

# The Genealogy of Troy

Client-side Cassandra in Java  
with Hector & Astyanax

Joe McTee

# About Me

- ◆ Principal Engineer at Tendril Networks
- ◆ Experience with embedded programming, server-side Java (haven't done much web development)
- ◆ JEKLsoft is my sandbox for experiment and play
- ◆ Contact me at
  - ◆ [mcjoe@jeklsoft.com](mailto:mcjoe@jeklsoft.com) / [jmctee@tendrilinc.com](mailto:jmctee@tendrilinc.com)
  - ◆ @jmctee on twitter
  - ◆ <https://github.com/jmctee> for teh codez

# About Tendril

- ◆ We develop products that bring consumers, utilities, and consumer product manufacturers together in a partnership to save energy while maintaining quality of life
- ◆ And we're hiring!

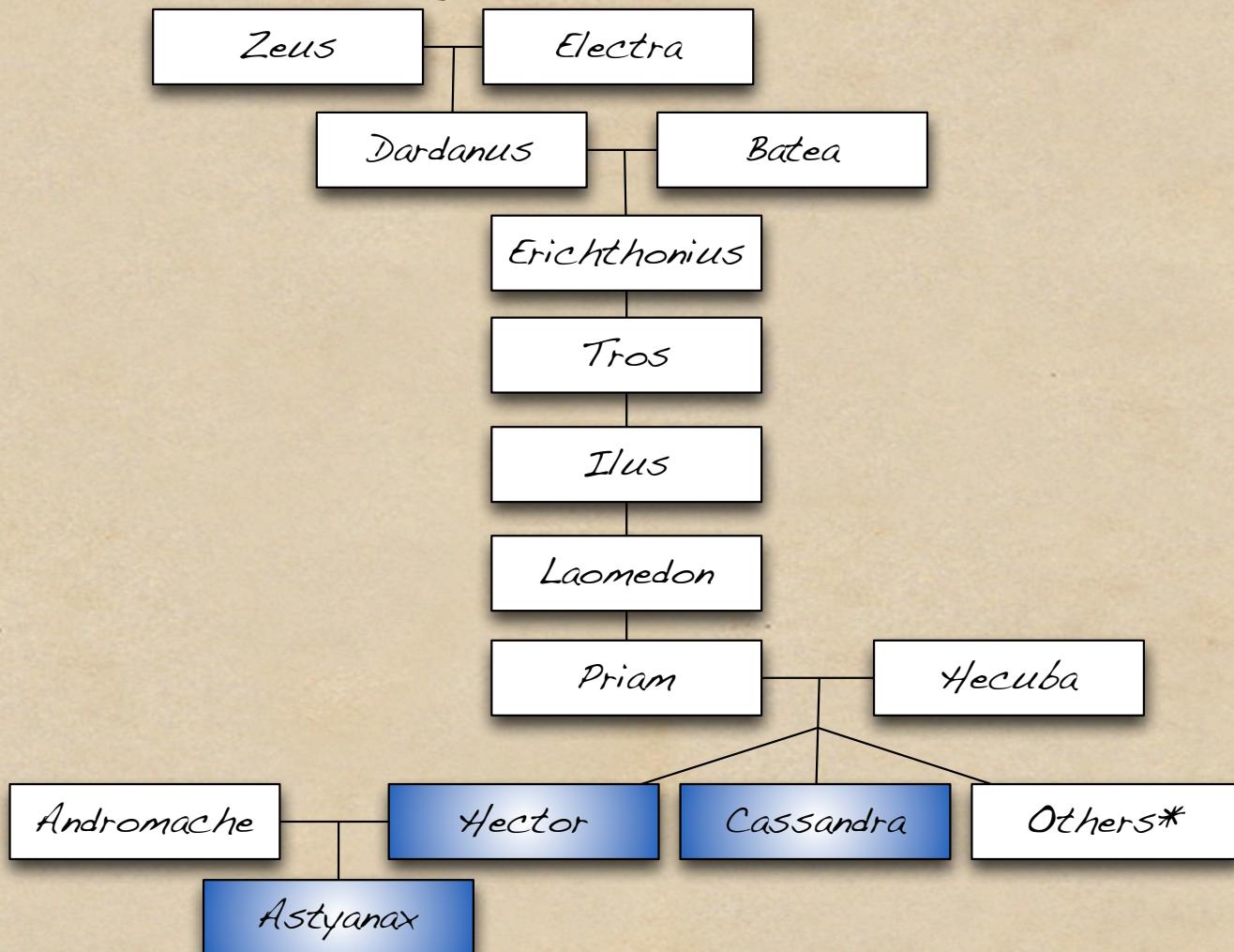
Hello Cassandra. I love you.  
You persist me.  
Eventually.

