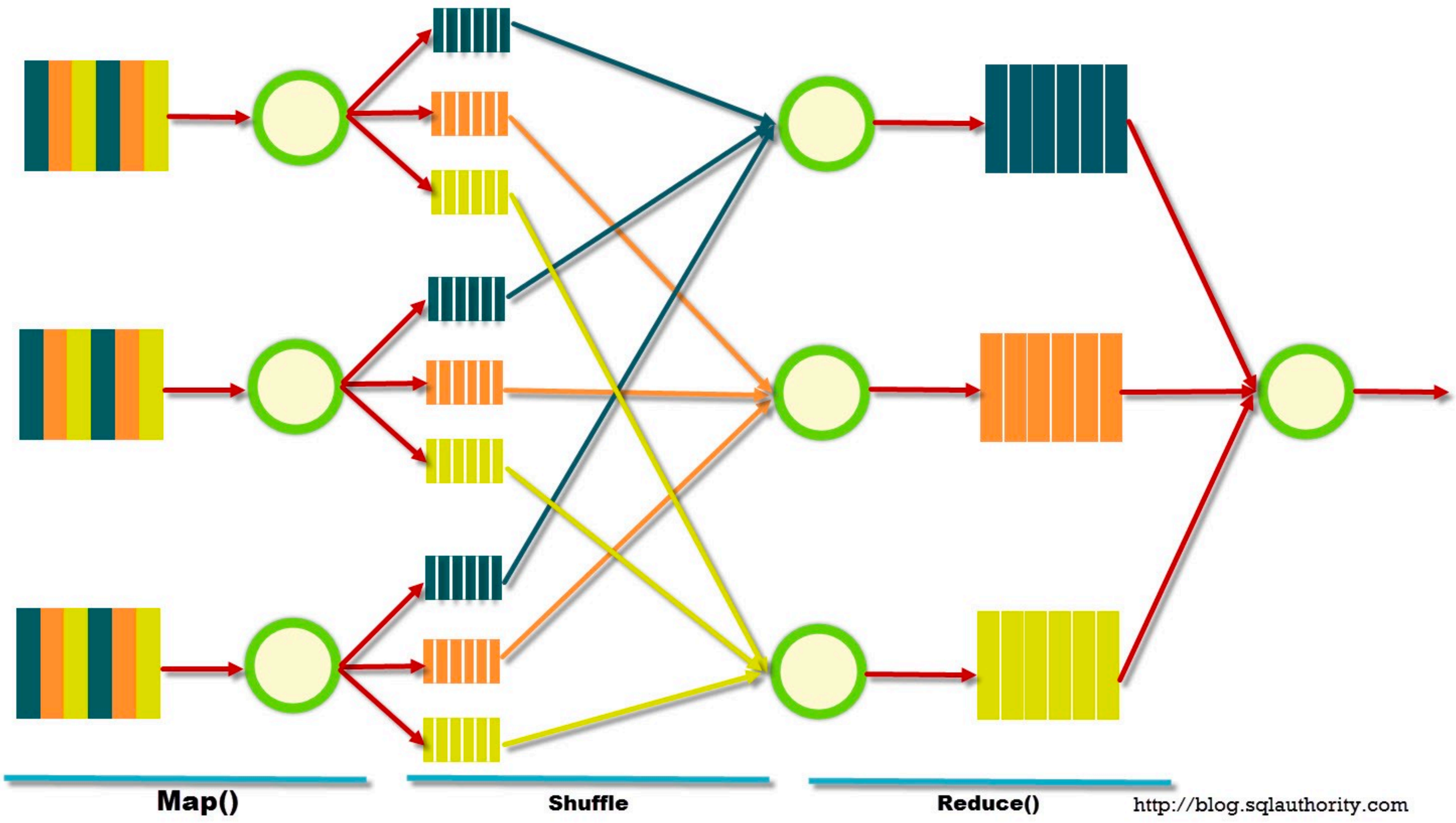
Riffle: Optimized Shuffle Service for Large-Scale Data Analytics

