MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjey Ghemawat

Google, Inc.

Presented By: Hani Khoshdel Nikkhoo

Monday, January 25, 2010

Outline

- Raison d'etre for MapReduce
- Background: Functional Programming
  - Map
  - Fold (Reduce)
- MapReduce in action
- How to use MapReduce
- Why MapReduce is useful

Raison d'etre for MapReduce

The need to process large amounts of raw data (e.g.1000 GB) in a short amount of time (a few minutes)

- Parallelism
- Fault-tolerance

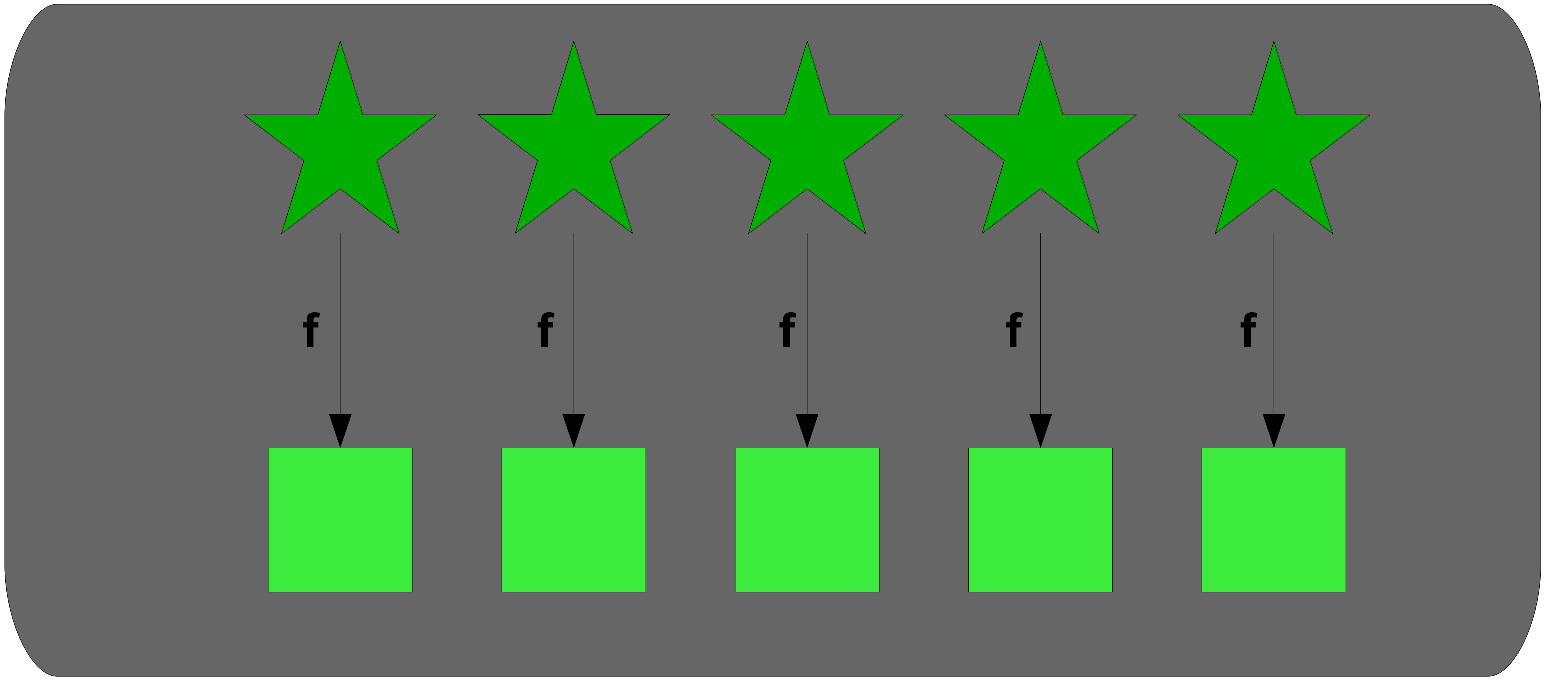
Functional Programming Principles

- When a function is applied to a data structure, the data structure does not change, rather the result is stored in a new data structure.
- A function can be used as the argument of another function.

