

Multi-row Partitions



Paul O'Fallon

@paulofallon

Multi-row Partitions

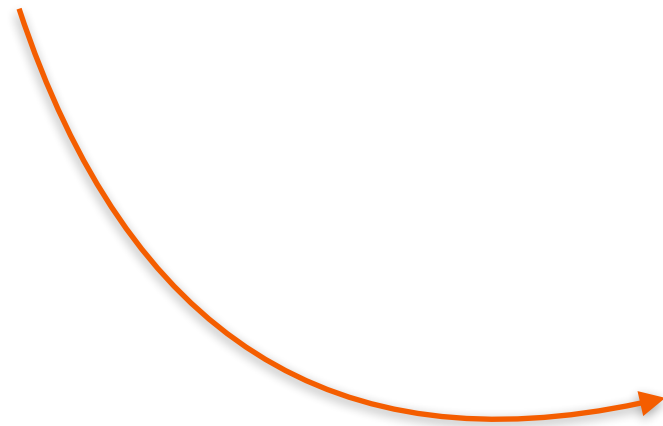
Composite Keys

Static Columns

Time Series Data

So Far...

```
CREATE TABLE courses (  
    id varchar PRIMARY KEY  
);
```



```
CREATE TABLE courses (  
    id varchar,  
    PRIMARY KEY (id)  
);
```

Keys, Keys and More Keys!

~~ONE ROW
PER PARTITION~~

PRIMARY KEY (*id*)

PRIMARY KEY (*partition_key*)

PRIMARY KEY (*partition_key, clustering_key*)

composite key

Composite Primary Keys

PRIMARY KEY (*p_key*, *c_key*)

PRIMARY KEY (*p_key*, *c_key*₁, ... *c_key*_N)

Composite Primary Keys

PRIMARY KEY (p_key, c_key)

PRIMARY KEY ($p_key, c_key_1, \dots c_key_N$)

PRIMARY KEY ($(p_key_1, \dots p_key_N), c_key_1, \dots c_key_N$)

Composite Primary Keys

PRIMARY KEY (p_key, c_key)

PRIMARY KEY ($p_key, c_key_1, \dots c_key_N$)

PRIMARY KEY (($p_key_1, \dots p_key_N$) , $c_key_1, \dots c_key_N$)

PRIMARY KEY (($p_key_1, \dots p_key_N$))








Courses, Now with Modules!

Introduction to Node.js

In this course we provide an overview of Node.js, including writing asynchronous code with callbacks and streams, and modularizing your application with NPM and require(). We also look at built-in API's for building and scaling web applications as well as a few key third party modules.

by **Paul O'Fallon**

Table of contents [Expand all](#)

	▶ Getting Started with Node.js	36:14
	▶ Modules, require() and NPM	17:43
	▶ Events and Streams	26:35
	▶ Accessing the Local System	17:20
	▶ Interacting with the Web	21:40
	▶ Testing and Debugging	27:38
	▶ Scaling Your Node Application	20:57

Course content

 **Table of contents**

 Description

 Exercise files

 Assessment

 Discussion

More info

Level **Intermediate**

Rating 

Duration **2h 48m**

Revisiting Our Courses Schema

```
CREATE TABLE courses (  
    id varchar,  
    name varchar,  
    PRIMARY KEY (id)  
);
```

```
CREATE TABLE courses (  
    id varchar,  
    name varchar,  
    module_id int,  
    module_name varchar,  
    PRIMARY KEY (id, module_id)  
);
```

Revisiting Our Courses Schema

Inserting

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

```
INSERT INTO courses (id, name, module_id, module_name)
VALUES (
    'node-intro', 'Introduction to Node.js', 2,
    'Node.js Background');
```

Revisiting Our Courses Schema

Selecting

SELECT * **FROM** courses **WHERE** id = 'node-intro'; ← two rows

SELECT * **FROM** courses
WHERE id = 'node-intro' **AND** module_id = 1; ← one row

SELECT * **FROM** courses **WHERE** id = 'node-intro'
ORDER BY module_id **DESC**; ← two rows

