
Parallel computing, MapReduce, Hadoop (Data Storage and Algorithms)

James G. Shanahan¹

¹*Church and Duncan Group and iSchool, UC Berkeley, CA*

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

June 13, 2016
Lecture 3



Parallel computing, MapReduce, Hadoop

- • Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)
- Sync time
 - Install Hadoop; run locally; run in the cloud?

V4



"Swiss army knife of the 21st century"

Media Guardian Innovation Awards



<http://www.guardian.co.uk/technology/2011/mar/25/media-guardian-innovation-awards-apache-hadoop>

-
- We start in Section 2.1 with an overview of functional programming, from which MapReduce draws its inspiration. Section 2.2 introduces the basic programming model, focusing on mappers and reducers. Section 2.3 discusses the role of the execution framework in actually running MapReduce programs (called jobs). Section 2.4 fills in additional details by introducing partitioners and combiners, which provide greater control over data flow. MapReduce would not be practical without a tightly-integrated distributed file system that manages the data being processed; Section 2.5 covers this in detail. Tying everything together, a complete cluster architecture is described in Section 2.6.

References

- **Data intensive text processing with MapReduce,**
Jimmy Lin and Chris Dyer
- **Hadoop, Tom White**

Parallel computing, MapReduce, Hadoop

- • Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)
- Sync time
 - Install Hadoop; run locally; run in the cloud?

V4

Assume 100 Billion webpages

- **How store them?**
 - $10K$ per page $10^{11} = 10^{15}$ Bytes (1 Petabyte of raw data)
- **How can we scan them?**
 - $10^{15} \times 10^{-8} \times 10^{-5} = 10^2$ days
- **How to do something useful with them?**
 - Document frequency for a term?
 - Extract outlinks of page and say just count the number of inlinks for a webpage

Why Parallelism?

- **Data size is increasing**
 - Single node architecture is reaching its limit
 - Scan 1000 TB on 1 node @ 100 MB/s = 120 days
 - Store that amount of data?
- **Growing demand to leverage data to do useful tasks**
- **Parallelism: Divide and conquer**
- **Standard/Commodity and affordable architecture emerging**
 - Cluster of commodity Linux nodes (CPU, memory and disk)
 - Gigabit ethernet interconnect

Design Goals for Parallelism

1. Scalability to large data volumes:

- Scan 1000 TB (1PB) on 1 node @ 50 MB/s = 120 days
- Scan on 10000-node cluster = 16 minutes!
- $10^{15} \times 10^{-8} \times 10^{-5} = 10^2$ days = 10^7 seconds
- 10^7 seconds X 10⁴ nodes = 10³ seconds = 16 minutes
- 100 Gig per node (10¹¹ bytes at 10⁸ / second)

2. Cost-efficiency:

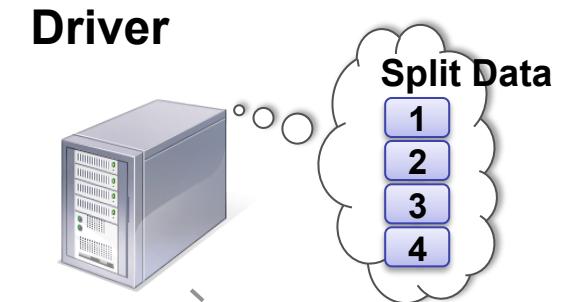
- Commodity nodes (cheap, but unreliable)
- Commodity network
- Automatic fault-tolerance (fewer admins)
- Easy to use (fewer programmers)

Command Line: Divide and Conquer

- Partitions the data
- The framework processes the objects within a partition in sequence, and can process multiple partitions in parallel

Partition and distribute data

Challenges?



Slave Processors or Computers
Executors

-
- **Challenge**
 - State the challenge
 - Tools and rules
 - Timeout to solve

-
- We are going to hack this up using a unix-based divide and conquer strategy
 - AKA poorman's distributed computational framework

Ground Rules

- **Limit ourselves to using the following Unix commands only:**
 - split, grep, wc, merge, cat, for, echo
 - cut, paste and bc #to get a total
 - |, &, wait #parallel computing

grep command

- **Problem: Need to search very large file**
 - Ex: Your java applet records each user that logs in to your website
 - Assume the log file generated is large (everyone wants to see your website!)
 - You want to look at only the lines in log file that contain ‘mcorliss’
- **In unix, use ‘grep’ command**
 - ‘grep <string> <filename>’ returns all lines in file that contain string
 - If string contains ‘ ‘ (space) then need to enclose in quotes
 - <string> is case-sensitive: “mcorliss” != “Mcorliss”

```
prompt$ grep "mcorliss" logfile
Jan 10 10:06:54 log in by mcorliss fr
Jan 12 11:36:22 log in by mcorliss fr
```

- What if we want only lines with “mcorliss” on January 10?

• • •

Regular Expressions

- **Fortunately, grep supports more powerful matching**

- Using regular expressions
 - ‘grep <reg. exp.> filename’

- **For example, to solve previous problem:**

```
prompt$ grep "Jan 10.*mcorliss" logfile
```

```
Jan 10 10:06:54 log in by mcorliss from 66.94.234.13
```

- **Searching for songs whose title starts with “love”**

```
prompt$ grep "^love" musiclibrary.txt
```

- **Searching for a 7-digit phone number in a file:**

```
prompt$ grep "[0-9]\{3\}[-]\{1\}[0-9]\{4\}" file
```

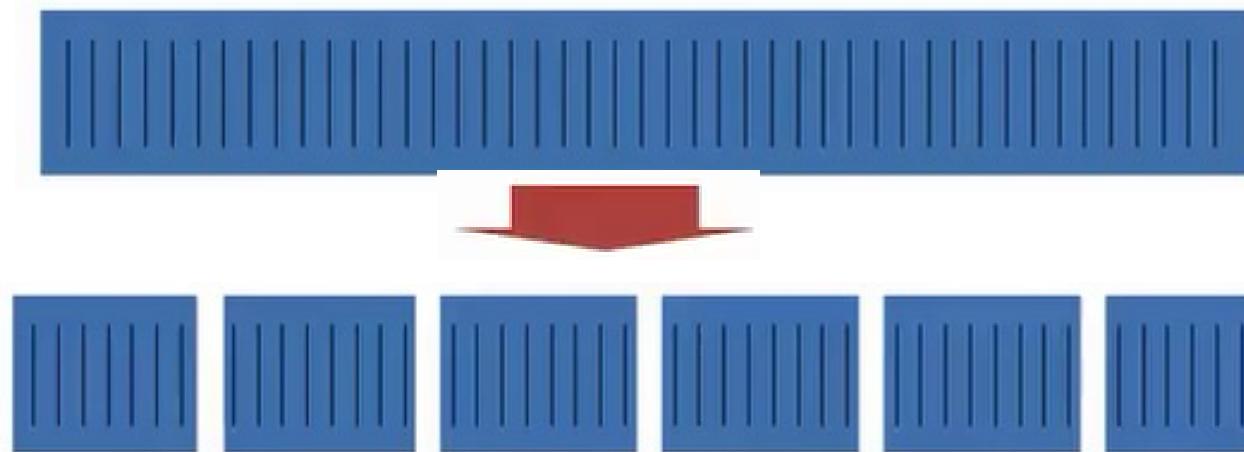
Pipe command “|”: Multiple Matches

- **Regular expressions are powerful**
 - But sometimes want to do multiple independent matches
 - For example, match on lines that contain ‘foo’, ‘goo’, and ‘zoo’
 - Trying to do this using regular expressions, would be very painful
- **Can use pipes denoted as | in Unix:**

```
prompt$ grep foo file | grep goo | grep zoc
```

split a large file into chunks/folds

- **split**
 - `cat logfile.txt | split -l 20 -d fold`
divide `logfile.txt` into files of 20 lines apiece, using “`fold`” as the prefix and with numeric suffixes
- **split -l 30 #30 lines per file**
- **split -b 24m #24 meg chunks**



Unix “split” and “cat” command

Split the file based up on the number of lines

Split the file into multiple pieces based up on the number of lines using -l option as shown below.

```
$ wc -l testfile  
2591 testfile  
$ split -l 1500 testfile importantlog  
$ wc -l *  
1500 importantlogaa  
1091 importantlogab  
2591 testfile
```

divide testfile into files of 1500 lines apiece, using “importantlog” as the prefix and with suffixes “aa”, “ab” in this case

To split *filename* to parts each 50 MB named *partaa*, *partab*, *partac*,....

Split
and
Cat

```
split -b50m filename part
```

To join the files back together again use the *cat* command

```
cat xaa xab xac > filename
```

or

```
cat xa[a-c] > filename
```

Split into chunks, wc

- **split**
 - `cat file.txt | split -l 20 -d fold`
divide file.txt into files of 20 lines apiece, using “fold” as the prefix and with numeric suffixes
- **WC**
 - WC is a counting utility
 - `wc -[l|c|w] file.txt`
counts number of lines, characters, or words in a file

`>>>$ wc -l setup.py`

`271 setup.py`

`271 lines in the setup.py file`

For loop in Unix + echo

- `for f in *.txt; do
 echo "file >> $f";
 #cat '$f' | wc -l ;
done`

```
setup.py  
JAMES-SANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ ls  
CHANGES.txt      LICENSE.txt      build          mlpv  
COPYRIGHT.txt    PKG-INFO       docs           setup.py  
INSTALL.txt     README.txt     gpl-3.0.txt  
JAMES-SANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ for f in *.txt; do      echo "$f" ; done  
CHANGES.txt  
COPYRIGHT.txt  
INSTALL.txt  
LICENSE.txt  
README.txt  
gpl-3.0.txt  
JAMES-SANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$
```

Get total using cut, paste and bc

```
--  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ seq 10  
1  
2 seq generates a sequence of numbers  
3  
4  
5  
6  
7  
8  
9  
10  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ seq 10 | paste -sd+ - | bc  
55  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ seq 10 >tmpp  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ cat tmpp
```

```
1  
2  
3 cat tmpp | cut -f 1 | paste -sd+ - | bc  
4  
5  
6 JAMES-SHANAHANS-Desktop-Pro-2:Notebooks jshanahan$ bc <<< "2+2"  
7 4  
8  
9  
10
```

```
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ cat tmpp|cut -f 1|paste -sd+ - | bc  
55  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ 1+2+3...+10|bc
```

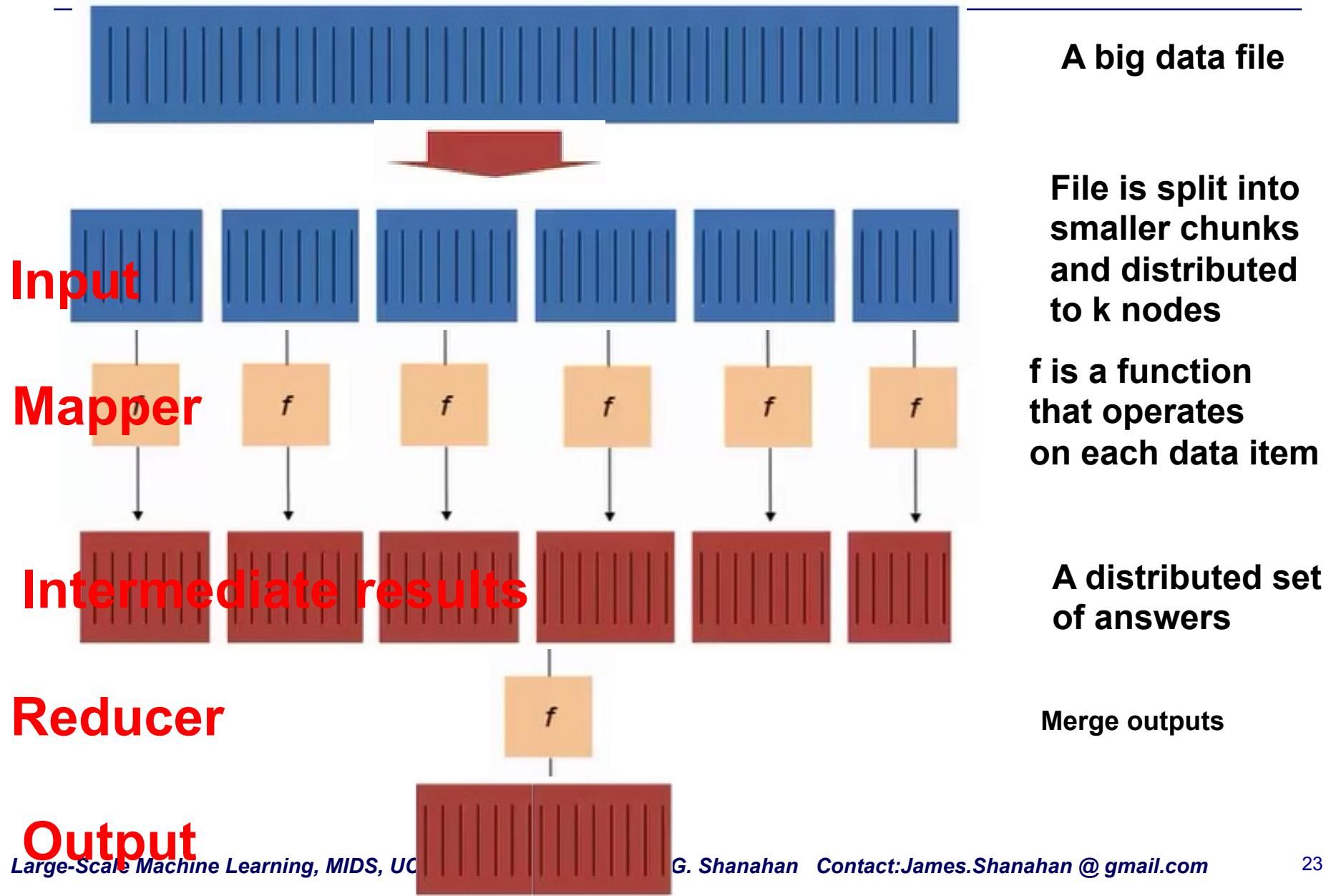
Parallel computing with “&” and wait

- The “**&**” causes parent process to spawn off parallel processes...
- And **wait** will cause the parent process to wait until child processes have finished

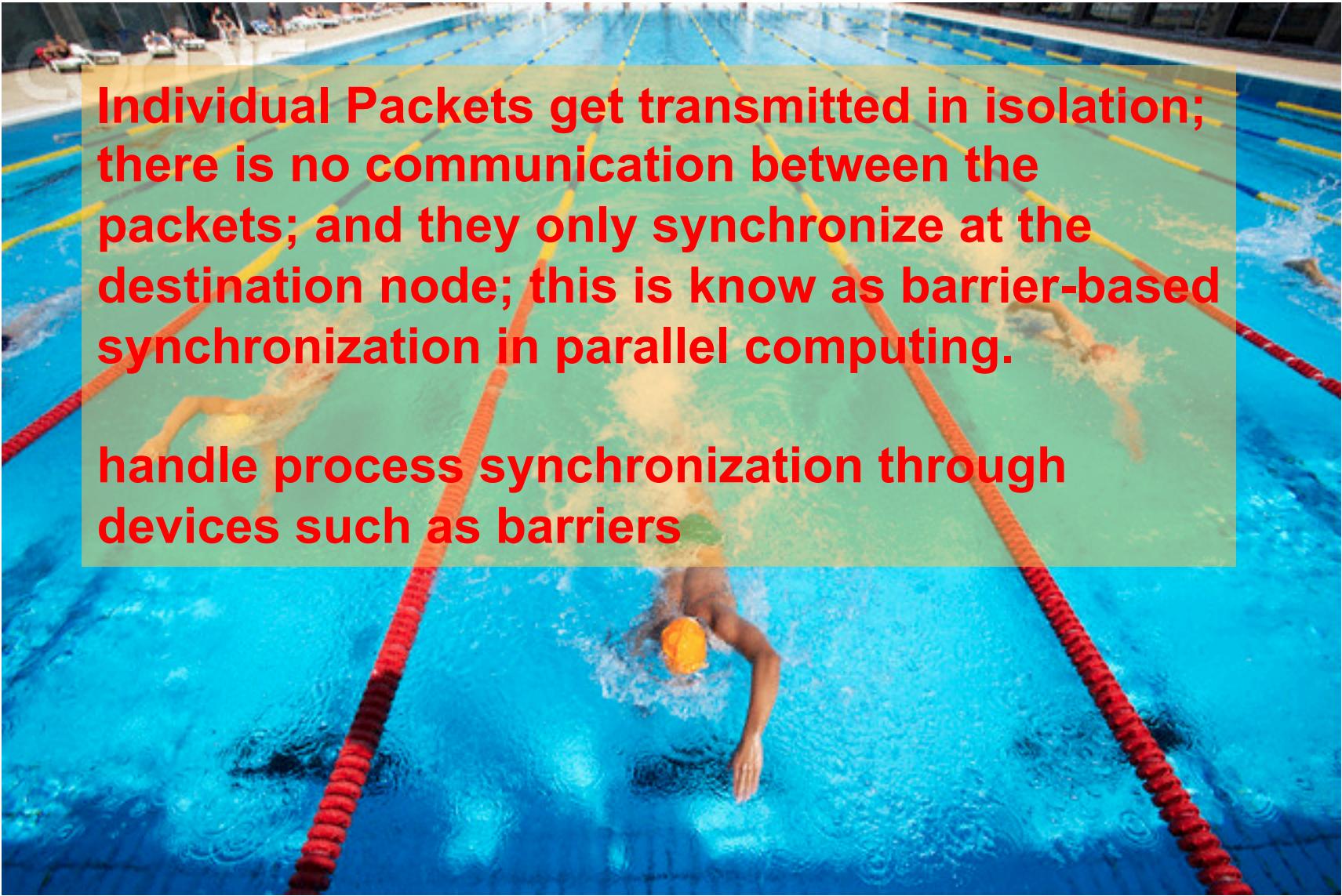
```
1 JAMES-SHANAHANS-Desktop-Pro:~ jshanahan$ seq 10000000|wc & Child proc
2 [1] 72233 takes time
3 JAMES-SHANAHANS-Desktop-Pro:~ jshanahan$ wait; echo "Finished waiting"
4 10000000 10000000 113888814
5 [1]+ Done seq 10000000 | wc
6 Finished waiting
7 JAMES-SHANAHANS-Desktop-Pro:~ jshanahan$ █ Prints only after wa
```

- Here **seq 1000000|wc &** causes the parent terminal process to spawn off a process to do this line count task.
- Meanwhile the next command “**wait**” causes the shell to wait until the child process (**seq 1000000|wc**) finishes. Once it finishes the shell can continue and run the echo command

Schematic of Parallel Processing



Individual Packets get transmitted in isolation

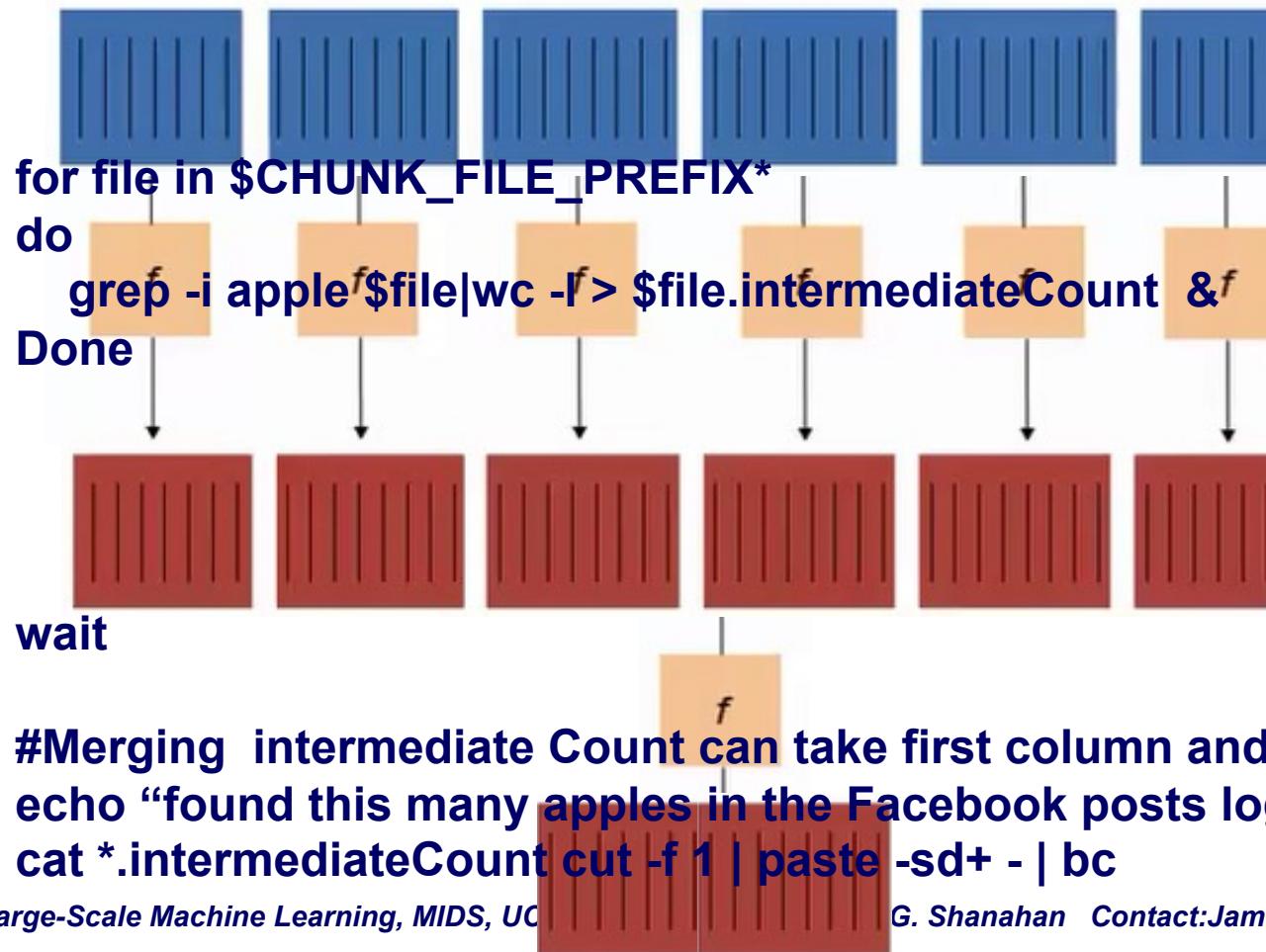


Individual Packets get transmitted in isolation; there is no communication between the packets; and they only synchronize at the destination node; this is known as barrier-based synchronization in parallel computing.

handle process synchronization through devices such as barriers

Schematic of Parallel Processing

1. #Splitting \$ORIGINAL_FILE into chunks ...
2. split -b 10000M \$ORIGINAL_FILE \$CHUNK_FILE_PREFIX



A big data file

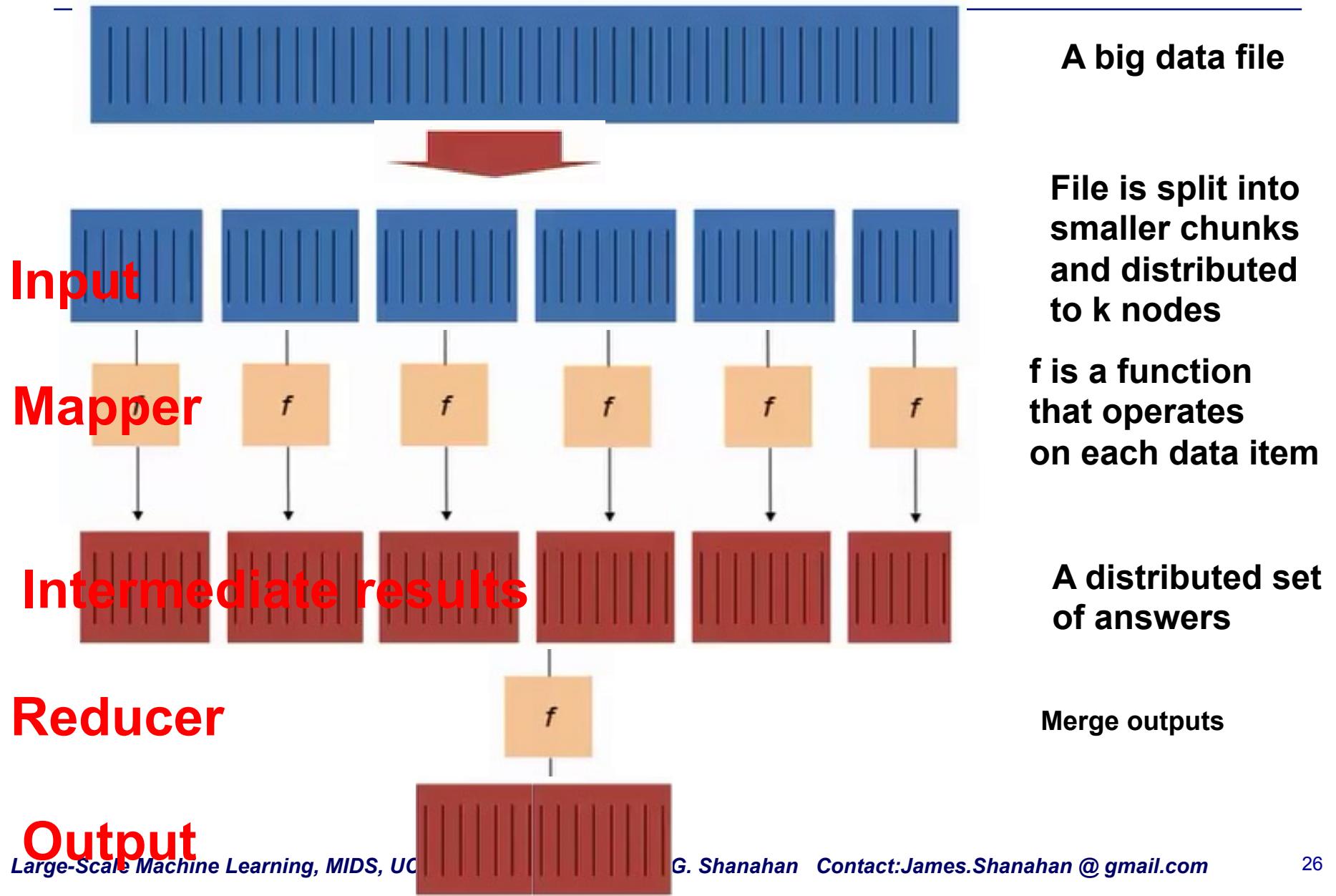
File is split into smaller 100Gig chunks and distributed to k nodes

f is a function that operates on each data item

A distributed set of answers

Merge outputs

Schematic of Parallel Processing



1. ORIGINAL_FILE=\$1

Solution

2. CHUNK_FILE_PREFIX=\${ORIGINAL_FILE}.split.

3. SORTED_CHUNK_FILES= pGrepCount
\$CHUNK_FILE_PREFIX*.sorted

4. usage ()

5. {

6. echo Parallel grep

7. echo usage: pGrepCount file1 100

8. echo greps file file1 for a “apple” and counts
the number of lines

Mapper

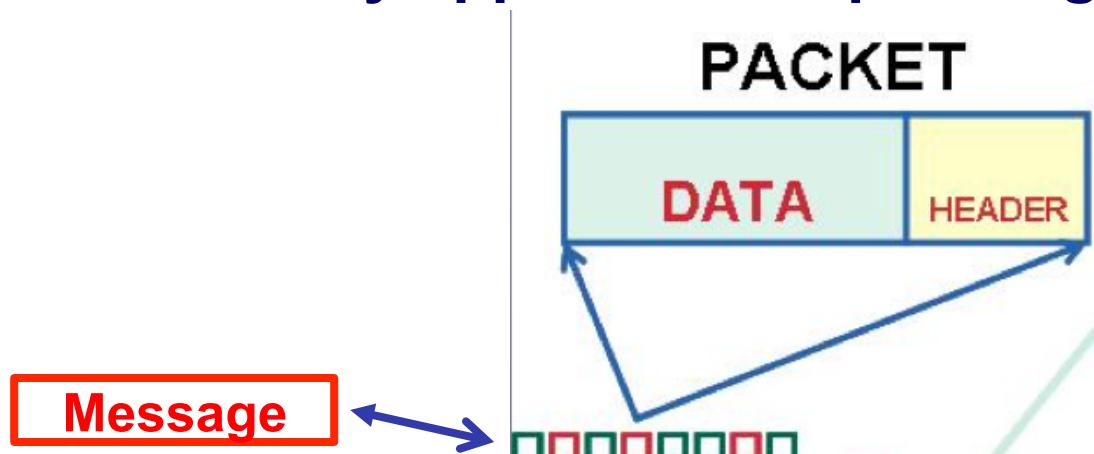
9. echo Note: file1 will be split in chunks up to
10Gig Chunks

10. echo and each chunk will be grepCounted in
parallel

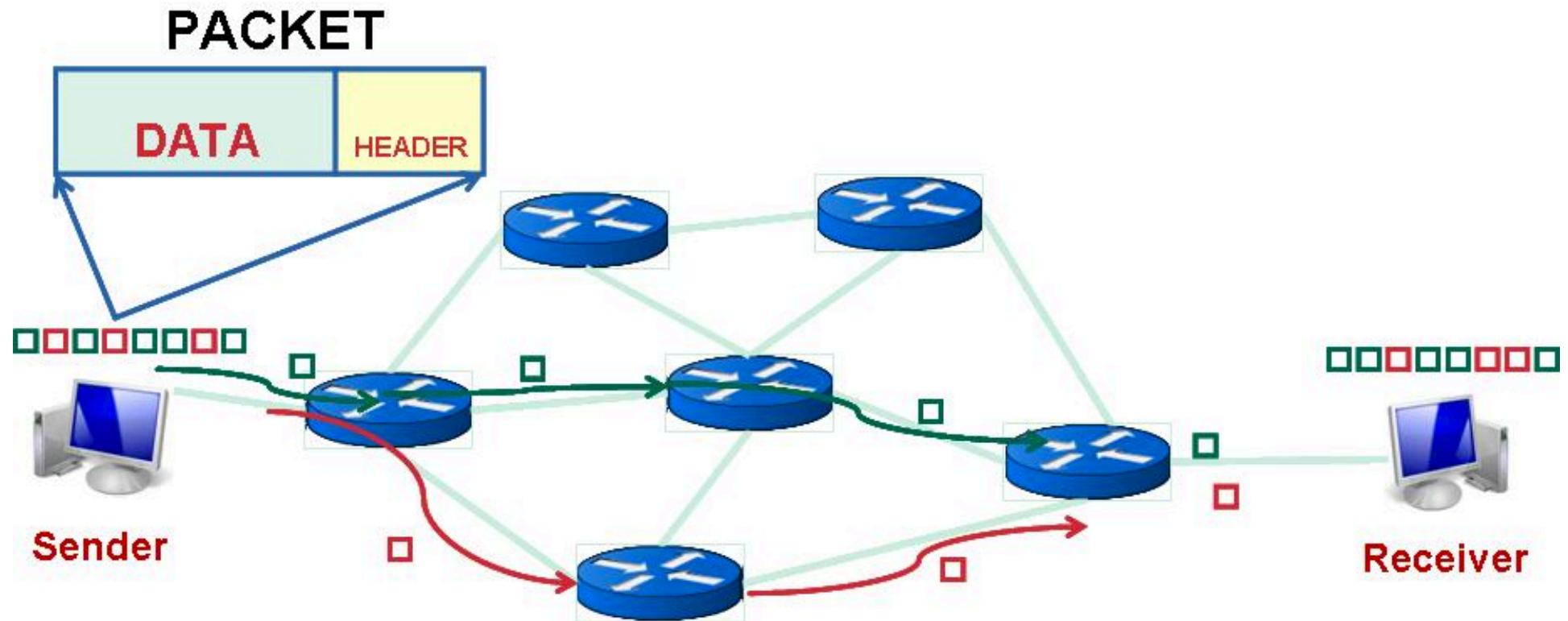
Reducer

D&C is common: e.g., Packet switching

- Packet switching is a digital networking communications method that transmits messages in chunks or blocks, called *packets*
- Packets are transmitted via a medium that may be shared by multiple simultaneous communication sessions.
- Packet switching increases network efficiency, robustness and enables technological convergence of many applications operating on the same network.

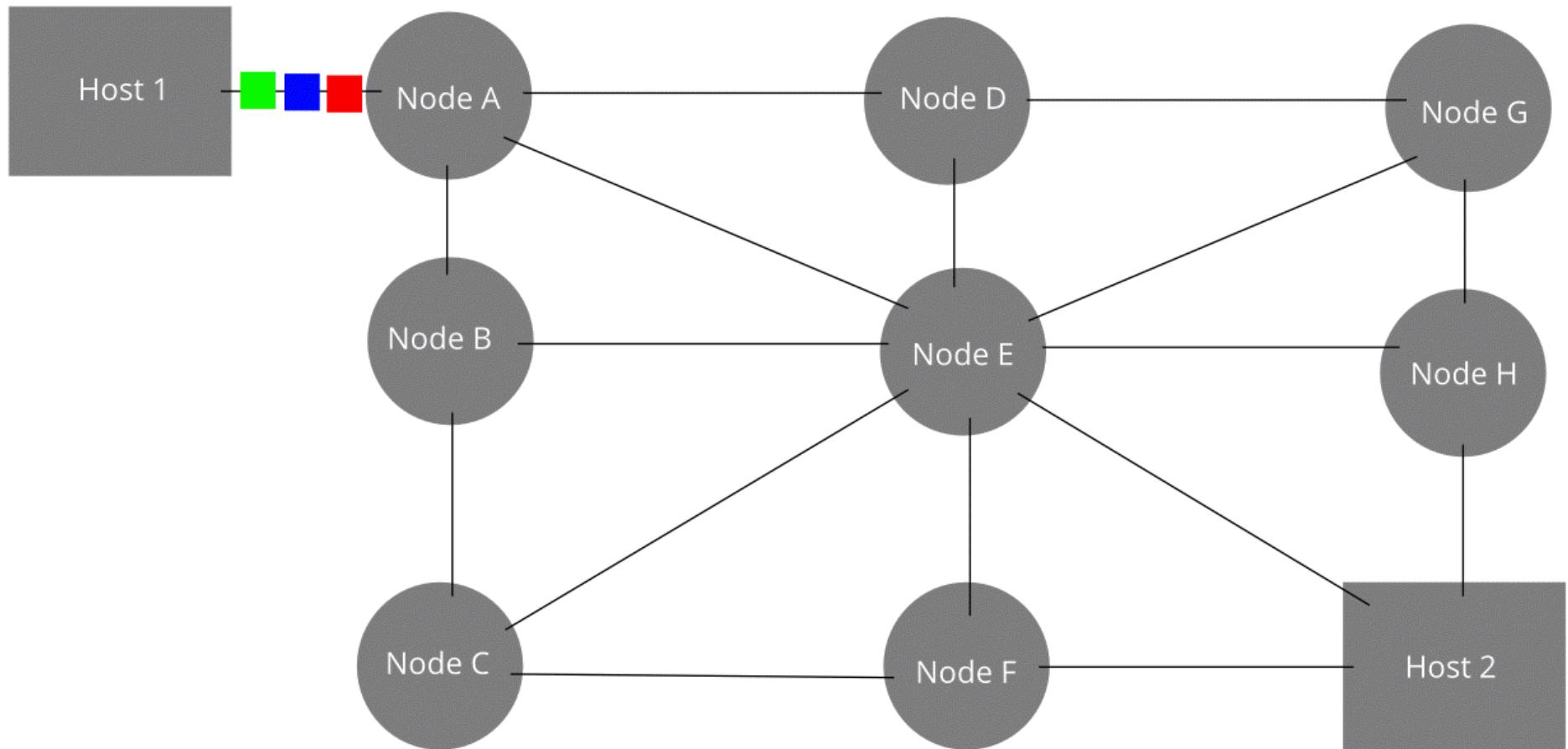


Packet = Header and Payload



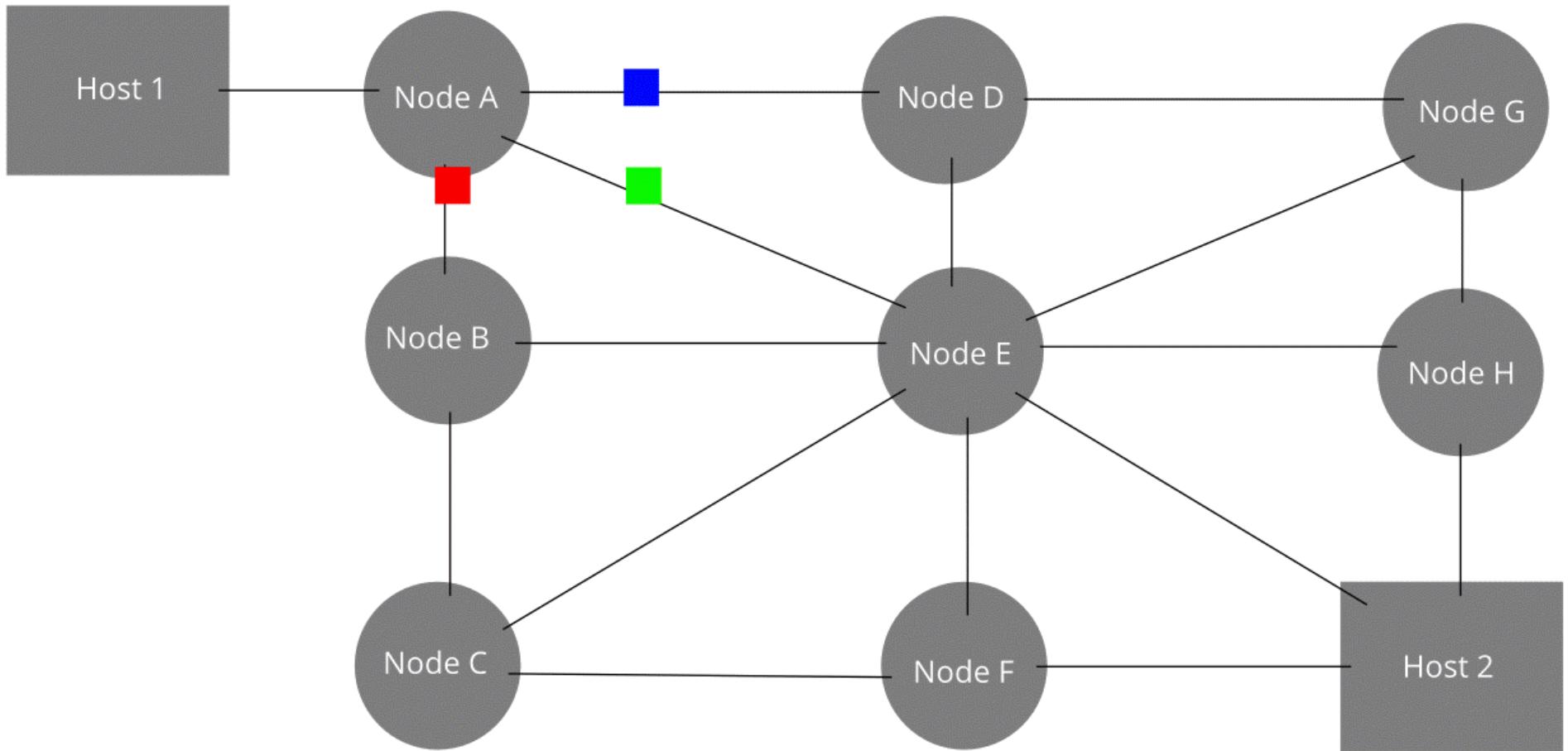
Message Passing: Divide and conquer/send

The original message is Green, Blue, Red.



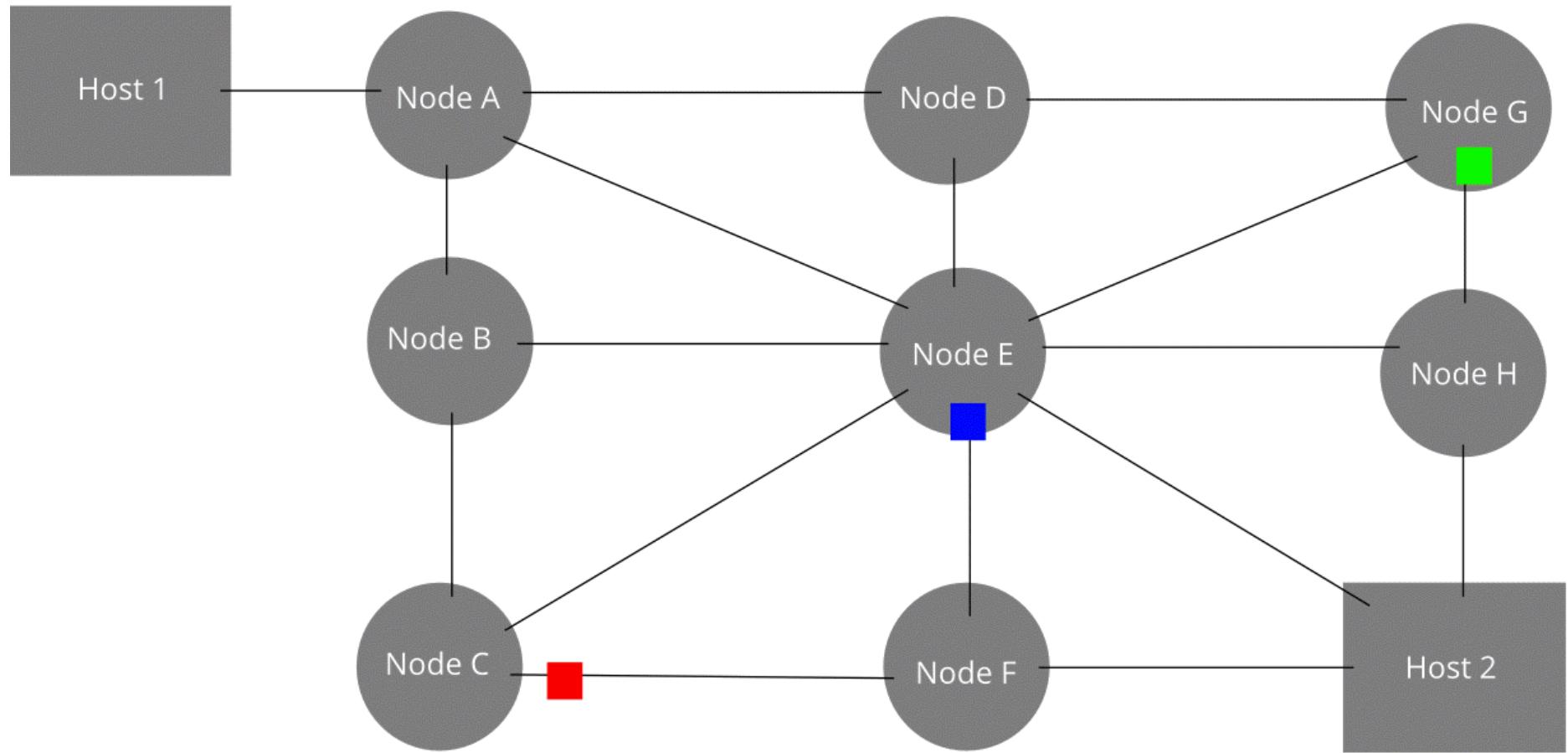
Route packets in parallel if possible

The data packets take different routes to their destinations.