TENDRIL™  
The Power is Yours

HIRING TOP SOFTWARE ENGINEERS

tendriljobs.com

# The Family Tree



\* Suggestions for aspiring library writers... Paris, Helenus, Ilione, Deiphobus, Troilus, Polites, Creusa, Laodice, Polyxena, Polydorus

# The “Contrived” Problem

- ◆ We want to collect atmospheric readings in multiple cities
- ◆ In each city, we deploy a large number of sensors
- ◆ Sensors transmit a reading every 15 minutes via a web service call
  - ◆ Web service back-end stores the data
- ◆ Data will be queried by sensor and range of reading times
- ◆ Each sensor “reading” is a combination of multiple measurements
- ◆ The types for each reading form the contrivance...
  - ◆ Strings are boring and don’t expose design issues
  - ◆ Chosen types probably not optimal for readings

# A Reading

- ◆ ID of sensor that took reading (UUID)
- ◆ Time of reading (DateTime)
- ◆ Air Temperature in Celsius (BigDecimal)
- ◆ Wind Speed in kph (Integer)
- ◆ Wind Direction as compass abbreviation (String)
- ◆ Relative Humidity as percent (BigInteger)
- ◆ Bad Air Quality Detected flag (Boolean)

Where are we going to store  
the data?

Cassandra!

# Why Cassandra?

- ◆ Optimized for fast writes
  - ◆ We need that because we have “lots” of sensors
- ◆ Highly scalable
  - ◆ Add a node, scale increases
- ◆ Tunable to provide high availability and/or high consistency
  - ◆ There are tradeoffs here, but key is we control it

# Cassandra in 1 Slide

- ◆ Node - A Cassandra instance responsible for storing a pre-defined range of information (row keys)
- ◆ Cluster - group of Cassandra nodes organized in a ring, such that all possible row key ranges are covered by one "or more" nodes
- ◆ Keyspace - A grouping of column (or super column) families
- ◆ Column Family - An ordered collection of Rows that contain columns
- ◆ Super Column Family - An ordered collection of Rows that contain super columns
- ◆ Row - Key to access a given set of orderable (by name, not value!) columns or super columns
- ◆ Column - A name/value pair
  - ◆ +meta data: timestamp and time-to-live (TTL)
- ◆ Super Column - A name/column pair

# OK, maybe not...

- ◆ Columns are multi-dimensional maps
  - ◆ [keyspace] [column family] [row] [column]
- ◆ And super columns add one more dimension
  - ◆ [keyspace] [column family] [row] [super column]  
[column]
- ◆ All columns (or super columns) for a given row key are stored on the same node
- ◆ Name/Value store
  - ◆ Not an RDMS
  - ◆ No joins, denormalization is your friend.

**Cluster:**

**Keyspace:**

**Column Family:**

**Row:**

    Column: name/value/timestamp/ttl

...

    Column: name/value/timestamp/ttl

...

**Row:**

    Column: name/value/timestamp/ttl

...

    Column: name/value/timestamp/ttl

**Super Column Family:**

**Row:**

**Super Column:**

        Column: name/value/timestamp/ttl

...

        Column: name/value/timestamp/ttl

...

**Super Column:**

        Column: name/value/timestamp/ttl

...

        Column: name/value/timestamp/ttl

...

**Row:**

**Super Column:**

        Column: name/value/timestamp/ttl

...

        Column: name/value/timestamp/ttl

...

**Super Column:**

        Column: name/value/timestamp/ttl

...

        Column: name/value/timestamp/ttl

# Compact JSONish Schema

- ◆ This notation is derived from WTF-is-a-SuperColumn (see references)
  - ◆ Ditch the timestamp & TTL from Column
  - ◆ Pull the Columns' and SuperColumns' name component out so that it looks like a key/value pair.

# Example JSONish

```
Info: {  
    Contacts: {  
        People: {  
            McTee_Joe: {  
                street: "Tendril Plaza",  
                city: "Boulder",  
                zip: "80301",  
                phone: "555.555.5555"  
            }  
        }  
    }  
}
```

# Download the code

- ◆ Public project, please fork/contribute/provide feedback!
- ◆ Pre-download requirements
  - ◆ Git
  - ◆ Maven 3
- ◆ In directory where you want project
  - ◆ `git clone git@github.com:jmctee/Cassandra-Client-Tutorial.git`

# Run with Maven

- ◆ Does not require Cassandra to be installed on machine
  - ◆ Uses Embedded Cassandra
- ◆ Note: POM specifies forked tests. This is required!
  - ◆ <forkMode>perTest</forkMode>
- ◆ mvn clean test
  - ◆ “Tests run: 70, Failures: 0, Errors: 0, Skipped: 0”
  - ◆ Ready to experiment

# Installing Cassandra

- ◆ Download tarball from <http://cassandra.apache.org/download/>
  - ◆ e.g., apache-cassandra-1.0.8-bin.tar.gz
  - ◆ I like to put these in /opt, then
    - ◆ sudo ln -s /opt/apache-cassandra-1.0.8 /opt/cassandra
    - ◆ Add /opt/cassandra/bin to your path
    - ◆ Upgrading Cassandra is simply a new sym-link
      - ◆ ...and editing config described in next step

# Configuring Cassandra

- ◆ This is for development use, not a production configuration!
- ◆ Assume \$HOME is absolute path below, e.g., /Users/joemctee
  - ◆ Relative paths, e.g., ~, or env vars, e.g. \$HOME don't work in config files
- ◆ Make some directories
  - ◆ \$HOME/cassandra/
  - ◆ \$HOME/cassandra/data
  - ◆ \$HOME/cassandra/commitlog
  - ◆ \$HOME/cassandra/saved\_caches
  - ◆ \$HOME/cassandra/log
- ◆ Edit /opt/cassandra/conf/cassandra.yaml
  - ◆ Globally replace /var/lib with \$HOME
- ◆ Edit /opt/cassandra/conf/log4j-server.properties
  - ◆ Globally replace /var/log/cassandra with \$HOME/cassandra/log

