DataStax Hands-On Modelling

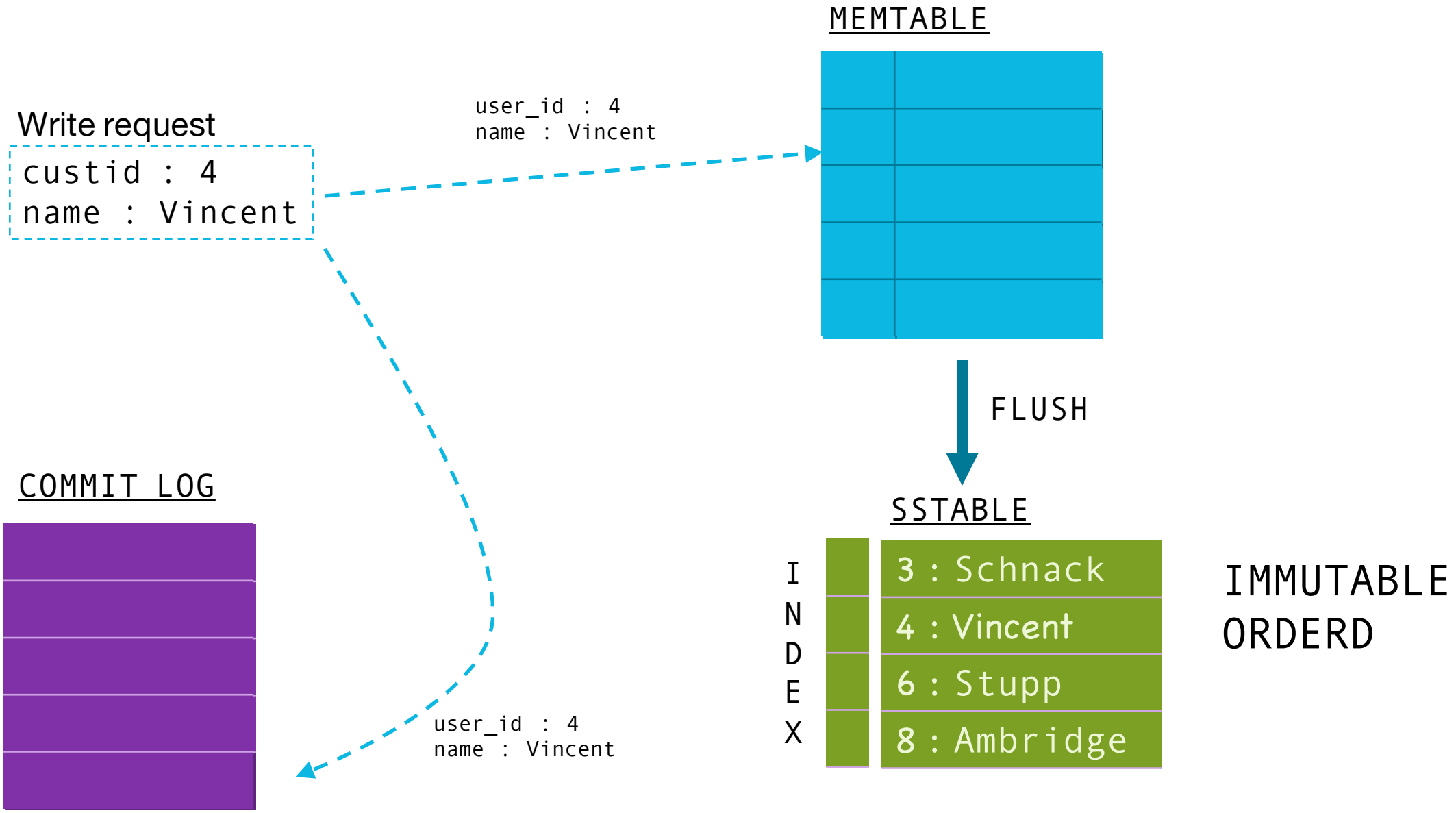
Negib Marhoul, Solution Engineer, DataStax

