Tutorials, June 19th 2009

Google

MapReduce

The Programming Model and Practice

Jerry Zhao

Technical Lead

Jelena Pjesivac-Grbovic

Software Engineer

Yet another MapReduce tutorial?

Google

Some tutorials you might have seen:

- Introduction to MapReduce Programming Model
- Hadoop Map/Reduce Programming Tutorial
- and more.

What makes this one different:

- Some complex "realistic" MapReduce examples
- Brief discussion of trade-offs between alternatives
- Google MapReduce implementation internals, tuning tips

About this presentation

- Edited collaboratively on Google Docs for domains

Tutorial Overview

Google

- MapReduce programming model
  - **Brief intro to MapReduce**
  - Use of MapReduce inside Google
  - MapReduce programming examples
  - MapReduce, similar and alternatives
- Implementation of Google MapReduce
  - Dealing with failures
  - Performance & scalability
  - Usability

What is MapReduce?

Google

A programming model for large-scale distributed data processing

- Simple, elegant concept
- Restricted, yet powerful programming construct
- Building block for other parallel programming tools
- Extensible for different applications

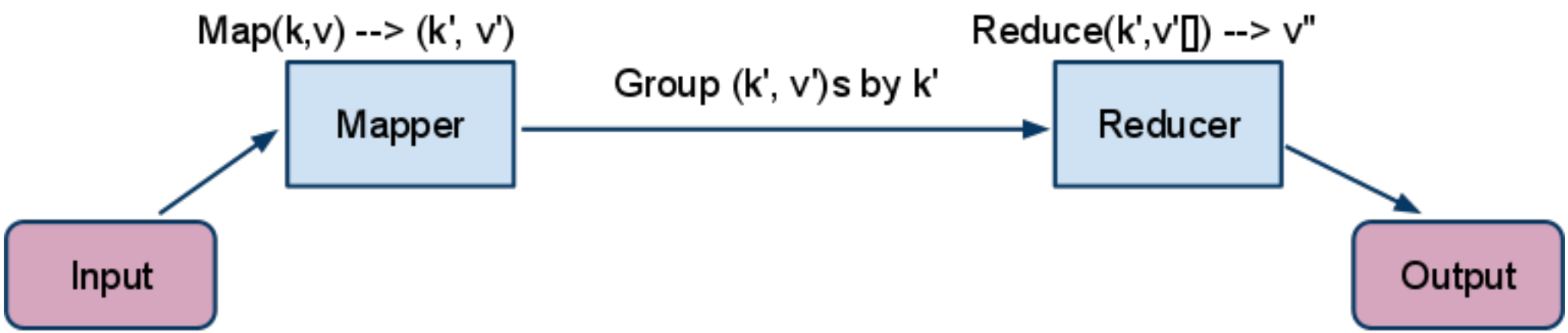
Also an implementation of a system to execute such programs

- Take advantage of parallelism
- Tolerate failures and jitters
- Hide messy internals from users
- Provide tuning knobs for different applications

Programming Model

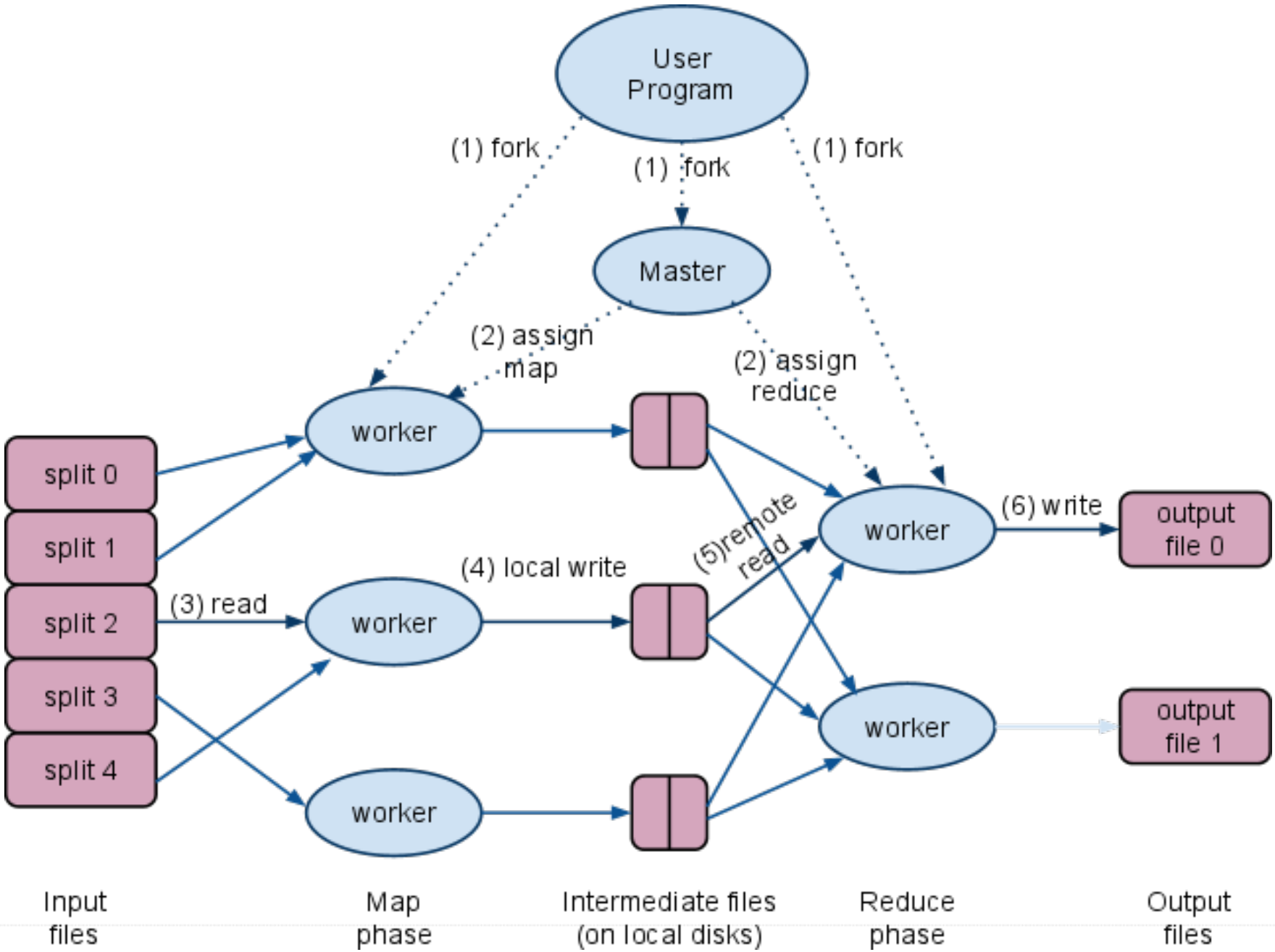
Google

Inspired by Map/Reduce in functional programming languages, such as LISP from 1960's, but not equivalent



MapReduce Execution Overview

Google



Tutorial Overview

Google

- MapReduce programming model
  - Brief intro to MapReduce
  - **Use of MapReduce inside Google**
  - MapReduce programming examples
  - MapReduce, similar and alternatives
- Implementation of Google MapReduce
  - Dealing with failures
  - Performance & scalability
  - Usability

Use of MapReduce inside Google

Google

Stats for Month	Aug.'04	Mar.'06	Sep.'07
Number of jobs	29,000	171,000	2,217,000
Avg. completion time (secs)	634	874	395
Machine years used	217	2,002	11,081
Map input data (TB)	3,288	52,254	403,152
Map output data (TB)	758	6,743	34,774
reduce output data (TB)	193	2,970	14,018
Avg. machines per job	157	268	394
Unique implementations			
Mapper	395	1958	4083
Reducer	269	1208	2418

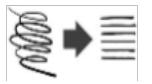
From "MapReduce: simplified data processing on large clusters"

MapReduce inside Google

Google

Googlers' hammer for 80% of our data crunching

- Large-scale web search indexing
- Clustering problems for Google News
- Produce reports for popular queries, e.g. Google Trend
- Processing of satellite imagery data
- Language model processing for statistical machine translation
- Large-scale machine learning problems
- Just a plain tool to reliably spawn large number of tasks
  - e.g. parallel data backup and restore

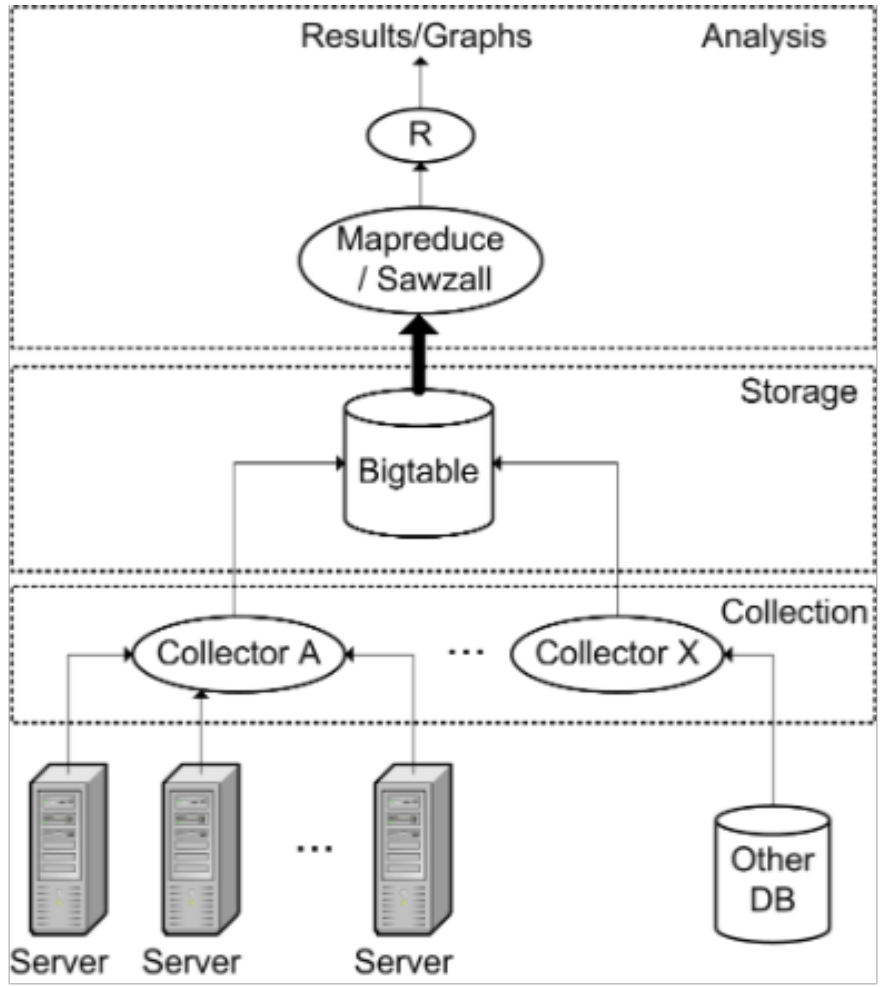


The other 20%? e.g. Pregel

Use of MR in System Health Monitoring

Google

- Monitoring service talks to every server frequently
- Collect
  - Health signals
  - Activity information
  - Configuration data
- Store time-series data forever
- Parallel analysis of repository data
  - MapReduce/Sawzall



Investigating System Health Issues

Google

- Case study
  - Higher DRAM errors observed in a new GMail cluster
  - Similar servers running GMail elsware not affected
    - Same version of the software, kernel, firmware, etc.
  - Bad DRAM is the initial culprit
    - ... but that same DRAM model was fairly healthy elsewhere
  - Actual problem: bad motherboard batch
    - Poor electrical margin in some memory bus signals
    - GMail got more than its fair share of the bad batch
    - Analysis of this batch allocated to other services confirmed the theory
- Analysis possible by having all relevant data in one place and processing power to digest it
  - MapReduce is part of the infrastructure

Tutorial Overview

Google

- MapReduce programming model
  - Brief intro to MapReduce
  - Use of MapReduce inside Google
  - **MapReduce programming examples**
  - MapReduce, similar and alternatives
- Implementation of Google MapReduce
  - Dealing with failures
  - Performance & scalability
  - Usability

