Inspire…Educate…Transform.

Engineering Big Data

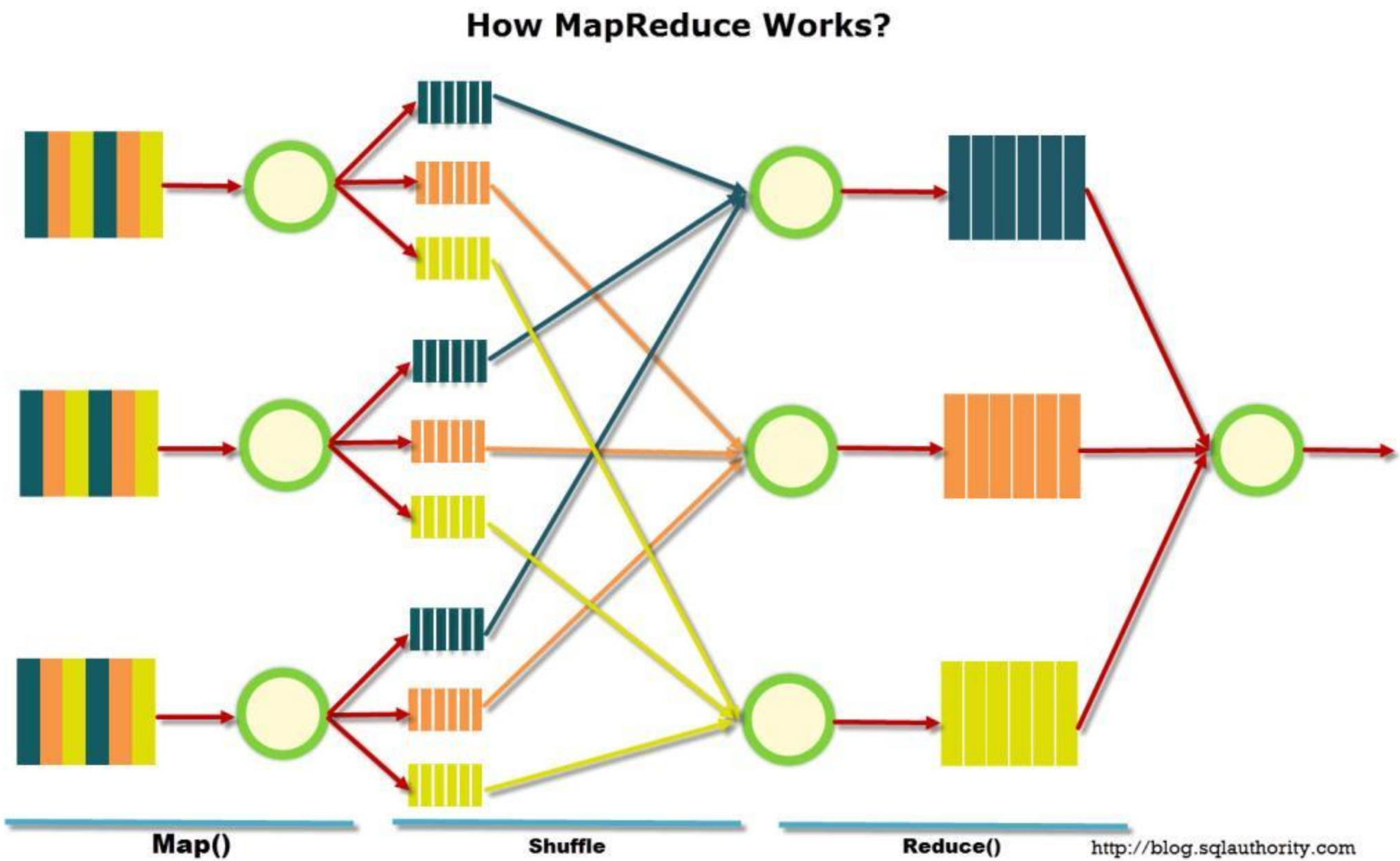
MR, Streaming, HDFS2

Dr. Sreerama KV Murthy
CEO, Quadratyx

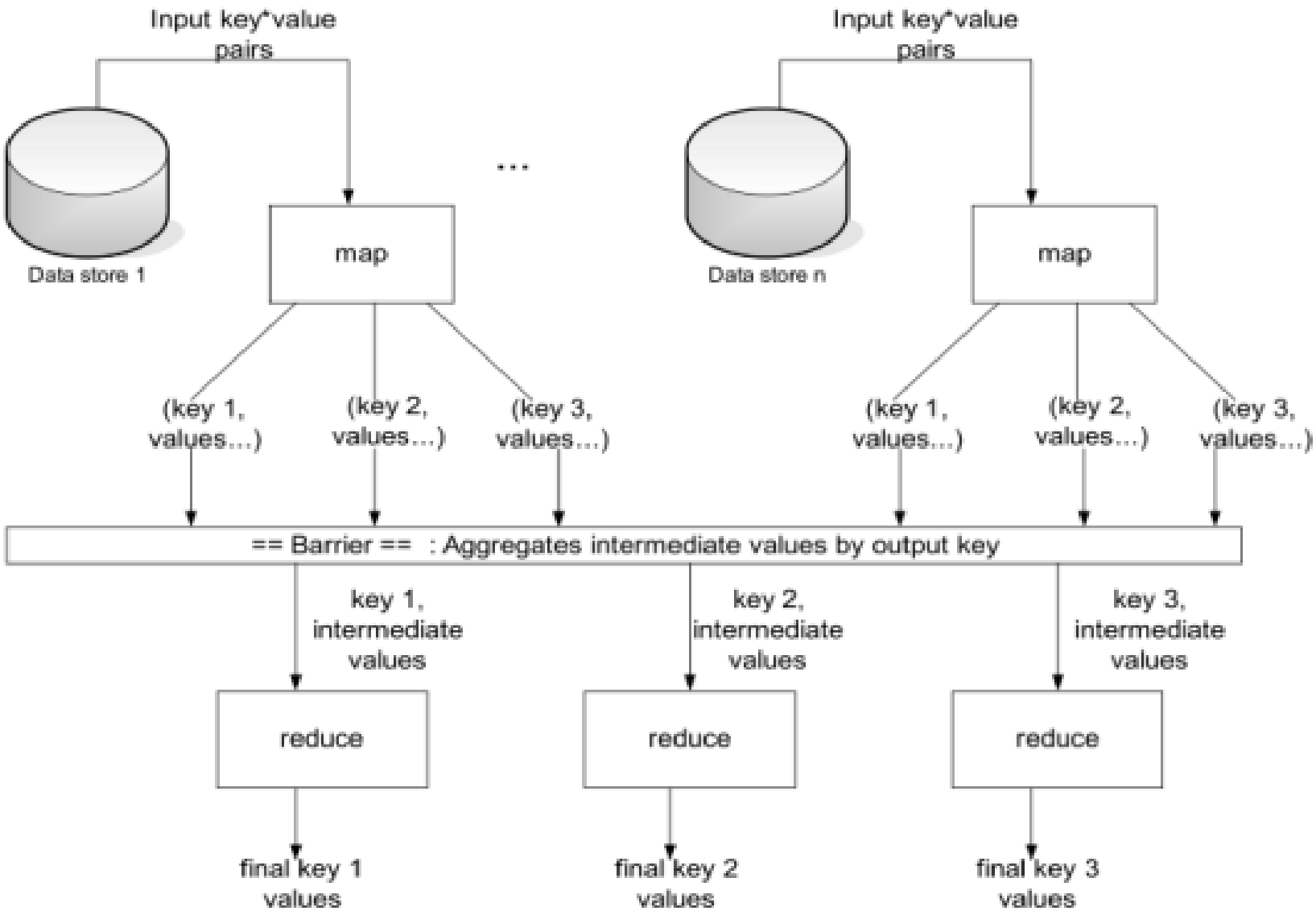
August 22, 2015

Wake-Up Quiz

RECAP


How MapReduce Works?
Map()
Shuffle
Reduce()
http://blog.sqlauthority.com

MapReduce – In more detail



Map Reduce: Keys and Values

- Programmers specify two functions:

  map (k, v) → <k′, v′>*

  reduce (k′, v′) → <k′, v′>*

  – All values with the same key are reduced together

- **Keys and values in Hadoop are `Objects`**
- **Values are objects which implement `Writable`**
- **Keys are objects which implement `WritableComparable`**

Mappers

- Mappers run on nodes which hold their portion of the data locally, to avoid network traffic
- Multiple Mappers run in parallel, each processing a portion of the input data
- Mapper reads data in the form of key/value pairs
  - Mapper may use, or completely ignore, the input key.
  - E.g., a standard pattern is to read a line of a file at a time. Key then is the byte offset into the file at which the line starts. Value is the contents of the line itself. Typically the key is considered irrelevant .
- It outputs zero or more key/value pairs
  - let map(k, v) = emit(k.toUpper(), v.toUpper())
  - ('foo', 'bar') -> ('FOO', 'BAR')

Explode mapper

- Output each input character separately (pseudo-code):

```
let map(k, v) =
    foreach char c in v:
        emit (k, c)
```

```
('foo', 'bar') ->    ('foo', 'b'), ('foo', 'a'),
                     ('foo', 'r')

('baz', 'other') -> ('baz', 'o'), ('baz', 't'),
                     ('baz', 'h'), ('baz', 'e'),
                     ('baz', 'r')
```

Filter mapper

- Only output key/value pairs where the input value is a prime number (pseudo-code):

```
let map(k, v) =
    if (isPrime(v)) then emit(k, v)
```

```
('foo', 7) ->    ('foo', 7)

('baz', 10) ->   nothing
```

Changing Key Spaces Mapper

- Output the word length as the key (pseudo-code):

```
let map(k, v) =
    emit(v.length(), v)
```

```
('foo', 'bar') ->    (3, 'bar')
('baz', 'other') -> (5, 'other')
('foo', 'abracadabra') -> (11, 'abracadabra')
```

Reducer

- **After the Map phase is over, all the intermediate values for a given intermediate key are combined together into a list**
- **This list is given to a Reducer**
