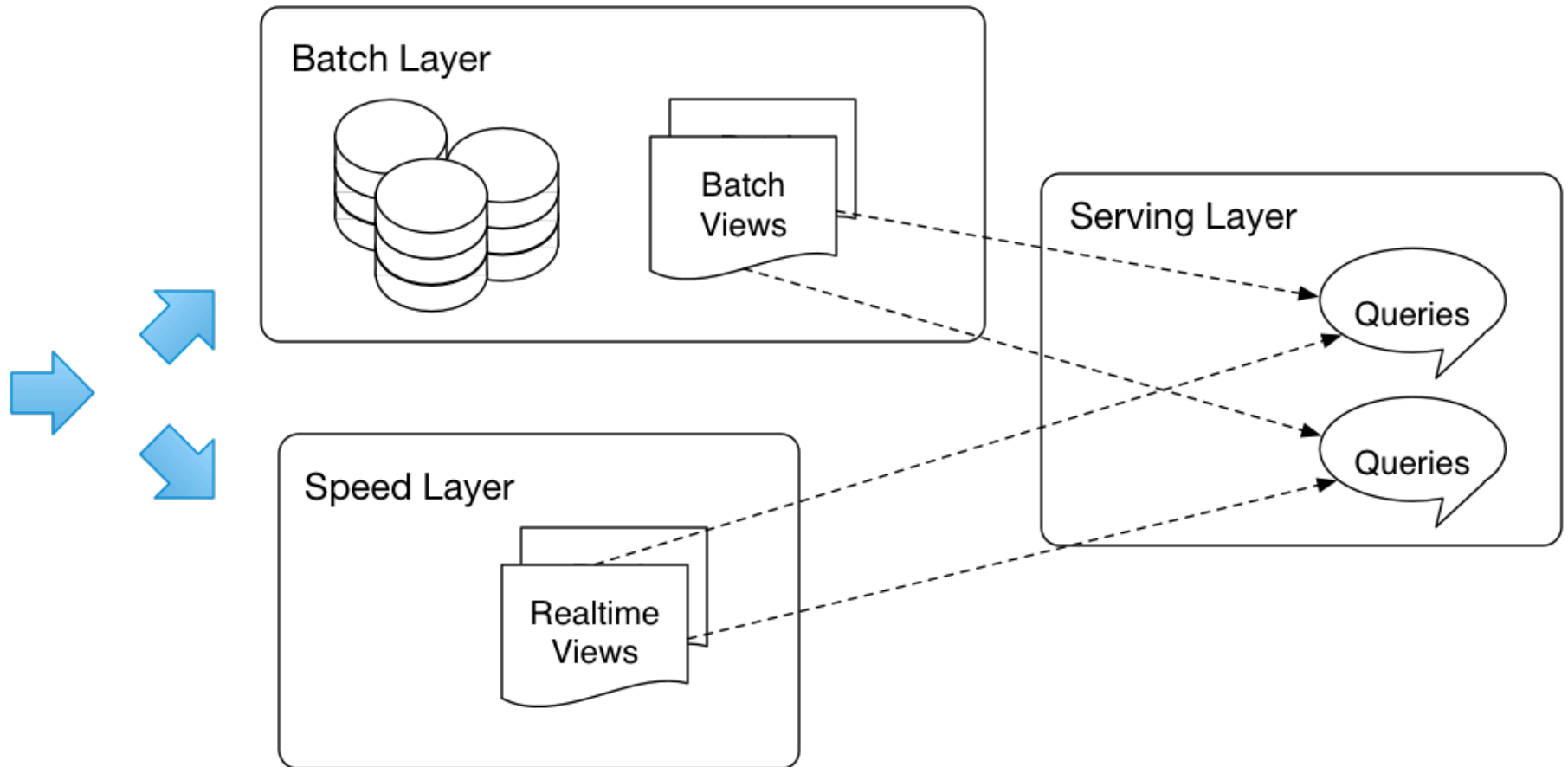
- Simple event processing
  - Working on an event at a time
    - e.g. filter out all events where the wind speed > 50 mph
- Event stream processing
  - Time-based processing of a single stream of events
    - Average wind speed over the last hour compared to the average over the last day
- Complex Event Processing
  - Correlation of events across different streams
    - Emergency calls correlated with wind speed in real time

