Building Java Applications

Technical Evangelist for Apache Cassandra
@chbatey

Cassandra Days brought to you by DataStax

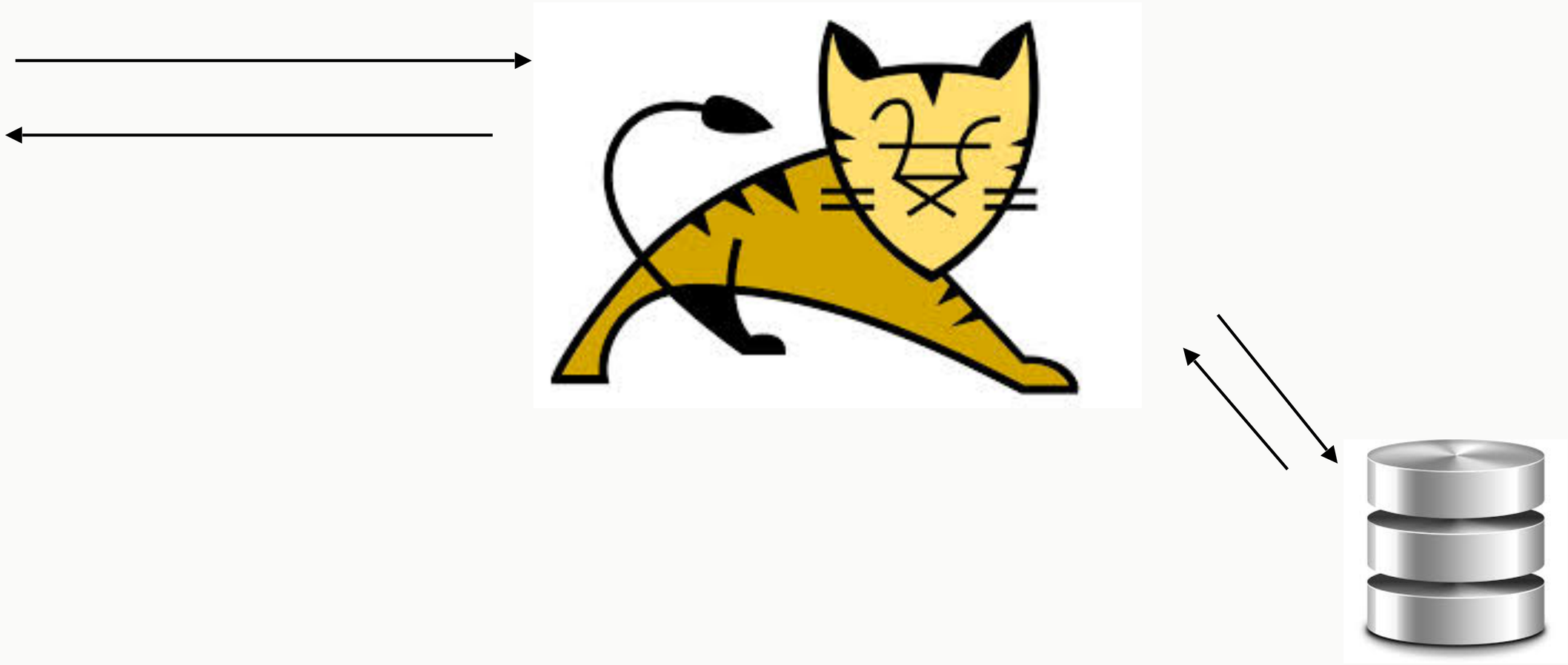
Who am I?

- Technical Evangelist for Apache Cassandra
  - Founder of Stubbed Cassandra
  - Help out Apache Cassandra users
- DataStax
  - Builds enterprise ready version of Apache Cassandra
- Previous: Cassandra backed apps at BSkyB

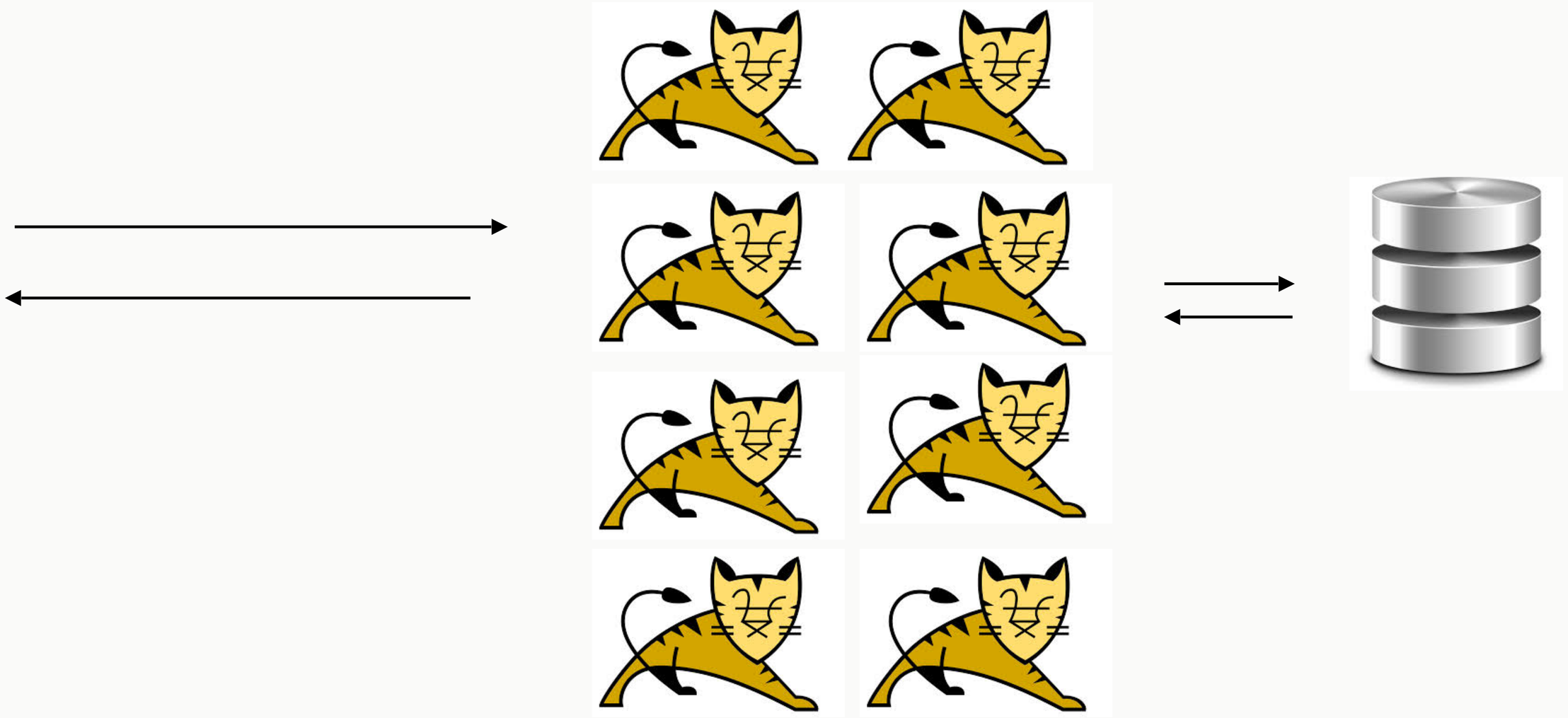
Agenda

- Everything I wish I knew before I started
- Example application: Auction service
  - Build + Deploy
  - Tech stack: Spring vs Dropwizard
  - Configuration
  - Designing schemas
  - Example Driver code including LWTs
  - Continuous Integration + Testing strategies

