# JUDCon

JBoss Users & Developers Conference

2012:India

Cassandra

# Cassandra Says: Let there be Data – Available, and in Abundance!

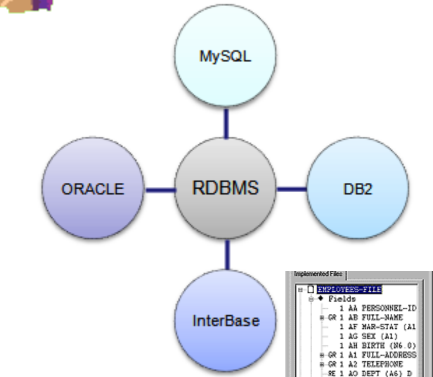
# Who exactly is Cassandra!

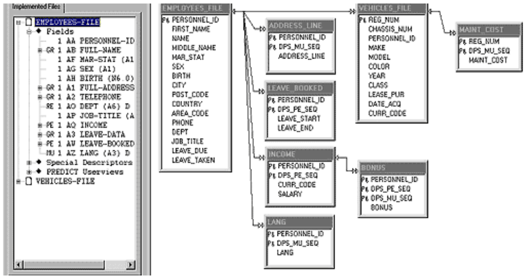
Cassandra kneeling before the Palladion gold ring, c. 400-380 BCE courtesy of University of Haifa Library.

- Cassandra was doomed to tell the truth, but never to be believed.

- “Oracle” connection!??

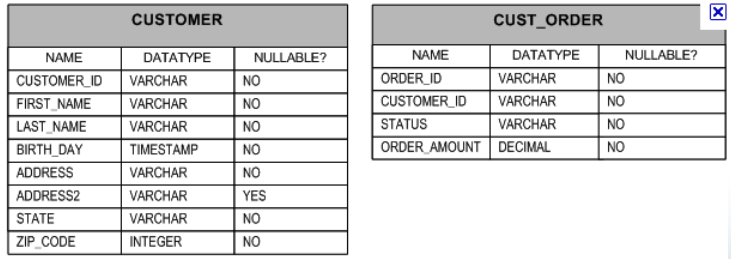
# Do we “REALLY” need her ?





```
⊟ EMPLOYEES-FILE
  ⊟ ◆ Fields
        1 AA PERSONNEL-ID
    ⊞ GR 1 AB FULL-NAME
        1 AF MAR-STAT (A1
        1 AG SEX (A1)
        1 AH BIRTH (N6.0)
    ⊞ GR 1 A1 FULL-ADDRESS
    ⊞ GR 1 A2 TELEPHONE
      RE 1 AO DEPT (A6) D
        1 AP JOB-TITLE (A
    ⊞ PE 1 AQ INCOME
    ⊞ GR 1 A3 LEAVE-DATA
    ⊞ PE 1 AV LEAVE-BOOKED
      MU 1 AZ LANG (A3) D
  ⊞ ◆ Special Descriptors
  ⊞ ◆ PREDICT Userviews
⊞ VEHICLES-FILE
```

- RDBMS …So strong
- so crisp
- so vast
- And WE know it well!

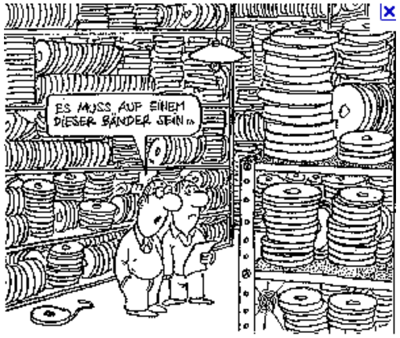


| CUSTOMER | | |
|---|---|---|
| NAME | DATATYPE | NULLABLE? |
| CUSTOMER_ID | VARCHAR | NO |
| FIRST_NAME | VARCHAR | NO |
| LAST_NAME | VARCHAR | NO |
| BIRTH_DAY | TIMESTAMP | NO |
| ADDRESS | VARCHAR | NO |
| ADDRESS2 | VARCHAR | YES |
| STATE | VARCHAR | NO |
| ZIP_CODE | INTEGER | NO |

| CUST_ORDER | | |
|---|---|---|
| NAME | DATATYPE | NULLABLE? |
| ORDER_ID | VARCHAR | NO |
| CUSTOMER_ID | VARCHAR | NO |
| STATUS | VARCHAR | NO |
| ORDER_AMOUNT | DECIMAL | NO |

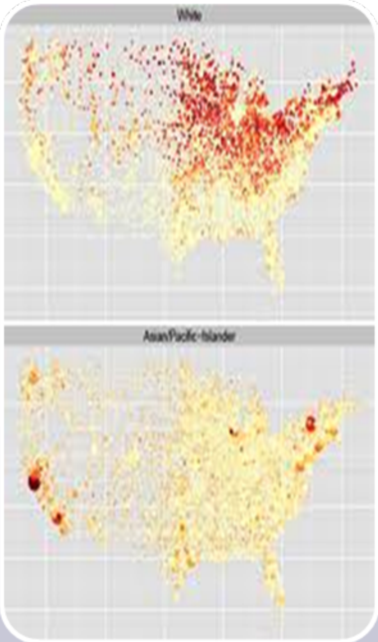
# Trends shrends!