Haoyu Zhang，Brian Cho，Ergin Seyfe，Avery Ching，Michael J. Freedman

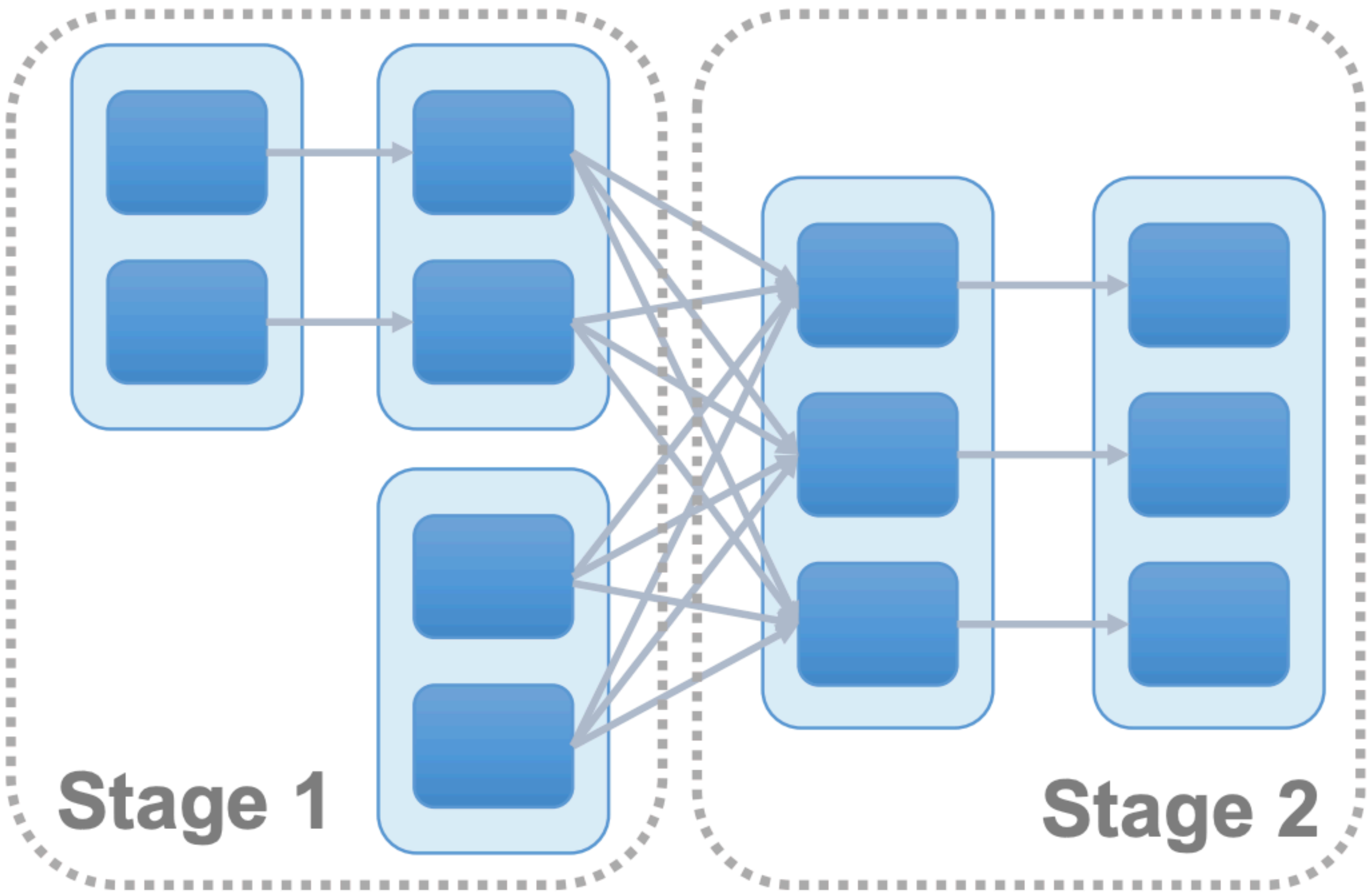
Zhiyuan Ma 2020.10.9

Hope you are doing well …

How MapReduce Works?



Batch analytics jobs: DAG execution plan



- Shuffle: all-to-all communication between stages
- >10x larger than available memory, strong fault tolerance requirements
  → on-disk shuffle files

Shuffle : All-to-All communications

- Each map task reads from a data partition (e.g., several rows of a large table), transforms the data into the intermediate format with the map task operators, sorts or aggregates the items by the partitioning function of the reduce stage (e.g., key ranges) to produce blocks of items, and saves the blocks to on-disk intermediate files.

- The map task also writes a separate index file which shows the offsets of blocks corresponding to each reduce task.

- Each reduce task brings together the designated data blocks and performs reduce task operators. By looking up the offsets in index files, each reduce task issues fetch requests to the target blocks from all the map output files.

The case for tiny tasks

- Benefits of slicing jobs into small tasks
  - Improve parallelism [Tinytasks HotOS 13] [Subsampling IC2E 14] [Monotask SOSP 17]
  - Improve load balancing [Sparrow SOSP 13]
  - Reduce straggler effect [Dolly NSDI 13] [SparkPerf NSDI 15]

