BlinkDB and G-OLA:

Supporting Approximate Answers in SparkSQL

Sameer Agarwal and Kai Zeng

Spark Summit | San Francisco, CA | June 15th 2015

About Us

1. Sameer Agarwal
   - Software Engineer at Databricks
   - PhD in Databases (UC Berkeley)
   - Research on Approximate Query Processing (BlinkDB)

2. Kai Zeng
   - Post-doc in AMP Lab/ Intern at Databricks
   - PhD in Databases (UCLA)
   - Research on Approximate Query Processing (ABM)

100 TB on 1000 machines

½ - 1 Hour

Hard Disks

→

1 - 5 Minutes

Memory

→

1 second

?

Continuous Query Execution on Samples of Data

Continuous Query Execution on Samples

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

34.6667

Continuous Query Execution on Samples

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

35

Continuous Query Execution on Samples

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

35 ± 2.1

Continuous Query Execution on Samples

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

35 ± 2.1

33.83 ± 1.3

Continuous Query Execution on Samples

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

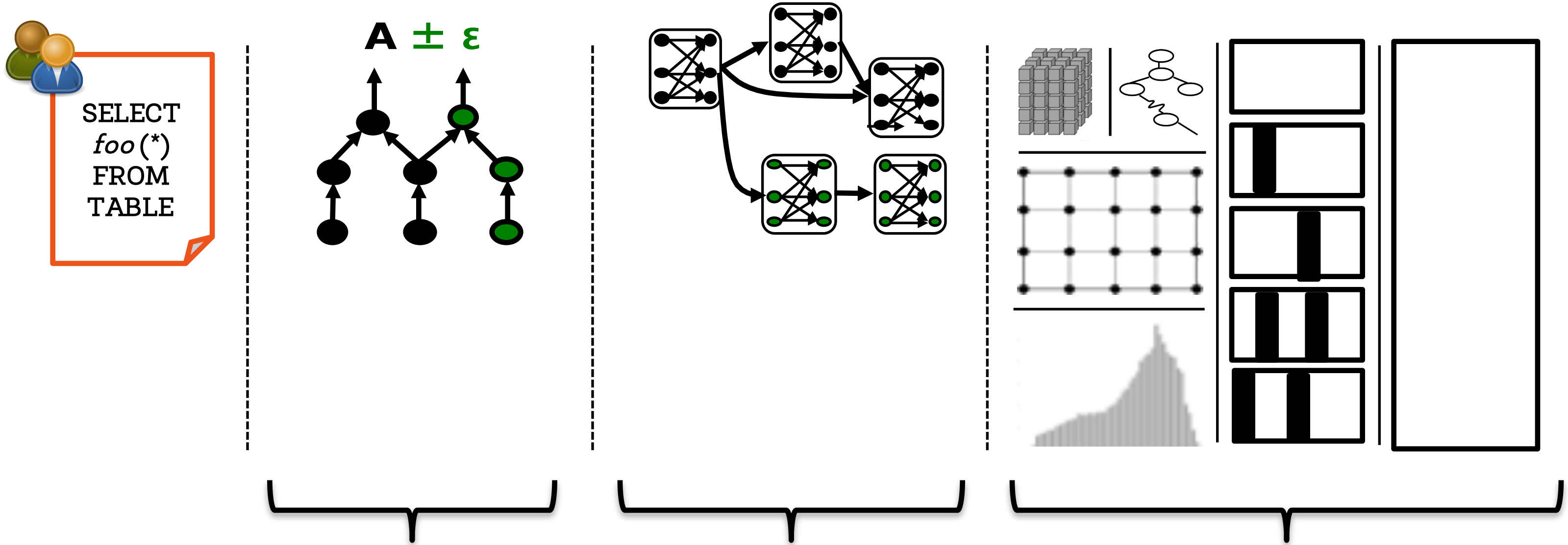
35 ± 2.1

33.83 ± 1.3

34.6667 ± 0.0

Demo

Continuous Query Execution on Samples

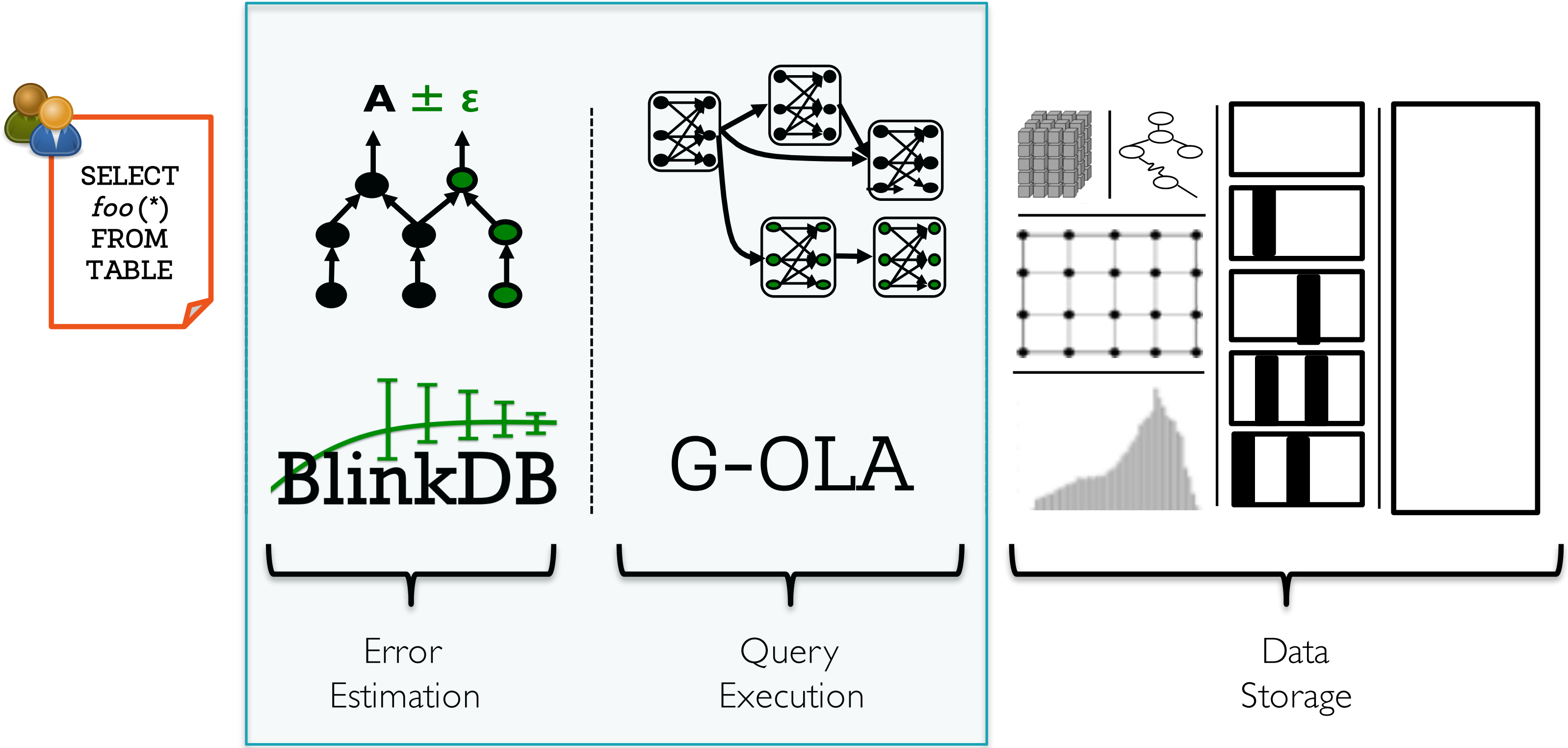


Error
Estimation

Query
Execution

Data
Storage

Continuous Query Execution on Samples



Interface

```
val dataFrame =
    sqlCtx.sql("select avg(latency) from log")

// batch processing
val result = dataFrame.collect() // 34.6667
```

Interface

```
val dataFrame =
    sqlCtx.sql("select avg(latency) from log")

// online processing
val onlineDataFrame = dataFrame.online
onlineDataFrame.collectNext() // 35 ±  2.1
onlineDataFrame.collectNext() // 33.83 ± 1.3
```

Interface

```
val dataFrame =
    sqlCtx.sql("select avg(latency) from log")

// online processing
val onlineDataFrame = dataFrame.online
while (onlineDataFrame.hasNext()) {
  onlineDataFrame.collectNext()
}
```

Interface

```
val dataFrame =
    sqlCtx.sql(“select avg(latency) from log”)

// online processing
val onlineDataFrame = dataFrame.online
while (onlineDataFrame.hasNext() &&
  responseTime <= 10.seconds) {
  onlineDataFrame.collectNext()
}
```

Interface

```
val dataFrame =
    sqlCtx.sql("select avg(latency) from log")

// online processing
val onlineDataFrame = dataFrame.online
while (onlineDataFrame.hasNext() &&
  errorBound >= 0.01) {
  onlineDataFrame.collectNext()
}
```

Interface

```
val dataFrame =
    sqlCtx.sql("select avg(latency) from log")

// online processing
val onlineDataFrame = dataFrame.online
while (onlineDataFrame.hasNext() &&
  userEvent.cancelled()) {
  onlineDataFrame.collectNext()
}
```