Ordering By Clustering Key

id='node-intro'	module_id=1	
	name	Introduction to Node.js
	module_name	Getting Started with Node.js
	module_id=2	
	name	Introduction to Node.js
	module_name	Node.js Background

Static Columns

```
CREATE TABLE courses (  
    id varchar,  
    name varchar STATIC,  
    module_id int,  
    module_name varchar,  
    PRIMARY KEY (id, module_id)  
);
```

Without Static Columns

id='node-intro'

module_id=1

name	Introduction to Node.js
module_name	Getting Started with Node.js

module_id=2

name	Introduction to Node.js
module_name	Node.js Background

Static Columns

id='node-intro'	name		Introduction to Node.js
	module_id=1		
	module_name		Getting Started with Node.js
	module_id=2		
	module_name		Node.js Background

Adding Data with Static Columns

```
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');
```

```
UPDATE courses
SET module_name='Node.js Background'
WHERE id='node-intro' AND module_id=2;
```

**SELECT
REMAINS
THE SAME**

Time Series Data



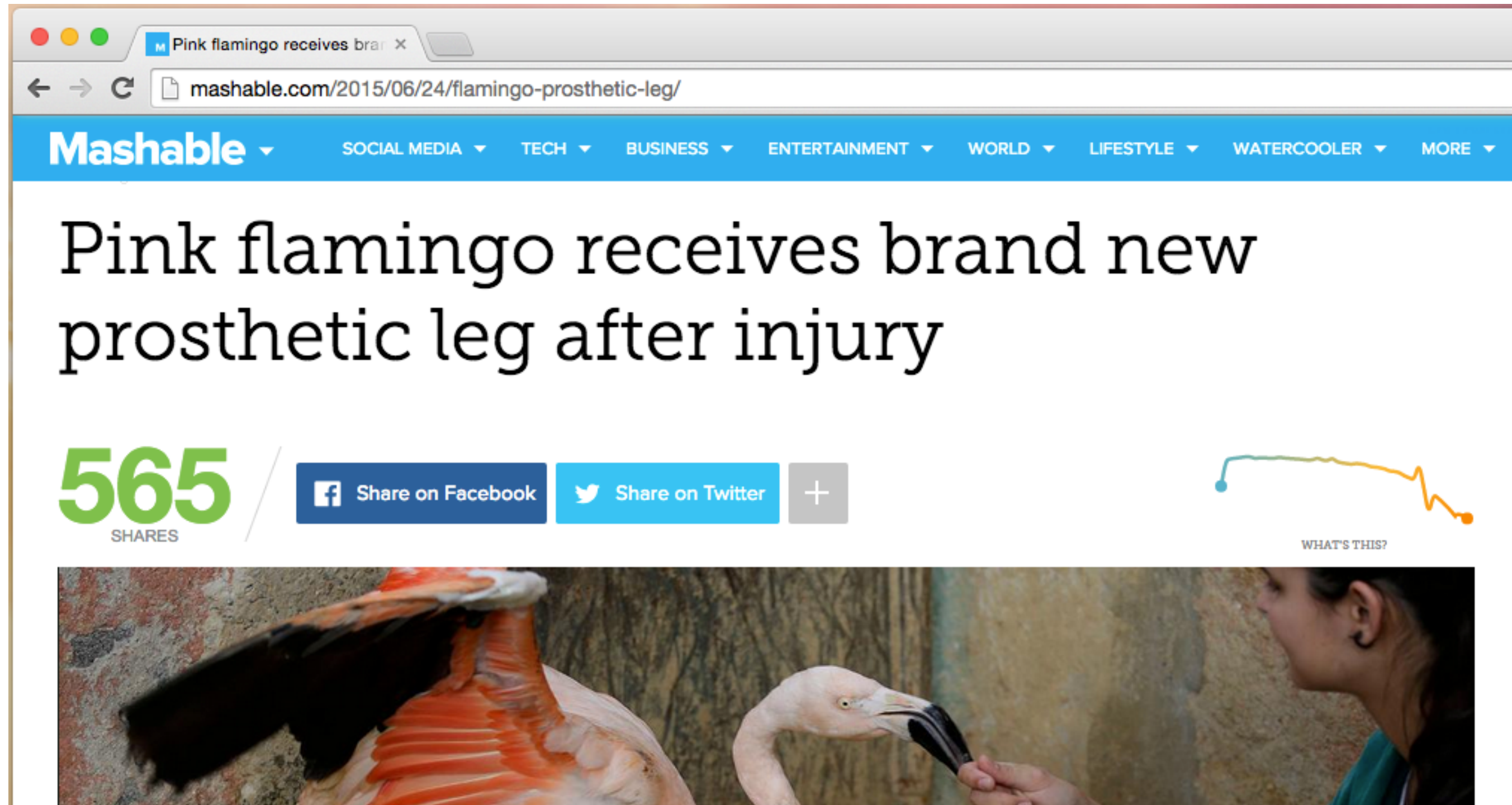
TimeUUID Data Type

45b94a50-12e5-11e5-9114-091830ac5256

Version 1 UUID comprised of:

- The number of 100 ns intervals since UUID epoch
- MAC address
- Clock sequence number to prevent duplicates

TimeUUID Use Case



