DataStax

Cassandra

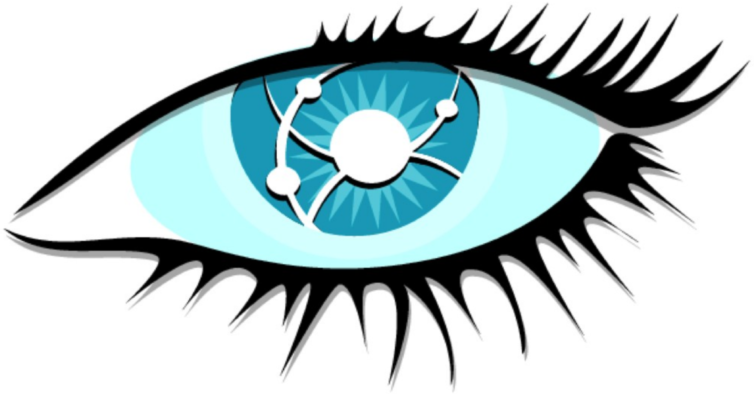
for

Python Developers

Tyler Hobbs

History

• Open-sourced by Facebook	2008
• Apache Incubator	2009
• Top-level Apache project	2010
• DataStax founded	2010



Strengths

- **Scalable**
  - 2x Nodes == 2x Performance
- **Reliable (Available)**
  - Replication that works
  - Multi-DC support
  - No single point of failure

Strengths

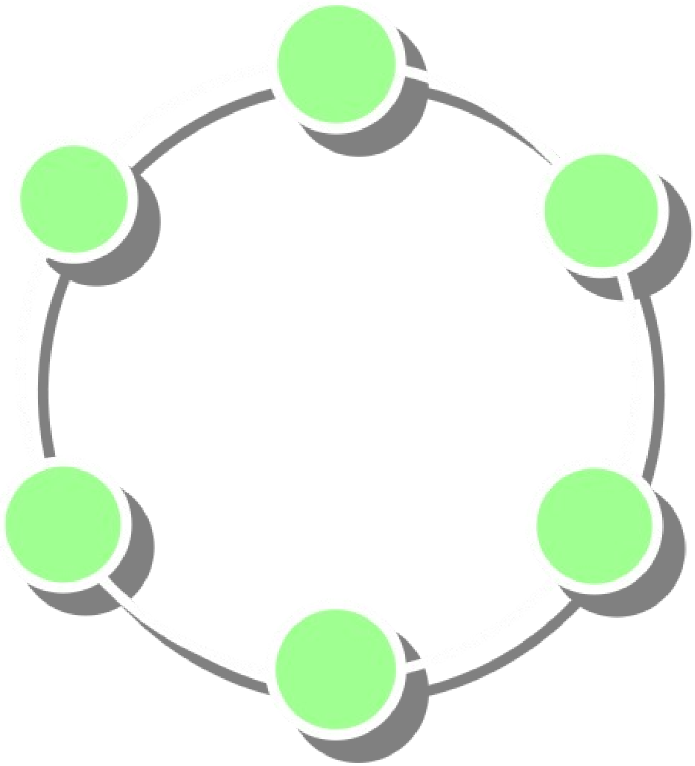
- **Fast**
  - 10-30k writes/sec, 1-10k reads/sec
- **Analytics**
  - Integrated Hadoop support