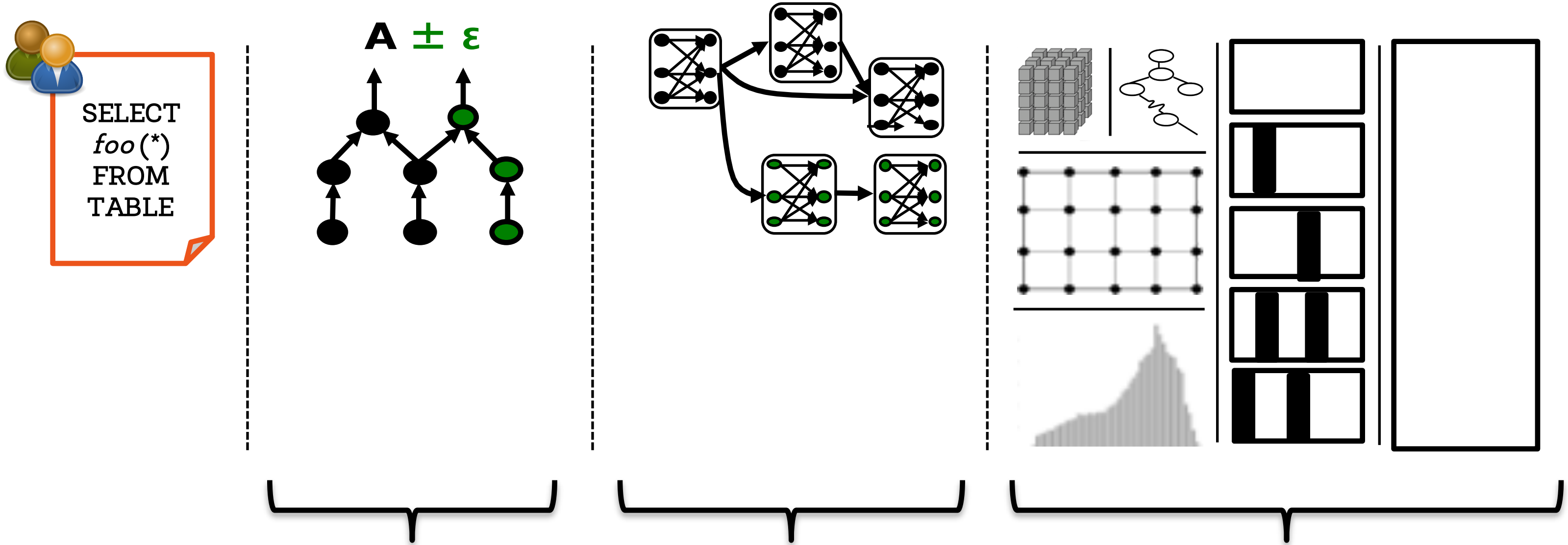
Interface

```
val dataFrame =
    sqlCtx.sql("select avg(latency) from log")

// online processing
val onlineDataFrame = dataFrame.online
while (onlineDataFrame.hasNext() &&
  userEvent.cancelled()) {
  onlineDataFrame.collectNext()
}
```

AGGREGATES/ UDAFs
JOINS/GROUP BYs
NESTED QUERIES

Continuous Query Execution on Samples

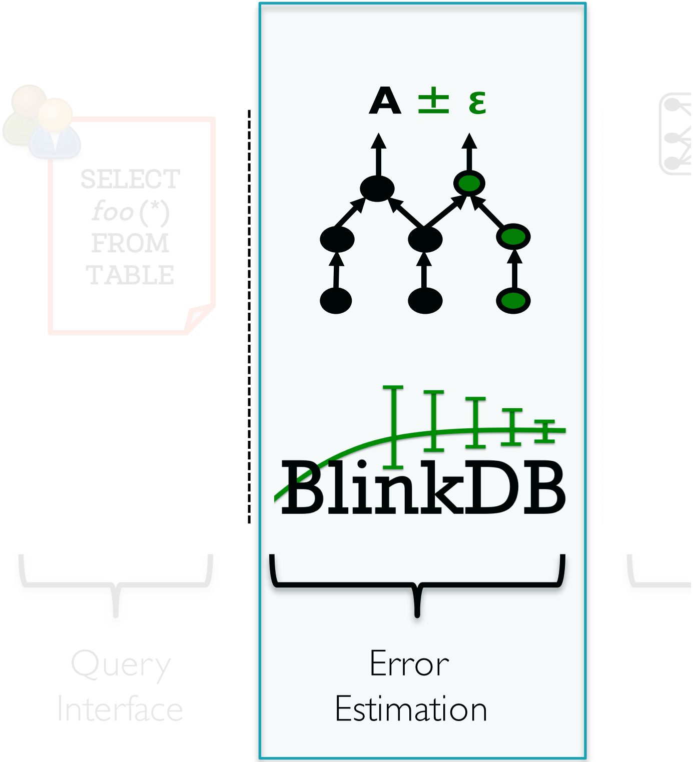


Error Estimation

Query Execution

Data Storage

Continuous Query Execution on Samples



Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, Ion Stoica. **BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data.** In ACM EuroSys 2013.

Ariel Kleiner, Ameet Talwalkar, Sameer Agarwal, Ion Stoica, Michael Jordan. **A General Bootstrap Performance Diagnostic.** In ACM KDD 2013

Sameer Agarwal, Henry Milner, Ariel Kleiner, Ameet Talwalkar,Michael Jordan, Samuel Madden, Barzan Mozafari, Ion Stoica. **Knowing When You're Wrong: Building Fast and Reliable Approximate Query Processing Systems.** In ACM SIGMOD 2014.

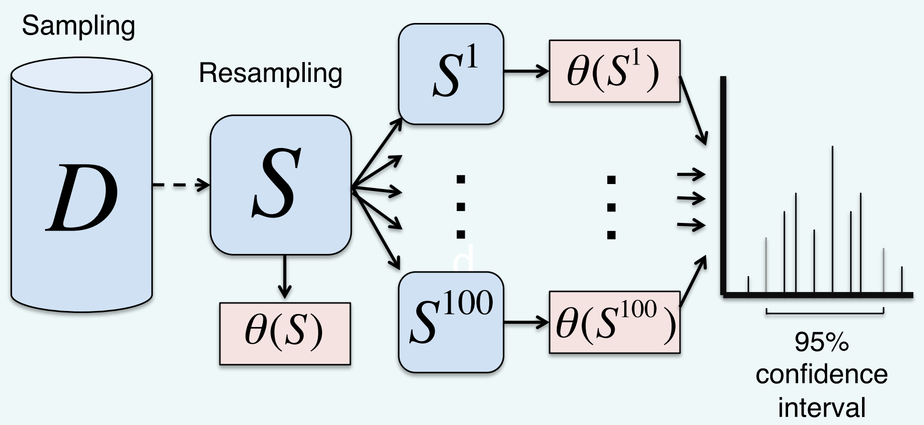
Error Estimation on a Sample of Data

Focused on *estimating aggregate errors* given *representative* samples

1. Count: $N(np, n(1-p)p)$
2. Sum: $N(np\mu, np(\sigma^2 + (1-p)\mu^2))$
3. Mean: $N(\mu, \sigma^2/n)$
4. Variance: $N(\sigma^2, (\mu_4 - \sigma^4)/n)$
5. Stddev: $N(\sigma, (\mu_4 - \sigma^4)/(4\sigma^2 n))$

Central Limit Theorem (CLT)

HOE:ASTAT63, BIL: WILEY86, CGL:ASTAT83, PH:IBM96



Error Estimation using Bootstrap

EFRON:JAS82, EFRON:JAS87, VP:TPMS80, FGK:IJCAI99, ET:CH93

Error Estimation using Bootstrap

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

Error Estimation using Bootstrap

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	SLC	34

$\theta_1 = 34$

ID	City	Latency
1	NYC	30
2	NYC	30
3	SLC	34
4	LA	36

$\theta_2 = 32.5$

…

ID	City	Latency
1	SLC	34
2	LA	36
3	SLC	34
4	LA	36

$\theta_{100} = 35$

34.5 ± 2

Error Estimation using Bootstrap

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	SLC	34

$\theta_1 = 34$

ID	City	Latency
1	NYC	30
2	NYC	30
3	SLC	34
4	LA	36

$\theta_2 = 32.5$

…

ID	City	Latency
1	SLC	34
2	LA	36
3	SLC	34
4	LA	36

$\theta_{100} = 35$

34.5 ± 2

databricks

Error Estimation using Bootstrap

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	SLC	34
5	SLC	37

$\theta_1 = 34.6$

ID	City	Latency
1	SLC	37
2	NYC	30
3	SLC	34
4	LA	36
5	NYC	30

$\theta_2 = 33.4$

…

ID	City	Latency
1	SLC	34
2	SLC	37
3	SLC	34
4	LA	36
5	LA	36

$\theta_{100} = 35.4$

35 ± 1.6

Error Estimation in BlinkDB

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	SLC	34
5	SLC	37

Leverage Poissonized Resampling to generate samples with replacement

Error Estimation in BlinkDB

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

ID	City	Latency	$\#_1$
1	NYC	30	2
2	NYC	38	1
3	SLC	34	0
4	SLC	34	1
5	SLC	37	1

Sample from a Poisson (1) Distribution

$$\theta_1 = 33.8$$

Error Estimation in BlinkDB

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

ID	City	Latency	$\#_1$
1	NYC	30	2
2	NYC	38	1
3	SLC	34	0
4	SLC	34	1
5	SLC	37	1
6	SF	28	2

Incremental Error Estimation

databricks

Error Estimation in BlinkDB

ID	City	Latency
1	NYC	30
2	NYC	38
3	SLC	34
4	LA	36
5	SLC	37
6	SF	28
7	NYC	32
8	NYC	38
9	LA	36
10	SF	35
11	NYC	38
12	LA	34

What is the average latency in the table?

