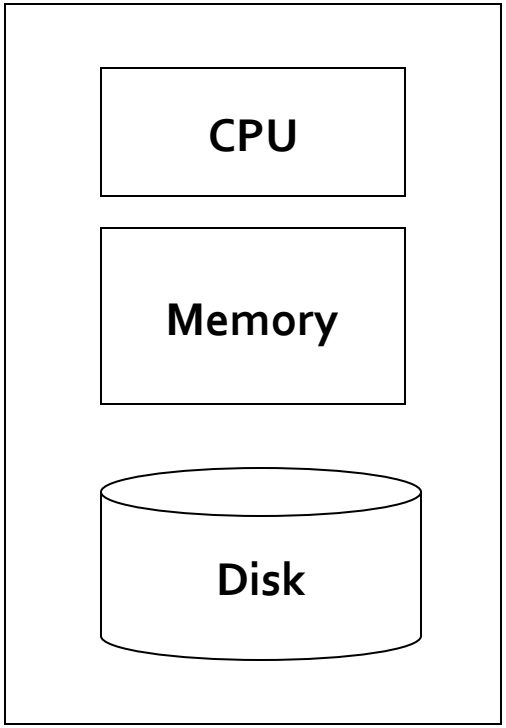
Map-Reduce

Distributed File System
Computational Model
Scheduling and Data Flow
Refinements

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

Single Node Architecture



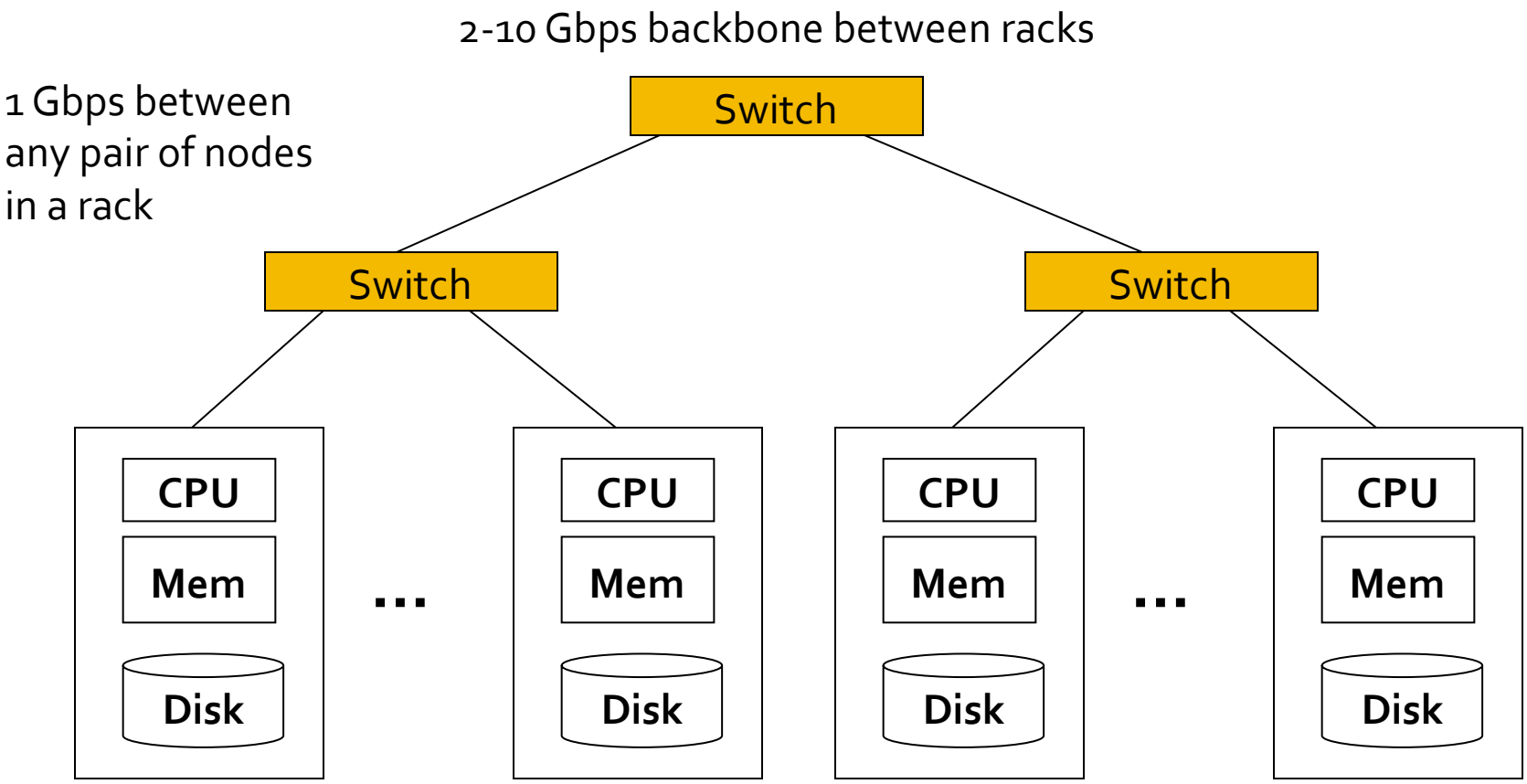
Machine Learning, Statistics

"Classical" Data Mining

Motivation: Google Example

- 10 billion web pages
- Average size of webpage = 20KB
- 10 billion * 20KB = 200 TB
- Disk read bandwidth = 50 MB/sec
- Time to read = 4 million seconds = 46+ days
- Even longer to do something useful with the data

Cluster Architecture



Each rack contains 16-64 commodity Linux nodes

In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

Cluster Computing Challenges (1)

- Node failures
  - A single server can stay up for 3 years (1000 days)
  - 1000 servers in cluster => 1 failure/day
  - 1M servers in cluster => 1000 failures/day
- How to store data persistently and keep it available if nodes can fail?
- How to deal with node failures during a long-running computation?

