Whiz: Data-driven Analytics Execution

Robert Grandl*, **Arjun Singhvi***, Raajay Viswanathan, Aditya Akella

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

* = co-primary authors

Problem Statement

Data analytics frameworks are used in diverse settings to analyze large datasets

Underlying compute-centric execution engines hinder performance and efficiency:

- Intermediate data unawareness
- Static parallelism and intermediate data partitioning
- Compute-driven scheduling
- Compute-based intermediate data organization

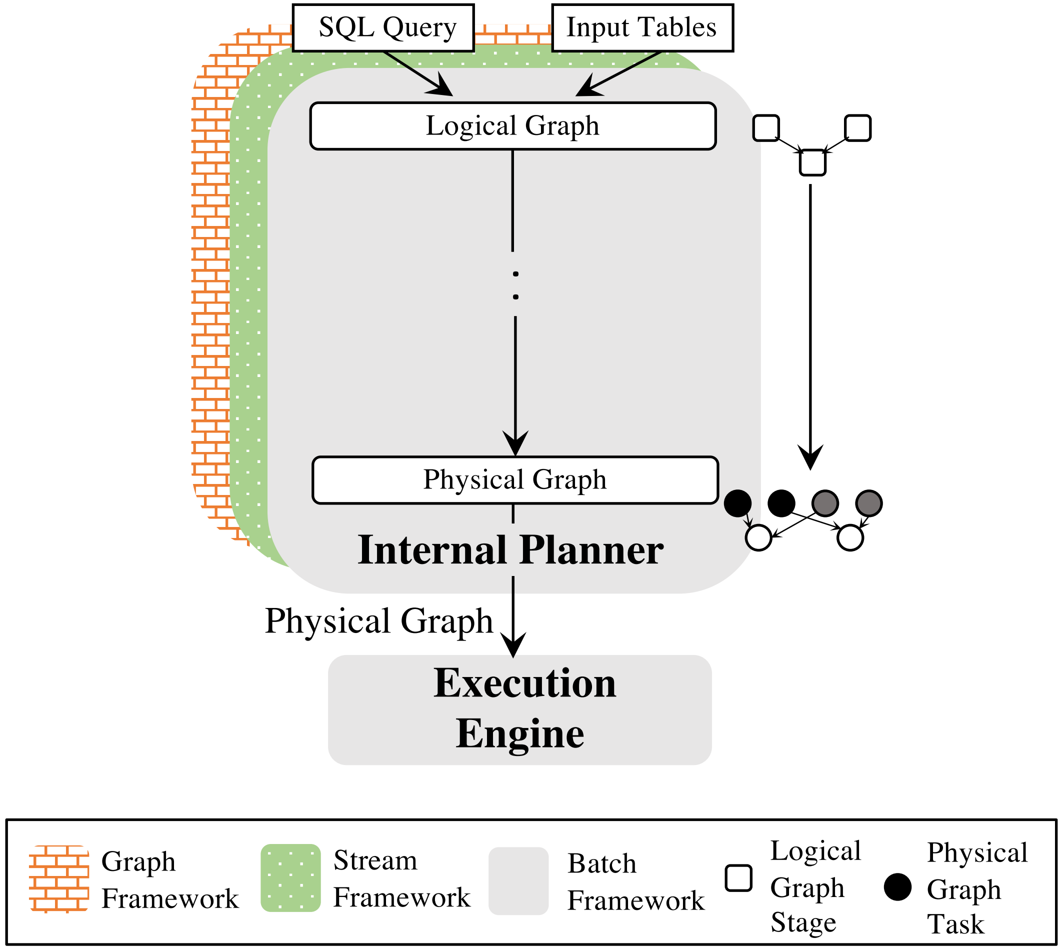
How do we overcome *all* these limitations of compute-centric execution engines?

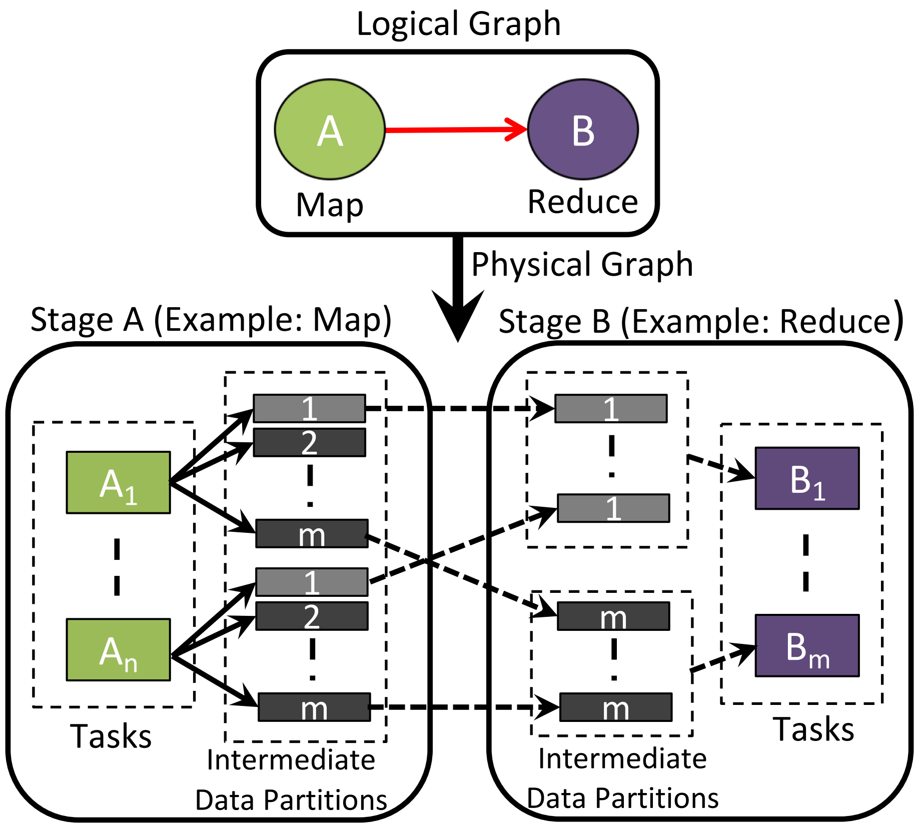
↓

Whiz

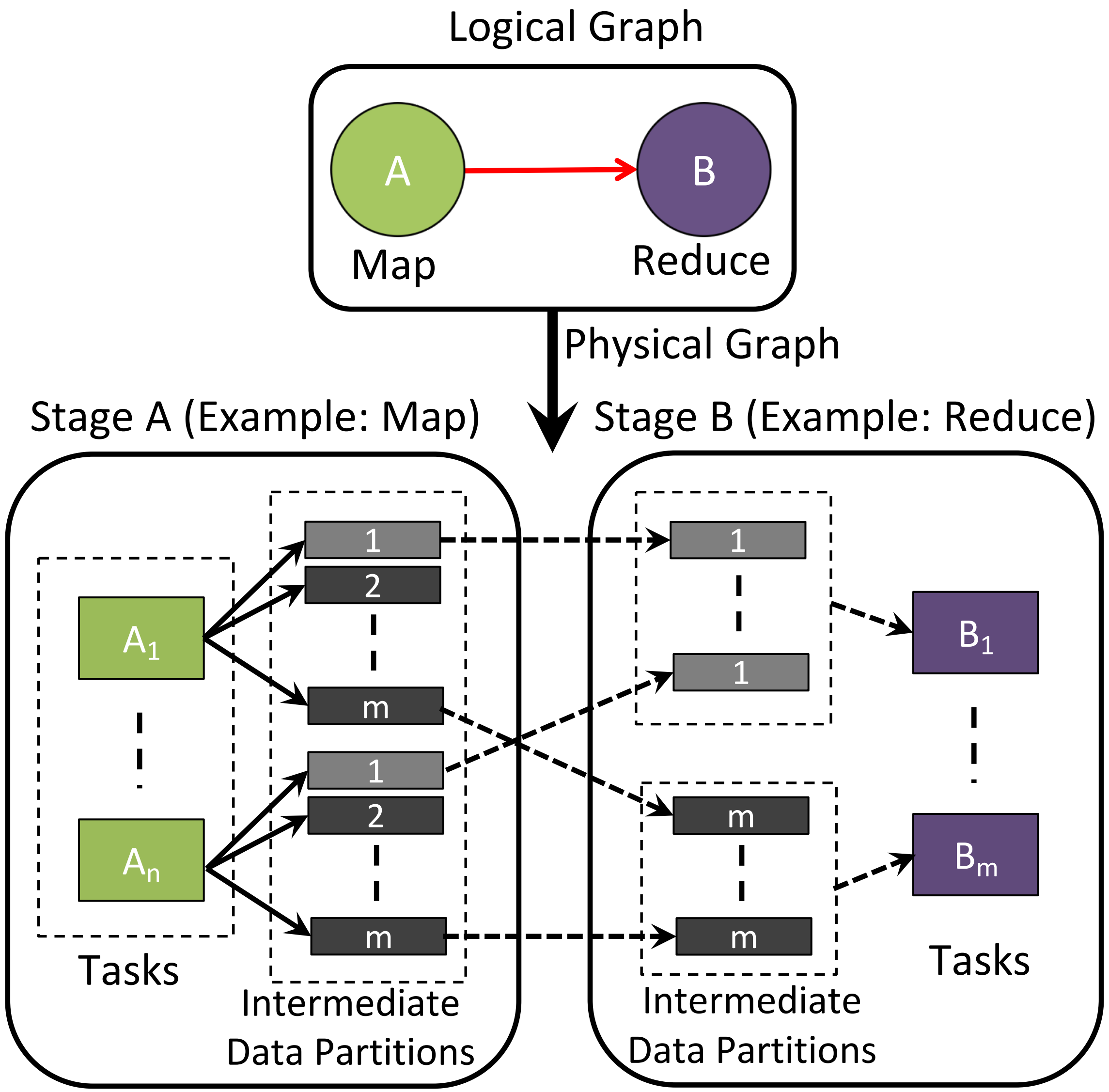
Data Analytics Frameworks 101

Diverse analytics frameworks exist today (e.g., batch, stream, graph)



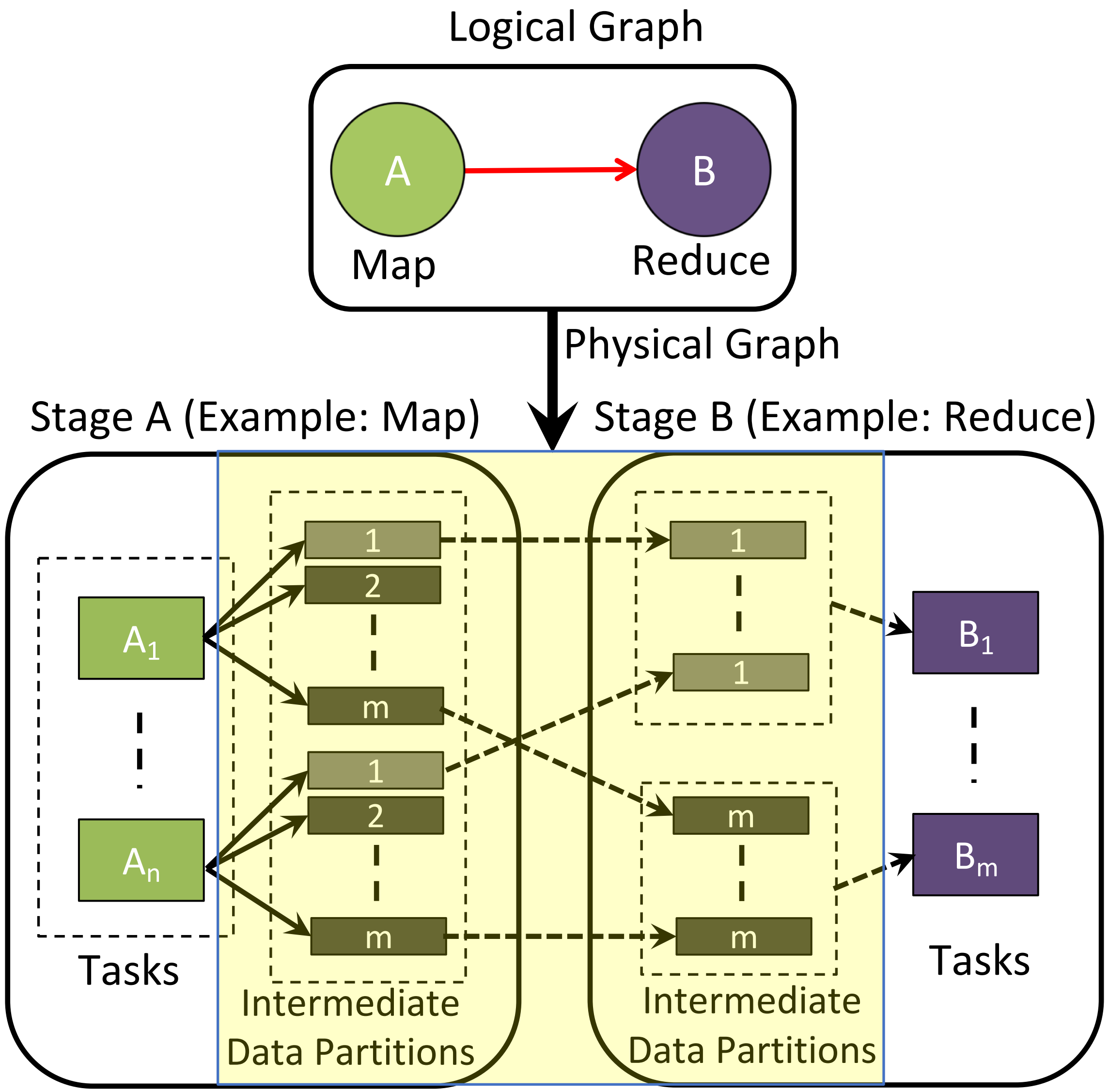


Analytics Limitation #1: Data Opacity + Compute Rigidity



Execution engine handles management of all intermediate data and how it is accessed

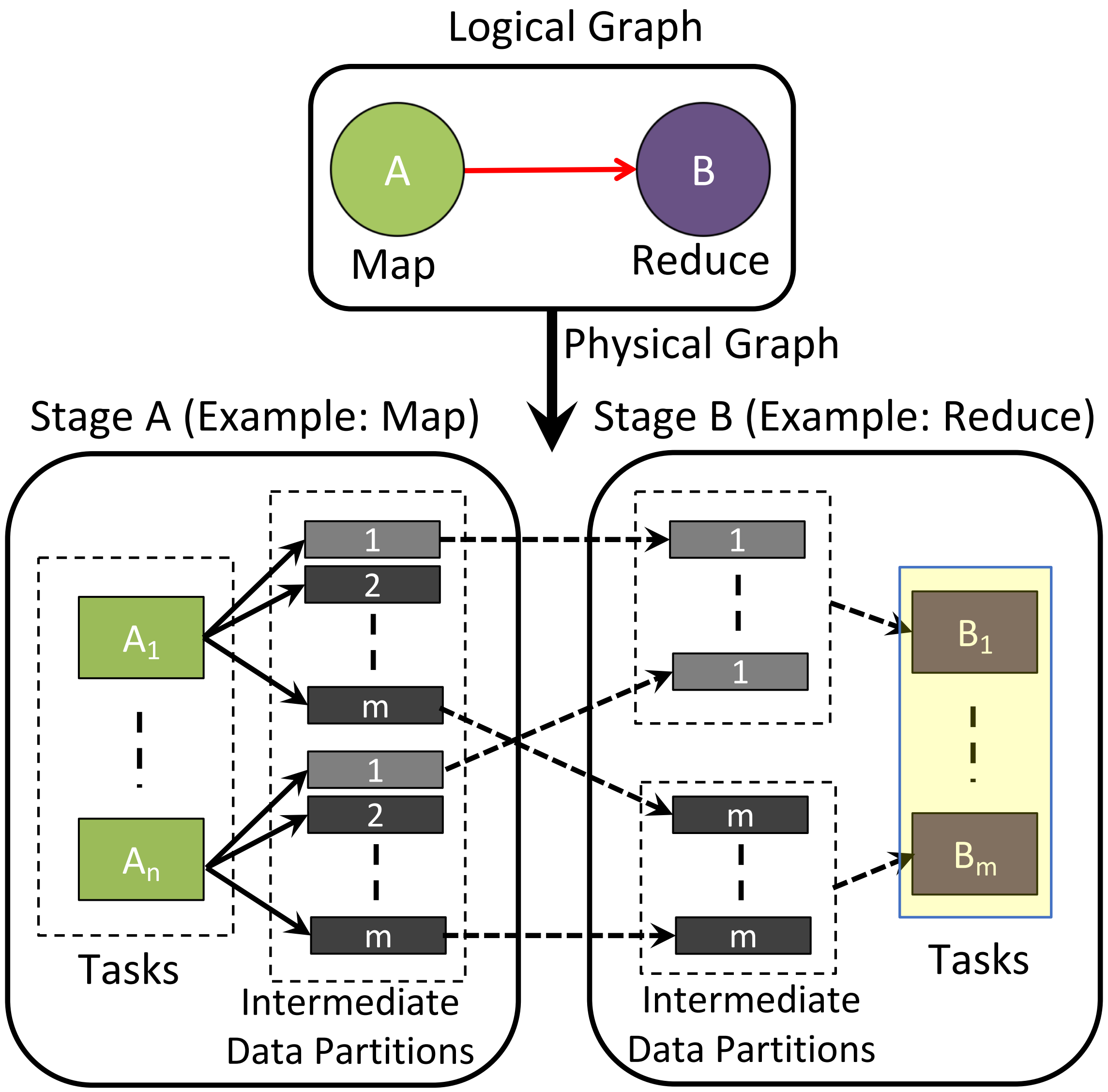
Analytics Limitation #1: Data Opacity + Compute Rigidity



