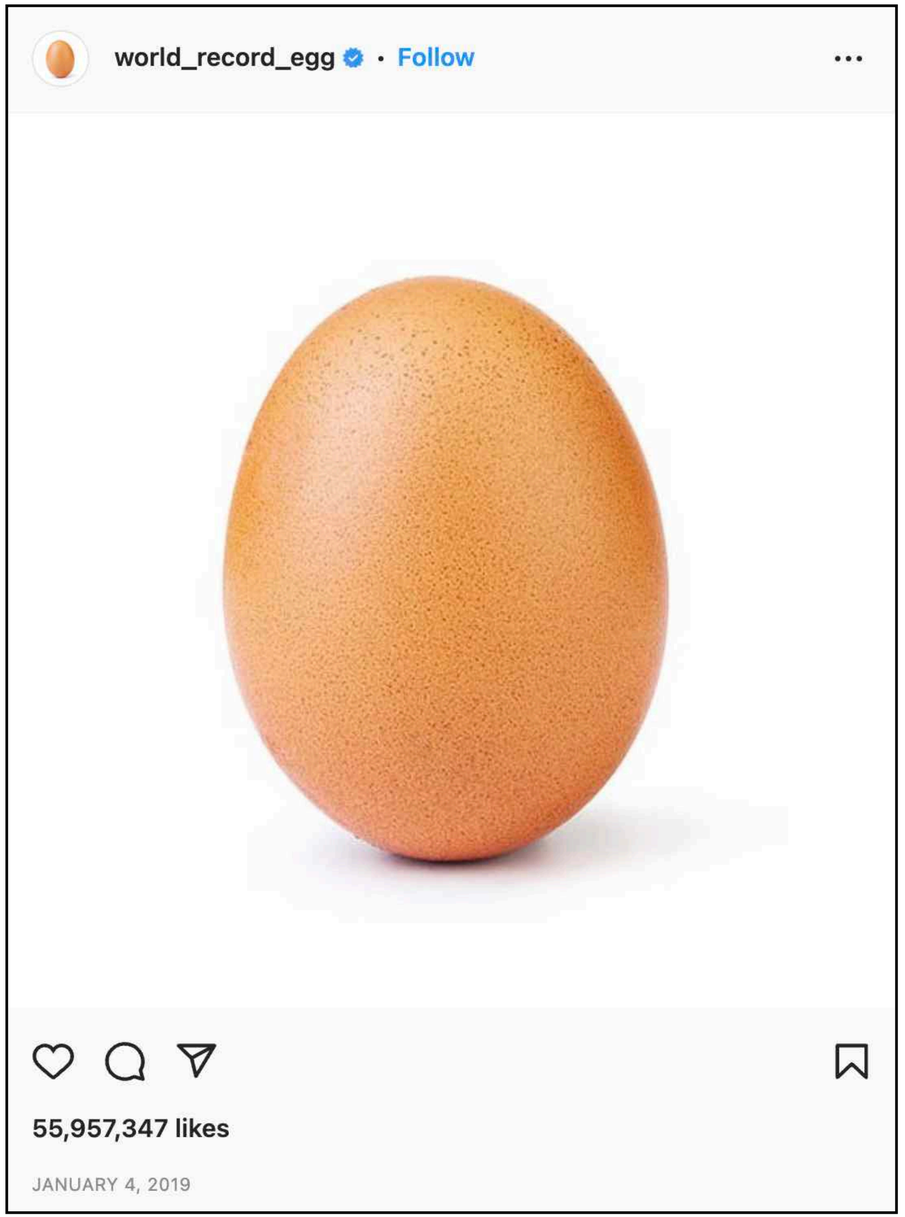
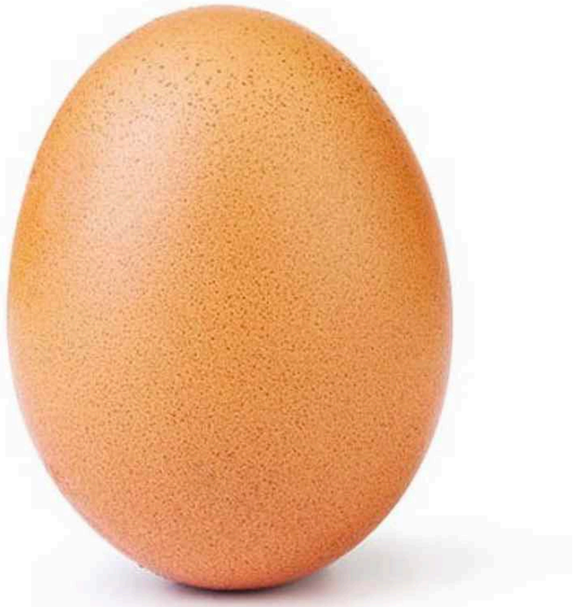
Cassandra - CQL

