Computations with Bounded Errors and Response Times on Very Large Data

Ion Stoica
UC Berkeley

(joint work with: Sameer Agarwal, Ariel Kleiner, Henry Milner, Barzan Mozafari, Ameet Talwalkar, Purnamrita Sarkar, Michael Jordan, Sam Madden)

Paris, May 15, 2013

Problem

Support **interactive ad-hoc** exploration queries over **very large** datasets

Why is This a Problem?

100 TB on 1000 cores/disks

1-2 Hours

Hard Disks

10-15 Minutes

Memory

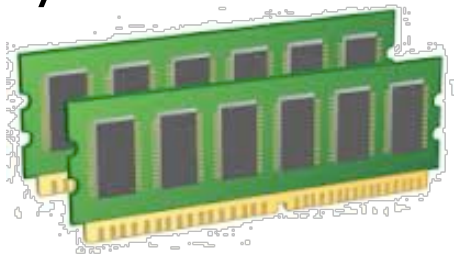
1 second

?

Why is This a Problem?

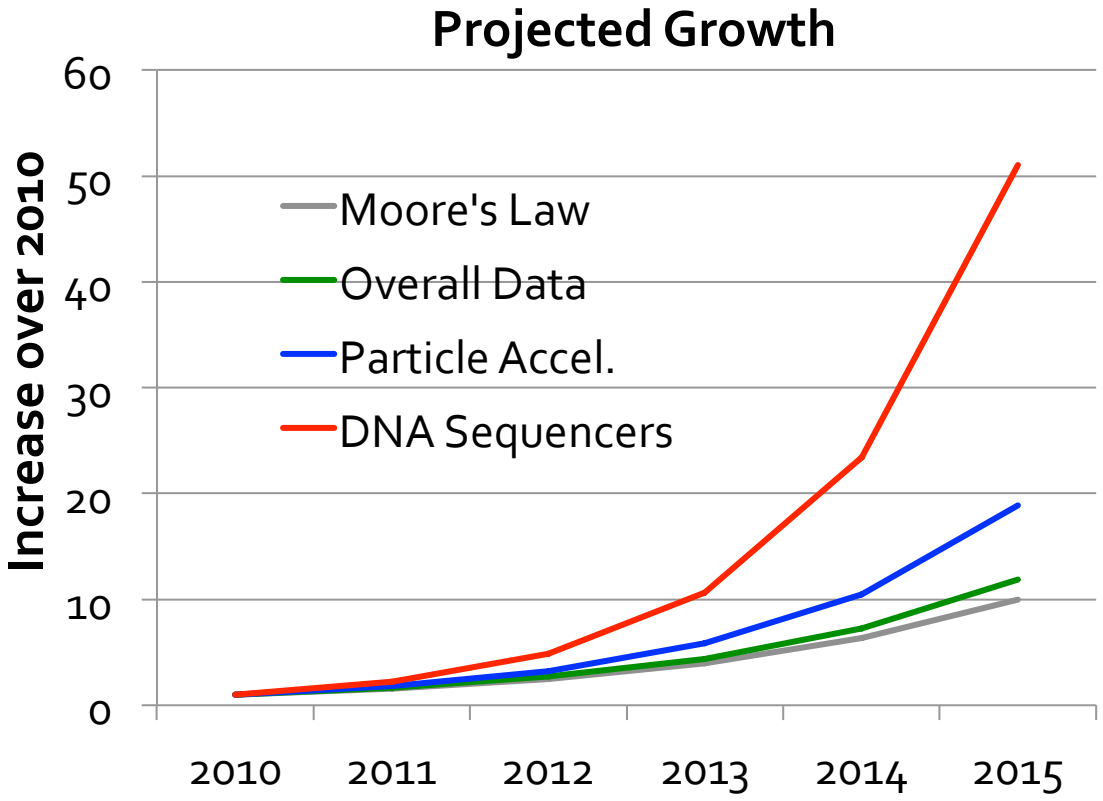
Even if no communication and all data in memory, query may take tens of sec

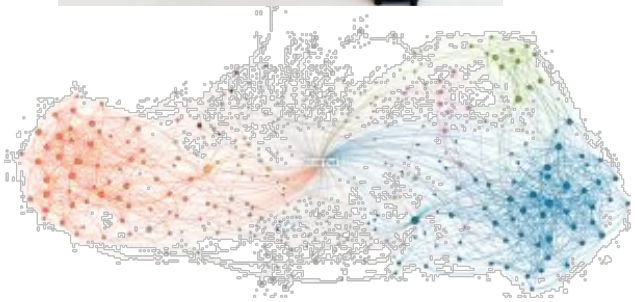
» Just scanning 200-300GB RAM may take 10 sec



Still slow for interactive queries

Why is This a Problem?





Data Grows faster than Moore's Law

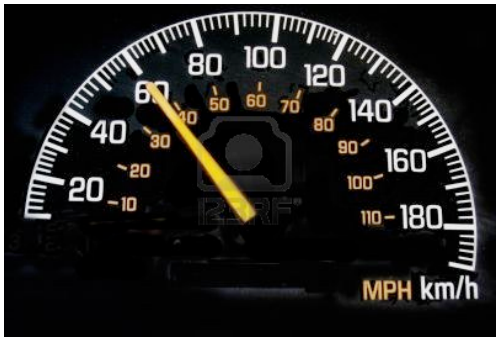
[IDC report, Kathy Yelick, LBNL]

Key Insight

Computations don't always need ***exact*** answers

- Input often ***noisy***: exact computations do ***not*** guarantee exact answers
- ***Error*** often acceptable if ***small*** and ***bounded***

Best scale
± 200g error



Speedometers
± 2.5 % error
(edmunds.com)

OmniPod Insulin Pump
± 0.96 % error
(www.ncbi.nlm.nih.gov/pubmed/22226273)

Approach: Sampling

Compute results on samples instead of full data

» Typically, error depends on sample size (n) ***not*** on original data size, i.e., error $\alpha \ 1/\sqrt{n}$

Can trade between answer's ***latency*** and ***accuracy***

Data rapid increase no longer a "problem":

» Error decreases with Moore's law: halves every **36** months

This Talk

BlinkDB: *approximate query engine for very large data sets using off-line sampling*

BlinkDB Interface

SELECT avg(sessionTime)

FROM Table

WHERE city='San Francisco' AND 'dt=2012-9-2'

WITHIN 1 SECONDS  234.23 ± 15.32

BlinkDB Interface

SELECT avg(sessionTime)

FROM Table

WHERE city='San Francisco' AND 'dt=2012-9-2'

WITHIN 2 SECONDS  ~~234.23 ± 15.32~~

239.46 ± 4.96

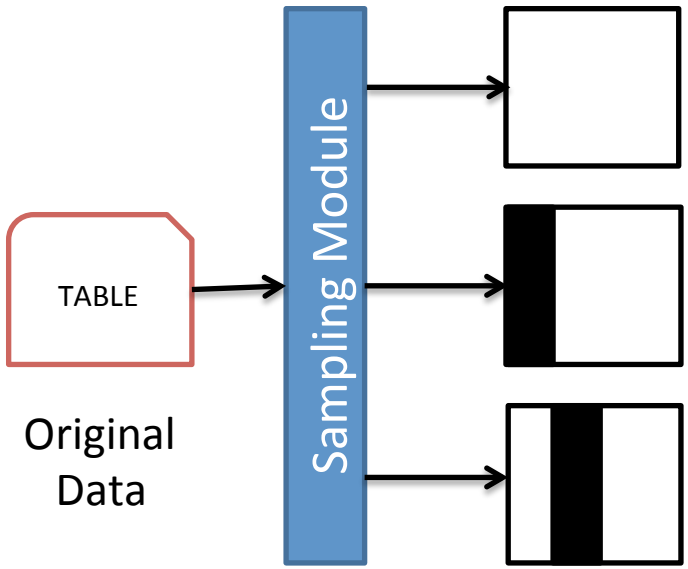
SELECT avg(sessionTime)

FROM Table

WHERE city='San Francisco' AND 'dt=2012-9-2'

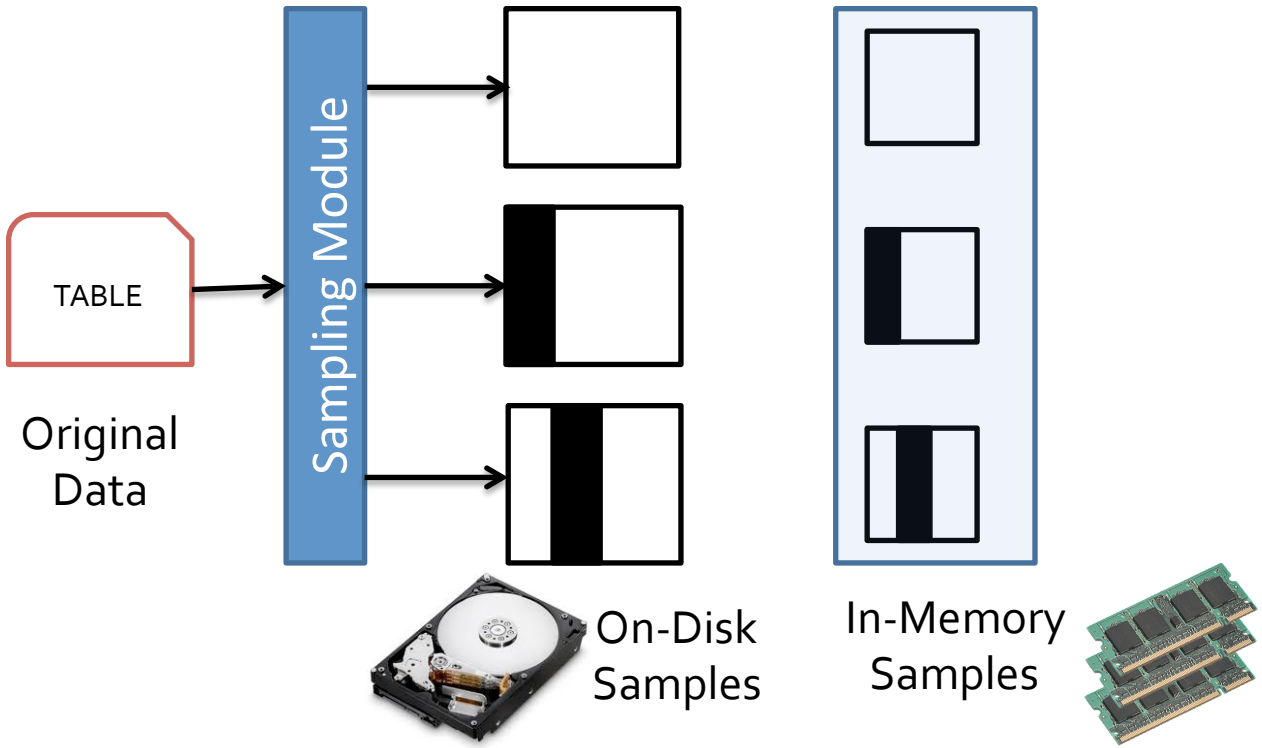
ERROR 0.1 **CONFIDENCE** 95.0%

BlinkDB Overview



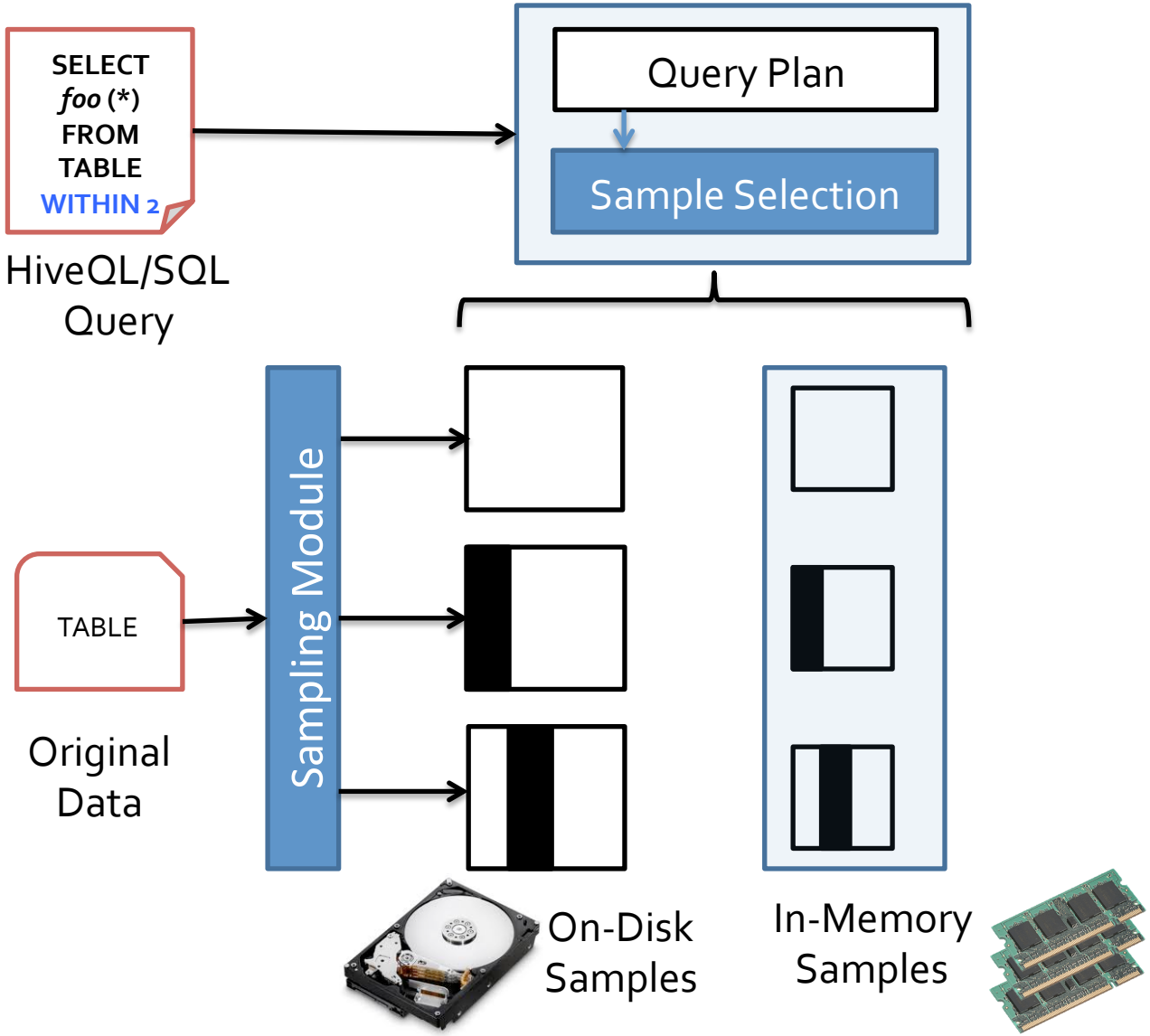
Offline-sampling: Optimal set of samples across different dimensions (*columns or sets of columns*) to support ad-hoc exploratory queries

BlinkDB Overview

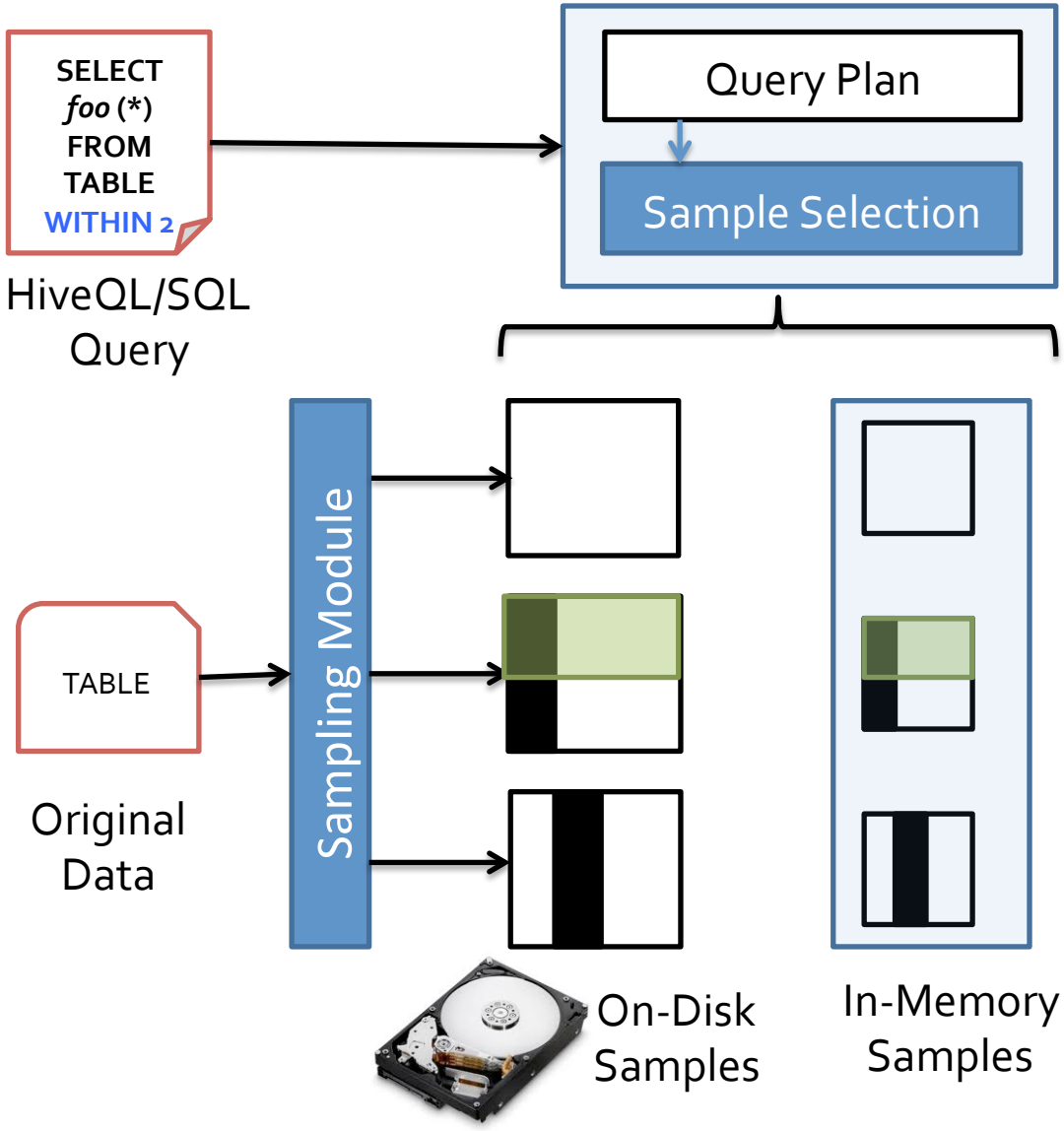


Sample Placement: Samples striped over 100s or 1,000s of machines both on disks and in-memory.