- Gartner's 10 key IT trends for 2012
  - **unstructured data** will grow some 80% over the course of the next five years

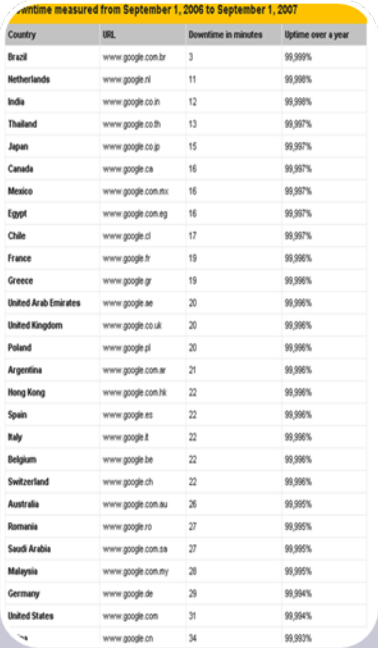


# Size matters but…

Performance



Geographic Distribution of users



wntime measured from September 1, 2006 to September 1, 2007

| Country | URL | Downtime in minutes | Uptime over a year |
|---|---|---|---|
| Brazil | www.google.com.br | 3 | 99,999% |
| Netherlands | www.google.nl | 11 | 99,998% |
| India | www.google.co.in | 12 | 99,998% |
| Thailand | www.google.co.th | 13 | 99,997% |
| Japan | www.google.co.jp | 15 | 99,997% |
| Canada | www.google.ca | 16 | 99,997% |
| Mexico | www.google.com.mx | 16 | 99,997% |
| Egypt | www.google.com.eg | 16 | 99,997% |
| Chile | www.google.cl | 17 | 99,997% |
| France | www.google.fr | 19 | 99,996% |
| Greece | www.google.gr | 19 | 99,996% |
| United Arab Emirates | www.google.ae | 20 | 99,996% |
| United Kingdom | www.google.co.uk | 20 | 99,996% |
| Poland | www.google.pl | 20 | 99,996% |
| Argentina | www.google.com.ar | 21 | 99,996% |
| Hong Kong | www.google.com.hk | 22 | 99,996% |
| Spain | www.google.es | 22 | 99,996% |
| Italy | www.google.it | 22 | 99,996% |
| Belgium | www.google.be | 22 | 99,996% |
| Switzerland | www.google.ch | 22 | 99,996% |
| Australia | www.google.com.au | 26 | 99,995% |
| Romania | www.google.ro | 27 | 99,995% |
| Saudi Arabia | www.google.com.sa | 27 | 99,995% |
| Malaysia | www.google.com.my | 28 | 99,995% |
| Germany | www.google.de | 29 | 99,994% |
| United States | www.google.com | 31 | 99,994% |
| [illegible] | www.google.cn | 34 | 99,993% |

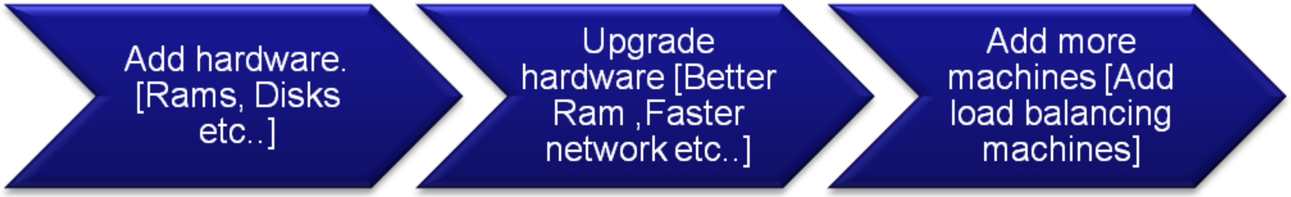
Availability

Scalability

# RDBMS..hmmm

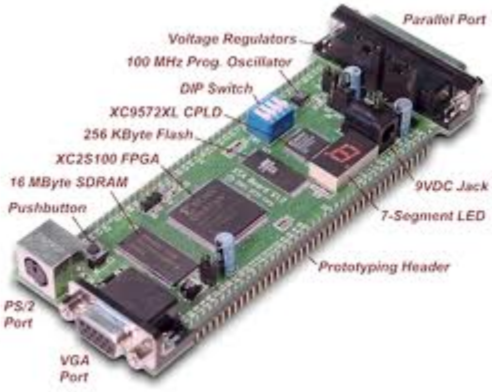
- Normalized implies Joins which implies Slow Queries /Complications
- Consistency = locks /transactions = Performance issues in distributed environments
- Scalability becomes a mess as our apps grow in size and demand

# Current Approach to Scalability

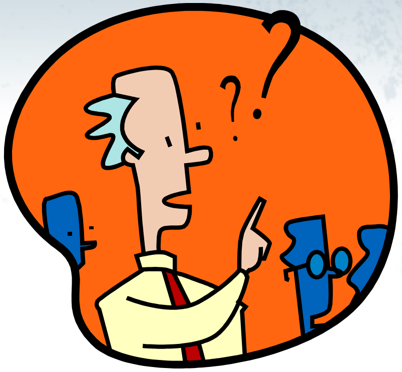

Add hardware. [Rams, Disks etc..]

Upgrade hardware [Better Ram ,Faster network etc..]