Weaknesses

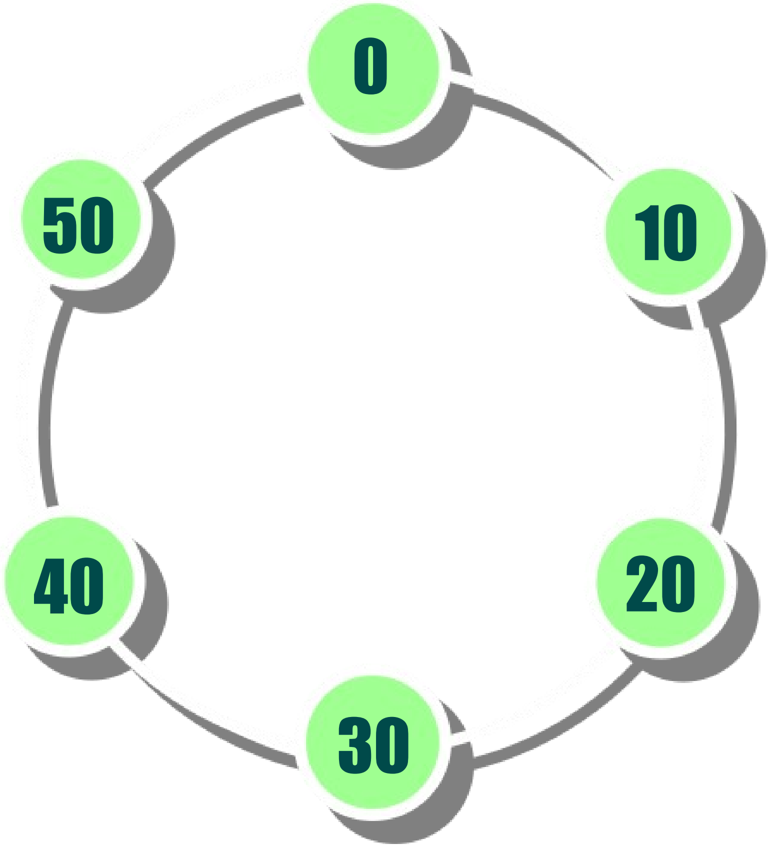
- **No ACID transactions**
  - Don't need these as often as you'd think
- **Limited support for ad-hoc queries**
  - You'll give these up anyway when sharding an RDBMS
- **Generally complements another system**
  - Not intended to be one-size-fits-all

