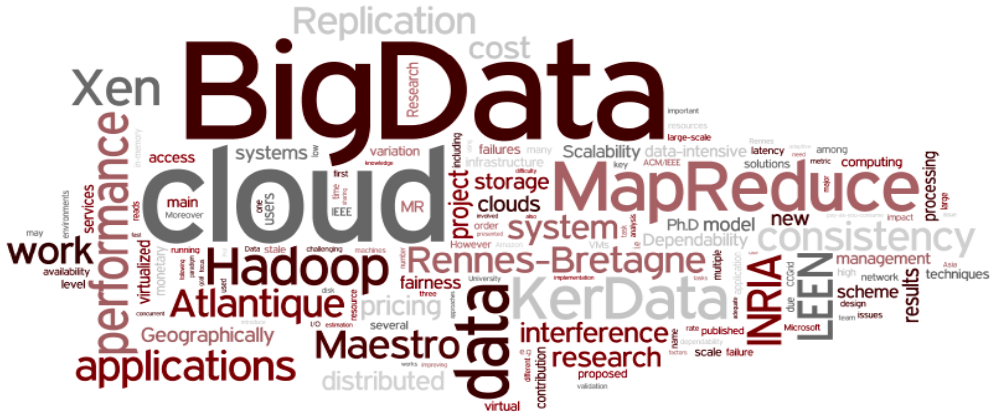
MapReduce System



Shadi Ibrahim

Inria, Rennes - Bretagne Atlantique Research Center

Turn Big Data into Big Value

Data is ONLY as useful as the decisions it enables

Turn Big Data into Big Value

Data is ONLY as useful as the decisions it enables

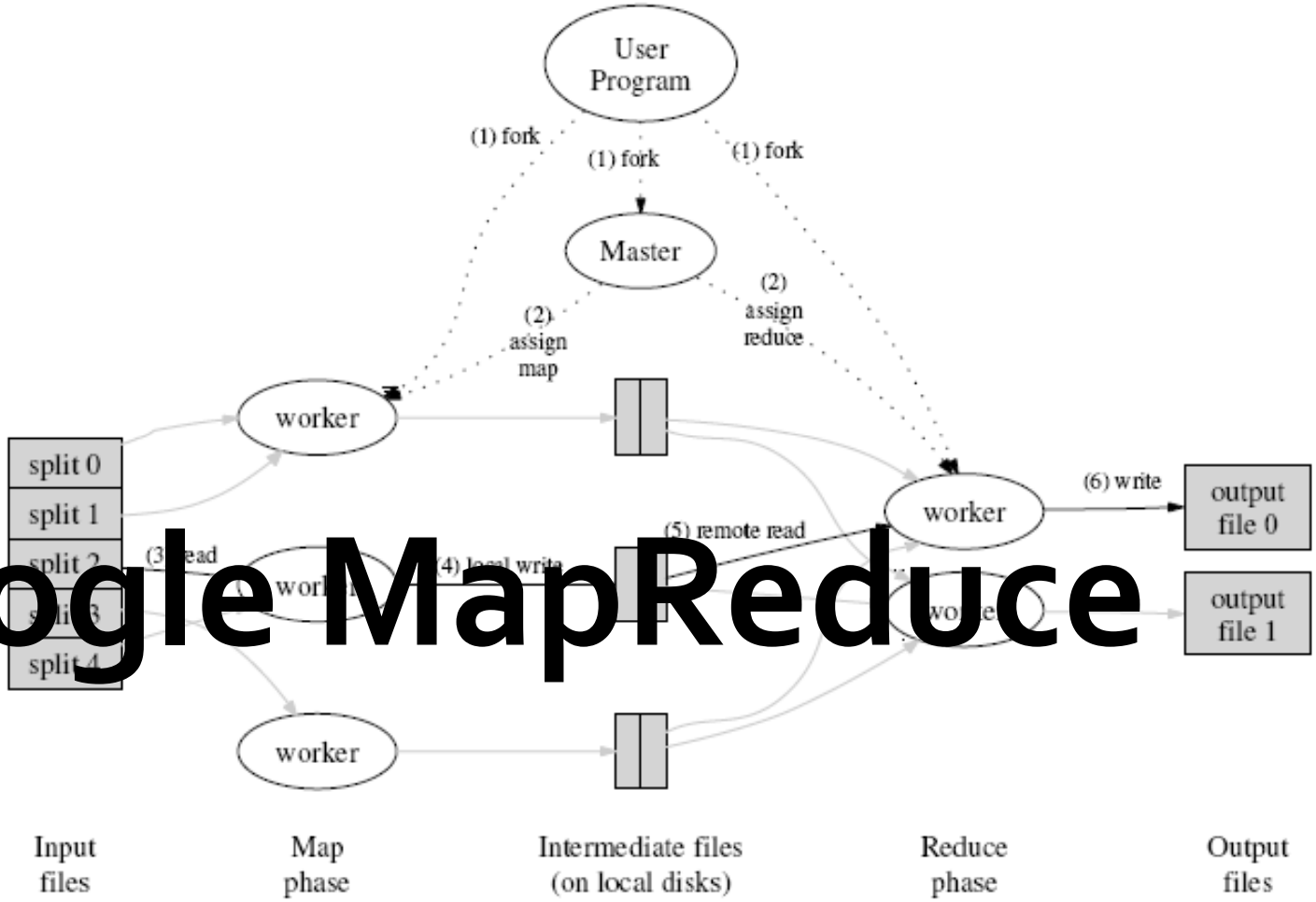
What is needed?

An efficient processing of Big Data is restricted by:

- Available computation/storage power
  - » Cloud computing: Allows users to lease computing and storage resources in a Pay-As-You-Go manner

- Programming model
  - » MapReduce: Simple yet scalable model

Google MapReduce



The Google File System (GFS)

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Motivation

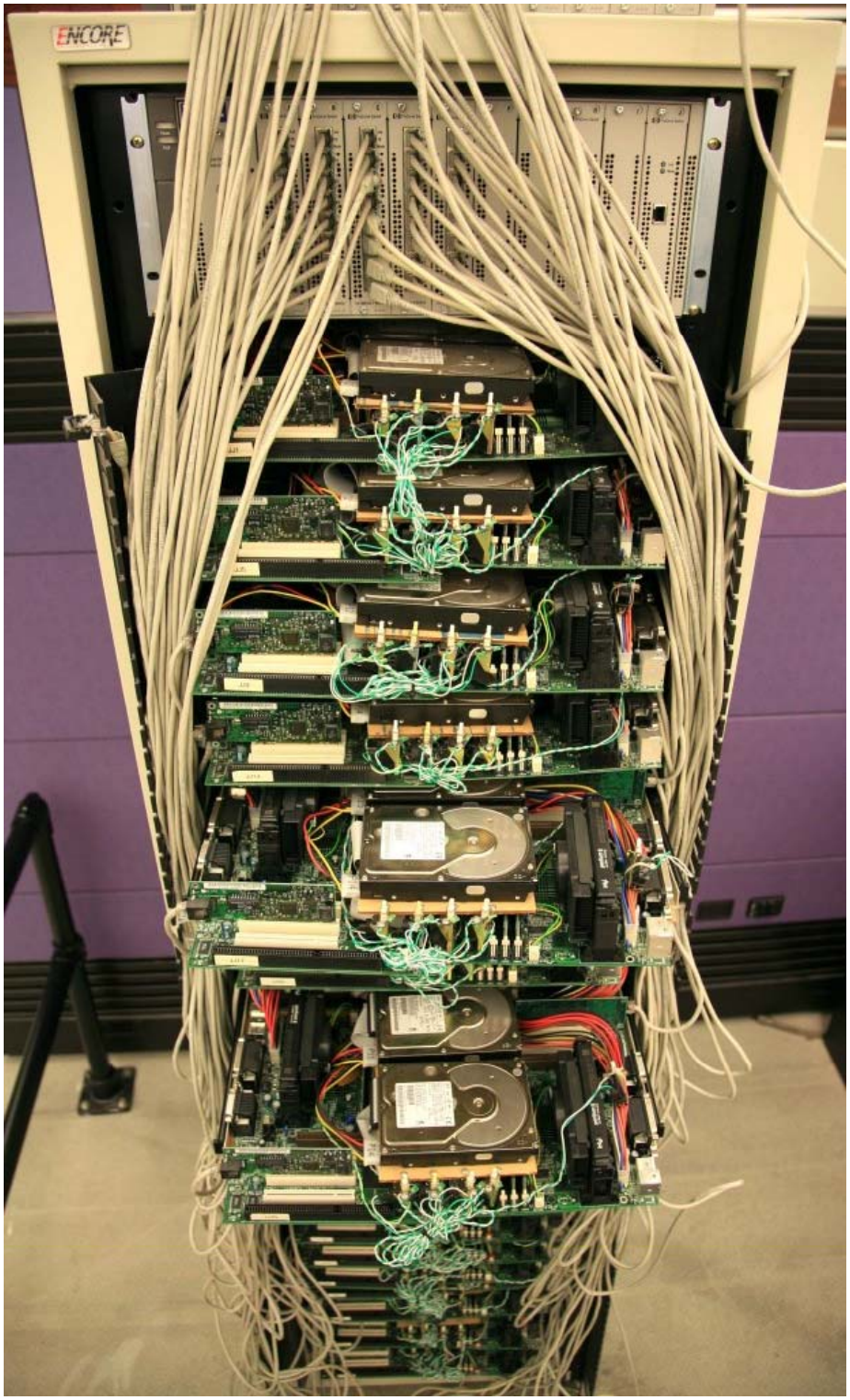
- Google needed a good distributed file system
  - » Redundant storage of massive amounts of data on cheap and unreliable computers

