Five data models for sharding

Craig Kerstiens, Head of Cloud at Citus

citusdata

Who am I?

Craig Kerstiens
http://www.craigkerstiens.com
@craigkerstiens

Postgres weekly
Run Citus Cloud

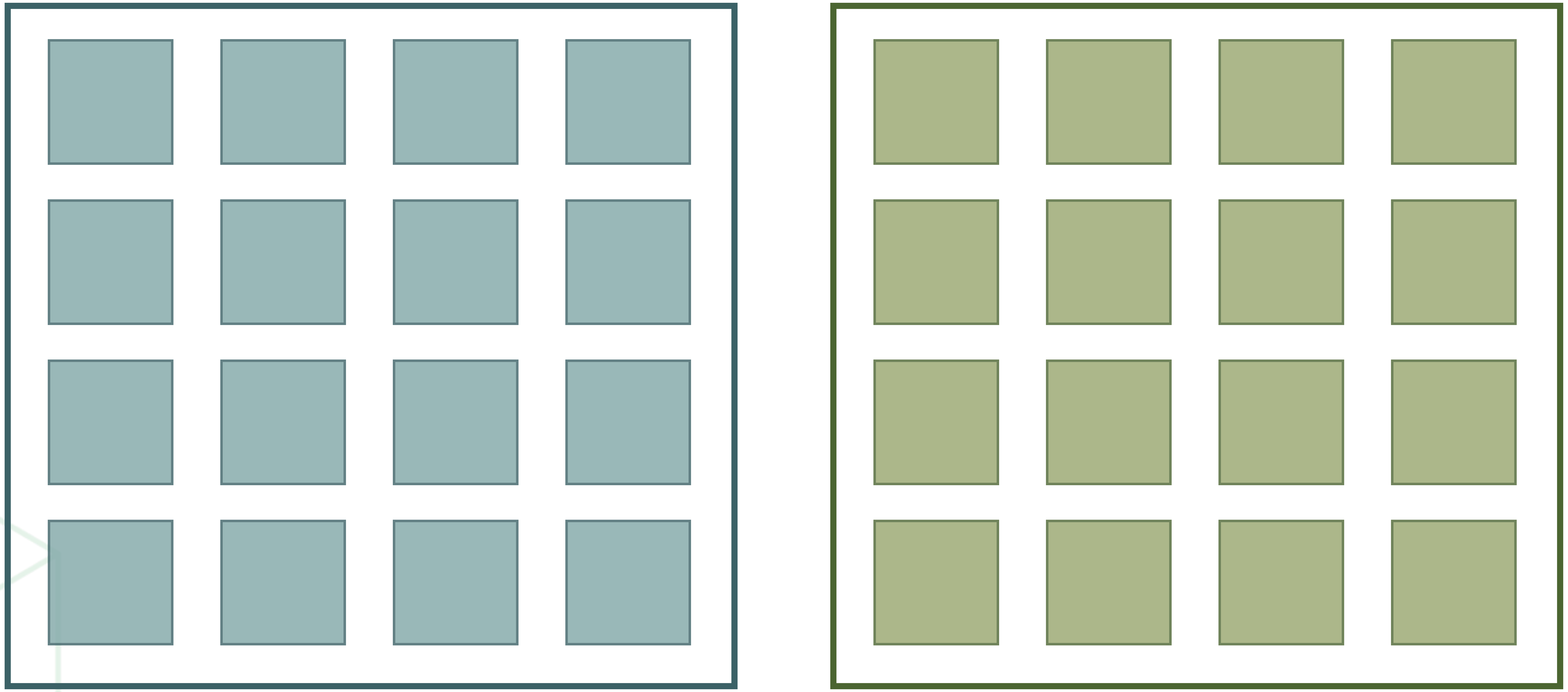
Previously Heroku,
Accenture, Truviso

What is sharding

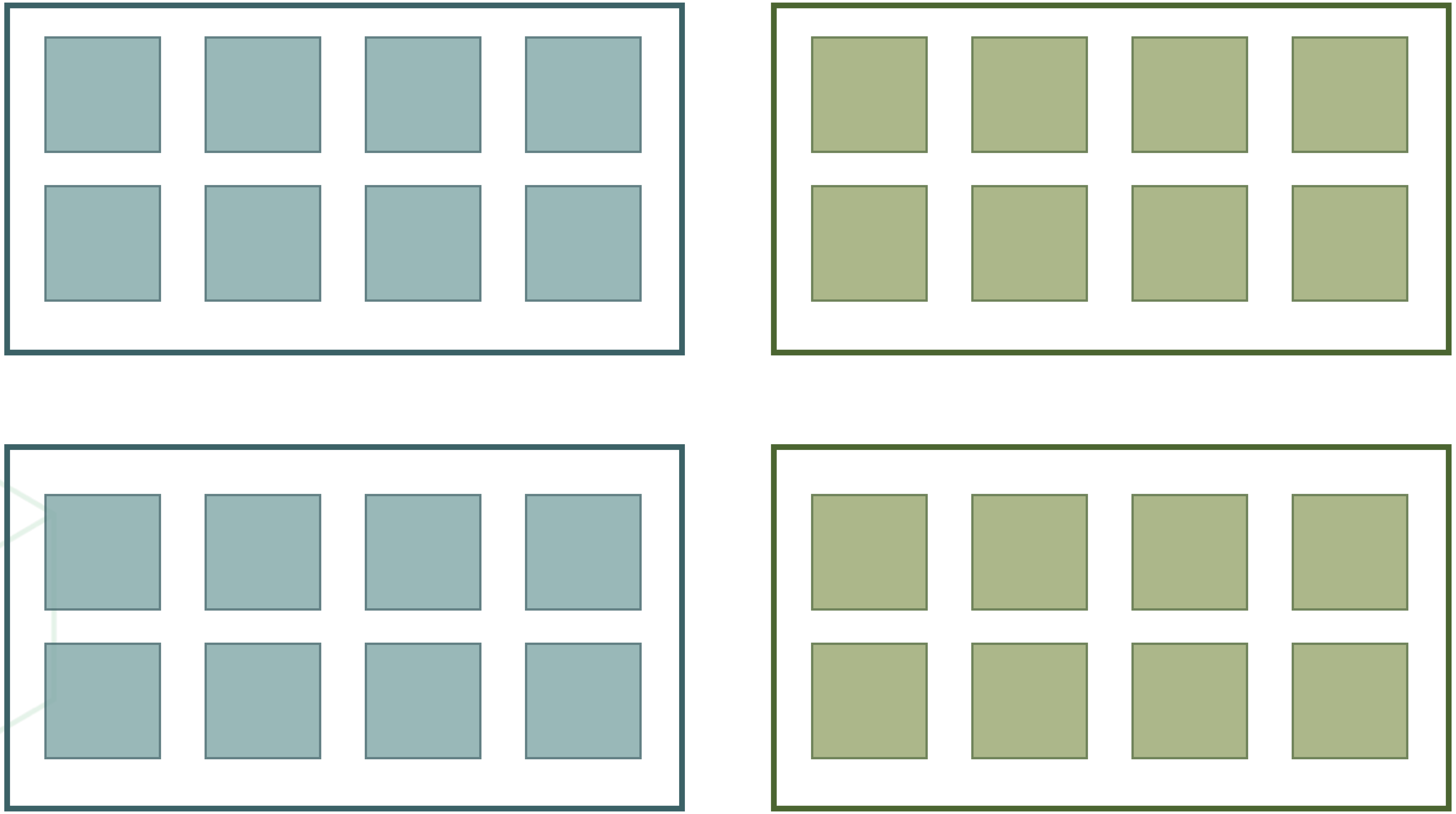
Practice of separating a large database into smaller, faster, more easily managed parts called data shards.

Source: the internet

2 Nodes - 32 shards



4 nodes - still 32 shards



Five models

- Geography
- Multi-tenant
- Entity id
- Graph model
- Time series

But first, two approaches

Hash

Range

Hash - The steps

1. Hash your id

2. Define a shard range x shards, and each contain some range of hash values. Route all inserts/updates/deletes to the shard

3. Profit

More details

- Hash based on some id
- Postgres internal hash can work fine, or so can your own
- Define your number of shards up front, make this larger than you expect to grow to in terms of nodes
  - (2 is bad)
  - (2 million is also bad)
  - Factors of 2 are nice, but not actually required

Don’t just route values

- 1-10 -> shard 1
- 2-20 -> shard 2

Create range of hash values

- hash 1 = 46154
- hash 2 = 27193
- Shard 13 = ranges 26624 to 28672

Range - The steps

1. Ensure you’ve created your new destination for your range

2. Route your range to the right bucket

3. Profit

Let’s build Google Analytics

Google analytics

- Accounts
  - Page view
    - Visitor id
    - Time
    - Page
    - Referrer
    - etc.