  - There may be a single Reducer, or multiple Reducers
    - This is specified as part of the job configuration (see later)
  - All values associated with a particular intermediate key are guaranteed to go to the same Reducer
  - The intermediate keys, and their value lists, are passed to the Reducer in sorted key order
  - This step is known as the 'shuffle and sort'
- **The Reducer outputs zero or more final key/value pairs**
  - These are written to HDFS
  - In practice, the Reducer usually emits a single key/value pair for each input key

Sum Reducer

- Add up all the values associated with each intermediate key (pseudo-code):

```
let reduce(k, vals) =
    sum = 0
    foreach int i in vals:
        sum += i
    emit(k, sum)
```

```
('bar', [9, 3, -17, 44]) ->   ('bar', 39)
('foo', [123, 100, 77]) -> ('foo', 300)
```

More Reducers

- Identity reducer
- No reducer
- Explode reducer?

Map Reduce: Simple Examples

- Distributed "grep" (pattern search)
  - —map: emits a line if it matches a supplied pattern
  - —reduce: identity function - copies intermediate data to the output
- Count of URL access frequency
  - —map: processes logs of web page requests, outputs a sequence of <URL, 1> tuples
  - —reduce: adds together all values for the same URL and emits a <URL, total count> pair
- Reverse web-link graph
  - —map: outputs <target, source> pairs for each link to a target URL found in a page named source
  - —reduce: concatenates the list of all source URLs associated with a given target URL
    - emits the pair: <target, list of sources>

Partition & Combine

- Optionally:

  partition (k', number of partitions) → partition for k'

  – Often a simple hash of the key, e.g., hash(k') mod n

  – Divides up key space for parallel reduce operations

  combine (k', v') → <k', v'>*

  – **Mini-reducers** that run in memory after the map phase

  – Used as an optimization to reduce network traffic

Word Count: Baseline

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)

1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count s)
```

What's the impact of combiners?

Word Count: Version 1

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1                ▷ Tally counts for entire document
6:         for all term t ∈ H do
7:             Emit(term t, count H{t})
```

Are combiners still needed?