ID	City	Latency	$\#_1$	$\#_2$
1	NYC	30	2	1
2	NYC	38	1	0
3	SLC	34	0	2
4	SLC	34	1	2
5	SLC	37	1	0
6	SF	28	2	1

Construct all Resamples in a Single Pass

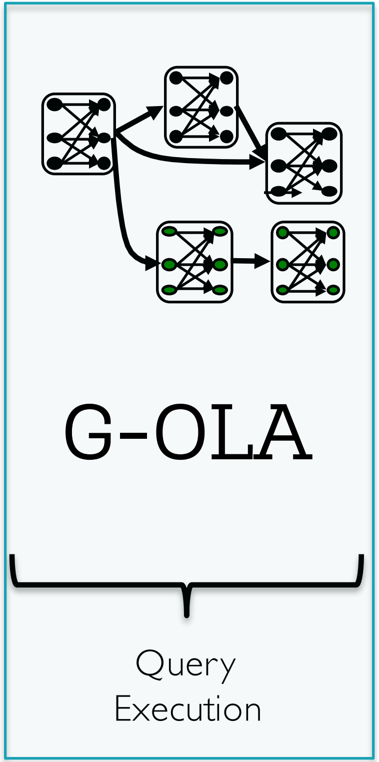
0.2-0.5% additional overhead

High Level Take-away:

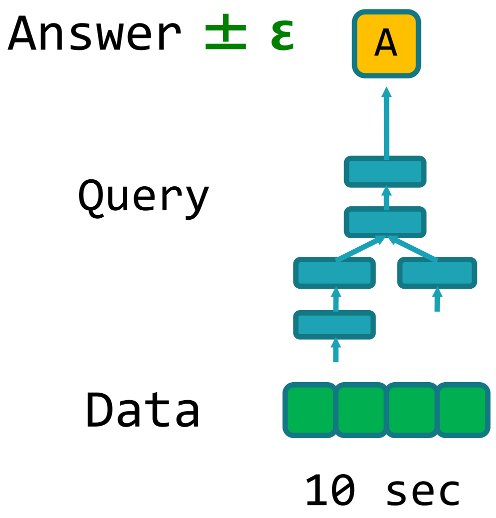
Bootstrap and Poissonized Resampling Techniques are the key towards achieving quick and continuous error bars for a general set of queries

Continuous Query Execution on Samples

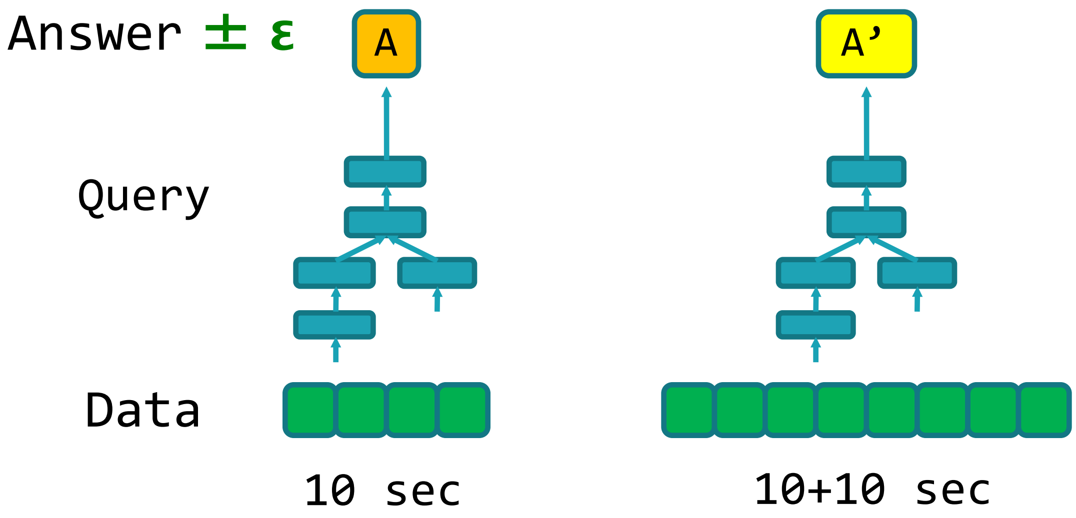
Kai Zeng, Sameer Agarwal, Ankur Dave, Michael Armbrust and Ion Stoica. **G-OLA: Generalized Online Aggregation for Interactive Analysis on Big Data.** In SIGMOD 2015.