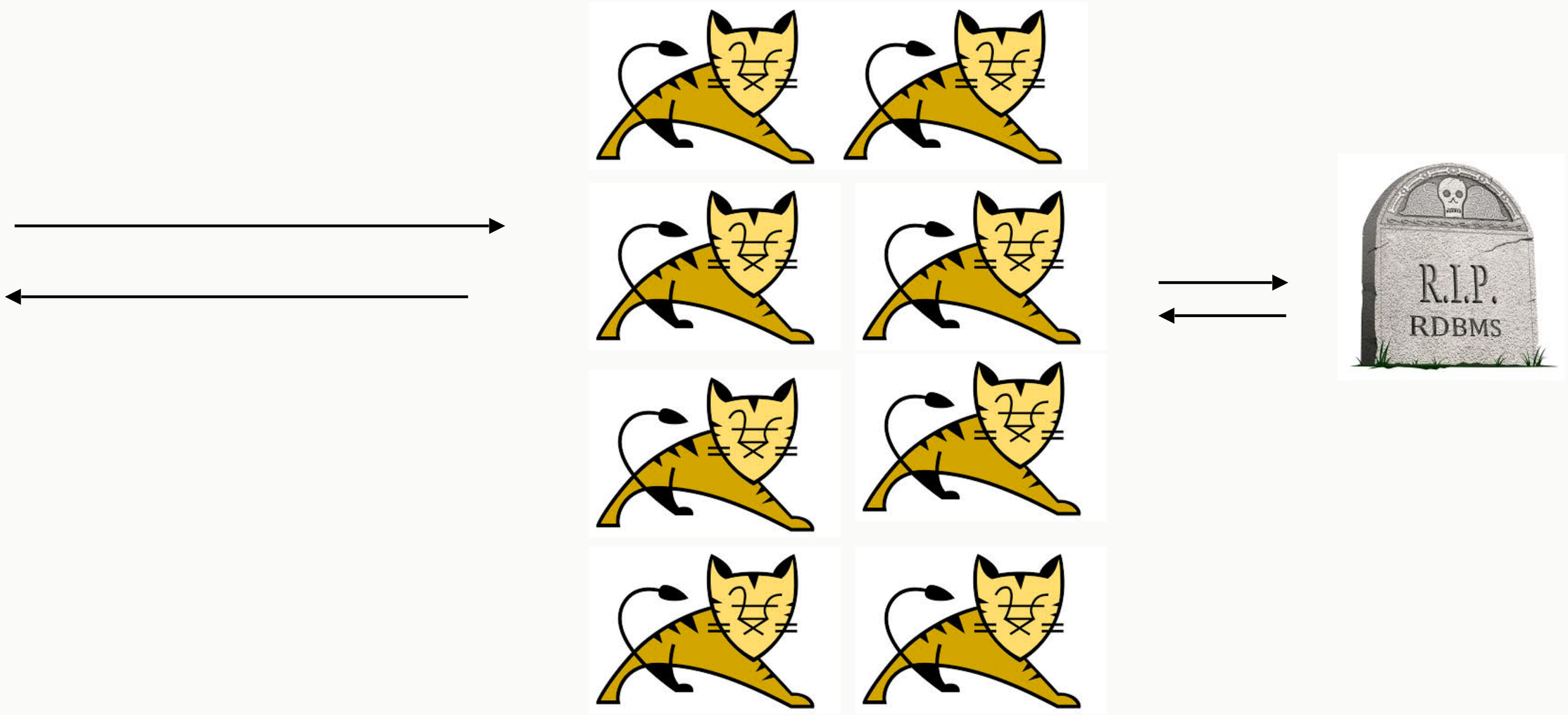
The old



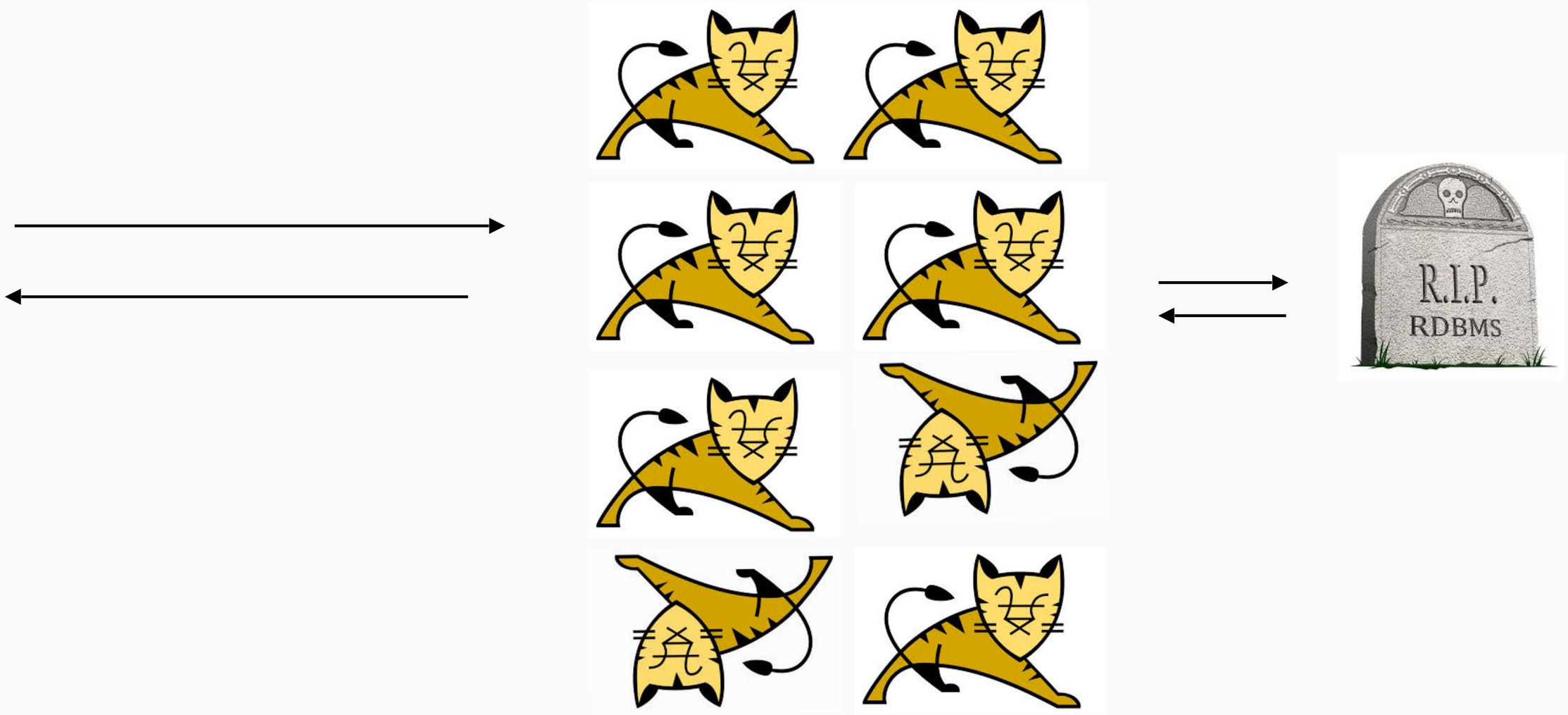
Scaling the old



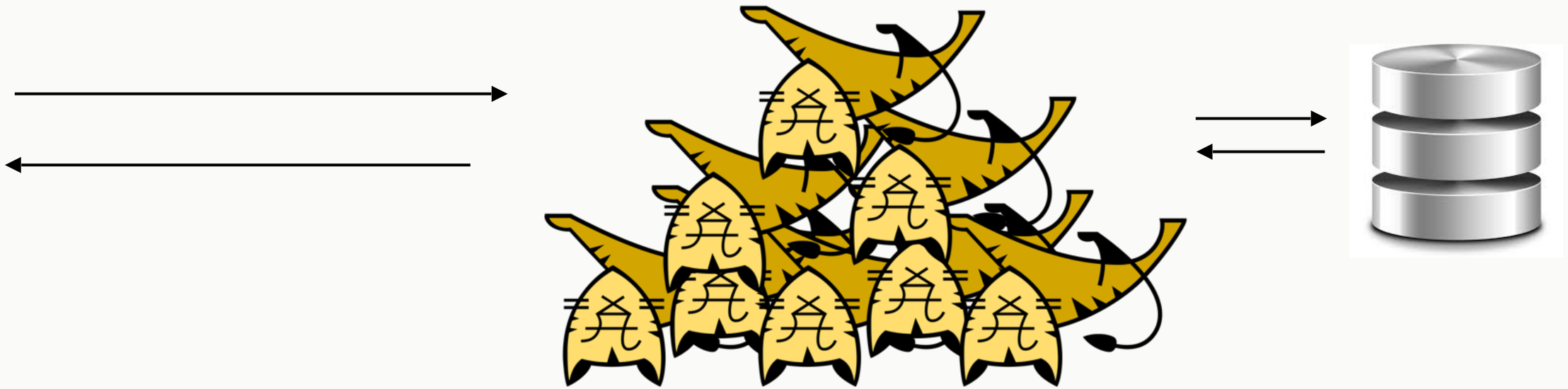
Uh oh :(



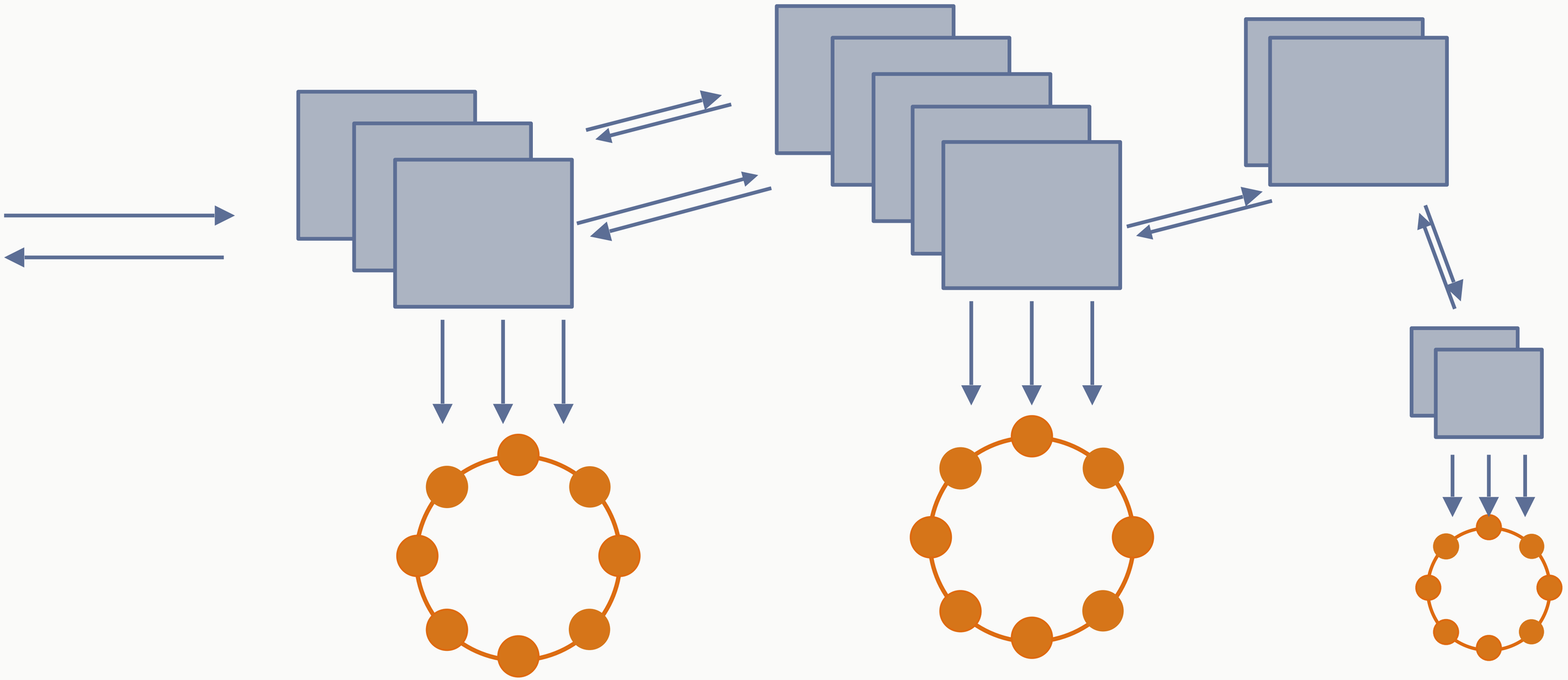
Uh oh :(



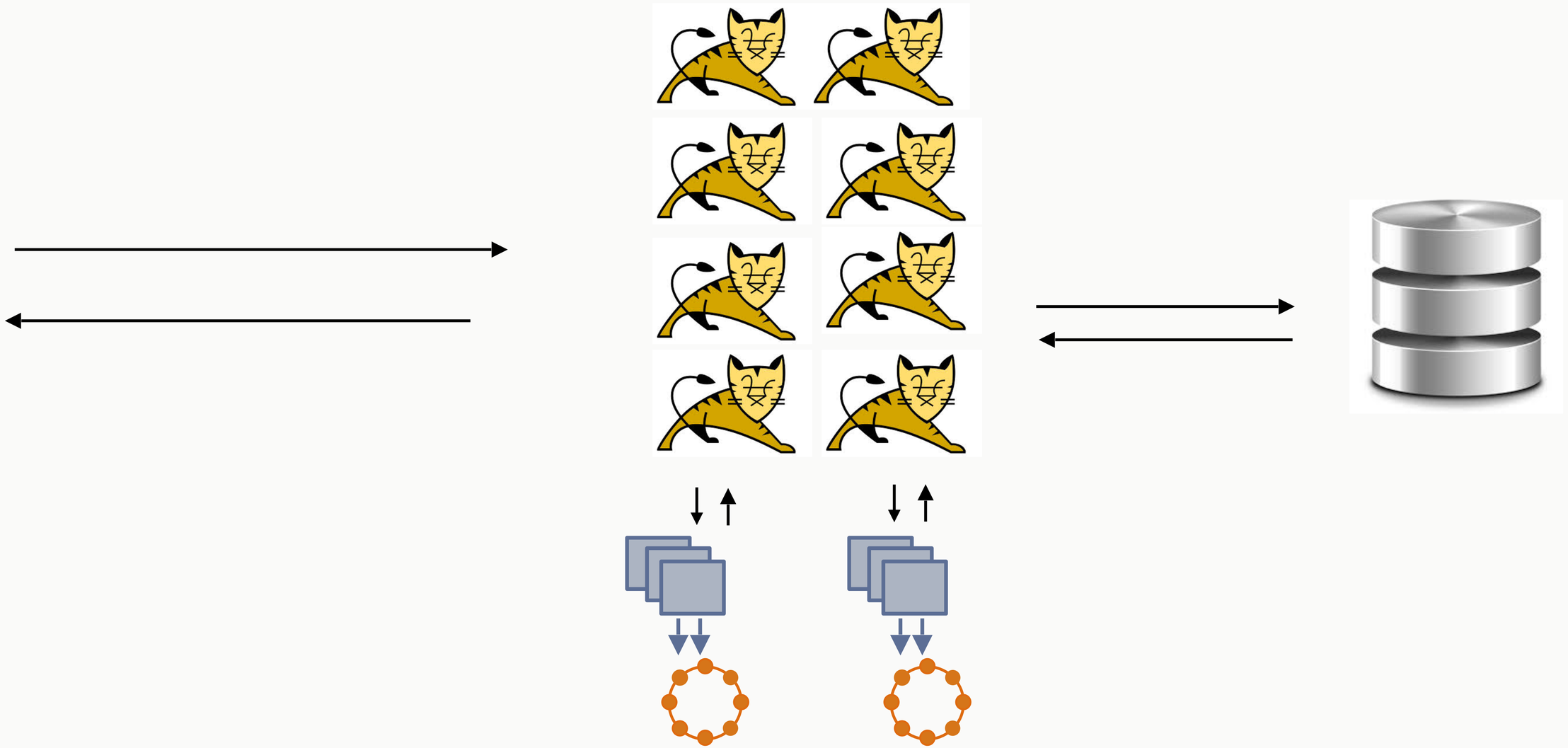
A pile of cats



The new

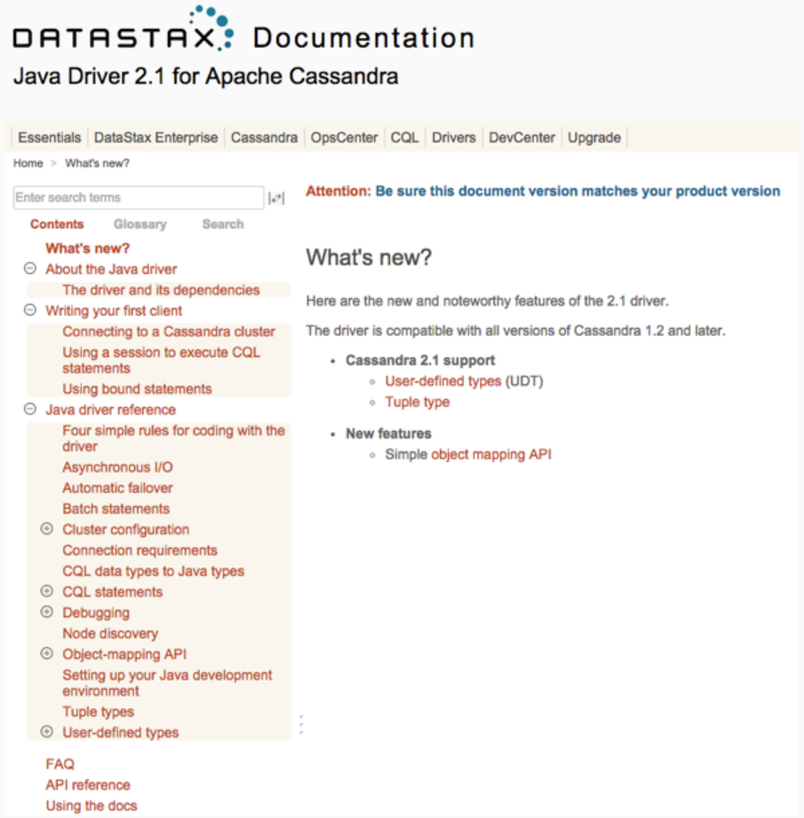


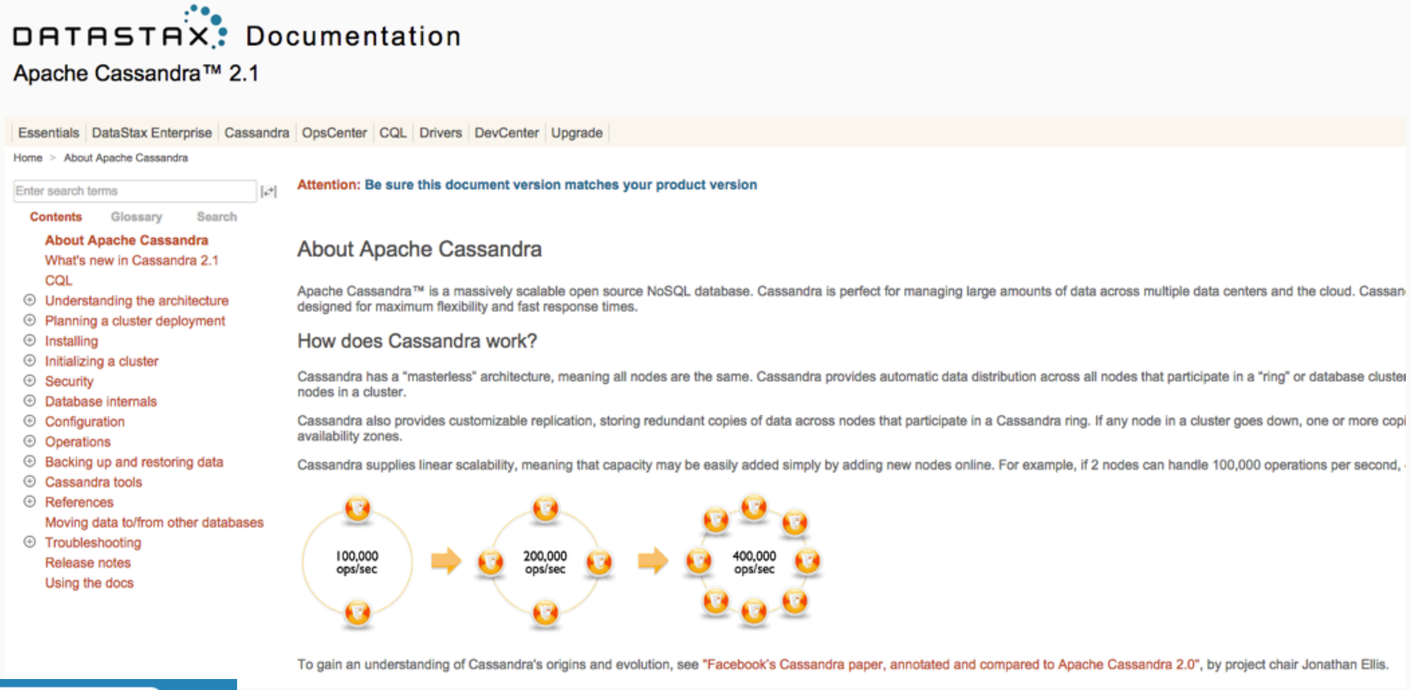
Strangulation



Prerequisites

Use the DataStax docs





Say no to thrift

http://exponential.io/blog/2015/01/08/cassandra-terminology/

Let's Build Something Together!

Production ready applications

- Build: Gradle
- Web framework: Spring boot
- Monitoring: Dropwizard metrics + Graphite
- Testing: Stubbed Cassandra + Cassandra Unit

