Scandit
mobile product interaction

Apache Cassandra for Big Data Applications

Christof Roduner

COO and co-founder
christof@scandit.com

Java User Group Switzerland
January 7, 2014

AGENDA

- Cassandra origins and use
- How we use Cassandra
- Data model and query language
- Cluster organization
- Replication and consistency
- Practical experience

WHAT IS CASSANDRA?

WHAT IS CASSANDRA?

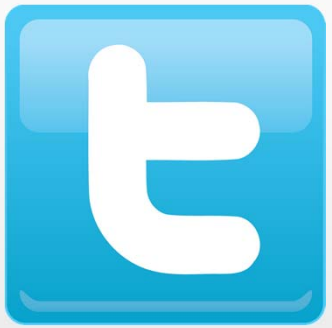
not only SQL

ORIGINS

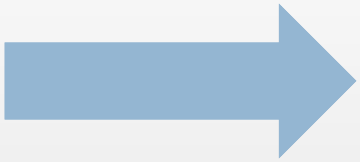
Dynamo
distributed storage

BigTable
data model

USED BY…

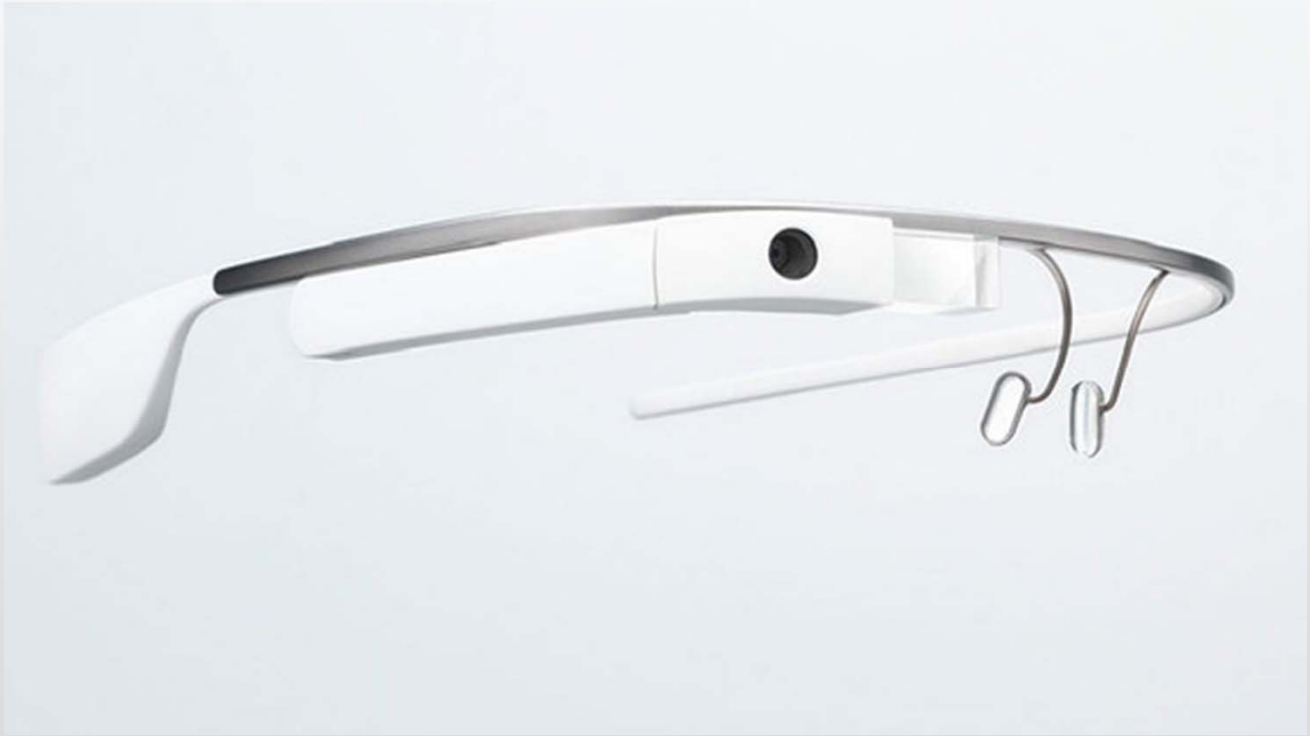


SCANDIT



- ETH Zurich startup company
- Our mission: provide the best **mobile barcode scanning** platform
- Customers: Bayer, Coop, CapitalOne, Saks 5th Avenue, Nasa, …
- Barcode scanning **SDKs** for:
  - iOS, Android
  - Phonegap
  - Titanium
  - Xamarin

SCANDIT



THE SCANALYTICS PLATFORM

Two purposes:

1. External tool for **app publishers:**
   - App-specific **real-time** usage **statistics**
   - Insights into **user behavior**
   - What do users scan?
     - Product categories? Groceries, electronics, books, cosmetics, …?
   - Where do users scan?
     - At home? Or while in a retail store?
     - Top products and brands

2. Internal tool for our **algorithms team**:
   - Improve our image processing algorithms
   - Detect devices and OS versions with **camera issues**
   - Monitor **scan performance** of our SDK

