Introduction to Cassandra

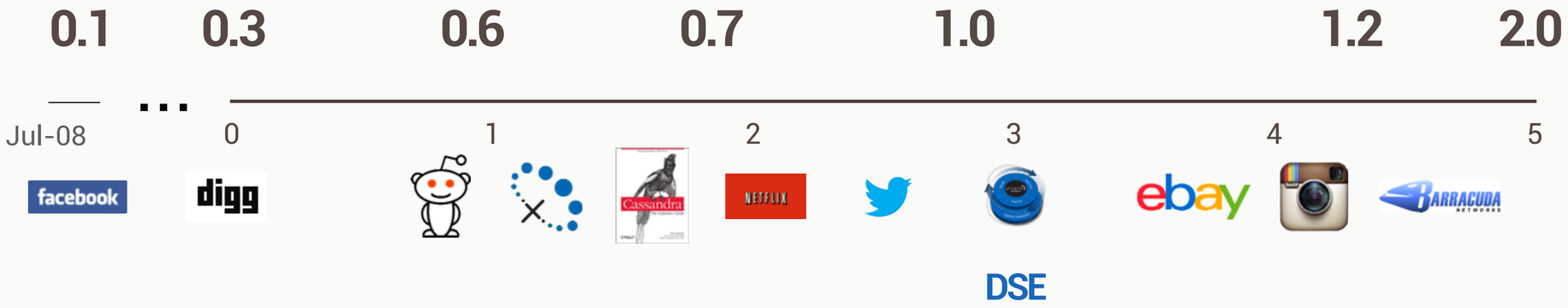
Patrick McFadin

Chief Evangelist/Solution Architect - DataStax

@PatrickMcFadin

Cassandra - An introduction

Five years of Cassandra



DATASTAX

1&1 · 1&1.com · [24]7 · 2lemetry · 46 LABS · 8digits · academicworks · accenture · ACTUATE · Acunu · adconion · AddThis · adform · adgear · Adobe · Aeris · AGENTIS · aiMatch

airpim · ALLIED · MYND · ALTOROS · AMARA · ampush · AMTEX · Answers.com · anubisnetworks · apereo · apigee · APISpark · APIXIO · AppDynamics · applifier · AppScale · AppsFlyer

Apptegic · Arise · ARKIVUM · ARRAYENT · AURELIUS · POST · avanade · avast! · Avira · AZUL SYSTEMS · backupify · BARRACUDA NETWORKS · base · be-mobile · BDOSS · BlueMountain

boundary · BOXEVER · BROCADE · Bronto · Burt. · bwin.party · C3 Energy · CABLEVISION · CABOT · CAKESOLUTIONS · CallFire · canary · CANONICAL · Captora · cardSPRING · CARTERA

caserta · CERN · chaordic · CharityUSA.com · CHARTCUBE · Chegg · Chill · Cinnober · CISCO · citypath · clearleap · clearspring · Cloudian · cloudkick · CloudPassage · cloudsoft · CLOUDTALK · cloze · club

CODEMETTLE · Cognitum · comcast · COMPUSCAN · Compuware · comverge · Constant Contact · cornastone · correlor · Crowd Riff · CSR · CURALATE · CXENSE · DachisGroup · DATADIO · DATADOG · dbsync

DELL · Demandforce · DeNA · despegar.com · digby · digg · Digital Reasoning · Digital River · Disney · DISQUS · docomo · DRILLSTER · druva · Dyn · EARLY WARNING · easou · ebay · bros

buddy · ecomnext · egnyte · elasticinbox · embedly · eNanoPay · ENGINE · Engine Yard · enkata · epsilon · EQUINIX · ERICSSON · EUROTECH · Eventbrite · exoscale · Expedia · FACE · FanBridge · fashiolista

feedzai · FINN · flipkart · fold3 · formspring · Frugal Mechanic · F-Secure · FullContact · GAMEFLY · Gavagai · Gazzang · GE · gemalto · GENERAL SENTIMENT · GIGASPACES · glassbeam

glispa · Globant · GlocalSpace · glu · GNIP · GoDaddy · GOGRID · goingon · GoSquared · GOTIME · gotosolr · GRAINGER · Grid Dynamics · GRIDPOINT · Groupalia · grubHub · GX · HAIL · Hangtime · has offers

HEALTH MARKET SCIENCE · HealthCare ANYTIME · Healthline · HealthX · hobsons · HOUGHTON MIFFLIN HARCOURT · hp · HubSpot · hulu · i2O WATER · IBM · iGivefirst · IGNITIONONE · IMPETUS · Infinite Convergence · inkling · instaclustr · Instagram · inStream Media

Integral Ad Science · Intelligent Systems Group · inteluma · IHG · IID · intuit · InWorldz · iovation · ISIDOREY · iSIGHTPARTNERS · ITERIS · ITHAKA · IXONOS · JASPERSOFT · JBoss · jetpac · Jini · Jumptap · JUNIPER NETWORKS

justSignal · KASPERSKY · Keen IO · KISSmetrics · KNEWTON · krux · kweo · LA POSTE · lacuna SYSTEMS · LeaseEagle · leaseweb · Libon · librato · Life360 · LIFESTREET media · lightt · Lingotek

liquidweb · Lithium · LiveOffice · LIVEPERSON · lohika · looplogic · lucid · Mahalo · MARKED UP · martinimedia · Mass Relevance · mate1.com · media6degrees · MediaCrossing · MEDIALETS · medidata · medio · MedProcure

Melt · memeni · meta broadcast · midokura · Mind Candy · MIRANTIS · mnubo · MolloM · monitis · Monty Program · moodyo · MORNINGSTAR · movile · MuleSoft · mydrive solutions · nanigans · napshare · NASA · nativeX · NAVTEQ

nearbuy · NETFLIX · NetSeer · neustar · newegg.com · NEWS WHIP · NEXGATE · NEXT BIG SOUND · Nextag · ngmoco:) · NICE TO WEB YOU · NREL · nSCALED · nToklo · NUTANIX · NYSE TECHNOLOGIES · одноклассники · ONEsite · one spot

onesto · onLive · onsip · OOYALA · OSI · OpenAdExchange · OpenMarket · openNMS · opensource connections · opentracker · OPENWAVE · OpenX · opsmatic · orange · Other Inbox · outbrain · PACE ANTI PIRACY · page·rss

Palantir · PANTHEON · paper.li · PAPERKUT · Parse · parseix · PBS · PCPhase · PDF SOLUTIONS · PEARSON · PELOTON · pentaho · PeopleBrowsr · PERFECT WORLD · PETER SERVICE · Ping Identity · ping4alerts! · PitneyBowes · Pivotal · PK ONLINE

platform46 · plaxo · POLYVORE · PRESS+ · Pressjack · priceline.com · pring · EPIC · proofpoint · PROS · ProtectWise · PULSAR · qafe · QUALOGY · Qualtera · quandl · quantcast

quantiFind · QUOVA · rackspace · Rakuten · RALLY · raptr · RAZORlabs · ReachLocal · real · Recruiting.com · RED BRICK MEDIA · reddit · referentia · Reltio · resultly · retailigence · re tickr · Rhapsody · RIGHTSCALE

riptide io · RockMelt · rockyou · Ruckus · Sabre · SAFEWAY · SanDisk · SBR · SBRnet · SCALE UNLIMITED · Scandit · Scoop.it! · screen9 · SeaChange · SecondSync · see | me

SEEMOTO · sensorly · SEOmoz · session m · SEVEN · sevocity · Shareablee · ShareThis · shazam · SHIFT · SHODAN · shopsavvy · shore.li · SimpleEnergy · SimpleGeo · simple reach · singlewire

SIQUERIES · sirca · SITA · sitevision · СКБ Контур · skillpages · sky · smeet · SMX · SocialArtisan · SOCIALFLOW · SOCIALPARENT · Software Projects · Sonico · SOUNDCLOUD · SourceNinja · SPAZIODATI · spidertracks

splunk · Spotify · spotright · spotXchange · Spredfast · Springer · springfuse · springpad · SPROUTSOCIAL · SQUIDOO · STACKDRIVER · Startapp · Stipple · Stormpath · Stratio · Sirmur · strug · sumologic

SUNCORP · SVENSKA SPEL · SwiftKey · Symantec · SYNAPSE · Synchronoss · Taboola · talentica · talkbits · Tapulous · TEADS · teamfocus · Telefonica · TENDRIL · echonest · thefind · THE FRONTSIDE · THE GLOBE AND MAIL

THE LINKBUILDERS · The Weather Channel · thinglink · THOMSON REUTERS · TILDE · tink · TinyCo · trapit · travelocity · TRICAST · TRIPLE POINT · trivadis · trueffect · tuplejump · Tutao · TWENGA · TWIMPACT

twitpic · twitter · Ubiet · UBISOFT · ULTRAVISUAL · UNBXD · URBAN AIRSHIP · V2i · ValueClick media · Vast · veespo · Veezio · verticalscope · ViaSat · videoplaza · VigLink · VisualDNA · volly · VONJOUR

VTEX · W2O · WaJam · Walmart · wayin · Webcollage · WE · Where · Whisper · WIDESPACE · wiggle · WILLIAMS-SONOMA · WIZECOMMERCE · WORKWARESYSTEMS · WSO2 · Wupa · Xceed · Xebia

xobni · Yakaz · Yomoneo · YUNIO · zangbezang · zeebox · Zemanta · zimp · zipwhip · ZipZapPlay · zoemob · ZOHO · ZONAR · ZoneFox · zoominfo · ZOOSK · zorallabs · zscaler

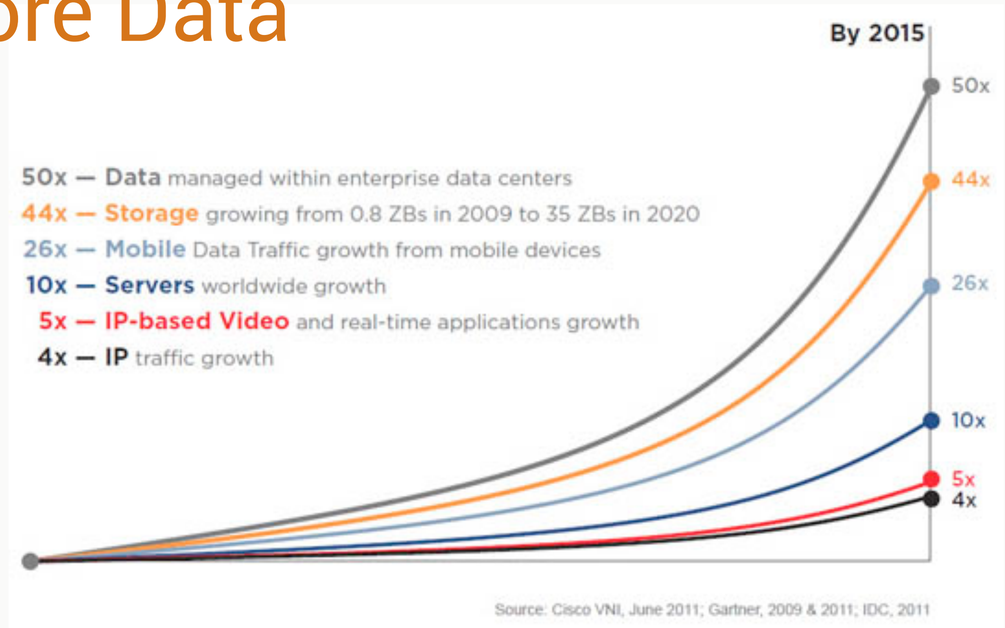
Why Cassandra?

Or any non-relational?

