SEATTLE
WA, USA
SIGMOD
PODS
2023

QaaD (Query-as-a-Data): Scalable Execution of Massive Number of Small Queries in Spark

Yeonsu Park[1], Byungchul Tak[2], and Wook-Shin Han[1]
[1]POSTECH
[2]Kyungpook National University

What is Apache Spark?

Fast and general cluster computing engine to process large-scale data

Key Uses

- SQL analytics
- Machine learning
- Streaming data

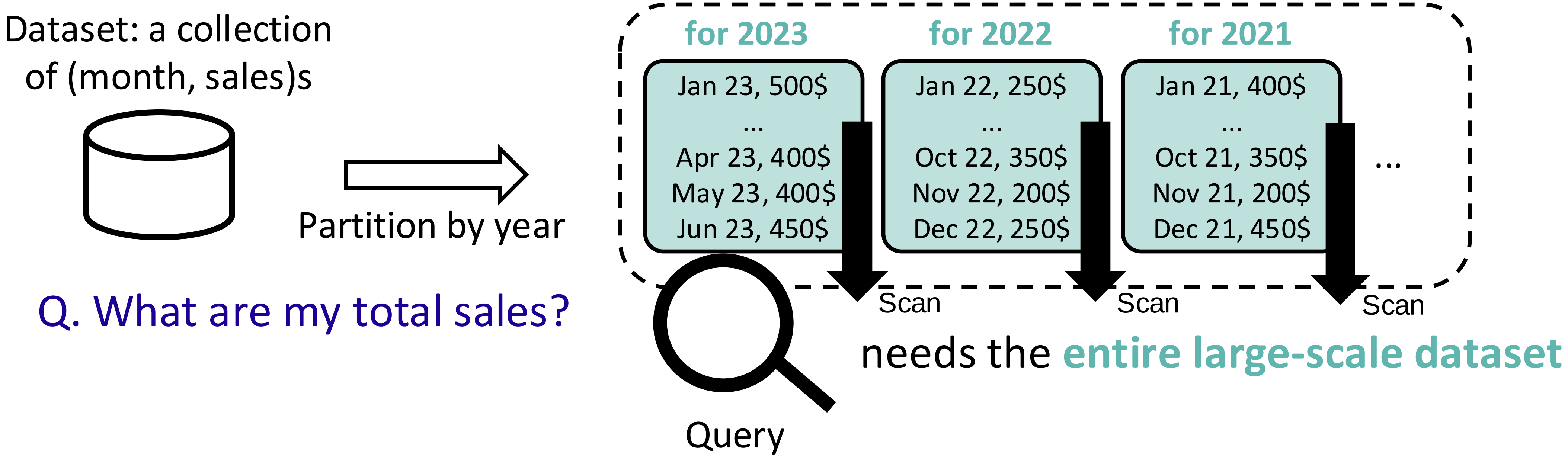
Design & Performance

- Designed for high-performance, heavy data workloads
- Enables high-degree of parallelism

Spark is the most widely-used big data processing platform.

Intended Workload of Spark

Spark is designed and optimized for **a query needing homogeneous operations on large datasets**.



Unintended Workload of Spark

Queries for small input data continue to grow in the workload of big data platforms.

What are my total sales for the **last three months**?

Query needs **the data only for this year**

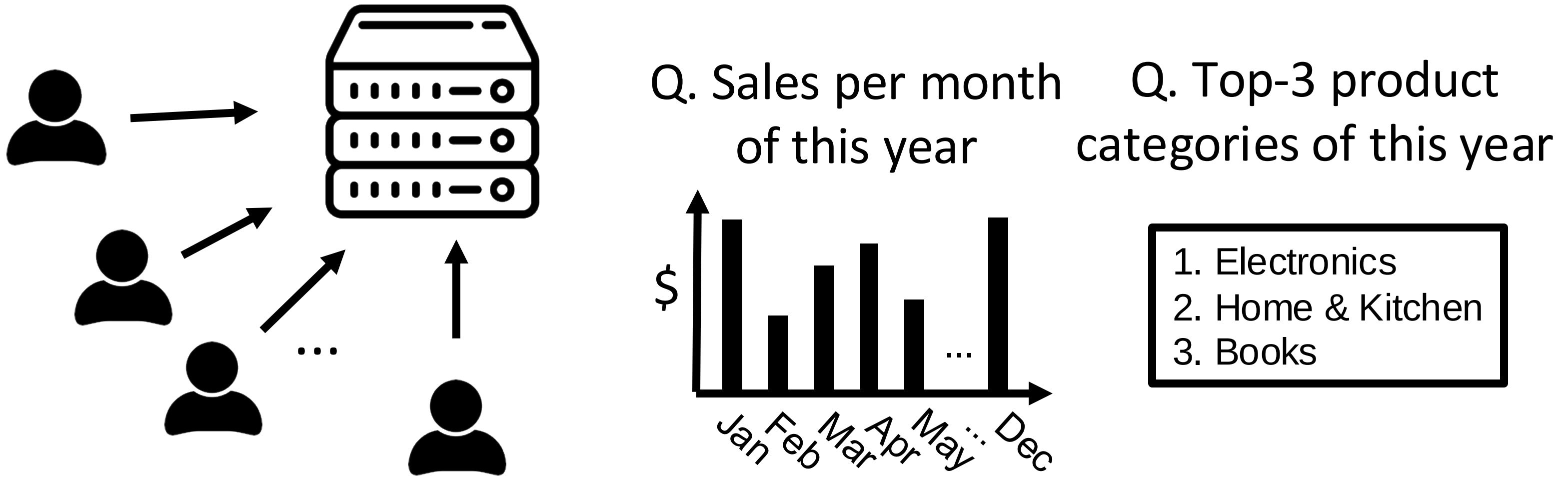
Characteristics: Light computation & A massive number

Observed in Youtube [1] , Alibaba Cloud [2], ...

