

# Towards Query Optimizer as a Service (QOaaS) In a Unified Lakehouse Ecosystem: Can One QO Rule Them All?



**Presenter: Yuanyuan Tian**  
**Gray Systems Lab (GSL), Microsoft**

**GSL:** Rana Alotaibi\*, Stefan Grafberger\*, Nicolas Bruno, Brian Kroth, Sergiy Matushevych, Ashvin Agrawal, Carlo Curino

**Fabric DW:** Jesus Camacho-Rodríguez, Cesar Galindo-Legaria, Milind Joshi, Milan Potocnik, Beysim Sezgin, Xiaoyu Li

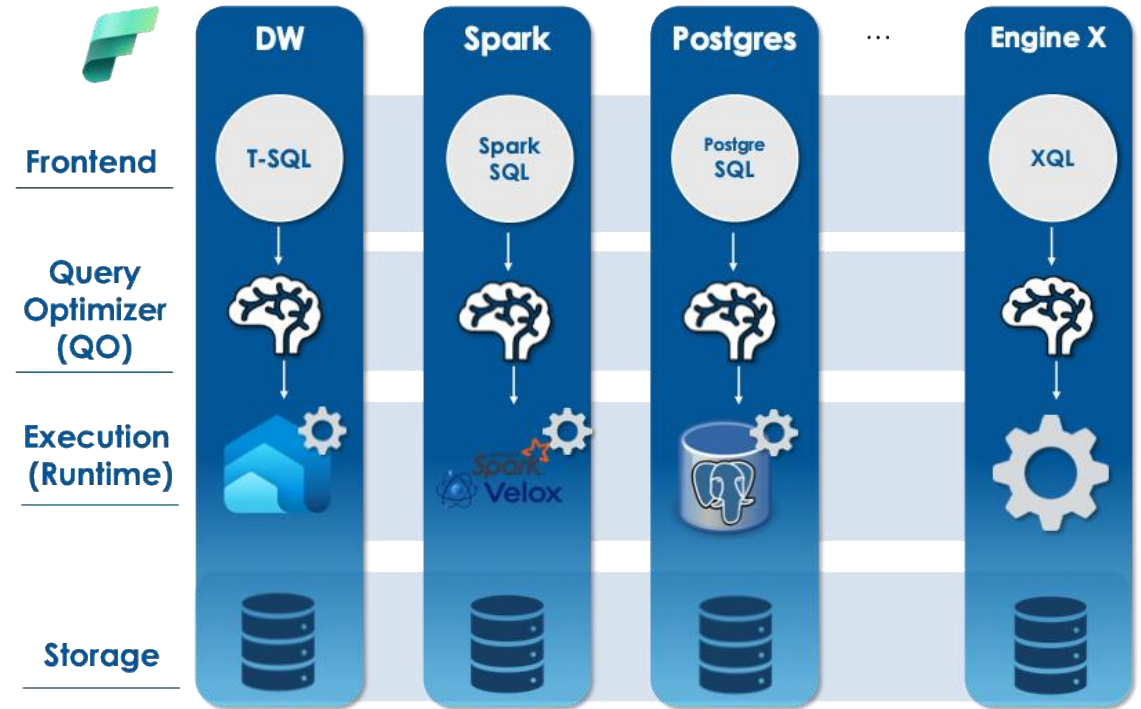
**Fabric Spark:** Mahesh Behera, Ashit Gosalia

