Optimizing Cloud Dataflow Pipelines

Janani Ravi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Structuring user code for parallelization and distribution

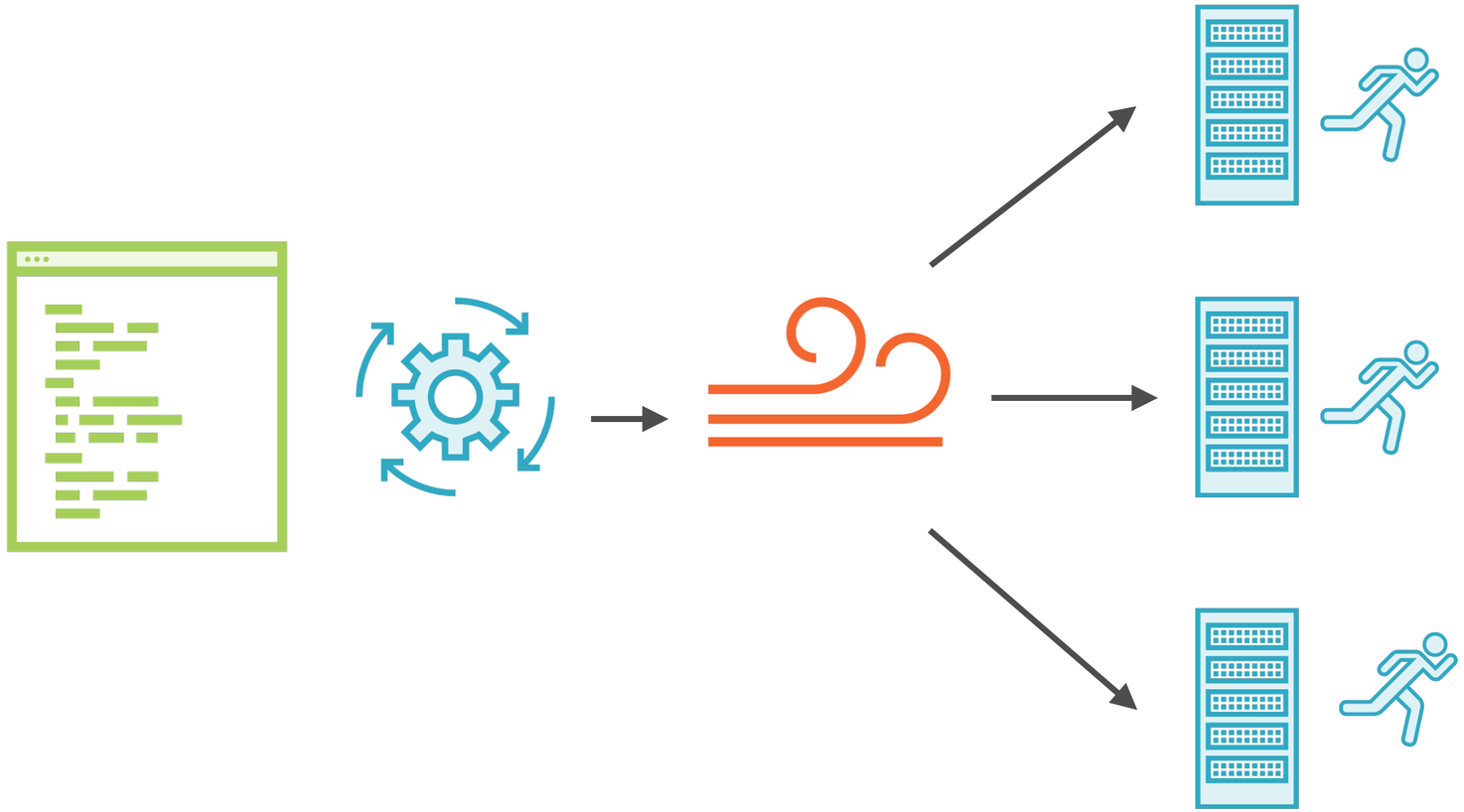
Understanding fusion and combine optimization of pipelines