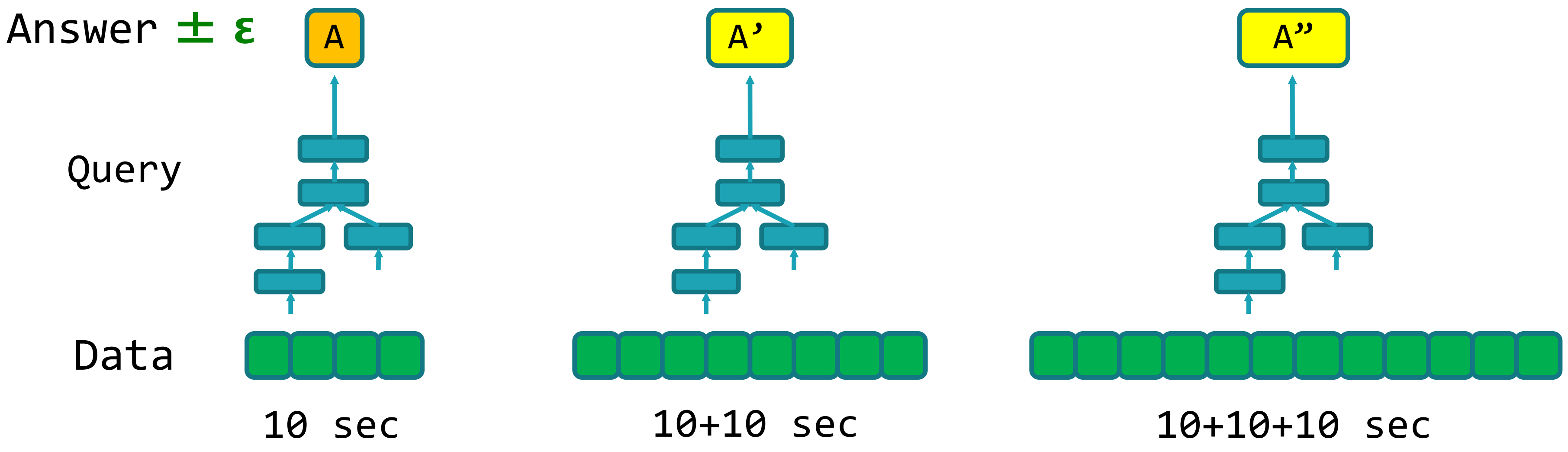
Query Execution: Under The Hood



Query Execution: Under The Hood

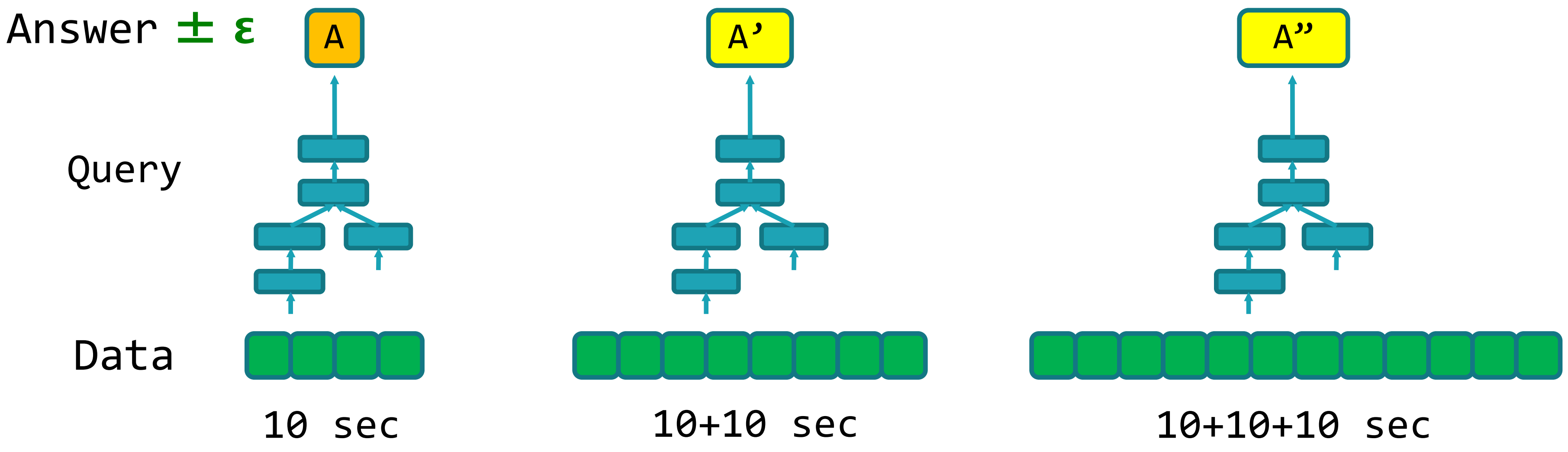


Query Execution: Under The Hood

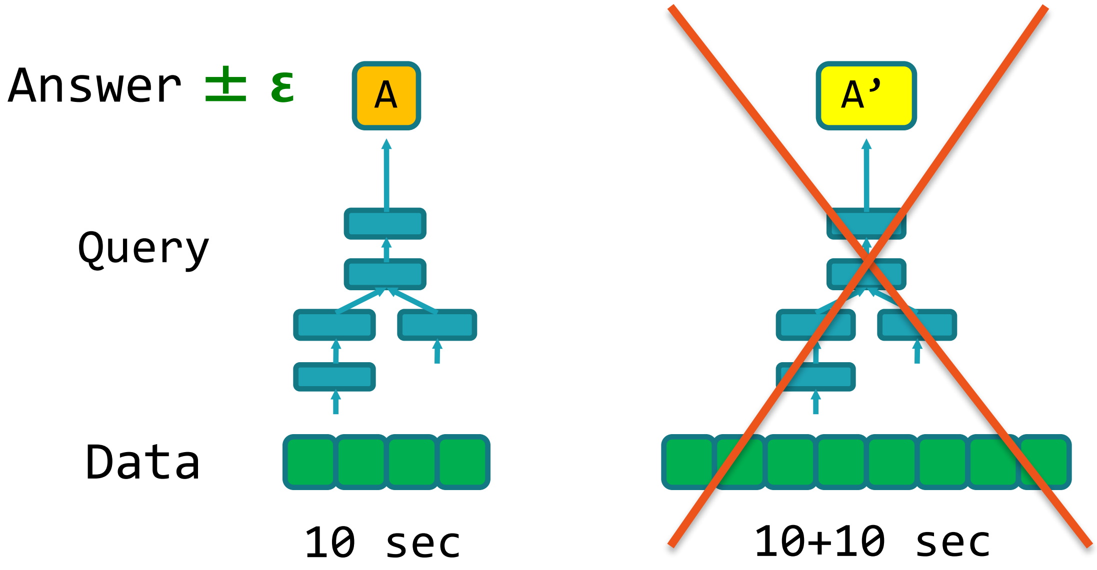


Query Execution: Under The Hood

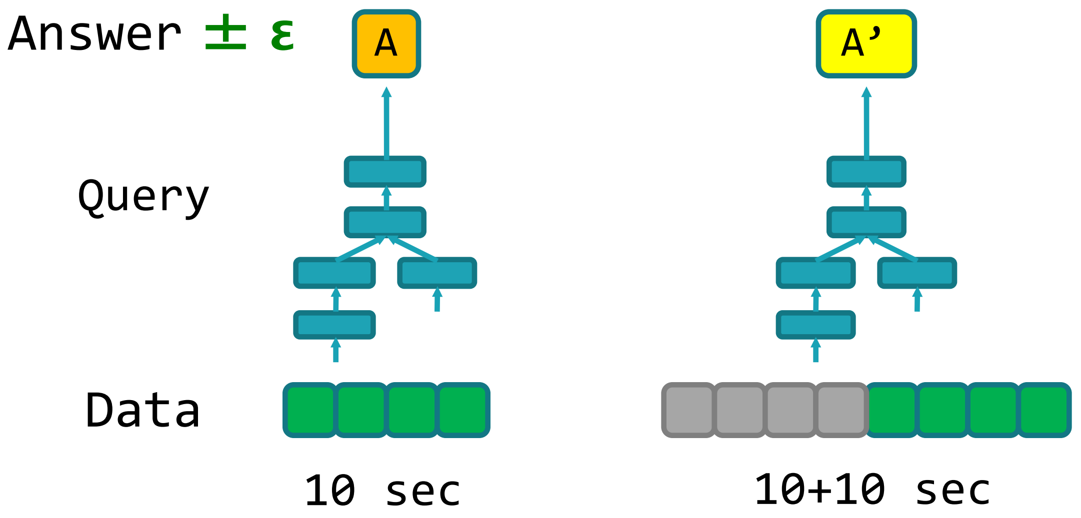
Overall Quadratic Cost!