# Industry Trends

 Demand: Fragmentation → Convergence

## Microsoft Fabric

- Shared Data on Lake
- Shared Compute Resource
- Shared Governance Experience

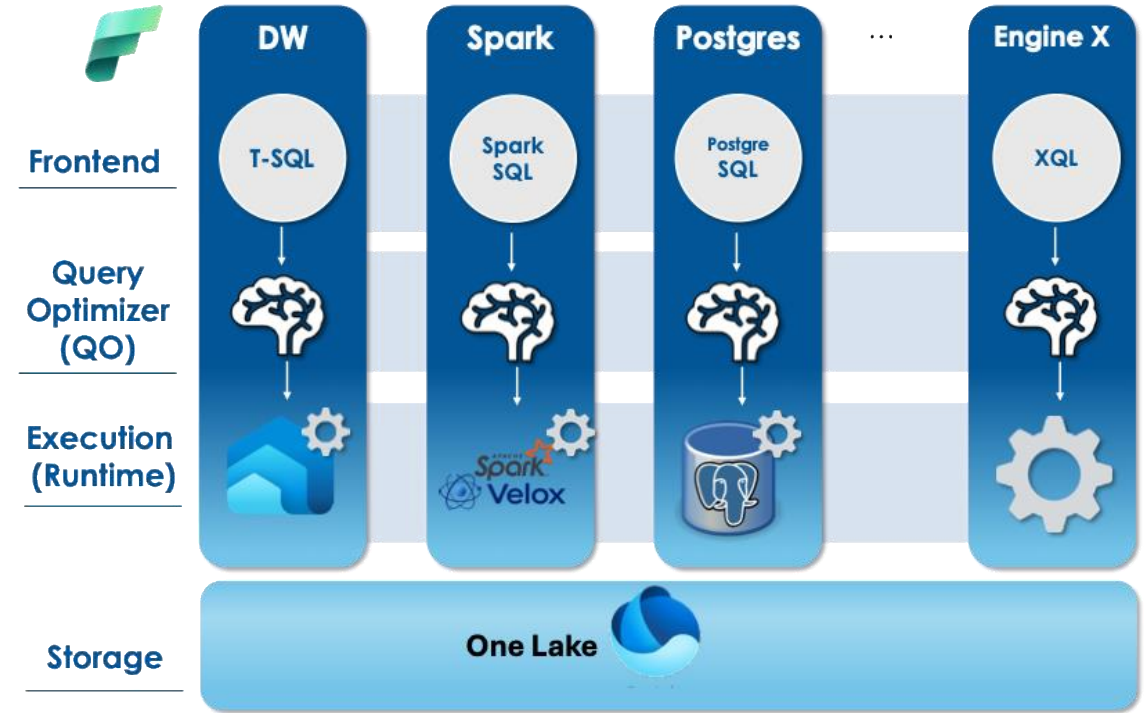


# Industry Trends

 Demand: Fragmentation → Convergence

Microsoft Fabric

- Shared Data on Lake
- Shared Compute Resource
- Shared Governance Experience



# Industry Trends

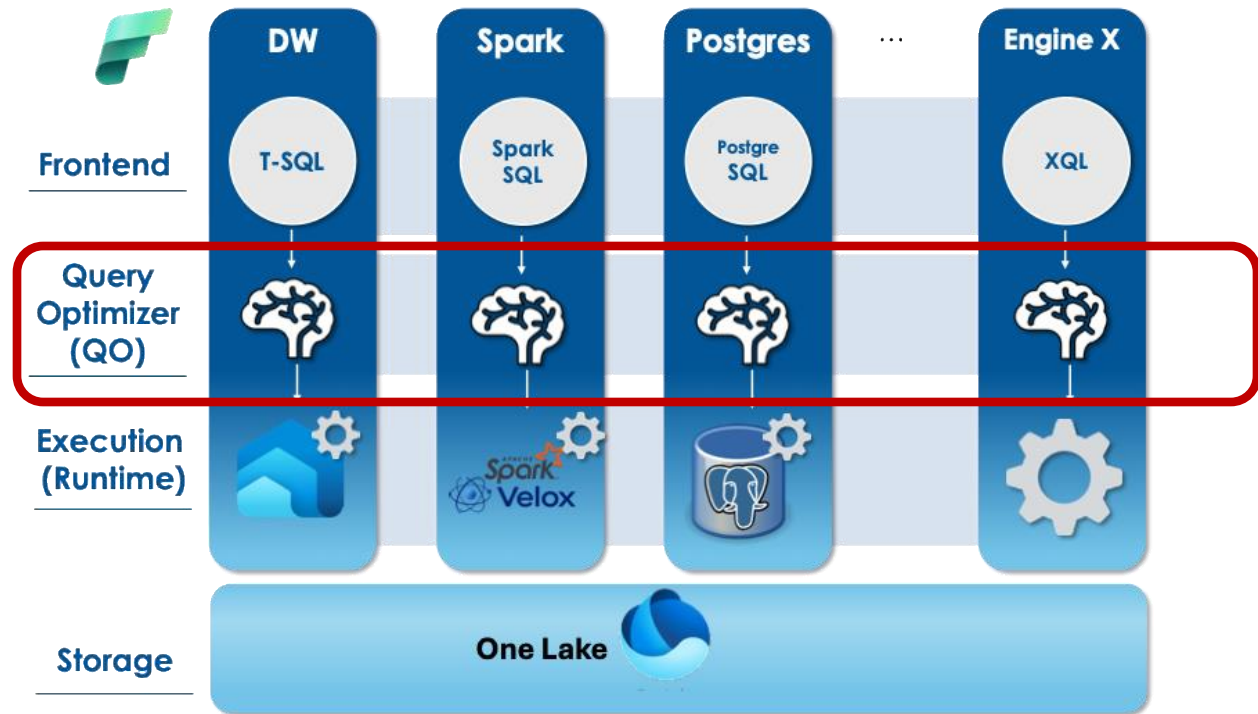
## Demand: Fragmentation → Convergence

### Microsoft Fabric

- Shared Data on Lake
- Shared Compute Resource
- Shared Governance Experience

## Architecture: Monolithic → Composable

- Cloud DB: separation of storage from compute
- Open standards
  - Parquet, Arrow, Substrait
- OSS system-building libraries
  - Calcite, Orca, Velox, Datafusion



