

Big Data Engineering

Realtime Big Data Processing

Julie Weeds

March 2020



© 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/>

Streaming

- Continuous data flow
 - “Unbounded streams of data”
- Usually uses a message distribution system
 - JMS, Apache Kafka, ZeroMQ, MQTT
- 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

A diagram showing the interaction between a user and a database. On the left, a user icon is shown at a computer monitor. A blue arrow labeled "request" points from the user to a central server icon. Another blue arrow labeled "response" points from the server back to the user.

A diagram showing the interaction between an event and a user. On the left, a cloud-like shape labeled "Event" has a blue arrow labeled "input stream" pointing to a central server icon. From the server, a blue arrow labeled "output stream" points to a user icon at a computer monitor.

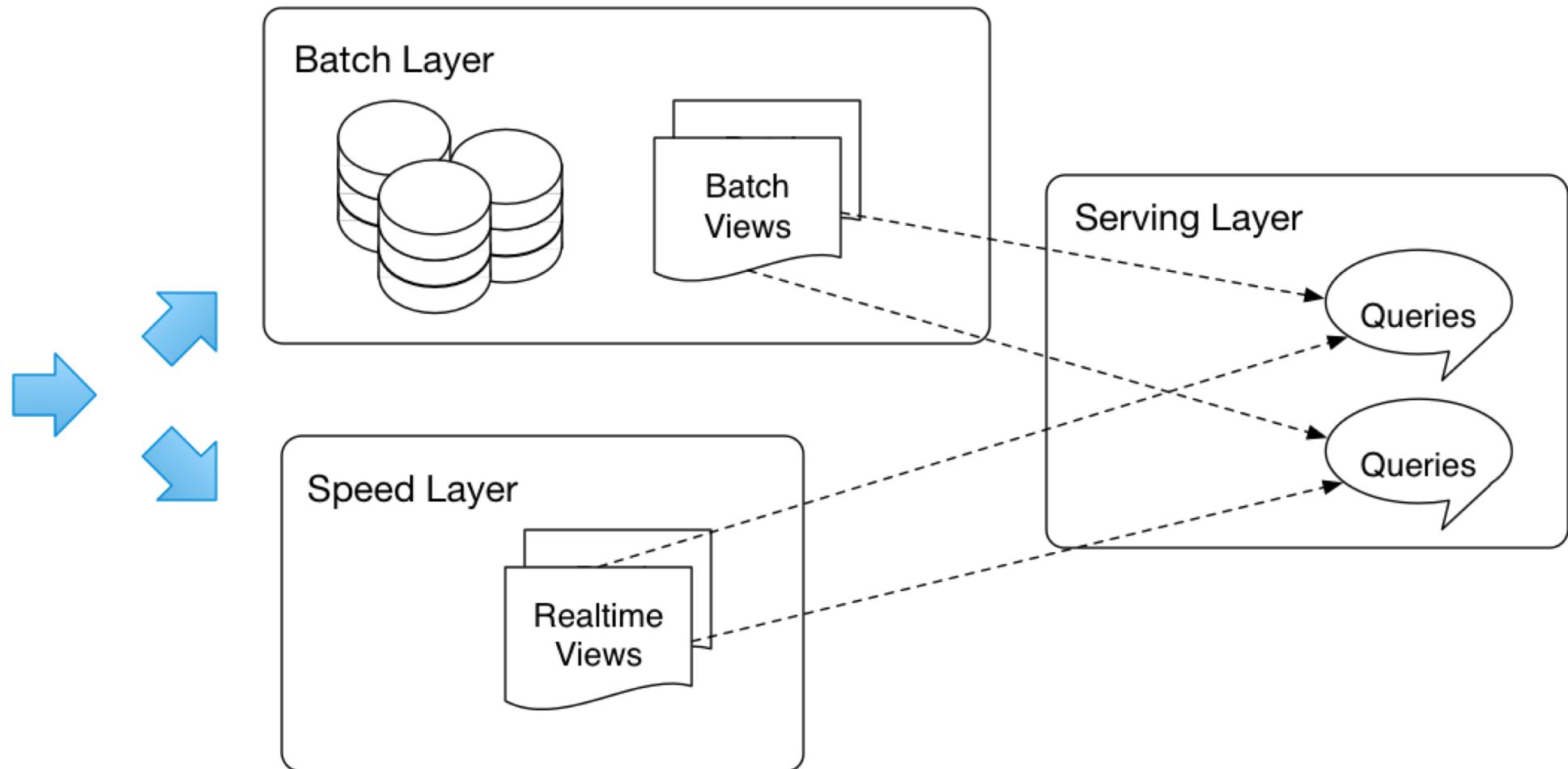


Approaches to Streaming

- 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



Lambda Architecture



Data distribution

- You need to get the events to the processing systems



© 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/>

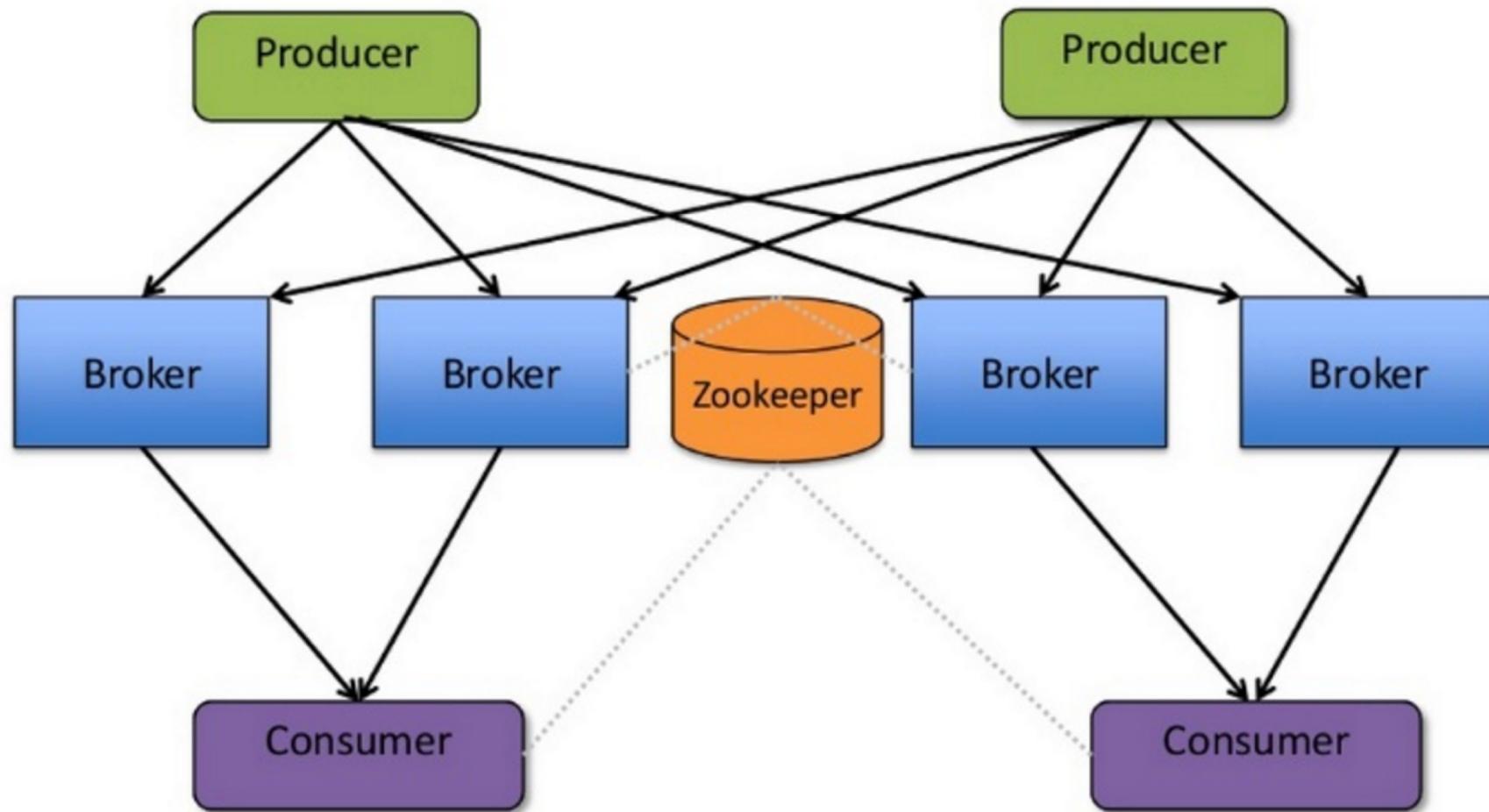
MQTT (Message Queuing Telemetry Transport)

- 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



© 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 Kafka



© 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/>

Source: <http://www.slideshare.net/charmalloc/>

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



Mathias Verraes

@mathiasverraes

Follow

There are only two hard problems in distributed systems:
2. Exactly-once delivery 1.
Guaranteed order of messages 2. Exactly-once delivery

