# Introduction to Parallel & Distributed Computing

# Parallel Programming with MapReduce

Lecture 15, Spring 2024

Instructor: 罗国杰

gluo@pku.edu.cn

# Overview

- **What is MapReduce?**
- **System support**
  - Hadoop/Google file system
  - (Schedule), Partition, Shuffle/Sort
  - Fault tolerance
- **MapReduce examples**

# Motivations


- **Motivations**
  - Large-scale data processing on clusters
  - Massively parallel (hundreds or thousands of CPUs)
  - Reliable execution with easy data access
- **Functions**
  - Automatic parallelization & distribution
  - Fault-tolerance
  - Status and monitoring tools
  - A clean abstraction for programmers
    - Functional programming meets distributed computing
    - A batch data processing system

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Parallel Data Processing in a Cluster

- **Scalability to large data volumes:**
  - Scan 1000 TB on 1 node @ 100 MB/s = 24 days
  - Scan on 1000-node cluster = 35 minutes

- **Cost-efficiency:**
  - Commodity nodes /network
    - Cheap, but not high bandwidth, sometime unreliable
  - Automatic fault-tolerance (fewer admins)
  - Easy to use (fewer programmers)

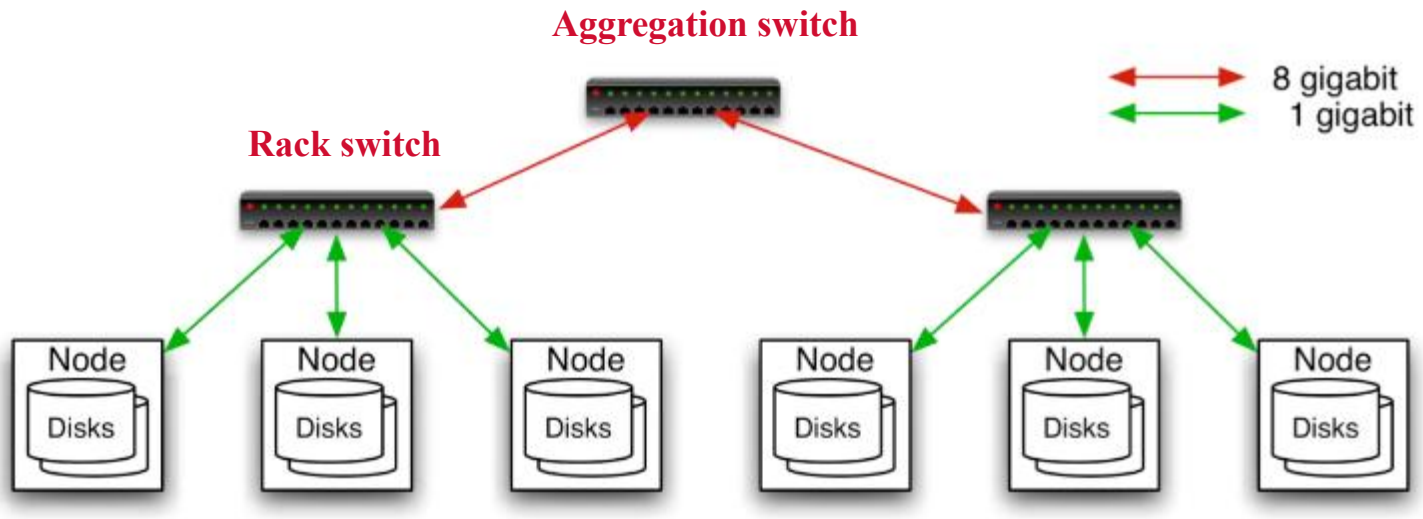
Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce and Hadoop

- **MapReduce**
  - Practice in Google
    - parallelize the computation
    - distribute the data
    - handle failures conspire
  - J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," OSDI'04.
- **Hadoop**
  - An implementation of the MapReduce idea

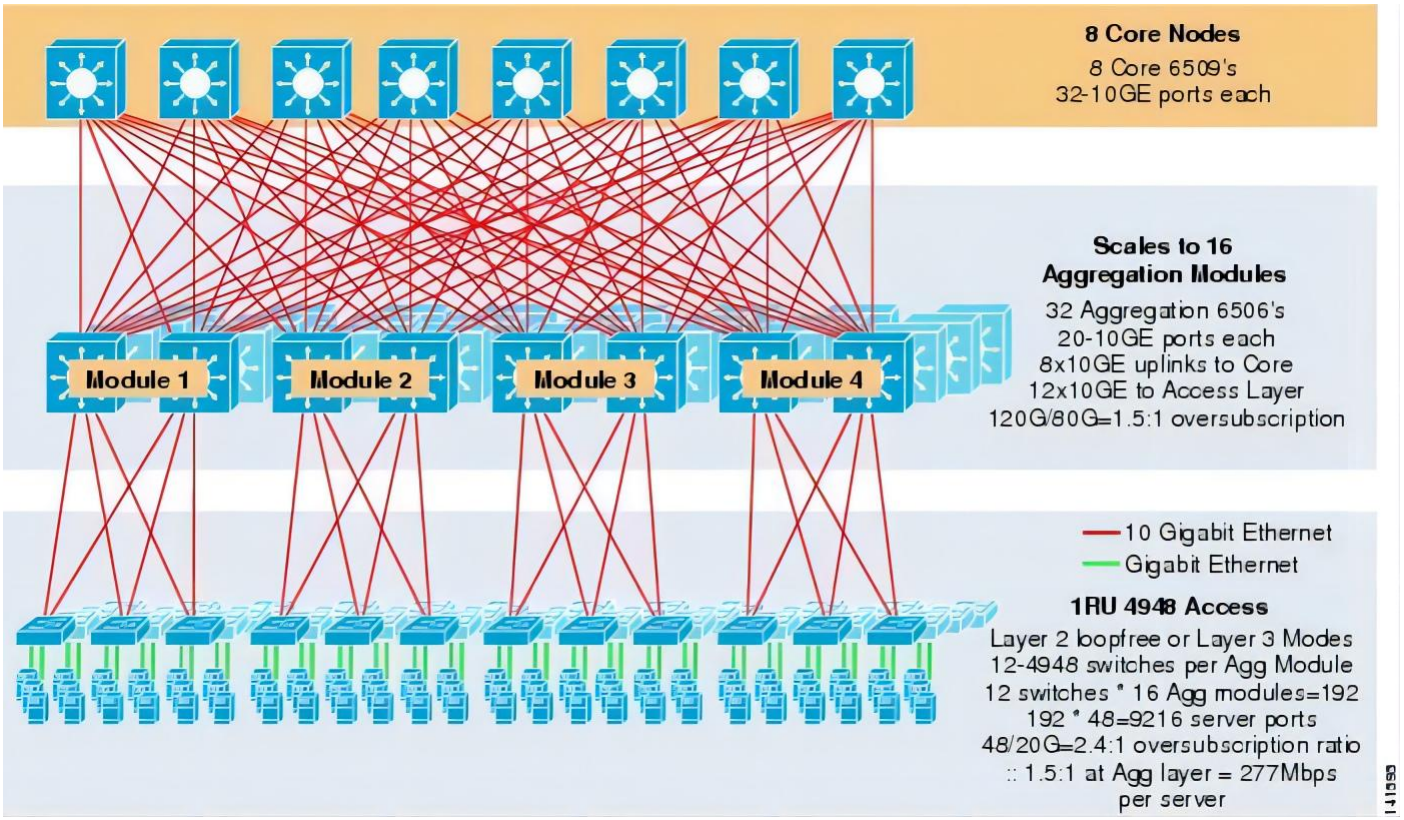
# Typical Hadoop Cluster (Facebook ~2010)

- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth in rack, 8 Gbps out of rack
- Node specs:
  - 8-16 cores, 32 GB RAM, 8 × 1.5 TB disks



# Layered Network Architecture in Conventional Data Centers

- **A layered example from Cisco: core, aggregation, the edge or top-of-rack switch.**
  - http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCInfra_3a.html



# Map and Reduce in Programming Languages

- **Inspired from map and reduce operations commonly used in functional programming languages like Lisp.**
  - `(map square '(1 2 3 4))  ; (1 4 9 16)`
  - `(reduce + '(1 4 9 16))  ; 30`

- **In Python3**

```
map(lambda x: x**2, [1, 2, 3, 4])
from functools import reduce
reduce(lambda x,y: x+y, [1, 4, 9, 16])
```

- **In Scala**

```
Seq(1,2,3,4).map(x => x*x)
Seq(1,4,9,16).reduce(_+_)
```

# Related Theories for Map and Reduce

- **A few data-parallel functionals**
  - $map\ f\ [x_1, x_2, \cdots, x_n] = [fx_1, fx_2, \cdots, fx_3]$
  - $red(\oplus)[x_1, x_2, \cdots, x_n] = x_1 \oplus x_2 \oplus \cdots \oplus x_n$
  - $scan(\oplus)[x_1, x_2, \cdots, x_n] = [x_1,\ x_1 \oplus x_2,\ \cdots,\ x_1 \oplus \cdots \oplus x_n]$
- **Functional composition**
  - $(f \circ g)x = f(gx)$

Gorlatch, S., Cole, M. (2011). Parallel Skeletons. In: Encyclopedia of Parallel Computing.

# Related Theories for Map and Reduce

- **List homomorphism**
  - A function $h$ on lists is called a homomorphism with combine operation $\otimes$, if and only if for arbitrary lists $x, y$:
  - $h(x \bowtie y) = (hx) \otimes (hy)$
- **Theorem: Factorization**
  - A function $h$ on lists is a homomorphism with combine operation $\otimes$, if and only if it can be factorised as follows:
  - $h = red(\otimes) \circ map\ \phi$, where $\phi a = h[a]$
  - ($h$ can computed in two stages: $map\ \phi$ and $red(\otimes)$)

Gorlatch, S., Cole, M. (2011). Parallel Skeletons. In: Encyclopedia of Parallel Computing.

# Related Theories for Map and Reduce

- **Theorem: Promotion**
  - If $h$ is a homomorphism with combine operation $\otimes$, then
  - $h \circ red(\bowtie) = red(\otimes) \circ map\ h$
- **Homomorphism partition rule (HPR)**
  - $(red(\otimes) \circ map\ \phi) \circ red(\bowtie) =$
  - $red(\otimes) \circ map(red(\otimes) \circ map\ \phi)$
  - Useful for parallelization via data partitioning

Gorlatch, S., Cole, M. (2011). Parallel Skeletons. In: Encyclopedia of Parallel Computing.

# Map-Reduce Model

- **Data: (key, value) pairs**
  - All data is stored as key/value paires
  - Initially stored on some shared disk
    - e.g. GFS (Google File System), HDFS (Hadoop FS)
  - During the computaiton, route (key/value) pairs to different servers to perform the computation

S. Gilbert, "Map-Reduce (MPC) Algorithms", NUS CS5234: Algorithm at Scale, Fall 2019.

# Map-Reduce Model

- **Basic round**
  - **Map**: process each (key, value) pair
  - **Shuffle**: group items by key
  - **Reduce**: process items with same key together

- **Plan:**
  - Load data from disk
  - Execute several rounds
  - Save (key, value) pairs, sorted by key

S. Gilbert, "Map-Reduce (MPC) Algorithms", NUS CS5234: Algorithm at Scale, Fall 2019.

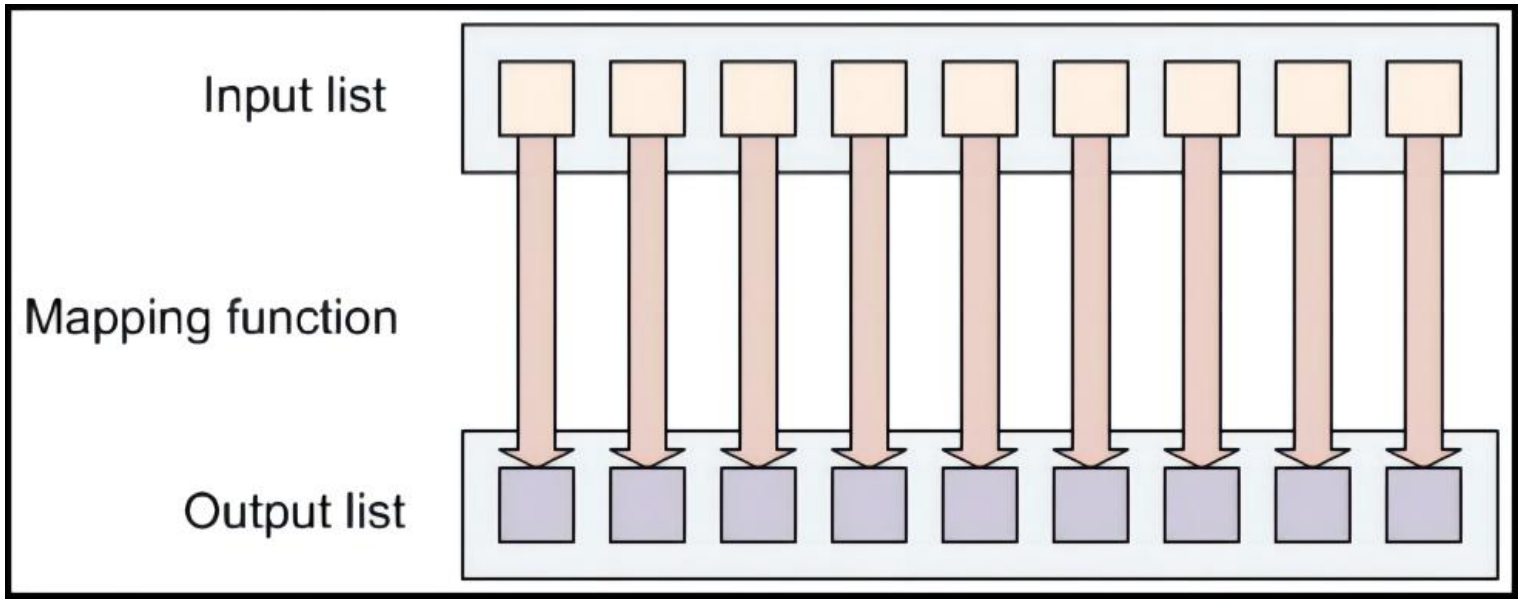
# Example: Map Processing in Hadoop

- **Given a file**
  - A file may be divided into multiple parts (splits).
- **Each record (line) is processed by a Map function,**
  - written by the user,
  - takes an input key/value pair
  - produces a set of intermediate key/value pairs.
  - e.g. (doc-id, doc-content)
- **Draw an analogy to SQL group-by clause**