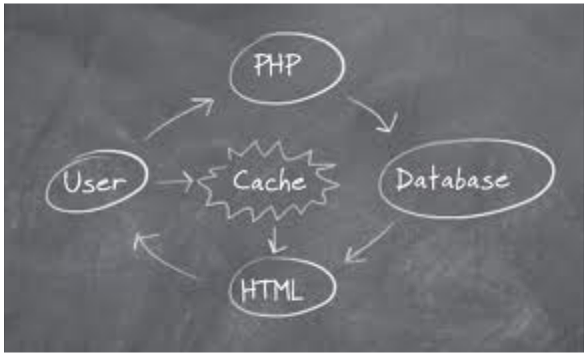
Add more machines [Add load balancing machines]



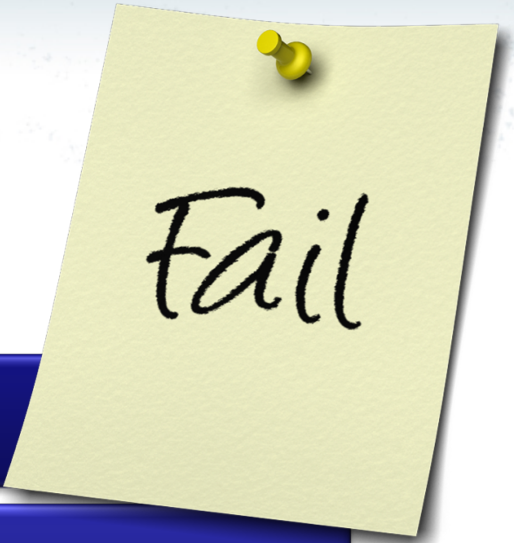
# Current Approach to Scalability



- Turn off “un-wanted” services [Journaling, Logging Backup etc..]
- Bring in caching ? Consistency of cache and DB
- De Normalise ?! Diluted form the pure
- Finally look at the application itself



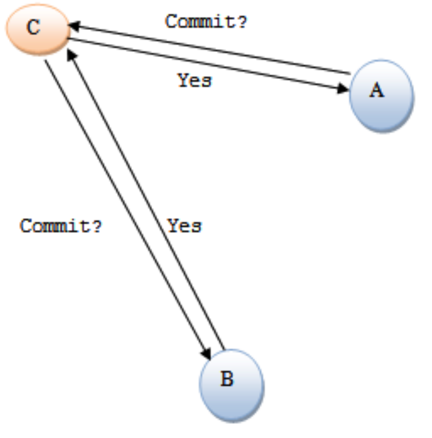
# RDBMS ..tends to

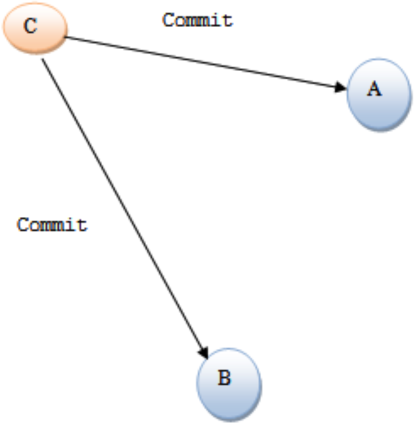


- Massive [terabytes]
- Elastic scalability
- Easily achieve Fault tolerance
- Tunable Consistency

# But Why..

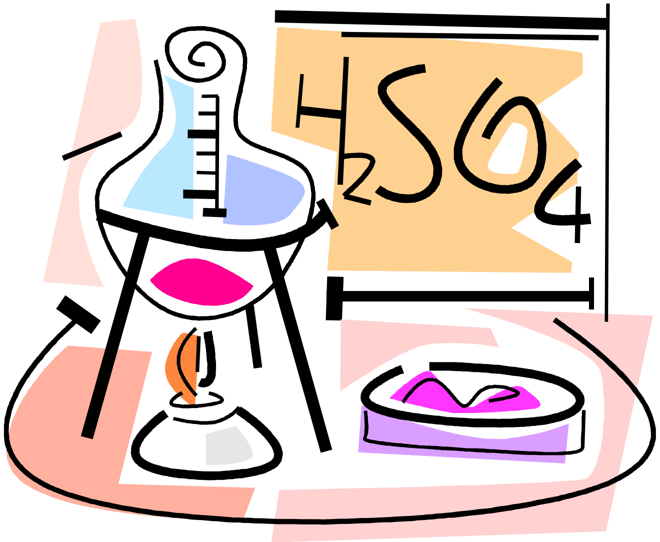
- ACID
- :- transaction slow under heavy load
- :- in distributed /replicated environment = 2 phase commit => infinite wait by either NODE or Coordinator





# Thanks to ACID we have:

- Loss of availability
- Higher latency during partial failures in distributed systems



# Cassandra to the Rescue!

- Open source,
- Distributed, Decentralized,
- Elastically scalable
- Highly available / fault-tolerant
- Tune ably consistent
- Column-oriented database
- Automatic sharding

# Distributed and Decentralized

Can be running on multiple machines

- appearing to users as single instance

DECENTRALIZE
TO
SURVIVE

Decentralized

- that there is no single point of failure.
- All the nodes in cluster function exactly the same [server symmetry]

# Elastic Scalability

- Vertical scaling : more hardware capacity /memory

- Horizontal scaling :

  More machines that have all or some of the data

  So that no machine is bearing the complete load

# Elastic Scalability

- *Elastic scalability :*
  - Cluster will be able to **seamlessly scale up** and **scale back down**

# Scale UP

- Add nodes and they can start serving clients!
  - – NO server restart / NO query change / NO balancing
  - – JUST add an another machine.

