instaclustr

Processing 200K Transactions per Second with Apache Spark and Apache Cassandra

Ben Bromhead

Boston Apache Spark Meetup

3 May 2018

Or…
we built our own metrics/monitoring stack and it was worth it…
but you probably shouldn’t do it… probably

/usr/bin/whoami

instaclustr

- Ben Bromhead, CTO of Instaclustr
- We provide managed Cassandra, Spark and Kafka in the cloud (AWS, GCP, Azure & Softlayer).
- We provide support and services as well for those in private data centers.
- Manage and support 2k+ nodes.

Agenda

instaclustr

- Introduction to Cassandra
- Why Spark + Cassandra
- Problem background and overall architecture
- Implementation process & lessons learned
- What's next?

Introduction to Cassandra

instaclustr

NoSQL database

- Highly available
- Master less
- Linear scalability
- Low latency

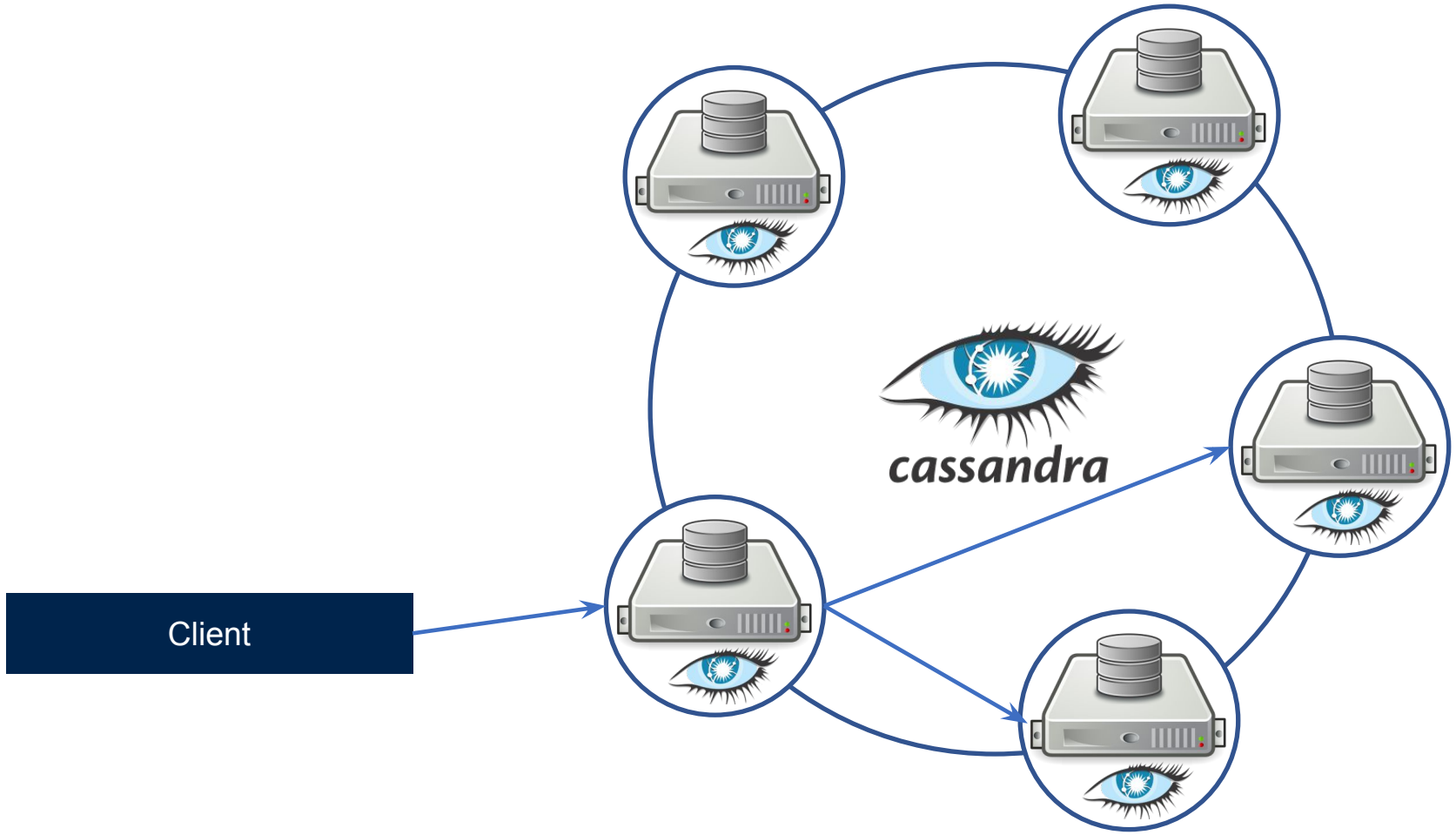
- No join
- Poor index
- Restricted filtering
- No ACID