Firefox

scandit.com https://ssl.scandit.com/account/reports/6398/categories

Scandit

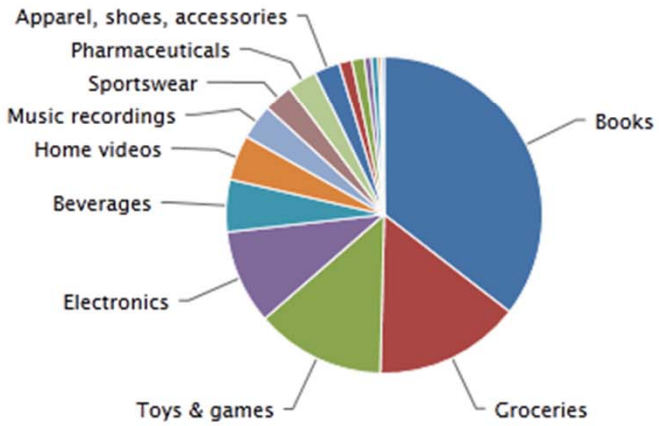
Scandit SDK Apps Documentation Company Blog My Account

You are here My Account Sign out

Product categories

Show: **day** week month

Jan 28, 2012



Your SDK

- Overview
- Download
- App key
- Get or update license

Analytics

- Number of scans
- Top products
- Products categories
- Scanning context
- Barcode types

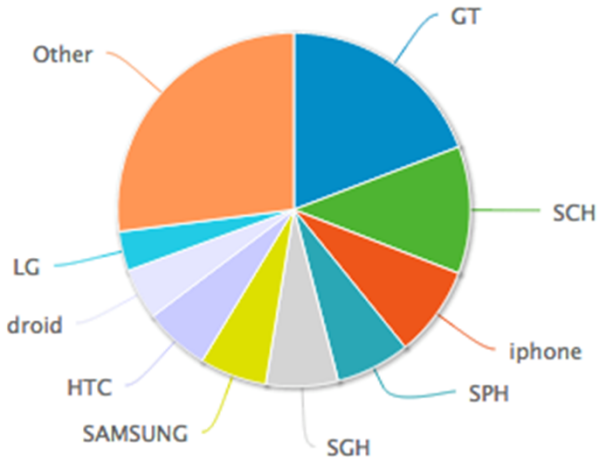
Activations by model type

1 January, 2013 to 31 December, 2014 *

- [x] Android
- [x] iOS

Show | Download CSV

* Dates are in Eastern Standard Time (UTC−05:00)



Activation details

By model | By model and version

Show 30 entries Search:

Platform	Model	Count (%)	Count
iOS	iPhone	8.3%	55122
Android	SCH-I535	4.7%	31376
Android	GT-I9300	3.9%	25909
Android	SPH-L710	3%	19905
Android	SAMSUNG-SGH-I747	2.4%	16152
Android	SCH-I545	2.3%	15285
Android	DROID RAZR	2.3%	15197
Android	GT-I9100	2.3%	15137
Android	GT-I9505	1.8%	12022

BACKEND REQUIREMENTS

- Analysis of scans
  - **Accept** and store high volumes of scans
  - Keep history of **billions** of camera parameters
  - Generate **statistics** over extended time periods
- Provide **reports** to developers

BACKEND DESIGN GOALS

- Scalability
  - High-volume storage
  - High-volume throughput
  - Support large number of concurrent client requests (mobile devices)
- Availability
- Low maintenance
  - Even as our customer base grows
- Multiple data centers

WHY DID WE CHOOSE CASSANDRA?

Partitioning

WHY DID WE CHOOSE CASSANDRA?

Simplicity

MORE REASONS…

- Looked very **fast**
  - Even when data is much larger than RAM
- Performs well in **write-heavy** environment
- Proven **scalability**
  - Without downtime
- Tunable **replication**
- **Data model**
  - YMMV…

WHAT YOU HAVE TO GIVE UP

- Joins
- Referential integrity
- Transactions
- Expressive query language (nested queries, etc.)

- Consistency (tunable, but not by default…)
- Limited support for secondary indices

HELLO CQL


```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  PRIMARY KEY (username)
);
```

HELLO CQL


```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  PRIMARY KEY (username)
);
```

```
INSERT INTO users (username, email, phone)
VALUES ('alice',
        'alice@example.com',
        '123-456-7890');

INSERT INTO users (username, email, web)
VALUES ('bob',
        'bob@example.com',
        'www.example.com');
```

HELLO CQL


```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  PRIMARY KEY (username)
);
```

```
INSERT INTO users (username, email, phone)
VALUES ('alice',
        'alice@example.com',
        '123-456-7890');

INSERT INTO users (username, email, web)
VALUES ('bob',
        'bob@example.com',
        'www.example.com');
```

```
cqlsh:demo> SELECT * FROM users;

 username | email             | phone        | web
----------+-------------------+--------------+-----------------
      bob | bob@example.com   |         null | www.example.com
    alice | alice@example.com | 123-456-7890 |            null
```

FAMILIAR… BUT DIFFERENT


```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  PRIMARY KEY (username)
);
```

No **auto increments**

(use natural key or UUID instead)

