

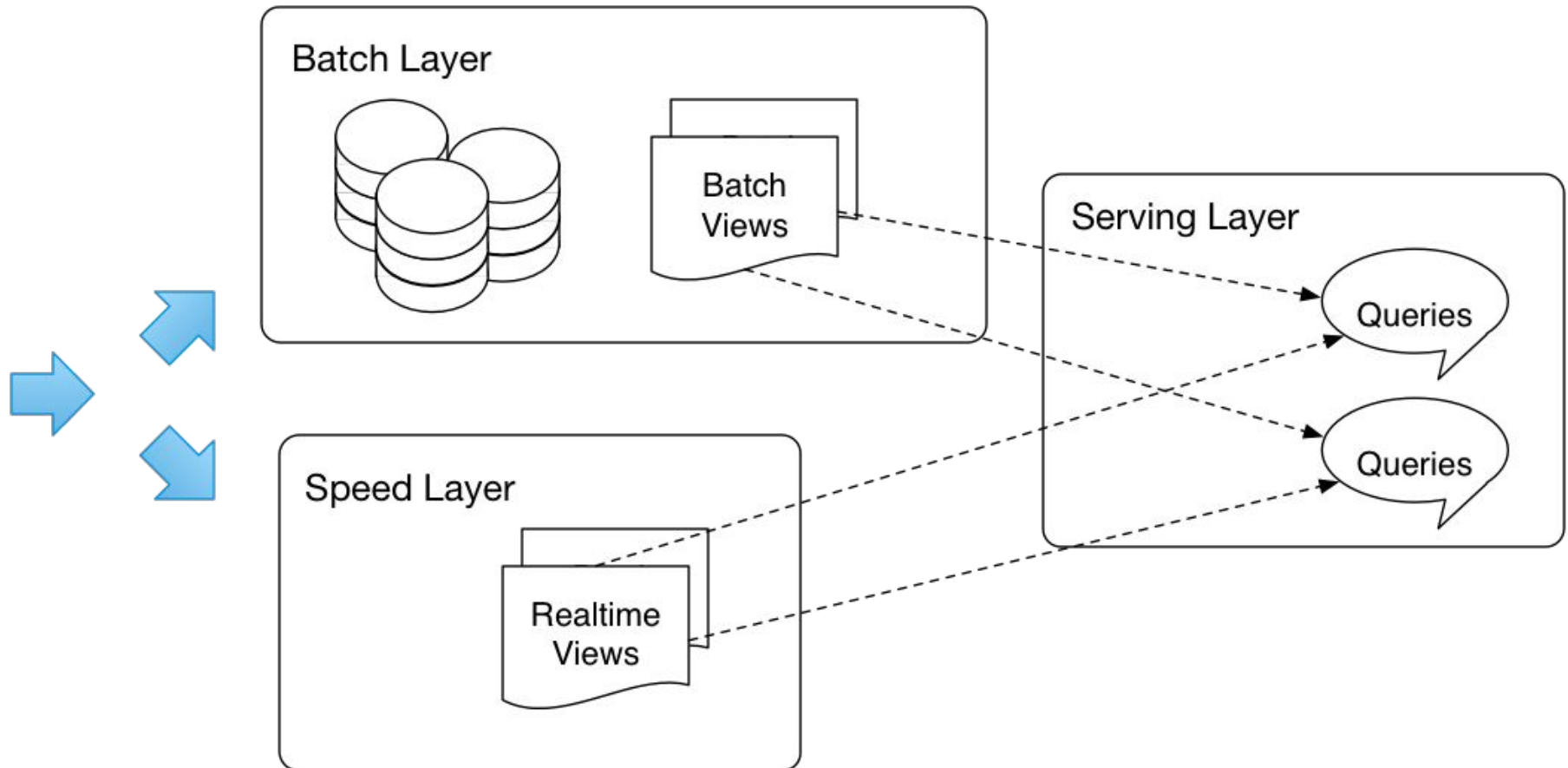
Cloud Computing and Big Data

Realtime Big Data

Oxford University
Software Engineering
Programme
July 2021



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

Size of the Data Handled
(per second)

100k events
(100MBs)

1k events
(1MBs)

100 events
(10 KBs)

In-Memory
Computing
(Spark, Hana,
VoltDB)

Interactive
Processing
(e.g. Drill,
BigQuery,
OLAP)

Realtime
Analytics
(CEP, Stream
Processing)

Indexed
Storage
(RDBMS, Solar)

MapReduce
(e.g. Spark, Hadoop)

millis

seconds

minutes

hours

days

Time to Act

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

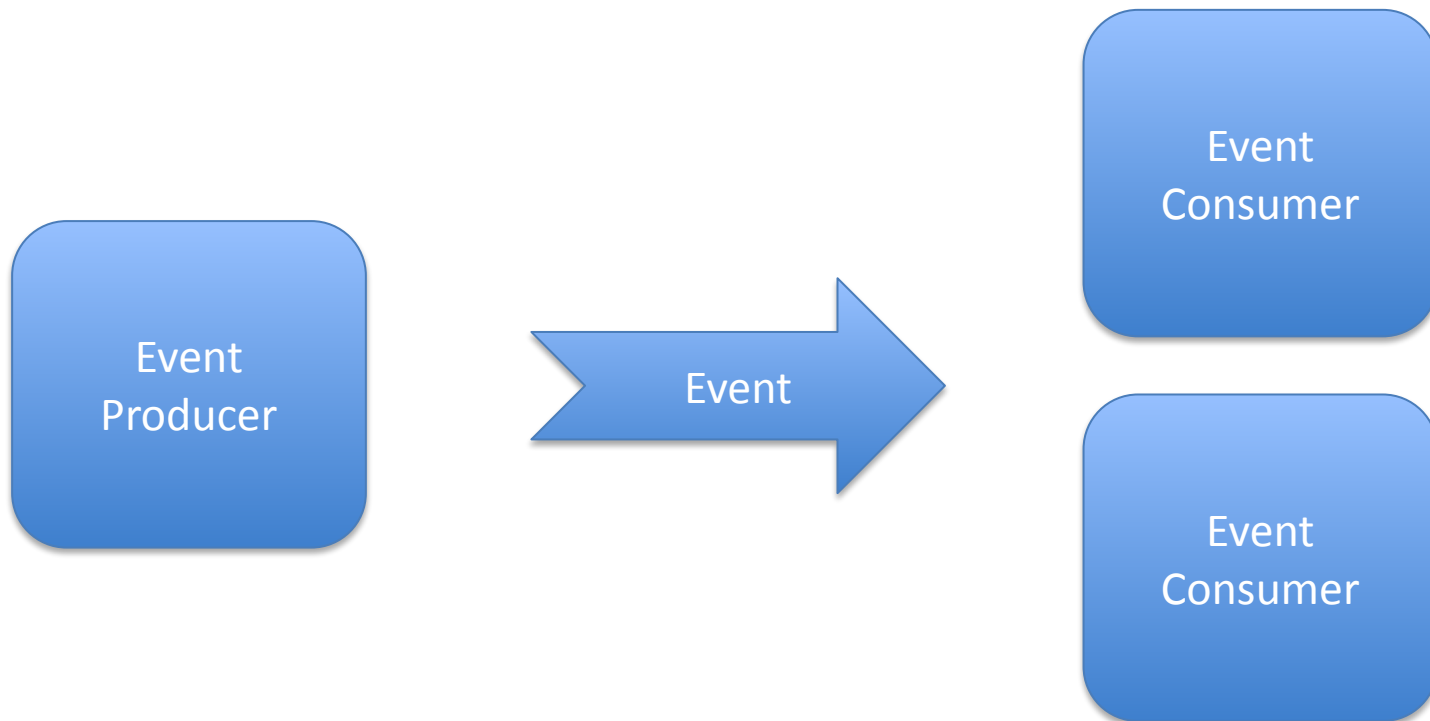


Data distribution

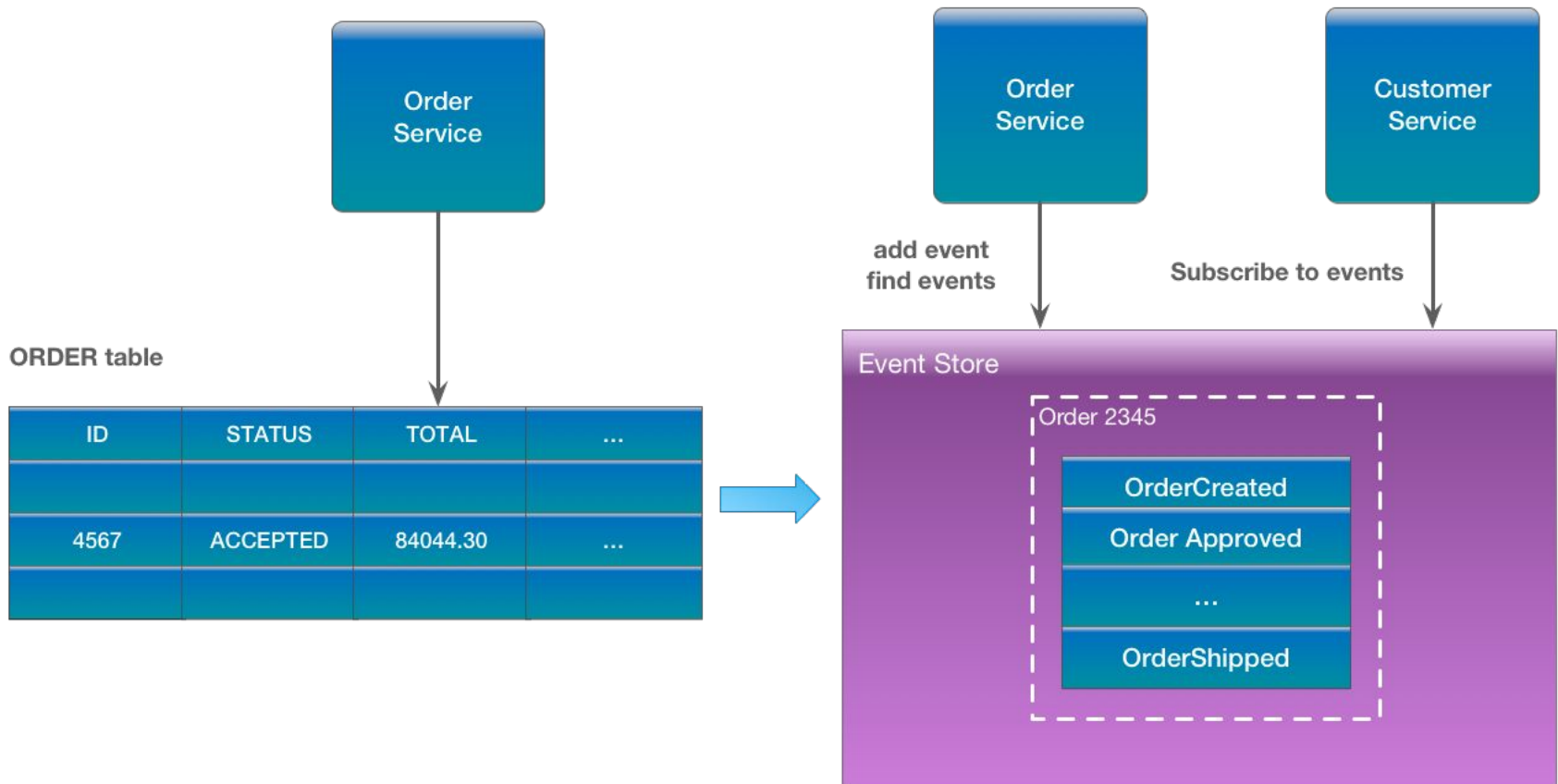
- You need to get the events to the processing systems



