Improving OCC by Transaction Batching and Reordering

Bailu Ding[1], Lucja Kot[2], Johannes Gehrke[3]

[1]Microsoft Research, [2]Gramma Tech, Inc, [3]Microsoft Corporation

Microsoft

DMX Group Overview

- Data management, exploration and mining
- Flexible resource allocation mechanisms and policies for cloud database
  - Contact: Vivek Narassaya
- Self-service data preparation
  - Contact: Yeye He
- Approximate query processing
  - Contact: Surajit
- Actor-Oriented Databases (Orleans, link)
  - Contact: Phil Bernstein
- Automated physical design in the cloud
  - Contact: Sudipto Das, Bailu Ding
- And many more!

Automated Physical Design in the Cloud

- A continuous indexing framework to automatically select and build indexes to reduce query execution time
  - *Closed-loop solution:* success measured in terms real execution time instead of query optimizer costs
  - *A hands-free solution: remove human intervention from the critical path of the loop*
- More than index recommendation: workload extraction, index implementation, validation, and monitoring
  - *Automatically Indexing Millions of Databases in Microsoft Azure SQL Database*, Sudipto Das, Miroslav Grbic, Igor Ilic, and Et al., SIGMOD 2019
- Improve plan quality with reduced execution cost at low risk with multiple executed plans of the same query
  - *Plan Stitch: Harnessing the Best of Many Plans*, Bailu Ding, Sudipto Das, Wentao Wu, and Et al., VLDB 2018

Improving OCC Through Transaction Batching and Operation Reordering

Optimistic Concurrency Control

- Read phase
- Validation phase
- Write phase

Optimistic Concurrency Control

- Read phase
- Validation phase
- Write phase

T1

Read (X) -> 10

Read (Y) -> 20

Write (X) -> 30

T2

Read (X) -> 10

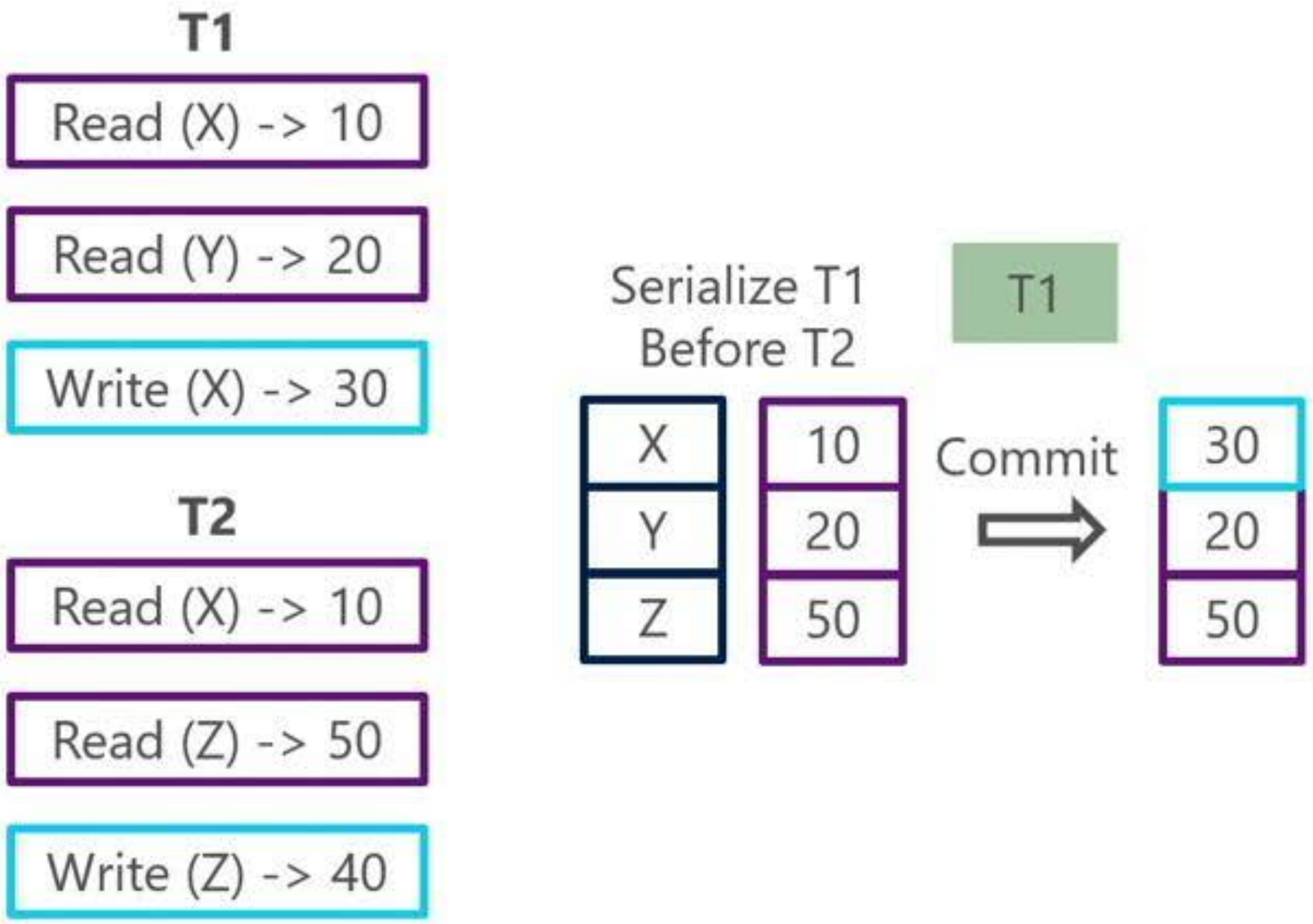
Read (Z) -> 50

Write (Z) -> 40

Read Phase

Optimistic Concurrency Control

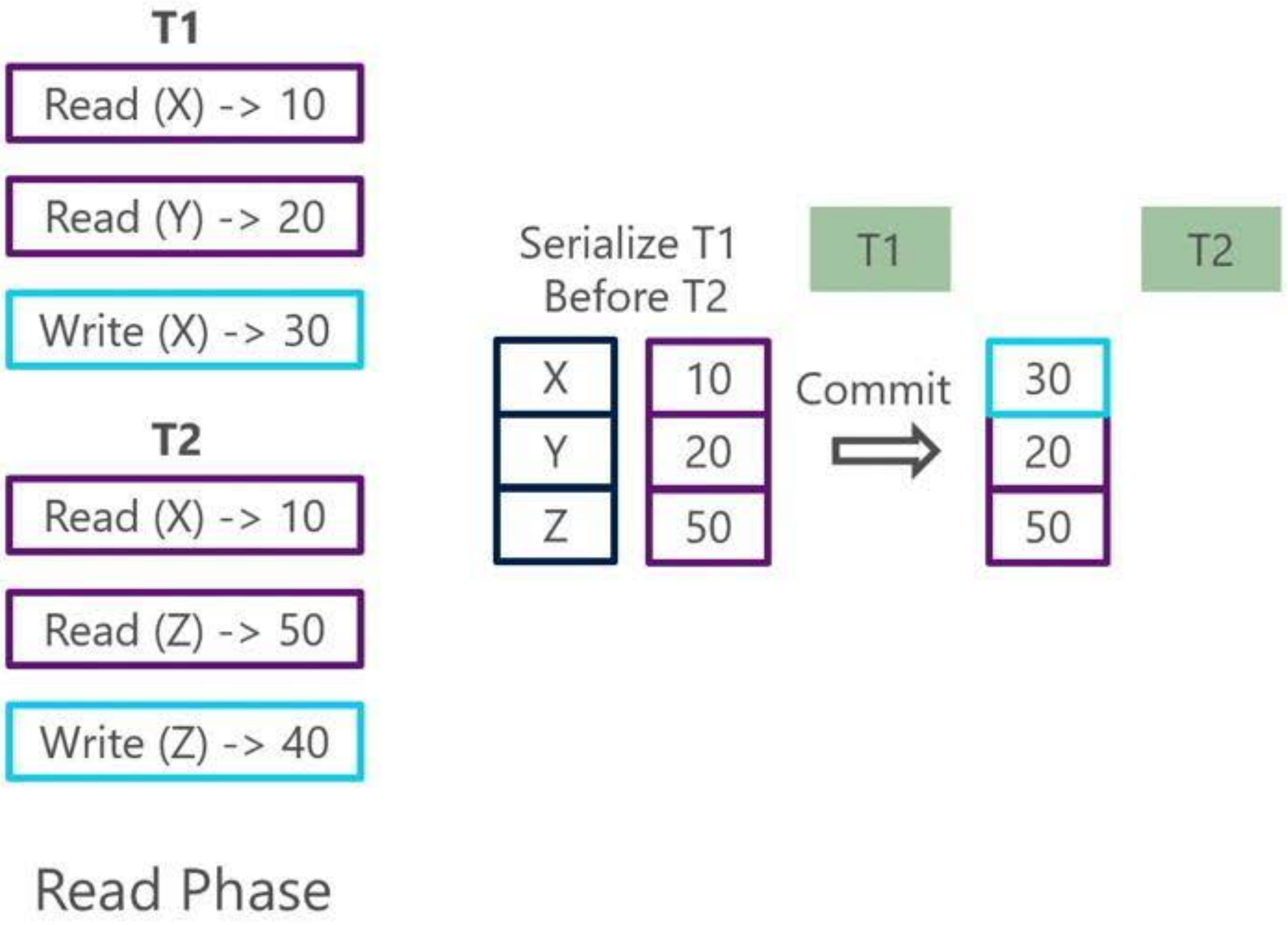
- Read phase
- Validation phase
- Write phase



Read Phase

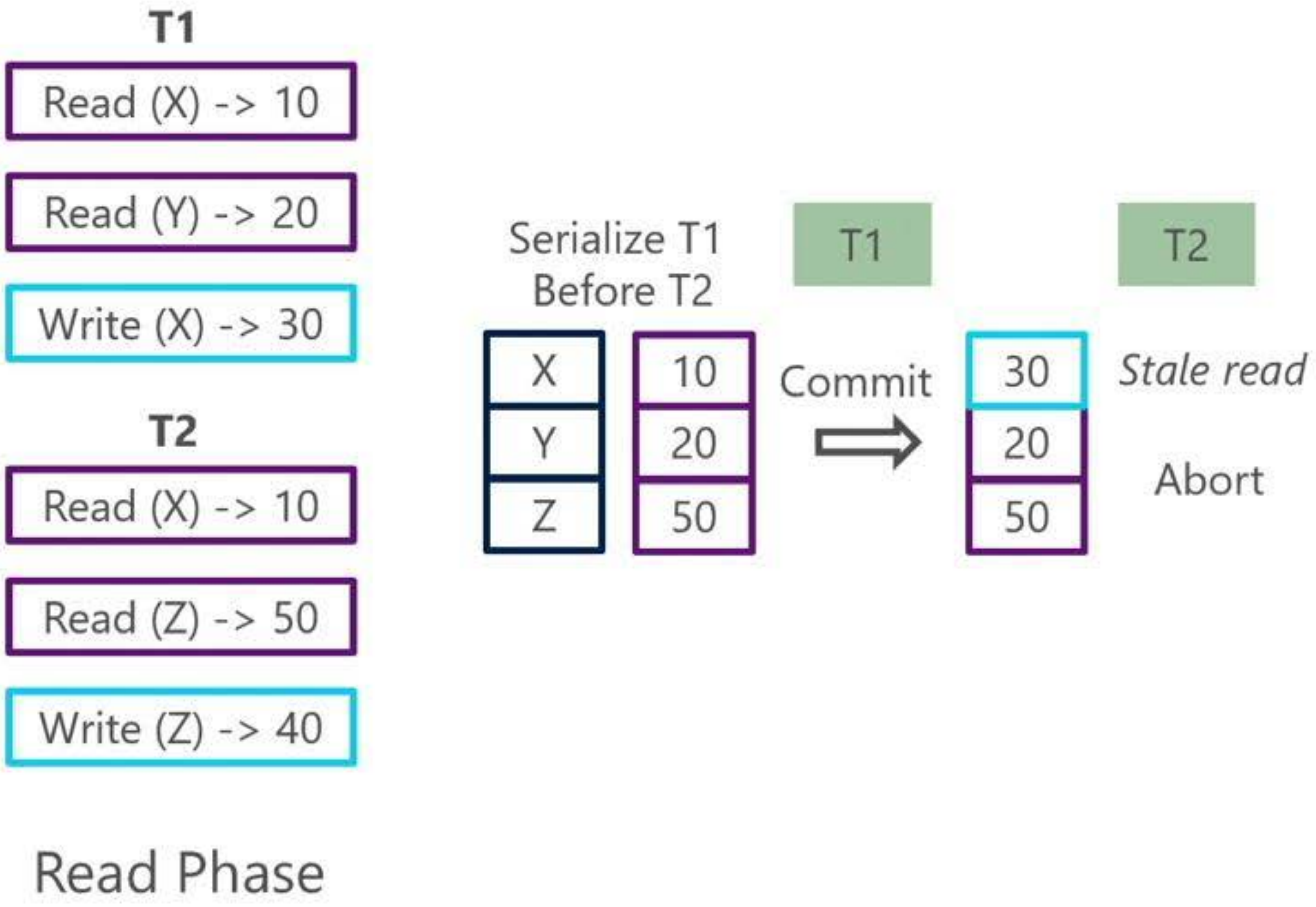
Optimistic Concurrency Control

- Read phase
- Validation phase
- Write phase

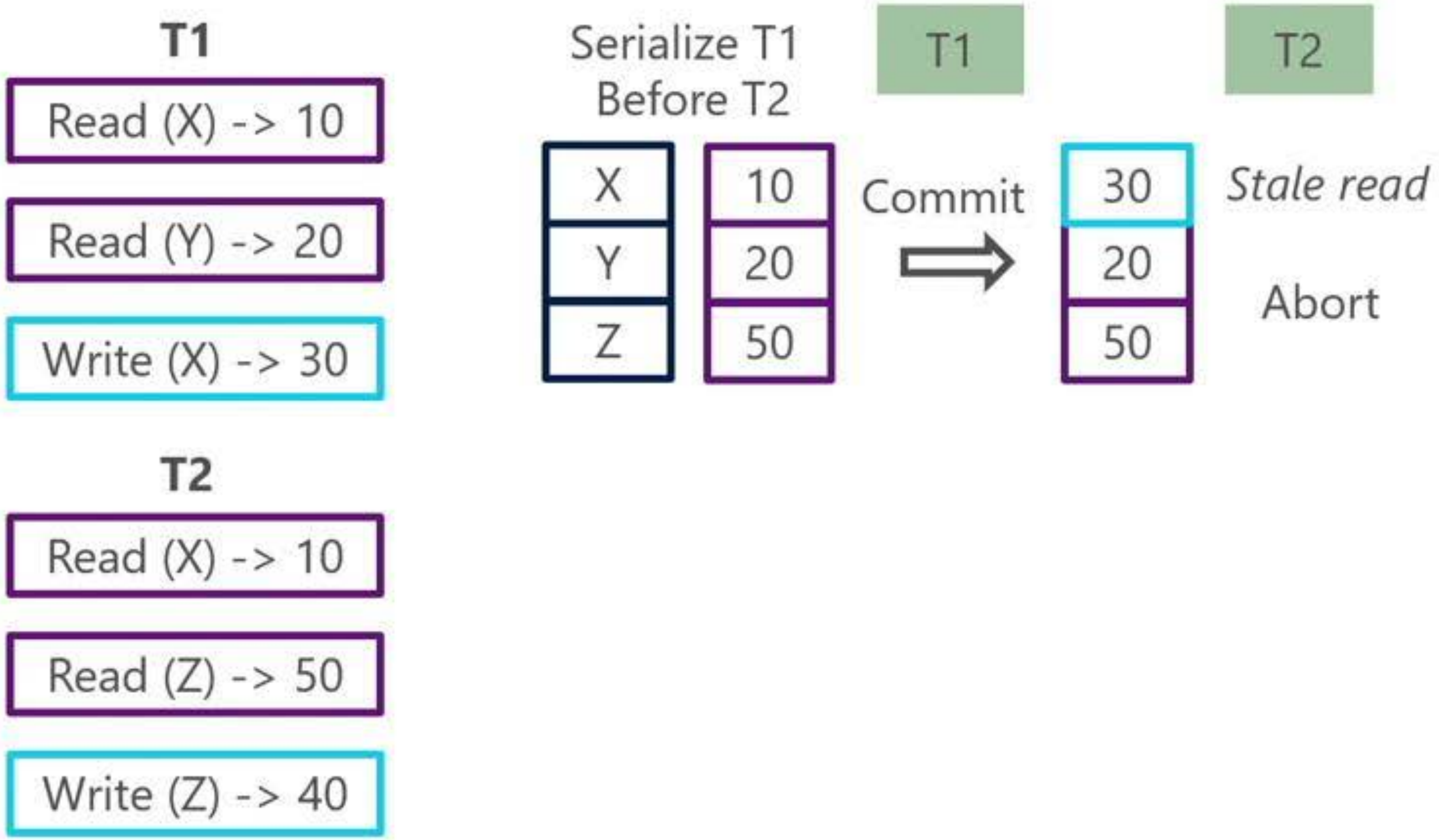


Optimistic Concurrency Control

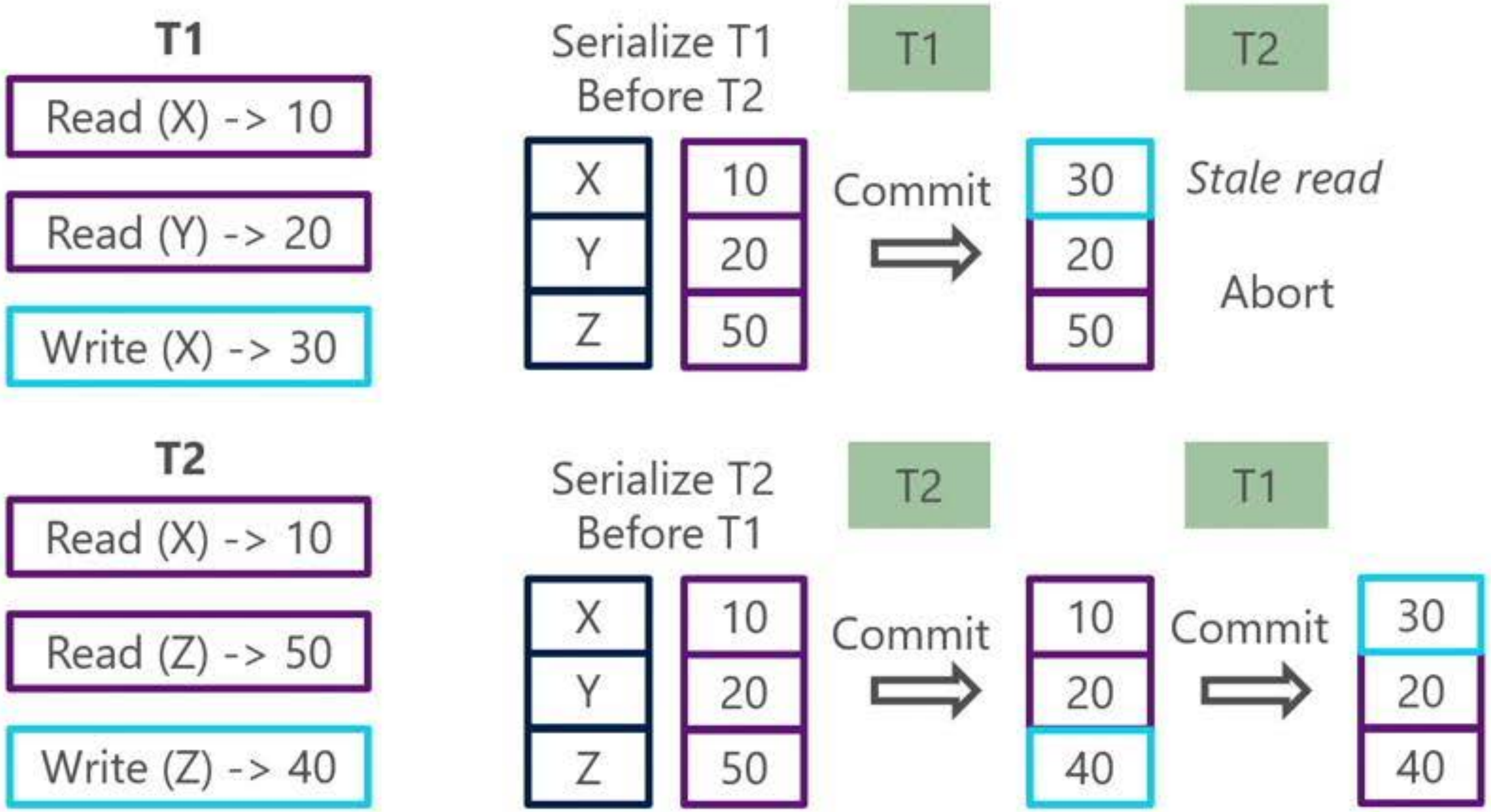
- Read phase
- Validation phase
- Write phase



Is T2 Destined to Abort, Really?

