PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

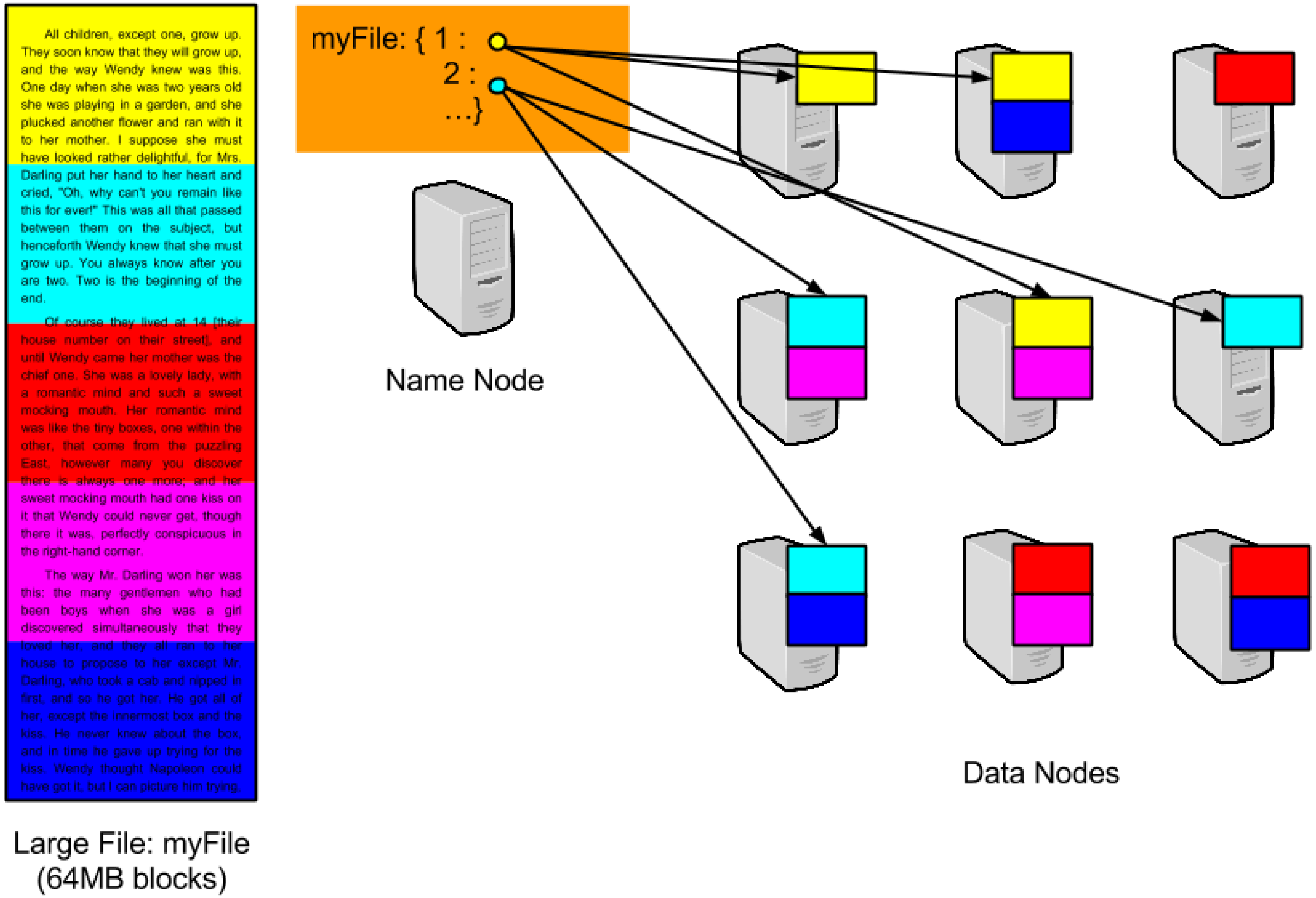
Hadoop

Amy Krause, Andreas Vroutsis

EPCC, The University of Edinburgh

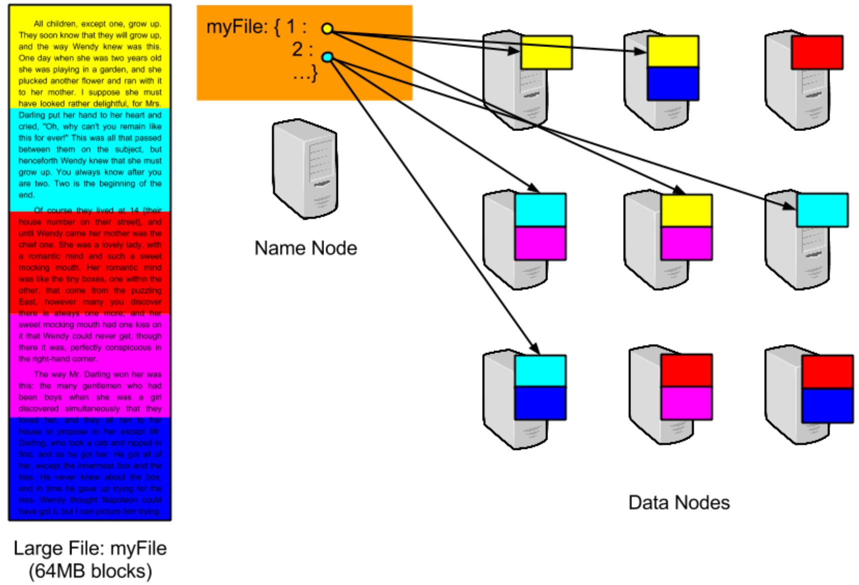
Slides thanks to Ally Hume, EPCC

Hadoop Distributed File System

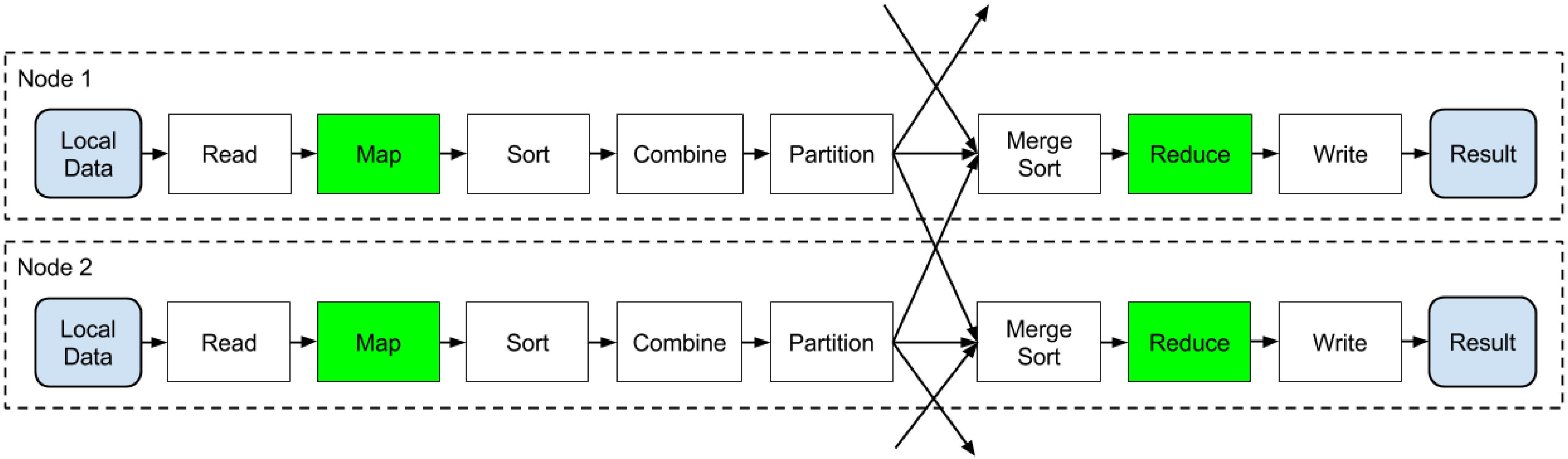


Hadoop Distributed File System

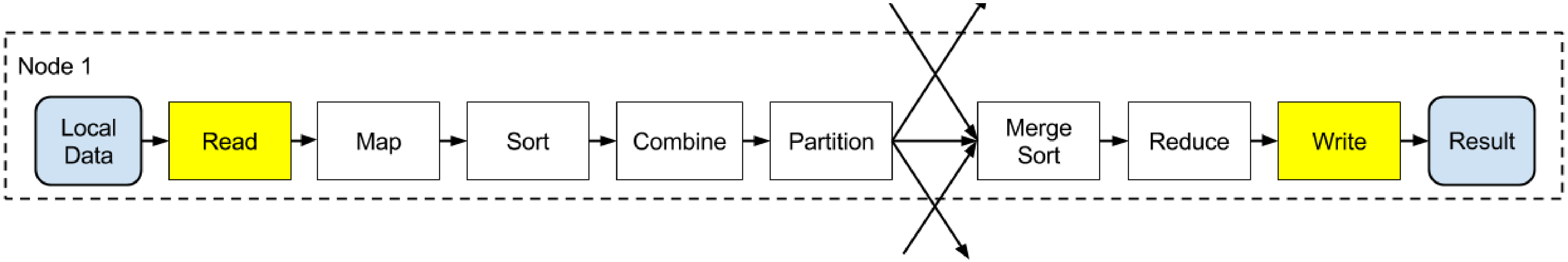
- Typical use: write once, read many