Hash based deeper dive

- Define your shards up front
- Remember: multiple shards within a single instance/VM

Setting things up

- Visits table
  - Hash based we're going to create visits 1-32
- CREATE TABLE visits_01 … (on node 1)
- CREATE TABLE visits_02 … (on node 2)
- CREATE TABLE visits_03 … (on node 1)
- CREATE TABLE visits_04 … (on node 2)

How’s our data look?

Account	page	occurred_at	visitor_id
1	https://www.craigkerstiens.com	6/5/2018 12:34:00	ab49e5-bc34-46d12
2	http://www.facebook.com	6/5/2018 13:10:00	ce52062-bc38-43d52

How’s our data look?

Hashed Account	page	occurred_at	visitor_id
46154	https://www.craigkerstiens.com	6/5/2018 12:34:00	ab49e5-bc34-46d12
27193	http://www.facebook.com	6/5/2018 13:10:00	ce52062-bc38-43d52

Mapping the data

Hash ranges	Table data
0-2047	visits_01
2048-4095	visits_02
…	…
26624 to 28672	visits_13
…	
63488-65536	visits_32

How’s our data look?

Hashed Account	page	occurred_at	visitor_id	Table
46154	https://www.craigkerstiens.com	6/5/2018 12:34:00	ab49e5-bc34-46d12	visits_27
27193	http://www.facebook.com	6/5/2018 13:10:00	ce52062-bc38-43d52	visits_13

How do we even query this thing?

```
SELECT *
FROM visits
WHERE account_id = 1
```

How do we even query this thing?

```
SELECT get_hash_value(1);
46154

SELECT tablename
FROM hash_buckets
WHERE 46154 > lower_range
  AND 46154 <= upper_range;
visits_13

SELECT *
FROM visits_13
WHERE account_id = 1
```

Let’s build Google Analytics (Again)

Range based

- Route based on rule based logic

- Create a new buckets

How’s our data look?

Visit ID	page	occurred_at	visitor_id
1	https://www.craigkerstiens.com	6/5/2018 12:34:00	ab49e5-bc34-46d12
2	http://www.facebook.com	6/5/2018 13:10:00	ce52062-bc38-43d52

Create new buckets as needed

- Thanks Postgres 10
- Can be minutely, hourly, daily, you choose

- CREATE TABLE visits_06_04_2018 …
- CREATE TABLE visits_06_05_2018 …
-

How do we even query this thing?

```
SELECT *
FROM visits
WHERE user_id = 1
```

How do we even query this thing?

```
SELECT *
FROM visits
WHERE user_id = 1
  AND occurred_at = '06-05-2018'
```

How do we even query this thing?

```
SELECT *
FROM visits_06_05_2018
WHERE user_id = 1
  AND occurred_at = '06-05-2018'
```

Sharding when do you need it?

You’re having trouble scaling

You’re having trouble scaling

Sharding to the rescue

SMITH

Single biggest factor is your data model

Five models

- Geography
- Multi-tenant
- Entity id
- Graph model
- Time series

geography

Shard by Geography

- Is there a clear line I can draw for a geographical boundary
    - Good examples: income by state, healthcare, etc.
- Bad examples:
    - Text messages: 256 sends to 510, both want a copy of this data…

Will geography sharding work for you?

- Do you join across geographies?
- Does data easily cross boundaries?
- Is data queries across boundaries or a different access **frequently**?

More specifics

- Granular vs. broad
- State vs. zip code
  - (California and texas are bad)
  - Zip codes might work, but does that work for your app?