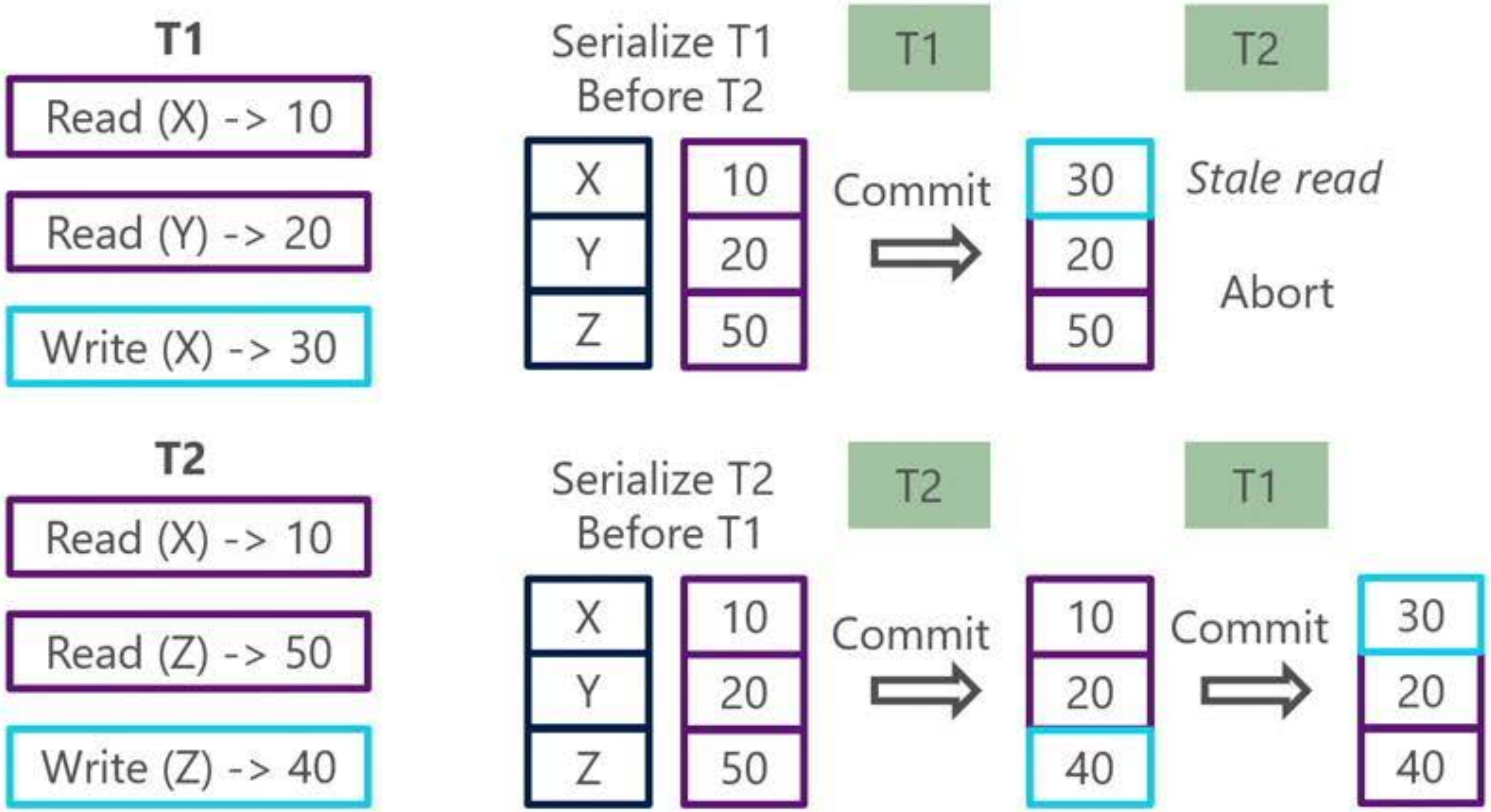


Is T2 Destined to Abort, Really?



Is T2 Destined to Abort, Really?

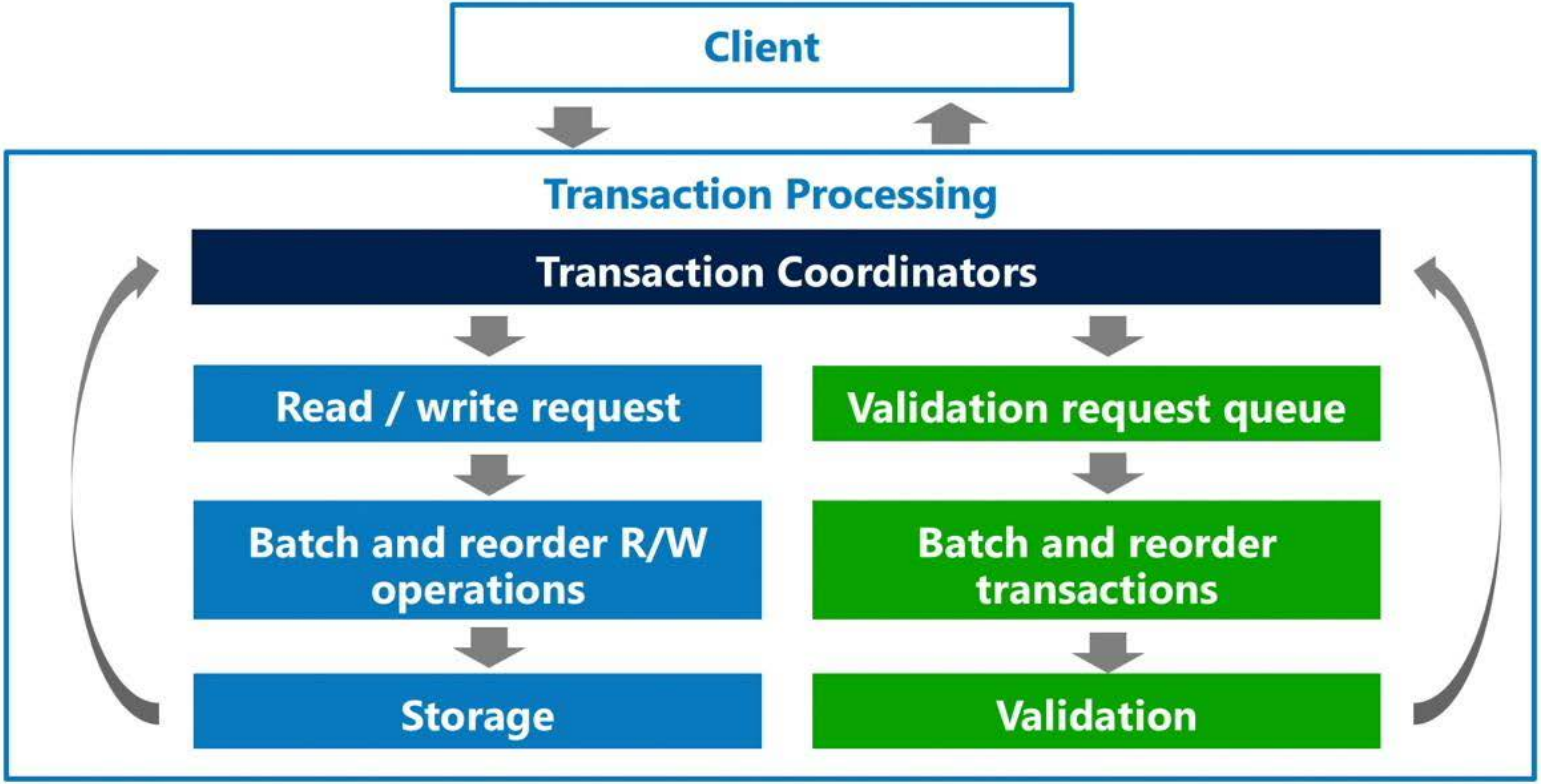
- Conflicting concurrent transactions can potentially all commit with an alternative serialization order



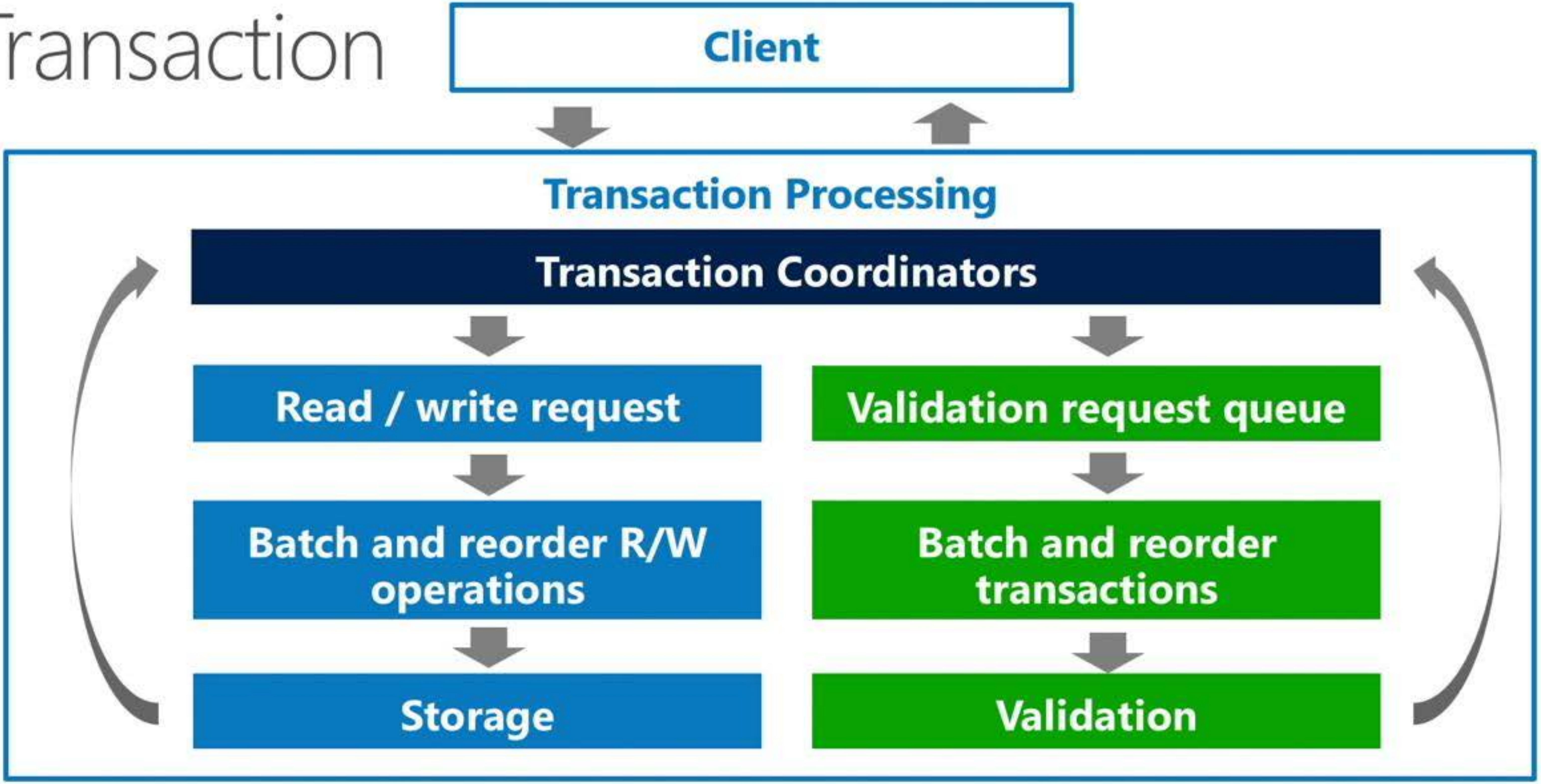
Transaction Reordering in OCC

- Optimistic concurrency control finalizes the serialization order *after* transaction execution
- Incorporate reordering throughout the life of a transaction
- Batch transactions explicitly to open doors for reordering
  - Limit the scope of reordering with a batch

A Life of a Transaction



A (New) Life of a (Batched and Reordered) Transaction



Transaction and Operation Reordering

- Reordering transactions at clients
  - Prior work on static transaction scheduling
- Reordering operations at storage
  - Optimal strategy: Prioritize writes before reads to avoid stale reads
- Reordering transactions at validation
  - How to create a serialization order with the least number of aborts from a batch of transactions?

Transaction Reordering at Validation

- Given a batch of transactions $\boldsymbol{B}$, construct subset $\boldsymbol{B'} \subseteq \boldsymbol{B}$, such that $\boldsymbol{B'}$ is serializable and $|\boldsymbol{B'}|$ is maximal among all $\boldsymbol{B'} \subseteq \boldsymbol{B}$
  - The number of aborts is minimized within the batch if $|B'|$ is maximal
- How to decide if $\boldsymbol{B'}$ is serializable and how to construct a serialization order?

Constructing a serialization order

- Dependency graph: the conflicts of transactions
  - Example: A transaction T1 cannot be serialized after a transaction T2 if T2 updates an item it reads. We have T2 -> T1.

Constructing the Maximal B'

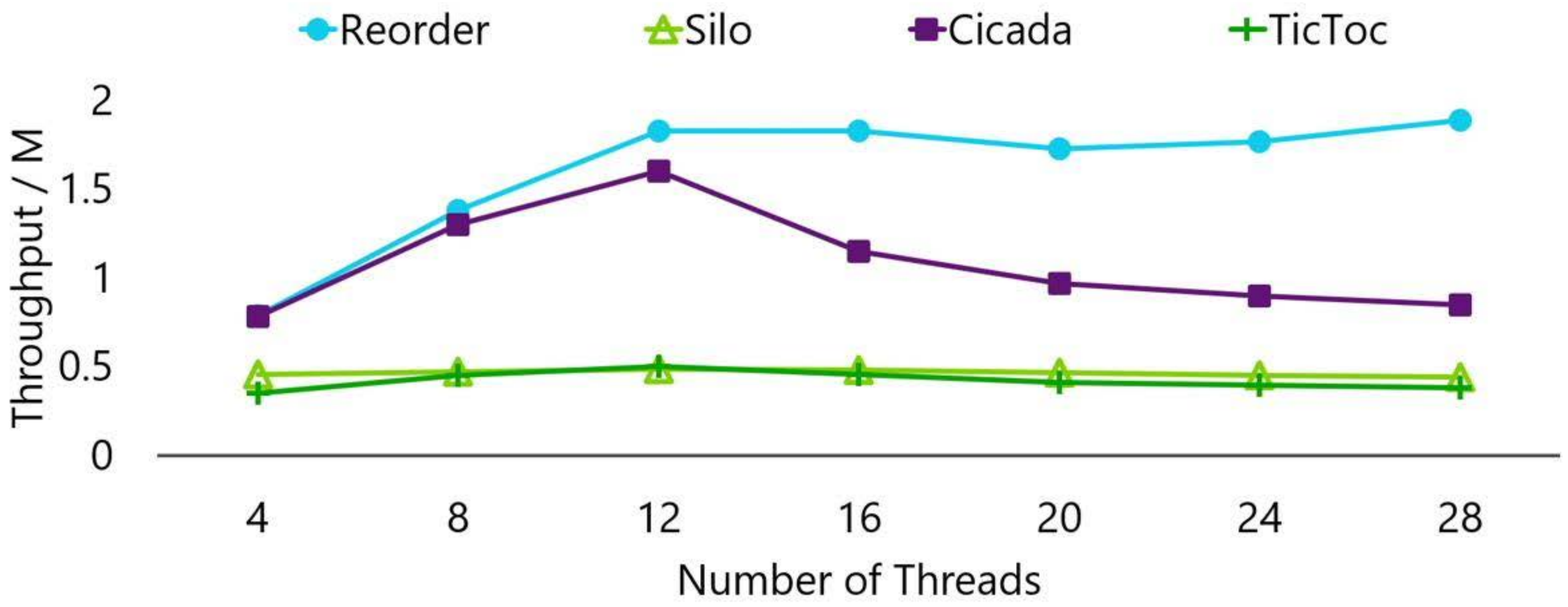
- Given B and a dependency graph $G(B)$, find the maximal $B' \subseteq B$ such that $G(B')$ is acyclic
- Find the minimal $V \subseteq B$ such that $\mathrm{G}(B \setminus V)$ is acyclic
  - $B' = B \setminus V$
- Feedback vertex set!
- But it is NP hard ...
- Greedy algorithms
  - Strongly connected component (SCC) based
  - Sort based: more efficient but less accurate

Policies for Alternative Performance Goals

- Minimize V: minimize the number of aborts
- Alternative goals
  - Minimize tail latency
  - Minimize the number of restarts
  - Maximize monetary value
- Weighted feedback vertex set
  - SCC and sort based greedy algorithms

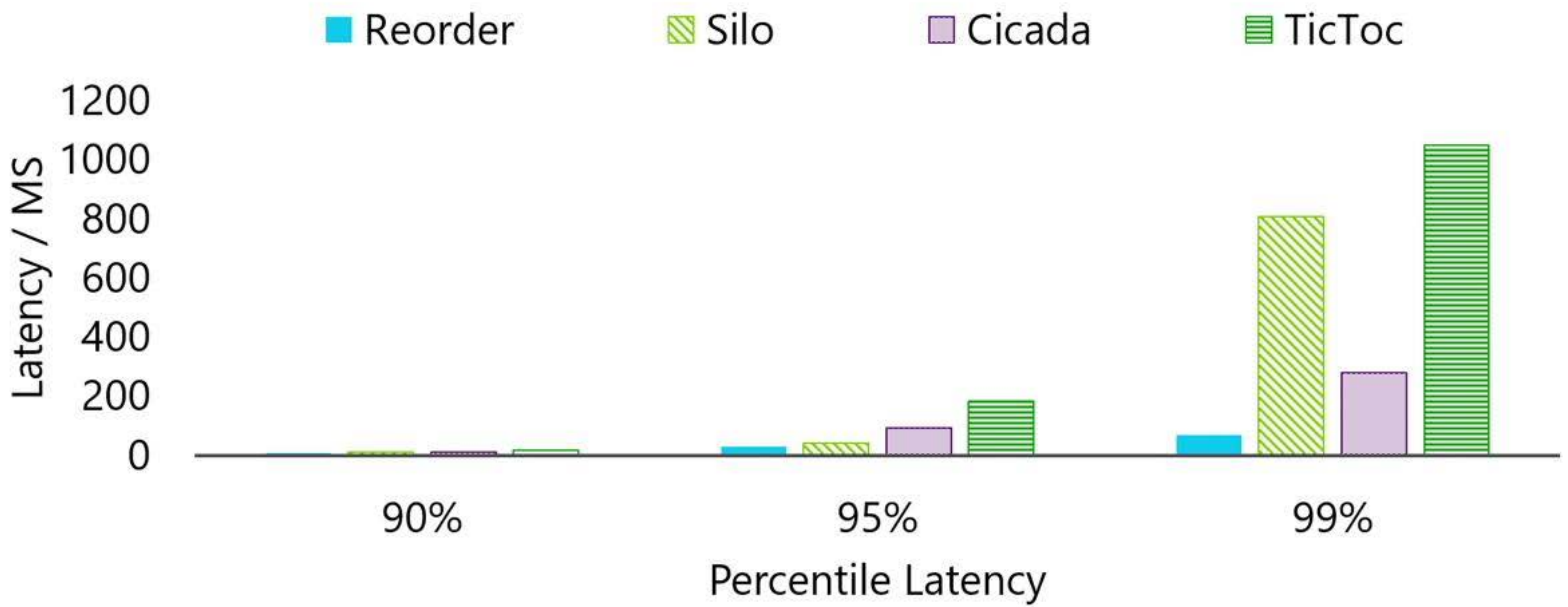
Evaluation: Write-Intensive Skewed YCSB

- Up to 2.2x improvement in throughput



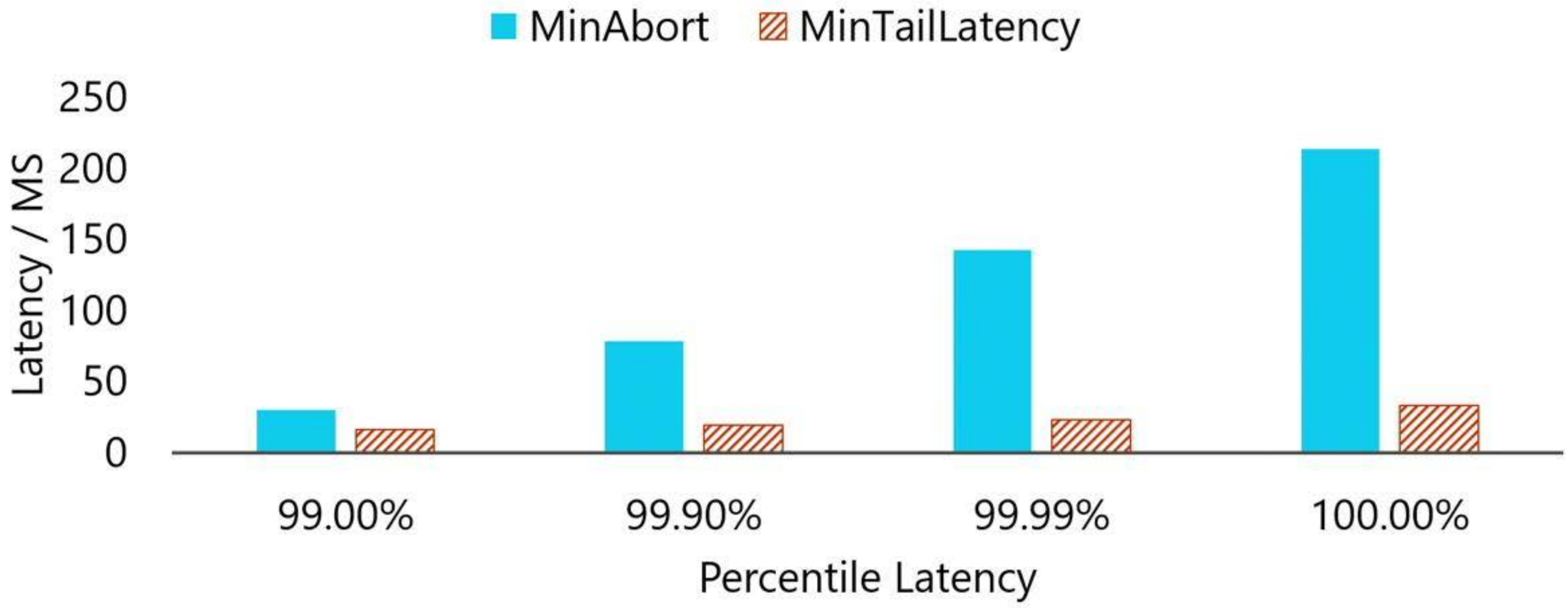
Evaluation: Write-Intensive Skewed YCSB

- Up to 4x reduction in percentile latency



Optimize for Tail Latency

- Up to 6.3x reduction in tail latency



Conclusion

- Explicit batching opens doors for transaction and operation reordering in optimist concurrency control
- Batching and reordering transactions improve throughput and reduce tail latency
- Weighted reordering policies enable optimization for alternative performance goals such as tail latency and monetary value

Towards a Learning Optimizer for Shared Clouds*

Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel,
Wangchao Le, Shi Qiao, Sriram Rao

February 8, 2019

** C. Wu, A. Jindal, S. Amizadeh, H. Patel, W. Le, S. Qiao, and S. Rao. Towards a Learning Optimizer for Shared Clouds. In PVLDB, 12(3): 210–222, 2018.*

Rise of Big Data Systems

Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

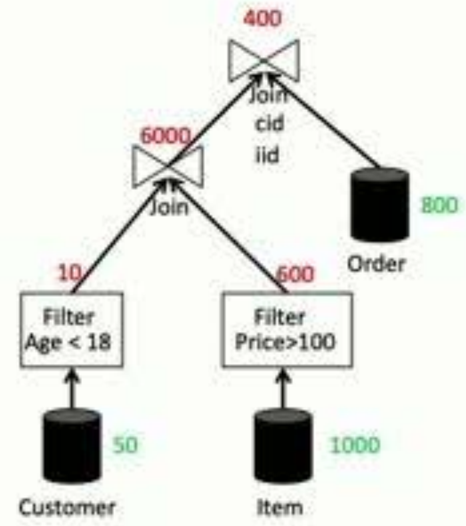
Declarative query interface
Cost-based query optimizer (CBO)