Clustering

- **Every node plays the same role**
  - No masters, slaves, or special nodes

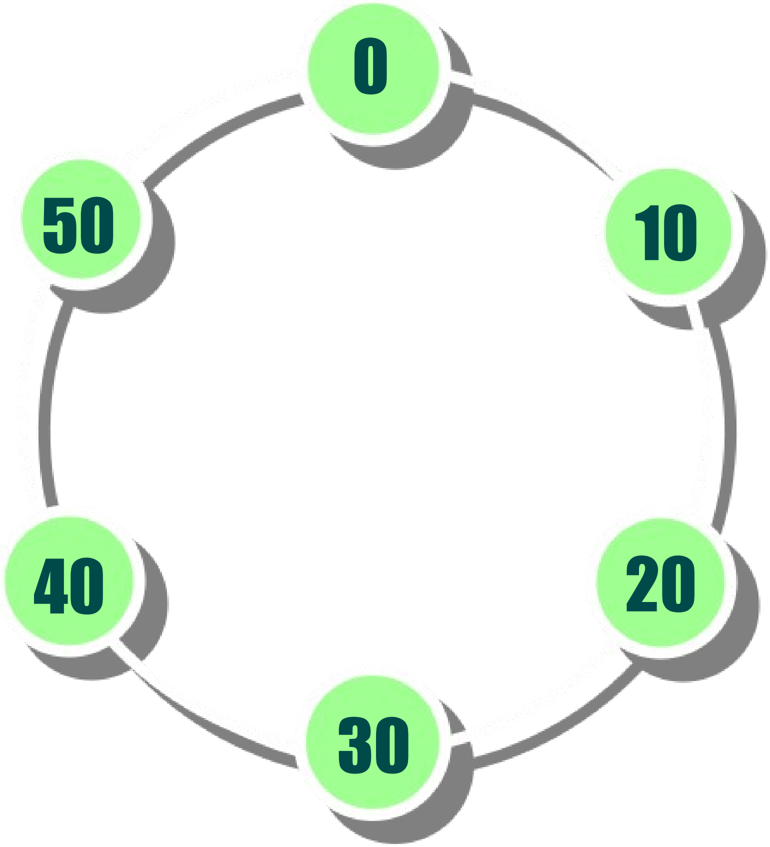


Clustering

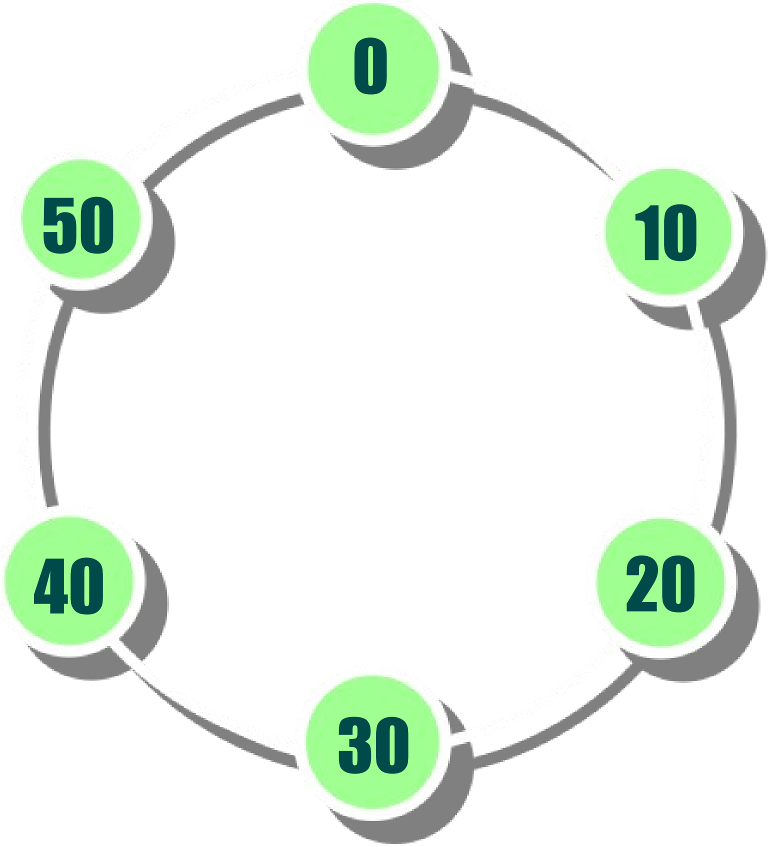


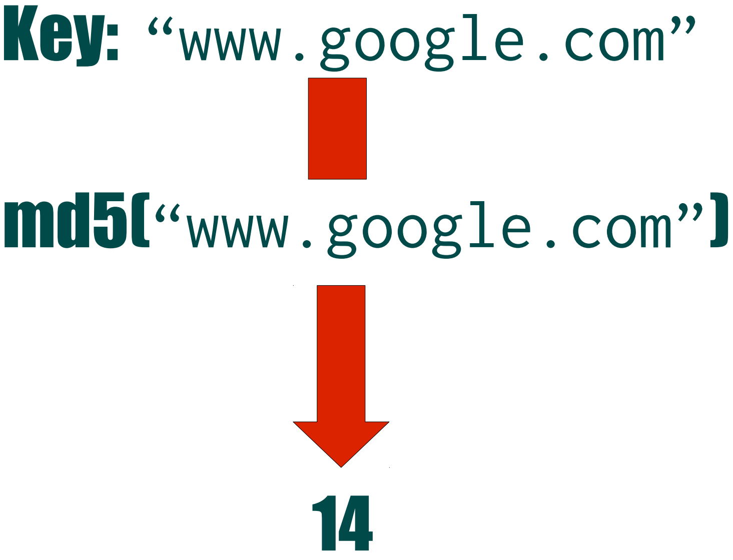
Clustering

Key: “www.google.com”



