Making the Most of Cassandra

Paul O'Fallon

@paulofallon

Making the Most of Cassandra

Indexes | Batches | Transactions

Secondary Indexes

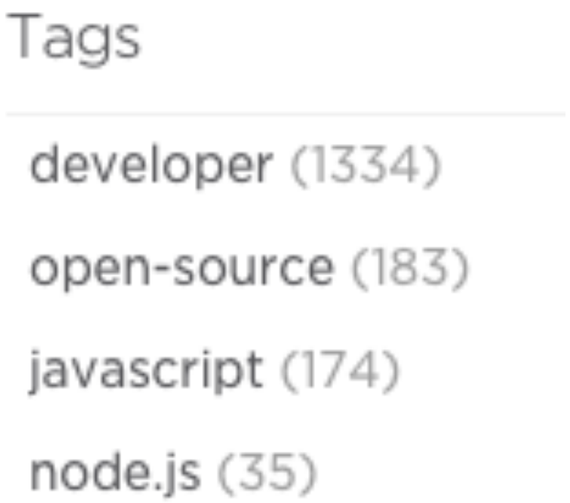
```
CREATE TABLE users (
    id varchar,
    first_name varchar,
    last_name varchar,
    company varchar,
    // ...
    PRIMARY KEY (id)
);

CREATE INDEX users_company ON users(company);
```

Secondary Indexes

```
INSERT INTO users (id, first_name, last_name, company)
VALUES ('john-doe','John','Doe','acme-corp');
```

```
SELECT * FROM users WHERE company = 'acme-corp';
```

Secondary Indexes on Collections




```
CREATE TABLE users (
     id varchar,
     first_name varchar,
     last_name varchar,
     company varchar,
     tags set<varchar>,
     // ...
     PRIMARY KEY (id)
);

CREATE INDEX ON users(tags);
```

Secondary Indexes on Collections

```
INSERT INTO users (id, first_name, last_name, company, tags)
VALUES ('john-doe','John','Doe','acme-corp',{'java'});
```

```
SELECT * FROM users WHERE tags CONTAINS 'java';
```

Secondary Indexes on Collections

✓Set
✓List
✓Map

```
CREATE INDEX ON <table>(<collection_column>)
WHERE <collection_column> CONTAINS <value>
```

✓Map

```
CREATE INDEX ON <table>(KEYS(<map_column>))
WHERE <map_column> CONTAINS KEY <key>
```

When Not to Use a Secondary Index

- High (or very low) cardinality columns
- Tables with counter columns
- Frequently updated or deleted columns
- Tables with very large partitions
- Static columns (for now)

Manually Maintained Indexes

Tags

developer (1334)

open-source (183)

javascript (174)

node.js (35)

```
CREATE TABLE courses (
    id varchar,
    // ...
    tags set<varchar> static,
    module_id int,
    // ...
    PRIMARY KEY (id, module_id)
);
```

Manually Maintained Indexes

```
CREATE TABLE course_tags (
   tag varchar,
   course_id varchar,
   PRIMARY KEY (tag, course_id)
);
```

Manually Maintained Indexes

```
INSERT INTO courses (id, name, tags)
VALUES ('node-intro','Introduction to Node.js',
{'developer','open-source','javascript','node.js'});

INSERT INTO course_tags (tag, course_id)
VALUES ('developer','node-intro');

INSERT INTO course_tags (tag, course_id)
VALUES ('open-source','node-intro');

// ... etc.
```

