Introducing Cassandra with Time Series Data

There is no silver bullet,
But sometimes a lot of lead ones do the trick.

The Plan

- Talk about some of Cassandra's Use Cases (and anti-use cases)
- Talk about how Cassandra is architected, and why it fits the use cases discussed previously.
- Get In The Code!
  - Deploy a Cassandra cluster.
  - Look at actual time-series data consumed into the cluster.
  - Talk about how data choices affect performance.

A Warning to the Smarties

- In this presentation I will make broad, sweeping generalizations, factual errors, and possibly even quote a statistic or wikipedia, both of which will be made up.

- That being said, I've run Cassandra in production for a few years now, and have the scars to prove it. Only you can decide if I have a clue or not.

A Warning to the Smarties

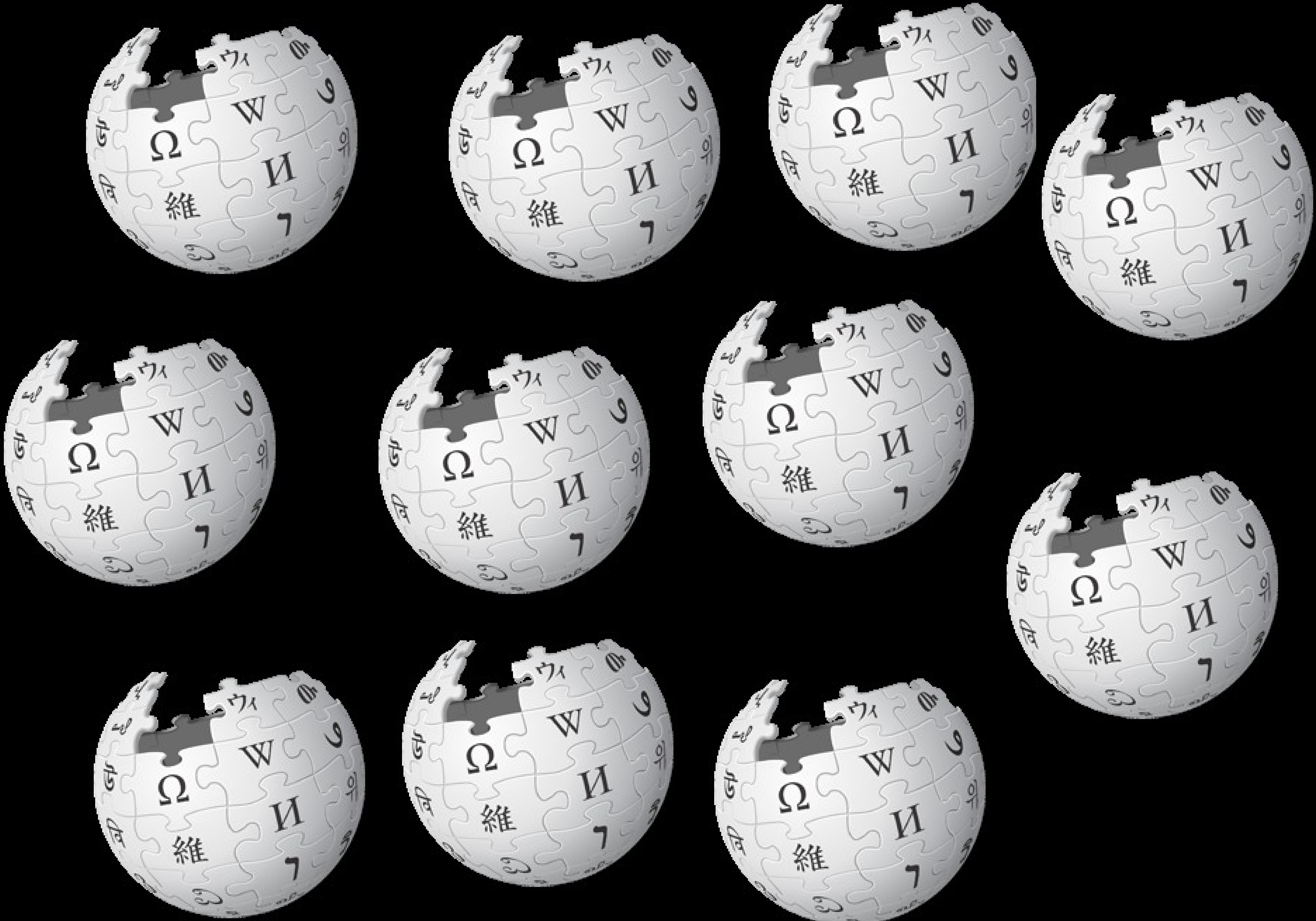
- I will try very hard not to read the slides to you.
- Please stop me if you have questions.

