

Hands-on Cassandra

OSCON
July 20, 2010

Eric Evans
eevans@rackspace.com
[@jericevans](#)
<http://blog.sym-link.com>



Background



Influential Papers

- BigTable
 - Strong consistency
 - Sparse map data model
 - GFS, Chubby, et al
- Dynamo
 - $O(1)$ distributed hash table (DHT)
 - BASE (aka eventual consistency)
 - Client tunable consistency/availability



NoSQL

- HBase
- MongoDB
- Riak
- Voldemort
- Neo4J
- Cassandra
- Hypertable
- HyperGraphDB
- Memcached
- Tokyo Cabinet
- Redis
- CouchDB



~~NoSQL~~ Big data

- HBase
- ~~MongoDB~~
- Riak
- Voldemort
- ~~Neo4J~~
- Cassandra
- Hypertable
- ~~HyperGraphDB~~
- ~~Memcached~~
- ~~Tokyo Cabinet~~
- ~~Redis~~
- ~~CouchDB~~



Bigtable / Dynamo

Bigtable

- HBase
- Hypertable

Dynamo

- Riak
- Voldemort

Cassandra ??



Dynamo-Bigtable Lovechild



CAP Theorem “Pick Two”

- **CP**

- Bigtable
- Hypertable
- HBase

- **AP**

- Dynamo
- Voldemort
- Cassandra



CAP Theorem ~~“Pick Two”~~



Consistency

- Availability
- Partition Tolerance



History

- Facebook (2007-2008)
 - Avinash, former Dynamo engineer
 - Motivated by “Inbox Search”
- Google Code (2008-2009)
 - Dark times
- Apache (2009-Present)
 - Digg, Twitter, Rackspace, Others
 - Rapidly growing community
 - Fast-paced development



Hands-on Setup



“Installation”

```
$ TUT_ROOT=$HOME
```

```
$ cd $TUT_ROOT
```

```
$ tar xfz apache-cassandra-xxxx-bin.tar.gz
```

```
$ tar xfz twissandra-xxxx.tar.gz
```

```
$ tar xfz pycassa-xxxx.tar.gz
```

Setup

```
$ cp twissandra/cassandra.yaml \
    apache-cassandra-xxxx/conf
```

```
$ mkdir $TUT_ROOT/log
```

```
$ mkdir -p $TUT_ROOT/lib/data
```

```
$ mkdir -p $TUT_ROOT/lib/commitlog
```

Setup (continued)

conf/cassandra.yaml

...

Where data is stored on disk

`data_file_directories:`

- TUT_ROOT/lib/data

...

Commit log

`commitlog_directory:` TUT_ROOT/lib/commitlog

...

Setup (continued)

conf/log4j-server.properties

...

```
log4j.rootLogger=DEBUG, stdout, R
```

...

```
log4j.appender.R.File=TUT_ROOT/log/system.log
```

...

Starting up / Initializing

```
$ cd $TUT_ROOT/apache-cassandra-xxxx  
$ bin/cassandra -f
```

In a new terminal

```
$ cd $TUT_ROOT/apache-cassandra-xxxx  
$ bin/loadSchemaFromYAML localhost 8080
```

Pycassa / Twissandra

```
$ cd $TUT_ROOT/pycassa
$ sudo python setup.py -cassandra install \
    [--prefix=/usr/local]
...
$ cd $TUT_ROOT/twissandra
$ python manage.py runserver 0.0.0.0:8000
```

Data Model



Users

```
CREATE TABLE user (  
    id INTEGER PRIMARY KEY,  
    username VARCHAR(64),  
    password VARCHAR(64)  
);
```



Friends and Followers

```
CREATE TABLE followers (  
    user INTEGER REFERENCES user(id),  
    follower INTEGER REFERENCES user(id)  
);
```

```
CREATE TABLE following (  
    user INTEGER REFERENCES user(id),  
    followee INTEGER REFERENCES user(id)  
);
```



Tweets

```
CREATE TABLE tweets (  
    id INTEGER PRIMARY KEY,  
    user INTEGER REFERENCES user(id),  
    body VARCHAR(140),  
    timestamp TIMESTAMP  
);
```



Overview

- Keyspace
 - Uppermost namespace
 - Typically one per application
- ColumnFamily
 - Associates records of a similar kind
 - Record-level Atomicity
 - Indexed
- Column
 - Basic unit of storage



Sparse Table

Key 1	A=1	C=5	D=20	E=9
Key 2	B=19	D=22	E=7	
Key 3	A=3	B=4	F=42	H=7



Column

- name
 - byte[]
 - Queried against (predicates)
 - Determines sort order
- value
 - byte[]
 - Opaque to Cassandra
- timestamp
 - long
 - Conflict resolution (Last Write Wins)



Column Comparators

- Bytes
- UTF8
- TimeUUID
- Long
- LexicalUUID
- Composite (third-party)

<http://github.com/edanuff/CassandraCompositeType>



Column Families

- User
- Username
- Friends
- Followers
- Tweet
- Timeline
- Userline



User / Username

- User
 - Stores users
 - Keyed on a unique ID (UUID).
 - Columns for username and password
- Username
 - Indexes User
 - Keyed on username
 - One column, the unique UUID for user



Friends and Followers

- Friends
 - Maps a user to the users they follow
 - Keyed on user ID
 - Columns for each user being followed
- Followers
 - Maps a user to those following them
 - Keyed on username
 - Columns for each user following



Tweets

- Keyed on a unique identifier
- Columns:
 - Unique identifier
 - User ID
 - Body of the tweet
 - timestamp



