The new face of Cassandra

Michaël Figuière
@mfiguiere

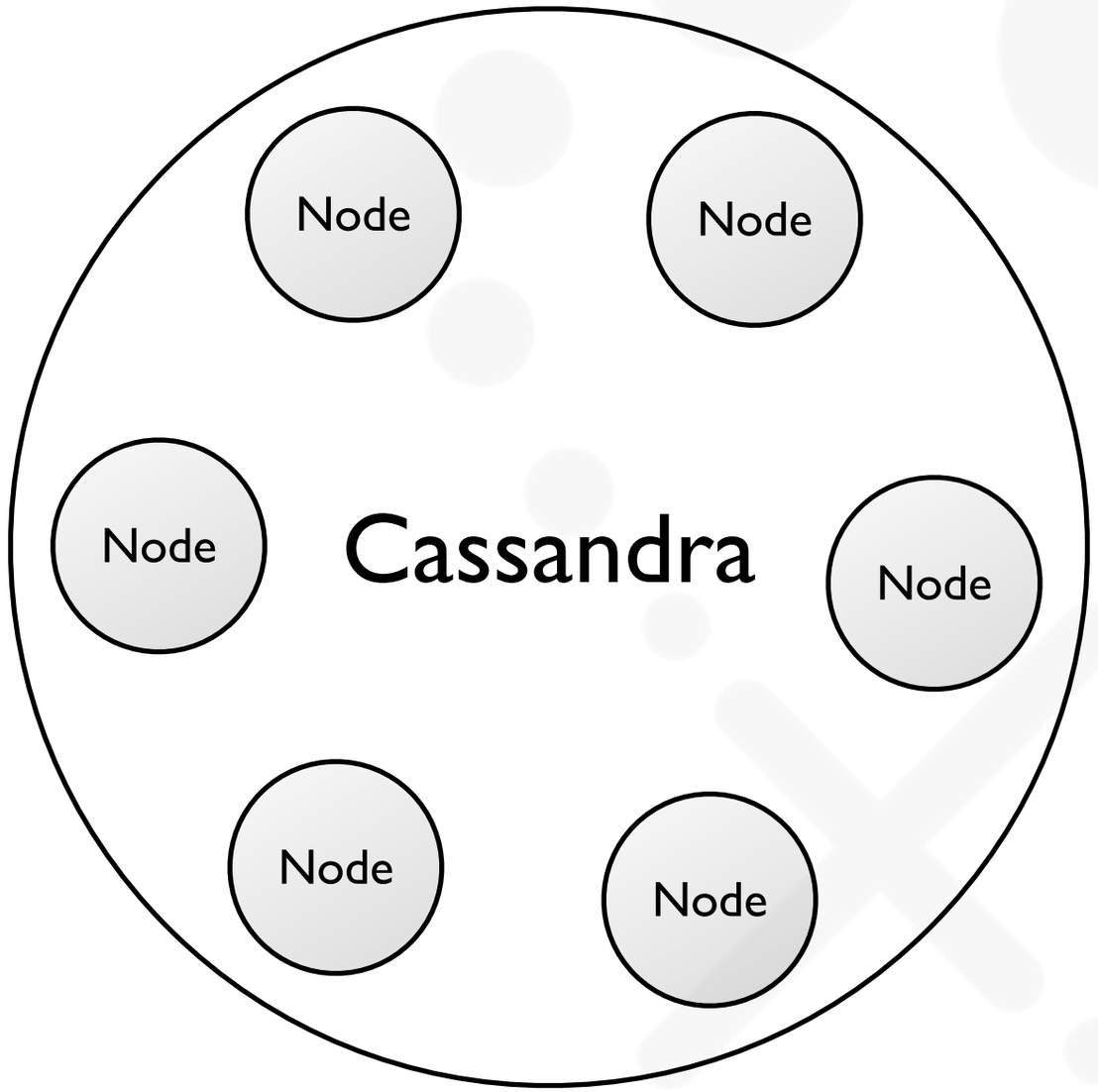
Speaker

Michaël Figuière

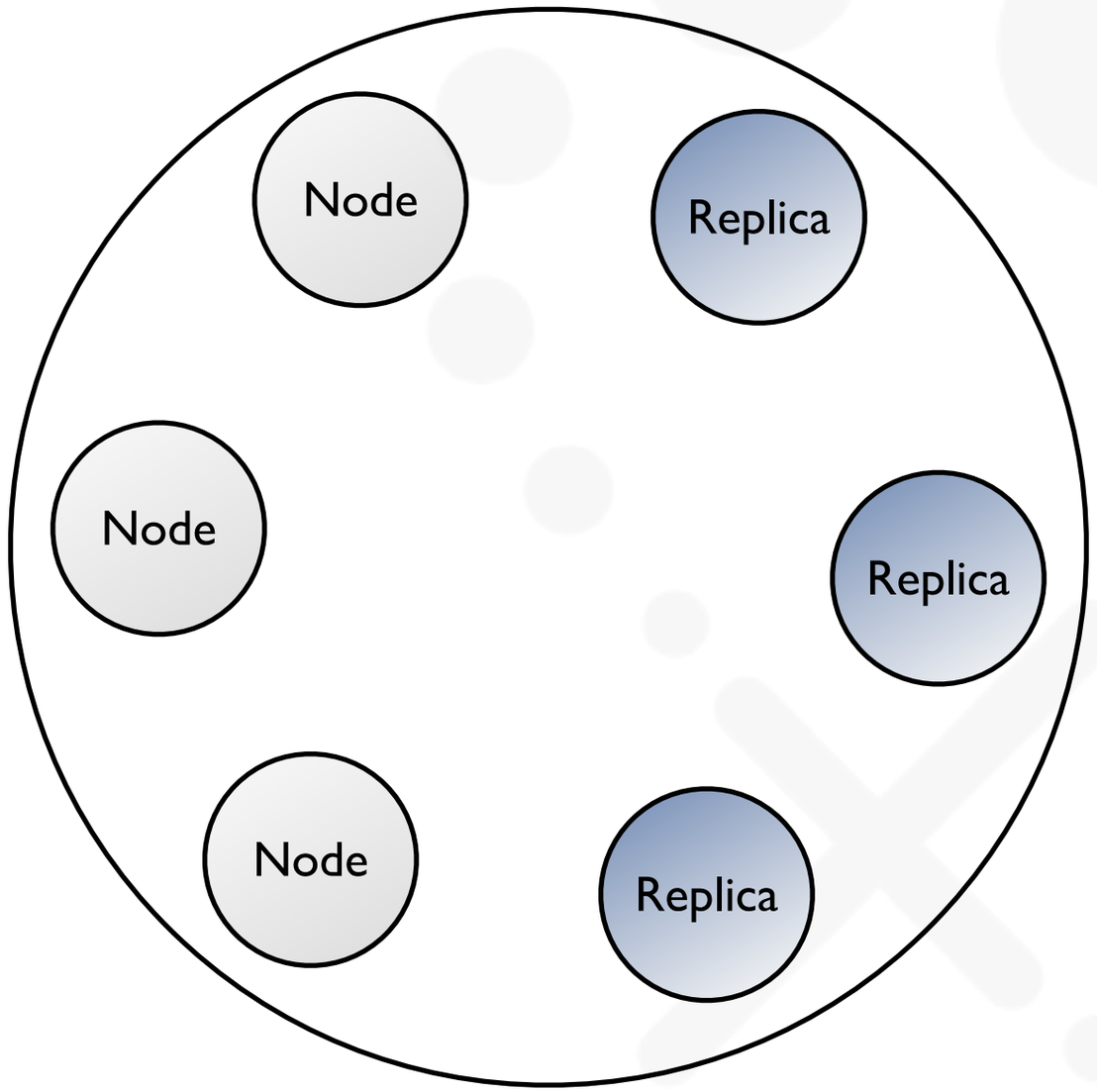
DATASTAX

@mfiguiere

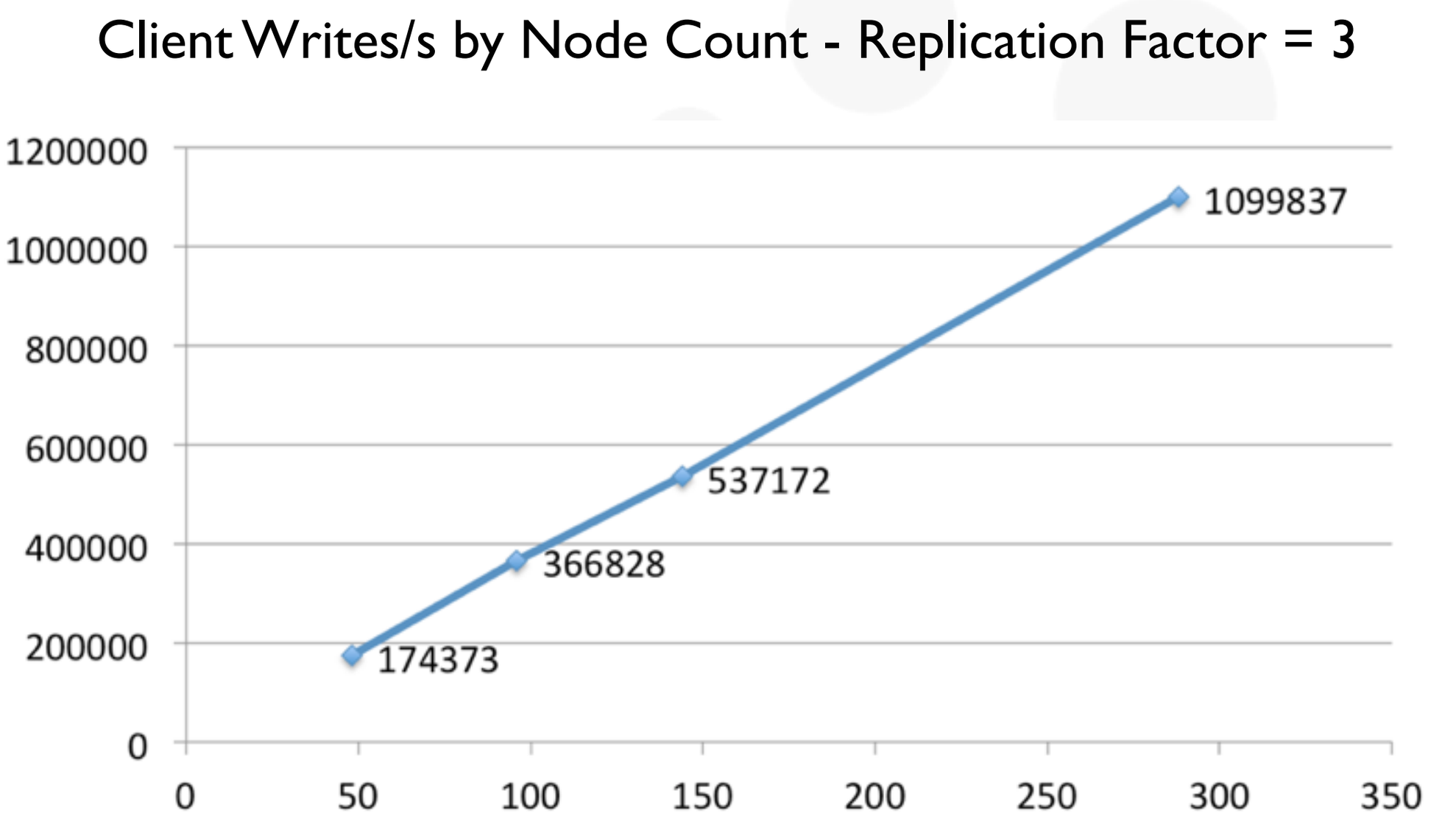