Execution engine handles management of all intermediate data and how it is accessed

Execution engine has limited runtime visibility into intermediate data

Analytics Limitation #1: Data Opacity + Compute Rigidity



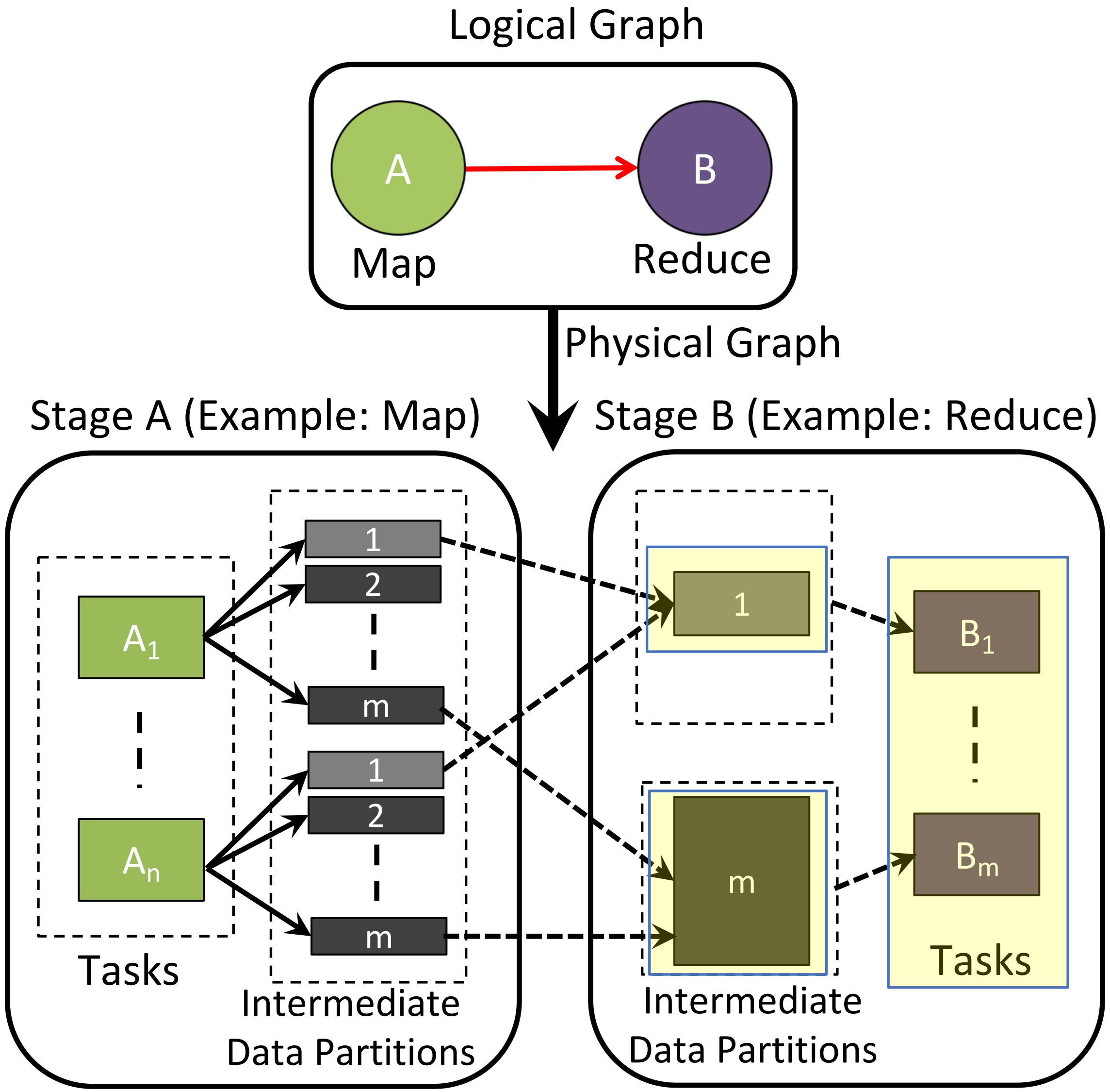
Execution engine handles management of all intermediate data and how it is accessed

Execution engine has limited runtime visibility into intermediate data

⬇

Cannot change the processing logic of a task depending on intermediate data

Analytics Limitation #2: Static Execution Structure



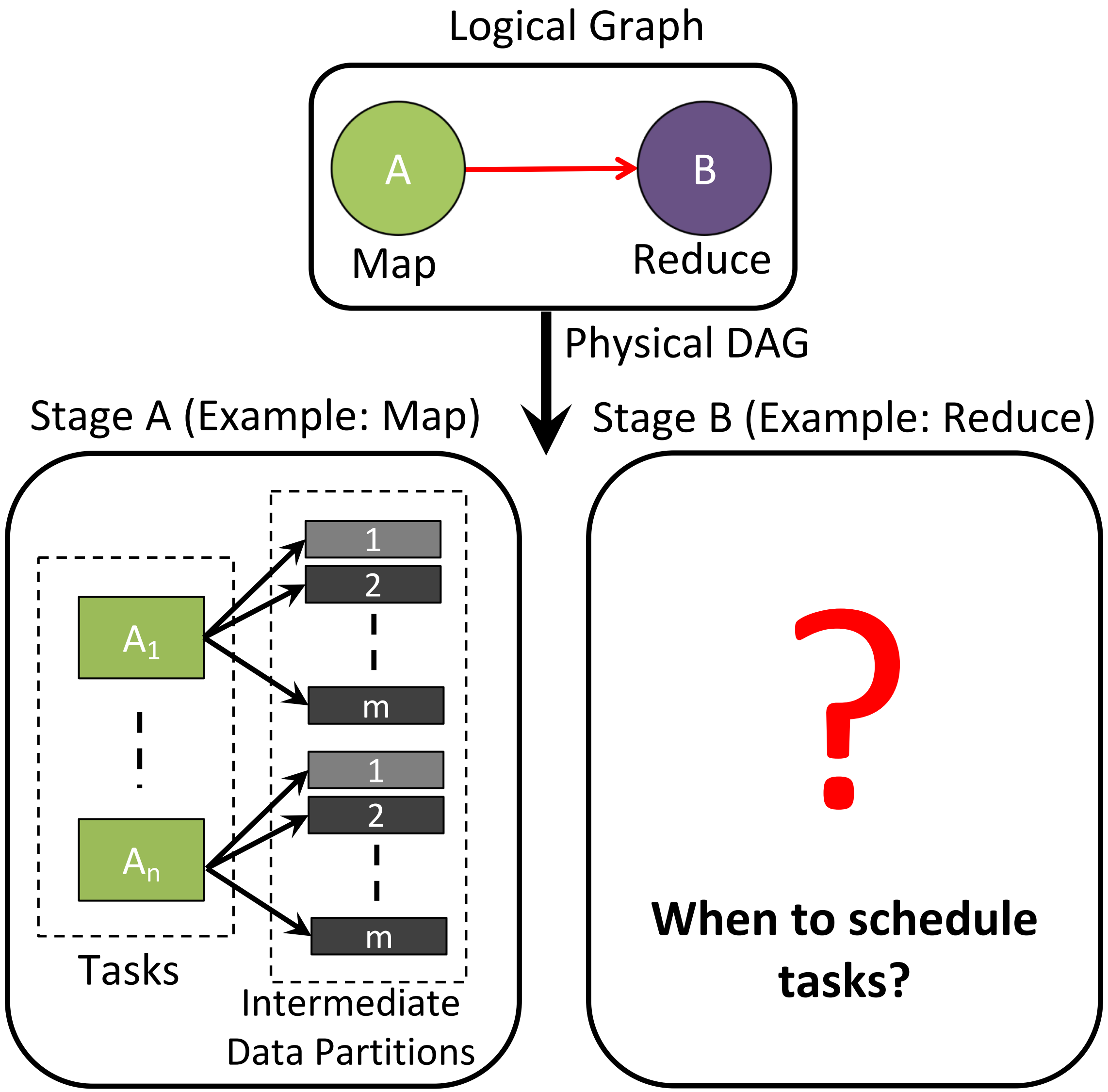
Task parallelism and **intermediate data partitioning strategy** needed by execution engine is often **static**

⬇

Data **skew** can lead to **degraded performance**

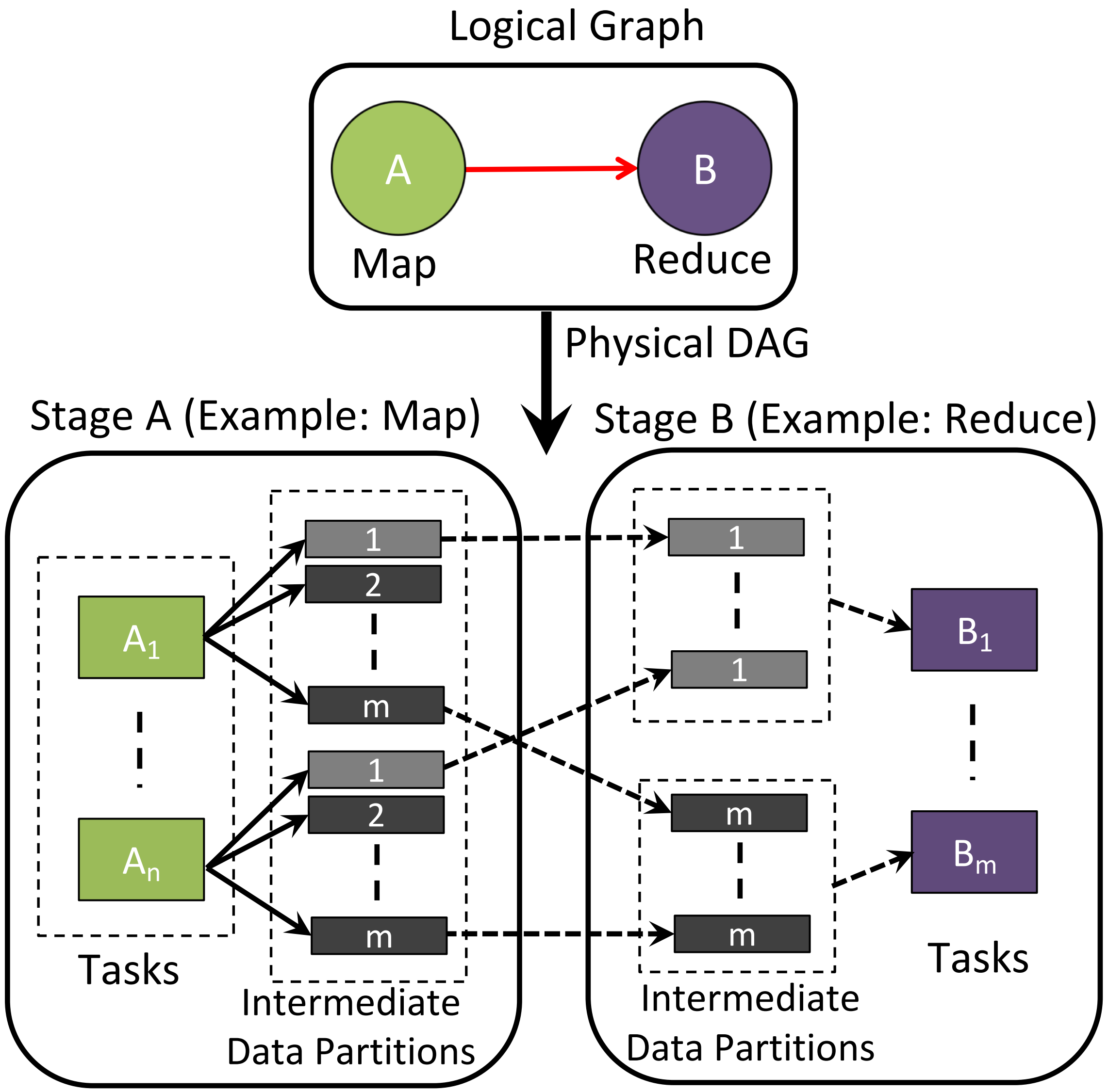
Inadaptable to **resource changes**

Analytics Limitation #3: Compute-driven Scheduling



When to schedule tasks?

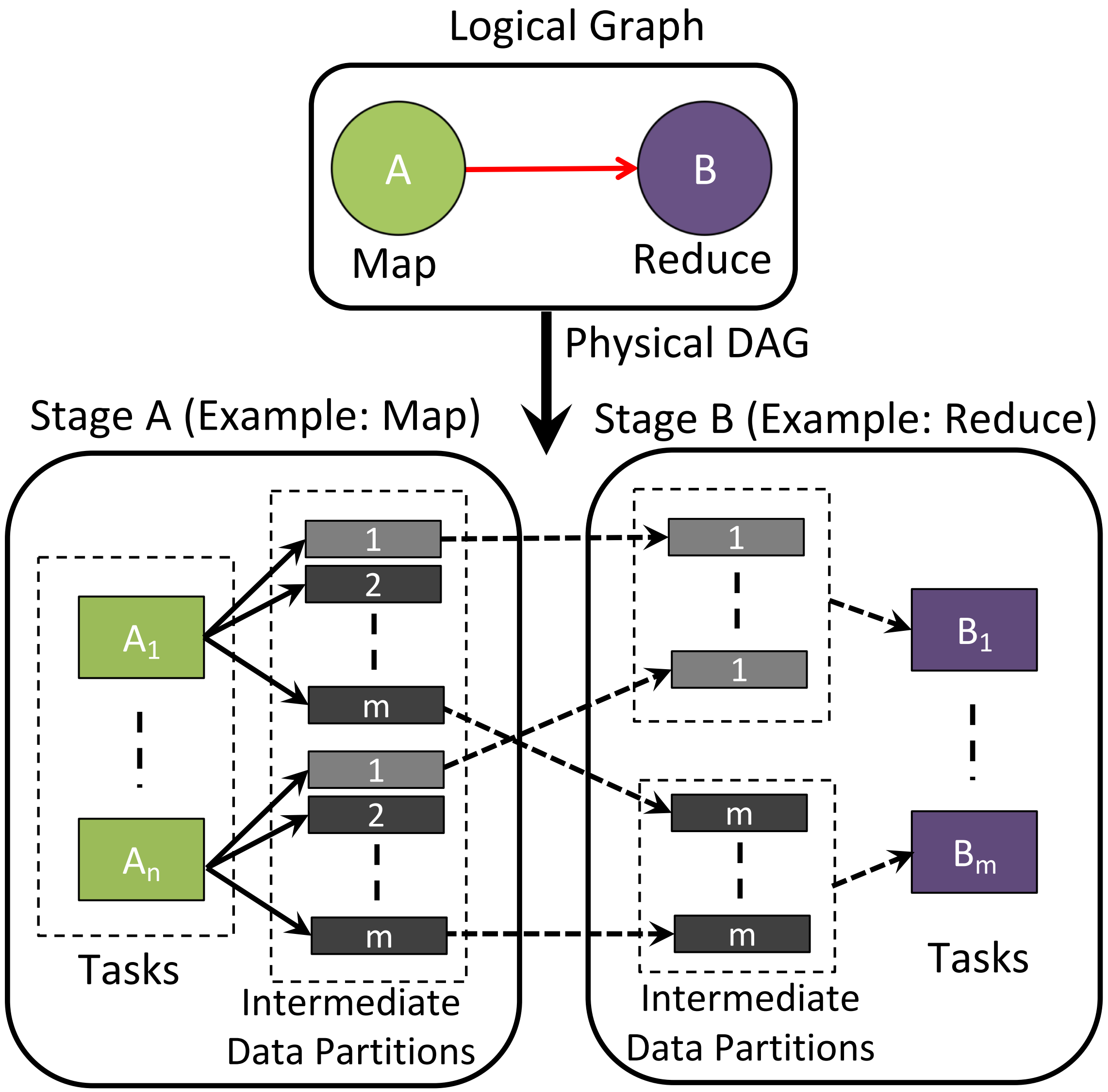
Analytics Limitation #3: Compute-driven Scheduling



Decisions regarding when to schedule tasks of downstream stage are based on static compute structure

For example: Schedule after x% of the upstream tasks are completed (commutative+associative logic)

Analytics Limitation #3: Compute-driven Scheduling



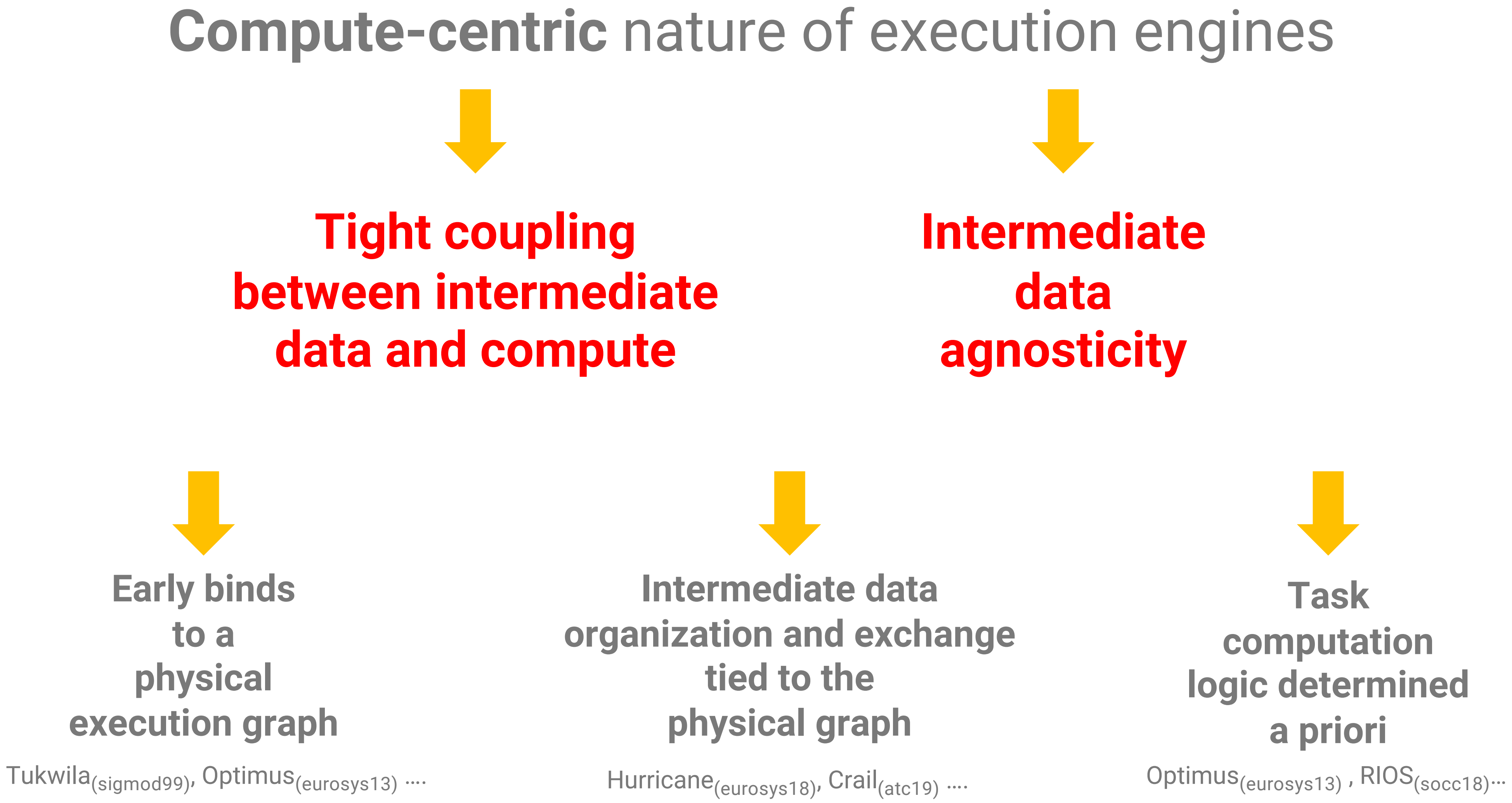
Decisions regarding when to schedule tasks of downstream stage are based on static compute structure

For example: Schedule after x% of the upstream tasks are completed (commutative+associative logic)

⬇

May lead to **compute idling** waiting for remaining data to be available

Analytics Limitations: Root Causes



Compute-centric nature of execution engines

Tight coupling between intermediate data and compute

Intermediate data agnosticity

Early binds to a physical execution graph

Tukwila$_{\text{(sigmod99)}}$, Optimus$_{\text{(eurosys13)}}$ ....