Clustering

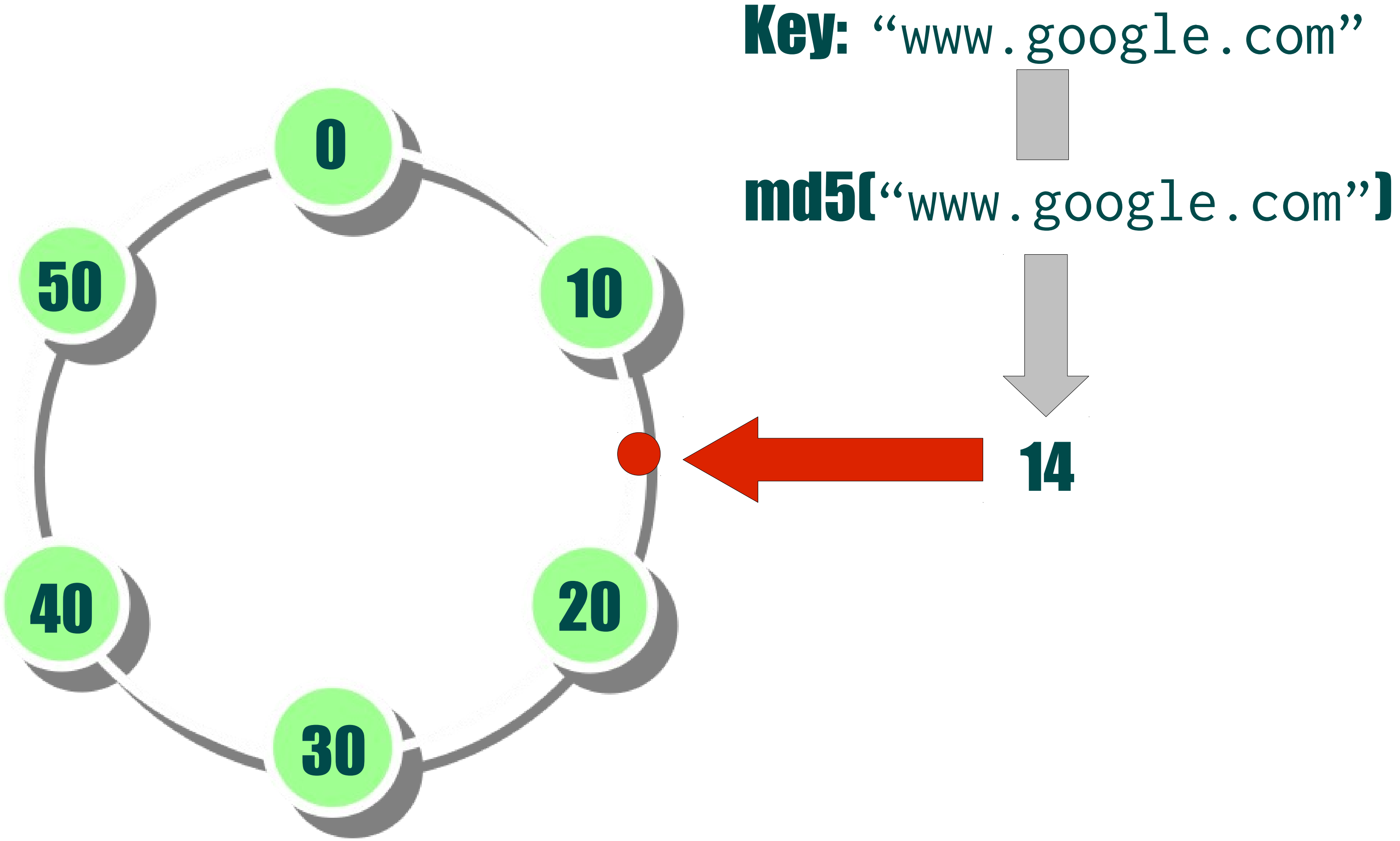



Key: “www.google.com”