BlinkDB Overview

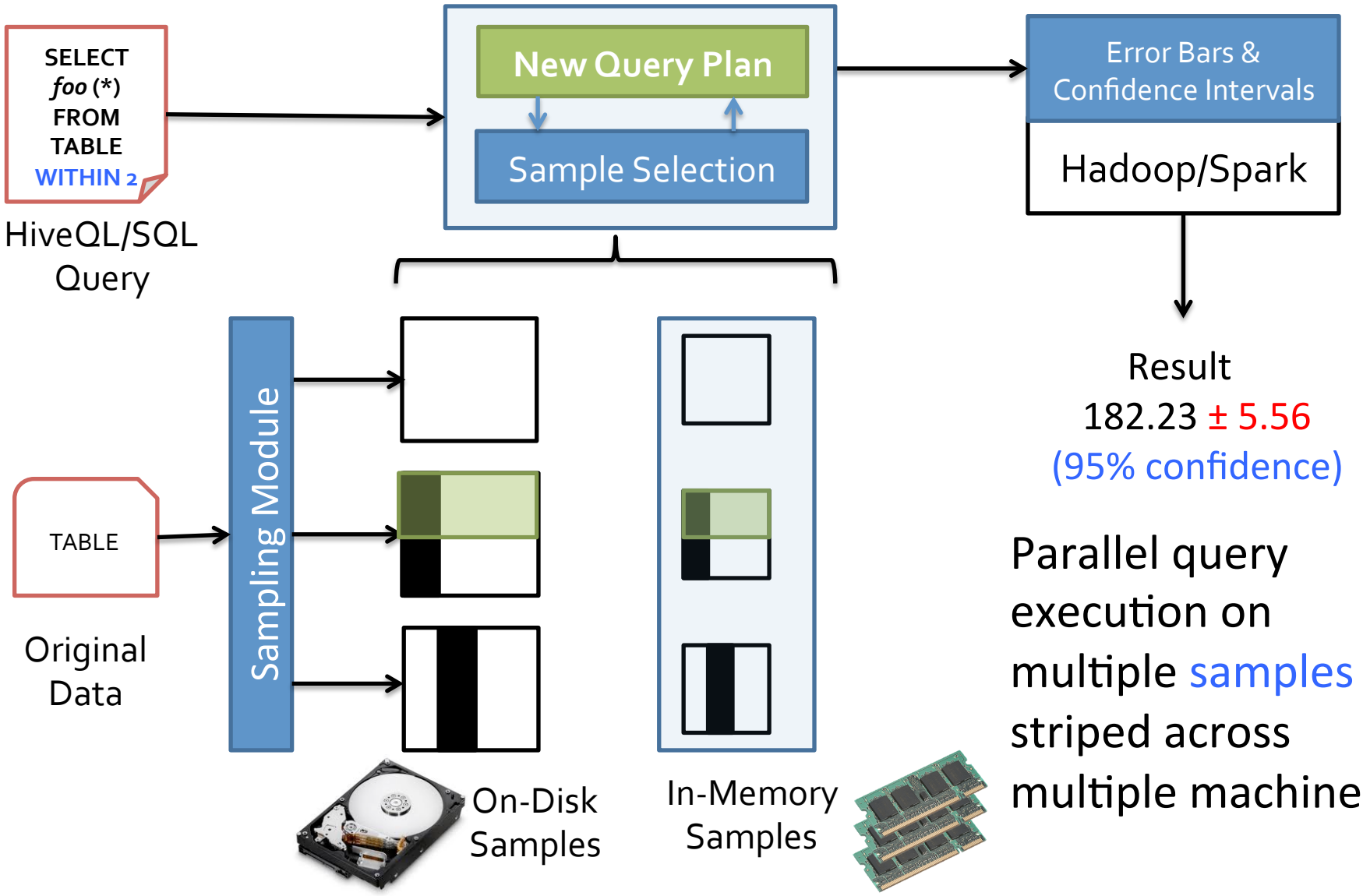


BlinkDB Overview



Online sample selection to pick best sample(s) based on query latency and accuracy requirements

BlinkDB Overview



Challenges

Which set of samples to build given a storage budget?

Which sample to run the query on?

How to accurately estimate the error?

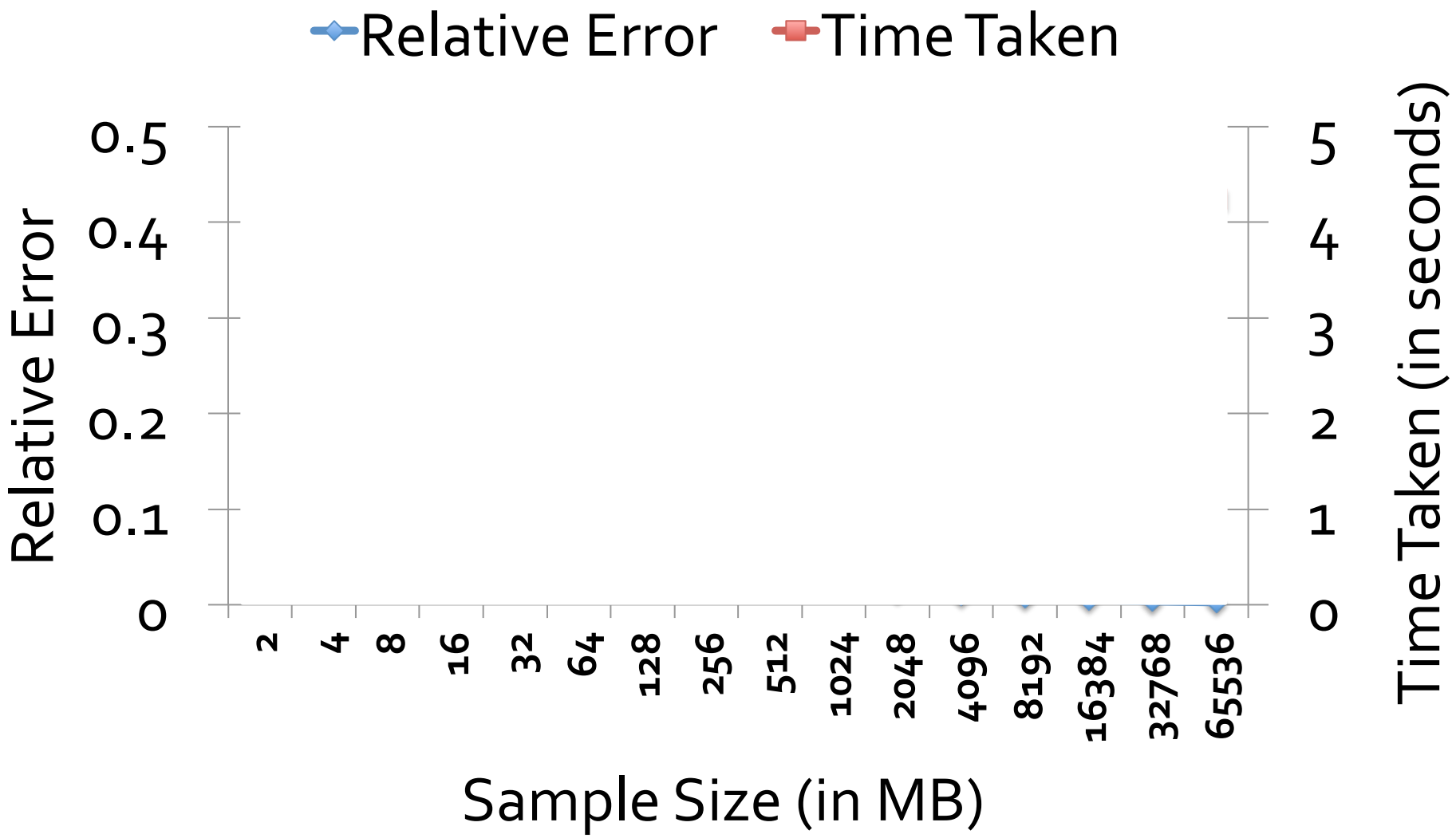
Challenges

Which set of samples to build given a storage budget?

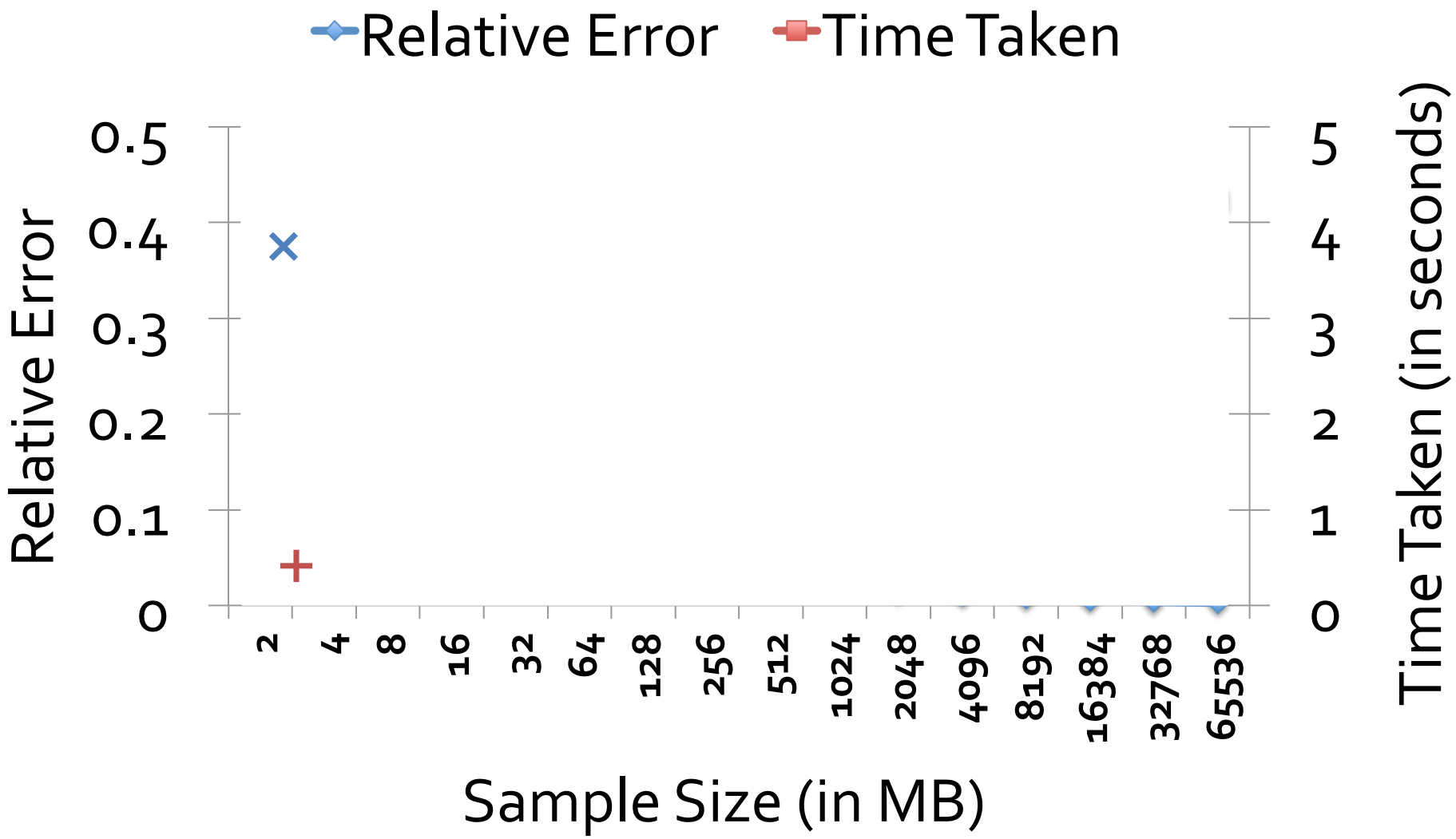
Which sample to run the query on?

How to accurately estimate the error?

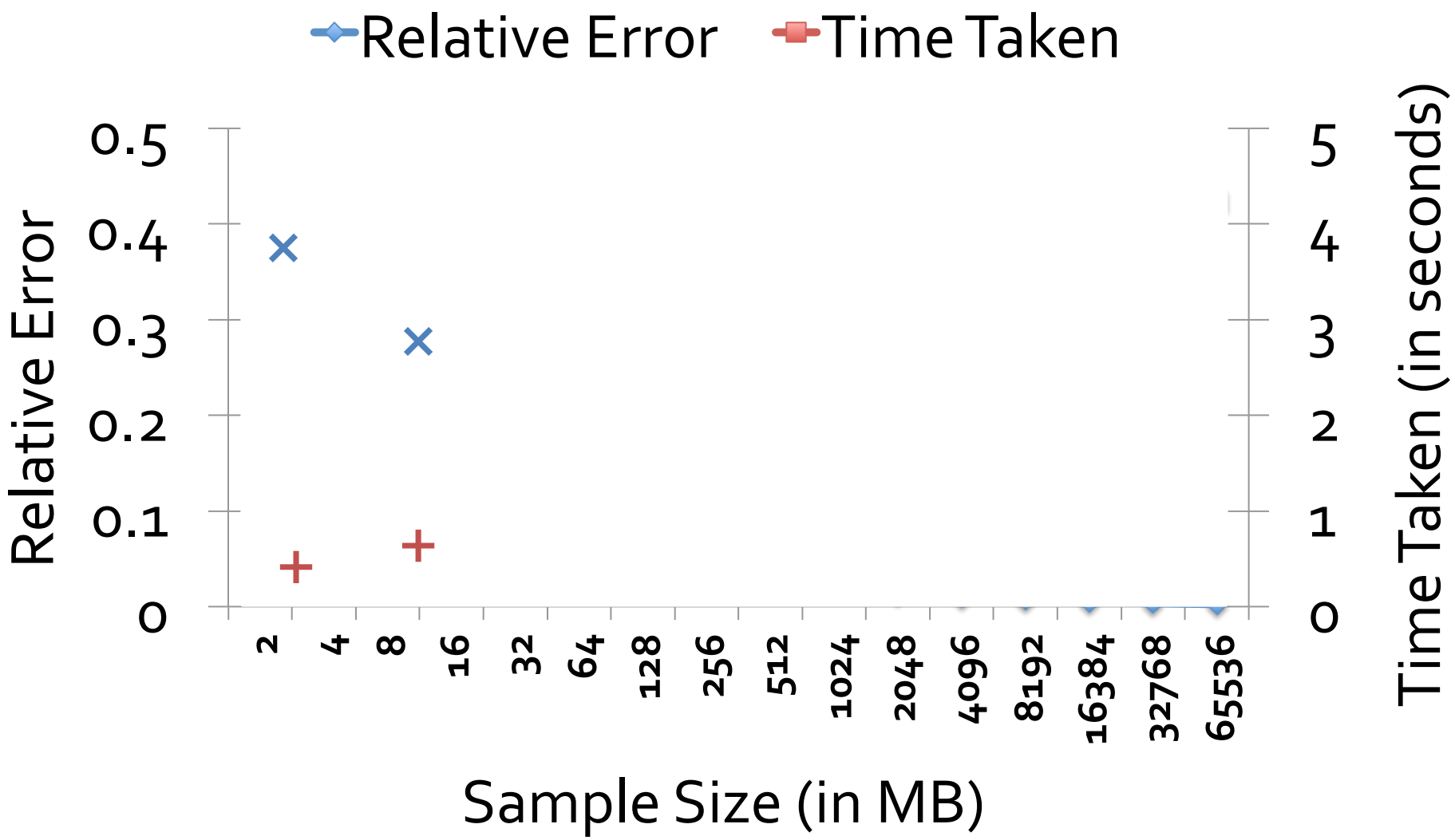
Error Latency Profile (ELP)



