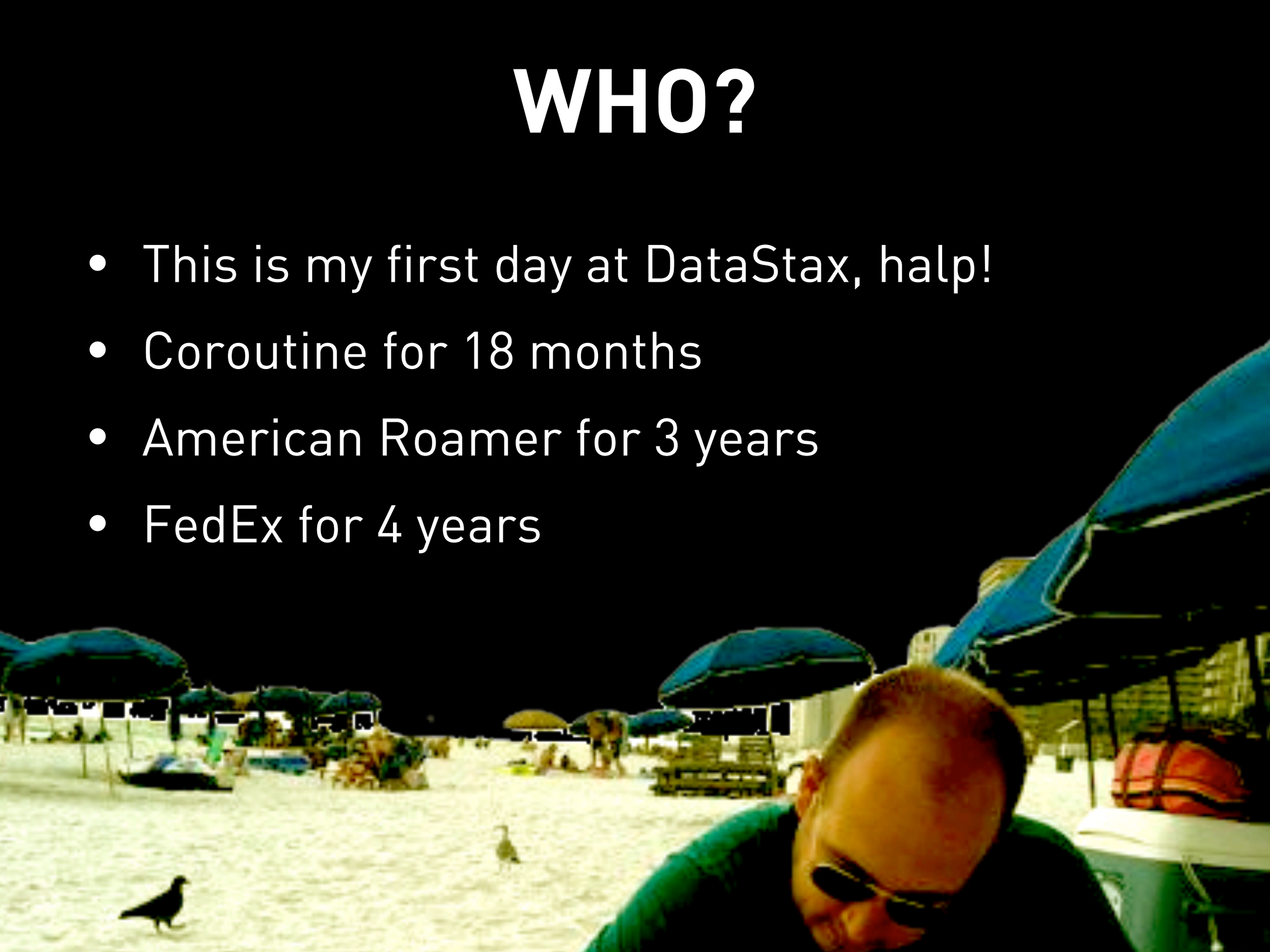
HOW DO I CASSANDRA?

Rick Branson, DataStax

Memphis Python Users Group
November 21st, 2011

WHO?

- This is my first day at DataStax, halp!
- Coroutine for 18 months
- American Roamer for 3 years
- FedEx for 4 years

