

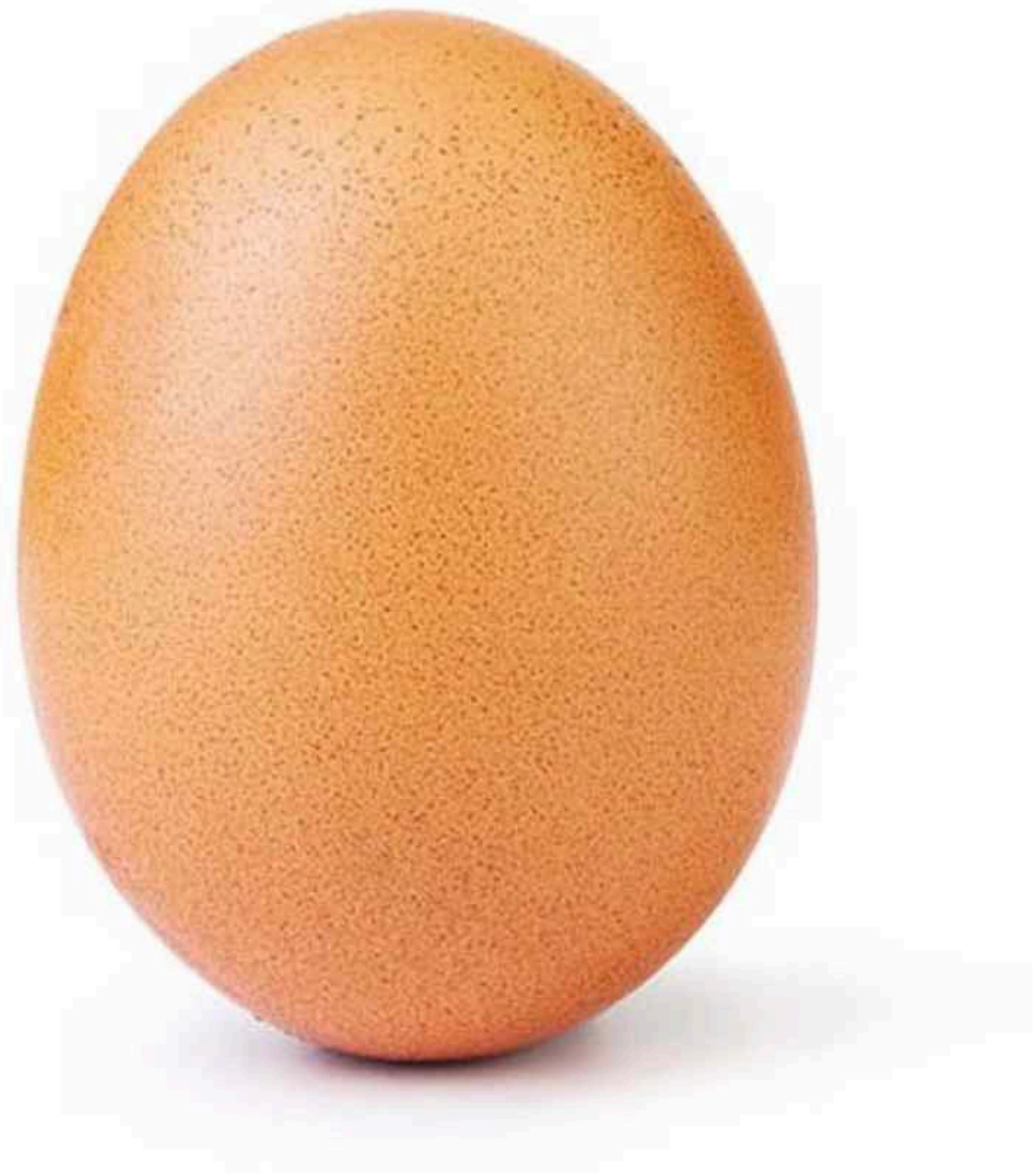
# Cassandra - CQL

Big Data Systems

Dr. Rubi Boim



world\_record\_egg  · [Follow](#)