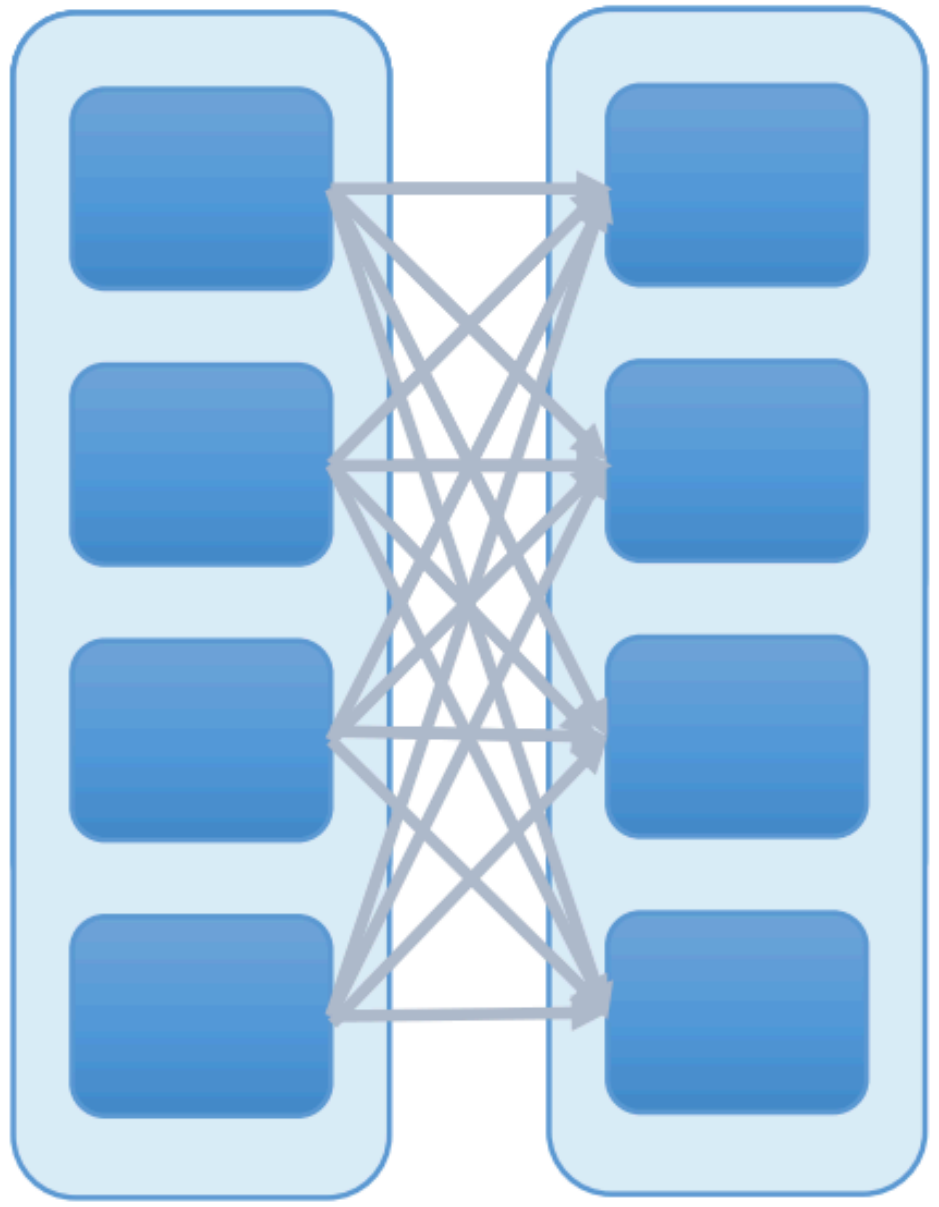
The case against tiny tasks

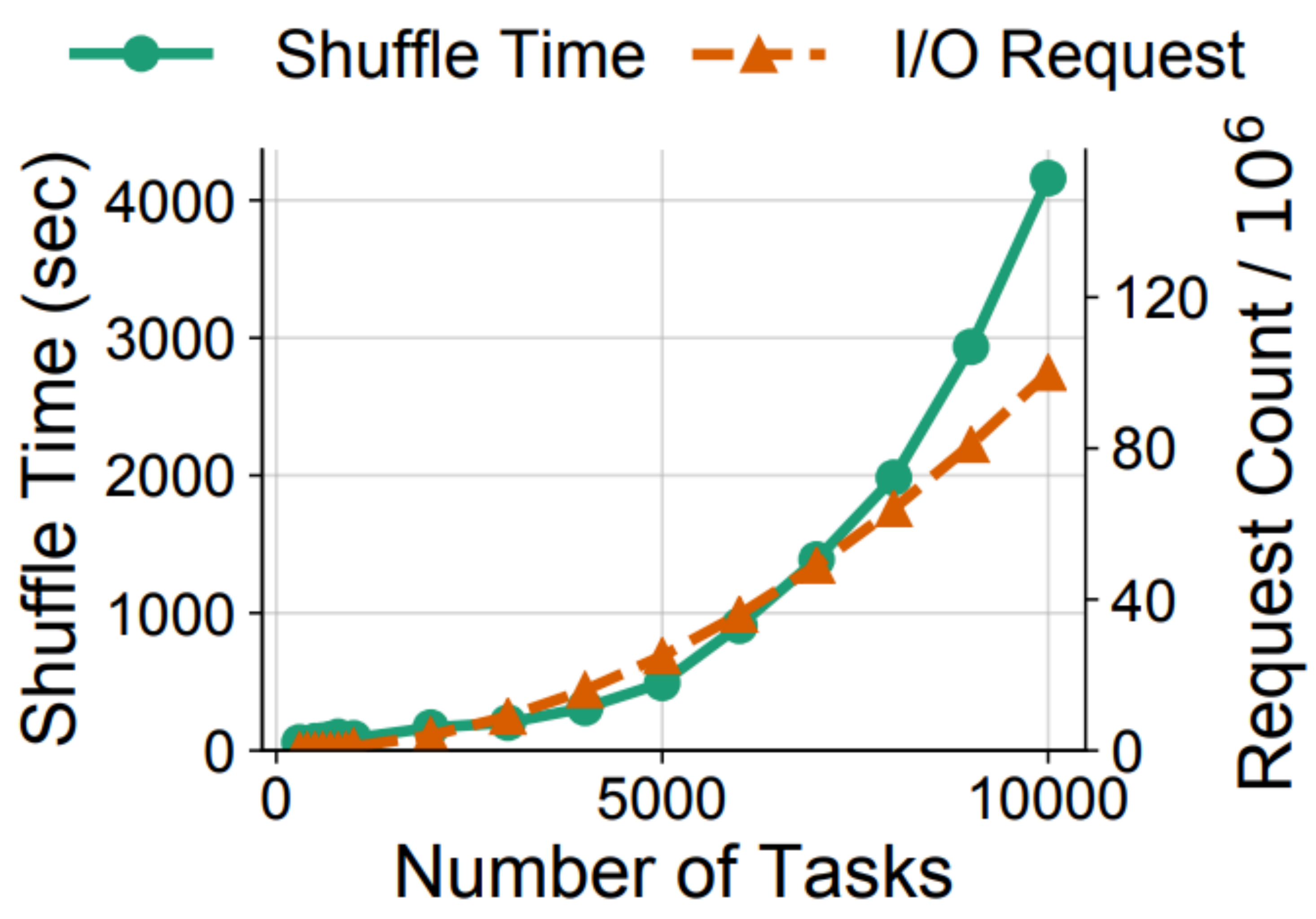
> *Although we were able to run the Spark job with such a high number of tasks, we found that there is significant performance degradation when the number of tasks is too high.*