Timeline / Userline

- Timeline
 - Keyed on user ID
 - Columns that map timestamps to Tweet ID
 - The materialized view of Tweets for a user.
- Userline
 - Keyed on user ID
 - Columns that map timestamps to Tweet ID
 - The collection of Tweets attributed to a user



Pycassa



Pycassa - Python Client API

- `connect()` → Thrift proxy
- `cf = ColumnFamily(proxy, ksp, cfname)`
- `cf.insert()` → long
- `cf.get()` → dict
- `cf.get_range()` → dict

<http://github.com/vomjom/pycassa>



Adding a User

`cass.save_user()`

```
username = 'jericcevans'  
password = '*****'  
useruuid = str(uuid())
```

```
columns = {  
    'id': useruuid,  
    'username': username,  
    'password': password  
}
```

```
USER.insert(useruuid, columns)  
USERNAME.insert(username, {'id': useruuid})
```



Following a Friend

`cass.add_friends()`

```
FRIENDS.insert(userid, {friendid: time()})  
FOLLOWERS.insert(friendid, {userid: time()})
```



Tweeting

`cass.save_tweet()`

```
columns = {
    'id': tweetid,
    'user_id': useruuid,
    'body': body,
    '_ts': timestamp
}
TWEET.insert(tweetid, columns)

columns = {pack('>d', timestamp): tweetid}
USERLINE.insert(useruuid, columns)

TIMELINE.insert(useruuid, columns)
for otheruuid in FOLLOWERS.get(useruuid, 5000):
    TIMELINE.insert(otheruuid, columns)
```



Getting a Timeline

`cass.get_timeline()`

```
start = request.GET.get('start')
limit = NUM_PER_PAGE

timeline = TIMELINE.get(
    userid,
    column_start=start,
    column_count=limit,
    column_reversed=True
)
tweets = TWEET.multiget(timeline.values())
```



Hands-on

pycassaShell



Retweet



Adding Retweet

```
$ cd $TUT_ROOT/twissandra  
$ patch -p1 < ../django.patch  
$ patch -p1 < ../retweet.patch
```

Retweet

`cass.save_retweet()`

```
ts = _long(int(time() * 1e6))  
  
for follower in get_follower_ids(userid):  
    TIMELINE.insert(follower_id, {ts: tweet_id})
```