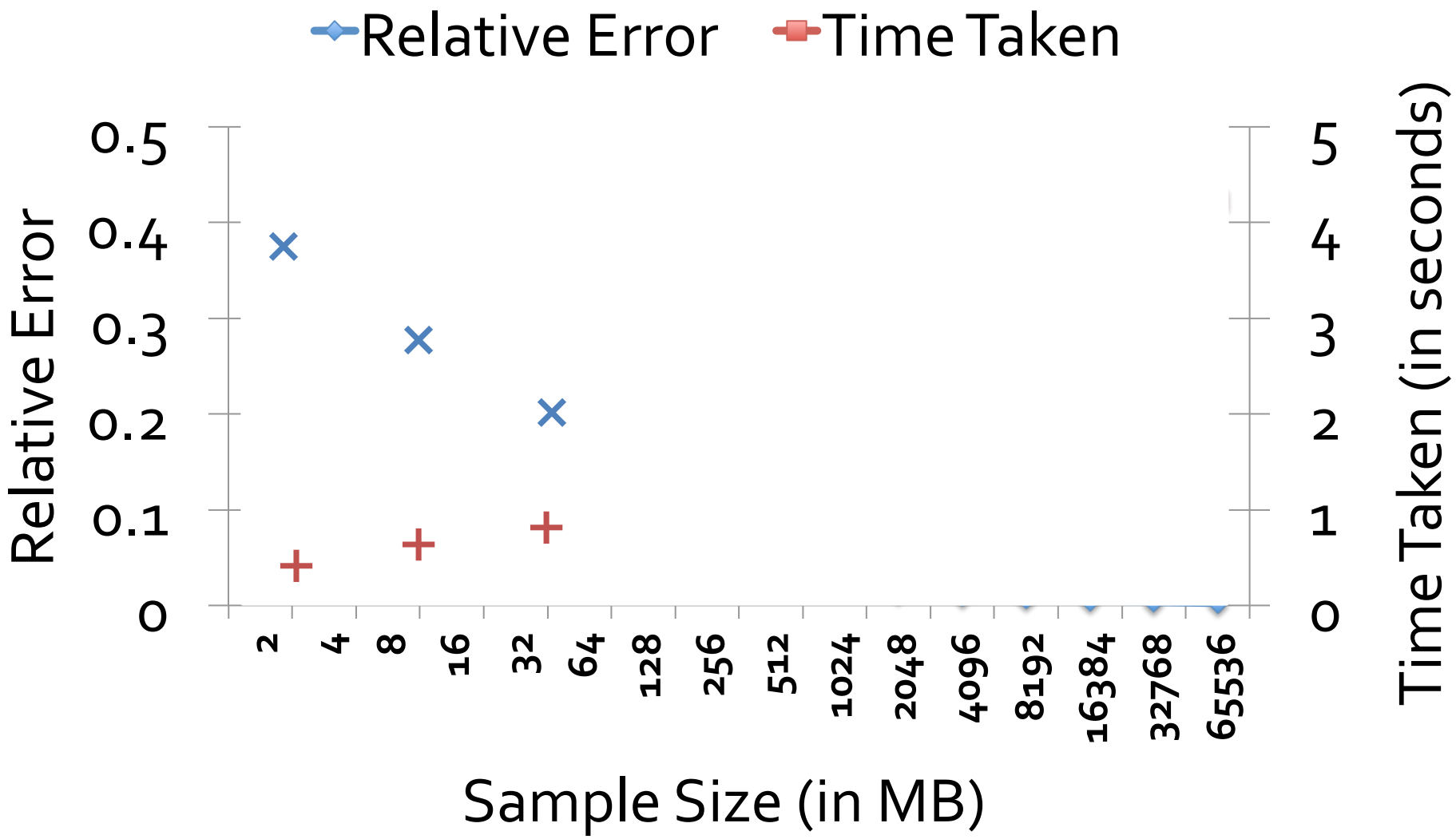
Error Latency Profile (ELP)



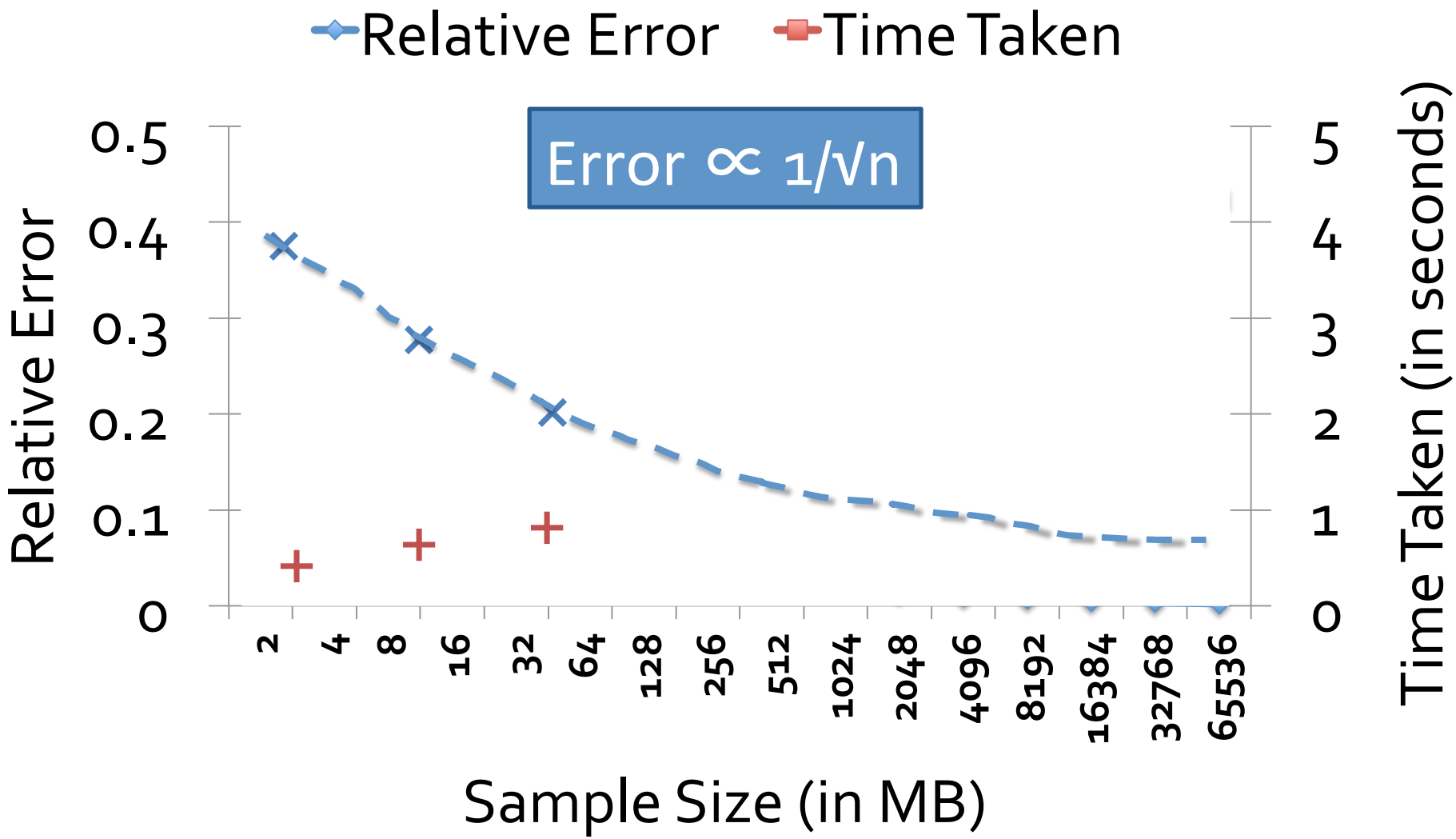
Error Latency Profile (ELP)



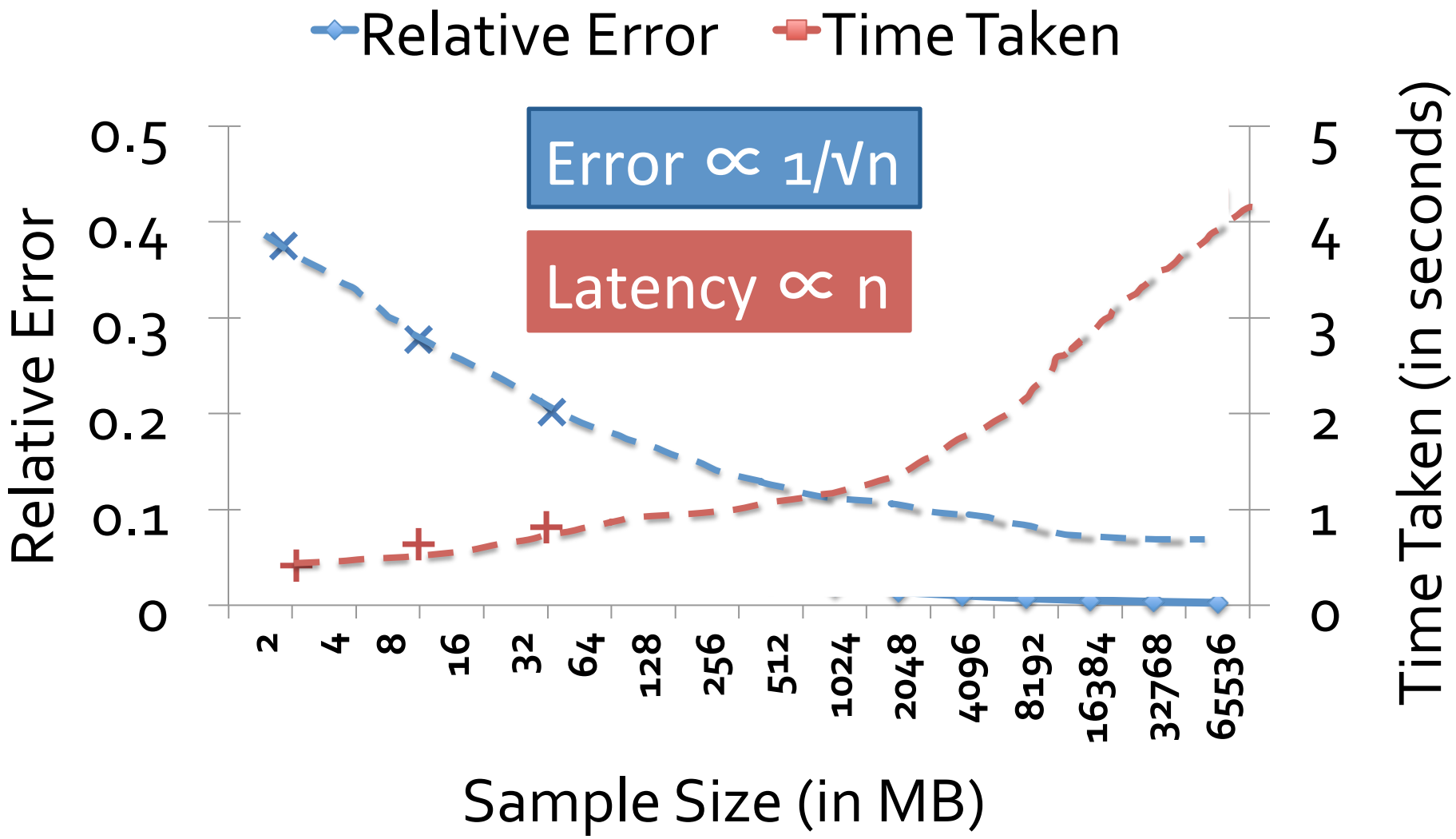
Error Latency Profile (ELP)



Error Latency Profile (ELP)

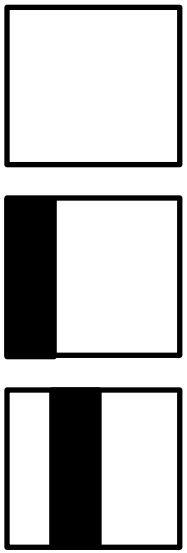


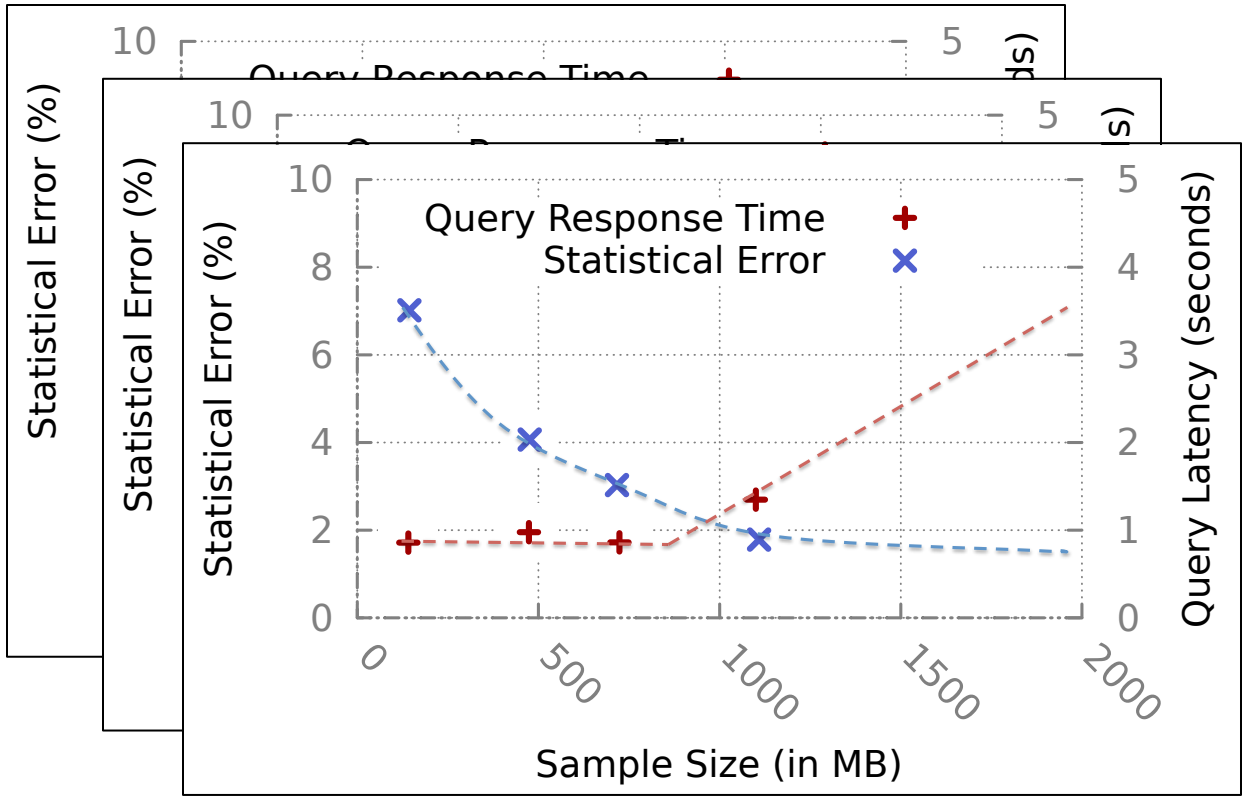
Error Latency Profile (ELP)



Error Latency Profile (ELP)

```
SELECT
foo (*)
FROM
TABLE
WITHIN 2
```

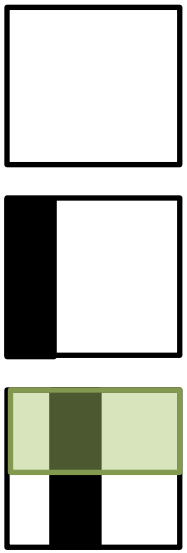


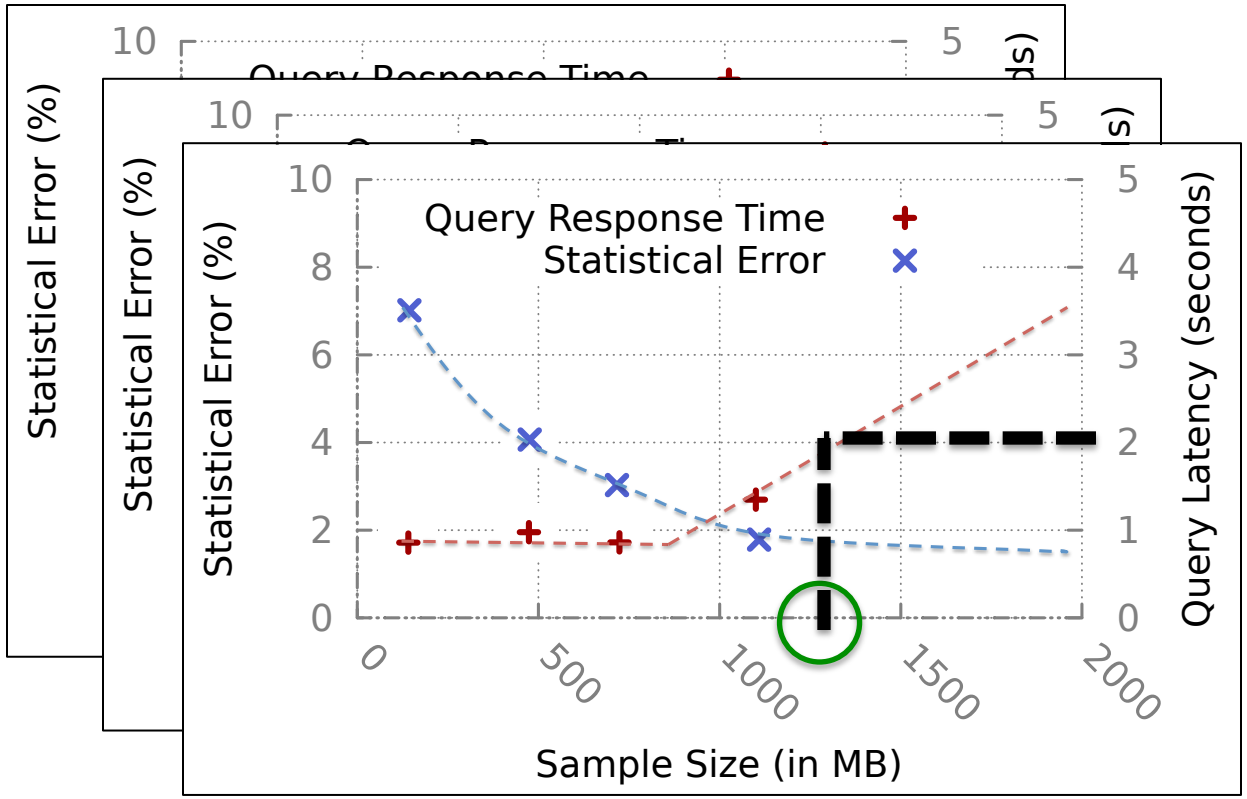


Error Latency Profile (ELP)

```
SELECT
foo (*)
FROM
TABLE
WITHIN 2
```

≈ 400 ms





Challenges

Which set of samples to build given a storage budget?

Which sample to run the query on?

How to accurately estimate the error?

How to Accurately Estimate Error?

Close formulas for limited number of operators
- » E.g., count, mean, percentiles

What about user defined functions (UDFs)?

