An Introduction to MapReduce:

Abstractions and Beyond!

What We'll Be Covering…

- Background information/overview
- Map abstraction
  - . example
- Reduce abstraction
  - example
- Combining the map and reduce abstractions
- Why MapReduce is "better"
- Examples and applications of MapReduce

Before MapReduce…

- Large scale data processing was difficult!
  - Managing hundreds or thousands of processors
  - Managing parallelization and distribution
  - I/O Scheduling
  - Status and monitoring
  - Fault/crash tolerance
- MapReduce provides all of these, easily!

MapReduce Overview

- What is it?
  - Programming model used by Google
  - A combination of the Map and Reduce models with an associated implementation
  - Used for processing and generating large data sets

MapReduce Overview

- How does it solve our previously mentioned problems?
  - MapReduce is highly scalable and can be used across many computers.
  - Many small machines can be used to process jobs that normally could not be processed by a large machine.

Input data is distributed to nodes

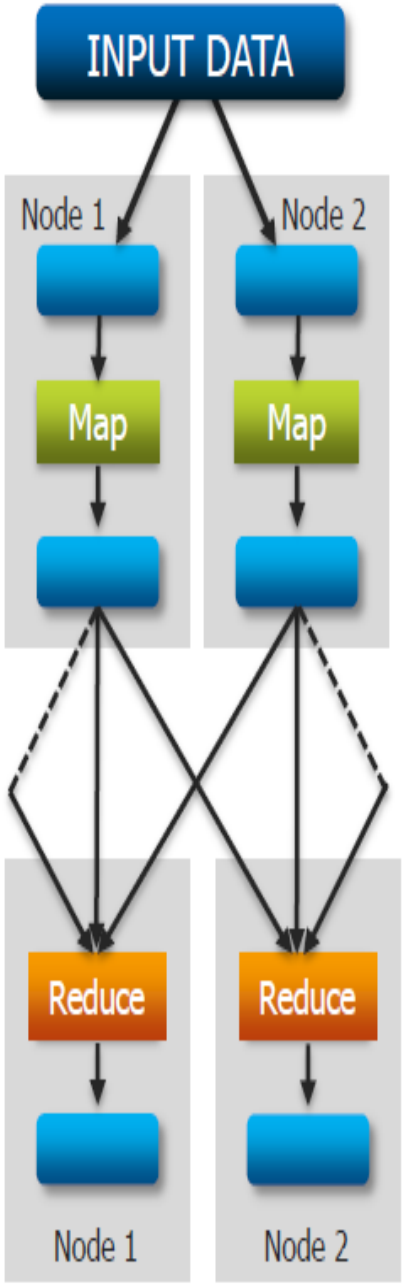
Each map task works on a “split” of data

Mapper outputs intermediate data

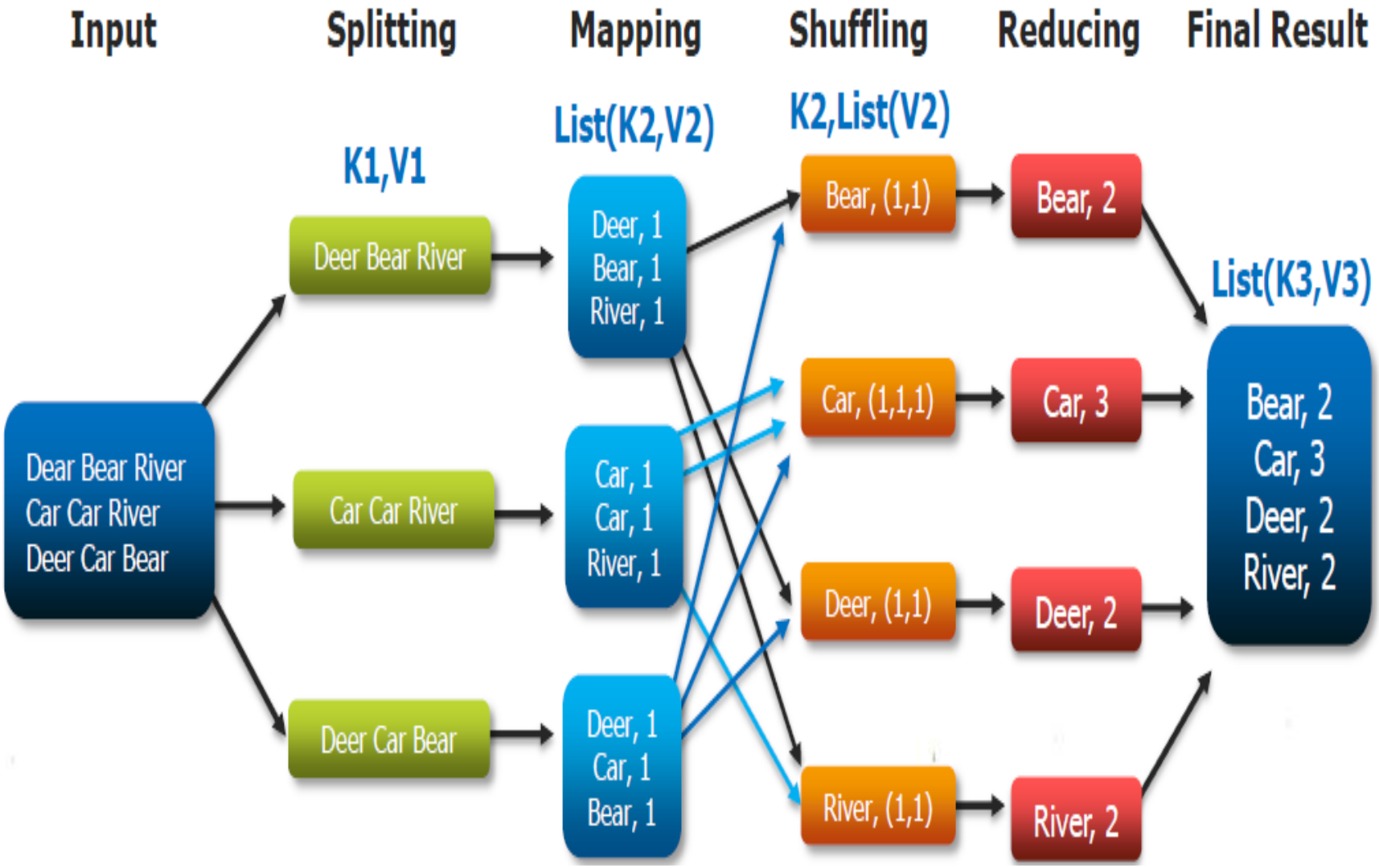
Data exchange between nodes in a “shuffle” process