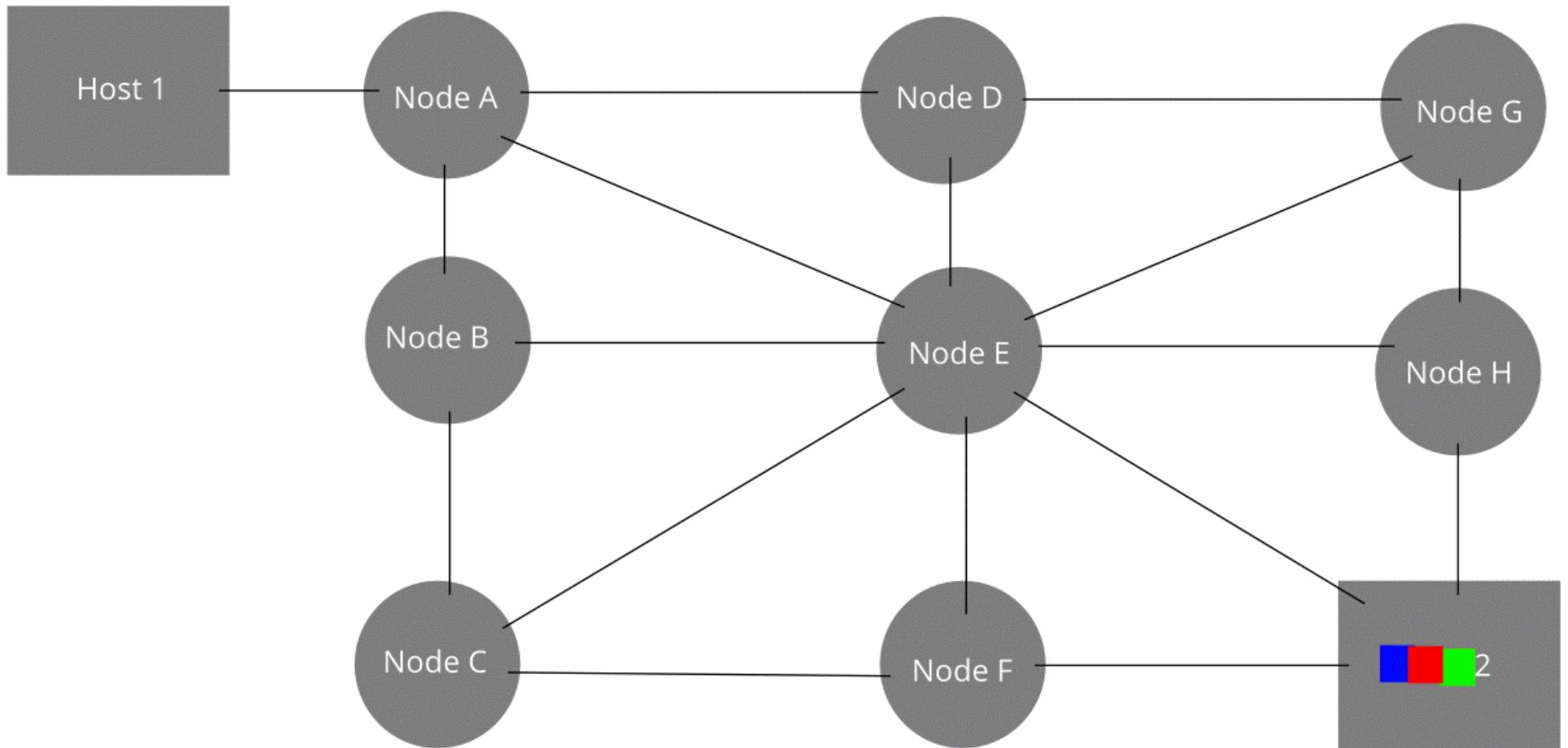
Routing packets in parallel if possible

The data packets take different routes to their destinations.



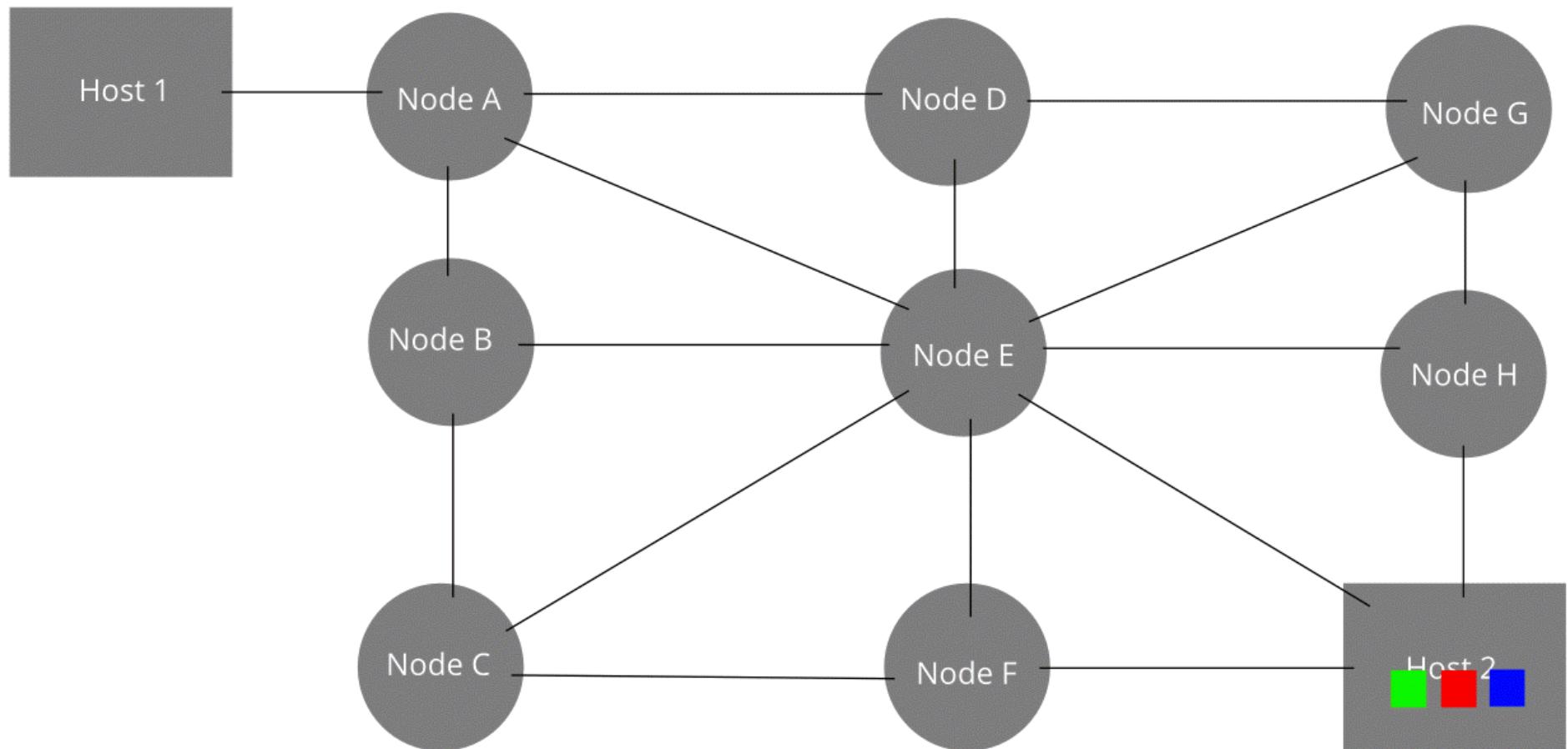
Barrier sync: wait until all packets arrive

The data packets take different routes to their destinations.



Barrier sync: reassemble payload

The received message is Green, Red, Blue



Individual Packets get transmitted in isolation



-
- **The key to success here was Divide and Conquer**
 - **Decompose a large task into smaller ones**
 - **We came up with a very nice framework for parallelizing tasks on the command line!**
 - But it is limited
 - Granularity of task is somewhat coarse
 - No fault tolerance
 - No control over shared filespace
 - **Divide and conquer does not come for free: there are obligations in terms of communication, synchronization, and fault tolerance**

Issues to be addressed

- ▶ How to break large problem into smaller problems? Decomposition for parallel processing
- ▶ How to assign tasks to workers distributed around the cluster?
- ▶ How do the workers get the data?
- ▶ How to synchronize among the workers?
- ▶ How to communicate with works?
- ▶ How to share partial results among workers?
- ▶ How to do all these in the presence of errors and hardware failures?

Divide and conquer does not come for free: there are obligations in terms of communication, synchronization, and fault tolerance

Parallel computing, MapReduce, Hadoop

- • Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)
- Sync time
 - Install Hadoop; run locally; run in the cloud?

V4

-
- **Background on Parallel computing**
 - General principles and concepts
 - **Map Reduce**
 - **Hadoop**

Serial computation versus parallel

- **Traditionally, computer software has been written for serial computation.**
 - To solve a problem, an algorithm is constructed and implemented as a serial stream/list of instructions.
 - These instructions are executed on a central processing unit on one computer. Only one instruction may execute at a time—after that instruction is finished, the next is executed.
- **Parallel computing, on the other hand, uses multiple processing elements simultaneously to solve a problem.**
 - This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others.
 - The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

Parallel computing

- Parallel computing is a form of computation in which many calculations are carried out simultaneously
- Operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

Parallel computers

- Parallel computers can be roughly classified according to the level at which the hardware supports parallelism,
 - Single computer
 - with multi-core and multi-processor computers having multiple processing elements within a single machine,
 - Clusters of computers
 - while clusters, MPPs, and grids use multiple computers to work on the same task.
 - Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks, e.g. GPUs for graphics processing.

http://en.wikipedia.org/wiki/Parallel_computing

Challenges of group work

- **Division of work**
- **Coordination costs (partition problem, communicate)**
 - Coordination costs represent time and energy that group work consumes that individual work does not,
 - They include the time it takes to coordinate schedules, arrange meetings, meet, correspond, make decisions collectively,
 - Integrate the contributions of group members, etc.
 - The time spent on each of these tasks may be small, but together they are significant.
- **Coordination costs can't be eliminated**
- **Synergy, Motivation also costs**
- **Similar challenges exist in distributed computing**



Parallel computer programs are challenging

- *Distributed computation is similar to group work*

Single Node

- Parallel computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common.
- Communication and synchronization between the different subtasks are typically some of the greatest obstacles to getting good parallel program performance.

Clusters

- Similar issues arise in clusters of computers

Developer Responsibilities

- In traditional parallel or distributed programming environments, the developer needs to explicitly address many (and sometimes, all) of the above issues
 - Concurrency, communication synchronization
 - (just like we did earlier in our cmdline framework).
- In shared memory programming,

Shared memory – the developer needs to explicitly coordinate access to shared data structures through synchronization primitives such as mutexes,

Process – to explicitly handle process synchronization through devices such as barriers,

- and to remain ever vigilant for common problems such as deadlocks and race conditions

Without Synchronization

- **Thread A**
 - Read variable X
 - compute X+1
 - Assign to X
- **Thread B**
 - Read variable X
 - compute X+1
 - Assign to X

Shared Variable X

Without Synchronization → Race Condition

- Depending on order, final value can be different
- **CASE 1: $x = x+1$**
 - A and B could both read the initial value of X and then overwrite each other
 - FINAL RESULT: $x = x+1$
- **CASE 2: $x = x+2$**
 - A reads and writes, then B reads and writes
 - FINAL RESULT: $x = x + 2$
- **That's a race condition**
 - and you will grow to hate them
 - if you don't already

Use synchronization to avoid race conditions

- **Use synchronization to avoid race condition**
- **There are several ways to synchronize and avoid race conditions**
 - Depending on the level of parallelism possible for the problem in hand
 - Mutex
 - Barrier

Avoid race conditions thru syncing using a mutex

- **Simplest is mutex**

- mutually exclusive

In computer science, mutual exclusion refers to the requirement of ensuring that no two concurrent processes[a] are in their critical section at the same time; it is a basic requirement in concurrency control, to prevent race conditions.

- In computer programming, a mutex is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously.
 - When a program is started, a mutex is created with a unique name.

- **Usually a mutex is associated with a variable or code block**

- `mutex_t count_lock;`
 - `mutex_lock(&count_lock);` #If you want to write to a variable, you need to own the lock first
 - Modify count variable
 - `mutex_unlock(&count_lock);` # release lock

Onus is on the Programmer

- **If you**
 - Alter a variable without a lock
 - Enter a code block without a lock
- **You won't know it's happening until you see the side effects**
 - at random intervals in a totally non-deterministic way

Mutexes can lead to Deadlock

- **Next issue: mutex base syncing can cause deadlocks**
- **Thread A and B both need locks for variables X and Y**
 - A acquires mutex_X
 - B acquires mutex_Y
 - A waits for mutex_Y to be free
 - B waits for mutex_X to be free

Mutexes can lead to Deadlock



Source: Ricardo Guimarães Herrmann

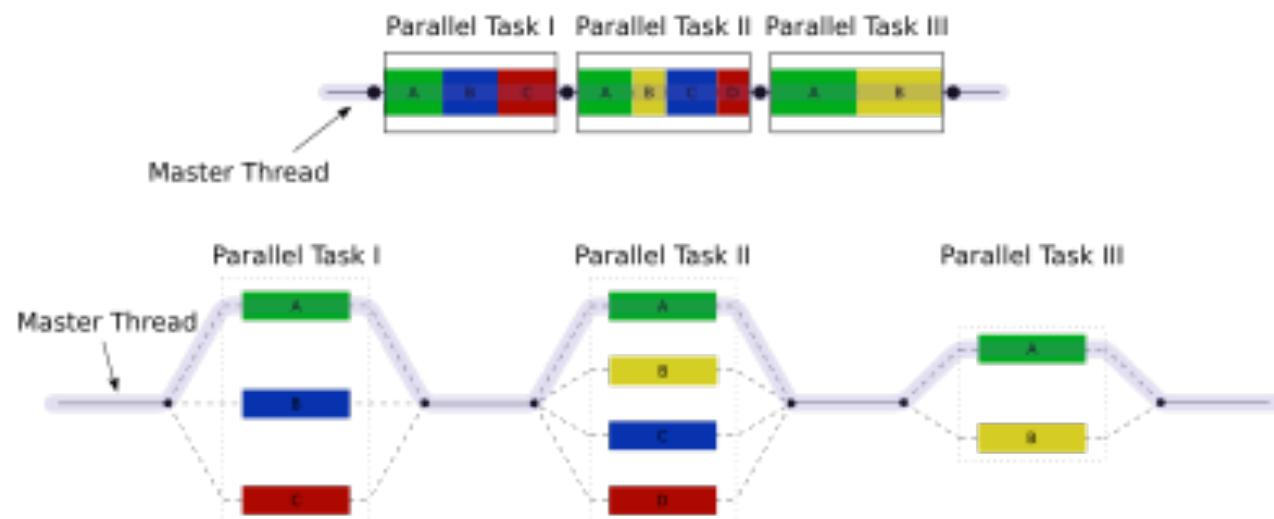
Deadlock

- **Thread A and B both need locks for variables X and Y**
 - A acquires mutex_X
 - B acquires mutex_Y
 - A waits for mutex_Y to be free
 - B waits for mutex_X to be free

Join pattern for synchronization

Communication/synchronization is key

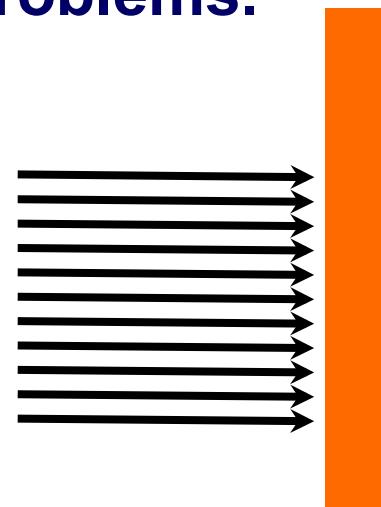
- Those problems that can be decomposed into independent subtasks, requiring no communication/synchronization between the subtasks except a join or barrier at the end.



embarrassingly parallel

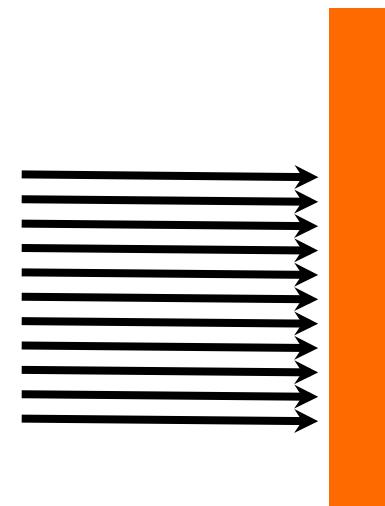
Synchronization thru a Barrier

- Another popular way of syncing is the barrier method;
- Explicitly handle process synchronization through devices such as barriers
- it is pretty effective, and very coarse grained and can be great in certain types of problems.



Synchronization thru a Barrier

- In parallel computing, a barrier is a type of synchronization method.
- A barrier for a group of threads or processes in the source code means any thread/process must stop at this point and cannot proceed until all other threads/processes reach this barrier.
- [http://en.wikipedia.org/wiki/
Barrier_\(computer_science\)](http://en.wikipedia.org/wiki/Barrier_(computer_science))



Shared memory programming

- **Onus on the programmer**
- **In shared memory programming, the developer needs as we have seen already**
 - to explicitly coordinate access to shared data structures through synchronization primitives such as mutexes,
 - to explicitly handle process synchronization through devices such as barriers,
 - and to remain ever vigilant for common problems such as deadlocks and race conditions.

Summary: Communication/Synchronization

- Those problems that can be decomposed into independent subtasks, requiring no communication/synchronization between the subtasks except a join or barrier at the end are very parallelizable (embarrassingly parallel)
 - Mutex pattern
 - Join pattern
 - Barrier pattern

Categories of Parallel Computation Tasks

- Applications are often classified according to how often their subtasks need to synchronize or communicate with each other.
- **Fine-grained parallelism**
 - An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; (share memory programming)
- **Coarse-grained parallelism**
 - It exhibits coarse-grained parallelism if they do not communicate many times per second,
- **Embarrassingly parallel**
 - An application is embarrassingly parallel if it rarely or never has to communicate. Divide and conquer. Summing a list of numbers.
 - Such applications are considered the easiest to parallelize.
 - Can be realized on a shared nothing architecture (see this shortly)

Examples of embarrassingly parallel problems

- An application is embarrassingly parallel if it rarely or never has to communicate
- Applications
 - Summing a list of numbers
 - Matrix multiplication
 - Lots of machine learning algorithms
 - Tree growth step of the random forest machine learning technique.
 - Genetic algorithms and other evolutionary computation metaheuristics.
 - Serving static files on a webserver to multiple users at once.
 - Computer simulations comparing many independent scenarios, such as climate models.
 - Ensemble calculations of numerical weather prediction.
 - Discrete Fourier Transform where each harmonic is independently calculated.
 - Many many more....

Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - BLT: embarrassingly parallel problems (3)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)

V4

BLT Question

- Embarassingly parallel?) [2 minutes]
- Question: What does "embarrassingly parallel" mean in the context of distributed systems? Give an example.
- Answer: An application is embarrassingly parallel if it rarely or never has to communicate. E.g., Summing a list of numbers.

Parallel computing, MapReduce, Hadoop

- • Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - BLT: embarrassingly parallel problems (3)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)

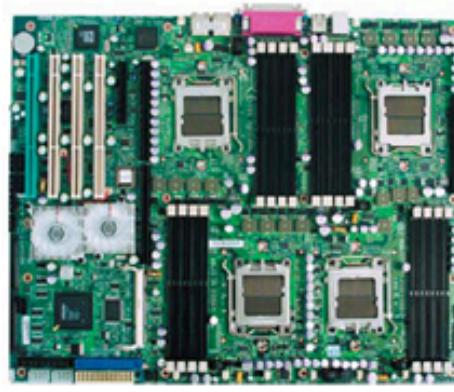
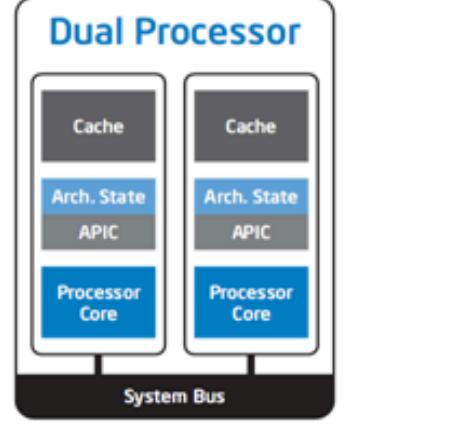
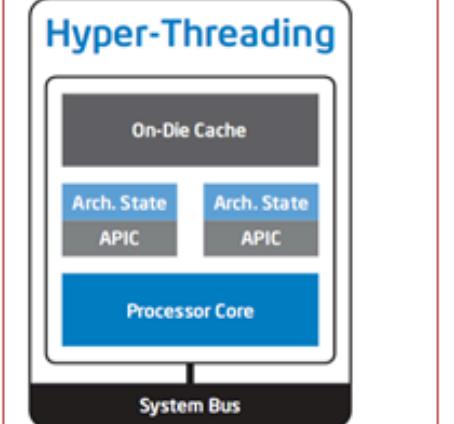
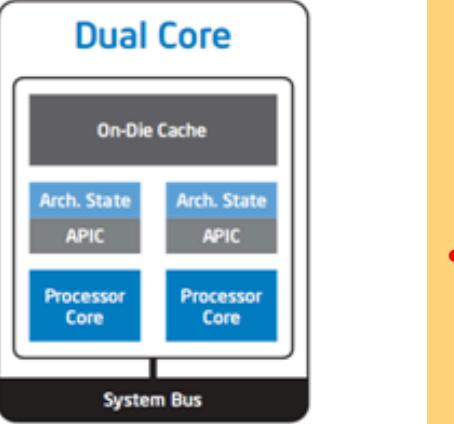
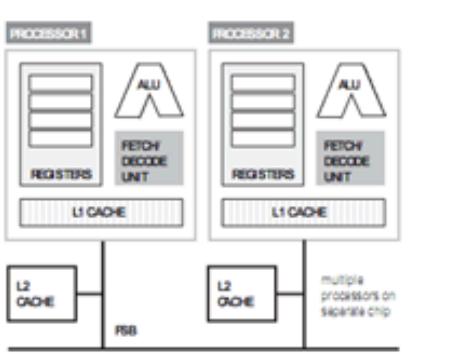
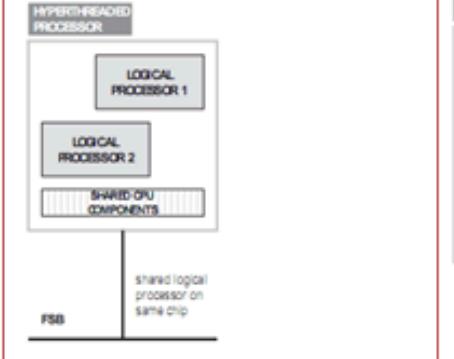
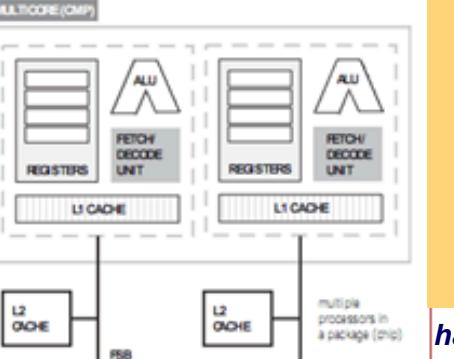
V4

-
- **In this section we look at different Architectures for Parallel Computation mainly from a hardware perspective**
 - **While in the next section we look at parallel architectures from a developers framework perspective**

CPUs: Multi-Processor vs Multi-Core

- When we think about computation we think in terms of CPU, memory and harddisks, and possibly network bandwidth
- **CPU (Central Processing Unit)**
 - A computer (motherboard) has multiple components for computation:
 - CPU (chip, memory, L1 and L2 caches)
 - GPU
 - A CPU, or Central Processing Unit, is what is typically referred to as a processor. A processor contains many discrete parts within it, such as one or more memory caches for instructions and data, instruction decoders, and various types of execution units for performing arithmetic or logical operations.
- **Multi-core CPU**
 - The CPU is NOT always busy, e.g., the GPU is off doing something while the CPU waits. A CPU can be better utilized by sharing with many tasks: In 2000, IBM first introduced dual-core CPU
 - A multi*core* CPU has multiple execution cores on one CPU, and shares memory also; may have its L2 caches
 - Now, this can mean different things depending on the exact architecture,
 - means that a certain subset of the CPU's components is duplicated, so that multiple "cores" can work in parallel on separate operations.
- **Cluster of computers**

Multiprocessor versus Multicore

Multi Processor (aka Multi CPU)	Hyper Threaded	Multi Core
 4 processors Board with 4 Processors: 		 Single processor Dual Core 
Dual Processor 	Hyper-Threading 	Dual Core 
	HYPERTHREADED PROCESSOR  multiple processors on same chip	MULTICORE (CPU)  multiple processors in a package (chip)

- In basic terms, a multicore processor is a single processor with multiple cores (dual-core has 2 cores, for example on the LHS)
- whereas a multiprocessor system contains more than one processor on the motherboard (which in turn can also be multicore).

iPhone 6: single CPU with dual cores

- iPhone has a single CPU with dual cores
- And a quadcore GPU
- iPad: 3-core CPU; 8-core GPU

Name	Model no.	Image	Semiconductor technology	Die size	CPU ISA	CPU	CPU cache	GPU	Memory technology	Introduced	Utilizing devices
A8	APL1011		20 nm HKMG ^[67]	89 mm ² ^[87]	ARMv8-A ^[66]	1.4 GHz dual-core Cyclone 2nd gen. ^[66]	L1i: 64 KB L1d: 64 KB L2: 1 MB L3: 4 MB ^[66]	PowerVR GX6450 (quad-core) @ 450 MHz (115.2 GFLOPS) ^{[88][89]}	64-bit Single-channel LPDDR3-1600 ^[67] (12.8 GB/sec) ^[85]	September 2014	• iPhone 6 • iPhone 6 Plus
A8X	APL1012		20 nm HKMG ^{[73][90]}	128 mm ² ^[73]	ARMv8-A	1.5 GHz triple-core Cyclone 2nd gen. ^[73]	L1i: 64 KB L1d: 64 KB L2: 2 MB L3: 4 MB ^[73]	PowerVR GX6850 (octa-core) @ 450 MHz (230.4 GFLOPS) ^{[73][90]}	64-bit Dual-channel LPDDR3-1600 ^[73] (25.6 GB/sec) ^[85]	October 2014	• iPad Air 2

iPad, 3-core

iPad, 8-core

[http://en.wikipedia.org/wiki/
Apple_system_on_a_chip](http://en.wikipedia.org/wiki/Apple_system_on_a_chip)

Parallel Systems

- **Resources**
 - CPU
 - Memory
 - Disk
 - Network
- **There are many parallel architectures that are commonly used in practice:**
- **Sharing nothing to Sharing all**
- **I list three (3) here.**
 - Shared nothing
 - Shared Disc
 - Shared all

Taxonomy of parallel architectures

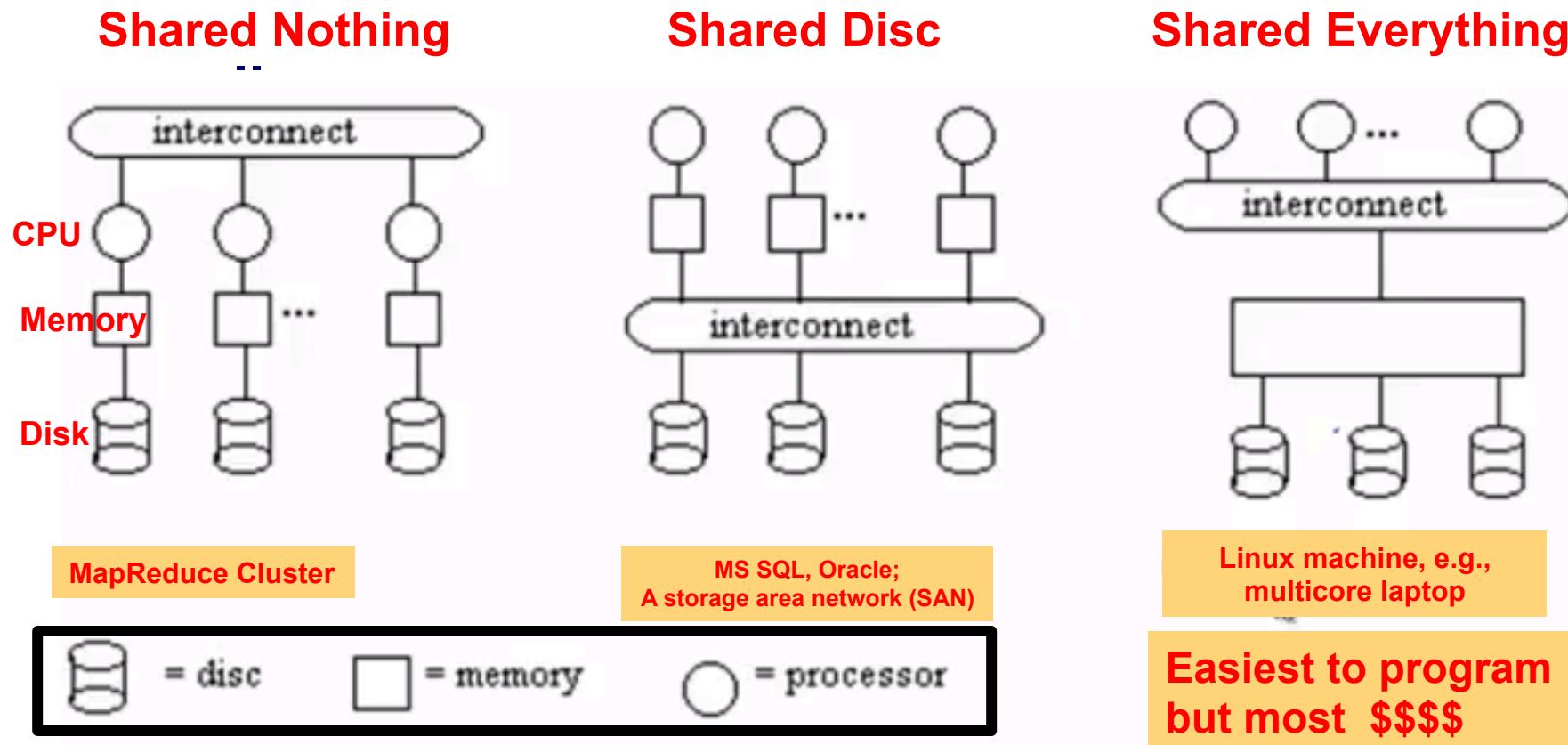


Fig. 3.1 Logical multi-processor database designs (diagram after [DEWI92])

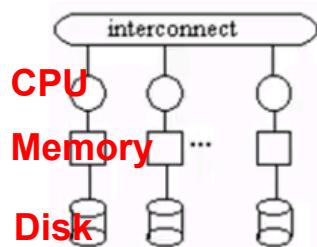
Shared memory programming

- **Onus on the programmer**
- **In shared memory programming, the developer needs as we have seen already**
 - to explicitly coordinate access to shared data structures through synchronization primitives such as mutexes,
 - to explicitly handle process synchronization through devices such as barriers,
 - and to remain ever vigilant for common problems such as deadlocks and race conditions.

Shared Nothing architecture

Scale out, linear scaling (cheap) versus scale up (expensive)

- **Shared Nothing: the machines are only connected by the network. (Scale out!)**
 - A shared nothing architecture (SN) is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system.
 - A “shared nothing” architecture means that each computer system has its own private memory and private disk.
 - The “shared nothing” architecture shares network bandwidth because it must transfer data from machines doing the Map tasks to machines doing the Reduce tasks over the network.
- **In order to achieve a good workload distribution MPP systems have to use a hash algorithm to distribute (partition) data evenly across available CPU cores.**
- **This architecture is followed by essentially all high performance, scalable, DBMSs, including MemSQL, Teradata, Netezza, Greenplum, Paraccel, DB2 and Vertica. It is also used by most of the high-end Internet companies, including Amazon, Akamai, Yahoo, Google, and Facebook.**



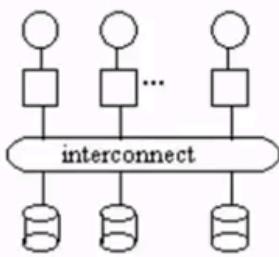
Shared nothing (SN) architecture is popular

- **Shared nothing is popular for web development because of its scalability.**
 - As Google has demonstrated, a pure SN system can scale almost infinitely simply by adding nodes in the form of inexpensive computers, since there is no single bottleneck to slow the system down.
- **Shard/partition data (e.g., distribute documents)**
 - Google calls this sharding.
 - A SN system typically partitions its data among many nodes on different databases (assigning different computers to deal with different users or queries), or may require every node to maintain its own copy of the application's data, using some kind of coordination protocol.
 - This is often referred to as database sharding.

Shared everything (disk or memory)

Scale up, vertical scaling (expensive) add more memory/disk

- Oracle RAC does not run on a shared nothing system. It was built many years ago to run on a “shared disk” architecture.
- In this world, a computer system has private memory but shares a disk system with other computer systems.
 - Such a “disk cluster” was popularized in the 1990’s by Sun and HP, among others. In the 2000’s this architecture has been replaced by “grid computing”, which uses shared nothing.
 - Shared disk has well-known scalability problems, when applied to DBMSs and is super expensive
 - SAN can cost \$500,000 and still struggle with terabytes of data



shared nothing versus shared something

- **Shared nothing architecture (SN)**
 - None of the nodes share memory or disk storage.
 - The “shared nothing” architecture shares network bandwidth
- **Shared something**
 - People typically contrast SN with systems that keep a large amount of centrally-stored state information, whether in a database, an application server, or any other similar single point of contention.
- **Shared nothing is popular**

Killer App is Large-Scale Data Systems (storage and processing)

- Many tasks process big data, produce big data
- Want to use hundreds or thousands of CPUs
 - ... but this needs to be easy
 - Parallel databases exist, but they are expensive, difficult to set up, and do not necessarily scale to hundreds of nodes.
- MapReduce is a *lightweight* framework, providing:
 - Automatic parallelization and distribution
 - Fault-tolerance
 - I/O scheduling
 - Status and monitoring

Web Search: Inverted Index

Document 1

The bright blue butterfly hangs on the breeze.

Document 2

It's best to forget the great sky and to retire from every wind.

Document 3

Under blue sky, in bright sunlight, one need not search around.

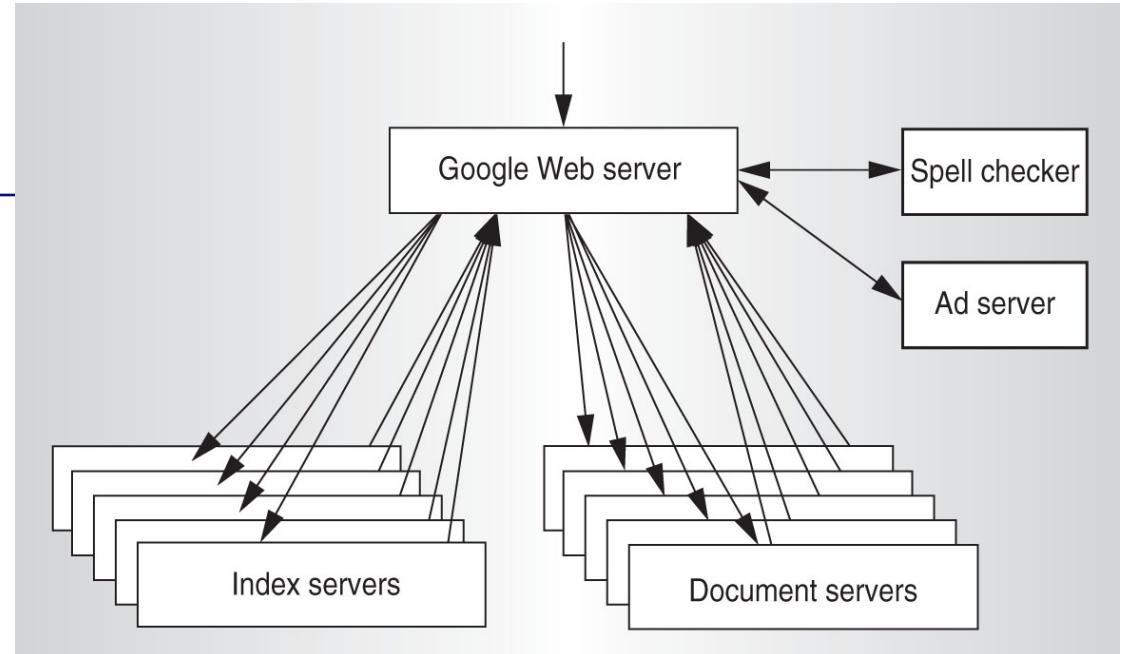
Stopword list

a
and
around
every
for
from
in
is
it
not
on
one
the
to
under

Inverted index

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

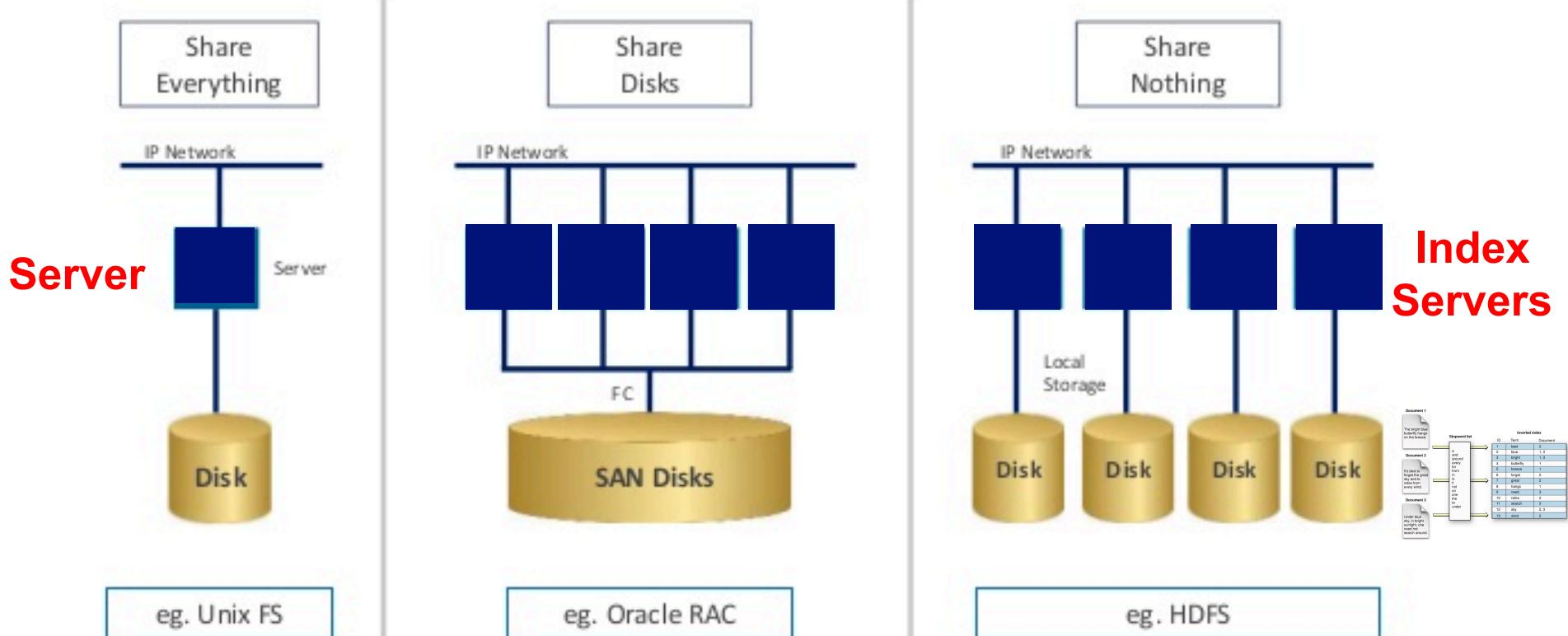
Web Search



- The docs, too, are partitioned into “shards” – the partitioning is a hash on the doc ID. Each shard contains the full text of a subset of the docs. Each shard can be searched by multiple computers – “doc servers”
- The GWS sends appropriate doc IDs to one doc server associated with each relevant shard
- When the dust has settled, the result is a URL, a title, and a summary for every relevant doc

SHARE NOTHING ARCHITECTURE

E.g., Web Search



The “shared nothing” architecture shares network bandwidth because it must transfer data from machines doing the Map tasks to machines doing the Reduce tasks over the network.

A “shared nothing” architecture means that each computer system has its own private memory and private disk.

Share Nothing Architecture

- [http://en.wikipedia.org/wiki/
Shared_nothing_architecture](http://en.wikipedia.org/wiki/Shared_nothing_architecture)
- [http://www.slideshare.net/PhilippeJulio/hadoop-
architecture](http://www.slideshare.net/PhilippeJulio/hadoop-architecture)

Parallel computing, MapReduce, Hadoop

- • Motivation for Parallel Computing (5)
- • Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- • Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- • Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- • MapReduce: Runtime Environment (5)
- • Hadoop 2.0 and what is Hadoop good at? (10)
 - Sync time

V4

BLT: QUESTION

What is shared in the "shared nothing" architecture?

- ..

- Nothing
- Memory
- Disk
- Network

-
- 3 minutes

BLT QUESTION: Answer

What is shared in the "shared nothing" architecture?

- ..

- Nothing

- Memory

- Disk

- Network

Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - BLT: embarrassingly parallel problems (3)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)

V4

-
- In the previous section we looked at different Architectures for Parallel Computation mainly from a hardware perspective
 - While in this section we look at parallel developer frameworks from a software perspective

Frameworks for Parallel Computation

Shared
Memory

Cluster

- Onus is on the Programmer (mutexes)
- Language extensions such as OpenMP can be used for shared memory parallelism
 - OpenMP has predefined locking paradigms
 - reduce the chance of deadlock
 - not foolproof (though possibly fool resistant)
- Cluster/Grid computing
 - Provide logical abstractions that hide details of operating system synchronization and communications primitives
 - MPI: Libraries implementing the Message Passing Interface (MPI) for cluster-level parallelism (same local network)
 - Map-Reduce: MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes); Cluster or grid (nodes are shared across geographically or heterogenous hardware)

MPI Distributed Computing Frameworks

- **Message Passing Interface (MPI) provides a powerful, efficient, and portable way to express parallel programs**
- **MPI is a specification for an API that allows many computers to communicate with one another.**
 - It is used in computer clusters and supercomputers.
 - MPI was created by William Gropp, Ewing Lusk and others.
- **MPI in short**
 - Hides the details of architecture
 - Hides the details of message passing, buffering
 - Provides message management services
 - packaging
 - send, receive
 - broadcast, reduce, scatter, gather message modes

MPI basics

MPI defines an environment where programs can run in parallel and communicate with each other by passing messages to each other. There are two major parts to MPI:

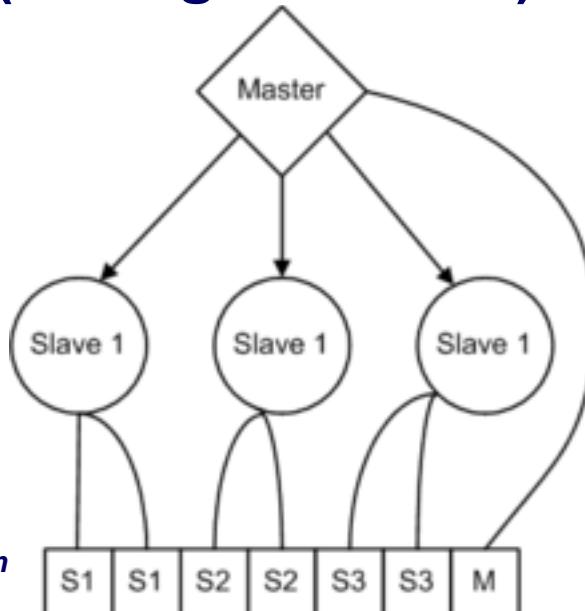
- ❶ the environment programs run in
- ❷ the library calls programs use to communicate

MPI is so preferred by users because it is so simple to use. Programs just use *messages* to communicate with each other. Can you think of another successful systems that functions using message passing? (the Internet!)