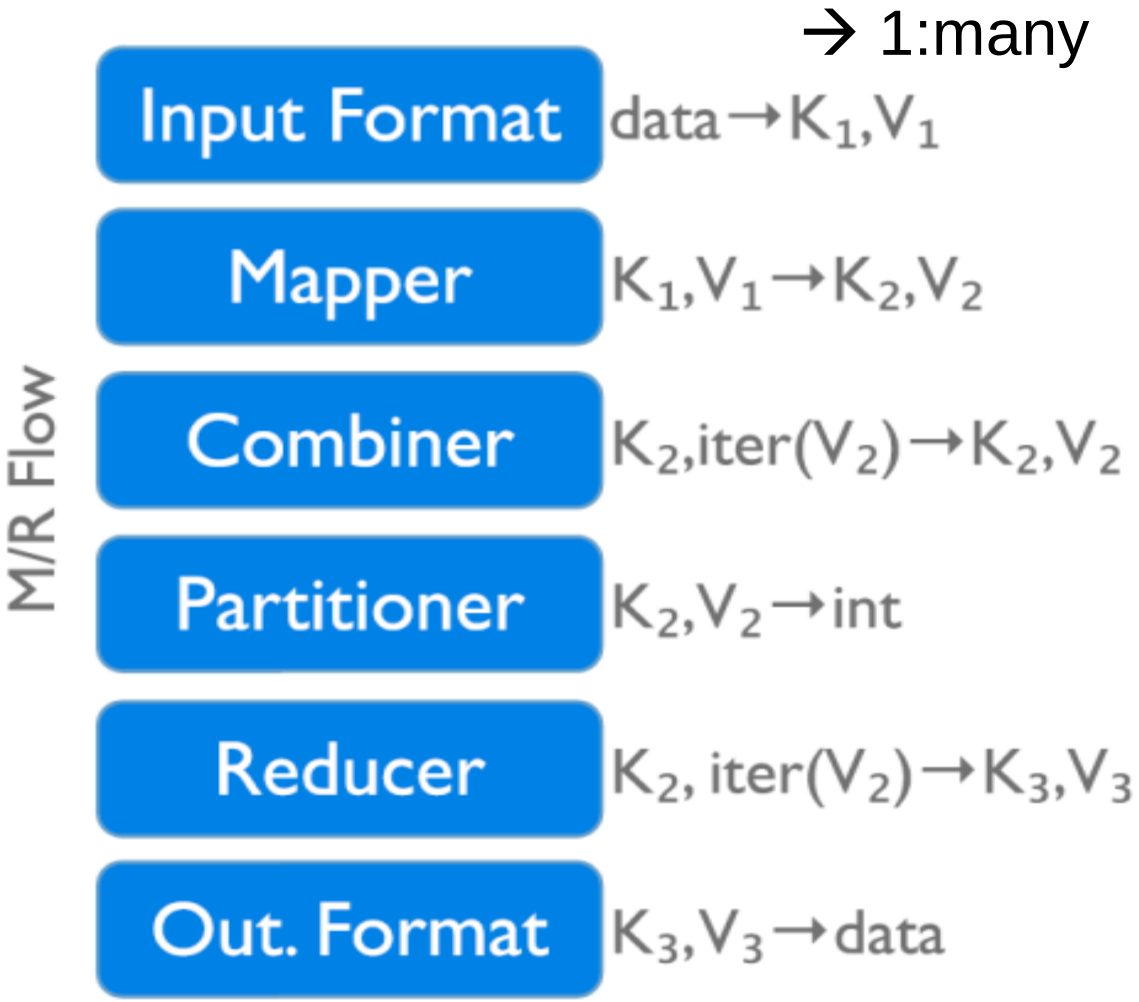
Word Count: Version 2

```
1: class Mapper
2:     method Initialize
3:         H ← new AssociativeArray
4:     method Map(docid a, doc d)
5:         for all term t ∈ doc d do
6:             H{t} ← H{t} + 1                ▷ Tally counts across documents
7:     method Close
8:         for all term t ∈ H do
9:             Emit(term t, count H{t})
```

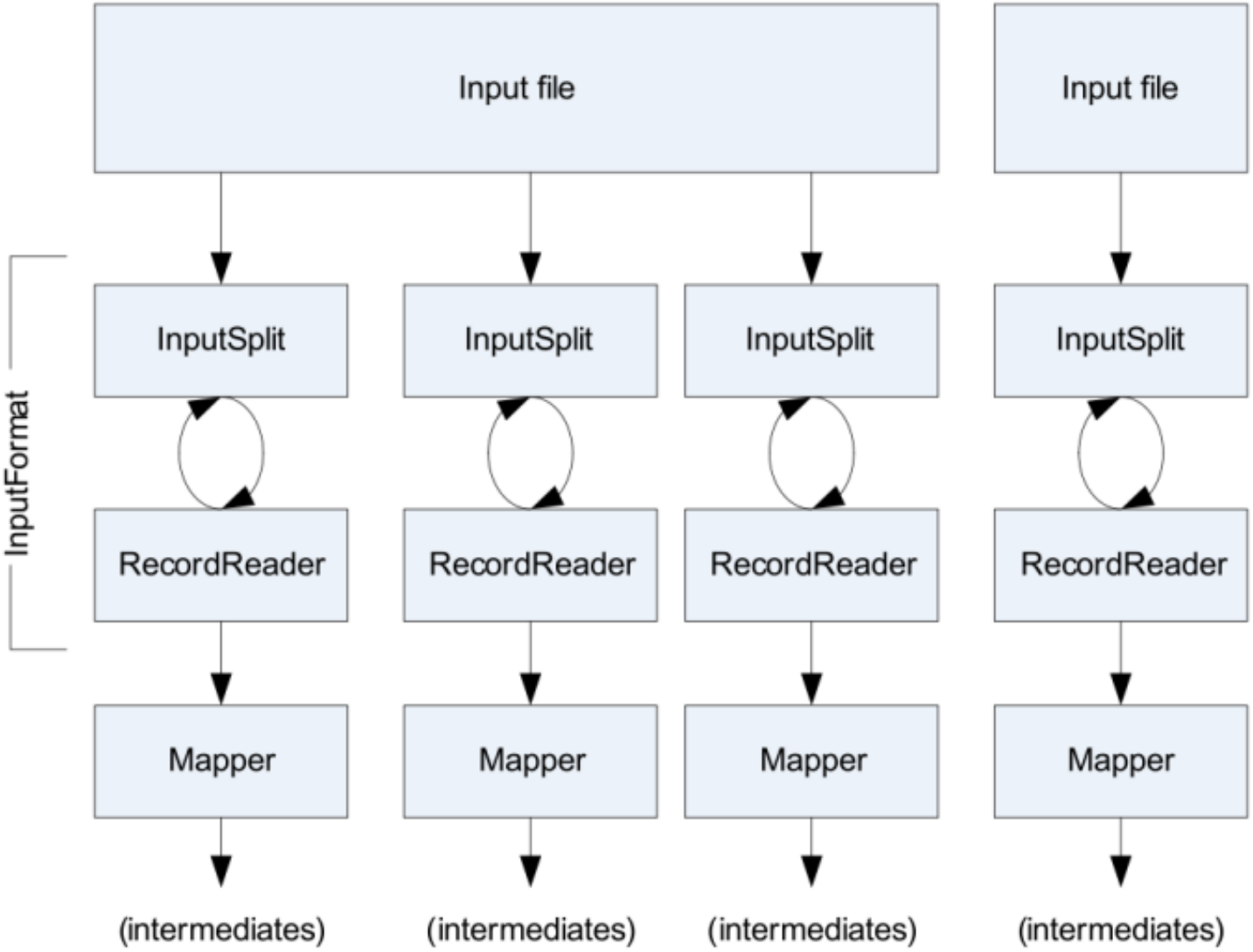
Key: preserve state across input key-value pairs!