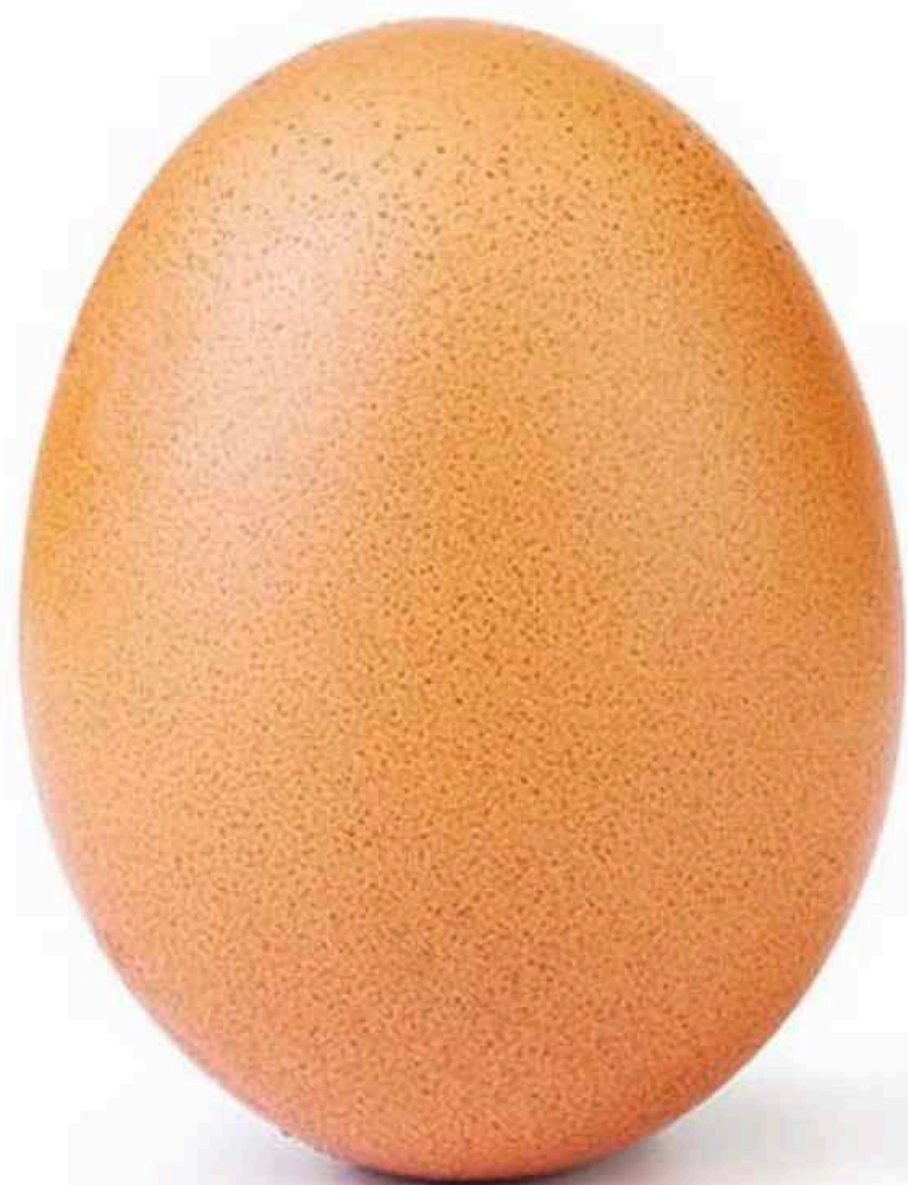


**55,957,347 likes**

JANUARY 4, 2019



world\_record\_egg • Follow



55,957,347 likes

JANUARY 4, 2019



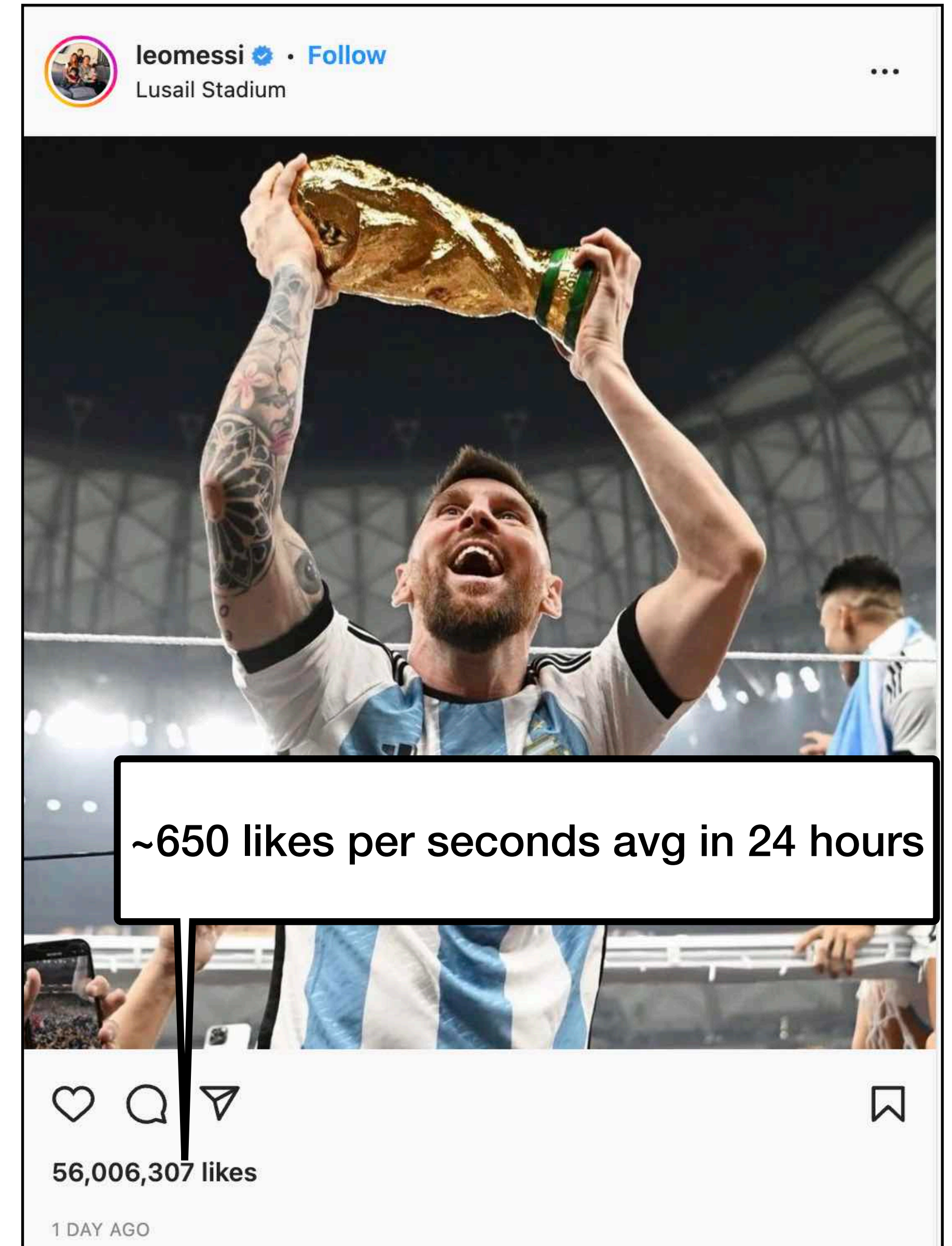
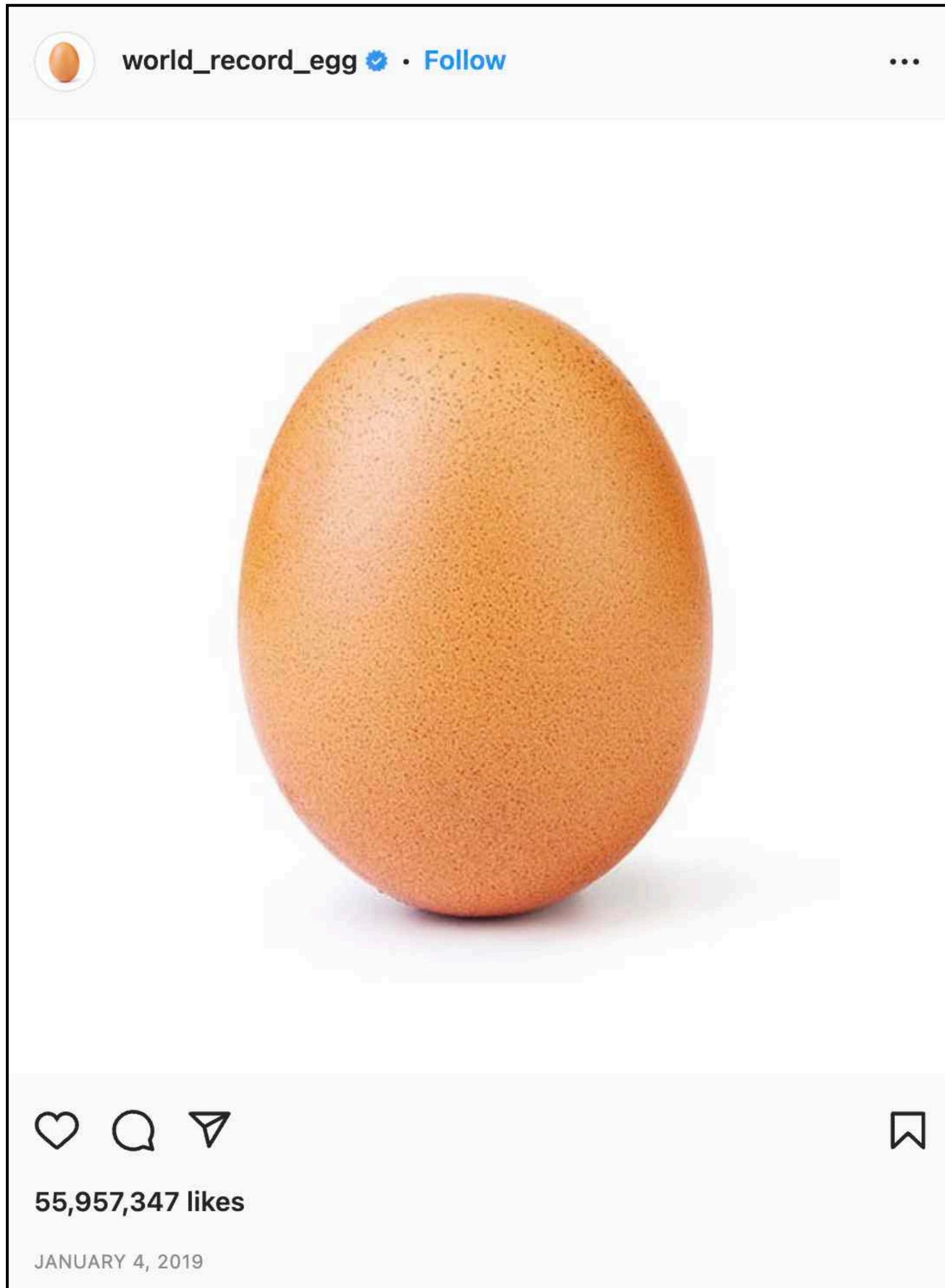
leomessi • Follow

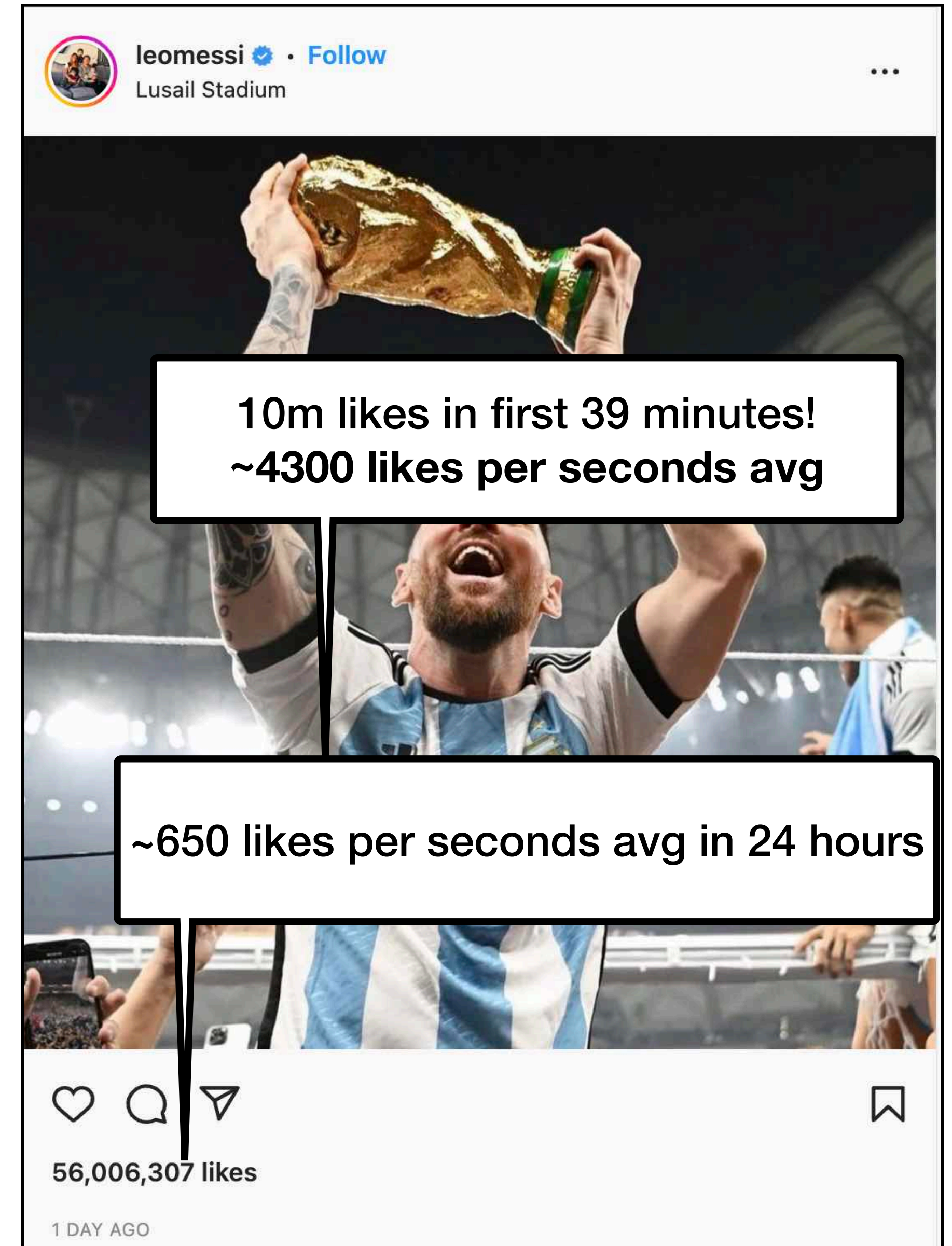
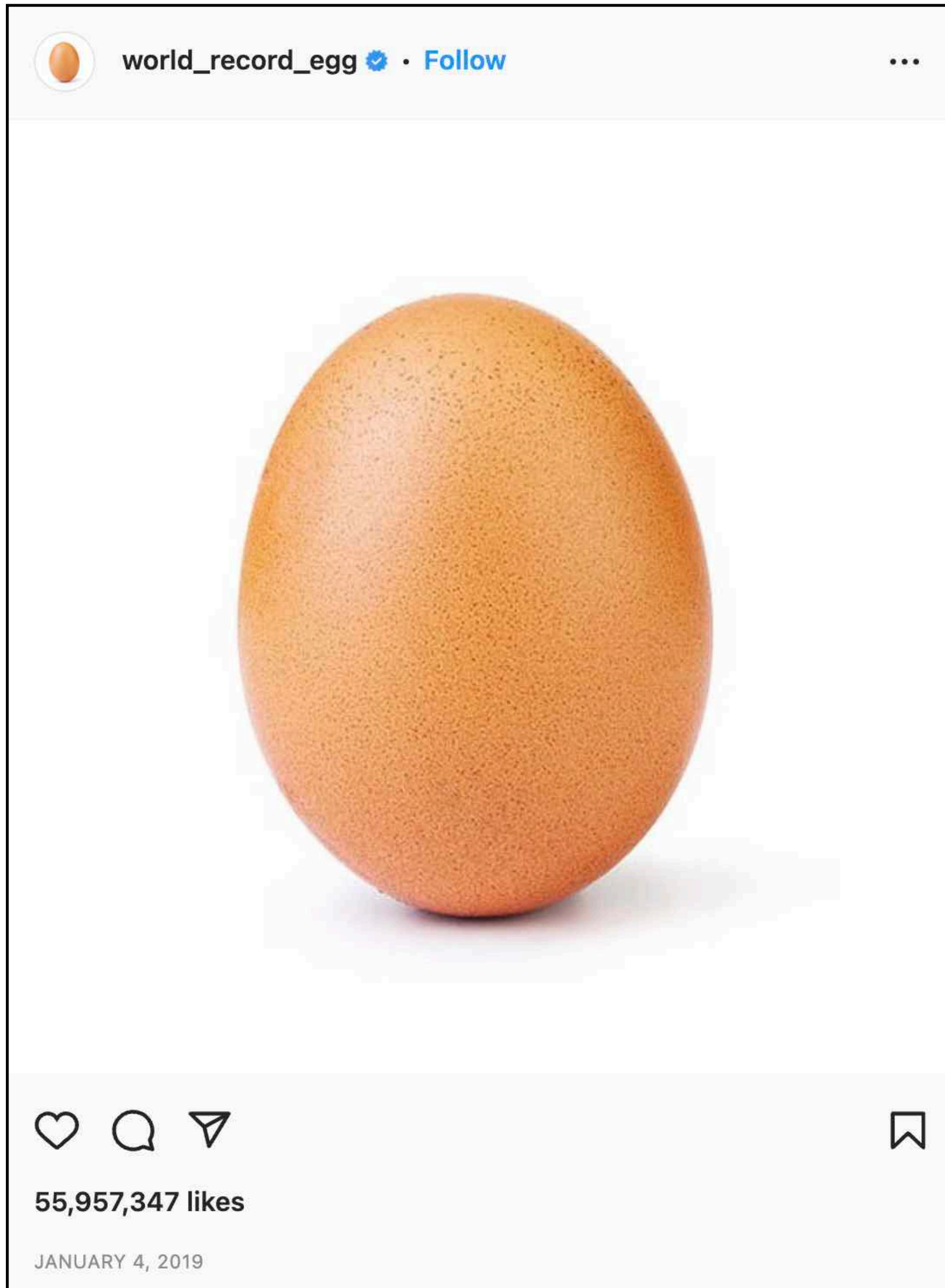
Lusail Stadium



