DS256:Jan17 (3:1)

L2,3:Programming for Large Datasets MapReduce

Yogesh Simmhan

10/12 Jan, 2017

©Department of Computational and Data Science, IISc, 2016
This work is licensed under a Creative Commons Attribution 4.0 International License
Copyright for external content used with attribution is retained by their original authors

Recap of L1

What is Big Data? The term ***is*** fuzzy … *Handle with care!*

Wordle of "Thought Leaders'" definition of Big Data, © Jennifer Dutcher, 2014
https://datascience.berkeley.edu/what-is-big-data/

So…What is Big Data?

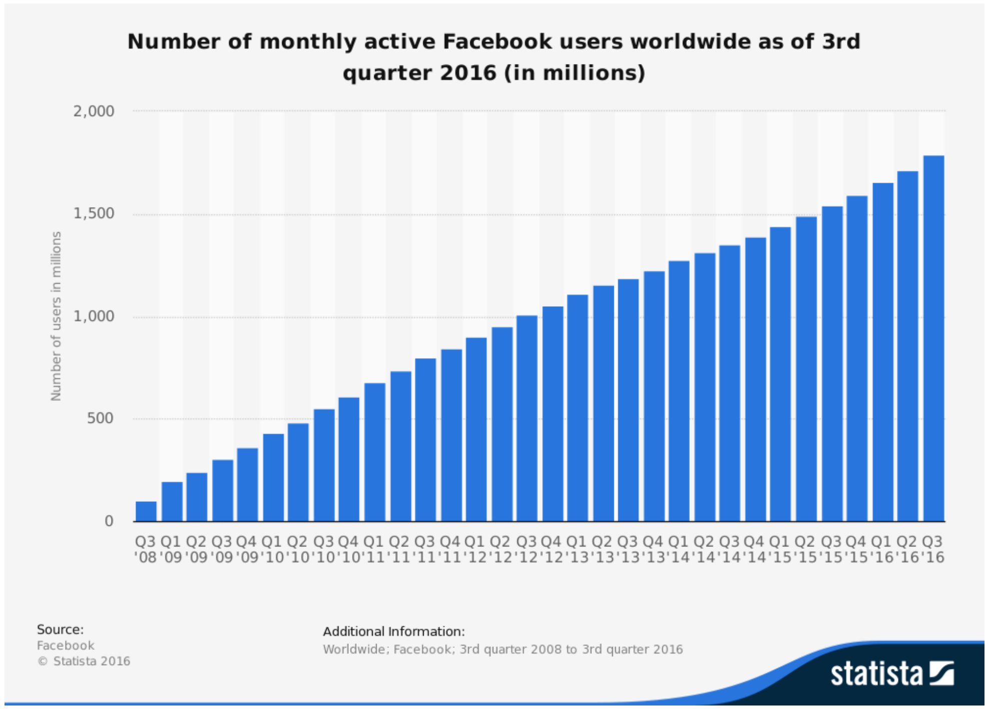
Data whose characteristics exceeds the capabilities of conventional *algorithms*, ***systems*** *and techniques* to derive useful value.

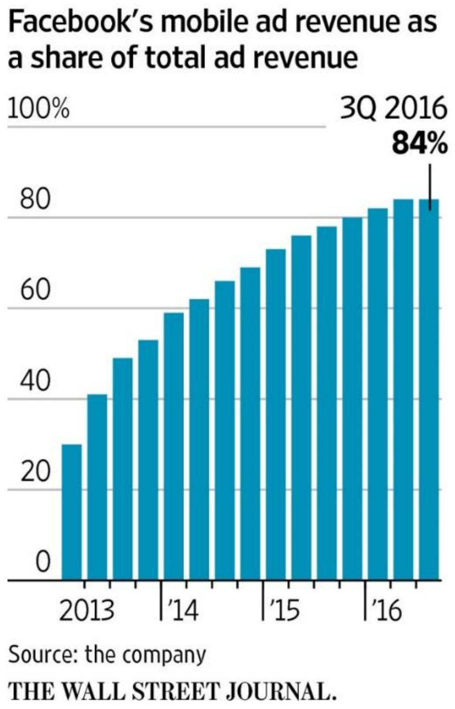
https://www.oreilly.com/ideas/what-is-big-data

Image Credits: https://community.uservoice.com/wp-content/uploads/benefits-of-effective-questions-800x448-300x168.jpg

Where does Big Data Come from?

- Web & Social media, Online retail & governments, scientific instruments, Internet of Everything

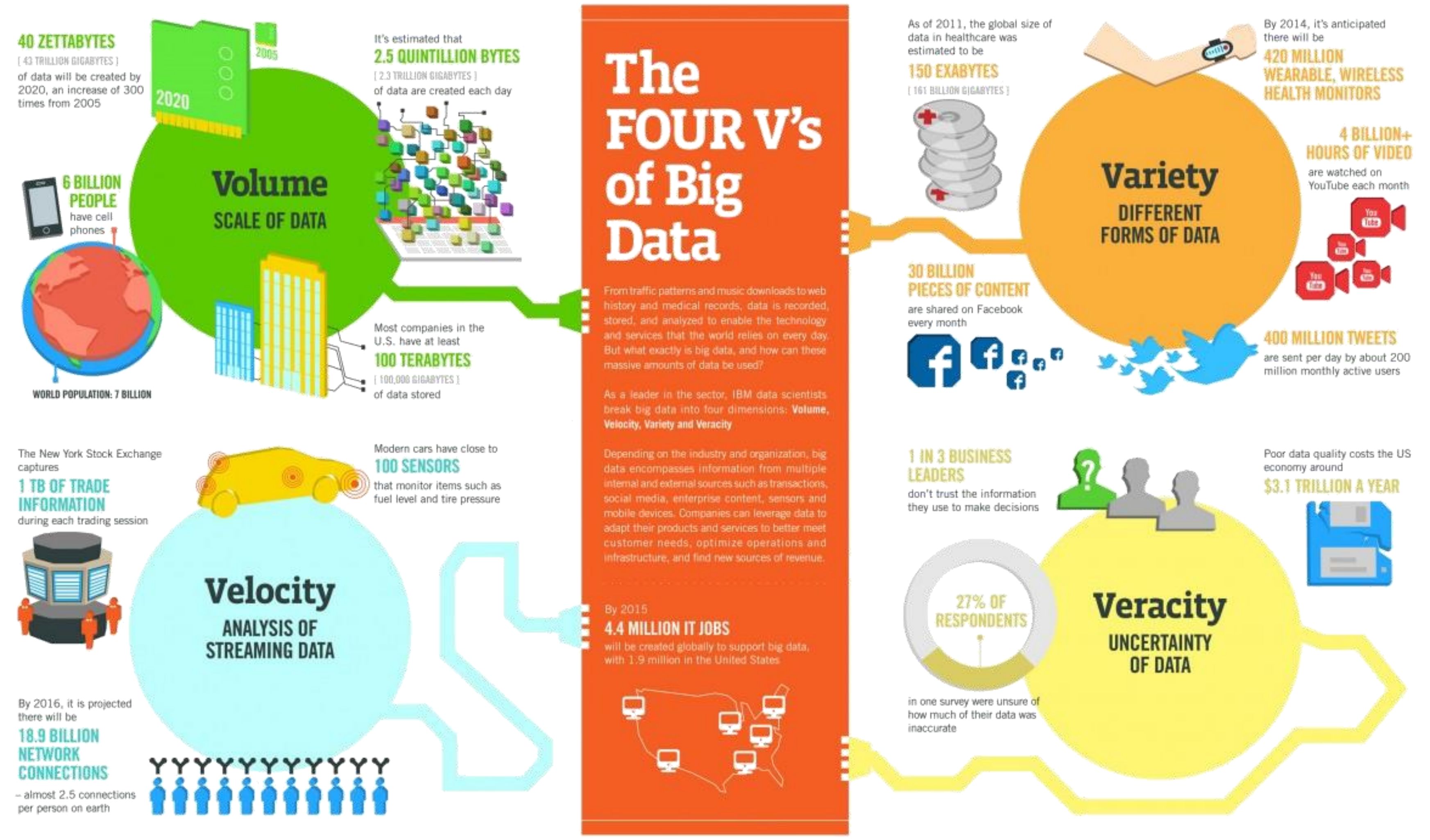




1.79 billion monthly active users as of September 30, 2016

https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/
http://www.wsj.com/articles/facebook-profit-jumps-sharply-1478117646
http://newsroom.fb.com/company-info/

Why is Big Data Difficult?



The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015 **4.4 MILLION IT JOBS** will be created globally to support big data, with 1.9 million in the United States

Volume SCALE OF DATA

40 ZETTABYTES [43 TRILLION GIGABYTES] of data will be created by 2020, an increase of 300 times from 2005

It's estimated that **2.5 QUINTILLION BYTES** [2.3 TRILLION GIGABYTES] of data are created each day

6 BILLION PEOPLE have cell phones

WORLD POPULATION: 7 BILLION

Most companies in the U.S. have at least **100 TERABYTES** [100,000 GIGABYTES] of data stored

Velocity ANALYSIS OF STREAMING DATA

The New York Stock Exchange captures **1 TB OF TRADE INFORMATION** during each trading session

Modern cars have close to **100 SENSORS** that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be **18.9 BILLION NETWORK CONNECTIONS** – almost 2.5 connections per person on earth

Variety DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be **150 EXABYTES** [161 BILLION GIGABYTES]

By 2014, it's anticipated there will be **420 MILLION WEARABLE, WIRELESS HEALTH MONITORS**

4 BILLION+ HOURS OF VIDEO are watched on YouTube each month

30 BILLION PIECES OF CONTENT are shared on Facebook every month

400 MILLION TWEETS are sent per day by about 200 million monthly active users

Veracity UNCERTAINTY OF DATA

1 IN 3 BUSINESS LEADERS don't trust the information they use to make decisions

Poor data quality costs the US economy around **$3.1 TRILLION A YEAR**

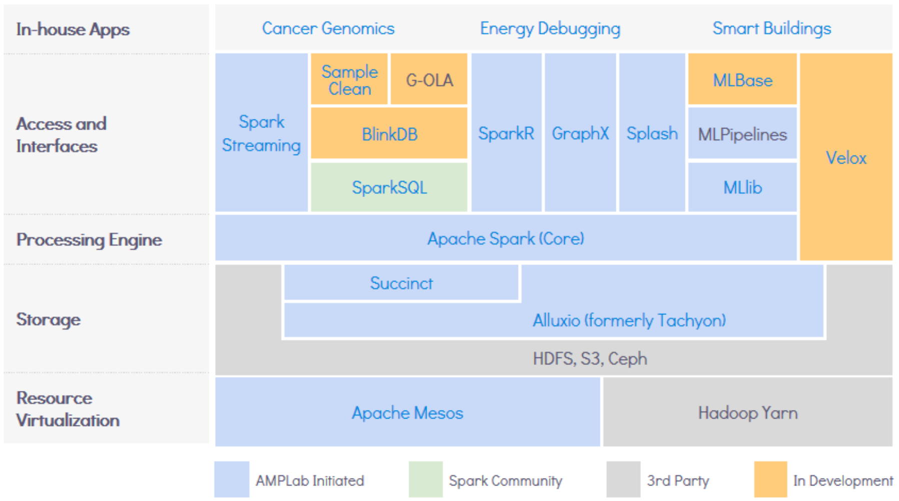
27% OF RESPONDENTS in one survey were unsure of how much of their data was inaccurate