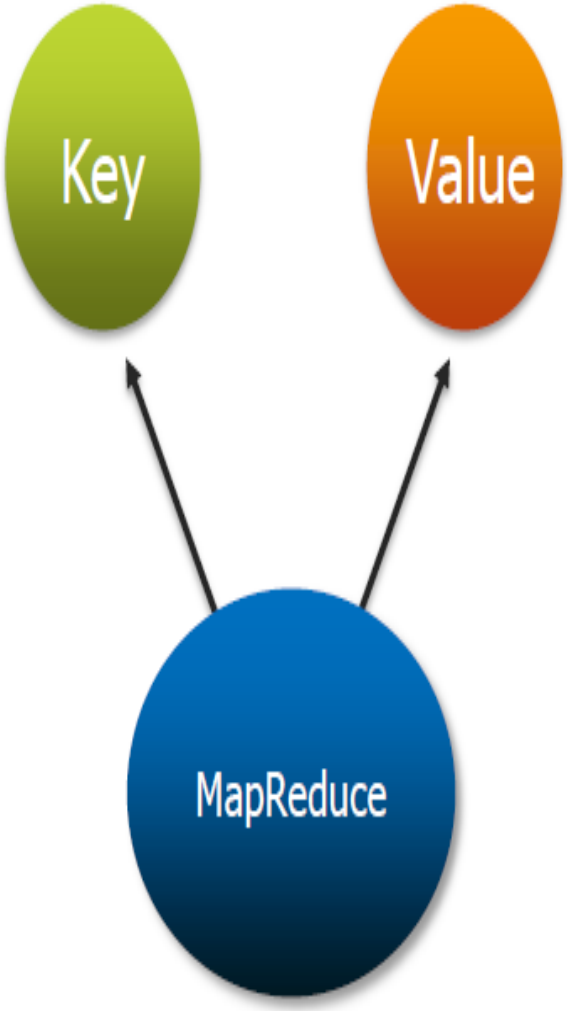
Intermediate data of the same key goes to the same reducer

Reducer output is stored



The Overall MapReduce Word Count Process





Map:

(K1, V1) → List (K2, V2)

Reduce:

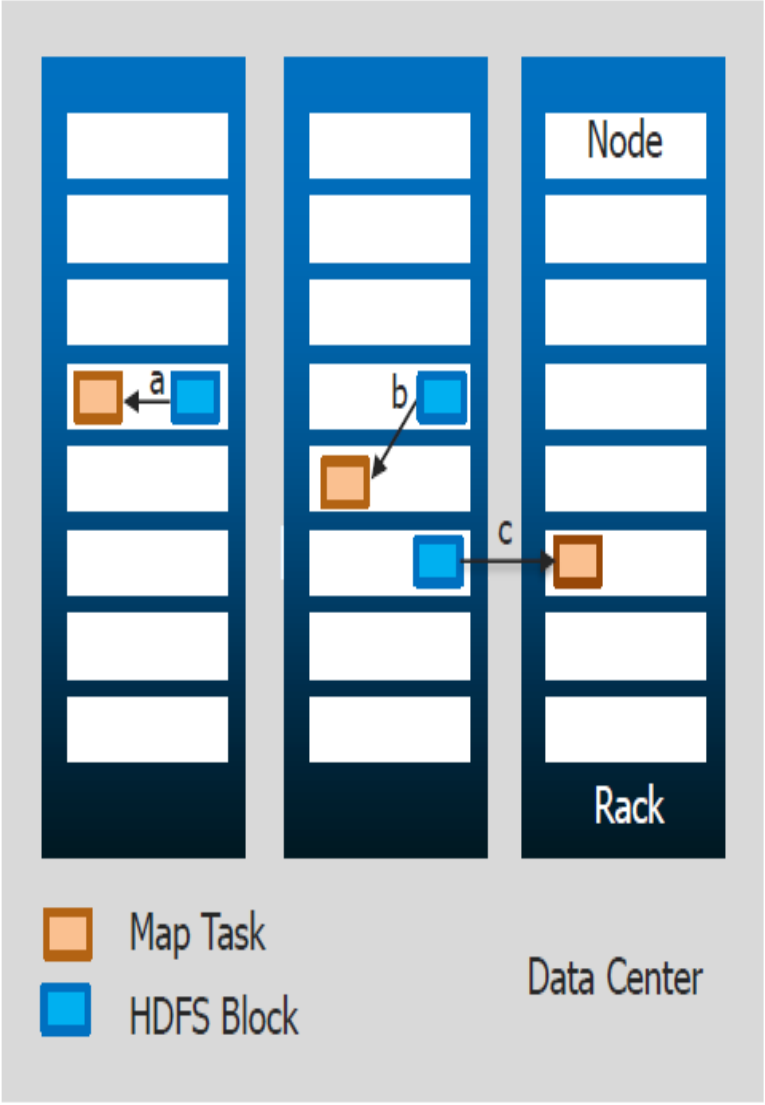
(K2, list (V2)) → List (K3, V3)

Input to the mapper is in the form of?

