Batcher: Learning to Construct Cost-Efficient Batches of Small Queries in Big Data Processing Platforms

Yeonsu Park[1], Taesung Lee[2], Byungchul Tak[3], and Wook-Shin Han[2]

[1] Kangwon National University

[2] POSTECH

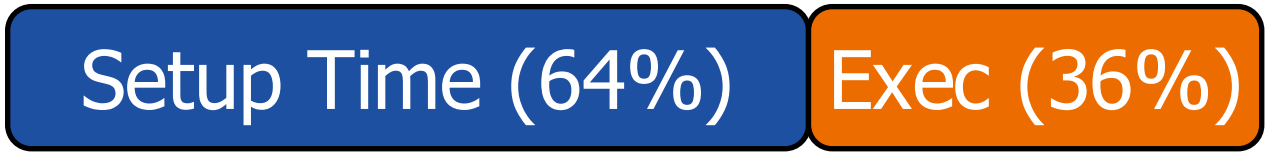
[3] Kyungpook National University

IEEE International Conference on Data Engineering (ICDE), May 2026

Motivation: The Rise of Small Queries

- **Big Data Platforms** (e.g., Spark, Flink) are designed for massive parallel data.
- **The Shift**: Emergence of **"Unintended"** small query workloads (e.g., real-time dashboards).
- **The Bottleneck**: Per-query Setup and I/O overhead dominate execution time.

Execution Profile of a Single Query

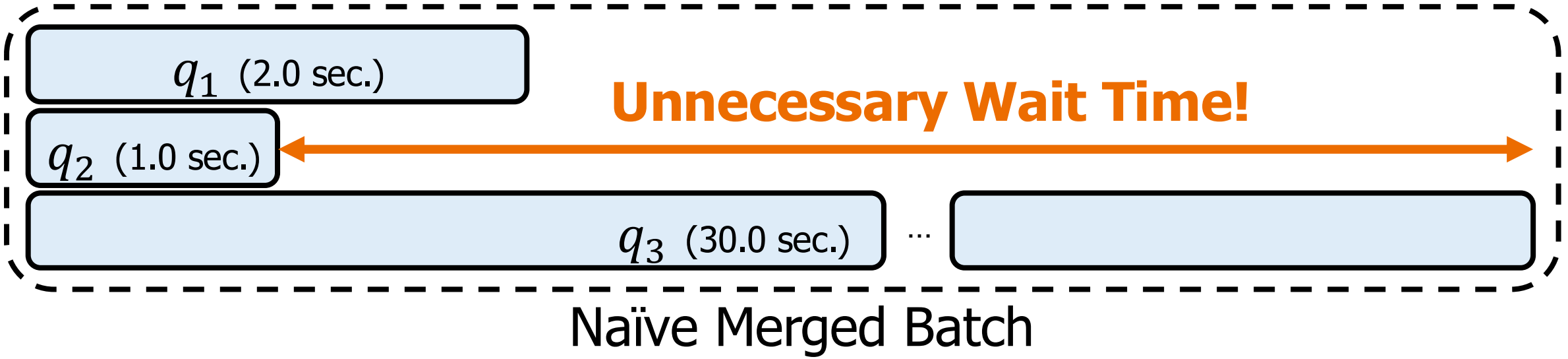


× 10,000 queries

= Massive Overhead

The Problem: Naïve Query Merging

- **State-of-the-Art (Park et al., 2023):** Merges all small queries into one large query to save setup costs.
- **The Flaw:** Indiscriminate First-Come-First-Served (FCFS) merging.
- **Straggler Effect:** Short queries wait for the deepest processing stages of complex queries.