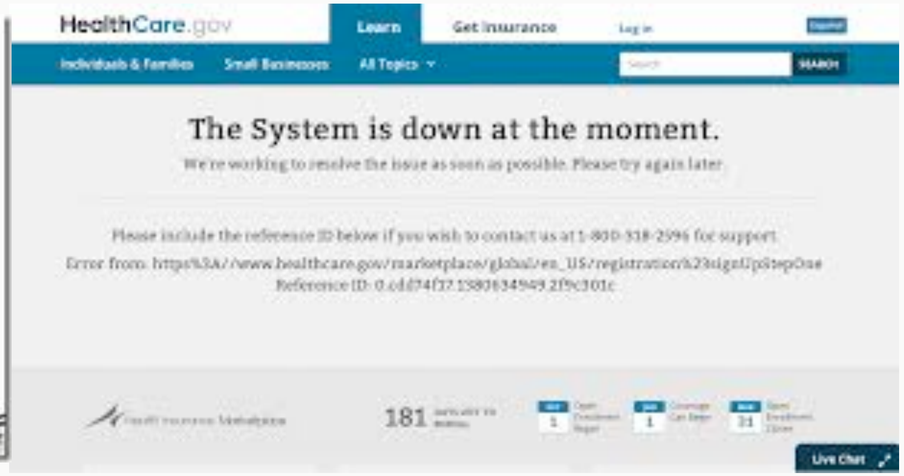
The world has changed

More Connected

More Data



Less Tolerant



For the record. Can you spell HTTP?

Traditional solutions…

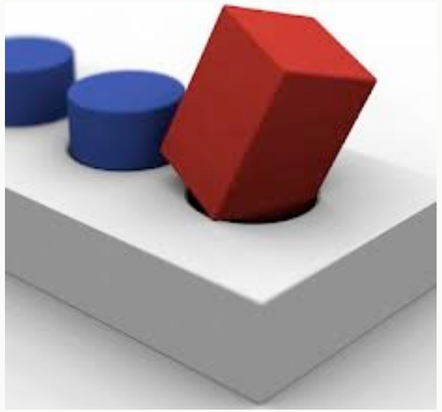
Microsoft® SQL Server®

IBM DB2

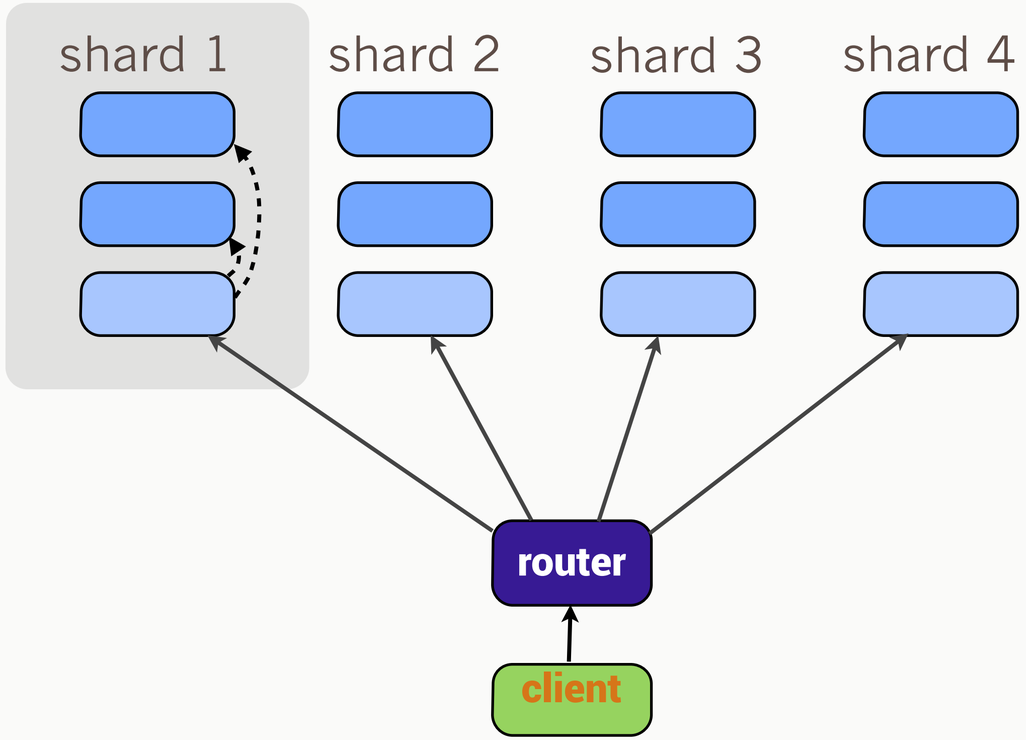
MySQL®

PostgreSQL

…may not be a good fit



We still try though



A new plan

Let's apply a little Computer Science

Dynamo Paper(2007)

DATASTAX

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

1. INTRODUCTION

Amazon runs a world-wide e-commerce platform that serves tens of millions customers at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Amazon's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable. Reliability is one of the most important requirements because even the slightest outage has significant financial consequences and impacts customer trust. In addition, to support continuous growth, the platform needs to be highly scalable.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance. Amazon's platform has a very diverse set of applications with different storage requirements. A select set of applications requires a storage technology that is flexible enough to let application designers configure their data store appropriately based on these tradeoffs to achieve high availability and guaranteed performance in the most cost effective manner.

There are many services on Amazon's platform that only need primary-key access to a data store. For many services, such as those that provide best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog, the common pattern of using a relational database would lead to inefficiencies and limit scale and availability. Dynamo provides a simple primary-key only interface to meet the requirements of these applications.

Dynamo uses a synthesis of well known techniques to achieve scalability and availability: Data is partitioned and replicated using consistent hashing [10], and consistency is facilitated by object versioning [12]. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Dynamo employs

SOSP'07, October 14–17, 2007, Stevenson, Washington, USA.

- How do we build a data store that is:
  - Reliable
  - Performant
  - "Always On"
- Nothing new and shiny
- 24 papers cited

amazon.com

Evolutionary. Real. Computer Science

Also the basis for Riak and Voldemort

BigTable(2006)

DATASTAX

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) → string

To appear in OSDI 2006

- Richer data model
- 1 key. Lots of values
- Fast sequential access
- 38 Papers cited

Google™

Cassandra(2008)

DATASTAX

Cassandra - A Decentralized Structured Storage System

Avinash Lakshman
Facebook

Prashant Malik
Facebook

ABSTRACT

Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. Cassandra aims to run on top of an infrastructure of hundreds of nodes (possibly spread across different data centers). At this scale, small and large components fail continuously. The way Cassandra manages the persistent state in the face of these failures drives the reliability and scalability of the software systems relying on this service. While in many ways Cassandra resembles a database and shares many design and implementation strategies therewith, Cassandra does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format. Cassandra system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency.

1. INTRODUCTION

Facebook runs the largest social networking platform that serves hundreds of millions users at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Facebook's platform in terms of performance, reliability and efficiency, and to support *continuous growth* the platform needs to be highly scalable. Dealing with failures in an infrastructure comprised of thousands of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such, the software systems need to be constructed in a manner that treats failures as the norm rather than the exception. To meet the reliability and scalability needs described above Facebook has developed Cassandra.

Cassandra uses a synthesis of well known techniques to achieve scalability and availability. Cassandra was designed to fulfill the storage needs of the Inbox Search problem. Inbox Search is a feature that enables users to search through their Facebook Inbox. At Facebook this meant the system was required to handle a very high write throughput, billions of writes per day, and also scale with the number of users. Since users are served from data centers that are geographically distributed, being able to replicate data across data centers was key to keep search latencies down. Inbox Search was launched in June of 2008 for around 100 million users and today we are at over 250 million users and Cassandra has kept up the promise so far. Cassandra is now deployed as the backend storage system for multiple services within Facebook.

This paper is structured as follows. Section 2 talks about related work, some of which has been very influential on our design. Section 3 presents the data model in more detail. Section 4 presents the overview of the client API. Section 5 presents the system design and the distributed algorithms that make Cassandra work. Section 6 details the experiences of making Cassandra work and refinements to improve performance. In Section 6.1 we describe how one of the applications in the Facebook platform uses Cassandra. Finally Section 7 concludes with future work on Cassandra.

2. RELATED WORK

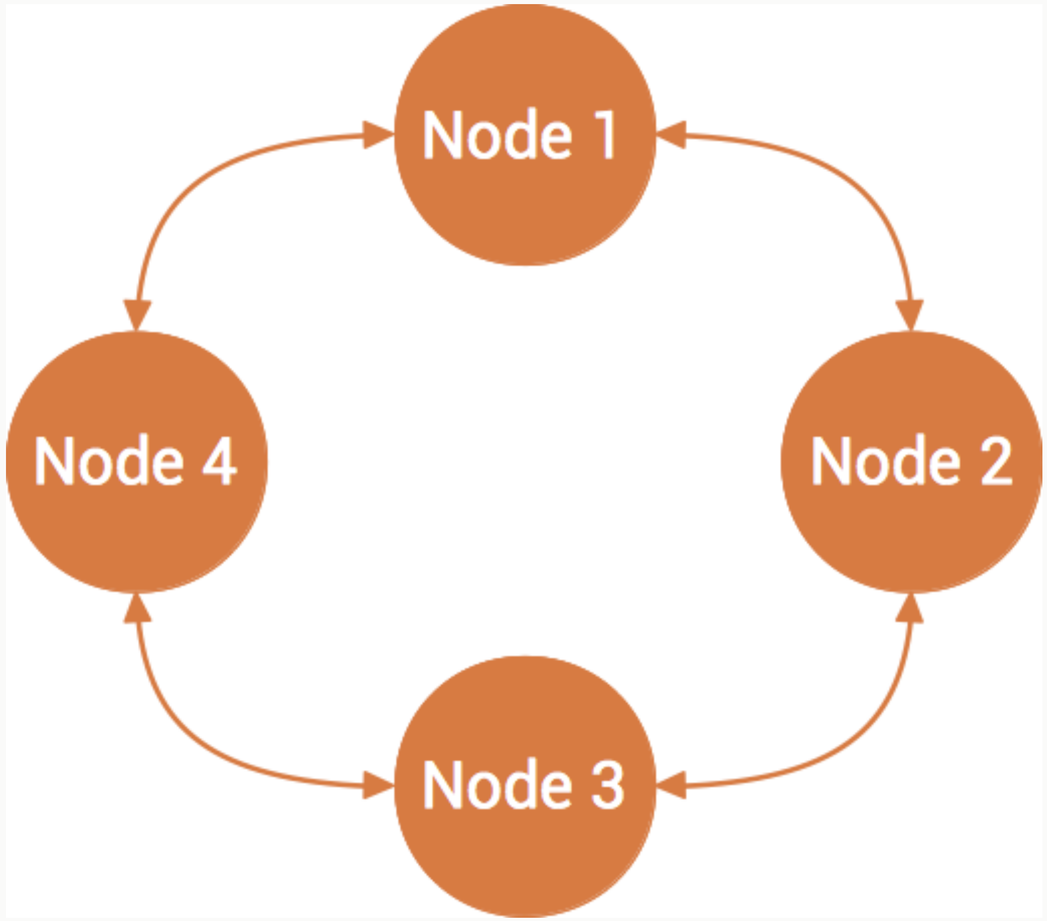
Distributing data for performance, availability and durability has been widely studied in the file system and database communities. Compared to P2P storage systems that only support flat namespaces, distributed file systems typically support hierarchical namespaces. Systems like Ficus[14] and Coda[16] replicate files for high availability at the expense of consistency. Update conflicts are typically managed using specialized conflict resolution procedures. Farsite[2] is a distributed file system that does not use any centralized server. Farsite achieves high availability and scalability using replication. The Google File System (GFS)[9] is another distributed file system built for hosting the state of Google's internal applications. GFS uses a simple design with a single master server for hosting the entire metadata and where the data is split into chunks and stored in chunk servers. However the GFS master is now made fault tolerant using the Chubby[3] abstraction. Bayou[18] is a distributed relational database system that allows disconnected operations and provides eventual data consistency. Among these systems, Bayou, Coda and Ficus allow disconnected operations and are resilient to issues such as network partitions and outages. These systems differ on their conflict resolution procedures. For instance, Coda and Ficus perform system level conflict resolution and Bayou allows application level

- Distributed features of Dynamo
- Data Model and storage from BigTable
- February 17, 2010 it graduated to a top-level Apache project