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Map

```
map  (in_key, in_value) ->
(out_key, intermediate_value) list
```




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

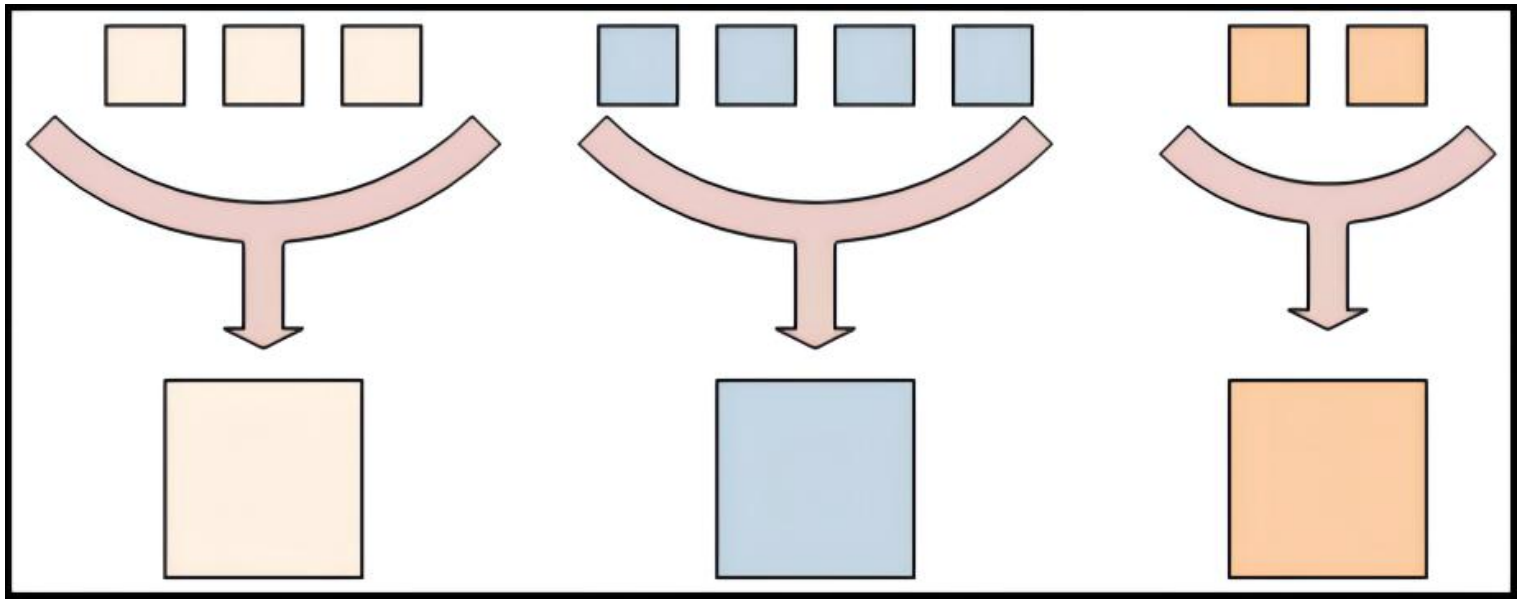
# Processing of Reducer Tasks

- **Given a set of (key, value) records produced by map tasks.**
  - all the intermediate values for a given output key are combined together into a list and given to a reducer.
  - Each reducer  further performs (key2, [val2]) → [val3]
- **Can be visualized as *aggregate* function (e.g., average) that is computed over all the rows with the same group-by attribute.**

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

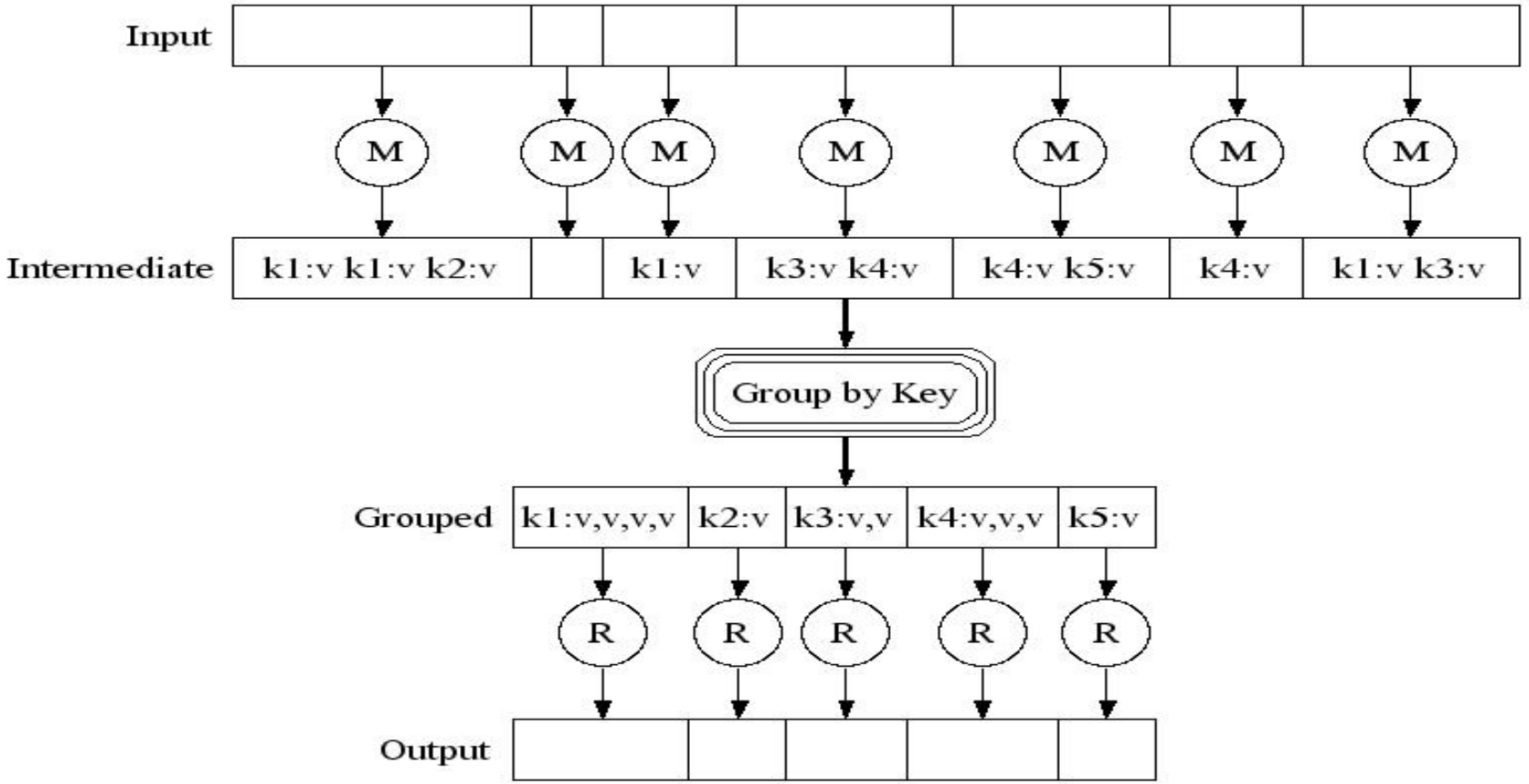
# Reduce

```
reduce (out_key, intermediate_value list) ->
        out_value list
```



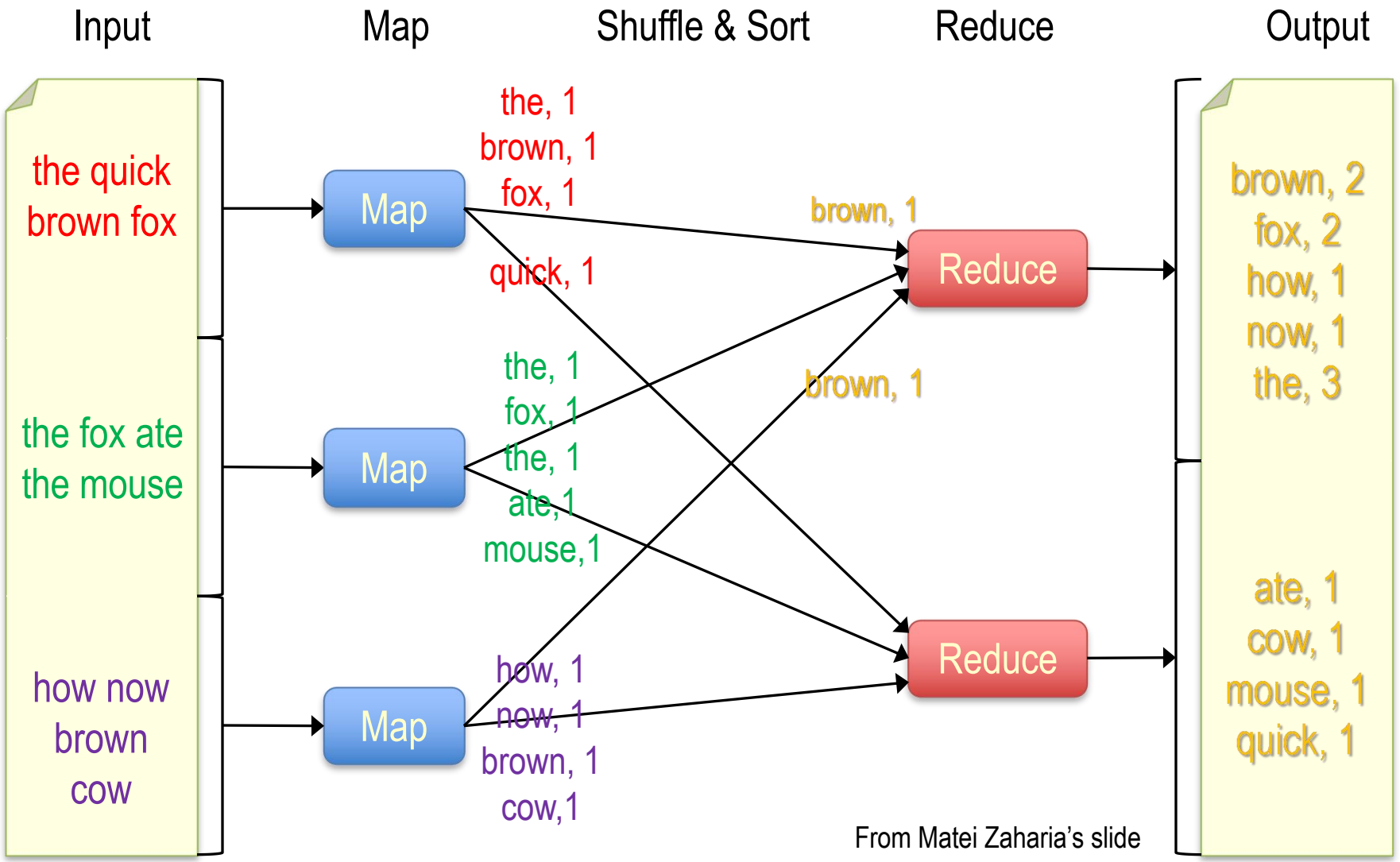
Tao Yang, “CS 240A: Applied Parallel Computing,” UCSB, Winter 2013.

# Put Map and Reduce Tasks Together




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Example: Word Count Execution




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

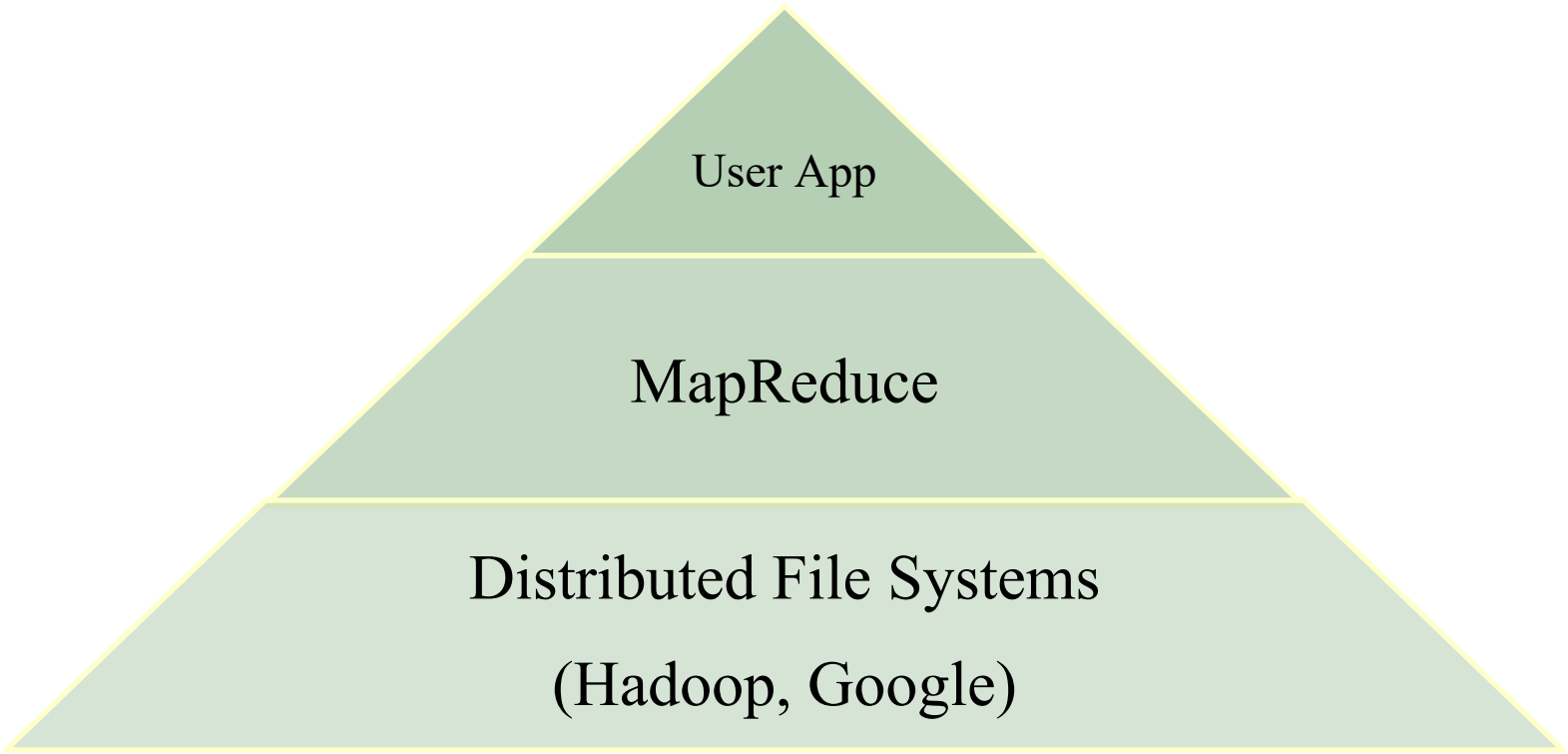
# Pseudo-code

```
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
   for each word w in input_value:
     EmitIntermediate(w, "1");


reduce(String output_key, Iterator
  intermediate_values):
// output_key: a word
// output_values: a list of counts
   int result = 0;
   for each v in intermediate_values:
     result  = result + ParseInt(v);
   Emit(output_key, AsString(result));
```