Ring Architecture



Ring Architecture

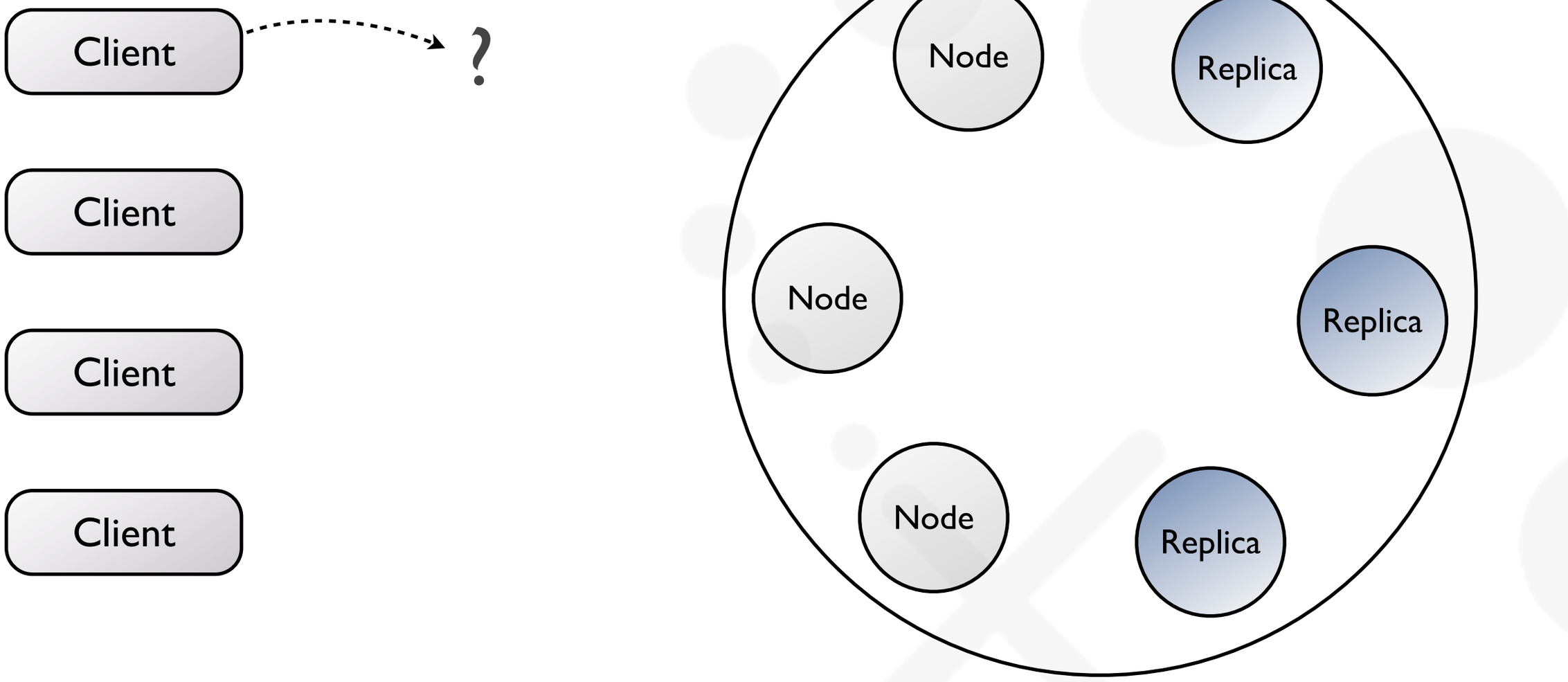


Linear Scalability

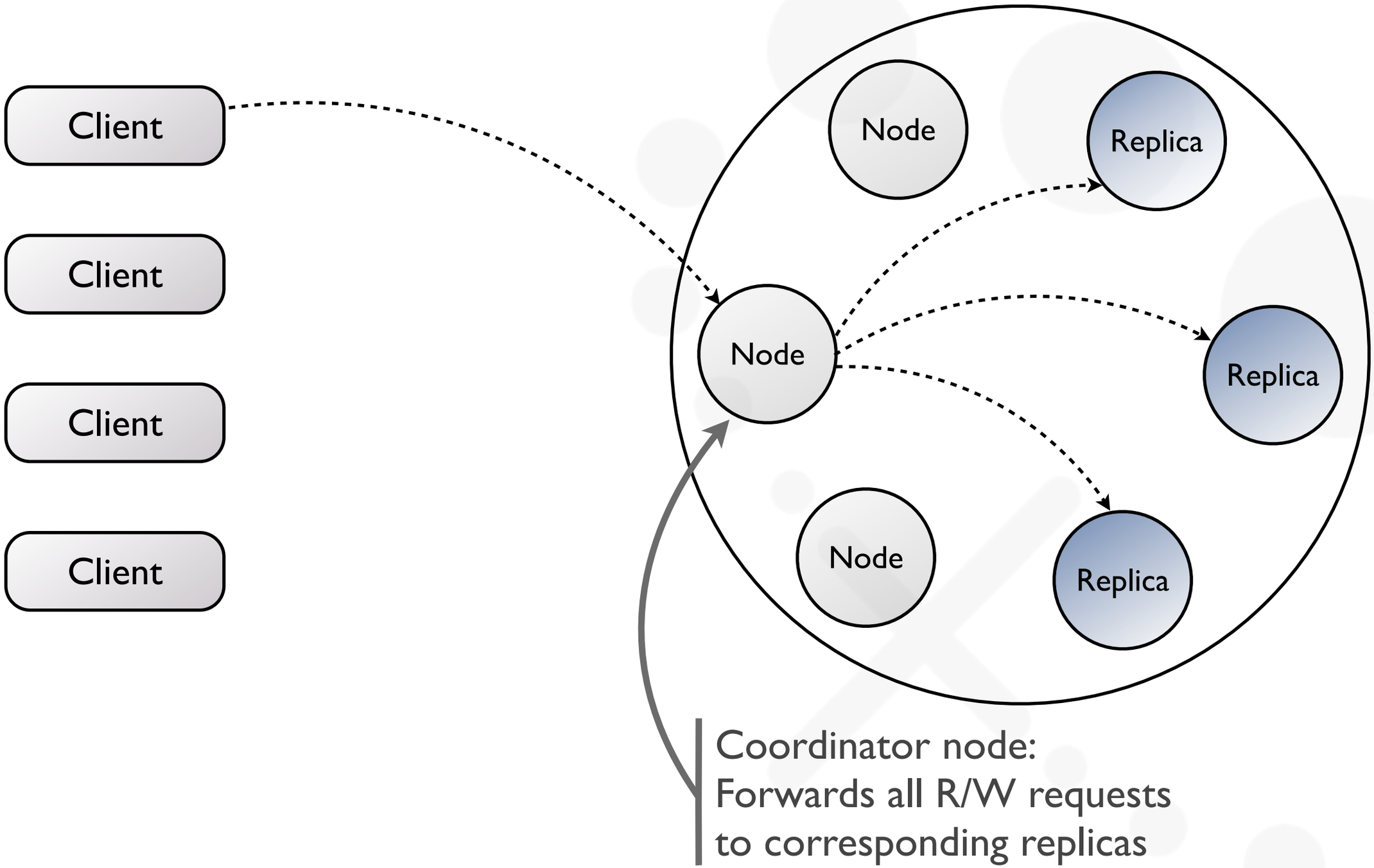


NETFLIX

Client / Server Communication

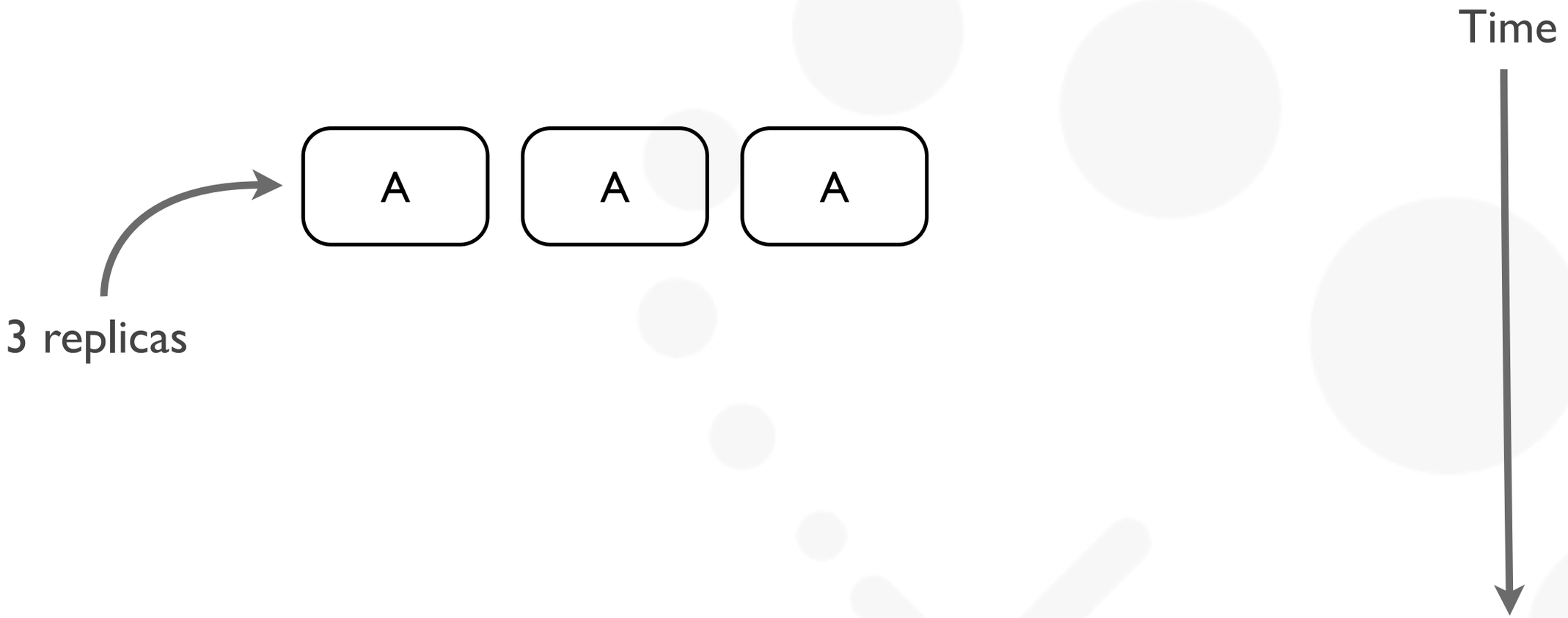