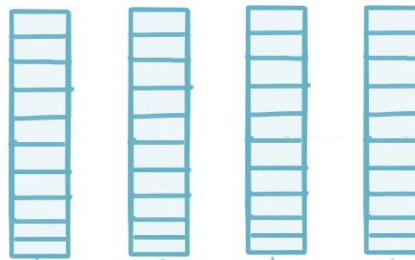
Event Driven Architecture



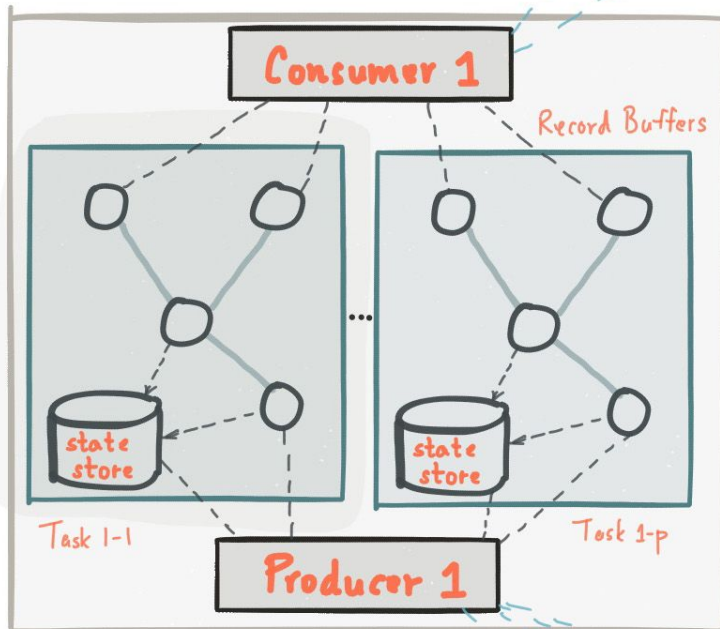
Event Sourcing



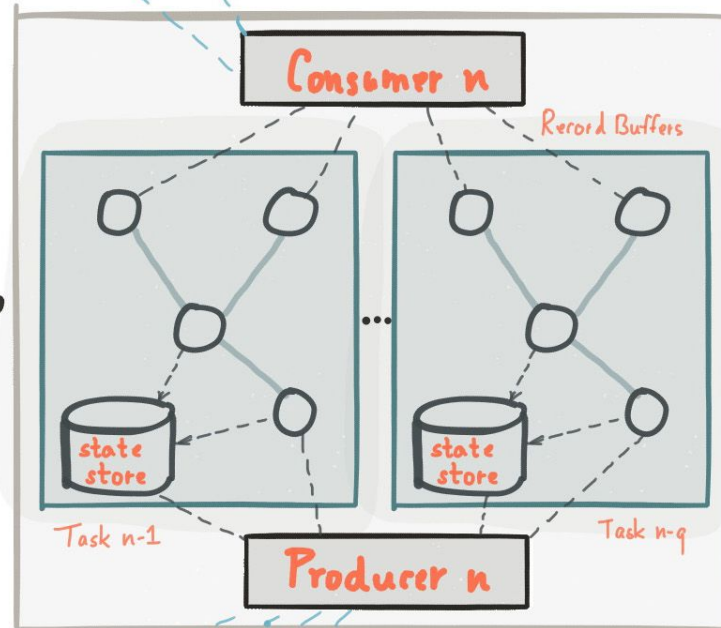
<https://eventuate.io/whyeventsourcing.html>



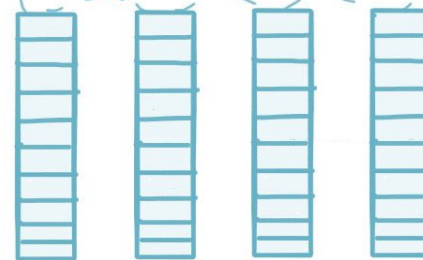
Input Kafka Streams



Stream Thread 1

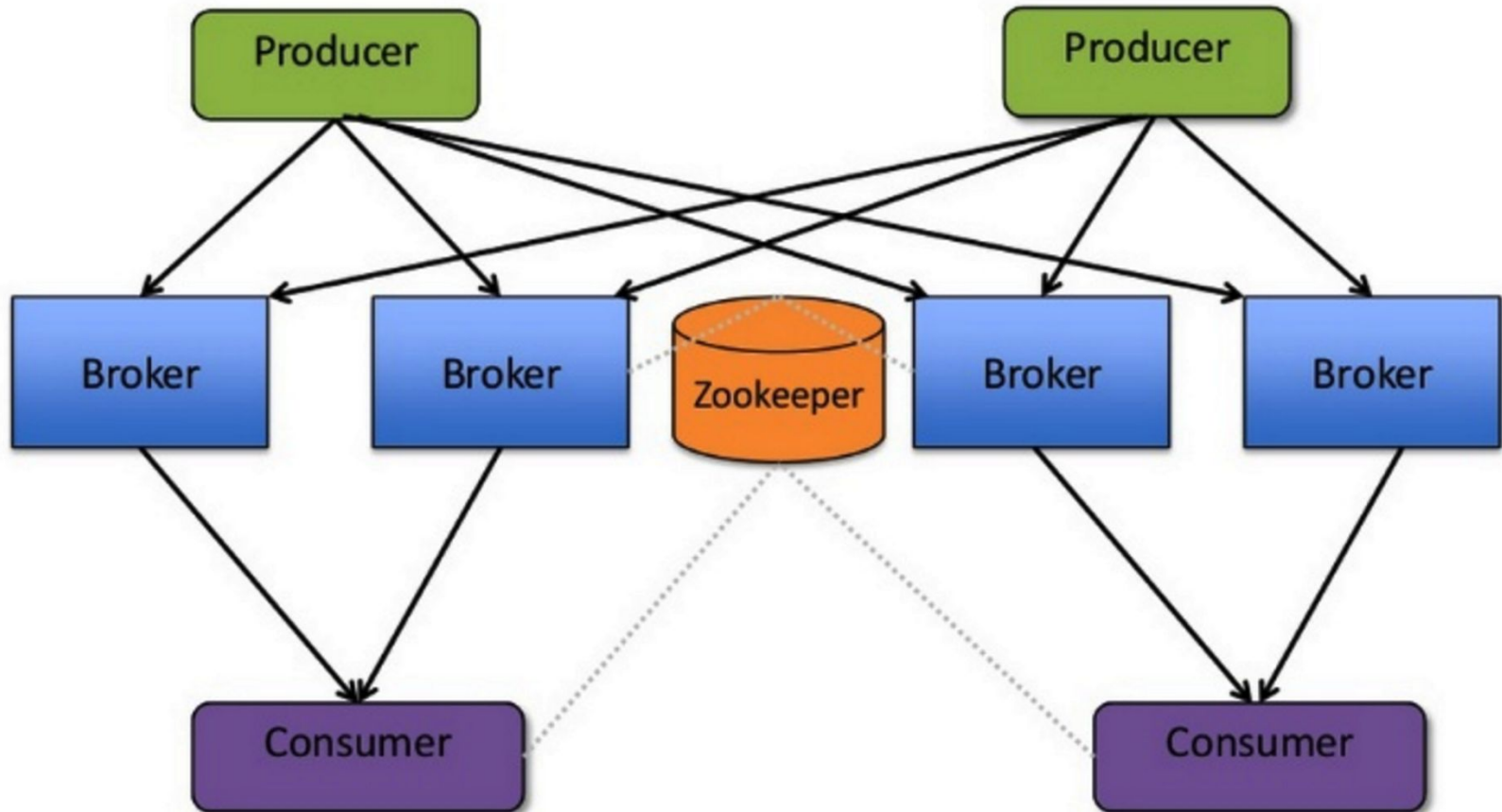


Stream Thread n



Output Kafka Streams

Apache Kafka



Kafka

- Applying “big data” approaches to messaging:
 - 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