Are combiners still needed?

Data Flow in a MapReduce Program



Role of InputFormat



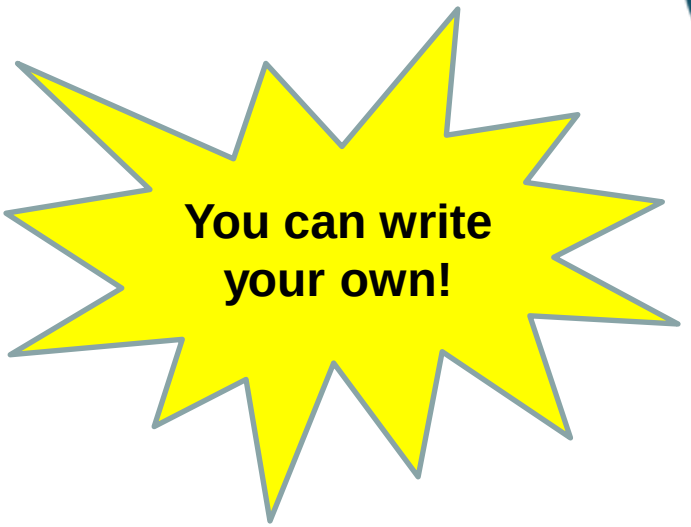
How InputFormats work

- **All file-based `InputFormats` inherit from `FileInputFormat`**
- **`FileInputFormat` computes `InputSplits` based on the size of each file, in bytes**
  - HDFS block size is used as upper bound for `InputSplit` size
  - Lower bound can be specified in your driver code
- **Important: `InputSplits` do not respect record boundaries!**
- **`InputSplits` are handed to the `RecordReaders`**
  - Specified by the path, starting position offset, length
- **`RecordReaders` must:**
  - Ensure each (key, value) pair is processed
  - Ensure no (key, value) pair is processed more than once
  - Handle (key, value) pairs which are split across `InputSplits`