# Industry Trends

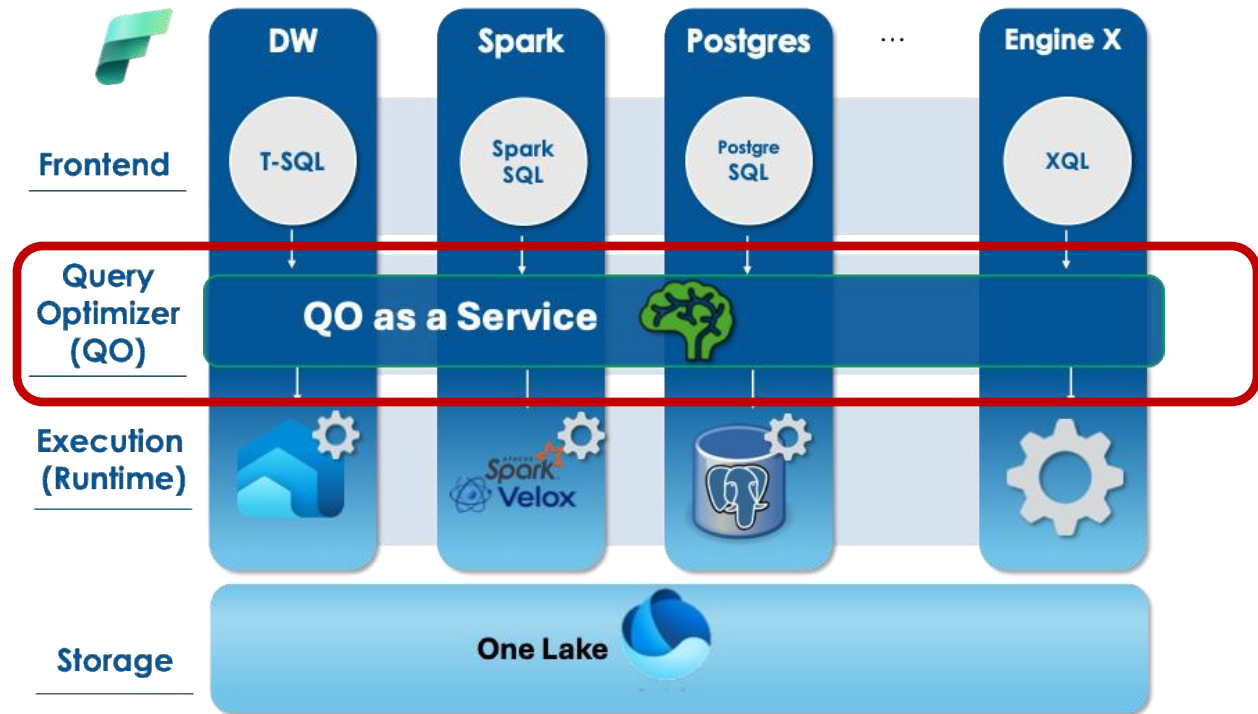
 Demand: Fragmentation → Convergence

Microsoft Fabric

- Shared Data on Lake
- Shared Compute Resource
- Shared Governance Experience

 Architecture: Monolithic → Composable

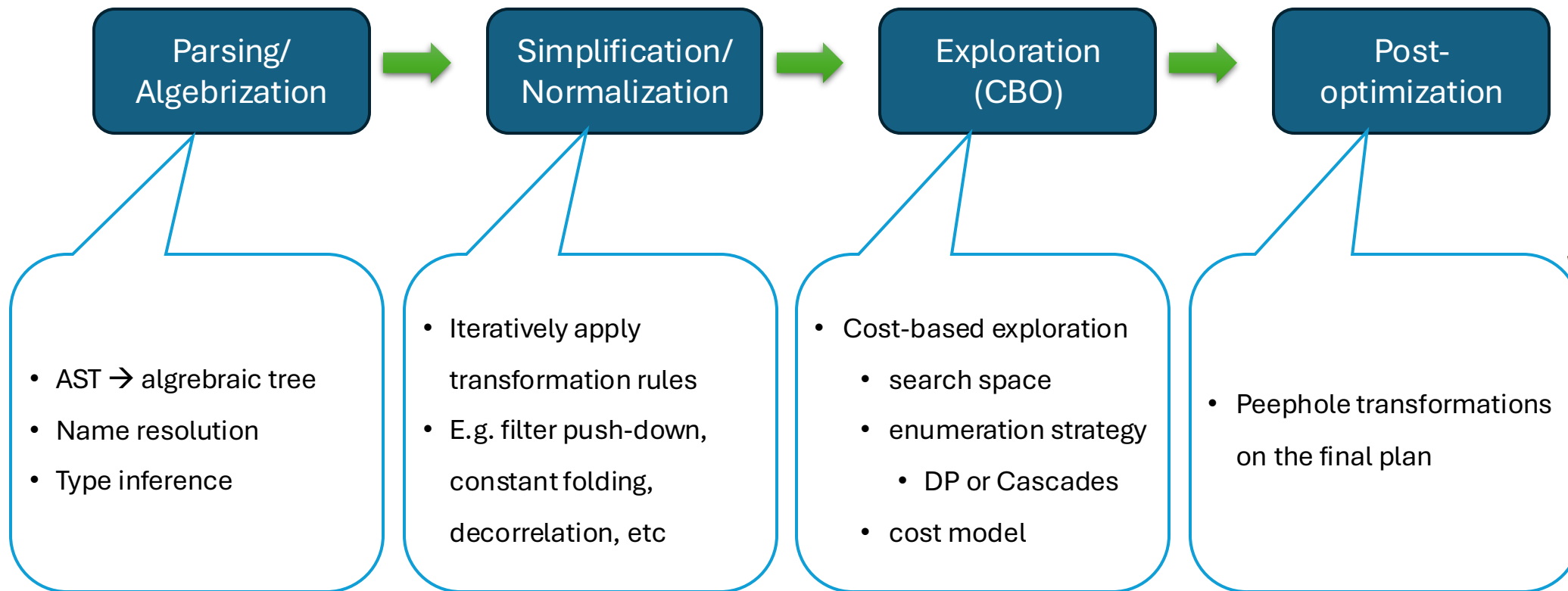
- Cloud DB: separation of storage from compute
- Open standards
  - Parquet, Arrow, Substrait
- OSS system-building libraries
  - Calcite, Orca, Velox, Datafusion



# QO Status Quo: Reinventing Wheels

A proliferation of analytical engines with their own QOs following similar patterns