Cluster Computing Challenges (2)

- Network bottleneck
  - Network bandwidth = 1 Gbps
  - Moving 10TB takes approximately 1 day

- Distributed programming is hard!
  - Need a simple model that hides most of the complexity

Map-Reduce

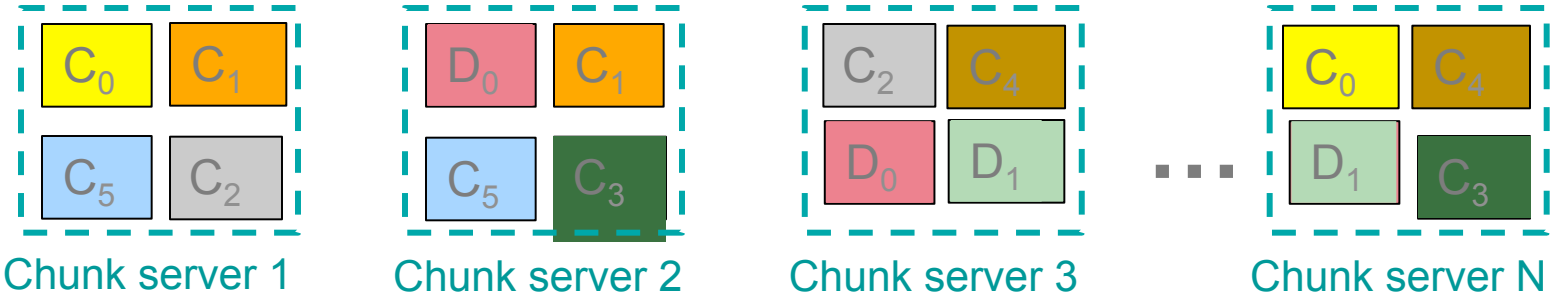
- Map-Reduce addresses the challenges of cluster computing
  - Store data redundantly on multiple nodes for persistence and availability
  - Move computation close to data to minimize data movement
  - Simple programming model to hide the complexity of all this magic

Redundant Storage Infrastructure

- **Distributed File System**
  - Provides global file namespace, redundancy, and availability
  - E.g., Google GFS; Hadoop HDFS
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- Data kept in "chunks" spread across machines
- Each chunk replicated on different machines
  - Ensures persistence and availability



Chunk servers also serve as compute servers

Bring computation to data!

Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks (16-64MB)
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Map-Reduce

Computational Model Examples

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

Programming Model: MapReduce

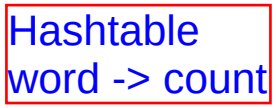
Warm-up task:

- We have a huge text document
- Count the number of times each distinct word appears in the file
- Sample applications
  - Analyze web server logs to find popular URLs
  - Term statistics for search

Task: Word Count

Case 1:

- File too large for memory, but all <word, count> pairs fit in memory



Word Count (2)

Case 2:

- Even the <word,count> pairs don't fit in memory

- `words(doc.txt) | sort | uniq -c`
  - where **`words`** takes a file and outputs the words in it, one per a line

- Case 2 captures the essence of MapReduce
  - Great thing is that it is naturally parallelizable

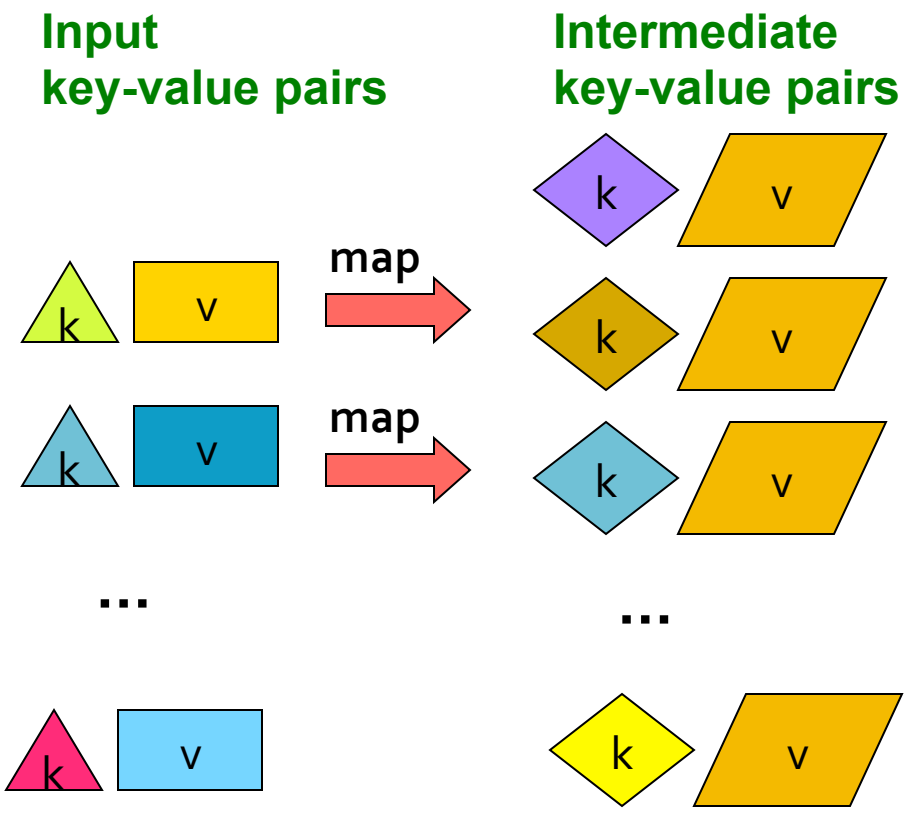
MapReduce: Overview

```
words(doc.txt) | sort | uniq -c
```