The screenshot shows a web browser window with a single tab titled "Pink flamingo receives brand new prosthetic leg". The address bar displays the URL "mashable.com/2015/06/24/flamingo-prosthetic-leg/". The Mashable website header is visible, featuring the logo and navigation links for Social Media, Tech, Business, Entertainment, World, Lifestyle, Watercooler, and More. The main headline reads "Pink flamingo receives brand new prosthetic leg after injury". Below the headline, a green "565" is displayed with "SHARES" underneath. To the right of the share count are buttons for "Share on Facebook", "Share on Twitter", and a plus sign for additional sharing options. A small, colorful line graph is positioned to the right of the share buttons, with the text "WHAT'S THIS?" below it. The article's main image shows a pink flamingo with a prosthetic leg being held by a person's hand.

Pink flamingo receives brand new prosthetic leg after injury

565
SHARES

Share on Facebook Share on Twitter

WHAT'S THIS?

TimeUUID Use Case

```
CREATE TABLE course_page_views (  
    course_id text,  
    view_id timeuuid,  
    PRIMARY KEY (course_id, view_id)  
) WITH CLUSTERING ORDER BY (view_id DESC);
```

TimeUUID Functions

now

```
INSERT INTO course_page_views (course_id, view_id)
VALUES ('node-intro', now());
```

dateOf / unixTimestampOf

```
SELECT course_id, dateOf(view_id)
FROM course_page_views WHERE course_id = 'node-intro';
```