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



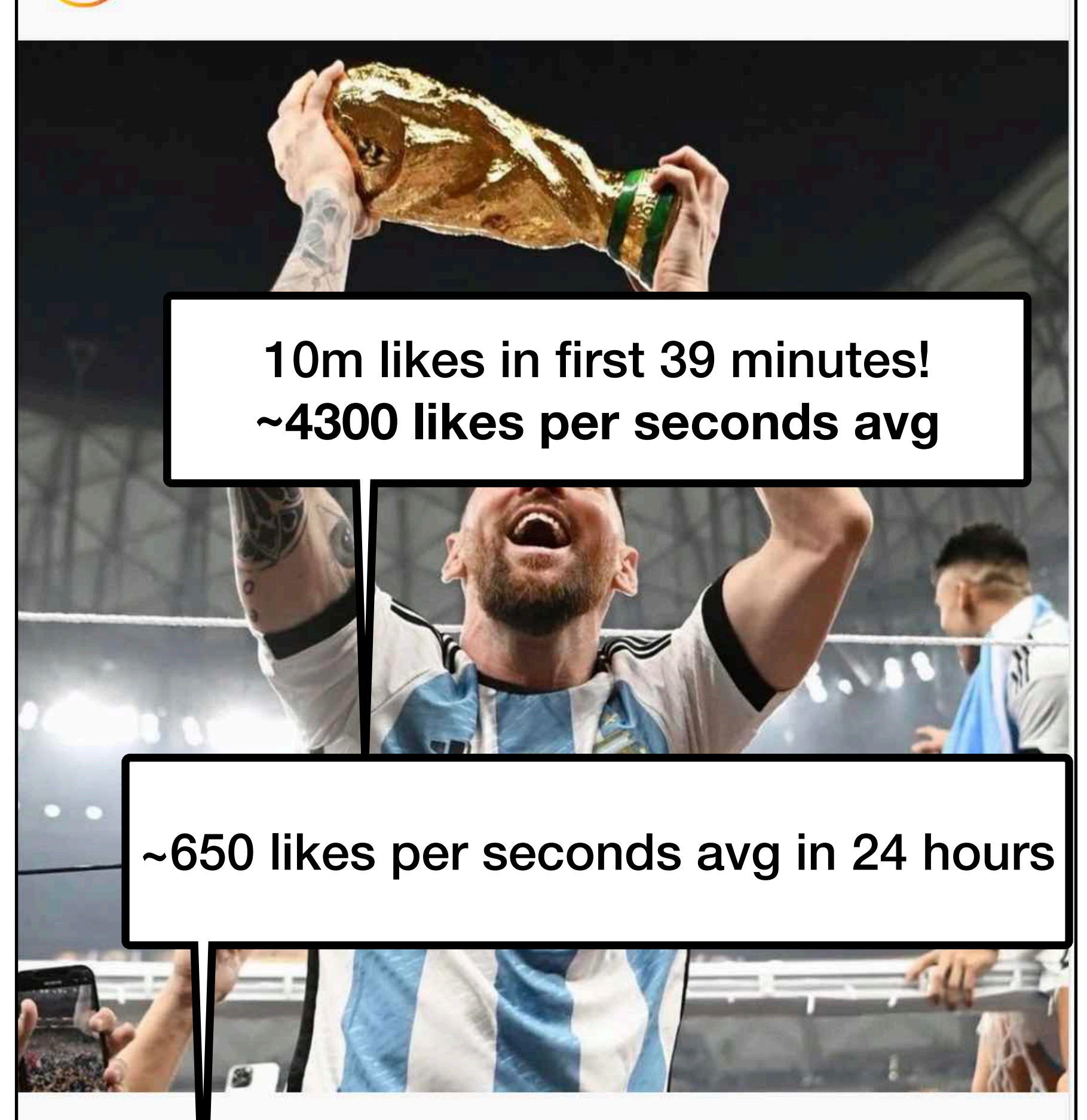
9m likes in first 24 hours  
~100 likes per seconds avg

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' is displayed.

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

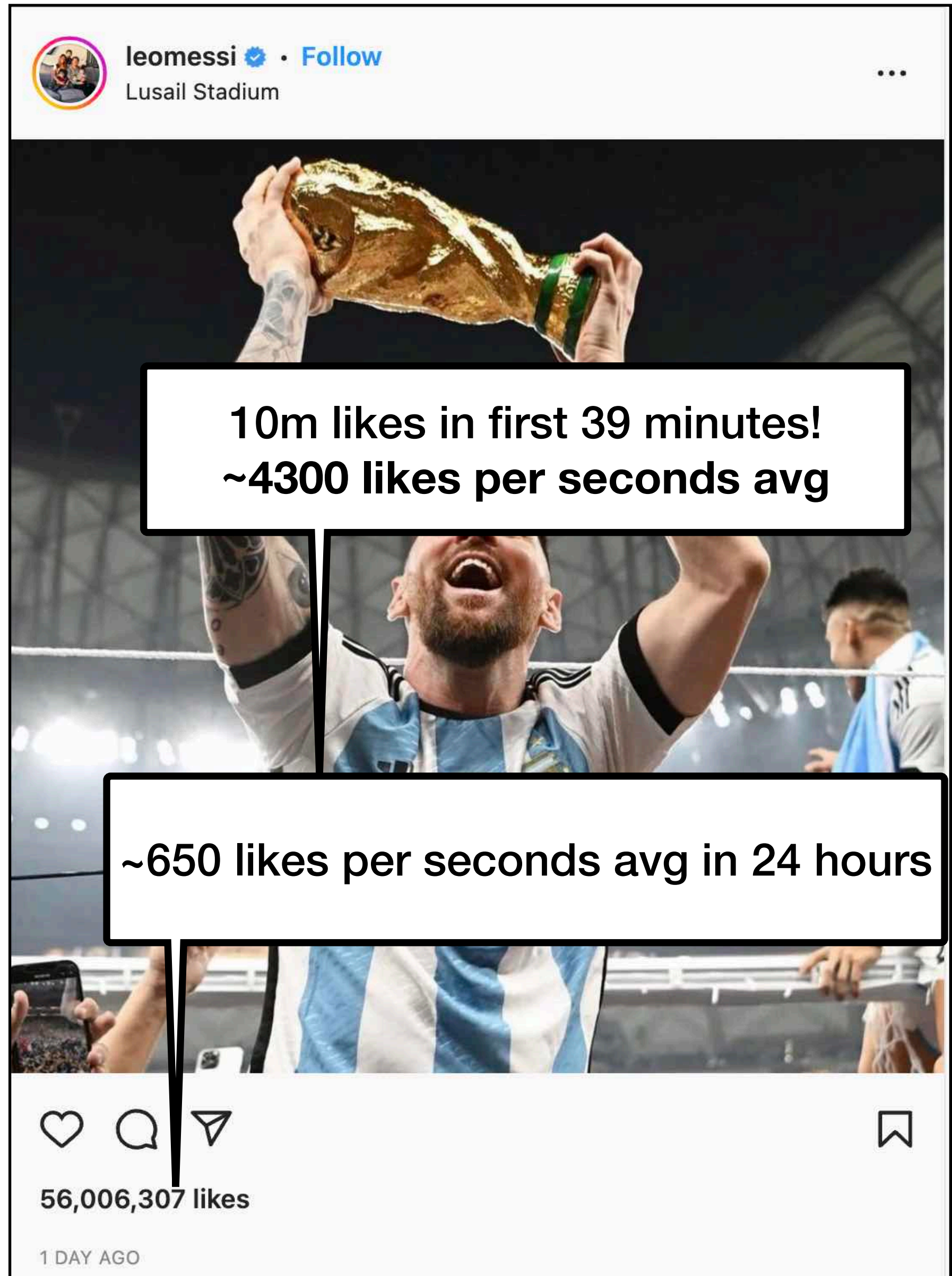
1 DAY AGO

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 his 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 a white box with a black border. At the bottom, the text '56,006,307 likes' and '1 DAY AGO' is displayed.



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 [\[ edit \]](#)

Two accounts have more than one post in the top 20 most-liked posts list: [Lionel Messi](#) and [Cristiano Ronaldo](#) have seven each.