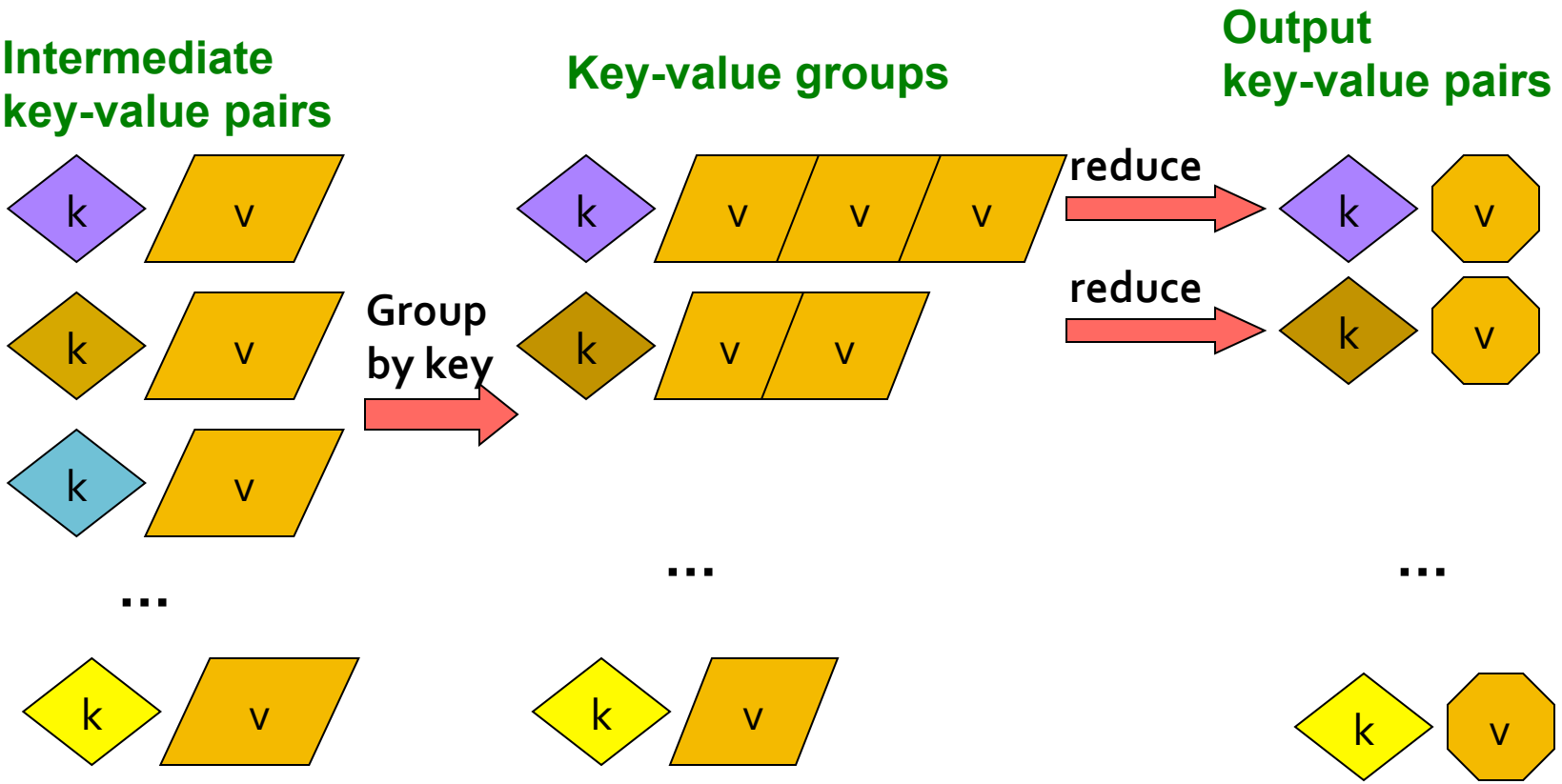
- **Map**
  - Scan input file record-at-a-time
  - Extract something you care about from each record (keys)
- **Group by key**
  - Sort and Shuffle
- **Reduce**
  - Aggregate, summarize, filter or transform
  - Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

MapReduce: The Map Step



MapReduce: The Reduce Step



More formally…

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** → <k', v'>*
    - Takes a key-value pair and outputs a set of key-value pairs
    - There is one Map call for every *(k,v)* pair
  - **Reduce(k', <v'>*)** → <k', v''>*
    - **All values *v'* with same key *k'* are reduced together**
    - There is one Reduce function call per unique key *k'*

e: Word Co

请注意，这个只搞sequential reading or access的，不搞random的.sequential 效率比较高

Copy to single node
也可以是多个，此处是三个。一样的key 从不一样的map node跑到了一样的reduce node，因为hashtable最后从client访问是一样的，系统会根据hashtable 自动从这reduce nodes中access data