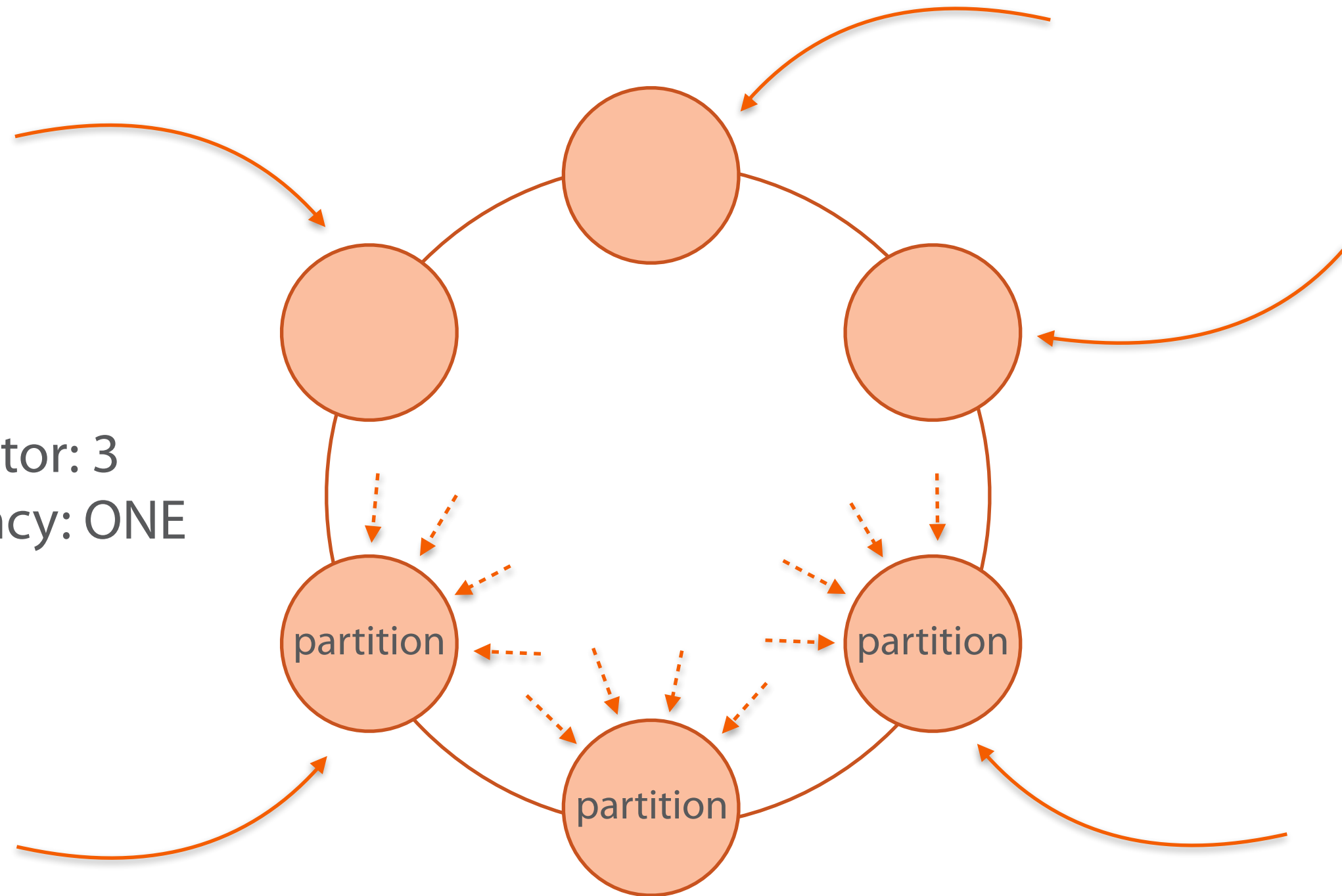
TimeUUID Functions

minTimeuuid / maxTimeuuid

```
SELECT dateOf(view_id)
FROM course_page_views
WHERE course_id = 'node-intro'
AND view_id >= maxTimeuuid('2015-01-01 00:00+0000')
AND view_id < minTimeuuid('2015-02-01 00:00+0000')
```

Storing Data with a Clustering Key

replication_factor: 3
write consistency: ONE



Storing Data with a Clustering Key

course_id='node-intro'

view_id='ece41c60-13bc-11e5-b559-4b636433f200'

Storing Data with a Clustering Key

course_id='node-intro'

view_id='f60b6730-13bc-11e5-b559-4b636433f200'

view_id='ece41c60-13bc-11e5-b559-4b636433f200'

Storing Data with a Clustering Key

course_id='node-intro'

view_id='f60b6730-13bc-11e5-b559-4b636433f200'

view_id='f208cab0-13bc-11e5-b559-4b636433f200'

view_id='ece41c60-13bc-11e5-b559-4b636433f200'

Bucketing Time Series Data

partition-key

A maximum of 2 billion cells
(rows x columns)

Must fit on a single node

Bucketing Time Series Data



Bucketing Use Case

```
CREATE TABLE clickstream (  
    year int,  
    month int,  
    click_id timeuuid,  
    ip inet,  
    url text,  
    PRIMARY KEY ((year, month), click_id)  
) WITH CLUSTERING ORDER BY (click_id DESC);
```

Inserting Bucketed Data

```
INSERT INTO clickstream  
    (year, month, click_id, ip, url)  
VALUES  
    (2015, 6, now(), '98.203.153.64',  
    'http://www.pluralsight.com');
```

