Integrating Distributed SQL Query Engines with Object-Based Computational Storage

Junghyun Ryu, Soon Hwang, Junhyeok Park, Seonghoon Ahn, Youngjae Kim
Sogang University, Seoul

JeoungAhn Park, Jeongjin Lee, Jinna Yang, Soonyeal Yang, Jungki Noh, Woosuk Chung, Hoshik Kim
SK hynix Inc.

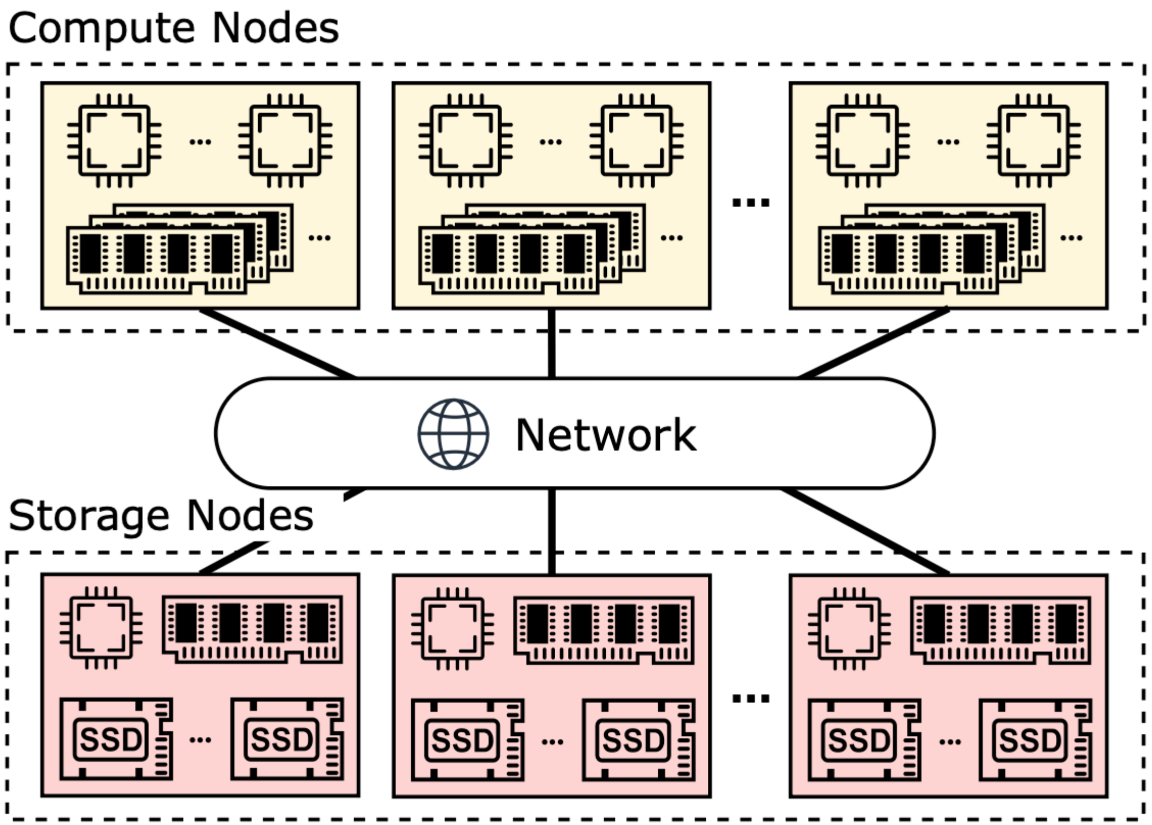
Qing Zheng
Los Alamos National Laboratory

Presenter: Junghyun Ryu

The 11th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD 2025), Nov 17th, 2025, St. Louis, MO
Held in conjunction with SC25: The International Conference for High Performance Computing, Networking, Storage, and Analysis

Excessive Data Movement in Analytical Workloads

Modern large-scale data processing analytics systems are now increasingly built on **disaggregated architectures** that physically separate compute and storage nodes.



- Disaggregated architecture -

Benefits:

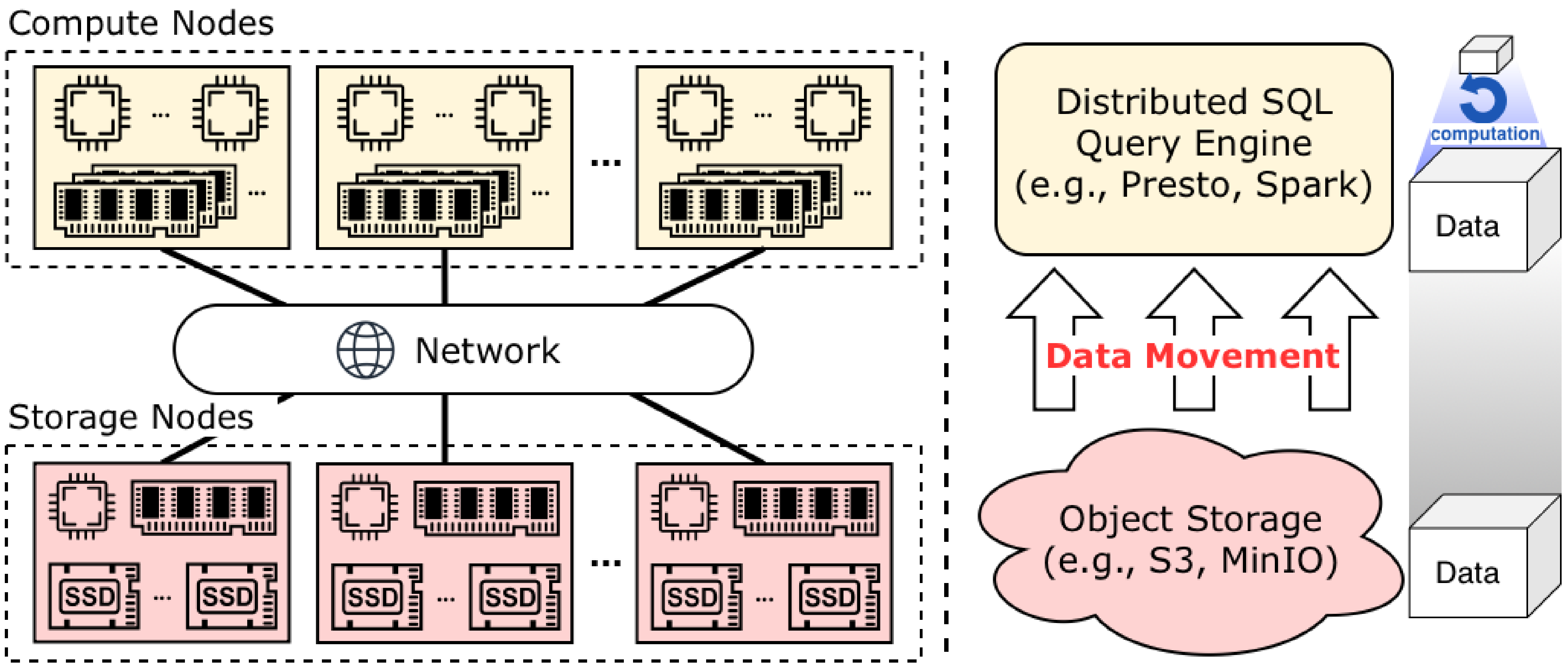
- Independent Scalability
  - Scale compute and storage separately
  - Optimize resources based on workload demands
- Simplified Management
  - Easier maintenance and upgrades
  - Flexible resource allocation

Trade-off:

- **Network Becomes Critical Path**
  - All data must traverse the network
  - Remote access vs. Local I/O:
    - Higher latency
    - Bandwidth constraints

SOGANG UNIVERSITY

Excessive Data Movement in Analytical Workloads



- Data analytical workloads used in HPC typically access only a small fraction of the dataset, yet still incur significant overhead from transferring entire files [1]
- More than half of all queries in Google's analytical workloads return less than 1% of total data [2]

[1] I. Park et al., "KV-CSD: A Hardware-Accelerated Key-Value Store for Data-Intensive Applications", IEEE International Conference on Cluster Computing (CLUSTER), 2023.
[2] S. Melnik et al., Dremel: A Decade of Interactive SQL Analysis at Web Scale. Proceedings of the VLDB Endowment 13, 12, 2020.

SOGANG UNIVERSITY

Object Storage

Object storage is a storage architecture that manages data as discrete objects.

- Features:
  - Flat namespace (bucket/object)
  - Globally unique object IDs
  - Highly scalable & stateless
- Naturally aligns with column-oriented data formats (Parquet, ORC)
  - This enables selective column retrieval without reading entire datasets, significantly reducing I/O overhead for analytical queries that typically access only a subset of columns.

Ex) SQL: SELECT **Col_A**, **Col_C** FROM table WHERE Col_A > 100

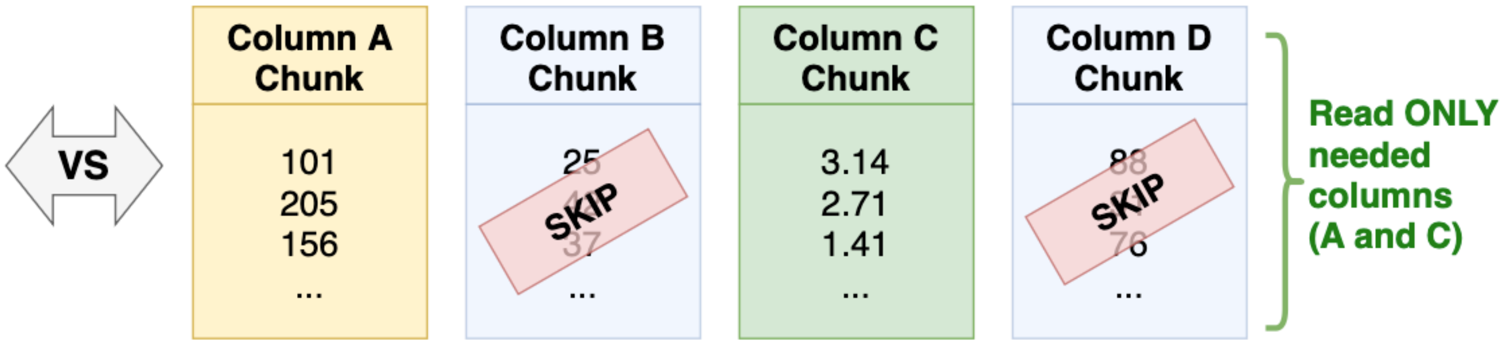
- Traditional row-based data -

Col A	Col B	Col C	Col D
101	25	3.14	88
205	42	2.71	91
156	37	1.41	76

...

Read ALL 4 columns

- Columnar Format (Parquet/ORC) -

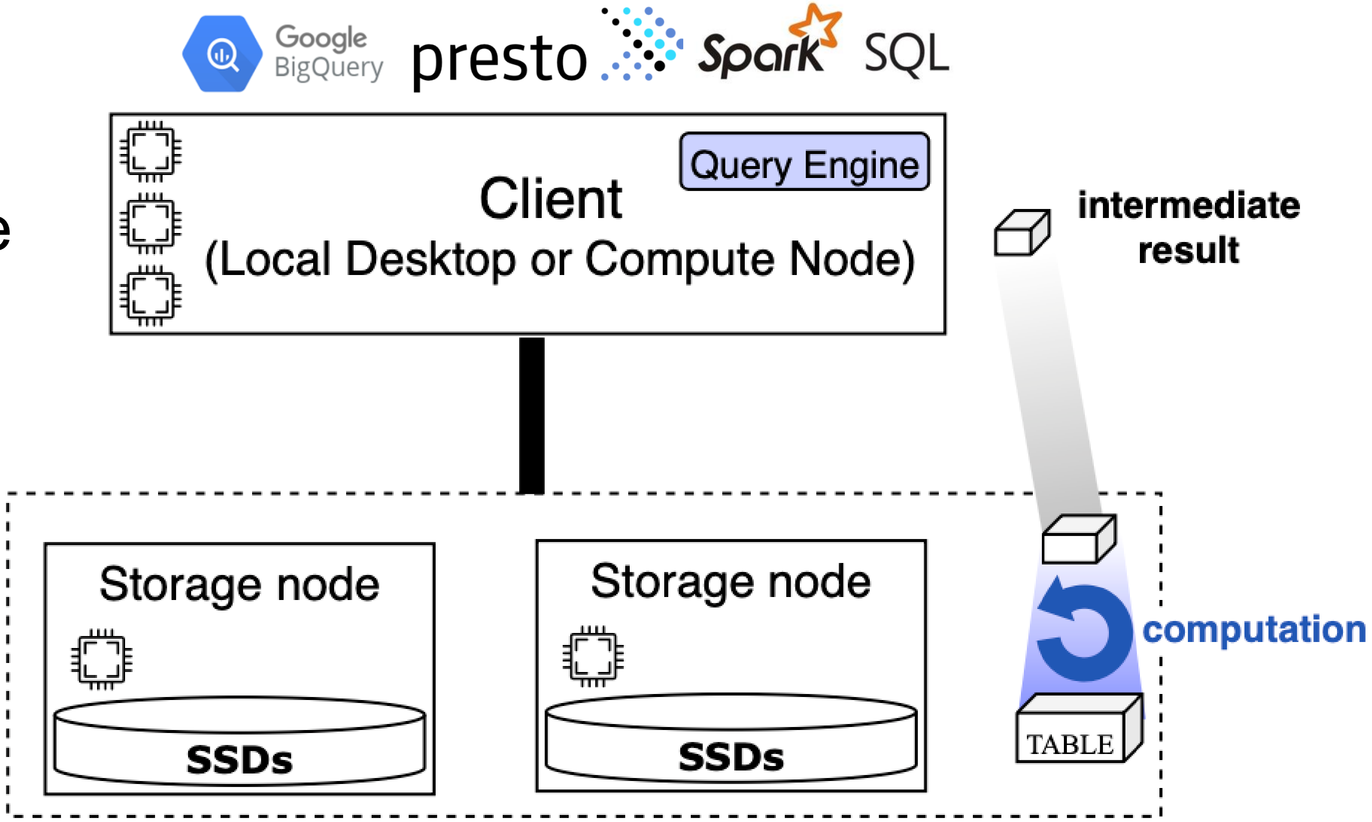


SOGANG UNIVERSITY

Query Pushdown

Query pushdown offloads certain SQL operators directly to the storage layer, allowing **data reduction** to occur before network transfer