Rise of Big Data Systems

Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

Declarative query interface
Cost-based query optimizer (CBO)

```
SELECT Customer.cname, Item.iname
  FROM Customer
  INNER JOIN Order
  ON Customer.cid == Order.cid
  INNER JOIN Item
  ON Item.iid == Order.iid
  WHERE Item.iprice > 100
  AND Customer.cage < 18;
```

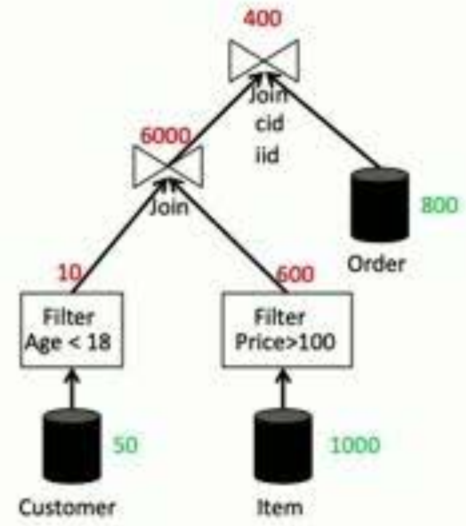


Rise of Big Data Systems

Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

Declarative query interface
Cost-based query optimizer (CBO)

```
SELECT Customer.cname, Item.iname
   FROM Customer
   INNER JOIN Order
   ON Customer.cid == Order.cid
   INNER JOIN Item
   ON Item.iid == Order.iid
   WHERE Item.iprice > 100
   AND Customer.cage < 18;
```



Good plan => Good performance
Problem: CBO can make mistakes
esp. *Cardinality Estimation*

Rise of Big Data Systems

Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

*The **root of all evil**, the Achilles Heel of query optimization, is the estimation of the size of intermediate results, known as **cardinalities**.* – [Guy Lohman, SIGMOD Blog 2014]

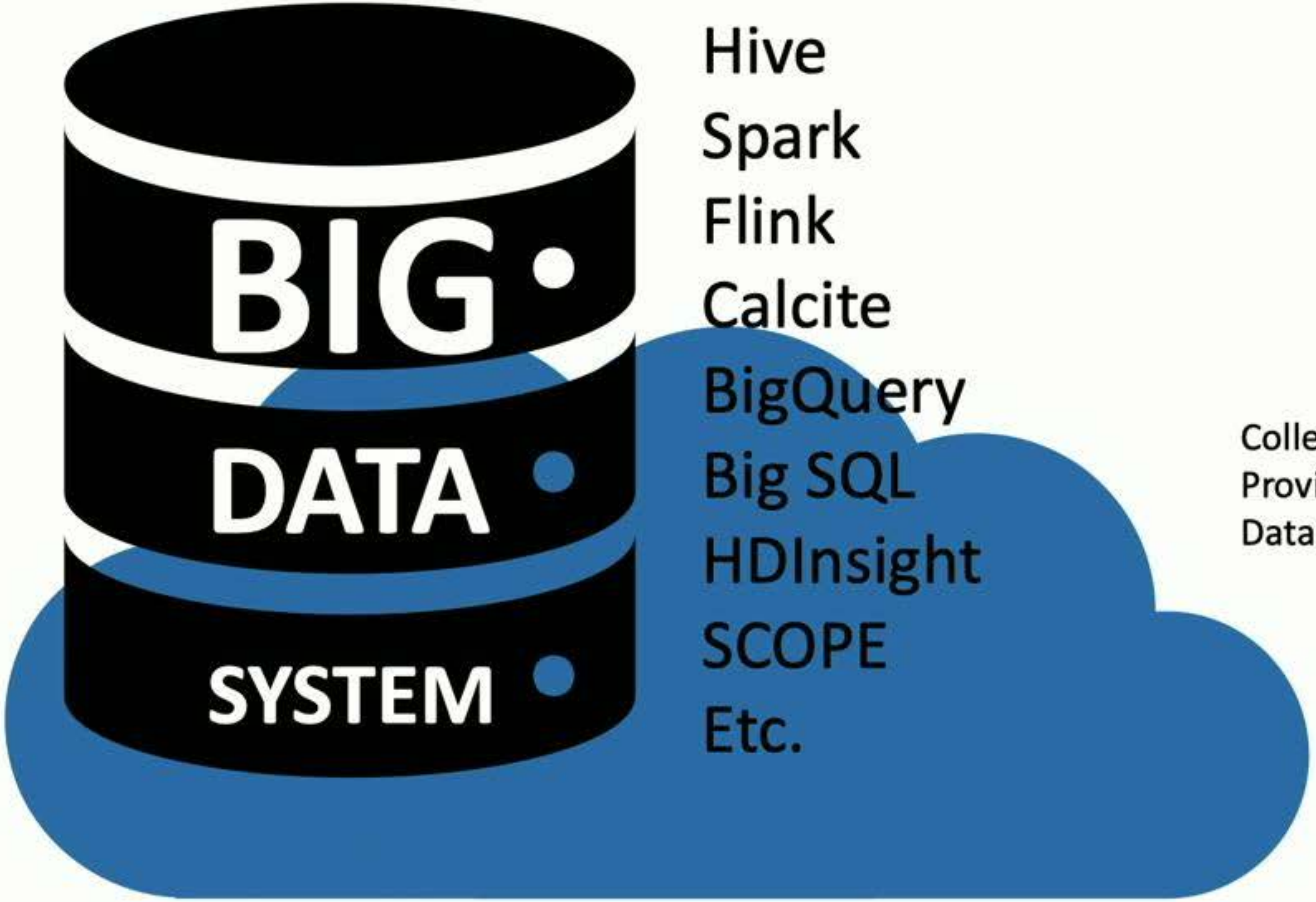
Rise of Big Data Systems

Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

TUNING!

Collecting Statistics
Providing Query Hints
Database Administration

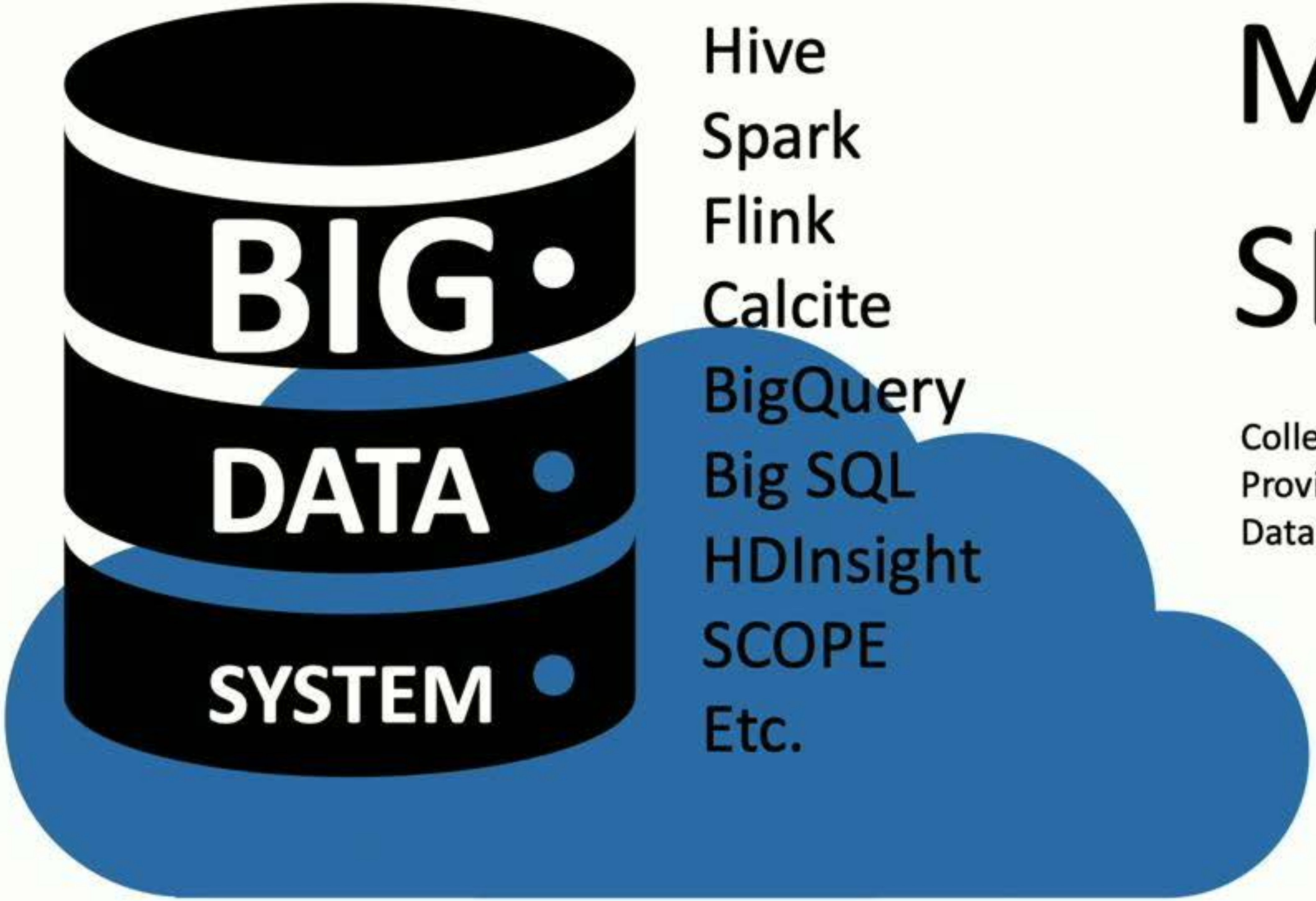
Rise of the Clouds



Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

Collecting Statistics
Providing Query Hints
Database Administration

Rise of the Clouds

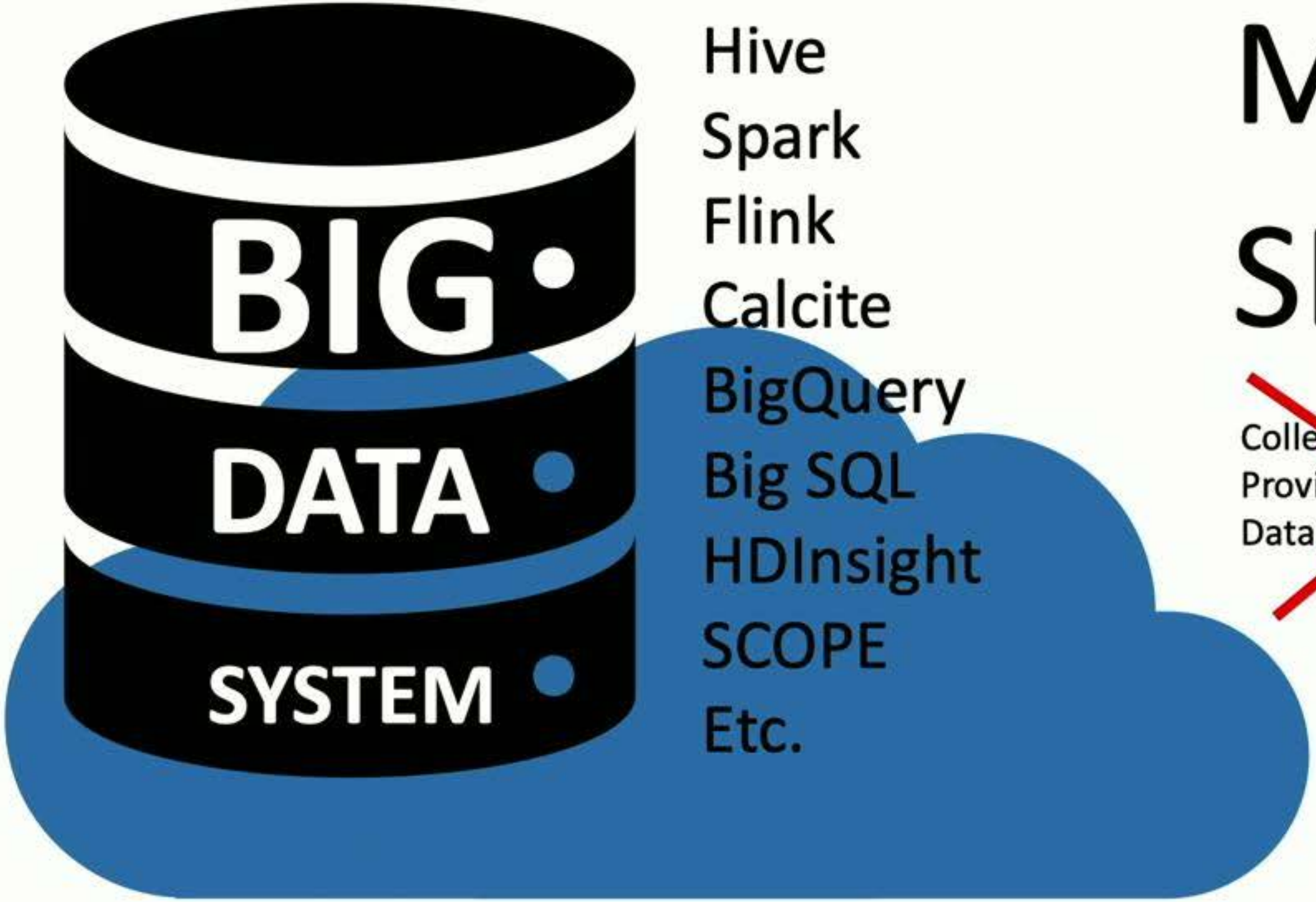


Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

MANAGED
SERVERLESS

Collecting Statistics
Providing Query Hints
Database Administration

Rise of the Clouds



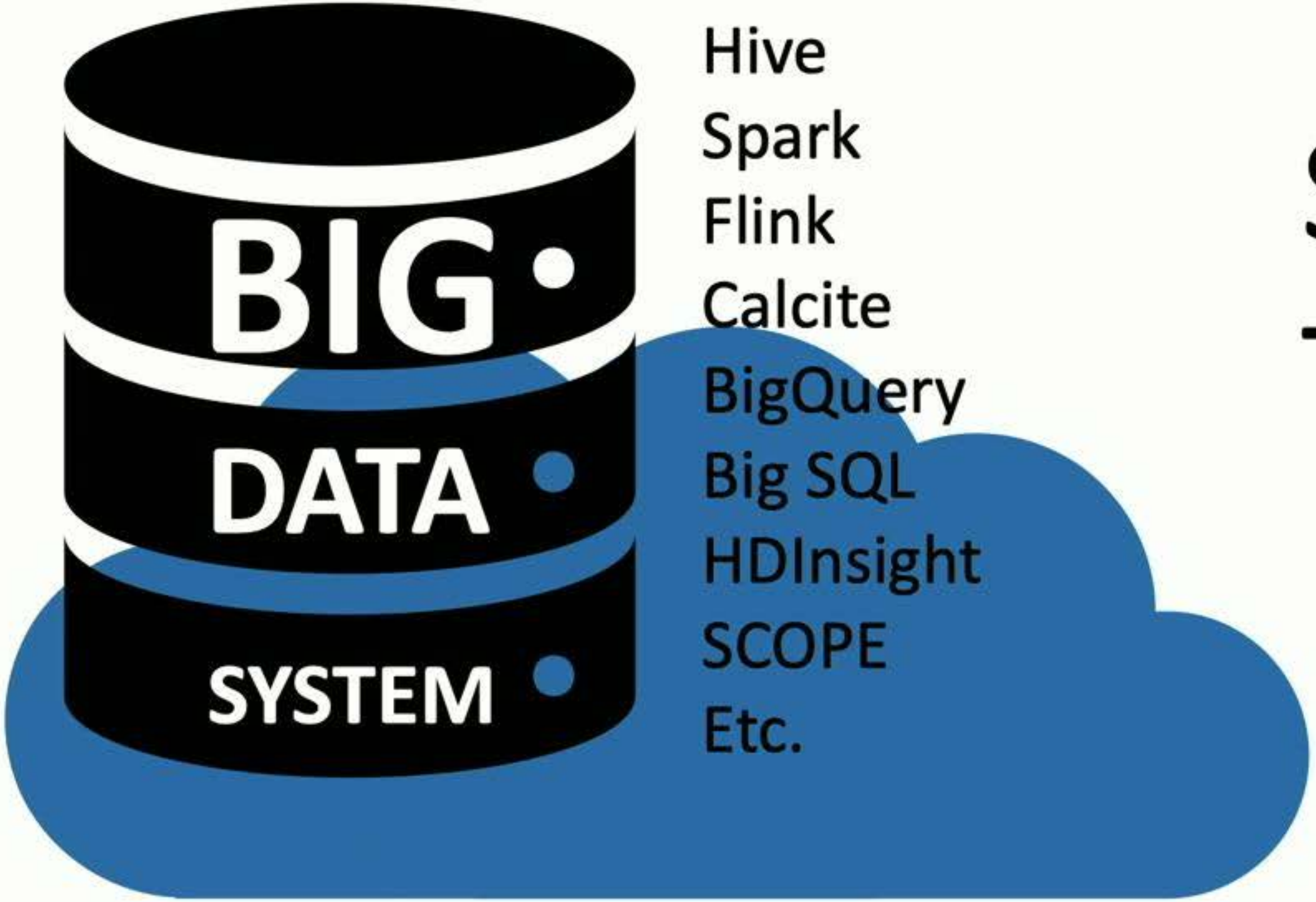
Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

MANAGED SERVERLESS

~~Collecting Statistics~~
~~Providing Query Hints~~
~~Database Administration~~

No Admin
No Expertise
No Control

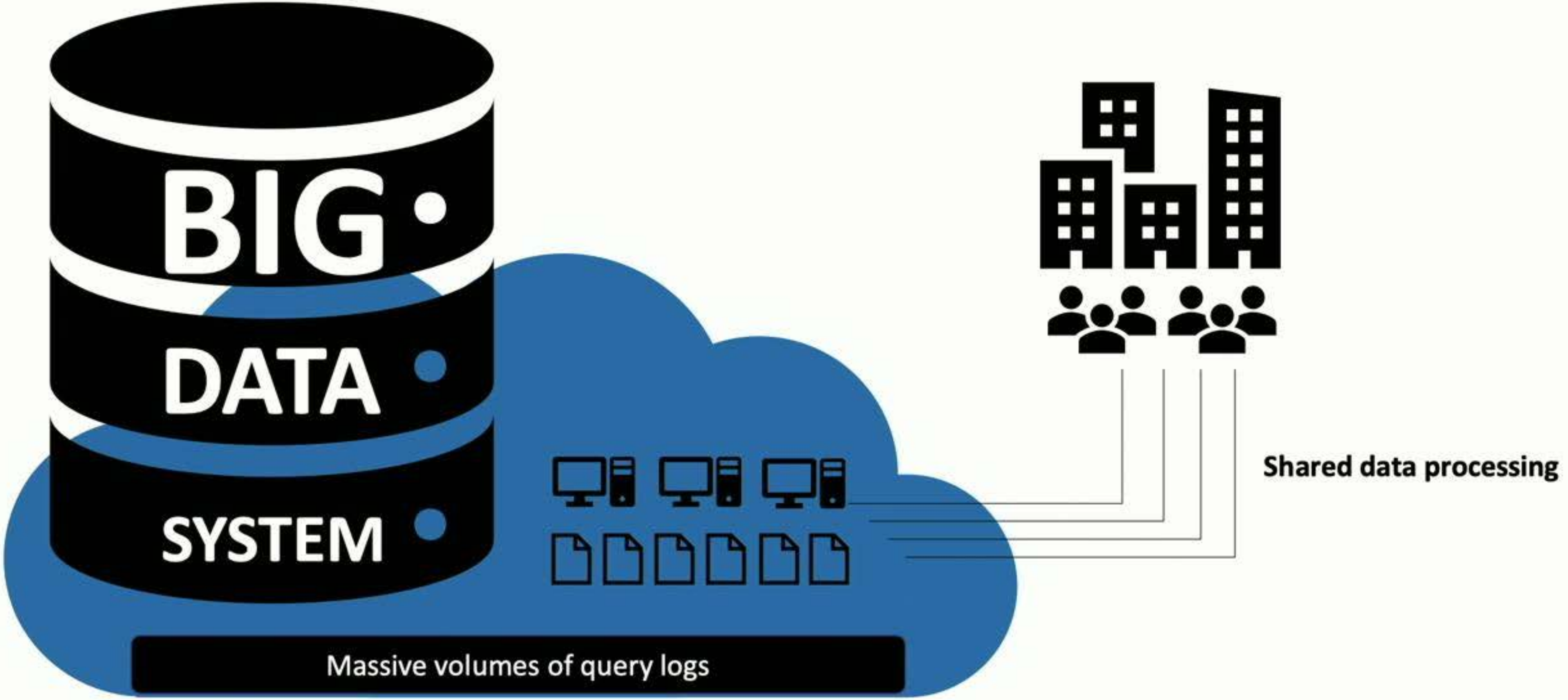
Rise of the Clouds



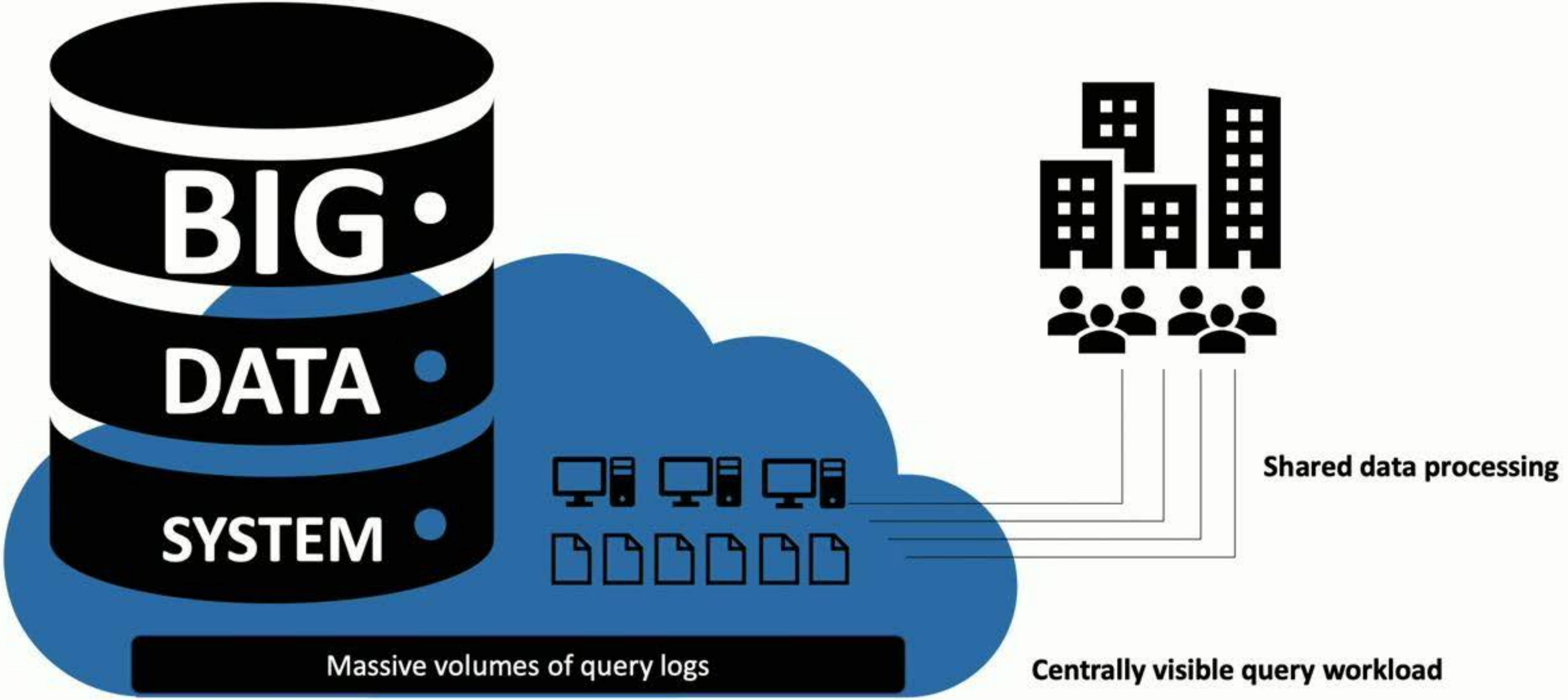
Hive
Spark
Flink
Calcite
BigQuery
Big SQL
HDInsight
SCOPE
Etc.

