

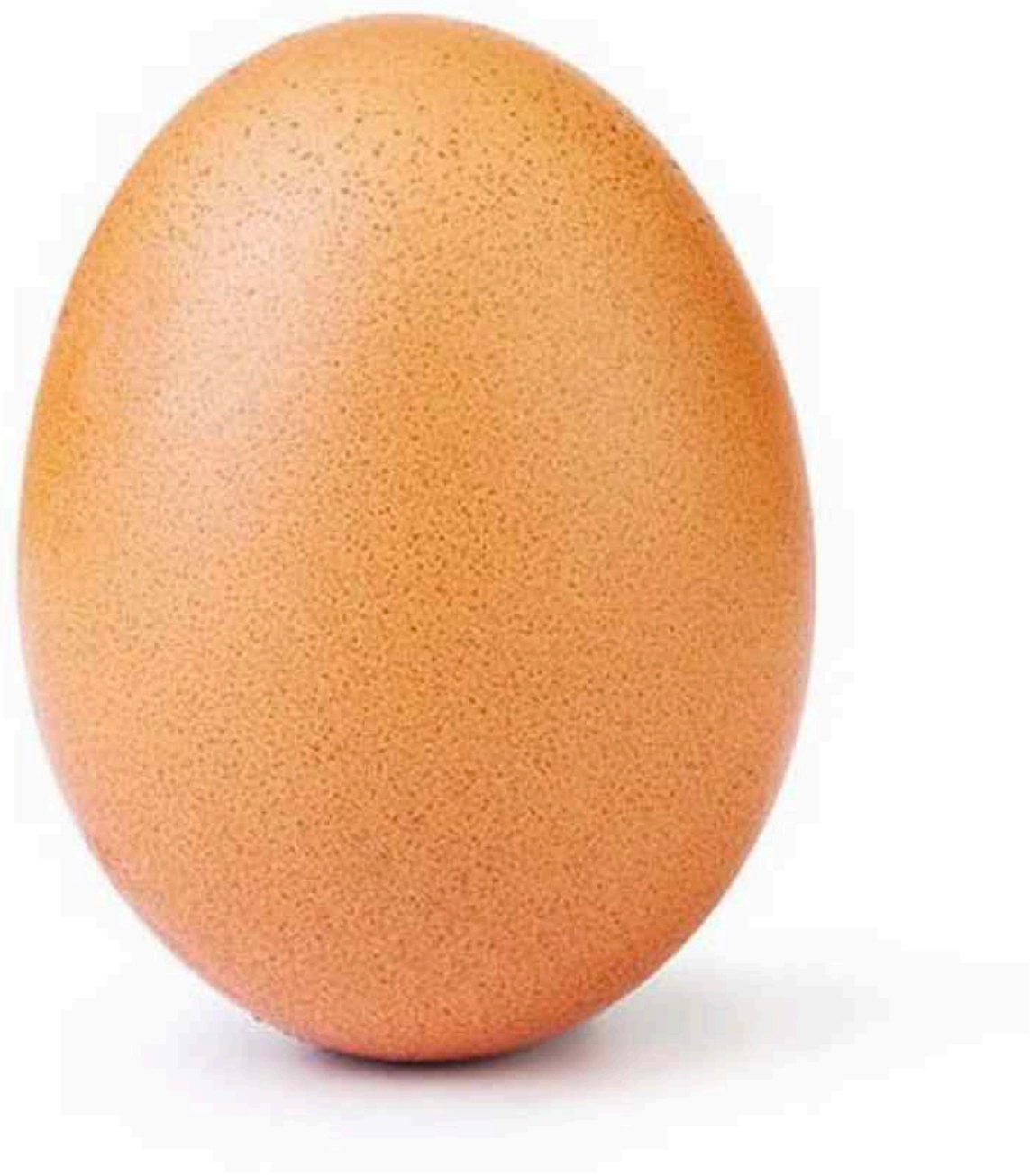
Cassandra - CQL

Big Data Systems

Dr. Rubi Boim



world_record_egg  · [Follow](#)

