

Data Analytics with HPC

Hadoop 2

EPSRC

NERC SCIENCE OF THE ENVIRONMENT



CRAY
THE SUPERCOMPUTER COMPANY

epcc



Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

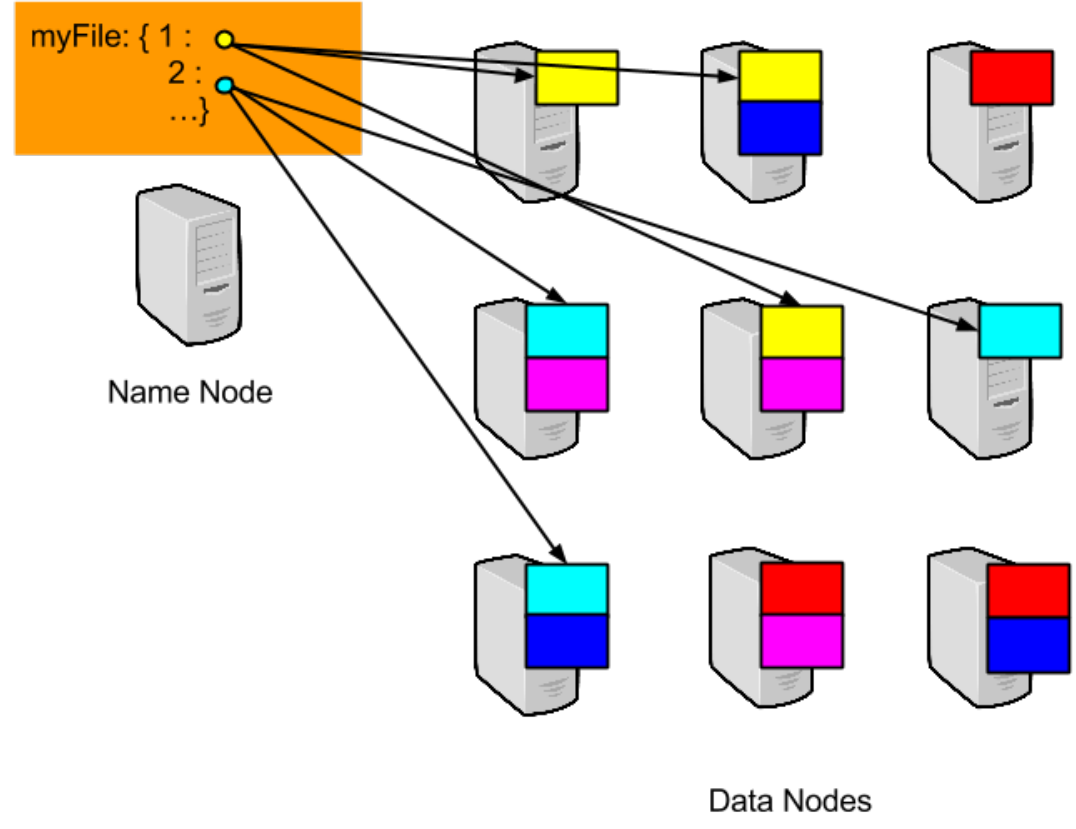


Hadoop Distributed File System

All children, except one, grow up. They soon know that they will grow up, and the way Wendy knew was this. One day when she was two years old she was playing in a garden, and she plucked another flower and ran with it to her mother. I suppose she must have looked rather delightful, for Mrs. Darling put her hand to her heart and cried, "Oh, why can't you remain like this for ever!" This was all that passed between them on the subject, but henceforth Wendy knew that she must grow up. You always know after you are two. Two is the beginning of the end.

Of course they lived at 14 (their house number on their street, and until Wendy came her mother was the chief one. She was a lovely lady, with a romantic mind and such a sweet mocking mouth. Her romantic mind was like the tiny boxes, one within the other, that come from the puzzling East, however many you discover there is always one more, and her sweet mocking mouth had one kiss on it that Wendy could never get, though there it was, perfectly conspicuous in the right-hand corner.