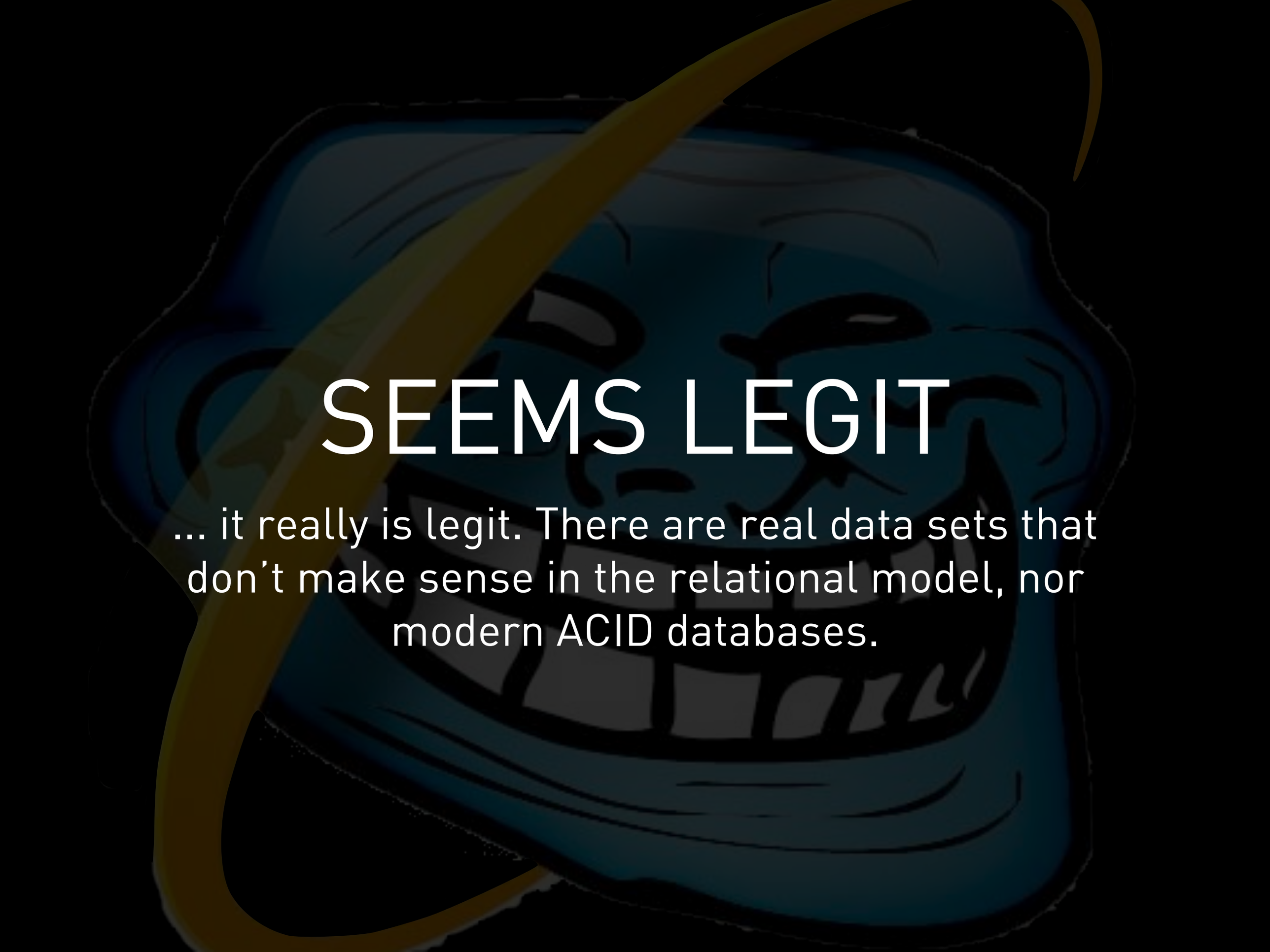


OCCUPY SQL

Why is it that 99% of the applications
go into 1% of database softwares?

HOW DO I NOSQL?

1. Anvil transitions. Deal with it.
2. Find data store that doesn't use SQL
3. **ANYTHING**
4. Cram all the things into it
5. Triumphantly blog this success
6. Complain a month later when it bursts into flames

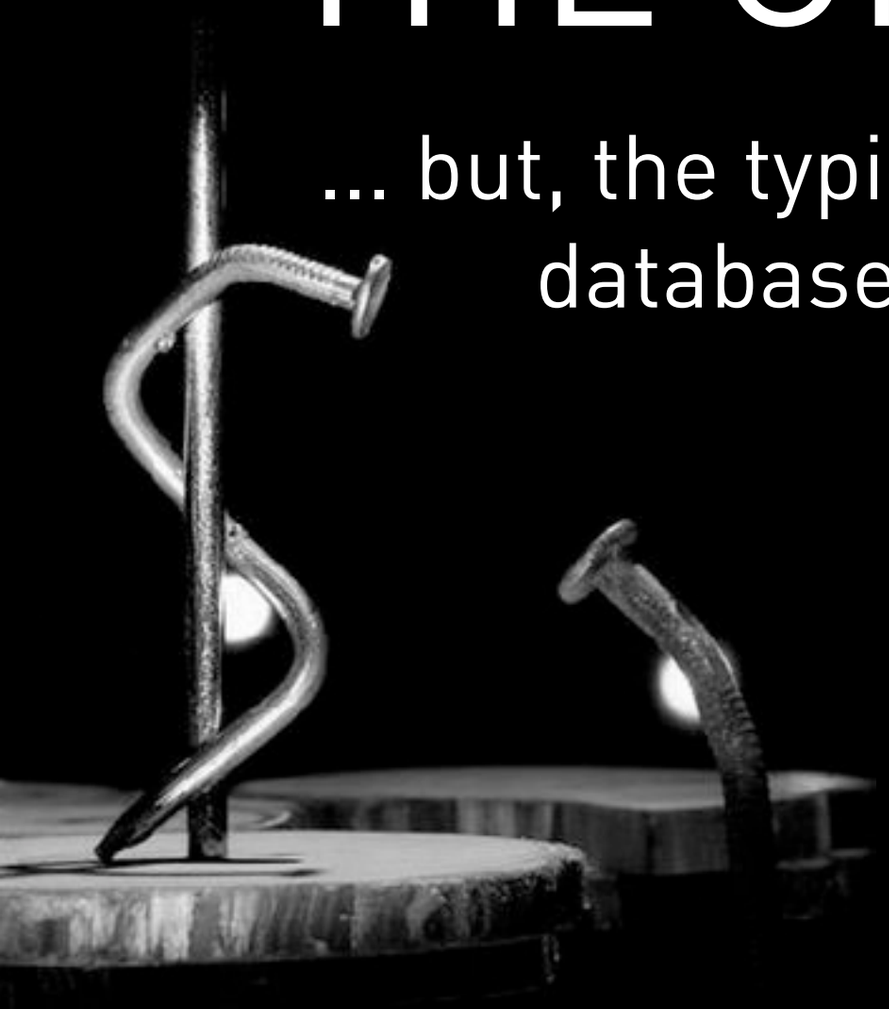


SEEMS LEGIT

... it really is legit. There are real data sets that don't make sense in the relational model, nor modern ACID databases.

THE GREAT HAMMER

... but, the typical developer only has a relational database; everything looks like a nail.



