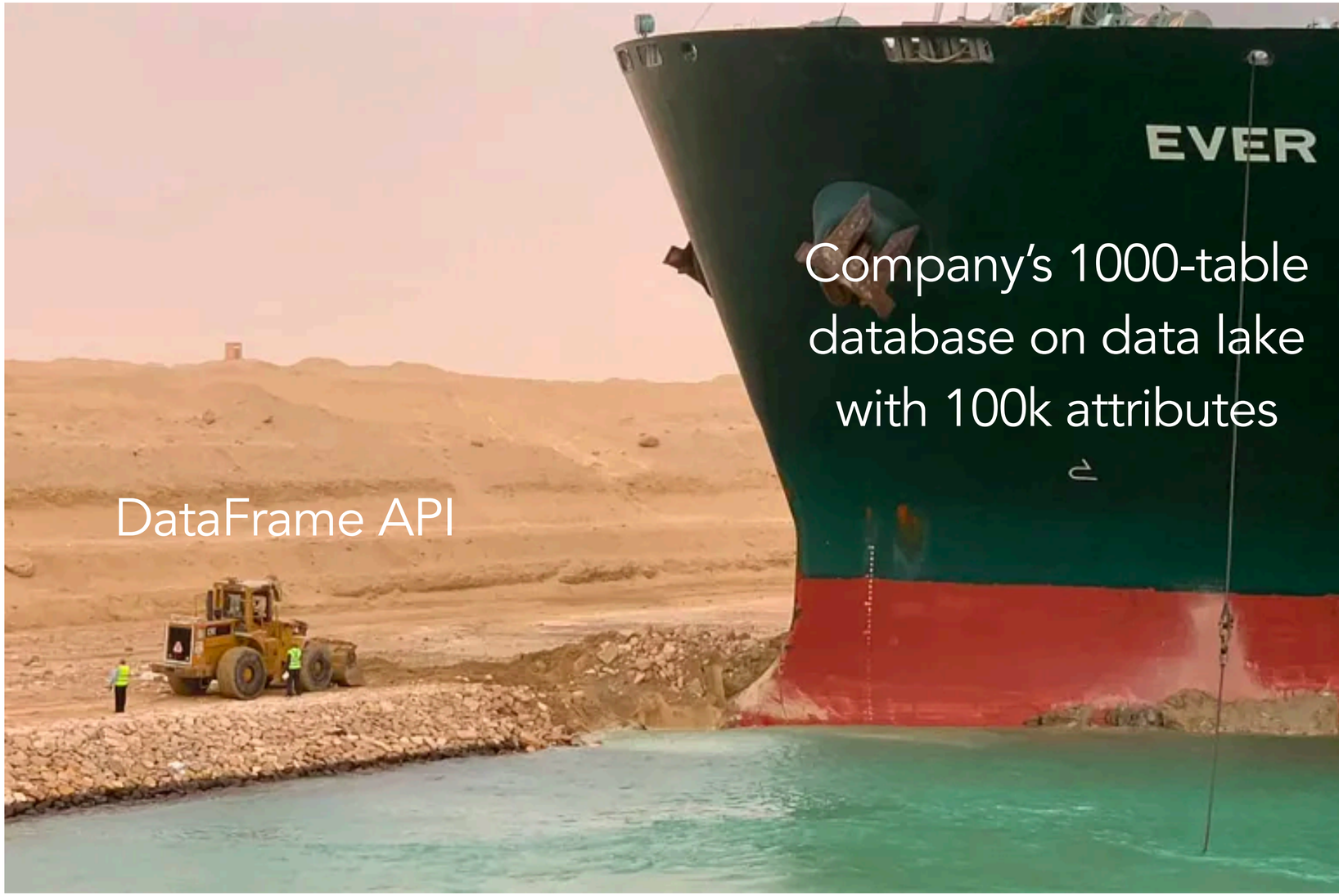
DSC 204a Scalable Data Systems

- Haojian Jin



Meme idea credit: https://datasystemsfun.tumblr.com/

Where are we in the class?

Foundations of Data Systems (2 weeks)

- Digital representation of Data → Computer Organization → Memory hierarchy → Process → Storage

Scaling Distributed Systems (3 weeks)

- Cloud → Network → Distributed storage → Parallelism → Partition and replication

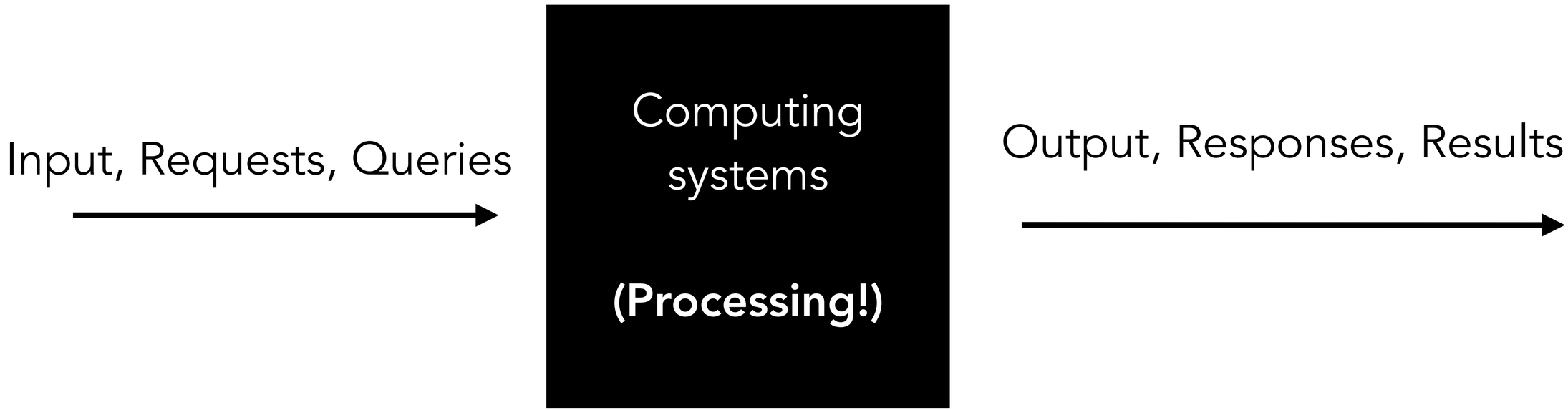
Data Processing and Programming model (5 weeks)

- Data Models evolution → Data encoding evolution → **IO & Unix Pipes** → Batch processing (MapReduce) → Stream processing (Spark)

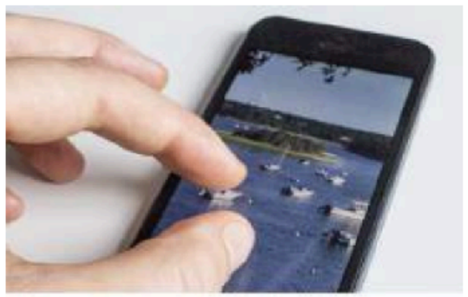
Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
- Beyond MapReduce

Basic Computing System Paradigm



Processing latency



feedback cycle time

Interactive data science!

direct manipulation
no visible lag

Online system: handle request ASAP

turn-taking
minutes to seconds

Stream processing systems
(near-real-time systems)

batch-processing
hours or overnight

Batch processing systems
(Offline systems)

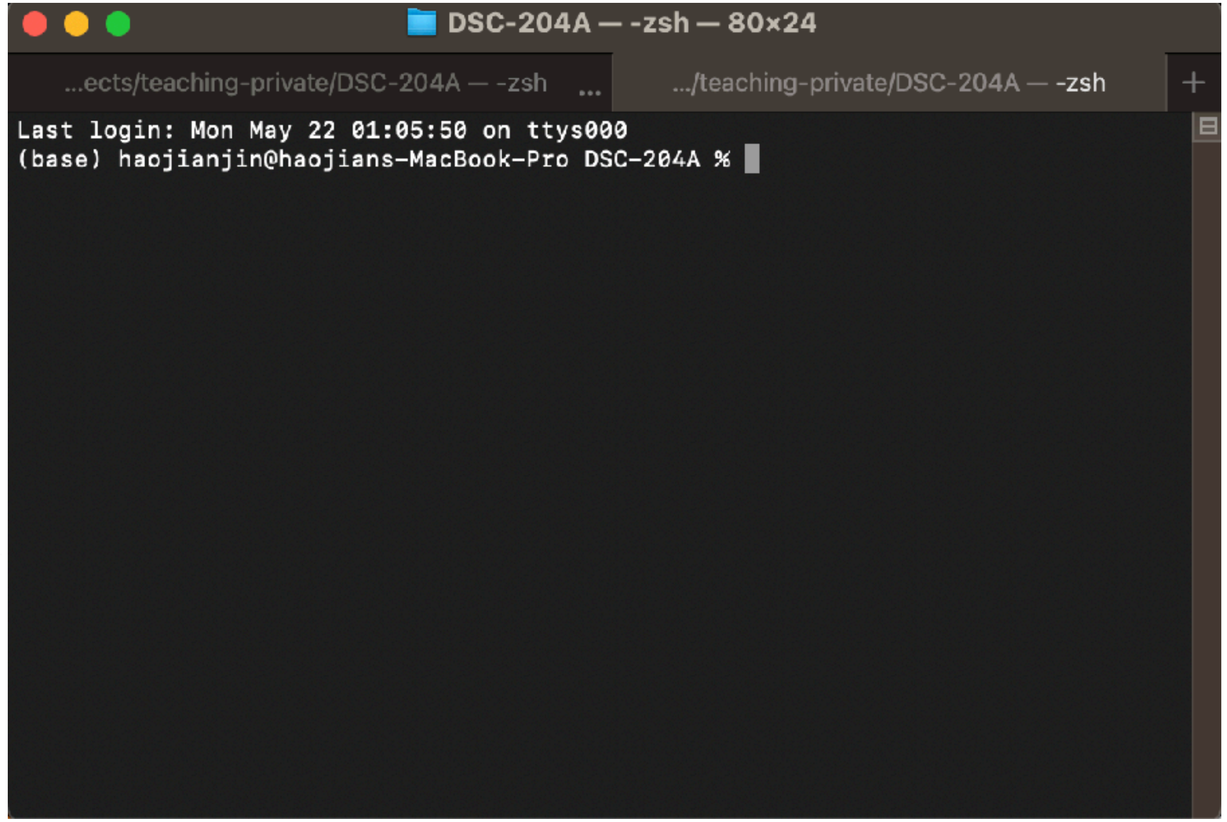
Figure from https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/10090/file/mueller_diss.pdf

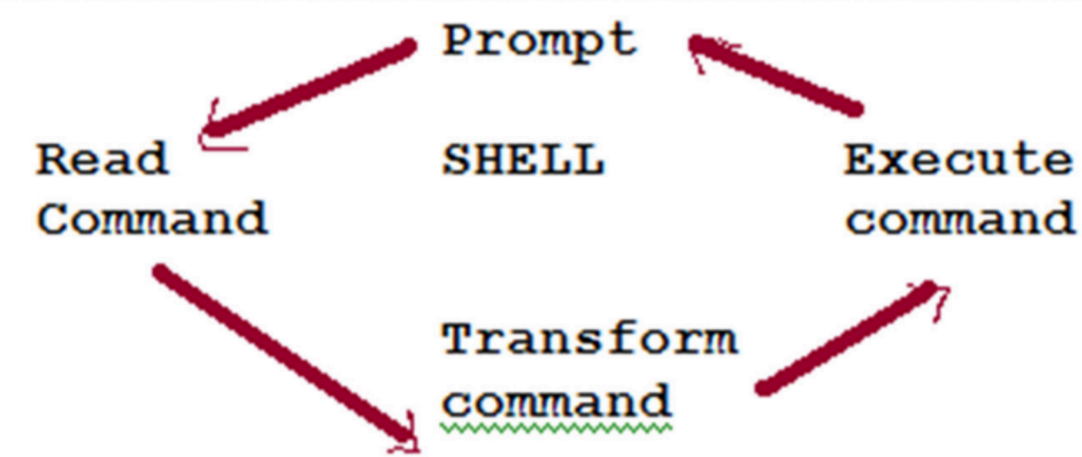
Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
- Beyond MapReduce

The shell

A command line interpreter that provides the interface to Unix OS.





Shell example

```
(base) haojianjin@haojians-MacBook-Pro ~ % ls -lah  ~/.
total 168
drwxr-x---+   46 haojianjin  staff   1.4K May 22 12:15 .
drwxr-xr-x     5 root        admin   160B Dec 20 10:26 ..
-r--------     1 haojianjin  staff     7B Dec 20 07:38 .CFUserTextEncoding
-rw-r--r--@    1 haojianjin  staff    14K May 22 11:57 .DS_Store
drwx------+ 1251 haojianjin  staff    39K May 22 09:43 .Trash
drwxr-xr-x     3 haojianjin  staff    96B Dec 30 20:34 .aws
-rw-------     1 haojianjin  staff   783B Apr 25 21:13 .bash_history
-rw-r--r--     1 haojianjin  staff   544B Feb 14 00:19 .bash_profile
drwxr-xr-x     3 haojianjin  staff    96B Dec 26 19:49 .bundle
drwxr-xr-x     3 haojianjin  staff    96B Feb 12 11:17 .cache
drwxr-xr-x     4 haojianjin  staff   128B Feb 14 00:19 .config
drwx------     3 haojianjin  staff    96B Dec 20 19:34 .cups
drwxr-xr-x    12 haojianjin  staff   384B Dec 26 20:05 .gem
-rw-r--r--     1 haojianjin  staff   222B Dec 30 20:24 .gitconfig
drwxr-xr-x     5 haojianjin  staff   160B Feb 14 00:23 .ipython
drwxr-xr-x     3 haojianjin  staff    96B Feb 14 15:16 .jupyter
-rw-------     1 haojianjin  staff    20B Apr 14 13:35 .lesshst
drwxr-xr-x     4 haojianjin  staff   128B Feb 12 11:28 .local
drwxr-xr-x     3 haojianjin  staff    96B Feb 14 00:25 .matplotlib
drwxr-xr-x     6 haojianjin  staff   192B Mar  6 14:14 .npm
-rw-------     1 haojianjin  staff     7B Feb 14 00:20 .python_history
drwxr-xr-x     3 haojianjin  staff    96B Mar  5 18:31 .rbenv
drwxr-xr-x     3 haojianjin  staff    96B Feb 12 11:25 .rubies
drwx------     7 haojianjin  staff   224B Apr  2 22:27 .ssh
-rw-r--r--     1 haojianjin  staff   332B Feb 14 00:19 .tcshrc
-rw-------     1 haojianjin  staff   9.6K May 17 14:41 .viminfo
drwxr-xr-x     4 haojianjin  staff   128B Dec 22 23:27 .vscode
-rw-r--r--     1 haojianjin  staff   615B Feb 14 00:19 .xonshrc
-rw-r--r--     1 haojianjin  staff   116B Mar  6 14:05 .yarnrc
-rw-------     1 haojianjin  staff    13K May 21 13:43 .zsh_history
drwx------    32 haojianjin  staff   1.0K May 22 12:15 .zsh_sessions
-rw-r--r--     1 haojianjin  staff   713B Feb 14 00:19 .zshrc
```

"**.****rc**"

Run commands

What's Shell good for?