[1] Biswapesh Chattopadhyay, et al. “Procella: Unifying Serving and Analytical Data at YouTube.” (VLDB’19)
[2] Rui Han, et al. “Adaptiveconfig: Run-time configuration of cluster schedulers for cloud short-running jobs.” (ICDCS’18)

Primary Sources of Queries for Small Data

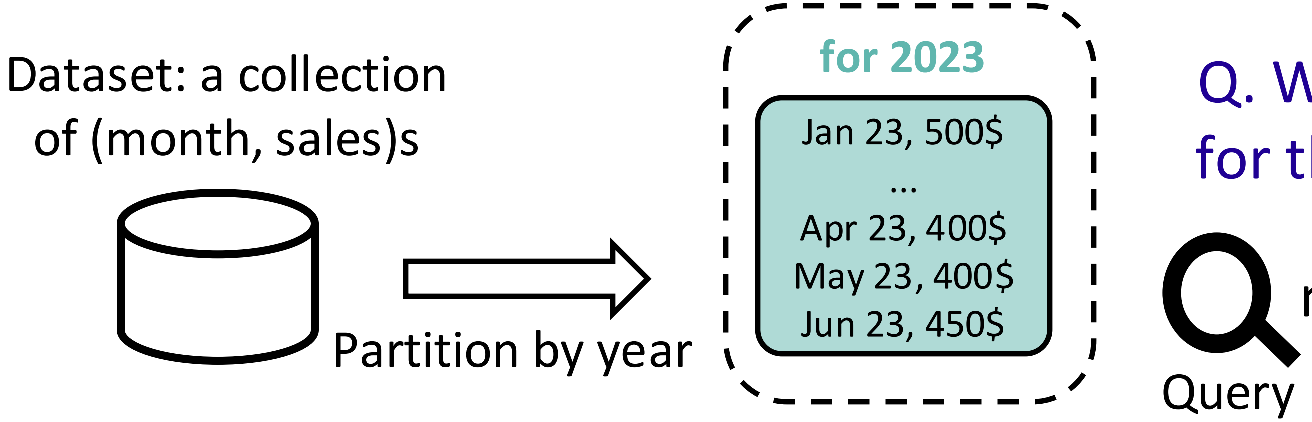
- **Dashboarding queries** for statistics of **recent data** by Amazon sellers



- **High-level libraries such as Pig and Hive**
  - High-level user queries ⟶ a large number of small Spark queries

Our Definition of Small Query

- We define a **small query** as the query whose input data can **fit into a single partition** specified in the Spark configuration.



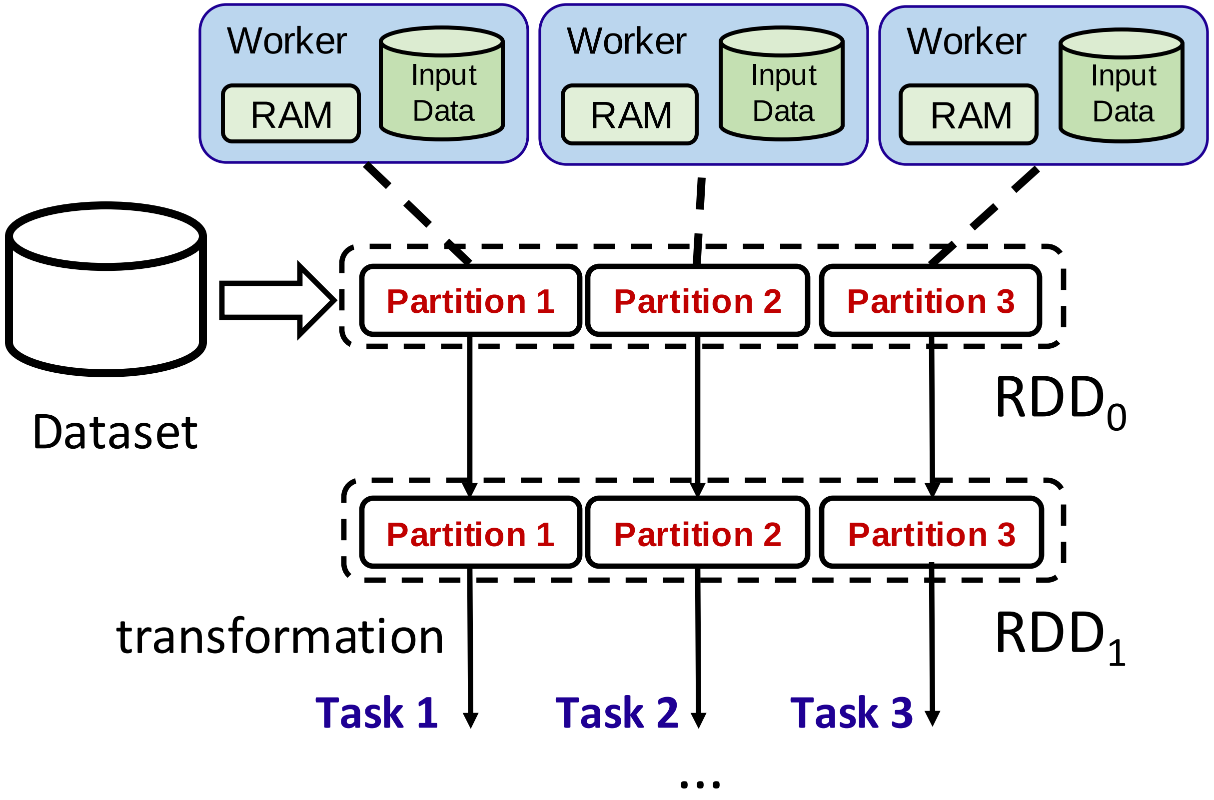
Q. What are my total sales for the **last three months**?