Application Examples

Google

- Word count and frequency in a large set of documents
  - Power of sorted keys and values
  - Combiners for map output
- Computing average income in a city for a given year
  - Using customized readers to
    - Optimize MapReduce
    - Mimic **rudimentary** DBMS functionality
- Overlaying satellite images
  - Handling various input formats using protocol bufers

Word Count Example

Google

- Input: Large number of text documents
- Task: Compute word count across all the document

Solution

- Mapper:
  - For every word in a document output (word, "1")
- Reducer:
  - Sum all occurrences of words and output (word, total_count)

Word Count Solution

Google

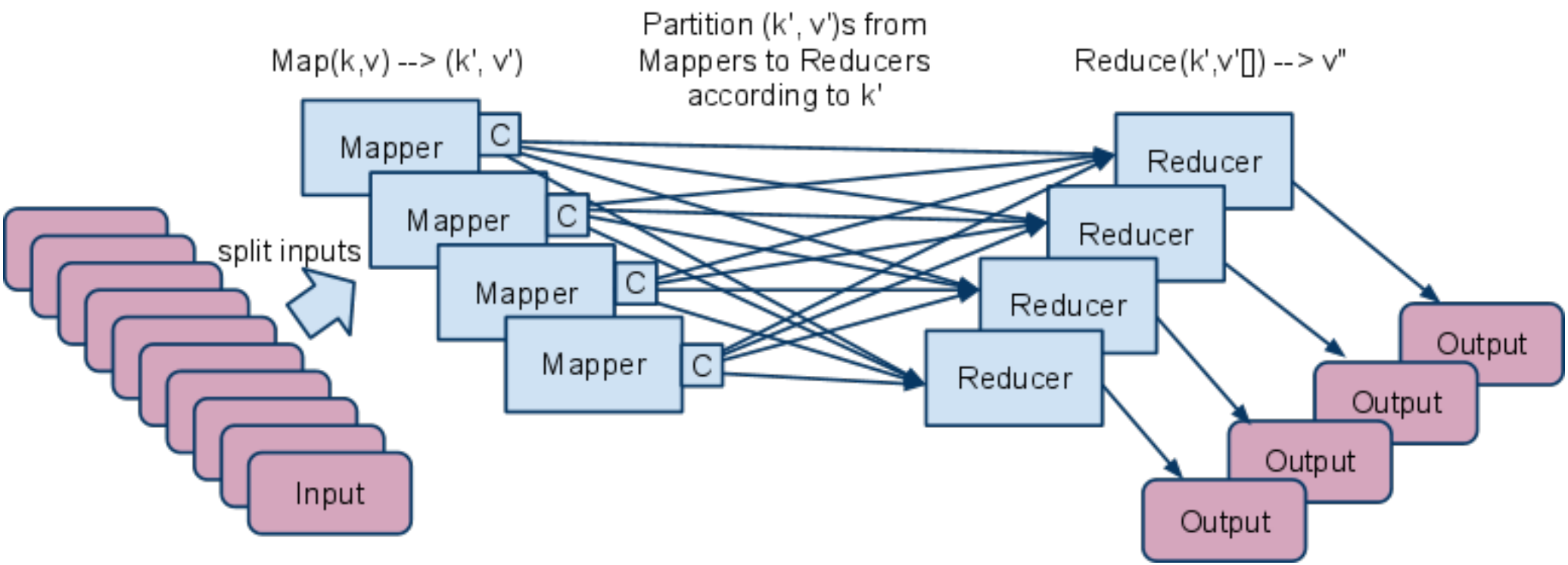
```
//Pseudo-code for "word counting"
map(String key, String value):
  // key: document name,
  // value: document contents
    for each word w in value:
      EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
    int word_count = 0;
    for each v in values:
      word_count += ParseInt(v);
    Emit(key, AsString(word_count));
```

No types, just strings*

Word Count Optimization: Combiner

Google

- Apply reduce function to map output before it is sent to reducer
  - Reduces number of records outputted by mapper!



Word Frequency Example

Google

- Input: Large number of text documents
- Task: Compute word frequency across all the document
  - Frequency is calculated using the total word count

- A naive solution with basic MapReduce model requires two MapReduces
  - MR1: count number of all words in these documents
    - Use combiners
  - MR2: count number of each word and divide it by the total count from MR1

Word Frequency Example

Google

- Can we do better?
- Two nice features of Google's MapReduce implementation
  - Ordering guarantee of reduce key
  - Auxiliary functionality: EmitToAllReducers(k, v)

- A nice trick: To compute the total number of words in all documents
  - Every map task sends its total world count with key "" to ALL reducer splits
  - Key "" will be the first key processed by reducer
    - Sum of its values → total number of words!

Word Frequency Solution: Mapper with Combiner

Google

```
map(String key, String value):
  // key: document name, value: document contents
  int word_count = 0;
  for each word w in value:
    EmitIntermediate(w, "1");
    word_count++;
  EmitIntermediateToAllReducers("", AsString(word_count));

combine(String key, Iterator values):
  // Combiner for map output
 // key: a word, values: a list of counts
 int partial_word_count = 0;
 for each v in values:
   partial_word_count += ParseInt(v);
 Emit(key, AsString(partial_word_count));
```

Word Frequency Solution: Reducer

Google

```
reduce(String key, Iterator values):
  // Actual reducer
  // key: a word
  // values: a list of counts
  if (is_first_key):
    assert("" == key);  // sanity check
    total_word_count_ = 0;
    for each v in values:
      total_word_count_ += ParseInt(v)
  else:
    assert("" != key);  // sanity check
    int word_count = 0;
    for each v in values:
      word_count += ParseInt(v);
    Emit(key, AsString(word_count / total_word_count_));
```

Application Examples

Google

- Word frequency in a large set of documents
  - Power of sorted keys and values
  - Combiners for map output
- **Computing average income in a city for a given year**
  - Using customized readers to
    - Optimize MapReduce
    - Mimic **rudimentary** DBMS functionality
- Overlaying satellite images
  - Handling various input formats using protocol bufers

Average Income In a City

Google

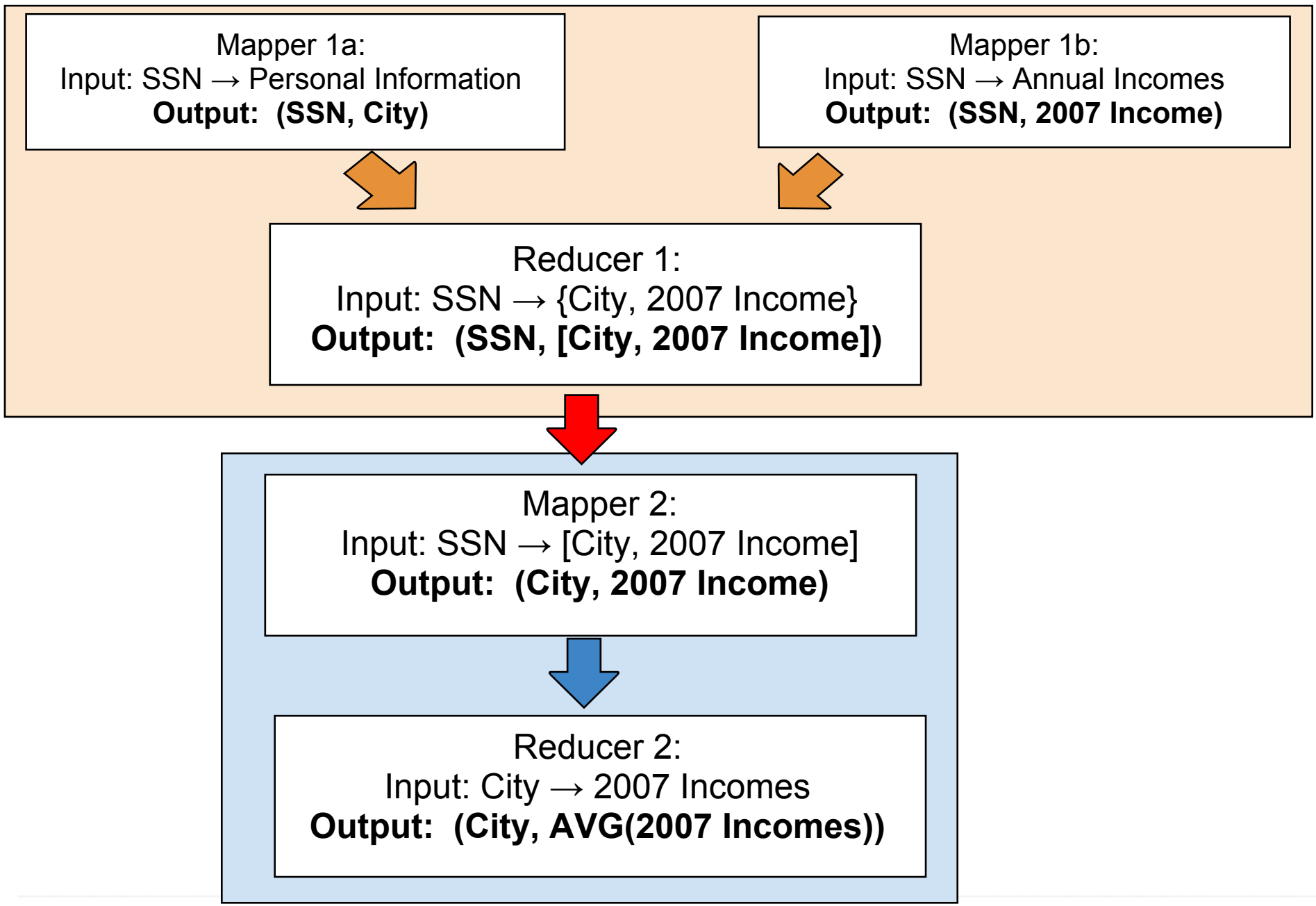
SSTable 1: (SSN, {Personal Information})
123456:(John Smith;Sunnyvale, CA)
123457:(Jane Brown;Mountain View, CA)
123458:(Tom Little;Mountain View, CA)

SSTable 2: (SSN, {year, income})
123456:(2007,$70000),(2006,$65000),(2005,$6000),...
123457:(2007,$72000),(2006,$70000),(2005,$6000),...
123458:(2007,$80000),(2006,$85000),(2005,$7500),...

