LJC: Building fault tolerant Java applications with Apache Cassandra

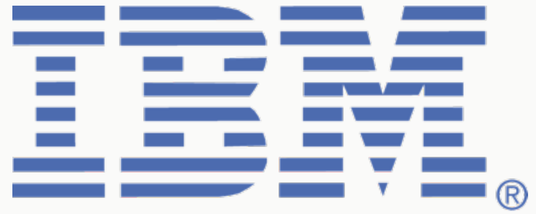
Christopher Batey

@chbatey

Who am I?

DATASTAX

- 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



Overview

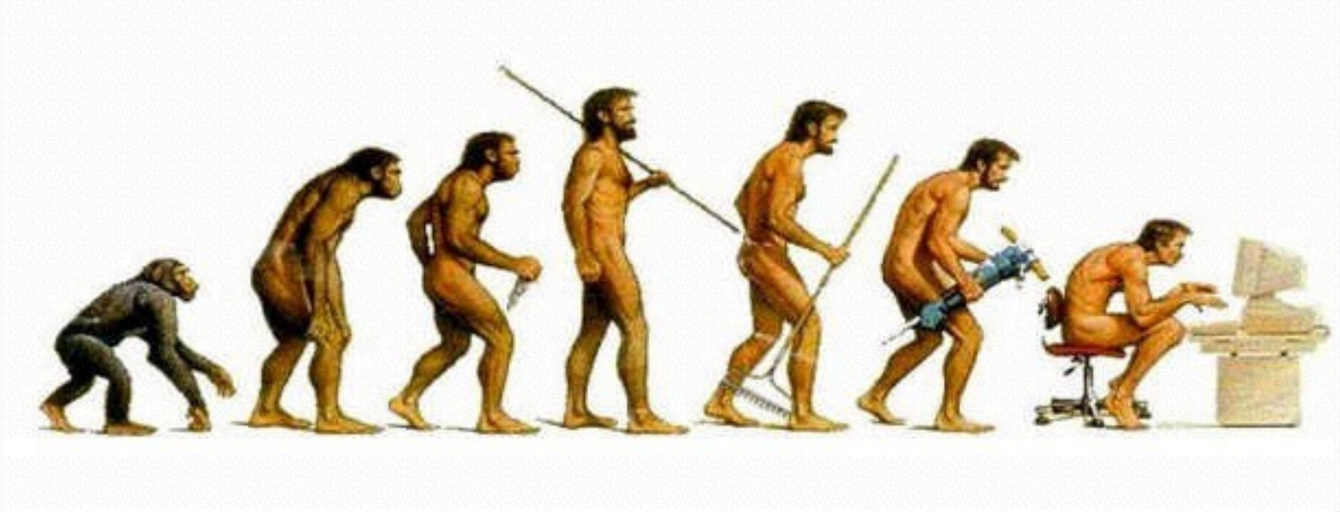
DATASTAX

- ‘Building fault tolerant Java applications with Apache Cassandra’
  - Fault tolerance?
  - Building an ‘always on’ (HA) service
  - Cassandra architecture overview
  - Cassandra failure scenarios
  - Cassandra Java Driver

The evolution of a developer

DATASTAX

- A few years ago I used to write a lot of code in an IDE
- Recently not so much:
  - Config management (Ansible, Puppet etc)
  - Solution architecture
- Frequently bootstrapping new services



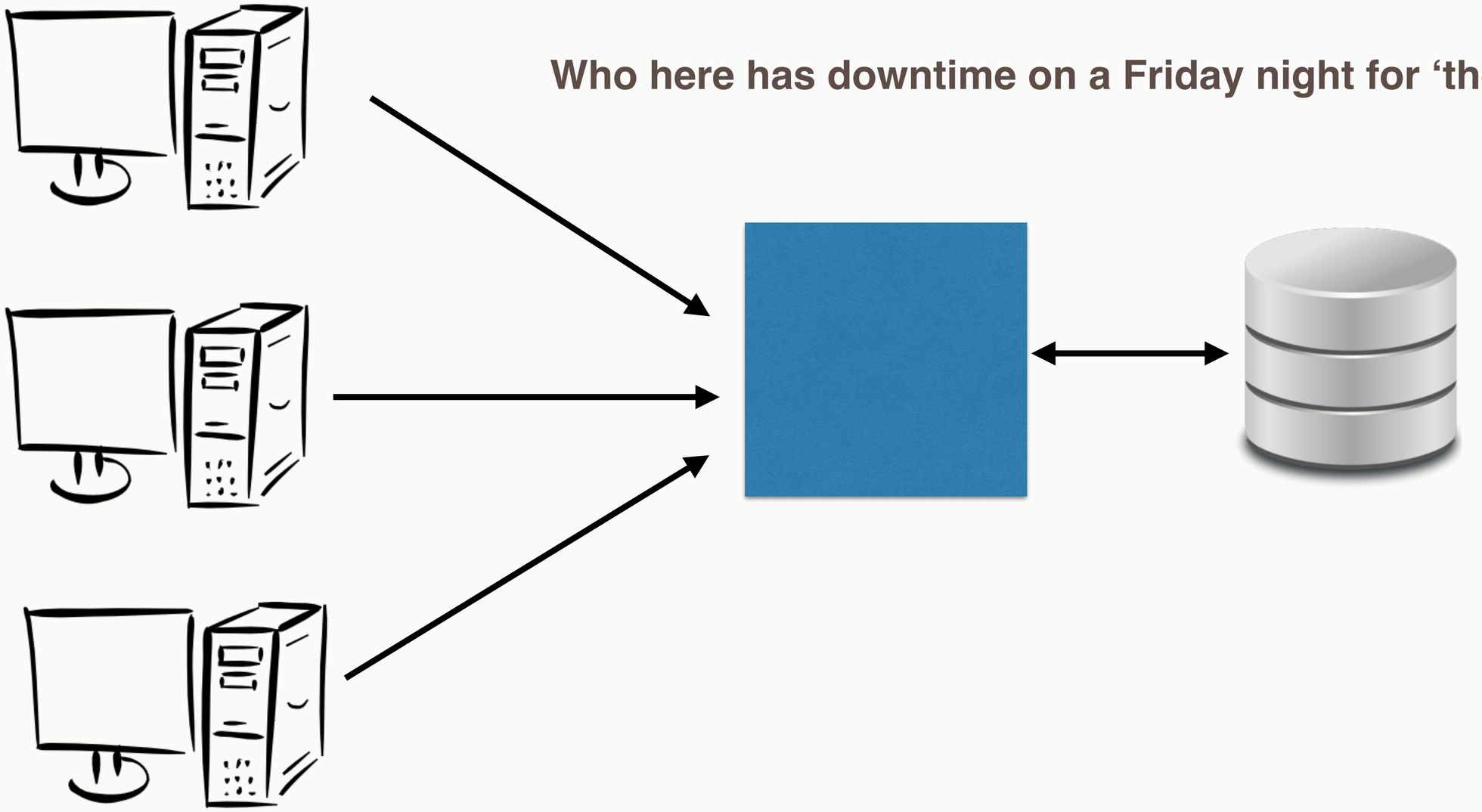

BARCELONA VS SEVILLA
STADIUM: CAMP NOU
TIME: 20:00
VALDES
PIQUE
MASCHERANO
ALVES
ALBA
BUSQUETS
XAVI
INIESTA
TELLO
NEYMAR
BARCA

Faults?

- Infrastructure failure
  - Node
  - Rack
  - Data center
- Dependency failure
  - HTTP services
  - Databases