needs **the data stored in a partition**

workloads consisting of **a massive number of small queries**

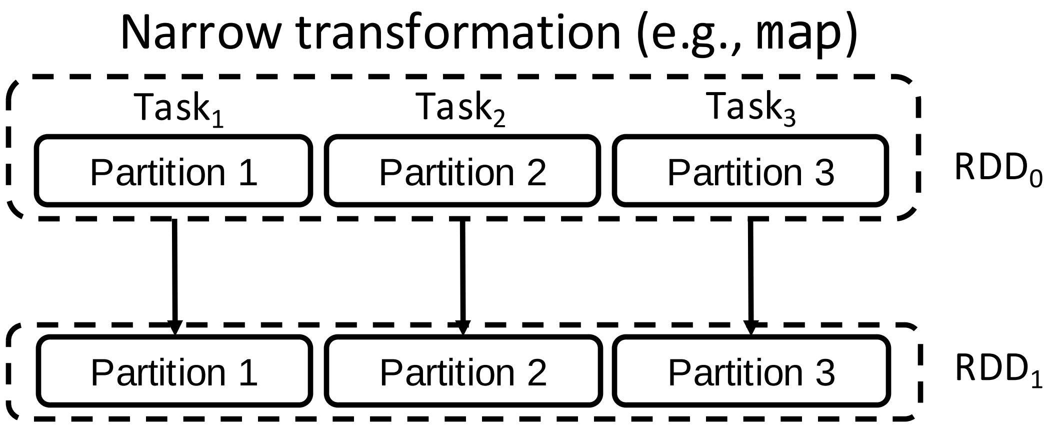
Key Concept in Spark: RDD

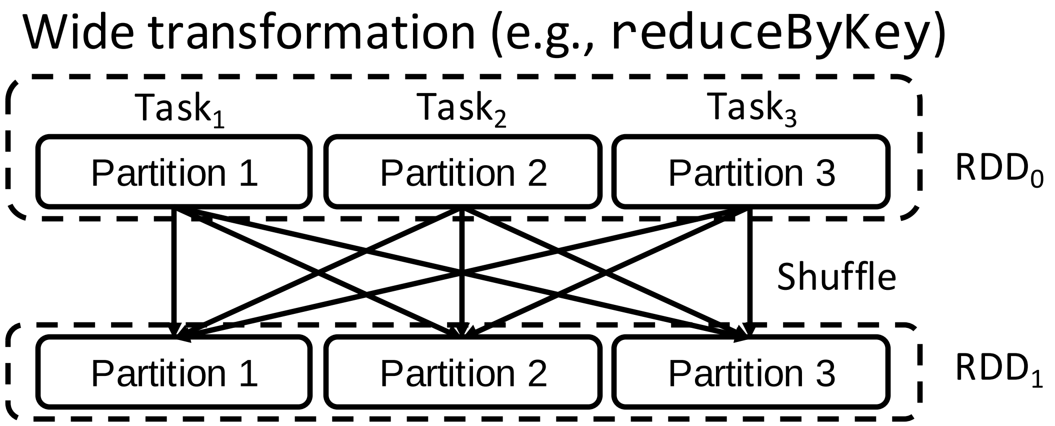
- RDD (Resilient Distributed Dataset): an immutable distributed collection of elements of data
  - Resilient: if data is lost, it can be recreated
  - Distributed: stored across the cluster
  - Dataset: collection of data records
- **Partition:** an atomic piece of the dataset stored in a node
- **Task:** an execution unit created by Spark



Key Concept in Spark: Transformations

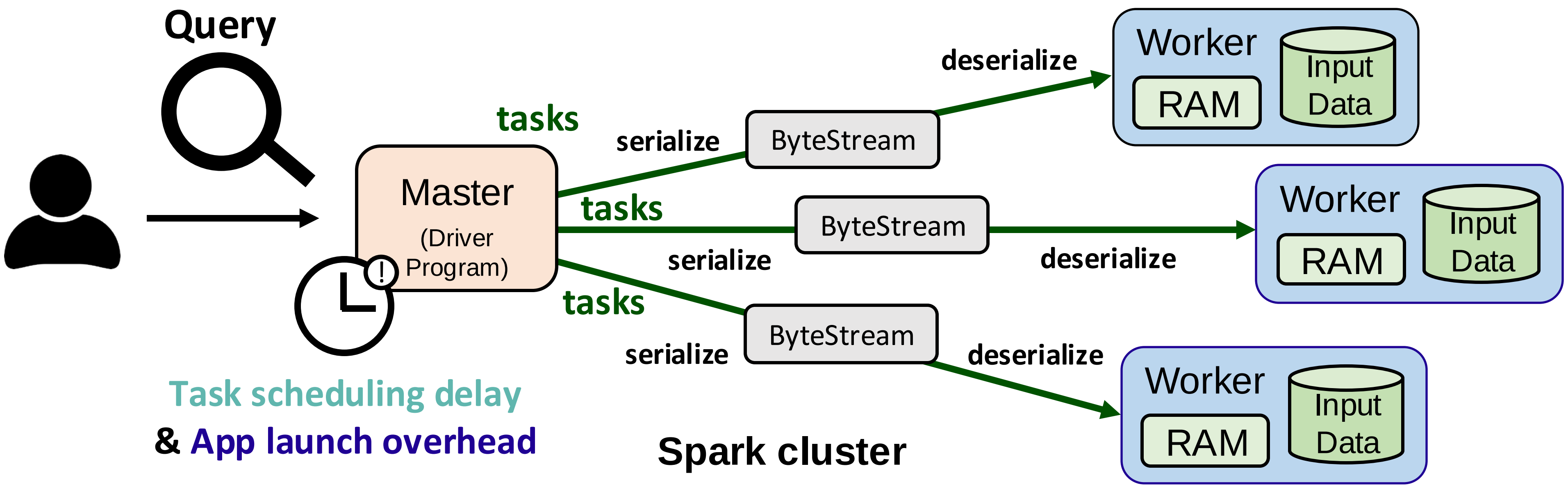
- **Narrow transformations** apply an operation to a single partition.
  - map, filter, flatMap, sample, ...
- **Wide transformations** require data to be shuffled or moved across multiple partitions.
  - join, groupByKey, reduceByKey, ...