Intermediate key-value function

可分成四个chunks，Map这个功能就直接在chunk上运行。也可以理解成在4个nodes上运行

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

Only sequential reads

Big document | **(key, value)** | **(key, value)** | **(key, value)**

Word Count Using MapReduce

```
map(key, value):
// key: document name; value: text of the 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(key, result)
```

Example: Host size

- Suppose we have a large web corpus with a metadata file formatted as follows:
  - Each record of the form: (URL, size, date, …)
- For each host, find the total number of bytes

- Map
  - For each record, output (hostname(URL), size)
- Reduce
  - Sum the sizes for each host

Example: Language Model

- Count number of times each 5-word sequence occurs in a large corpus of documents
- Map
  - Extract (5-word sequence, count) from document
- Reduce
  - Combine the counts

Map-Reduce

Scheduling and Data Flow

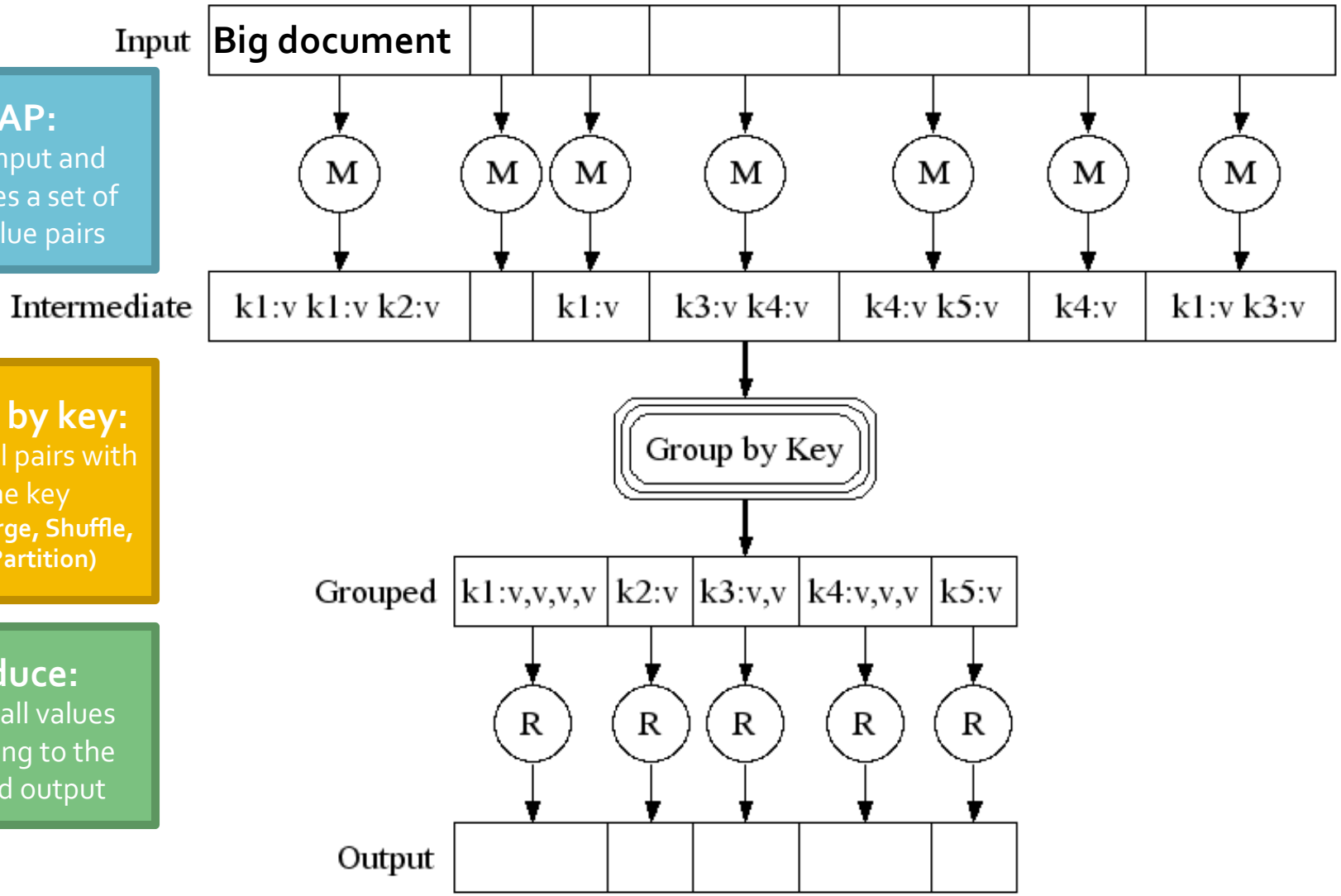
Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

