



# Distributed Computing at Google

Pim van Pelt <[pim@google.com](mailto:pim@google.com)>

Open Source Days - Copenhagen, DK  
Saturday March 6 2010



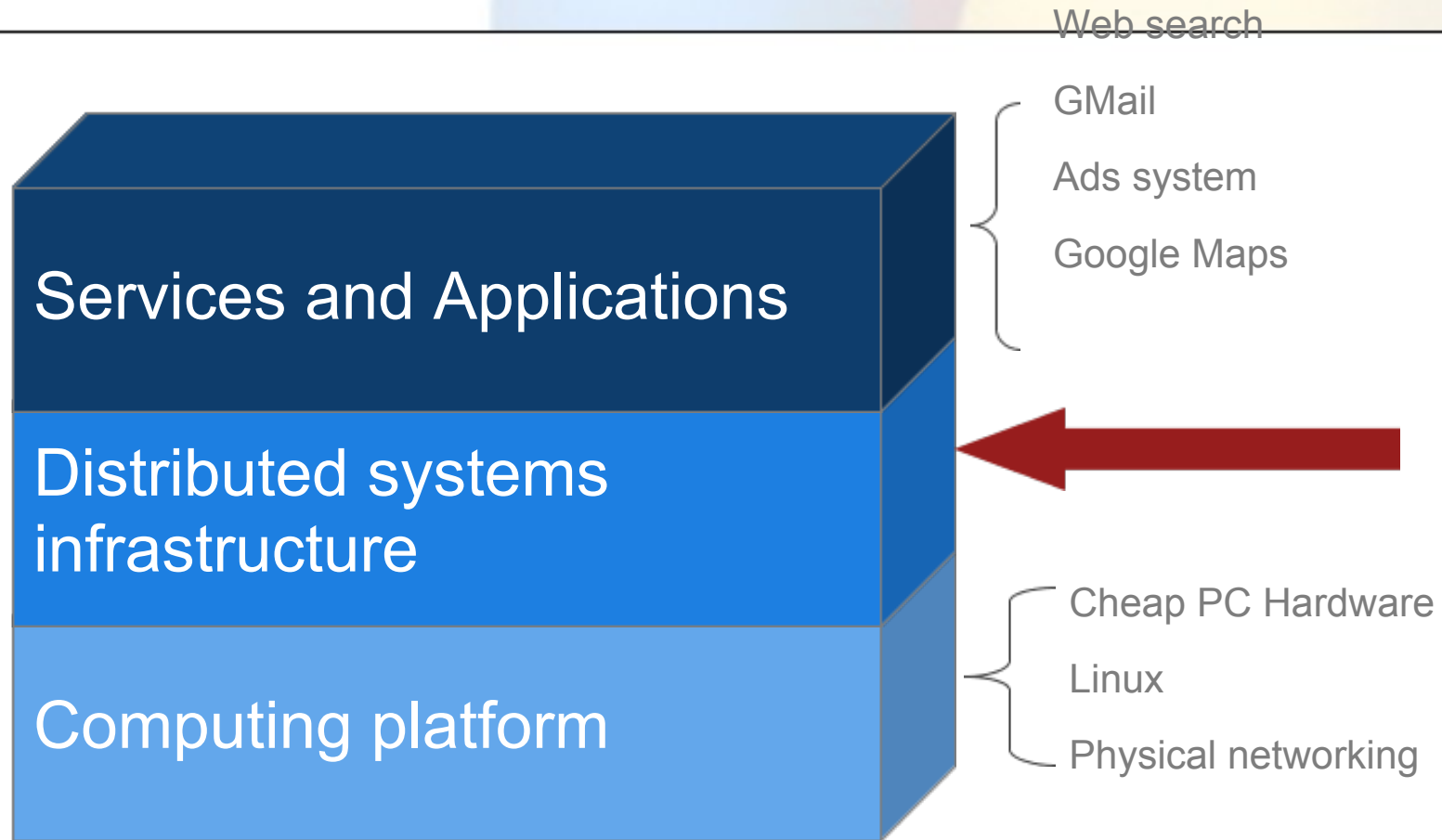
# Guidelines for this talk

---

- I am not presenting my own personal work. I did not invent the Google infrastructure components I will describe.
- Requests for clarification or quick questions welcomed throughout, please save longer questions until the end.
- I won't tell you how many machines Google currently uses, how many queries we answer and how many megabits we generate.