- Exploits the structural characteristic (disk I/O bandwidth > network bandwidth)
- Execute data-intensive operations directly at storage nodes
- Transfer only intermediate results
- Trade-off:
  Lower compute capacity at storage vs.
  Massive reduction in data movement



SOGANG UNIVERSITY

Object Storage Query Pushdown: Current Limitations

Existing object storage's Query Pushdown support : AWS S3 Select & MinIO Select

Supported: SQL *SELECT* (column projection) + *WHERE* (row filtering) clause
Benefit: Reduce data transfer via storage-side filtering — Only matching rows and columns sent to compute

❖ S3 SELECT & MinIO Select: Critical Limitations

Limited Operator Support

- High-Impact Operators NOT Supported:
  - Aggregation (GROUP BY)
    Functions: SUM, AVG, COUNT, MIN, MAX
  - Top-N (ORDER BY + LIMIT)
    Reduces all rows → top N rows
  - Result: Must execute on compute nodes
    → Massive data movement

Not Suitable for HPC

- No double-precision floating-point support
  - Impact: HPC simulations require high numeric precision
    (climate modeling, fluid dynamics, quantum computing, ...)
  - Financial analytics with precise calculations
  - Result: Unsuitable for scientific workloads

- Data movement bottleneck only partially addressed
- Substantial optimization opportunities remain untapped
- Scientific workloads cannot leverage query pushdown

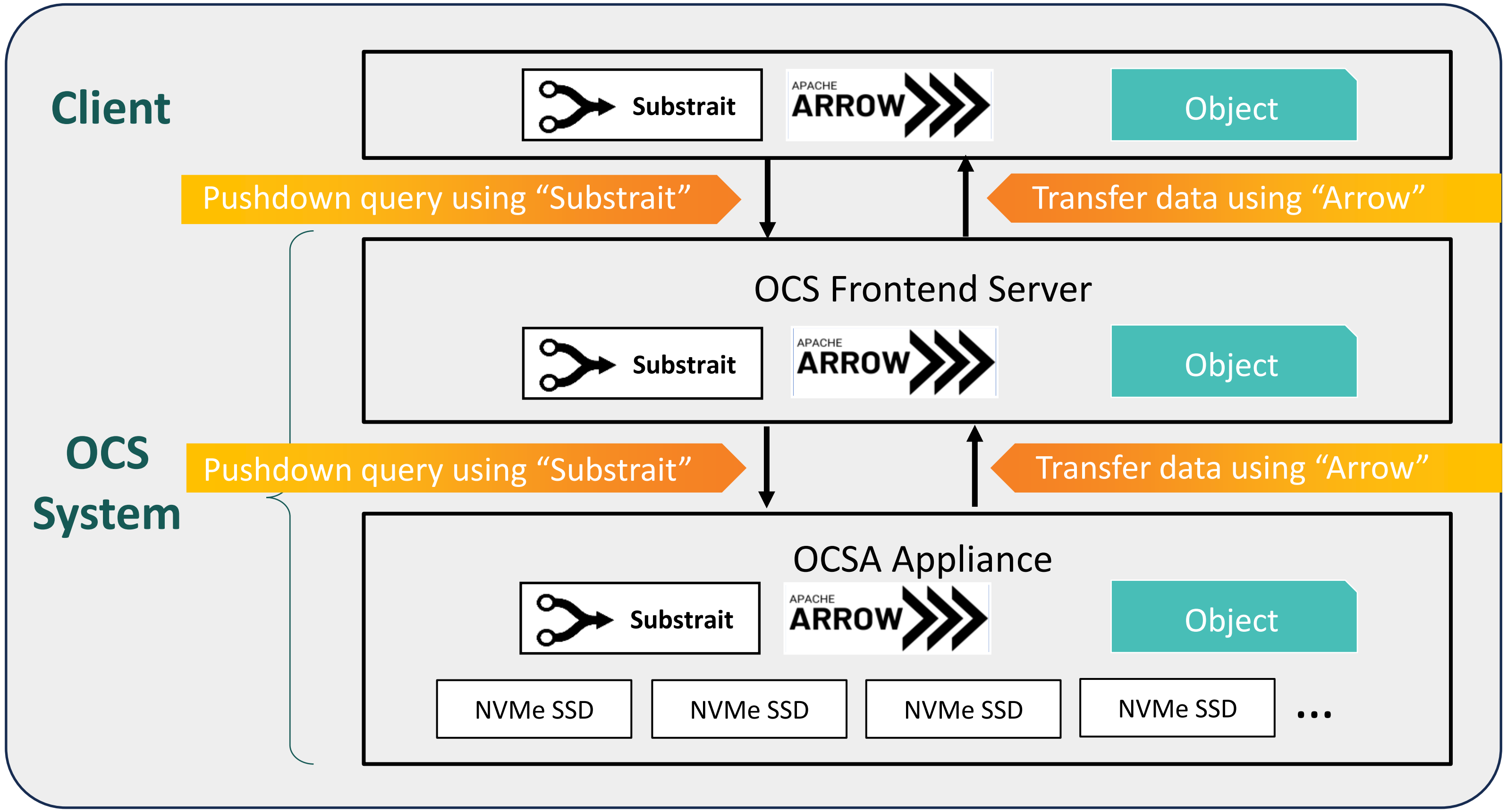
→ **Need: More powerful computational storage solution**

SOGANG UNIVERSITY

Towards Computational Object Storage

SK Hynix has introduced Object-based Computational Storage (OCS)

- Supports various SQL operators
- Embedded SQL engine integrated within storage system
- Works with S3-compatible interface
- Uses standard interfaces for
  - Query repr.: Substrait IR
  - Data transfer: Apache Arrow

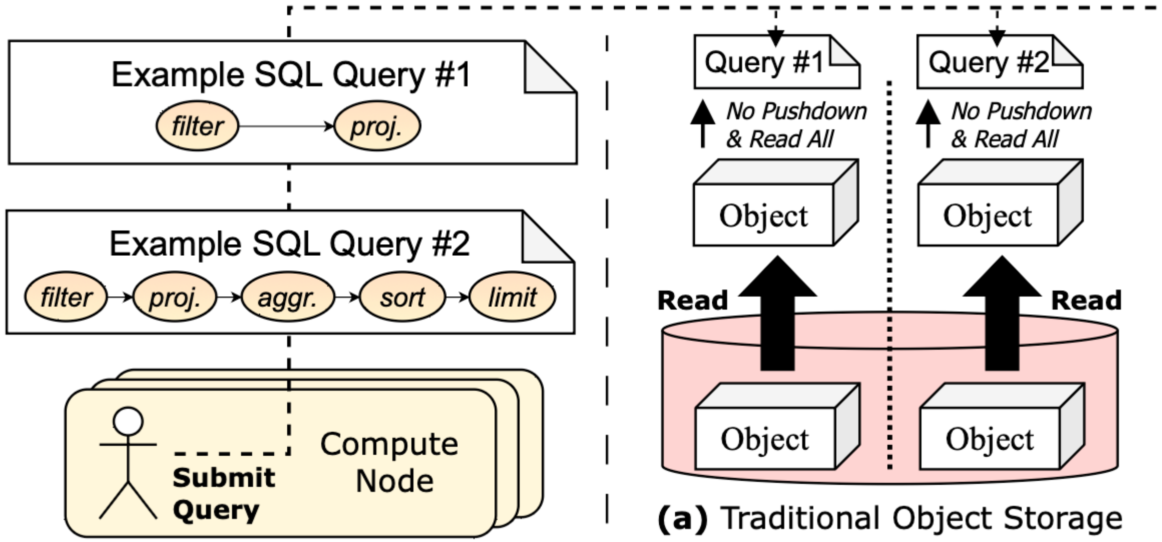


- SK Hynix Object-based Computational Storage [3] -

[3] Jongryool Kim, Accelerating Data Analytics Using Object-based Computational Storage System in HPC, 2023, https://sc23.conference-program.com/presentation/?id=exforum116&sess=sess255

SOGANG UNIVERSITY

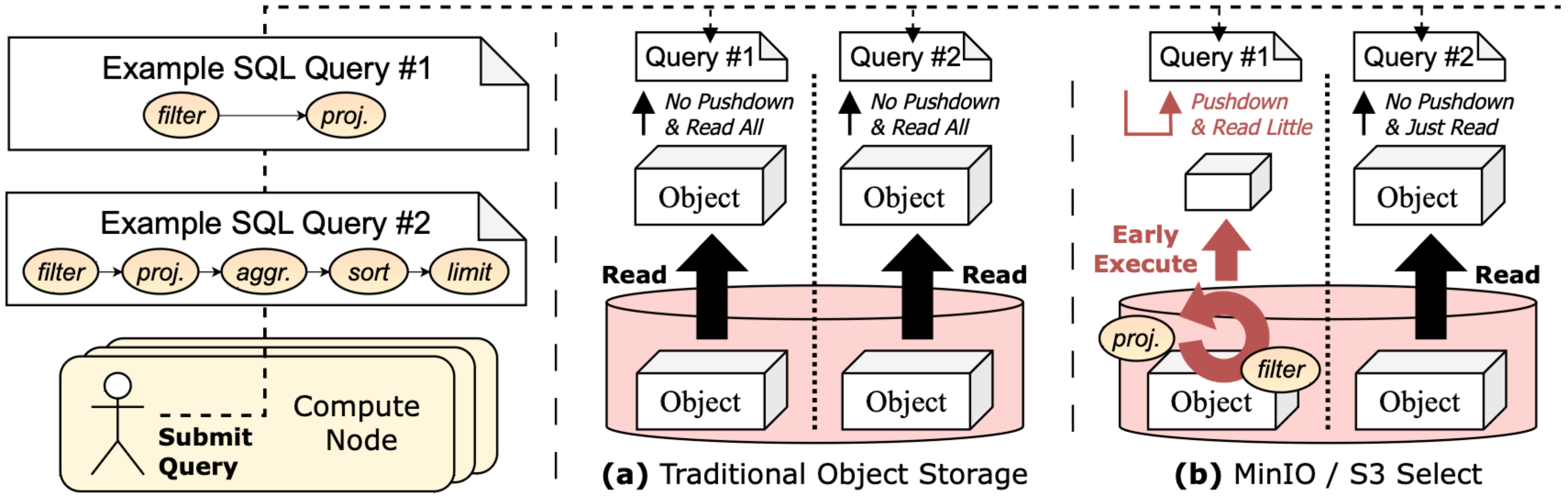
Towards Computational Object Storage



(a) Traditional Object Storage

Operator	SQL Clause	Data Reduction	Traditional	S3, MinIO SELECT	SK Hynix OCS
Column Projection	SELECT col1, col2	Medium	⚠		
Row Filtering	WHERE condition	High	⚠		
Aggregation	GROUP BY	**Very High**	❌		
Sorting	ORDER BY	No	❌		
Limiting	LIMIT N	**Very High**	❌		
Top-N	ORDER BY + LIMIT	**Very High**	❌		

Towards Computational Object Storage



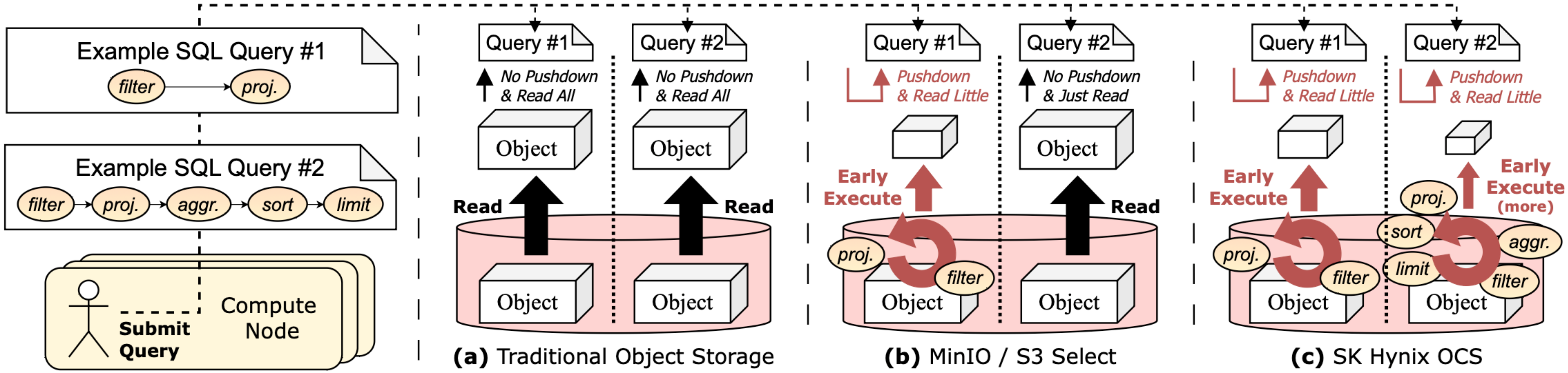
(a) Traditional Object Storage

(b) MinIO / S3 Select

Operator	SQL Clause	Data Reduction	Traditional	S3, MinIO SELECT	SK Hynix OCS
Column Projection	SELECT col1, col2	Medium	⚠	✅	
Row Filtering	WHERE condition	High	⚠	✅	
Aggregation	GROUP BY	**Very High**	❌	❌	
Sorting	ORDER BY	No	❌	❌	
Limiting	LIMIT N	**Very High**	❌	❌	
Top-N	ORDER BY + LIMIT	**Very High**	❌	❌	