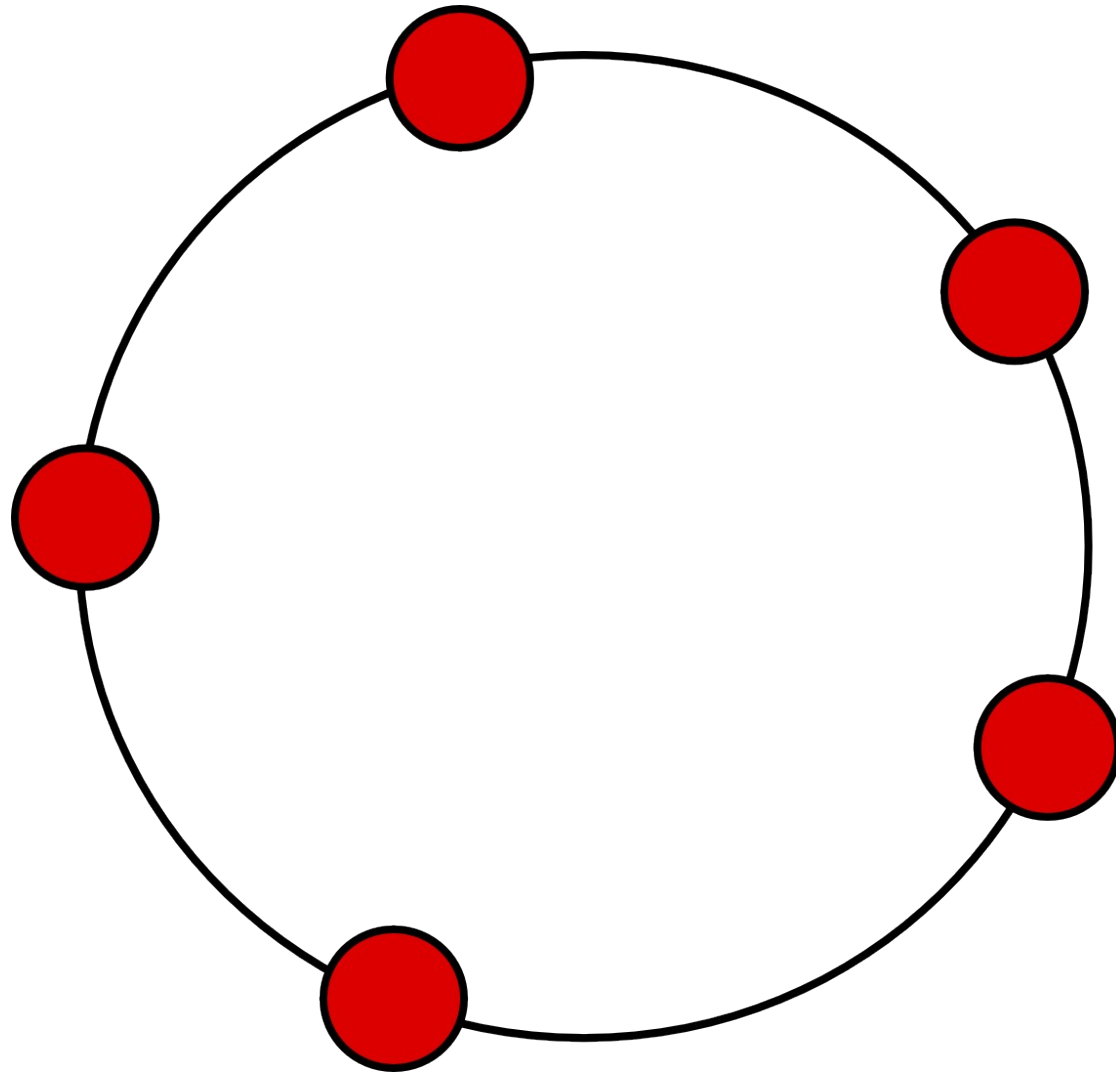


Clustering

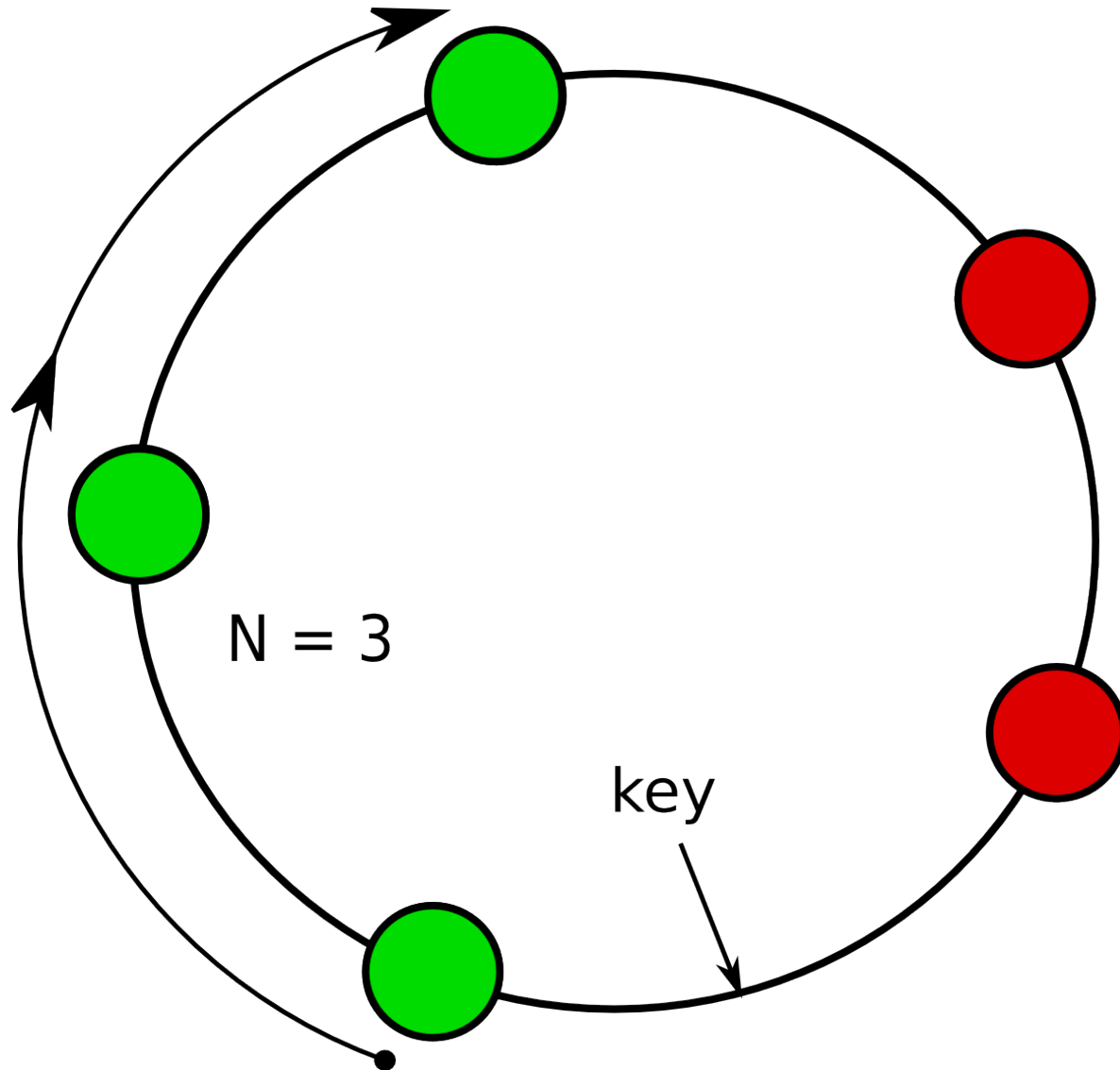
Concepts



P2P Routing



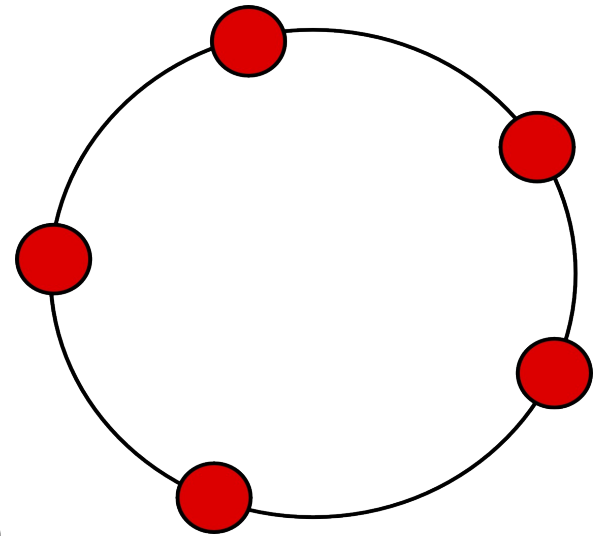
P2P Routing



Partitioning

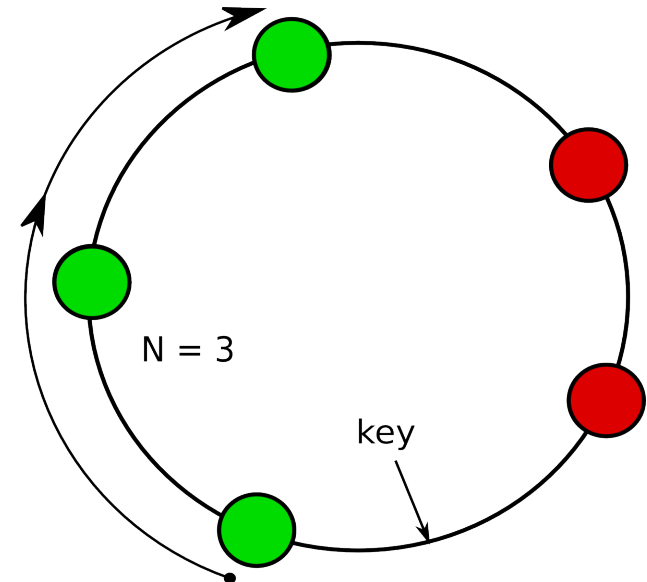
(see partitioner)

- Random
 - 128bit namespace, (MD5)
 - Good distribution
- Order Preserving