Big Data Systems

Dr. Rubi Boim


world_record_egg
Follow
55,957,347 likes
JANUARY 4, 2019

world_record_egg · Follow



55,957,347 likes

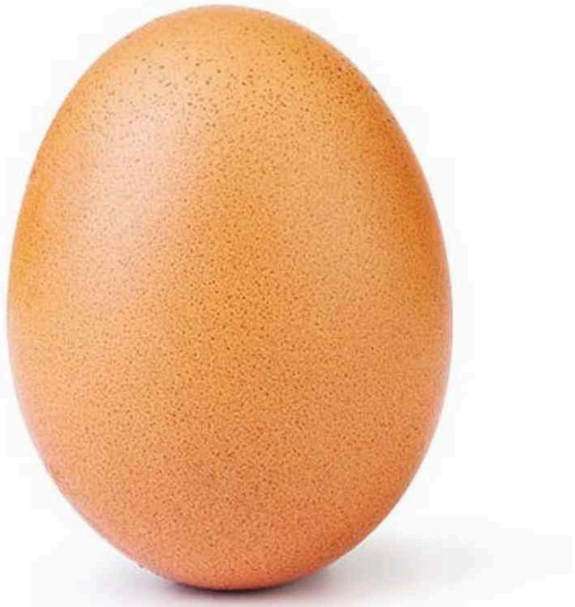
JANUARY 4, 2019

leomessi · Follow
Lusail Stadium

56,006,307 likes

1 DAY AGO

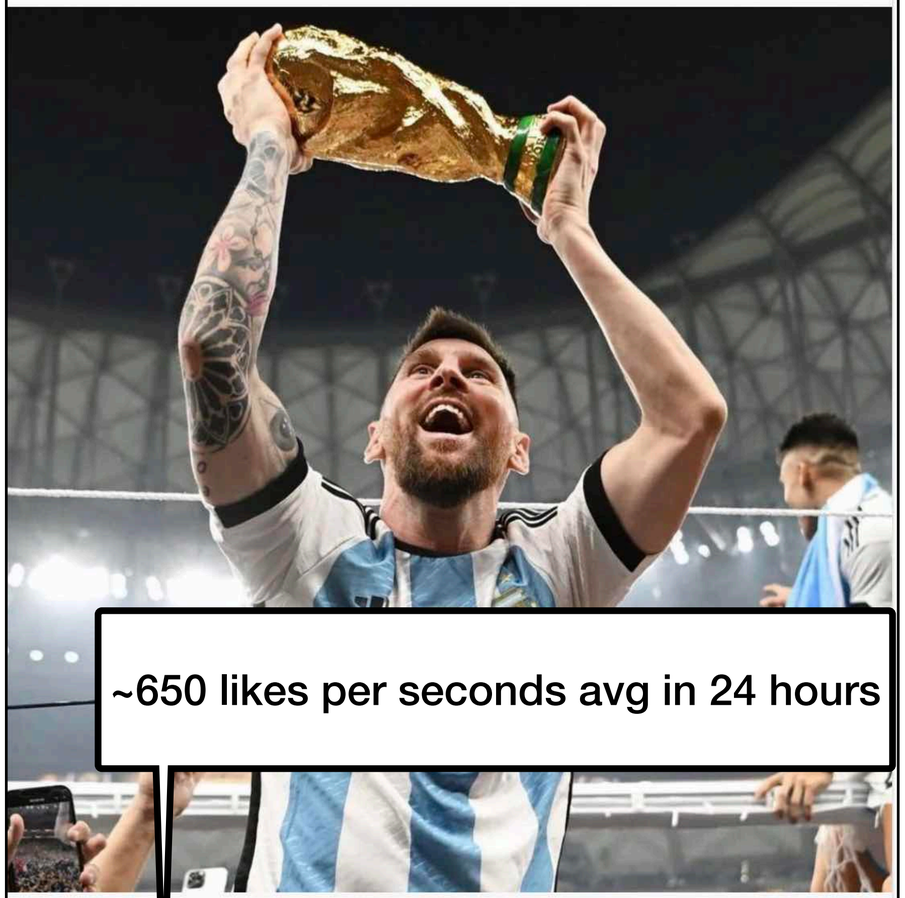
world_record_egg • Follow



55,957,347 likes

JANUARY 4, 2019

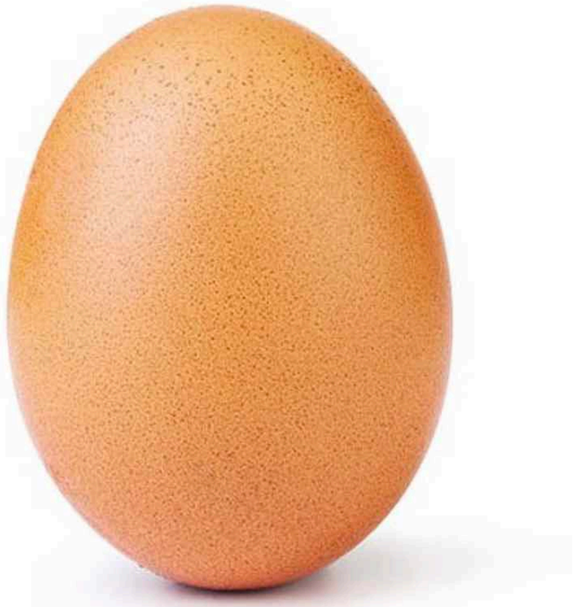
leomessi • Follow
Lusail Stadium



56,006,307 likes

1 DAY AGO

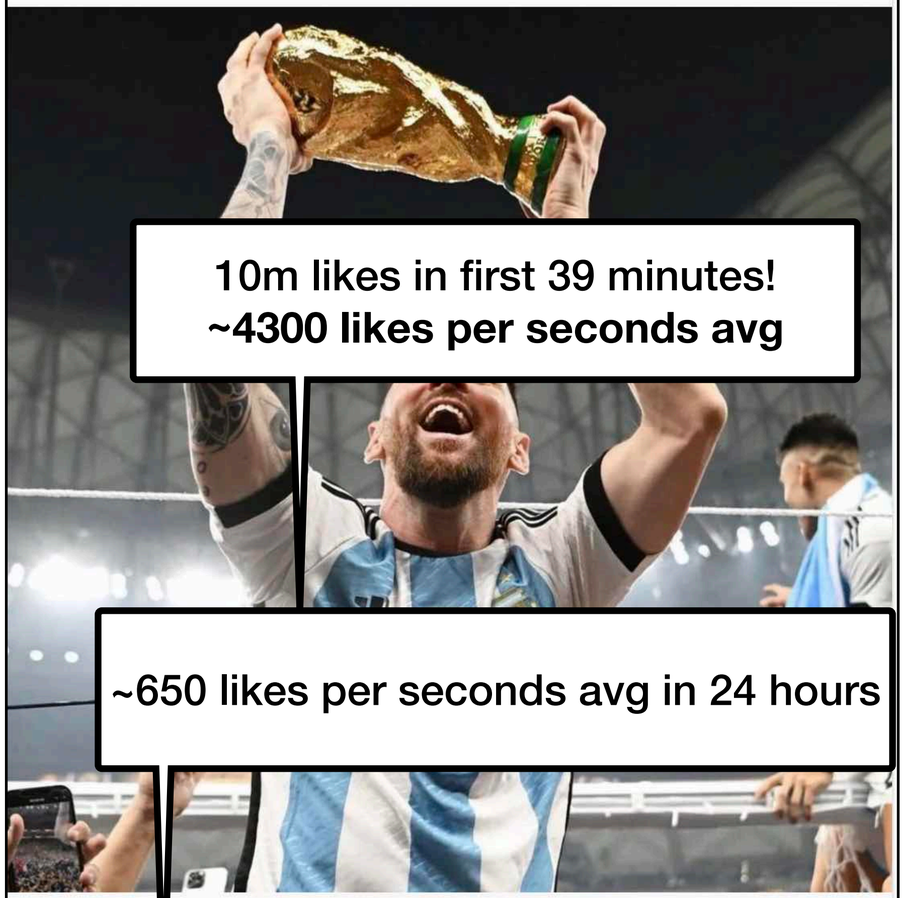
world_record_egg · Follow



55,957,347 likes

JANUARY 4, 2019

leomessi · Follow
Lusail Stadium



56,006,307 likes

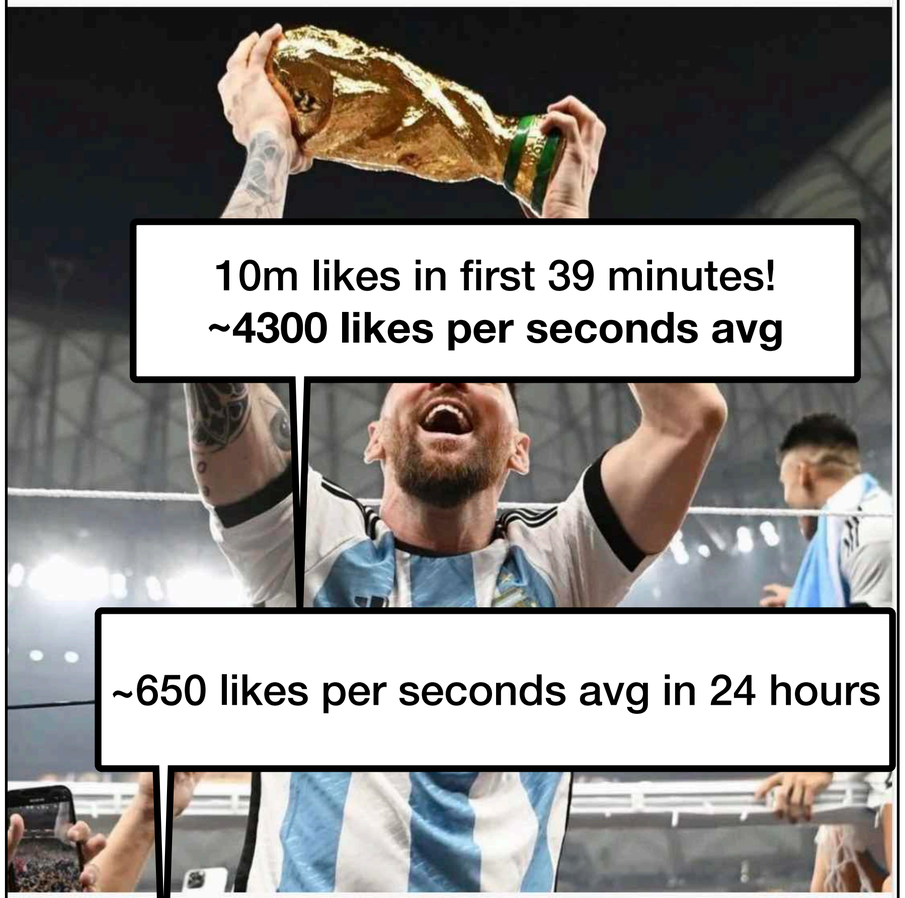
1 DAY AGO

jenniferaniston • Follow

16,458,620 likes

OCTOBER 15, 2019

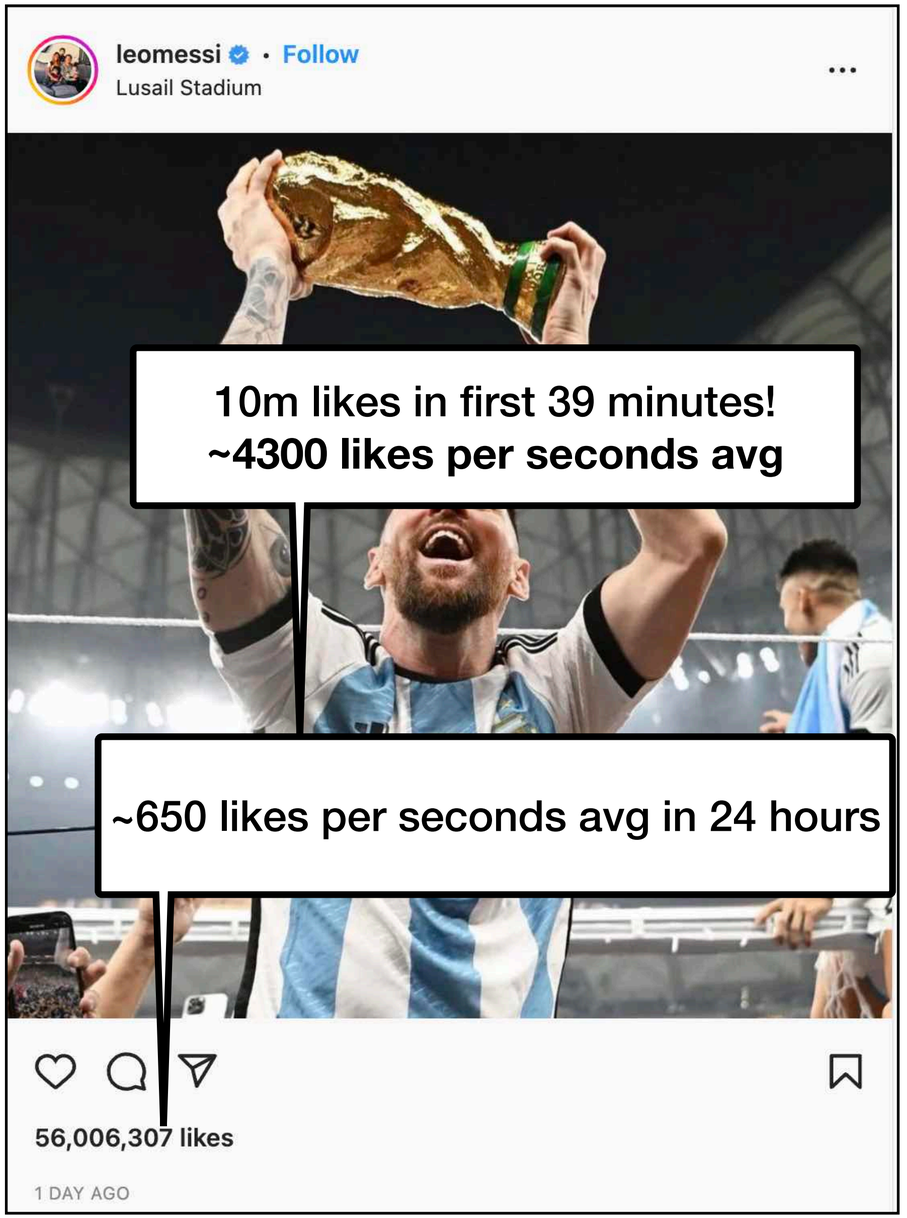
leomessi • Follow
Lusail Stadium



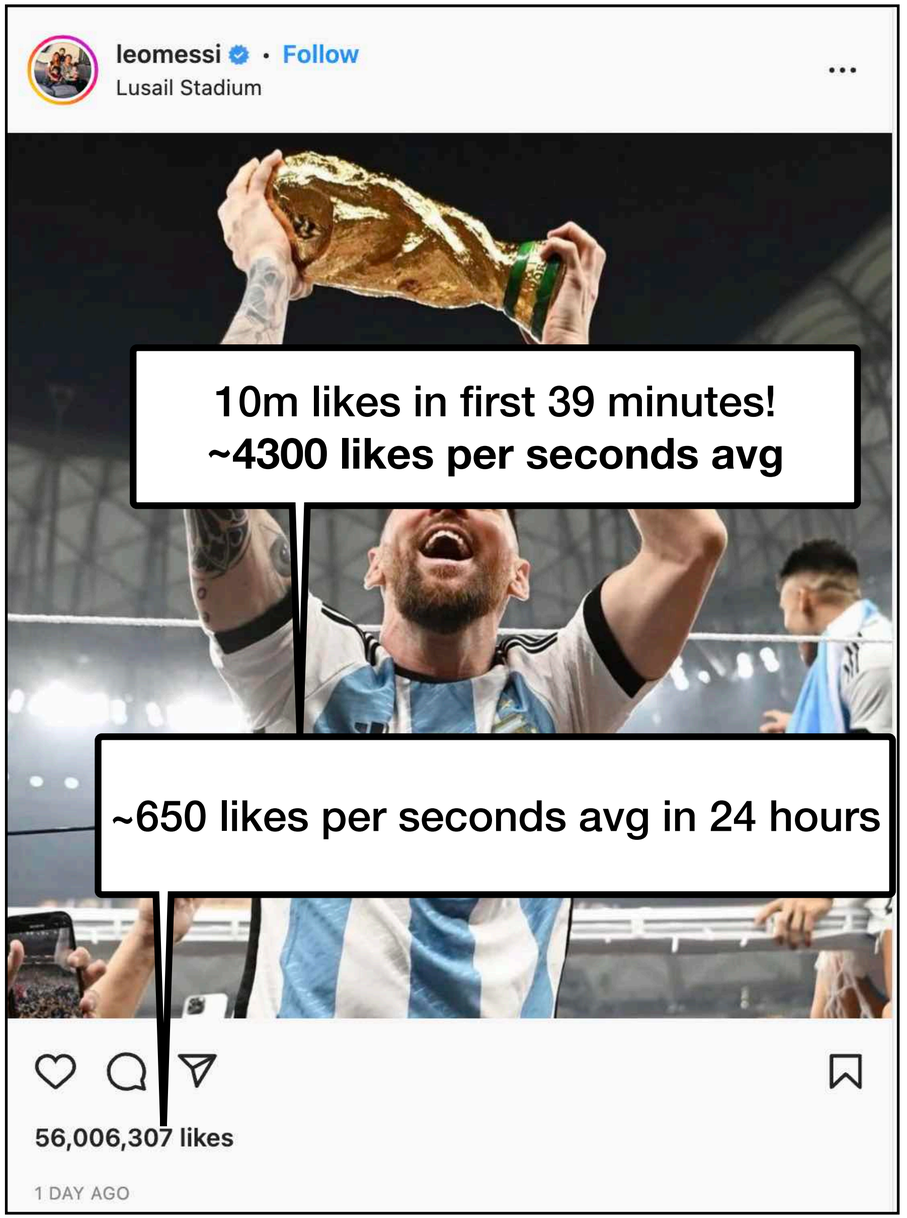
56,006,307 likes

1 DAY AGO

jenniferaniston
Follow
9m likes in first 24 hours
~100 likes per seconds avg
16,458,620 likes
OCTOBER 15, 2019


leomessi
Follow
Lusail Stadium
10m likes in first 39 minutes!
~4300 likes per seconds avg
~650 likes per seconds avg in 24 hours
56,006,307 likes
1 DAY AGO

jenniferaniston
Follow
So why did Instagram crashed?
9m likes in first 24 hours
~100 likes per seconds avg
16,458,620 likes
OCTOBER 15, 2019

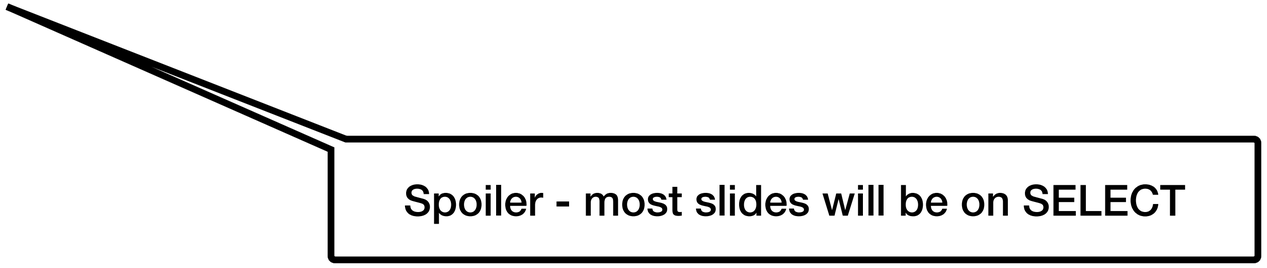

leomessi
Follow
Lusail Stadium
10m likes in first 39 minutes!
~4300 likes per seconds avg
~650 likes per seconds avg in 24 hours
56,006,307 likes
1 DAY AGO

Lionel Messi makes 6 while Cristiano Ronaldo makes 5 out of the top 20 most-liked posts, with posts mainly released around the 2022 FIFA World Cup.