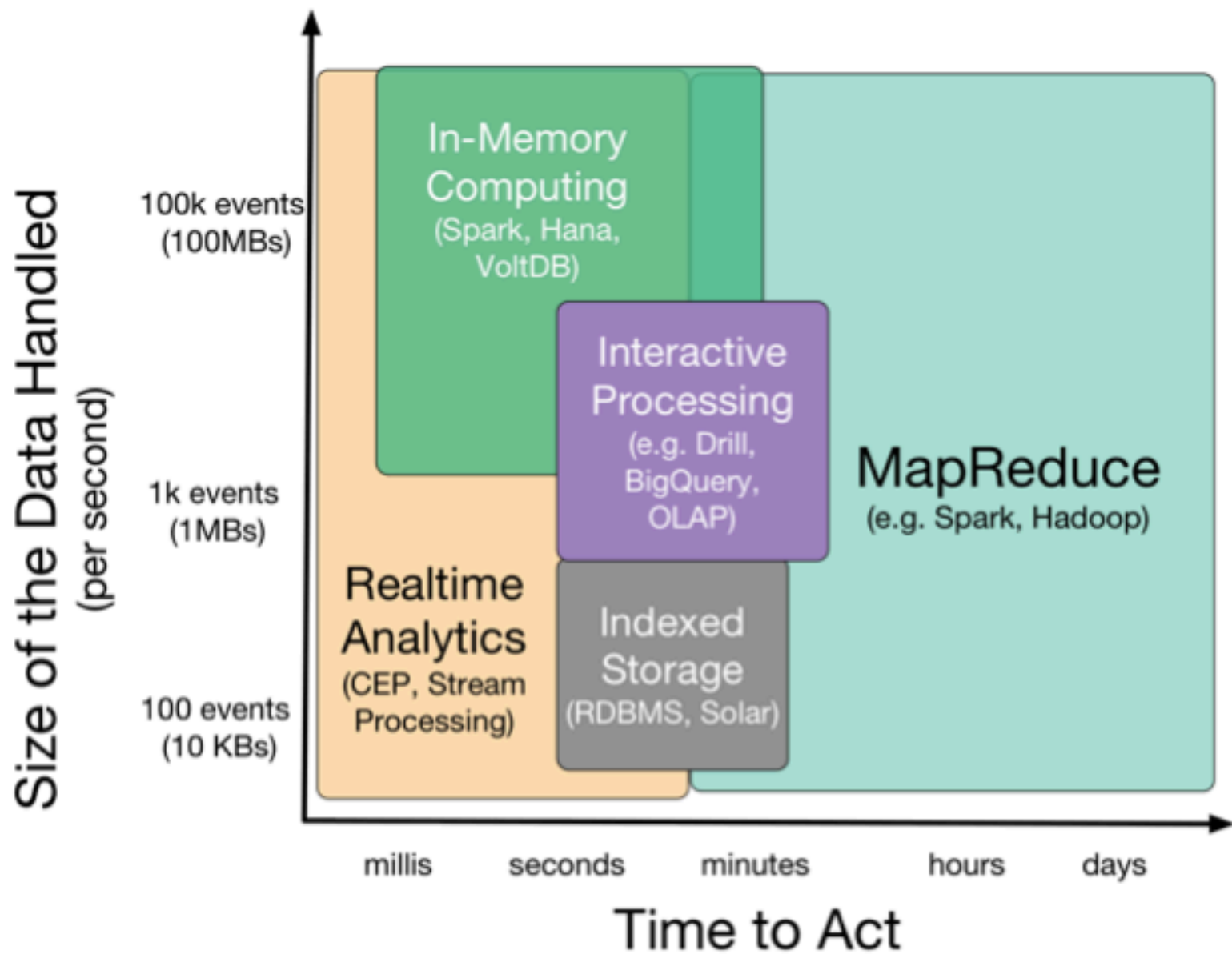


# Comparing Databases with Real-Time systems

	Database Applications	Event-driven Applications
Query Paradigm	Ad-hoc queries or requests	Continuous standing queries
Latency	Seconds, hours, days	Milliseconds or less
Data Rate	Hundreds of events/sec	Tens of thousands of events/sec or more
		

# Lambda Architecture





# Approaches to Streaming

- Pure streaming
  - Each event is processed as it comes in
- Micro-batch
  - Small batches of events are processed
  - Typically trades flexibility for performance
- Shared nothing
  - You can process events on any system in the cluster
- Stateful / Partitioned
  - The event must be processed on a system that has the correct state in memory