Map-Reduce: A diagram

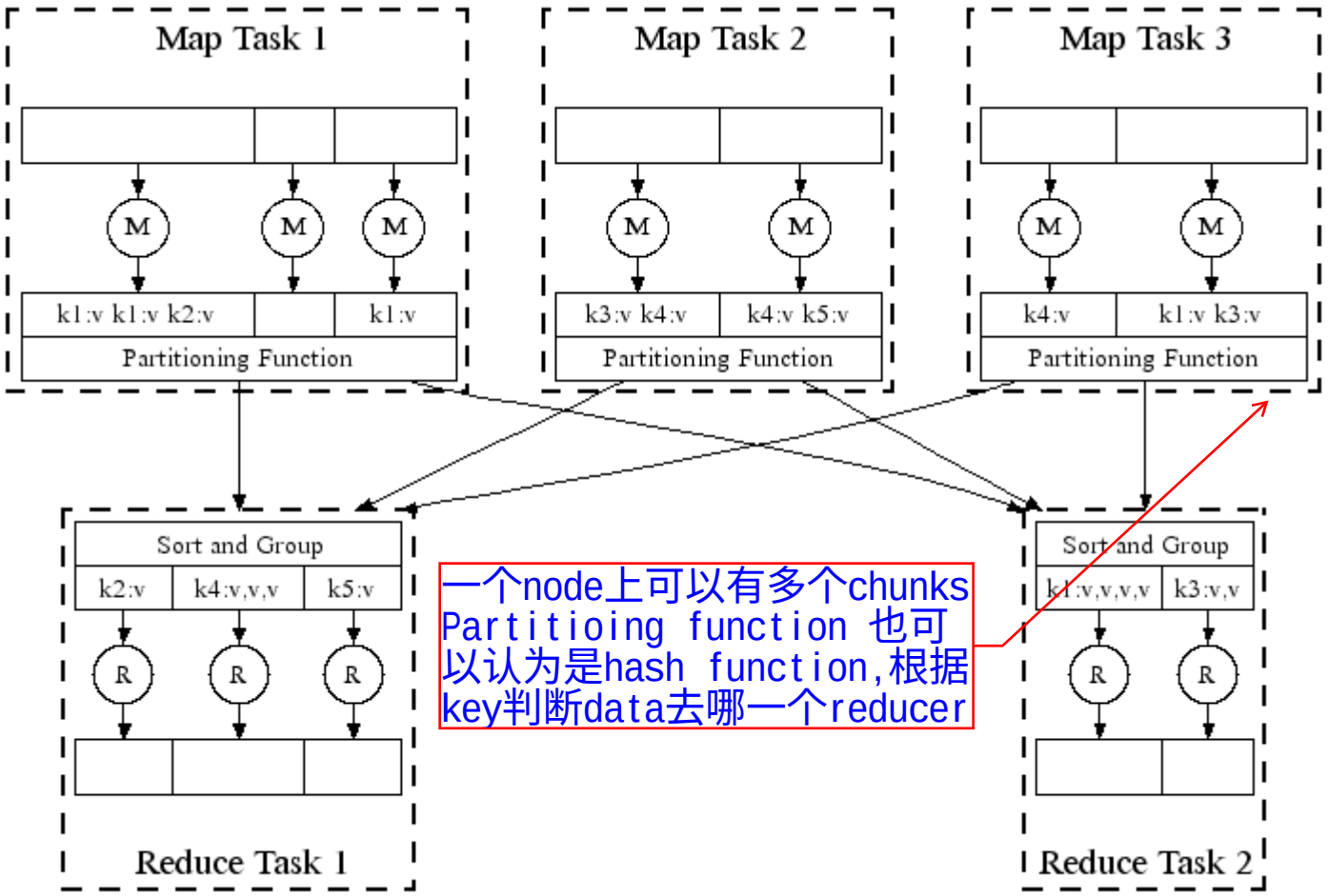
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