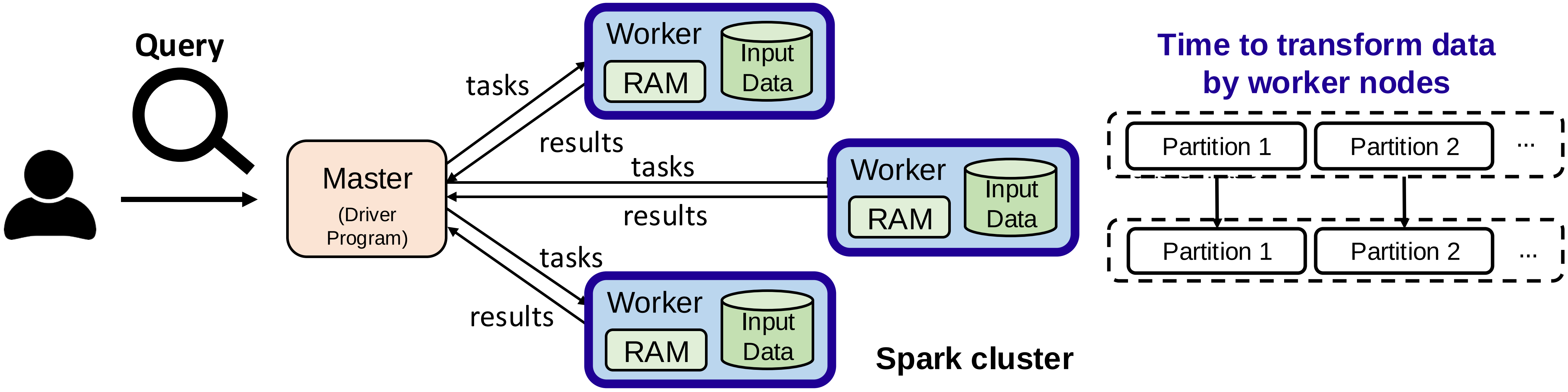
Setup Cost of Spark

- The total execution time = **setup time** + compute time
- The setup time includes
  - **Scheduler delay time**: waiting time to determine the order of tasks
  - **Task (de)serialization time**: time to (de)serialize tasks to send tasks over the network
  - **Application launch overhead**: startup of executor JVMs, resource allocation



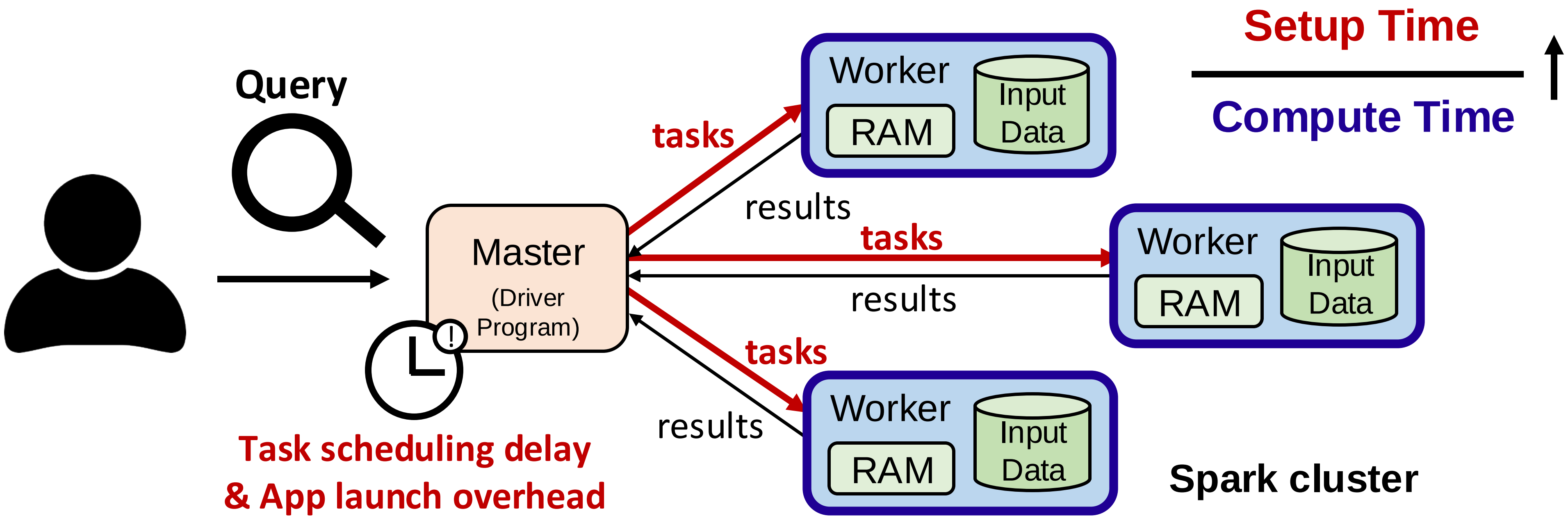
Compute Cost of Spark

- The total execution time = setup time + **compute time**
- The compute time includes
  - **Executor computing time**
  - **shuffle read/write time**



