Duality Between Data and Approximation Error

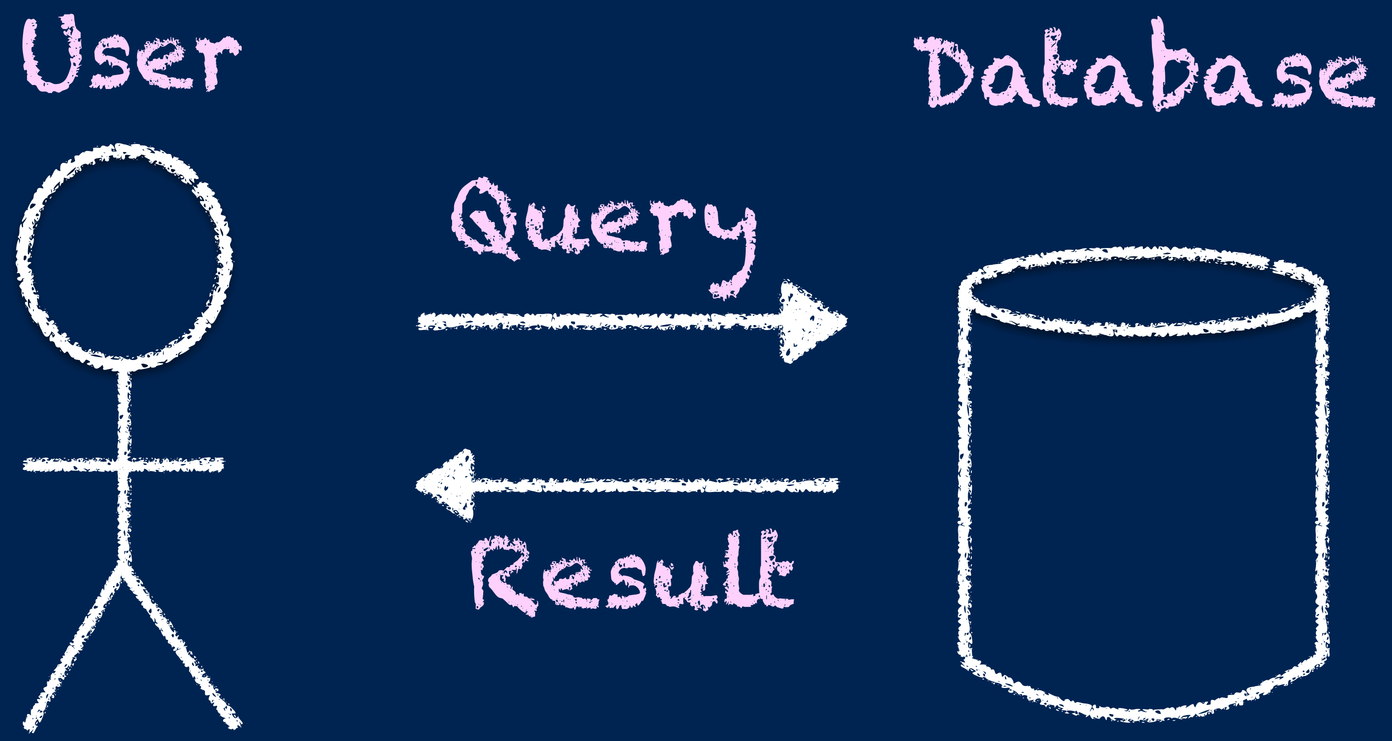
Sanjay Krishnan @ MemSQL
March 12, 2015

In Collaboration With: Jiannan Wang, Daniel Haas, Juan Sanchez, Eugene Wu, Wenbao Tao, Tim Kraska, Tova Milo, Michael Franklin, Ken Goldberg