Map

map f lst

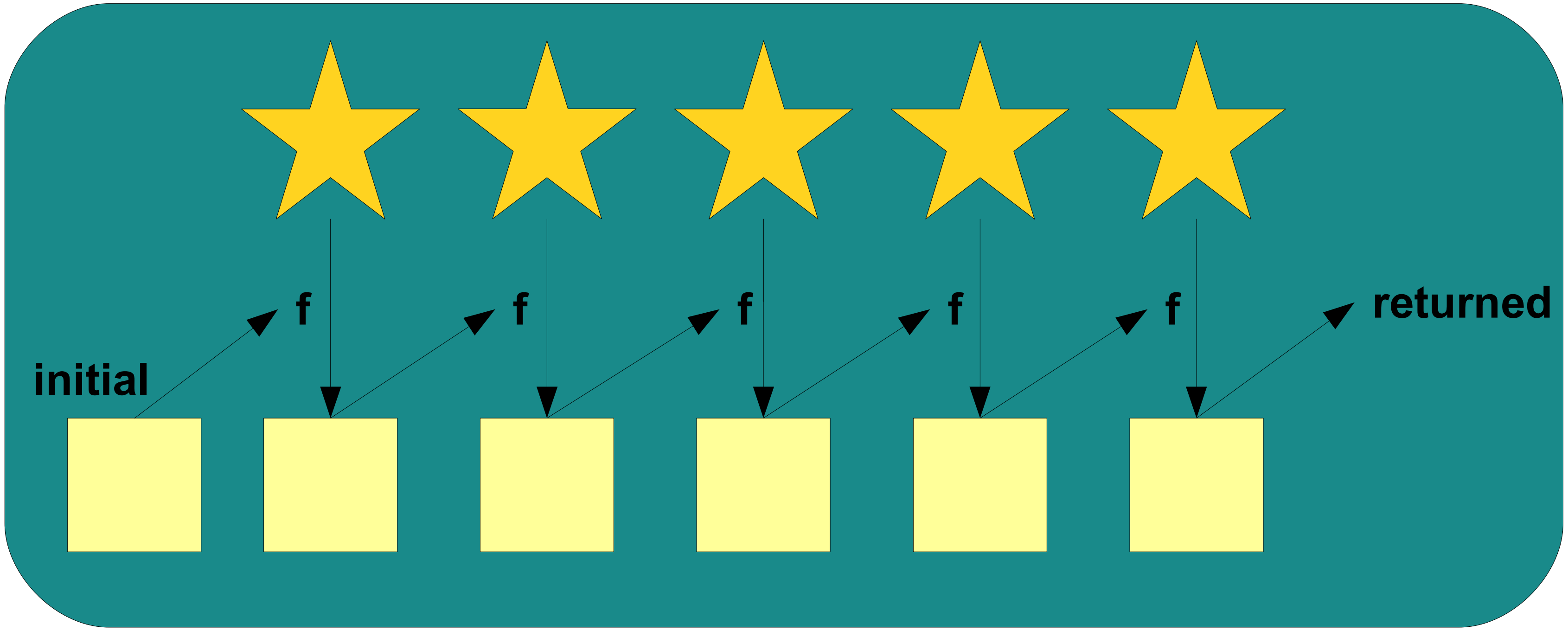
Creates a new list by applying f to each element of the input list; returns output in order. (Adapted from [2,3])



Fold (Reduce)

fold f x_0 lst

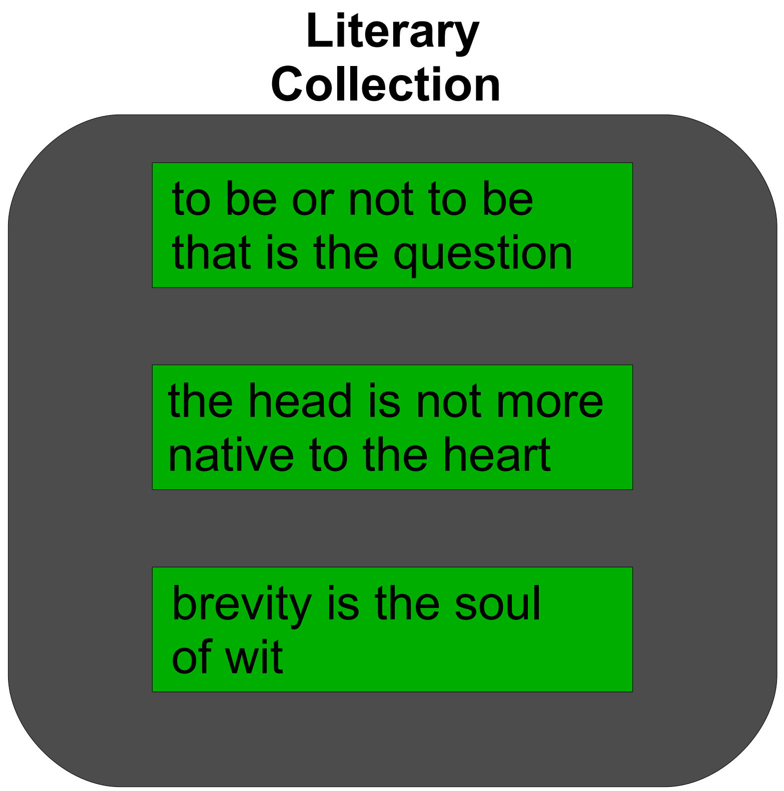
Moves across a list, applying *f* to each element plus an *accumulator*. f returns the next *accumulator* value, which is combined with the next element of the list (Adapted from [2,3])