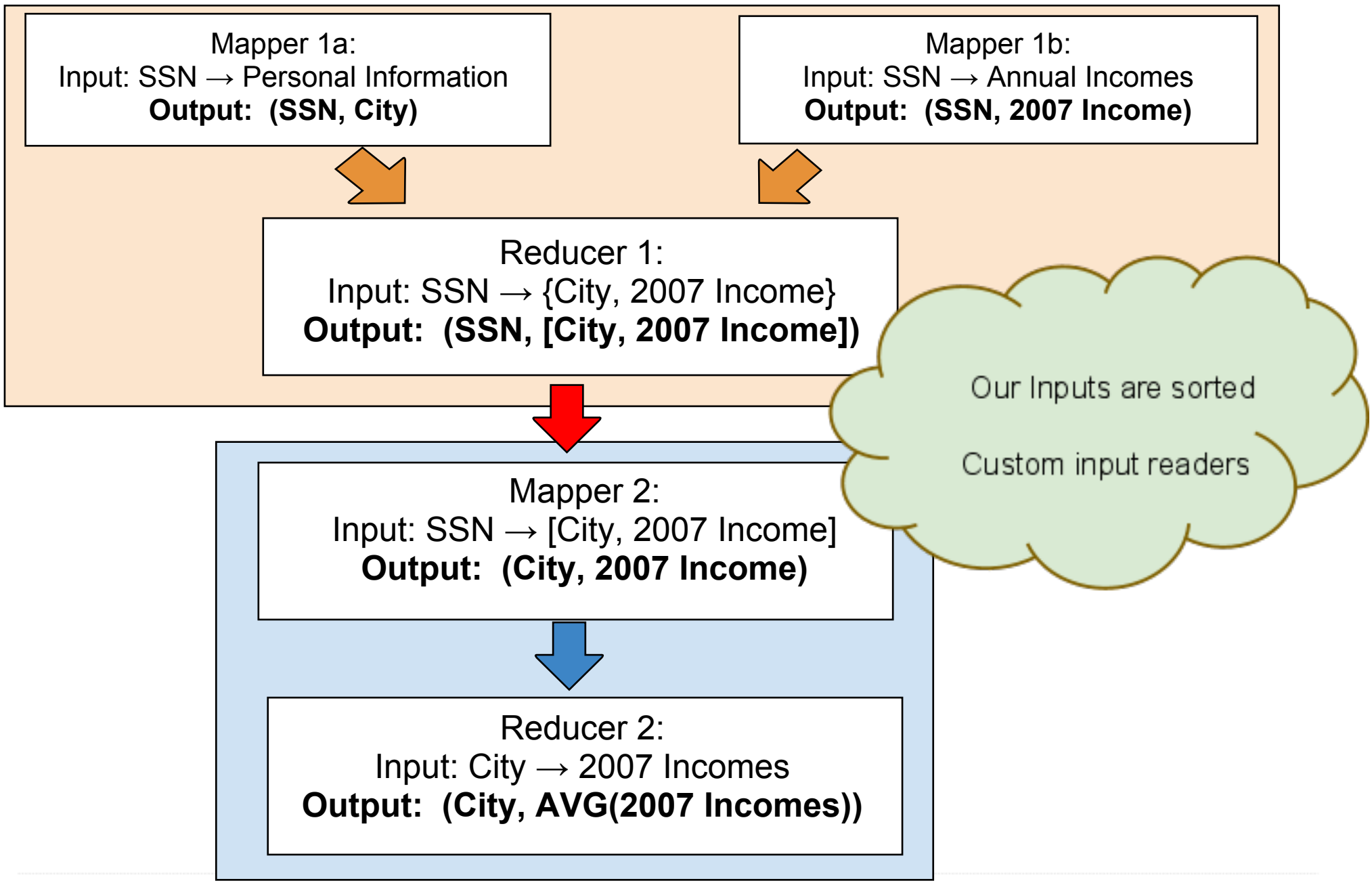
Task: Compute average income in each city in 2007

Note: **Both inputs sorted by SSN**

Average Income in a City Basic Solution

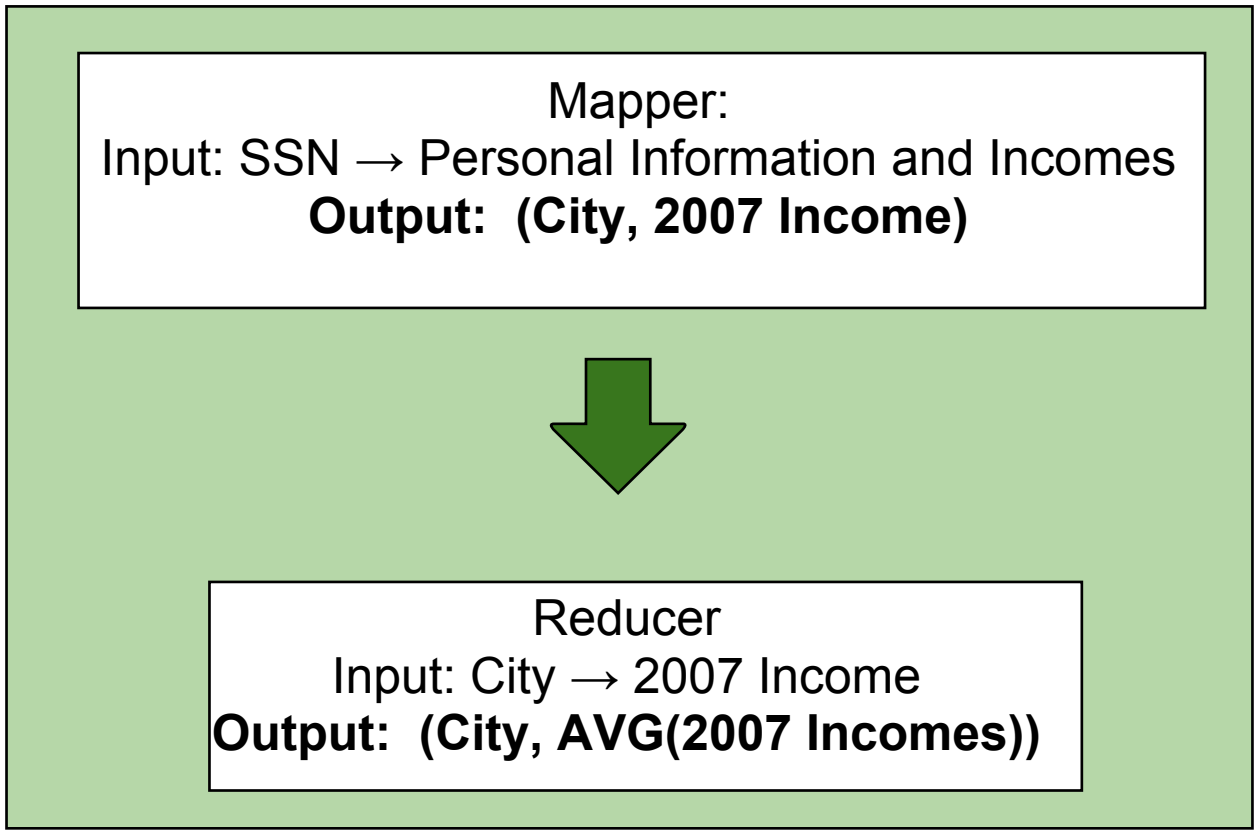


Average Income in a City Basic Solution



Average Income in a Joined Solution

Google



Application Examples

Google

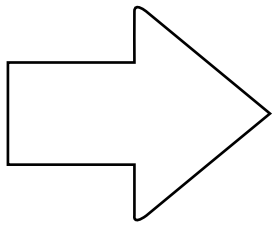
- Word frequency in a large set of documents
  - Power of sorted keys and values
  - Combiners for map output
- Computing average income in a city for a given year
  - Using customized readers to
    - Optimize MapReduce
    - Mimic **rudimentary** DBMS functionality
- **Overlaying satellite images**
  - Handling various input formats using protocol bufers

Stitch Imagery Data for Google Maps

Google

A simplified version could be:

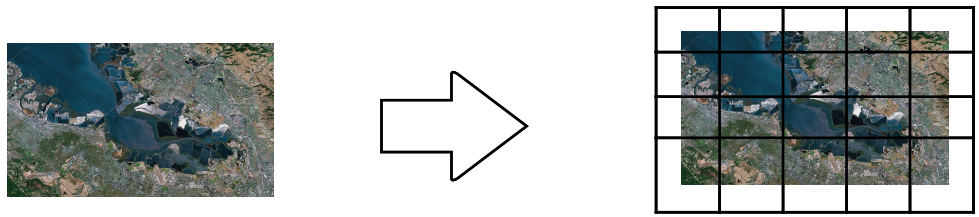
- Imagery data from different content providers
  - Different formats
  - Different coverages
  - Different timestamps
  - Different resolutions
  - Different exposures/tones
- Large amount to data to be processed
- Goal: produce data to serve a "satellite" view to users



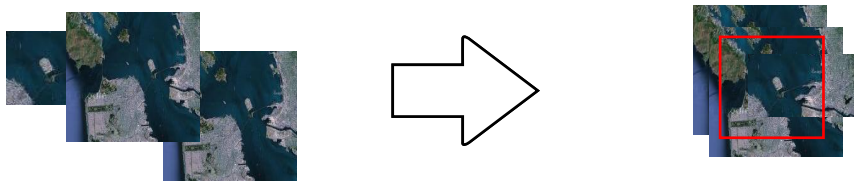
Stitch Imagery Data Algorithm

Google

1. Split the whole territory into "tiles" with fixed location IDs
2. Split each source image according to the tiles it covers



3. For a given tile, stitch contributions from different sources, based on its freshness and resolution, or other preference



4. Serve the merged imagery data for each tile, so they can be loaded into and served from a image server farm.

Using Protocol Buffers to Encode Structured Data

Google

- Open sourced from Google, among many others:
  http://code.google.com/p/protobuf/
- It supports C++, Java and Python.
- A way of encoding structured data in an efficient yet extensible format. e.g. we can define

```
message Tile {
    required int64 location_id = 1;
    group coverage {
        double latitude = 2;
        double longitude = 3;
        double width = 4;     // in km
        double length = 5;    // in km
    }
    required bytes image_data = 6; // Bitmap Image data
    required int64 timestamp = 7;
    optional float resolution = 8 [default = 10];
    optinal string debug_info = 10;
}
```

Google uses Protocol Buffers for almost all its internal RPC protocols, file formats and of course in MapReduce.

Stitch Imagery Data Solution: Mapper

Google

```
map(String key, String value):
  // key: image file name
  // value: image data
  Tile whole_image;

  switch (file_type(key)):
    FROM_PROVIDER_A: Convert_A(value, &whole_image);
    FROM PROVIDER_B: Convert_B(...);
    ...

  // split whole_image according to the grid into tiles
  for each Tile t in whole_image
        string v;
        t.SerializeToString(&v);
        EmitIntermediate(IntToStr(t.location_id(),v);
```

Stitch Imagery Data Solution: Reducer

Google

```
reduce(String key, Iterator values):
  // key: location_id,
  // values: tiles from different sources

  sort values according v.resolution() and v.timestamp();

  Tile merged_tile;
  for each v in values:
    overlay pixels in v to merged_tile based on
        v.coverage();

  Normalize merged_tile to be the serve tile size;

  Emit(key, ProtobufToString(merged_tile));
```

Tutorial Overview

Google

- MapReduce programming model
  - Brief intro to MapReduce
  - Use of MapReduce inside Google
  - MapReduce programming examples
  - **MapReduce, similar and alternatives**
- Implementation of Google MapReduce
  - Dealing with failures
  - Performance & scalability
  - Usability