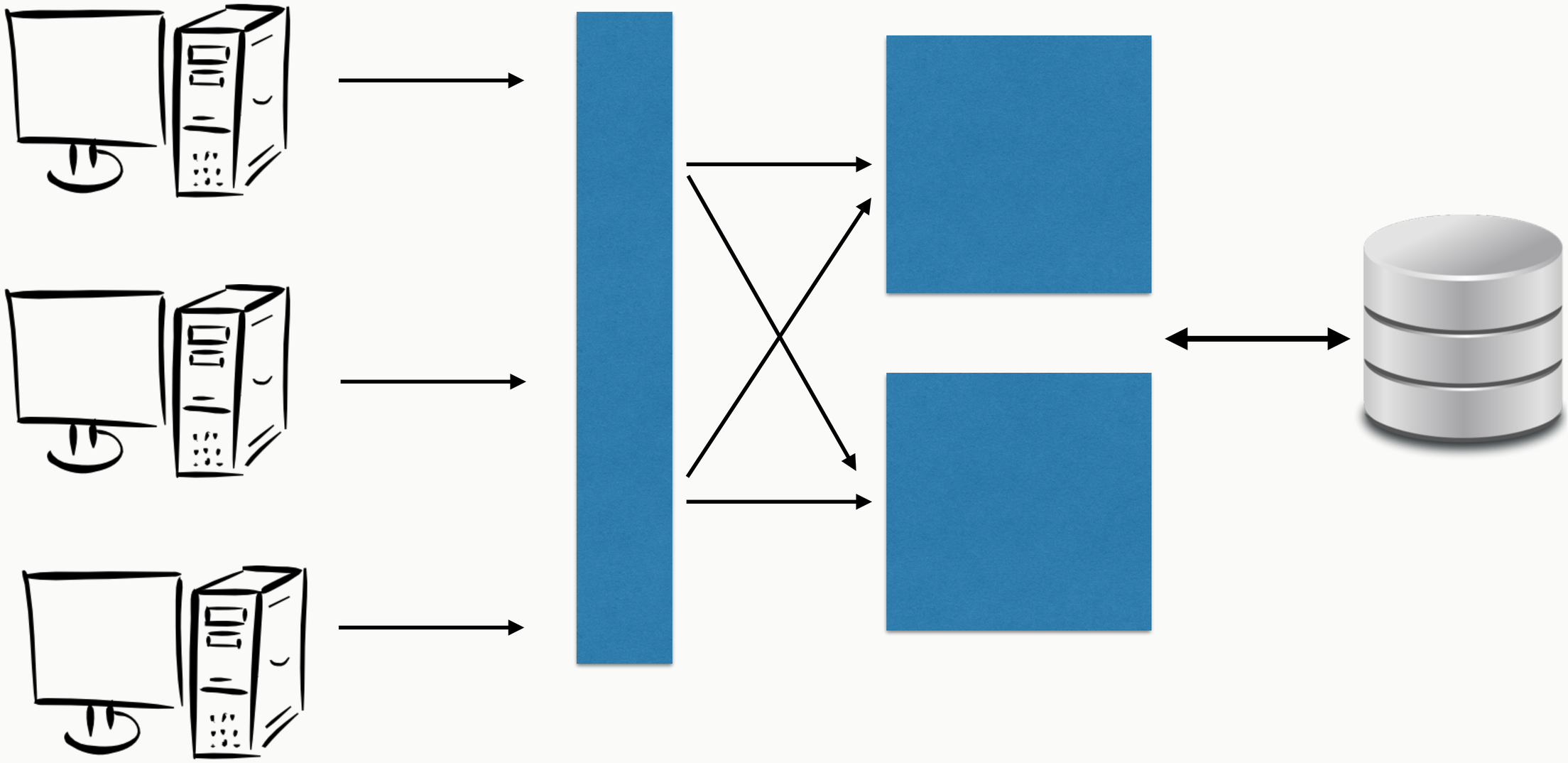
Building a web app

DATASTAX

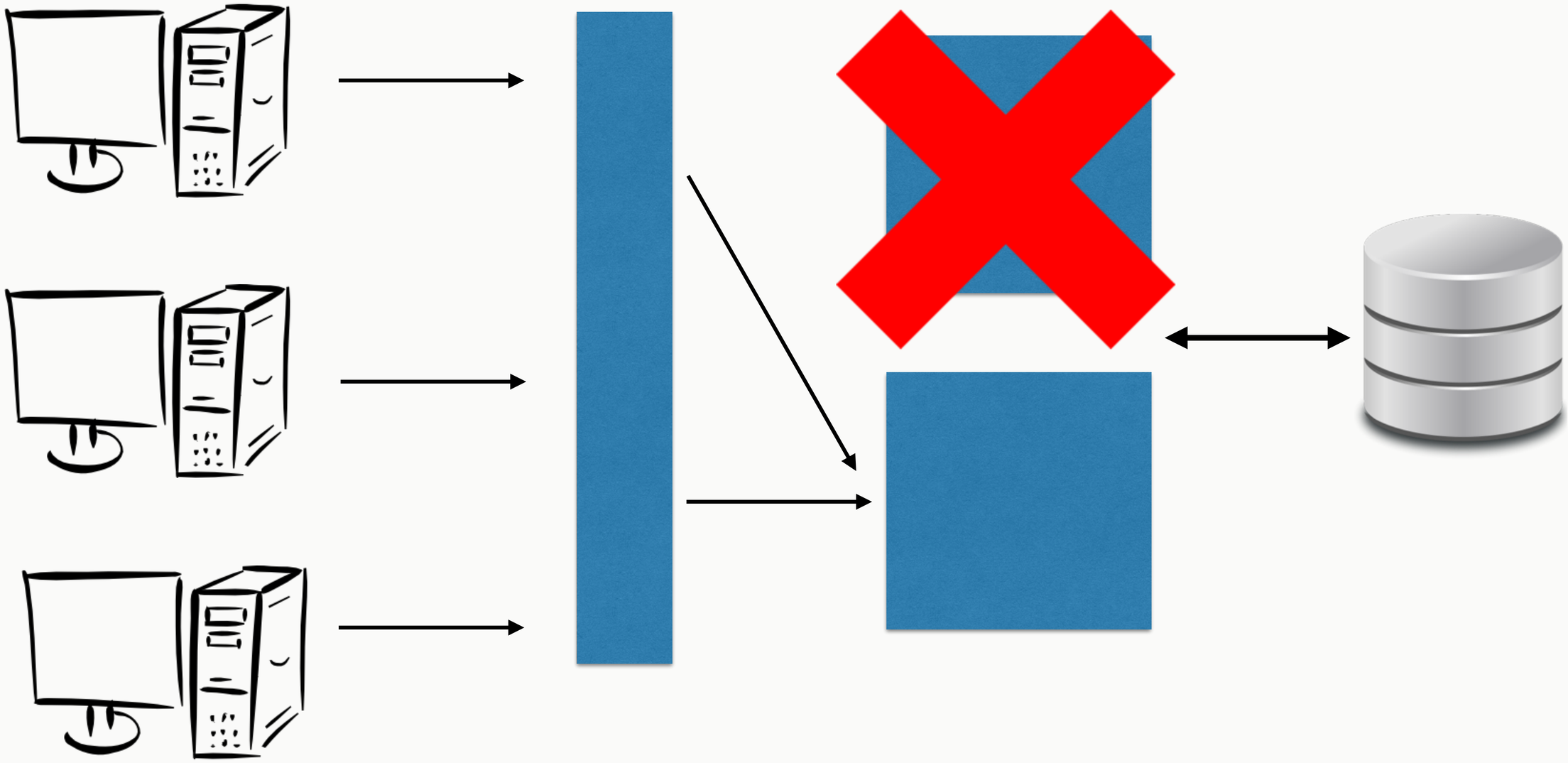


Running multiple copies of your app

DATASTAX



So when one dies…

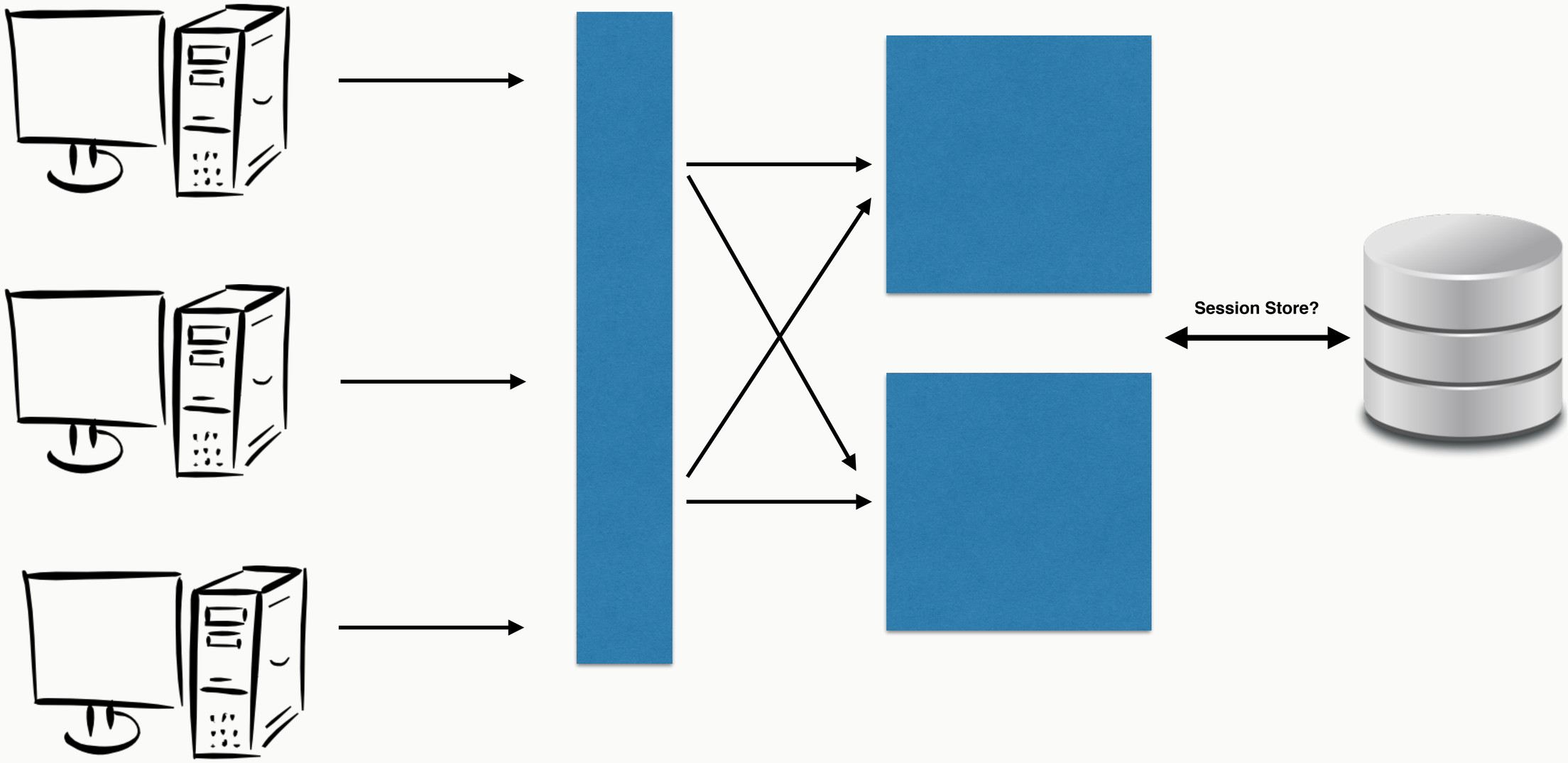


Thou shalt be stateless…

DATASTAX

- Using container session store = 1/nth of your customers unhappy if a server dies

Storing state in a database?

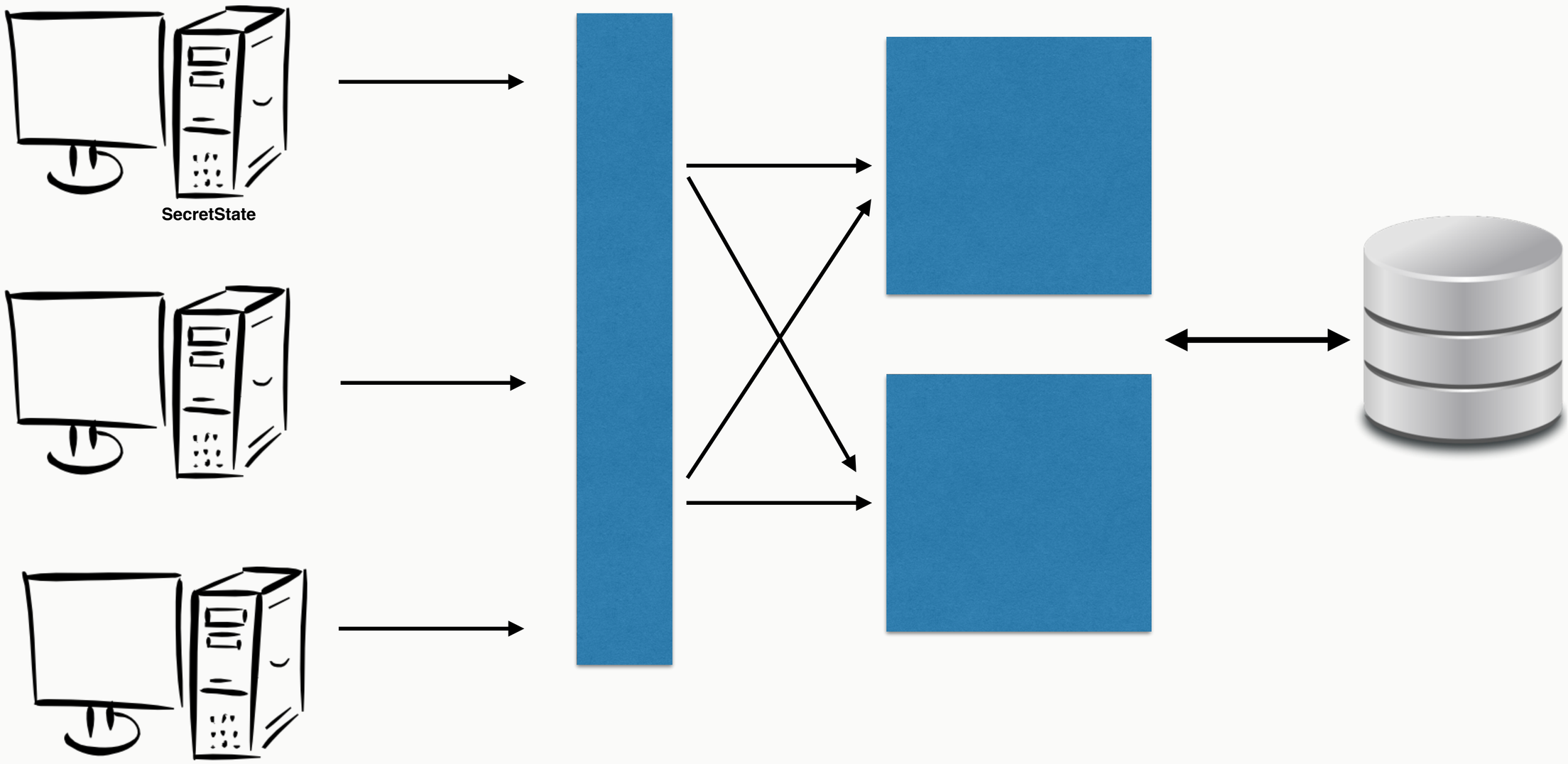


The drawbacks

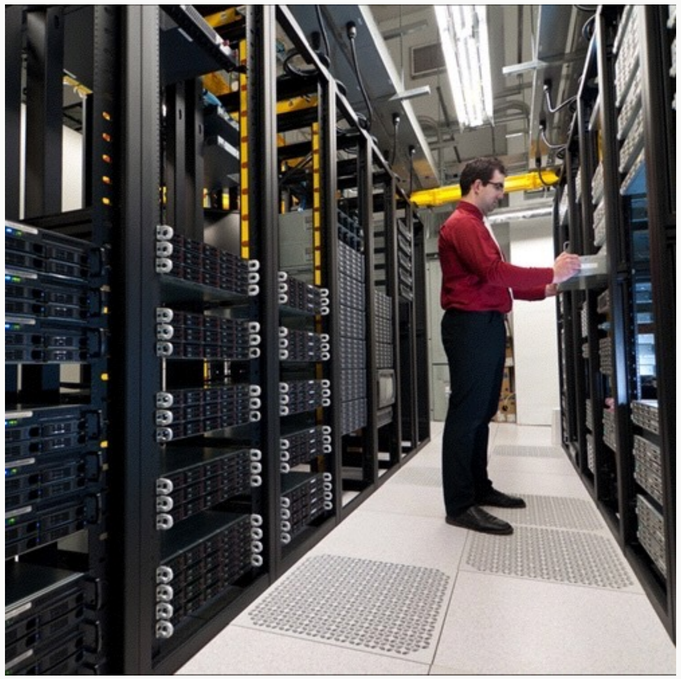
DATASTAX

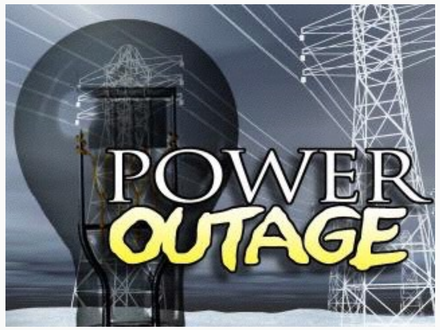
- Latency :(
- Tricky to setup
- New DB/Cache = new component in your architecture to scale and make HA
- Tightly coupled to your container now
- Move to embedded container in executable jars :(

Client side state?