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

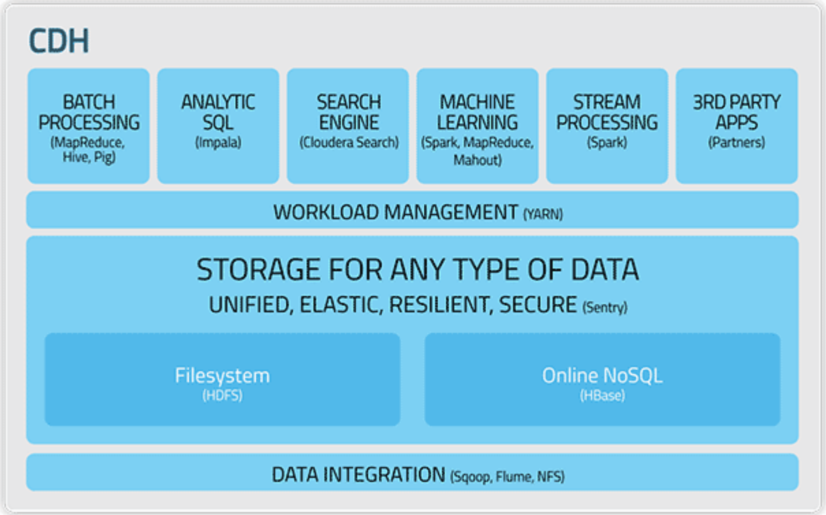
IBM

http://www.ibmbigdatahub.com/infographic/four-vs-big-data

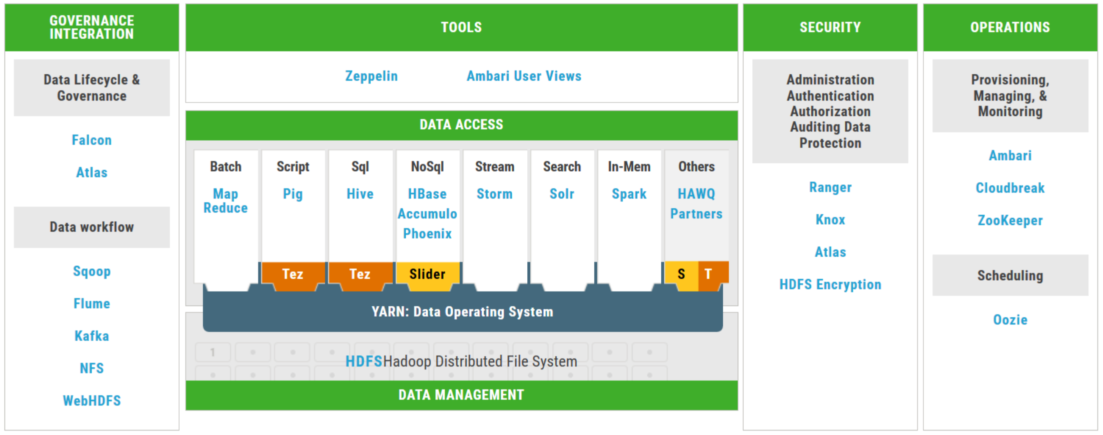
Big Data Platform Stacks



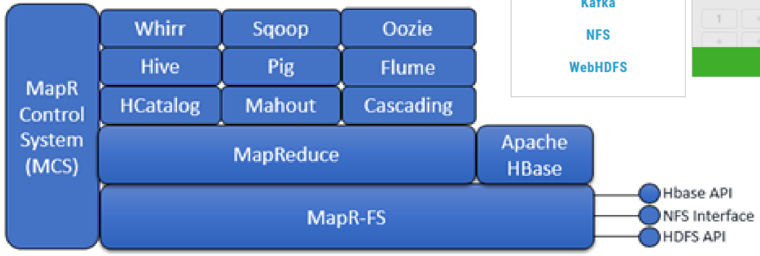
https://amplab.cs.berkeley.edu/software/


CDH
BATCH PROCESSING (MapReduce, Hive, Pig)
ANALYTIC SQL (Impala)
SEARCH ENGINE (Cloudera Search)
MACHINE LEARNING (Spark, MapReduce, Mahout)
STREAM PROCESSING (Spark)
3RD PARTY APPS (Partners)
WORKLOAD MANAGEMENT (YARN)
STORAGE FOR ANY TYPE OF DATA
UNIFIED, ELASTIC, RESILIENT, SECURE (Sentry)
Filesystem (HDFS)
Online NoSQL (HBase)
DATA INTEGRATION (Sqoop, Flume, NFS)

https://www.cloudera.com/documentation/enterprise/5-6-x/topics/cdh_intro.html

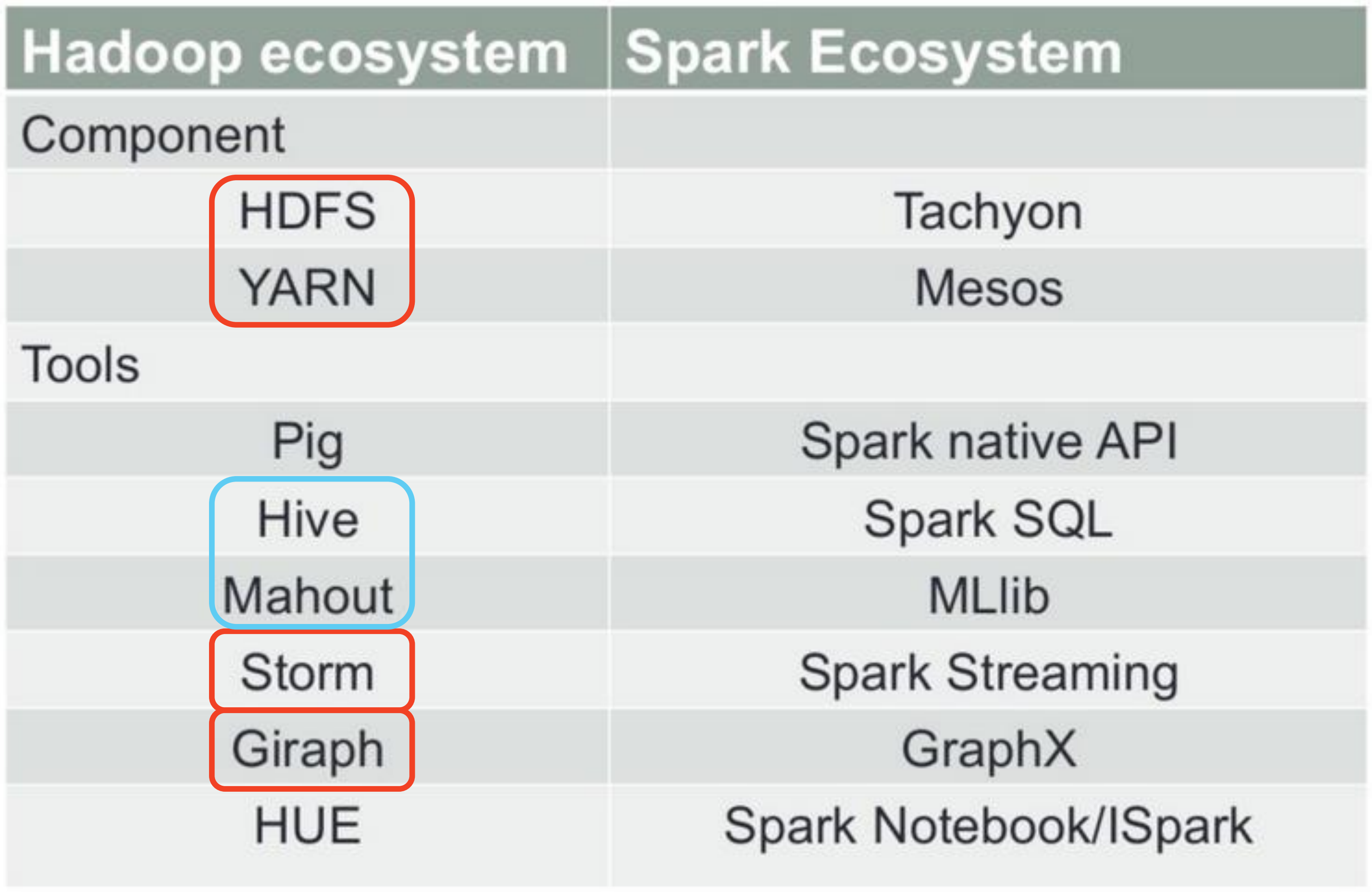


http://doc.mapr.com/display/MapR/MapR+Overview



http://hortonworks.com/products/data-center/hdp/

Hadoop vs. Spark…*Fight!*



Hadoop ecosystem	Spark Ecosystem
Component	
HDFS	Tachyon
YARN	Mesos
Tools	
Pig	Spark native API
Hive	Spark SQL
Mahout	MLlib
Storm	Spark Streaming
Giraph	GraphX
HUE	Spark Notebook/ISpark

https://www.razormind.co.uk/news/the-big-data-answer-hadoop-with-spark

L2 Learning Objectives

1. What is MapReduce & Why is it useful?
2. How does the MapReduce programming model work?
3. How can you design and write simple MR applications?
4. How can you design and write more advanced MR applications? [L3]

Motivation

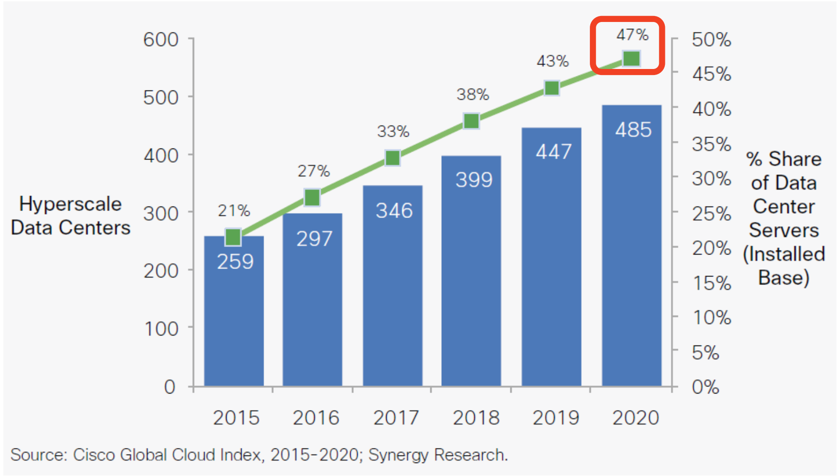
Distributed Systems

- Distributed Computing
  - Clusters of machines
  - Connected over network
- Distributed Storage
  - Disks attached to clusters of machines
  - Network Attached Storage
- *How can we make effective use of multiple machines?*

- **Commodity** clusters vs. **HPC** clusters
  - Commodity: Available off the shelf at large volumes
  - Lower Cost of Acquisition
  - Cost vs. Performance
    - Low disk bandwidth, and high network latency
    - CPU typically comparable (Xeon vs. i3/5/7)
    - Virtualization overhead on Cloud
- *How can we use many machines of modest capability?*

Growth of Cloud Data Centers

Figure 1. Data Center Growth



24 Operators: Microsoft/Azure, Amazon/AWS, Rackspace, Google, Salesforce, ADP, Facebook, Yahoo, Apple, Amazon, Alibaba, eBay,…