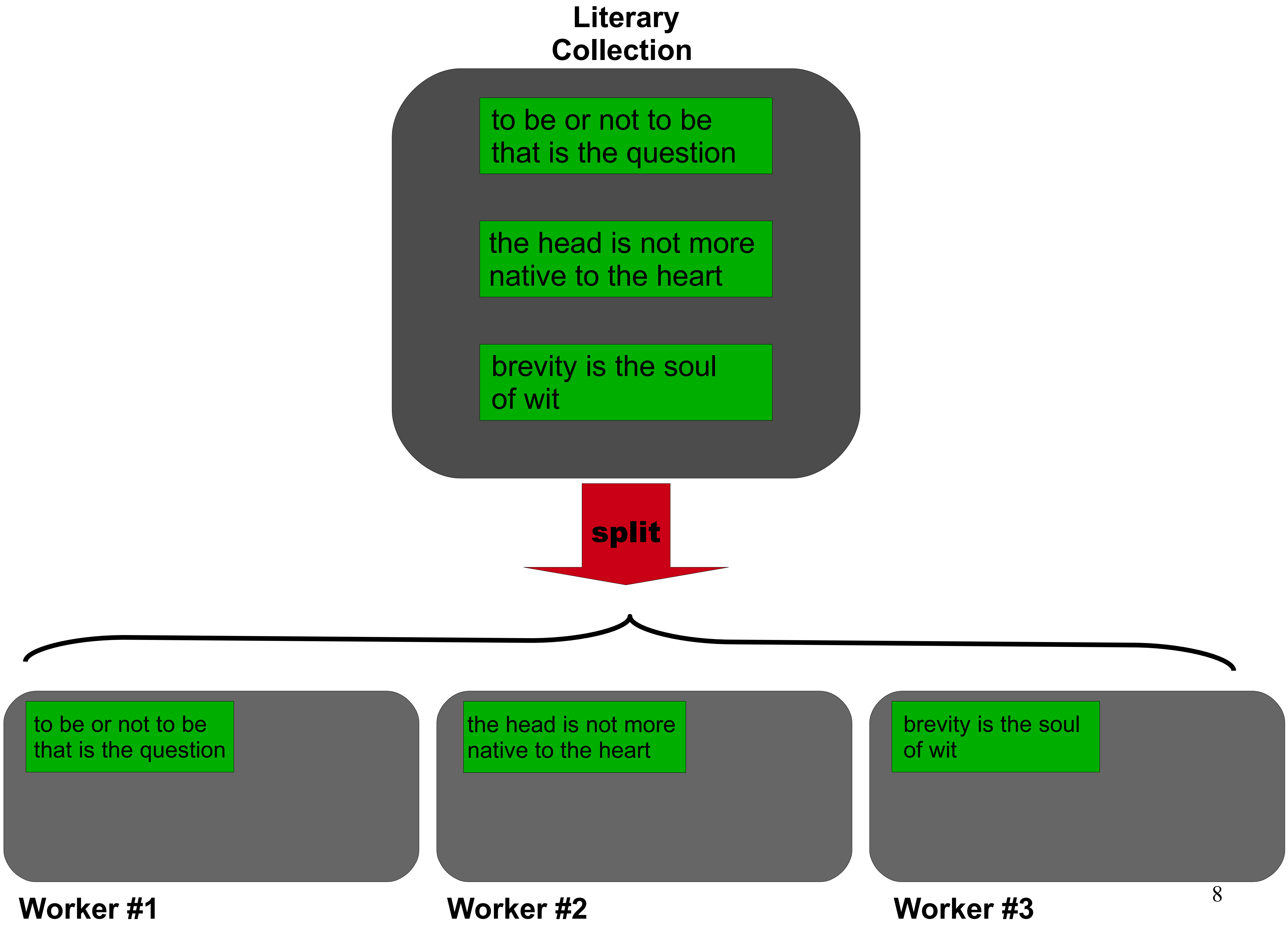


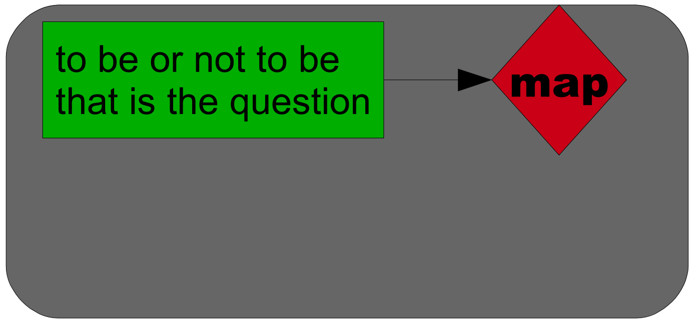
MapReduce in Action

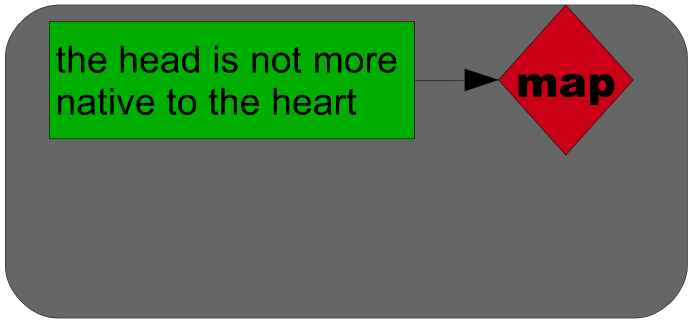
- Problem: counting the number of occurrences of each word in a literary collection.

