Big Data Summer School

Approximate Query Processing

Our Goal

Support **interactive** SQL-like aggregate queries over **massive sets of data**

Our Goal

Support **interactive** SQL-like aggregate queries over **massive sets of data**

- blinkdb> SELECT AVG(jobtime)
- FROM very_big_log

AVG, COUNT, SUM, STDEV, PERCENTILE etc.

Our Goal

Support **interactive** SQL-like aggregate queries over **massive sets of data**

```
blinkdb> SELECT AVG(jobtime)
                 FROM very_big_log
                 WHERE src = ‘hadoop’
```

FILTERS, GROUP BY clauses

Our Goal

Support **interactive** SQL-like aggregate queries over **massive sets of data**

```
blinkdb> SELECT AVG(jobtime)
                    FROM very_big_log
                    WHERE src = 'hadoop'
                    LEFT OUTER JOIN logs2
ON very_big_log.id = logs.id
```

JOINS, Nested Queries etc.

Our Goal

Support **interactive** SQL-like aggregate queries over **massive sets of data**

```
blinkdb> SELECT my_function(jobtime)
         FROM very_big_log
         WHERE src = 'hadoop'
         LEFT OUTER JOIN logs2
         ON very_big_log.id = logs.id
```

ML Primitives, User Defined Functions

100 TB on 1000 machines

½ - 1 Hour

Hard Disks

1 - 5 Minutes

Memory

1 second

?

Query Execution on Samples

Query Execution on Samples

ID	City	Buff Ratio
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

What is the average <u>buffering ratio</u> in the table?

0.2325

Query Execution on Samples

ID	City	Buff Ratio
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

What is the average buffering ratio in the table?

Uniform Sample

ID	City	Buff Ratio	Sampling Rate
2	NYC	0.13	1/4
6	Berkeley	0.25	1/4
8	NYC	0.19	1/4

~~0.2325~~

0.19

Query Execution on Samples

ID	City	Buff Ratio
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

What is the average <u>buffering ratio</u> in the table?

Uniform Sample

ID	City	Buff Ratio	Sampling Rate
2	NYC	0.13	1/4
6	Berkeley	0.25	1/4
8	NYC	0.19	1/4

~~0.2325~~

0.19 +/- 0.05

Query Execution on Samples

ID	City	Buff Ratio
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

Uniform Sample

What is the average <u>buffering ratio</u> in the table?

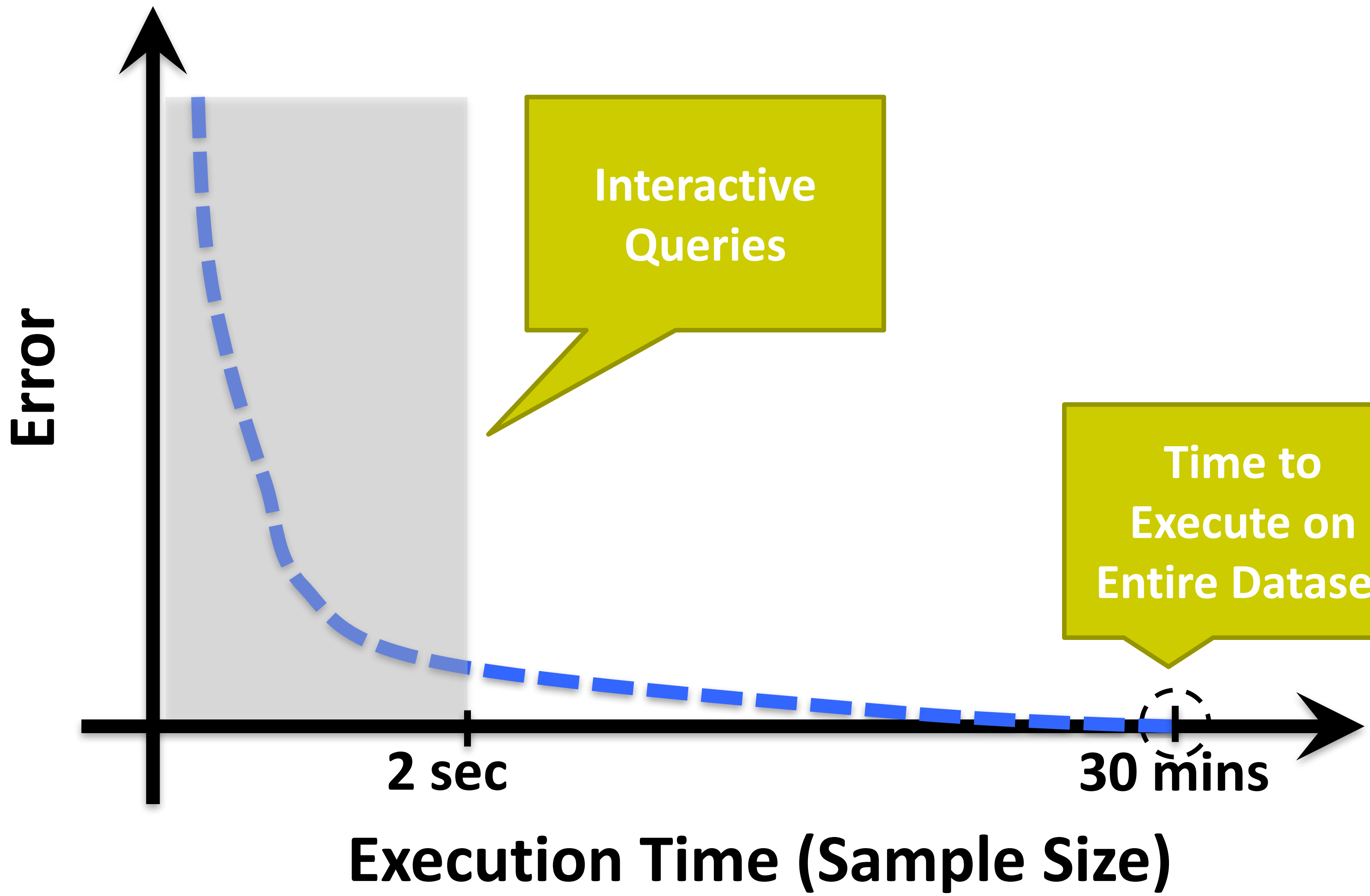
ID	City	Buff Ratio	Sampling Rate
2	NYC	0.13	1/2
3	Berkeley	0.25	1/2
5	NYC	0.19	1/2
6	Berkeley	0.09	1/2
8	NYC	0.18	1/2
12	Berkeley	0.49	1/2