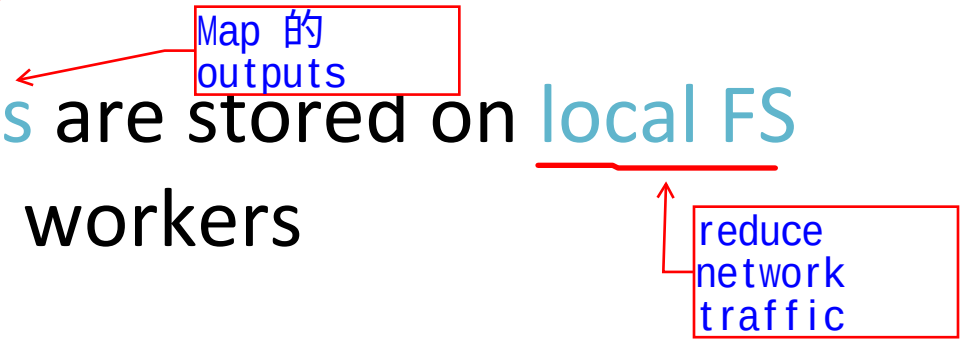
Map-Reduce: Environment

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling node failures
- Managing required inter-machine communication

Data Flow

- Input and final output are stored on the distributed file system (DFS):
  - 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 MapReduce task



Coordination: Master

- **Master node takes care of coordination:**
  - **Task status:** (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

Dealing with Failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Idle tasks eventually rescheduled on other worker(s)
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Idle Reduce tasks restarted on other worker(s)
- **Master failure**
  - MapReduce task is aborted and client is notified

因为reduce的output是最终的output已经被写到DFS里面保存了

通常比较少挂掉，但是也要注意

How many Map and Reduce jobs?

- *M* map tasks, *R* reduce tasks
- **Rule of thumb:**
  - Make *M* much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures

一个node 有好多chunk servers

- **Usually *R* is smaller than *M***
  - Because output is spread across *R* files

Map-Reduce

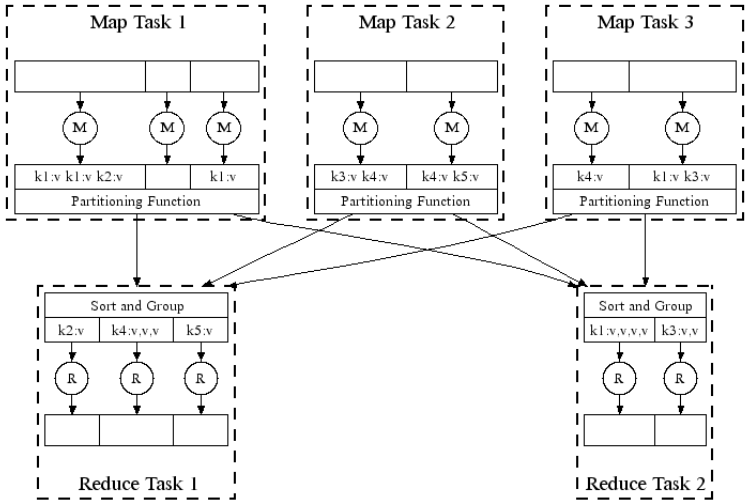
Refinements
Implementations

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

Refinement: Combiners (1)

- Often a Map task will produce many pairs of the form $(k,v_1), (k,v_2), \ldots$ for the same key k
  - E.g., popular words in the word count example