Figure 20. Global Data Center Storage Capacity: Traditional vs. Cloud

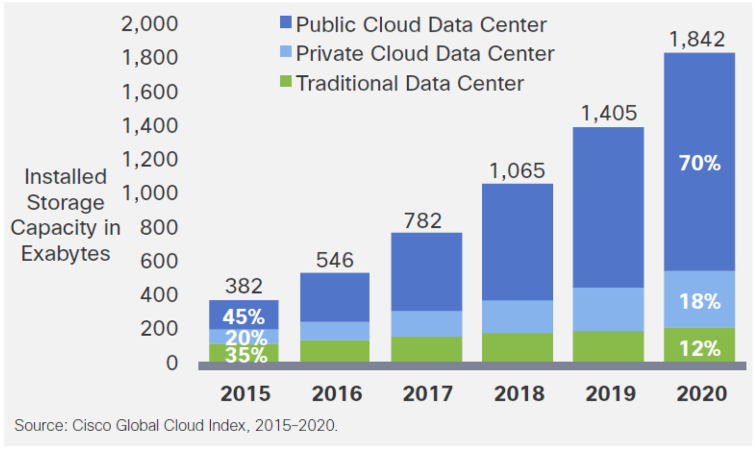


Figure 6. Total Data Center Traffic Growth

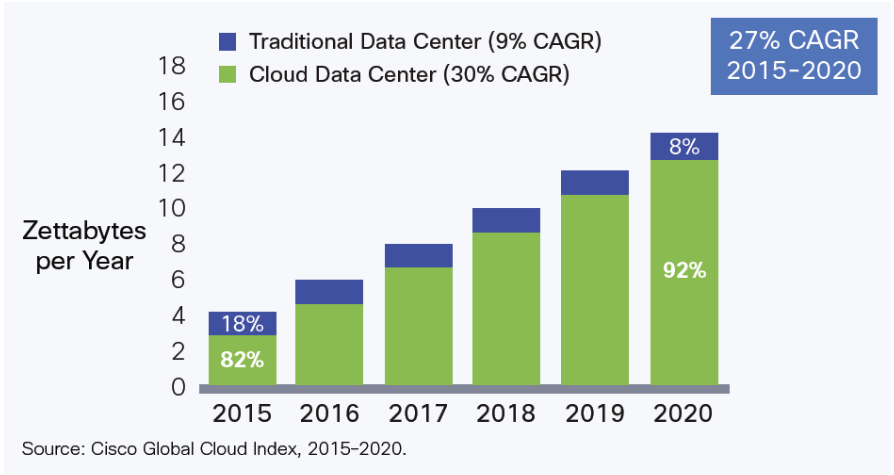
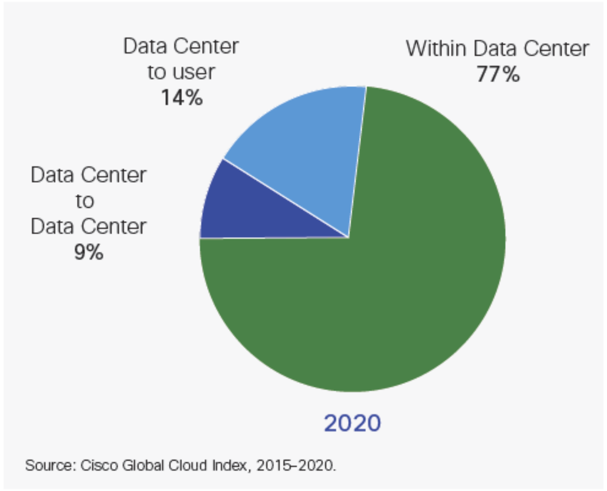


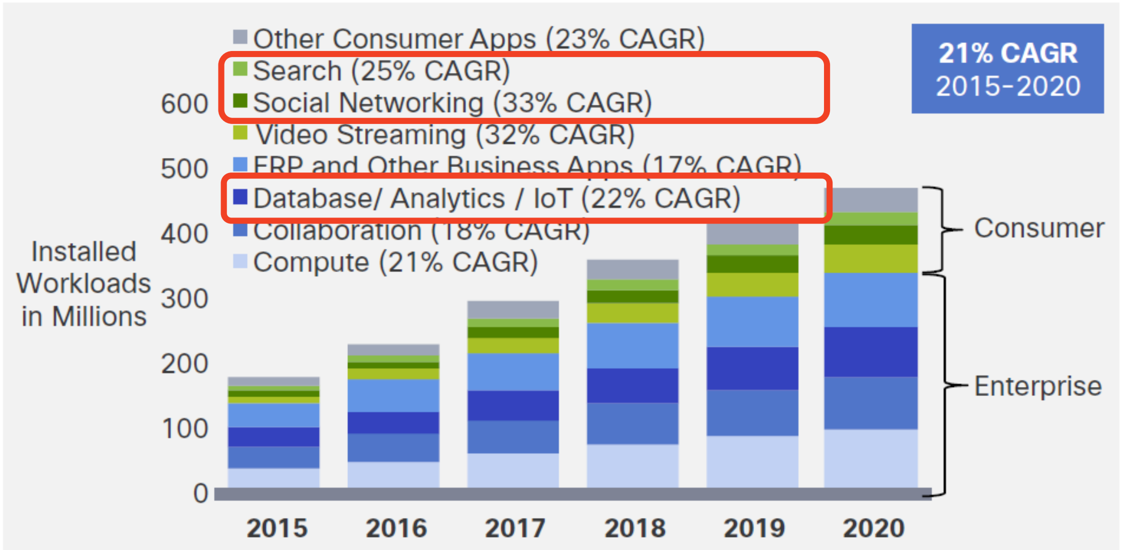
Figure 5. Global Data Center Traffic by Destination in 2020



Cisco Global Cloud Index: Forecast and Methodology, 2015–2020, White Paper © 2016, Cisco

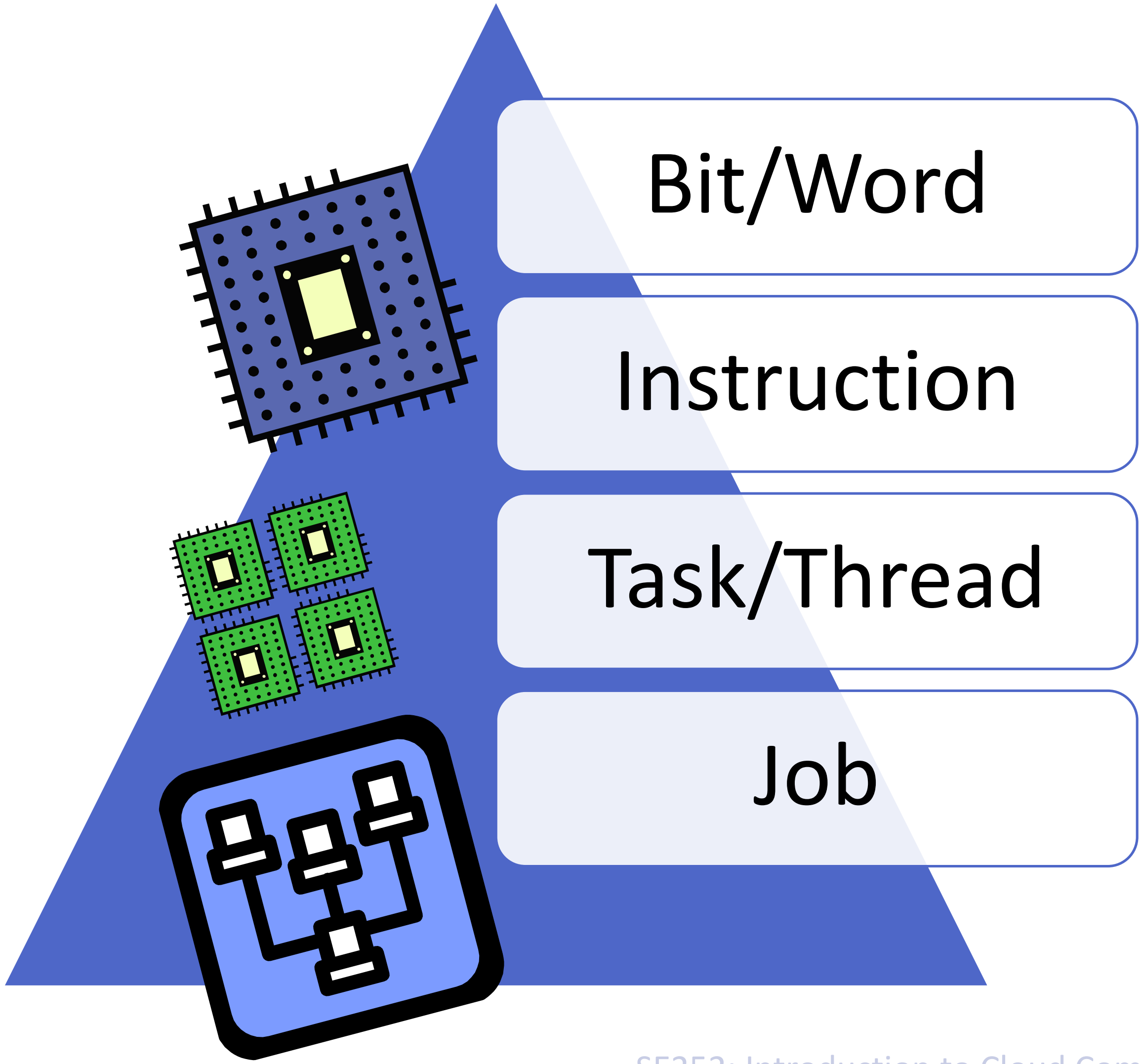
Growth of Cloud Data Centers

Figure 17. Global Data Center Workloads by Applications

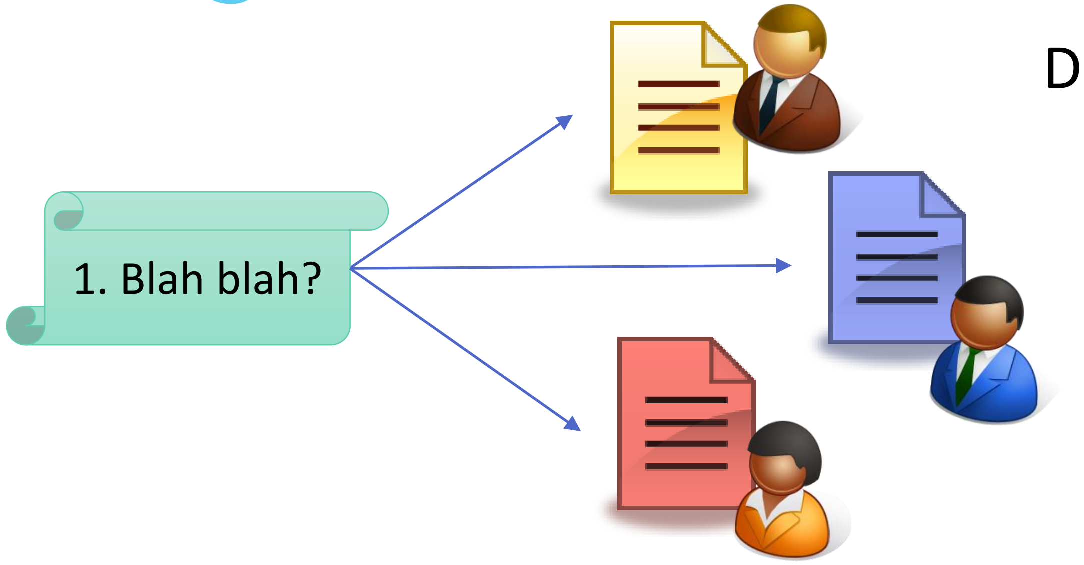


Source: Cisco Global Cloud Index, 2015–2020.

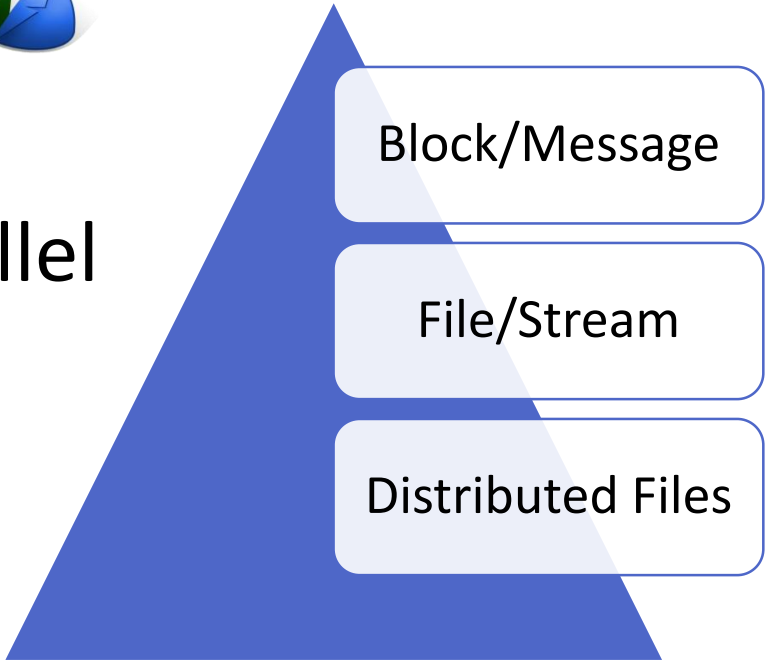
Degrees of parallelism



Degrees of Parallelism



Do your review…*collectively?*



- Data parallel vs. Task Parallel
  - Independent processes
  - Independent data dependency
- Tight vs. Loose Coupling

Scalability

- **System Size**: Higher performance when adding more machines
- **Software**: Can framework and middleware work with larger systems?
- **Technology:** Impact of scaling on time, space and diversity
- **Application**: As problem size grows (compute, data), can the system keep up?
- **Vertical vs Horizontal:** ?
- …