Client / Server Communication

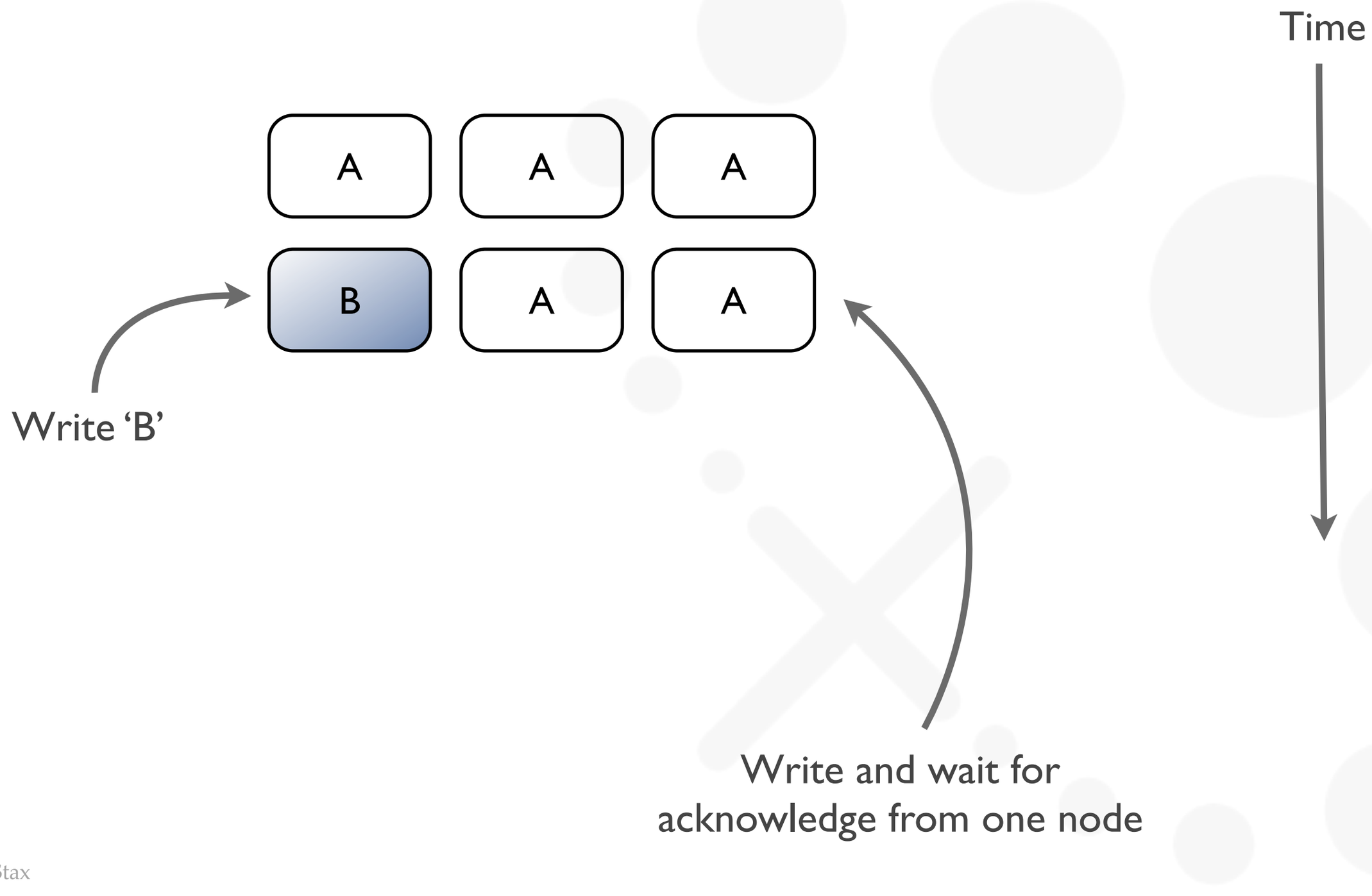


Coordinator node:
Forwards all R/W requests
to corresponding replicas

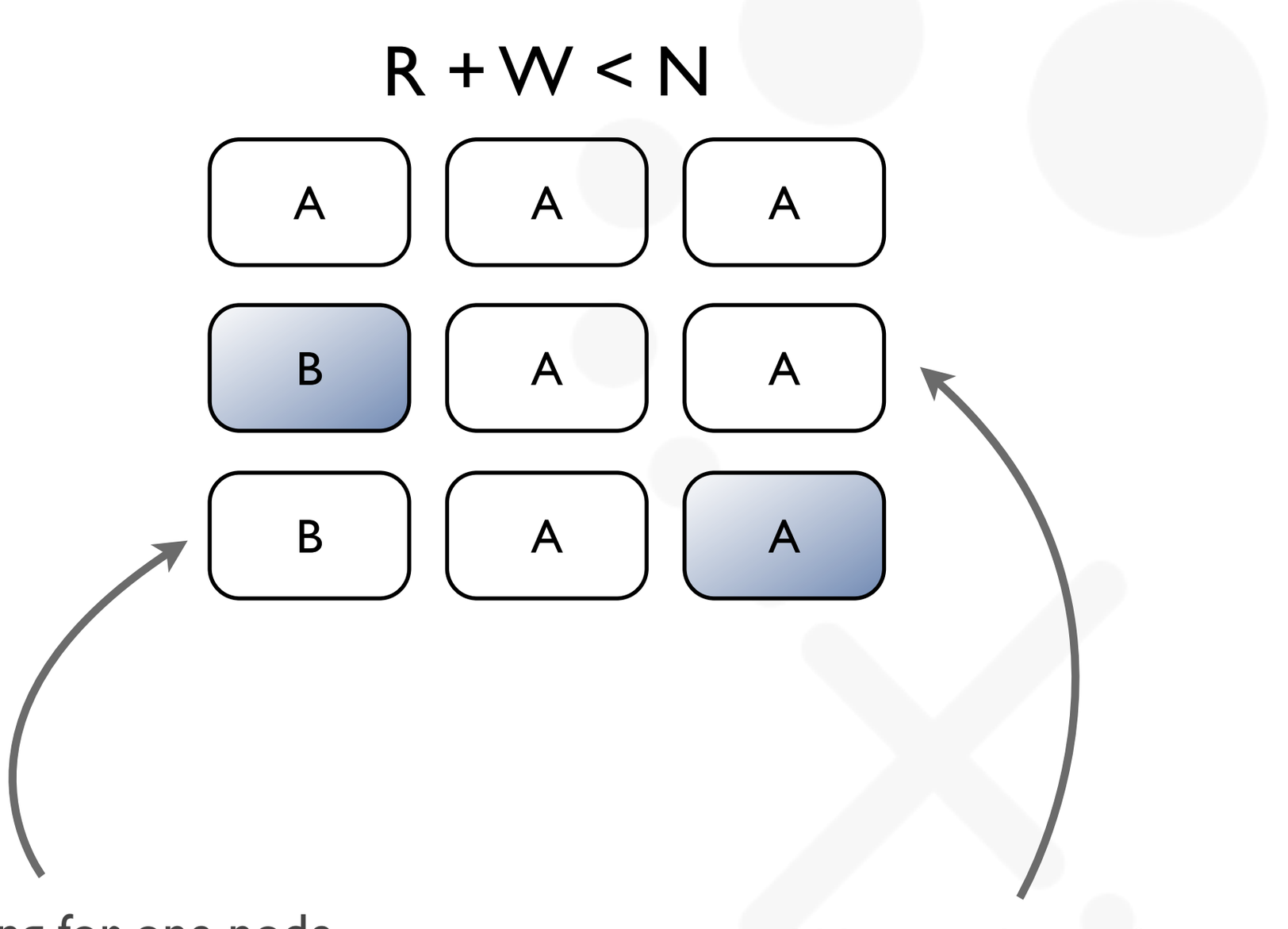
Tunable Consistency



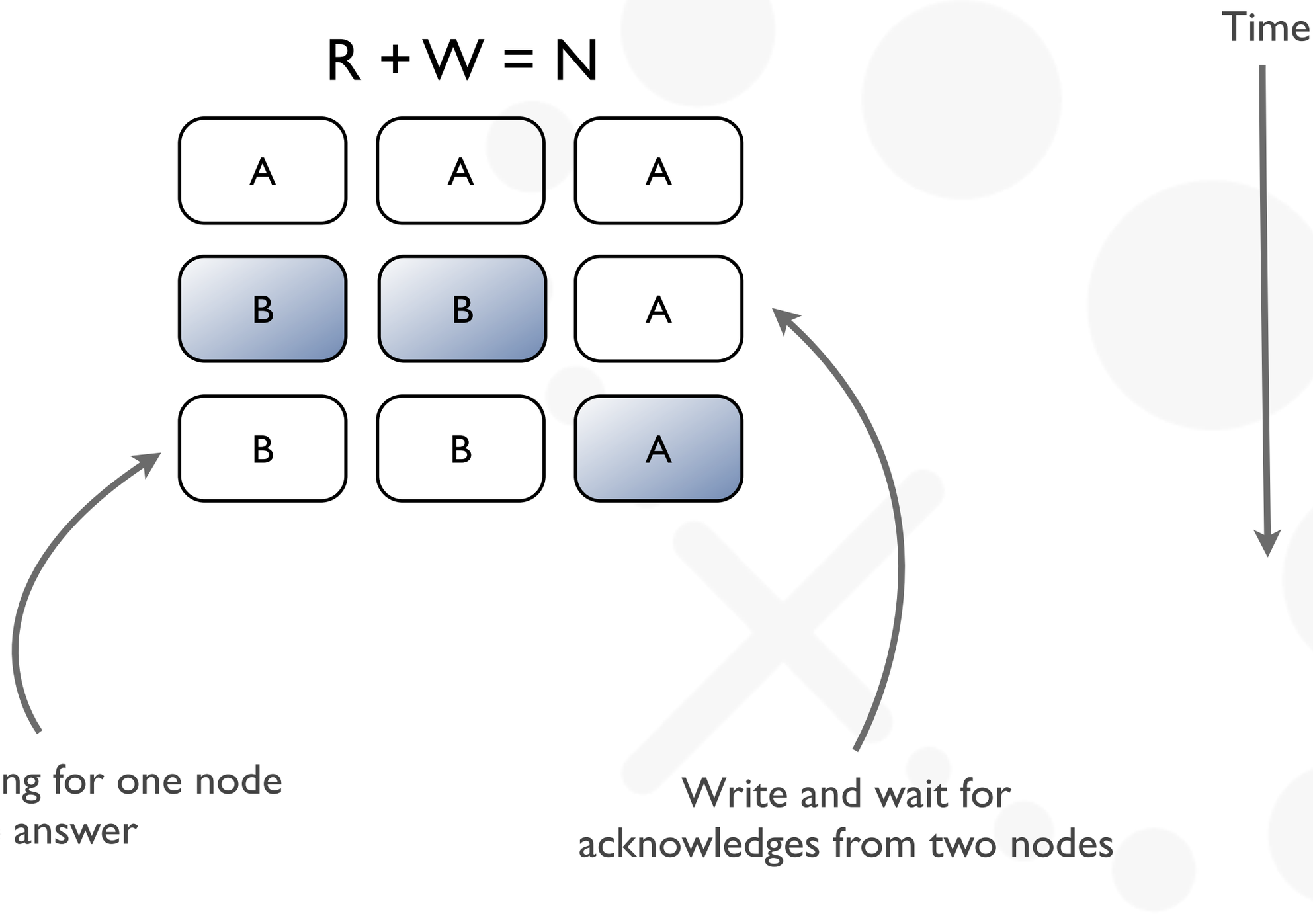