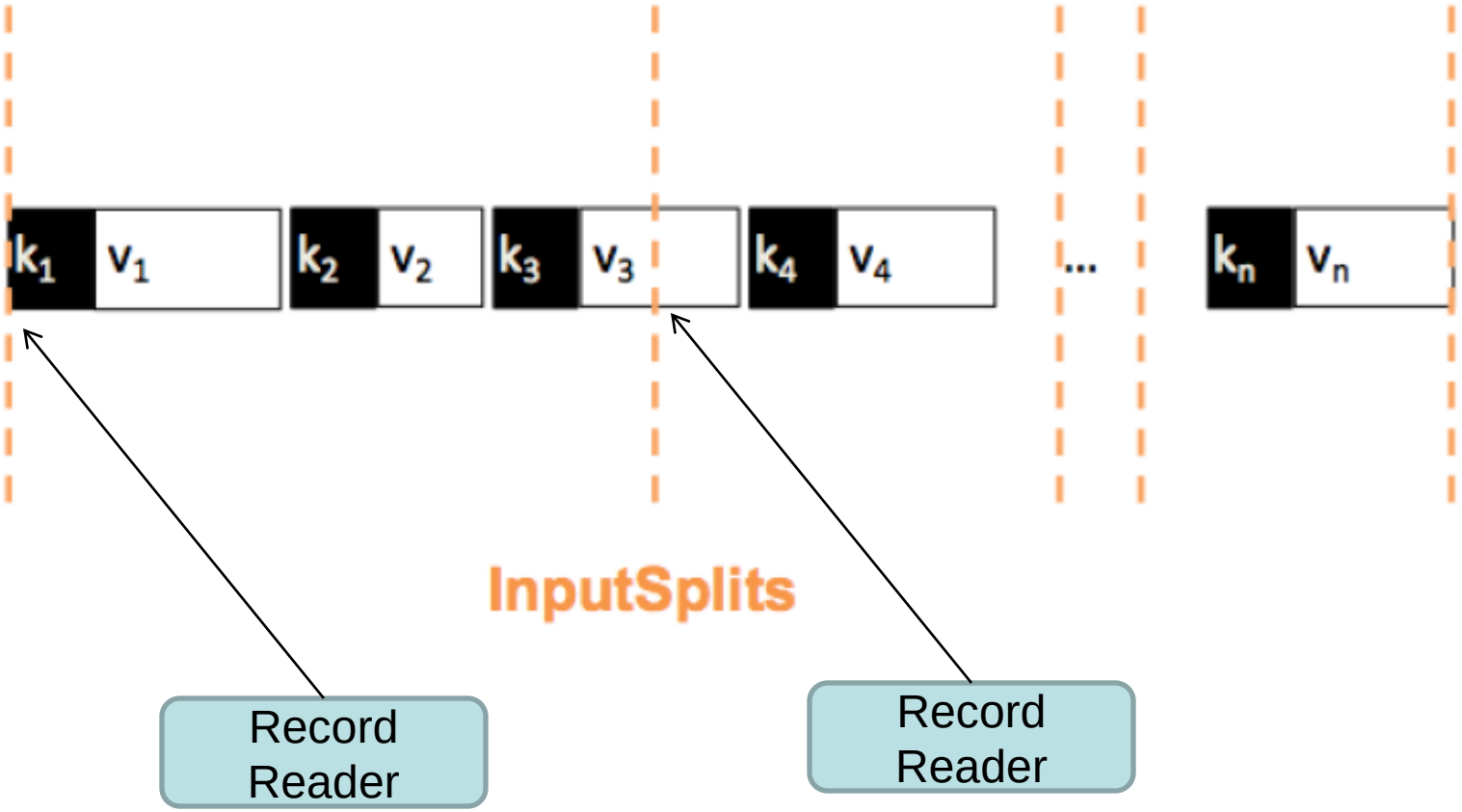
Multiple Formats Available

- **Most common InputFormats:**
  - `TextInputFormat`
  - `KeyValueTextInputFormat`
  - `SequenceFileInputFormat`
- **Others are available**
  - `NLineInputFormat`
    - Every *n* lines of an input file is treated as a separate InputSplit
    - Configure in the driver code with

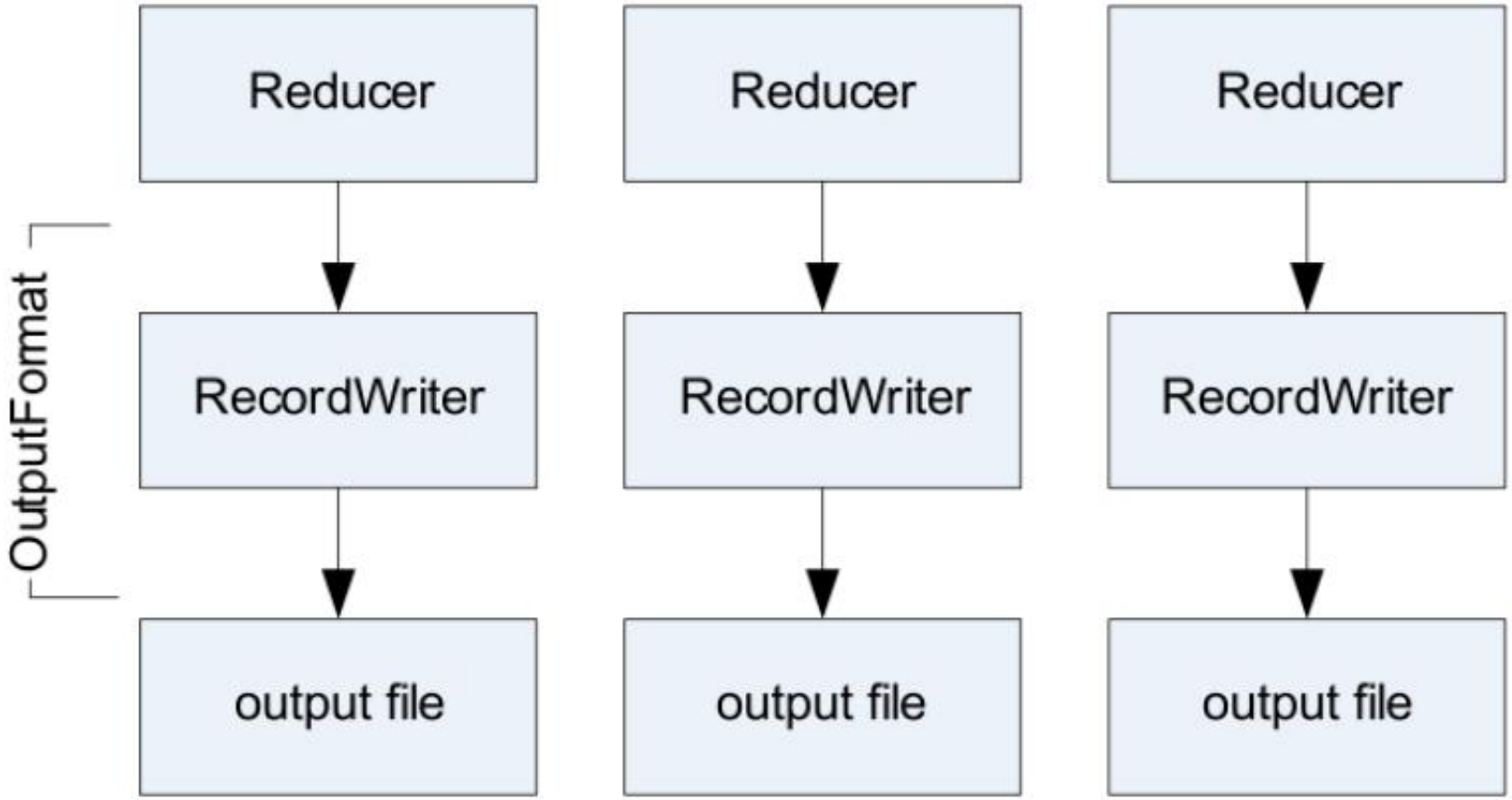
      `mapred.line.input.format.linespermap`
  - `MultiFileInputFormat`
    - Abstract class which manages the use of multiple files in a single task
    - You must supply a `getRecordReader()` implementation



Sample Input Splits



Same with OutputFormats and RecordWriters



The Distributed Cache

- **A common requirement is for a Mapper or Reducer to need access to some 'side data'**
  - Lookup tables
  - Dictionaries
  - Standard configuration values
- **One option: read directly from HDFS in the `configure` method**
  - Works, but is not scalable
- **The DistributedCache provides an API to push data to all slave nodes**
  - Transfer happens behind the scenes before any task is executed
  - Note: DistributedCache is read-only
  - Files in the DistributedCache are automatically deleted from slave nodes when the job finishes