Scalability Metric

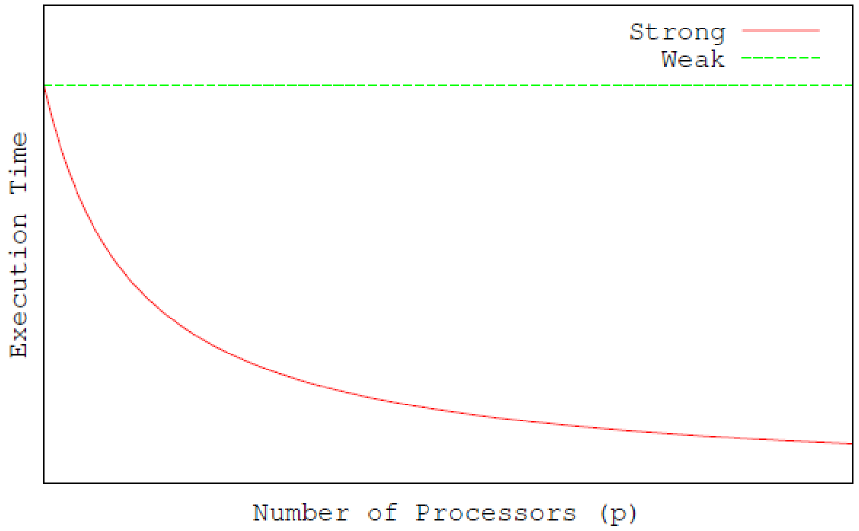
- If the *problem size* is fixed as x and the number of processors available is p

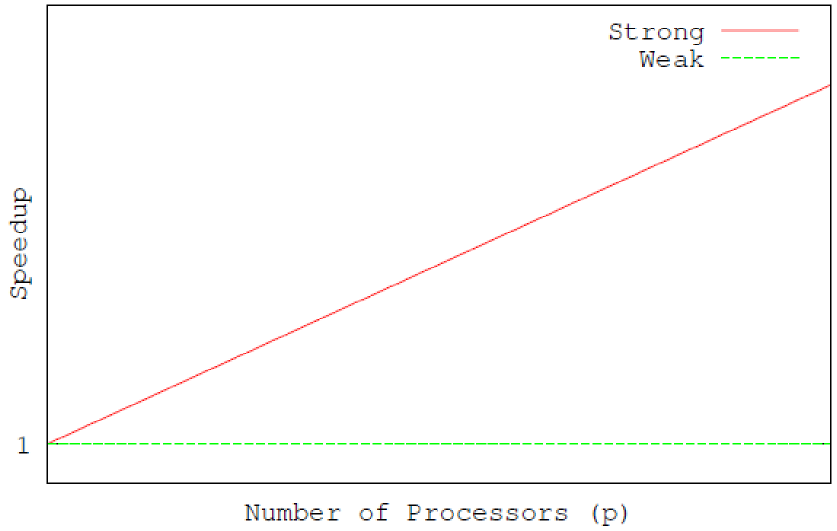
$$Speedup(p,x) = \frac{time(1,x)}{time(p,x)}$$

- If the *problem size per processor* is fixed as x and the number of processors available is p

$$Speedup(p,x.p) = \frac{time(1,x)}{time(p,x.p)}$$

Ideal Strong/Weak Scaling





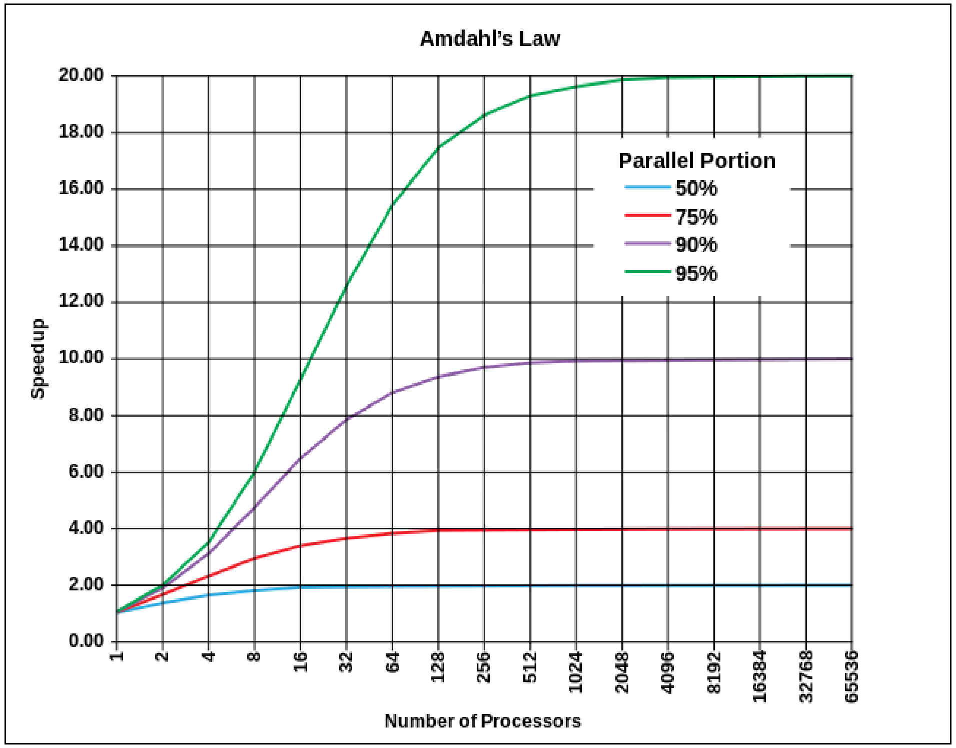
Strong Scaling

- Amdahl's Law for Application Scalability
  - Total problem size is fixed
  - *Speedup limited by sequential bottleneck*
- f_s is *serial* fraction of application
- f_p is fraction of application that can be *parallelized*
- p is number of processors

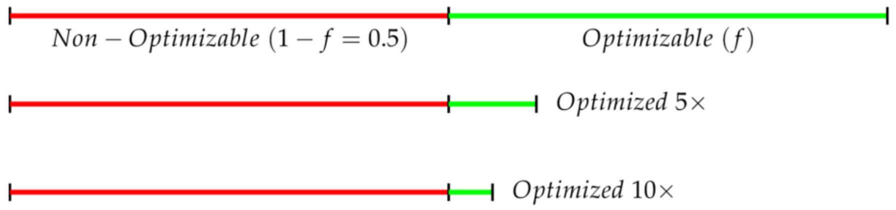
$$Speedup(p,x) = \frac{time(1,x)}{time(p,x)}$$

$$= \frac{1}{f_s + \frac{f_p}{p}}$$

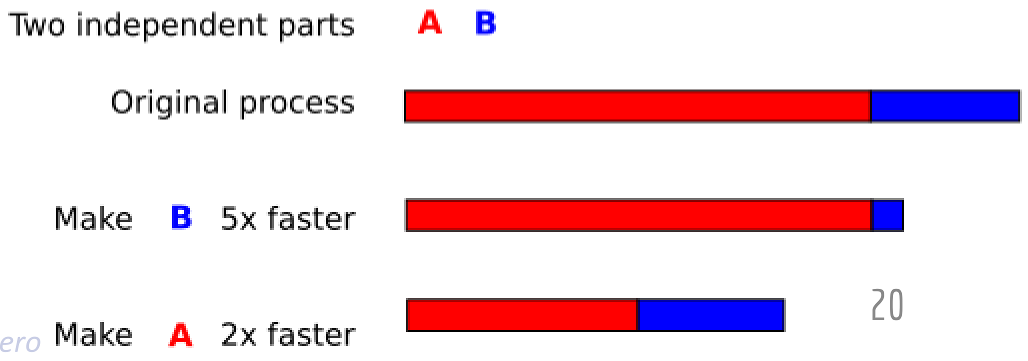
Amdahl's Law



© Daniels220 at English Wikipedia



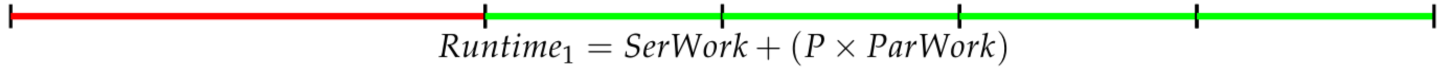
© Martha A. Kim

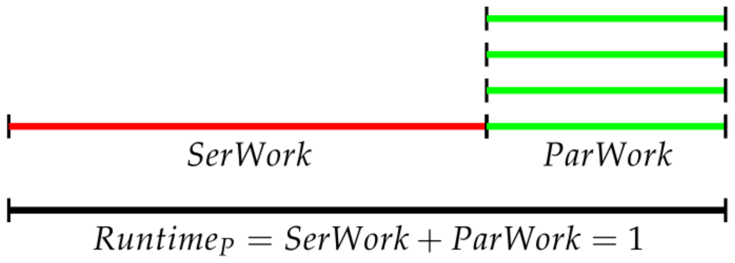


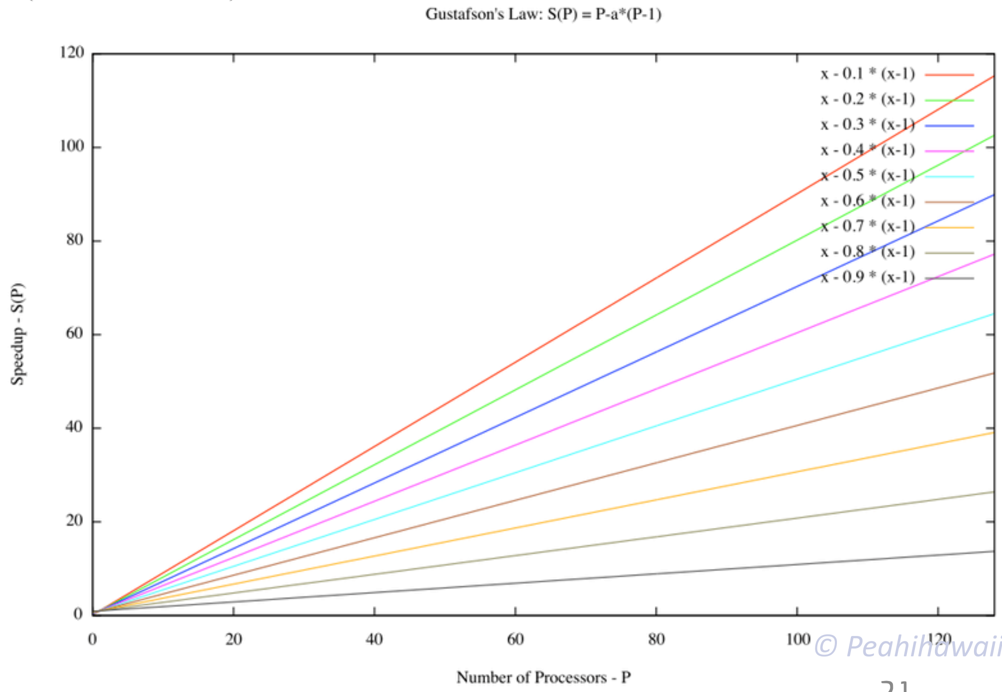
© Gorivero

Weak Scaling

- Gustafson's Law of weak scaling
  - Problem size increases with number of processors
  - "Scaled speedup"







Scalability

- Strong vs. Weak Scaling
- **Strong Scaling**: How the performance varies with the # of processors for a *fixed total problem size*
- **Weak Scaling**: How the performance varies with the # of processors for a *fixed problem size per processor*
    - MapReduce is intended for “Weak Scaling”

Ease of Programming

- Programming distributed systems is difficult
  - Divide a job into multiple tasks
  - Understand dependencies between tasks: Control, Data
  - Coordinate and synchronize execution of tasks
  - Pass information between tasks
  - Avoid race conditions, deadlocks
- Parallel and distributed programming models/languages/abstractions/platforms try to make these easy
  - E.g. Assembly programming vs. C++ programming
  - E.g. C++ programming vs. Matlab programming

Availability, Failure

- Commodity clusters have lower reliability
  - Mass-produced
  - Cheaper materials
  - Smaller lifetime (~3 years)
- *How can applications easily deal with failures?*
- *How can we ensure availability in the presence of faults?*

Map Reduce

Patterns & Technologies

- **MapReduce** is a distributed data-parallel programming model from Google
- MapReduce works best with a distributed file system, called **Google File System (GFS)**
- **Hadoop** is the open source framework implementation from Apache that can execute the MapReduce programming model
- **Hadoop Distributed File System (HDFS)** is the open source implementation of the GFS design
- **Elastic MapReduce (EMR)** is Amazon's PaaS

MapReduce

“A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.”