- OLTP
- Data ingestion
- Design your requests first, your model second.

Introduction a Cassandra

instaclustr

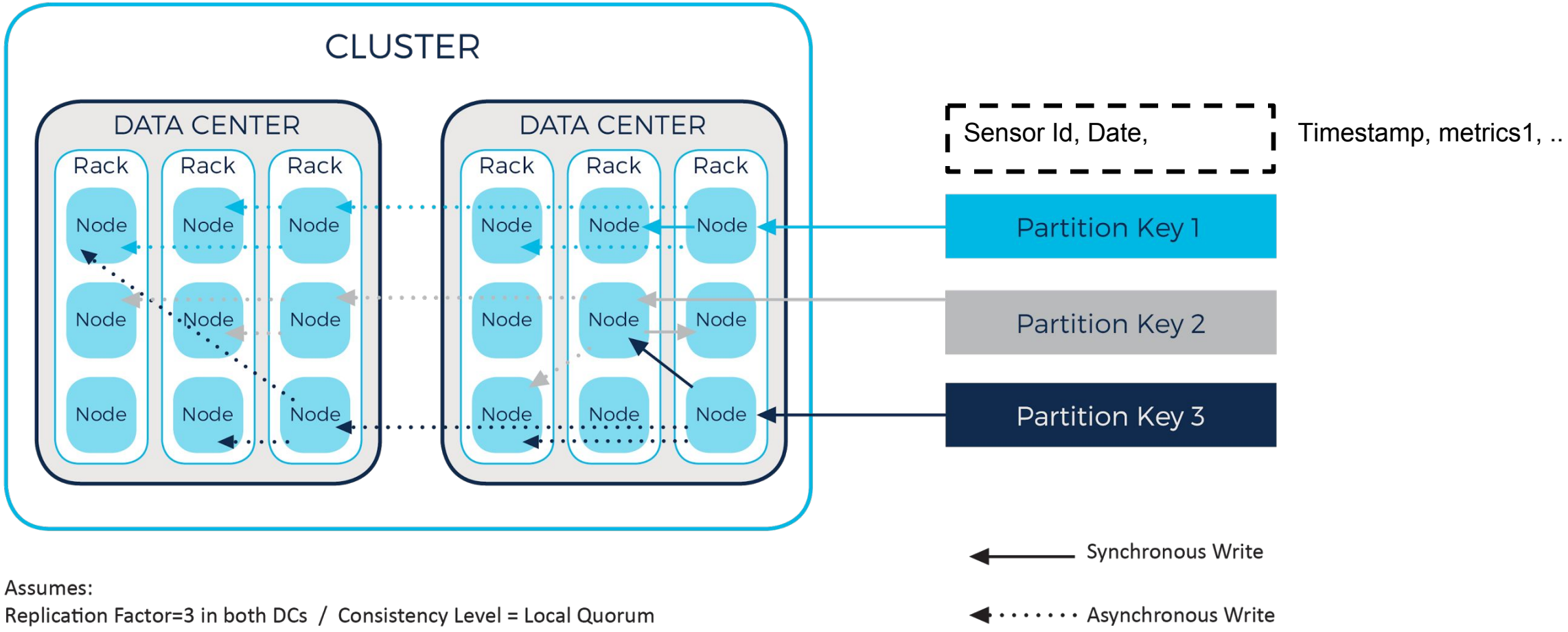
Cassandra is a Distributed Hash Table