Primary key always **mandatory**

FAMILIAR… BUT DIFFERENT

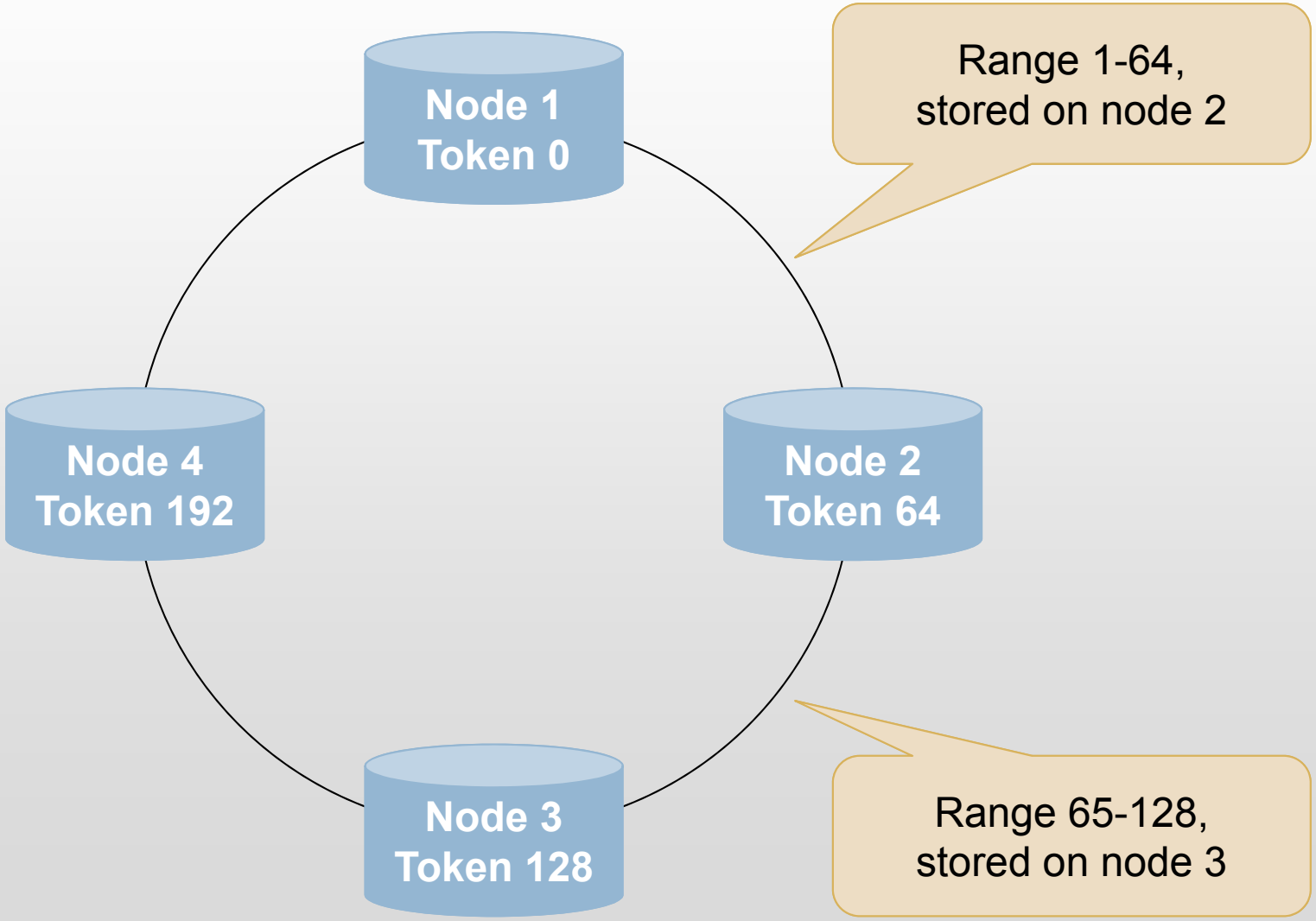

```
cqlsh:demo> SELECT * FROM users;

 username | email               | phone        | web
----------+---------------------+--------------+----------------
      bob | bob@example.com     |         null | www.example.com
    alice | alice@example.com   | 123-456-7890 |            null
```

FAMILIAR… BUT DIFFERENT


```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  PRIMARY KEY (username)
);
```


```
cqlsh:demo> SELECT * FROM users;

 username | email             | phone       | web
----------+-------------------+-------------+----------------
      bob | bob@example.com   |        null | www.example.com
    alice | alice@example.com | 123-456-7890 |            null
```