# Scale Down!

- Just unplug the system.
  - – Since cassandra has multiple copies of the same data in more than one node [configurable] there wont be any loss of data.

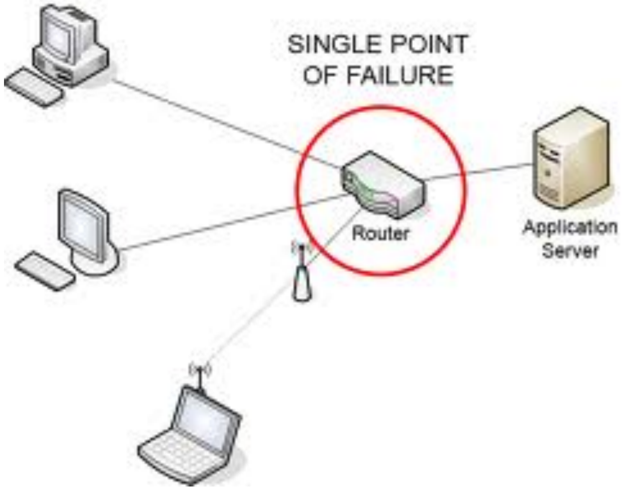
# High Availability and Fault Tolerance

- High availability + central server based system = problem
  - – Internal Hard ware redundancy ,Hot Swap
  - – Sounds cool but Extremely Costly

# Single Point Failure

- Master Slave issue

"I THOUGHT WE WERE JUST BUYING A HOUSE!"



# High Availability and Fault Tolerance

– Cassandra allows to :

- replace failed nodes in with no downtime
- replicate data to multiple data centers to prevent downtime  [automatic]

# Tuneable Consistency

- *Consistency :* All Reads return the most recently written value
  - Cassandra is “eventually consistent” model **by defau**

# Eventually consistency is for Kids!

- “My data is very important and CANT tolerate any kind of inconsistency”

# But then!

- Amazon, Facebook, Google, Twitter which uses this model.
  - **DATA** is their main sales item
  - High performance!

PLACE STAMP HERE
GOOGLE
CLASSIC
QUERY:
□ IMAGES □ NEWS □ VIDEO □ MAPS □ OTHER
SEND YOUR QUERY TO: GOOGLE INC., 1600 AMPHITHEATRE PARKWAY, MOUNTAIN VIEW, CA 94043, UNITED STATES
PLEASE ALLOW 30 DAYS FOR SEARCH RESULTS

# Closer look on consistency

- *degrees* of consistency
  - Strict consistency:
    - any read will always return the most recently written value
  - What is meant by “most recently written”?
  - And more over Most recently to who?

- Geographically dispersed data centers + servicing multiple requests form multiple clients; the answer is no more simple.

# *Weak (eventual) consistency*

- The system will be in Consistent state; in defined predictable future..but not NOW.

# Lets write and worry about reading later!

- Cassandra choose to be always writable
  - opting to defer the complexity of reconciliation to read operations
  - tremendous performance gains

# Tuneable Consistency Technicalities

- *Tunealble Consistency :*consistency level *against the replication factor.*

  – ***replication factor*** *[cluster setting]*
    - the number of nodes in the cluster you want the updates to propagate