RETWEETS

6,775

LIKES

4,727



10:40 AM - 14 Aug 2015



69



6.8K



4.7K



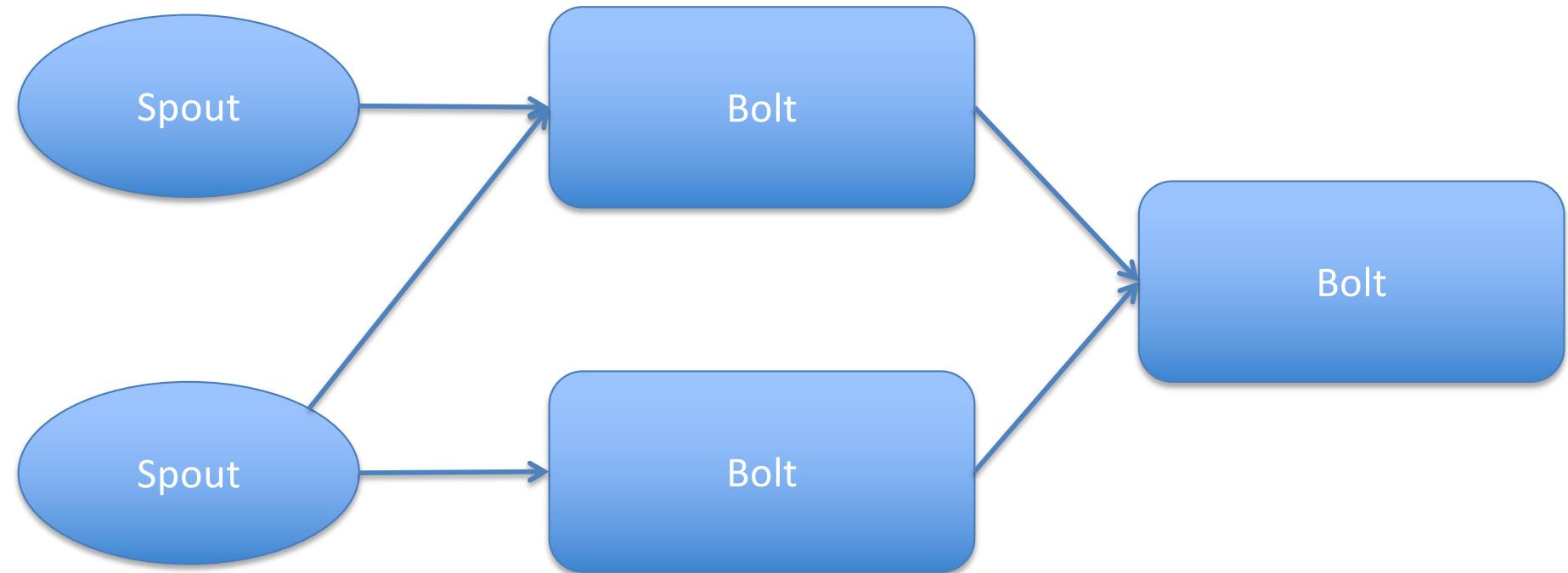
© Paul Fremantle 2015. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

