

COMP 322: Fundamentals of Parallel Programming

Lecture 7: Memoization and Map/Reduce

Mack Joyner and Zoran Budimlić
{mjoyner, zoran}@rice.edu

<http://comp322.rice.edu>



Worksheet #6 solution: 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) $h(s1,s2) = \text{concat}(s1, s2)$ for strings $s1, s2$, e.g., $h(\text{“ab”}, \text{“cd”}) = \text{“abcd”}$, **is associative but not commutative**



Background: Functional Programming

- Eliminate side-effects
 - emphasizes functions whose results that depend only on their inputs and not on any other program state
 - calling a function, $f(x)$, twice with the same value for the argument x will produce the same result both times

Helpful Link: http://en.wikipedia.org/wiki/Functional_programming



Example: Binomial Coefficient

- The coefficient of the x^k term in the polynomial expansion of the binomial power $(1 + x)^n$
- Number of sets of k items that can be chosen from n items
- Indexed by n and k
 - written as $C(n, k)$
 - read as “ n choose k ”
- Factorial Formula: $C(n, k) = \left(\frac{n!}{k!(n-k)!} \right)$
- Recursive Formula
$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$$

Base cases: $C(n, n) = C(n, 0) = C(0, k) = 1$

Helpful Link: http://en.wikipedia.org/wiki/Binomial_coefficient



Example: Binomial Coefficient (Recursive Sequential version)

```
1. int choose(int N, int K) {  
2.     if (N == 0 || K == 0 || N == K) {  
3.         return 1;  
4.     }  
5.     int left = choose (N-1, K - 1);  
6.     int right = choose (N-1, K);  
7.     return left + right;  
8. }
```



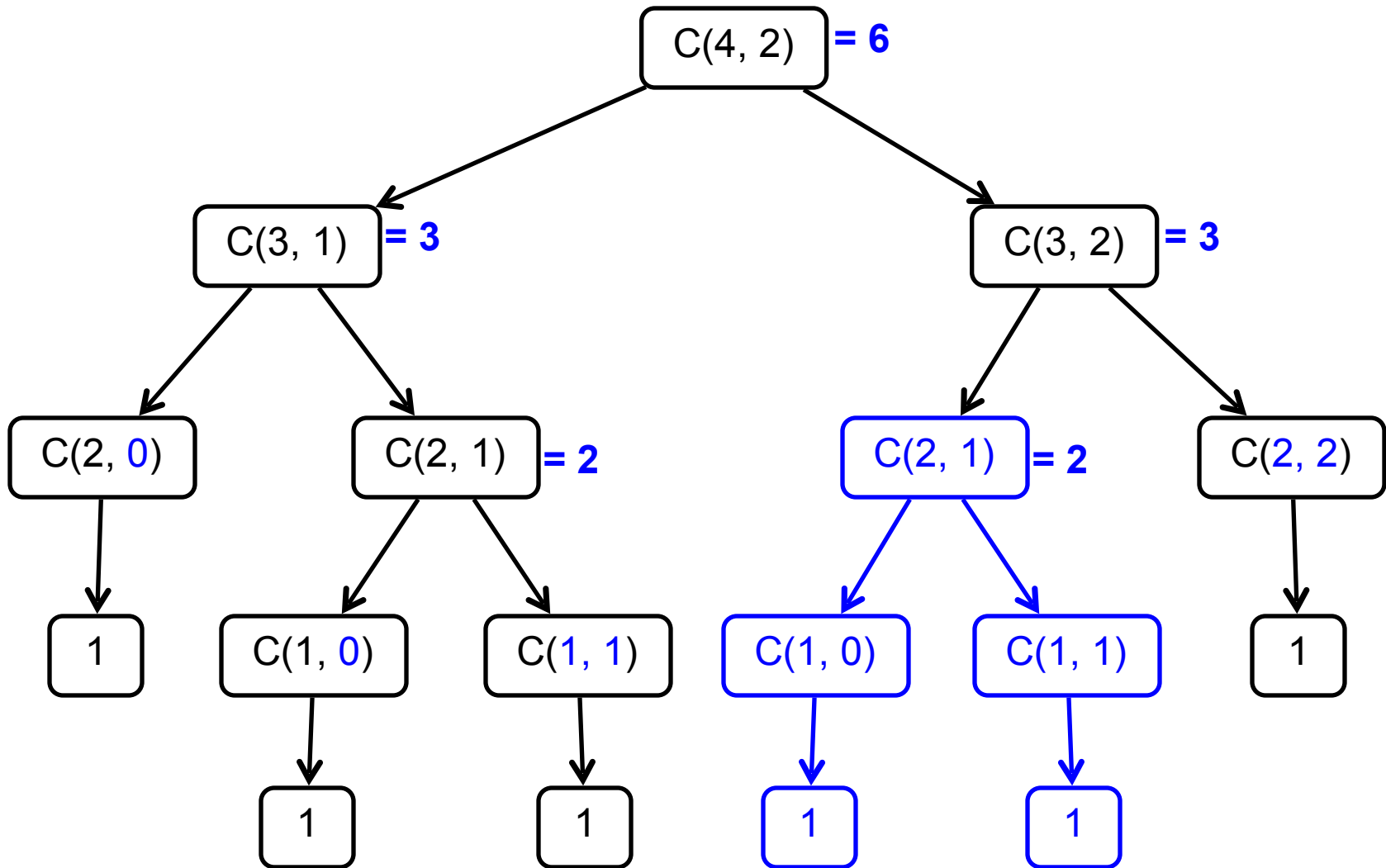
Example: Binomial Coefficient (Parallel Recursive Pseudocode)

```
1. Integer choose(int N, int K) {
2.     if (N == 0 || K == 0 || N == K) {
3.         return 1;
4.     }
5.     future<Integer> left =
6.         future { return choose (N-1, K-1); }
7.     future<Integer> right =
8.         future { return choose (N-1, K); }
9.     return left.get() + right.get();
10. }
```

