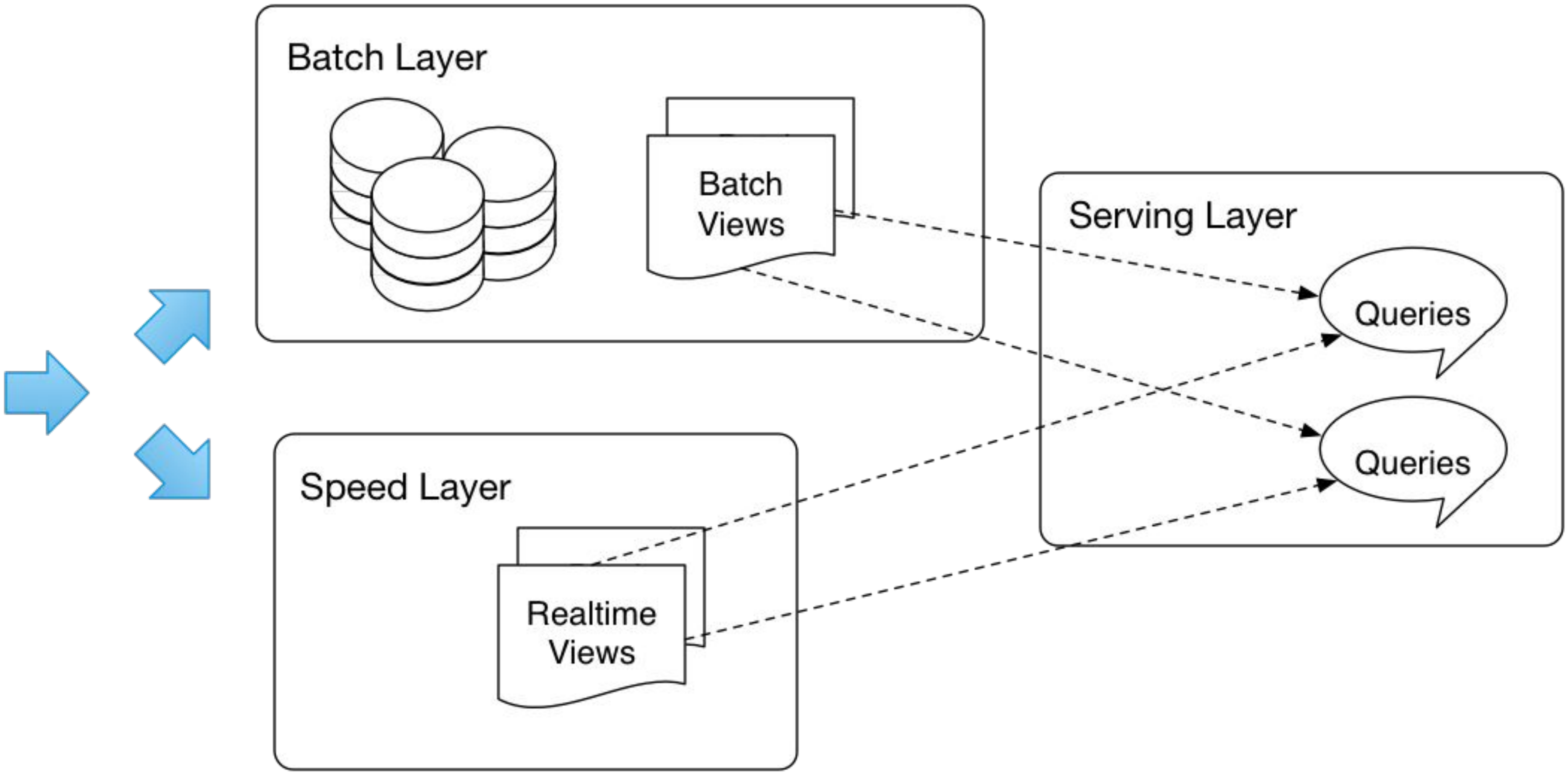
Cloud Computing and Big Data

Realtime Big Data

Oxford University
Software Engineering Programme
July 2021

Recap on the Lambda Architecture



Streaming

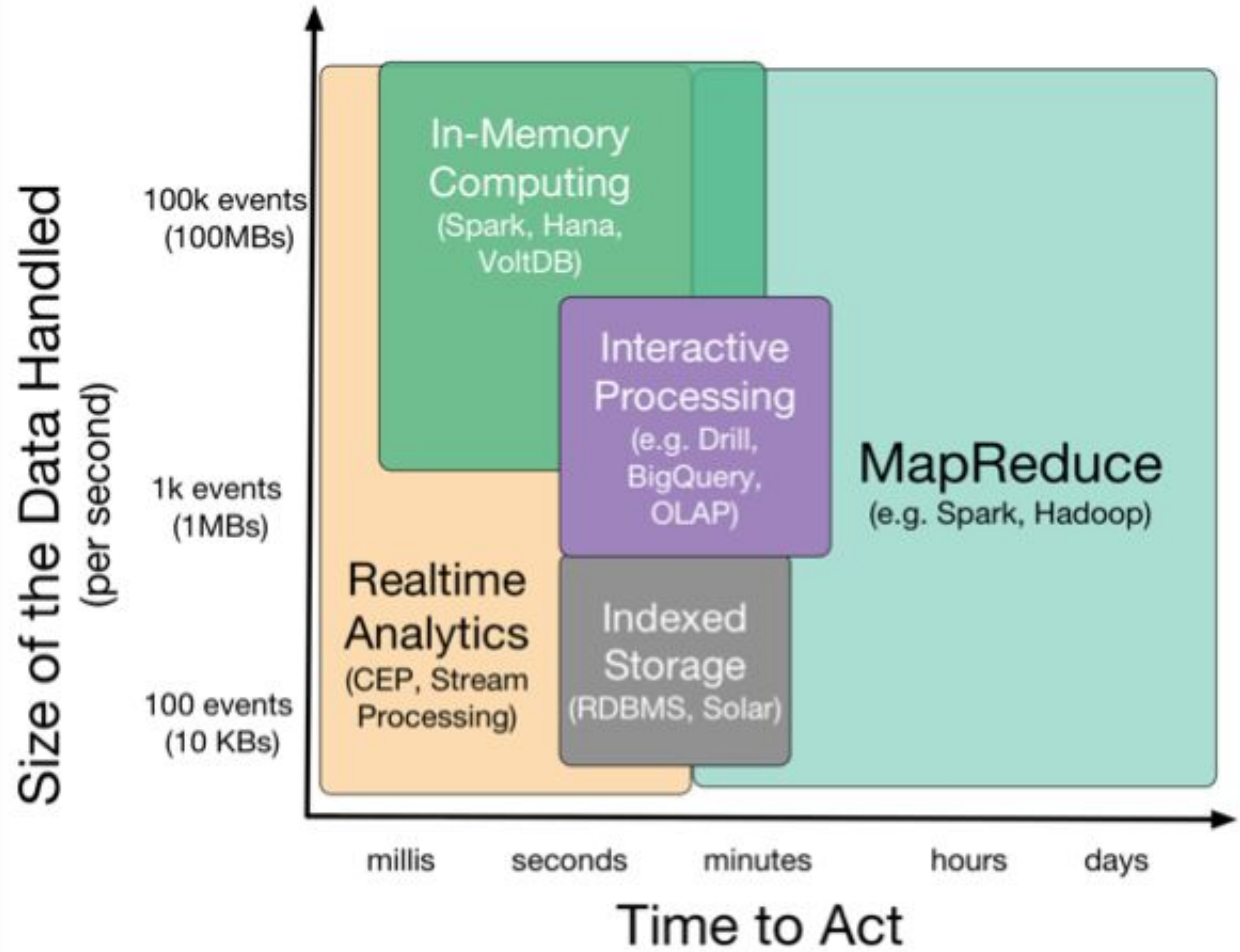
- Continuous data flow
  - "Unbounded streams of data"
- Usually uses a message distribution system
  - JMS
  - Apache Kafka
  - MQTT
  - Etc
- An unbounded set of events with time
  - <t1, E1>, <t2, E2>, ….., <tn, En>, ….

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)
MapReduce
(e.g. Spark, Hadoop)
Realtime
Analytics
(CEP, Stream
Processing)
Indexed
Storage
(RDBMS, Solar)
millis
seconds
minutes
hours
days
Time to Act