Rank <span>↕</span>	Account name <span>↕</span>	Owner <span>↕</span>	Post description	Post	Likes (millions) <span>↕</span>	Date posted (UTC) <span>↕</span>
1	<a href="#">@leomessi</a>	<a href="#">Lionel Messi</a>	Photos of <a href="#">Lionel Messi</a> and the <a href="#">Argentina national football team</a> after winning the <a href="#">2022 FIFA World Cup</a>	<a href="#">[1]</a> <span>↗</span>	75.5	December 18, 2022
2	<a href="#">@world_record_egg</a>	<a href="#">Chris Godfrey</a>	Photo of <a href="#">an egg</a>	<a href="#">[2]</a> <span>↗</span>	60.1	January 4, 2019
3	<a href="#">@leomessi</a>	<a href="#">Lionel Messi</a>	<a href="#">Lionel Messi</a> in bed with <a href="#">Mate</a> drink and the <a href="#">FIFA World Cup Trophy</a>	<a href="#">[3]</a> <span>↗</span>	54.6	December 20, 2022
4	<a href="#">@cristiano</a>	<a href="#">Cristiano Ronaldo</a>	<a href="#">Lionel Messi</a> and <a href="#">Cristiano Ronaldo</a> playing chess, advertising for <a href="#">Louis Vuitton</a>	<a href="#">[4]</a> <span>↗</span>	42.6	November 19, 2022
5	<a href="#">@leomessi</a>	<a href="#">Lionel Messi</a>	<a href="#">Lionel Messi</a> on an airplane with the <a href="#">FIFA World Cup Trophy</a>	<a href="#">[5]</a> <span>↗</span>	41.9	December 19, 2022
6	<a href="#">@jiangzhibin24</a>	<a href="#">Liz 6</a>	Reel of a sunset	<a href="#">[6]</a> <span>↗</span>	34.3	August 5, 2023
7	<a href="#">@leomessi</a>	<a href="#">Lionel Messi</a>	Celebrating the <a href="#">2022 FIFA World Cup</a> win in <a href="#">Argentina</a>	<a href="#">[7]</a> <span>↗</span>	34.3	December 21, 2022
8	<a href="#">@cristiano</a> <a href="#">@alnassr</a>	<a href="#">Cristiano Ronaldo</a> <a href="#">Al Nassr FC</a>	Announcement of <a href="#">Cristiano Ronaldo</a> joining <a href="#">Al Nassr FC</a>	<a href="#">[8]</a> <span>↗</span>	34.3	December 30, 2022
9	<a href="#">@cristiano</a>	<a href="#">Cristiano Ronaldo</a>	After elimination of <a href="#">Portugal</a> from the <a href="#">2022 FIFA World Cup</a>	<a href="#">[9]</a> <span>↗</span>	34.0	December 11, 2022
10	<a href="#">@leomessi</a>	<a href="#">Lionel Messi</a>	<a href="#">Lionel Messi</a> and <a href="#">Cristiano Ronaldo</a> playing chess, advertising for <a href="#">Louis Vuitton</a>	<a href="#">[10]</a> <span>↗</span>	32.7	November 19, 2022
11	<a href="#">@xxxtentacion</a>	<a href="#">XXXTentacion</a>	Final post before <a href="#">his death</a>	<a href="#">[11]</a> <span>↗</span>	32.5	May 19, 2018
12	<a href="#">@cristiano</a> <a href="#">@georginagio</a>	<a href="#">Cristiano Ronaldo</a> <a href="#">Georgina Rodríguez</a>	Their twins pregnancy announcement	<a href="#">[12]</a> <span>↗</span>	32.3	October 28, 2021
13	<a href="#">@cristiano</a>	<a href="#">Cristiano Ronaldo</a>	Post in remembrance of <a href="#">Pelé</a>	<a href="#">[13]</a> <span>↗</span>	32.2	December 29, 2022
14	<a href="#">@leomessi</a>	<a href="#">Lionel Messi</a>	After <a href="#">2022 FIFA World Cup</a> match against <a href="#">Croatia</a>	<a href="#">[14]</a> <span>↗</span>	29.6	December 14, 2022
15	<a href="#">@cristiano</a>	<a href="#">Cristiano Ronaldo</a>	Post of <a href="#">Cristiano Ronaldo</a> being presented to the <a href="#">Al Nassr FC</a> fans	<a href="#">[15]</a> <span>↗</span>	27.8	January 3, 2023
16	<a href="#">@cristiano</a>	<a href="#">Cristiano Ronaldo</a>	Post of friendly between <a href="#">PSG</a> and <a href="#">Riyadh XI</a>	<a href="#">[16]</a> <span>↗</span>	27.8	January 19, 2023
17	<a href="#">@zendaya</a>	<a href="#">Zendaya</a>	Happy birthday post to <a href="#">Tom Holland</a>	<a href="#">[17]</a> <span>↗</span>	26.2	June 1, 2022
18	<a href="#">@leomessi</a>	<a href="#">Lionel Messi</a>	Post in remembrance of <a href="#">Pelé</a>	<a href="#">[18]</a> <span>↗</span>	25.9	December 29, 2022
19	<a href="#">@selenagomez</a>	<a href="#">Selena Gomez</a>	Selfie photo with caption "Violet Chemistry"	<a href="#">[19]</a> <span>↗</span>	25.8	March 14, 2023
20	<a href="#">@tomholland2013</a>	<a href="#">Tom Holland</a>	Recreating the " <a href="#">Spider-Man</a> pointing" meme with <a href="#">Tobey Maguire</a> and <a href="#">Andrew Garfield</a>	<a href="#">[20]</a> <span>↗</span>	25.1	February 23, 2022

As of 24 January 2024

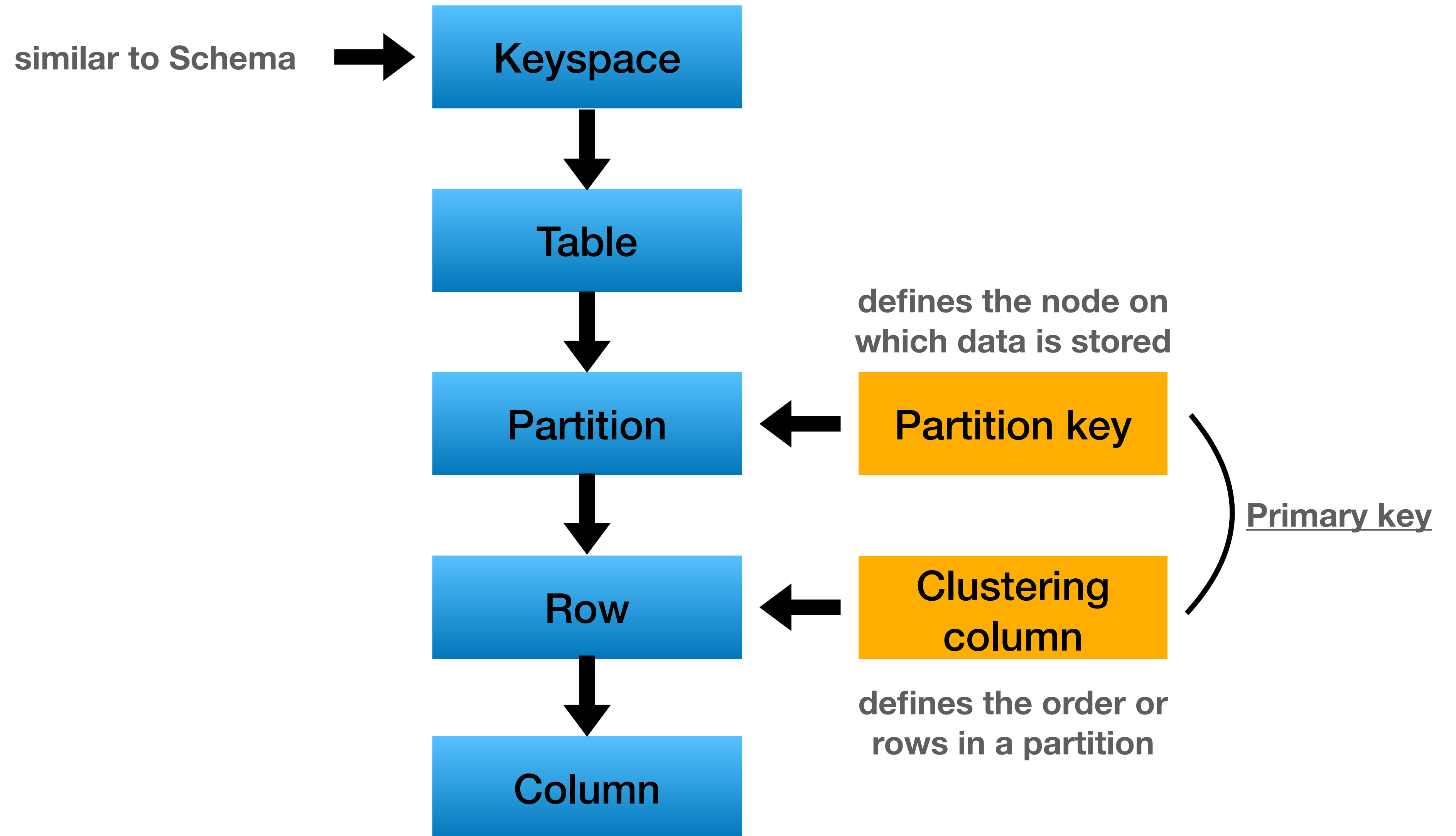
# Cassandra CQL

- 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 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** **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
```

- “Limited” compared to RDBMS

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

no joins / having / union...

**ANTI PATTERN**

Can be very slow and expensive - when?

# 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

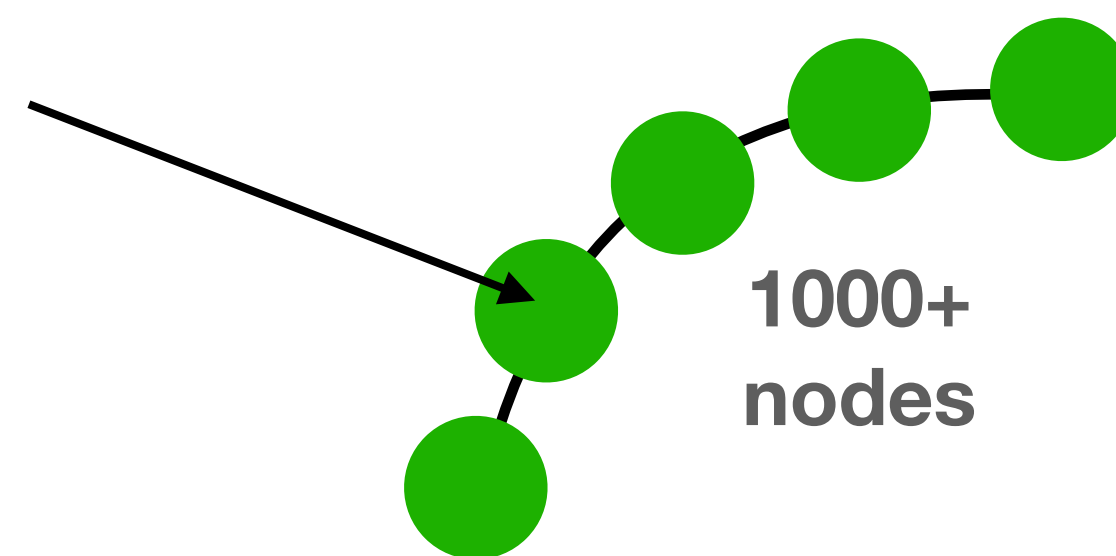
sum / avg / min / max or only supported on new versions

no joins / having / union...

# SELECT - partitions and keys

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

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



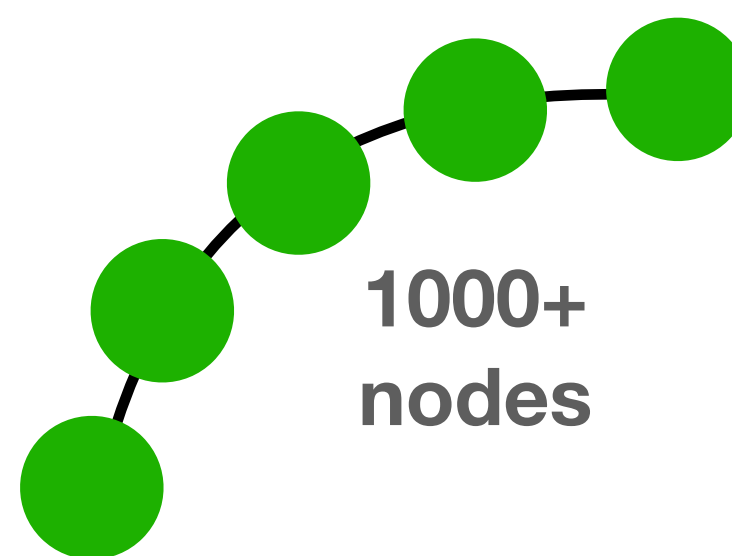
users	
user_id	K
name	
birth_year	
...	



# SELECT - partitions and keys

- What happens if no partition is given?