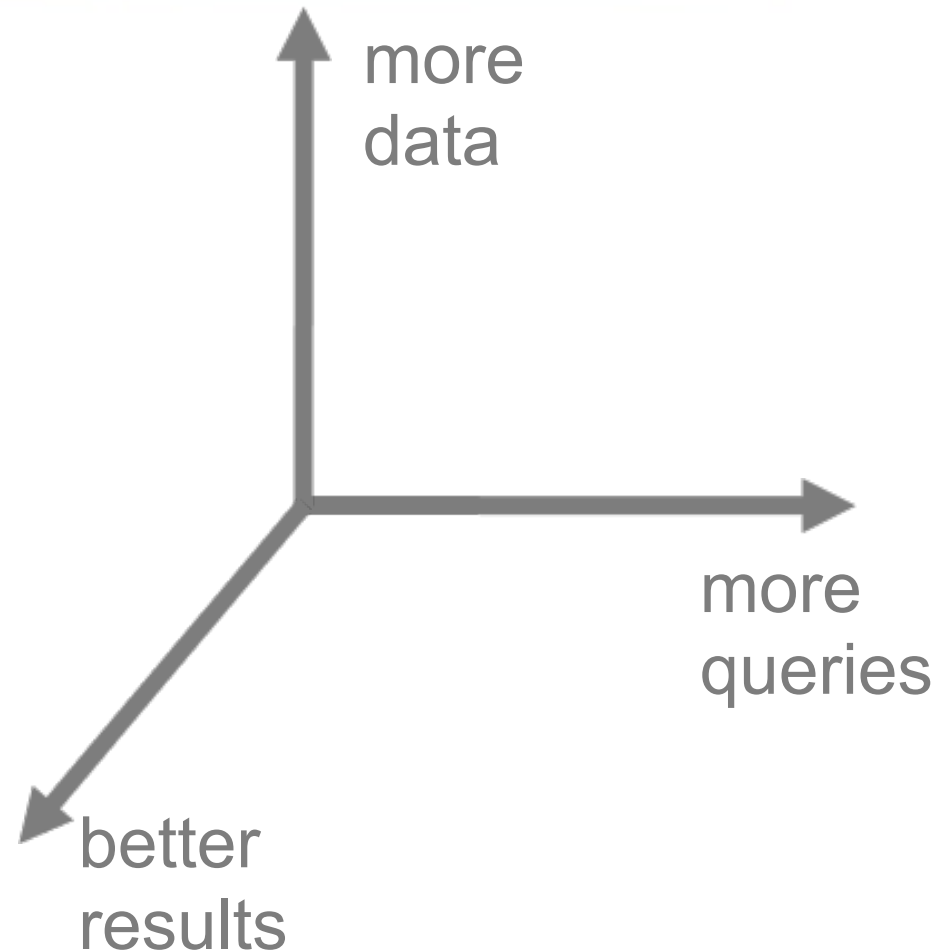
# Google Technology Layers



# Google's Explosive Computational Requirements

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
- Affordable operations
  - Get the most of disk, ram, cpu and network



# A Simple Challenge For Compute Infrastructure

---

- Create the world's largest computing infrastructure
- Make it easier for developers to write parallel code than not to
- Need to drive efficiency of the computing infrastructure to unprecedented levels



# The Building Blocks of google.com

---

- Distributed lockserver: chubby.
- Distributed storage: GFS, BigTable.
- The workqueue.
- Parallel computation: MapReduce, Sawzall.