Integrating pipelines with Pub/Sub and BigQuery

Debugging slow pipelines, errors, and retries

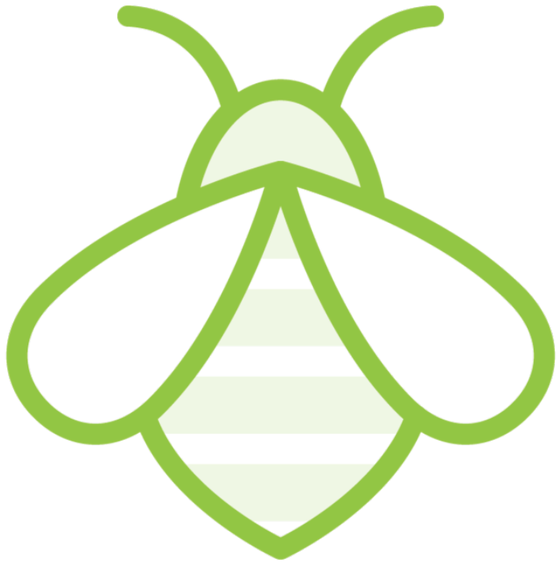
Performing windowing operations on streaming pipelines

Parallelization and Distribution



Dataflow automatically partitions your data and distributes your worker code for parallel processing

Worker Code



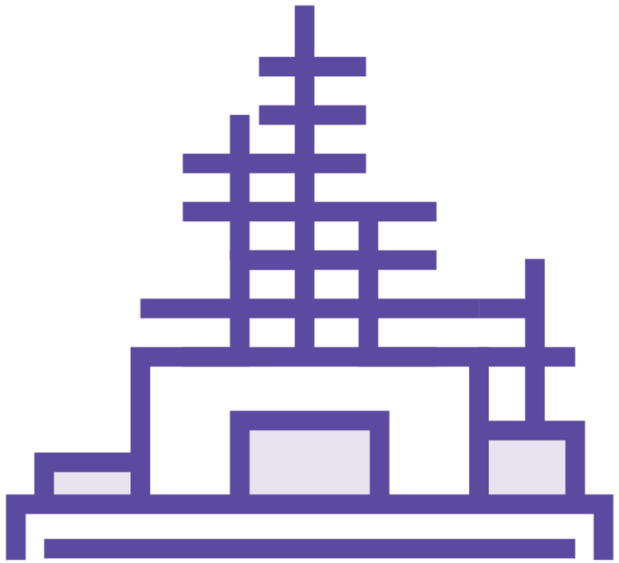
Processing code written using the `DoFn` **interface**

`DoFns` **are pure functions**

Many `DoFns` **run in parallel on multiple instances**

Transforms such as `ParDo` **cause** `DoFns` **to be distributed and run**

Structuring User Code



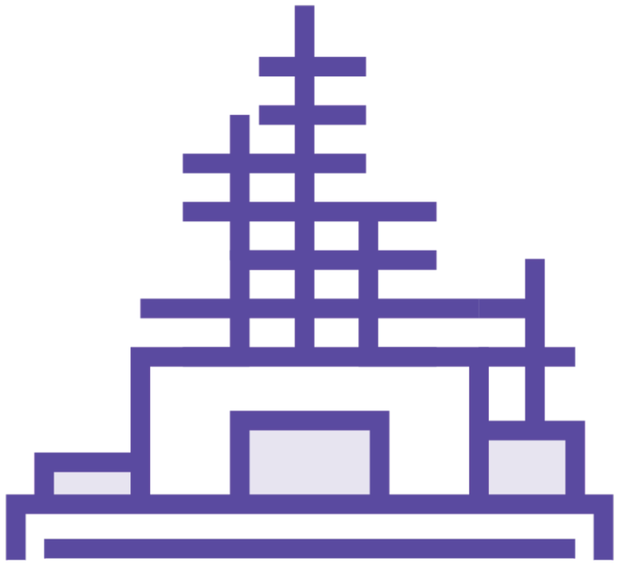
Every element in your PCollection will be processed by a `DoFn` exactly once