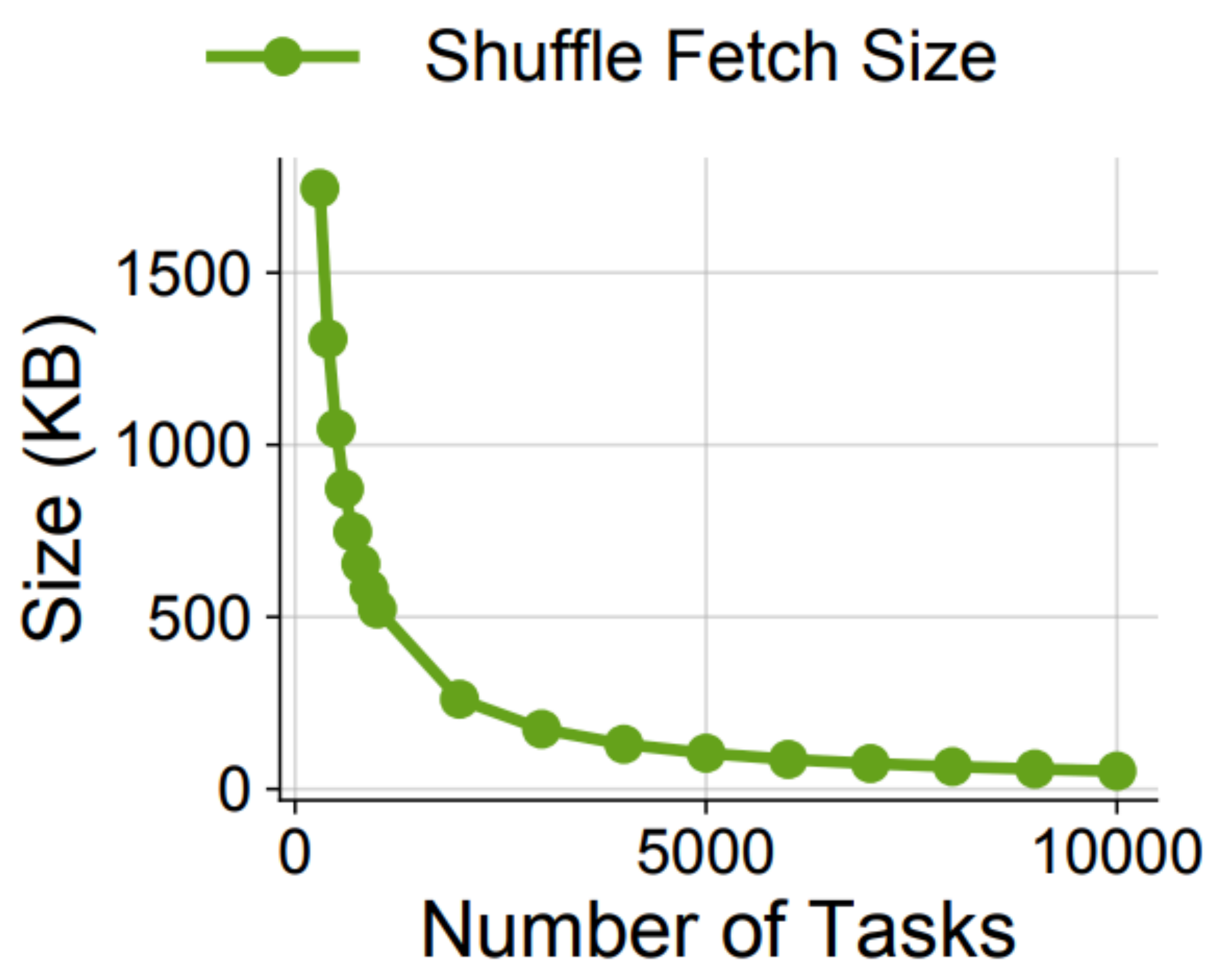
- Engineering experience often argues against running too many tasks
  - Medium scale → very large scale (10x larger than memory space)
  - Single-stage jobs → multi-stage jobs (> 50%)

[*] Apache Spark @Scale: A 60 TB+ Production Use Case. https://tinyurl.com/yadx29gl

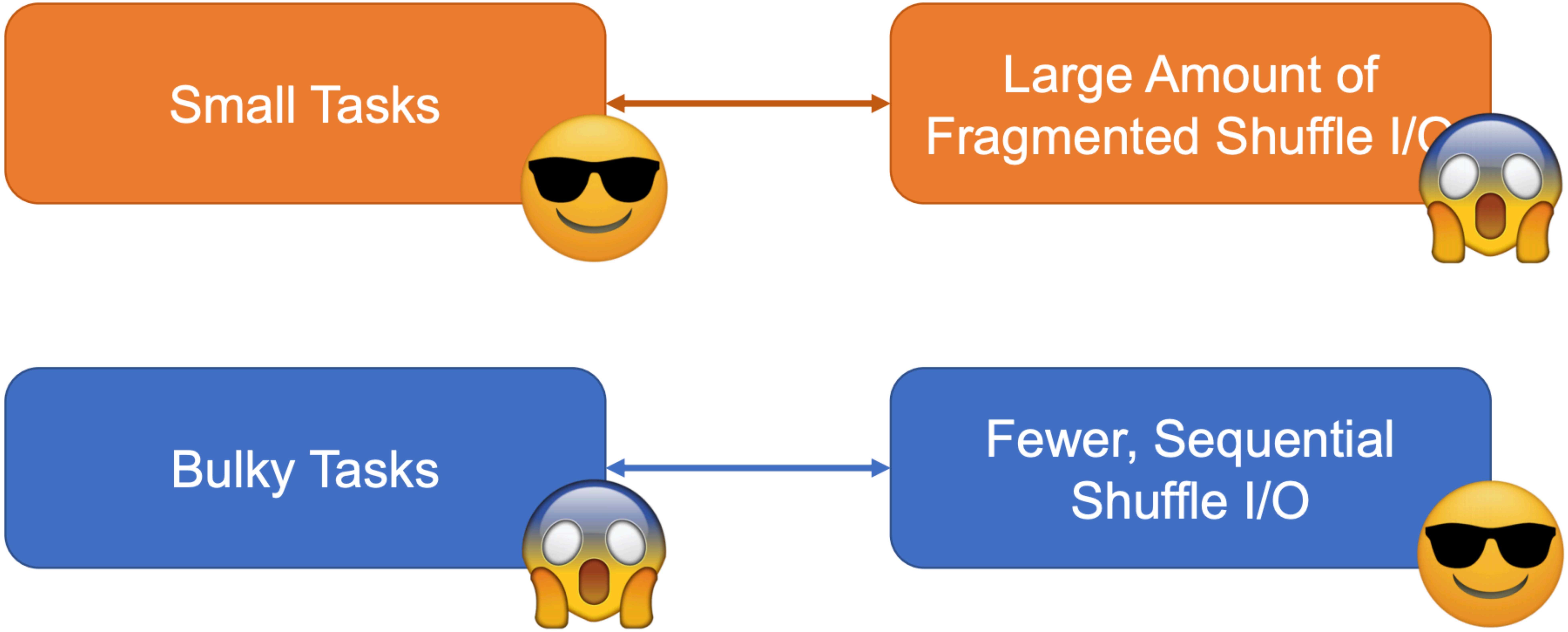
Shuffle I/O grows *quadratically* with data