# Chubby – A Distributed lockservice

---

- Exports a filesystem interface, similar to unix
- Directory based approach: /ls/foo/wombat/pouch
- Generic operations: create, read, write, lock
- Callbacks on data stored in Chubby:
  - File contents changed
  - Child node added, removed, changed
  - Lock expired, lock competition, ...
  - Chubby master failure

See the OSDI 2006 paper on:

<http://research.google.com/archive/chubby-osdi06.pdf>



# GFS: The Google File System

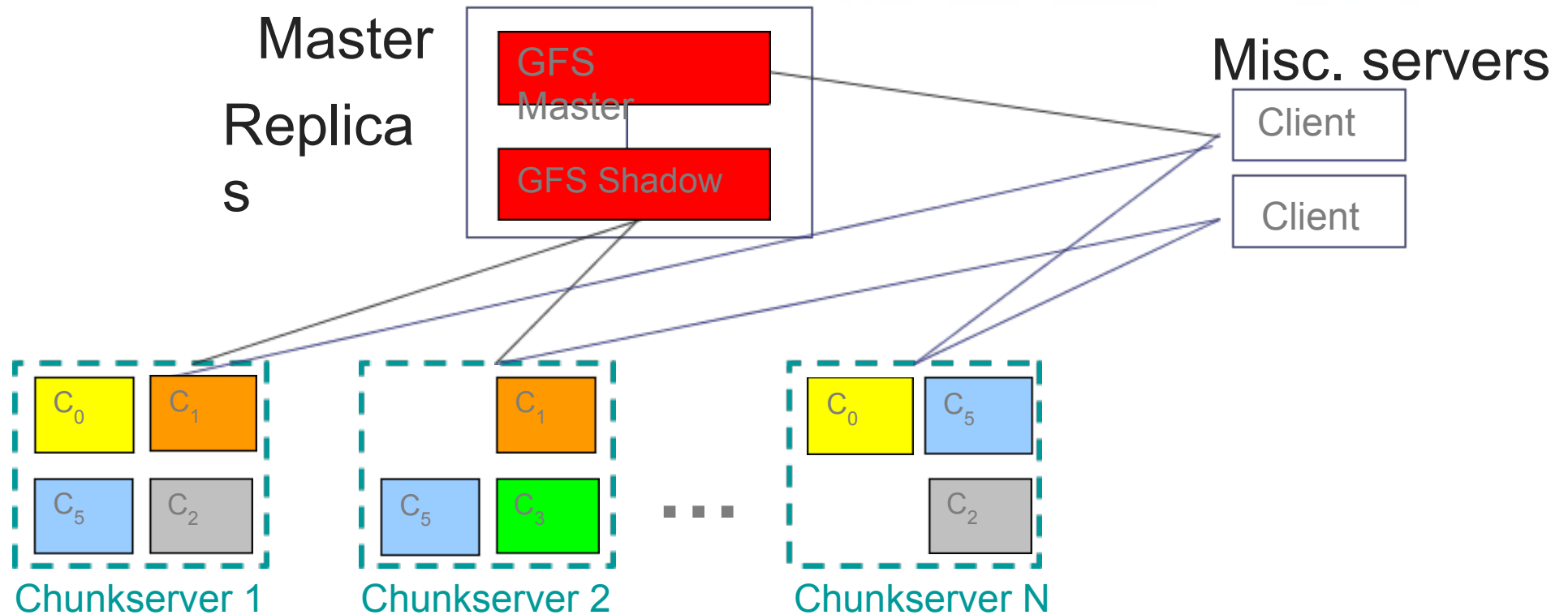
- Reliable distributed storage system for petabyte scale filesystems.
- Data kept in 64-megabyte “chunks” stored on disks spread across thousands of machines
- Each chunk replicated, usually 3 times, on different machines so that GFS can recover seamlessly from disk or machine failure.
- A GFS cluster consists of a single *master*, multiple *chunkservers*, and is accessed by multiple *clients*.