 - Tokens determine namespace
 - Natural order (lexicographical)
 - Range / cover queries
- Yours ??



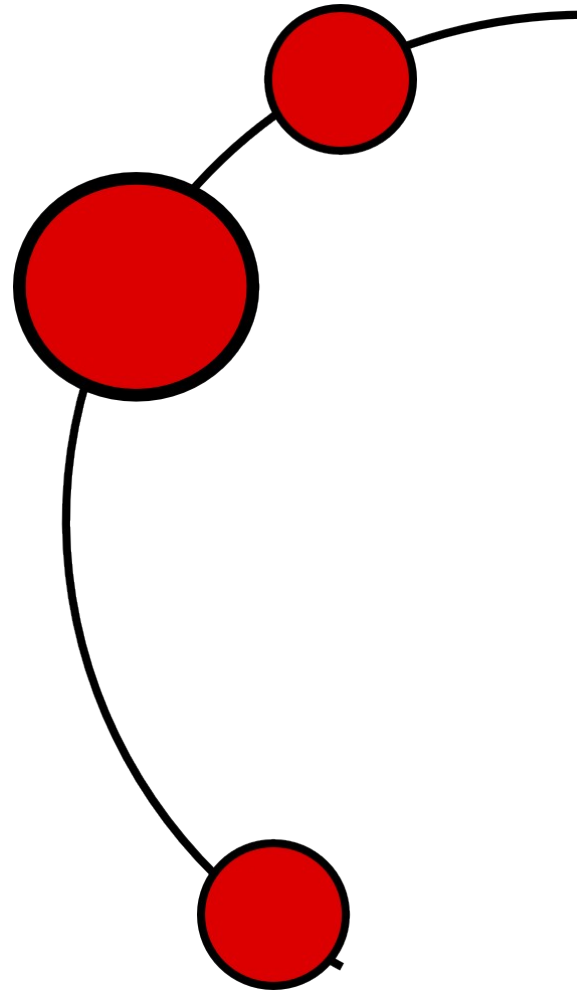
Replica Placement (see endpoint_snitch)

- SimpleSnitch
 - Default
 - N-1 successive nodes
- RackInferringSnitch
 - Infers DC/rack from IP
- PropertyFileSnitch
 - Configured w/ a properties file

