instaclustr

Apache Cassandra

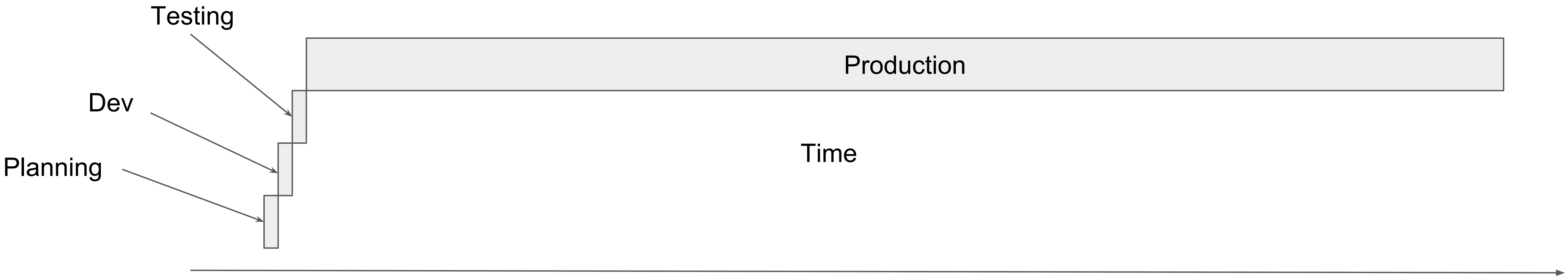
The first 6 months

Agenda - 6 months in production

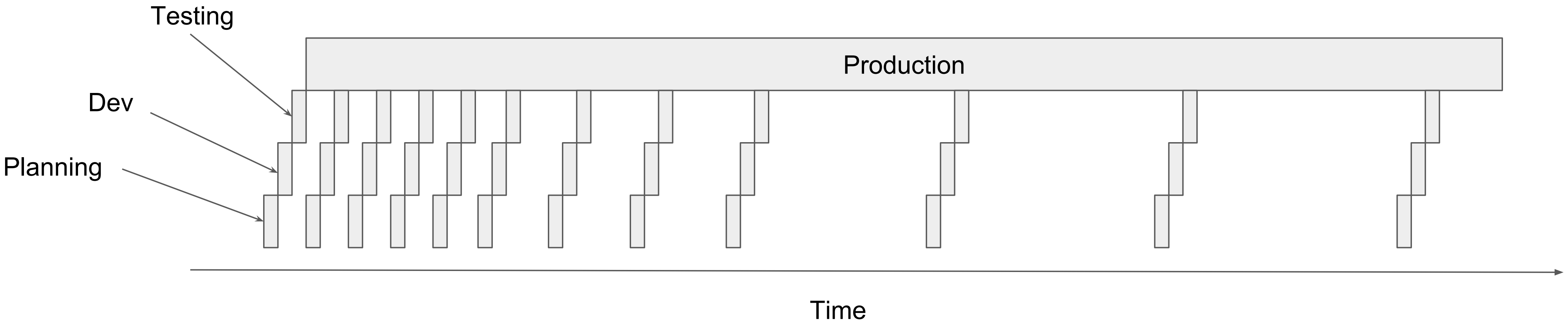
instaclustr

- Putting a new DB into production
- Introduction to Cassandra
- Design and Test
- Getting ready for production
- The first 6 months