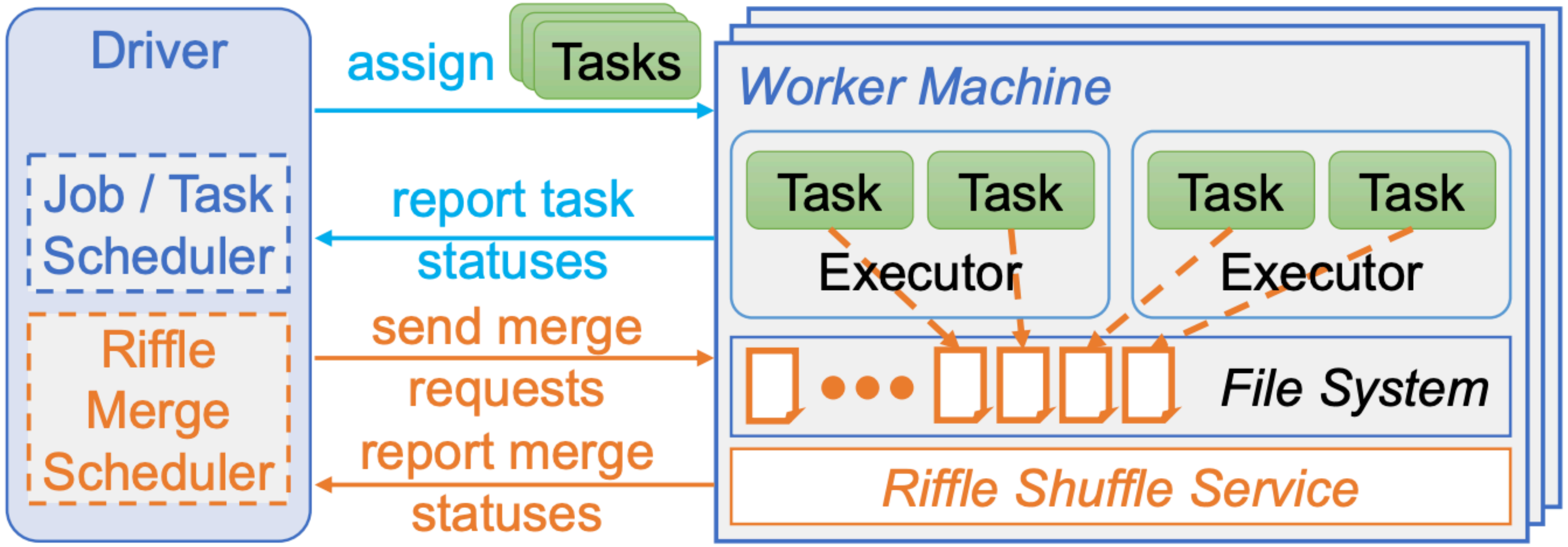




- Large amount of fragmented I/O requests
  - Adversarial workload for hard drives!


Small Tasks
Large Amount of Fragmented Shuffle I/O
Bulky Tasks
Fewer, Sequential Shuffle I/O