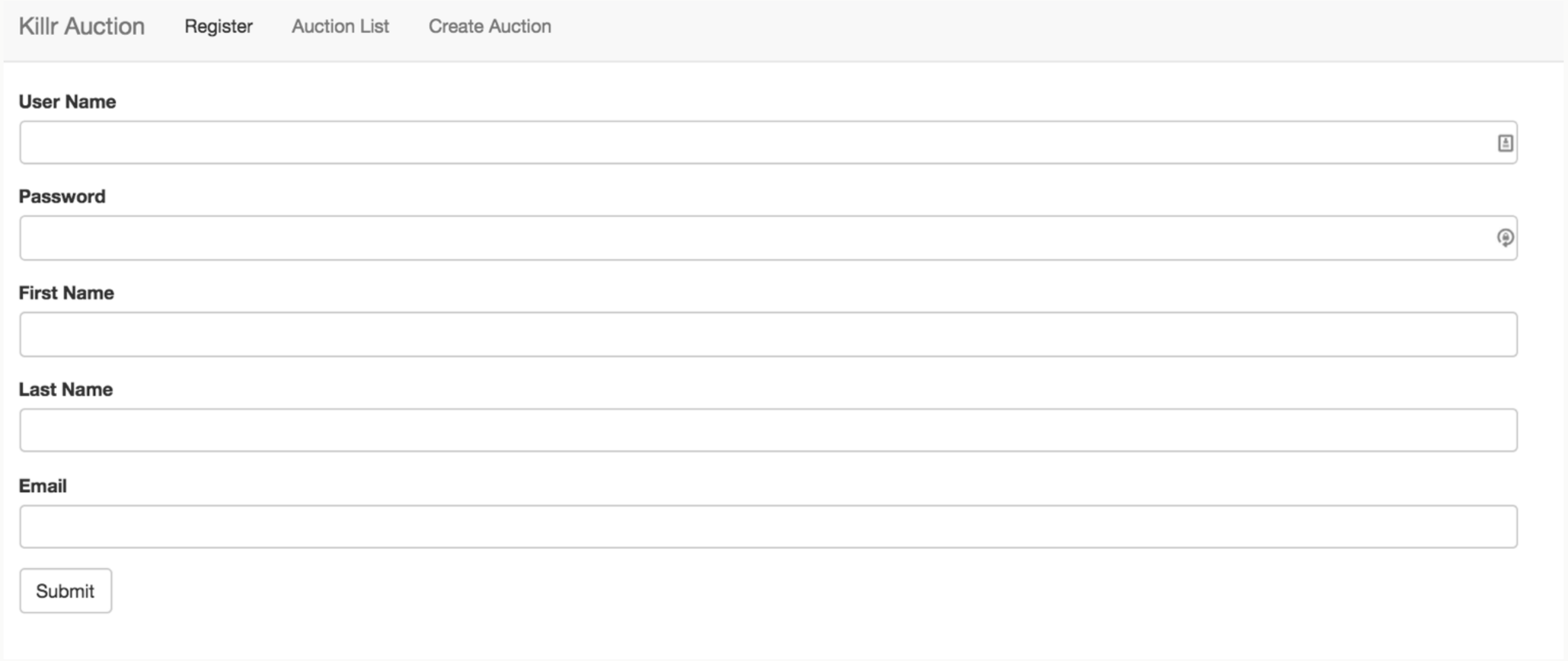
KillrAuction

- Register users
- Create auctions
- Bid on auctions
- Stream auction bids

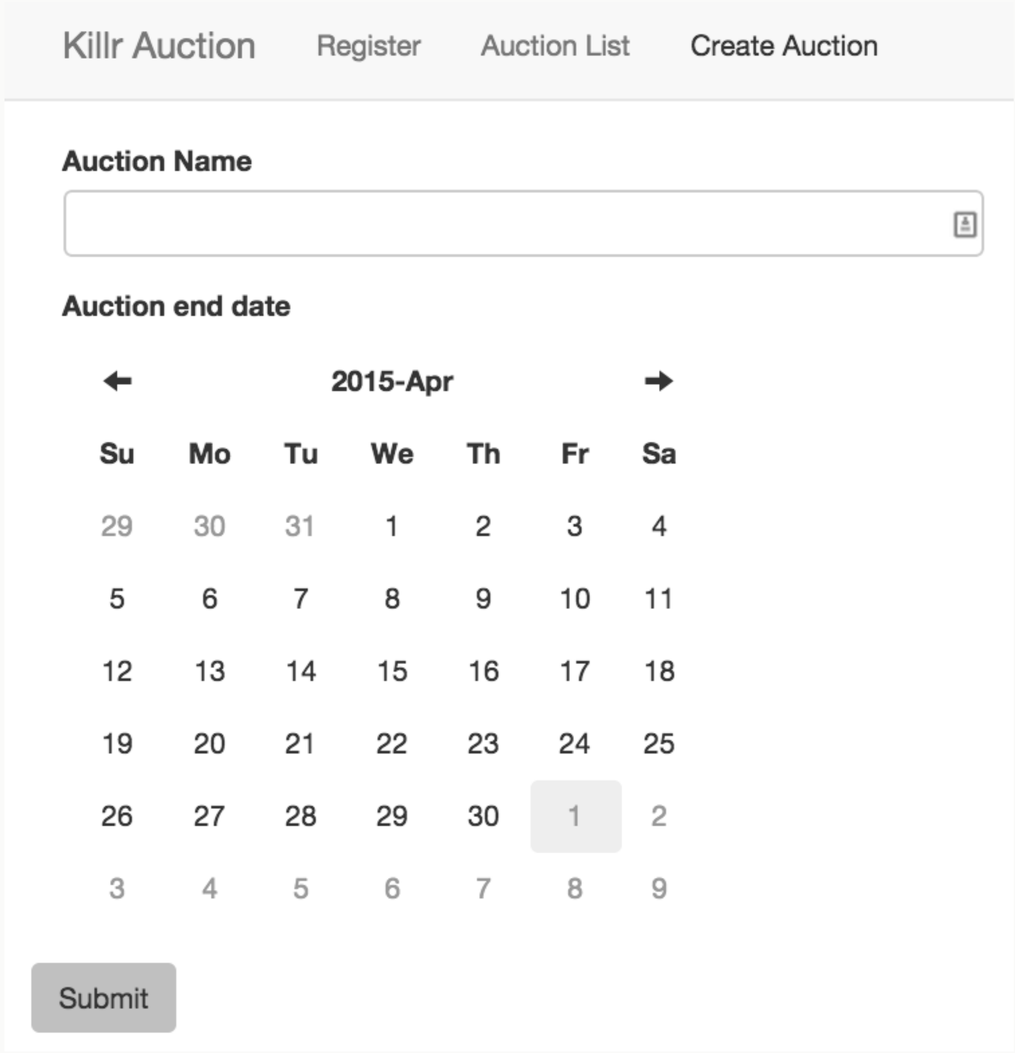
https://github.com/chbatey/killrauction

https://github.com/chbatey/cassandra-customer-events

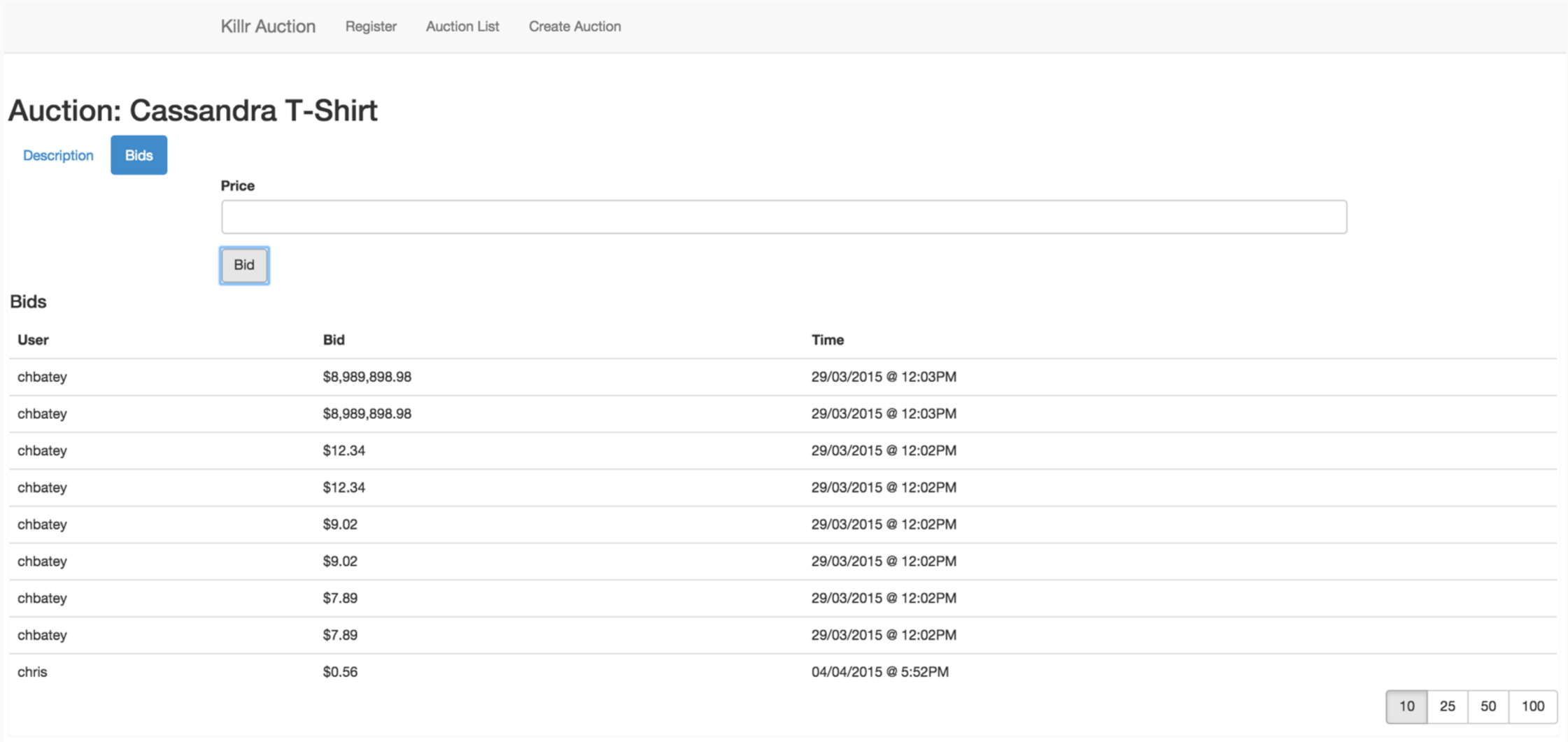
Creating a user



Creating an Auction



Viewing and bidding on an Auction



The tech stack

Front end

Interaction

Backend end

Storage layer

Gradle for building

- Or Maven/SBT
- Download dependencies
- Build + unit test
- Acceptance tests with cucumber
- Load tests with Gatling

Build and deployment

```
dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    compile("org.springframework.boot:spring-boot-starter-security")
    compile("org.springframework.boot:spring-boot-starter-websocket")
    compile("org.springframework:spring-websocket:4.2.0.BUILD-SNAPSHOT")
    compile("org.springframework:spring-messaging:4.2.0.BUILD-SNAPSHOT")
    compile("com.datastax.cassandra:cassandra-driver-core:${cassandraDriverVersion}")
    compile("com.datastax.cassandra:cassandra-driver-mapping:${cassandraDriverVersion}")
    compile('io.reactivex:rxjava:1.0.8')

    // unit + acceptance testing
    testCompile("org.springframework.boot:spring-boot-starter-test")
    testCompile("info.cukes:cucumber-junit:${cucumberVersion}")
    testCompile("info.cukes:cucumber-spring:${cucumberVersion}")
    testCompile("org.apache.httpcomponents:httpclient:${apacheHttpVersion}")
    testCompile("org.apache.httpcomponents:fluent-hc:${apacheHttpVersion}")

    // for performance testing
    testCompile "org.scala-lang:scala-library:2.11.5"
    testCompile "io.gatling:gatling-app:${gatlingVersion}"ß
}
```

Languages

- DataStax (open source)
  - C#, **Java**, C++, Python, Node, Ruby
  - Very similar programming API
- Other open source
  - Go
  - Clojure
  - Erlang
  - Haskell
  - Many more Java/Python drivers
  - Perl

DataStax Java Driver

- Open source

DataStax Java Driver for Apache Cassandra

A Java client driver for Apache Cassandra. This driver works exclusively with the Cassandra Query Language version 3 (CQL3) and Cassandra's binary protocol.

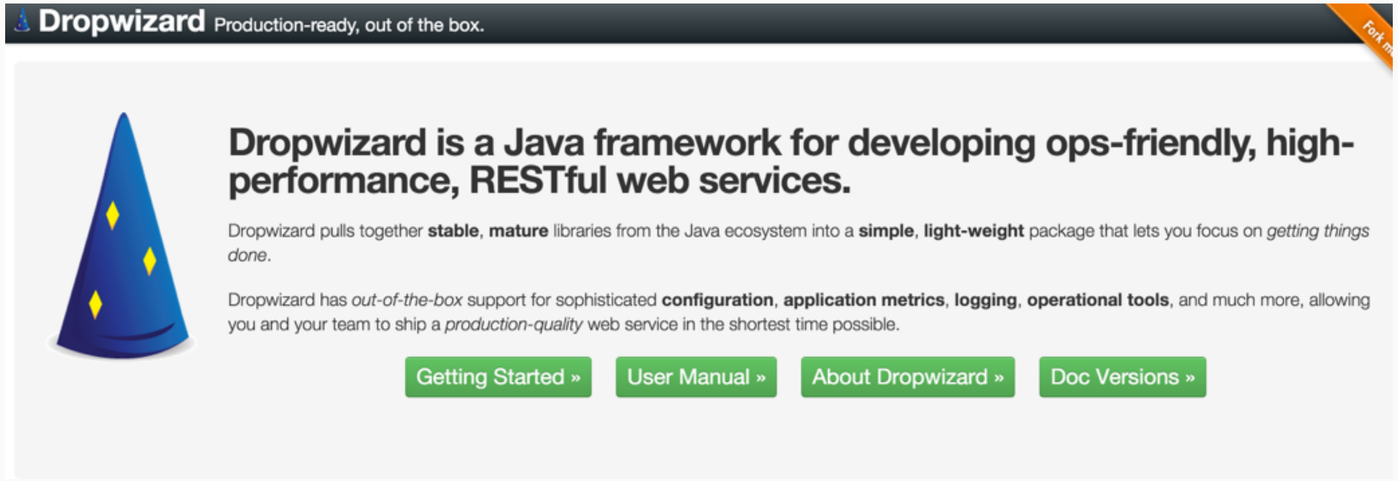
- JIRA: https://datastax-oss.atlassian.net/browse/JAVA
- MAILING LIST: https://groups.google.com/a/lists.datastax.com/forum/#!forum/java-driver-user
- IRC: #datastax-drivers on irc.freenode.net
- TWITTER: Follow the latest news about DataStax Drivers - @olim7t, @mfiguiere
- DOCS: http://www.datastax.com/documentation/developer/java-driver/2.1/index.html
- API: http://www.datastax.com/drivers/java/2.1
- CHANGELOG: https://github.com/datastax/java-driver/blob/2.1/driver-core/CHANGELOG.rst

The driver architecture is based on layers. At the bottom lies the driver core. This core handles everything related to the connections to a Cassandra cluster (for example, connection pool, discovering new nodes, etc.) and exposes a simple, relatively low-level API on top of which higher level layer can be built.

```
<dependency>
  <groupId>com.datastax.cassandra</groupId>
  <artifactId>cassandra-driver-core</artifactId>
  <version>2.1.2</version>
</dependency>
```

```
<dependency>
  <groupId>com.datastax.cassandra</groupId>
  <artifactId>cassandra-driver-mapping</artifactId>
  <version>2.1.2</version>
</dependency>
```

Dropwizard vs Spring boot



play

akka

spring

VERT.X

Vert.x is a lightweight, high performance application platform for the JVM that's designed for modern mobile, web, and enterprise applications.