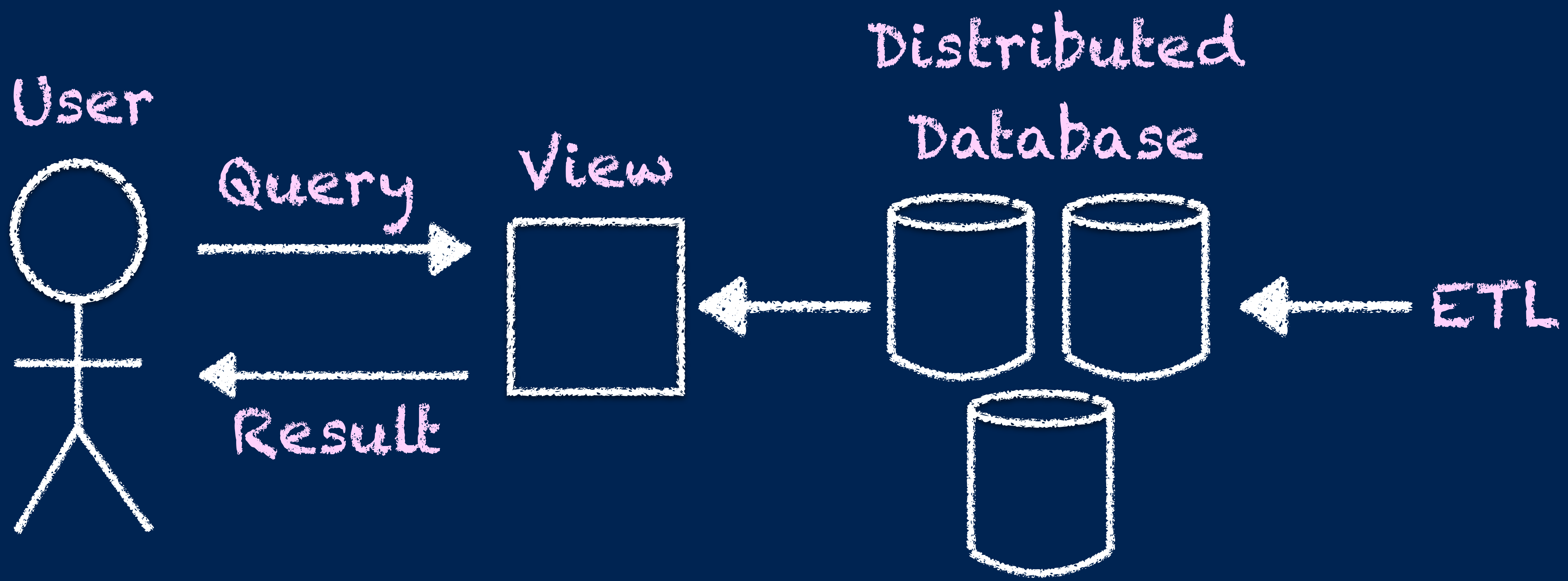
Outline

- Not all errors are created equal
- Approximating Materialized Views
- Results

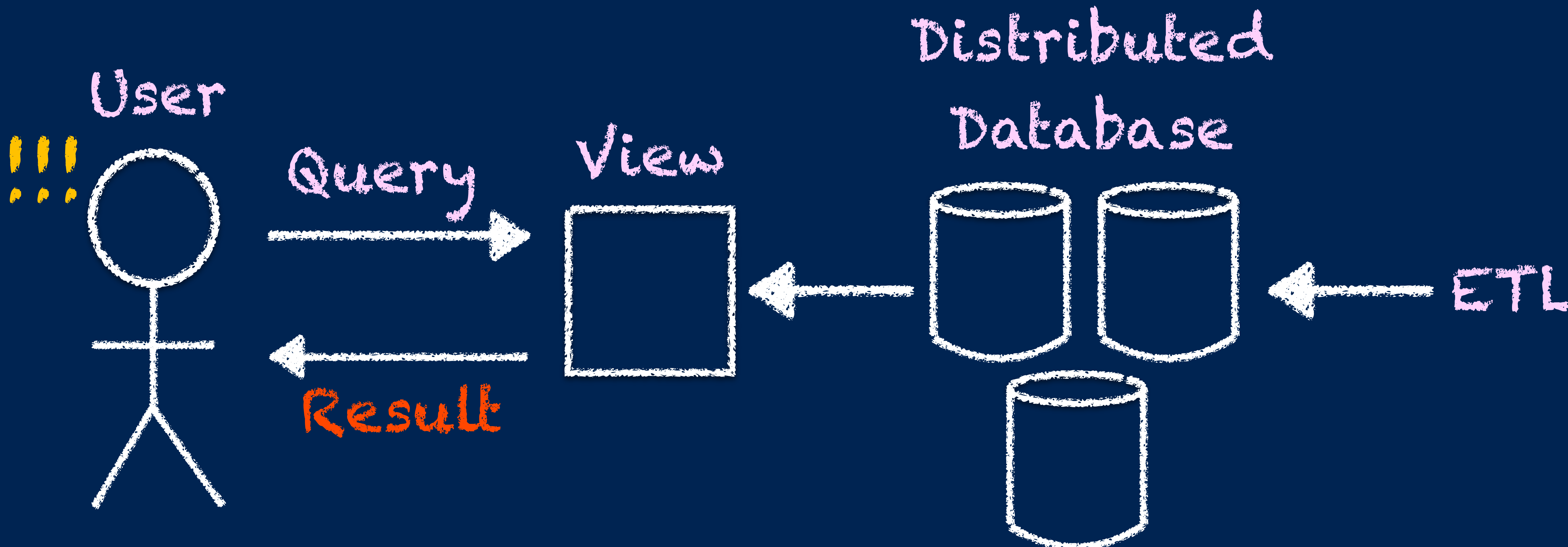
Database-User Interaction Model



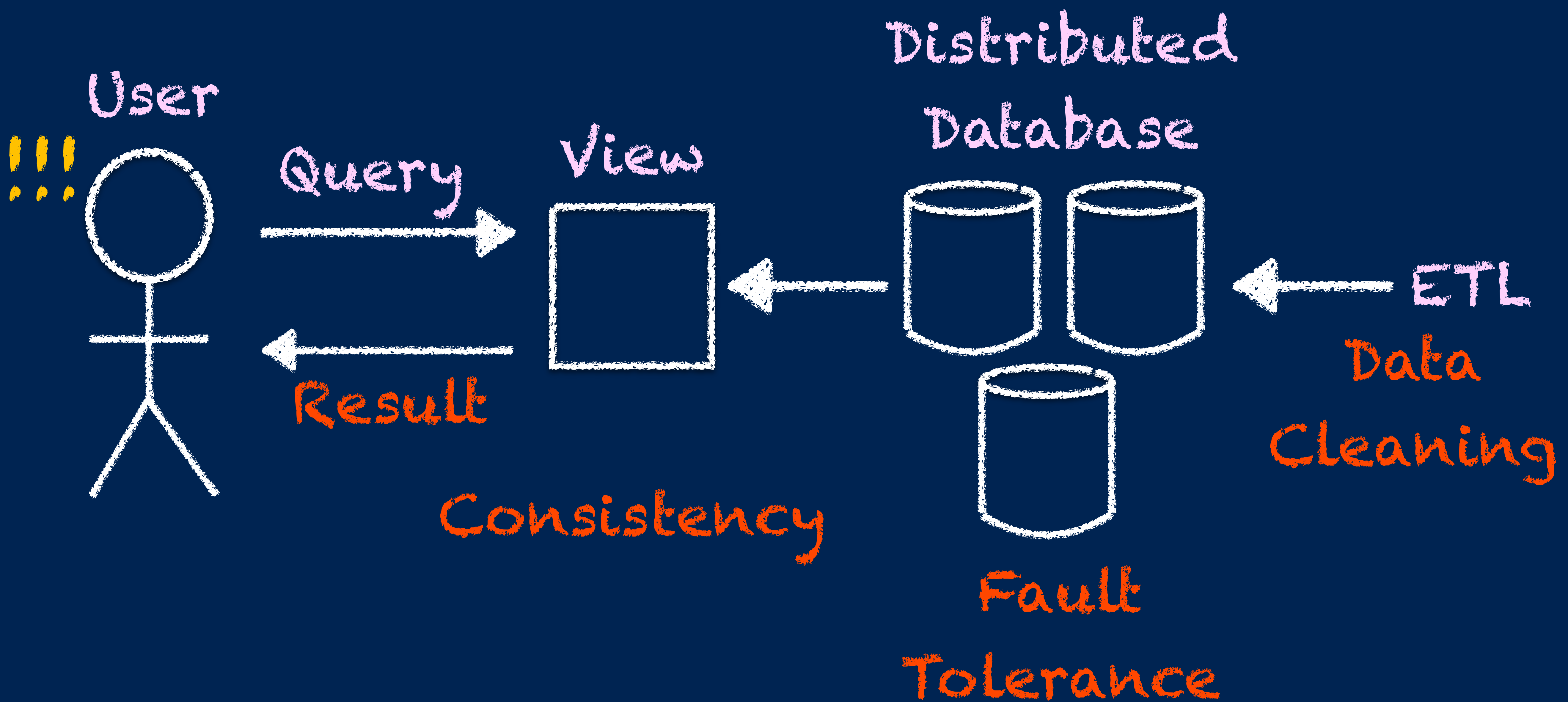
Database-User Interaction Model



Database Research Is Largely About Error



Database Research Is Largely About Error

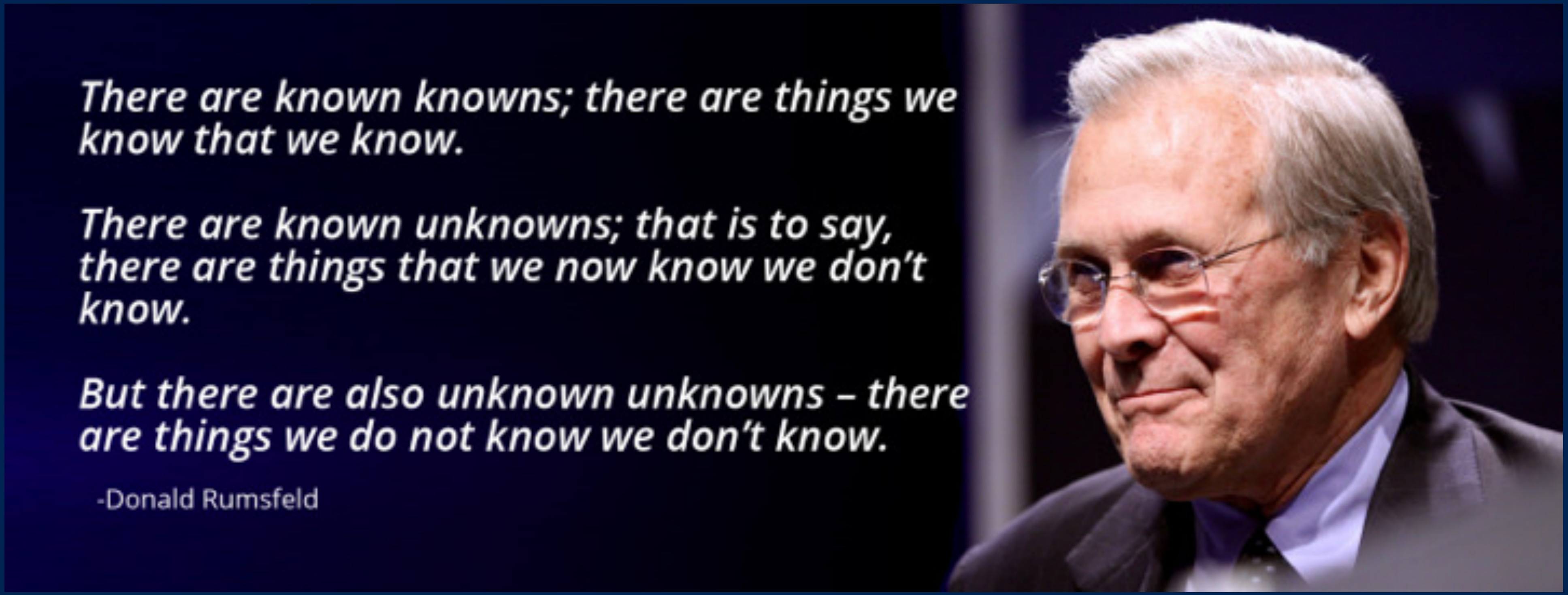


Why is error troubling?

- We use approximations all the time
- Recipe for a VLDB/SIGMOD paper