UNDER THE HOOD: CLUSTER ORGANIZATION

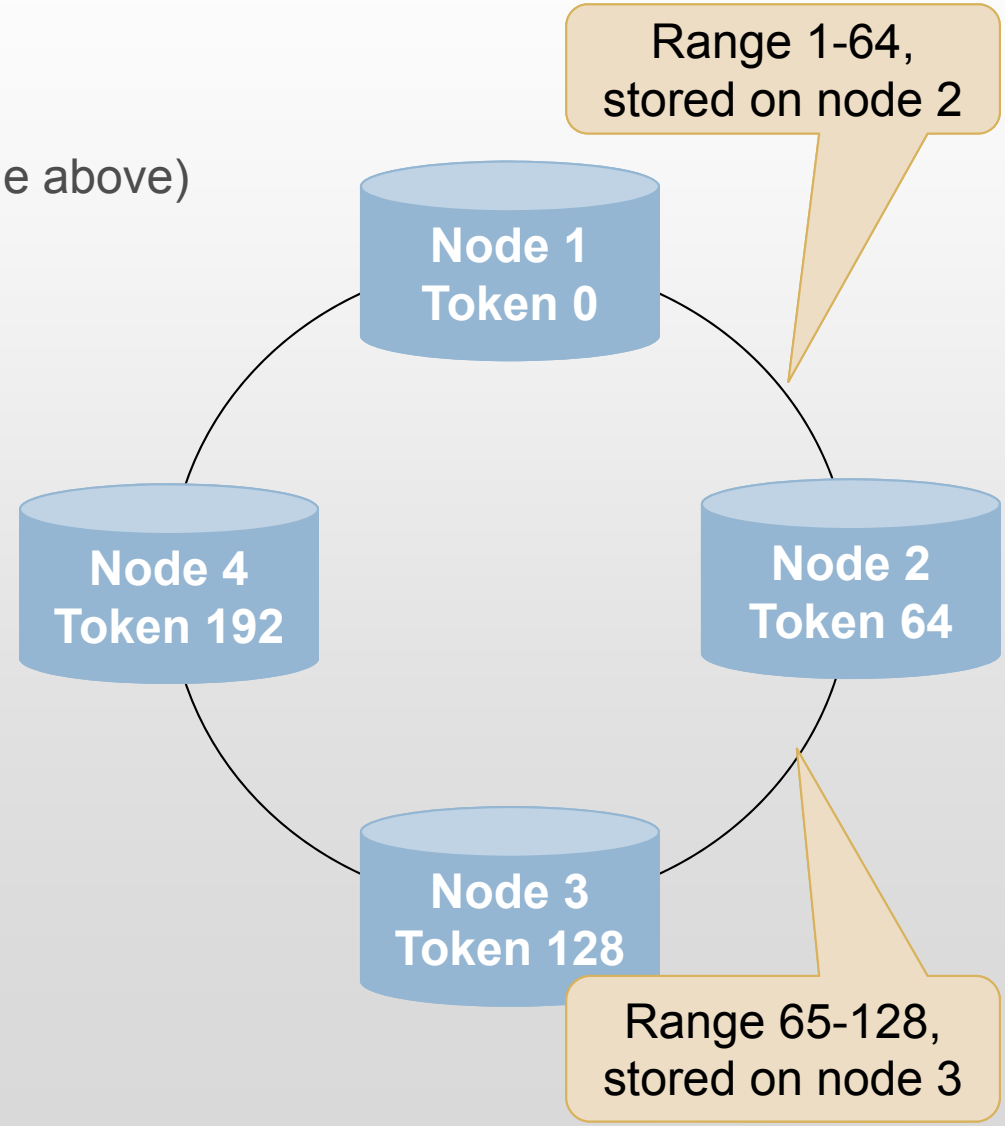


STORING A ROW

1. Calculate **md5** hash for **row key**
   (the “username” field in the example above)
   Example: md5(“alice") = 48

2. Determine data **range** for hash
   Example: 48 lies within range 1-64

3. Store row on **node** responsible for range
   Example: store on node 2



IMPLICATIONS

- Cluster **automatically balanced**
  - Load is shared equally between nodes
  - No hotspots
- **Scaling** out?
  - Easy
  - Divide data ranges by adding more nodes
  - Cluster rebalances itself automatically
- **Range queries** not possible
  - You can't retrieve «all rows from A-C»
  - Rows are not stored in their «natural» order
  - Rows are stored in order of their md5 hashes

FAMILIAR… BUT DIFFERENT

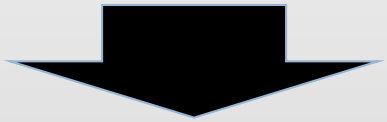

```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  PRIMARY KEY (username)
);
```