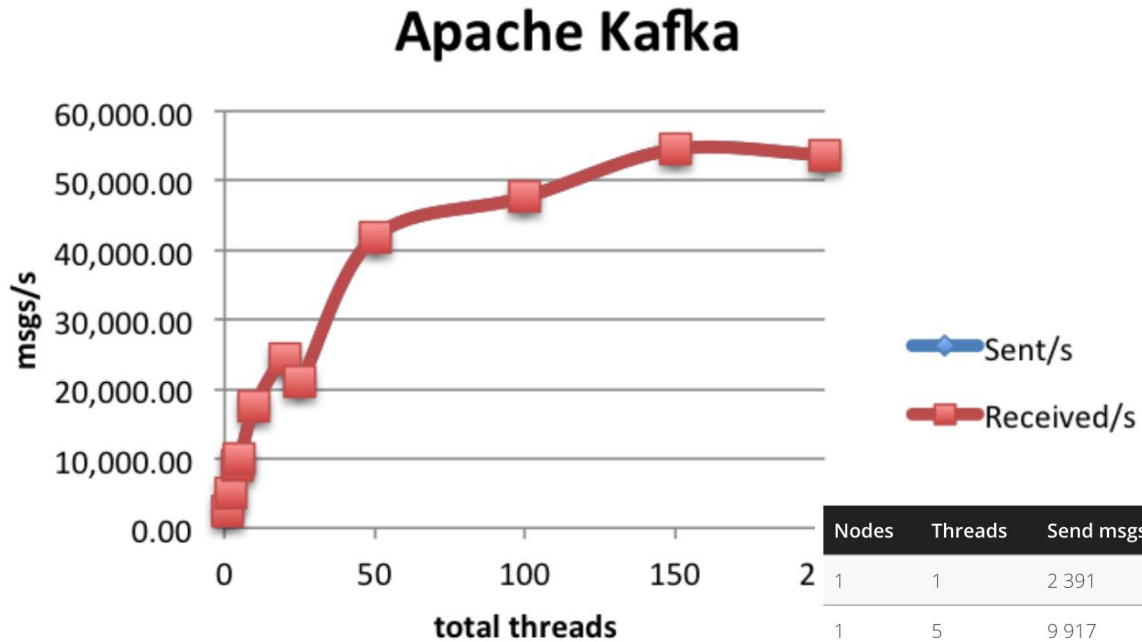


10:40 AM - 14 Aug 2015

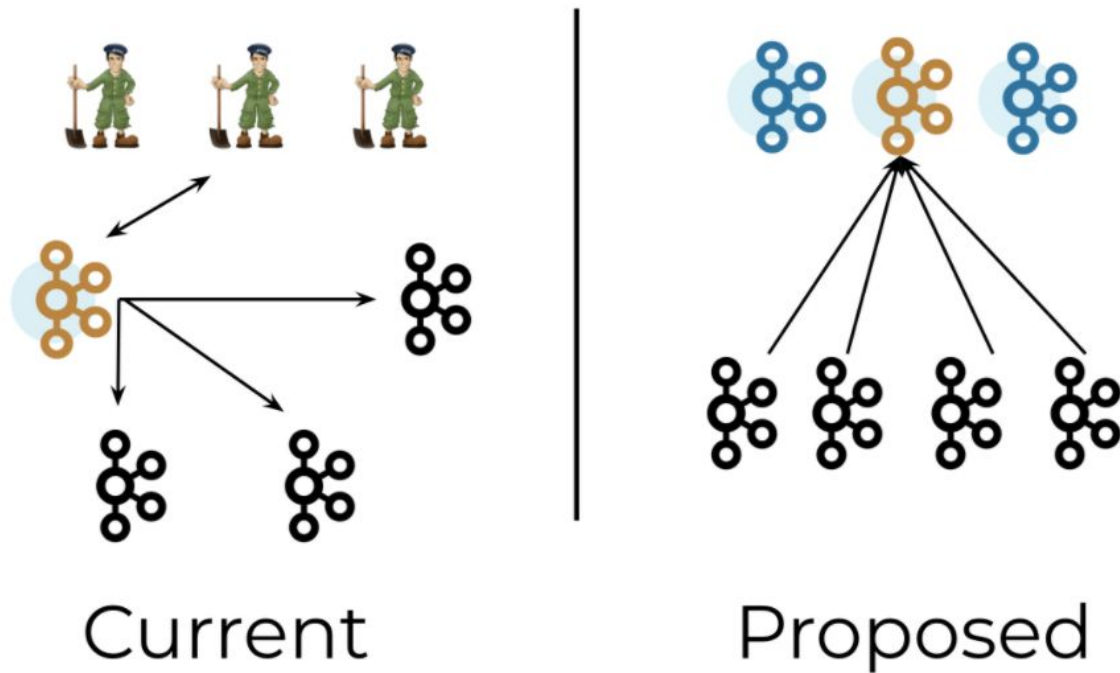


© Paul Fremantle 2015. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Kafka Performance



Nodes	Threads	Send msgs/s	Receive msgs/s	Processing latency	Send latency
1	1	2 391	2 391	48	48
1	5	9 917	9 917	48	48
1	25	20 982	20 982	46	48
2	1	4 957	4 957	47	
2	5	17 470	17 470	47	
2	25	41 902	41 901	45	48
4	1	9 149	9 149	47	
4	5	24 381	24 381	47	48
4	25	47 617	47 618	47	48
6	25	54 494	54 494	47	48
8	25	53 696	53 697	47	48



Roadmap

Removing ZooKeeper from Kafka's administrative tools

Several administrative tools shipped as part of the Kafka release still allow direct communication with ZooKeeper. Worse still, there are still one or two operations that can't be done except through this direct ZooKeeper communication.

We have been working hard to close these gaps. Soon, there will be a public Kafka API for every operation that previously required direct ZooKeeper access. We will also disable or remove the unnecessary `--zookeeper` flags in the next major release of Kafka.

NATS

3Mb Docker image

Minimum HA deployment is 2 servers (automatic hot-cold)

3. root@nats-server: ~ (ssh)

NATS server version 1.3.0 (uptime: 6m50s)

Server:

Load: CPU: 262.0% Memory: 33.1M Slow Consumers: 0

In: Msgs: 122.1M Bytes: 1.9G Msgs/Sec: 1139740.5 Bytes/Sec: 17.4M

Out: Msgs: 557.7M Bytes: 8.7G Msgs/Sec: 5680200.0 Bytes/Sec: 86.7M

NATS Pub/Sub stats: 6,384,935 msgs/sec ~ 97.43 MB/sec

Pub stats: 1,064,543 msgs/sec ~ 16.24 MB/sec

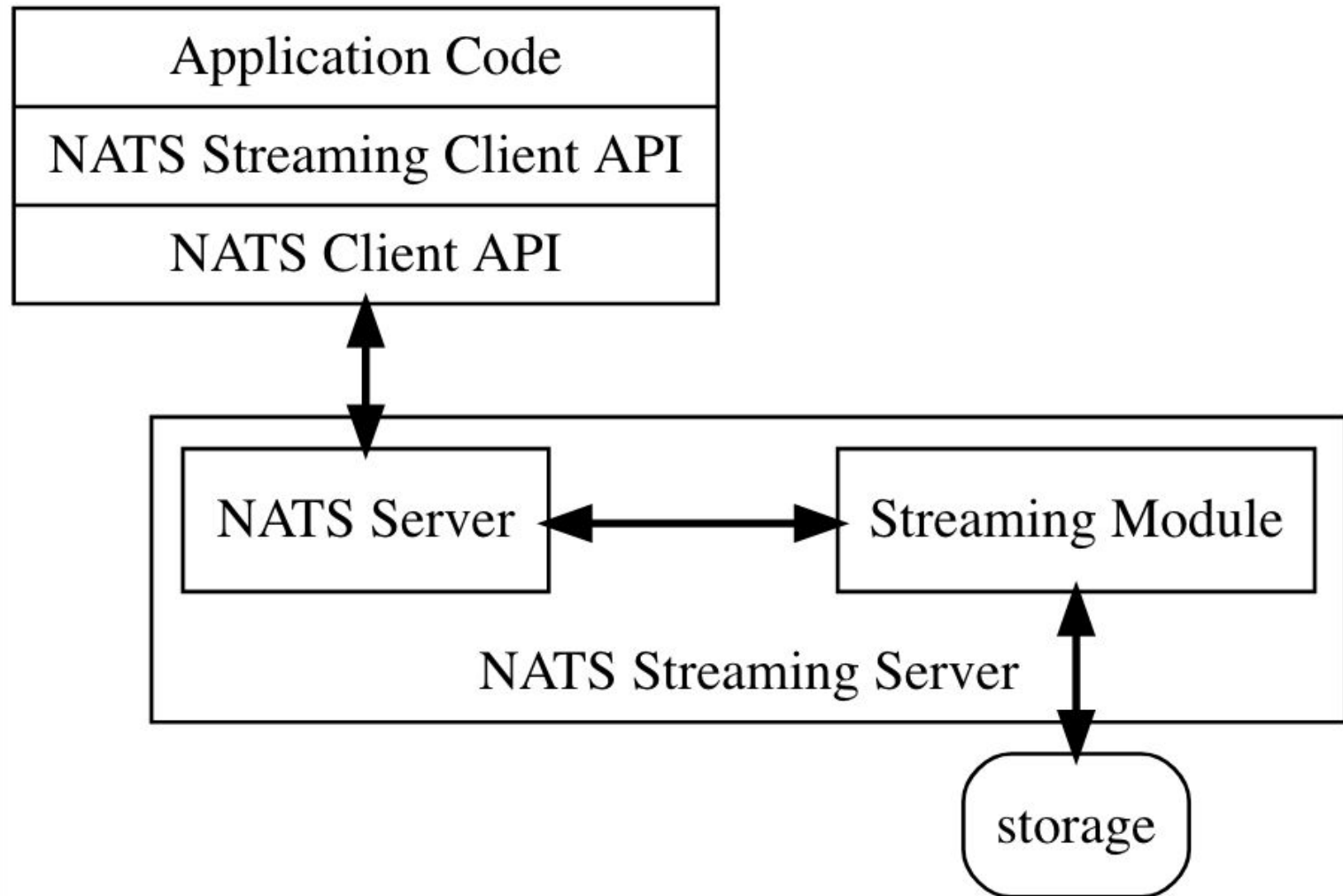
min 212,925 | avg 214,970 | max 218,169 | stddev 1,945 msgs