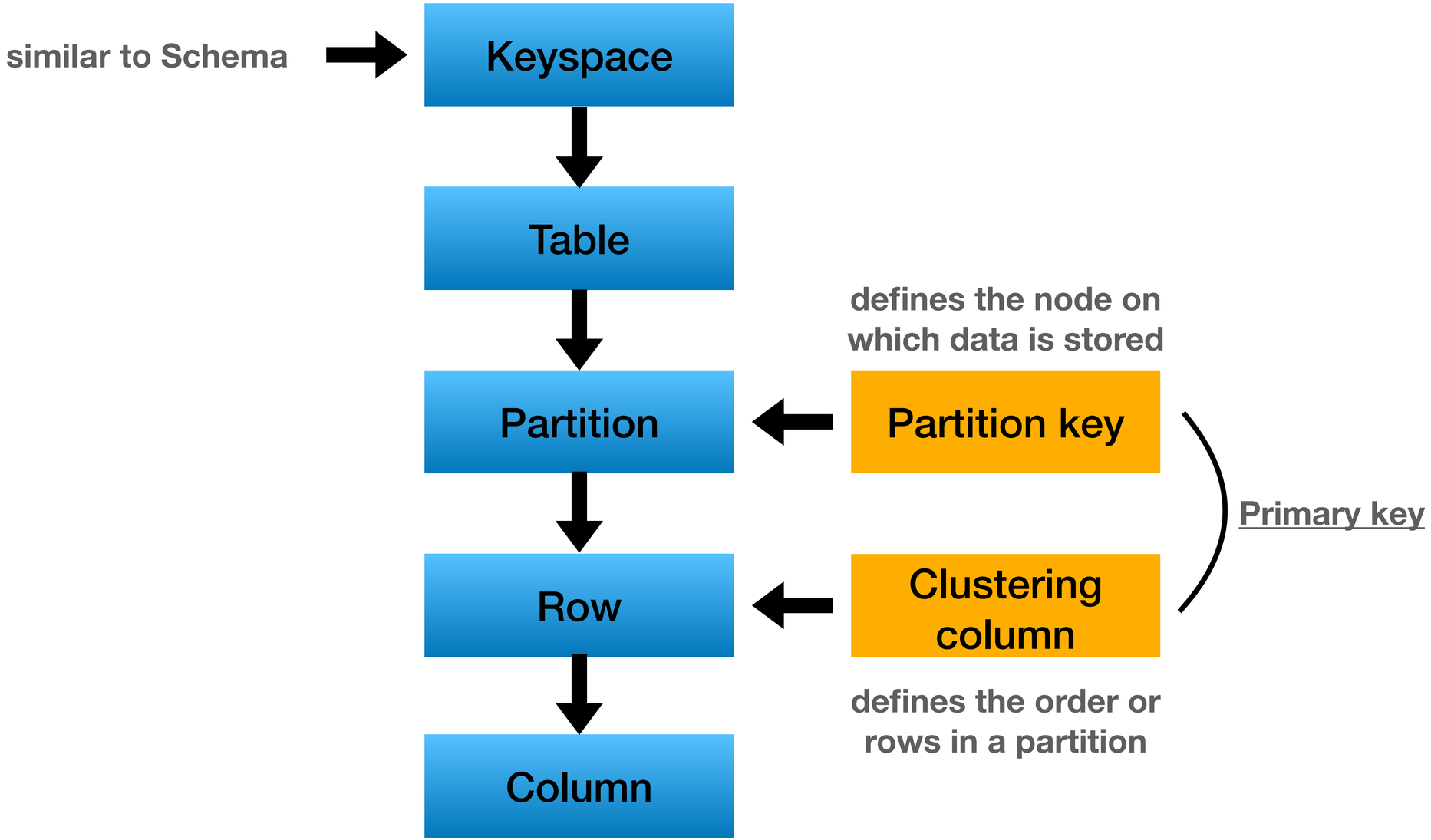
Rank	Account name(s)	Owner	Post description	Post	Likes (millions)	Date posted (UTC)
1	@leomessi	Lionel Messi	Celebrating winning the 2022 FIFA World Cup	[1]	74.8	December 18, 2022
2	@world_record_egg	Chris Godfrey	An egg	[2]	60.5	January 4, 2019
3	@leomessi	Lionel Messi	Lionel Messi in bed with the FIFA World Cup Trophy	[3]	53.6	December 20, 2022
4	@kishore_mondal_official	Kishore Mondal	Reel singing "Jeene Laga Hoon"	[4]	51.9	April 11, 2025
5	@Insta360	Insta360	Video of a tiny camera filming a toddler walking[16]	[5]	48.4	May 11, 2025
6	@cristiano	Cristiano Ronaldo	Lionel Messi and Cristiano Ronaldo playing chess, advertising for Louis Vuitton	[6]	41.4	November 19, 2022
7	@leomessi	Lionel Messi	Lionel Messi on an airplane with the FIFA World Cup Trophy	[7]	41.1	December 19, 2022
8	@taylorswift @killatrav	Taylor Swift Travis Kelce	Engagement announcement	[8]	37.6	August 26, 2025
9	@emilio.piano	Emil Reinert	Reel of a little girl singing "My Heart Will Go On"	[9]	37.1	September 8, 2024
10	@cristiano	Cristiano Ronaldo	Cristiano Ronaldo taking an icy dip in snowy Lapland, Finland	[10]	35.5	December 24, 2024
11	@leomessi	Lionel Messi	Lionel Messi's visit to Camp Nou	[11]	35.1	November 10, 2025
12	@xxxtentacion	XXXTentacion	Final post before his death	[12]	34.1	May 19, 2018
13	@leomessi	Lionel Messi	Celebrating the 2022 FIFA World Cup win in Argentina	[13]	33.5	December 21, 2022
14	@cristiano @alnassr	Cristiano Ronaldo Al-Nassr FC	Announcement of Cristiano Ronaldo joining Al-Nassr FC	[14]	33.2	December 30, 2022
15	@cristiano	Cristiano Ronaldo	After elimination of Portugal from the 2022 FIFA World Cup	[15]	33.0	December 11, 2022
16	@jiangzhibin24	liz_6	Reel of an AI-generated sunset	[16]	32.9	August 5, 2023
17	@leomessi	Lionel Messi	Lionel Messi and Cristiano Ronaldo playing chess, advertising for Louis Vuitton	[17]	31.9	November 19, 2022
18	@cristiano @georginagio	Cristiano Ronaldo Georgina Rodríguez	Pregnancy announcement	[18]	31.6	October 28, 2021
19	@kavadi_druvam_kalasamithi_	Druvam Kalasamithi	Reel of a colorful wedding entrance	[19]	31.4	May 12, 2025
20	@aline_sarakbii @mostafa_srakbi	aline-sarakbii mostafa_srakbi	Reel of a baby	[20]	31.3	June 10, 2024

Cassandra CQL

- Terminology
- Keyspaces
- Tables
- Data types
- DDL / DML



Terminology (Cassandra)



Keyspace

- High level container - AKA “schemas” from rDB
- **replication factor strategy**
  - “`SimpleStrategy`”: entire cluster
  - “`NetworkTopologyStrategy`”: different settings for each DC

Keyspace

```
CREATE KEYSPACE BigDataCourse WITH REPLICATION = {
 'class'                : 'SimpleStrategy',
 'replication_factor': 1
};
```

```
CREATE KEYSPACE BigDataCourse WITH REPLICATION = {
 'class'                :'NetworkTopologyStrategy',
 'israel'               : 3 , // Datacenter 1
 'us'                   : 2   // Datacenter 2
};
```

Use & Describe

- USE: switch between key spaces in CQL

```
USE bigdatacourse
```

```
JAVA:
CassandraConnectionPool connectionPool.setKeyspace(“bigdatacourse”)
```

- DESCRIBE: display detailed information in CQL
  (see manual for more options)

```
DESCRIBE KEYSPACES/KEYSPACE/TABLES/TABLE/...
```

CREATE TABLE

```
CREATE TABLE students (
   column1    TEXT,
   column2    INT,
   column3    UUID,
   PRIMARY KEY (column1)
);
```

```
CREATE TABLE [IF NOT EXISTS] [keyspace_name.]table_name (
   column_definition [, ...]
   PRIMARY KEY (column_name [, column_name ...])
[WITH table_options
   | CLUSTERING ORDER BY (clustering_column_name order])
   | ID = 'table_hash_tag'
   | COMPACT STORAGE]
```

Data types (basic)

- `TEXT` utf8
- `INT` signed 32bits
- `BIGINT` signed 64bits
- `TIMESTAMP` 64bits
- `FLOAT` 32bits floating point
- `DOUBLE` 64bits floating point
- `DECIMAL` variable-precision decimal
- `UUID` universally unique identifier, 128bits
- `TIMEUUID` sortable UUID, embedded timestamp
- `BLOB` arbitrary bytes

Data types (basic)

- `TEXT` utf8
- `INT` signed 32bits
- `BIGINT` signed 64bits
- `TIMESTAMP` 64bits
- `FLOAT` 32bits floating point
- `DOUBLE` 64bits floating point
- `DECIMAL` variable-precision decimal
- **`UUID`** **<u>universally</u> unique identifier, 128bits**
- **`TIMEUUID`** **sortable UUID, embedded timestamp**
- `BLOB` arbitrary bytes

**Unique across all nodes,
regardless of the number of nodes**

Note on generating unique IDs

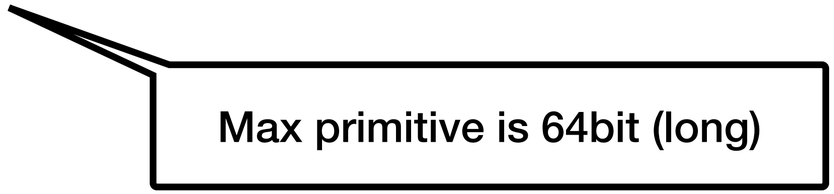
- Not trivial for distributed systems
- UUID / TIMEUUID are great
  - Downside - requires 128bit

    what's the problem with java primitives?

Note on generating unique IDs

- Not trivial for distributed systems
- UUID / TIMEUUID are great
  - Downside - requires 128bit

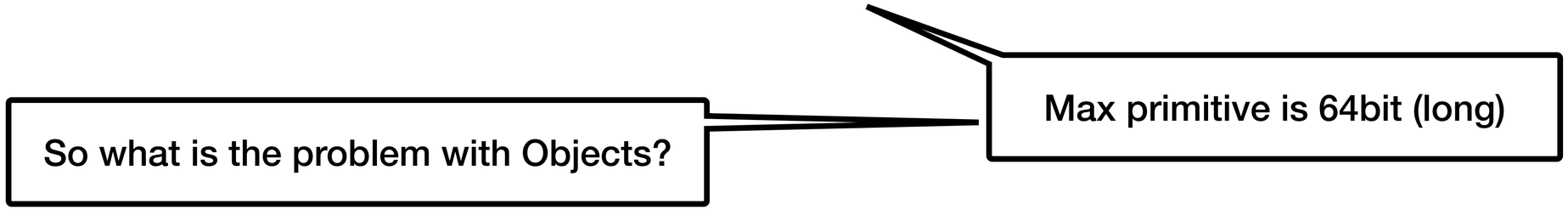
    what's the problem with java primitives?