Intermediate data organization and exchange tied to the physical graph

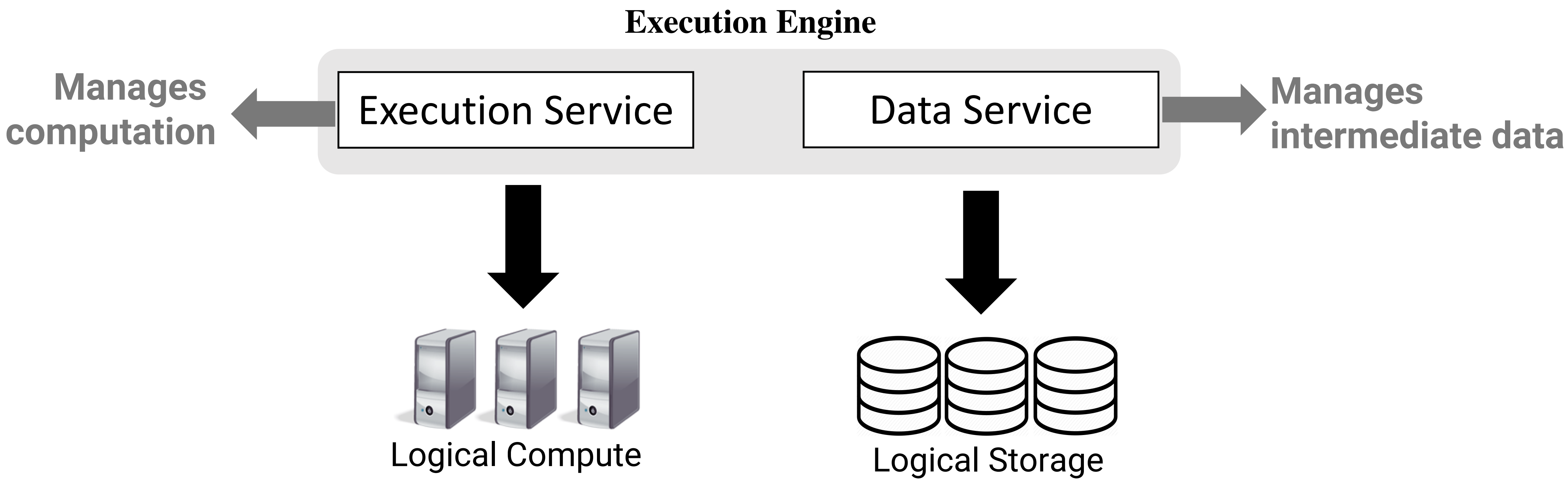
Hurricane$_{\text{(eurosys18)}}$, Crail$_{\text{(atc19)}}$ ....

Task computation logic determined a priori

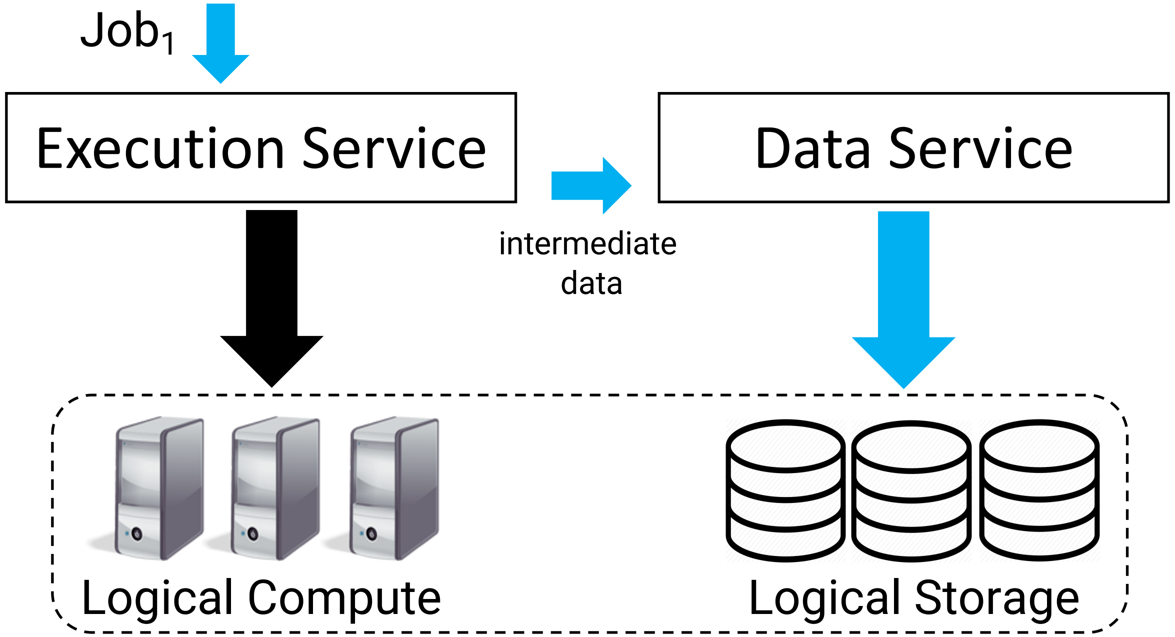
Optimus$_{\text{(eurosys13)}}$, RIOS$_{\text{(socc18)}}$...

Whiz Approach

Make intermediate data and compute **equal entities** during job execution by a clean **logical separation** between computation and intermediate data



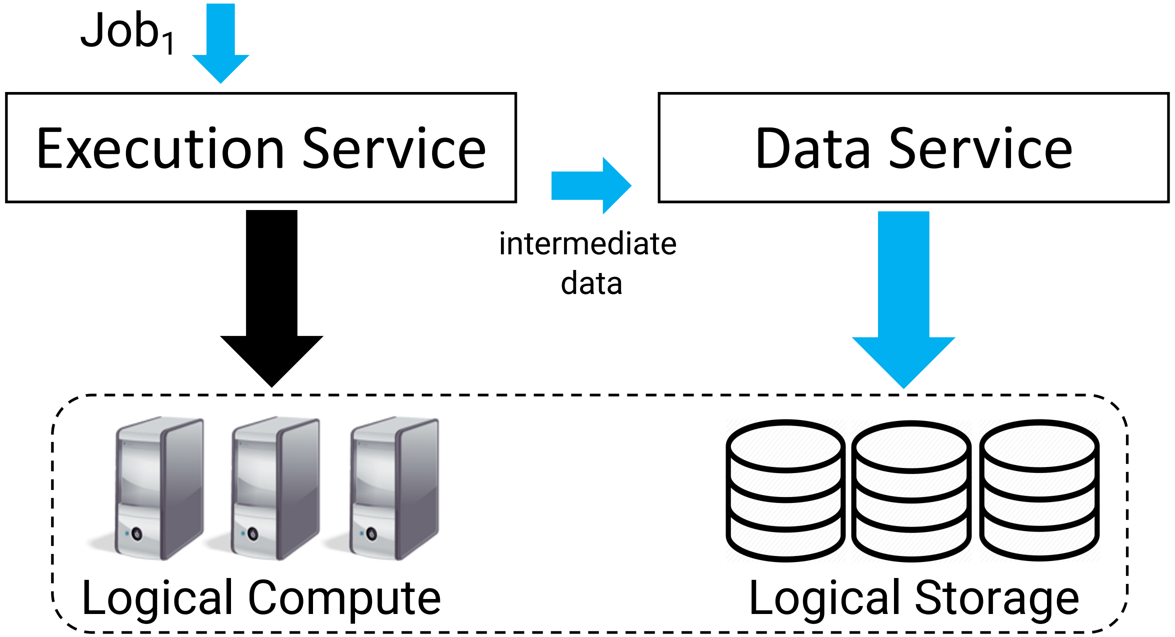
Whiz Key Idea #1: Intermediate Data Visibility



Decoupling enables intermediate data awareness

Data Service **gathers custom runtime properties** of intermediate data

Whiz Key Idea #1: Intermediate Data Visibility

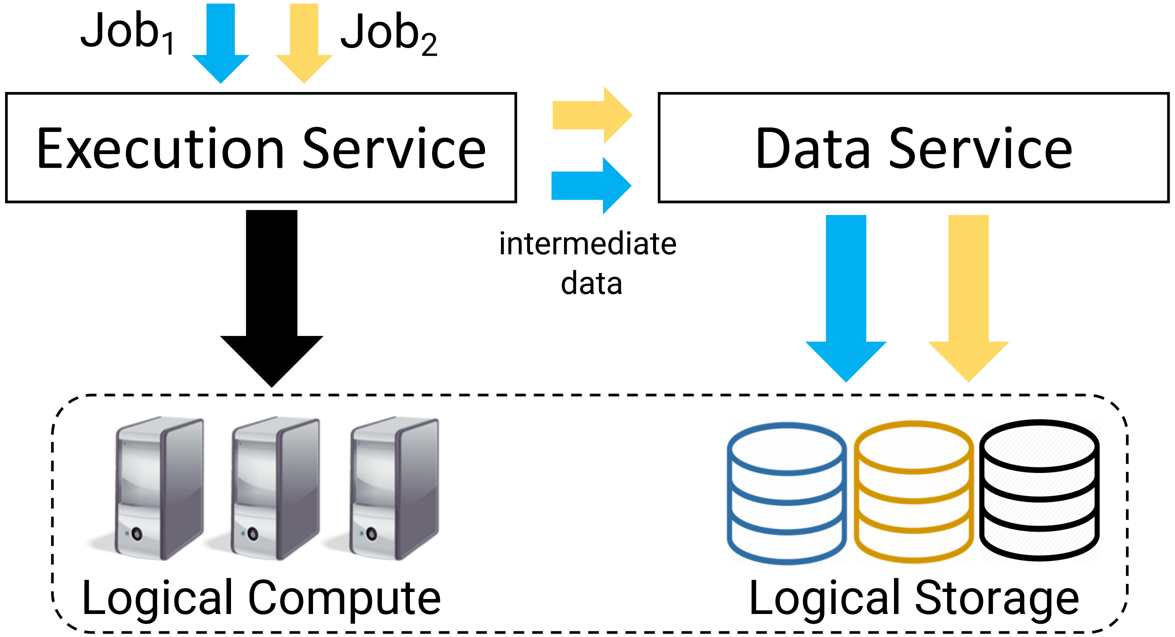


Decoupling enables intermediate data awareness

Data Service **gathers custom runtime properties** of intermediate data

Enables driving all aspects of job execution based on data properties

Whiz Key Idea #1: Intermediate Data Visibility



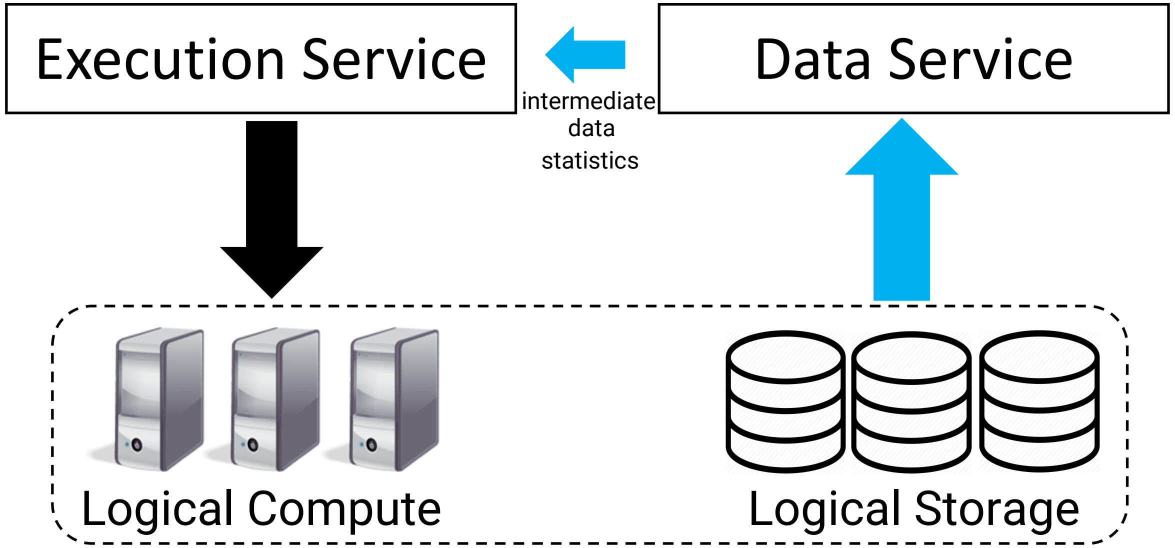
Decoupling enables intermediate data awareness

Data Service **gathers custom runtime properties** of intermediate data

Enables driving all aspects of job execution based on data properties

Intrinsically provides cross-job isolation and avoids I/O hotspots

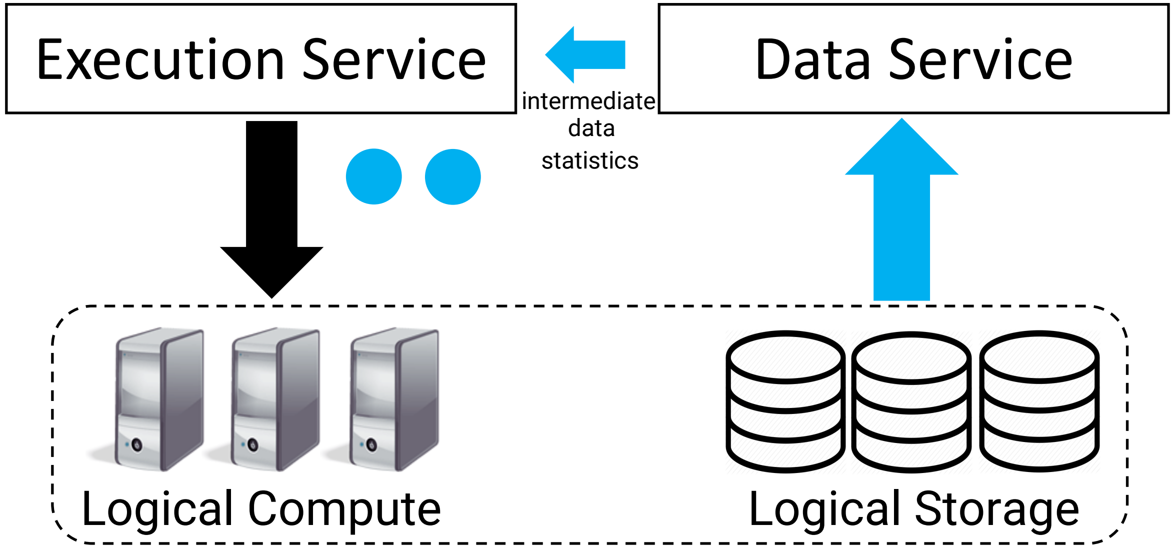
Whiz Key Idea #2: Runtime Physical Graph Generation