Spring

Configuration

```java
@Component
@ConfigurationProperties(prefix = "cassandra")
public class CassandraConfiguration {

    @NotNull
    private String[] contactPoints;

    @NotNull
    private String keyspace;

    public String[] getContactPoints() {
        return this.contactPoints;
    }

    public String getKeyspace() {
        return keyspace;
    }

    @Override
    public String toString() {
        return "CassandraConfiguration{" +
                "contactPoints=" + Arrays.toString(contactPoints) +
                ", keyspace='" + keyspace + '\'' +
                '}';
    }
}
```

```yaml
cassandra:
  contactPoints:
    - localhost
  keyspace: killrauction
```

- **SSL?**
- **Socket options**
- **Compression**

Contents Glossary Search
What's new?
About the Java driver
Writing your first client
Java driver reference
Four simple rules for coding with the driver
Asynchronous I/O
Automatic failover
Batch statements
Cluster configuration
Tuning policies
Connection options
Connection requirements
CQL data types to Java types
CQL statements
Debugging
Node discovery
Object-mapping API
Setting up your Java development environment
Tuple types
User-defined types
FAQ
API reference
Using the docs

Connection options

Related concepts
Tuning policies

Protocol options

Description

Protocol options

Option	Description	Default
port	The port to connect to a Cassandra node on.	9042
compression	What kind of compression to use when sending data to a node: either no compression or snappy. Snappy compression isoptimized for high speeds and reasonable compression.	ProtocolOptions.Compression.NONE

Pooling options

Description

The driver only needs to maintain a relatively small number of connections to each Cassandra host. These options allow you to control how many connections are kept exactly. The defaults should be fine for most applications.

Connection pooling options

Option	Description	Default value
coreConnectionsPerHost	The core number of connections per host.	2 for HostDistance.LOCAL, 1 for HostDistance.REMOTE
maxConnectionPerHost	The maximum number of connections per host.	8 for HostDistance.LOCAL, 2 for HostDistance.REMOTE
maxSimultaneousRequestsPerConnectionThreshold	The number of simultaneous requests on all connections to an host after which more connections are created.	128
minSimultaneousRequestsPerConnectionThreshold	The number of simultaneous requests on a connection below which connections in excess are reclaimed.	25

Socket options

Description

All but one of these socket options comes from the Java runtime library's SocketOptions class. The connectTimeoutMillis option though is from Netty's ChannelConfig class.

Pooling options

Option	Corresponds to	Description
connectTimeoutMillis	org.jboss.netty.channel.ChannelConfig "connectTineoutMillis"	The connect timeout in milliseconds for the underlying Netty channel.
keepAlive	java.net.SocketOptions.SO_KEEPALIVE	
receiveBufferSize	java.net.SocketOptions.SO_RCVBUF	A hint on the size of the buffer used to receive data.
reuseAddress	java.net.SocketOptions.SO_REUSEADDR	Whether to allow the same port to be bound to multiple times.
sendBufferSize	java.net.SocketOptions.SO_SNDBUF	A hint on the size of the buffer used to send data.
soLinger	java.net.SocketOptions.SO_LINGER	When specified, disables the immediate return from a call to close() on a TCP socket.
tcpNoDelay	java.net.SocketOptions.TCPNODELAY	Disables Nagle's algorithm on the underlying socket.

Cluster + Session

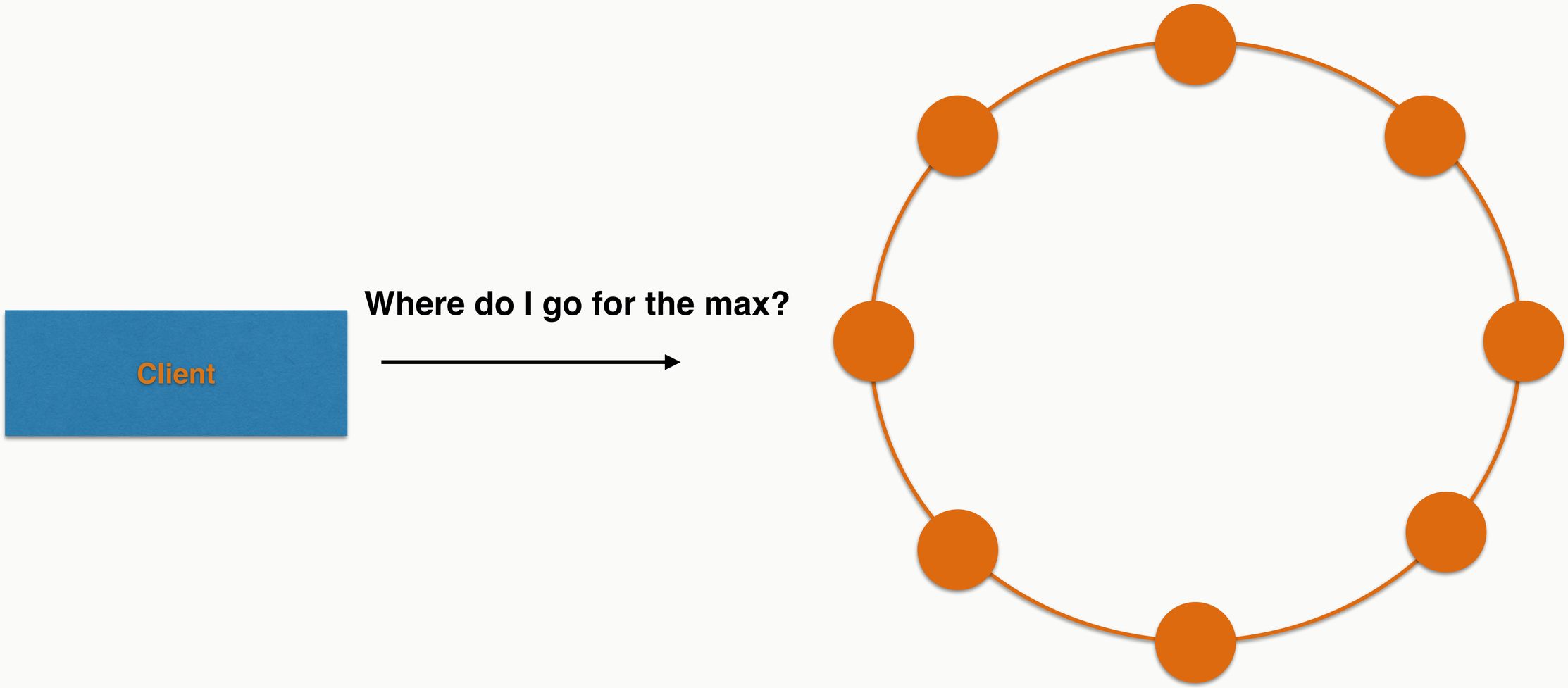
```
@Component
public class CassandraSessionFactoryBean implements FactoryBean<Session> {
    @Inject
    private CassandraConfiguration cassandraConfiguration;

    private Cluster cluster;

    @Override
    public Session getObject() throws Exception {
        cluster = Cluster.builder()
                .addContactPoints(cassandraConfiguration.getContactPoints())
                .build();
        return cluster.connect(cassandraConfiguration.getKeyspace());
    }

    @Override
    public boolean isSingleton() {
        return true;
    }

    @PreDestroy
    public void shutdown() {
        LOGGER.info("Shutting down cluster");
        cluster.close();
    }
}
```

Session per keyspace

Singleton!!

Data modelling time

Cassandra can not join or aggregate



Modelling in Cassandra

```
CREATE TABLE customer_events(
    customer_id text,
    staff_id text,
    time timeuuid,
    store_type text,
    event_type text,
    tags map<text, text>,
    PRIMARY KEY ((customer_id), time));
```

Partition Key

Clustering Column(s)

```
cqlsh:customers> select * from customer_events where customer_id = 'chbatey' and time > minTimeuuid(1) and time < maxTimeuuid(2000000000000) ;

 customer_id | time                                 | event_type | staff_id | store_type | tags
-------------+--------------------------------------+------------+----------+------------+--------------------
     chbatey | 2b329cc0-73f0-11e4-ac06-4b05b98cc84c | basket_add |   trevor |     online | {'item': 'coffee'}
     chbatey | 6a823160-73f0-11e4-ac06-4b05b98cc84c | basket_add |   trevor |     online | {'item': 'coffee'}
```

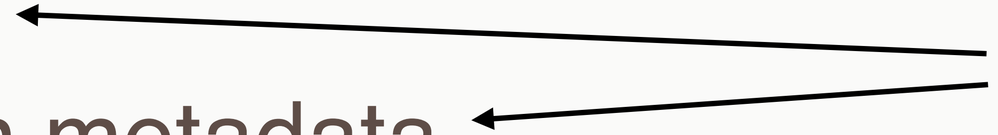
How it is stored on disk

customer	time	event_type	store_type	tags
charles	2014-11-18 16:52:04	basket_add	online	{'item': 'coffee'}
charles	2014-11-18 16:53:00	basket_add	online	{'item': ‘wine'}
charles	2014-11-18 16:53:09	logout	online	{}
chbatey	2014-11-18 16:52:21	login	online	{}
chbatey	2014-11-18 16:53:21	basket_add	online	{'item': 'coffee'}
chbatey	2014-11-18 16:54:00	basket_add	online	{'item': 'cheese'}

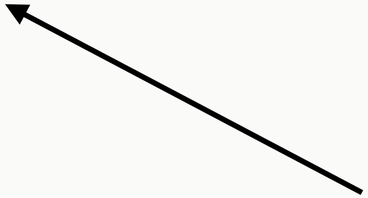
charles	**event_type** basket_add	**staff_id** n/a	**store_type** online	**tags:item** coffee	**event_type** basket_add	**staff_id** n/a	**store_type** online	**tags:item** wine	**event_type** logout	**staff_id** n/a	**store_type** online
chbatey	**event_type** login	**staff_id** n/a	**store_type** online	**event_type** basket_add	**staff_id** n/a	**store_type** online	**tags:item** coffee	**event_type** basket_add	**staff_id** n/a	**store_type** online	**tags:item** cheese

Requirements

- Store users
- Store auction metadata
- Store auction bids



Lowish throughput



High throughput / reads of many rows
Get top N bids quickly

Designing a schema for users

- Fields
  - Username
  - First name
  - Last name
  - Email addresses?
- Unique user names