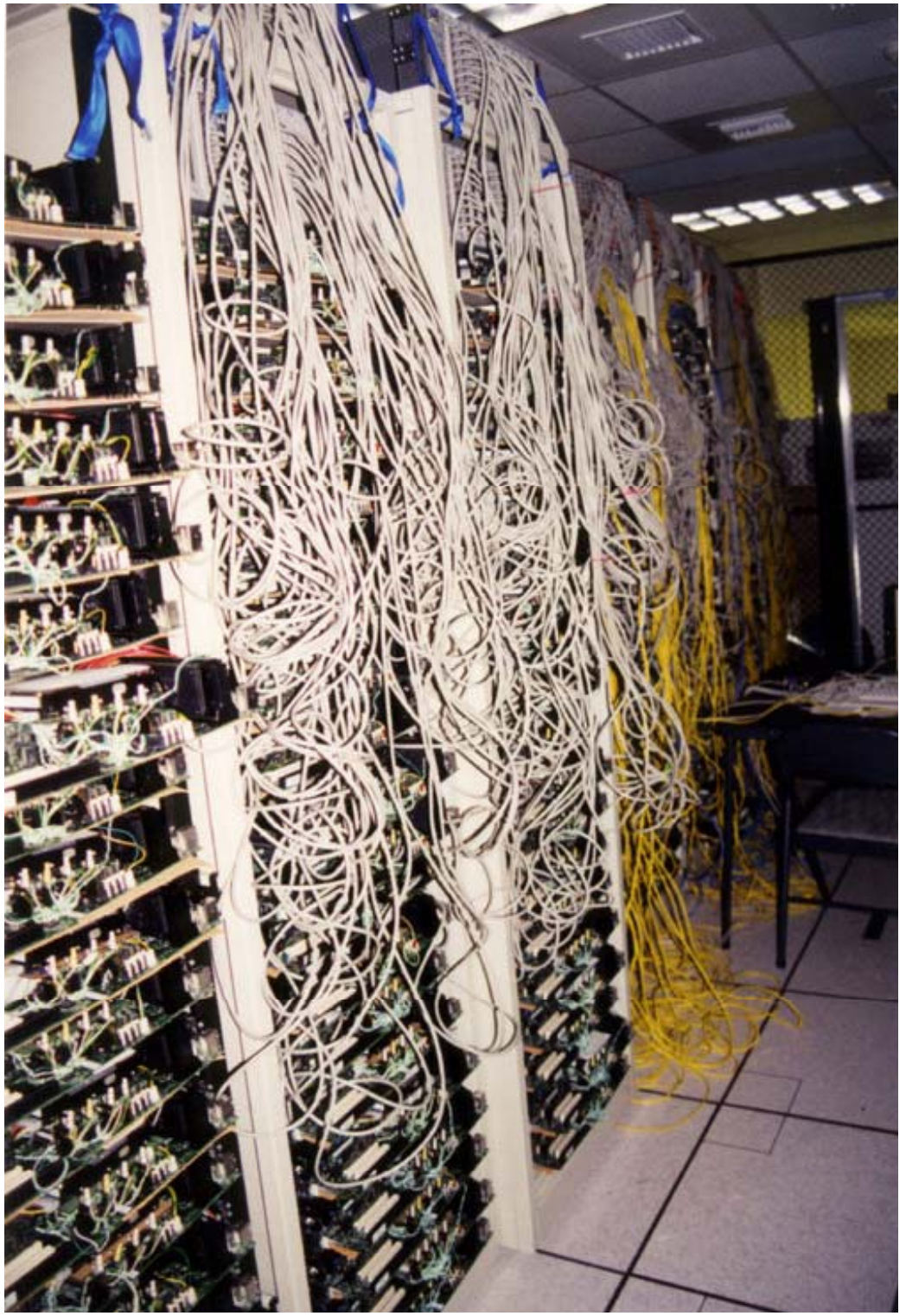
- Why not use an existing file system?
  - » Google's problems are different from anyone else's
    - Different workload and design priorities
  - » GFS is designed for Google apps and workloads
  - » Google apps are designed for GFS

Motivation

- Google needed a good distributed file system
  - » Redundant storage of massive amounts of data on cheap and unreliable computers

- Why not use an existing file system?
  - » Google's problems are different from anyone else's
    - Different workload and design priorities
  - » GFS is designed for Google apps and workloads
  - » Google apps are designed for GFS


ENCORE



Operating Environment

- Component Failure
  - » It's the norm, not the exception
  - » Expect:
    - Application, OS bugs
    - Human error
    - Hardware failure:
      - Disks
      - Memory
      - Power Supply
  - » Need for:
    - Constant monitoring
    - Error detection
    - Fault tolerance
    - Automatic Recovery

Non-standard data needs

- Files are huge
  - » Typically multi-GB
  - » Data sets can be many TB
  - » Cannot deal with typical file sizes
    - Would mean billions of KB sized files- I/O problems!
  - » Must rethink block sizes

Non-standard writing

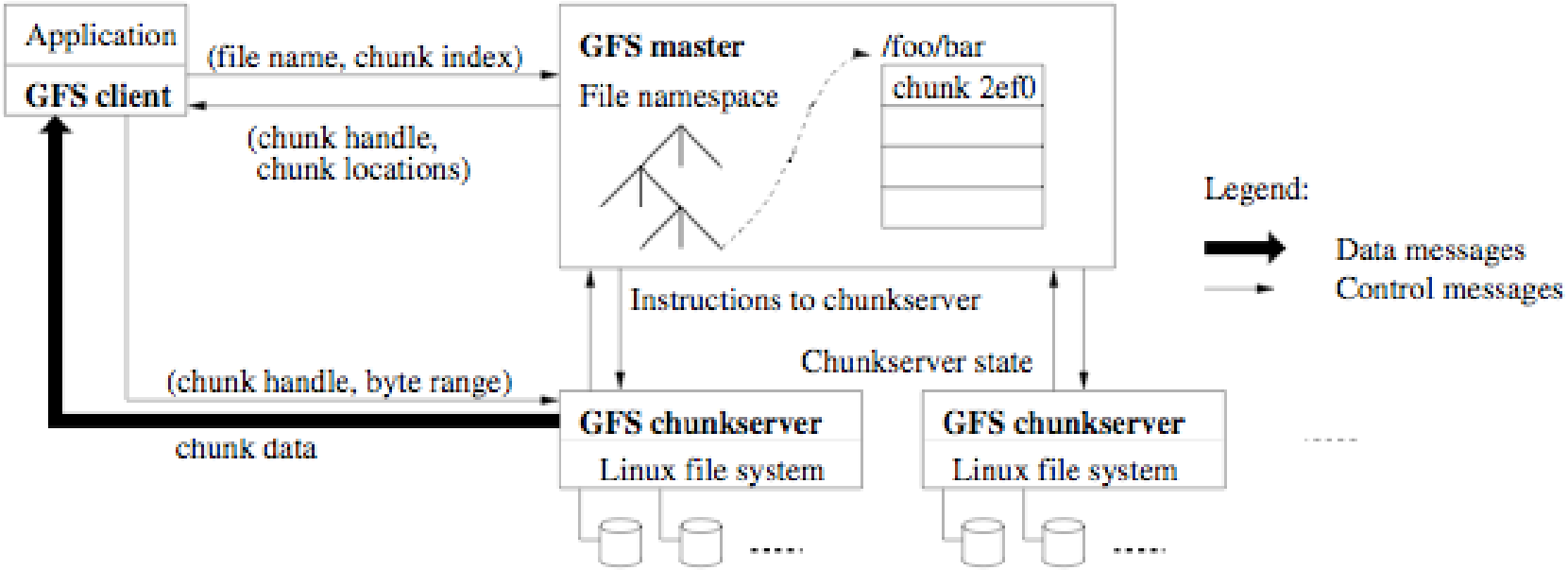
- File mutation
  - » Data is often appended, rather than overwritten
  - » Once a file is written it is read only, typically read only sequentially
  - » Optimization focuses on appending to large files and atomicity, not on caching

Design Assumptions

- Built from cheap commodity hardware
- Expect large files: 100MB to many GB
- Support large streaming reads and small random reads
- Support large, sequential file appends
- Support producer-consumer queues for many-way merging and file atomicity
- Sustain high bandwidth by writing data in bulk

GFS Architecture

- 1 *master*, at least 1 *chunkserver*, many *clients*



Master

- Responsibilities
  - » Maintain *metadata*: information like *namespace, access control info, mappings from files to chunks, chunk locations*
  - » Activity control
    - Chunk lease management
    - Garbage collection
    - Chunk migration
  - » Communication with chunks
    - *Heartbeats*: periodic syncing with chunkservers
  - » Lack of caching
    - Block size too big (64 MB)
    - Applications stream data
    - (Clients will cache metadata, though)

Chunks

- Identified by 64 bit *chunk handles*
  - » Globally unique and assigned by *master* at chunk creation time
- Triple redundancy: copies of chunks kept on multiple chunkservers
- Chunk size: 64 MB!
  - » Plain Linux file
  - » Advantages
    - Reduces R/W & communication between clients & master
    - Reduces network overhead- persistent TCP connection
    - Reduces size of metadata on master
  - » Disadvantage
    - Hotspots- small files of just one chunk that many clients need

Metadata

- 3 main types to keep in memory
  - » File and chunk namespaces
  - » Mappings from files to chunks
  - » Locations of chunk's replicas
- First two are also kept *persistently* in log files to ensure reliability and recoverability
- Chunk locations are held by chunkservers- master polls them for locations with a *heartbeat* occasionally (and on startup)

What About Consistency?

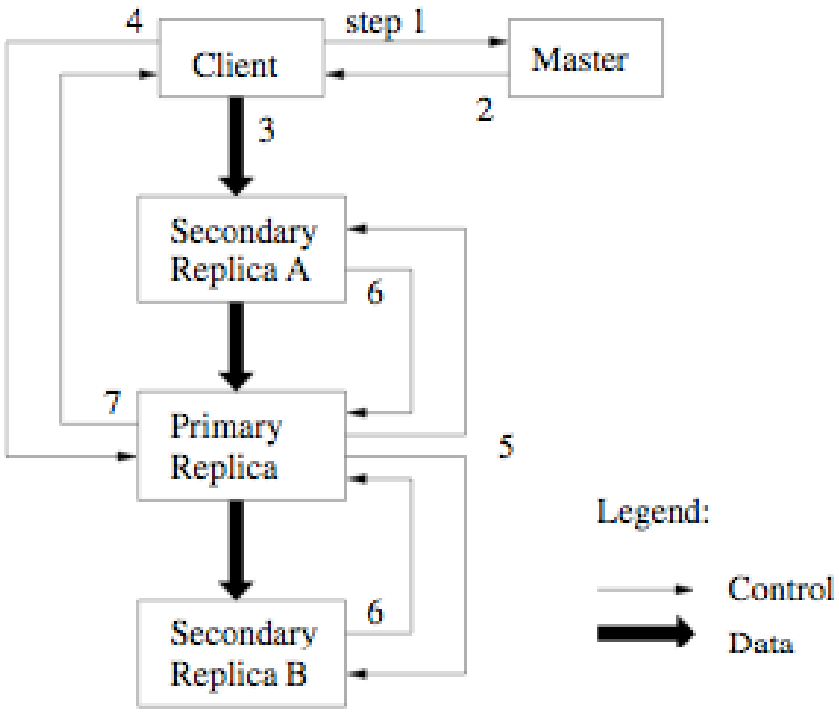
- GFS Guarantees
  - » File namespace mutations are atomic (file creation…)
- File Regions
  - » Consistent: all clients see same data, regardless which replicated chunk they use
  - » Defined: after data mutation (*writes* or *record appends*) if it is consistent & all clients will see the entire mutation effect