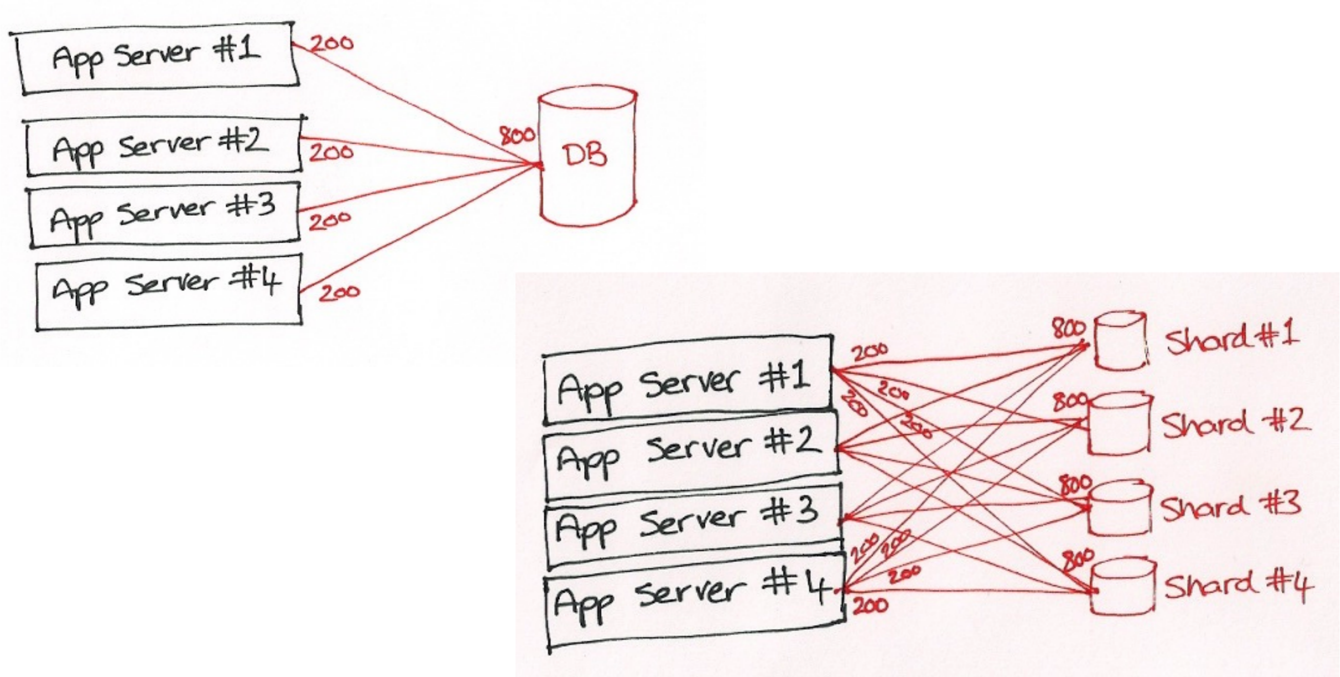
# Tuneable Consistency Technicalities

- *consistency level [Client operation setting]*
  - how many replicas
    - must acknowledge a write operation
    - respond to a read operation
    - for the operation to be considered successful

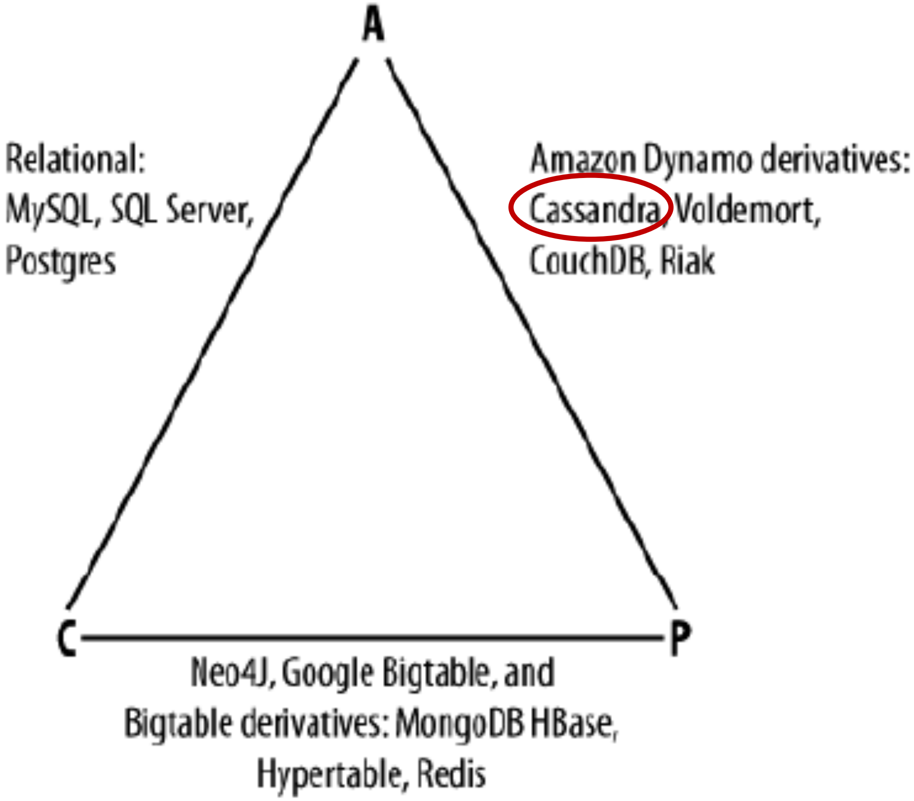
# Tuneable Consistency Technicalities

- If CL= RF
  - – High consistency
    - Low performance
    - Availability hit!

# Sharding


App Server #1
App Server #2
App Server #3
App Server #4
200
200
200
200
800
DB
App Server #1
App Server #2
App Server #3
App Server #4
200
200
200
200
200
200
200
200
800
800
800
800
Shard #1
Shard #2
Shard #3
Shard #4

