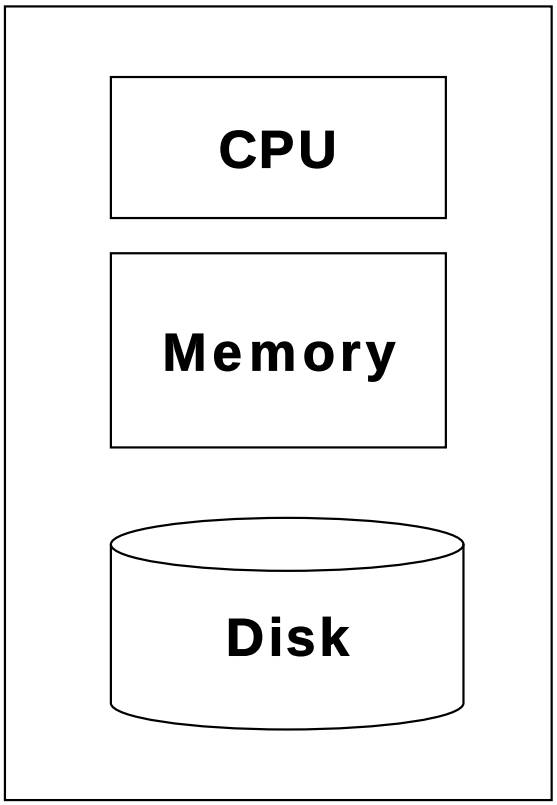
Data Mining and Information Retrieval

MapReduce

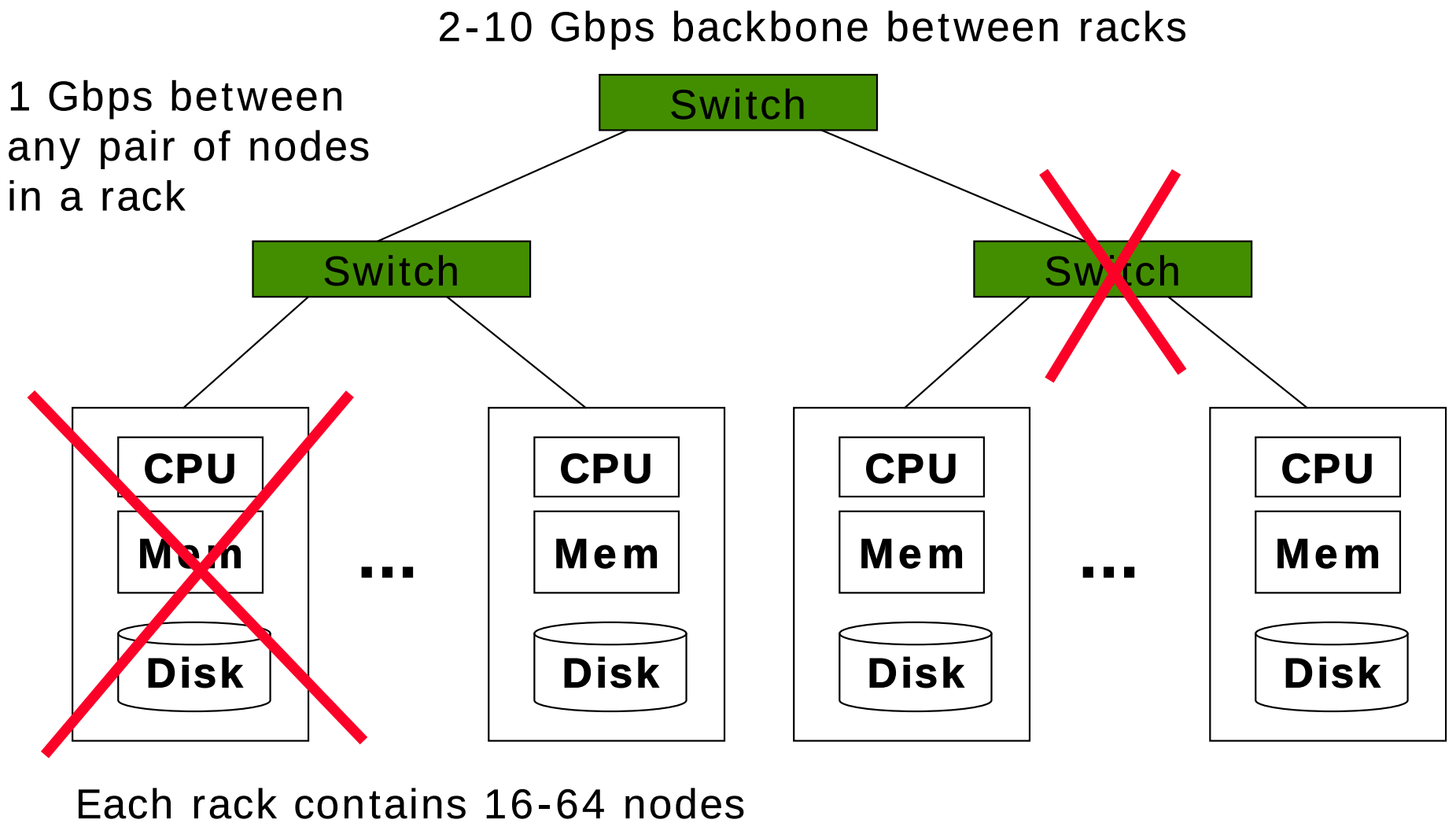
Single-node architecture



Machine Learning, Statistics

“Classical” Data Mining

Cluster Architecture



Stable storage

- First order problem: if nodes can fail, how can we store data persistently?
- Answer: Distributed File System
  - Provides global file namespace
  - Google GFS; Hadoop HDFS; Kosmix KFS
- Typical usage pattern
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- Chunk Servers
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- Master node