Assume Replication Factor of 3

Introduction a Cassandra

instaclustr

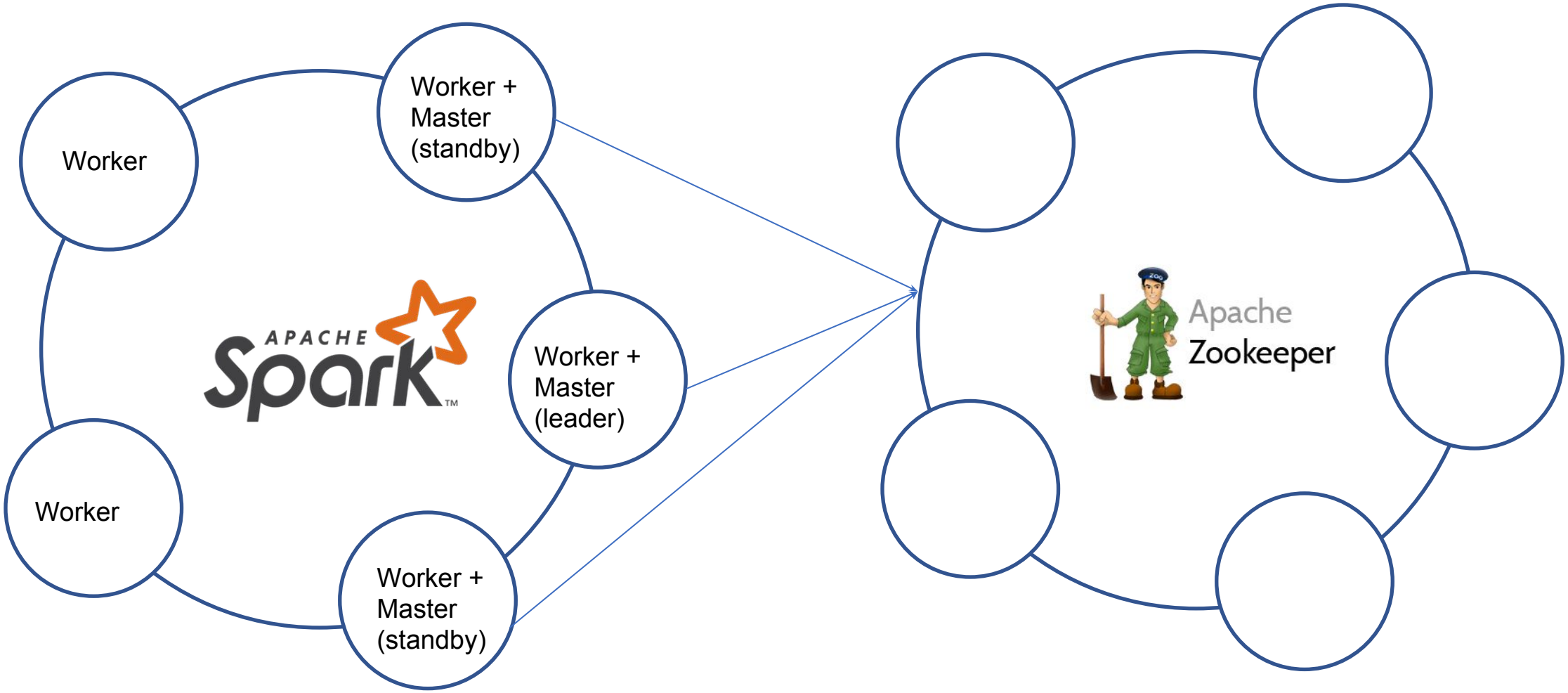


Assumes:
Replication Factor=3 in both DCs / Consistency Level = Local Quorum

Spark