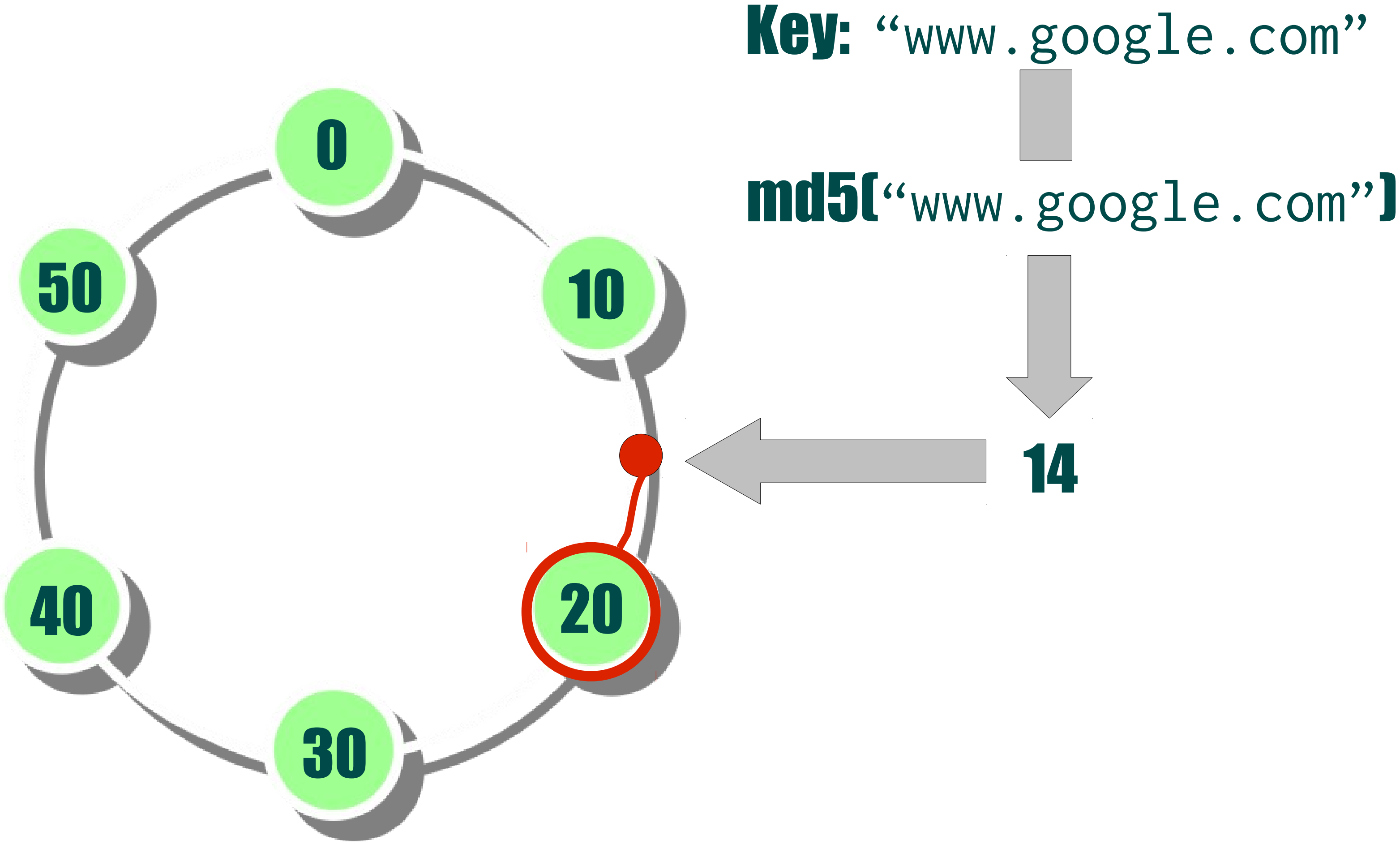
md5(“www.google.com”**)**

14

Clustering



Clustering