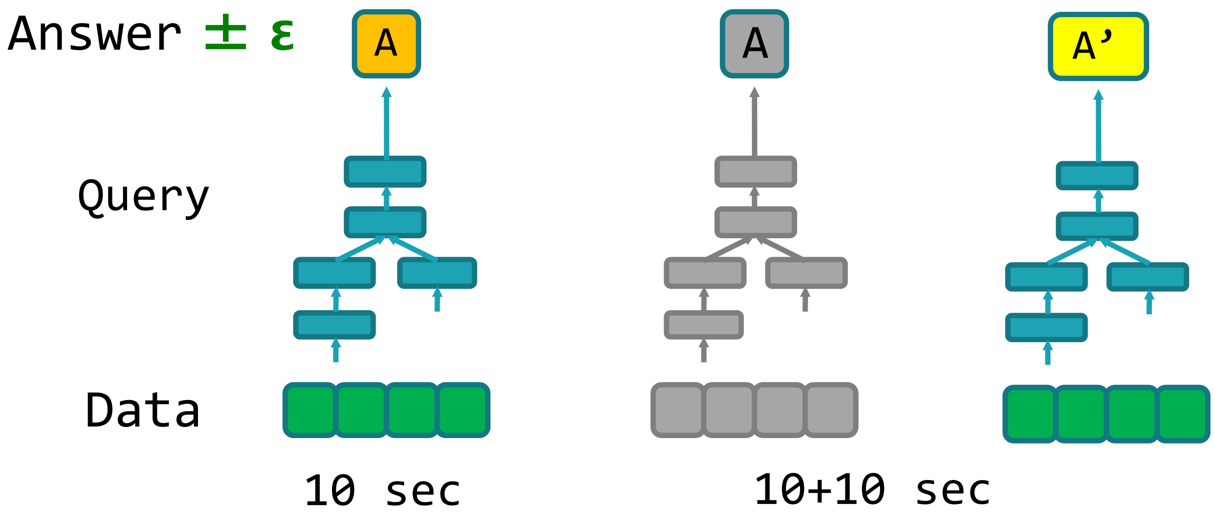
Query Execution: Under The Hood



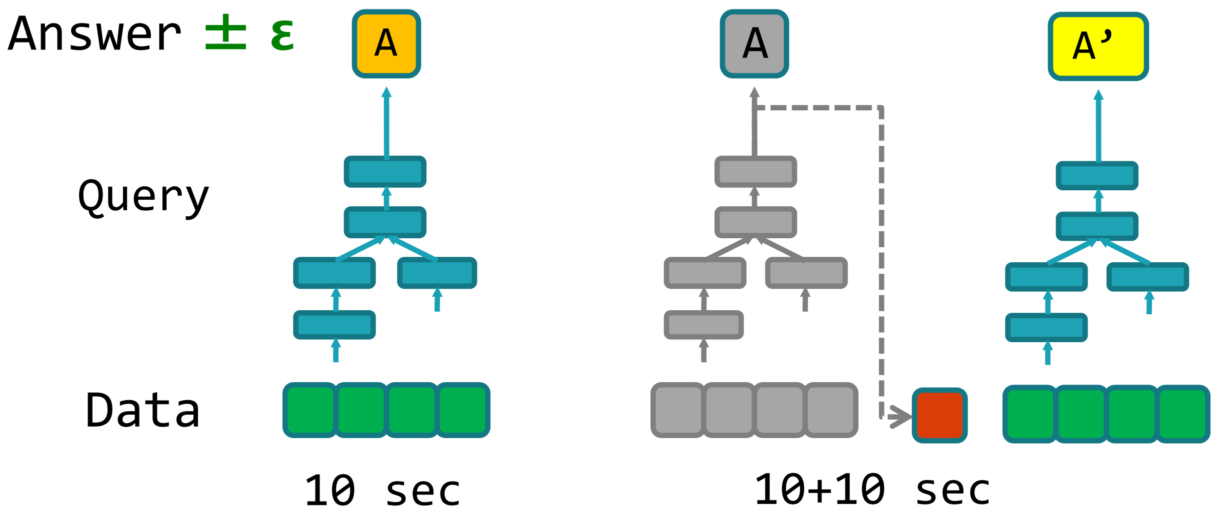
Query Execution: Under The Hood



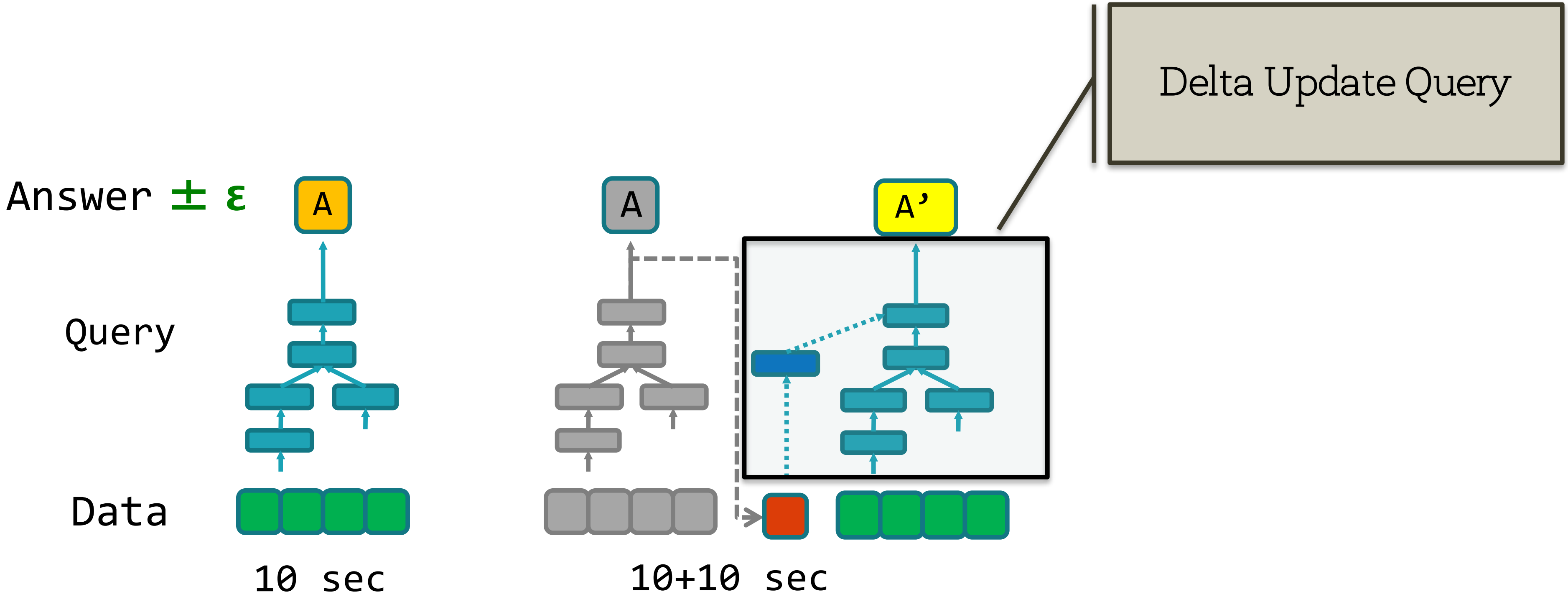
Query Execution: Under The Hood



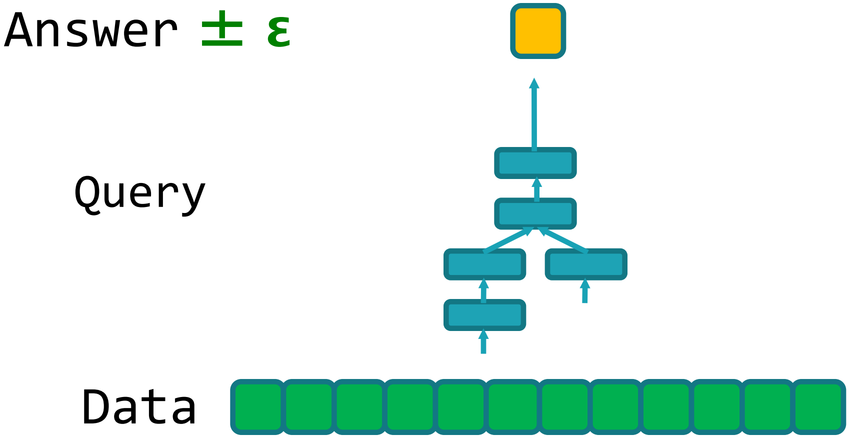
Query Execution: Under The Hood



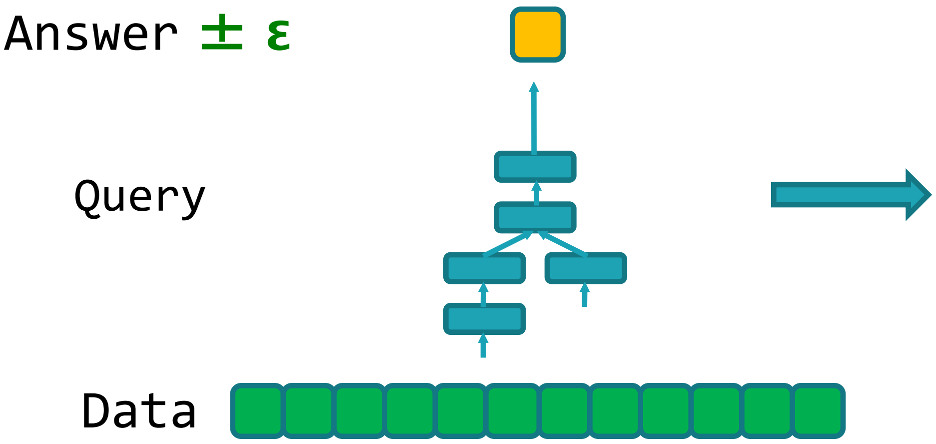
Query Execution: Under The Hood



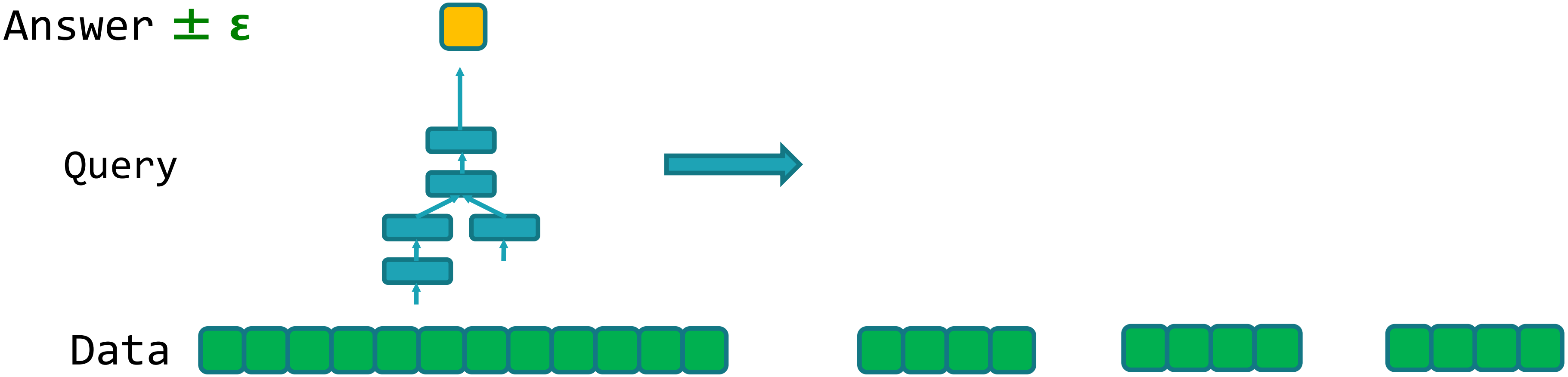
Delta Update Queries



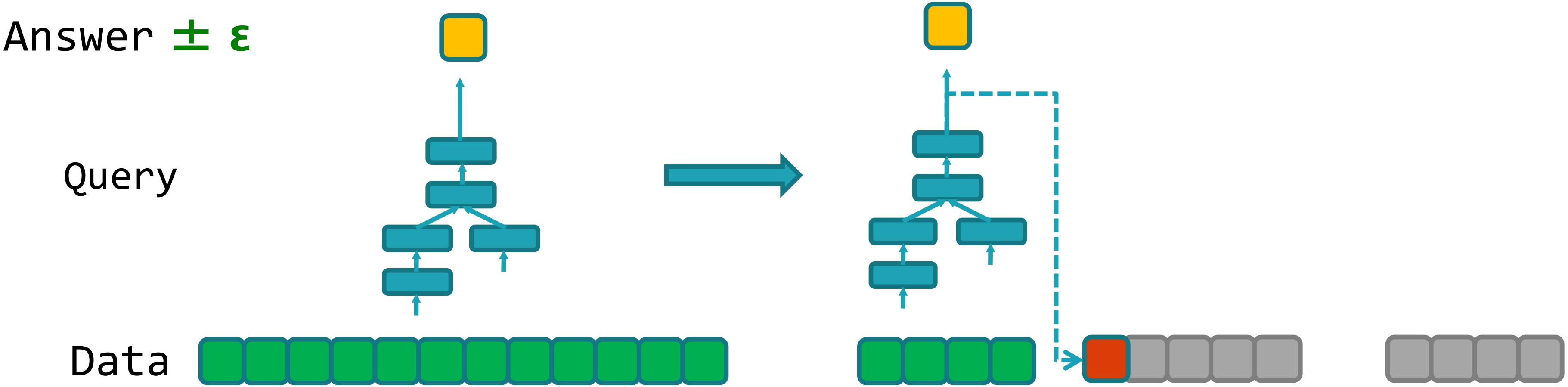
Delta Update Queries



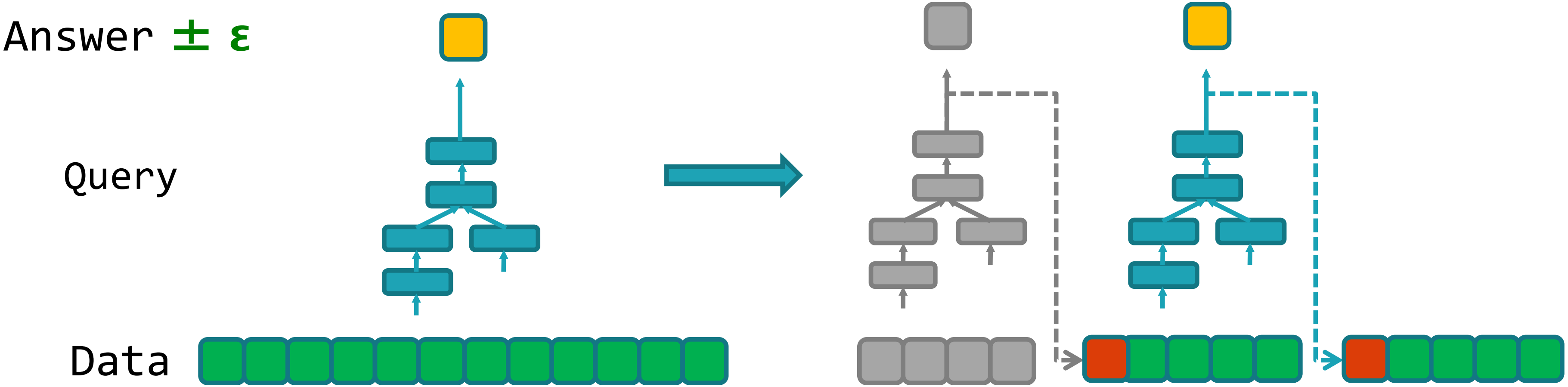
Delta Update Queries