Decides the task parallelism and task sizing **based on data properties**

- Track intermediate data partition sizes

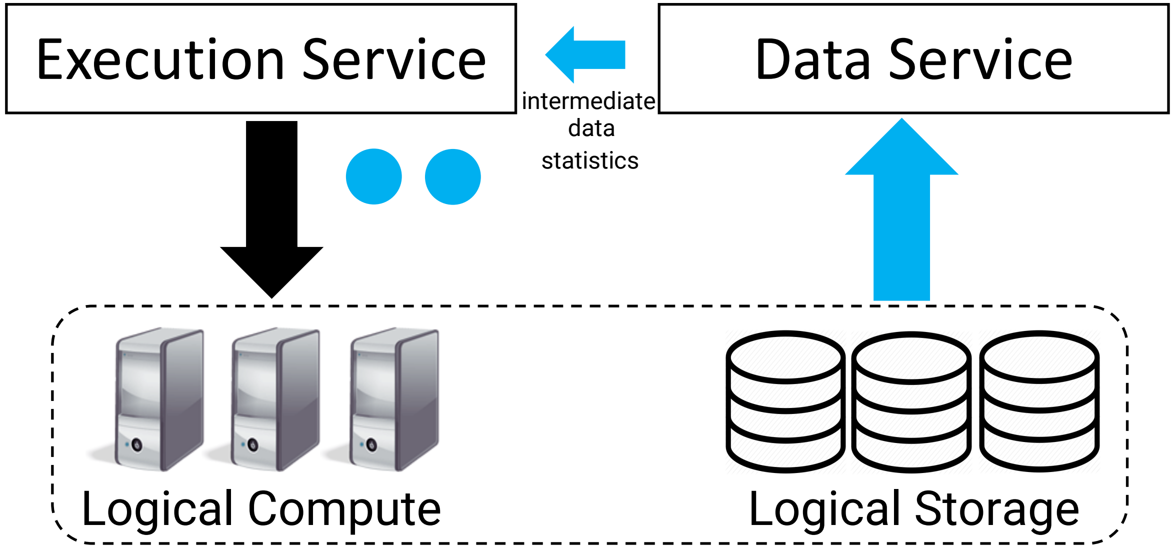
Whiz Key Idea #2: Runtime Physical Graph Generation



Decides the task parallelism and task sizing **based on data properties**

- Track intermediate data partition sizes

Whiz Key Idea #2: Runtime Physical Graph Generation



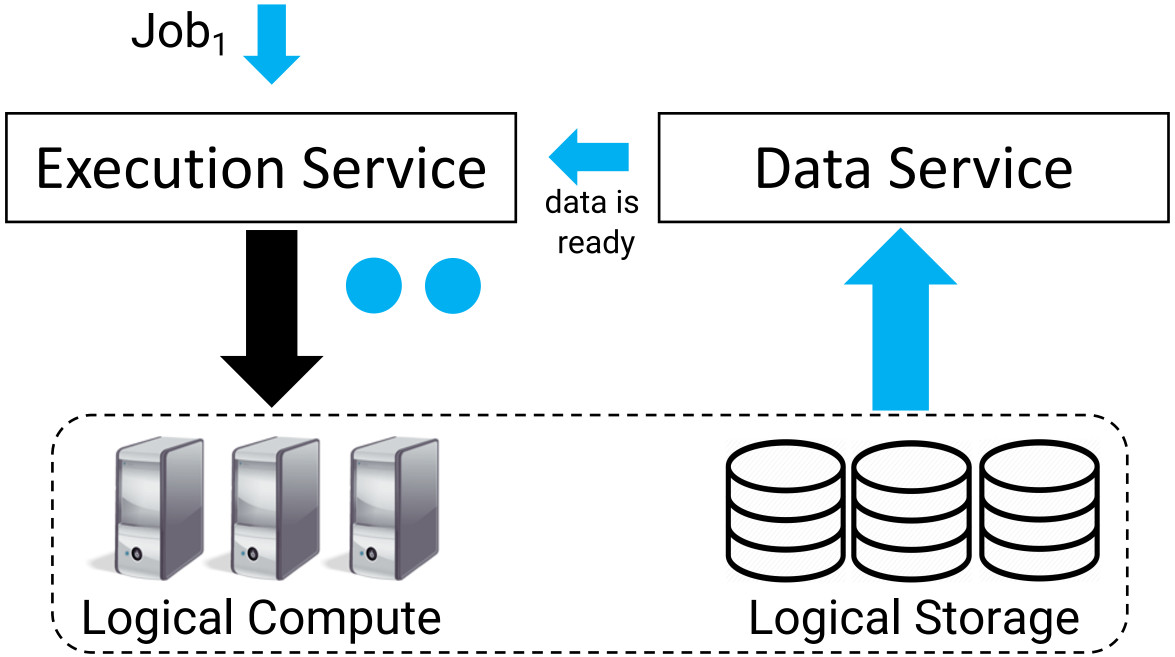
Decides the task parallelism and task sizing **based on data properties**

- Track intermediate data partition sizes

Enables handling intermediate data skew

Allows adapting to resource flux

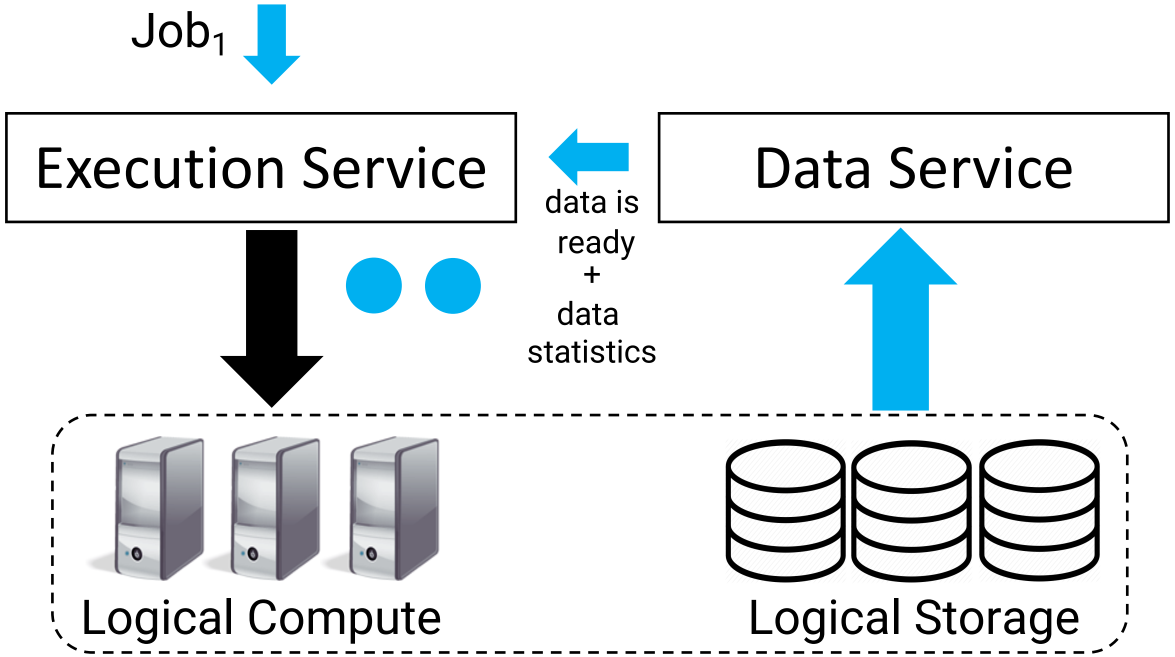
Whiz Key Idea #3: Data-driven Computation



Schedule computation based on intermediate data properties - when **data meets pre-defined execution predicates**

Leads to efficient use of resources

Whiz Key Idea #3: Data-driven Computation

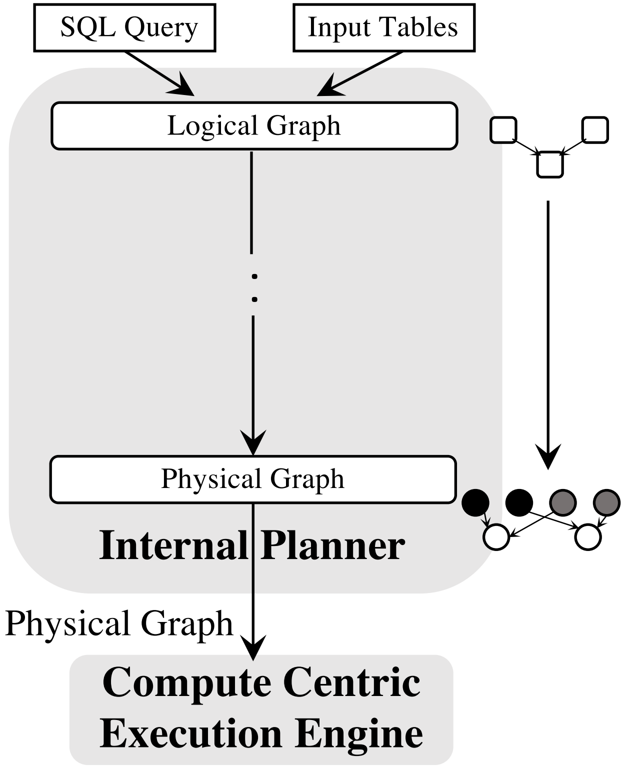


Schedule computation based on intermediate data properties - when **data meets pre-defined execution predicates**