  - Computation runs on Data Nodes
- Distributed
- Data redundancy
- Cluster of commodity nodes
- Designed to withstand failure
  - But Name Node is a single point of failure (see secondary name node)
- Not a POSIX file system
- Placement strategies can be aware of data centre configuration



Hadoop Framework

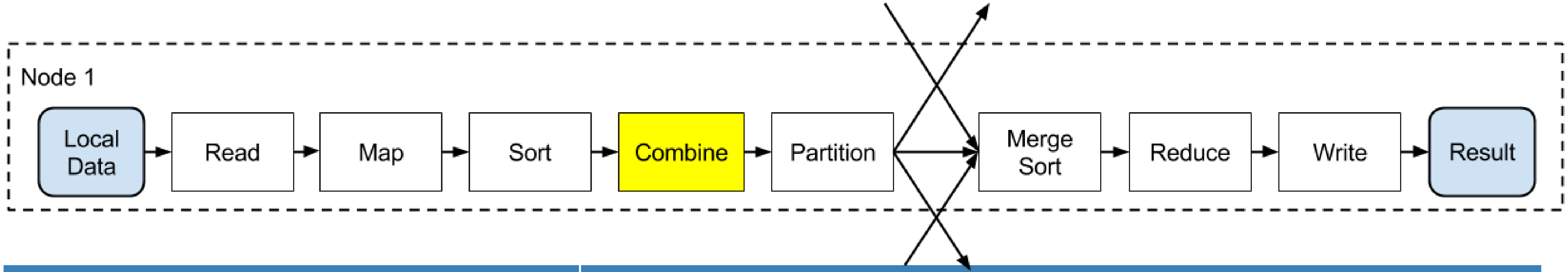


Reading and writing the data



- InputFormat interface
  - TextInputFormat (key: byte offset of line, value: line text)
  - KeyValueTextInputFormat (each line has key/separator/value)
  - SequenceFileInputFormat (Hadoop's compressed binary format)
  - NLineInputFormat (like TextInputFormat but multi-line)
- OutputFormat interface
  - TextOutputFormat (one record per line, key/separator/value)
  - SequenceFileOutputFormat (compressed binary)
  - Filename is "part-*xxxx*" where *xxxx* is the partition ID