Sub stats: 5,320,801 msgs/sec ~ 81.19 MB/sec

min 1,064,160 | avg 1,064,283 | max 1,064,384 | stddev 72 msgs



NATS Streaming



NATS Streaming

- At least once delivery
- Publisher rate limiting
- Subscriber rate limiting
- Message Replay
- Durable Subscriptions



NATS security model

Introduction to NATS 2.0 Security



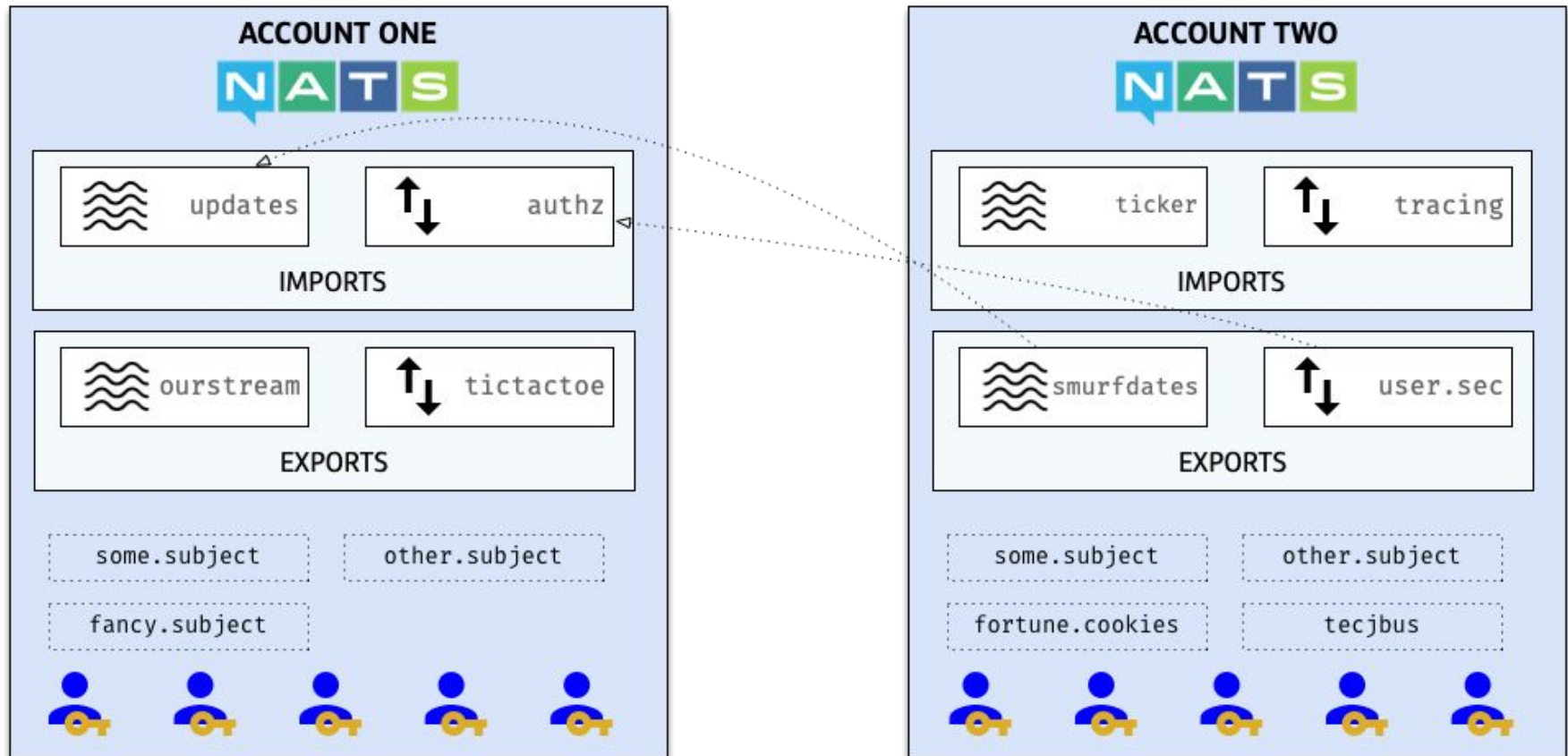
Kevin Hoffman

Apr 9 · 7 min read

Decentralized Authorization and Authentication with JWTs

NATS is a lightweight, cloud native, open-source high-performance messaging system. In this post, I want to talk about security in the upcoming NATS 2.0 release—what it is, why you should care, and what it can do for you and your organization. But before I get into those details, I want to take a moment to explain a journey common to people adopting messaging systems and asynchronous architectures.

NATS security model





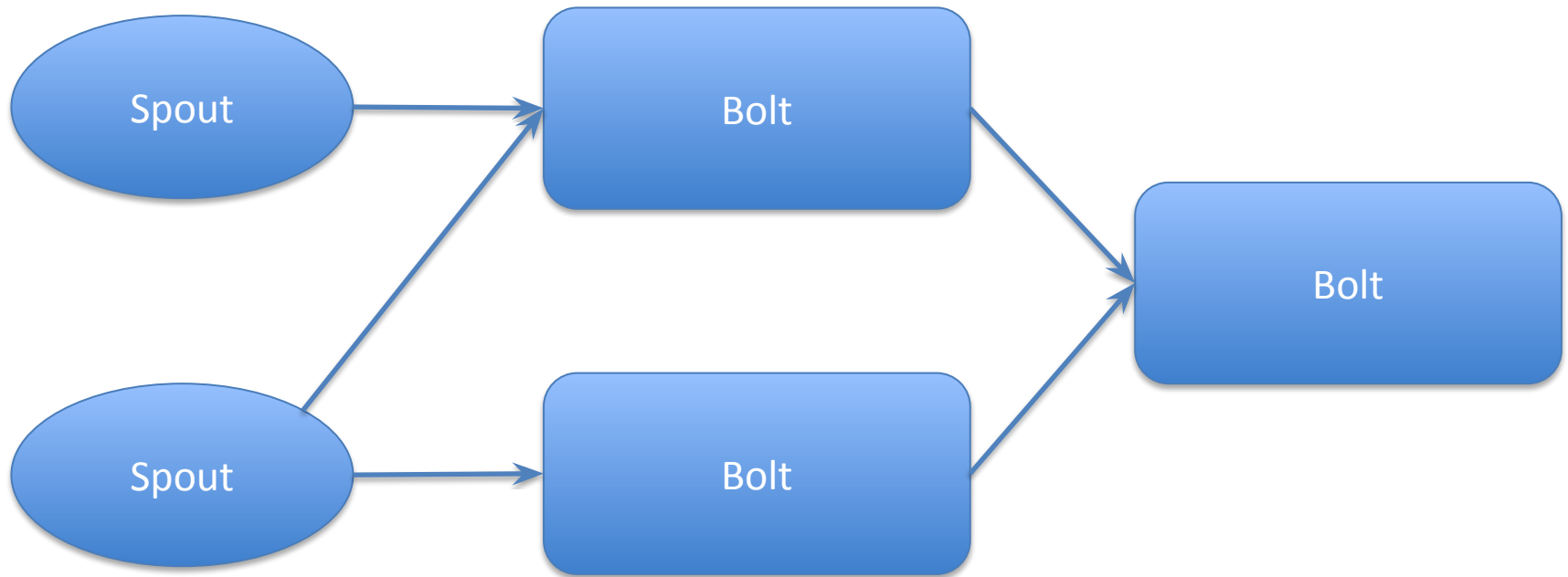
© Paul Fremantle 2015. This work is licensed under a Creative Commons
Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

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



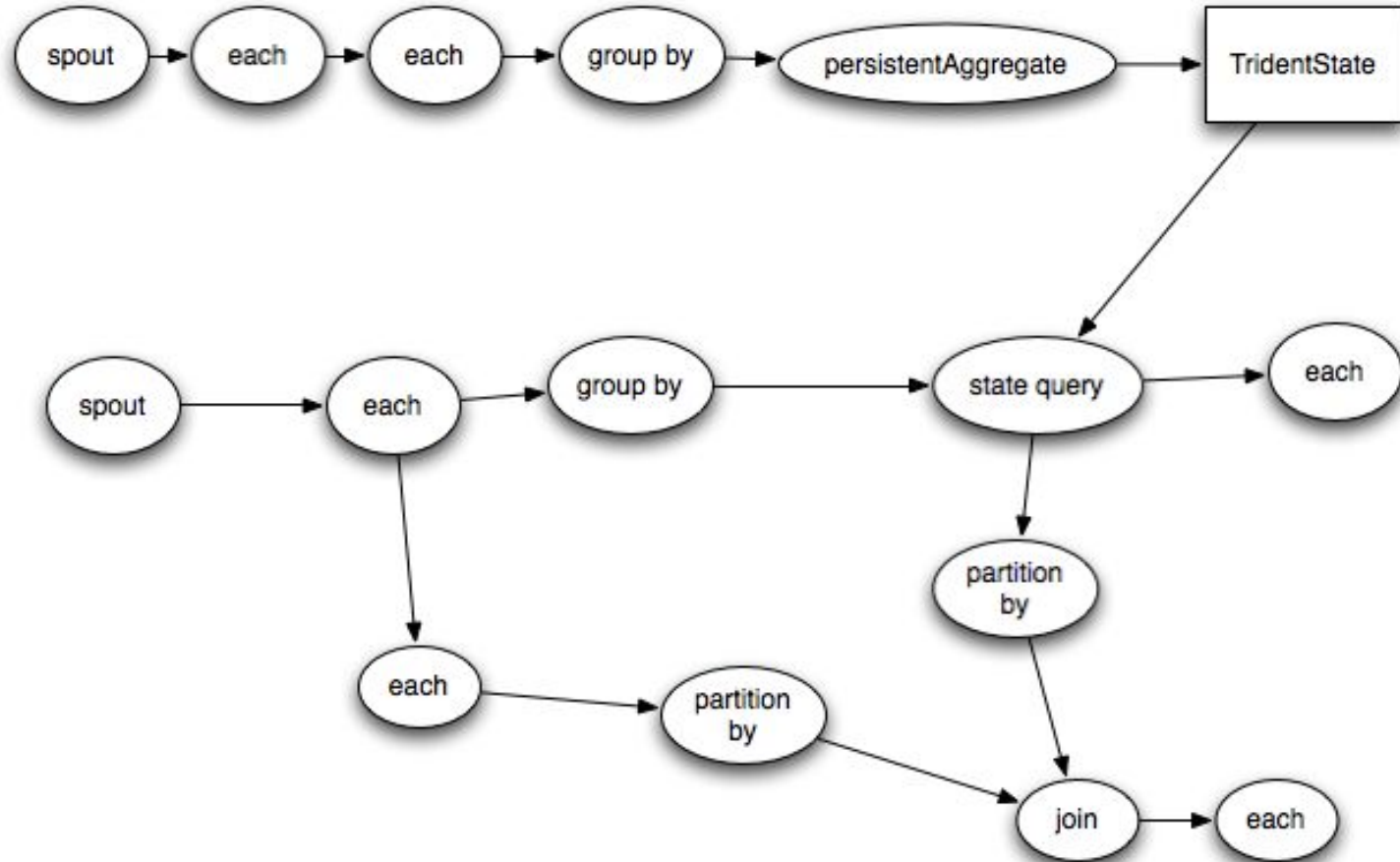
Note: another DAG

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)