SELF TUNING!

Hope: Shared Cloud Infrastructures



Hope: Shared Cloud Infrastructures

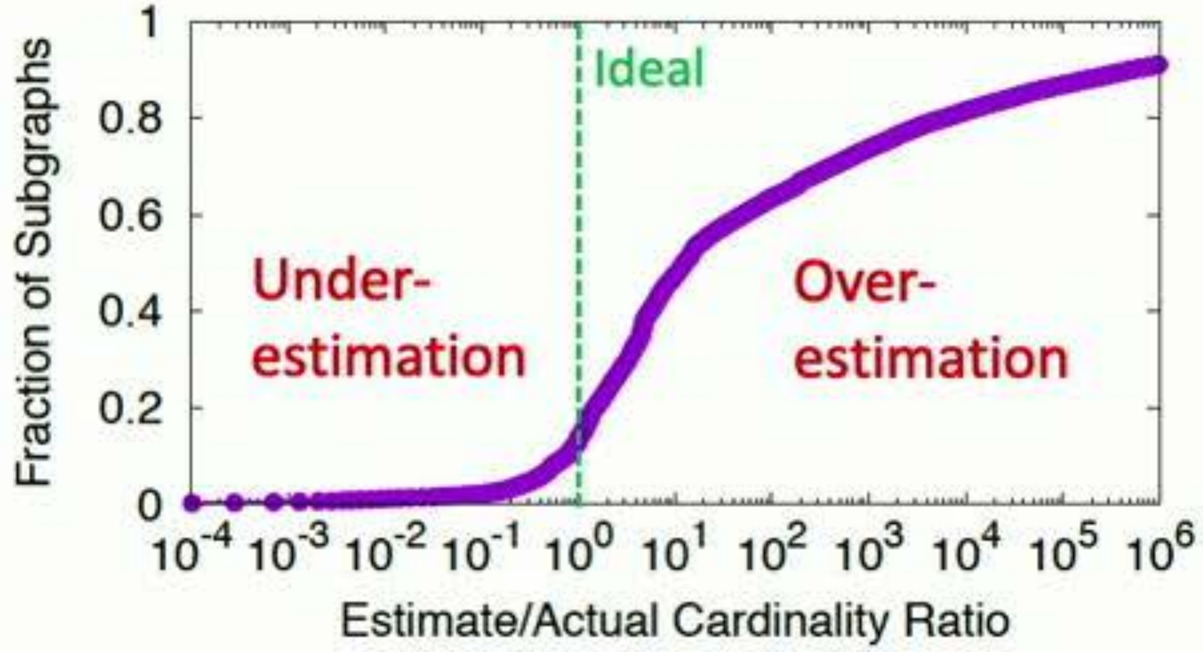


Cosmos: shared cloud infra at Microsoft

- SCOPE Workloads:
  - Batch processing in a job service
  - 100Ks jobs; 1000s users; EBs data; 100Ks nodes
- Cardinality estimation in SCOPE:
  - 1 day's log from Asimov

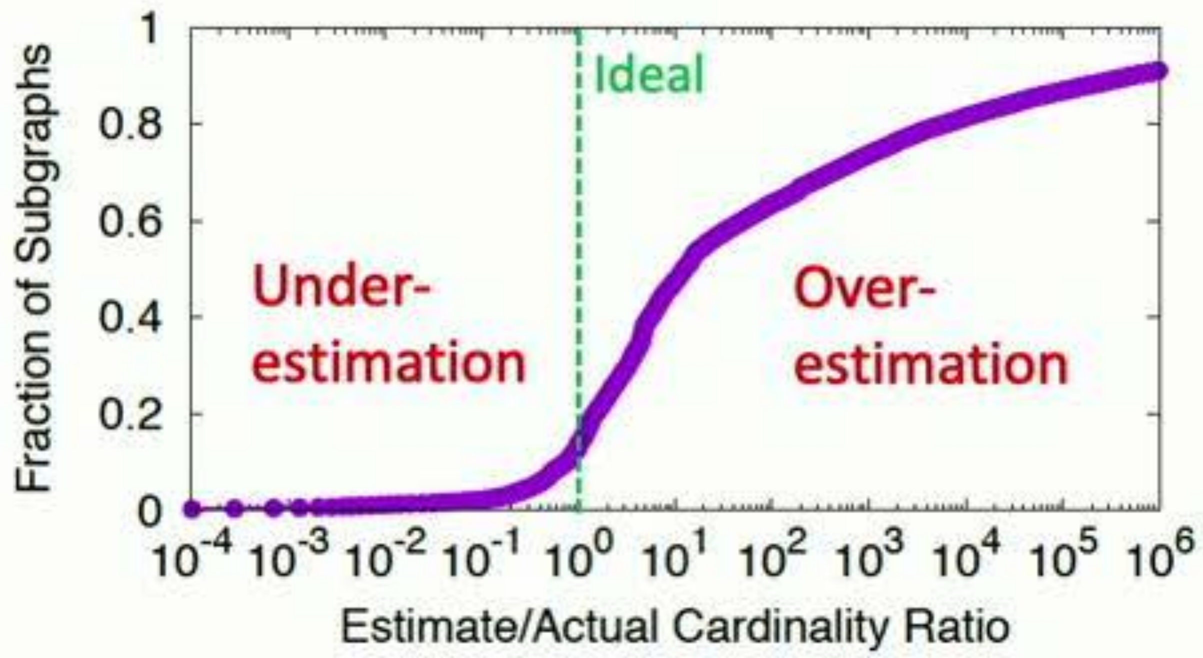
Cosmos: shared cloud infra at Microsoft

- SCOPE Workloads:
  - Batch processing in a job service
  - 100Ks jobs; 1000s users; EBs data; 100Ks nodes
- Cardinality estimation in SCOPE:
  - 1 day's log from Asimov
  - Lots of constants for best effort estimation
  - Big data, unstructured Data, custom code



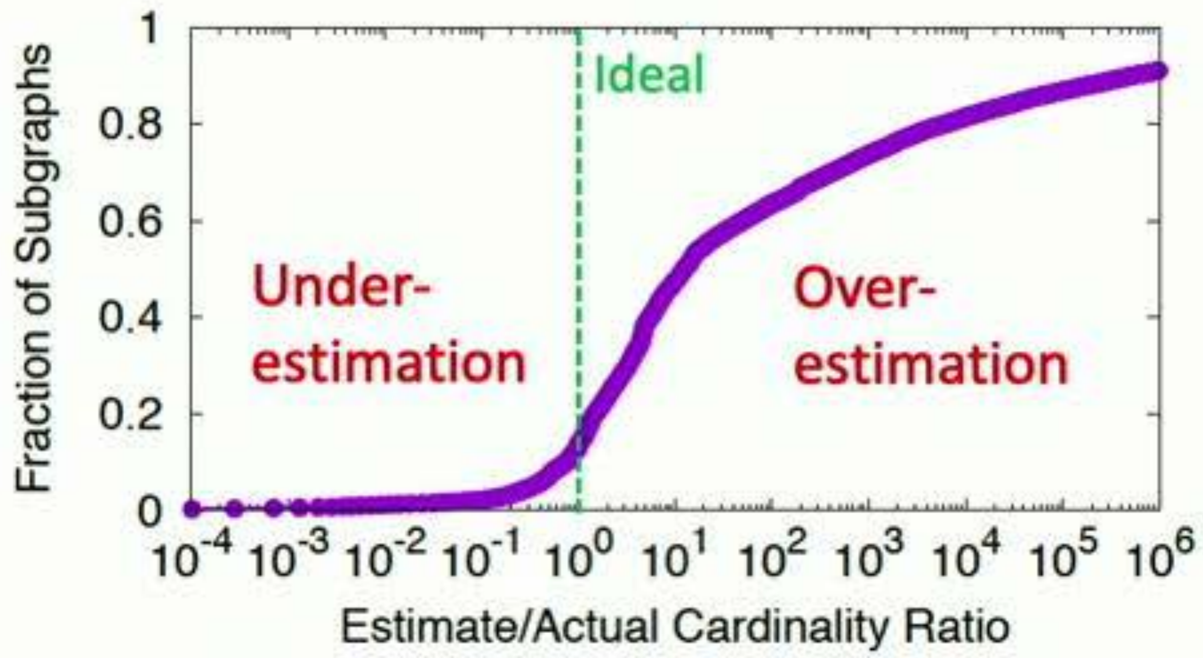
Cosmos: shared cloud infra at Microsoft

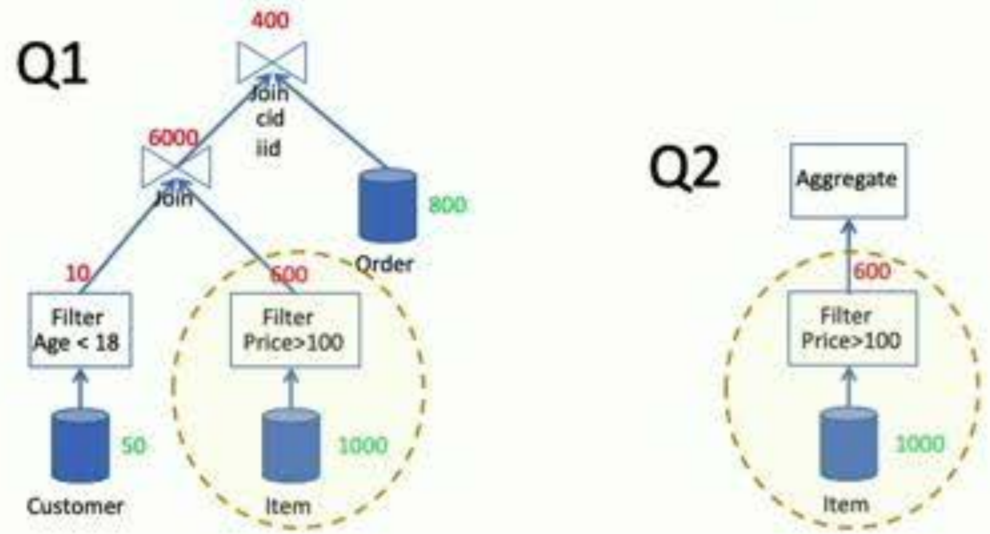
- SCOPE Workloads:
  - Batch processing in a job service
  - 100Ks jobs; 1000s users; EBs data; 100Ks nodes
- Cardinality estimation in SCOPE:
  - 1 day's log from Asimov
  - Lots of constants for best effort estimation
  - Big data, unstructured Data, custom code
- Workload patterns
  - Recurring jobs
  - Shared query subgraphs



Cosmos: shared cloud infra at Microsoft

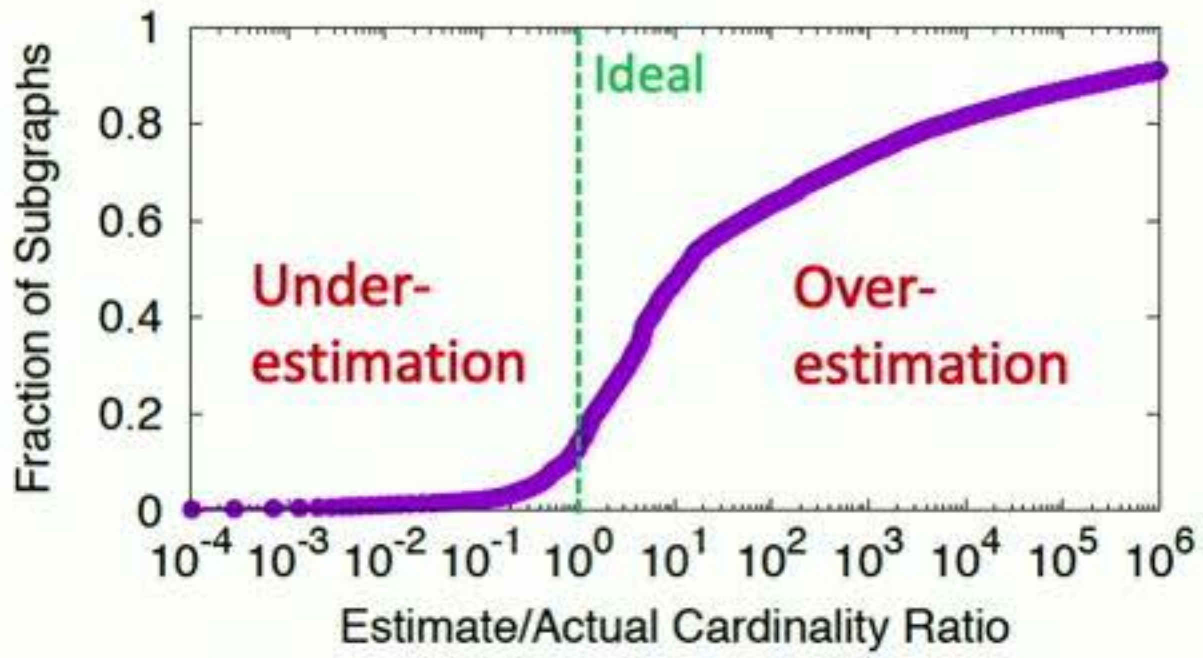
- SCOPE Workloads:
  - Batch processing in a job service
  - 100Ks jobs; 1000s users; EBs data; 100Ks nodes
- Cardinality estimation in SCOPE:
  - 1 day's log from Asimov
  - Lots of constants for best effort estimation
  - Big data, unstructured Data, custom code
- Workload patterns
  - Recurring jobs
  - Shared query subgraphs

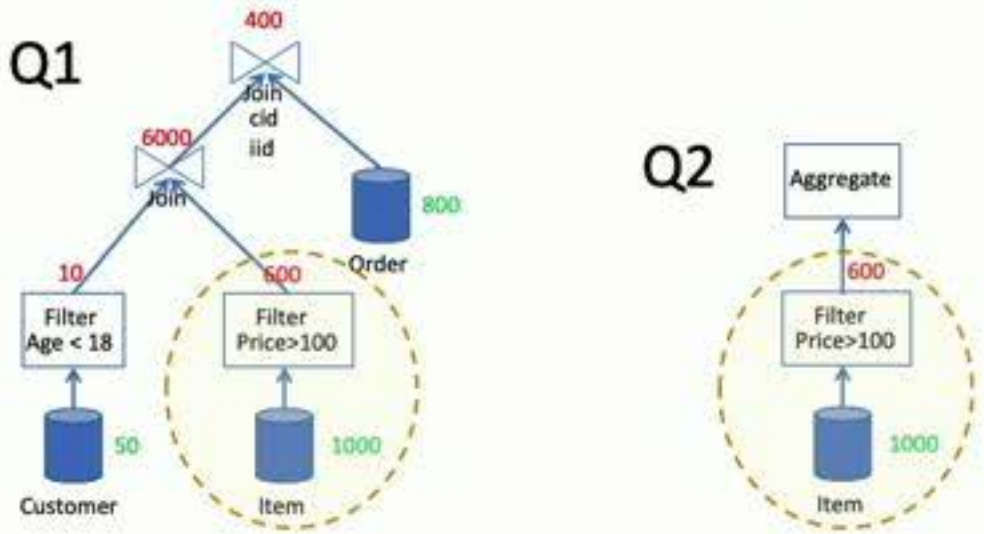




Cosmos: shared cloud infra at Microsoft

- SCOPE Workloads:
  - Batch processing in a job service
  - 100Ks jobs; 1000s users; EBs data; 100Ks nodes
- Cardinality estimation in SCOPE:
  - 1 day's log from Asimov
  - Lots of constants for best effort estimation
  - Big data, unstructured Data, custom code
- Workload patterns
  - Recurring jobs
  - Shared query subgraphs
- Can we *learn* cardinality models?





Learning Cardinality Model

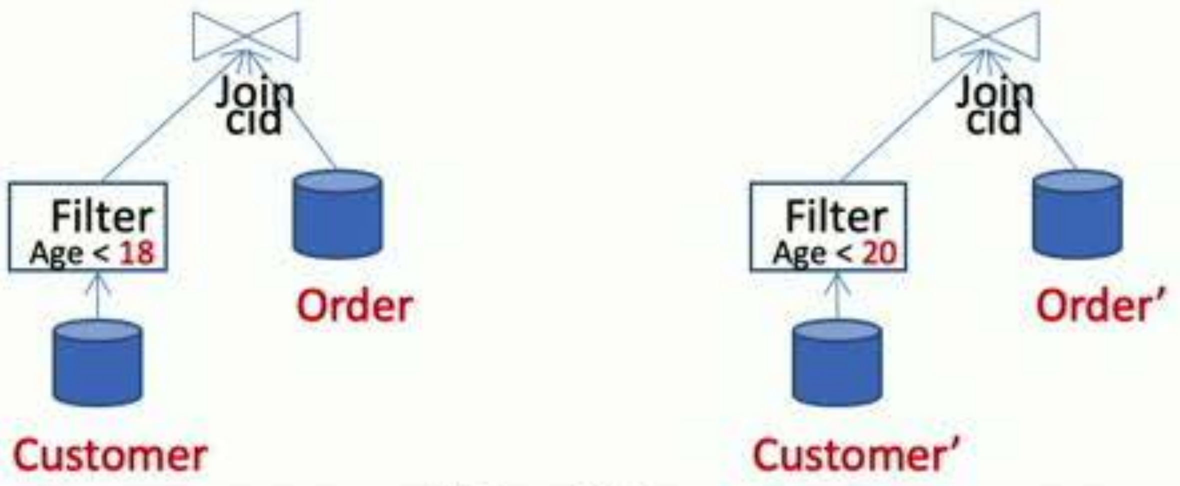
- Strict: cache previously seen values
  - Low coverage
  - Online feedback
- General: learning a single model
  - Hard to featurize
  - Hard to train
  - Prediction latency
  - Low accuracy
- Template: learning a model per subgraph template

  => No one-size-fits-all

Subgraph Type	Logical Expression	Parameter Values	Data Inputs
Strict	Fixed	Fixed	Fixed
General	Variable	Variable	Variable
Template	Fixed	Variable	Variable

Learned Cardinality Models

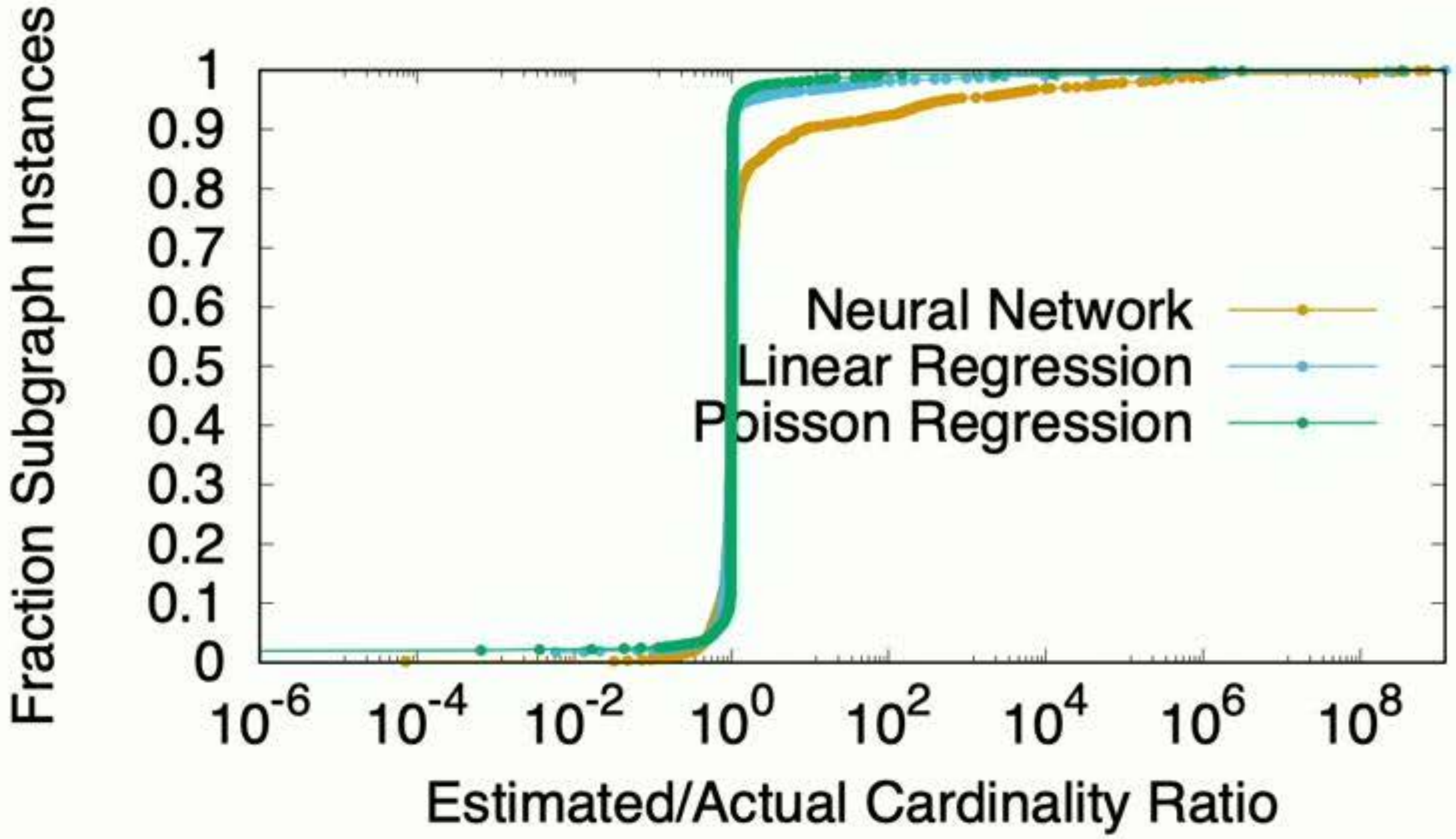
- Subgraph Template:
  - Same logical subexpression
  - Different physical implementation
  - Different parameters and inputs
- Feature Selection
- Model Selection
  - Generalized liner models due to their interpretability
  - More complex models, such as multi-layer perceptron harder to train