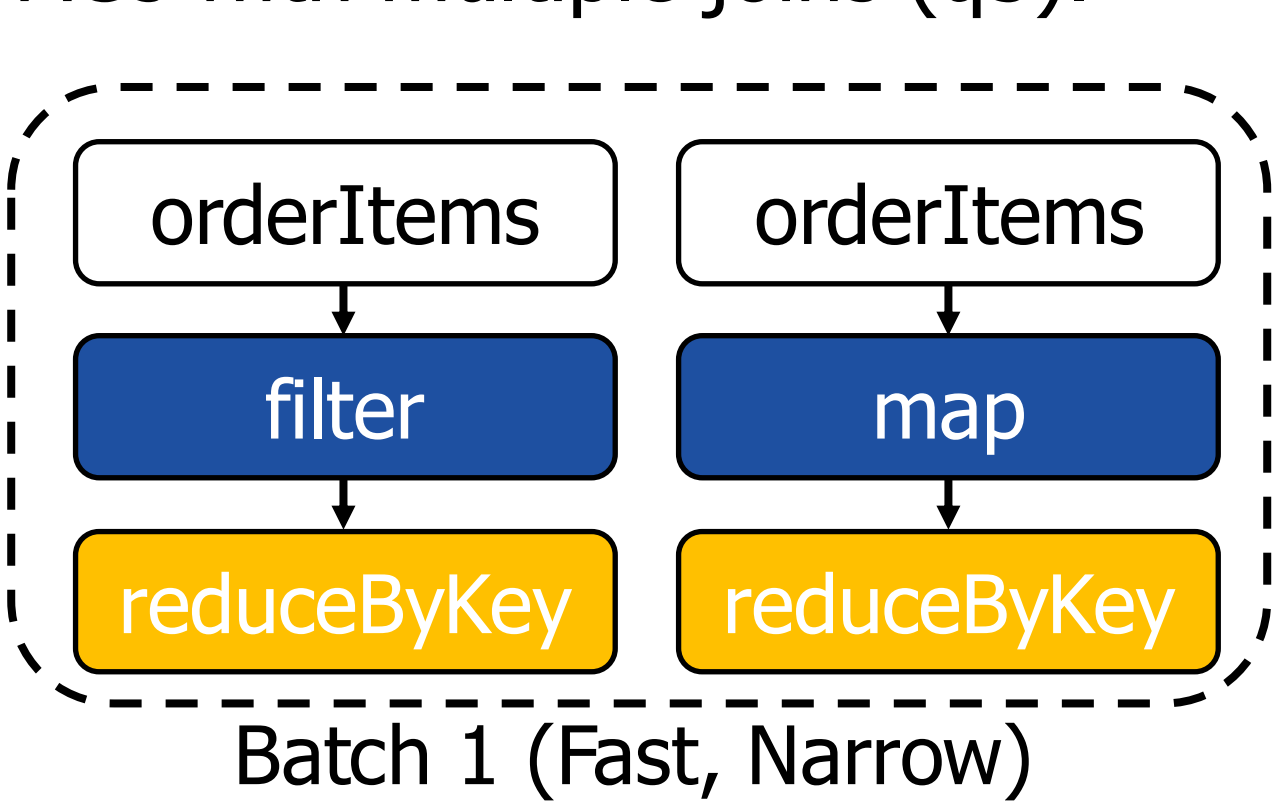


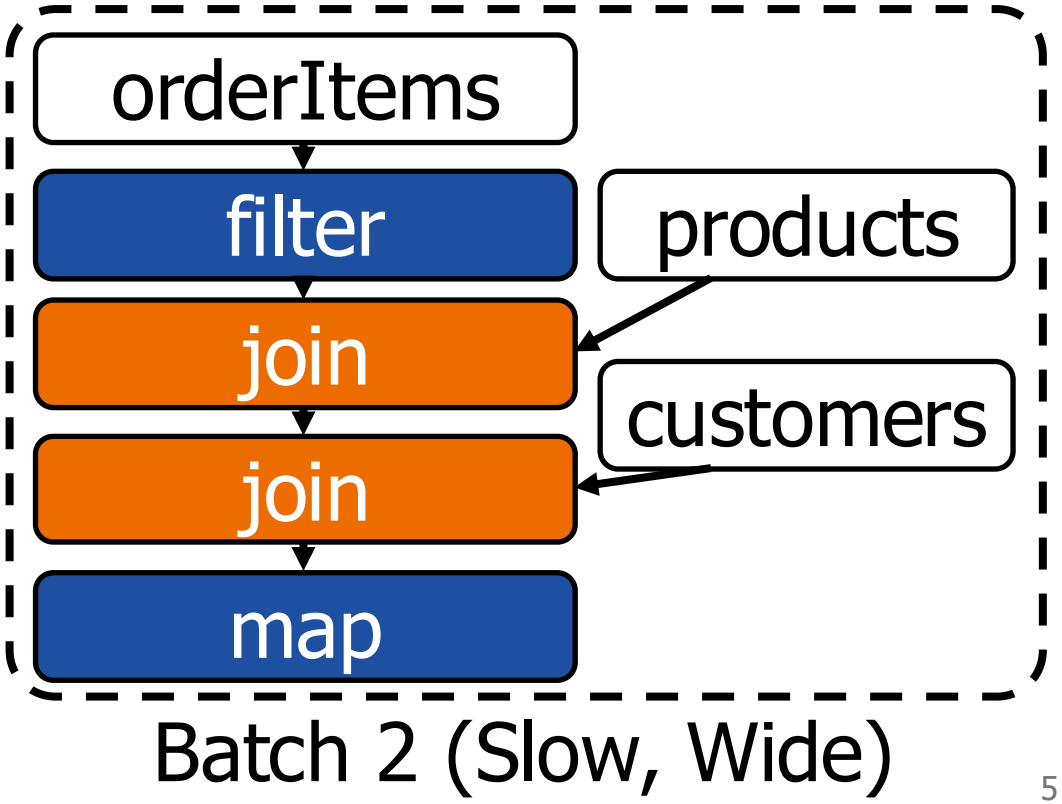
Outline

- **Running Example:** Amazon Seller Central
- **System Architecture:** Where Batcher Sits
- **Core Methodology:**
  - Intra-Batch Strategy (Two-Step Clustering)
  - Iterative Refinement
  - Batch Cost Estimation
  - Inter-Batch Strategy & Optimization
- **Evaluations**
- **Conclusion**

Example: Amazon Seller Central (DAGs)

- **Scenario:** 10,000 concurrent queries hit Spark. We must group them by DAG structure.
- **Goal:** Isolate **fast, narrow** DAGs (q1, q2) from **slow, wide** DAGs with multiple joins (q3).