Riffle: optimized shuffle service



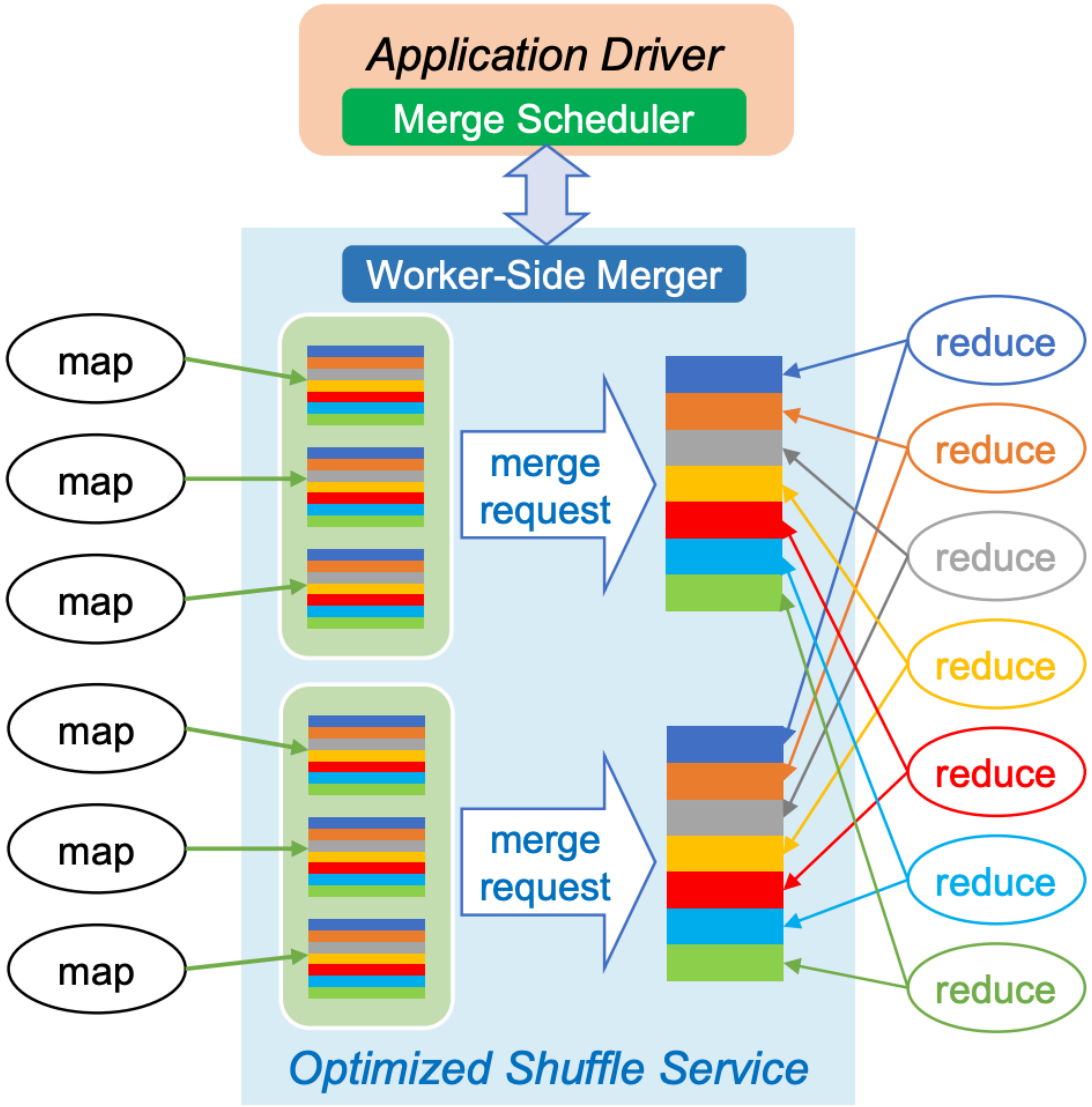
- Riffle shuffle service: a long running instance on each physical node
- Riffle scheduler: keeps track of shuffle files and issues merge requests