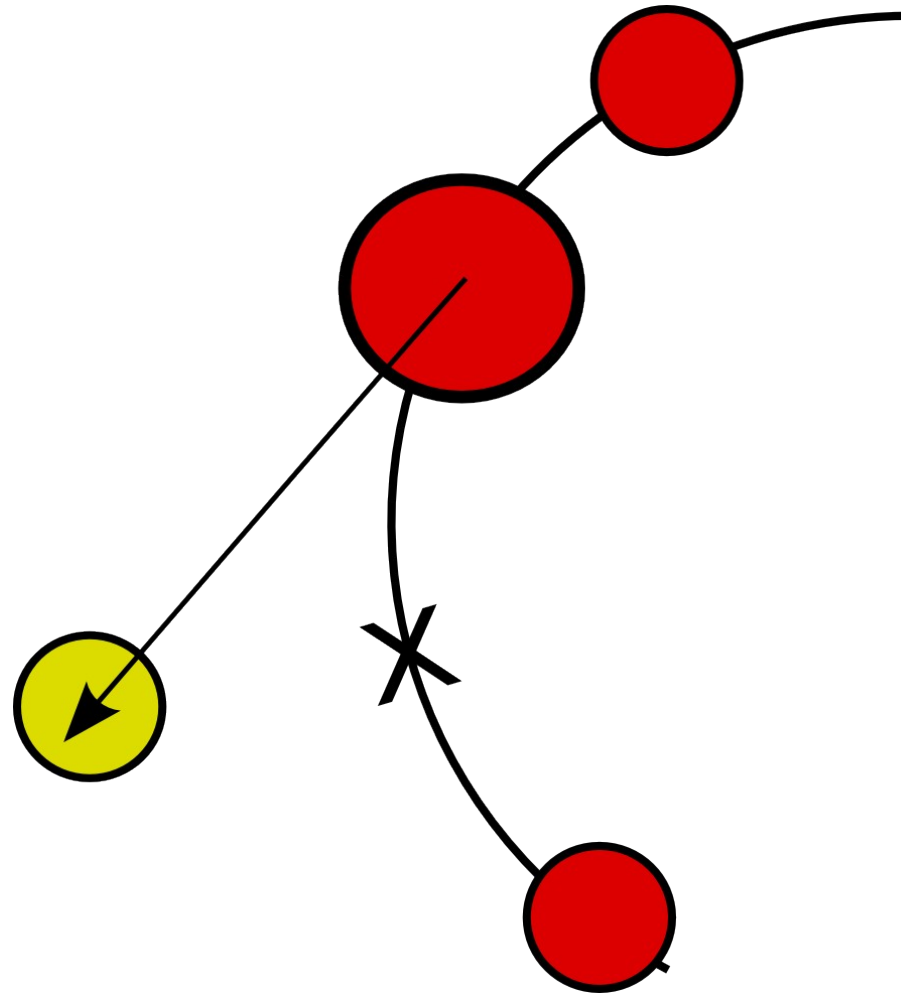


Bootstrap

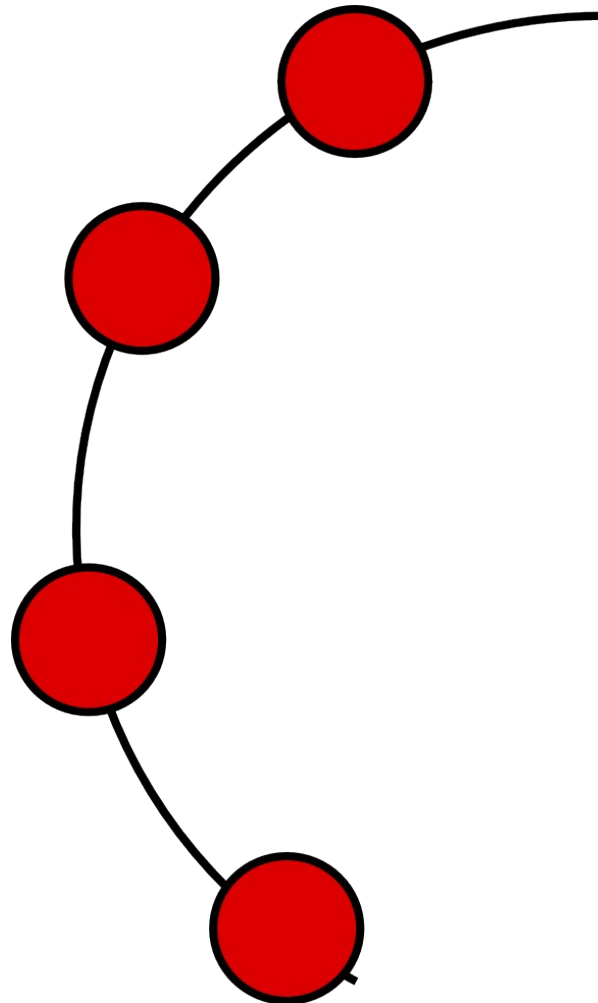
(see auto_bootstrap)



Bootstrap



Bootstrap



Remember CAP?