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



© 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/>

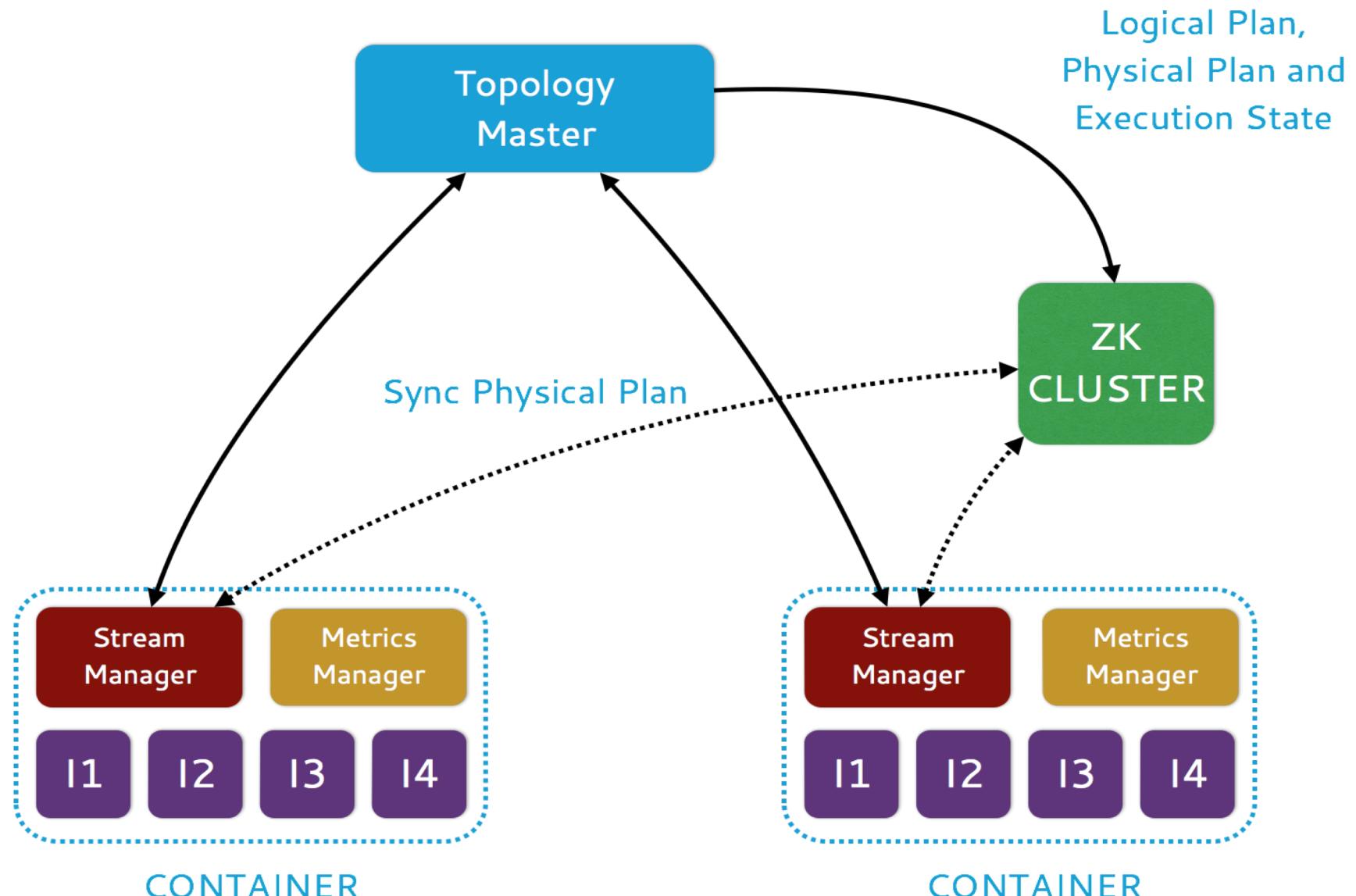
Apache Storm

- 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>