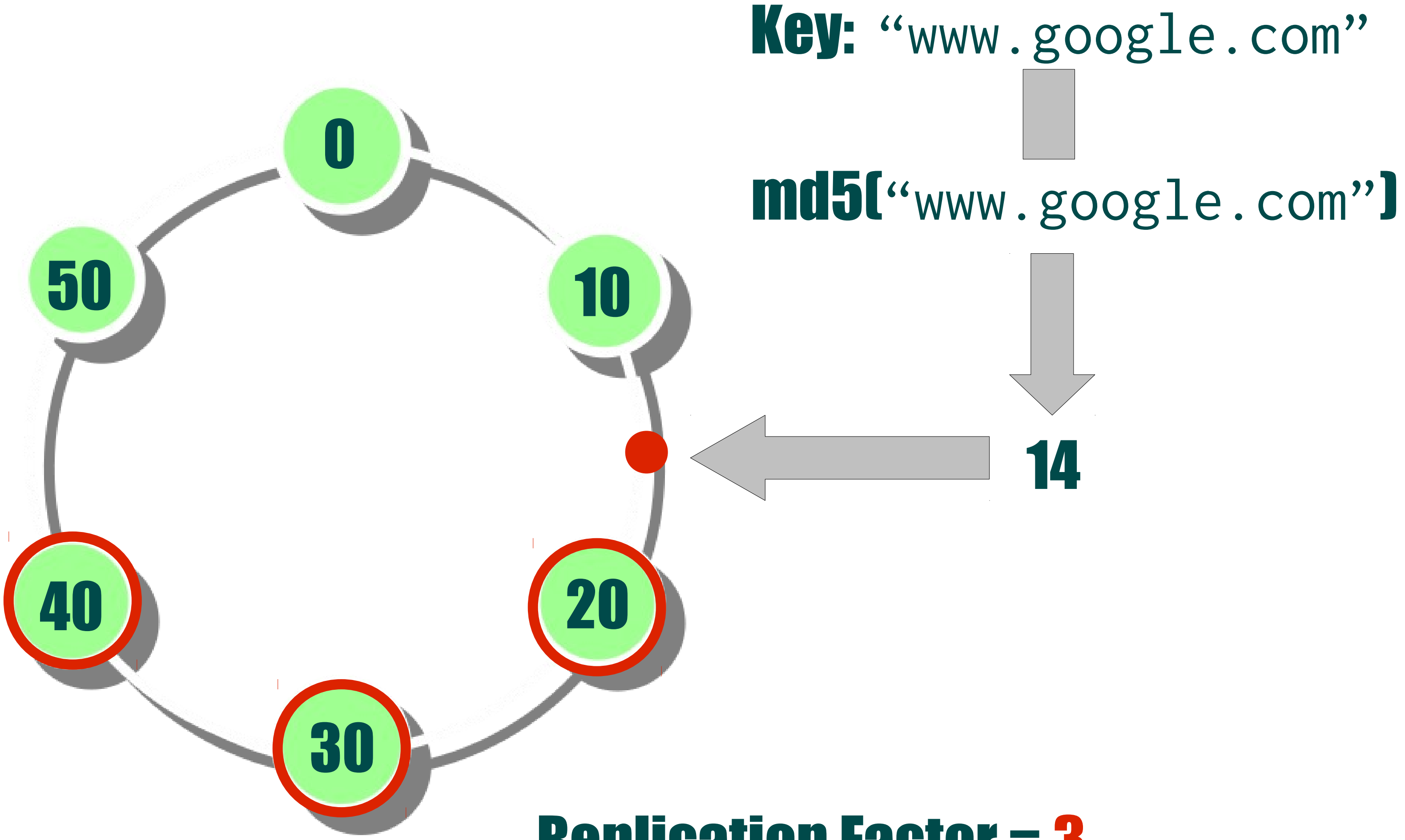


Key: "www.google.com"

md5("www.google.com"**)**

14

Clustering



Key: “www.google.com”