SOGANG UNIVERSITY

Towards Computational Object Storage



Operator	SQL Clause	Data Reduction	Traditional	S3, MinIO SELECT	SK Hynix OCS
Column Projection	SELECT col1, col2	Medium	⚠	✅	✅
Row Filtering	WHERE condition	High	⚠	✅	✅
Aggregation	GROUP BY	**Very High**	❌	❌	✅
Sorting	ORDER BY	No	❌	❌	✅
Limiting	LIMIT N	**Very High**	❌	❌	✅
Top-N	ORDER BY + LIMIT	**Very High**	❌	❌	✅

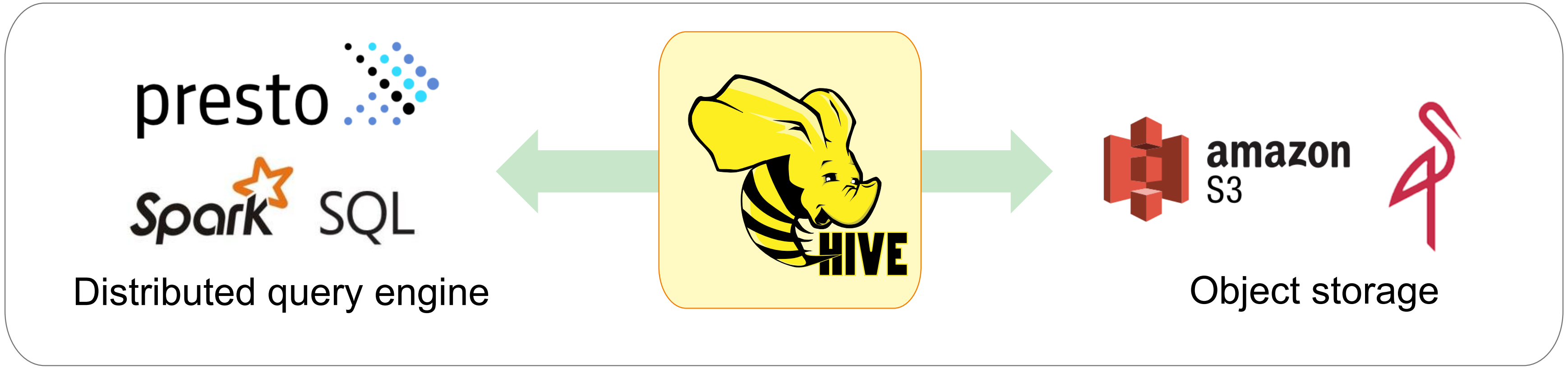
Standard APIs Limit Computational Storage

Current State: Hive Connector as Standard Interface for S3-compatible Object Storages

- Unified interface for S3-compatible object storage
- Wide compatibility across storage backends
- Standard pushdown API support

Hive connector remains limited to standard pushdown API (S3 Select specification)

- Problem:
    - Cannot expose OCS's extended operator support
    - Performance gains from OCS remain unrealized



Standard APIs Limit Computational Storage

Current State: Hive Connector as Standard Interface for S3-compatible Object Storages

- Unified interface for S3-compatible object storage
- Wide compatibility across storage backends
- Standard pushdown API support

Hive connector remains limited to standard pushdown API (S3 Select specification)

- Problem:
  - Cannot expose OCS's extended operator support
  - Performance gains from OCS remain unrealized

✅ Filter
✅ Projection
✅ Aggregation
✅ Top-N

OCS Pushdown Capabilities

✅ Filter
✅ Projection
❌ Aggregation ← Gap
❌ Top-N ← Gap
❌ Double Prec. ← Gap

Hive Exposes (S3 SELECT)

SOGANG UNIVERSITY

Standard APIs Limit Computational Storage

Current State: Hive Connector as Standard Interface for S3-compatible Object Storages

- Unified interface for S3-compatible object storage
- Wide compatibility across storage backends
- Standard pushdown API support

Hive connector remains limited to standard pushdown API (S3 Select specification)

- Problem:
  - Cannot expose OCS's extended operator support

Although OCS supports complex operations such as aggregation and top-N, these capabilities remain inaccessible

→ OCS-specific connector essential to bridge this integration gap

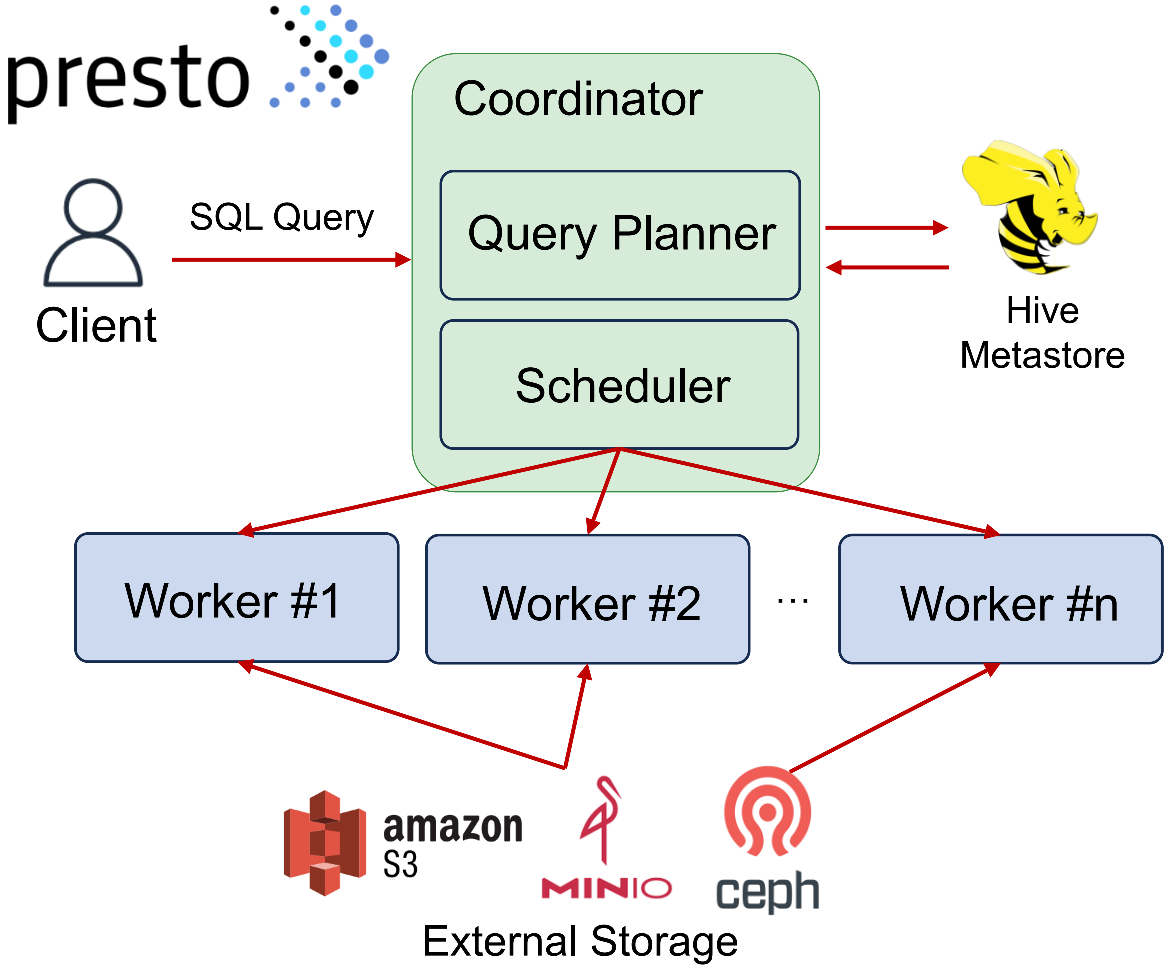
OCS Pushdown Capabilities

Hive Exposes (S3 SELECT)

SOGANG UNIVERSITY

Case Study: Presto as Distributed SQL Engine

To demonstrate OCS benefits in distributed SQL engines → we use Presto as a case study
Goal: Design and implement an OCS connector that fully exploits advanced pushdown capabilities



❖ Why Presto?

- Modular Architecture
  - Service Provider Interface (SPI): Well-defined extension points
  - Connector-specific optimization
- Connector-Based Extensibility
  - New connectors added independently
  - Storage-specific features exposed
  - Optimization hooks via SPI
- Wide Industry Adoption
  - Enterprise Users: Meta (Facebook), Uber, Netflix, Airbnb
  - Petabyte-scale data processing
  - Thousands of concurrent queries

Design Overview

- **High-Level Design Goals**
  - Intercept query operators during optimization phase
  - Detect pushdown-eligible operators (filter, aggregation, top-N)
  - Translate operators into Substrait IR for in-storage execution
- **Key Principle: Preserve Presto's Modular Architecture**
  - No modifications to Presto's core execution pipeline
  - Extends Connector SPI for storage-specific optimizations

SOGANG UNIVERSITY

Presto's Query Planning Workflow

1. SQL Parsing

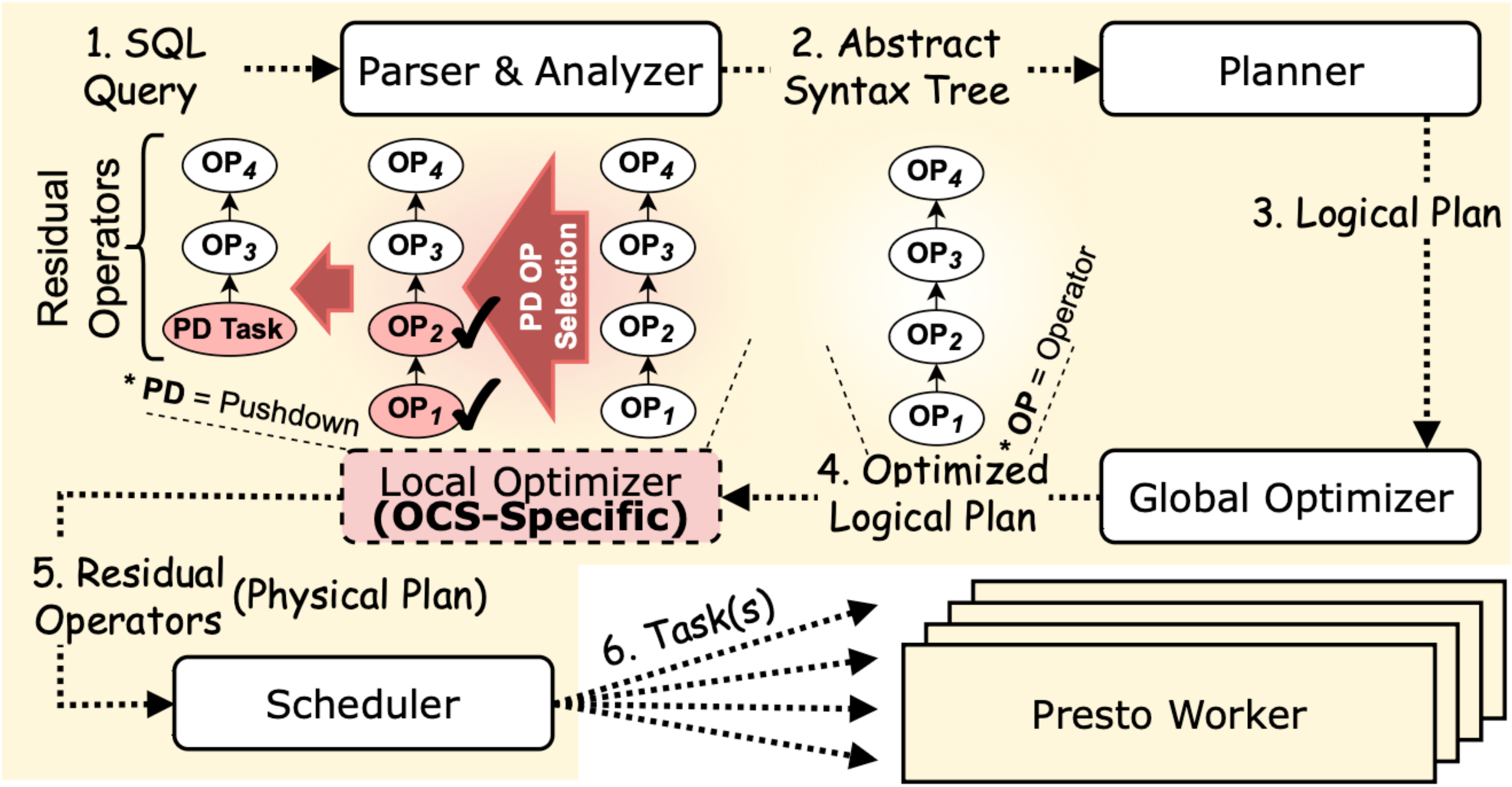
Input query → Abstract Syntax Tree (AST)

2. Logical Plan Construction

AST → Logical plan (TableScanNode, FilterNode, AggregationNode)

3. Global Optimization

Rule-based transformations (join reordering, projection pruning)