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Systems Support for MapReduce




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Distributed Filesystems

- **The interface is the same as a single-machine file system**
  - create(), open(), read(), write(), close()
- **Distribute file data to a number of machines (storage units).**
  - Support replication
- **Support concurrent data access**
  - Fetch content from remote servers. Local caching
- **Different implementations sit in different places on complexity/feature scale**
  - Google file system and Hadoop HDFS
    - Highly scalable for large data-intensive applications.
    - Provides redundant storage of massive amounts of data on cheap and unreliable computers

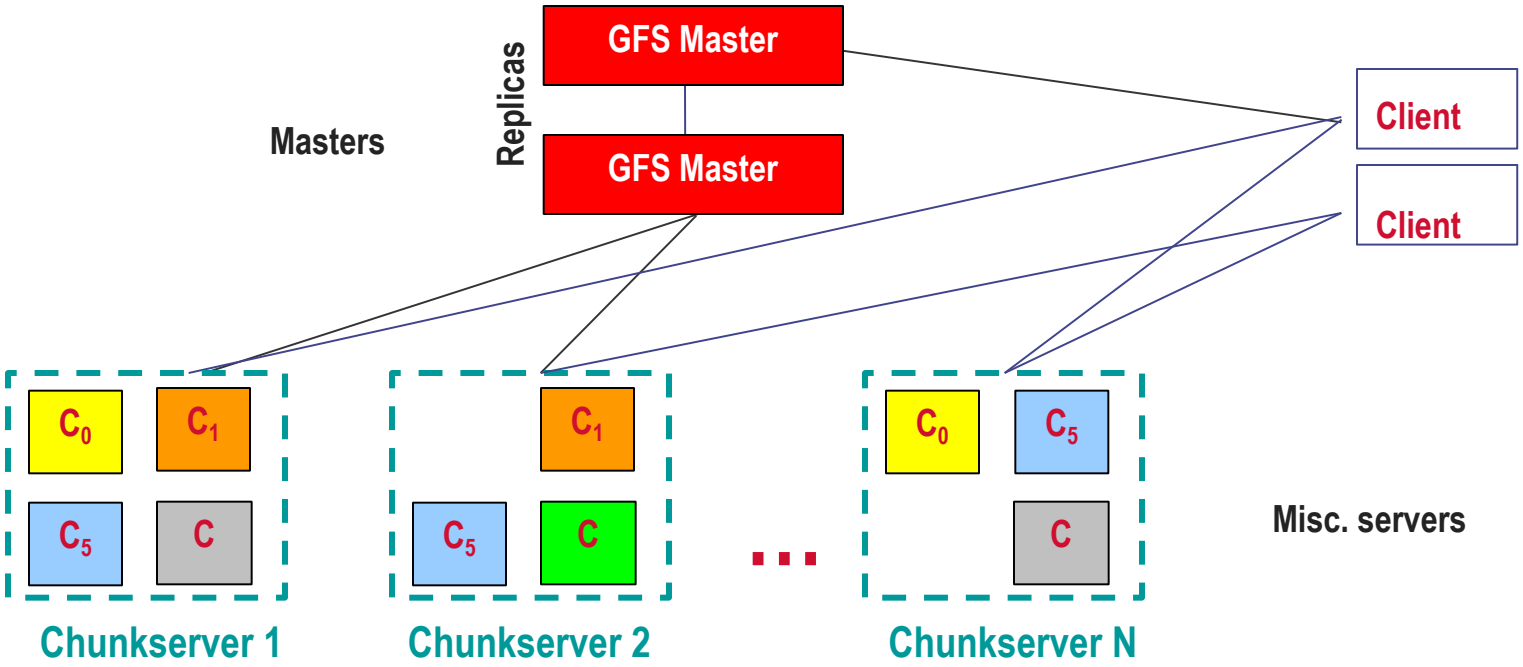
Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Assumptions of GFS/Hadoop DFS

- **High component failure rates**
  - Inexpensive commodity components fail all the time
- **"Modest" number of HUGE files**
  - Just a few million
  - Each is 100MB or larger; multi-GB files typical
- **Files are write-once, mostly appended to**
  - Perhaps concurrently
- **Large streaming reads**
- **High sustained throughput favored over low latency**

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# GFS Design

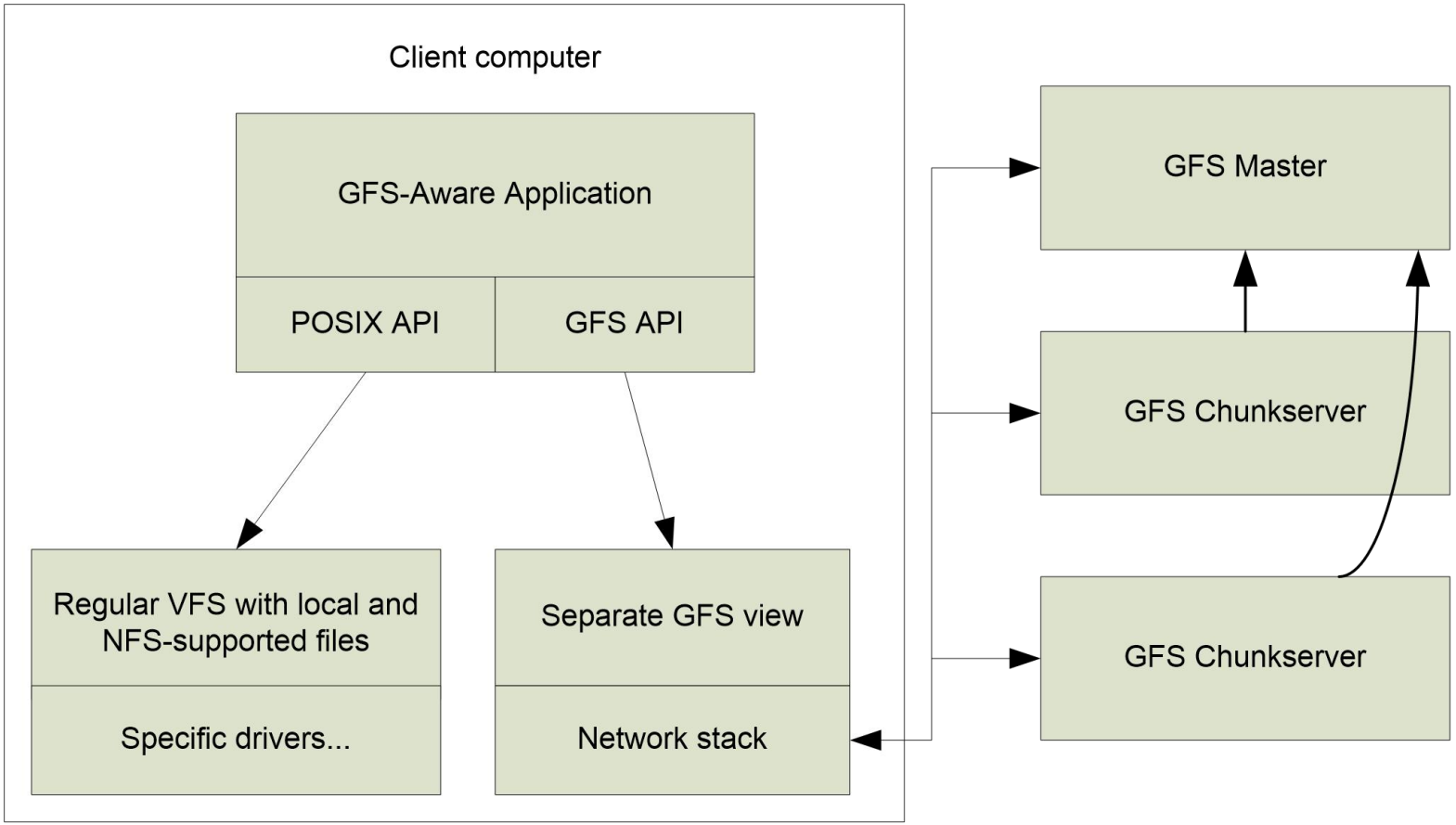



- **Files are broken into chunks (typically 64 MB) and serve in chunk servers**
- **Master manages metadata, but clients may cache meta data obtained.**
  - Data transfers happen directly between clients/chunk-servers
- **Reliability through replication**
  - Each chunk replicated across 3+ *chunk-servers*

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

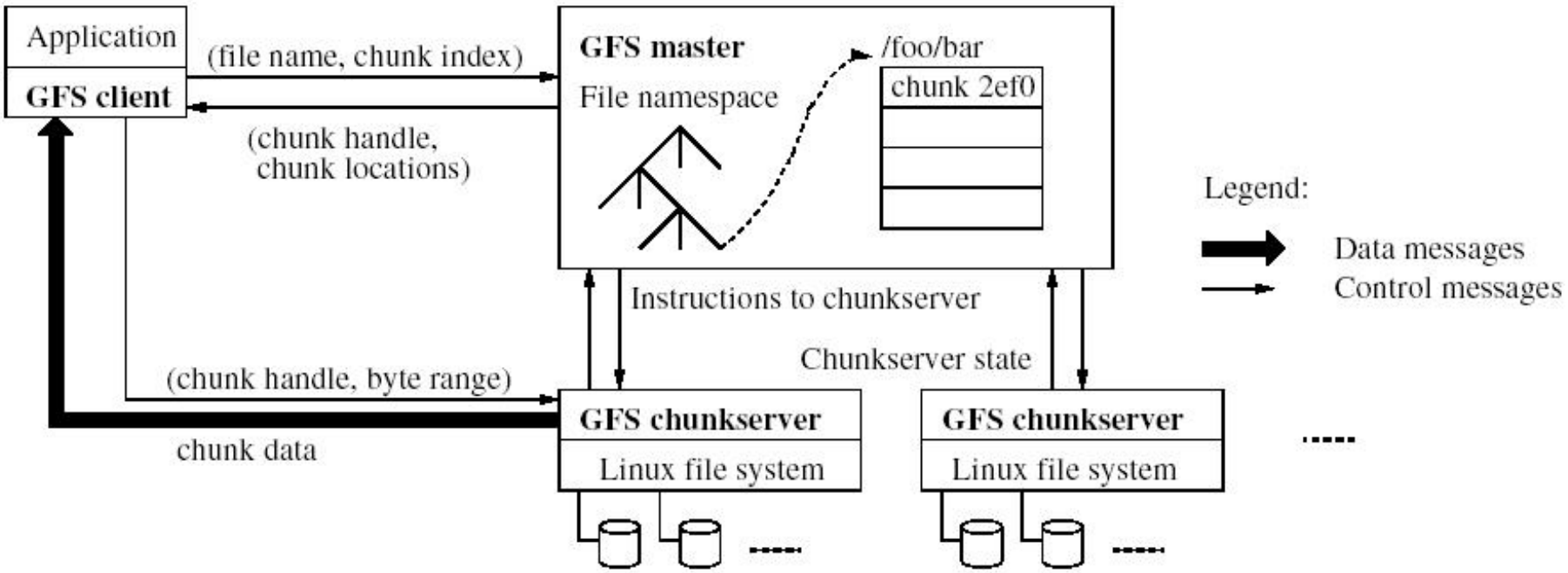
# GFS Client Block Diagram

- Provide both POSIX standard file interface, and costumed API
- Can cache meta data for direct client-chunk server access



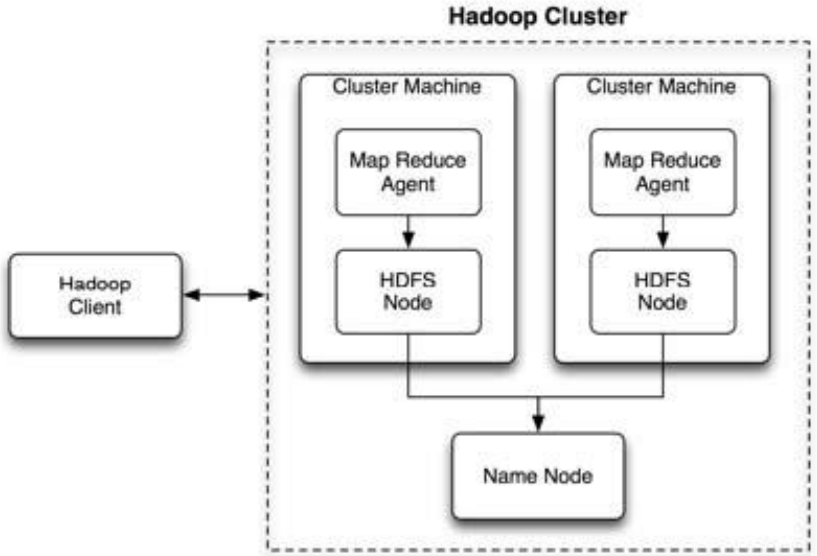

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

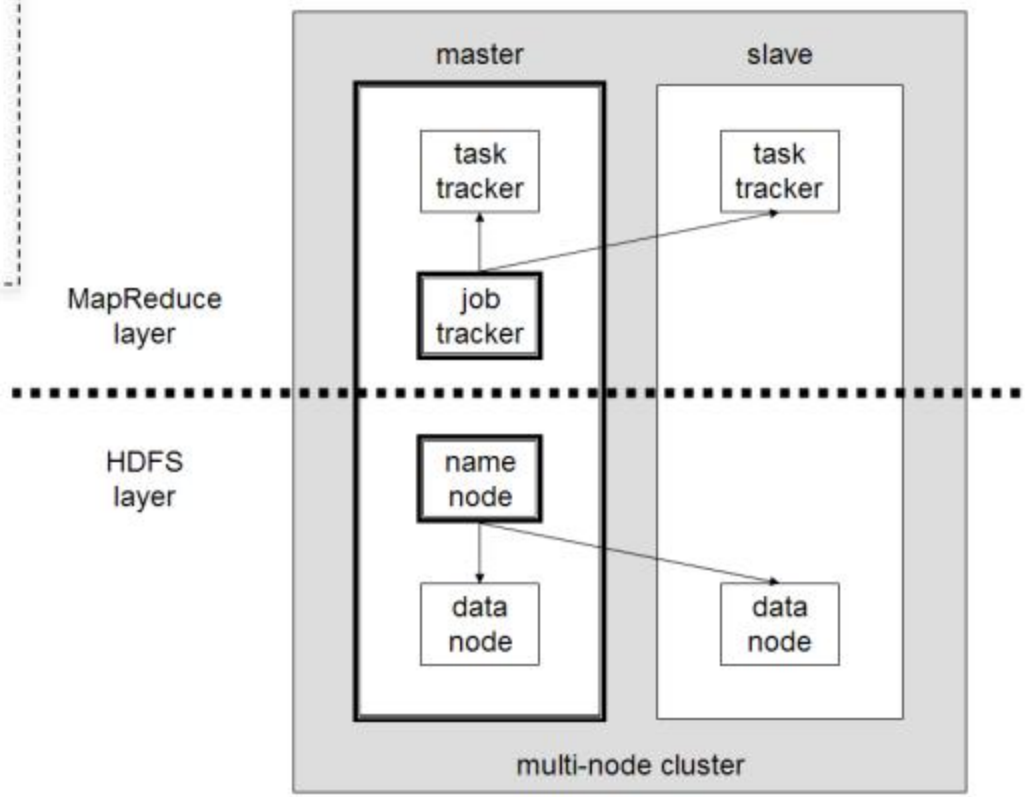
# Read/write Access Flow in GFS




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Hadoop DFS with MapReduce




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce: Execution Overview

Master Server distributes M map tasks to machines and monitors their progress.