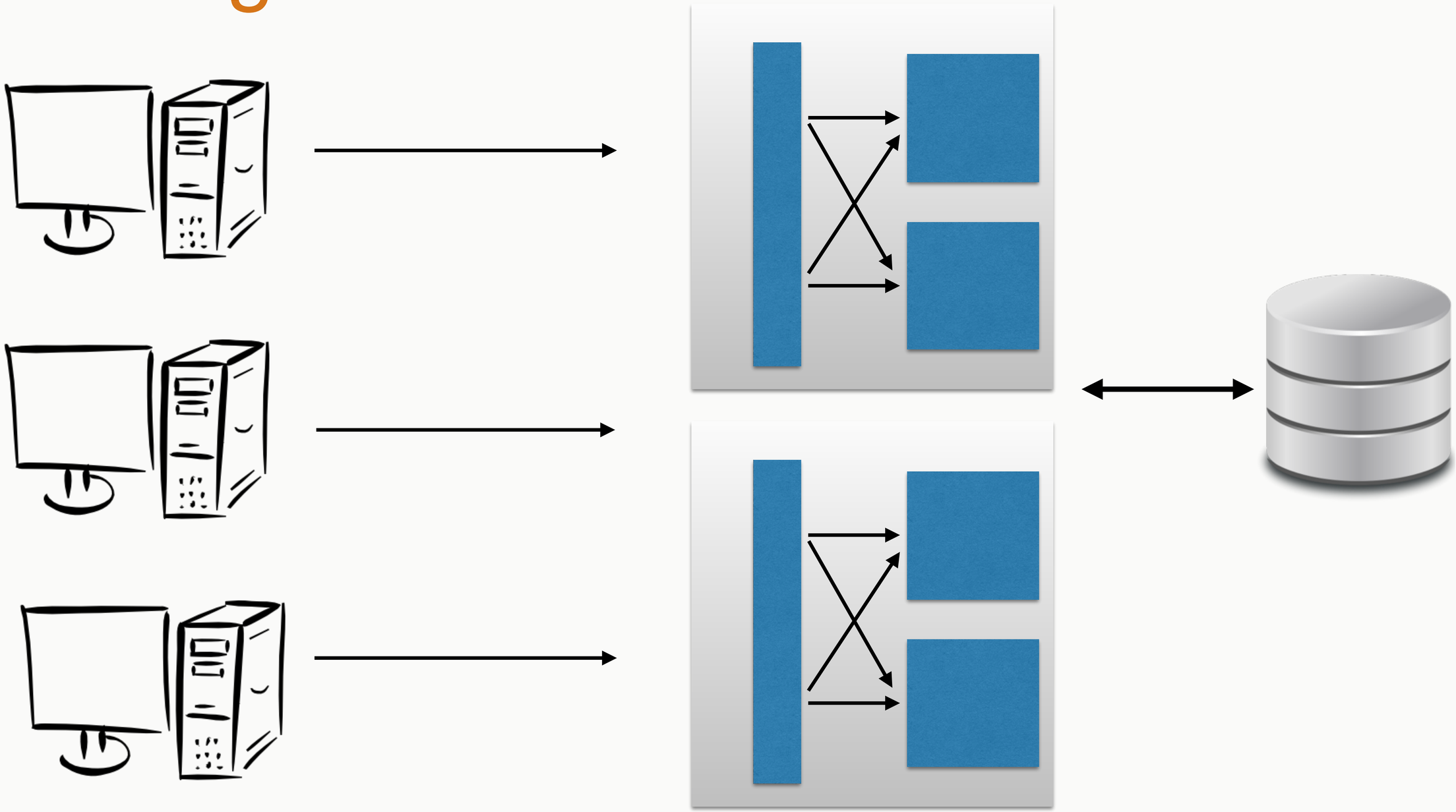
Still in one DC?