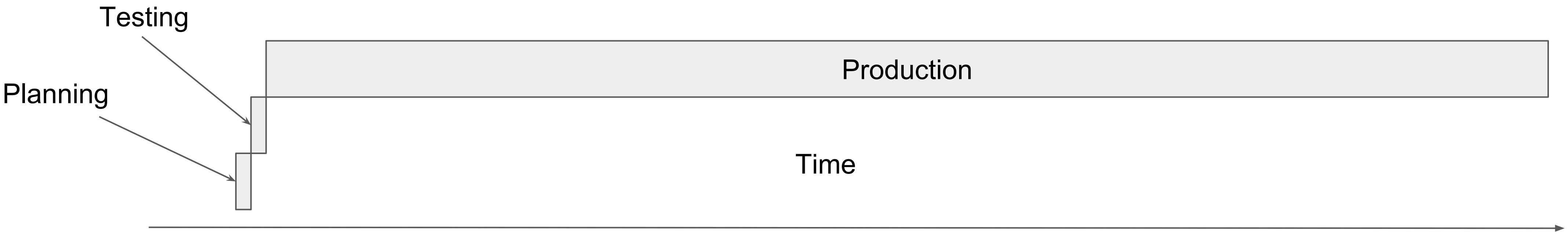
Planning for production - Apps



Planning for production - Apps

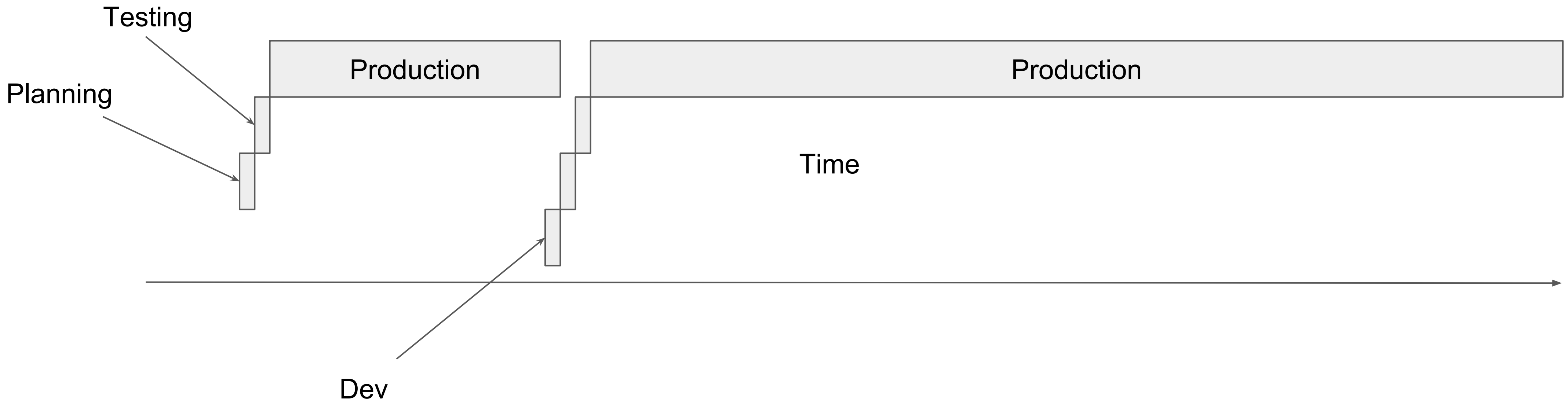


Planning for production - Databases



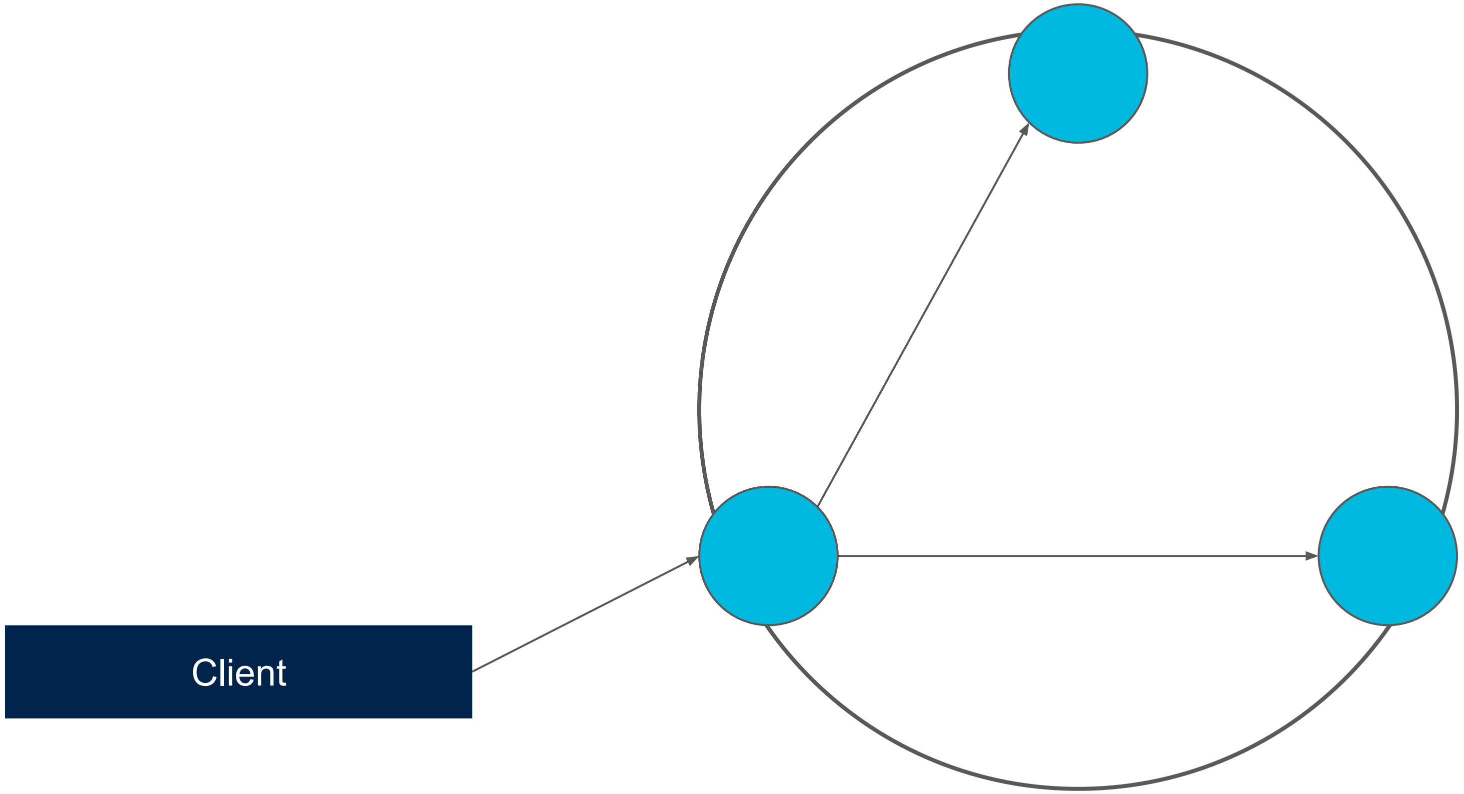
Planning for production - Databases

Planning for production - Databases



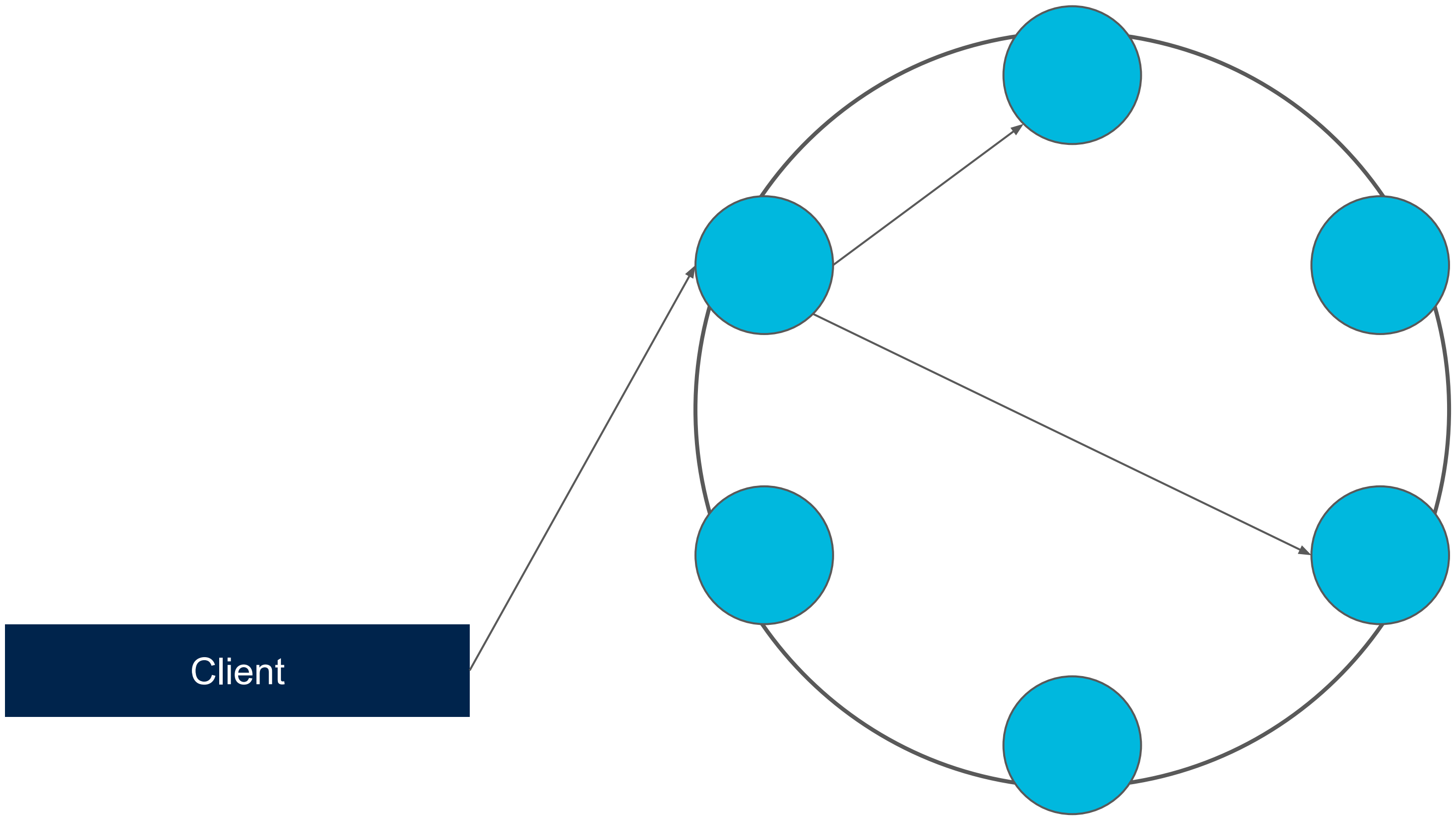
Quick introduction to Cassandra

instaclustr



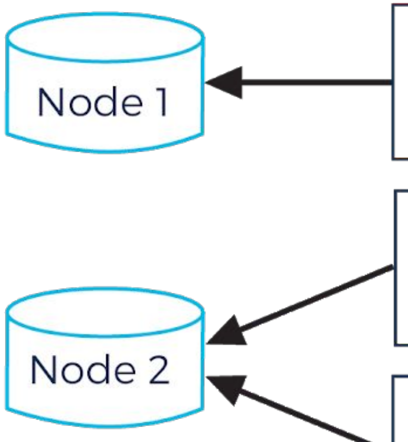
Quick introduction to Cassandra

instaclustr



Partitioning