- Starting and stopping processes
  - Controlling the terminal
  - Interacting with unix system
  - Solving complex problems with simple scripts
  - Life saver for system administrators
- What is a "shell script" ?
  - A collection of shell commands supported by control statements
  - Shell scripts are interpreted and instructions executed

A Shell Script

#!/bin/sh

– above line should always be the first line in your script

A simple script

who am I

Date

- Execute with *sh first.sh*

Another example

```
#!/bin/sh
# run the script as: sh handin.sh SL/SL1 all.txt

dir=$1
basedir="/afs/andrew/course/15/123/handin"
mkdir -p $basedir"/"$dir
cat $2 |
while read id
 do
  mkdir -p $basedir/$dir/$id
  #cp notdone.txt $basedir/$dir/$id
  fs sa $basedir/$dir/$id $id all
  fs sa $basedir/$dir/$id system:anyuser l
  fs sa $basedir/$dir/$id areece all
  fs sa $basedir/$dir/$id mengh all
  fs sa $basedir/$dir/$id jmburges all
  fs sa $basedir/$dir/$id ylung all
 done
```

Command Line Arguments

- $# - represents the total number of arguments (much like argv) – except command
- $0 - represents the name of the script, as invoked
  - $1, $2, $3, .., $8, $9 - The first 9 command line arguments
  - Use “shift” command to handle more than 9 args
- $* - all command line arguments OR
- $@ - all command line arguments

Example

```
#!/bin/bash
echo $0
echo $1
echo "Street: $2"
echo "City: $3"
```

```
(base) haojianjin@haojians-MBP demo % sh parameters.txt
parameters.txt

Street:
City:
```

```
(base) haojianjin@haojians-MBP demo % sh parameters.txt a b c
parameters.txt
a
Street: b
City: c
```

```
(base) haojianjin@haojians-MBP demo % sh parameters.txt "a b c"
parameters.txt
a b c
Street:
City:
```

Control statements - loops and conditionals

```
for var in "$@"
    do
        printf "%s\n" $var
    done

for (( i = 1 ; i < 20 ; i++ ))
 do

done
```

```
while read file
 do
      echo $file
 done
```

Useful shell commands

- Shell already has a collection of rich commands
  - Some Useful commands
    - uptime, cut, date, cat, finger, hexdump, man, md5sum, quota,
    - mkdir, rmdir, rm, mv, du, df, find, cp, chmod, cd
    - uname, zip, unzip, gzip, tar
    - tr, sed, sort, uniq, ascii
    - Type “man command” to read about shell commands

What do these shell commands do?

- cat dups.txt | sort | uniq
- cat dups.txt | sort -V| uniq
- cat dups.txt | sort -V| uniq > outfile.txt
- tr "a" "e" < z.txt
- cat z.txt | tr a e

Input & Output

- File descriptors
  - Stdin(0), stdout(1), stderror(2)
- Input from Stdin(0)
- Output to stdout
  - read data
  - echo $data
- redirecting rm filename 1>&2

Examples

- echo test > file.txt
- echo test 1>file.txt
- echo test 2>file.txt # To redirect stderr to file.txt
- echo test 1>&2. # echo test >&2
- echo test 2>&1
  - 2> redirects stderr to an (unspecified) file.
  - &1 redirects stderr to stdout.

Interprocess communication

- Communication between processes
- Using Pipes
  - Pipes is the mechanism for IPC
  - ls | sort | echo
  - 4 processes in play
- Each call spans a new process

Example

- Tr: Text replacer.
- cat somefile.txt | tr "a" "e" > somefile.txt
- What does it do?
- What are some of the problems?
- Problems are caused by the way pipes work

How does pipes work

- A finite buffer to allow communication between processes
  - Typically size 8K
- If input file is less than the buffer
  - We may be ok.
- What if input file is more than the buffer
  - Redirecting output to the same file is a bad idea

Use a temp file

- Use a temp file
  - cat file | tr -d "\015" "\012" | fold > file.tmp
  - mv file.tmp file
- A better way
  - cat file| tr -d "\015" "\012" | fold > "/usr/tmp/file.$$" mv "/usr/tmp/file.$$" “file“
  - /usr/tmp is cleared upon reboot

Unix tools v.s. Ruby

- Make each program do one thing well.
- Example:
  - A web server that appends a line to the log file every time it serves a request

```
216.58.210.78 - - [27/Feb/2015:17:55:11 +0000] "GET /css/typography.css HTTP/1.1"
200 3377 "http://martin.kleppmann.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X
10_9_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.115
Safari/537.36"
```

- Schema:

```
$remote_addr - $remote_user [$time_local] "$request"
$status $body_bytes_sent "$http_referer" "$http_user_agent"
```

Batch processing with Unix Tools

```
cat /var/log/nginx/access.log | ❶
  awk '{print $7}' | ❷
  sort             | ❸
  uniq -c          | ❹
  sort -r -n       | ❺
  head -n 5          ❻
```

```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

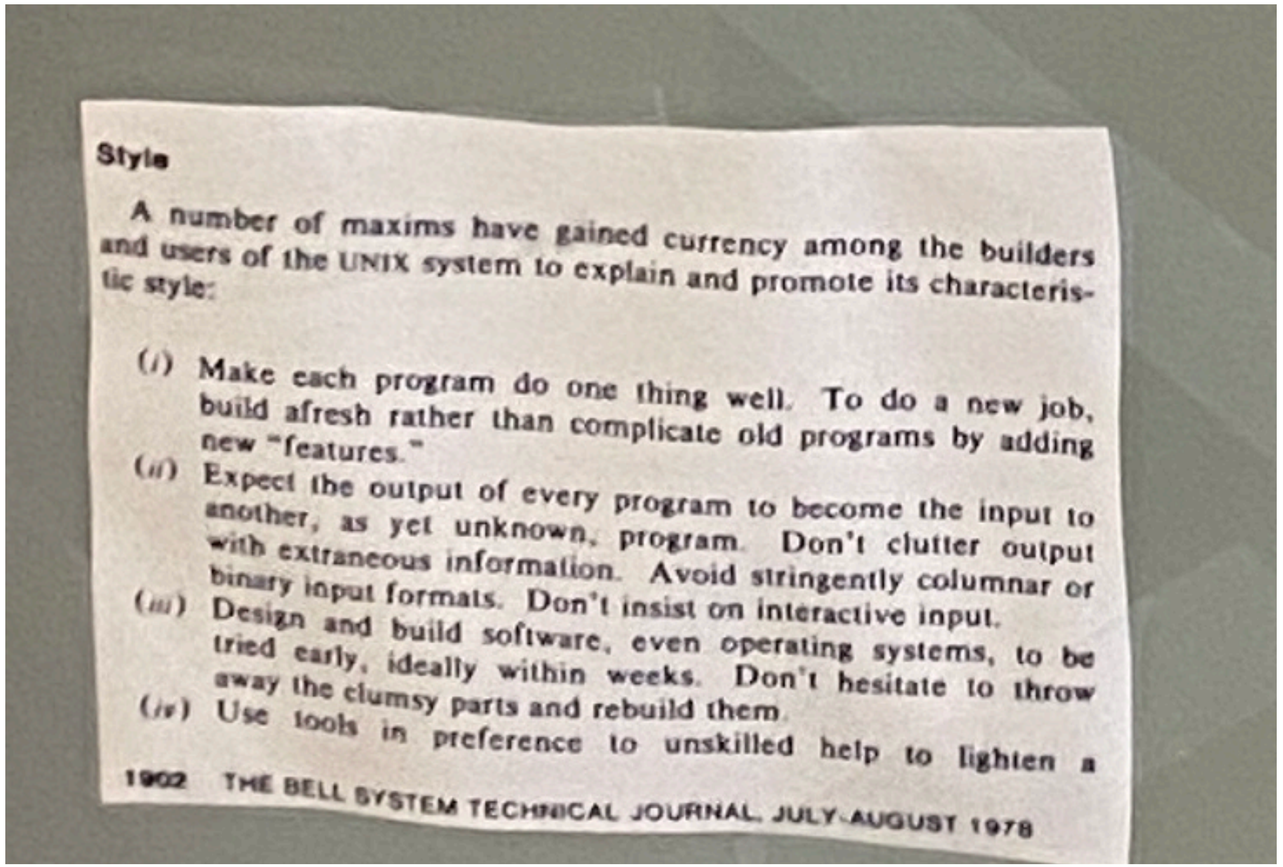
- Read the log file.
- Split each line into fields by white space, output only the 7th element (requested URL).
- Alphabetically sort
- Filter out repeated lines.
- Sort it again based on the line number (-n)
- Out put the first five lines.

Batch processing with custom code

```ruby
counts = Hash.new(0) ❶

File.open('/var/log/nginx/access.log') do |file|
  file.each do |line|
    url = line.split[6] ❷
    counts[url] += 1 ❸
  end
end

top5 = counts.map{|url, count| [count, url] }.sort.reverse[0...5] ❹
top5.each{|count, url| puts "#{count} #{url}" } ❺
```

- Ruby uses a hash_table while Unix relies on sorting.
- The Ruby solution is less scalable, as it is not able to handle large-than-memory datasets.

Unix philosophy



Style

A number of maxims have gained currency among the builders and users of the UNIX system to explain and promote its characteristic style:

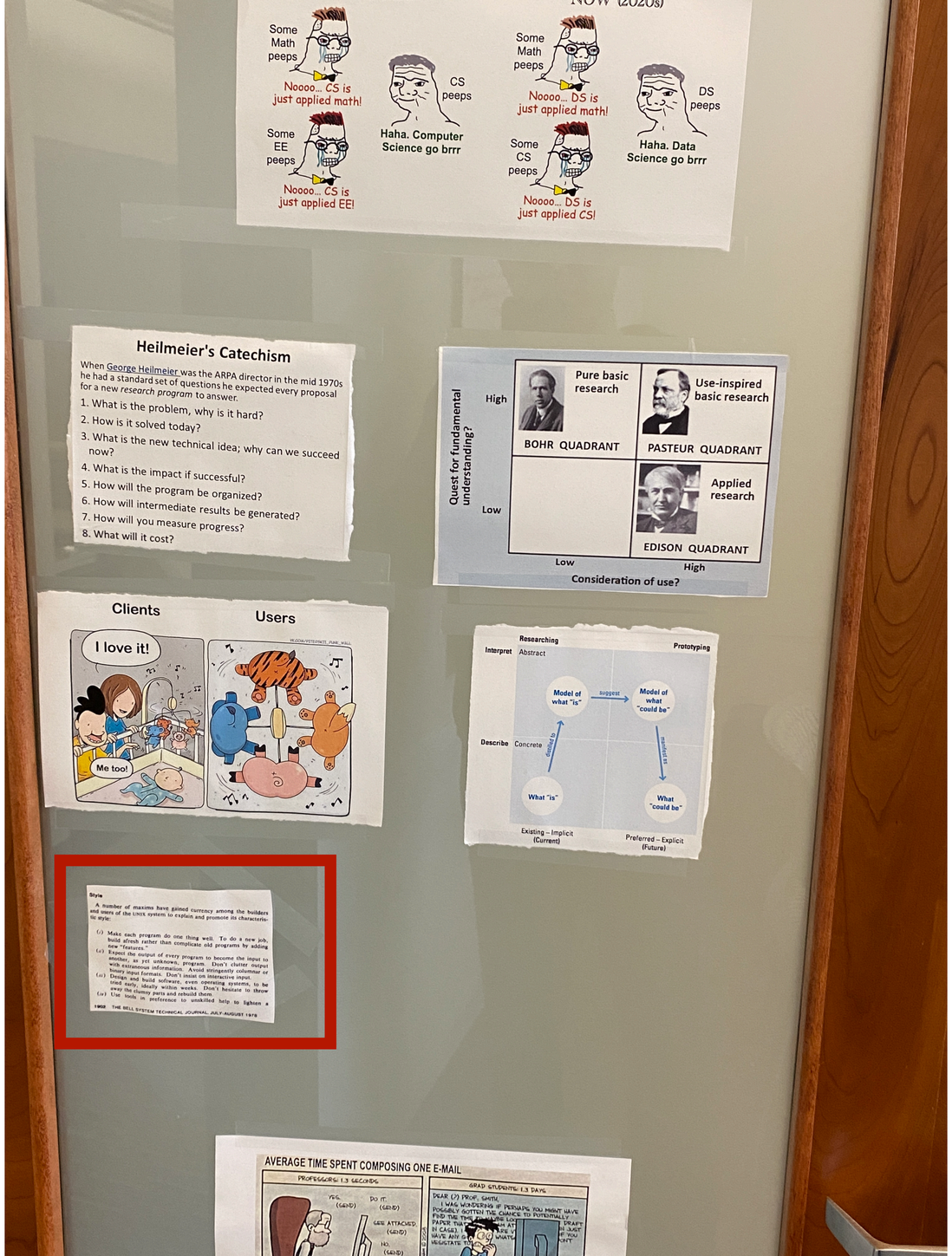
(*i*) Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features."

(*ii*) Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

(*iii*) Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.

(*iv*) Use tools in preference to unskilled help to lighten a

1902 THE BELL SYSTEM TECHNICAL JOURNAL, JULY-AUGUST 1978



Unix philosophy

- Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new “features”.
- Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.
- Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.
- Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.

Unix philosophy

- Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new “features”.
- Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.
- Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.
- Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.

Unix philosophy

- Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new “features”.
- Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.
- Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.
- Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.

Unix philosophy

- Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new “features”.
- Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.
- Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.
- Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.

Transparency and experimentation

- The input files to Unix commands are normally treated as immutable.
  - Run most commands without damaging the input files.
- You can end the pipeline at any point, pipe the output into less, and look at it to see if it has the expected form.
  - Great for debugging.
- You can write the output of one pipeline stage to a file and use that file as input to the next stage.
  - Restart process.

The biggest limitation of Unix tools is that they run only on a single machine — and that's where tools like Hadoop come in.

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Job execution
  - Programming models
  - Workflow
- Beyond MapReduce

Batch processing with Unix Tools

```
cat /var/log/nginx/access.log |  ❶
  awk '{print $7}' |             ❷
  sort             |             ❸
  uniq -c          |             ❹
  sort -r -n       |             ❺
  head -n 5                      ❻
```

```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

- Read the log file.
- Split each line into fields by white space, output only the 7th element (requested URL).
- Alphabetically sort
- Filter out repeated lines.
- Sort it again based on the line number (-n)
- Output the first five lines.