~~0.2325~~

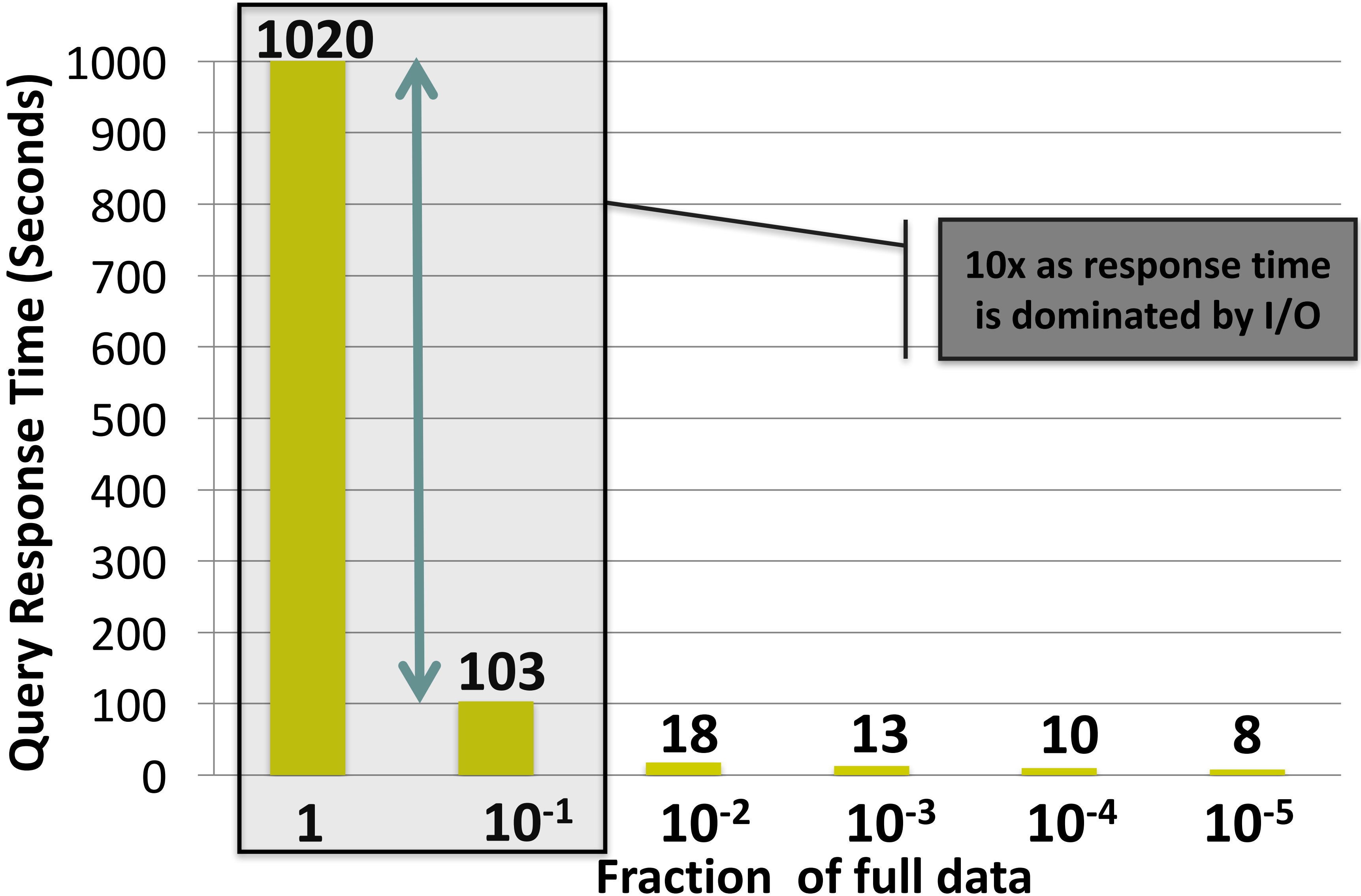
~~0.19 +/- 0.05~~

$0.22 +/- 0.02

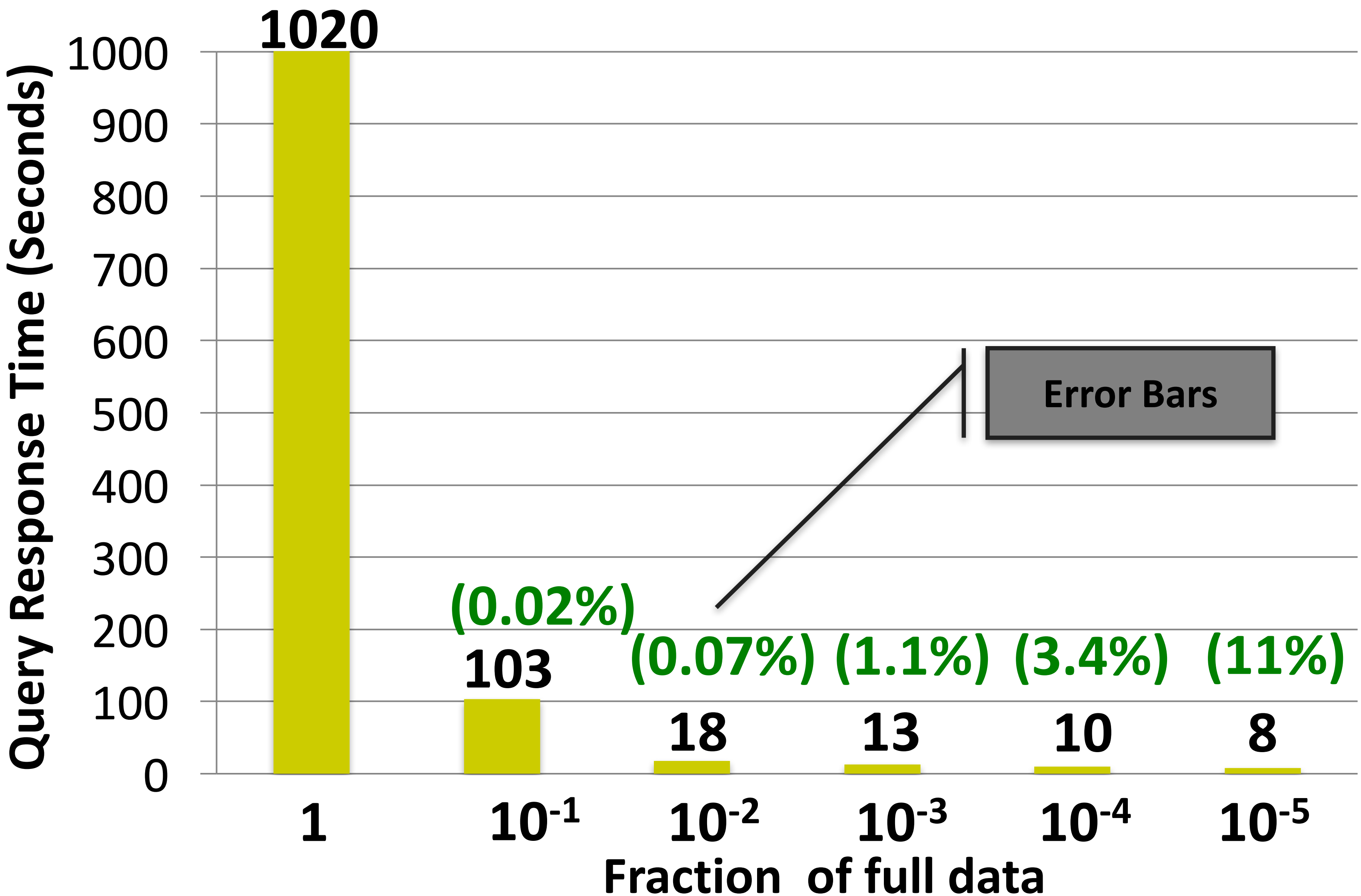
Speed/Accuracy Trade-off



Sampling Vs. No Sampling



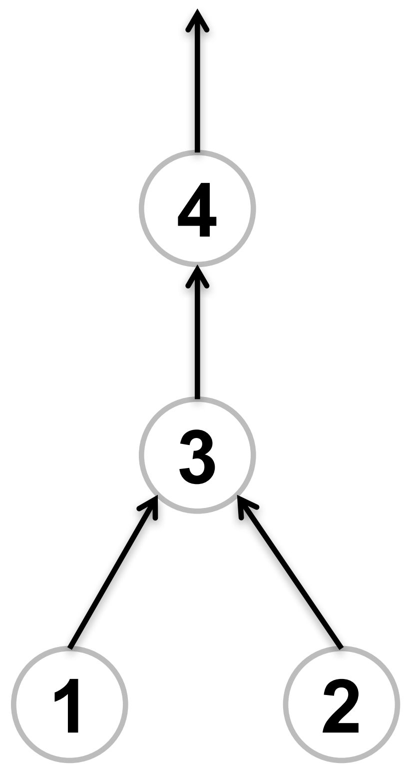
Sampling Vs. No Sampling



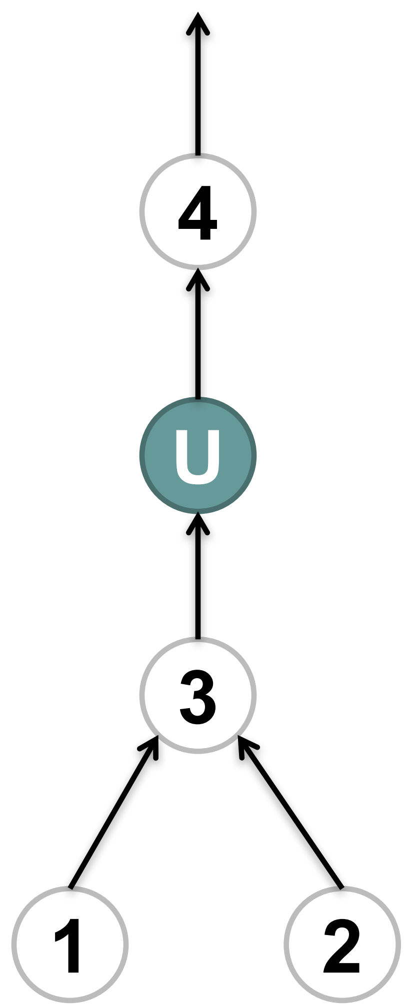
What is BlinkDB?

- A framework built on Shark and Spark that …
  - creates and maintains a variety of uniform and stratified samples from underlying data
  - returns fast, approximate answers with error bars by executing queries on samples of data
  - verifies the correctness of the error bars that it returns at runtime

Uniform Samples

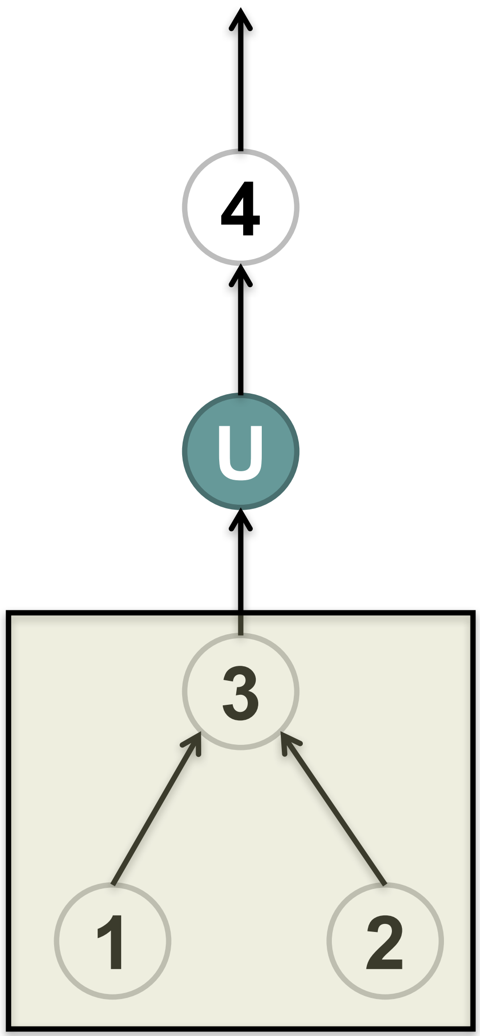


Uniform Samples



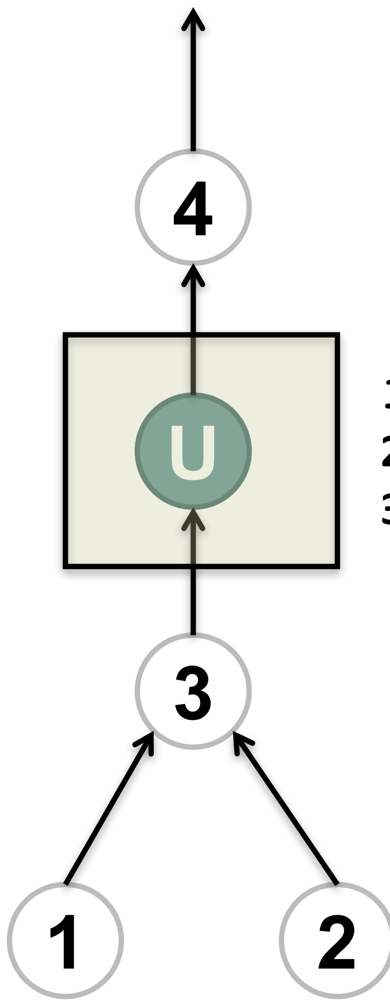
Uniform Samples

ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10



Uniform Samples

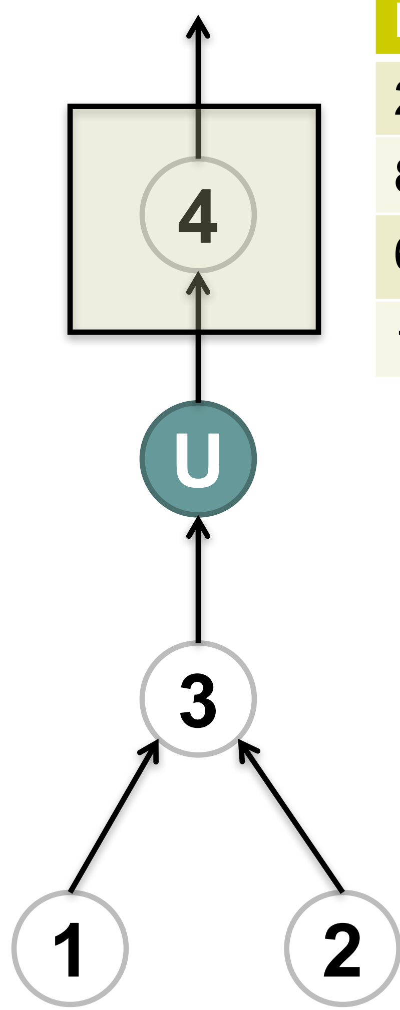
ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10



1. `FILTER rand() < 1/3`
2. Adds per-row Weights
3. (Optional) `ORDER BY rand()`

Uniform Samples

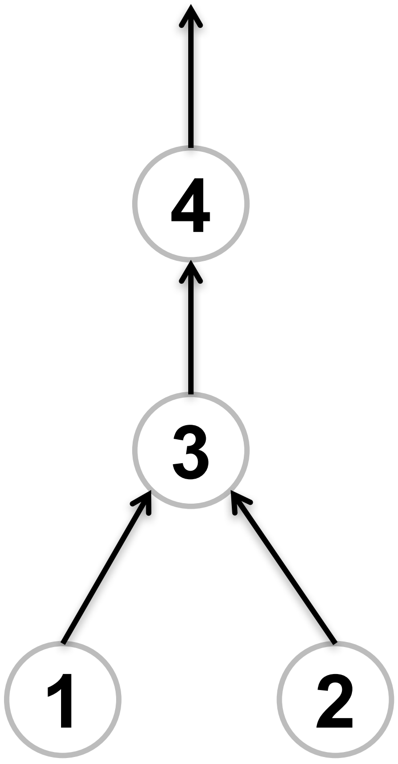
ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10