instaclustr



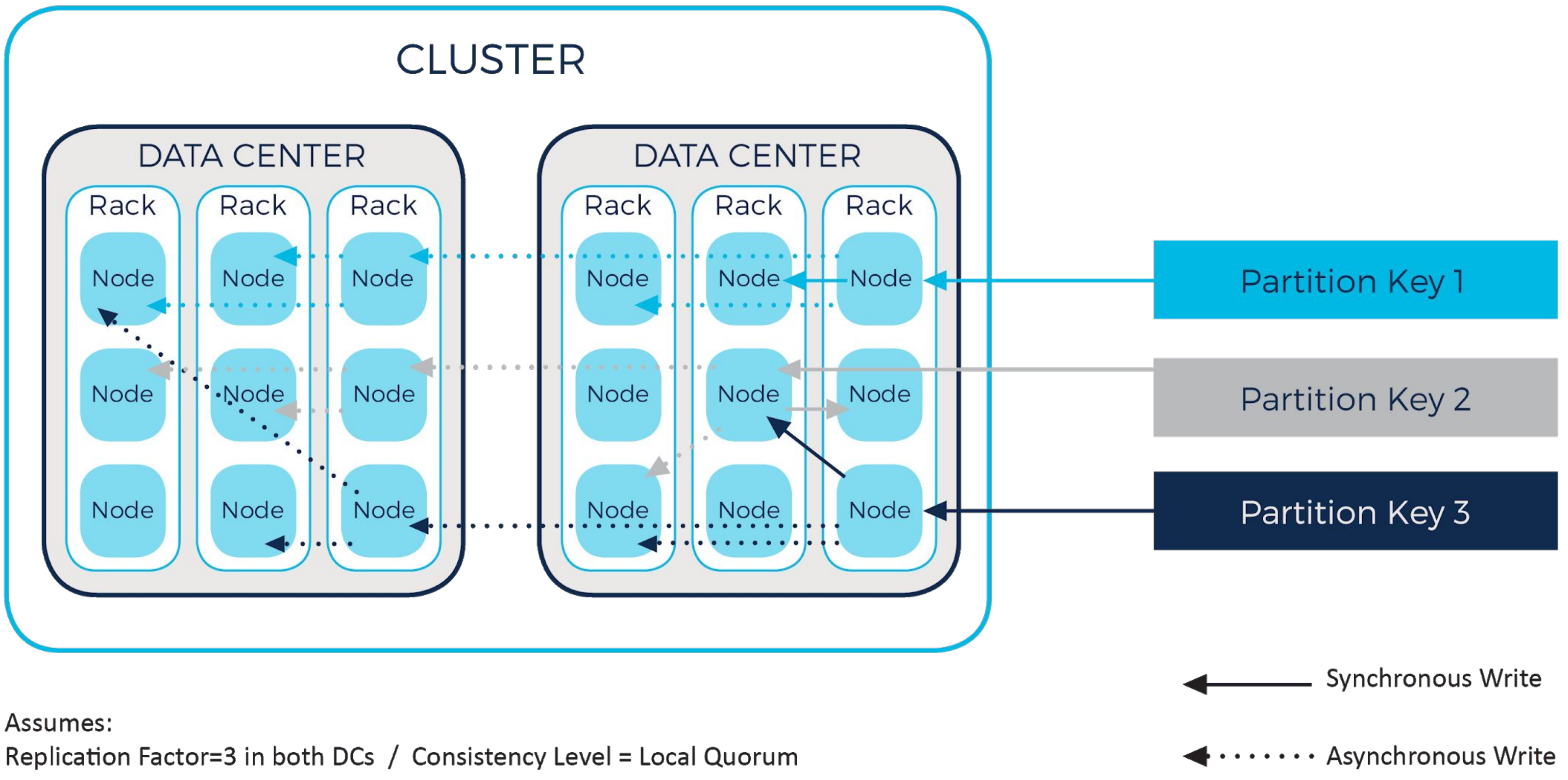
Sensor #	Date	Timestamp	Metric1	Metric2	Metric3
1	2015-01-01	20150101-000000	5.01	5.67	0.678
1	2015-01-01	20150101-000010	5.01	5.67	0.678
1	2015-01-01	20150101-000020	5.05	5.8	0.678
1	2015-01-02	20150101-000000	5.01	5.67	0.678
1	2015-01-02	20150101-000010	5.01	5.67	0.678
1	2015-01-02	20150101-000020	5.05	5.8	0.678
2	2015-01-02	20150101-000000	6.01	7.67	0.978
2	2015-01-02	20150101-000010	6.01	7.67	0.698
2	2015-01-02	20150101-000020	6.05	8.8	0.679