- file
- key, value
- only string
- all the above

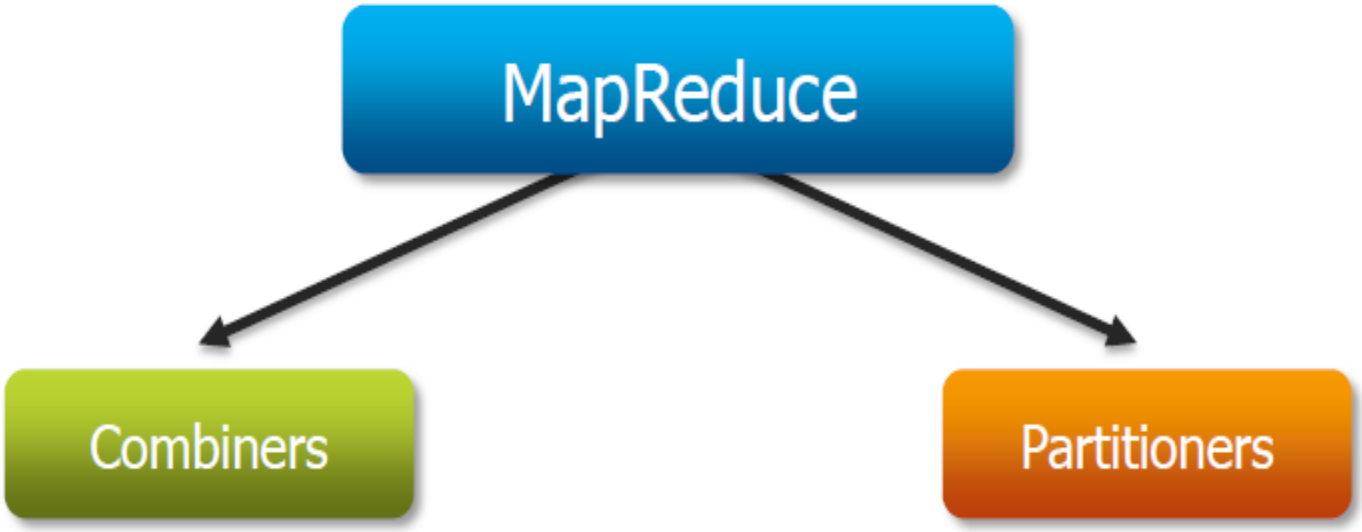
Why MapReduce?

- ✓ Two biggest Advantages:
  - ✓ Taking processing to the data
  - ✓ Processing data in parallel



Overview Of MapReduce

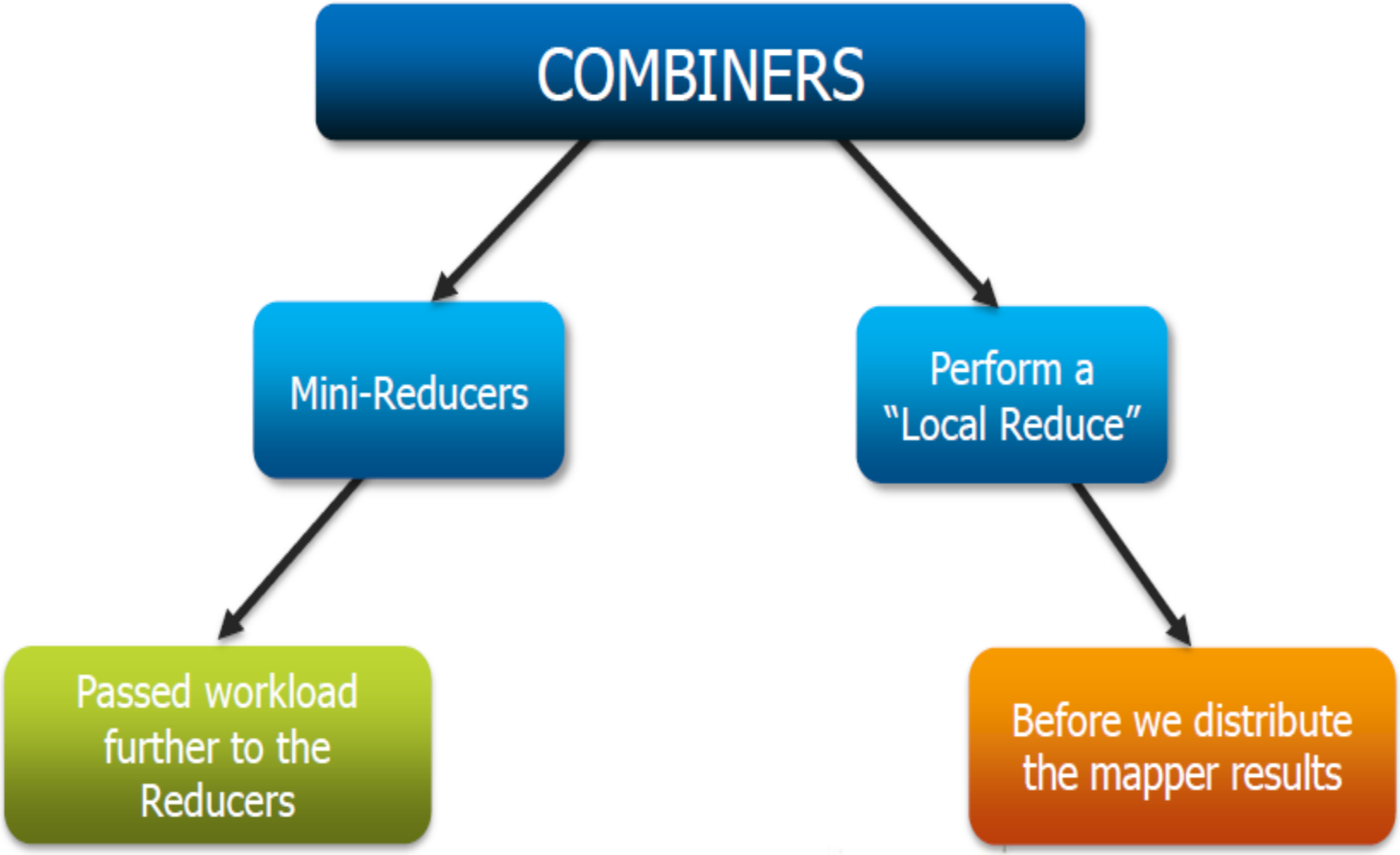
Complete view of MapReduce, illustrating combiners and partitioners in addition to Mappers and Reducers