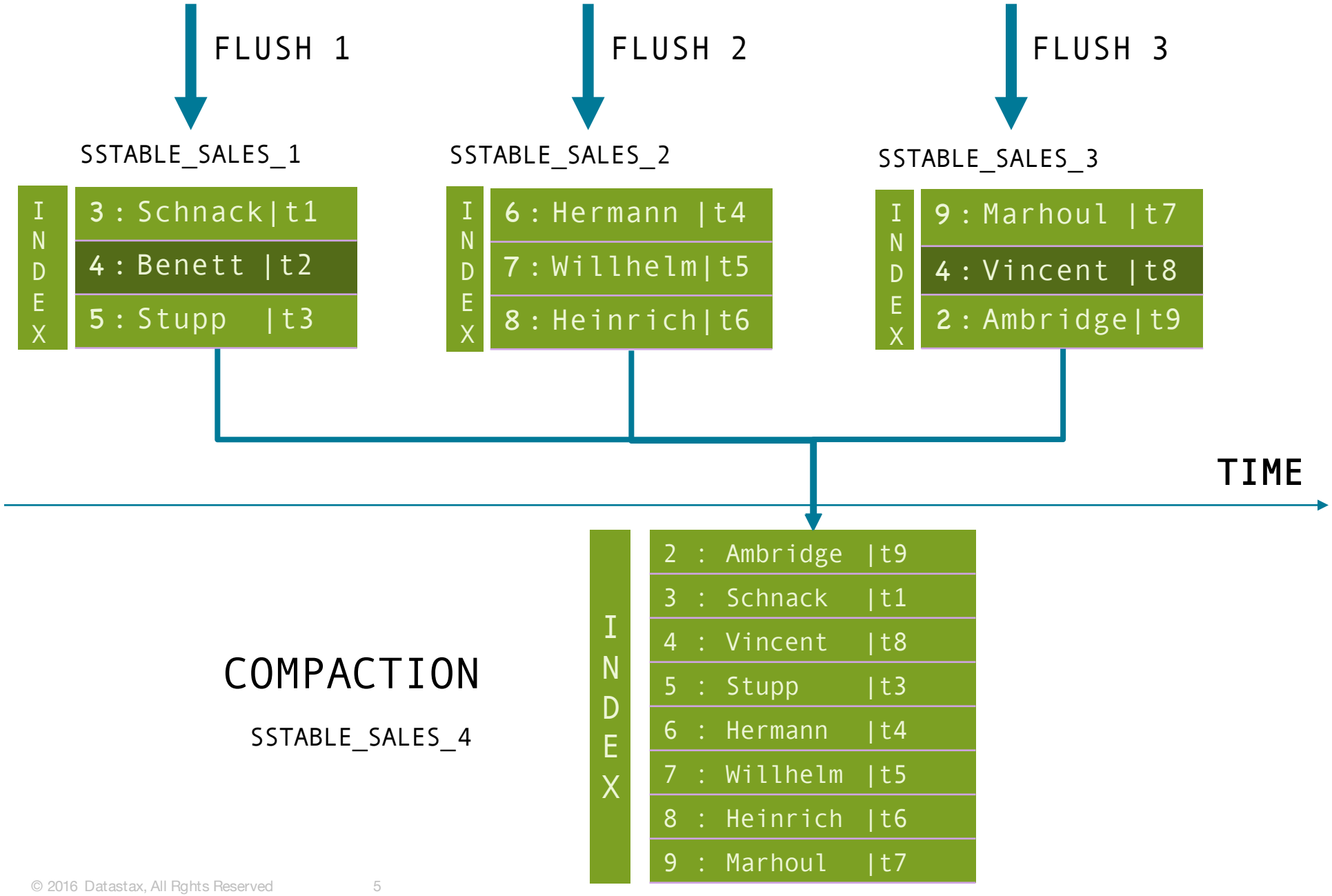
5. October 2017

Agenda

1	Storing data in Cassandra
2	Data Modelling, CQL basics
3	Hands-On Primary Keys

Storing data in Cassandra





DATASTAX

Data Modeling Objectives

Data modeling objectives

1. Get your data out of Cassandra
2. Reduce query latency, make your queries faster
3. Avoid disaster in production

Data modeling methodology

Design by query

- first, know your functional queries
- then design the table(s) for direct access
- just denormalize if necessary
- Spread data evenly around the cluster
- Minimize the number of partitions read
- Make sure to take your read/update ratio into account when designing your schema

Output of design phase = schema.cql

Then start coding

Know your functional queries

```
Query:
find users by id group by region and orderd by join date
```

- Grouping by an attribute
- Ordering by an attribute
- Filtering based on some set of conditions
- Enforcing uniqueness in the result set

DATASTAX

The partition key

Role

Partition key

- main entry point for query (INSERT/SELECT …)
- help distribute/locate data on the cluster

No partition key = full cluster scan

How to choose correct partition key ?