Tunable Consistency



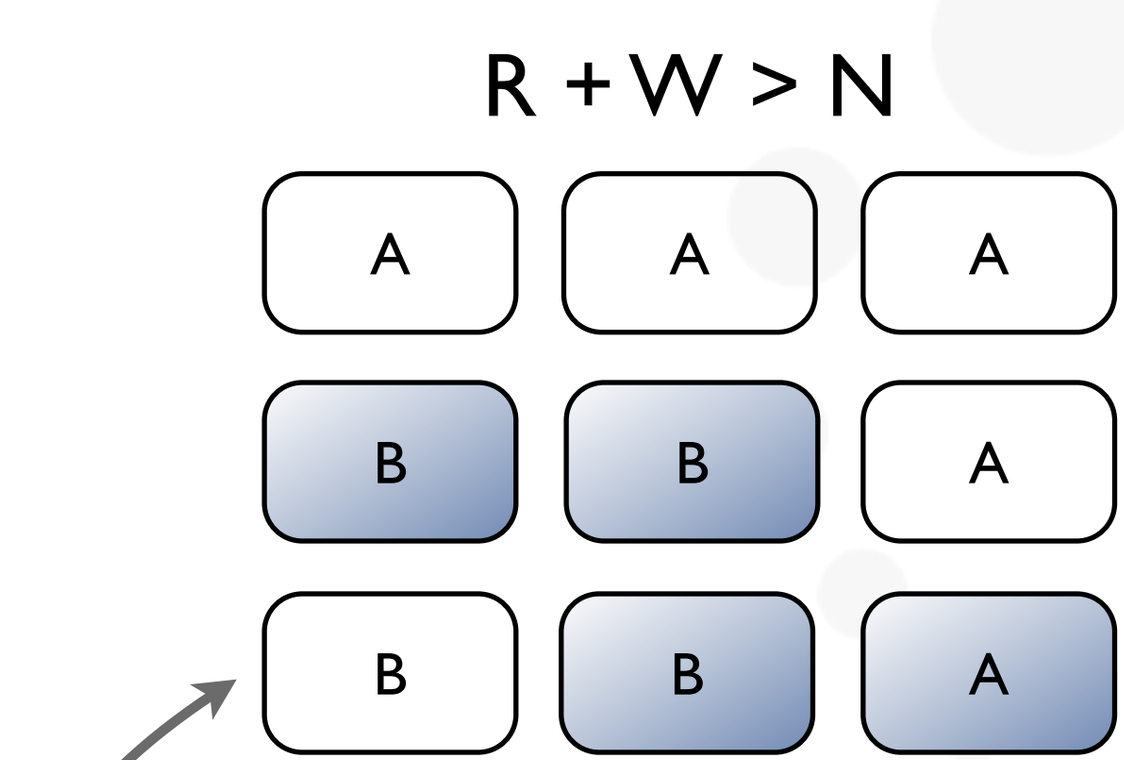
Tunable Consistency

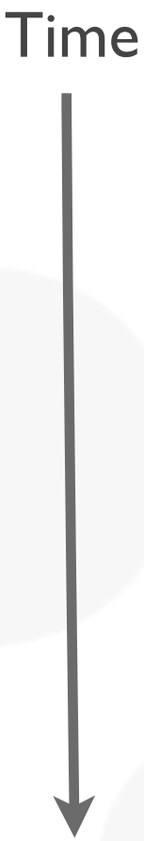


Tunable Consistency



Tunable Consistency

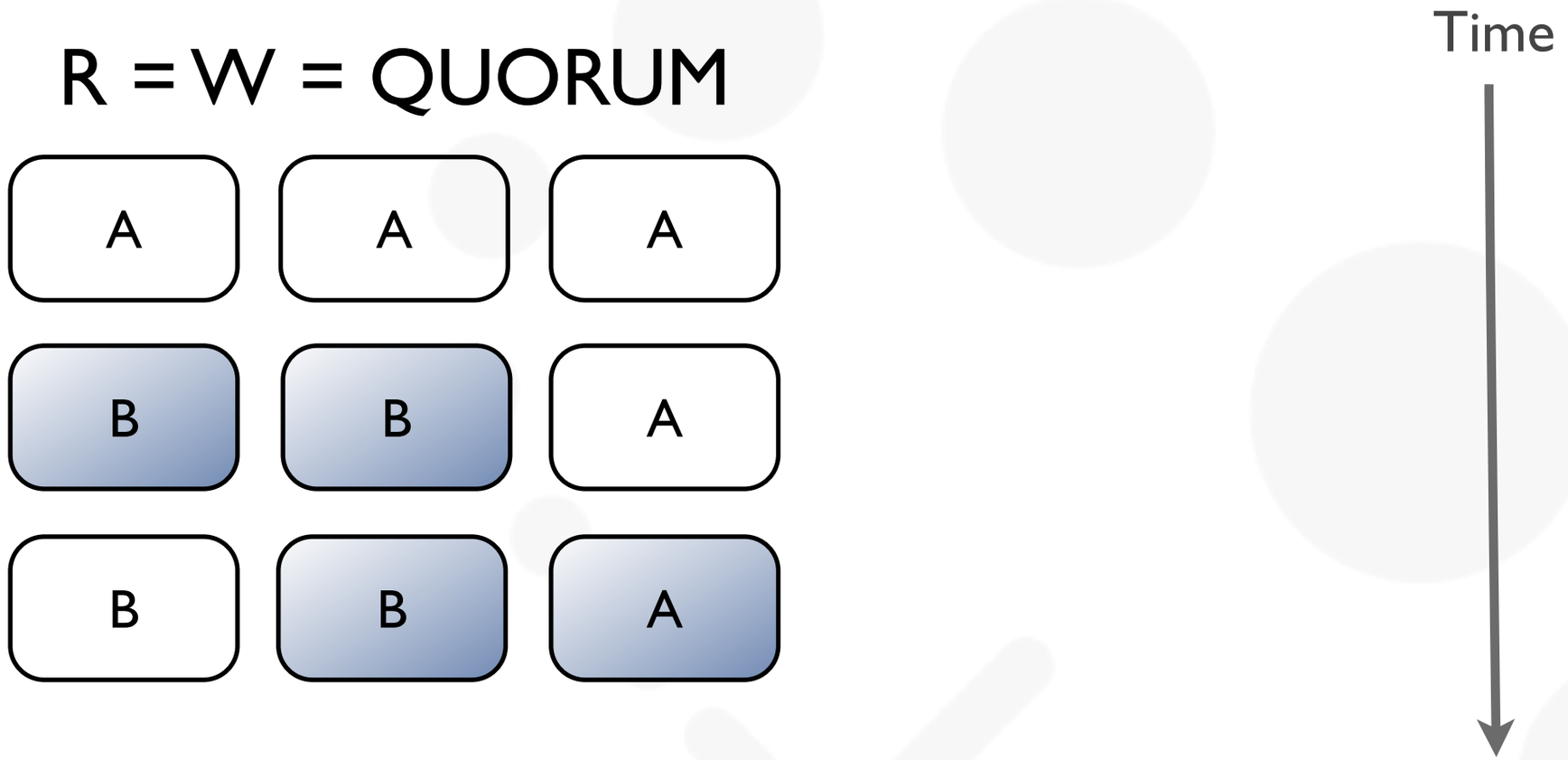