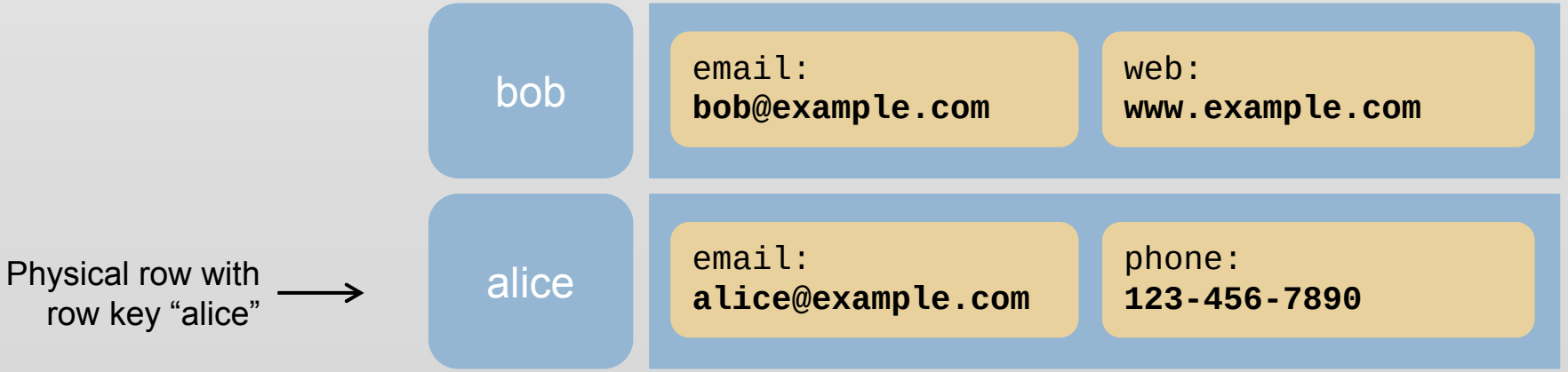

```
cqlsh:demo> SELECT * FROM users;

 username | email               | phone        | web
----------+---------------------+--------------+----------------
      bob | bob@example.com     |         null | www.example.com
    alice | alice@example.com   | 123-456-7890 |            null
```

UNDER THE HOOD: PHYSICAL STORAGE

- A physical row stores data in **name-value pairs** (“cells”)
  - Cell name is CQL field name (e.g. “email”)
  - Cell value is field data (e.g. “bob@example.com”)
- Cells in row are **automatically sorted** by name (“email” < “phone” < “web”)
- Cell names can be different in rows
- Up to **2 billion cells** per row

```
INSERT INTO users
  (username, email, phone) VALUES
  ('alice',
   'alice@example.com',
   '123-456-7890');
INSERT INTO users
  (username, email, web) VALUES
  ('bob',
   'bob@example.com',
   'www.example.com');
```





FAMILIAR… BUT DIFFERENT


```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  PRIMARY KEY (username)
);
```


```
cqlsh:demo> SELECT * FROM users;

 username | email             | phone        | web
----------+-------------------+--------------+----------------
      bob | bob@example.com   |         null | www.example.com
    alice | alice@example.com | 123-456-7890 |            null
```

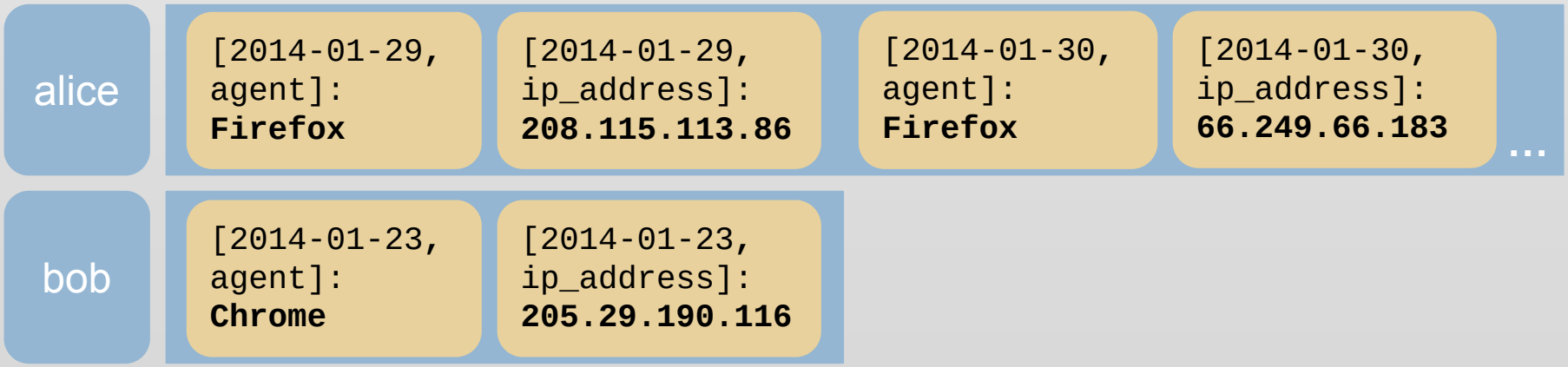
TWO BILLION CELLS


```
CREATE TABLE users (
  username TEXT,
  email TEXT,
  web TEXT,
  phone TEXT,
  address TEXT,
  spouse TEXT,
  hobbies TEXT,
  …
  hair_color TEXT,
  favorite_dish TEXT,
  pet_name TEXT,
  favorite_bands TEXT,
  …
  two_billionth_field TEXT,
  PRIMARY KEY (username)
);
```

Who needs 2 billion fields in a table?!?

2 BILLION CELLS: WIDE ROWS

- Use case: track logins of users
- Data model:
  - One (wide) physical row per user
  - User name as row key
  - Login details (time, IP address, user agent) in cells
  - Cells ordered and grouped ("**clustered**") by login timestamp
  - Cells are now tuple-value pairs
- Advantage: range queries!