- Each program has a mailbox (called a queue).
- Any program can put a message into the another program's mailbox.
- When a program is ready to process a message, it receives a message from its own mailbox and does something with it.

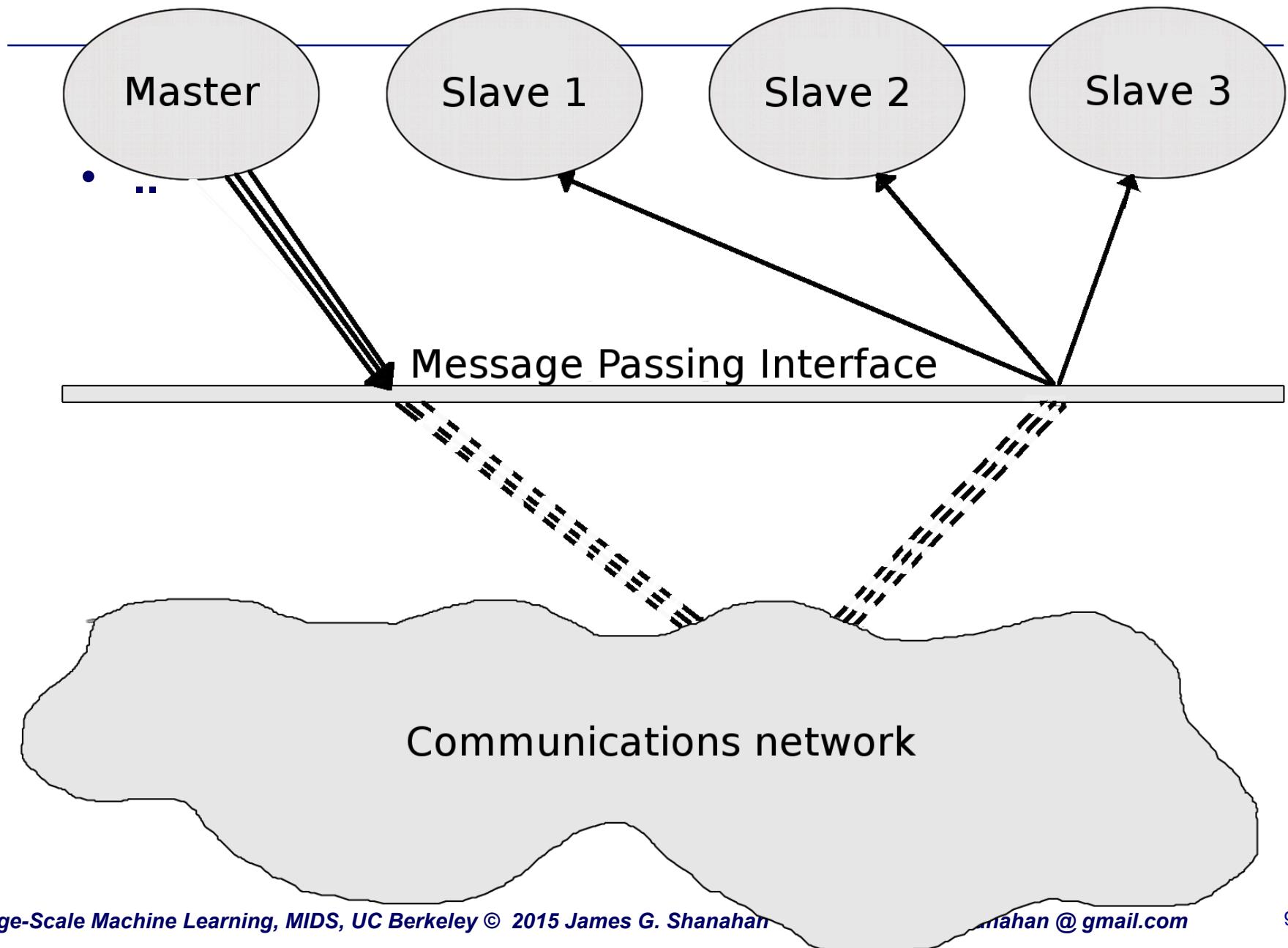
MPI: Master divides and sends data to workers

- Example Task: sum an array of numbers
- In this program, the master divides the array into sub-arrays S^i and sends these to each slave.
- Each slave will then calculate the sum of its sub-array. In the case where the size of the array is not an even multiple of the number of slaves, the master will finish the remaining work by calculating the sum of the last values of the array (see figure below).



- Master Divides data into subsets S^i and sends to workers
- Master also does some work on the leftover subset M

Example: 1 master, 3 slaves. The master sends information (data) to the slaves to work on.



Why not MPI for big data?

- **But...**
 - Data transfer.
 - Requires cooperation of sender and receiver
 - Cooperation not always apparent in code
- **Hard for programming**
 - Explicitly communicate between nodes via message passing
 - Synchronization is hard and error-prone
- **Not designed for fault tolerance**
- **Not designed for data locality**
 - Move data to computation nodes
 - Network bandwidth is limited
 - http://en.wikipedia.org/wiki/Message_Passing_Interface

Frameworks for Parallel Computation

Shared
Memory

Cluster

- Onus is on the Programmer (mutexes)
- Language extensions such as OpenMP can be used for shared memory parallelism
 - OpenMP has predefined locking paradigms
 - reduce the chance of deadlock
 - not foolproof (though possibly fool resistant)
- Cluster/Grid computing
 - Provide logical abstractions that hide details of operating system synchronization and communications primitives
 - MPI: Libraries implementing the Message Passing Interface (MPI) for cluster-level parallelism (same local network)
 - Map-Reduce: MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes); Cluster or grid (nodes are shared across geographically or heterogeneous hardware)

Data locality and isolation

- Various patterns for parallel computation have been proposed over the years.
- These patterns, when composed, can be used to express computations in a wide variety of domains.
- I list some here

Designs Patterns for parallelization

A straightforward way to express both data locality and isolation

[Structured Patterns: An Overview](#)

[Parallel Pattern 1: Superscalar Sequences and Task Graphs](#)

[Parallel Pattern 2: Selection](#)

[Parallel Pattern 3: Map](#)

[Parallel Pattern 4: Gather](#)

[Parallel Pattern 5: Stencil](#)

[Parallel Pattern 6: Partition](#)

[Parallel Pattern 7: Reduce](#)

[Parallel Pattern 8: Scan](#)

[Parallel Pattern 9: Pack](#)

Data locality and isolation

- **Scalable approaches to parallel computation have to take two things into account:**
 - tasks and data.
- **Many approaches to parallel computation over-emphasize the former (task) at the expense of the latter (e.g., MPI)**
- **However, in order to achieve scalable parallel performance, algorithms need to be designed in a way that emphasizes data locality and isolation**

Data locality and isolation:MAP-Reduce

- The MAP-Reduce pattern provides a simple way for the software developer to express both data locality and isolation

Extending For loops to maps (e.g., a set of numbers)

- A common pattern in sequential programming is iteration
- If we DISALLOW dependencies between loop iterations, we do get a type of computation that can be parallelized easily: the map pattern.
- In the map pattern, a set of loop indices are generated and independent computations are performed for every unique index.
- These computations are not allowed to communicate with one another.
- At the end of the map, however, there is an implied barrier, and then other operations can depend on the output of the map operation as a whole.

Map is a design pattern (along with Reduce)

- Map is a design pattern in parallel computing where a simple operation is applied to all elements of a sequence, potentially in parallel.^[1]
- It is used to solve embarrassingly parallel problems:
 - those problems that can be decomposed into independent subtasks, requiring no communication/synchronization between the subtasks except a join or barrier at the end.
- The map pattern is typically combined with other parallel design patterns. E.g., map combined with category reduction gives the MapReduce pattern.
- A straightforward way to express both data locality and isolation

[http://en.wikipedia.org/wiki/Map_\(parallel_pattern\)](http://en.wikipedia.org/wiki/Map_(parallel_pattern))

Map-Reduce Frameworks

- Some parallel programming systems, such as [OpenMP](#) and [Cilk](#), have language support for the map pattern in the form of a parallel *for* loop; languages such as [OpenCL](#) and [CUDA](#) support elemental functions (as "[kernels](#)") at the language level.
- Limitations of these frameworks,
 - Developers are still burdened to keep track of how resources are made available to workers.
 - Additionally, these frameworks are mostly designed to tackle **processor-intensive problems** and have only rudimentary support for dealing with very large amounts of input data
- Map-Reduce/Hadoop/Spark to the rescue!
 - Distributed data handling, and distributed computation, framework that hides system level details

Map-Reduce: move the code to the data

- **Distributed data handling and storage**
 - Terabytes and petabytes in size
 - Large-data processing by definition requires bringing data and code together for computation to occur no small feat for huge datasets!
 - MapReduce addresses this challenge by providing a simple abstraction for the developer, transparently handling most of the details behind the scenes in a scalable, robust, and efficient manner.
- **Distributed computation : Move the code to the data**
 - Like OpenMP and MPI, MapReduce provides a means to distribute computation without burdening the programmer with the details of distributed computing (but at a different level of granularity).
 - **BUT instead of moving large amounts of data around, it is far more efficient, if possible, to move the code to the data.**
 - This is operationally realized by spreading data across the local disks of nodes in a cluster and running processes on nodes that hold the data.
- **A straightforward way to express both data locality and isolation**

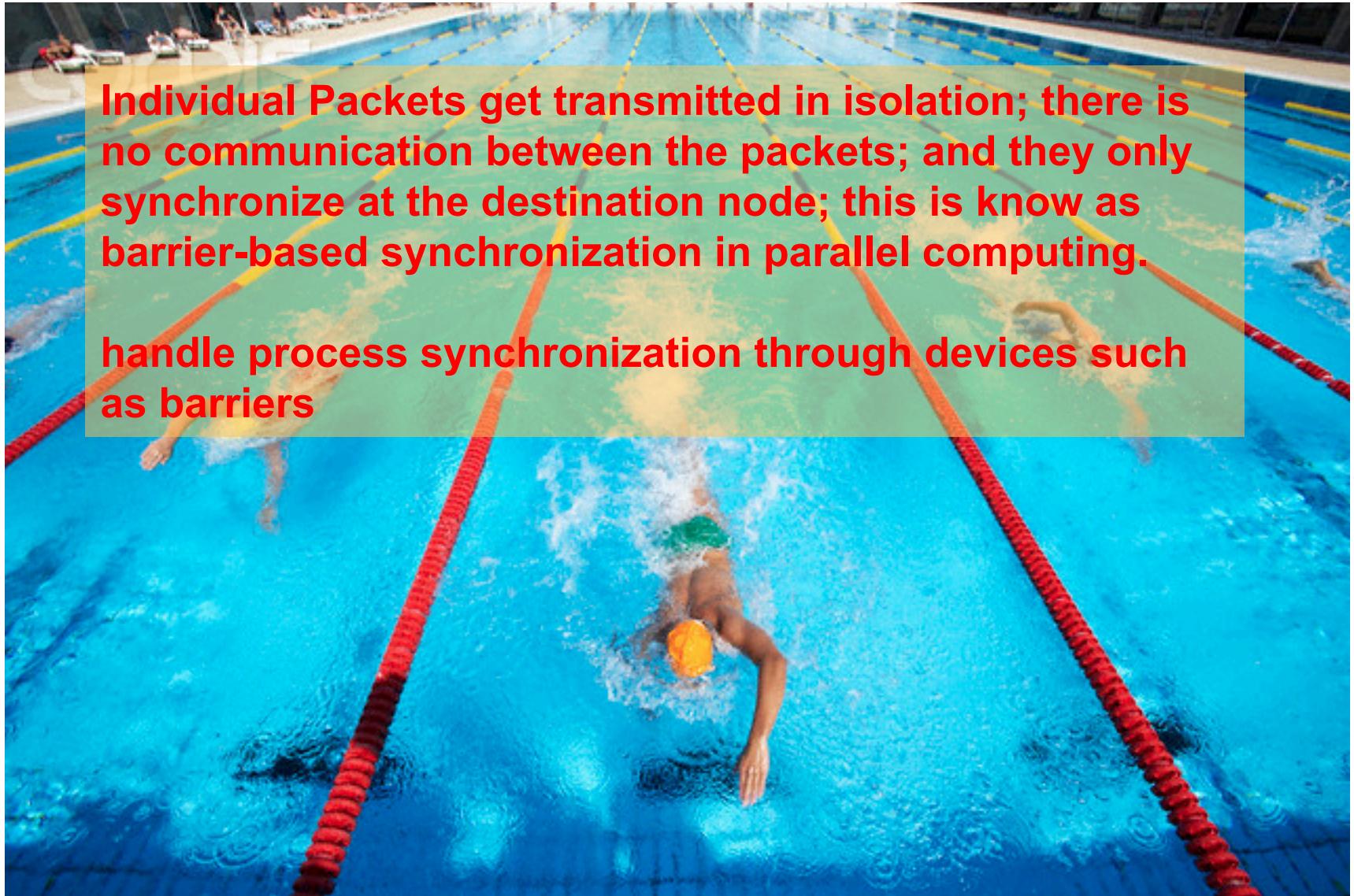
Map Reduce + Scalability and fault-tolerance

- **Inspired by functional programming**
 - The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms.
- **Scalability and fault-tolerance**
 - The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once.
- **Optimizing the communication**
 - Optimizing the communication cost is essential to a good MapReduce algorithm.
- **The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play.**

MapReduce frameworks

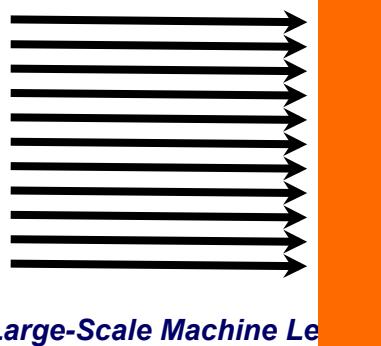
- **MapReduce libraries have been written in many programming languages, with different levels of optimization.**
- **A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop.**
- **The name MapReduce originally referred to the proprietary Google technology, but has since been genericized.**

Individual Packets get transmitted in isolation



How to Sync in MapReduce

- Independent computations are performed for every unique chunk of data.
- These computations are NOT allowed to communicate with one another.
- At the end of the map, however, there is an implied barrier, and then other operations can depend on the output of the map operation as a whole.
 - Barrier
 - Everybody waits for the last thread



The reducer phase waits Until all mapper threads are finished but it can start doing intermediate merge sorting of mappers that have finished....but it can NOT stream to the Reducer code until all mappers have completed.

Why use Map-Reduce?

- **Increase your computational power**
- **Processing large volumes of data**
 - Must be able to split-up the data in chunks for processing, which are then recombined later
 - Requires a constant flow of data from one simple state to another
- **The Hadoop framework handles the processing details, leaving developers free to focus on application logic**
- **Within the Hadoop Core framework, MapReduce is often referred to as mapred, and HDFS is often referred to as dfs**

Not everything is a MR

- MRs are ideal for “embarrassingly parallel” problems
- Very little communication
- Easily distributed
- Linear computational flow

-
- Next up is a map reduce implementation, Hadoop

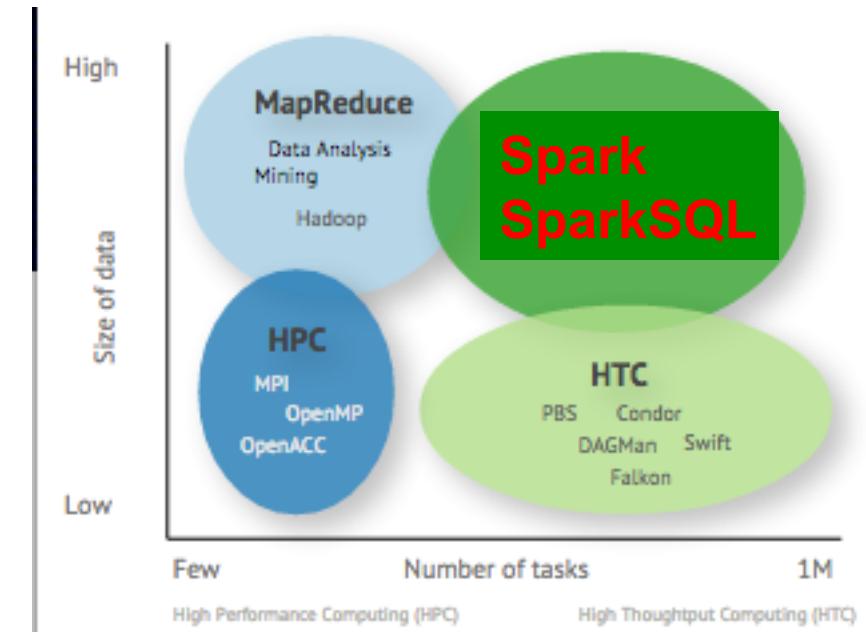
Big data and fast throughput!

In the lower left corner, the low number of tasks and small input size makes tightly-coupled Message Passing Interface (MPI) quite manageable. This is the traditional terrain of HPC.

As the data size increases (vertically), we move into the analytics category, such as data mining and analysis.

In the lower right corner, data size remains modest, but the increasing number of tasks moves us into loosely-coupled applications involving many tasks. HTC can be considered a subset of this category.

Finally, the combination of both many tasks and large datasets in the upper right corner moves us into the province of Many-Task Computing. MTC can also be considered as part of the high-task, low data (lower right) area.



Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)
 - Sync time

V4

BLT:Question

- Limitations of MPI framework

BLT:Question: Answer

- Limitations of MPI framework

• LEFT OVERS

-
- A problem in which we can find repeated patterns of identical sub problems may be used for parallel processing. This is something we look for when we are writing a multithreaded or parallelized application.

Hadoop, Why?

- **Need to process Multi Petabyte (1000 Harddrives) Datasets**
- **Expensive to build reliability in each application.**
- **Nodes fail every day**
 - Failure is expected, rather than exceptional.
 - The number of nodes in a cluster is not constant.
- **Need common infrastructure**
 - Efficient, reliable, Open Source Apache License
- **The above goals are same as Condor, but**
 - Workloads are IO bound and not CPU bound
 - Condor supports the standard [MPI](#) and [PVM](#)

Divide and Conquer Basics (Map-Reduce)

Q1

- The only feasible approach to tackling large-data problems today is to divide and conquer, a fundamental concept in computer science that is introduced very early in typical undergraduate curricula. The basic idea is to partition a large problem into smaller subproblems.
- To the extent that the sub-problems are independent, they can be tackled in parallel by different workers –
 - threads in a processor core, cores in a multi-core processor, multiple processors in a machine, or many machines in a cluster.
- Intermediate results from each individual worker are then combined to yield the final output.
- The general principles behind divide-and-conquer algorithms are broadly applicable to a wide range of problems in many different application domains.

Implementations of D&C are varied and complex

- How do we break up a large problem into smaller tasks? More specifically, how do we decompose the problem so that the smaller tasks can be executed in parallel?
- How do we assign tasks to workers distributed across a potentially large number of machines (while keeping in mind that some workers are better suited to running some tasks than others, e.g., due to available resources, locality constraints, etc.)?
- How do we ensure that the workers get the data they need?
- How do we coordinate synchronization among the different workers?
- How do we share partial results from one worker that is needed by another?
- How do we accomplish all of the above in the face of software errors and hardware faults?

• LEFT OVERS END

L2 Parallel computing, MapReduce, Hadoop

- • Motivation for Parallel Computing (5)
- • Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]

- **Hadoop (40)**

- Background and history (5)
- Hadoop File System (HDFS) (10)
- MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5

- **Full code Examples (20) [optional]**

- WordCount example on local machine (10) [ScreenFlow]
- WordCount example on cluster(10) [ScreenFlow 10]

- **MapReduce: Runtime Environment (5)**

- **Hadoop 2.0 and what is Hadoop good at? (10)**

- Sync time

V4

•Hadoop

What is MapReduce?

- **MapReduce is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster**
- **Pioneered by Google**
 - Processes 20 PB of data per day
- **Popularized by open-source Hadoop project**
 - Used by Yahoo!, Facebook, Amazon, NativeX...

Hadoop

- **Distributed file system (HDFS)**
 - Single namespace for entire cluster
 - Replicates data 3x for fault-tolerance
- **MapReduce implementation**
 - Executes user jobs specified as “map” and “reduce” functions
 - Manages work distribution & fault-tolerance



Map-Reduce

- MapReduce is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
- A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies).
- The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

Hadoop

- **A Map-Reduce framework “for running applications on large clusters built of commodity hardware”**
 - A way of processing large amounts of data across many machines
 - Map-Reduce is a way of breaking down a large task into smaller manageable tasks
 - First we Map, then we Reduce
- **Requires a shared file system**
 - HDFS (The file system of Hadoop Stands for “Hadoop Distributed File System”)
 - Hadoop also supports Amazon’s simple storage service (S3)
 - CloudStorage

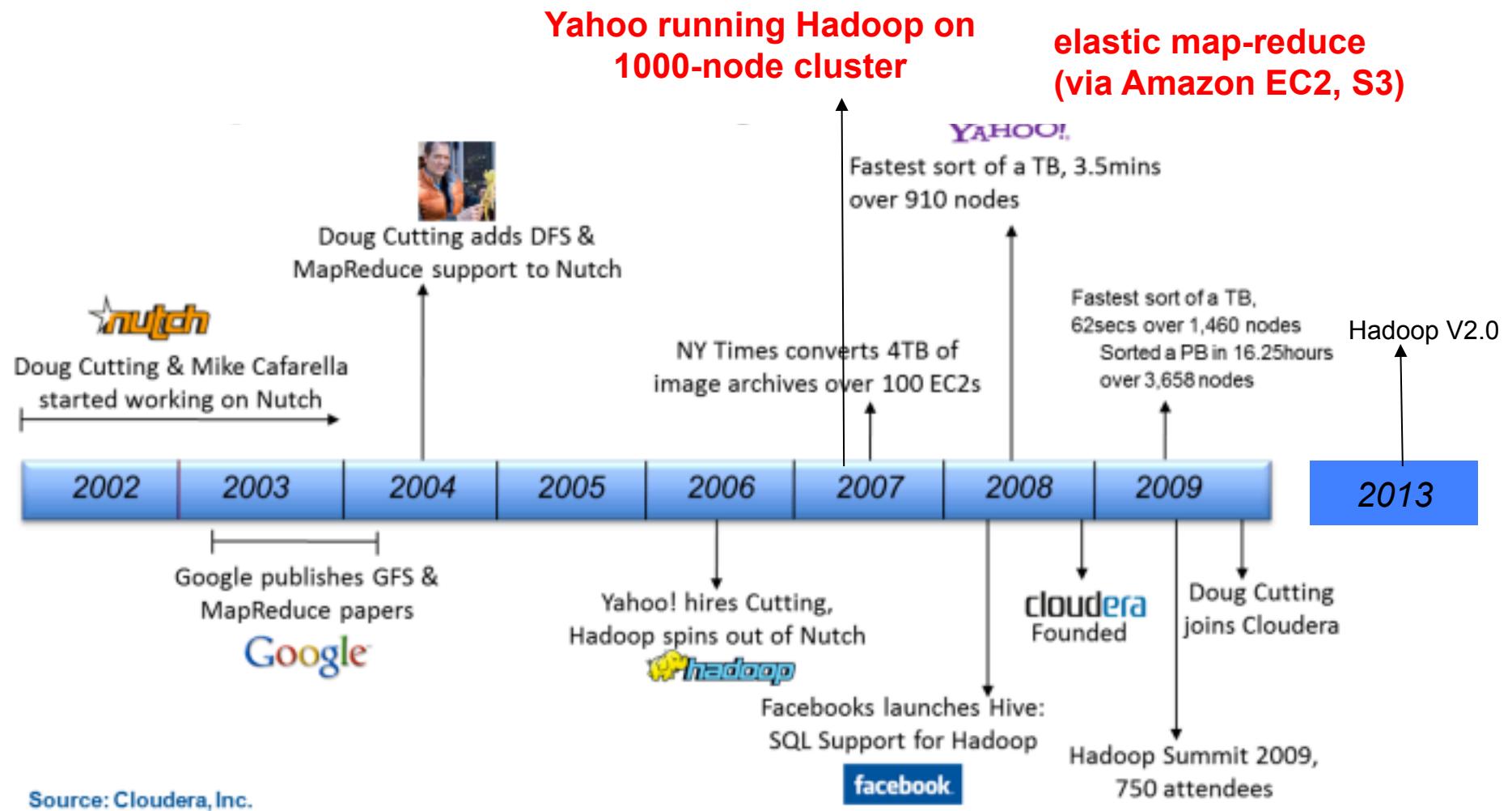
What is MapReduce used for?

- **At Google:**
 - Index building for Google Search
 - Article clustering for Google News
 - Statistical machine translation
 - **At Yahoo!:**
 - Index building for Yahoo! Search
 - Spam detection for Yahoo! Mail
 - **At Facebook:**
 - Data mining
 - Ad optimization
 - Spam detection
 - **NativeX (Mobile Ad Tech):**
 - Ad optimization
 - Building predictive models with 10s of Gig of data
 - Process 10s of TBs of data
 - **In research:**
 - Analyzing Wikipedia conflicts (PARC)
 - Natural language processing (CMU)
 - Bioinformatics (Maryland)
 - Astronomical image analysis (Washington)
 - Ocean climate simulation (Washington)
 - Graph OLAP (NUS)
 - ...
 - <Your application>
- And for machine learning*

Hadoop: More Background

- Apache Hadoop is a Java software framework that supports data-intensive distributed applications under a free license.
- Hadoop was created by Doug Cutting (now a Cloudera employee), who named it after his child's stuffed elephant.
 - It was originally developed to support distribution for the Nutch search engine project.
- It enables applications to work with thousands of nodes and petabytes of data.
- Hadoop was inspired by Google's MapReduce and Google File System (GFS) papers.
- Hadoop is a top-level Apache project, being built and used by a community of contributors from all over the world. Yahoo! has been the largest contributor to the project and uses Hadoop extensively in its web search and advertising businesses.

History of Hadoop

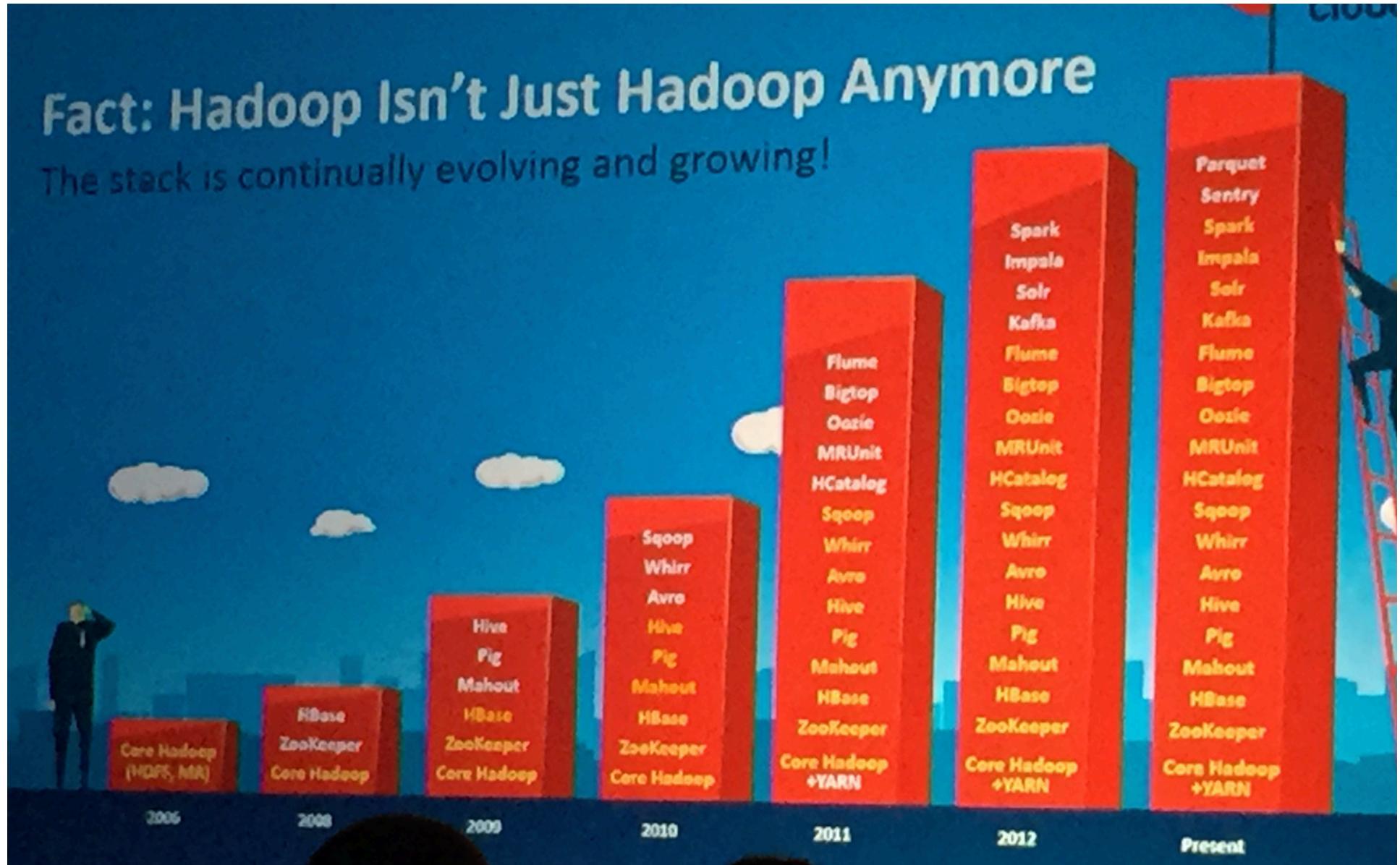


History of Hadoop

- Feb. 2003 -- First MapReduce library at Google
- Oct 2003 – Google File System paper published
- Dec 2004 – Google MapReduce paper published
- Jul 2005 – Doug Cutting reports that Nutch uses MapReduce implementation
- April 2007 – Yahoo running Hadoop on 1000-node cluster
- July 2008 -- Hadoop won Terabyte sort benchmark (209 second, 910 nodes)
- 2009 – elastic map-reduce (via Amazon EC2, S3)
- 2013 Hadoop V2.0

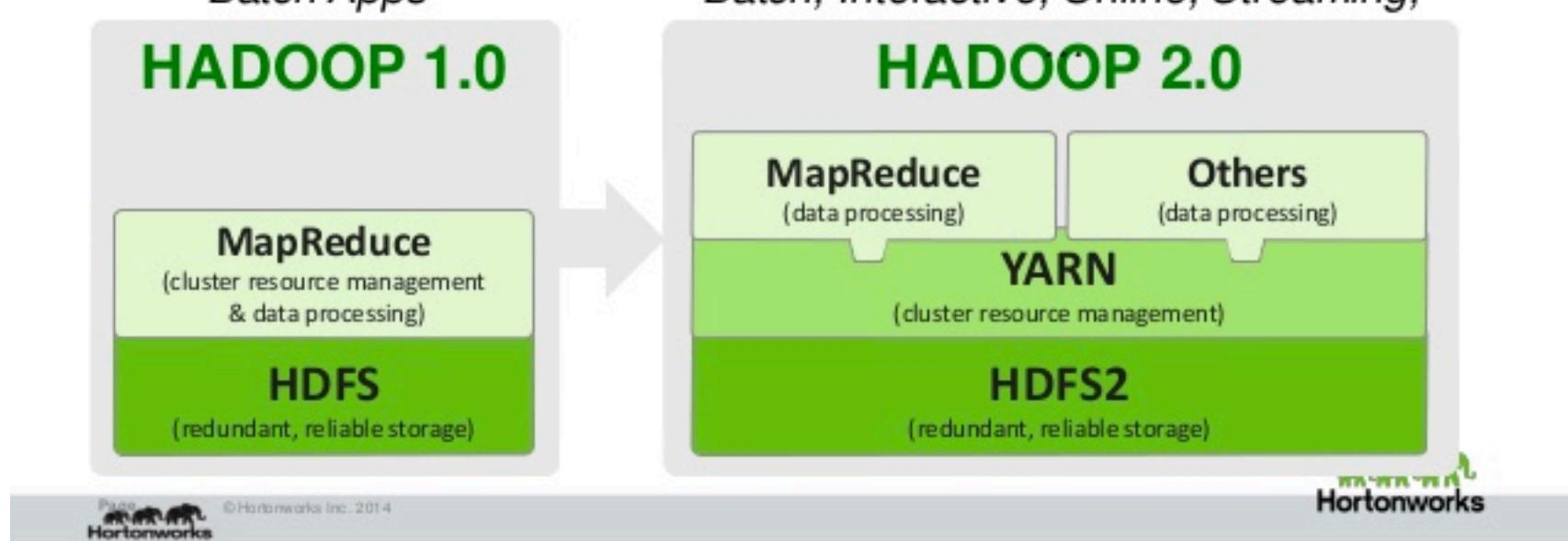
Fact: Hadoop Isn't Just Hadoop Anymore

The stack is continually evolving and growing!

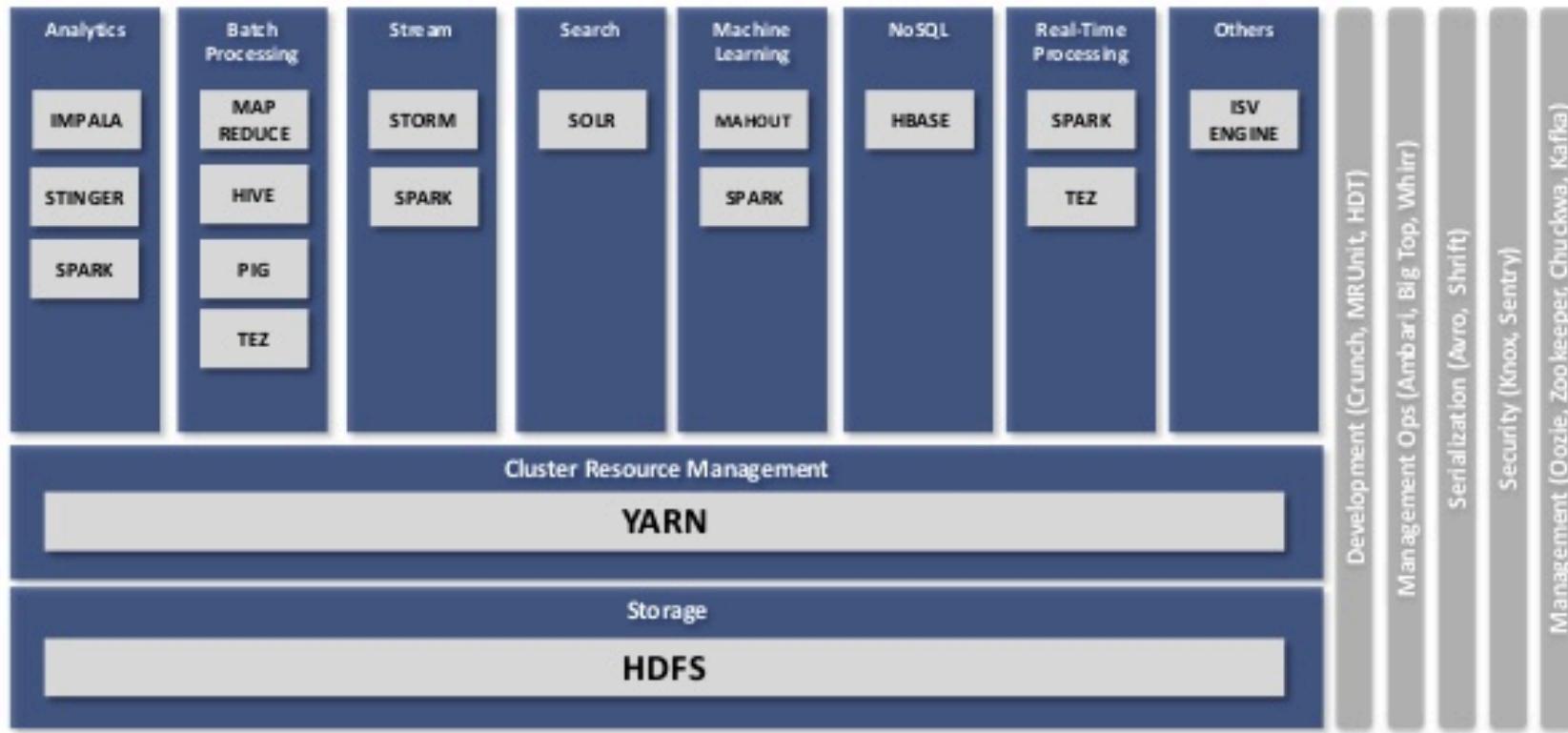


Hadoop v2.+

Multi-Workload Processing



APACHE HADOOP 2.0 ECOSYSTEM



<http://incubator.apache.org/projects/>

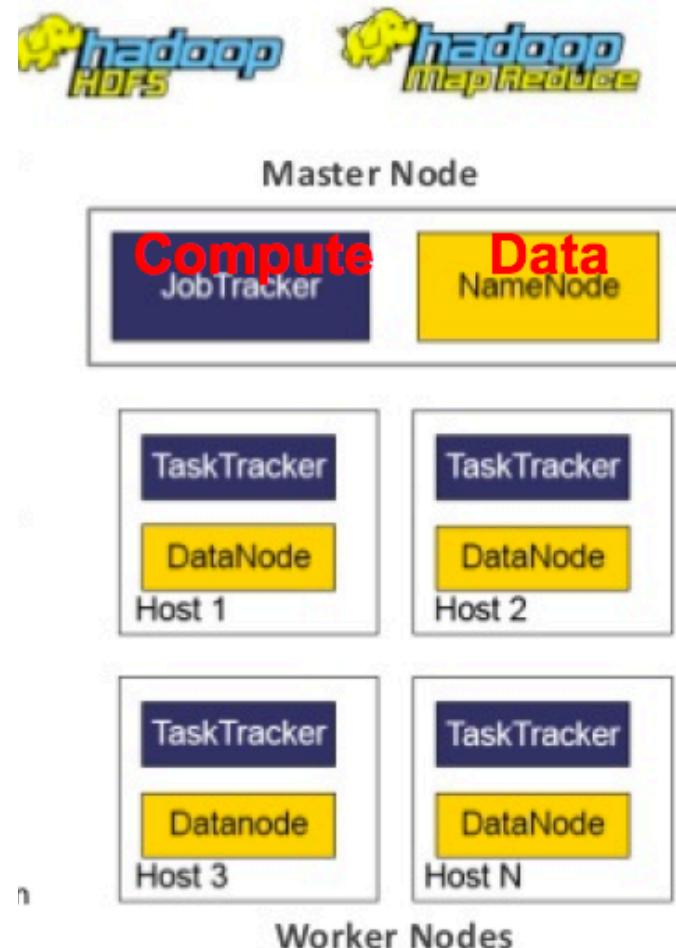
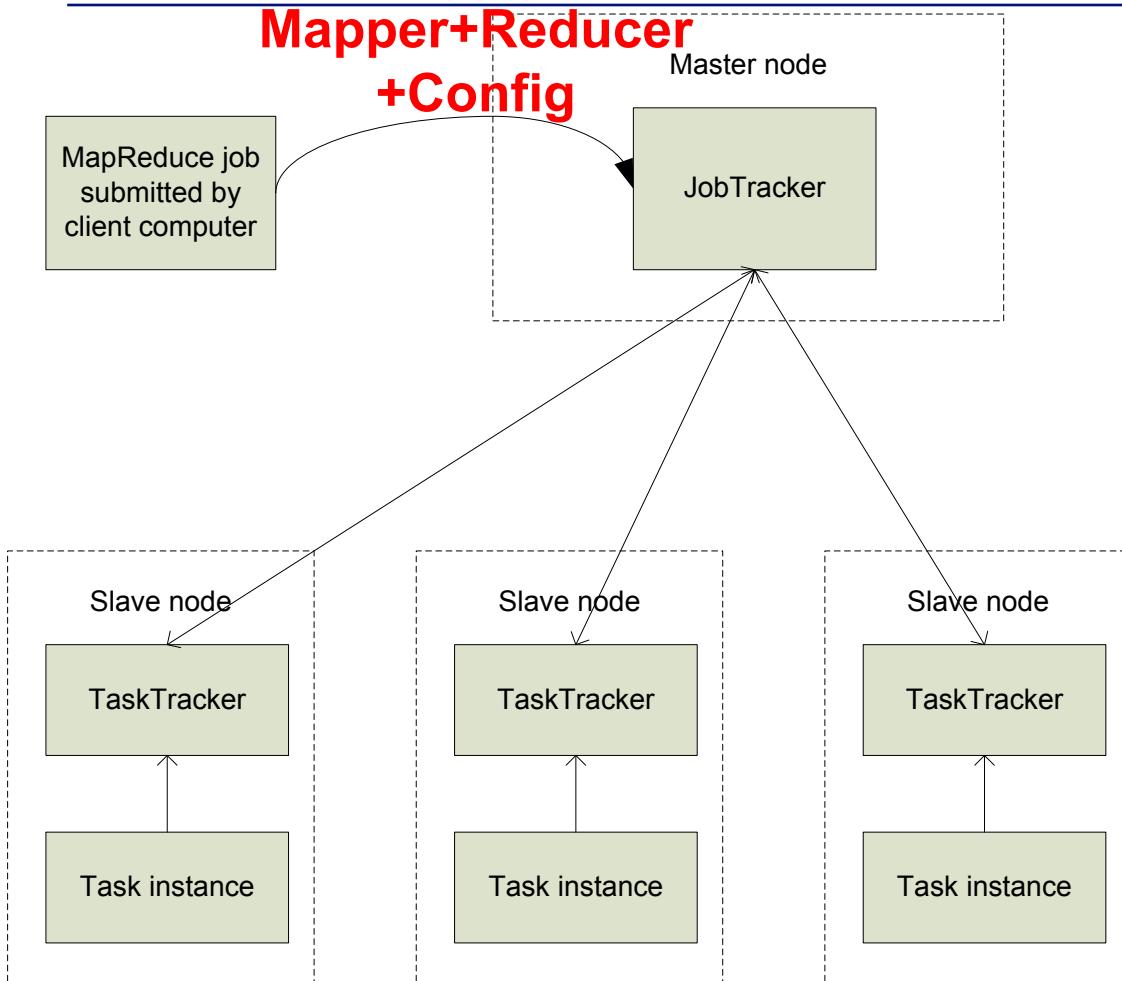
What Is **hadoop** ?

- **The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. Hadoop includes:**
 - Hadoop Common utilities
 - Avro: A data serialization system with scripting languages.
 - HDFS: A distributed file system.
 - Hive: data summarization and ad hoc querying.
 - MapReduce: distributed processing on compute clusters.