Using the Distributed Cache

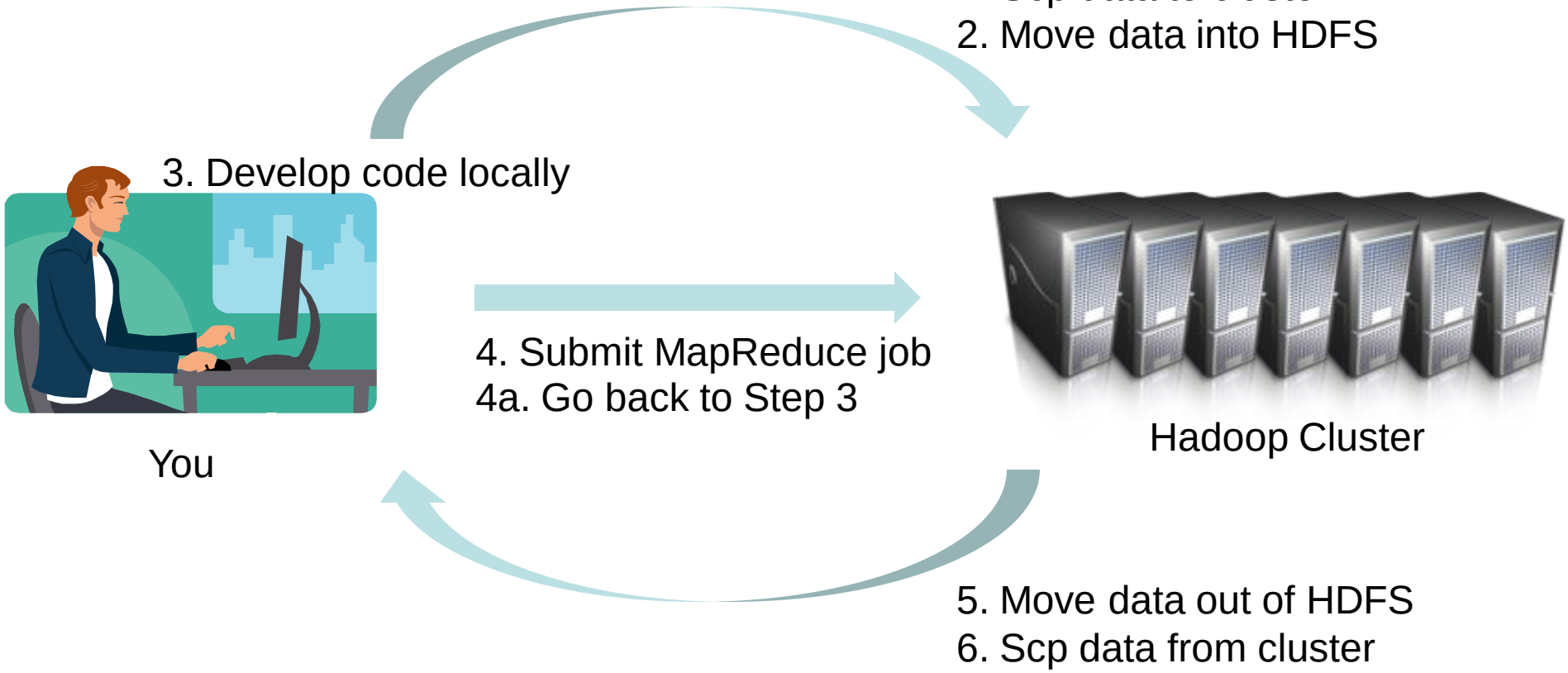
- Use the `-files` option to add files

```
hadoop jar myjar.jar MyDriver -files file1, file2, file3, ...
```

- Files added to the DistributedCache are made available in your task's local working directory
  - Access them from your Mapper or Reducer the way you would read any ordinary local file

```
File f = new File("file_name_here");
```

Map Reduce Development Cycle



Some features of map-reduce jobs

- MapReduce jobs tend to be relatively short in terms of lines of code
- It is typical to combine multiple small MapReduce jobs together in a single workflow
  - Oozie
- You are likely to find that many of your MapReduce jobs use very similar code

Refresher (contd.)

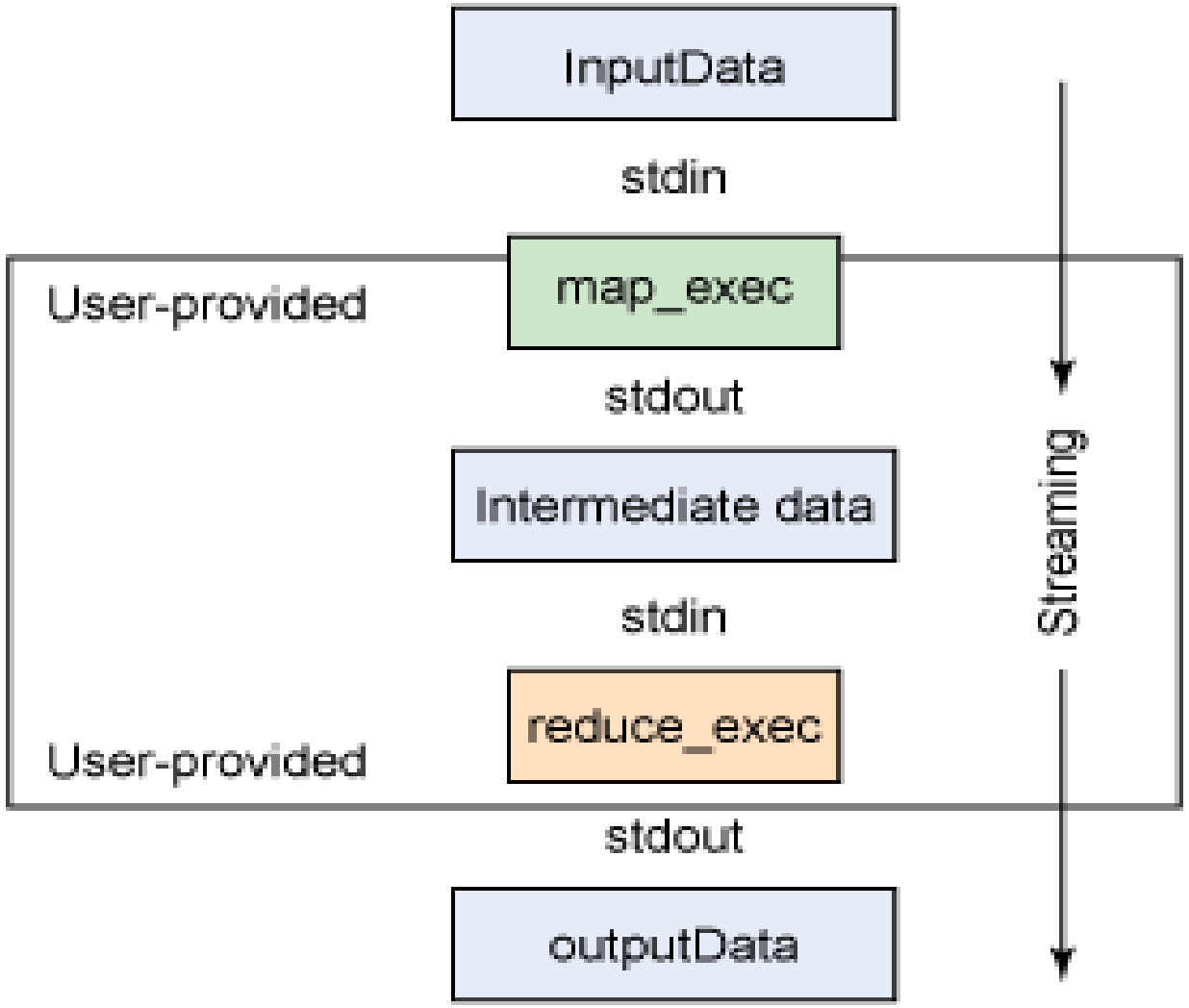
- All algorithms must expressed in m, r, c, p