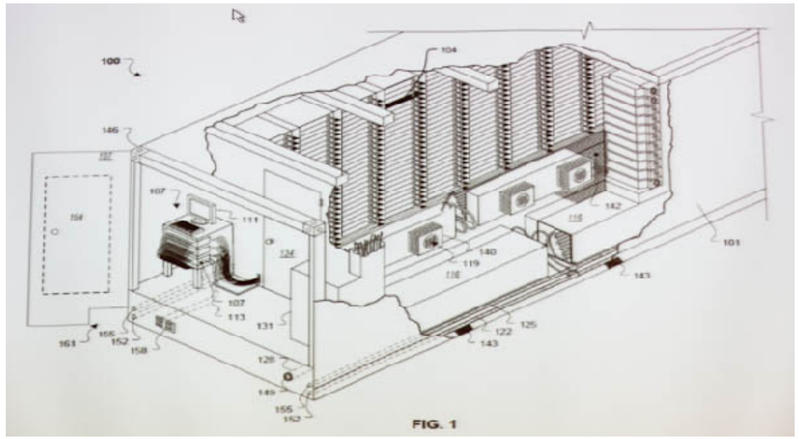
Today's topic: Batch Processing

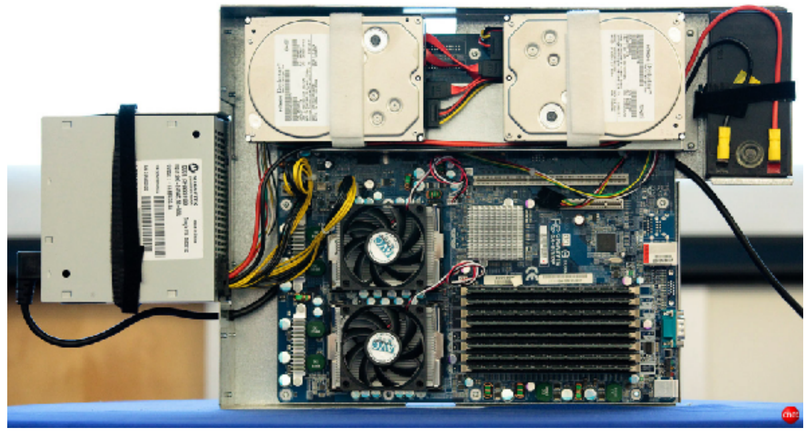
- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models
  - Job execution
  - Workflow
- Beyond MapReduce

Google Data Centers

Dalles, Oregon

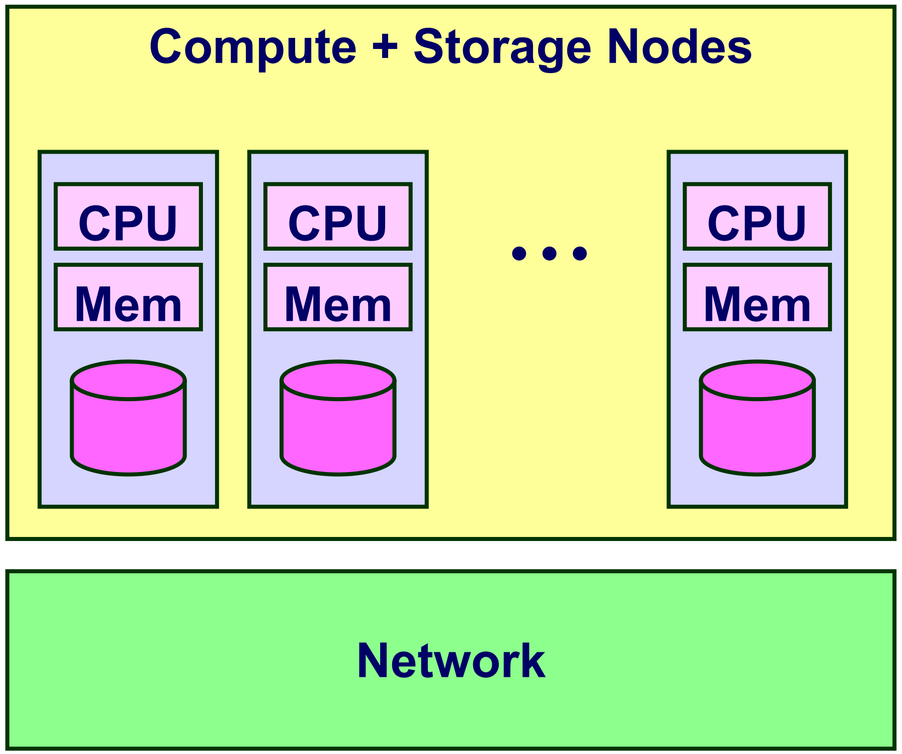
- **Hydroelectric power @ 2¢ / KW Hr**
- **50 Megawatts**
- Enough to power 60,000 homes





- **Engineered for maximum modularity & power efficiency**
- **Container: 1160 servers, 250KW**
- **Server: 2 disks, 2 processors**

Typical Cluster Machine



Compute + Storage Nodes

- Medium-performance processors
- Modest memory
- 1-2 disks

Network

- Conventional Ethernet switches
  - 10 Gb/s within rack
  - 100 Gb/s across racks

Machines with Disks

Lots of storage for cheap

- 3 TB @ $150
  (5¢ / GB)
- Compare 2007:
  0.75 TB @ $266
  35¢ / GB

Drawbacks

- Long and highly variable delays
- Not very reliable

Not included in HPC Nodes

Roll over image to zoom in

WD Black 3TB Performance Desktop Hard Disk Drive - 7200 RPM SATA 6 Gb/s 64MB Cache 3.5 Inch - WD3003FZEX

by Western Digital

1,615 customer reviews | 28 answered questions

List Price: $249.99
Price: $149.99 Prime
You Save: $100.00 (40%)

In Stock.
Ships from and sold by Amazon.com. Gift-wrap available.

Want it Friday, Oct. 30? Order within **55 mins** and choose **Two-Day Shipping** at checkout. Details

Capacity: **3 TB**

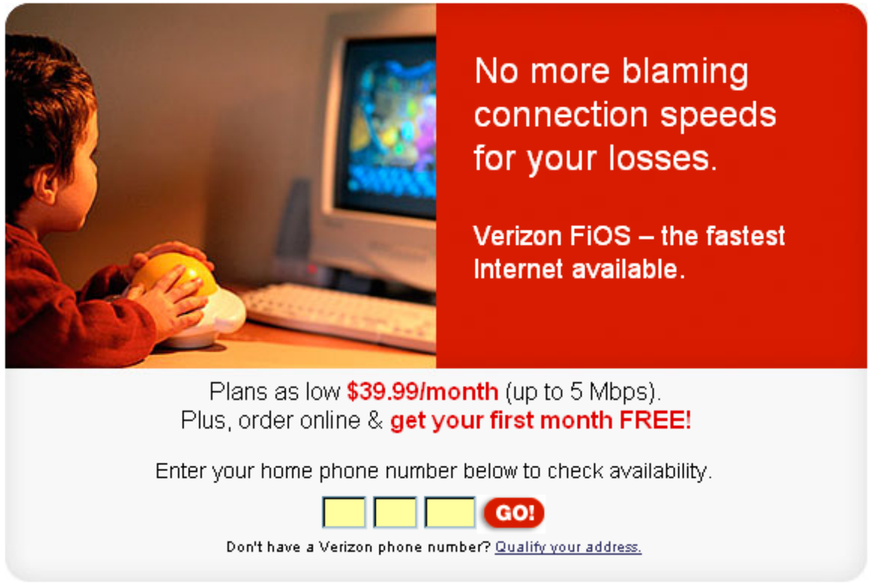
1 TB	2 TB	3 TB	4 TB
$71.79 Prime	$116.99 Prime	$149.99 Prime	$198.00 Prime

5 TB	6 TB
$260.99	$298.91

Oceans of Data, Skinny Pipes

1 Terabyte

- Easy to store
- Hard to move



Seagate

Disks	MB / s	Time
Seagate Barracuda	115	2.3 hours
Seagate Cheetah	125	2.2 hours
Networks	**MB / s**	**Time**
Home Internet	< 0.625	> 18.5 days
Gigabit Ethernet	< 125	> 2.2 hours
PSC Teragrid Connection	< 3,750	> 4.4 minutes

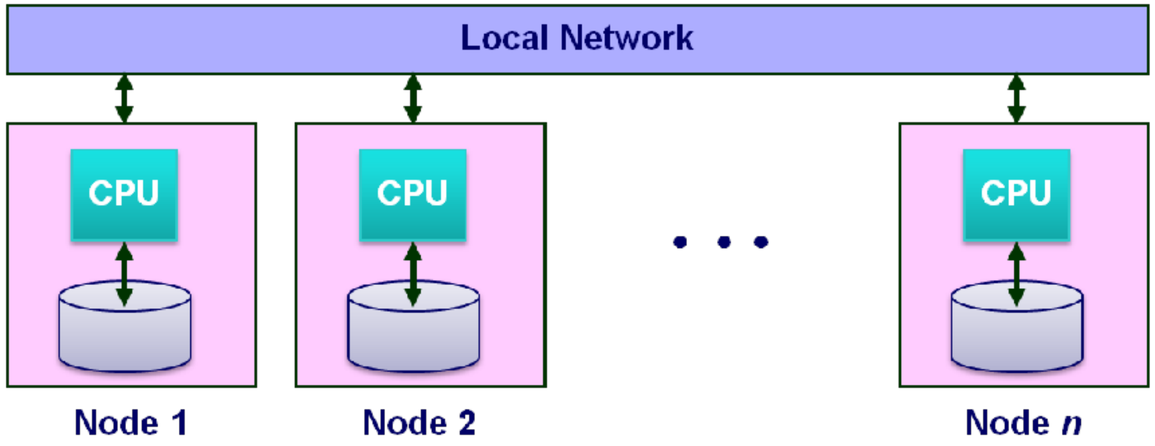
Data-Intensive System Challenge

For Computation That Accesses 1 TB in 5 minutes

- Data distributed over 100+ disks
  - Assuming uniform data partitioning
- Compute using 100+ processors
- Connected by gigabit Ethernet (or equivalent)

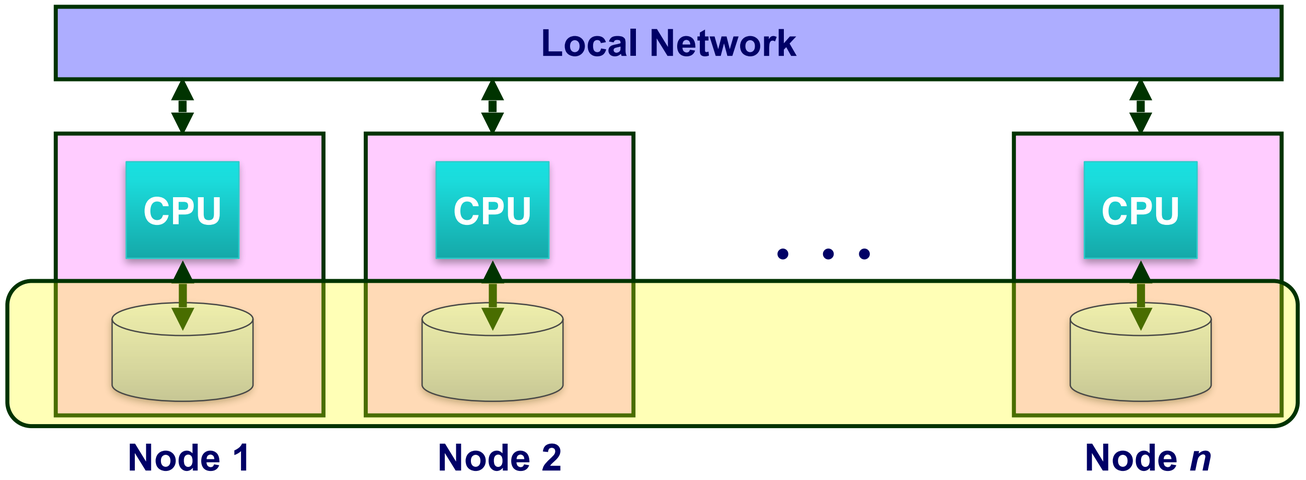
System Requirements

- Lots of disks
- Lots of processors
- Located in close proximity
  - Within reach of fast, local-area network



Hadoop Project

File system with files distributed across nodes

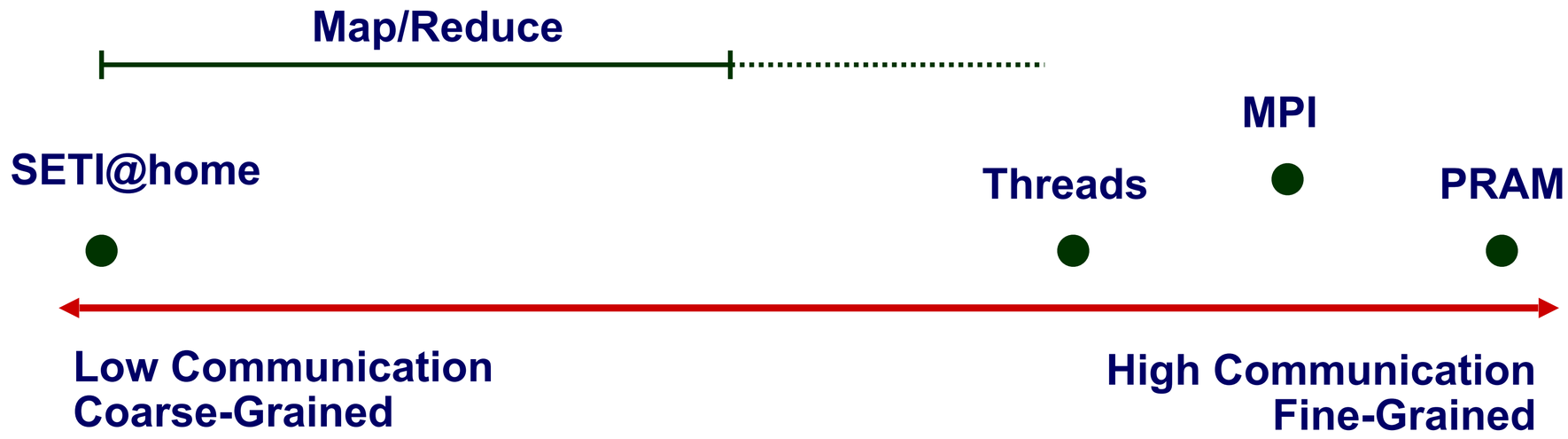


- Store multiple (typically 3 copies of each file)
  - If one node fails, data still available
- Logically, any node has access to any file
  - May need to fetch across network (ideally, leverage locality for perf.)

Map / Reduce programming environment

- Software manages execution of tasks on nodes

Exploring Parallel Computation Models



Map/Reduce Provides Coarse-Grained Parallelism

- Computation done by independent processes
- File-based communication

Observations

- Relatively "natural" programming model
- Research issue to explore full potential and limits

Cluster Scalability Advantages

- Distributed system design principles lead to scalable design
- Dynamically scheduled tasks with state held in replicated files

Provisioning Advantages

- Can use consumer-grade components
  - maximizes cost-peformance
- Can have heterogenous nodes
  - More efficient technology refresh

Operational Advantages

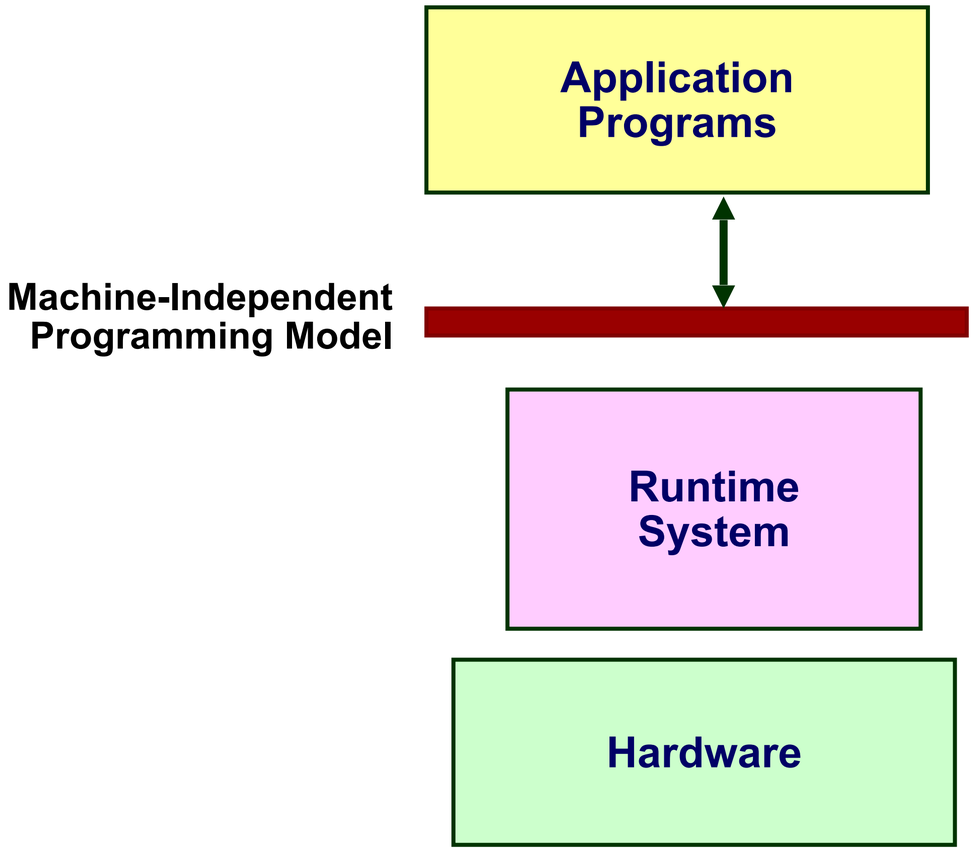
- Minimal staffing
- No downtime

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
- Beyond MapReduce

Ideal Cluster Programming Model

- Application programs written in terms of high-level operations on data
- Runtime system controls scheduling, load balancing, …



MapReduce: Simplified Data Processing on Large Clusters

by Jeffrey Dean and Sanjay Ghemawat

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a *map* and a *reduce* function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day.

TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

Term frequency

Number of times term t appears in a doc, d

Inverse document frequency

$$\log \frac{1 + n}{1 + \text{df}(d, t)} + 1$$

n ← # of documents

df(d, t) ← Document frequency of the term t

TF-IDF Examples

Text 1	i love natural language processing but i hate python
Text 2	i like image processing
Text 3	i like signal processing and image processing

Term	and	but	hate	i	image	language	like	love	natural	processing	python	signal
IDF	0.47712	0.47712	0.4771	0	0.1760913	0.477121	0.1760913	0.477121	0.47712125	0	0.477121	0.477121

	and	but	hate	i	image	language	like	love	natural	processing	python	signal
Text 1	0	0.47712	0.4771	0	0	0.477121	0	0.477121	0.47712125	0	0.477121	0
Text 2	0	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0
Text 3	0.47712	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0.477121

Count the number of occurrences of word in a large collection of documents

```
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 result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

- Functional programming
- Functions are stateless
- They takes an input, processes and output a result.

Data models

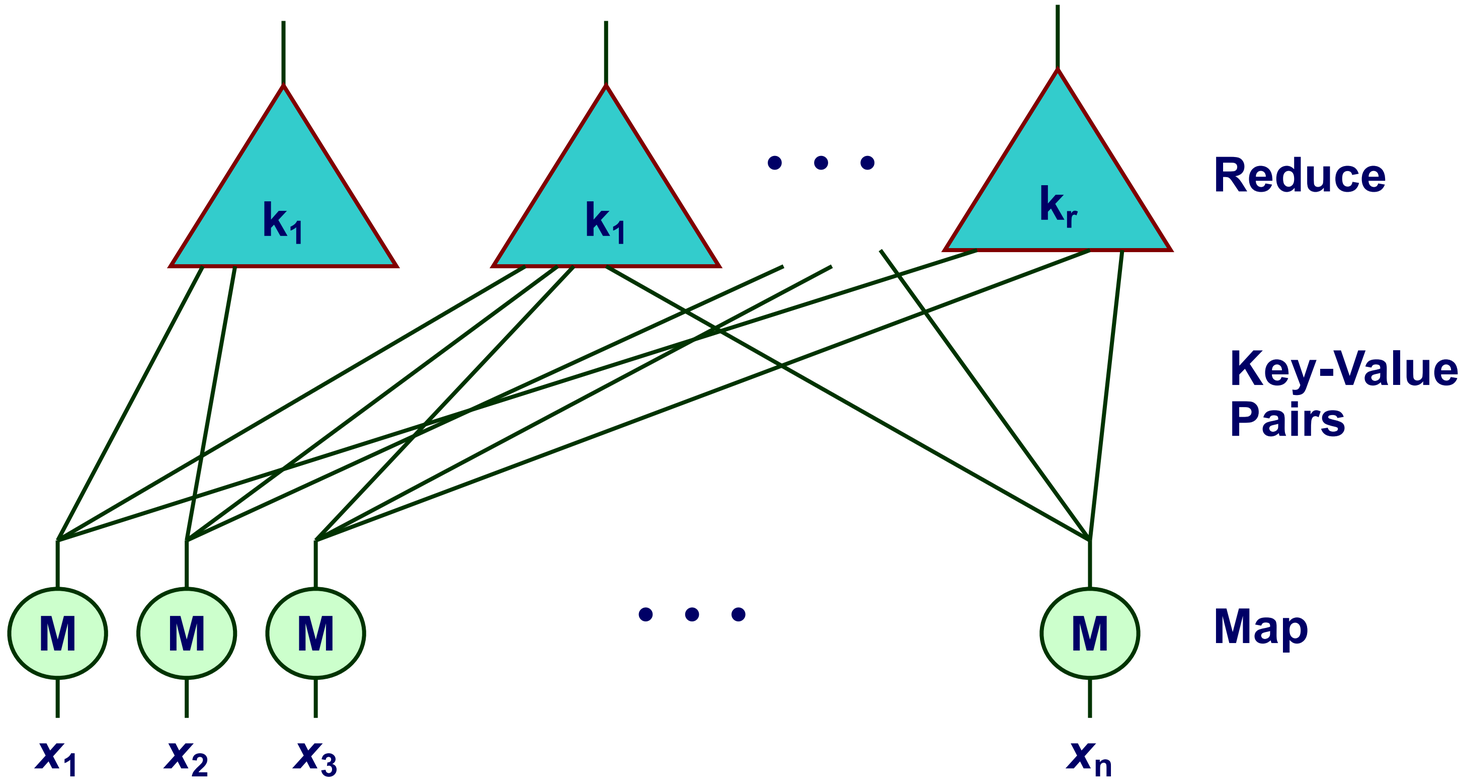
```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

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

map	(k1,v1)	→ list(k2,v2)
reduce	(k2,list(v2))	→ list(v2)

- Values can be anything, image, video, audio, …

Map/Reduce Programming Model

- Map computation across many objects
  - E.g., 10^{10} Internet web pages
- Aggregate results in many different ways
- System deals with issues of resource allocation & reliability



Dean & Ghemawat: "MapReduce: Simplified Data Processing on Large Clusters", OSDI 2004

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
- Beyond MapReduce

MapReduce Example

- Create a word index of set of documents

Come, Dick

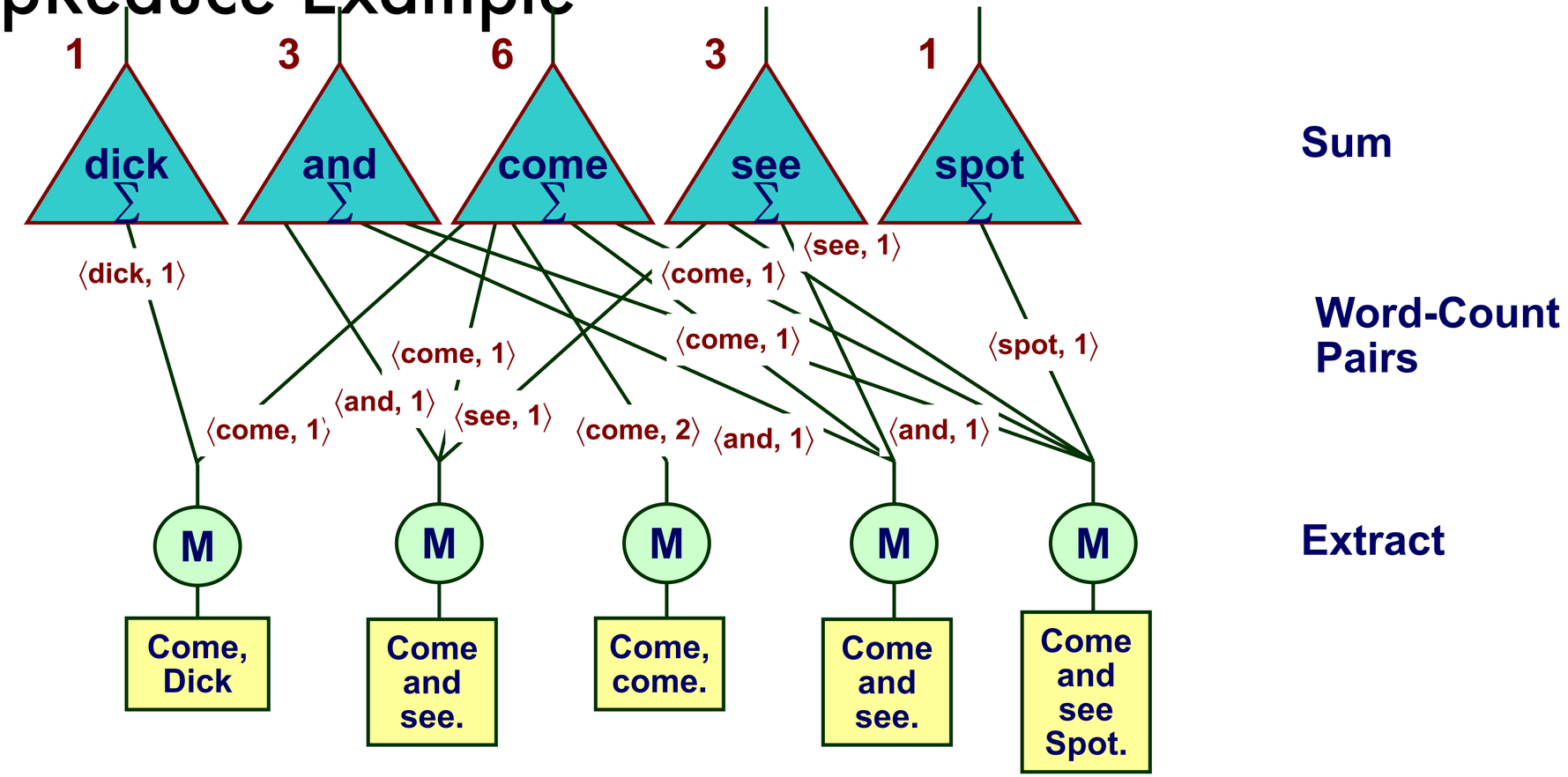
Come and see.

Come, come.

Come and see.

Come and see Spot.

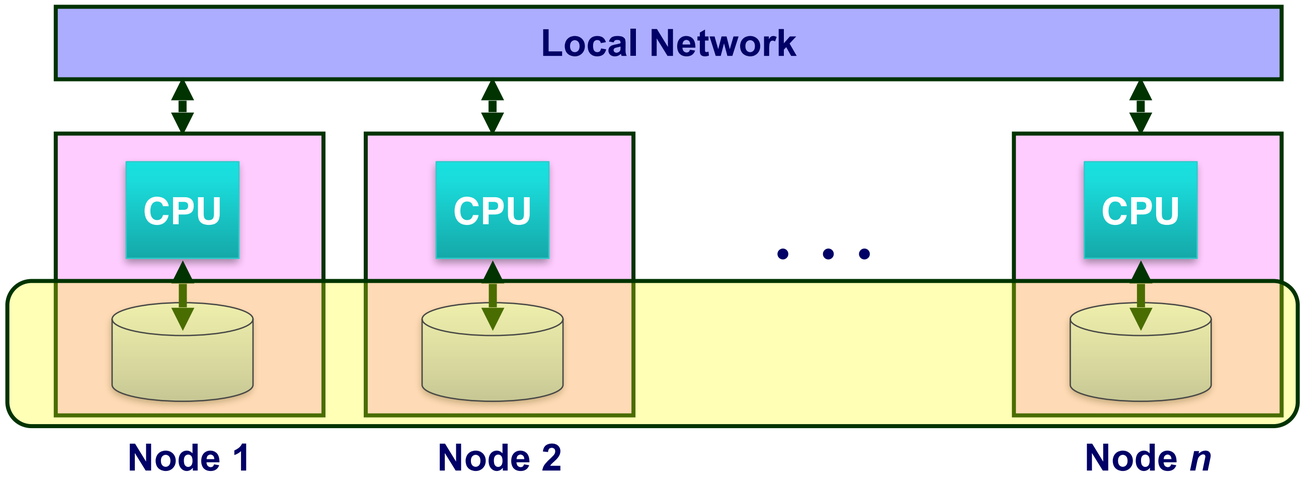
MapReduce Example



- Map: generate ⟨word, count⟩ pairs for all words in document
- Reduce: sum word counts across documents

Hadoop Project

File system with files distributed across nodes



- Store multiple (typically 3 copies of each file)
  - If one node fails, data still available
- Logically, any node has access to any file
  - May need to fetch across network (ideally, leverage locality for perf.)