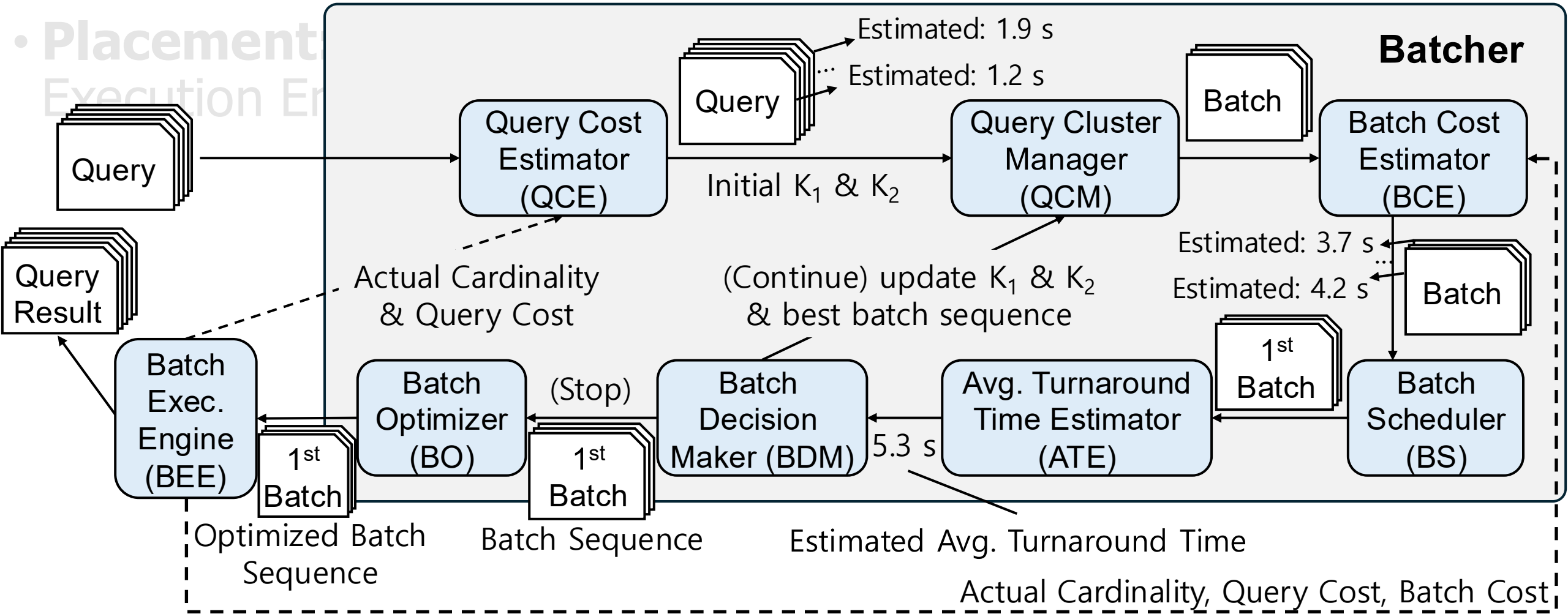


System Context: Batcher Architecture

- **Goal:** Find optimal batch sequence $\vec{B}$ minimizing average turnaround time.
- **Placement:** Intelligent middleware operating before the Batch Execution Engine (e.g., Spark, Flink).

System Context: Batcher Architecture

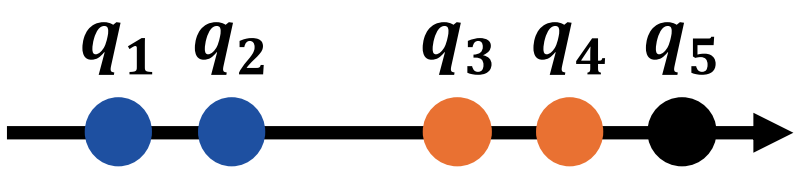
- **Goal:** Find optimal batch sequence $\vec{B}$ minimizing average turnaround time.
- **Placement**
  Execution En



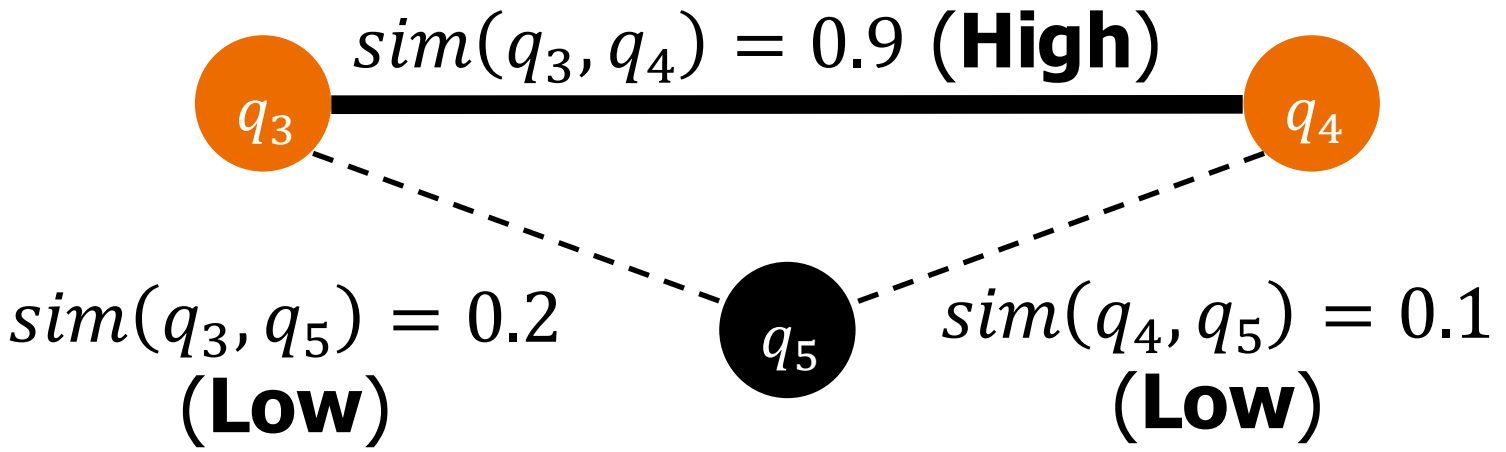
Intra-Batch Strategy: Two-Step Clustering