# Data distribution

- You need to get the events to the processing systems

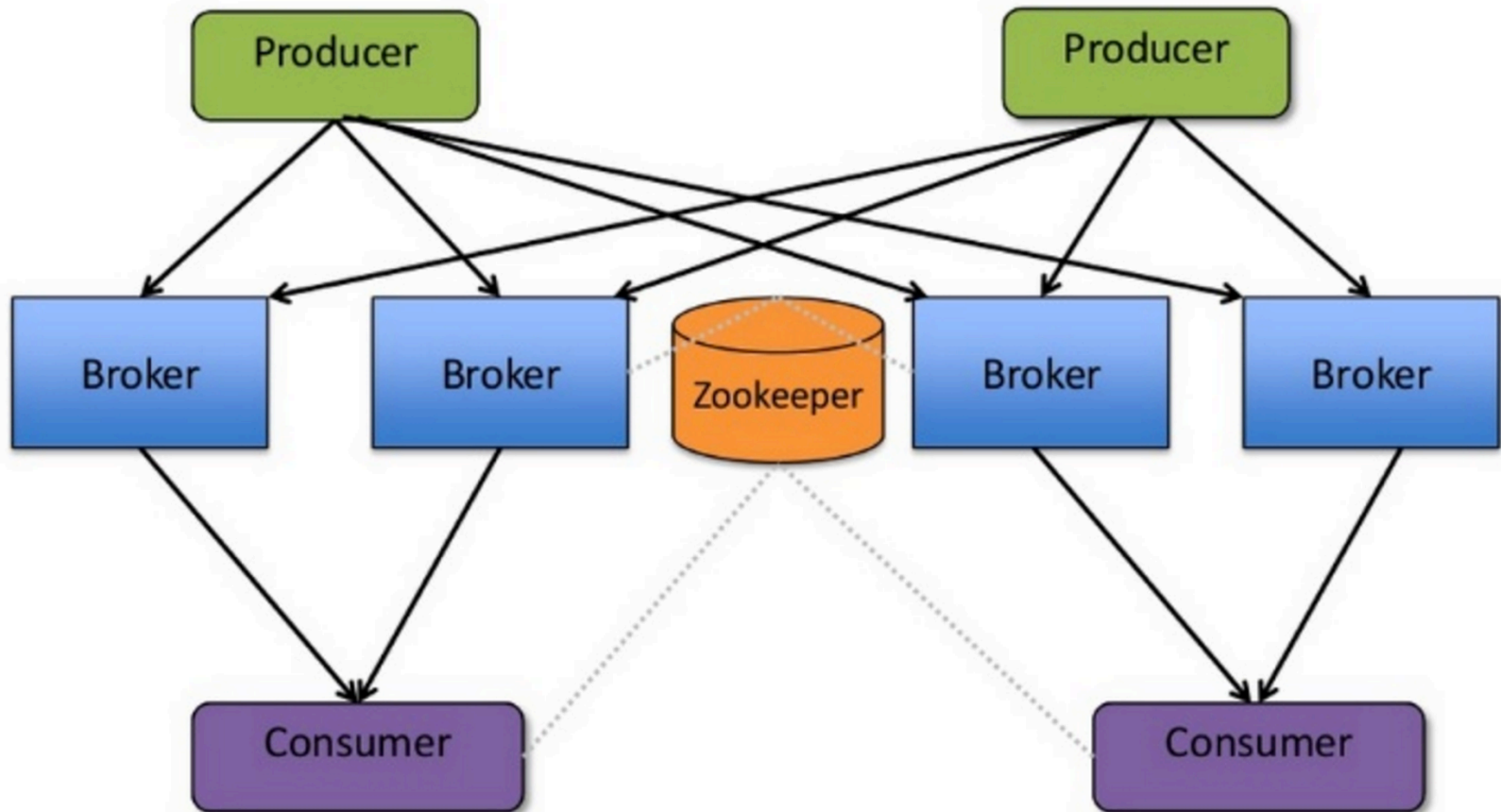


# MQTT

- Very simple, lightweight, fast
- No built in support for clustering / big-data
  - But can make up for it by being very fast
- Used a lot in IoT



# Apache Kafka



# Kafka

- Many of the approaches we've seen:
  - Partitioning
  - Multiple brokers
  - Elastically scalable
  - Supports clusters of co-ordinated consumers
  - Automatic re-election of leaders

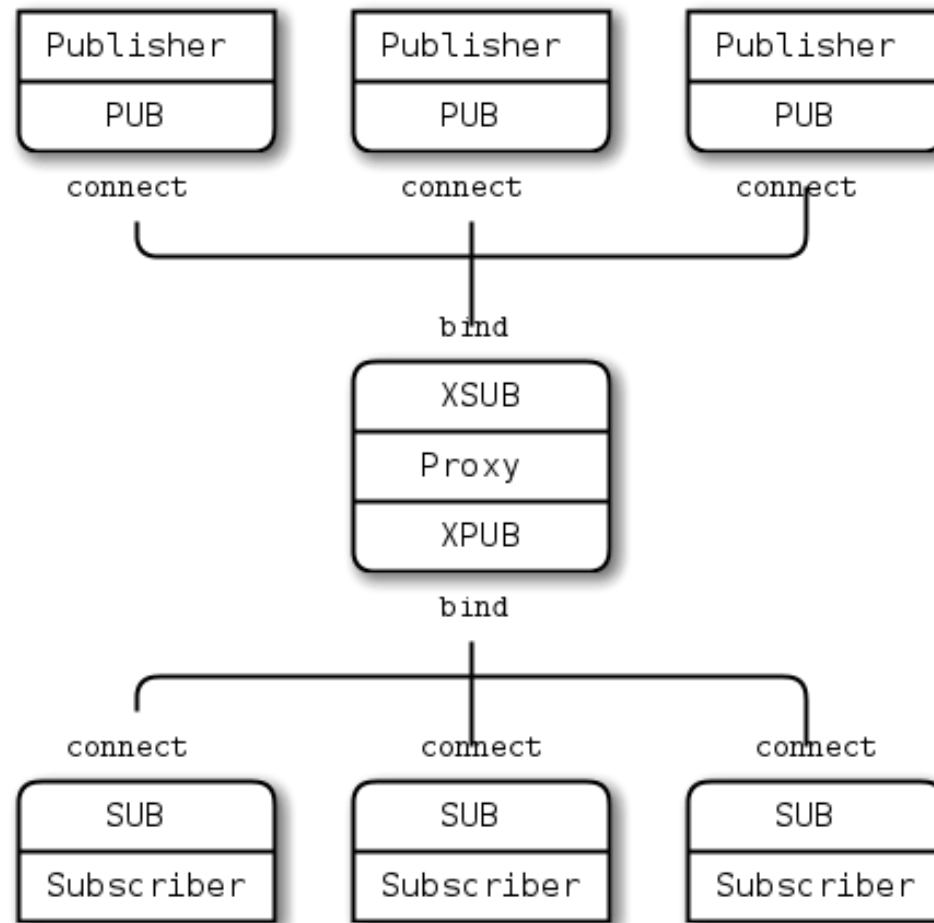


# Kafka exactly-once semantics



© Paul Fremantle 2015. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# ZeroMQ