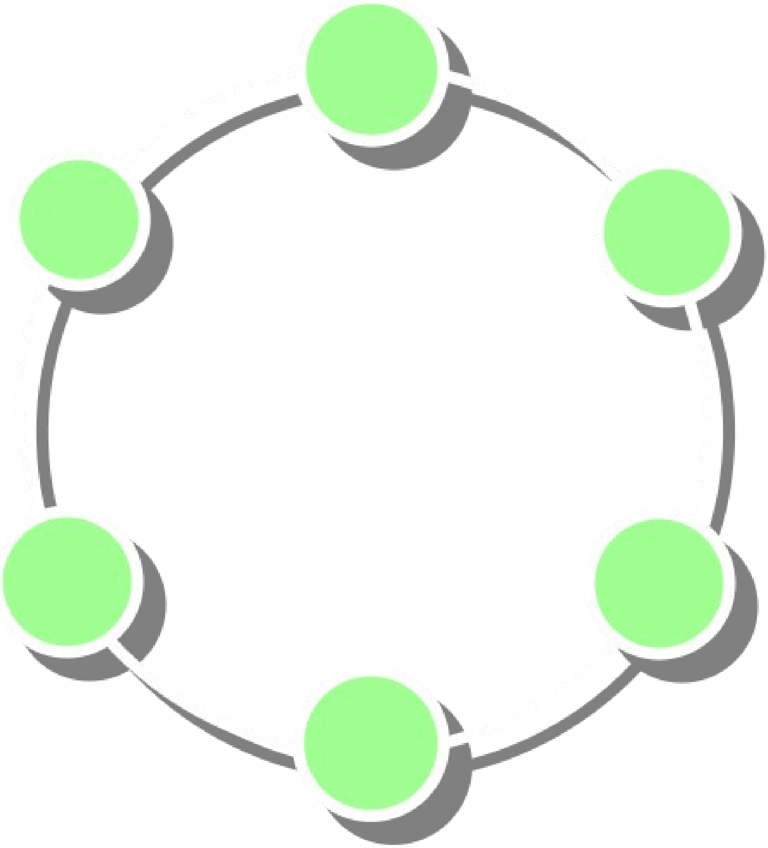
md5(“www.google.com”**)**

14

Replication Factor = 3

Clustering

- **Client can talk to any node**



Data Model

- **Keyspace**
  - A collection of Column Families
  - Controls replication settings
- **Column Family**
  - Kinda resembles a table

ColumnFamilies

- **Static**
  - Object data
- **Dynamic**
  - Pre-calculated query results

Static Column Families

Users

zznate	password: *	name: Nate	
driftx	password: *	name: Brandon	
thobbs	password: *	name: Tyler	
jbellis	password: *	name: Jonathan	site: riptano.com

Dynamic Column Families

Following

zznate	driftx:	thobbs:				
driftx						
thobbs	zznate:					
jbellis	driftx:	mdennis:	pcmanus	thobbs:	xedin:	zznate

Dynamic Column Families

- Timeline of tweets by a user
- Timeline of tweets by all of the people a user is following
- List of comments sorted by score
- List of friends grouped by state

Pycassa

- **Python client library for Cassandra**
- **Open Source (MIT License)**
  - `www.github.com/pycassa/pycassa`
- **Users**
  - Reddit
  - ~10k github downloads of every version

Installing Pycassa

- `easy_install pycassa`
  - or `pip`

Basic Layout

- `pycassa.pool`
  - Connection pooling
- `pycassa.columnfamily`
  - Primary module for the data API
- `pycassa.system_manager`
  - Schema management

The Data API

- **RPC-based API**
- **Rows are like a sorted list of** `(name,value)` **tuples**
  - Like a `dict`, but sorted by the names
  - `OrderedDicts` are used to preserve sorting

Inserting Data

```
>>> from pycassa.pool import ConnectionPool
>>> from pycassa.columnfamily import ColumnFamily
>>>
>>> pool = ConnectionPool(“MyKeyspace”)
>>> cf = ColumnFamily(pool, “MyCF”)
>>>
>>> cf.insert(“key”, {“col_name”: “col_value”})
>>> cf.get(“key”)
{“col_name”: “col_value”}
```