# Starting Cassandra

- ◆ If /opt/cassandra/bin is in your path, from a terminal
  - ◆ cassandra -f
    - ◆ -f option keeps process in foreground, easier to kill that way
  - ◆ ctrl-c to stop
    - ◆ Without -f option, need to find and kill process

# Using the CLI

- ◆ If /opt/cassandra/bin is in your path, from a terminal
  - ◆ cassandra-cli
- ◆ help; for help
- ◆ exit; to quit
- ◆ Example session (super column):

```
connect localhost/9160;
drop keyspace Climate;
create keyspace Climate;
use Climate;
create column family BoulderSensors with column_type = 'Super';
list BoulderSensors;
```

- ◆ Example session (super column):

```
connect localhost/9160;
drop keyspace Climate;
create keyspace Climate;
use Climate;
create column family BoulderSensors;
list BoulderSensors;
```

# Embedded Cassandra

- ◆ com.jeklsoft.cassandraclient.EmbeddedCassandra
  - ◆ wraps org.apache.cassandra.service.EmbeddedCassandraService
  - ◆ No dependencies on Hector or Astyanax
  - ◆ Great tool for unit testing after things are working
  - ◆ No CLI, so not as great for debugging
  - ◆ Uses builder pattern for easy construction
    - ◆ Set and forget
  - ◆ Uses port 9161, so can run embedded and stand-alone instances simultaneously
    - ◆ Note: These are separate instances, not data sharing

# Starting Embedded Cassandra

```
public class TestEmbeddedCassandra {

    private static final String embeddedCassandraHostname = "localhost";
    private static final Integer embeddedCassandraPort = 9161;
    private static final String embeddedCassandraKeySpaceName = "TestKeyspaceName";
    private static final String columnFamilyName = "TestColumnName";
    private static final String configurationPath = "target/cassandra";

    @Test
    public void sunnyDayTest() throws Exception {
        List<String> cassandraCommands = new ArrayList<String>();
        cassandraCommands.add("create keyspace " + embeddedCassandraKeySpaceName + ";");
        cassandraCommands.add("use " + embeddedCassandraKeySpaceName + ";");
        cassandraCommands.add("create column family " + columnFamilyName);

        URL cassandraYamlUrl = TestEmbeddedCassandra.class.getClassLoader()
            .getResource("cassandra.yaml");
        File cassandraYamlFile = new File(cassandraYamlUrl.toURI());

        EmbeddedCassandra embeddedCassandra = EmbeddedCassandra.builder()
            .withCleanDataStore()
            .withStartupCommands(cassandraCommands)
            .withHostname(embeddedCassandraHostname)
            .withHostport(embeddedCassandraPort)
            .withCassandraConfigurationDirectoryPath(configurationPath)
            .withCassandraYamlFile(cassandraYamlFile)
            .build();

        assertNotNull(embeddedCassandra);
    }
}
```

# Last building block, Serializers

- ◆ Everything is stored as a byte array in Cassandra
  - ◆ Serializers convert objects to / from ByteBuffer
  - ◆ `public ByteBuffer toByteBuffer(T obj);`
  - ◆ `public T fromByteBuffer(ByteBuffer byteBuffer);`
  - ◆ Hector and Astyanax provide serializers for most basic types
  - ◆ We need to provide serializers for non-standard types: `DateTime`, `BigDecimal`, `Reading`

Let's Model a Reading

# Keyspace & Column Family

- ◆ Keyspace: Climate
- ◆ Column Family: BoulderSensors
  - ◆ One Column Family per City