Cassandra - More than one server

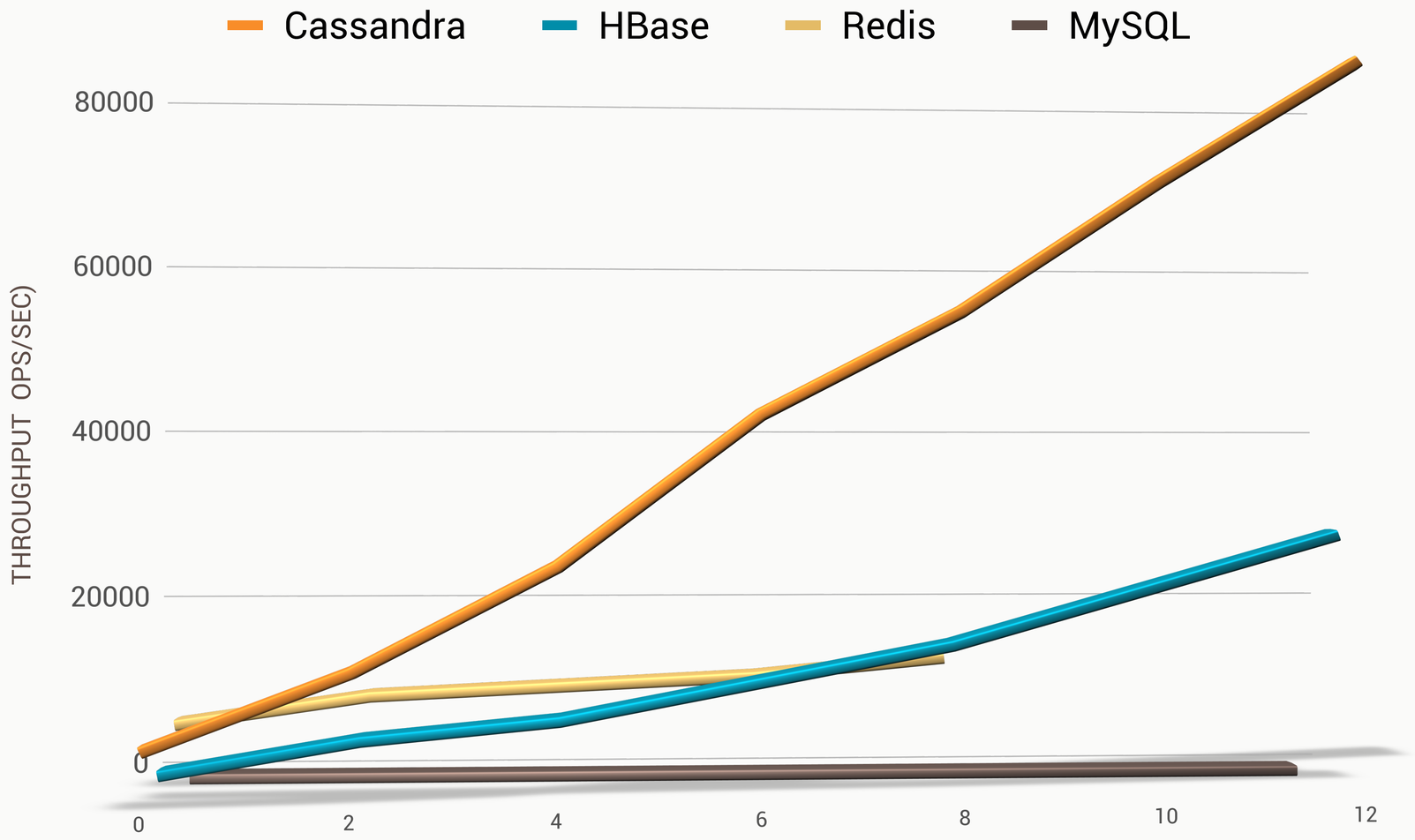
DATASTAX

- All nodes participate in a cluster
- Shared nothing
- Add or remove as needed
- More capacity? Add a server



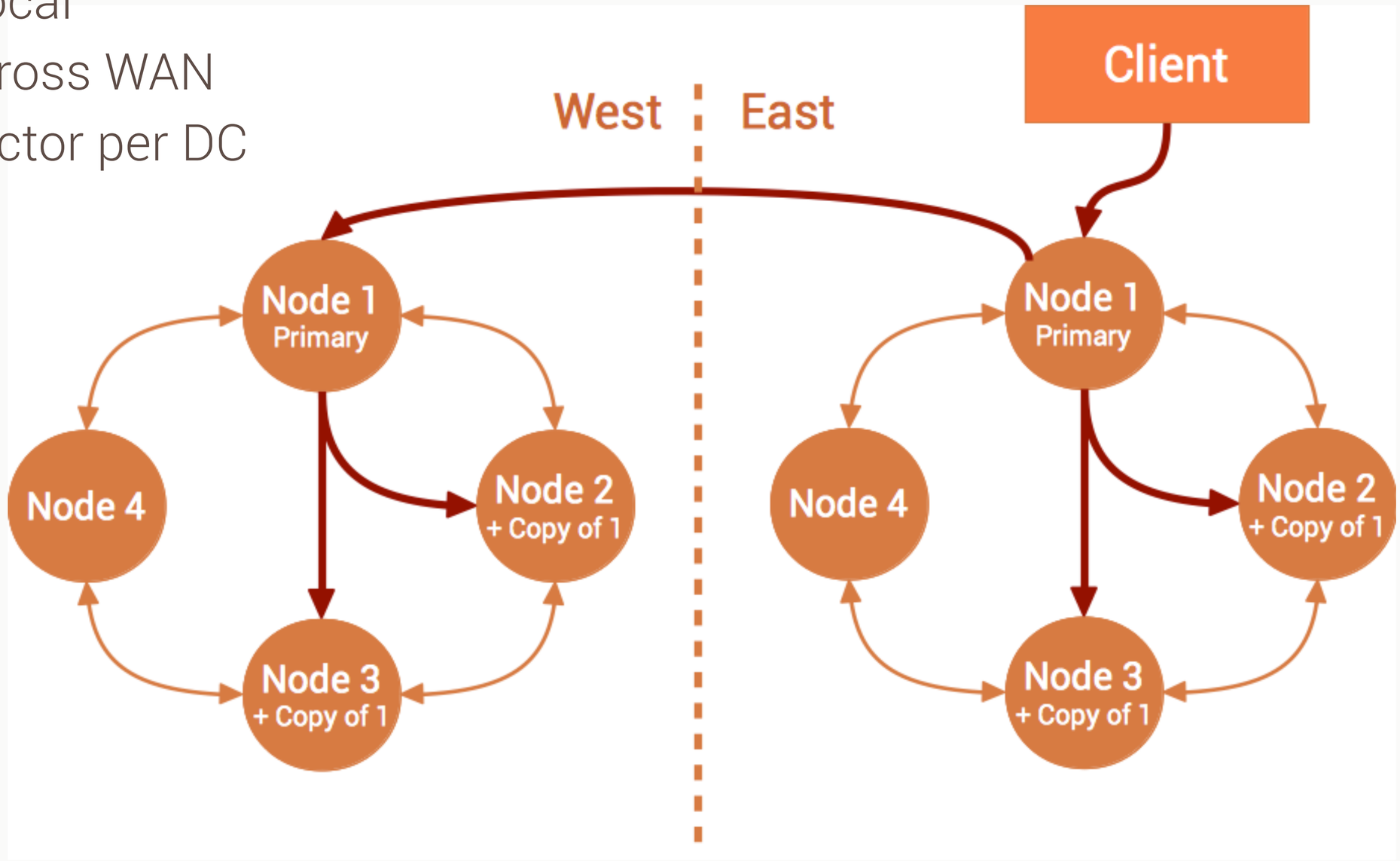
DATASTAX

VLDB benchmark (RWS)