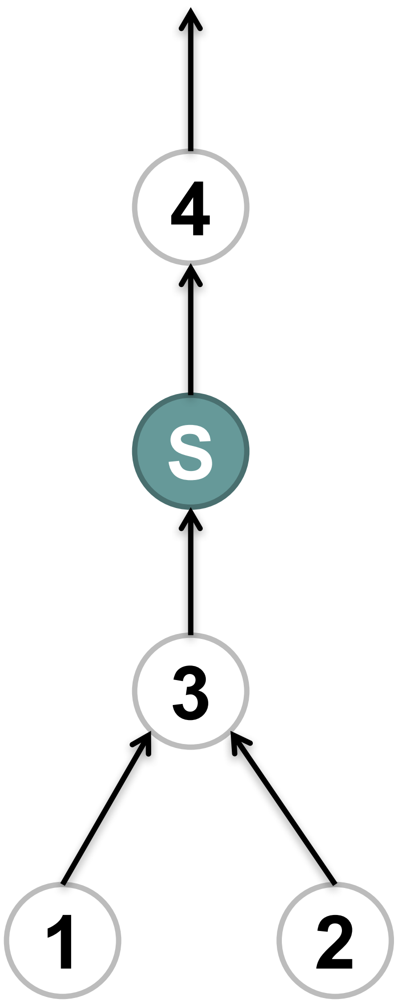
ID	City	Data	Weight
2	NYC	0.13	1/3
8	NYC	0.25	1/3
6	Berkeley	0.09	1/3
11	NYC	0.19	1/3

Doesn't change Shark RDD Semantics

Stratified Samples

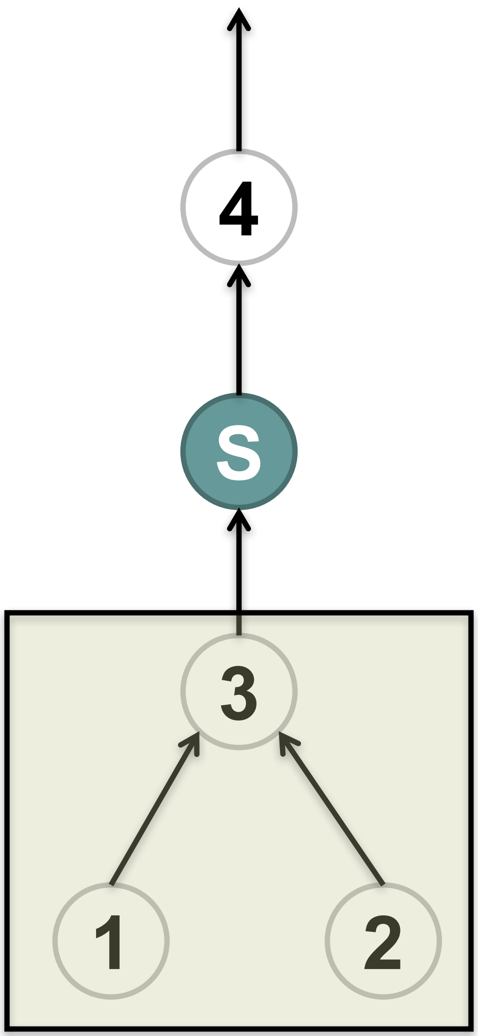


Stratified Samples



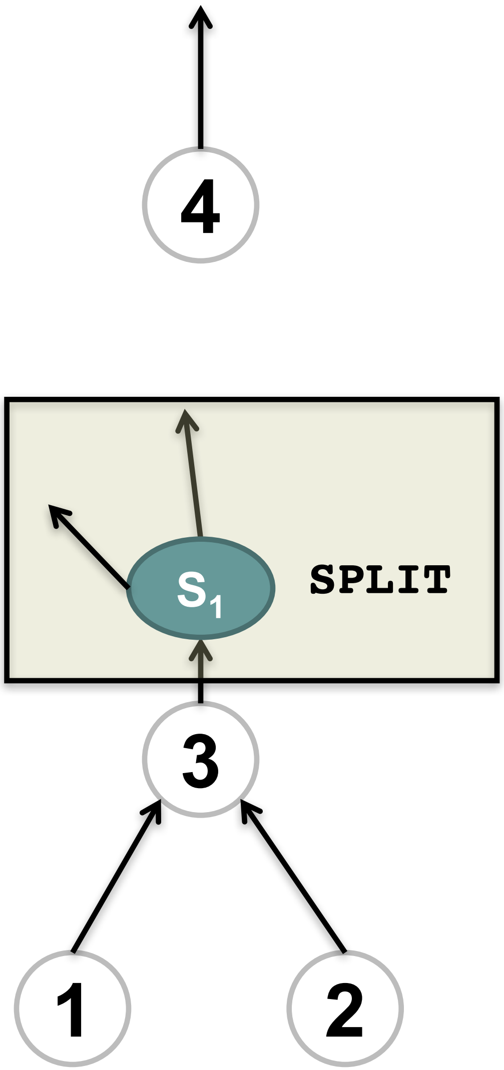
Stratified Samples

ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10



Stratified Samples

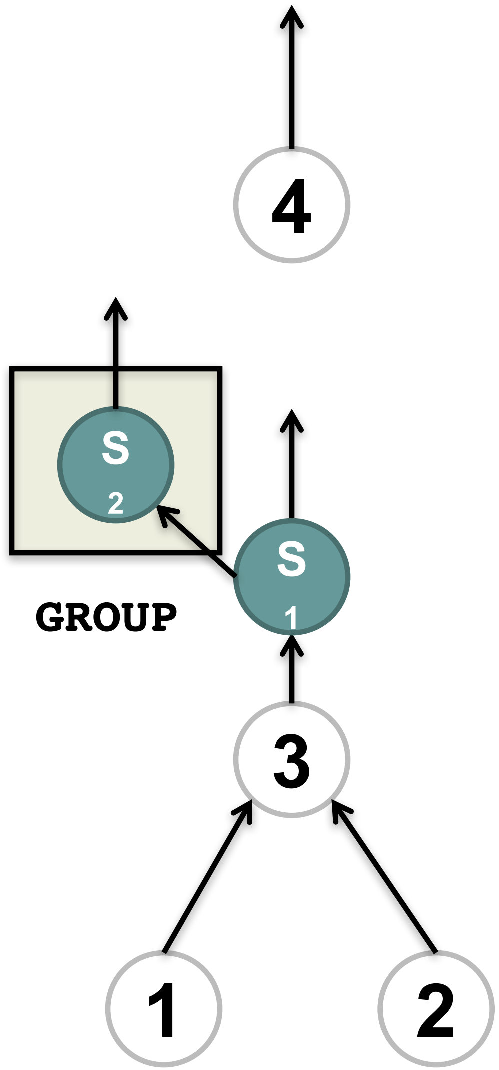
ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10



ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

Stratified Samples

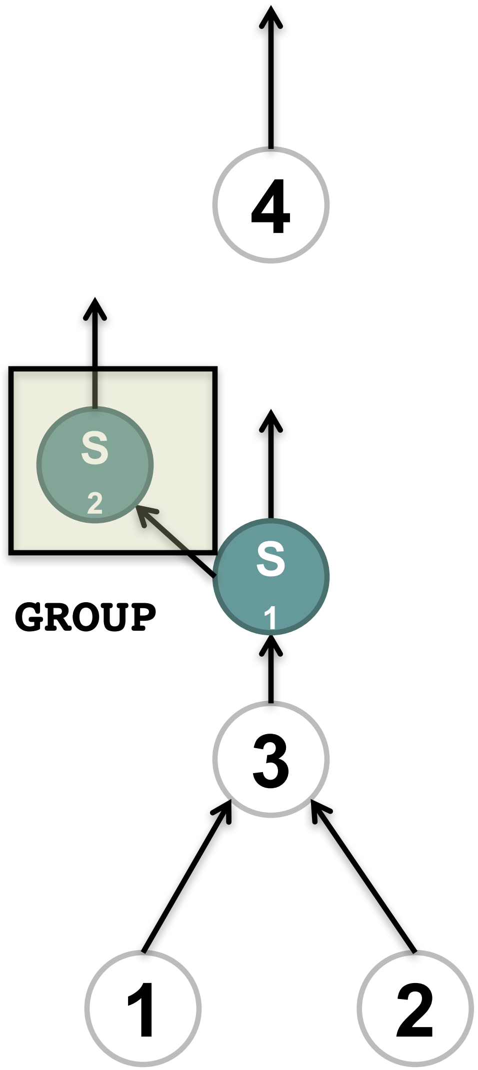
City	Count
NYC	7
Berkeley	5



ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

Stratified Samples

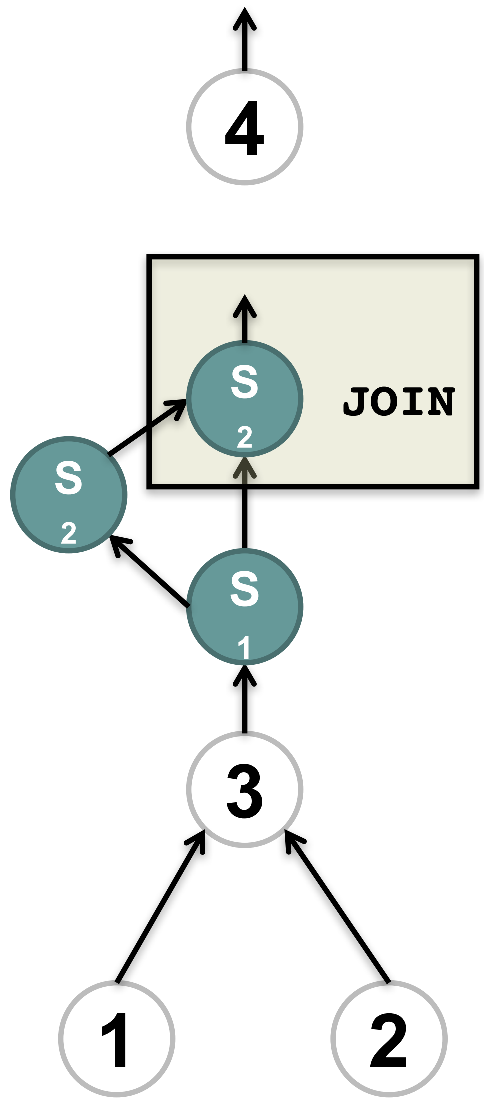
City	Count	Ratio
NYC	7	2/7
Berkeley	5	2/5



ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

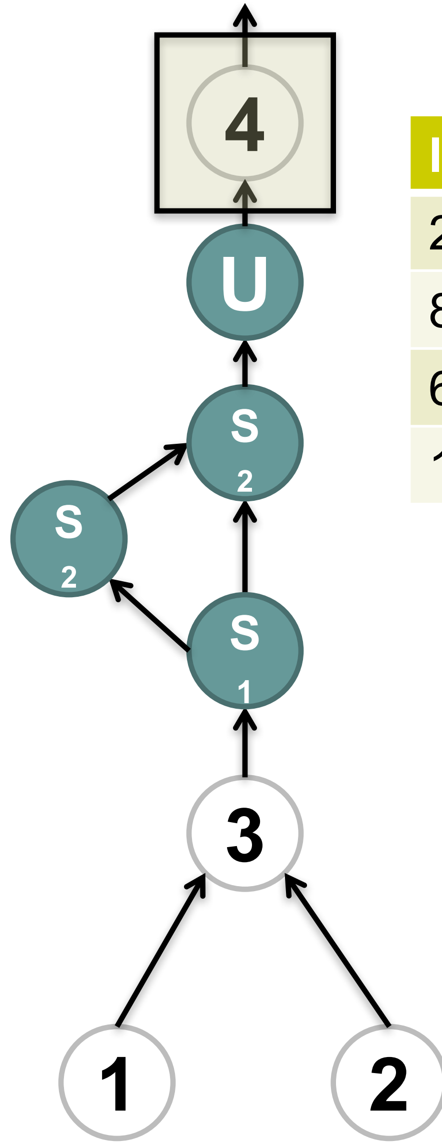
Stratified Samples

City	Count	Ratio
NYC	7	2/7
Berkeley	5	2/5



ID	City	Data
1	NYC	0.78
2	NYC	0.13
3	Berkeley	0.25
4	NYC	0.19
5	NYC	0.11
6	Berkeley	0.09
7	NYC	0.18
8	NYC	0.15
9	Berkeley	0.13
10	Berkeley	0.49
11	NYC	0.19
12	Berkeley	0.10

Stratified Samples



ID	City	Data	Weight
2	NYC	0.13	2/7
8	NYC	0.25	2/7
6	Berkeley	0.09	2/5
12	Berkeley	0.49	2/5

Doesn't change Shark RDD Semantics

What is BlinkDB?

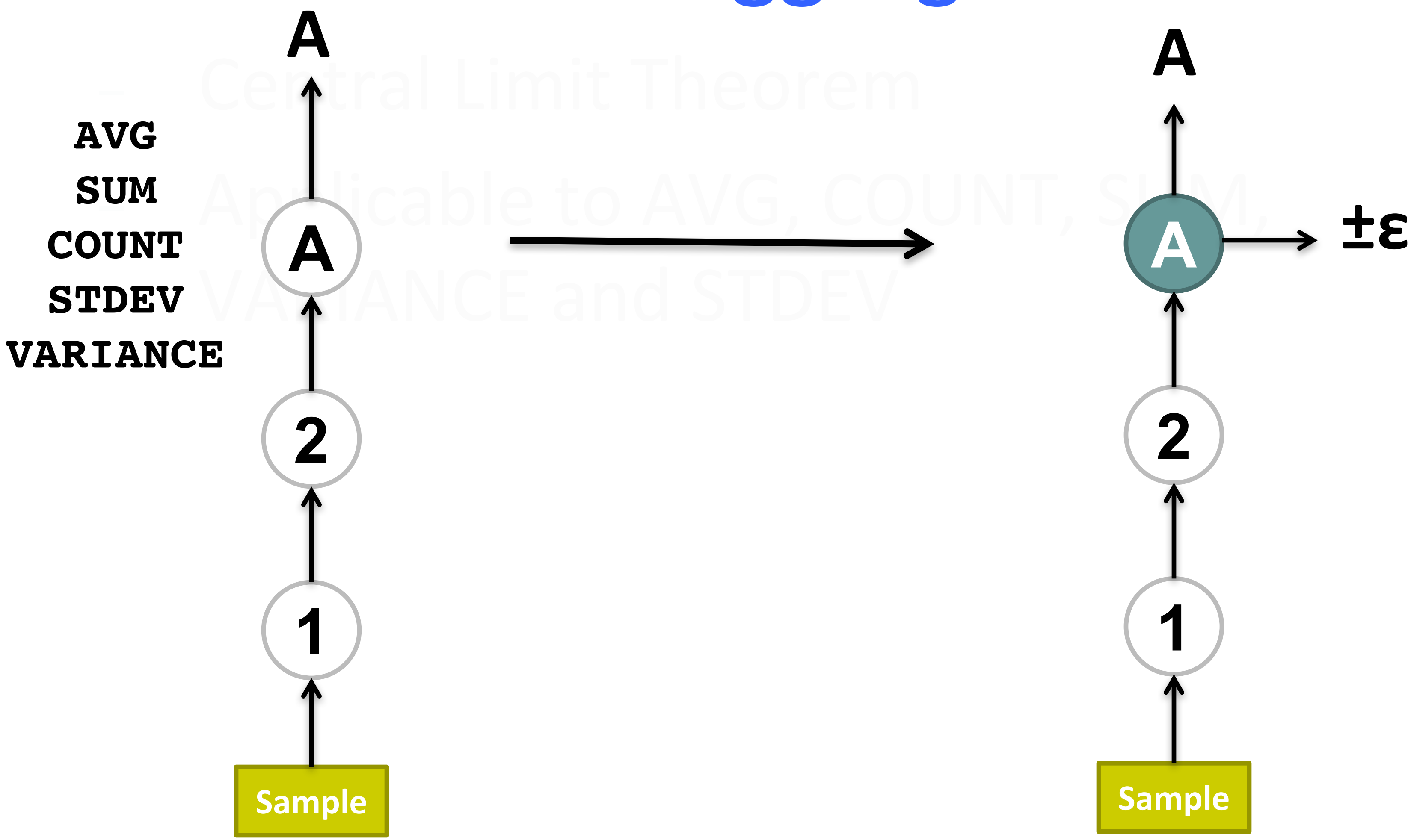
- A framework built on Shark and Spark that …
  - creates and maintains a variety of uniform and stratified samples from underlying data
  - returns fast, approximate answers with error bars by executing queries on samples of data
  - verifies the correctness of the error bars that it returns at runtime

Error Estimation

- Closed Form Aggregate Functions
  - Central Limit Theorem
  - Applicable to `AVG, COUNT, SUM, VARIANCE` and `STDEV`

Error Estimation

- Closed Form Aggregate Functions
  - Central Limit Theorem
  - Applicable to AVG, COUNT, SUM, VARIANCE and STDEV

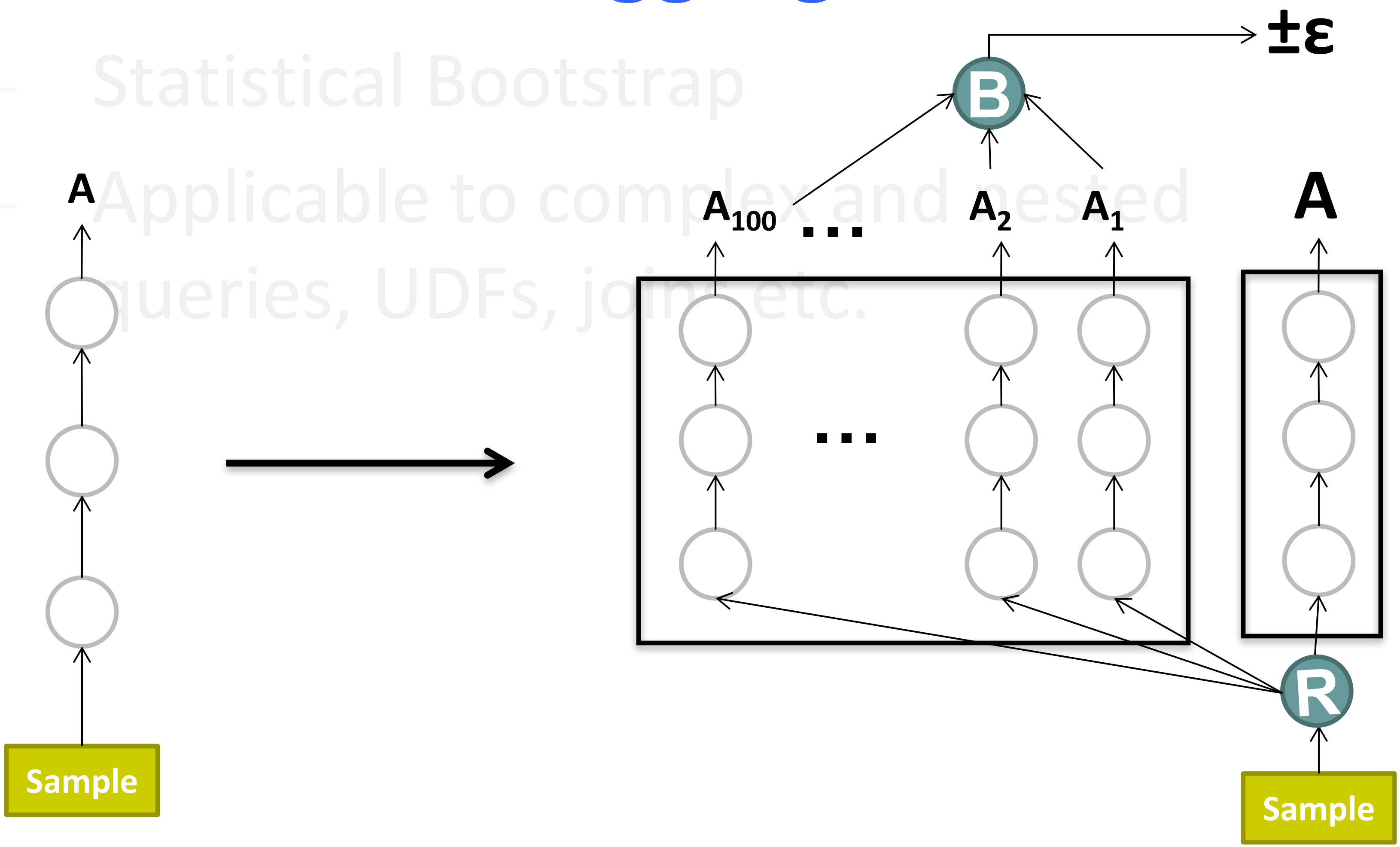


Error Estimation

- Generalized Aggregate Functions
  - Statistical Bootstrap
  - Applicable to complex and nested queries, UDFs, joins etc.

Error Estimation

- Generalized Aggregate Functions
  - Statistical Bootstrap
  - Applicable to complex and nested queries, UDFs, joins etc.