SNAKE OIL

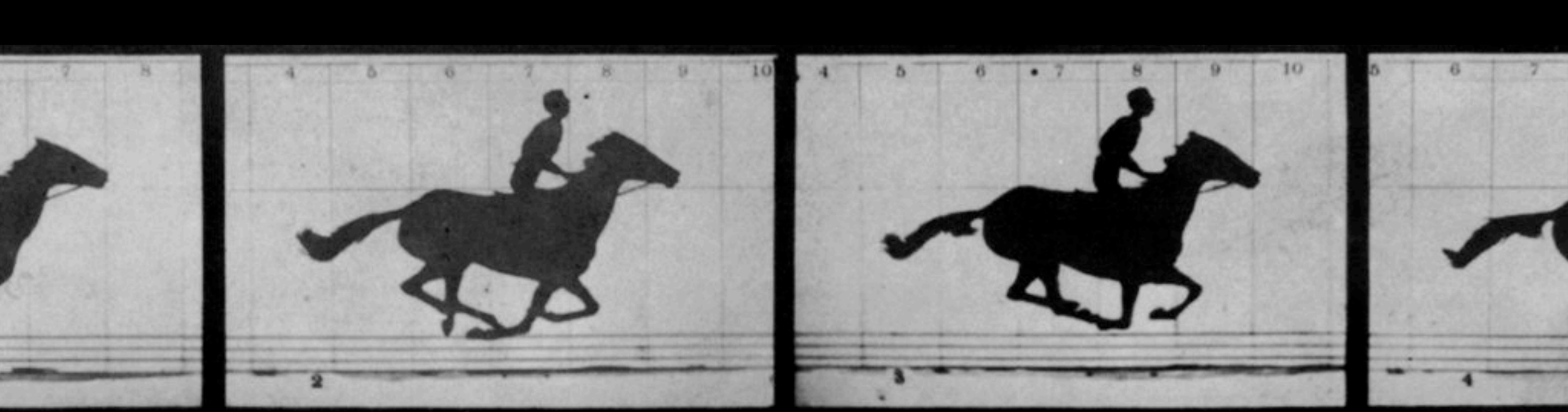
Typically, developers pick data structures suited to each use case. The purveyors of relational would have you believe everything fits into one mold.

FIT WHAT INTO WHERE?

Trees, Semi-Structured Data, Web Content,
Multi-Dimensional Cubes, Graphs

HAMMERS GONNA HAMMER

It does work – so does COBOL. Just like a motion picture film, the speed of modern computers creates an **illusion**.



MAKE A WHAT OUTTA WHO?

ASSUMPTIONS MADE ON YOUR BEHALF

- Data **must** be synchronized everywhere, instantaneously
- The structure of data doesn't change over time
- Hardware never fails and networks are perfect
- Humans always do the right thing
- No upgrading or patches are ever needed
- Planned outages are good for the soul

THE MOVEMENT

"No one tool is appropriate for solving every problem."

~ Cliff McKinney

Gettysburg Address, cir. 1863

SRS BZNS

10gen, Basho, DataStax, Cloudant, Cloudera,
HortonWorks, VMware, Oracle

Guess the number of craps Russ here gives about databases?

MOVING ON

OH: “The last DBA in the room went **data**way”

AN DEFINITION

Cassandra – affectionately **“C*”** – is an open source, distributed store for structured data that scales-out on cheap, commodity hardware and stays up **even when things get really bad.**

FUH-GET ABOUT IT

- ~~Strict Consistency~~

Ruins availability

- ~~Transactions~~

Deadlocks & Not scalable

- ~~Ad hoc Queries~~

aka "Dumbass" Queries

- ~~Indexes~~

Scumbag index covers every column; ignored by planner

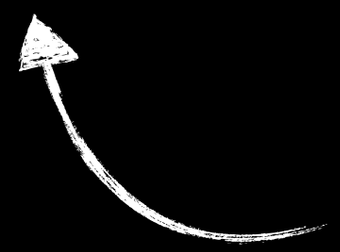
THE REWARDS

- **Simplicity of Operations**
- Transparency
- Very High Availability
- Painless Scale-Out
- Solid, Predictable Performance on Commodity and Cloud Servers

Cassandra is about **trading** some developer time for operational pain – or 3am phone calls as some of us know it.

DEM GOOD GENES

Put simply: the scalable, efficient storage model from **Google's BigTable** atop the incredibly fault resistant distributed design of **Amazon's Dynamo**



resisting, not just tolerating!

VS OTHER NoSQL

- Truly Distributed Design
- No Special Servers, No SPOF
- 2D Data Model
- Rocks on Spinning Disks
- Single-Server Durability

HORRIBLE IDEAS

THINGS NOT TO USE CASSANDRA FOR

- Prototyping
- Temporal Data
- Distributed Locks
- Message Queues
- … anywhere availability doesn't matter

UNBOUNDED HUMANS

... OR DATA THAT THRIVES ON CASSANDRA

- Social

  Voluntary, human-generated data: text messages, comments, check-ins, real-time collaboration

- Activity

  Involuntary or indirectly human-generated data: news feeds, timelines, clickstreams, system + application logs

- Images

  Smaller files ← ~10MB fit well into columns

- ... anywhere availability is paramount,

  (not necessarily big data or scale either)

TO START

- `easy_install pycassa`
- Pretty much ignore any info pre-0.7
- Use DataStax Community Edition
- Unfortunately, the “official” wiki sucks, so use DataStax docs & tutorials
- IRC, Mailing List, StackOverflow, Quora all monitored regularly and kick ass

WHY CASSANDRA + PYTHON?

- **pycassa** is ~~one of~~ the best client
- Healthy Cassandra+Python Community
- Many examples of real, solid, production Python applications built on-top of Cassandra

PYTHON + CASSANDRA

NOTABLE IMPLEMENTATIONS

- Reddit
  Caches generated comment trees in Cassandra
- UrbanAirship
  Delivers 20M push notifications per day; bursts to 100k/sec+
- GameFly
  The database behind their social network
- Digg
  Pretty much the whole site

DATA MODEL

"I gotcha column family ratttt heeeeyyyyaahhh!"

AN COLUMN FAMILY

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

AN COLUMN FAMILY

- Don’t think in relational database terms
- Columns are `{ name, value }` tuples
- It’s dictionaries all the way down...