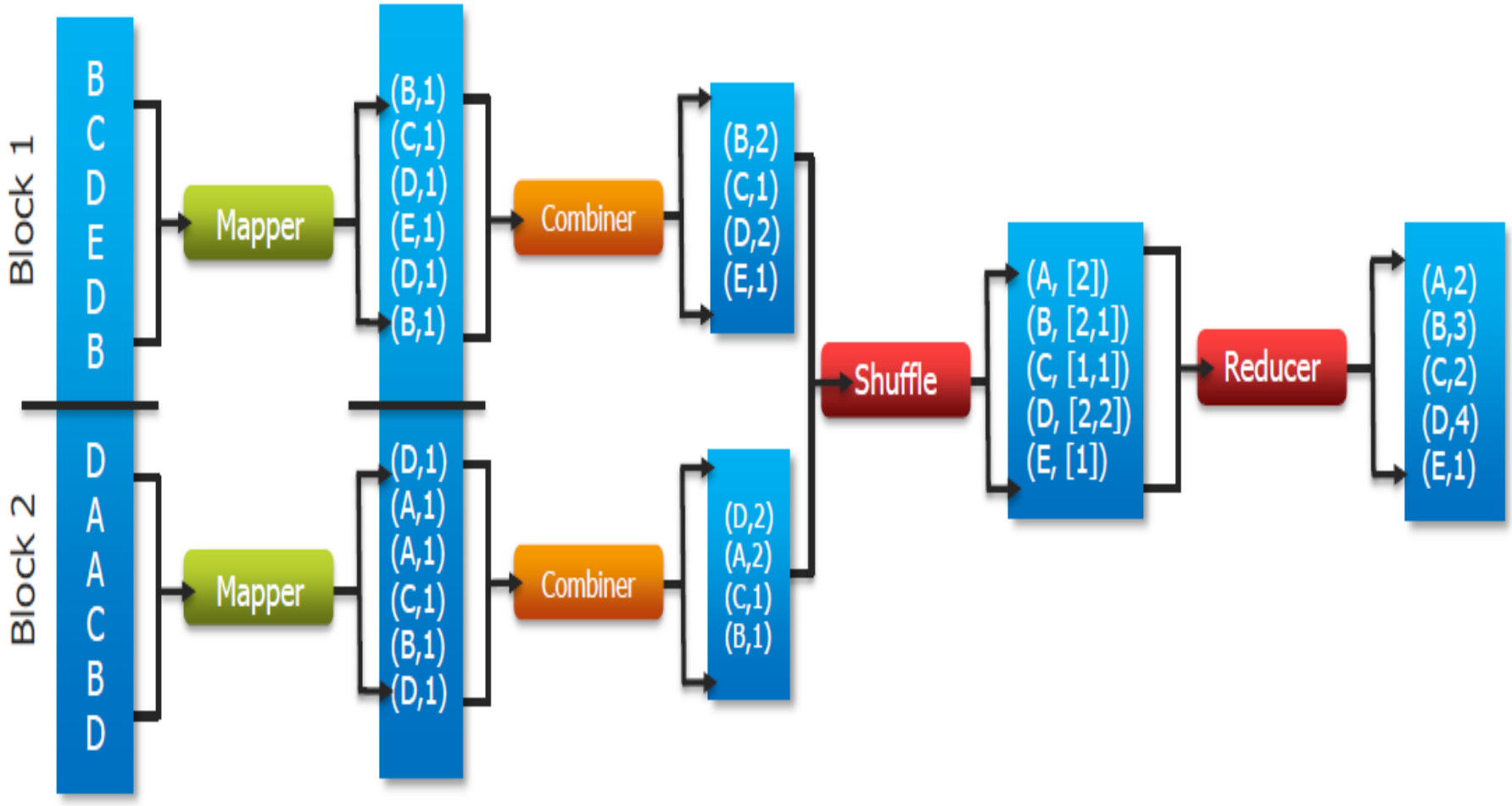
Combiners can be viewed as 'mini-reducers' in the Map phase.

