COMP 322: Fundamentals of Parallel Programming

Lecture 7: Map/Reduce

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu

Worksheet #6: Associativity and Commutativity

Recap:
A binary function f is *associative* if f(f(x,y),z) = f(x,f(y,z)).
A binary function f is *commutative* if f(x,y) = f(y,x).

Worksheet problems:
1) Claim: a Finish Accumulator (FA) can only be used with operators that are *associative and commutative.* Why? What can go wrong with accumulators if the operator is non-associative or non-commutative?
You may get different answers in different executions if the operator is non-associative or non-commutative e.g., an accumulator can be implemented using one "partial accumulator" per processor core.

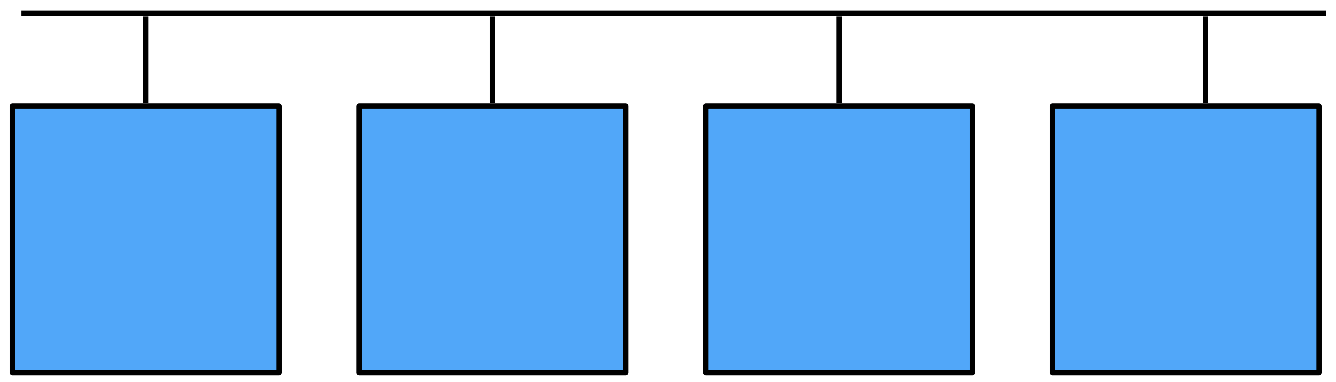
2) For each of the following functions, indicate if it is associative and/or commutative.
a) f(x,y) = x+y, for integers x, y, is associative and commutative
b) g(x,y) = (x+y)/2, for integers x, y, is commutative but not associative
c) h(s1,s2) = concat(s1, s2) for strings s1, s2, e.g., h("ab","cd") = "abcd", is associative but not commutative

Map/Reduce: Streaming data requirements have skyrocketed

- AT&T processes roughly 30 petabytes per day through its telecommunications network
- Facebook, Amazon, Twitter, etc, have comparable throughputs
- IBM Watson knowledge base stored roughly 4 terabytes of data when winning at Jeopardy