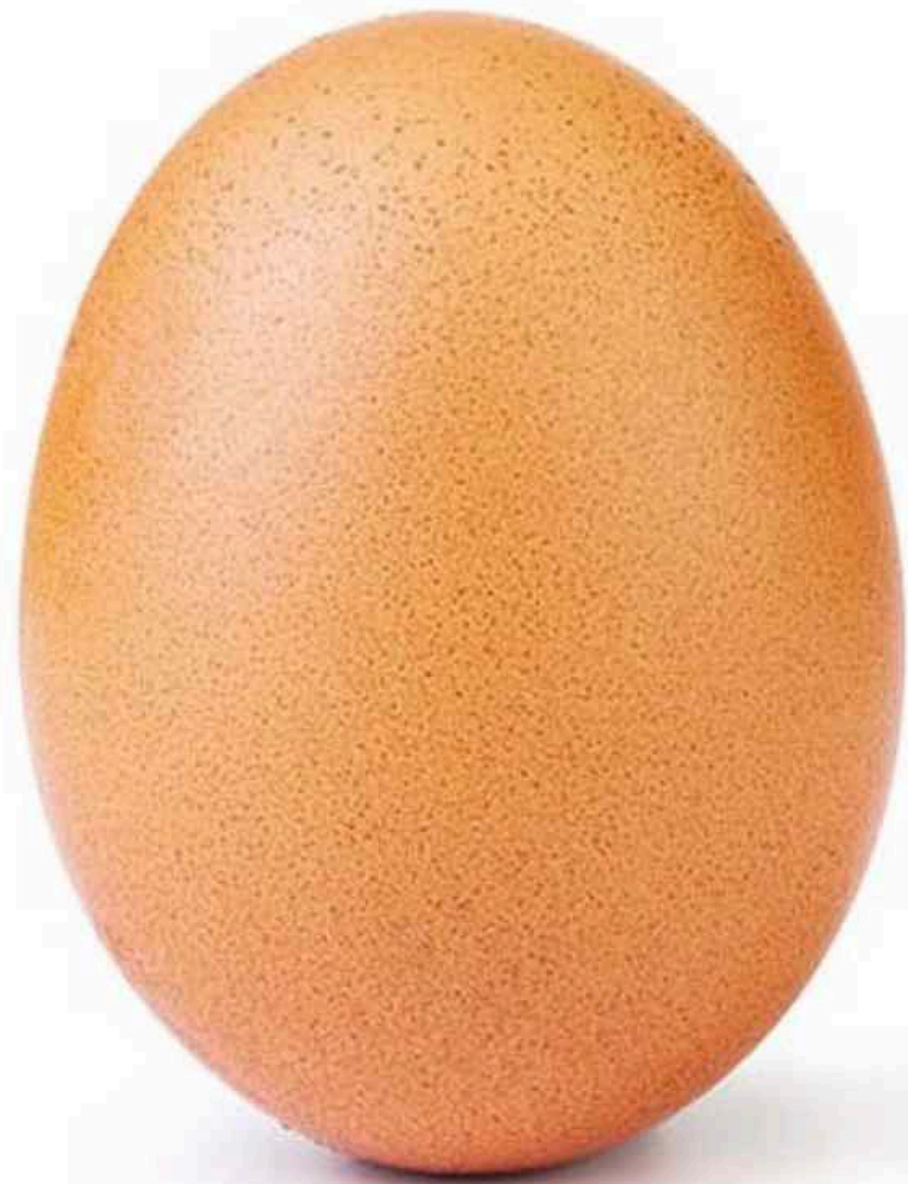


55,957,347 likes

JANUARY 4, 2019



world_record_egg • Follow



55,957,347 likes

JANUARY 4, 2019



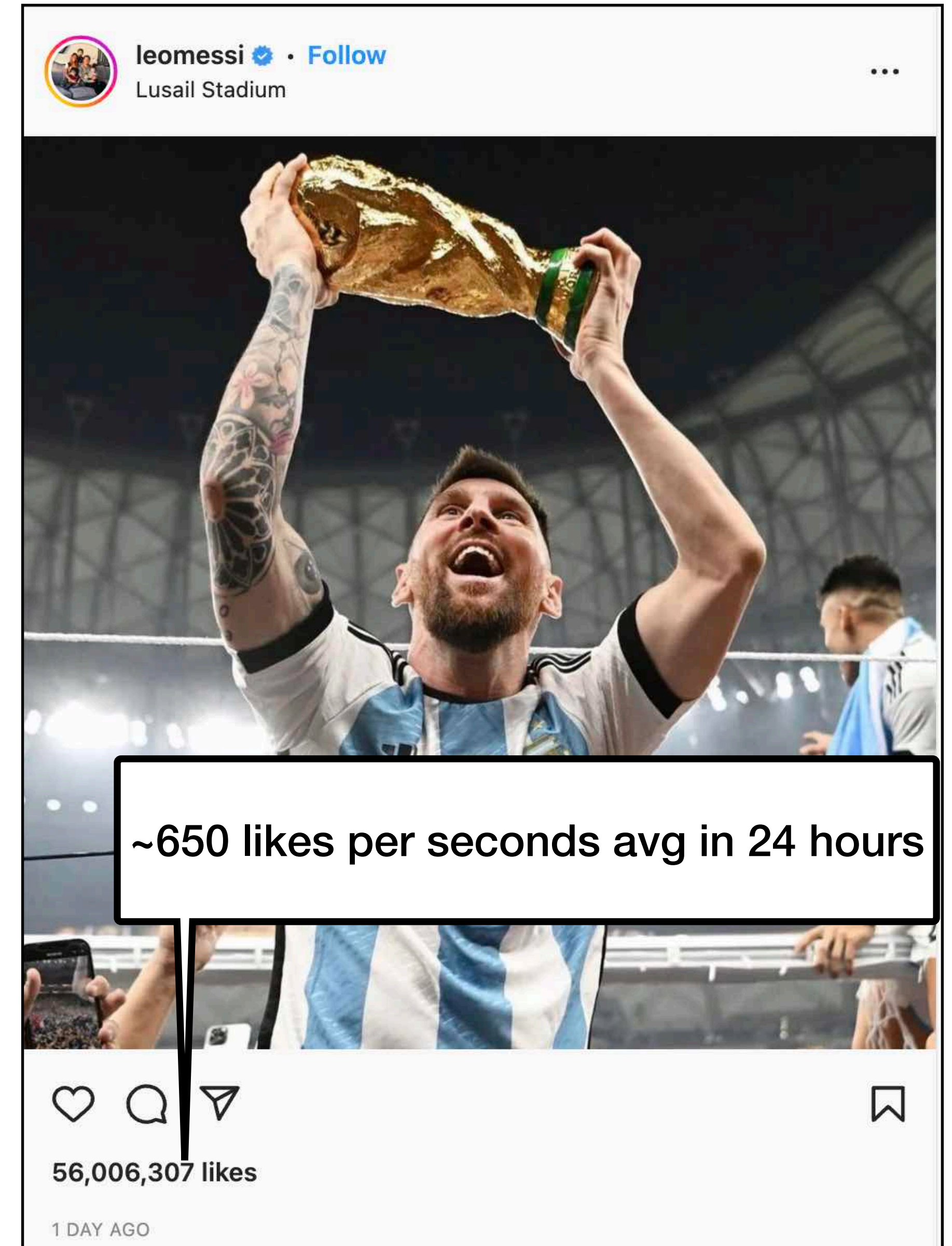
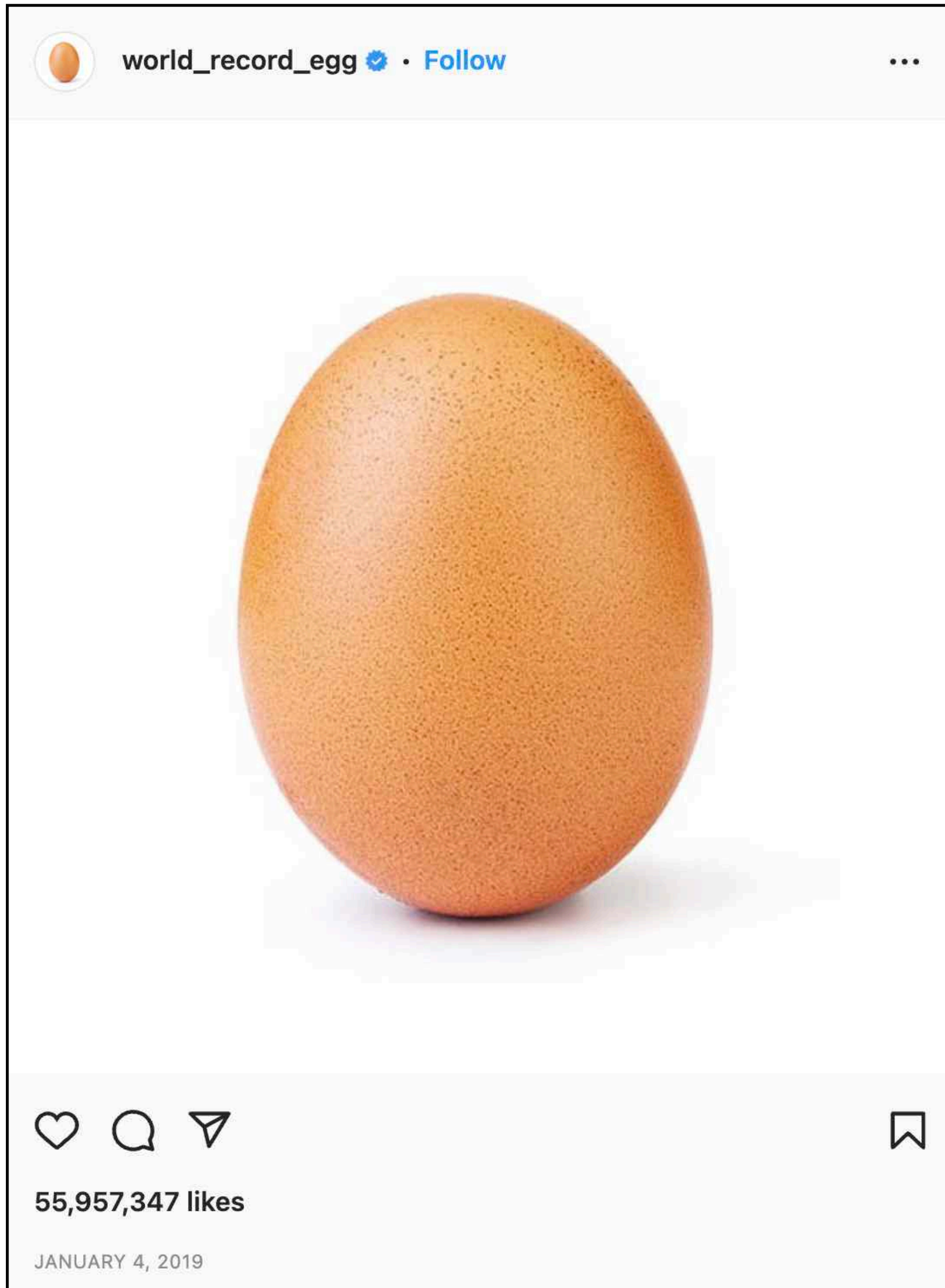
leomessi • Follow