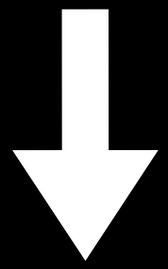
```
{ “key”: { “name”: “value” } }
```

Column Family

Row

Column

Row Key



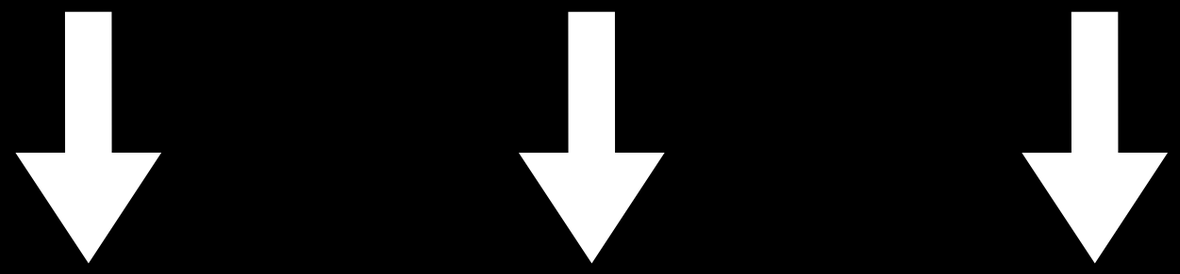
jim	age: 36	car: camaro	gender: M
carol	age: 37	car: subaru	gender: F
johnny	age:12	gender: M	
suzy	age:10	gender: F	

Row Key

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

Rows are **randomly** ordered

Columns



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

Column Names

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

Column Names

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

Column Name dictates order

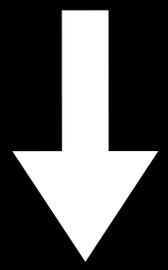
Column Values

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

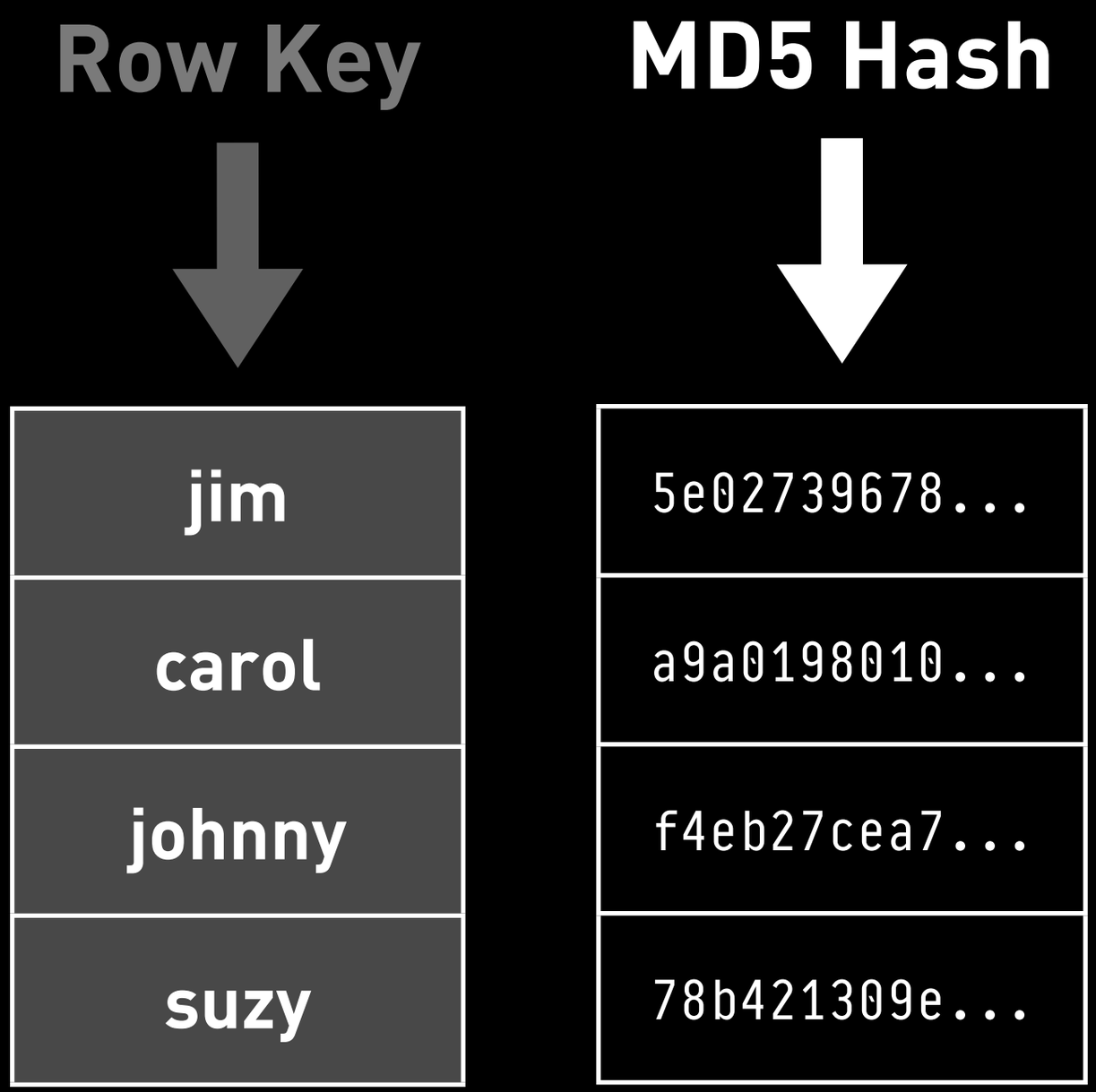
PARTITIONING

Cassandra divides the data set into ranges and distributes rows among multiple servers to achieve scale-out.