Riffle: optimized shuffle service

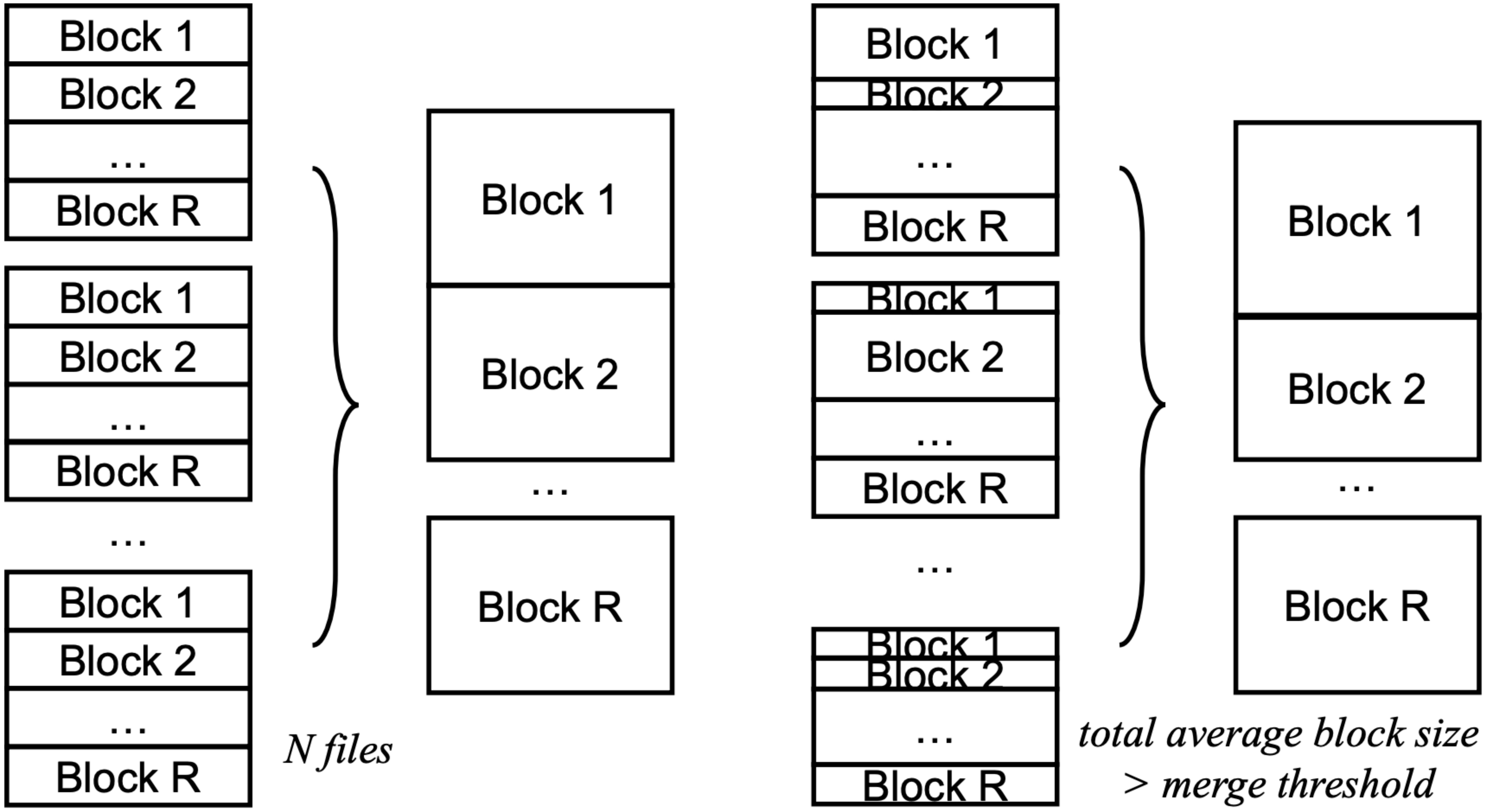
- When receiving a merge request

1. Combines small shuffle files into larger ones
2. Keeps original file layout

- Reducers fetch fewer, large blocks instead of many, small blocks

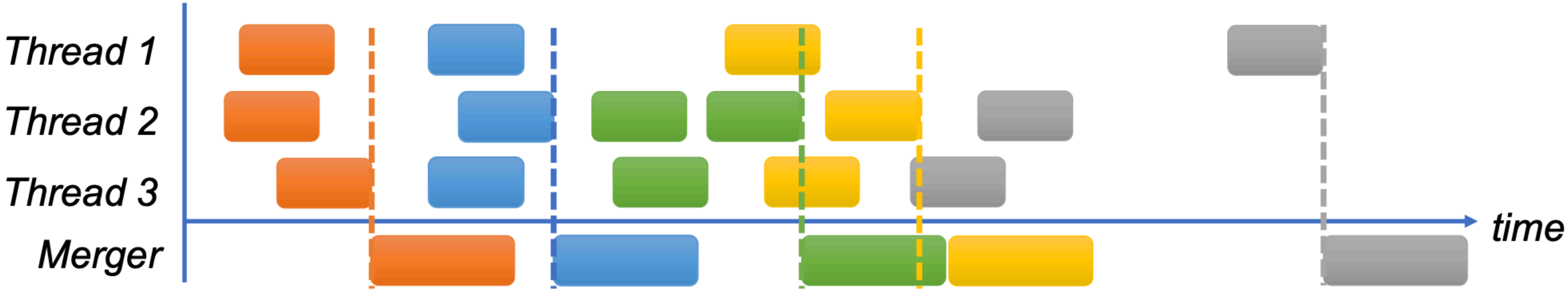


Riffle merge policies



Best-effort merge

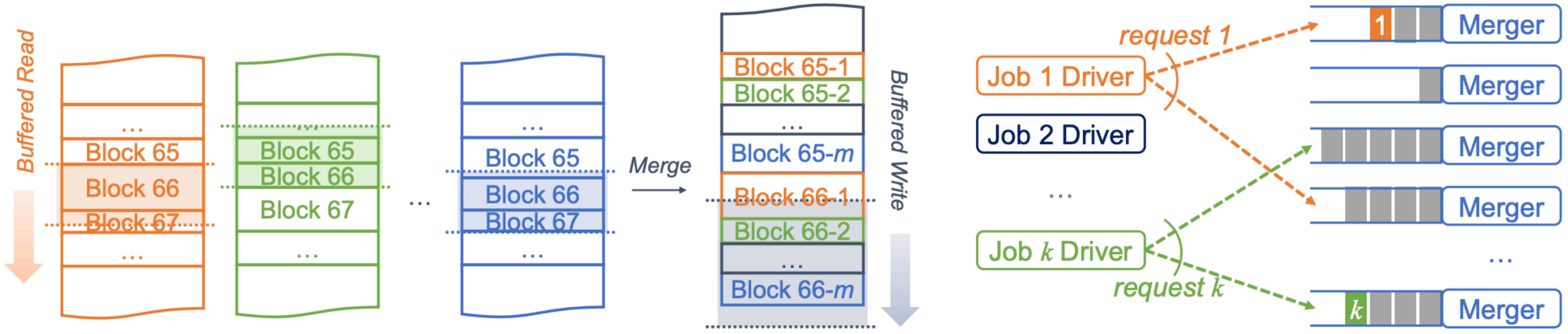
- Observation: slowdown in map stage is mostly due to stragglers



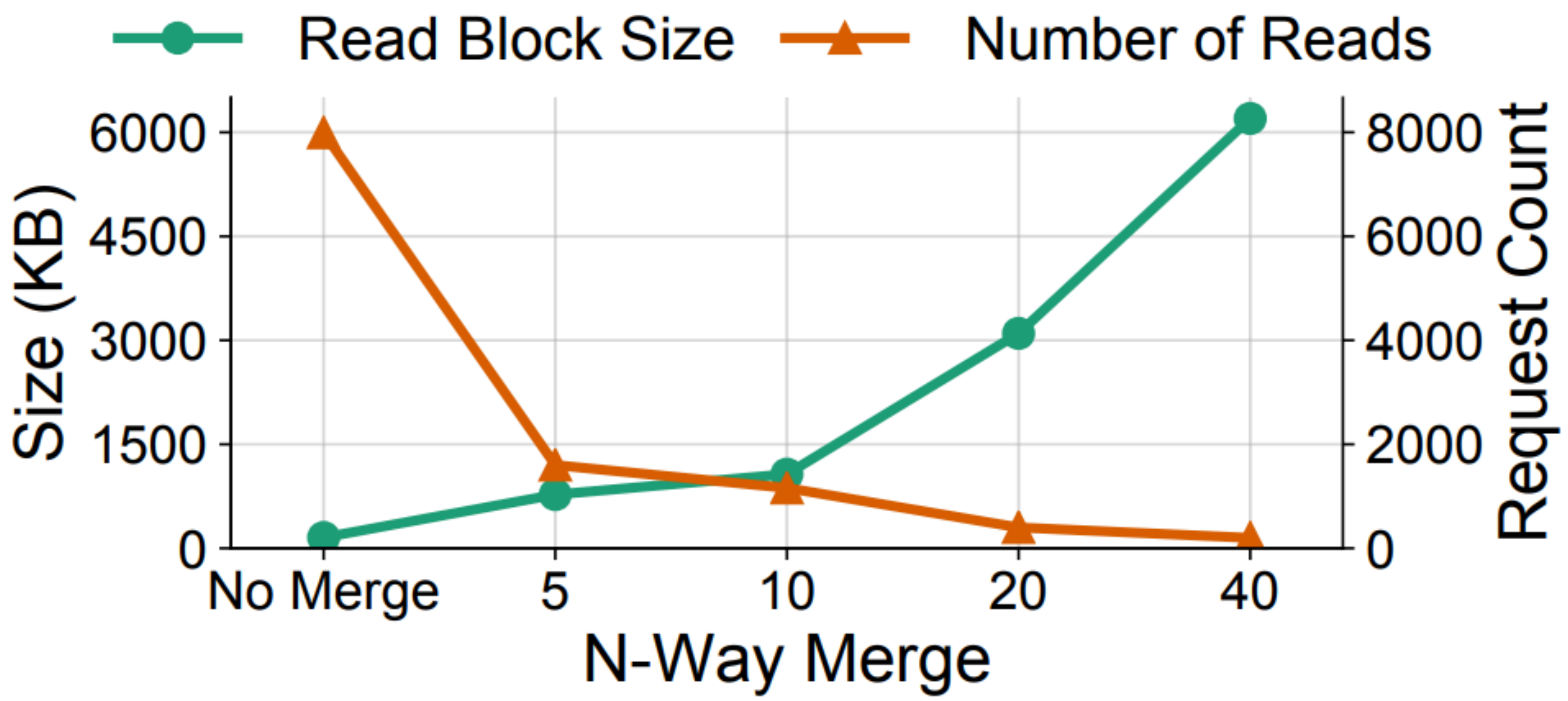
- Best-effort merge: mixing merged and unmerged shuffle files
  - When number of finished merge requests is larger than a user specified percentage threshold, stop waiting for more merge results