Use Cases

Cassandra is a database, sooo....

- Lots of Data
  - Traditional databases (MySQL, PostGRES) fall down, or require increasingly complex configuration.
  - Single commodity machines can't handle the data.

  If you have <= 1M rows, and <= 2TB of data, a traditional DB may be better.



Use Cases

- Very Fast Failover / Real Multi-Master Mode
  - A Single Database Machine Failing is a use case you *must* handle with 0 downtime
  - You don't have a Multi-Master MySQL Guru already on staff.

  If your application will be fine if you just restore the SQL dump from last week tomorrow morning, a traditional database may be for you!

Use Cases

- More writes than you can shake a stick at!
  - The ingress of data is capable of saturating multiple network cards.
  - The ingress of data has significant hotspots, which could require boatloads of sharding, or some semi-maintained auto-sharding proxy some dude wrote for his PHD thesis in 2009.

Thanks Pluma

Use Cases

- Geographic Distribution
  - You have data that you need to have locality in geographically diverse regions.

Anti-Patterns

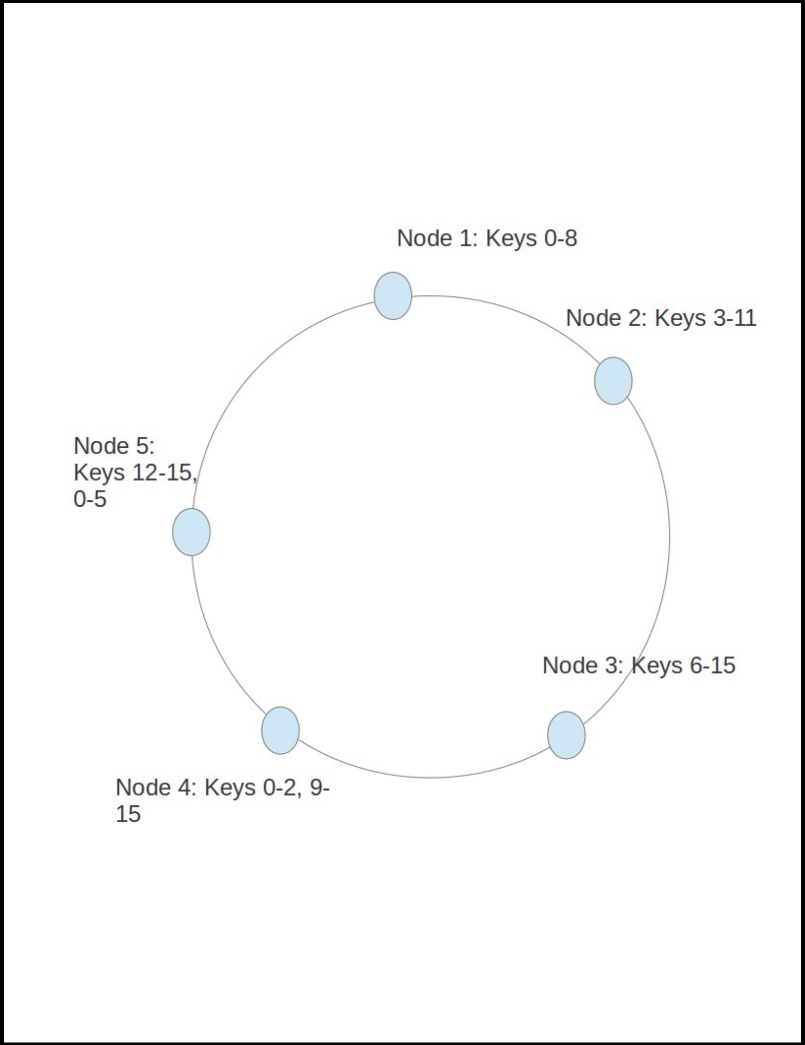
- 3 or fewer nodes.
- Static Data
- “I'm still running on CentOS 3”
- “I refuse to try to understand a consistency model. I don't know what ACID is, but I've heard it's over-rated.”
- I need ad-hoc structured queries.