`DoFns` can be invoked any number of times

Any number of `DoFn` instances can be created when a pipeline executes

Any state shared between `DoFn` instances must be thread-safe

Structuring User Code



The Dataflow service does not guarantee:

- how elements will be grouped
- which elements will be processed together

User code may be retried multiple times

Ensure user code does not have unintended side effects

Demo

Implementing a Dataflow pipeline to write results to Pub/Sub

Demo

Implementing a Dataflow pipeline to write results to BigQuery

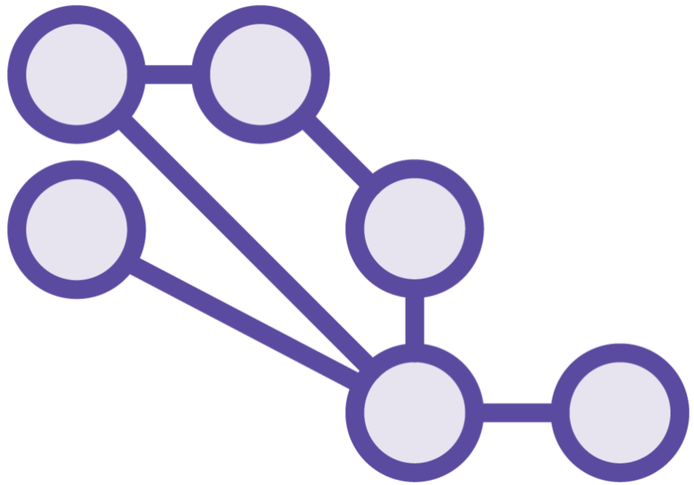
Demo

Performing join operations