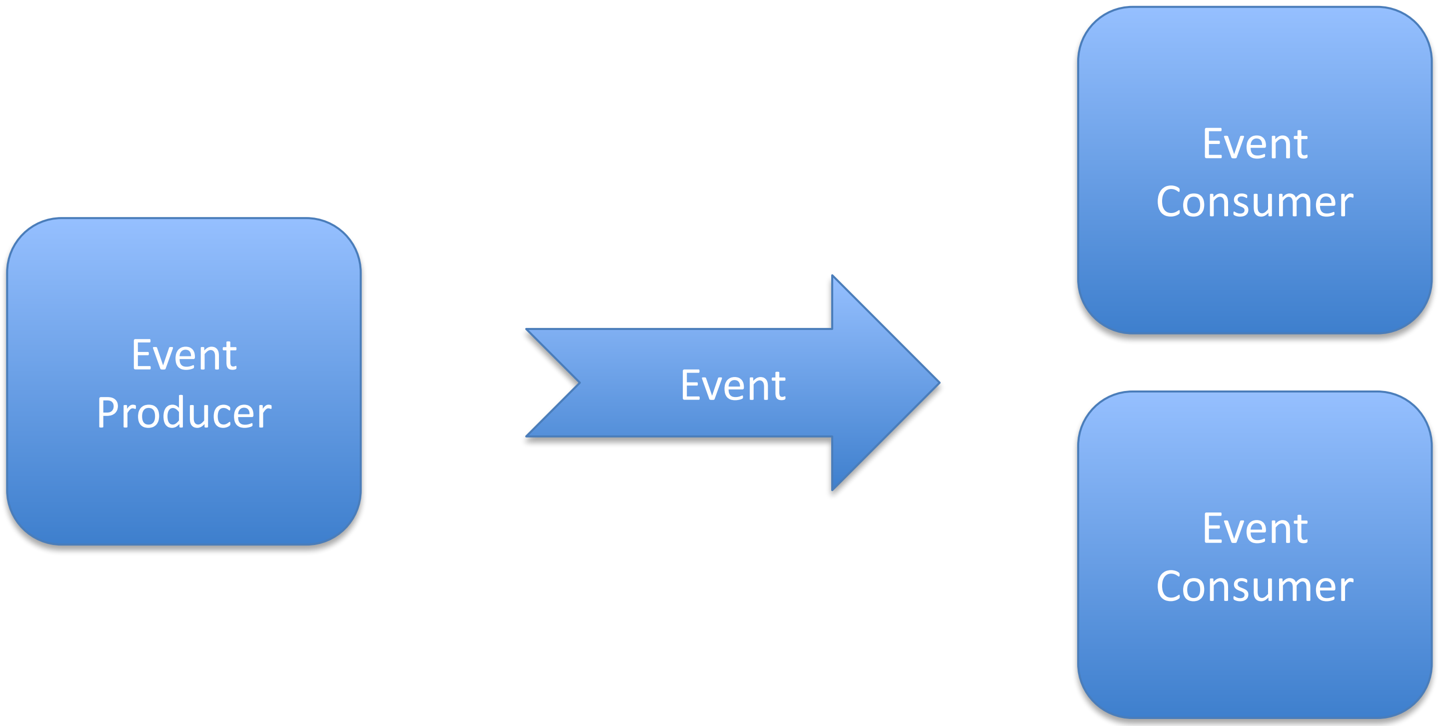
Approaches to Streaming

- 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

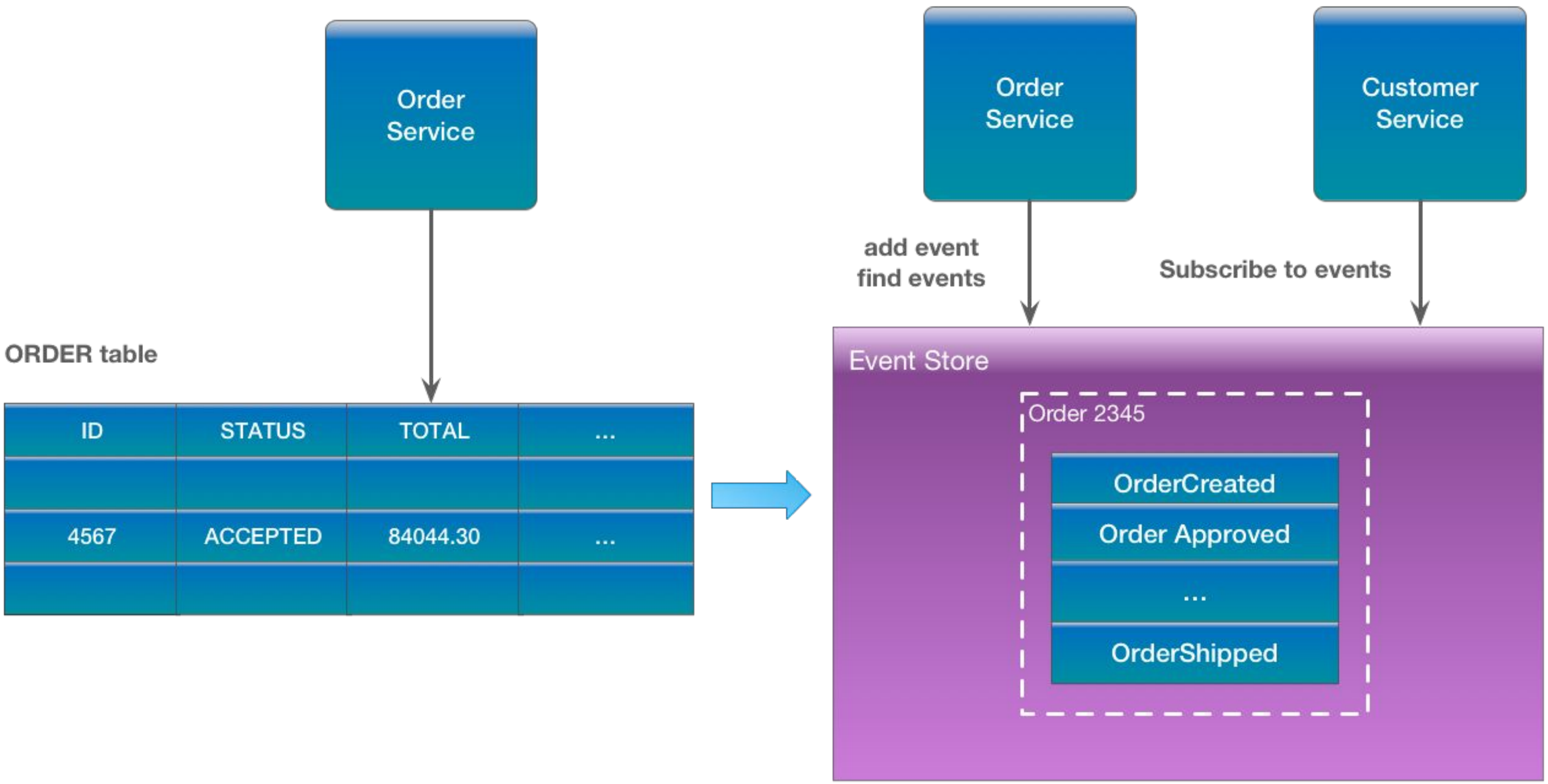
Data distribution

- You need to get the events to the processing systems

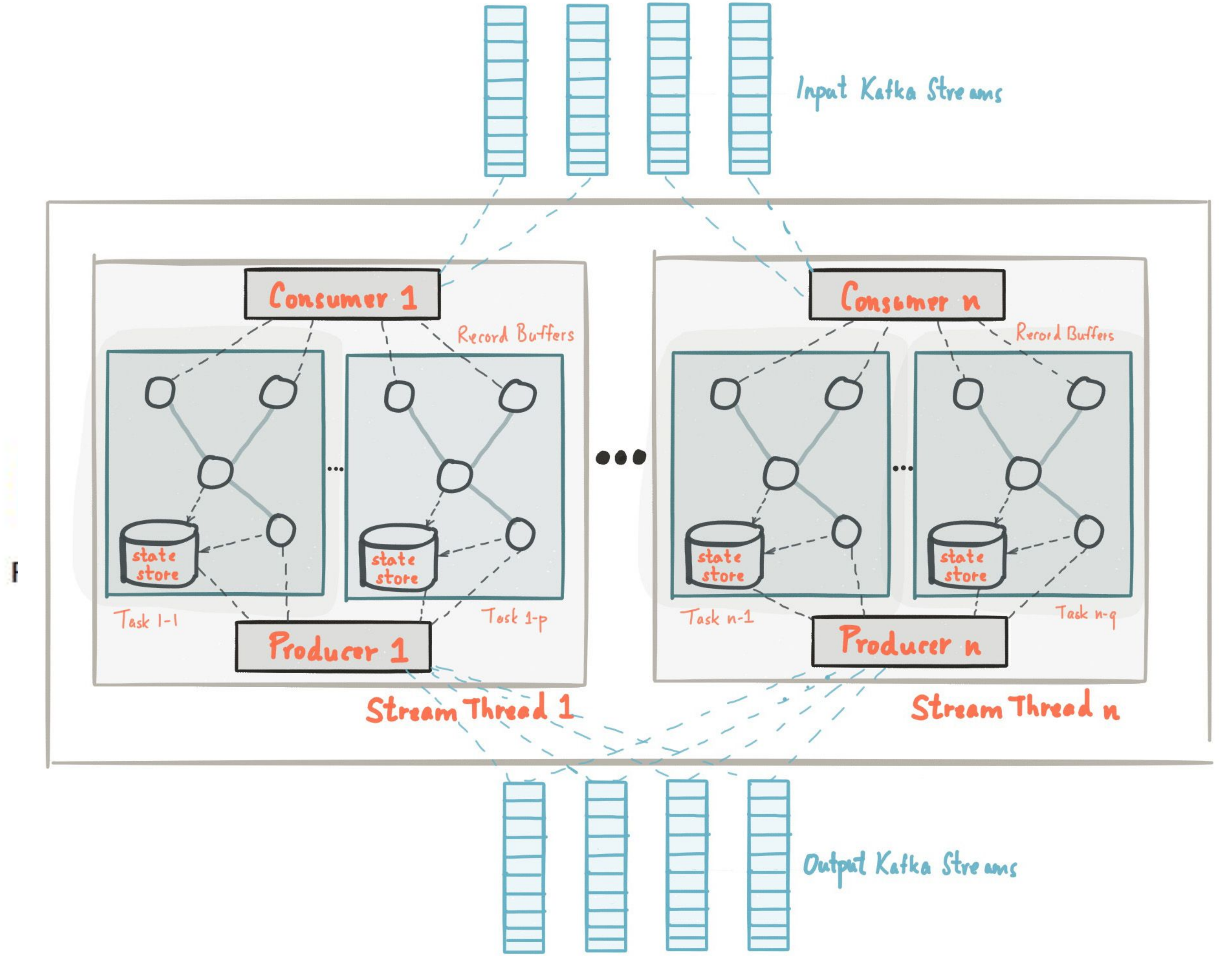
Event Driven Architecture



Event Sourcing

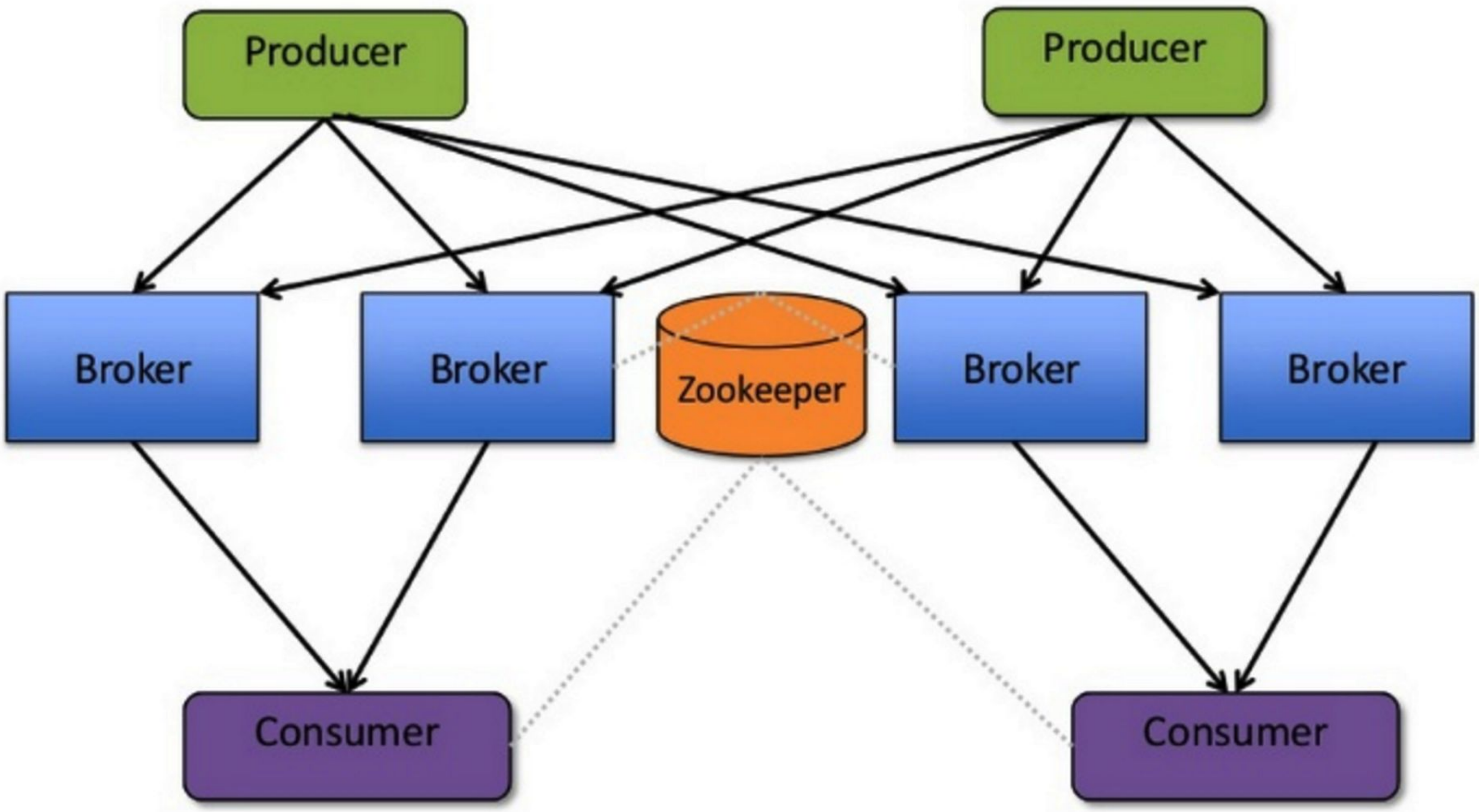


https://eventuate.io/whyeventsourcing.html



Input Kafka Streams
Consumer 1
Record Buffers
state store
state store
Task 1-1