- **Step 1 (Time):** Custer by predicted running time (1D K-means) to isolate stragglers.
- **Step 2 (Structure):** Sub-divide using pairwise structural similarity $sim(d_i, d_j)$.

Step 1: Cluster by Time



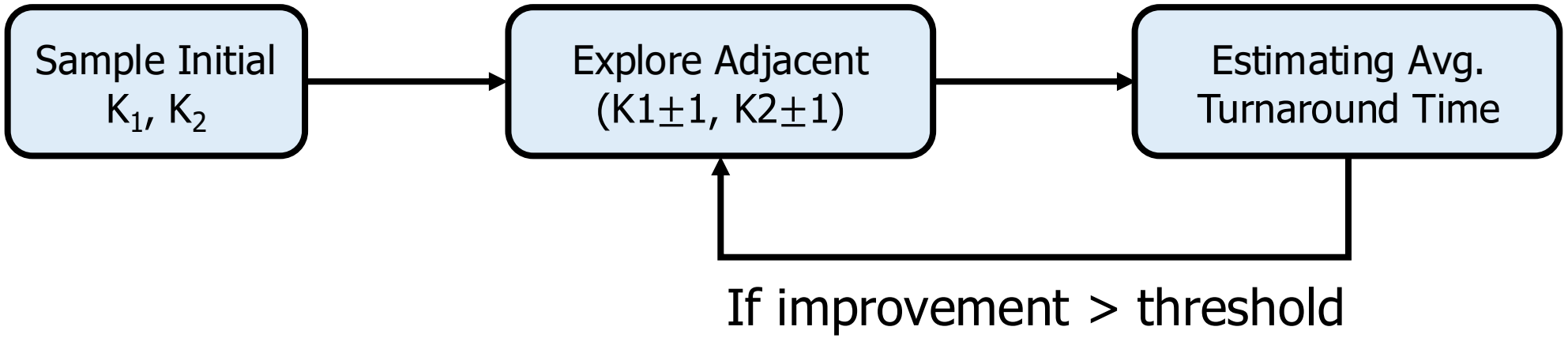
Step 2: Sub-divide by Pairwise Structure



Queries are represented as multidimensional feature vectors of pairwise similarities.