Consistency

- Availability
- Partition Tolerance



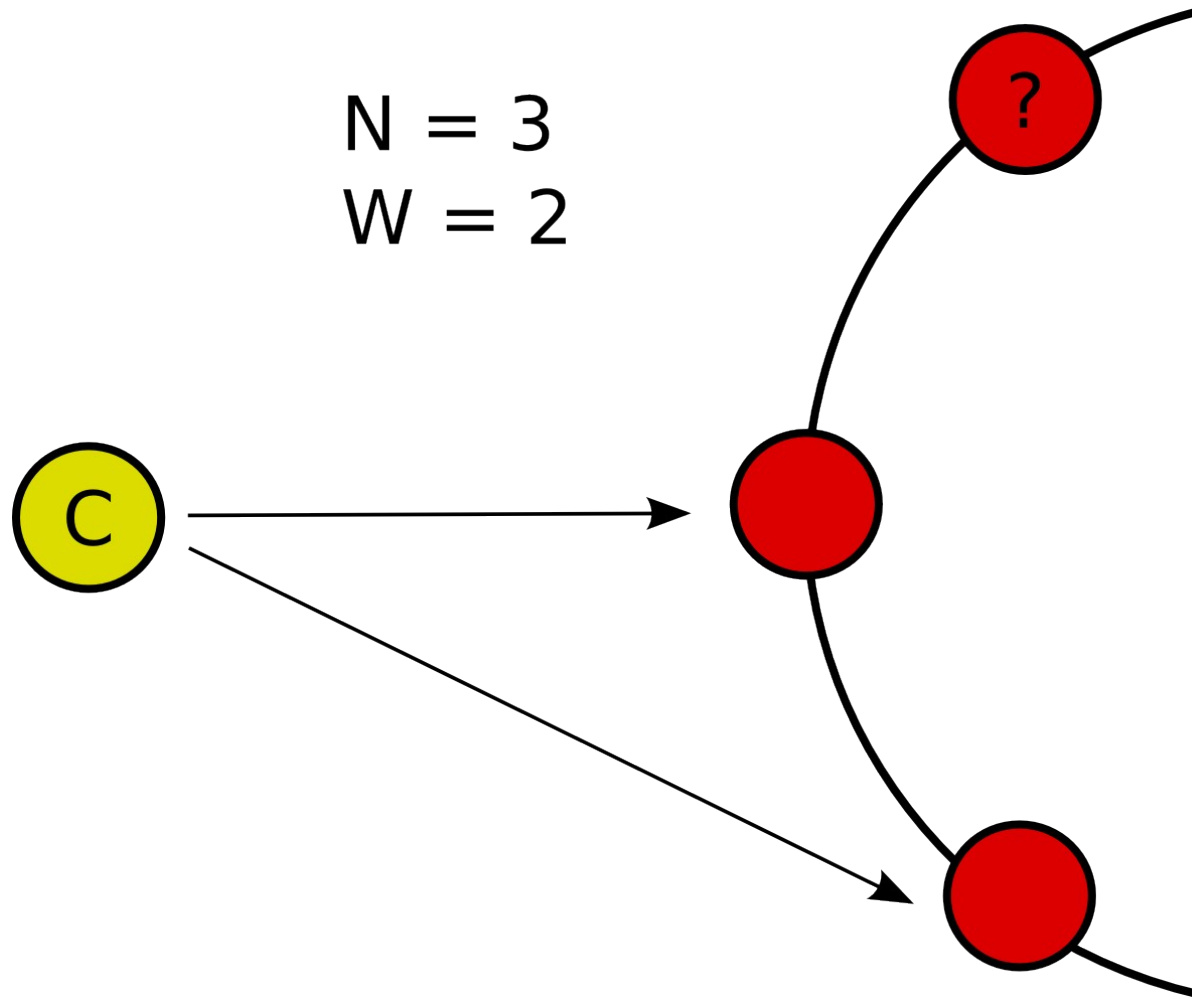
Choosing Consistency

Write		Read	
Level	Description	Level	Description
ZERO	Hail Mary	ZERO	N/A
ANY	1 replica (HH)	ANY	N/A
ONE	1 replica	ONE	1 replica
QUORUM	$(N / 2) + 1$	QUORUM	$(N / 2) + 1$
ALL	All replicas	ALL	All replicas

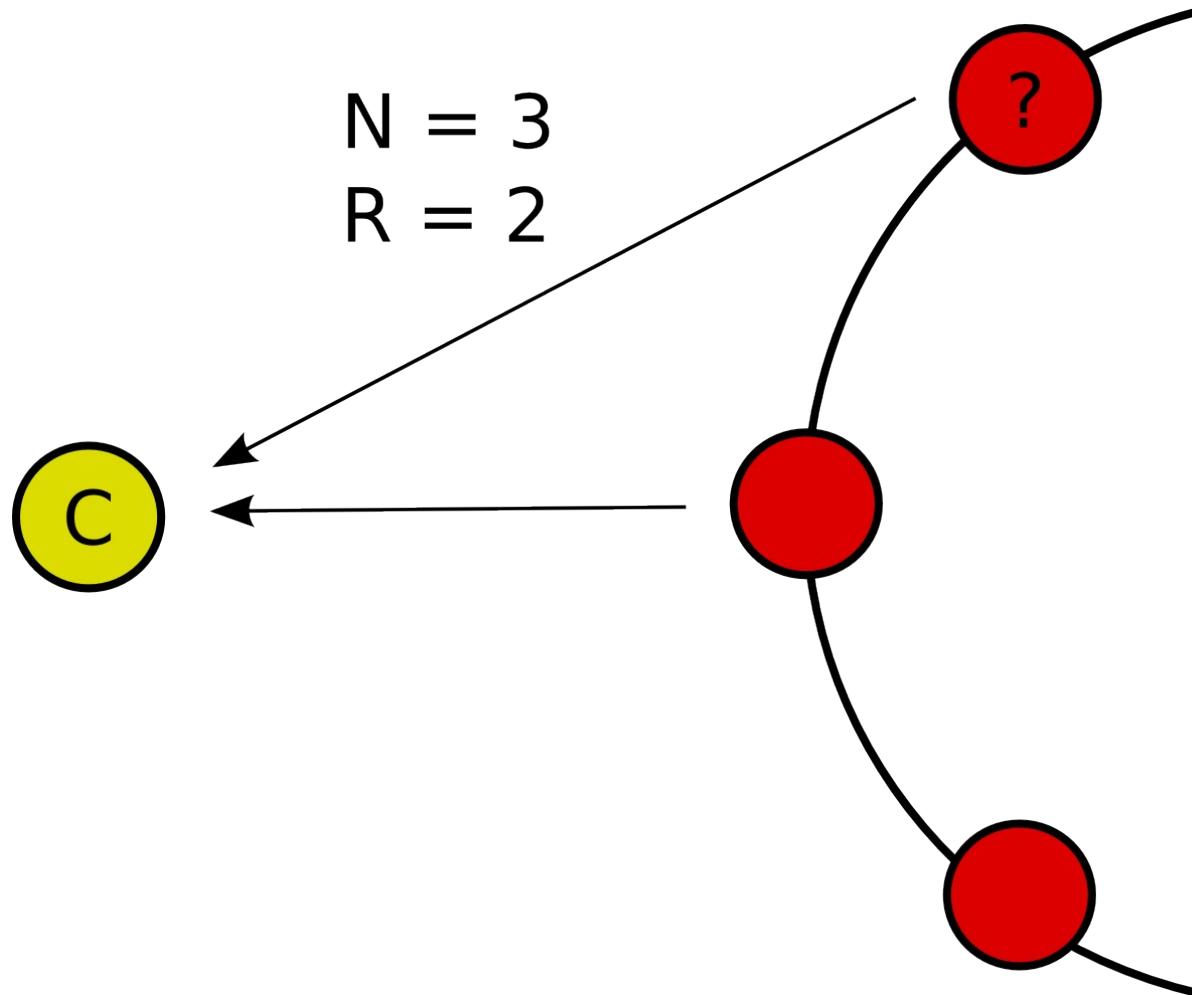
$$R + W > N$$



Quorum $((N/2) + 1)$



Quorum $((N/2) + 1)$



Operations



Cluster sizing

- Data size and throughput
- Fault tolerance (replication)
- Data-center / hosting costs