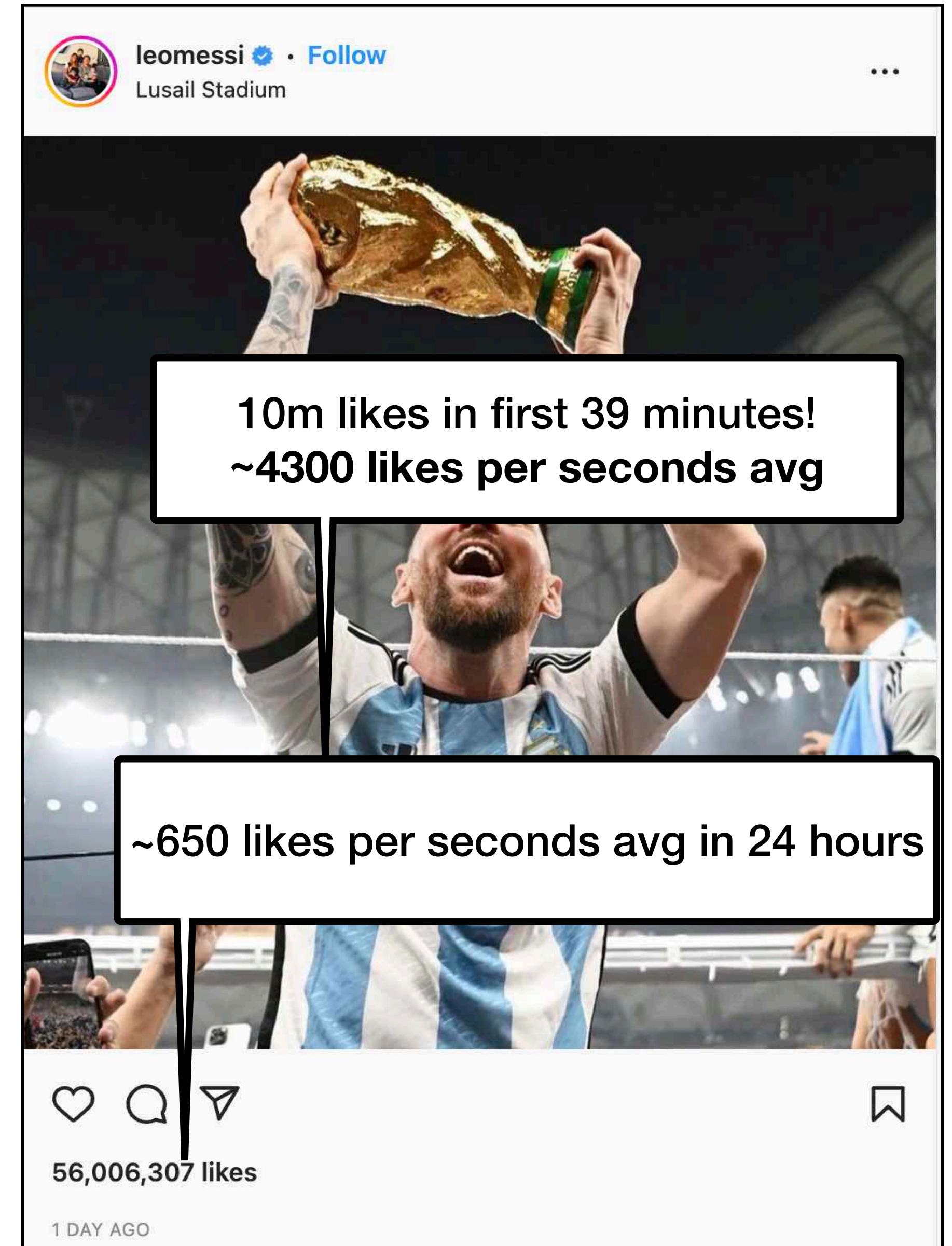
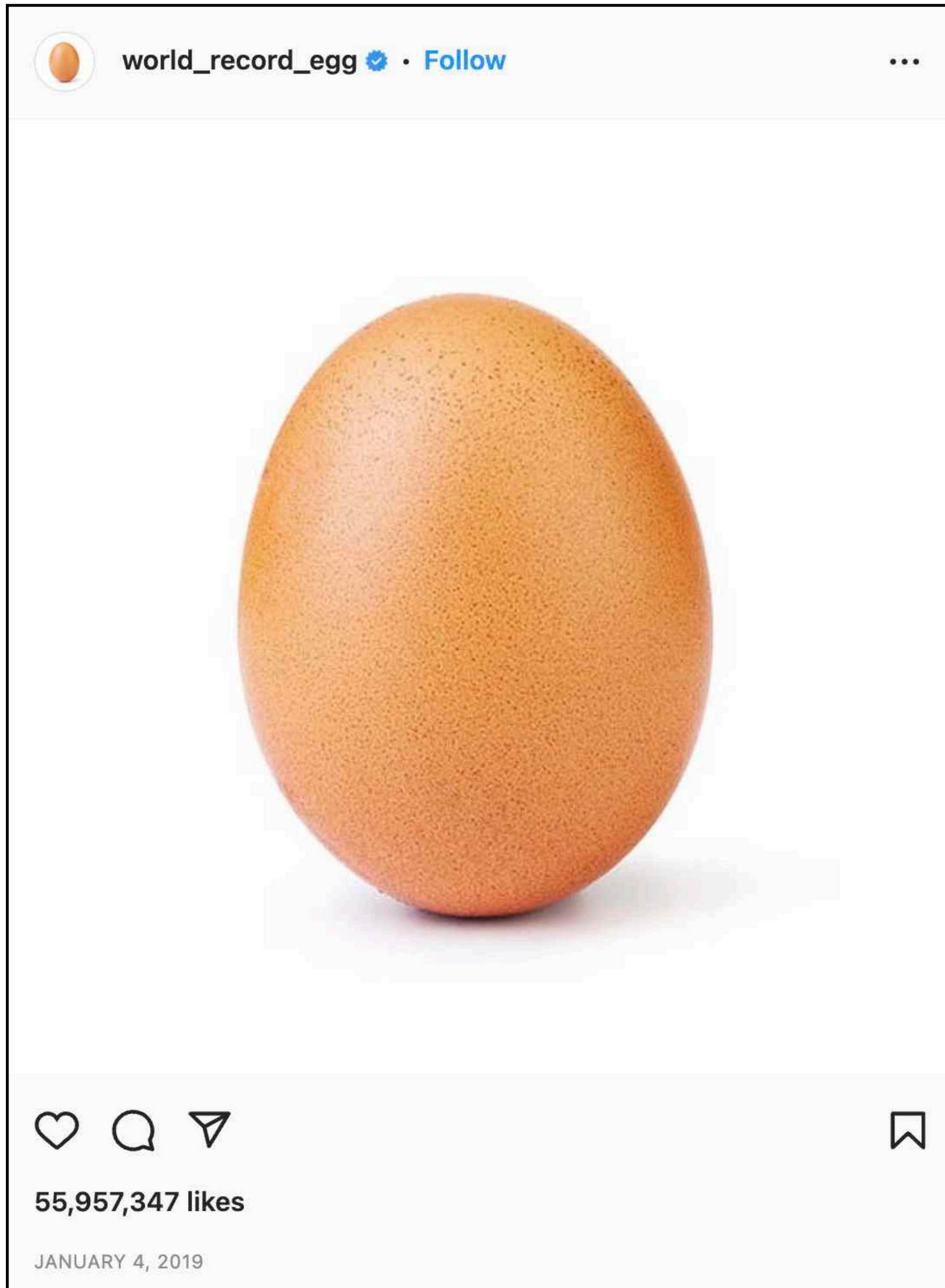
Lusail Stadium



56,006,307 likes

1 DAY AGO






 jenniferaniston • Follow




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




9m likes in first 24 hours
~100 likes per seconds avg

16,458,620 likes

OCTOBER 15, 2019

This image shows a social media post by Jennifer Aniston. The main image is a candid shot of her with her family, including her husband and children, all smiling and looking towards the camera. The post has a white background with a black border. At the top left, there is a profile picture of Jennifer Aniston, her name 'jenniferaniston', a verified badge, and a 'Follow' button. At the top right, there are three dots. At the bottom left, there are icons for likes, comments, and shares. At the bottom right, there is a bookmark icon. The text '9m likes in first 24 hours ~100 likes per seconds avg' is overlaid in a white box with a black border. Below the image, the text '16,458,620 likes' and 'OCTOBER 15, 2019' are visible.

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

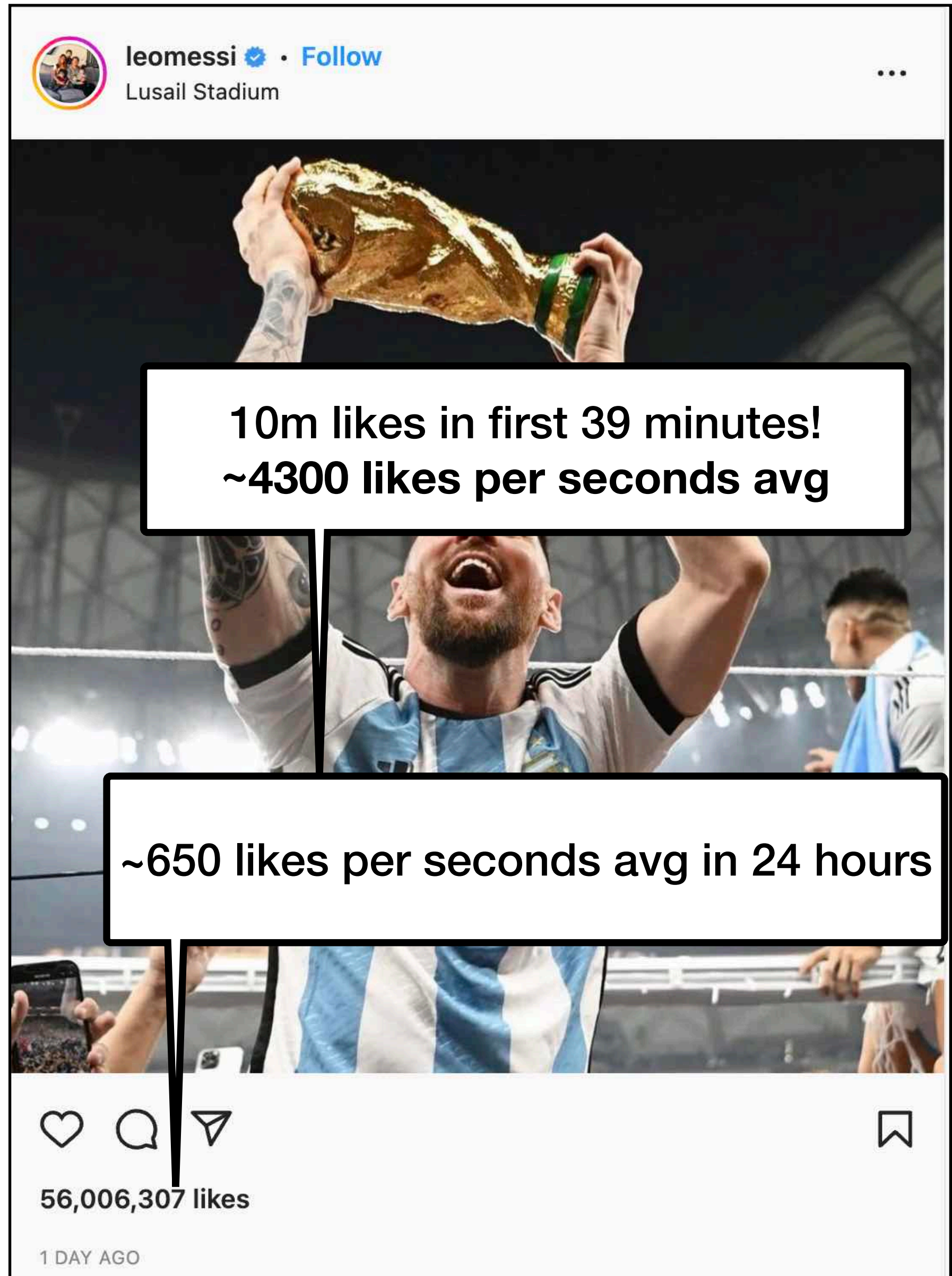
1 DAY AGO

This image shows a social media post by Lionel Messi. The main image is a photo of him in an Argentina jersey, celebrating and holding up a large golden trophy. The post has a white background with a black border. At the top left, there is a profile picture of Lionel Messi, his name 'leomessi', a verified badge, and a 'Follow' button. Below the name is the location 'Lusail Stadium'. At the top right, there are three dots. At the bottom left, there are icons for likes, comments, and shares. At the bottom right, there is a bookmark icon. The text '10m likes in first 39 minutes! ~4300 likes per seconds avg' is overlaid in a white box with a black border. Below the image, the text '~650 likes per seconds avg in 24 hours' is overlaid in another white box with a black border. At the bottom, the text '56,006,307 likes' and '1 DAY AGO' are visible.