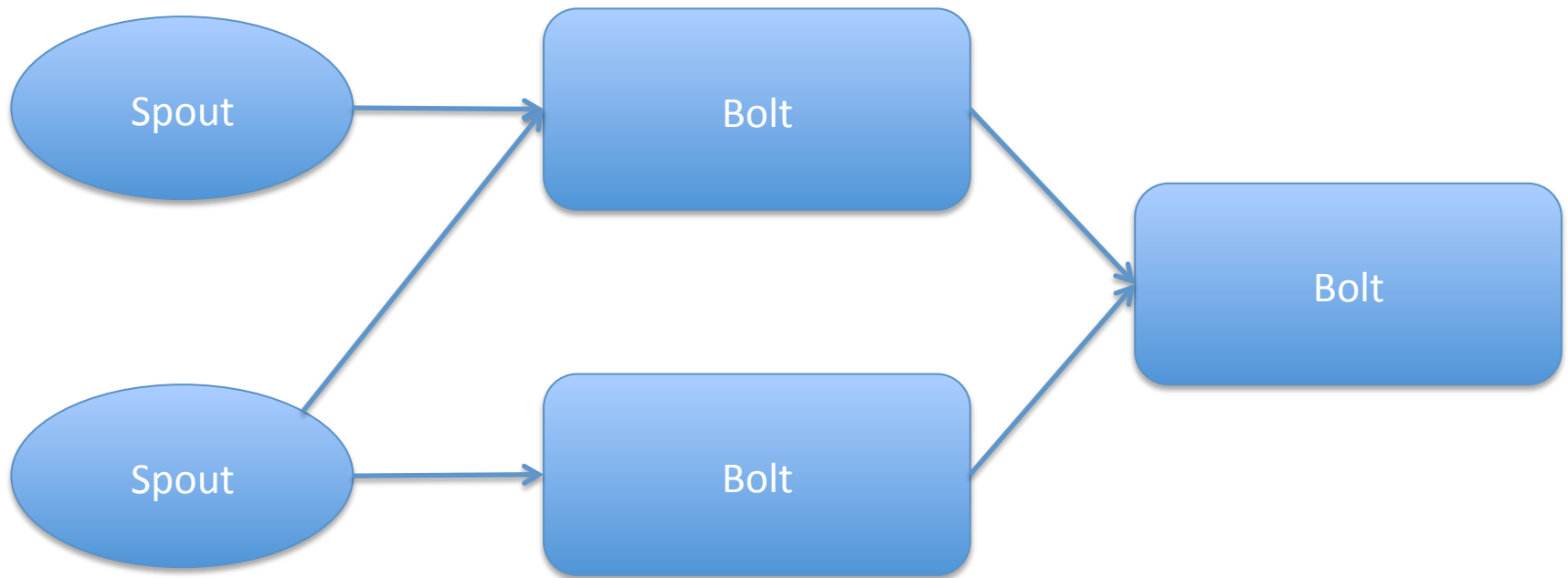


# Processing the data



© Paul Fremantle 2015. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Apache Storm