Note on generating unique IDs

- Not trivial for distributed systems
- UUID / TIMEUUID are great
  - Downside - requires 128bit

    what's the problem with java primitives?



Note on generating unique IDs

THIS IS SUPER CRUCIAL!!!

Java can be SUPER SUPER FAST (like C++).

The problem is with the GC…

So how to avoid the GC?

what's the problem with java primitives?

Max primitive is 64bit (long)

So what is the problem with Objects?

Note on generating unique IDs

THIS IS SUPER CRUCIAL!!!

Java can be SUPER SUPER FAST (like C++).

The problem is with the GC…

So how to avoid the GC?

what's th problem with java primitives?

So what is the problem wi

If possible - create the object once and reuse them.
—> Do not use the key word "new"…

(easier said than done)

Note on

THI

Java can be

The

So how to avoid the GC?

This is also true for C++
(avoid as much as possible “new” and “delete”)

what’s th problem with java primitives?

So what is the problem wi

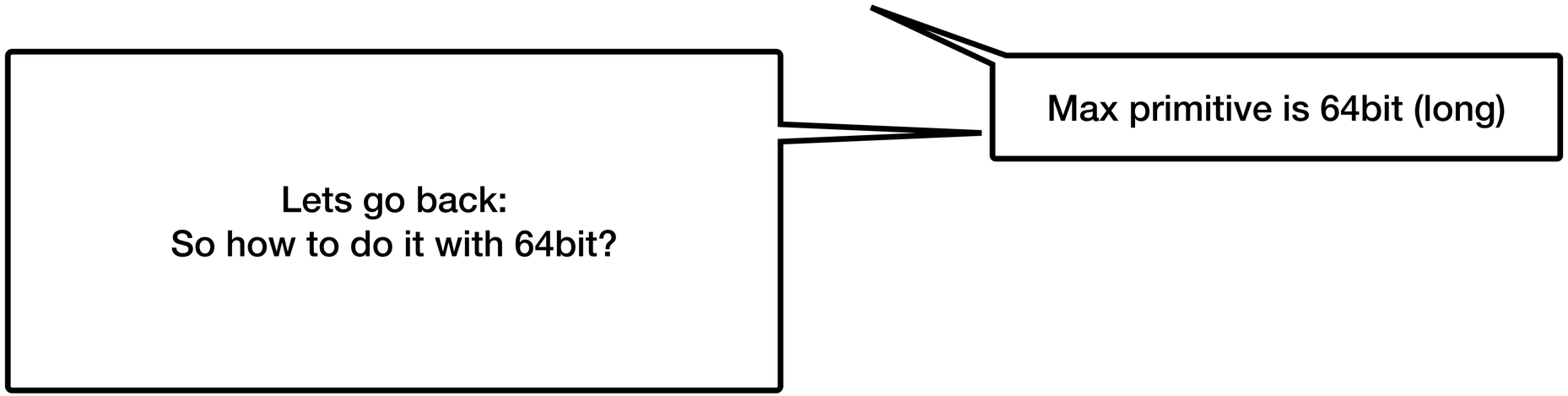
If possible - create the object once and reuse them.
—> Do not use the key word “new”…

(easier said than done)

Note on generating unique IDs

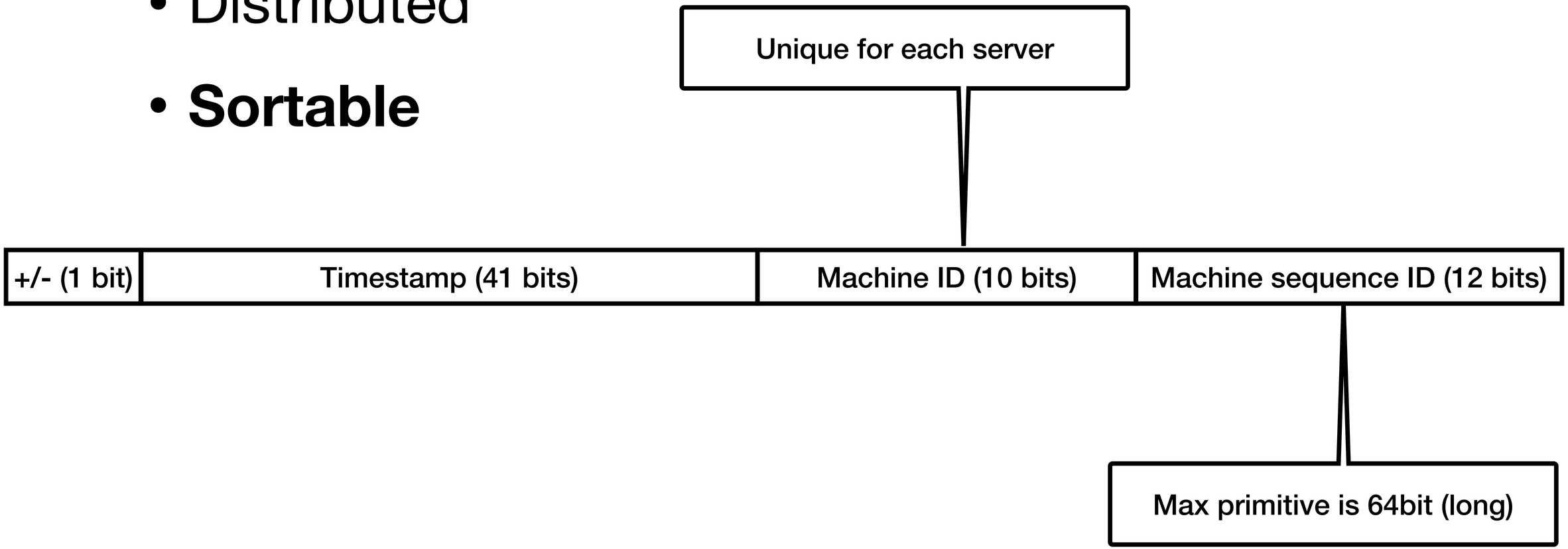
- Not trivial for distributed systems
- UUID / TIMEUUID are great
  - Downside - requires 128bit

    what's the problem with java primitives?



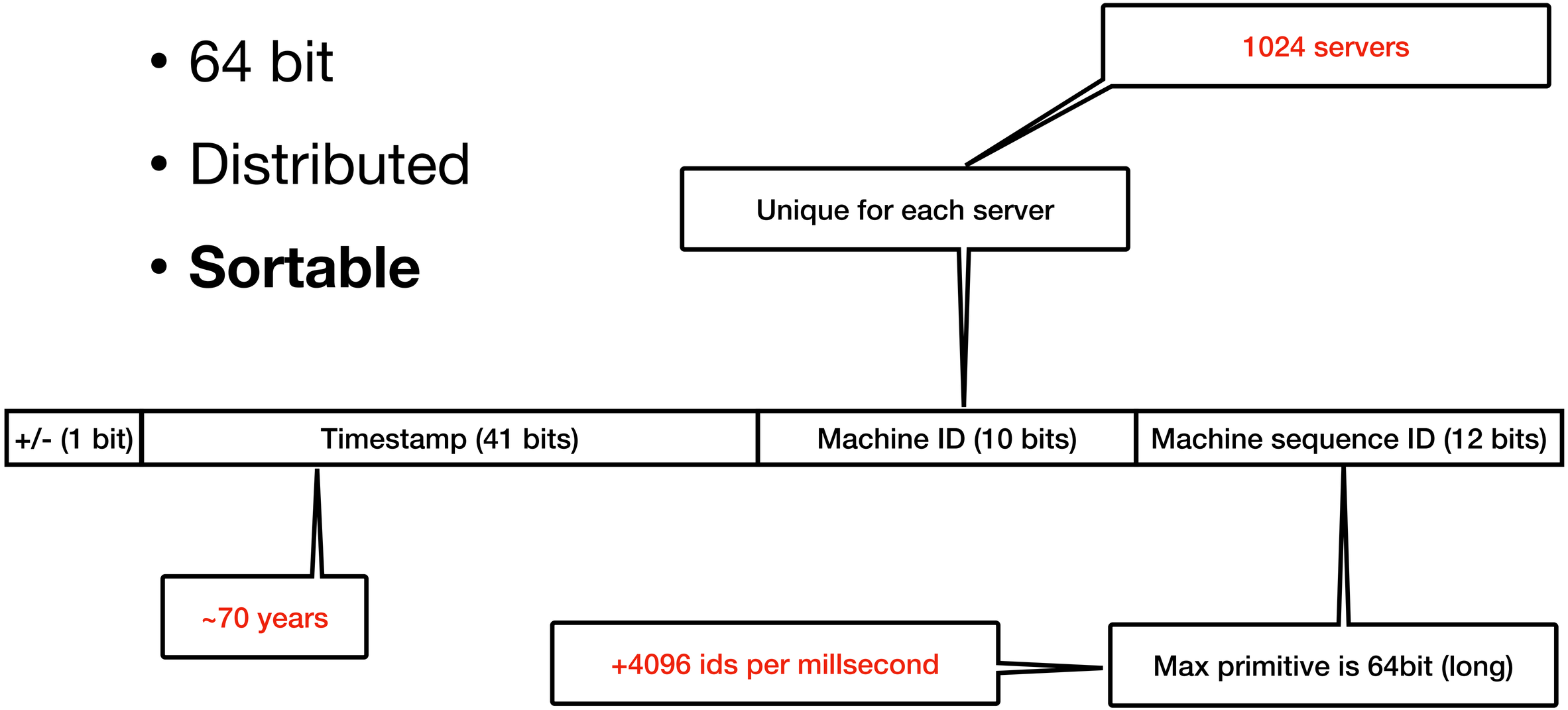
Twitter's Snowflake ID

- 64 bit
- Distributed
- **Sortable**



Twitter's Snowflake ID

- 64 bit
- Distributed
- **Sortable**



More data types

- `COUNTER`
- `LIST`
- `SET`
- `MAP`
- More on these later…

SELECT

```
SELECT * FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse
WHERE column1 = "1234" LIMIT 100

SELECT count(*) FROM BigDataCourse
```

- "Limited" compared to RDBMS
  sum / avg / min / max or only supported on new versions
  no joins / having / union…

SELECT

```
SELECT * FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse
WHERE column1 = "1234" LIMIT 100

SELECT count(*) FROM BigDataCourse
```

What will happen in this query?

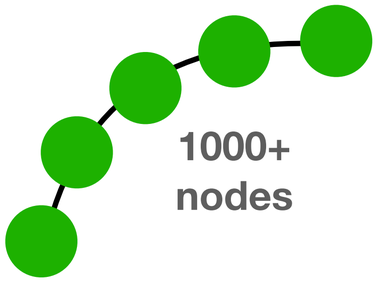
- "Limited" compared to RDBMS
  sum / avg / min / max or only supported on new versions
  no joins / having / union…

SELECT

```
SELECT * FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse
WHERE column1 = "1234" LIMIT 100

SELECT count(*) FROM BigDataCourse
```

ANTI PATTERN - but valid
Can be very slow and expensive - when?

- "Limited" compared to RDBMS
  sum / avg / min / max or only supported on new versions
  no joins / having / union…



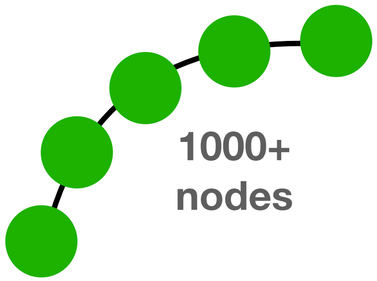
SELECT

```
SELECT * FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse

SELECT column1,column2 FROM BigDataCourse
WHERE column1 = "1234" LIMIT 1

SELECT count(*) FROM BigDataCourse
```

Even if counting a single row, it can be expensive (on a really big wide row)

ANTI PATTERN - but valid
Can be very slow and expensive - when?