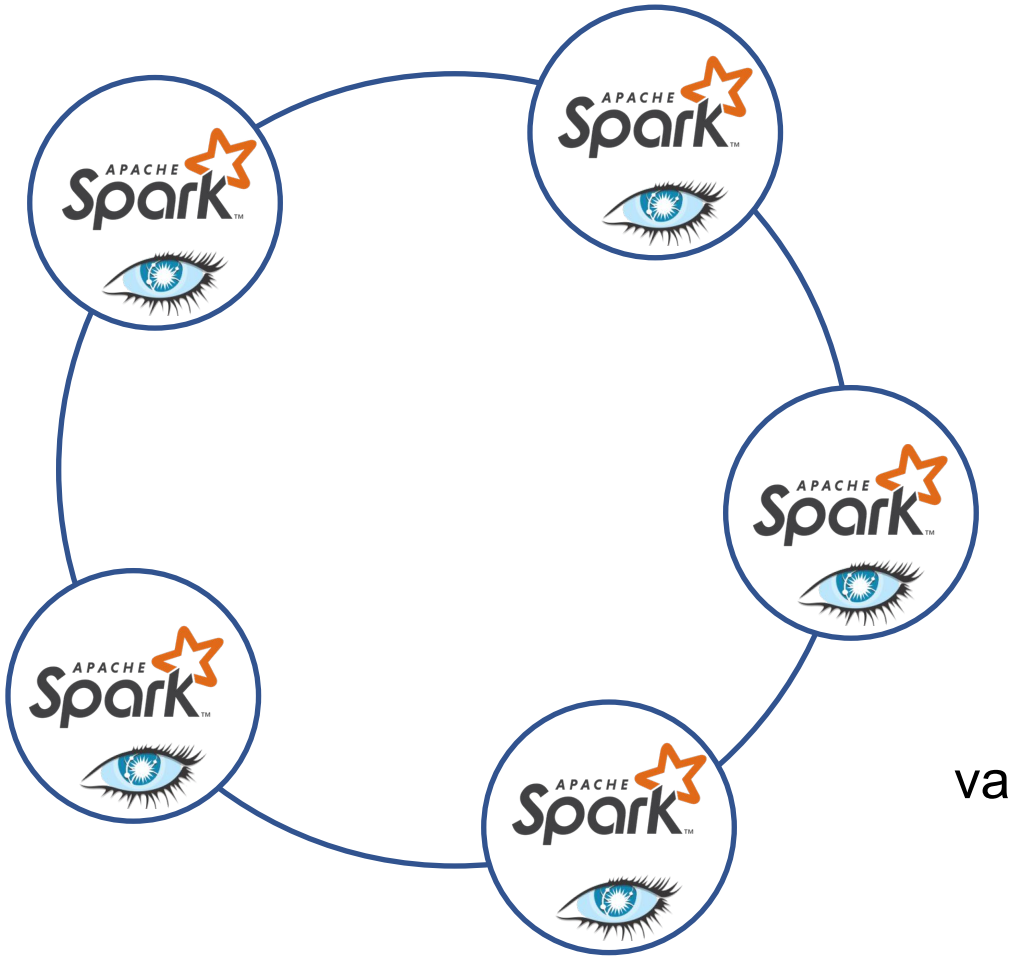
instaclustr

Spark is a Distributed Big Data Processing Framework



Spark + Cassandra

instaclustr



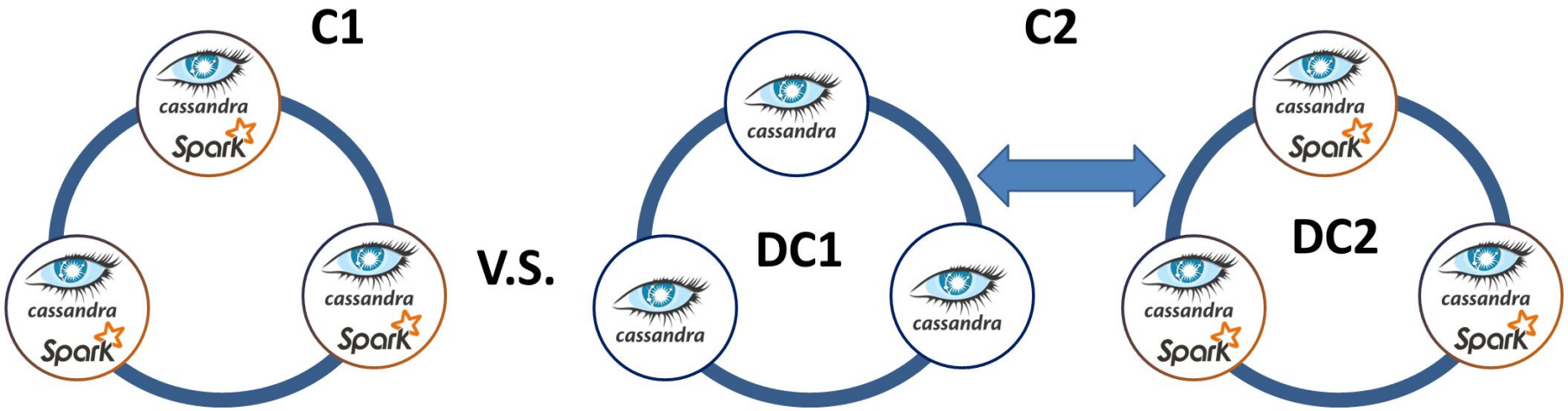
- Joins!
- Filtering!