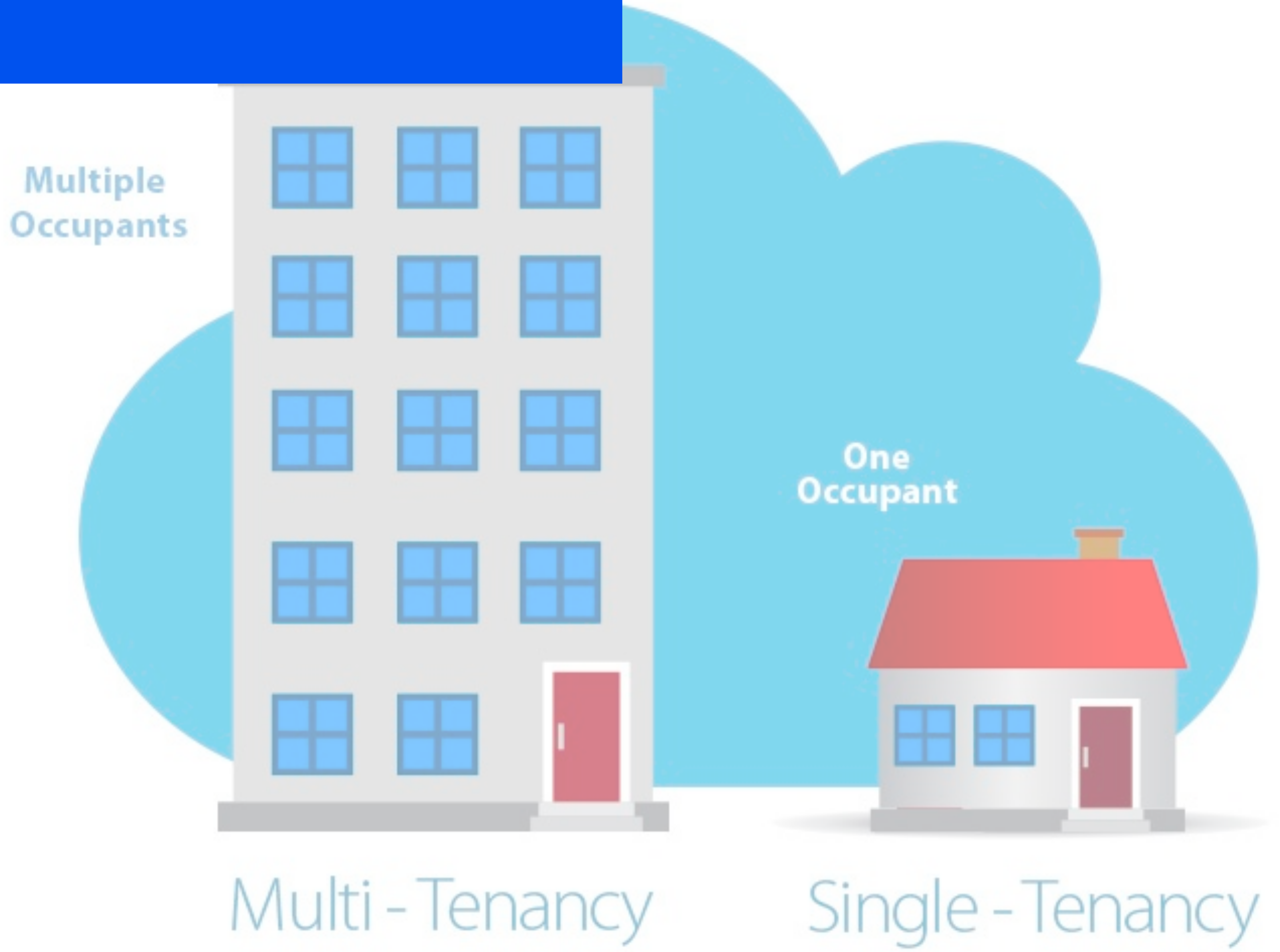
Common use cases

- If your go to market is geography focused
  - Instacart/Shipt
  - Uber/Lyft

Real world application

- Range sharding makes moving things around harder here
- Combining the geography and giving each and id, then hashing (but using smaller set of shards) can give better balance to your data

multi-tenant



Sharding by tenant

- Is each customer's data their own?
- What's your data's distribution?
    - (If one tenant/customer is 50% of your data tenant sharding won't help)
    - If it's 10% of your data you may be okay

Common use cases

- Saas/B2B
  - Salesforce
  - Marketing automation
  - Any online SaaS

Guidelines for multi-tenant sharding

- Put your tenant_id on every table that's relevant
    - Yes, denormalize
- Ensure primary keys and foreign keys are composite ones (with tenant_id)
- Enforce your tenant_id is on all queries so things are appropriately scoped

Salesforce schema

```sql
CREATE TABLE leads (
  id serial primary key,
  first_name text,
  last_name text,
  email text
);
CREATE TABLE accounts (
  id serial primary key,
  name text,
  state varchar(2),
  size int
);
CREATE TABLE opportunity (
  id serial primary key,
  name text,
  amount int
);
```

Salesforce schema - with orgs

```
CREATE TABLE leads (
  id serial primary key,
  first_name text,
  last_name text,
  email text,
  org_id int
);
CREATE TABLE accounts (
  id serial primary key,
  name text,
  state varchar(2),
  size int
  org_id int
);
CREATE TABLE opportunity (
  id serial primary key,
  name text,
  amount int
  org_id int
);
```

Salesforce schema - with orgs

```
CREATE TABLE leads (
  id serial primary key,
  first_name text,
  last_name text,
  email text,
  org_id int
);
CREATE TABLE accounts (
  id serial primary key,
  name text,
  state varchar(2),
  size int
  org_id int
);
CREATE TABLE opportunity (
  id serial primary key,
  name text,
  amount int
  org_id int
);
```