Experimental Workload

Conviva: 30-day log of media accesses by Conviva users. Raw data 17 TB, partitioned this data across 100 nodes

Log of 20,000 queries

» 43.6% queries have one or more UDFs

Storage budget: 50% of original data

» 8 stratified samples

Experimental Setting

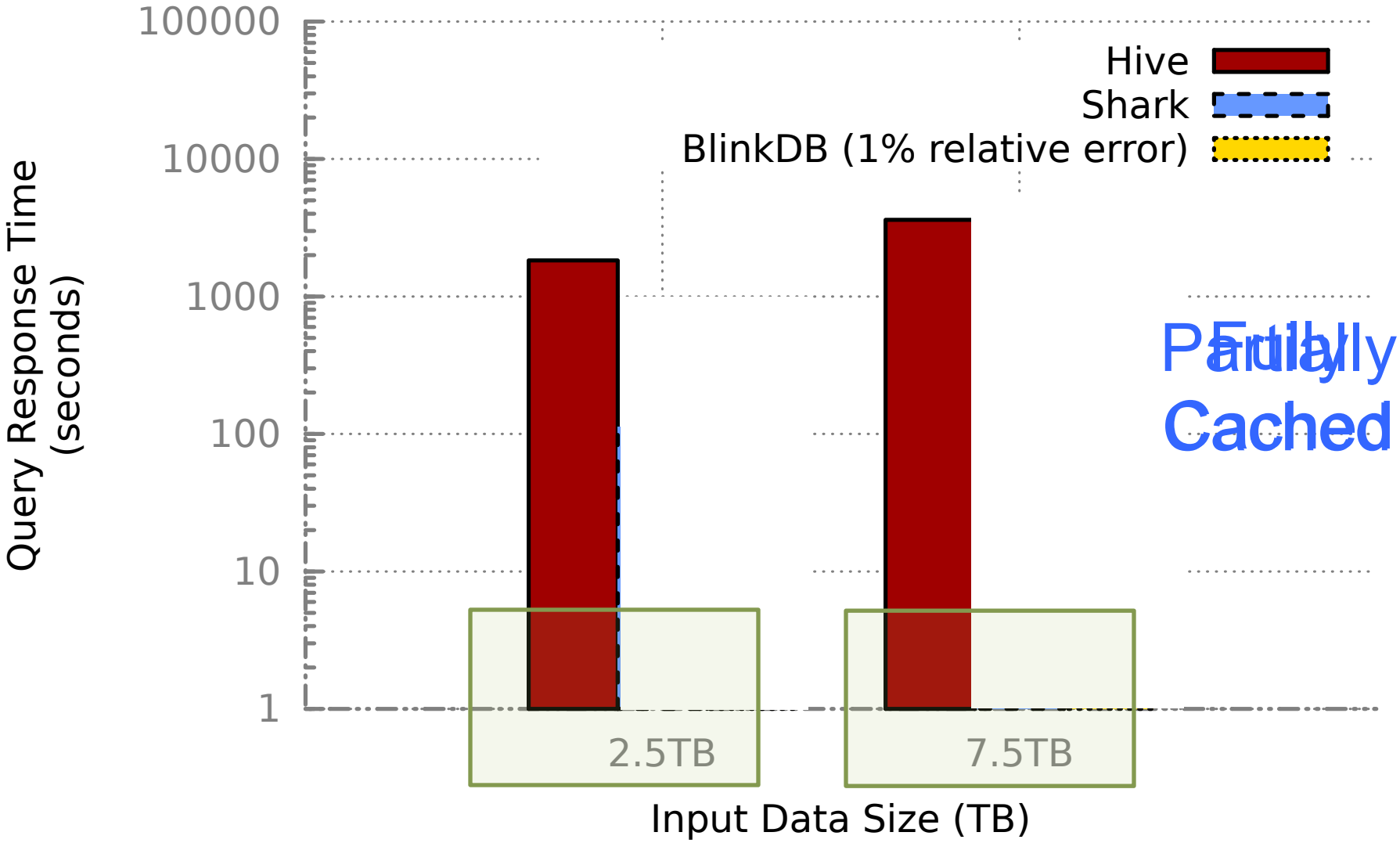
100 node cluster of EC2 extra large instances:

- » 800 cores
- » 6.8TB RAM
- » 75TB disk

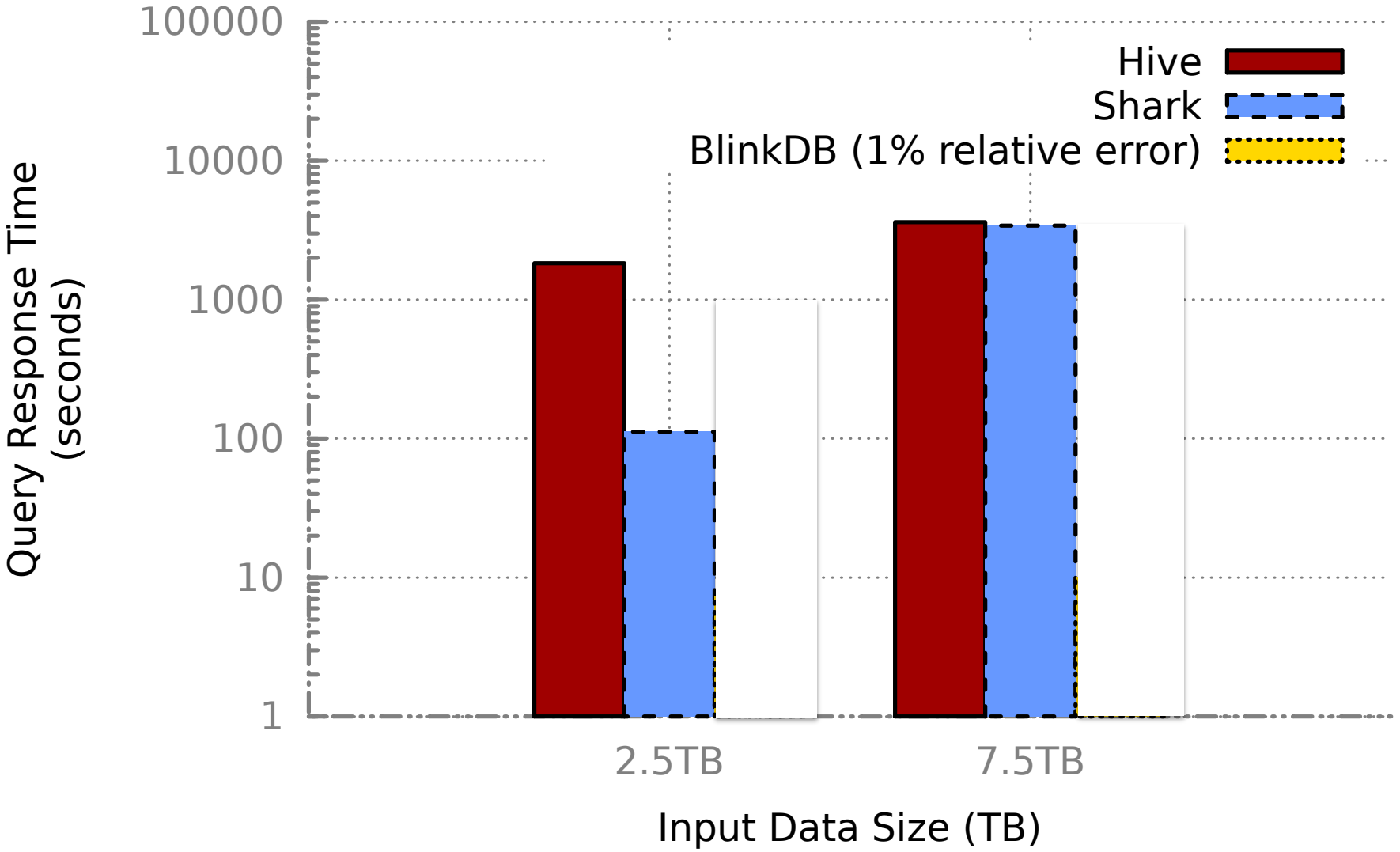
Two datasests: 2.5TB, and 7.5TB, respectively

- » Significantly larger when stored in memory

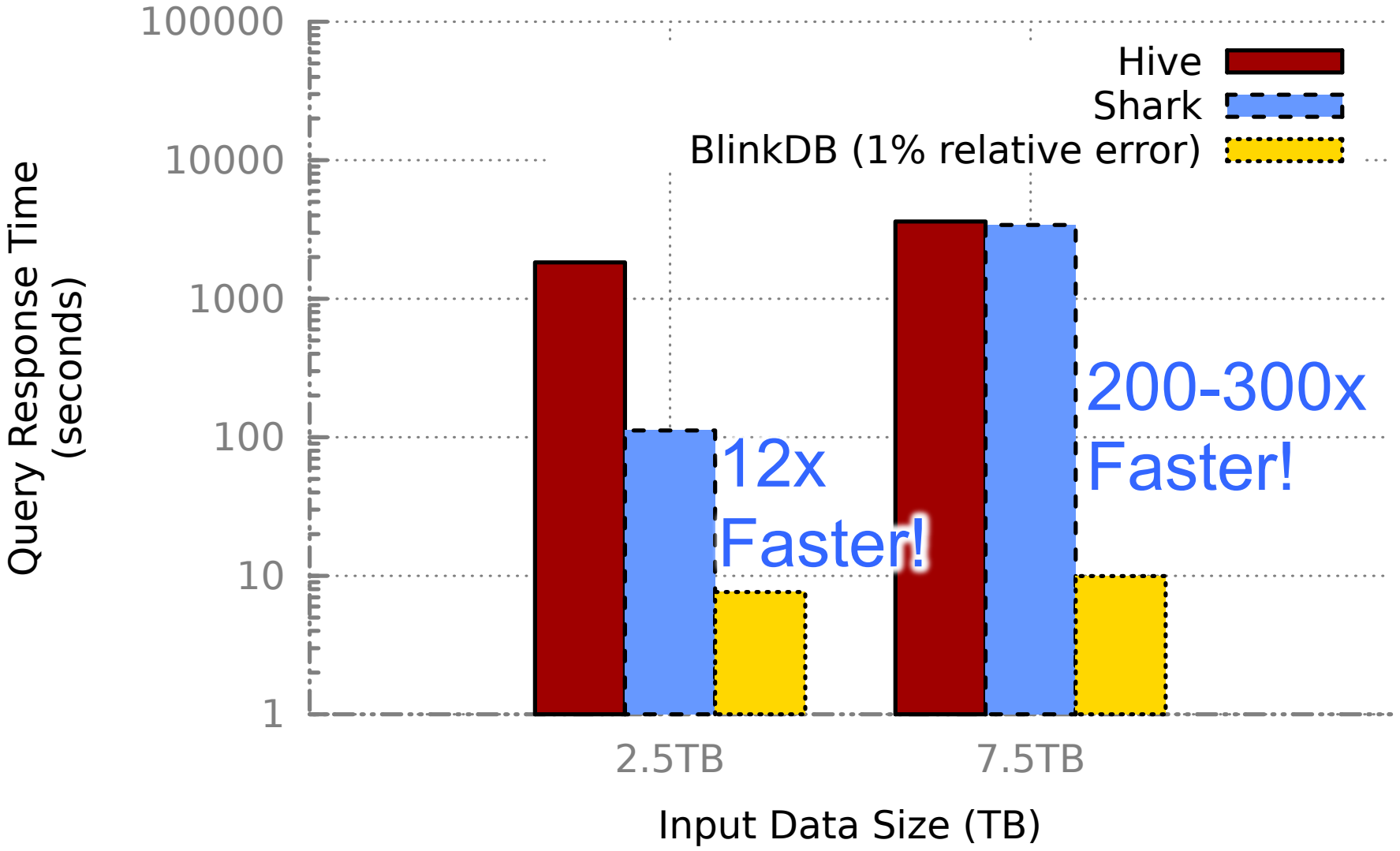
Sampling vs. No Sampling
(close formula)



Sampling vs. No Sampling
(close formula)



Sampling vs. No Sampling
(close formula)



How to Accurately Estimate Error?

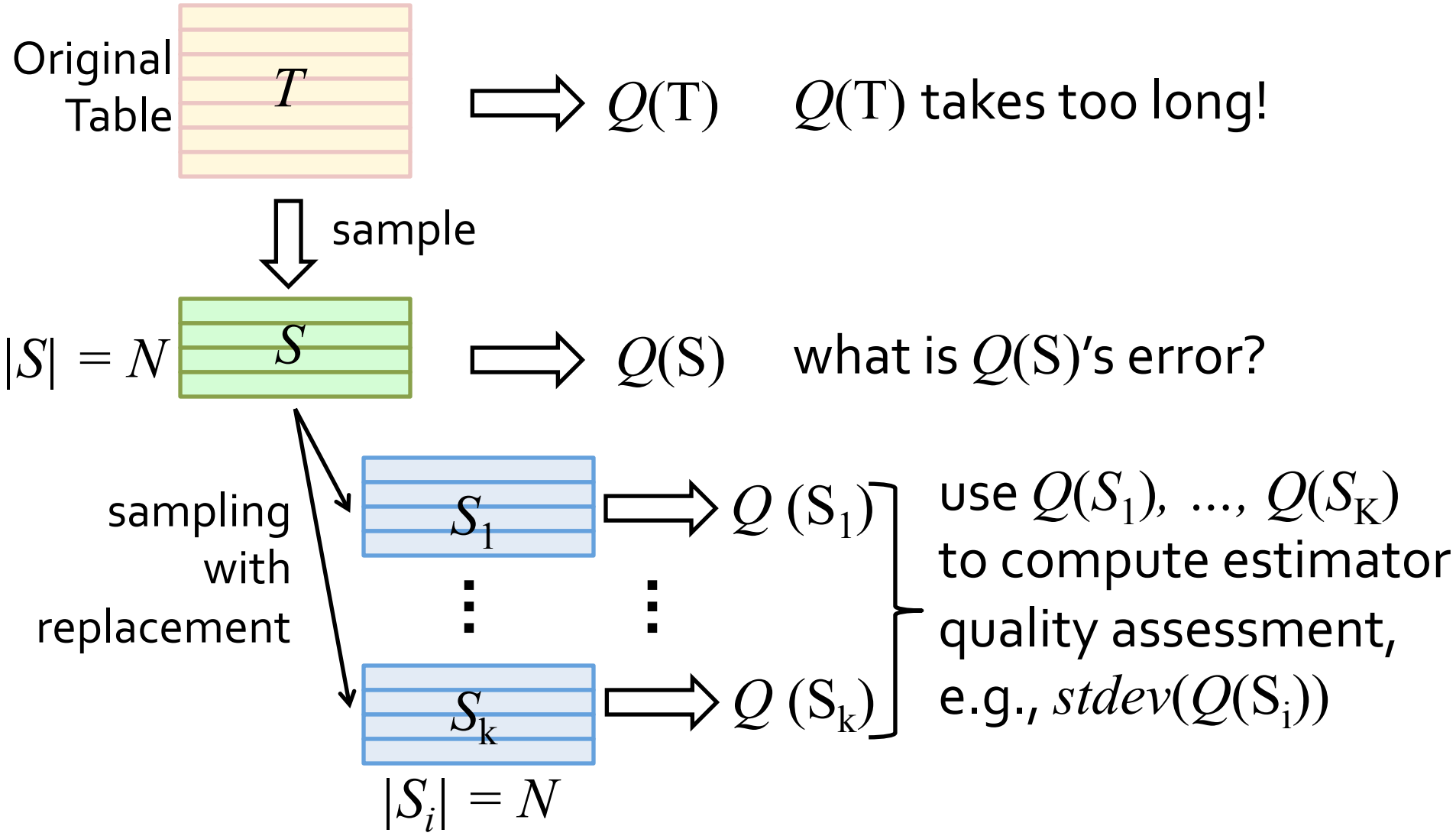
Close formulas for limited number of operators

» E.g., count, mean, percentiles

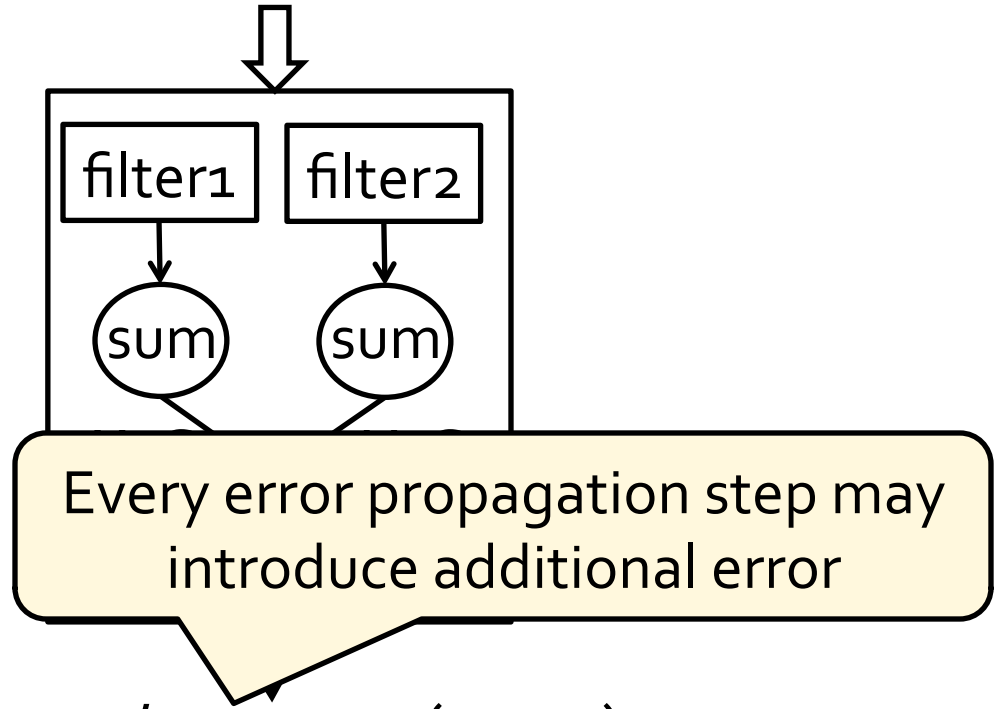
What about user defined functions (UDFs)?

