Executing and Optimizing Hadoop MapReduce

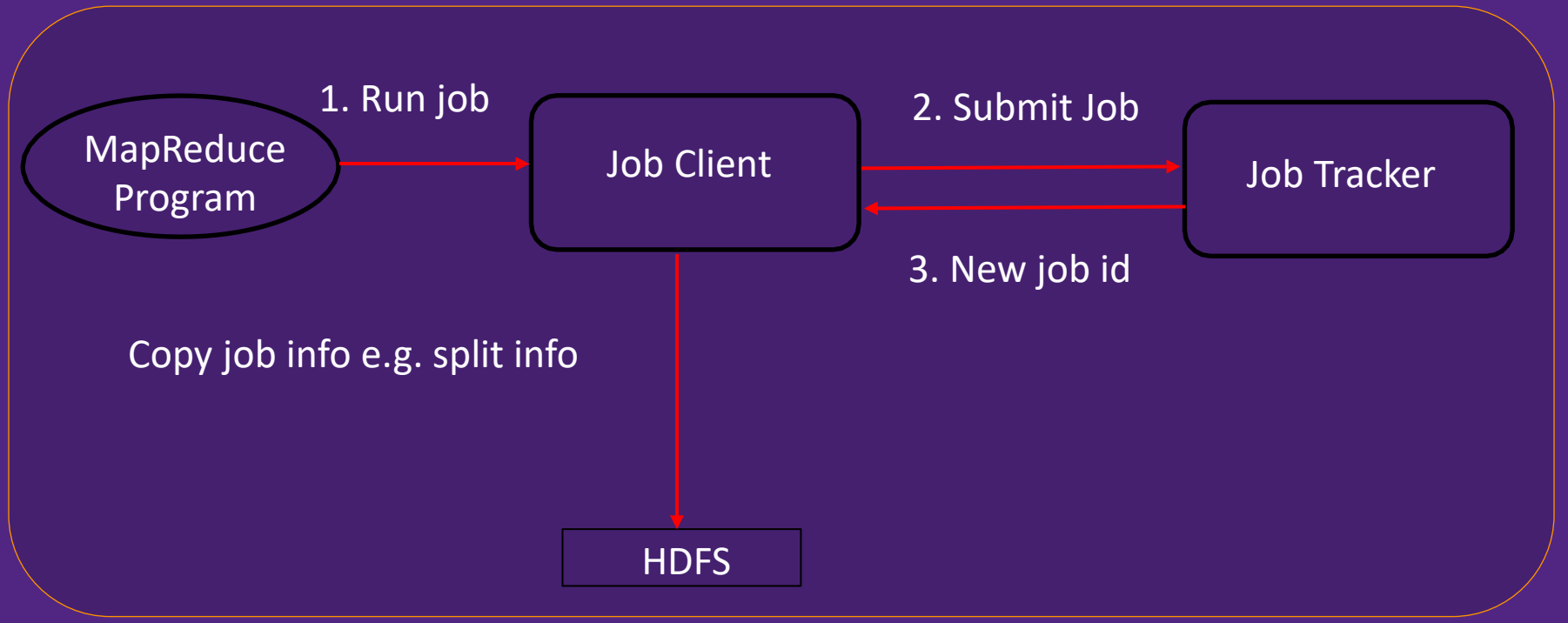
CS 4417B

The University of Western Ontario

Terminology

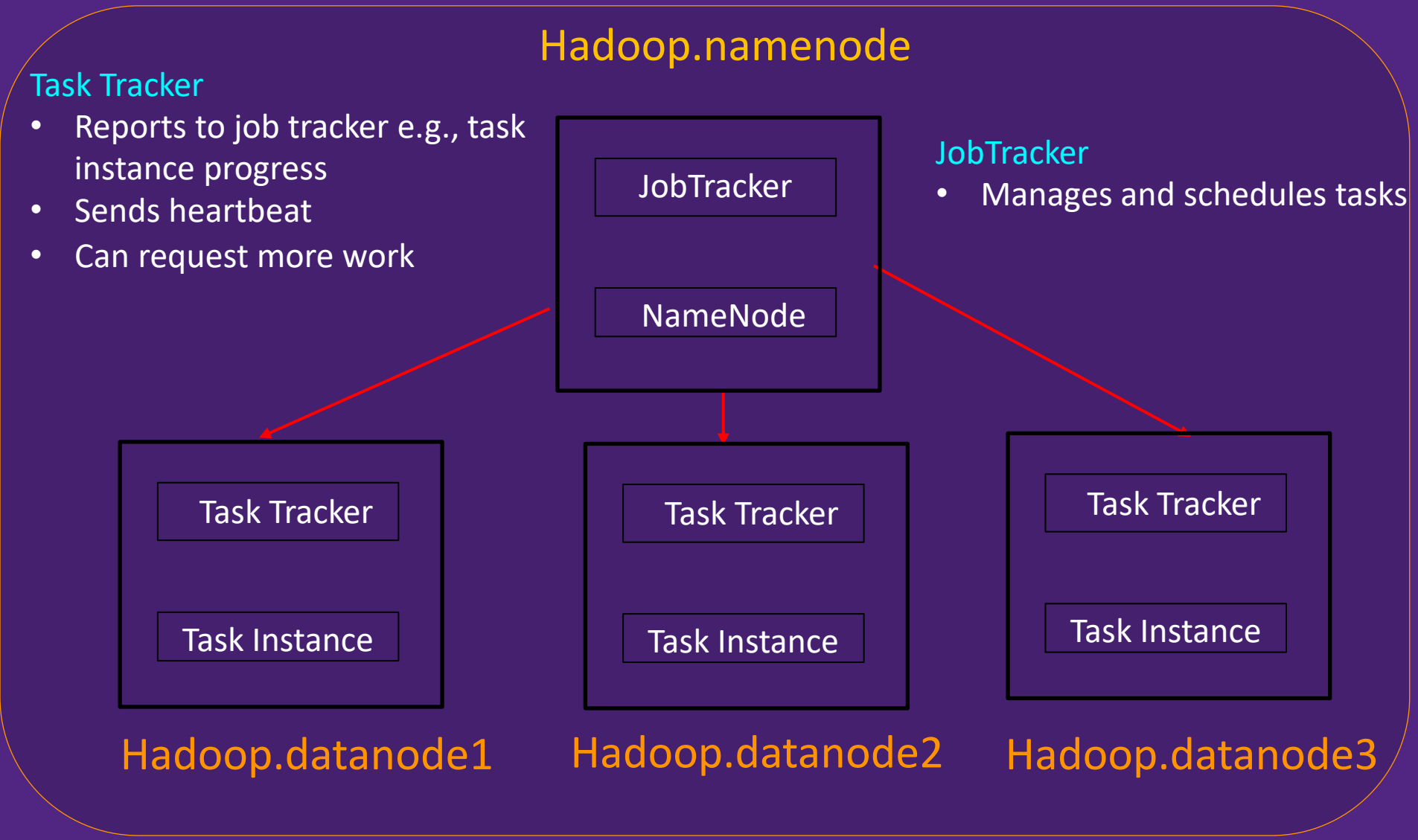
- A job is a “full program” with mapper and reducer code, input dataset and a location for the output
- A task is an execution of a mapper or a reducer on part of the data
- Example: Run “Word Count” across a file with 20 blocks is one job
  - Assume a file is assigned to one map task
  - 20 blocks typically result in 20 map tasks
  - These may send data to many reduce tasks

Execution Architecture



- JobClient
  - Submit job request

Execution Architecture



Job Processing

- JobTracker assigns work to TaskTrackers
- After map, the TaskTrackers exchange map-output information to build the reduce key space
- JobTracker partitions reduce() keyspace into m chunks, where m is set by user
  - Assigns work to reducers

Assigning to Reducers

- Given a set of keys $\{k_1 \ldots k_n\}$ how do we assign?
- Use a $\text{hash}(k_i)$
  - Actually $\text{hash}(k_i) \bmod R$ where R is number of reducers
- Issues?
  - Assumes equal distribution of data over hashes (that's why hashes work well)