Partitioners determine which reducer is responsible for a particular key.

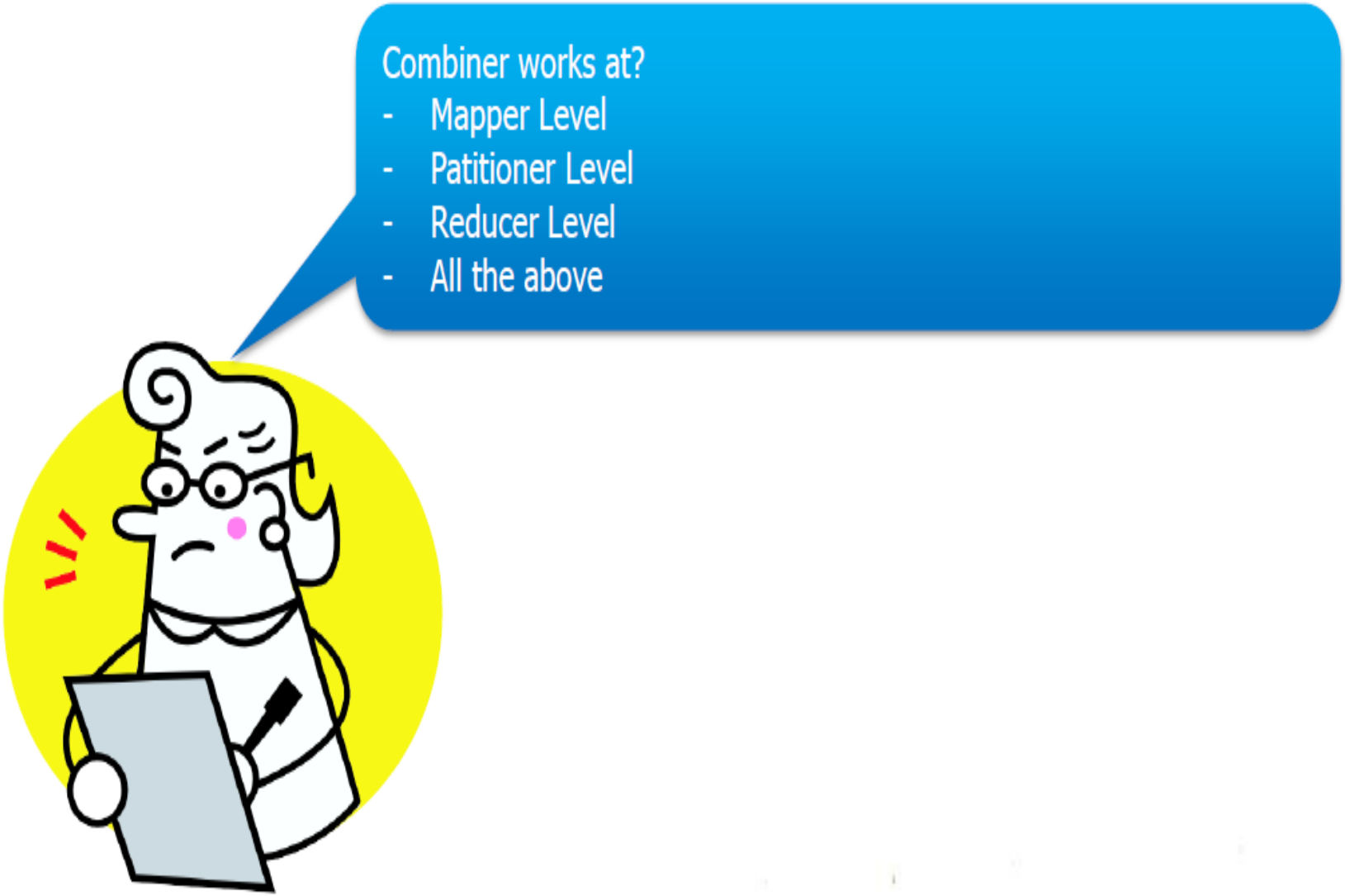
Combiner – Local Reduce



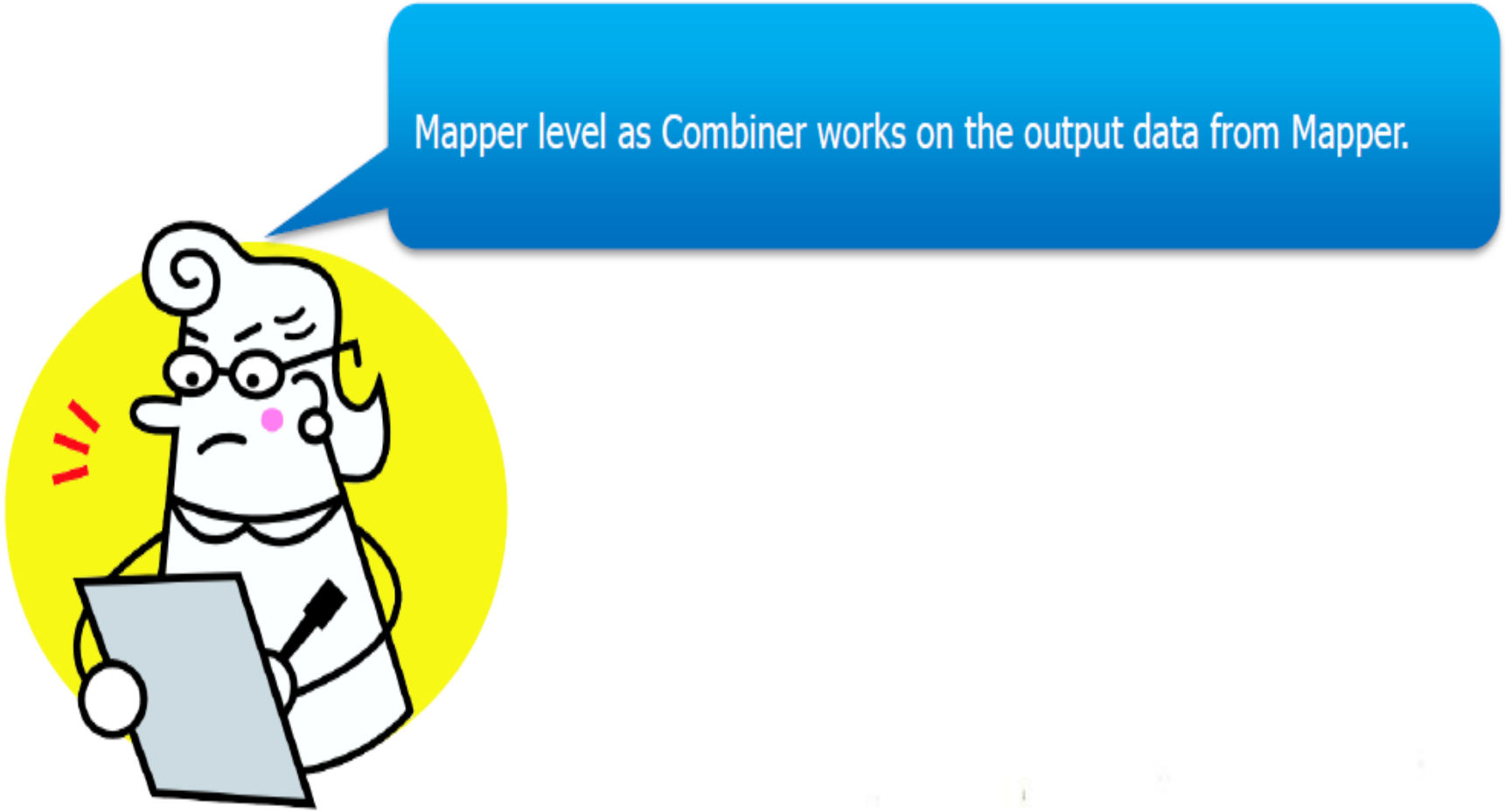
Combiner


Block 1
B C D E D B
Block 2
D A A C B D
Mapper
Mapper
(B,1) (C,1) (D,1) (E,1) (D,1) (B,1)