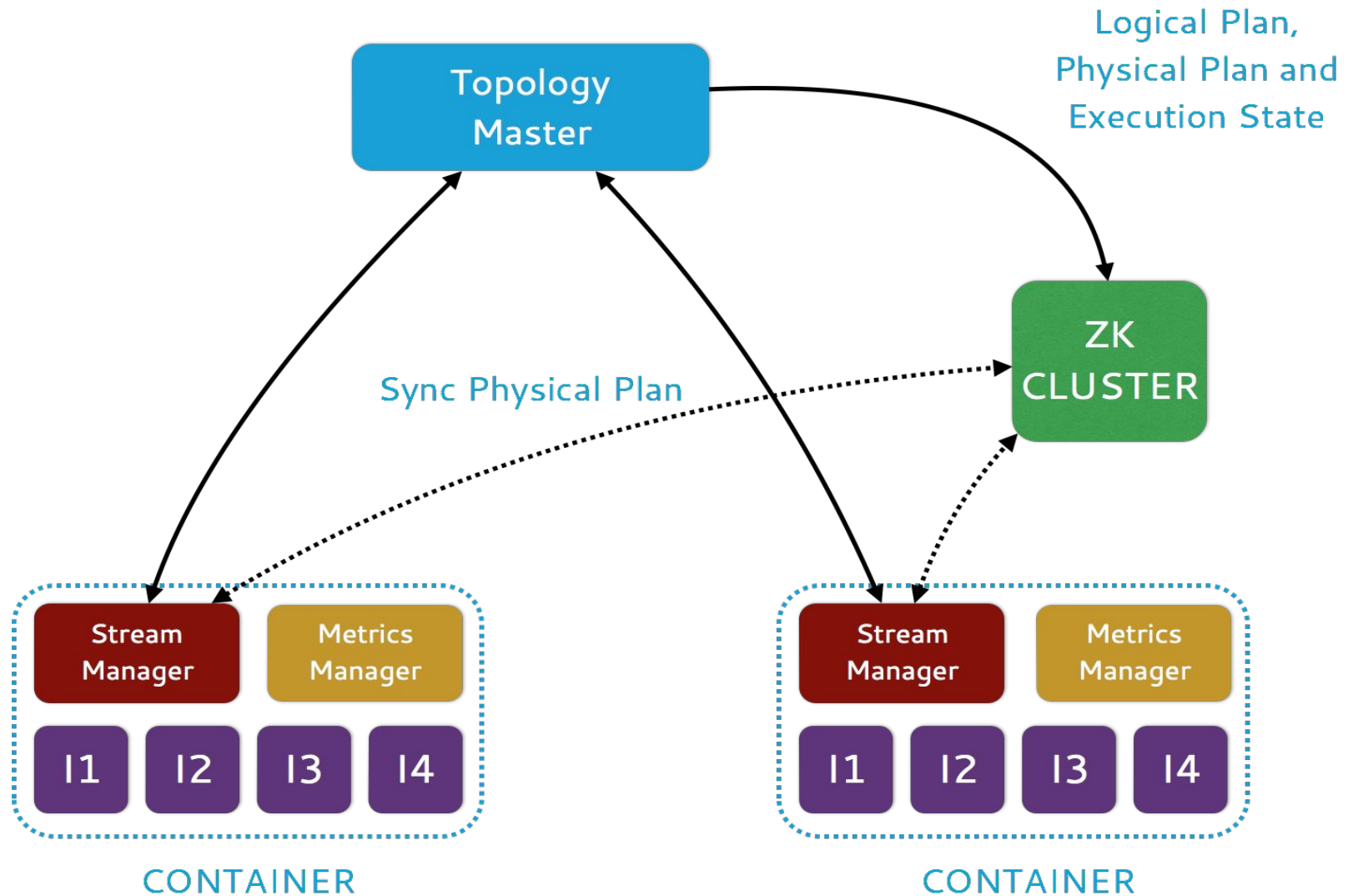


Storm vs Spark Streaming

- “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



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

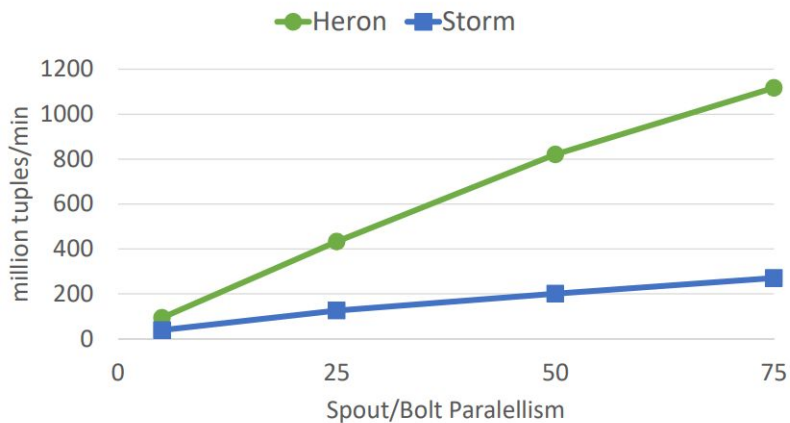


Fig. 2. Throughput with acks

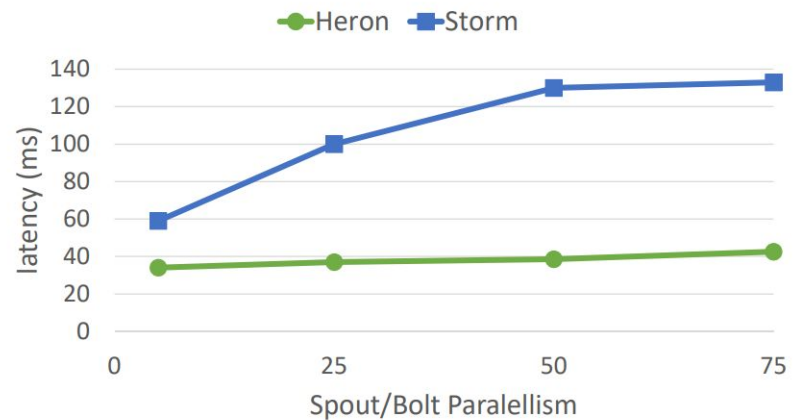
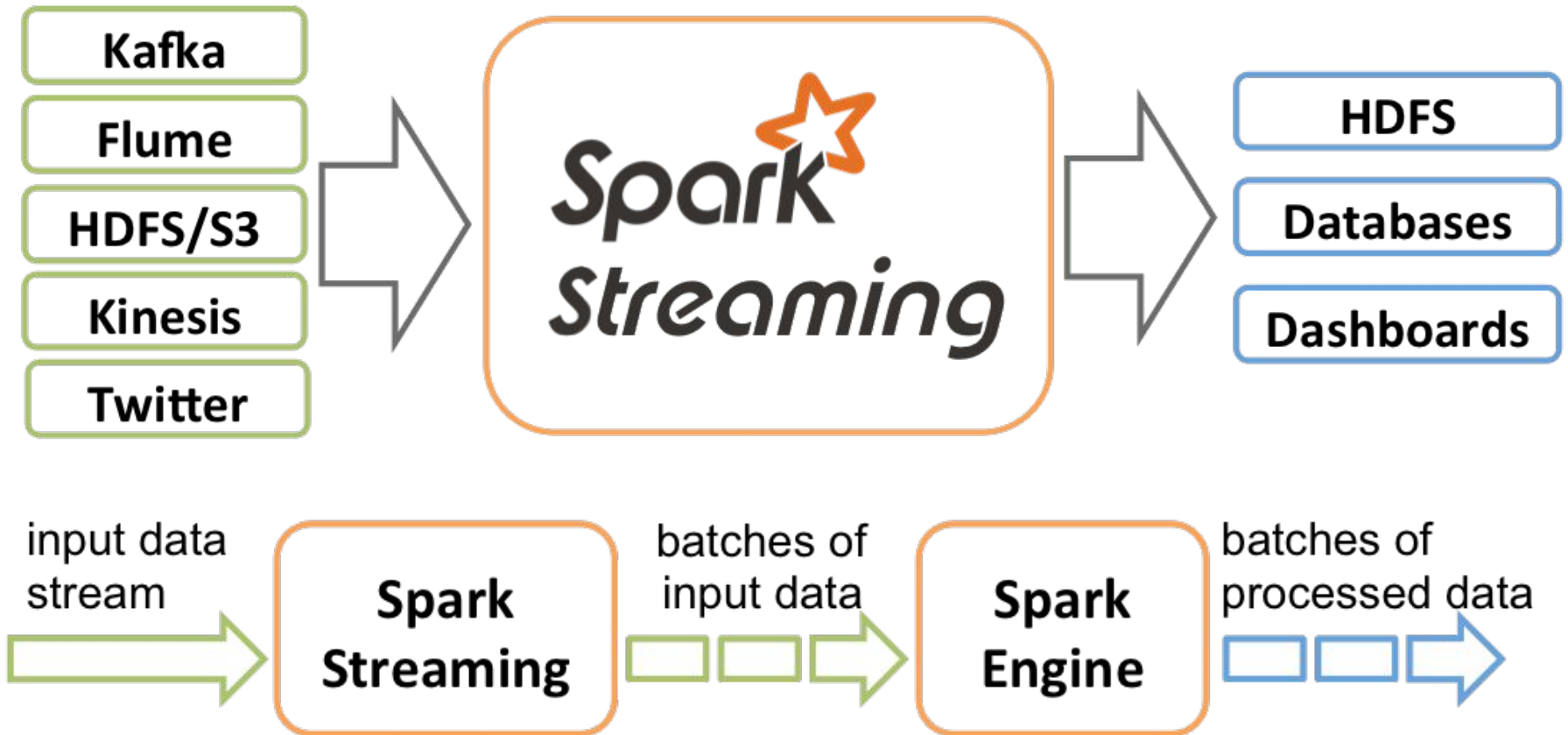