So what the Heck Is It?

“Cassandra provides a structured key-value store with tunable consistency.” - Wikipedia

- Bigtable-style sparse 3-deep tree data store
- Amazon Dynamo-style headless reliability.
- “Tunable Consistency”
- Replication Built in from the start!

So what does it look like?



So what does tunable consistency mean?

You pick how many replicas you write a change (“Mutation”) to.

You pick how many replicas you need to check for data.

So what does tunable consistency mean?

You pick how many replicas you write a change (“Mutation”) to.

You pick how many replicas you need to check for data.

You can have as much ammo as you want to shoot yourself in the foot!

So how does it write?

1. Take your key.
2. Compute the hash.
3. Find the replicas.
4. Send to any available.
5. Wait for the “tuned” number to confirm.
6. Ack the client

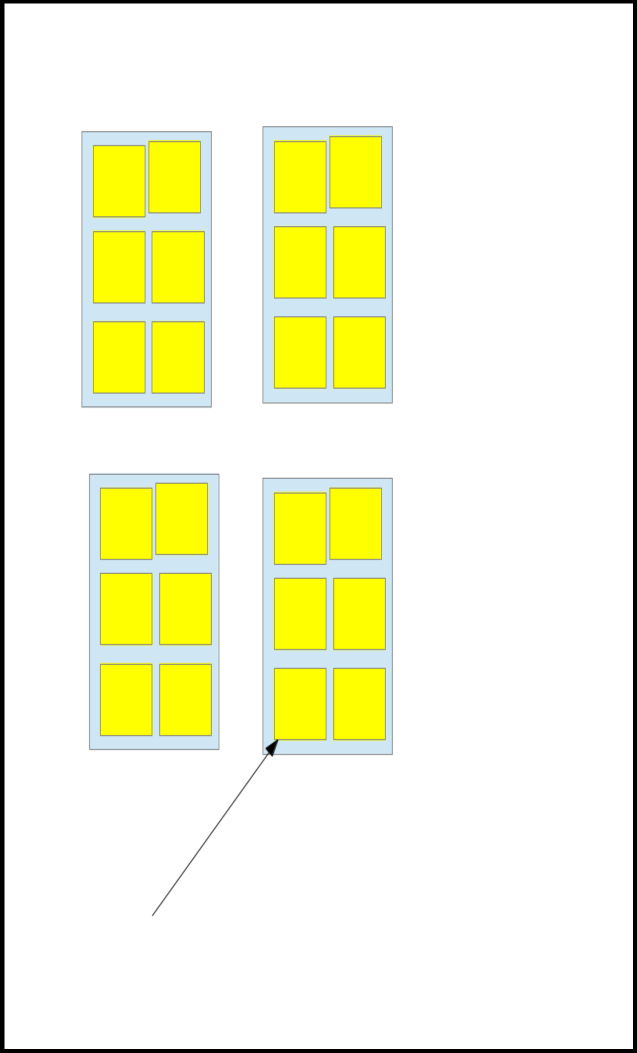
So how does it read?

1. Take your key.
2. Compute the hash.
3. Ask the replicas.
4. Wait for the “tuned” number to respond.
5. Sanity-check responses.
6. Send data back to the client.

Why time series data?

- Many things that store time series data use it as the primary key. This means normal row-based datastores that store nearby primary keys next to each other end up contending for a single resource, *even if they're distributed*.

Why time series data?



Hashing: The Upshot

- Hashes don't usually preserver order.
- Hashes aren't always reversible. (these are not)
- Hashes don't really <u>prevent</u> hotspots.

What now?!? Well, depends (as always) on the use case.

Hashing: The Upshot