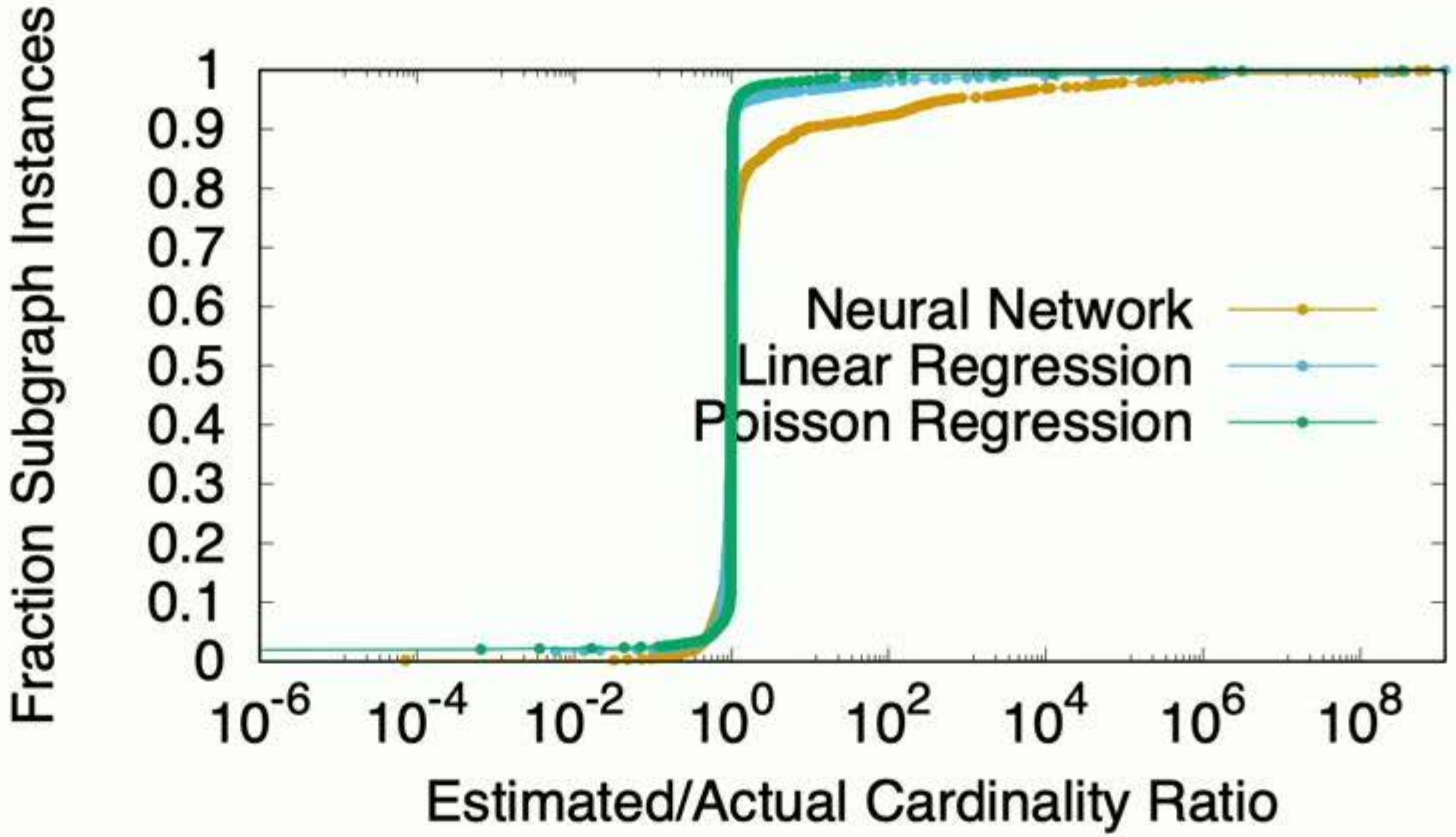
Name	Description
JobName	Name of the job containing the subgraph
NormJobName	Normalize job name
InputCardinality	Total cardinality of all inputs to the subgraph
Pow(InputCardinality, 2)	Square of InputCardinality
$Sqrt$(InputCardinality)	Square root of InputCardinality
Log(InputCardinality)	Log of InputCardinality
AvgRowLength	Average output row length
InputDataset	Name of all input datasets to the subgraph
Parameters	One or more parameters in the subgraph

Model	Percentage Error	Pearson Correlation
Default Optimizer	2198654	0.41
Adjustment Factor (LEO)	1477881	0.38
Linear Regression	11552	0.99
Neural Network	9275	0.96
Poisson Regression	696	0.98

Accuracy: 10-fold cross validation



Accuracy: 10-fold cross validation

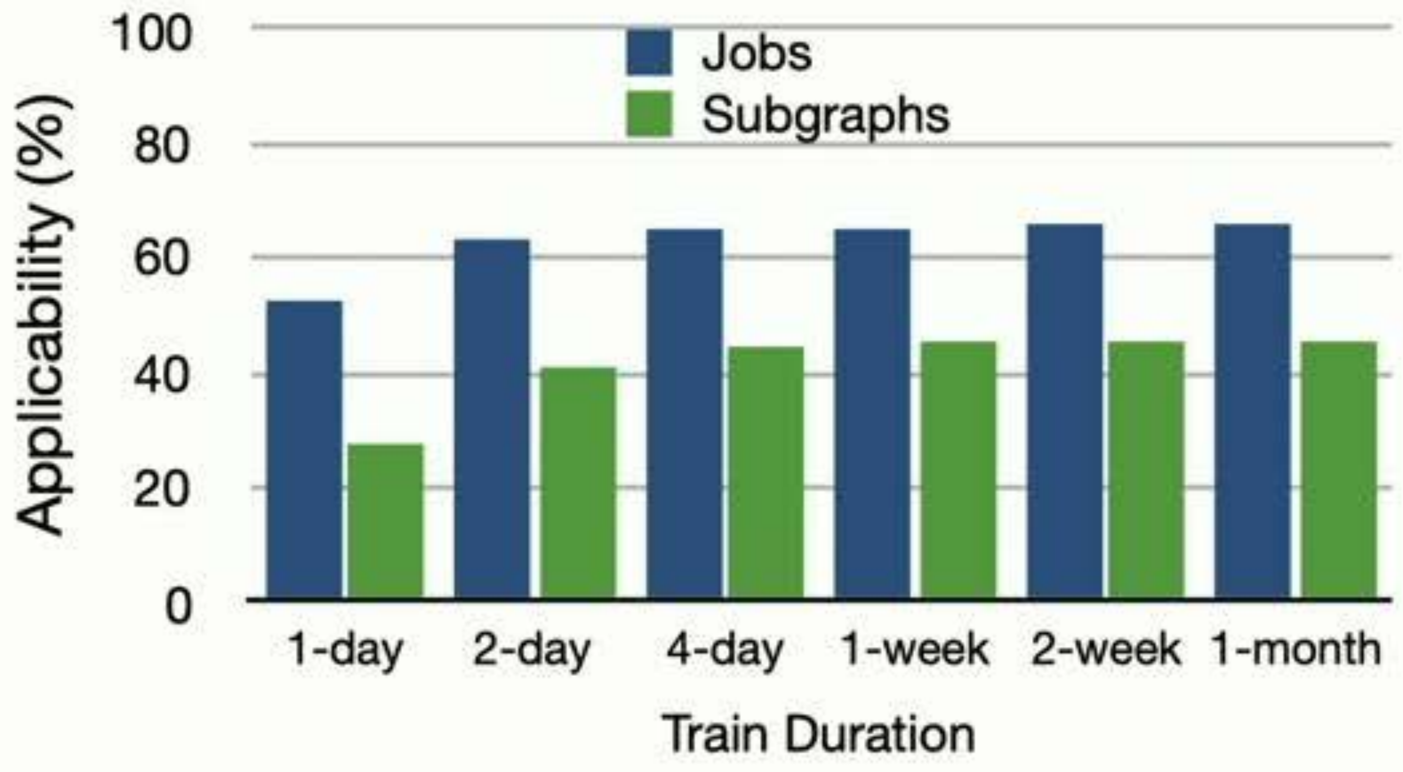


Model	75th Percentile Error	90th Percentile Error
Default SCOPE	74602%	5931418%
Poisson Regression	1.5%	32%

Note: Neural network overfits due to small observation and feature space per model

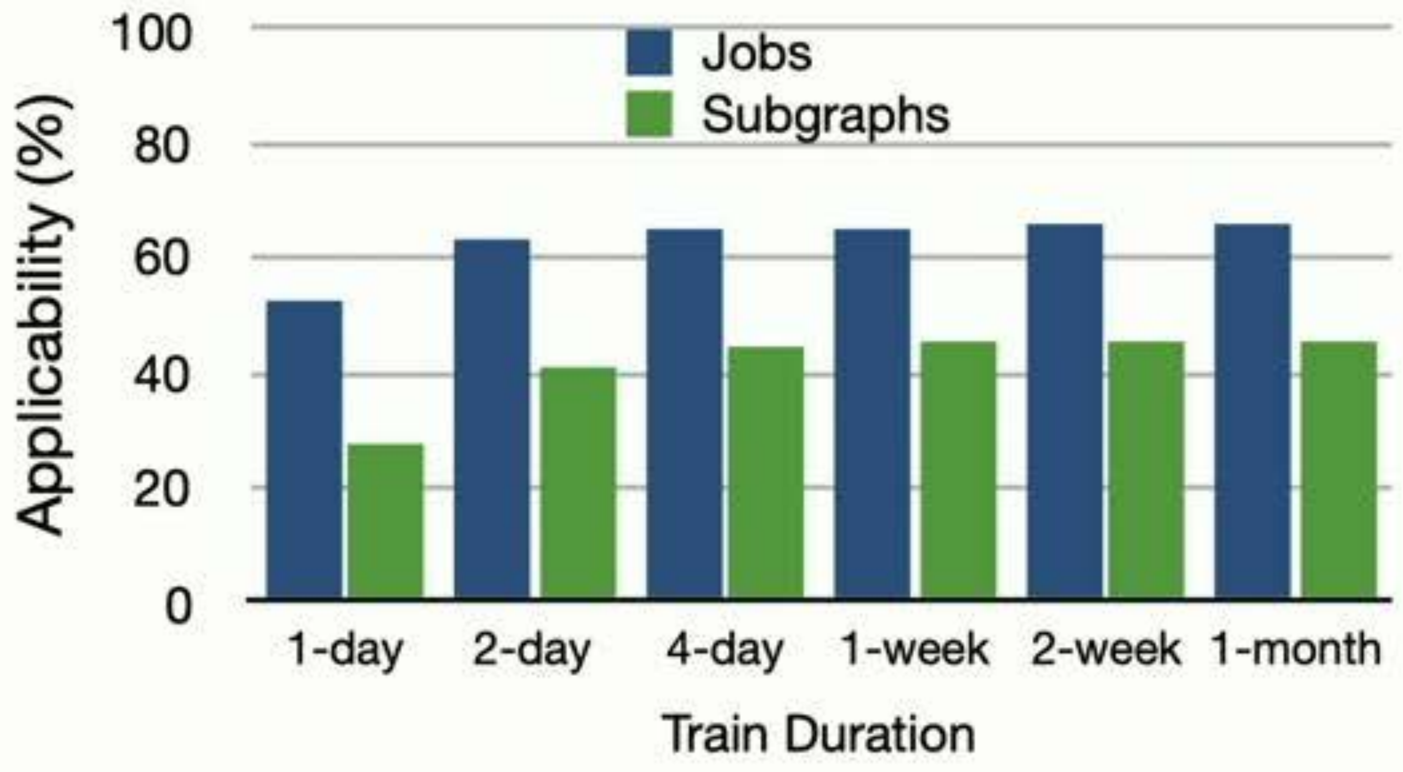
Applicability: %tage subgraphs having models