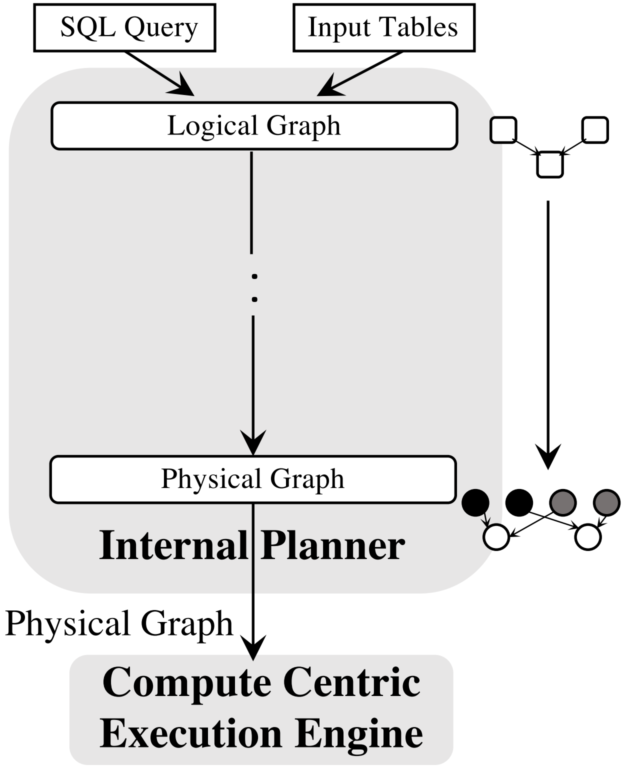
Leads to efficient use of resources

Determine **exact task logic based on intermediate data properties** at runtime

Whiz Job Execution Pipeline



Whiz Job Execution Pipeline



SQL Query

Input Tables

Logical Graph

Physical Graph

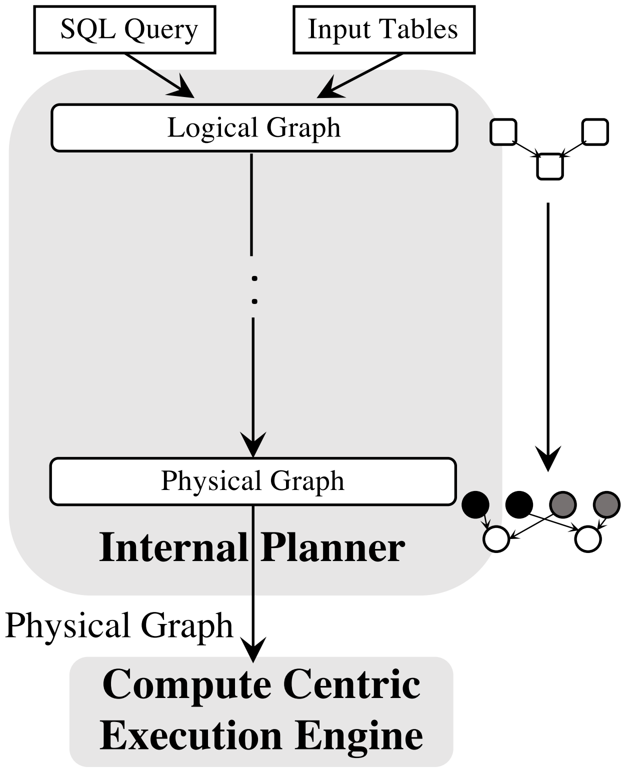
Internal Planner

Physical Graph

Compute Centric Execution Engine

Whiz Data Driven Execution Engine

Whiz Job Execution Pipeline



SQL Query

Input Tables

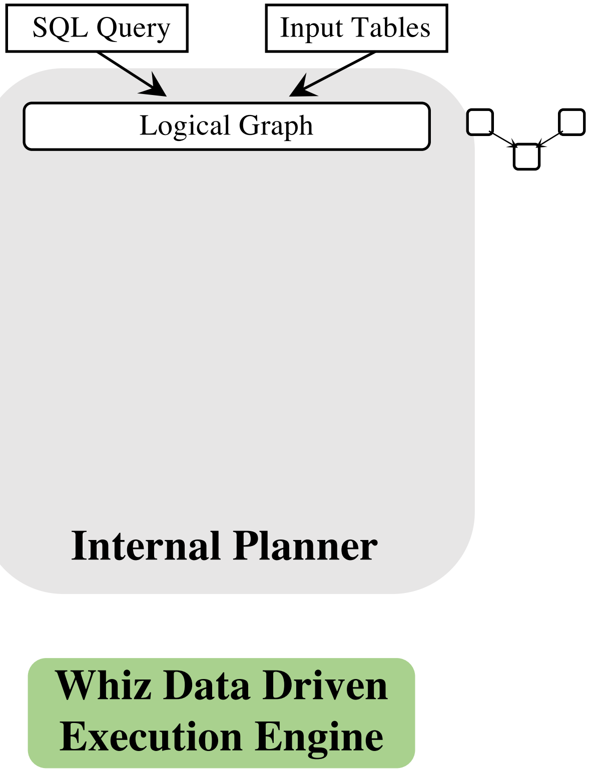
Logical Graph

Physical Graph

Internal Planner

Physical Graph

Compute Centric Execution Engine



SQL Query

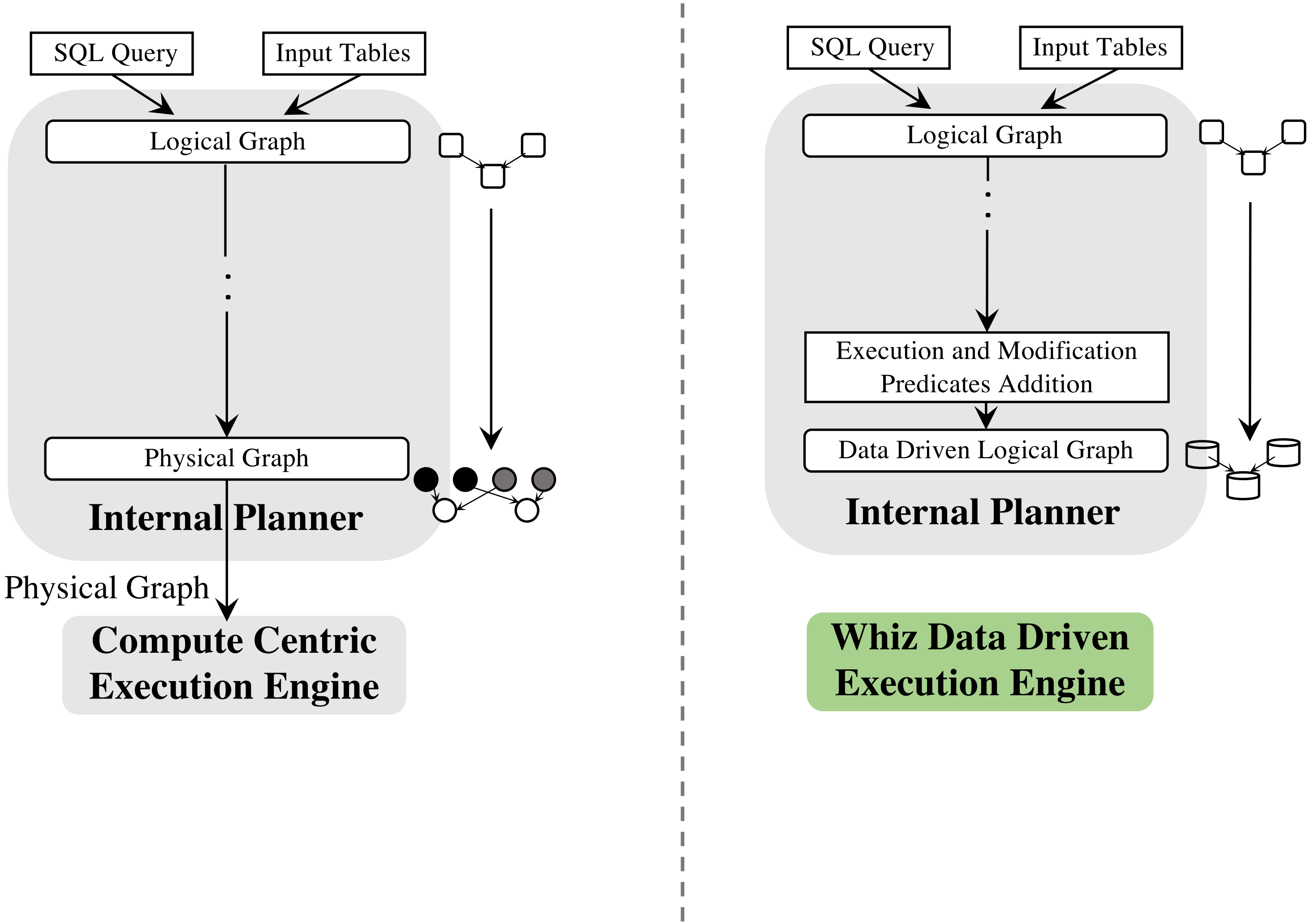
Input Tables

Logical Graph

Internal Planner

Whiz Data Driven Execution Engine

Whiz Job Execution Pipeline



SQL Query

Input Tables

Logical Graph

Physical Graph

Internal Planner

Physical Graph

Compute Centric Execution Engine

SQL Query

Input Tables

Logical Graph

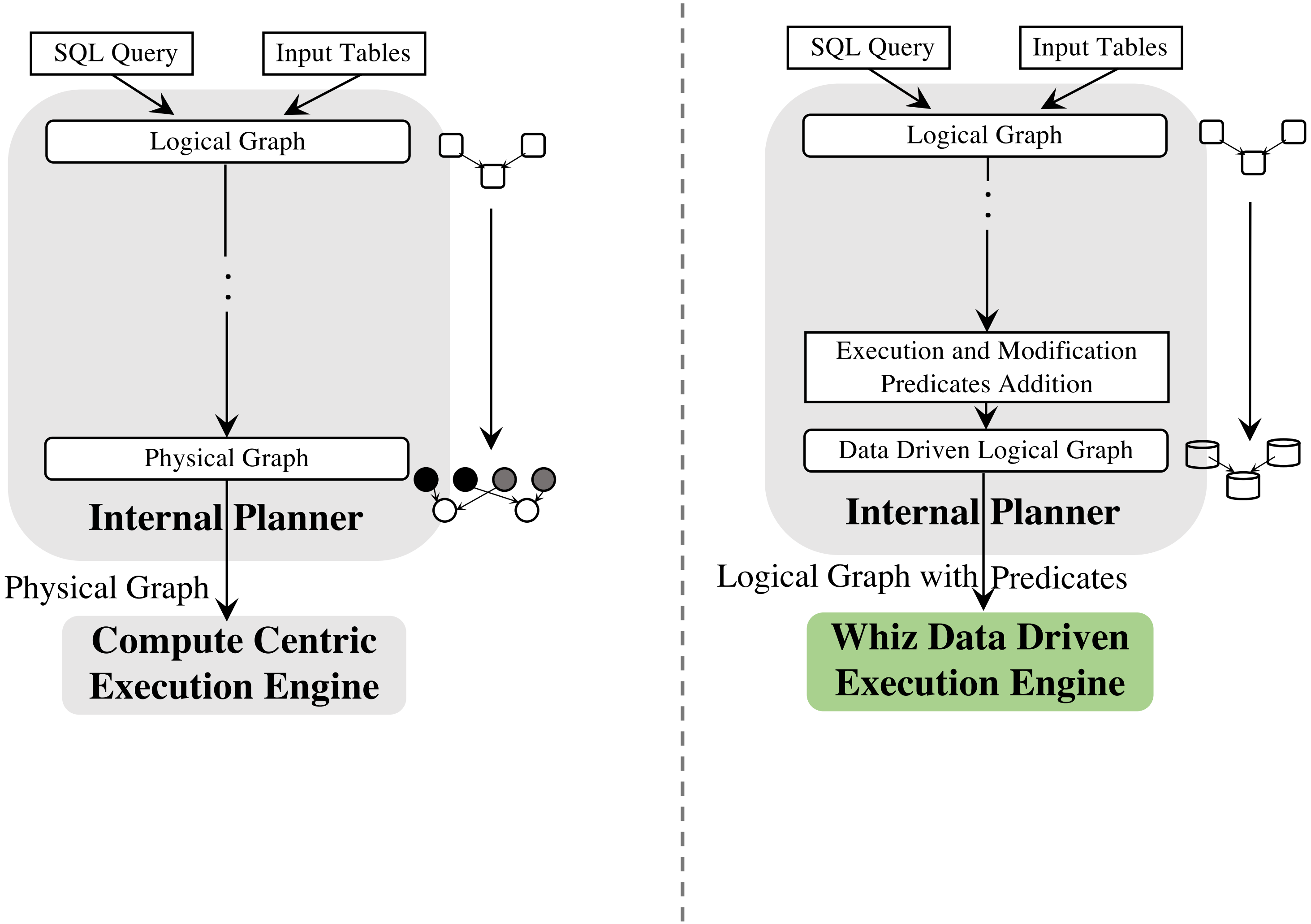
Execution and Modification Predicates Addition

Data Driven Logical Graph

Internal Planner

Whiz Data Driven Execution Engine

Whiz Job Execution Pipeline



SQL Query

Input Tables

Logical Graph

Physical Graph

Internal Planner

Physical Graph

Compute Centric Execution Engine

SQL Query

Input Tables

Logical Graph

Execution and Modification Predicates Addition

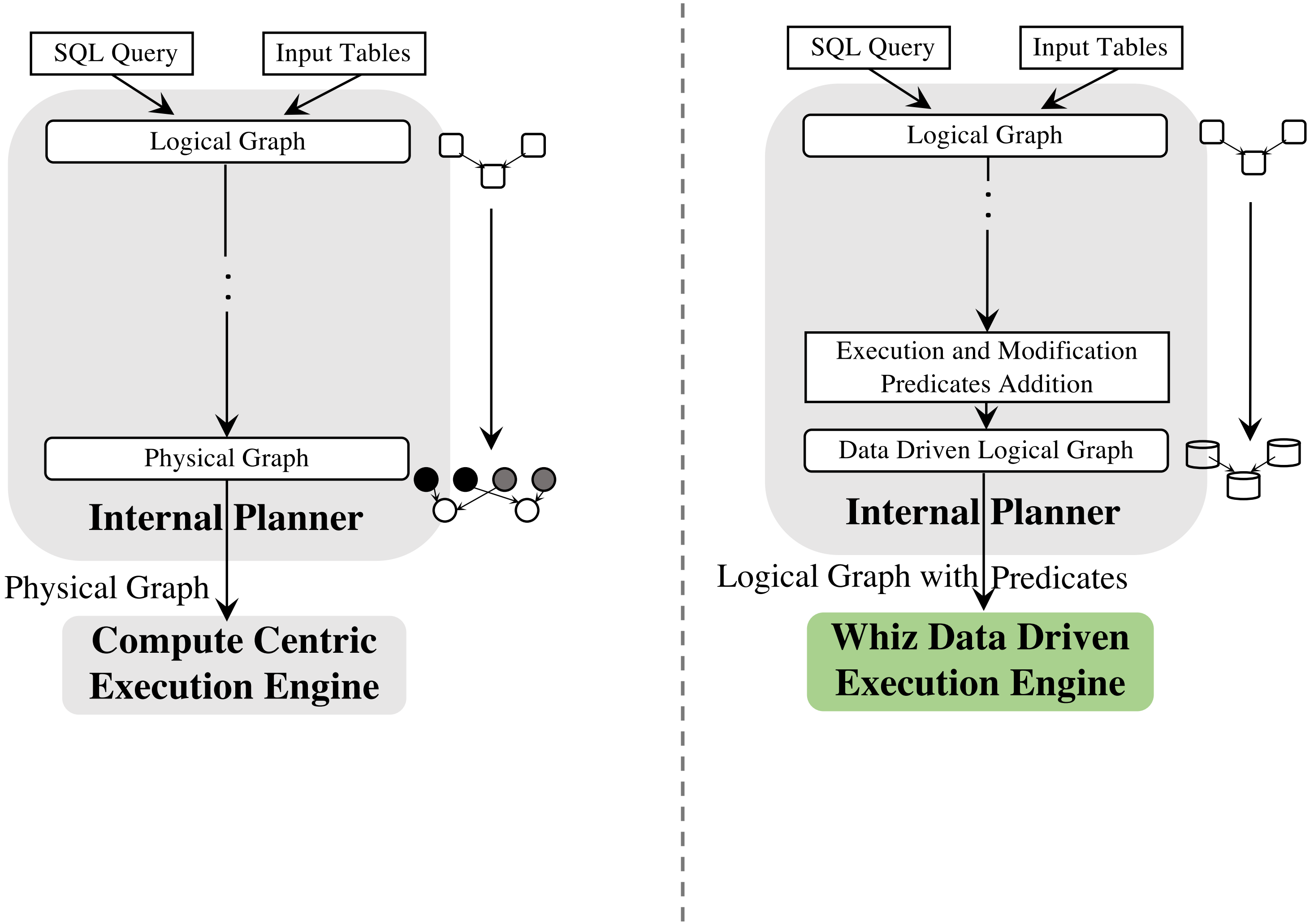
Data Driven Logical Graph

Internal Planner

Logical Graph with Predicates

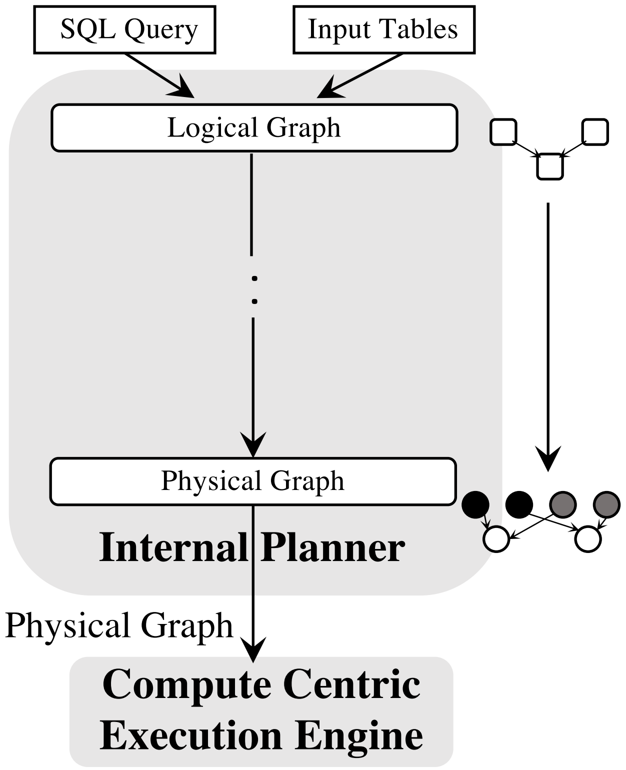
Whiz Data Driven Execution Engine

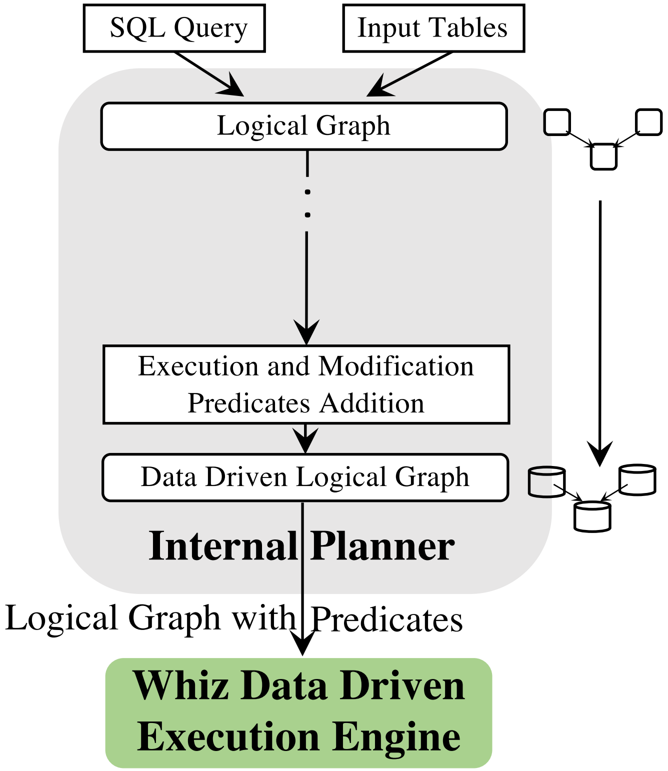
Whiz Job Execution Pipeline


SQL Query
Input Tables
Logical Graph
Physical Graph
Internal Planner
Physical Graph
Compute Centric Execution Engine
SQL Query
Input Tables
Logical Graph
Execution and Modification Predicates Addition
Data Driven Logical Graph
Internal Planner
Logical Graph with Predicates
Whiz Data Driven Execution Engine

Execution predicates
determine when intermediate data is ready to be consumed by the downstream stage

Whiz Job Execution Pipeline

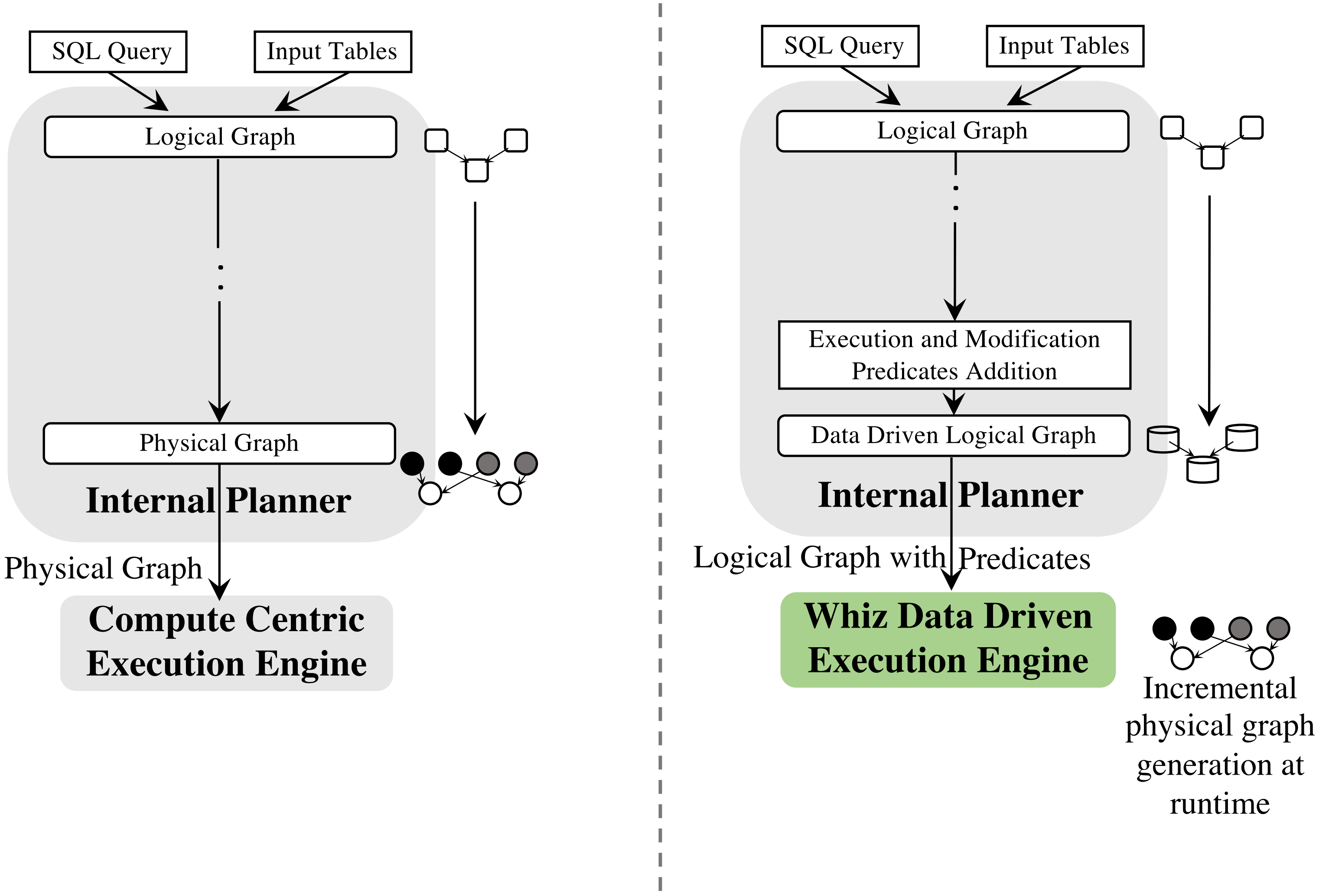



SQL Query
Input Tables
Logical Graph
Execution and Modification Predicates Addition
Data Driven Logical Graph
Internal Planner
Logical Graph with Predicates
Whiz Data Driven Execution Engine

Execution predicates
determine when intermediate data is ready to be consumed by the downstream stage

Modification predicates
determine which processing logic should be chosen at runtime