© 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/>

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

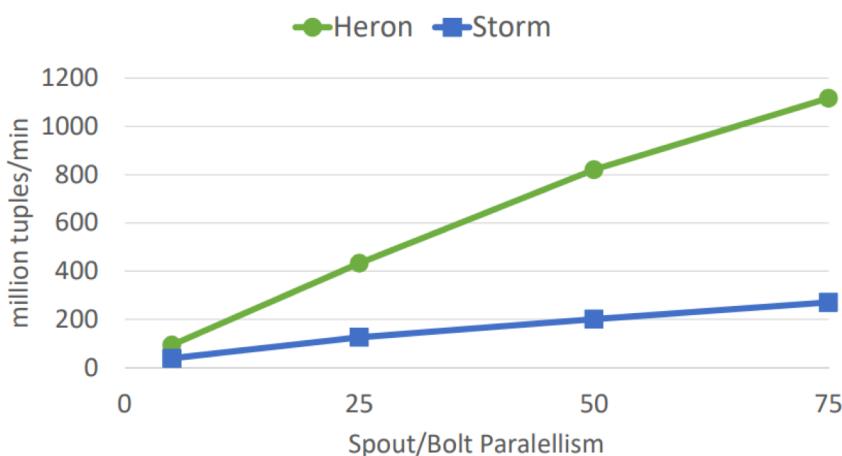


Fig. 2. Throughput with acks

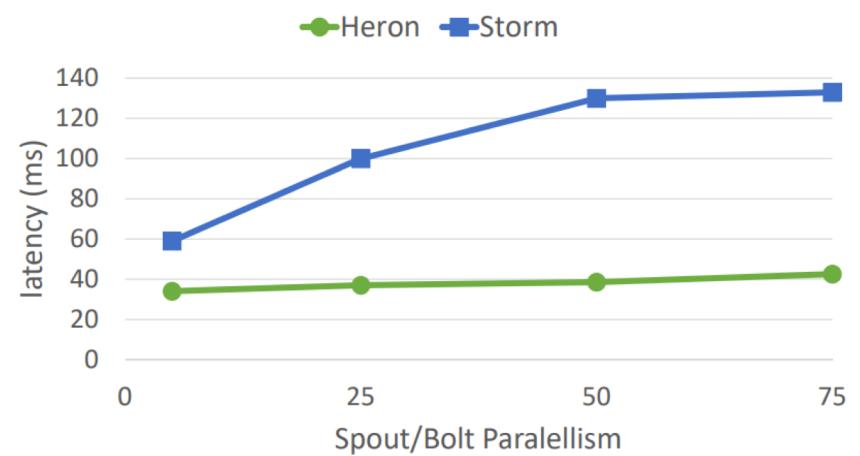
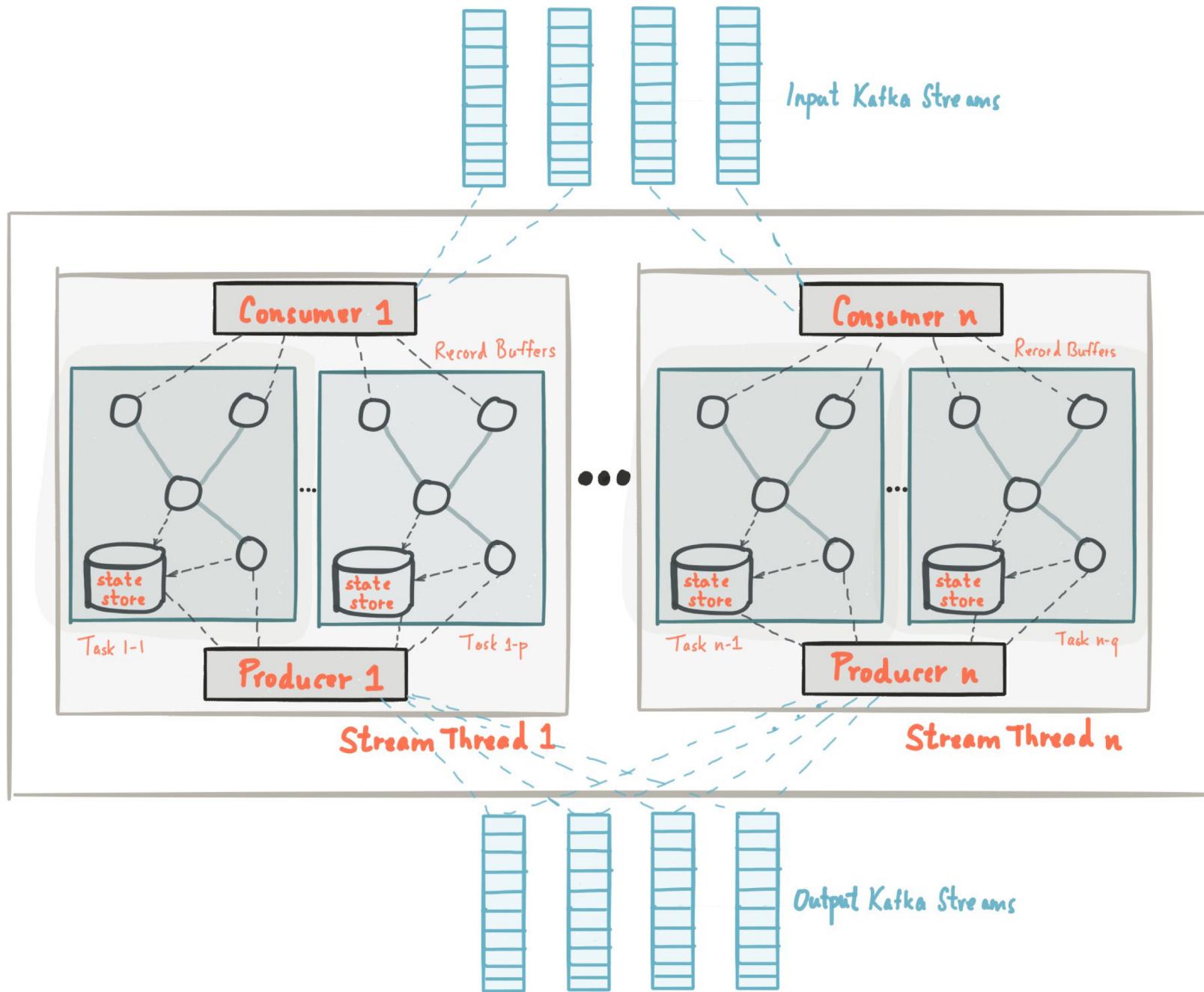