Row Key determines node placement



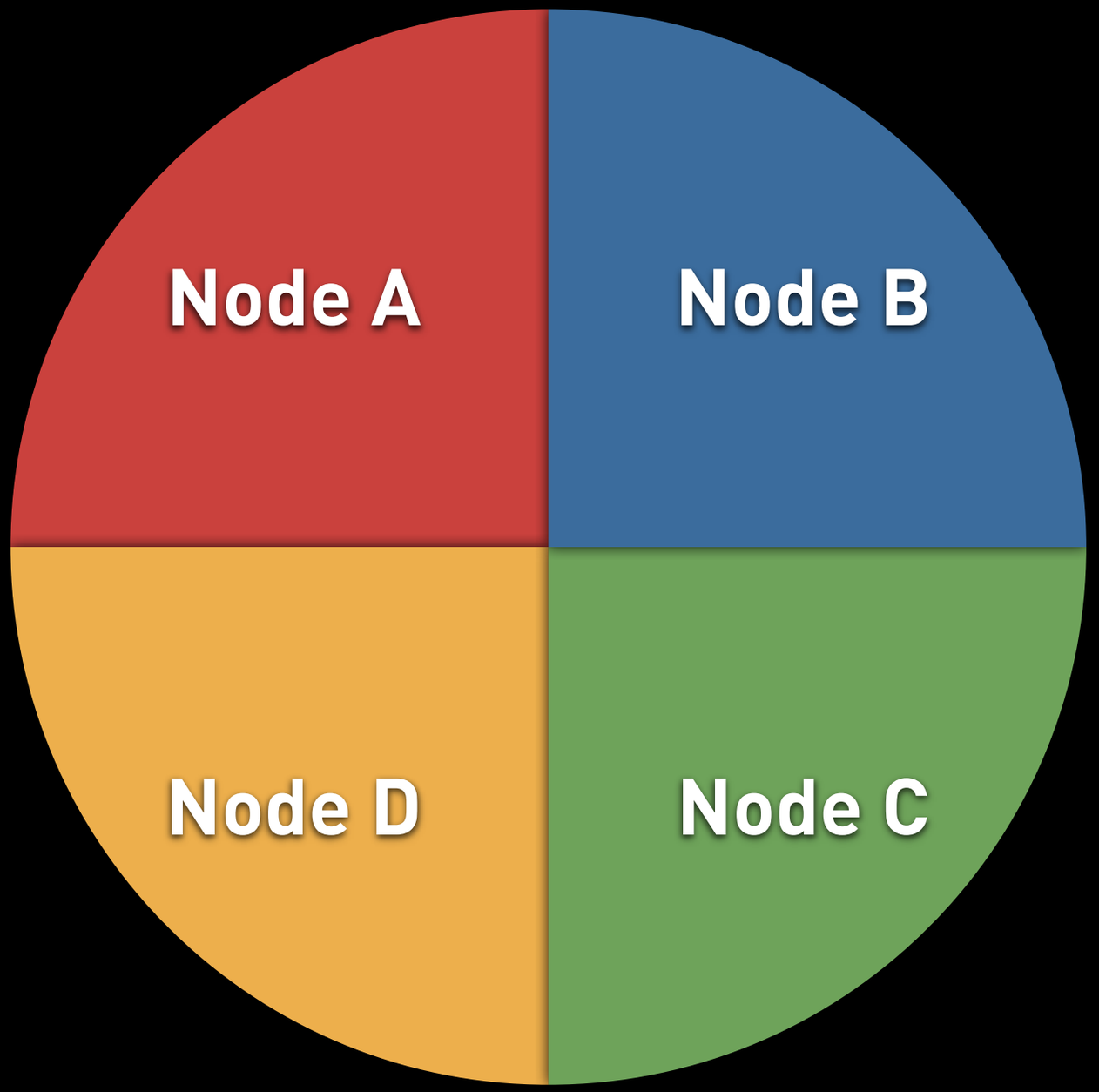
jim	age: 36	car: camaro	gender: M
carol	age: 37	car: subaru	gender: F
johnny	age:12	gender: M	
suzy	age:10	gender: F	



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

Conceptually, hashed keys are placed along a ring representing the keyspace.

The ring is divided up into ranges, one for each node in the cluster.



In a 4-node cluster, each node holds roughly a quarter of the rows.

	End (Token)	Start
Node A	0x00000000000000000000000000000000	0xc0000000000000000000000000000001
Node B	0x40000000000000000000000000000000	0x00000000000000000000000000000001
Node C	0x80000000000000000000000000000000	0x40000000000000000000000000000001
Node D	0xc0000000000000000000000000000000	0x80000000000000000000000000000001

	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...

	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...

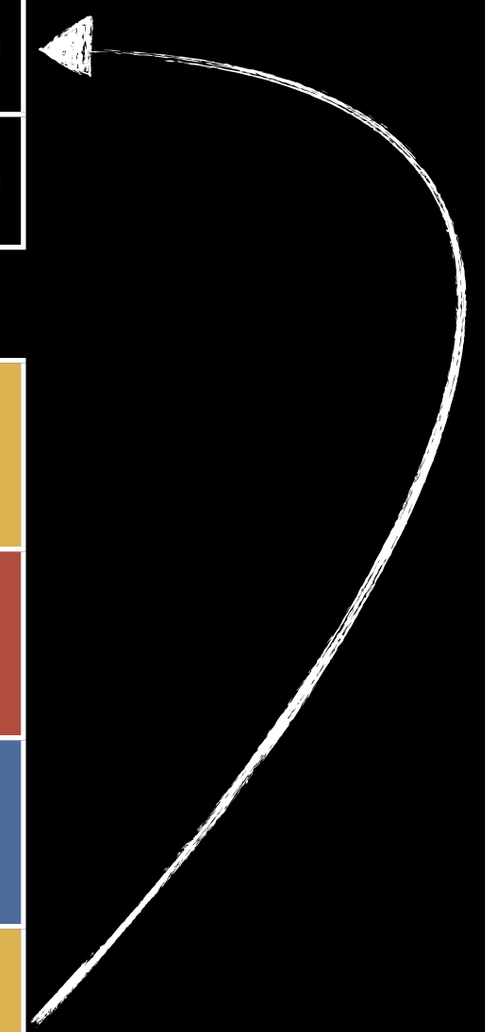
	Start	End
A	0xc000000000..1	0x0000000000..0
B	0x0000000000..1	0x4000000000..0
C	0x4000000000..1	0x8000000000..0
D	0x8000000000..1	0xc000000000..0

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

	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...

	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...