Distributed Computing Landscape

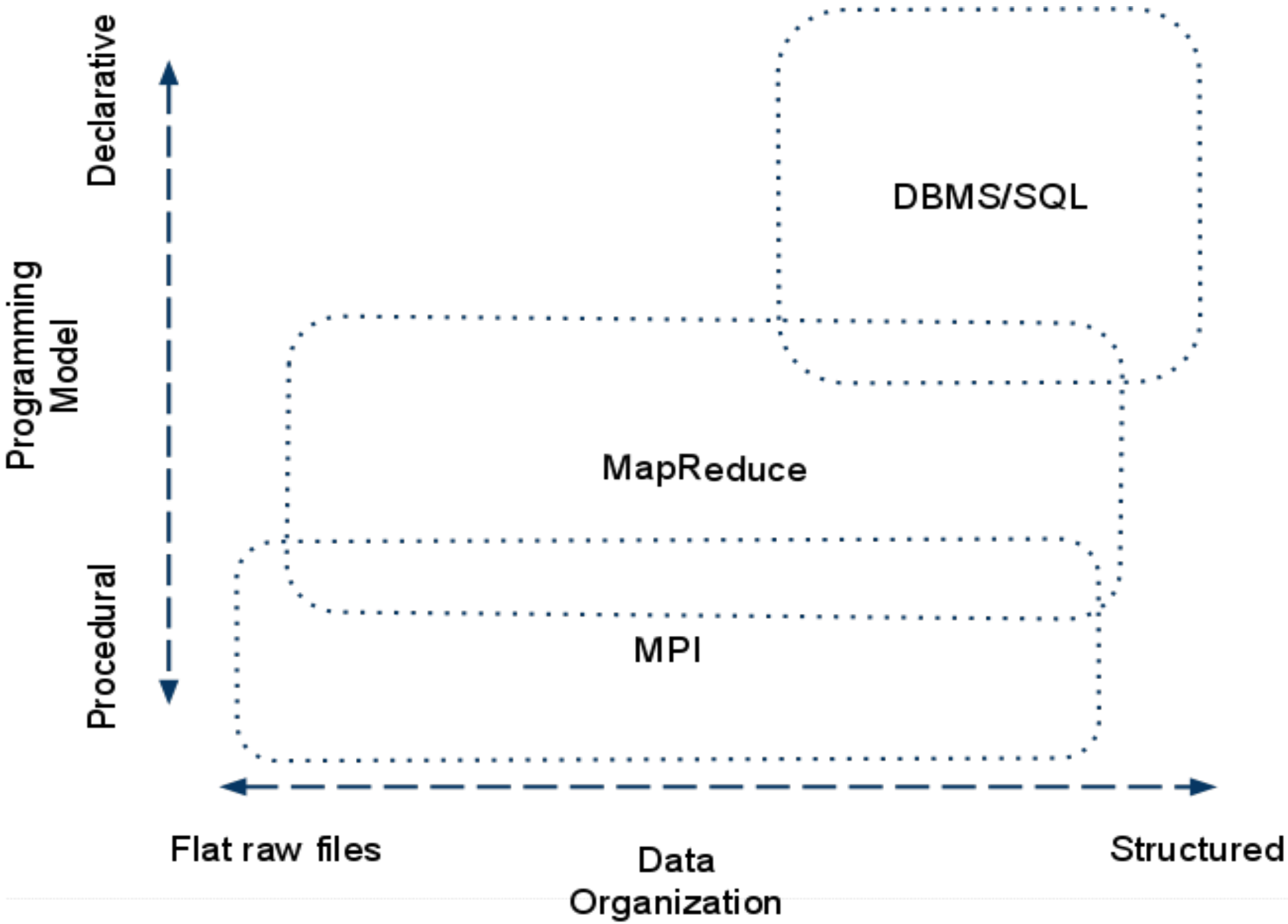
Google

Dimensions to compare Apples and Oranges

- Data organization
- Programming model
- Execution model
- Target applications
- Assumed computing environment
- Overall operating cost

My Basket of Fruit

Google



Nutritional Information of My Basket

Google

	MPI	MapReduce	DBMS/SQL
What they are	A general parrellel programming paradigm	A programming paradigm and its associated execution system	A system to store, manipulate and serve data.
Programming Model	Messages passing between nodes	Restricted to Map/Reduce operations	Declarative on data query/retrieving; Stored procedures
Data organization	No assumption	"files" can be sharded	Organized datastructures
Data to be manipulated	Any	k,v pairs: string/protomsg	Tables with rich types
Execution model	Nodes are independent	Map/Shuffle/Reduce Checkpointing/Backup Physical data locality	Transaction Query/operation optimization Materialized view
Usability	Steep learning curve*; difficult to debug	Simple concept Could be hard to optimize	Declarative interface; Could be hard to debug in runtime
Key selling point	Flexible to accommodate various applications	Plow through large amount of data with commodity hardware	Interactive querying the data; Maintain a consistent view across clients

See what others say: [1], [2], [3]

Taste Them with Your Own Grain of Salt

Google

Dimensions to choose between Apples and Oranges for an application developer:

- Target applications
  - Complex operations run frequently v.s. one time plow
  - Off-line processing v.s. real-time serving
- Assumed computing environment
  - Off-the-shelf, custom-made or donated
  - Formats and sources of your data
- Overall operating cost
  - Hardware maintenance, license fee
  - Manpower to develop, monitor and debug

Existing MapReduce and Similar Systems

Google

Google MapReduce

- Support C++, Java, Python, Sawzall, etc.
- Based on proprietary infrastructures
  - GFS(SOSP'03), MapReduce(OSDI'04) , Sawzall(SPJ'05), Chubby (OSDI'06), Bigtable(OSDI'06)
  - and some open source libraries

Hadoop Map-Reduce

- Open Source! (Kudos to Doug and the team.)
- Plus the whole equivalent package, and more
  - HDFS, Map-Reduce, Pig, Zookeeper, HBase, Hive
- Used by Yahoo!, Facebook, Amazon and Google-IBM NSF cluster

Dryad

- Proprietary, based on Microsoft SQL servers
- Dryad(EuroSys'07), DryadLINQ(OSDI'08)
- Michael's Dryad TechTalk@Google (Nov.'07)

And others

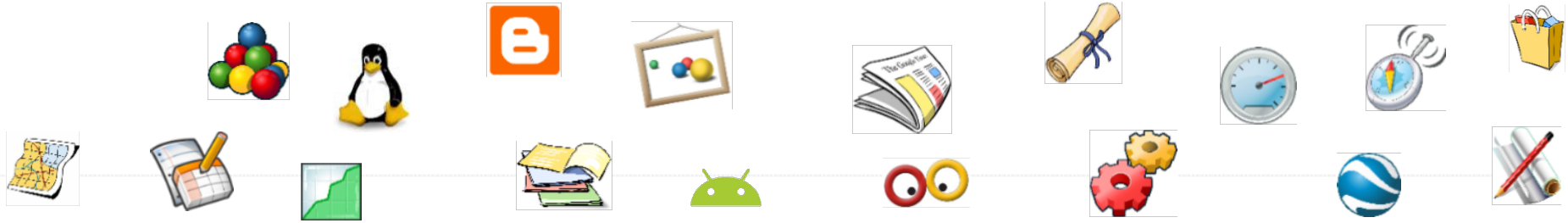
Tutorial Overview

Google

- MapReduce programming model
  - Brief intro to MapReduce
  - Use of MapReduce inside Google
  - MapReduce programming examples
  - MapReduce, similar and alternatives
- Implementation of Google MapReduce
  - **Dealing with failures**
  - Performance & scalability
  - Usability

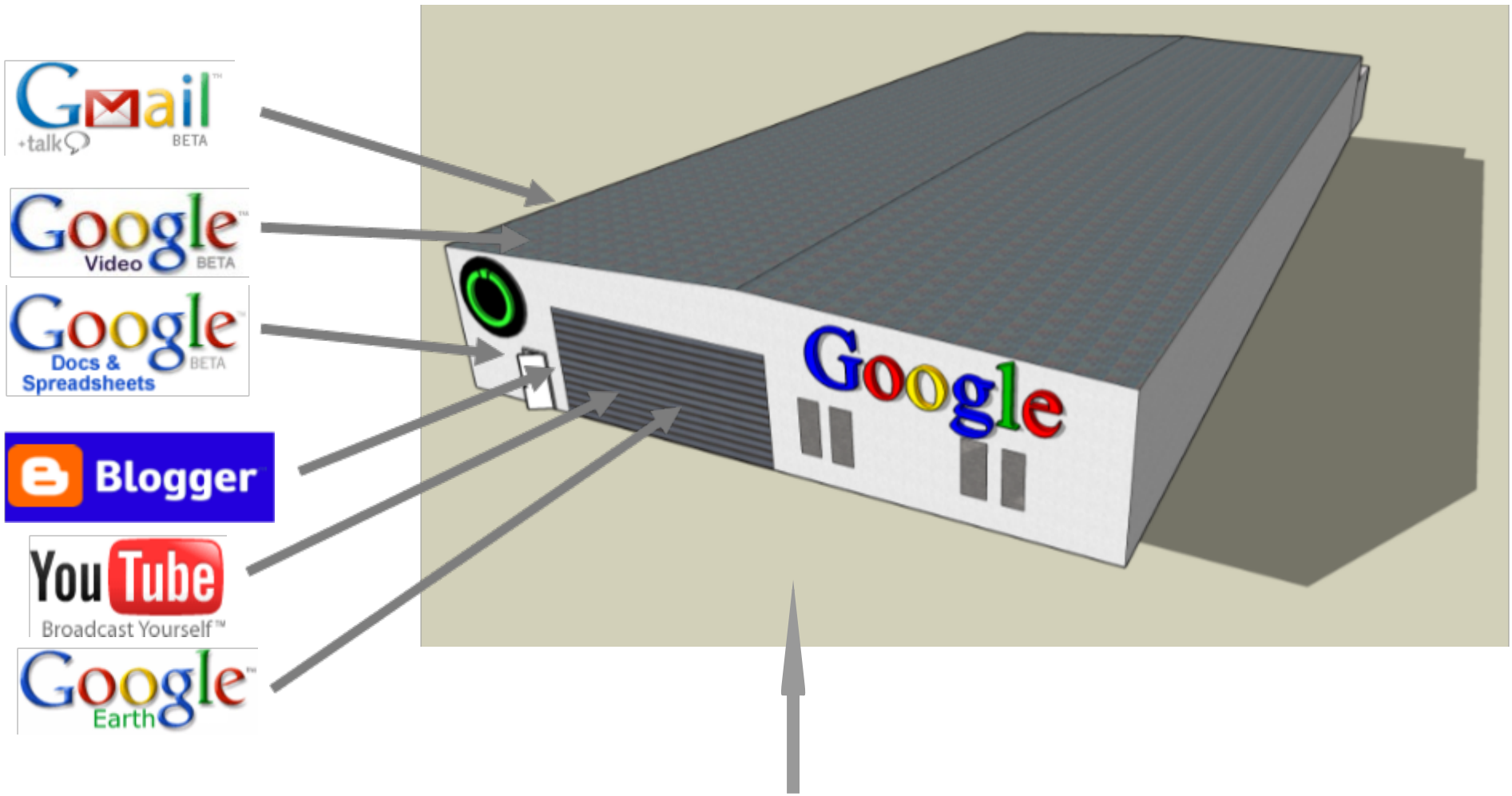
Google Computing Infrastructure

- Infrastructure must support
  - Diverse set of applications
    - Increasing over time
  - Ever-increasing application usage
  - Ever-increasing computational requirements
  - Cost effective
- Data centers
  - Google-specific mechanical, thermal and electrical design
  - Highly-customized PC-class motherboards
  - Running Linux
  - In-house management & application software



Sharing is the Way of Life

Google



+ Batch processing
(MapReduce, Sazwall)

Major Challenges

Google

To organize the world's information and make it universally accessible and useful.

- Failure handling
  - Bad apples appear now and there
- Scalability
  - Fast growing dataset
  - Broad extension of Google services
- Performance and utilization
  - Minimizing run-time for individual jobs
  - Maximizing throughput across all services
- Usability
  - Troubleshooting
  - Performance tuning
  - Production monitoring

Failures in Literature

Google

- LANL data (DSN 2006)
  - Data collected over 9 years
  - Covered 4750 machines and 24101 CPUs
  - Distribution of failures
    - Hardware ~ 60%, Software ~ 20%, Network/Environment/Humans ~ 5%, Aliens ~ 25%*
    - Depending on a system, failures occurred between once a day to once a month
  - Most of the systems in the survey were the cream of the crop at their time
- PlanetLab (SIGMETRICS 2008 HotMetrics Workshop)
  - Average frequency of failures per node in a 3-months period
    - Hard failures: 2.1
    - Soft failures: 41
    - Approximately failure every 4 days

Failures in Google Data Centers

Google

- DRAM errors analysis (SIGMETRICS 2009)
  - Data collected over 2.5 years
  - 25,000 to 70,000 errors per billion device hours per Mbit
    - Order of magnitude more than under lab conditions
  - 8% of DIMMs affected by errors
  - Hard errors are dominant cause of failure
- Disk drive failure analysis (FAST 2007)
  - Annualized Failure Rates vary from 1.7% for one year old drives to over 8.6% in three year old ones
  - Utilization affects failure rates only in very old and very old disk drive populations
  - Temperature change can cause increase in failure rates but mostly for old drives

Failures in Google

Google

- Failures are a part of everyday life
  - Mostly due to the scale and shared environment
- Sources of job failures
  - Hardware
  - Software
  - Preemption by a more important job
  - Unavailability of a resource due to overload
- Failure types
  - Permanent
  - Transient