# Brewer's CAP Theorem

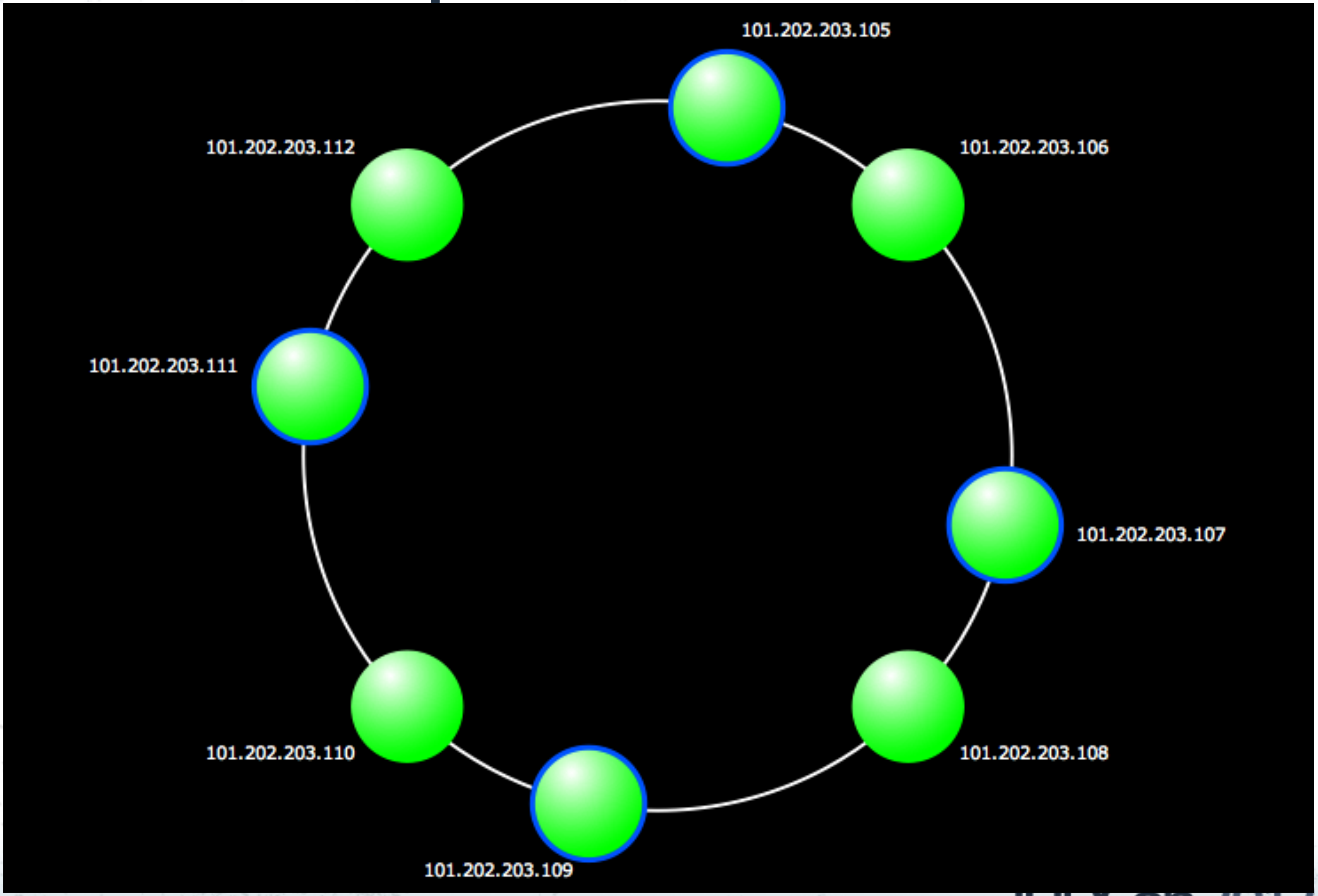


Relational:
MySQL, SQL Server,
Postgres

Amazon Dynamo derivatives:
Cassandra, Voldemort,
CouchDB, Riak

Neo4J, Google Bigtable, and
Bigtable derivatives: MongoDB HBase,
Hypertable, Redis

# Top Down Look :



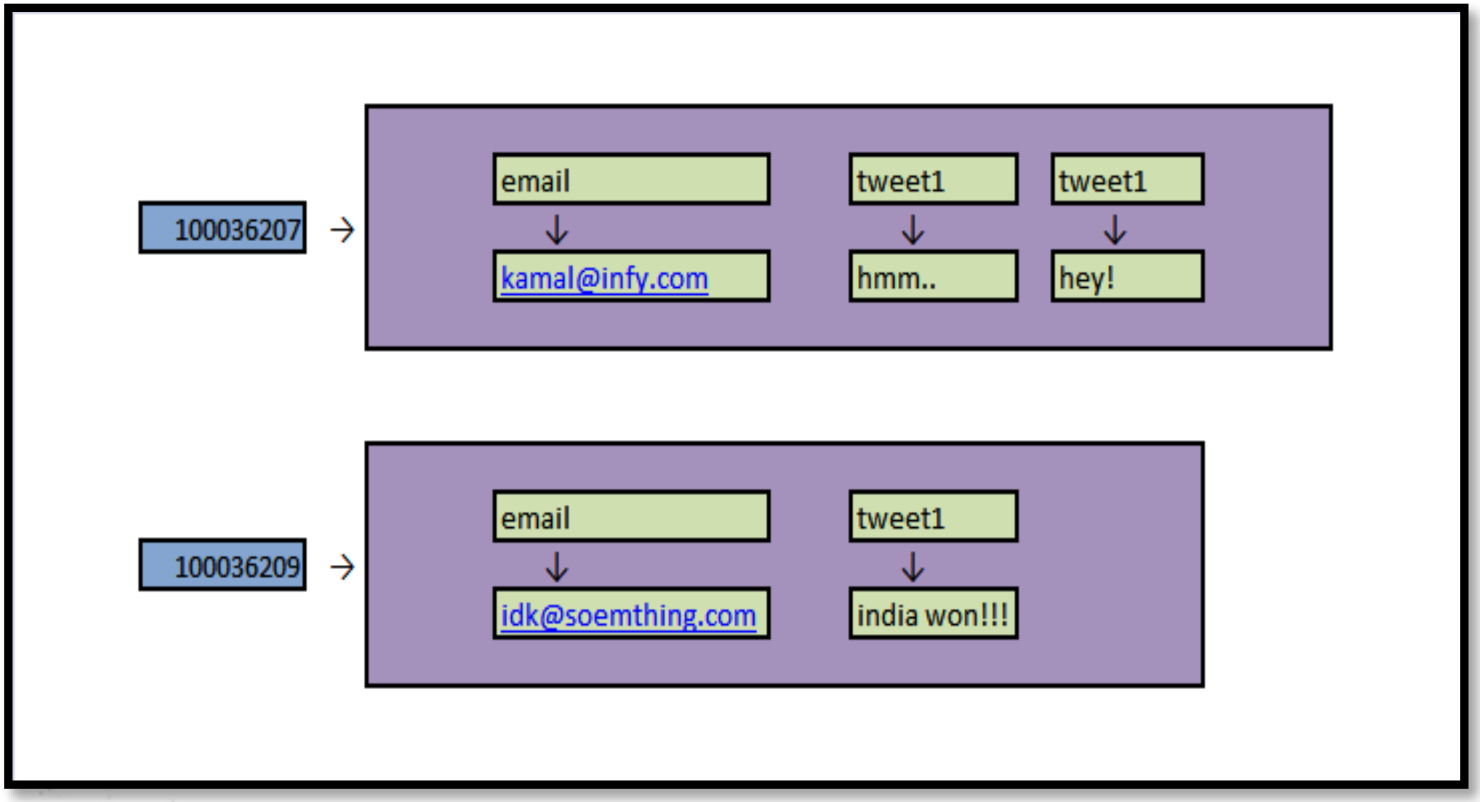
# Node :