Errors and retries in Dataflow

Optimizations in Dataflow Pipelines

Execution Graph

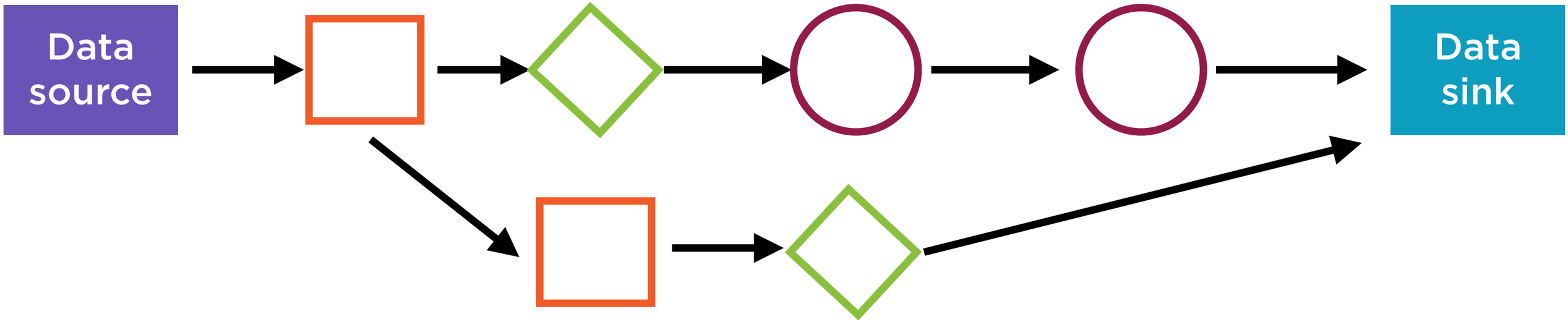


Dataflow creates an execution graph including all processing operations

Validates resources referenced by the pipeline

Checks for errors and illegal operations

Execution Graph



Once validated, Dataflow may modify the execution graph to perform optimizations

Optimizations in Dataflow

Fusion

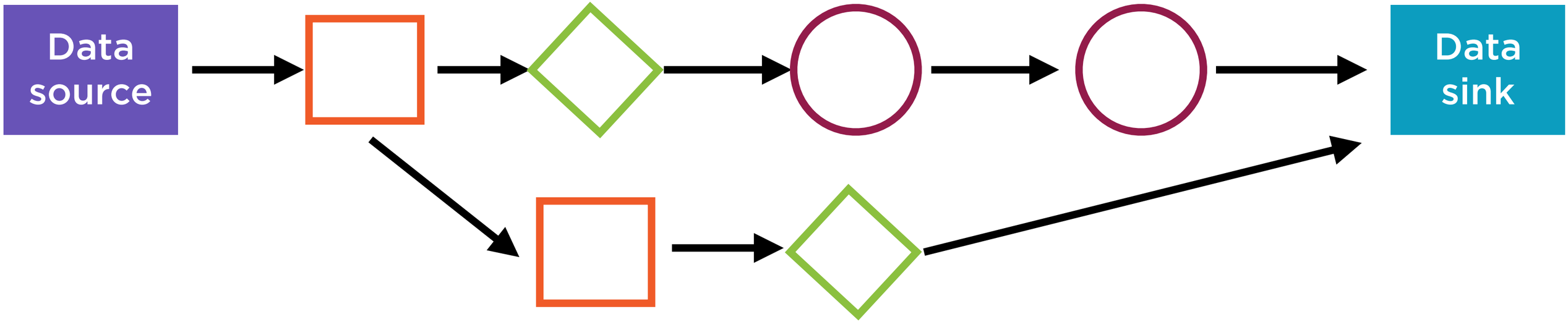
Combine

Optimizations in Dataflow

Fusion

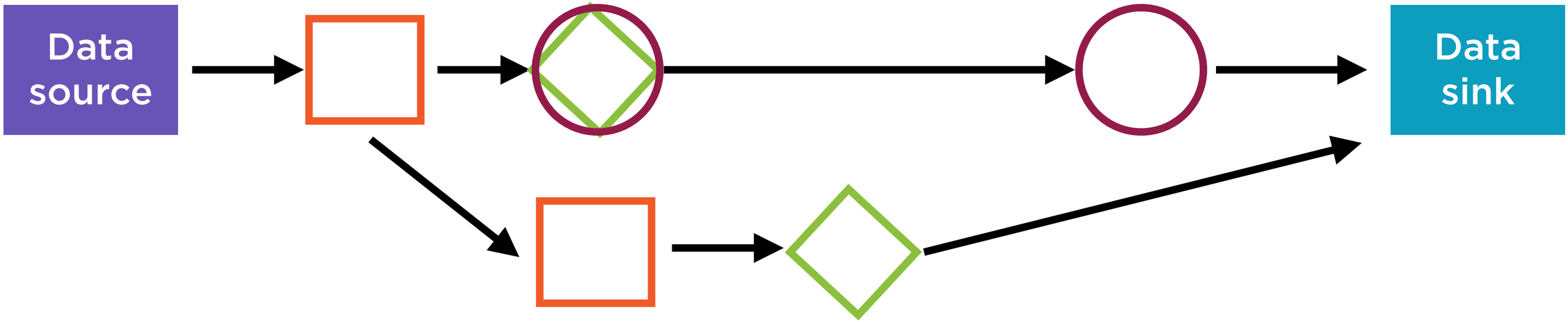
Combine

Fusion Optimization



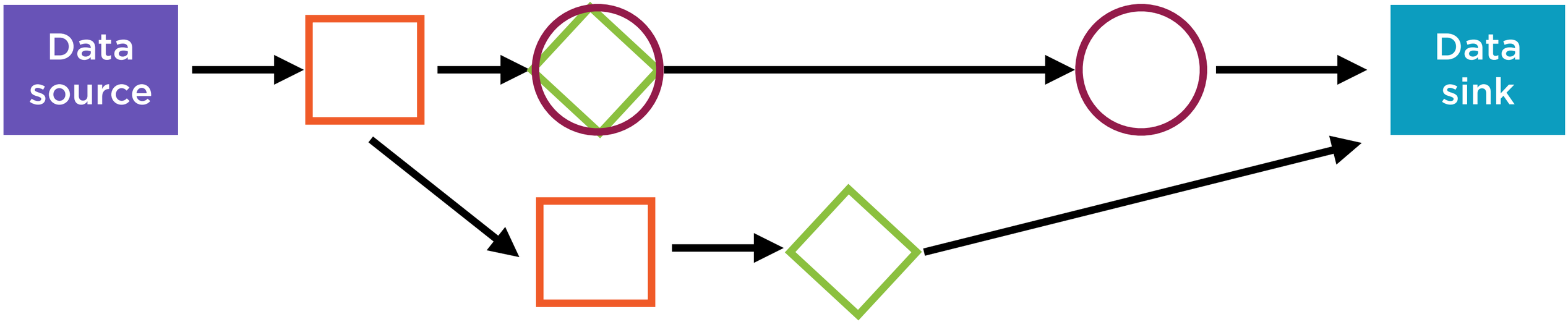