- Use of futures supports incremental parallelization with low developer effort



What inefficiencies do you see in the recursive Binomial Coefficient algorithm?



Memoization

- Memoization - saving and reusing previously computed values of a function rather than recomputing them
 - A optimization technique with space-time tradeoff
- A function can only be memoized if it is *referentially transparent*, i.e. functional
- Related to caching
 - memoized function "remembers" the results corresponding to some set of specific inputs
 - memoized function populates its cache of results transparently on the fly, as needed, rather than in advance

Helpful Link: <http://en.wikipedia.org/wiki/Memoization>



Example: Binomial Coefficient (sequential memoized version)

```
1.  final Map<Pair<Int, Int>, Int> cache = new ...;

2.  int choose(int N, int K) {
3.      Pair<Int, Int> key = Pair.factory(N, K);
4.      if (cache.contains(key)) {
5.          return cache.get(key);
6.      }
7.      if (N == 0 || K == 0 || N == K) {
8.          return 1;
9.      }
10.     int left = choose (N - 1, K - 1);
11.     int right = choose (N - 1, K);
12.     int result = left + right;
13.     cache.put(key, result);
14.     return result;
15. }
```



Example: Binomial Coefficient (parallel memoized version w/ futures)

1. final Map<Pair<Int, Int>, future<Integer>> cache = new ...;
 2. Integer choose(final int N, final int K) {
 3. final Pair<Int, Int> key = Pair.factory(N, K);
 4. if (cache.contains(key)) {
 5. return cache.get(key).get();
 6. }
 7. future<Integer> f = future {
 8. if (N == 0 || K == 0 || N == K) return 1;
 9. future<Integer> left = future { return choose (N-1, K-1); }
 10. future<Integer> right = future { return choose (N-1, K); }
 12. return left.get() + right.get();
 13. }
 14. cache.put(key, f);
 15. return f.get();
 16. }
- Assumes availability of a “thread-safe” cache library, e.g., ConcurrentHashMap