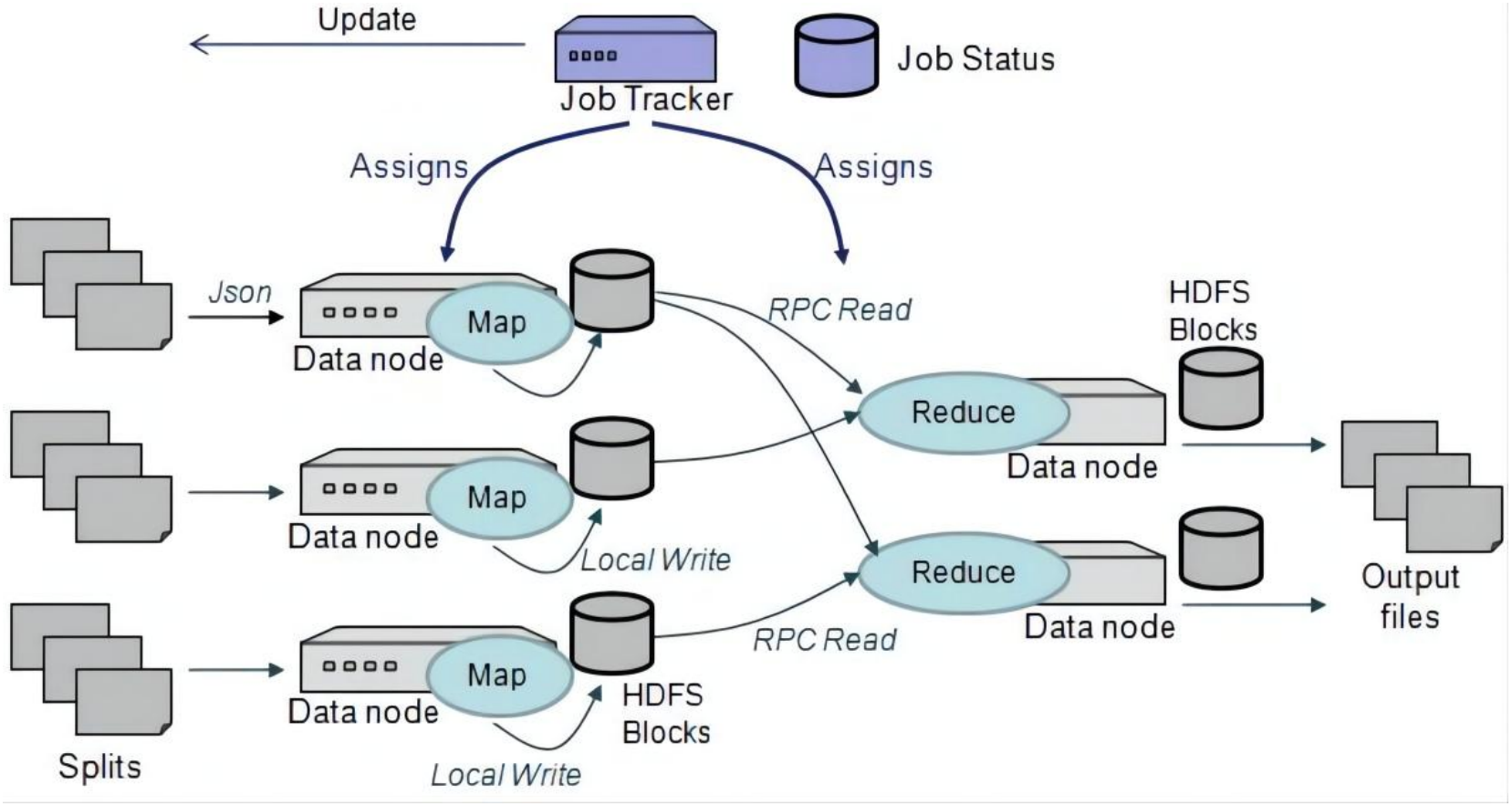
Map task reads the allocated data, saves the map results in local buffer.

Shuffle phase assigns reducers to these buffers, which are remotely read and processed by reducers.

Reducers output the result on stable storage.

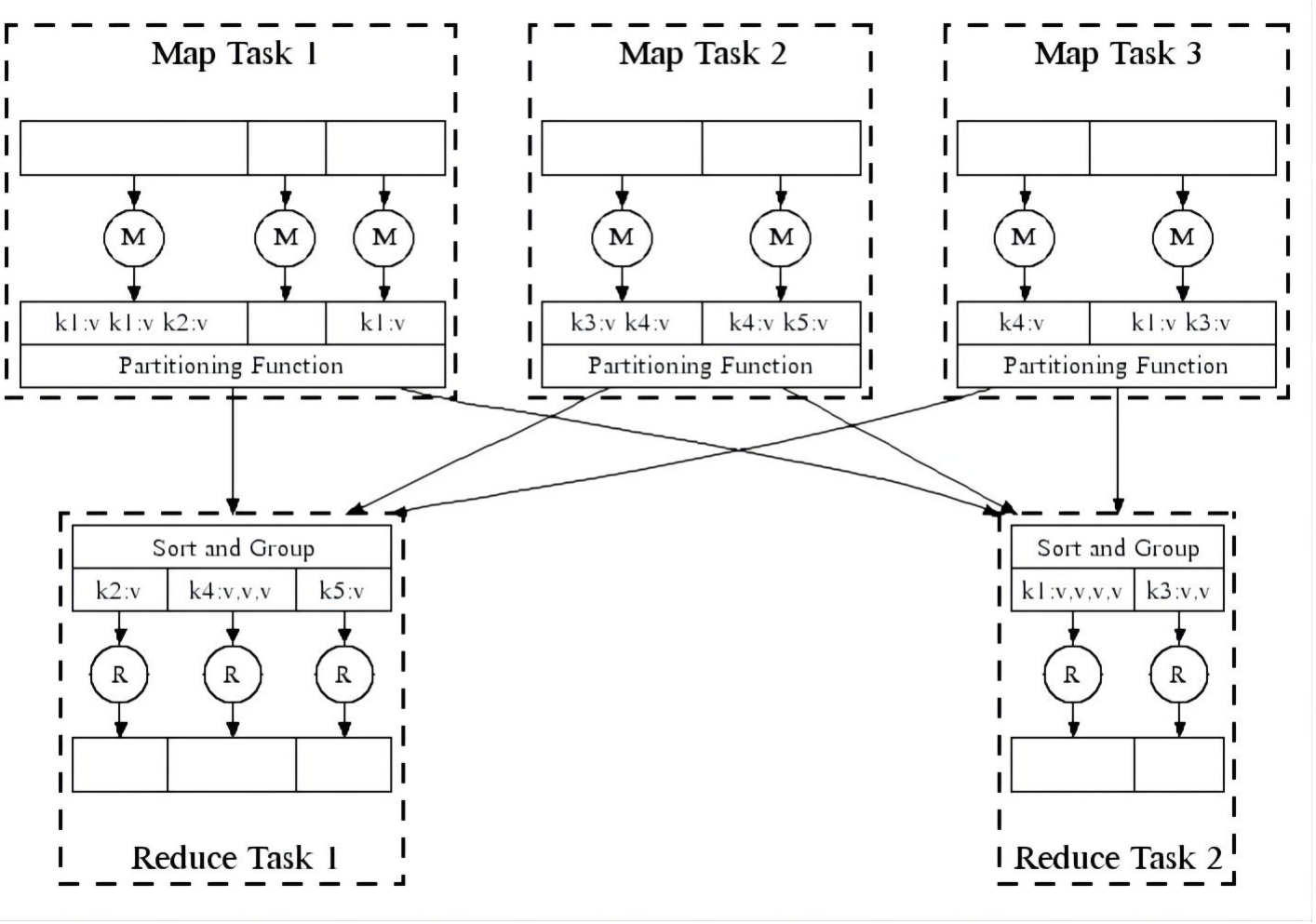
Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Execute MapReduce on a Cluster of Machines with Hadoop DFS




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce in Parallel: Example



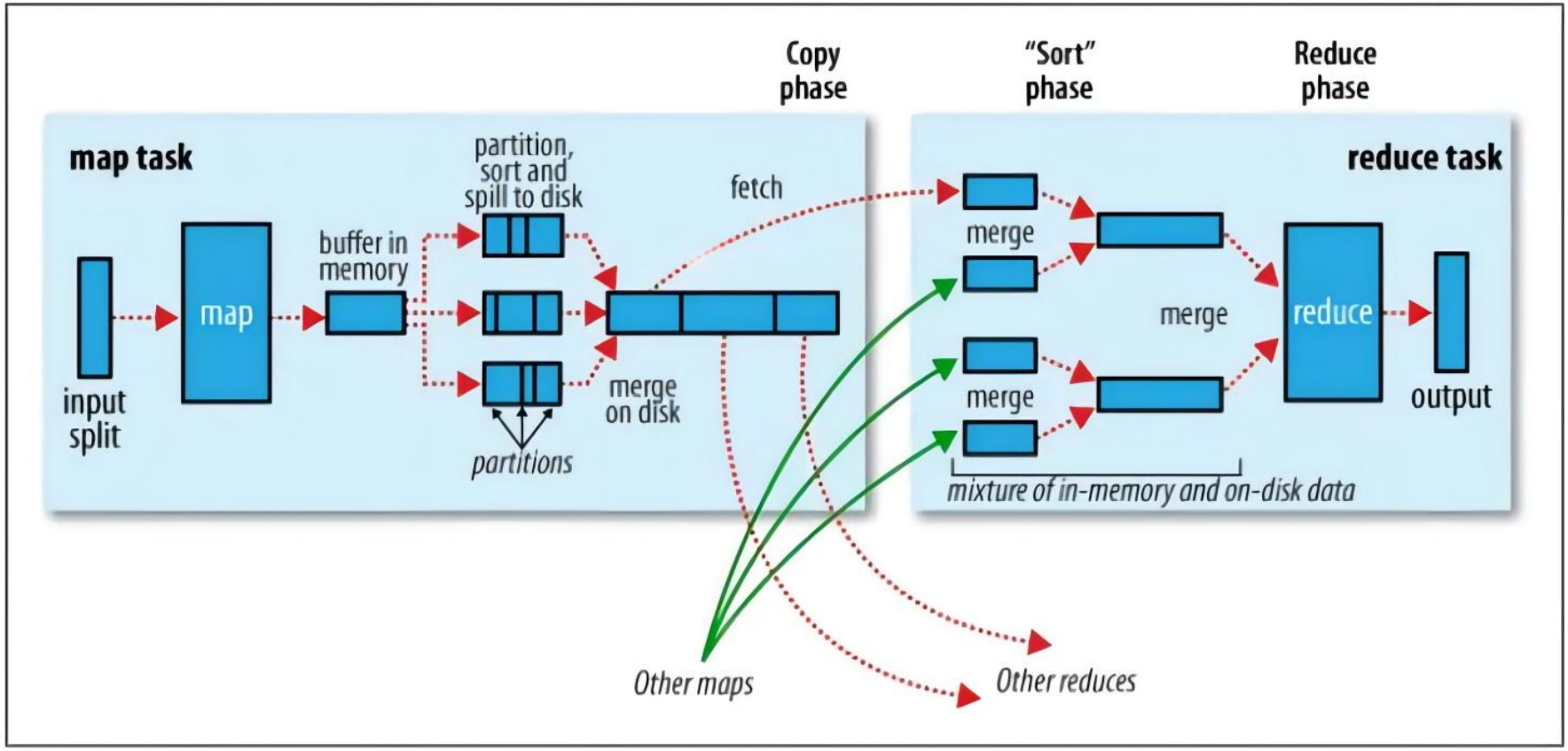

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce: Execution Details

- **Input reader**
  - Divide input into splits, assign each split to a Map task
- **Map task**
  - Apply the Map function to each record in the split
  - Each Map function returns a list of (key, value) pairs
- **Shuffle/Partition and Sort**
  - Shuffle distributes sorting & aggregation to many reducers
  - All records for key *k* are directed to the same reduce processor
  - Sort groups the same keys together, and prepares for aggregation
- **Reduce task**
  - Apply the Reduce function to each key
  - The result of the Reduce function is a list of (key, value) pairs