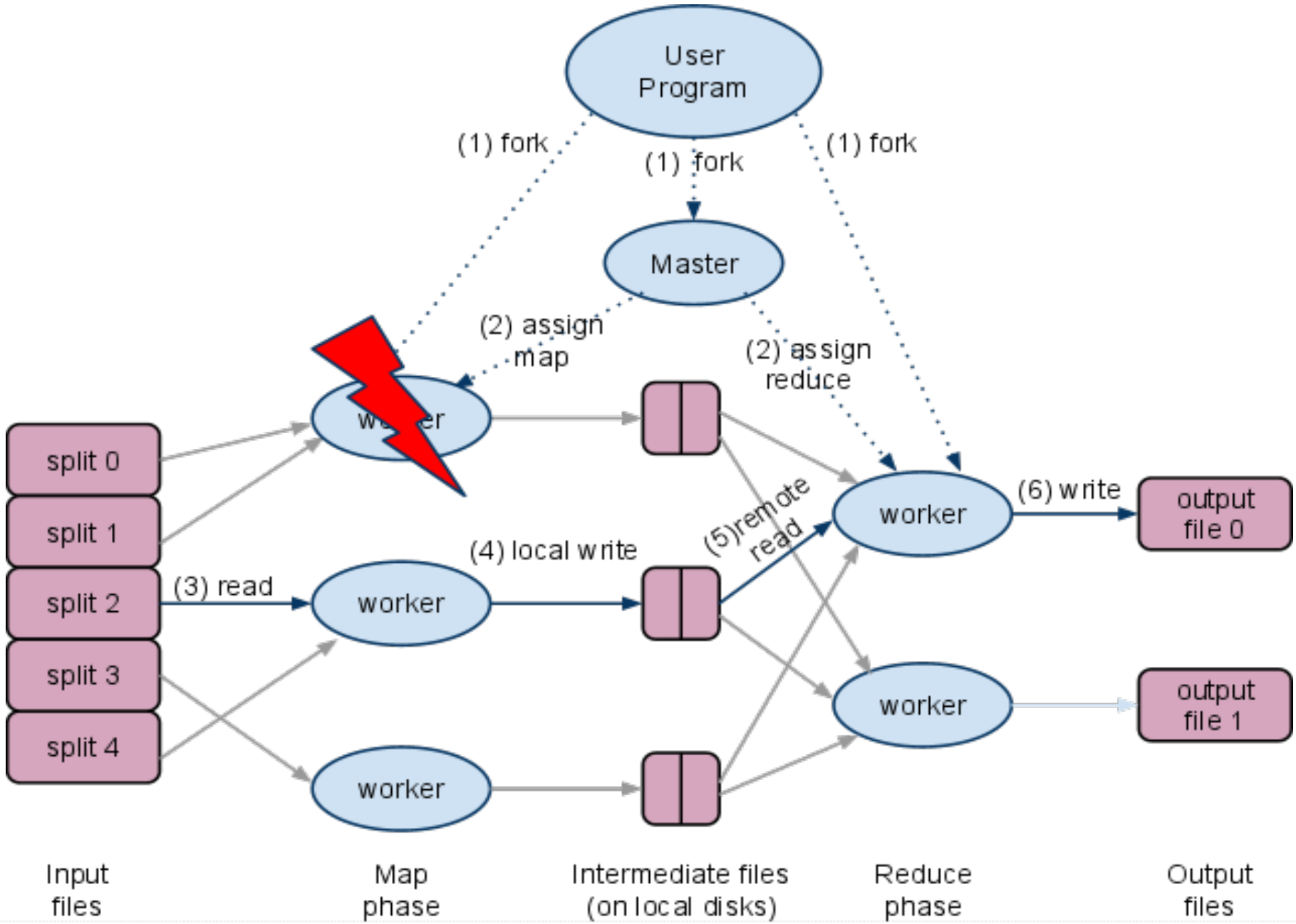
Different Failures Require Different Actions

Google

- Fatal failure (the whole job dies)
  - Simplest case around :)
  - You'd prefer to resume computation rather than recompute
- Transient failures
  - You'd want your job to adjust and finish when issues resolve
- Program hangs.. forever.
  - Define "forever"
  - Can we figure out why?
  - What to do?
- "It's-Not-My-Fault" failures

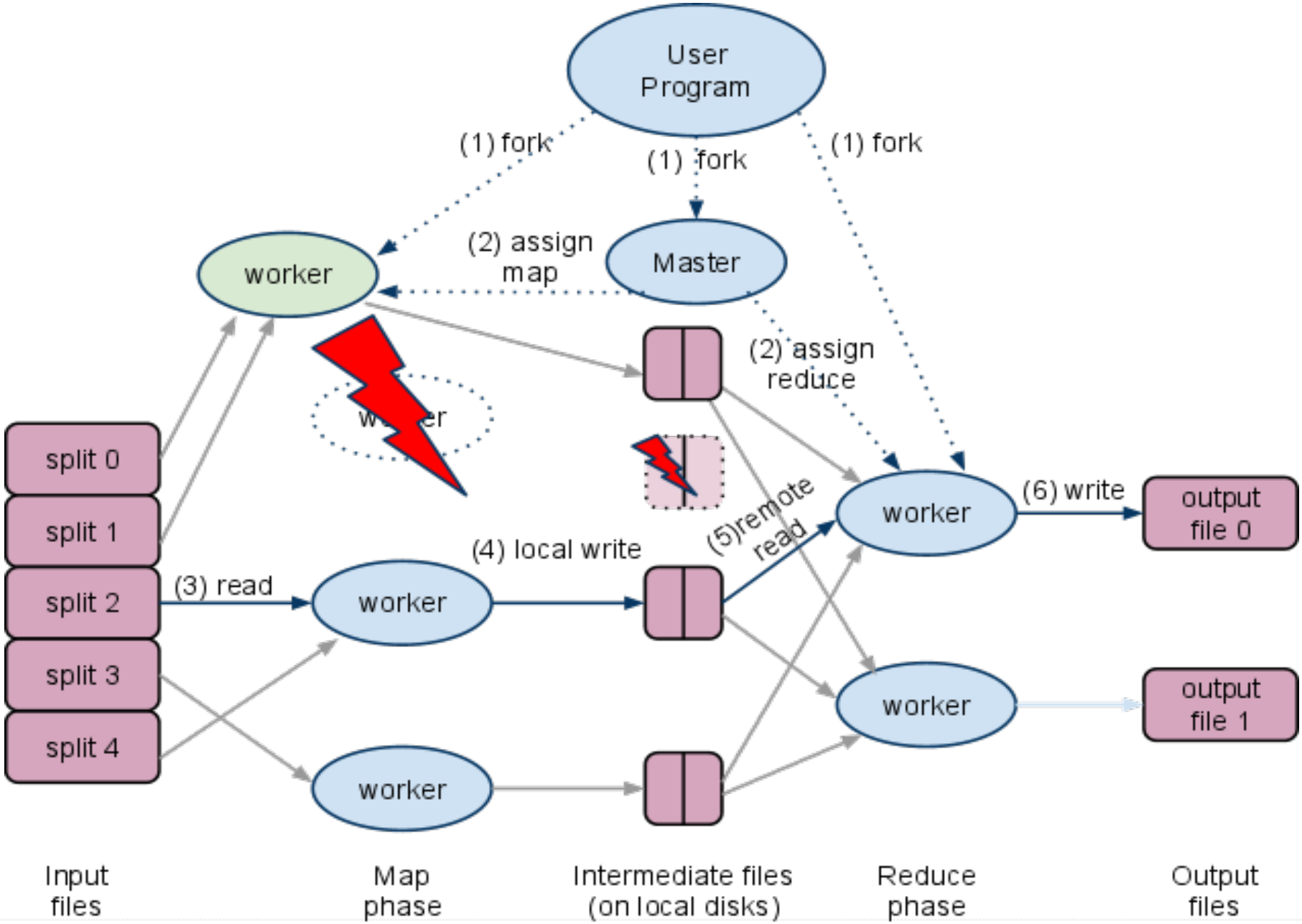
MapReduce: Task Failure

Google



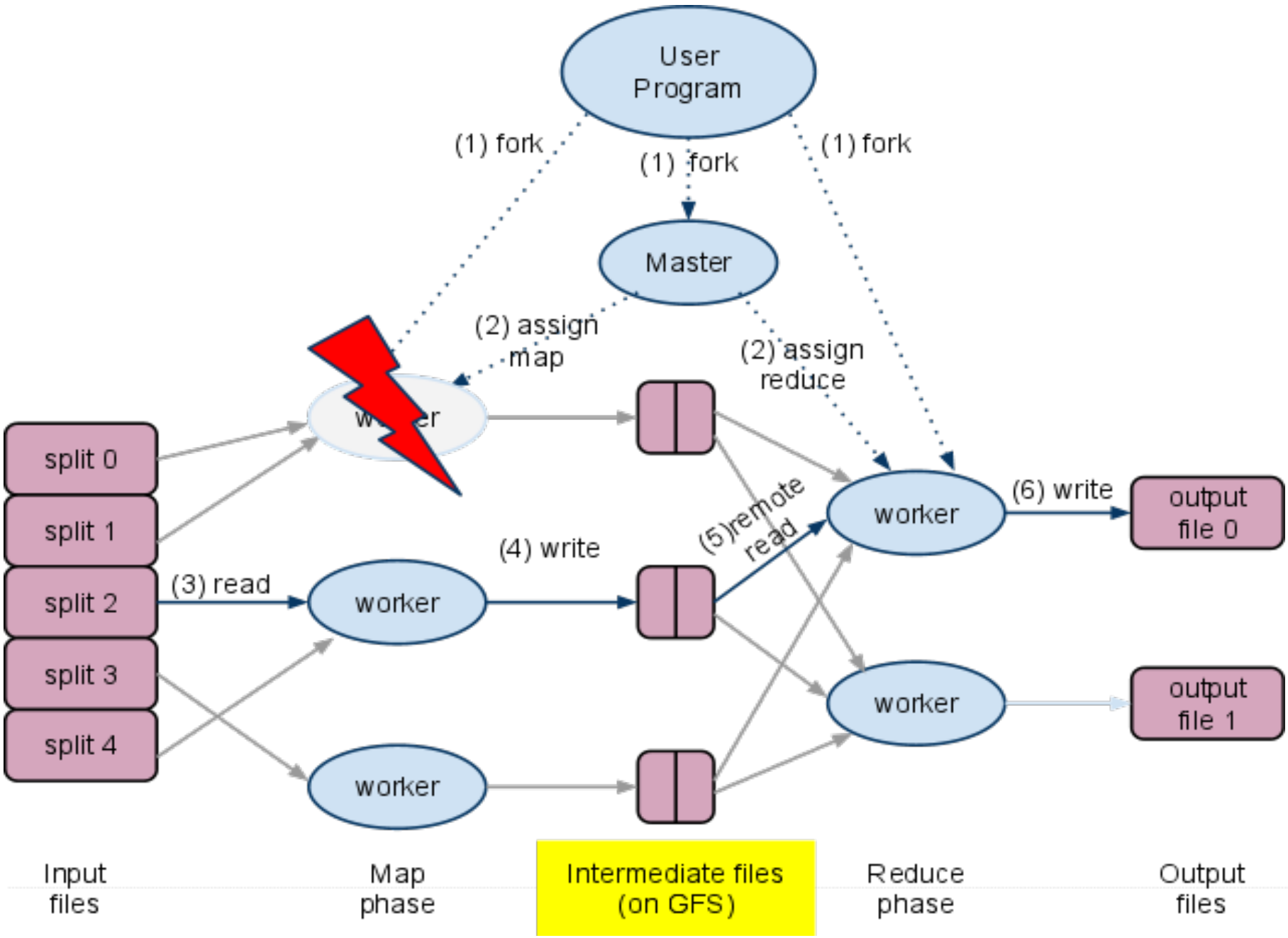
Recover from Task Failure by Re-execution