Parallelism enables processing of big data

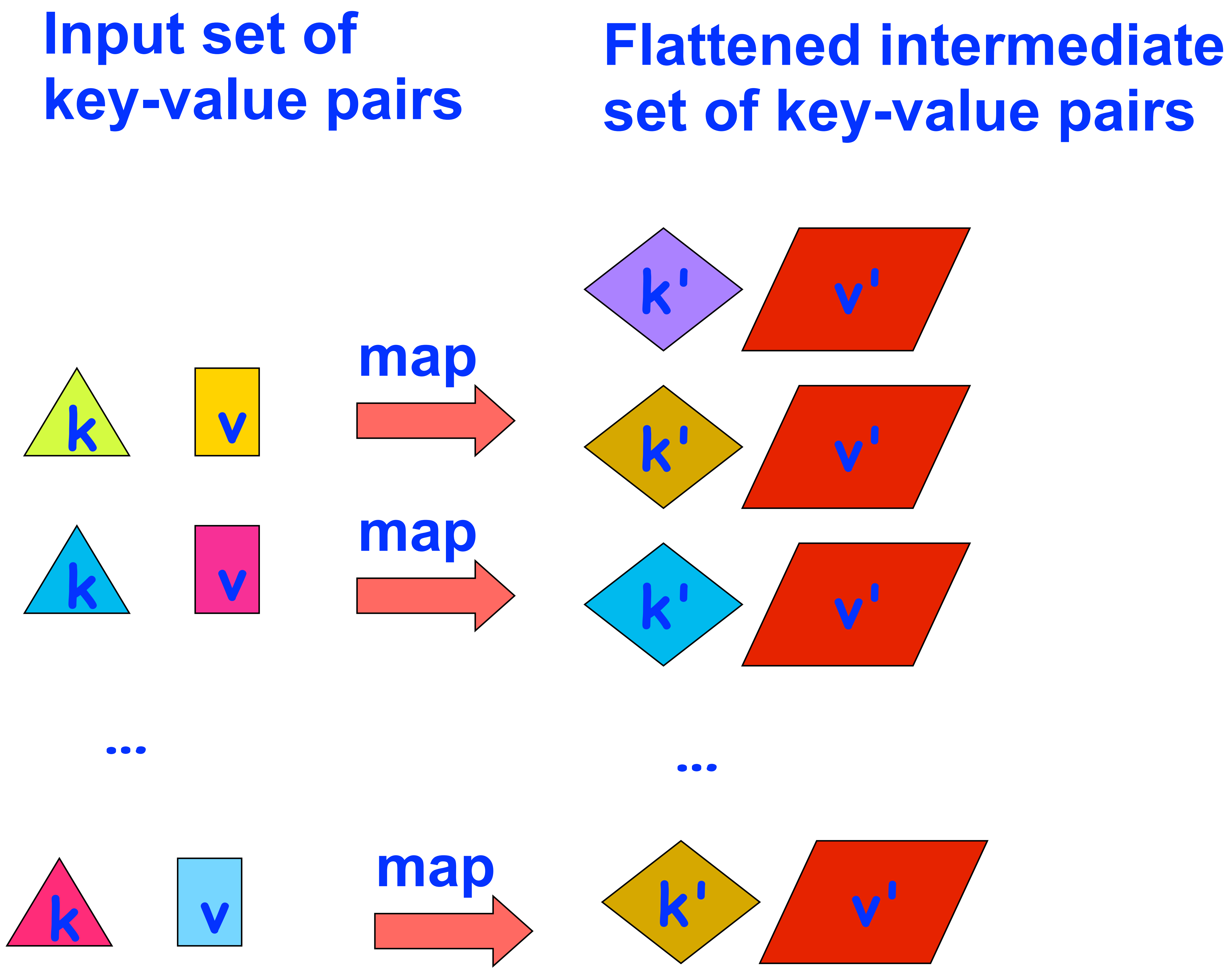
- Continuously streaming data needs to be processed at least as fast as it is accumulated, or we will never catch up
- The bottleneck in processing very large data sets is dominated by the speed of disk access
- More processors accessing more disks enables faster processing