Inserting Data

```
>>> columns = {“aaa”: 1, “ccc”: 3}
>>> cf.insert(“key”, columns)
>>> cf.get(“key”)
{“aaa”: 1, “ccc”: 3}
>>>
>>> # Updates are the same as inserts
>>> cf.insert(“key”, {“aaa”: 42})
>>> cf.get(“key”)
{“aaa”: 42, “ccc”: 3}
>>>
>>> # We can insert anywhere in the row
>>> cf.insert(“key”, {“bbb”: 2, “ddd”: 4})
>>> cf.get(“key”)
{“aaa”: 42, “bbb”: 2, “ccc”: 3, “ddd”: 4}
```

Fetching Data

```
>>> cf.get(“key”)
{“aaa”: 42, “bbb”: 2, “ccc”: 3, “ddd”: 4}
>>>
>>> # Get a set of columns by name
>>> cf.get(“key”, columns=[“bbb”, “ddd”])
{“bbb”: 2, “ddd”: 4}
```

Fetching Data

```
>>> # Get a slice of columns
>>> cf.get(“key”, column_start=”bbb”,
...               column_finish=”ccc”)
{“bbb”: 2, “ccc”: 3}
>>>
>>> # Slice from “ccc” to the end
>>> cf.get(“key”, column_start=”ccc”)
{“ccc”: 3, “ddd”: 4}
>>>
>>> # Slice from “bbb” to the beginning
>>> cf.get(“key”, column_start=”bbb”,
...               column_reversed=True)
{“bbb”: 2, “aaa”: 42}
```

Fetching Data

```
>>> # Get the first two columns in the row
>>> cf.get(“key”, column_count=2)
{“aaa”: 42, “bbb”: 2}
>>>
>>> # Get the last two columns in the row
>>> cf.get(“key”, column_reversed=True,
...               column_count=2)
{“ddd”: 4, “ccc”: 3}
```

Fetching Multiple Rows

```
>>> columns = {“col”: “val”}
>>> cf.batch_insert({“k1”: columns,
...                  “k2”: columns,
...                  “k3”: columns})
>>>
>>> # Get multiple rows by name
>>> cf.multiget([“k1”,“k2”])
{“k1”: {”col”: “val”},
 “k2”: {“col”: “val”}}


>>> # You can get slices of each row, too
>>> cf.multiget([“k1”,“k2”], column_start=”bbb”) …
```

Fetching a Range of Rows

```
>>> # Get a generator over all of the rows
>>> for key, columns in cf.get_range():
...     print key, columns
“k1” {”col”: “val”}
“k2” {“col”: “val”}
“k3” {“col”: “val”}


>>> # You can get slices of each row
>>> cf.get_range(column_start=”bbb”) …
```

Fetching Rows by Secondary Index

```
>>> from pycassa.index import *
>>>
>>> # Build up our index clause to match
>>> exp = create_index_expression(“name”, “Joe”)
>>> clause = create_index_clause([exp])
>>> matches = users.get_indexed_slices(clause)
>>>
>>> # results is a generator over matching rows
>>> for key, columns in matches:
...     print key, columns
“13” {”name”: “Joe”, “nick”: “thatguy2”}
“257” {“name”: “Joe”, “nick”: “flowers”}
“98” {“name”: “Joe”, “nick”: “fr0d0”}
```

Deleting Data

```
>>> # Delete a whole row
>>> cf.remove(“key1”)
>>>
>>> # Or selectively delete columns
>>> cf.remove(“key2”, columns=[“name”, “date”])
```

Connection Management

- `pycassa.pool.ConnectionPool`
  - Takes a list of servers
    - Can be any set of nodes in your cluster
  - `pool_size, max_retries, timeout`
  - Automatically retries operations against other nodes
    - Writes are idempotent!
  - Individual node failures are transparent
  - Thread safe

Async Options

- **eventlet**
  - Just need to monkeypatch `socket` and `threading`
- **Twisted**
  - Use Telephus instead of Pycassa
  - `www.github.com/driftx/telephus`
  - Less friendly, documented, etc

DataStax

Tyler Hobbs
@tylhobbs
tyler@datastax.com