Salesforce schema - with keys

```
CREATE TABLE leads (
  id serial,
  first_name text,
  last_name text,
  email text,
  org_id int,
  primary key (org_id, id)
);
CREATE TABLE accounts (
  id serial,
  name text,
  state varchar(2),
  size int,
  org_id int,
  primary key (org_id, id)
);
CREATE TABLE opportunity (
  id serial,
  name text,
  amount int,
```

Salesforce schema - with keys

```
CREATE TABLE leads (
  id serial,
  first_name text,
  last_name text,
  email text,
  org_id int,
  primary key (org_id, id)
);
CREATE TABLE accounts (
  id serial,
  name text,
  state varchar(2),
  size int,
  org_id int,
  primary key (org_id, id)
);
CREATE TABLE opportunity (
  id serial,
  name text,
  amount int,
```

Warnings about multi-tenant implementations

- Danger ahead if using schemas on older PG versions
- Have to reinvent the wheel for even the basics
  - Schema migrations
  - Connection limits
- Think twice before using a schema or database per tenant

entity_id

Google Analytics
Home
Reporting
Customization
Admin
http://kikolani.com - http://kikolani.c...
kikolani.com
Intelligence Events
Real-Time
Audience
Overview
Active Users BETA
Cohort Analysis BETA
Demographics
Interests
Geo
Behavior
Technology
Mobile
Custom
Benchmarking
Users Flow
Acquisition
Behavior
Sessions
VS.
Select a metric
Hourly
Day
Week
Month
Sessions
600
300
May 22
May 29
Jun 5
Jun 12
New Visitor
Returning Visitor
15.4%
84.6%

May 17, 2015 - Jun 16, 2015

Sessions	Users	Pageviews	Pages / Session
12,682	10,956	19,838	1.56

Avg. Session Duration	Bounce Rate	% New Sessions
00:01:15	80.18%	84.61%

Demographics

- Language
- Country
- City

	Language	Sessions	% Sessions
1.	en-us	8,411	66.32%
2.	ru	1,002	7.90%
3.	en-gb	743	5.86%

Entity id

- What's an entity id?
- Something granular
  - Want to join where you can though
- Optimizing for parallelism and less for data in memory

Examples tell it best

- Web analytics

- Shard by visitor_id
  - Shard both sessions and views
  - Key is to co-locate things you’ll join on

Key considerations

- SQL will be more limited OR slow
- Think in terms of map reduce

Map reduce examples

- Count (*)
  - SUM of 32 smaller count (*)
- Average
  - SUM of 32 smaller SUM(foo) / SUM of 32 smaller count(*)
- Median
  - uh….

But I like medians and more

- Count distinct
  - HyperLogLog
- Ordered list approximation
  - Top-n
- Median
  - T-digest or HDR

graph model

First Last Lorem ipsum

Lorem · ipsum dolo

Lorem ipsum dolor sit amet, gravida enim in sollicitudin eleifend corporis. Ac etiam sit quam non libero, nec suspendisse nullam nonummy bibendum, aptent nunc phasellus, semper tortor ullamcorper wisi, blandit pulvinar

https:/Lorem ipsum dolor

Lorem ipsum dolor sit ame 1

Tips for New Advertisers

Lorem ipsum dolor sit amet, ut quam nulla leo dolor interdum nullam, nunc nec vitae, nato

This Week

138 Post Reach

13 People Engaged

Today's Results

Lorem ipsum dolor sit amet, ut quam nulla leo dolor interdum nullam, nunc nec vitae, natoque parturient urna vitae massa quisque nullam.