Google



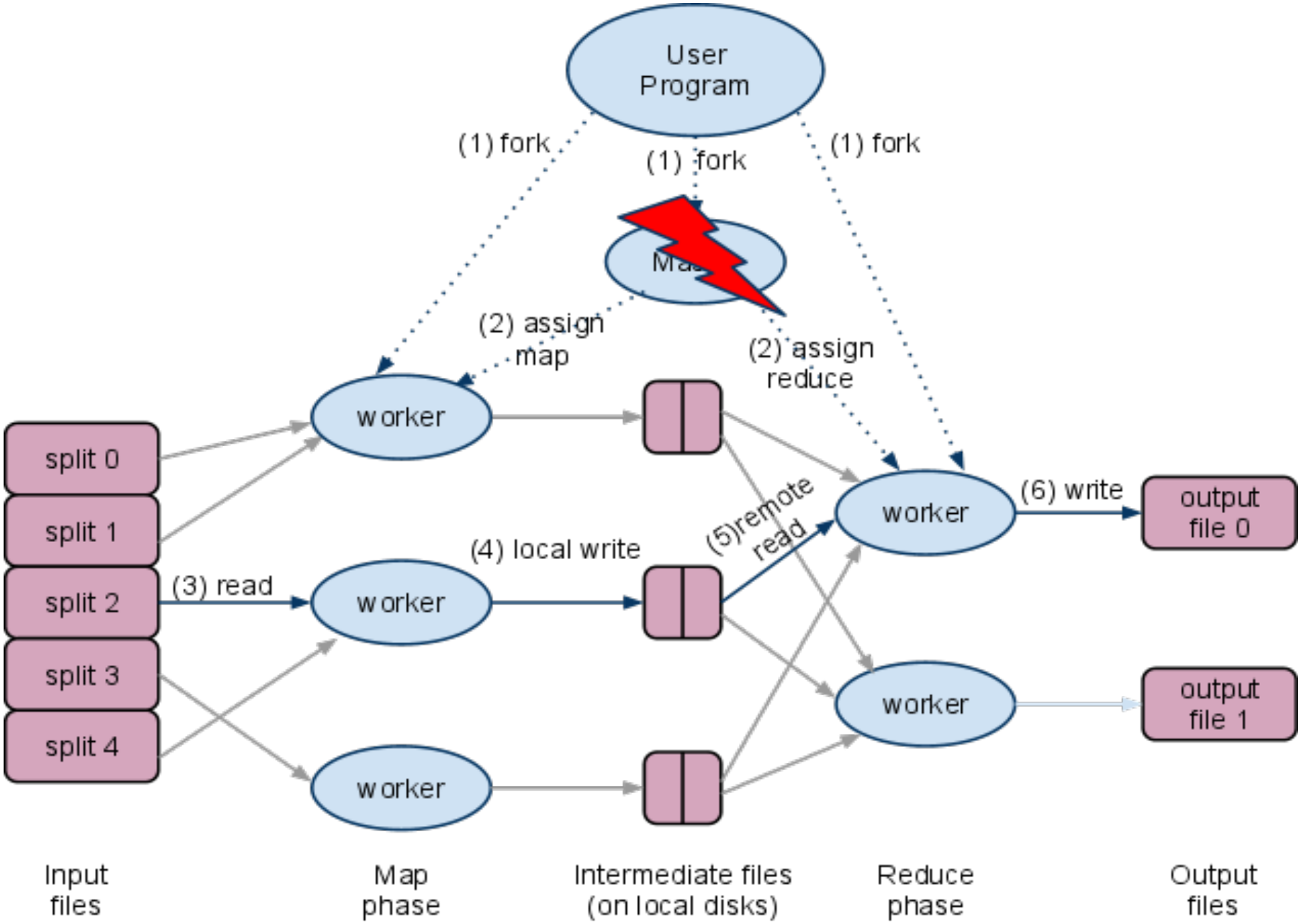
Recover by Checkpointing Map Output

Google



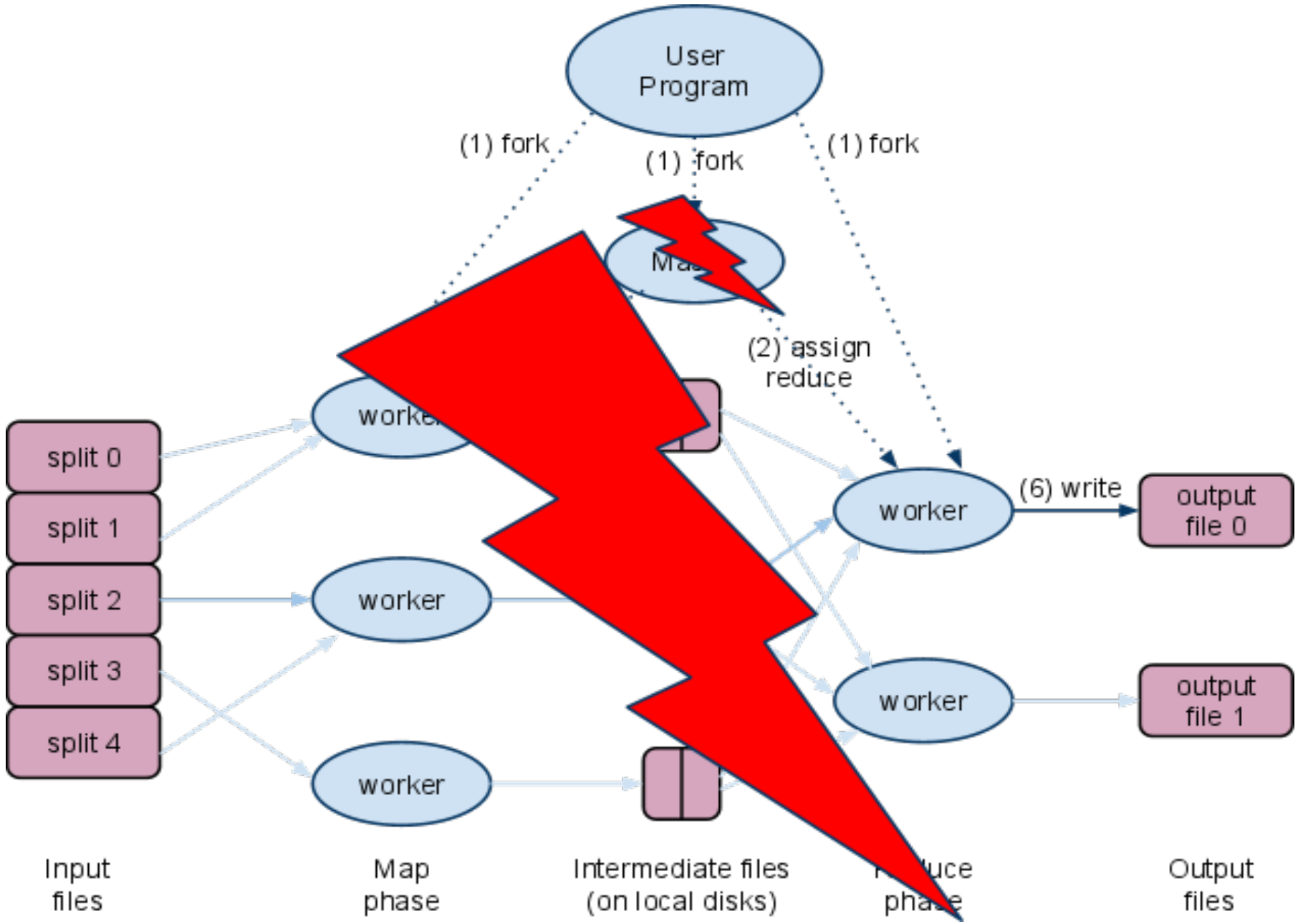
MapReduce: Master Failure

Google



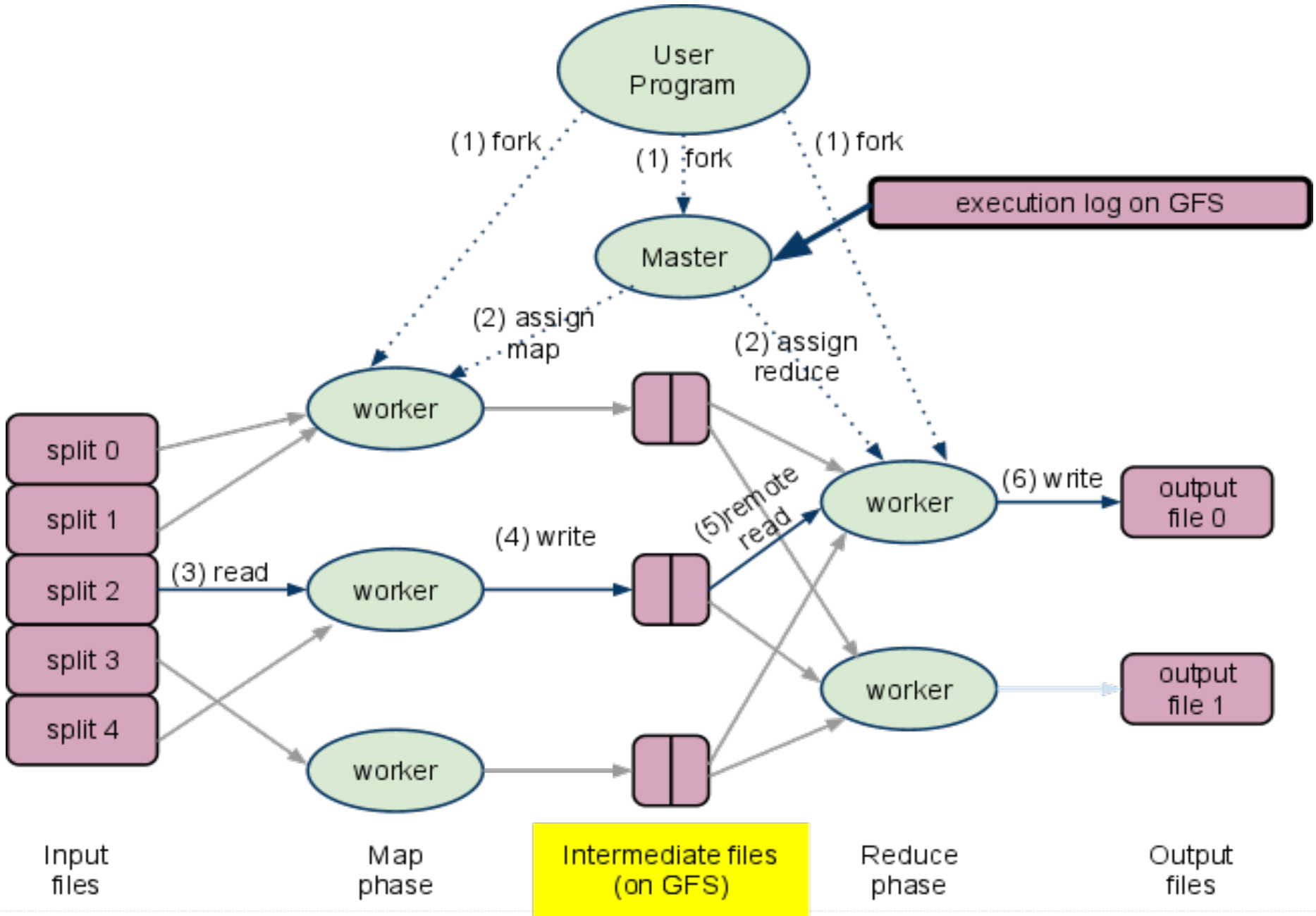
Master as a Single Point of Failure

Google



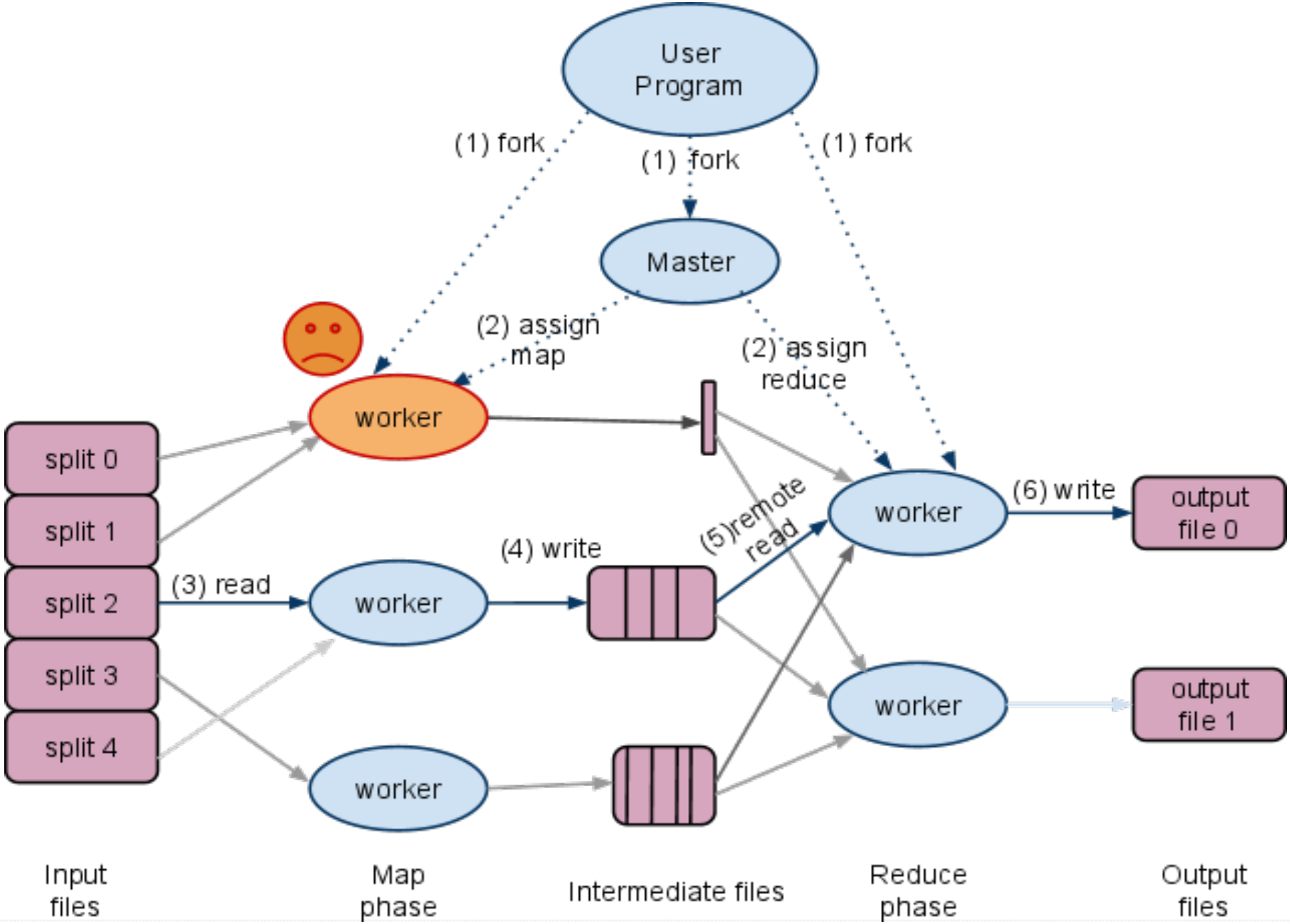
Resume from Execution Log on GFS

Google



MapReduce: Slow Worker/Task

Google



Handle Unfixable Failures

Google

- Input data is in a partially wrong format or is corrupted
  - Data is mostly well-formatted, but there are instances where your code crashes
  - Corruptions happen rarely, but they are possible at scale
- Your application depends on an external library which you do not control
  - Which happens to have a bug for a particular, yet very rare, input pattern
- What would you do?
  - Your job is critical to finish as soon as possible
  - The problematic records are very rare
  - IGNORE IT!

Tutorial Overview

Google

- MapReduce programming model
  - Brief intro to MapReduce
  - Use of MapReduce inside Google
  - MapReduce programming examples
  - MapReduce, similar and alternatives
- Implementation of Google MapReduce
  - Dealing with failures
  - **Performance & scalability**
    - Some techniques and tuning tips
    - Dealing with stragglers
  - Usability

Performance and Scalability of MapReduce

Google

Terasort and Petasort with MapReduce in Nov 2008

- Not particularly representative for production MRs
- An important benchmark to evaluate the whole stack
- Sorted 1TB (as 10 billion 100-byte uncompressed text) on 1,000 computers in 68 seconds
- Sorted 1PB (10 trillion 100-byte records) on 4,000 computers in 6 hours and 2 minutes

With Open-source Hadoop in May 2009 (TechReport)

- Terasort: 62 seconds on 1460 nodes
- Petasort: 16 hours and 15 minutes on 3658 nodes

Built up on Great Google Infrastructure

Google

Google MapReduce is built upon an set of high performance infrastructure components:

- Google file system (GFS) (SOSP'03)
- Chubby distributed lock service (OSDI'06)
- Bigtable for structured data storage (OSDI'06)
- Google cluster management system
- Powerful yet energy efficient* hardware and finetuned platform software
- Other house-built libraries and services

Take Advantage of Locality Hints from GFS

Google