Varying Training Window

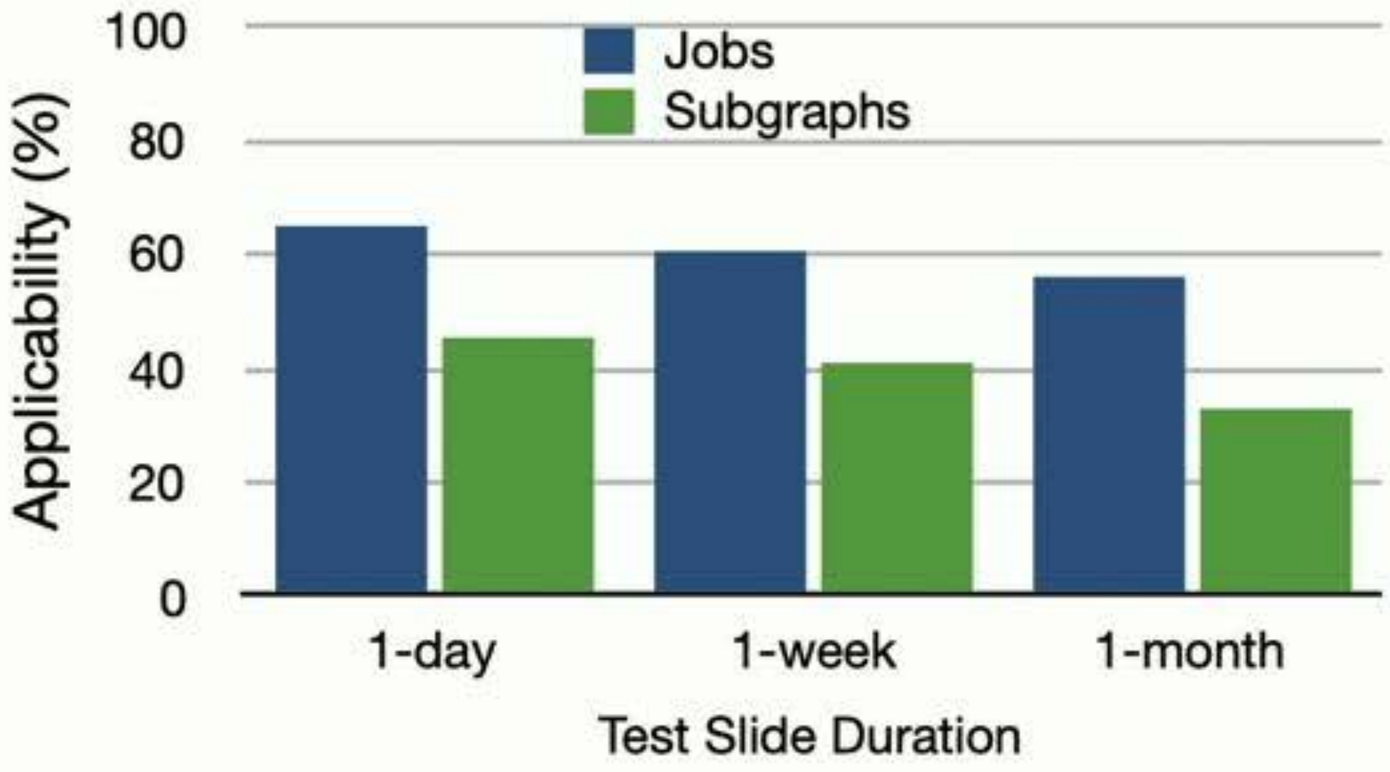


Applicability: %tage subgraphs having models

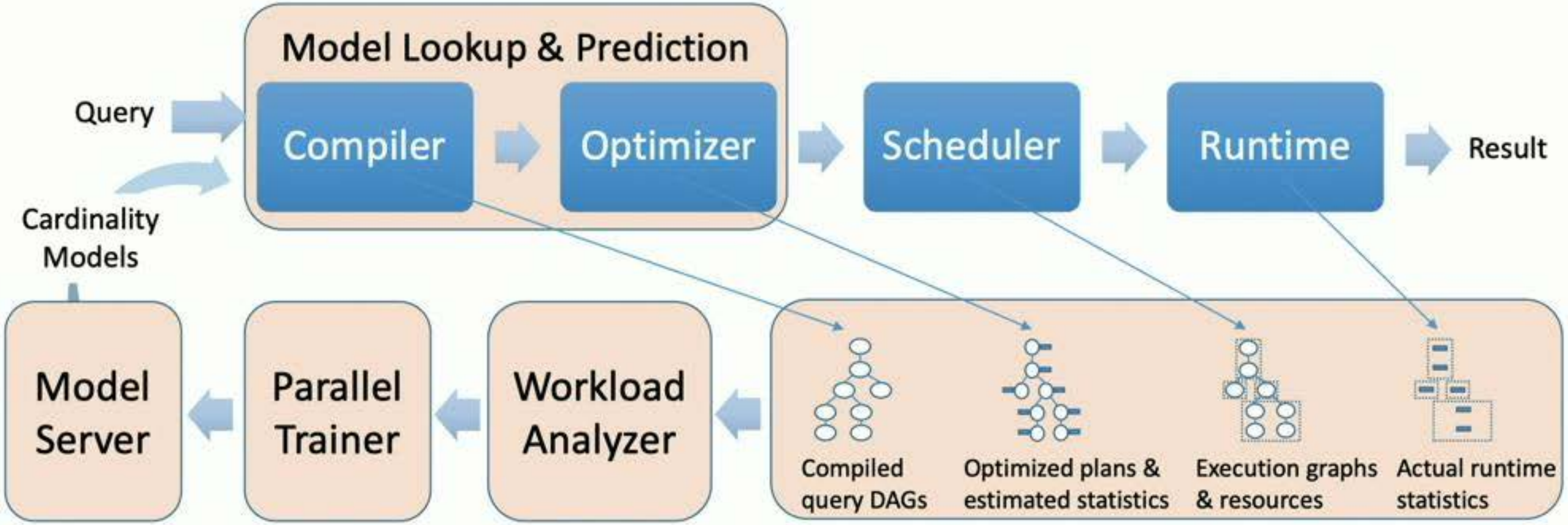
Varying Training Window



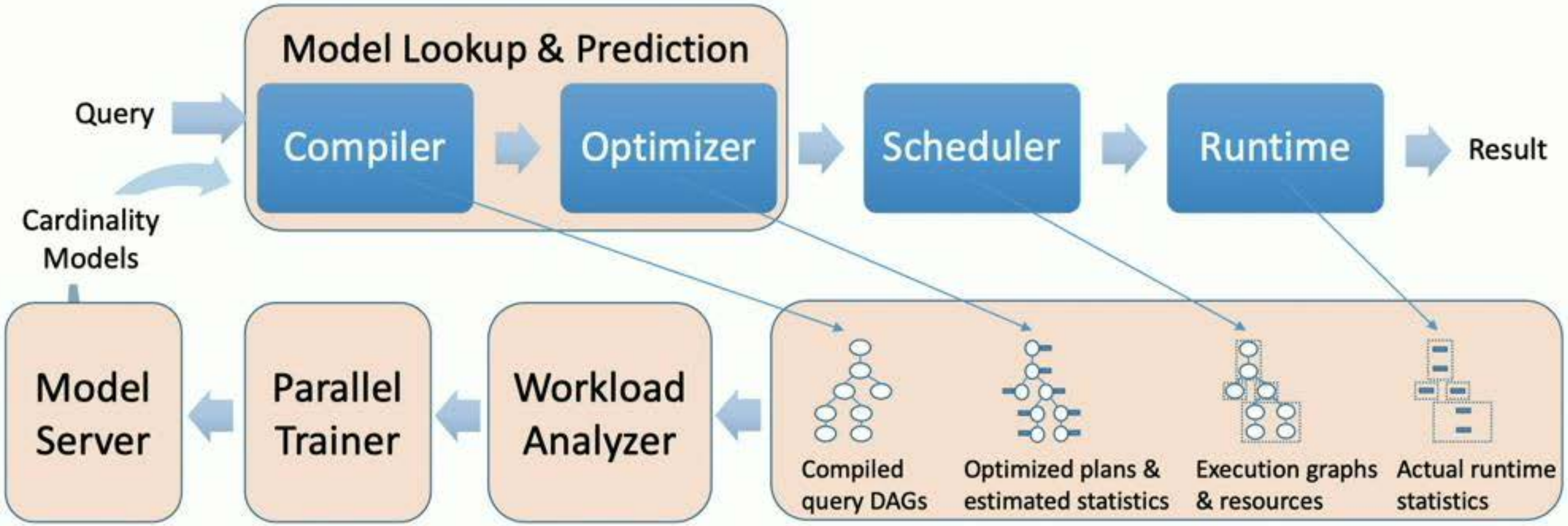
Sliding Test Window



End-to-end Feedback Loop



End-to-end Feedback Loop

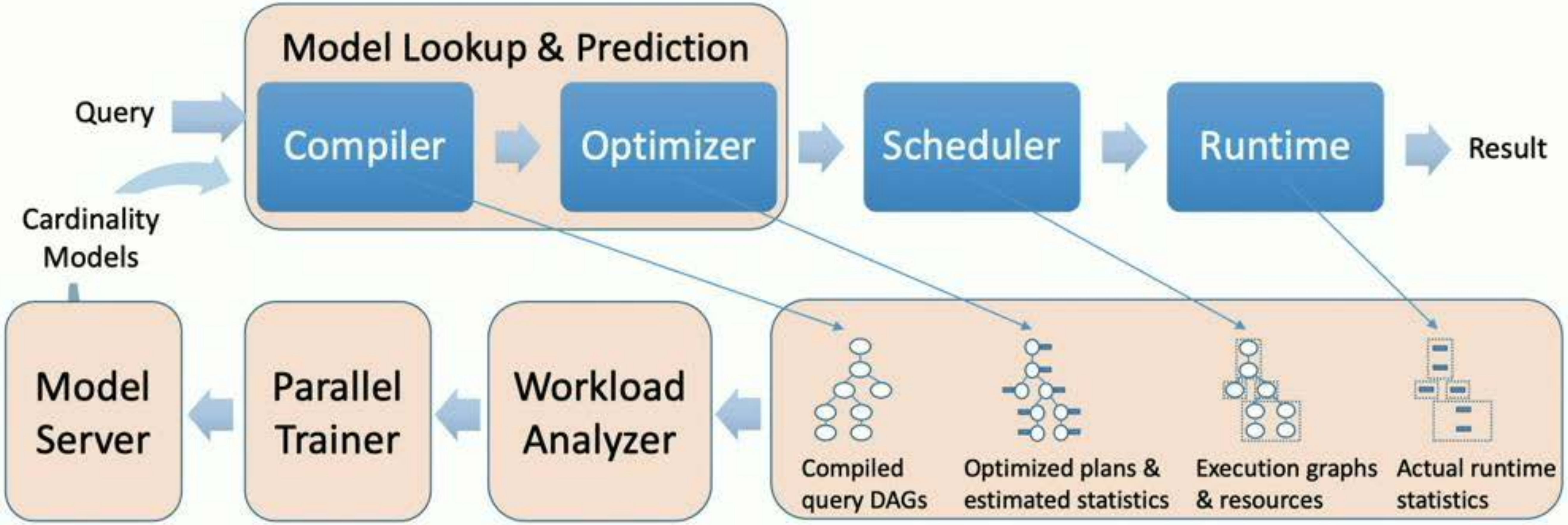


Trained <u>offline</u> over new batches of data

Large number of smaller, <u>highly accurate</u> models

End-to-end Feedback Loop

Annotation hints to the query optimizer



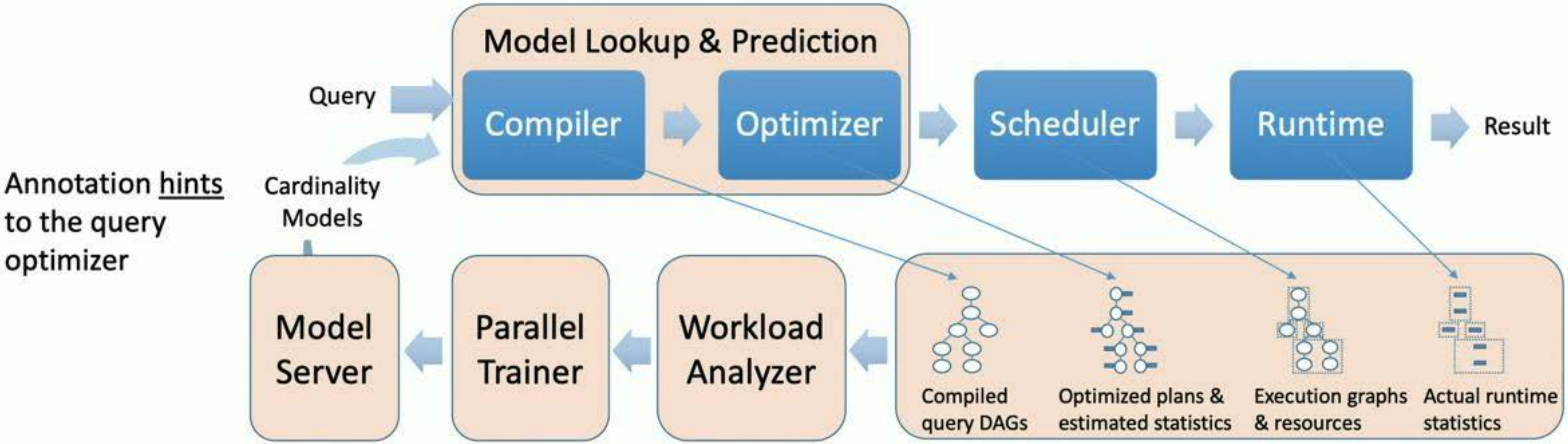
Trained offline over new batches of data

Large number of smaller, highly accurate models

End-to-end Feedback Loop

Easy to featurize with low overhead

Accurate and easy to understand

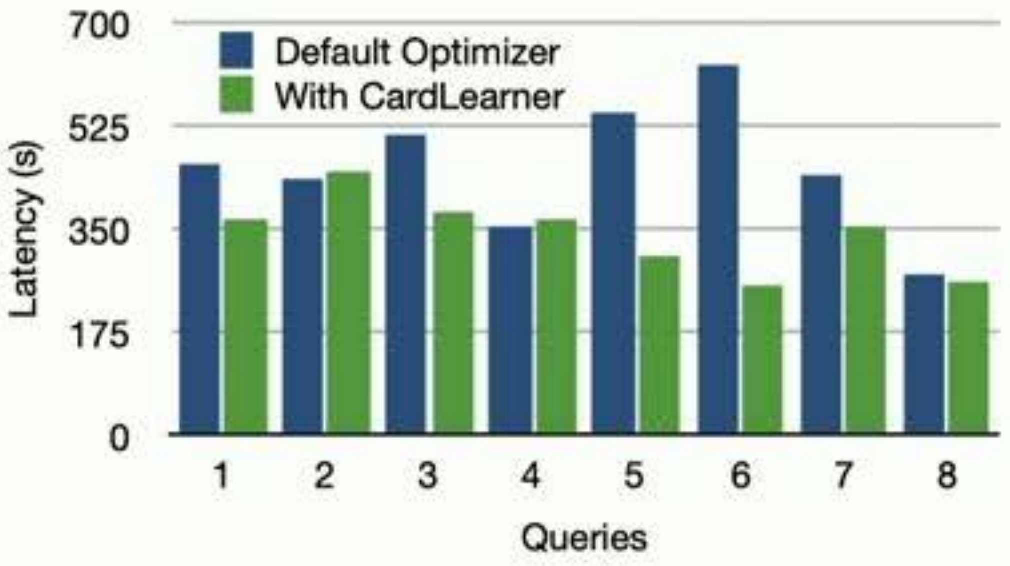


Trained offline over new batches of data

Large number of smaller, highly accurate models

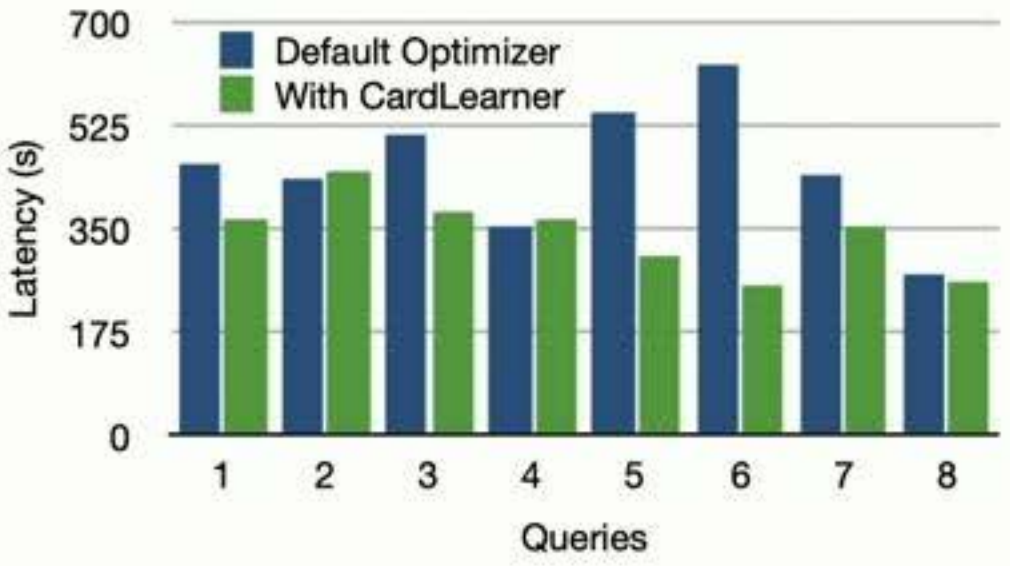
Performance

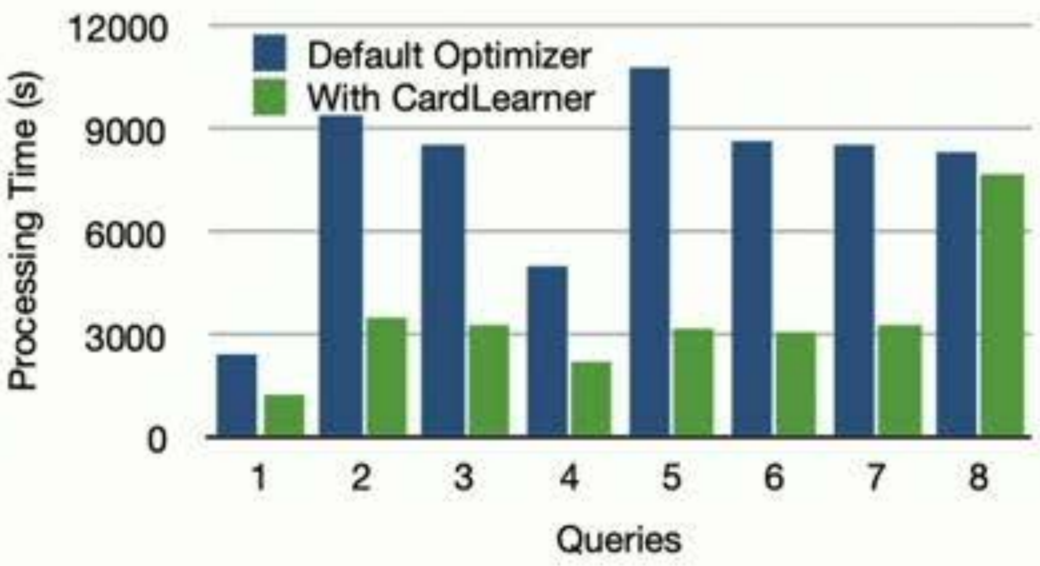
- Subset of hourly jobs from Asimov
- These queries process unstructured data, use SPJA operators, and a UDO
- Re-ran the queries over same production data, but with redirected output



Performance

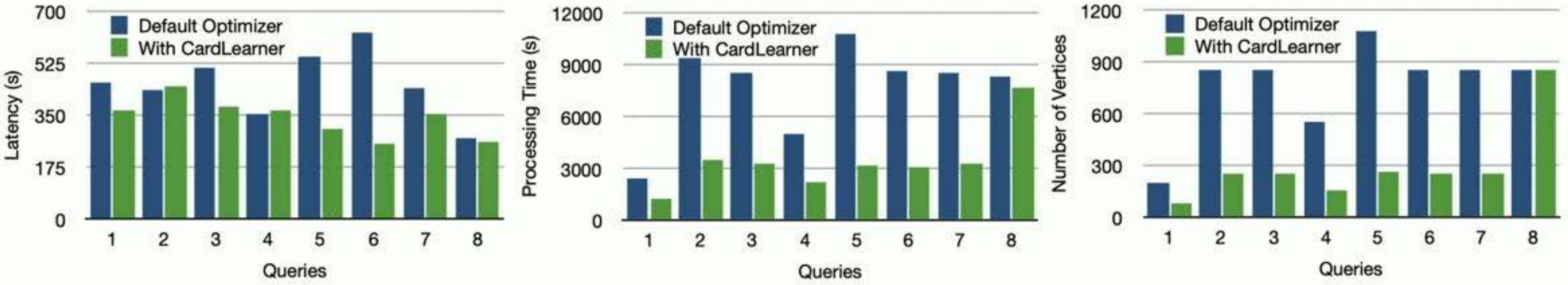
- Subset of hourly jobs from Asimov
- These queries process unstructured data, use SPJA operators, and a UDO
- Re-ran the queries over same production data, but with redirected output



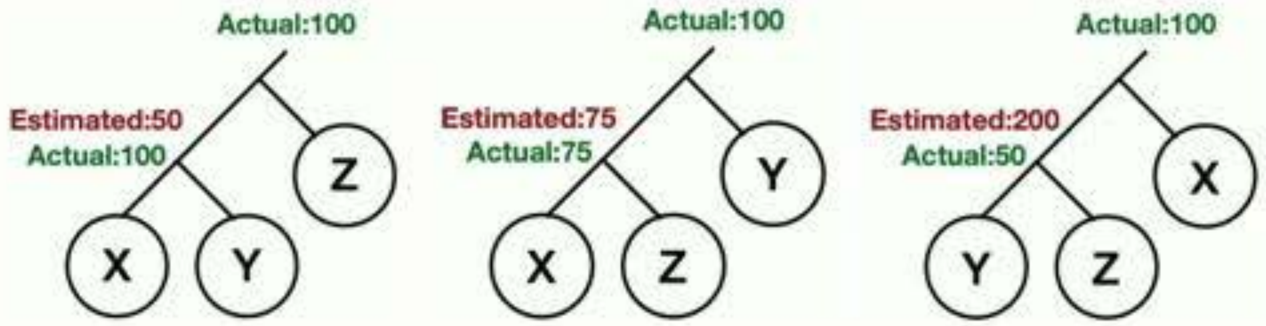


Performance

- Subset of hourly jobs from Asimov
- These queries process unstructured data, use SPJA operators, and a UDO
- Re-ran the queries over same production data, but with redirected output



Avoiding Learning Bias



- Learning only what is seen
- Exploratory join ordering
  - Actively try different join orders
  - Pruning: discard plans with subexpressions that are more expensive than at least one other plan
  - Maximize new observations when comparing plans
- Execution strategies
  - Static workload tuning
  - Using sample data
  - Leveraging recurring/overlapping jobs

Takeaways

- Big data systems increasingly use cost-based optimization
- Users cannot tune these systems in managed/serverless services
- Hard to achieve a one-size-fits-all query optimizer
- Instance optimized systems are more feasible
- Very promising results from SCOPE workloads:
  - Could achieve very high accuracy
  - Reasonably large applicability, could further apply exploration
  - Performance gains, most significant being less resource consumption
- Learned cardinality models a step towards self-learning optimizers

Machine Learning in Google BigQuery

Amir Hormati (hormati@google.com)

Google Cloud

Agenda

BigQuery

Why BigQuery ML?

Syntax

Iterative Gradient Descent

Closed Form Solution

Questions

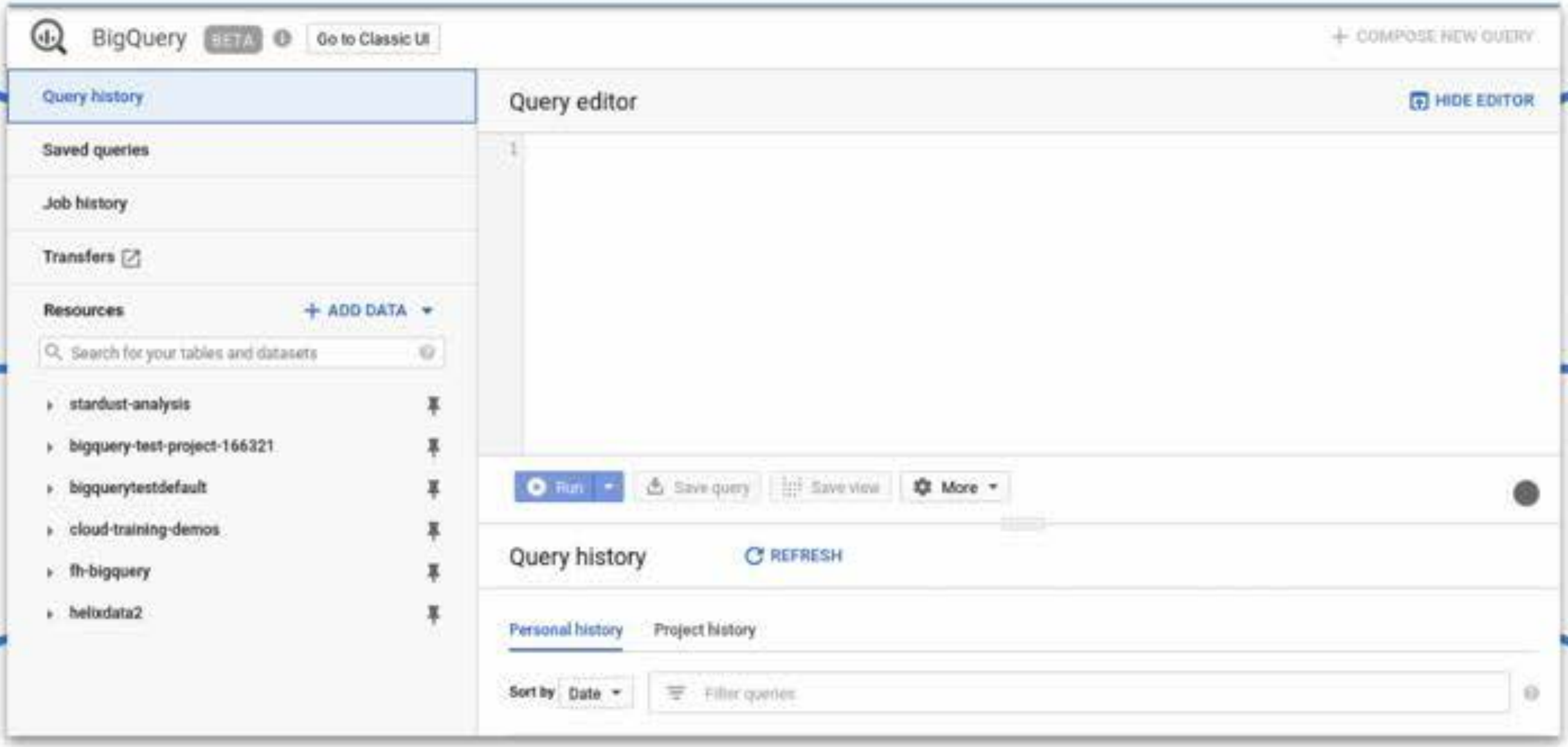
BigQuery

- Google's cloud-based SQL datawarehouse-as-a-service for analytics:

Enterprise data warehouse for analytics

Convenience of standard SQL

Fully managed and serverless



Petabyte-scale storage and queries

Encrypted, durable and highly available

Real-time analytics on streaming data

https://cloud.google.com/bigquery

Google Cloud

BigQuery ML

- SQL analysts use databases to extract insights from their data.

```
> SELECT AVG(income) FROM census_data GROUP BY state;

> SELECT cid, COUNT(*) FROM orders GROUP BY cid ORDER BY COUNT(*) DESC
```

- Give SQL analysts access to familiar math concepts, statistical methods, and algorithms without learning new tools and languages.

BigQuery ML

- Democratizes ML for business customers.
  - Experts in TensorFlow, scikit-learn, etc are rare.
  - Experts in SQL are far more common.

- Analyze large datasets without sampling.
  - Scale to petabytes of data

- Avoids slow, cumbersome moving of data to/from of database.
  - Learn ML models directly in BigQuery UI.

Existing Syntax:

Example 1:

```
CREATE PROCEDURE [dbo].[RxTrainLogitModel] (@trained_model
varbinary(max) OUTPUT)

AS
BEGIN
...
  EXEC sp execute external script @language = N'R',
                                  @script = N'
## Create model
logitObj <- rxLogit(tipped ~ passenger count +
trip distance + trip_time_in_secs + direct_distance, data =
InputDataSet)
summary(logitObj)

## Serialize model
trained_model <- as.raw(serialize(logitObj, NULL));
'...
```

Example 2:

```
SELECT glm('warpbreaks_dummy',
           'glm_model',
           'breaks',
           'ARRAY[1.0,"wool_B","tension_M", "tension_H"]',
           'family=poisson, link=log');

SELECT
    w.id,
      glm_predict(
        coef,
        ARRAY[1, "wool_B", "tension_M", "tension_H"]::float8[],
        'log') AS mu
FROM warpbreaks_dummy w, glm_model m
```

BigQuery ML Syntax

- Extension of standard SQL DDLs for creating models:

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE
MODEL}
model name
[OPTIONS(model option_list)]
[AS query_statement]
```

```
> CREATE MODEL income_model
    OPTIONS (model_type='linear_reg')
    AS SELECT state, job, income as label FROM census_data;
```

BigQuery ML Syntax

- TVFs for prediction and other model operations:

```
ML.PREDICT(MODEL model_name,
          {TABLE table_name | (query_statement)})
```

```
> SELECT predicted_income FROM ML.PREDICT(MODEL 'income_model',
    SELECT state, job FROM customer_data);
```

```
ML.EVALUATE(MODEL model_name
           [, {TABLE table_name | (query_statement)}]
           [, STRUCT(<T> AS threshold)])
```

Iterative Gradient Descent (IGD)

- Find w such that:

  X w = y

  X : Training data (rows: training examples, cols: features)

  w : Weights

  y : Observations (income in our running example)

- Core learning algorithm is **gradient descent.**
  - Minimizes the objective:

    $$\min_{\mathbf{w}} \Sigma_i L(\mathbf{w}^T\mathbf{x}_i, y_i) + \lambda_1||\mathbf{w}||_1 + \lambda_2(||\mathbf{w}||_2)^2$$

  - Includes support for L1 and L2 regularization.

BigQuery ML Syntax

- TVFs for prediction and other model operations:

```
ML.PREDICT(MODEL model_name,
          {TABLE table_name | (query_statement)})

> SELECT predicted_income FROM ML.PREDICT(MODEL 'income_model',
    SELECT state, job FROM customer_data);

ML.EVALUATE(MODEL model_name
           [, {TABLE table_name | (query_statement)}]
           [, STRUCT(<T> AS threshold)])
```

Iterative Gradient Descent (IGD)

- Find w such that:

  $X w = y$

  X : Training data (rows: training examples, cols: features)

  w : Weights

  y : Observations (income in our running example)

- Core learning algorithm is **gradient descent.**
  - Minimizes the objective:

    $$\min_{\mathbf{w}} \Sigma_i L(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda_1 ||\mathbf{w}||_1 + \lambda_2 (||\mathbf{w}||_2)^2$$

  - Includes support for L1 and L2 regularization.

Google Cloud

Iterative Gradient Descent (IGD)

- Gradient descent implemented as sequence of **pure SQL queries.**

- **Represents data and models as tables:**

`data`

state	job	Income
NY	nurse	65000
CA	chef	55000
...	...	...

`model`

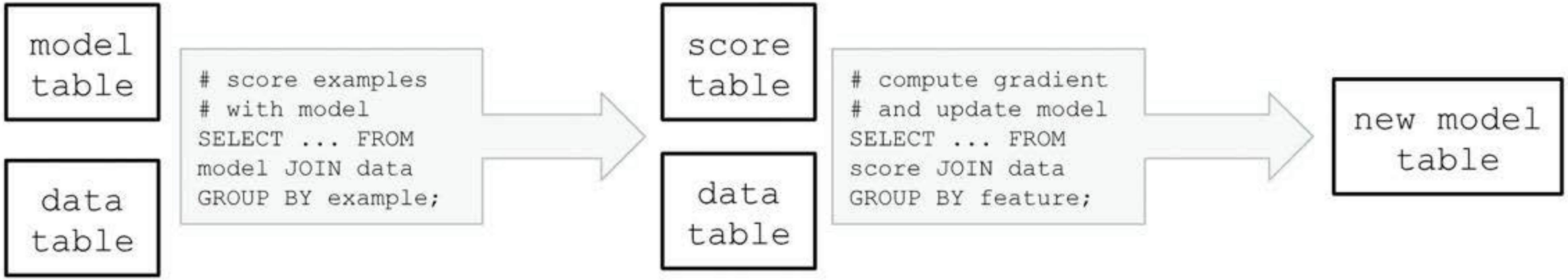
feature	weight
state:CA	+5.7
job:nurse	-3.5
...	...

- *[Umar Syed, Sergei Vassilvitskii:*
  SQML: large-scale in-database machine learning with pure SQL. *SoCC 2017: 659]*

Google Cloud

Iterative Gradient Descent (IGD)

- Each algorithm iteration issues SQL queries that join model to data, update model, then write model back to disk.



- Techniques from stats and ML fields to make the queries scale and hide the complexity while achieving high accuracy..

Closed form solution

- Find w such that:

  $X\,w = y$

  X : Training data $\rightarrow$ m x n matrix (m >> n)

  w : Weights $\rightarrow$ n x 1

  y : Observations $\rightarrow$ m x 1

Closed Form Solution

- ML training Algorithms mainly focus on computational linear algebra and optimizations.

- Closed form solutions expressed in matrix and vector operations.

  - Least square normal equation (linear regression)

$$w = (X^T X + \lambda I)^{-1} X^T y$$

Closed Form Solution

- Why closed form solution is not preferred in most ML platforms?
    - Load all of the training data X into memory for computing.
    - Parallel computing of extra-large linear algebra.
        - Matrix multiplication
        - Matrix Inversion

Closed Form Solution

- ML training Algorithms mainly focus on computational linear algebra and optimizations.
- Closed form solutions expressed in matrix and vector operations.
  - Least square normal equation (linear regression)

$$w = (X^T X + \lambda I)^{-1} X^T y$$

Closed Form Solution

- Matrix are represented as table:
  with schema <row:string, col:string, data:double>

- Matrix Multiplication is done via inner join.