	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

System Operations

- Leases
  - » Mutations are done at all chunk's replicas
  - » Master defines *primary chunk* which then decides serial order to update replicas
  - » Timeout of 60 seconds- can be extended with help of heartbeats from master

Write Control & Data Flow



1. Client asks master for chunkserver
2. Master replies with primary & secondaries (client caches)
3. Client pushes data to all
4. Client sends *write* to primary
5. Primary forwards serially
6. Secordaries return completed write
7. Primary replies to client

Decoupling

- Data & flow control are separated
  - » Use network efficiently- pipeline data linearly along chain of chunkservers
  - » Goals
    - Fully utilize each machine's network bandwidth
    - Avoid network bottlenecks & high-latency
  - » Pipeline is "carefully chosen"
    - Assumes distances determined by IP addresses

Master operation

- Executes all namespace operations
- Manages chunk replicas throughout system
  - » Placement decisions
  - » Creation
  - » Replication
- Load balancing chunkserver
- Reclaim unused storage throughout system

Replica Placement Issues

- Worries
  - » Hundreds of chunkservers spread over many machine racks
  - » Hundreds of clients accessing from any rack
  - » Aggregate bandwidth of rack < aggregate bandwidth of all machines on rack
- Want to distribute replicas to maximize data
  - » Scalability
  - » Reliability
  - » Availability

Where to Put a Chunk

- Considerations
  - » It is desirable to put chunk on chunkserver with below-average disk space utilization
  - » Limit the number of recent creations on each chunkserver- avoid heavy write traffic
  - » Spread chunks across multiple racks

Re-replication & Rebalancing

- Re-replication occurs when the number of available chunk replicas falls below a user-defined limit
  - » When can this occur?
    - Chunkserver becomes unavailable
    - Corrupted data in a chunk
    - Disk error
    - Increased replication limit
- Rebalancing is done periodically by master
  - » Master examines current replica distribution and moves replicas to even disk space usage
  - » Gradual nature avoids swamping a chunkserver

Collecting Garbage

- Lazy
  - » Update master log immediately, but…
  - » Do not reclaim resources for a bit (lazy part)
  - » Chunk is first renamed, not removed
  - » Periodic scans remove renamed chunks more than a few days old (user-defined)
- Orphans
  - » Chunks that exist on chunkservers that the master has no metadata for
  - » Chunkserver can freely delete these

Lazy Garbage Pros/Cons

- Pros
  - » Simple, reliable
  - » Makes storage reclamation a background activity of master- done in batches when the master has the time (lets master focus on responding to clients, putting speed at priority)
  - » A simple safety net against accidental, irreversible deletion
- Con
  - » Delay in deletion can make it harder to fine tune storage in tight situations

Stale replica detection

- Missed mutations on a chunk make it stale
  - » *Chunk version number*- makes it possible to detect stale chunks
  - » New leases increment chunk version number
- Stale chunks are removed in garbage collection

Fault Tolerance

- Cannot trust hardware
- Goal is high availability. How?
  - » Fast recovery
  - » Replication
- Fast Recovery
  - » Master & chunkservers can restore state and restart in a matter of seconds regardless of failure type
  - » No distinction between failure types at all
- Chunk Replication
  - » Default to keep replicas on at least 3 different racks

Scalability

- Scales with available machines, subject to bandwidth
  - » Rack- and datacenter-aware locality and replica creation policies help
- Single master has limited responsibility, does not rate-limit system

Security

- … Basically none
- Relies on Google's network being private
- File permissions not mentioned in paper
  - » Individual users / applications must cooperate

Deployment in Google

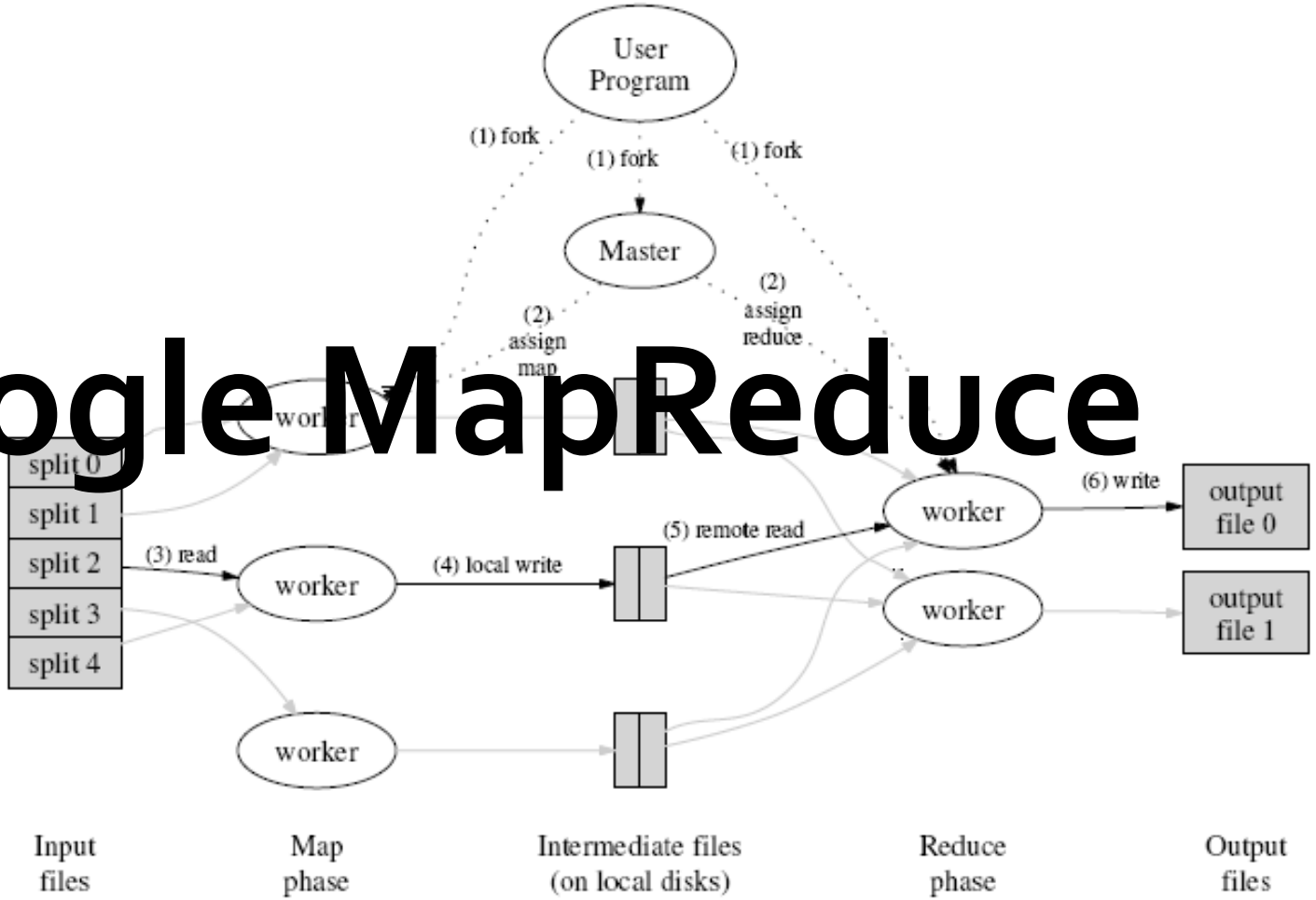
- 50+ GFS clusters
- Each with thousands of storage nodes
- Managing petabytes of data
- GFS is under BigTable, etc.

Conclusion

- GFS demonstrates how to support large-scale processing workloads on commodity hardware
  - » design to tolerate frequent component failures
  - » optimize for huge files that are mostly appended and read
  - » feel free to relax and extend FS interface as required
  - » go for simple solutions (e.g., single master)

- GFS has met Google's storage needs… it must be good!

Google MapReduce



Adapted from a Presentation by Walfredo Cirne (Google)

"Google Infrastructure for Massive Parallel Processing", Presentation in the industrial track in CCGrid'2007.

MapReduce – Motivation

- Introduced by Google in 2004
- Big Data @ Google:
  - »20+ billion web pages x 20KB = 400+ TB
- One computer can read 30-35 MB/sec from disk
  - »~4 months to read the web
  - »~1,000 hard drives just to store the web
- Even more Time/HDD, to do something with the data (e.g., data processing)

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Solution

Spread the work over many machines

Good news: "easy" parallelisation

» Reading the web with 1000 machines ⇒less than 3 hours

Bad news: programming work

» Communication and coordination
» Debugging
» Fault tolerance
» Management and monitoring
» Optimization

Worse news: repeat for every problem you want to solve

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

And the Problem Size is Ever Growing…

Every Google service sees continuing growth in computational needs

More queries

» More users, happier users

More data

» Bigger web, mailbox, blog, etc.

Better results

» Find the right information, and find it faster!

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Conclusion:Infrastructure is a Real Challenge

At Google's scale, building and running a computing infrastructure that is efficient, cost-effective, and easy-to-use is one of the most challenging technical points!

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Typical Computer

Multicore machine

- ~ 1-2 TB of disk
- ~ 4GB-16GB of RAM

Typical machine runs:

- Google File System (GFS)
- Scheduler daemon for starting user tasks
- One or many user tasks

Tens of thousands of such machines

Problem : *What programming model to use as a basis for scalable parallel processing ?*

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

What Is Needed?

A simple programming model that applies to many data-intensive computing problems

Approach: hide messy details in a runtime library:

- » Automatic parallelization
- » Load balancing
- » Network and disk transfer optimization
- » Handling of machine failures
- » Robustness
- » Improvements to core library benefit all users of library!

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Such a Model is… MapReduce!