Map / Reduce programming environment

- Software manages execution of tasks on nodes

Batch processing with Unix Tools

```
cat /var/log/nginx/access.log |  ❶
  awk '{print $7}' |             ❷
  sort             |             ❸
  uniq -c          |             ❹
```

```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

1. Read the log file.
2. Split each line into fields by white space, output only the 7th element (requested URL).
3. Alphabetically sort
4. Filter out repeated lines and count the number of adjacent records with the same key

Hadoop MapReduce API

Requirements

- Programmer must supply Mapper & Reducer classes

Mapper

- Steps through file one line at a time
- Code generates sequence of <key, value> pairs
  - Call output.collect(key, value)
- Default types for keys & values are strings
  - Lots of low-level machinery to convert to & from other data types
  - But can use anything “writable”

Reducer

- Given key + iterator that generates sequence of values
- Generate one or more <key, value> pairs
  - Call output.collect(key, value)

Batch processing with Unix Tools

```
cat /var/log/nginx/access.log |  ❶
  awk '{print $7}' |             ❷
  sort             |             ❸
  uniq -c          |             ❹
```

```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

1. Read the log file.
2. Split each line into fields by white space, output only the 7th element (requested URL). => Map
3. Alphabetically sort =>
4. Filter out repeated lines and count the number of adjacent records with the same key => Reduce

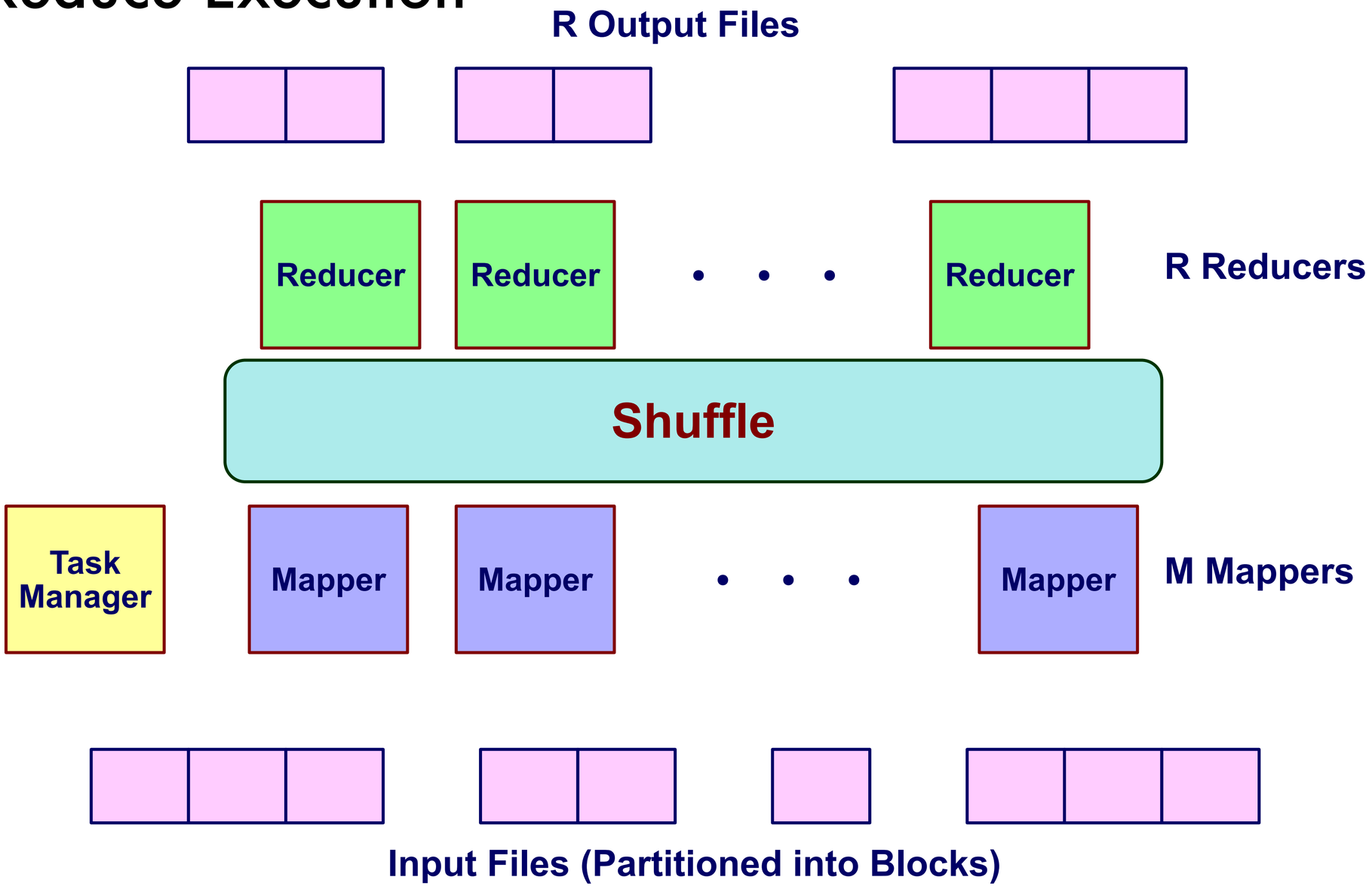
Hadoop Word Count Mapper

```
public class WordCountMapper extends MapReduceBase
        implements Mapper {

    private final static Text word = new Text();

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

    public void map(WritableComparable key, Writable values,
                    OutputCollector output, Reporter reporter)
                throws IOException {
        /* Get line from file */
        String line = values.toString();
        /* Split into tokens */
        StringTokenizer itr = new StringTokenizer(line.toLowerCase(),
                                        " \t.!?:()[],'&-;|0123456789");
        while(itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            /* Emit <token,1> as key + value
            output.collect(word, count);
        }
    }

}
```

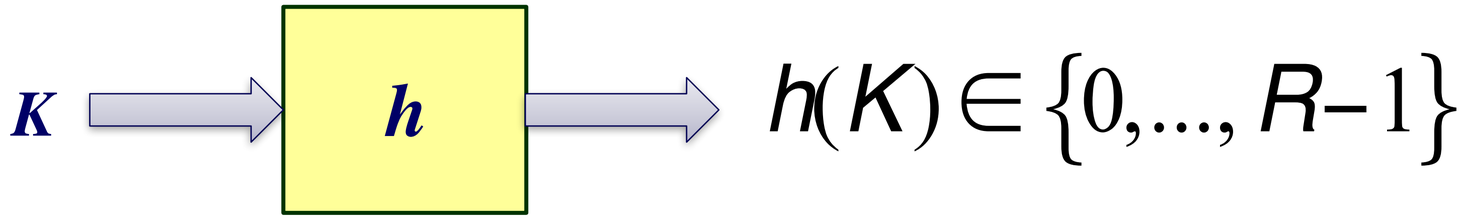
Hadoop Word Count Reducer

```
public class WordCountReducer extends MapReduceBase
        implements Reducer {

        public void reduce(WritableComparable key, Iterator values,
                           OutputCollector output, Reporter reporter)
                           throws IOException {
            int cnt = 0;
            while(values.hasNext()) {
                IntWritable ival = (IntWritable) values.next();
                cnt += ival.get();
            }
            output.collect(key, new IntWritable(cnt));
        }

}
```

MapReduce Execution



Mapping

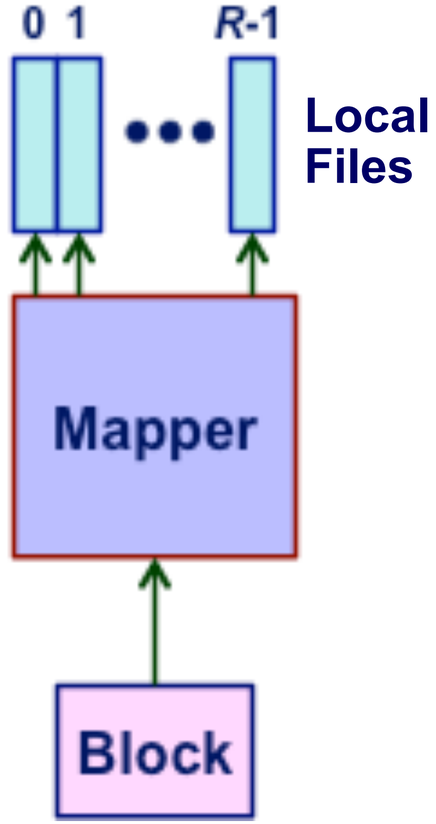


Hash Function *h*

- Maps each key *K* to integer *i* such that $0 \le i < R$

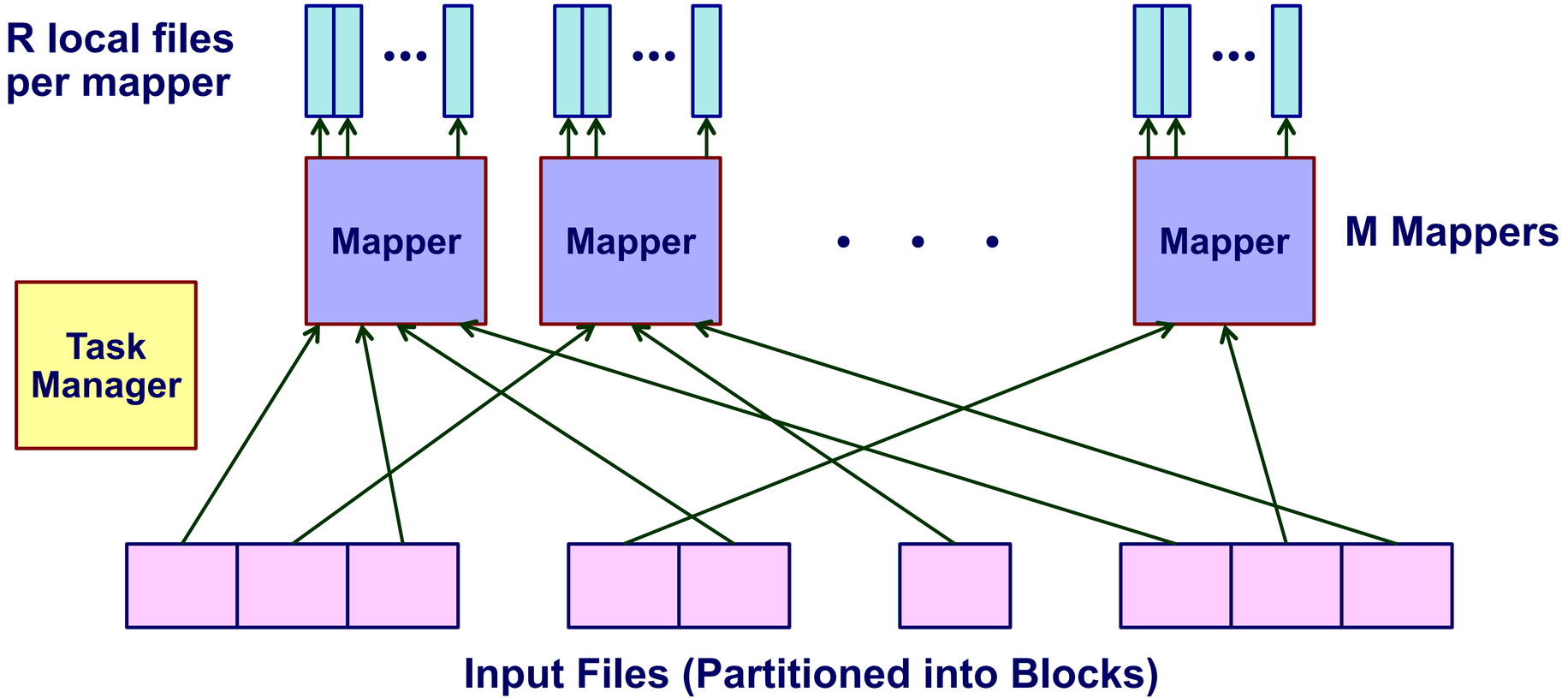
Mapper Operation

- Reads input file blocks
- Generates pairs $\langle K, V \rangle$
- Writes to local file *h*(*K*)



Mapping

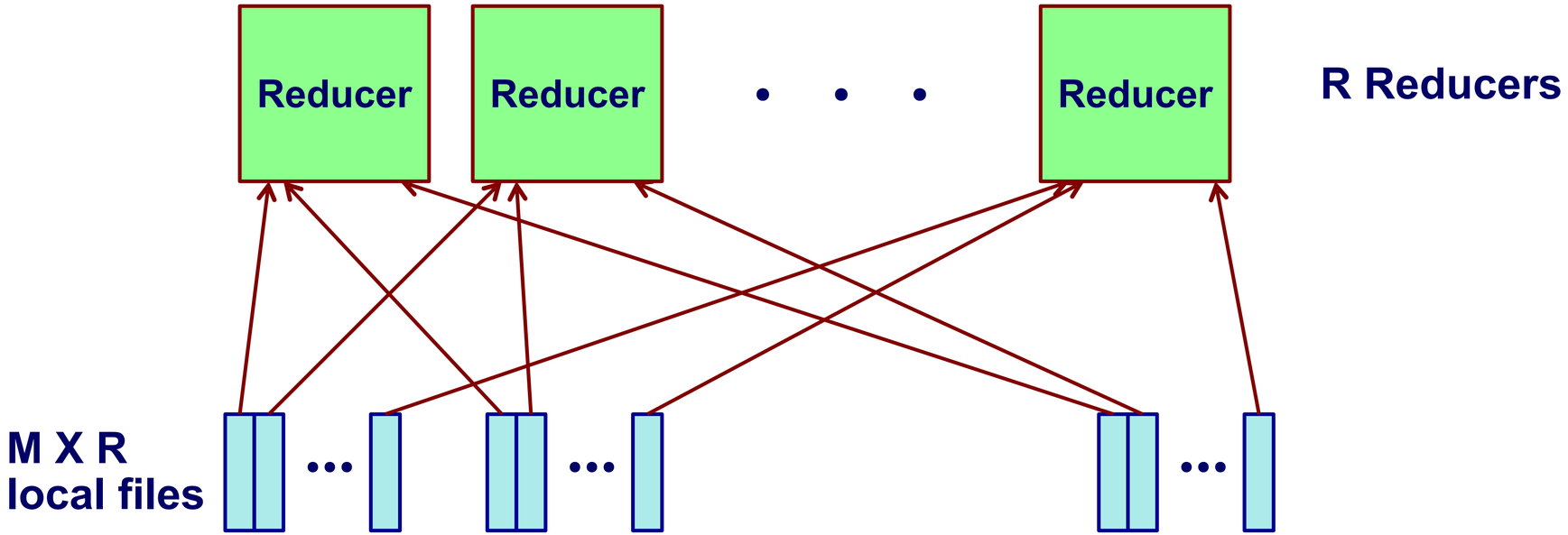
- Dynamically map input file blocks onto mappers
- Each generates key/value pairs from its blocks
- Each writes R files on local file system



Shuffling

Each Reducer:

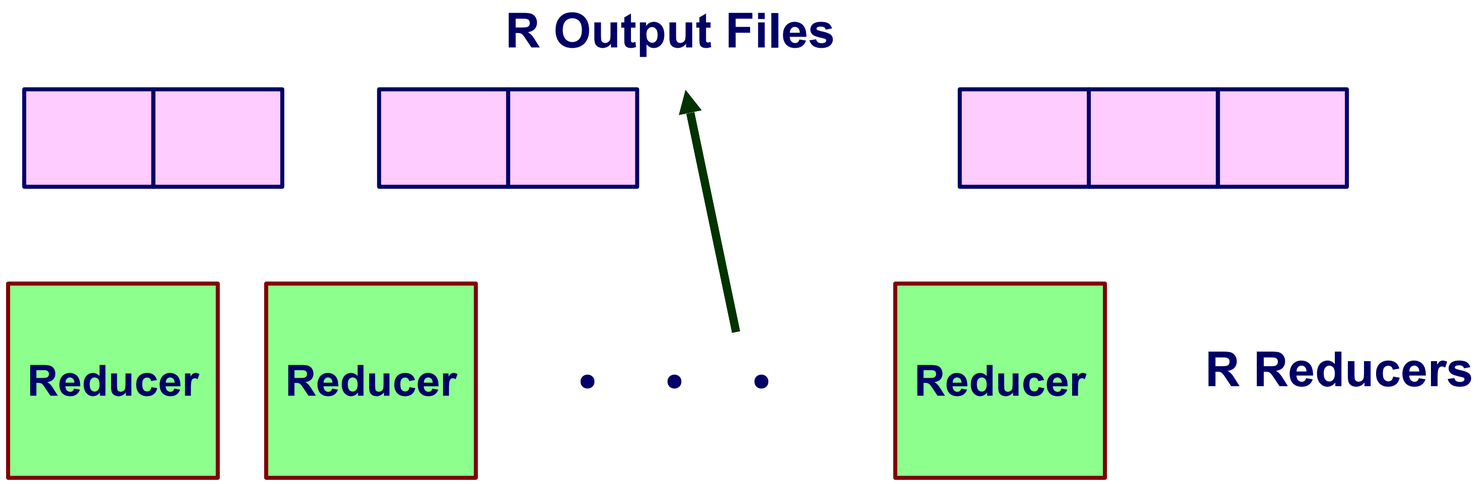
- Handles 1/R of the possible key values
- Fetches its file from each of M mappers
- Sorts all of its entries to group values by keys



Reducing

Each Reducer:

- Executes reducer function for each key
- Writes output values to parallel file system