See SOSP'03 paper at:

<http://labs.google.com/papers/gfs.html>



# GFS: The Google File System



- Master manages metadata
- Data transfers happen directly between clients/chunkservers
- Files broken into chunks (typically 64 MB)
- Chunks triplicated across three machines for safety

# BigTable

- A distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.
- Built on top of GFS.
- Used by more than 100 Google products and projects including Google Earth, Google Finance, Orkut, ...
- Paper by Fay Chang, Jeffrey Dean, Sanjay Ghemawat, et al appeared at OSDI'06.

See the OSDI'06 paper on:

<http://labs.google.com/papers/bigtable.html>



# BigTable: An Example for crawl data

- A web crawling system might use BigTable that stores web pages.
- Each row key could represent a specific URL, with columns for the page contents, the language, the references to that page, or other metadata.
- The row range for a table is dynamically partitioned between servers.
- Rows are clustered together on machines by key, so using URLs as keys would minimize the number of machines where pages from a single domain are stored.
- Each cell is timestamped so there could be multiple versions of the same data in the table.



# Workqueue: Scheduling jobs on machines

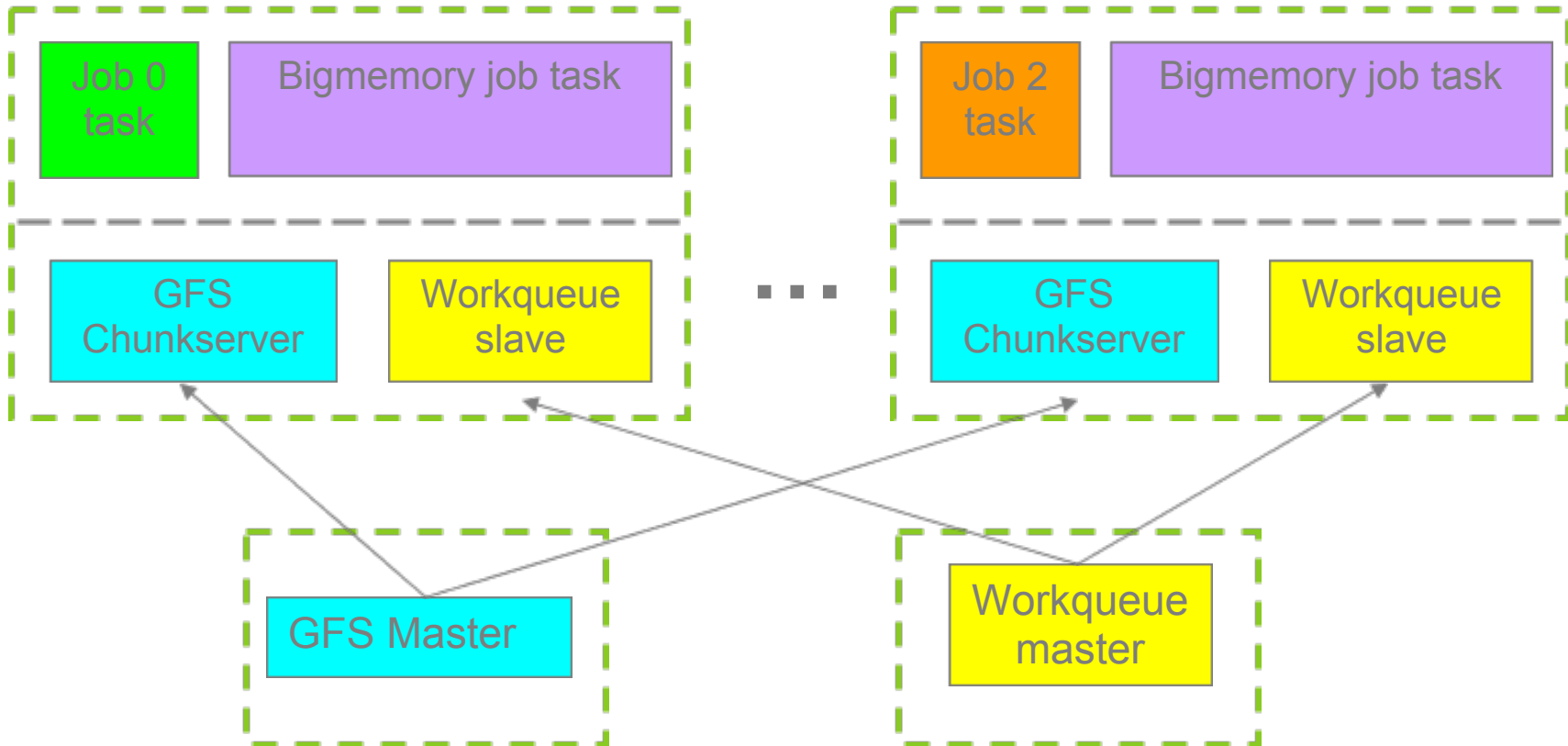
---

- A large scale time sharing system built out of an array of computers and their RAM, CPUs and disks.
- Schedules jobs, allocates resources, reports status, and collects the results.
- Similar to other distributed systems described in the literature, such as Condor.
- Machines that serve GFS also contribute CPU/RAM to a workqueue, compute requirements of GFS is light
- So a machine A may contribute to storage cluster, and to compute cluster these clusters overlap.

# Workqueue: Scheduling jobs on machines

Machine 1

Machine N



# MapReduce

- A programming model and an associated implementation for processing and generating large data sets.
- A user specified map function processes a key/value pair to generate a set of intermediate key/value pairs.
- A user specified reduce function merges all intermediate values associated with the same intermediate key.
- Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines.

MapReduce homepage:

<http://labs.google.com/papers/mapreduce.html>



# MapReduce: Runtime Environment

---

- The MapReduce runtime environment takes care of:
  - Partitioning the input data.
  - Scheduling the program's execution across a set of machines.
  - Handling machine failures.
  - Managing required inter-machine communication.
- Allows programmers without experience with distributed systems to easily utilize a large distributed cluster.
- All at the application level built on top of Linux.