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

MapReduce Design Pattern

- Clean abstraction for programmers
- Automatic parallelization & distribution

- Fault-tolerance
- A batch data processing system
- Provides status and monitoring tools

MapReduce: Data-parallel Programming Model

- Process data using map & reduce functions
- `map(`k_i`, `v_i`)` → `List<`k_m`, `v_m`>[]`
  - *map* is called on every input item
  - Emits a series of intermediate key/value pairs
- All values with a given key are **grouped** together
- `reduce(`k_m`, List<`v_m`>[])` → `List<`k_r`, `v_r`>[]`
  - ***reduce*** is called on every unique key & all its values
  - Emits a value that is added to the output

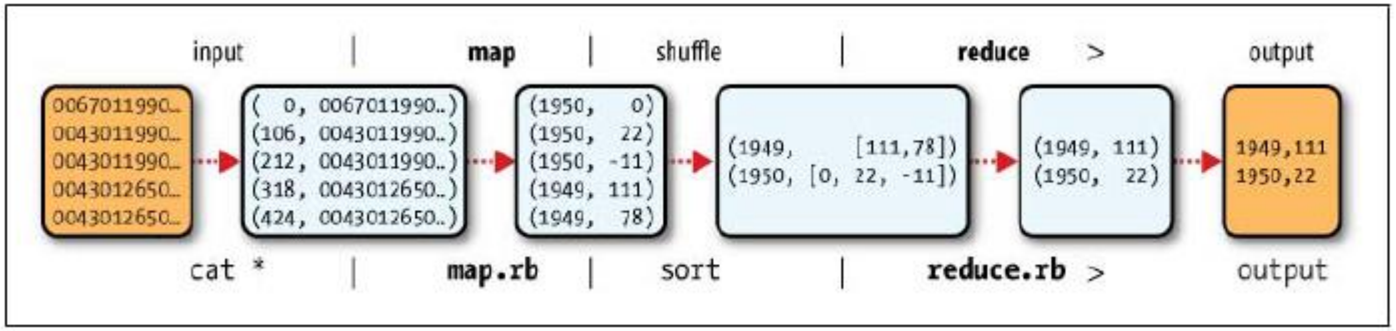


Figure 2-1. MapReduce logical data flow

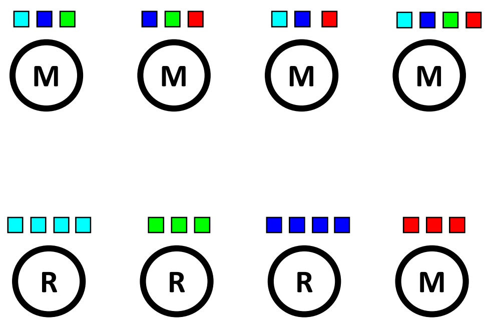
MR Borrows from Functional Programming

- Functional operations do not modify data structures
  - *They always create new ones*
  - Original data still exists in unmodified form (read only)
- Data flows are implicit in program design
- Order of operations does not matter
  - *Commutative*: **a ◊ b ◊ c = b ◊ a ◊ c = c ◊ b ◊ a**

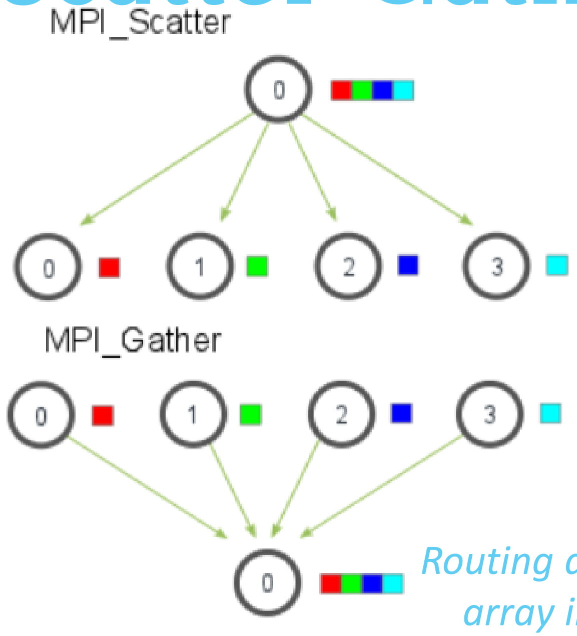
MR Borrows from Functional Programming

- In a purely functional setting
  - Elements computed by **map** cannot see the effects of map on other elements
  - Order of applying **reduce** is *commutative*
    - **a ◊ b = b ◊ a**
    - Allowing parallel/reordered execution
  - More optimizations possible if **reduce** is also *associative*
    - **(a ◊ b) ◊ c = a ◊ (b ◊ c)**

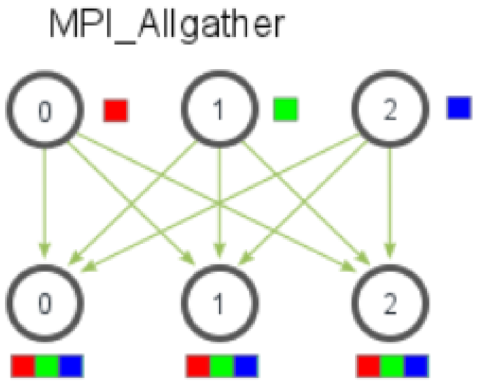
MapReduce & MPI Scatter-Gather



Routing determined by key

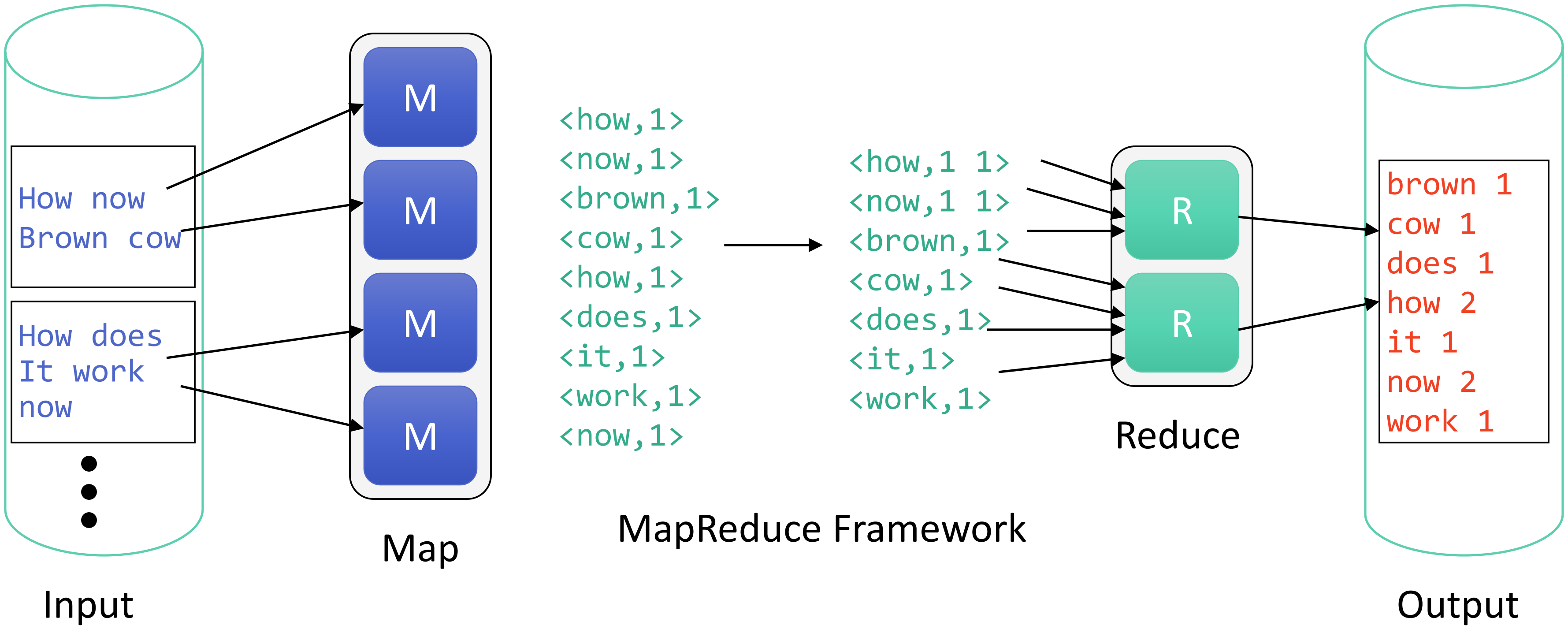


Routing determined by array index/element position



MapReduce: Word Count

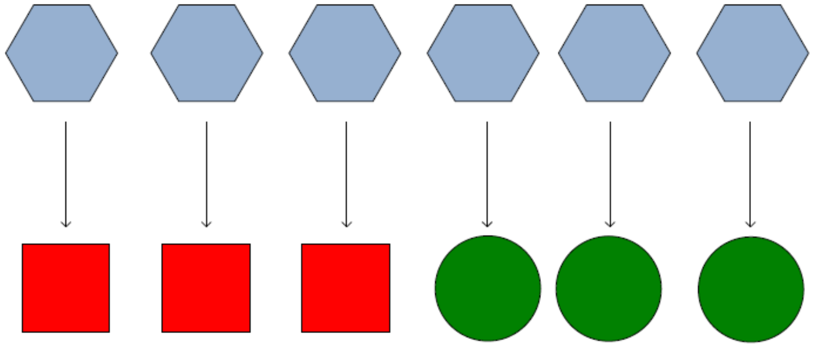
```
Map(k1,v1) → list(k2,v2)
Reduce(k2, list(v2)) → list(k2,v2)
```



Distributed Word Count

Map

- Input records from the data source
  - lines out of files, rows of a database, etc.
- Passed to map function as key-value pairs
  - Line number, line value
- map() produces *zero or more intermediate values*, each associated with an output key



Map

- **Example Wordcount**

```
map(String input_key, String input_value):
  // input_key: line number
  // input_value: line of text
  for each Word w in input_value.tokenize()
    EmitIntermediate(w, "1");
```

```
(0, “How now brown cow”) →
   [(“How”, 1), (“now”, 1), (“brown”, 1), (“cow”, 1)]
```

- **Example: Upper-case Mapper**

```
map(k, v) { emit(k.toUpper(), v.toUpper()); }

("foo", "bar") → ("FOO", "BAR")
("Foo", "other") → ("FOO", "OTHER")
("key2", "data") → ("KEY2", "DATA")
```

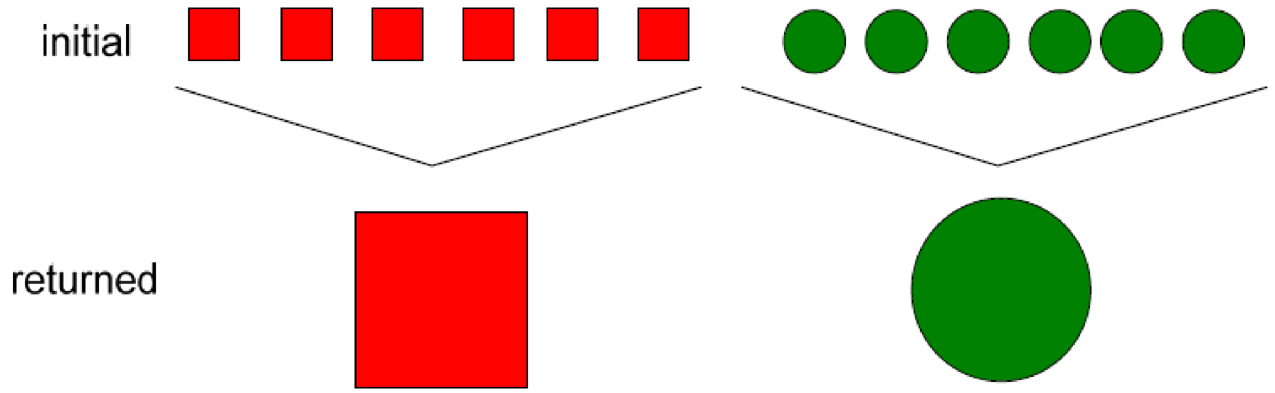
- **Example: Filter Mapper**

```
map(k, v) { if (isPrime(v)) then emit(k, v); }

("foo", 7) → ("foo", 7)
("test", 10) → ()        //nothing emitted
```