SOGANG UNIVERSITY

Presto's Query Planning Workflow

1. SQL Parsing

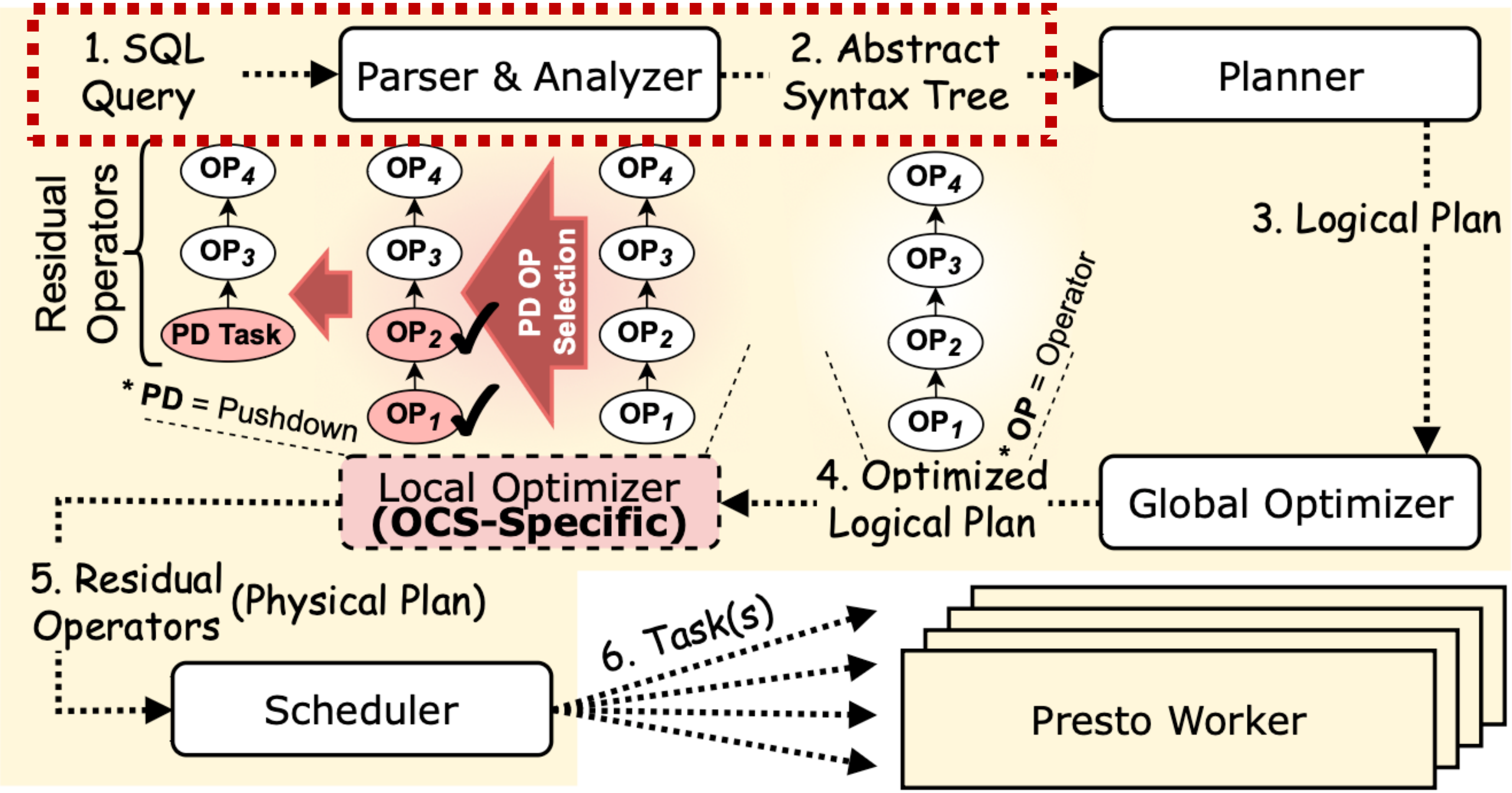
Input query → Abstract Syntax Tree (AST)

2. Logical Plan Construction

AST → Logical plan (TableScanNode, FilterNode, AggregationNode)

3. Global Optimization

Rule-based transformations (join reordering, projection pruning)



SOGANG UNIVERSITY

Presto's Query Planning Workflow

1. SQL Parsing

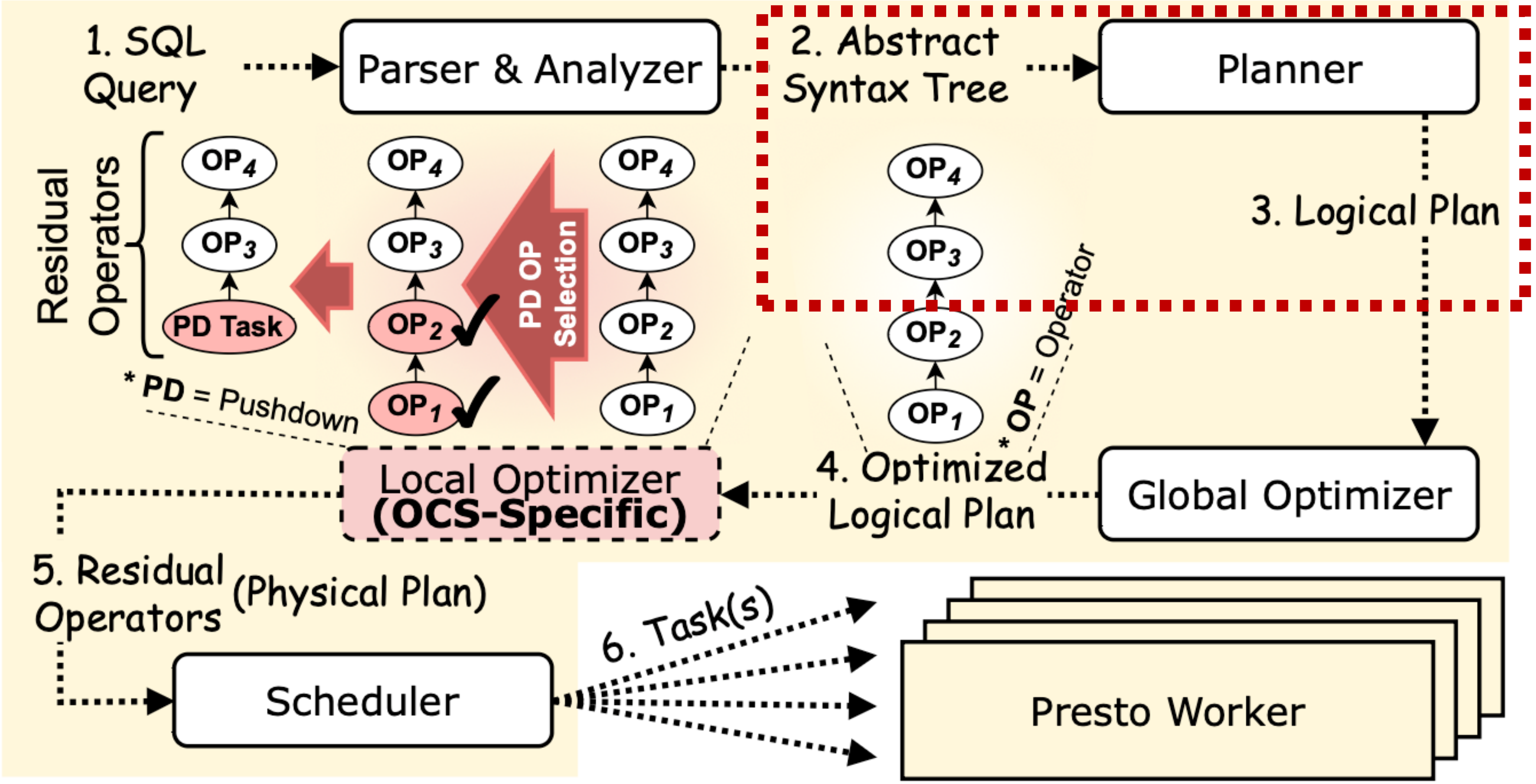
Input query → Abstract Syntax Tree (AST)

2. Logical Plan Construction

AST → Logical plan (TableScanNode, FilterNode, AggregationNode)

3. Global Optimization

Rule-based transformations (join reordering, projection pruning)



SOGANG UNIVERSITY

Presto's Query Planning Workflow

1. SQL Parsing

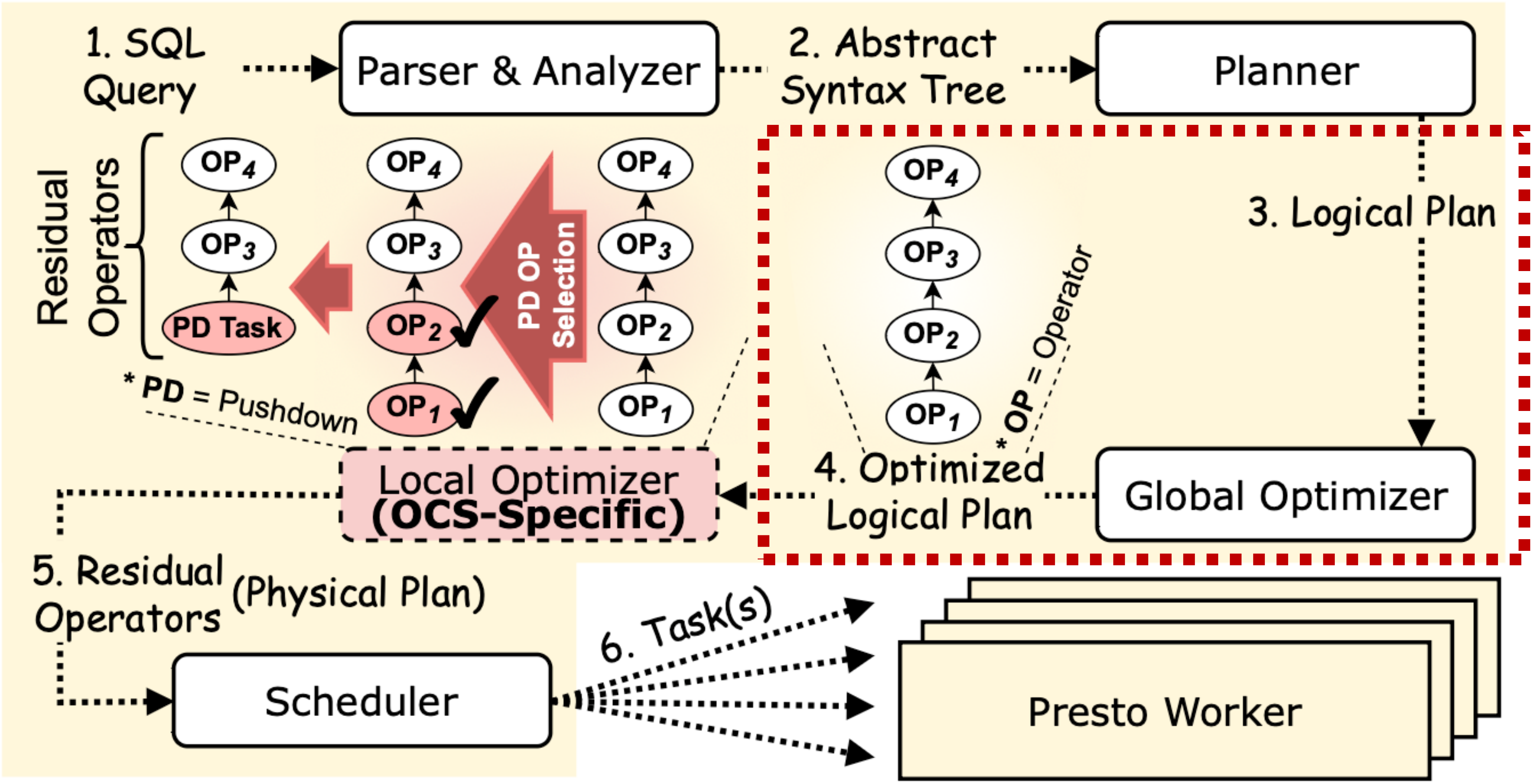
Input query → Abstract Syntax Tree (AST)

2. Logical Plan Construction

AST → Logical plan (TableScanNode, FilterNode, AggregationNode)

3. Global Optimization

Rule-based transformations (join reordering, projection pruning)



SOGANG UNIVERSITY

Presto's Query Planning Workflow

4. Local Optimization

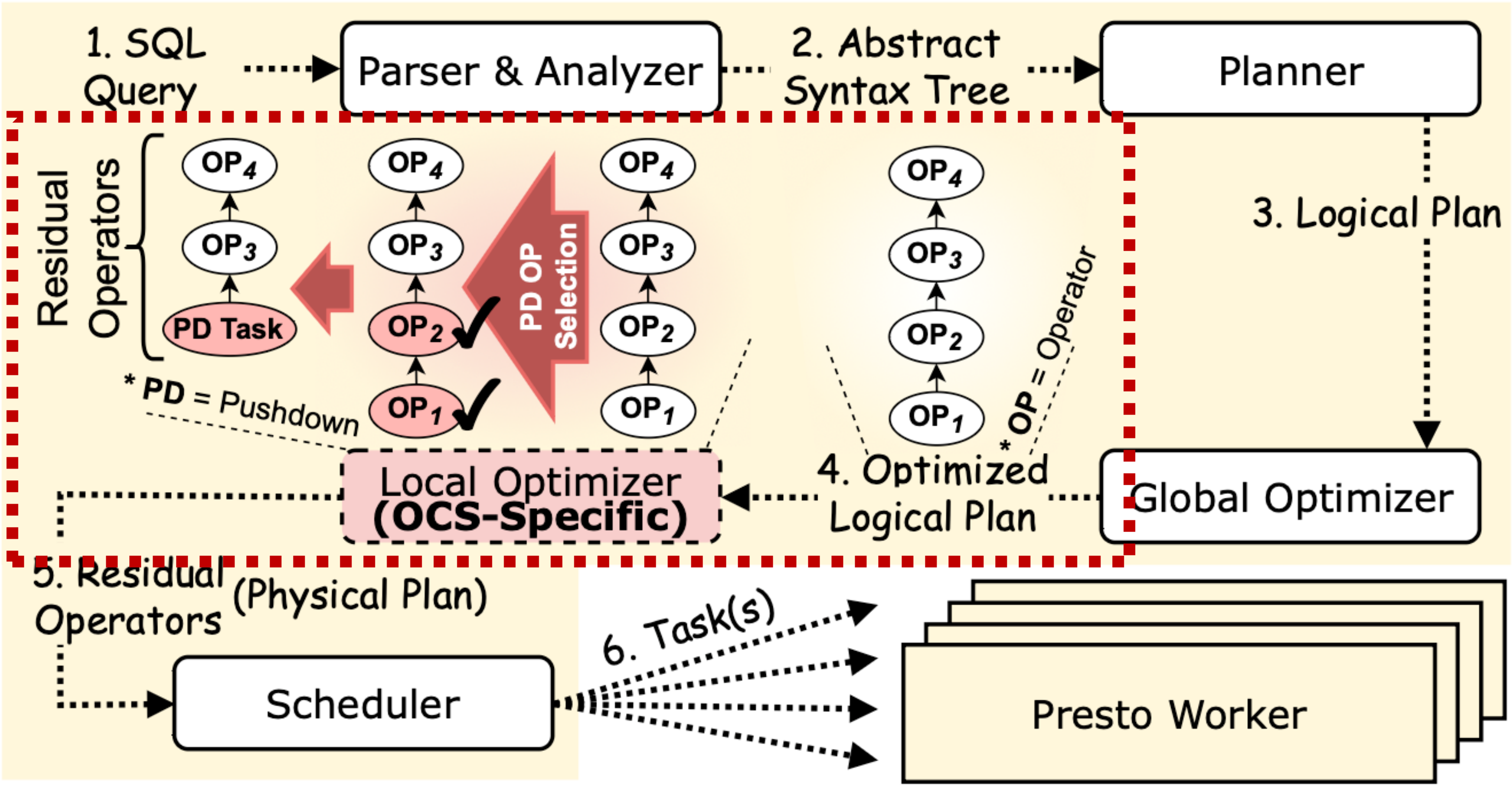
Connector-specific optimizations (OCS pushdown selection)