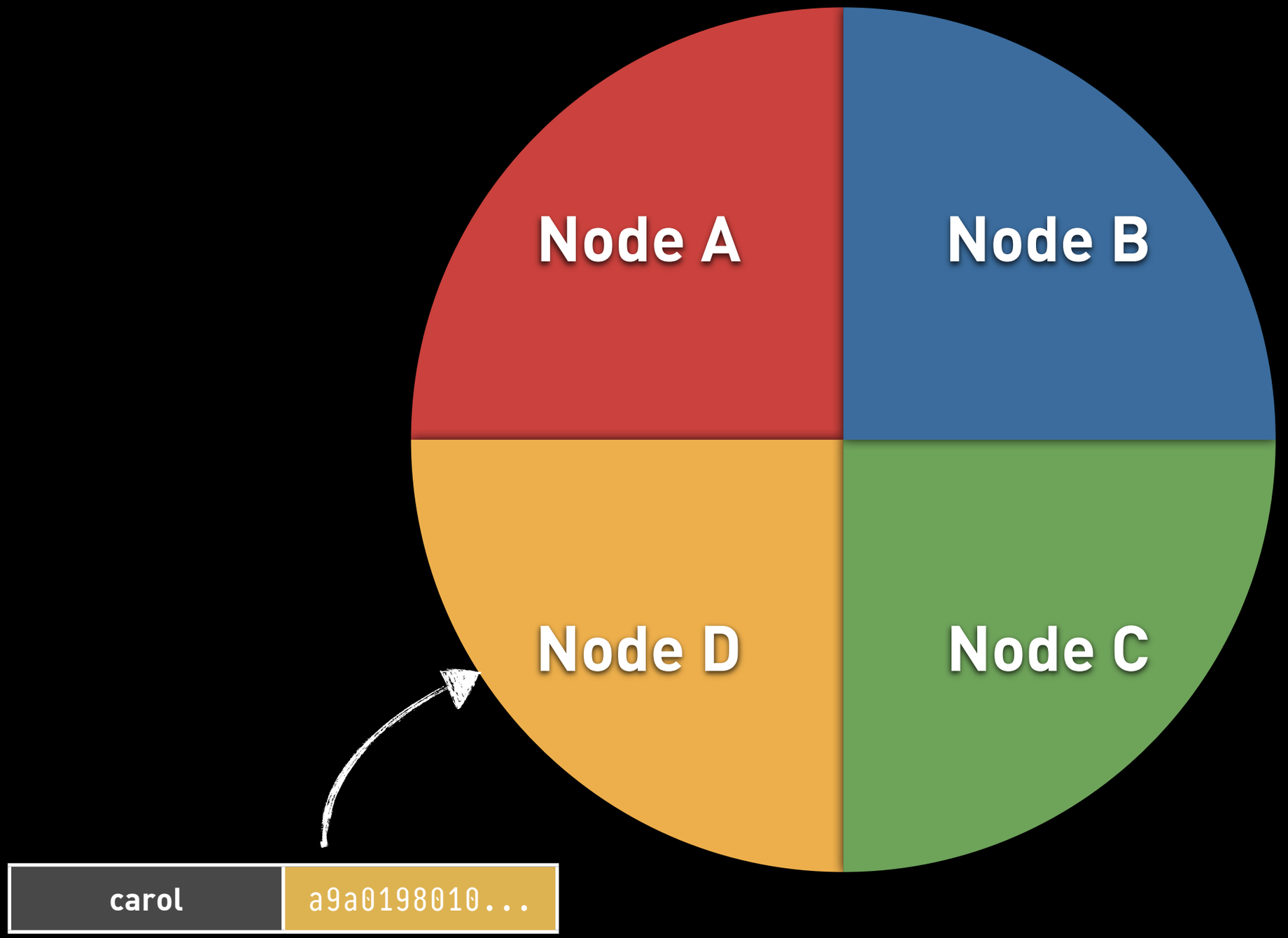
REPLICATION

Cassandra creates multiple copies of rows so that if any one node is behaving badly, all of the data stays available and fully operational.

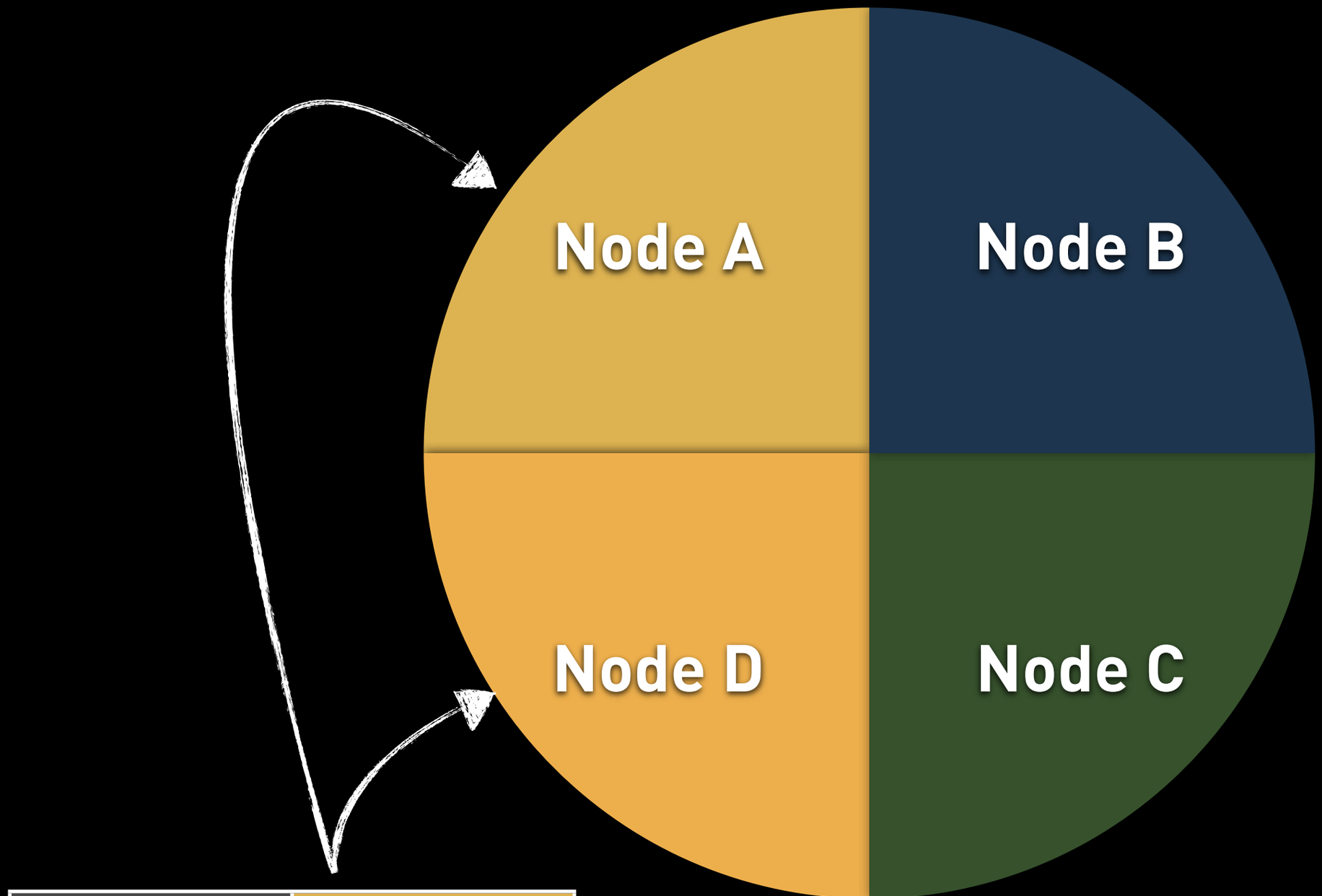
REPLICATION

- Column Family has “Replication Factor”
- RF = total number of copies of each row
- Don’t be fail; NEVER use 1
- Use 3+ in the “cloud”
- Use 2+ on redundant hardware
- Write operations on any replica