Inserting Bucketed Data

year=2015, month=5

click_id='f208cab0-13bc-11e5-b559-4b636433f200'

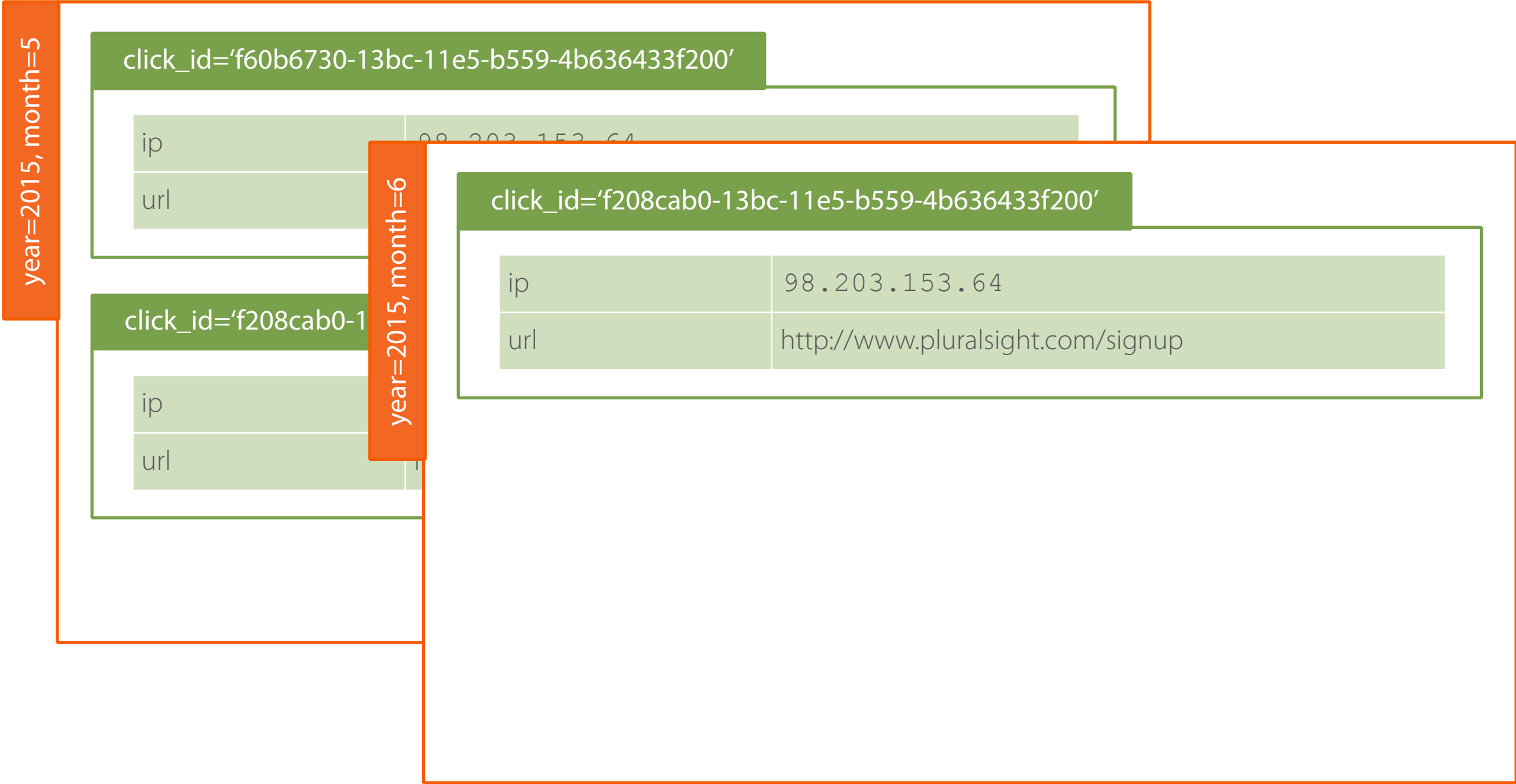
ip

98.203.153.64

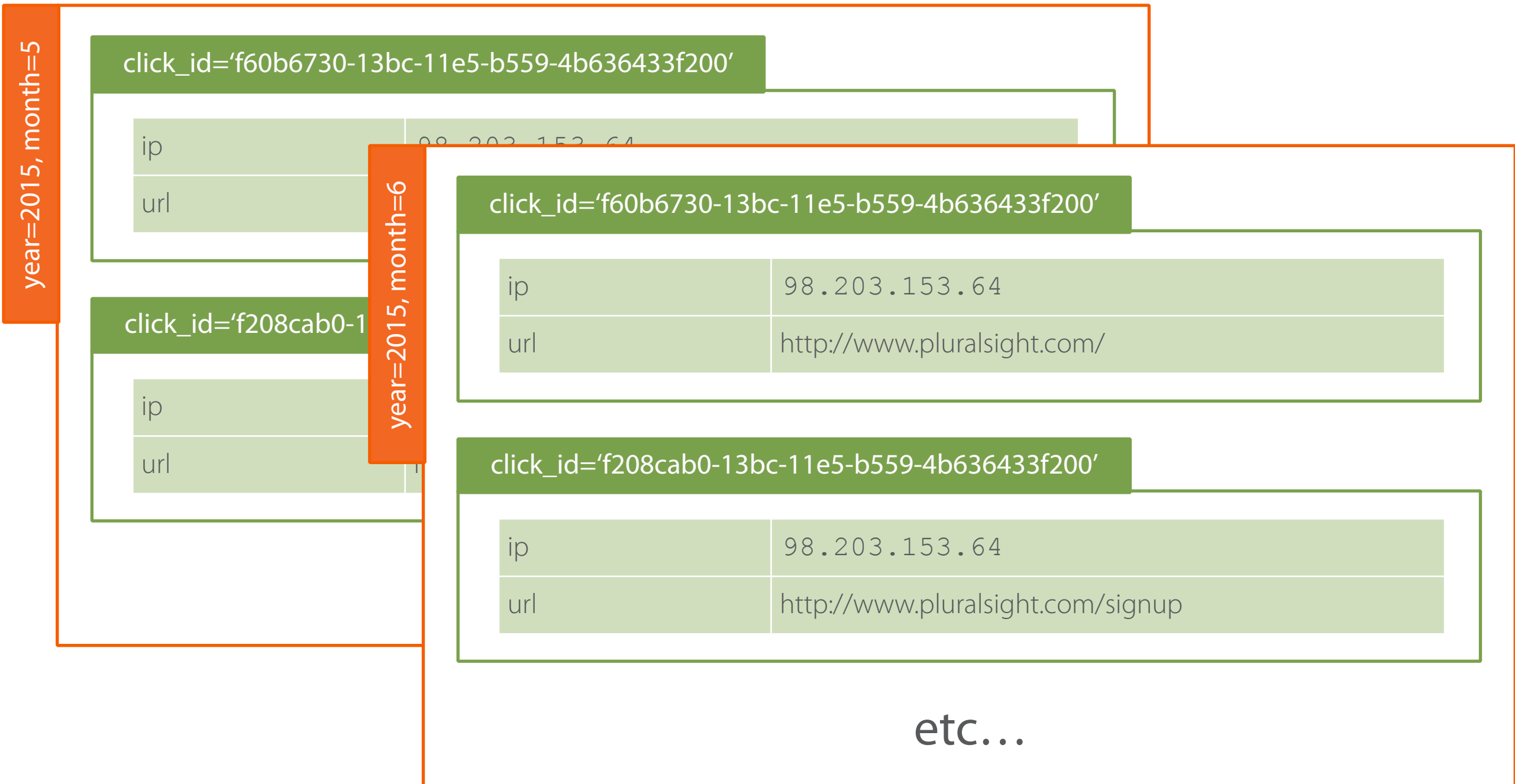
url