Reduce

- All the intermediate values from map for a given output key are combined together into a list
- reduce() combines these intermediate values into one or more final values for that same output key … Usually one final value per key
- One output "file" per reducer



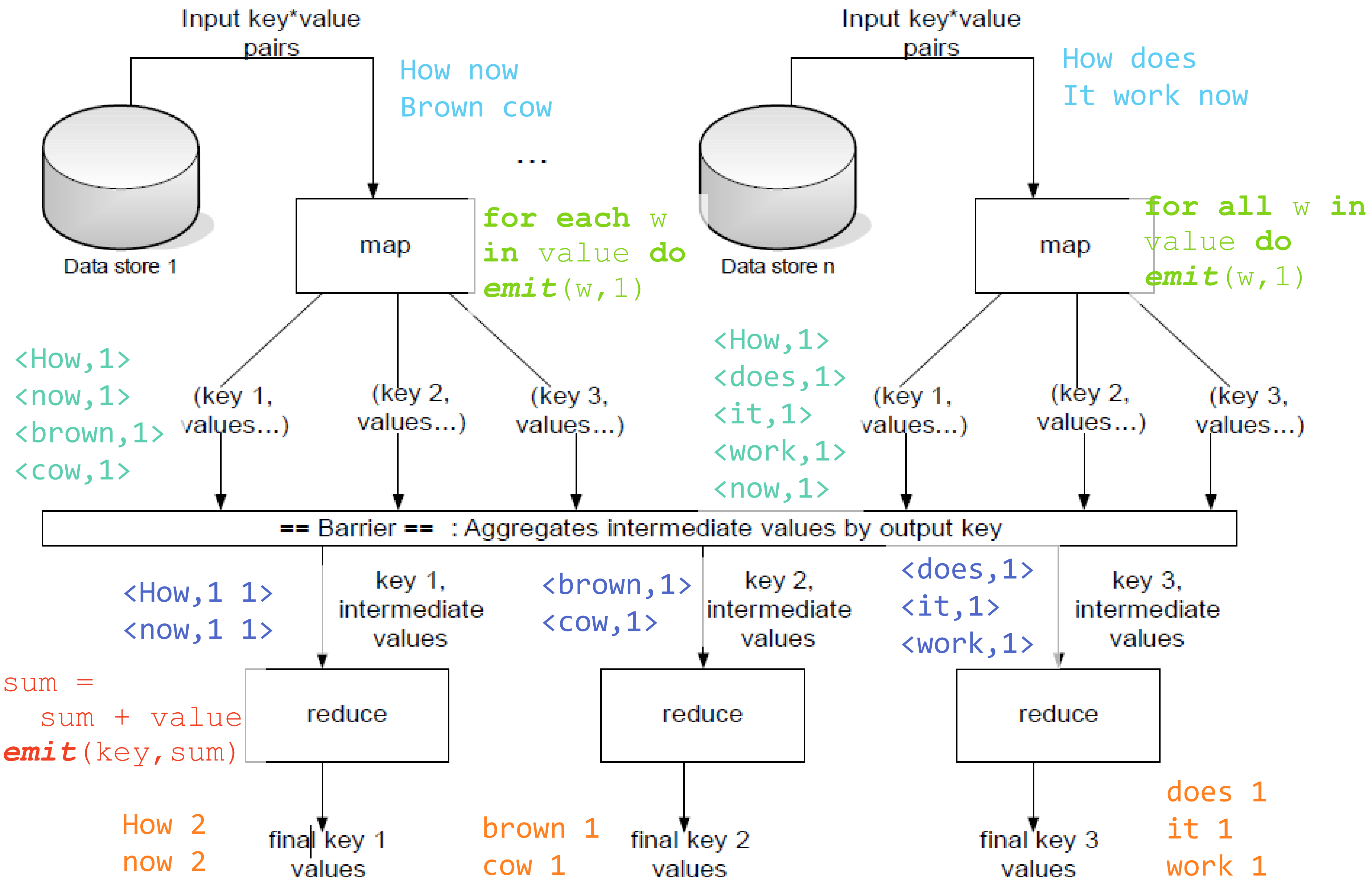
Reduce

- **Example Wordcount**

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

```
(“A”, [1, 1, 1]) → (“A”, 3)
(“B”, [1, 1]) → (“B”, 2)
```

MapReduce: Word Count Drilldown



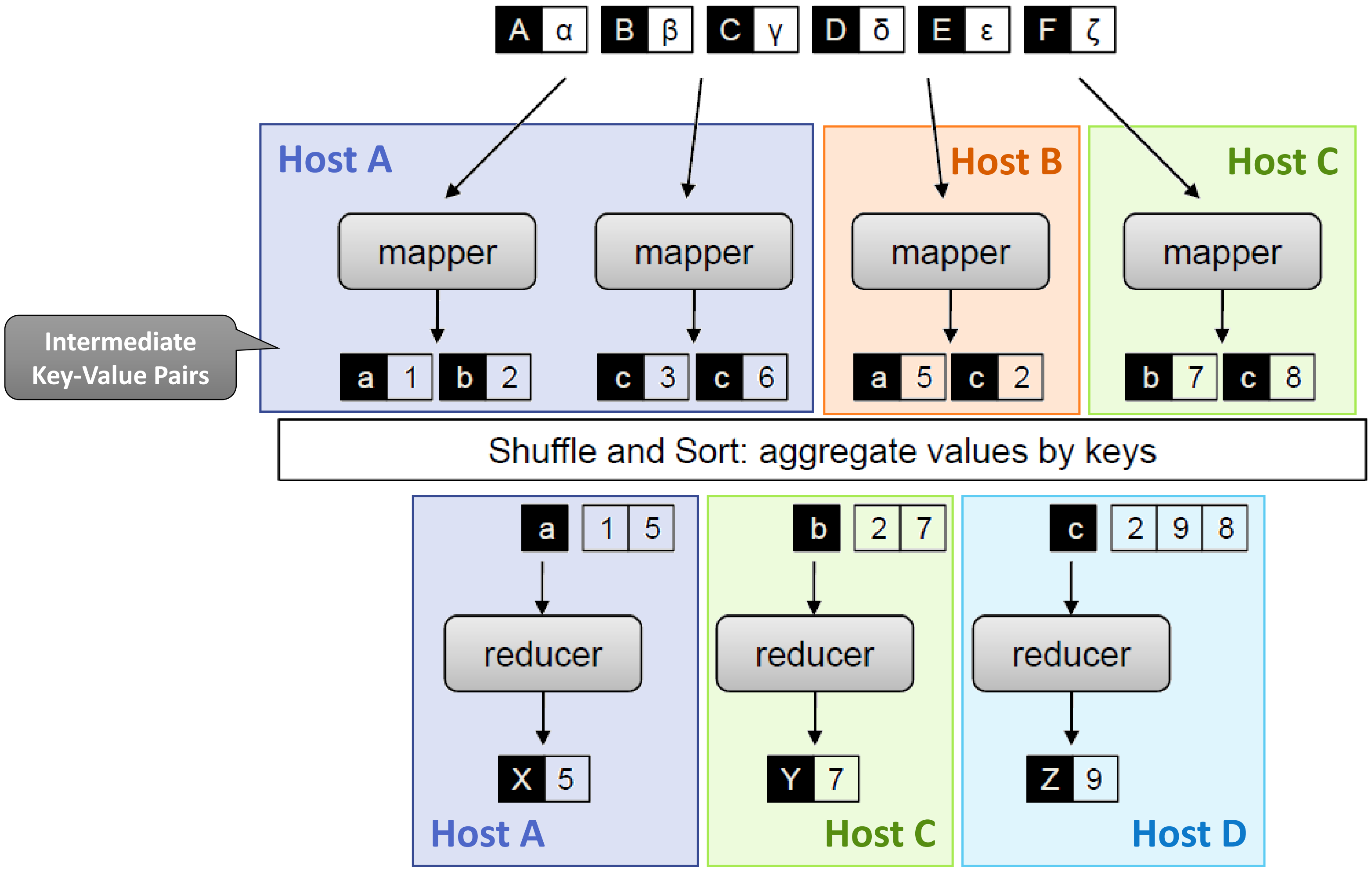
Mapper/Reducer Tasks *vs.* Map/Reduce Methods

- Number of Mapper and Reducer tasks is specified by user
- Each **Mapper/Reducer** **task** can make multiple calls to **Map/Reduce** **method**, sequentially
- Mapper and Reducer tasks may run on different machines
- Implementation framework decides
  - Placement of Mapper and Reducer tasks on machines
  - Keys assigned to mapper and reducer tasks
  - But can be controlled by user…

Shuffle & Sort
The Magic happens here!

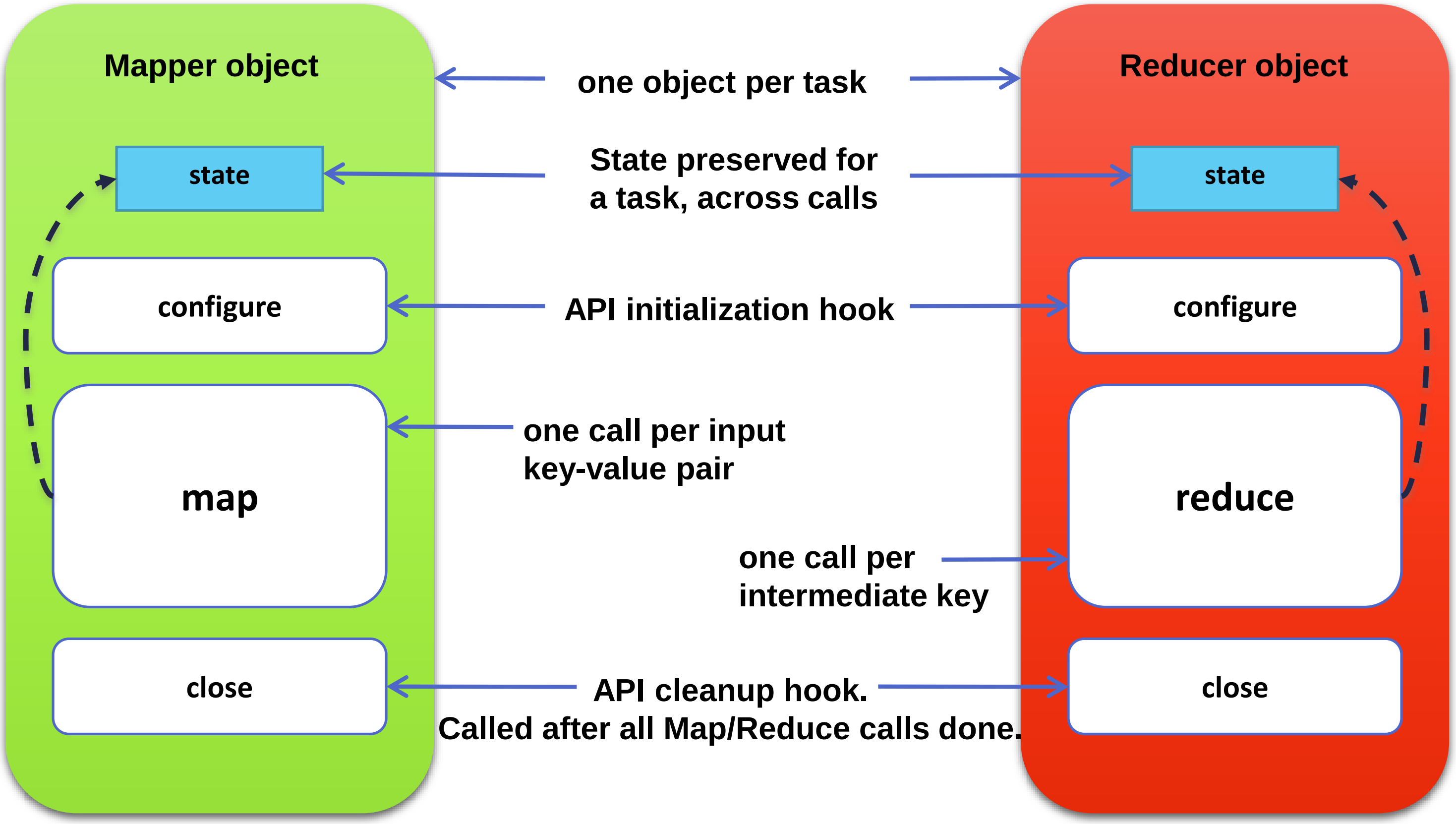
- **Shuffle** does a "group by" of keys from all mappers
  - Similar to SQL goupBy operation
- **Sort** of *local keys* to *Reducer task* performed
  - Keys arriving at each reducer are sorted
  - No sorting guarantee of keys across reducer tasks
- No ordering guarantees of values for a key
  - Implementation dependent
- Shuffle and Sort *implemented efficiently* by framework