Replicas are chosen by jumping to the next node(s) on the ring

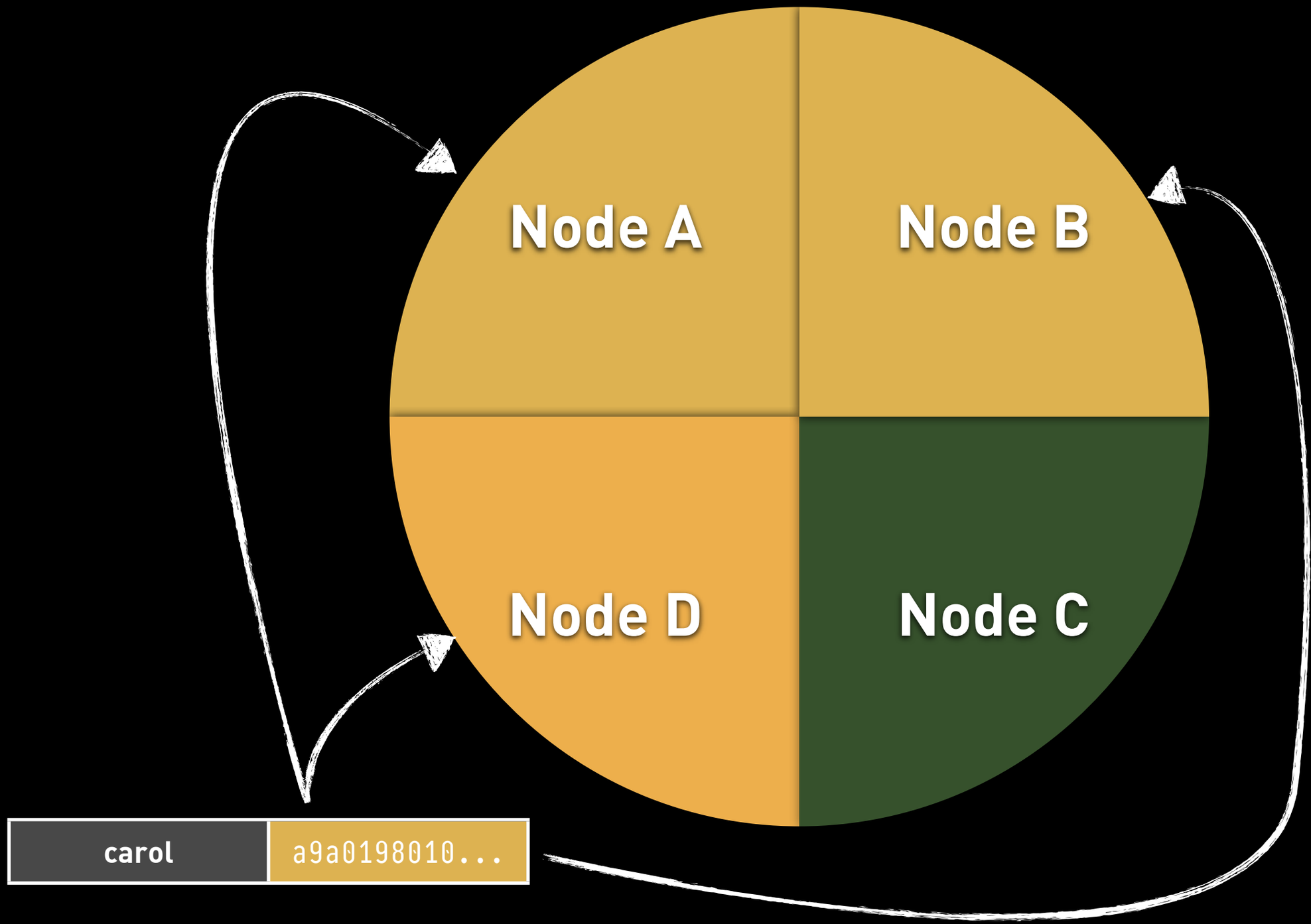


Replication Factor = 2



carol	a9a0198010...

Replication Factor = 3



DEEPER: TIMESTAMPS

- I lied earlier
- Columns are actually tuples of:
  `{ name, value, timestamp }`
- Distributed conflict resolution
- Highest timestamp value wins!
- Provided by client for every write

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

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

jim	age: 36 2011/01/01 12:35	car: camaro 2011/01/01 12:35	gender: M 2011/01/01 12:35
carol	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
johnny	age:12 2011/05/15 07:35	gender: M 2011/05/15 07:35	
suzy	age:10 2011/09/17 19:13	gender: F 2011/09/17 19:13	

Per-Column Timestamp

jim	age: 36 2011/01/01 12:35	car: camaro 2011/01/01 12:35	gender: M 2011/01/01 12:35
carol	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
johnny	age:12 2011/05/15 07:35	gender: M 2011/05/15 07:35	
suzy	age:10 2011/09/17 19:13	gender: F 2011/09/17 19:13	

jim	age: 36 2011/01/01 12:35	car: camaro 2011/01/01 12:35	gender: M 2011/01/01 12:35
carol	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
johnny	age:12 2011/05/15 07:35	gender: M 2011/05/15 07:35	
suzy	age:10 2011/09/17 19:13	gender: F 2011/09/17 19:13	

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

```
insert(“carol”,
       { “car”: “daewoo”, 2011/11/15 15:00 })
```

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

```
insert(“carol”,
       { “car”: “daewoo”, 2011/11/15 15:00 })
```

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

```
insert("carol",
       { "car": "daewoo", 2011/11/15 15:00 })
```