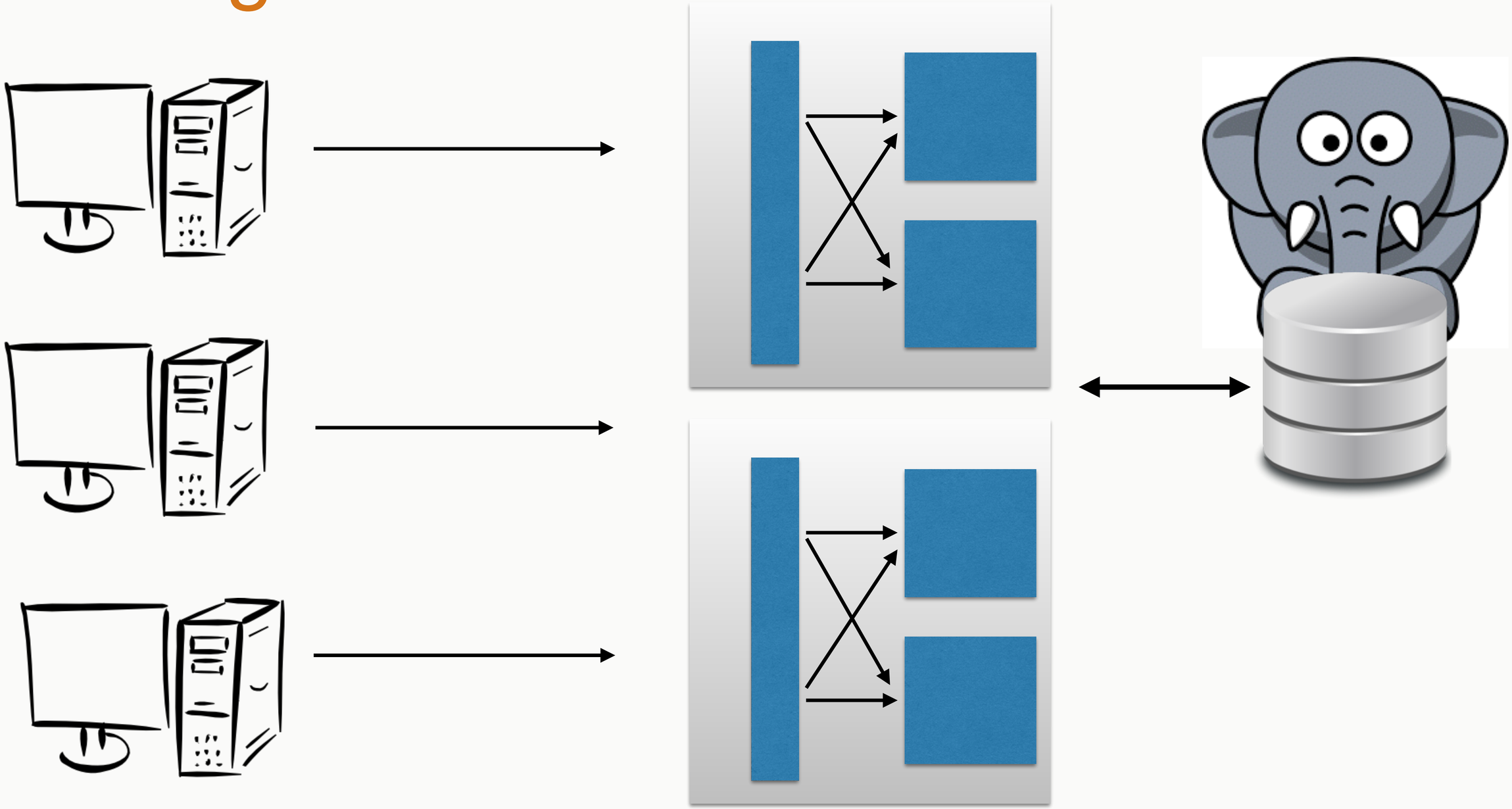


@chbatey

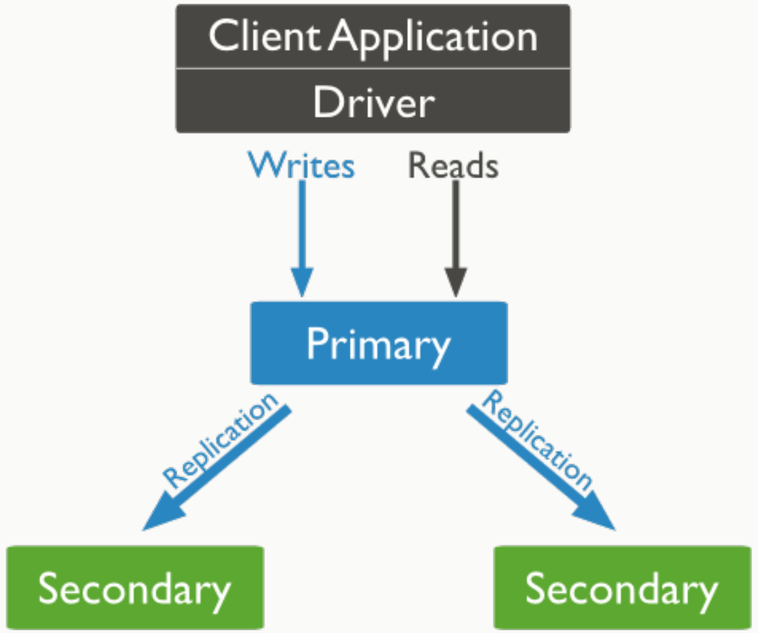
Handling hardware failure

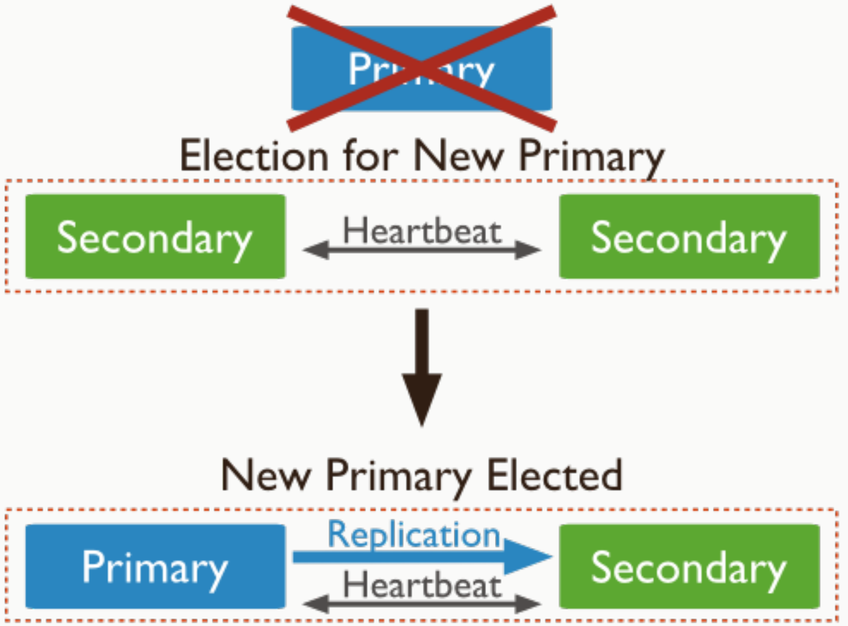


Handling hardware failure



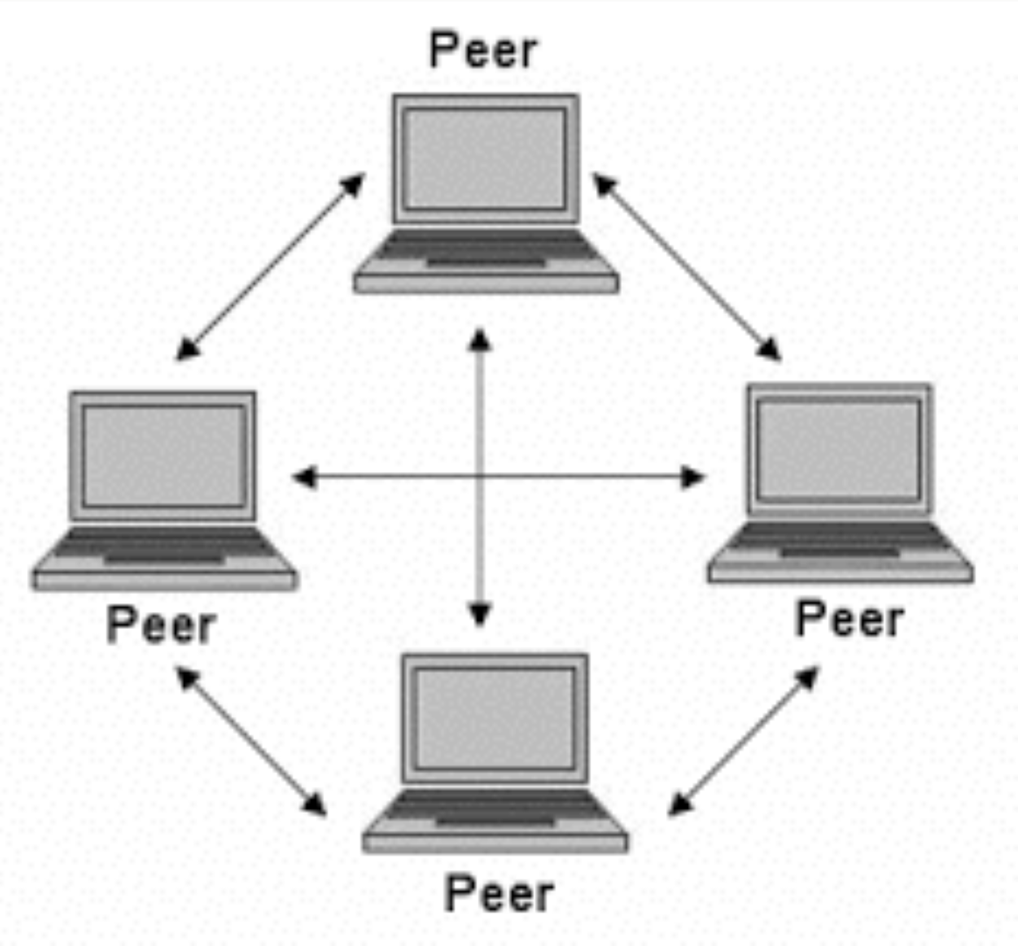
Master/slave





- Master serves all writes
- Read from master and optionally slaves

Peer-to-Peer

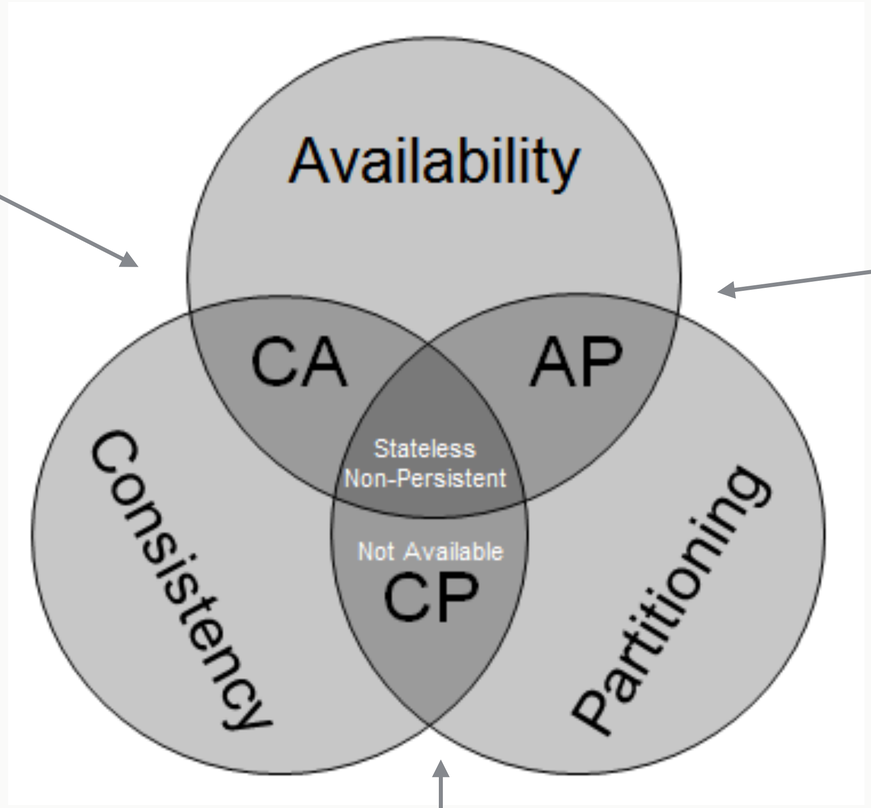


- No master
- Read/write to any
- Consistency?

Decisions decisions… CAP theorem

DATASTAX

Relational Database



Highly Available Databases: Voldermort, Cassandra

Are these really that different??

Mongo, Redis

PAC-ELC

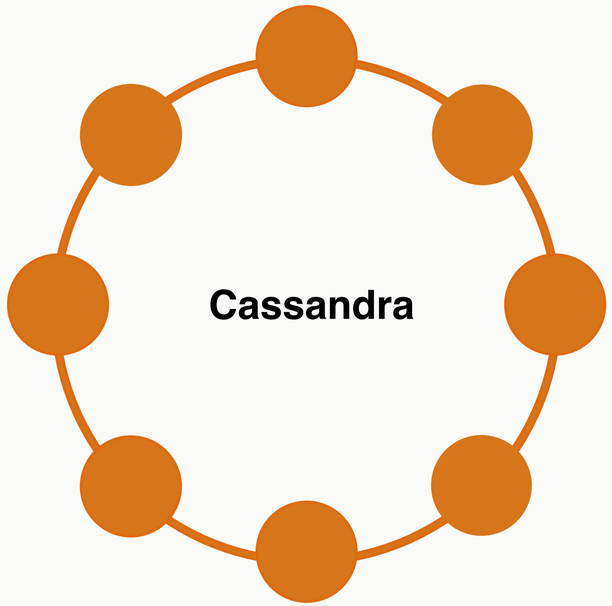
DATASTAX

- If **P**artition: Trade off **A**vailability vs **C**onsistency
- **E**lse: Trade off **L**atency vs **C**onsistency
- http://dbmsmusings.blogspot.co.uk/2010/04/problems-with-cap-and-yahoos-little.html

DATASTAX

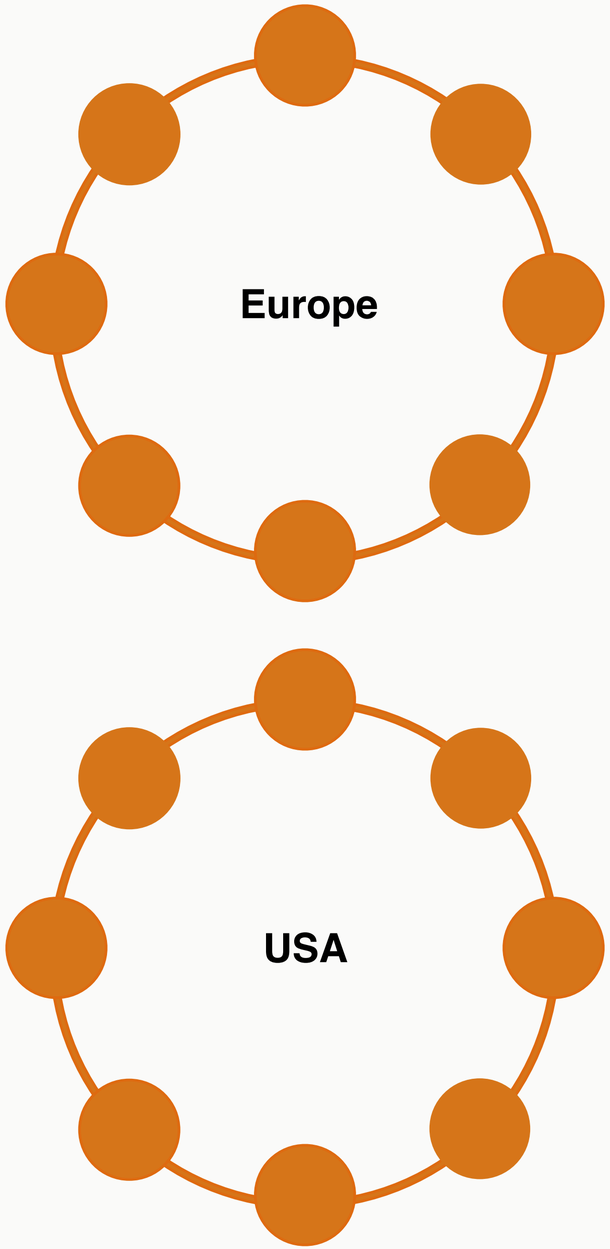
Cassandra

Cassandra



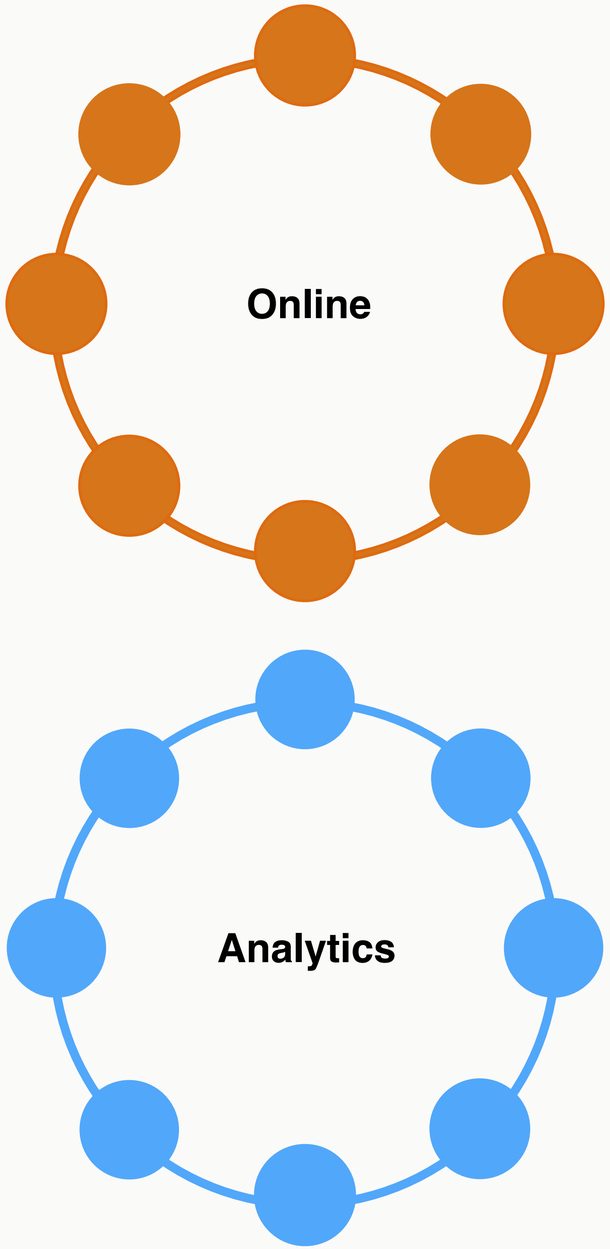
- Distributed masterless database (Dynamo)
- Column family data model (Google BigTable)

Datacenter and rack aware



- Distributed master less database (Dynamo)
- Column family data model (Google BigTable)
- Multi data centre replication built in from the start

Cassandra



- Distributed master less database (Dynamo)
- Column family data model (Google BigTable)
- Multi data centre replication built in from the start
- Analytics with Apache Spark

Dynamo 101

DATASTAX

- The parts Cassandra took
  - Consistent hashing
  - Replication
  - Strategies for replication
  - Gossip
  - Hinted handoff
  - Anti-entropy repair
- And the parts it left behind
  - Key/Value
  - Vector clocks

Picking the right nodes

DATASTAX

- You don’t want a full table scan on a 1000 node cluster!
- Dynamo to the rescue: Consistent Hashing
- Then the replication strategy takes over:
  - Network topology
  - Simple

Murmer3 Example

DATASTAX

- **Data:**

jim	age: 36	car: ford	gender: M
carol	age: 37	car: bmw	gender: F
johnny	age: 12	gender: M	
suzy:	age: 10	gender: F	

- **Murmer3 Hash Values:**

Primary Key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

Murmer3 Example

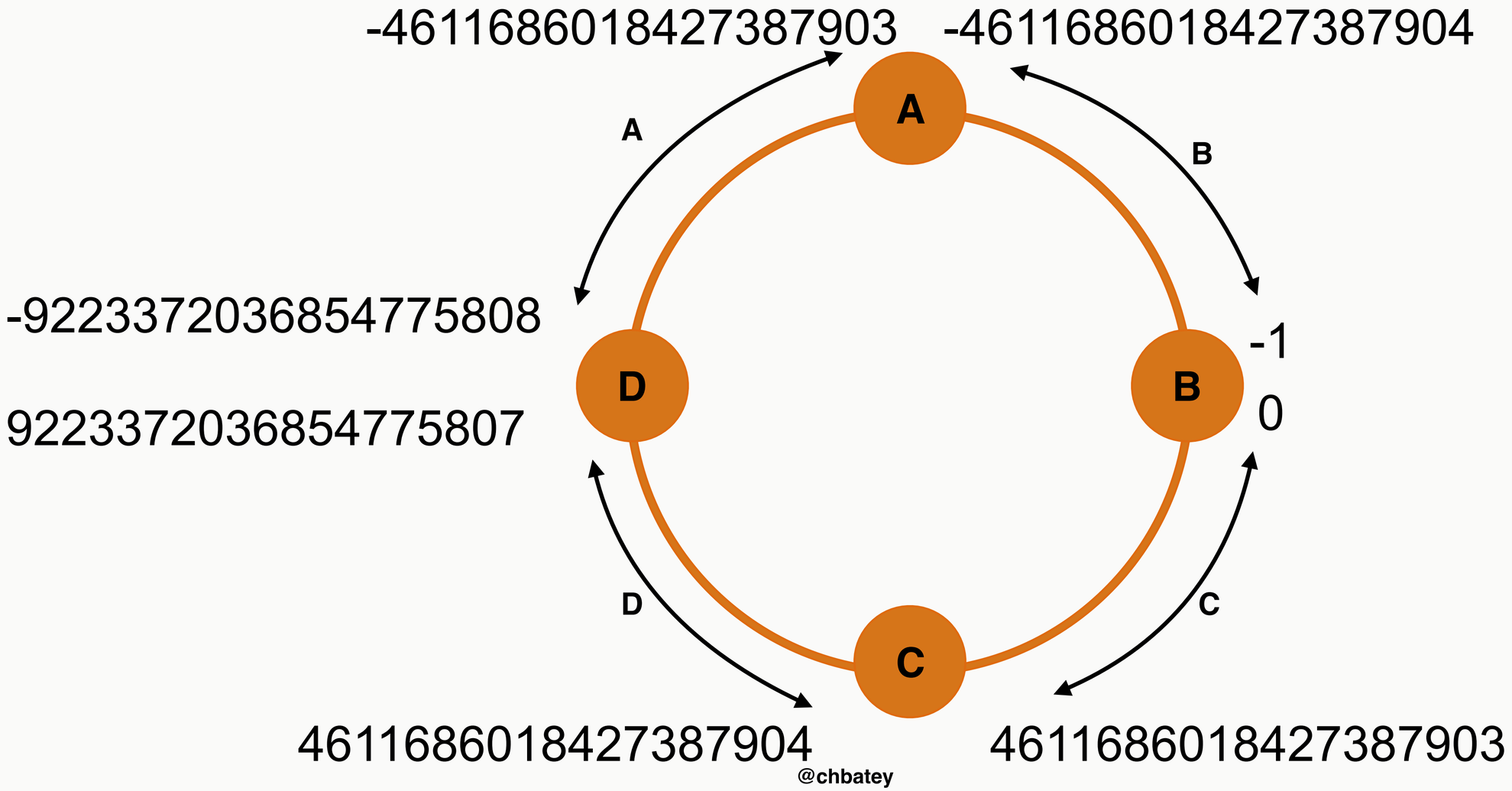
DATASTAX

Four node cluster:

Node	Murmur3 start range	Murmur3 end range
A	-9223372036854775808	-4611686018427387903
B	-4611686018427387904	-1
C	0	4611686018427387903
D	4611686018427387904	9223372036854775807

Pictures are better

Hash range: -9223372036854775808 to 9223372036854775807



Murmer3 Example

DATASTAX

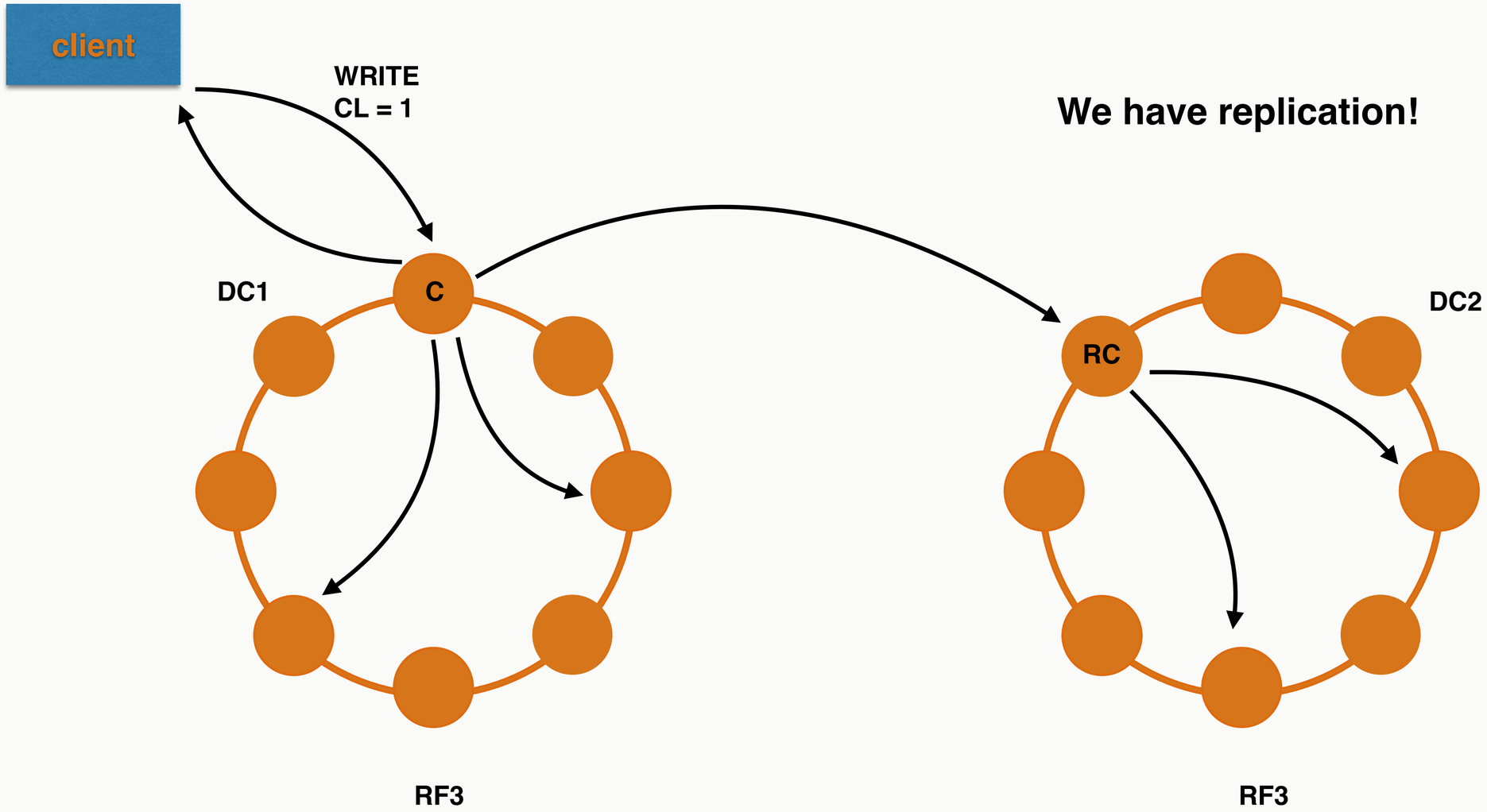
Data is distributed as:

Node	Start range	End range	Primary key	Hash value
A	-9223372036854775808	-4611686018427387903	johnny	-6723372854036780875
B	-4611686018427387904	-1	jim	-2245462676723223822
C	0	4611686018427387903	suzy	1168604627387940318
D	4611686018427387904	9223372036854775807	carol	7723358927203680754

DATASTAX

Replication

Replication



Replication strategy

DATASTAX

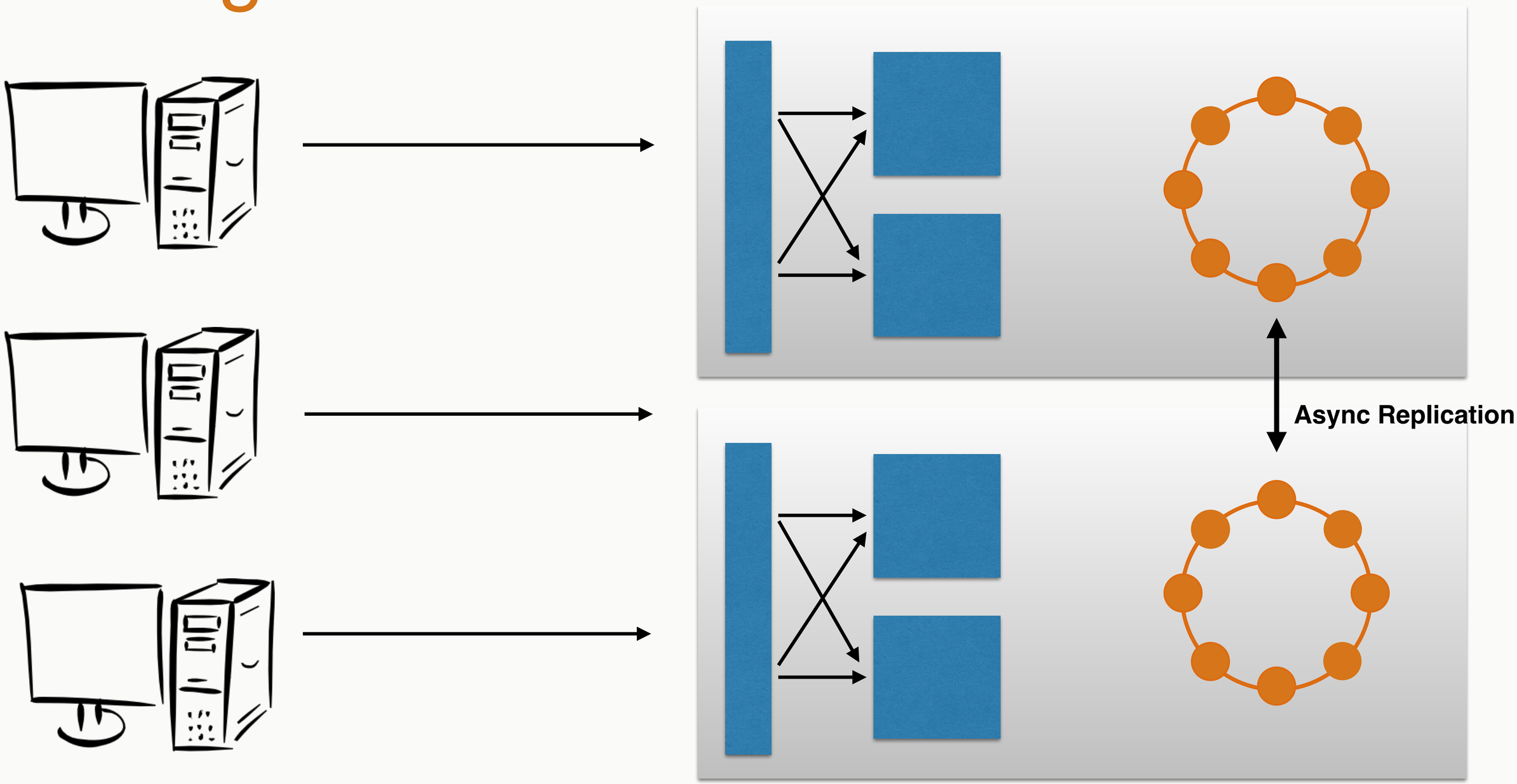
- Simple
  - Give it to the next node in the ring
  - Don’t use this in production
- NetworkTopology
  - Every Cassandra node knows its DC and Rack
  - Replicas won’t be put on the same rack unless Replication Factor > # of racks
  - Unfortunately Cassandra can’t create servers and racks on the fly to fix this :(

Tunable Consistency

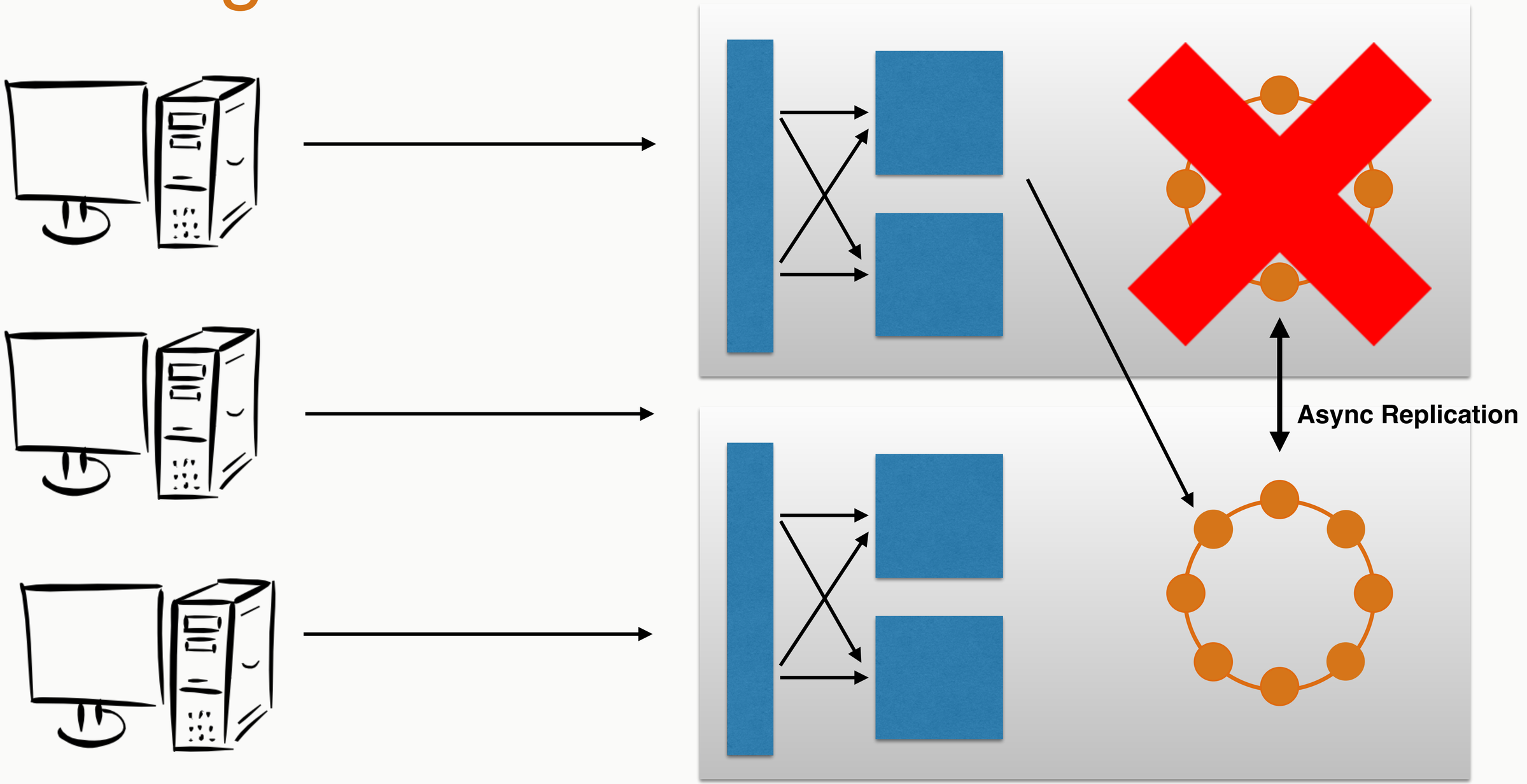
DATASTAX

- Data is replicated N times
- Every query that you execute you give a consistency
  - ALL
  - QUORUM
  - LOCAL_QUORUM
  - ONE
- **Christos Kalantzis** Eventual Consistency != Hopeful Consistency: http://youtu.be/A6qzx_HE3EU?list=PLqcm6qE9lgKJzVvwHprow9h7KMpb5hcUU