Map-*Shuffle-Sort*-Reduce



Maintainer State in Tasks

- Capture state & dependencies across multiple keys and values


Mapper object
state
configure
map
close
one object per task
State preserved for a task, across calls
API initialization hook
one call per input key-value pair
one call per intermediate key
API cleanup hook. Called after all Map/Reduce calls done.
Reducer object
state
configure
reduce
close

www.cs.bu.edu/faculty/gkollios/ada14

Improve Word Count using State?

```
map(String k, String v)
  foreach w in v.tokenize()
    emit(w, "1")
```

```
reduce(String k, int[] v)
  int sum = 0
  foreach n in v[]  sum += v
  emit(k, sum)
```

```
  mapperInit()
    H = new HashMap<String,int>()
  map(String k, String v)
    foreach w in v.tokenize()
      H[w] = H[w] + 1
  mapperClose()
   foreach w in H.keys()
    emit(w, H[w])
```

```
  reduce(String k, int[] v)
    int sum = 0;
    foreach n in v[]  sum += v
    emit(k, sum)
```

Anagram Example

- *"An **anagram** is a type of word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once; for example **orchestra** can be rearranged into **carthorse**." … Wikipedia*
  - thickens = kitchens, reserved = reversed,
  - cheating = teaching, cause = sauce
  - Tom Marvolo Riddle = I am Lord Voldemort
- Problem: Find ALL anagrams in the English dictionary of ~1M words (10^6)
- *1M X 1M comparisons?*

Anagram Example

```
public class AnagramMapper extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, Text> {
  private Text sortedText = new Text();
  private Text orginalText = new Text();
  public void map(LongWritable key, Text value,
    OutputCollector<Text, Text> outputCollector, Reporter reporter) {
    String word = value.toString();
    char[] wordChars = word.toCharArray();
    Arrays.sort(wordChars);
    String sortedWord = new String(wordChars);
    sortedText.set(sortedWord);
    orginalText.set(word);
    // Sort word and emit <sorted word, word>
    outputCollector.collect(sortedText, orginalText);
  }
}
```

http://code.google.com/p/hadoop-map-reduce-examples/

Anagram Example…

```
public void reduce(Text anagramKey, Iterator<Text> anagramValues,
       OutputCollector<Text, Text> results, Reporter reporter) {
    String output = "";
    while(anagramValues.hasNext()) {
       Text anagram = anagramValues.next();
       output = output + anagram.toString() + "~";
    }
    StringTokenizer outputTokenizer =
       new StringTokenizer(output,"~");
    // if the values contain more than one word
    // we have spotted a anagram.
    if(outputTokenizer.countTokens()>=2) {
       output = output.replace("~", ",");
       outputKey.set(anagramKey.toString());
       outputValue.set(output);
       results.collect(outputKey, outputValue);
    }
}
```

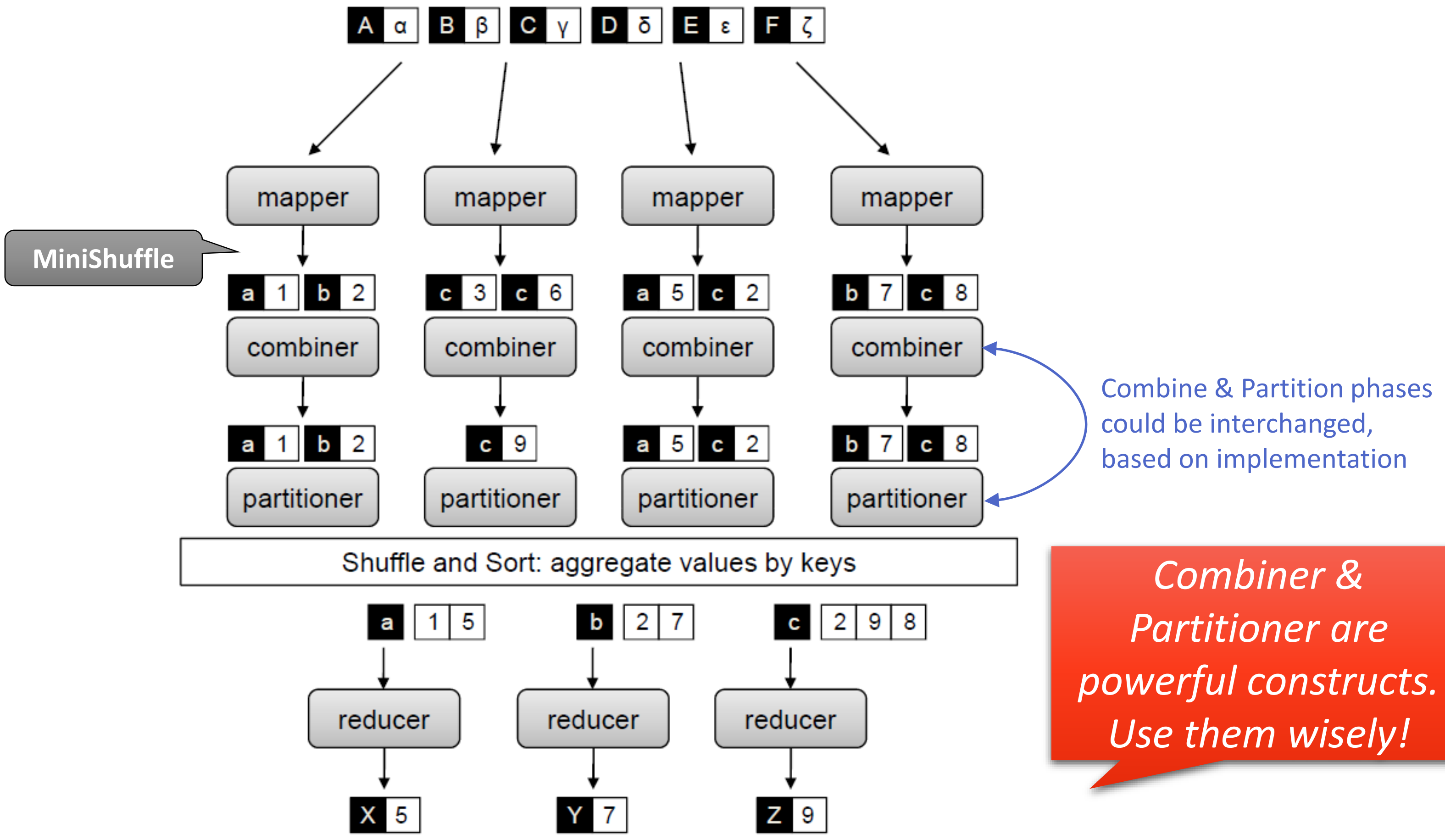
Optimization: Combiner

- Logic runs on output of Map tasks, on the map machines
  - "Mini-Reduce," only on local Map output
- Output of Combiner sent to shuffle
  - Saves bandwidth before sending data to Reducers
- Same *input* and *output types* as Map's *output* type
  - `Map(k,v) → (k',v')`
  - `Combine(k',v'[]) → (k',v')`
  - `Reduce(k',v'[]) → (k'',v'')`
- Reduce task logic can be **used directly** as combiner **if** commutative & associative. Usually for trivial ops.
- Combiner may be called 0, 1 or more times

Optimization: **Partitioner**

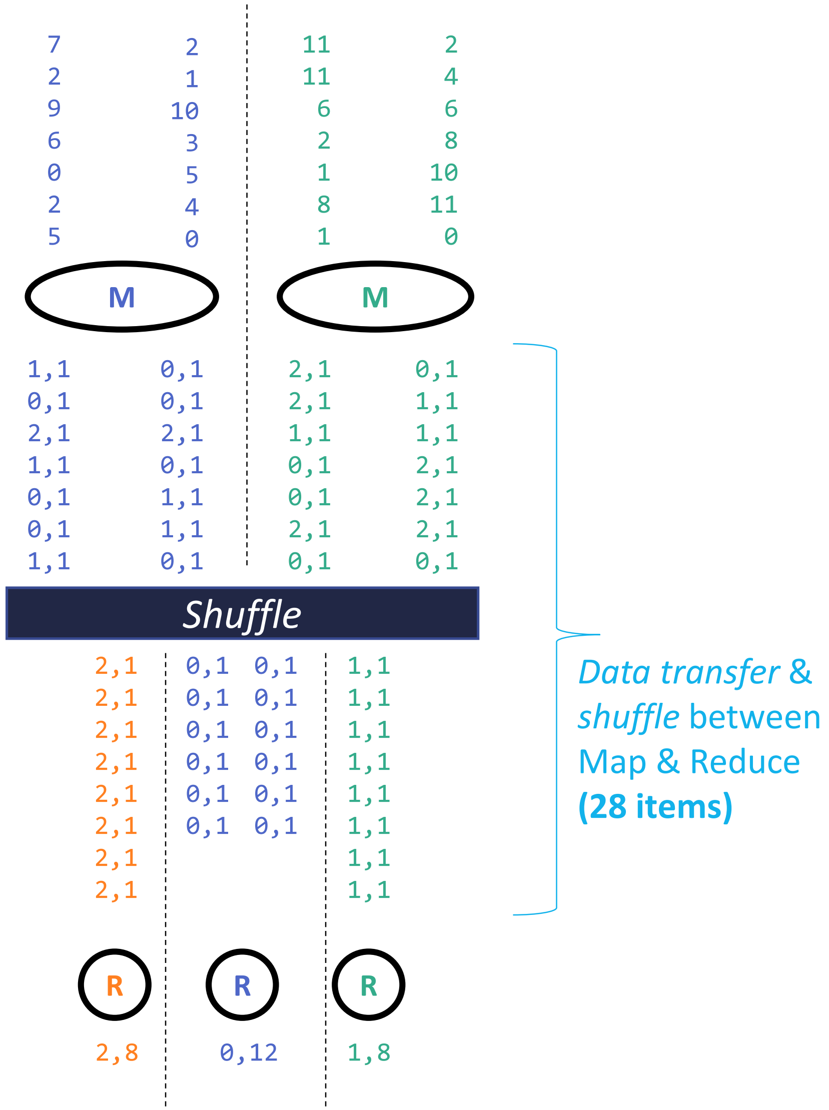
- Decides assignment of intermediate keys grouped to specific Reducer tasks
  - Affects the load on each reducer task
- Sorting of local keys for Reducer task done after partitioning
- Default is hash partitioning
  - **`HashPartitioner(key, nParts)` → `part`**
  - Number of Reducer (**nParts**) tasks known in advance
  - Returns a partition number [0, nParts)
  - Default partitioner balances number of keys per Reducer … *assuming uniform key distribution*
  - May not balance the number of values processed by a Reducer

Map-*MiniShuffle-Combine-Partition*-Shuffle-Sort-**Reduce**



Combiner & Partitioner are powerful constructs. Use them wisely!