Dataflow execution materializes intermediate results in this graph - this incurs a heavy processing overhead

Fusion Optimization



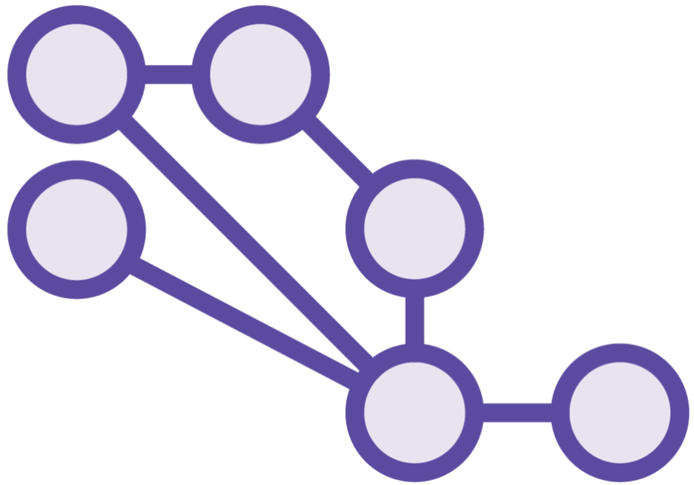
The fusion optimization involves fusing multiple steps or transforms into a single step

Fusion Optimization



Fusing reduces the number of intermediate results that need to be materialized

Fusion Optimization



Data dependencies between steps are respected

Transforms can be executed in any order for the most efficient execution

Preventing Fusion

Fusion may not always find the most optimal execution

Fusion can be prevented by adding operations which force results to be materialized