Users table

```
CREATE TABLE killrauction.users (
 user_name text PRIMARY KEY,
 password text,
 salt bigint,
 first_name text,
 last_name text,
 emails set<text> ) ;
```

User DAO

```
private static final String CREATE_USER_STATEMENT = "INSERT INTO users
(user_name, password, salt, first_name , last_name , emails ) values
(?, ?, ?, ?, ?, ?)";
```

```
@Inject
private Session session;
@Inject
private Md5PasswordEncoder md5PasswordEncoder;
@Inject
private SecureRandom secureRandom;

private PreparedStatement createUser;
```

```
@PostConstruct
public void prepareStatements() {
    createUser = session.prepare(CREATE_USER_STATEMENT);
}
```

```
public void createUser(UserCreate userCreate) {
    Object salt = secureRandom.nextLong();
    String endcodedPassword = md5PasswordEncoder.encodePassword(userCreate.getPassword(), salt);

    BoundStatement boundStatement = createUser.bind(userCreate.getUserName(),
          endcodedPassword,
          salt,
          userCreate.getFirstName(), userCreate.getLastName(), userCreate.getEmails());

    session.execute(boundStatement);
}
```

Avoiding overwriting existing users

```
private static final String CREATE_USER_STATEMENT = "INSERT INTO users
(user_name, password, salt, first_name , last_name , emails ) values
(?, ?, ?, ?, ?, ?) IF NOT EXISTS";
```

```
public boolean createUser(UserCreate userCreate) {
    Object salt = secureRandom.nextLong();
    String endcodedPassword = md5PasswordEncoder.encodePassword(userCreate.getPassword(), salt);
    BoundStatement boundStatement = createUser.bind(userCreate.getUserName(),
          endcodedPassword,
          salt,
          userCreate.getFirstName(), userCreate.getLastName(), userCreate.getEmails());
    ResultSet response = session.execute(boundStatement);
    boolean applied = response.wasApplied();
    return applied;
}
```

AuctionDao

```
CREATE TABLE IF NOT EXISTS killrauction.auctions (
 name text primary key,
 owner text,
 ends bigint);
```

```
@PostConstruct
public void prepareStatements() {
    createAuction = session.prepare("insert INTO auctions (name, owner, ends) VALUES (?, ?, ?)");
    getAuction = session.prepare("select * from auctions where name = ?");
    getAllAuctionSparse = session.prepare("select * from auctions");
}
```

UUID for auction or use the name? Raise a PR :)

Auction Bids

```
CREATE TABLE IF NOT EXISTS killrauction.auction_bids (
 name TEXT,
 bid_time TIMEUUID,
 bid_user TEXT,
 bid_amount BIGINT,
 PRIMARY KEY (name, bid_amount, bid_time ) )
 WITH CLUSTERING ORDER BY (bid_amount DESC )
```

Bids stored on disk in order of amount

Descending so we can always get the top N bids

All bids for the same auction on the same node

Auction Bids

```
cqlsh:killrauction> select * from auction_bids where name = 'Cassandra T-Shirt';

 name               | bid_amount | bid_time                             | bid_user
--------------------+------------+--------------------------------------+----------
 Cassandra T-Shirt |  898989898 | 2ae0d0a0-d603-11e4-a23d-097f255080d8 |  chbatey
 Cassandra T-Shirt |       1234 | 23332d80-d603-11e4-a23d-097f255080d8 |  chbatey
 Cassandra T-Shirt |        902 | 2632b1e0-d603-11e4-a23d-097f255080d8 |  chbatey
 Cassandra T-Shirt |        789 | 28c25d70-d603-11e4-a23d-097f255080d8 |  chbatey
```

```
cqlsh:killrauction> select name, bid_amount, dateOf(bid_time), bid_user from auction_bids where name = 'Cassandra T-Shirt';

 name               | bid_amount | dateOf(bid_time)         | bid_user
--------------------+------------+--------------------------+----------
 Cassandra T-Shirt |  898989898 | 2015-03-29 12:03:02+0100 |  chbatey
 Cassandra T-Shirt |       1234 | 2015-03-29 12:02:49+0100 |  chbatey
 Cassandra T-Shirt |        902 | 2015-03-29 12:02:54+0100 |  chbatey
 Cassandra T-Shirt |        789 | 2015-03-29 12:02:58+0100 |  chbatey
```

Auction Bids

```
@PostConstruct
public void prepareStatements() {
    createAuction = session.prepare("insert INTO auctions (name, owner, ends) VALUES (?, ?, ?)");
    getAuction = session.prepare("select * from auctions where name = ?");
    getAllAuctionSparse = session.prepare("select * from auctions”);

    getAuctionBids = session.prepare("select * from auction_bids where name = ?");
    storeBid = session.prepare("INSERT INTO auction_bids (name, bid_time , bid_amount , bid_user) VALUES ( ?, ?, ?, ?);");
}
```

```
public List<Auction> getAllAuctionsSparse() {
    BoundStatement bound = getAllAuctionSparse.bind();
    return session.execute(bound).all().stream().map(row ->
            new Auction(row.getString("name"), row.getString("owner"), Instant.ofEpochMilli(row.getLong("ends"))))
            .collect(Collectors.toList());
}
```

Auction Bids

```
@PostConstruct
public void prepareStatements() {
    createAuction = session.prepare("insert INTO auctions (name, owner, ends) VALUES (?, ?, ?)");
    getAuction = session.prepare("select * from auctions where name = ?");
    getAllAuctionSparse = session.prepare("select * from auctions”);

    getAuctionBids = session.prepare("select * from auction_bids where name = ?");
    storeBid = session.prepare("INSERT INTO auction_bids (name, bid_time , bid_amount , bid_user) VALUES ( ?, ?, ?, ?);");
}
```

```
public Optional<Auction> getAuction(String auctionName) {
    BoundStatement auctionBoundStatement = getAuction.bind(auctionName);
    Row auction = session.execute(auctionBoundStatement).one();

    BoundStatement bidsBound = getAuctionBids.bind(auctionName);
    List<BidVo> bids = session.execute(bidsBound).all().stream().map(row ->
            new BidVo(row.getString("bid_user"),
                    row.getLong("bid_amount"),
                    UUIDs.unixTimestamp(row.getUUID("bid_time"))))
            .collect(Collectors.toList());

    return Optional.of(new Auction(auction.getString("name"),
            Instant.ofEpochMilli(auction.getLong("ends")),
            bids,
            auction.getString("owner")));
}
```

DATASTAX

Mapping API

Modelling in Cassandra

```
CREATE TABLE customer_events(
    customer_id text,
    staff_id text,
    time timeuuid,
    store_type text,
    event_type text,
    tags map<text, text>,
    PRIMARY KEY ((customer_id), time));
```

Mapping API

```
@Table(keyspace = "customers", name = "customer_events")
public class CustomerEvent {
    @PartitionKey
    @Column(name = "customer_id")
    private String customerId;

    @ClusteringColumn
    private UUID time;

    @Column(name = "staff_id")
    private String staffId;

    @Column(name = "store_type")
    private String storeType;

    @Column(name = "event_type")
    private String eventType;

    private Map<String, String> tags;
    // ctr / getters etc
}
```

Mapping API

```
@Accessor
public interface CustomerEventDao {
    @Query("select * from customers.customer_events where customer_id = :customerId")
    Result<CustomerEvent> getCustomerEvents(String customerId);

    @Query("select * from customers.customer_events")
    Result<CustomerEvent> getAllCustomerEvents();

}


@Bean
public CustomerEventDao customerEventDao() {
    MappingManager mappingManager = new MappingManager(session);
    return mappingManager.createAccessor(CustomerEventDao.class);
}
```

Adding some type safety

```
public enum StoreType {
    ONLINE, RETAIL, FRANCHISE, MOBILE
}
```

```
@Table(keyspace = "customers", name = "customer_events")
public class CustomerEvent {
    @PartitionKey
    @Column(name = "customer_id")
    private String customerId;

    @ClusteringColumn()
    private UUID time;

    @Column(name = "staff_id")
    private String staffId;

    @Column(name = "store_type")
    @Enumerated(EnumType.STRING) // could be EnumType.ORDINAL
    private StoreType storeType;
```

User defined types

```
create TYPE store (name text, type text, postcode text) ;


CREATE TABLE customer_events_type(
 customer_id text,
 staff_id text,
 time timeuuid,
 store frozen<store>,
 event_type text,
 tags map<text, text>,
 PRIMARY KEY ((customer_id), time));
```

Mapping user defined types

```
@UDT(keyspace = "customers", name = "store")
public class Store {
    private String name;
    private StoreType type;
    private String postcode;
    // getters etc
 }

@Table(keyspace = "customers", name = "customer_events_type")
public class CustomerEventType {
    @PartitionKey
    @Column(name = "customer_id")
    private String customerId;

    @ClusteringColumn()
    private UUID time;

    @Column(name = "staff_id")
    private String staffId;

    @Frozen
    private Store store;

    @Column(name = "event_type")
    private String eventType;

    private Map<String, String> tags;
```

Mapping user defined types

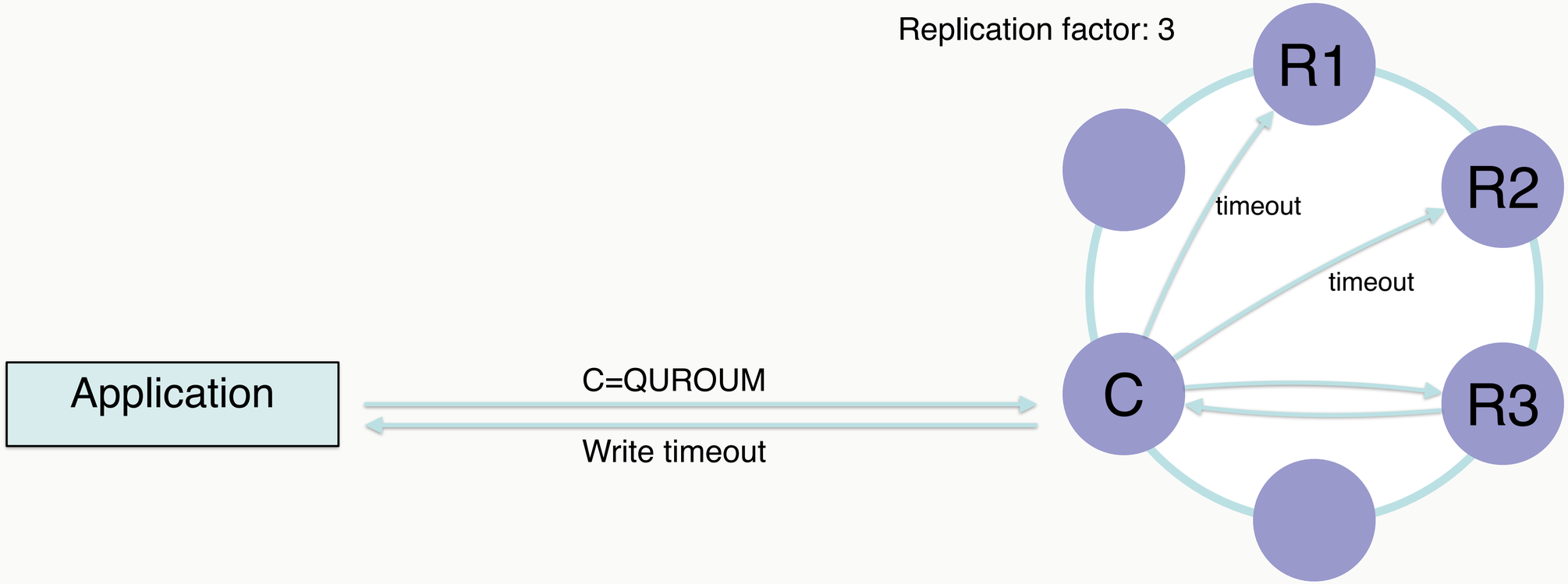
```
@UDT(keyspace = "customers", name = "store")
public class Store {
    private String name;
    private StoreType type;
    private String postcode;
    // getters etc
 }

@Table(keyspace = "customers", name = "customer_events_type")
public class CustomerEventType {
    @PartitionKey
    @Column(name = "customer_id")
    private String customerId;

    @ClusteringColumn()
    private UUID time;

    @Column(name = "staff_id")
    private String staffId;

    @Frozen
    private Store store;

    @Column(name = "event_type")
    private String eventType;

    private Map<String, String> tags;
```