Good partition column

- choose functional identifier
- high cardinality (lots of distinct values)

Query:

Find sales by session?

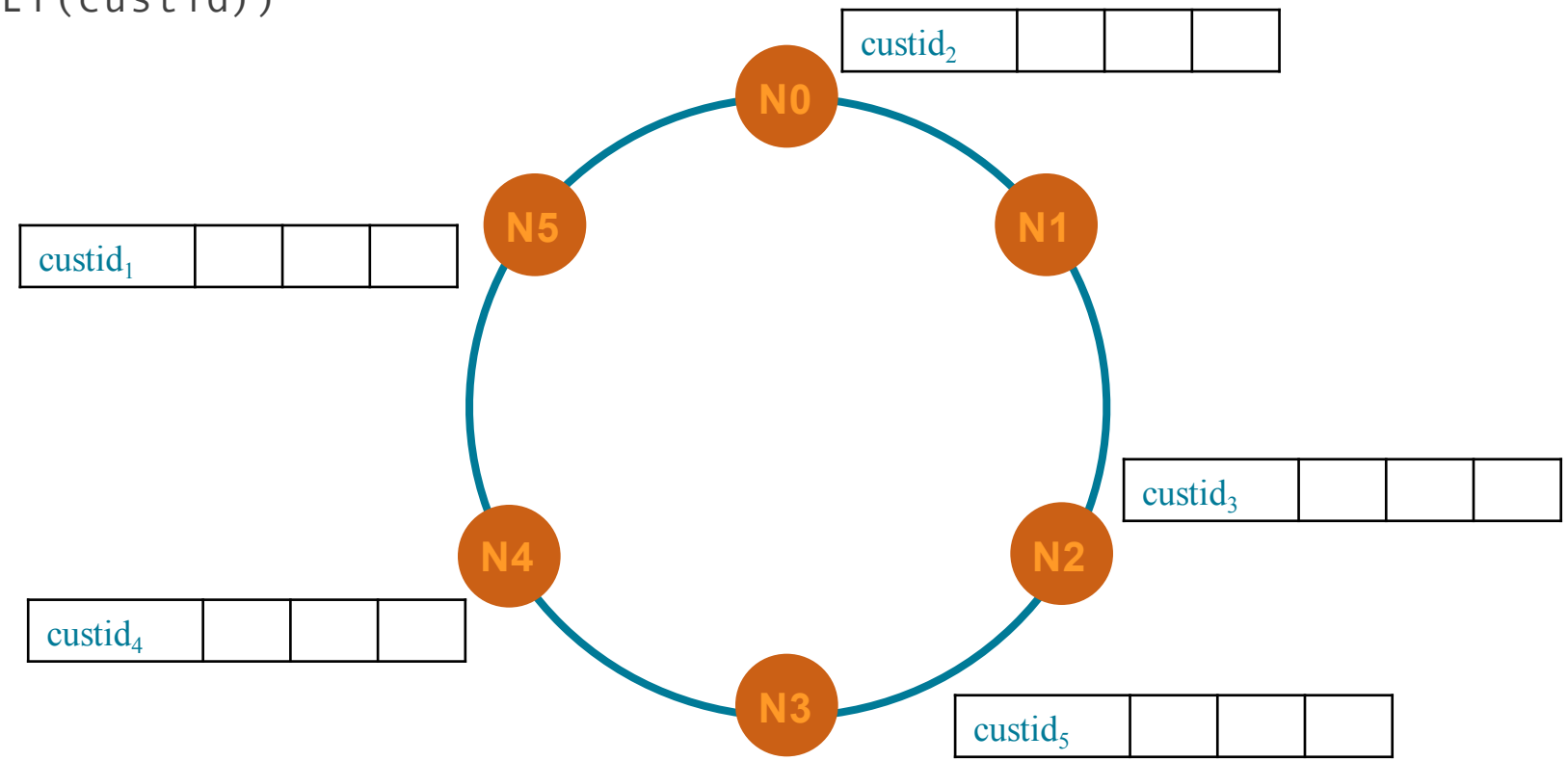
```
CREATE TABLE sales_by_customer (
    custid  int,
    salesdt  date,
    revenue  double,
    discount double,
    comment  txt,
    PRIMARY KEY(custid));
```

partition key (#partition)