- Same relational algebra, similar search spaces, and same stages



# QOaaS

- Independent QO service interacting with multiple engines over RPC

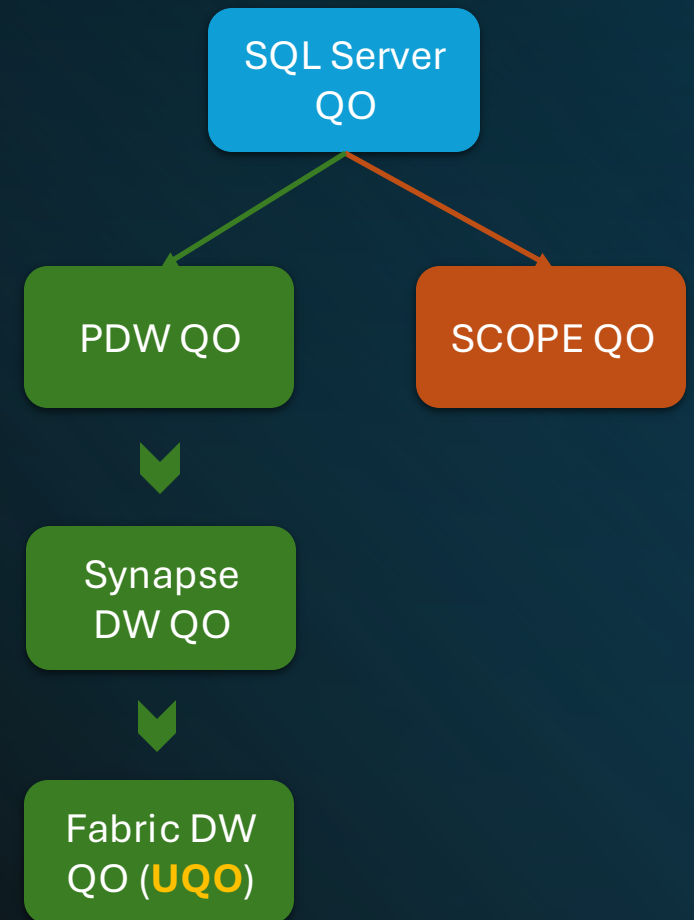
Focus: for *analytical* engines in a *unified Lakehouse* ecosystem, e.g. Microsoft Fabric

Features	Custom QO	QO as a Library (Calcite, Orca)
Innovation speed	✗	✓
Engineering efficiency	✗	✓
New engine time-to-market	✗	✓



# Steps Towards QOaaS

- Building on our own experience
  - Developing Calcite
  - Evolving Cascades framework within Microsoft
- Initial focus
  - Two engines: DW and Spark on Fabric Ecosystem
  - Adapting UQO (Fabric DW QO) to QOaaS
- Key Challenges:
  - **CH1: Exchanging plans in and out of QO**
  - **CH2: Adapting UQO for different engines**
  - **CH3: Adjusting the cost model**



**QO Evolution in Microsoft**



# CH1: Standardizing Plan Specification

- **Substrait**: open-source, *cross-language* plan specification for relational algebra
  - Various serialization formats
  - Extensibility for custom operations
  - Ecosystem for libraries and toolings
- Making Substrait as the cross-engine plan specification on Fabric
  - Ongoing collaborative effort across GSL, DW, Spark, and Power BI
  - Current coverage: TPC-H, TPC-DS, internal benchmarking workloads



# CH2: Can UQO optimize Spark Queries?

---

## Spark QO

- Mostly non-CBO
- CBO only applies to join ordering and broadcast-vs-shuffle join decision

## UQO

- Full-stack Cascades framework with 255 CBO rules
- Sophisticated cost model

### Naïve replacement won't work!

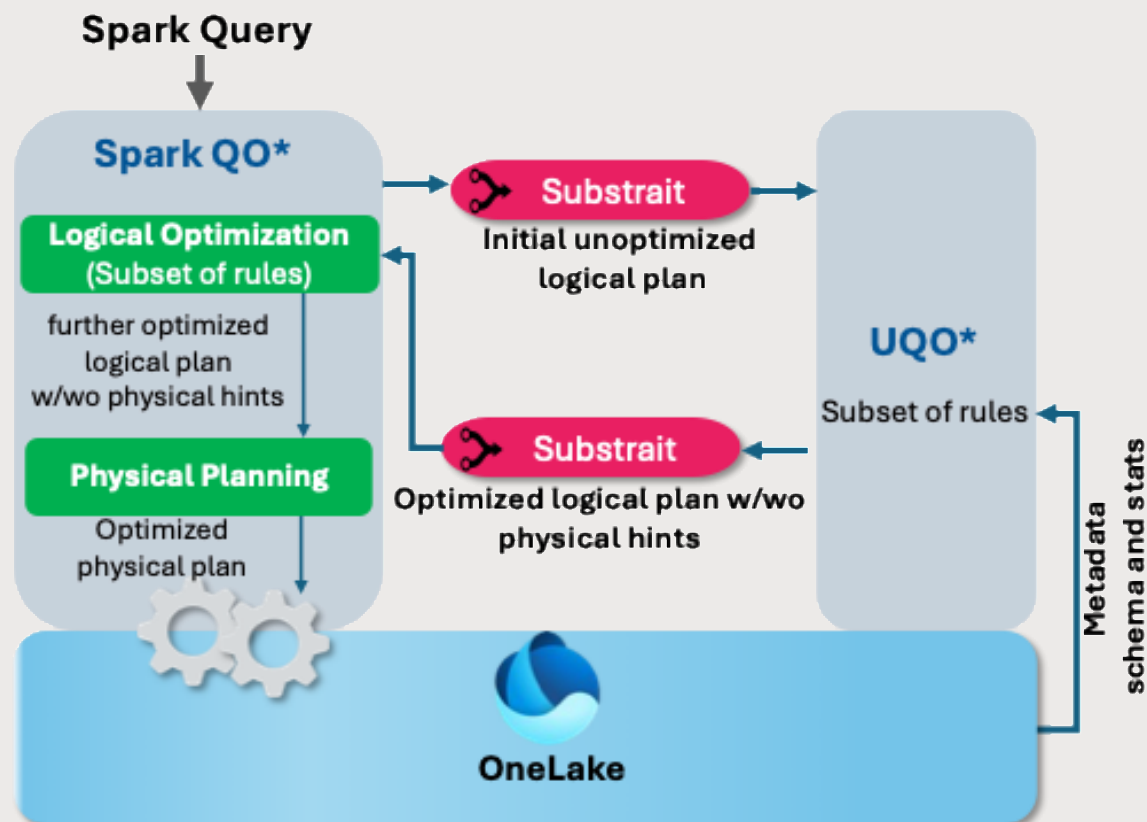
#### • Physical Operator Gaps

- Some Fabric DW physical operators are unsupported in Spark
  - Example: merge/hash-based union