```
SELECT * FROM users
```



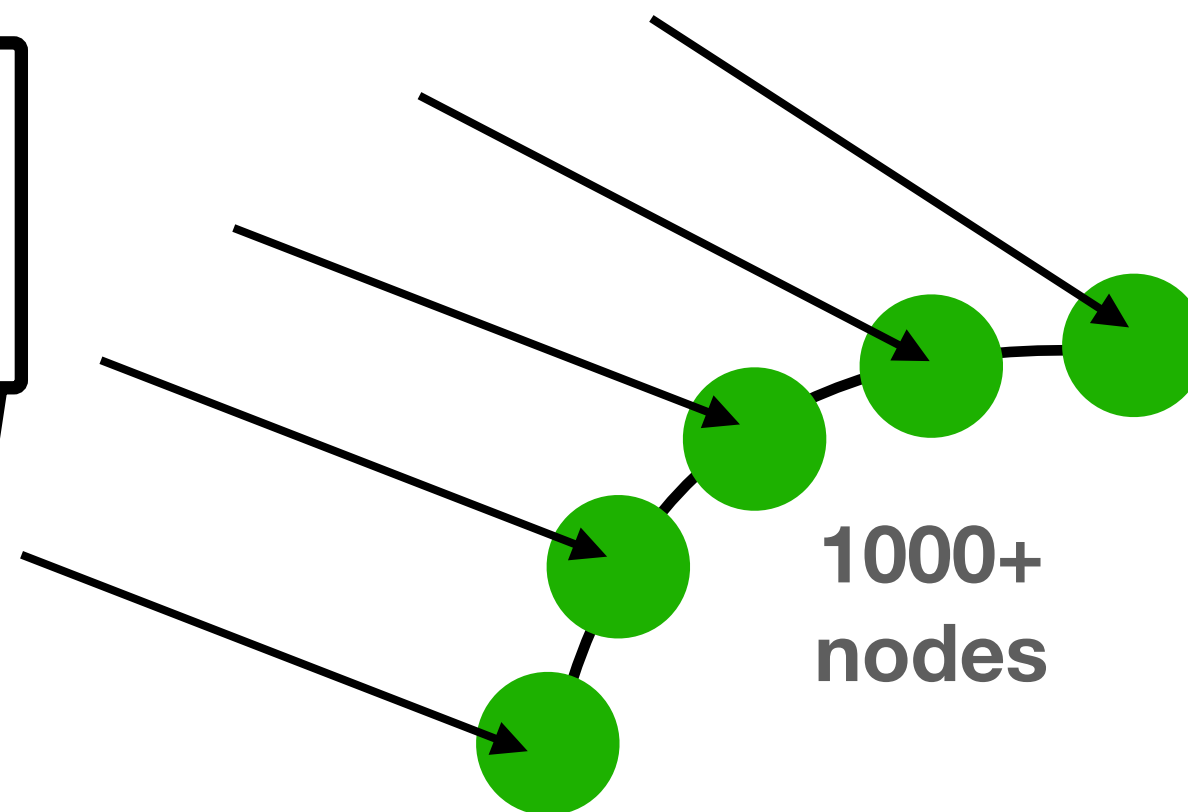
users	
user_id	K
name	
birth_year	
...	

# 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)



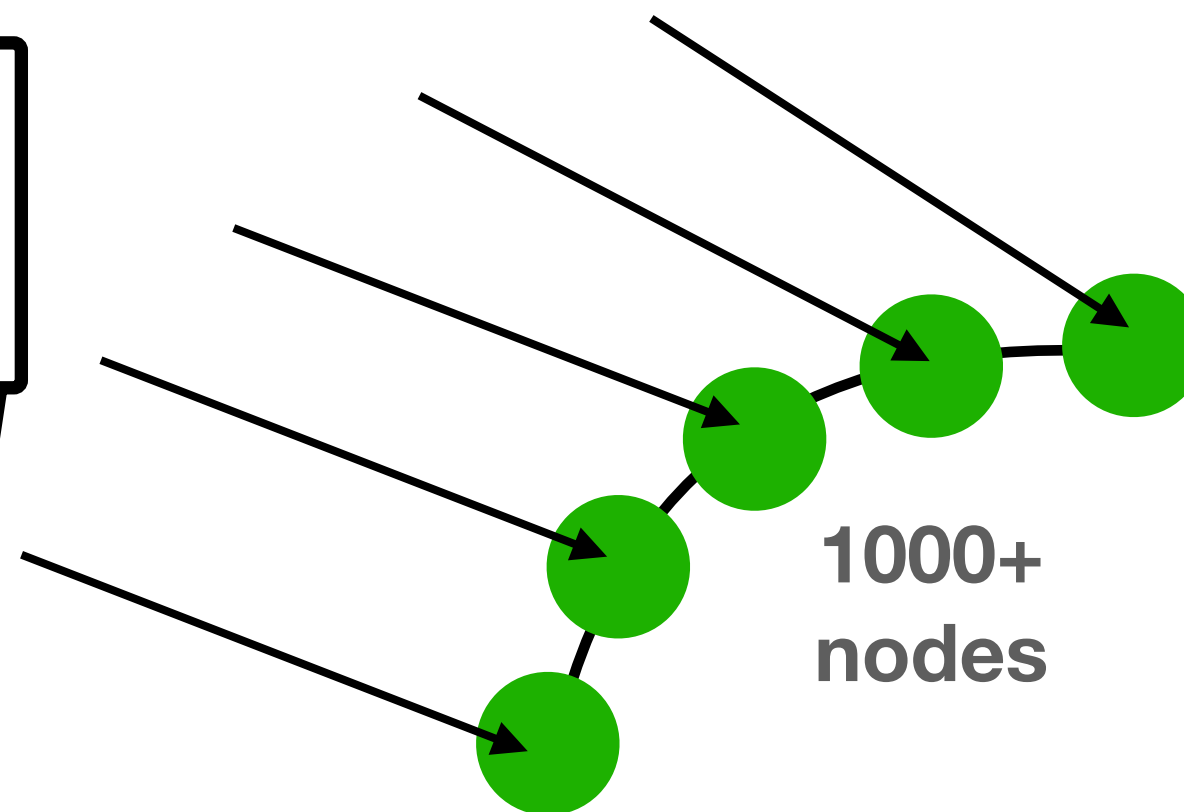
users	
user_id	K
name	
birth_year	
...	

# 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	
...	

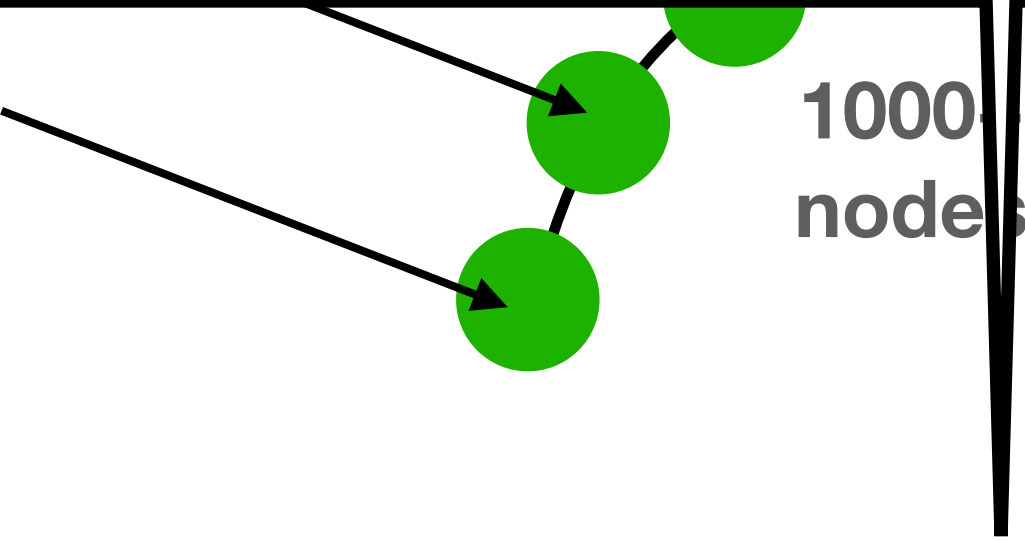
# SELECT - partitions and keys

Each user “creates” a partition (user\_id is partition\_key)

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

?

users	
	K



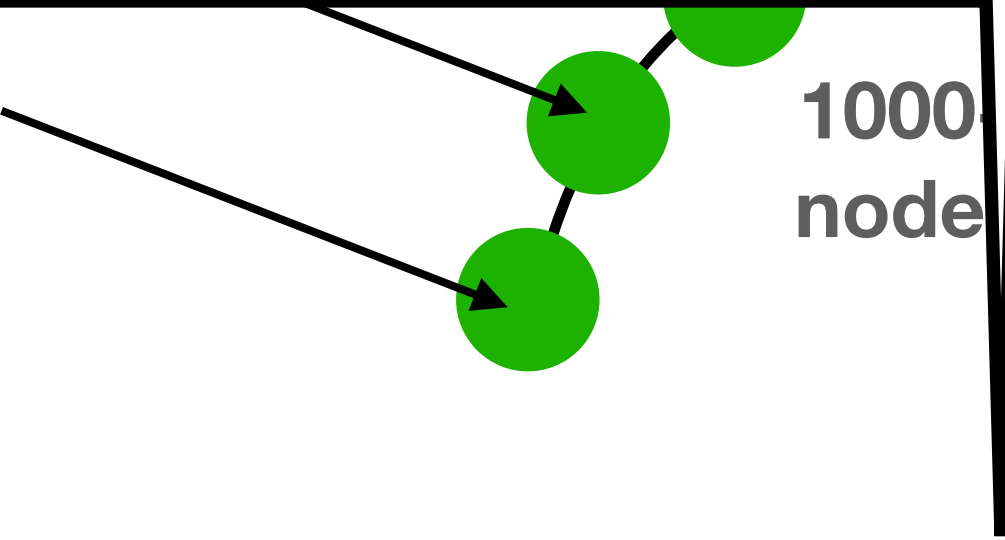
# SELECT - partitions and keys

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?**

?



# 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**

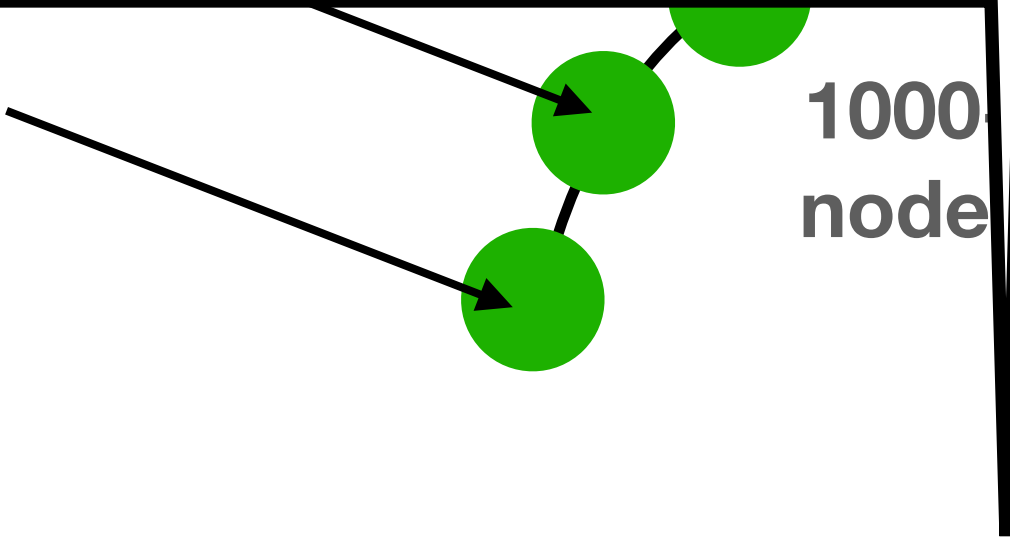
- If there are **100k users**, would the query be optimal?  
(that is, we would not check unnecessary nodes/partitions)

**YES - why?**

There are 100k partitions which are distributed on 10k nodes

?

users	
	K



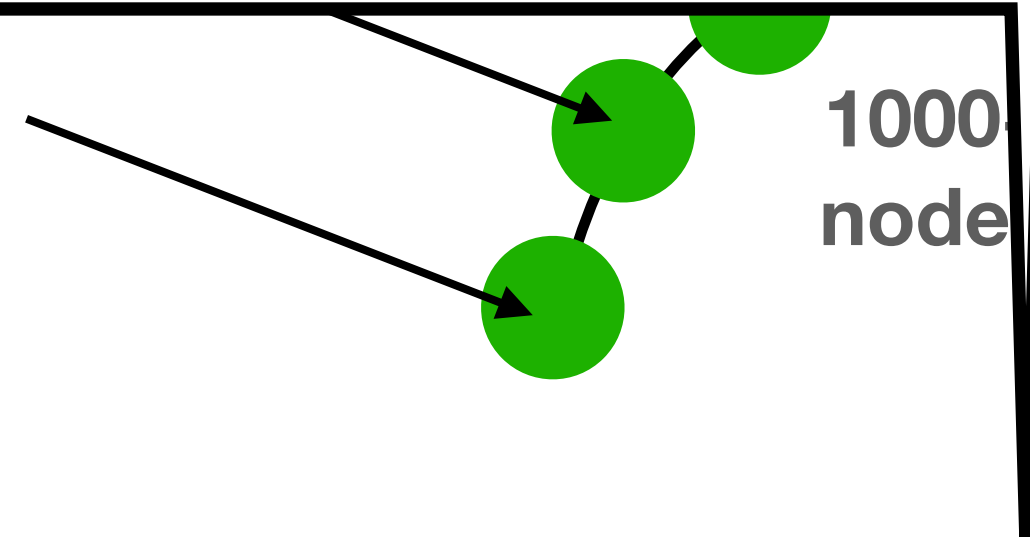
# SELECT - partitions and keys

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)

?

users
K



# SELECT - partitions and keys

Each user “creates” a partition (user\_id is partition\_key)

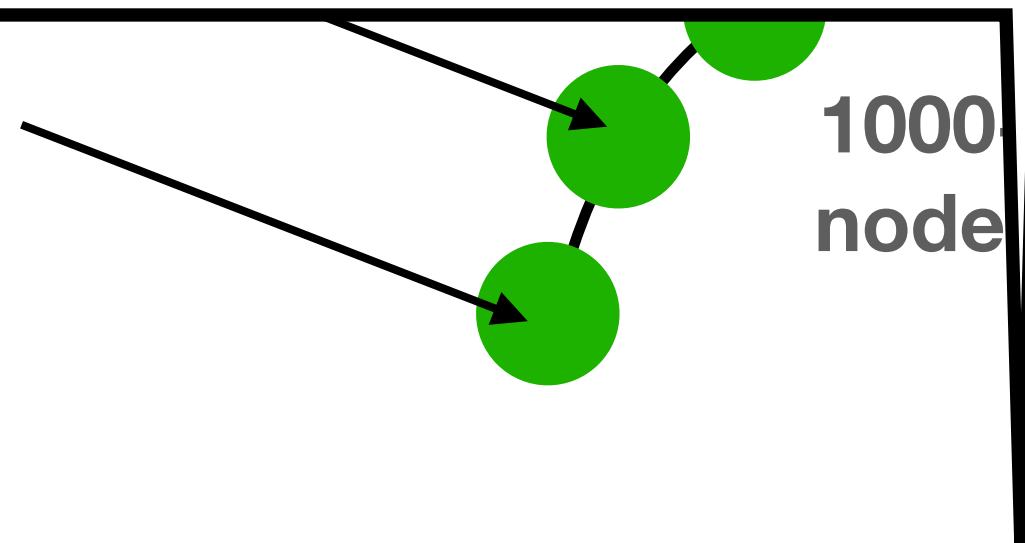