```
SELECT
  A.row AS row,
  B.col AS col,
  SUM(A.value * B.value) AS value
FROM
  A JOIN B
  ON A.col = B.row
GROUP BY
  row, col;
```

Closed Form Solution

- Matrix Inversion
    - Matrix size is N x N, with N number of features
        - Single shard compute
    - Symmetric Positive-Definite(SPD) Matrix
    - Fast solver (using Cholesky decomposition)

Google Cloud

When to use IGD vs closed form?

1. If total cardinalities of training features are more than 10000, IGD strategy is used.

2. If there is overfitting issue, i,e., num of training examples is less than 10x of total cardinality, IGD is used.

3. If l1_reg or warm_start is specified, IGD strategy is used.

4. Normal equation strategy is used for all other cases.

Conclusion

- SQL analysts want to extract insight from their data
- Pure SQL works for insights from historic data
- BQML
  - Minimal ML knowledge
  - SQL syntax
  - Pure SQL implementation → Petabyte scale ML
  - In database execution
- Try it for free: https://cloud.google.com/bigquery

Acknowledgements: Thanks to the BigQuery team.

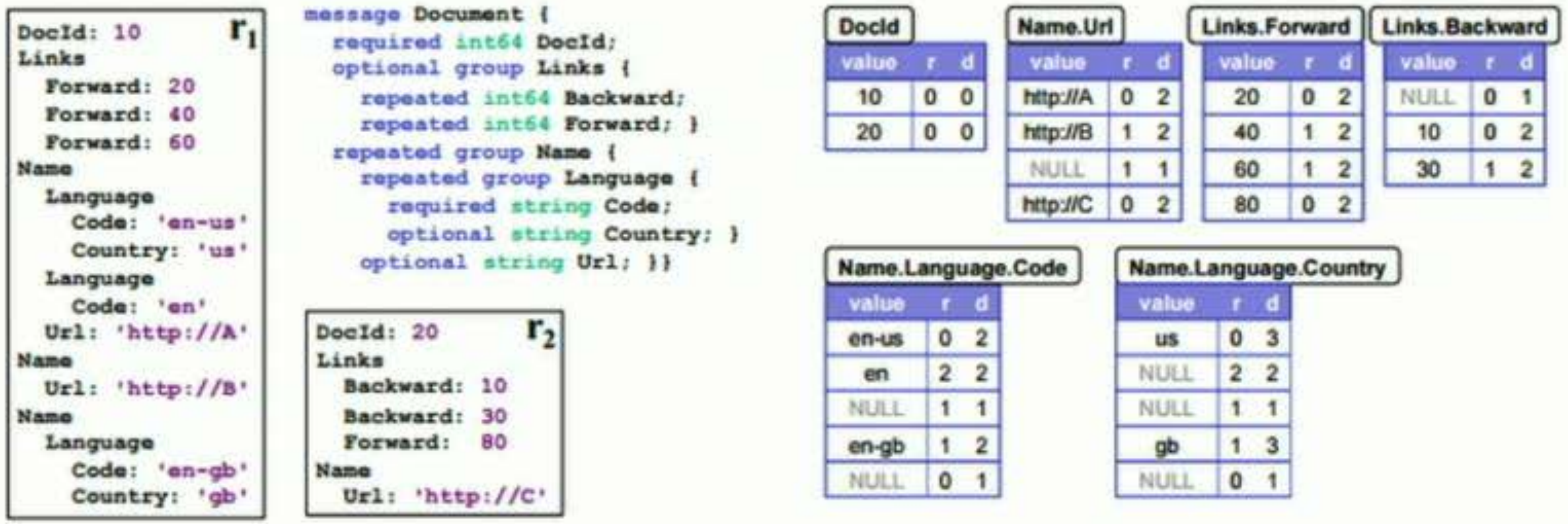
Google Cloud

BigQuery Analytics Storage Storage

Pavan Edara, Mosha Pasumansky
Software Engineer, Google
02/08/2019

Google Cloud

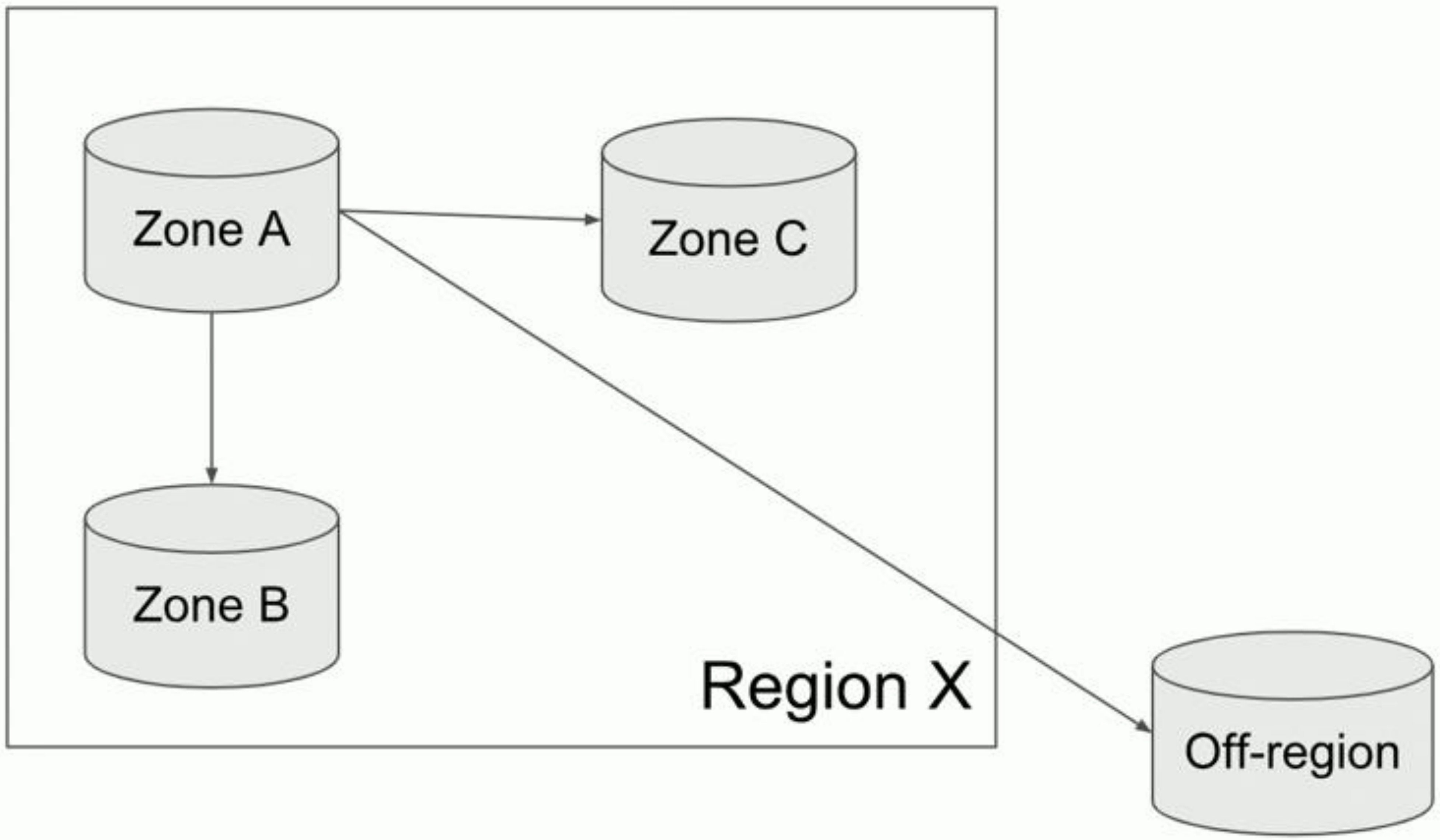
Columnar Data Storage Format: Capacitor



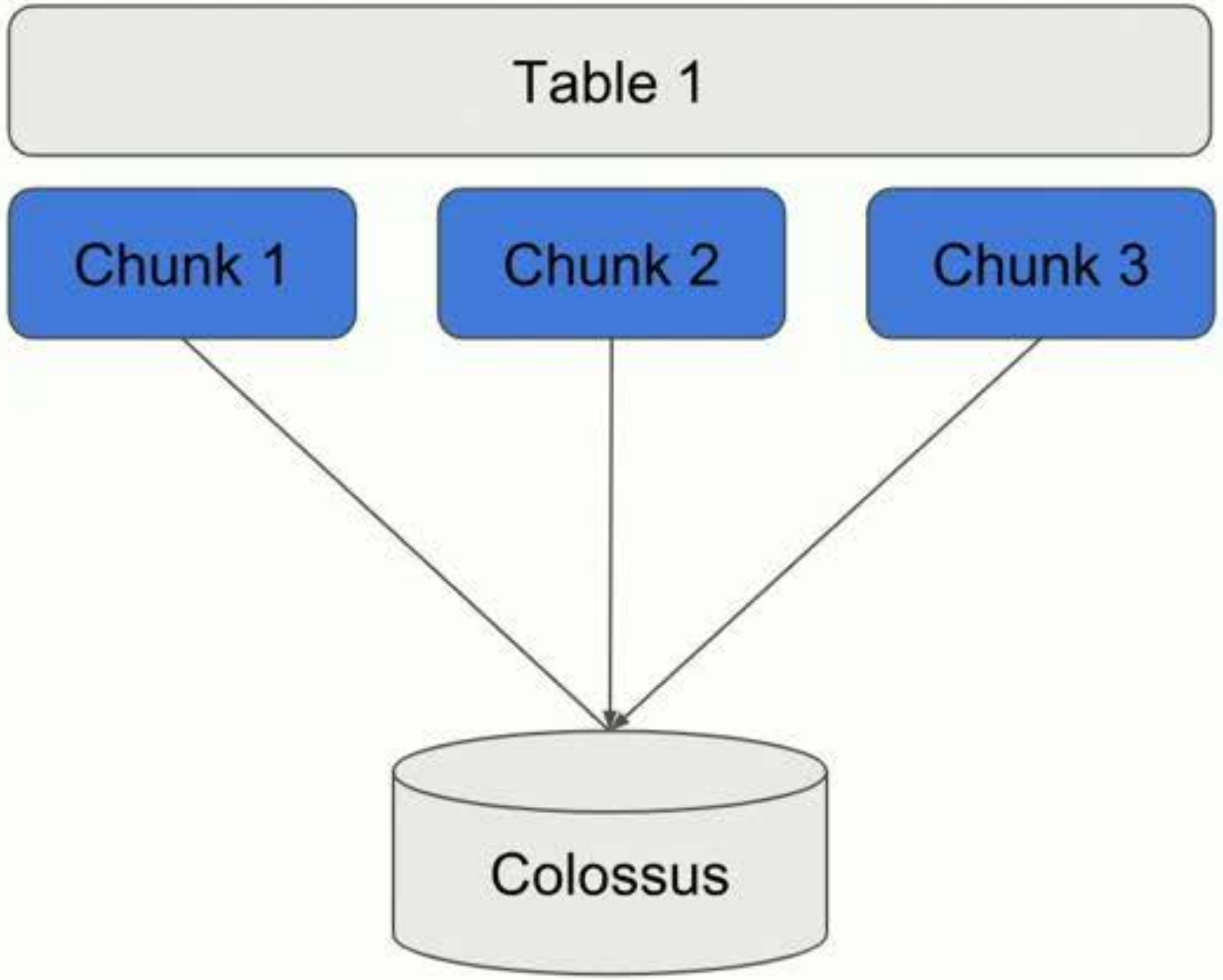
- Partial dictionary encoding
- RLE
- Bloom Filters
- Statistics
- Row Reordering
- Execution Pushdown

https://cloud.google.com/blog/products/gcp/inside-capacitor-bigquerys-next-generation-columnar-storage-format