Partition Key   Clustering Key

Primary Key

PRIMARY KEY ((Sensor,Date),Timestamp)

Quick introduction to Cassandra



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

Eventual Consistency

- **Replication Factor (RF)** - defines how many copies (replicas) of a row should be stored in the cluster.
- **Consistency level (CL)** - How many acknowledgements/responses from replicas before a query is considered a success.

- Inconsistency means that not *all* replicas have a copy of the data, and this can happen for a few reasons:
  - Application uses a low consistency level for writes (eg LOCAL_ONE)
  - Nodes have dropped mutation messages under load
  - Nodes have been DOWN for longer than hinted handoff window (3 hours)

- Strong consistency (where R+W > RF) means the above won't impact you during outages.
- **Repairs** are how Cassandra fixes inconsistencies and ensures that *all* replicas hold a copy of the data.

Design Approach

- Phase 1: Understand the data
  - Define the data domain: E-R logical model
  - Define the required access patterns: how will you select an update data?
- Phase 2: Denormalize based on access patterns
  - Identify primary access entities: driven by the access keys
  - Allocate secondary entities: denormalize by pushing up or down to the primary entities
- Phase 3: Review & tune
  - Review partition keys and clusters
    - Do partition keys have sufficient cardinality?
    - Is the number of records in each partition bounded?
    - Does the design consider delete and update impact?
  - Test & tune: check updates, review compaction strategy

Testing Cassandra applications

- Long running tests with background load are vital
  - Can run extremely high write loads for an hour or two but might take days to catch up on compactions
  - Don’t forget repairs
- Make sure your data volumes on disk are representative as well as read/write ops - cache hit rates can make a big difference to performance
- Mirror production data demographics as closely as possible (eg partition size)
- Don’t forget to include update/delete workload if applicable
- For core cassandra features, can test on reduce size and rely on scale-up but beware:
  - Secondary indexes
  - MVs
  - LWTs

Monitoring Cassandra (Metrics + Alerting)

Metric	Description	Frequency
**Node Status	Nodes DOWN should be investigated immediately. `org.apache.cassandra.net:type=FailureDetector`	*Continuous, with alerting*
**Client read latency	Latency per read query over your threshold `org.apache.cassandra.metrics:type=ClientRequest,scope=Read`	*Continuous, with alerting*
**Client write latency	Latency per write query over your threshold `org.apache.cassandra.metrics:type=ClientRequest,scope=Write`	*Continuous, with alerting*
CF read latency	Local CF read latency per read, useful if some CF are particularly latency sensitive.	*Continuous if required*
Tombstones per read	A large number of tombstones per read indicates possible performance problems, and compactions not keeping up or may require tuning	*Weekly checks*
SSTables per read	High number (>5) indicates data is spread across too many SSTables	*Weekly checks*
**Pending compactions	Sustained pending compactions (>20) indicates compactions are not keeping up. This will have a performance impact. `org.apache.cassandra.metrics:type=Compaction,name=PendingTasks`	*Continuous, with alerting*

*Items marked ** give an overall indication of cluster performance and availability*

Security

- At a minimum
  - Enable password auth
  - Enable client->server encryption (particularly if using public IPs to connect)
  - Enable internode encryption
  - Don’t use the default Cassandra user
- Best practice
  - Encrypt sensitive data at the client
    - Works well with typical C* access patterns where PK values are hashed anyway
    - Dates are the most common case of range selects and typically are not sensitive if other identifying data is encrypted

Production - Day 0

Everything works!

Day 14 - AWS instance replacement

instaclustr

- No big deal
    - Terminate instance
    - Replace node, no impact on application, no failover, done during business hours

Replacing Nodes

- Replacing a dead node is similar to adding a new one, but add this line in the cassandra-env.sh *before* bootstrapping:

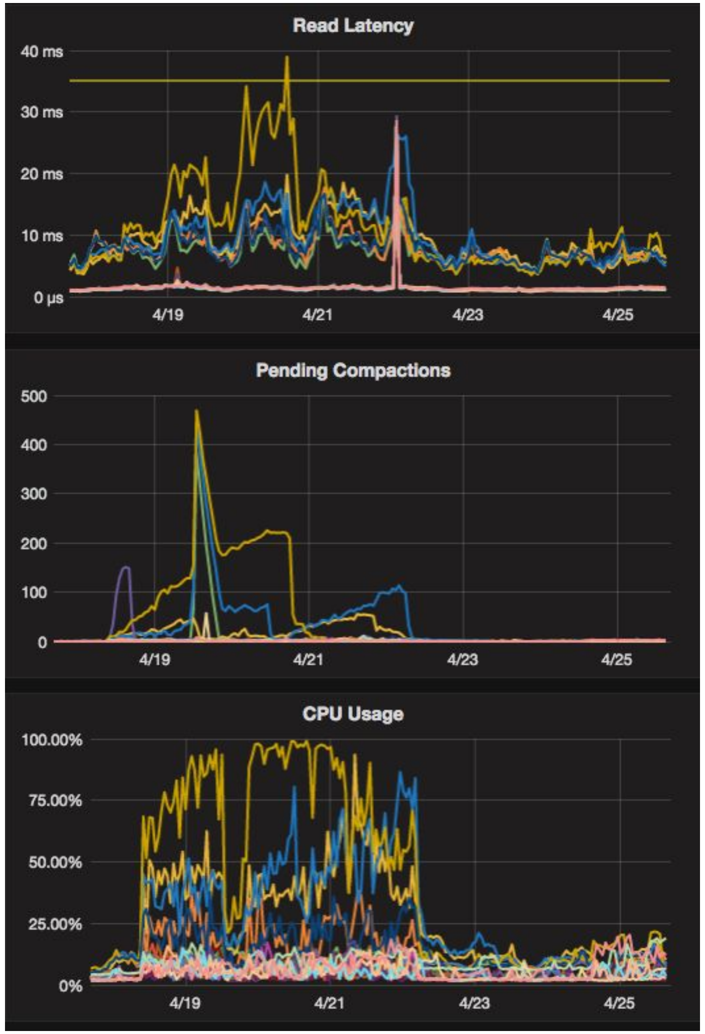
```
-Dcassandra.replace_address_first_boot=<dead_node_ip>
```

- This tells Cassandra to stream data from the other replicas.
  - *Note this can take quite a long time depending on data size*
  - *Monitor with nodetool netstats*

- If on >2.2.8 and replacing with a different IP address, the node will receive all the writes while joining.