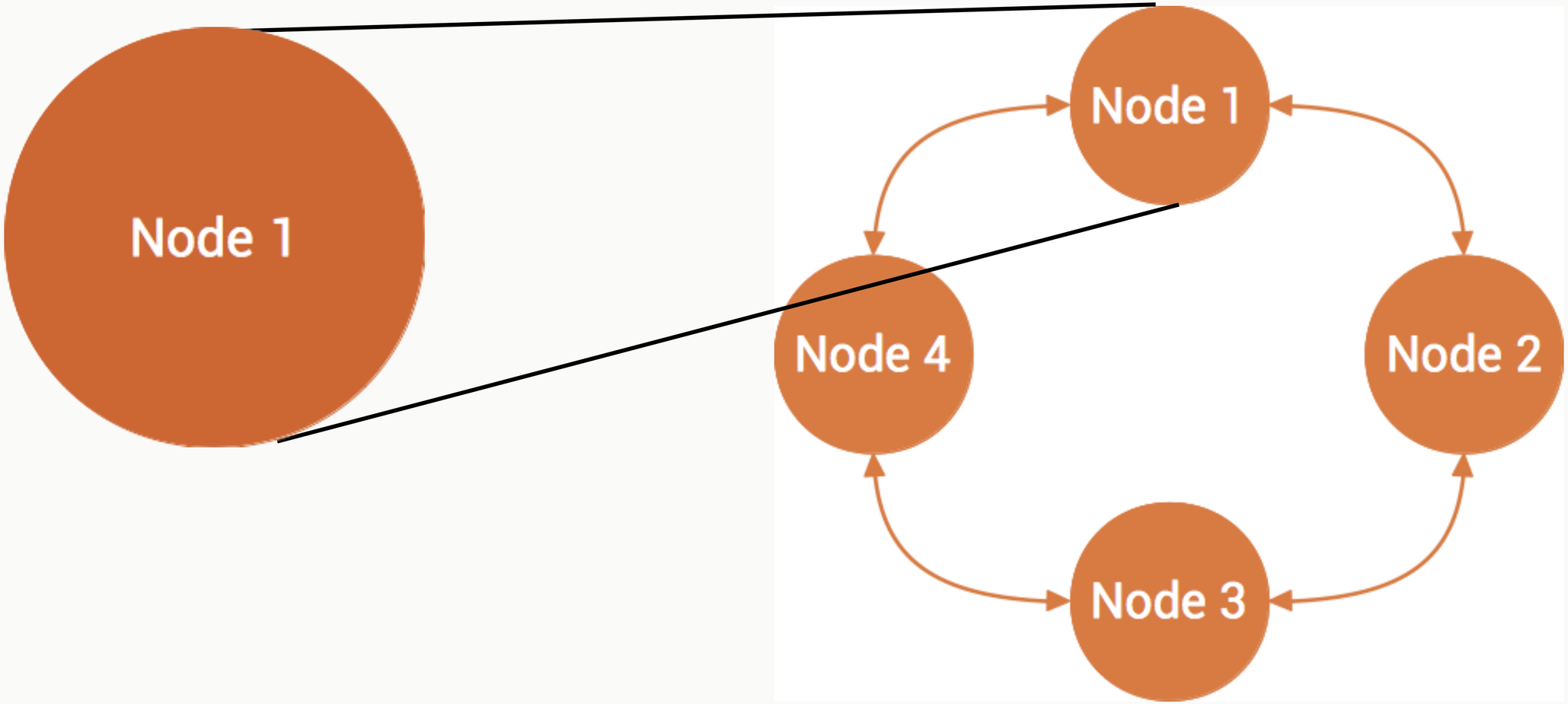


Cassandra - Fully Replicated

- Client writes local
- Data syncs across WAN
- Replication Factor per DC

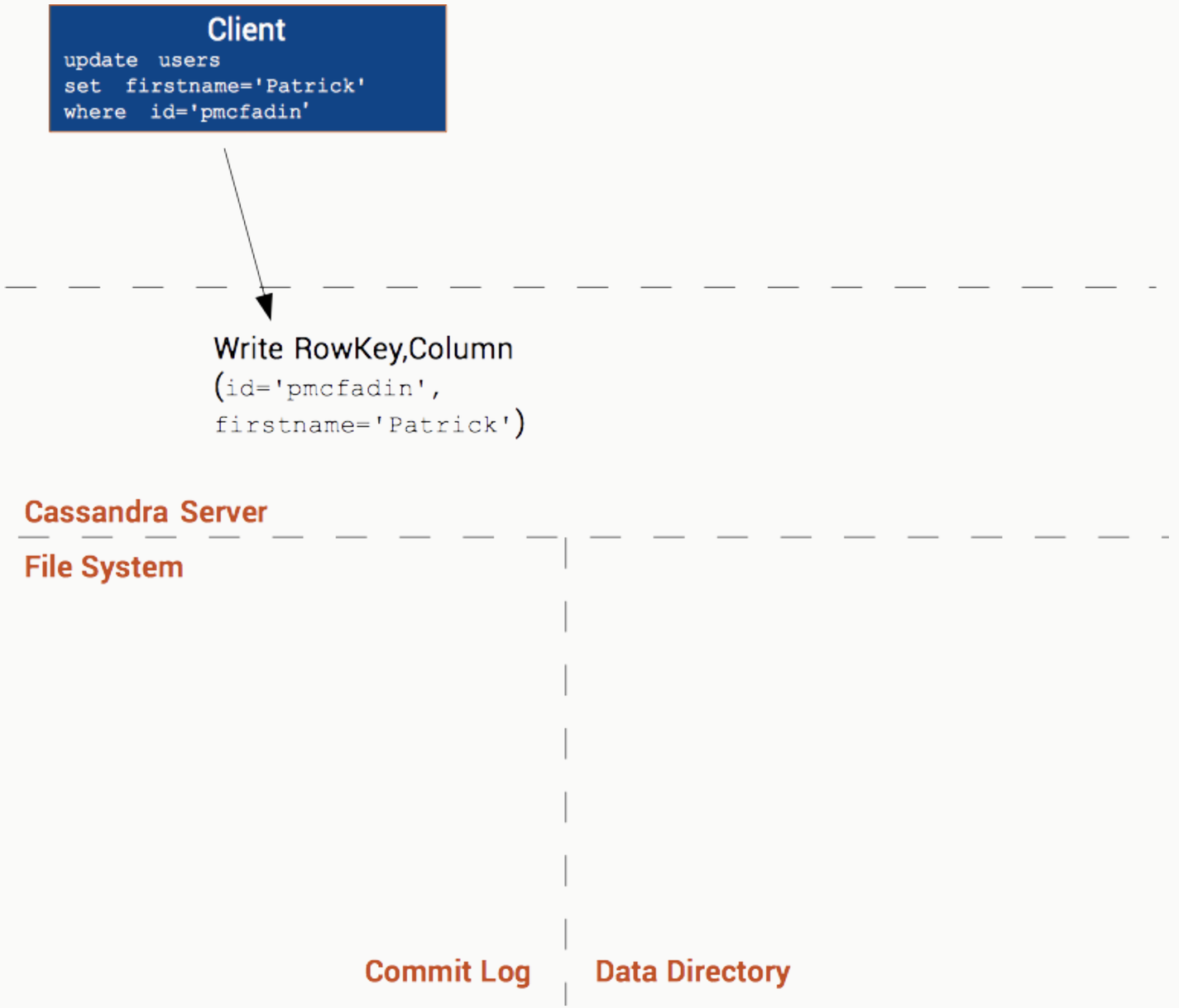


Focus on a single server

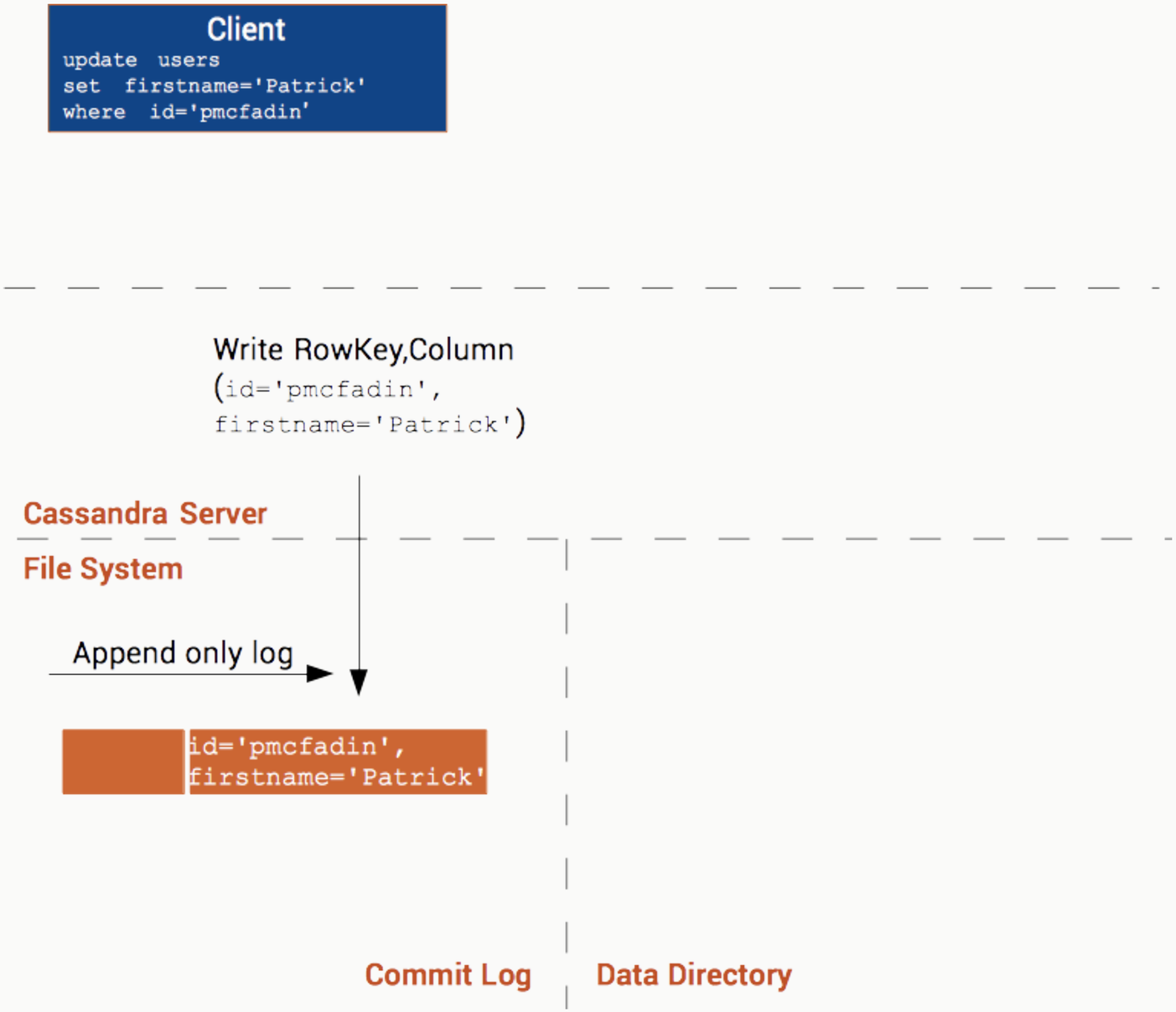


Cassandra - Writes on a single node

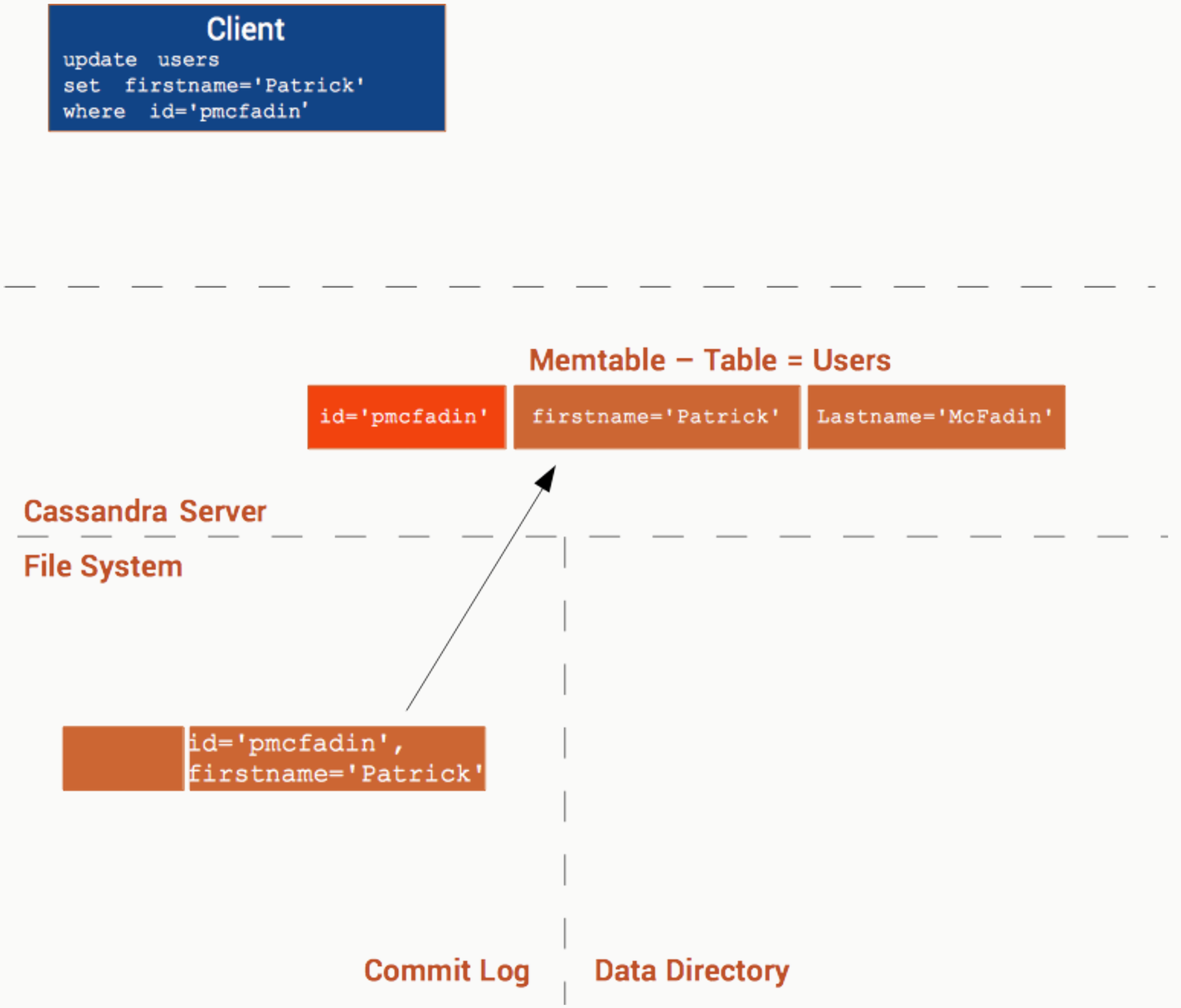
Client writes to server



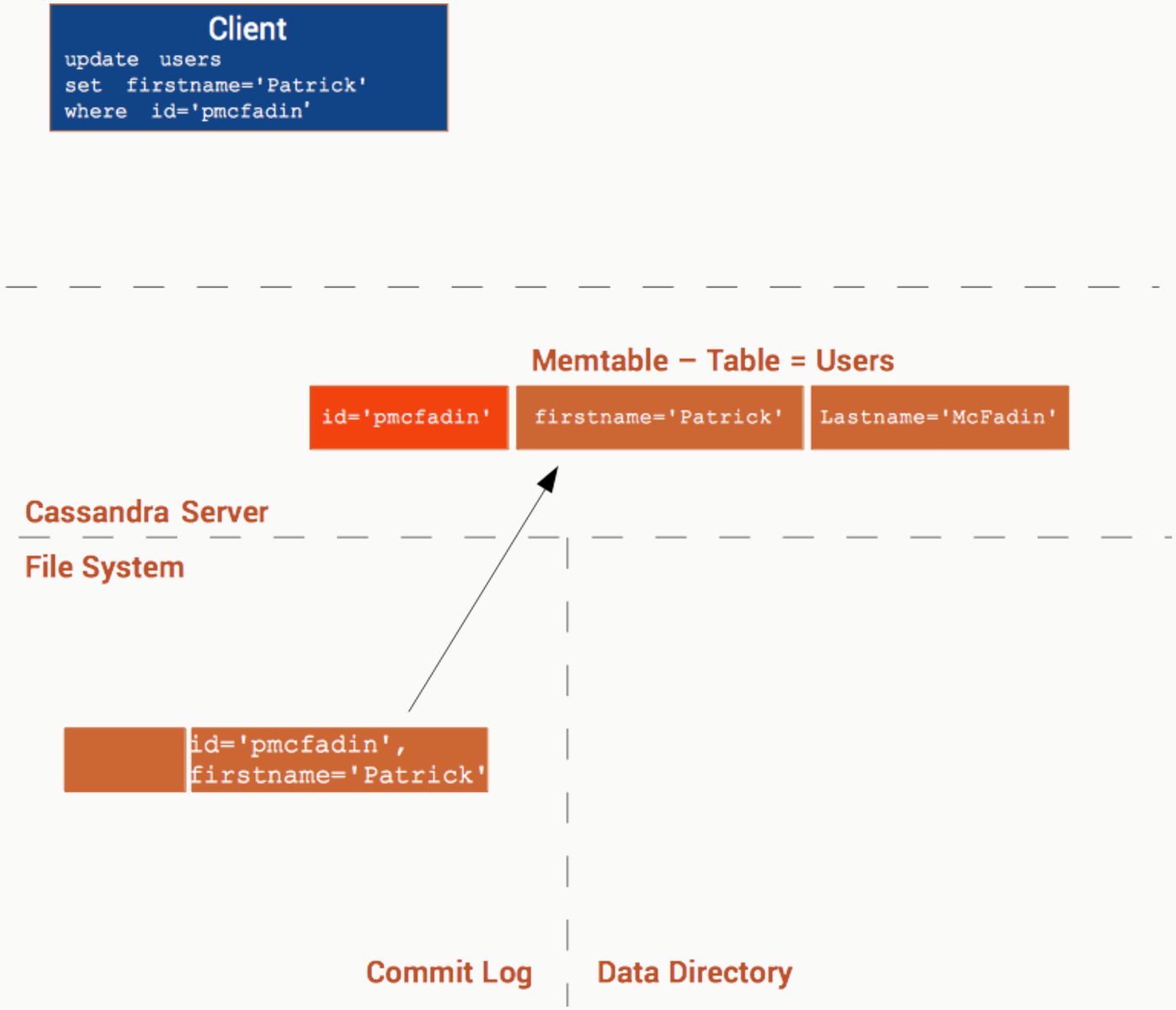
Data written to commit log



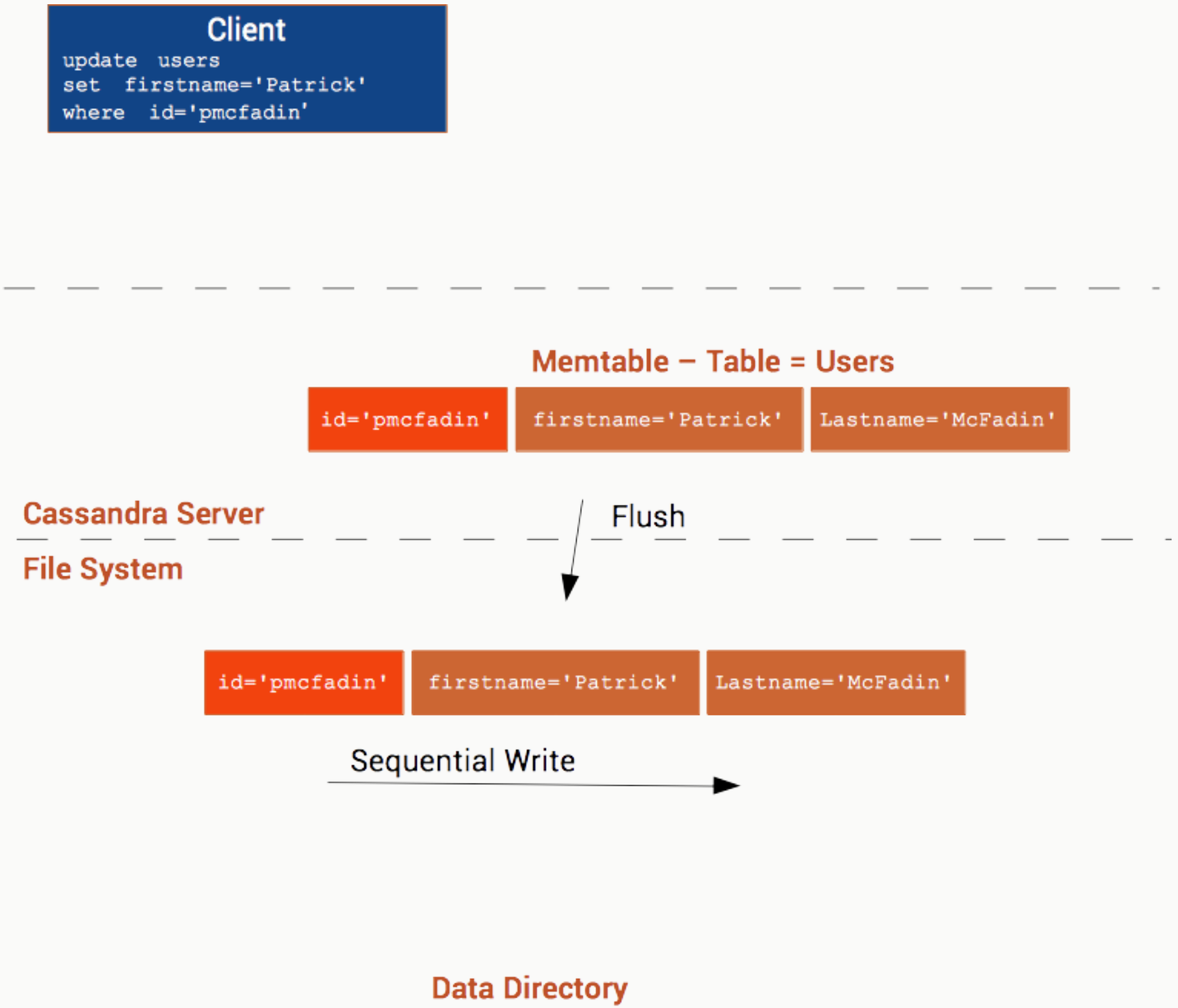
Data written to memtable