27 Lorem ipsum dolor at 7:30pm

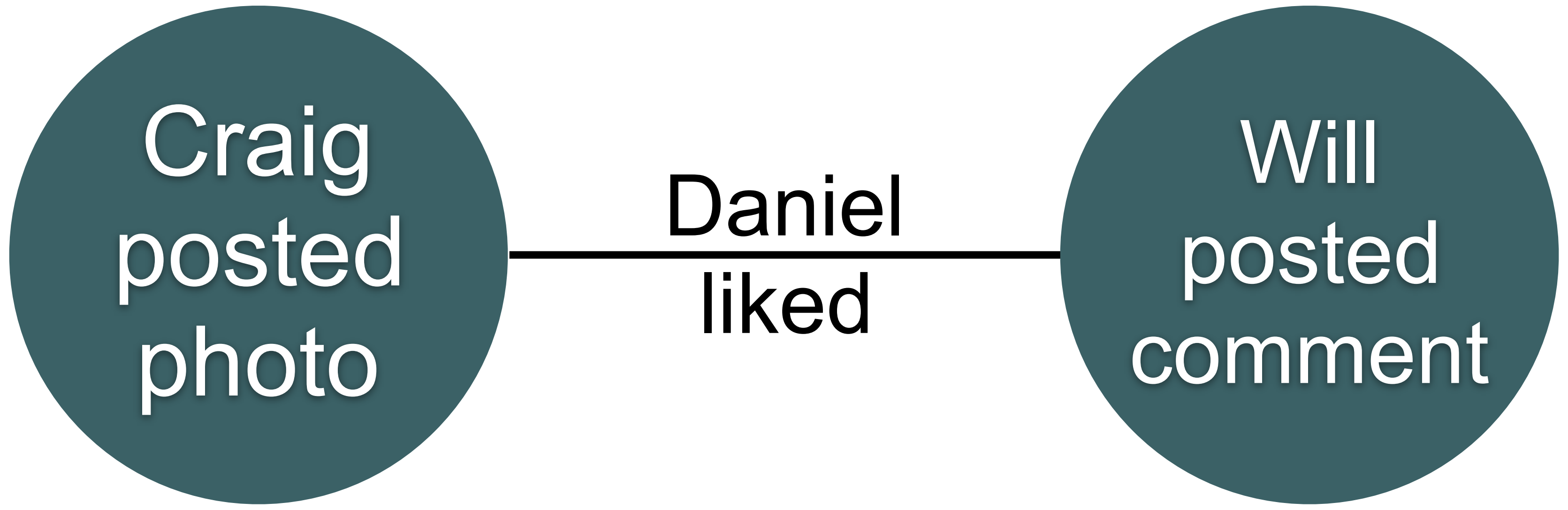
Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, ut quam nulla leo dolor interdum nullam, nunc nec vitae, natoque parturient urna vitae.