Batches

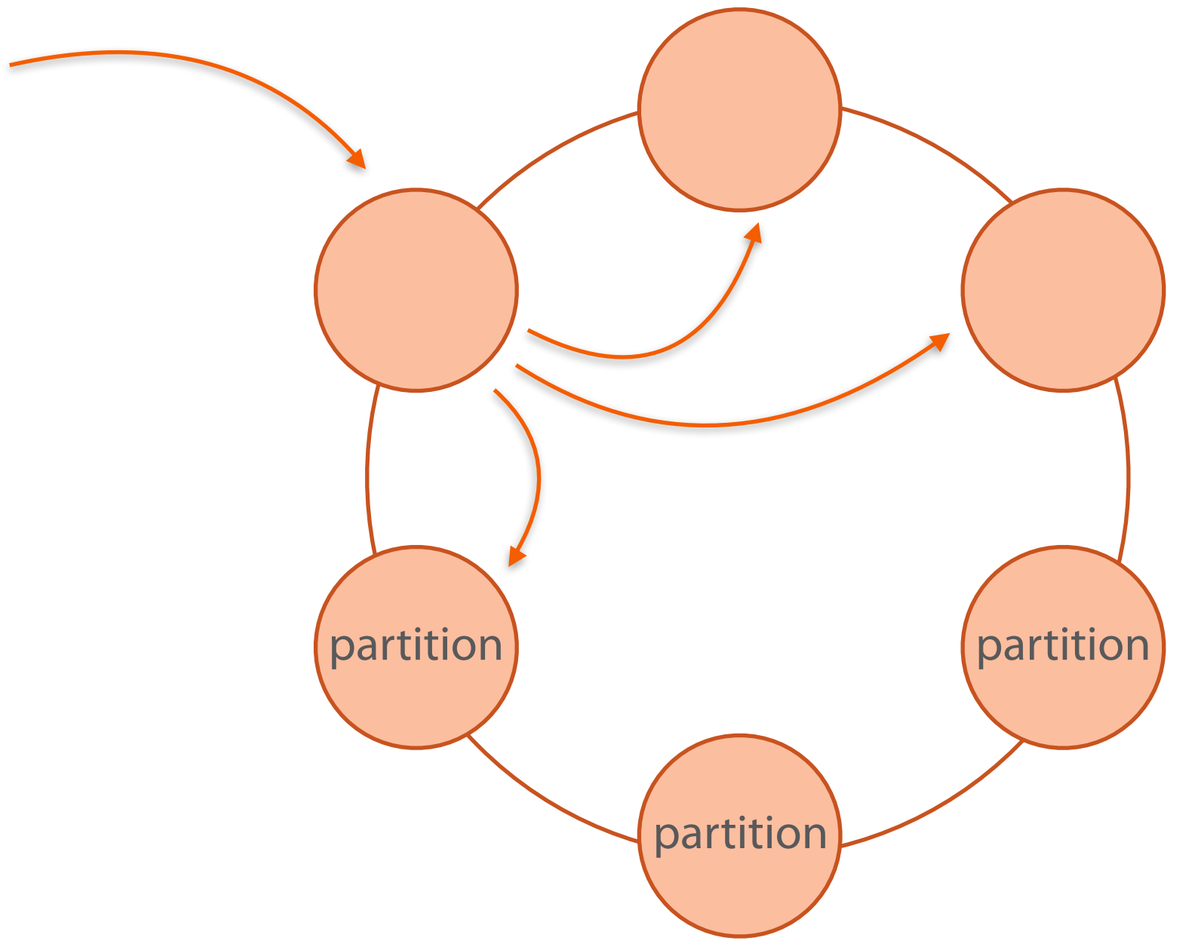
Intended for:

Keeping tables in sync

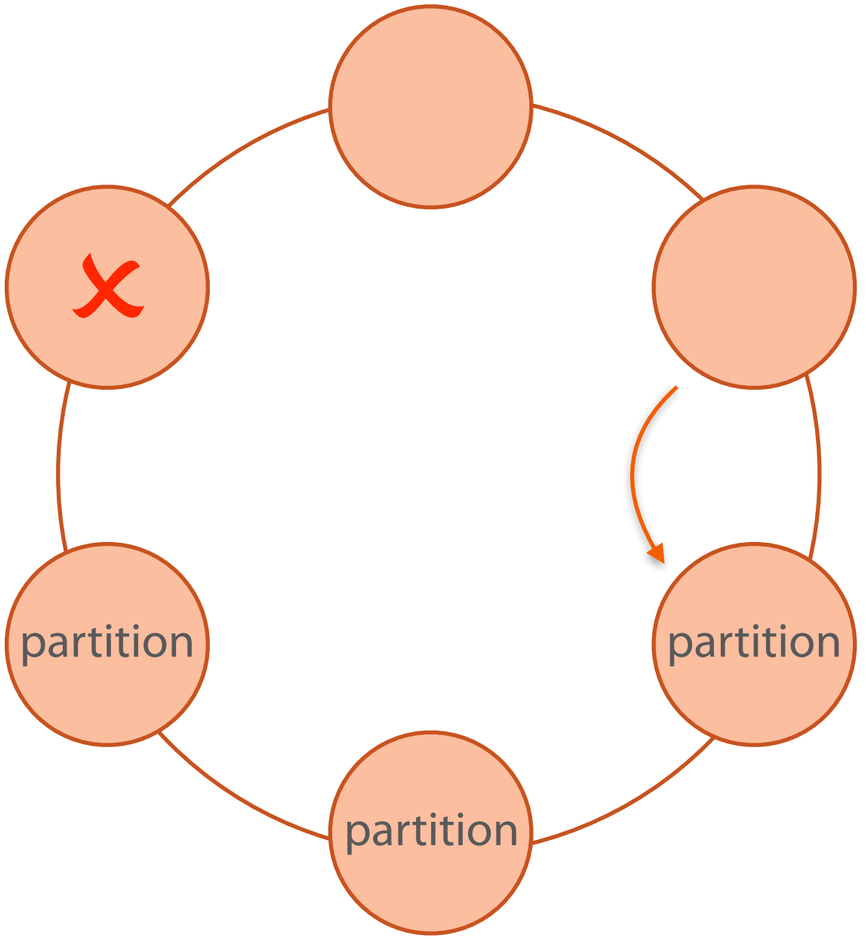
Not intended for:

Fast loading of data

Batches



Batches

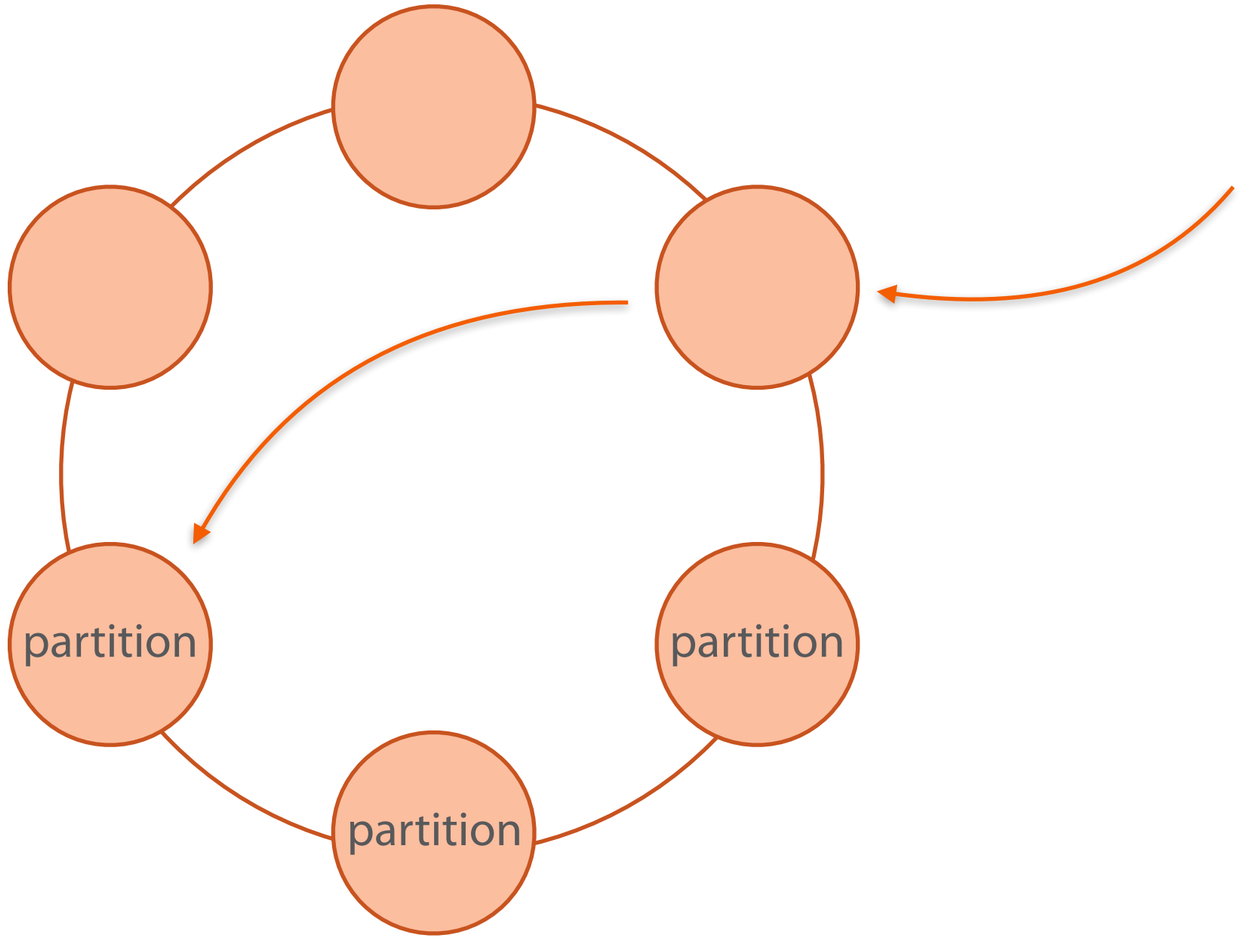


Batches

```
BEGIN BATCH

INSERT INTO courses (id, tags)
VALUES ('node-intro', {'developer', 'javascript', 'node.js',
'open-source'});

INSERT INTO course_tags (tag, course_id, course_name)
VALUES ('developer','node-intro', 'Introduction to Node.js');

// ... etc.

APPLY BATCH;
```

Unlogged Batches

ONE WRITE!