5. Physical Planning

Logical plan → Physical plan with execution strategies

6. Split Generation & Scheduling

Partition TableScan into splits, distribute to workers



SOGANG UNIVERSITY

Presto's Query Planning Workflow

4. Local Optimization

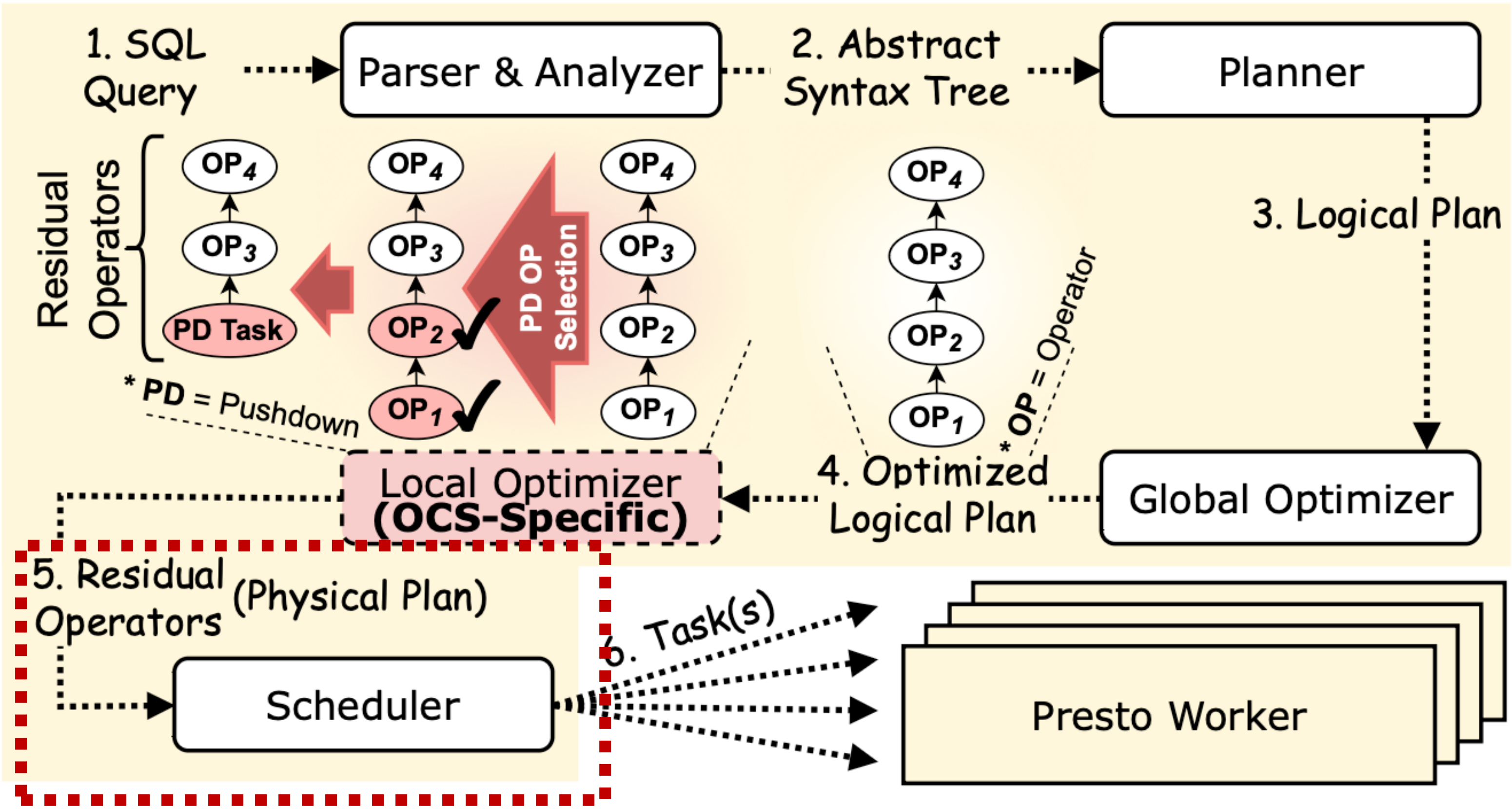
Connector-specific optimizations (OCS pushdown selection)

5. Physical Planning

Logical plan → Physical plan with execution strategies

6. Split Generation & Scheduling

Partition TableScan into splits, distribute to workers



SOGANG UNIVERSITY

Presto's Query Planning Workflow

4. Local Optimization

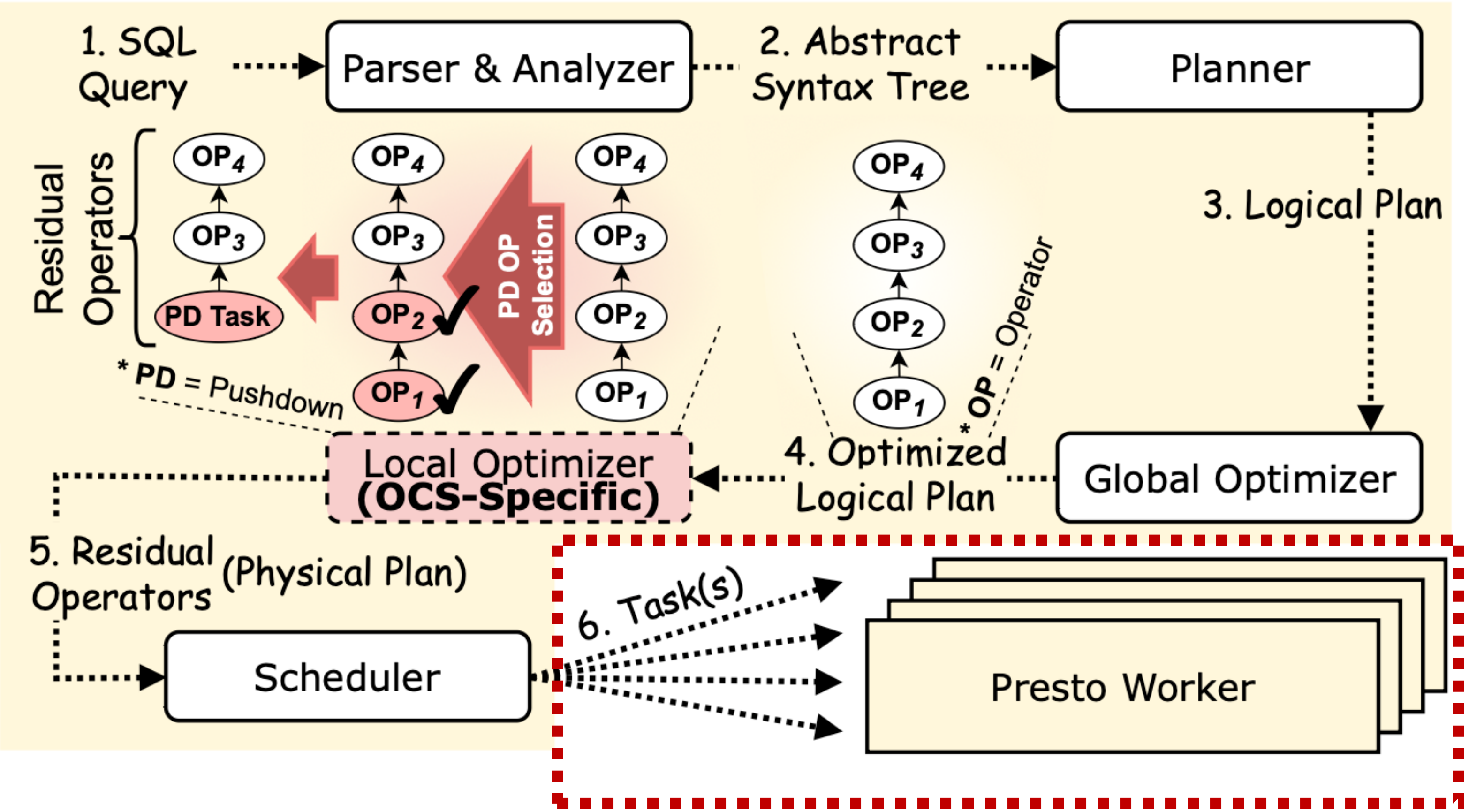
Connector-specific optimizations (OCS pushdown selection)

5. Physical Planning

Logical plan → Physical plan with execution strategies

6. Split Generation & Scheduling

Partition TableScan into splits, distribute to workers



Presto-OCS Connector: Key Components

1. Selectivity Analyzer

- Evaluates operator data reduction potential
- Uses Hive metastore statistics (min/max, NDV, row count)
- Estimates selectivity for filter, aggregation, top-N operators

2. Operator Extractor

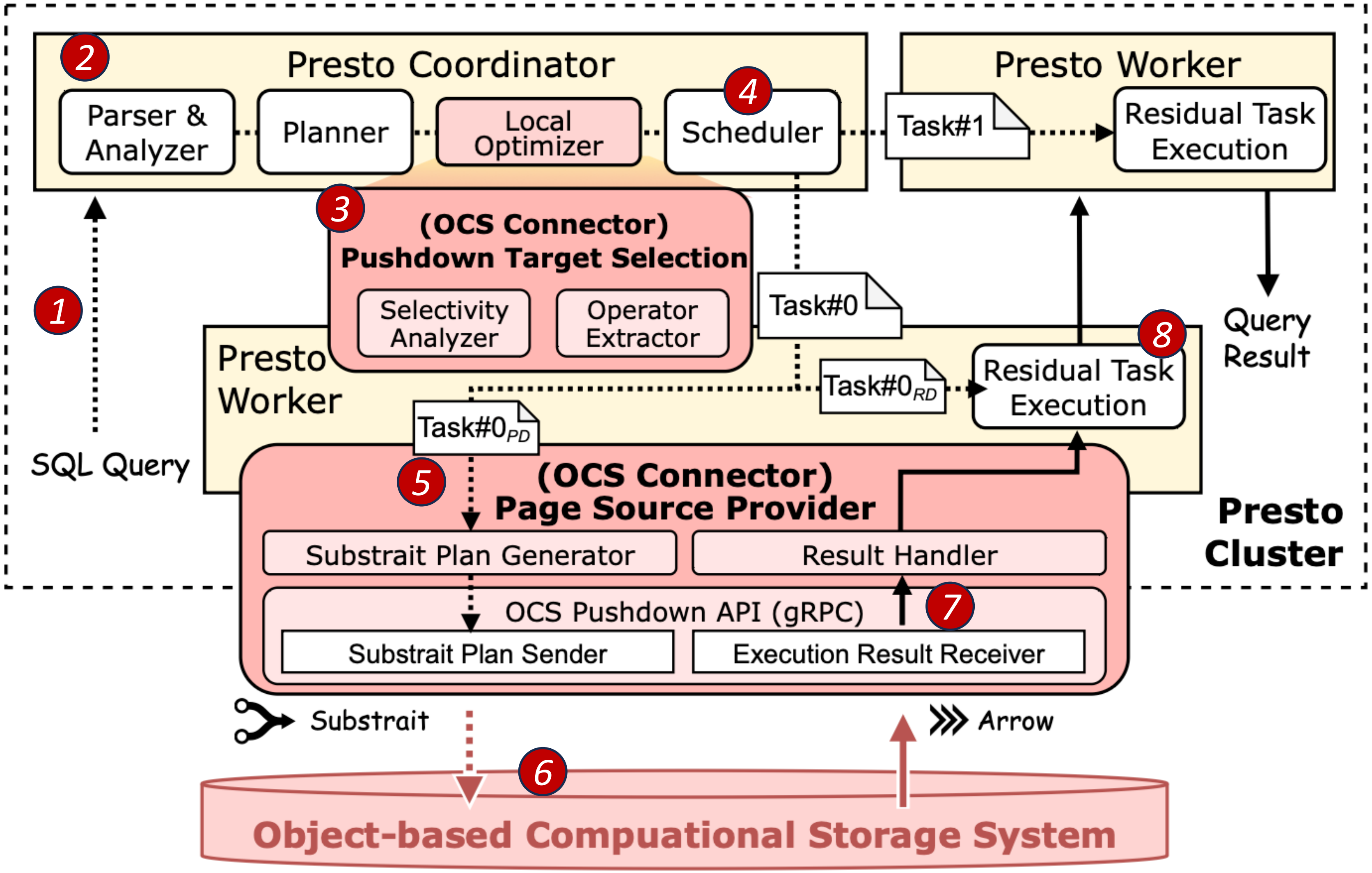
- Captures pushdown-eligible operators from logical plan
- Preserves SQL conditions: filter predicates, GROUP BY keys, ORDER BY criteria

3. Page Source Provider

- Reconstructs operators into SQL statements
- Translates SQL to Substrait IR (cross-system query plan)
- Handles type normalization and function mapping
- Communicates with OCS via gRPC, deserializes Arrow results