  - Step 1. Prove your problem is NP-Hard
  - Step 2. Apply Heuristic
- Something deeper going on?

Good Errors

- Approximations in real life
  - Opinion Polls, Clinical Trials
- Why do we trust these?



Approximation vs. Data Error

The Sample-and-Clean Problem [SIGMOD 2014]

Focus on Aggregate Queries (e.g. sum, count, avg)

Intuitive Example: Average Age of Survey Participants

- Large datasets are time consuming to fix

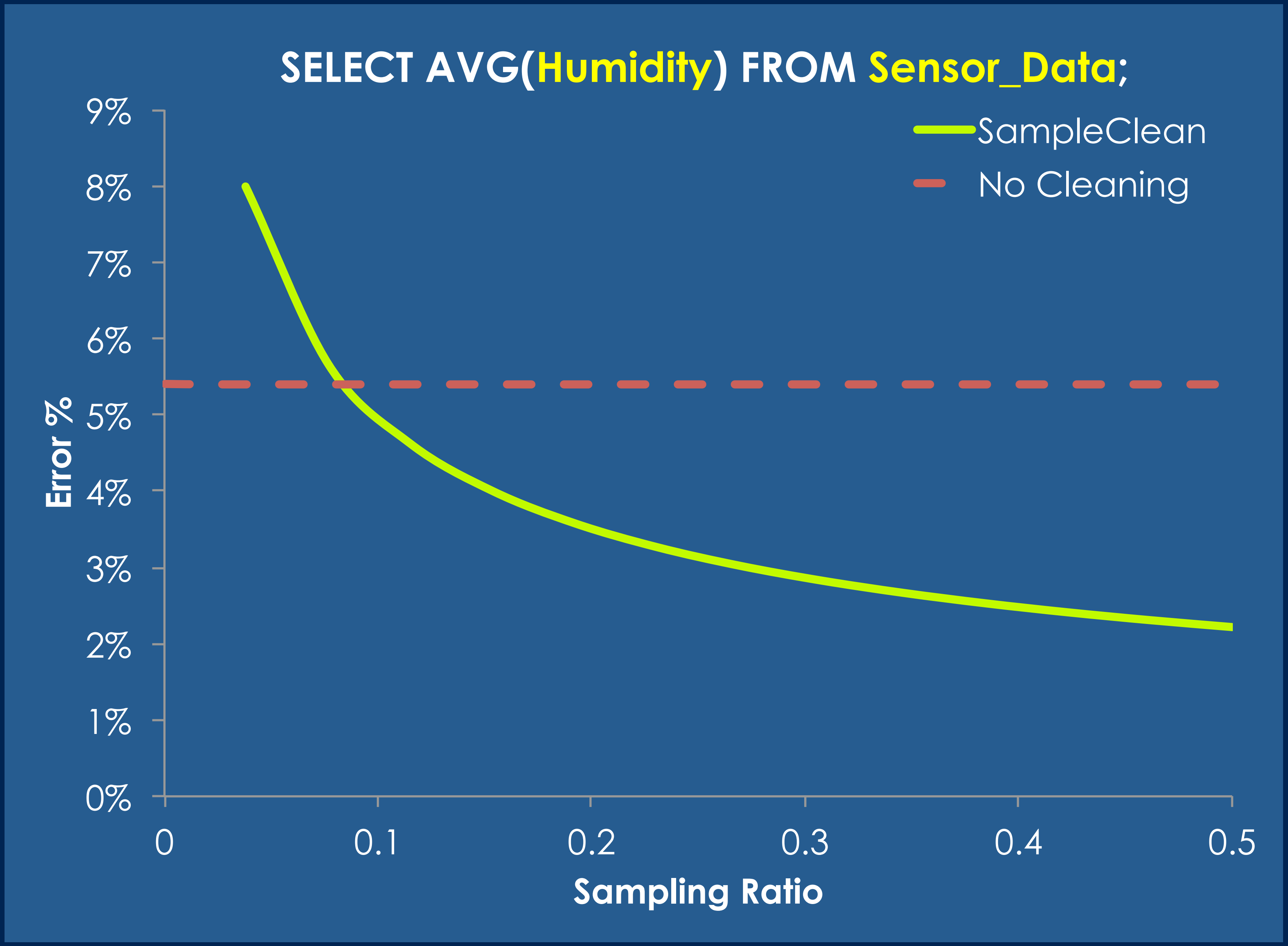
The SampleClean approach:

1. Take a uniform sample of records
2. Fix errors in the sample
3. **(AQP)** Average age in the sample.
4. **(CORR)** Average change in age after cleaning.

	A	B	C	D
1	**Date**	**Participant ID Number**	**Age**	**What parish do you live in?**
2	18-06-14	249	28	Naluwoli
3	17-06-14	2977	20	
4	17/06/2014	03500	52	Butansi
5	19/06/2014	4194	32	Naluwoli
6	17/06/2014	07420	19 1/2	Butansi
7	17/06/2014	07428	21	Naluwoli
8	17/06/2014	10011	Twenty	Butansi
9	17/06/2014	10061	30	Butansi
10	13-06-14	10431	27	Butansi
11	18/06/2014	10685	27 years	Butansi
12	19/06/2014	10920	19 years	Naluwoli
13	19/06/2014	10982	25	Naluwoli
14	13-06-14	11164	22	Naluwoli
15	17/06/2014	12138	Twenty-Two	Naluwoli

[HumTech 15]

Aggregate Queries Do Not Need Fully "Clean" Data



Outline

- Not all errors are created equal
- Approximating Materialized Views
- Results

Materialized Views

- Stored, pre-computed query result

```
SELECT avg(salary)
FROM employees, payroll
WHERE payroll.id = employee.id
```

```
SELECT sum(salary)
FROM employees, payroll
WHERE payroll.id = employee.id
AND employee.location = 2
```

```
SELECT avg(salary)
FROM employees, payroll
WHERE payroll.id = employee.id
GROUP BY employee.location
```

```
CREATE VIEW employee_payroll
AS SELECT *
FROM employees, payroll
WHERE payroll.id = employee.id
```

Materialized Views

- Re-write queries on the view

```
CREATE VIEW employee_payroll
AS SELECT *
FROM employees, payroll
WHERE payroll.id = employee.id
```

```
SELECT avg(salary)
FROM employee_payroll
```

```
SELECT sum(salary)
FROM employee_payroll
WHERE employee.location = 2
```

```
SELECT avg(salary)
FROM employee_payroll
GROUP BY employee.location
```

Materialized View Staleness

- If the database is updated the materialized view is stale.

id	Name	Salary	Location
1	John	40k	1
2	Al	66k	2
3	Sally	100k	2
4	Sue	48k	1

```
CREATE VIEW employee_payroll
AS SELECT *
FROM employees, payroll
WHERE payroll.id = employee.id
```

Incremental Maintenance

- Many algorithms have been proposed to keep MVs up-to-date.
- Avoids recomputation but still expensive for every incoming update.
- DBToaster [Koch et al. 2014]

Materialized View Staleness

- Batched Maintenance

id	Name	Salary	Location
1	John	40k	1
2	Al	66k	2
3	Sally	100k	2
4	**Sue**	**48k**	**1**

```
CREATE VIEW employee_payroll
AS SELECT *
FROM employees, payroll
WHERE payroll.id = employee.id
```

Maintenance Workflow

- Stale View: S
- Up-to-date View: S’
- Maintenance Strategy M

$$S \longrightarrow M(S, Updates, Base\ Data) \longrightarrow S'$$

Every Night

SampleClean Philosophy

- To answer aggregate queries we don't need the full view
- Stale View: S_{sample}
- Up-to-date View: S'_{sample}
- Cleaning Strategy C

$$S_{sample} \longrightarrow C(S_{sample}, \text{Updates, Base Data}) \longrightarrow S'_{sample}$$

Frequency Determined By Sampling Ratio

Deriving The Cleaning Strategy C

- Bad Alternative 1: Maintain then sample (correct but slow)
- Bad Alternative 2: Sample then maintain (incorrect but fast)

```
CREATE VIEW employee_payroll
AS SELECT sum(salary), count(*)
FROM employees, payroll
WHERE payroll.id = employee.id
GROUP BY employee.location
```

Problem of Provenance

- Sampling does not commute with all operations.
- If a row is sampled from a derived relation, we must ensure that all contributing rows are also sampled.
- We can do this by enforcing unique keys.