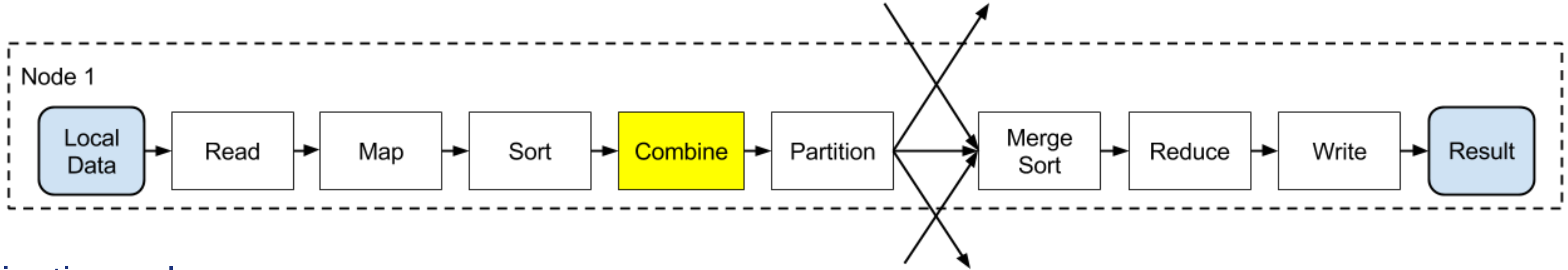
Optimising with a combiner



Map Input	Map Output
<0 : "A boy drove a car">	[<a,1>, <boy,1>, <drove,1>, <a,1>, <car,1>]
<1 : "A car drove at a bus">	[<a,1>, <car,1>, <drove,1>, <at,1>, <a,1>, <bus,1>]
<2 : "Can a boy drive a car?">	[<can,1>, <a,1>, <boy,1>, <drive,1>, <a,1>, <car,1>]
<3 : "A danger – a banana!">	[<a,1>, <danger,1>, <a,1>, <banana,1>]

Combiner Input	Combiner output
<a,[1,1]>	<a, [2]>
<boy, [1,1]>	<boy, [2]>
<car,[1,1,1]>	<car, [3]>
<drove,[1,1]>	<drove, [2]>