Master/Slave architecture

- Hadoop follows the Master/Slave architecture described in the original MapReduce paper.
- Since Hadoop consists of two parts: Data storage (HDFS) and processing engine (MapReduce), there are two types of master node:
- Types of node
 - For HDFS, the master node is namenode, and slave is datanode;
 - For MapReduce in Hadoop, the master node is jobtracker, and slave is tasktracker.

MapReduce: Data and Compute



One instance of your Mapper is initialized by the *MapTaskRunner* for a *input block*

Exists in separate process from all other instances of Mapper – no data sharing!

THE HADOOP SERVICES FOR EXECUTING MAPREDUCE JOBS

THE HADOOP SERVICES FOR EXECUTING MAPREDUCE JOBS

Hadoop MapReduce comes with two primary services for scheduling and running MapReduce jobs. They are the *Job Tracker* (*JT*) and the *Task Tracker* (*TT*). Broadly speaking the *JT* is the master and is in charge of allocating tasks to task trackers and scheduling these tasks globally. A *TT* is in charge of running the Map and Reduce tasks themselves.

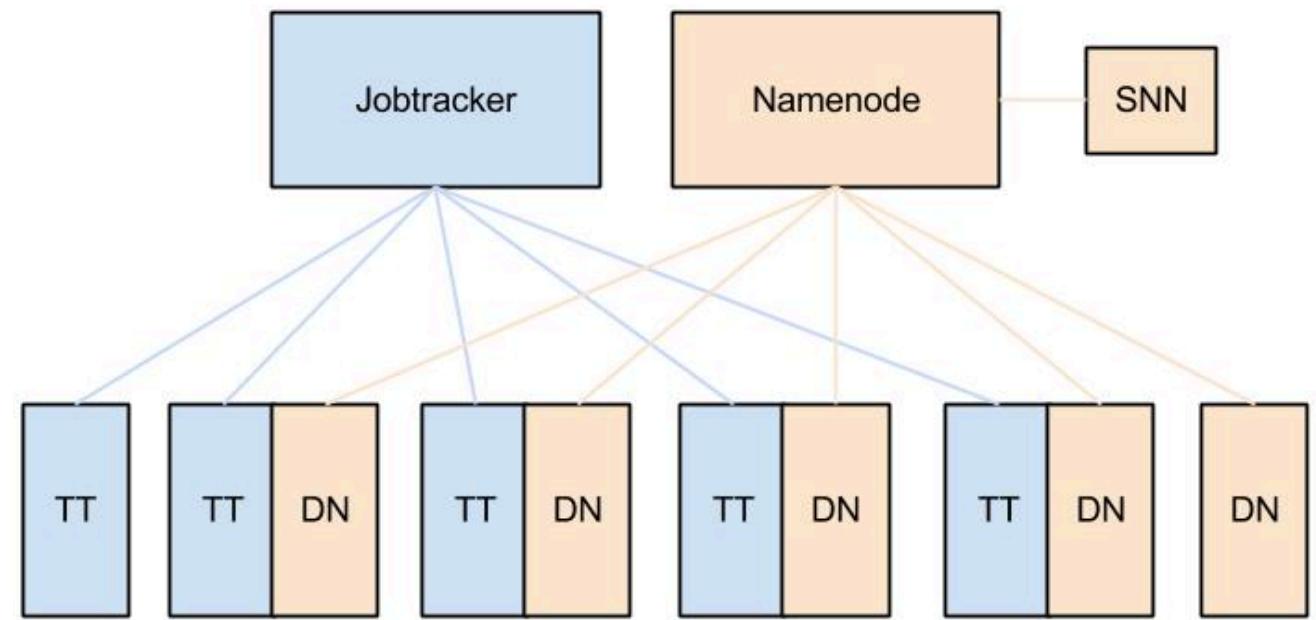
When running, each *TT* registers itself with the *JT* and reports the number of 'map' and 'reduce' slots it has available, the *JT* keeps a central registry of these across all *TTs* and allocates them to jobs as required. When a task is completed, the *TT* re-registers that slot with the *JT* and the process repeats.

Many things can go wrong in a big distributed system, so these services have some clever tricks to ensure that your job finishes successfully:

- **Automatic retries** - if a task fails, it is retried N times (usually 3) on different task trackers.
- **Data locality optimizations** - if you co-locate a *TT* with a HDFS Datanode (which you should) it will take advantage of data locality to make reading the data faster
- **Blacklisting a bad TT** - if the *JT* detects that a *TT* has too many failed tasks, it will blacklist it. No tasks will then be scheduled on this task tracker.
- **Speculative Execution** - the *JT* can schedule the same task to run on several machines at the same time, just in case some machines are slower than others. When one version finishes, the others are killed.

<http://blog.matthewrathbone.com/>

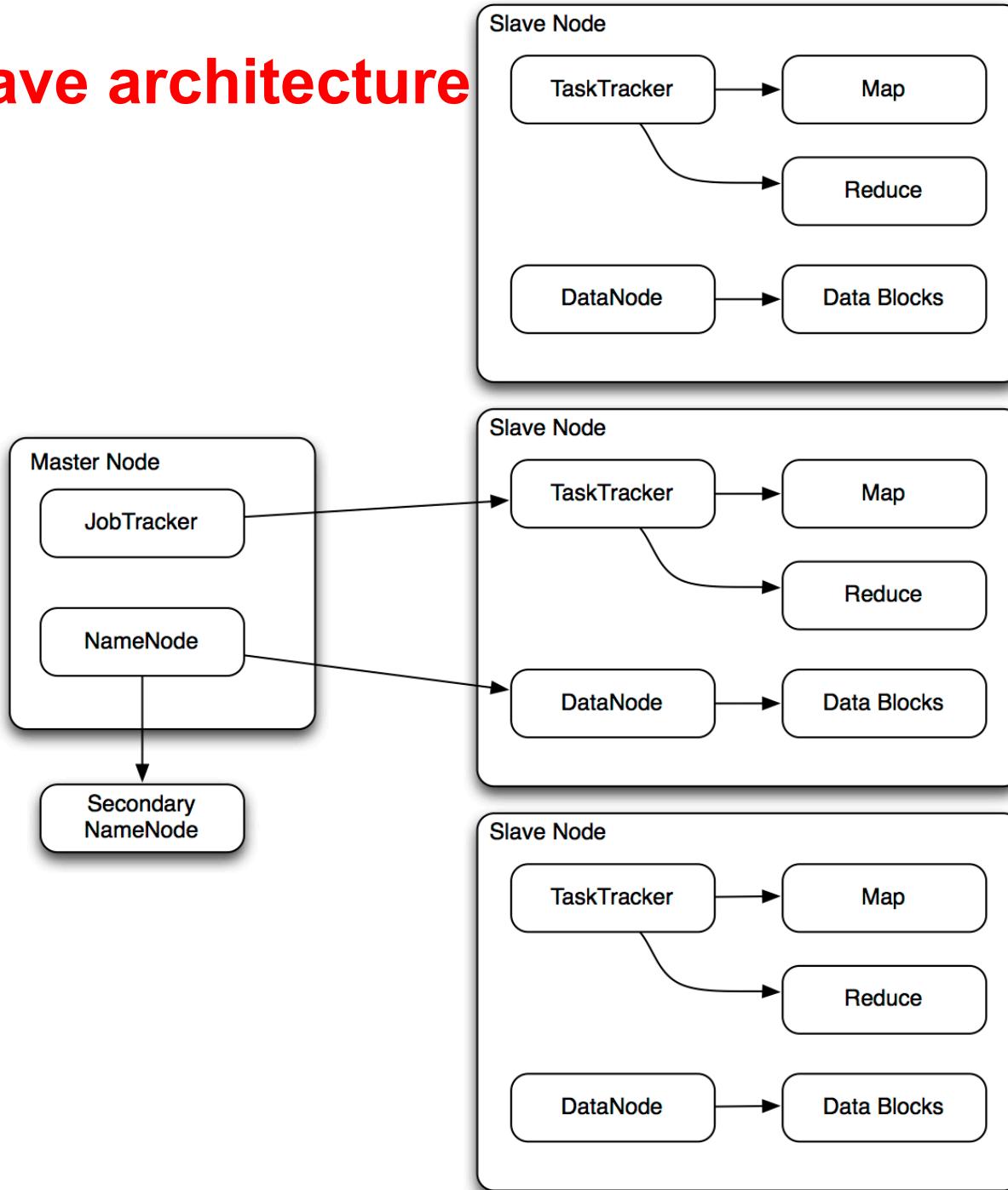
Job Tracker (JT) and the Task Tracker (TT)



Broadly speaking the JT is the master and is in charge of allocating tasks to task trackers and scheduling these tasks globally. A TT is in charge of running the

Master/Slave architecture

• ..



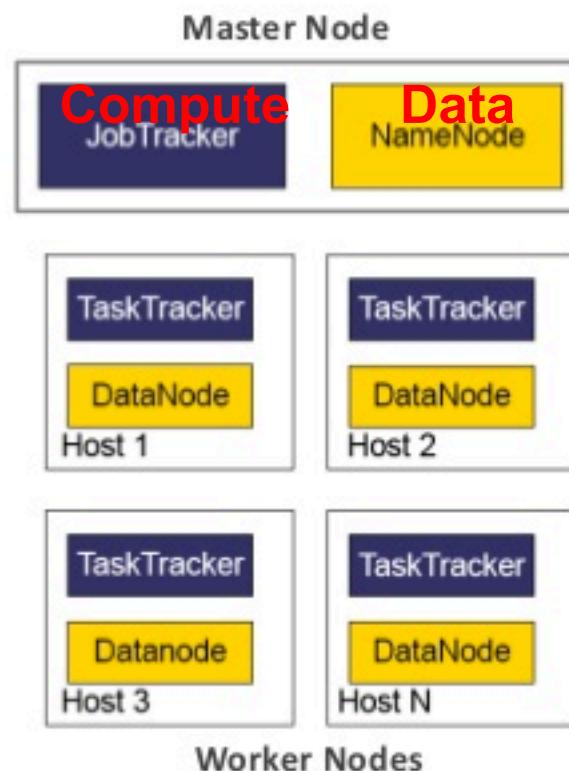
HDFS & MAPREDUCE

- **Hadoop Distributed File System**

- A scalable, Fault tolerant, High performance distributed file system
- Asynchronous replication
- Write-once and read-many (WORM)
- Hadoop cluster with 3 DataNodes minimum
- Data divided into 64MB (default) or 128MB blocks, each block replicated 3 times (default)
- No RAID required for DataNode
- Interfaces: Java, Thrift, C Library, FUSE, WebDAV, HTTP, FTP
- **NameNode** holds filesystem metadata
- Files are broken up and spread over the **DataNodes**

- **Hadoop Map Reduce**

- Software framework for distributed computation
- Input | Map() | Copy/Sort | Reduce() | Output
- **JobTracker** schedules and manages jobs
- **TaskTracker** executes individual map() and reduce() tasks on each cluster node



MapReduce Compute

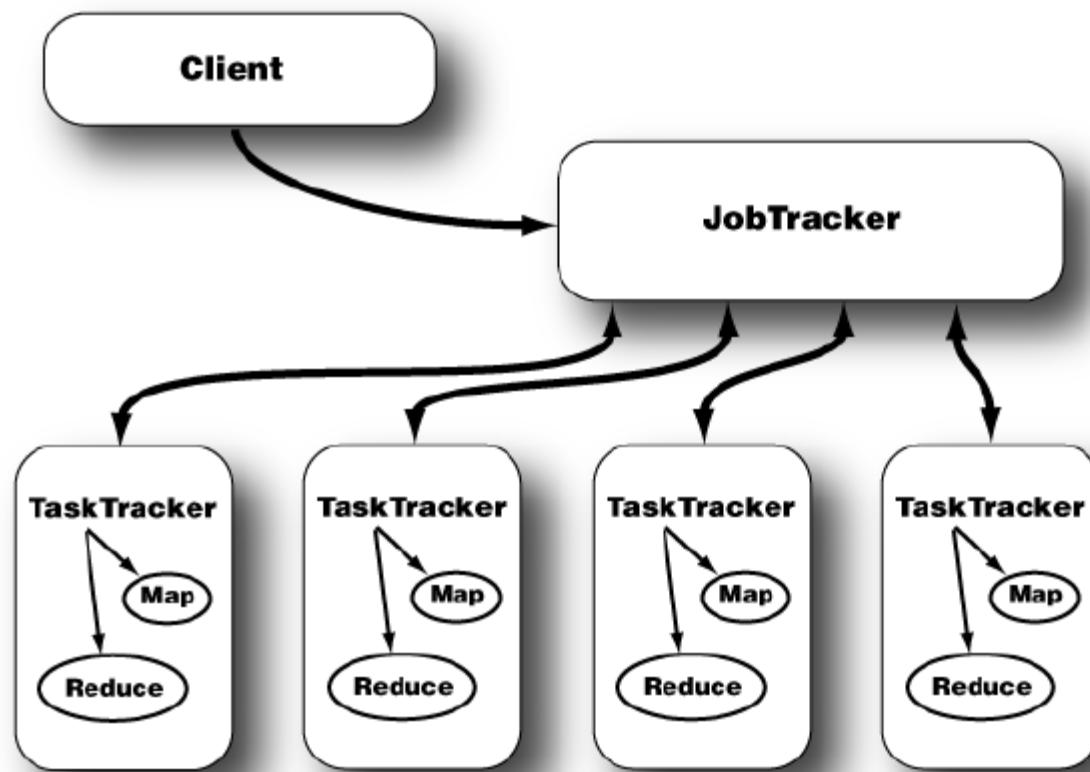


Figure 2.2. JobTracker and TaskTracker interaction. After a client calls the JobTracker to begin a data processing job, the JobTracker partitions the work and assigns different map and reduce tasks to each TaskTracker in the cluster.

-
- **That was an overview of hadoop**
 - **Next up we look at the internals of HDFS**
 - **MR programming framework**

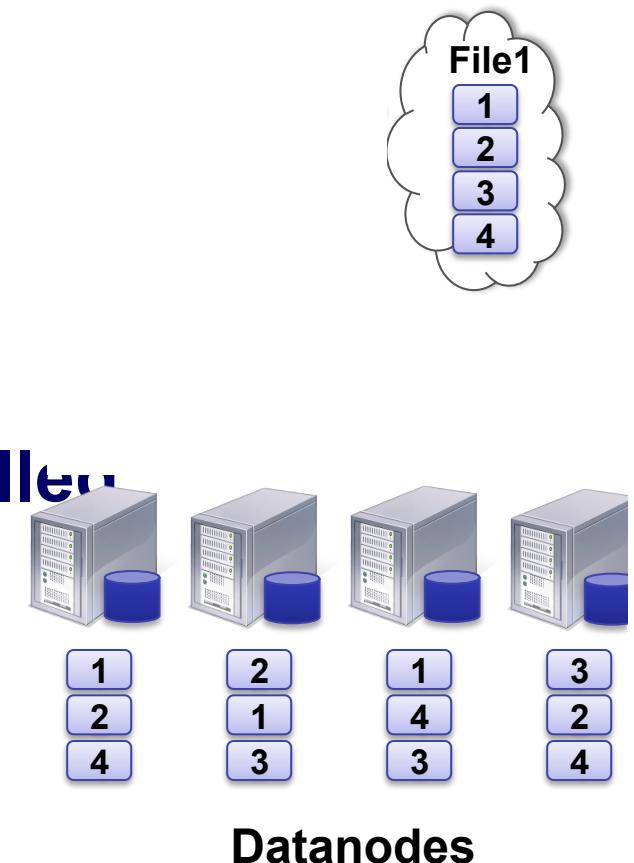
Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)
- Sync time
 - Install Hadoop; run locally; run in the cloud?

V4

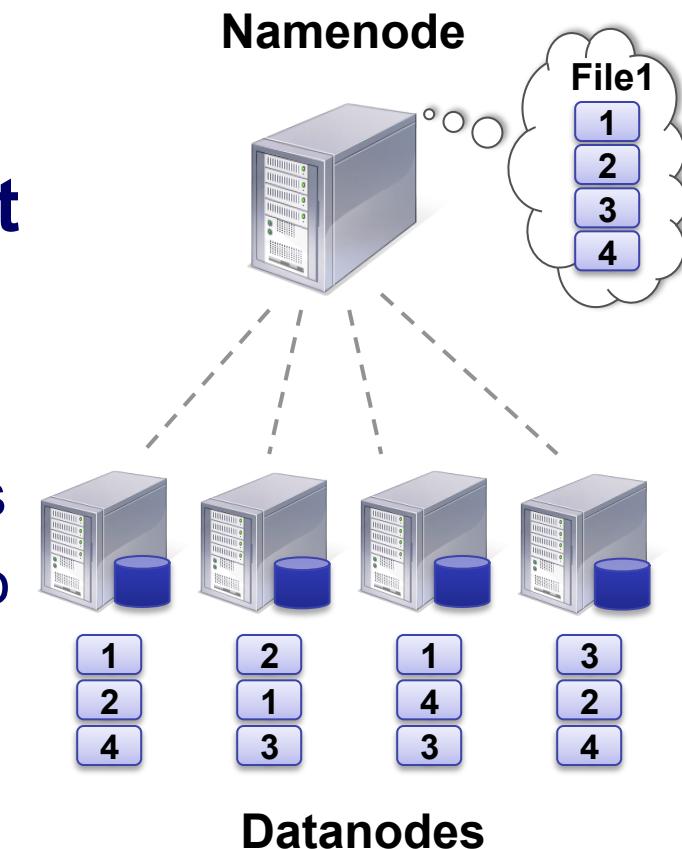
Hadoop Distributed File System

- **Files are BIG**
 - 100s of GBs/TB even PBs)
- **Typical usage patterns**
 - Append-only
 - Data are rarely updated in place
 - Reads common
- **Optimized for large files, sequential (why??) reads**
- **Files split into 64-128MB blocks (called chunks); today 1G-2G**
 - Blocks replicated (usually 3 times) across several *datanodes* (called chunk or slave nodes)

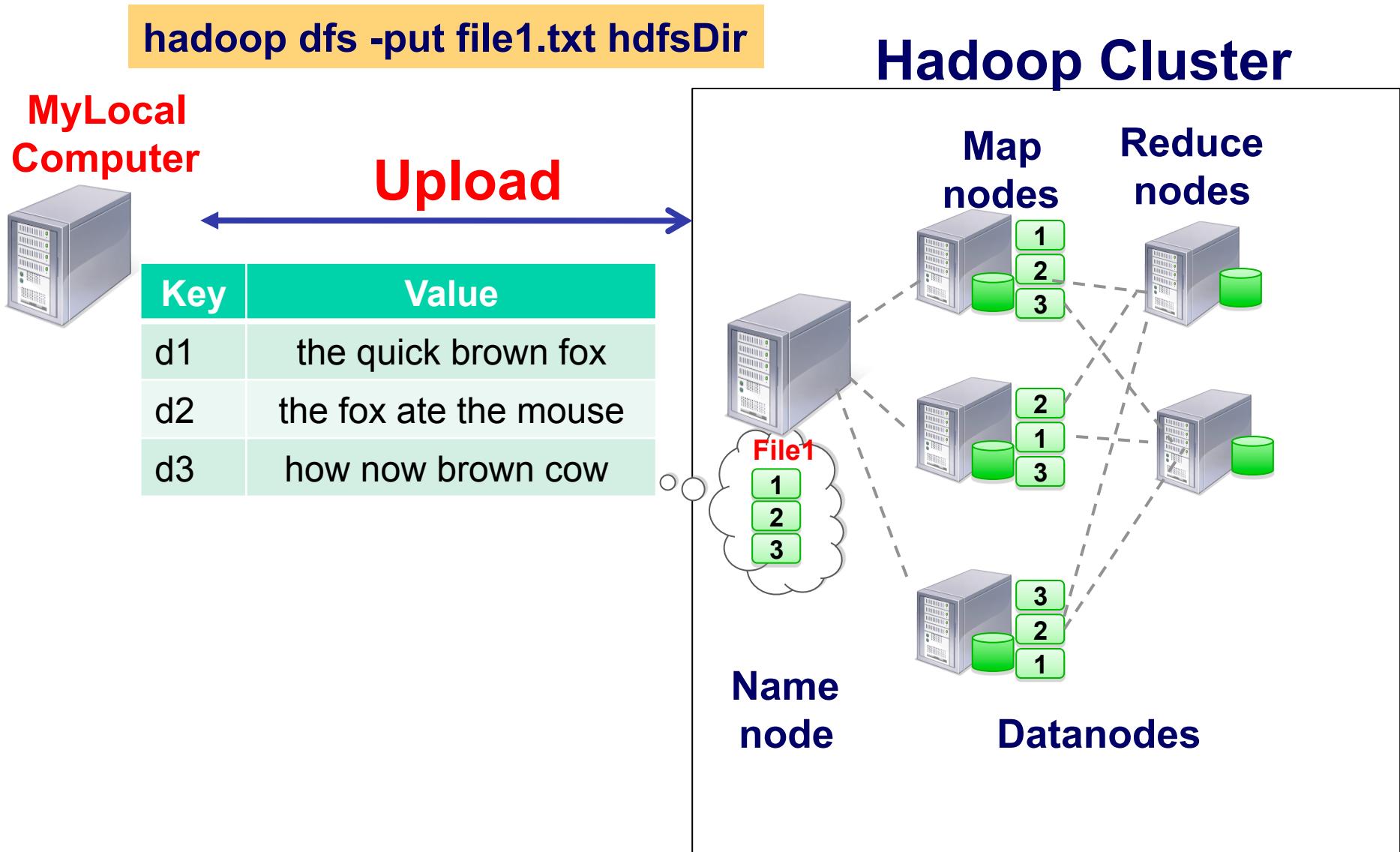


Hadoop Distributed File System

- Single *namenode* (master node) stores metadata (file names, block locations, etc)
 - May be replicated also
- Datanodes provides redundant store for the data blocks
- Client library for file access
 - Talks to master to find chunk servers
 - Connects directly to chunk servers to access data
 - Master node is not a bottleneck
 - Computation is done at chuck node (close to data)

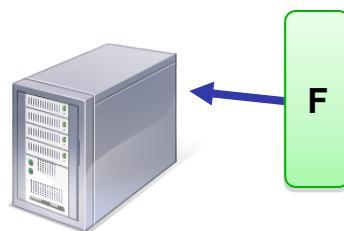


Hadoop Cluster: 1 Name; 4 Data Nodes



Hadoop Cluster: 1 Name; 4 Data Nodes

MyLocalComputer

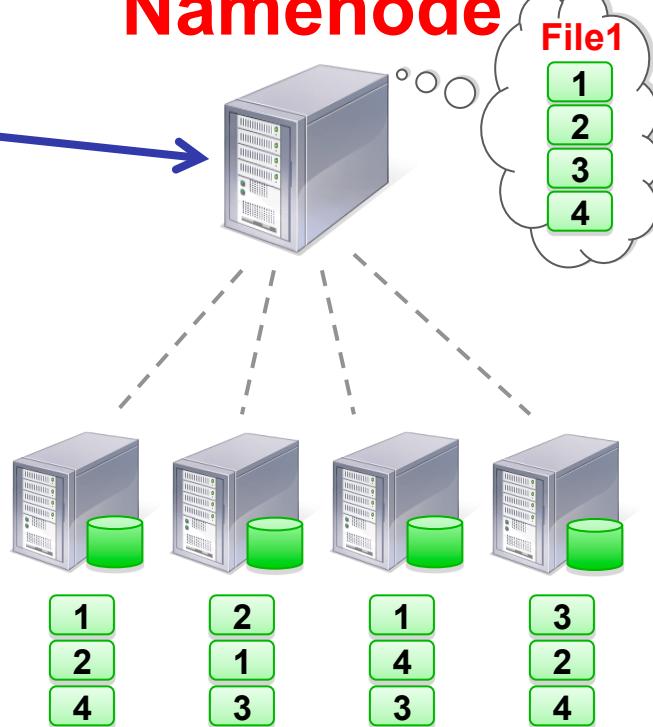


Upload
Download

hadoop dfs -put file1.txt hdfsDir

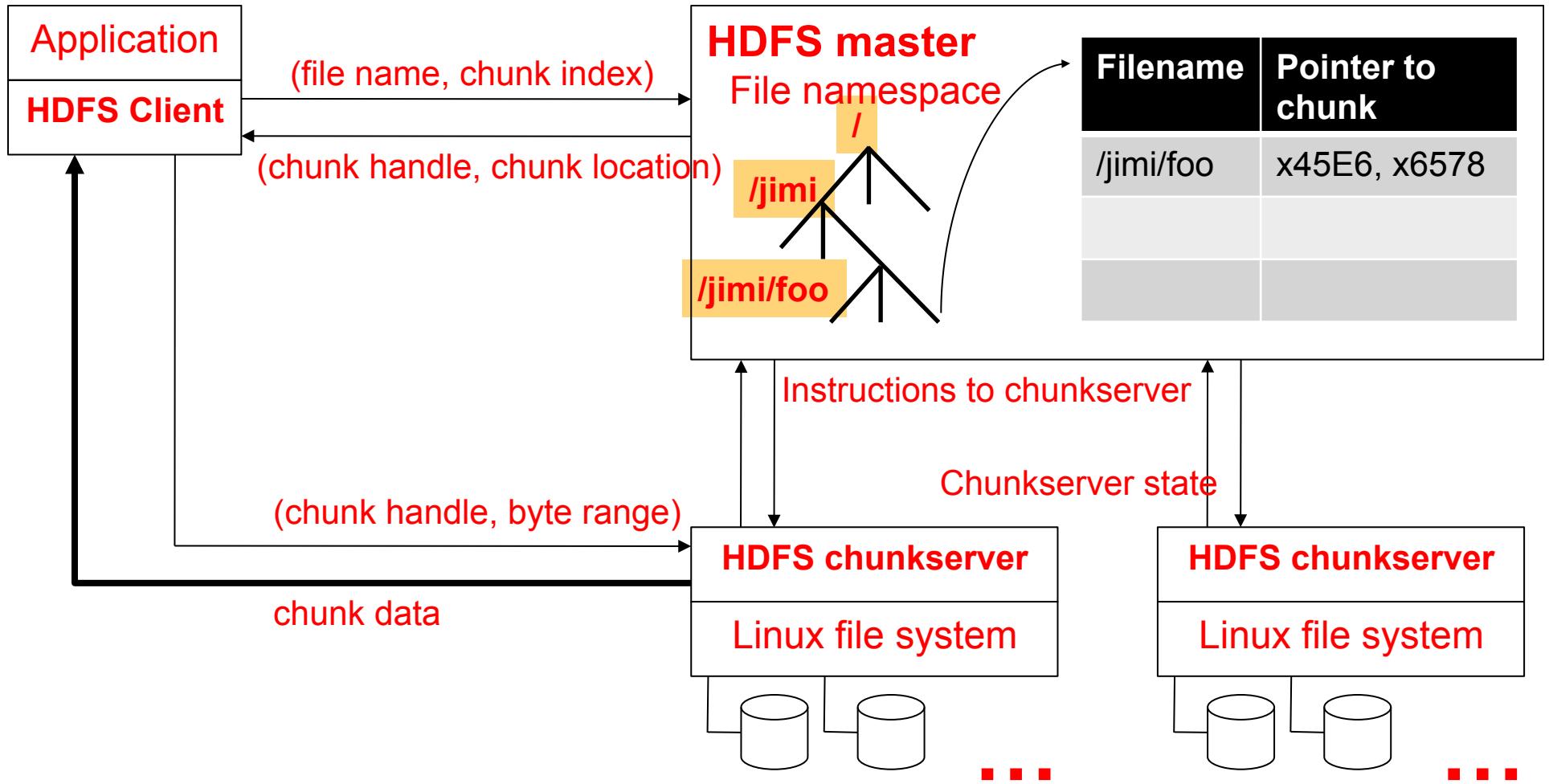
Hadoop Cluster

Namenode



Datanodes

My Computer and Hadoop (HDFS)



Loosely based on (Ghemawat et al., SOSP 2003)

Two Files on my local computer

1. Foo (150Meg)
2. Bar (100Meg)

HDFS Topology (hierarchical)

Upload to HDFS via Hadoop gateway
hadoop dfs -put file1.txt hdfsDir

NameNode:
Stores metadata only

METADATA:
`/user/aaron/foo → 1, 2, 4`
`/user/aaron/bar → 3, 5`

Block No

Foo is split into 3 blocks

64MB
block

DataNodes: Store blocks from files

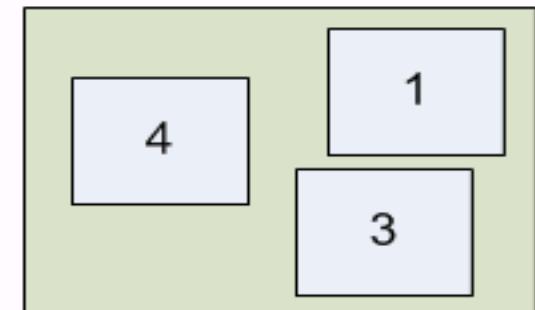
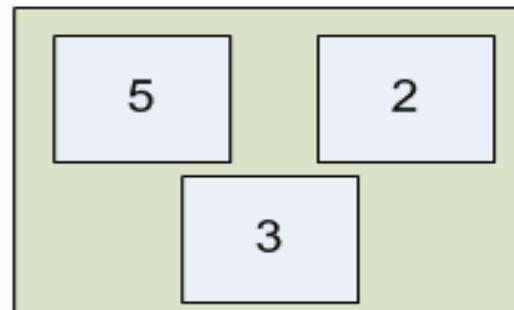
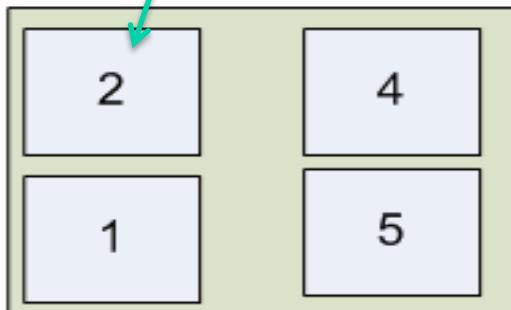


Figure 2.1. NameNode/DataNode interaction in HDFS. The NameNode keeps track of the file metadata—which files are in the system and how each file is broken down into "blocks." The DataNodes provide redundant store of the blocks and constantly report to the NameNode to keep the metadata current.

HDFS-Replication

- Default is 3x replication
- Block placement algorithms is rack-aware
- Dynamic control of replication factor
- Balancer application to re-balance cluster in background



Interacting with HDFS

- HDFS supports familiar syntax

hadoop fs -cat /root/example/file1.txt

hadoop fs -chmod -R 755 /root/example/file1.txt

hadoop fs -chown phil /root/example/file1.txt

**hadoop fs -cp /root/example/file1.txt /root/example/
file1.new**

hadoop fs -ls /root/example/

hadoop fs -mkdir /root/example/new_directory

HDFS Commands

- You can run "`hadoop dfs -help`" to understand most HDFS file operations.
- File list: `hadoop dfs -ls hdfs_path`
- Create a HDFS directory: `hadoop dfs -mkdir your_hdfs_directory`
- Delete a HDFS directory: `hadoop dfs -rm r your_hdfs_directory`
- Copy a file from local machine to HDFS: `hadoop dfs -copyFromLocal your_local_file your_hdfs_folder/file_name`
- Copy a file from HDFS to local machine: `hadoop dfs -copyToLocal your_hdfs_folder/filename your_local_file`
- Delete a file from HDFS : `hadoop dfs -rm your_hdfs_file_path`
- View a text file in HDFS: `hadoop dfs -cat your_hdfs_file|more`
- View a gzipped text file in HDFS: `hadoop dfs -cat your_gzipped_hdfs_text_file|zcat|more`

Hadoop Example

hadoop dfs -help ls

hadoop dfs -ls

hadoop dfs -mkdir example

#put two files in the /user/root/example

hadoop dfs -put file1.txt example

hadoop dfs -put file2.txt example

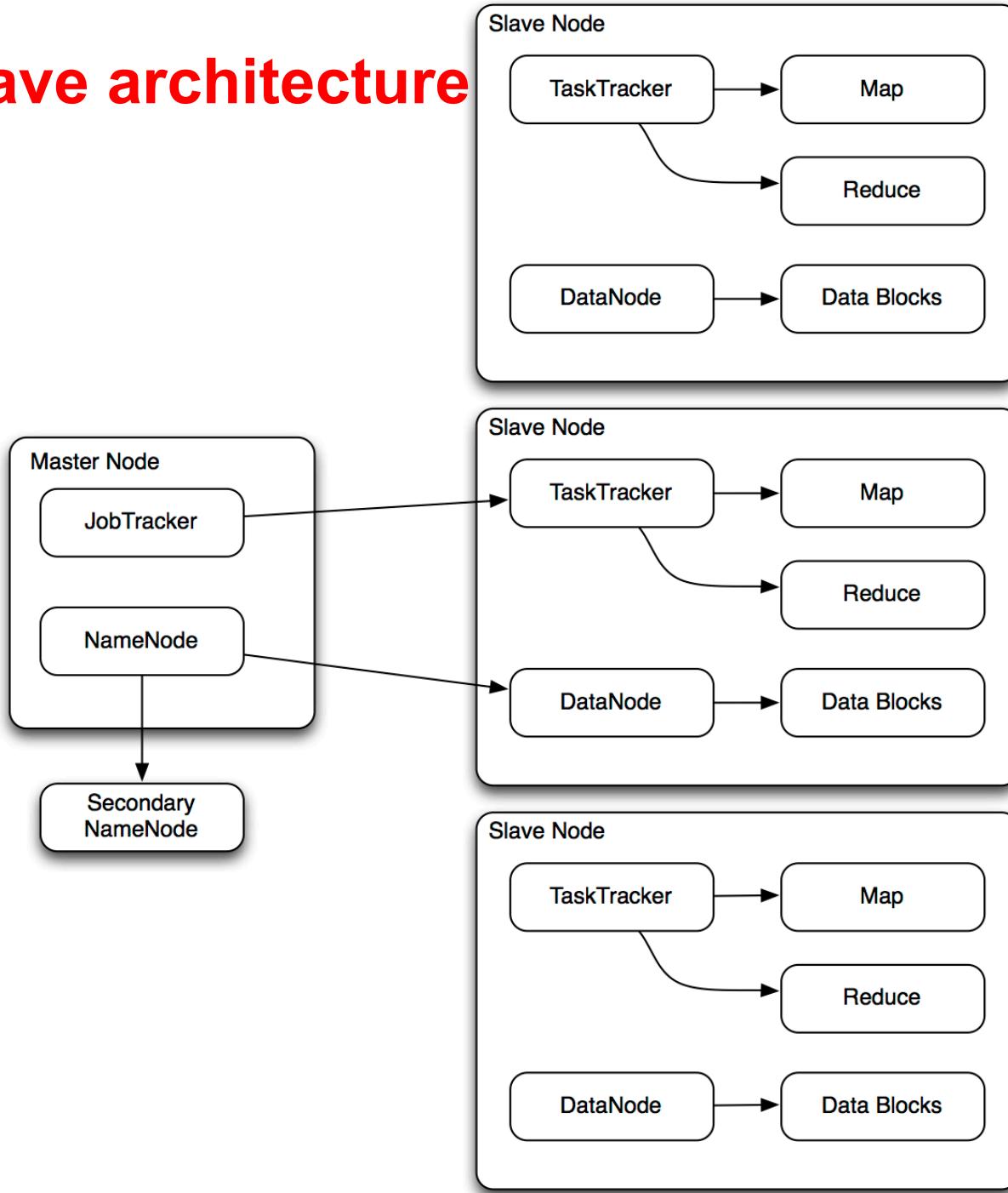
Distributed File-Systems

- Share the file-system transparently across many machines; simply copy data to master node
 - Distributed File-Systems Share the file-system transparently across many machines
 - You simply see the usual file structure
- ***Hadoop dfs -ls /user/root/example/***

```
drwxr-xr-x 34 phil staff 1156 12 Sep 2008 file1.txt  
drwxr-xr-x 34 phil staff 1156 12 Sep 2008 file2.txt  
drwxr-xr-x 34 phil staff 1156 12 Sep 2008 file3.txt  
drwxr-xr-x 34 phil staff 1156 12 Sep 2008 file4.txt
```

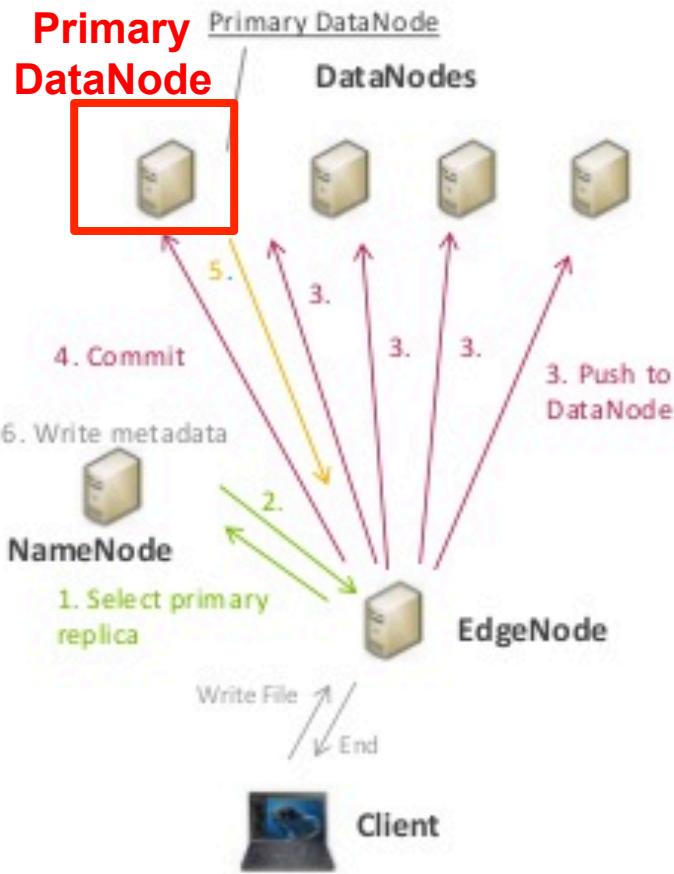
Master/Slave architecture

• ..



HDFS - WRITE FILE

1. Client contacts the NameNode who designates one of the replica as the primary
2. The response of the NameNode contains who is the primary and who are the secondary replicas
3. The client pushes its changes to all DataNodes in any order, but this change is stored in a buffer of each DataNode
4. The client sends a “commit” request to the primary, which determines an order to update and then push this order to all other secondaries
5. After all secondaries complete the commit, the primary response to the client about the success
6. All changes of blocks distribution and metadata changes be written to an operation log file at the NameNode



HDFS Summary

- **That is Hadoop file system**
 - Master-Slave architecture
 - Data is organized
 - Get data in
 - Get data out
- **HDFS Goals**
 - Store large data sets
 - Deal with hardware failures
 - Emphasize streaming data access

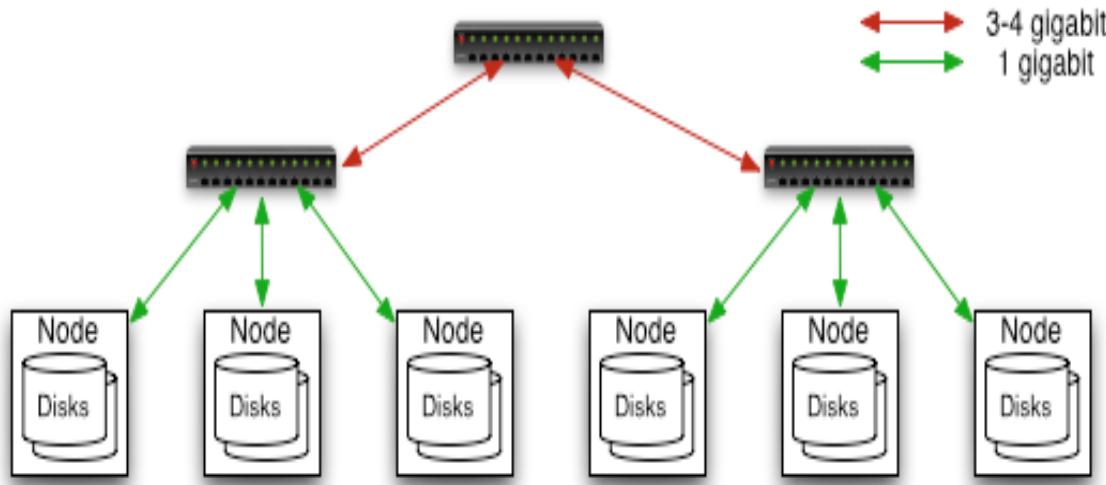
L2 Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)
 - Sync time

V4

-
- Advanced and detailed HDFS
 - Sync time

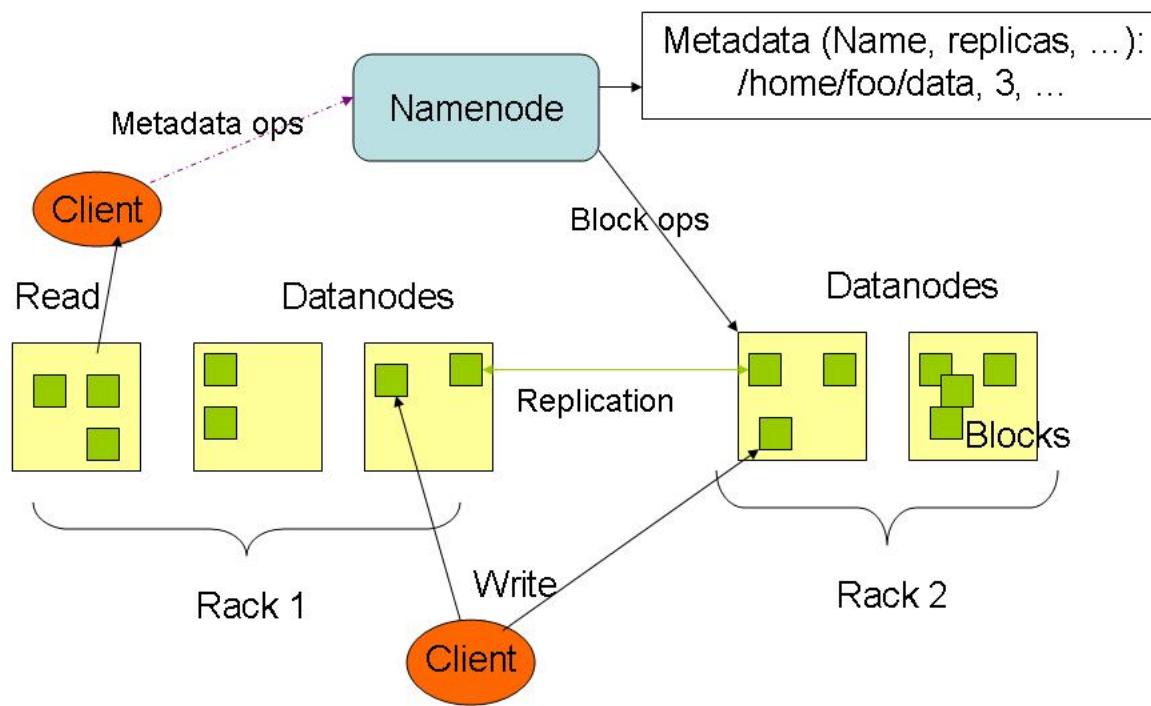
Commodity Hardware in same data center if possible. Why?