#### • Feature Support Disparities

- UQO cannot fully exploit Spark-specific features
  - Example: Hive-style partitioning

# A Simple QOaaS Prototype



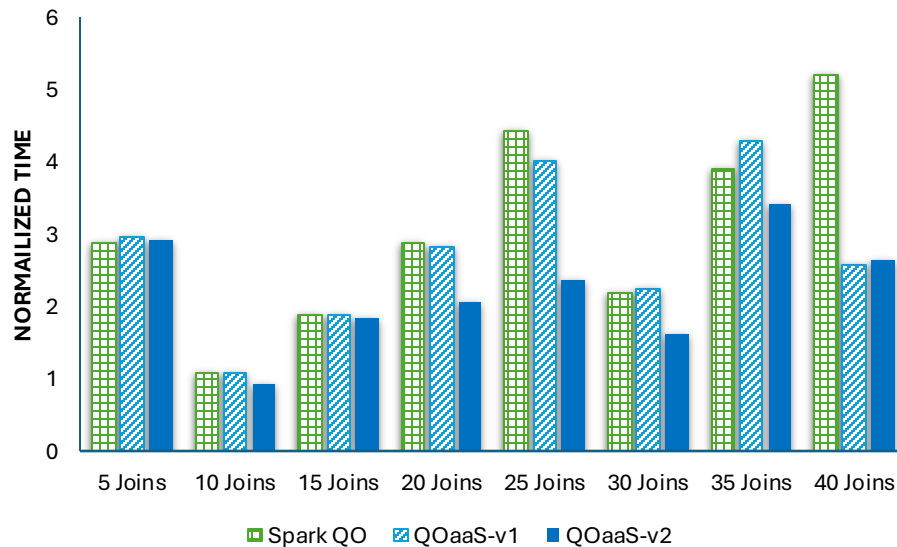
- **UQO\***
  - Not generating unsupported operators in Spark
- **Spark QO\***
  - Adding Spark specific optimization rules lacking in UQO

	UQO*	Spark QO*
QOaaS-v1	logical optimization	further optimization + physical implementation
QOaaS-v2	logical + physical optimization	further optimization + physical implementation based on hints from UQO*

# Performance Study

## MSSales Workload

- 627 tables on OneLake (5TB, delta parquet)
- Highly templated queries, join heavy



## Takeaway

- UQO-based QOaaS looks promising!
- QOaaS-v2 performs better than QOaaS-v1

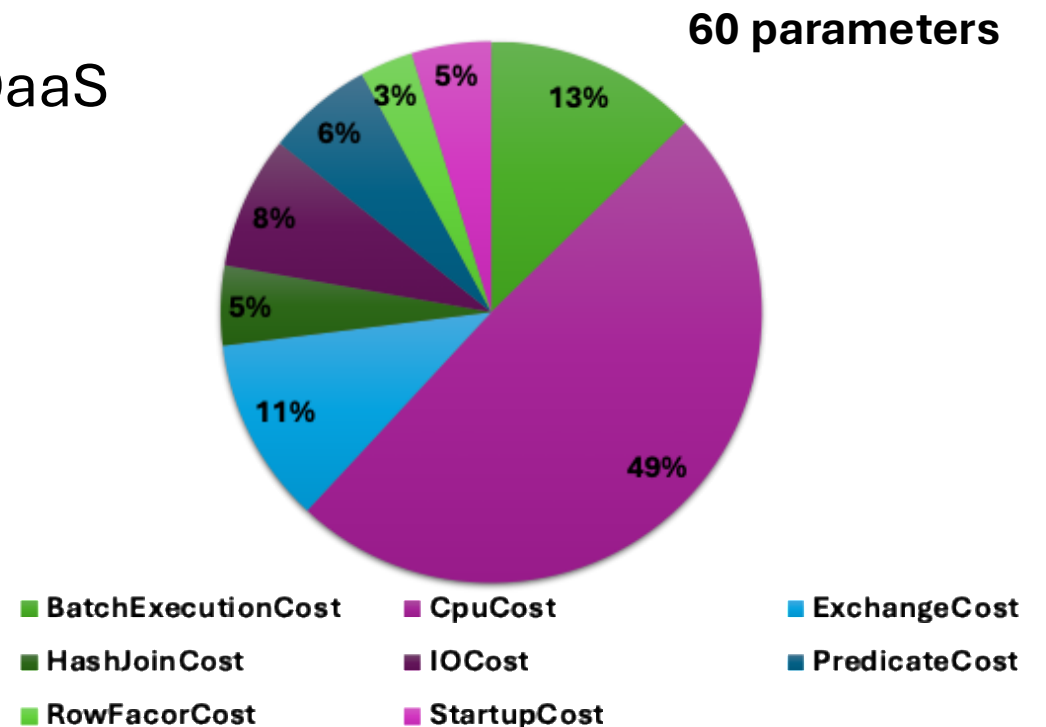
## TPC-H SF1000 (1TB)

- QOaaS-v2 is comparable to SparkQO
  - Average diff <6%
- Q5 is 1.5x slow
  - Not fully utilizing Bloom filters

- Adding optimizations retroactively is suboptimal, all optimization opportunities should be explored!