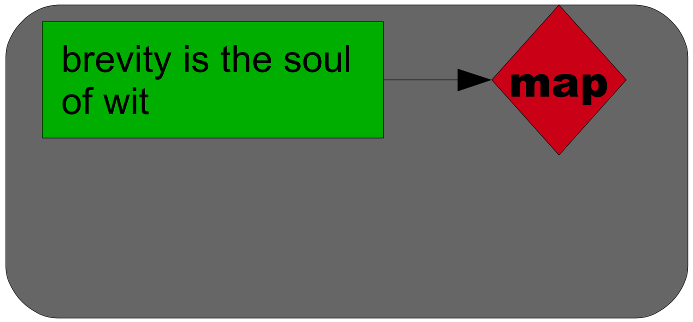
(Ex. Adapted from [4])

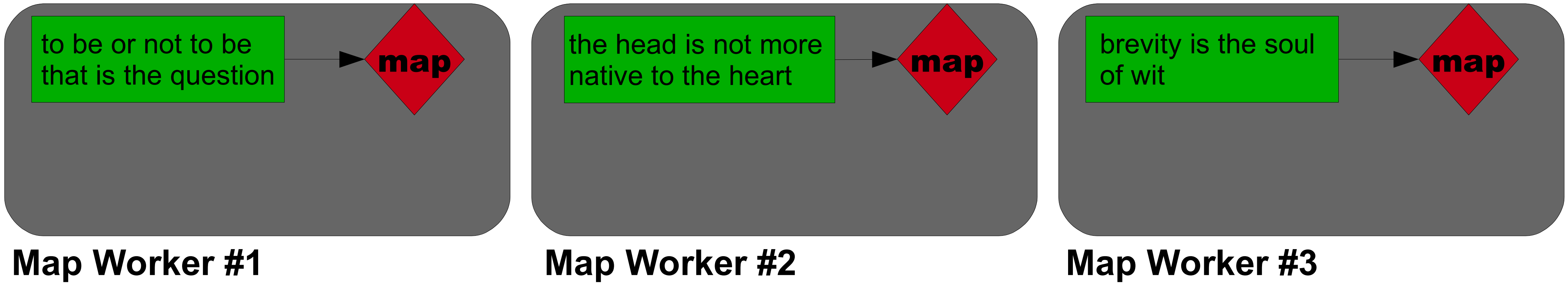

Literary Collection
to be or not to be that is the question
the head is not more native to the heart
brevity is the soul of wit

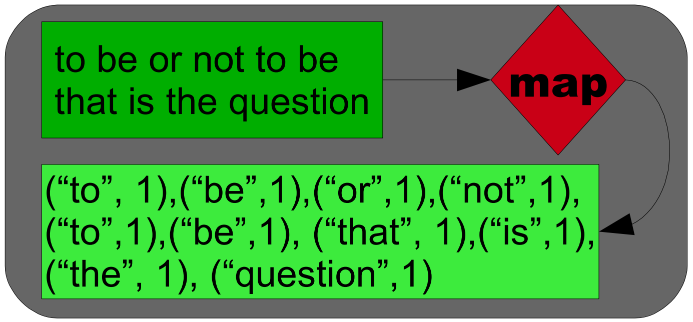

Literary Collection
to be or not to be that is the question
the head is not more native to the heart
brevity is the soul of wit
split
to be or not to be that is the question
the head is not more native to the heart
brevity is the soul of wit
Worker #1
Worker #2
Worker #3


to be or not to be that is the question
map

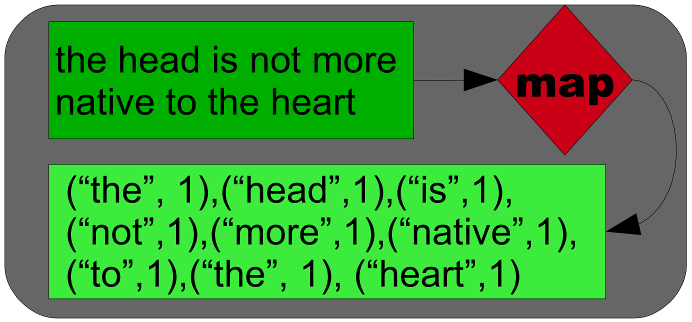

the head is not more native to the heart
map


brevity is the soul of wit
map

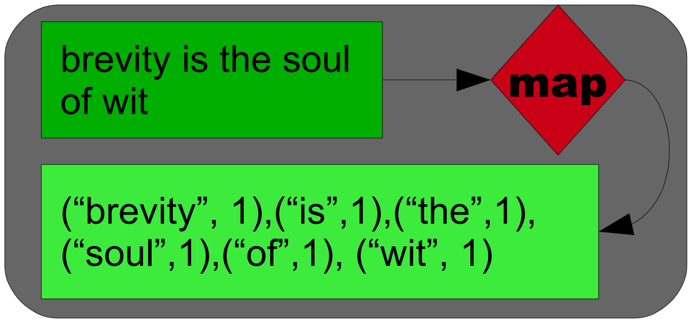

to be or not to be that is the question
map
Map Worker #1
the head is not more native to the heart
map
Map Worker #2
brevity is the soul of wit
map
Map Worker #3


to be or not to be that is the question
map
("to", 1),("be",1),("or",1),("not",1), ("to",1),("be",1), ("that", 1),("is",1), ("the", 1), ("question",1)