Note: another DAG

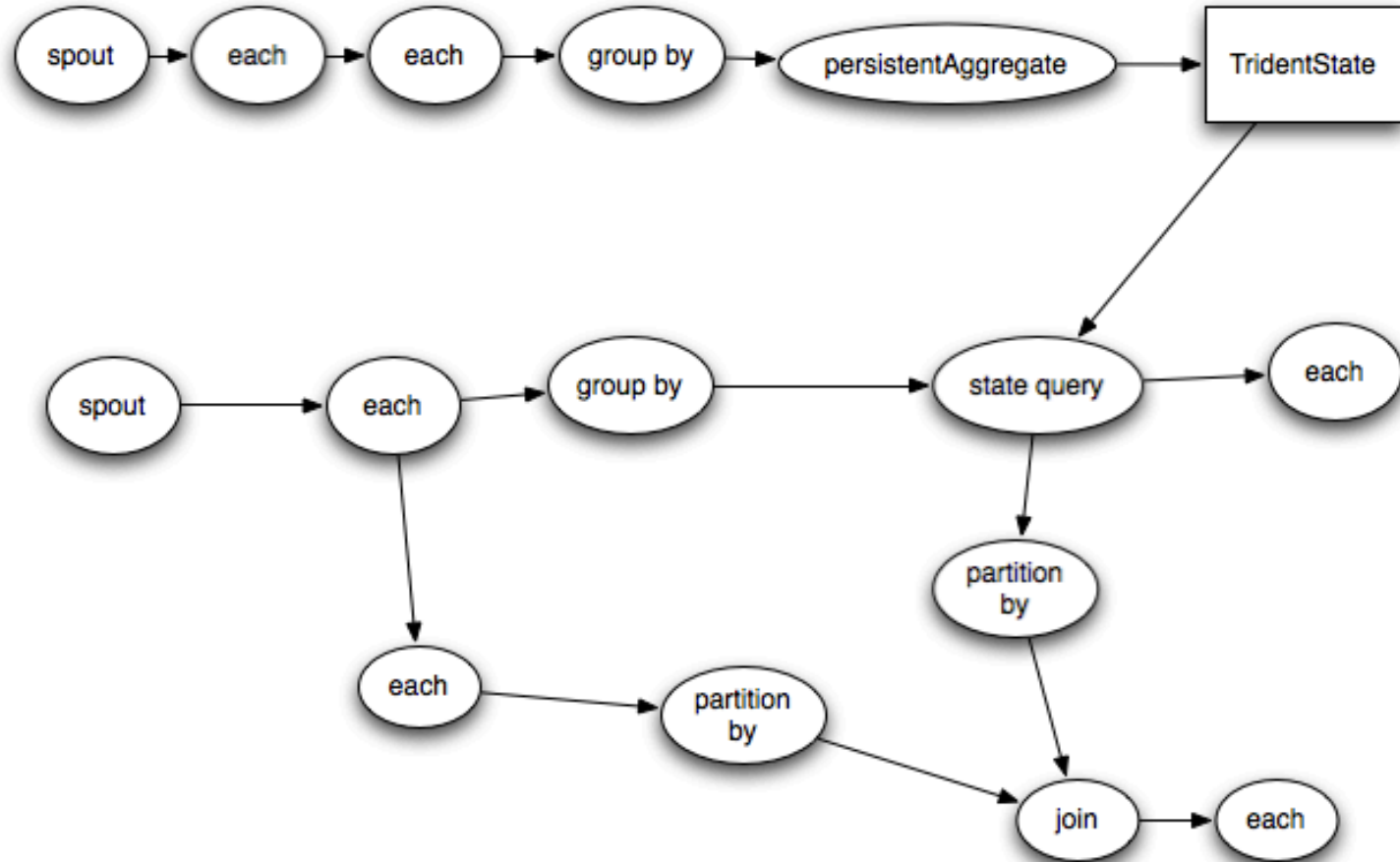


# Apache Storm

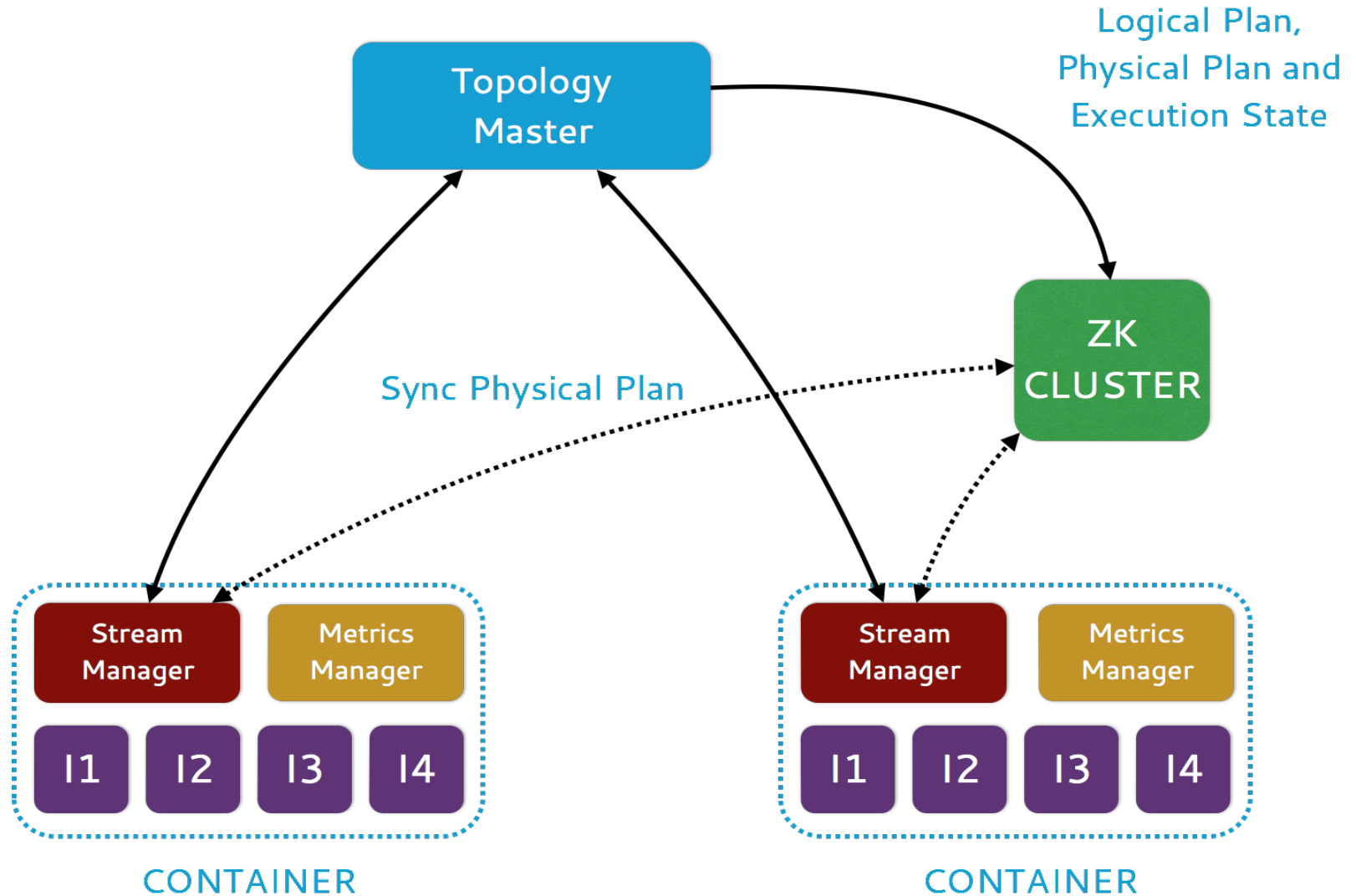
- Originally developed by BackType
  - Nathan Marz
- Acquired by Twitter
- Open Sourced and then donated to Apache
- Became a top level project in 2014
  - <http://storm.apache.org>



# Apache Storm Trident (micro-batch)



# Heron



# Heron: Key Features

- Fully API compatible with Apache Storm
- Task isolation
- Developer productivity
- Ease of manageability
- Use of mainstream languages C++/Java/Python



# Heron

- In production at Twitter for >2 years
- Going into production at Microsoft, WeChat
- Donation to CNCF