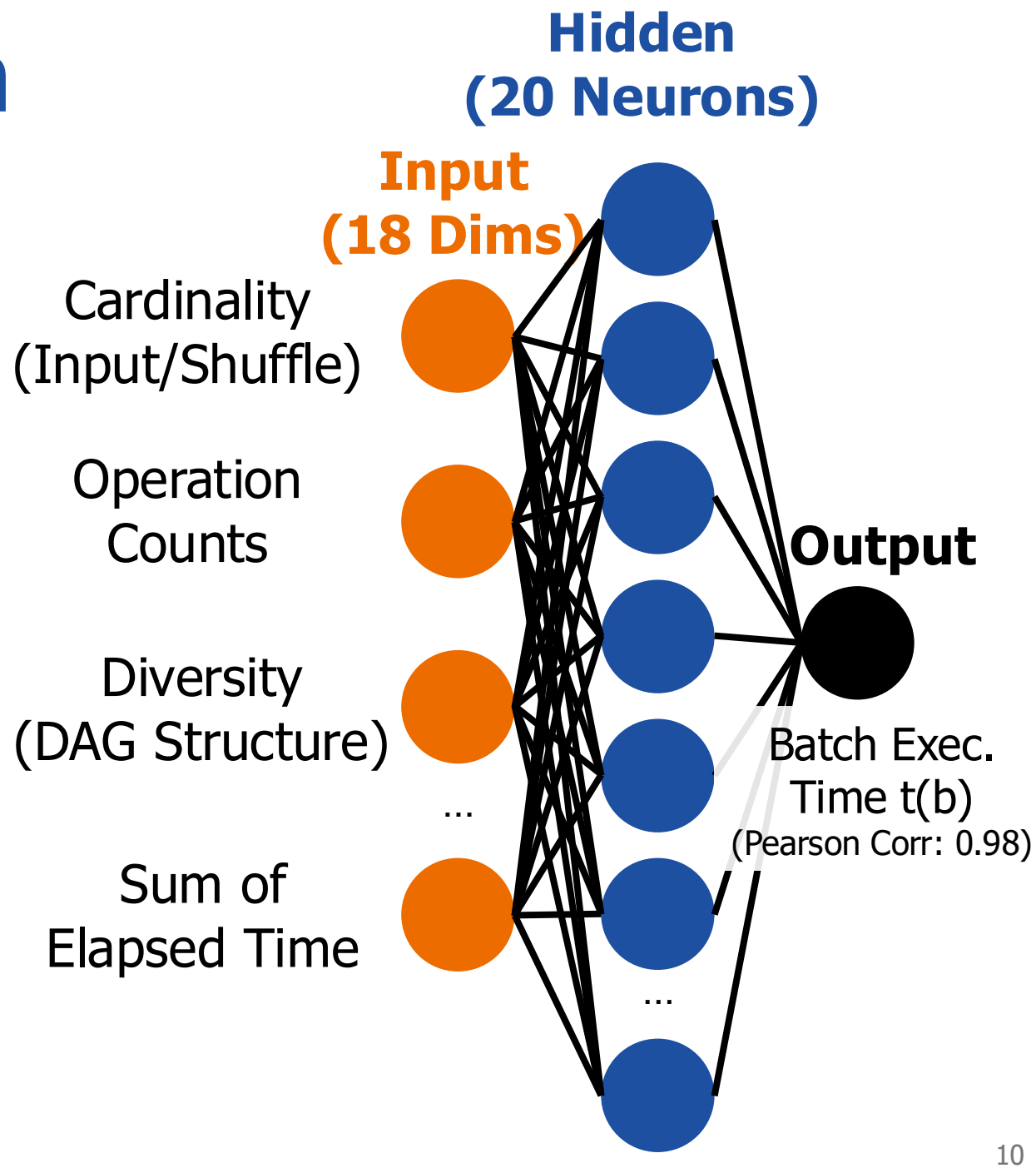
Iterative Refinement of Batches

- **Challenge:** Finding the optimal number of clusters (K_1, K_2) is NP-hard.
- **Solution:** ε-greedy strategy to incrementally explore adjacent configurations.
- **Benefit:** Lightweight, on-the-fly decision making with low overhead.