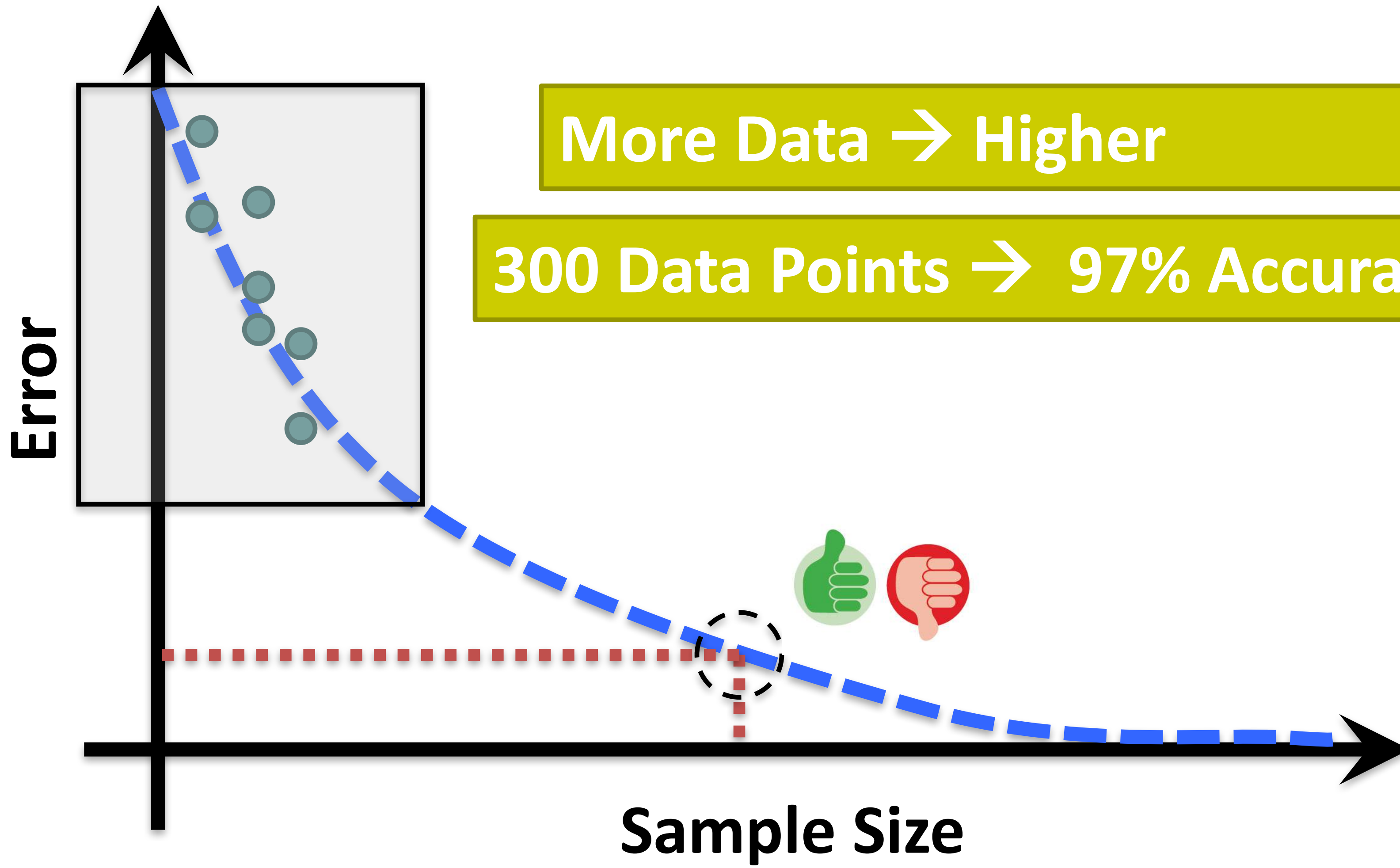
What is BlinkDB?

- A framework built on Shark and Spark that …
  - creates and maintains a variety of random and stratified samples from underlying data
  - returns fast, approximate answers with error bars by executing queries on samples of data
  - verifies the correctness of the error bars that it returns at runtime

Kleiner's Diagnostics

More Data → Higher

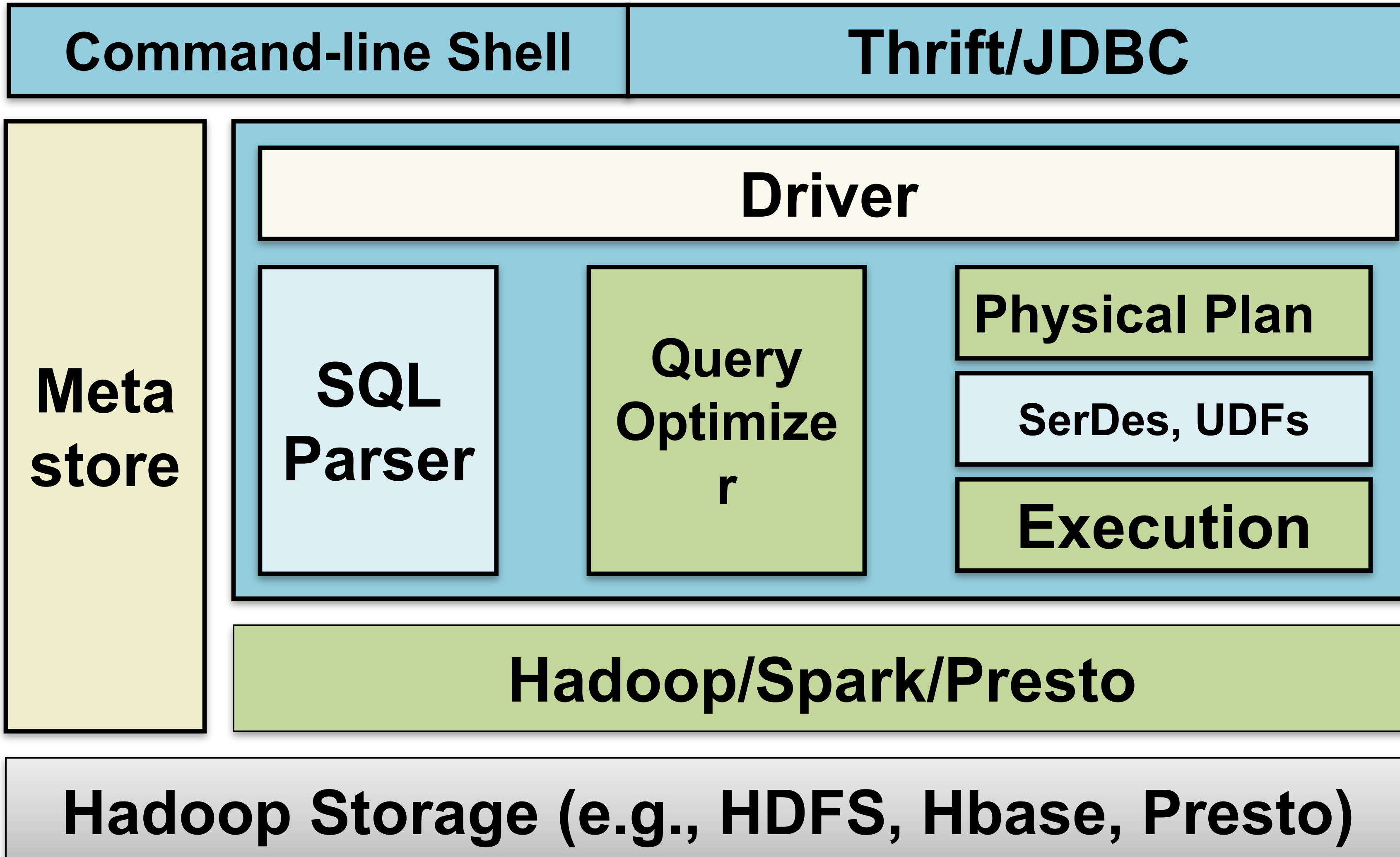
300 Data Points → 97% Accuracy



What is BlinkDB?

- A framework built on Shark and Spark that …
  - creates and maintains a variety of random and stratified samples from underlying data
  - returns fast, approximate answers with error bars by executing queries on samples of data
  - verifies the correctness of the error bars that it returns at runtime

BlinkDB Architecture



DAQ: A New Paradigm for Approximate Query Processing

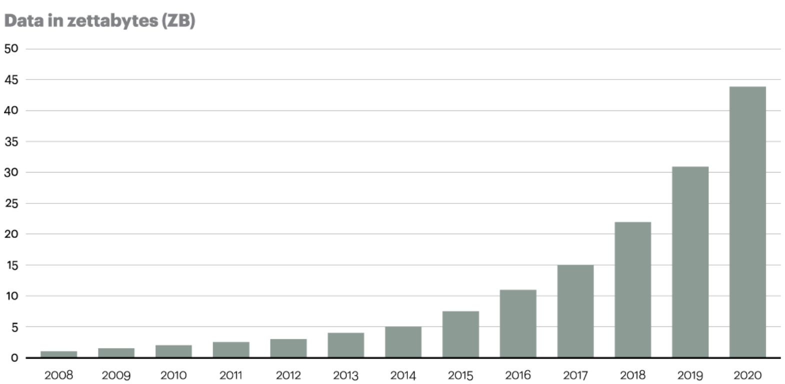
Approximate Query Processing

Data volume is growing exponentially

Queries are interactive to support real-time decisions

Decisions are resilient to small errors

Figure 1
Data is growing at a 40 percent compound annual rate, reaching nearly 45 ZB by 2020



Source: Oracle, 2012

Exploratory analysis demands responsiveness

Quick Approximate Answer
is better than
Slow Exact Answer

eg: *Average Revenue* estimate
$12M
is about as good as
$12,345,678

SAQ
Sampling-based Approximate Querying

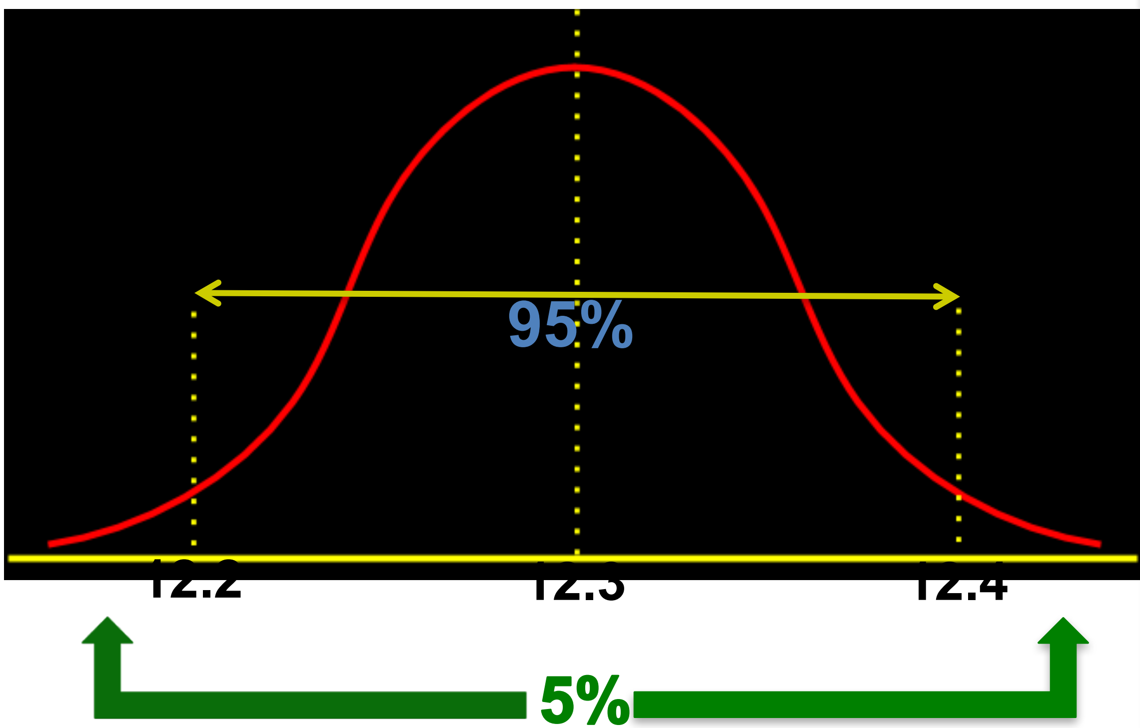
- Run query on a small random subset of data
- Error in estimate presented as *confidence interval*
  Avg revenue = \$12.3 $\pm$ 0.1 million with *95% confidence*
- Can be **“online”**
  - error eventually shrinks to zero => exact estimate

Confidence Intervals

- Avg revenue = \$12.3 $\pm$ 0.1 million with *95% confidence*

What does this mean?