Task 1-p
Producer 1
Stream Thread 1
Consumer n
Record Buffers
state store
state store
Task n-1
Task n-q
Producer n
Stream Thread n
Output Kafka Streams

Apache Kafka

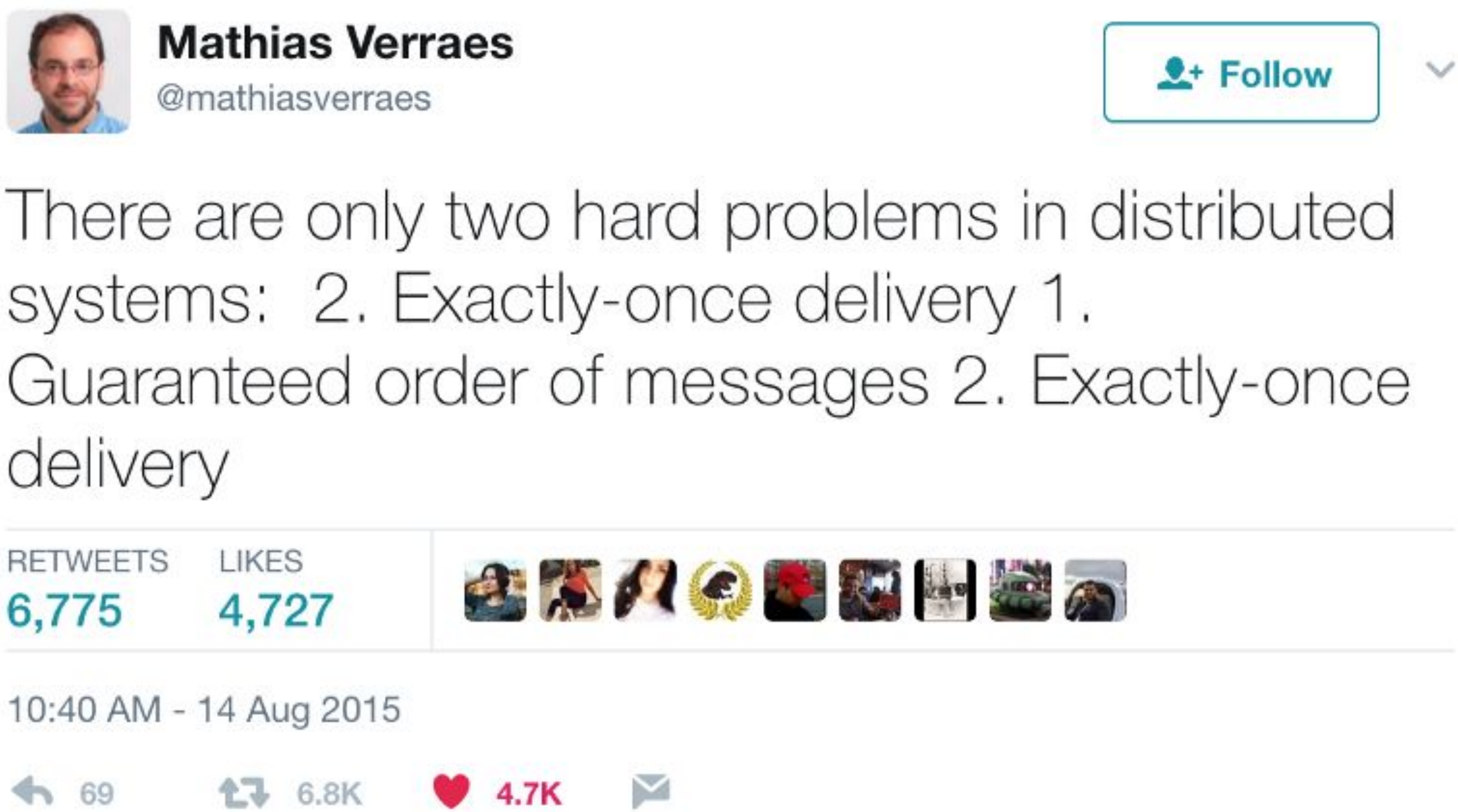


Source: http://www.slideshare.net/charmalloc/

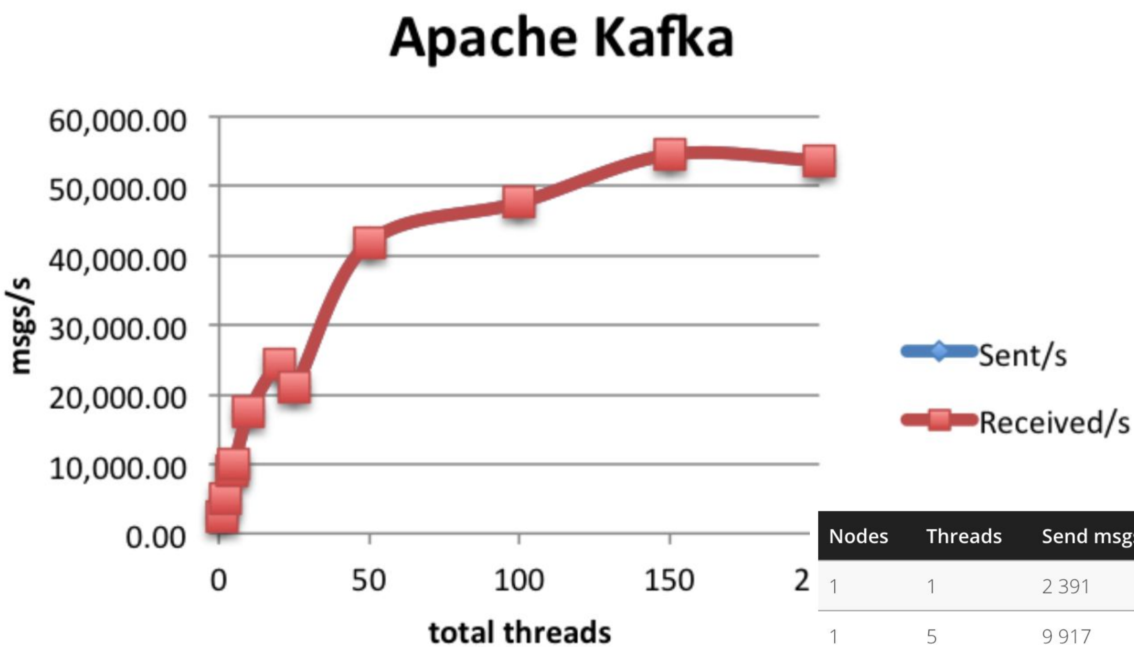
Kafka

- Applying “big data” approaches to messaging:
  - Partitioning
  - Multiple brokers
  - Elastically scalable
  - Supports clusters of co-ordinated consumers
  - Automatic re-election of leaders