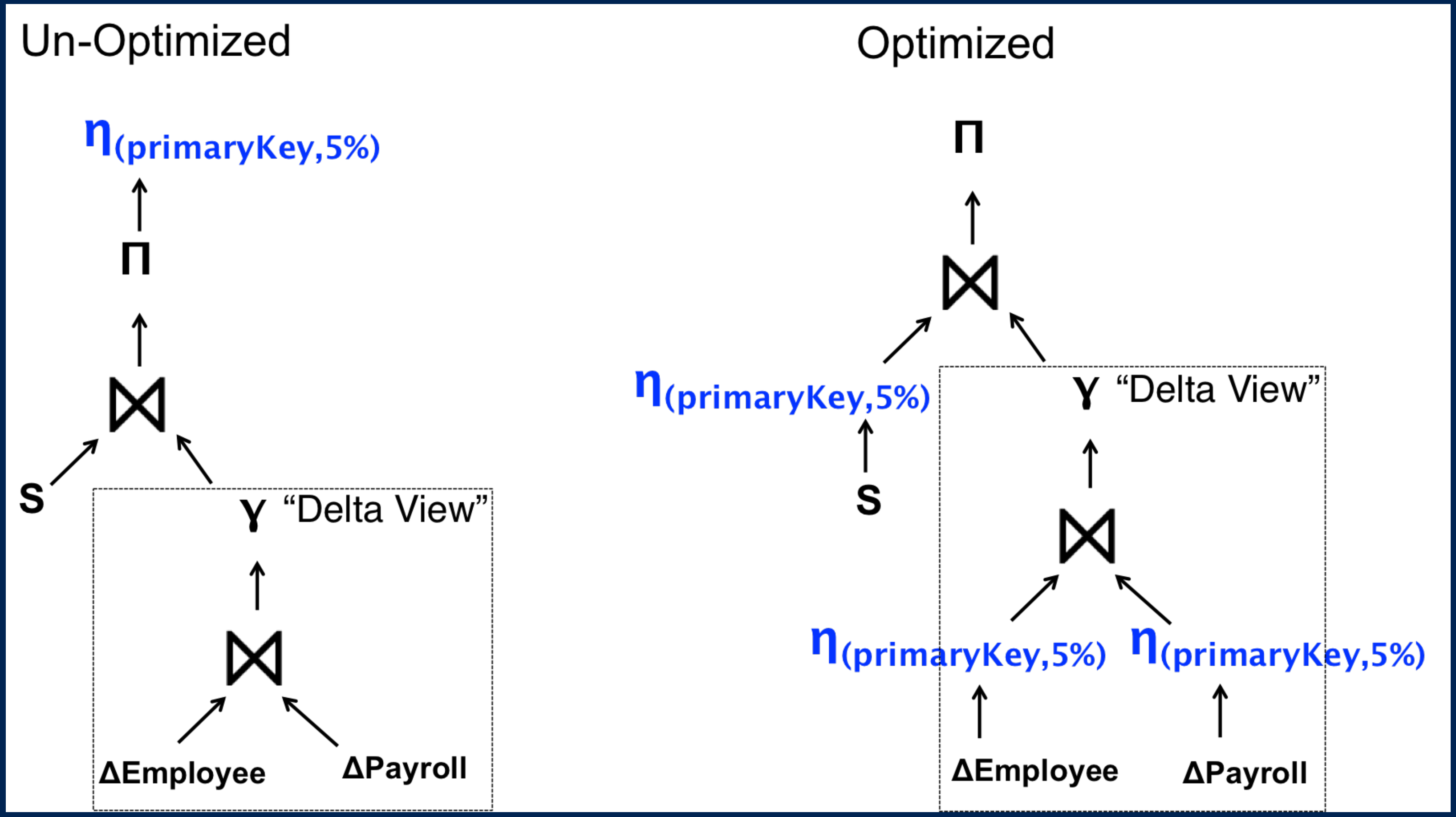
DEFINITION 2 (PRIMARY KEY GENERATION). *For every relational expression R, we define the primary key attribute(s) of every expression to be:*

- *Base Case: All relations (leaves) must have an attribute p which is designated as a primary key. That uniquely identifies rows.*
- $\sigma_\phi(R)$: *Primary key of the result is the primary key of R*
- $\Pi_{(a_1,\ldots,a_k)}(R)$: *Primary key of the result is the primary key of R. The primary key must always be included in the projection.*
- $\bowtie_{\phi(r1,r2)}(R_1, R_2)$: *The primary key of the result is the tuple of the primary keys of R_1 and R_2.*
- $\gamma_{f,A}(R)$: *The primary key of the result is the group by key A (which may be a set of attributes).*
- $R_1 \cup R_2$: *Primary key of the result is the union of the primary keys of R_1 and R_2*
- $R_1 \cap R_2$: *Primary key of the result is the intersection of the primary keys of R_1 and R_2*
- $R_1 - R_2$: *Primary key of the result is the primary key of R_1*

For every node at the expression tree, these keys are guaranteed to uniquely identify a row.

Deterministic Sampling

- Use a hash mod operation to ensure that all rows with a given primary key are sampled.
- Optimization posed as sampling push down.



Estimating A Query Result

- (AQP) Given a clean sample of data how to estimate an aggregate query result.