Typical problem solved by MapReduce

- » Read a lot of data
- » Map: extract something you care about from each record
- » Shuffle and Sort
- » Reduce: aggregate, summarize, filter, or transform
- » Write the results

Outline stays the same, map and reduce change to fit the problem

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

More Specifically…

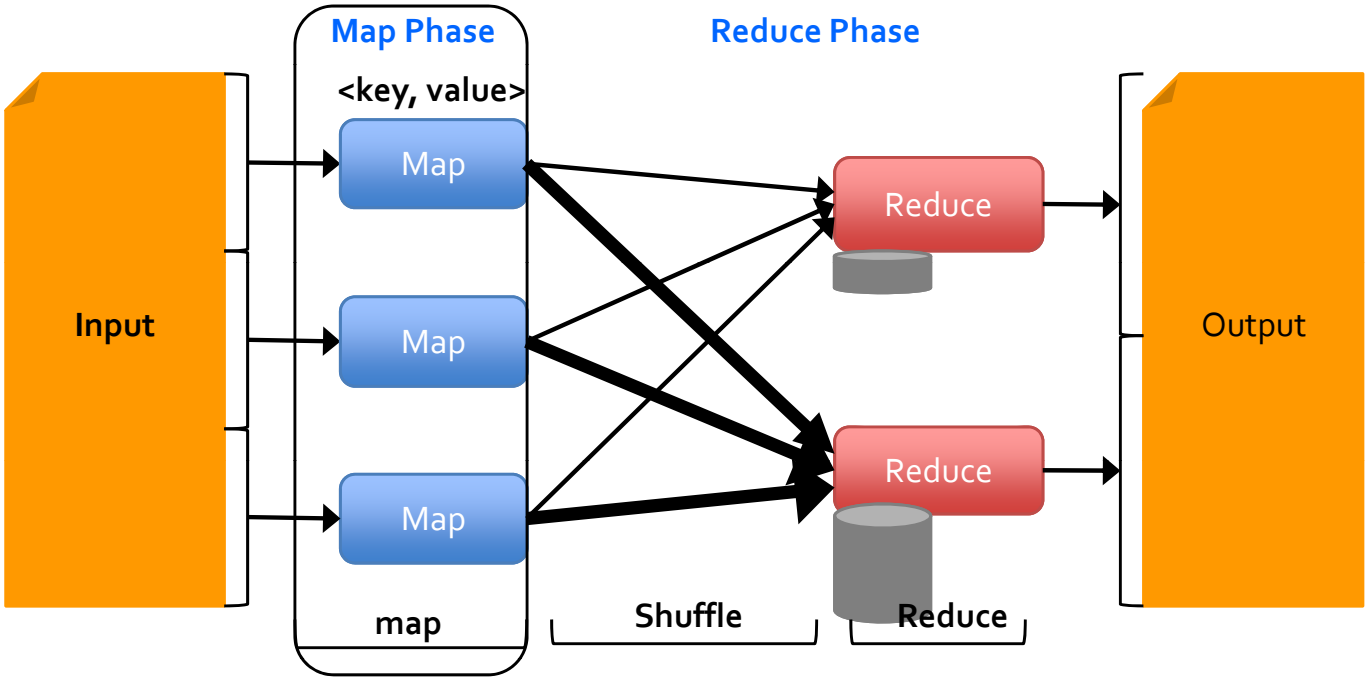
- It is inspired by the Map and Reduce functions (i.e., It borrows from functional programming)

- Users implement interface of two primary functions
  - » map(k, v) → <k', v'>*
  - » reduce(k', <v'>*) → <k', v''>*

All v' with same k' are reduced together, and processed in v' order

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

More Specifically…

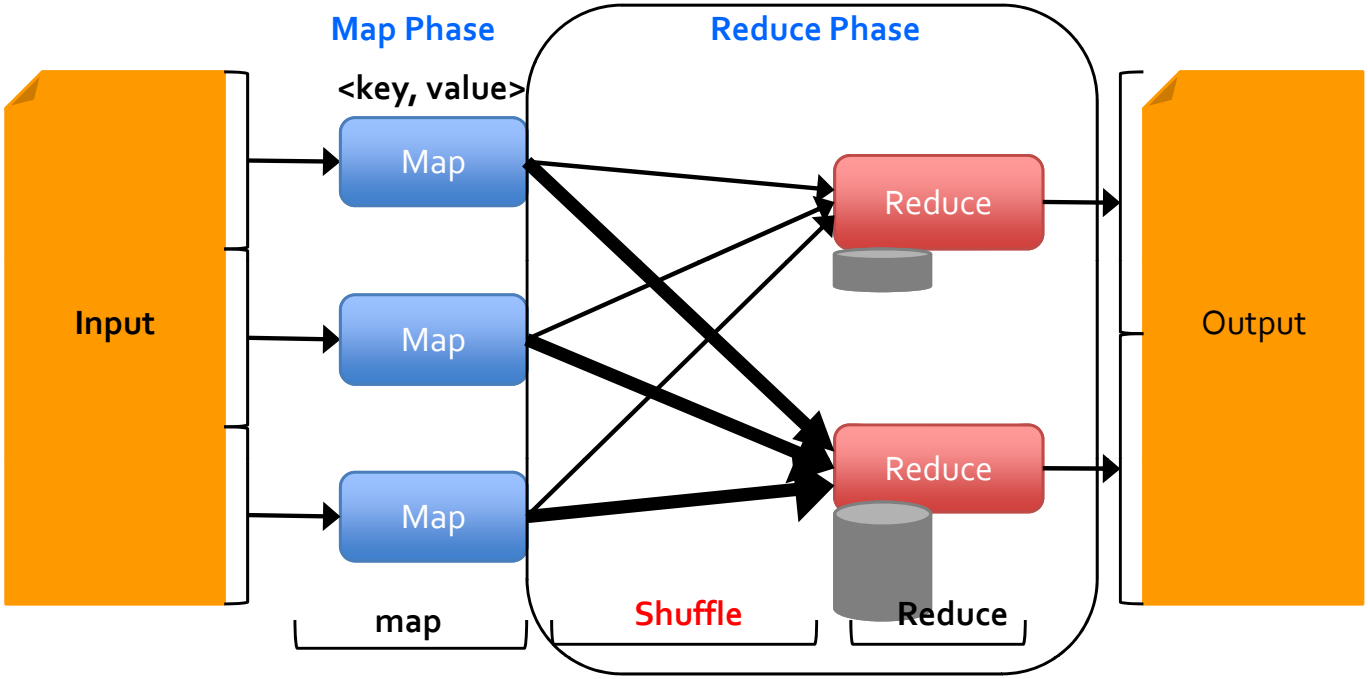


map(k, v) → <k', v'>*

Records from the data source (lines out of files, rows of a database, etc) are fed into the map function as key*value pairs: e.g., (filename, line)

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

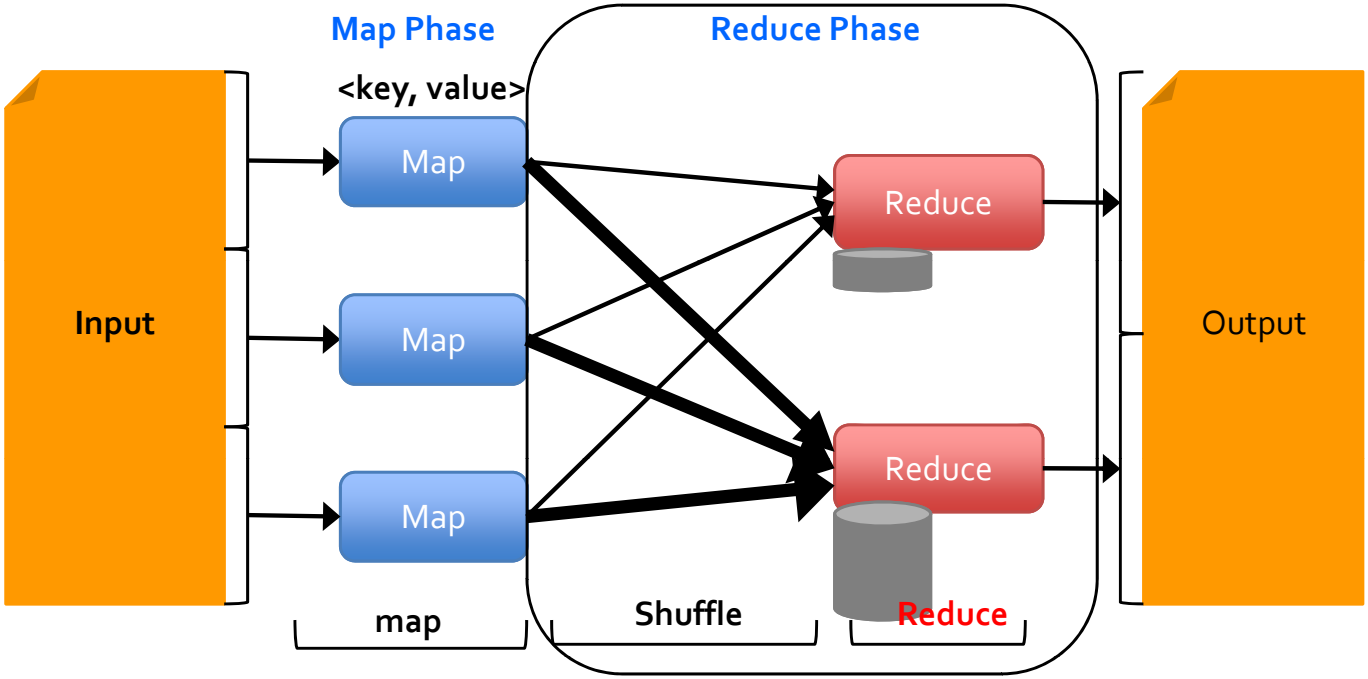
More Specifically…



After the map phase is over, all the intermediate values for a given output key are combined together into a list

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

More Specifically…



reduce(k', <v'>*) → <k', v''>*

reduce() combines those intermediate values into one or more final values per key (usually only one)

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

MapReduce:Scheduling

One master, many workers

- » Input data split into M map tasks (typically 64 MB in size)
- » Reduce phase partitioned into R reduce tasks
- » Tasks are assigned to workers dynamically
- » Per worker input size = GFS chunk size!
- » Often: M=200,000; R=4,000; workers=2,000

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

MapReduce:Scheduling

Master assigns each map task to a free worker

» Considers locality of data to worker when assigning task
» Worker reads task input (often from local disk!)
» Worker produces R local files containing intermediate k/v pairs

Master assigns each reduce task to a free worker

» Worker reads intermediate k/v pairs from map workers
» Worker sorts & applies user's Reduce op to produce the output

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

MapReduce:Features

- Exploit Parallelization:
  - »Tasks are executed in Parallel
- Fault tolerance
  - »Re-execute only the tasks on Failed machines
- Exploit data locality
  - »Co-locate data and computation power therefore avoid network bottleneck

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Parallelism

map() functions run in parallel, creating different intermediate values from different input data sets

reduce() functions also run in parallel, each working on a different output key

All values are processed independently

Bottleneck: reduce phase can't start until map phase is completely finished.

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Fault Tolerance

On worker failure

» Detects failure via periodic heartbeats
» Re-executes completed & in-progress map() tasks
» Re-executes in-progress reduce() tasks

Master notices particular input key/values that cause crashes in map(), and skips those values on re-execution.

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Backup Task

Slow workers significantly lengthen completion time

- »Other jobs consuming resources on machine
- »Bad disks with soft errors transfer data very slowly
- »Weird things: processor caches disabled (!!)