- The execution framework handles everything else…
  - Scheduling: assigns workers to map and reduce tasks
  - Data distribution: moves processes to data
  - Synchronization: gathers, sorts, and shuffles intermediate data
  - Errors and faults: detects worker failures and restarts

- You don't know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

HADOOP STREAMING

Hadoop Streaming



Hadoop Streaming

- Allows you to create and run map/reduce jobs with any executable
- Similar to unix pipes, e.g.:
  - format is: Input | Mapper | Reducer
  - echo “this sentence has five lines” | cat | wc

Hadoop Streaming

- Mapper and Reducer receive data from stdin and output to stdout
- Hadoop takes care of the transmission of data between the map/reduce tasks
  - It is still the programmer's responsibility to set the correct key/value
  - Default format: "key \t value\n"
- Let's look at a Python example of a MapReduce word count program…

Streaming_Mapper.py

```
# read in one line of input at a time from stdin
for line in sys.stdin:
    line = line.strip()          # string
    words = line.split()         # list of strings

    # write data on stdout
    for word in words:
        print '%s\t%i' % (word, 1)
```

Hadoop Streaming

- What are we outputting?
  - Example output: “the 1”
  - By default, “the” is the key, and “1” is the value
- Hadoop Streaming handles delivering this key/value pair to a Reducer
  - Able to send similar keys to the same Reducer or to an intermediary Combiner

Streaming_Reducer.py

```
wordcount = { }          # empty dictionary
# read in one line of input at a time from stdin
for line in sys.stdin:
    line = line.strip()          # string
    key,value = line.split()
    wordcount[key] = wordcount.get(key, 0) + value

    # write data on stdout
    for word, count in sorted(wordcount.items()):
        print '%s\t%i' % (word, count)
```

Hadoop Streaming Gotcha

- Streaming Reducer receives single lines (which are key/value pairs) from stdin
  - Regular Reducer receives a collection of all the values for a particular key
  - It is still the case that all the values for a particular key will go to a single Reducer

NEXT GEN HDFS AND MR

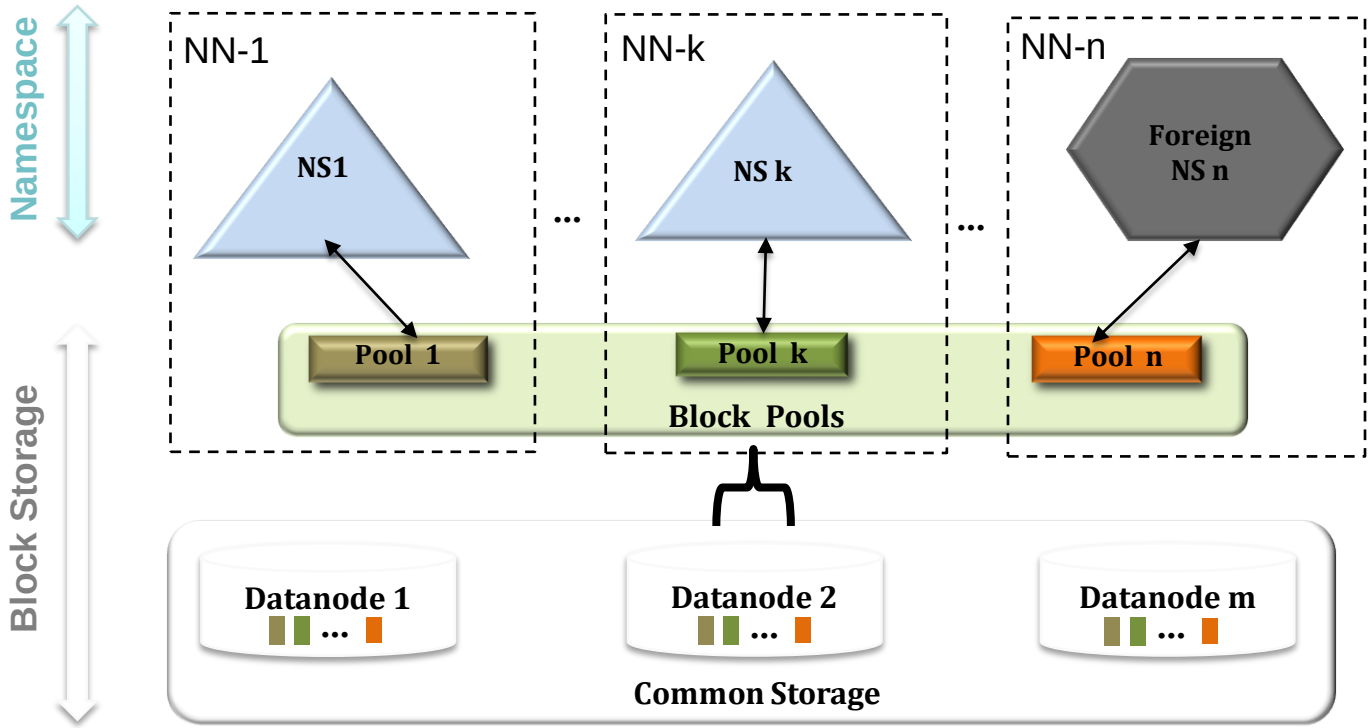
CDH3 HDFS and Map Reduce: Limitations

- Utilization
- Scalability
  - Maximum Cluster size – 4,000 nodes
  - Maximum concurrent tasks – 40,000
  - Coarse synchronization in JobTracker
- Single point of failure
  - Failure kills all queued and running jobs
  - Jobs restarted on bounce

Map Reduce Limitations – Contd.