(D,1) (A,1) (A,1) (C,1) (B,1) (D,1)
Combiner
Combiner
(B,2) (C,1) (D,2) (E,1)
(D,2) (A,2) (C,1) (B,1)
Shuffle
(A, [2]) (B, [2,1]) (C, [1,1]) (D, [2,2]) (E, [1])
Reducer
(A,2) (B,3) (C,2) (D,4) (E,1)

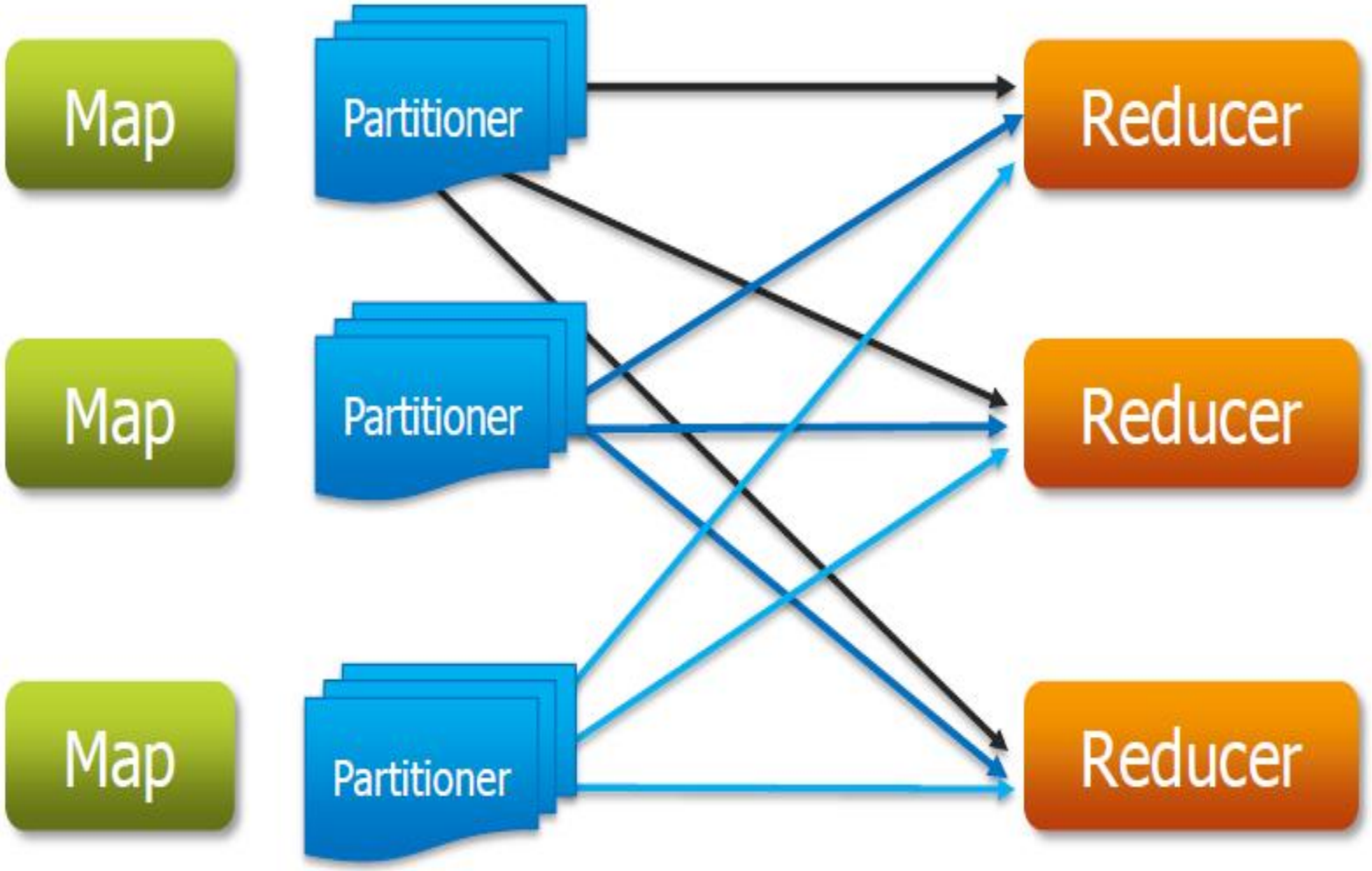
Question?