How many Mappers and Reducers

- Mappers
  - By default this is the number of HDFS blocks being processed
  - The number of maps can also be controlled by specifying the minimum split size
- Reducers
  - This is a function of the number of nodes and the amount of data – for n nodes; two suggestions:
    - 0.95n (all reduces start immediately)
    - 1.75n (fast nodes can end up doing 2 reduce jobs)

References

- Data-Intensive Text Processing with MapReduce
  - Jimmy Lin and Chris Dyer
  - http://lintool.github.io/MapReduceAlgorithms/MapReduce-book-final.pdf

- Hadoop docs: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Optimizing using Local Aggregation: "Combiners"

Optimization

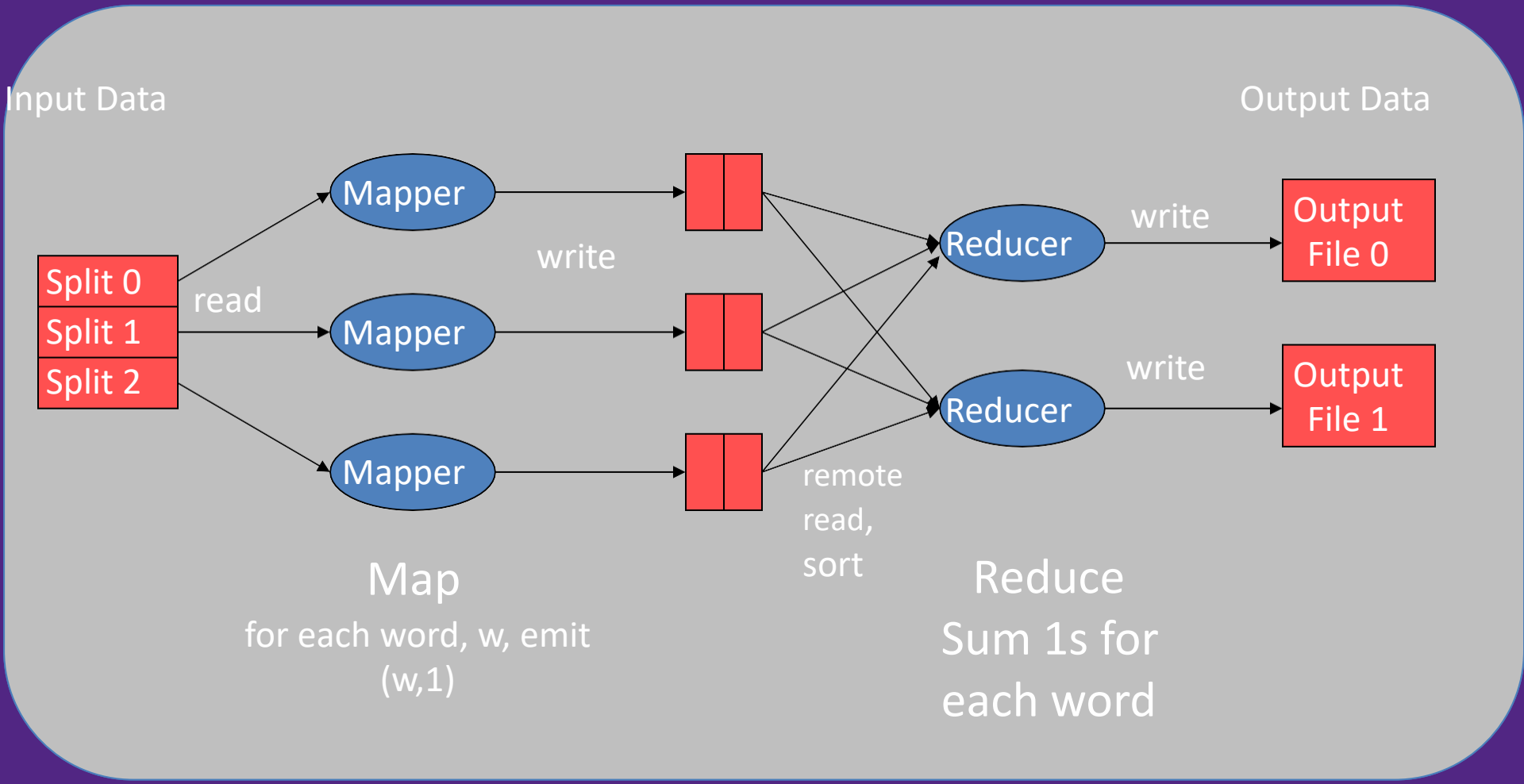
"Plain" MapReduce

- Map
  - One input pair -> one output pair
- Reduce
  - Many input pairs with same key -> one output pair
- By loosening this definition, we can improve performance