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

2011/11/15 15:00 > 2011/01/01 12:38

```
insert(“carol”,
       { “car”: “daewoo”, 2011/11/15 15:00 })
```

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

```
insert(“carol”,
       { “car”: “daewoo”, 2011/11/15 15:00 })
```

carol[1]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39

Cassandra replicates data automagically

DEEPER: CONSISTENCY

- I ~~lied again~~ left something out
- All ops also have a consistency level
  - Controls number of replicas involved in an operation (read or write)
  - Eventual Consistency → Full Consistency
  - Use lowest you can get away with
  - Use quorum in place of “all” consistency

WRITE CONSISTENCY LEVEL:
ONE

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

Writes to any replica and acknowledges to client, other nodes eventually get the update

WRITE CONSISTENCY LEVEL:
QUORUM

carol[1]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

Writes to majority replicas, other nodes eventually get the update

READ CONSISTENCY LEVEL:
ONE

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

Reads from first replica to respond to query, so could be inconsistent

READ CONSISTENCY LEVEL:
QUORUM

Timestamp resolves conflict, "daewoo" wins

carol[1]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39
carol[2]	age: 37 2011/01/01 12:38	car: daewoo 2011/11/15 15:00	gender: F 2011/01/01 12:39
carol[3]	age: 37 2011/01/01 12:38	car: subaru 2011/01/01 12:38	gender: F 2011/01/01 12:39

Reads from majority replicas, so if paired with quorum writes, guarantees consistency

QUICKIE LIST-AS-COLUMNS

TIMELINE COLUMN FAMILY

jim	time-uuid: “hello!”	time-uuid: “echo!”	time-uuid: “anyone!”
carol	time-uuid: “first!”	time-uuid: “second!”	

Time-sortable globally unique IDs orders columns by their insertion time

THINK IN QUERIES

- Break your normalization habit
- Roughly ~one CF per query
- Denormalize!
- Use in-column entity caching

QUESTIONS YET?

PYCASSA

CREATE A CONNECTION

```
>>> pool = pycassa.ConnectionPool("Keyspace",
                                  server_list=["192.168.1.1"])
```

Moar servers here will distribute this client's load over multiple nodes in the cluster.

PYCASSA

OPEN A COLUMN FAMILY

```
>>> cf = pycassa.ColumnFamily(pool, “People”)
```

Column Family

PYCASSA

GET A ROW

Row Key

```
>>> cf.get(“carol”)
{ “age”: “37”, “car”: “subaru”, “gender”: “F” }
```

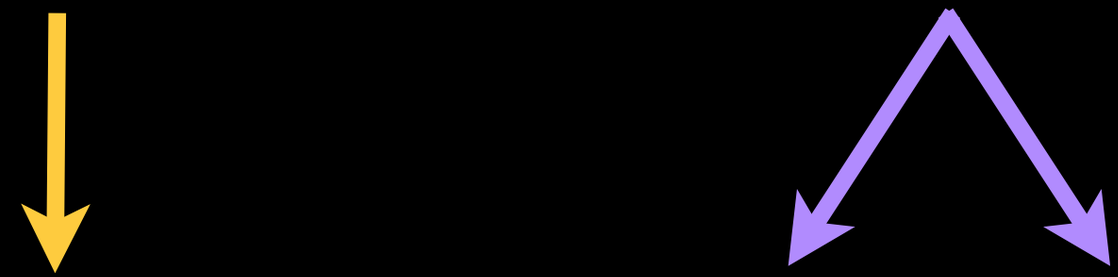
All columns returned as a dictionary

PYCASSA

GET SPECIFIC COLUMNS

Row Key

Column Names



```
>>> cf.get(“carol”, columns=[“age”, “gender”])
{ “age”: “37”, “gender”: “F” }
```

PYCASSA

GET SLICE OF COLUMNS

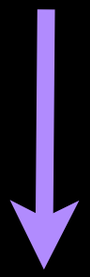
Row Key **Column Name Range**

```
>>> cf.get(“carol”, column_start=”a”, column_end=”d”)
{ “age”: “37”, “car”: “subaru” }
```

PYCASSA

“UPSERT” A COLUMN

Column Name



```
>>> cf.insert(“carol”, { “car”: “mercedes” })
```

Row Key

Column Value

PYCASSA
DELETE A ROW

```
>>> cf.remove("carol")
```

Row Key

PYCASSA
DELETE A COLUMN

Column Names

```
>>> cf.remove("carol", columns=["car"])
```

Row Key

PYCASSA
CUSTOM TIMESTAMPS

- Default is GMT timestamp
- Overridden for inserts & removes
- Per-CF generator function can be specified.

Insert beats delete!

```
>>> cf.insert("carol", { "car": "subaru" },
              timestamp=10000001)

>>> cf.remove("carol", columns=["car"],
              timestamp=10000000)
```

PYCASSA
CONSISTENCY LEVEL

- Can be overridden on all operations
- Global default read & write is ONE
- Per-CF override for default

```
>>> cf.insert(“carol”, { “car”: “subaru” },
        write_consistency_level=ConsistencyLevel.QUORUM)

>>> cf.get(“carol”,
        read_consistency_level=ConsistencyLevel.QUORUM)
```

PYCASSA

UUIDv4

```
>>> import uuid
>>> cf.insert(uuid.uuid4().bytes_le, { “foo”: “bar” })
>>> cf.insert(“joe”, { uuid.uuid4(): “bar” })
```

PYCASSA

TimeUUID (aka UUIDv1)

```
>>> import uuid
>>> timeline.insert(“joe”, { uuid.uuid1(): “hello!” })
```

DIDN'T COVER

- CQL
- Batch Operations
- Schema Definition
- SuperColumns (don't use them)
- Key Range Queries (don't use them)
- Secondary Indexes
- ... and the list goes on

SHAMELESS

- DataStax is the leading provider of Apache Cassandra training, support, and consulting services
- DataStax Enterprise is Hadoop on top of Cassandra; really nice for operations
- Seriously smart peeps, no lie
- www.datastax.com

QUESTIONS?

THANKS

http://www.datastax.com/docs/1.0/

http://www.datastax.com/dev/tutorials/

http://www.datastax.com/products/community/

http://pycassa.github.com/

http://apache.cassandra.org/

Twitter: @rbranson