Typically in 2 level architecture

- Nodes are commodity PCs
- 30-40 nodes/rack
- Uplink from rack is 3-4 gigabit
- Rack-internal is 1 gigabit

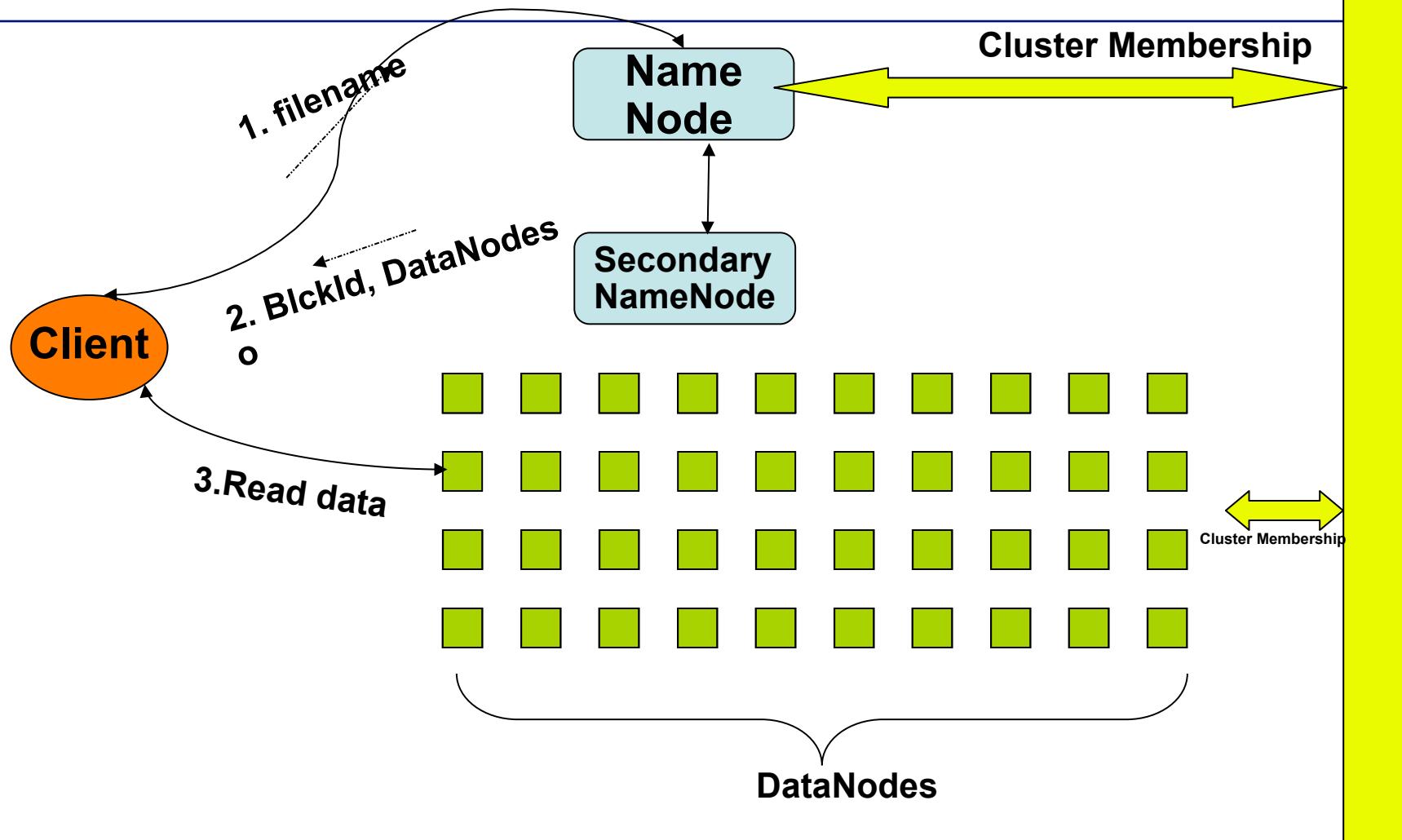
HDFS Architecture



Goals of HDFS

- **Very Large Distributed File System**
 - 10K nodes, 100 million files, 10 PB
- **Assumes Commodity Hardware**
 - Files are replicated to handle hardware failure
 - Detect failures and recovers from them
- **10 machines with 8 cores -> 80 tasks**
- **Optimized for Batch Processing**
 - Data locations exposed so that computations can move to where data resides
 - Provides very high aggregate bandwidth
- **User Space, runs on heterogeneous OS**

HDFS Architecture



NameNode : Maps a file to a file-id and list of MapNodes

DataNode : Maps a block-id to a physical location on disk

SecondaryNameNode: Periodic merge of Transaction log

Distributed File System

- Single Namespace for entire cluster
- Data Coherency
 - Write-once-read-many access model
 - Client can only append to existing files
- Files are broken up into blocks
 - Typically 128 MB (user specified) block size
 - Each block replicated on multiple DataNodes
- Intelligent Client
 - Client can find location of blocks
 - Client accesses data directly from DataNode

NameNode Metadata

- **Meta-data in Memory**
 - The entire metadata is in main memory
 - No demand paging of meta-data
- **Types of Metadata**
 - List of files
 - List of Blocks for each file
 - List of DataNodes for each block
 - File attributes, e.g creation time, replication factor
- **A Transaction Log**
 - Records file creations, file deletions. etc

DataNode

- **A Block Server**
 - Stores data in the local file system (e.g. ext3)
 - Stores meta-data of a block (e.g. CRC Checksums)
 - Serves data and meta-data to Clients
- **Block Report**
 - Periodically sends a report of all existing blocks to the NameNode
- **Facilitates Pipelining of Data**
 - Forwards data to other specified DataNodes

Block Placement

- **Current Strategy**
 - One replica on local node
 - Second replica on a remote rack
 - Third replica on same remote rack
 - Additional replicas are randomly placed
- Clients read from nearest replica
- Would like to make this policy pluggable

Data Correctness

- Use Checksums to validate data
 - Use CRC32
- File Creation
 - Client computes checksum per 512 byte
 - DataNode stores the checksum
- File access
 - Client retrieves the data and checksum from DataNode
 - If Validation fails, Client tries other replicas

NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories
 - A directory on the local file system
 - A directory on a remote file system (NFS/CIFS)
- Need to develop a better solution

Data Pipelining

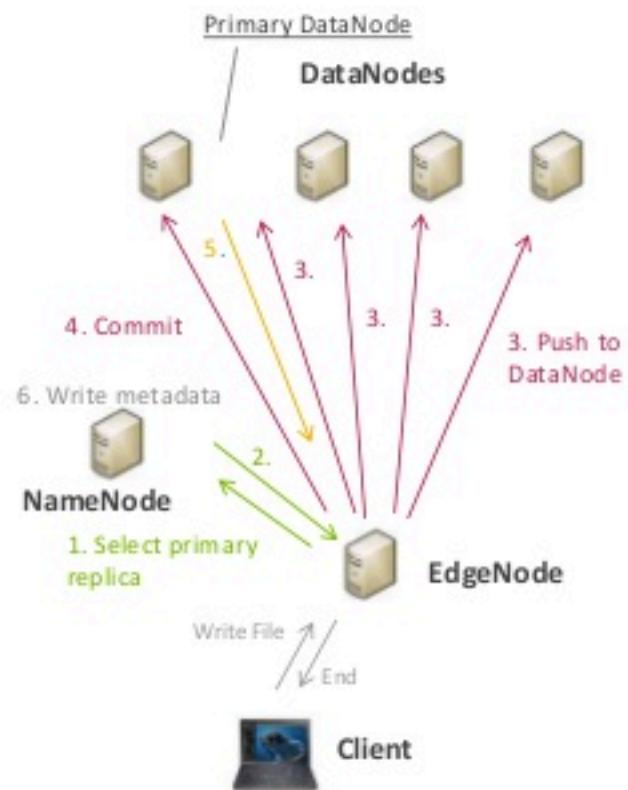
- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next DataNode in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file

Rebalancer

- **Goal: % disk full on DataNodes should be similar**
 - Usually run when new DataNodes are added
 - Cluster is online when Rebalancer is active
 - Rebalancer is throttled to avoid network congestion
 - Command line tool

HDFS - WRITE FILE

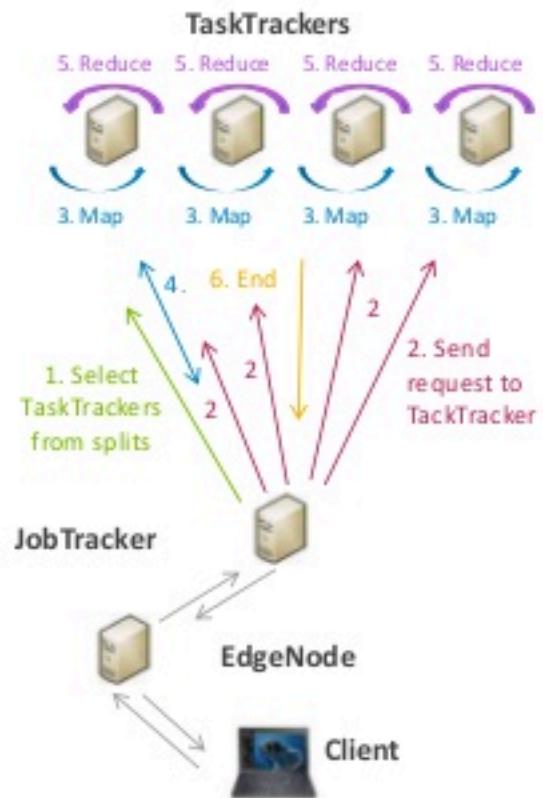
1. Client contacts the NameNode who designates one of the replica as the primary
2. The response of the NameNode contains who is the primary and who are the secondary replicas
3. The client pushes its changes to all DataNodes in any order, but this change is stored in a buffer of each DataNode
4. The client sends a “commit” request to the primary, which determines an order to update and then push this order to all other secondaries
5. After all secondaries complete the commit, the primary response to the client about the success
6. All changes of blocks distribution and metadata changes be written to an operation log file at the NameNode



MAPREDUCE - EXEC FILE

The client program is copied on each node

1. The JobTracker determines the number of splits from the input path, and select some TaskTrackers based on their network proximity to the data sources
2. JobTracker sends the task requests to those selected TaskTrackers
3. Each TaskTracker starts the map phase processing by extracting the input data from the splits
4. When the map task completes, the TaskTracker notifies the JobTracker. When all the TaskTrackers are done, the JobTracker notifies the selected TaskTrackers for the reduce phase
5. Each TaskTracker reads the region files remotely and invokes the reduce function, which collects the key/aggregated value into the output file (one per reducer node)
6. After both phase completes, the JobTracker unblocks the client program



SERIALIZATION



- A data serialization system that provides dynamic integration with scripting languages
- Avro Data
 - Expressive
 - Smaller and Faster
 - Dynamic
 - Schema store with data
 - APIs permit reading and creating
 - Include a file format and a textual encoding
- Avro RPC
 - Leverage versioning support
 - For Hadoop service provide cross-language access

DATA COMPRESSION



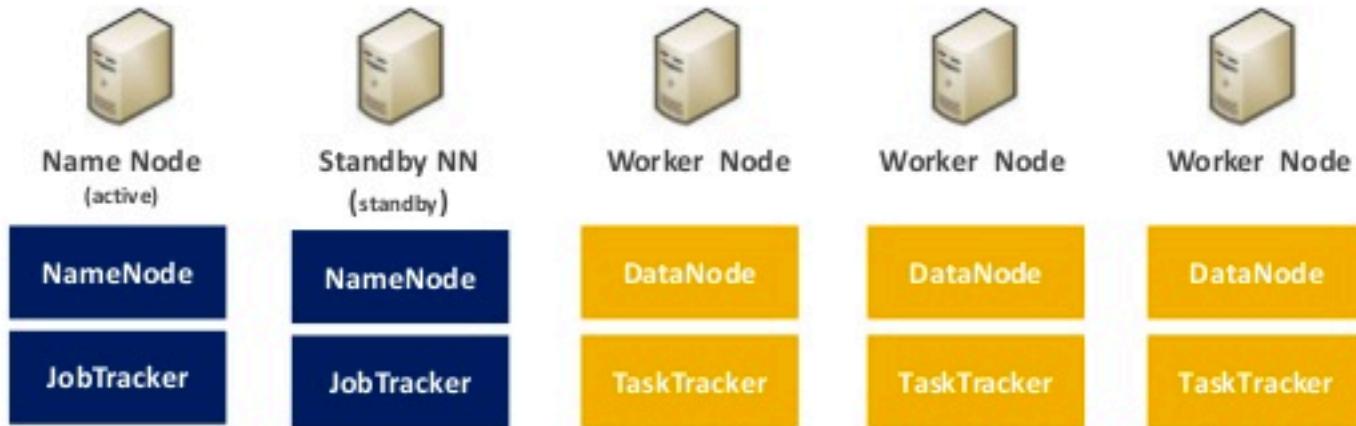
LZO

- <https://github.com/toddipcon/hadoop-lzo>
- By enabling compression, the store file uses a compression algorithm on blocks as they are written (during flushes and compactions) and thus must be decompressed when reading
- Compression reduces the number of bytes written/read to/from HDFS
- Compression effectively improves the efficiency of network bandwidth and disk space
- Compression reduces the size of data needed to be read when issuing a read

SNAPPY

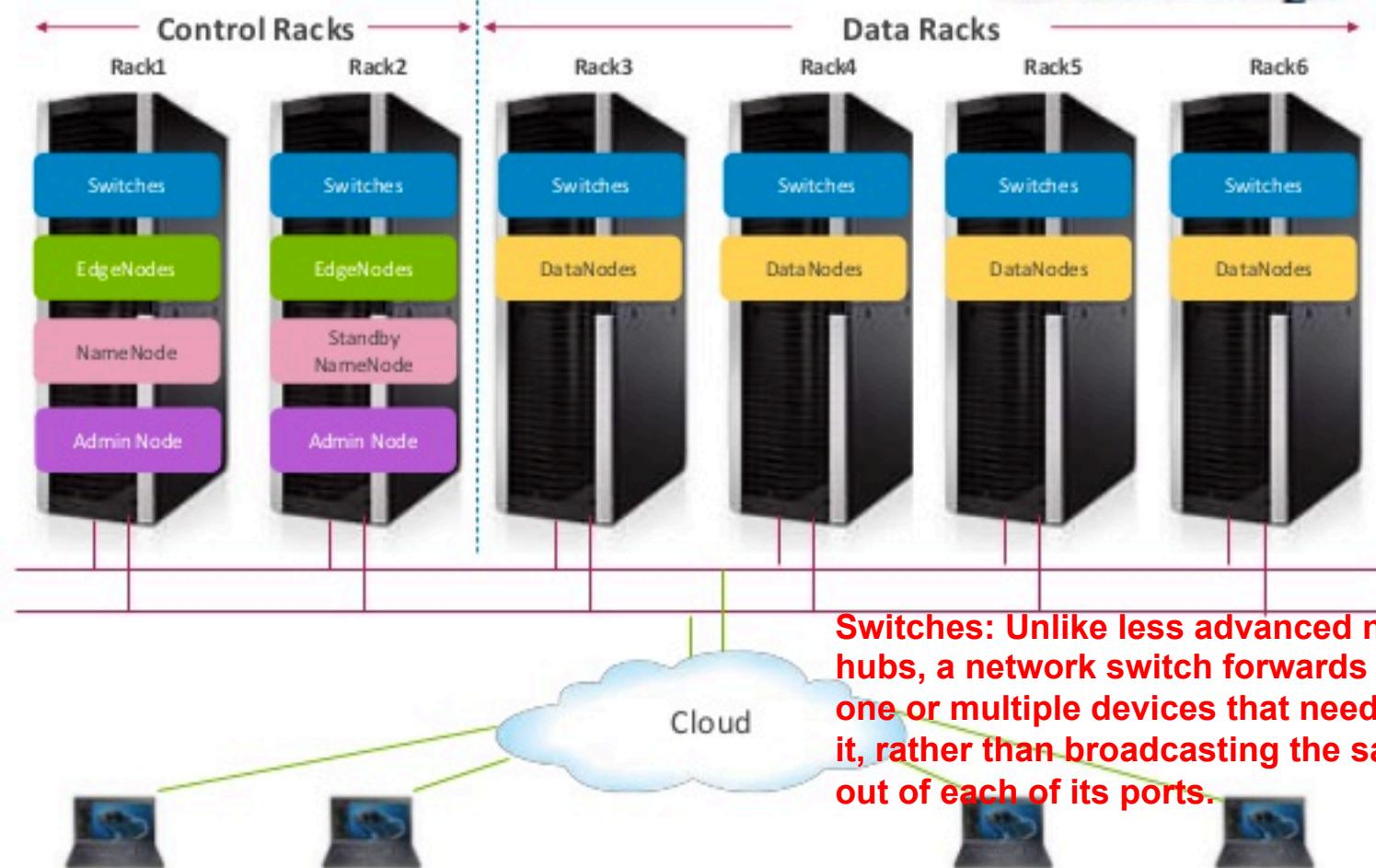
- Hadoop Snappy is a project for Hadoop that provide access to the snappy compression. <http://code.google.com/p/snappy/>
- Hadoop-Snappy can be used as an add-on for recent (released) versions of Hadoop that do not provide Snappy Codec support yet
- Hadoop-Snappy is being kept in synch with Hadoop Common
- Snappy is a compression/decompression library. It does not aim for maximum compression, or compatibility with any other compression library

HIGH AVAILABILITY SOLUTIONS



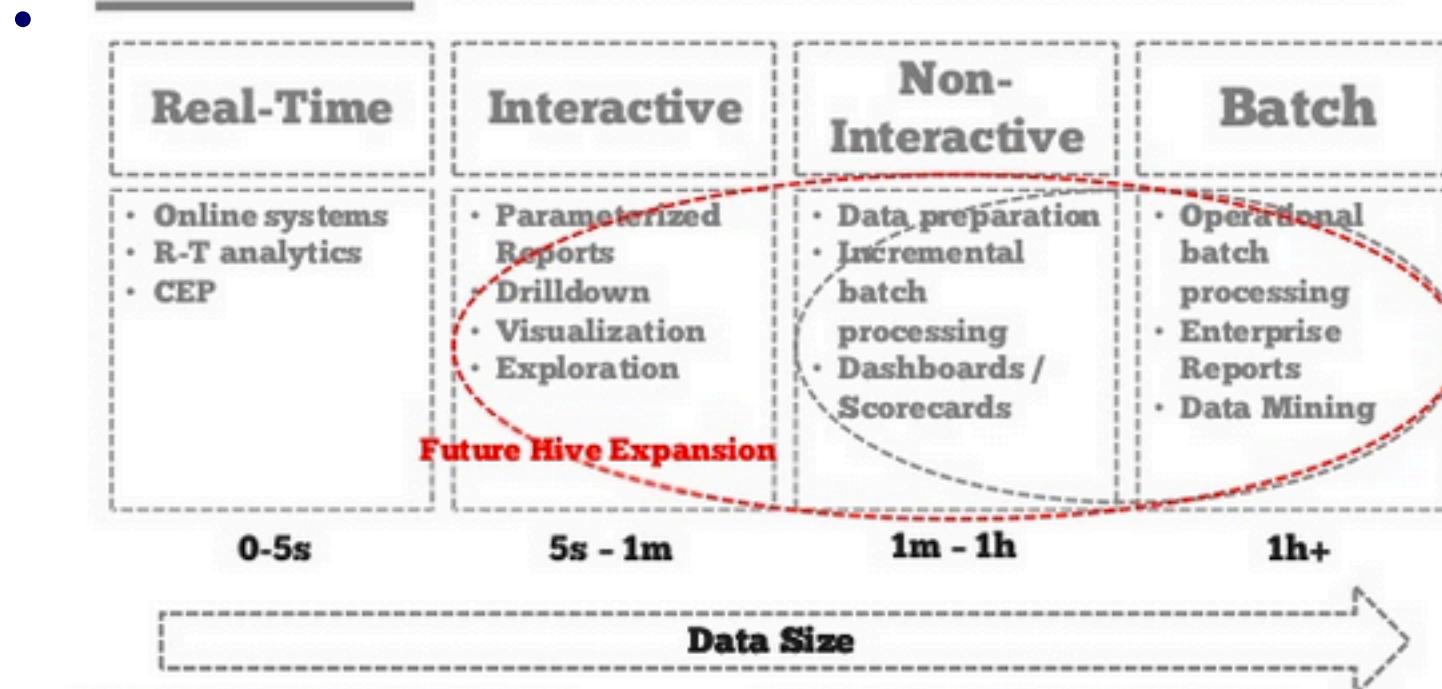
- Automatic blocks replication on 3 DataNodes – Rack awareness
- NameNode and Standby NameNode
- Quorum Journal Manager and Zookeeper
- Disaster Recovery by replication
- Hive and NameNode Metastores backup
- HDFS Snapshots

RACKS CONFIGURATION OVERVIEW





Stinger: Extending the sweet spot



Improve Latency & Throughput

- Query engine improvements
- New "Optimized RCFile" column store
- Next-gen runtime (elim's M/R latency)

Extend Deep Analytical Ability

- Analytics functions
- Improved SQL coverage
- Continued focus on core Hive use cases

© 2013 Hortonworks Inc.

Related projects to Hadoop

-
-

Other Hadoop-related projects at Apache include:

- [**Ambari™**](#): A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
- [**Avro™**](#): A data serialization system.
- [**Cassandra™**](#): A scalable multi-master database with no single points of failure.
- [**Chukwa™**](#): A data collection system for managing large distributed systems.
- [**HBase™**](#): A scalable, distributed database that supports structured data storage for large tables.
- [**Hive™**](#): A data warehouse infrastructure that provides data summarization and ad hoc querying.
- [**Mahout™**](#): A Scalable machine learning and data mining library.
- [**Pig™**](#): A high-level data-flow language and execution framework for parallel computation.
- [**Spark™**](#): A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- [**Tez™**](#): A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases. Tez is being adopted by Hive™, Pig™ and other frameworks in the Hadoop ecosystem, and also by other commercial software (e.g. ETL tools), to replace Hadoop™ MapReduce as the underlying execution engine.
- [**ZooKeeper™**](#): A high-performance coordination service for distributed applications.

Hadoop is IO bound/limited

- In computer science, I/O bound refers to a condition in which the time it takes to complete a computation is determined principally by the period of time spent waiting for input/output operations to be completed.
- This is the opposite of a task being CPU bound. This circumstance arises when the rate at which data is requested is slower than the rate it is consumed or, in other words, more time is spent requesting data than processing it.
- Hadoop is generally limited by IO
 - Storing compressed data in HDFS allows your hardware allocation to go further since compressed data is often 25% of the size of the original data. Furthermore, since MapReduce jobs are nearly always IO-bound, storing compressed data means there is less overall IO to do, meaning jobs run faster.

[

<http://www.cloudera.com/blog/2009/11/17/hadoop-at-twitter-part-1-splittable-izo-compression/>

Hadoop is IO bound/limited

- Furthermore, since MapReduce jobs are nearly always IO-bound, storing compressed data means there is less overall IO to do, meaning jobs run faster.
 - There are two caveats to this, however: some compression formats cannot be split for parallel processing,
 - and others are slow enough at decompression that jobs become CPU-bound, eliminating your gains on IO.
- Gzip files cannot be distributed
 - Imagine you have a 1.1 GB gzip file, and your cluster has a 128 MB block size. This file will be split into 9 chunks of size approximately 128 MB. In order to process these in parallel in a MapReduce job, a different mapper will be responsible for each chunk. But this means that the second mapper will start on an arbitrary byte about 128MB into the file. The contextful dictionary that gzip uses to decompress input will be empty at this point, which means the gzip decompressor will not be able to correctly interpret the bytes. The upshot is that large gzip files in Hadoop need to be processed by a single mapper, which defeats the purpose of parallelism. For an example of the second caveat in which jobs become CPU-bound, we can look to the bzip2

[

<http://www.cloudera.com/blog/2009/11/17/hadoop-at-twitter-part-1-splittable-iso-compression/>

Distributed File System

- **Single petabyte file system for entire cluster**
 - Managed by a single *namenode*.
 - Files are written, read, renamed, deleted, but append-only.
 - Optimized for streaming reads of large files.
- **Files are broken in to large blocks.**
 - Transparent to the client
 - Data is checksummed with CRC32
 - Replicated to several *datanodes*, for reliability
- **Client library talks to both namenode and datanodes**
 - Data is not sent through the namenode.
 - Throughput of file system scales nearly linearly.
- **Access from Java, C, or command line.**

-
- Advanced and detailed HDFS
 - Sync time
 - END

L2 Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - BLT: embarrassingly parallel problems (3)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)

V4

-
- **This next couple of sections will focus on Hadoop MAprEduce**
 - **MapReduce**
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5

Functional Programming

Functional Programming: Map and Reduce

- **The idea of Map, and Reduce is 40+ year old**
 - Present in all Functional Programming Languages.
 - See, e.g., APL, Lisp and ML
- **A key feature of functional languages is the concept of higher order functions, or functions that can accept other functions as arguments**
 - Map and Reduce are higher-order functions.
- **Alternate names for Map: Apply-All**

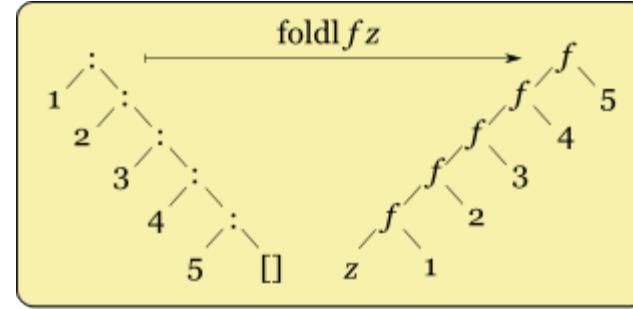
Although their purpose in the MapReduce framework is not the same as in their original forms

Map: A Higher Order Function

- **Let**
 - $F(x: \text{int})$ returns $r: \text{int}$
 - V be an array of integers.
- **$W = \text{map}(F, V)$**
 - $W[i] = F(V[i])$ for all i
 - i.e., apply F to every element of V
- **Map Examples in Haskell**
 - $\text{map } (+1) [1,2,3,4,5]$
 $\quad == [2, 3, 4, 5, 6]$
 - $\text{map } (\text{toLowerCase}) "abcDEFG12!@#"$
 $\quad == "abcdefg12!@#"$
 - $\text{map } (\text{`mod` } 3) [1..10]$
 $\quad == [1, 2, 0, 1, 2, 0, 1, 2, 0, 1]$

reduce: A Higher Order Function

- reduce also known as fold, accumulate, compress or inject
- Reduce/fold takes in a function and folds it in between the elements of a list.



Trace

Recursive defn but this unrolls into a loop

- Definition (recursive function)

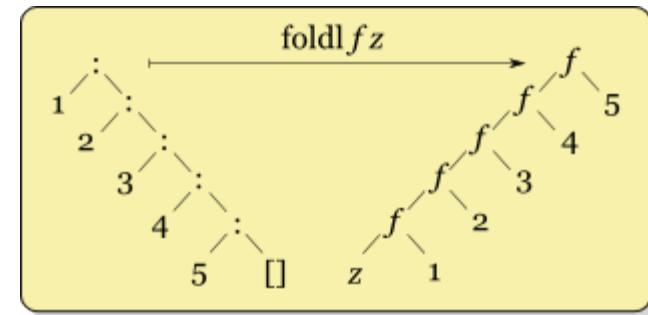
- $\text{foldl } f \ z \ [] = z$
 - $\text{foldl } f \ z \ (x:xs) = \text{foldl } f \ (f \ z \ x) \ xs$

- Examples

- $\text{foldl } (+) \ 0 \ [1..5] == 15$
 - $\text{foldl } (+) \ 10 \ [1..5] == 25$

Fold-Left in Haskell

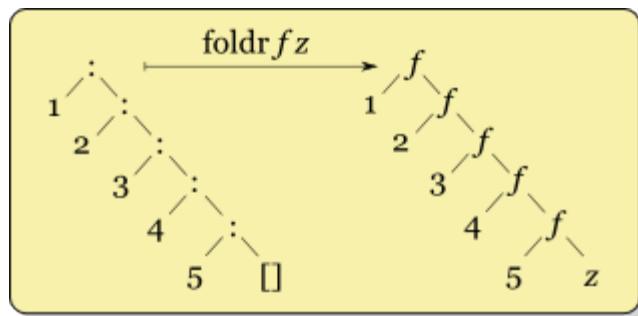
- **Definition (recursive function)**
 - $\text{foldl } f z [] = z$
 - $\text{foldl } f z (x:xs) = \text{foldl } f (f z x) xs$
- **Examples**
 - $\text{foldl } (+) 0 [1..5] == 15$
 - $\text{foldl } (+) 10 [1..5] == 25$



Fold-Right in Haskell

- **Definition**
 - $\text{foldr } f z [] = z$
 - $\text{foldr } f z (x:xs) = f x (\text{foldr } f z xs)$
- **Example**
 - $\text{foldr (div) 7 [34,56,12,4,23]} == 8$

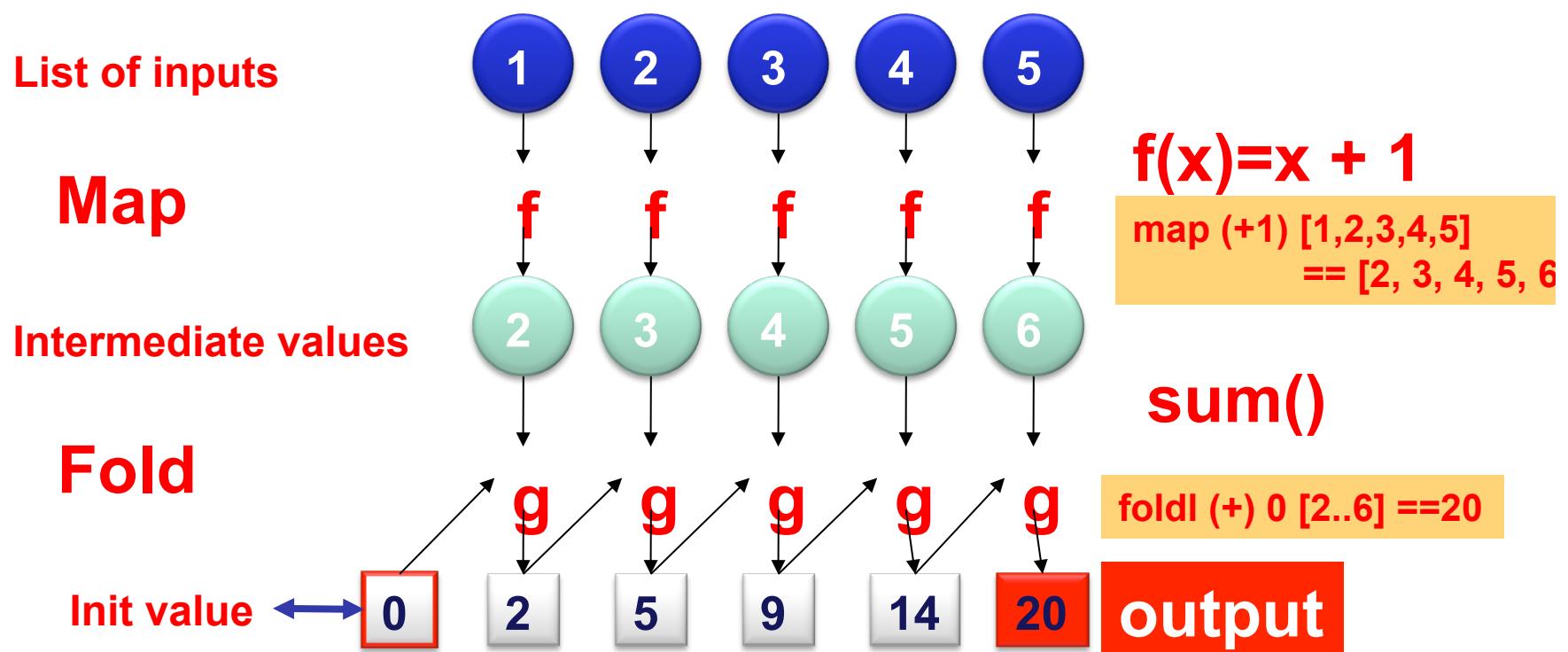
NOTE: $\text{foldl (div) 7 [34,56,12,4,23]} == 0$



MapReduce is derived from FP

MapReduce ~ Map + Fold from functional programming!

Recursive defn but this unrolls into a loop



FP is

- FP is all about the evaluation of mathematical functions and avoids changing state and mutable data.
- It is a declarative programming paradigm, which means programming is done with expressions.
- In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time.
- This is an abstract diagram, despite this you can see the potential for parallelism especially in the map phase

Map can be parallelized and so can Reduce

- **Map and Reduce (transform and aggregate)**
 - We can view map as a concise way to represent the transformation of a dataset (as defined by the function f).
 - In the same vein, we can view fold as an aggregation operation, as defined by the function g.
- **Map can be parallelized**
 - One immediate observation is that the application of **f** to each item in a list (or more generally, to elements in a large dataset) can be parallelized in a straightforward manner,
 - since each functional application happens in isolation.
 - In a cluster, these operations can be distributed across many different machines.
- **Fold/Reduce (barrier) can also be parallelized**
 - The fold operation, on the other hand, has more restrictions on data locality -- elements in the list must be brought "together" before the function **g** can be applied.

Map-Reduce primitives

- **Map-Reduce/Hadoop was inspired by the concept of functional languages:**
 - “Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages....
 - Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.” Jeffrey Dean and Sanjay Ghemawat, Google
- **MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, Google, 2004**

Functional Programming as a guide for Spark

- **apply()**
- **map()**
- **filter()**
- **reduce()**
- **Lambda functions**
- **List comprehensions**

Functional programming background

- Treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- It is a declarative programming paradigm, which means programming is done with expressions.
- In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time.
- No side effects
 - (side effects, i.e. changes in state that do not depend on the function inputs)
 - Can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional Programming

In computer science, **functional programming** is a [programming paradigm](#), a style of building the structure and elements of computer programs, that treats [computation](#) as the evaluation of [mathematical functions](#) and avoids [changing-state](#) and [mutable data](#). It is a [declarative programming paradigm](#), which means programming is done with [expressions](#). In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time. Eliminating [side effects](#), i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional programming has its roots in [lambda calculus](#), a [formal system](#) developed in the 1930s to investigate [computability](#), the [Entscheidungsproblem](#), function definition, function application, and [recursion](#). Many functional programming languages can be viewed as elaborations on the lambda calculus. In the other well-known declarative [programming paradigm](#), [logic programming](#), [relations](#) are at the base of respective languages.^[1]

In contrast, [imperative programming](#) changes state with commands in the source language, the most simple example being assignment. Imperative programming does have functions, not in the mathematical sense, but in the sense of [subroutines](#). They can have [side effects](#) that may change the value of program state. Functions without return values therefore make sense. Because of this, they lack [referential transparency](#), i.e. the same language expression can result in different values at different times depending on the state of the executing program.^[1]

Functional programming languages, especially [purely functional](#) ones such as [Hope](#) and [Rex](#), have largely been emphasized in [academia](#) rather than in commercial software development. However, prominent functional programming languages such as [Common Lisp](#), [Scheme](#),^[2]^[3]^[4]^[5] [Clojure](#), [Racket](#),^[6] [Erlang](#),^[7]^[8]^[9] [OCaml](#),^[10]^[11] [Haskell](#),^[12]^[13] and [F#](#)^[14]^[15] have been used in industrial and commercial applications by a wide variety of organizations. Functional programming is also supported in [Large-Scale Machine Learning](#), [MIDS](#), UC Berkeley © 2015 James G. Shanahan Contact:james.shanahan@gmail.com

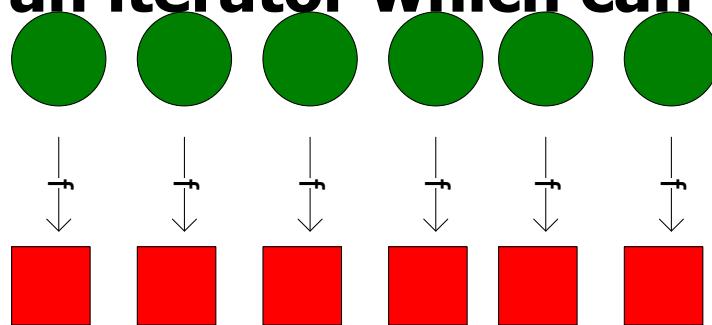
Higher-Order Functions

- A key feature of functional languages is the concept of higher order functions, or functions that can accept other functions as arguments (e.g., APL, Lisp and ML)
- A higher-order function is a function that takes another function as a parameter
- They are “higher-order” because it’s a function of a function
- Examples
 - Map (aka apply to all)
 - Reduce (aka fold)
 - Filter
- Lambda works great as a parameter to higher-order functions if you can deal with its limitations

Map

`map(function, iterable, ...)`

- **Map applies function to each element of iterable and creates a list of the results**
- **You can optionally provide more iterables as parameters to map and it will place tuples in the result list**
- **Map returns an iterator which can be cast to list**

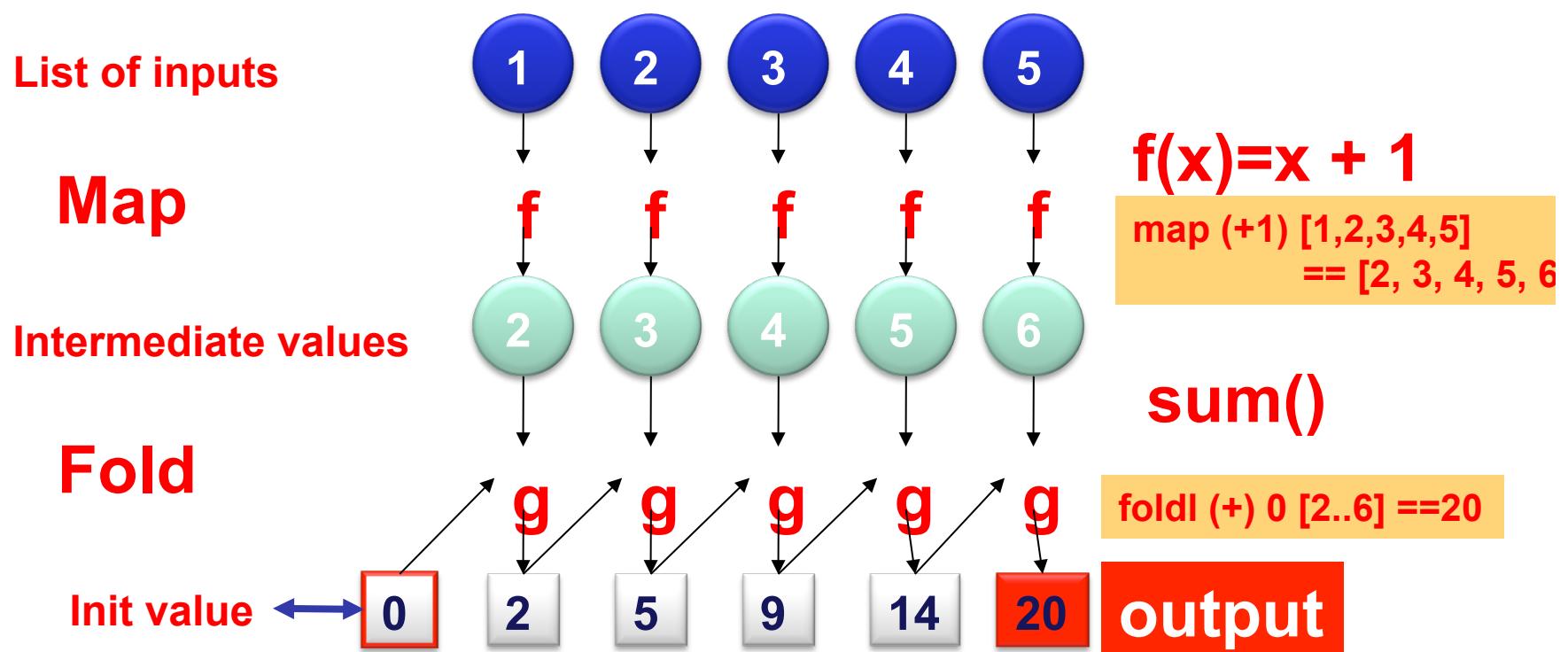


This is an abstract diagram, despite this you can see the potential for parallelism

MapReduce is derived from FP

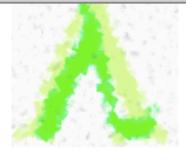
MapReduce ~ Map + Fold from functional programming!

Recursive defn but this unrolls into a loop



Lambda functions

- Shorthand version of def statement, useful for “Inlining” functions and other situations where it's convenient to keep the code of the function close to where it's needed
- Can only contain an expression in the function definition, not a block of statements (e.g., no if statements, etc)
- A lambda returns a function; the programmer can decide whether or not to assign this function to a name



Python 2 Tutorial

- History and Philosophy of Python
- Why Python?
- Interactive Mode
- Execute a Script
- Structuring with Indentation
- Data Types and Variables
- Operators
- input and raw_input via the keyboard
- Conditional Statements
- While Loops
- For Loops
- Formatted output
- Output with Print
- Sequential Data Types
- Dictionaries
- Sets and Frozen Sets
- Shallow and Deep Copy

Lambda, filter, reduce and map

Lambda Operator

Some like it, others hate it and many are afraid of the lambda operator. We are confident that you will like it, when you have finished with this chapter of our tutorial. If not, you can learn all about "[List Comprehensions](#)", Guido van Rossum's preferred way to do it, because he doesn't like Lambda, map, filter and reduce either.

becasu The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination with the functions filter(), map() and reduce(). The lambda feature was added to Python due to the demand from Lisp programmers.

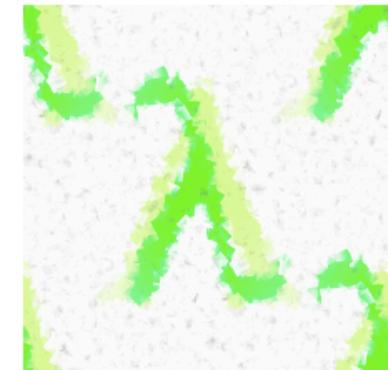
The general syntax of a lambda function is quite simple:

lambda argument_list: expression

The argument list consists of a comma separated list of arguments and the expression is an arithmetic expression using these arguments. You can assign the function to a variable to give it a name.

The following example of a lambda function returns the sum of its two arguments:

```
>>> f = lambda x, y : x + y
>>> f(1,1)
2
```



The map() Function

The advantage of the lambda operator can be seen when it is used in combination with the map() function. map() is a function with two arguments:

```
r = map(func, seq)
```

The first argument *func* is the name of a function and the second a sequence (e.g. a list) *seq*. *map()* applies the function *func* to all the elements of the sequence *seq*. It returns a new list with the elements changed by *func*

```
def fahrenheit(T):
    return ((float(9)/5)*T + 32)
def celsius(T):
    return (float(5)/9)*(T-32)
temp = (36.5, 37, 37.5, 39)

F = map(fahrenheit, temp)
C = map(celsius, F)
```

In the example above we haven't used lambda. By using lambda, we wouldn't have had to define and name the functions fahrenheit() and celsius(). You can see this in the following interactive session:

```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]
>>> Farenheit = map(lambda x: (float(9)/5)*x + 32, Celsius)
>>> print Farenheit
[102.56, 97.70000000000003, 99.14000000000001, 100.03999999999991]
```

Lambda example

- Simple example:

```
>>> def sum(x,y): return x+y  
>>> ...  
>>> sum(1,2)  
3
```

```
>>> sum2 = lambda x, y: x+y  
>>> sum2(1,2)  
3
```

Closure (computer programming)

From Wikipedia, the free encyclopedia

Closure

For other uses of this term, including in mathematics and computer science, see [Closure](#).