Bootstrap

Quantify accuracy of a query on a sample table

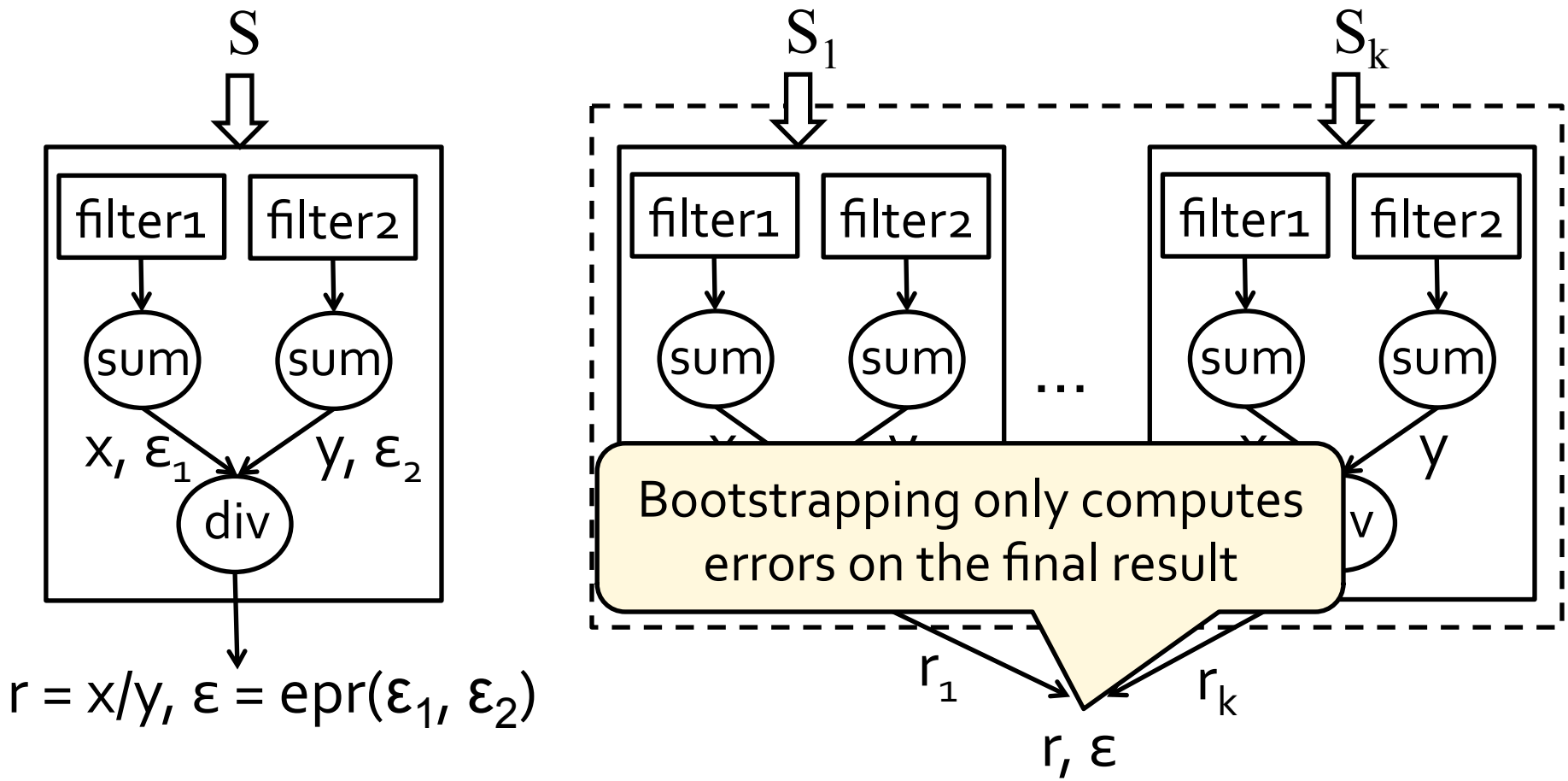


Also Useful for Close Formulas

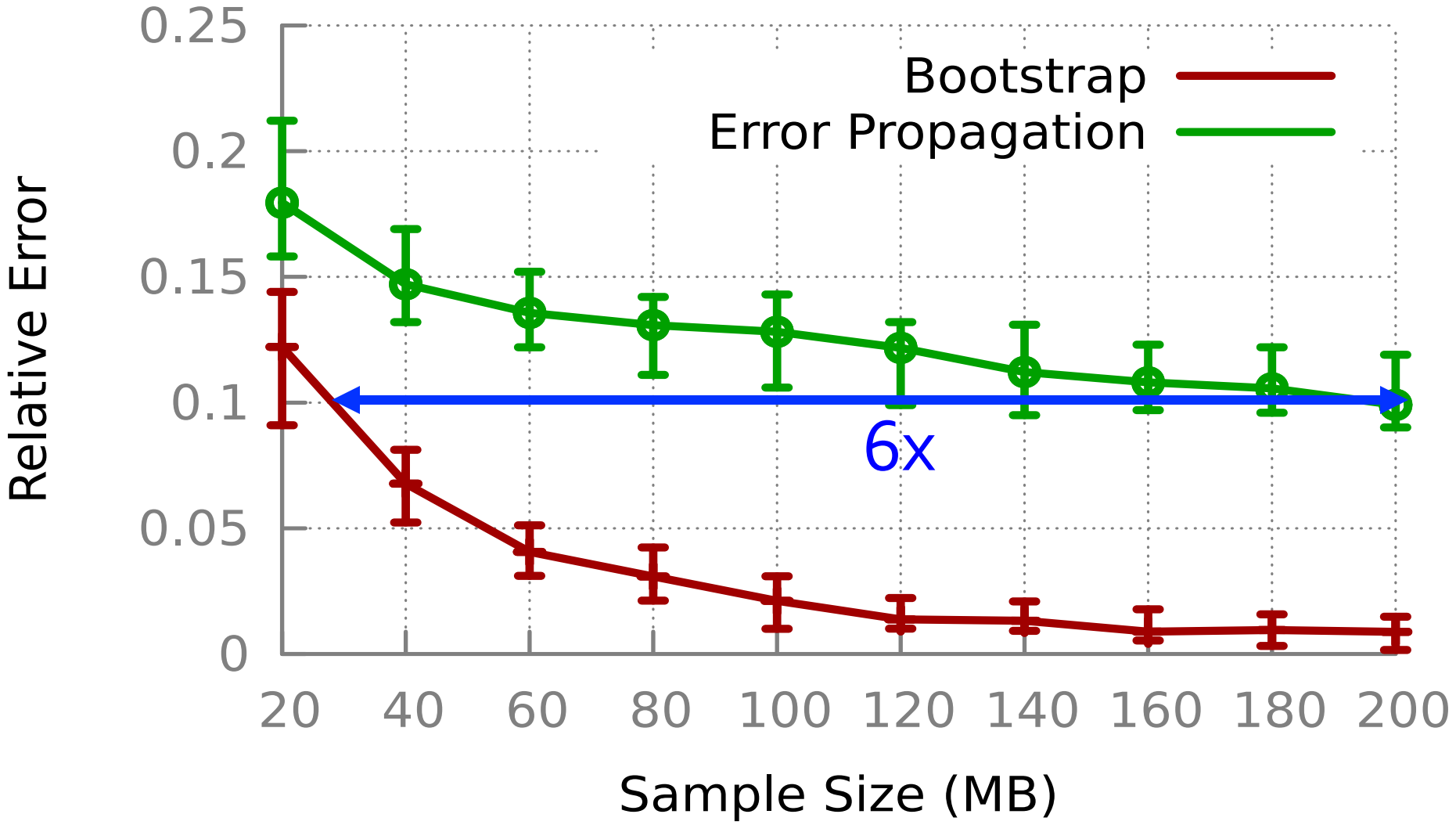


$r = x/y,\ \varepsilon = epr(\varepsilon_1, \varepsilon_2)$

Also Useful for Close Formulas



Also Useful for Close Formulas



Bootstrap Challenges

How do you know the bootstrap is working?

» Depends on distribution, computation, sample size

Overhead

How Do You Know Bootstrap is Working?

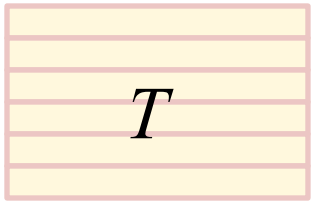
Assumption: $f()$ is Hadamard differentiable

- How do you know an UDF is Hadamard differentiable?
- Only **asymptotic** consistency
- Sufficient, not necessary condition

Developed data-driven diagnostic for Bootstrap

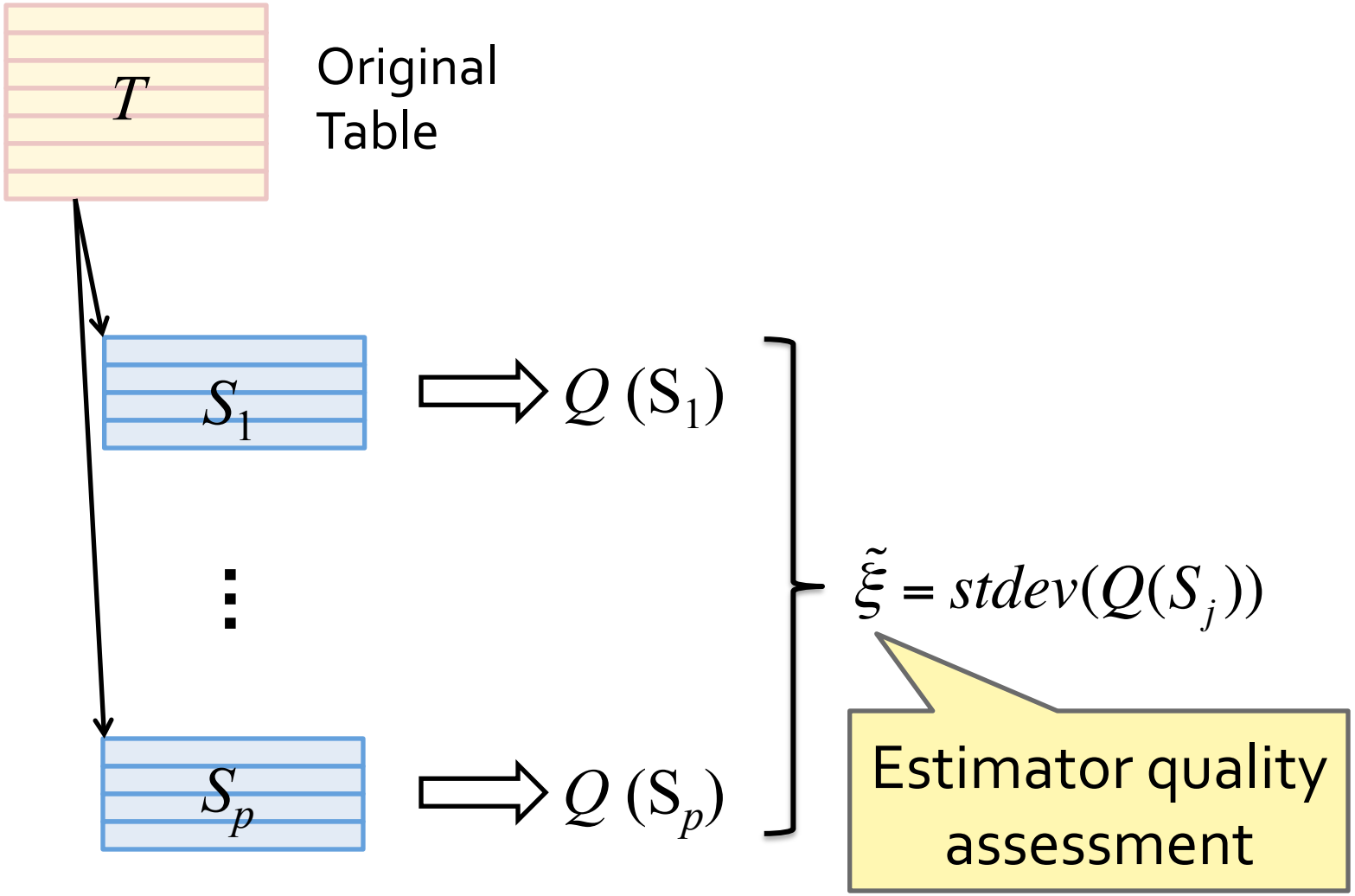
- Compare bootstrapping with **ground truth** for small samples
- Check whether error improves as sample size increases

Ground Truth (Approximation)

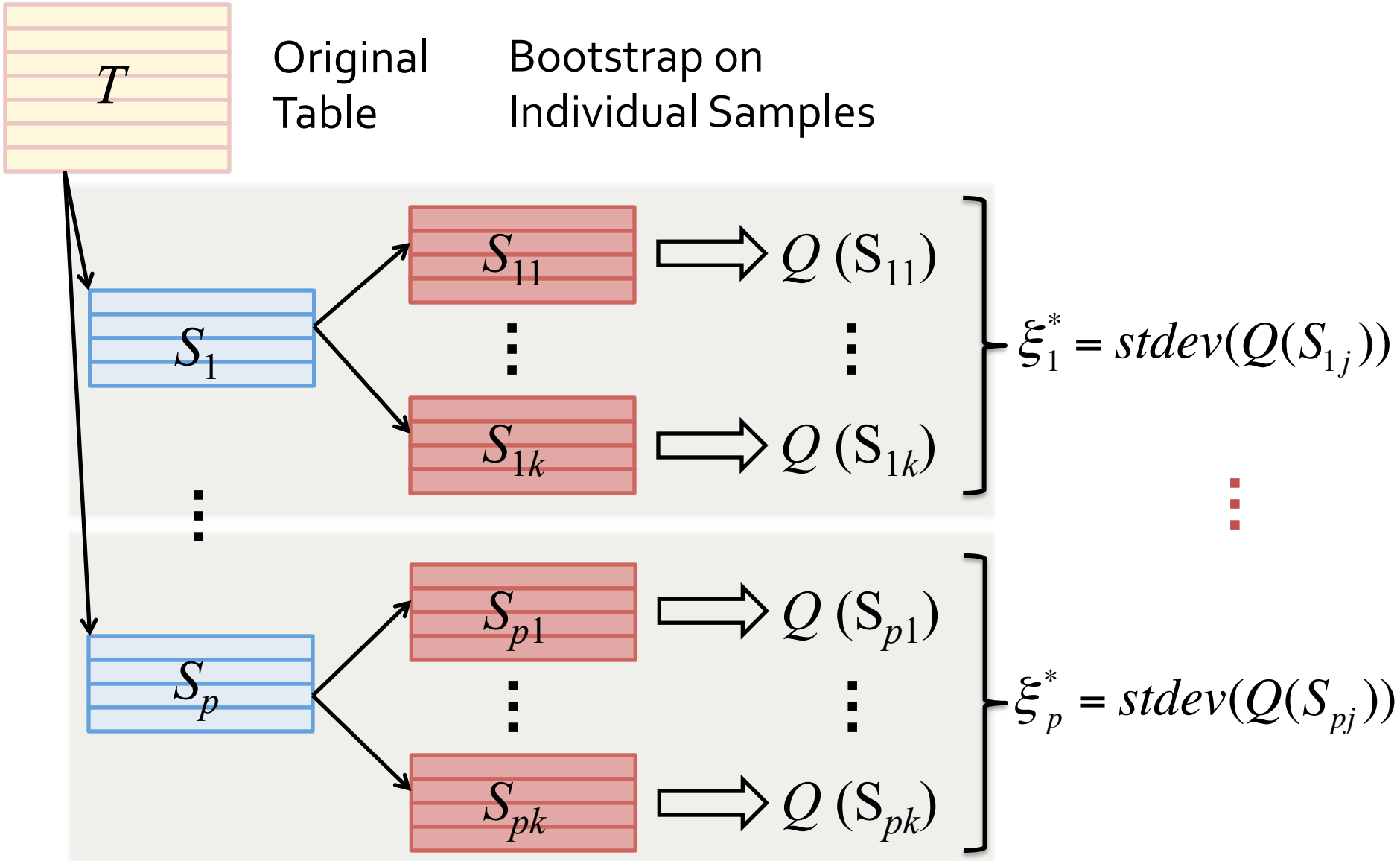


Original
Table

Ground Truth (Approximation)