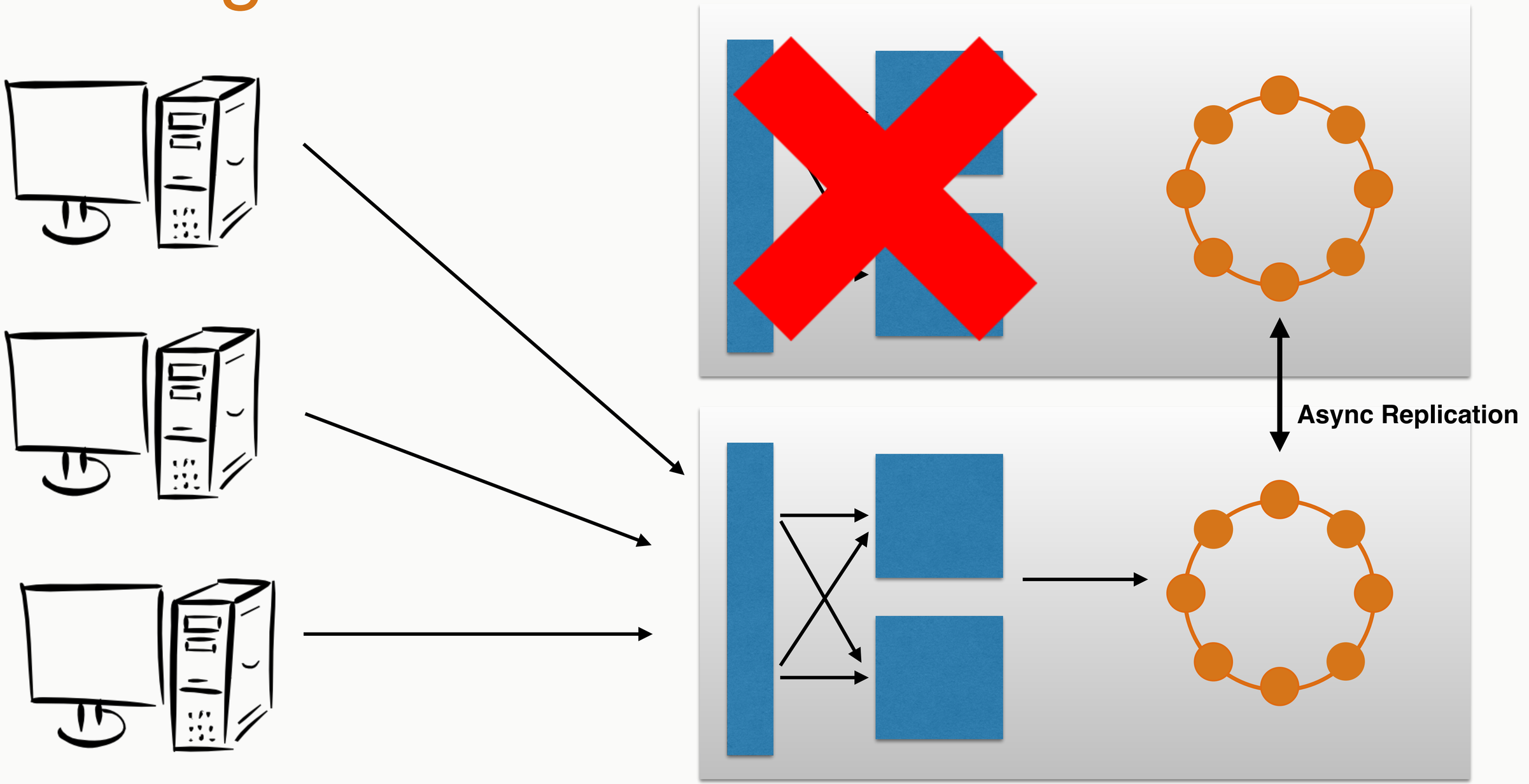
Handling hardware failure



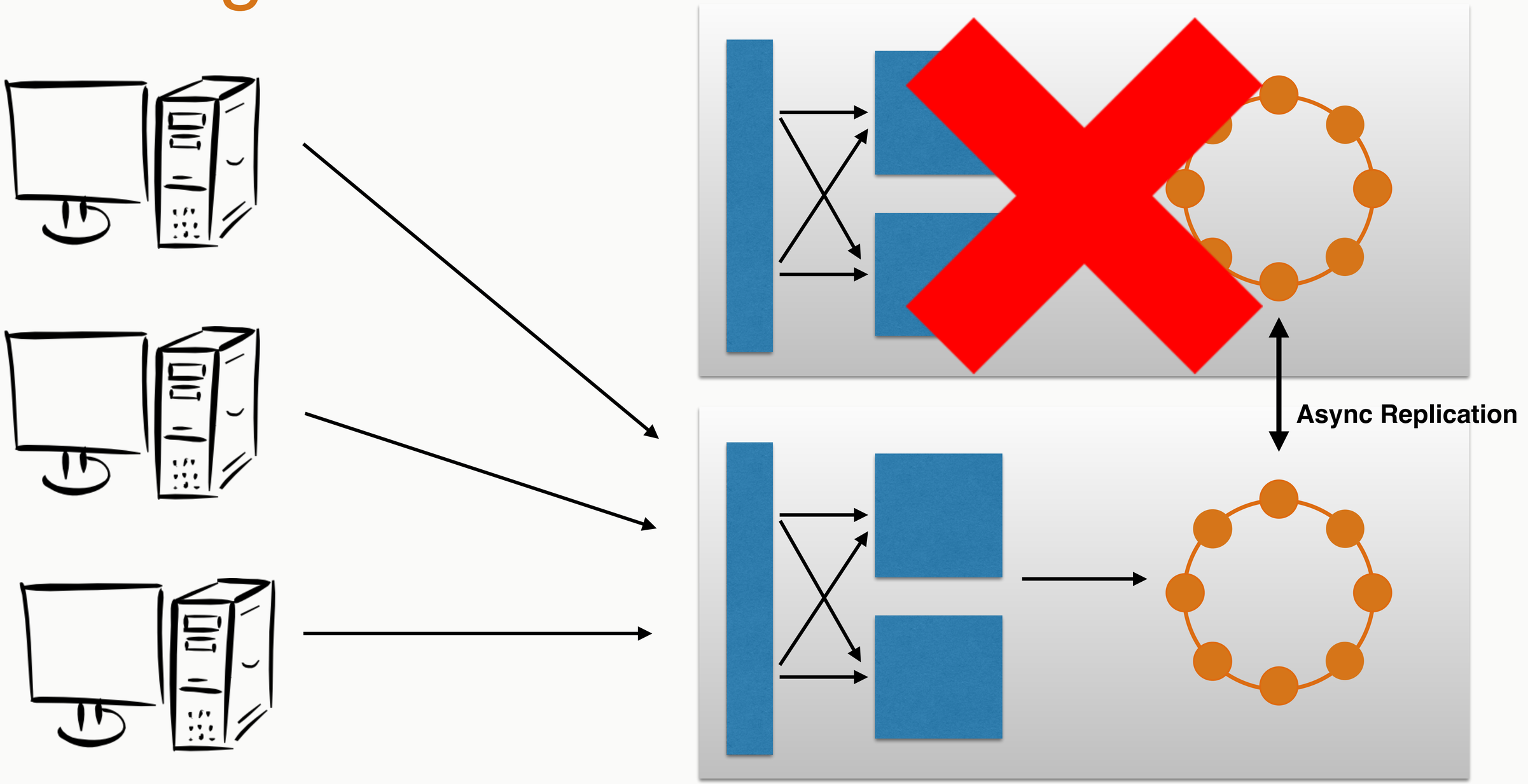
Handling hardware failure



Handling hardware failure

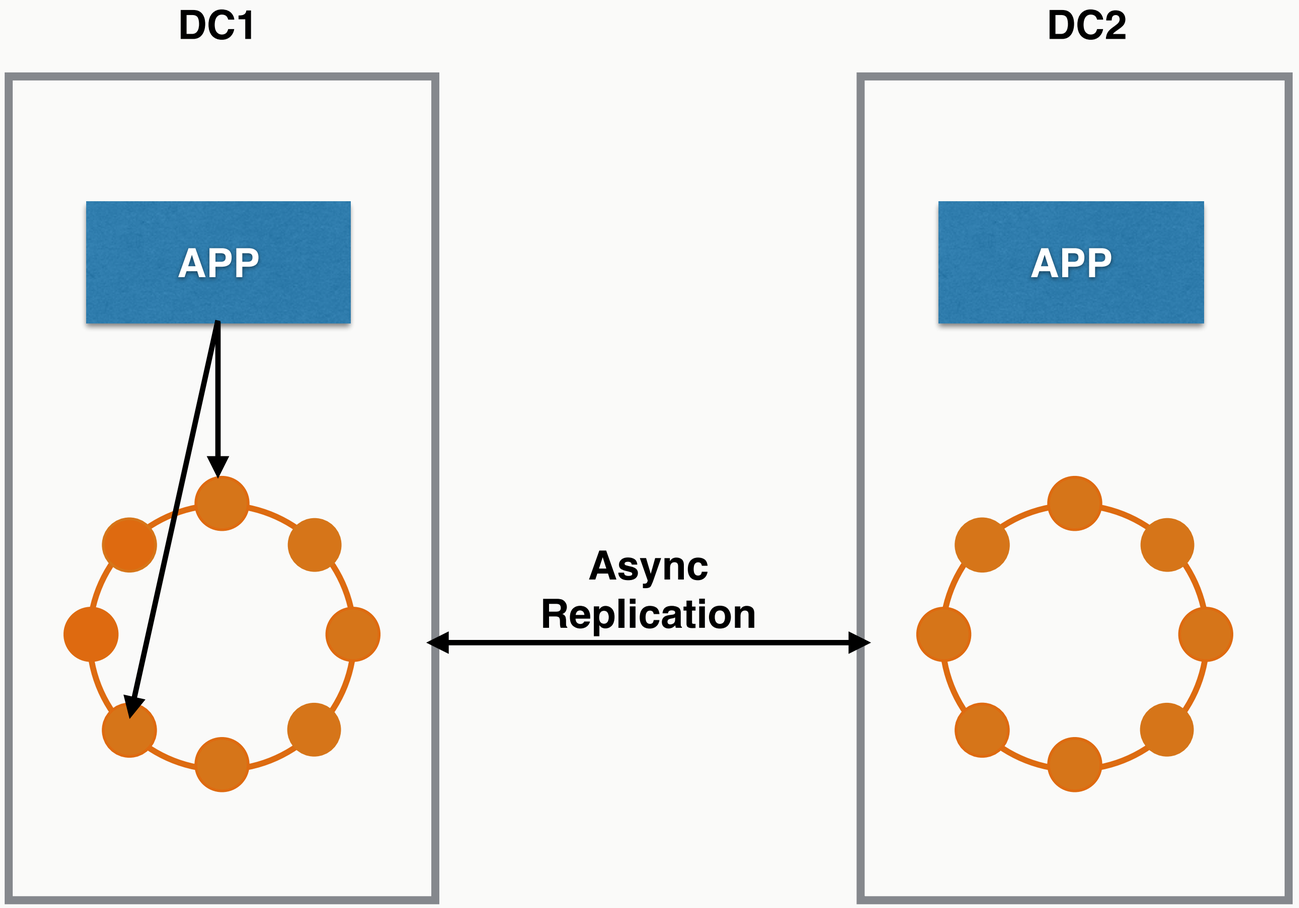


Handling hardware failure



Load balancing

- **Data centre aware policy**
- Token aware policy
- Latency aware policy
- Whitelist policy