Combiner properties

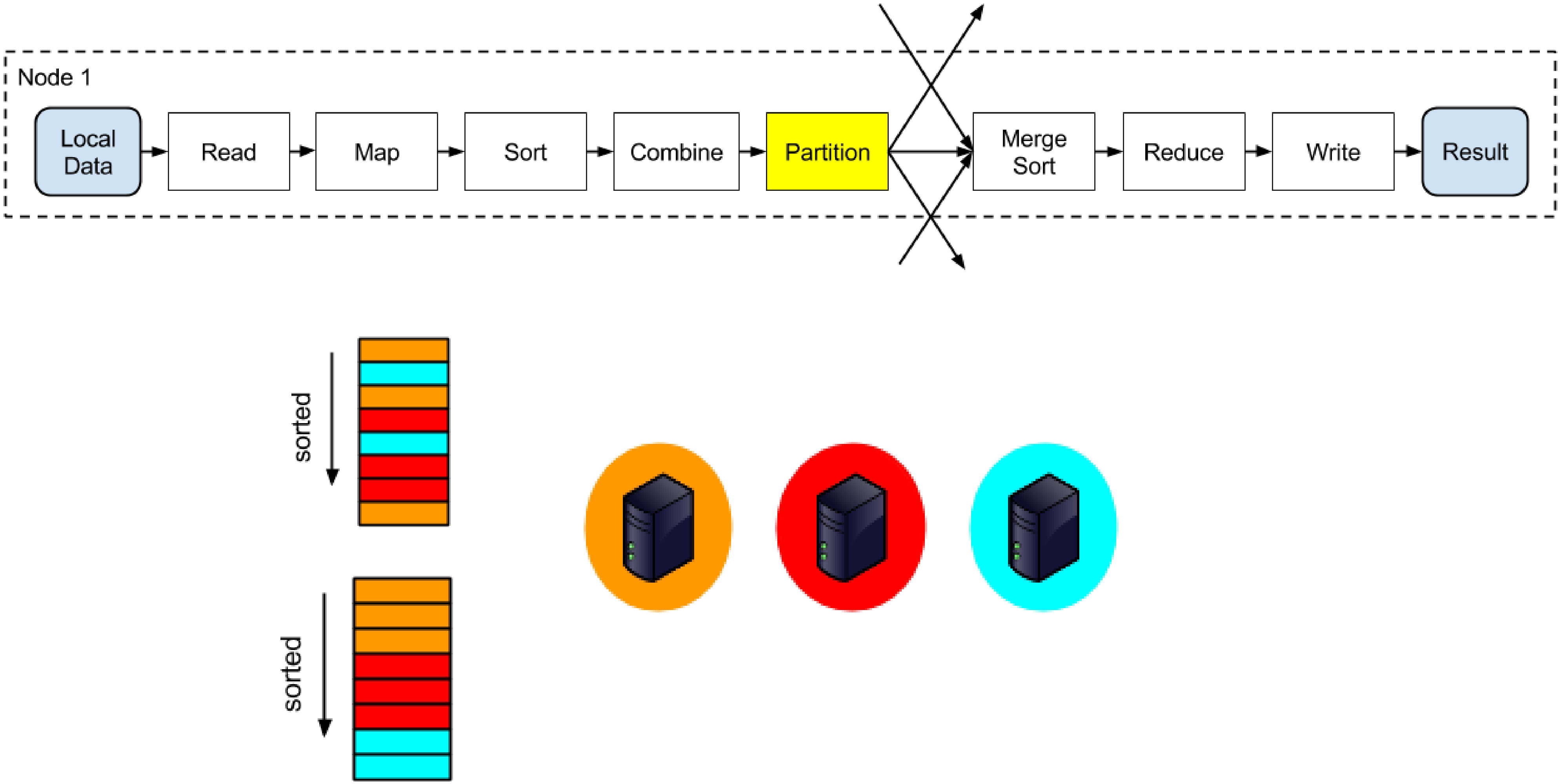


- Optimisation only
  - Framework may execute zero, one or more times
  - Must not alter the final result
  - A helper to the reducer
- Keys *must* not be altered
  - Hadoop does not re-sort after the Combine stage

	Input	Output
Map	<Key1 : Value1>	List(<Key2 : Value2>)
Combine	<Key2 : List(Value2) >	<Key2 : List(Value2) >
Reduce	<Key2 : List(Value2) >	List(<Key3 : Value3>)

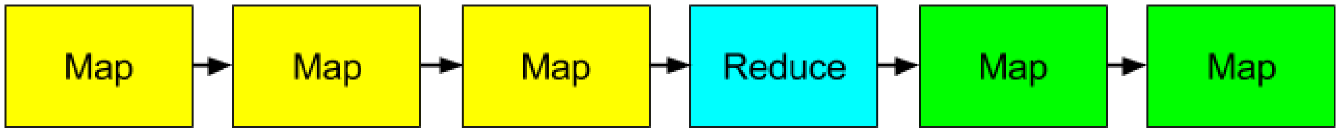
Partitioner

- Hash Partitioner
  - Default
- Total Order Partitioner
  - Maintains order
  - Configure to partition evenly
- Bespoke
  - For highly skewed data hash partitioner may not partition work evenly
  - Maybe some keys require more processing by Reducer


Node 1
Local Data
Read
Map
Sort
Combine
Partition
Merge Sort
Reduce
Write
Result
sorted
sorted

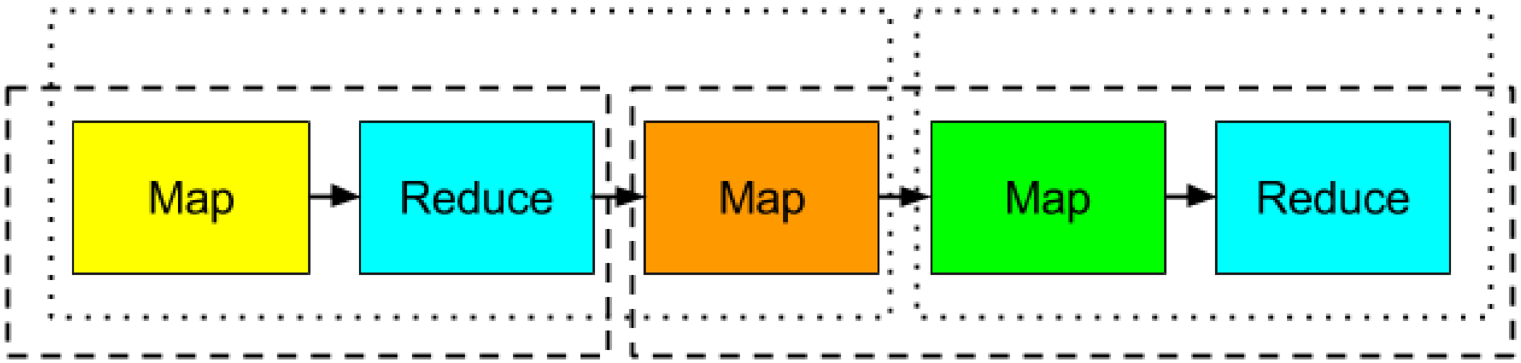
Chaining MapReduce Jobs

- A single map reduce job has
  - One REDUCE stage
  - One or more MAP stages before the reduce
  - Zero or more MAP stages after the reduce