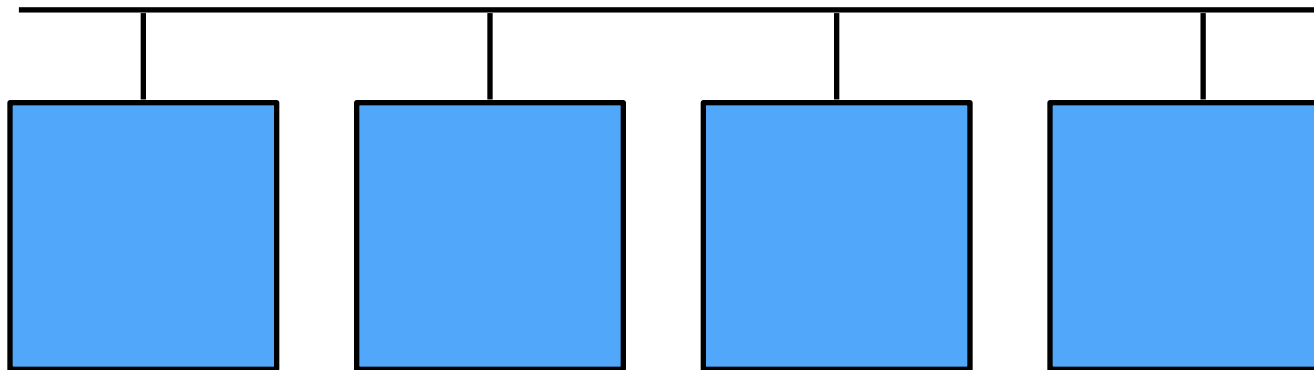
Map/Reduce: Streaming data requirements have skyrocketed

- **AT&T processes roughly 30 petabytes per day through its telecommunications network**
- **Google processed roughly 24 petabytes per day in 2009**
- **Facebook, Amazon, Twitter, etc, have comparable throughputs**
- **Two Sigma maintains over 100 teraflops of private computing power, continuously computing over 11 petabytes of quantitative data**
- **In comparison, the 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

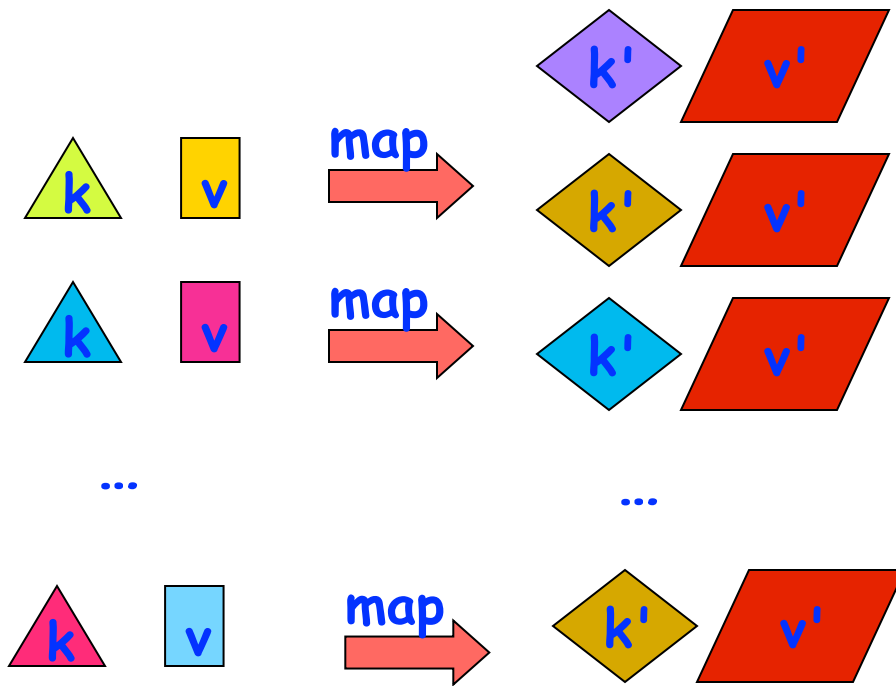
- Apply **Map** function **f** to user supplied record of key-value pairs
- Compute set of intermediate key/value pairs
- Apply **Reduce** operation **g** to all values that share same key to combine derived data properly
 - **Often produces smaller set of values**
- User supplies Map and Reduce operations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