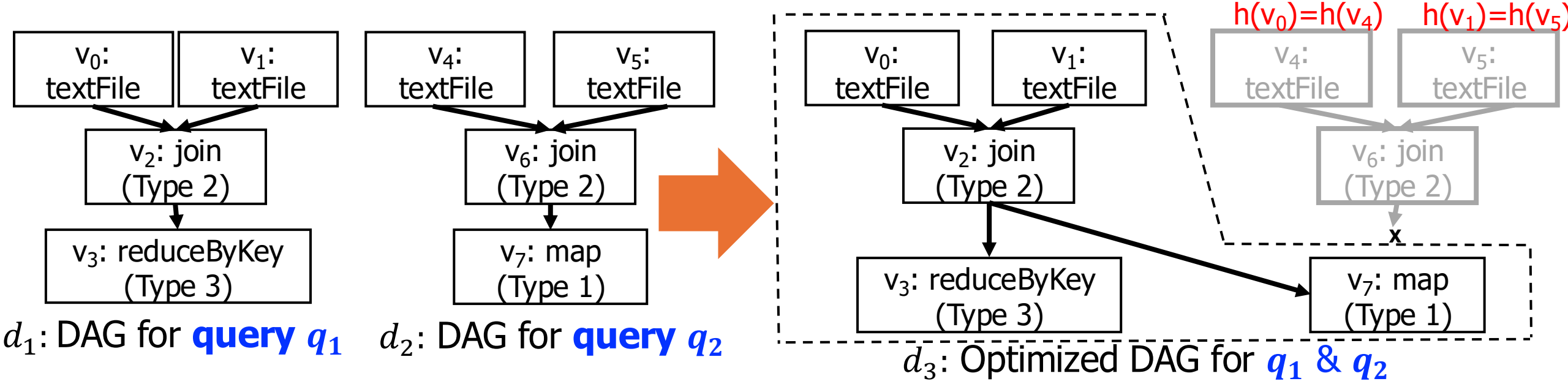
Batch Cost Estimation

- **The Need:** Accurately predict batch processing time t(b) to evaluate cluster configurations.
- **The Model:** Multi-Layer Perceptron (MLP) with 1 hidden layer (20 neurons), ReLU, and Adam.
- **Why MLP?** Captures non-linear feature interactions (e.g., Cardinality × Row Length).



Intra-Batch Strategy & Optimization

- **Scheduling:** Weighted Shortest Job First (WSJF) policy.
- **Batch Optimizer (BO):** Common subquery result reuse via hash-based signatures.



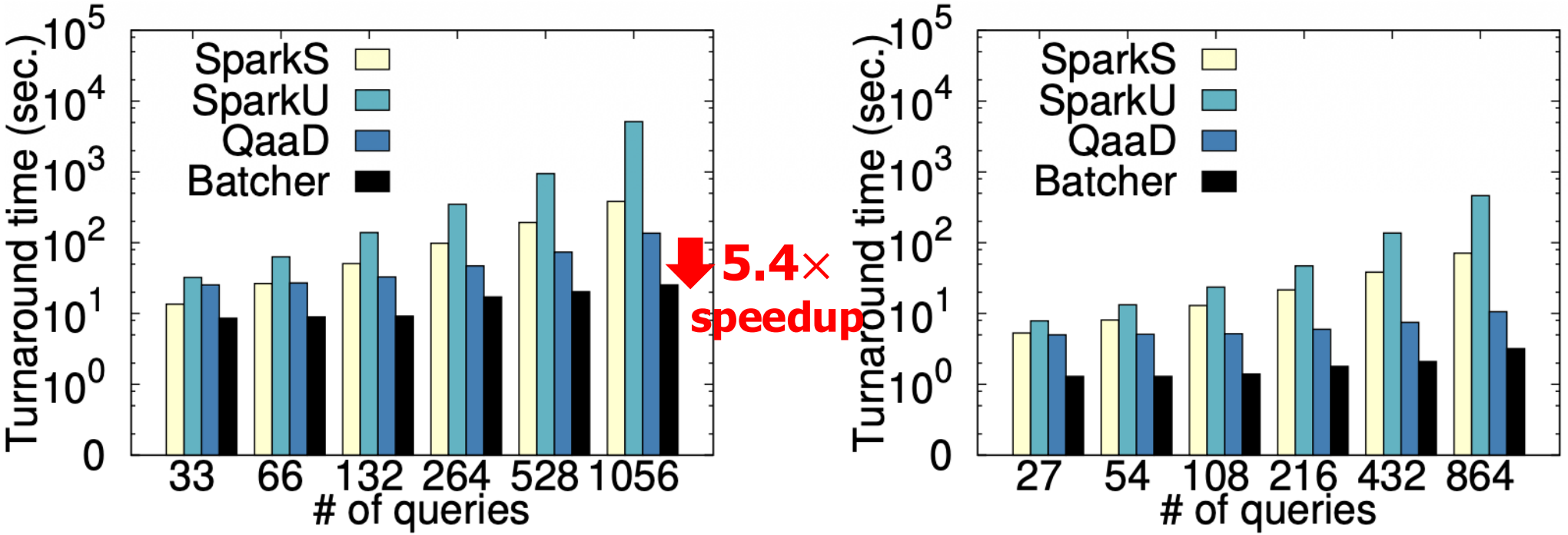
d_1: DAG for **query** q_1 d_2: DAG for **query** q_2 d_3: Optimized DAG for q_1 & q_2