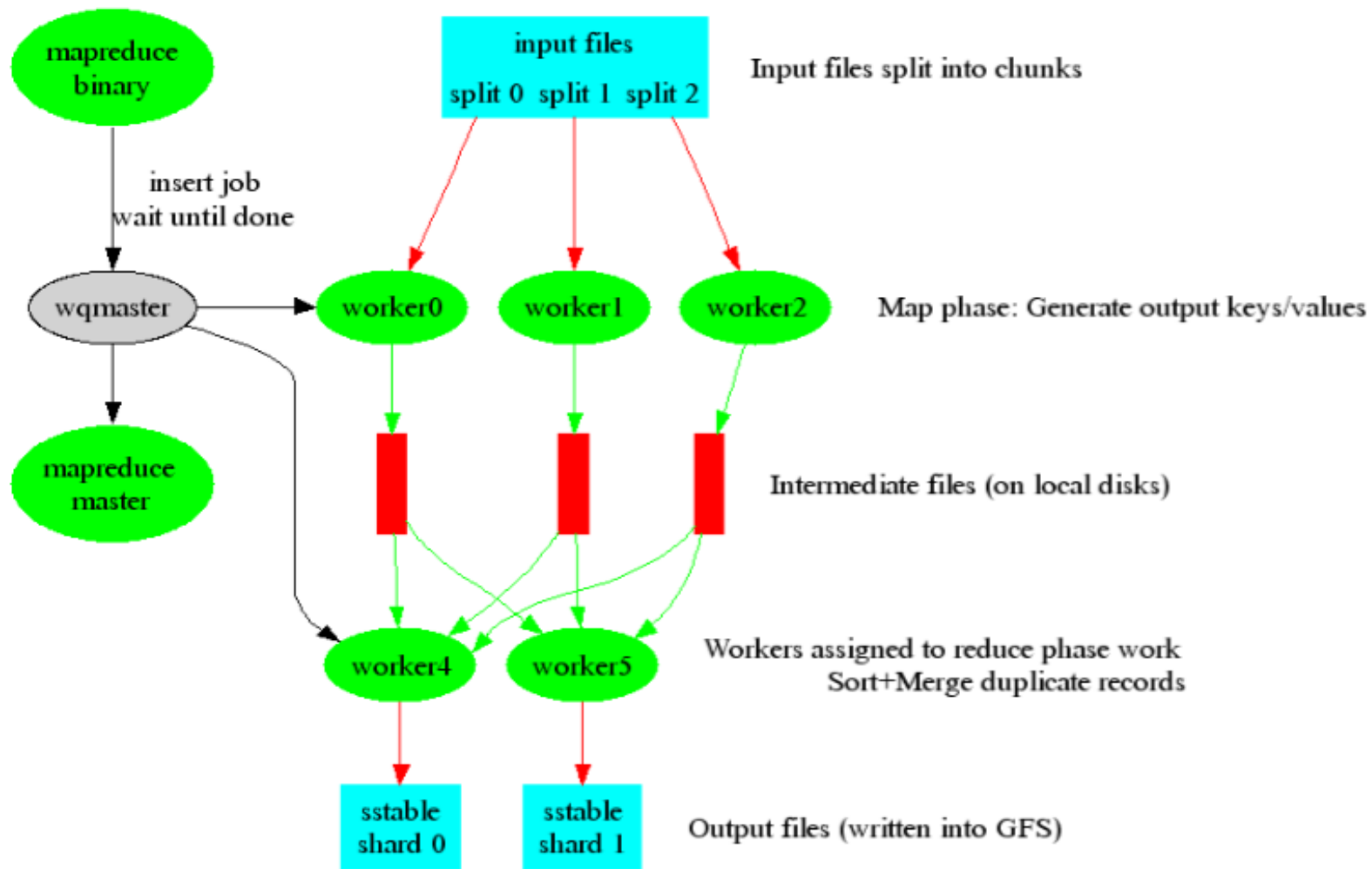


# MapReduce: Grep for Sysadmins

- **Distribute:** `grep "130.225.226.91" the_web > matches.txt`
- A distributed grep with MapReduce would:
  - Split up the file into individual chunks.
  - Send each small chunk to a different machine.
  - Have each of the machines run the same mapper on their data chunk (grep "ip address" in this case)
  - Collate the matches from the individual worker machines into an output file (the reducer, in this example the *identity* function).
- System optimizations possible:
  - Data locality
  - Network topology (rack diversity, etc..)



# MapReduce: System Structure



See: "MapReduce: Simplified Data Processing on Large Clusters"

OSDI'04: <http://labs.google.com/papers>



# MapReduce: Another Example

## Example: *compute the Inverse Document Frequency*

for a list of urls, return the number of documents each word occurs in

- Input is sequence of key/value pairs
  - **BigTable** : { "http://www.opensourcedays.org/elgoog.html": "organize world information", "http://code.google.com/": "opensource information", ... }
- Users write two simple functions:
  - **Map** : input [k1,v1], produce bag of intermediate [k2,v2]+ pairs
    - E.g. `map(url, contents) -> [[organize,1],[world,1],[information,1]]` and `[[opensource,1],[information,1]]`
  - **Reduce** : takes intermediate [k2,v2], combines to output [k2,v3]
    - E.g. `reduce(information => [1,1]) -> information => 2`
- [k2,v3] value are emitted to an output file



## One step further - Sawzall

- An interpreted language built on top of the Map Reduce framework.
- Examines one record at a time

Write a Map Reduce using 7 lines of code :-)

***Scientific Programming Issue: Volume 13, Number 4, 2005:  
Interpreting the data: Parallel analysis with Sawzall, Rob Pike,  
Sean Dorward, Robert Griesemer, Sean Quinlan***



# Sawzall – An Example

- Uses MapReduce
  - Do stuff with raw data on many machines (mapper)
  - Aggregate sorted data (reducer)
- Collect, collate and format the output, example:

```
count: table sum of int;
```

```
total: table sum of float;
```

```
sum_of_squares: table sum of float;
```

```
x: float = input;
```

```
emit count <- 1;
```

```
emit total <- x;
```

```
emit sum_of_squares <- x * x;
```