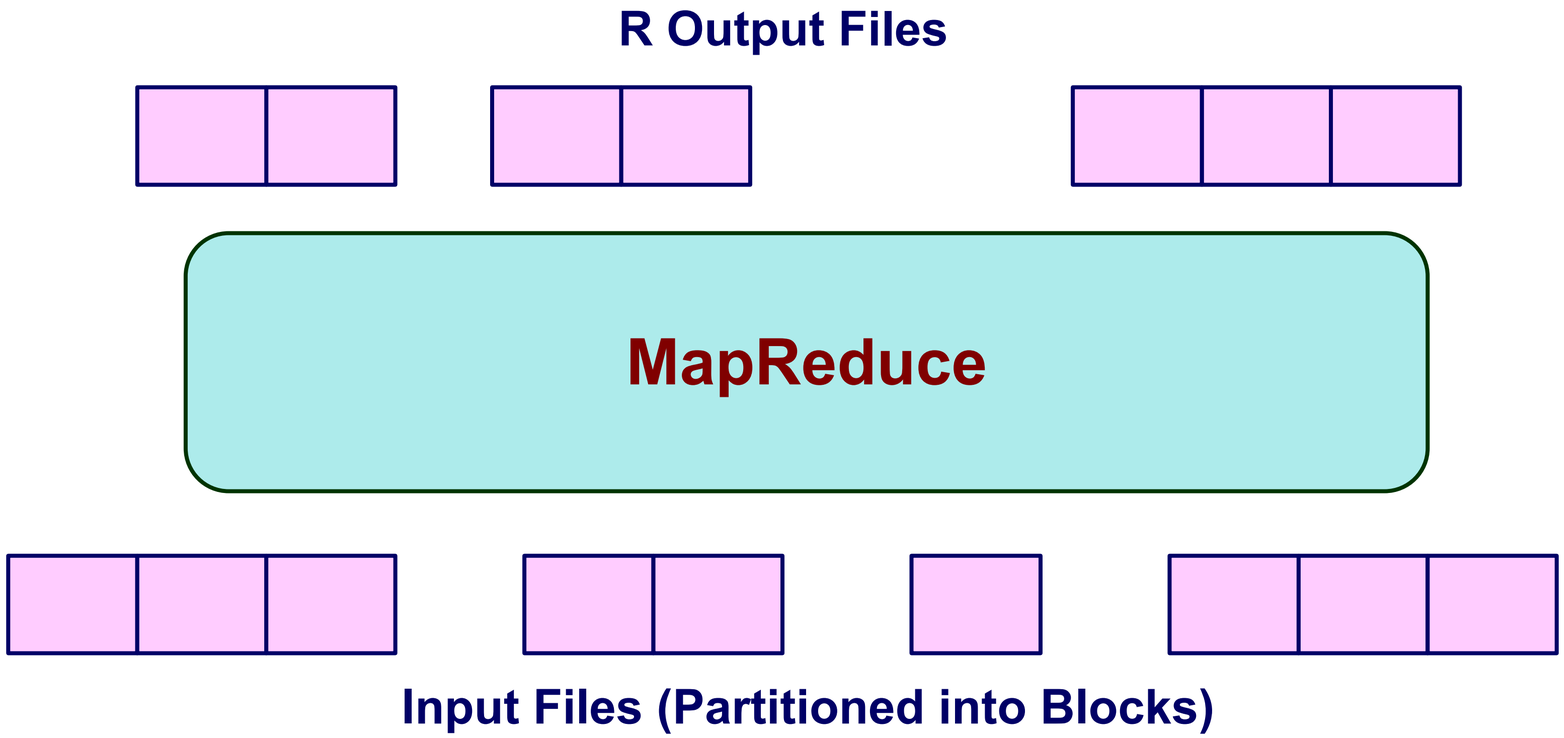


MapReduce Effect

MapReduce Step

- Reads set of files from file system
- Generates new set of files

Can iterate to do more complex processing



Batch processing with Unix Tools

```
cat /var/log/nginx/access.log |  ❶
  awk '{print $7}' |             ❷
  sort             |             ❸
  uniq -c          |             ❹
  sort -r -n       |             ❺
  head -n 5                      ❻
```