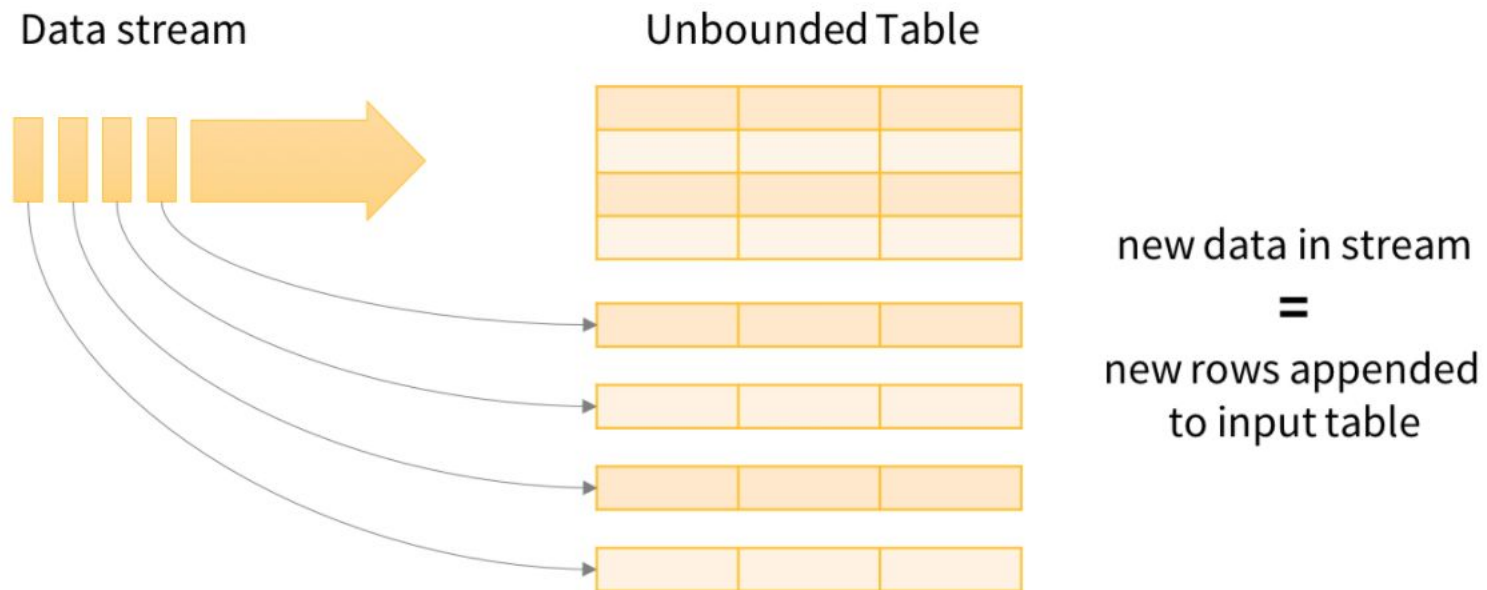
Fig. 3. End-to-end latency with acks

Apache Spark Streaming

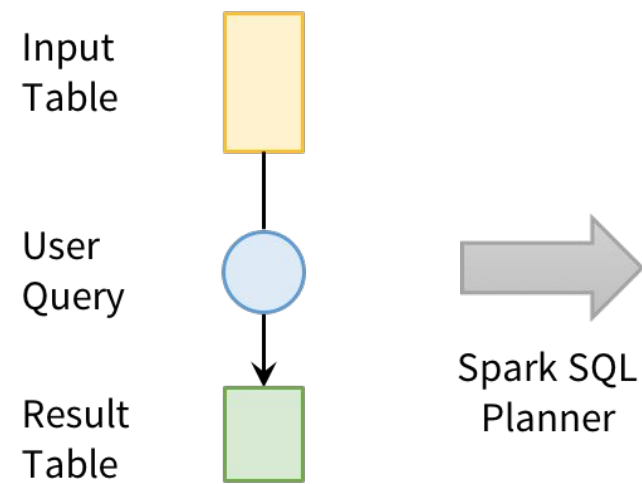


Structured Streams in Spark

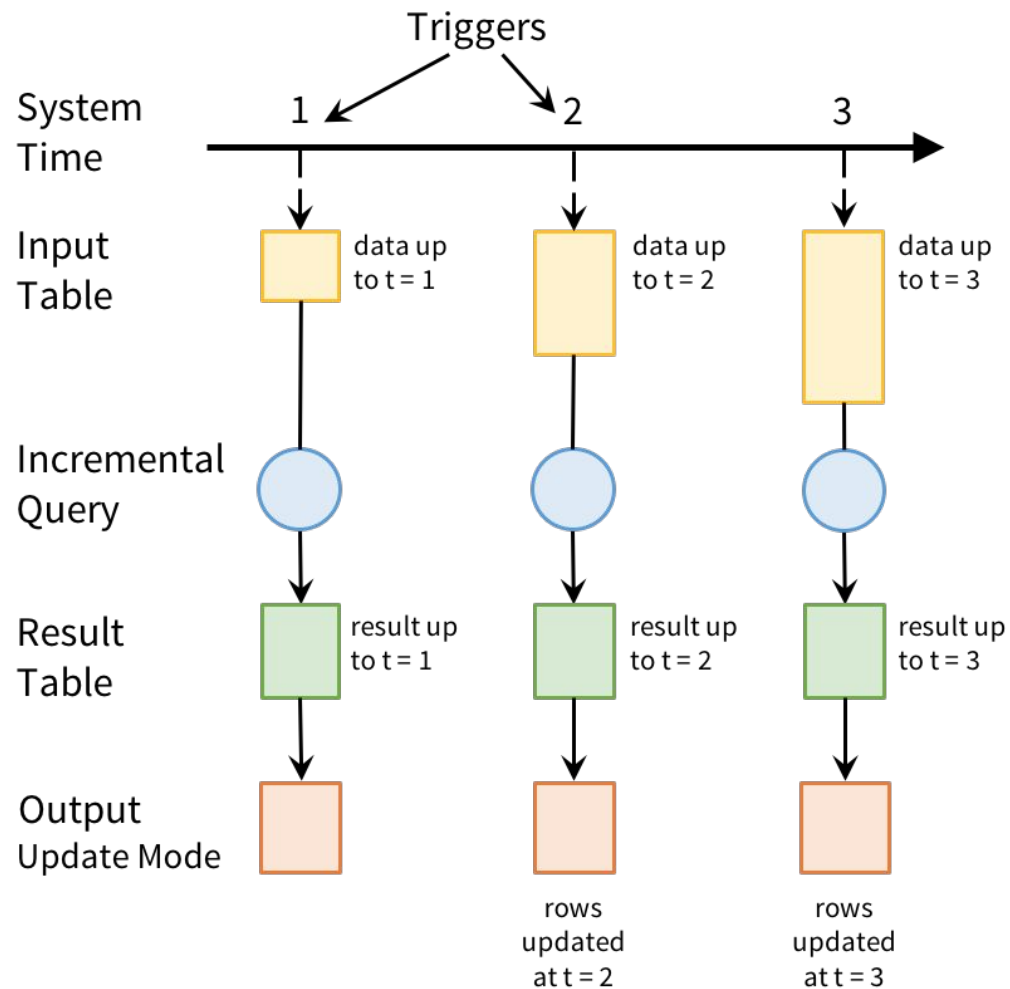
- Since Spark 2.0, there is a much better approach



Data stream as an unbounded Input Table



User's batch-like query on input table



Incremental execution on streaming data

Structured Streaming Processing Model

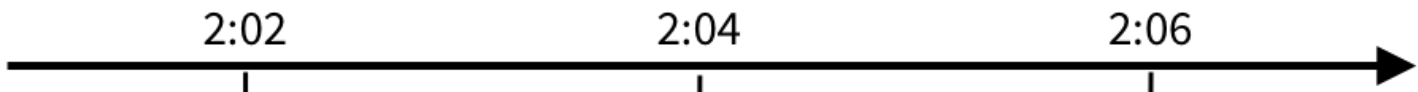
Users express queries using a batch API; Spark incrementalizes them to run on streams