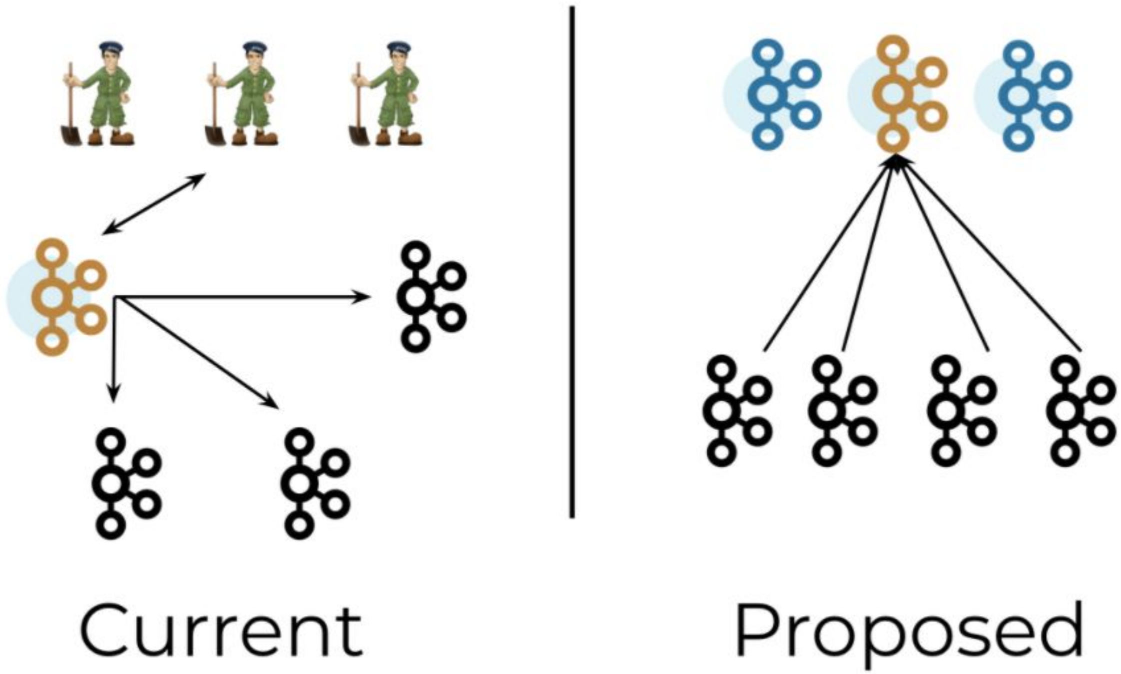
Kafka exactly-once semantics



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

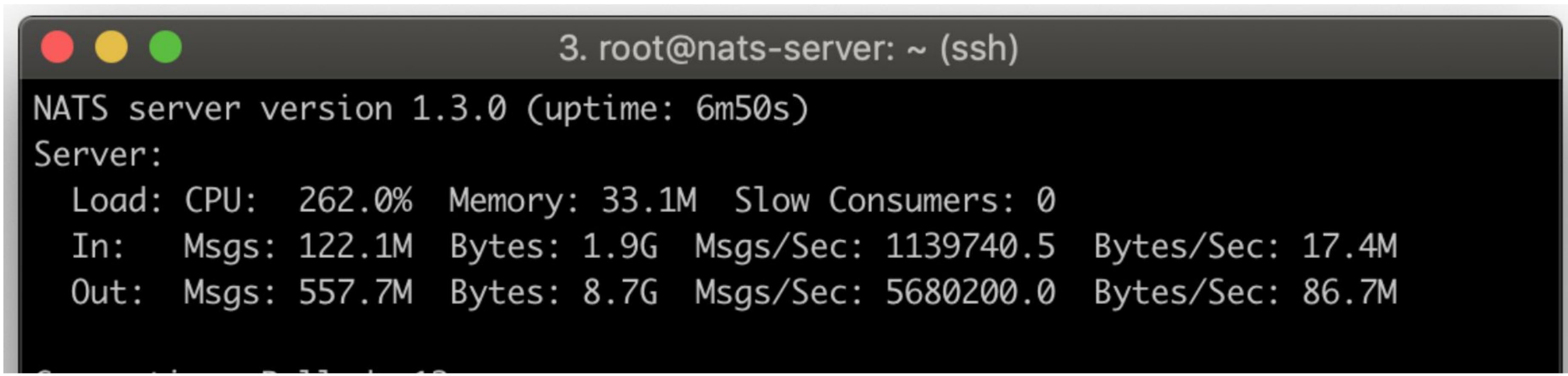
Several administrative tools shipped as part of the Kafka release still allow direct communication with ZooKeeper. Worse still, there are still one or two operations that can't be done except through this direct ZooKeeper communication.