So why did Instagram crashed?

9m likes in first 24 hours
~100 likes per seconds avg



10m likes in first 39 minutes!
~4300 likes per seconds avg

~650 likes per seconds avg in 24 hours

Top 20 posts

Three accounts have more than one of the most-liked posts in the top 20: [Lionel Messi](#) has eight, [Cristiano Ronaldo](#) has three and [Kylie Jenner](#) has two.

Rank ↕	Account name ↕	Owner ↕	Post description	Post	Likes (millions) ↕	Date posted (UTC) ↕
1	@leomessi	Lionel Messi	Photos of Lionel Messi and his teammates after winning the 2022 FIFA World Cup with Argentina	[1] ↗	57.1	December 18, 2022
2	@world_record_egg	Chris Godfrey	Photo of an egg	[2] ↗	56.1	January 4, 2019
3	@cristiano	Cristiano Ronaldo	Photo of Lionel Messi and Cristiano Ronaldo playing chess, advertising for Louis Vuitton	[3] ↗	42.1	November 19, 2022
4	@leomessi	Lionel Messi	Photo of Lionel Messi with the FIFA World Cup trophy	[4] ↗	34.2	December 19, 2022
5	@cristiano @georginagio	Cristiano Ronaldo Georgina Rodríguez	Their twins pregnancy announcement	[5] ↗	32.6	October 28, 2021
6	@leomessi	Lionel Messi	Photo of Lionel Messi and Cristiano Ronaldo playing chess, advertising for Louis Vuitton	[6] ↗	32.4	November 19, 2022
7	@cristiano	Cristiano Ronaldo	Photo after 2022 FIFA World Cup elimination	[7] ↗	31.2	December 11, 2022
8	@xxxtentacion	XXXTentacion	Final post before his death	[8] ↗	30.4	May 20, 2018
9	@leomessi	Lionel Messi	Photos after 2022 FIFA World Cup game against Croatia	[9] ↗	28.5	December 13, 2022
10	@arianagrande	Ariana Grande	Photos from her wedding with Dalton Gomez	[10] ↗	26.3	May 26, 2021
11	@zendaya	Zendaya	Happy birthday post to Tom Holland	[11] ↗	25.7	June 1, 2022
12	@kyliejenner	Kylie Jenner	Second pregnancy announcement	[12] ↗	24.8	September 8, 2021
13	@tomholland2013	Tom Holland	Recreating the famous Spider-Man meme <small>Printable version of this page [ctrl-option-p]</small> rfield	[13] ↗	24.8	February 23, 2022
14	@leomessi	Lionel Messi	Photo after winning the 2022 FIFA World Cup game against Mexico	[14] ↗	24.2	November 26, 2022
15	@kyliejenner	Kylie Jenner	Photo of her daughter with her newborn brother	[15] ↗	22.9	February 6, 2022
16	@neymarjr	Neymar	Photo after 2022 FIFA World Cup elimination	[16] ↗	22.7	December 10, 2022
17	@billieeilish	Billie Eilish	Reveal of her blonde hair	[17] ↗	22.6	March 17, 2021
18	@leomessi	Lionel Messi	Photo after 2022 FIFA World Cup game against Netherlands	[18] ↗	22.3	December 9, 2022
19	@leomessi	Lionel Messi	First post after signing with PSG	[19] ↗	21.8	August 11, 2021
20	@leomessi	Lionel Messi	Photo with the Copa América trophy	[20] ↗	21.8	July 10, 2021

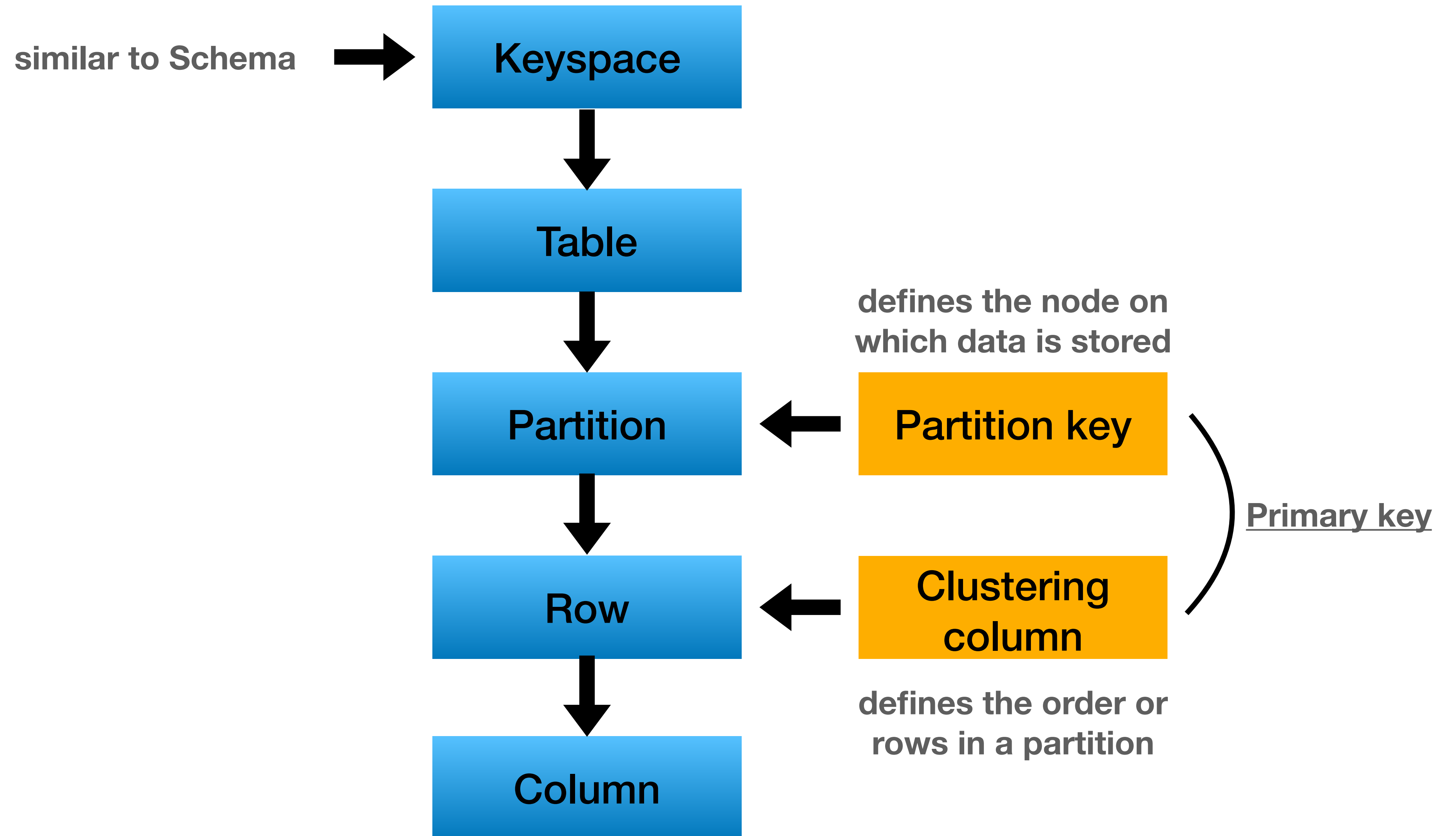
Cassandra CQL

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



Spoiler - most slides will be on SELECT

Terminology (Cassandra)



Keyspace

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

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 KEYSACES/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** universally 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?



Max primitive is 64bit (long)

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
```

ANTI PATTERN

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

Can be very slow and expensive - when?

- “Limited” compared to RDBMS

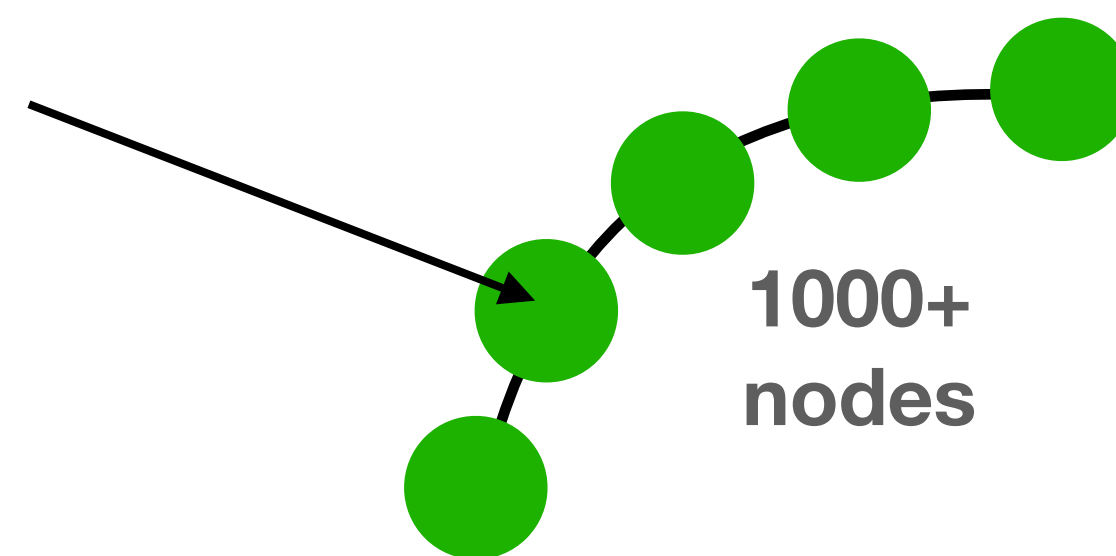
sum / avg / min / max or only supported on new versions

no joins / having / union...

SELECT - partitions and keys

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