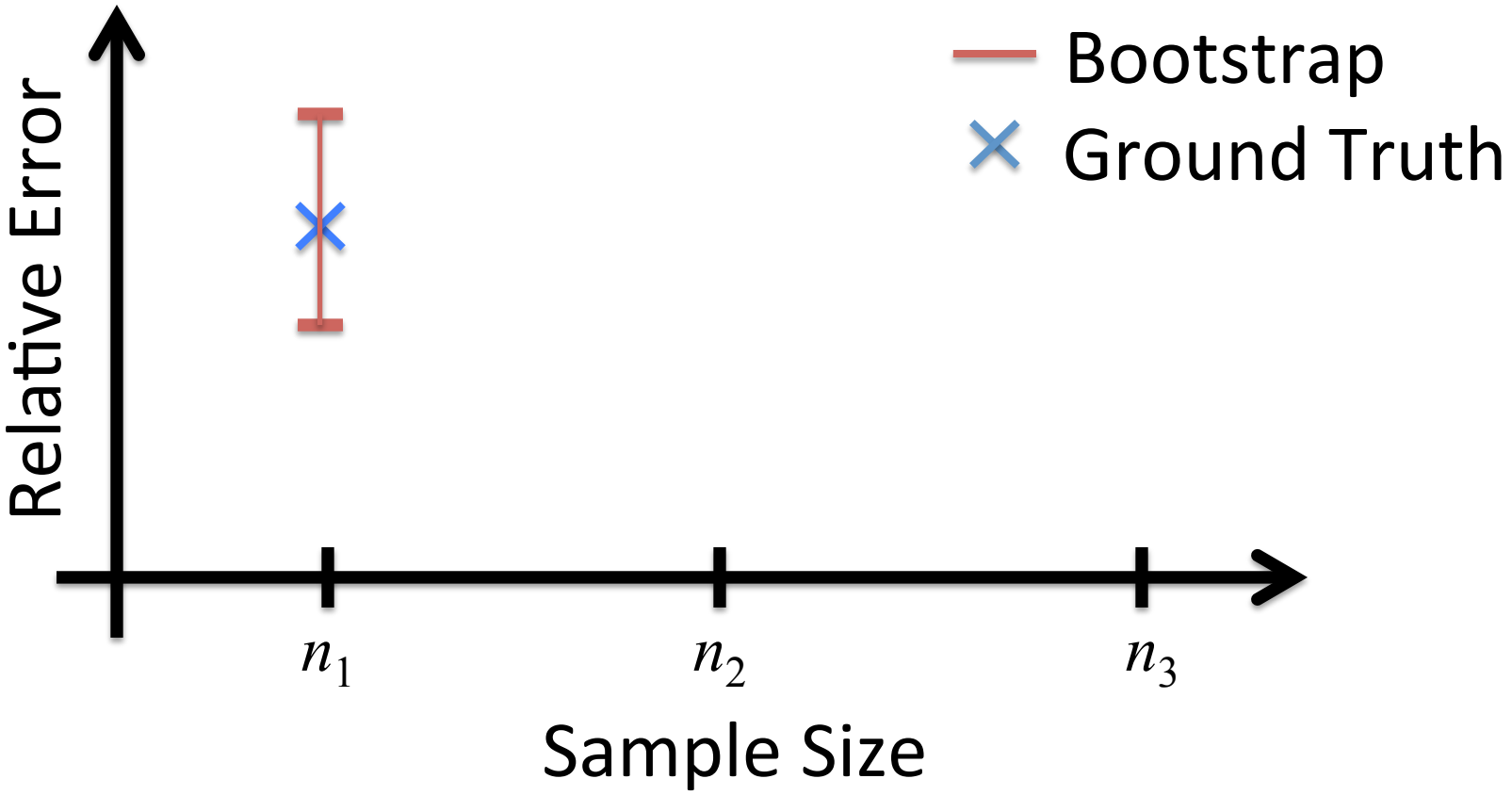
Ground Truth and Bootstrap



Ground Truth vs. Bootstrap

$$\tilde{\xi}_i = mean(\xi_{ij})$$

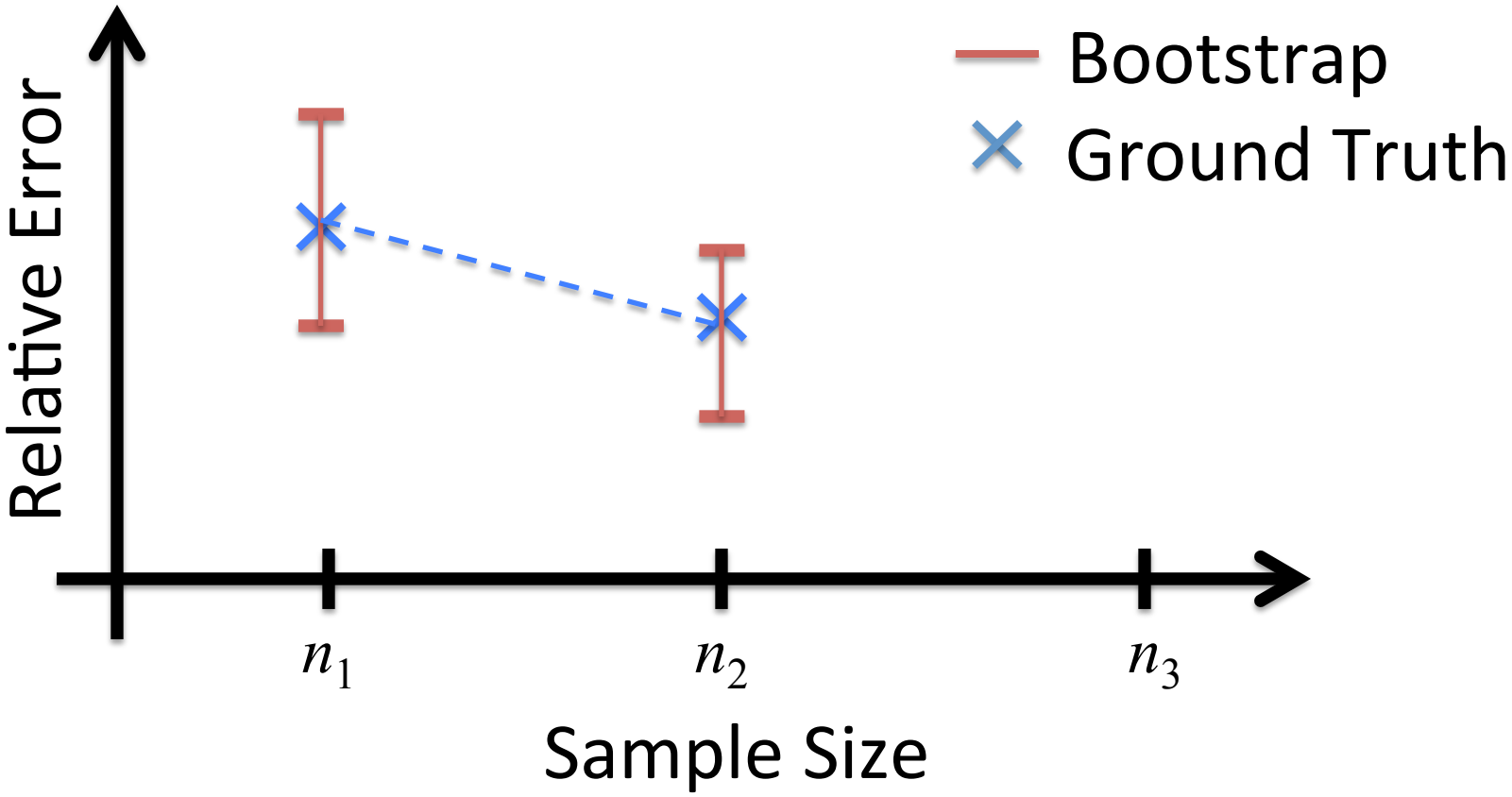
$$\xi^*_{i1} = stdev(Q(S_{i1j})) \quad \cdots \quad \xi^*_{ip} = stdev(Q(S_{ipj}))$$



Ground Truth vs. Bootstrap

$$\tilde{\xi}_i = mean(\xi_{ij})$$

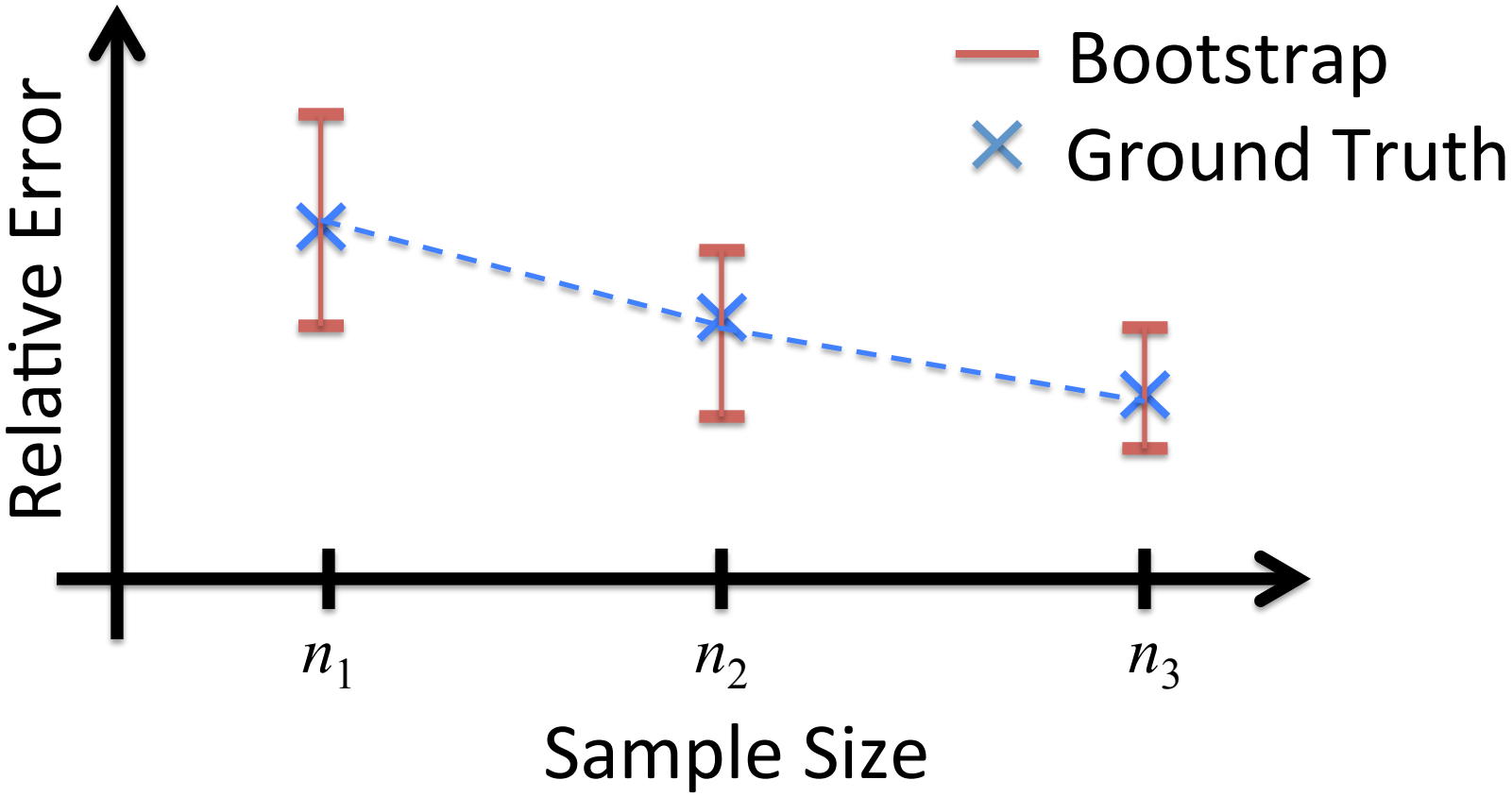
$$\xi^*_{i1} = stdev(Q(S_{i1j})) \quad \cdots \quad \xi^*_{ip} = stdev(Q(S_{ipj}))$$



Ground Truth vs. Bootstrap

$$\tilde{\xi}_i = mean(\xi_{ij})$$

$$\xi^*_{i1} = stdev(Q(S_{i1j})) \quad \cdots \quad \xi^*_{ip} = stdev(Q(S_{ipj}))$$



Bootstrap Diagnosis

Expectation test:

» Bootstrap results should not deviate "too much" from ground truth, on average

Standard deviation test:

» Bootstrap results should not vary "too much"

Confidence interval test:

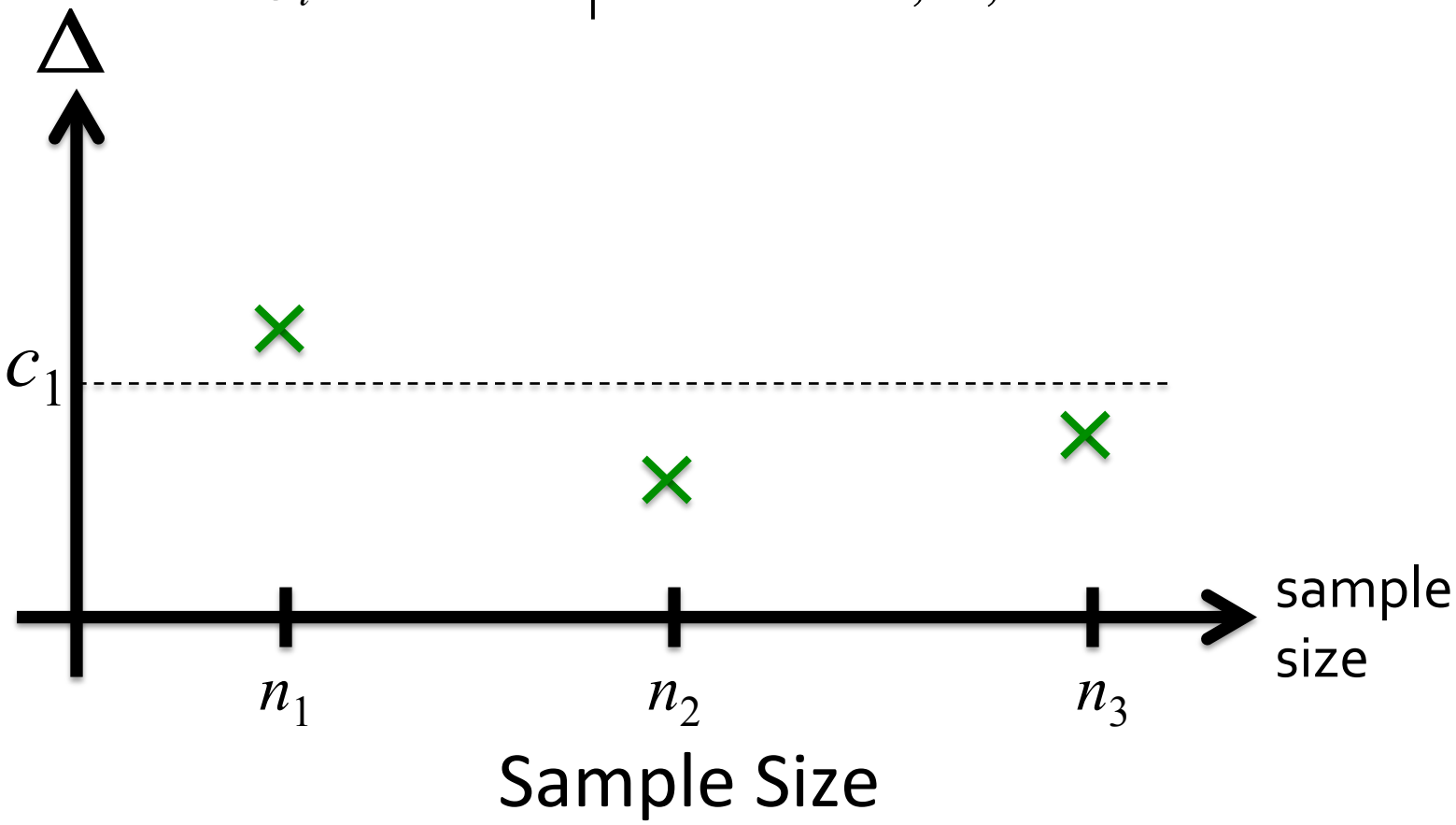
» Most (e.g., 95%) of bootstrap results should be "close" to ground truth

Expectation Test

$$\Delta_i \leftarrow \left| \frac{mean(\xi^*_{i1},\ldots,\xi^*_{ip}) - \tilde{\xi}_i}{\tilde{\xi}_i} \right|$$

$$(\Delta_{i+1} < \Delta_i) \vee (\Delta_{i+1} \leq c_1)$$

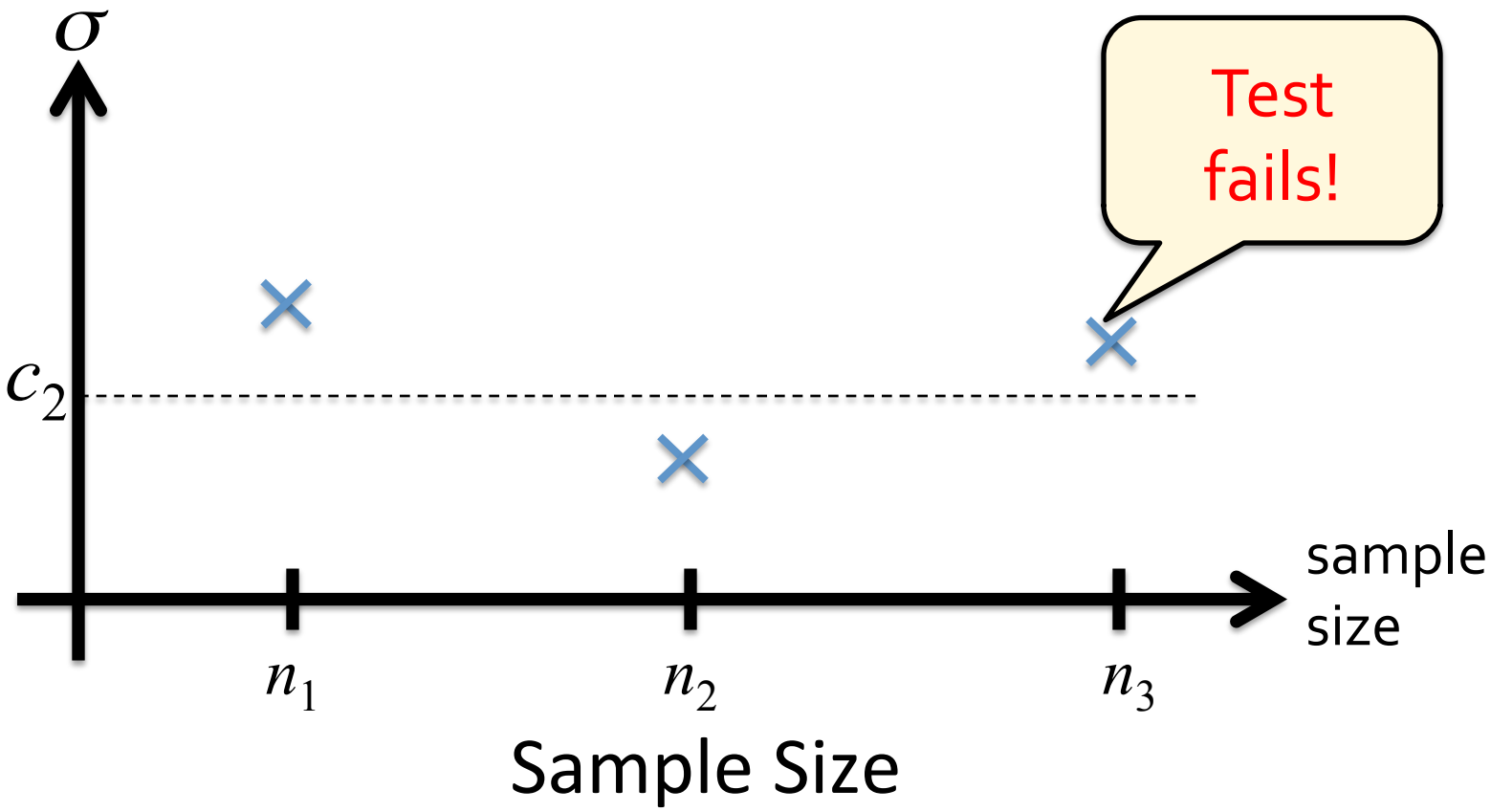
$$\forall i = 1,\ldots,s$$



Standard Deviation Test

$$\sigma_i \leftarrow \left| \frac{stddev(\xi^*_{i1},\ldots,\xi^*_{ip})}{\tilde{\xi}_i} \right|$$

$$(\sigma_{i+1} < \sigma_i) \vee (\sigma_{i+1} \leq c_2),$$
$$\forall i = 1,\ldots,s$$



Confidence Interval Test

$$\frac{\#\left\{ j \in 1,\ldots,p : \left| \frac{\xi_{ij}^{*} - \tilde{\xi}_i}{\tilde{\xi}_i} \right| \leq c_3 \right\}}{p} \geq \alpha$$

Fraction α of bootstrap results have rel. error of at most c_3 from **ground truth** (e.g., $\alpha = 0.95,\ c_3 = 0.5$)

How Well Does it Work in Practice?