Combiner works at?
- Mapper Level
- Patitioner Level
- Reducer Level
- All the above

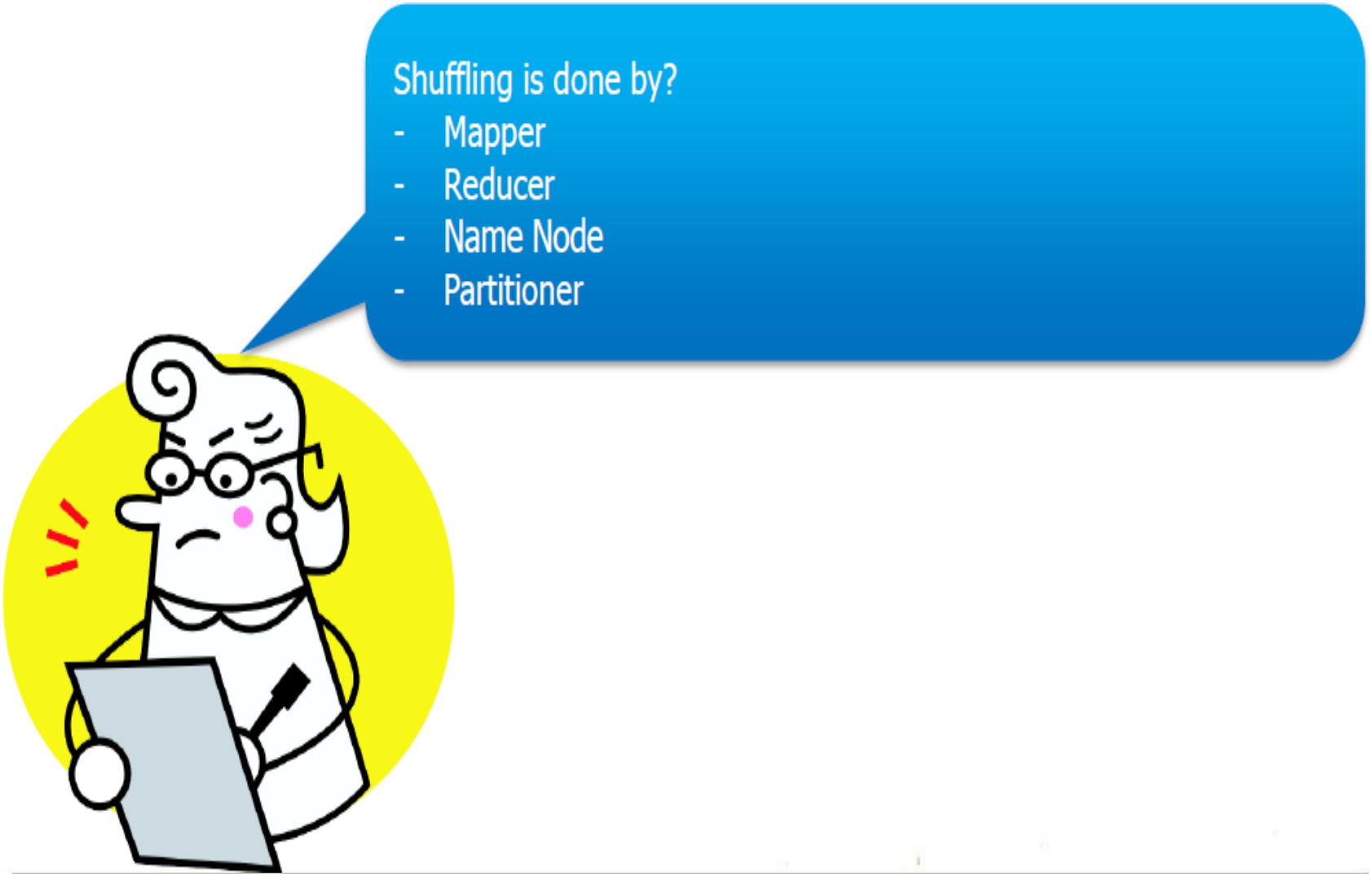
Answer


Mapper level as Combiner works on the output data from Mapper.