# CH3: Recalibrating and Tuning the Cost Model

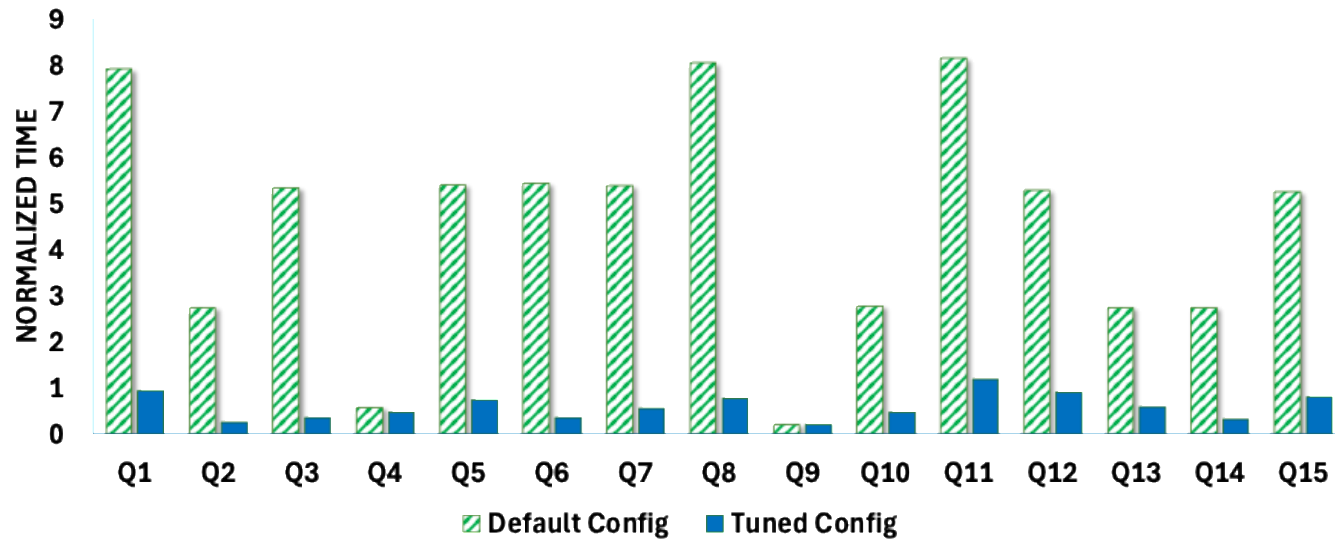
- A fixed cost model is unlikely to work for QOaaS
- 1<sup>st</sup> attempt: changing cost model without rewrite
  - Recalibrating and tuning *constant parameters* in UQO's cost formula
  - MLOS [1]: OSS ML-powered tuner



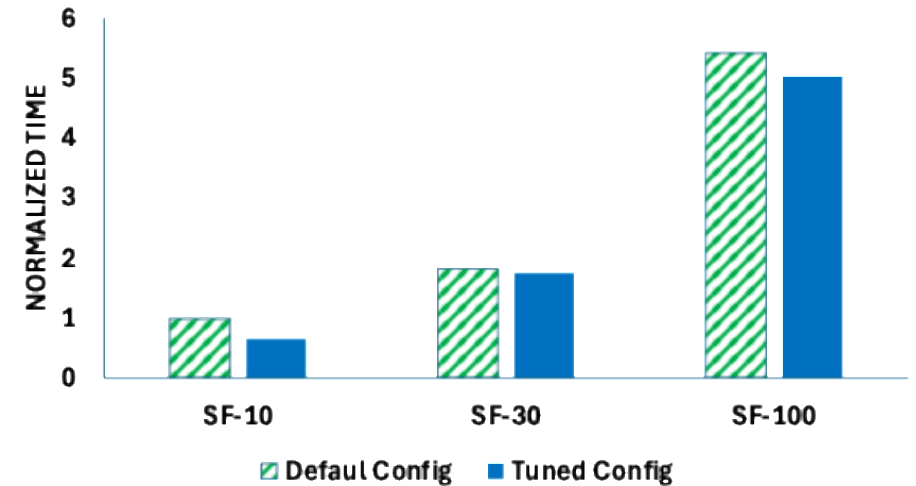
[1] MLOS. <https://github.com/microsoft/MLOS>

# Performance Study

## MSSales Workload



## TPC-H



## Observation

- Really encouraging results for cost model tuning!
- Tuned parameters are not transferrable!
  - Overfitting to a workload → a benchmark workload with coverage of all operators
  - Interplay with cardinality estimation errors → injecting true cardinality leveraging prior work [2]