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

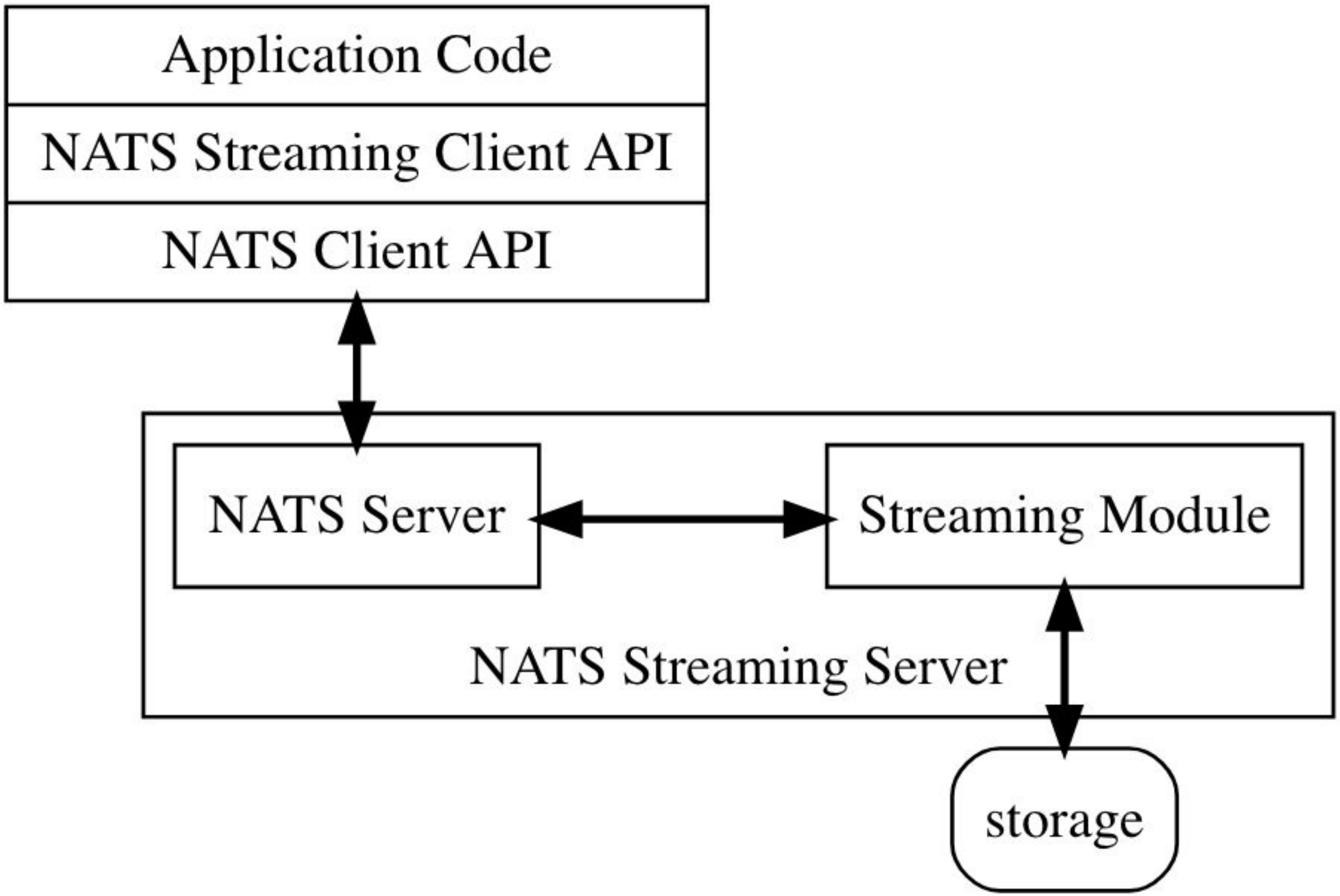
3Mb Docker image

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




```
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

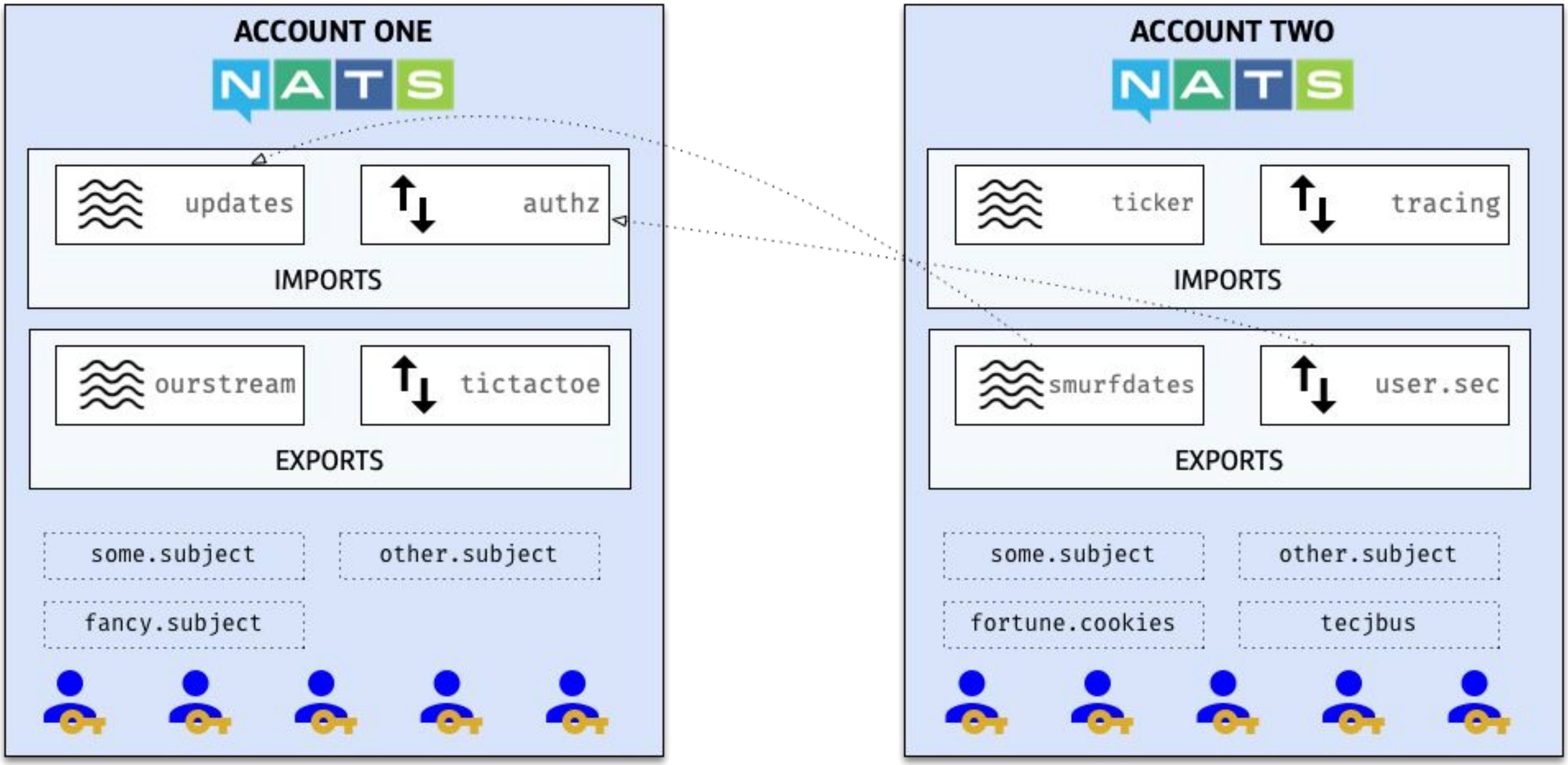
Kevin Hoffman

Apr 9 · 7 min read

Decentralized Authorization and Authentication with JWTs

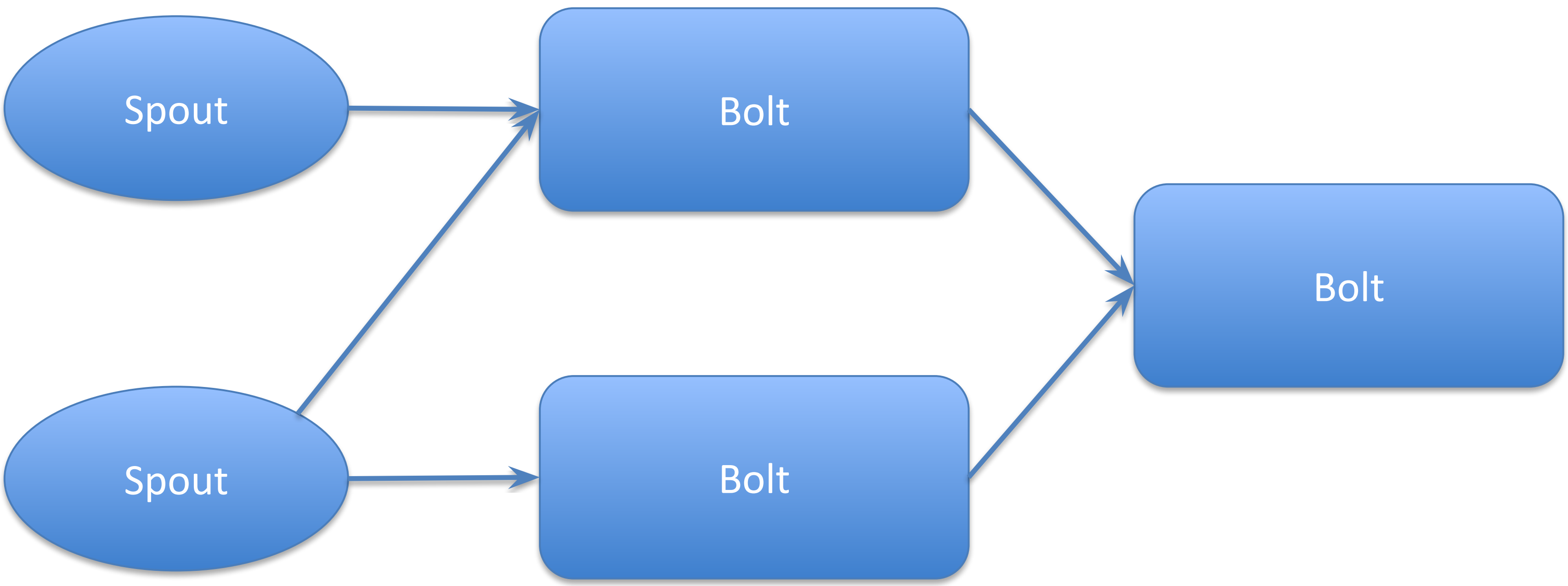
NATS is a lightweight, cloud native, open-source high-performance messaging system. In this post, I want to talk about security in the upcoming NATS 2.0 release—what it is, why you should care, and what it can do for you and your organization. But before I get into those details, I want to take a moment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

NATS security model



Processing the data

Apache Storm

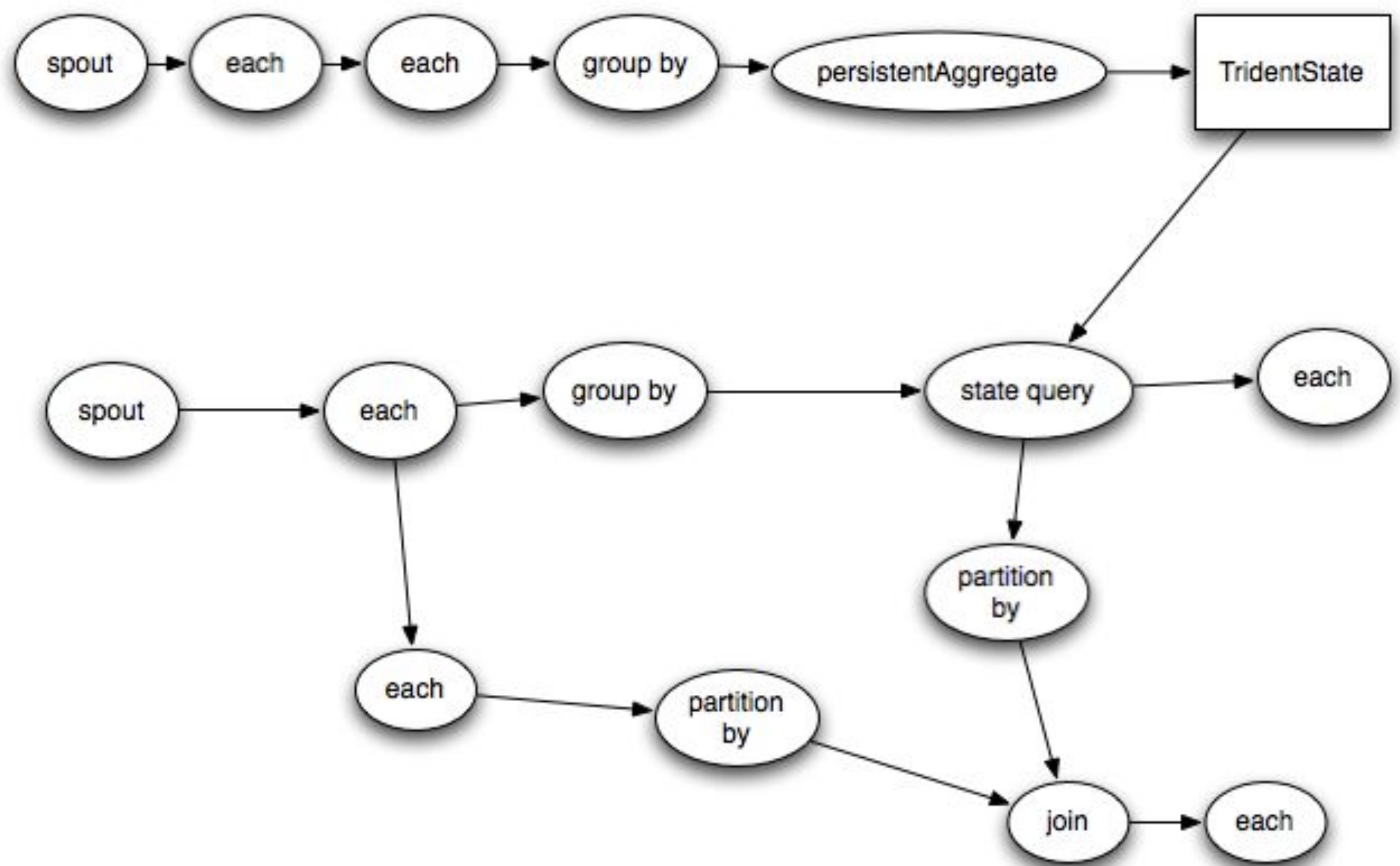


Note: another DAG