MapReduce: The Map Step

Input set of
key-value pairs

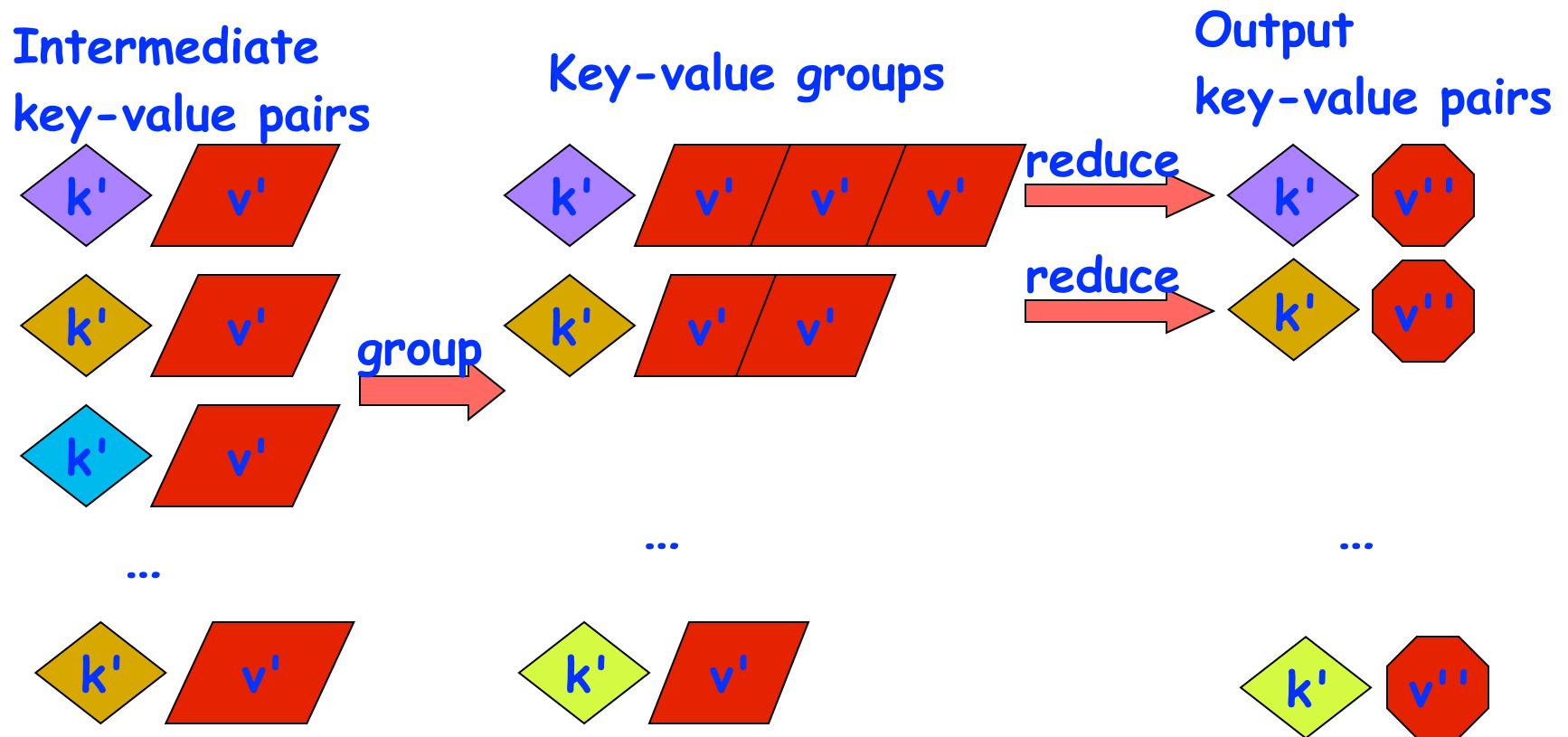
Flattened intermediate
set of key-value pairs