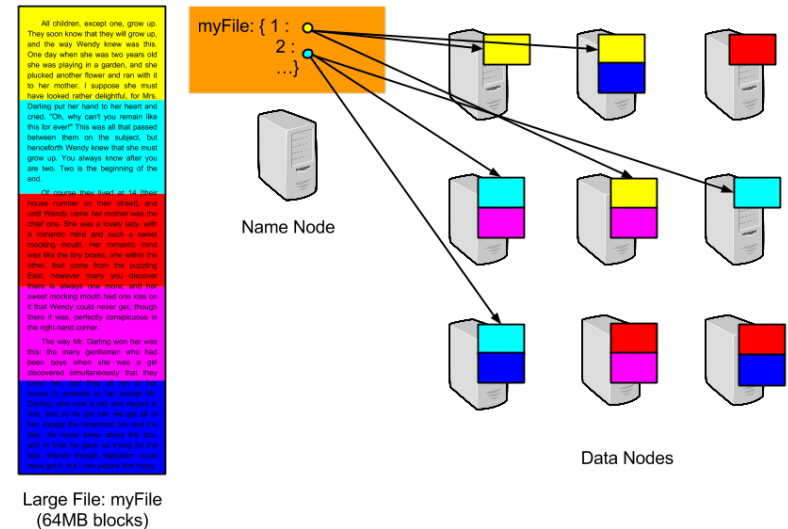
The way Mr. Darling won her was this: the many gentlemen who had been boys when she was a girl discovered simultaneously that they loved her, and they all ran to her house to propose to her except Mr. Darling, who took a cab and ripped in first, and so he got her. He got all of her, except the innermost box and the kiss. He never knew about the box, and in time he gave up trying for the kiss. Wendy thought Napoleon could have got it, but I can picture him trying.