Fig. 3. End-to-end latency with acks



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

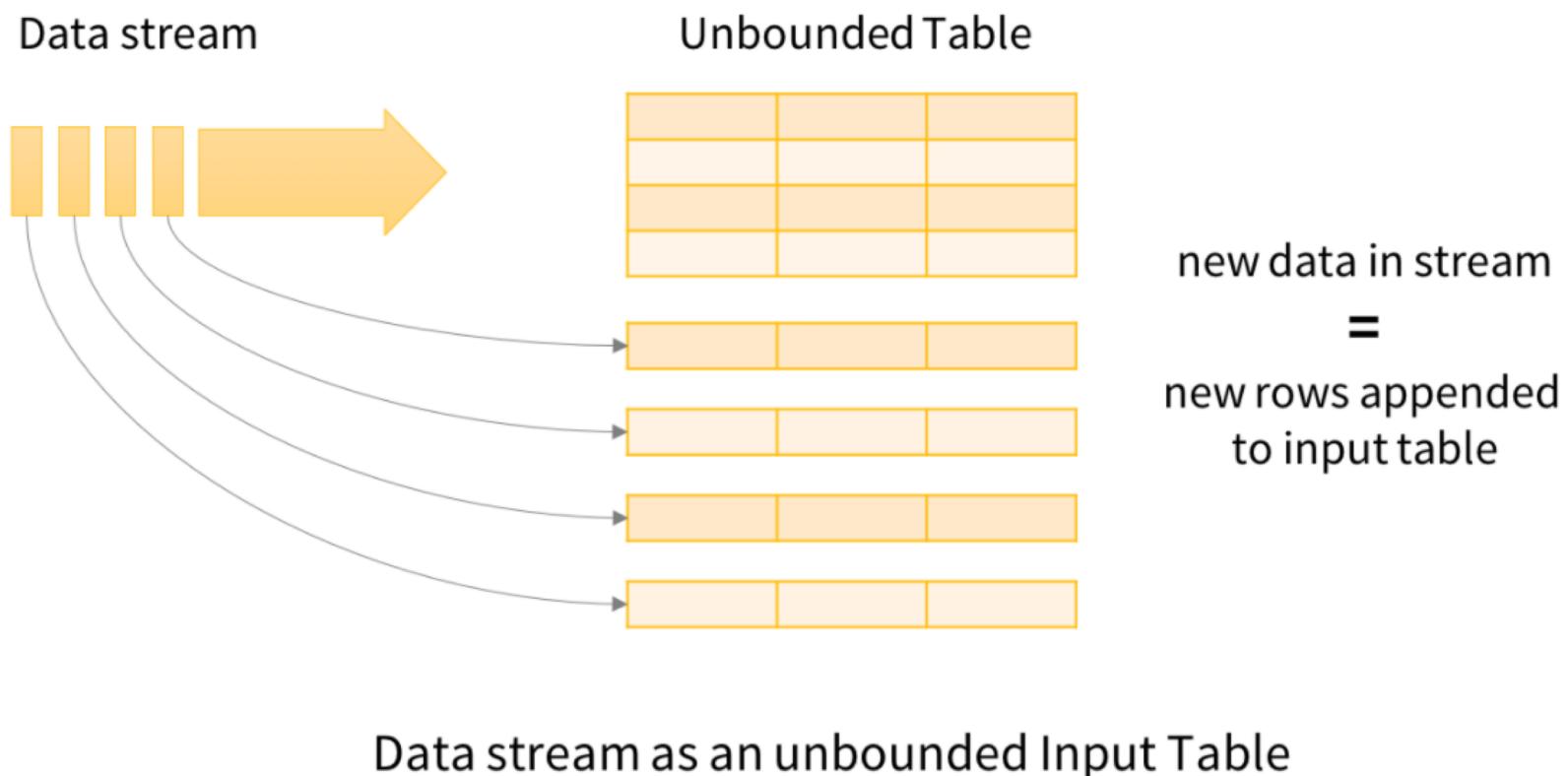


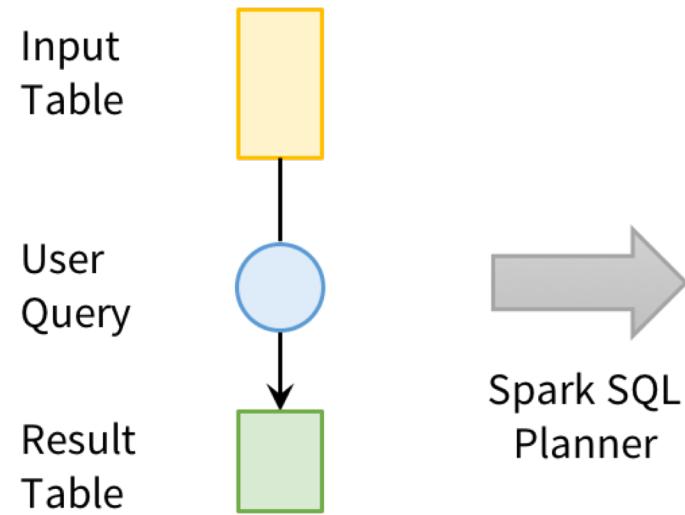
Apache Spark Streaming



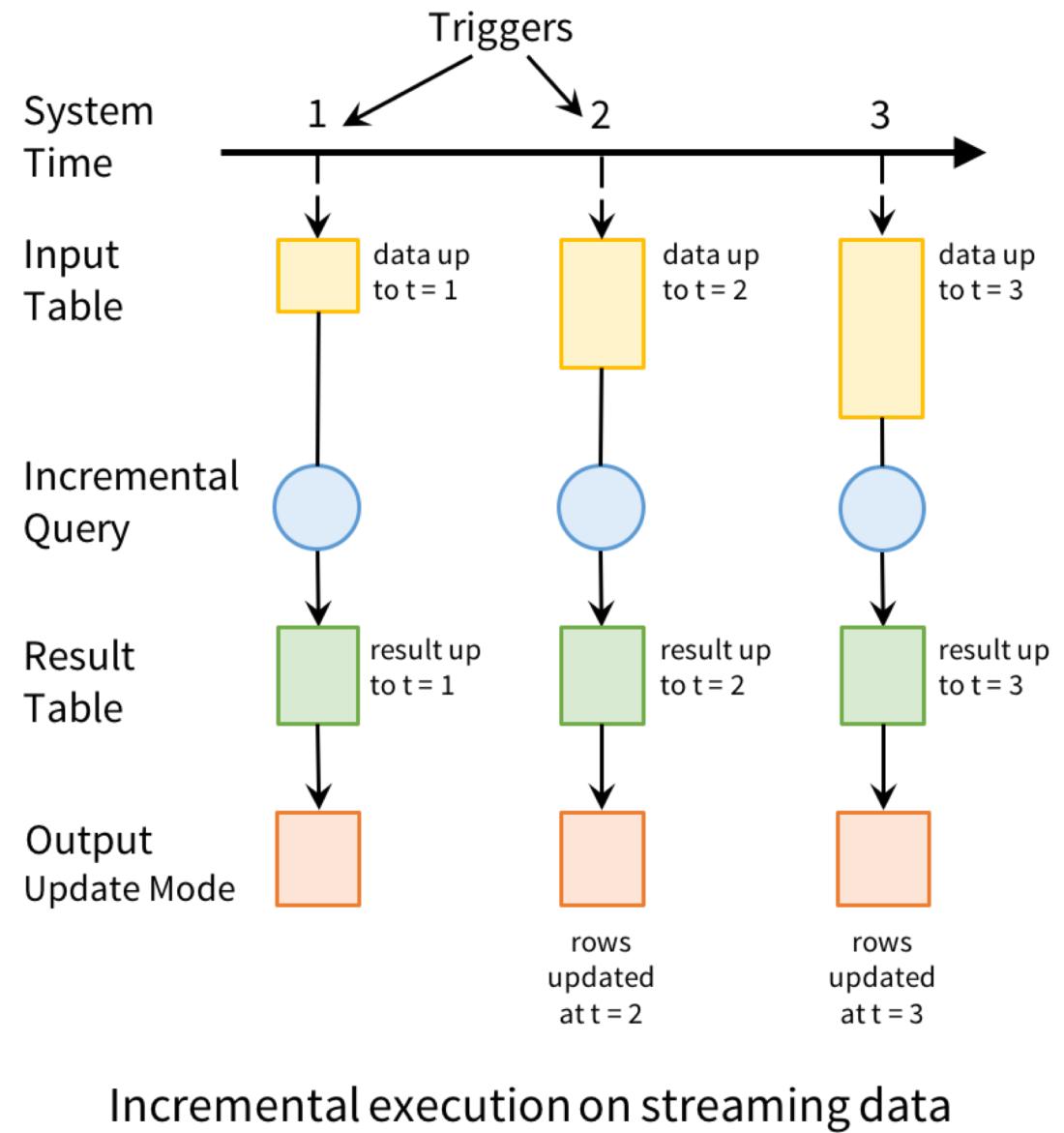
Structured Streams in Spark

- Since Spark 2.0, there is a much better approach





User's batch-like query on input table



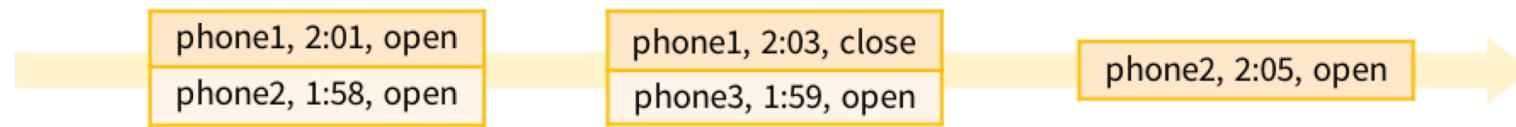
Incremental execution on streaming data

Structured Streaming Processing Model

Users express queries using a batch API; Spark incrementalizes them to run on streams



Arriving Records



System Time

2:02

2:04

2:06

Input Table

phone1, 2:01, open
phone2 1:58, open

data up to 2:02