  - a.k.a. Name Nodes in HDFS
  - Stores metadata
  - Might be replicated
- Client library for file access
  - Talks to master to find chunk servers
  - Connects directly to chunkservers to access data

Warm up: Word Count

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Sample application: analyze web server logs to find popular URLs

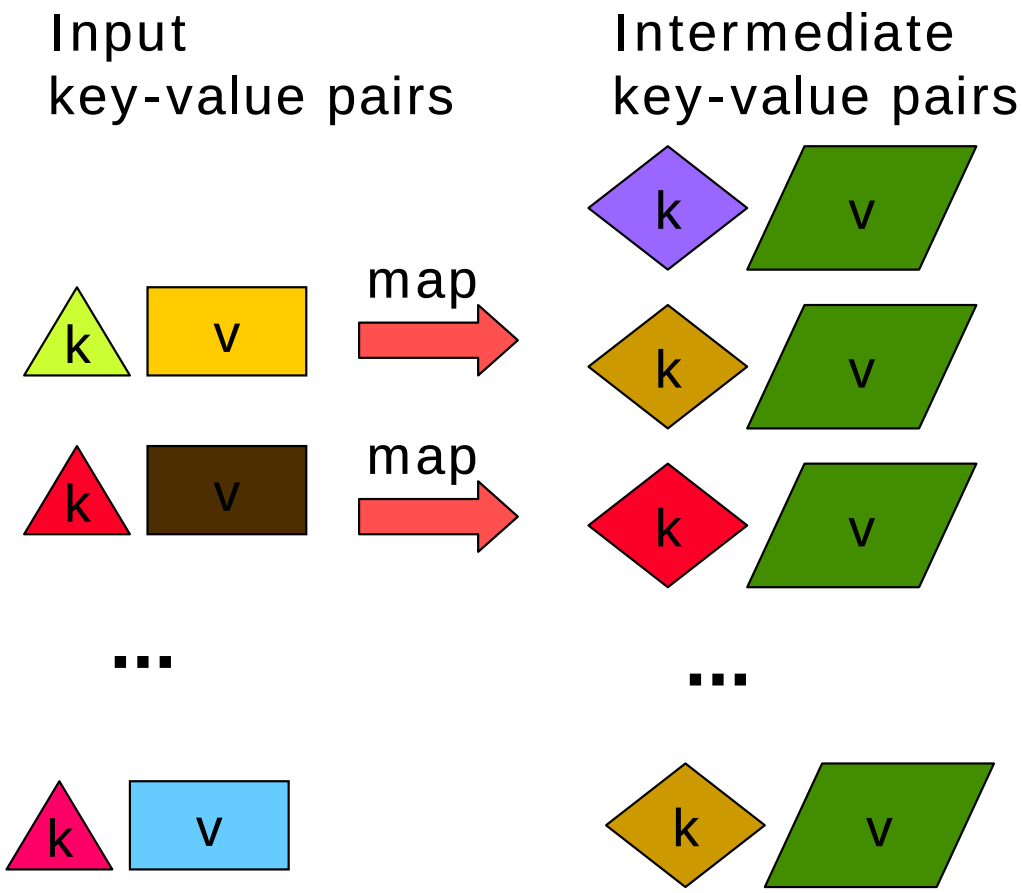
Word Count (2)

- Case 1: Entire file fits in memory
- Case 2: File too large for mem, but all <word, count> pairs fit in mem
- Case 3: File on disk, too many distinct words to fit in memory
  - `sort datafile | uniq -c`