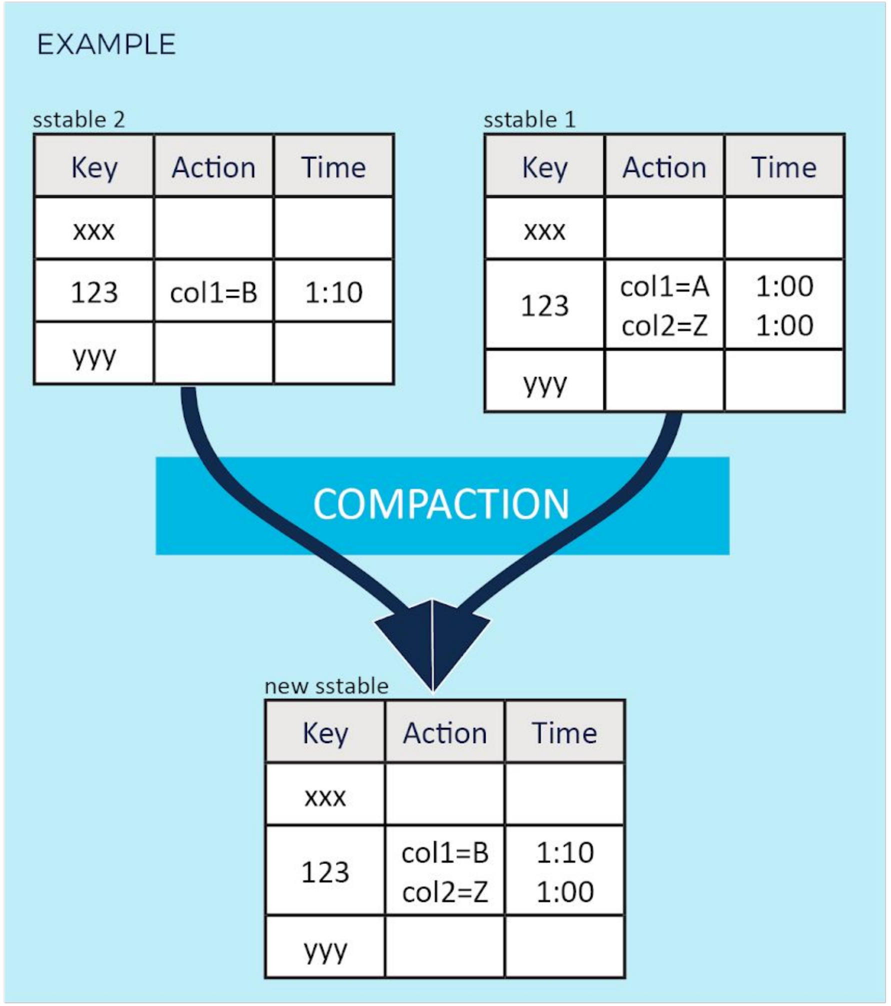
- Otherwise, you should run repair.
  - If the replacement process takes longer than `max_hint_window_in_ms` you **should** run repair to make the replaced node consistent again, since it missed ongoing writes during bootstrapping (streaming).