Not to be confused with [Clojure](#).

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with [first-class functions](#). Operationally, a closure is a data structure storing a function^[a] together with an environment:^[1] a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the function to access those captured variables later.

Example The following program defines a function `startAt` that returns a function `incrementBy`. The nested function `incrementBy` adds its argument to the value of `x`, even though `x` is not local to `incrementBy`. It does this by capturing the `y` value to the `x` value, and referring to it once invoked:

```
function startAt(x)
  function incrementBy(y)
    return x + y
  return incrementBy

variable closure1 = startAt(1)
variable closure2 = startAt(2)
```

Note that, as `startAt` returns a function, it is not a closure. It returns `4`, while invoking `closure1` returns `5`. The associated environments differ, and both closures capture different values, thus evaluating to different results.

A closure is a data structure storing a function[a] together with an environment:

- **a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.**
- **A closure is a function that encloses its surrounding state by referencing fields external to its body. The enclosed state remains across invocations of the closure.**

Closure (computer programming)

From Wikipedia, the free encyclopedia

For other uses of this term, including in mathematics and computer science, see Closure.

Not to be confused with Clojure.

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with **first-class functions**. Operationally, a closure is a data structure storing a **function**^[a] together with an environment:^[1] a mapping associating each **free variable** of the function (variables that are used locally, but defined in an enclosing scope) with the **value** or **storage location** the name was bound to at the time the closure was created.^[b] A closure—unlike a plain function—allows the function to access those *captured variables* through the closure's reference to them, even when the function is invoked outside their scope.

Example The following program fragment defines a **higher-order function** `startAt` with a parameter `x` and a **nested function** `incrementBy`. The nested function `incrementBy` has access to `x`, because `incrementBy` is in the lexical scope of `x`, even though `x` is not local to `incrementBy`. The function `startAt` returns a closure containing the function `incrementBy`, which adds the `y` value to the `x` value, and a reference to the variable `x` from this invocation of `startAt`, so `incrementBy` will know where to find it once invoked:

```
function startAt(x)
    function incrementBy(y)
        return x + y
    return incrementBy

variable closure1 = startAt(1)
variable closure2 = startAt(5)
```

A closure is a data structure storing a function[a] together with an environment:[1] a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing

Note that, as `startAt` returns a function, the variables `closure1` and `closure2` are of **function type**. Invoking `closure1(3)` will return `4`, while invoking `closure2(3)` will return `8`. While closures `closure1` and `closure2` have the same function `incrementBy`, the associated environments differ, and invoking the closures will bind the name `x` to two distinct variables in the two invocations, with different values, thus evaluating the function to different results.

Lambda as a closure

Spark Essentials: Transformations

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

A closure is a function that encloses its surrounding state by referencing fields external to its body. The enclosed state remains across invocations of the closure.

closures

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

Map Example

Example

```
1  nums = [0, 4, 7, 2, 1, 0, 9, 3, 5, 6, 8, 0, 3]
2
3  nums = list(map(lambda x : x % 5, nums)) Lambda/inline
4
5  print(nums)
6  #[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
7
8
9  def mod5(val) :
10     return x % 5
11
12 nums1 = list(map(mod5, nums))
print(nums1)
#[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
```

apply()

- In very general contexts, you may not know ahead of time how many arguments need to get passed to a function (perhaps the function itself is built dynamically)
- The apply() function calls a given function with a list of arguments packed in a tuple:
`def sum(x, y): return x+y
apply(sum, (3, 4))`
7
- Apply can handle functions defined with def or with lambda

Map Example: distance from origin

Goal: given a list of three dimensional points in the form of tuples, create a new list consisting of the distances of each point from the origin

Loop Method:

- $\text{distance}(x, y, z) = \sqrt{x^2 + y^2 + z^2}$
- loop through the list and add results to a new list

Map Problem: distance from origin

Solution

```
1 from math import sqrt  
2  
3 points = [(2, 1, 3), (5, 7, -3), (2, 4, 0), (9, 6, 8)]  
4  
5 def distance(point) :  
6     x, y, z = point  
7     return sqrt(x**2 + y**2 + z**2)  
8  
9 distances = list(map(distance, points))
```

Filter: higher order function

`filter(function, iterable)`

- The filter runs through each element of iterable (any iterable object such as a List or another collection)
- It applies function to each element of iterable
- If function returns True for that element then the element is put into a List
- This list is returned from filter in versions of python under 3
- In python 3, filter returns an iterator which must be cast to type list with list()

Filter Example: filter all non-zeros

Example

```
1  nums = [0, 4, 7, 2, 1, 0, 9, 3, 5, 6, 8, 0, 3]
2
3  nums = list(filter(lambda x : x != 0, nums))
4
5  print(nums)          #[4, 7, 2, 1, 9, 3, 5, 6, 8, 3]
6
```

Filter Problem

```
NaN = float("nan")
scores = [[NaN, 12, .5, 78, math.pi],
          [2, 13, .5, .7, math.pi / 2],
          [2, NaN, .5, 78, math.pi],
          [2, 14, .5, 39, 1 - math.pi]]
```

Goal: given a list of lists containing answers to an algebra exam, filter out those that did not submit a response for one of the questions, denoted by NaN

Filter Problem

Solution

```
1  NaN = float("nan")
2  scores = [[NaN, 12, .5, 78, pi],[2, 13, .5, .7, pi / 2],
3           [2,NaN, .5, 78, pi],[2, 14, .5, 39, 1 - pi]]
4  #solution 1 - intuitive
5  def has_NaN(answers) :
6      for num in answers :
7          if isnan(float(num)) :
8              return False
9      return True
0  valid = list(filter(has_NaN, scores))
1  print(valid)
2
3  #Solution 2 - sick python solution
4  valid2 = list(filter(lambda x : NaN not in x, scores))
5  print(valid2)
```

Lambda/inline

Reduce

`reduce(function, iterable[, initializer])`

- Reduce will apply function to each element in iterable along with the sum so far and create a cumulative sum of the results
- function must take two parameters
- If initializer is provided, initializer will stand as the first argument in the sum
- Unfortunately in python 3 reduce() requires an import statement
 - `from functools import reduce`

Reduce Example

Example

```
1  nums = [1, 2, 3, 4, 5, 6, 7, 8]
2
3  nums = list(reduce(lambda x, y : (x, y), nums))
4
5  Print(nums)          # (((((1, 2), 3), 4), 5), 6), 7), 8)
6
7
```

Reduce Problem

Goal: given a list of numbers I want to find the average of those numbers in a few lines using `reduce()`

For Loop Method:

- sum up every element of the list
- divide the sum by the length of the list

Reduce Problem

Solution

```
1  nums = [92, 27, 63, 43, 88, 8, 38, 91, 47, 74, 18, 16,  
2      29, 21, 60, 27, 62, 59, 86, 56]  
3  
4  sum = reduce(lambda x, y : x + y, nums) / len(nums)
```

MapReduce in Python

A framework for processing huge datasets on certain kinds of distributable problems

Map Step:

- master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes.
- worker node may chop its work into yet small pieces and redistribute again

MapReduce

Reduce Step:

- master node then takes the answers to all the sub-problems and combines them in a way to get the output

MapReduce

Problem: Given an email how do you tell if it is spam?

- Count occurrences of certain words. If they occur too frequently the email is spam.

MapReduce in Python: single core

map_reduce.py

```
1 email = ['the', 'this', 'annoy', 'the', 'the', 'annoy']
2
3 def inEmail (x):
4     if (x == "the"):
5         return 1;
6     else:
7         return 0;
8
9 map(inEmail, 1)                      #[1, 0, 0, 0, 1, 1, 0]
10
11 reduce((lambda x, xs: x + xs), map(inEmail, email)) #3
```

-
- Purely functional

Purely functional

Every function in Haskell is a function in the mathematical sense (i.e., "pure"). Even side-effecting IO operations are but a description of what to do, produced by pure code. There are no statements or instructions, only expressions which cannot mutate variables (local or global) nor access state like time or random numbers.

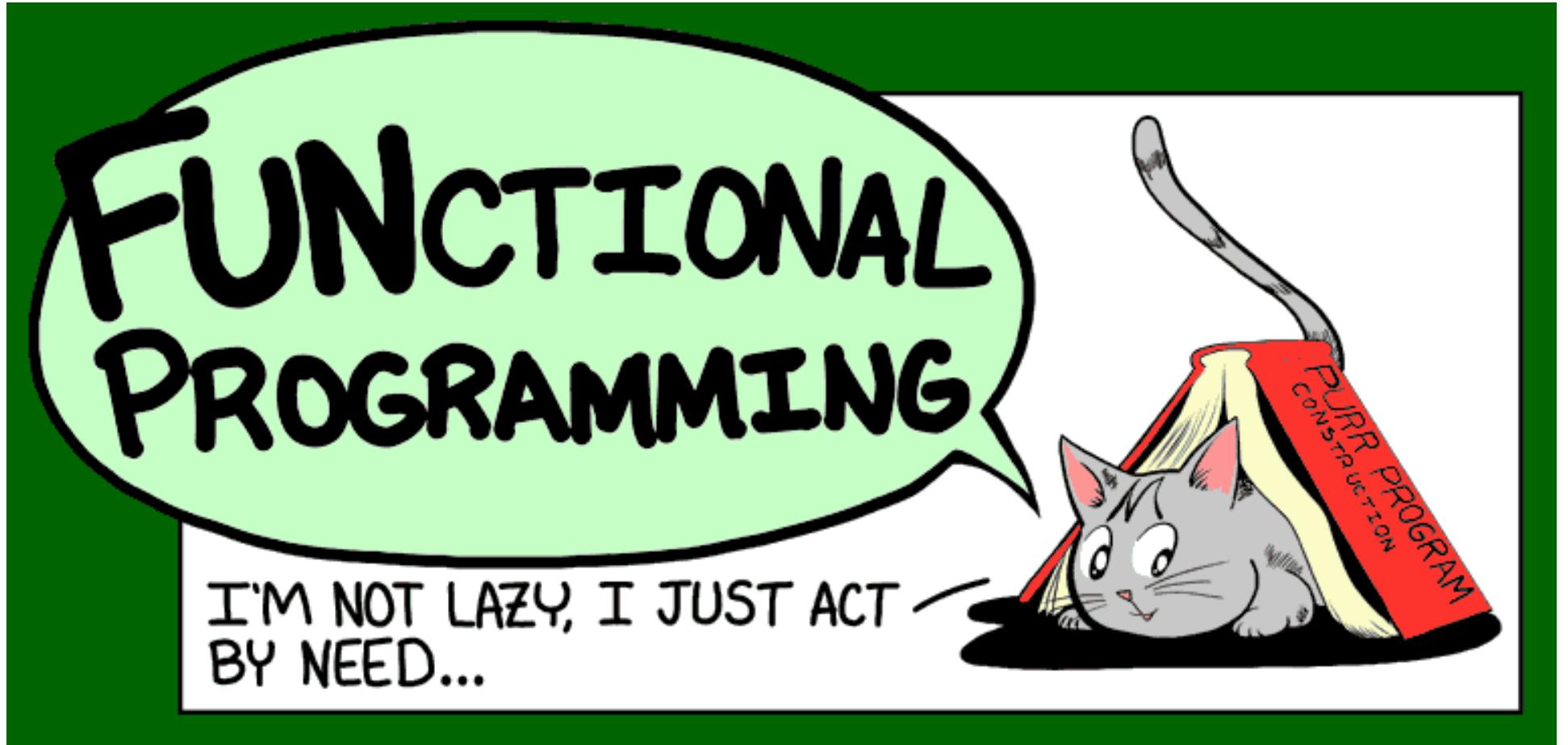
The following function takes an integer and returns an integer. By the type it cannot do any side-effects whatsoever, it cannot mutate any of its arguments.

```
square :: Int -> Int
square x = x * x
```

Functional Programming Review

- Functional operations do not modify data structures: They always create new ones
- Original data still exists in unmodified form
- Data flows are implicit in program design
- Order of operations does not matter
- Lists are primitive data types

Hadoop is not so lazy but Spark is!



-
- In this section we talked about functional programming and have seen the map and reduce higher order functions and we seen the potential to parallelize at least the map function
 - Over the next couple of sections we will see how the Spark framework has adopted FP constructs (albeit not purely observing the FP high standards)
 - Lazy evaluation
 - Data is immutable
 - Map, reduce, and 80 other functions

-
- In this section we talked about functional programming and have seen the map and reduce higher order functions and we seen the potential to parallelize at least the map function
 - Over the next couple of sections we will see how the map-reduce paradigm has been adapted in Hadoop map-reduce framework (albeit not purely observing the FP high standards)

L2 Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - BLT: embarrassingly parallel problems (3)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)

V4

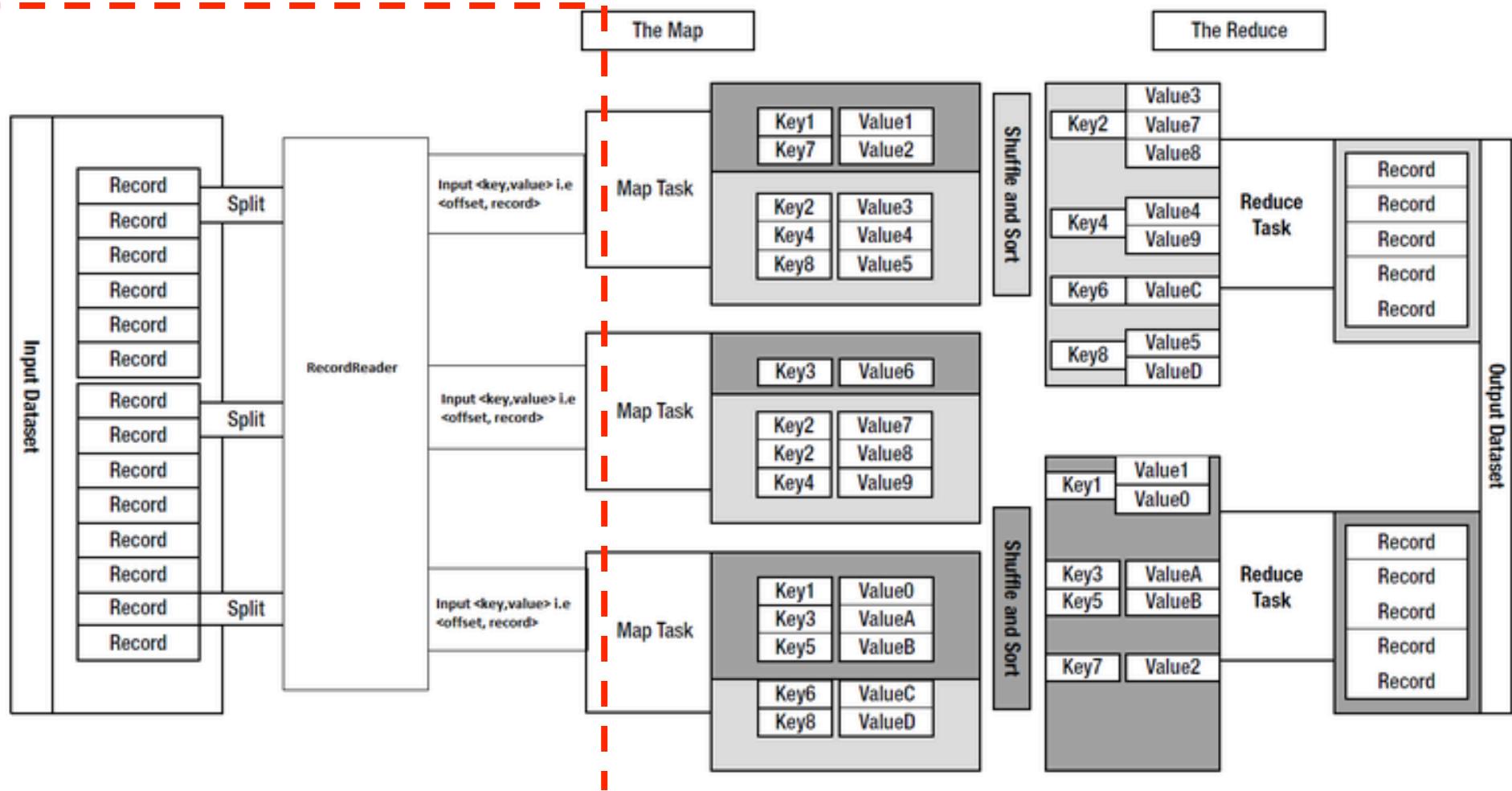
-
- **Move on from functional programming**
 - **Hadoop map-reduce framework for big data processing**

From functional programming to MapReduce

- **Maps correspond and reduce approx.**
 - The map phase in MapReduce roughly corresponds to the map operation in functional programming, whereas the reduce phase in MapReduce roughly corresponds to the fold operation in functional programming.
- **MapReduce execution framework does it all**
 - HDFS takes care of the data
 - While the MR framework takes care of everything else (almost)
 - As we will discuss in detail shortly, the MapReduce execution framework coordinates the map and reduce phases of processing over large amounts of data on large clusters of commodity machines.

First things first: Data

- File – a bag of (key, value) records
 - Data is chunked in blocks
 - Data type: key-value *records*

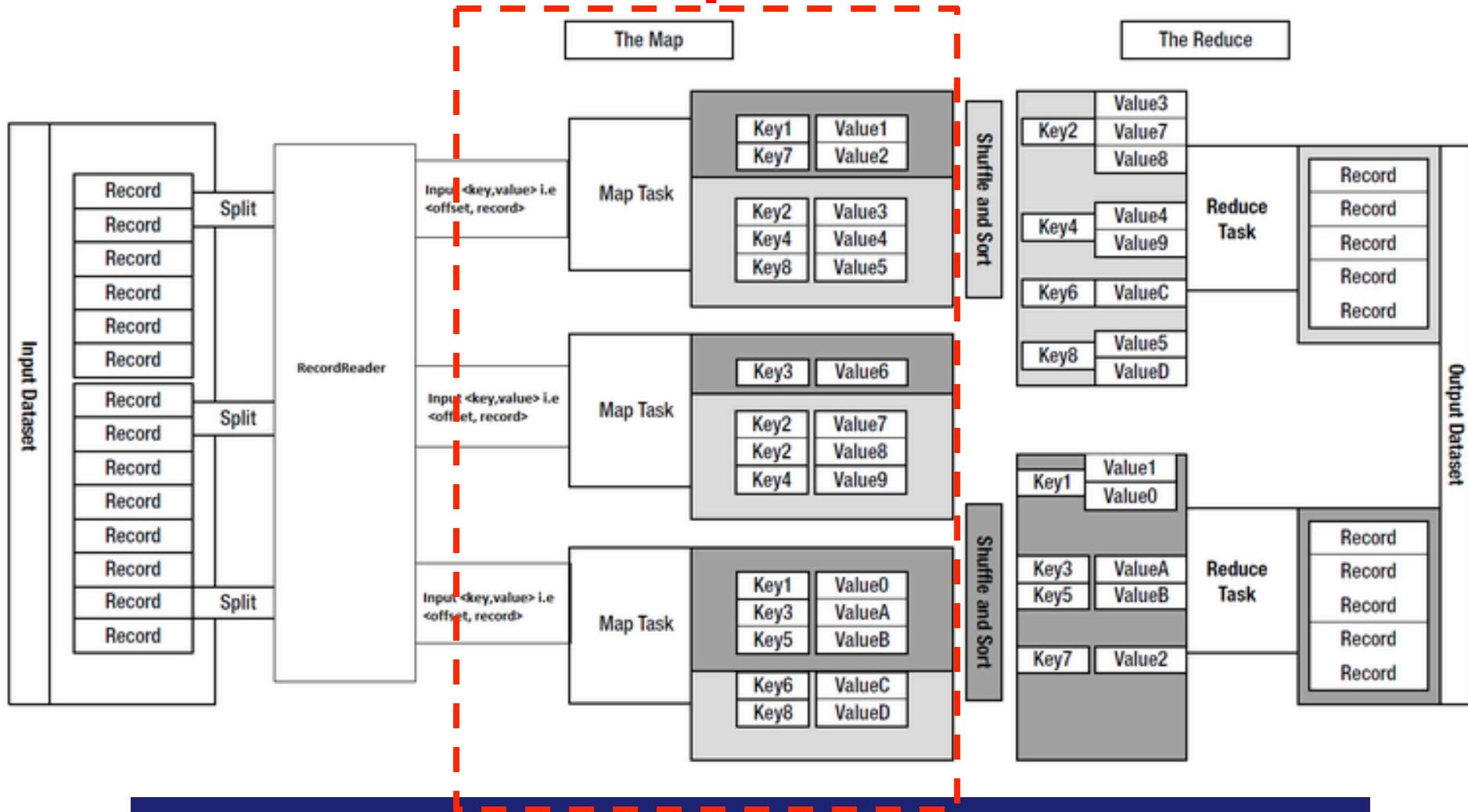


MapReduce Programming Model

- Data type: key-value *records*
- File – a bag of (key, value) records
- Map function:
$$(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter}) \quad \# \text{intermediate values}$$
 - Takes a key-value pair and outputs a set of key-value pairs, e.g., key is the line number, value is a single line in the file
 - There is one Map call for every (k_{in}, v_{in}) pair
- Reduce function:
$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$
 - All values V_{inter} with same key K_{inter} are reduced together and processed in V_{inter} order
 - There is one Reduce function call per unique key K_{inter}

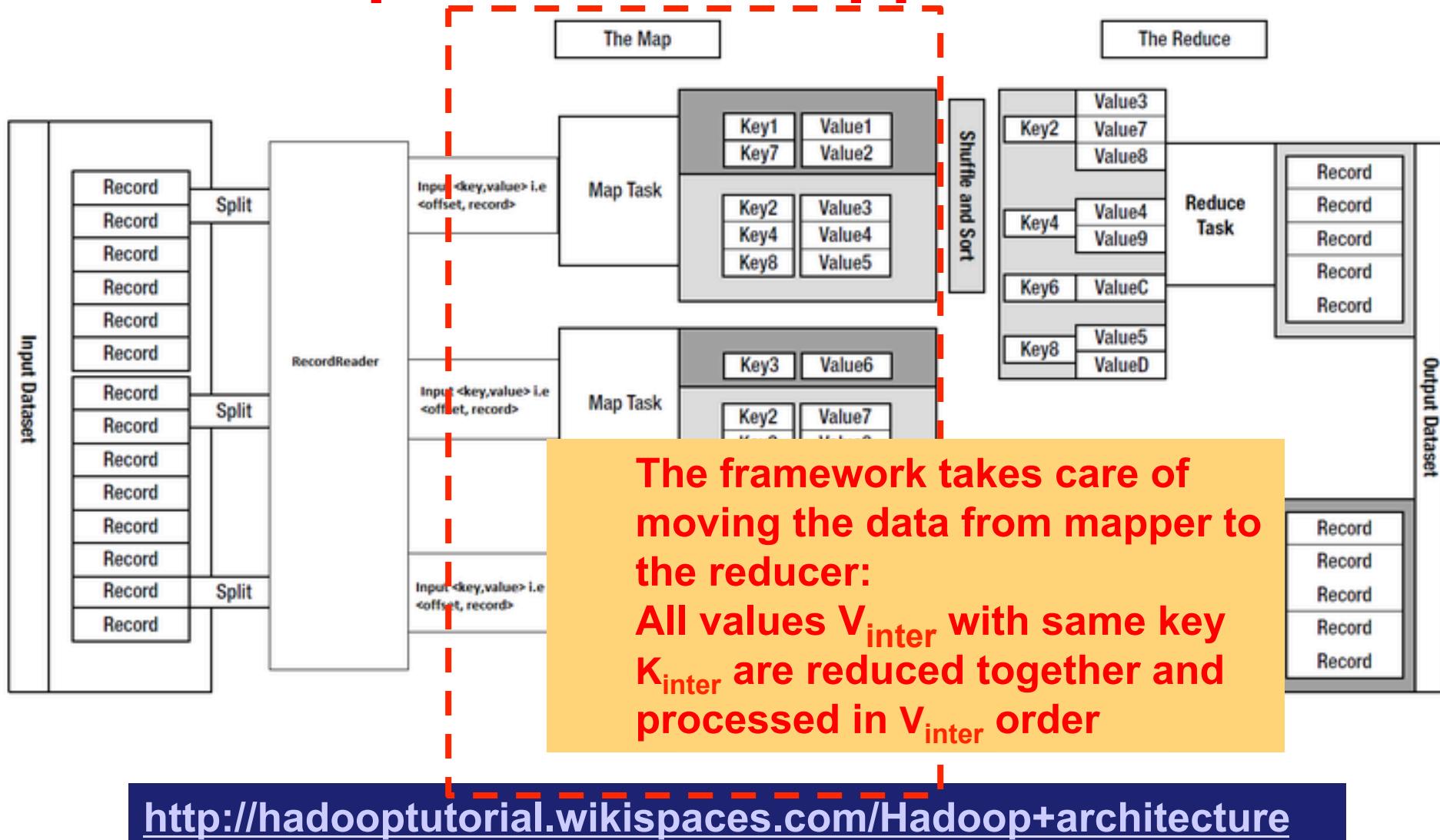
-
- **What is the restriction on the size of (key, value) pairs?**
 - **Each pair must be small enough to fit into one single machine, e.g, a document should be fine, an image should be fine but 1PB value is probably not.**

Map Reduce: Data flow



<http://hadooptutorial.wikispaces.com/Hadoop+architecture>

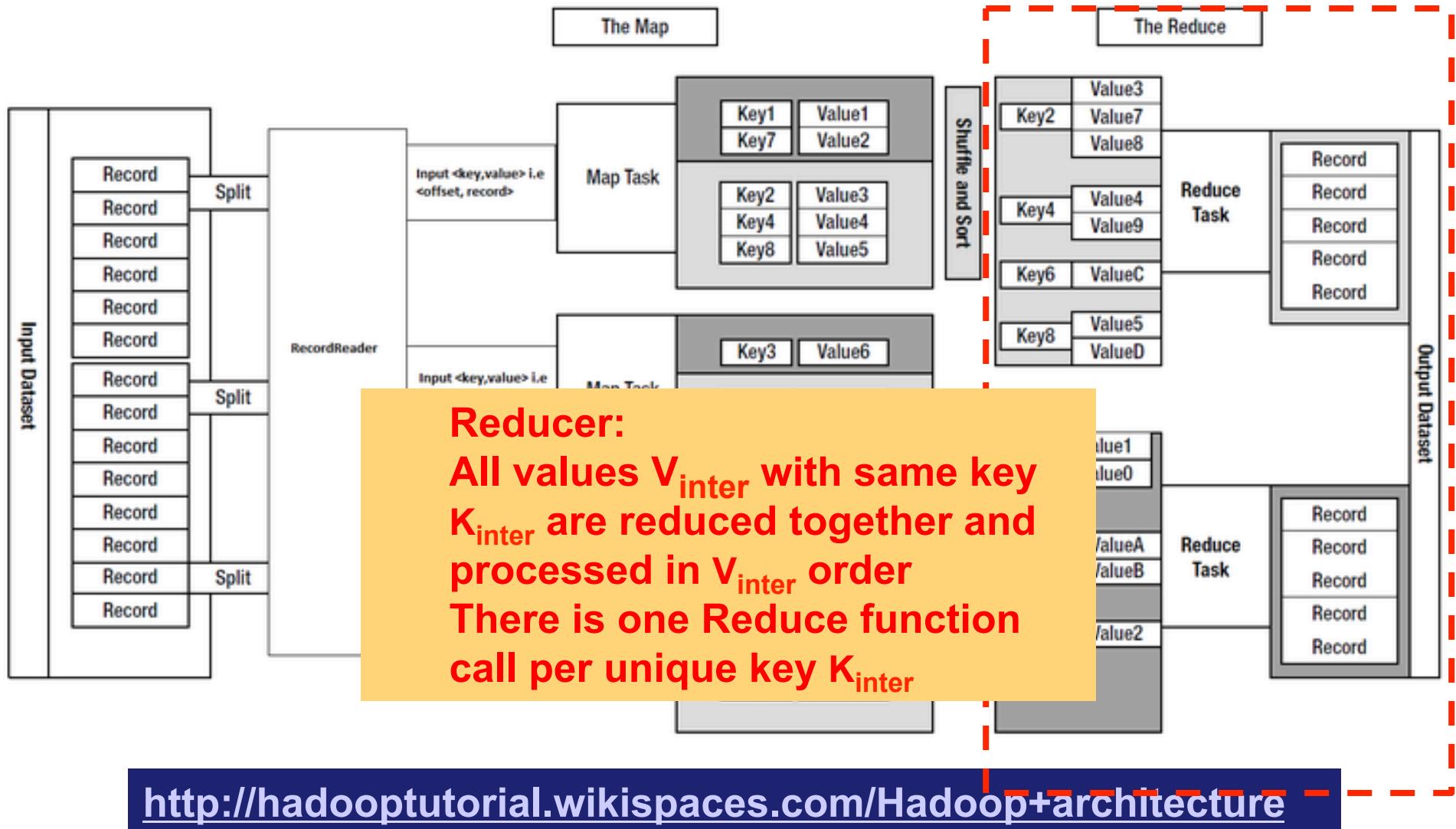
Map Reduce: Mapper → Reducer



The framework takes care of moving the data from mapper to the reducer:
All values V_{inter} with same key K_{inter} are reduced together and processed in V_{inter} order

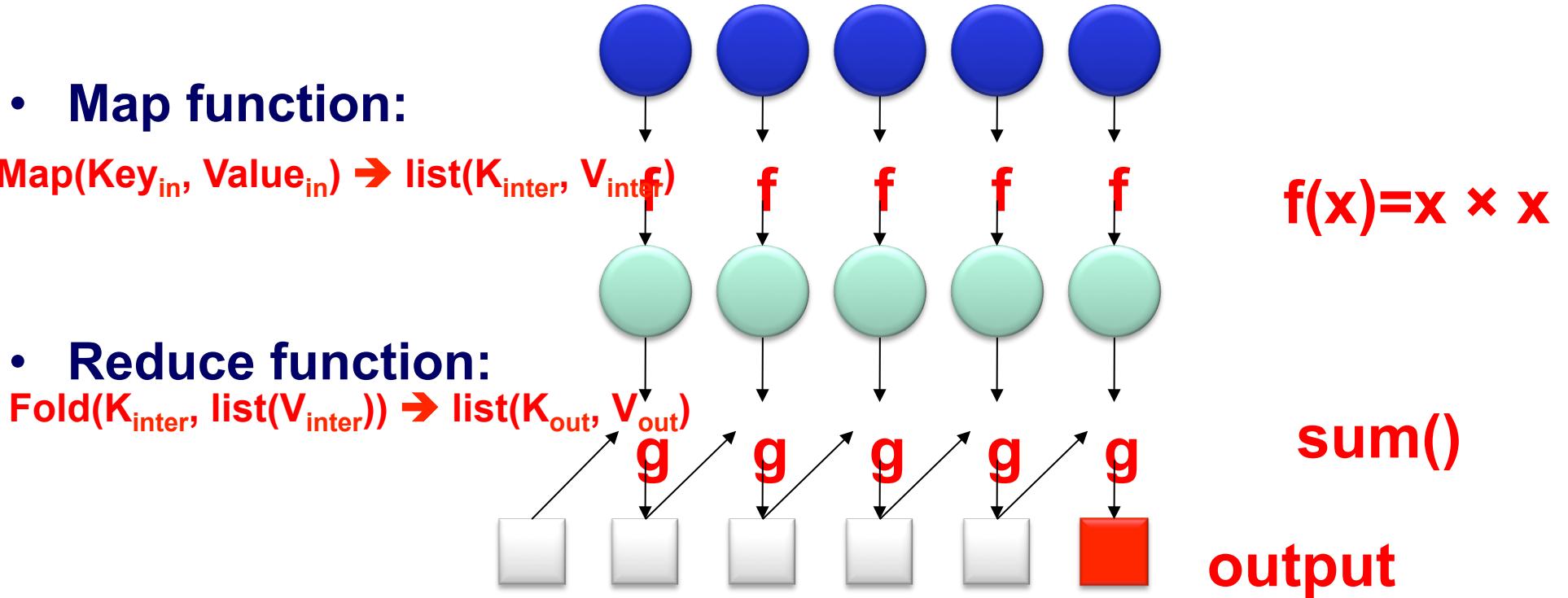
<http://hadooptutorial.wikispaces.com/Hadoop+architecture>

Map Reduce: Reduce



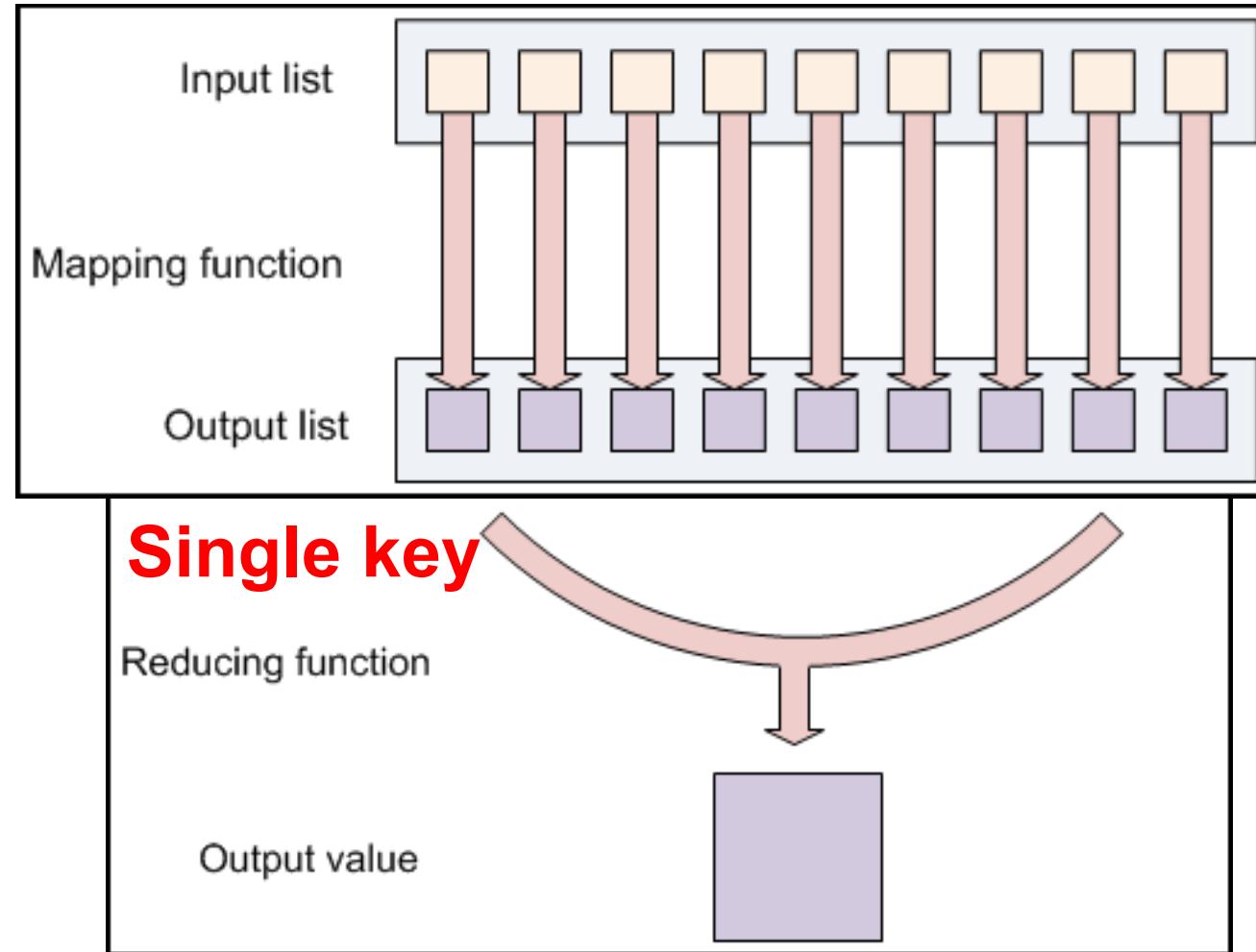
MapReduce: Example Sum(X^2)

- MapReduce is oriented around Key-Value pairs records
- Data is stored as **key-value records** in Hadoop
 - output= Run(Mapper=f($x=x \times x$, Reducer=sum(), input=inFile)



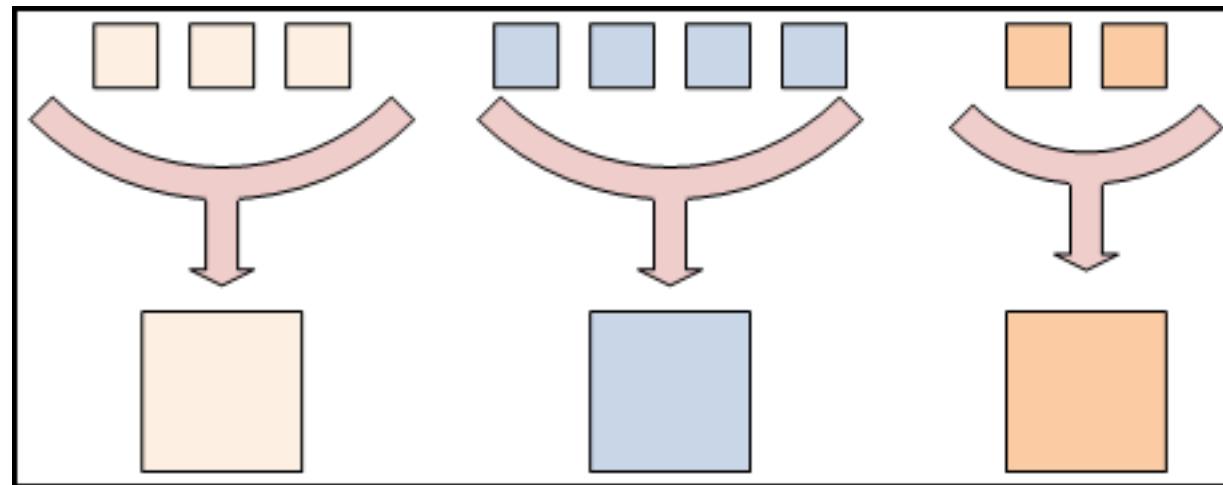
MapReduce: Example Sum(X^2)

•



Reduce groups data with common keys

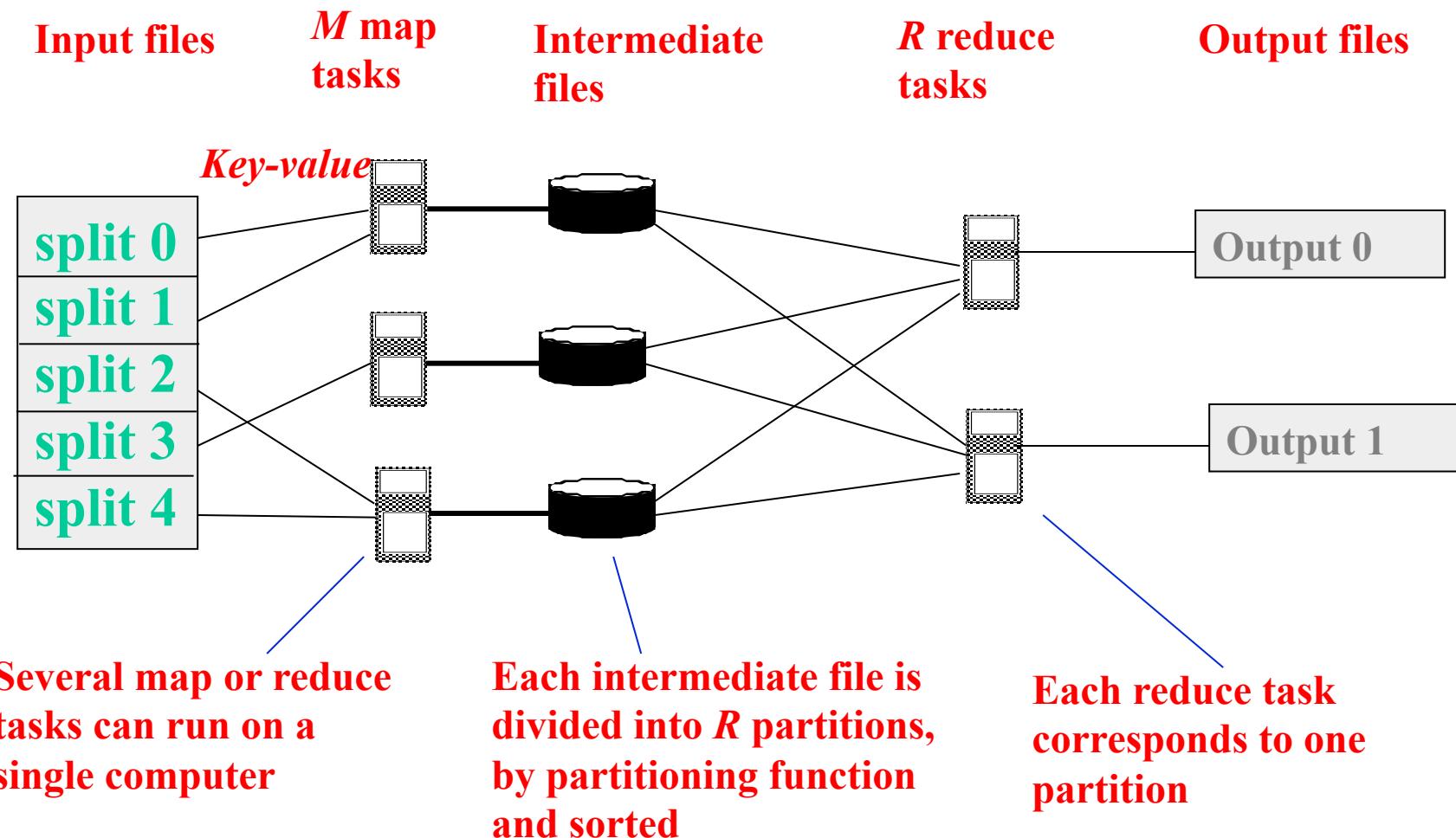
• ...