- Files in GFS
  - Divided into chunks (default 64MB)
  - Stored with replications, typical r=3
  - Reading from local disk is much faster and cheaper than reading from a remote server
- MapReduce uses the locality hints from GFS
  - Try to assign a task to a machine with a local copy of input
  - Or, less preferable, to a machine where a copy stored on a server on the same network switch
  - Or, assign to any available worker

Tuning Task Granularity

Google

Questions often asked in production:

- How many Map tasks I should split my input into?
- How many Reduce splits I should have?

Implications on scalability

- Master has to make O(M+R) decisions
- System has to keep O(M*R) metadata for distributing map output to reducers

To balance locality, performance and scalability

- By default, each map task is 64MB (== GFS chunksize)
- Usually, #reduce tasks is a small multiple of #machine

More on Map Task Size

Google

- Small map tasks allow fast failure recovery
  - Define "small": input size, output size or processing time
- Big map tasks may force mappers to read from multiple remote chunkservers
- Too many small map shards might lead to excessive overhead in map output distribution

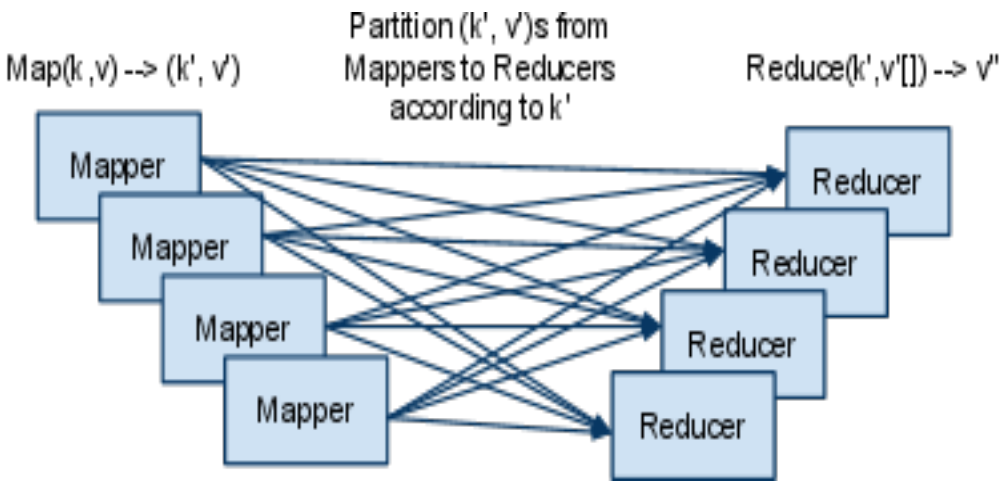
Reduce Task Partitioning Function

Google

It is relatively easy to control Map input granularity

- Each map task is independent

For Reduce tasks, we can tweak the partitioning function instead.



Reduce key	Reduce input size
*.blogspot.com	82.9G
cgi.ebay.com	58.2G
profile.myspace.com	56.3G
yellowpages.superpages.com	49.6G
www.amazon.co.uk	41.7G
average reduce input size for a given key	300K

Tutorial Overview

Google

- MapReduce programming model
  - Brief intro to MapReduce
  - Use of MapReduce inside Google
  - MapReduce programming examples
  - MapReduce, similar and alternatives
- Implementation of Google MapReduce
  - Dealing with failures
  - Performance & scalability
    - **Dealing with stragglers**
  - Usability

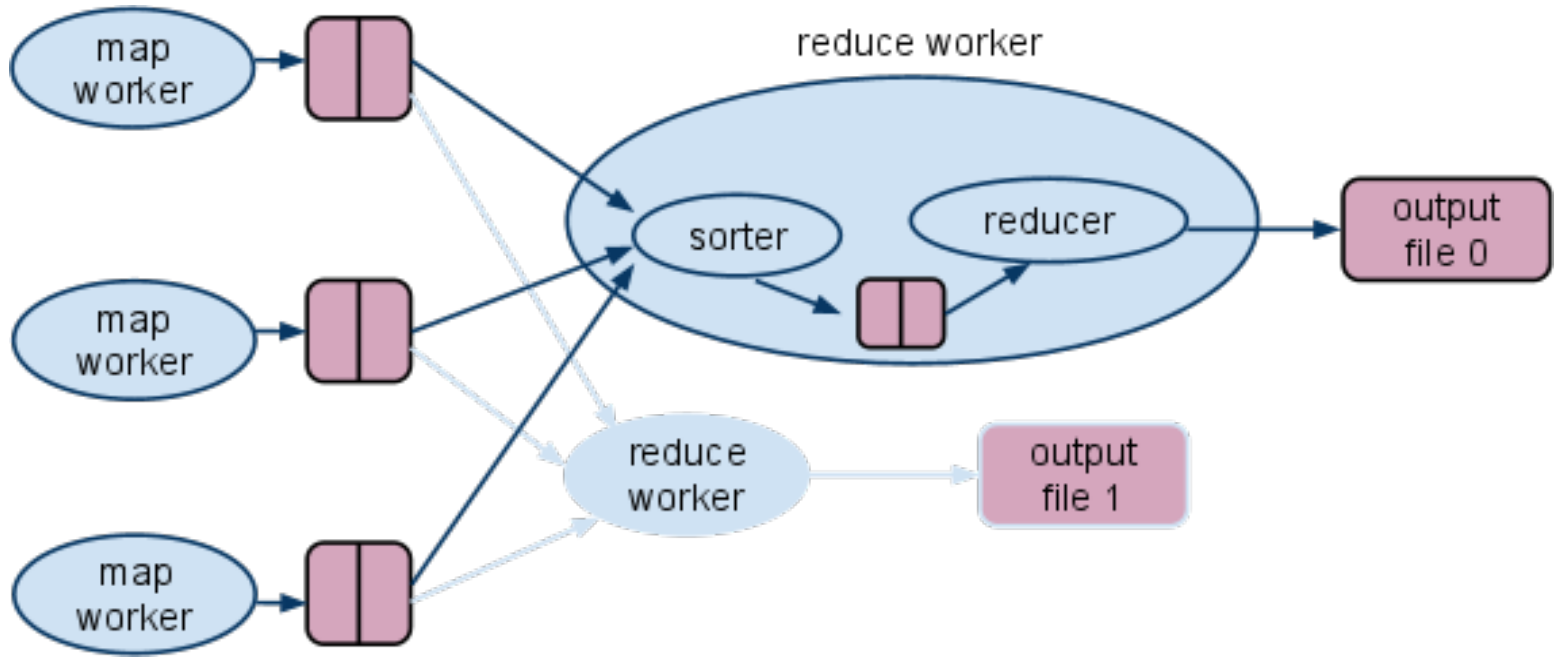
Dealing with Reduce Stragglers

Google

Many reason leads to stragglers but reducing is inherently expensive:

- Reducer retrieves data remotely from many servers
- Sorting is expensive on local resources
- Reducing usually can not start until Mapping is done