Spark Cassandra connector

```
val rdd = sc.cassandraTable(“my_keyspace", “my_table")
```

Spark + Cassandra

instaclustr



instaclustr

Problem to solve….

Problem background

instaclustr

- How to efficiently monitor > 2000 servers all running Cassandra
  - Alerting
  - Metric history
  - Alert tuning
  - Graph / dashboard
  - Multi-tenant approach

- Off the shelf systems are available but:
  - Flexible enough?
  - Learn by using our technology
  - Optimizations opportunities.

Problem background

instaclustr

Friend: You know ~~drinking is bad for you~~ you should just use off the shelf

Me:

Implementation Approach

instaclustr

1. Collecting Metrics + Alert
2. Writing metrics
3. Rolling Up metrics
4. Presenting metrics

~ 9(!) months
(with quite a few detours
and distractions)

Solution Overview: instaclustr monitoring pipeline

2000 nodes * ~2,000 metrics /
20 secs = 140k metrics/sec

Managed Node (AWS) x many

Managed Node (Azure) x many

Managed Node (SoftLayer) x many

Managed Node (GCP) x many

RabbitMQ (x2)

Riemann (x3)

Console/ API (x2)

Cassandra + Spark (x27)

Admin Tools

PagerDuty

Monitoring

2000 nodes * ~2,000 metrics /
20 secs = 200k metrics/sec

Managed Node (AWS) x many

Managed Node (Azure) x many

Managed Node (SoftLayer) x many

Managed Node (GCP) x many

RabbitMQ (x2)

Riemann (x3)

PagerDuty

Console/ API (x2)

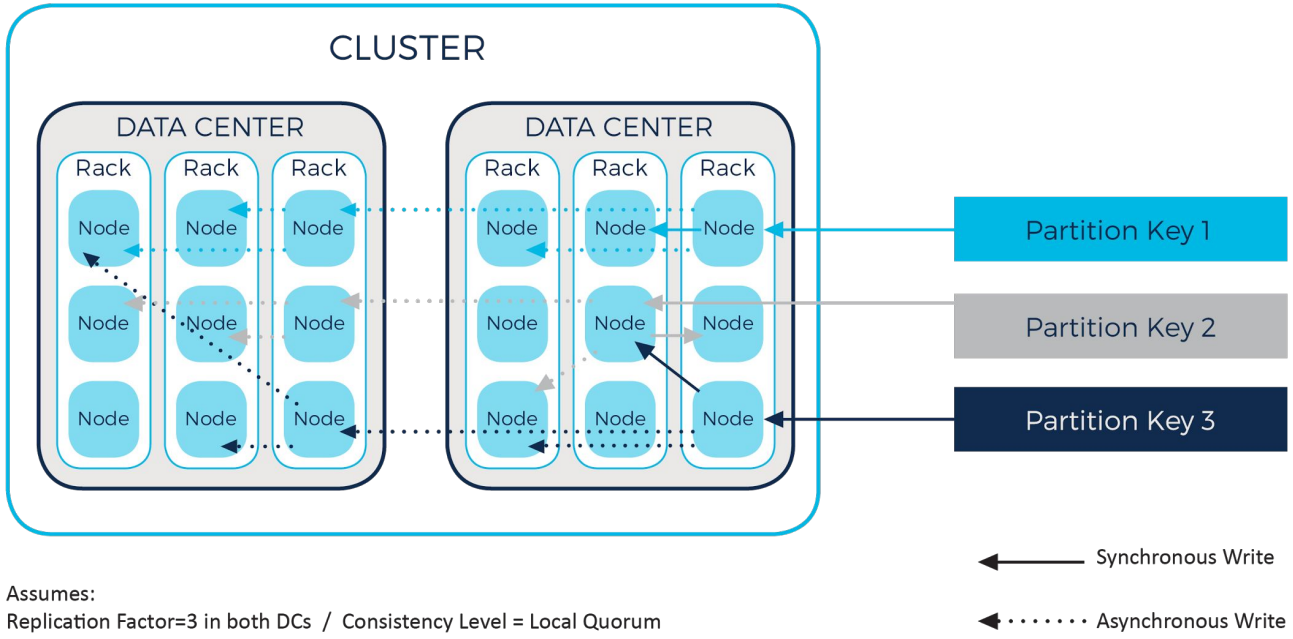
Cassandra + Spark (x15)