Day 15 - Compaction & Repairs



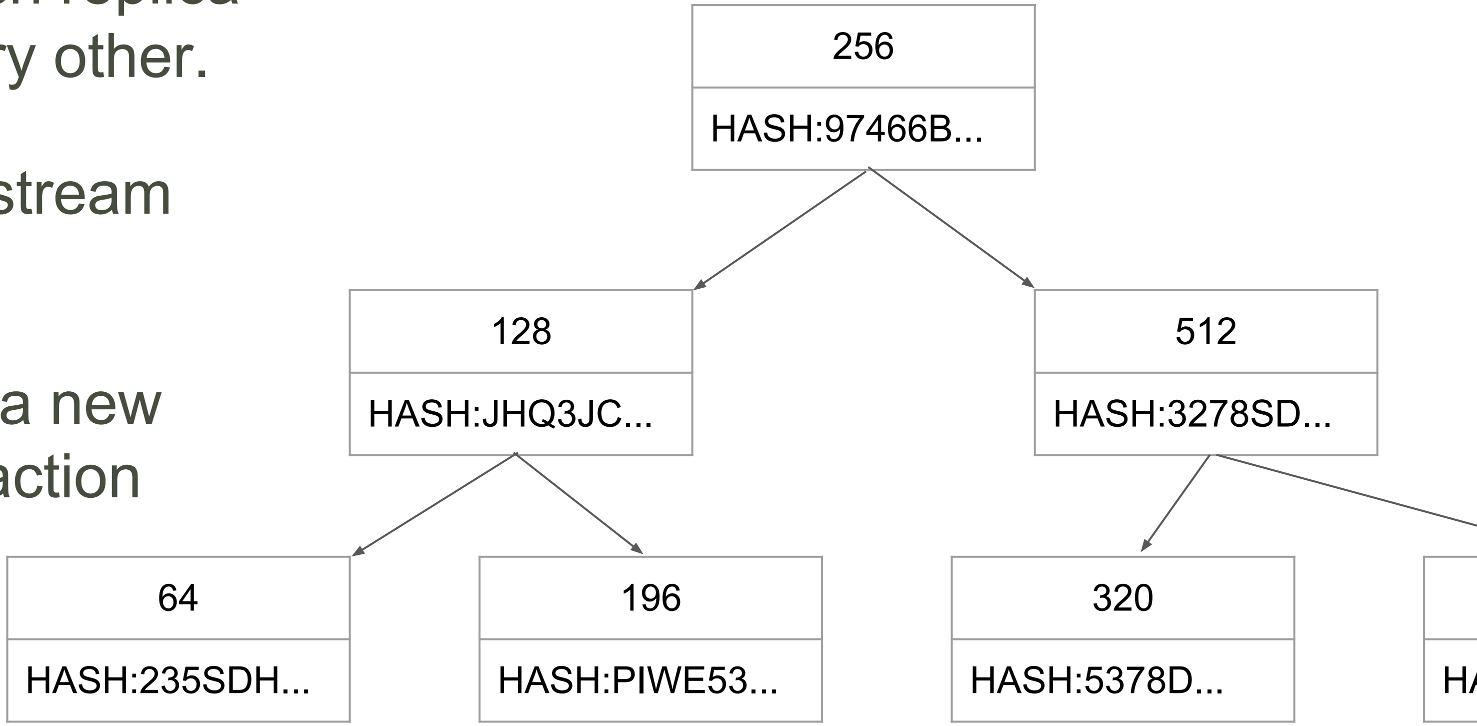
Compaction Intro

- Cassandra never updates sstable files once written to disk
- Instead all inserts and updates are essentially (logically) written as transaction logs that are reconstituted when read
- Compaction is the process of consolidating transaction logs to simplify reads
- It’s an ongoing background process in Cassandra
- Compaction ≠ Compression

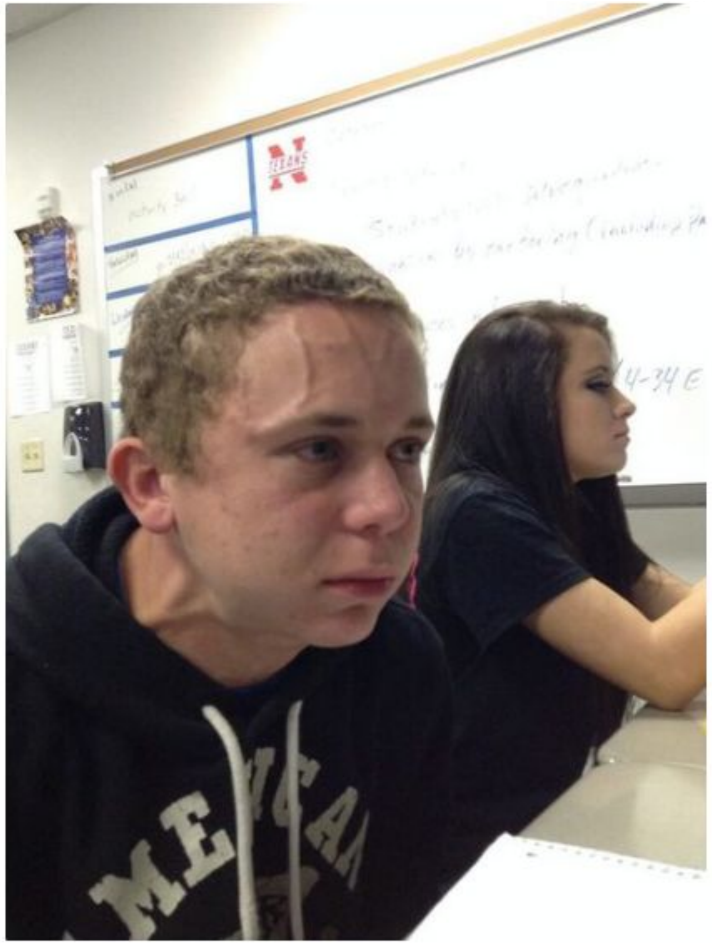


Repair Intro

- Reads every SSTable to be repaired
- Generates a merkle tree of data read.
- Send merkle tree to replicas, each replica compares each tree against every other.
- Any differences, Cassandra will stream the missing data
- Streamed data will be written as a new SSTable generating more compaction



Waiting for repair to finish



Compaction and Repair

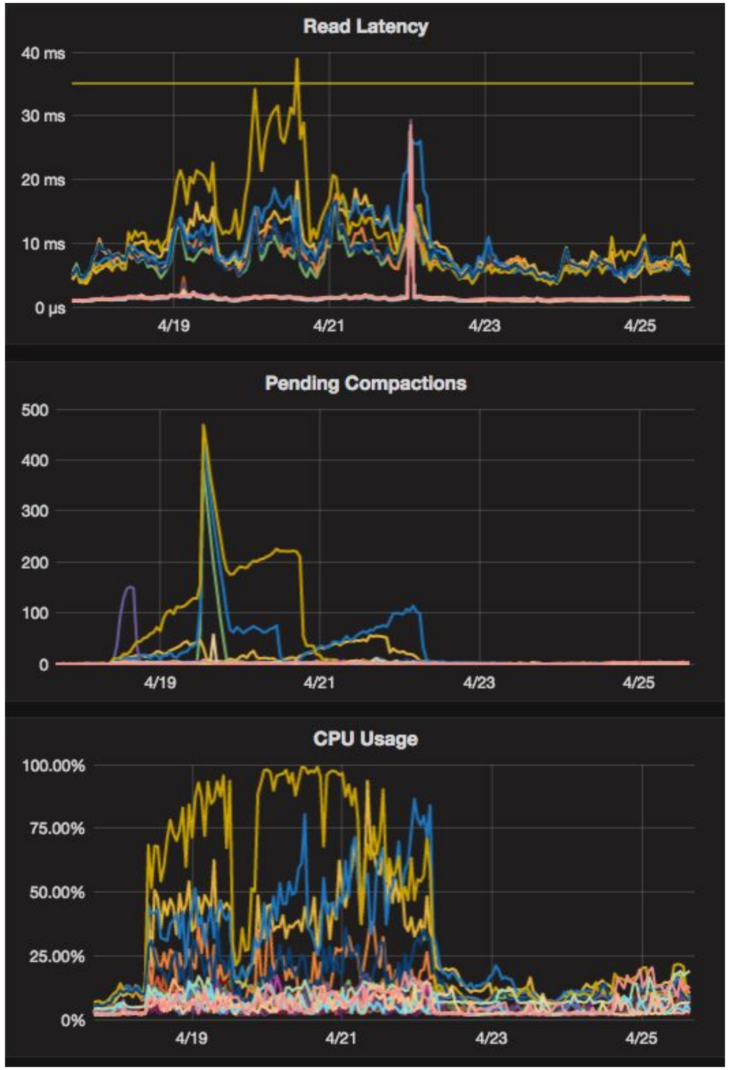
- Regular compactions are an integral part of any healthy Cassandra cluster.
- Repairs need to be run to ensure data consistency every gc_grace period.
- Can have a significant disk, memory (GC), cpu, IO overhead.
- Are often the cause of “unexplained” latency or IO issues in the cluster.
- Repair has a number of different strategies (sequential, parallel, incremental, sub range).
- Choose one that works best for you (likely to be either sub range or incremental).