When you use a graph database

- You’ll know, really you will

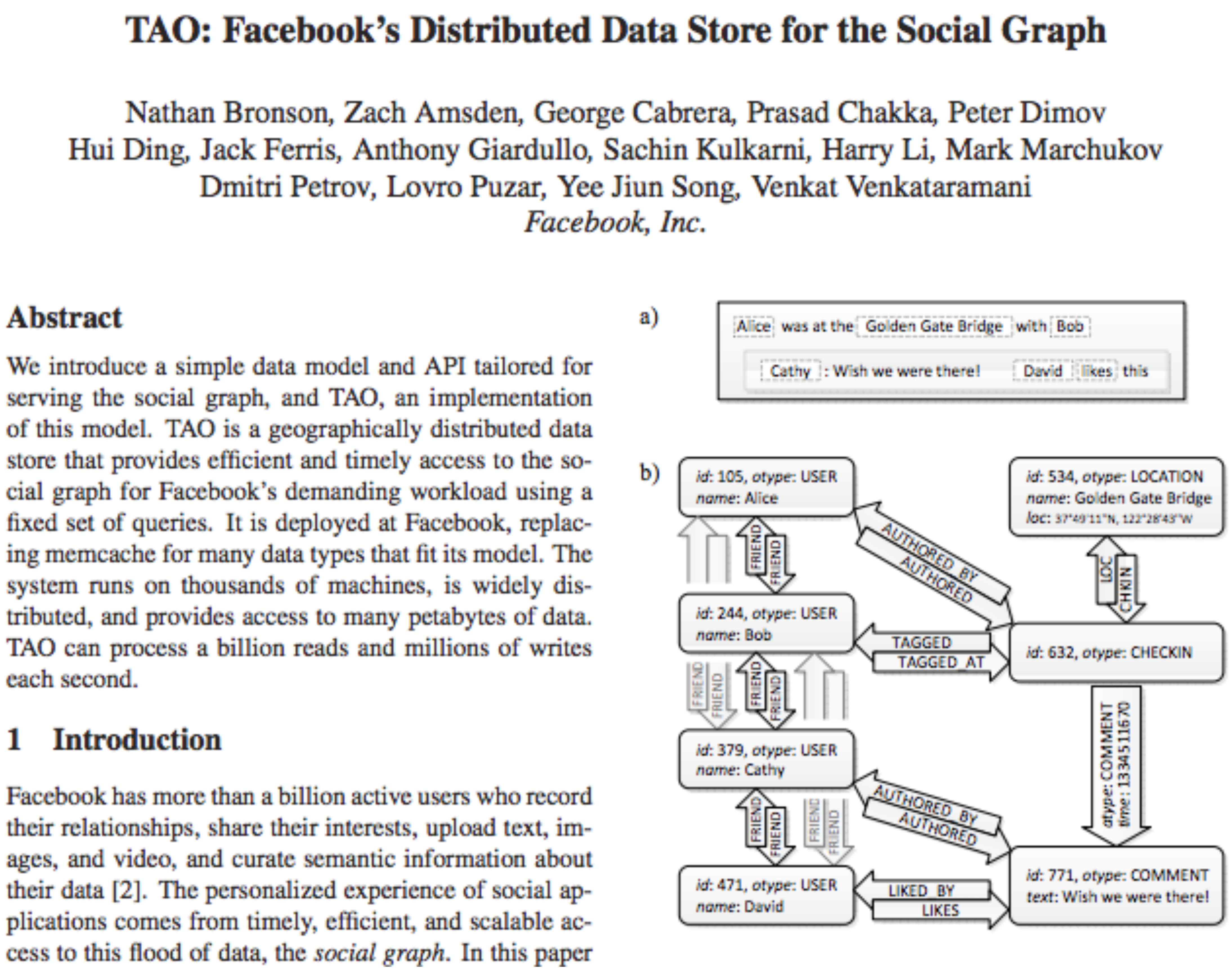
Very different approach



But what about sharding?

- Within a graph model you’re going to duplicate your data
- Shard based on both:
  - The objects themselves
  - The objects subscribed to other objects

Read this



TAO: Facebook's Distributed Data Store for the Social Graph

Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov
Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov
Dmitri Petrov, Lovro Puzar, Yee Jiun Song, Venkat Venkataramani
Facebook, Inc.

Abstract

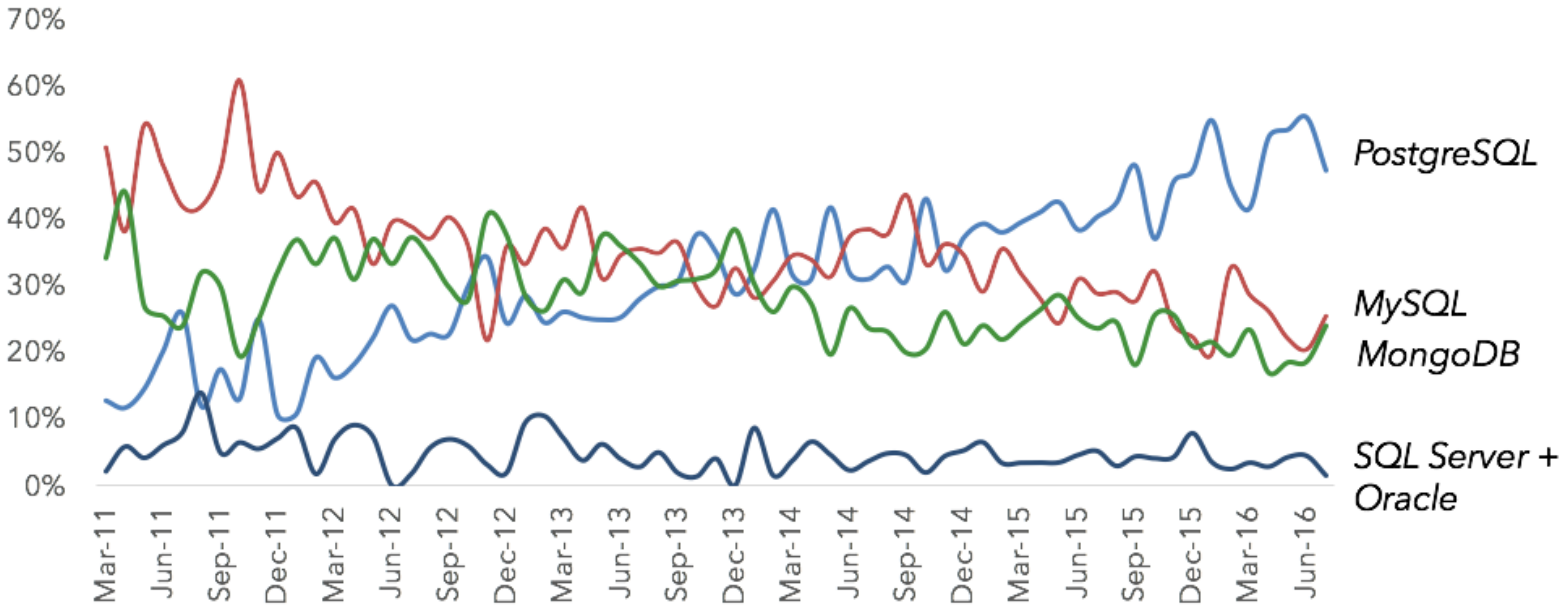
We introduce a simple data model and API tailored for serving the social graph, and TAO, an implementation of this model. TAO is a geographically distributed data store that provides efficient and timely access to the social graph for Facebook's demanding workload using a fixed set of queries. It is deployed at Facebook, replacing memcache for many data types that fit its model. The system runs on thousands of machines, is widely distributed, and provides access to many petabytes of data. TAO can process a billion reads and millions of writes each second.