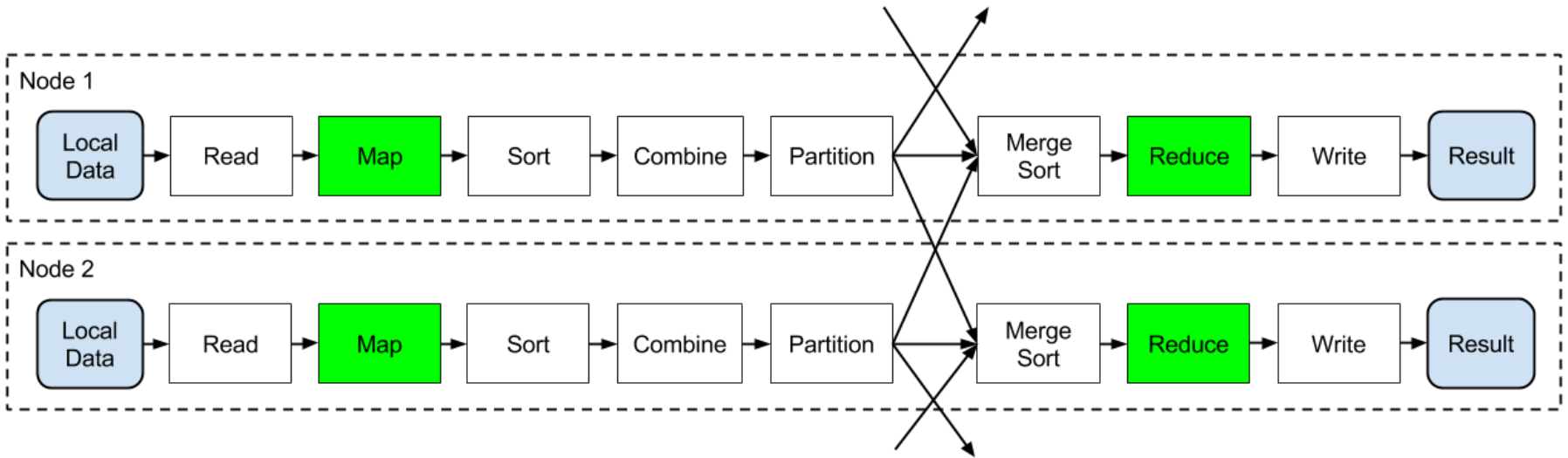


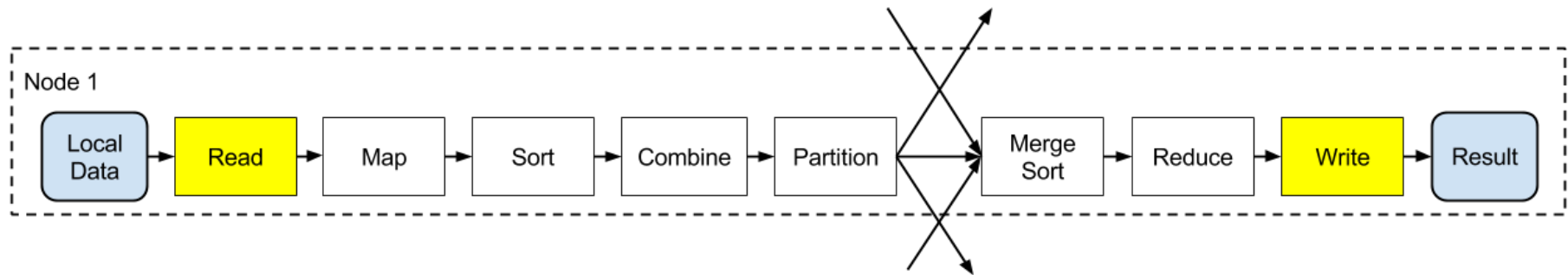
Large File: myFile
(64MB blocks)

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)
- Optimised for the tasks in hand
 - Not a POSIX file system
- Placement strategies can be aware of data centre configuration







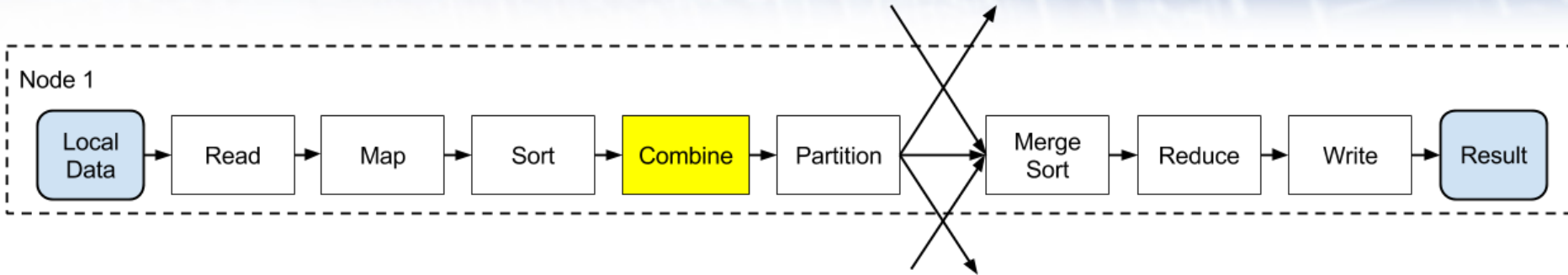
- **InputFormat interface**

- 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

<0 : "A boy drove a car">

<1 : "A car drove at a bus">

<2 : "Can a boy drive a car?">

<3 : "A danger – a banana!">

Map Output

[<a,1>, <boy,1>, <drove,1>, <a,1>, <car,1>]

[<a,1>, <car,1>, <drove,1>, <at,1>, <a,1>, <bus,1>]

[<can,1>, <a,1>, <boy,1>, <drive,1>, <a,1>, <car,1>]

[<a,1>, <danger,1>, <a,1>, <banana,1>]