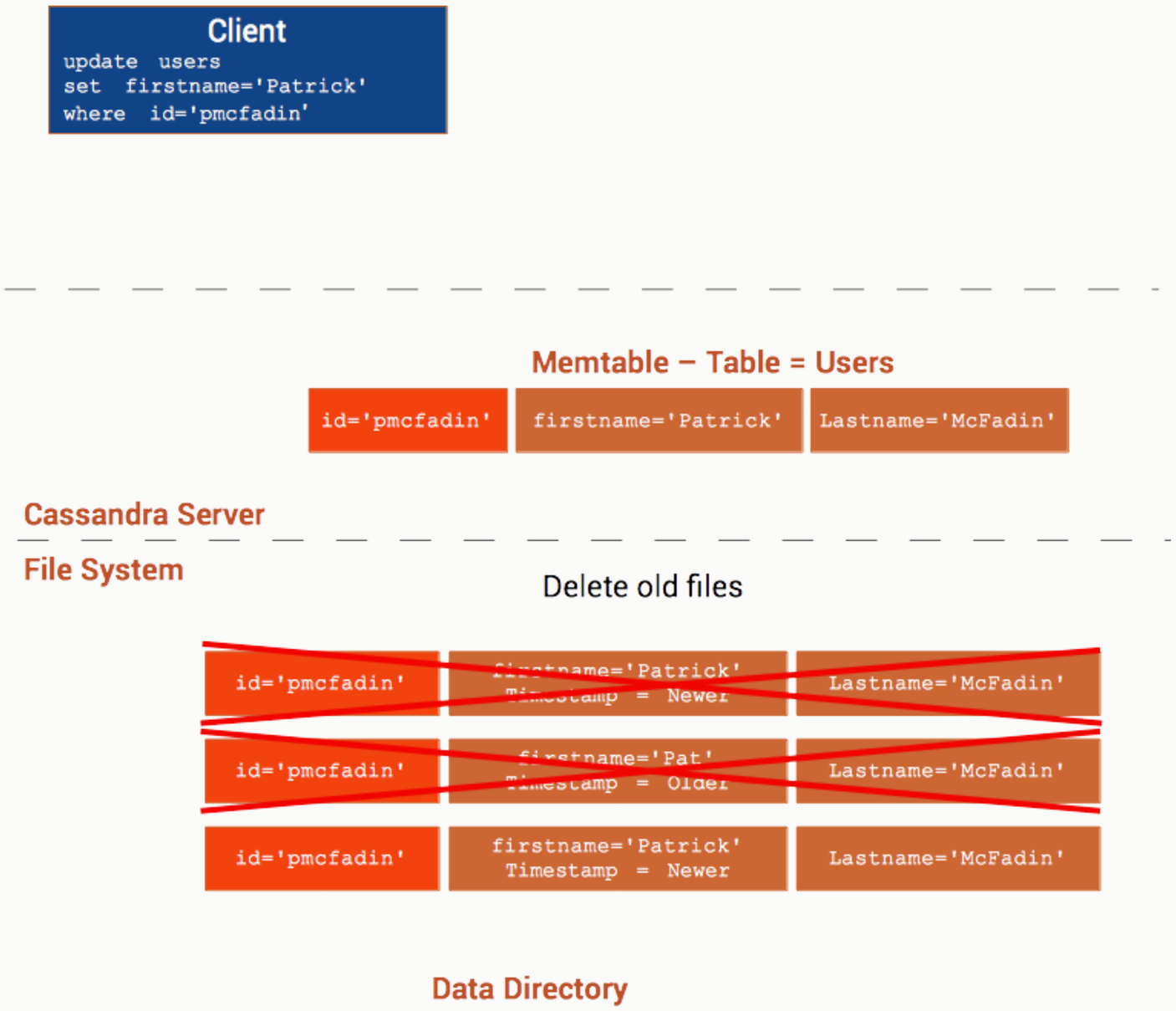
Server acknowledges to client



Sequential file write



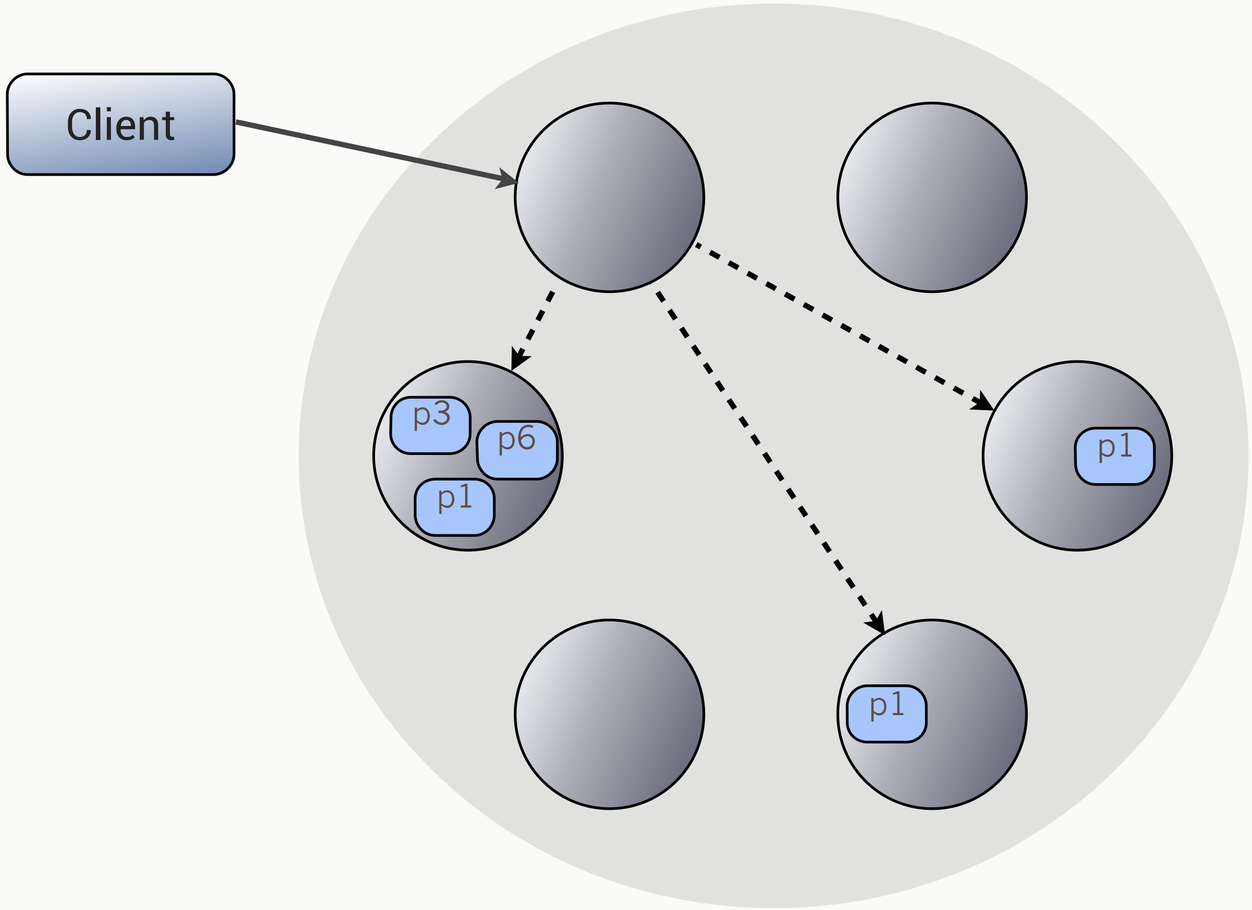
Compaction


Client
update users
set firstname='Patrick'
where id='pmcfadin'
Memtable – Table = Users
id='pmcfadin'
firstname='Patrick'
Lastname='McFadin'
Cassandra Server
File System
Delete old files
id='pmcfadin'
Timestamp = Newer
Lastname='McFadin'
id='pmcfadin'
Timestamp = Older
Lastname='McFadin'
id='pmcfadin'
firstname='Patrick'
Timestamp = Newer
Lastname='McFadin'
Data Directory