DATASTAX

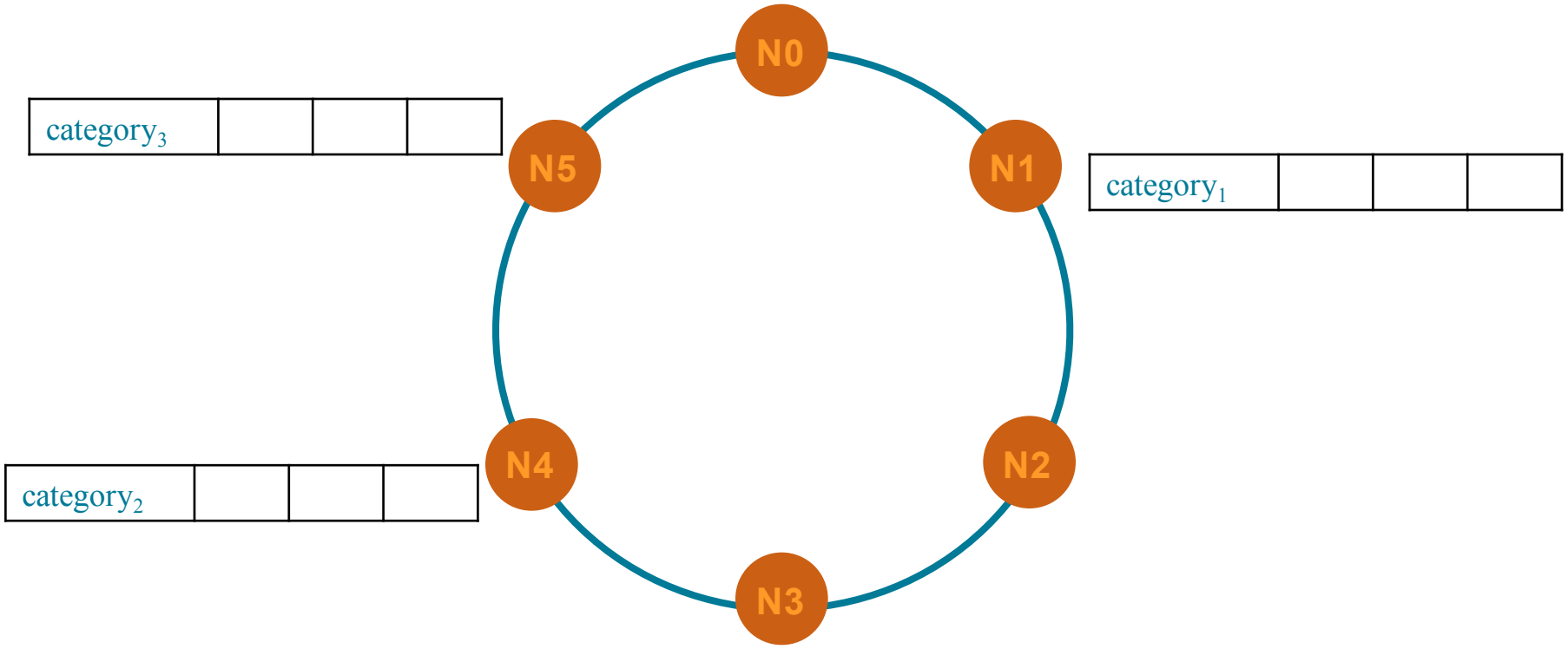
Example of good partition key

```
CREATE TABLE sales_by_order(
   custid int,
   …,
   PRIMARY KEY(custid))
```



Example of bad partition key

```
CREATE TABLE sales_by_cat(
   category text,
   …,
   PRIMARY KEY(category))
```



No Clustering Columns

PRIMARY KEY (custid) // no duplicate primary keys

custid	salesdt	revenue	discount	comment
1	20160101	300	10	PlayStation4
~~2~~	~~20160102~~	~~699~~	~~25~~	~~IPhone 7~~
2	20160103	750	10	IPhone 7

```
SELECT * FROM sales_by_customer WHERE custid = '2';
```

DATASTAX

Composite partition key

Multiple columns for partition key

- always known in advance (INSERT/SELECT …)
- are hashed together to the same token value

```
CREATE TABLE sales_by_customer (
       custid  int,
       salesdt  date,
       revenue  double,
       discount double,
       comment  txt,
       PRIMARY KEY((custid, salesdt)) );
```

```
SELECT * FROM sales_by_customer WHERE custid = … AND salesdt = …
```

```
SELECT * FROM sales_by_customer  WHERE custid = … AND salesdt IN (xxx, yyy …)
```