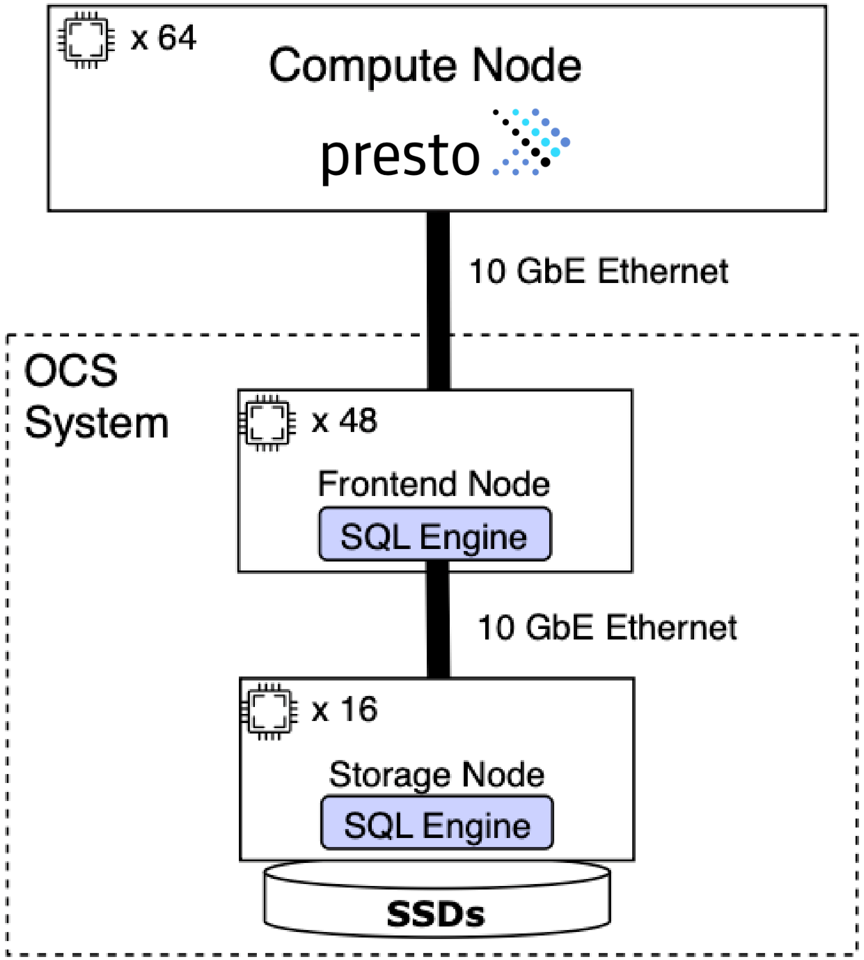
SOGANG UNIVERSITY

Design of Presto-OCS Connector



SOGANG UNIVERSITY

Experimental Setup: Testbeds

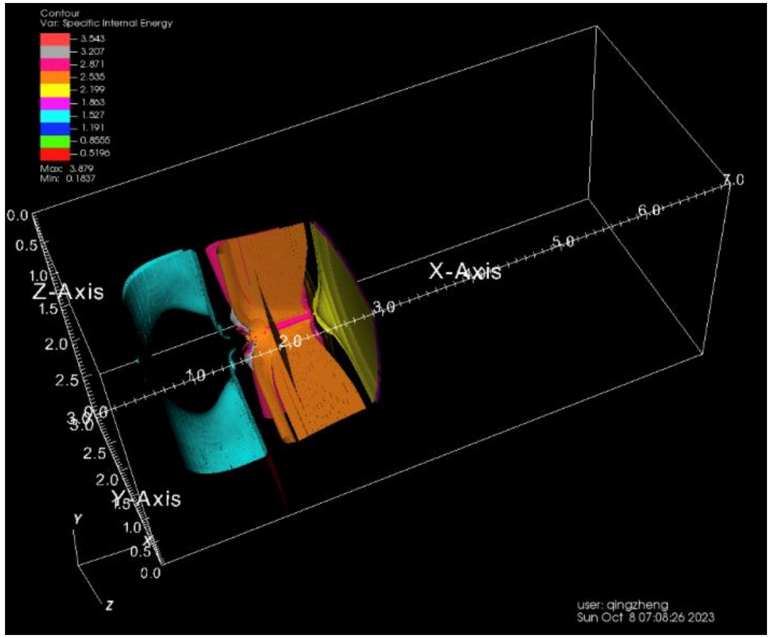


Compute Node Specifications	
CPU	Intel(R) Xeon(R) Gold 6226R (64 cores, 2.9 GHz max)
Memory	384 GB DDR4
Storage	1 TB NVMe SSD
Frontend Node Specifications	
CPU	Intel® Xeon® Silver 4410Y (48 cores, 3.9 GHz max)
Memory	64 GB DDR4
Storage	1 TB NVMe SSD
Storage Node Specifications	
CPU	Intel® Xeon® Silver 4410Y (16 cores, 2.0 GHz max)
Memory	64 GB DDR4
Storage	1 TB NVMe SSD + 512 GB SATA SSD

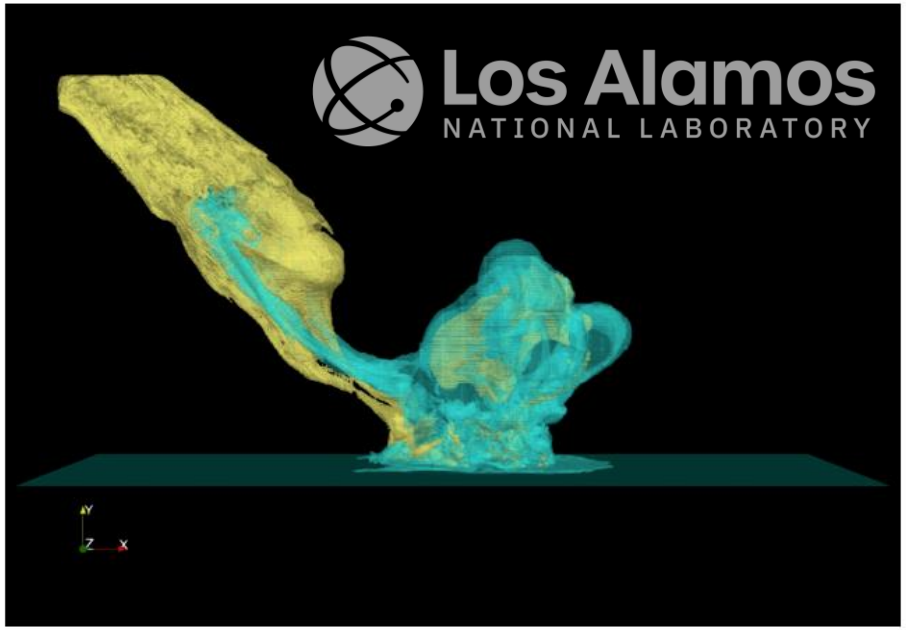
SOGANG UNIVERSITY

Experimental Setup: Workloads

- We employ scientific simulation datasets with their corresponding analytical queries used at Los Alamos National Laboratory (LANL), as well as a standard decision-support benchmark (TPC-H).



- LAGrangian High-Order Solver (Laghos) dataset [4] -



- Deep Water Asteroid Impact dataset [5] -

TPC®

- TPC-H -

[4] Los Alamos National Laboratory. 2024. Laghos Sample Dataset. https://github.com/lanl-ocs/laghos-sample-dataset.
[5] Q. Zheng et al., "Accelerating Viz Pipelines Using Near-Data Computing: An Early Experience," SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, GA, USA, 2024, pp. 326-335

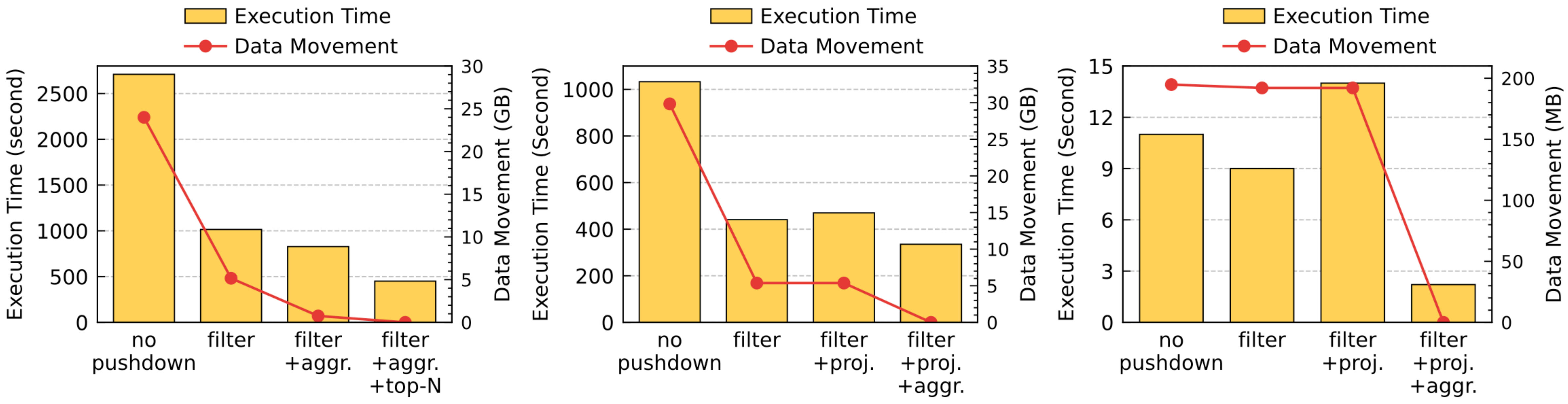
Experimental Setup: Workloads

Dataset	Query	Selectivity	Execution Plan
Laghos	SELECT min(vertex_id) AS VID, min(x), min(y), min(z), avg(e) FROM parquet WHERE x, y, z BETWEEN 0.8 AND 3.2 GROUP BY vertex_id ORDER BY E LIMIT 100	0.0023842%	TableScan →Filter → Aggregation →Top-N
Deep Water	SELECT MAX((rowid % (500*500))/500), timestep FROM parquet WHERE v02 >0.1 GROUP BY timestep	0.0000032% (average)	TableScan →Filter → Project → Aggregation
TPC-H	SELECT returnflag, linestatus, SUM(quantity), SUM(extendedprice), SUM(extendedprice * (1 - discount)), SUM(extendedprice * (1 - discount) * (1 + tax)), AVG(quantity), AVG(extendedprice), AVG(discount), COUNT(*) FROM lineitem WHERE shipdate ≤ DATE ’1998-12-01’ - INTERVAL ’90 DAY’ GROUP BY returnflag, linestatus ORDER BY returnflag, linestatus	0.0000667%	TableScan → Filter → Project → Aggregation → Sort

SOGANG UNIVERSITY

Evaluation: Pushdown Impact

- *Q1: Does reducing data movement through pushdown improve query execution time?*

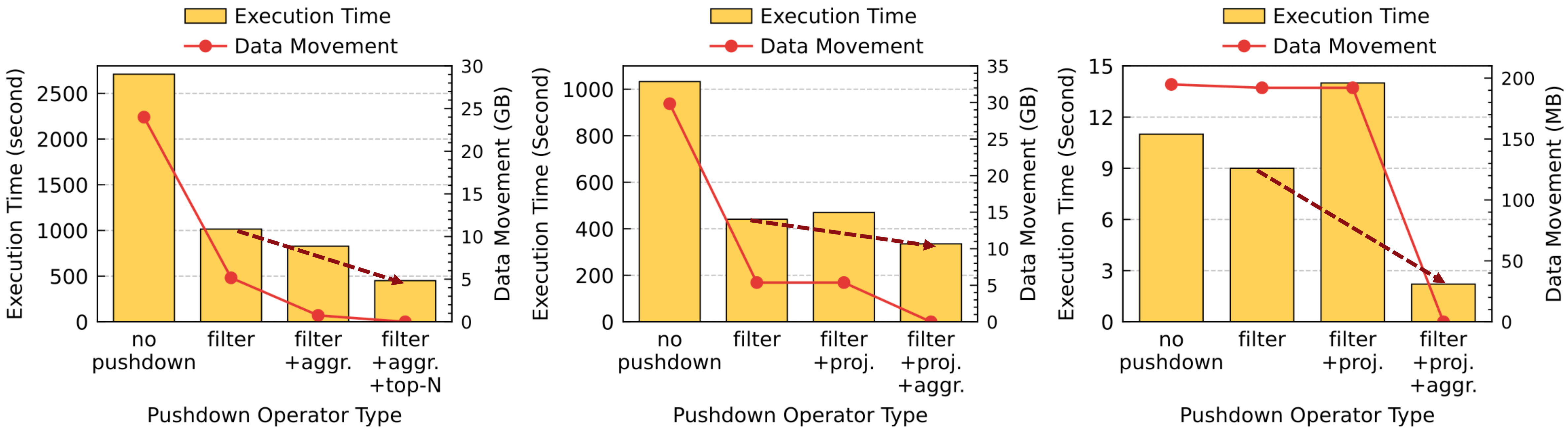


(a) Laghos (b) Deep Water Impact (c) TPC-H

SOGANG UNIVERSITY

Evaluation: Pushdown Impact

- *Q1: Does reducing data movement through pushdown improve query execution time?*