Problems with Running Small Queries in Spark

Problem 1. **Too large setup time** compared to **actual computation time**



Problem 2. **Insufficient degree of parallelism**

Too few number of partitions ⟶ low parallelism

Key Idea in Our Solution: Query Merging

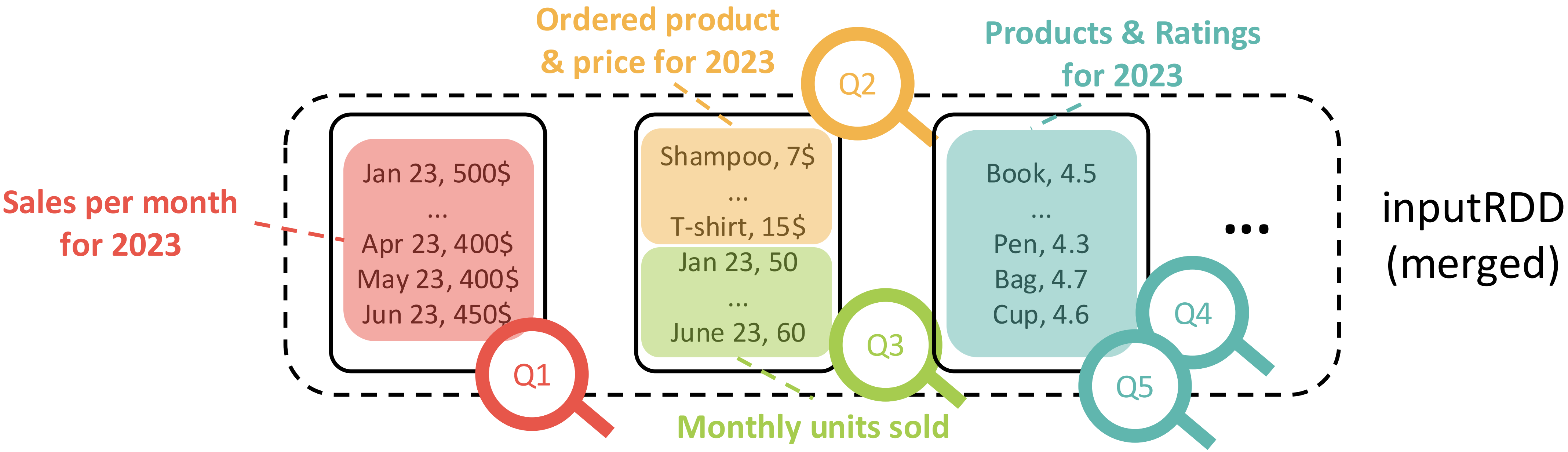
Query Merging: a massive number of small queries ⟶ a big query

Spark performs  for this merged inputRDD

A big query with Q1-Q5 merged together

Key Idea in Our Solution: Query Merging

Query Merging: a massive number of small queries ⟶ a big query



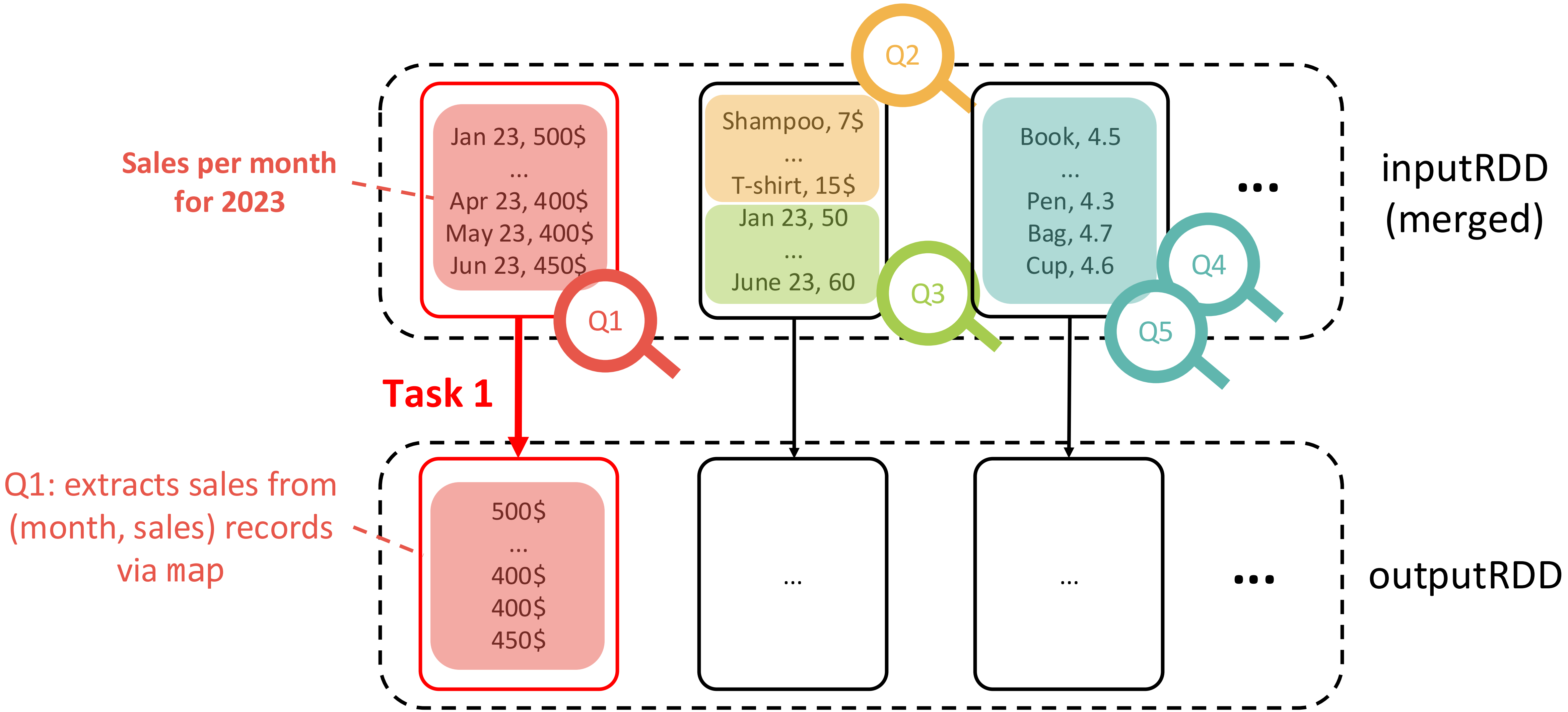
Solving Problem 1. Improvement of setup-to-compute time ratio