Admin Tools

Data model

instaclustr

```
CREATE TABLE instametrics.events_raw_5m (
    host text,
    bucket_time timestamp,
    service text,
    time timestamp,
    metric double,
    state text,
    PRIMARY KEY ((host, bucket_time, service), time)
)
```



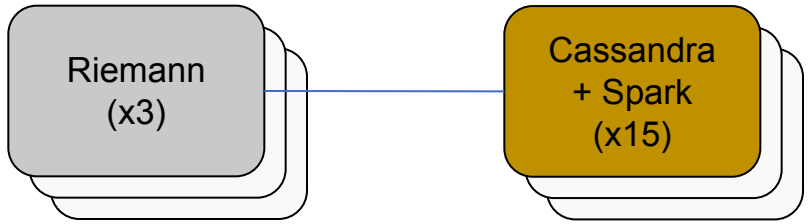
Data Model

```
CREATE TABLE instametrics.host (
    host text PRIMARY KEY
)


CREATE TABLE instametrics.service_per_host (
    host text,
    service text,
    PRIMARY KEY (host, service)
)
```

Writing metrics

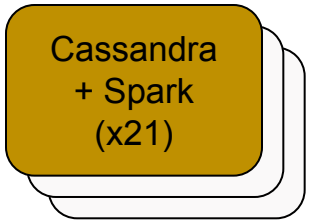
instaclustr



Key lessons:

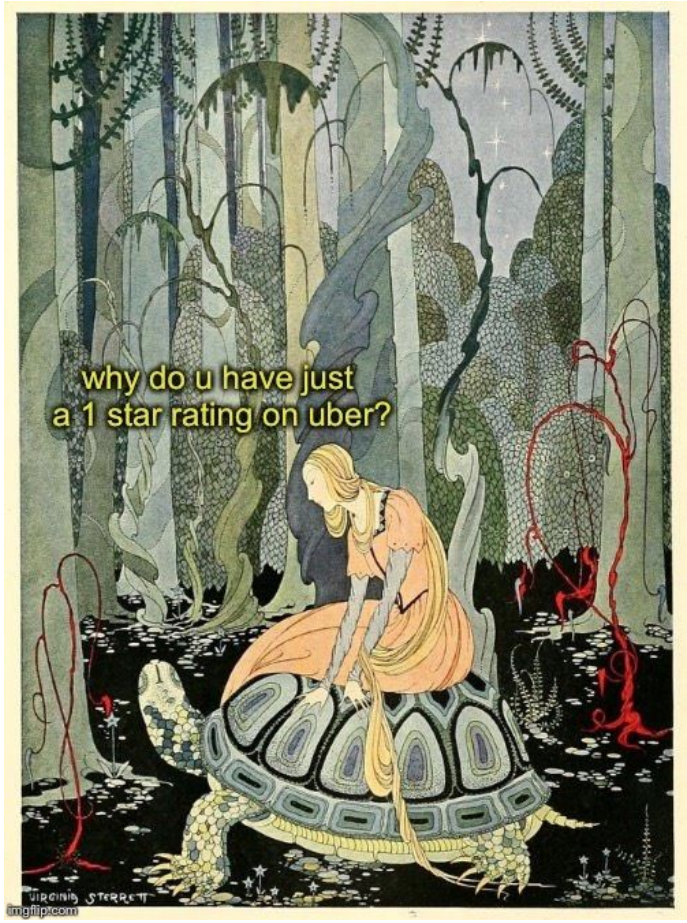
- Aligning Data Model with DTCS (now TWCS)
  - Initial design did not have time value in partition key
  - Settled on bucketing by 5 mins
    - Enables DTCS to work
    - Works really well for extracting data for roll-up
    - Adds complexity for retrieving data
- Batching of writes
  - Found batching of 200 rows per insert to provide optimal throughput and client load
- Controlling data volumes from column family metrics
  - Limited, rotating set of CFs per check-in
- Managing back pressure is important