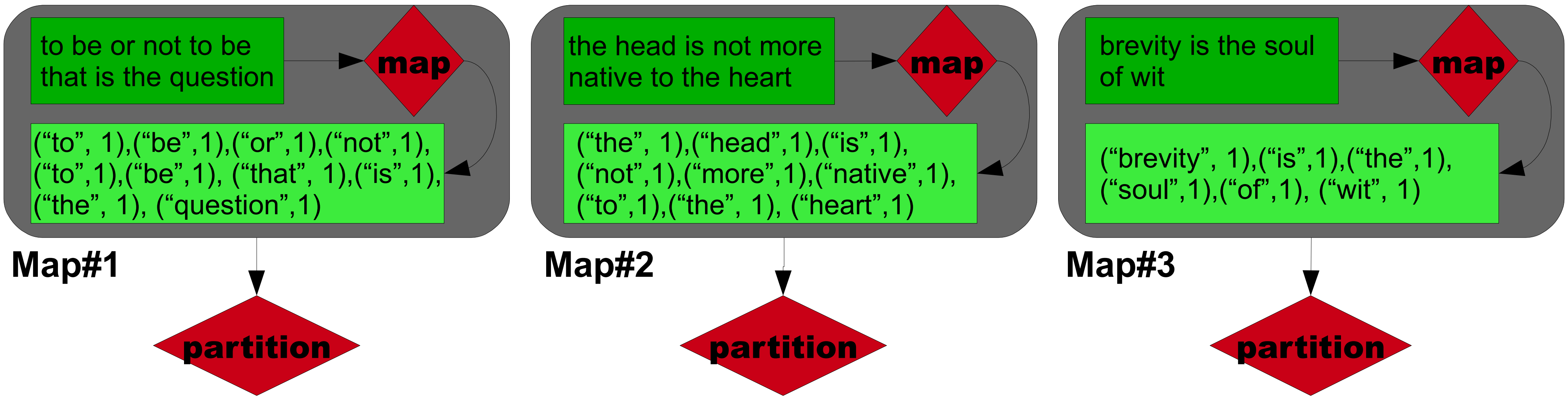
Map#1

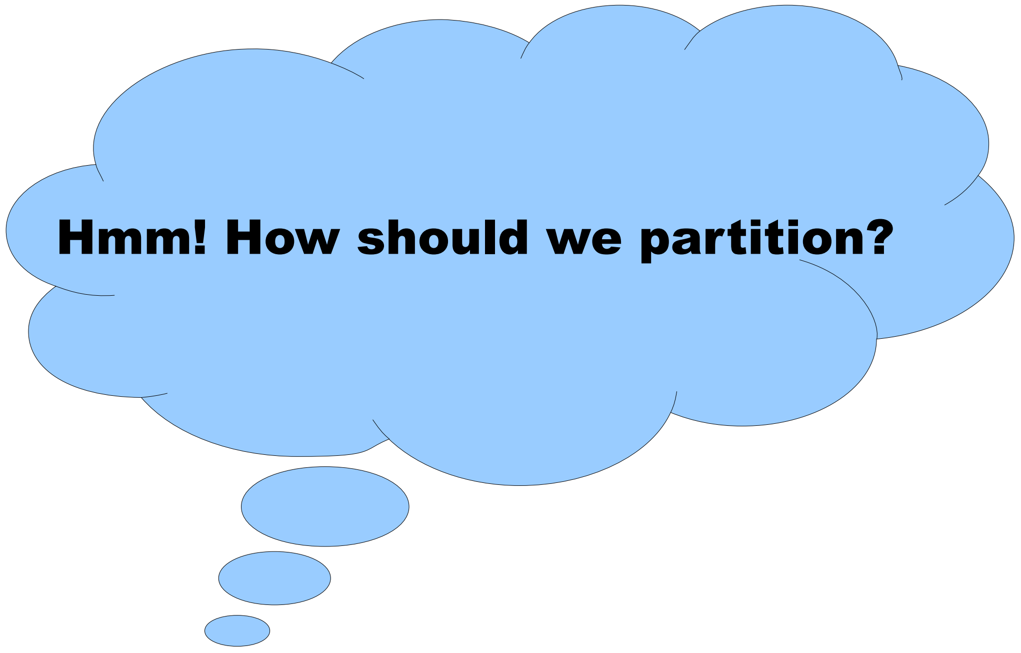

the head is not more native to the heart
map
("the", 1),("head",1),("is",1), ("not",1),("more",1),("native",1), ("to",1),("the", 1), ("heart",1)