- Holds replica for different ranges of data.
  - Fail over nodes
  - peer-to-peer protocol [gossip architecture]

# Keyspaces

- Cluster is a container for keyspaces
- *keyspace :*outermost container for **data** in Cassandra
  - – a bunch of attributes which define keyspace-wide behavior

# Data Model of Cassandra :ColumnFamily



100036207 →

email → kamal@infy.com

tweet1 → hmm..

tweet1 → hey!

100036209 →

email → idk@soemthing.com

tweet1 → india won!!!

# Lets see if we can get it right!

Company {

key: 1001 { name: Apple, phone: 111-444-4444, address: 400 N. Hayden Rd., state: AZ }

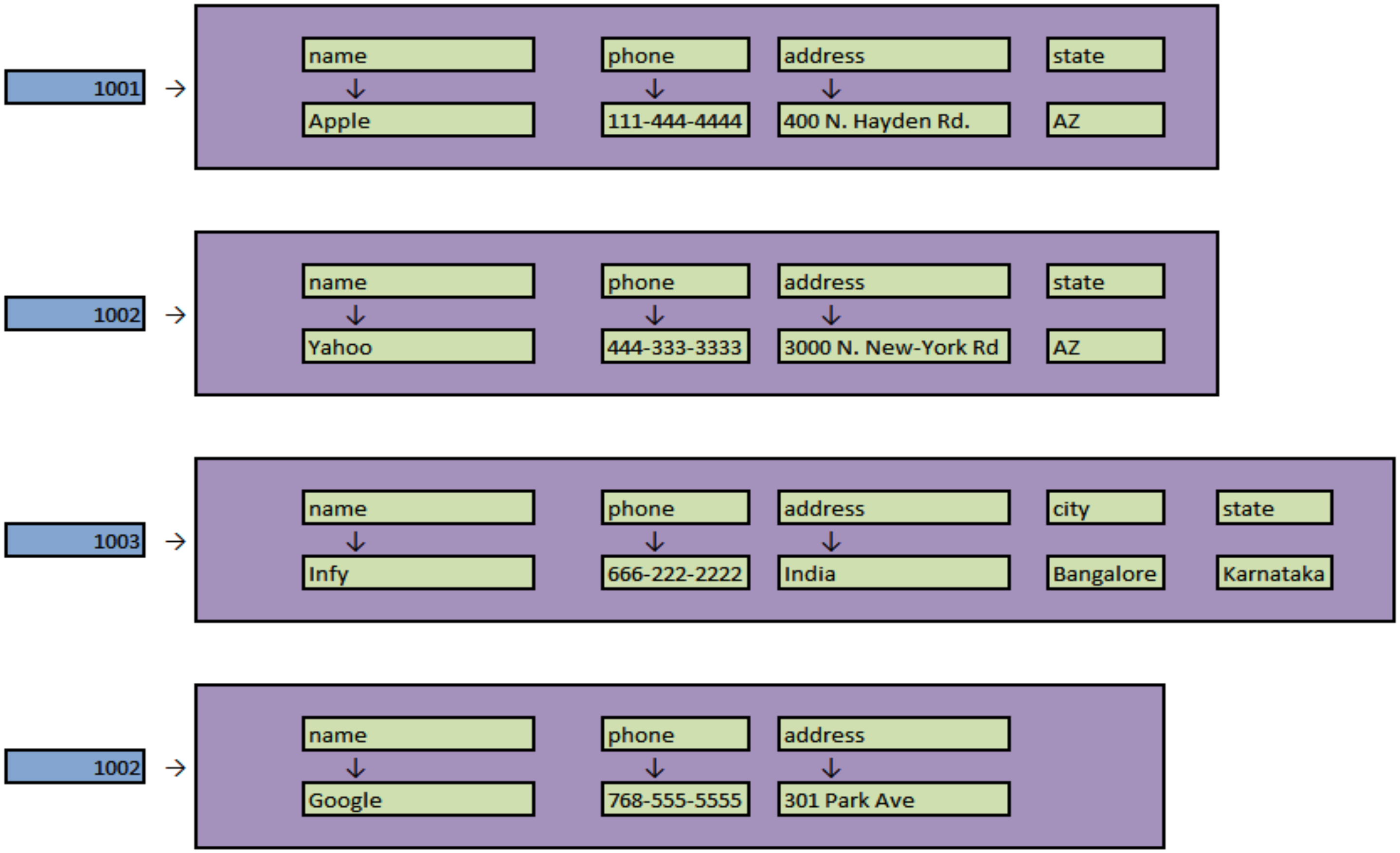
key: 1002 { name: Yahoo, phone: 444-333-3333,address: 3000 N. New-York Rd, state: AZ }