2 BILLION CELLS: WIDE ROWS

- Use case: track logins of users
- Data model:
  - One (wide) physical row per user
  - User name as row key
  - Login details (time, IP address, user agent) in cells
  - Cells ordered and grouped ("**clustered**") by login timestamp
  - Cells are now tuple-value pairs
- Advantage: range queries!

```
CREATE TABLE logins (
  username TEXT,
  timestamp TIMESTAMP,
  ip_address TEXT,
  agent TEXT,
  PRIMARY KEY (username, timestamp)
);
```

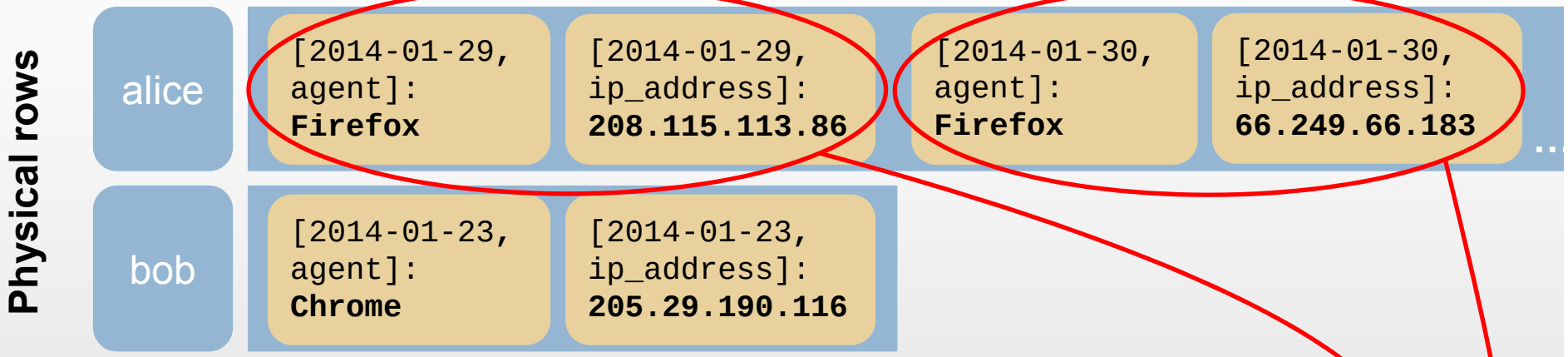
alice	[2014-01-29, agent]: **Firefox**	[2014-01-29, ip_address]: **208.115.113.86**	[2014-01-30, agent]: **Firefox**	[2014-01-30, ip_address]: **66.249.66.183**	...
bob	[2014-01-23, agent]: **Chrome**	[2014-01-23, ip_address]: **205.29.190.116**			

QUERYING THE LOGINS


```
INSERT INTO logins (username, timestamp, ip_address, agent) VALUES
  ('alice', '2014-01-29 16:22:30 +0100', '208.115.113.86', 'Firefox');
```

```
cqlsh:demo> SELECT * FROM logins;

 username | timestamp                 | agent   | ip_address
----------+---------------------------+---------+-----------------
      bob | 2014-01-23 01:12:49+0100 |  Chrome |  205.29.190.116
    alice | 2014-01-29 16:22:30+0100 | Firefox |  208.115.113.86
    alice | 2014-01-30 07:48:03+0100 | Firefox |   66.249.66.183
    alice | 2014-01-30 18:06:55+0100 | Firefox |  208.115.111.70
    alice | 2014-01-31 12:37:26+0100 | Firefox |   66.249.66.183
```

ONE CQL ROW FOR EACH CELL CLUSTER



CQL rows

```
cqlsh:demo> SELECT * FROM logins;

 username | timestamp                       | agent   | ip_address
----------+---------------------------------+---------+-----------------
      bob | 2014-01-23 01:12:49+0100 |  Chrome |  205.29.190.116
    alice | 2014-01-29 16:22:30+0100 | Firefox |  208.115.113.86
    alice | 2014-01-30 07:48:03+0100 | Firefox |   66.249.66.183
    alice | 2014-01-30 18:06:55+0100 | Firefox |  208.115.111.70
    alice | 2014-01-31 12:37:26+0100 | Firefox |   66.249.66.183
```

RANGE QUERIES REVISITED

- **Range queries** involving “timestamp” field **are possible** (because cells are ordered by timestamp):

```
cqlsh:demo> SELECT * FROM logins WHERE username = 'bob' AND timestamp >
'2014-01-01' AND timestamp < '2014-01-31';

 username | timestamp                       | agent   | ip_address
----------+---------------------------------+---------+----------------
      bob | 2014-01-23 01:12:49+0100 | Chrome | 205.29.190.116
```

- But you still have to provide a **row key**:

```
cqlsh:demo> SELECT * FROM logins WHERE timestamp > '2014-01-01' AND
timestamp < '2014-01-31';
Bad Request: Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want to
execute this query despite the performance unpredictability, use ALLOW
FILTERING
```

SECONDARY INDICES

Queries involving a non-indexed field are not possible:

```
cqlsh:demo> SELECT * FROM users WHERE email = 'bob@example.com';
Bad Request: No indexed columns present in by-columns clause with
Equal operator
```

Secondary indices can be defined for (single) fields:

```
CREATE INDEX email_key ON users (email);
SELECT * FROM users WHERE email = 'alice@example.com';
```