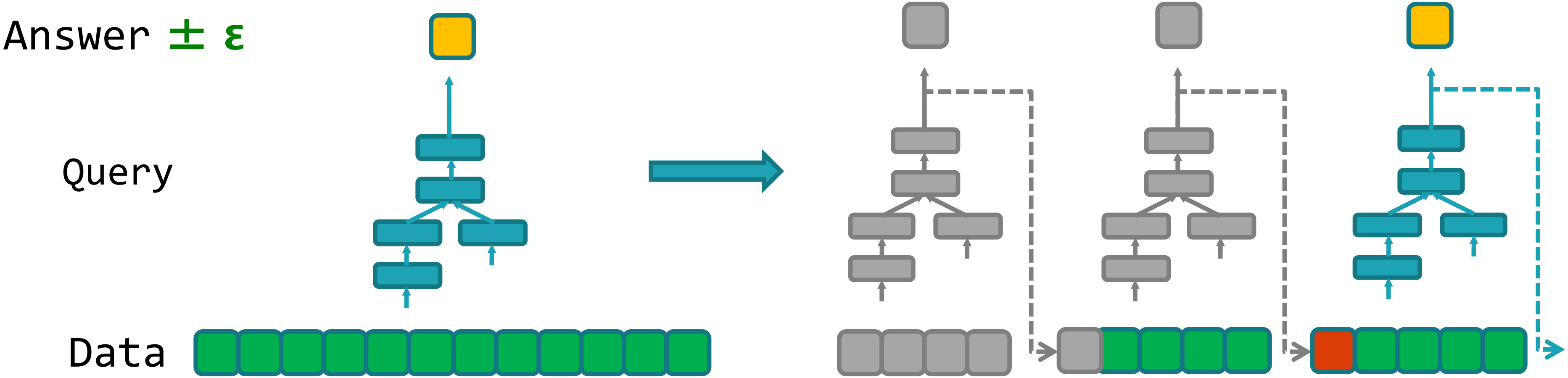
Delta Update Queries



Delta Update Queries

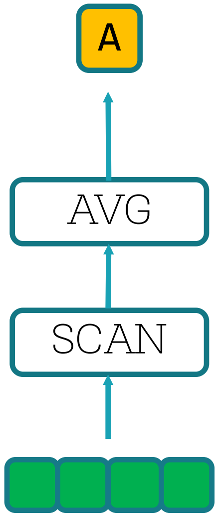


Delta Update Queries



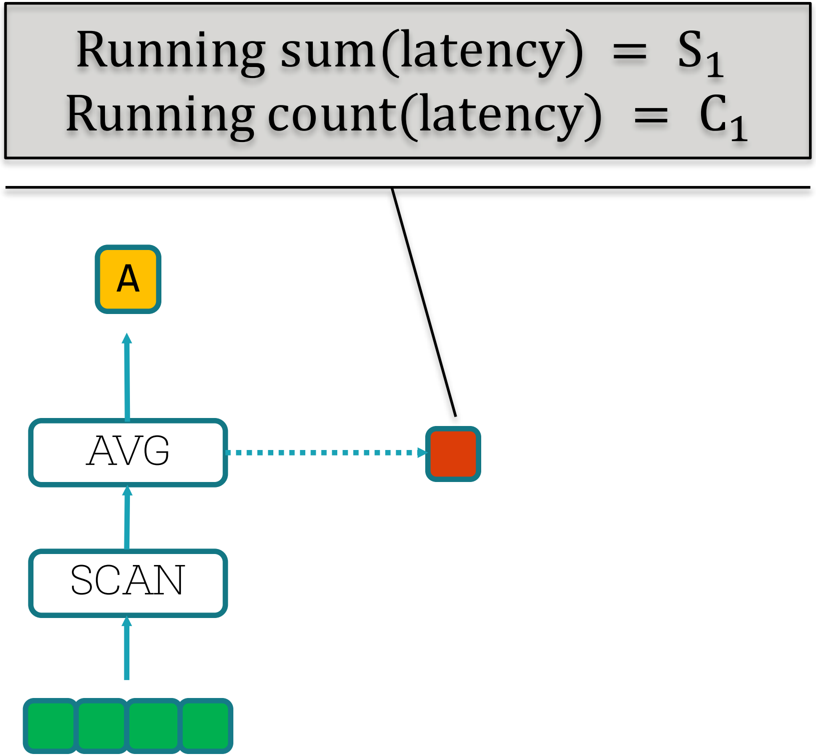
Delta Update: Simple Queries

```
SELECT avg(latency)
FROM log
```



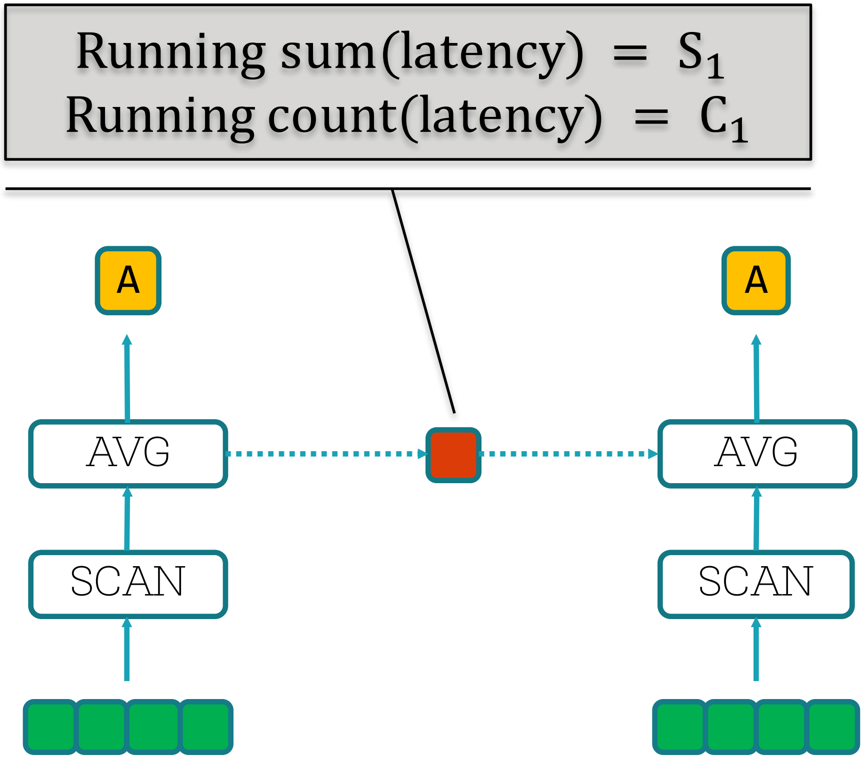
Delta Update: Simple Queries

```
SELECT avg(latency)
FROM log
```



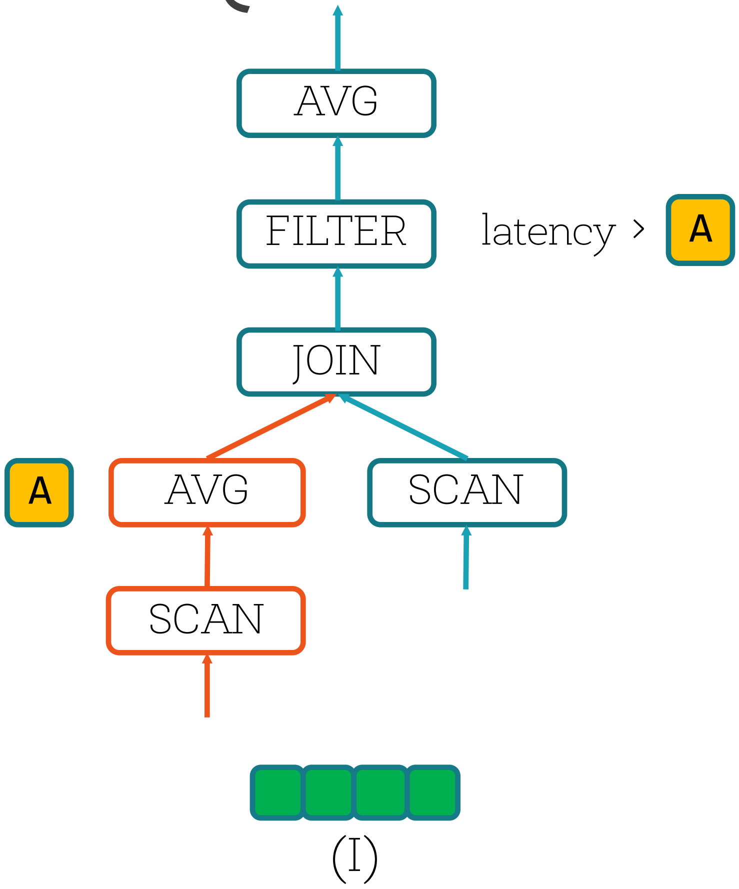
Delta Update: Simple Queries

```
SELECT avg(latency)
FROM log
```



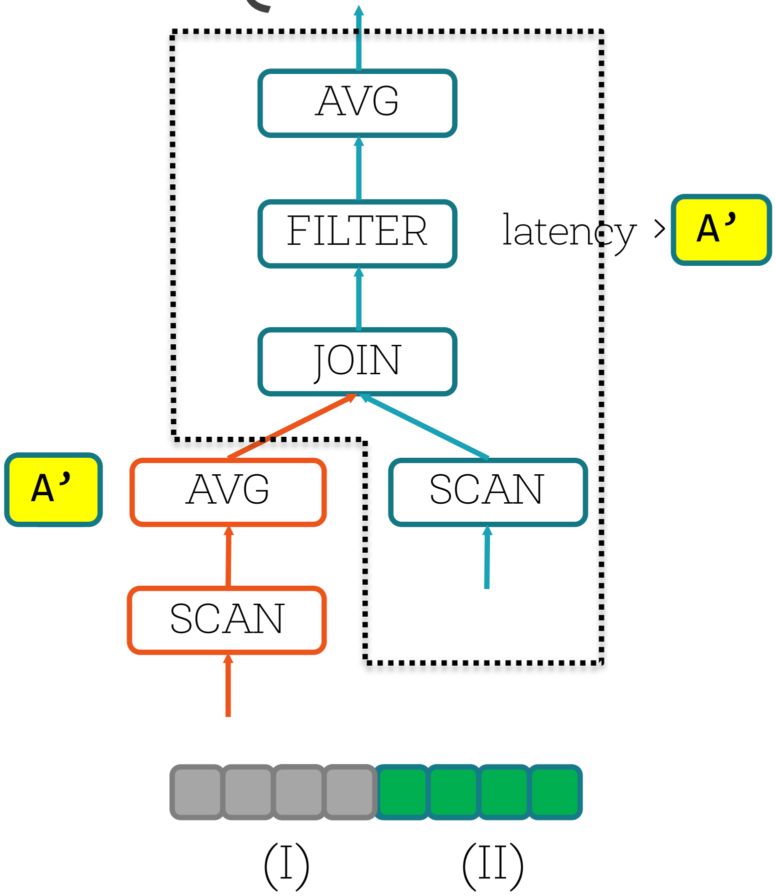
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