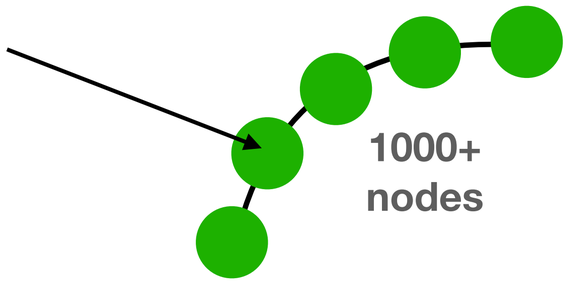
- "Limited" compared to RDBMS
  sum / avg / min / max or only supported on new versions
  no joins / having / union…



SELECT - partitions and keys

- TLDR; **provide the partition key to the query**

```
SELECT * FROM users
WHERE user_id = “1234”
```

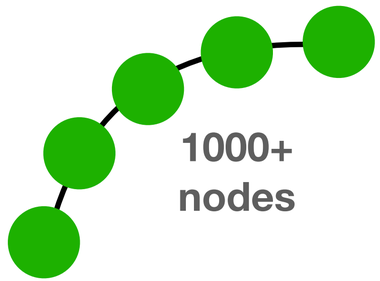


users	
user_id	K
name	
birth_year	
…	

SELECT - partitions and keys

- What happens if no partition is given?

```
SELECT * FROM users
```



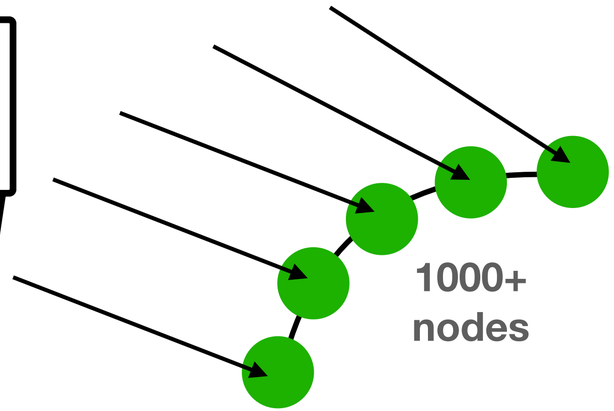
users	
user_id	K
name	
birth_year	
…	

SELECT - partitions and keys

- What happens if no partition is given?

```
SELECT * FROM users
```

We need to contact all servers (as all partitions are valid)

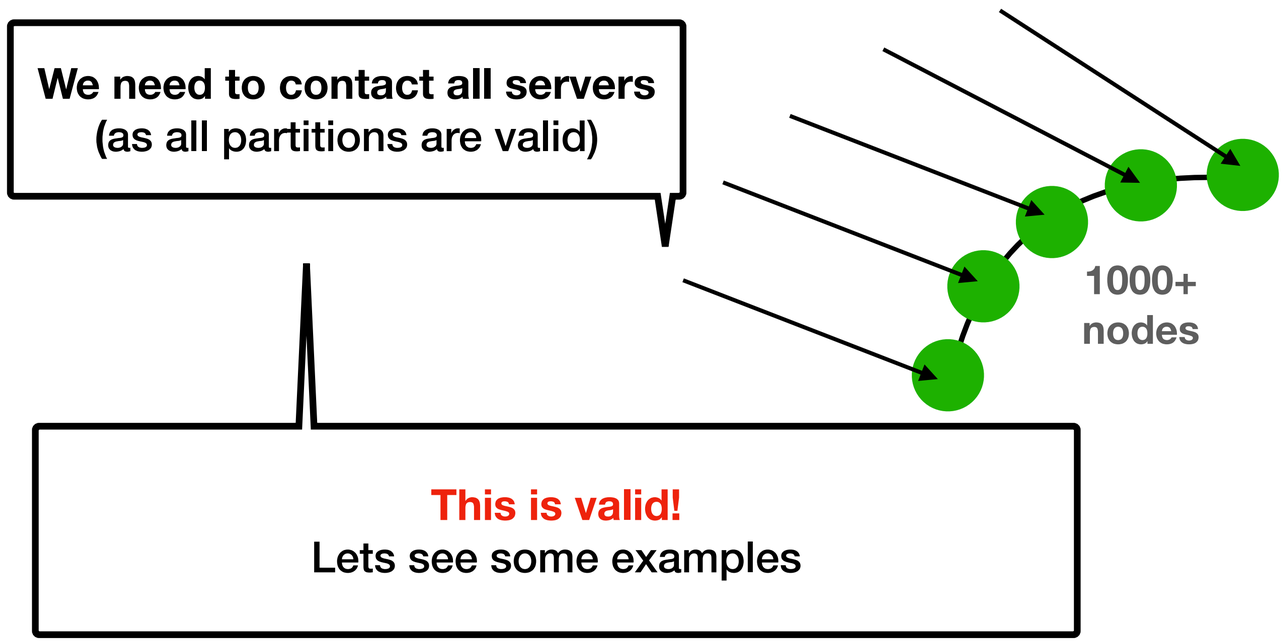


users	
user_id	K
name	
birth_year	
…	

SELECT - partitions and keys

- What happens if no partition is given?

```
SELECT * FROM users
```



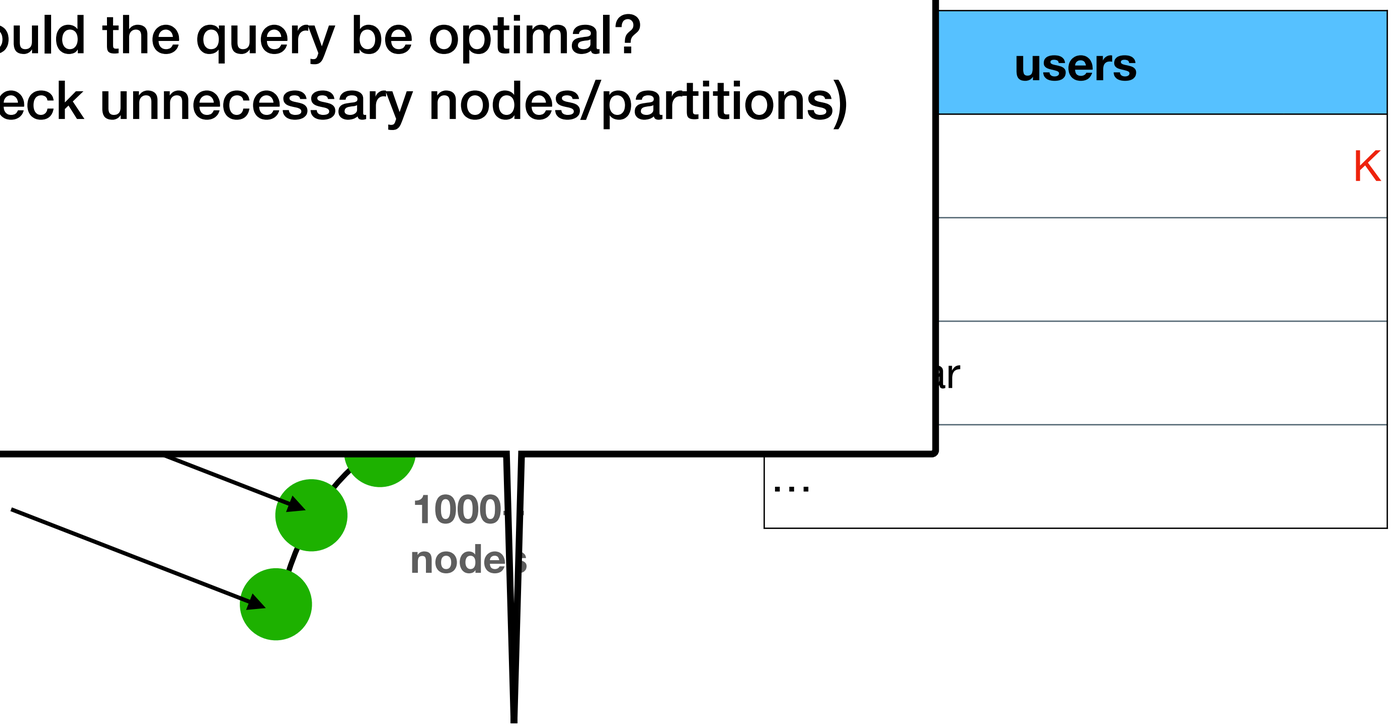
users	
user_id	K
name	
birth_year	
…	

SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

Assume there are **10k nodes** in the cluster and **no replication**

- If there are **100k users**, would the query be optimal?
  (that is, we would not check unnecessary nodes/partitions)

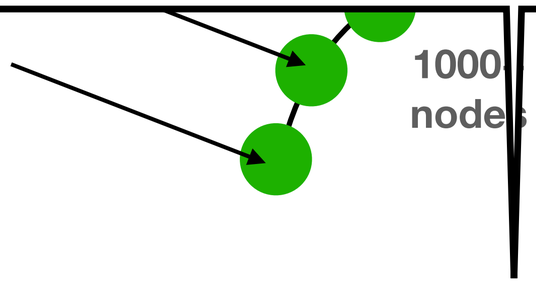


SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

Assume there are **10k nodes** in the cluster and **no replication**
- If there are **100k users**, would the query be optimal?
(that is, we would not check unnecessary nodes/partitions)

YES - why?



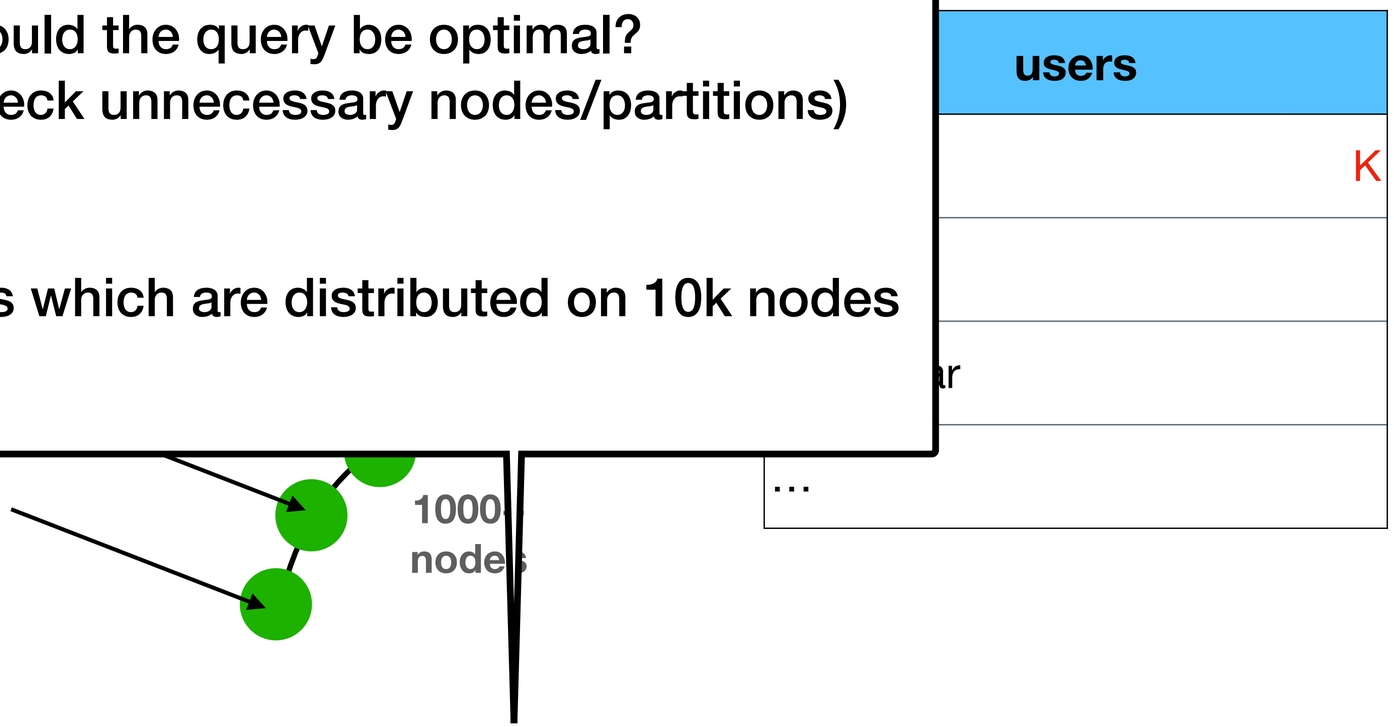
users
K
ar
…

SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

Assume there are **10k nodes** in the cluster and **no replication**
- If there are **100k users**, would the query be optimal?
(that is, we would not check unnecessary nodes/partitions)