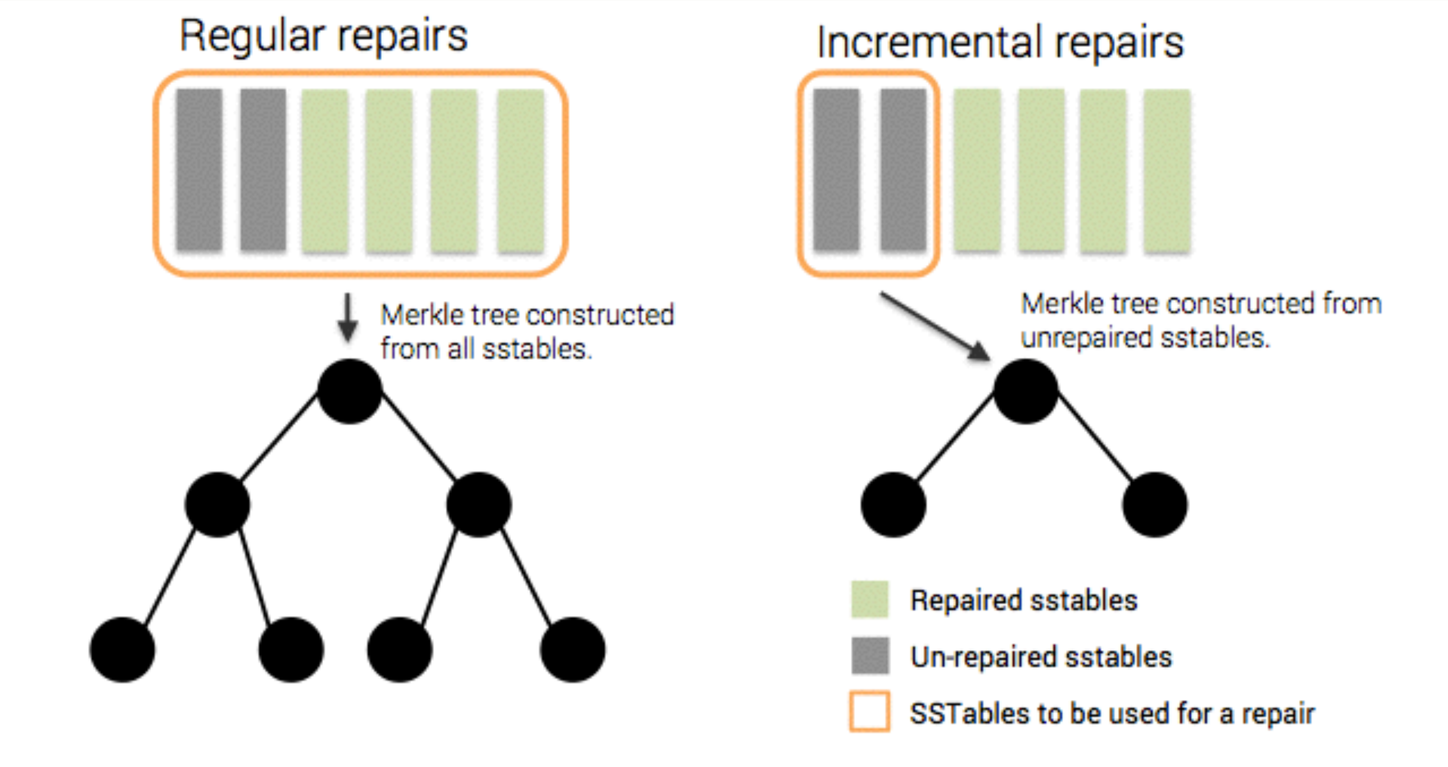


But what happens when they come back?

- Hinted handoff to the rescue
- Coordinators keep writes for downed nodes for a configurable amount of time, default 3 hours
- Longer than that run a repair

Anti entropy repair

- Not exciting but mandatory :)
- New in 2.1 - incremental repair <— awesome



Don’t forget to be social

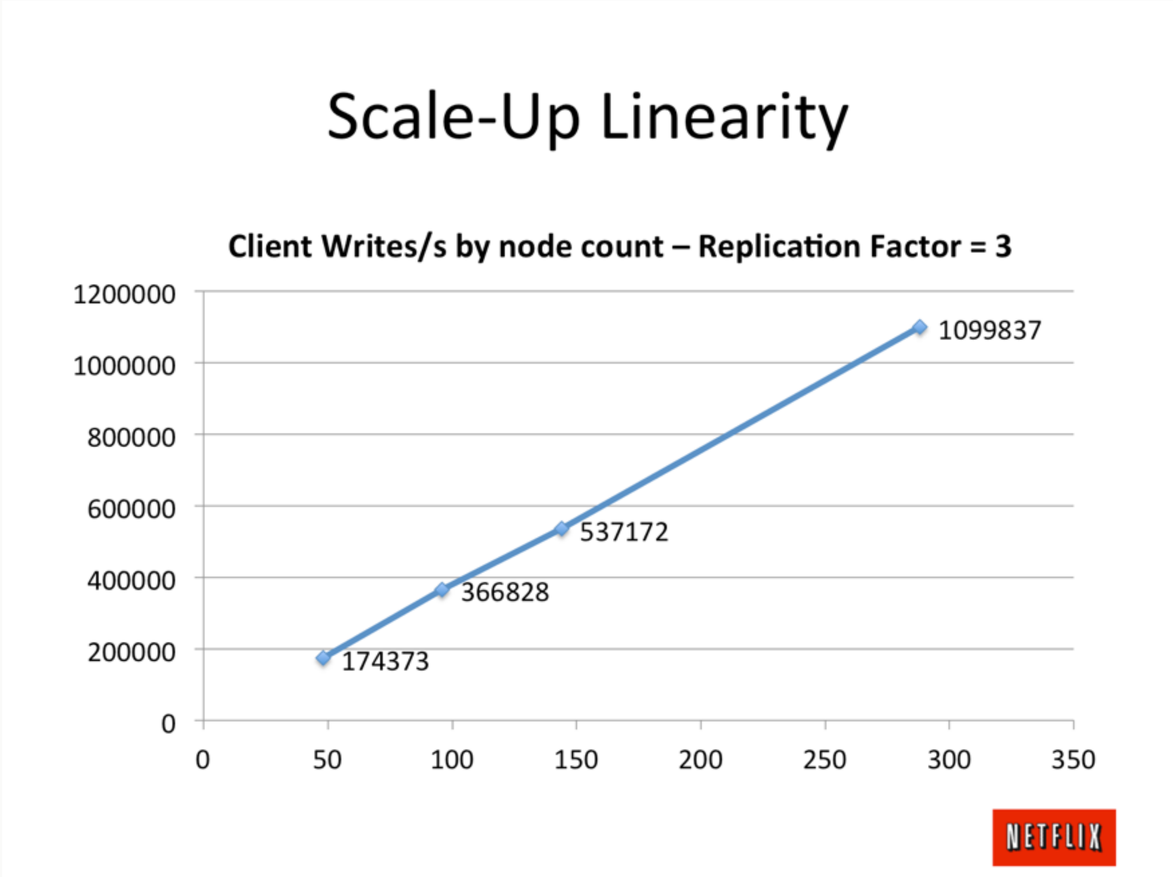
- Each node talks to a few of its other and shares information

@chbatey

Scaling shouldn't be hard

DATASTAX

- Throw more nodes at a cluster
- Bootstrapping + joining the ring
  - For large data sets this can take some time



DATASTAX

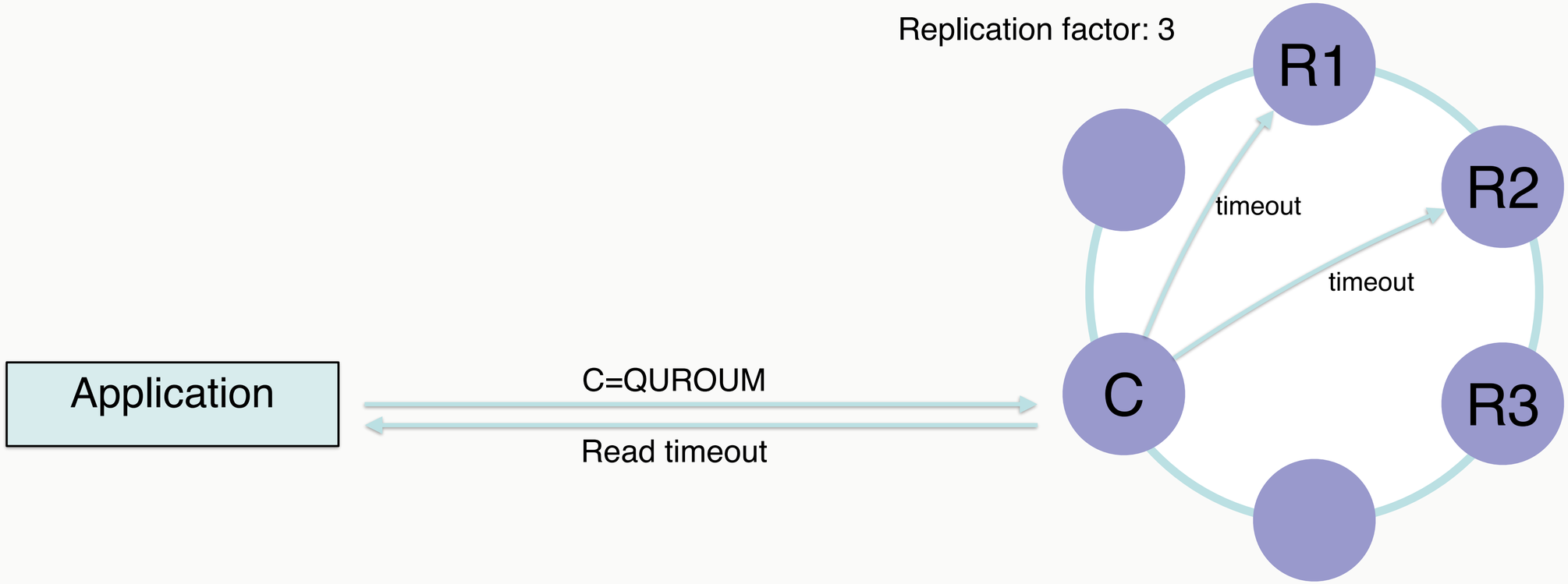
Cassandra failure modes

Types of failure

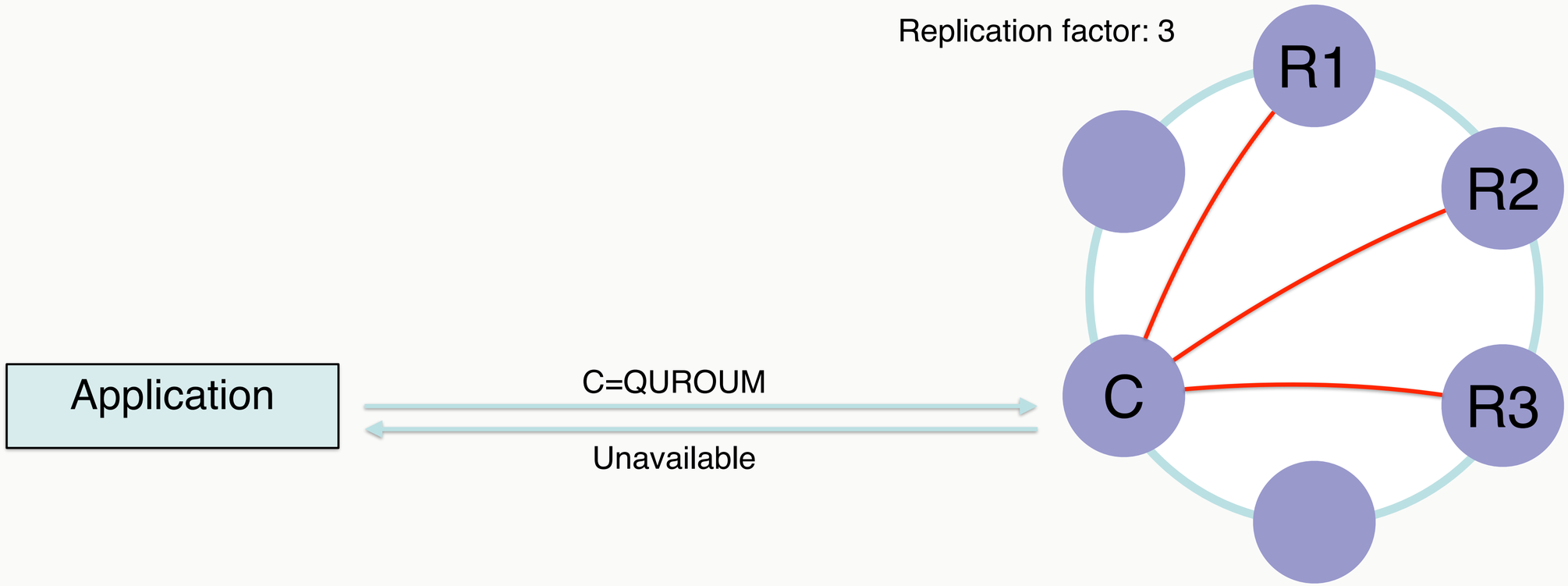
DATASTAX

- Server failure
  - Fail to write to disk
  - Whole node fail
  - Socket read time out
- Consistency failure
  - Read timeout
  - Write timeout
  - Unavailable
- Application failures
  - Idempotent?