MapReduce a framework for big data

- **MapReduce codifies a generic recipe for processing large datasets that consists of two stages.**
 - In the first stage, a user-specified computation is applied over all input records in a dataset.
 - These operations occur in parallel and yield intermediate output that is then aggregated by another user-specified computation.
- **Programmer and execution framework synergy**
- **Just provide the mapper and reducer functions**
 - The programmer defines these two types of computations, and the execution framework coordinates the actual processing (very loosely, MapReduce provides a functional abstraction).
- **Very powerful: many interesting algorithms can be expressed quite concisely**
 - Although such a two-stage processing structure may appear to be very restrictive, many interesting algorithms can be expressed quite concisely, especially if one decomposes complex algorithms into a sequence of MapReduce jobs

In summary: Map/Reduce Cluster



In summary: Data Records flow from Map to Reduce

- **Framework will convert each record of input into a key/value pair**
 - The framework will convert each record of input into a key/value pair, and each pair will be input to the map function once.
- **The map output pairs are grouped and sorted by key.**
 - The map output is a set of key/value pairs—nominally one pair that is the transformed input pair, but it is perfectly acceptable to output multiple pairs.
- **The reduce function is called one time for each key, in sort sequence, with the key and the set of values that share that key.**
- **Reduce outputs to file**
 - The reduce method may output an arbitrary number of key/value pairs, which are written to the output files in the job output directory.
 - *If the reduce output keys are unchanged from the reduce input keys, the final output will be sorted.*

-
- **So that's MapReduce!!**
 - **Aren't excited about what you can do with MapReduce?!**
 - **Next section we will get concrete and we will work thru a couple of examples of using Hadoop MapReduce to solve problems**

L2 Parallel computing, MapReduce, Hadoop

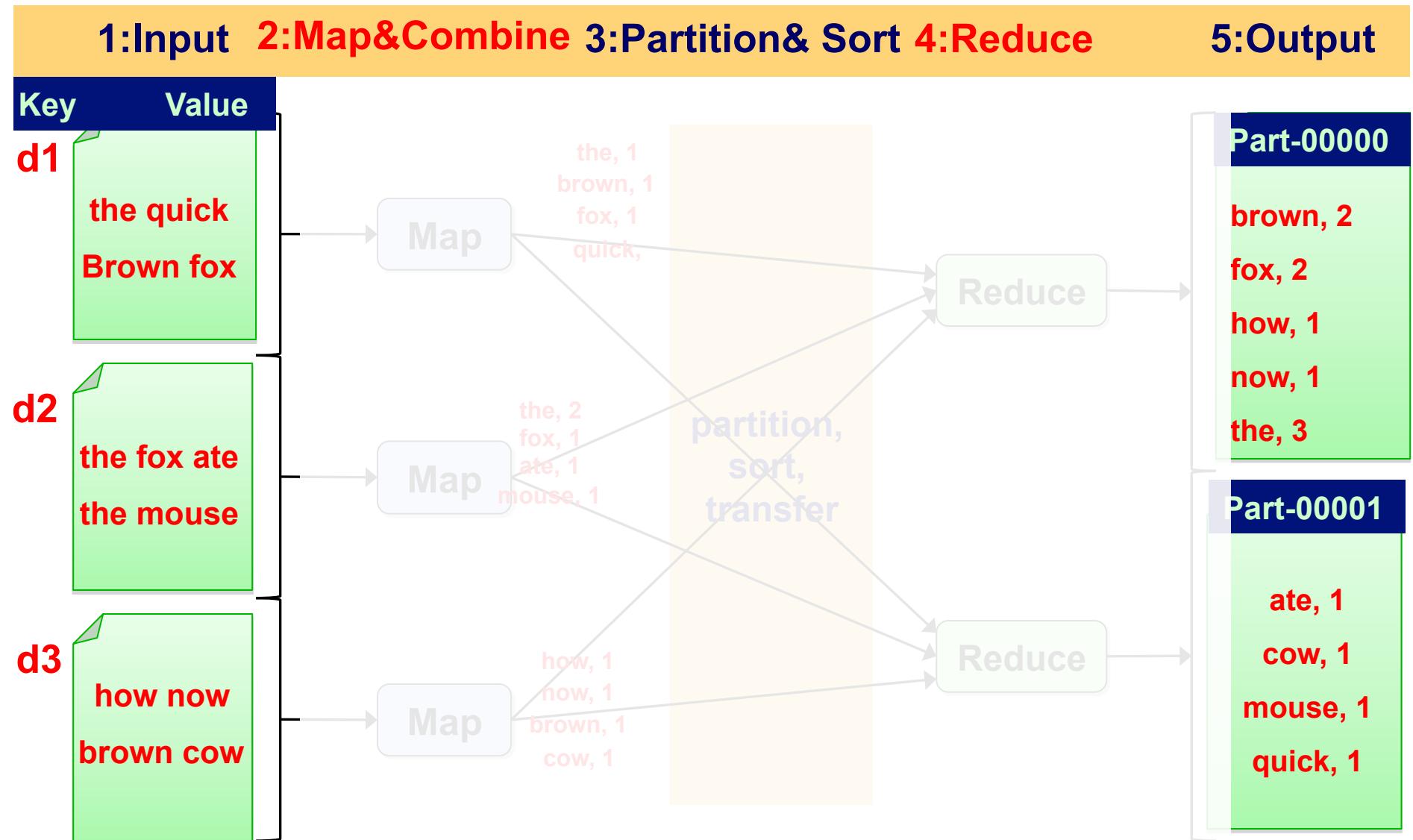
- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication/synchronization, types of PC tasks (10)
 - BLT: embarrassingly parallel problems (3)
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: multichoice Question (limitations of MPI?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
 - Hadoop 2.0 and what is Hadoop good at? (10)

V4

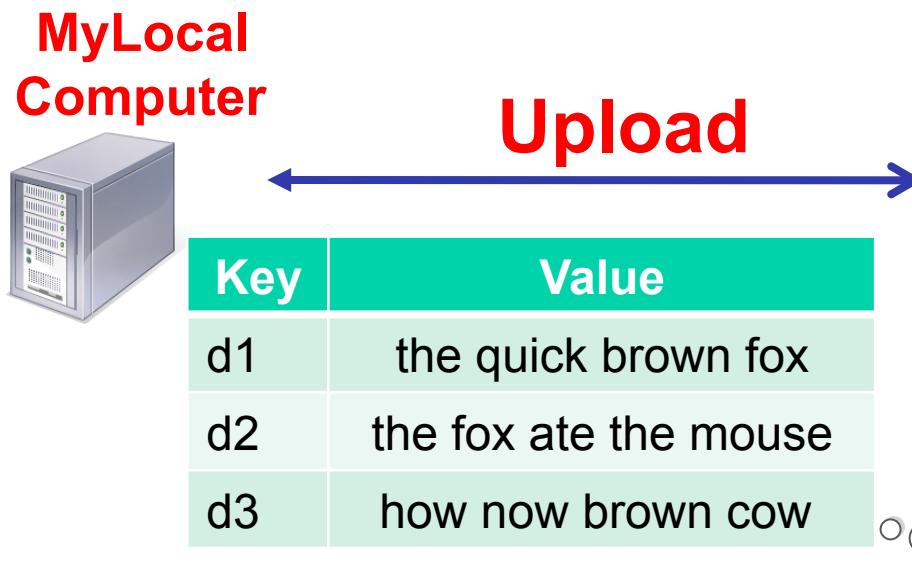
-
- Full example workflow of Hadoop

-
- In this we will get concrete and we will work thru a couple of examples of using Hadoop MapReduce to solve problems
 - First up is the word count example
 - Goal to count the number of occurrences of each word in a bunch of input files

Word Count Workflow



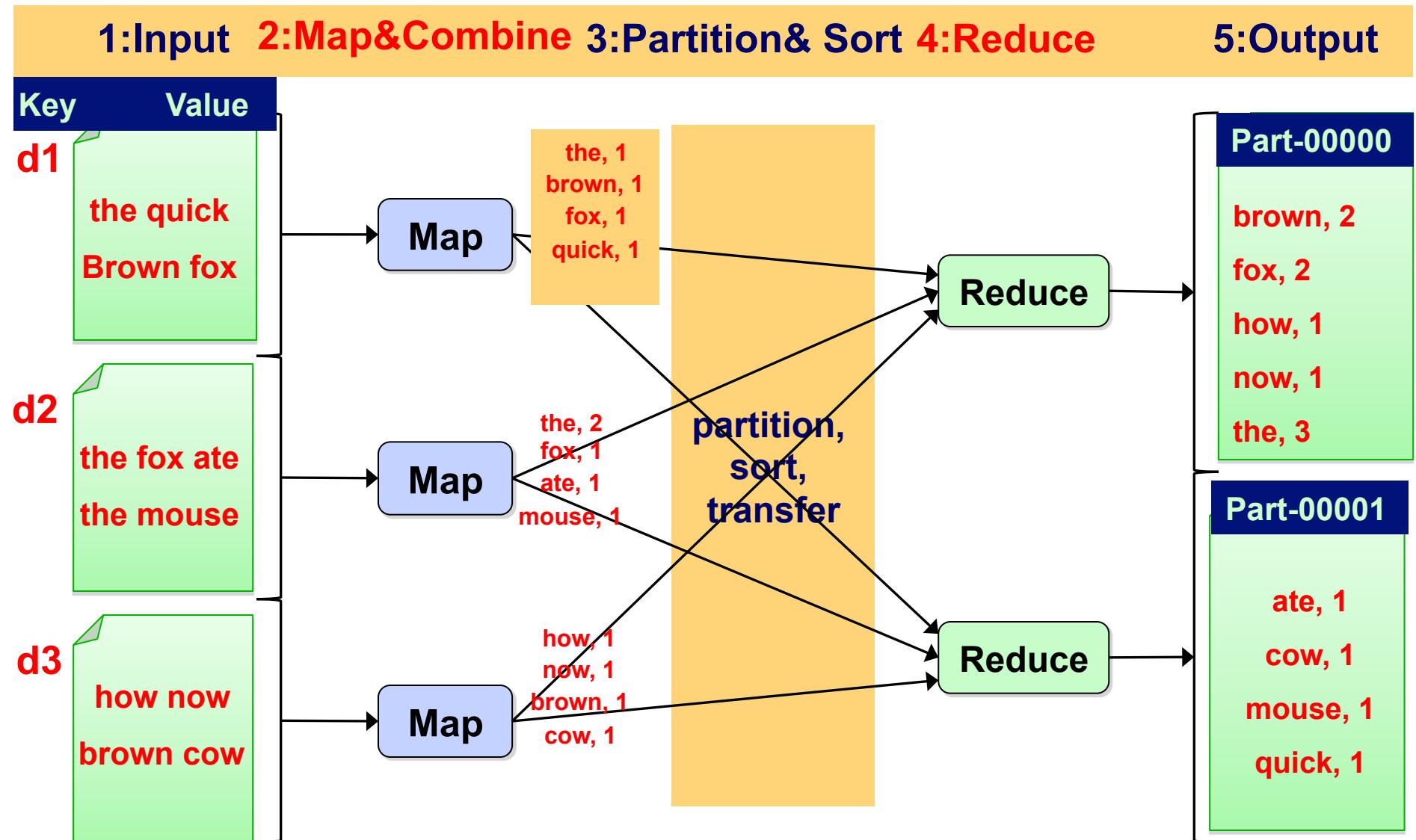
Hadoop Cluster: 1 Name; 3 data node+2 task nodes



```
> hadoop dfs -put f1.txt exampleDir
```

Assume block size is 20 Characters

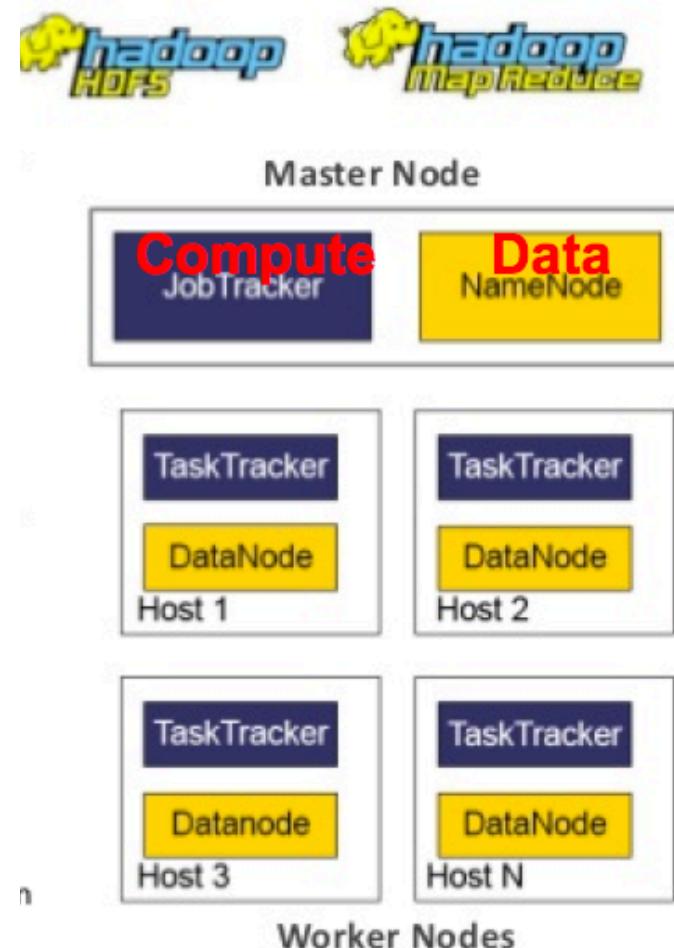
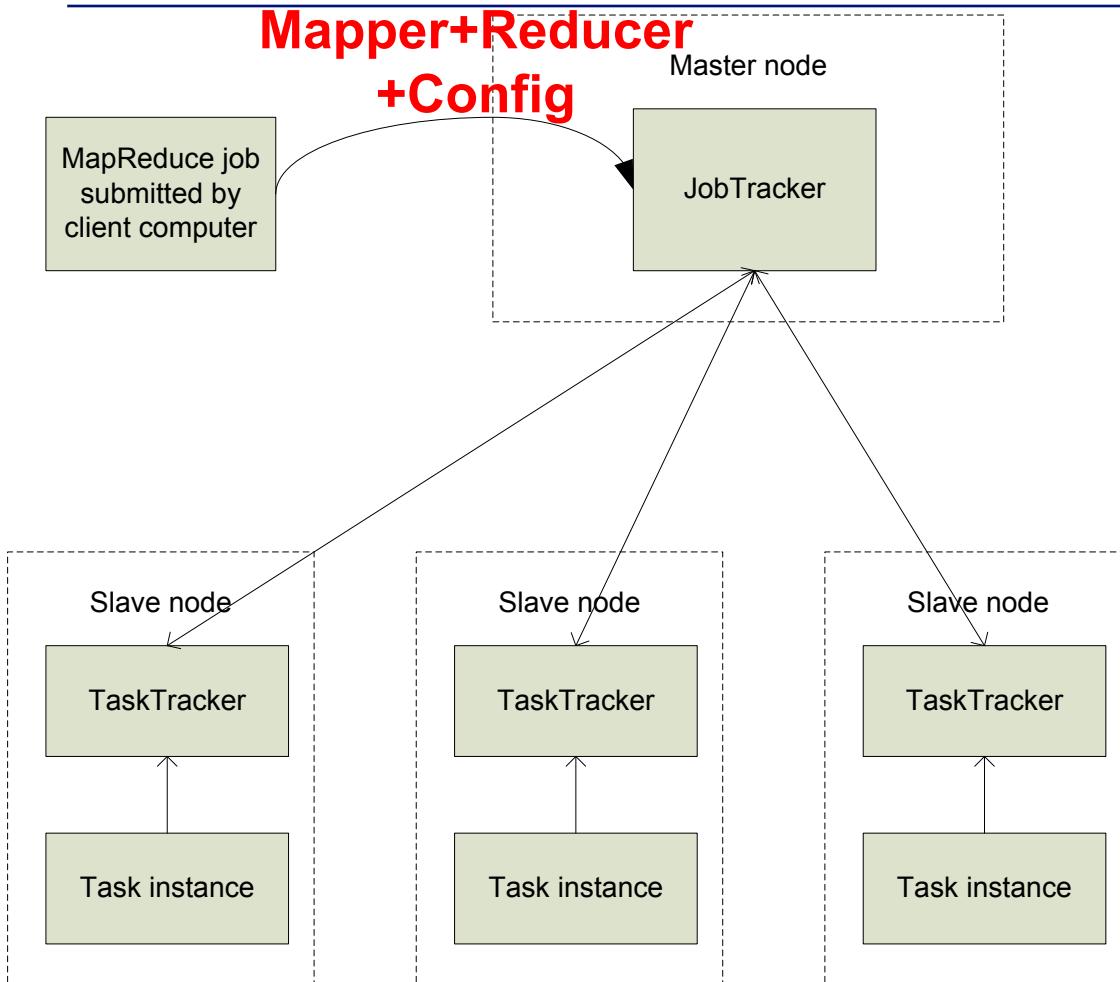
Word Count Workflow



Execution on Clusters

1. Input files split (M splits)
2. Assign Master & Workers
3. Map tasks
4. Writing intermediate data to disk (R regions)
5. Intermediate data read & sort
6. Reduce tasks
7. Return partition files

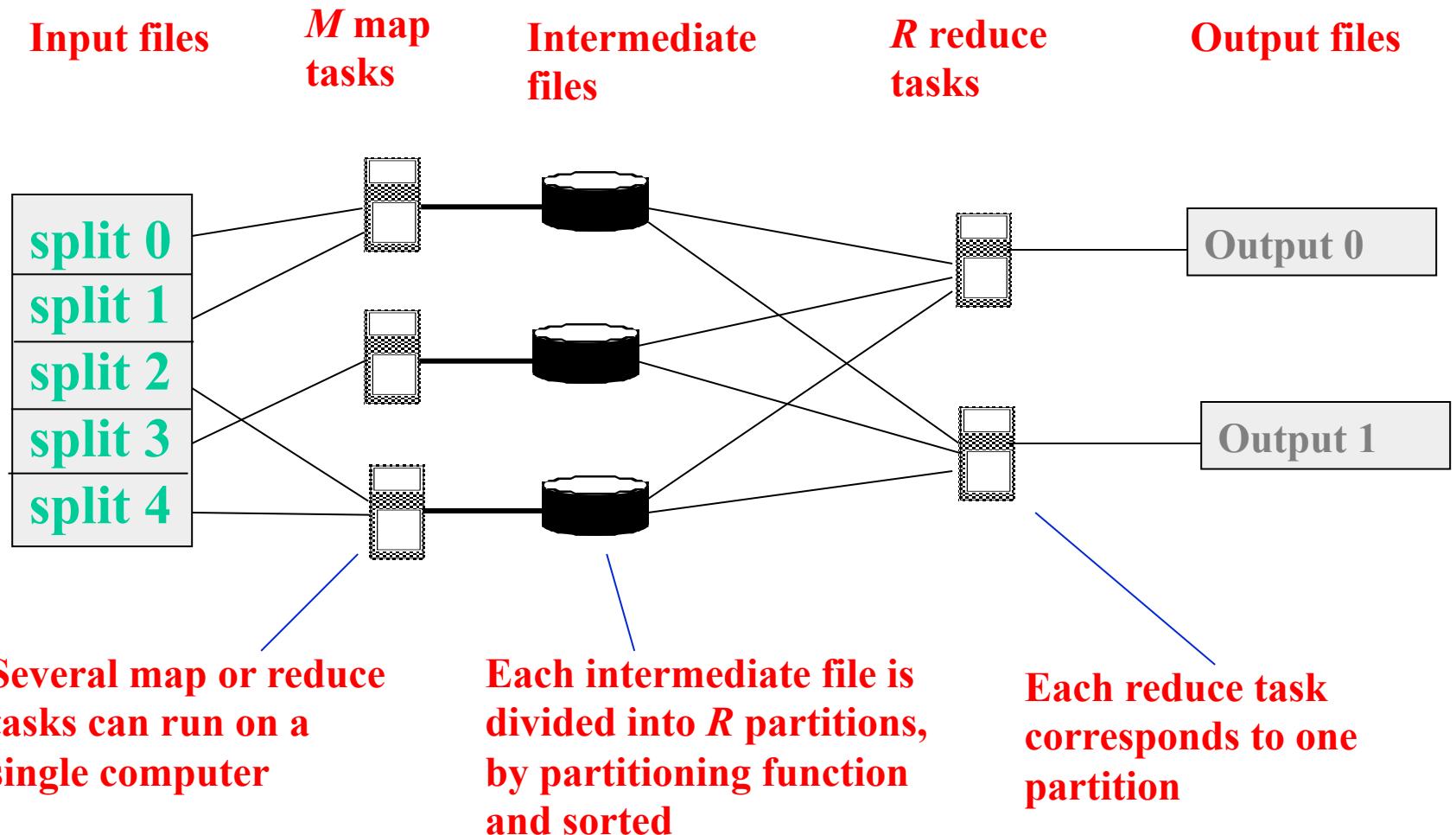
MapReduce: High Level



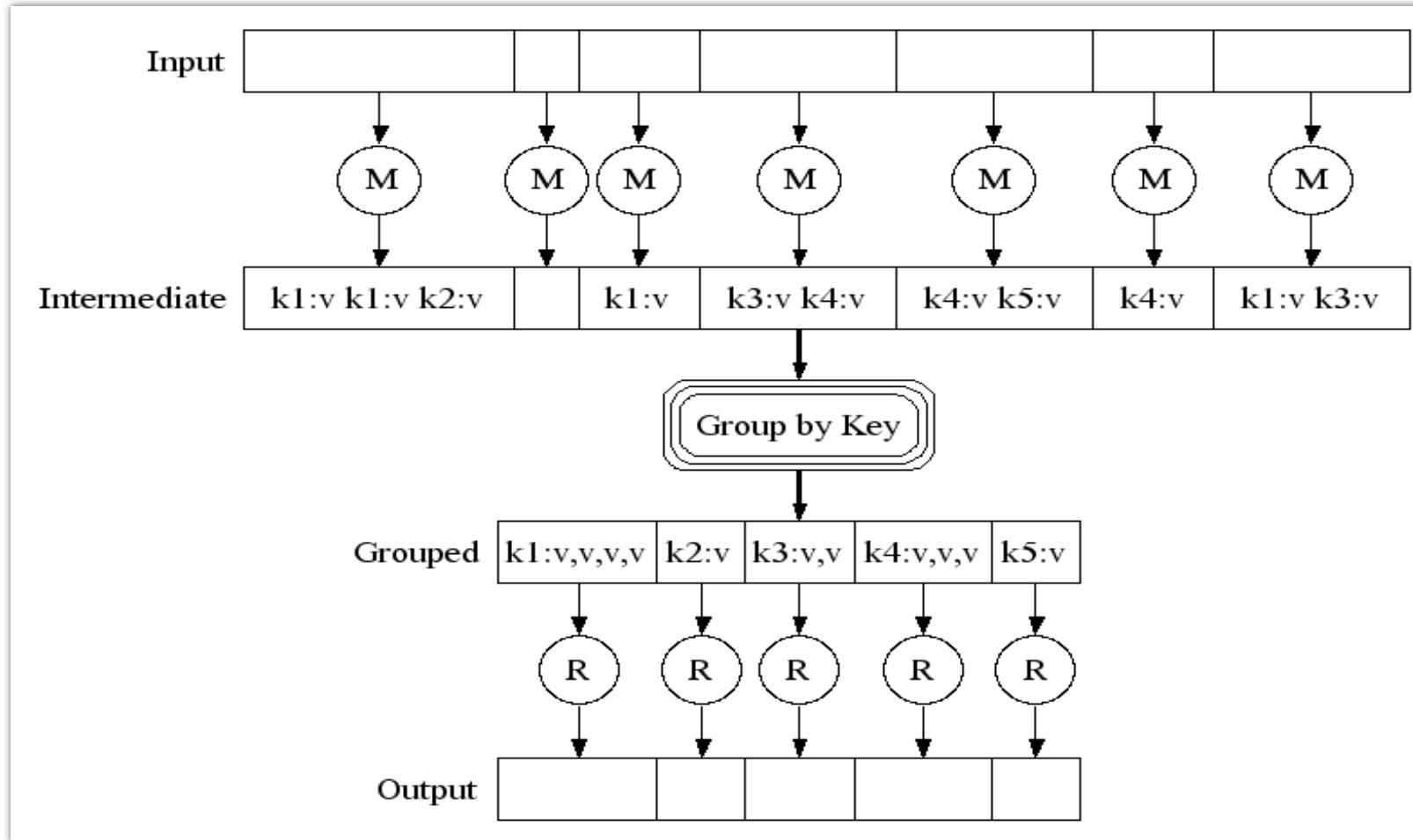
One instance of your Mapper is initialized by the *MapTaskRunner* for a *input block*

Exists in separate process from all other instances of Mapper – no data sharing!

Map/Reduce Cluster Implementation

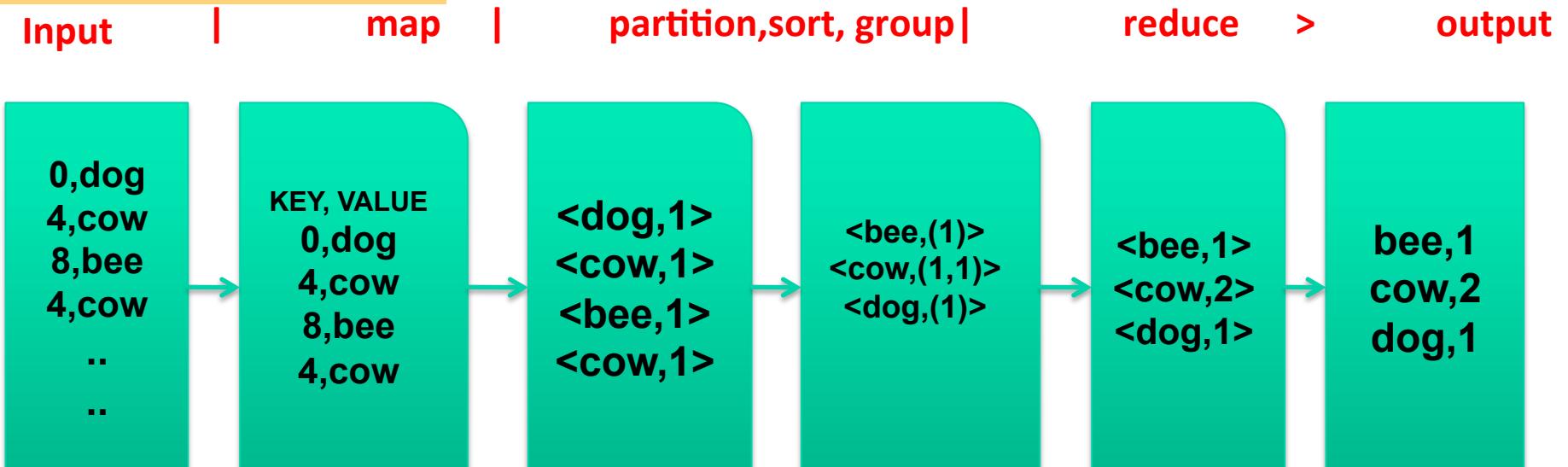


Execution



MapReduce:Ex2: WordCount from a word stream

In MapReduce terms



In unix terms

```
cat input | cut -f 2 -d, | sort | uniq -c > output
```

Example: Word Count Pseudo-code

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
```

Key	Value
d1	the quick brown fox
d2	the fox ate the mouse
d3	how now brown cow

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)
```

*partition,
sort,
transfer*

```
(‘brown’, 1)
(‘brown’, 1)
(‘brown’, (1, 1))
```

```
def reducer(key, values):
    output(key, sum(values))
```

```
(‘brown’, 2)
.....
```

Example: Word Count

```
public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

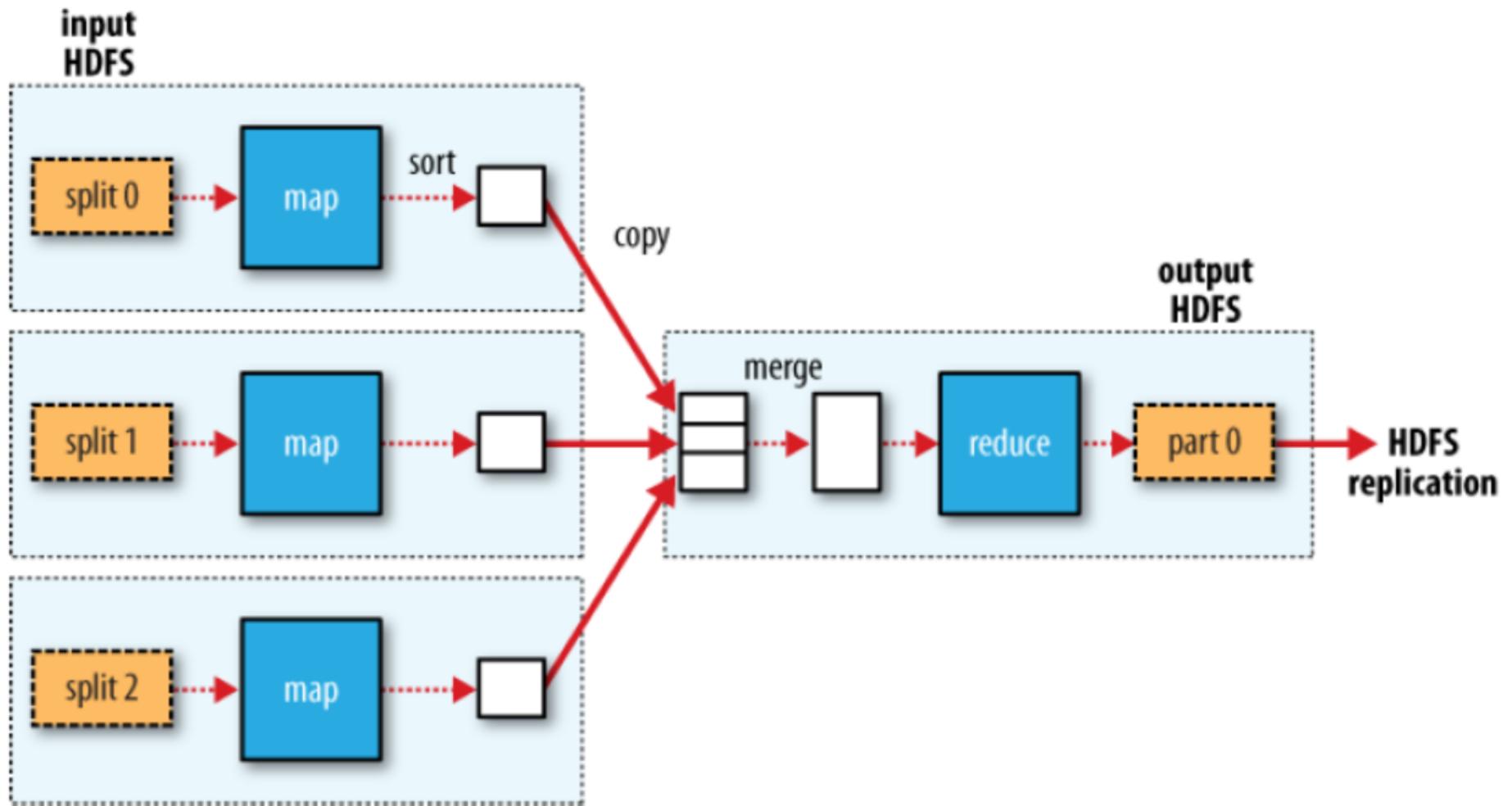
    private final static IntWritable ONE = new IntWritable(1);

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.nextToken()), ONE);
        }
    }
}

public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

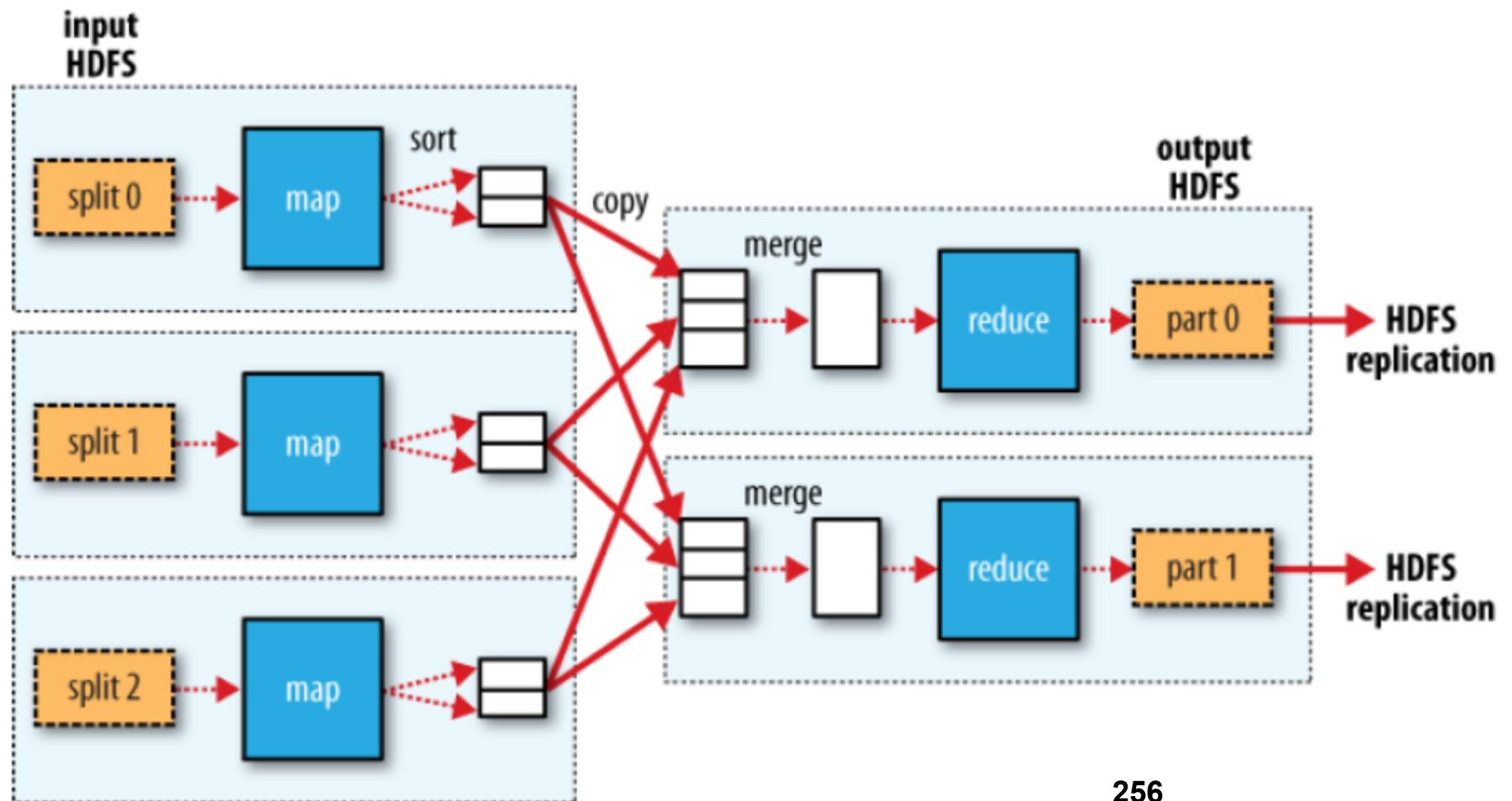
    public void reduce(Text key, Iterator<IntWritable> values,
                      OutputCollector<Text, IntWritable> output,
                      Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

MapReduce – Single reduce task



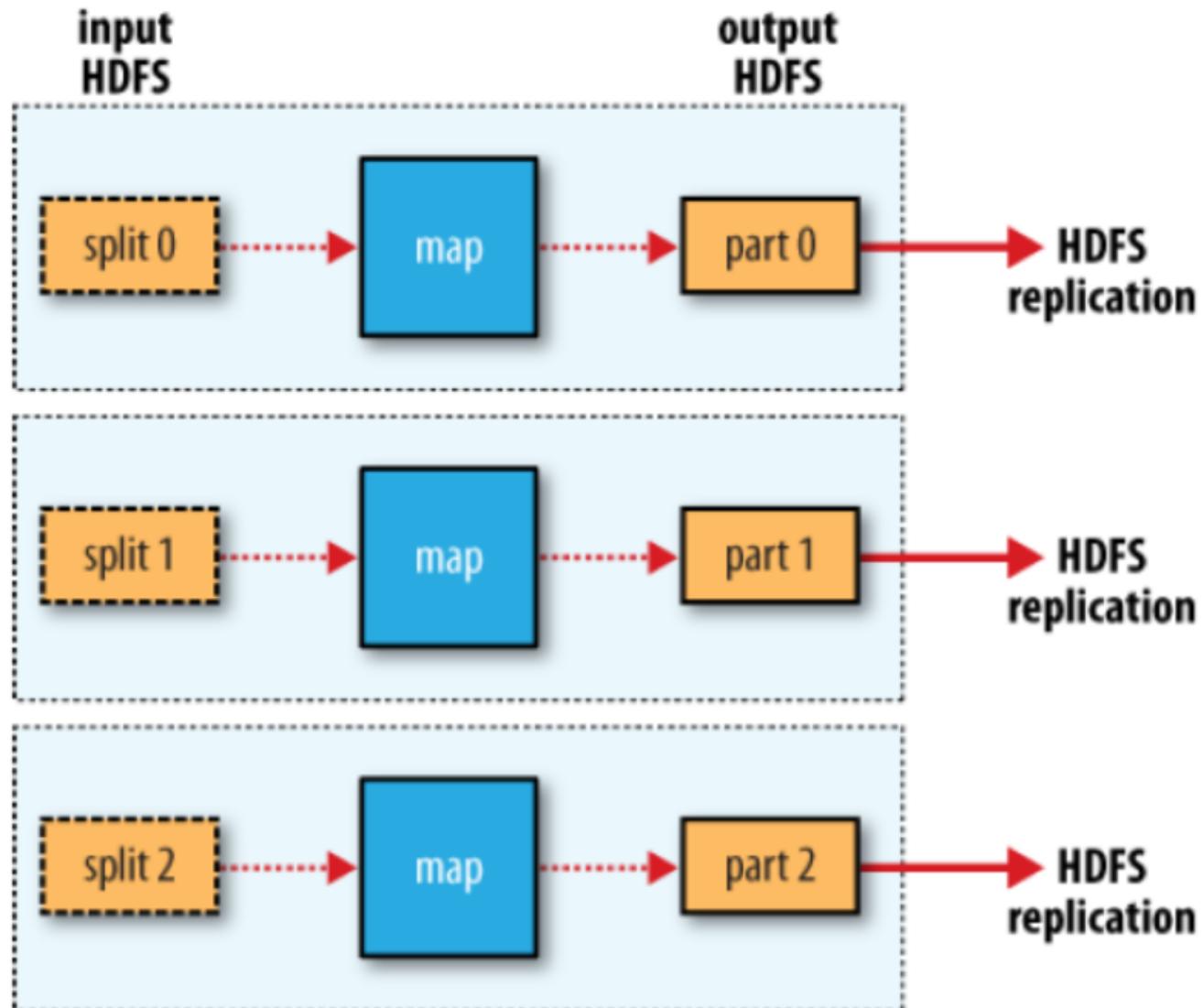
Tom White, *Hadoop: The Definitive Guide*

MapReduce – Multiple reduce tasks



256

MapReduce – No reduce tasks

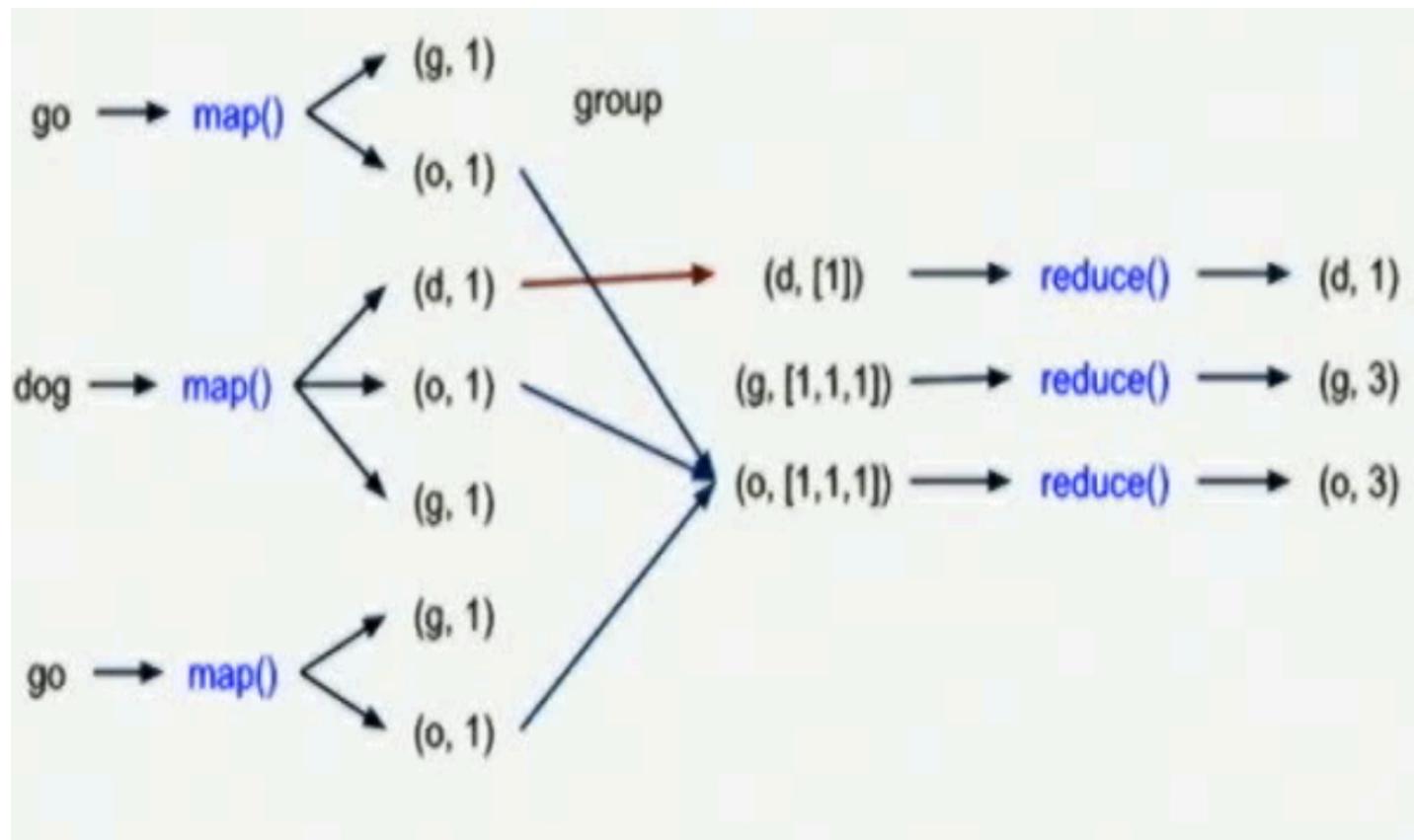


Contract with the programmer

- The frozen part of the MapReduce framework is a large distributed sort. The hot spots, which the application defines, are:
 - *an input reader*
 - **a Map function***
 - **a partition function**
 - **a compare function**
 - **a Reduce function***
 - *an output writer*
 - ***required from programmer (generally!)**

Character counter

- Write the map-reduce code to count the number times characters appear in the input data



Vector by vector multiplication: Dense

$$\begin{matrix} L & & B \\ & A & \\ (-3 & -1 & -1) & \begin{pmatrix} 0 \\ 5 \\ 3 \end{pmatrix} & L \end{matrix}$$

$$AB = \begin{pmatrix} -8 \end{pmatrix}$$

$$\begin{aligned} & (-3)(0) + (-1)(5) + (-1)(3) \\ & = \\ & 0 + -5 + -3 \\ & = \\ & -8 \end{aligned}$$

INPUT File
A, (-3, -1, -1)
B, (0, 5, 3)

MAP

Yield(0, -3)
Yield(1, -1)
Yield(2, -1)
Yield(0, 0)
Yield(1, 5)
Yield(2, 3)

SHUFFLE

(0, -3)
(0, 0)
(1, -1)
(1, 5)
(2, -1)
(2, 3)

REDUCE

$$\begin{aligned} & \overline{(-3)(0) + (-1)(5) + (-1)(3)} \\ & = \\ & 0 + -5 + -3 \\ & = \\ & -8 \end{aligned}$$

Vector by vector multiplication: Sparse

$$\begin{matrix} L \\ A \\ (-3 \quad -1 \quad -1) \end{matrix} \quad \begin{matrix} B \\ 0 \\ 5 \\ 3 \end{matrix} \quad \begin{matrix} L \\ \end{matrix}$$

$$AB = \begin{pmatrix} -8 \end{pmatrix}$$

$$\begin{aligned} & (-3)(0) + (-1)(5) + (-1)(3) \\ & \quad = \\ & \quad 0 \quad + \quad -5 \quad + \quad -3 \\ & \quad = \\ & \quad -8 \end{aligned}$$

INPUT File
A, (0 -3, 1 -1, 2 -1)
B, (1, 5, 2 3)

MAP

Yield(0, -3)
Yield(1, -1)
Yield(2, -1)
~~Yield(0, 0)~~
Yield(1, 5)
Yield(2, 3)

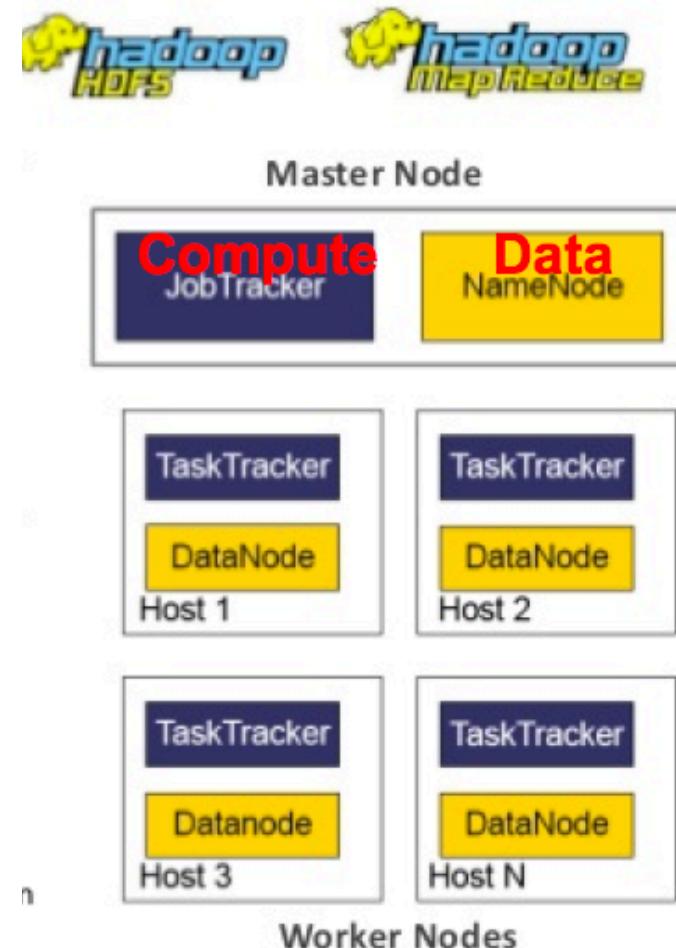
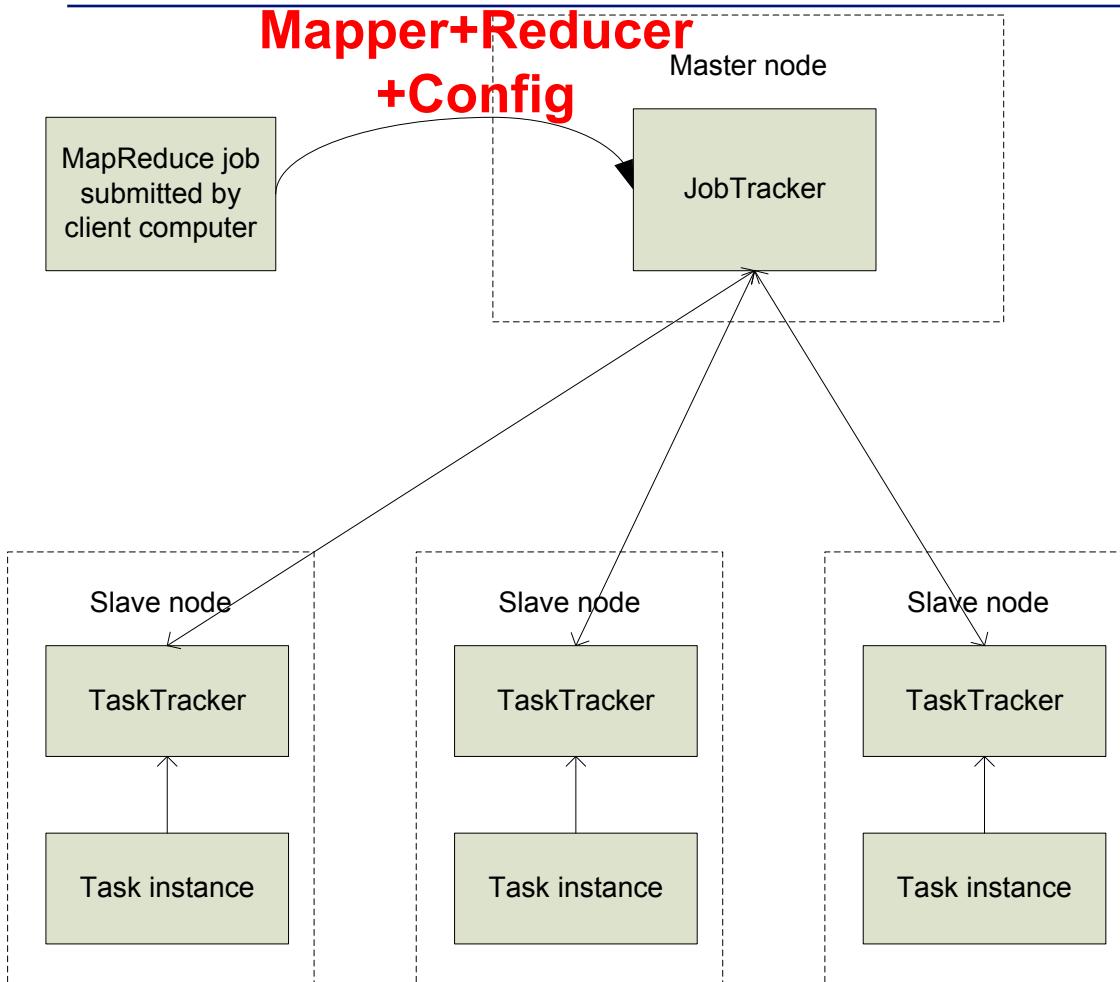
SHUFFLE

(0, -3)
~~(0, 0)~~
(1, -1)
(1, 5)
(2, -1)
(2, 3)

REDUCE

$$\begin{aligned} & \overline{(-3)(0) + (-1)(5) + (-1)(3)} \\ & \quad = \\ & \quad -5 \quad + \quad -3 \\ & \quad = \\ & \quad -8 \end{aligned}$$

MapReduce: High Level



One instance of your Mapper is initialized by the *MapTaskRunner* for a *input block*

Exists in separate process from all other instances of Mapper – no data sharing!

Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - BLT: Embarassingly parallel?) [2 minutes]
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: MPI limitations?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)

V4

-
- In practice, writing and deployingputting the rubber to the road

MapReduce Job Processing

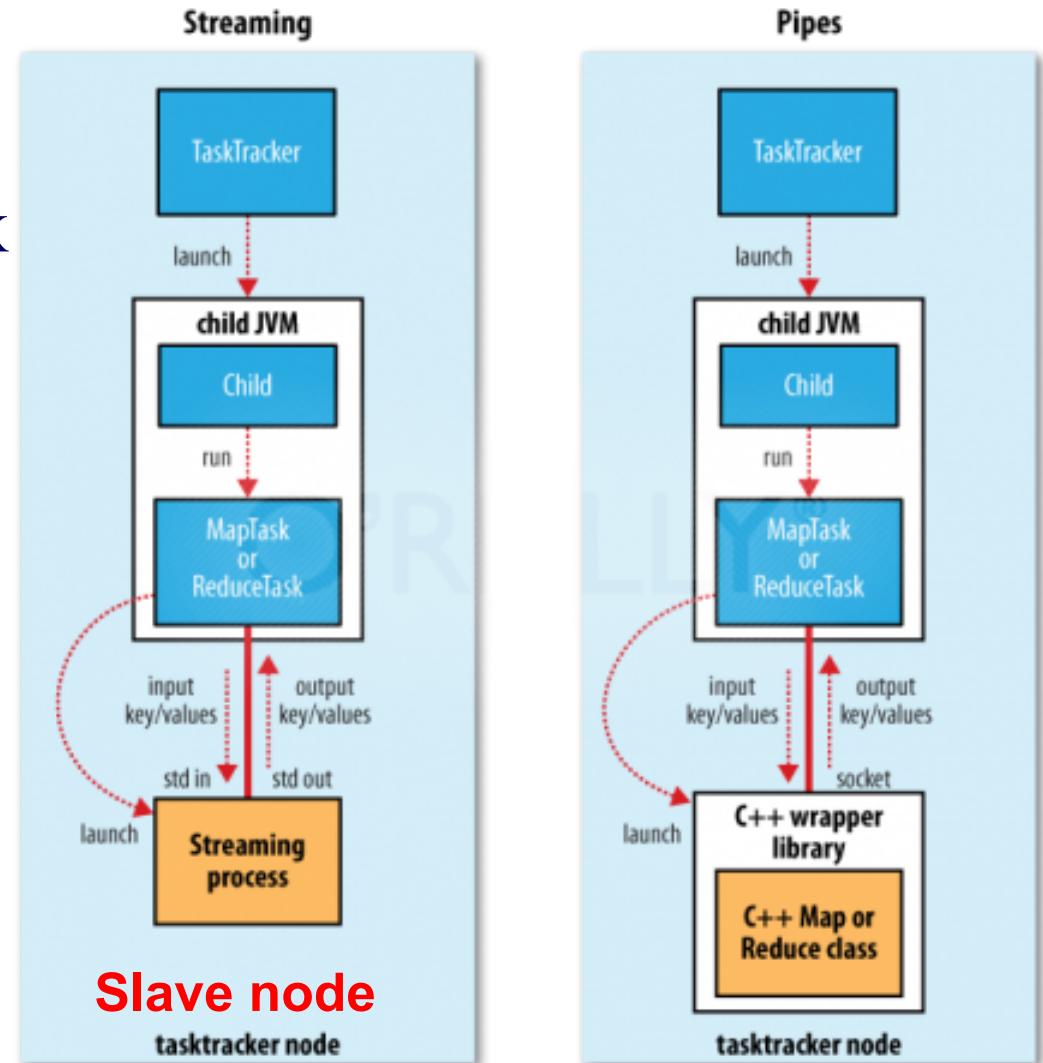
- An entire Hadoop execution of a client request is called a job. Users can submit job requests to the Hadoop framework, and the framework processes the jobs. Before the framework can process a job, the user must specify the following:
- The location of the input and output files in the distributed file system
- The input and output formats
- The classes containing the map and reduce functions
- Hadoop has four entities involved in the processing of a job:[1]
- The user, who submits the job and specifies the configuration

Hadoop: Native Java or Streaming

- Although the Hadoop framework is implemented in Java™, Map/Reduce applications need not be written in Java.
- Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. E.g., Python, Ruby, Perl, etc.
 - *Hadoop Streaming* uses Unix standard streams as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce program.
 - Hadoop Streaming is a utility which allows users to create and run jobs with any executables (e.g. shell utilities) as the mapper and/or the reducer.
- Hadoop Pipes is a SWIG- compatible C++ API to implement Map/Reduce applications (non JNI™ based).

Hadoop: Streaming or Pipes

- **Streaming and Pipe**
 - Not familiar with Java?
 - Python or C++ is also OK
 - Detail:
[http://hadoop.apache.org/
common/docs/r0.17.2/
streaming.html](http://hadoop.apache.org/common/docs/r0.17.2/streaming.html)



Word Count in Python with Hadoop Streaming

The prototypical MapReduce example counts the appearance of each word in a set of documents

1: Input

Doc1, the quick brown fox
Doc2, the fox ate the mouse
Doc3, how now brown cow