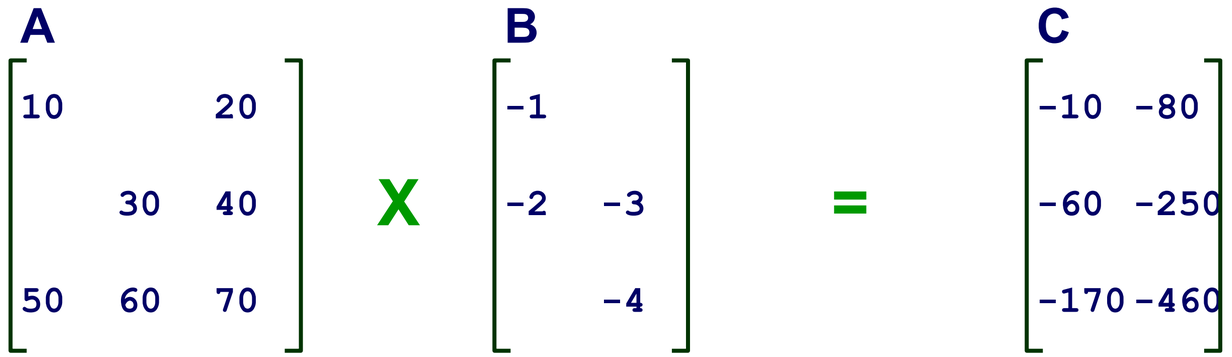
```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

1. Read the log file.
2. Split each line into fields by white space, output only the 7th element (requested URL).
3. Alphabetically sort
4. Filter out repeated lines.
5. Sort it again based on the line number (-n)
6. Out put the first five lines.

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
  - MapReduce Recap
- Beyond MapReduce

Example: Sparse Matrices with Map/Reduce

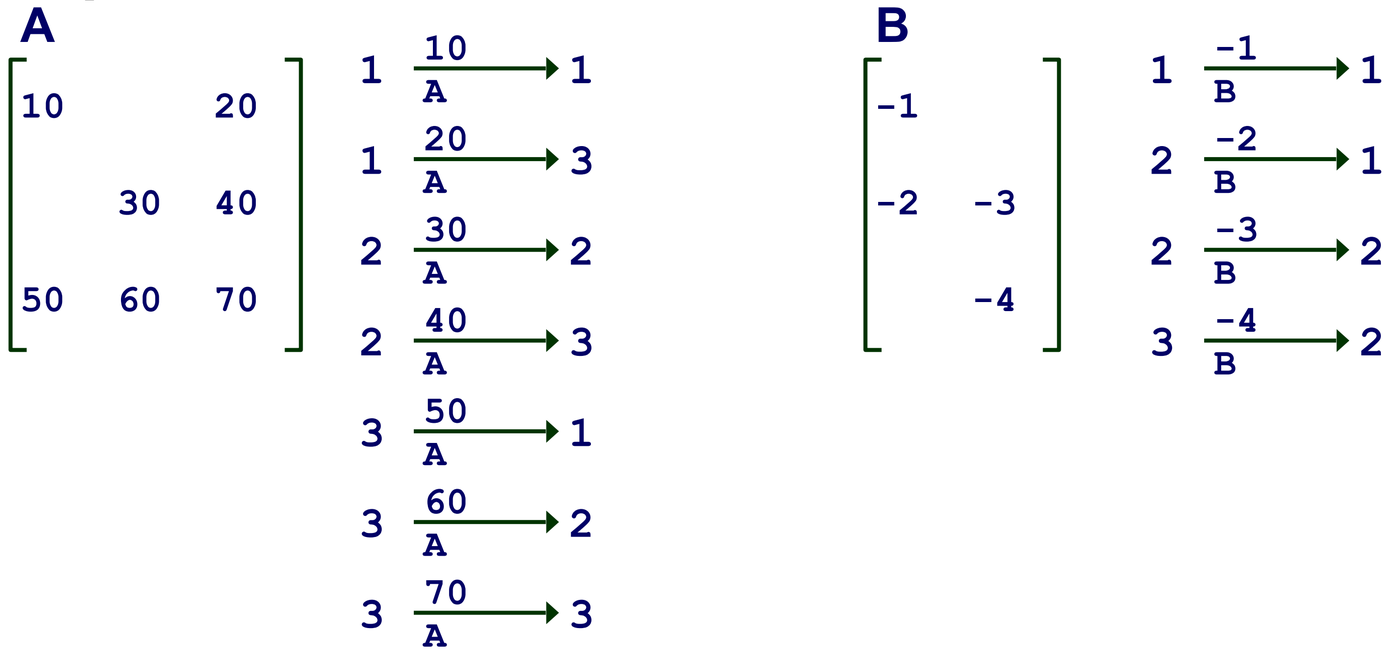


- Task: Compute product C = A·B
- Assume most matrix entries are 0

Motivation

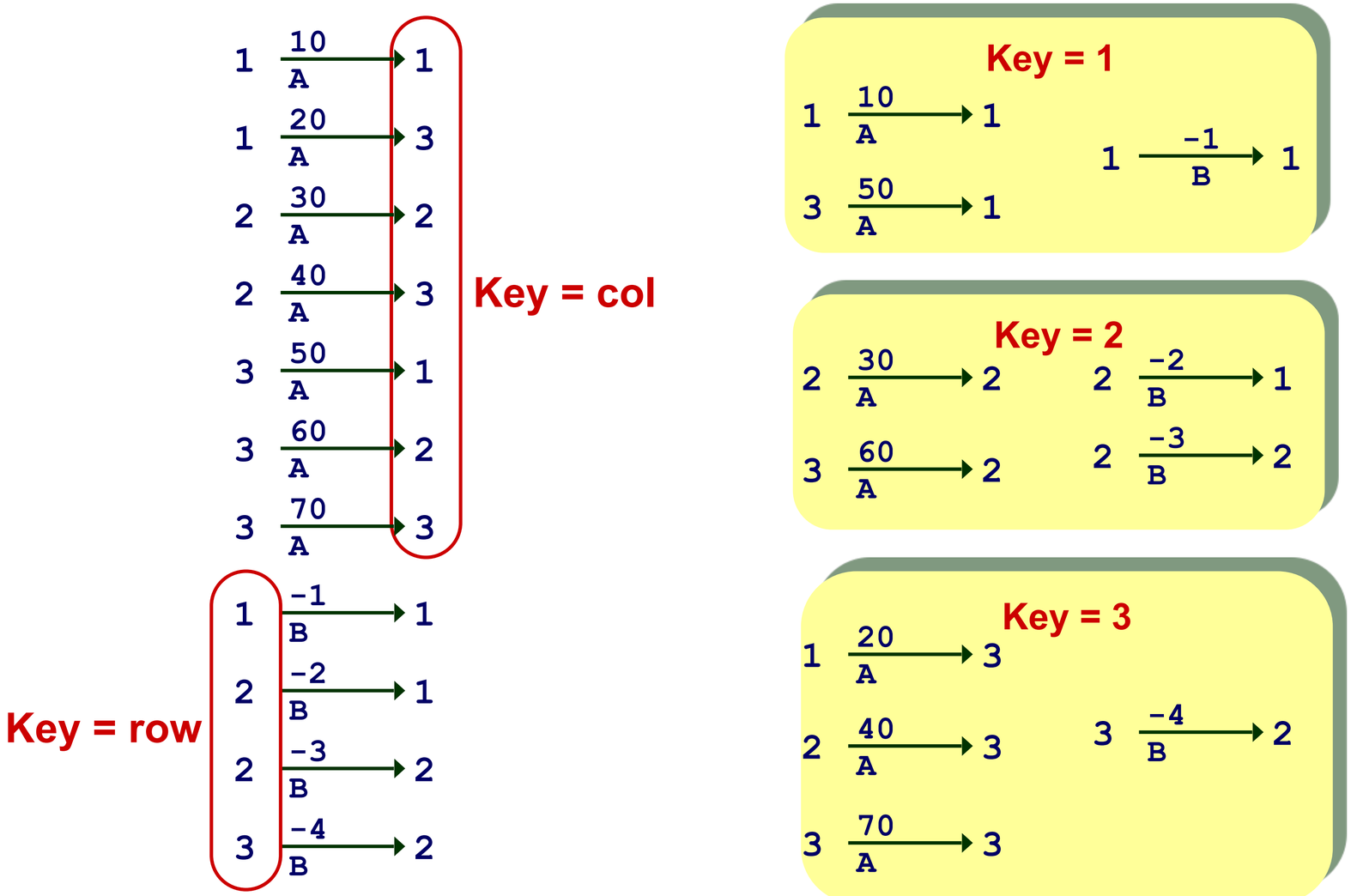
- Core problem in scientific computing
- Challenging for parallel execution
- Demonstrate expressiveness of Map/Reduce

Computing Sparse Matrix Product



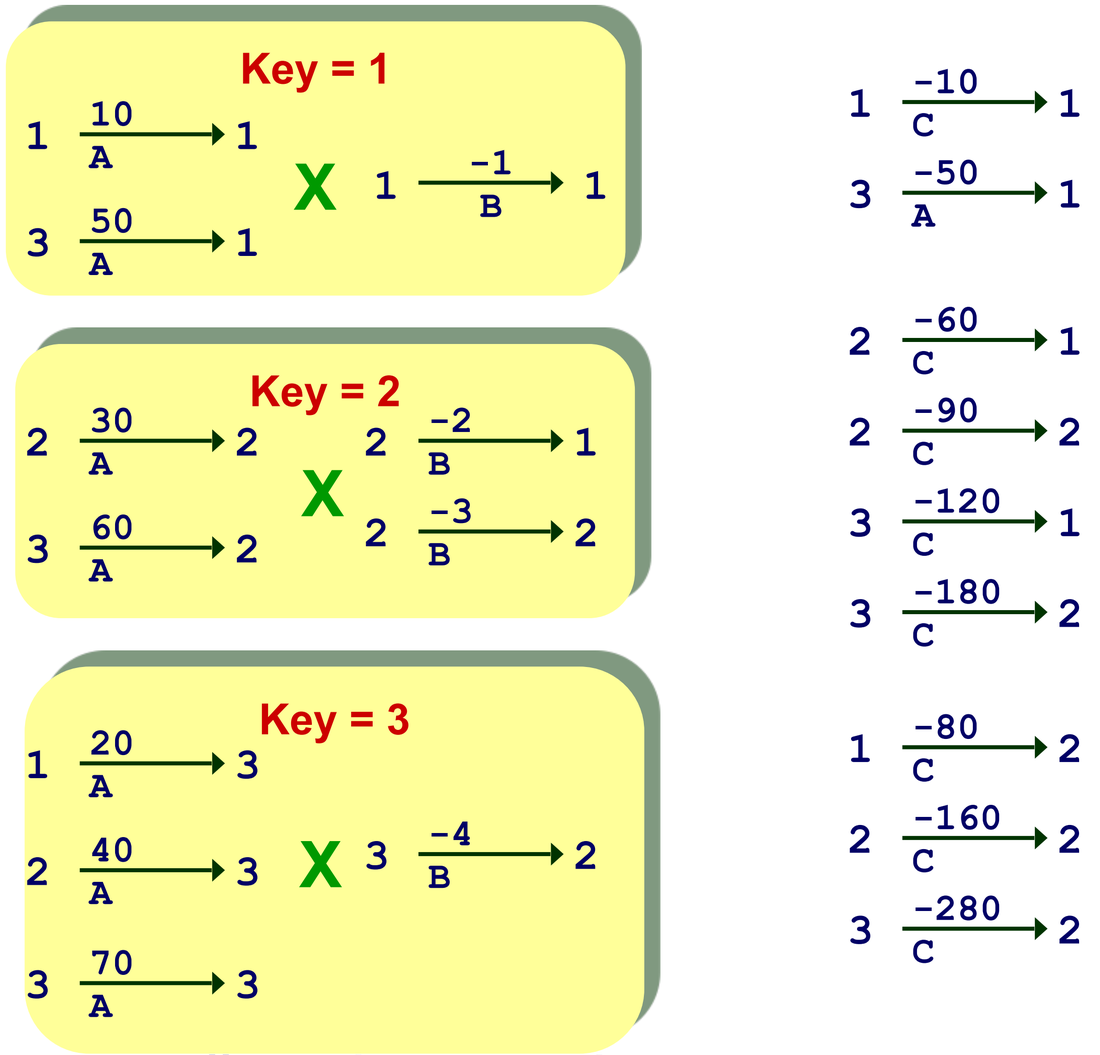
- Represent matrix as list of nonzero entries
  ⟨row, col, value, matrixID⟩
- Strategy
  - Phase 1: Compute all products $a_{i,k} \cdot b_{k,j}$
  - Phase 2: Sum products for each entry i,j
  - Each phase involves a Map/Reduce

Phase 1 Map of Matrix Multiply



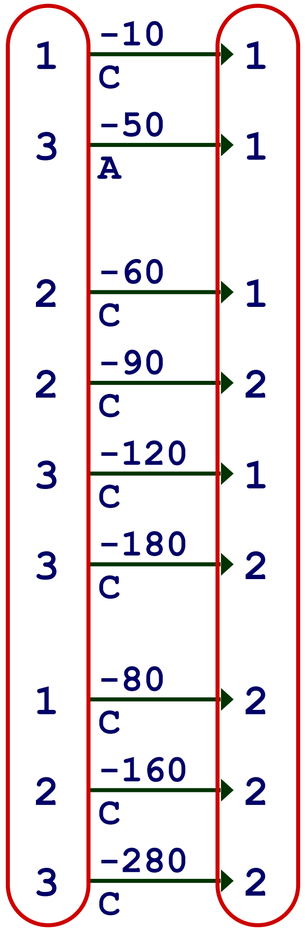
- Group values $a_{i,k}$ and $b_{k,j}$ according to key k

Phase 1 "Reduce" of Matrix Multiply

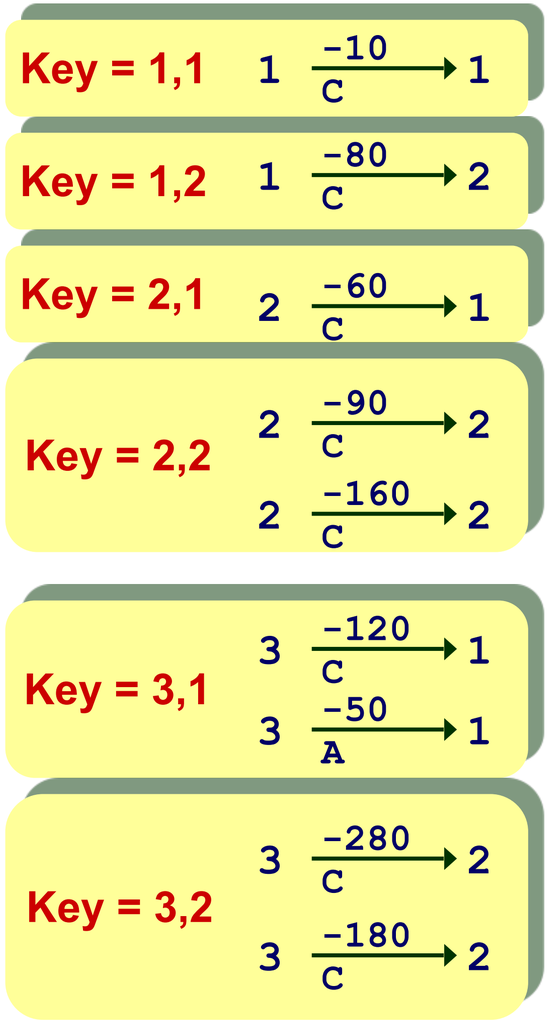


- Generate all products $a_{i,k} \cdot b_{k,j}$

Phase 2 Map of Matrix Multiply

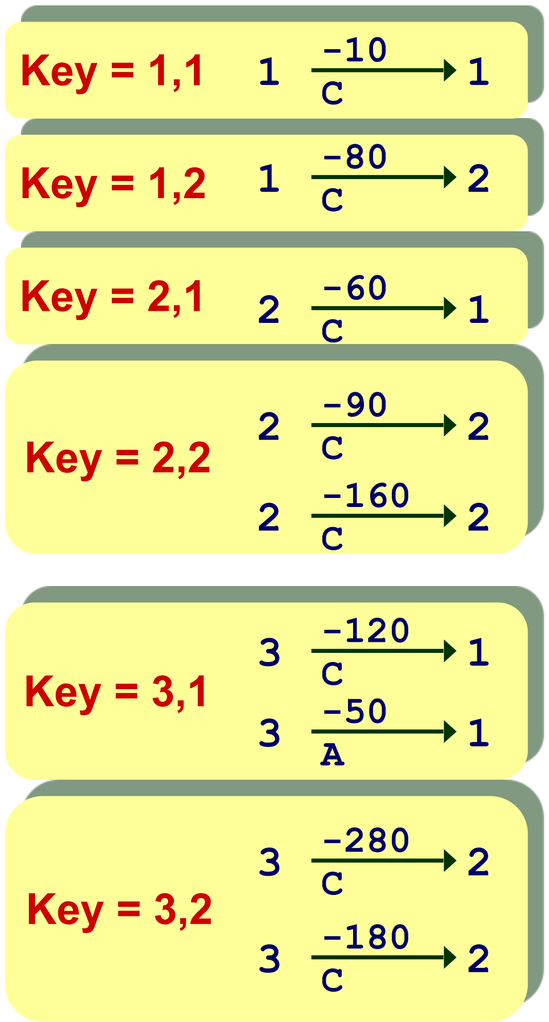


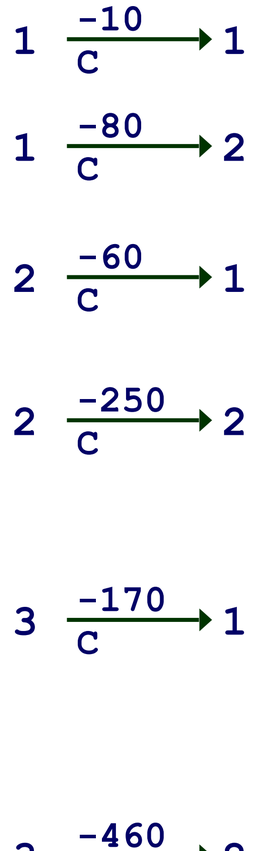
Key = row,col



- Group products $a_{i,k} \cdot b_{k,j}$ with matching values of i and j

Phase 2 Reduce of Matrix Multiply





C

-10	-80
-60	-250
-170	-460

- Sum products to get final entries

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
  - **MapReduce Recap**
- Beyond MapReduce

MapReduce Implementation

Built on Top of Parallel File System

- Google: GFS, Hadoop: HDFS
- Provides global naming
- Reliability via replication (typically 3 copies)

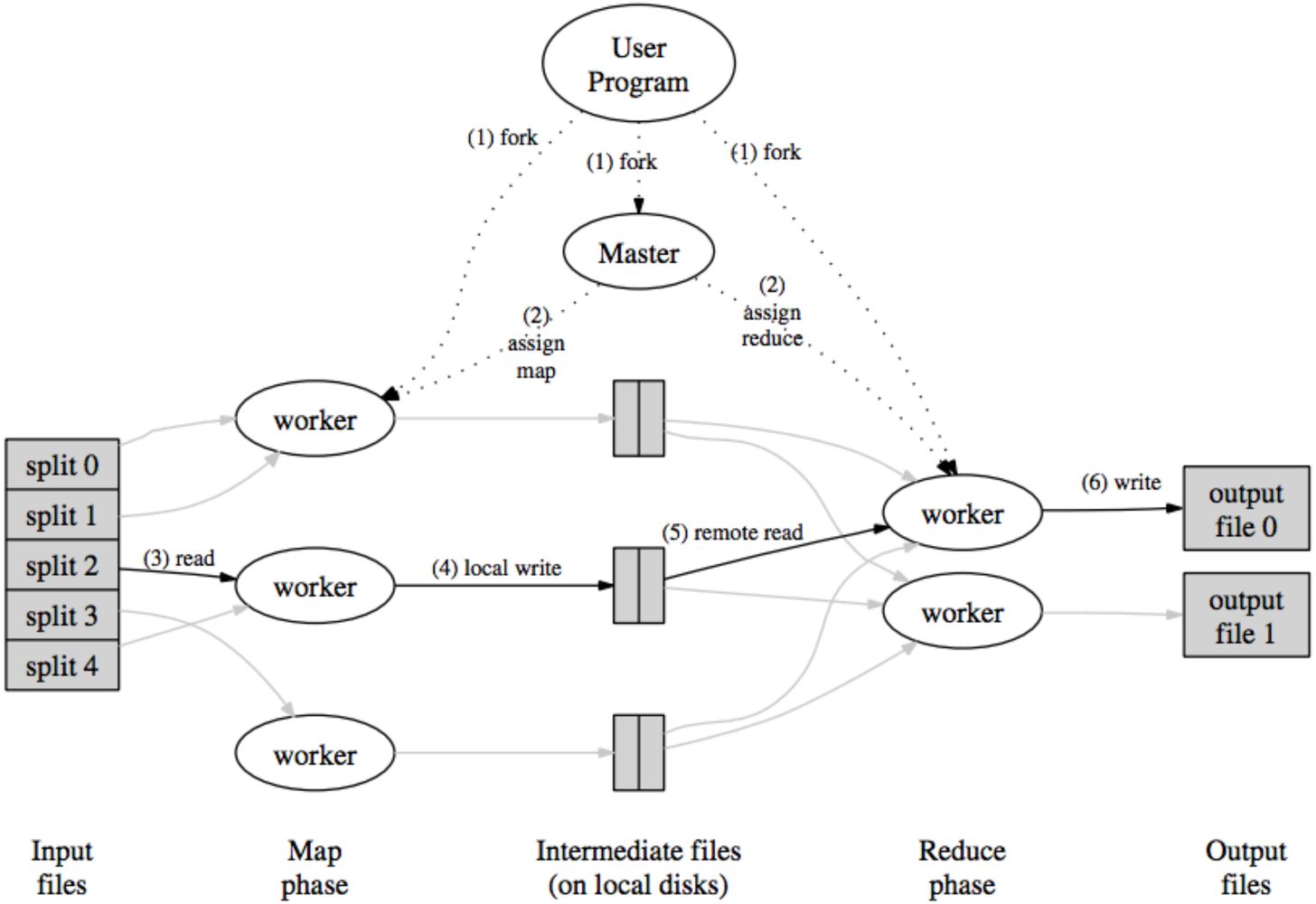
Breaks work into tasks

- Master schedules tasks on workers dynamically
- Typically #tasks >> #processors

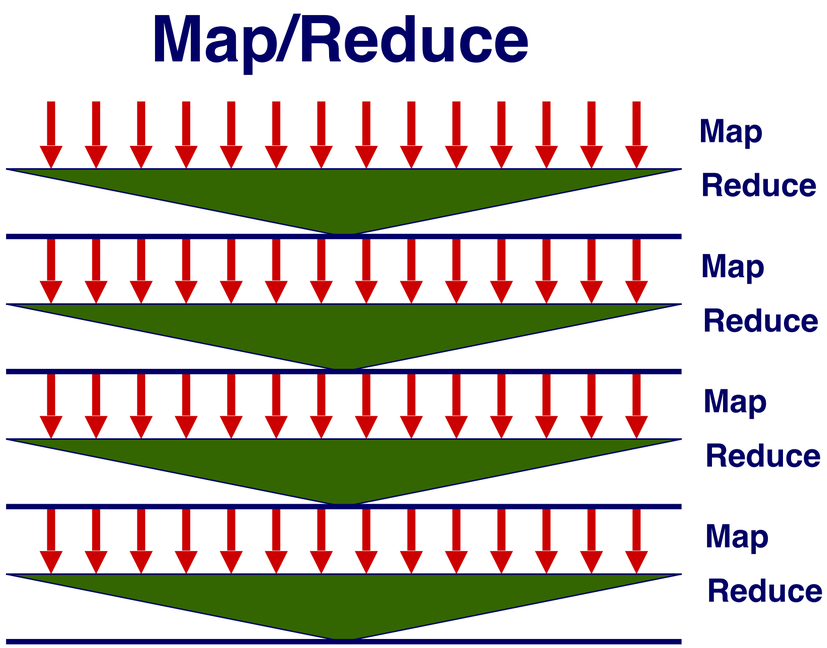
Net Effect

- Input: Set of files in reliable file system
- Output: Set of files in reliable file system

MapReduce Execution (authors view)



Map/Reduce Operation



Characteristics

- Computation broken into many, short-lived tasks