Cassandra - Writes in the cluster

Fully distributed, no SPOF

DATASTAX



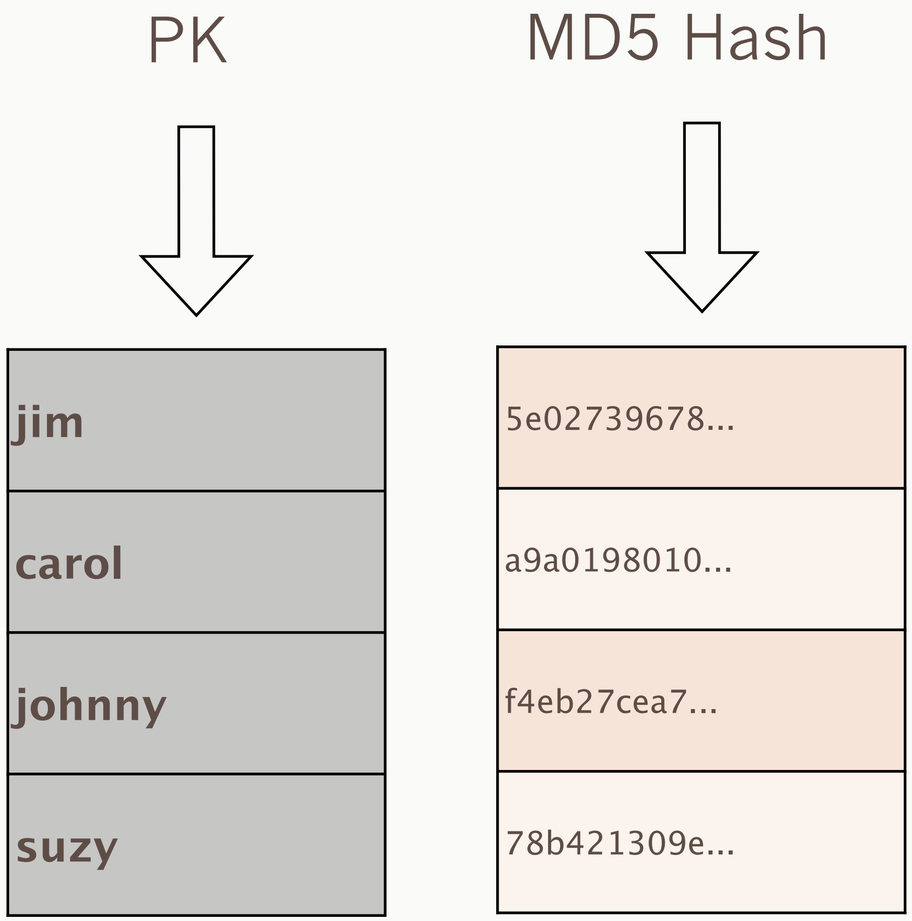
Partitioning

DATASTAX

Primary key determines placement*

jim	age: 36	car: camaro	gender: M
carol	age: 37	car: subaru	gender: F
johnny	age:12	gender: M	
suzy	age:10	gender: F	

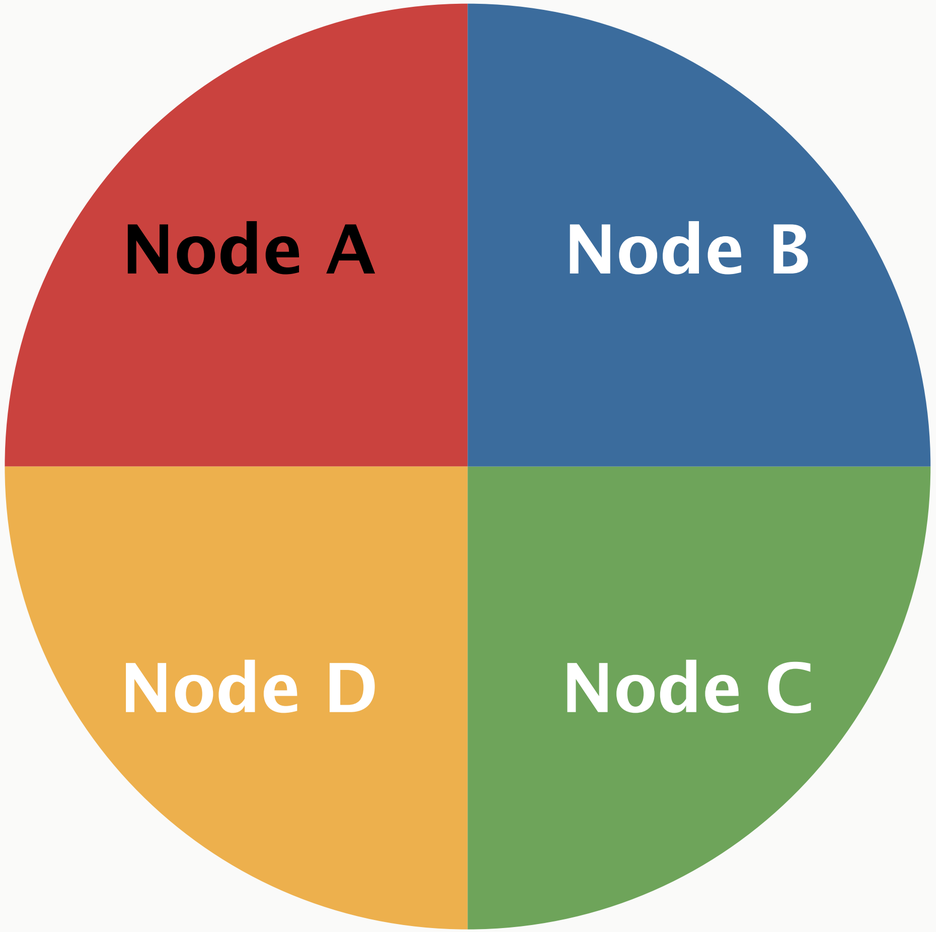
Key Hashing



MD5* hash operation yields a 128-bit number for keys of any size.

The Token Ring

DATASTAX



DATASTAX

	Start	End
A	0xc000000000..1	0x0000000000..0
B	0x0000000000..1	0x4000000000..0
C	0x4000000000..1	0x8000000000..0
D	0x8000000000..1	0xc000000000..0

jim	5e02739678...
carol	a9a0198010...
johnny	f4eb27cea7...
suzy	78b421309e...

DATASTAX

	Start	End
A	0xc000000000..1	0x0000000000..0
B	0x0000000000..1	0x4000000000..0
C	0x4000000000..1	0x8000000000..0
D	0x8000000000..1	0xc000000000..0

jim	5e02739678...
carol	a9a0198010...
johnny	f4eb27cea7...
suzy	78b421309e...

DATASTAX

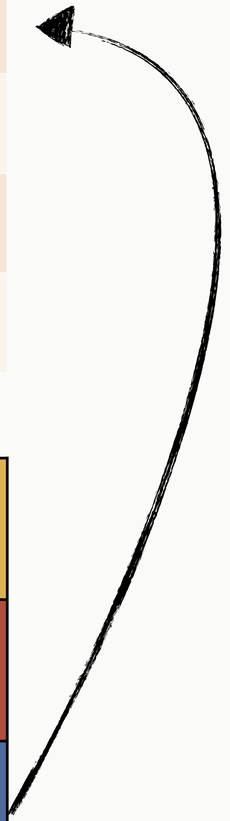
	Start	End
A	0xc000000000..1	0x0000000000..0
B	0x0000000000..1	0x4000000000..0
C	0x4000000000..1	0x8000000000..0
D	0x8000000000..1	0xc000000000..0

jim	5e02739678...
carol	a9a0198010...
johnny	f4eb27cea7...
suzy	78b421309e...

DATASTAX

	Start	End
A	0xc000000000..1	0x0000000000..0
B	0x0000000000..1	0x4000000000..0
C	0x4000000000..1	0x8000000000..0
D	0x8000000000..1	0xc000000000..0

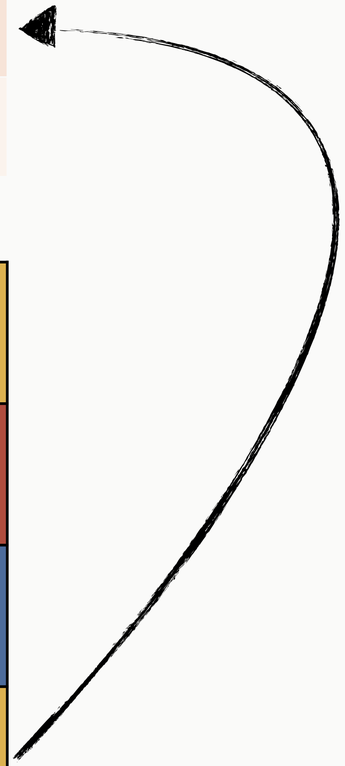
jim	5e02739678...
carol	a9a0198010...
johnny	f4eb27cea7...
suzy	78b421309e...