Assume there are **10k nodes** in the cluster and **no replication**  
- If there are **10 users**, would the query be optimal?  
(that is, we would not check unnecessary nodes/partitions)

**NO - why?**

?

users
K

...





# 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
...

1000 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)

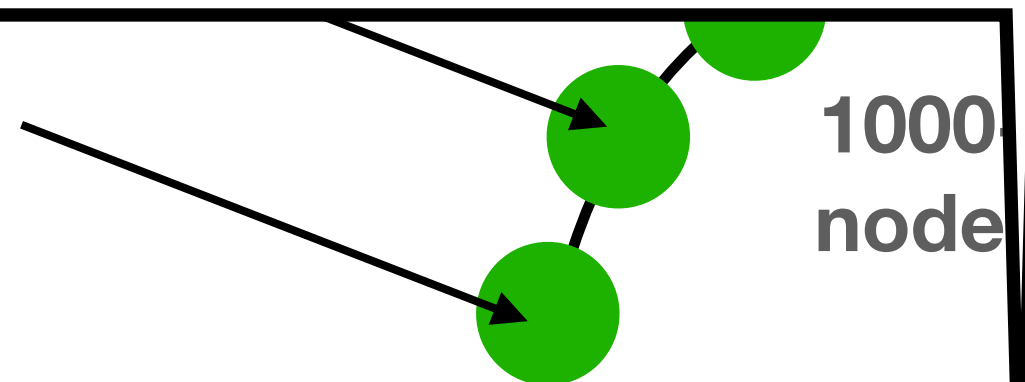
**NO - why?**

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 u**  
(that is, we wo

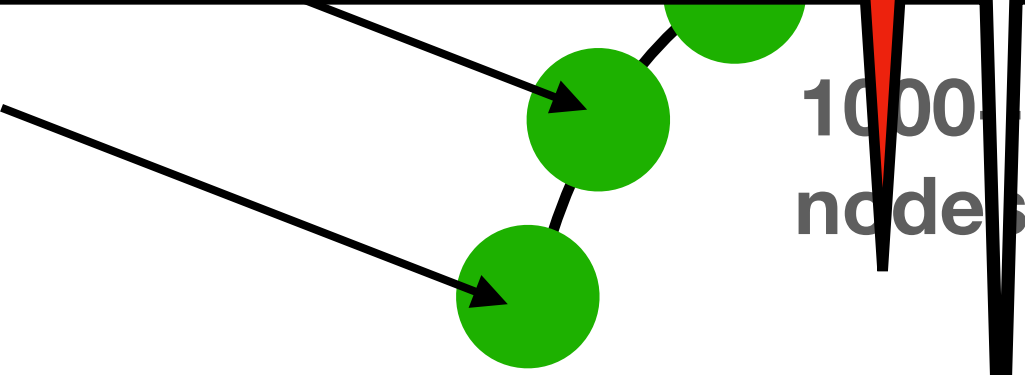
```
SELECT * from <TABLE> - Summary
```

**NO - why?**

The there are  
nodes. We wi

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

K

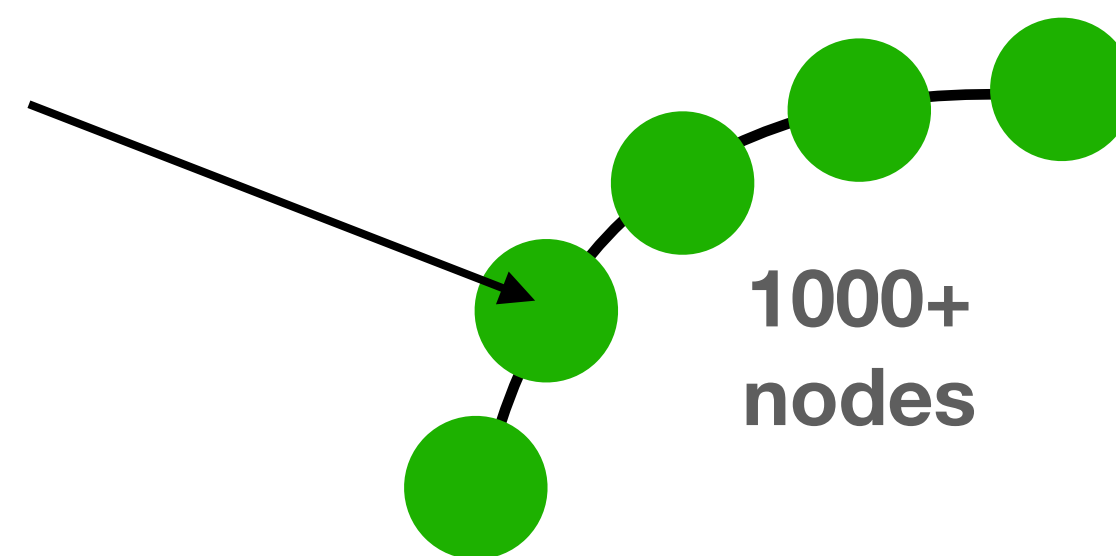


The right way for this scenario is to create a single partition for these 10 users, then read 1 partition

# 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

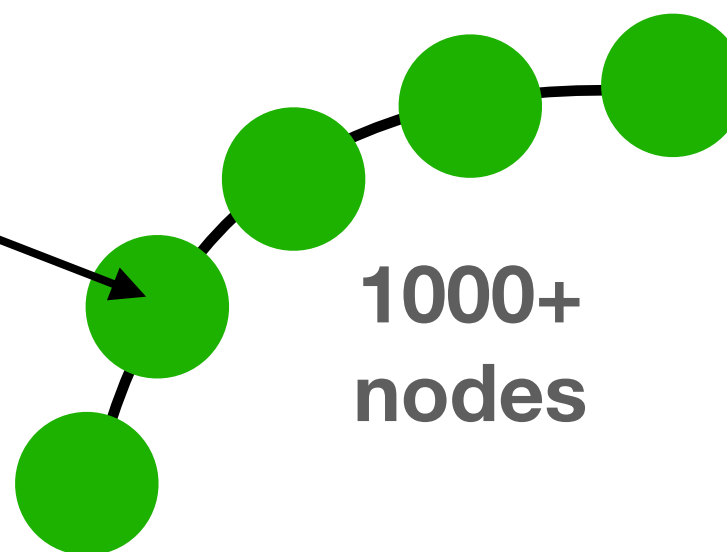
users	
country	K
user_id	▼ C
name	
birth_year	
...	

# 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	
...	

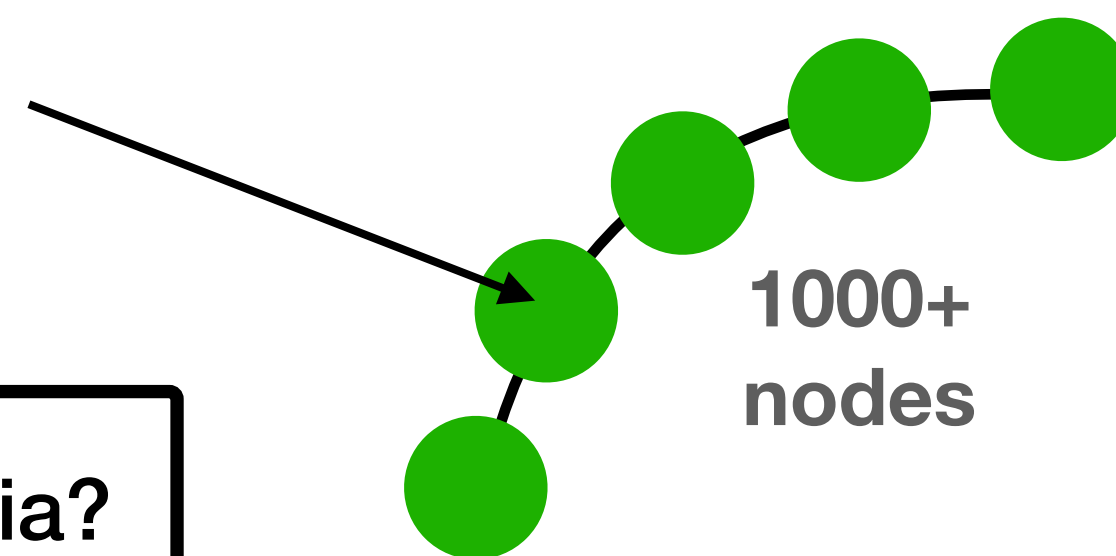
# SELECT - partitions and keys

- 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

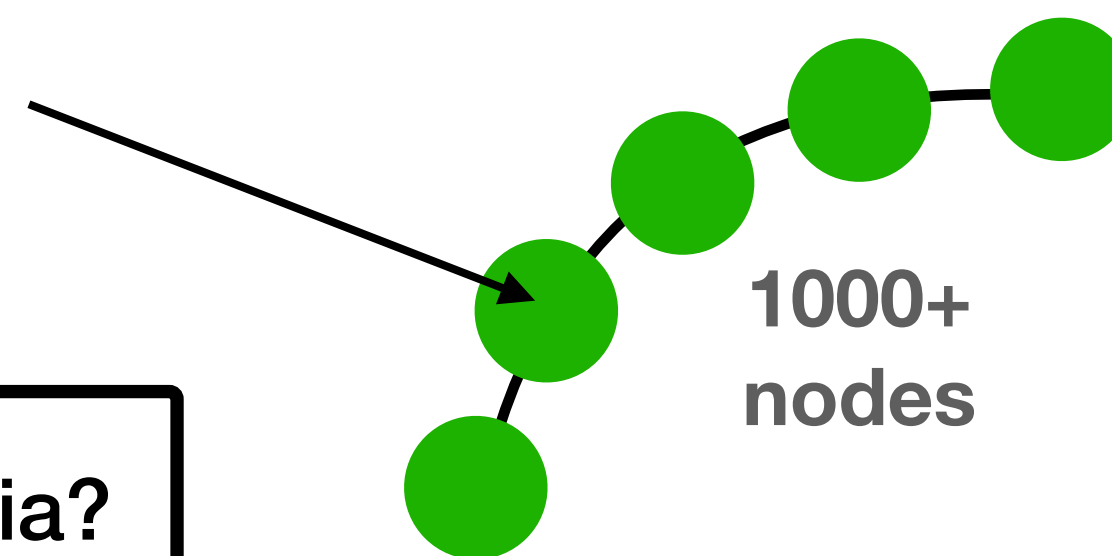
- Try a different model