```
SELECT * FROM users  
WHERE user_id = "1234"
```

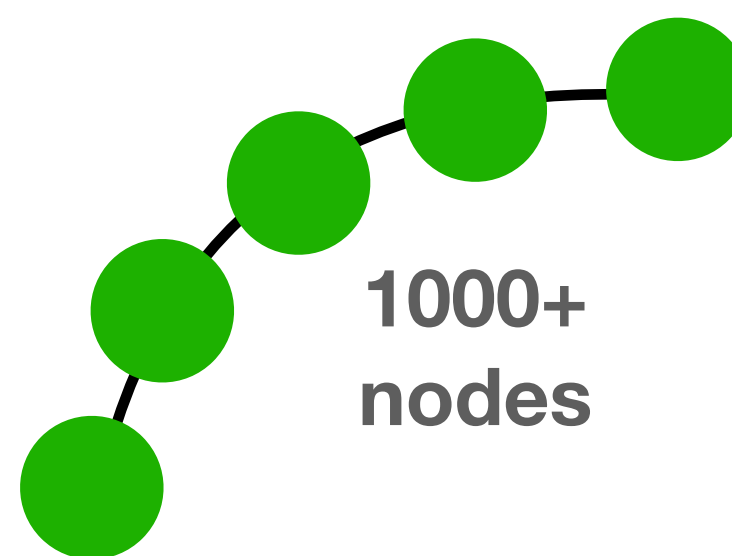


users	
user_id	K
name	
birth_year	
...	

SELECT - partitions and keys

- What happens if no partition is given?

```
SELECT * FROM users
```



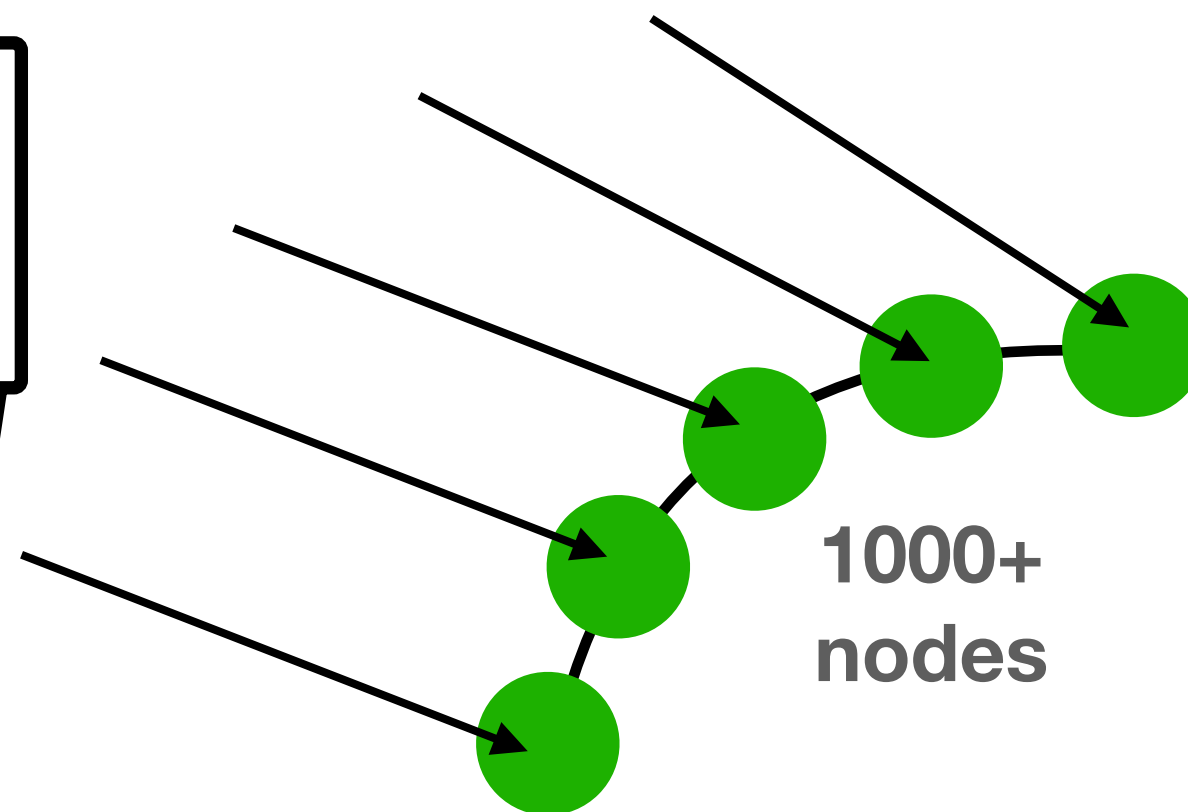
users	
user_id	K
name	
birth_year	
...	

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)



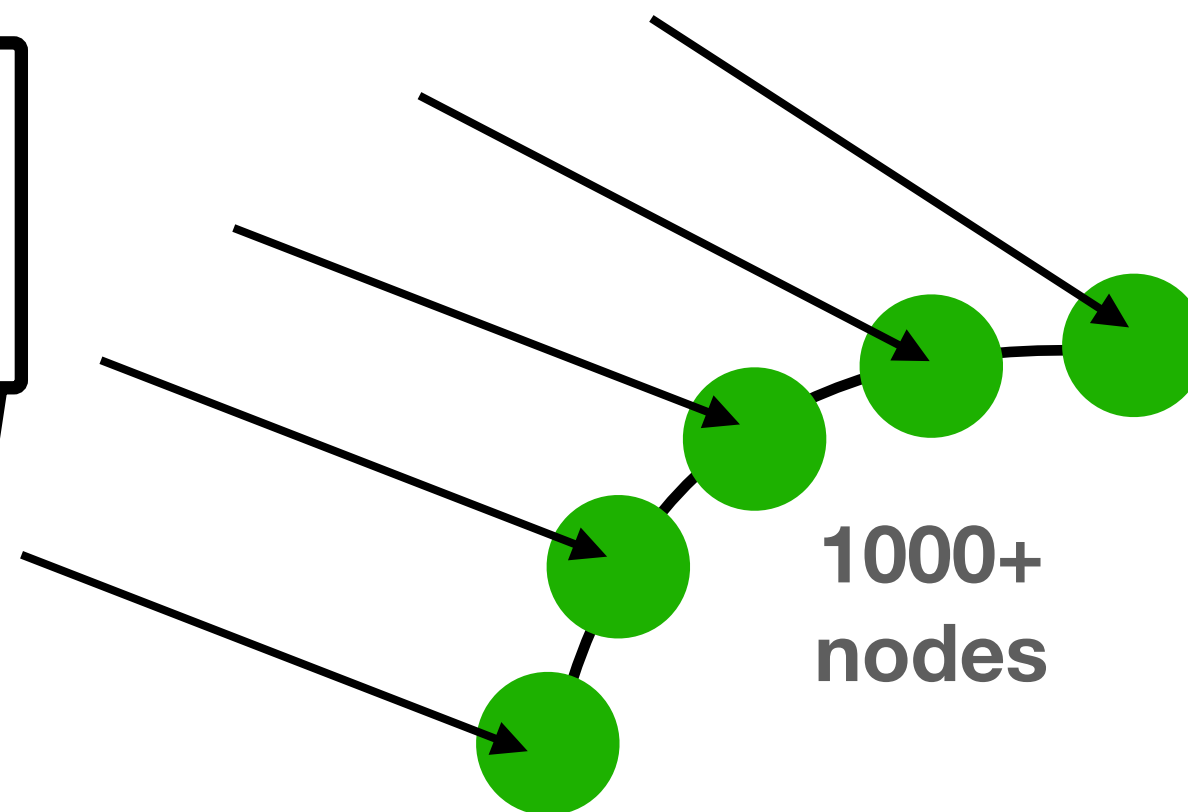
users	
user_id	K
name	
birth_year	
...	

SELECT - partitions and keys

- What happens if no partition is given?