Dev & Test environments

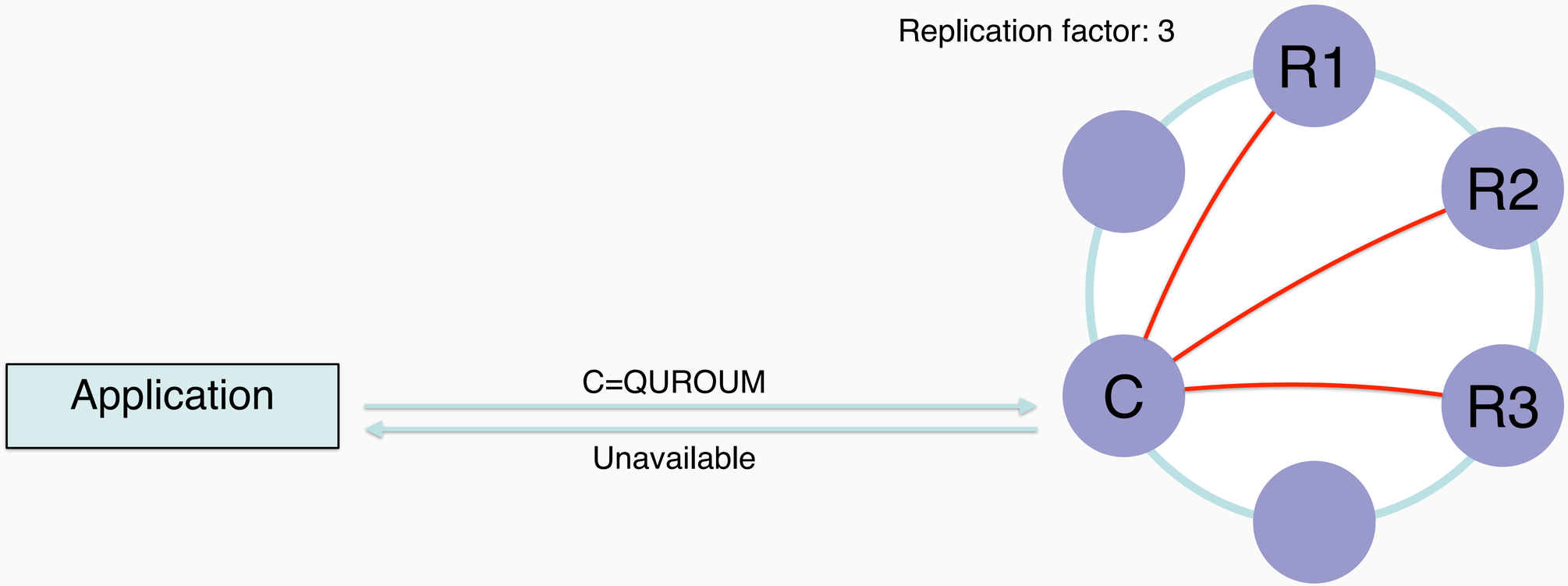
Development Env + Testing

- Cassandra Unit
  - Embeds a single Cassandra node in the same JVM for unit testing
    - https://github.com/jsevellec/cassandra-unit
- Cassandra Cluster Manager
  - Spins up small clusters locally
    - https://github.com/pcmanus/ccm
- Stubbed Cassandra
  - Pretends to be a real Cassandra

Write timeout



Unavailable



Example Cassandra Unit with JUnit rule

```
<dependency>
   <groupId>org.cassandraunit</groupId>
   <artifactId>cassandra-unit</artifactId>
   <version>2.1.3.1</version>
   <scope>test</scope>
</dependency>
```

```
@Rule
public CassandraUnit cassandraUnitRule = new CassandraUnit(new
ClassPathXmlDataSet("extendedDataSet.xml"));

@Test
public void shouldHaveLoadAnExtendDataSet() throws Exception {
    // test dao
}
```

Cassandra cluster manager

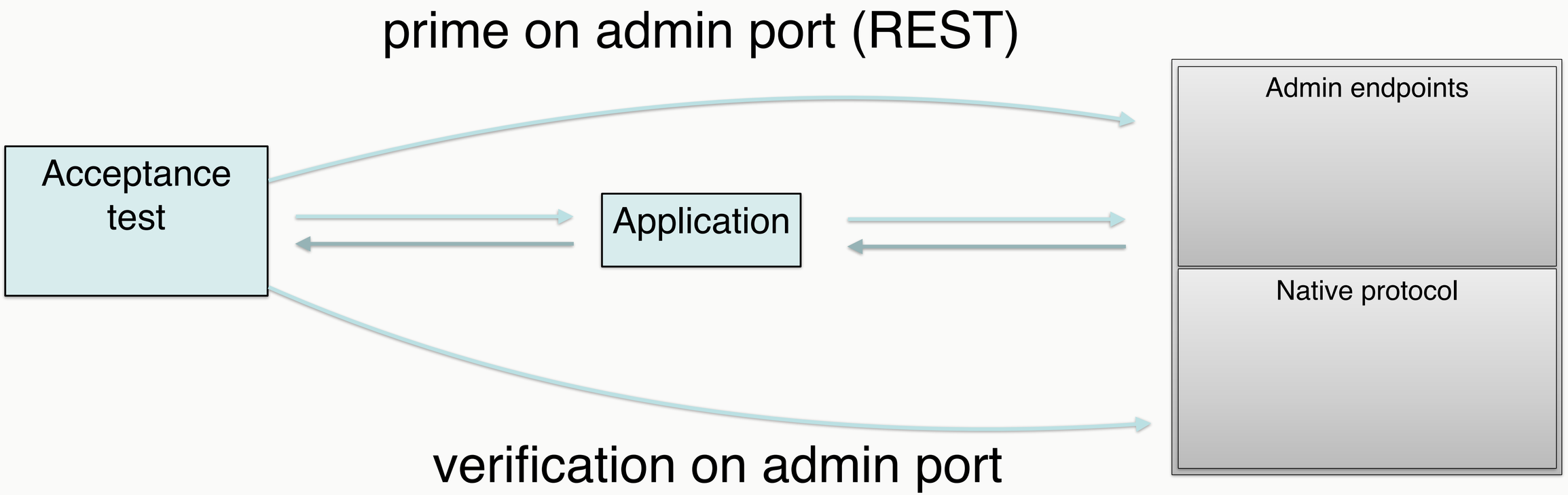
ccm create test -v 2.0.5 -n 6 -s

```
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns    Host ID                               Rack
UN  127.0.0.1  102.27 KB  256     ?       15ad7694-3e76-4b74-aea0-fa3c0fa59532  rack1
UN  127.0.0.2  102.18 KB  256     ?       cca7d0bb-e884-49f9-b098-e38fbe895cbc  rack1
UN  127.0.0.3  93.16 KB   256     ?       1f9737d3-c1b8-4df1-be4c-d3b1cced8e30  rack1
UN  127.0.0.4  102.1 KB   256     ?       fe27b958-5d3a-4f78-9880-76cb7c9bead1  rack1
UN  127.0.0.5  93.18 KB   256     ?       66eb3f23-8889-44d6-a9e7-ecdd57ed61d0  rack1
UN  127.0.0.6  102.12 KB  256     ?       e2e99a7b-c1fb-4f2a-9e4f-7a4666f8245e  rack1
```

Trace

```
Execute CQL3 query | 2015-02-02 06:39:58.759000 | 127.0.0.1 |              0
Parsing select * from staff where name in ('chbatey', 'luket', 'jonh'); [SharedPool-Worker-1] | 2015-02-02
06:39:58.766000 | 127.0.0.1 |           7553
Preparing statement [SharedPool-Worker-1] | 2015-02-02 06:39:58.768000 | 127.0.0.1 |           9249
Executing single-partition query on staff [SharedPool-Worker-3] | 2015-02-02 06:39:58.773000 | 127.0.0.1 |          14255
Sending message to /127.0.0.3 [WRITE-/127.0.0.3] | 2015-02-02 06:39:58.773001                   | 127.0.0.1 |          14756
Sending message to /127.0.0.5 [WRITE-/127.0.0.5] | 2015-02-02 06:39:58.773001                   | 127.0.0.1 |          14928
Sending message to /127.0.0.3 [WRITE-/127.0.0.3] | 2015-02-02 06:39:58.774000                   | 127.0.0.1 |          16035
Executing single-partition query on staff [SharedPool-Worker-1] | 2015-02-02 06:39:58.777000 | 127.0.0.5 |           1156
Enqueuing response to /127.0.0.1 [SharedPool-Worker-1] | 2015-02-02 06:39:58.777001           | 127.0.0.5 |           1681
Sending message to /127.0.0.1 [WRITE-/127.0.0.1] | 2015-02-02 06:39:58.778000                   | 127.0.0.5 |           1944
Executing single-partition query on staff [SharedPool-Worker-1] | 2015-02-02 06:39:58.778000 | 127.0.0.3 |           1554
Processing response from /127.0.0.5 [SharedPool-Worker-3] | 2015-02-02 06:39:58.779000         | 127.0.0.1 |          20762
Enqueuing response to /127.0.0.1 [SharedPool-Worker-1] | 2015-02-02 06:39:58.779000           | 127.0.0.3 |           2425
Sending message to /127.0.0.5 [WRITE-/127.0.0.5] | 2015-02-02 06:39:58.779000                   | 127.0.0.1 |          21198
Sending message to /127.0.0.1 [WRITE-/127.0.0.1] | 2015-02-02 06:39:58.779000                   | 127.0.0.3 |           2639
Sending message to /127.0.0.6 [WRITE-/127.0.0.6] | 2015-02-02 06:39:58.779000                   | 127.0.0.1 |          21208
Executing single-partition query on staff [SharedPool-Worker-1] | 2015-02-02 06:39:58.780000 | 127.0.0.5 |            304
Enqueuing response to /127.0.0.1 [SharedPool-Worker-1] | 2015-02-02 06:39:58.780001           | 127.0.0.5 |            574
Executing single-partition query on staff [SharedPool-Worker-2] | 2015-02-02 06:39:58.781000 | 127.0.0.3 |           4075
Sending message to /127.0.0.1 [WRITE-/127.0.0.1] | 2015-02-02 06:39:58.781000                   | 127.0.0.5 |            708
Enqueuing response to /127.0.0.1 [SharedPool-Worker-2] | 2015-02-02 06:39:58.781001           | 127.0.0.3 |           4348
Sending message to /127.0.0.1 [WRITE-/127.0.0.1] | 2015-02-02 06:39:58.782000                   | 127.0.0.3 |           5371
Executing single-partition query on staff [SharedPool-Worker-1] | 2015-02-02 06:39:58.783000 | 127.0.0.6 |           2463
Enqueuing response to /127.0.0.1 [SharedPool-Worker-1] | 2015-02-02 06:39:58.784000           | 127.0.0.6 |           2905
Sending message to /127.0.0.1 [WRITE-/127.0.0.1] | 2015-02-02 06:39:58.784001                   | 127.0.0.6 |           3160
Processing response from /127.0.0.6 [SharedPool-Worker-2] | 2015-02-02 06:39:58.785000         | 127.0.0.1 |             --
Request complete | 2015-02-02 06:39:58.782995                                                     | 127.0.0.1 |          23995
```