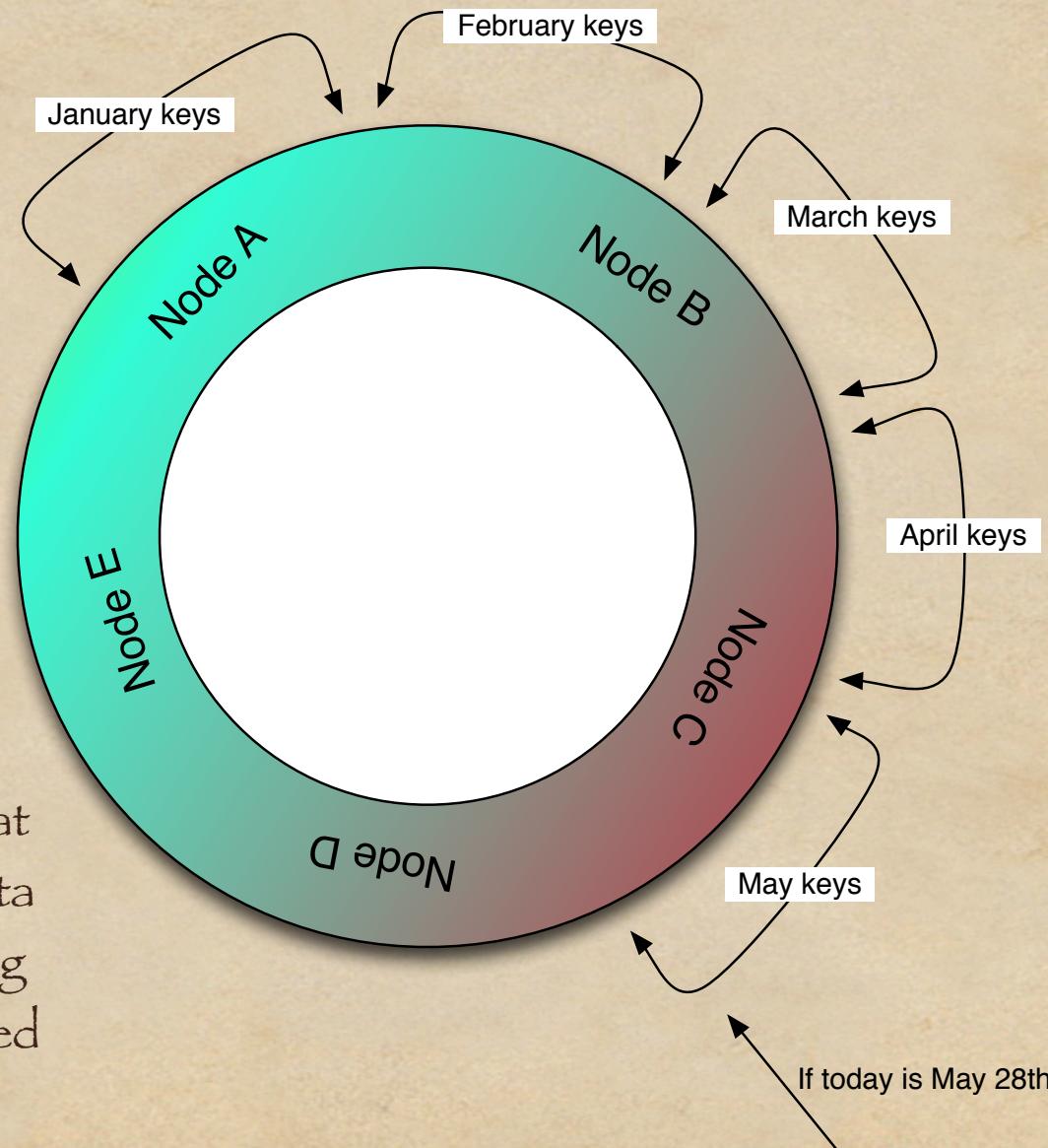
# Selection of Row Key

- ◆ Need to query by Sensor ID and TimeStamp range
  - ◆ So two possible row key candidates

# If We Choose TimeStamp

- ◆ We need to query on sequential ranges of TimeStamps
  - ◆ This implies we must use Order-Preserving Partitioning (OP) for the cluster
  - ◆ Cassandra requires OP for range queries of row keys (performance optimization)
  - ◆ No such requirement for range queries of column names (because all columns are co-located on one node)
- ◆ Column key would be Sensor ID