Evaluation Setup

- **Platform**: Apache Spark 3.0 (5-node cluster).
- **Datasets**:
  - **Brazilian E-commerce Dataset**: 1.5M e-commerce order.
  - **eBay**: Auction details.
- **Baselines**:
  - **SparkS**: Vanilla Spark (independent queries).
  - **SparkU**: UNION operator merging.
  - **QaaD**: State-of-the-art single-batch query merging.

Evaluation: Average Turnaround Time

- **X-Axis**: Number of Queries / **Y-Axis**: Turnaround Time (in sec.)
- **Takeaway**: **Batcher** achieves up to 5.4× speedup over **QaaD** by avoiding stragglers.



(a) BRA dataset.

(b) eBay dataset.

Evaluation: Ablation Study

- **Why does Batcher work?**
  - Disabling components reveals their impact.
- **Takeaway:** Every component (**Clustering**, **Refinement**, **Scheduling**, **Reuse**) is critical.

Two-step Clustering	Iterative Refinement	WSJF Scheduling	Result Reuse	# of queries					
				33	66	132	264	528	1056
QaaD				25.4 (3.0×)	27.1 (3.0×)	32.9 (3.6×)	47.0 (2.7×)	73.8 (3.6×)	136.8 (5.4×)
QaaD-MB (Minimal Batching)				11.0 (1.3×)	20.5 (2.3×)	33.6 (3.7×)	56.8 (3.3×)	101.7 (5.0×)	160.7 (6.3×)
✗	✓	✓	✗	20.5 (2.4×)	25.4 (2.8×)	26.1 (2.8×)	33.0 (1.9×)	34.9 (1.7×)	54.6 (2.1×)
✓	✗	✓	✗	11.9 (1.4×)	19.4 (2.2×)	19.6 (2.1×)	41.9 (2.4×)	52.1 (2.6×)	54.5 (2.1×)
✓	✓	✗	✗	16.0 (1.9×)	23.8 (2.6×)	28.9 (3.1×)	32.1 (1.9×)	59.2 (2.9×)	81.6 (3.2×)
✓	✓	✓	✗	11.0 (1.3×)	13.8 (1.5×)	16.0 (1.7×)	27.4 (1.6×)	33.2 (1.6×)	43.3 (1.7×)
Batcher				8.6	9.0	9.2	17.2	20.4	25.4

Conclusion

- **The Problem:** Small query-dominant workloads cripple big data platforms; naïve merging suffers from the **straggler effect**.
- **Our Solution (Batcher):**
  - **Learns** optimal batching strategies using **batch cost estimation**.
  - **Groups** queries by **running time** and **structural similarity**.
  - **Schedules** via **WSJF** and **reuses** common subqueries.
- **Impact:** Up to **5.4× performance improvement** with minimal overhead. Highly scalable and adaptable to big data platforms like Spark and Flink.

Thank You!

Questions?

Key message:
Batcher intelligently groups massive small queries using ML-based batch cost predictions, delivering up to 5.4x faster performance.