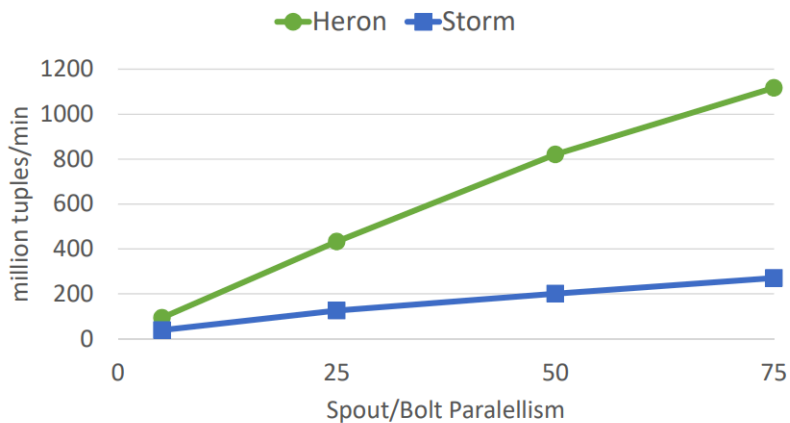


Fig. 2. Throughput with acks

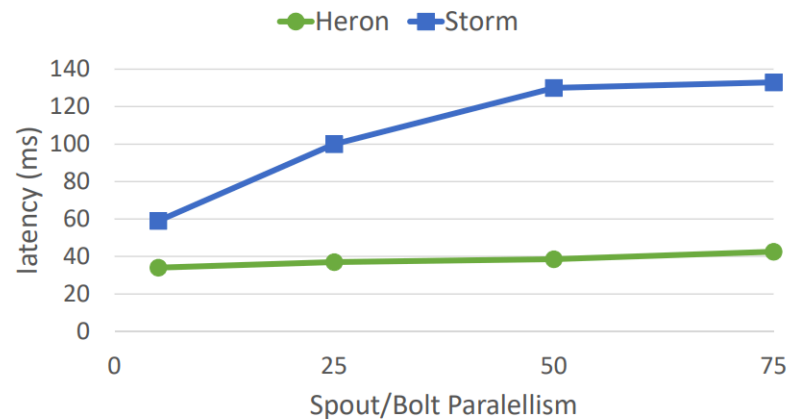
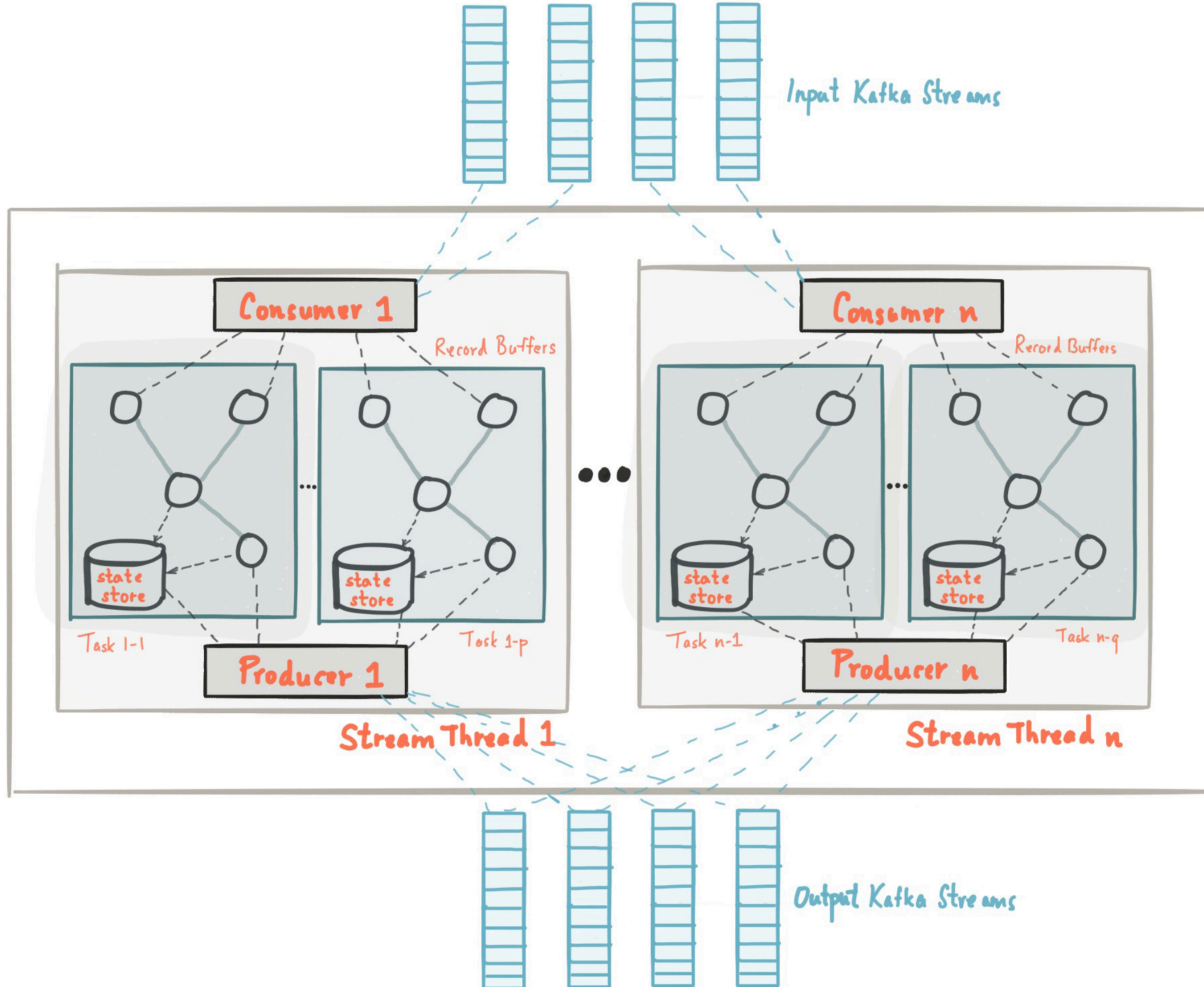


Fig. 3. End-to-end latency with acks

# Kafka Streams



# Kafka Streams

- Event-at-a-time processing (not microbatch) with millisecond latency
- Stateful processing including distributed joins and aggregations
- A convenient DSL
- Windowing with out-of-order data using a DataFlow-like model
- Distributed processing and fault-tolerance with fast failover
- Reprocessing capabilities so you can recalculate output when your code changes
- No-downtime rolling deployments