Combiner Input

<a,[1,1]>

<boy, [1,1]>

<car,[1,1,1]>

<drove,[1,1]>

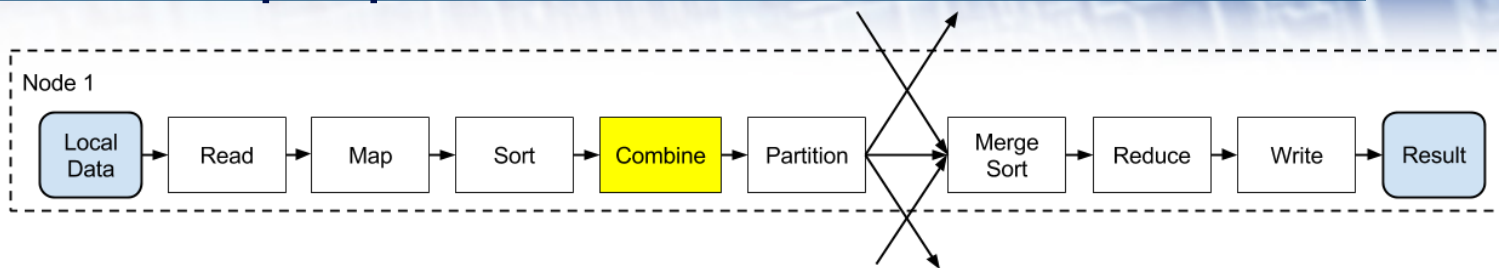
Combiner output

<a, [2]>

<boy, [2]>

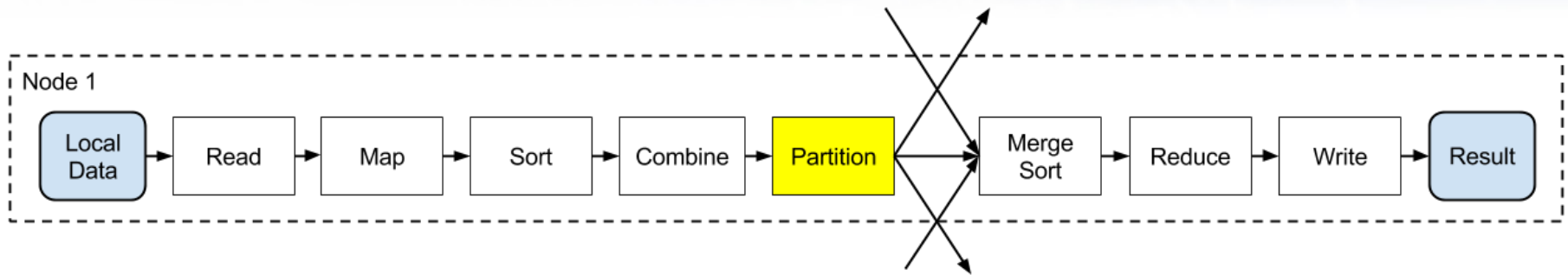
<car, [3]>

<drove, [2]>



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

- 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



- Hash Partitioner

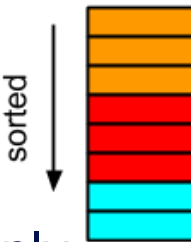
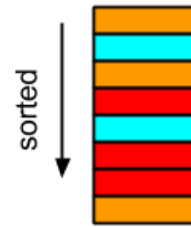
- Default

- Total Order Partitioner

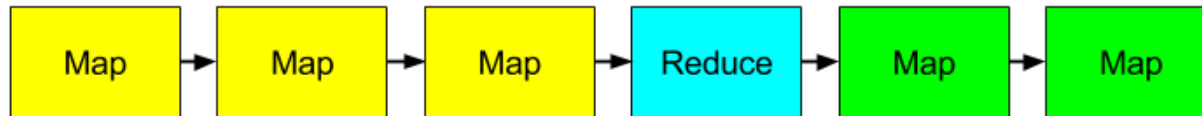
- Maintains order
- Configure to partition evenly