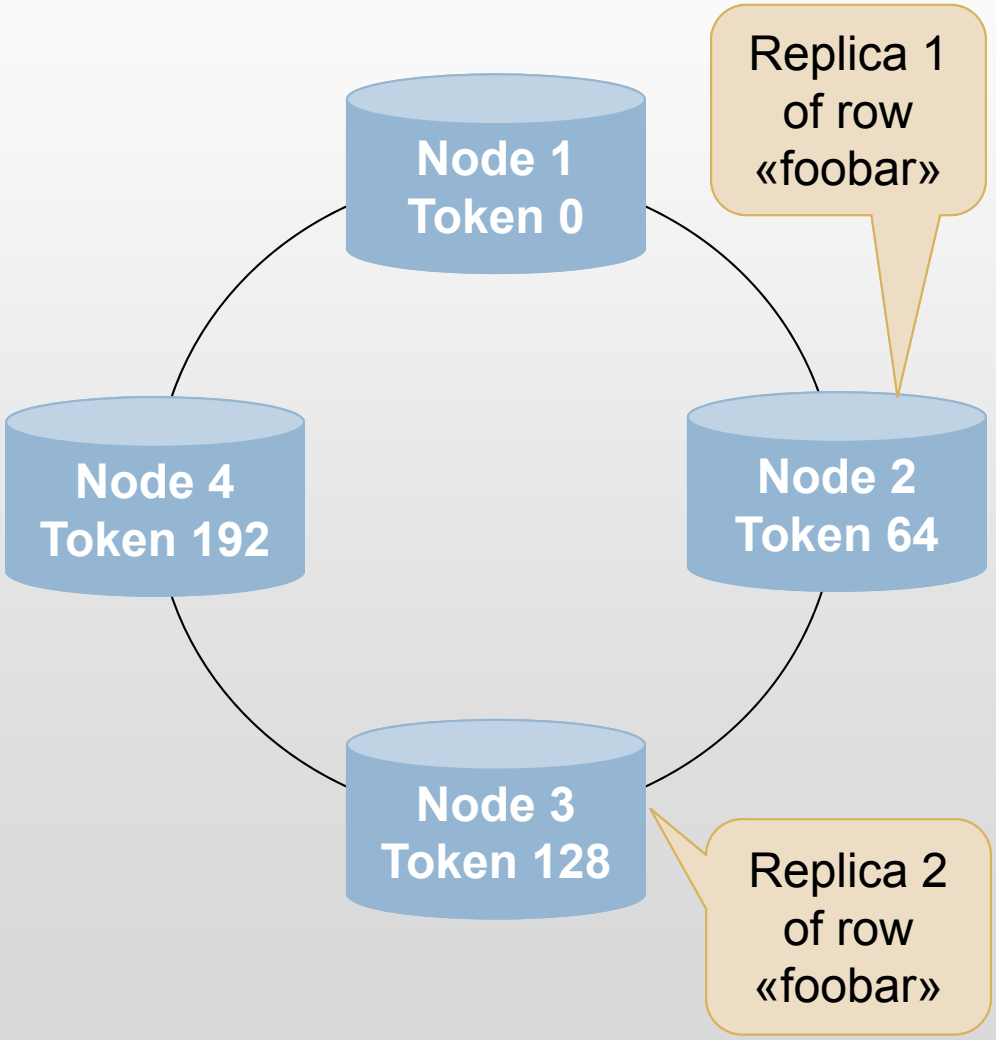
SECONDARY INDICES

- Secondary indices **only support equality** predicate (=) in queries

- Each node maintains index for data it owns
  - Request must be **forwarded to all nodes**
  - Sometimes not the most efficient approach
  - Often better to **denormalize and manually maintain** your own index