Solution: Near end of phase, spawn backup copies of tasks – Whichever one finishes first "wins"

Effect: Dramatically shortens job completion time

Locality optimization

Master scheduling policy:

»Asks GFS for locations of replicas of input file blocks

»Map tasks typically split into 64MB (== GFS block size)

»Map tasks scheduled so GFS input block replica are on same machine or same rack

Effect: Thousands of machines read input at local disk speed

- Without this, rack switches limit read rate

Actual Google MapReduce

Example is written in pseudo-code

Actual implementation is in C++, using a MapReduce library

Bindings for Python and Java exist via interfaces

True code is somewhat more involved (defines how the input key/values are divided up and accessed, etc.)

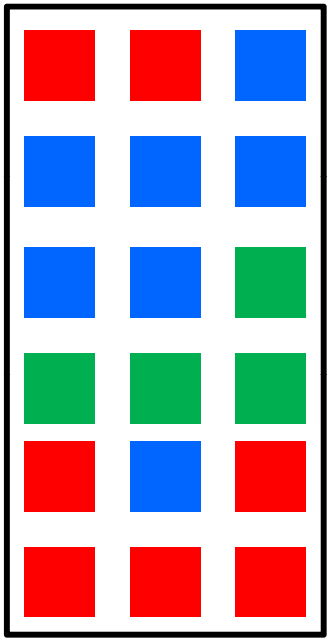
"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

Widely Applicable at Google

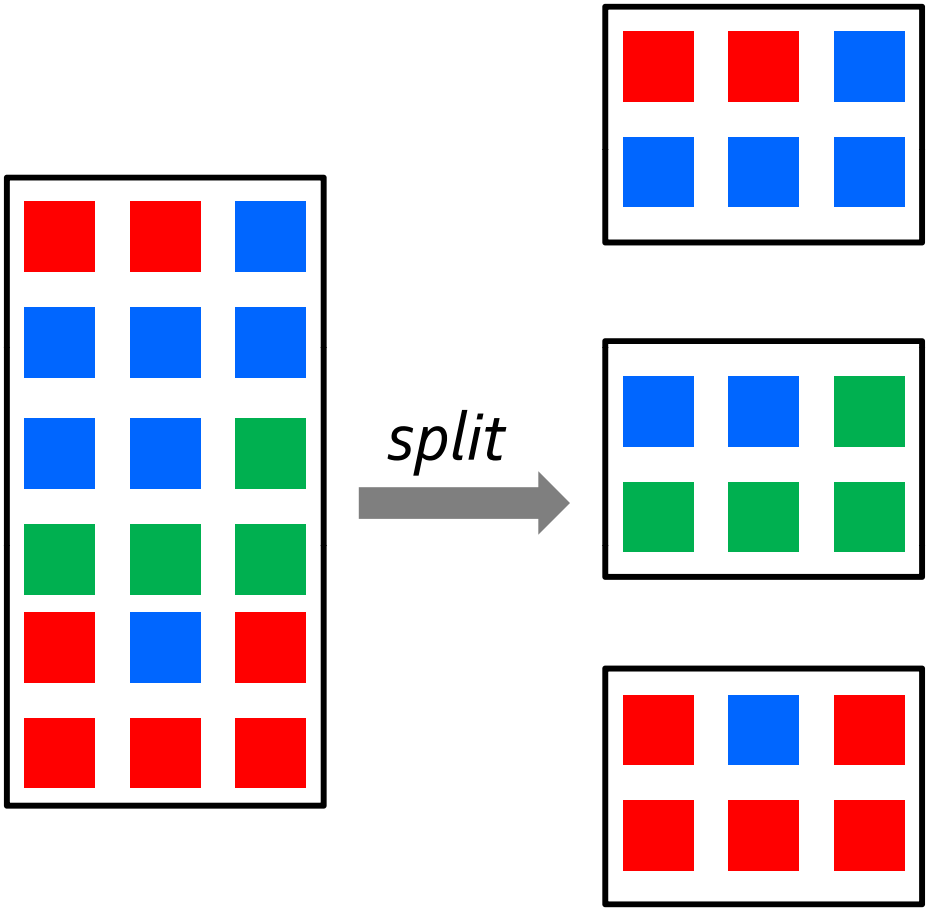
- distributed grep
- distributed sort
- term-vector per host
- document clustering
- machine learning
- …
- web access log stats
- web link-graph reversal
- inverted index construction
- statistical machine translation
- …

"Google Infrastructure for Massive Parallel Processing", Walfredo Cirne, Presentation in the industrial track in CCGrid'2007.