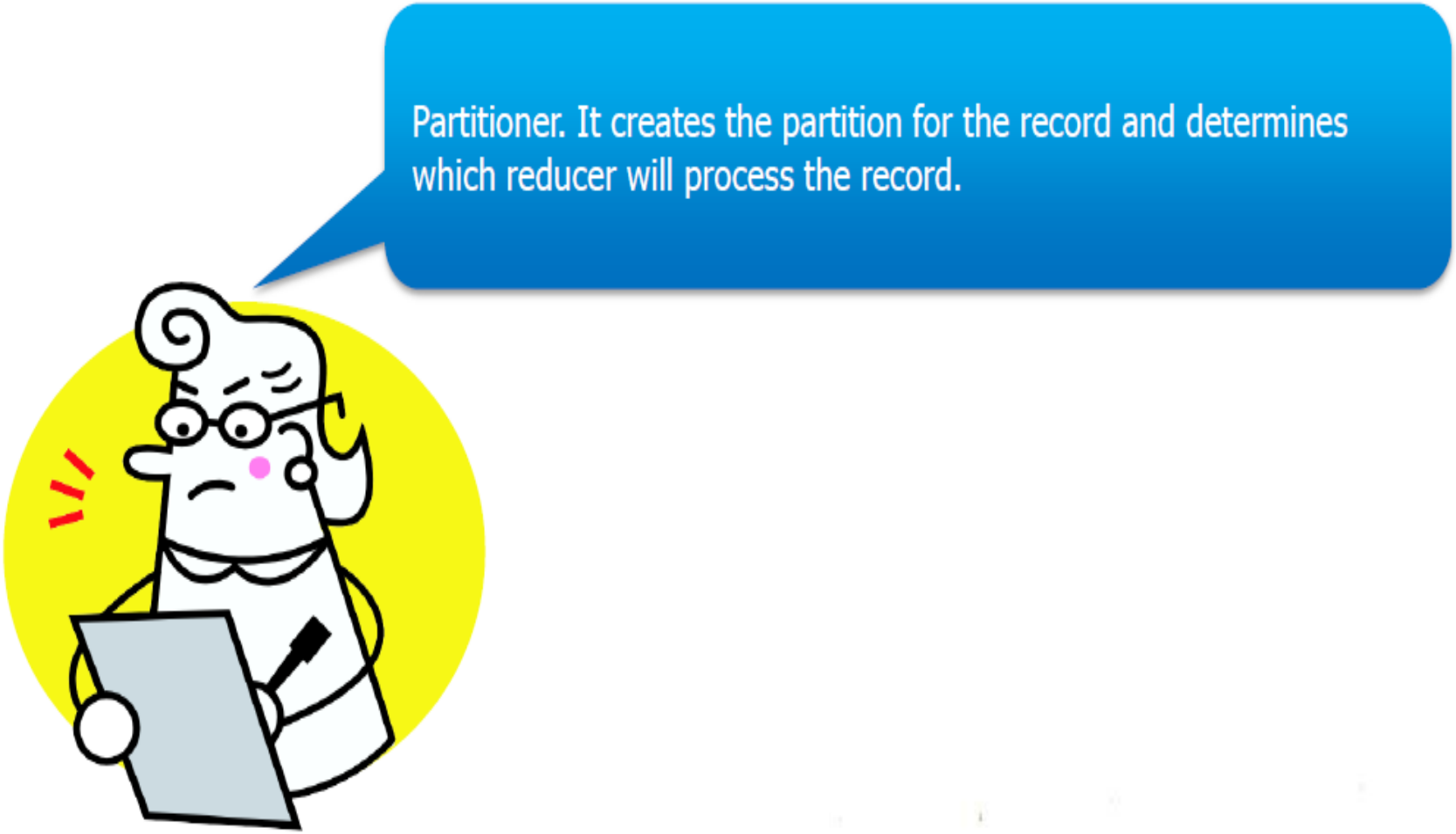
Partitioner: Redirecting output from mapper



Question?



Answer


Partitioner. It creates the partition for the record and determines which reducer will process the record.

Data flow

- Input, final output are stored on a distributed file system
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local FS of map and reduce workers
- Output is often input to another map reduce task

Other Applications

- Yahoo!
  - Webmap application uses Hadoop to create a database of information on all known webpages
- Facebook
  - Hive data center uses Hadoop to provide business statistics to application developers and advertisers
- Rackspace
  - Analyzes sever log files and usage data using Hadoop

Why is this approach better?

- Creates an abstraction for dealing with complex overhead
  - The computations are simple, the overhead is messy
- Removing the overhead makes programs much smaller and thus easier to use
  - Less testing is required as well. The MapReduce libraries can be assumed to work properly, so only user code needs to be tested
- Division of labor also handled by the MapReduce libraries, so programmers only need to focus on the actual computation

Conclusions

- MapReduce proven to be useful abstraction
- Greatly simplifies large-scale computations
- Fun to use:
  - focus on problem,
  - let library deal w/ messy details