```
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?



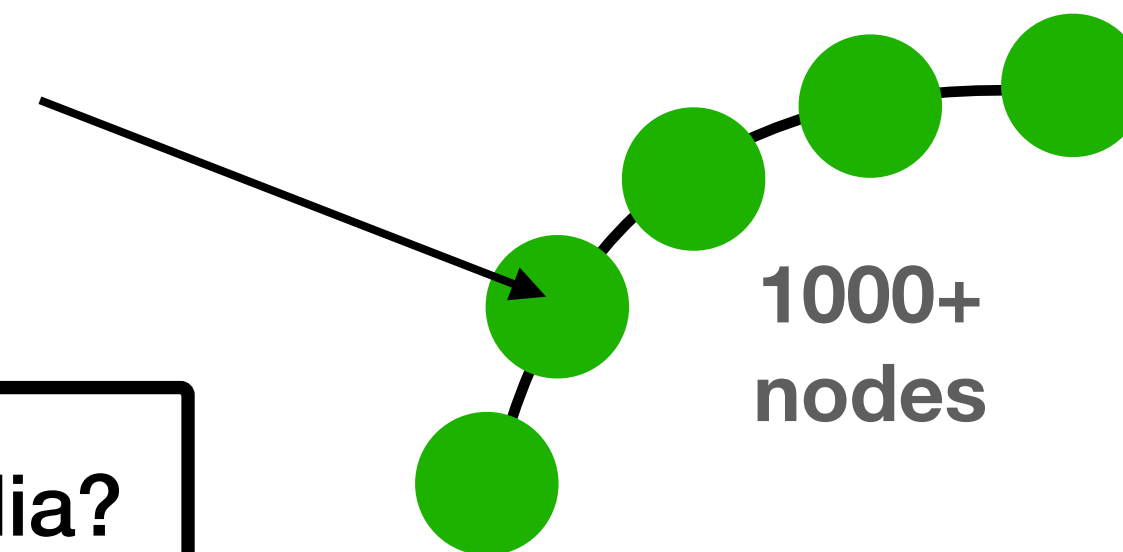
# SELECT - partitions and keys

- 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?

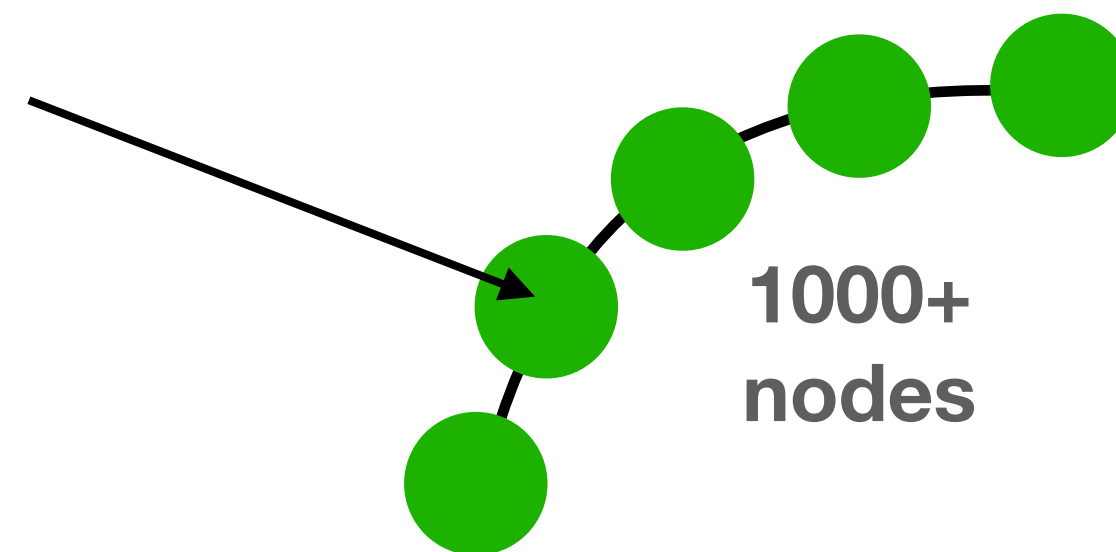
We can add "buckets" - more on this later



# 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?

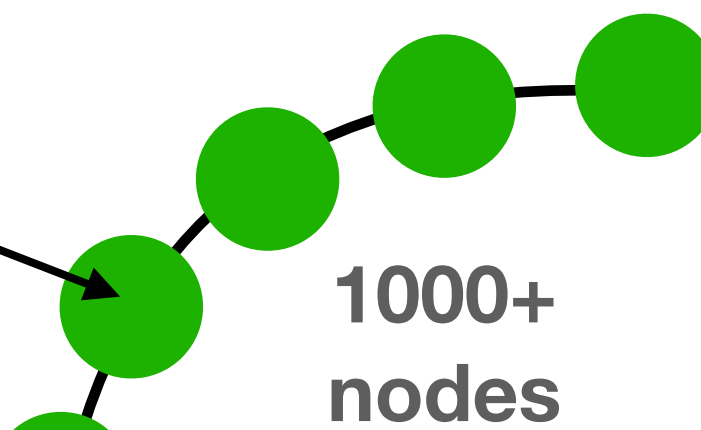
Cassandra will need to read the entire partition.  
If there are 1m users, and only 10k were born in 1982,  
there would be an unnecessary read/filter of 990k users

# 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)

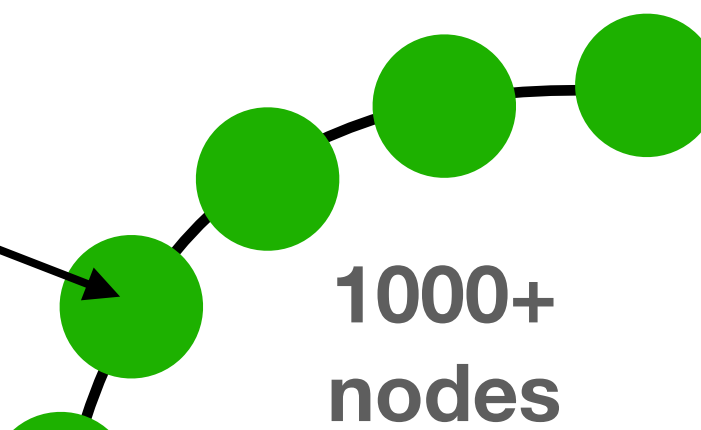
# SELECT - partitions and keys

- 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	

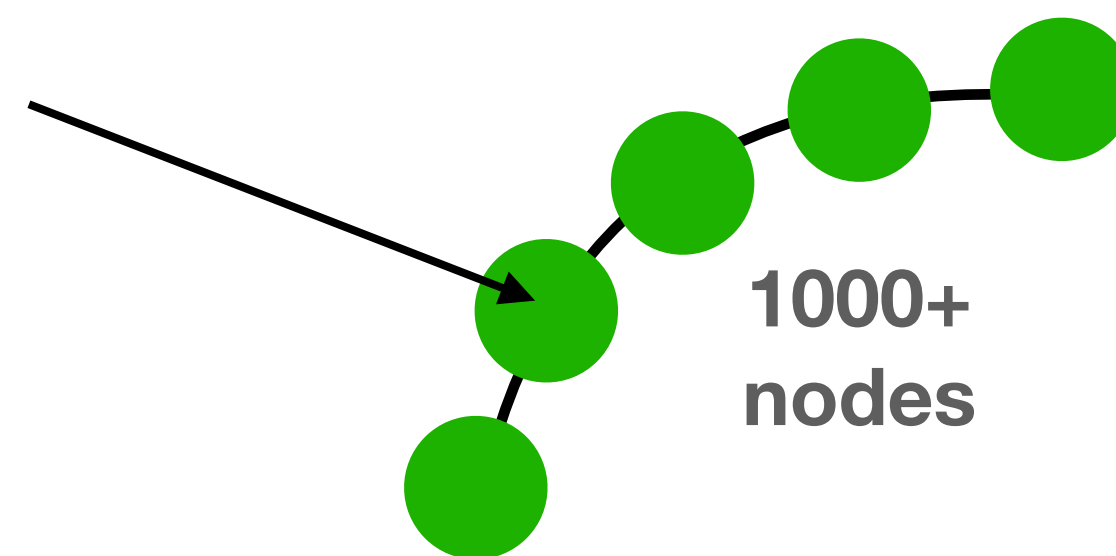


With “ALLOW FILTERING” Cassandra will approve the query  
(ANTI PATTERN)