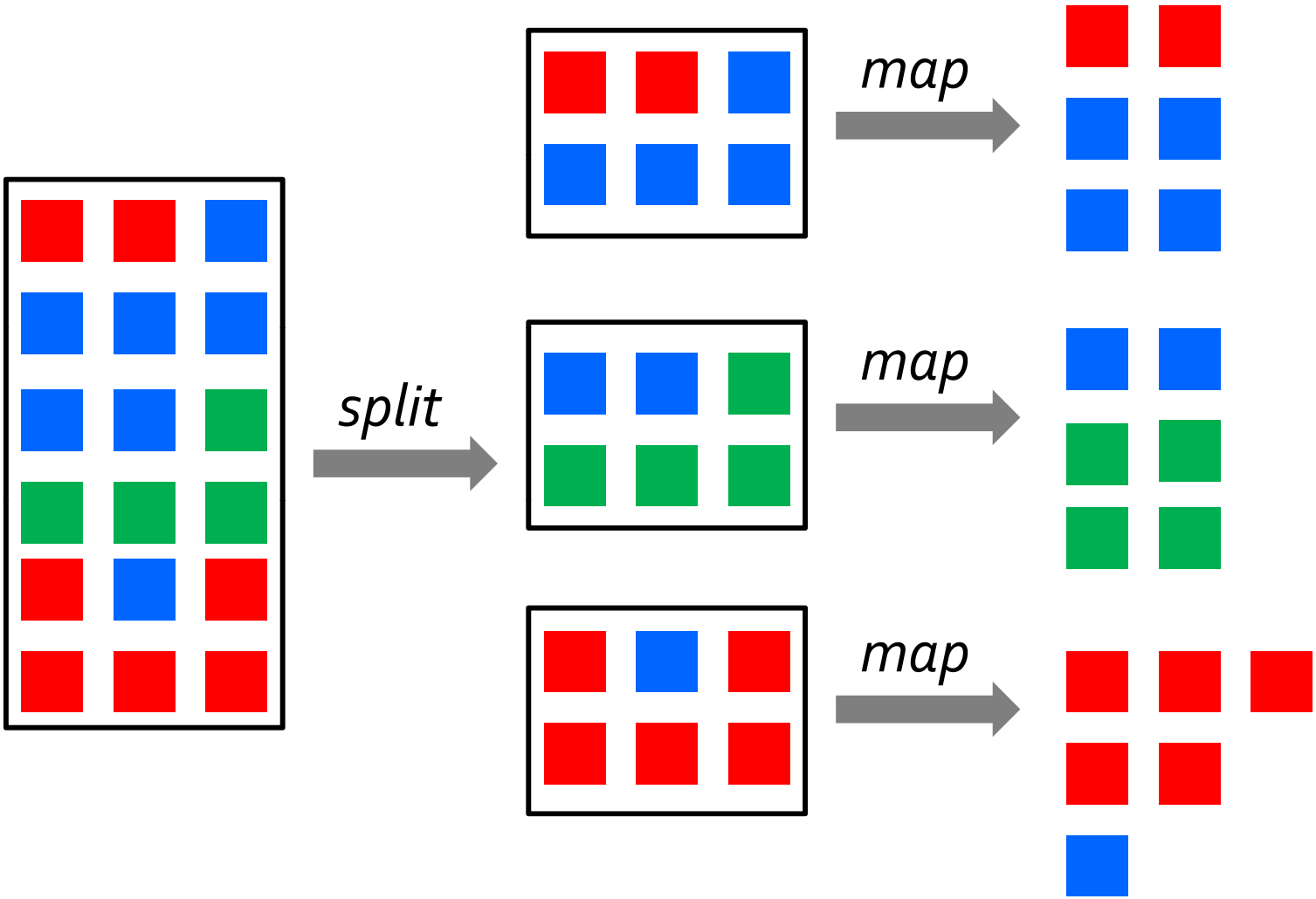
The Colored Squares Counter



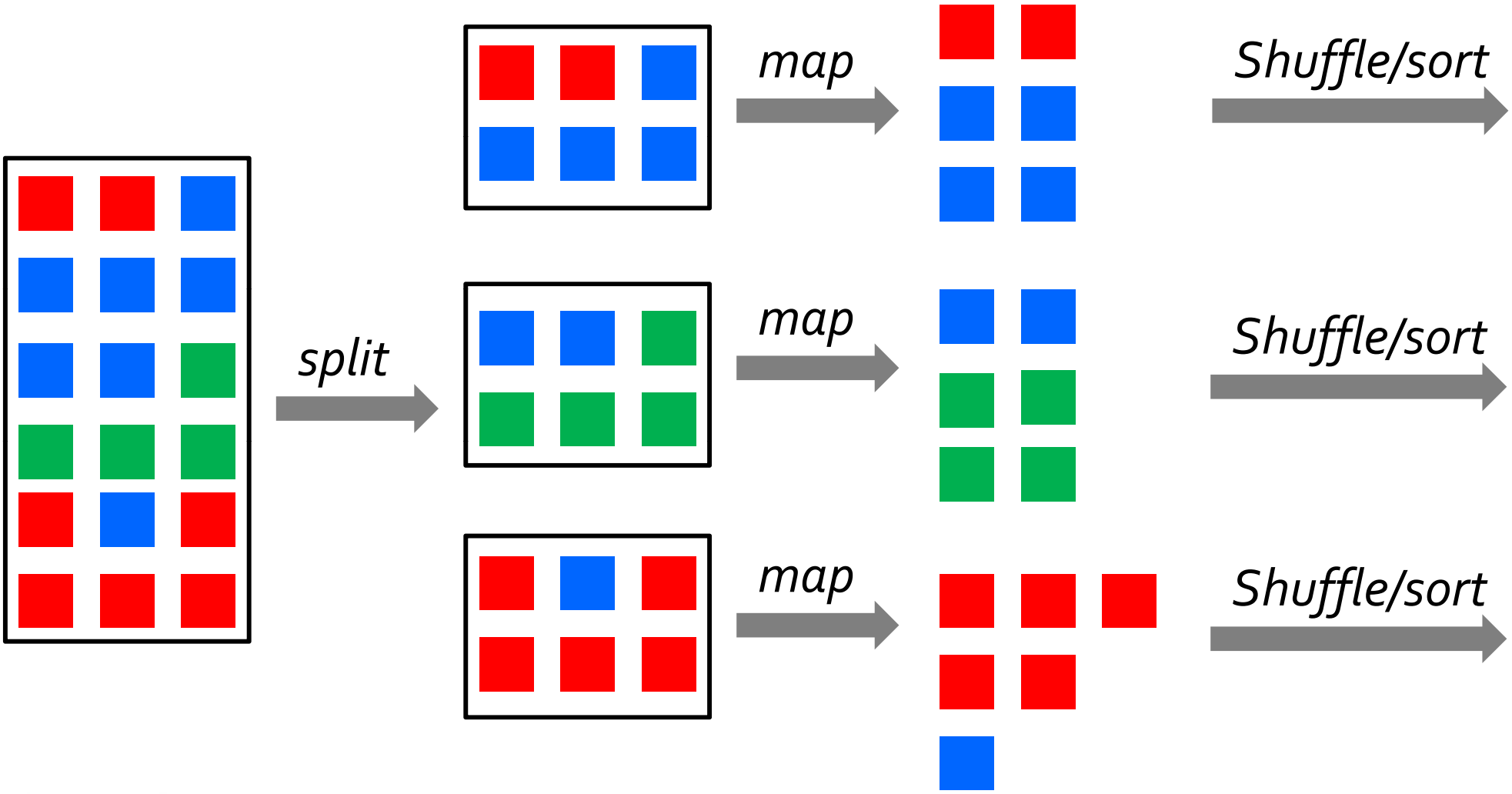
The Colored Squares Counter



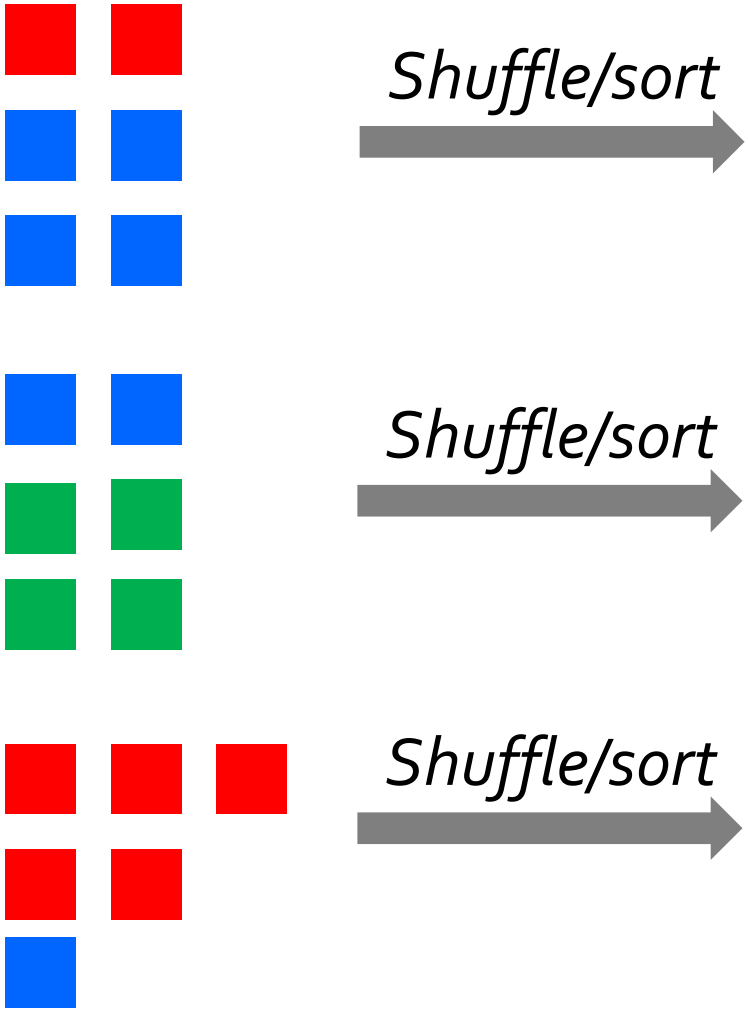
The Colored Squares Counter



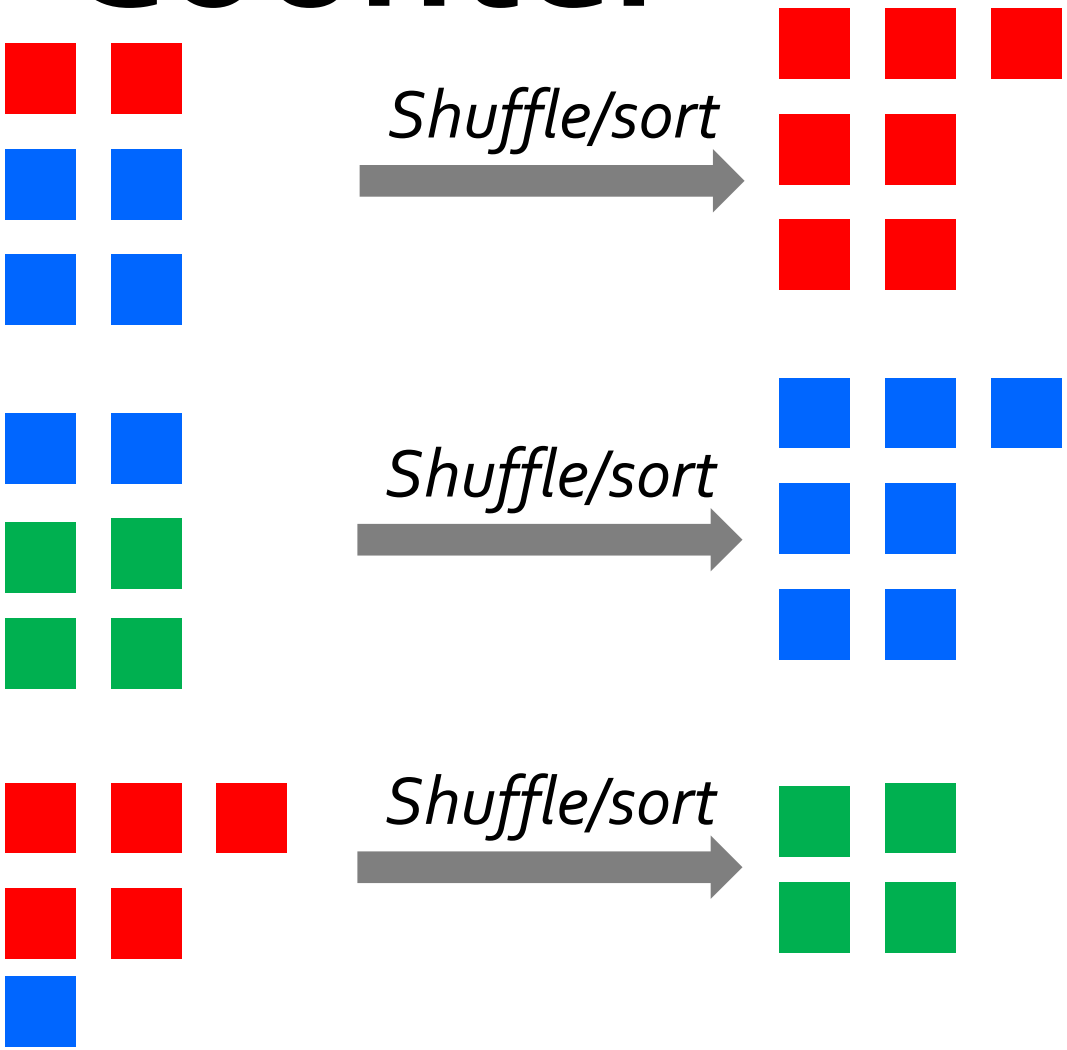
The Colored Squares Counter



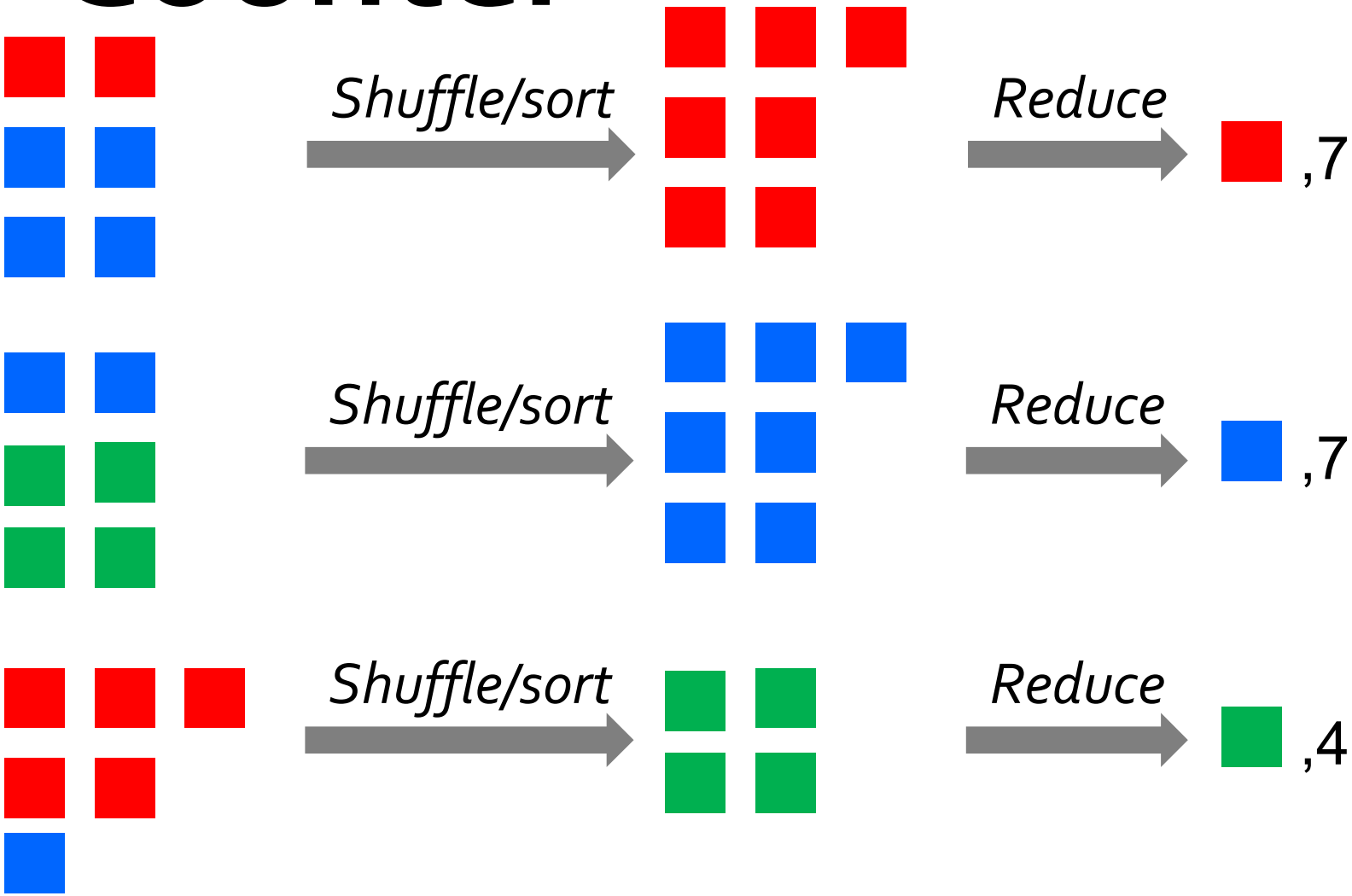