Probability Distribution of Average Revenue



With 95% probability, true average revenue lies in 12.3 $\pm$ 0.1 million

With 5% probability, true average revenue lies outside 12.3 $\pm$ 0.1 million

Confidence Intervals

- eg: Avg revenue = \$12.3 $\pm$ 0.1 million with *95% confidence*

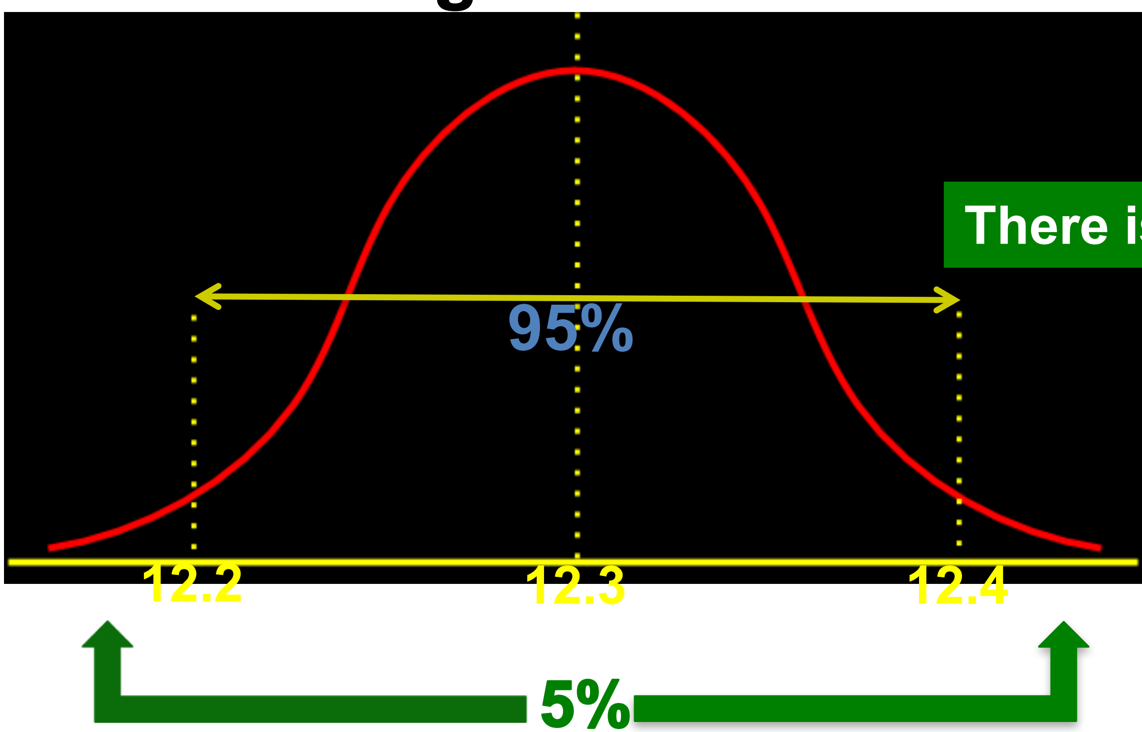
  How should we interpret the tails?

Probability Distribution of Average Revenue

Tails occur 5% of the time.

In Avg Regional Revenue for 100 states, 5 estimates are in the tails

There is no bound on error in the tail.



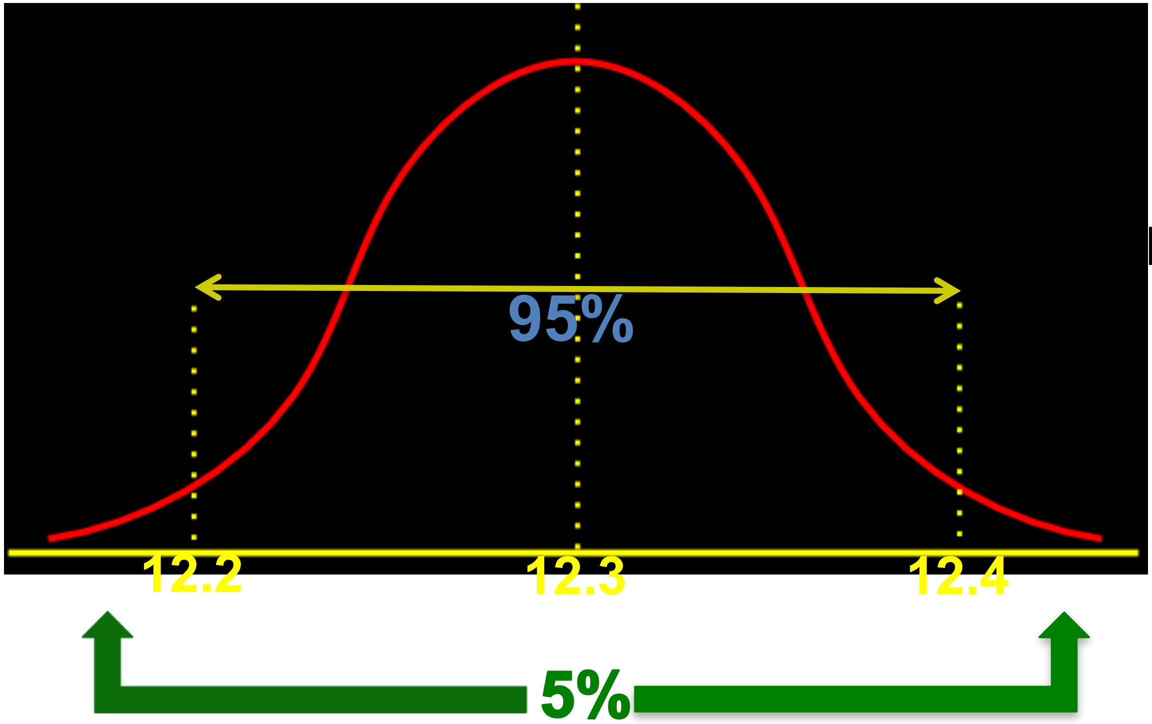
The Avg Revenue *could* be as small as \$1M or as large as \$100M

SAQ: Shortcomings

Semantics of the tails of confidence intervals are hard to interpret.

Intervals are very broad for outlier aggregates like MAX or Top 100.

Intervals are hard to manipulate. No closed algebra.



Confidence interval bounds are unintuitive

Need to see more of the data to find outliers. Slow convergence.

Is 100 $\pm$ 10 "*greater than*" 90 $\pm$ 20?
How do we *add* these intervals?

DAQ
Deterministic Approximate Querying

Pop quiz!

Estimate the sum.

```
      5,321,656
+     3,151,709
+     1,362,296
---------------
=     9,83?,???
```

a. Approximately 2.4 million?

b. Approximately 9.7 million?

c. Approximately 13.8 million?

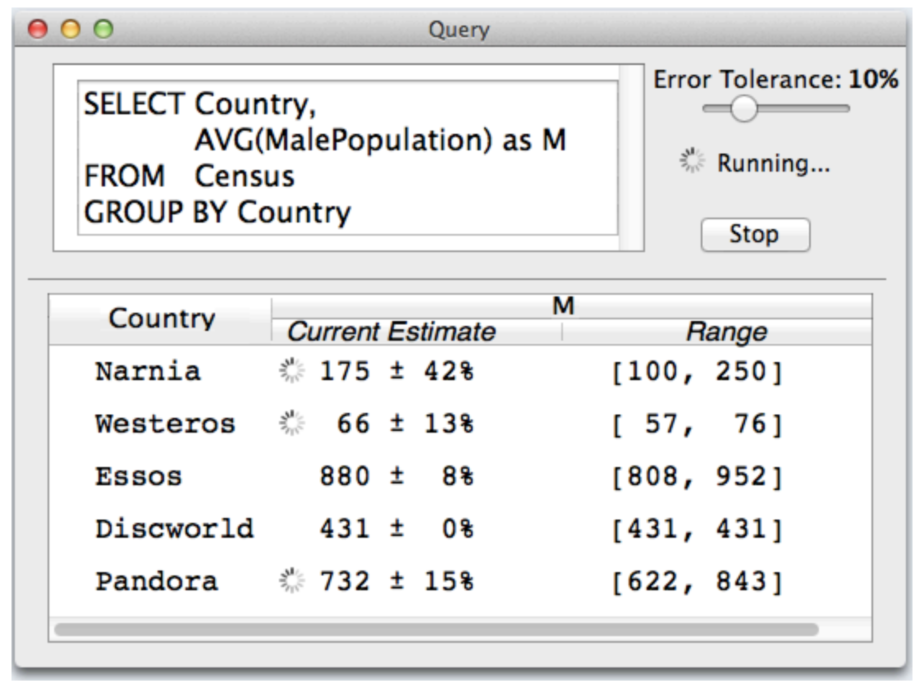
d. Approximately 17.0 million?

DAQ
Deterministic Approximate Querying

- Use *deterministic intervals* instead of probabilistic (confidence) intervals

- Guaranteed upper and lower bounds
  Avg revenue = $12.3 $\pm$ 0.2 million

- Can be "**online**"
  - Error interval eventually becomes degenerate => exact estimate



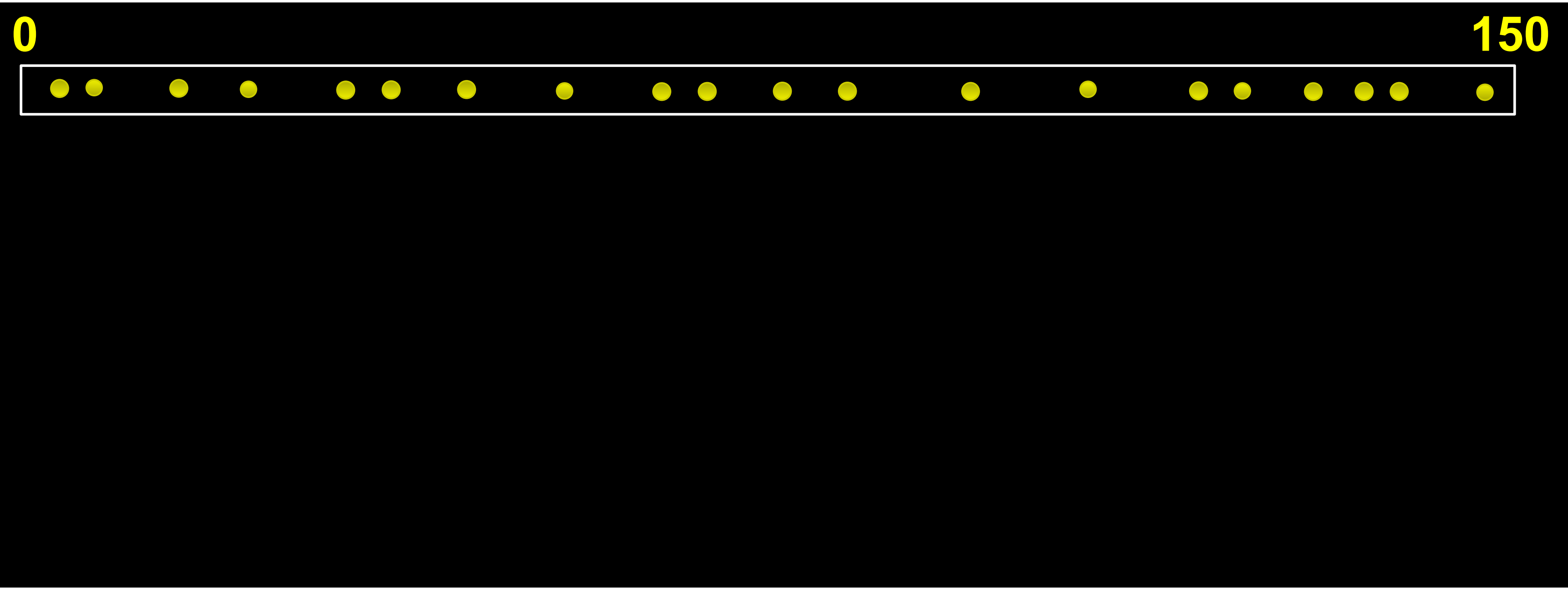
DAQ UI Mockup

SAQ vs DAQ
(at a glance)