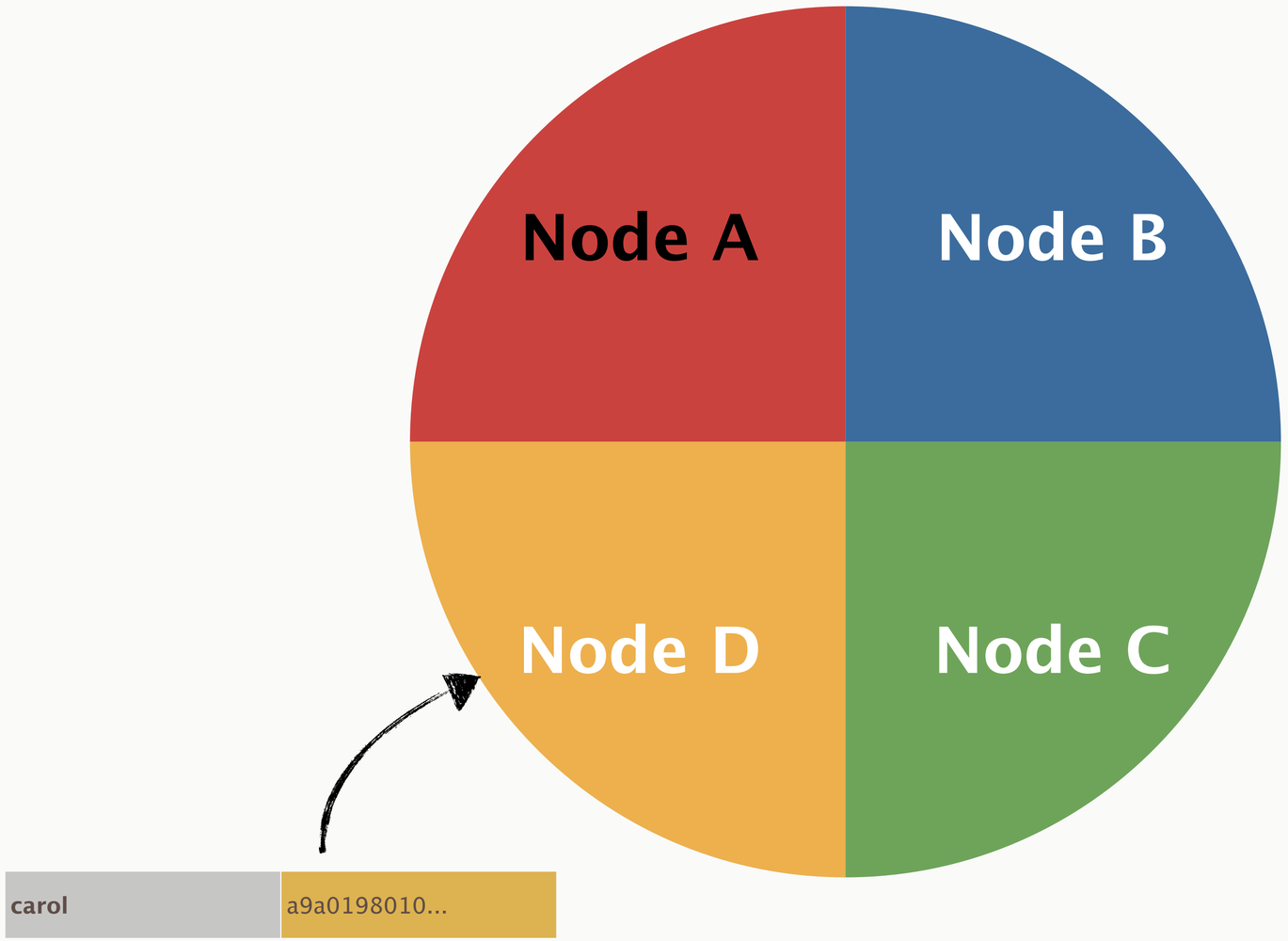
DATASTAX

	Start	End
A	0xc000000000..1	0x0000000000..0
B	0x0000000000..1	0x4000000000..0
C	0x4000000000..1	0x8000000000..0
D	0x8000000000..1	0xc000000000..0

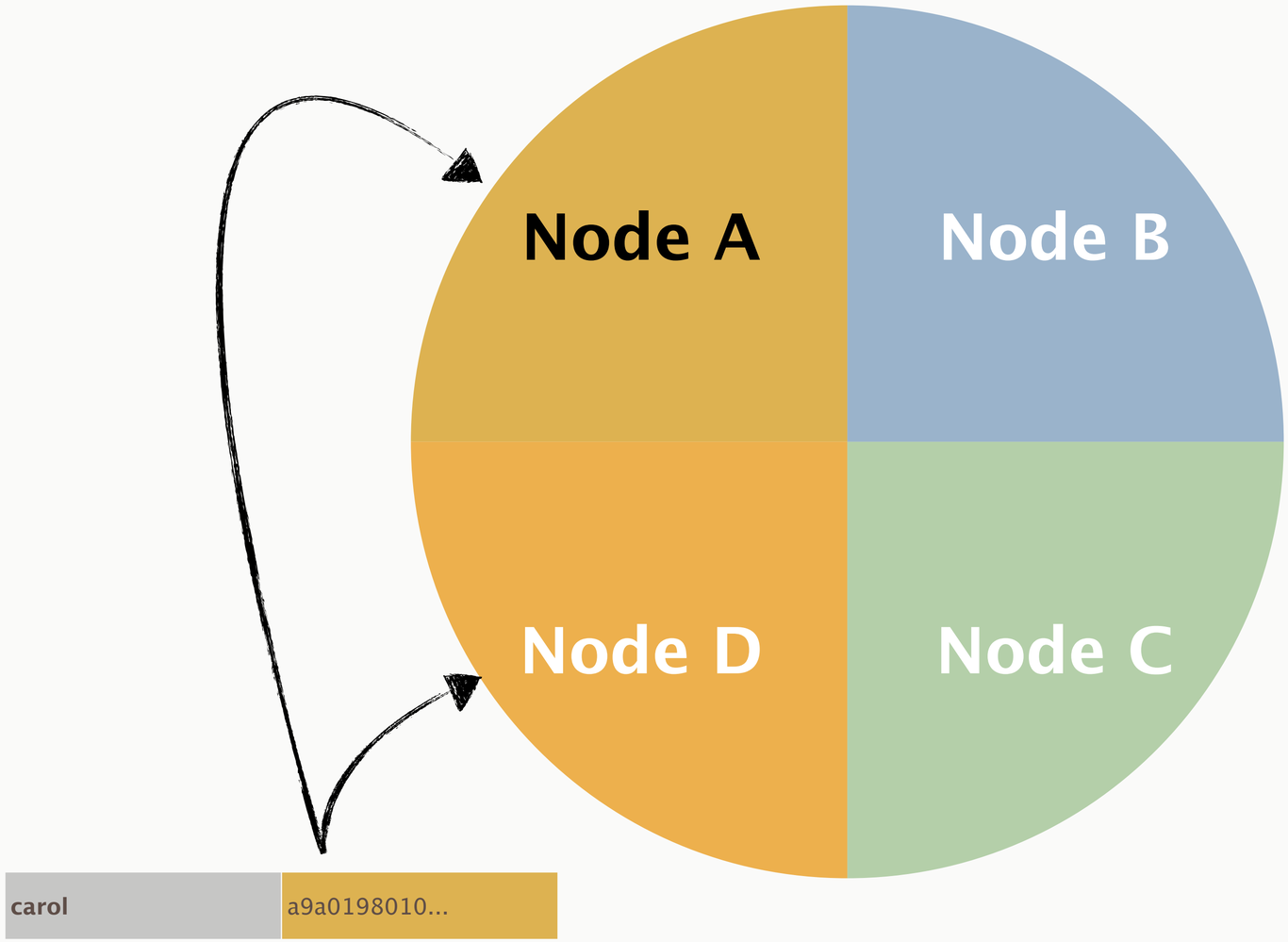
jim	5e02739678...
carol	a9a0198010...
johnny	f4eb27cea7...
suzy	78b421309e...