# Key Lessons Learned

*Time for a new design?*



A standard plan specification is essential for QOaaS



QOaaS should explore all possible optimization opportunities



QOaaS needs to generate engine-specific costs

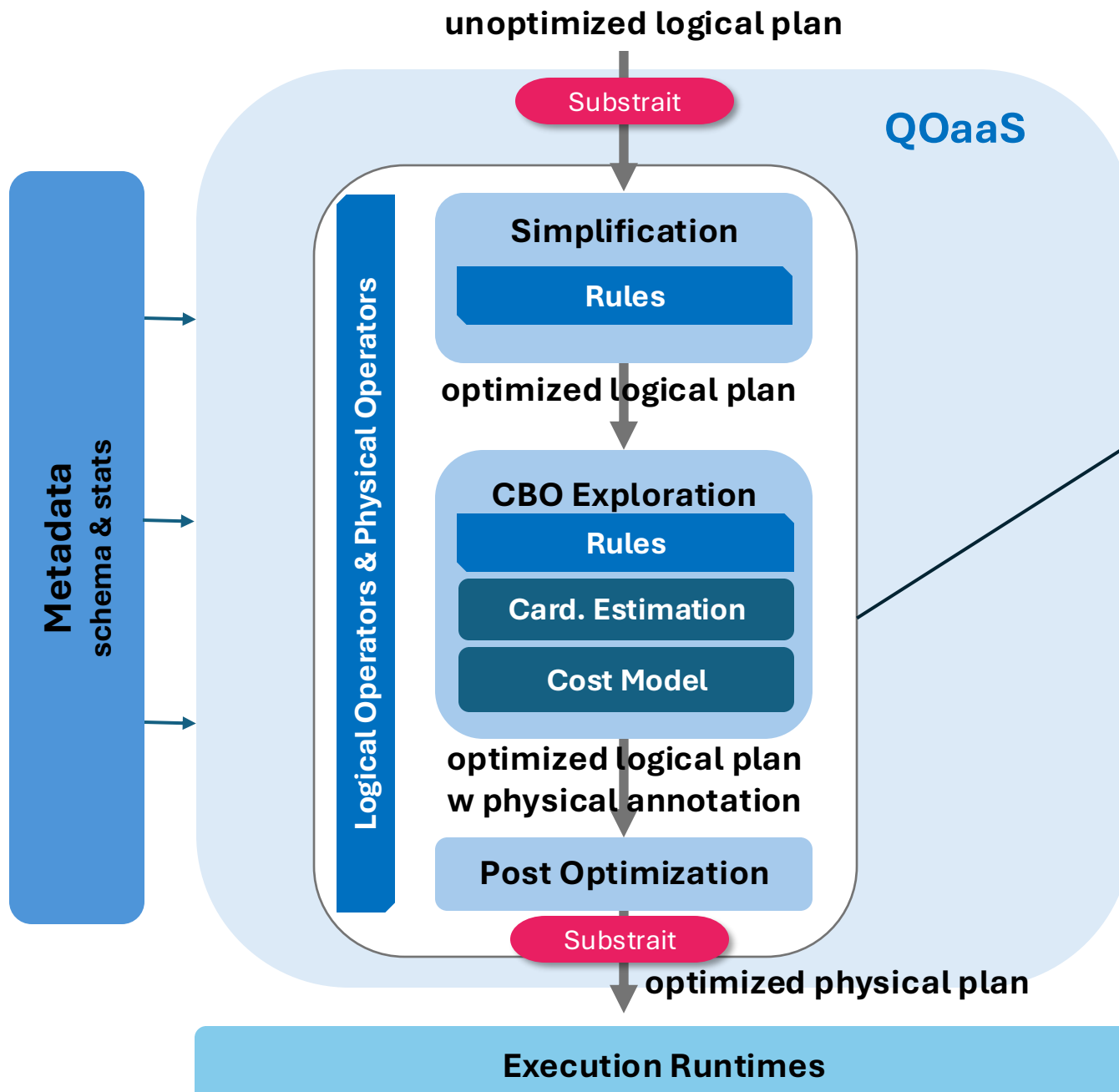


QOaaS should allow instance-based optimization (e.g. ML-based QO enhancement)



Fiddling with a production-level customized QO for QOaaS requires significant engineering effort