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



MapReduce: The Reduce Step



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



Map Reduce: Summary

- Input set is of the form $\{(k_1, v_1), \dots, (k_n, v_n)\}$, where (k_i, v_i) consists of a key, k_i , and a value, v_i .
 - Assume that the key and value objects are immutable, and that equality comparison is well defined on all key objects.
- Map function f generates sets of intermediate key-value pairs, $f(k_i, v_i) = \{(k_1', v_1'), \dots, (k_{m'}', v_{m'}')\}$. The $k_{m'}'$ keys can be different from k_i 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, $\{(k', v_j'')\}$ with the same k' , and generates a reduced key-value pair, (k', v'') , for each such k' , using reduce function g



Google Uses MapReduce For ...

- **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

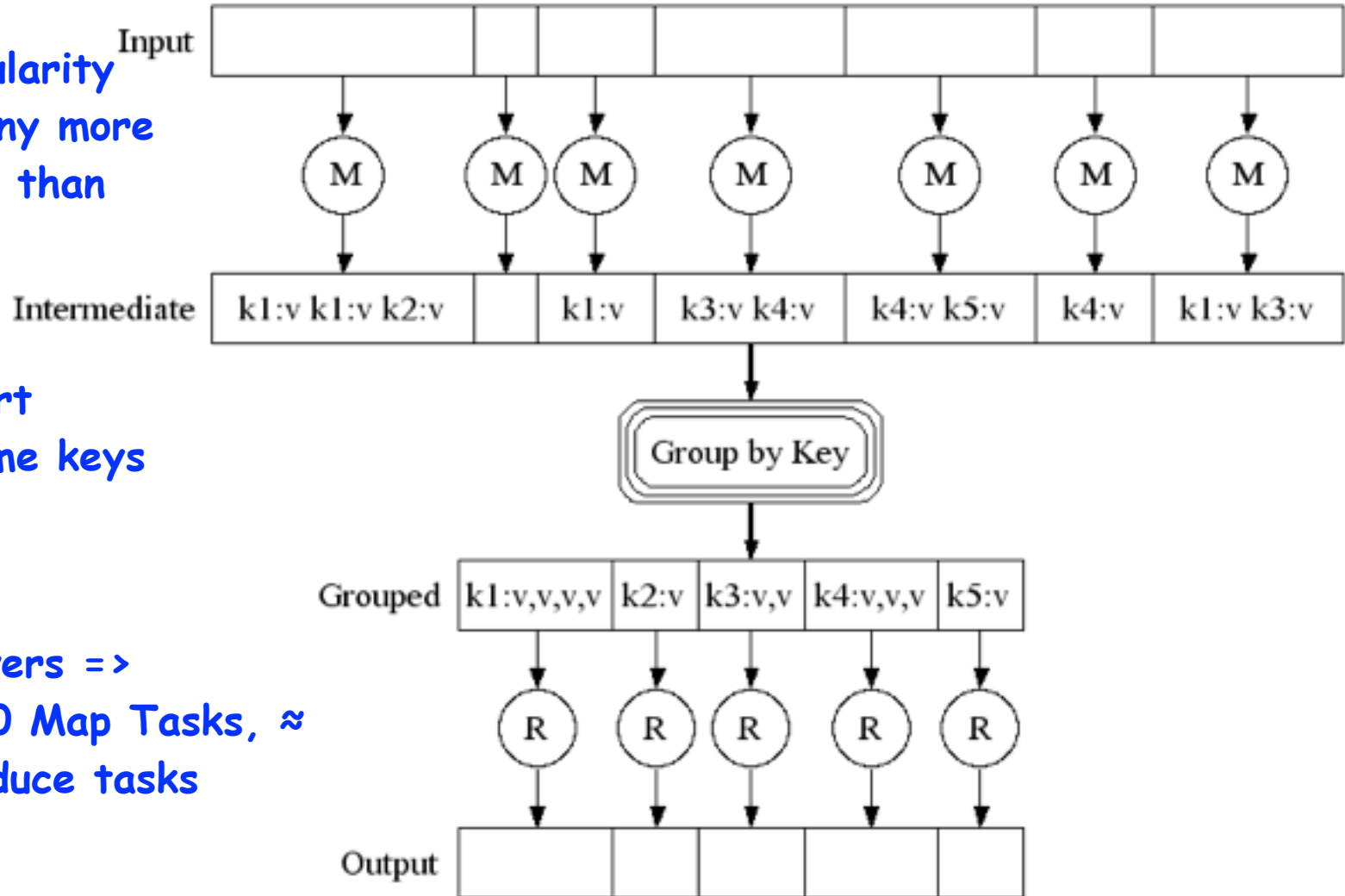


MapReduce Execution

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



WordCount 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.**



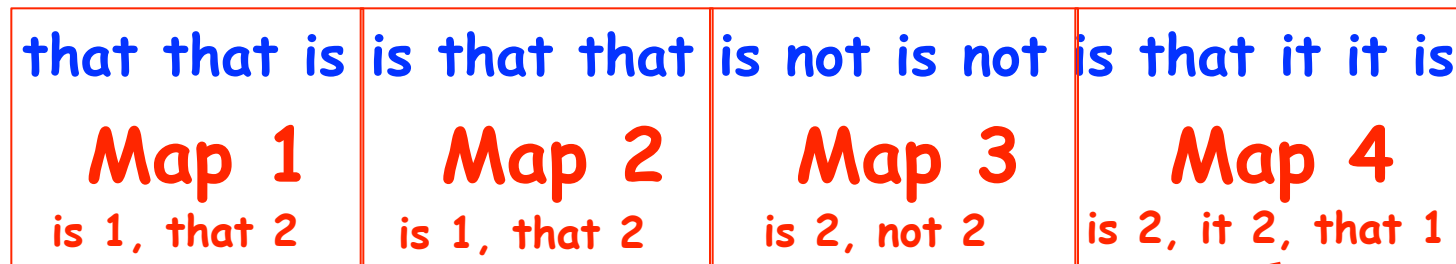
PseudoCode for WordCount

```
1. <String, Integer> map(String inKey, String inValue):
2.     // inKey: document name
3.     // inValue: document contents
4.     for each word w in inValue:
5.         emitIntermediate(w, 1) // Produce count of words
6.
7. <Integer> reduce(String outKey, Iterator<Integer> values):
8.     // outKey: a word
9.     // values: a list of counts
10.    Integer result = 0
11.    for each v in values:
12.        result += v // the value from map was an integer
13.    emit(result)
```



Example Execution of WordCount Program

Distribute



Shuffle



Collect

is 6; it 2; not 2; that 5



Announcements & Reminders

- **IMPORTANT:**
 - Watch video & read handout for topic 2.5 and 2.6 for next lecture on Monday, Jan 29th**
- **HW2 is available and due by Wednesday, Feb 7th**
- **Quiz for Unit 1 (topics 1.1 - 1.5) is due by 11:59pm **TODAY** on Canvas**
- **See course web site for all work assignments and due dates**
- **Use Piazza (public or private posts, as appropriate) for all communications re. COMP 322**
- **See Office Hours link on course web site for latest office hours schedule.**