$$q(S') \approx k(m) * q(S'_{sample})$$

- (CORR) Estimate a correction to existing results

$$q(S) - q(S') \approx k(m) * (q(S_{sample}) - q(S'_{sample}))$$

CORR works better when the error is small
AQP is more robust

Outline

- Not all errors are created equal
- Approximating Materialized Views
- Results

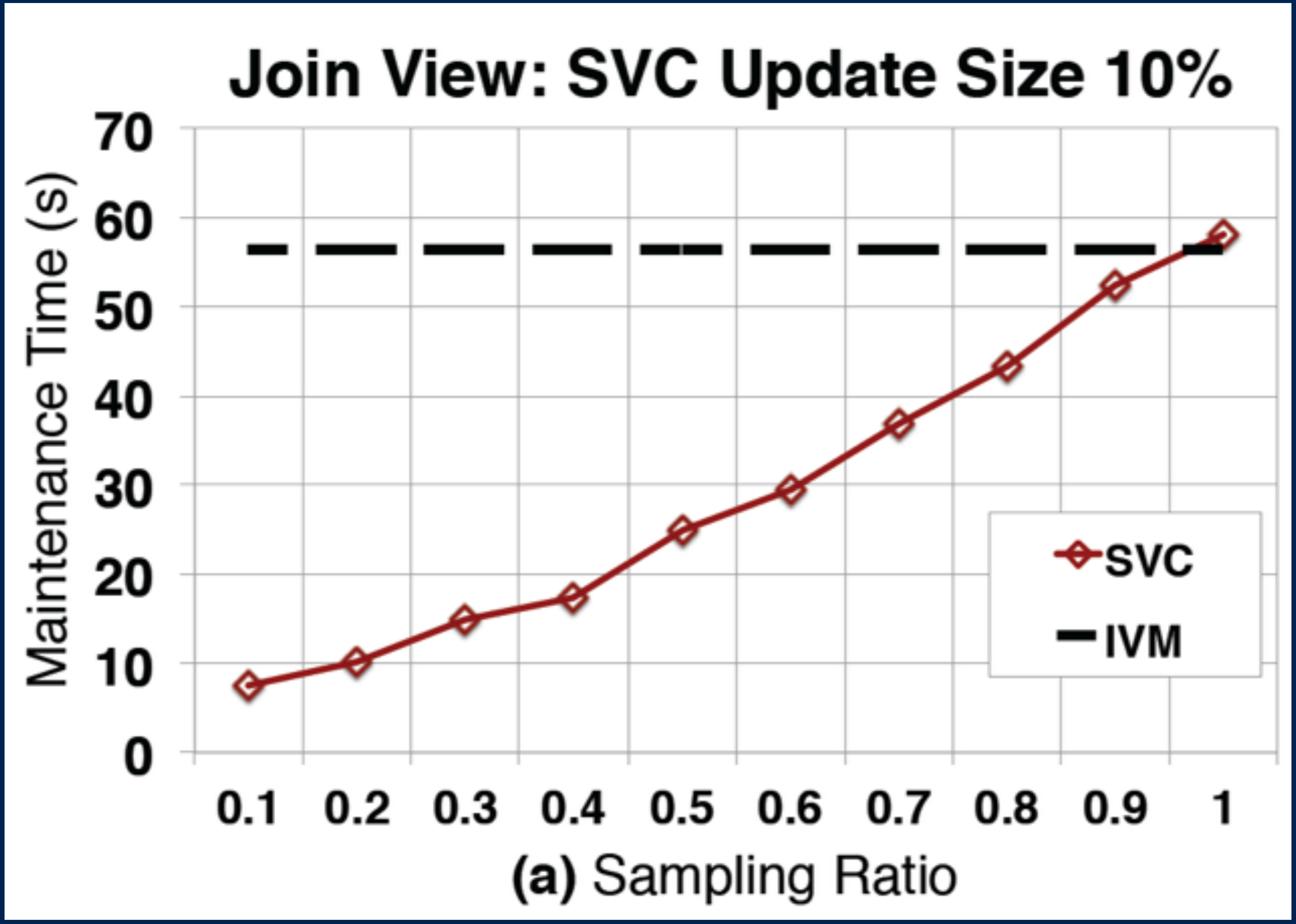
Experimental Setup

- TPCD on MySQL (Join Views, Complex Nested Views)
  - 10 GB
  - Only modification to MySQL was the hashing primitive

- Conviva Inc. on Spark (Aggregate Views)
  - 1TB
  - Apache Spark 1.1 Catalyst for push down
  - RDDs are immutable