Complex semantics using *confidence intervals* due to the "tail".	**Simple** semantics using *deterministic intervals* as there is no "tail".
Slow for *outlier* aggregates like MAX or Top 100 and *heavy-tailed* data.	**Fast** for *outlier* aggregates like MAX or Top 100 and *heavy-tailed* data.
No closed algebra. No clear semantics for predicates and arithmetic operations on estimates.	**Closed relational algebra.** Clear semantics for predicates and arithmetic ops using *interval algebra*.

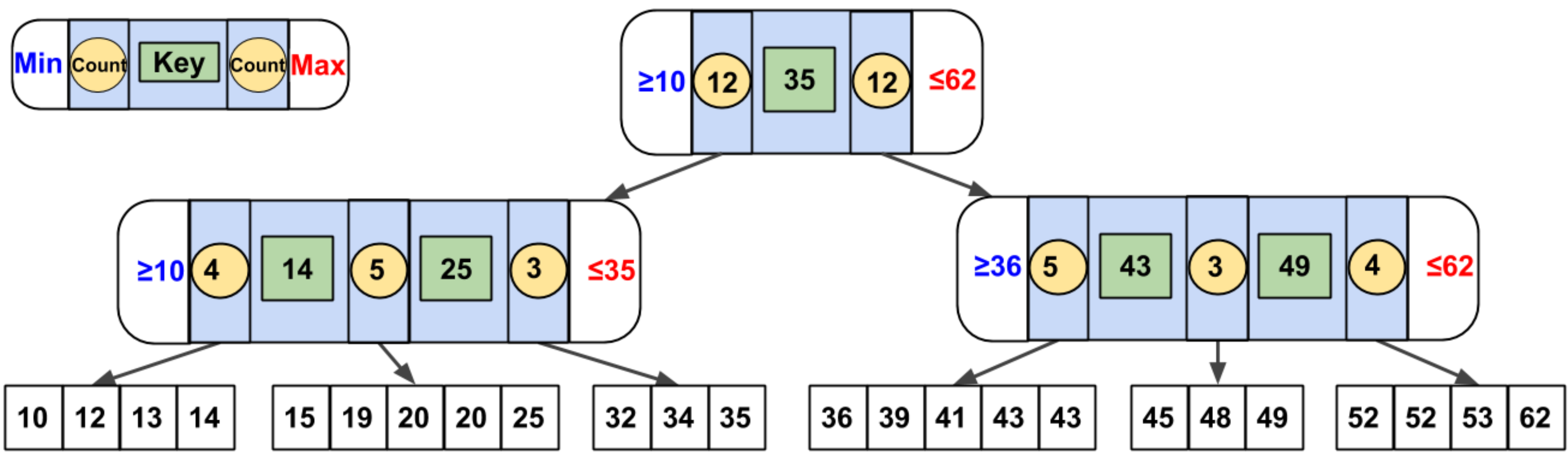
Conceptual DAQ Scheme

- Hierarchically partition the attribute's domain
- Estimates are represented as intervals *[a,b]*



Conceptual DAQ Scheme

- Hierarchically partition the attribute's domain
- Estimates are represented as intervals *[a,b]*
- *e.g., Count B-Tree*



Interval Algebra

- Predicate evaluation
- Interval representation for relations

City	Est. Population
Shire	[110,120]
Rivendell	[70, 90]
Gondor	[80,120]

Which cities have population > 100?

City	Est. Population
Shire	[110,120]
~~Rivendell~~	~~[70, 90]~~
~~Gondor~~	~~[80,120]~~

Certainly > 100

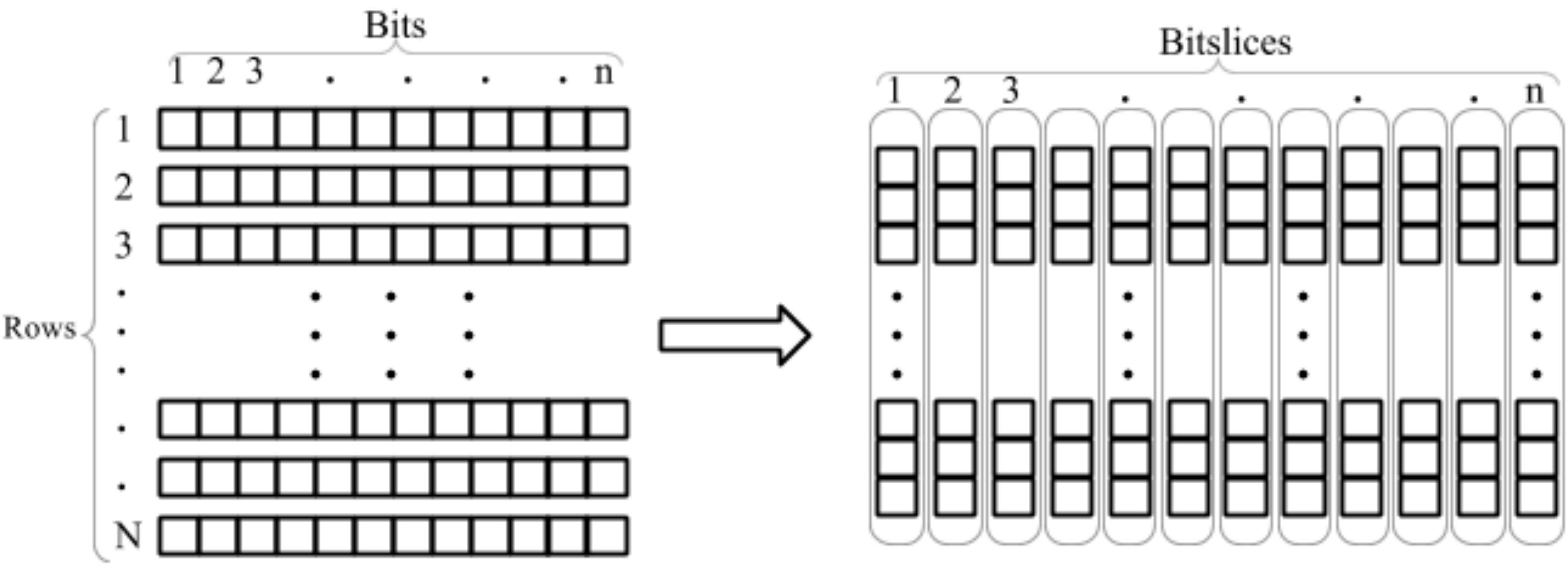
,

Shire	[110,120]
~~Rivendell~~	~~[70, 90]~~
Gondor	[80,120]

Potentially > 100

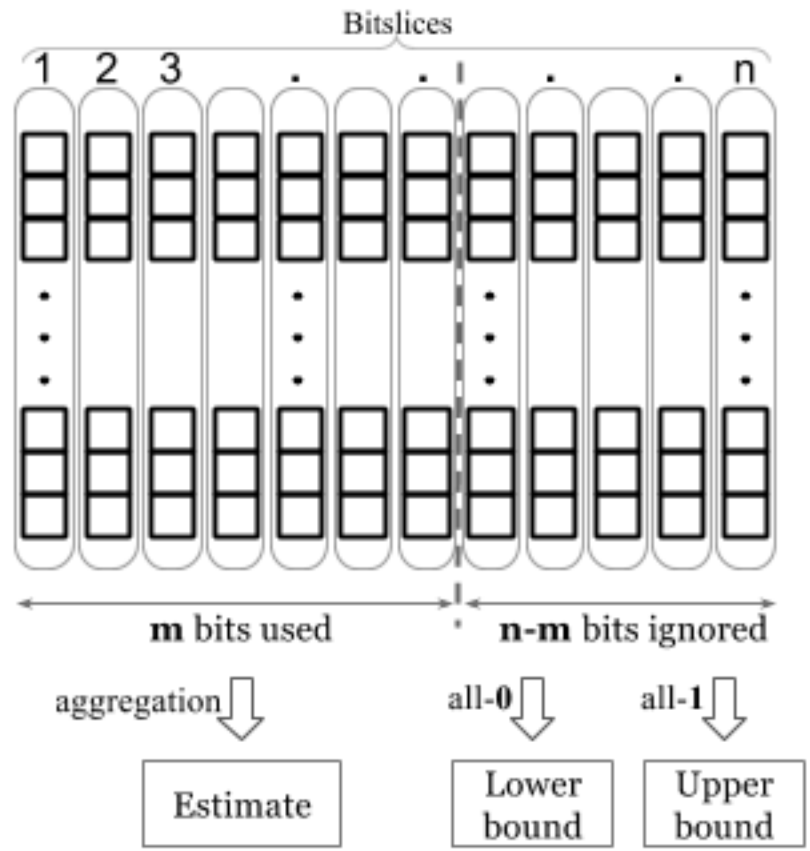
Bitwise DAQ Scheme

- Similar to the decimal digit-wise sum example
- Uses *Bitsliced Index* representation