Revisiting Word Count

- Map function
  - For each word, w, generate (w,1)
- Reduce function
  - Reducers sum the ones for each word
  - Output: For each word, w,
    emit (w, number of times w occurs)

MapReduce Workflow



Revisiting Word Count

- Issue: Large number of key-value pairs
  - Example: (w,1)
- The output of the mapper is written to disk and read from the disk by reducer
- Writing and reading to disks is relatively slow
- Potential for network congestion

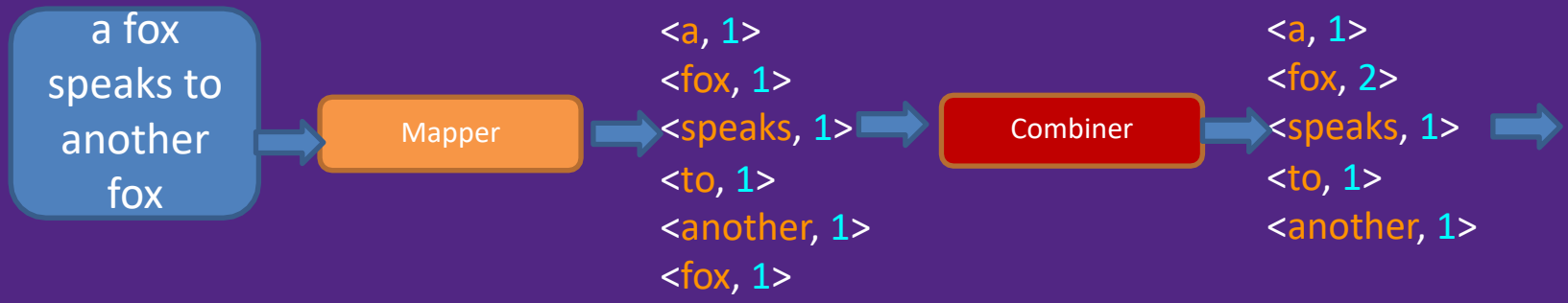
Use Combiners

- Perform local aggregation on the output of the map function but before the shuffle/sort phase
  - Word Count: Count occurrence of each word locally resulting in a pair (w,n) representing n occurrences of the word w
  - The number of intermediate pairs that go to the shuffle/sort is the number of unique words from that map
- This works when the operation on the pairs is associative and commutative

Use Combiners

- Hadoop Combiner is an optional class in the MapReduce framework
  - The combiner receives intermediate pairs from mappers
- You can save networking time by doing local aggregation
- When might this not be a good idea?

Combiner Example



- The reducer receives pairs (w,n) and sums all the n's
- You can also incorporate the local aggregation in the map function

Word Count: Improved Mapper

```
        Mapper (doc id a, doc d)
1         H ← New Associative Array (dictionary)
2         for all term t ∈ d do
3               H{t} ← H{t}+ 1
4         for all term t ∈ H do
5               emit (t, H(t))
```

Make use of an associative array (or dictonary type) which is an abstract data type that is composed of a collection of (key,value) pairs such that a key only appears once

Hadoop Ecosystem

Other available tools

Other Tools

- Hive
  - SQL Database on Hadoop
- HBase
  - NoSQL Database on Hadoop
- Flume
  - Log processing and storage on Hadoop
- And more…

Summary

- MapReduce on Hadoop
  - Optimization – mappers, reducers, Combiners
- Other Hadoop Tools
  - Hive – SQL
  - HBase – NoSQL
  - Flume – Log Processing