YES - why?
There are 100k partitions which are distributed on 10k nodes

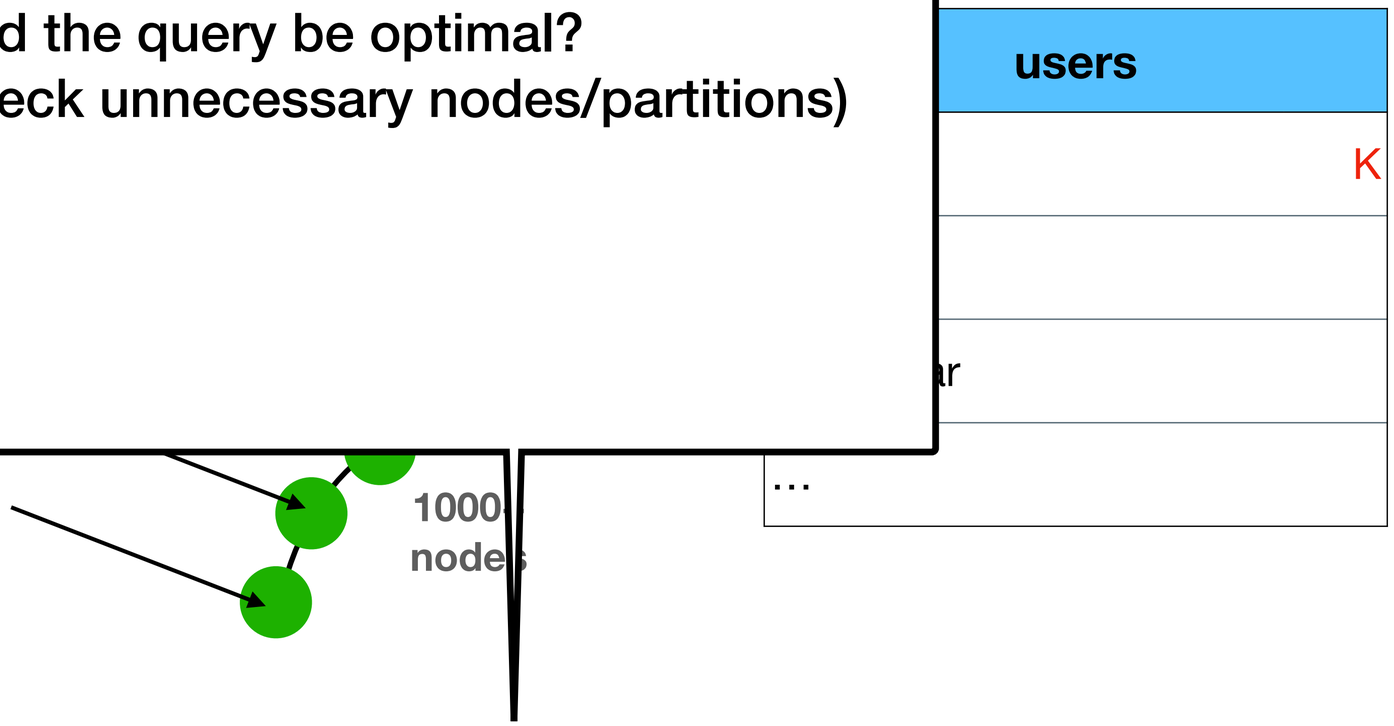


SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

Assume there are **10k nodes** in the cluster and **no replication**

- If there are **10 users**, would the query be optimal?
  (that is, we would not check unnecessary nodes/partitions)

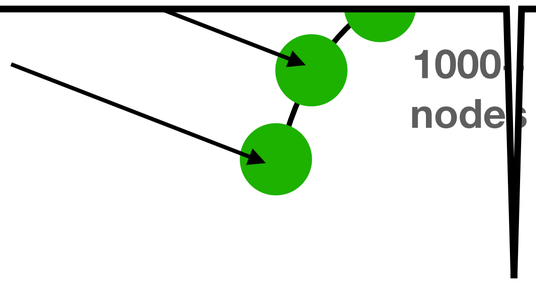


SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

Assume there are **10k nodes** in the cluster and **no replication**
- If there are **10 users**, would the query be optimal?
(that is, we would not check unnecessary nodes/partitions)

NO - why?



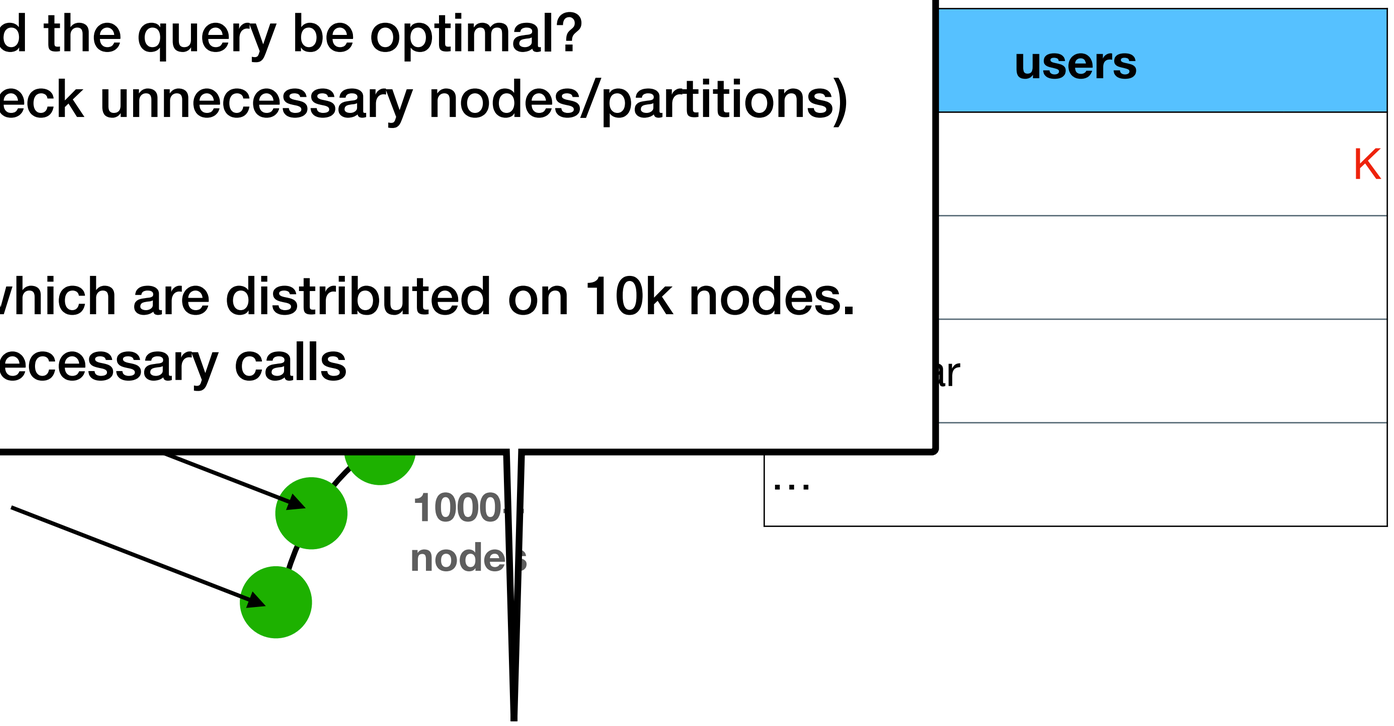
users
K
ar
…

SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

Assume there are **10k nodes** in the cluster and **no replication**
- If there are **10 users**, would the query be optimal?
(that is, we would not check unnecessary nodes/partitions)

NO - why?
There are 10 partitions which are distributed on 10k nodes.
We will initiate 9990 unnecessary calls

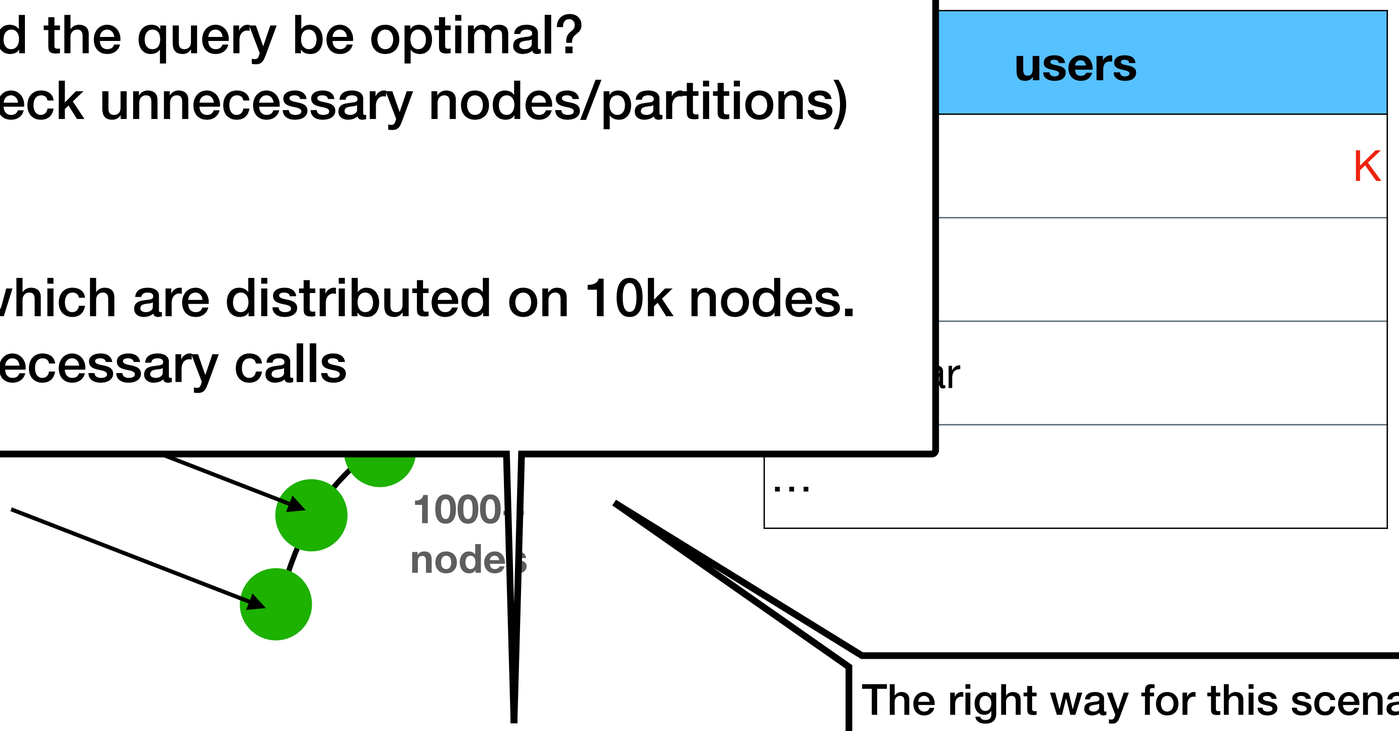


SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

Assume there are **10k nodes** in the cluster and **no replication**
- If there are **10 users**, would the query be optimal?
(that is, we would not check unnecessary nodes/partitions)

NO - why?
There are 10 partitions which are distributed on 10k nodes.
We will initiate 9990 unnecessary calls



The right way for this scenario is to create a single partition for these 10 users, then read 1 partition

SELECT - partitions and keys

Each user “creates” a partition (user_id is partition_key)