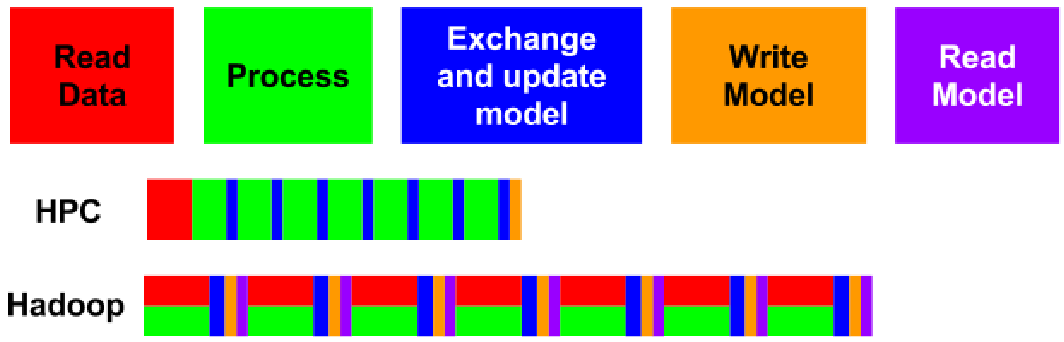


Chaining MapReduce Jobs

- Need to chain multiple map reduce jobs when:
  - There is more than one REDUCE stage (grouping of data by key)
  - MAP stages between REDUCE jobs could be part of either job



Chain, but don’t iterate



- Each Hadoop job reads data from the HDFS and writes output to the HDFS
  - No data is maintained in memory between jobs
- Fine for short chains of processing
- Very inefficient for iterative algorithms
  - Data (even static data) must be read from disk at each iteration

- Spark – supports caching data
- Twister – iterative map reduce

Programming Hadoop

- Hadoop framework is written in Java
- Two models for writing Map, Reduce and Combine functions
  - Java classes
  - Hadoop streaming
    - Functions are scripts that read from standard input and write to standard output
- If writing your own partitioners or getting into the internals of Hadoop you will need to use Java
  - But for most problems you do not need to do this.

Map class in Java

Must implement function:
void map(InputKeyType, InputValueType, Context)

Mapper<InputKeyType, InputValueType, OutputKeyType, OutputValueType >

```
public static class MapClass
  extends Mapper<Text, Text, Text, Text>
{
    public void map(Text key, Text value, Context context)
    {
        context.write(value, key);
    }
}
```

This mapper simply swaps the key and value

write output data using context.write(outputKey, outputValue)

Can call multiple times and hence output List(<OutputKeyType, OutputValueType>)

Reduce class in Java

Reducer<
InputKeyType, InputValueType,
OutputKeyType, OutputValueType >

```
public static class Reduce
   extends Reducer<Text, Text, Text, Text>
{
  public void reduce( Text key,
                      Iterable<Text> values,
                      Context context)
  {
    String csv = "";
    for (Text val:values)
    {
      if (csv.length() > 0) csv += ",";
      csv += val.toString();
    }
    context.write(key, new Text(csv));
  }
}
```

Uses iterator to get list of values – can thus support large lists with low memory footprint. So long as the rest of the method is similarly low memory. This example is not!

write output data using context.write(outputKey, outputValue)
Can call multiple times if desired

Streaming Mapper

```
1<TAB>A long time ago
2<TAB>in a galaxy far
3<TAB>far away
```

→

```
a<TAB>1
long<TAB>1
time<TAB>1
ago<TAB>1
in<TAB>1
a<TAB>1
galaxy<TAB>1
...
```

- Input: rows of key/value pairs separated by TAB character
- Output: rows of key/value pairs separated by TAB character
- **Stateless**
  - Process one line at a time with no state maintained between lines.

Streaming Reducer

- Input is rows of key/value pairs separated by TAB character
- Input guarantees that all the key/value pairs associated with a specific key will be contiguous in the input stream
  - When key changes you know you have seen all the values associated with that key
- Output rows of key/value pairs separated by TAB character
- **Stateless**
  - Can maintain state while processing rows with the same key.
  - Must not maintain state across rows with different keys

Streaming Reducer

```
a<TAB>1
a<TAB>1
a<TAB>1
far<TAB>1
far<TAB>1
time<TAB>1
```

→

```
a<TAB>3
far<TAB>2
time<TAB>1
```

Hadoop vs MPI/HPC

- Fault tolerance
  - Hadoop is designed specifically with fault tolerance in mind
  - MPI provides little support for fault tolerance and most MPI programs assume the system hardware will not fail
- Specific vs general
  - Hadoop is a framework for a specific data processing pattern
  - MPI allows you to code any algorithm you wish
- Iterative algorithms
  - Hadoop very poor at multiple iterations over the data
  - Very easy to write such programs in MPI
- Speed
  - If you have a reliable HPC system an optimised MPI implementation should perform considerably better than a Hadoop solution

Hadoop vs MPI/HPC cont.