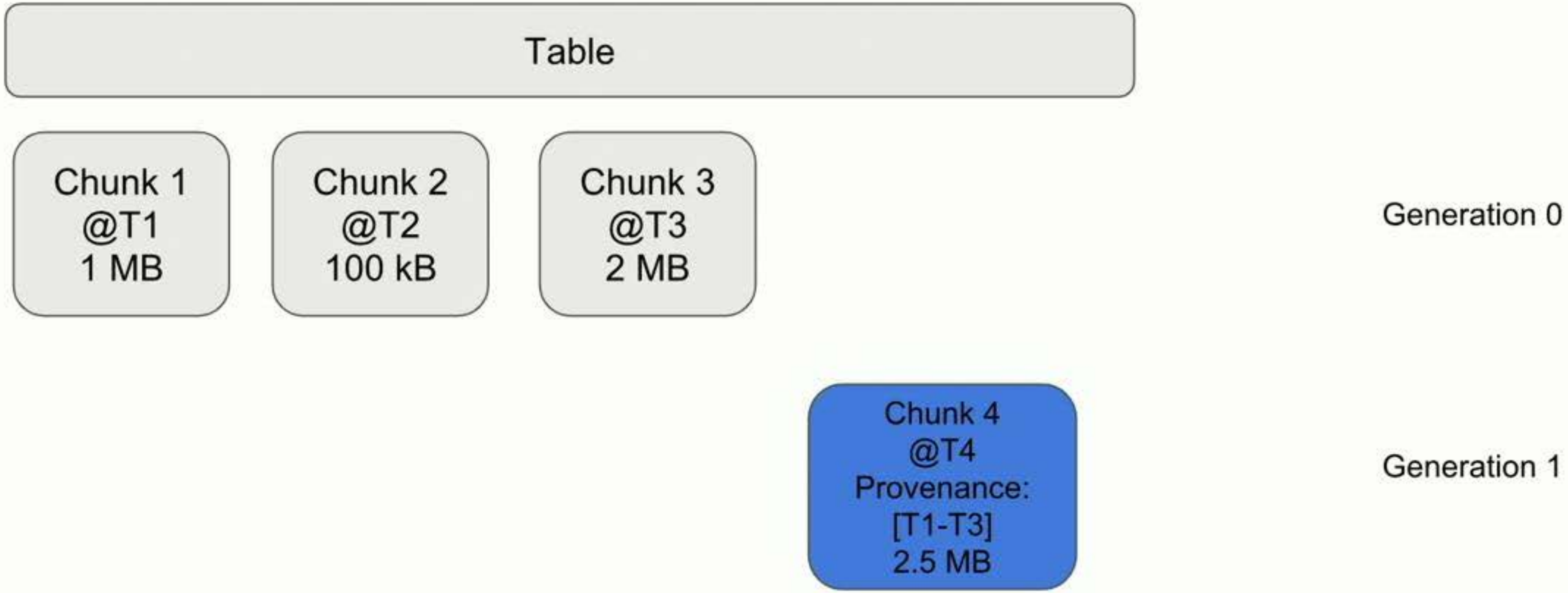
Data replication



Physical Metadata



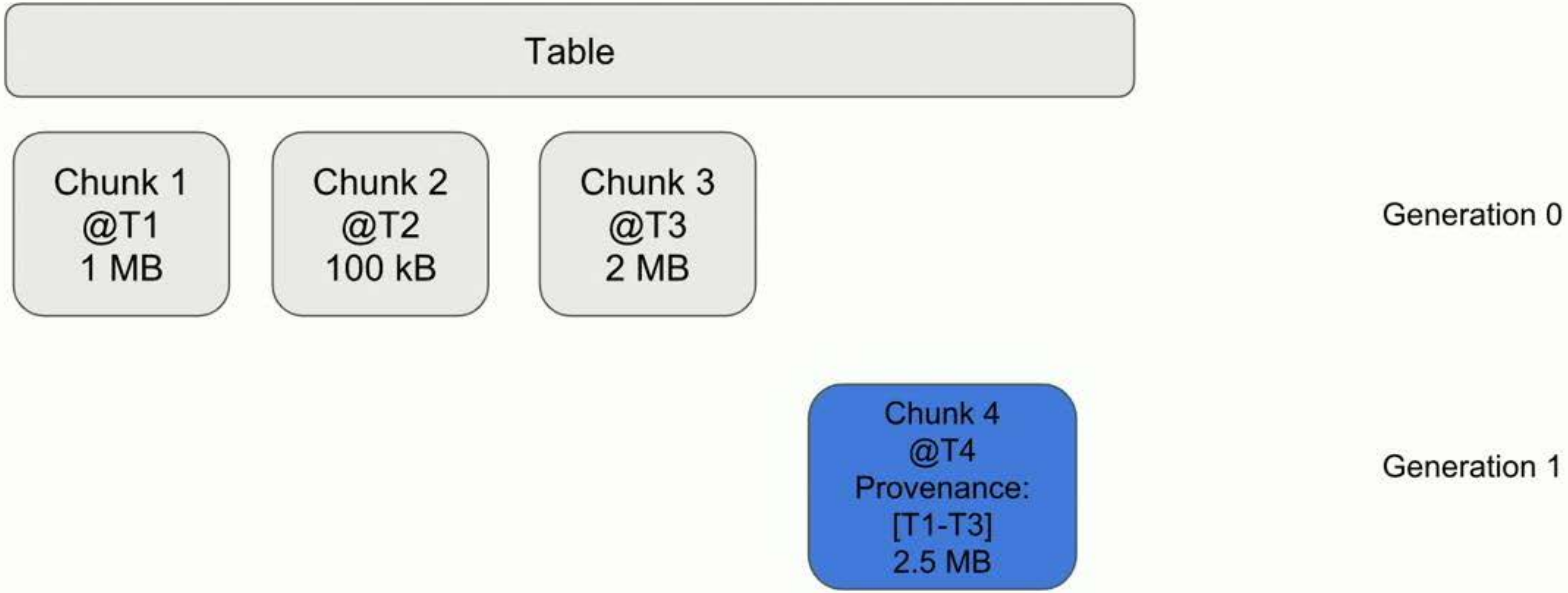
Storage Optimizer



Storage management

- Data layout
  - Columns that are often queried together - placed close to each other
  - Rows reordered to match query patterns
- File Encoding
  - Replicated: Faster
  - Reed Solomon: Smaller
- Block Sizes
  - Larger: less overhead
  - Smaller: more parallelism
- Storage Media
  - SSD: faster
  - HDD: less expensive

Storage Optimizer



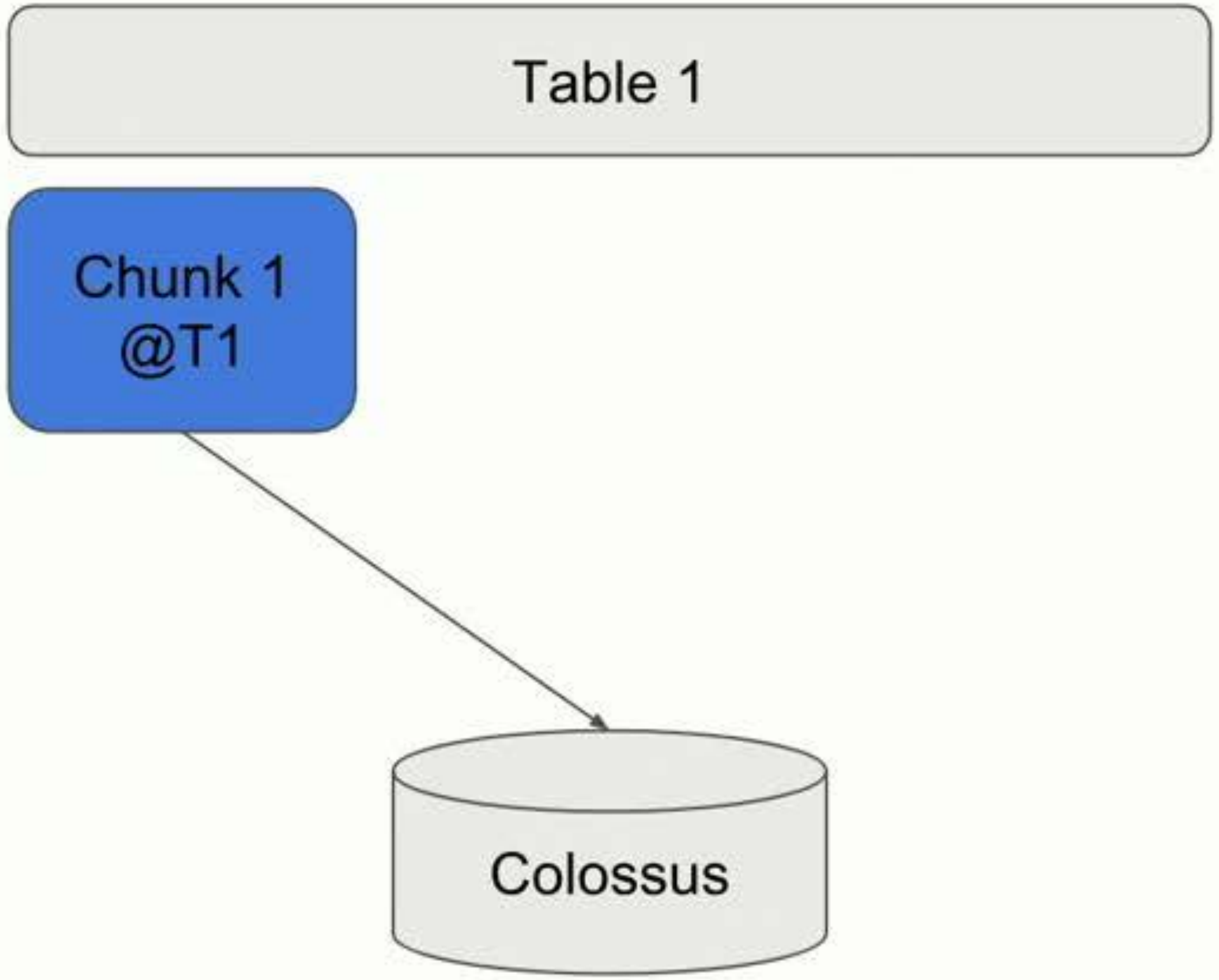
Storage management

- Data layout
  - Columns that are often queried together - placed close to each other
  - Rows reordered to match query patterns
- File Encoding
  - Replicated: Faster
  - Reed Solomon: Smaller
- Block Sizes
  - Larger: less overhead
  - Smaller: more parallelism
- Storage Media
  - SSD: faster
  - HDD: less expensive

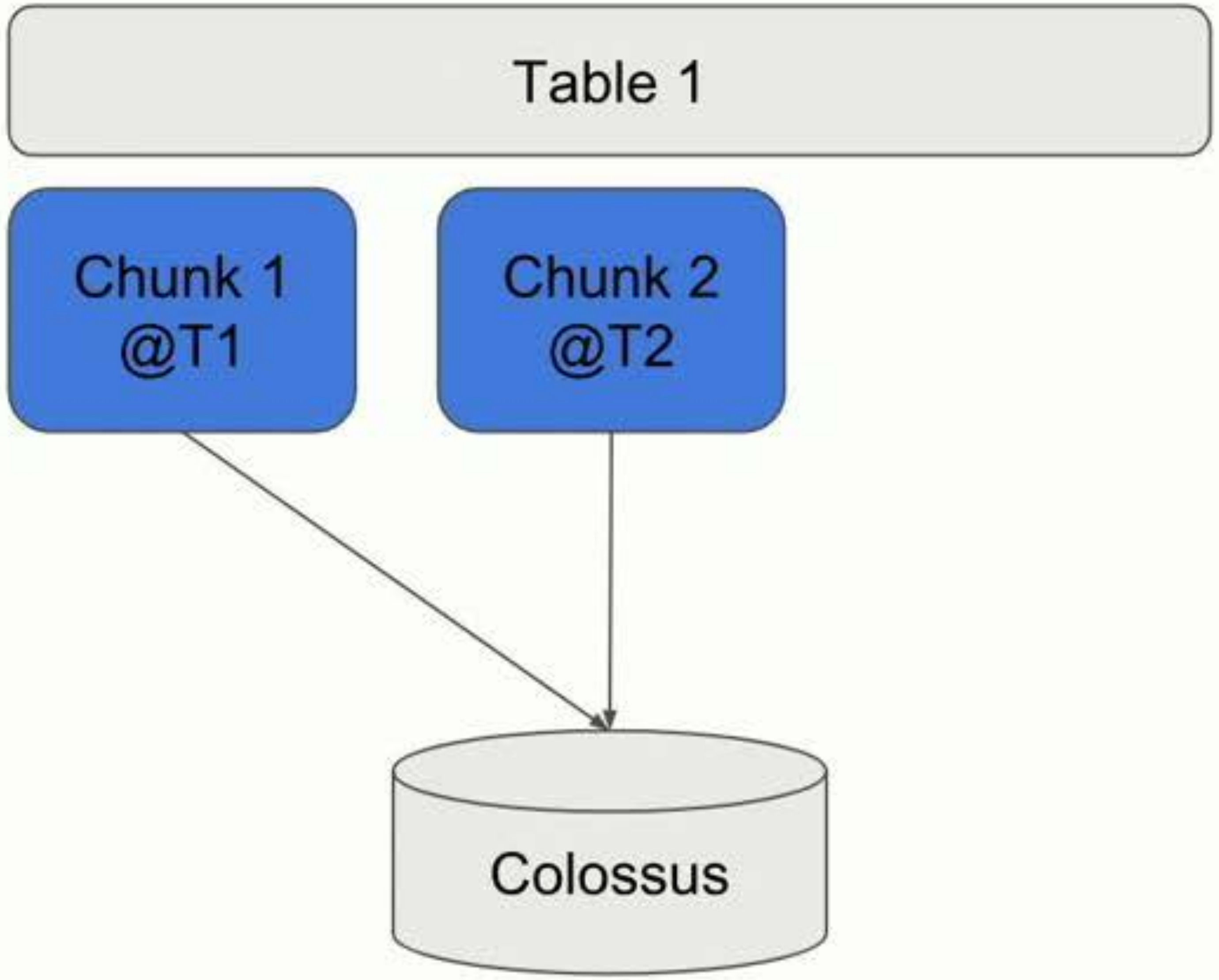
Time Travel (FOR SYSTEM_TIME AS OF)

Table 1

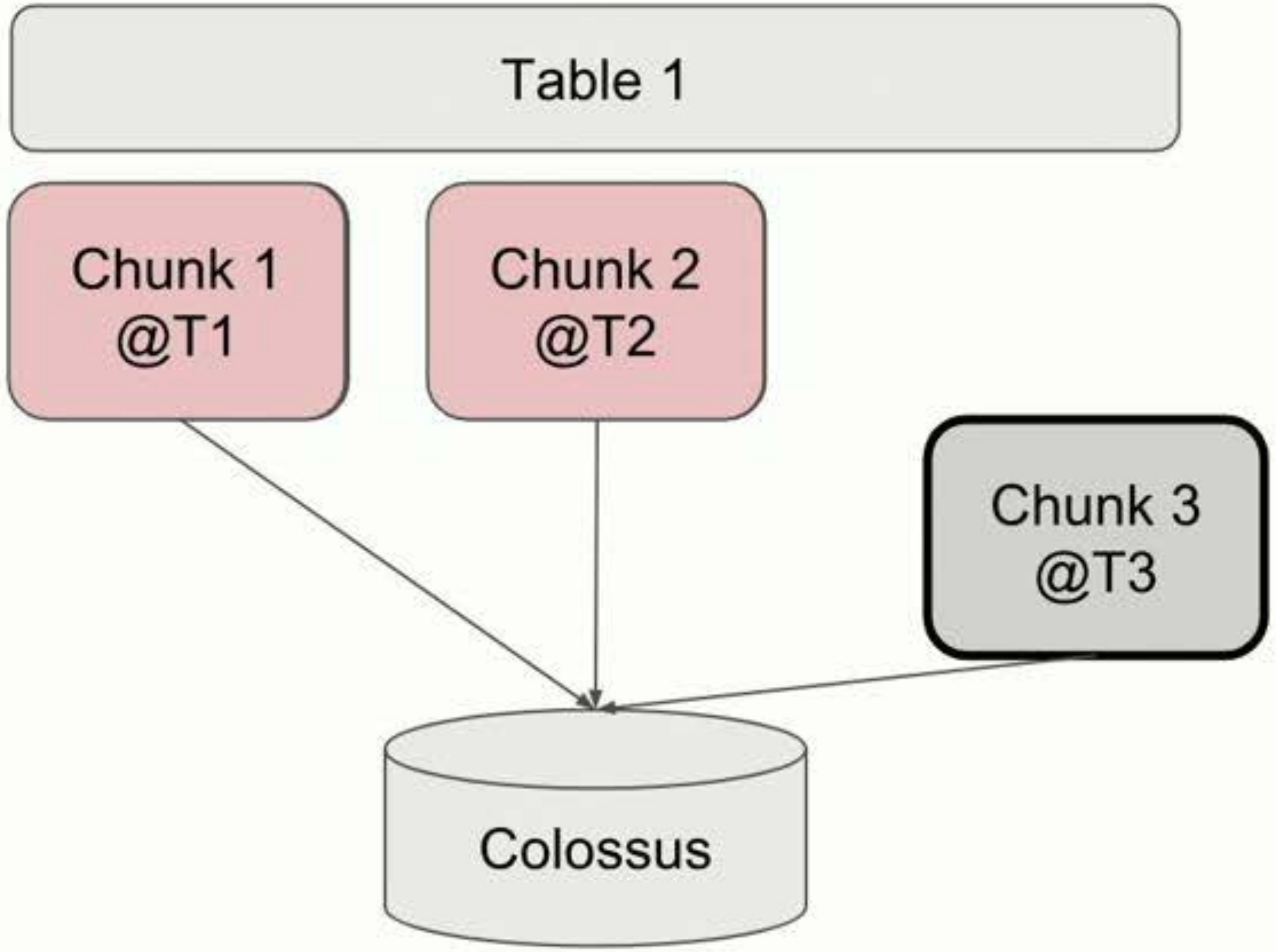
Time Travel (FOR SYSTEM_TIME AS OF)



Time Travel (FOR SYSTEM_TIME AS OF)

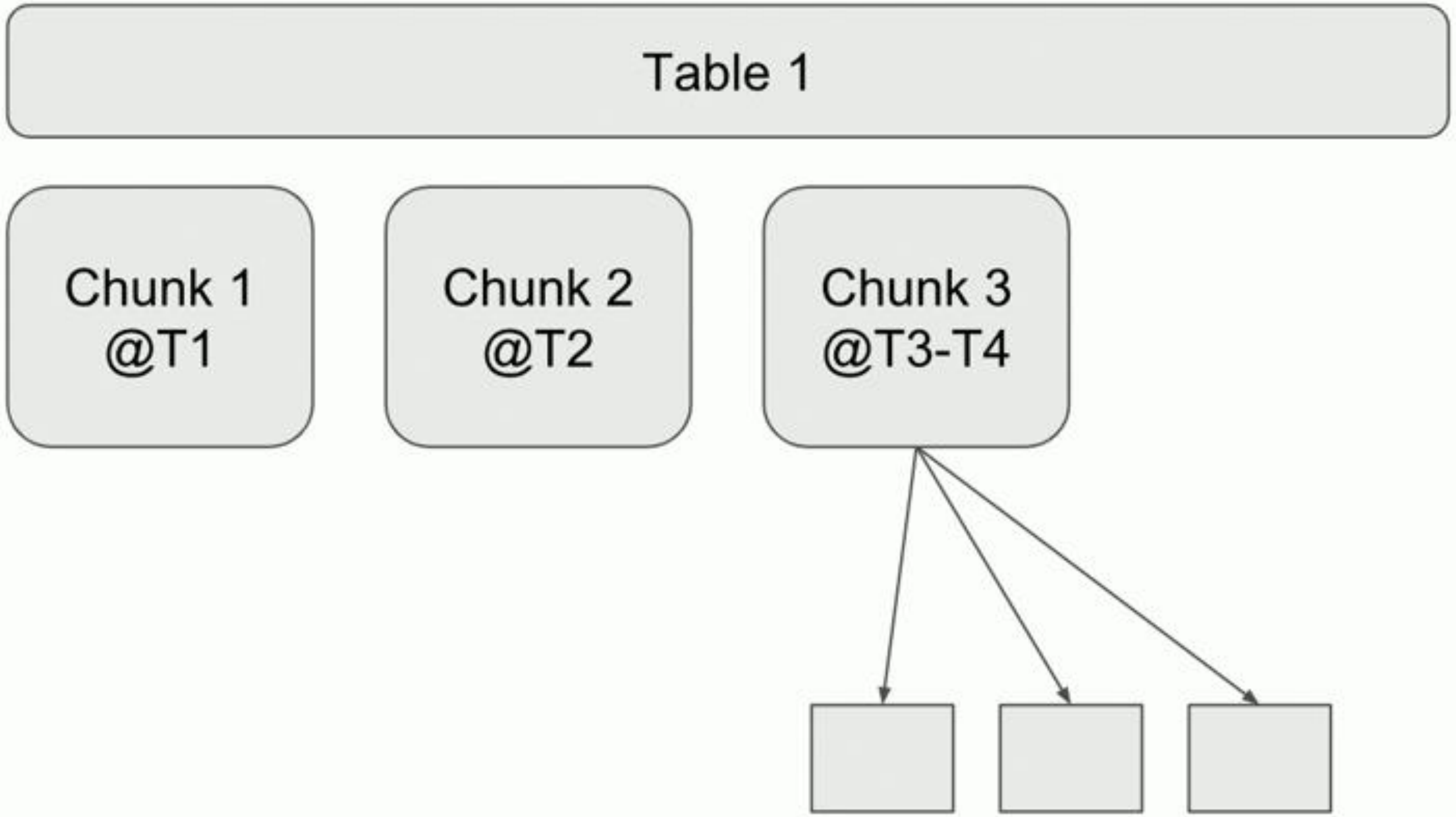


Time Travel (FOR SYSTEM_TIME AS OF)



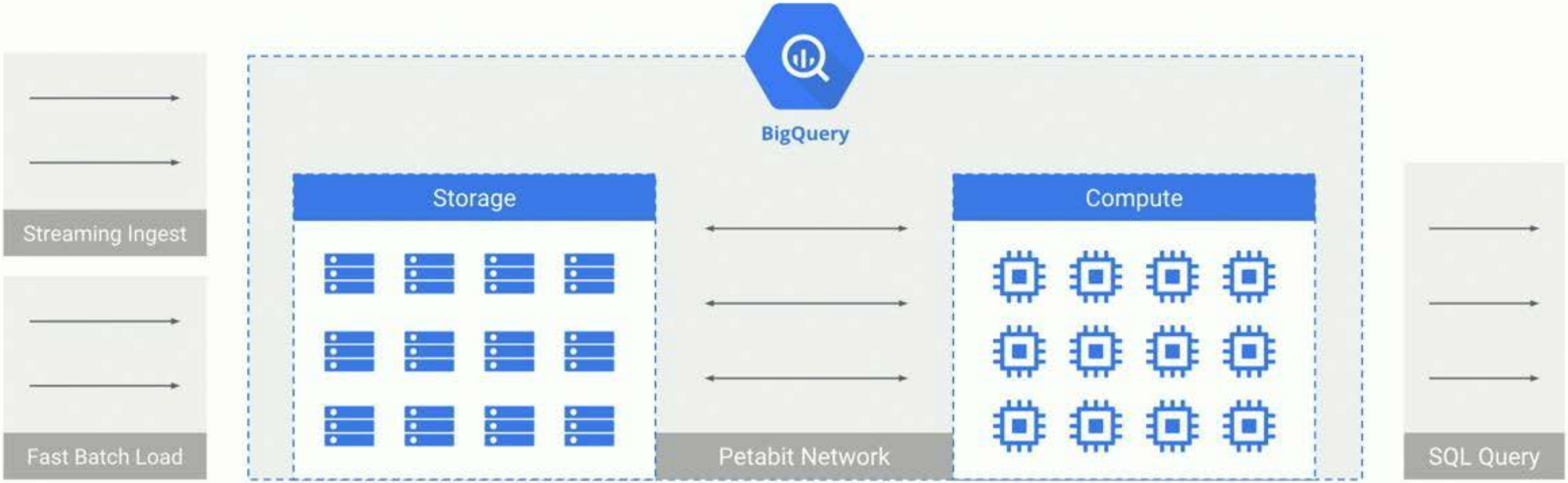
At T3: Read as of T2.5 uses Chunk 3
At T3: Read as of T1.5 uses Chunk 1

DML

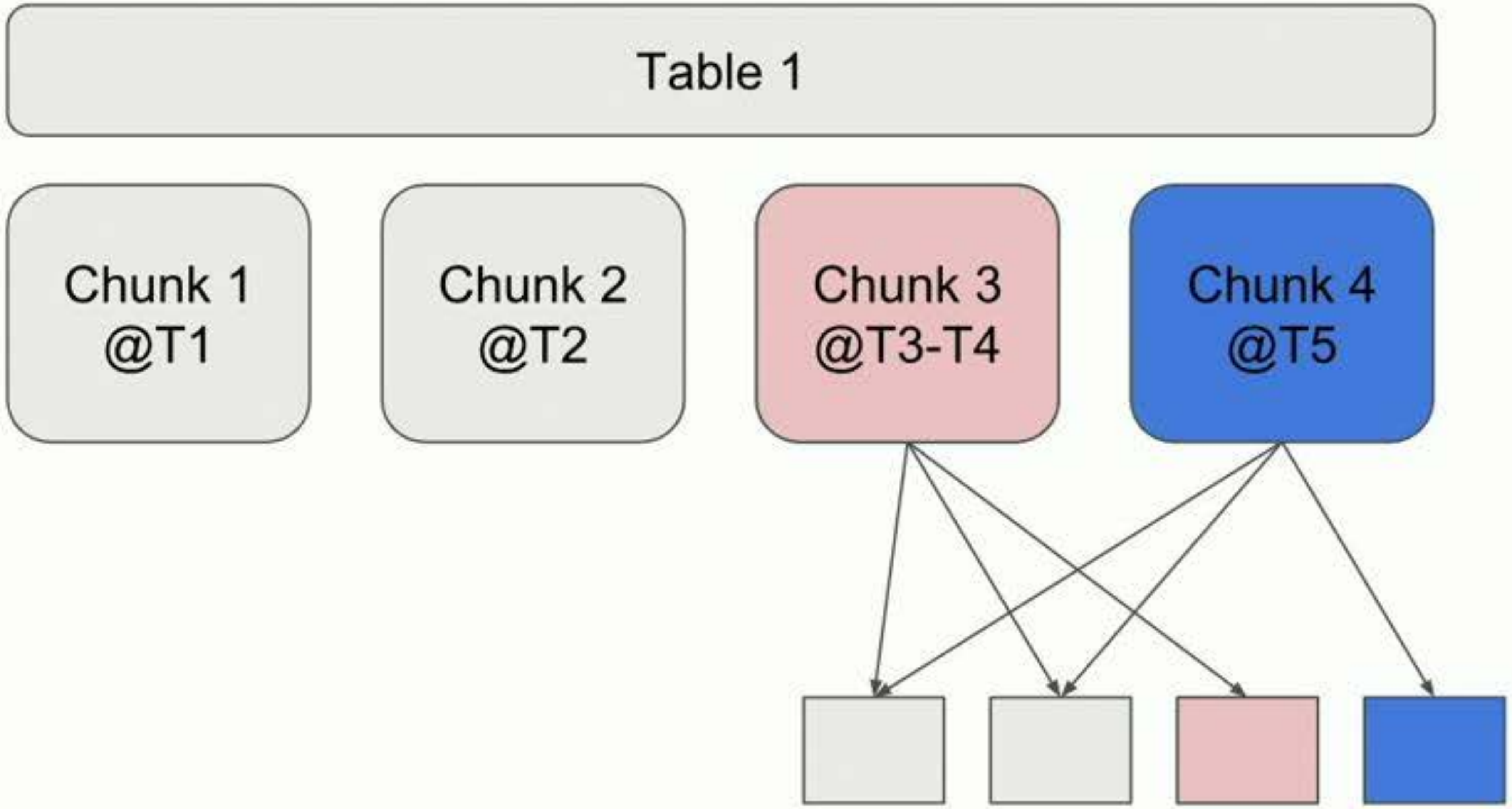


UPDATE … WHERE customerID = “1234”

Streaming Ingestion

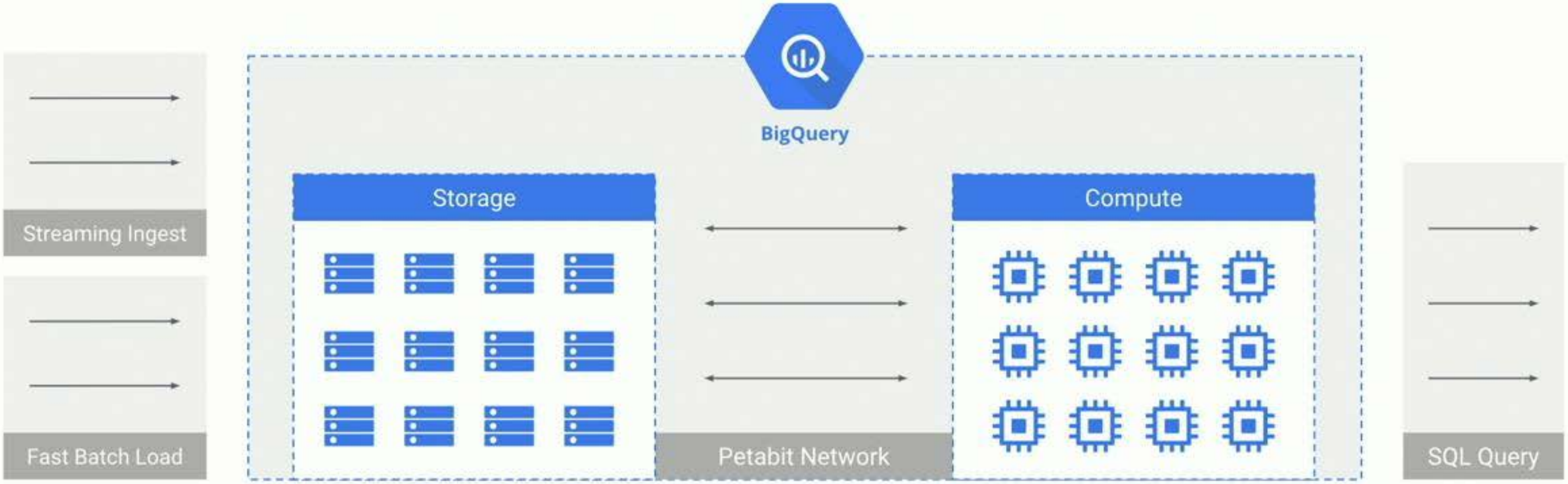


DML



UPDATE … WHERE customerID = “1234”

Streaming Ingestion



Streaming Ingestion