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

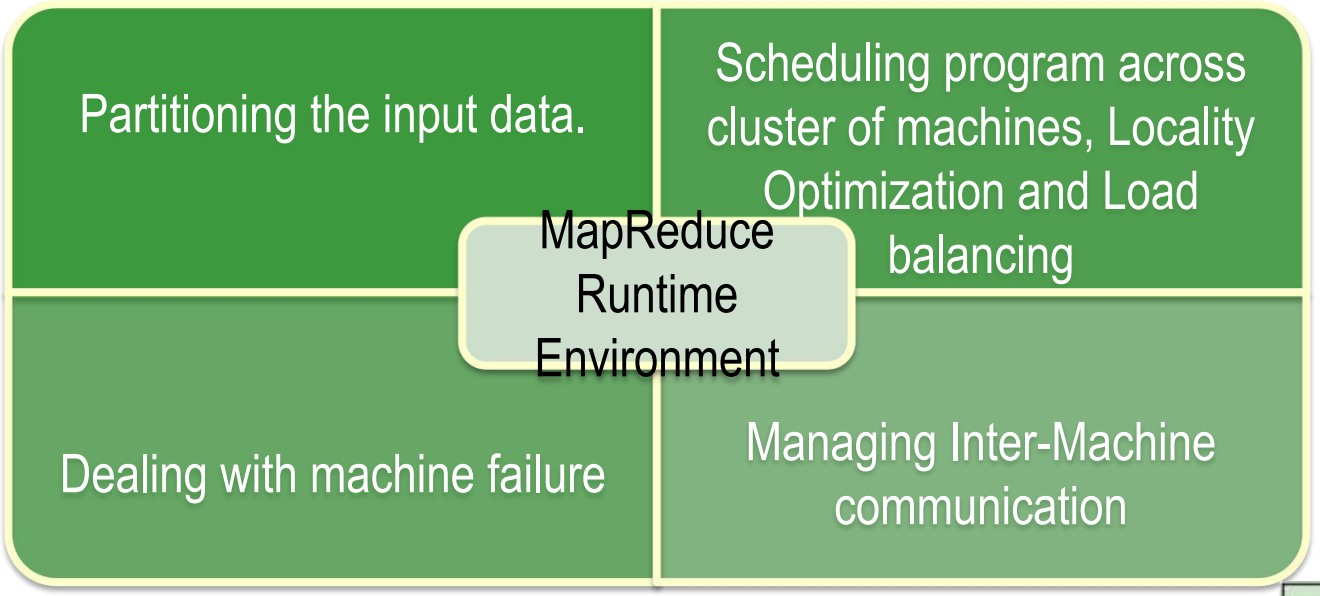
# MapReduce with Data Shuffling & Sorting

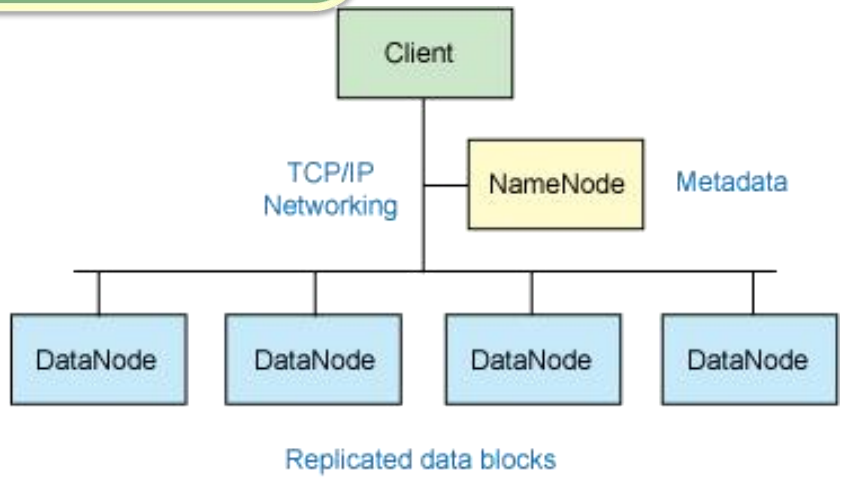



Tom White, *Hadoop: The Definitive Guide*

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

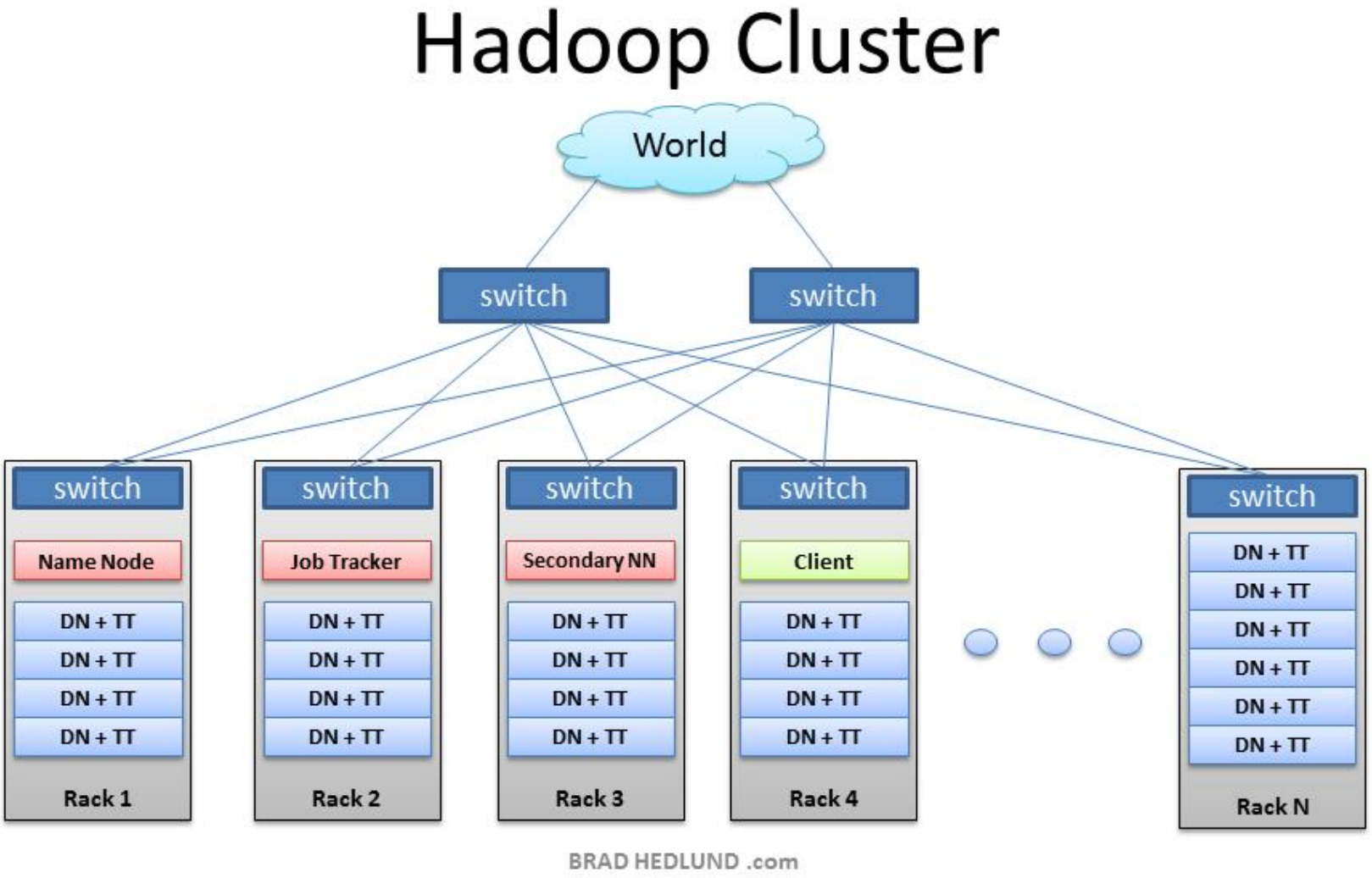
# MapReduce: Runtime Environment & Hadoop




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Hadoop Cluster with MapReduce




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce: Fault Tolerance

- **Handled via re-execution of tasks.**
  - Task completion committed through master
- **Mappers save outputs to local disk before serving to reducers**
  - Allows recovery if a reducer crashes
  - Allows running more reducers than # of nodes
- **If a task crashes:**
  - Retry on another node
    - OK for a map because it had no dependencies
    - OK for reduce because map outputs are on disk
  - If the same task repeatedly fails, fail the job or ignore that input block
  - : For the fault tolerance to work, user tasks must be deterministic and side-effect-free2.
- **If a node crashes:**
  - Relaunch its current tasks on other nodes
  - Relaunch any maps the node previously ran
    - Necessary because their output files were lost along with the crashed node

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce: Locality Optimization

- Leverage the distributed file system to schedule a map task on a machine that contains a replica of the corresponding input data.
- Thousands of machines read input at local disk speed
- Without this, rack switches limit read rate

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce: Redundant Execution

- **Slow workers are source of bottleneck, may delay completion time.**
- **Near end of phase, spawn backup tasks, one to finish first wins.**
- **Effectively utilizes computing power, reducing job completion time by a factor.**

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce: Skipping Bad Records

- **Map/Reduce functions sometimes fail for particular inputs.**
- **Fixing the Bug might not be possible : Third Party Libraries.**
- **On Error**
  - Worker sends signal to Master
  - If multiple error on same record, skip record

# MapReduce: Miscellaneous Refinements

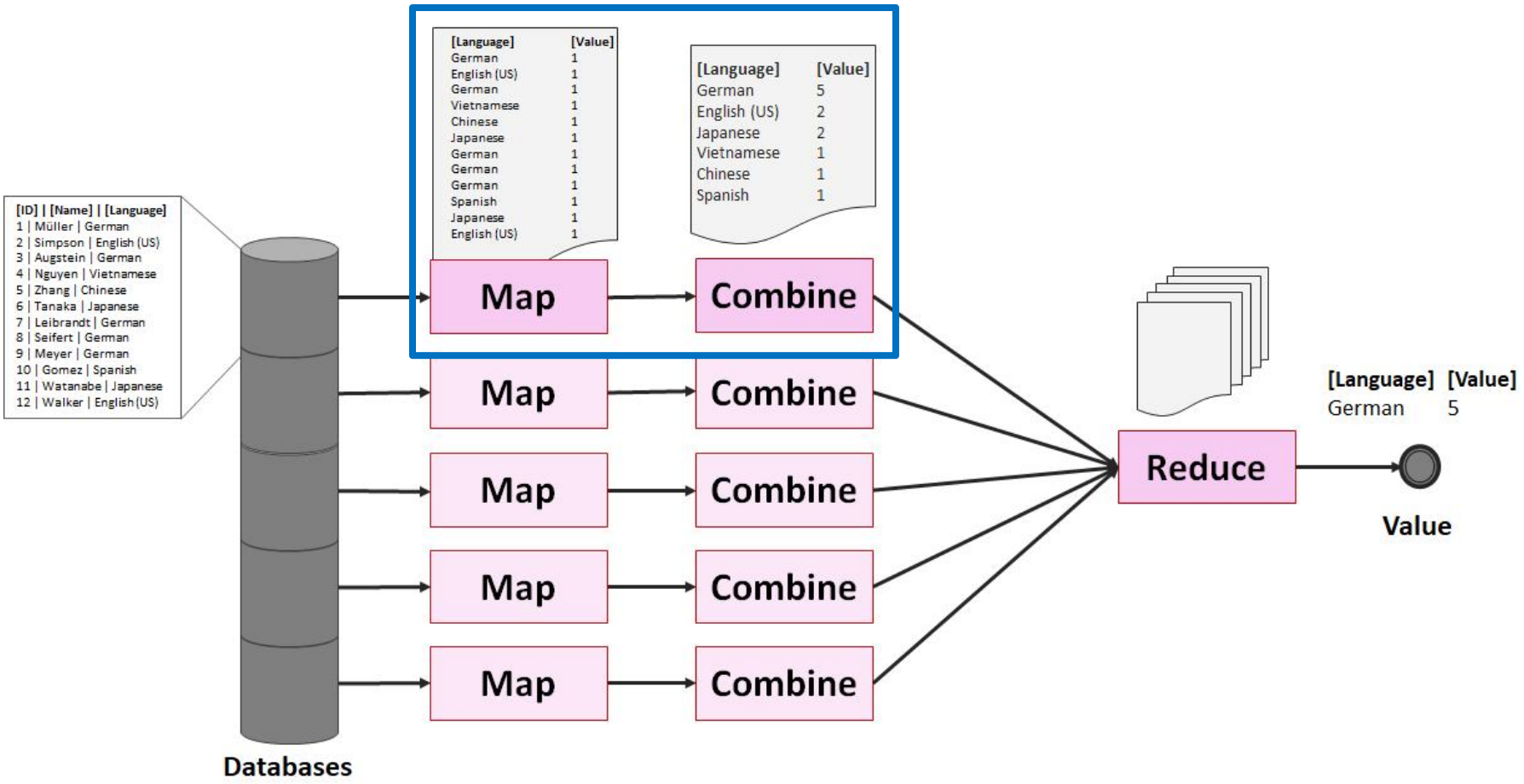
- Combiner function at a map task
- Sorting Guarantees within each reduce partition.
- Local execution for debugging/testing
- User-defined counters

# Combining Phase

- Run on map machines after map phase
- "Mini-reduce," only on local map output
- Used to save bandwidth before sending data to full reduce tasks
- Reduce tasks can be combiner if commutative & associative

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Combiner, Graphically




https://data-science-blog.com/blog/2018/03/26/distributed-computing-mapreduce-algorithmus/

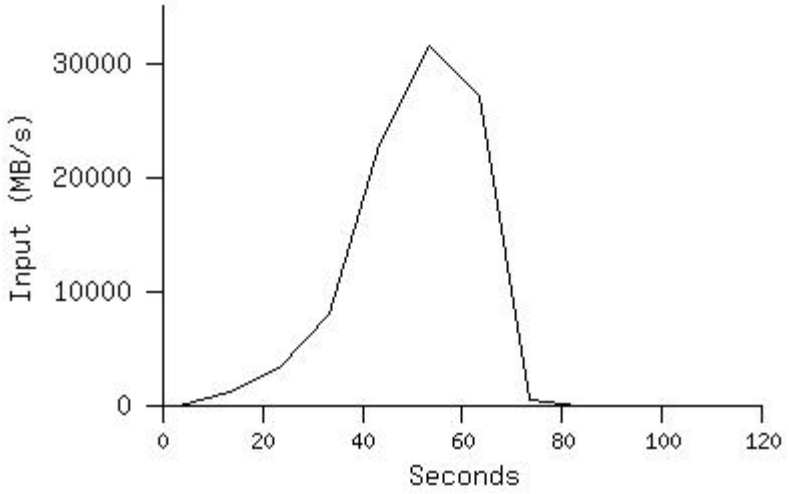
# Performance evaluation

- **Tests run on cluster of 1800 machines:**
  - 4 GB of memory
  - Dual-processor 2 GHz Xeons with Hyperthreading
  - Dual 160 GB IDE disks
  - Gigabit Ethernet per machine
  - Bisection bandwidth approximately 100 Gbps
- **Two benchmarks:**
  - MR_GrepScan $10^{10}$ 100-byte records to extract records matching a rare pattern (92K matching records)
  - MR_SortSort $10^{10}$ 100-byte records (modeled after TeraSort benchmark)

J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI'04.