Stubbed Cassandra



Starting / Stopping

```
Scassandra scassandra = ScassandraFactory.createServer();

scassandra.start();

PrimingClient primingClient = scassandra.primingClient();

ActivityClient activityClient = scassandra.activityClient();
```

```
@ClassRule
public static final ScassandraServerRule SCASSANDRA = new
ScassandraServerRule();
```

Activity Client

```
public List<Query> retrieveQueries();

public List<PreparedStatementExecution> retrievePreparedStatementExecutions()

public List<Connection> retrieveConnections();

public void clearAllRecordedActivity()

public void clearConnections();

public void clearQueries();

public void clearPreparedStatementExecutions();
```

- Query
  - Query text
  - Consistency
- PrepreparedStatementExecution
  - Prepared statement text
  - Bound variables

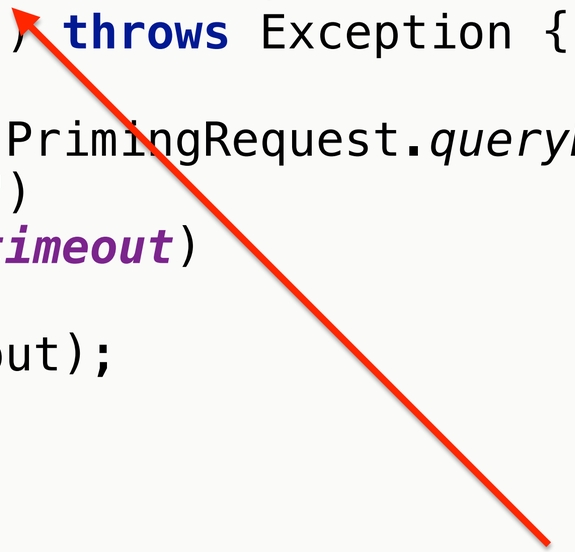
Priming Client

```
public void prime(PrimingRequest prime)

public List<PrimingRequest> retrievePreparedPrimes()
public List<PrimingRequest> retrieveQueryPrimes()

public void clearAllPrimes()
public void clearQueryPrimes()
public void clearPreparedPrimes()
```

- PrimingRequest
  - Either a Query or PreparedStatement
  - Query text or QueryPattern (regex)
  - Consistency (default all)
  - Result (success, read timeout, unavailable etc)
  - Rows for successful response
  - Column types for rows
  - Variable types for prepared statements

Testing errors

```
@Test(expected = UnableToRetrievePeopleException.class)
public void testHandlingOfReadRequestTimeout() throws Exception {
    // given
    PrimingRequest primeReadRequestTimeout = PrimingRequest.queryBuilder()
            .withQuery("select * from person")
            .withResult(Result.read_request_timeout)
            .build();
    primingClient.prime(primeReadRequestTimeout);

    //when
    underTest.retrievePeople();

    //then
}
```



Expecting custom exception

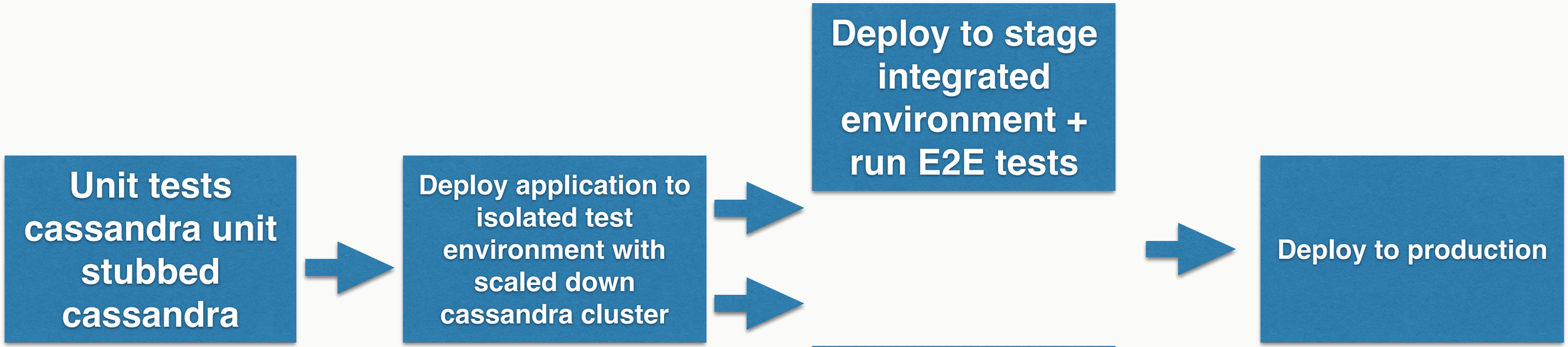
Testing slow connection

```
@Test(expected = UnableToSavePersonException.class)
public void testThatSlowQueriesTimeout() throws Exception {
    // given
    PrimingRequest preparedStatementPrime = PrimingRequest.preparedStatementBuilder()
            .withQueryPattern("insert into person.*")
            .withVariableTypes(VARCHAR, INT, list(TIMESTAMP))
            .withFixedDelay(1000)
            .build();
    primingClient.prime(preparedStatementPrime);
    underTest.connect();

    underTest.storePerson(new Person("Christopher", 29, Collections.emptyList()));
}
```

Expect a custom exception

Delays the response by 1000ms

Continuous integration

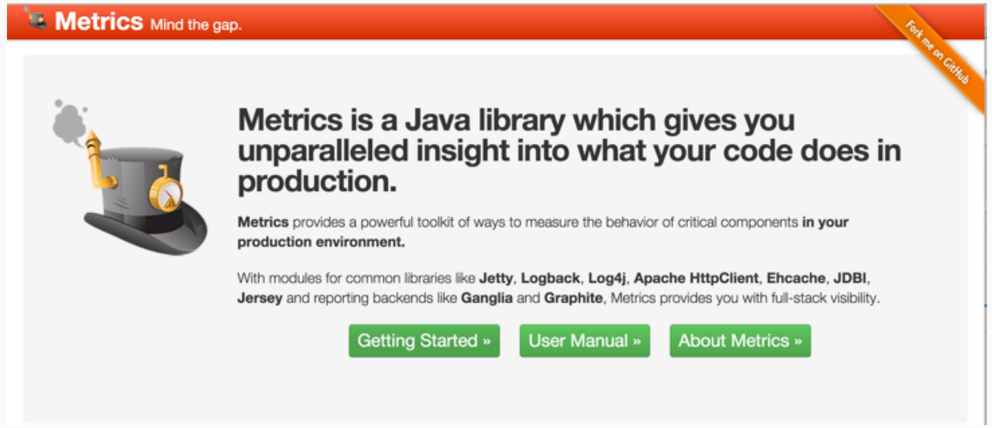


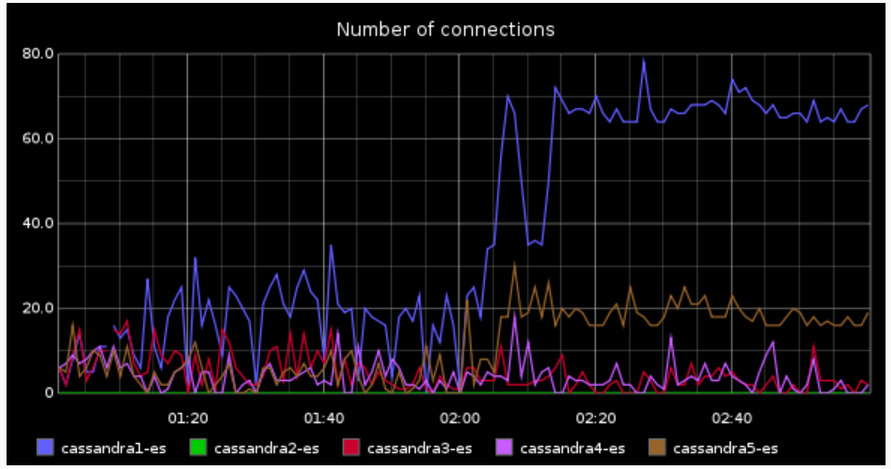
Cassandra Stress

```
### DML ###

# Keyspace Name
keyspace: stresscql

# The CQL for creating a keyspace (optional if it already exists)
keyspace_definition: |
  CREATE KEYSPACE stresscql WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

# Table name
table: blogposts

# The CQL for creating a table you wish to stress (optional if it already exists)
table_definition: |
  CREATE TABLE blogposts (
        domain text,
        published_date timeuuid,
        url text,
        author text,
        title text,
        body text,
        PRIMARY KEY(domain, published_date)
  ) WITH CLUSTERING ORDER BY (published_date DESC);
    AND compaction = { 'class':'LeveledCompactionStrategy' }
    AND comment='A table to hold blog posts'
```

```
columnspec:
  - name: domain
    size: gaussian(5..100)       #domain names are relatively short
    population: uniform(1..10M)  #10M possible domains to pick from

  - name: published_date
    cluster: fixed(1000)         #under each domain we will have max 1000 posts

  - name: url
    size: uniform(30..300)

  - name: title                  #titles shouldn't go beyond 200 chars
    size: gaussian(10..200)

  - name: author
    size: uniform(5..20)         #author names should be short

  - name: body
    size: gaussian(100..5000)    #the body of the blog post can be long
```

Monitoring - Dropwizard metrics





Metric	Description
connectedToHosts	A gauge that presents the number of hosts that are currently connected to (at least once).
errorMetrics	An Error.Metrics object which groups errors which have occurred.
knownHosts	A gauge that presents the number of nodes known by the driver, regardless of whether they are currently up or down).
openConnections	A gauge that presents the total number of connections to nodes.
requestsTimer	A timer that presents metrics on requests executed by the user.

Summary

- Know the eco-system
  - Community: Get on IRC / JIRA
  - Tools: DataStax drivers, CCM, Cassandra-unit, DevCenter
- Get used to looking at trace

Thanks for listening

- Follow me on twitter @chbatey
- Cassandra + Fault tolerance posts a plenty:
  - http://christopher-batey.blogspot.co.uk/
- Cassandra resources: http://planetcassandra.org/