Read waiting for two nodes to answer

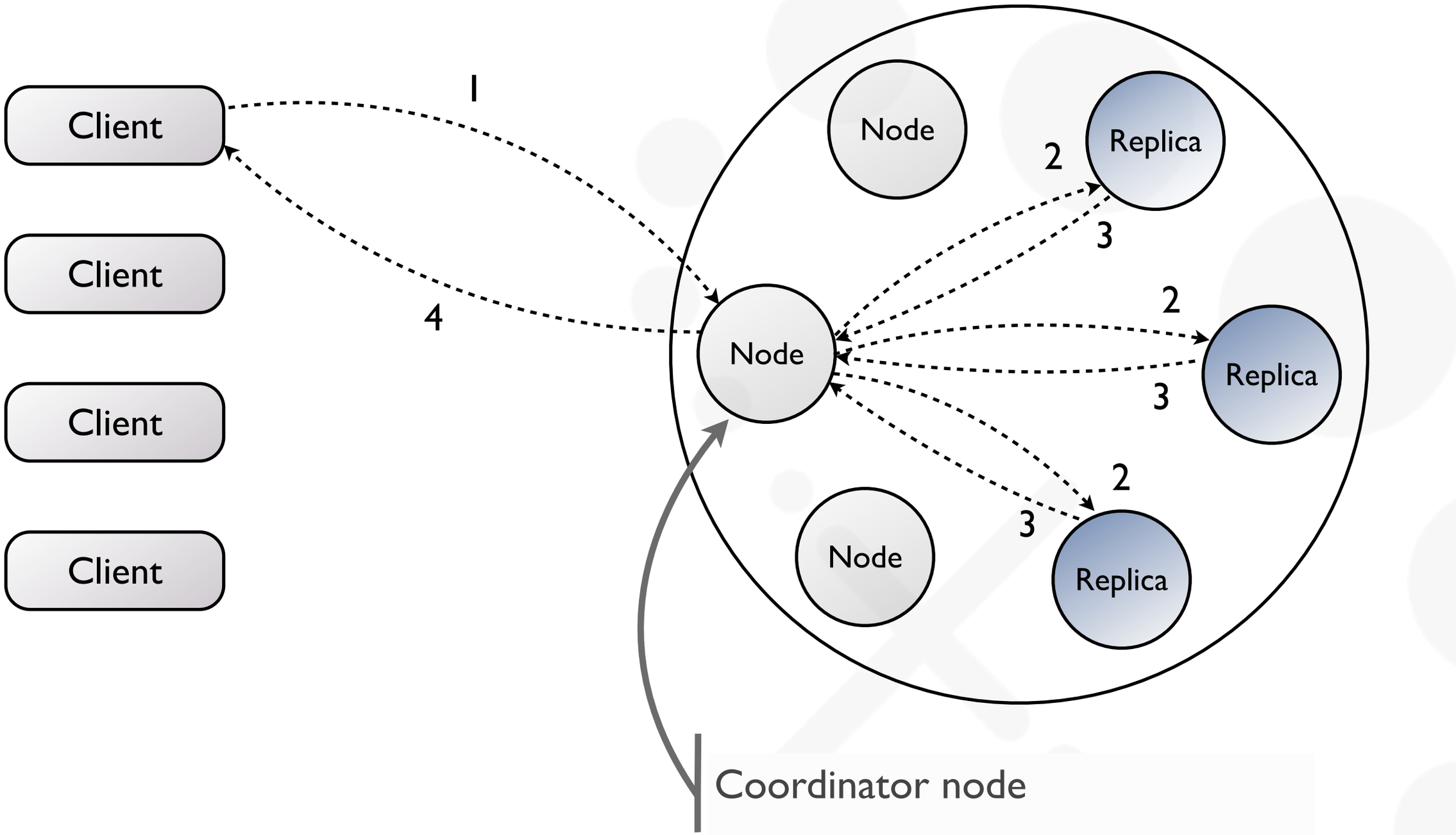
Write and wait for acknowledges from two nodes

Tunable Consistency

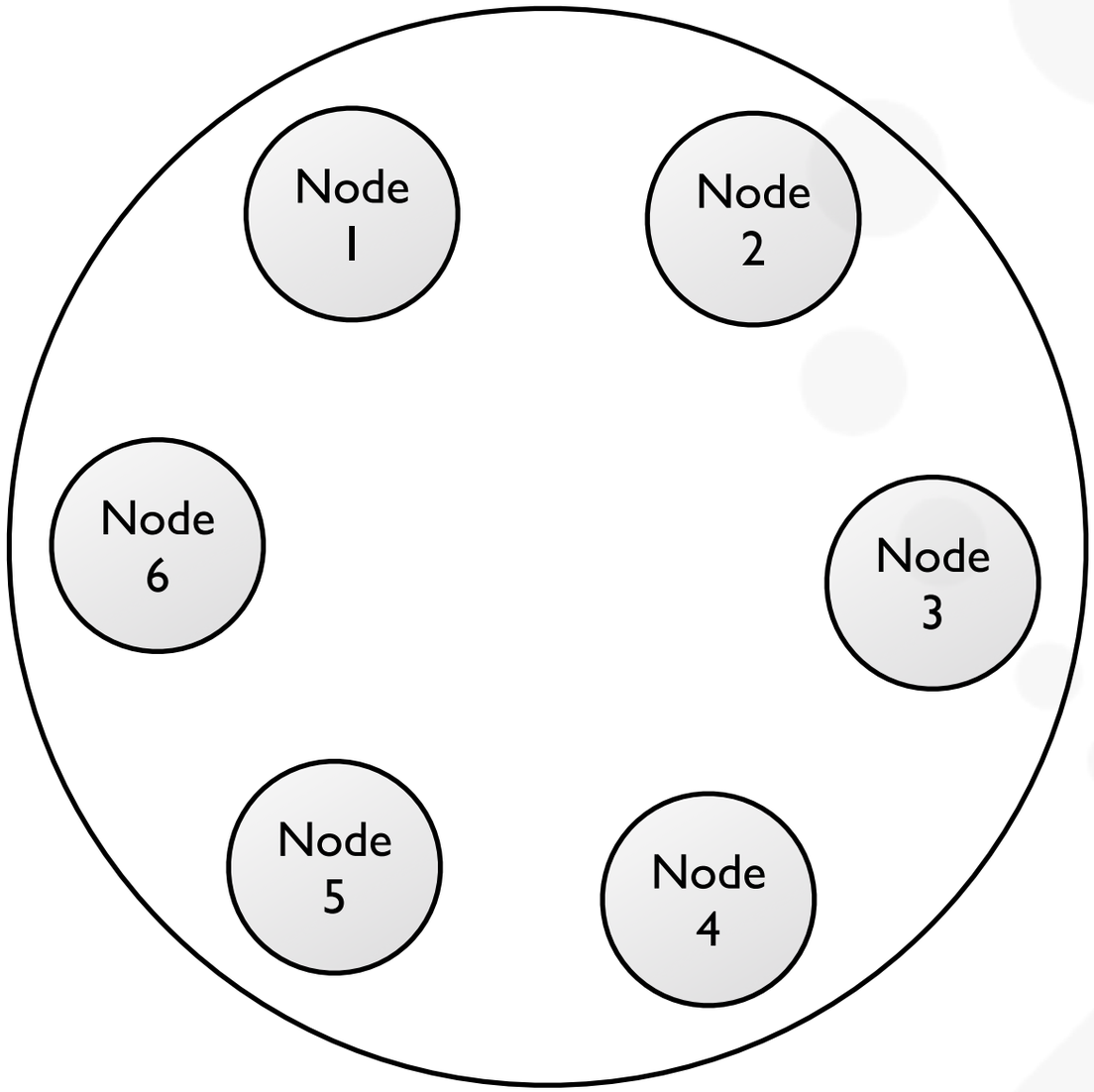


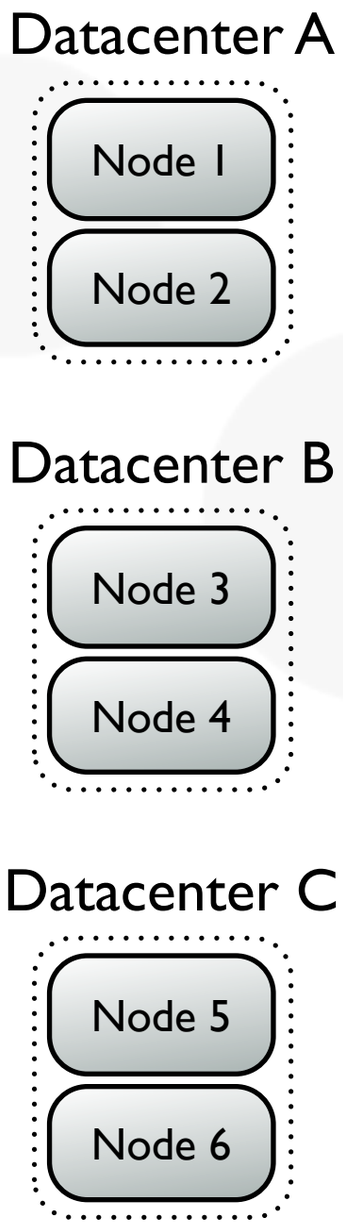
QUORUM = (N / 2) + 1

Request Path



Multi-Datacenter





Column Family Data Model

jbellis	address	email	name	state
	123 main	jb@ds.com	Jonathan	TX
dhutch	address	email	name	state
	45 2nd st	dh@ds.com	Daria	CA
egilmore	email	name		
	eg@ds.com	Eric		