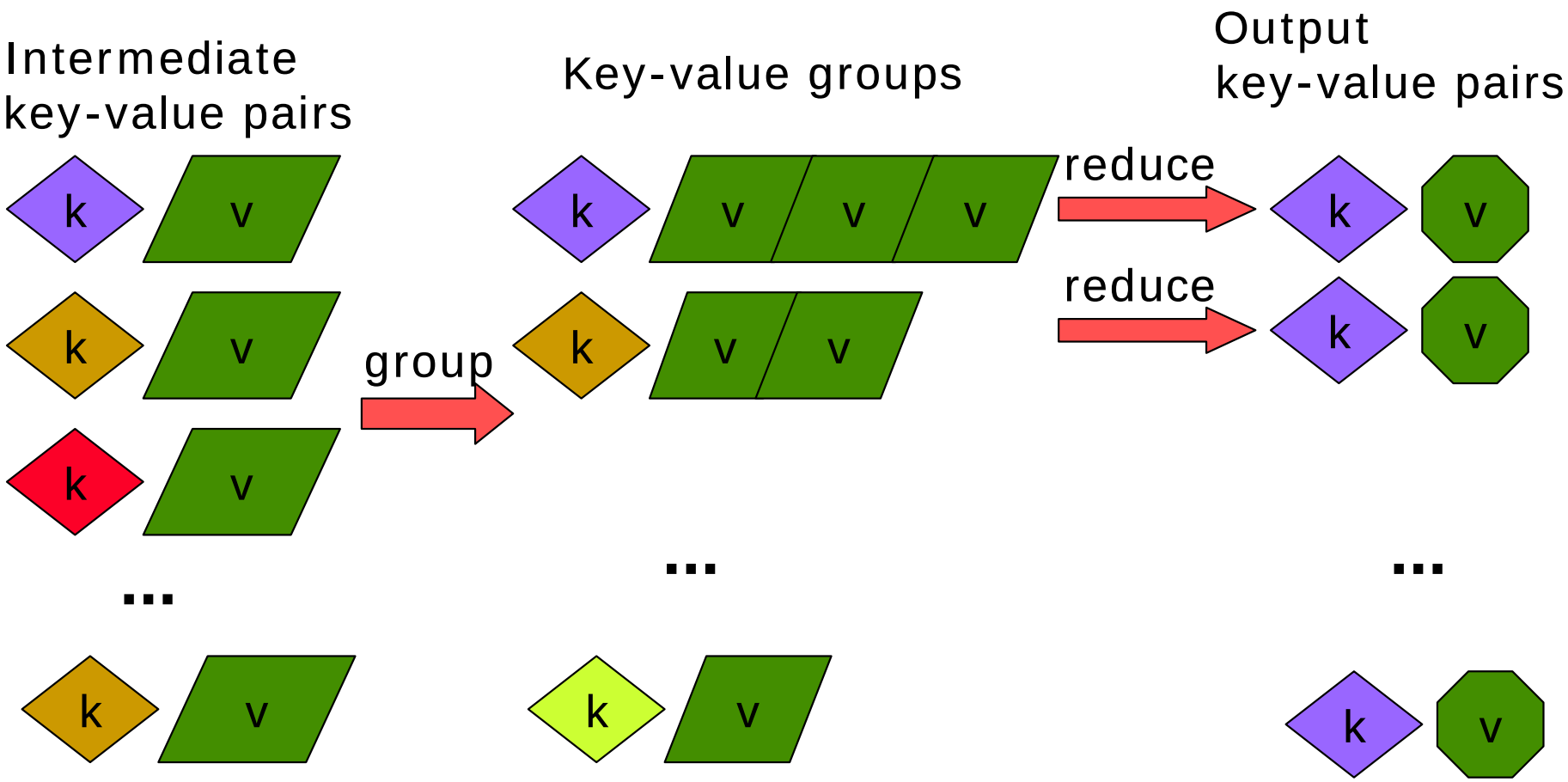
Word Count (3)

- To make it slightly harder, suppose we have a large corpus of documents
- Count the number of times each distinct word occurs in the corpus
  - **`words(docs/*) | sort | uniq -c`**
  - where **`words`** takes a file and outputs the words in it, one to a line
- The above captures the essence of MapReduce
  - Great thing is it is naturally parallelizable