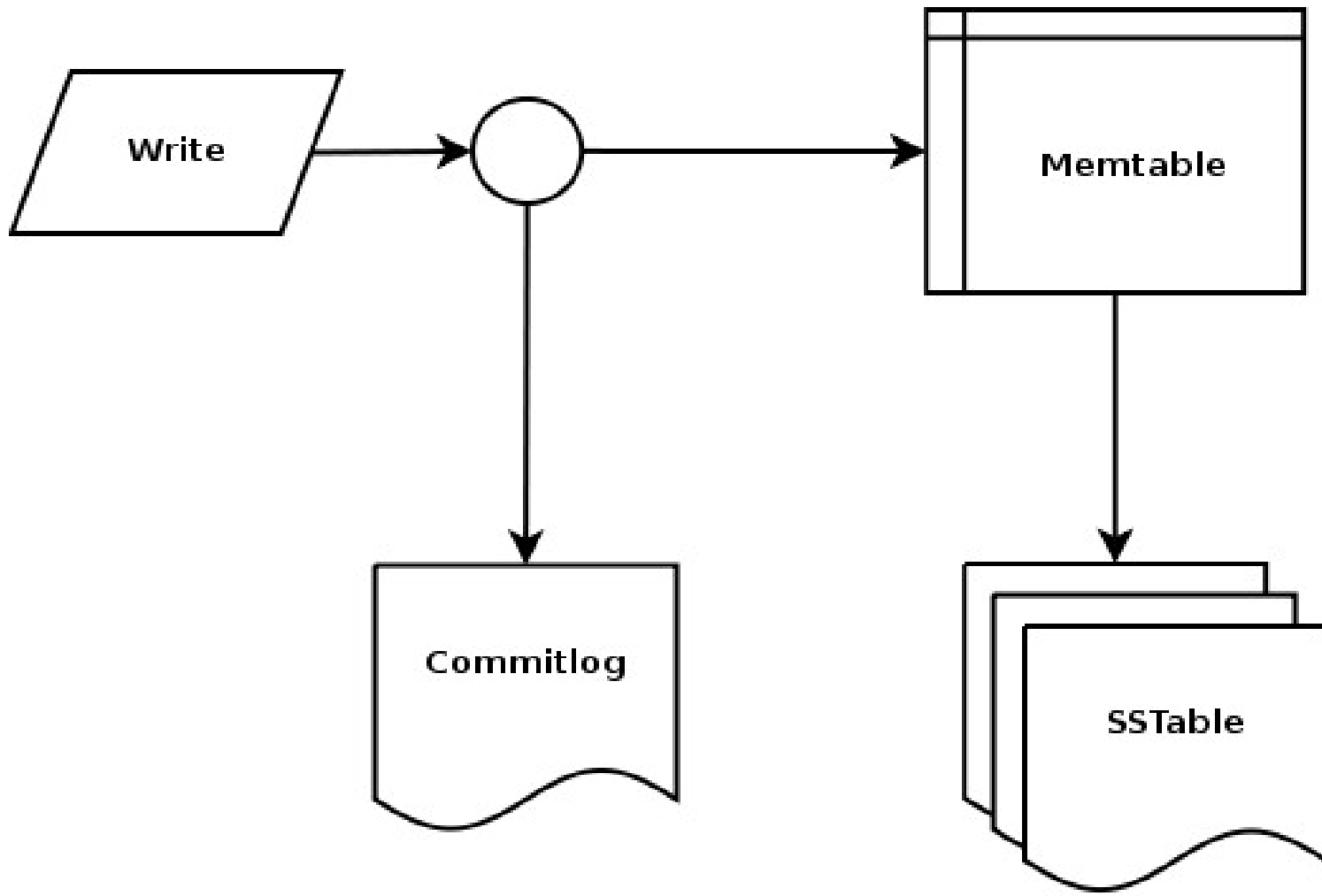


Nodes

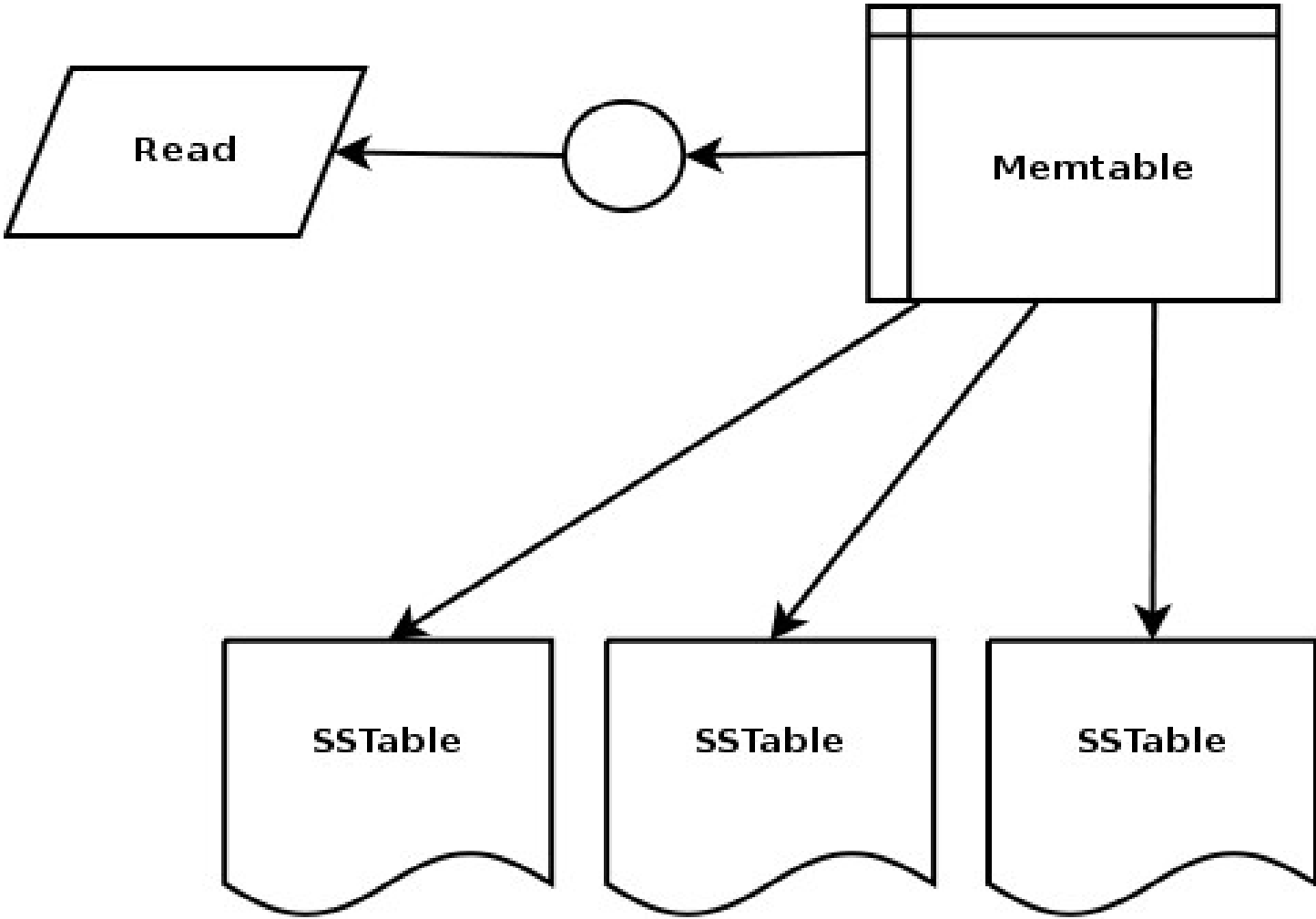
- Go commodity!
- Cores (more is better)
- Memory (more is better)
- Disks
 - Commitlog
 - Storage
 - double-up for working space



Writes



Reads



Tuning (heap size)

bin/cassandra.in.sh

```
# Arguments to pass to the JVM
JVM_OPTS="" \
...
    -Xmx1G \
...
```



Tuning (memtable) conf/cassandra.yaml

Amount of data written

memtable_throughput_in_mb: 64

Number of objects written

memtable_operations_in_millions: 0.3

Time elapsed

memtable_flush_after_mins: 60



Tuning (column families) conf/cassandra.yaml

```
keyspaces :  
  - name : Twissandra  
...  
  column_families :  
    - name : User  
      keys_cached : 100  
      preload_row_cache : true  
      rows_cached : 1000  
...
```



Tuning (mmap)

conf/cassandra.yaml

```
# Choices are auto, standard, mmap, and  
# mmap_index_only.  
disk_access_mode: auto
```



Nodetool

`bin/nodetool -host <arg> command`

- ring
- info
- cfstats
- tpstats



Nodetool (cont.)

`bin/nodetool -host <arg> command`

- compact
- snapshot [name]
- flush
- drain
- repair
- decommission
- move
- loadbalance



Clustertool