# Siddhi

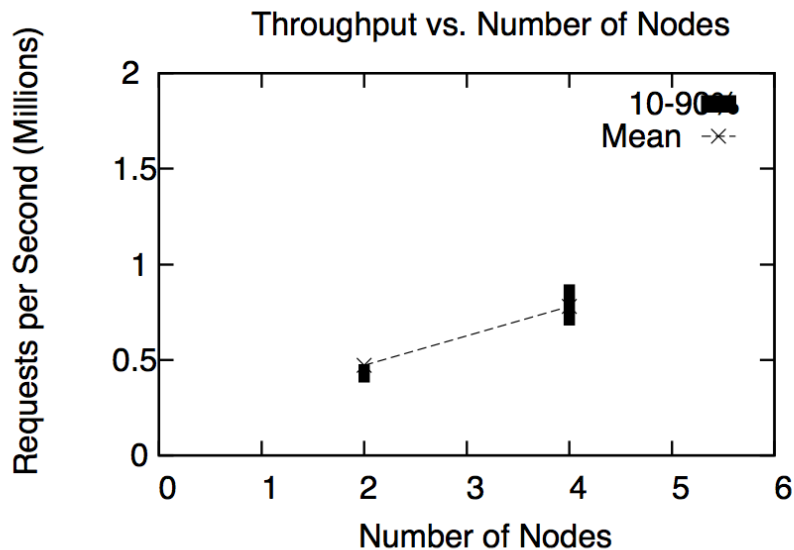
- A stateful query model
- SQL-like language for querying streams of data
  - Extended with **windows**
    - Time, Event count, batches
  - Partitioned
    - Based on data in the events
  - Pattern matching
    - A then B then C within window



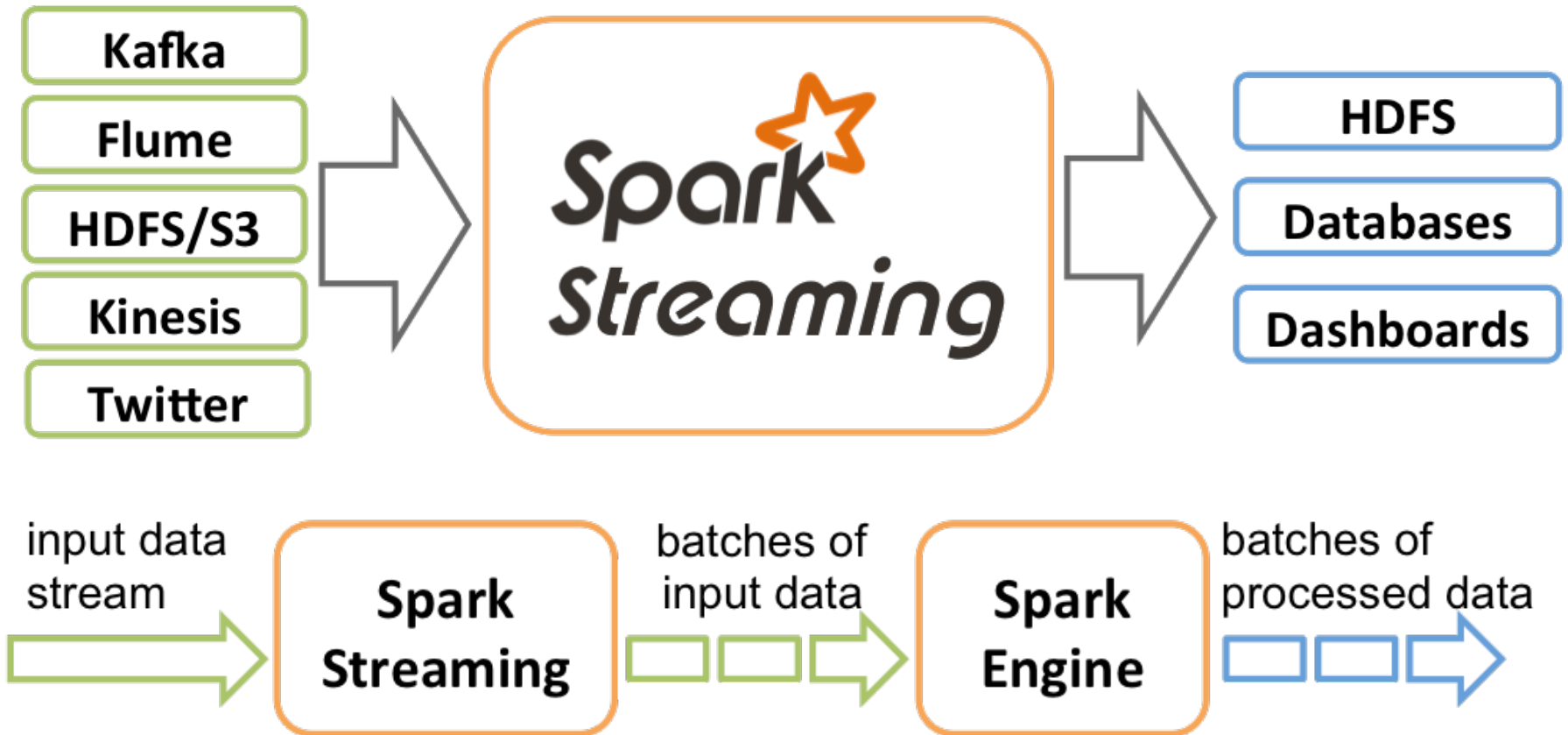


# Siddhi

- Apache Licensed Open Source on Github
  - <https://github.com/wso2/siddhi/>
- Pluggable into Storm and Spark
- Supports millions of events/sec
- [http://freo.me/DEBS\\_Siddhi](http://freo.me/DEBS_Siddhi)