# Hot Spots!

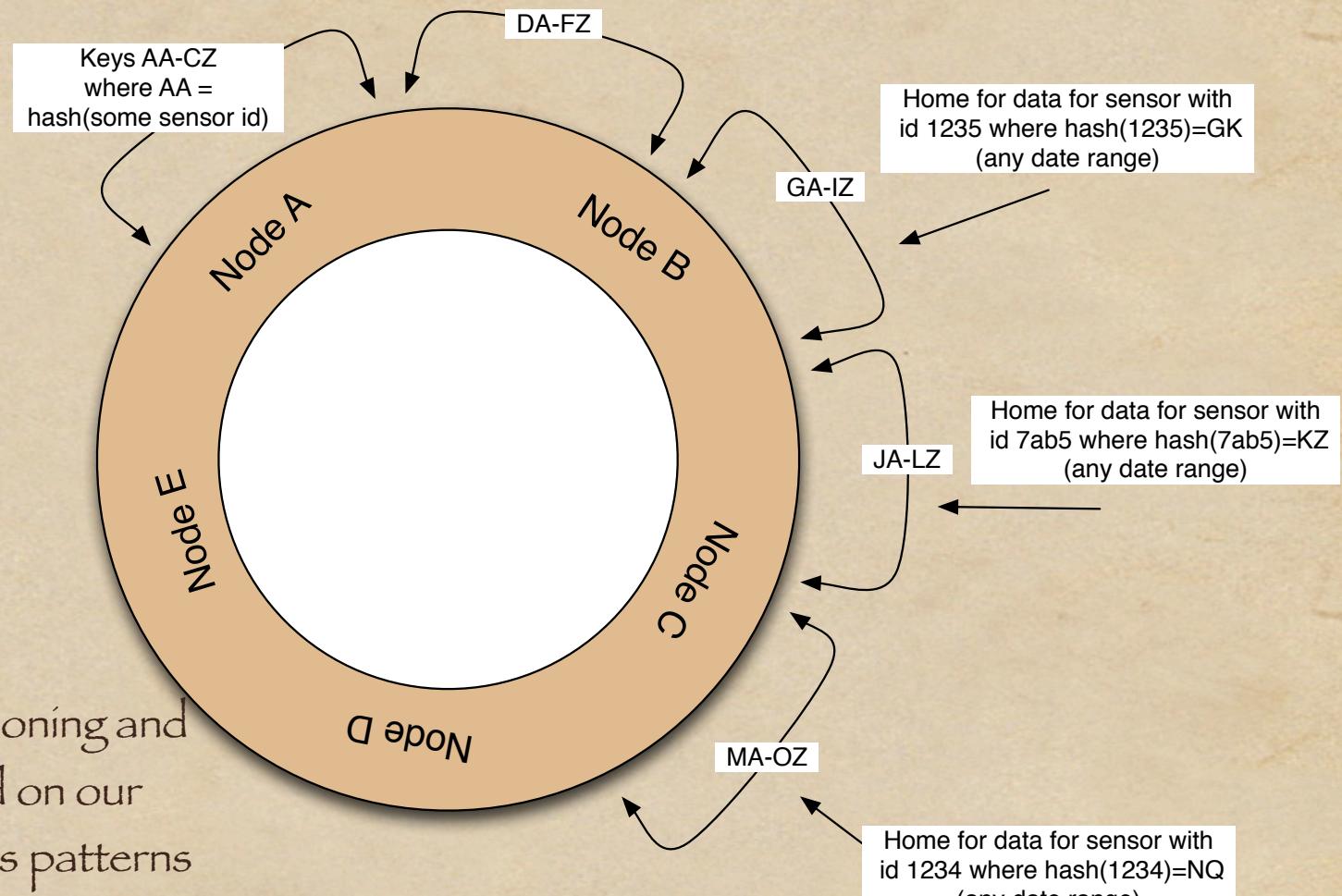


Theoretical partitioning and heat map based on writing sensor data for a ring using order-preserving partitioning with timestamp-based row keys.

# If We Choose Sensor ID

- ◆ Don't need to query for ranges of IDs
- ◆ Can use Random Partitioner (RP)
  - ◆ Nodes are assigned ranges of hash values
  - ◆ Row keys are hashed and stored based on value
  - ◆ For large number of row keys distribution is random
- ◆ Column key would be TimeStamp

# Writes are evenly distributed



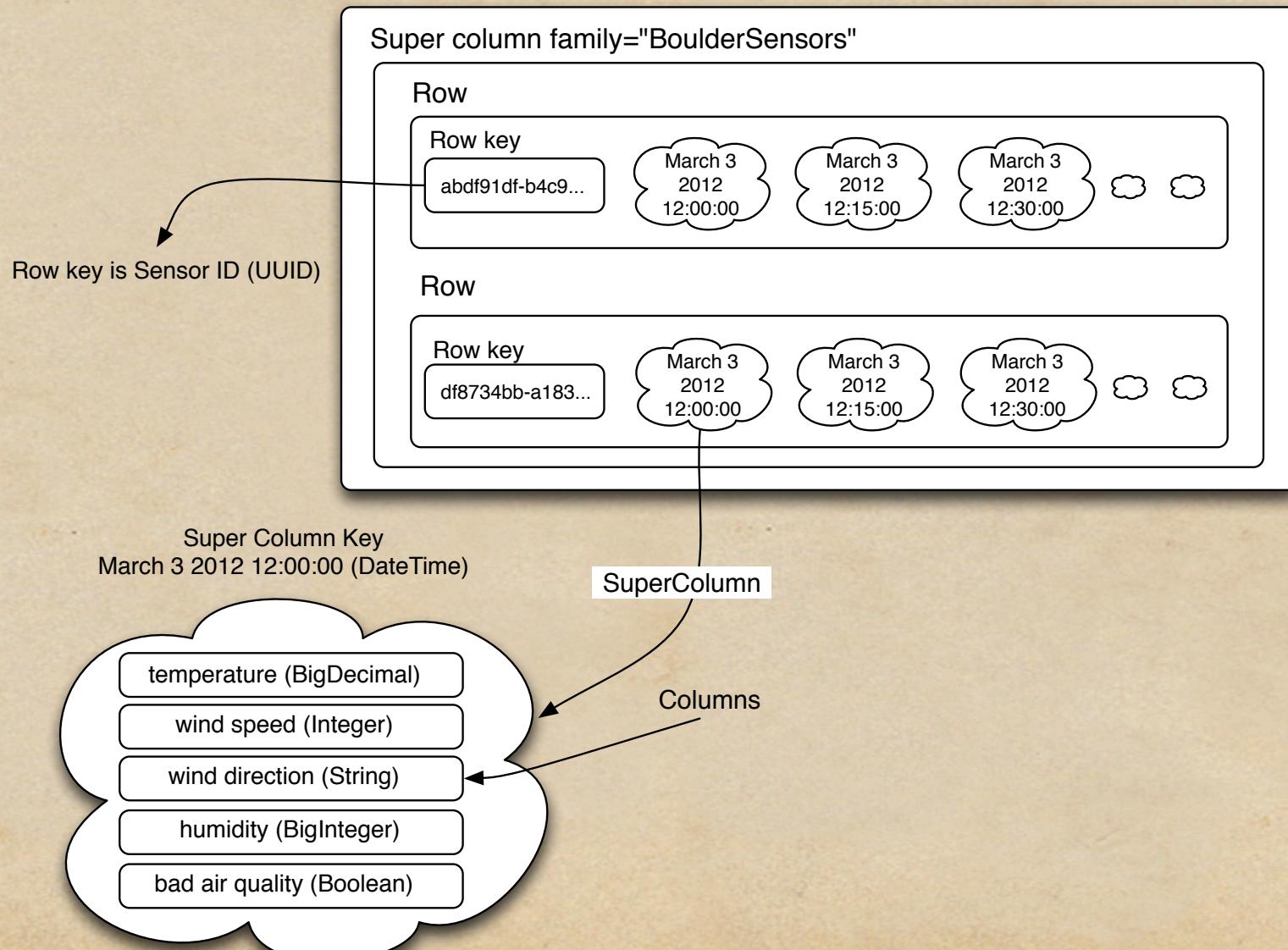
Theoretical partitioning and heat map based on our sensor data access patterns for a ring using random partitioning with hashed row keys based on Sensor IDs