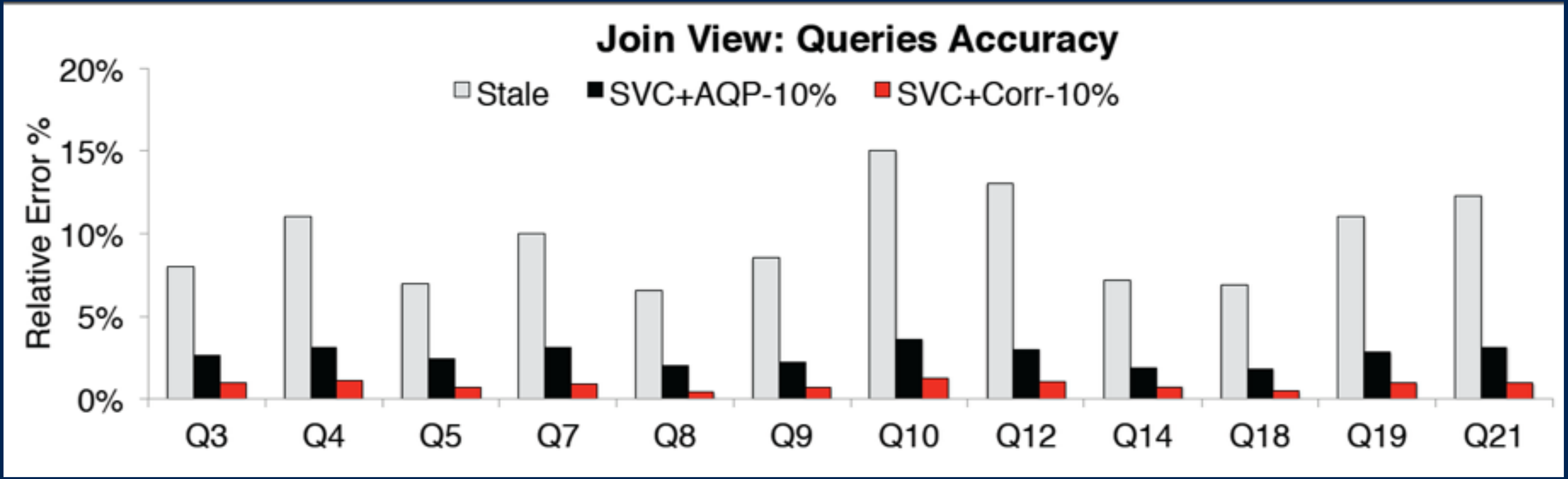
TPCD

TPCD: LINEITEM ORDER JOIN



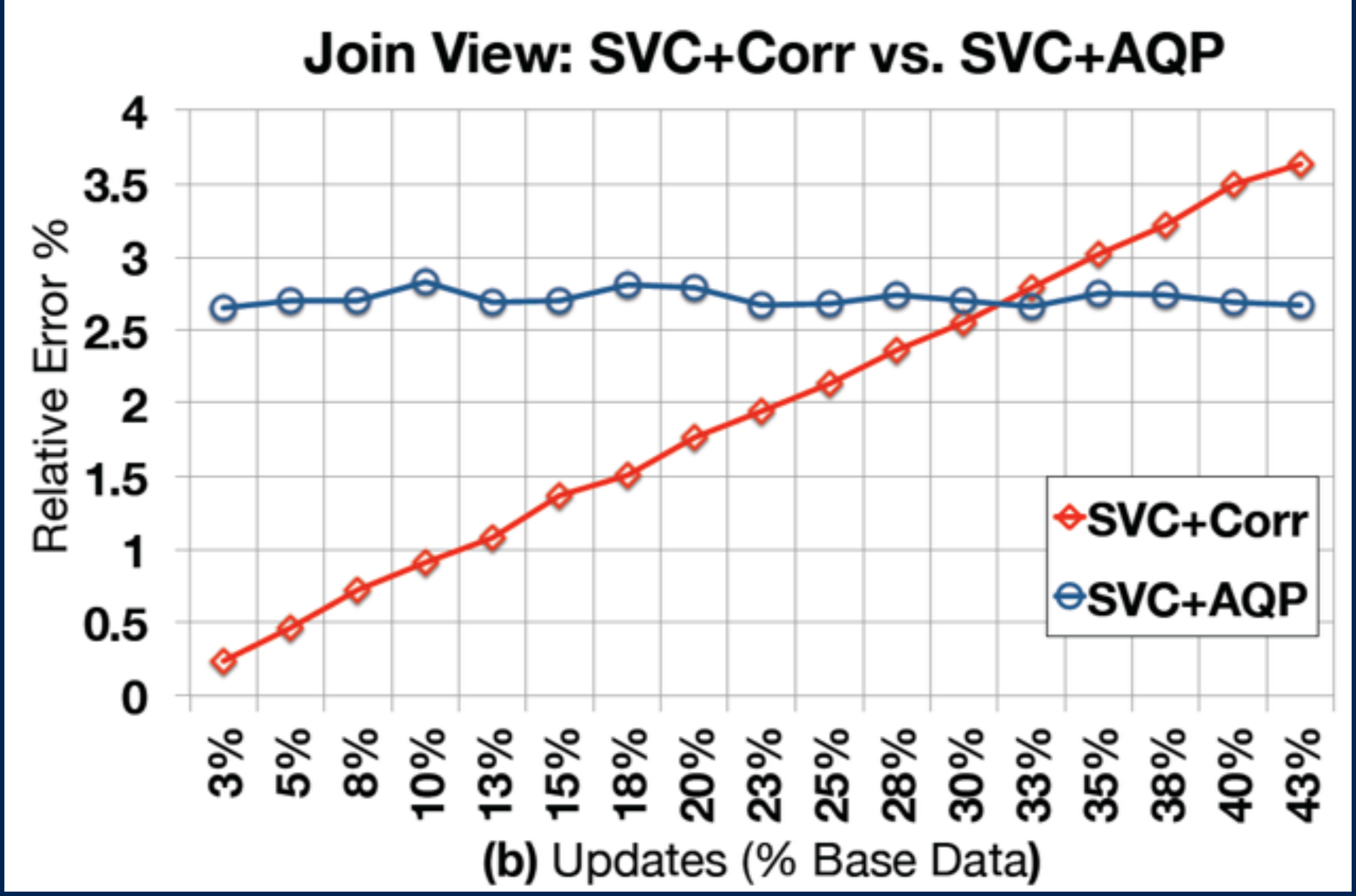
TPCD

TPCD: LINEITEM ORDER JOIN ACCURACY



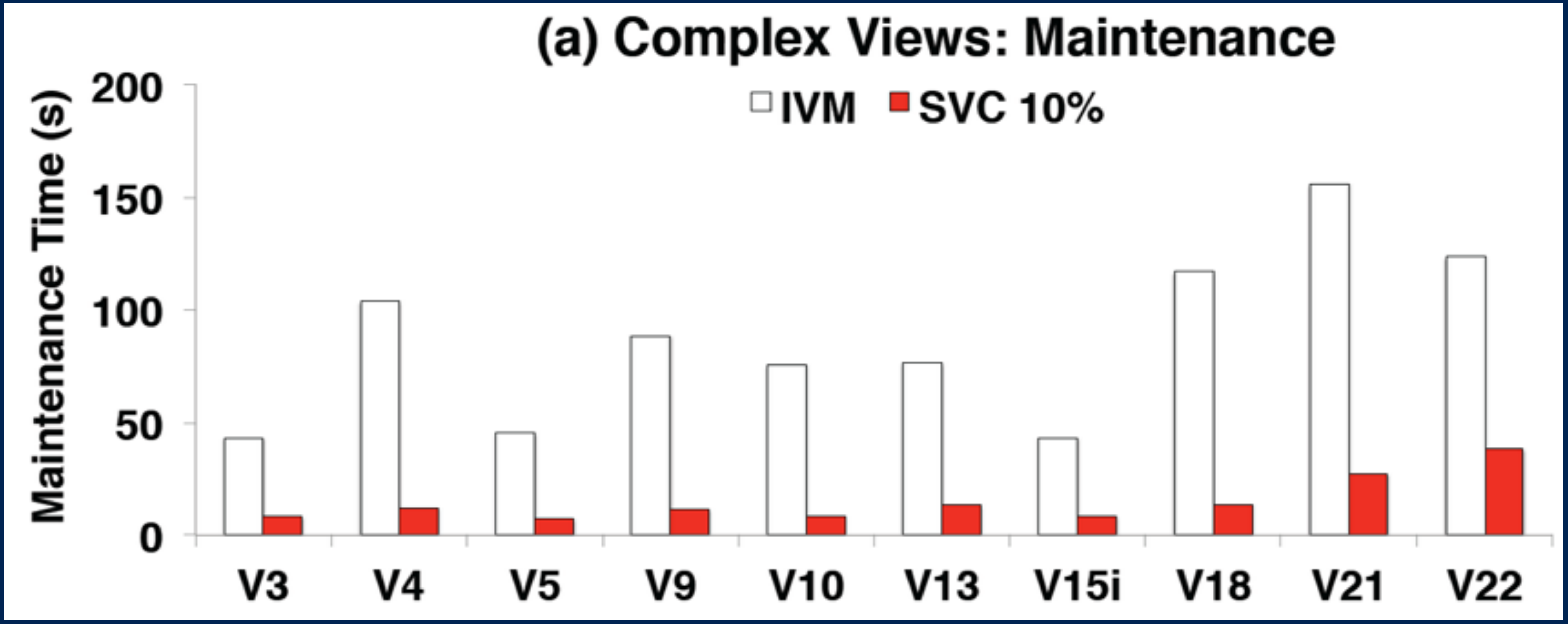
TPCD

TPCD: LINEITEM ORDER JOIN ACCURACY

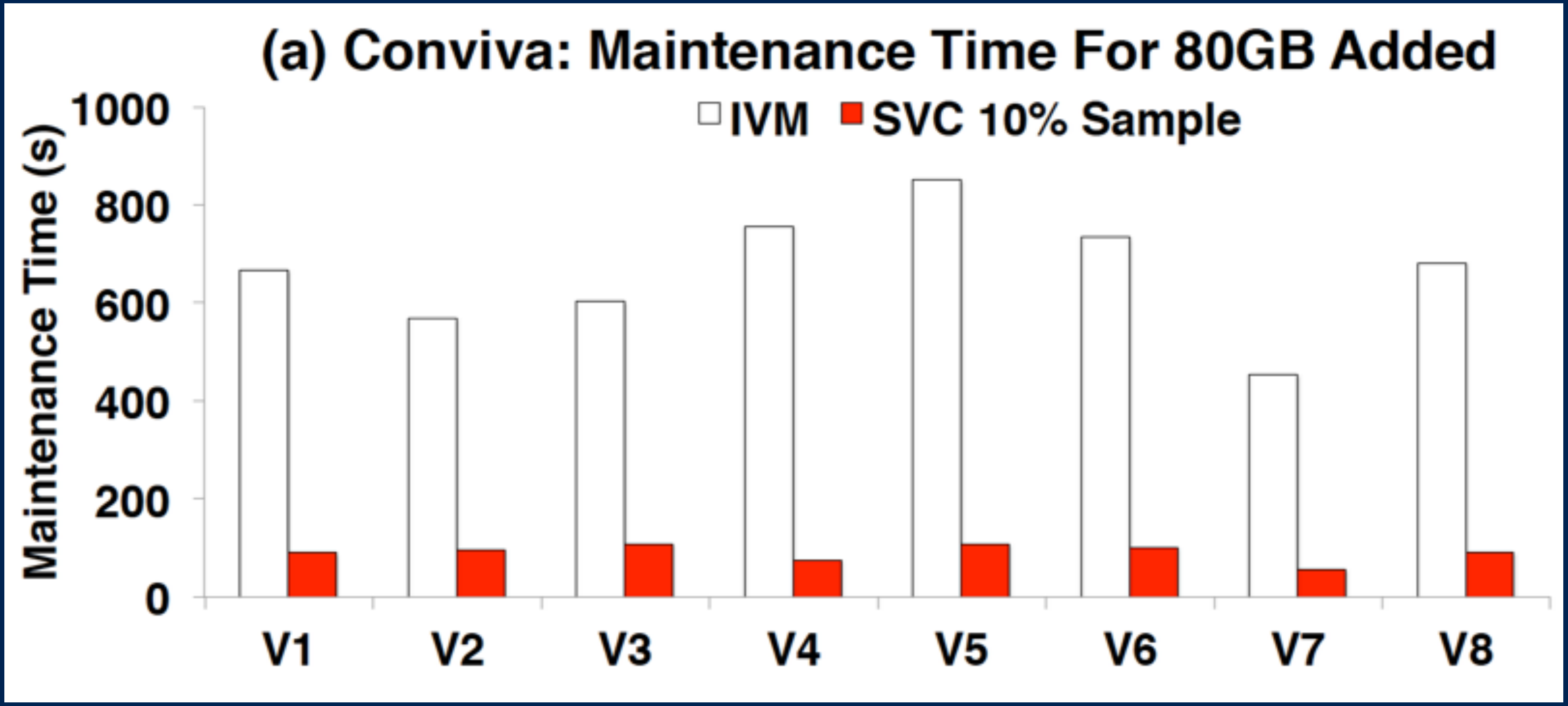


TPCD

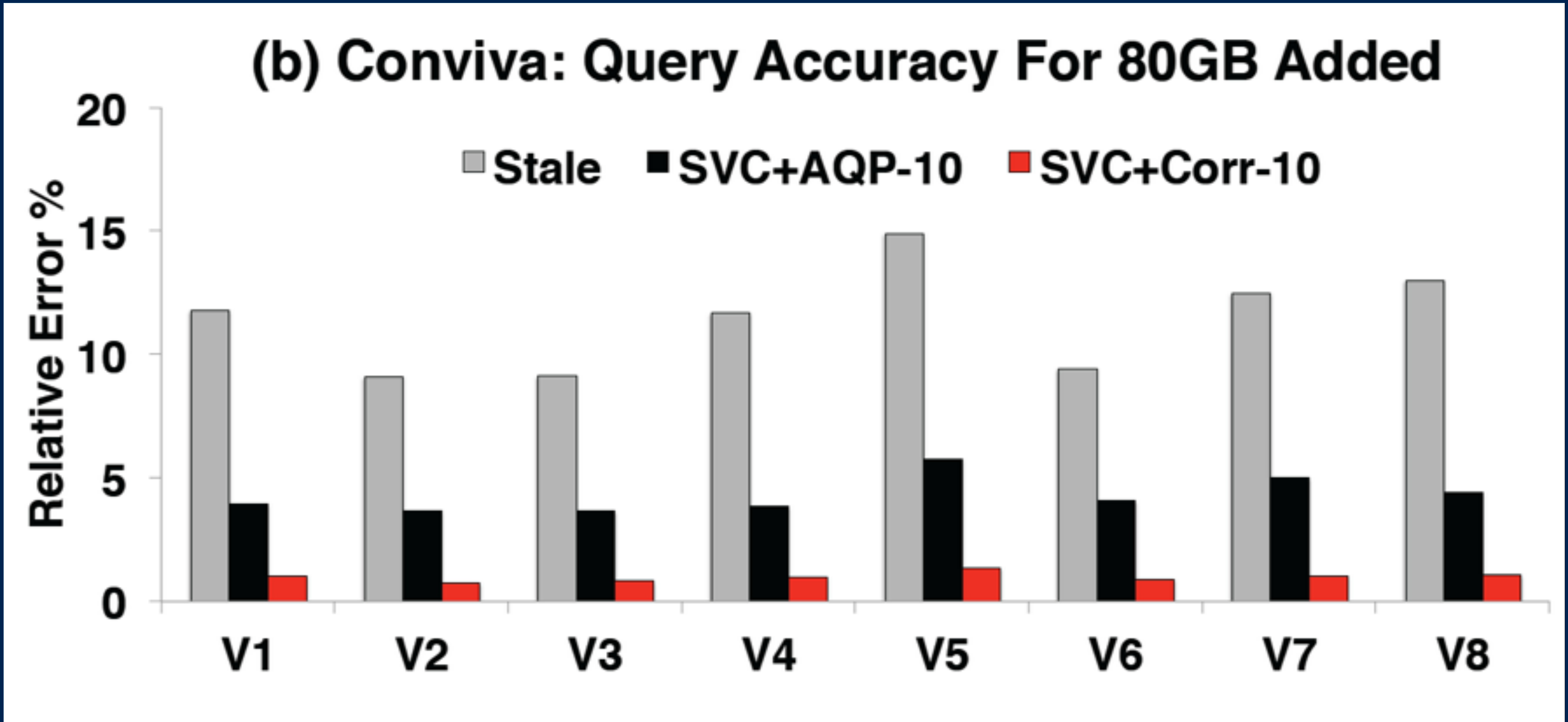
TPCD: Queries as Views



Conviva


(a) Conviva: Maintenance Time For 80GB Added
IVM
SVC 10% Sample
Maintenance Time (s)
1000
800
600
400
200
0
V1
V2
V3
V4
V5
V6
V7
V8

Conviva


(b) Conviva: Query Accuracy For 80GB Added
Relative Error %
20
15
10
5
0
Stale
SVC+AQP-10
SVC+Corr-10
V1
V2
V3
V4
V5
V6
V7
V8