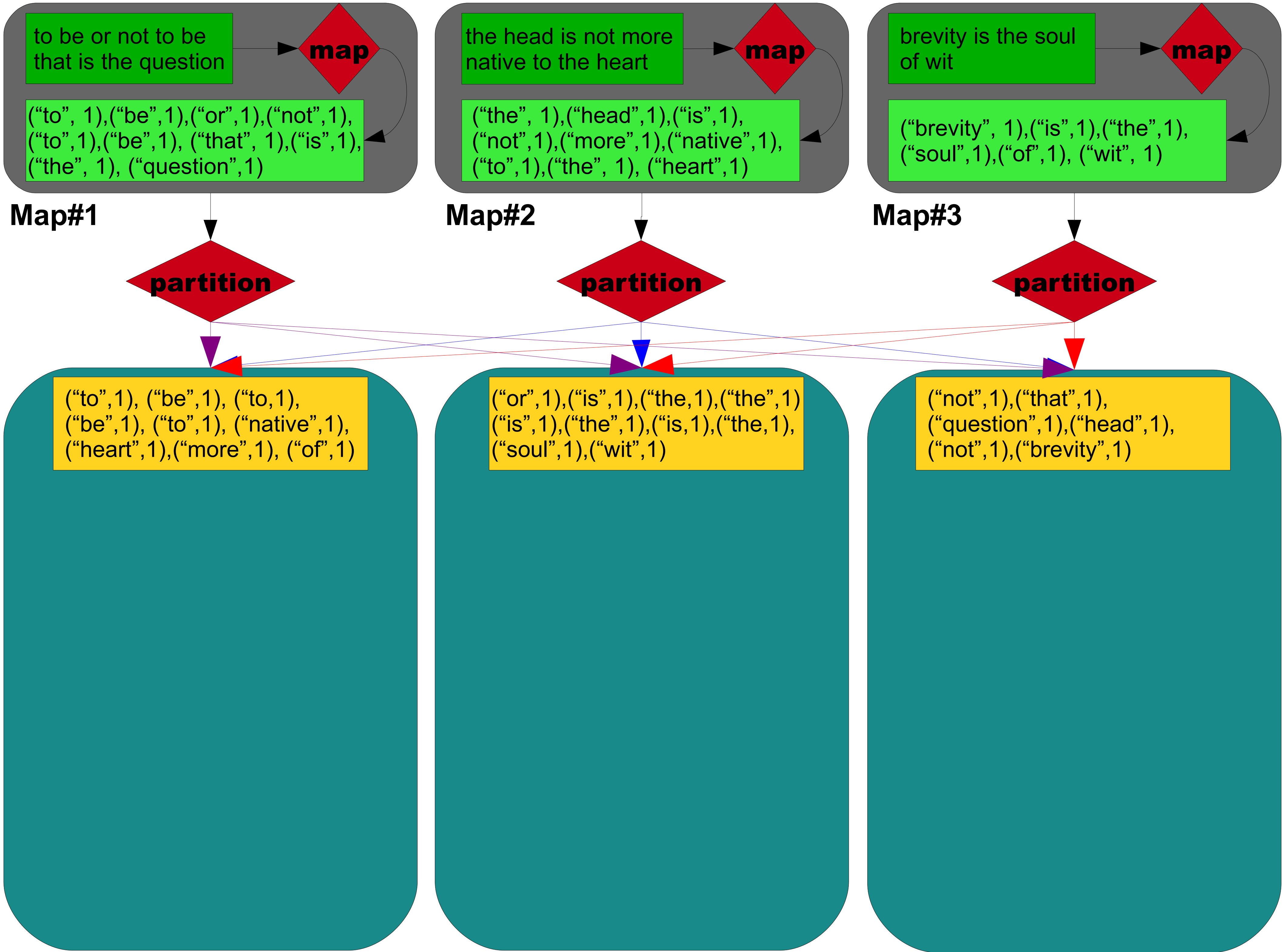
Map#2

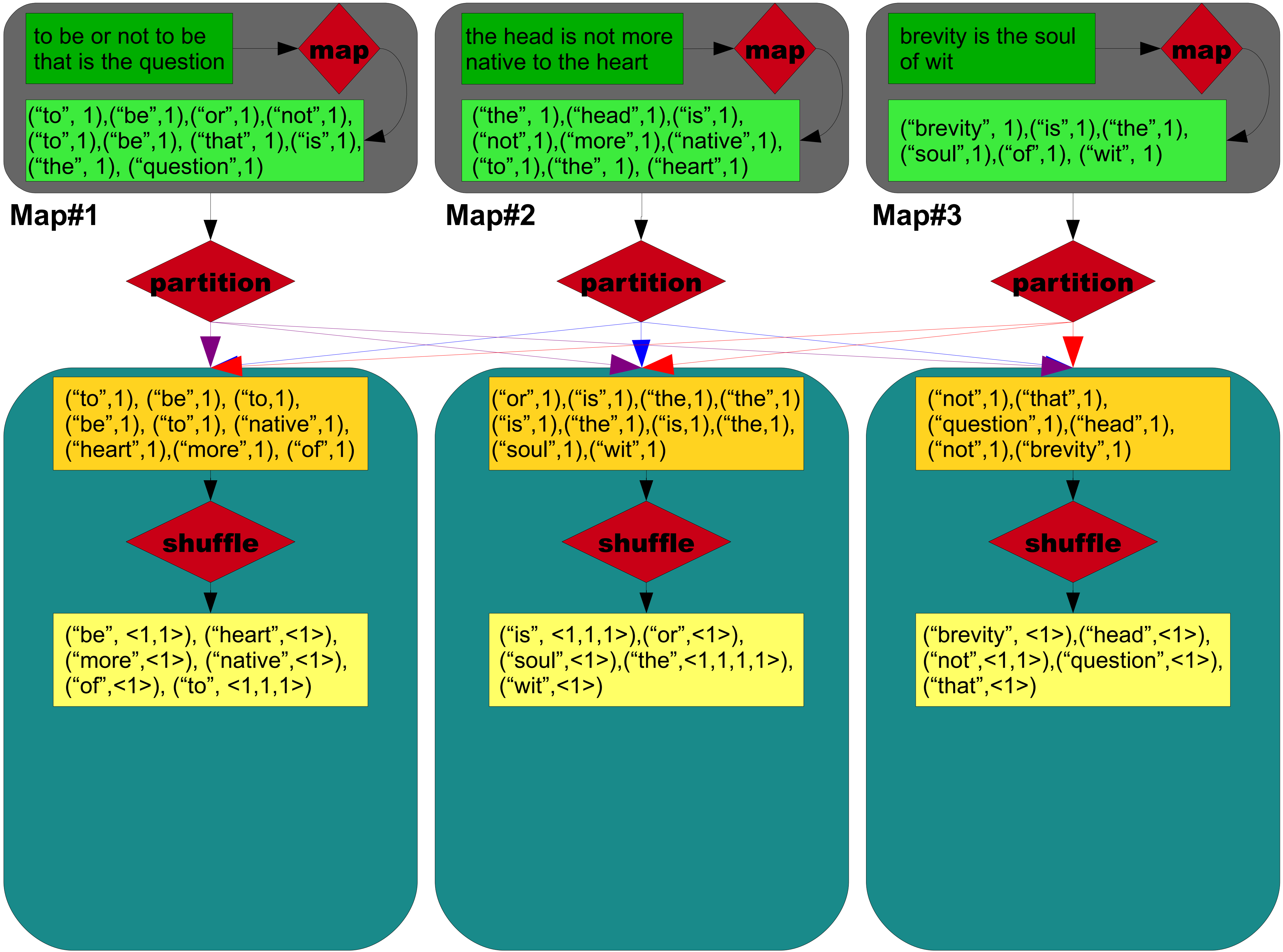

brevity is the soul of wit
map
("brevity", 1),("is",1),("the",1), ("soul",1),("of",1), ("wit", 1)

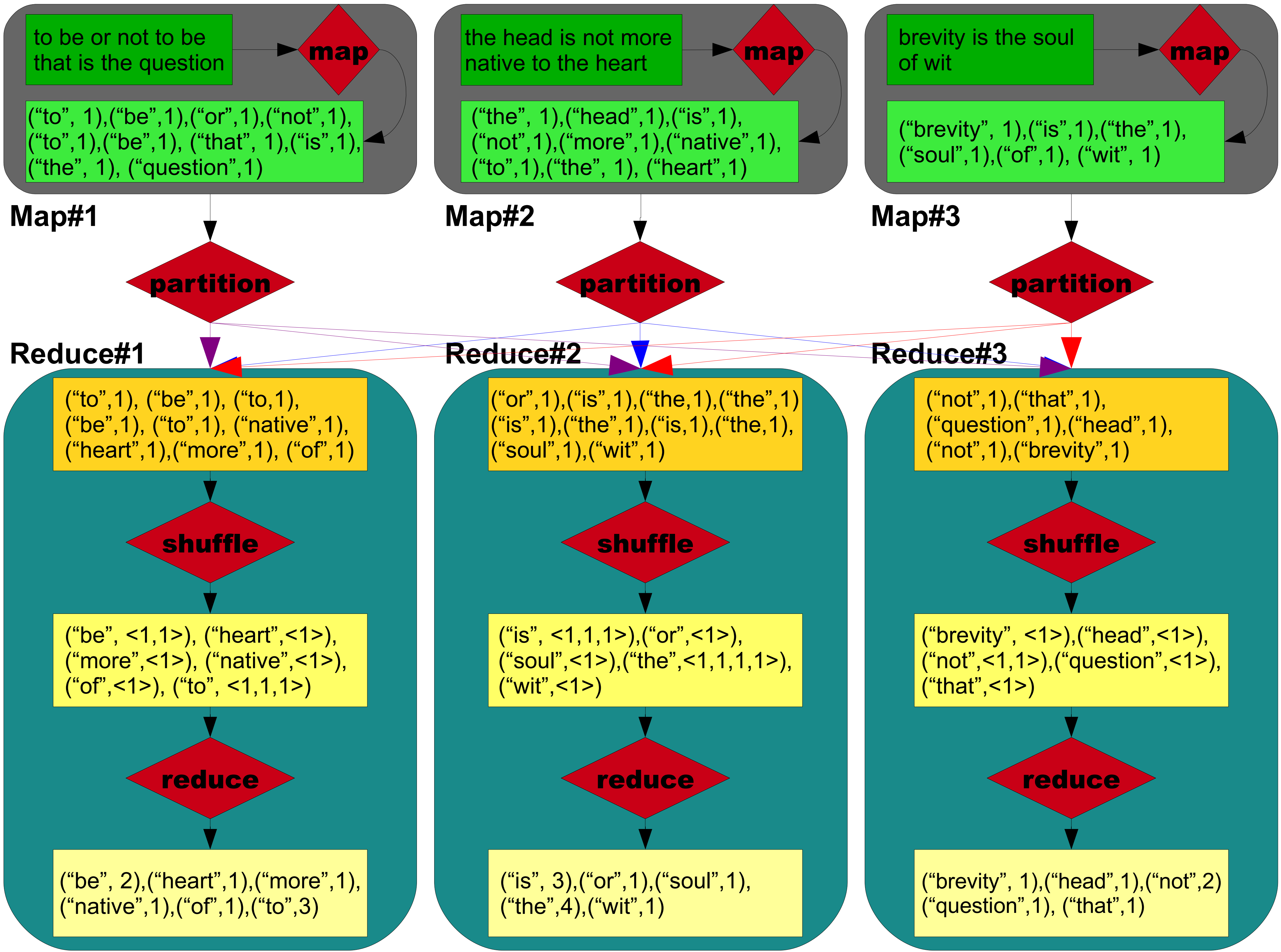
Map#3


to be or not to be that is the question
map
("to", 1),("be",1),("or",1),("not",1), ("to",1),("be",1), ("that", 1),("is",1), ("the", 1), ("question",1)
Map#1
partition
the head is not more native to the heart
map
("the", 1),("head",1),("is",1), ("not",1),("more",1),("native",1), ("to",1),("the", 1), ("heart",1)
Map#2
partition
brevity is the soul of wit
map
("brevity", 1),("is",1),("the",1), ("soul",1),("of",1), ("wit", 1)
Map#3
partition