# So Row Key is Sensor ID

- ◆ Note: if we needed to query ranges of sensors, would have to handle in application code
  - ◆ Essentially, multiple queries with the application doing the join
  - ◆ A good candidate for Map-Reduce?
  - ◆ YAGNI

# Looks like a Job for Super Columns!

Keyspace = "Climate"



# Using JSONish Notation

```
SensorNet: {  
    Climate: {  
        BoulderSensors: {  
            March 3 2012 12:00:00: {  
                Temperature: 23.0  
                WindSpeed: 16  
                WindDirection: "W"  
                Humidity: 17  
                BadAirQualityDetected: false  
            }  
        }  
    }  
}
```

Hector with Super Columns

# Snapshot from the CLI

```
[default@Climate] list BoulderSensors;
Using default limit of 100
-----
RowKey: 000000000000000000000000000000c8
=> (super_column=00000135fe7bcebd,
    (column=4261644169725175616c6974794465746563746564, value=00, timestamp=1331414421255004, ttl=31536000)
    (column=48756d6964697479, value=11, timestamp=1331414421255003, ttl=31536000)
    (column=54656d7065726174757265, value=31392e35, timestamp=1331414421255000, ttl=31536000)
    (column=57696e64446972656374696f6e, value=455345, timestamp=1331414421255002, ttl=31536000)
    (column=57696e645370656564, value=00000018, timestamp=1331414421255001, ttl=31536000))

...
=> (super_column=00000135fef7675d,
    (column=4261644169725175616c6974794465746563746564, value=00, timestamp=1331414421257039, ttl=31536000)
    (column=48756d6964697479, value=11, timestamp=1331414421257038, ttl=31536000)
    (column=54656d7065726174757265, value=31382e36, timestamp=1331414421257035, ttl=31536000)
    (column=57696e64446972656374696f6e, value=455345, timestamp=1331414421257037, ttl=31536000)
    (column=57696e645370656564, value=00000018, timestamp=1331414421257036, ttl=31536000))

-----
RowKey: 00000000000000000000000000000064
=> (super_column=00000135fe7bcebd,
    (column=4261644169725175616c6974794465746563746564, value=00, timestamp=1331414421246003, ttl=31536000)
    (column=48756d6964697479, value=11, timestamp=1331414421246002, ttl=31536000)
    (column=54656d7065726174757265, value=32332e30, timestamp=1331414421240000, ttl=31536000)
    (column=57696e64446972656374696f6e, value=57, timestamp=1331414421246001, ttl=31536000)
    (column=57696e645370656564, value=00000010, timestamp=1331414421246000, ttl=31536000))

...
=> (super_column=00000135fef7675d,
    (column=4261644169725175616c6974794465746563746564, value=00, timestamp=1331414421257033, ttl=31536000)
    (column=48756d6964697479, value=11, timestamp=1331414421257032, ttl=31536000)
    (column=54656d7065726174757265, value=32332e39, timestamp=1331414421257029, ttl=31536000)
    (column=57696e64446972656374696f6e, value=57, timestamp=1331414421257031, ttl=31536000)
    (column=57696e645370656564, value=00000010, timestamp=1331414421257030, ttl=31536000))

2 Rows Returned.
Elapsed time: 86 msec(s).
[default@Climate]
```

# Super Column Issues

- ◆ Every sub-column has overhead
  - ◆ Choice of string for key adds some overhead
    - ◆ We control that, could change
  - ◆ Every sub-column has Cassandra timestamp and TTL
    - ◆ We don't control that, stuck with it
- ◆ Remember, queries are at the row level
  - ◆ We must pull in the entire super column for each request we want
- ◆ So super column queries can be expensive

# Enter Protocol Buffers

- ◆ Developed by Google
- ◆ Provide efficient way to serialize/deserialize objects
- ◆ Fall back to standard Cassandra column and store an entire reading as a serialized protocol buffer
- ◆ ProtoBuf uses ByteString, not ByteBuffer
  - ◆ Google type
- ◆ Both approaches are valid, we have to handle the impedance mismatch

# Protocol Buffer Intro

- ◆ Developed and open-sourced by Google
  - ◆ <http://code.google.com/apis/protocolbuffers/>
- ◆ ProtoBuf definition created in .proto file using a DSL
- ◆ Then translated to Java file using ProtoBuf complier
  - ◆ `protoc --java_out=../src/main/java/ reading_buffer.proto`
- ◆ Provides a builder interface to create and use the object

# Example .proto file

```
package com.jeklsoft.cassandraclient;

option java_package = "com.jeklsoft.cassandraclient";
option java_outer_classname = "ReadingBuffer";

message Reading {
    optional bytes temperature = 1; // BigDecimal
    optional int32 wind_speed = 2; // Integer
    optional string wind_direction = 3; // String
    optional bytes humidity = 4; // BigInteger
    optional bool bad_air_quality_detected = 5; // Boolean
}
```