Row Key

Columns = List of K/V pairs sorted by Key

Column Family Data Model

jbellis	datastax	dhutch	egilmore	mzcassie

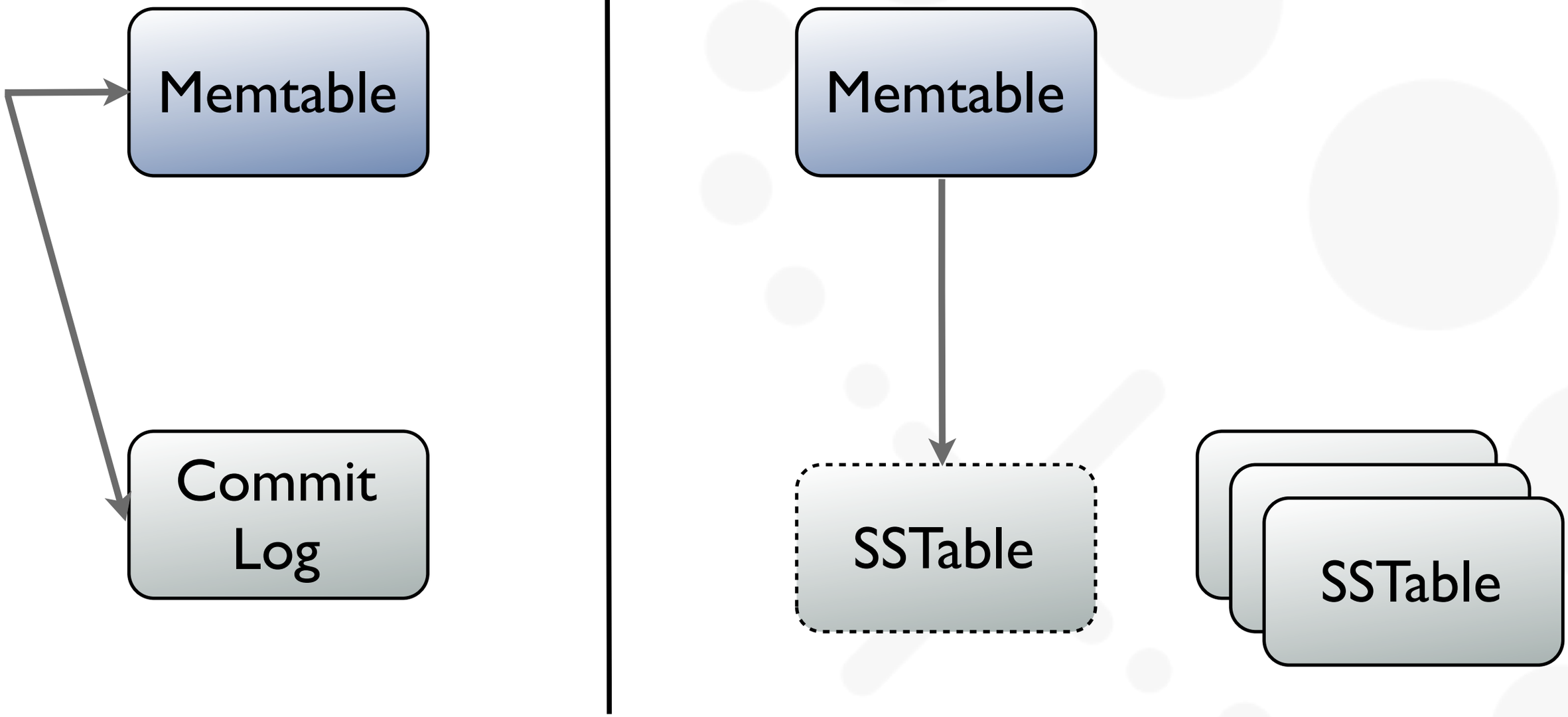
dhutch	egilmore

egilmore	datastax	mzcassie

Row Key

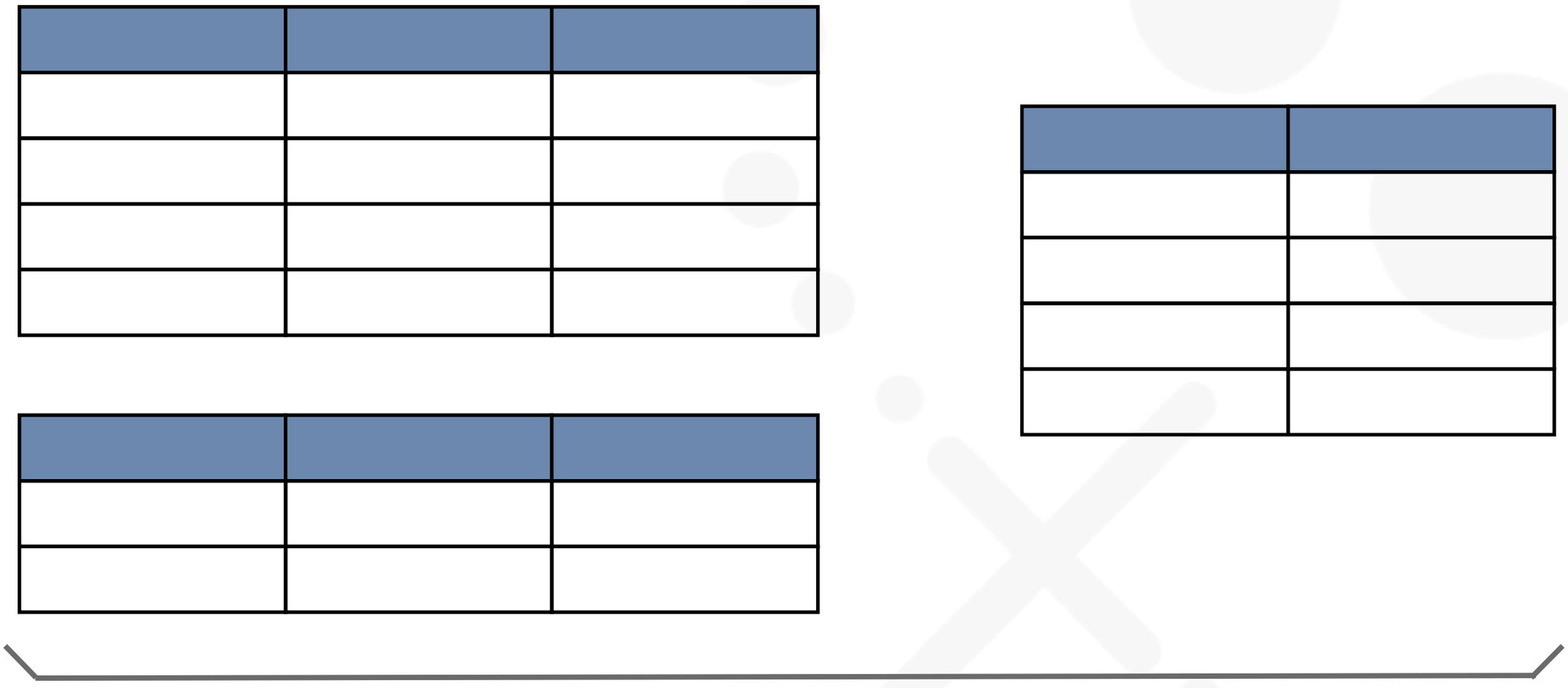
Columns = List of K/V pairs sorted by Key

Storage Engine



CQL3 Denormalized Model

CQL3 Query Language comes with a new Data Model abstraction made of denormalized, statically defined tables



Data duplicated over several tables

CQL3 Data Model

Timeline Table

user_id	tweet_id	author	body
gmason	1735	phenry	Give me liberty or give me death
gmason	1742	gwashington	I chopped down the cherry tree
ahamilton	1767	jadams	A government of laws, not men
ahamilton	1794	gwashington	I chopped down the cherry tree

Partition Key

Clustering Key

CQL3 Data Model

Timeline Table

user_id	tweet_id	author	body
gmason	1735	phenry	Give me liberty or give me death
gmason	1742	gwashington	I chopped down the cherry tree
ahamilton	1767	jadams	A government of laws, not men
ahamilton	1794	gwashington	I chopped down the cherry tree

CQL

```
CREATE TABLE timeline (
        user_id varchar,
        tweet_id timeuuid,
        author varchar,
        body varchar,
        PRIMARY KEY (user_id, tweet_id));
```