Monitoring Compactions/Repair

- Monitor with nodetool compactionstats, tpstats & netstats

```
~ $ nodetool compactionstats -H
pending tasks: 518
   compaction type    keyspace    table    completed       total     unit    progress
        Compaction        data       cf     18.71 MB   111.16 MB    bytes      16.83%
Active compaction remaining time :    0h00m05s
```

- A single node doing compactions can cause latency issues **<u>across the whole cluster</u>**, as it will become slow to respond to queries.
- Repair can often the cause a large spike in compactions



Compaction - Other things to check

- STCS – Insert heavy and general workloads.
- LCS – Read heavy workloads, or more updates than inserts.
- DTCS/TWCS – Compact and arrange SSTables based on write time.

Table	Live Cells Per Read: Average	Live Cells Per Read: Max	SSTables Per Read: Average	SSTables Per Read: Max	Tombstones Per Read Average	Tombstones Per Read: Max
table_write_stats	67.6	286	2.000	5	0	0
spark_time	0.154	1	0.308	4	0	0
patterns_ten_min	0	0	0	0	0	0
pattern_details_chunks	0	0	0	0	0	0
pattern_details	0.938	1	0.908	1	0	0
olap_analytics_by_analytics_group_ten_min	0	0	0	0	0	0
network_states_ten_min	0	0	0	0	0	0
network_states_kpi_ten_min	0	0	0	0	0	0
metrics_by_uuid_one_min	28.585	5,002	9.623	156	0	2,882

instaclustr

Day 85 - Large Partitions and Tombstones

Partitioning: Diagnosing & Correcting

instaclustr

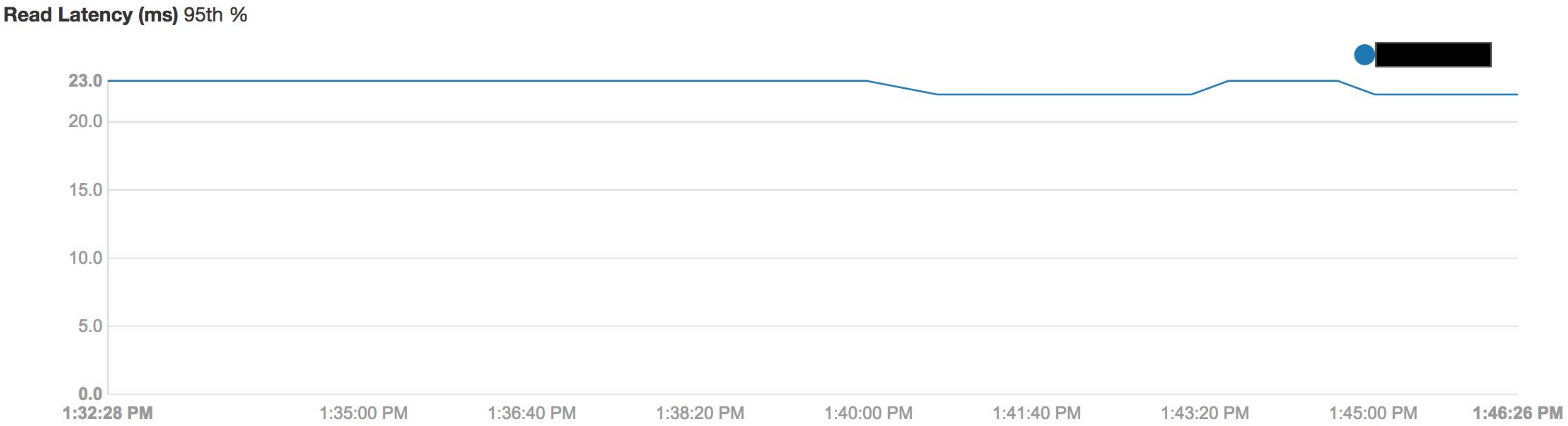
- Diagnosing
  - Overlarge partitions will also show up through long GC pauses and difficulty streaming data to new nodes
  - Many issues can be identified from data model review
  - nodetool cfstats / tablestats and cfhistograms provide partition size info. <10MB green, <100MB amber
  - Log file warnings - compacting large partition
- Correcting
  - Correcting generally requires data model change although depending on the application, application level change may be possible
  - ic-tools can help by providing info about partition keys of large partitions

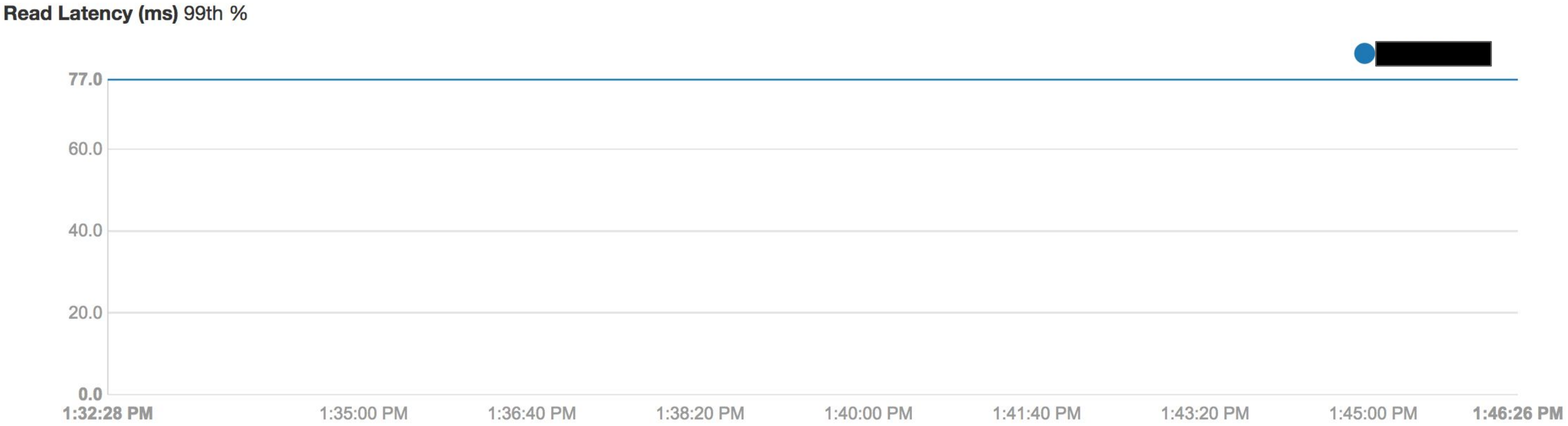
Examples of Large partitions

```
~ $ nodetool cfstats -H keyspace.columnfamily
  …
  Compacted partition minimum bytes: 125 bytes
  Compacted partition maximum bytes: 11.51 GB
  Compacted partition mean bytes: 844 bytes
$ nodetool cfhistograms keyspace columnfamily
```