`bin/clustertool -host <arg> command`

- `get_endpoints <keyspace> <key>`
- `global_snapshot [name]`
- `clear_global_snapshot`
- `truncate <keyspace> <cfname>`



Wrapping Up



When Things Go Wrong

Where to look

- Logs
 - ERRORS, stack traces
 - Enable DEBUG
 - Isolate if possible
- Crash files (java_pid*.hprof, hs_err_pid*.log)
- nodetool / jconsole / etc
 - Thread pool stats
 - Column family stats
 - ...



When Things Go Wrong

What to do

- user@cassandra.apache.org
 - user-subscribe@cassandra.apache.org
 - <http://www.mail-archive.com/user@cassandra.apache.org/>
- <http://wiki.apache.org/cassandra>
- <https://issues.apache.org/jira/browse/CASSANDRA>
- #cassandra on irc.freenode.net



Further Reading

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, and Robert E. Gruber

<http://labs.google.com/papers/bigtable-osdi06.pdf>

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

http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html



Thanks

- Apache Cassandra:
 - <http://cassandra.apache.org>
- Twissandra: Eric Florenzano (and others)
 - <http://github.com/ericflo/twissandra>
- Pycassa: Jonathan Hseu (and others)
 - <http://github.com/vomjom/pycassa>



Fin