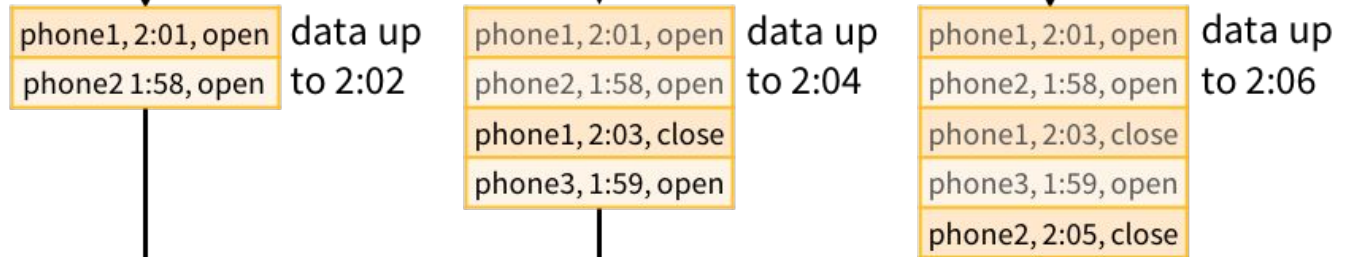
Arriving Records



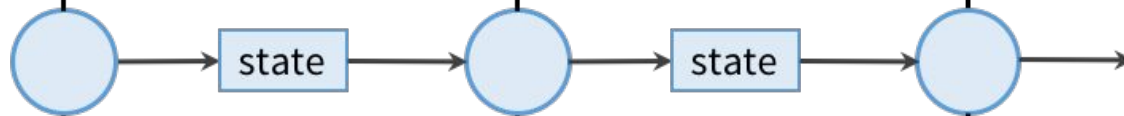
System Time



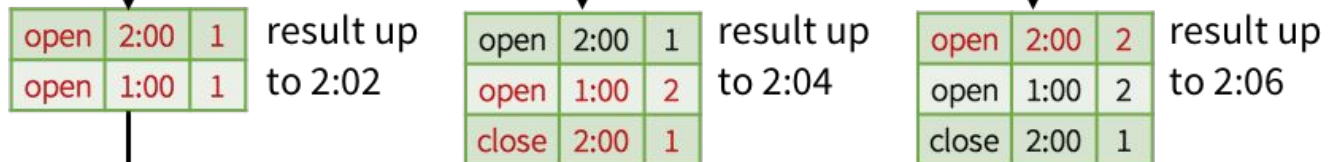
Input Table



Query



Result Table



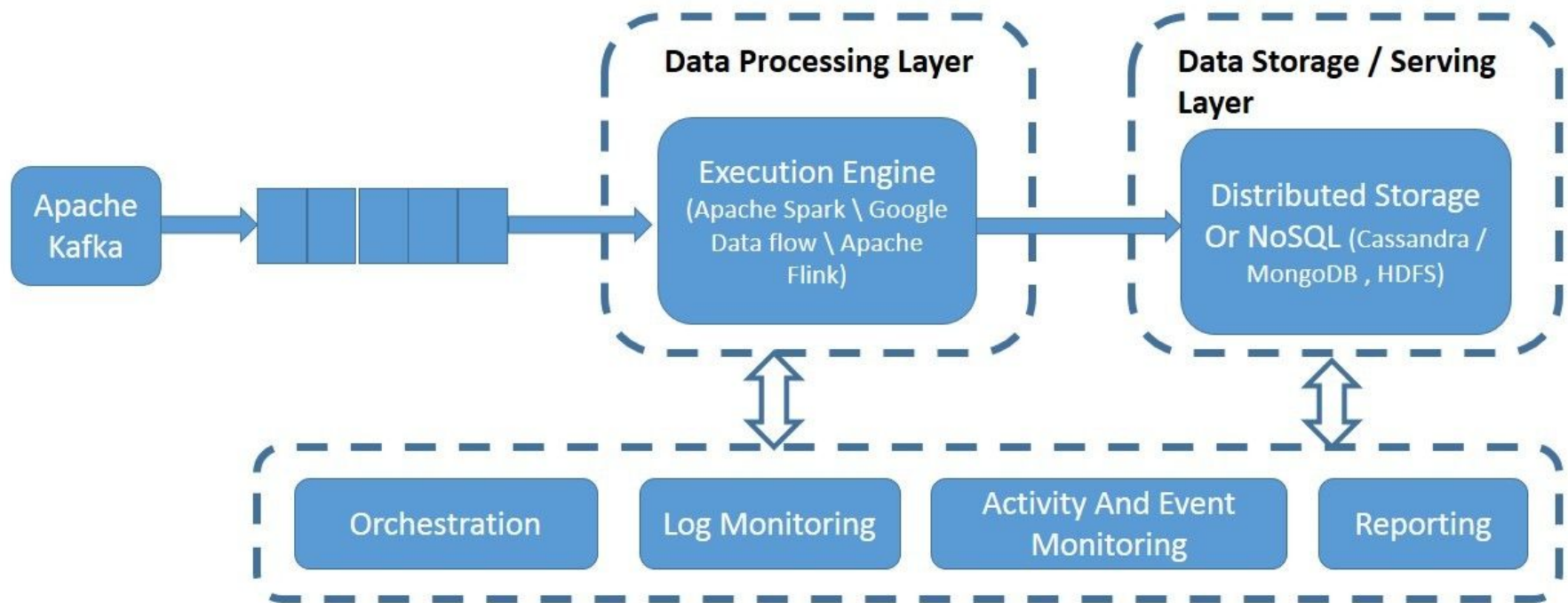
Output

Update Mode



Kappa Architecture

Kappa Architecture



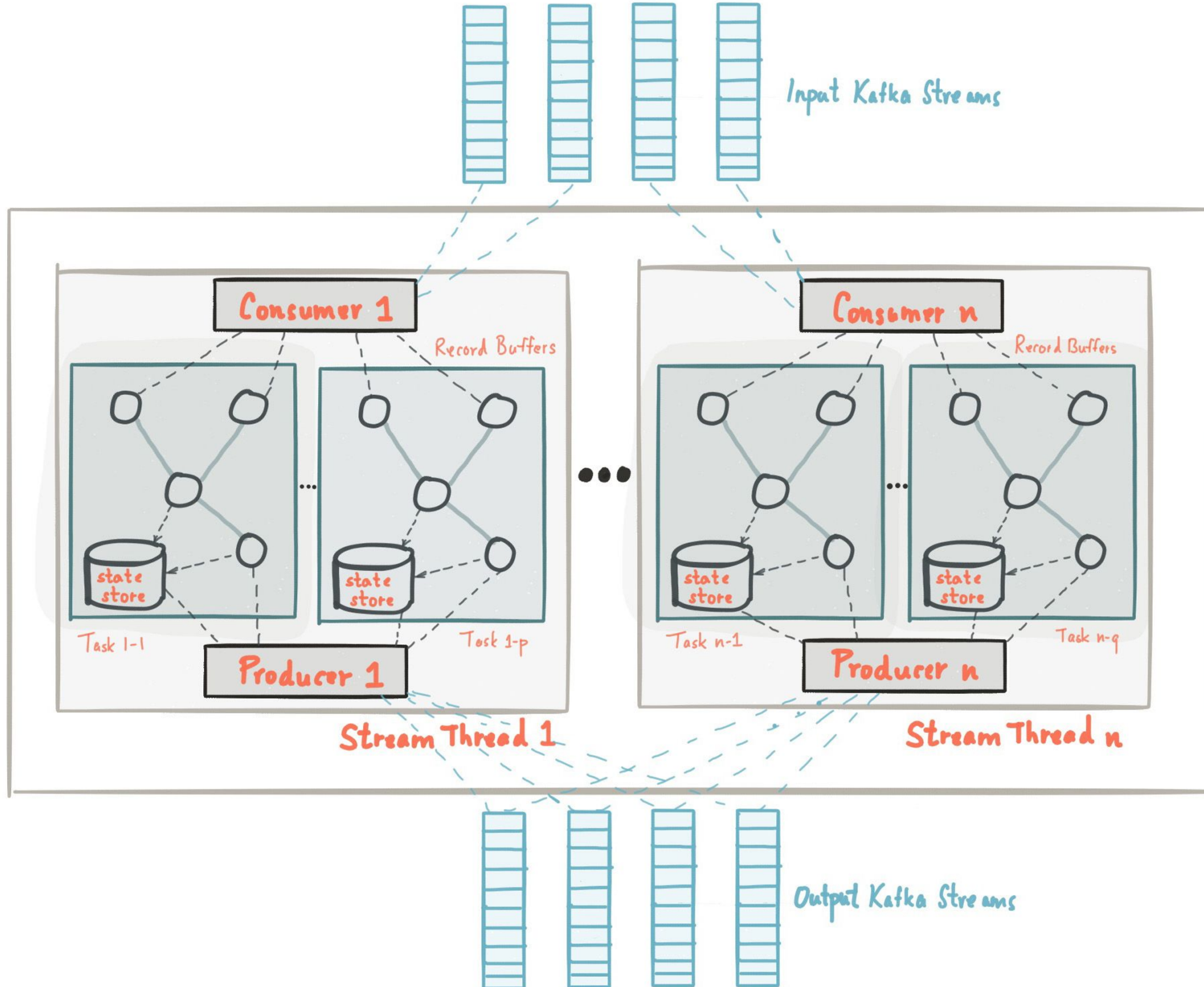
<https://jonboulineau.me/blog/architecture/kappa-architecture>