phone1, 2:01, open
phone2, 1:58, open
phone1, 2:03, close
phone3, 1:59, open

data up to 2:04

phone1, 2:01, open
phone2, 1:58, open
phone1, 2:03, close
phone3, 1:59, open
phone2, 2:05, close

data up to 2:06

Query



Result Table

open	2:00	1
open	1:00	1

result up to 2:02

open	2:00	1
open	1:00	2
close	2:00	1

result up to 2:04

open	2:00	2
open	1:00	2
close	2:00	1

result up to 2:06

Output Update Mode

open	2:00	1
open	1:00	1

open	1:00	2
close	2:00	1

open	2:00	2
------	------	---



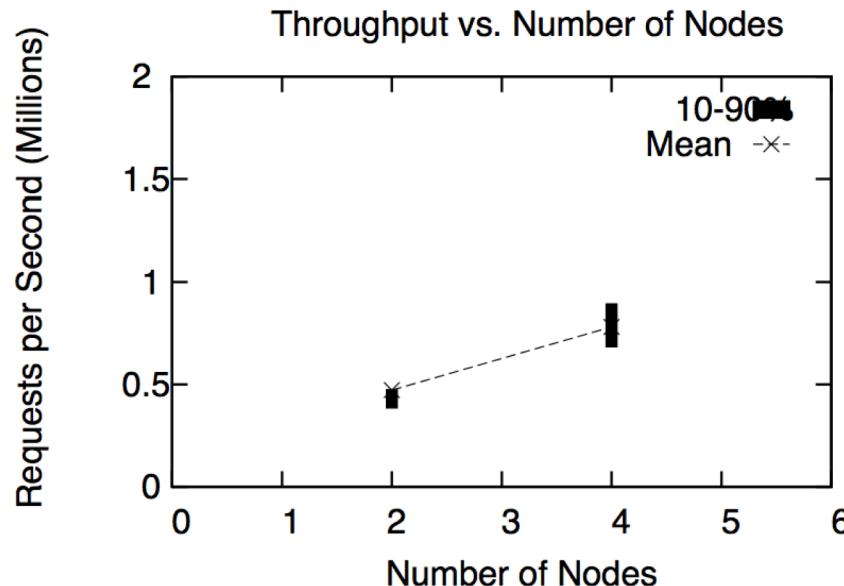
Siddhi

- 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, Spark and Kafka Streams
- Supports millions of events/sec
- [http://freo.me/DEBS Siddhi](http://freo.me/DEBS_Siddhi)