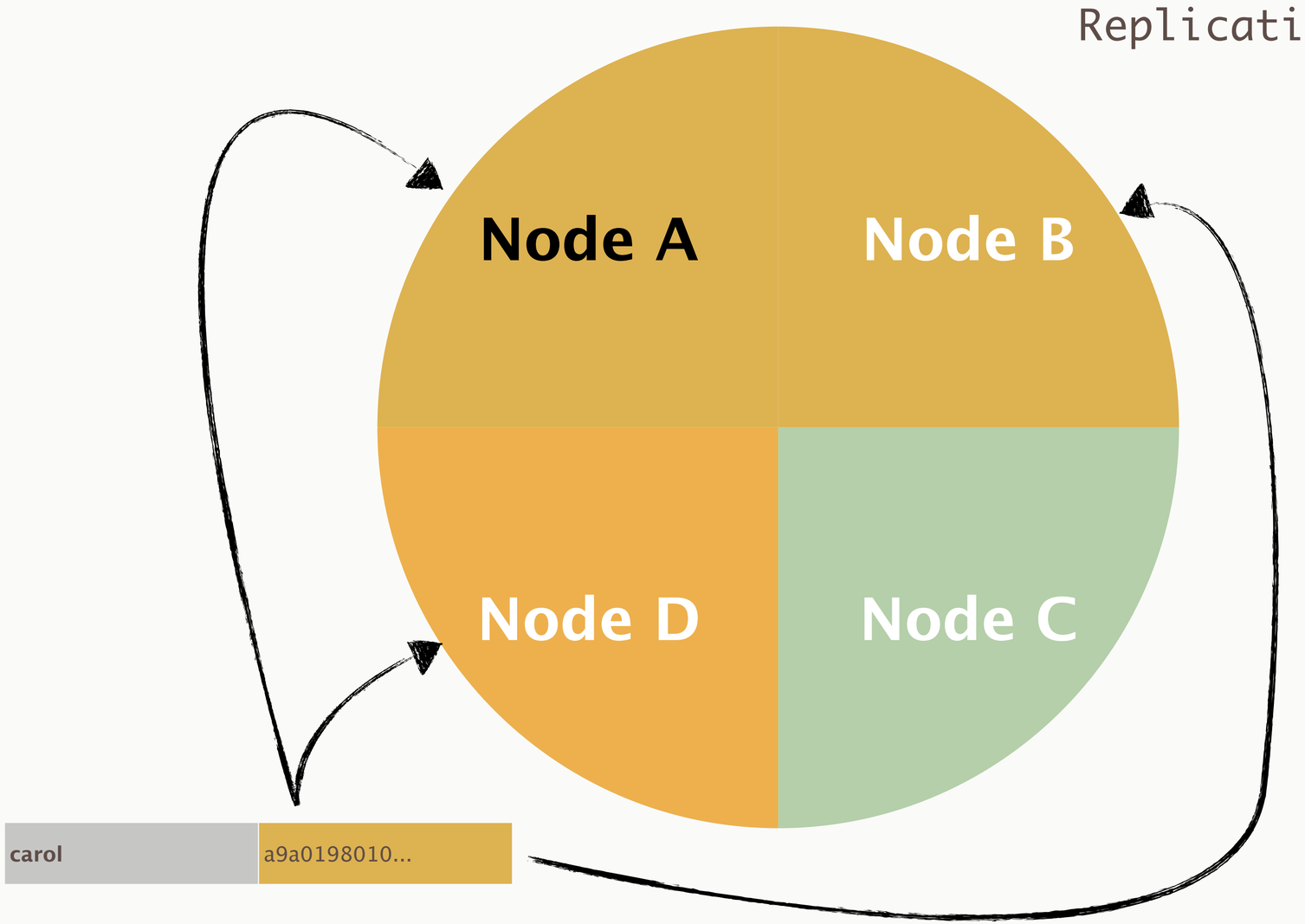
Replication



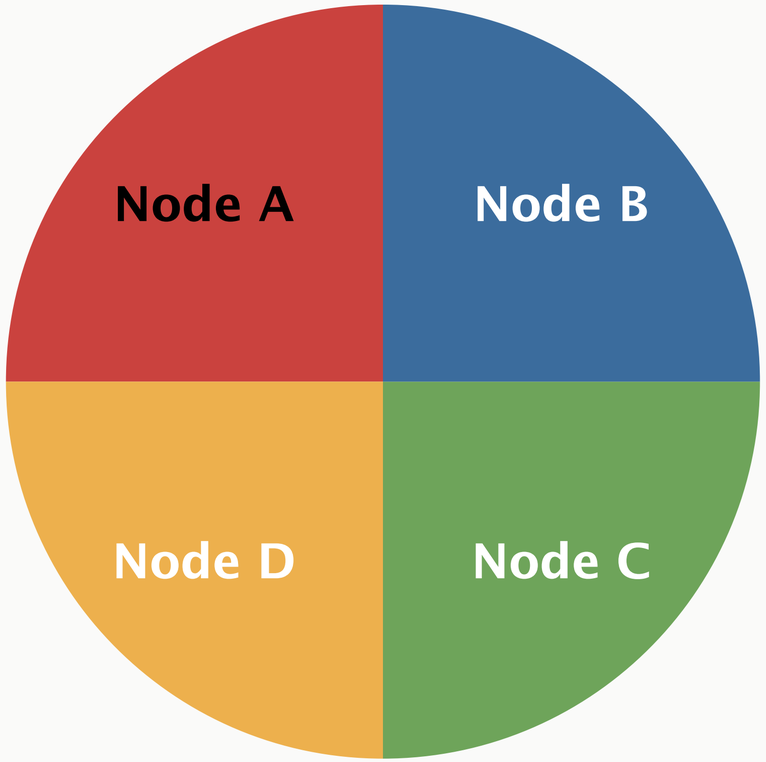
Replication



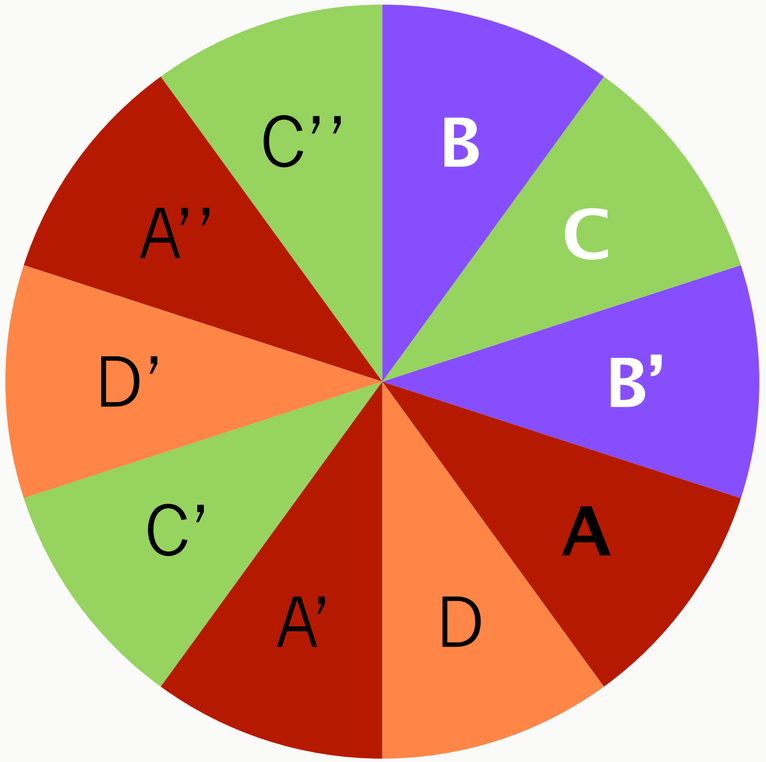
Replication



Virtual Nodes



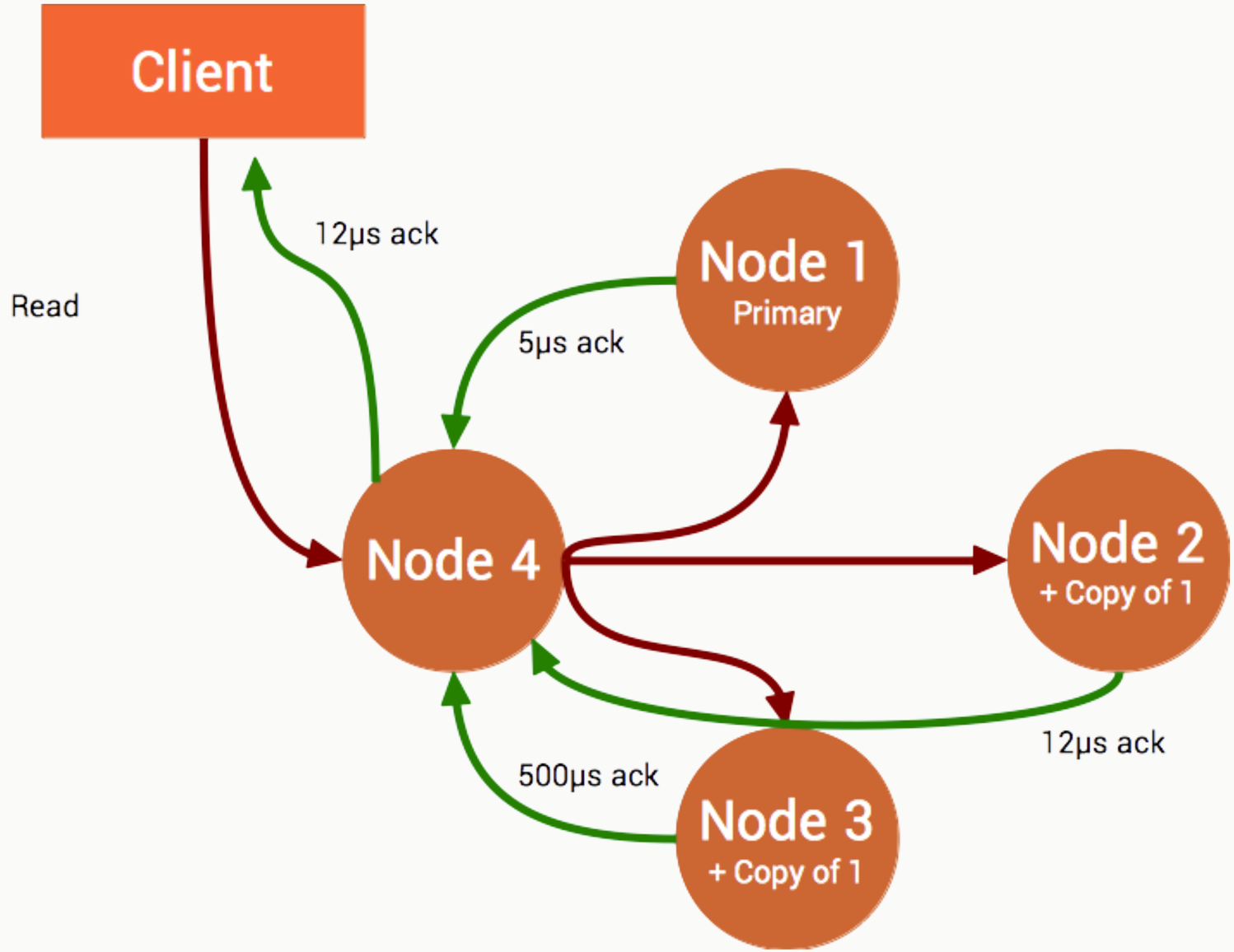
Without vnodes



With vnodes

Cassandra - Reads

Coordinated reads

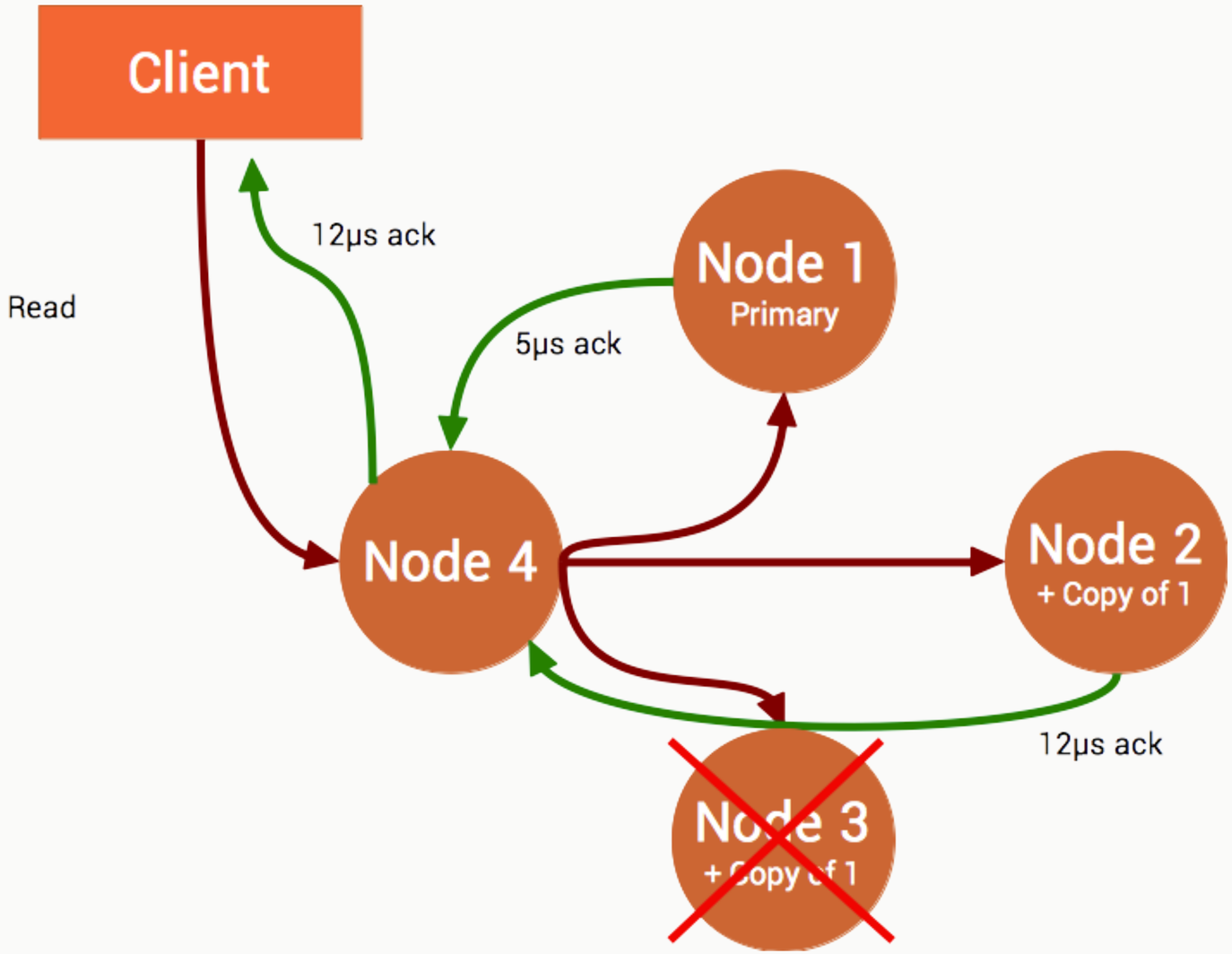


Consistency Level

DATASTAX

- Set with every read and write
- ONE
- QUORUM - >51% replicas ack
- LOCAL_QUORUM - >51% replicas ack in local DC
- LOCAL_ONE - Read repair only in local DC
- TWO
- ALL - All replicas ack. Full consistency

QUORUM and availability



Rapid Read Protection

DATASTAX

Cassandra Query Language - CQL

CQL Tables

DATASTAX

- List of columns
- No sizes?
- First item in PRMARY KEY is the partition key

```
CREATE TABLE users (
   username varchar,
   firstname varchar,
   lastname varchar,
   email list<varchar>,
   password varchar,
   created_date timestamp,
   PRIMARY KEY (username)
);
```

```
INSERT INTO users (username, firstname, lastname,
 email, password, created_date)
VALUES ('pmcfadin','Patrick','McFadin',
 ['patrick@datastax.com'],'ba27e03fd95e507daf2937c937d499ab',
 '2011-06-20 13:50:00');
```

CQL Inserts

DATASTAX

- Insert will always overwrite

```
INSERT INTO users (username, firstname, lastname,
 email, password, created_date)
VALUES ('pmcfadin','Patrick','McFadin',
 ['patrick@datastax.com'],'ba27e03fd95e507daf2937c937d499ab',
 '2011-06-20 13:50:00');
```

Advanced Data Models

DATASTAX

```
CREATE TABLE user_activity (
  username varchar,
  interaction_time timeuuid,
  activity_code varchar,
  detail varchar,
  PRIMARY KEY (username, interaction_time)
) WITH CLUSTERING ORDER BY (interaction_time DESC);
```

Reverse order based on timestamp

```
INSERT INTO user_activity
(username,interaction_time,activity_code,detail)
VALUES ('pmcfadin',0D1454E0-
D202-11E2-8B8B-0800200C9A66,'100','Normal login')
USING TTL 2592000;
```

Expire after 30 days

Real time data access

```
select * from user_activity limit 5;
```

```
 username | interaction_time                     | detail                                        | activity_code
----------+--------------------------------------+-----------------------------------------------+---------------
 pmcfadin | 9ccc9df0-d076-11e2-923e-5d8390e664ec |          Entered shopping area: Jewelry       |           301
 pmcfadin | 9c652990-d076-11e2-923e-5d8390e664ec | Created shopping cart: Anniversary gifts      |           202
 pmcfadin | 1b5cef90-d076-11e2-923e-5d8390e664ec |    Deleted shopping cart: Gadgets I want      |           205
 pmcfadin | 1b0e5a60-d076-11e2-923e-5d8390e664ec |     Opened shopping cart: Gadgets I want      |           201
 pmcfadin | 1b0be960-d076-11e2-923e-5d8390e664ec |                           Normal login        |           100
```

Maybe put a sale item for flowers too?