Evaluated on **268** real-world Conviva Queries of which **113** had custom User-Defined Functions

Diagnostic predicted that **207 (77%)** queries can be approximated

- » False Positives: 3 (conditional UDFs)
- » False Negatives: 18

Bootstrap Challenges

How do you know the bootstrap is working?

» Depends on distribution, computation, sample size

Overhead

Very, Very Preliminary Results

Setting:

- » 25 EC2 instances with 4 slots and 15GB RAM
- » Input: 365mil rows, 204GB on disk, > 600GB in memory (deserialized format)
- » Workload: query computing 95-th percentile

Overheads:

- » Bootstrap to estimate result's error
- » Bootstrap diagnosis

Query Resp. Time & Overhead

Operation	Computation complexity	I/O complexity
Full data	$C(N)$	$O(N)$

N – data size

Query Resp. Time & Overhead

Operation	Computation complexity	I/O complexity
Full data	$C(N)$	$O(N)$
Sample	$C(n)$	$O(n)$

N – data size
n – sample size

Query Resp. Time & Overhead

Operation	Computation complexity	I/O complexity
Full data	$C(N)$	$O(N)$
Sample	$C(n)$	$O(n)$
Bootstrap	$k \times C(n)$	$k \times O(n)$

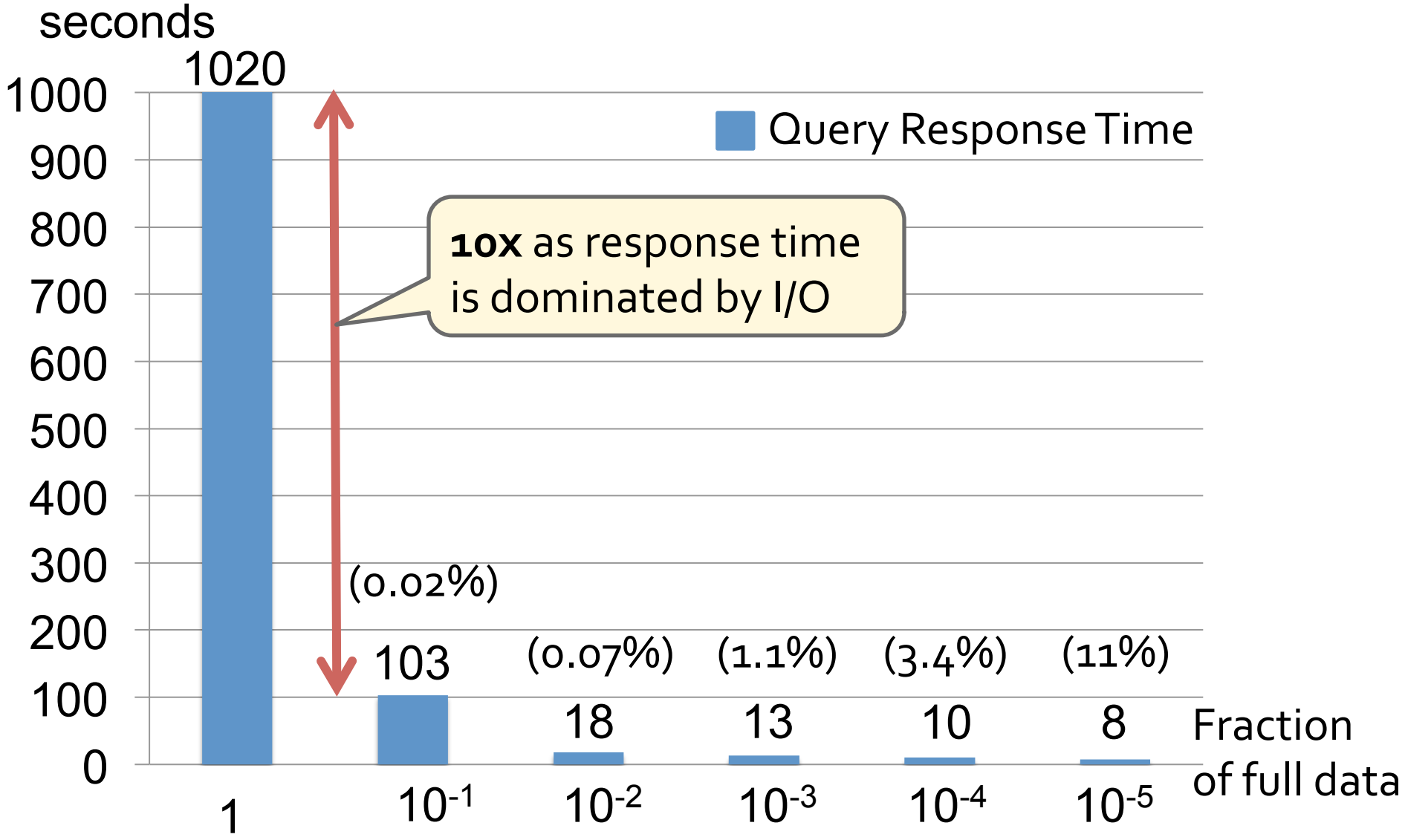
N – data size
n – sample size
k – # of samples used by bootstrap

Query Resp. Time & Overhead

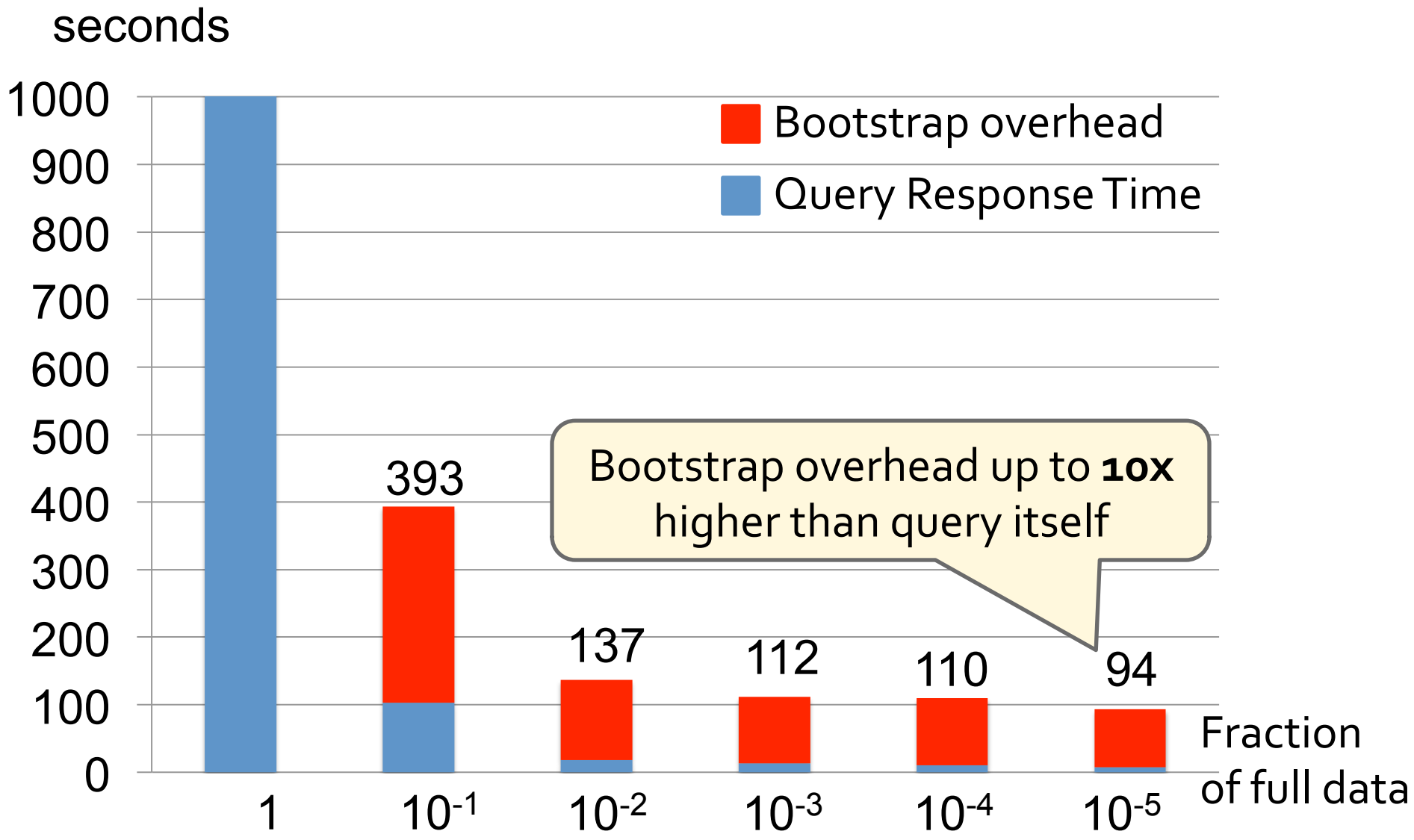
Operation	Computation complexity	I/O complexity
Full data	$C(N)$	$O(N)$
Sample	$C(n)$	$O(n)$
Bootstrap	$k \times C(n)$	$k \times O(n)$
Diagnosis	$p \times k \times \sum_{i=1}^{s} C(n_i)$	$p \times k \times \sum_{i=1}^{s} O(n_i)$

N – data size
n – sample size
n_i – sample size for iteration i of diagnostic
k – # of samples used by bootstrap
p – # of samples used by ground truth
s – # of iteration used by diagnostic