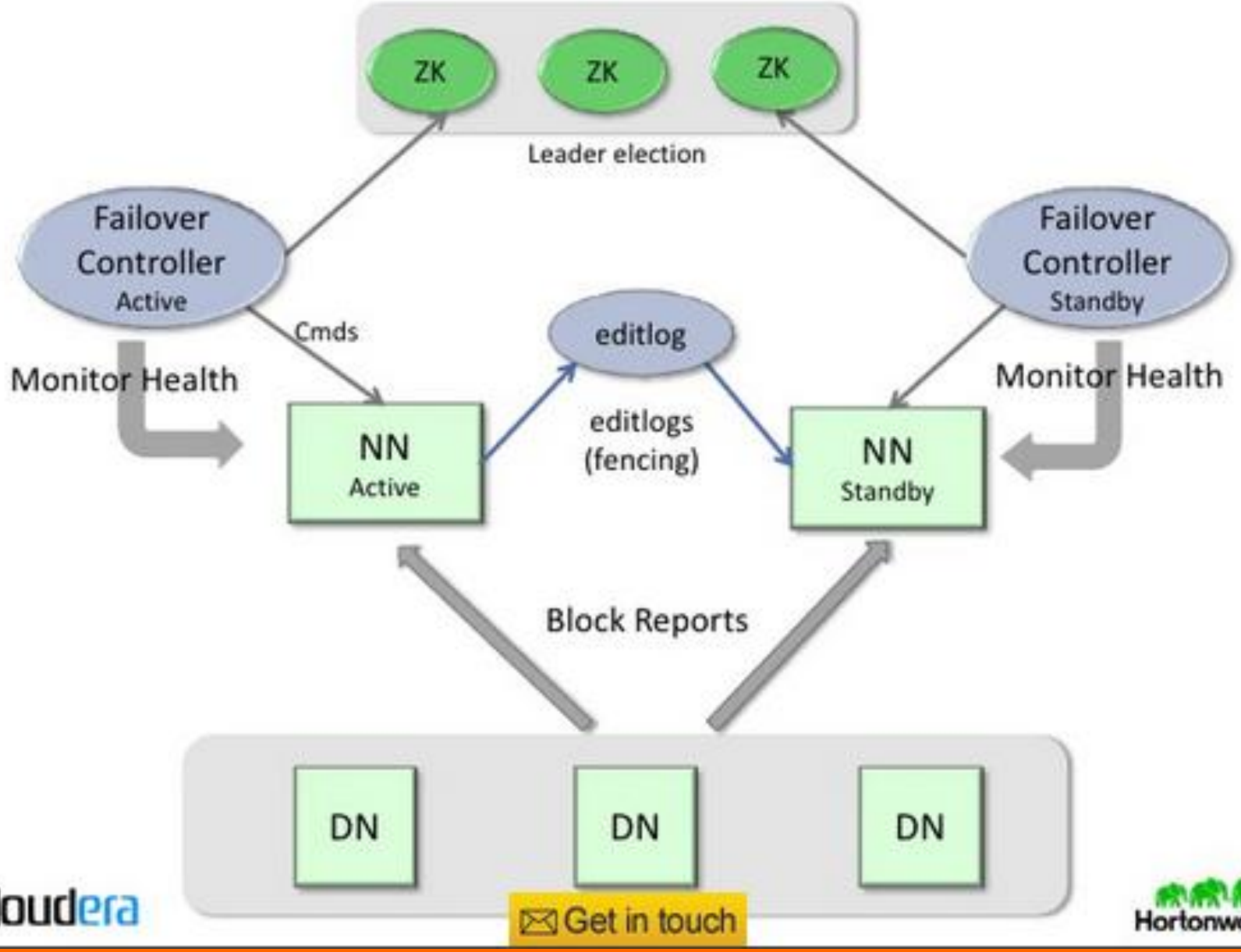
- Hard partition of resources into map and reduce slots
  - Low resource utilization
- Lacks support for alternate paradigms
  - Iterative applications implemented using MapReduce are 10x slower
  - Hacks for the likes of MPI/Graph Processing
- Lack of wire-compatible protocols
  - Client and cluster must be of same version
  - Applications and workflows cannot migrate to different clusters

CDH4 HDFS: (a) Name Node Federation



- Multiple **independent** Namenodes and Namespace Volumes in a cluster
  - Namespace Volume = Namespace + Block Pool
- Block Storage as generic storage service
  - Set of blocks for a Namespace Volume is called a ***Block Pool***
  - DNs store blocks for all the Namespace Volumes – no partitioning

CDH4 HDFS: (b) High Availability



Reference: Hadoop Summit 2012 | HDFS High Availability talk

How Good is CDH-3 HDFS Anyway?

- Data Reliability
  - Lost 19 out of 329 Million blocks on 10 clusters with 20K nodes in 2009
  - 7-9's of reliability
  - Related bugs fixed in 20 and 21.
- NameNode Availability
  - 18 months Study: 22 failures on 25 clusters - 0.58 failures per year per cluster
  - ***Only 8 would have benefitted from HA failover!! (0.23 failures per cluster year)***
  - NN is very reliable
    - ✓ Resilient against overload caused by misbehaving apps
- Maintainability
  - Large clusters see failure of one DataNode/day and more frequent disk failures
  - Maintenance once in 3 months to repair or replace DataNodes

cloudera

Get in touch

Hortonworks

Reference: Hadoop Summit 2012 | HDFS High Availability talk

INSOFE
Inspire...Educate...Transform.

International School of Engineering

Plot 63/A, 1st Floor, Road # 13, Film Nagar, Jubilee Hills, Hyderabad - 500 033

For Individuals: +91-9502334561/63 or 040-65743991

For Corporates: +91-9618483483

Web: http://www.insofe.edu.in

Facebook: https://www.facebook.com/insofe

Twitter: https://twitter.com/Insofeedu

YouTube: http://www.youtube.com/InsofeVideos

SlideShare: http://www.slideshare.net/INSOFE

LinkedIn: http://www.linkedin.com/company/international-school-of-engineering