<http://www.pluralsight.com/signup>

Inserting Bucketed Data



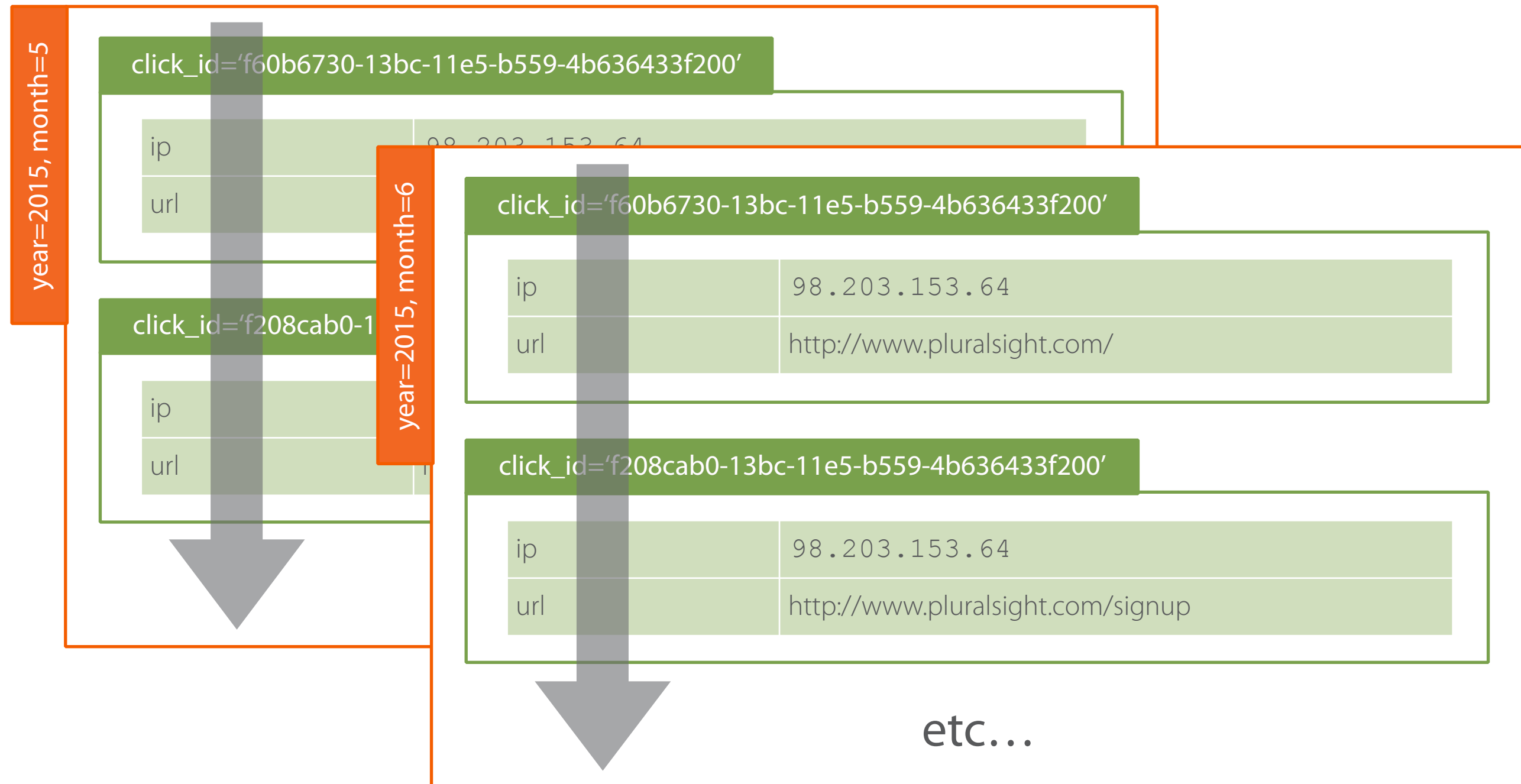
Inserting Bucketed Data



Selecting Bucketed Data

```
SELECT * FROM clickstream  
WHERE year=2015 AND month IN (6, 5)  
LIMIT 100;
```

Selecting Bucketed Data



Conclusion

- Clustering Keys and Composite Primary Keys
- Static Columns
- Time Series Data
- TimeUUID Data Type
- TTL vs. Bucketing