- Individual setup time per query is eliminated

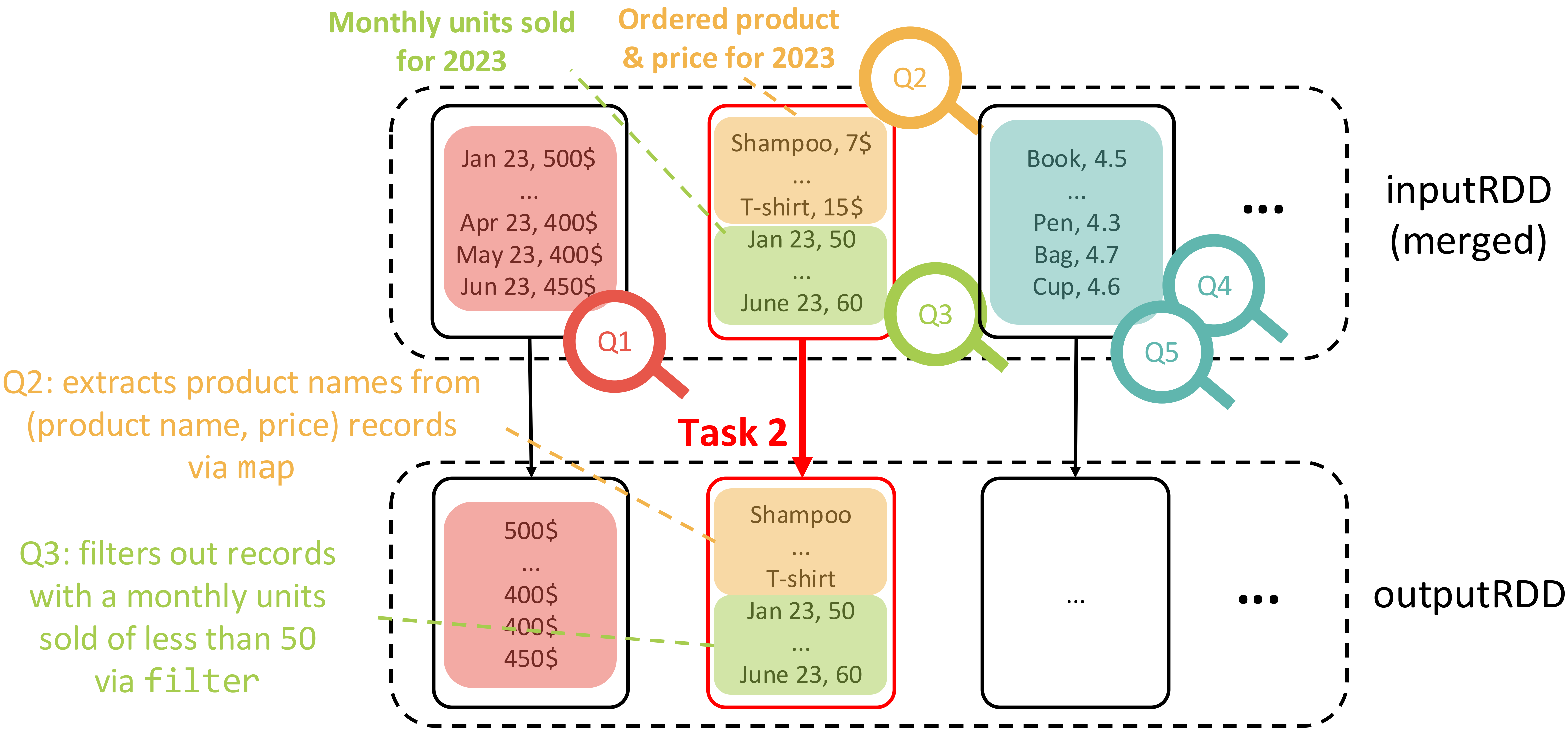
Solving Problem 2. Higher parallelism

- Large merged data leads to many partitions

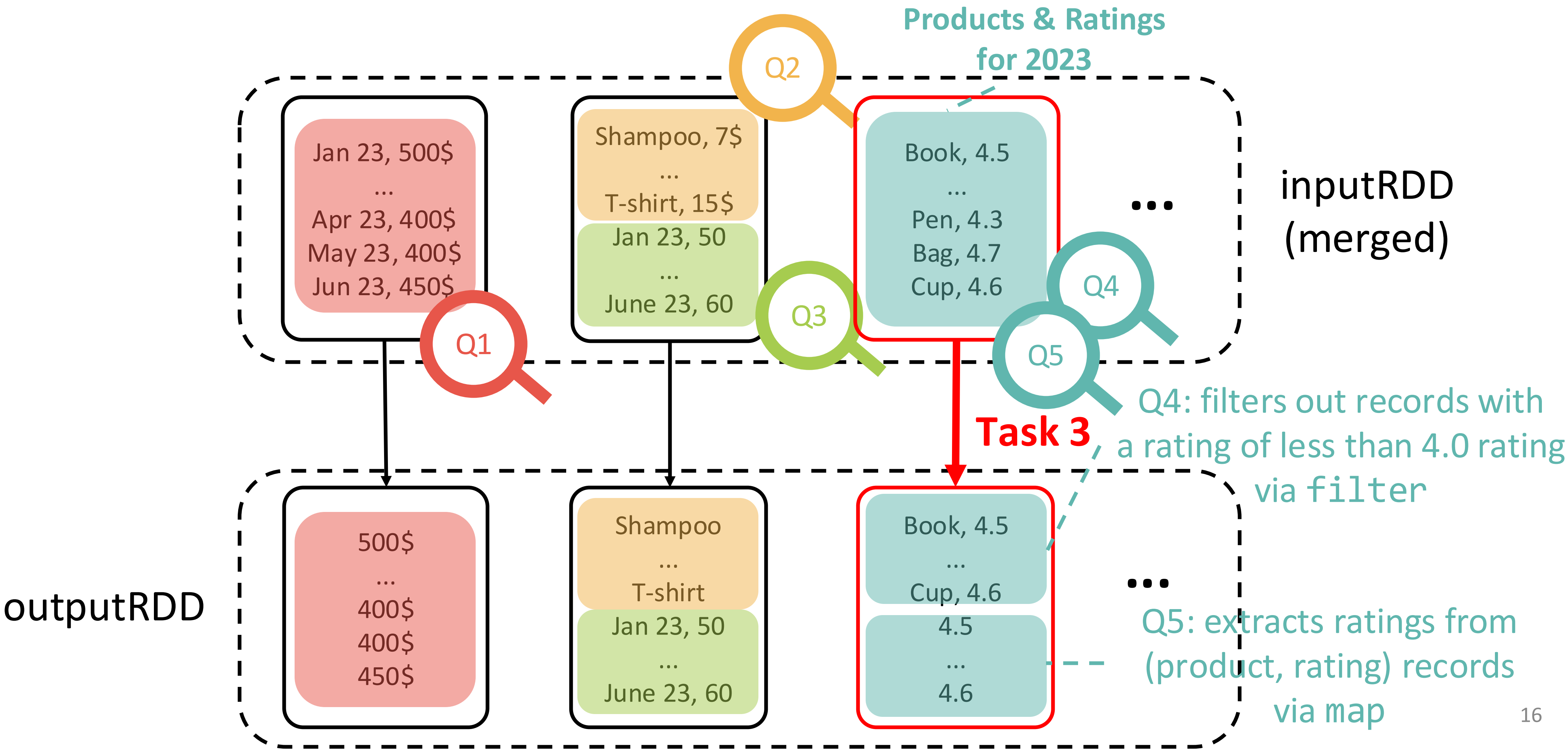
Key Idea in Our Solution: Query Processing of Task 1