```
SELECT * FROM users
```

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



This is valid!
Lets see some examples

users	
user_id	K
name	
birth_year	
...	

SELECT - partitions and keys

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

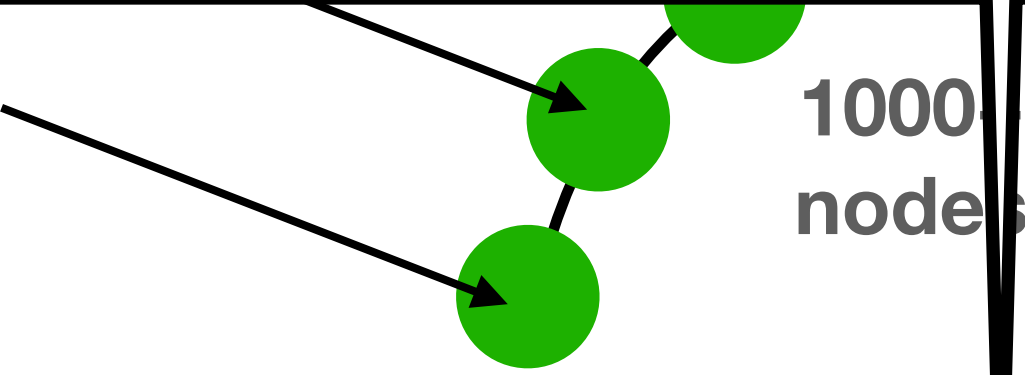
Assume there are **10k nodes** in the cluster and **no replication**
 - If there are **10 users**, would the query be optimal?
 (that is, we would not check unnecessary nodes/partitions)

NO - why?

There are 10 partitions which are distributed on 10k nodes.
 We will initiate 9990 unnecessary calls

?

users	
	K



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**

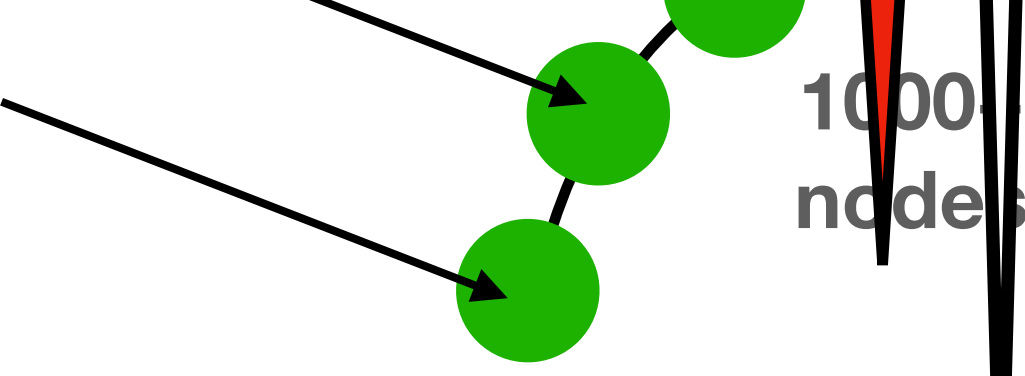
- If there are 10 users
(that is, we would have 10 partitions)

NO - why?
The there are 10k nodes. We will have 10 partitions.

```
SELECT * from <TABLE> - Summary
```

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

K

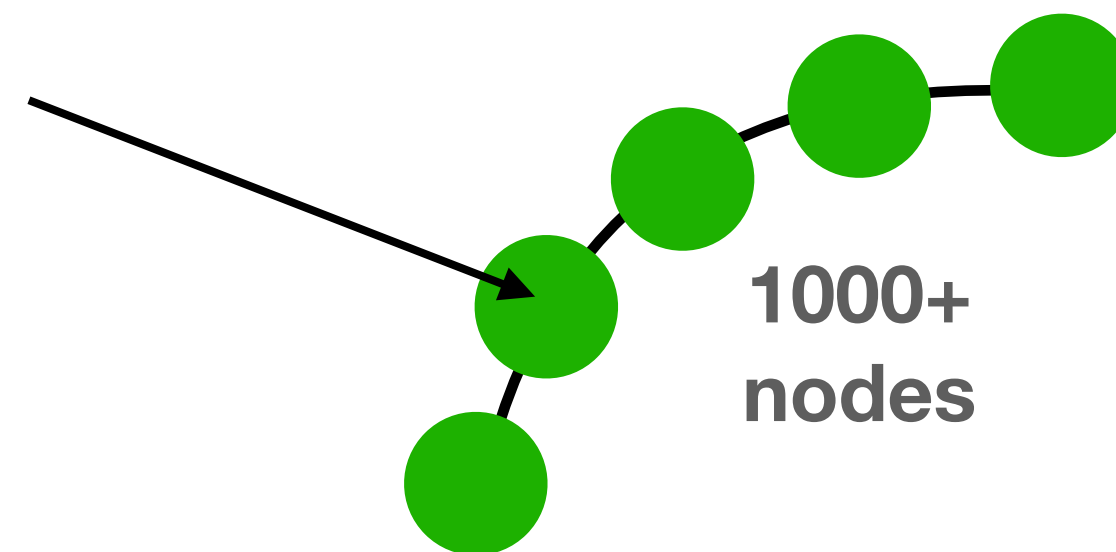


The right way for this scenario is to create a single partition for these 10 users, then read 1 partition

SELECT - partitions and keys

- Try a different model

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



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

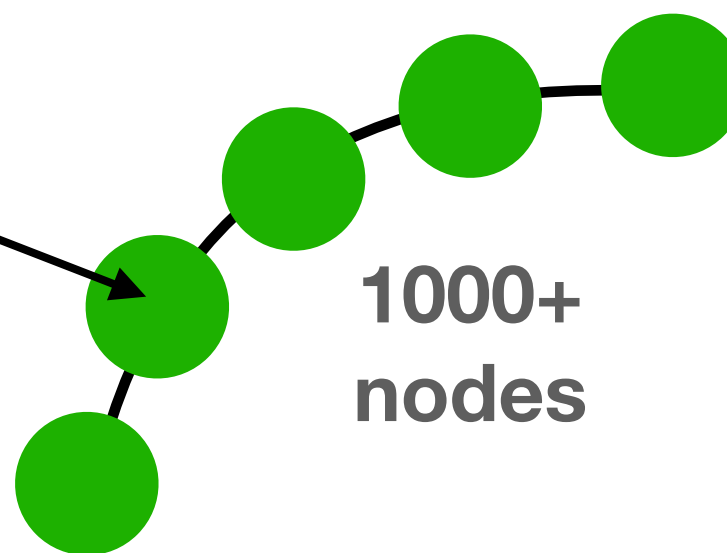
users	
country	K
user_id	▼ C
name	
birth_year	
...	

SELECT - partitions and keys

- Try a different model

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

Reading the users from Israel is fast



users	
country	K
user_id	▼C
name	
birth_year	
...	

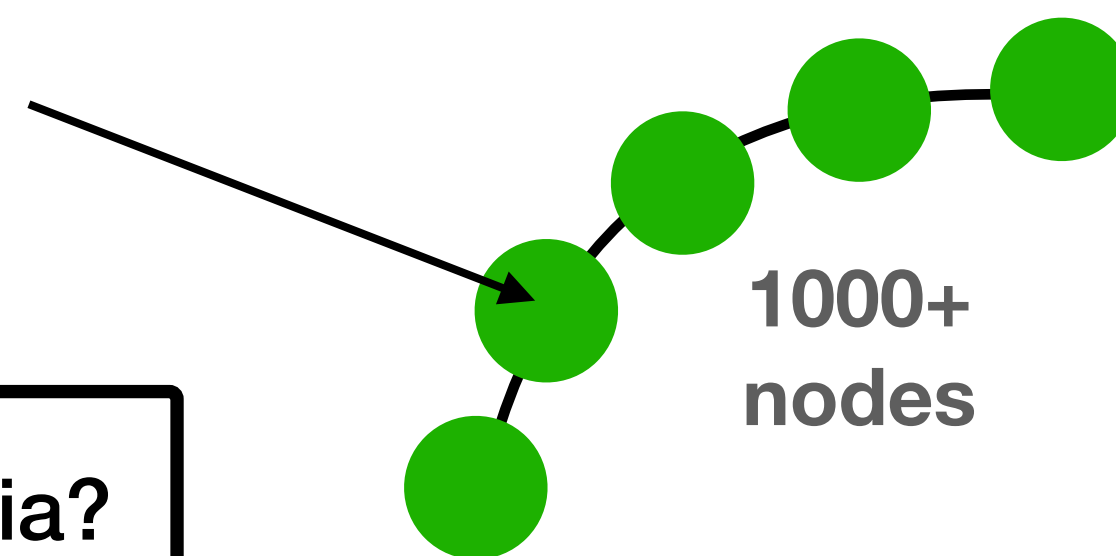
SELECT - partitions and keys

- 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?



SELECT - partitions and keys

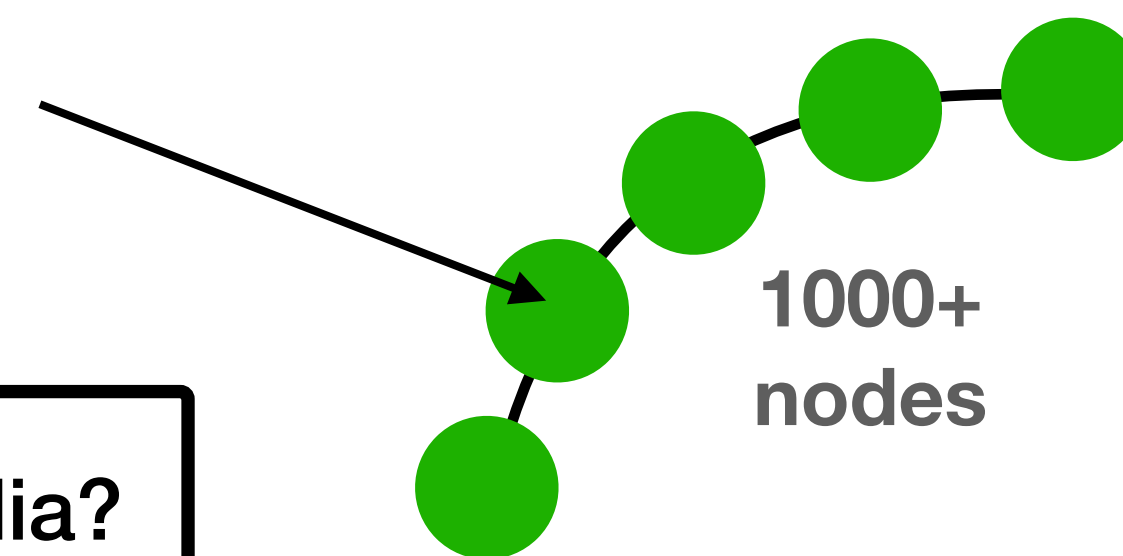
- Try a different model

```
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?