key: 1003 { name: Infy , phone: 666-222-2222, address: India , city : Bangalore,state: Karnataka}

key:1 004 { name: Google, phone: 768-555-5555, address: 301 Park Ave}

}

# Tada!



1001 →

| name | phone | address | state |
|---|---|---|---|
| Apple | 111-444-4444 | 400 N. Hayden Rd. | AZ |

1002 →

| name | phone | address | state |
|---|---|---|---|
| Yahoo | 444-333-3333 | 3000 N. New-York Rd | AZ |

1003 →

| name | phone | address | city | state |
|---|---|---|---|---|
| Infy | 666-222-2222 | India | Bangalore | Karnataka |

1002 →

| name | phone | address |
|---|---|---|
| Google | 768-555-5555 | 301 Park Ave |

# Deep dive into Architecture

- System Keyspace
- Peer-to-Peer
- Gossip and Failure Detection
- Anti-Entropy and Read Repair

# Deep dive into Architecture

- Memtables, SSTables, Commit Logs
- Hinted Handoff
- Bloom Filters
- Tombstones

# Deep dive into Architecture

- System Keyspace
  – internal keyspace called system /store metadata about the cluster
- Stores metadata for the local node
  – And Hinted handoff information

# Peer-to-Peer : p2p

- MySQL, Bigtable etc
  - – Some nodes are masters and some are slaves
- Disadvantage:
  - – replication is one-way [master -> client]
  - – Ie : all writes must be sent to the master
    - potential single point of failure
    - Performance bottle neck

# Peer-to-Peer :

- Cassandra has a peer-to-peer
  - any given node is identical to any other node
- Advantages: availability/scaling

# Gossip and Failure Detection

- Goals :
  - Decentralization / Partition tolerance

- Uses *gossip* protocol:
  - In short :gossip is used for failure detection
  - *gossiper* runs every second on a timer

# Cassandra loves Gossip :O

- “gossip protocol” originally coined in 1987 by Alan Demers,
  - who was studying ways to route information through unreliable networks
  - Based on the concept of human gossip
  - assume a faulty network

# What happens when its not all ideal!?

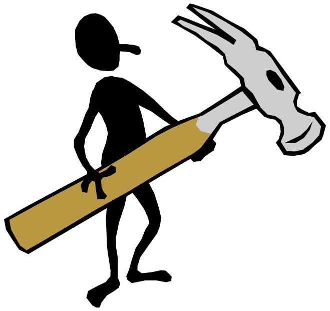
- When G finds that endpoint is dead,
  - “convicts” that endpoint ie it marks it as dead in the local list and logs this fact
  - Also known as Accrual failure detection
    - failure detection should be flexible
      - Achieved by decoupling main application from the responsibility of failure detection

# No Heart beat = Dead right?

- Heartbeat vs suspision level
  - If no heart beat = dead [traditional]
  - If no response = possibly dead!
    - account fluctuations in the network environment

# Anti-Entropy and Read Repair

- *Anti-entropy* is the replica synchronization mechanism which ensures that data on different nodes is up to date with the newest version.



# Read Repair

- When a client reads a data
  - Some of them may have old data.
- – Now read repair starts
  - better probability of getting most recent data.

# Memtables, SSTables, and Commit Logs

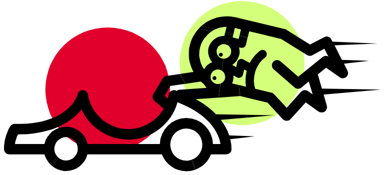
- Durability
  - Once written never lost
  - Commit logs :all writes go in for recovery
- *memtable*
  - memory-resident data structure
  - When contents become too big. Flushed into SStable
- SSTable : File in Hdisk

# Hinted Handoff

- Node which was supposed to hold data is Down!
    - “I have the write information that is intended for node B. I’m going to hang onto this write, and I’ll notice when node B comes back online; when it does, I’ll send it the write request”
  - User can keep on writing.

# Bloom Filters

- Goal : performance booster



- very fast, nondeterministic algorithms for testing whether an element is a member of a set
- The filters are stored in memory and are used to improve performance by reducing disk access on key lookups

# Tombstones

- idea similar to “soft delete.”
  - to support audit trails
- On execute of a delete operation, the data is not immediately deleted

# Thank You

Disclaimer : All logos and images belong to the creator and companies which own them