# 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	
...	

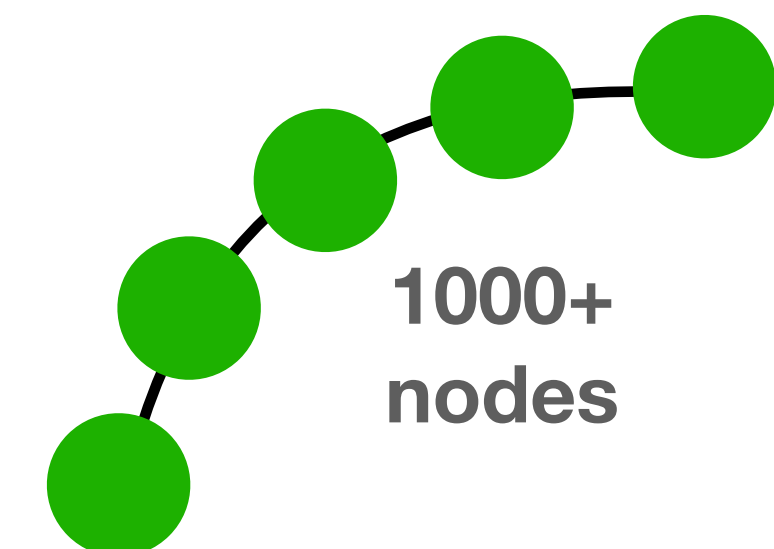
users_by_birth_year	
country	K
birth_year	▼C
user_id	▼C
name	
...	

# 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	



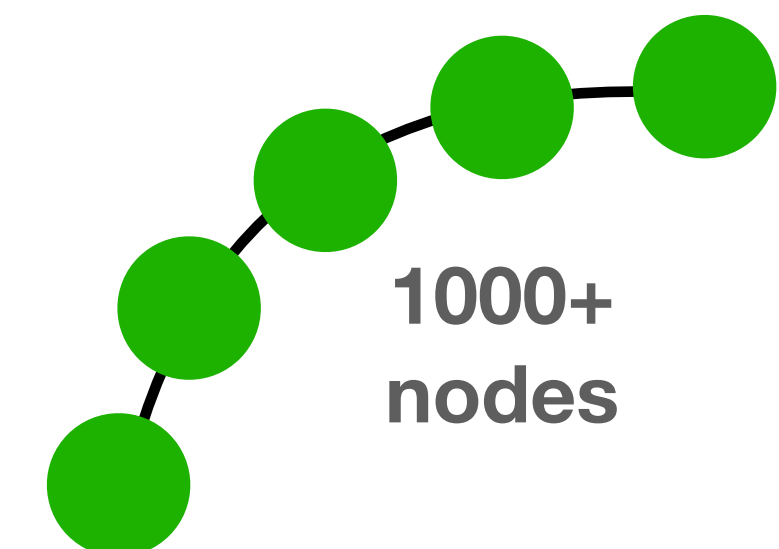
# SELECT - partitions and keys

- 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

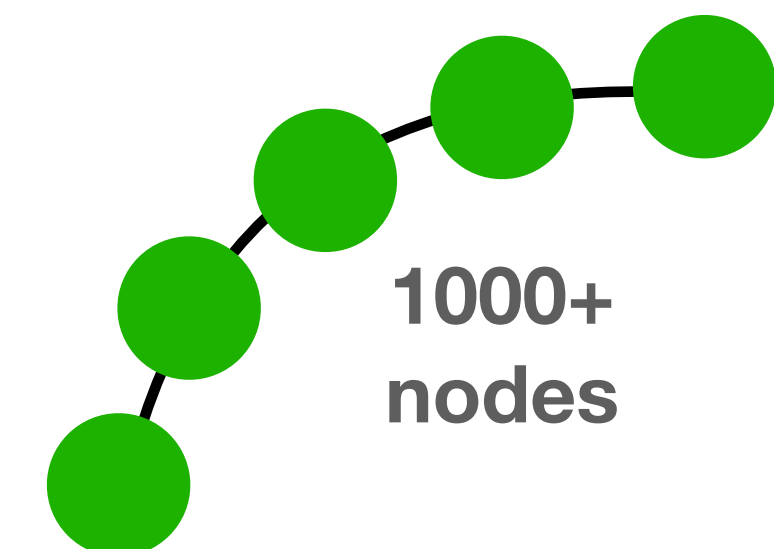
- And what about this case?

```
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



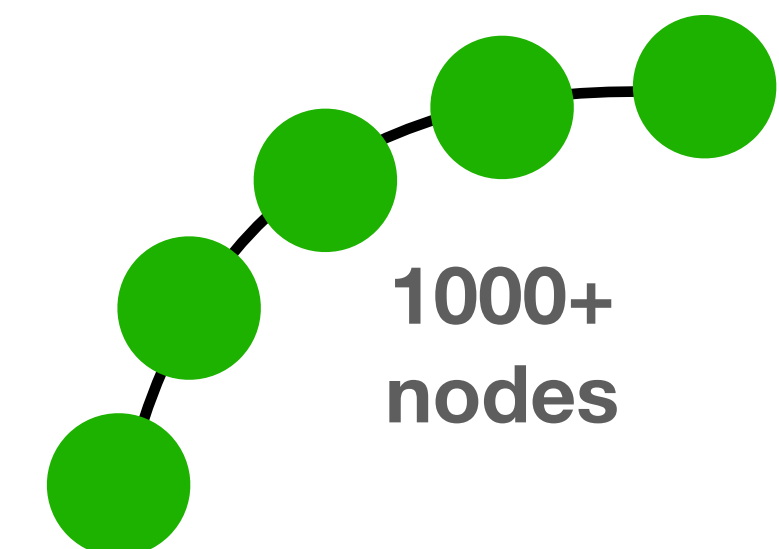
# SELECT - partitions and keys

- 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)

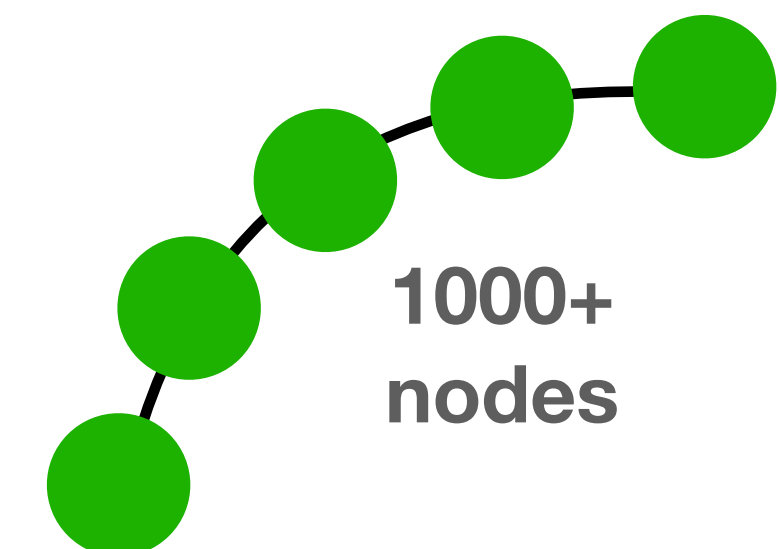
users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	



# SELECT - ALLOW FILTERING

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

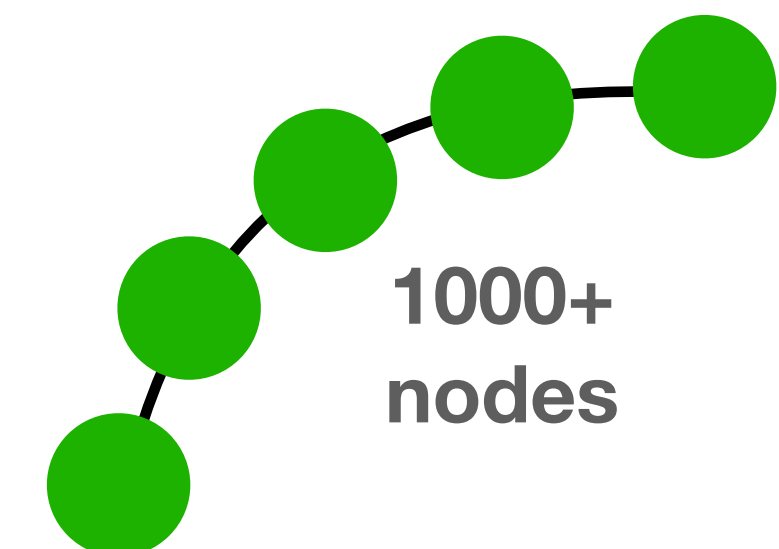
users	
country	K
city	K
neighborhood	K
user_id	▼C
name	
birth_year	



# 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	



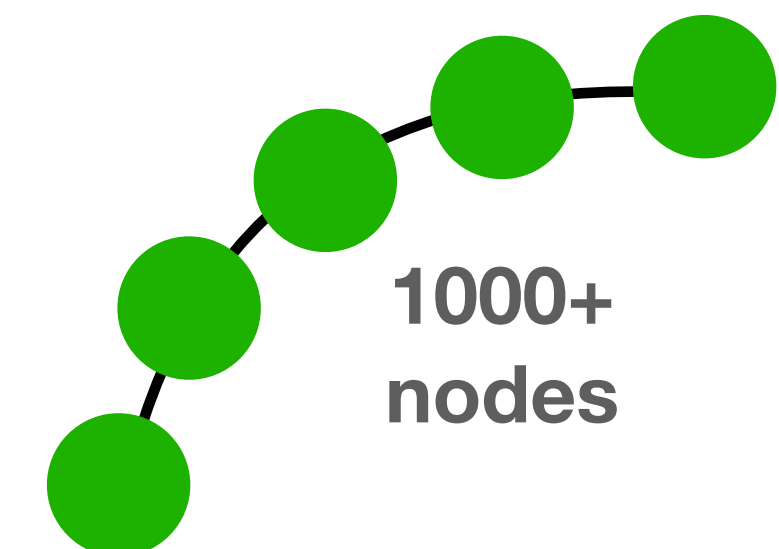
# SELECT - ALLOW FILTERING

- 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 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	
...	

# 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];
```