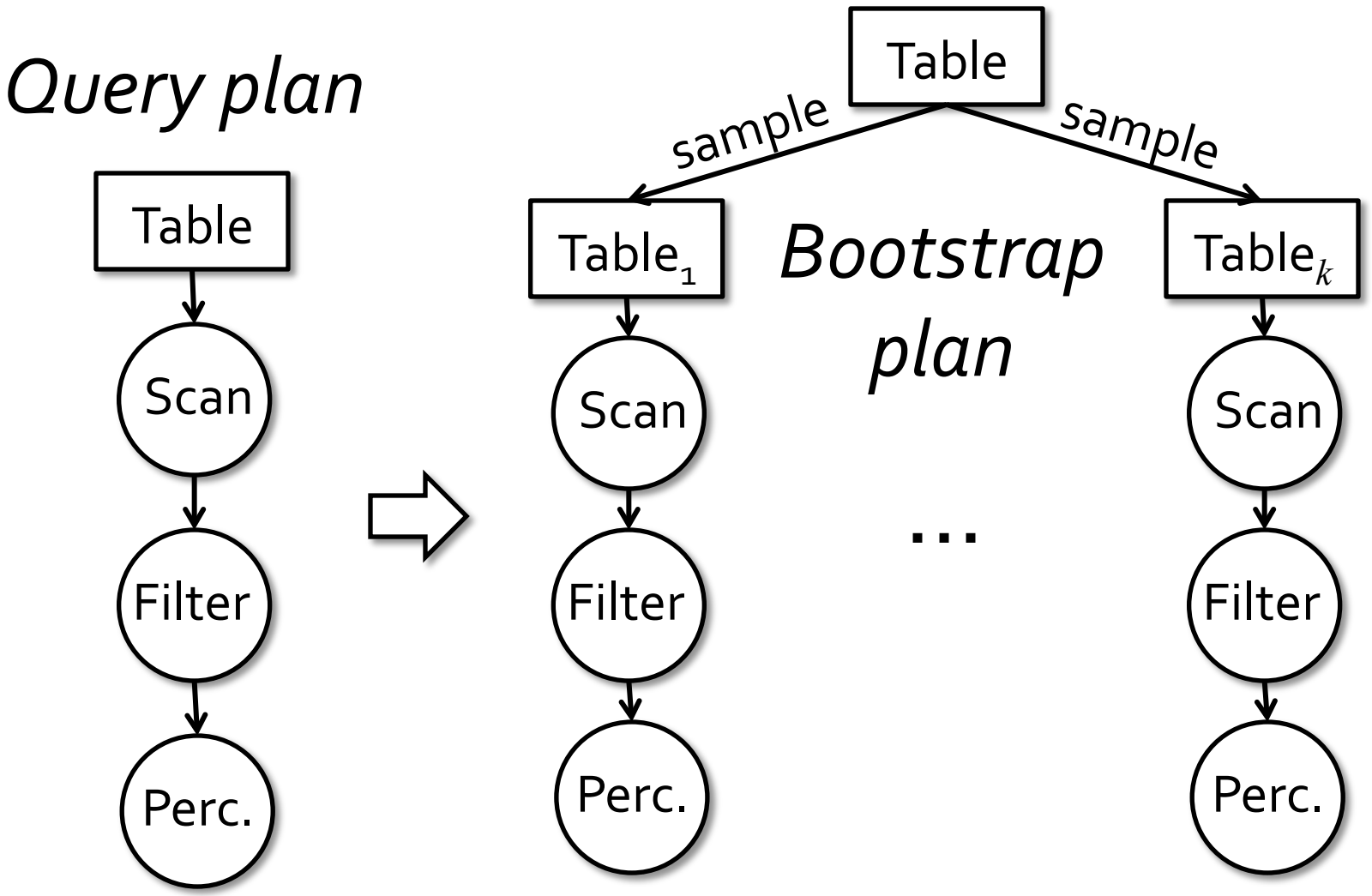
Query Response Time



Query Resp. Time + Bootstrap

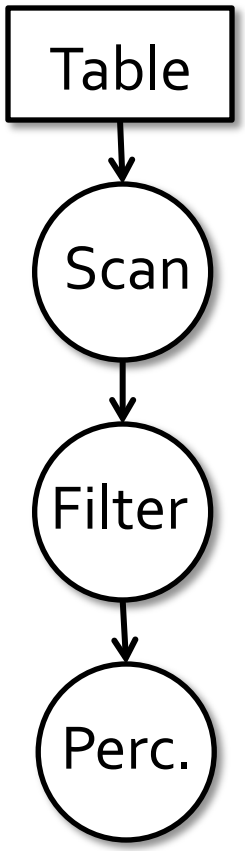


Bootstrap Query Plan



Detailed Query Plan

Query plan

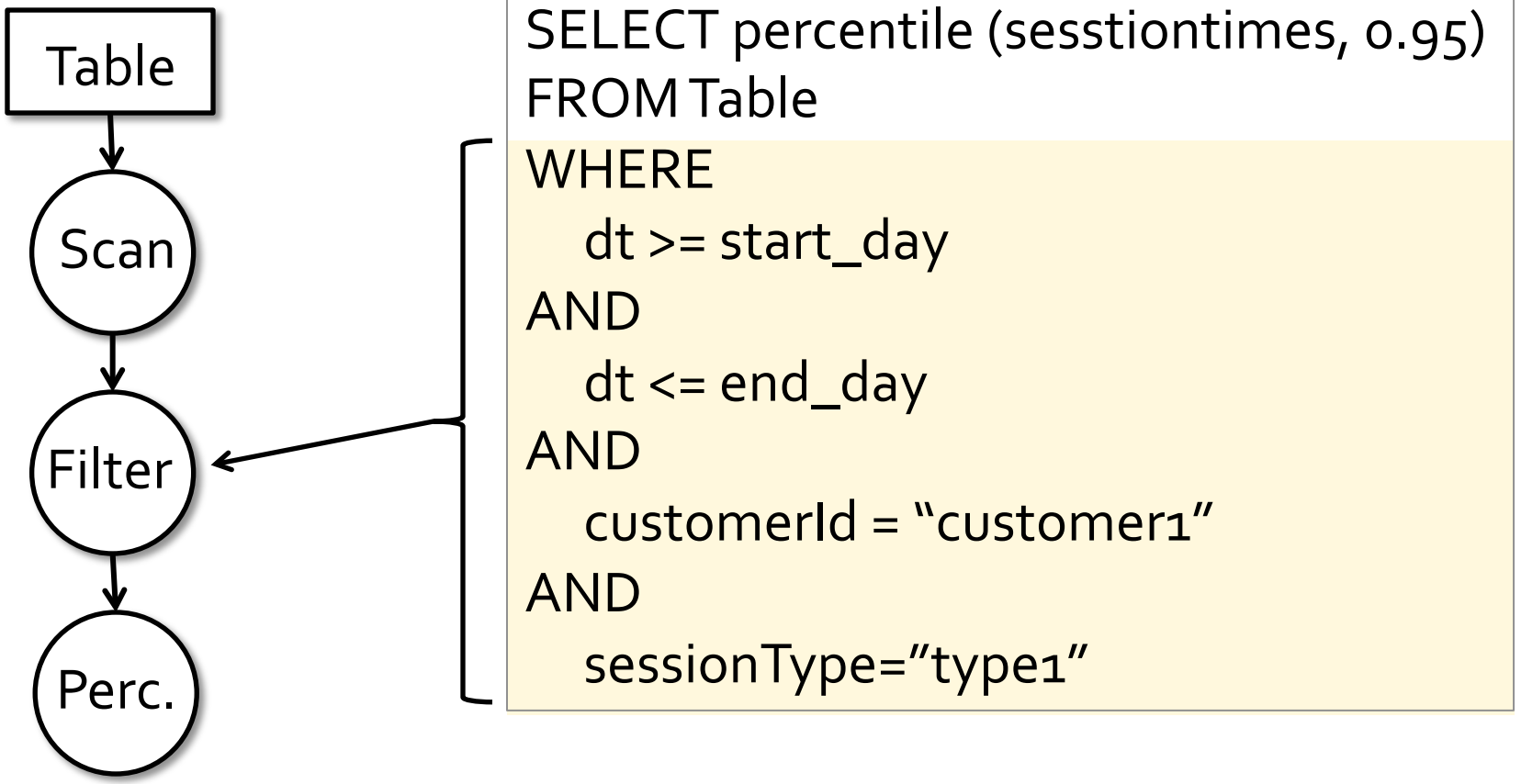


Query

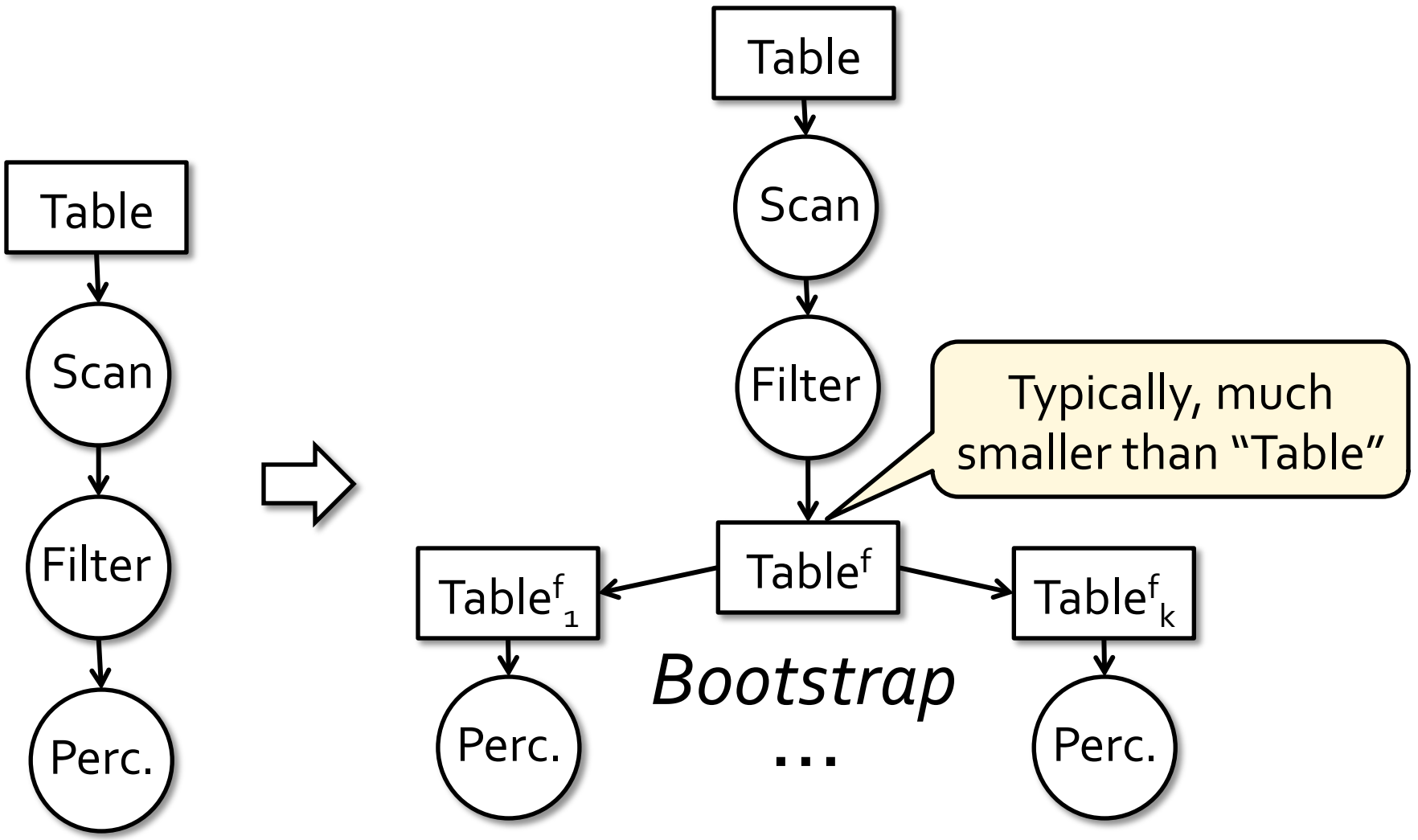
```
SELECT percentile (sesstiontimes, 0.95)
FROM Table
WHERE
    dt >= start_day
AND
    dt <= end_day
AND
    customerId = "customer1"
AND
    sessionType="type1"
```

Detailed Query Plan

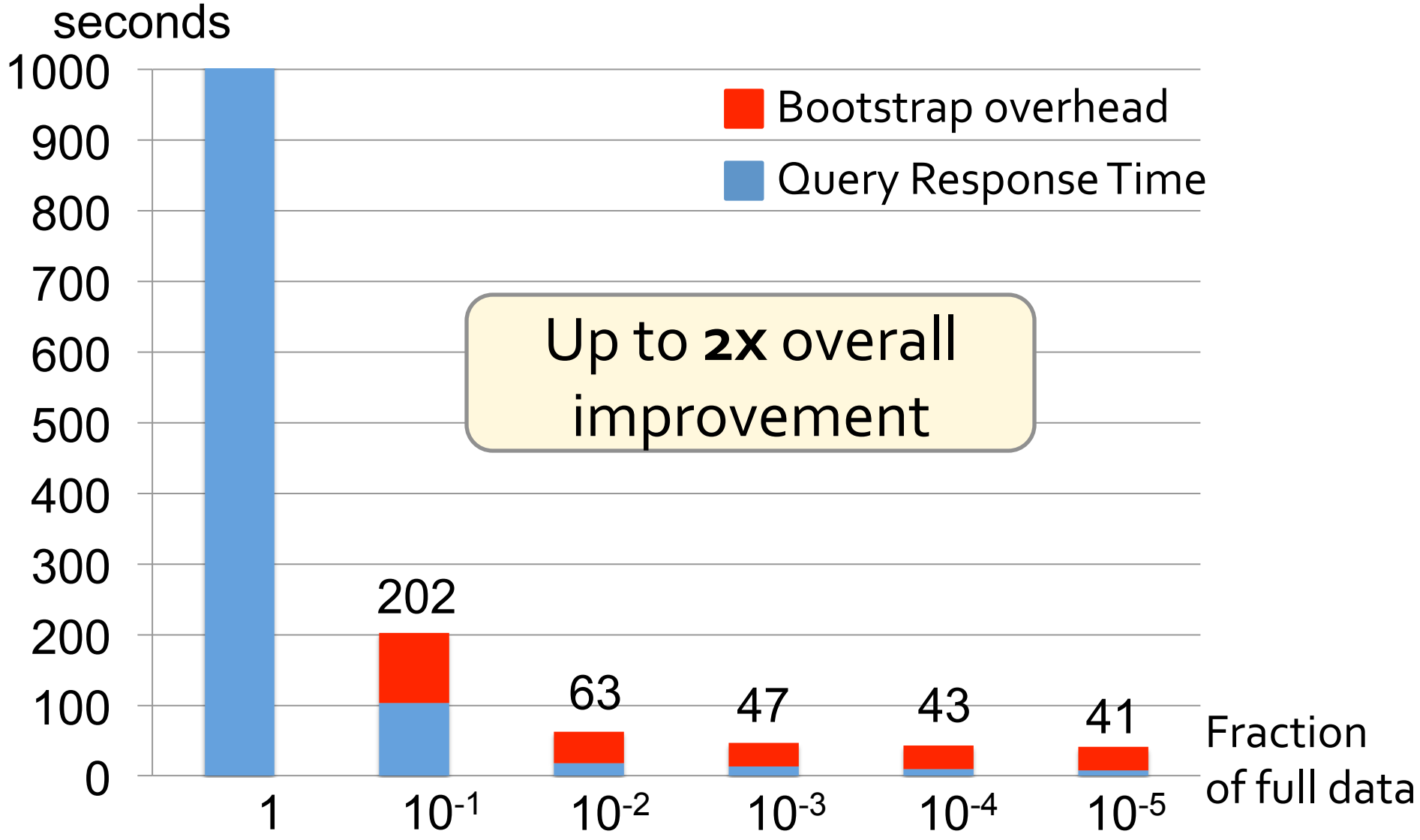
Typically, filters remove ***many*** rows from input table



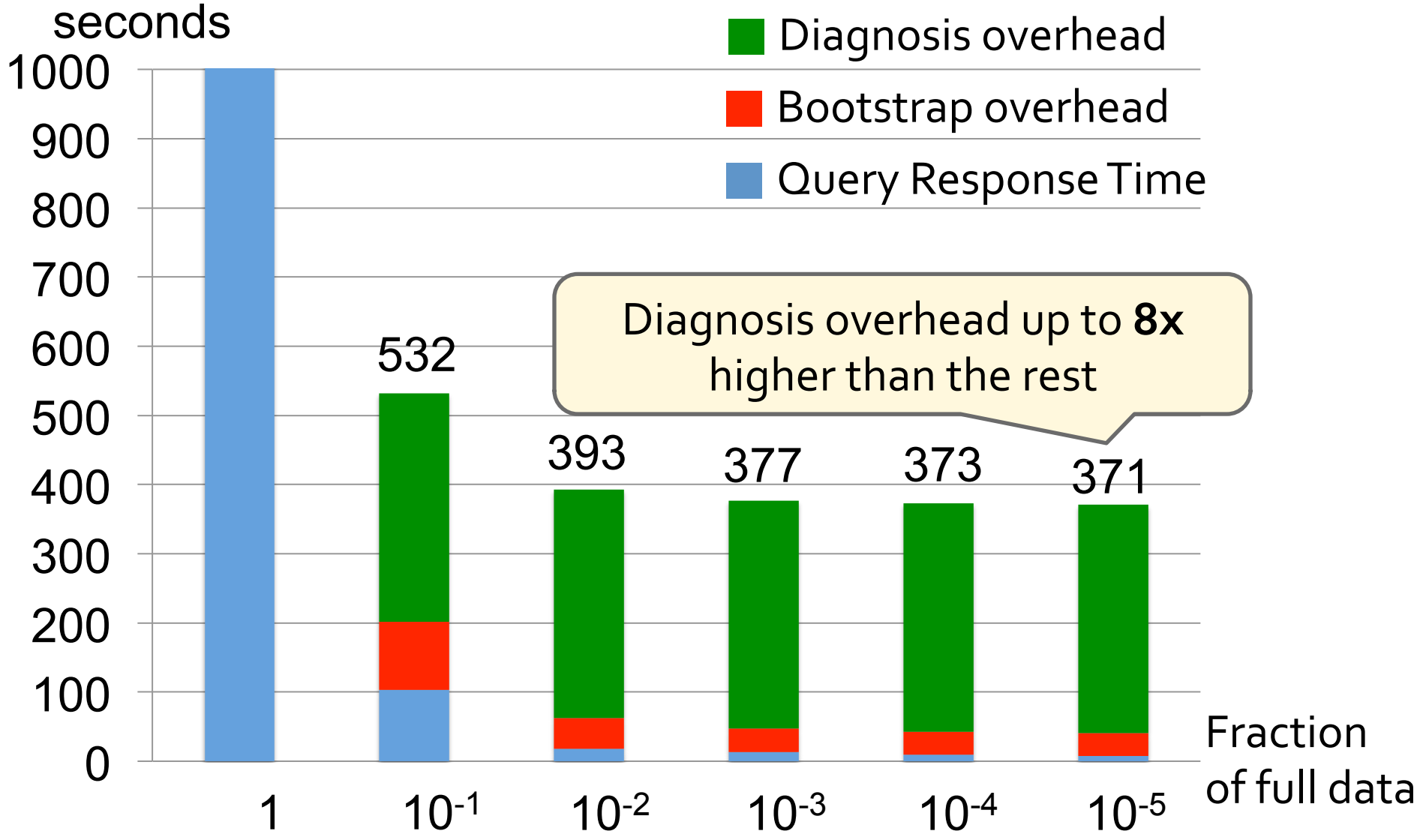
Filter Pushdown Optimization



Query + Bootstrap
(with Filter Pushdown)



Query + Bootstrap + Diagnosis
(with Filter Pushdown)



Overhead

Operation	Comp. complexity	Comm. complexity	# of tasks
Full data	$C(N)$	$O(N)$	$O(d)$
Sample	$C(n)$	$O(n)$	$O(d)$
Bootstrap	$k \times C(n)$	$k \times O(n)$	
Diagnosis	$p \times k \times \sum_{i=1}^{s} C(n_i)$	$p \times k \times \sum_{i=1}^{s} O(n_i)$	$s \times p \times k \times O(d)$

Can generate millions of tasks

N – data size
n – sample size
n_i – sample size for iteration i of diagnostic

k – # of samples used by bootstrap
p – # of samples used by ground truth
s – # of iteration used by diagnostic
d – degree of parallelism

Diagnosis Optimization

Problem: too many tasks to launch & schedule

- » Fixed overhead dominates

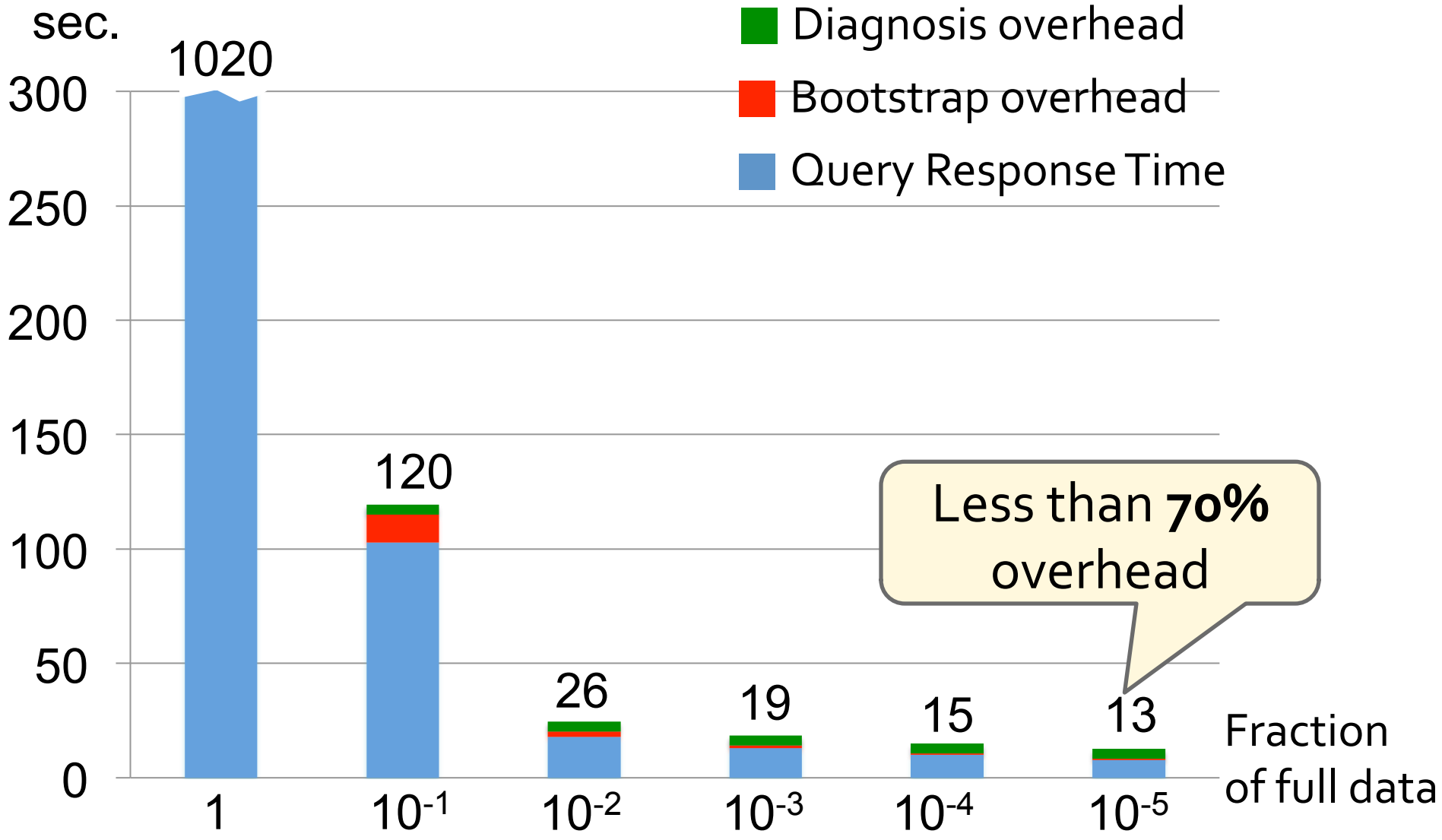
Solution: task consolidation

- » Consolidate the computation into d tasks, where d is the number of slots in the system
- » Caveat: currently a hack; not part of BlinkDB codebase

Result:

- » Reduce diagnosis from 330s to 4.5s
- » Reduce bootstrap overhead by up to 10X

Query + Bootstrap + Diagnosis
(with Filter Pushdown and Task Consolidation)



Summary

*Computing error bounds for approximate queries on massive data sets: a **hard**, **important**, and **exciting** problem*

At the intersection between:

- » ML: error computation
- » Databases: query plan optimization, sample creation and selection
- » Systems: improved parallelism, scheduling

Preliminary results encouraging

Future Work

Improve bootstrap diagnostic:

- » Provable properties for specific settings

Improve coverage for error estimation

- » E.g., use static analysis to decompose programs in multiple Hadamard differentiable components

Improve Bootstrap scalability with Bag of Little Bootstraps [Kleiner et al. 2012]:

- » No need to distribute query computation even for huge samples (e.g., 100 billion records @ 1KB per record)

Scheduling concurrent parallel, dependent jobs …