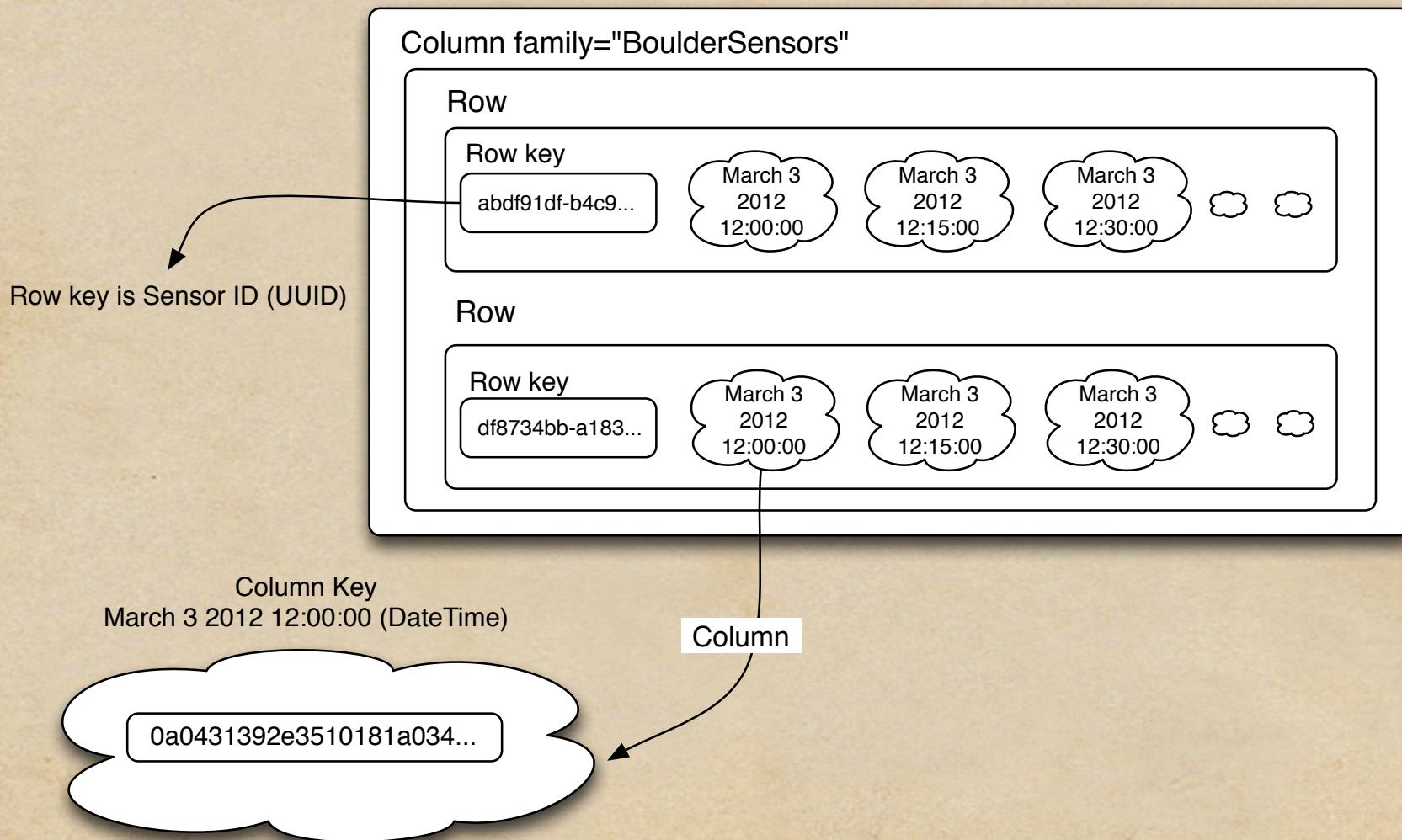
# Using a ProtoBuf

```
private static ReadingBuffer.Reading getBufferedReading(Reading reading) {  
    return ReadingBuffer.Reading.newBuilder()  
        .setTemperature(getByteString(BigDecimalSerializer.get(),  
                                      reading.getTemperature()))  
        .setWindSpeed(reading.getWindSpeed())  
        .setWindDirection(reading.getDirection())  
        .setHumidity(getByteString(BigIntegerSerializer.get(),  
                                      reading.getHumidity()))  
        .setBadAirQualityDetected(reading.getBadAirQualityDetected())  
        .build();  
}  
  
private static Reading getReading(ReadingBuffer.Reading bufferedReading) {  
    return new Reading((BigDecimal) getObject(BigDecimalSerializer.get(),  
                                              bufferedReading.getTemperature()),  
                      bufferedReading.getWindSpeed(),  
                      bufferedReading.getWindDirection(),  
                      (BigInteger) getObject(BigIntegerSerializer.get(),  
                                      bufferedReading.getHumidity()),  
                      bufferedReading.getBadAirQualityDetected());  
}
```

Now we're ready to model  
using a column

# Schema using Column

Keyspace = "Cimate"



# Using JSONish Notation

```
SensorNet: {  
    Climate: {  
        BoulderSensors: {  
            March 3 2012 12:00:00: 0a0431392e3510181a034553452201112800  
        }  
    }  
}
```