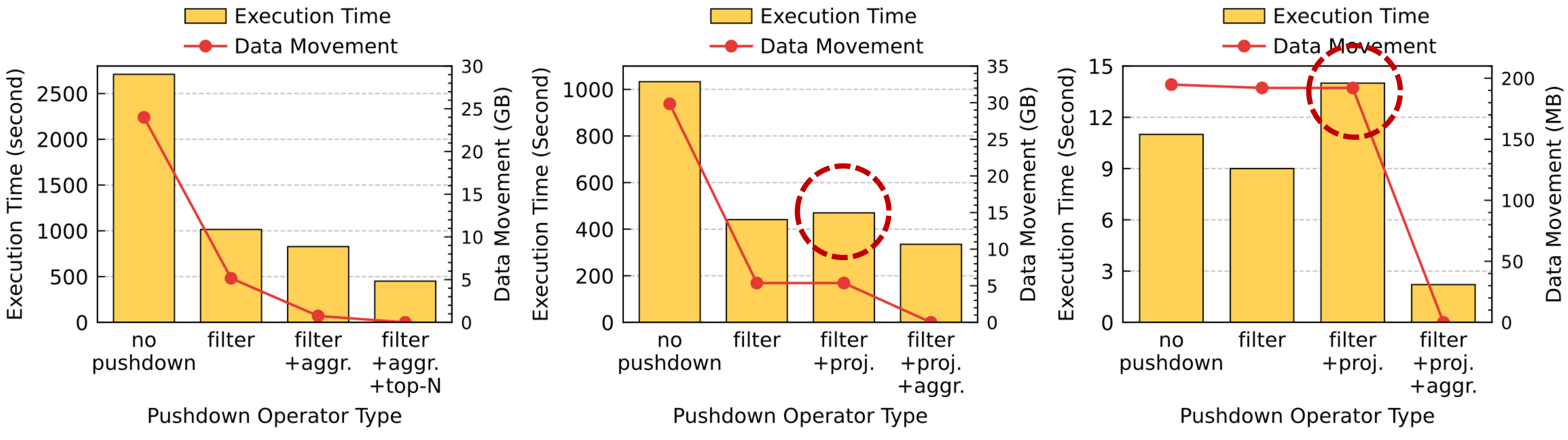


(a) Laghos (b) Deep Water Impact (c) TPC-H

- Progressive operator pushdown consistently reduces data movement and execution time
- Full pushdown (filter + aggregation + top-N): 2.25× speedup vs. filter-only
- Data movement reduction: 5.1GB → 0.5MB (99.99% reduction)
- Result demonstrates limitations of traditional object storage (filter-only pushdown)

Evaluation: Pushdown Impact

- *Q2. Is pushdown always beneficial regardless of operator type?*

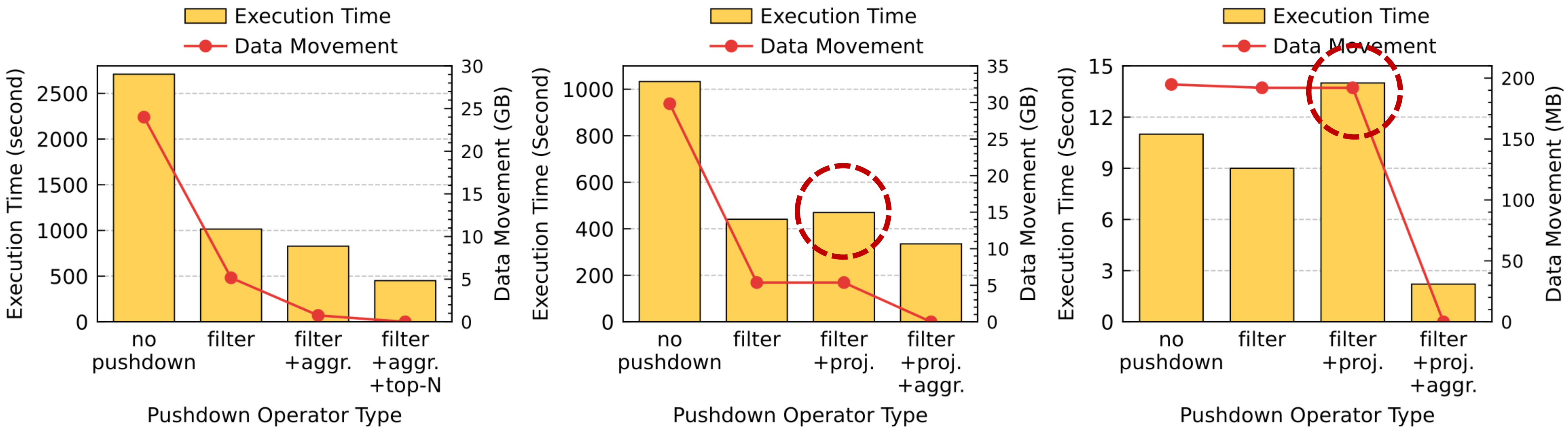


(a) Laghos (b) Deep Water Impact (c) TPC-H

- Deep Water Impact: Expression projection pushdown causes 7% slowdown
- TPC-H Q1: Projection pushdown causes 55% slowdown
- **Computational overhead** > data movement savings
- Complex arithmetic on multiple columns at weaker storage CPUs

Evaluation: Pushdown Impact

- *Q2. Is pushdown always beneficial regardless of operator type?*



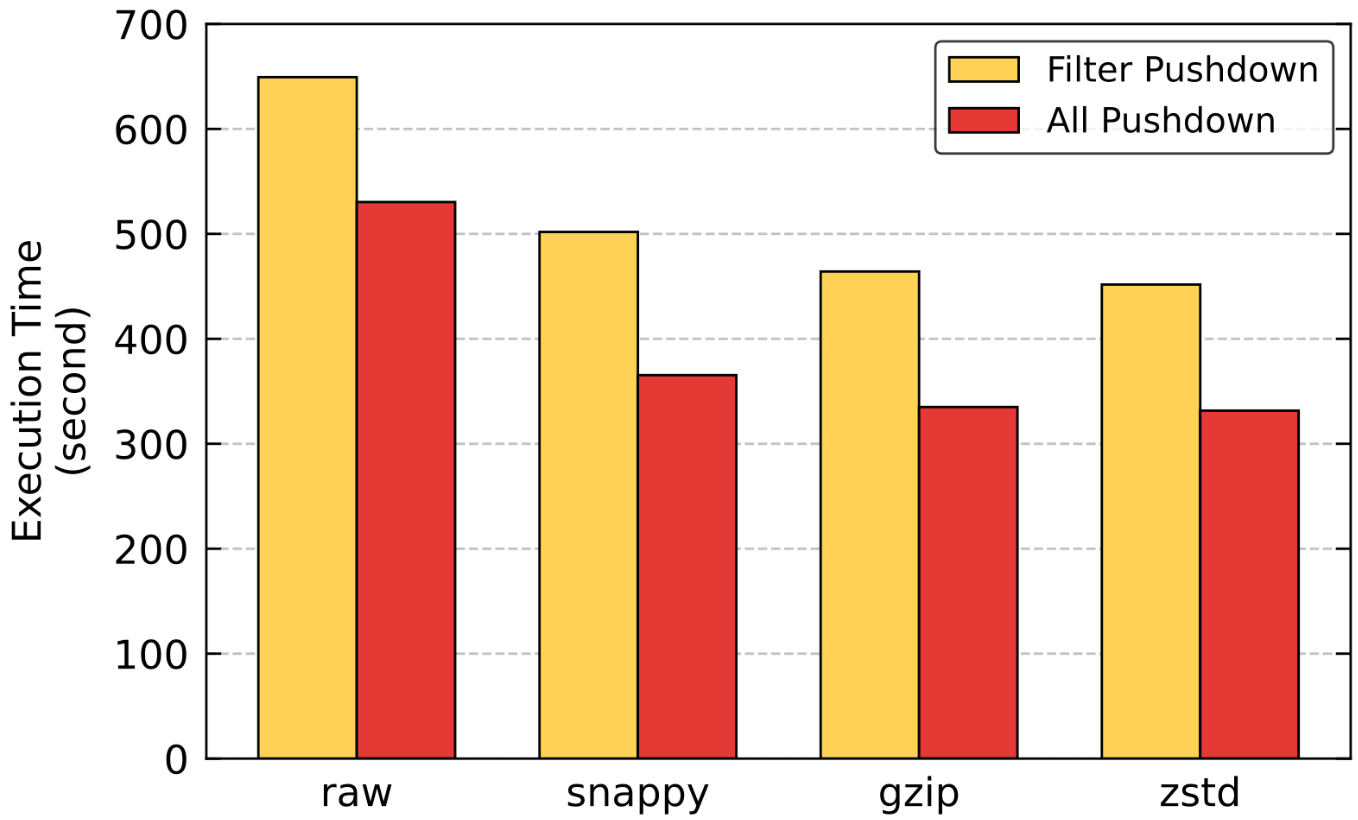
(a) Laghos (b) Deep Water Impact (c) TPC-H

- Aggregation pushdown recovers performance
  - Deep Water: 1.32× speedup vs. filter-only (441s → 335s, 5.37GB → 1MB)
  - TPC-H Q1: 4.07× speedup vs. filter-only (9s → 2.21s, 192MB → 0.5MB)
- Not all operators benefit from pushdown; selective pushdown based on operator complexity and data reduction ratio is critical

Evaluation: Pushdown with Compression

- *Q3. How does OCS pushdown perform with data compression?*

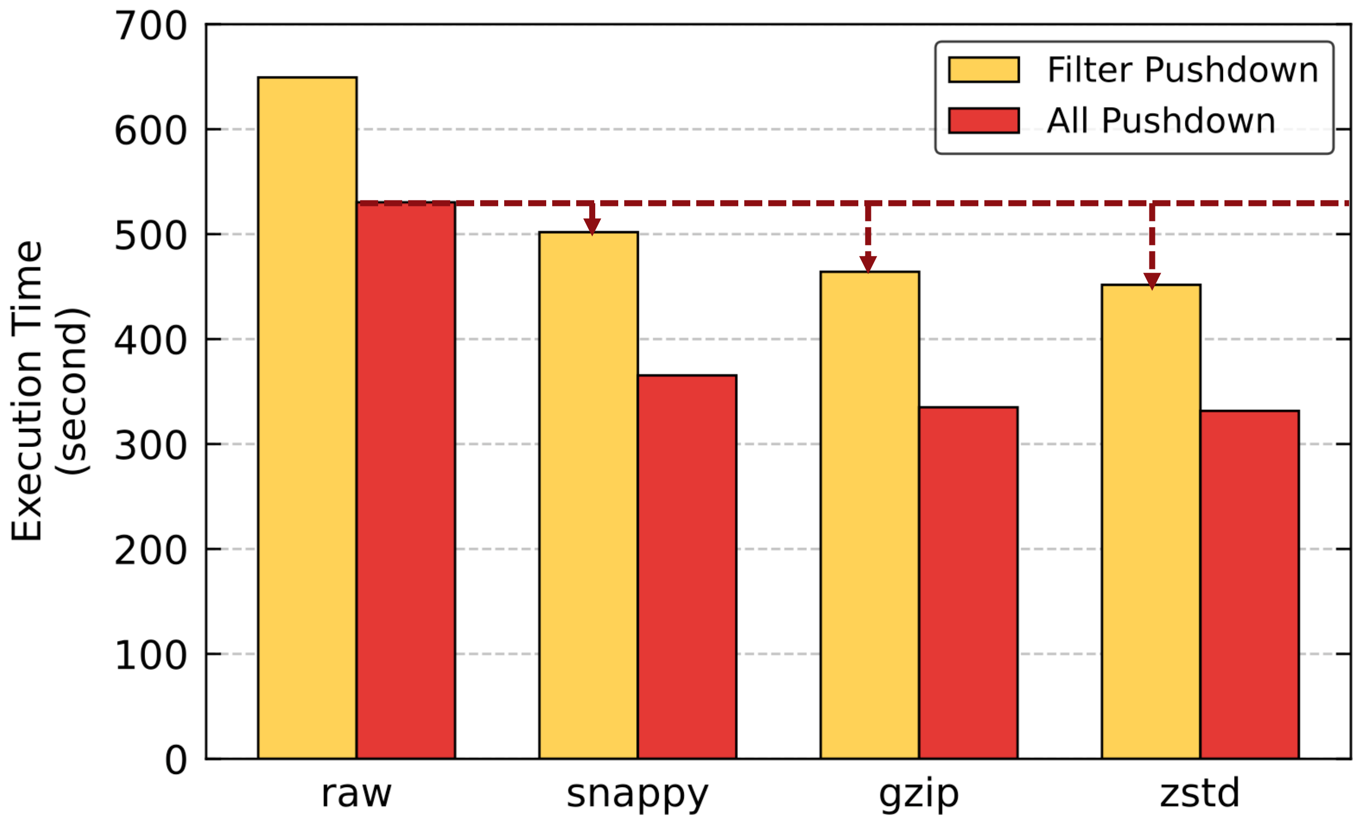
- Compressed filter-only (Zstd, 451.7s) outperforms uncompressed full pushdown (530.4s)
  - Demonstrates compression remains valuable even with advanced pushdown