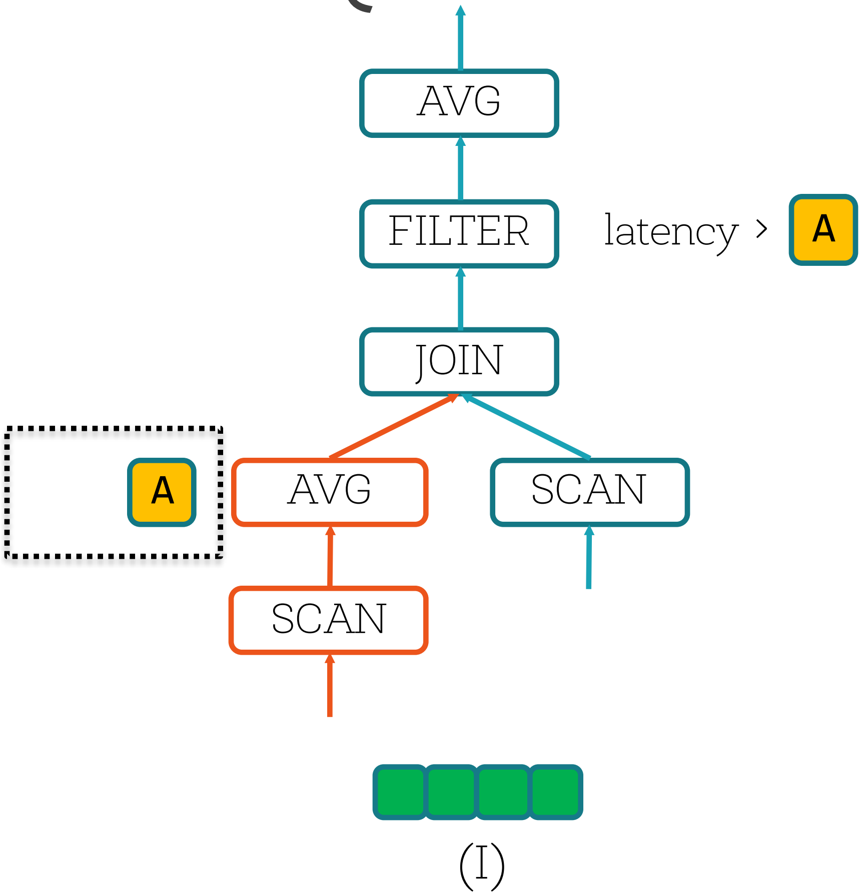
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



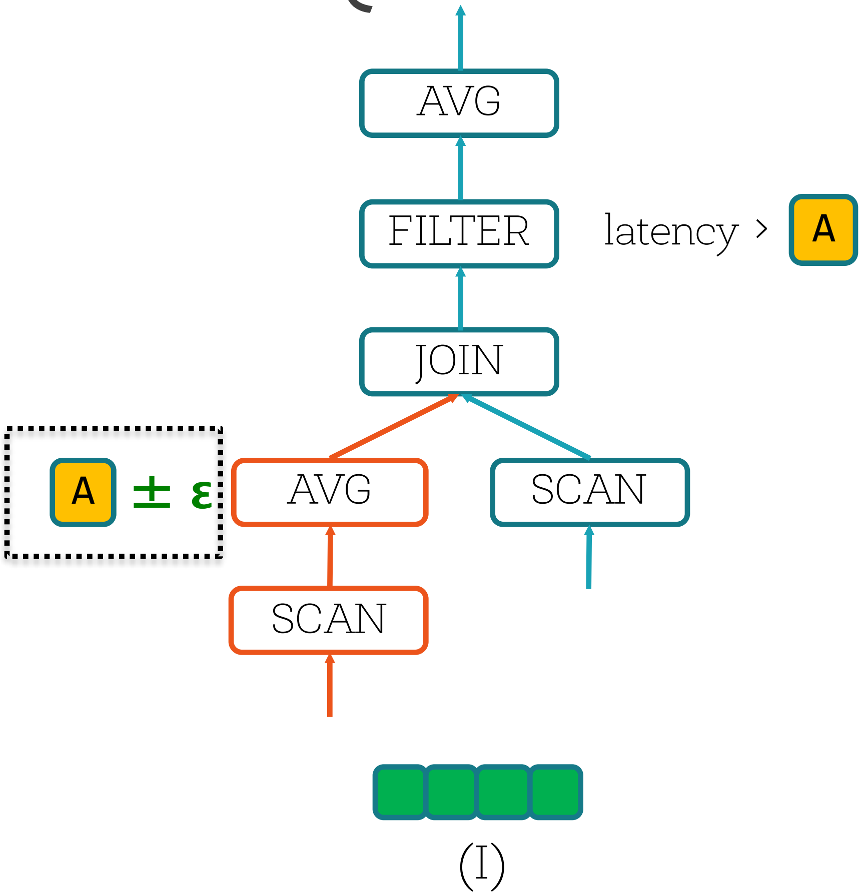
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



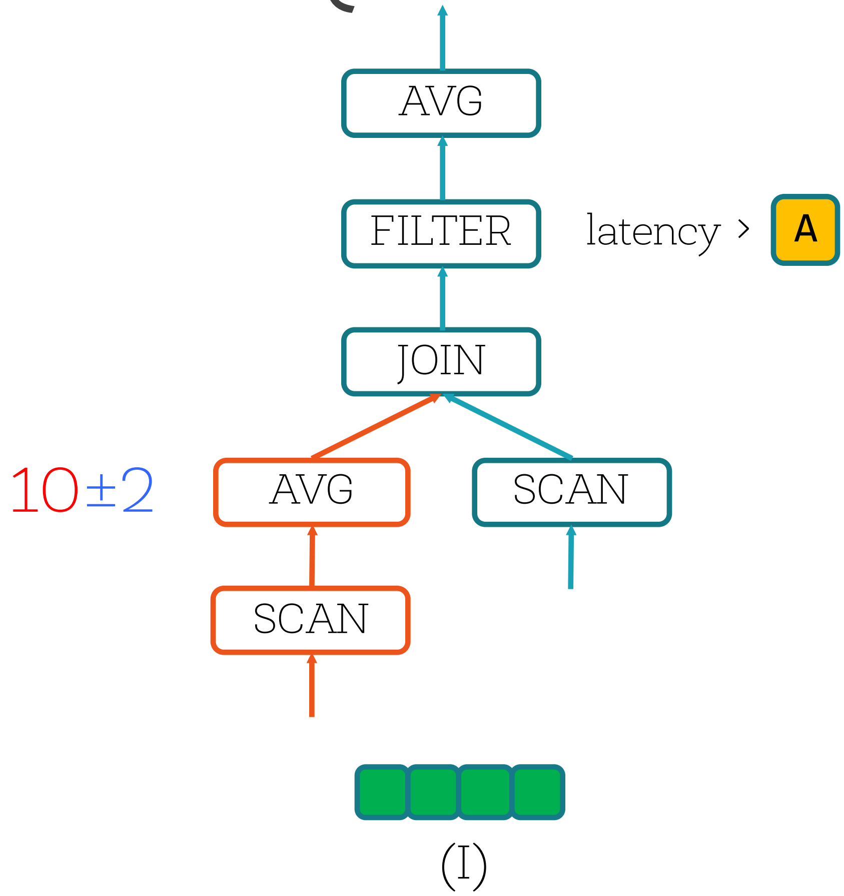
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



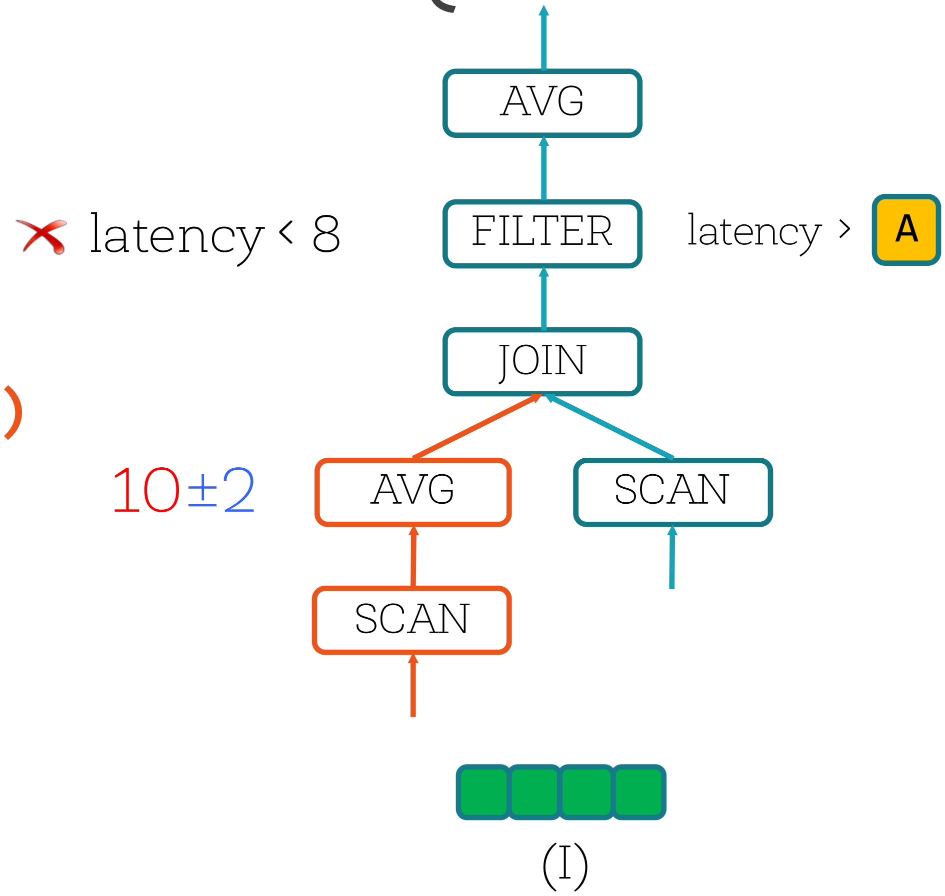
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