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

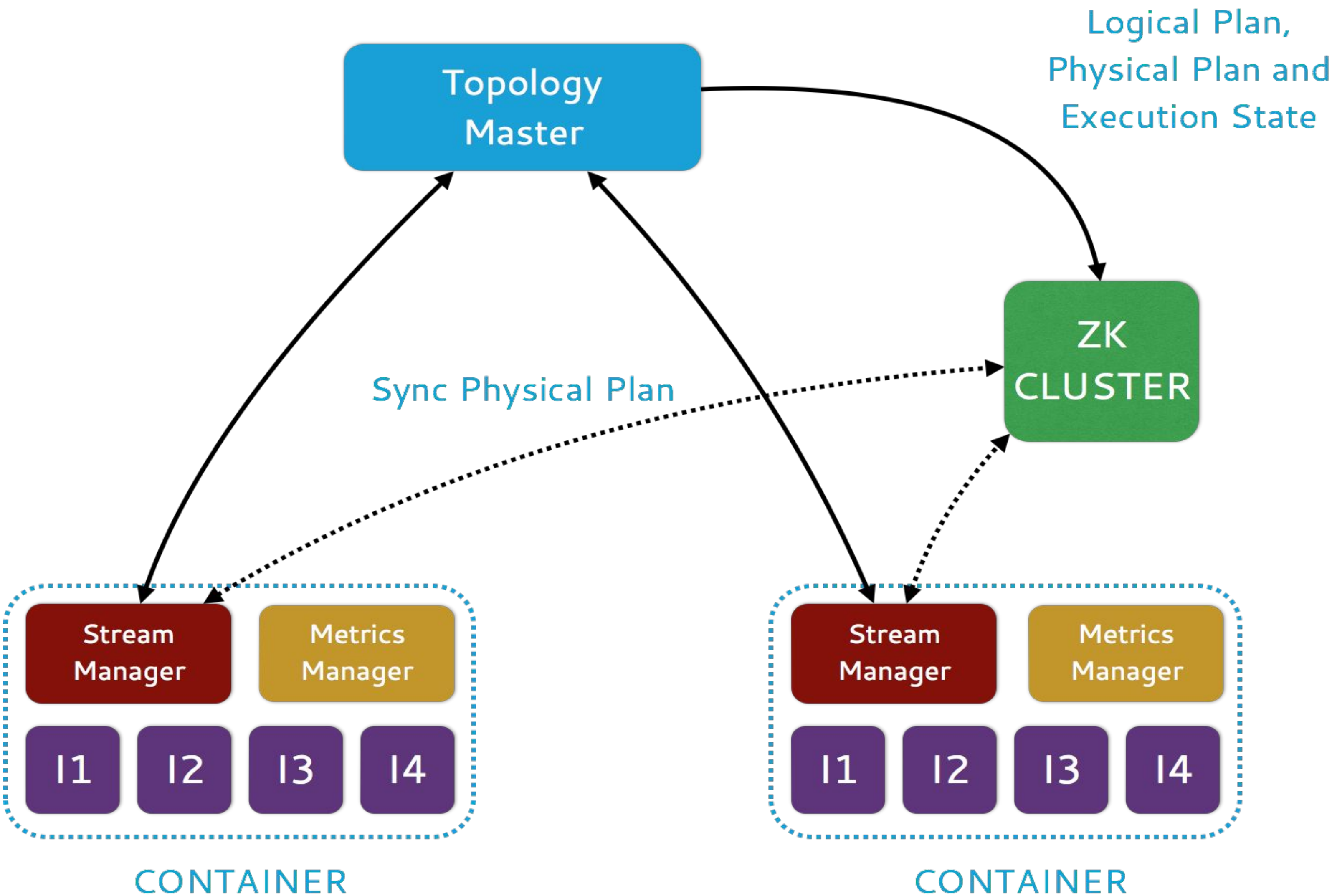
Apache Storm
Trident (micro-batch)



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



Logical Plan,
Physical Plan and
Execution State
Topology
Master
ZK
CLUSTER
Sync Physical Plan
Stream
Manager
Metrics
Manager
I1
I2
I3
I4
Stream
Manager
Metrics
Manager
I1
I2
I3
I4
CONTAINER
CONTAINER

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

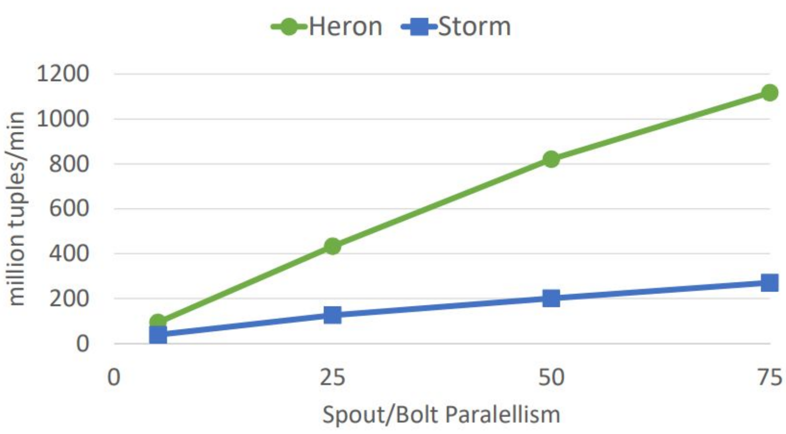


Fig. 2. Throughput with acks

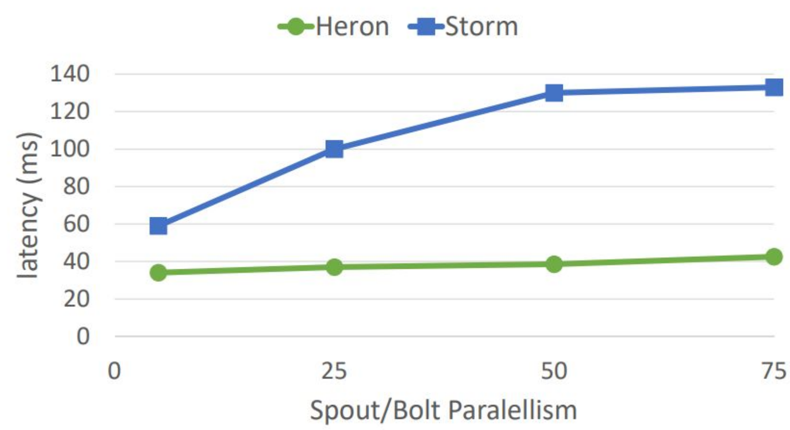
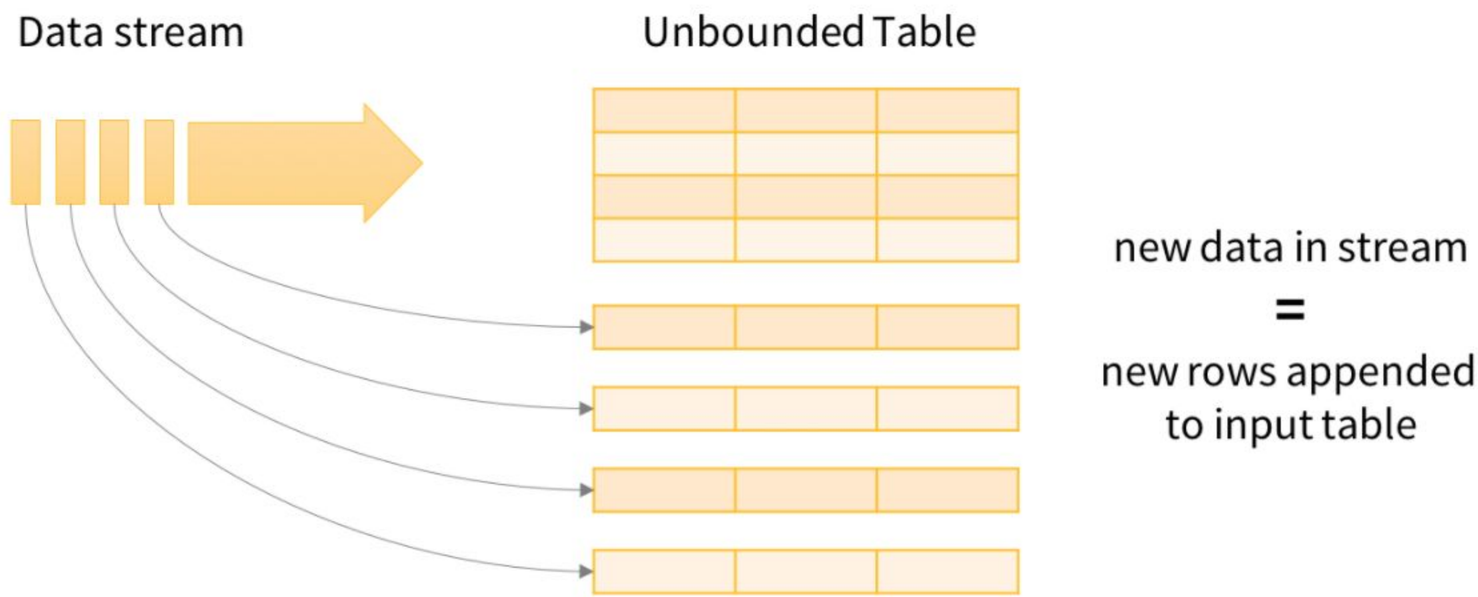


Fig. 3. End-to-end latency with acks

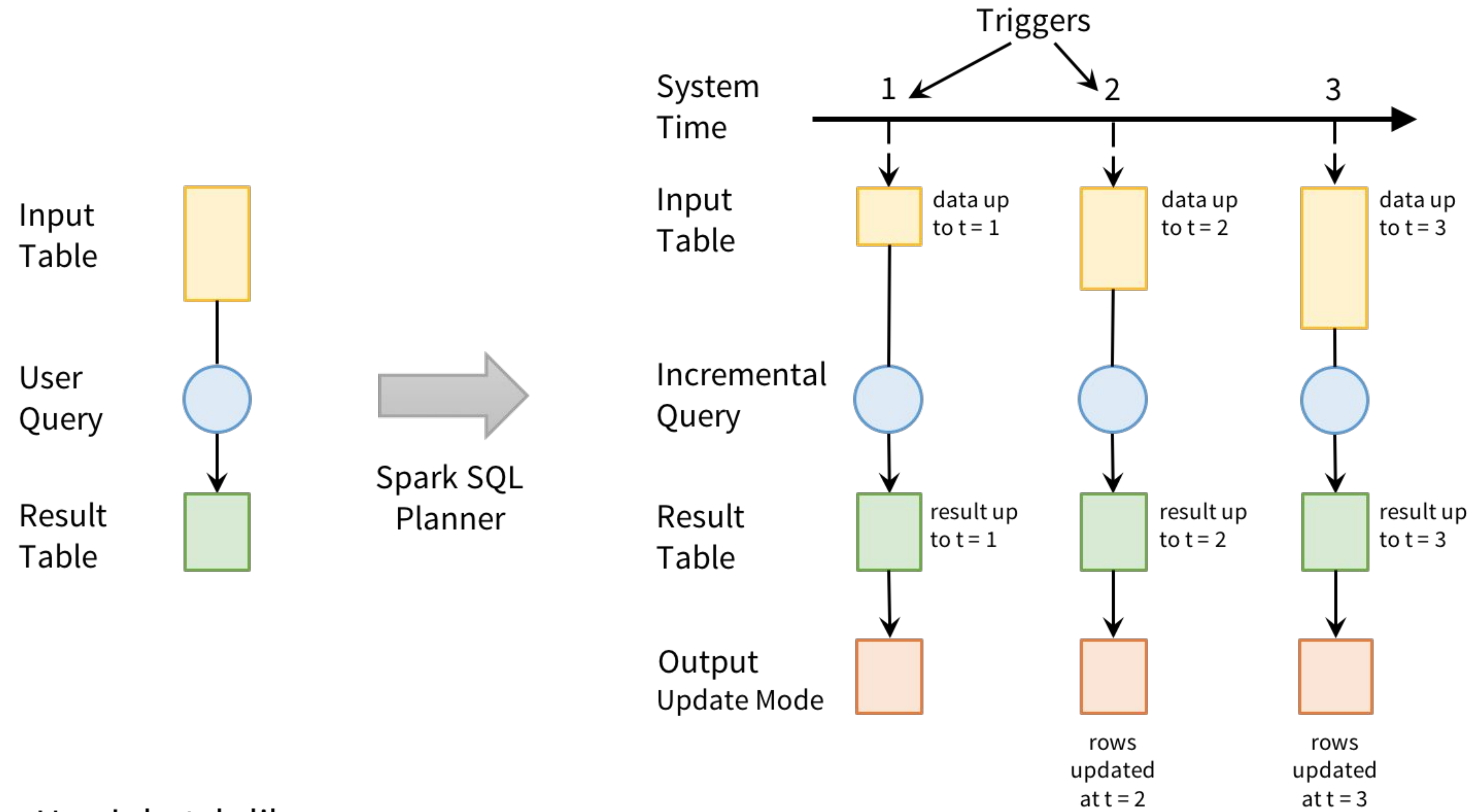
Apache Spark Streaming

Structured Streams in Spark

- Since Spark 2.0, there is a much better approach



Data stream as an unbounded Input Table

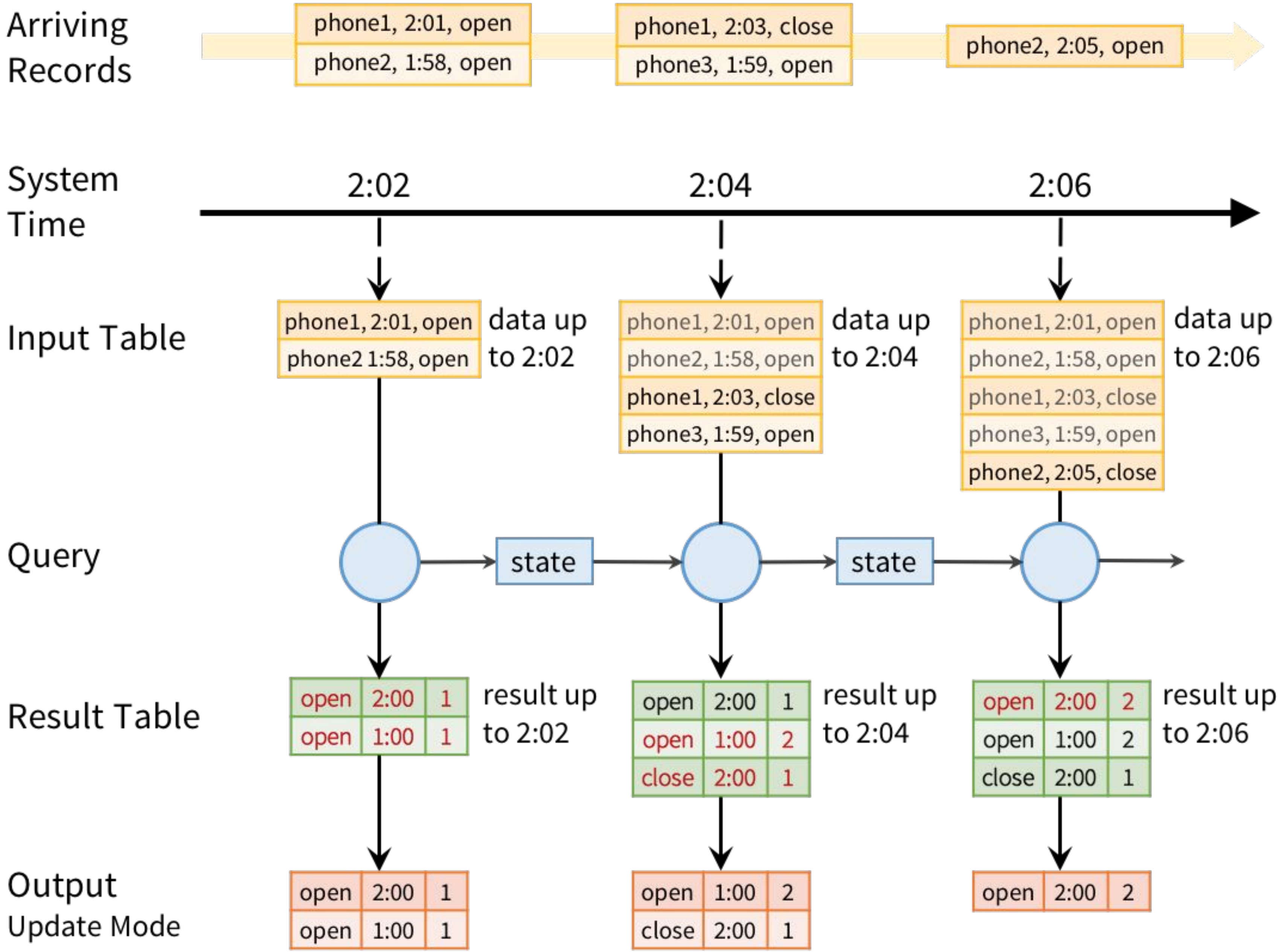