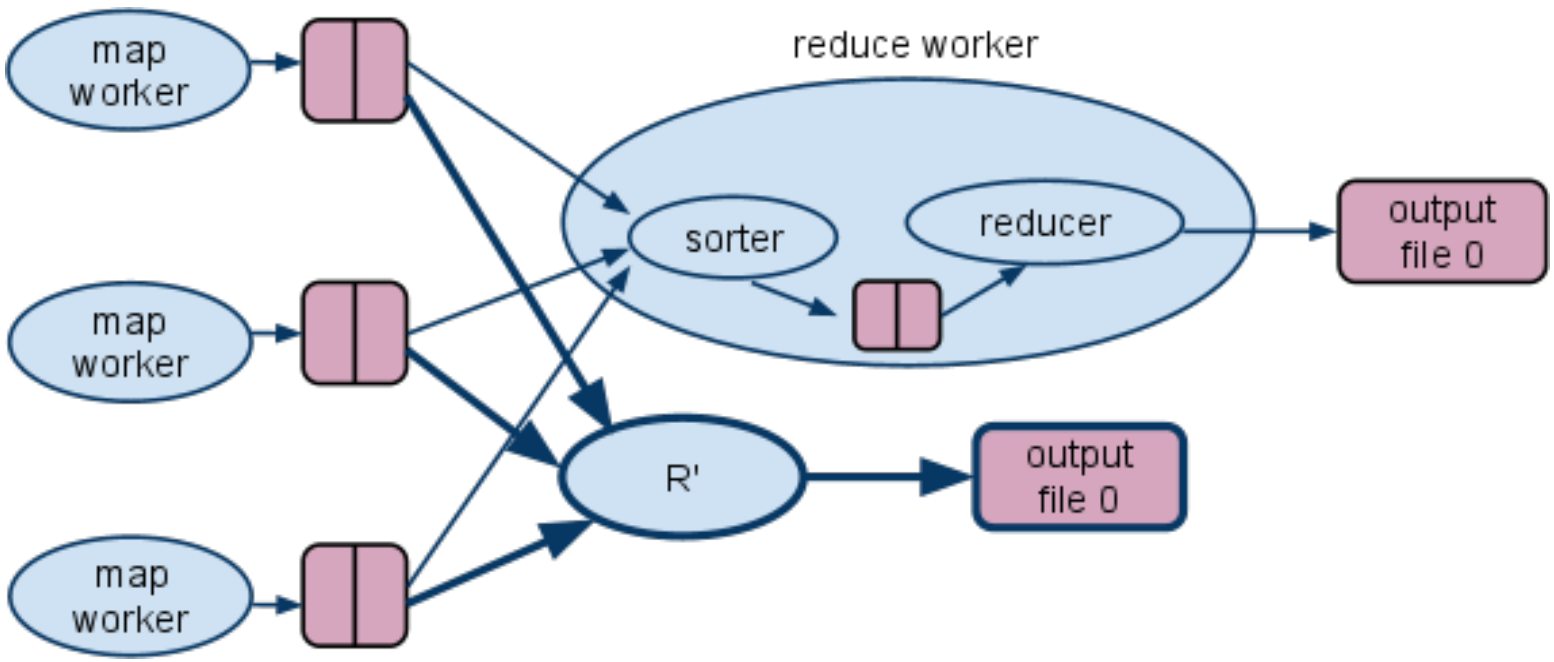
Re-execution due to machine failures could double the runtime.



Dealing with Reduce Stragglers

Google

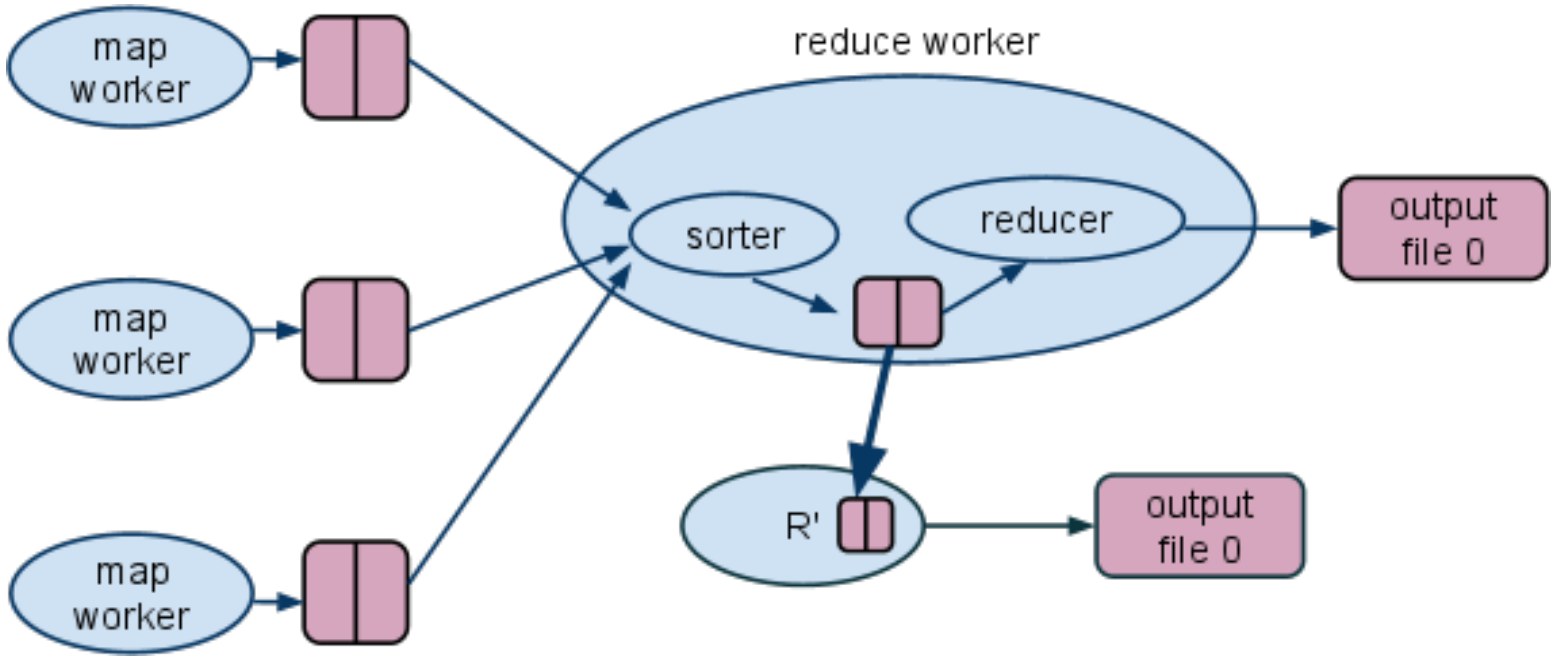
Technique 1:
Create a backup instance as early and as necessary as possible



Steal Reduce Input for Backups

Google

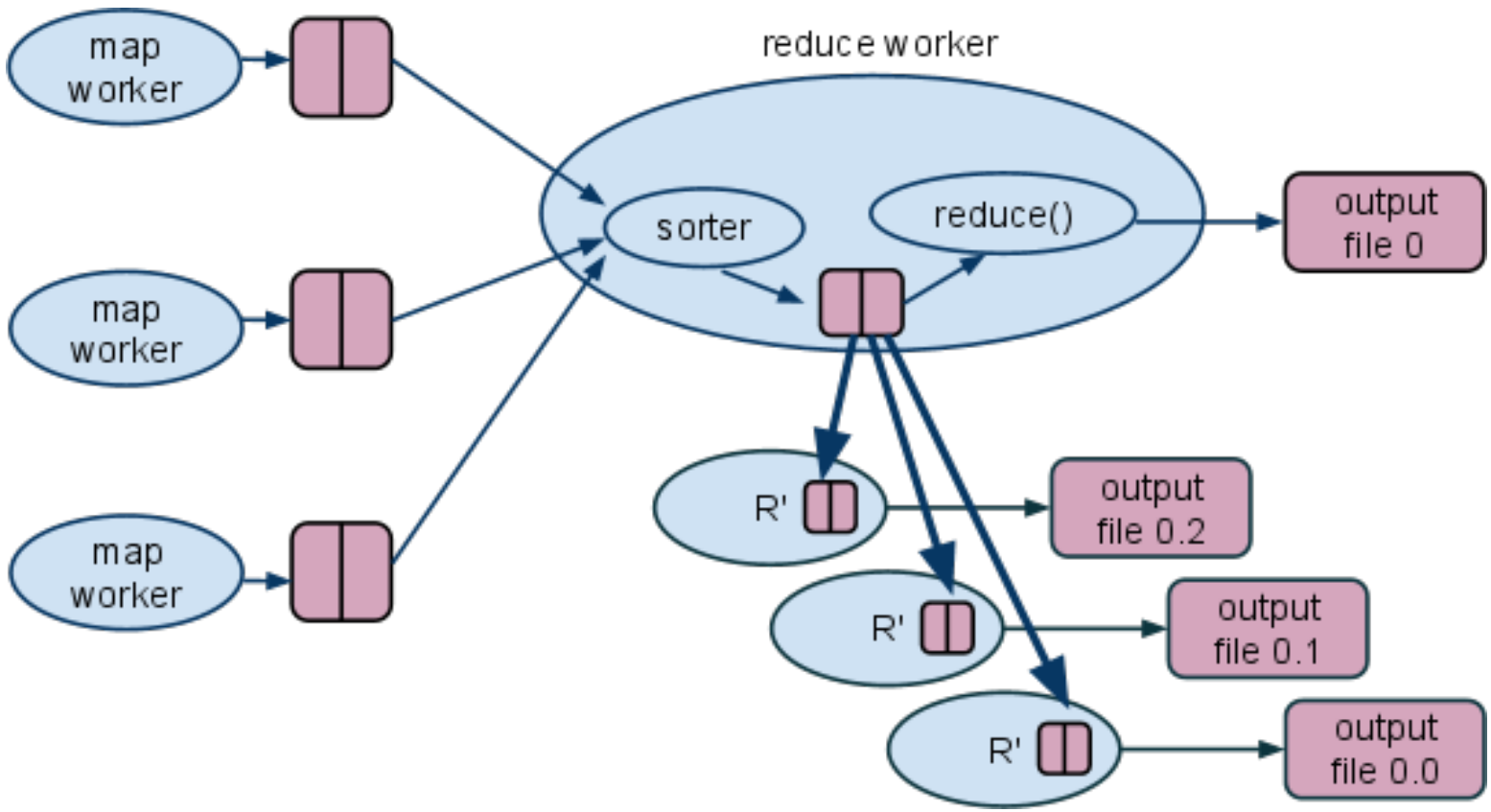
Technique 2:
Retrieving map output and sorting are expensive, but we can transport the sorted input to the backup reducer



Reduce Task Splitting

Google

Technique 3:
Divide a reduce task into smaller ones to take advantage of more parallelism.



Tutorial Overview

Google

- MapReduce programming model
  - Brief intro to MapReduce
  - Use of MapReduce inside Google
  - MapReduce programming examples
  - MapReduce, similar and alternatives
- Implementation of Google MapReduce
  - Dealing with failures
  - Performance & scalability
  - **(Operational) Usability**
    - monitoring, debugging, profiling, etc.

Tools for Google MapReduce

Google

Local run mode for debugging/profiling MapReduce applications

Status page to monitor and track progress of MapReduce executions, also

- Email notification
- Replay progress postmortem

Distributed counters used by MapReduce library and application for validation, debugging and tuning

- System invariant
- Performance profiling

MapReduce Counters

Google

Light-weighted stats with only "increment" operations

- per task counters: contributed by each M/R task
  - only counted once even there are backup instances
- per worker counters: contributed by each worker process
  - aggregated contributions from all instances
- Can be easily added by developers

Examples:

- num_map_output_records == num_reduce_input_records
- CPU time spend in Map() and Reduce() functions

MapReduce Development inside Google

Google

Support C++, Java, Python, Sawzall, etc.

Nurtured greatly by Google engineer community

- Friendly internal user discussion groups
- Fix-it! instead of complain-about-it! attitude
- Users contribute to both the core library and contrib
  - Thousands of Mapper Reducer implementations
  - Tens of Input/Output formats
  - Endless new ideas and proposals

Summary

Google

- MapReduce is a flexible programming framework for many applications through a couple of restricted Map() /Reduce() constructs
- Google invented and implemented MapReduce around its infrastructure to allow our engineers scale with the growth of the Internet, and the growth of Google products/services
- Open source implementations of MapReduce, such as Hadoop are creating a new ecosystem to enable large scale computing over the off-the-shelf clusters
- So happy MapReducing!

Google

Thank you!