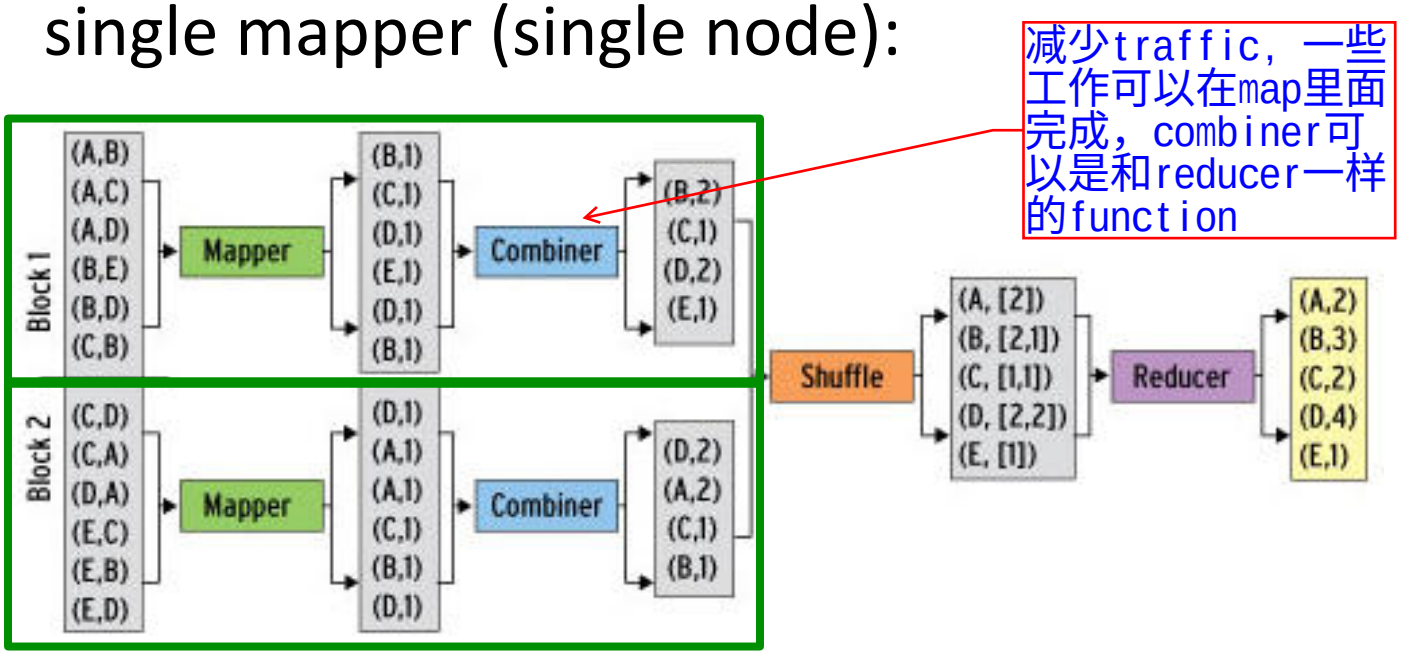
- **Can save network time by pre-aggregating values in the mapper:**
  - combine(k, list(v_1)) → v_2
  - Combiner is usually same as the reduce function



减少traffic，一些工作可以在map里面完成

Refinement: Combiners (2)

- **Back to our word counting example:**
  - Combiner combines the values of all keys of a single mapper (single node):



  - Much less data needs to be copied and shuffled!

Refinement: Combiners (3)

- Combiner trick works only if reduce function is commutative and associative
- Sum

满足交换律和结合律

- Average

这个可以记录count和sum，修改一下即可满足交换律和结合律

- Median

这个不行，中位数

Refinement: Partition Function

- **Want to control how keys get partitioned**
  - The set of keys that go to a single reduce worker
- **System uses a default partition function:**
  - **hash(key) mod *R***
- **Sometimes useful to override the hash function:**
  - E.g., **hash(hostname(URL)) mod *R*** ensures URLs from a host end up in the same output file

Implementations

- Google MapReduce
  - Uses Google File System (GFS) for stable storage
  - Not available outside Google
- Hadoop
  - Open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/
- Hive, Pig
  - Provide SQL-like abstractions on top of Hadoop Map-Reduce layer

Cloud Computing

- Ability to rent computing by the hour
  - Additional services e.g., persistent storage
- E.g., Amazon's "Elastic Compute Cloud" (EC2)
  - S3 (stable storage)
  - Elastic Map Reduce (EMR)

Pointers and Further Reading

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

Resources

- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/

Resources

- Releases from Apache download mirrors
  - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
- Nightly builds of source
  - http://people.apache.org/dist/lucene/hadoop/nightly/
- Source code from subversion
  - http://lucene.apache.org/hadoop/version_control.html