MapReduce Pattern

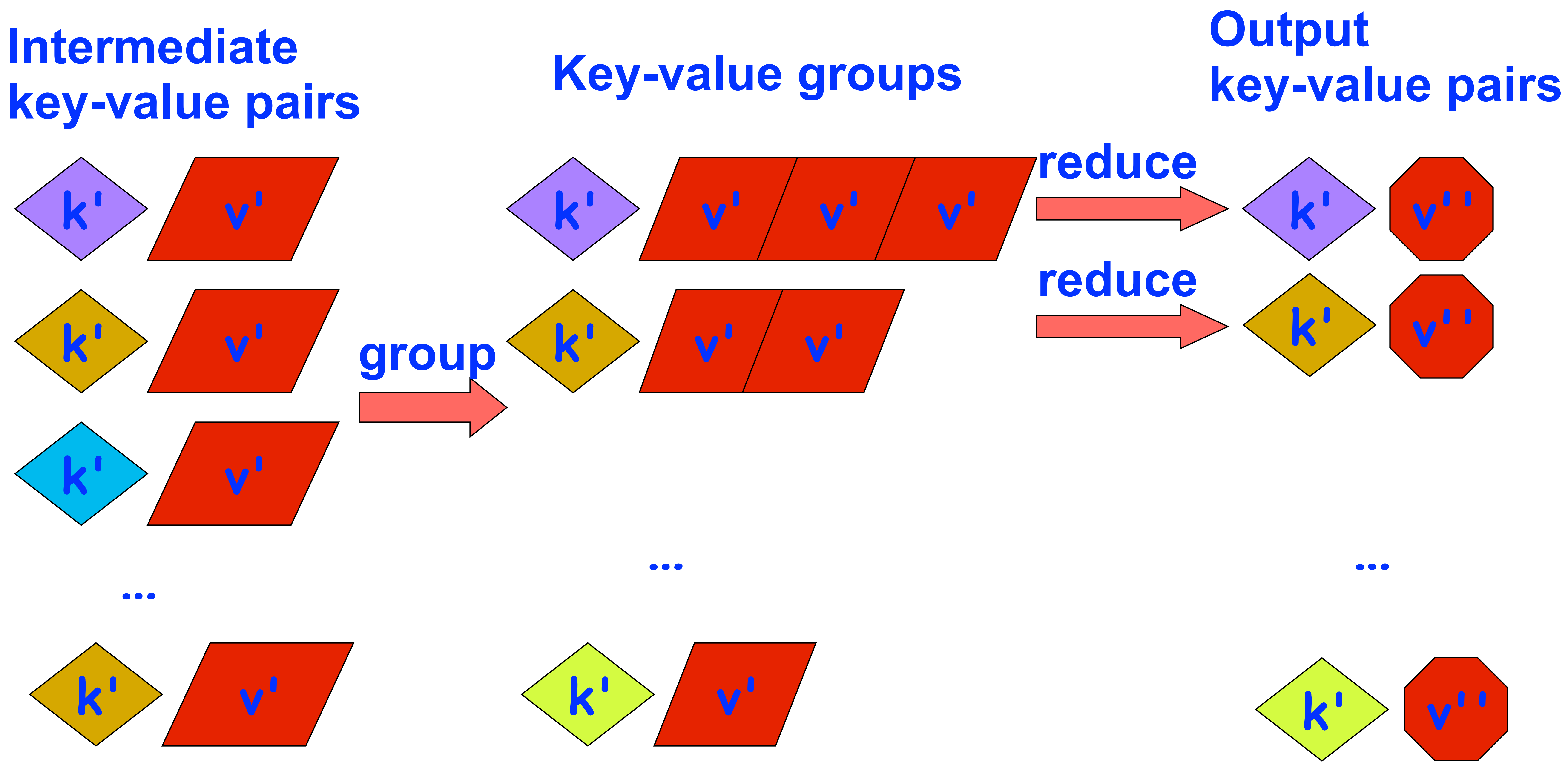
- Apply Map function f to user supplied record of key-value pairs
- Compute set of intermediate key/value pairs
- Apply Reduce operation g to all values that share same key to combine derived data properly
  — Often produces smaller set of values
- User supplies Map and Reduce operations in functional model so that the system can parallelize them, and also re-execute them for fault tolerance

MapReduce: Map Step



Source: http://infolab.stanford.edu/~ullman/mining/2009/mapreduce.ppt

MapReduce: Reduce Step



Source: http://infolab.stanford.edu/~ullman/mining/2009/mapreduce.ppt

Map Reduce: Summary

- Input set is of the form {(k1, v1), . . . (kn, vn)}, where (ki, vi) consists of a key, ki, and a value, vi.
  - Assume key and value objects are immutable

- Map function f generates sets of intermediate key-value pairs, f(ki,vi) = {(k1' ,v1'),...(km',vm')}. The km' keys can be different from ki key in the map function.
  - Assume that a flatten operation is performed as a post-pass after the map operations, so as to avoid dealing with a set of sets.

- Reduce operation groups together intermediate key-value pairs, {(k', vj')} with the same k’, and generates a reduced key-value pair, (k',v''), for each such k’, using reduce function g