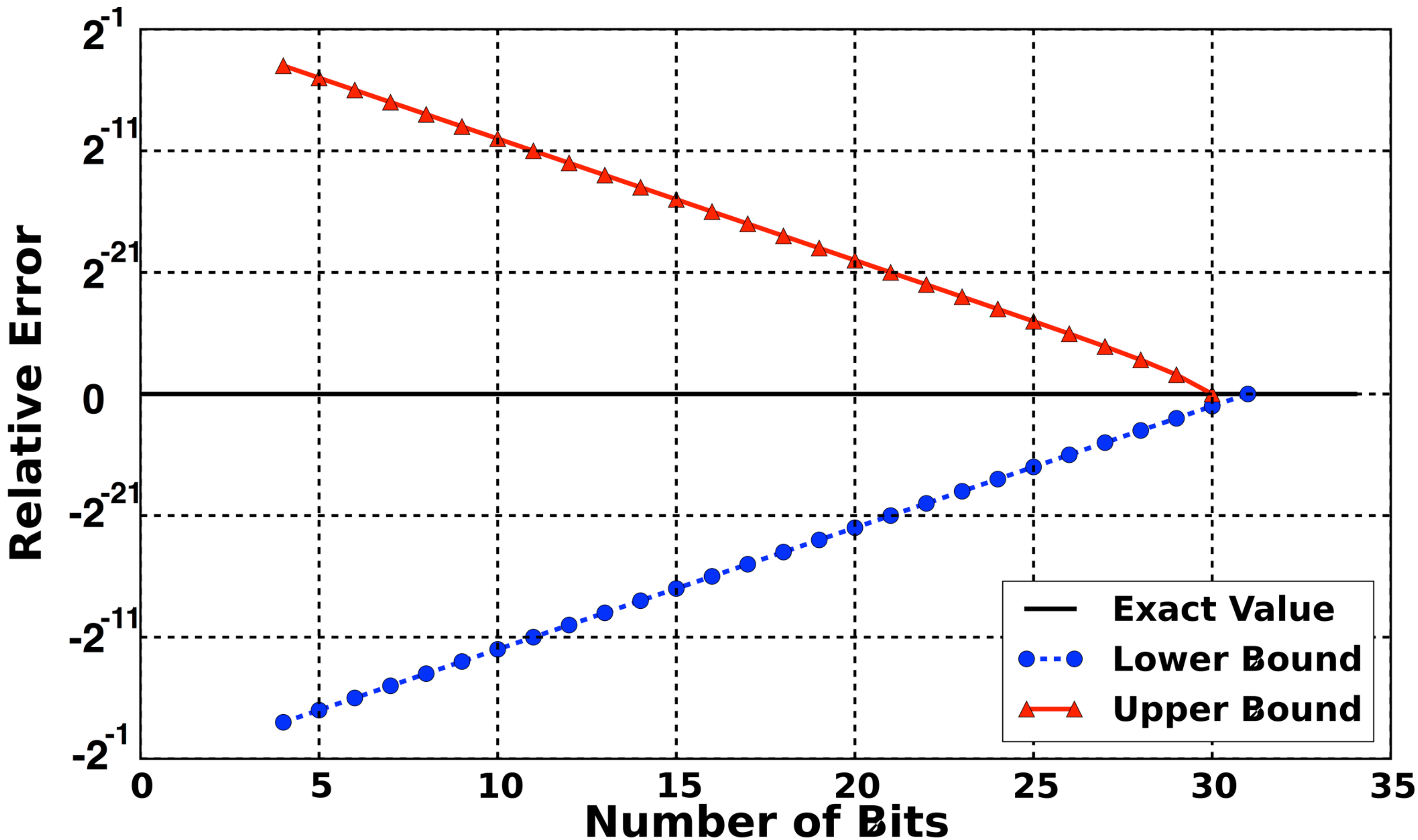
Bitwise DAQ Scheme

- Use most significant *m* bits for evaluation
- Remaining *n-m* bits set to all-0 and all-1 for bounds
- Error bound decreases exponentially: 2^{n-m}



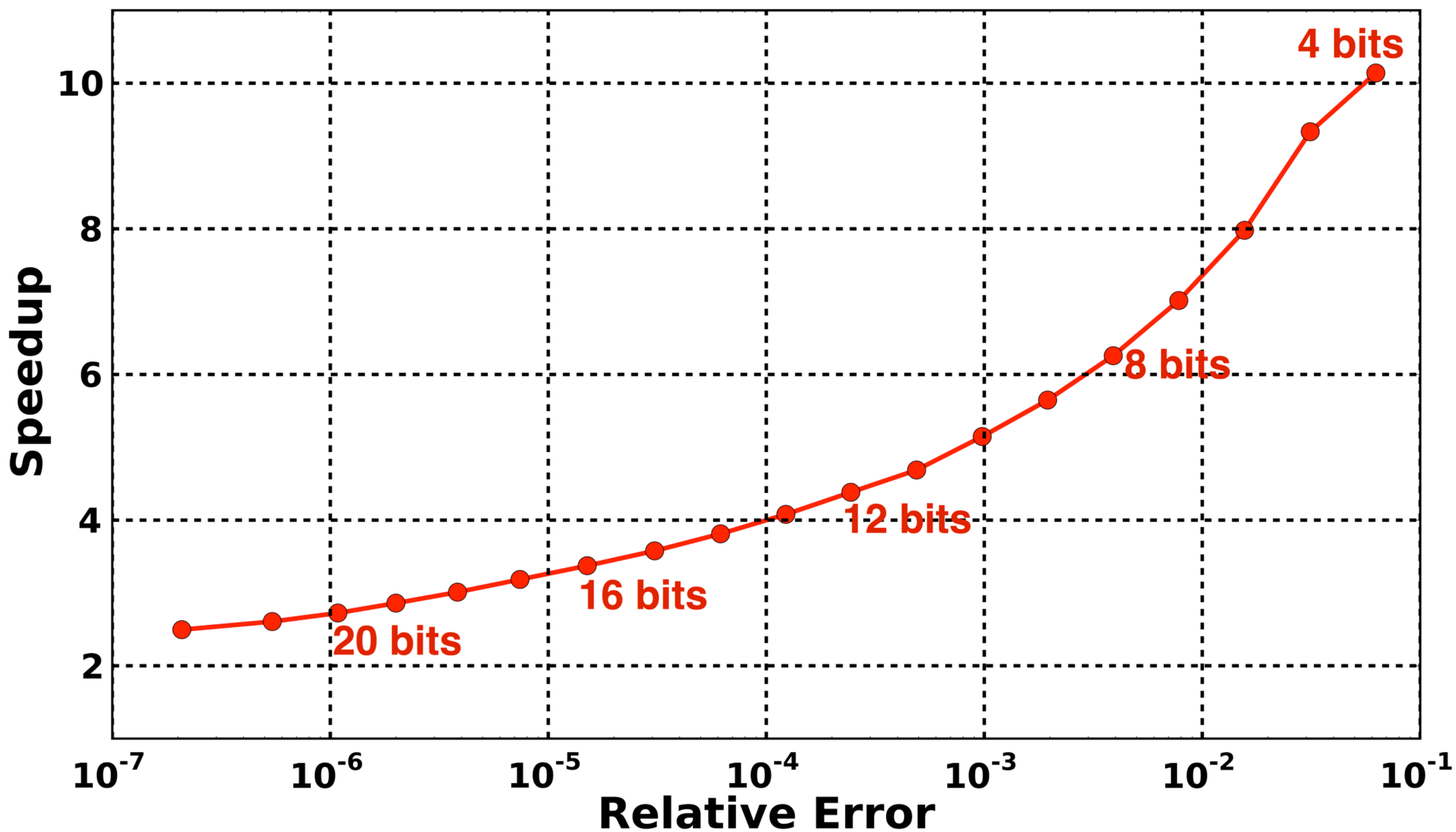
Bitwise DAQ vs. Baseline

Exponentially decreasing error bounds in estimating Avg



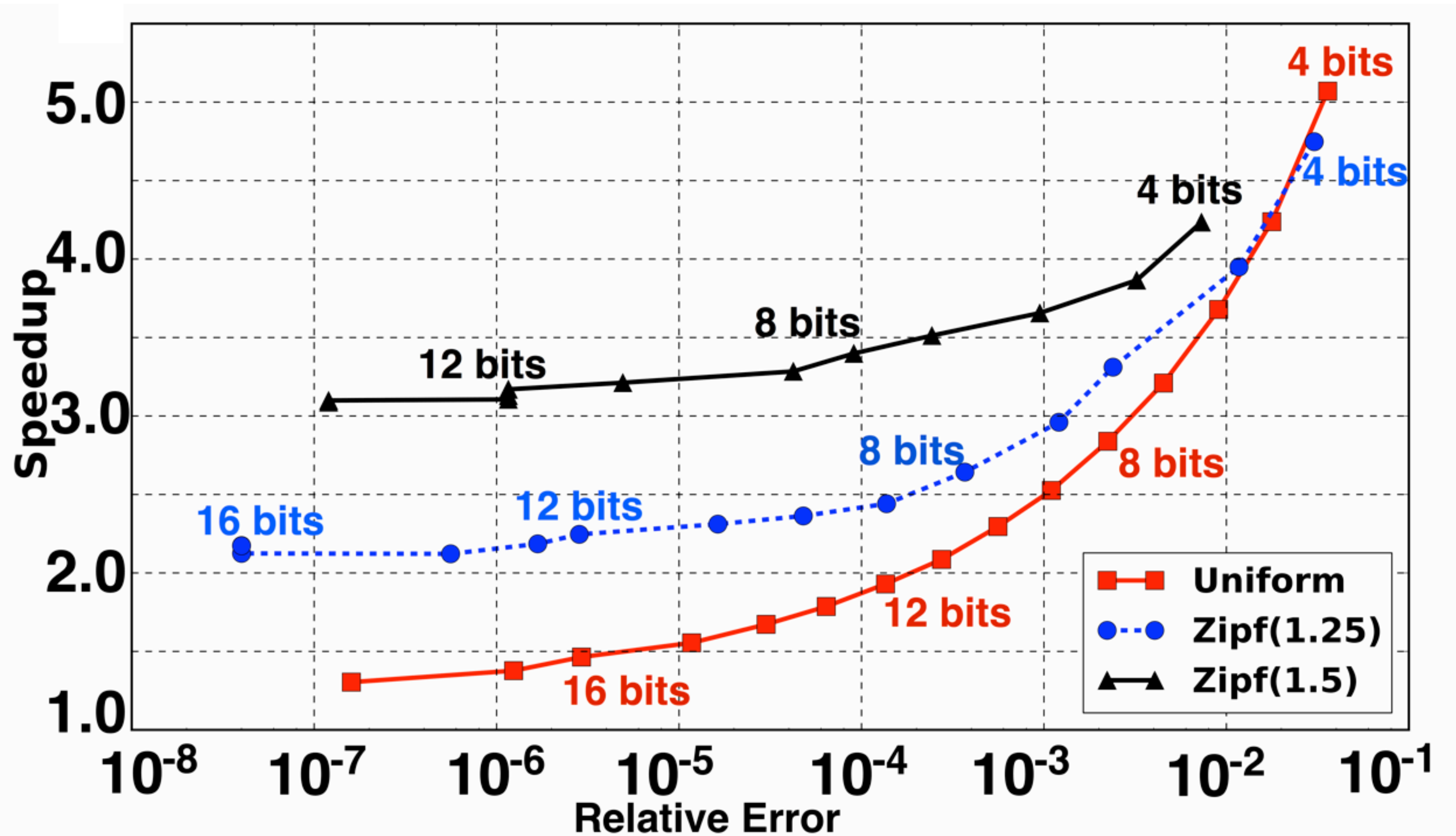
Bitwise DAQ vs. Baseline

Predicate evaluation: 6x speedup using 8 bits for < 1% error



Bitwise DAQ vs. Baseline

Top 100: 3.5x speedup for < 1% error on Uniform, Zipf data



Bitwise DAQ vs. SAQ

Top 100: DAQ performs better for heavy-tailed data