The Colored Squares Counter



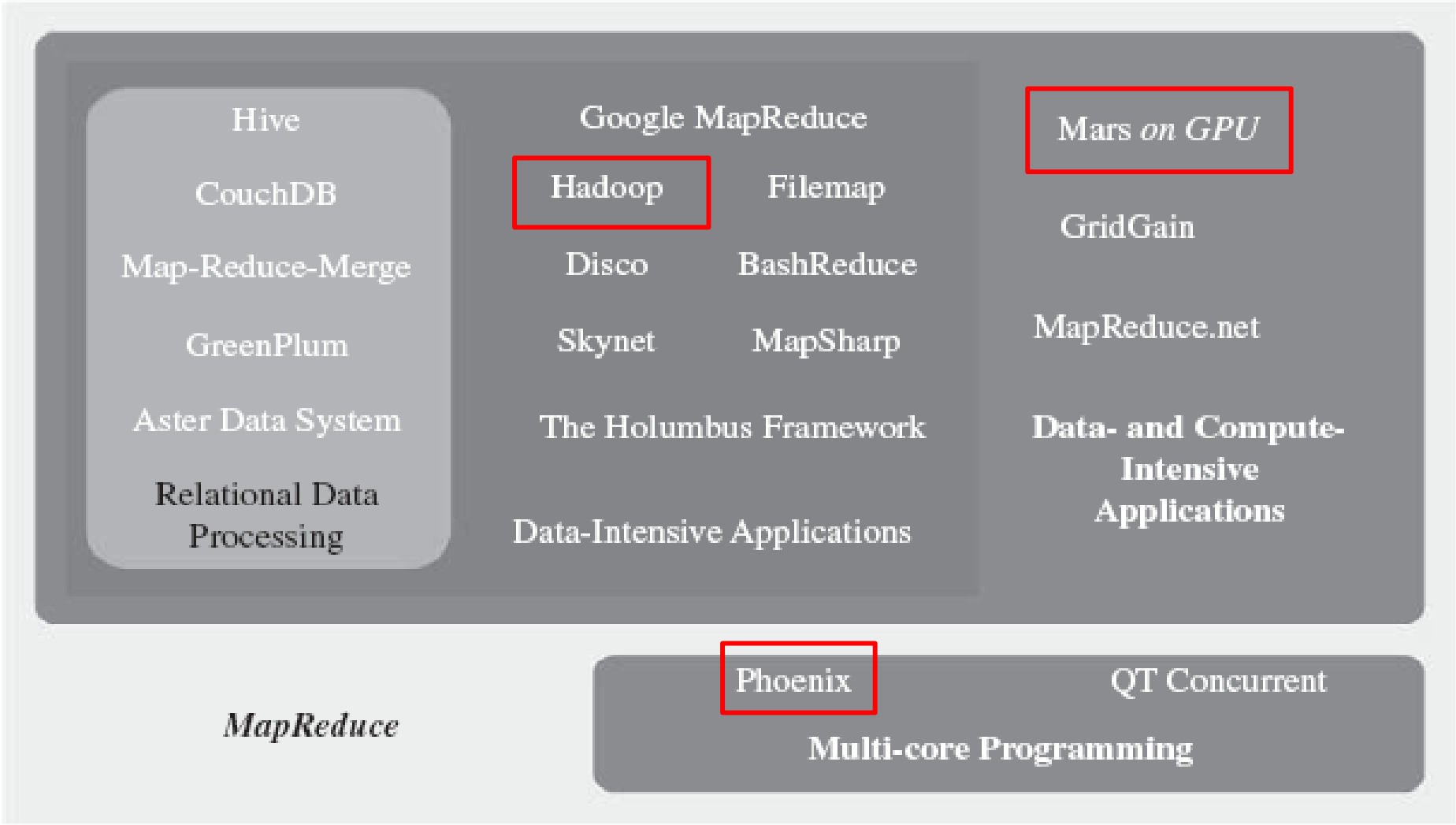
The Colored Squares Counter



The Colored Squares Counter



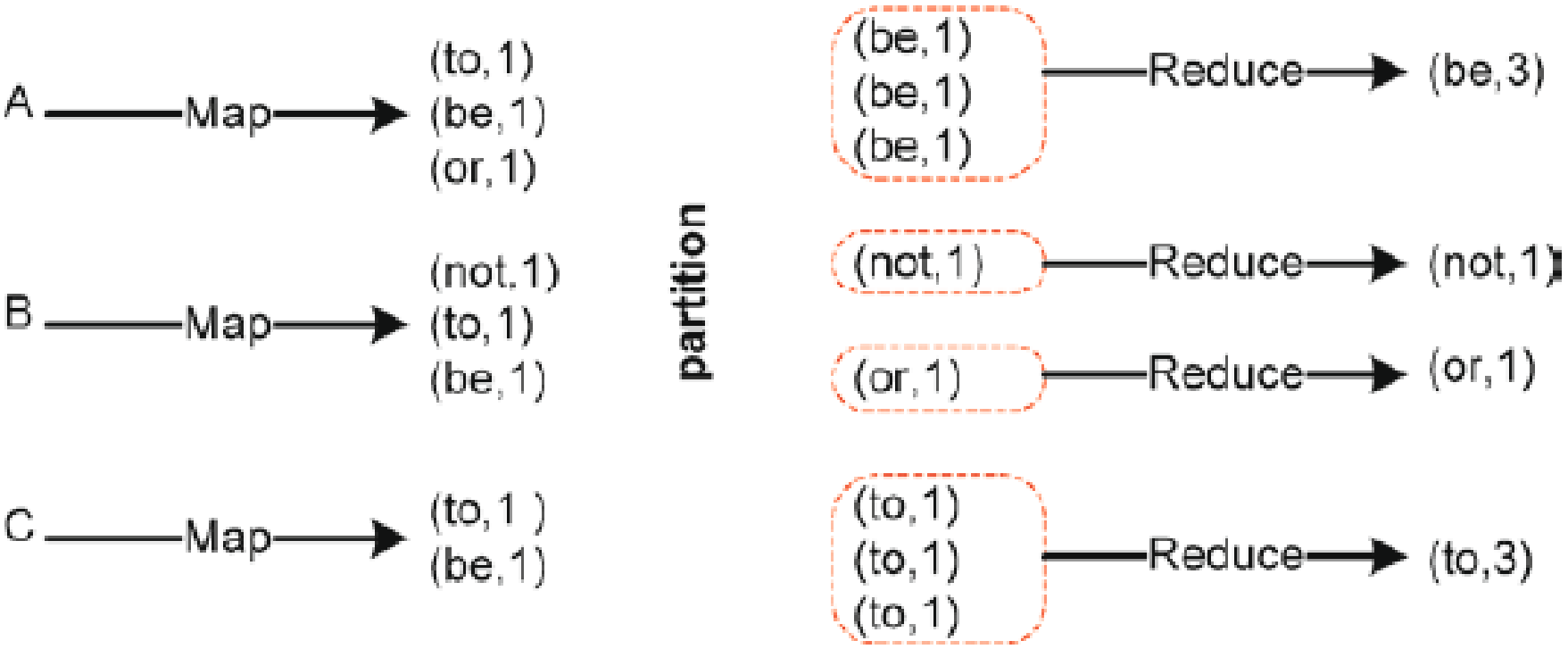
MapReduce: Implementations



Word Count Example

Count the appearance of each word in a set of documents

(A.txt = to be or) (B.txt = not to be) (C.txt= to be)