Google Uses MapReduce

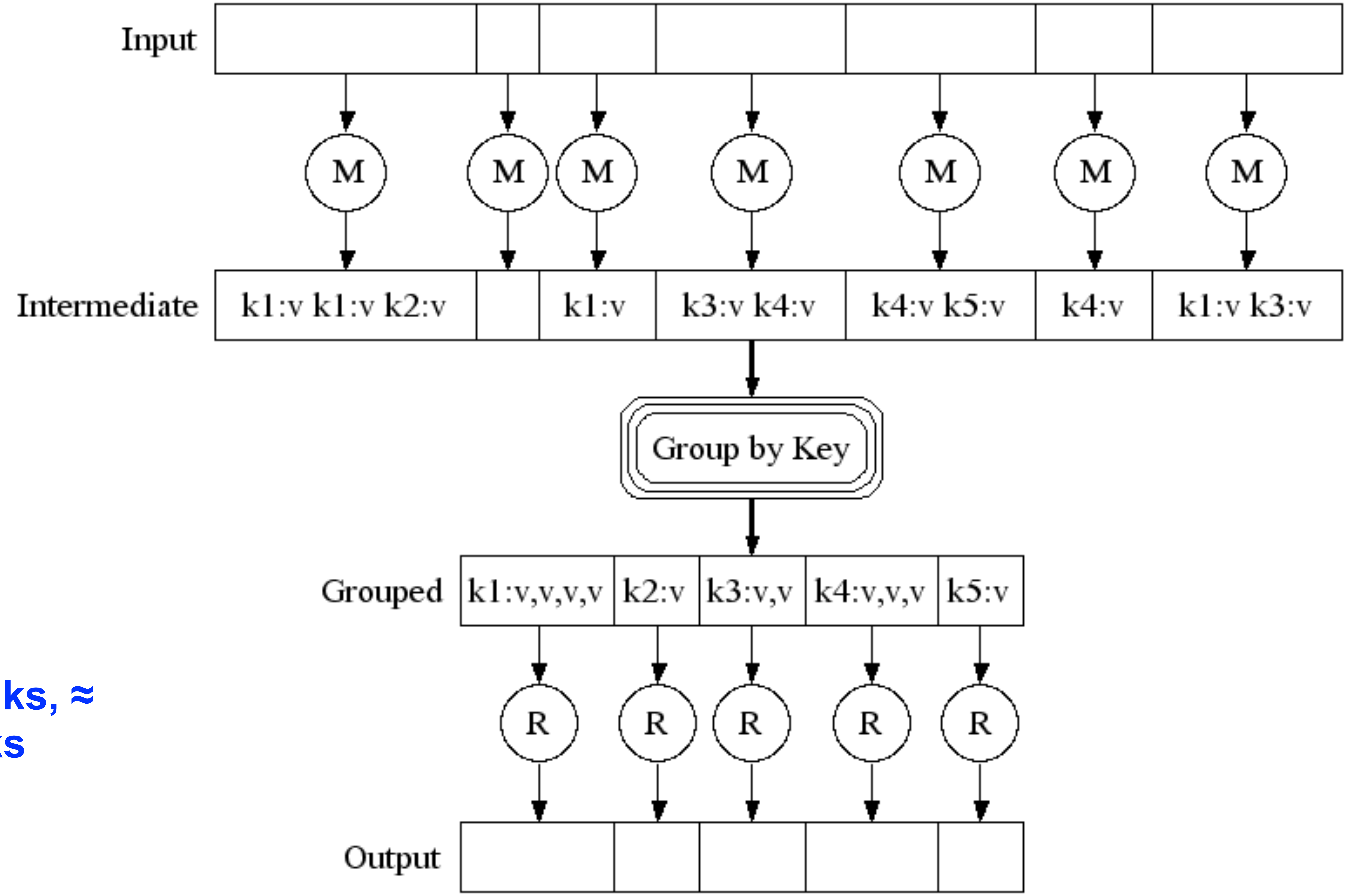
- Web crawl: Find outgoing links from HTML documents, aggregate by target document
- Google Earth: Stitching overlapping satellite images to remove seams and to select high-quality imagery
- Google Maps: Processing all road segments on Earth and render map tile images that display segments

MapReduce Execution

Fine granularity tasks: many more map tasks than machines

Bucket sort to get same keys together

2000 servers => ≈ 200,000 Map Tasks, ≈ 5,000 Reduce tasks



Word Count Example

In: set of words

Out: set of (word,count) pairs

Algorithm:

1. For each in word W, emit (W, 1) as a key-value pair (map step).
2. Group together all key-value pairs with the same key (reduce step).
3. Perform a sum reduction on all values with the same key(reduce step).