Hmm! How should we partition?


to be or not to be
that is the question
map
("to", 1),("be",1),("or",1),("not",1),
("to",1),("be",1), ("that", 1),("is",1),
("the", 1), ("question",1)
Map#1
the head is not more
native to the heart
map
("the", 1),("head",1),("is",1),
("not",1),("more",1),("native",1),
("to",1),("the", 1), ("heart",1)
Map#2
brevity is the soul
of wit
map
("brevity", 1),("is",1),("the",1),
("soul",1),("of",1), ("wit", 1)
Map#3
partition
partition
partition
("to",1), ("be",1), ("to,1),
("be",1), ("to",1), ("native",1),
("heart",1),("more",1), ("of",1)
("or",1),("is",1),("the,1),("the",1)
("is",1),("the",1),("is,1),("the,1),
("soul",1),("wit",1)
("not",1),("that",1),
("question",1),("head",1),
("not",1),("brevity",1)


to be or not to be that is the question
map
(“to”, 1),(“be”,1),(“or”,1),(“not”,1),(“to”,1),(“be”,1), (“that”, 1),(“is”,1),(“the”, 1), (“question”,1)
Map#1
the head is not more native to the heart
map
(“the”, 1),(“head”,1),(“is”,1),(“not”,1),(“more”,1),(“native”,1),(“to”,1),(“the”, 1), (“heart”,1)
Map#2
brevity is the soul of wit
map
(“brevity”, 1),(“is”,1),(“the”,1),(“soul”,1),(“of”,1), (“wit”, 1)
Map#3
partition
partition
partition
(“to”,1), (“be”,1), (“to,1), (“be”,1), (“to”,1), (“native”,1), (“heart”,1),(“more”,1), (“of”,1)
shuffle
(“be”, <1,1>), (“heart”,<1>), (“more”,<1>), (“native”,<1>), (“of”,<1>), (“to”, <1,1,1>)
(“or”,1),(“is”,1),(“the,1),(“the”,1) (“is”,1),(“the”,1),(“is,1),(“the,1), (“soul”,1),(“wit”,1)
shuffle
(“is”, <1,1,1>),(“or”,<1>), (“soul”,<1>),(“the”,<1,1,1,1>), (“wit”,<1>)
(“not”,1),(“that”,1), (“question”,1),(“head”,1), (“not”,1),(“brevity”,1)
shuffle
(“brevity”, <1>),(“head”,<1>), (“not”,<1,1>),(“question”,<1>), (“that”,<1>)


to be or not to be that is the question
map
("to", 1),("be",1),("or",1),("not",1), ("to",1),("be",1), ("that", 1),("is",1), ("the", 1), ("question",1)
Map#1
the head is not more native to the heart
map
("the", 1),("head",1),("is",1), ("not",1),("more",1),("native",1), ("to",1),("the", 1), ("heart",1)
Map#2
brevity is the soul of wit
map
("brevity", 1),("is",1),("the",1), ("soul",1),("of",1), ("wit", 1)
Map#3
partition
partition
partition
Reduce#1
Reduce#2
Reduce#3
("to",1), ("be",1), ("to,1), ("be",1), ("to",1), ("native",1), ("heart",1),("more",1), ("of",1)
shuffle
("be", <1,1>), ("heart",<1>), ("more",<1>), ("native",<1>), ("of",<1>), ("to", <1,1,1>)
reduce
("be", 2),("heart",1),("more",1), ("native",1),("of",1),("to",3)
("or",1),("is",1),("the,1),("the",1) ("is",1),("the",1),("is,1),("the,1), ("soul",1),("wit",1)
shuffle
("is", <1,1,1>),("or",<1>), ("soul",<1>),("the",<1,1,1,1>), ("wit",<1>)
reduce
("is", 3),("or",1),("soul",1), ("the",4),("wit",1)
("not",1),("that",1), ("question",1),("head",1), ("not",1),("brevity",1)
shuffle
("brevity", <1>),("head",<1>), ("not",<1,1>),("question",<1>), ("that",<1>)
reduce
("brevity", 1),("head",1),("not",2) ("question",1), ("that",1)

How to use MapReduce

- The user needs to worry only about two things:
  - The *Map* function
  - The *Reduce* function

Why is MapReduce useful?

- The model is easy to use
  - Complexities hidden from users
  - A variety of problems expressible in this framework
- Scalability
  - Parallelism
- Fault-tolerance
  - Recovery

References

(1)Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In Proc. Symposium on Operating Systems Design and Implementation (OSDI'04), pages 137-150, 2004.

(2)Christophe Bisciglia, Aaron Kimball, & Sierra Michels-Slettvet. “MapReduce Theory and Implementation”, Distributed Computing Seminar, Summer 2007

(3)Aaron Kimball,“Cluster Computing and MapReduce Lecture 2”, Google Inc., Summer 2007, Google Code University

http://www.youtube.com/watch?v=-vD6PUdf3Js

(4)Buettcher S., Clarke C.L.A. and Cormack G.V., Information Retrieval: Implementing and Evaluating Search Engines, MIT Press, 2010.