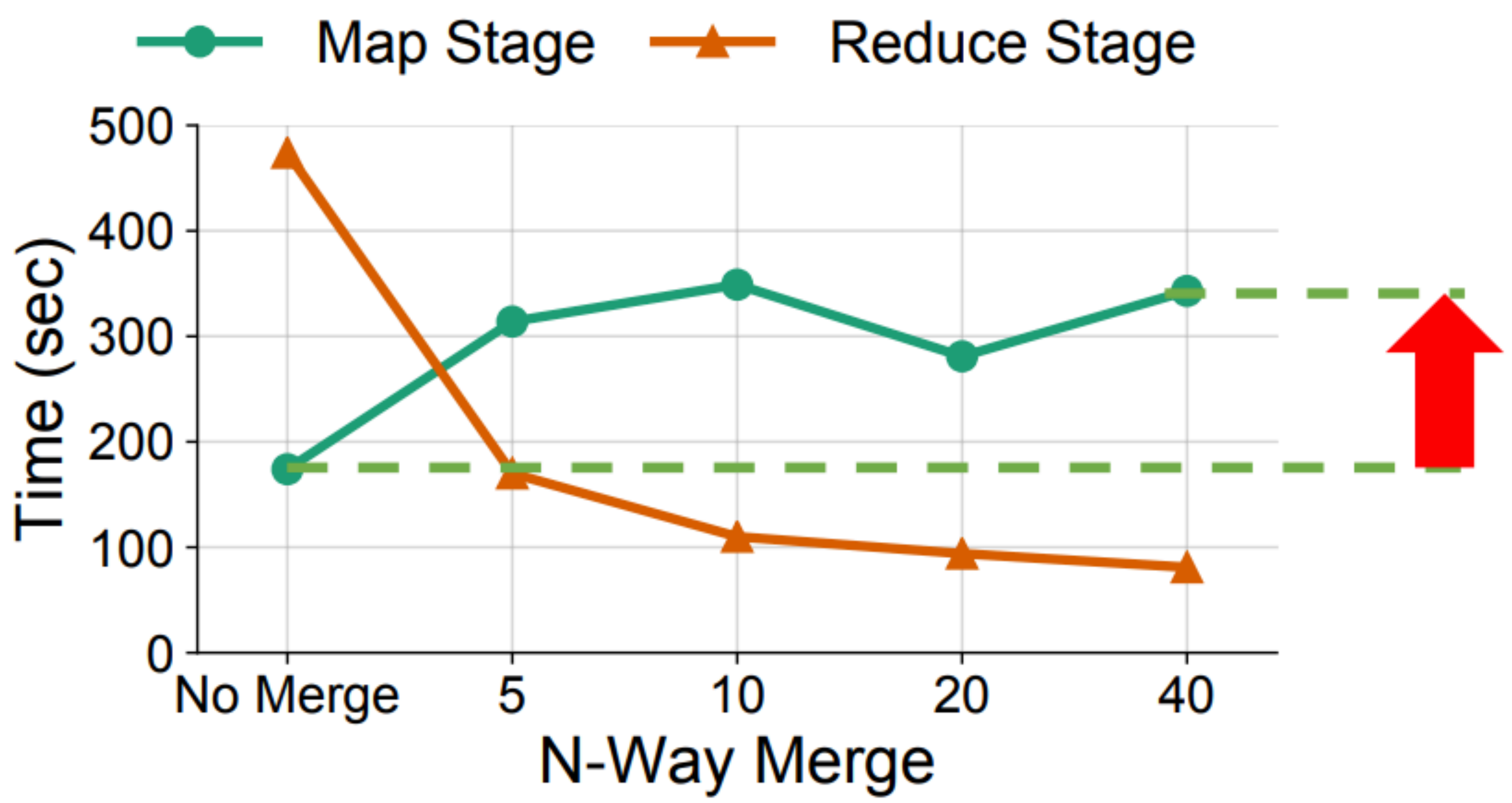
Additional enhancements

- Handling merge operation failures
- Efficient memory management
- Balance merge requests in clusters



Results with merge operations on synthetic workload





- Riffle reduces number of fetch requests by 10x
- Reduce stage -393s, map stage +169s → job completes 35% faster

Thanks for your attention