User's batch-like query on input table

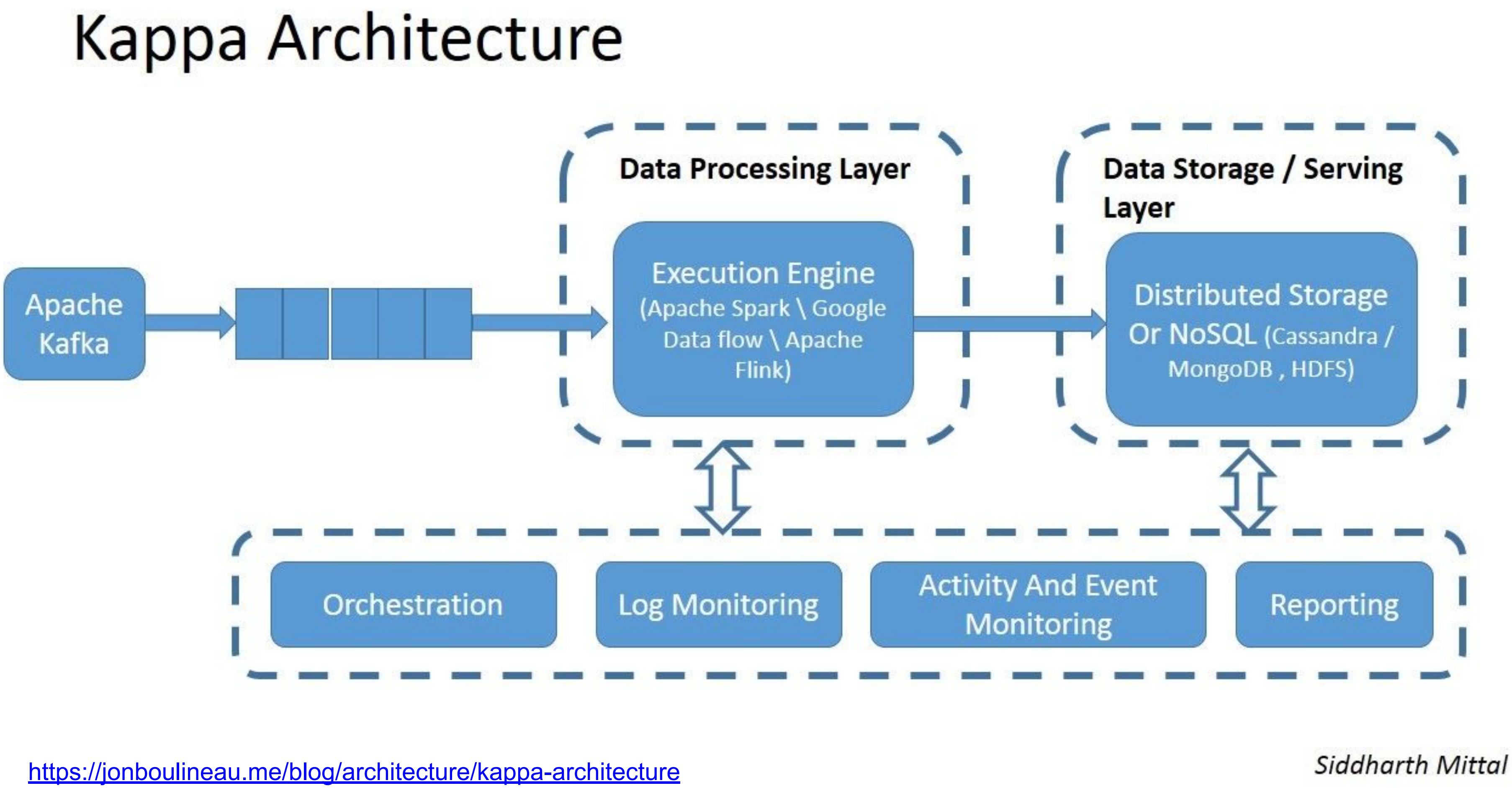
Incremental execution on streaming data

Structured Streaming Processing Model

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

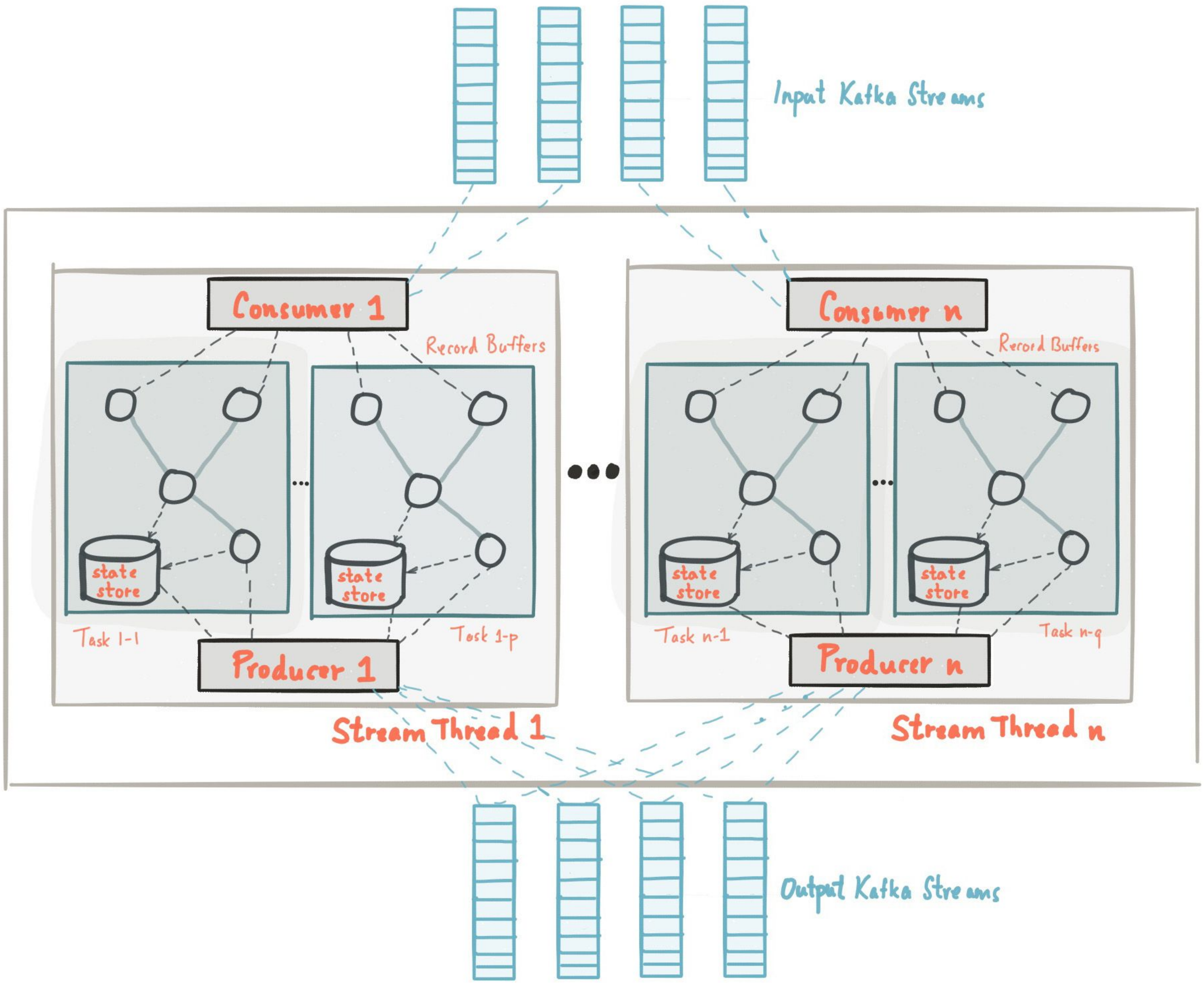

Arriving
Records
phone1, 2:01, open
phone2, 1:58, open
phone1, 2:03, close
phone3, 1:59, open
phone2, 2:05, open
System
Time
2:02
2:04
2:06
Input Table
phone1, 2:01, open
phone2 1:58, open
data up
to 2:02
phone1, 2:01, open
phone2, 1:58, open
phone1, 2:03, close
phone3, 1:59, open
data up
to 2:04
phone1, 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
state
state
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

Kappa Architecture



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

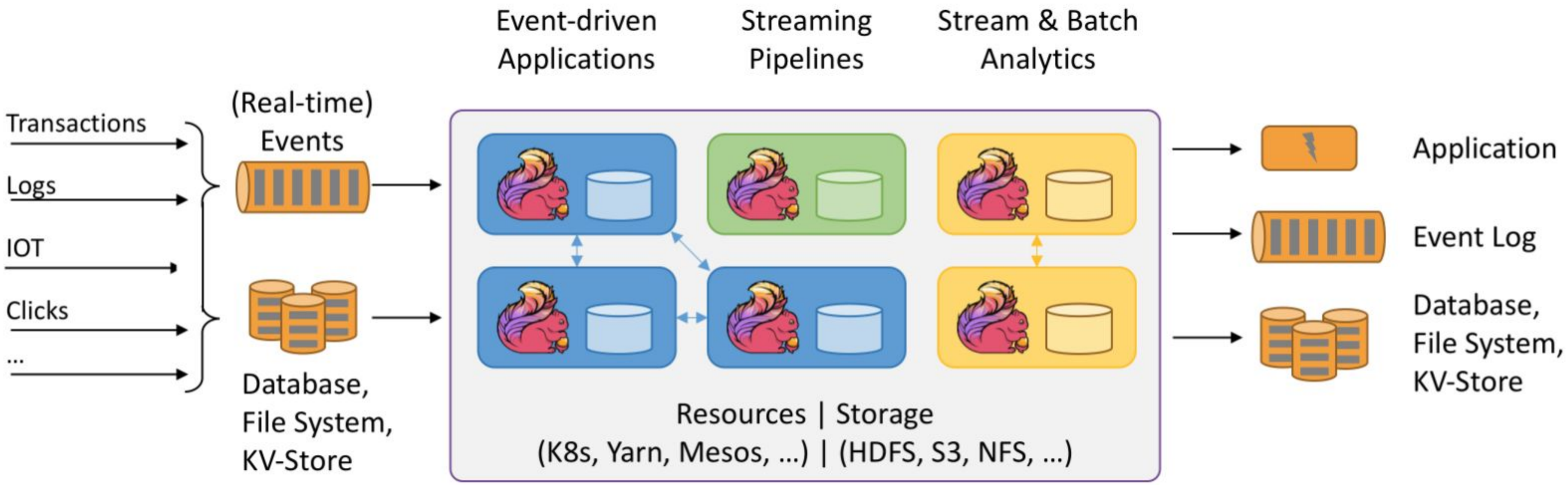
Kafka Streams



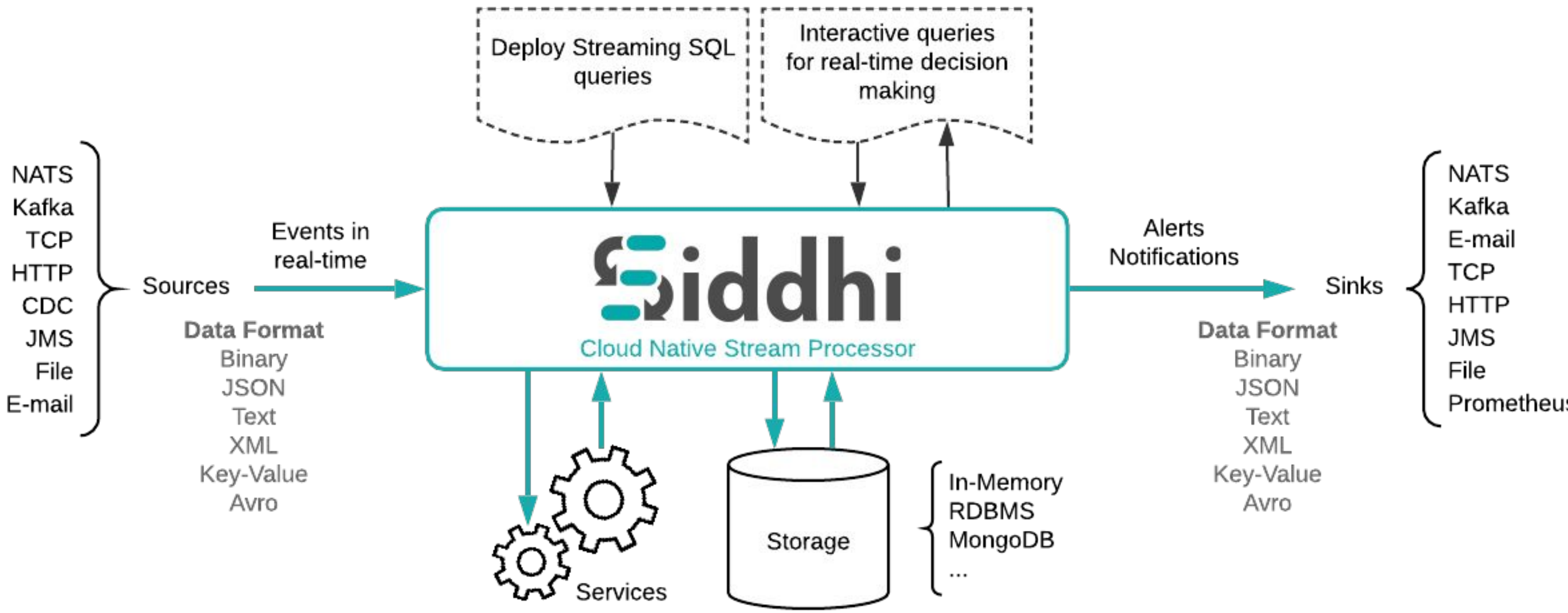
Kafka Streams