CQL3 Data Model

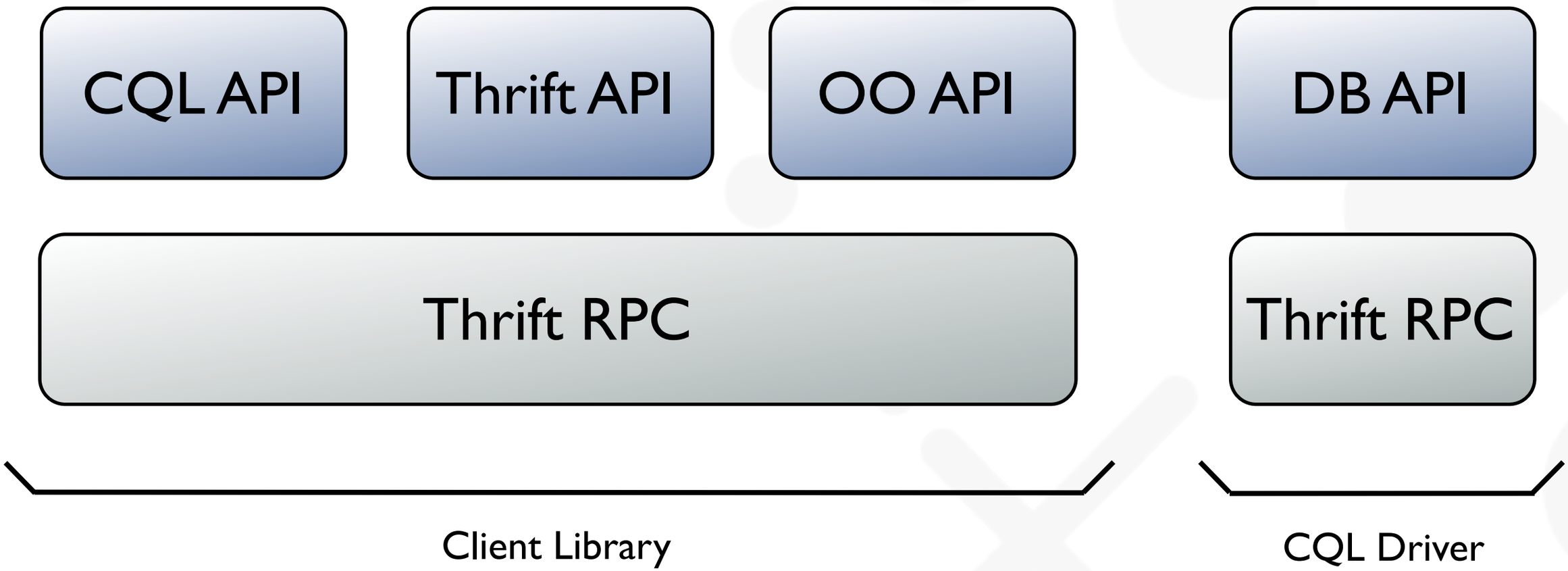
Timeline Table

user_id	tweet_id	author	body
gmason	1735	phenry	Give me liberty or give me death
gmason	1742	gwashington	I chopped down the cherry tree
ahamilton	1767	jadams	A government of laws, not men
ahamilton	1794	gwashington	I chopped down the cherry tree

Timeline Physical Layout

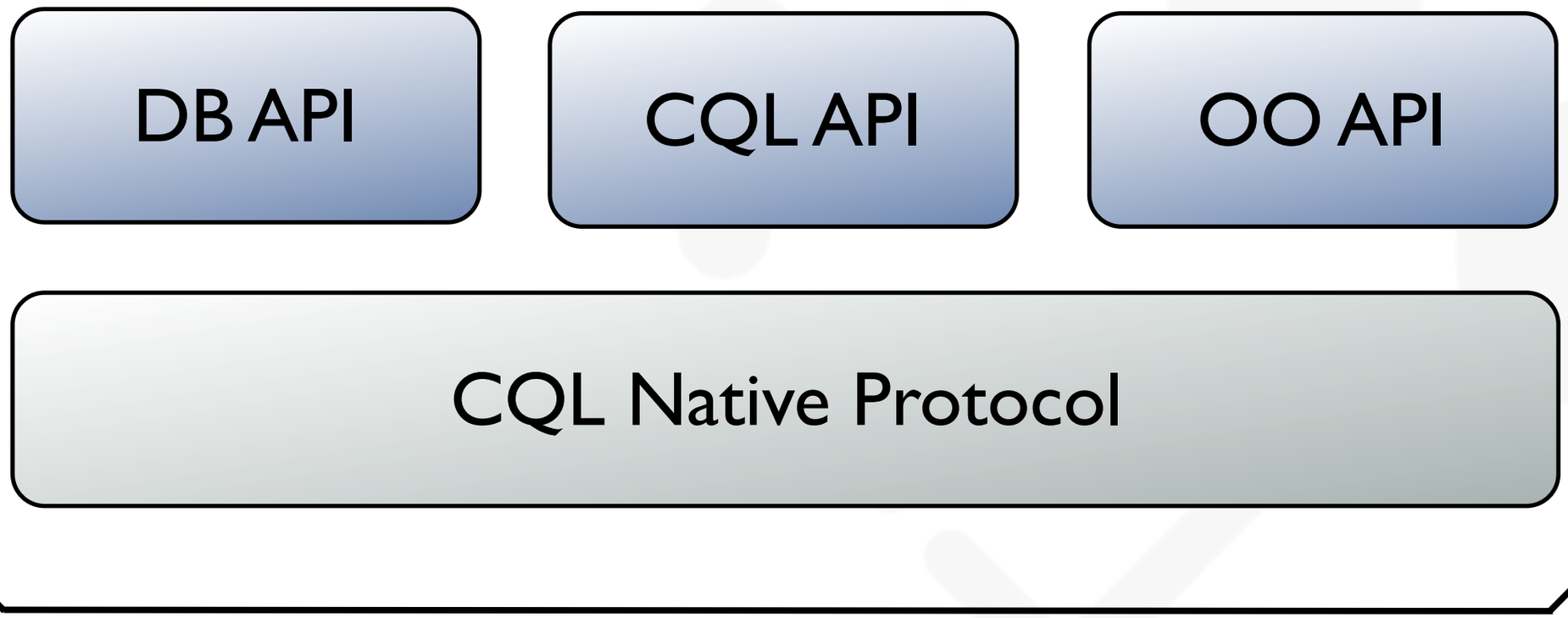
gmason	[1735, author]	[1735, body]	[1742, author]	[1742, body]
	gwashington	I chopped down the...	phenry	Give me liberty or give...
ahamilton	[1767, author]	[1767, body]	[1794, author]	[1794, body]
	gwashington	I chopped down the...	jadams	A government of laws...

Current Drivers Architecture



* This is a simplified view of drivers architecture. Details vary from one language to another.

New Drivers Architecture



* This is a simplified view of drivers architecture. Details vary from one language to another.

DataStax Java Driver

- Reference Implementation
- Asynchronous architecture based on Netty
- Prepared Statements Support
- Automatic Fail-over
- Node Discovery
- Cassandra Tracing Support
- Tunable policies
  - LoadBalancingPolicy
  - ReconnectionPolicy
  - RetryPolicy

Connect and Write

```
Cluster cluster = new Cluster.Builder()
                    .addContactPoints("10.0.0.1", "10.0.0.2")
                    .build();

Session session = cluster.connect("myKeyspace");

session.execute(
    "INSERT INTO user (user_id, name, email)
     VALUES (12345, 'johndoe', 'john@doe.com')"
);
```

Read

```
ResultSet rs = session.execute("SELECT * FROM test");

List<Row> rows = rs.fetchAll();

for (Row row : rows) {

   String userId = row.getString("user_id");
   String name = row.getString("name");
   String email = row.getString("email");
}
```

Asynchronous Read

```
ResultSet.Future future =
   session.executeAsync("SELECT * FROM test");

for (Row row : future.get()) {

   String userId = row.getString("user_id");
   String name = row.getString("name");
   String email = row.getString("email");
}
```

Object Mapping

```
@Table(name = "user")
public class User {

   @PartitionKey
   @Column(name = "user_id")
   private String userId;

   private String name;

   private String email;

   private Gender gender;
}
```

```
public enum Gender {

   @EnumValue("m")
   MALE,

   @EnumValue("f")
   FEMALE;
}
```

Inheritance

```
@Table(name = "catalog")
@Inheritance(
 subClasses = {Phone.class, TV.class},
 column = "product_type"
)
public abstract class Product {

   @Column(name = "product_id")
   private String productId;

   private float price;

   private String vendor;

   private String model;

}
```

```
@InheritanceValue("tv")
public class TV
extends Product {

   private float size;
}
```

Now Available!

https://github.com/datastax/java-driver

Stay Tuned!

blog.datastax.com

@mfiguiere

DATASTAX