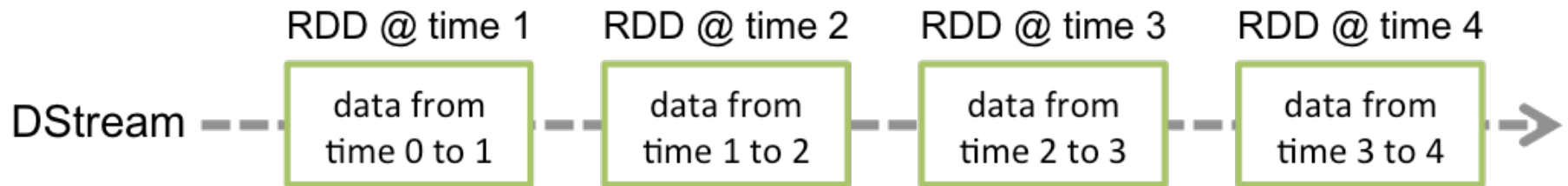


# Apache Spark Streaming

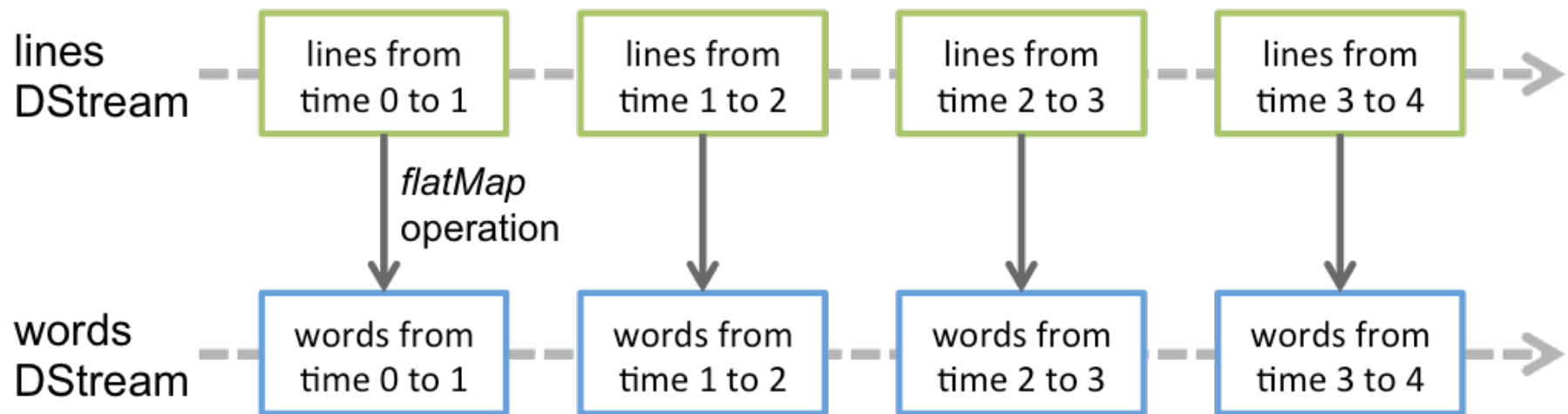


# Dstreams

## Discretized Streams

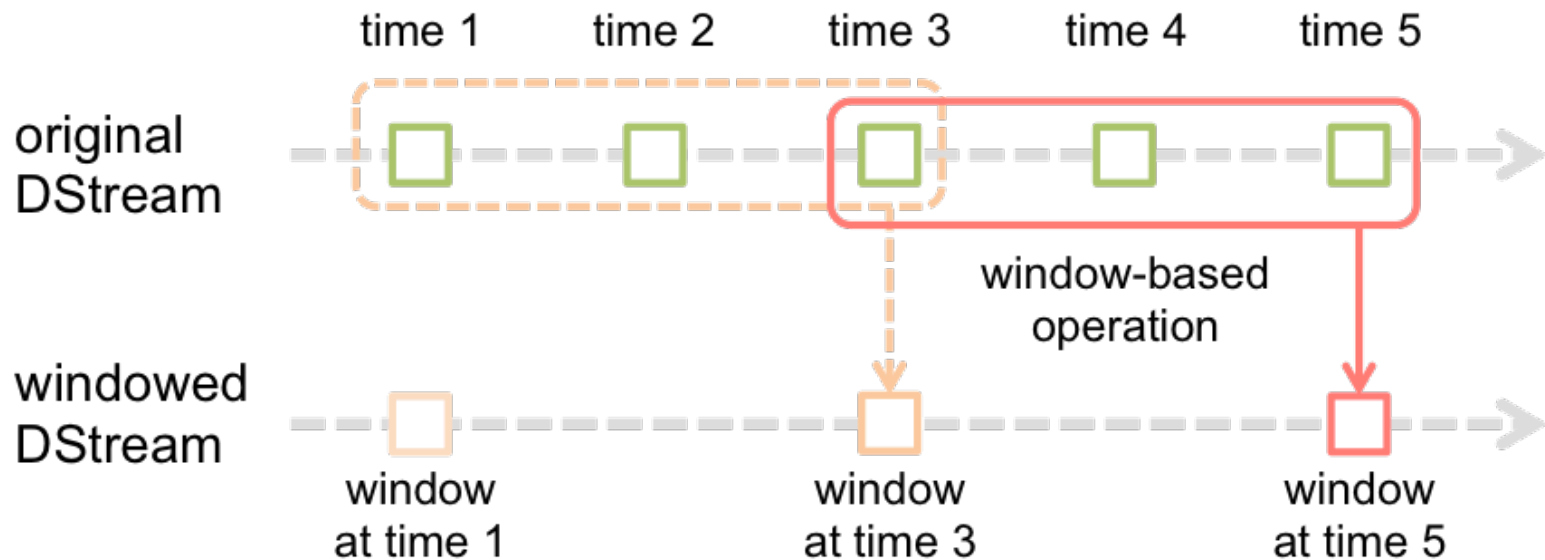


# Operations on DStreams



# CEP capabilities in Spark Streaming

- DStream “updateStateByKey”
- Joins
- Windows



# Storm vs Spark Streaming

- “Classic” Storm has no counterpart in Spark
  - Spouts and Bolts
  - Event by event processing
- Trident and Streaming both offer micro-batch models
  - More performant but less flexible
- Storm is more flexible for pure streaming systems
- Spark offers a much more unified programming model for Batch and Streaming



# Summary

- Realtime processing is hard
  - Requires large memory and state
  - The lambda architecture splits the problem into batch and realtime challenges
- Multiple approaches:
  - Pure Streaming
  - Micro-batch
  - CEP



# Questions?



© Paul Fremantle 2015. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>