Rolling Up metrics



instaclustr

- Developing functional solution was easy, getting to acceptable performance was hard (and time consuming) but seemed easy once we'd solved it

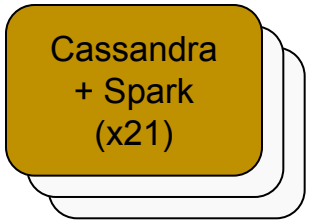


Data Model

instaclustr

```
CREATE TABLE instametrics_rollup.events_rollup_300 (
    bucket_time timestamp,
    host text,
    service text,
    time timestamp,
    avg double,
    max double,
    min double,
    state text,
    PRIMARY KEY ((bucket_time, host, service), time)
```

Rolling Up metrics

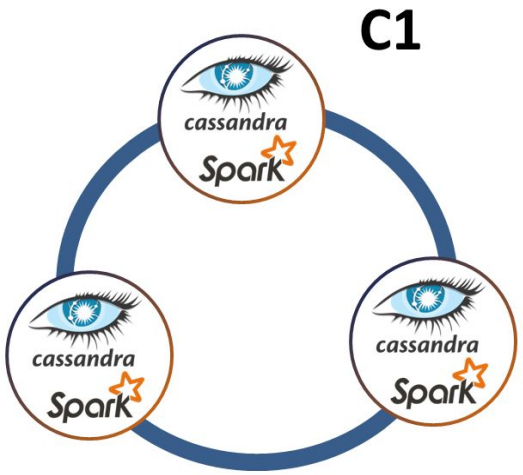


instaclustr

- Developing functional solution was easy, getting to acceptable performance was hard (and time consuming) but seemed easy once we'd solved it

- Keys to performance?
  - Align raw data partition bucketing with roll-up timeframe (5 mins)
  - Use repartitionByCassandraReplica to align Spark partitions with Cassandra partitions
  - Use joinWithCassandra table to extract the required data – 2-3x performance improvement over alternate approaches

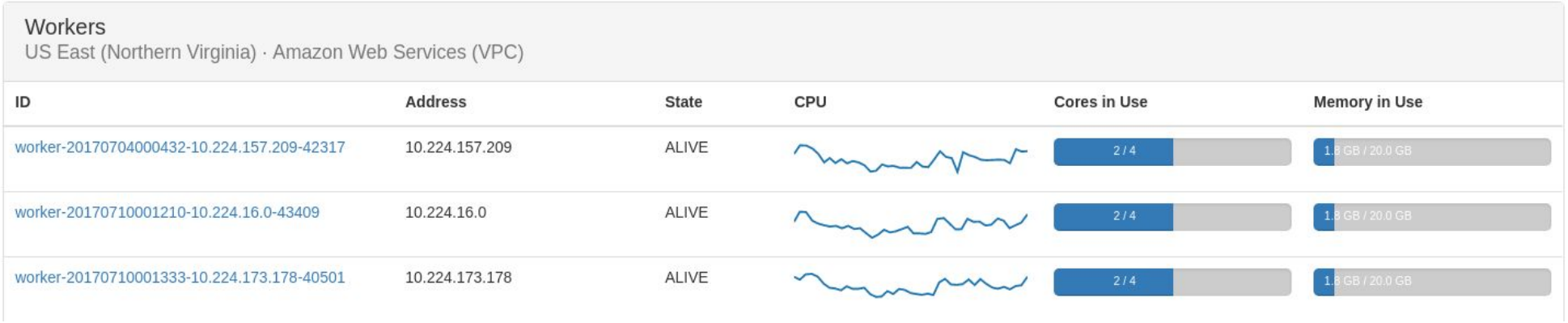
instaclustr



Read Tuning:
spark.cassandra.input.fetch.size_in_rows
spark.cassandra.input.reads_per_sec