- Hadoop good when data written once – processed often
- Hadoop uses key value pair structures that can be fragmented in memory leading to poor cache efficiency
- Trade off between simple, highly scalable on commodity hardware against highly optimised implementation on very expensive hardware

Hadoop vs MPI/HPC cont.

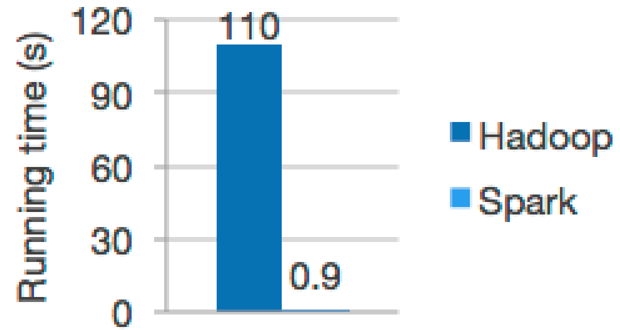
- Cost
  - Hadoop simple to write and can run reliably on commodity hardware.
  - MPI typically run on expensive HPC systems
    - MPI can run on clouds but have to build your own fault tolerance.
- Dynamic nature of data
  - Hadoop is good for processing massive amounts of data that is written once and processed often
  - HPC systems may not scale well to such massive datasets being uploaded.

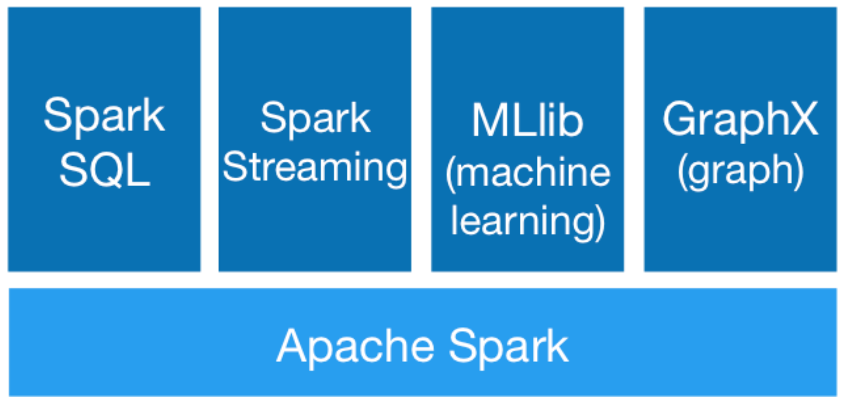
Hadoop Ecosystem

- HBASE
  - Distributed, scalable big data store
  - Columnar database
- PIG
  - Higher level data flow language for programming Hadoop
- Mahout
  - Scalable machine learning and data mining over Hadoop
- Spark
  - Machine learning algorithms

A little more on Spark

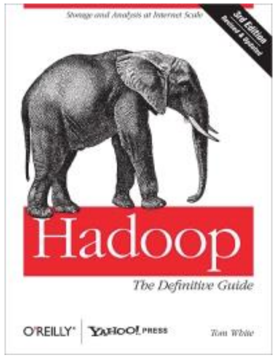
- Explicitly supports caching data
  - Speeds up iterative algorithms
- Can use HDFS as the data source
- More that just map/reduce
  - Transformations:
    - map, filter, union, Cartesian, join, sample…
  - Actions:
    - reduce, collect, count, first, countBy, foreach…

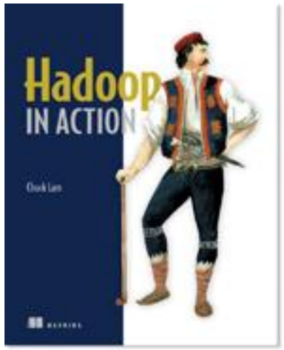




Additional reading

- Google File System
  - http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf
- Map Reduce
  - http://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf





- Examples taken from Hadoop in Action
  - http://www.manning.com/lam/
- For Hadoop 3 I like O’Reilly’s Hadoop, The Definitive Guide
- Plenty online – course does not require material outside of the lectures and practical.