DATASTAX

Compound Partition Key

PRIMARY KEY ((custid, salesdt)) // hash(custid, salesdt) ➔ token

custid	salesdt	revenue	discount	comment
1	20160101	300	10	PlayStation4
2	20160102	699	25	IPhone 7
2	20160103	750	10	IPhone 7

```
SELECT * FROM sales_by_customer WHERE custid = 2 AND salesdt = '1/2/2016';
```

The clustering column(s)

Role

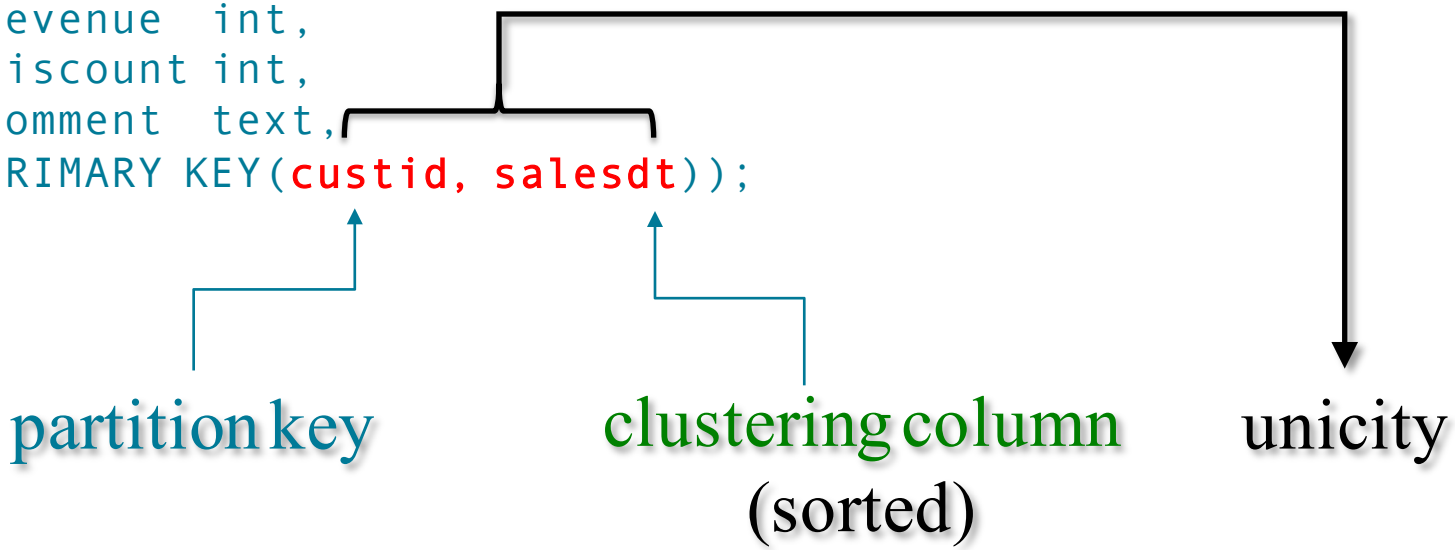
Clustering column(s)

- simulate 1 – N relationship
- and sort data (logically & on disk)

DATASTAX

Clustered table (1 – N)

```
CREATE TABLE sales_by_customer (
    custid   int,
    salesdt  date,
    revenue  int,
    discount int,
    comment  text,
    PRIMARY KEY(custid, salesdt));
```



Recommended syntax

```
PRIMARY KEY((custid), salesdt);
```

Clustering Columns Create Wide Rows

PRIMARY KEY ((custid),salesdt) // default sort and range queries

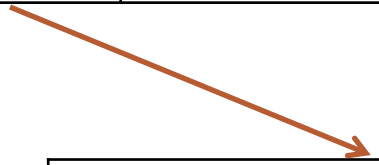
custid	salesdt		
1	20160102		
	719, 10, new customer iPhone7		
2	20160101	20160102	20160103
	719, 10, new customer iPhone7	45, 50, marketing campaign xyz	187, 10, sonyplaystation 3

Merged, Sorted and Stored Sequentially

```
SELECT * FROM sales_by_customer WHERE custid = 1 and salesdt > '1/1/2016';
```

What's Stored With Each Column?