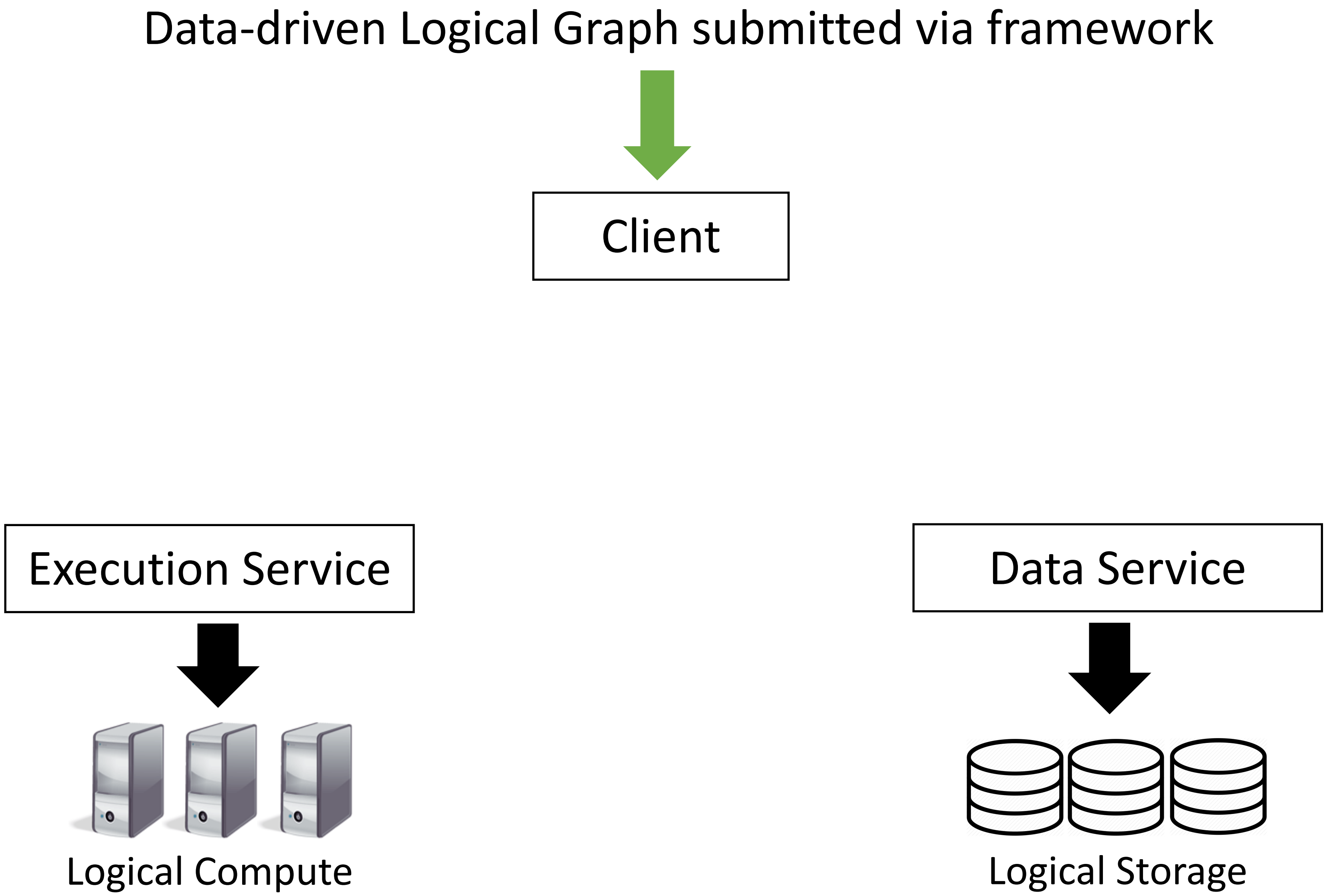
Whiz Job Execution Pipeline


SQL Query
Input Tables
Logical Graph
Physical Graph
Internal Planner
Physical Graph
Compute Centric Execution Engine
SQL Query
Input Tables
Logical Graph
Execution and Modification Predicates Addition
Data Driven Logical Graph
Internal Planner
Logical Graph with Predicates
Whiz Data Driven Execution Engine
Incremental physical graph generation at runtime

Execution predicates determine when intermediate data is ready to be consumed by the downstream stage

Modification predicates determine which processing logic should be chosen at runtime