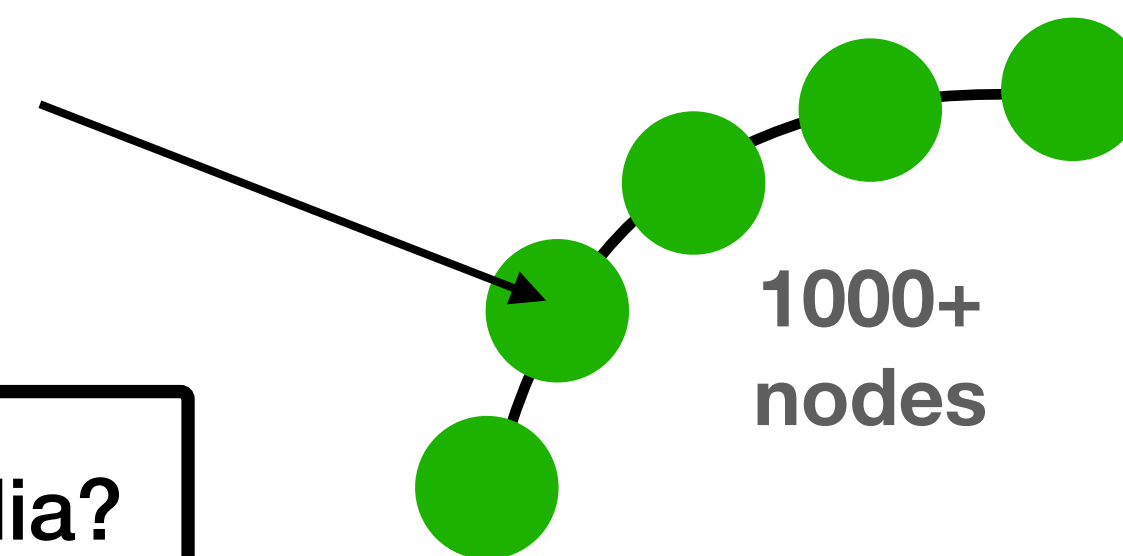
SELECT - partitions and keys

- 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?



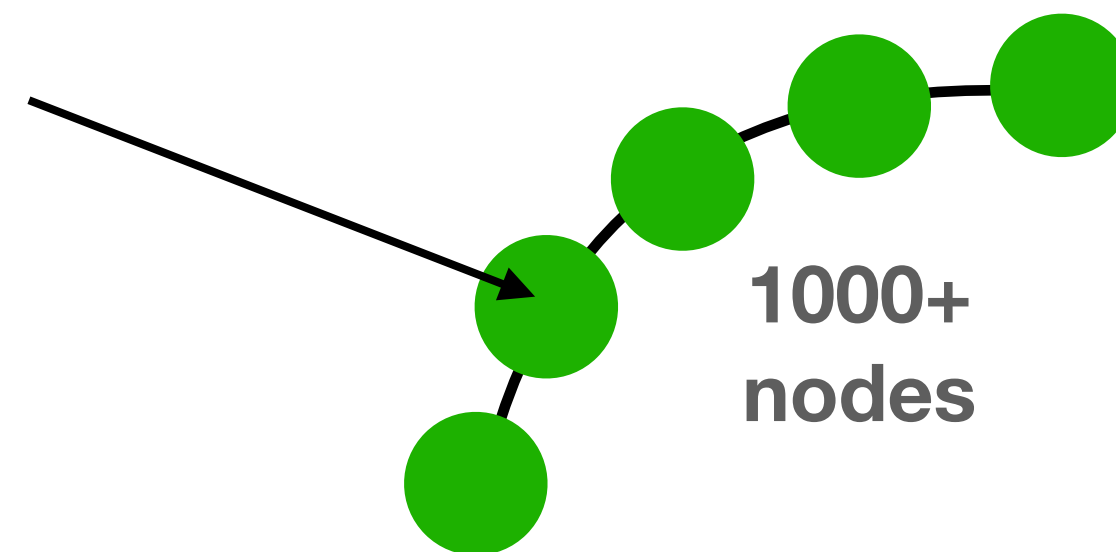
How can you solve this issue?

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	
...	



Error - why?

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?

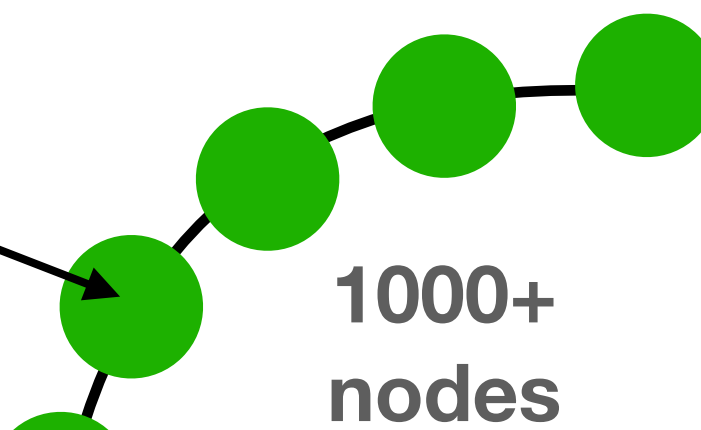
Cassandra will need to read the entire partition.
If there are 1m users, and only 10k were born in 1982,
there would be an unnecessary read/filter of 990k users

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)

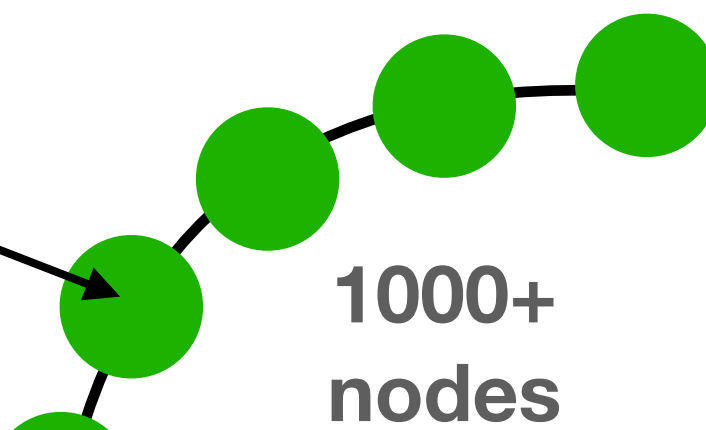
SELECT - partitions and keys

- 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	

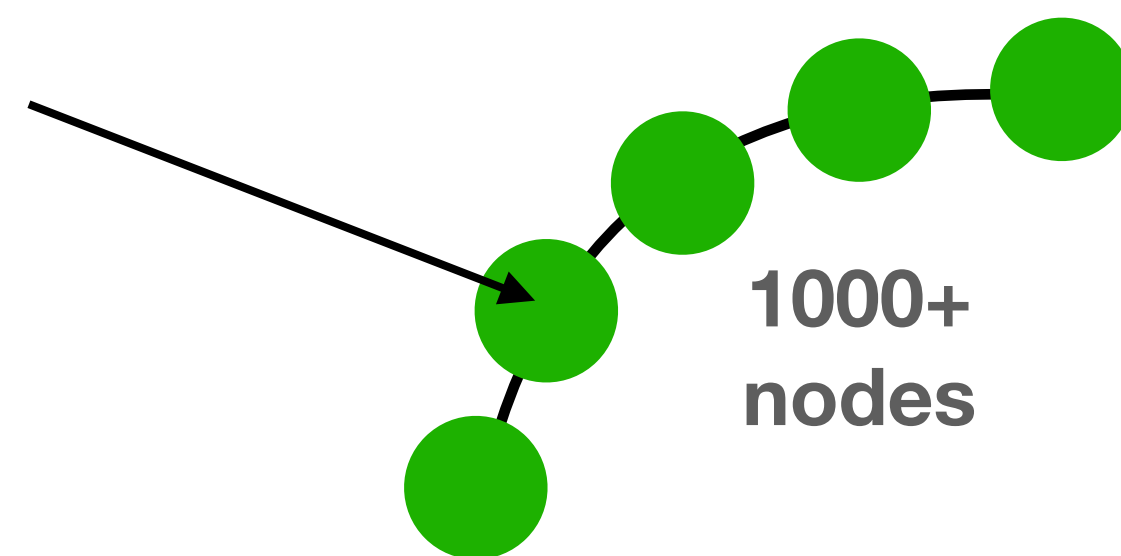


With “ALLOW FILTERING” Cassandra will approve the query
(ANTI PATTERN)

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	
...	