# Hector with Protocol Buffers

# Snapshot from the CLI

```
[default@Climate] list BoulderSensors;
Using default limit of 100
-----
RowKey: 0000000000000000000000000000c8
=> (column=00000135fe7fdb04, value=0a0431392e3510181a034553452201112800, timestamp=1331414686561000, ttl=31536000)
=> (column=00000135fe8d96a4, value=0a0431392e3410181a034553452201112800, timestamp=1331414686561002, ttl=31536000)
=> (column=00000135fe9b5244, value=0a0431392e3310181a034553452201112800, timestamp=1331414686561004, ttl=31536000)
=> (column=00000135fea90de4, value=0a0431392e3210181a034553452201112800, timestamp=1331414686561006, ttl=31536000)
=> (column=00000135feb6c984, value=0a0431392e3110181a034553452201112800, timestamp=1331414686562000, ttl=31536000)
=> (column=00000135fec48524, value=0a0431392e3010181a034553452201112800, timestamp=1331414686562002, ttl=31536000)
=> (column=00000135fed240c4, value=0a0431382e3910181a034553452201112800, timestamp=1331414686562004, ttl=31536000)
=> (column=00000135fedffc64, value=0a0431382e3810181a034553452201112800, timestamp=1331414686562006, ttl=31536000)
=> (column=00000135feedb804, value=0a0431382e3710181a034553452201112800, timestamp=1331414686562008, ttl=31536000)
=> (column=00000135fefbf73a4, value=0a0431382e3610181a034553452201112800, timestamp=1331414686562010, ttl=31536000)
-----
RowKey: 000000000000000000000000000064
=> (column=00000135fe7fdb04, value=0a0432332e3010101a01572201112800, timestamp=1331414686529000, ttl=31536000)
=> (column=00000135fe8d96a4, value=0a0432332e3110101a01572201112800, timestamp=1331414686561001, ttl=31536000)
=> (column=00000135fe9b5244, value=0a0432332e3210101a01572201112800, timestamp=1331414686561003, ttl=31536000)
=> (column=00000135fea90de4, value=0a0432332e3310101a01572201112800, timestamp=1331414686561005, ttl=31536000)
=> (column=00000135feb6c984, value=0a0432332e3410101a01572201112800, timestamp=1331414686561007, ttl=31536000)
=> (column=00000135fec48524, value=0a0432332e3510101a01572201112800, timestamp=1331414686562001, ttl=31536000)
=> (column=00000135fed240c4, value=0a0432332e3610101a01572201112800, timestamp=1331414686562003, ttl=31536000)
=> (column=00000135fedffc64, value=0a0432332e3710101a01572201112800, timestamp=1331414686562005, ttl=31536000)
=> (column=00000135feedb804, value=0a0432332e3810101a01572201112800, timestamp=1331414686562007, ttl=31536000)
=> (column=00000135fefbf73a4, value=0a0432332e3910101a01572201112800, timestamp=1331414686562009, ttl=31536000)

2 Rows Returned.
Elapsed time: 15 msec(s).
[default@Climate]
```

# The Benefits of Column

- ◆ At the expense of an extra application-level serialization on write, deserialization on read
  - ◆ Storage space on nodes is reduced
  - ◆ Network traffic between nodes is reduced
  - ◆ Benefits scale as a multiple of Replication Factor!
- ◆ This is the way to go

# Astyanax with Protocol Buffers

# Hector vs Astyanax

- ◆ No clear winner, both libraries work very well
- ◆ Hector
  - ◆ The incumbent
  - ◆ Great support from DataStax via the Google Group (they also have commercial support)
  - ◆ Will be playing catch-up on some Astyanax features, namely async support
- ◆ Astyanax
  - ◆ Netflix has a great rep for quality code
  - ◆ Support channel still unclear to me
  - ◆ Learned from Hector, kept the best, modified the rest
  - ◆ Asynchronous model probably better approach

# References

- ◆ This project, <https://github.com/jmctee/Cassandra-Client-Tutorial>
- ◆ Tim Berglund, “Radical NoSQL Scalability with Cassandra”
  - ◆ Catch him at NFJS or UberConf for everything I didn’t talk about
- ◆ Berglund and McCullough, Mastering Cassandra for Architects
  - ◆ <http://shop.oreilly.com/product/0636920024811.do>
- ◆ Apache Cassandra, <http://cassandra.apache.org>
- ◆ DataStax - Commercial Cassandra support, <http://www.datastax.com/>
- ◆ Hector, <http://rantav.github.com/hector>
- ◆ Hector mailing list, <http://groups.google.com/group/hector-users>
- ◆ Astyanax, <https://github.com/Netflix/astyanax/wiki/Getting-Started>

# References, cont.

- ◆ WTF Is A Super Column, <http://arin.me/blog/wtf-is-a-supercolumn-cassandra-data-model>
- ◆ Hewitt, Cassandra: The Definitive Guide
  - ◆ <http://shop.oreilly.com/product/0636920010852.do>
- ◆ Capriolo, Cassandra: High Performance Cookbook
  - ◆ <http://www.packtpub.com/cassandra-apache-high-performance-cookbook/book>
- ◆ Also, thanks to Ben Hoyt at Tendril for advice and technical contributions to this preso!

# Thank You!



Did I mention Tendril is hiring?