Siddharth Mittal



© 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/>

Kafka Streams

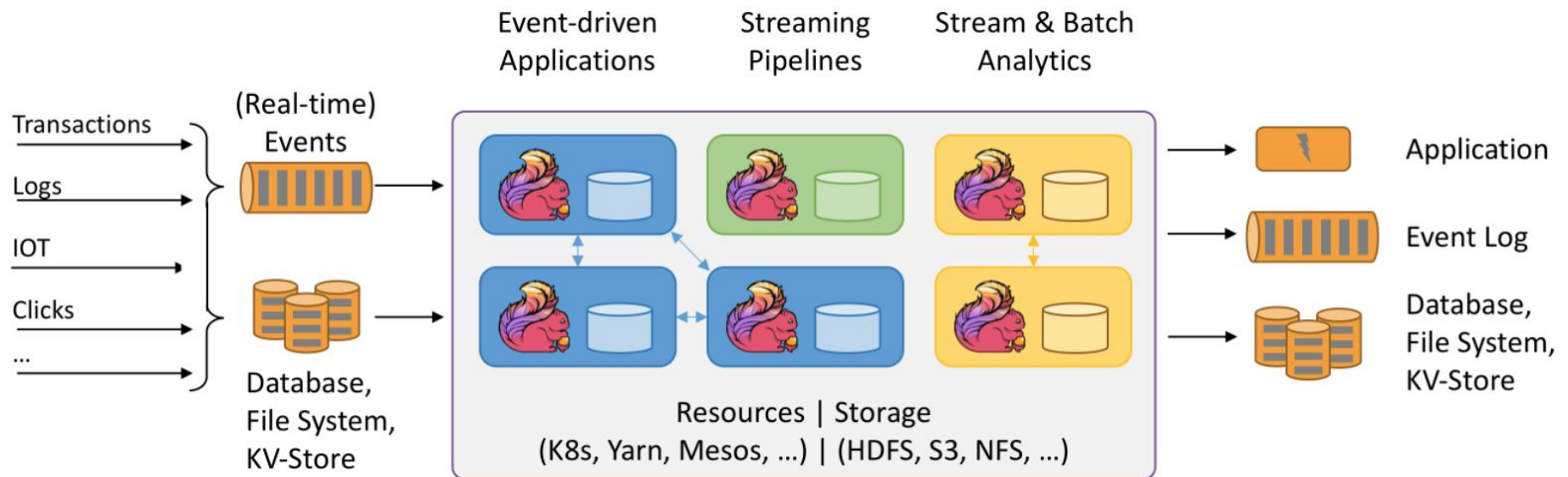


Kafka Streams

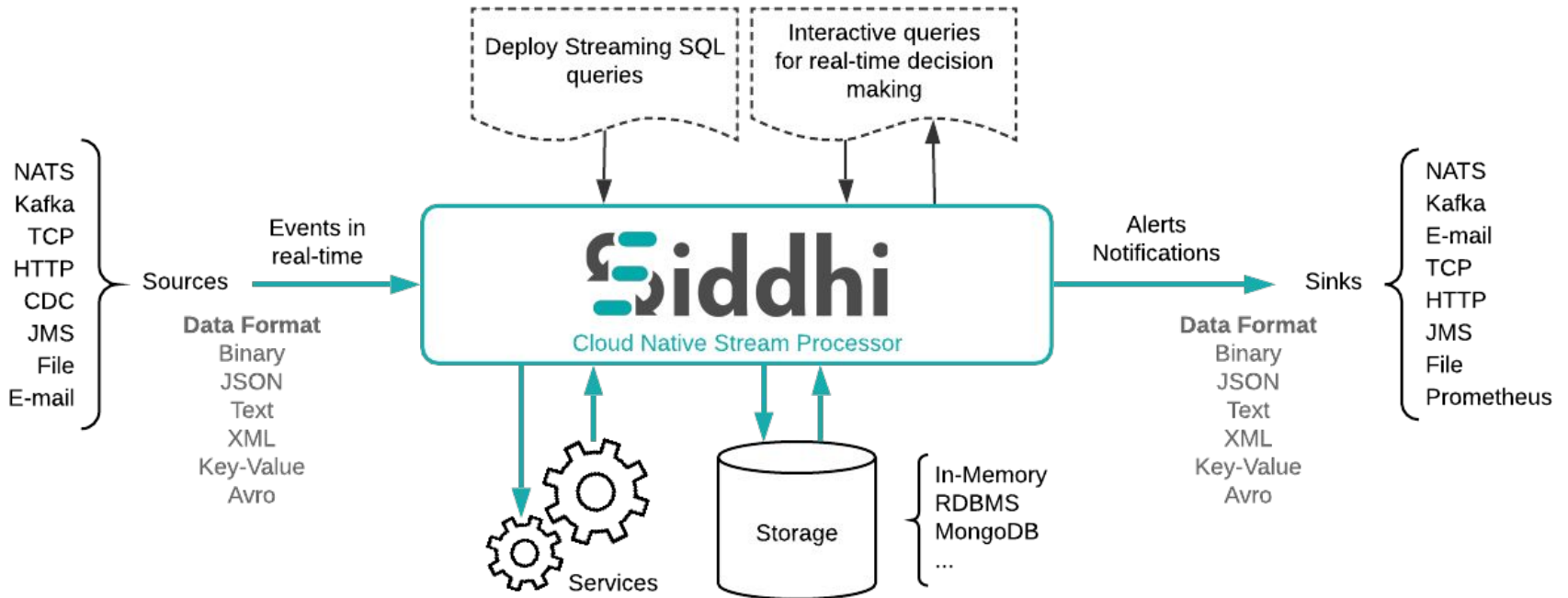
- Event-at-a-time processing (not microbatch) with millisecond latency
- Stateful processing including distributed joins and aggregations
- A convenient DSL
- Windowing with out-of-order data using a DataFlow-like model
- Distributed processing and fault-tolerance with fast failover
- Reprocessing capabilities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



Apache Flink



siddhi.io



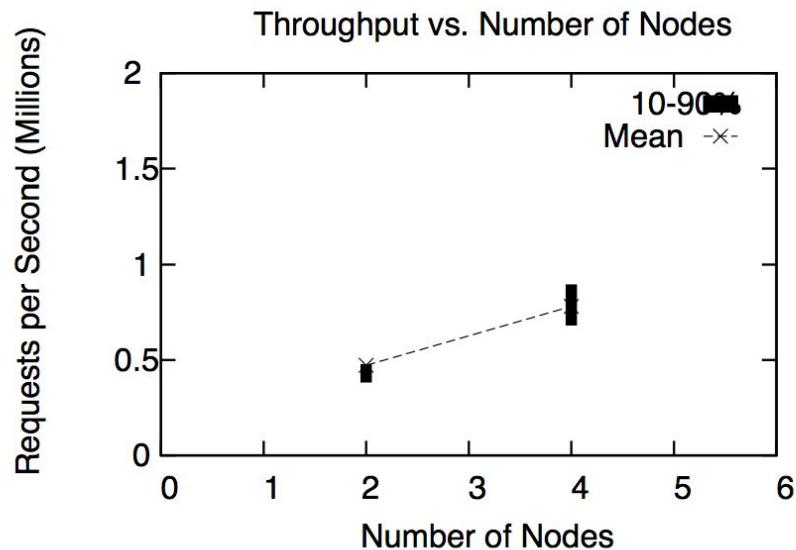
Siddhi

- 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

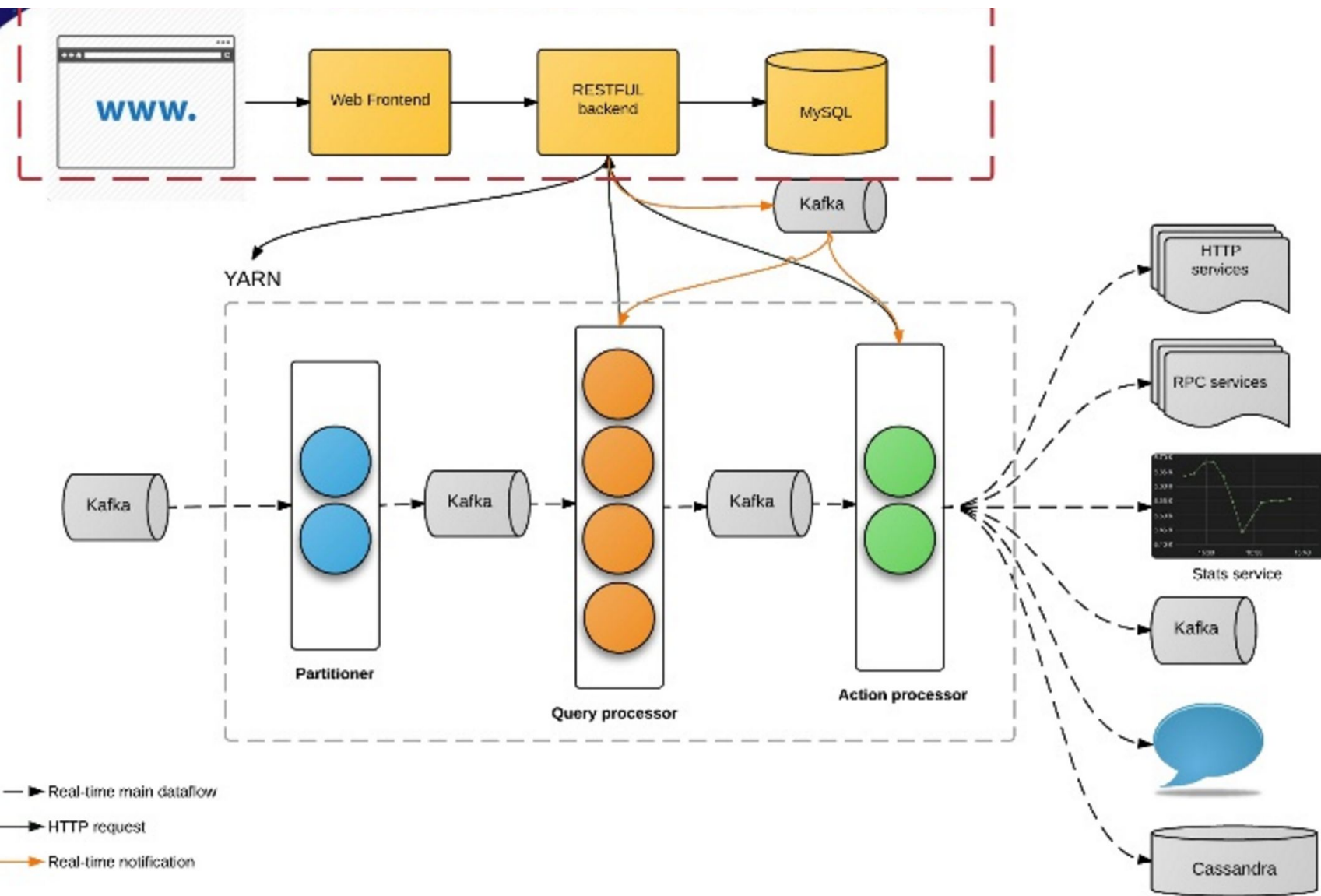


SiddhiQL

```
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 (a few years ago)



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/>

Long-running aggregation

```
define aggregation SweetProductionAggregation
from SweetProductionStream
select name, sum(amount) as totalAmount
group by name
aggregate every hour...year
```

```
from GetTotalSweetProductionStream as b join SweetProductionAggregation as a
on a.name == b.name
within b.duration
per b.interval
select a.AGG_TIMESTAMP, a.name, a.totalAmount
insert into HourlyProductionStream;
```



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
 - Stateful Stream Processing



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/>