# A QOaaS Proposal

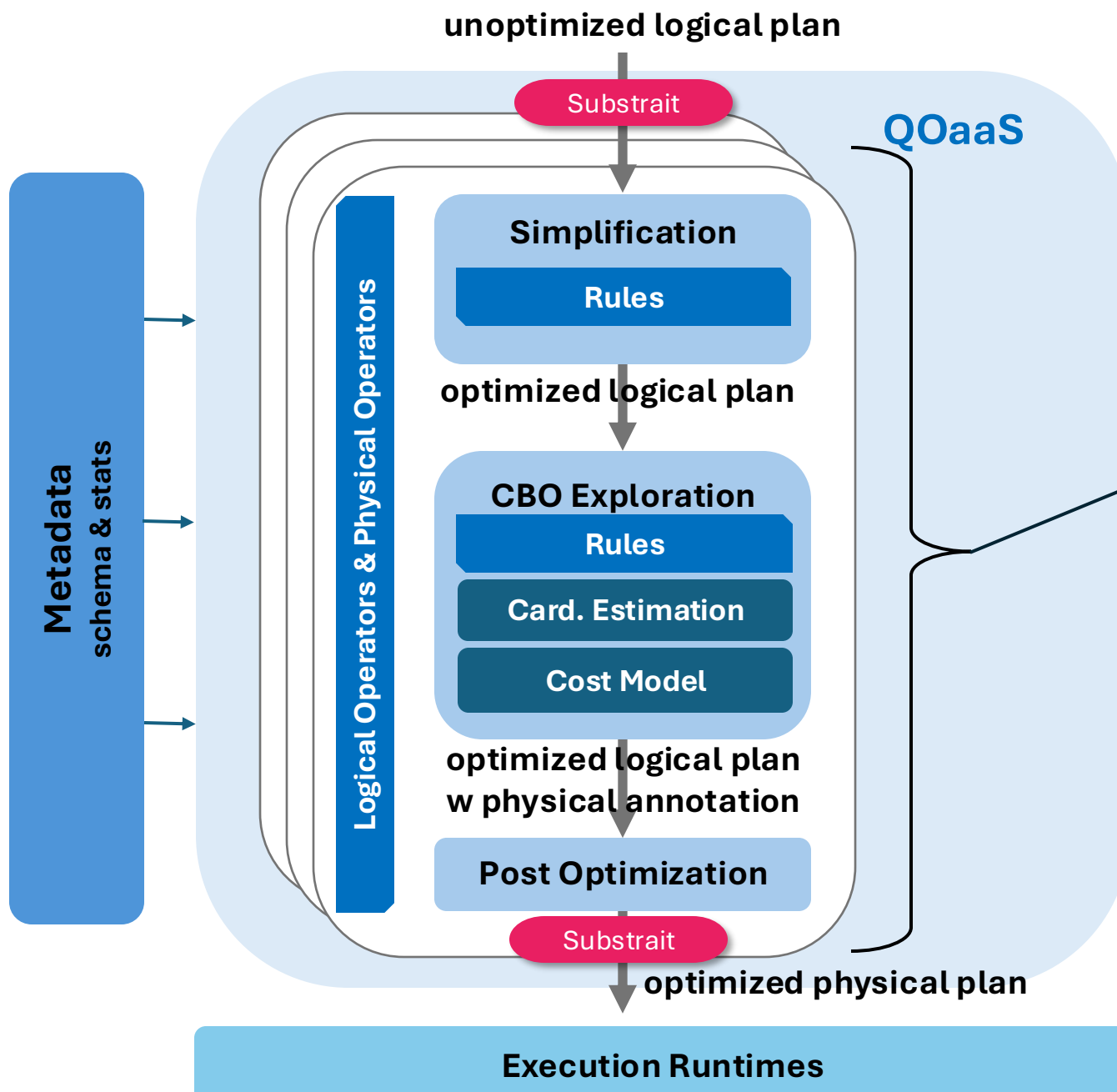


## Core Component

- Standard plan specification
- Modular, extensible components
- Adding *engines-property* to operators and rules
- New cross-engine data exchange operator
- Engines-property is enforced during optimization
- Cost model takes target engine as an additional input



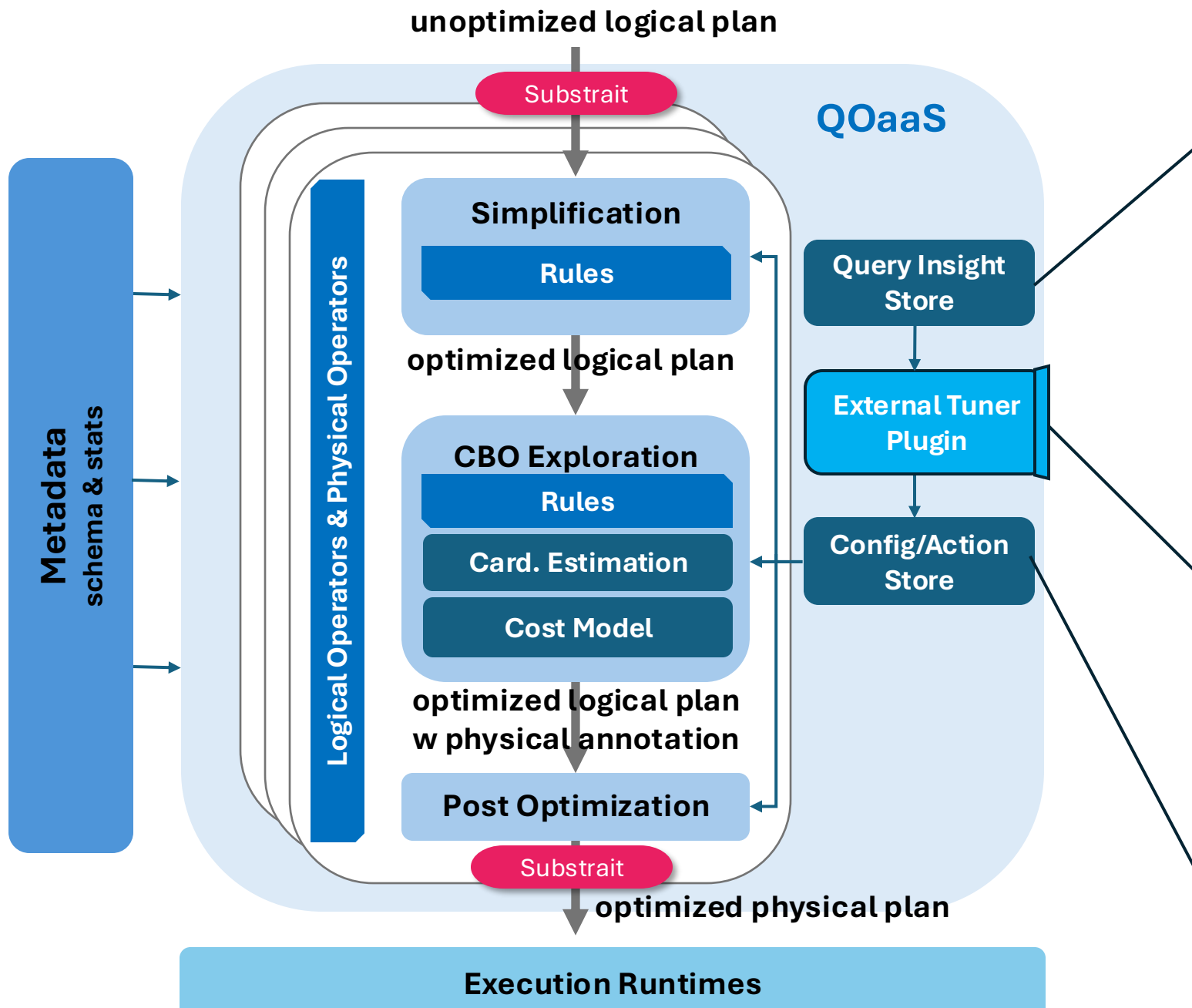
# A QOaaS Proposal



## Servification

- Dedicated resources for QOaaS
- Elastic scale up and out independently
- Can run different versions of QO simultaneously
- Easy deployment and testing

# A QOaaS Proposal



## Observability


- Automatically capturing queries, plans, and runtime statistics

## Pluggable External Service

- e.g. MV/index selection, ML-based QO enhancement
- APIs to read info from Query Insight Store
- APIs to store information into the Config/Action Store

## Feedback to QO

- Enhancement from the stored info



# Open Discussion and Debate

Is QOaaS a fantasy?

Will it work?



Can One QO Rule Them All?