MapReduce for Histogram



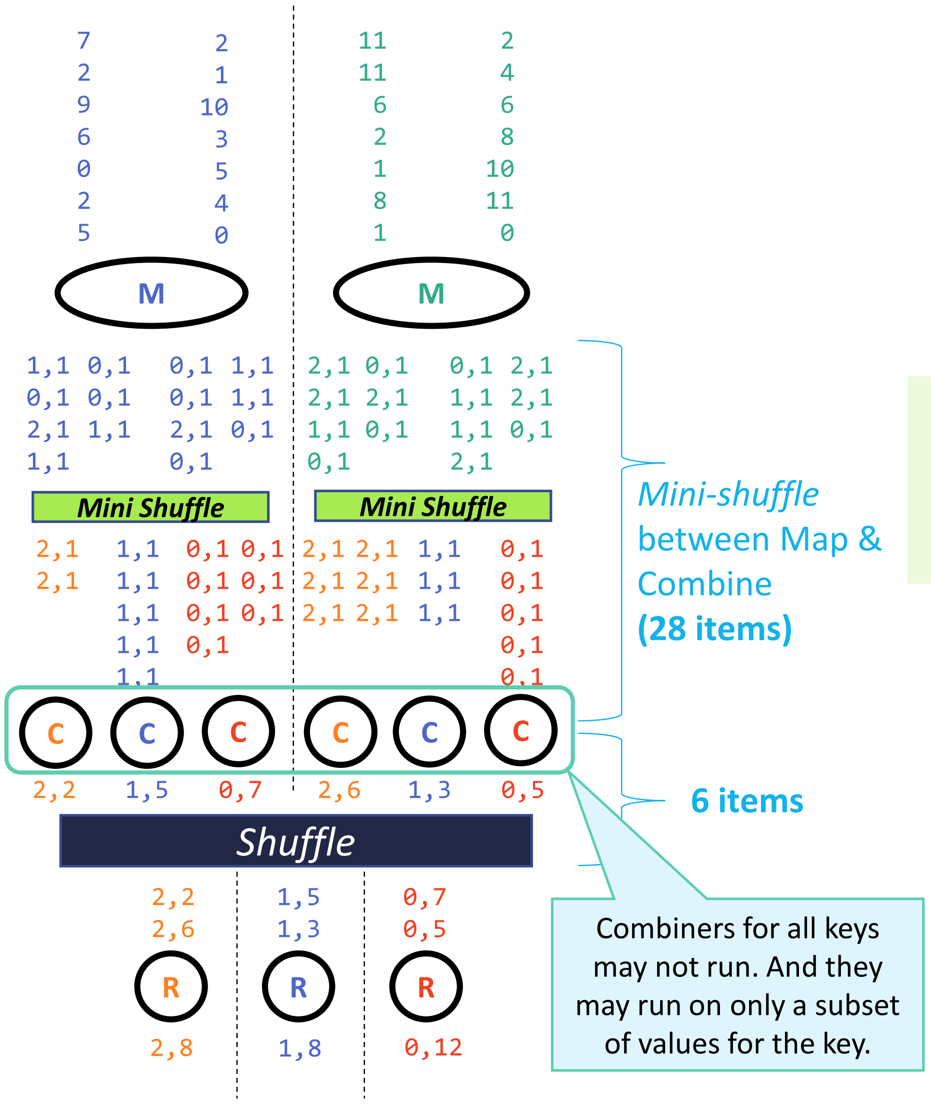
Data transfer & shuffle between Map & Reduce **(28 items)**

```
int bucketWidth = 4 // input

Map(k, v) {
    emit(floor(v/bucketWidth), 1)
    // <bucketID, 1>
}

// one reduce per bucketID
Reduce(k, v[]){
    sum=0;
    foreach(n in v[])  sum++;
    emit(k, sum)
    // <bucketID, frequency>
}
```

MapReduce for Histogram



```
int bucketWidth = 4 // input

Map(k, v) {
    emit(floor(v/bucketWidth), 1)
    // <bucketID, 1>
}

Combine(k, v[]){
    // same code as Reduce()
}

// one reduce per bucketID
Reduce(k, v[]){
    sum=0;
    foreach(n in v[])  sum++;
    emit(k, sum)
    // <bucketID, frequency>
}
```

Since Reducer is *commutative* and *associative*, its logic can be used as a Combiner

Combiner Advantage

- Mini-Shuffle lowers the overall cost for Shuffle
- E.g. *n* total items emitted from *m* mappers
- NW Transfer and Disk IO costs
  - In ideal case, `m` items vs. `n` items *written and read from disk, transferred over network* (`m<<n`)
- Shuffle, less of an impact
  - If more mapper tasks are present than reducers, higher parallelism for doing groupby and mapper-side partial sort.
  - Local Sort on reducer is based on number of unique keys, which does not change due to combiner.

Quick Assignment

Find the Mean of a set of numbers

Map Input: ×, int e.g., <×,8>,<×,32>,<×,20>,<×,4>

Reduce Output: ×, int e.g. <×,16>

Computing the Mean: *Simple Approach*

```
1: class MAPPER
2:     method MAP(string $t$, integer $r$)
3:         EMIT(string $t$, integer $r$)
```

All work performed by single Reducer!

```
1: class REDUCER
2:     method REDUCE(string $t$, integers $[r_1, r_2, \ldots]$)
3:         $sum \leftarrow 0$
4:         $cnt \leftarrow 0$
5:         for all integer $r \in$ integers $[r_1, r_2, \ldots]$ do
6:             $sum \leftarrow sum + r$
7:             $cnt \leftarrow cnt + 1$
8:         $r_{avg} \leftarrow sum/cnt$
9:         EMIT(string $t$, integer $r_{avg}$)
```

Optimization: Can we use Reducer as Combiner?

Computing the Mean: *Using a Combiner*

```
1: class Mapper
2:     method Map(string $t$, integer $r$)
3:         Emit(string $t$, integer $r$)

1: class Combiner
2:     method Combine(string $t$, integers $[r_1, r_2, \ldots]$)
3:         $sum \leftarrow 0$
4:         $cnt \leftarrow 0$
5:         for all integer $r \in$ integers $[r_1, r_2, \ldots]$ do
6:             $sum \leftarrow sum + r$
7:             $cnt \leftarrow cnt + 1$
8:         Emit(string $t$, pair $(sum, cnt)$)                ▷ Separate sum and count

1: class Reducer
2:     method Reduce(string $t$, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
3:         $sum \leftarrow 0$
4:         $cnt \leftarrow 0$
5:         for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ do
6:             $sum \leftarrow sum + s$
7:             $cnt \leftarrow cnt + c$
8:         $r_{avg} \leftarrow sum/cnt$
9:         Emit(string $t$, integer $r_{avg}$)
```

Is this correct?

Computing the Mean: Fixed?

```
class MAPPER
    method MAP(string $t$, integer $r$)
        EMIT(string $t$, pair $(r, 1)$)
```

```
class COMBINER
    method COMBINE(string $t$, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ do
            $sum \leftarrow sum + s$
            $cnt \leftarrow cnt + c$
        EMIT(string $t$, pair $(sum, cnt)$)
```

```
class REDUCER
    method REDUCE(string $t$, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ do
            $sum \leftarrow sum + s$
            $cnt \leftarrow cnt + c$
        $r_{avg} \leftarrow sum/cnt$
        EMIT(string $t$, pair $(r_{avg}, cnt)$)
```

www.cs.bu.edu/faculty/gkollios/ada14

TeraSort: Sorting Large Files

- Input is a list of positive numbers (or words)
  - Say a **terabyte** of data, 10^{11} entries of 10 bytes each
- Output is the list of numbers or words in sorted order

- *How can we use MapReduce for this?*

TeraSort: Sorting Large Files

- Approach 1
  - Have ***n*** mappers and ***1*** reducer tasks
  - **`Map(key, num) → (num, 0)`**
  - Shuffle & Local Sort: All numbers (intermediate keys) to the single reducer is sorted by framework
  - **`Reduce(num, [0]) → (num⁺)`**
  - Output from the reducer task is in sorted order
  - NOTE: repeat printing of num if there are duplicate '0'
  - *Do we have any scaling?*

TeraSort: Sorting Large Files

- Approach 2
  - ***n*** mapper, ***m*** reducer tasks
  - **`Map(key, num) → (num, 0)`**
  - Shuffle & Local Sort: All numbers (intermediate keys) to a single reducer are sorted
  - **`Reduce(num, [0]) → (num⁺)`**
  - Local output of numbers from each reducer is sorted, e.g. *m* sorted files
  - *Merge Sort separately?*
  - *What is the scaling? Balancing?*

TeraSort: Sorting Large Files

- Approach 3
  - ***n*** mapper, ***m*** reducer tasks
  - **`Map(key, num) → (num/(MAX/m), num)`**
  - Map does a histogram distribution of *num* into reduce method buckets
  - **`Reduce(bucketID, num[]) → sort(num[])`**
  - Reduce performs a local sort of all local numbers
    - Sort managed by us, needs to fit in memory, etc.
  - Concatenate output of *m* sorted files
  - *What is the scaling?*

TeraSort: Sorting Large Files

- Approach 4
  - ***n*** mapper, ***m*** reducer tasks
  - **`Map(key, num) → (num, 0)`**
  - **`Partition(num, m) → floor(num/(MAX/m))`**
  - Partitioner causes numbers to be *range-partitioned* to each reducer
    - Range of values required, 0..MAX
    - Words (string) requires a trie for efficiency
  - Shuffle & Sort: Local range of numbers to a reducer is sorted
  - **`Reduce(num, 0) → (num)`**
  - Concatenate sorted output from each reducer
  - *What is the scaling?*

MapReduce: Recap

- Programmers must specify:
  - **map** (k, v) → <k', v'>*
  - **reduce** (k', v'[]) → <k'', v''>*
    - All values with the same key are reduced together
- Optionally, also:
  - **partition** (k', number of partitions) → partition for k'
    - Often a simple hash of the key, e.g., hash(k') mod n
    - Divides up key space for parallel reduce operations
  - **combine** (k', v') → <k', v'>*
    - Mini-reducers that run in memory after the map phase
    - Used as an optimization to reduce network traffic
- The execution framework handles *everything else*…

"Everything Else"

- The execution framework handles everything else…
  - Scheduling: assigns workers to map and reduce tasks
  - "Data distribution": moves processes to data
  - Synchronization: gathers, sorts, and shuffles intermediate data
  - Errors and faults: detects worker failures and restarts
- Limited control over data and execution flow
  - All algorithms must expressed in m, r, c, p
- You don't know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

Announcements, etc.

Admin Stuff

- Add yourself to **coursereg.iisc.ac.in** if you are an IISc Student crediting/auditing the course
- Add yourself to the DS256.Jan17 mailing list
  - http://mailman.serc.iisc.in/mailman/admindb/ds256.jan17
  - All announcements, etc. will be sent to this list
  - Accounts on turing cluster will be created *only* for those on this list

Assignment 0 (by 12/Jan)

- **By Jan 12**
  - Setup IDE like Eclipse, IntelliJ
  - *Pseudo-distributed* setup of Apache Hadoop v2 on laptop/workstation
  - *Wordcount, Distributed Grep, PageRank* on local setup
  - Look into WordCount code. With and without *combiner*.
- **By Jan 17**
  - turing account details to be email by **16 Jan**
  - *Wordcount, Distributed Grep, Sort, PageRank* on **turing** cluster
  - Monitoring, Logging and Performance measurement
  - How long does *grep* and *sort* Linux commands take?
    - 1MB, 10MB, 100MB, 1GB integer files

Assignment 1 to be posted on Jan 17, due Feb 7

Reading

- **Textbook**: Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, 2010, https://lintool.github.io/MapReduceAlgorithms/
  - Chapters 1, 2, 3
- Mining of Massive Datasets, Jure Leskovec, Anand Rajaraman and Jeff Ullman, 2nd Edition (v2.1), 2014, http://www.mmds.org/#ver21
  - Chapters 1.3, 2.1-2.3, 2.5-2.6

Additional Resources

- Hadoop, HDFS & YARN
  - Hadoop: The Definitive Guide, **4th Edition**, 2015
  - Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop, 2015

MSR India Academic Research Summit: Data Science Track

- **Venue:** Satish Dhawan Auditorium, Indian Institute of Science
- 24th January 2017
- 3:30 – 5:00 PM: **Track: Data Science for Societal Impact** – *Vani Mandava, Manohar Swaminathan, Yogesh Simmhan*

https://www.microsoft.com/en-us/research/event/msr-india-academic-research-summit/