# MR_Grep

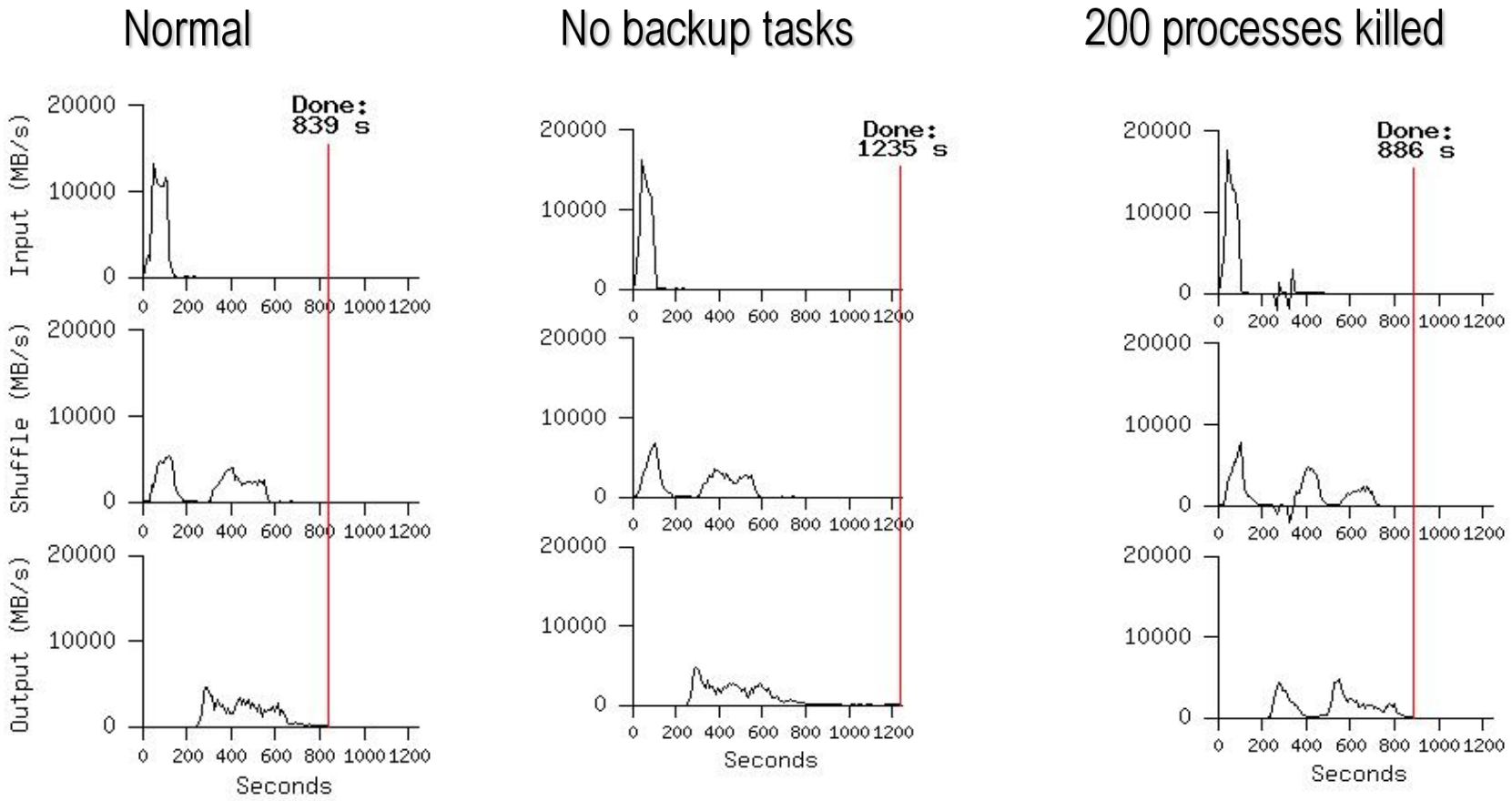
- **Locality optimization helps:**
  - 1800 machines read 1 TB at peak ~31 GB/s
  - Without this, rack switches would limit to 10 GB/s
- **Startup overhead is significant for short jobs**




J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI'04.

# MR_Sort

- **Backup tasks reduce job completion time a lot!**
- **System deals well with failures**




J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI'04.

# Google Experience: Rewrite of Production Indexing System

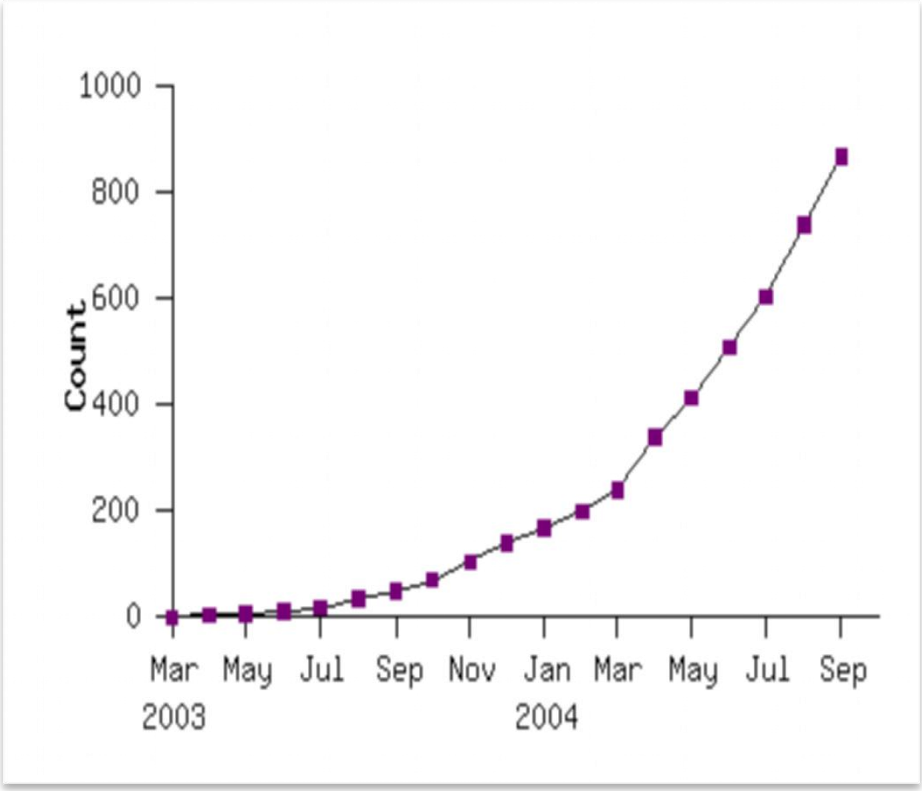
- **Rewrote Google's production indexing system using MapReduce**
  - Set of ~~10, 14, 17, 21,~~ 24 MapReduce operations
  - New code is simpler, easier to understand
  - MapReduce takes care of failures, slow machines
  - Easy to make indexing faster by adding more machines

J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI'04.

# Examples of MapReduce Usage in Web Applications

- Distributed Grep.
- Count of URL Access Frequency.
- Clustering (K-means)
- Graph Algorithms.
- Indexing Systems

**MapReduce Programs In Google Source Tree**




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# “Jeff Dean Facts”

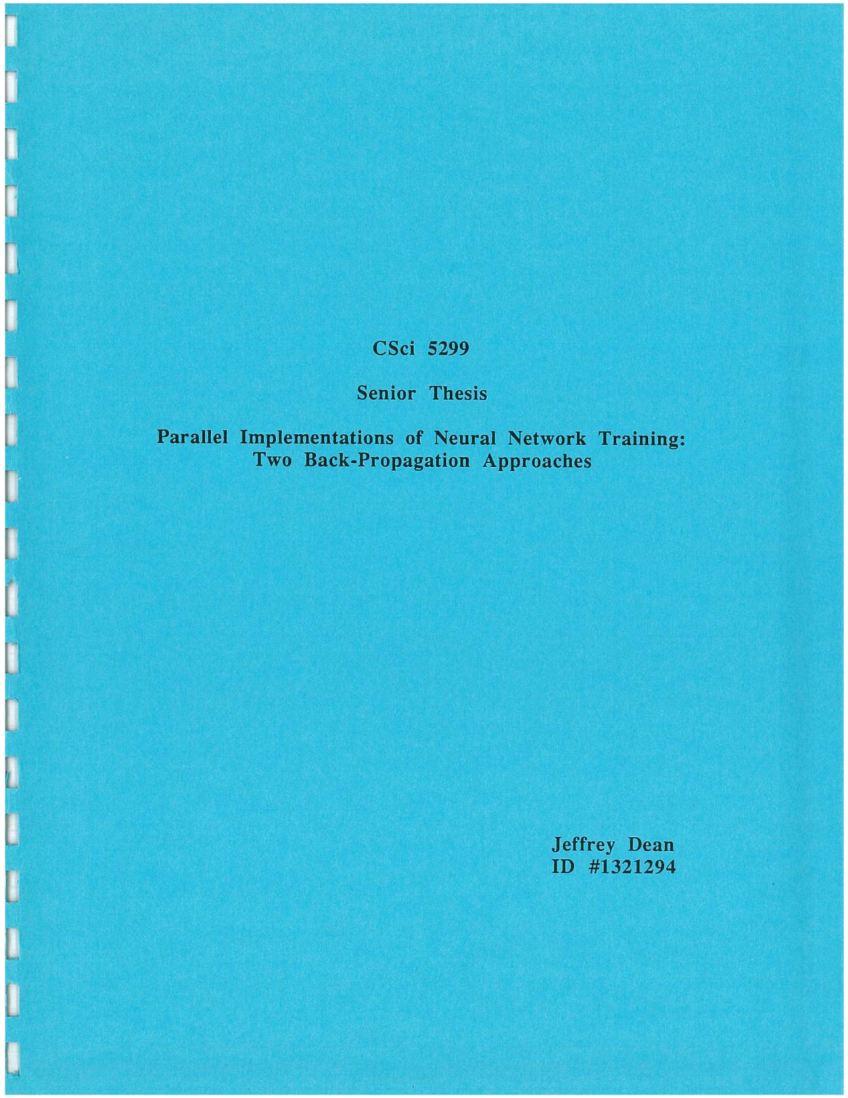
# *"Jeff Dean Facts"*

- Compilers don't warn Jeff Dean. Jeff Dean warns compilers.
- Jeff Dean builds his code before committing it, but only to check for compiler and linker bugs.
- Jeff Dean puts his pants on one leg at a time, but if he had more legs, you would see that his approach is O(log n).
- When he heard that Jeff Dean's autobiography would be exclusive to the platform, Richard Stallman bought a Kindle.
- Jeff Dean writes directly in binary. He then writes the source code as a documentation for other developers.

# “Jeff Dean Facts”

- During his own Google interview, Jeff Dean was asked the implications if P=NP were true. He said, "P = 0 or N = 1." Then, before the interviewer had even finished laughing, Jeff examined Google’s public certificate and wrote the private key on the whiteboard.
- The x86-64 spec includes several undocumented instructions marked 'private use.' They are actually for Jeff Dean's use.
- When Jeff Dean has an ergonomic evaluation, it is for the protection of his keyboard.
- All pointers point to Jeff Dean.
- The rate at which Jeff Dean produces code jumped by a factor of 40 in late 2000 when he upgraded his keyboard to USB 2.0

# Jeff Dean's Senior Thesis (1990)



CSci 5299

Senior Thesis

Parallel Implementations of Neural Network Training:
Two Back-Propagation Approaches

Jeffrey Dean
ID #1321294

# Hadoop and Tools

- **Various Linux Hadoop clusters around**
  - Cluster +Hadoop
    - http://hadoop.apache.org
  - Amazon EC2
- **Windows and other platforms**
  - The NetBeans plugin simulates Hadoop
  - The workflow view works on Windows
- **Hadoop-based tools**
  - For Developing in Java, NetBeans plugin
- **Pig Latin,** a SQL-like high level data processing script language
- **Hive,** Data warehouse, SQL
- **Mahout,** Machine Learning algorithms on Hadoop
- **HBase,** Distributed data store as a large table

# More MapReduce Applications

- Map Only processing
- Filtering and accumulation
- Database join
- Reversing graph edges
- Producing inverted index for web search
- PageRank graph processing

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce Use Case 1: Map Only

**Data distributive tasks – Map Only**

- **e.g., classify individual documents**
- **Map does everything**
  - Input: (docno, doc_content), …
  - Output: (docno, [class, class, …]), …
- **No reduce tasks**

# MapReduce Use Case 2: Filtering and Accumulation

**Filtering & Accumulation – Map and Reduce**

- **e.g., Counting total enrollments of two given student classes**
- **Map** selects records and outputs initial counts
  - In: (Jamie, 11741), (Tom, 11493), …
  - Out: (11741, 1), (11493, 1), …
- **Shuffle/Partition** by class_id
- **Sort**
  - In: (11741, 1), (11493, 1), (11741, 1), …
  - Out: (11493, 1), …, (11741, 1), (11741, 1), …
- **Reduce accumulates counts**
  - In: (11493, [1, 1, …]), (11741, [1, 1, …])
  - Sum and Output: (11493, 16), (11741, 35)

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce Use Case 3: Database Join

- A JOIN is a means for combining fields from two tables by using values common to each.
- Example: For each employee, find the department he works in

| Employee Table | |
|---|---|
| **LastName** | **DepartmentID** |
| Rafferty | 31 |
| Jones | 33 |
| Steinberg | 33 |
| Robinson | 34 |
| Smith | 34 |