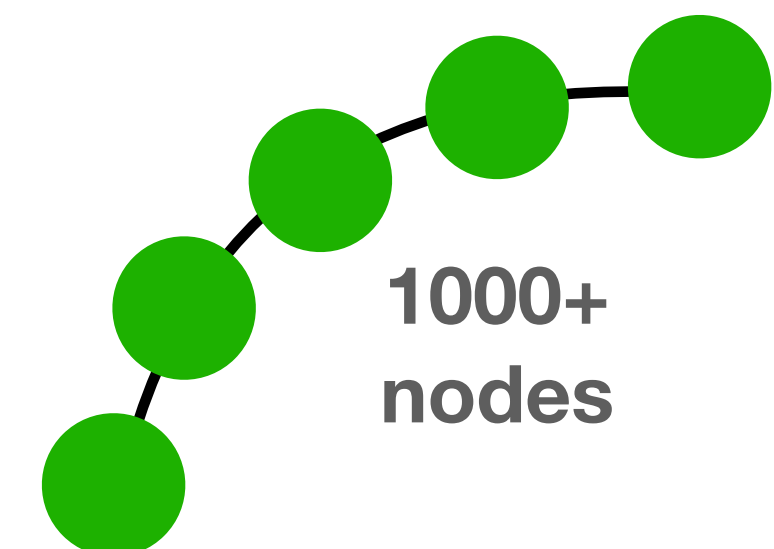
users_by_birth_year	
country	K
birth_year	▼C
user_id	▼C
name	
...	

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	



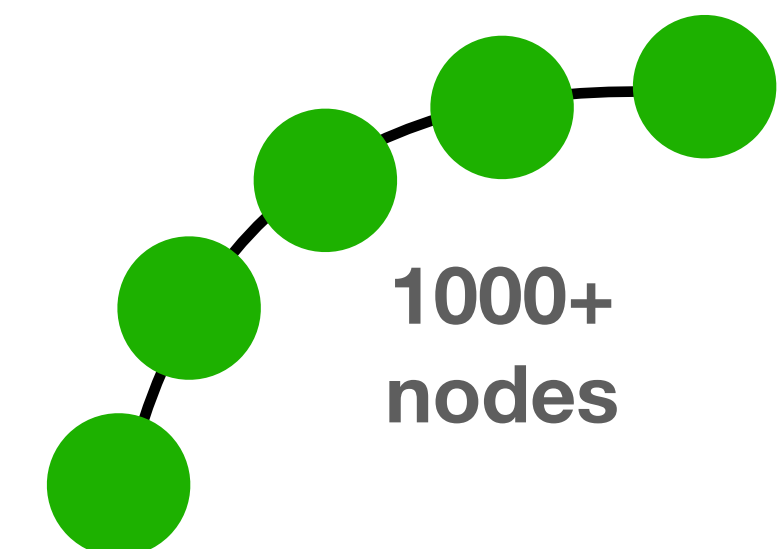
SELECT - partitions and keys

- 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

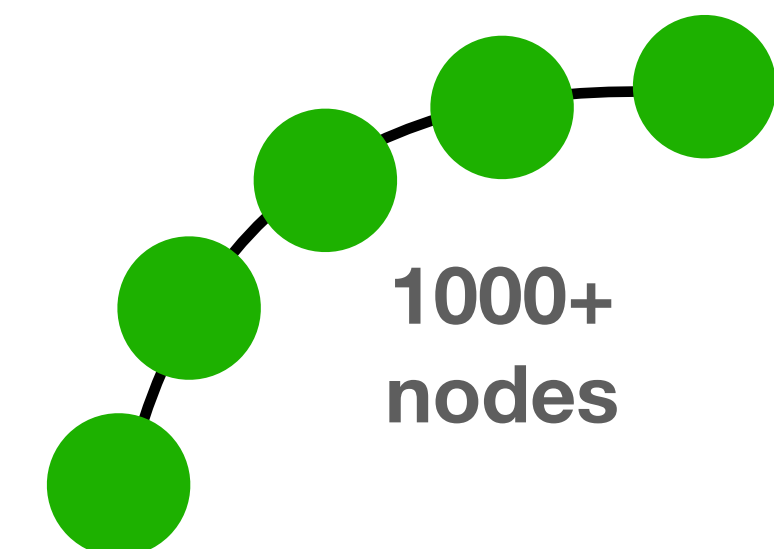
- And what about this case?

```
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



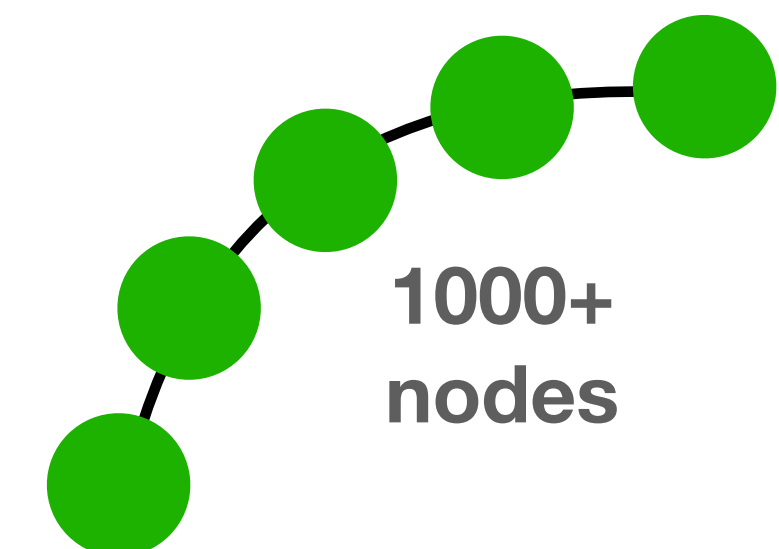
SELECT - partitions and keys

- And what about this case?

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

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

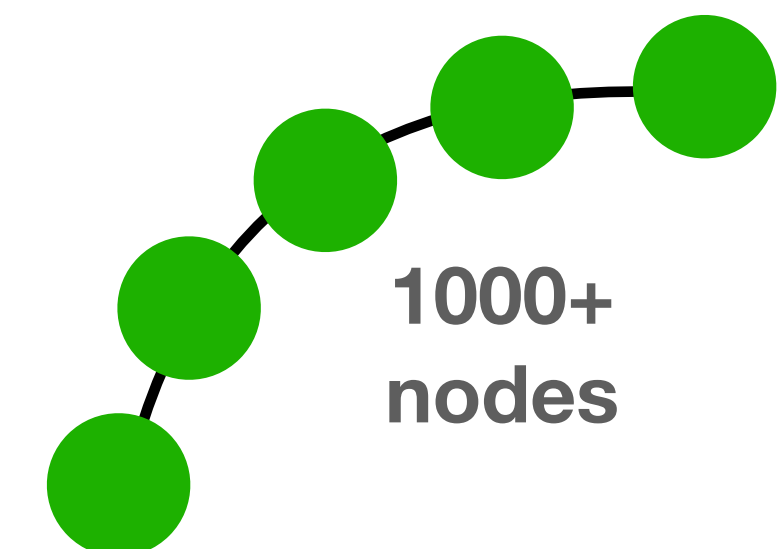
users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	



SELECT - ALLOW FILTERING

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

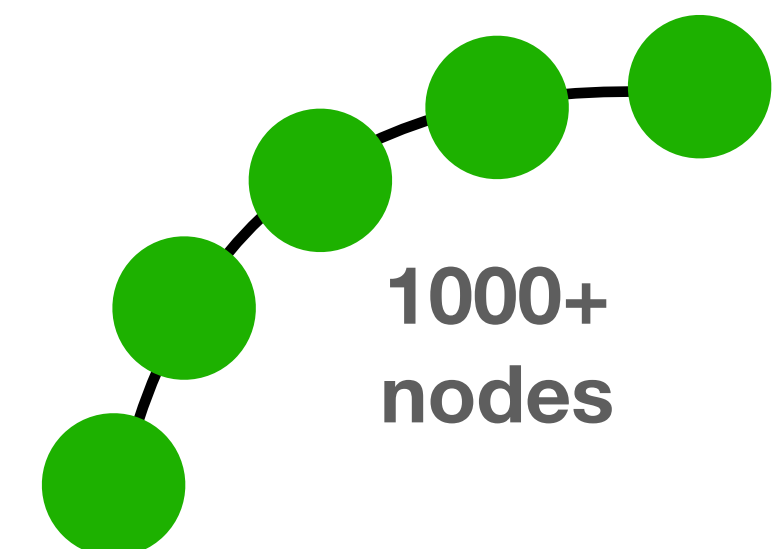
users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	



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	



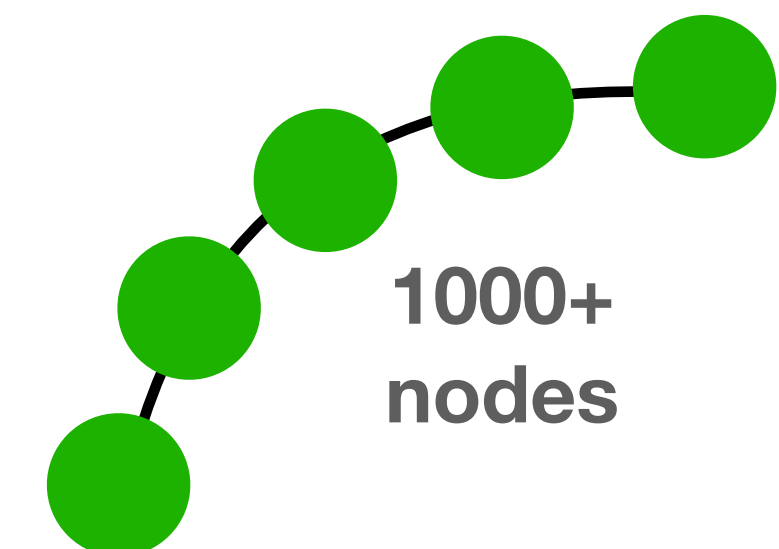
SELECT - ALLOW FILTERING

- Almost always ANTI PATTERN

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

- We save
 - 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 EXISTS  
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 is 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	
...	

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];
```