Whiz Job Execution Pipeline



Data-driven Logical Graph submitted via framework

Client

Execution Service

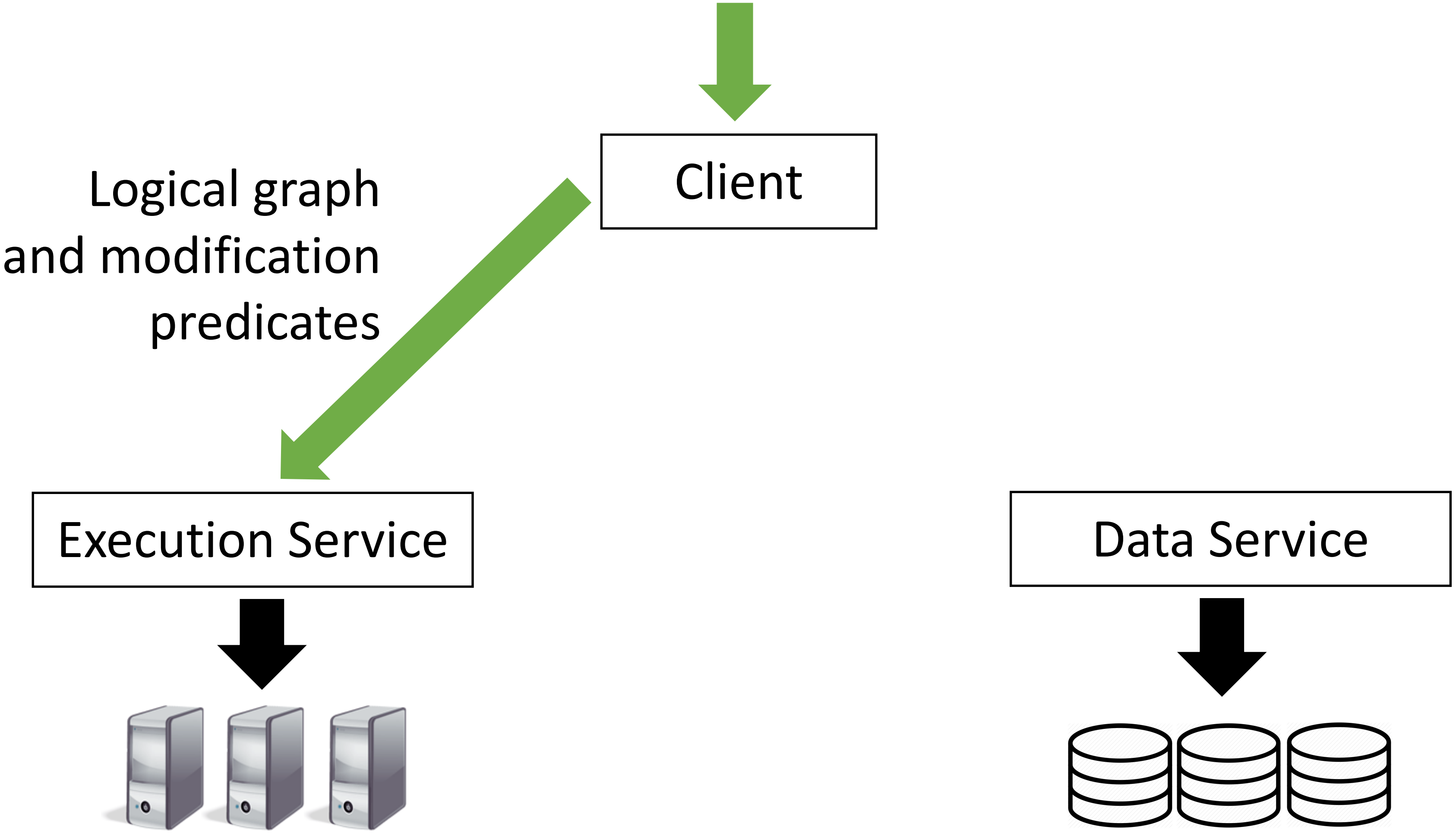
Logical Compute

Data Service

Logical Storage

Whiz Job Execution Pipeline

Data-driven Logical Graph submitted via framework



Client

Logical graph
and modification
predicates

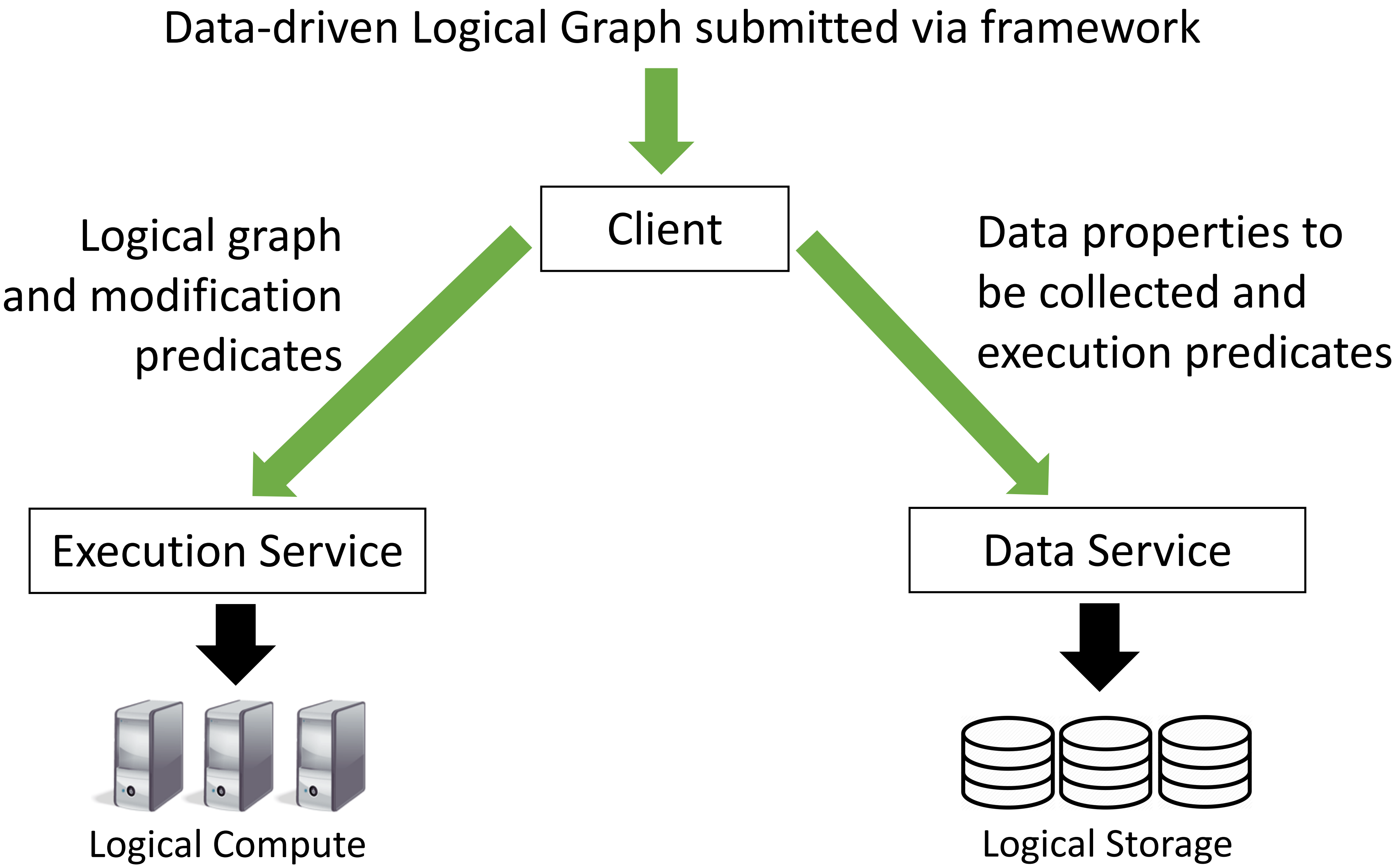
Execution Service

Data Service

Logical Compute

Logical Storage

Whiz Job Execution Pipeline



Data-driven Logical Graph submitted via framework

Client

Logical graph
and modification
predicates

Data properties to
be collected and
execution predicates

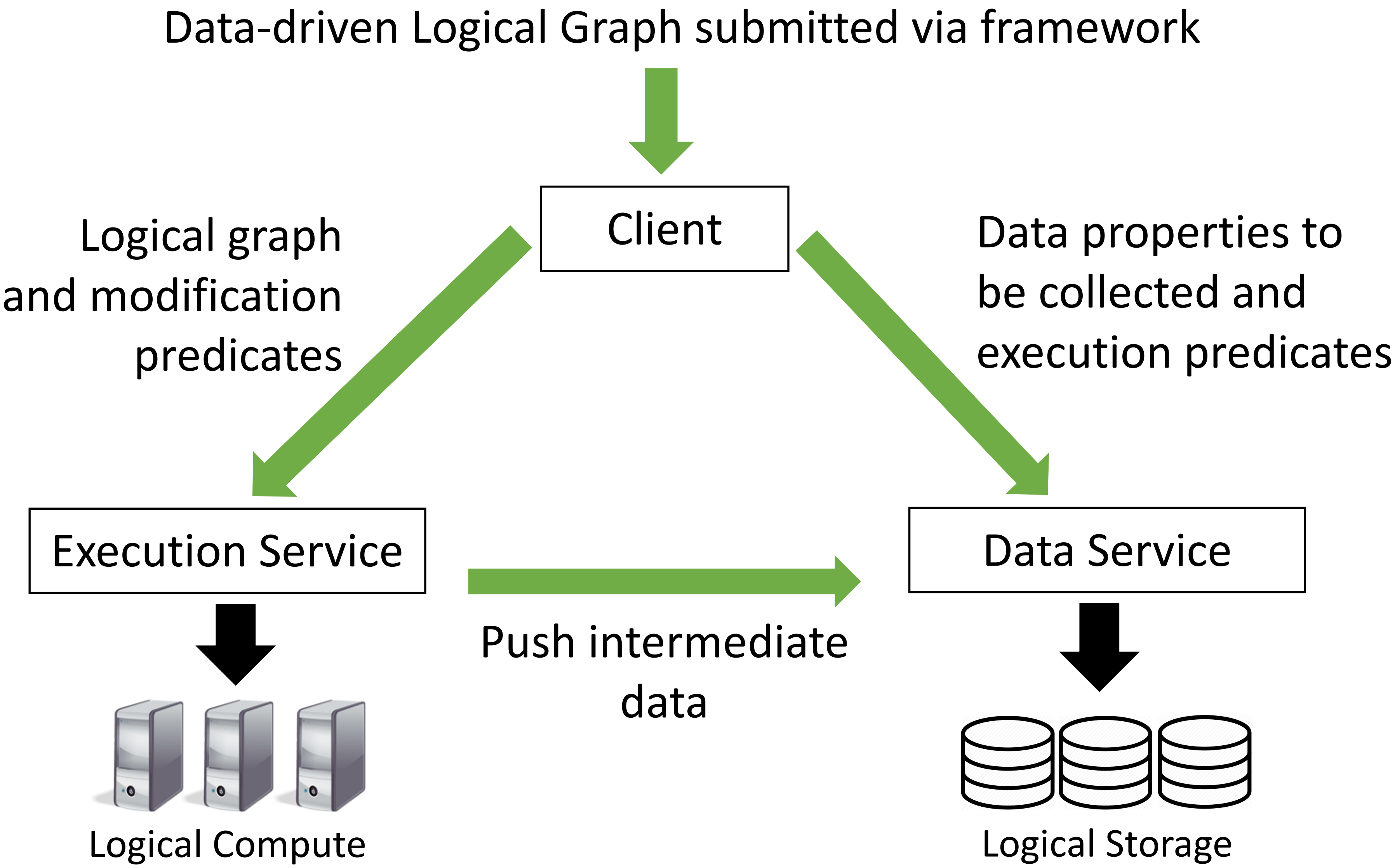
Execution Service

Data Service

Logical Compute

Logical Storage

Whiz Job Execution Pipeline



Data-driven Logical Graph submitted via framework

Client

Logical graph and modification predicates

Data properties to be collected and execution predicates

Execution Service

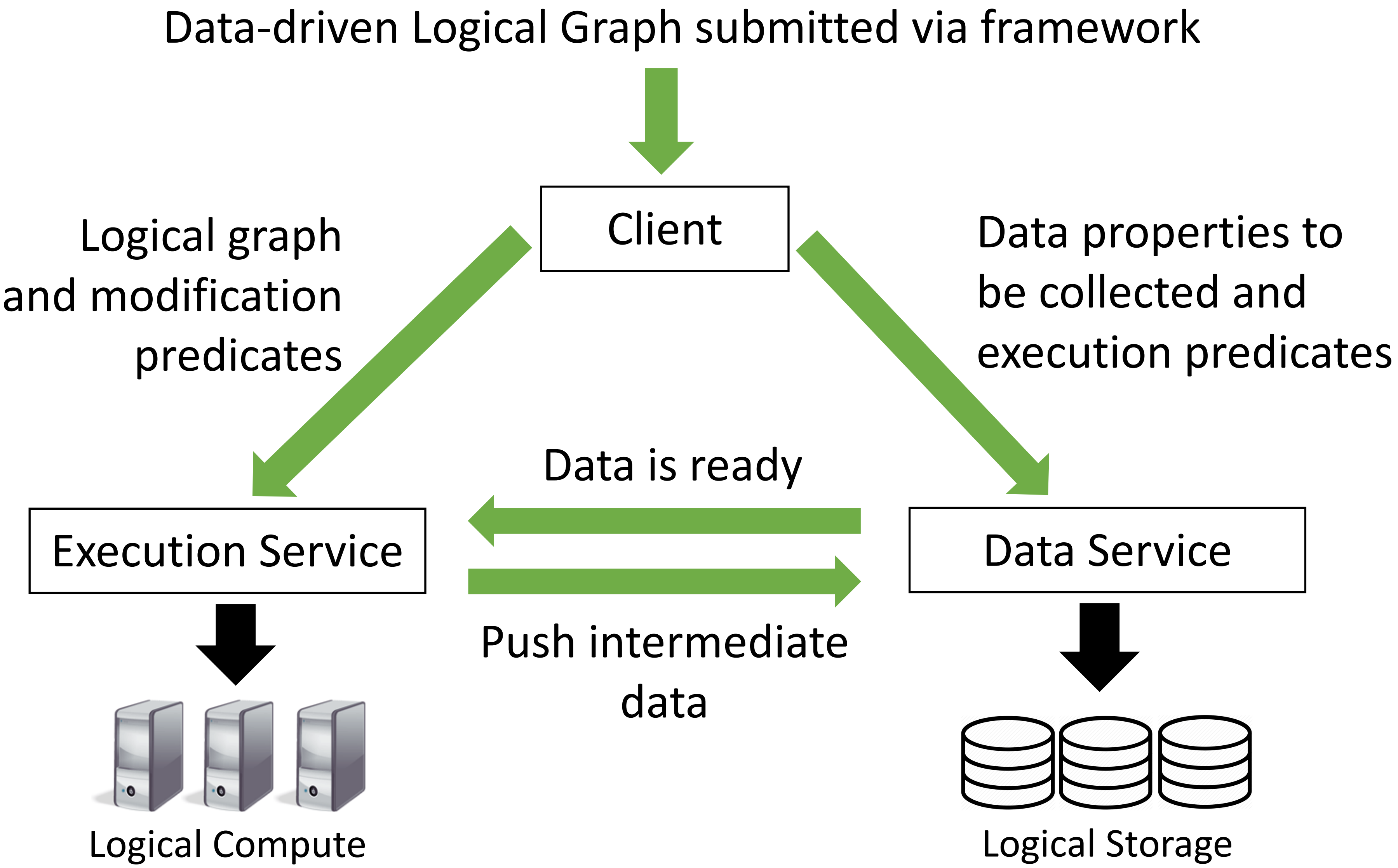
Push intermediate data

Data Service

Logical Compute

Logical Storage

Whiz Job Execution Pipeline



Data-driven Logical Graph submitted via framework

Client

Logical graph
and modification
predicates

Data properties to
be collected and
execution predicates

Execution Service

Data is ready

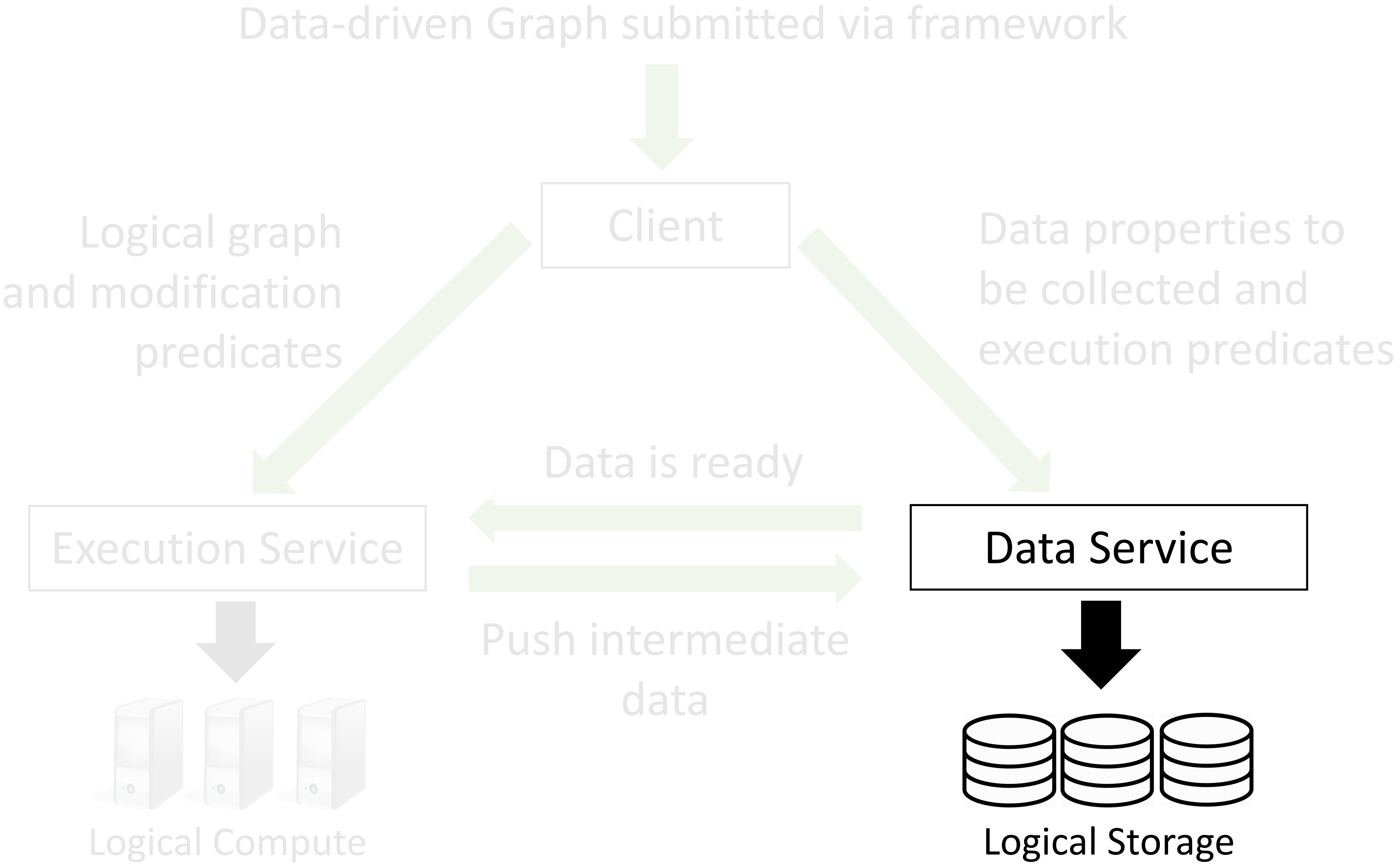
Push intermediate
data

Data Service

Logical Compute

Logical Storage

Whiz Job Execution Pipeline



Data-driven Graph submitted via framework
Client
Logical graph and modification predicates
Data properties to be collected and execution predicates
Data is ready
Execution Service
Data Service
Push intermediate data
Logical Compute
Logical Storage

Whiz Data Service

How to organize the intermediate data (from a job stage)?

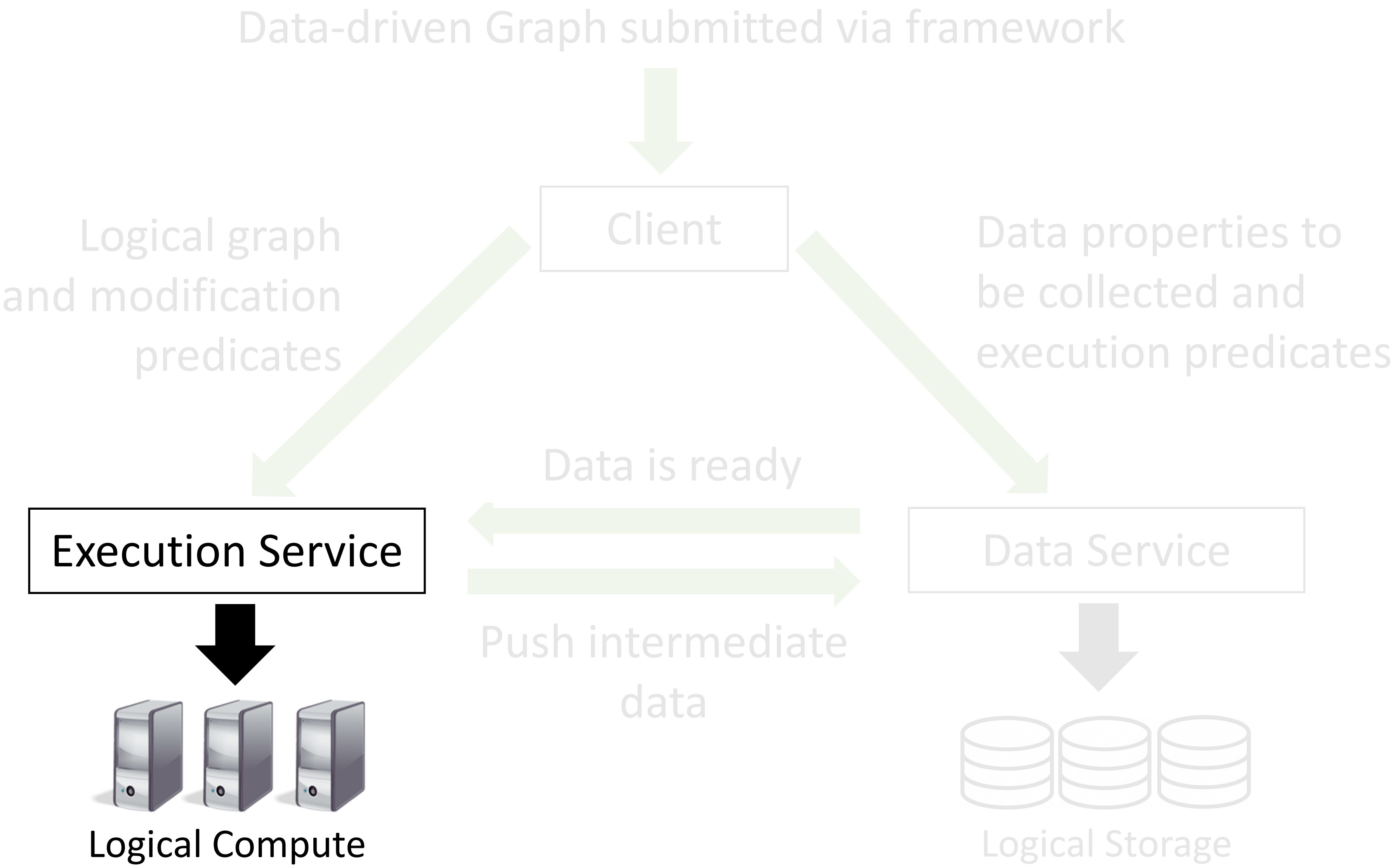
Uses a **linear-time rule based heuristic** to pick machines so as to **maximally ensure load balance, data locality and fault tolerance**

Initialize **fixed number** of intermediate data partitions on each machine
(chosen so as to minimize scheduling and storage overheads)

Intermediate data organization is **no longer tied** to compute structure

- Minimizes within-job skew across tasks
- Avoids hotspots
- Enables rapid task processing
- Minimizes failure recovery time

Whiz Job Execution Pipeline



Data-driven Graph submitted via framework

Client

Logical graph and modification predicates

Data properties to be collected and execution predicates

Data is ready

Execution Service

Data Service

Push intermediate data

Logical Compute

Logical Storage

Whiz Execution Service

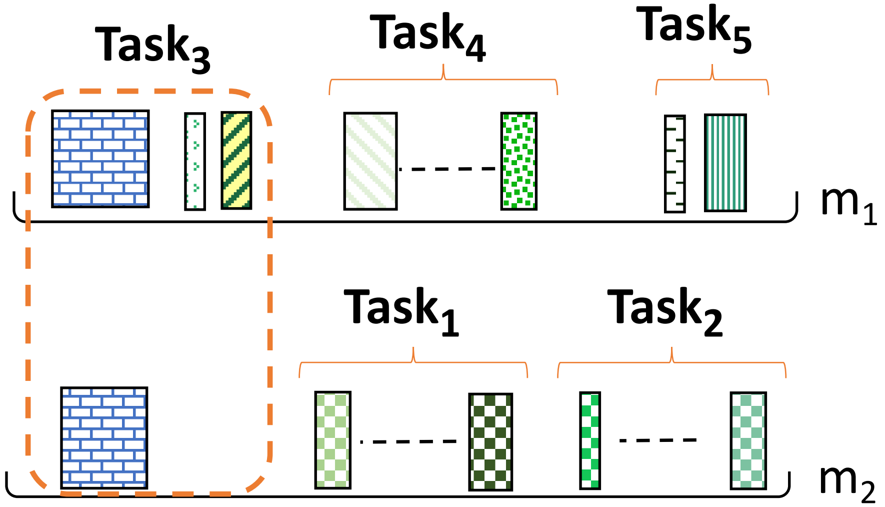
How to decide the task parallelism and placement?

Groups **ready** data partitions subject to an **upper bound**:

- Group *local data partitions*
- Group each *remote partition* (spread across multiple machines)
- Group any remaining data partitions

Each group is processed by a task

Minimizes cross-task skew and **data shuffling**



Whiz Evaluation: Implementation and Setup

Implementation: Modified Tez and YARN

Setup: 50-machine cluster on CloudLab

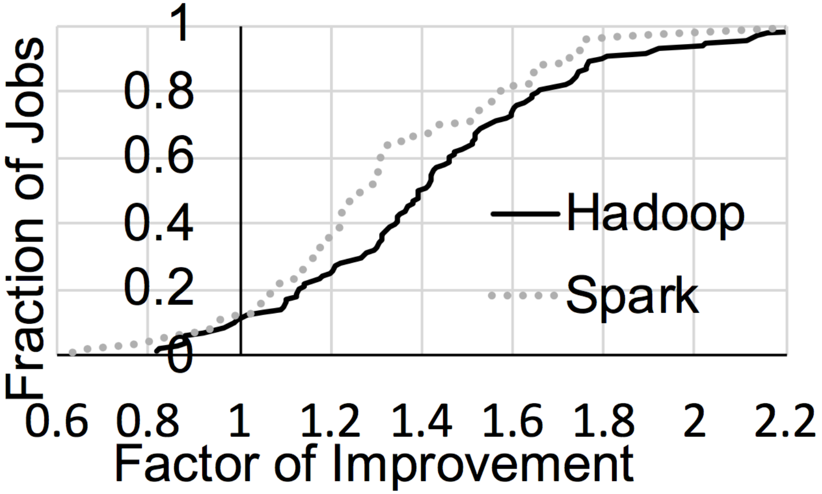
Workloads: TPC-DS queries (for batch) and Page Rank (for graph)

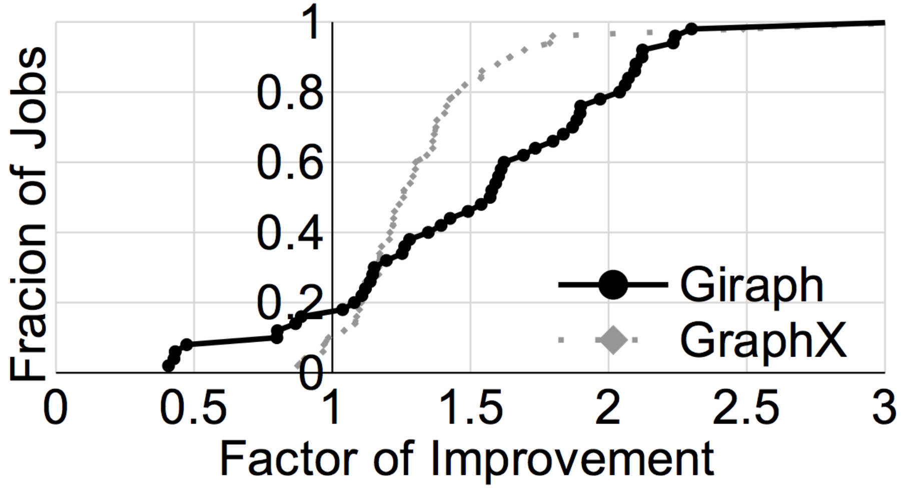
- Poisson arrivals with 20s inter-arrival time

Metrics:

- Job Completion Time and Factor of Improvement = $JCT_{baseline}/JCT_{Whiz}$
- Makespan

Whiz Evaluation: Batch Analytics and Graph Analytics



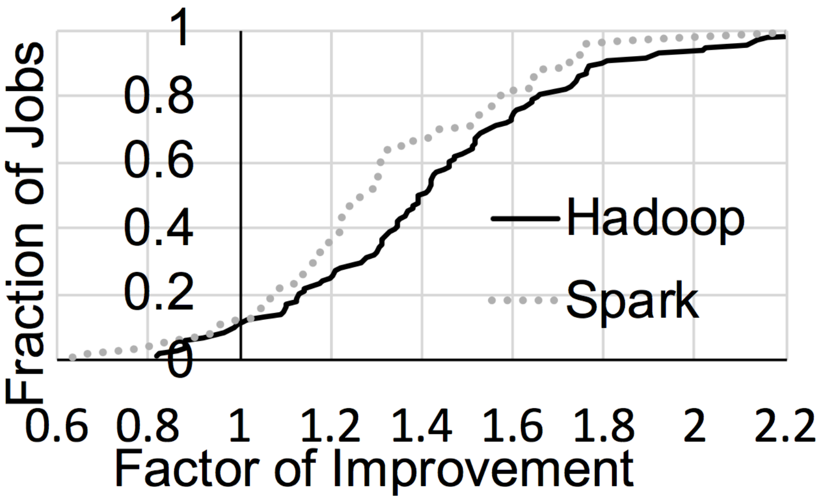


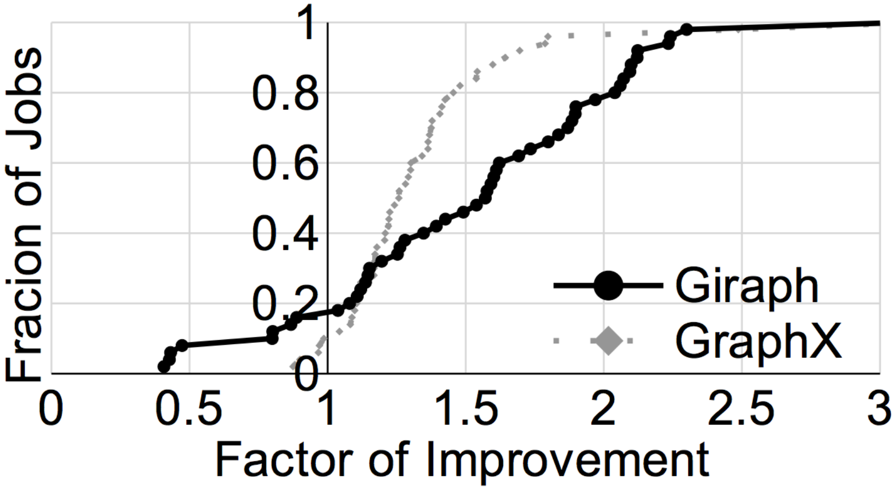
Whiz improves JCT by a factor of **1.4x (1.2x)** on average, and **2.02x (1.75x)** on 95th percentile w.r.t Hadoop (Spark)

Whiz improves JCT by a factor of **1.33x (1.57x)** on average, and **1.57x (2.24x)** on 95th percentile w.r.t GraphX (Giraph)