Write Tuning:
spark.cassandra.output.throughput_mb_per_sec

5min – hourly – daily rollup



Presenting metrics

instaclustr

- Generally, just worked

- Main challenge was dealing with how to find latest data in rollup buckets when not all data is reported in each data set

Optimisation with Cassandra Aggregation

instaclustr

- Upgraded to Cassandra 3.7 and change code to use Cassandra aggregates:

```
 val RDDJoin = sc.cassandraTable[(String, String)]("instametrics" ,
"service_per_host")
     .filter(a => broadcastListEventAll.value.map(r =>
a._2.matches(r)).foldLeft(false)(_ || _))
     .map(a => (a._1, dateBucket, a._2))
     .repartitionByCassandraReplica("instametrics", "events_raw_5m", 100)
     .joinWithCassandraTable("instametrics", "events_raw_5m",
       SomeColumns("time", "state", FunctionCallRef("avg",
Seq(Right("metric")), Some("avg")), FunctionCallRef("max",
Seq(Right("metric")), Some("max")), FunctionCallRef("min",
Seq(Right("metric")), Some("min")))).cache()
```

- 50% reduction in roll-up job runtime (from 5-6 mins to 2.5-3mins) with reduced CPU usage

Rolling Up metrics

instaclustr

What's Next

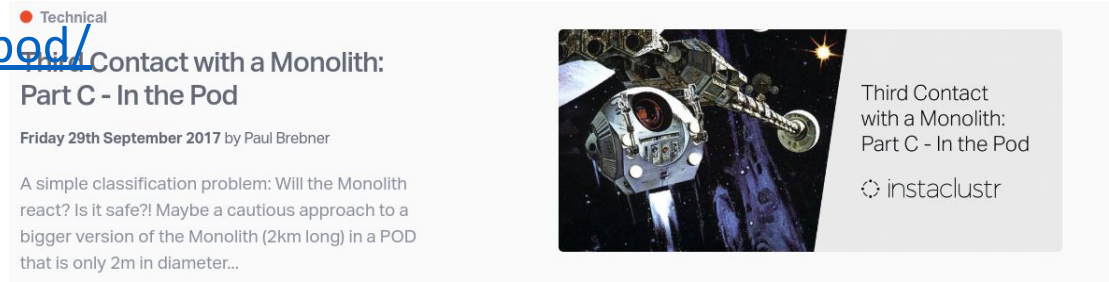
instaclustr

- Riemann straight to Spark Streaming
  - Spark Streaming for 5 min roll-ups rather than save and extract

- Scale-out by adding nodes is working as expected
- Continue to add additional metrics to roll-ups as we add functionality
- Plan to introduce more complex analytics & feed historic values back to Reimann for use in alerting

Further info:

- ✓ Scaling Riemann: https://www.instaclustr.com/blog/2016/05/03/post-500-nodes-high-availability-scalability-with-riemann/
- ✓ Riemann Intro: https://www.instaclustr.com/blog/2015/12/14/monitoring-cassandra-and-it-infrastructure-with-riemann/
- ✓ Instametrics Case Study: https://www.instaclustr.com/project/instametrics/
- ✓ Multi-DC Spark Benchmarks: https://www.instaclustr.com/blog/2016/04/21/multi-data-center-sparkcassandra-benchmark-round-2/
- ✓ Top Spark Cassandra Connector Tips: https://www.instaclustr.com/blog/2016/03/31/cassandra-connector-for-spark-5-tips-for-success/
- ✓ Cassandra 3.x upgrade: https://www.instaclustr.com/blog/2016/11/22/upgrading-instametrics-to-cassandra-3/
- ✓ Cassandra – Spark MLIB: https://www.instaclustr.com/third-contact-monolith-part-c-pod/



instaclustr

Ben Bromhead
CTO, Instaclustr
ben@instaclustr.com