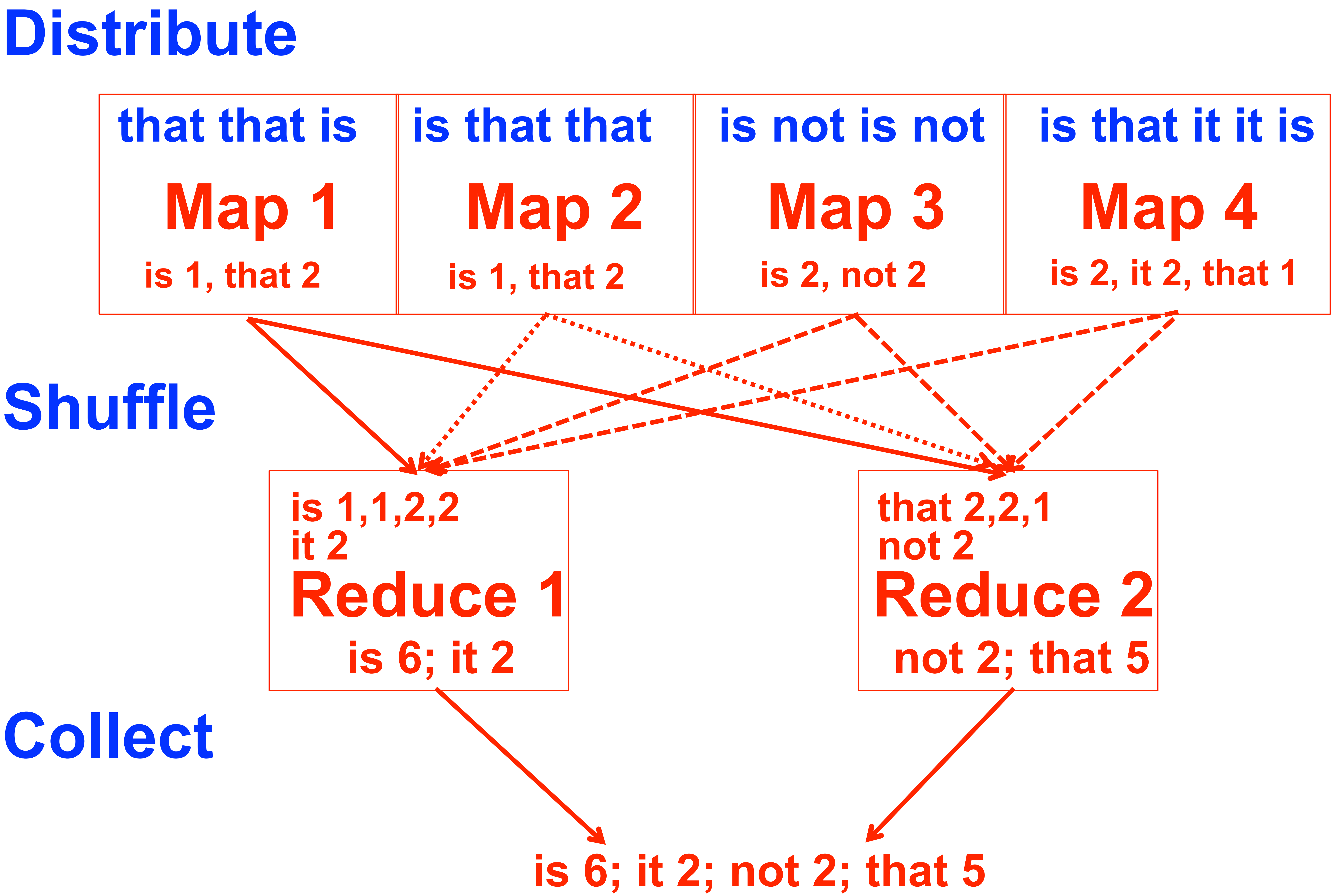
- All map operations in step 1 can execute in parallel with only local data accesses
- Step 2 may involve a major reshuffle of data as all key-value pairs with the same key are grouped together.
- Step 3 performs a standard reduction algorithm for all values with the same key, and in parallel for different keys.

Pseudocode for Word Count

```
<String, Integer> map(String inKey, String inValue):
  // inKey: document name
  // inValue: document contents
  for each word w in inValue:
    emitIntermediate(w, 1) // Produce count of words

<Integer> reduce(String outKey, Iterator<Integer> values):
  // outKey: a word
  // values: a list of counts
  Integer result = 0
  for each v in values:
    result += v // the value from map was an integer
  emit(result)
```

Example Execution of Word Count Program



Announcements & Reminders

- IMPORTANT:
  - —Watch video & read handout for topic 2.5 and 2.6 for Wednesday's lecture
- HW1 is due Wednesday, Feb 10th by 11:59pm
- Lab 2 is this week (Tu at 1:30pm, Th at 4:50pm)
- See Office Hours link on course web site for latest office hours schedule.

Worksheet #7: Analysis of Map Reduce Example

Analyze the total WORK and CPL for the Map reduce example:

- Assume that each Map step has WORK = number of input words, and CPL=1
- Assume that each Reduce step has WORK = number of input word-count pairs, and CPL = log2(# occurrences for input word with largest # pairs)