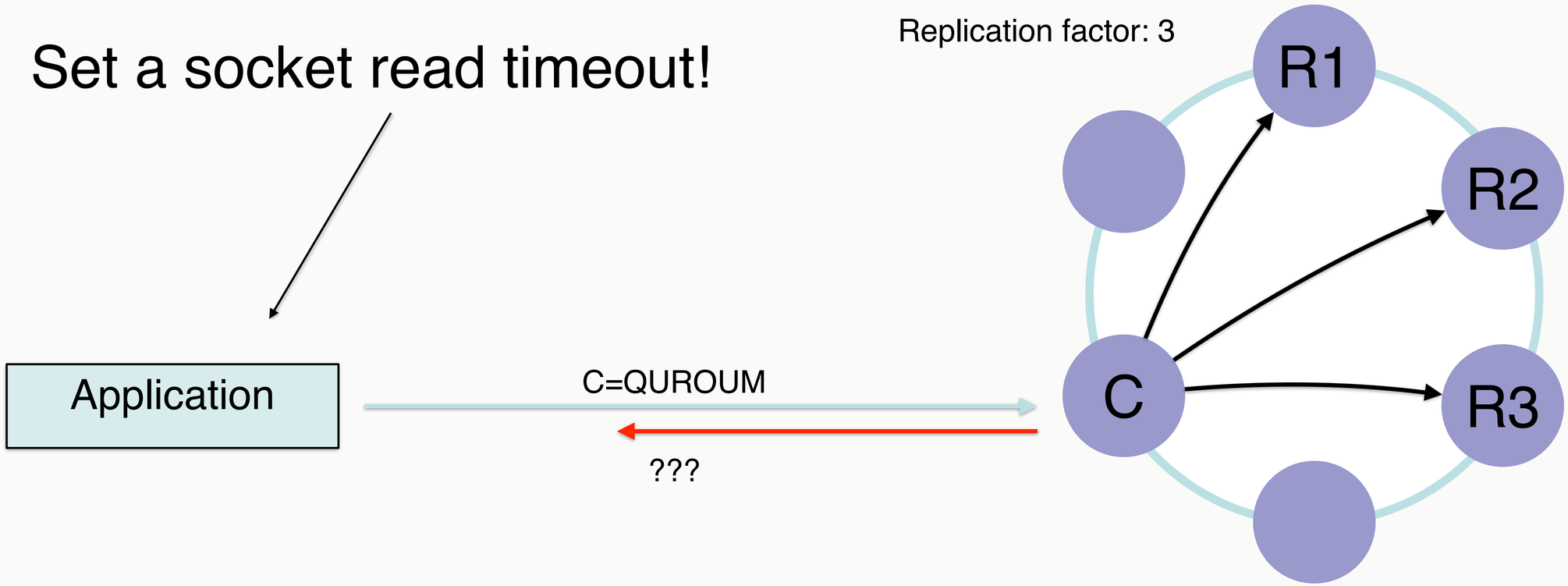
Read and write timeout



Unavailable



Coordinator issue



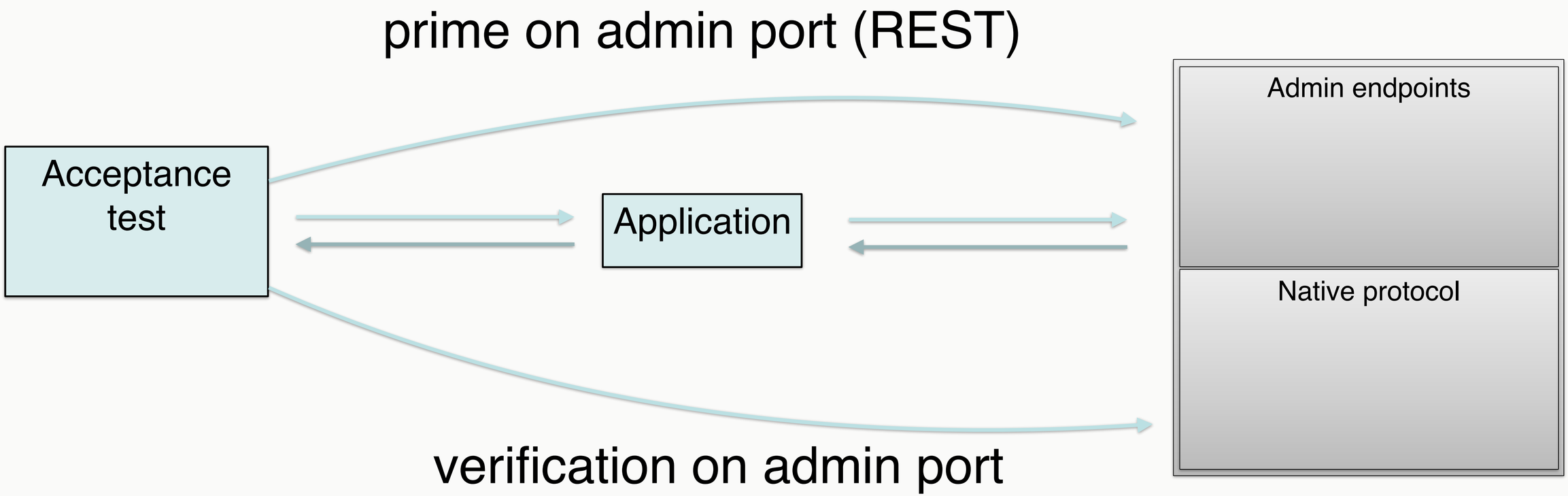
How do I test this? The test double

DATASTAX

- Release it - great book
- Wiremock - mocking HTTP services
- Saboteur - adding network latency

If you want to build a fault tolerant application you better test faults!

Stubbed Cassandra



Test with a real Cassandra

DATASTAX

- Reduced throughput / increased latency
  - Nodes down
  - Racks down
  - DC down

Test with a real Cassandra

DATASTAX

- Reduced throughput / increased latency
  - Nodes down
  - Racks down
  - DC down
- Automate this if possible
- Load test your schema
  - Cassandra stress
  - JMeter Cassandra plugin

Summary

DATASTAX

- Don’t build services with a single point of failure
- Cassandra deployed correctly means you can handle node, rack and entire DC failure
- Know when to pick availability over consistency
- Accessing Cassandra from Java is boringly easy so I left it out

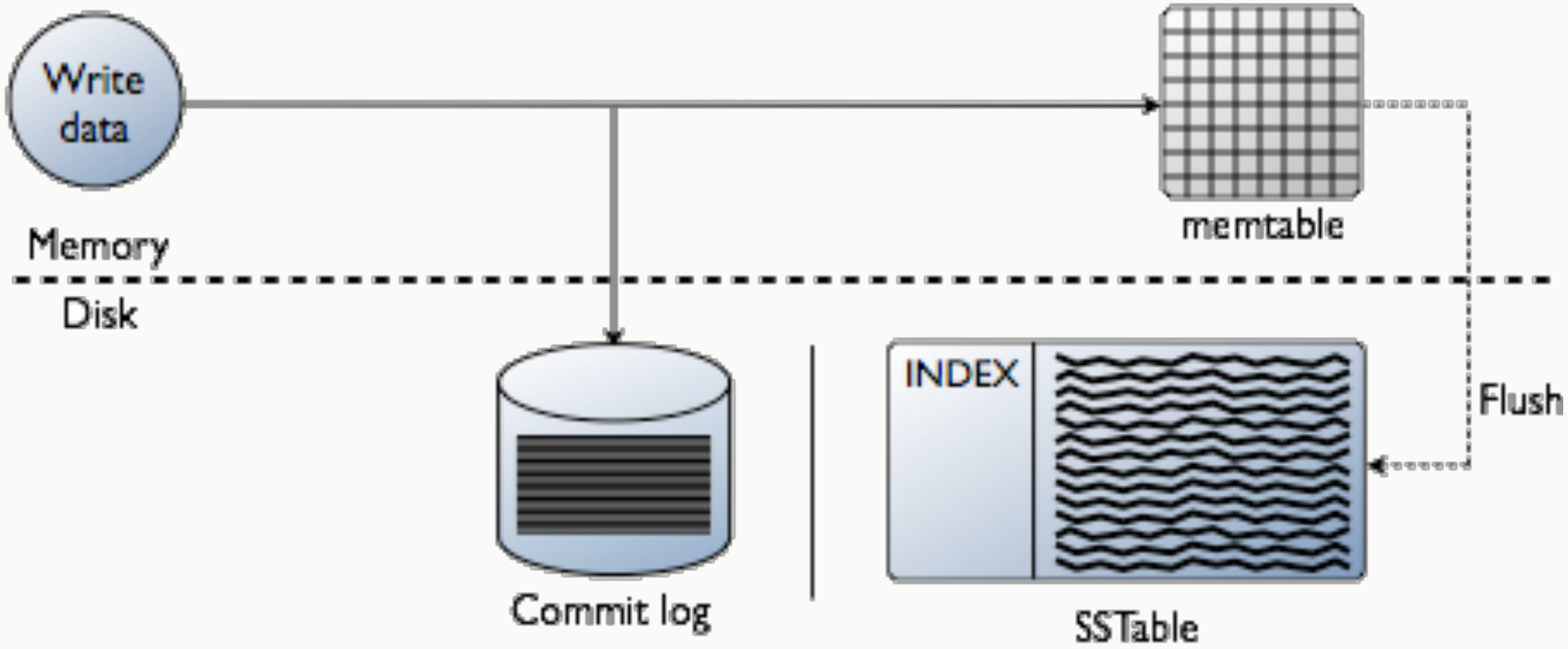
Thanks for listening

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

Write path on an individual server

DATASTAX

- A commit log for durability
- Data also written to an in-memory structure (memtable) and then to disk once the memory structure is full (an SStable)



DATASTAX

Cassandra + Java

DataStax Java Driver

DATASTAX

- 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>
```

Modelling in Cassandra

DATASTAX

```
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'}
```

Get all the events

DATASTAX

```
public List<CustomerEvent> getAllCustomerEvents() {
    return session.execute("select * from customers.customer_events")
            .all().stream()
            .map(mapCustomerEvent())
            .collect(Collectors.toList());
}


private Function<Row, CustomerEvent> mapCustomerEvent() {
    return row -> new CustomerEvent(
            row.getString("customer_id"),
            row.getUUID("time"),
            row.getString("staff_id"),
            row.getString("store_type"),
            row.getString("event_type"),
            row.getMap("tags", String.class, String.class));
}
```

All events for a particular customer

DATASTAX

```
private PreparedStatement getEventsForCustomer;

@PostConstruct
public void prepareSatements() {
    getEventsForCustomer =
     session.prepare("select * from customers.customer_events where customer_id = ?");
}

public List<CustomerEvent> getCustomerEvents(String customerId) {
    BoundStatement boundStatement = getEventsForCustomer.bind(customerId);
    return session.execute(boundStatement)
            .all().stream()
            .map(mapCustomerEvent())
            .collect(Collectors.toList());
}
```

Customer events for a time slice

DATASTAX

```
public List<CustomerEvent> getCustomerEventsForTime(String customerId, long startTime,
long endTime) {

    Select.Where getCustomers = QueryBuilder.select()
            .all()
            .from("customers", "customer_events")
            .where(eq("customer_id", customerId))
            .and(gt("time", UUIDs.startOf(startTime)))
            .and(lt("time", UUIDs.endOf(endTime)));


    return session.execute(getCustomers).all().stream()
            .map(mapCustomerEvent())
            .collect(Collectors.toList());
}
```

Mapping API

DATASTAX

```
@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

DATASTAX

```
@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();

    @Query("select * from customers.customer_events where customer_id = :customerId
and time > minTimeuuid(:startTime) and time < maxTimeuuid(:endTime)")
    Result<CustomerEvent> getCustomerEventsForTime(String customerId, long startTime,
long endTime);
}


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

Adding some type safety

DATASTAX

```
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

DATASTAX

```
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

DATASTAX

```
@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

DATASTAX

```
@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;
```

```
@Query("select * from customers.customer_events_type")
Result<CustomerEventType> getAllCustomerEventsWithStoreType();
```

Snooze…

Summary

DATASTAX

- Don’t build services with a single point of failure
- Cassandra deployed correctly means you can handle node, rack and entire DC failure
- Know when to pick availability over consistency
- Accessing Cassandra from Java is boringly easy so I left it out

Thanks for listening

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