Example 2: word length count

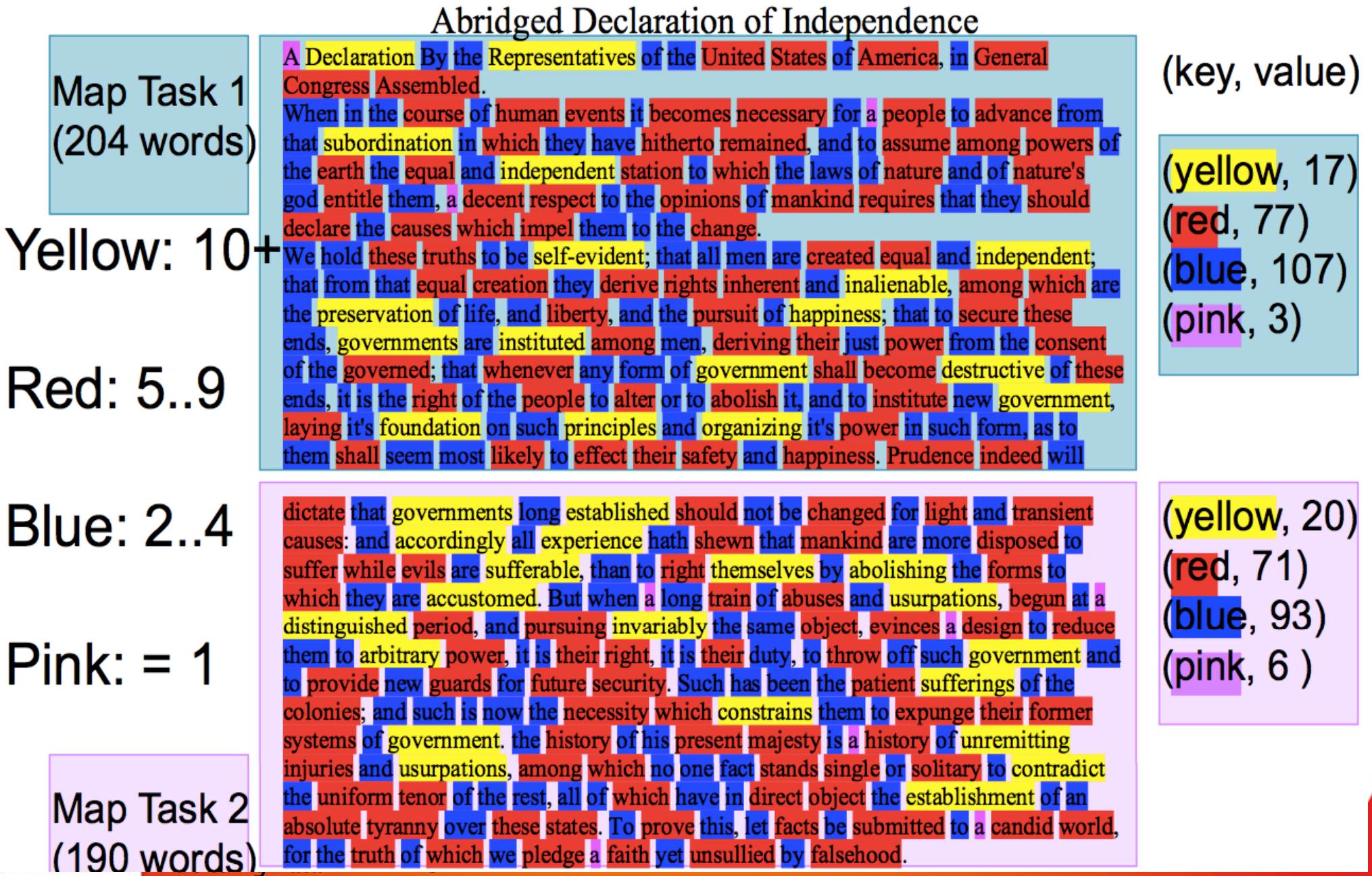
Abridged Declaration of Independence

A Declaration By the Representatives of the United States of America, in General Congress Assembled.

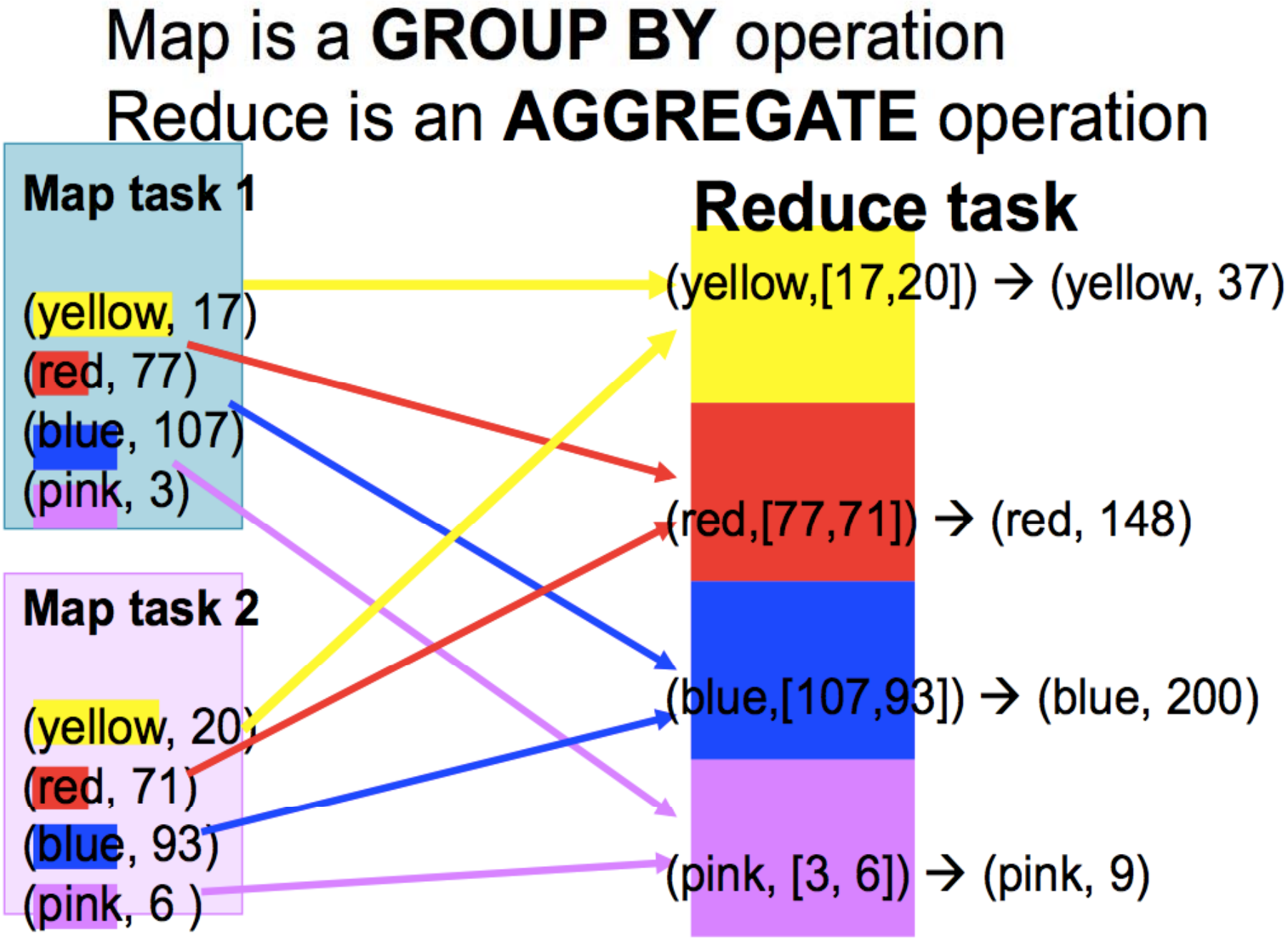
When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change.

We hold these truths to be self-evident; that all men are created equal and independent; that from that equal creation they derive rights inherent and inalienable, among which are the preservation of life, and liberty, and the pursuit of happiness; that to secure these ends, governments are instituted among men, deriving their just power from the consent of the governed; that whenever any form of government shall become destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying it's foundation on such principles and organizing it's power in such form, as to them shall seem most likely to effect their safety and happiness. Prudence indeed will dictate that governments long established should not be changed for light and transient causes: and accordingly all experience hath shewn that mankind are more disposed to suffer while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, begun at a distinguished period, and pursuing invariably the same object, evinces a design to reduce them to arbitrary power, it is their right, it is their duty, to throw off such government and to provide new guards for future security. Such has been the patient sufferings of the colonies; and such is now the necessity which constrains them to expunge their former systems of government. the history of his present majesty is a history of unremitting injuries and usurpations, among which no one fact stands single or solitary to contradict the uniform tenor of the rest, all of which have in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world, for the truth of which we pledge a faith yet unsullied by falsehood.

Example 2: word length count



Example 2: word length count



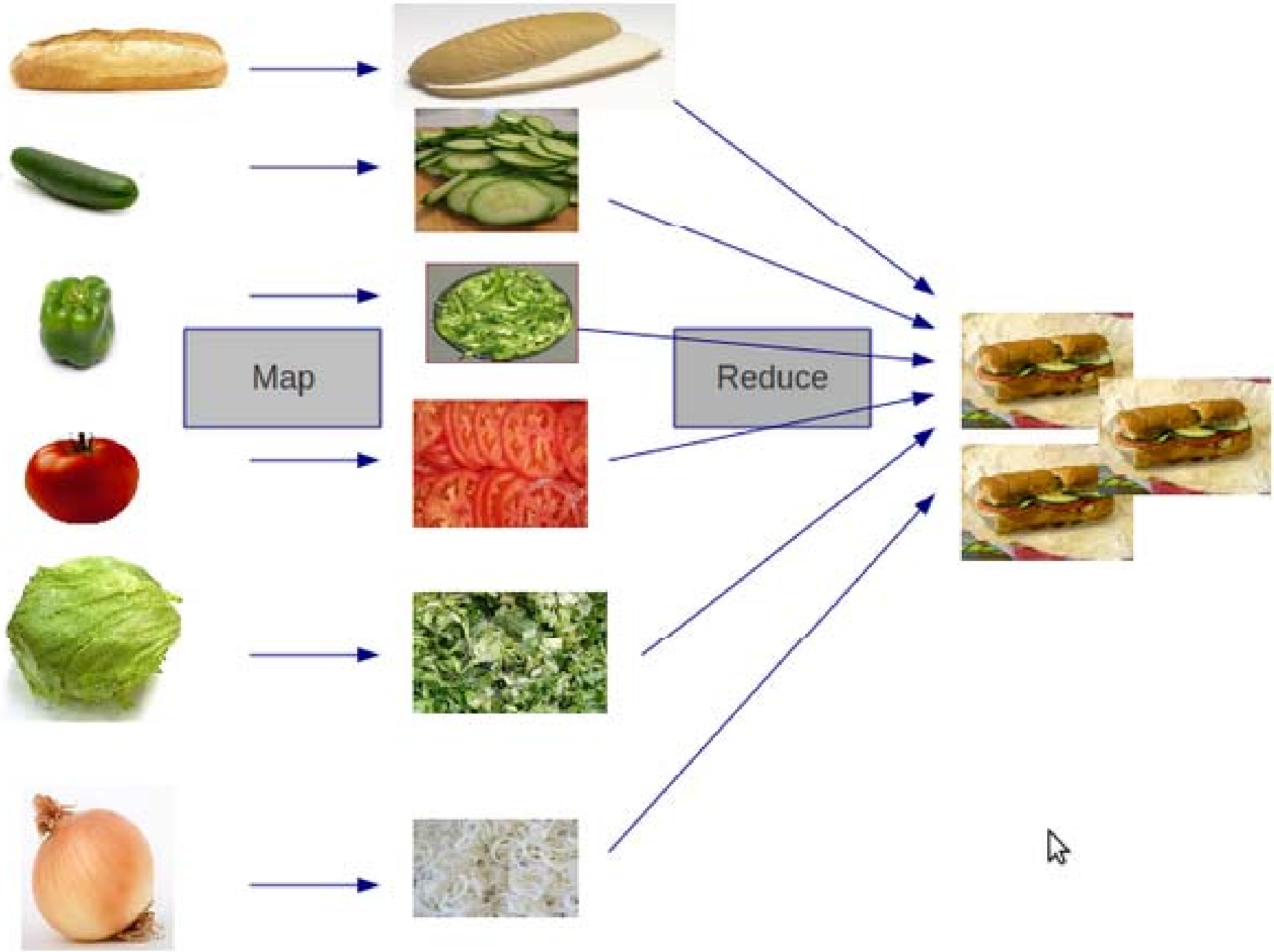
Thank you!

Inria
INVENTORS FOR THE DIGITAL WORLD

KerData

Shadi Ibrahim

shadi.ibrahim@inria.fr


Map
Reduce