MapReduce: The Map Step



MapReduce: The Reduce Step



MapReduce

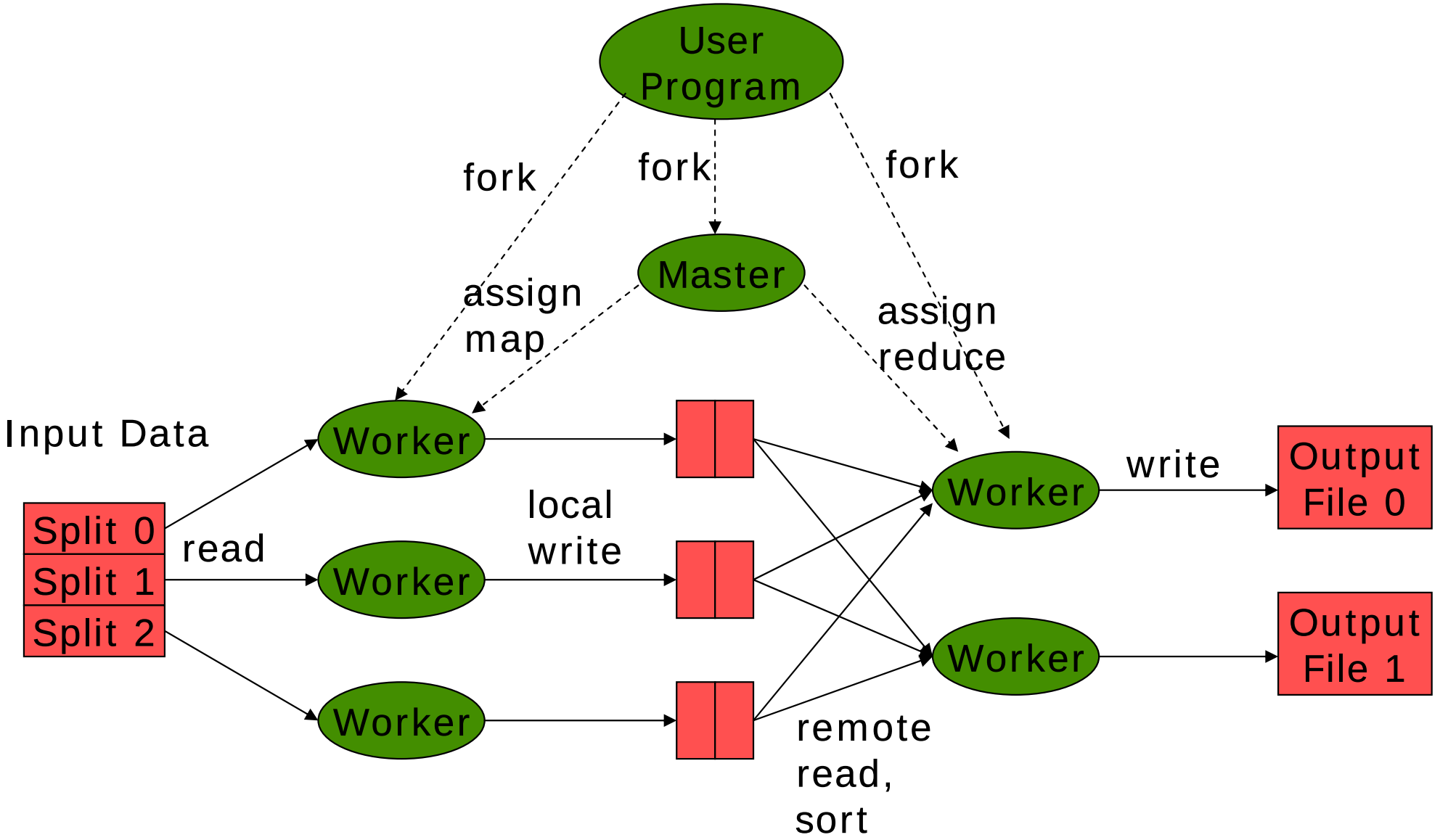
- Input: a set of key/value pairs
- User supplies two functions:
  - map(k,v) → list(k1,v1)
  - reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- Output is the set of (k1,v2) pairs

Word Count using MapReduce

```
map(key, value):
// key: document name; value: text of document
    for each word w in value:
        emit(w, 1)
```

```
reduce(key, values):
// key: a word; value: an iterator over counts
        result = 0
        for each count v in values:
            result += v
        emit(result)
```

Distributed Execution Overview



Exercise: Frequent Pairs

- Given a large set of market baskets, find all frequent pairs
  - Remember definitions from Association Rules lectures

Implementations

- Google
  - Not available outside Google
- Hadoop
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/
- Aster Data
  - Cluster-optimized SQL Database that also implements MapReduce

Reading

- Jeffrey Dean and Sanjay Ghemawat,

  MapReduce: Simplified Data Processing on Large Clusters

http://labs.google.com/papers/mapreduce.html

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, **The Google File System**

http://labs.google.com/papers/gfs.html