Preventing Fusion

Insert `GroupByKey` **and ungroup after first** `ParDo`

Pass intermediate PCollection as side input to another `ParDo`

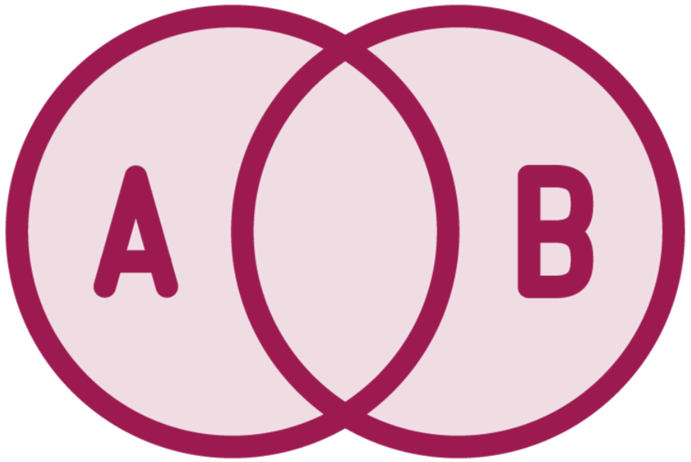
Insert a `Reshuffle` **step**

Optimizations in Dataflow

Fusion

Combine

Combine Optimization

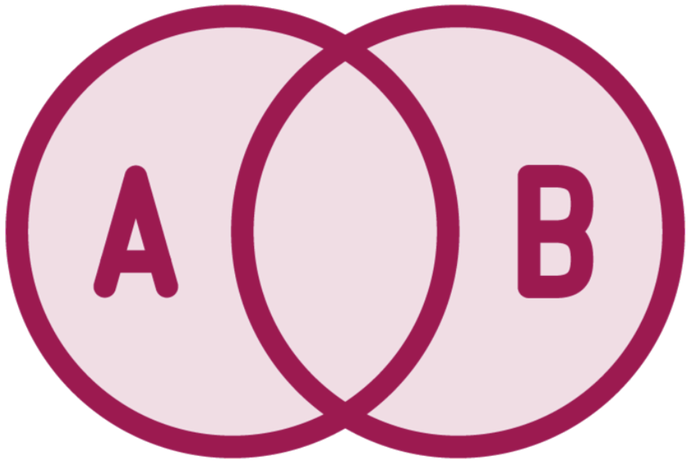


Aggregation operations in Dataflow:

- GroupByKey
- CoGroupByKey
- Combine

Aggregation combines data across multiple workers

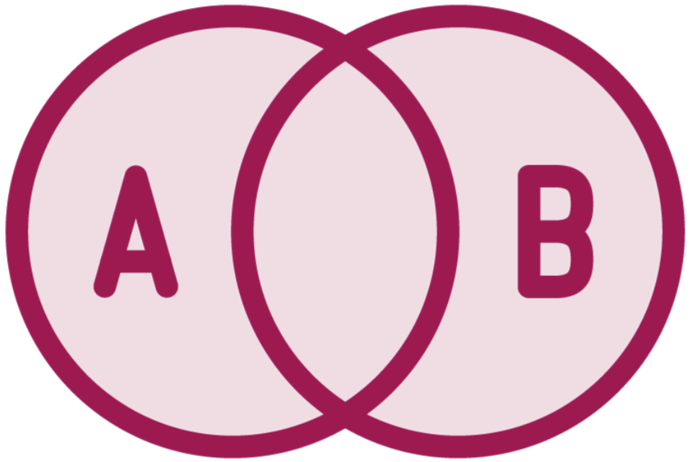
Combine Optimization



Efficient to combine as much data as possible locally i.e. on a single worker

Dataflow automatically performs partial combining locally

Combine Optimization



For batch data, Dataflow will combine locally as much as possible

For streaming data, Dataflow may not perform partial combining

Partial combining increases latency

Autoscaling and Dynamic Work Rebalancing

Autoscaling

Dataflow chooses the right number of workers to run your job

Dynamically re-allocates more workers or fewer workers

Based on changes in load and resource utilization

Batch Autoscaling

Choose the number of workers based on estimated work

Re-evaluate the amount of work every 30 seconds to scale up or down

Streaming Autoscaling

Minimize backlog while maximizing worker utilization

No additional charge for autoscaling

Designed to reduce cost of resources used when executing streaming pipelines

Dynamic Work Rebalancing

Dynamically re-partition work based on runtime conditions

- Imbalances in work assignments
- Workers taking longer than expected
- Workers finishing faster than expected

Dynamic Work Rebalancing

Dynamically reassign work to unused or underused workers

Decreases job's processing time

Dynamic Work Rebalancing: Limitations

Fused steps may limit how much data processing can be rebalanced

Rebalancing cannot parallelize data finer than a single record

Demo

Reading streaming data from Pub/Sub

Performing windowing operations on streaming data

Summary

Structuring user code for parallelization and distribution

Understanding fusion and combine optimization of pipelines

Integrating pipelines with Pub/Sub and BigQuery

Debugging slow pipelines, errors, and retries

Performing windowing operations on streaming pipelines

Up Next:

Running Cloud Dataflow Pipelines Using Templates