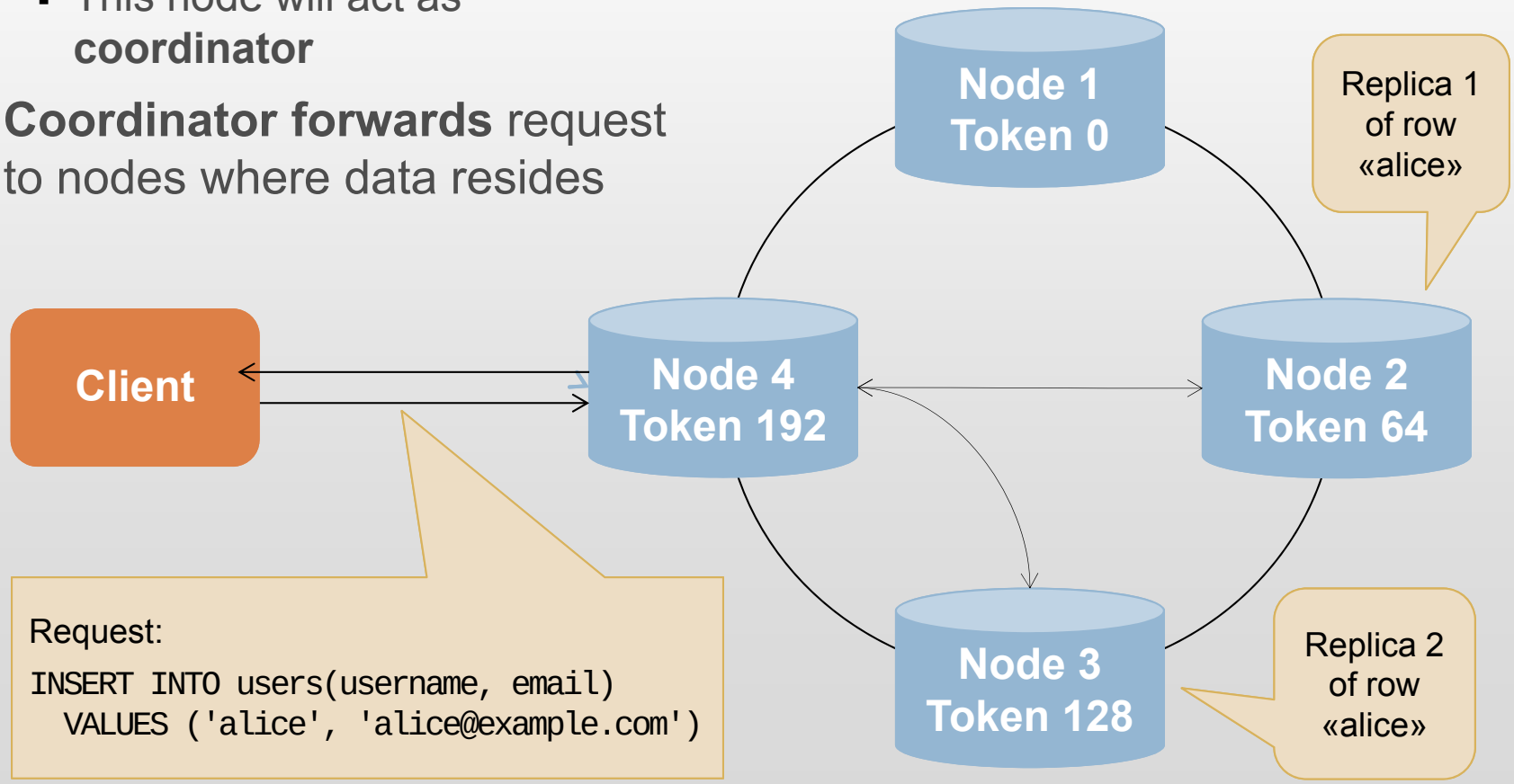
REPLICATION

- Tunable **replication factor** (RF)
- RF > 1: rows are automatically replicated to next RF-1 nodes
- Tunable replication **strategy**
  - «Ensure two replicas in different **data centers**, racks, etc.»



CLIENT ACCESS

- Clients can send read and write requests **to any node**
  - This node will act as **coordinator**
- **Coordinator forwards** request to nodes where data resides



CONSISTENCY LEVELS

- Cassandra offers **tunable consistency**
  - For all requests, clients can set a **consistency level** (CL)
- For **writes**:
  - CL defines how many replicas must be written **before «success» is returned** to client
- For **reads**:
  - CL defines how many replicas must respond **before a result is returned** to client
- Consistency levels:
  - ONE
  - QUORUM
  - ALL
  - … (data center-aware levels)

INCONSISTENT DATA

- Example scenario:
  - Replication factor 2
  - Two existing replica for row «foobar»
  - Client **overwrites existing data** in «foobar»
  - Replica 2 is down
- What happens:
  - Cells are **updated in replica 1, but not replica 2** (even with CL=ALL !)
- **Timestamps** to the rescue
  - Every cell has a timestamp
  - Timestamps are **supplied by clients**
  - Upon read, the cell with the **latest timestamp wins**
- →Use NTP

PREVENTING INCONSISTENCIES

- Read repair
- Hinted handoff
- Anti entropy

EXPIRING DATA

- Data will be deleted automatically after a given amount of time

```
INSERT INTO users (username, email, phone)
VALUES ('alice',
        'alice@example.com',
        '123-456-7890')
USING TTL 86400;
```

DISTRIBUTED COUNTERS

- Useful for analytics applications
- Atomic increment operation

```
UPDATE counters SET access = access + 1
    WHERE url = 'http://www.example.com/foo/bar'
```

PRODUCTION EXPERIENCE: CLUSTER AT SCANDIT

- We've had Cassandra in production use for almost **4 years**
- Nodes in **three data centers**
- **Linux** machines
- **Identical setup** on every node
  - Allows for easy **failover**

PRODUCTION EXPERIENCE

- Mature, no stability issues
- Very fast
- Language bindings don't always have the same quality
  - Sometimes out of sync with server, buggy
- Data model is a mental twist
- Design-time decisions sometimes hard to change
- No support for geospatial data

TRYING OUT CASSANDRA

- Set up a single-node cluster

- Install binary:
  - Debian, Ubuntu, RHEL, CentOS packages
  - Windows 7 MSI installer
  - Mac OS X (tarball)
  - Amazon Machine Image

DOCUMENTATION

- DataStax website
  - Company founded by Cassandra developers
- Apache website
- Mailing lists

THANK YOU!

Questions?

(By the way, we're hiring… ☺)