Percentile	SSTables	Write Latency (micros)	Read Latency (micros)	Partition Size (bytes)	Cell Count
50%	1.00	14.00	124.00	372	2
75%	1.00	14.00	1916.00	372	2
95%	3.00	24.00	17084.00	1597	12
98%	4.00	35.00	17084.00	3311	24
99%	**5.00**	**50.00**	**20501.00**	**4768**	**42**
Min	0.00	4.00	51.00	125	0
Max	**5.00**	**446.00**	**20501.00**	**12359319162**	**129557750**

Otherwise known as chasing 9's





Tombstones

instaclustr

- When a row is deleted in C* it is marked with a tombstone (virtual delete). Tombstones remain in the sstables for at least 10 days by default.
- A high ratio of tombstones to live data can have significant negative performance impacts
- Be wary of tombstones when: deleting data, updating with nulls or updating collection data types.
- Diagnosing
  - nodetool cfstats/cfhistograms and log file warnings
  - slow read queries, sudden performance issues after a bulk delete
- Correcting
  - tune compaction strategy - LCS or TWCS can help in the right circumstances
  - reduce GC grace period & force compaction for emergencies
  - review data model/application design to reduce tombstones

Tombstones


Tombstones Per Read Max
[redacted]
702.0
600.0
400.0
200.0
0.0
11:00 AM
11:13 AM
11:30 AM
11:46 AM
12:03 PM
12:20 PM
12:36 PM
12:53 PM
1:10 PM
1:26 PM
1:45 PM

instaclustr

Day 100 - AWS outage

HOME › CONTENT TRACKS › COLO + CLOUD

Amazon Web Services US-East-1 goes down

28 February 2017 | By Sebastian Moss, Max Smolaks

Share 119 Tweet Like 0

AWS was out, therefore so was much of the Internet

AWS Region outage

instaclustr

No big deal

- You recently put in a second region
- DNS shifted end clients to second region
- Nothing to do on the Cassandra side
- Sit back with a beer and read HN while folks complain about it

Day 172 - Expansion

Cluster Changes

Ensure the cluster is 100% healthy and stable before making ANY changes.

Adding Nodes

instaclustr

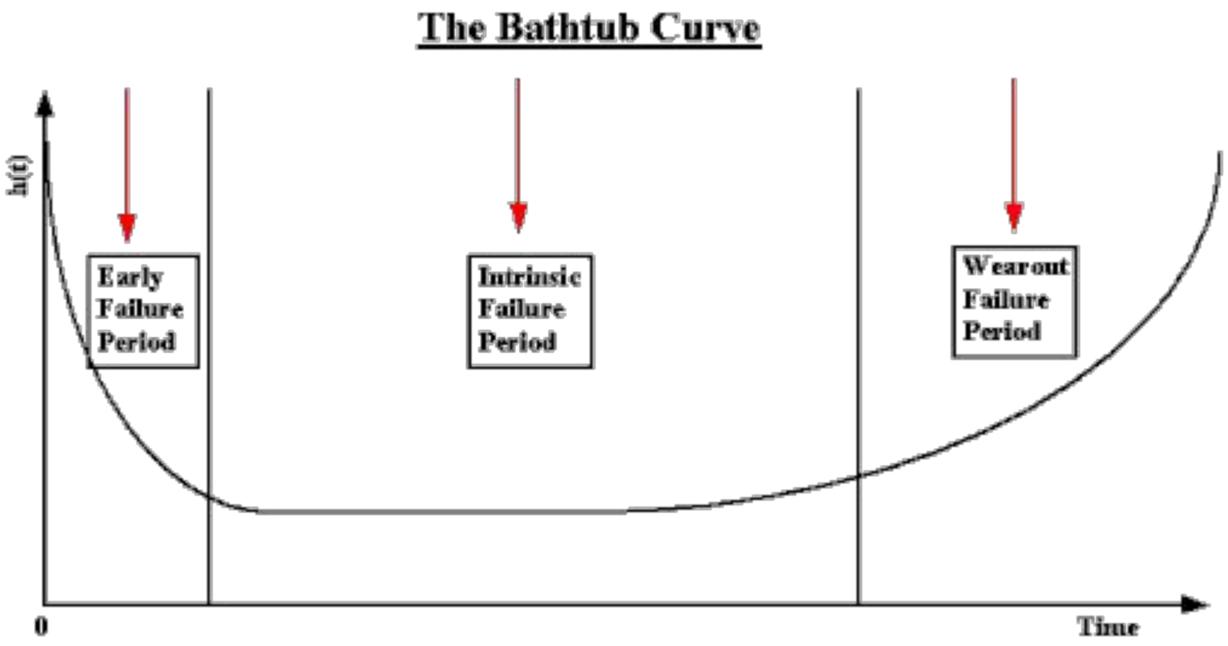
- **How do you know when to add nodes?**
  - When disks are becoming >70% full.
  - When CPU/OS load is consistently high during peak times.

- **Tips for adding new nodes:**
  - If using logical racks, add one node to every rack (keep distribution even)
  - Add one node at a time.
  - During the joining process, the new node will stream data from the existing node.
  - A joining node will accept writes but not reads.
  - Unthrottle compactions on the JOINING node “nodetool setcompactionthroughput 0”
    - *But throttle again once node is joined.*
  - Monitor joining status with “nodetool netstats”
  - After the node has streamed and joined it will have a backlog of compactions to get through.
  - Versions <2.2.x Cassandra will lose level info (LCS) during streaming and have to recompact *all* sstables again.

You are now a decorated veteran

instaclustr

There is still a lot to learn, but you will have a solid foundation to build on.



instaclustr

Ben Bromhead
CTO and Co-Founder
ben@instaclustr.com