- 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



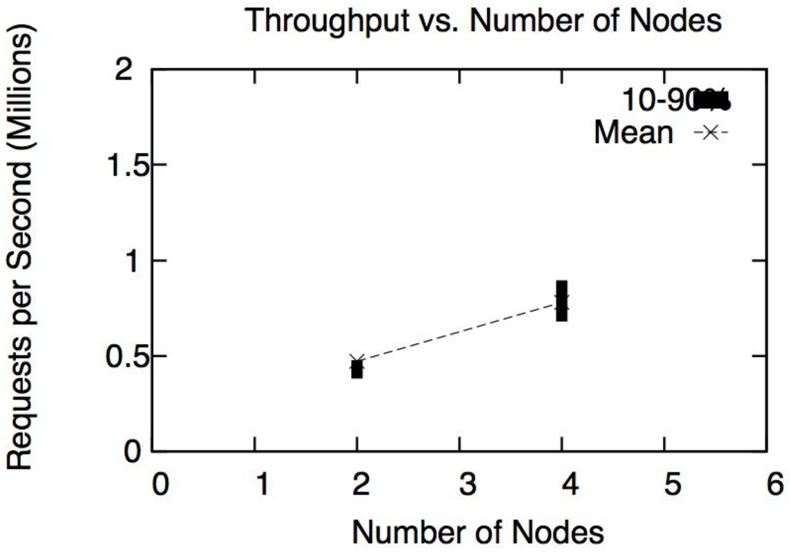
Deploy Streaming SQL queries
Interactive queries for real-time decision making
NATS
Kafka
TCP
HTTP
CDC
JMS
File
E-mail
Sources
Events in real-time
Data Format
Binary
JSON
Text
XML
Key-Value
Avro
Siddhi
Cloud Native Stream Processor
Alerts Notifications
Sinks
NATS
Kafka
E-mail
TCP
HTTP
JMS
File
Prometheus
Data Format
Binary
JSON
Text
XML
Key-Value
Avro
Services
Storage
In-Memory
RDBMS
MongoDB
...

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

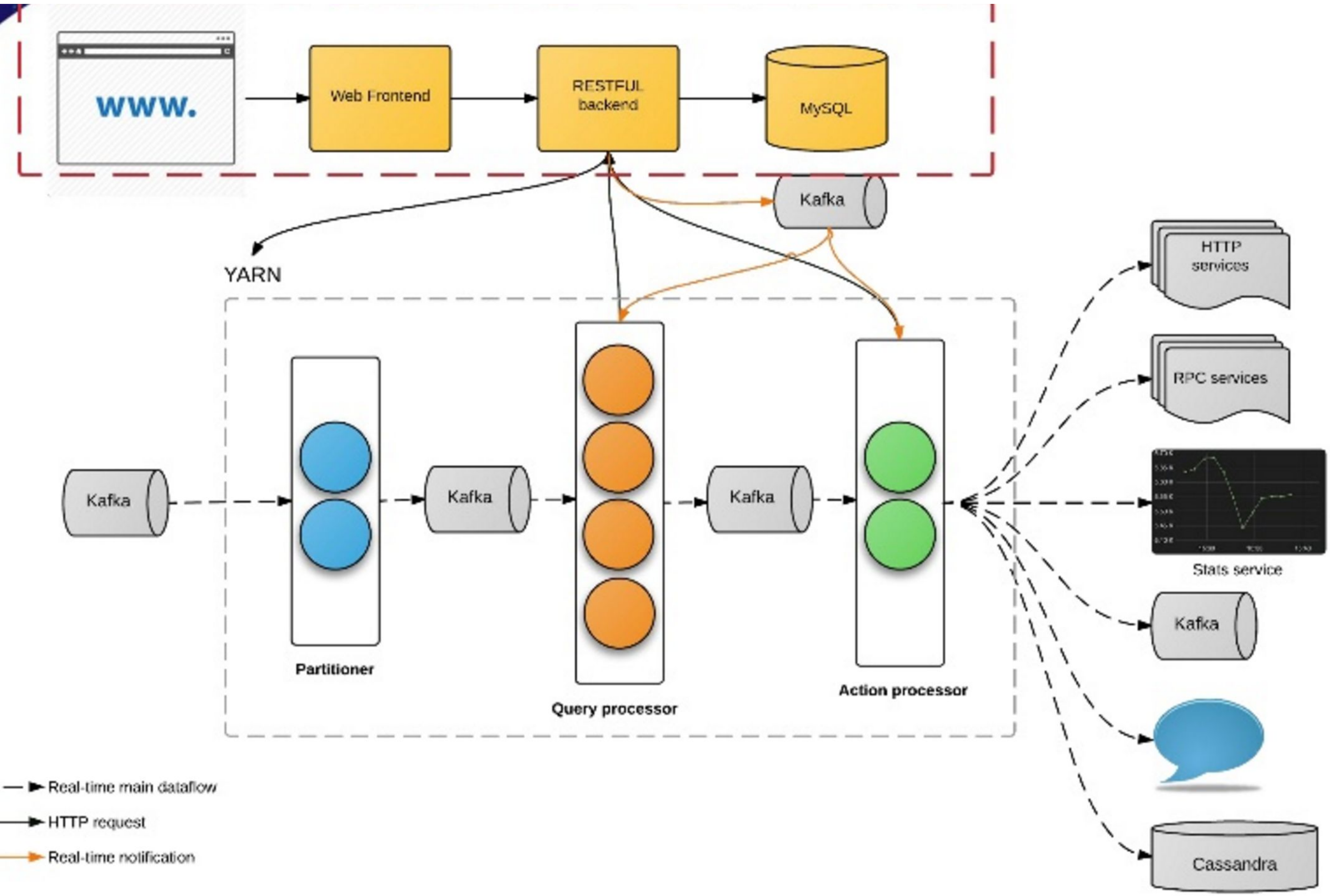
- Apache Licensed Open Source on Github
  - https://github.com/wso2/siddhi/
- Pluggable into Storm, Spark and Kafka Streams
- Supports millions of events/sec
- http://freo.me/DEBS_Siddhi



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

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?