Evaluation: Pushdown with Compression

- *Q3. How does OCS pushdown perform with data compression?*

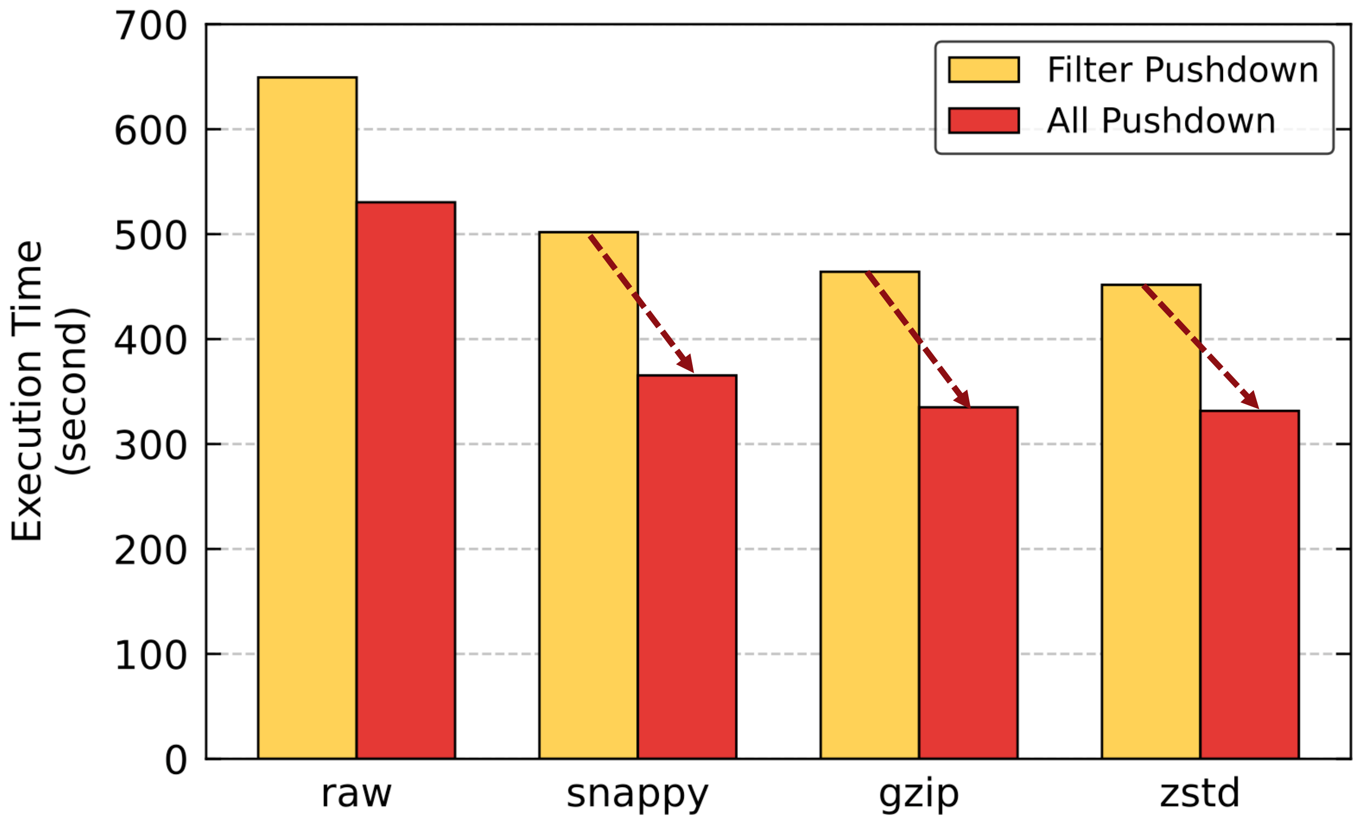
- Compressed filter-only (Zstd, 451.7s) outperforms uncompressed full pushdown (530.4s)
  - Demonstrates compression remains valuable even with advanced pushdown



Evaluation: Pushdown with Compression

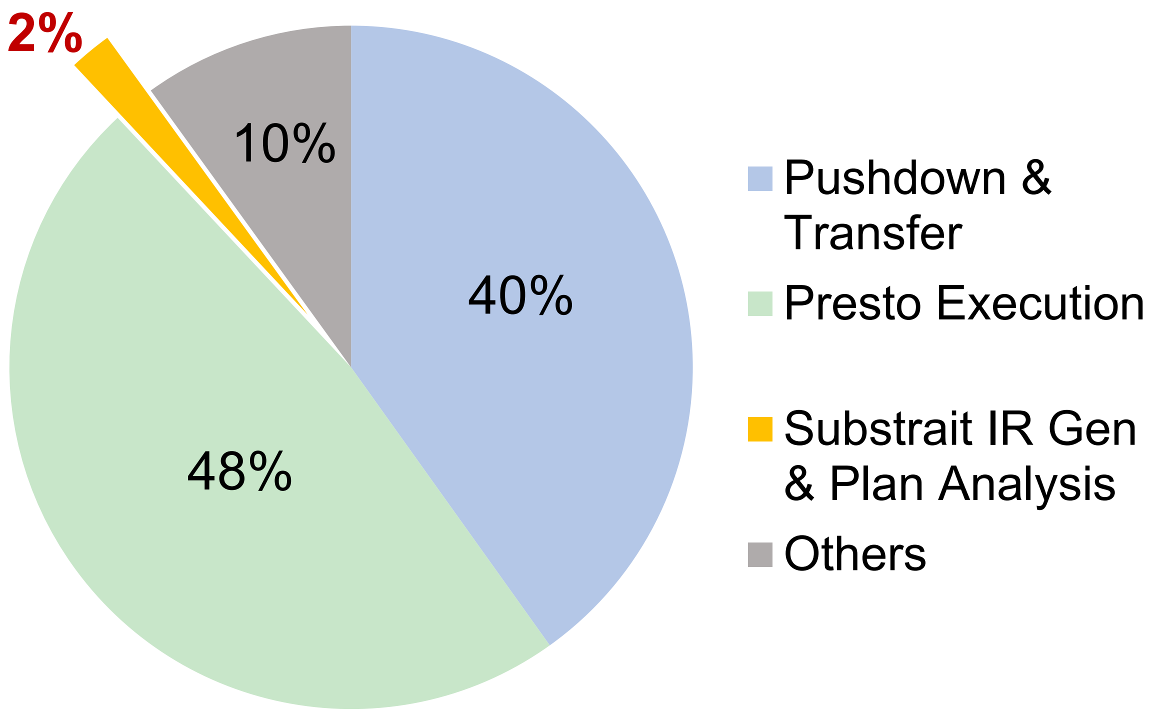
- *Q3. How does OCS pushdown perform with data compression?*

- OCS pushdown consistently outperforms within same compression scheme: (filter-only → full pushdown)
  - Snappy: 1.37× speedup
  - GZip: 1.39× speedup
  - Zstd: 1.36× speedup
  - Best result: Zstd + full pushdown = 331.6s (vs. 649.3s uncompressed filter-only)
- OCS-enabled operator pushdown **serves as a powerful complement** to existing compression techniques



SOGANG UNIVERSITY

Evaluation: Connector Overhead



Execution Stage	Time (ms)	Share (%)
Logical Plan Analysis	1	0.06%
Substrait IR Generation	33	1.94%
Pushdown & Result Transfer	682	40.12%
Presto Execution (Post-Scan)	814	47.90%
Others	169	9.97%
Total	1,700	100%

Negligible Overhead: Less than 2%

SOGANG UNIVERSITY

Conclusion

Problem:

- Existing object storage lacks full query pushdown support
- SK Hynix Object-based Computational Storage (OCS) provides advanced capabilities, but standard interfaces (Hive connector) cannot expose them

Summary:

- Designed Presto-OCS connector to integrate OCS with distributed SQL engines
- Leverage Presto's Service Provider Interface (SPI) framework for seamless integration without core modifications

Key Results:

- Up to 4.07× speedup and 99% data movement reduction
- 1.36-1.39× speedup over compressed filter-only pushdown
- Effective across HPC (Laghos, Deep Water) and OLAP (TPC-H) workloads
- OCS's advanced pushdown complements existing data reduction techniques like compression

SOGANG UNIVERSITY

Thank you!

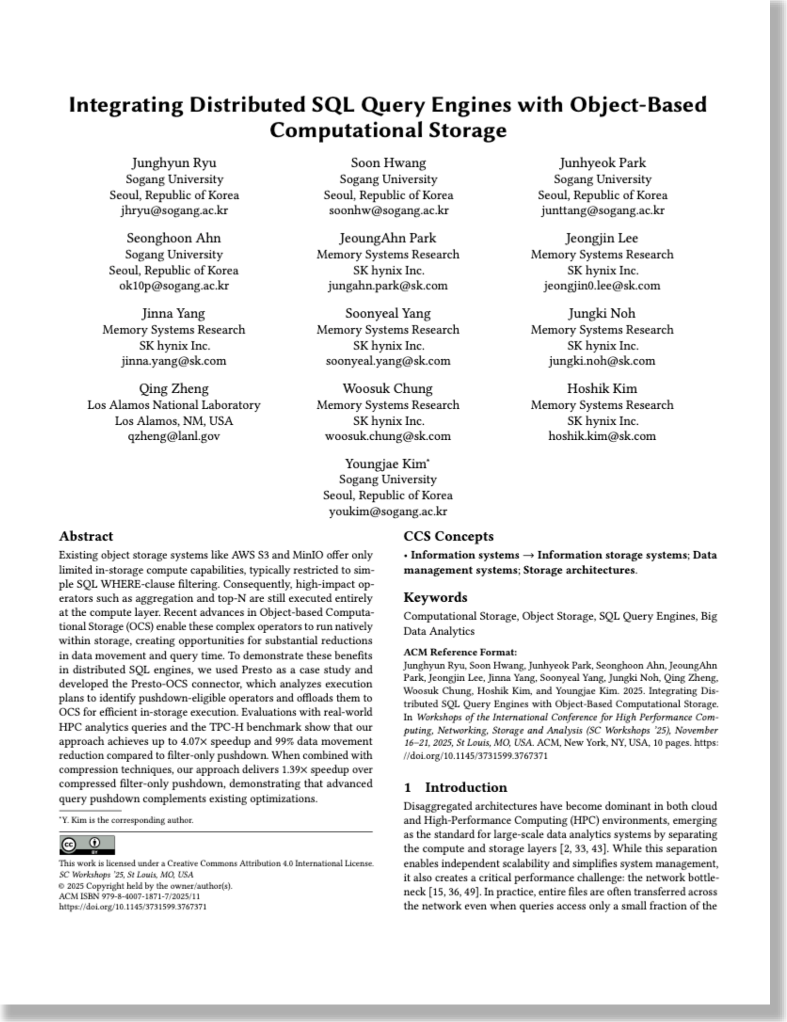
- Contact
  - Junghyun Ryu / jhryu@sogang.ac.kr
  - Youngjae Kim / youkim@sogang.ac.kr /

    https://sites.google.com/site/youkim/home

You can find more design and evaluation details in our paper.

We'd love for you to check it out!

<Camera-ready paper>



Integrating Distributed SQL Query Engines with Object-Based Computational Storage

Junghyun Ryu, Sogang University, Seoul, Republic of Korea, jhryu@sogang.ac.kr
Soon Hwang, Sogang University, Seoul, Republic of Korea, soonhw@sogang.ac.kr
Junhyeok Park, Sogang University, Seoul, Republic of Korea, junttang@sogang.ac.kr
Seonghoon Ahn, Sogang University, Seoul, Republic of Korea, ok10p@sogang.ac.kr
JeoungAhn Park, Memory Systems Research, SK hynix Inc., jungahn.park@sk.com
Jeongjin Lee, Memory Systems Research, SK hynix Inc., jeongjin0.lee@sk.com
Jinna Yang, Memory Systems Research, SK hynix Inc., jinna.yang@sk.com
Soonyeal Yang, Memory Systems Research, SK hynix Inc., soonyeal.yang@sk.com
Jungki Noh, Memory Systems Research, SK hynix Inc., jungki.noh@sk.com
Qing Zheng, Los Alamos National Laboratory, Los Alamos, NM, USA, qzheng@lanl.gov
Woosuk Chung, Memory Systems Research, SK hynix Inc., woosuk.chung@sk.com
Hoshik Kim, Memory Systems Research, SK hynix Inc., hoshik.kim@sk.com
Youngjae Kim*, Sogang University, Seoul, Republic of Korea, youkim@sogang.ac.kr

Abstract

Existing object storage systems like AWS S3 and MinIO offer only limited in-storage compute capabilities, typically restricted to simple SQL WHERE-clause filtering. Consequently, high-impact operators such as aggregation and top-N are still executed entirely at the compute layer. Recent advances in Object-based Computational Storage (OCS) enable these complex operators to run natively within storage, creating opportunities for substantial reductions in data movement and query time. To demonstrate these benefits in distributed SQL engines, we used Presto as a case study and developed the Presto-OCS connector, which analyzes execution plans to identify pushdown-eligible operators and offloads them to OCS for efficient in-storage execution. Evaluations with real-world HPC analytics queries and the TPC-H benchmark show that our approach achieves up to 4.07× speedup and 99% data movement reduction compared to filter-only pushdown. When combined with compression techniques, our approach delivers 1.39× speedup over compressed filter-only pushdown, demonstrating that advanced query pushdown complements existing optimizations.

*Y. Kim is the corresponding author.

SC Workshops '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1871-7/2025/11
https://doi.org/10.1145/3731599.3767371

CCS Concepts

• **Information systems** → **Information storage systems**; **Data management systems**; **Storage architectures**.

Keywords

Computational Storage, Object Storage, SQL Query Engines, Big Data Analytics

ACM Reference Format:
Junghyun Ryu, Soon Hwang, Junhyeok Park, Seonghoon Ahn, JeoungAhn Park, Jeongjin Lee, Jinna Yang, Soonyeal Yang, Jungki Noh, Qing Zheng, Woosuk Chung, Hoshik Kim, and Youngjae Kim. 2025. Integrating Distributed SQL Query Engines with Object-Based Computational Storage. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3731599.3767371

1 Introduction

Disaggregated architectures have become dominant in both cloud and High-Performance Computing (HPC) environments, emerging as the standard for large-scale data analytics systems by separating the compute and storage layers [2, 33, 43]. While this separation enables independent scalability and simplifies system management, it also creates a critical performance challenge: the network bottleneck [15, 36, 49]. In practice, entire files are often transferred across the network even when queries access only a small fraction of the

Appendix

Implementation: Selectivity Estimation

Statistical Approach Using Hive Metastore

- **Filter Selectivity**
  - Assumes normal distribution between min/max column values
  - Estimates proportion of rows within query's range predicate
  - Example: WHERE value BETWEEN 100 AND 200
- **Aggregation Selectivity**
  - Output cardinality = row count / NDV of GROUP BY columns
  - Low NDV → High data reduction → Prioritized for pushdown
  - Example: GROUP BY status (3 distinct values) → ~33% of rows
- **Top-N Selectivity**
  - Direct calculation: LIMIT value / total row count
  - Example: ORDER BY score LIMIT 100 from 1M rows → 0.01%

Implementation: Translation & Communication

Substrait IR Generation Process

- Reconstruct operators into SQL statements
- SQL clauses → Substrait relations (standardized representation)
- Type normalization: Handle nulls, decimals, timestamps
- Function mapping: Presto functions → Substrait namespace
- Serialize using Protocol Buffers

Communication & Result Processing

- gRPC: High-performance RPC for Substrait plan transmission
- OCS executes via embedded SQL engine on storage nodes
- Apache Arrow: Efficient columnar format for result serialization
- Connector deserializes Arrow → Presto's internal page format