Unlogged Batches

```
BEGIN UNLOGGED BATCH

INSERT INTO courses (id, name)
VALUES ('node-intro', 'Introduction to Node.js');

INSERT INTO courses (id, module_id, module_name)
VALUES ('node-intro', 1, 'Getting Started with Node.js');

// ... etc.

APPLY BATCH;
```

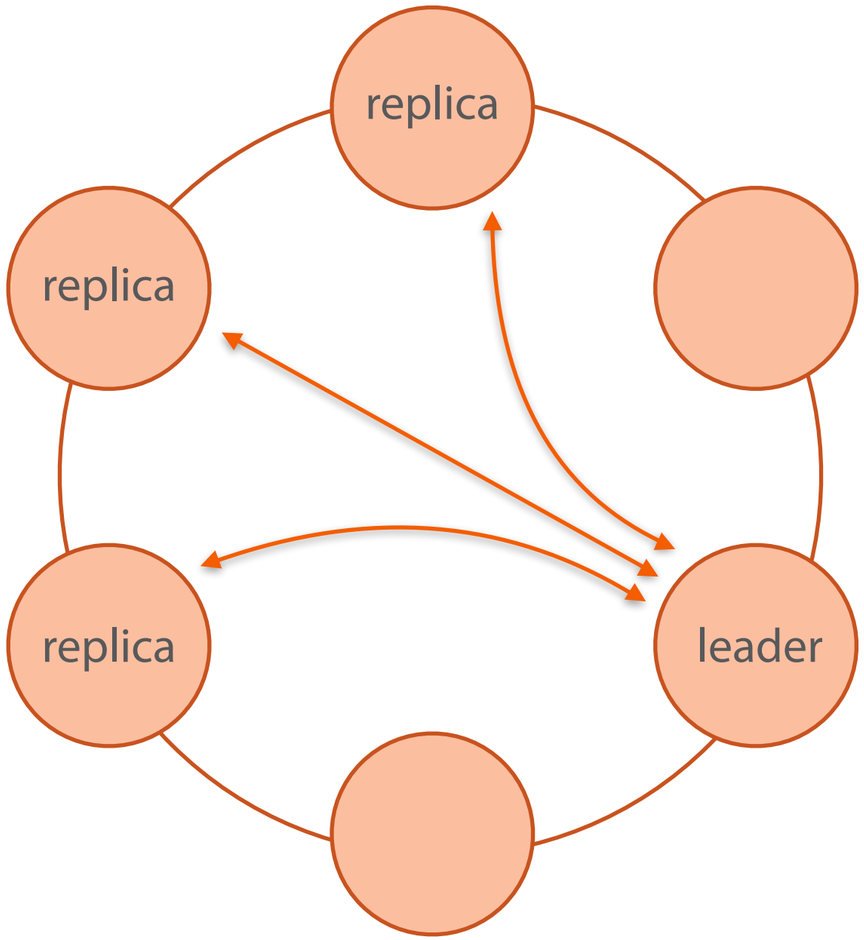
Alternative to Batches

Client pseudocode

```
PREPARE <insert_statement>
LOOP_OVER_DATA
  EXECUTE(<prepared_statement>,<row_of_data>)
END_LOOP
```

Let the client optimize for you!

Lightweight Transactions



1. Prepare ↔ Promise
2. Read ↔ Results
3. Propose ↔ Accept
4. Commit ↔ Ack

Compare-and-Set Operations

Insert

```
INSERT INTO users (id, first_name, last_name)
VALUES ('john-doe', 'John','Doe')
IF NOT EXISTS;
```

Update

```
UPDATE users SET password = 'mypass', reset_token = null
WHERE id = 'john-doe'
IF reset_token = '1GRhEs1';
```

Works with Batches Too!

```
BEGIN BATCH

  INSERT INTO <table> ... IF NOT EXISTS;
  INSERT INTO <table> ...
  INSERT INTO <table> ...

APPLY BATCH;
```

Conclusion

- Secondary Indexes
- Indexed Collections
- Manually Maintained Indexes
- Batches (Logged & Unlogged)
- Lightweight Transactions