- Bespoke

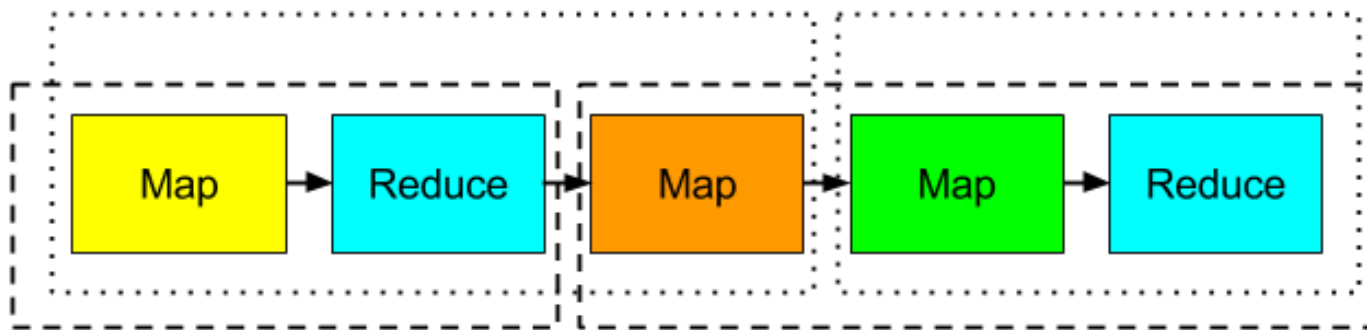
- For highly skewed data hash partitioner may not partition work evenly
- Maybe some keys require more processing by Reducer



- 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



- 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



- 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



- 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.

**Mapper<
InputKeyType, InputValueType,
OutputKeyType, OutputValueType >**

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

```
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>)**

```
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));
    }
}
```

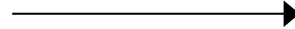
**Reducer<
InputKeyType, InputValueType,
OutputKeyType, OutputValueType >**

**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**

- 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.

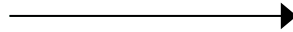
```
1<TAB>A long time ago  
2<TAB>in a galaxy far  
3<TAB>far away
```



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

- Input 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


```
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
```

- 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 a Hadoop solution

- 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.

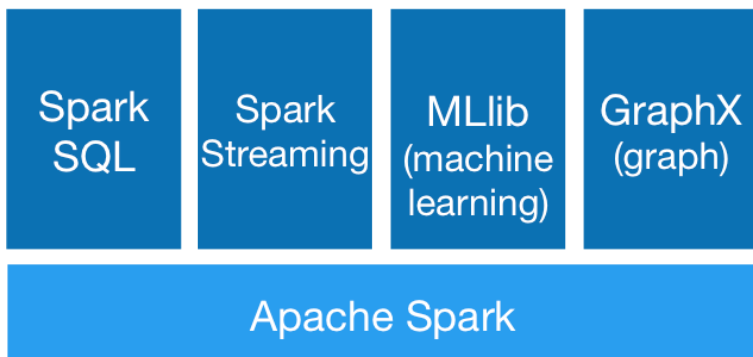
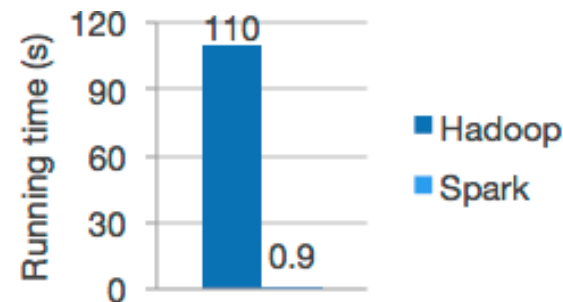
- HBASE
 - Distributed, scalable big data store
 - Columnar database
- PIG
 - Higher level data flow language for programming Hadoop
- Mahout
 - Scalable machine learning and data mining over Hadoop
- Spark
 - Machine learning algorithms

A P A C H E
HBASE



Spark

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



- 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, O'Reilly's *Hadoop, The Definitive Guide* is good.
- Plenty online