Key Idea in Our Solution: Query Processing of Task 2



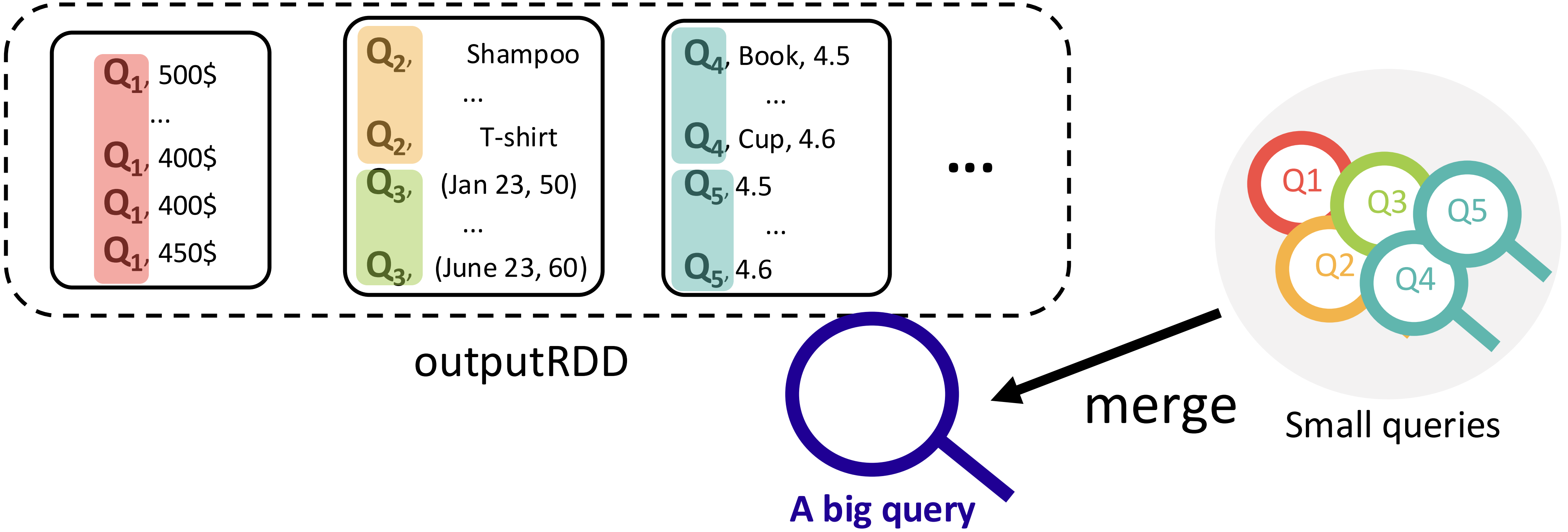
Key Idea in Our Solution: Query Processing of Task 3



Query Embedding

How to recognize records for different queries in an RDD?

- We need to identify which query each record is associated with in an RDD.
- **Embedding of the query information** (i.e., query ID Q) **into data** (i.e., records)



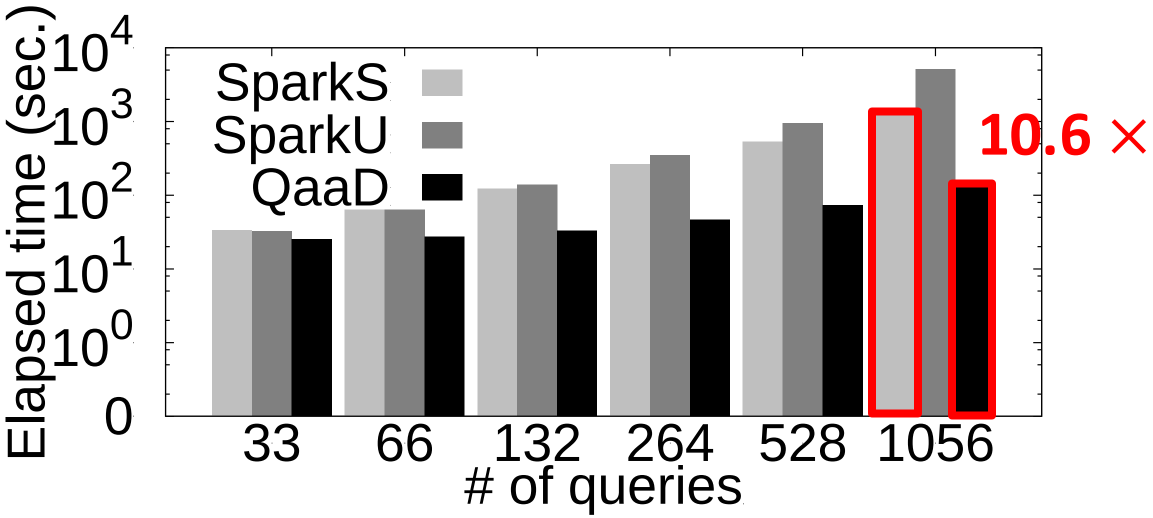
Details in Our Paper

- APIs for small query processing
  - Supporting the same transformation methods as RDD
- Detailed RDD transformations for merged operations
  - Including wide-dependency operations (e.g., `join`, `reduceByKey`)
- Adaptive partitioner (*microPart*)
  - Optimizing the partitions for small queries to reduce network overheads