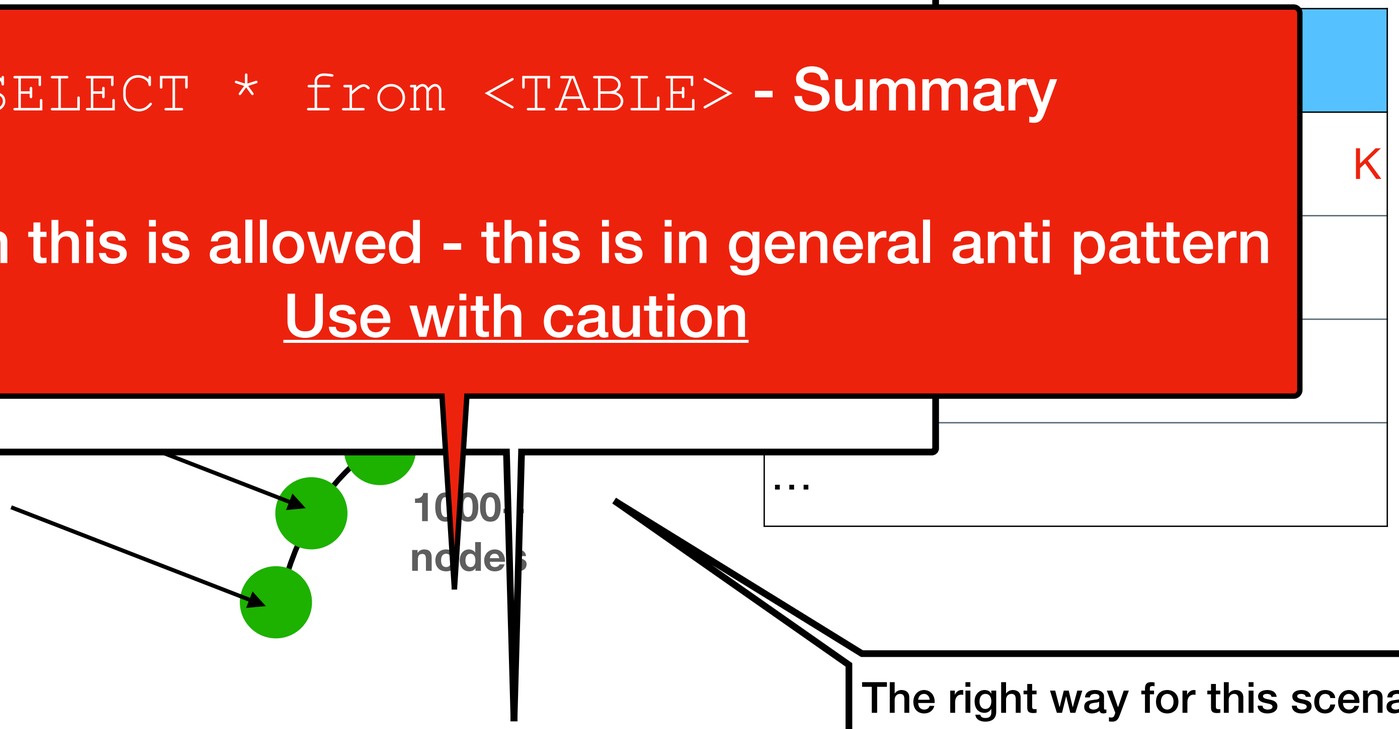
Assume there are 10k nodes in the cluster and no replication

- If there are 10 u
  (that is, we wo

NO - why?
The there are
nodes. We wi

`SELECT * from <TABLE>` - Summary

Although this is allowed - this is in general anti pattern
Use with caution

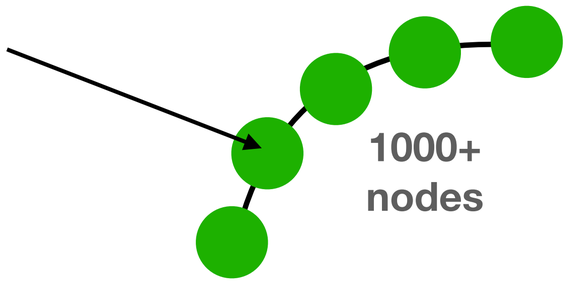


The right way for this scenario is to create a single partition for these 10 users, then read 1 partition

SELECT - partitions and keys

- Try a different model

```
SELECT * FROM users
WHERE country = “israel”
```



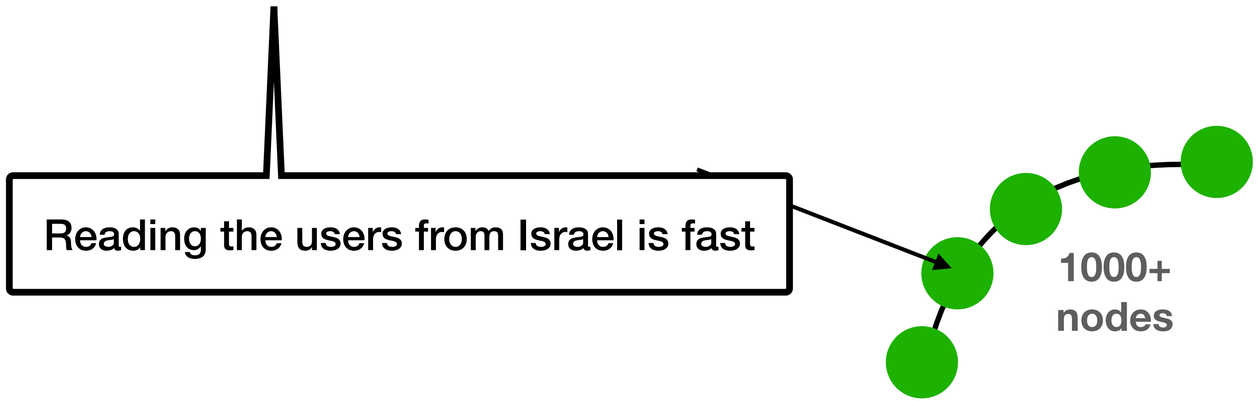
users	
country	K
user_id	▼C
name	
birth_year	
…	

Note
K is the partition key (NOT the key)
▼C is the clustering column,
Together both are the key

SELECT - partitions and keys

- Try a different model

```
SELECT * FROM users
WHERE country = “israel”
```

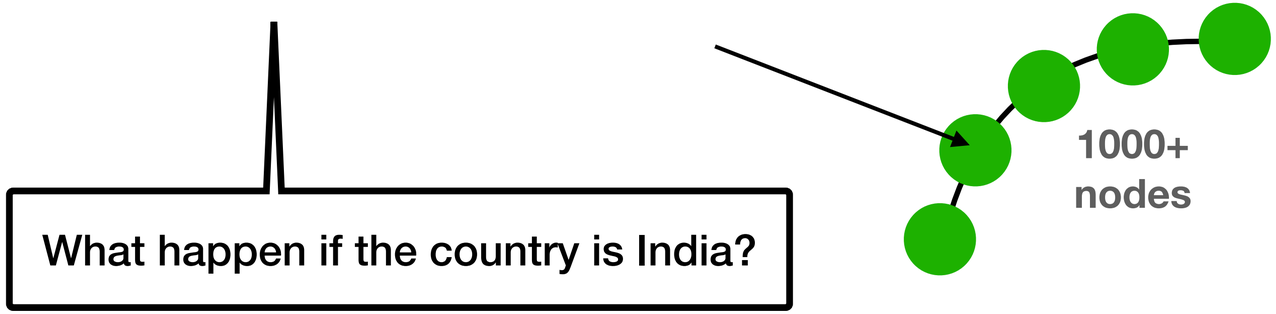


users	
country	K
user_id	▼C
name	
birth_year	
…	

SELECT - partitions and keys

- Try a different model

```
SELECT * FROM users
WHERE country = "israel"
```

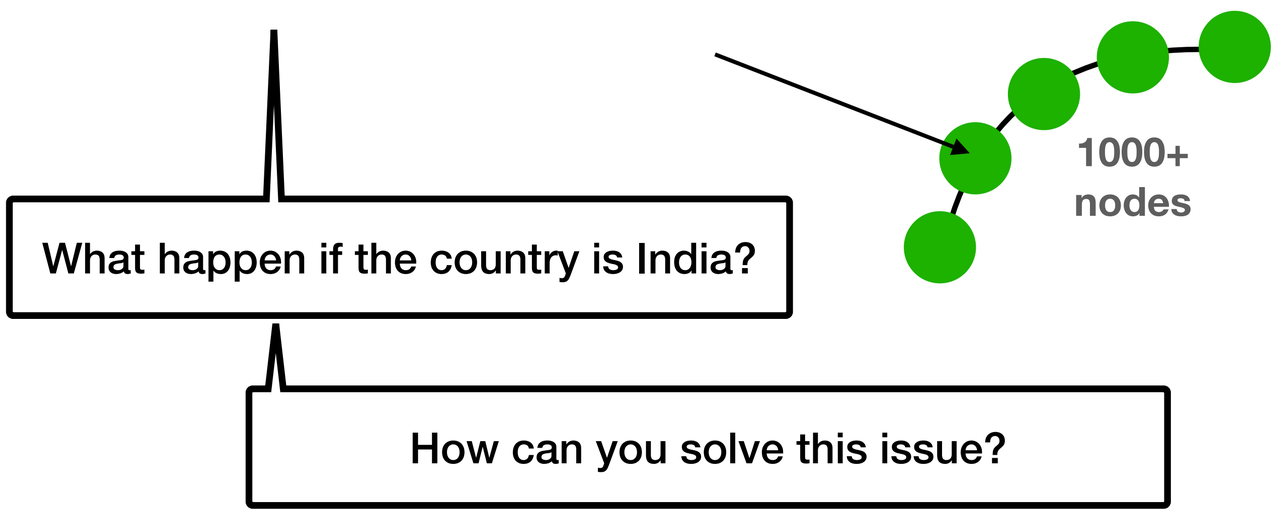


users	
country	K
user_id	▼C
name	
birth_year	
…	

SELECT - partitions and keys

- Try a different model

```
SELECT * FROM users
WHERE country = "israel"
```



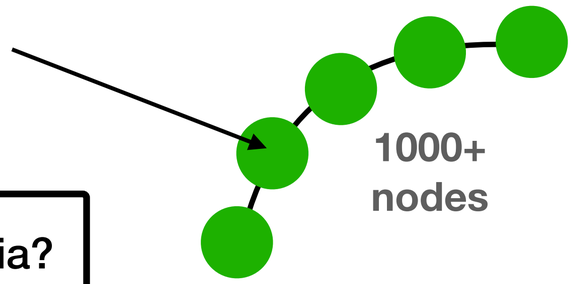
users	
country	K
user_id	▼C
name	
birth_year	
…	

SELECT - partitions and keys

- Try a different model

```
SELECT * FROM users
WHERE country = “israel”
```

users	
country	K
user_id	▼C
name	
birth_year	
…	



What happen if the country is India?

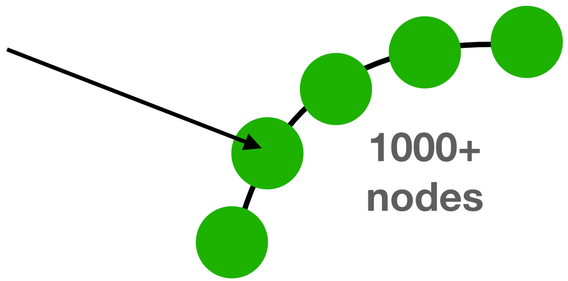
How can you solve this issue?