**JOIN**

**Pred:**

**EMPLOYEE.DepID= DEPARTMENT.DepID**

| Department Table | |
|---|---|
| **DepartmentID** | **DepartmentName** |
| 31 | Sales |
| 33 | Engineering |
| 34 | Clerical |
| 35 | Marketing |

| JOIN RESULT | |
|---|---|
| **LastName** | **DepartmentName** |
| Rafferty | Sales |
| Jones | Engineering |
| Steinberg | Engineering |
| … | … |

# MapReduce Use Case 3: Database Join

**Problem: Massive lookups**

- Given two large lists: (URL, ID) and (URL, doc_content) pairs
- Produce (ID, doc_content)

**Solution:**

- **Input stream**: both (URL, ID) and (URL, doc_content) lists
  - (http://del.icio.us/post, 0), (http://digg.com/submit, 1), …
  - (http://del.icio.us/post, <html0>), (http://digg.com/submit, <html1>), …
- **Map** simply passes input along,
- **Shuffle and Sort** on URL (group ID & doc_content for the same URL together)
  - Out: (http://del.icio.us/post, 0), (http://del.icio.us/post, <html0>), (http://digg.com/submit, <html1>), (http://digg.com/submit, 1), …
- **Reduce** outputs result stream of (ID, doc_content) pairs
  - In: (http://del.icio.us/post, [0, html0]), (http://digg.com/submit, [html1, 1]), …
  - Out: (0, <html0>), (1, <html1>), …

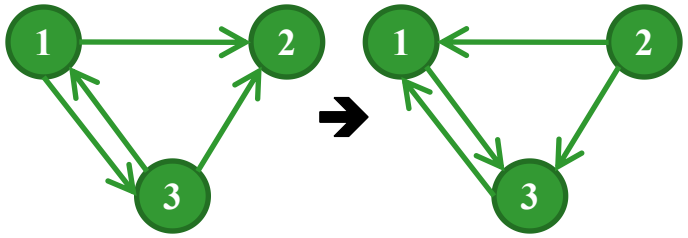
Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce Use Case 4: Reverse graph edge directions & output in node order

- **Input example: adjacency list of graph (3 nodes and 4 edges)**

(3, [1, 2])
(1, [2, 3]) ➔

(1, [3])
(2, [1, 3])
(3, [1])




- **MapReduce format**
  - Input: (3, [1, 2]), (1, [2, 3]).
  - Intermediate: (1, [3]), (2, [3]), (2, [1]), (3, [1]).
    - (reverse edge direction)
  - Out: (1,[3]) (2, [1, 3]) (3, [1]).

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce Use Case 5: Inverted Indexing Preliminaries

**Construction of inverted lists for document search**

- Input: documents:
  - (docid, [term, term..]), (docid, [term, ..]), ..
- Output:
  - (term, [docid, docid, …])
  - e.g., (apple, [1, 23, 49, 127, …])

**A document id is an internal document id,**

e.g., a unique integer

- Not an external document id such as a URL

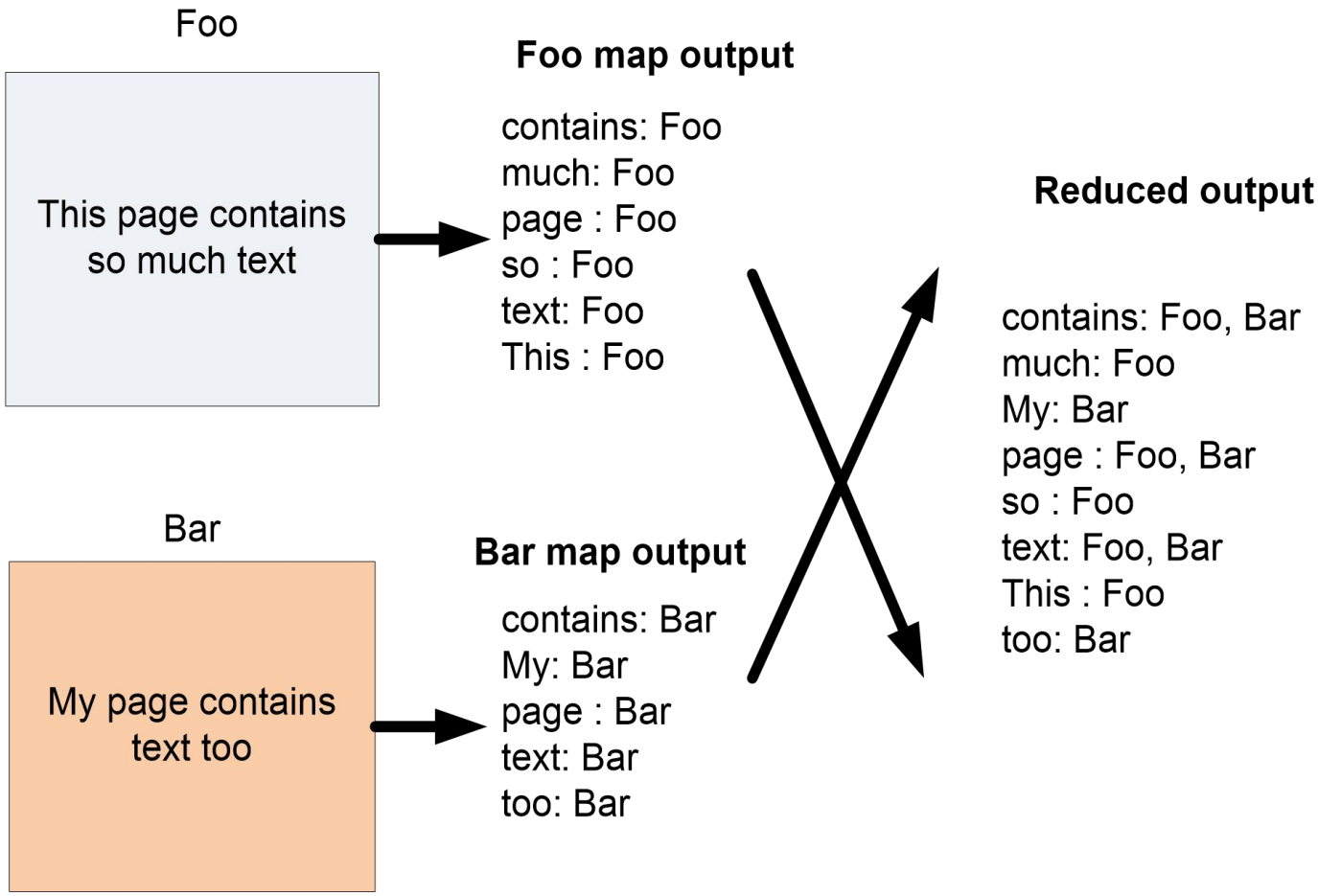
Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Using MapReduce to Construct Indexes: A Simple Approach

**A simple approach to creating inverted lists**

- **Each Map task is a document parser**
  - Input: A stream of documents
  - Output: A stream of (term, docid) tuples
    - (long, 1) (ago, 1) (and, 1) … (once, 2) (upon, 2) …
    - We may create internal IDs for words.
- **Shuffle sorts tuples by key and routes tuples to Reducers**
- **Reducers convert streams of keys into streams of inverted lists**
  - Input: (long, 1) (long, 127) (long, 49) (long, 23) …
  - The reducer sorts the values for a key and builds an inverted list
  - Output: (long, [df:492, docids:1, 23, 49, 127, …])

# Inverted Index: Data flow




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# Processing Flow Optimization

**A more detailed analysis of processing flow**

- **Map:** ($docid_1$, $content_1$) → ($t_1$, $docid_1$) ($t_2$, $docid_1$) …
- **Shuffle** by t, prepared for map-reducer communication
- **Sort** by t, conducted in a reducer machine
  ($t_5$, $docid_1$) ($t_4$, $docid_3$) … → ($t_4$, $docid_3$) ($t_4$, $docid_1$) ($t_5$, $docid_1$) …
- **Reduce:** ($t_4$, [$docid_3$ $docid_1$ …]) → (t, ilist)

docid: a unique integer

t: a term, e.g., "apple"

ilist: a complete inverted list

**but a) inefficient, b) docids are sorted in reducers, and c) assumes ilist of a word fits in memory**

# Using Combine () to Reduce Communication

- **Map:** $(docid_1, content_1) \rightarrow (t_1, ilist_{1,1})\ (t_2, ilist_{2,1})\ (t_3, ilist_{3,1})\ \ldots$
  - Each output inverted list covers just one document
- **Combine locally**

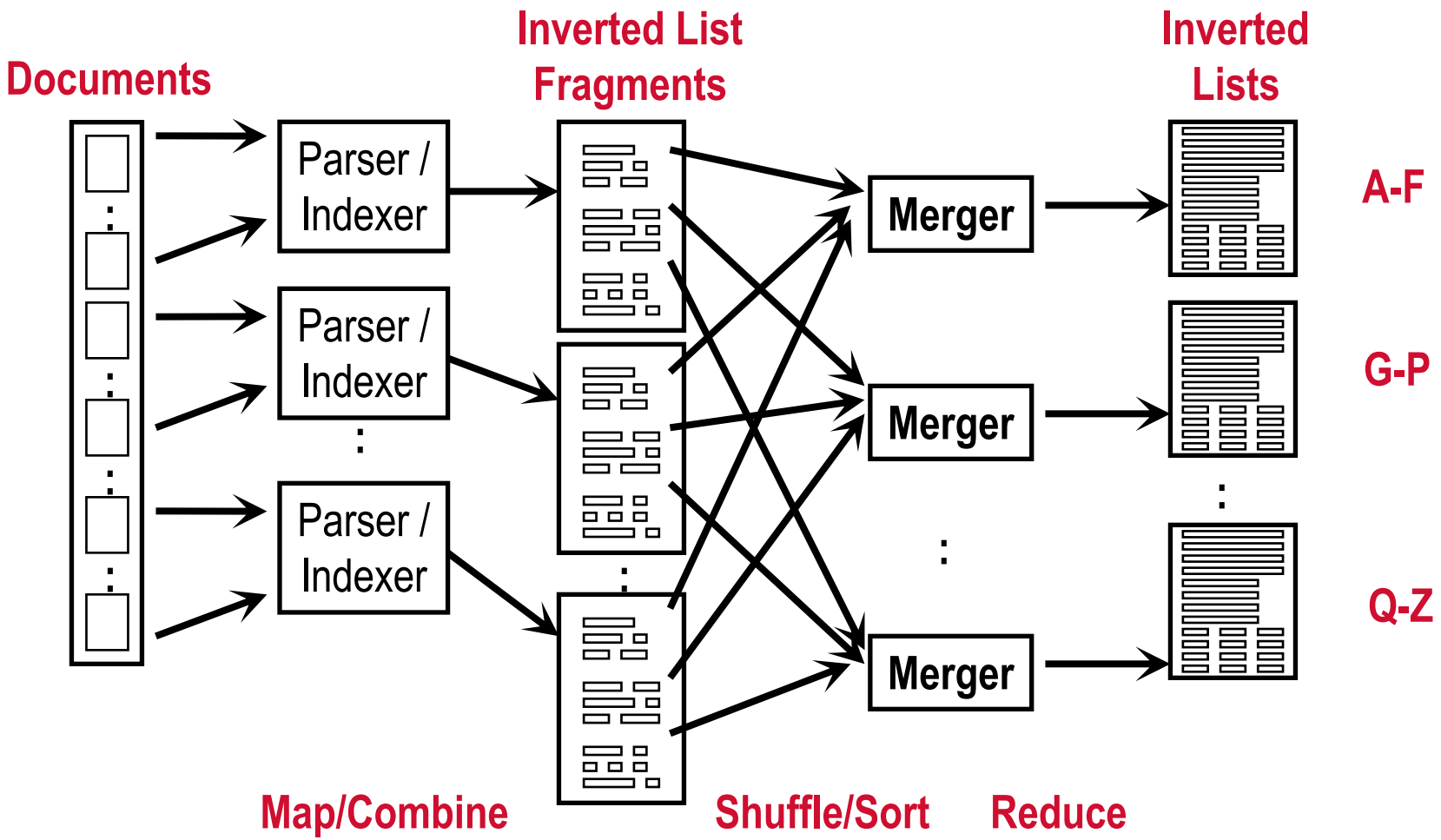
  Sort by t

  Combine: $(t_1\ [ilist_{1,2}\ ilist_{1,3}\ ilist_{1,1}\ \ldots]) \rightarrow (t_1, ilist_{1,27})$
  - Each output inverted list covers a sequence of documents
- **Shuffle** by t
- **Sort** by t

  $(t_4, ilist_{4,1})\ (t_5, ilist_{5,3})\ \ldots \rightarrow (t_4, ilist_{4,2})\ (t_4, ilist_{4,4})\ (t_4, ilist_{4,1})\ \ldots$
- **Reduce:** $(t_7, [ilist_{7,2}, ilist_{3,1}, ilist_{7,4}, \ldots]) \rightarrow (t_7, ilist_{final})$