```
function map(String name, String document):  
    // name: document name  
    // document: document contents  
    for each word w in document:  
        emit (w, 1)  
  
function reduce(String word, Iterator partialCounts):  
    // word: a word  
    // partialCounts: a list of aggregated partial counts  
    sum = 0  
    for each pc in partialCounts:  
        sum += ParseInt(pc)  
    emit (word, sum)
```

2: Map

the, 1
quick, 1
brown, 1
fox, 1
the, 1
fox, 1.....

Hadoop Streaming in Python

- Hadoop streaming is a utility that comes with the Hadoop distribution.
- The utility allows you to create and run map/reduce jobs with any executable or script as the mapper and/or the reducer.
- For example:

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/  
hadoop-streaming.jar \
```

```
-input myInputDirs \
```

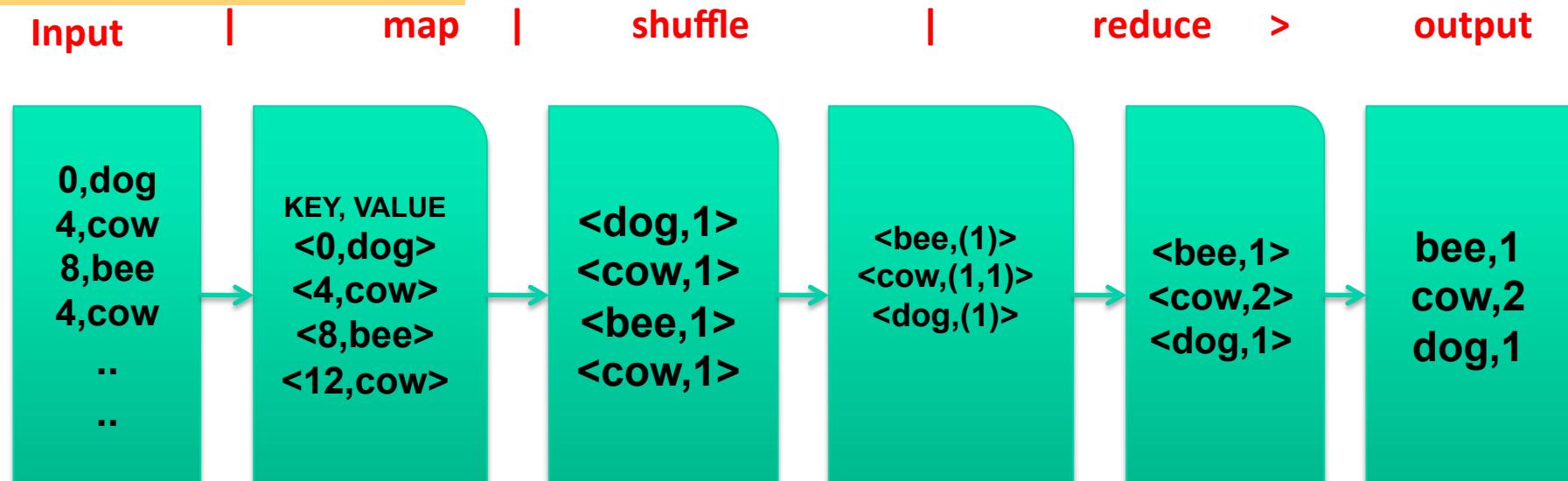
```
-output myOutputDir \
```

```
-mapper wordCountMapper.py \
```

```
-reducer wordCountReducer.py
```

MapReduce – WordCount of WordStream

In MapReduce terms



Unit Test

1: cat input.txt | mapper.py

2: cat input.txt | mapper.py | sort -k1,1

3: cat input.txt | mapper.py | sort -k1,1 | reducer.py

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar  
-input myInputDirs -output myOutputDir \  
-mapper wordCountMapper.py -reducer wordCountReducer.py
```

```
1 hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar -mappe
2 additionalConfSpec_:null
3 null=@@userJobConfProps_.get(stream.shipped.hadoopstreaming
4 packageJobJar: [/app/hadoop/tmp/hadoop-unjar54543/]
5 [] /tmp/streamjob54544.jar tmpDir=null
6 [...] INFO mapred.FileInputFormat: Total input paths to process : 7
7 [...] INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
8 [...] INFO streaming.StreamJob: Running job: job_200803031615_0021
9 [...]
10 [...] INFO streaming.StreamJob: map 0%  reduce 0%
11 [...] INFO streaming.StreamJob: map 43%  reduce 0%
12 [...] INFO streaming.StreamJob: map 86%  reduce 0%
13 [...] INFO streaming.StreamJob: map 100%  reduce 0%
14 [...] INFO streaming.StreamJob: map 100%  reduce 33%
15 [...] INFO streaming.StreamJob: map 100%  reduce 70%
16 [...] INFO streaming.StreamJob: map 100%  reduce 77%
17 [...] INFO streaming.StreamJob: map 100%  reduce 100%
18 [...] INFO streaming.StreamJob: Job complete: job_200803031615_0021
19 [...] INFO streaming.StreamJob: Output: /user/hduser/gutenberg-output
20 hduser@ubuntu:/usr/local/hadoop$
```

• ..

A screenshot of Hadoop's JobTracker web interface, showing the details of the MapReduce job we just ran

Hadoop job_200709211549_0003 on localhost

User: hadoop

Job Name: streamjob34453.jar

Job File: /usr/local/hadoop-dataset/hadoop-hadoop/mapred/system/job_200709211549_0003/job.xml

Status: Succeeded

Started at : Fri Sep 21 16:07:10 CEST 2007

Finished at: Fri Sep 21 16:07:26 CEST 2007

Finished in: 16sec

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	<u>Failed/Killed Task Attempts</u>
map	100.00%	3	0	0	3	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched map tasks	0	0	3
	Launched reduce tasks	0	0	1
	Data-local map tasks	0	0	3
Map-Reduce Framework	Map input records	77,637	0	77,637
	Map output records	103,909	0	103,909
	Map input bytes	3,659,910	0	3,659,910
	Map output bytes	1,083,767	0	1,083,767
	Reduce input groups	0	85,095	85,095
	Reduce input records	0	103,909	103,909
	Reduce output records	0	85,095	85,095

Check if the result is successfully stored in HDFS directory `/user/hduser/gutenberg-output`:

```
1 hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls /user/hduser/gutenberg-output
2 Found 1 items
3 /user/hduser/gutenberg-output/part-00000      <r 1> 903193 2007-09-21 13:00
4 hduser@ubuntu:/usr/local/hadoop$
```

You can then inspect the contents of the file with the `dfs -cat` command:

```
1 hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -cat /user/hduser/gutenberg-output/part-00000
2 "(Lo)cra" 1
3 "1490" 1
4 "1498," 1
5 "35" 1
6 "40," 1
7 "A" 2
8 "AS-IS". 2
9 "A_" 1
10 "Absoluti" 1
11 [...]
12 hduser@ubuntu:/usr/local/hadoop$
```

-
- **Summary of class**

Parallel computing, MapReduce, Hadoop

- • Motivation for Parallel Computing (5)
- • Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - Architectures for Parallel Computation (10)
 - Developer frameworks for Parallel Computation (10) (35, 55)
- • Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- • Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- • MapReduce: Runtime Environment (5)
- • Hadoop 2.0 and what is Hadoop good at? (10)
- • Sync time
 - Install Hadoop; run locally; run in the cloud?
 - 10 most popular words

V4

-
- Happy hadooping!!!

-
- End of lecture
 - 6/1/2015

Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - BLT: Embarassingly parallel?) [2 minutes]
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: MPI limitations?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]
- MapReduce: Runtime Environment (5)
- Hadoop 2.0 and what is Hadoop good at? (10)

V4

Full Example

- **SCREEN Flow**
- **Full code Examples (20)**
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]

<http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>

Michael G. Noll

Applied Research. Big Data. Distributed Systems. Open Source.

Blog Archive Tutorials Projects Publications

Writing an Hadoop MapReduce Program in Python

In this tutorial I will describe how to write a simple [MapReduce](#) program for [Hadoop](#) in the [Python](#) programming language.

Motivation

Even though the Hadoop framework is written in Java, programs for Hadoop need not to be coded in Java but can also be developed in other languages like Python or C++ (the latter since version 0.14.1). However, [Hadoop's documentation](#) and the most prominent

Table of Contents

- [Motivation](#)
- [What we want to do](#)
- [Prerequisites](#)
- [Python MapReduce Code](#)
 - [Map step: mapper.py](#)
 - [Reduce step: reducer.py](#)
 - [Test your code \(cat data | map | sort | reduce\)](#)
- [Running the Python Code on Hadoop](#)
 - [Download example input data](#)
 - [Copy local example data to HDFS](#)
 - [Run the MapReduce job](#)

Parallel computing, MapReduce, Hadoop

- Motivation for Parallel Computing (5)
- Parallel computing (PC) (30)
 - Definition, Communication&syncrhonization, types of PC tasks (10)
 - BLT: Embarassingly parallel?) [2 minutes]
 - Architectures for Parallel Computation (10)
 - BLT: multichoice Question (Shared nothing?) [2 minutes]
 - Developer frameworks for Parallel Computation (10) (35, 55)
 - BLT: MPI limitations?) [2 minutes]
- Hadoop (40)
 - Background and history (5)
 - Hadoop File System (HDFS) (10)
 - MapReduce
 - Functional programming (5)
 - MapReduce 5
 - Animated Examples 10
 - Hadoop in practice 5
- Full code Examples (20) [optional]
 - WordCount example on local machine (10) [ScreenFlow]
 - WordCount example on cluster(10) [ScreenFlow 10]

V4

Large

- MapReduce: Runtime Environment (5)

281

- Hadoop 2.0 and what is Hadoop good at? (10)

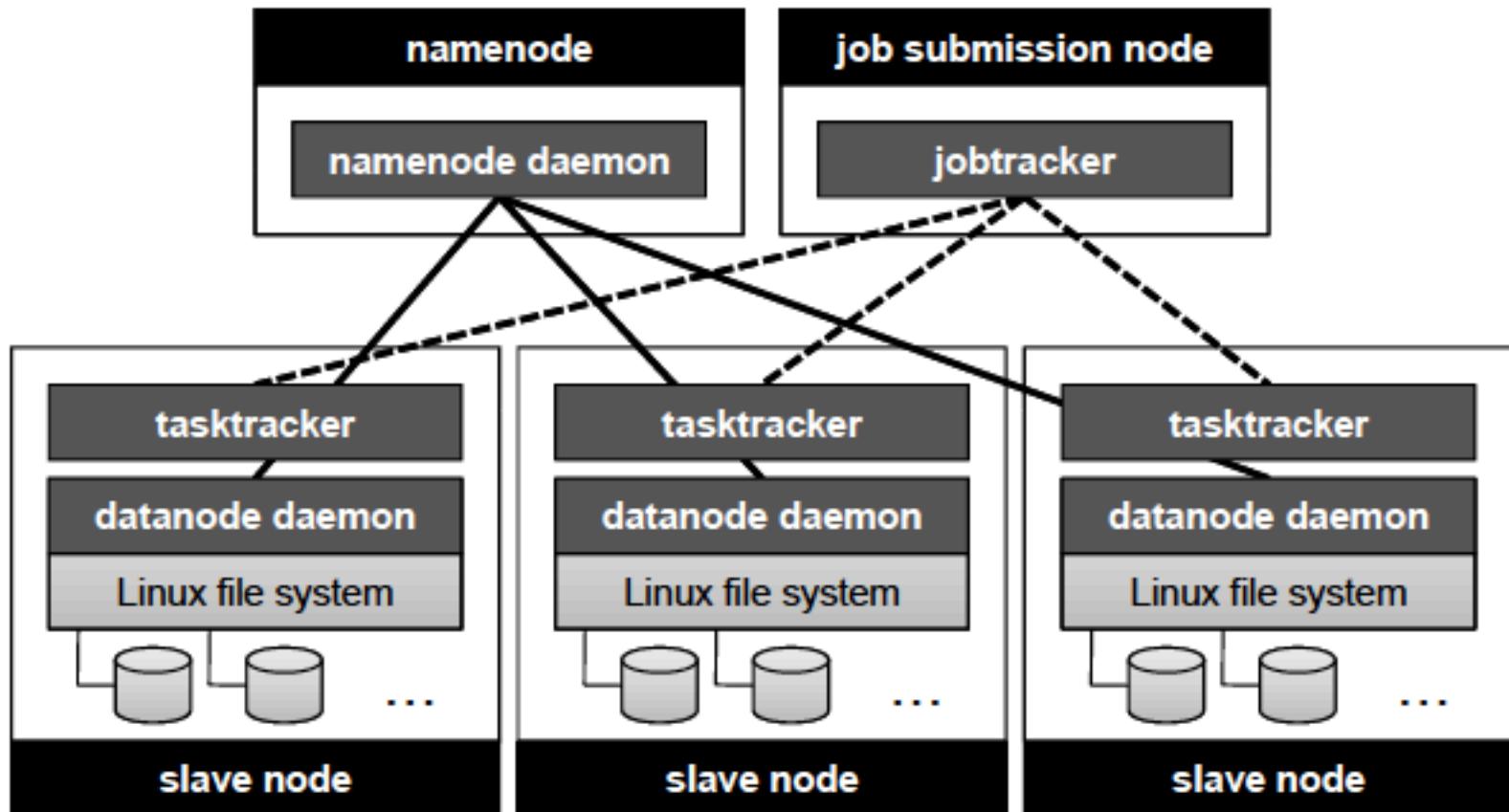
-
- MapReduce: Runtime Environment (optional)

- One of the most important idea behind MapReduce is separating the what of distributed processing from the how.
- A MapReduce program, referred to as a job, consists of code for mappers and reducers (as well as combiners and partitioners to be discussed in the next section) packaged together with configuration parameters (such as where the input lies and where the output should be stored).
- The developer submits the job to the submission node of a cluster (in Hadoop, this is called the jobtracker) and execution framework (sometimes called the "runtime") takes care of everything else: it transparently handles all other aspects of distributed code execution, on clusters ranging from a single node to a few thousand nodes.

MapReduce: Runtime Environment

- **Scheduling: task management; queues;**
 - Another aspect of scheduling involves coordination among tasks belonging to different jobs (e.g., from different users)
 - Manage stragglers, or tasks that take an usually long time to complete
- **Data/code colocation**
- **Synchronization**
 - In general, synchronization refers to the mechanisms by which multiple concurrently running processes join up, for example, to share intermediate results or otherwise exchange state information. In MapReduce, synchronization is accomplished by a barrier between the map and reduce phases of processing. [partition, sort]
- **Error and fault handling (Hardware and software)**
 - The MapReduce execution framework must accomplish all the tasks above in an environment where errors and faults are the norm, not the exception

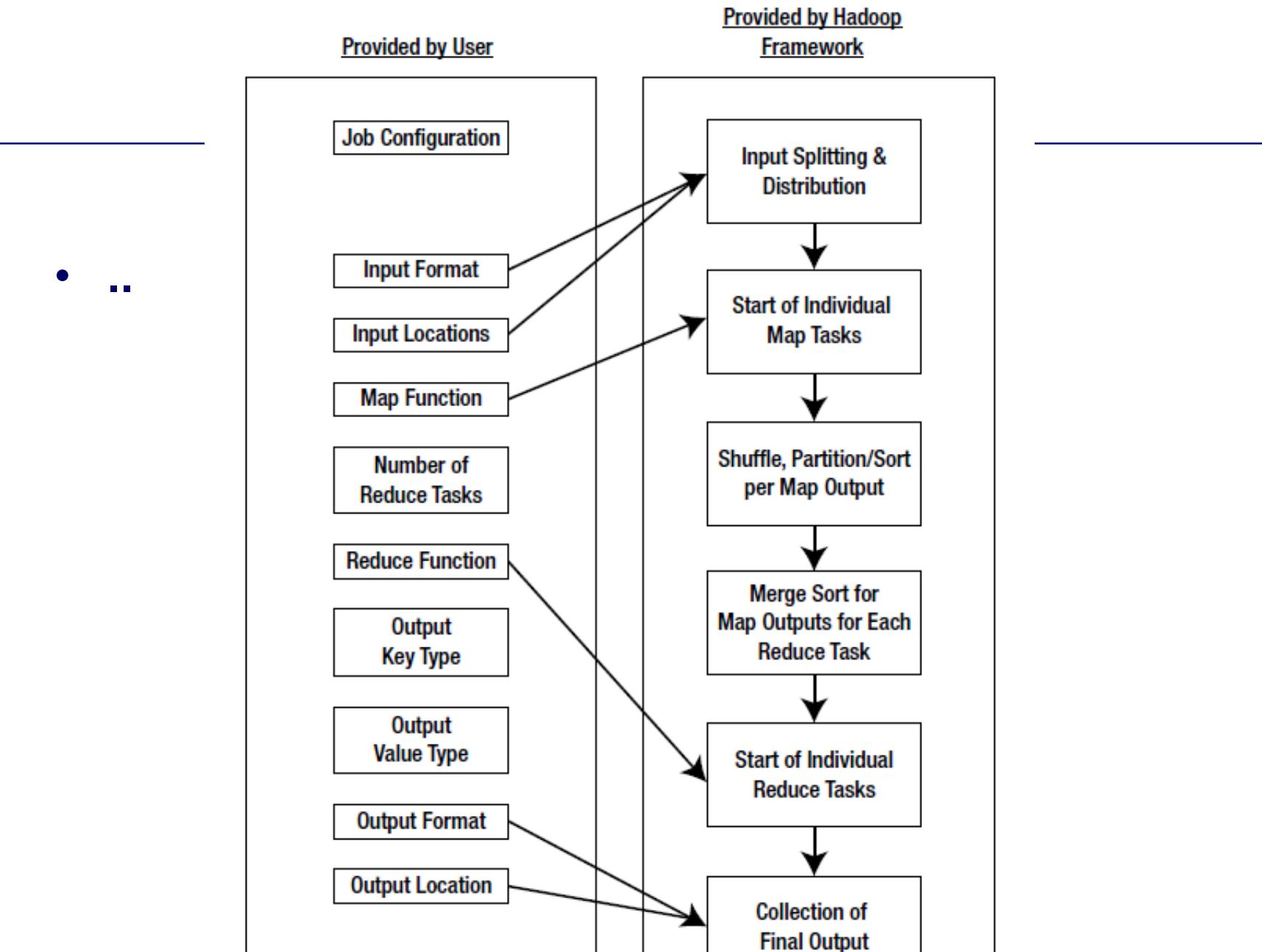
HADOOP CLUSTER ARCHITECTURE



Putting everything together, the architecture of a complete Hadoop cluster is shown here. The HDFS namenode runs the namenode daemon.

The job submission node runs the jobtracker, which is the single point of contact for a client wishing to execute a MapReduce job.

The jobtracker monitors the progress of running MapReduce jobs and is responsible for coordinating the execution of the mappers and reducers



[Pro Hadoop by Jason Venner [page 28], on
<http://hadooptutorial.wikispaces.com/Hadoop+architecture>]

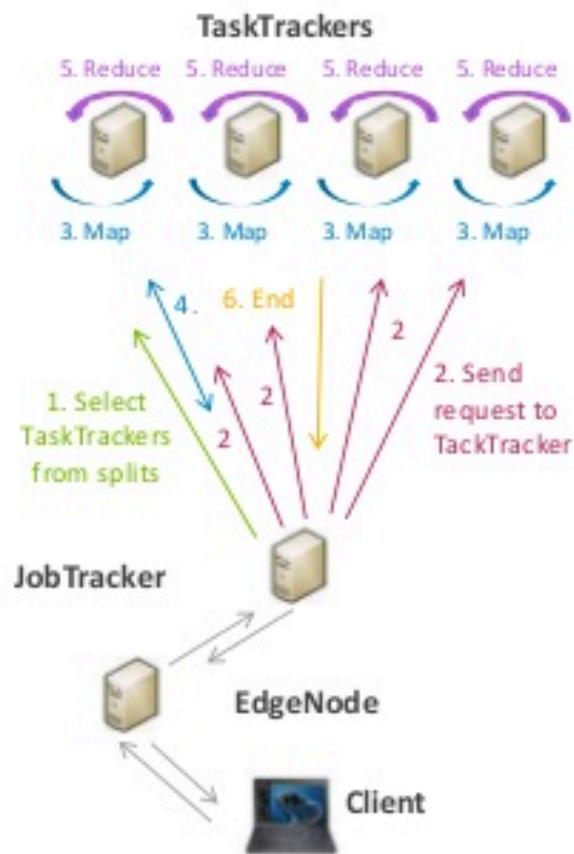
@ gmail.com

286

MAPREDUCE - EXEC FILE

The client program is copied on each node

1. The JobTracker determines the number of splits from the input path, and select some TaskTrackers based on their network proximity to the data sources
2. JobTracker sends the task requests to those selected TaskTrackers
3. Each TaskTracker starts the map phase processing by extracting the input data from the splits
4. When the map task completes, the TaskTracker notifies the JobTracker. When all the TaskTrackers are done, the JobTracker notifies the selected TaskTrackers for the reduce phase
5. Each TaskTracker reads the region files remotely and invokes the reduce function, which collects the key/aggregated value into the output file (one per reducer node)
6. After both phase completes, the JobTracker unblocks the client program

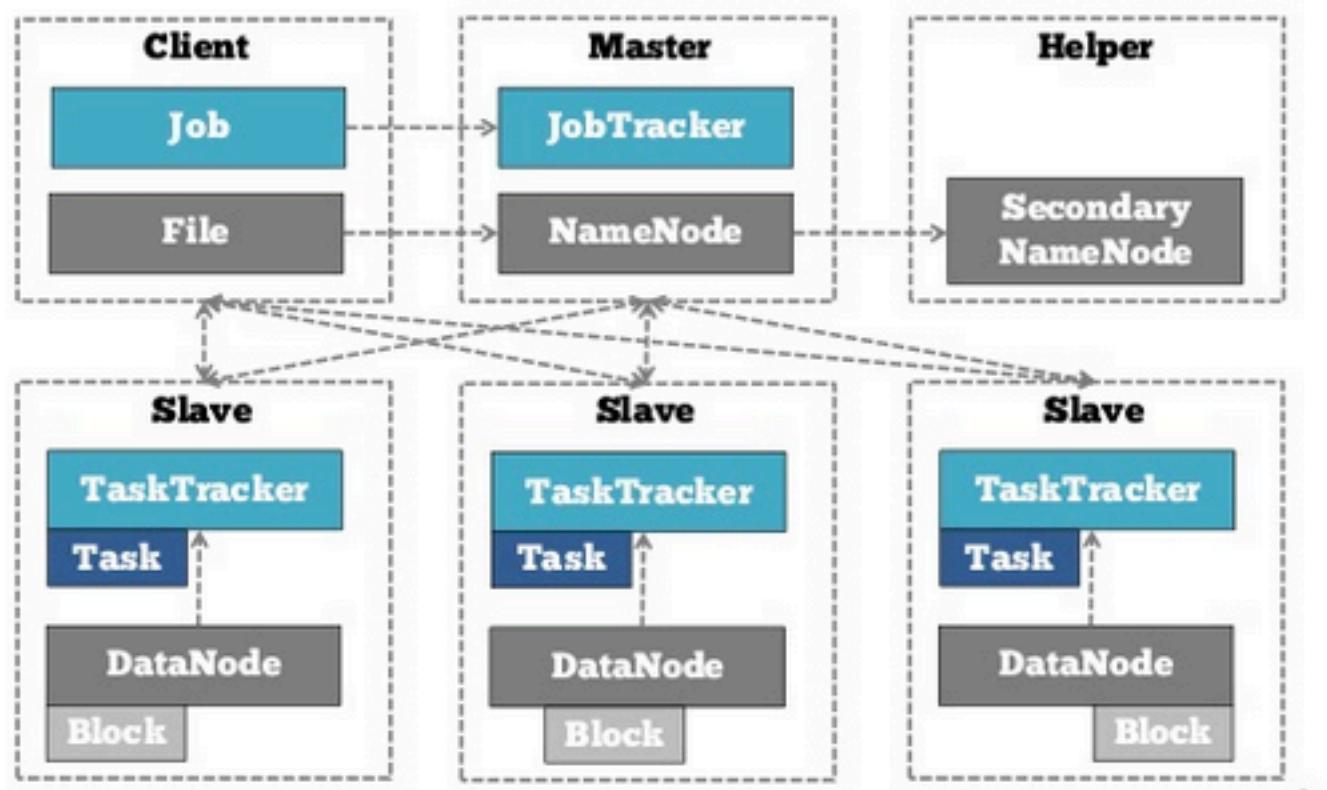


Hadoop Summary

- **Summary (next slides)**
- **Q: What is Hadoop good at?**

Hadoop, Why?

- **Need to process Multi Petabyte (1000 Harddrives) Datasets**
- **Expensive to build reliability in each application.**
- **Nodes fail every day**
 - Failure is expected, rather than exceptional.
 - The number of nodes in a cluster is not constant.
- **Need common infrastructure**
 - Efficient, reliable, Open Source Apache License
- **The above goals are same as Condor, but**
 - Workloads are IO bound and not CPU bound
 - Condor supports the standard [MPI](#) and [PVM](#)



codecentric 



- **Embarrassingly parallel algorithms**
- **Summing, grouping, filtering, joining**
- **Off-line batch jobs on massive data sets**
- **Analyzing an entire large dataset**



- **Iterative jobs (i.e., graph algorithms)**
 - **Each iteration must read/write data to disk**
 - **I/O and latency cost of an iteration is high**





MapReduce is not good for...

- **Jobs that need shared state/coordination**
 - Tasks are shared-nothing
 - Shared-state requires scalable state store
- **Low-latency jobs**
- **Jobs on small datasets**
- **Finding individual records**



-

- **Scalability**

- Maximum cluster size ~ 4,500 nodes
 - Maximum concurrent tasks ~ 40,000
 - Coarse synchronization in JobTracker

+Yarn/Mesos for task mgt

- **Availability**

- Failure kills all queued and running jobs

- **Hard partition of resources into map & reduce slots**

- Low resource utilization

- **Lacks support for alternate paradigms and services**

- Iterative applications implemented using MapReduce are 10x slower

Parallel programming

- **What if you need more synchronization?**
 - Future results depend on past

Big Ideas behind Map-Reduce

- **Scale out not up**
- **Assume failures (MTBF meantime between failures is 1000 days)**
- **Move processing to the data**
- **Process data sequential (not random access)**
- **Hide filesystem details from application developer**
- **Seamless scalability**

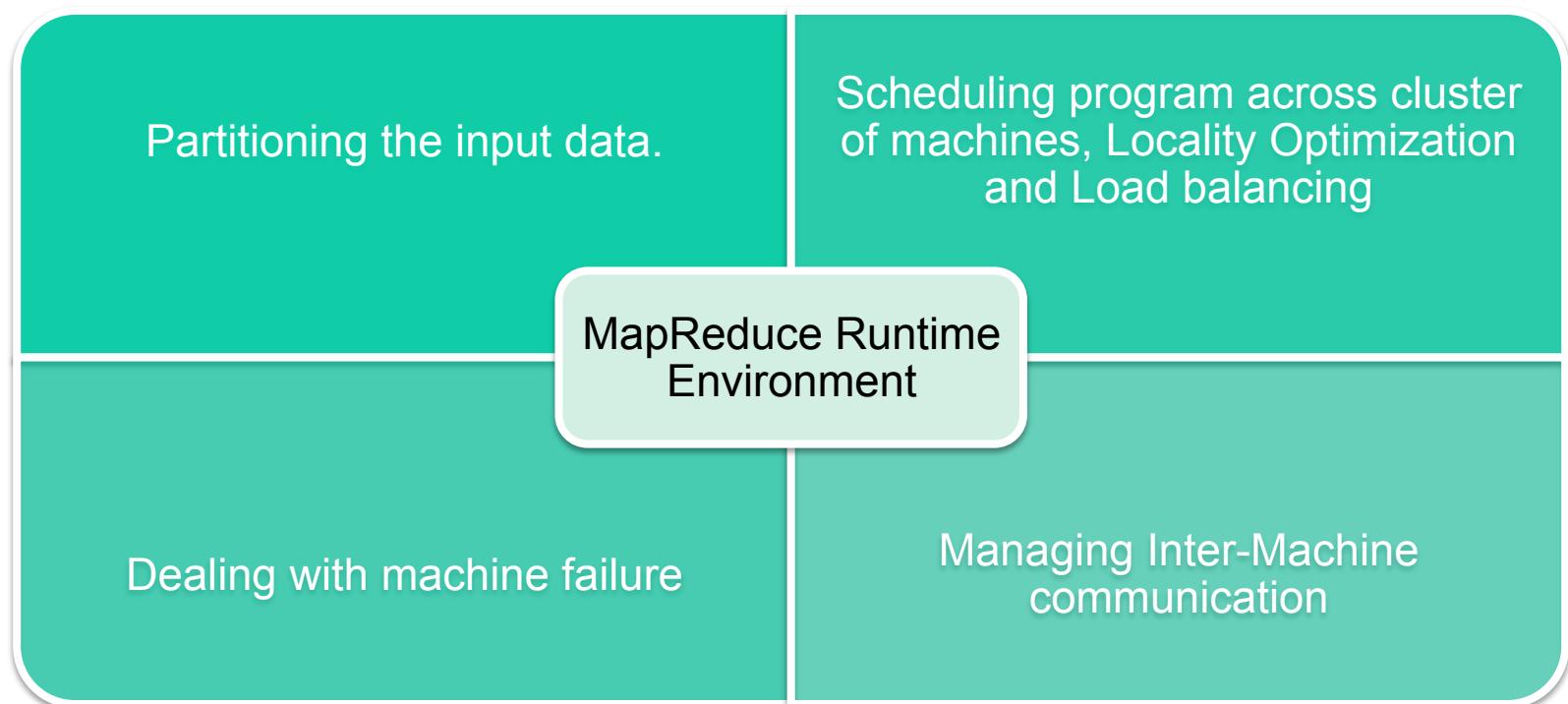
Hadoop Map/Reduce features

- **Java, C++, and text-based APIs**
 - In Java use Objects and and C++ bytes
 - Text-based (streaming) great for scripting or legacy apps (Python/Perl)
 - Higher level interfaces: Pig, Hive, Jaql
- **Automatic re-execution on failure**
 - In a large cluster, some nodes are always slow or flaky
 - Framework re-executes failed tasks
- **Locality optimizations**
 - With large data, bandwidth to data is a problem
 - Map-Reduce queries HDFS for locations of input data
 - Map tasks are scheduled close to the inputs when possible

MapReduce Environment

- **MapReduce environment takes care of:**
 - Partitioning the input data
 - Scheduling the program's execution across a set of machines
 - Perform the “group-by key” step in the Sort and Shuffle phase
 - Handling machine failures
 - Managing required inter-machine communication

MapReduce: Runtime Environment



MapReduce: Fault Tolerance

- Handled via re-execution of tasks.
- Task completion committed through master
- **What happens if Mapper fails ?**
 - Re-execute completed + in-progress *map* tasks
- **What happens if Reducer fails ?**
 - Re-execute in progress *reduce* tasks
- **What happens if Master fails ?**
 - Potential trouble !!

Fault Recovery

- **Workers are pinged by master periodically**
 - Non-responsive workers are marked as failed
 - All tasks in-progress or completed by failed worker become eligible for rescheduling
- **Master could periodically checkpoint**
 - Current implementations abort on master failure

Fault Tolerance in MapReduce

1. If a task crashes:

- Retry on another node
 - OK for a map because it has no dependencies
 - OK for reduce because map outputs are on disk
- If the same task fails repeatedly, fail the job or ignore that input block (user-controlled)

➤ **Note: For these fault tolerance features to work, your map and reduce tasks must be side-effect-free**

Fault Tolerance in MapReduce

2. If a node crashes:

- Re-launch its current tasks on other nodes
- Re-run any maps the node previously ran
 - Necessary because their output files were lost along with the crashed node

- End of lecture