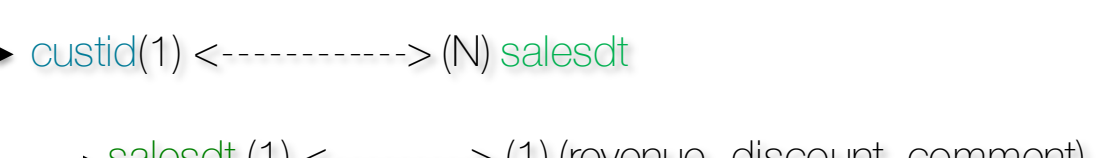
custid	salesdt		
1	20160102		
	719, 10, new customer iPhone7		
2	20160101	20160102	20160103
	719, 10, new customer iPhone7	45, 50, marketing campaign xyz	187, 10, sonyplaystation 3



column name : "comment"
column value : "new customer iPhone7"
timestamp : 1353890782373000
TTL : 3600

Columns relationship and ordering

```
CREATE TABLE sales_by_customer (
   custid   int,
   salesdt  date,
   revenue  int,
   discount int,
   comment  text,
PRIMARY KEY((custid), salesdt))
WITH CLUSTERING ORDER BY (dt ASC)
```



custid	salesdt		
1	20160102		
	719, 10, new customer iPhone7		
2	20160101	20160102	20160103
	719, 10, new customer iPhone7	45, 50, marketing campaign xyz	187, 10, sonyplaystation 3

```
SELECT * FROM sales_by_customer WHERE custid = 1 and salesdt > '1/1/2016';
```


DATASTAX

Multiple clustering columns

```
CREATE TABLE sales_by_cust (
   custid   int,
   salesch  text,
   salesdt  date,
   revenue  int,
   discount int,
   comment  text,
PRIMARY KEY((custid), salesch, salesdt))
WITH CLUSTERING ORDER BY (salesch ASC, salesdt DESC)
```

custid	salesch, salesdt		
1	online		
	20160102		
	719, 10, new customer iPhone7		
2	online	online	store
	20160101	20160102	20160103
	719, 10, new customer iPhone7	45, 50, marketing campaign xyz	187, 10, sonyplaystation 3

```
SELECT * FROM sales_by_customer
   WHERE custid = 1 AND
   salesch = "online" AND salesdt >= 1/2/16 AND salesdt <= 1/3/16;
```


DATASTAX

Primary key summary

```
PRIMARY KEY((custid), salesch, salesdt)
```

Unicity of (custid, salesch, salesdt)

Primary key summary

```
PRIMARY KEY((custid), salesch, salesdt)
```

Used to locate node in the cluster
Used to locate partition in the node

DATASTAX

Primary key summary

```
PRIMARY KEY((custid), salesch, salesdt)
```

Used to lookup rows in a partition
Used for data sorting and range queries

DATASTAX

Lab 3 : Hands-on Primary Keys

Modelling Challenge (almost no challenge)

Query:

- Find all user details by user id
- Give me one user transaction by user id and date
- Give me all user transaction by user id between two dates

- **User details:**
  - user_id, firstname, lastname, ...

- **User transactions:**
  - user_id, date, item, quantity, price, ...

DATASTAX

Other critical details

Huge partitions

```
PRIMARY KEY((sensor_id), dt))
```

Data for the same sensor stay in the same partition on disk

DATASTAX

Huge partitions

```
PRIMARY KEY((sensor_id), dt))
```

Data for the same sensor stay in the same partition on disk

If insert rate = 100/sec, how big is my partition after 1 year ?

→ 100 x 3600 x 24 x 365= 3 153 600 000 cells on disks

Huge partitions

```
PRIMARY KEY((sensor_id), dt))
```

Theorical limit of # cells for a partition = 2×10^9

Practical limit for a partition on disk

- 100Mb
- 100 000 – 1000 000 cells

Reasons ? Make maintenance operations easier

- compaction
- repair
- bootstrap …

DATASTAX

Sub-partitioning techniques

```
PRIMARY KEY((sensor_id, day), dt))
```

→ 100 x 3600 x 24 = 8 640 000 cells on disks ✔

Sub-partitioning techniques

```
PRIMARY KEY((sensor_id, day), dt))
```

→ 100 x 3600 x 24 = 8 640 000 cells on disks ✔☐

But impact on queries:

- need to provide sensor_id & day for any query
- how to fetch data across N days ?

Data deletion and tombstones

```
DELETE FROM sensor_data
   WHERE sensor_id = .. AND dt = …
```

Logical deletion of data but:

- new physical "tombstone" column on disk
- disk space usage will increase !

The "tombstone" columns will be purged later by compaction process …

Lab 3 : Hands-on Primary Keys

Vielen Dank!