$ilist_{i,j}$: the j'th inverted list fragment for term i

# Using MapReduce to Construct Indexes




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

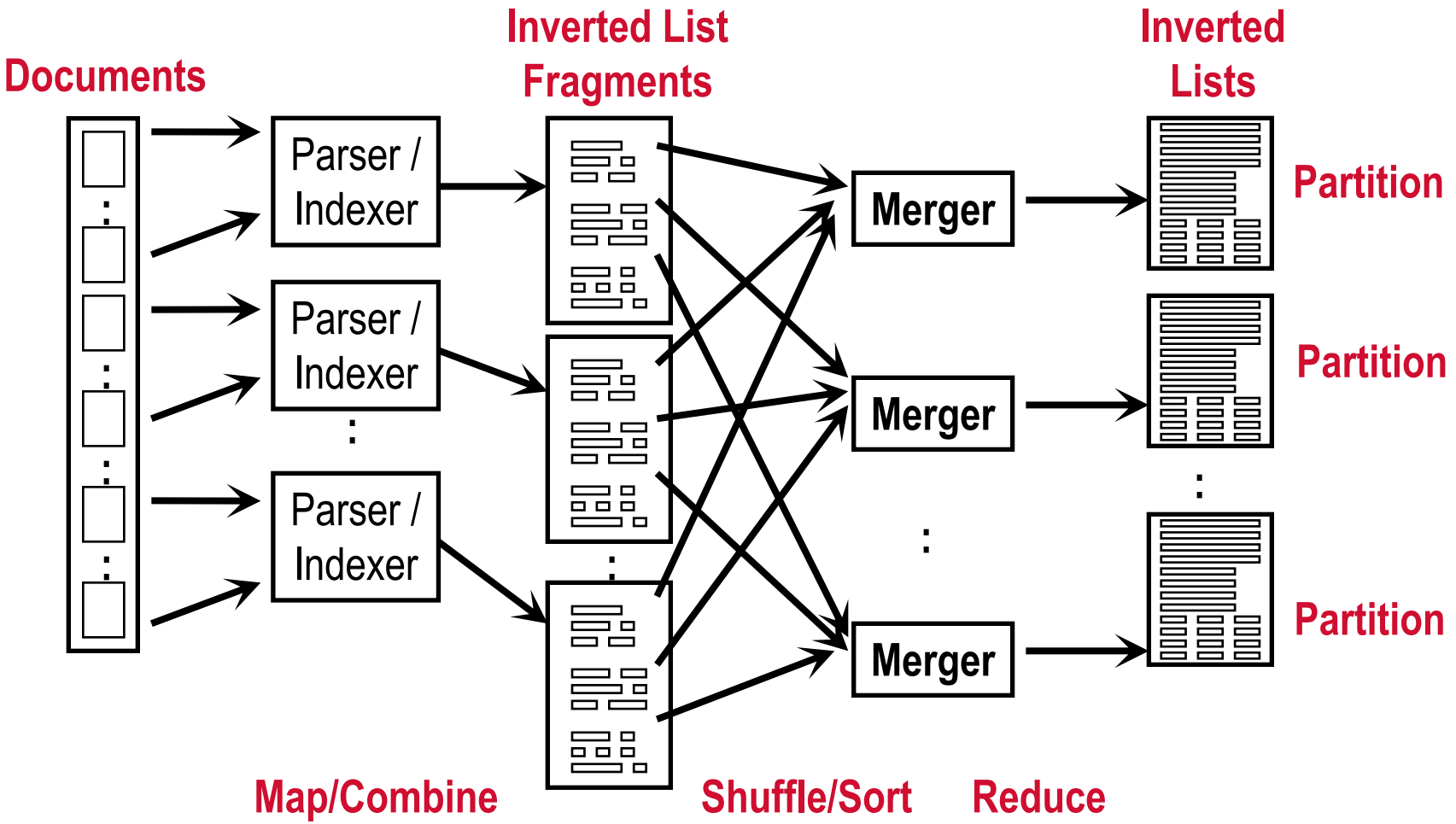
# Construct Partitioned Indexes

- **Useful when the document list of a term does not fit memory**
- **Map:** ($docid_1$, $content_1$) → ([p, $t_1$], $ilist_{1,1}$)
- **Combine** to sort and group values

  ([p, $t_1$] [$ilist_{1,2}$ $ilist_{1,3}$ $ilist_{1,1}$ …]) → ([p, $t_1$], $ilist_{1,27}$)
- **Shuffle** by p
- **Sort** values by [p, t]
- **Reduce:** ([p, $t_7$], [$ilist_{7,2}$, $ilist_{7,1}$, $ilist_{7,4}$, …]) → ([p, $t_7$], $ilist_{final}$)

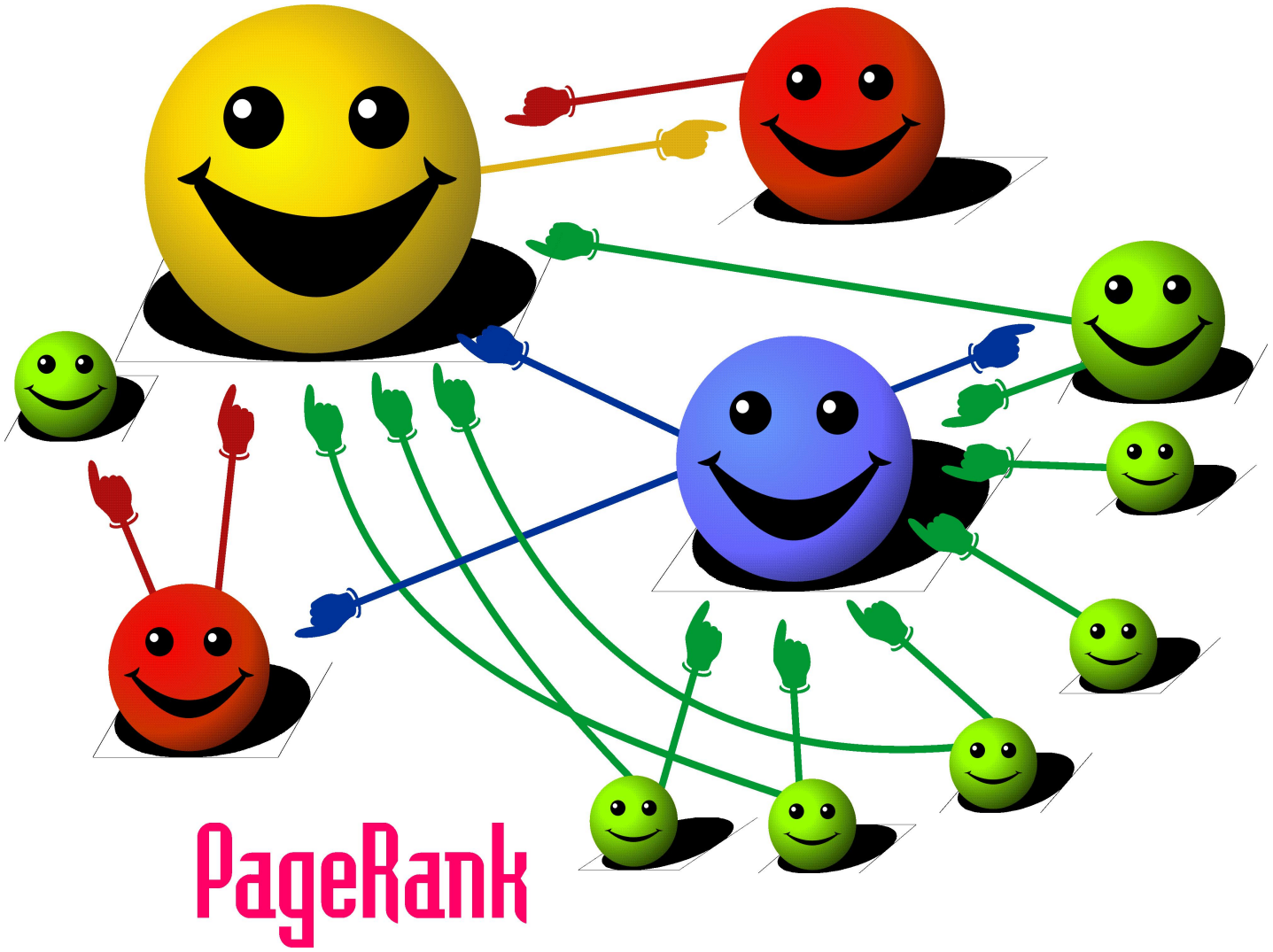
p: partition (shard) id

# Generate Partitioned Index




Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# MapReduce Use Case 6: PageRank

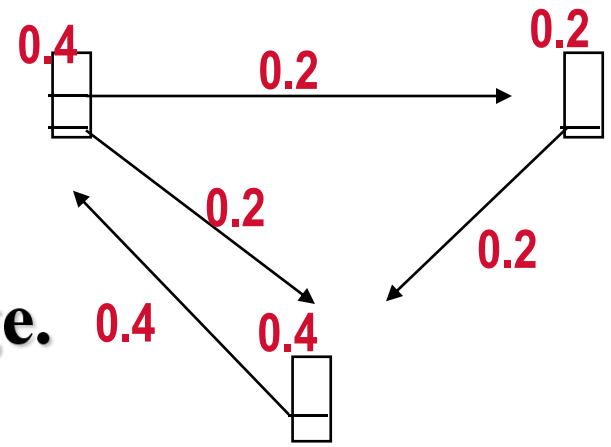



Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# PageRank

- **Model page reputation on the web**

$$PR(x) = (1-d) + d\sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$




- **i=1,n lists all parents of page x.**
- **PR(x) is the page rank of each page.**
- **C(t) is the out-degree of t.**
- **d is a damping factor.**

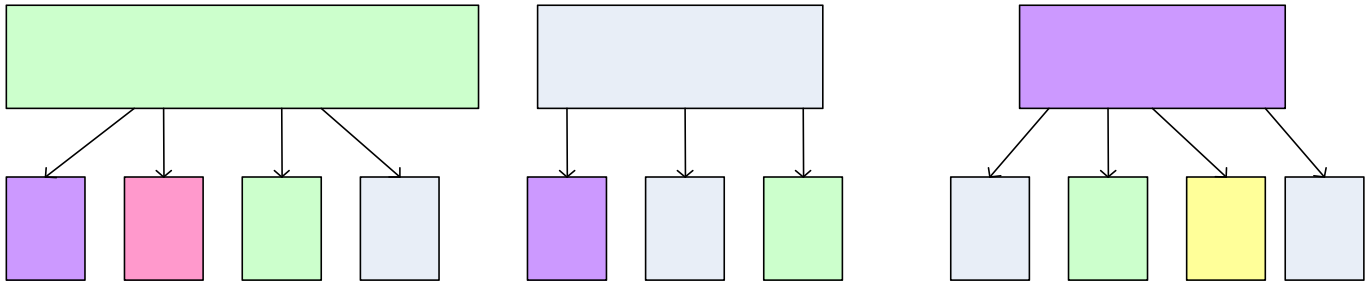
# Computing PageRank Iteratively


- **Effects at each iteration is local. i+1$^{th}$ iteration depends only on i$^{th}$ iteration**
- **At iteration i, PageRank for individual nodes can be computed independently**

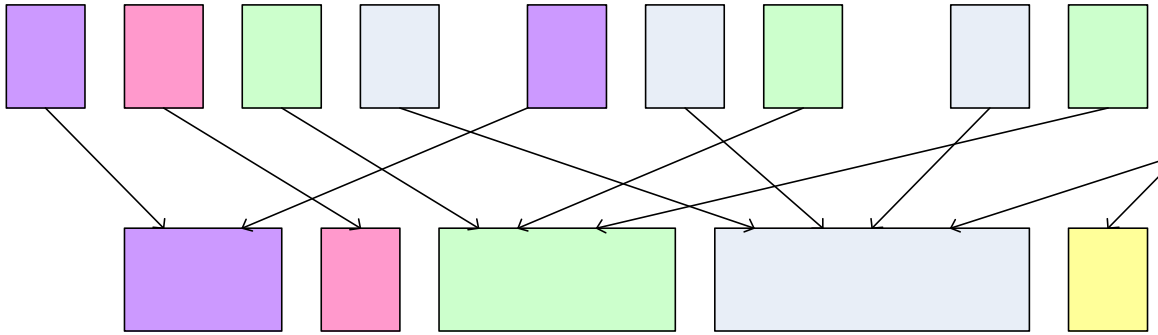
Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# PageRank using MapReduce

**Map**: distribute PageRank "credit" to link targets



**Reduce**: gather up PageRank "credit" from multiple sources to compute new PageRank value



**Iterate until convergence**

Source of Image: Lin 2008

# PageRank Calculation: Preliminaries

**One PageRank iteration:**

- Input:
  - $(id_1, [score_1^{(t)}, out_{11}, out_{12}, ..]), (id_2, [score_2^{(t)}, out_{21}, out_{22}, ..])$ ..
- Output:
  - $(id_1, [score_1^{(t+1)}, out_{11}, out_{12}, ..]), (id_2, [score_2^{(t+1)}, out_{21}, out_{22}, ..])$ ..

**MapReduce elements**

- Score distribution and accumulation
- Database join

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# PageRank: Score Distribution and Accumulation

- **Map**
  - In: $(id_1, [score_1^{(t)}, out_{11}, out_{12}, ..]), (id_2, [score_2^{(t)}, out_{21}, out_{22}, ..])$ ..
  - Out: $(out_{11}, score_1^{(t)}/n_1), (out_{12}, score_1^{(t)}/n_1)$ .., $(out_{21}, score_2^{(t)}/n_2)$, ..
- **Shuffle & Sort by node_id**
  - In: $(id_2, score_1), (id_1, score_2), (id_1, score_1)$, ..
  - Out: $(id_1, score_1), (id_1, score_2)$, .., $(id_2, score_1)$, ..
- **Reduce**
  - In: $(id_1, [score_1, score_2, ..]), (id_2, [score_1, ..])$, ..
  - Out: $(id_1, score_1^{(t+1)}), (id_2, score_2^{(t+1)})$, ..

Tao Yang, "CS 240A: Applied Parallel Computing," UCSB, Winter 2013.

# PageRank: Database Join to associate outlinks with score

- **Map**
  - In & Out: $(id_1, score_1^{(t+1)})$, $(id_2, score_2^{(t+1)})$, .., $(id_1, [out_{11}, out_{12}, ..])$, $(id_2, [out_{21}, out_{22}, ..])$ ..
- **Shuffle & Sort by node_id**
  - Out: $(id_1, score_1^{(t+1)})$, $(id_1, [out_{11}, out_{12}, ..])$, $(id_2, [out_{21}, out_{22}, ..])$, $(id_2, score_2^{(t+1)})$, ..
- **Reduce**
  - In: $(id_1, [score_1^{(t+1)}, out_{11}, out_{12}, ..])$, $(id_2, [out_{21}, out_{22}, .., score_2^{(t+1)}])$, ..
  - Out: $(id_1, [score_1^{(t+1)}, out_{11}, out_{12}, ..])$, $(id_2, [score_2^{(t+1)}, out_{21}, out_{22}, ..])$ ..

# Related Work

- Programming model inspired by functional language primitives
  Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]

Jeff Dean, “MapReduce: Simplified Data Processing on Large Clusters ,” OSDI 2004

# Conclusions

- **MapReduce advantages**
- **Application cases**
  - Map only: for totally distributive computation
  - Map+Reduce: for filtering & aggregation
  - Database join: for massive dictionary lookups
  - Secondary sort: for sorting on values
  - Inverted indexing: combiner, complex keys
  - PageRank: side effect files

# For More Information

- J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137-150. 2004.
- S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google File System." *OSDI 200?*
- http://hadoop.apache.org/common/docs/current/mapred_tutorial.html. "Map/Reduce Tutorial". Fetched January 21, 2010.
- Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media. June 5, 2009
- http://developer.yahoo.com/hadoop/tutorial/module4.html
- J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*, Book Draft. February 7, 2010.