We can add “buckets” - more on this later

SELECT - partitions and keys

- What happens now?

```
SELECT * FROM users
WHERE country = “israel”
AND birth_year = 1982
```

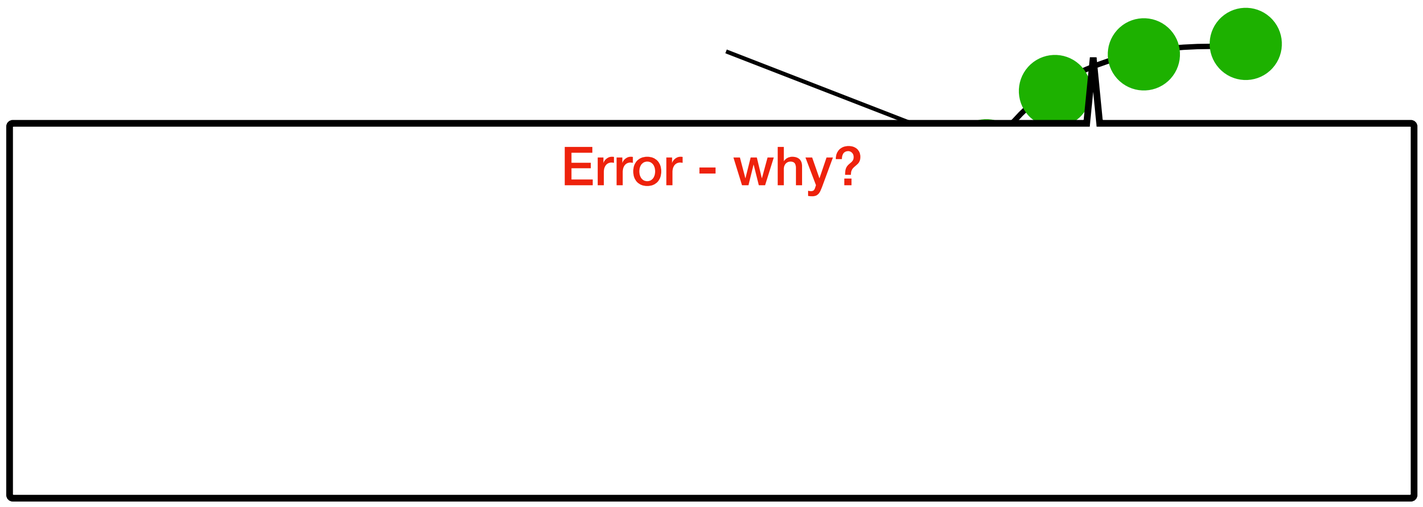


users	
country	K
user_id	▼C
name	
birth_year	
…	

SELECT - partitions and keys

- What happens now?

```
SELECT * FROM users
WHERE country = “israel”
AND birth_year = 1982
```



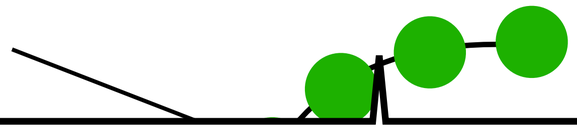
users	
country	K
user_id	▼C
name	
birth_year	
…	

SELECT - partitions and keys

- What happens now?

```
SELECT * FROM users
WHERE country = “israel”
AND birth_year = 1982
```

users	
country	K
user_id	▼C
name	
birth_year	
…	



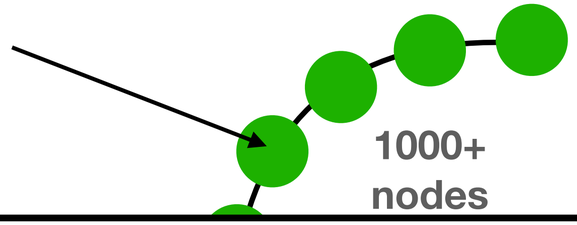
Error - why?

Cassandra will need to read the entire partition.
If there are 1m users, and only 10k were born in 1982,
there would be an unnecessary read/filter of 990k users

SELECT - partitions and keys

- What happens now?

```
SELECT * FROM users
WHERE country = “israel”
AND birth_year = 1982
ALLOW FILTERING
```



users	
country	K
user_id	▼C
name	
birth_year	

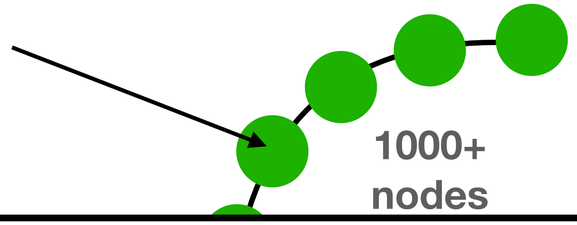
With “ALLOW FILTERING” Cassandra will approve the query (ANTI PATTERN)

SELECT - partitions and keys

- What happens now?

How can you support the query without “ALLOW FILTERING”?

```
SELECT * FROM users
WHERE country = “israel”
AND birth_year = 1982
ALLOW FILTERING
```



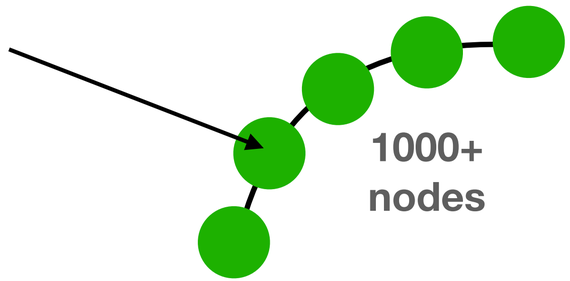
users	
country	K
user_id	▼C
name	
birth_year	

With “ALLOW FILTERING” Cassandra will approve the query (ANTI PATTERN)

SELECT - partitions and keys

- Solved with denormalization

```
SELECT * FROM users_by_birth_year
WHERE country = “israel”
AND birth_year = 1982
```



- (we will talk about correct modeling later)

users	
country	K
user_id	▼C
name	
birth_year	
...	

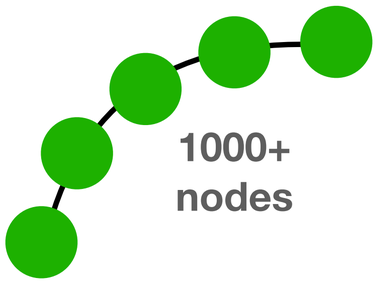
users_by_birth_year	
country	K
birth_year	▼C
user_id	▼C
name	
...	

SELECT - partitions and keys

- And what about this case?

```
SELECT * FROM users
WHERE city = “tel aviv”
```

users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	



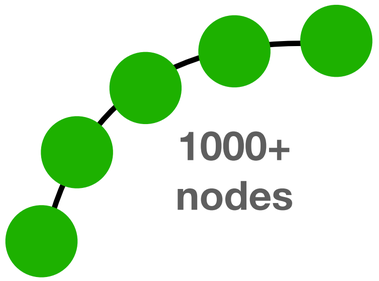
SELECT - partitions and keys

- And what about this case?

```
SELECT * FROM users
WHERE city = "tel aviv"
```

users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	

Error - why?



SELECT - partitions and keys

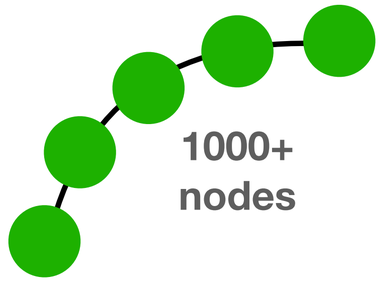
- And what about this case?

```
SELECT * FROM users
WHERE city = “tel aviv”
```

users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	

Error - why?

Cassandra will need to contact all nodes and to check if such partition exists



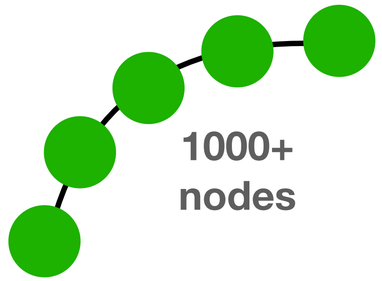
SELECT - partitions and keys

- And what about this case?

```
SELECT * FROM users
WHERE city = “tel aviv”
ALLOW FILTERING
```

users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	

With “ALLOW FILTERING” Cassandra will approve the query
(again - ANTI PATTERN)

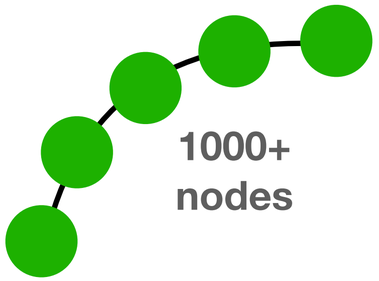


SELECT - ALLOW FILTERING

- Almost always ANTI PATTERN

- We saw these use cases
  - To “filter” columns in a single partition
  - To “filter” partitions across nodes

users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	

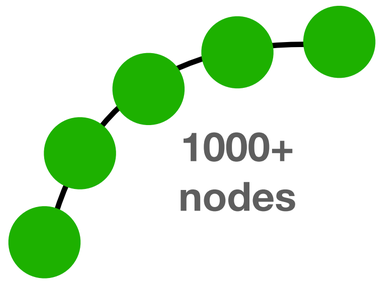


SELECT - ALLOW FILTERING

- Almost always ANTI PATTERN

- We saw these use cases
  - To “filter” columns in a single partition
  - To “filter” partitions across nodes
  - Can you think of another example?

users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	



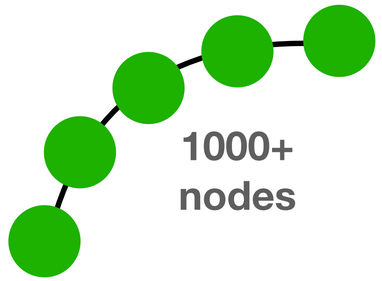
SELECT - ALLOW FILTERING

- Almost always ANTI PATTERN

```
SELECT * FROM users
WHERE name = "rubi boim"
ALLOW FILTERING
```

- We sav
  - To “filter” columns in a single partition
  - To “filter” partitions across nodes
  - To “filter” columns across partitions

users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	



INSERT

- Primary key is obviously required

```
INSERT INTO BigDataCourse(column1,column2)
VALUES (123,”name”)
```

INSERT - IF NOT EXISTS

- Requires read before write!
- Use with caution

```
INSERT INTO BigDataCourse(column1,column2)
IF NOT EXSITS
VALUES (123,"name")
```

INSERT - IF NOT EXISTS

- Requires read before write!
- Use with caution

```
INSERT INTO BigDataCourse(column1,column2)
IF NOT EXSITS
VALUES (123,”name”)
```

Note - **writes are cheaper than reads.** If there are not too many writes, it is better to overwrite the same data instead of using “if not exists”

INSERT - USING TTL

- Time To Live - allows for automatic expiration (delete)
  in seconds

```
INSERT INTO BigDataCourse(column1,column2)
VALUES (123,”name”)
USING TTL 86400     // 24 hours
```

INSERT - USING TTL

- Time To Live - allows for automatic expiration (delete)
  in seconds

```
INSERT INTO BigDataCourse(column1,column2)
VALUES (123,”name”)
USING TTL 86400     // 24 hours
```

Creates tombstones
more on this later

UPDATE

- Primary key is obviously required

```
UPDATE BigDataCourse
SET column2 = “name”, column3 = “abc”
WHERE column1 = 123
```

DELETE

- Warning:
  DELETEs in distributed databases are NOT TRIVIAL
- In Cassandra in particular
- Deleted data is not removed immediately
  a tombstone is created
- More on this later

DELETE

- Delete data from a row

```
DELETE name FROM users
WHERE country = “israel”
AND user_id = “123”
```

- Delete an entire row

```
DELETE FROM users
WHERE country = “israel”
```

users	
country	K
user_id	▼C
name	
birth_year	
…	

DELETE

Creates 1 tombstone

- Delete data from a row

```
DELETE name FROM users
WHERE country = “israel”
AND user_id = “123”
```

- Delete an entire row

```
DELETE FROM users
WHERE country = “israel”
```

Creates 1 tombstone

users	
country	K
user_id	▼C
name	
birth_year	
…	

Truncate

- Removes all SSTables holding data
- Use with care
- (Avoids tombstones)

```
TRUNCATE users
```

ALTER TABLE

- Add / drop / rename existing columns
- *change datatypes (with restrictions)
- Change table properties
- Can NOT alter PRIMARY KEY columns
- RTFM :)

```
ALTER TABLE [keyspace_name.] table_name
[ALTER column_name TYPE cql_type]
[ADD (column_definition_list)]
[DROP column_list | COMPACT STORAGE ]
[RENAME column_name TO column_name]
[WITH table_properties];
```