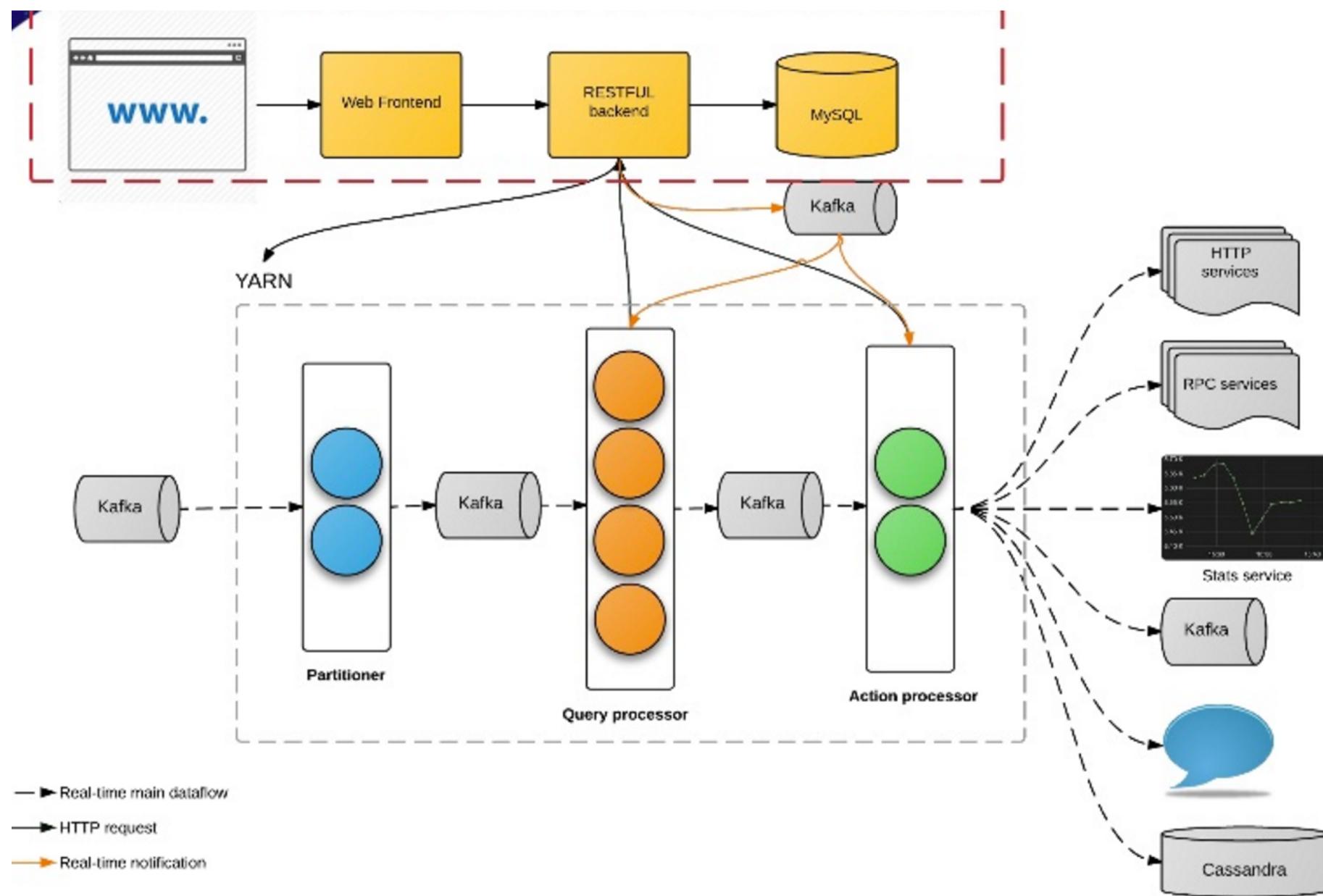


SiddhiQL

```
FROM login_stream#window.time(10 min)
SELECT ip,
       count(ip) as loginCount,
       cityId
GROUP BY ip
HAVING loginCount > 10
INSERT INTO login_attemp_repeatedly_stream;
```



Siddhi at Uber



Siddhi at Uber

- 100+ production apps
- 30 billion messages / day
- Fraud, anomaly detection
- Marketing, promotion
- Monitoring, feedback
- Real time analytics and visualization

<https://freo.me/siddhi-uber>



© 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/>

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/>