1 Introduction

Facebook has more than a billion active users who record their relationships, share their interests, upload text, images, and video, and curate semantic information about their data [2]. The personalized experience of social applications comes from timely, efficient, and scalable access to this flood of data, the *social graph*. In this paper

https://www.usenix.org/system/files/conference/atc13/atc13-bronson.pdf

time series


70%
60%
50%
40%
30%
20%
10%
0%
Mar-11
Jun-11
Sep-11
Dec-11
Mar-12
Jun-12
Sep-12
Dec-12
Mar-13
Jun-13
Sep-13
Dec-13
Mar-14
Jun-14
Sep-14
Dec-14
Mar-15
Jun-15
Sep-15
Dec-15
Mar-16
Jun-16
PostgreSQL
MySQL
MongoDB
SQL Server +
Oracle

Time series: It’s obvious right?

- Well it depends

Good

Always querying time
Querying a subset
Remove old data

Okay/Bad

Querying long ranges
Not removing data

Time series

- Range partitioning
  - 2016 in a bucket, 2017 in a bucket
  - 2016-01-01 in a bucket, 2016-01-02 in a bucket…
- Key steps
  - Determine your ranges
  - Make sure you setup enough in advance, or automate creating new ones
  - Delete

Sensor data

```
CREATE TABLE measurement (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
);
```

Sensor data - initial partition

```
CREATE TABLE measurement (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
) PARTITION BY RANGE (logdate);
```

Sensor data - initial partition

```
CREATE TABLE measurement (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
) PARTITION BY RANGE (logdate);
```

Sensor data - setting up partitions

```
CREATE TABLE measurement_y2017m10 PARTITION OF measurement
    FOR VALUES FROM ('2017-10-01') TO ('2017-10-31');

CREATE TABLE measurement_y2017m11 PARTITION OF measurement
    FOR VALUES FROM ('2017-11-01') TO ('2017-11-30');
```

Sensor data - indexing

```
CREATE TABLE measurement_y2017m10 PARTITION OF measurement
    FOR VALUES FROM ('2017-10-01') TO ('2017-10-31');

CREATE TABLE measurement_y2017m11 PARTITION OF measurement
    FOR VALUES FROM ('2017-11-01') TO ('2017-11-30');

CREATE INDEX ON measurement_y2017m10 (logdate);
CREATE INDEX ON measurement_y2017m11 (logdate);
```

Sensor data - inserting

```
CREATE TRIGGER insert_measurement_trigger
    BEFORE INSERT ON measurement
    FOR EACH ROW EXECUTE PROCEDURE measurement_insert_trigger();
```

Sensor data - inserting

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF ( NEW.logdate >= DATE '2017-02-01' AND
         NEW.logdate < DATE '2017-03-01' ) THEN
        INSERT INTO measurement_y2017m02 VALUES (NEW.*);
    ELSIF ( NEW.logdate >= DATE '2017-03-01' AND
            NEW.logdate < DATE '2017-04-01' ) THEN
        INSERT INTO measurement_y2017m03 VALUES (NEW.*);
    ...
    ELSIF ( NEW.logdate >= DATE '2018-01-01' AND
            NEW.logdate < DATE '2018-02-01' ) THEN
        INSERT INTO measurement_y2018m01 VALUES (NEW.*);
    ELSE
        RAISE EXCEPTION 'Date out of range.  Fix the measurement_insert_trigger() function!';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

Time series tips

- Postgres 10 covers this pretty natively
  - Lots of rough edges

- Make sure to leverage pg_partman

Five models

- Geography
- Multi-tenant
- Entity id
- Graph model
- Time series

Recap

- Not sharding is always easier than sharding
- Identify your sharding approach/key early, denormalize it even when you’re small
- Don’t force it into one model. No model is perfect, but disqualify where you can
- Sharding used to be much more painful, it’s not quite a party yet, but it’s now become predictable based on learnings of others

citusdata

Thanks

Craig Kerstiens
@craigkerstiens

www.citusdata.com | @citusdata