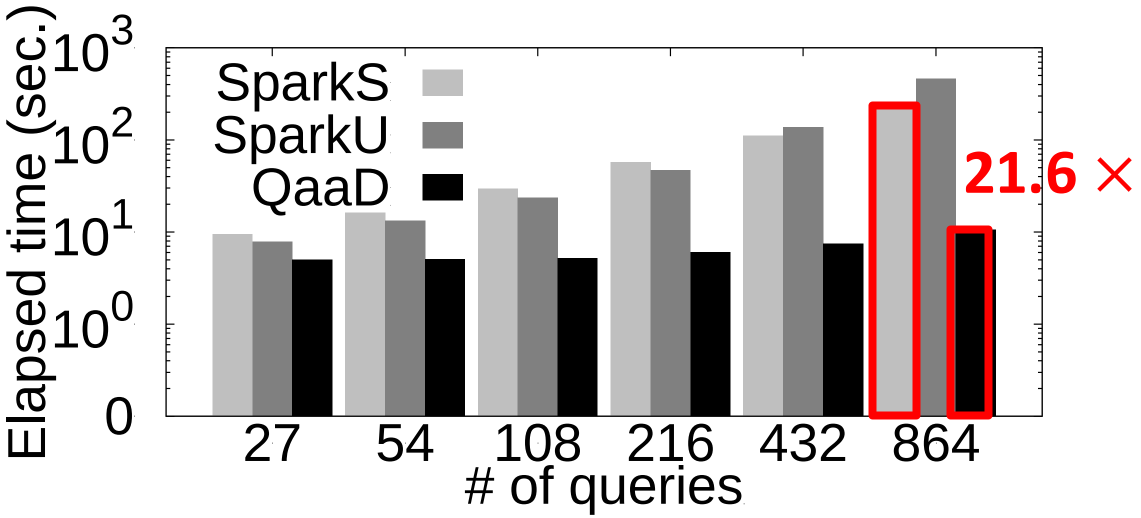
Experimental Setup

- Cluster setup
  - One master and four worker machines
  - Each executor used 14 cores and 128 GB RAM to run Spark applications.
- Compared techniques
  - SparkS: the standard way of using Spark where all queries are submitted and processed **individually** and **independently**
  - SparkU: combining **small queries in a given workload with a UNION operator**
- Two real-world datasets
  - BRA: A dataset with 100K records of orders collected between 2016 and 2018 on a Brazilian online marketplace
  - eBay: Transactions for auction details on eBay
- Query workloads obtained from the interface of amazon seller central

Evaluation – Number of Queries on Performance



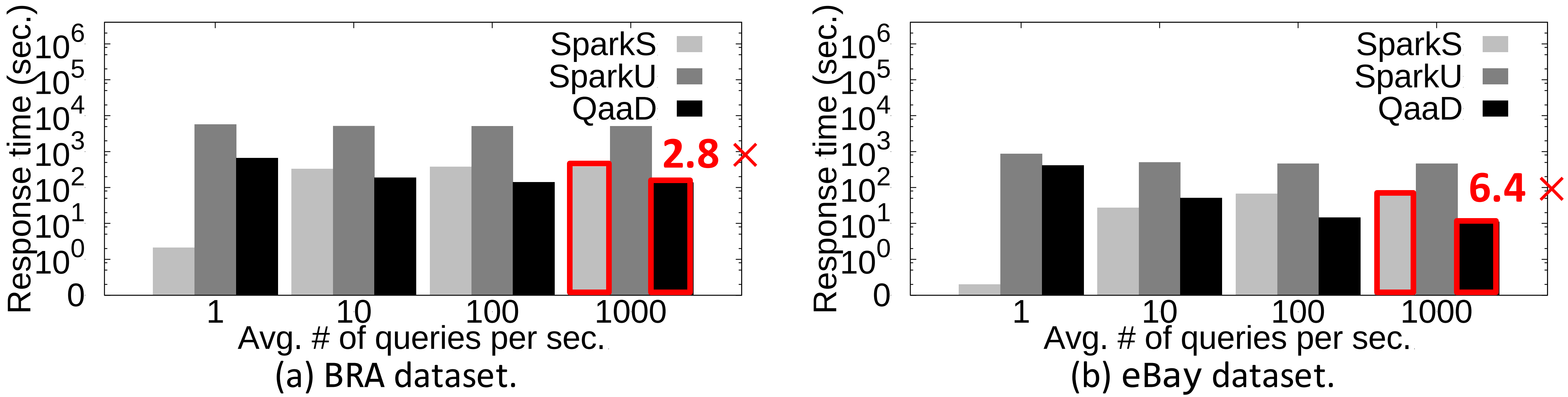
(a) BRA dataset.



(b) eBay dataset.

- Clear trends of the widening performance gap between QaaD and the other two compared techniques as the query size scales up
- 10.6 × and 21.6 × speed-ups against SparkS for BRA and eBay datasets at the highest workload

Evaluation – Arrival Rate on Performance



- The response time of QaaD improves quickly as the arrival rate increases.
- QaaD outperformed SparkS by 2.8 × and 6.4 × at the arrival rate of 1000 queries/sec for BRA and eBay datasets.

Conclusion

- A significant performance improvement of the Spark on **workloads made of a large number of small queries**

- **'Transform the workload'** to conform to what Spark was designed for to utilize its strong point - distributed parallel processing on a large-sized dataset

- Verification of **an order of magnitude improved performance** on small query workloads through comprehensive evaluations