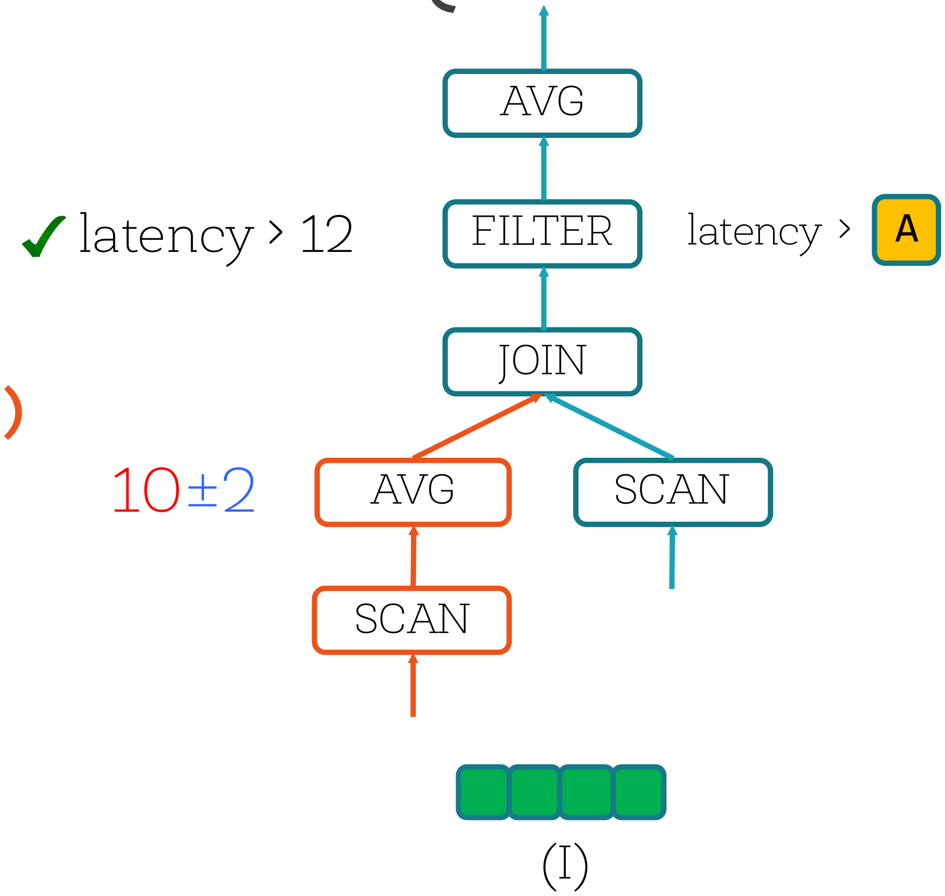
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



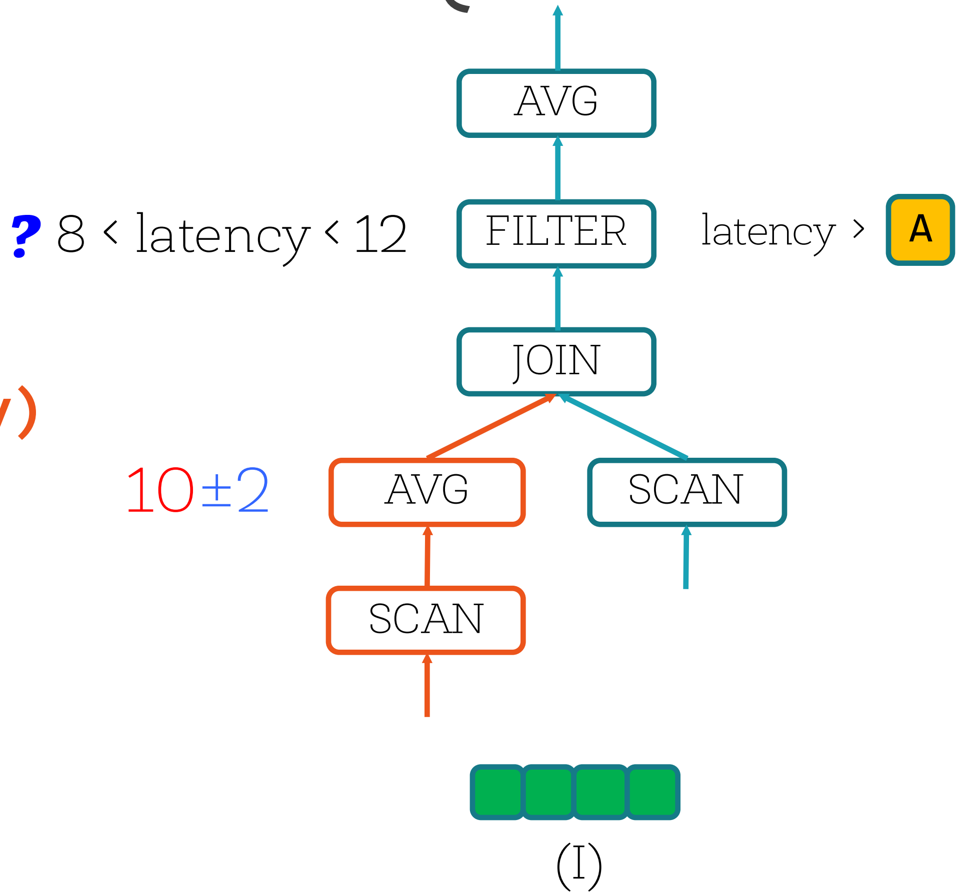
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



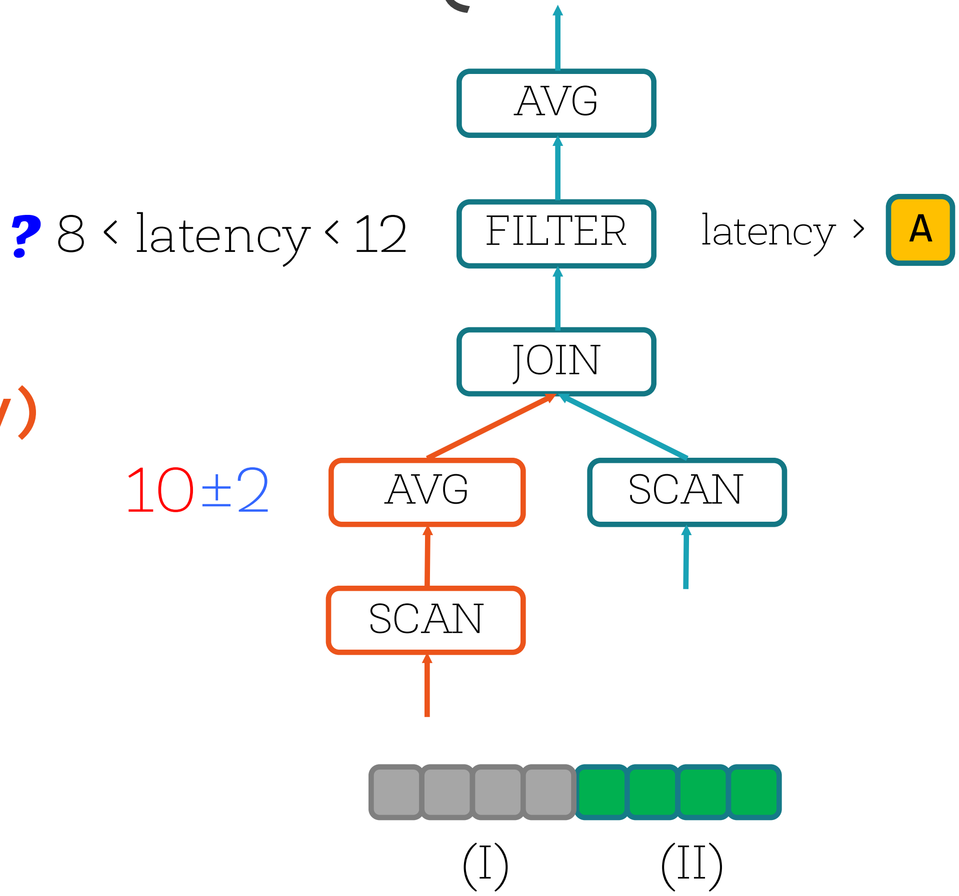
Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



Delta Update: Nested Queries

```
SELECT avg(latency)
FROM log
WHERE latency >
(
  SELECT avg(latency)
  FROM log
)
```



High Level Take-away:

Introduce Delta Update Queries as a First Class Citizen in Query Execution

Check out our code!

1. **Code Preview:** http://github.com/amplab/bootstrap-sql. Send us an email to kaizeng@cs.berkeley.edu and sameer@databricks.com to get access!

2. Spark Package in July'15

3. Gradual Native SparkSQL Integration in 1.5, 1.6 and beyond

Conclusion

1. Continuous Query Execution on Samples of Data is an important means to achieve interactivity in processing large datasets

2. New SparkSQL Libraries:
   - BlinkDB for Continuous Error Bars
   - G-OLA for Continuous Partial Answers

Thank you.

Sameer Agarwal (sameer@databricks.com)
Kai Zeng (kaizeng@cs.berkeley.edu)

databricks™