# Sawzall – Another Example

- Create a count of visits, broken down per country

```
countries: table sum[country: string] of count: int;  
if (!def(logentry) || remotehost.internal) {  
  return;  
}  
emit countries[remotehost.geo.country] <- 1;
```

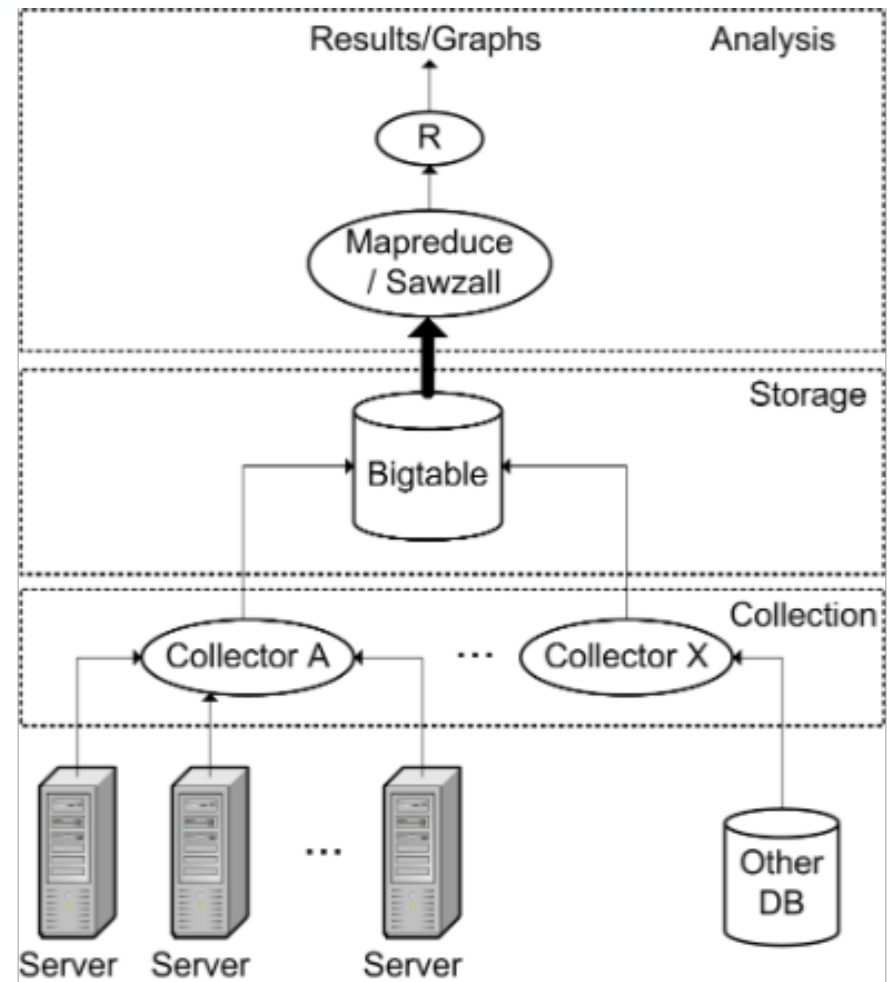
- Output of the sawzall job on some access\_log:

```
countries[dnk] = 11517818  
countries[nld] = 96207152  
countries[usa] = 1581457029
```



# Dogfood: The System Health Infrastructure

- Talks to every server frequently
- Collects
  - Health signals
  - Activity information
  - Configuration data
- Stores time-series data forever
- Enable highly parallel analysis of repository data





# Many Thanks!

---

- Murray Stokely
- Jeff Dean
- Sanjay Ghemawat
- Rob Pike
- Tom Limoncelli
- Shun-Tak Leung
- Howard Gobioff
- Luiz Barroso
- Doug Orr