Whiz improves makespan by a factor of **1.2x – 1.4x**

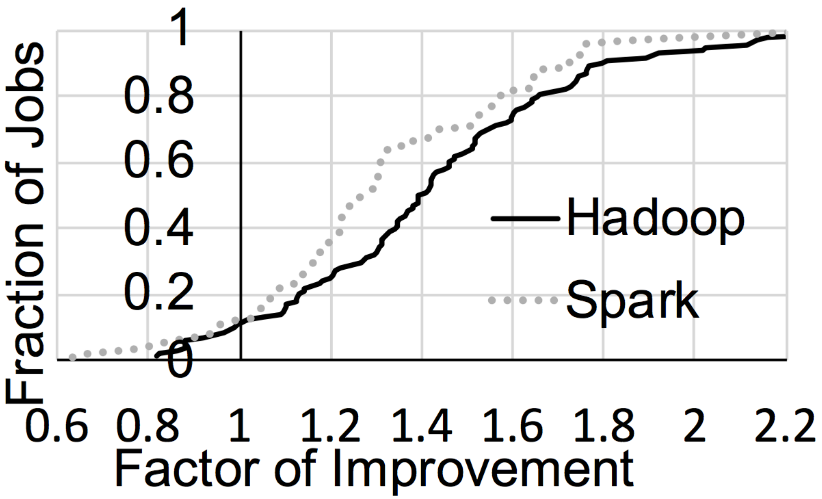
Whiz Evaluation: Sources of Improvement

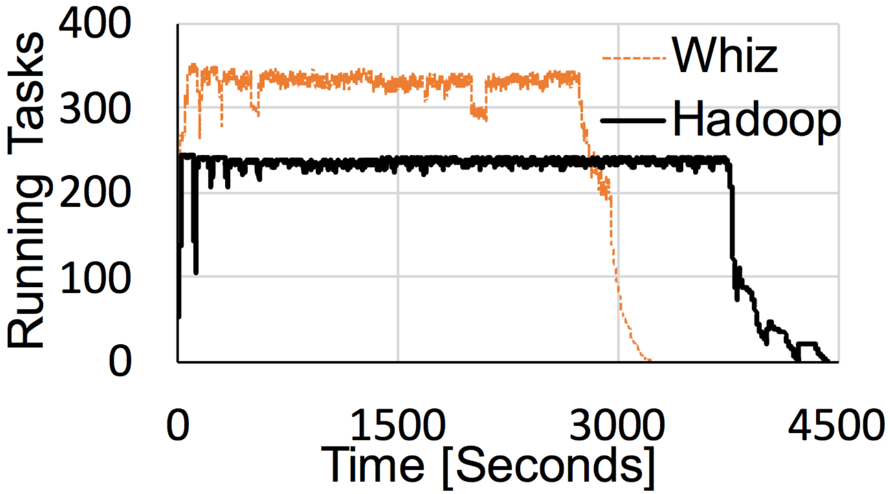




Gains from **more rapid processing** due to **data-driven execution** and **better data management**

Whiz Evaluation: Sources of Improvement

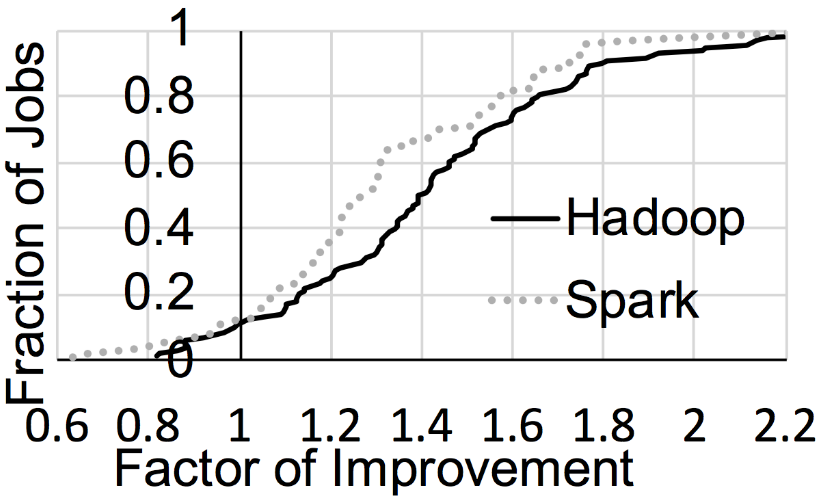


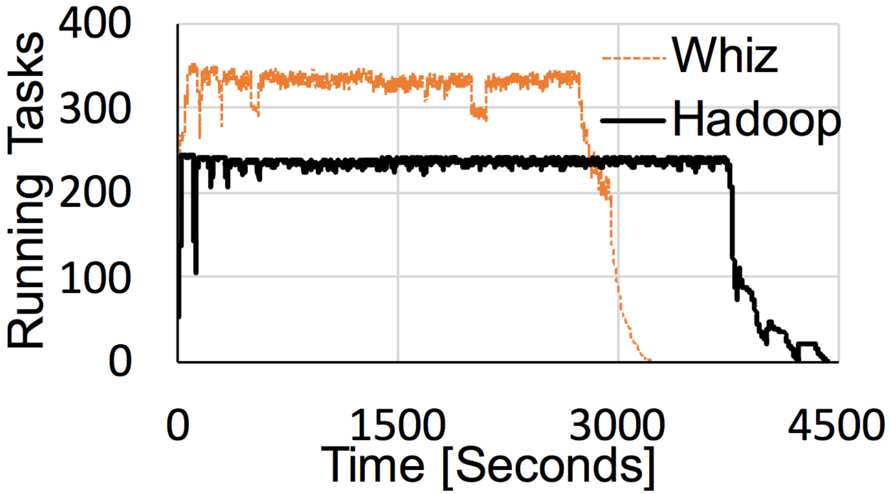


Gains from **more rapid processing** due to **data-driven execution** and **better data management**

Schedules more tasks due to data local tasks

Whiz Evaluation: Sources of Improvement



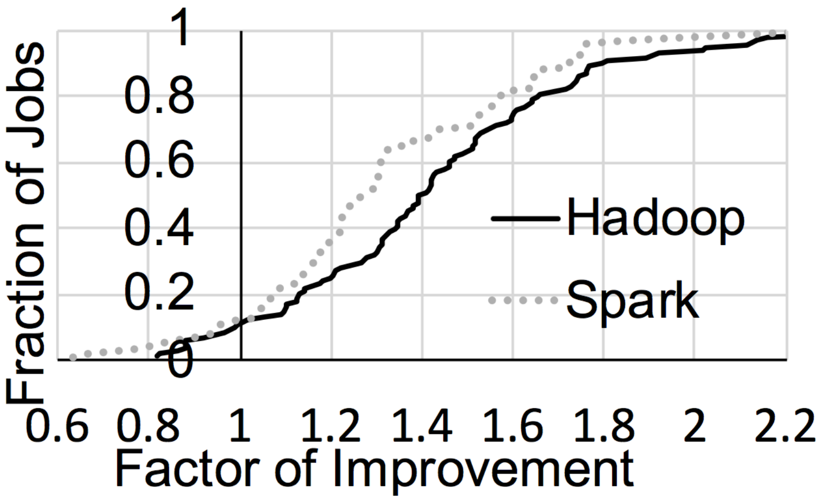


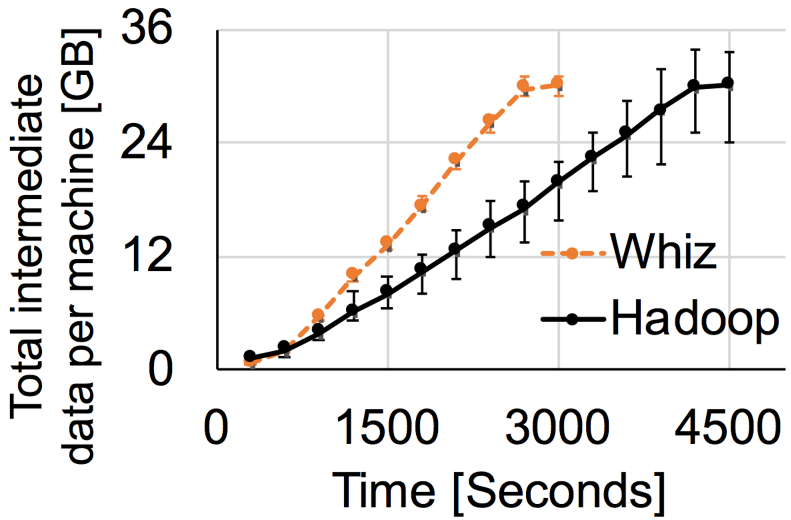
Gains from **more rapid processing** due to **data-driven execution** and **better data management**

Schedules more tasks due to data local tasks

Similar input sizes for tasks in a stage

Whiz Evaluation: Sources of Improvement





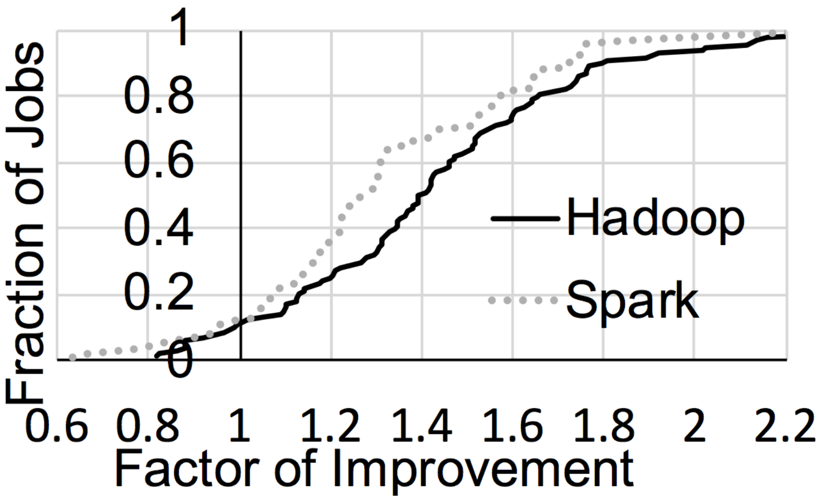
Gains from **more rapid processing** due to **data-driven execution** and **better data management**

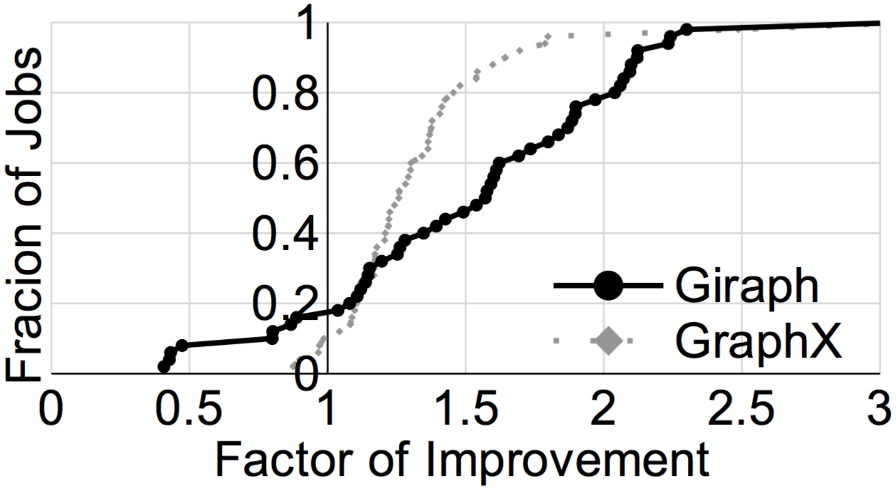
Schedules more tasks due to data local tasks

Similar input sizes for tasks in a stage

Avoids storage hotspots

Whiz Evaluation: Sources of Improvement

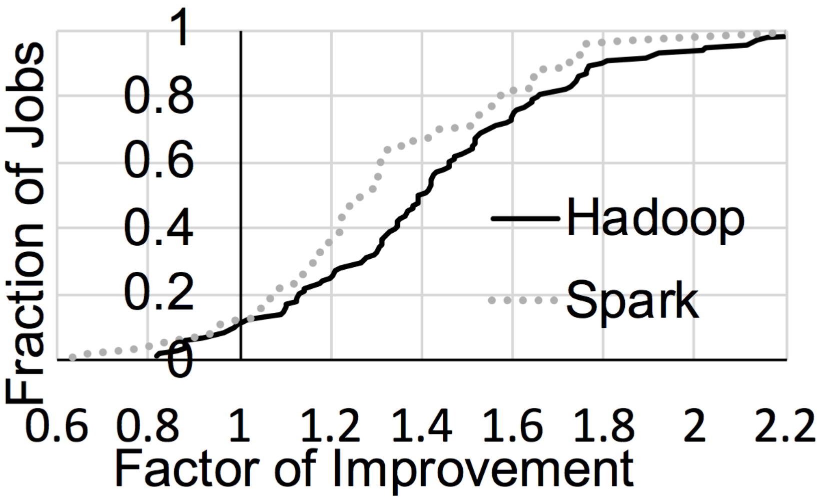


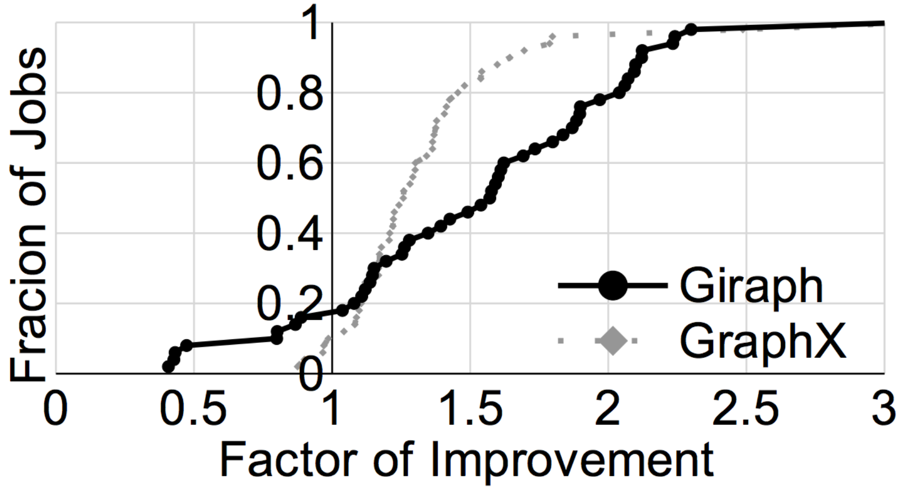


⬇

Optimal Parallelism @ Runtime

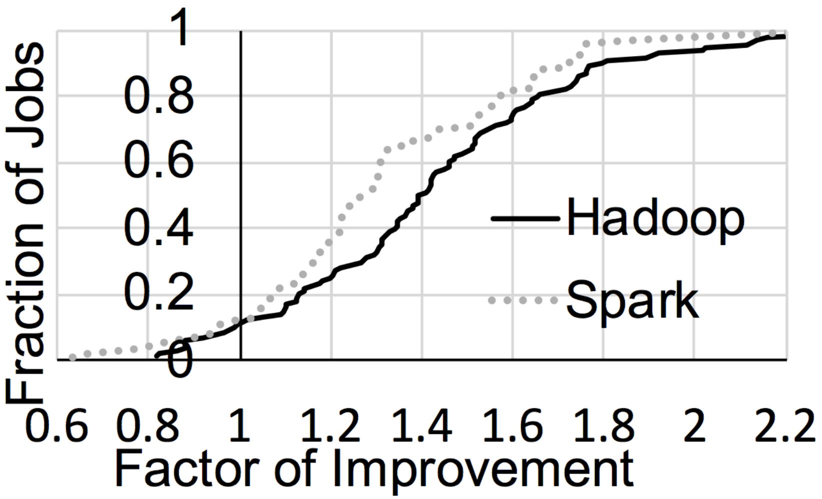
Whiz Evaluation: Sources of Improvement

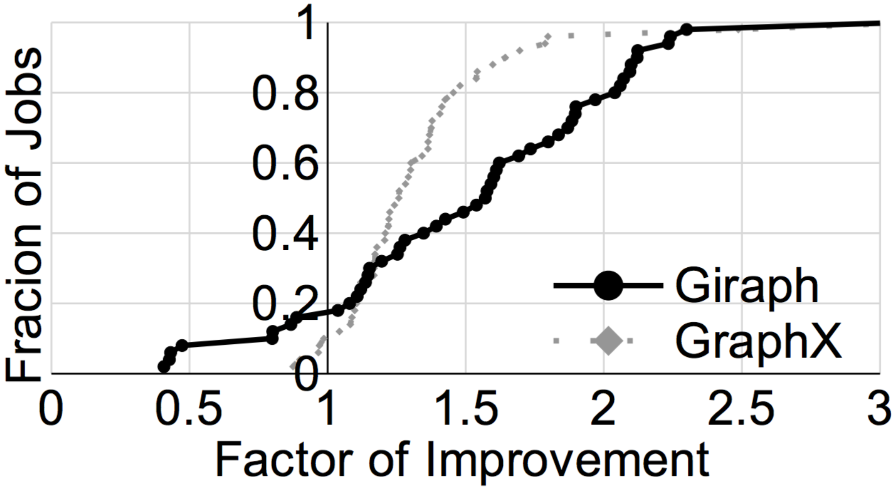




Execution Predicates ⟵ Optimal Parallelism @ Runtime

Whiz Evaluation: Sources of Improvement





Execution Predicates ⬅ Optimal Parallelism @ Runtime

Use of **modification predicates improves performance and efficiency**

Fault-tolerant data organization ensures **minimal performance degradation** during failures

Summary

Compute-centric execution engines hurt flexibility, performance and efficiency

- Tight coupling between compute and intermediate data
- Intermediate data agnosticity

Whiz is a **data-driven** execution engine that drives all aspects of execution based on intermediate data properties

- Makes compute and data equal entities by logically decoupling them
- Brings in intermediate data visibility

Thank You!

asinghvi@cs.wisc.edu

Whiz: Data-driven Analytics Execution

Robert Grandl[*], **Arjun Singhvi**[*], Raajay Viswanathan, Aditya Akella

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

* = co-primary authors