  - Mapping, reducing
- Use disk storage to hold intermediate results

Strengths

- Great flexibility in placement, scheduling, and load balancing
- Can access large data sets

Weaknesses

- Higher overhead
- Lower raw performance

Example Parameters

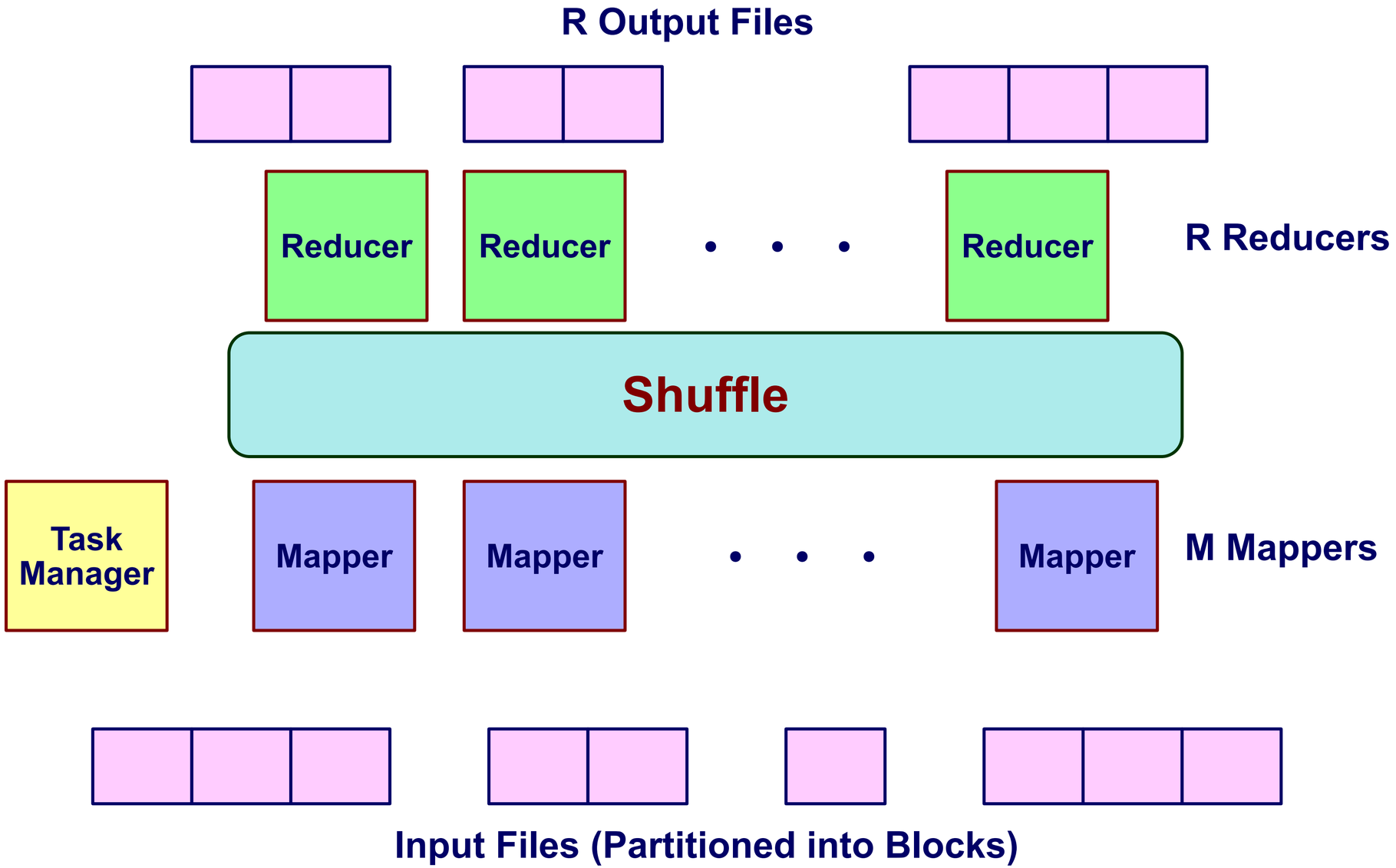
Sort Benchmark

- 10^{10} 100-byte records
- Partition into M = 15,000 64MB pieces
  - Key = value
  - Partition according to most significant bytes
- Sort locally with R = 4,000 reducers

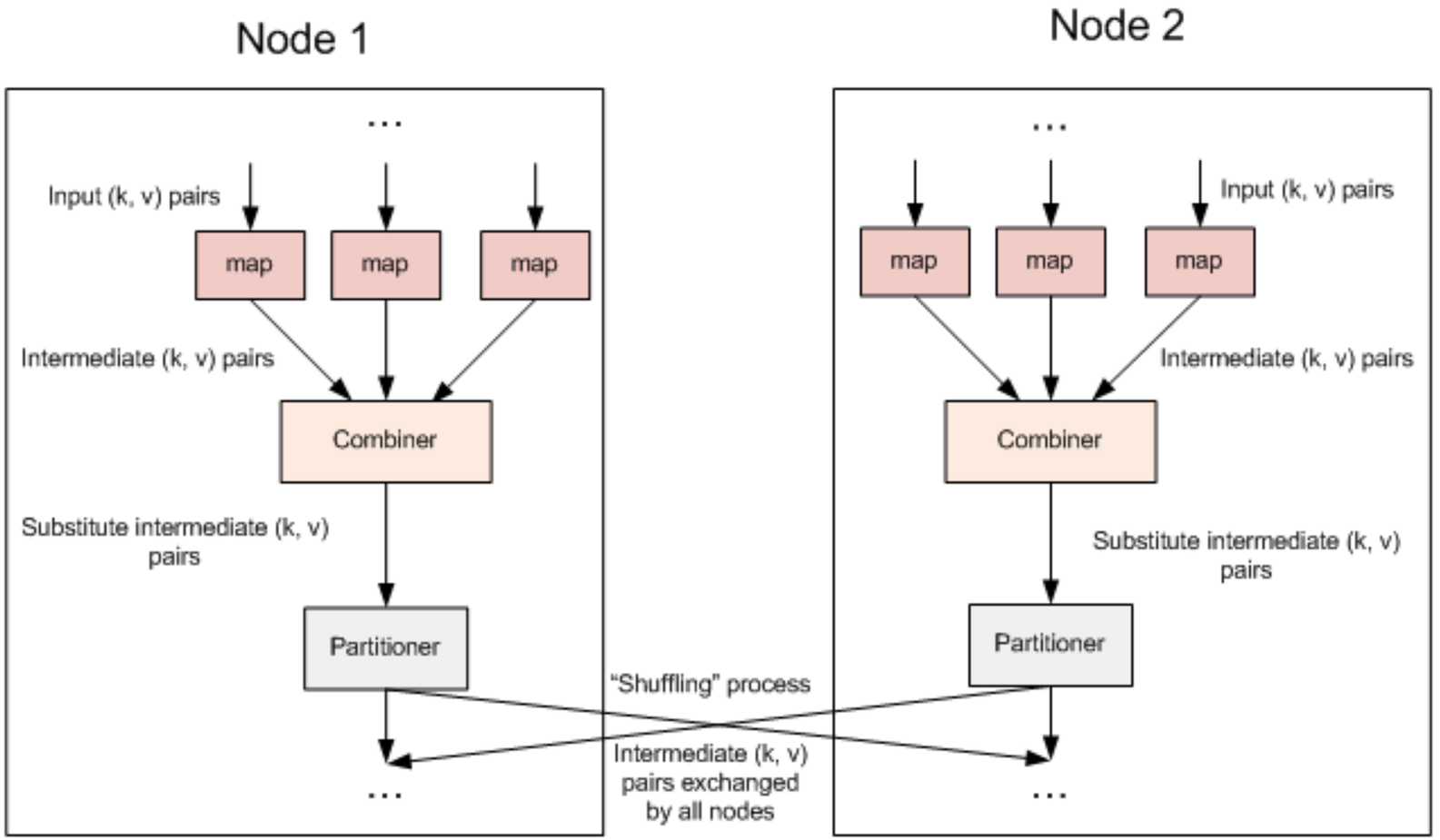
Machine

- 1800 2Ghz Xeons
- Each with 2 160GB IDE disks
- Gigabit ethernet
- 891 seconds total

Diving into MapReduce Execution

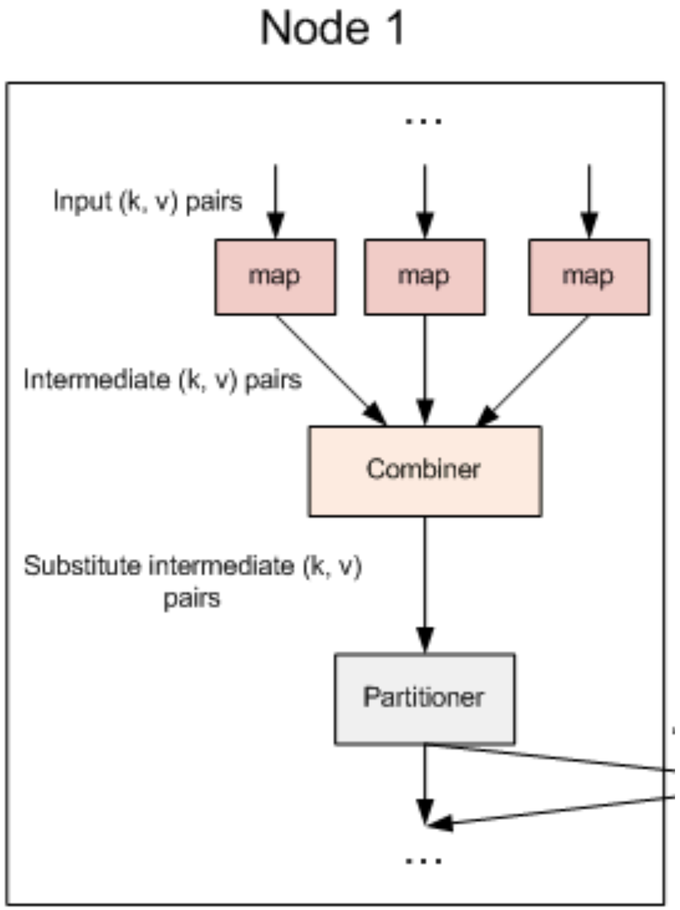


Input → Map → Combiner → Partitioner → Reducer → Output



Combiners & Partitioners are optional.

Input → Map → Combiner → Partitioner → Reducer → Output



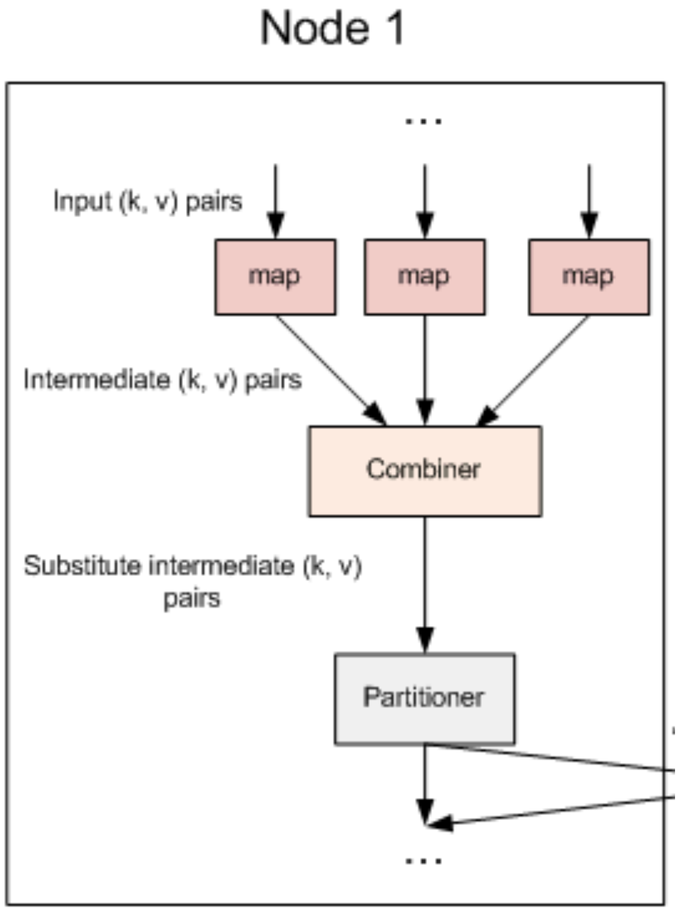
Input:

What do you mean by Object
What do you know about Java
What is Java Virtual Machine
How Java enabled High Performance

Record reader:

<1, What do you mean by Object>
<2, What do you know about Java>
<3, What is Java Virtual Machine>
<4, How Java enabled High Performance>

Combiner (mini-reducer):
optional, to summarize the map output records with the same key



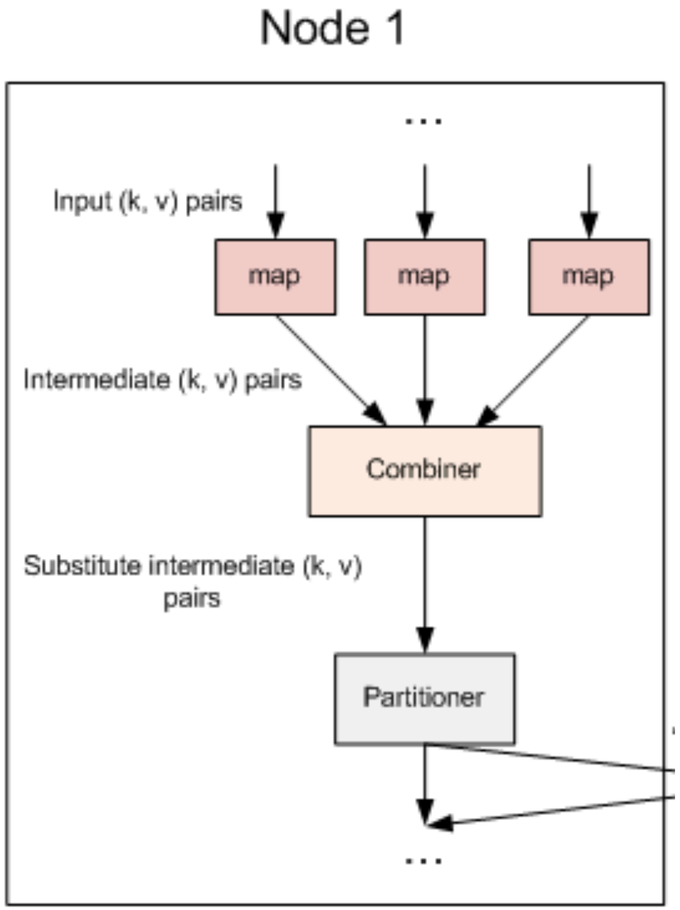
Map output:

<What,1> <do,1> <you,1> <mean,1> <by,1>
<Object,1> <What,1> <do,1> <you,1> <know,1>
<about,1> <Java,1> <What,1> <is,1> <Java,1>
<Virtual,1> <Machine,1> <How,1> <Java,1>
<enabled,1> <High,1> <Performance,1>

Combiner output:

<What,1,1,1> <do,1,1> <you,1,1> <mean,1>
<by,1> <Object,1> <know,1> <about,1>
<Java,1,1,1> <is,1> <Virtual,1> <Machine,1>
<How,1> <enabled,1> <High,1> <Performance,1>

Partitioner:
optional, a condition in processing an input dataset



Combiner output:

<What,1,1,1> <do,1,1> <you,1,1> <mean,1>
<by,1> <Object,1> <know,1> <about,1>
<Java,1,1,1> <is,1> <Virtual,1> <Machine,1>
<How,1> <enabled,1> <High,1> <Performance,1>

The number of partitioners is equal to the number of reducers.

Partitioner output:

<What,1,1,1>: long sentence,
<do,1,1>: long sentence,
……

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
  - MapReduce Recap
- Beyond MapReduce

Interesting Features

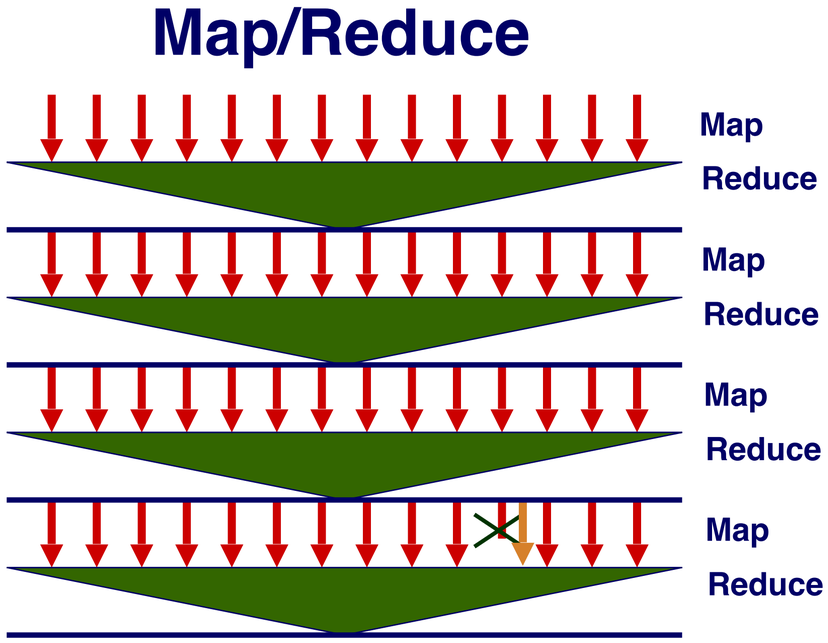
Fault Tolerance

- Assume reliable file system
- Detect failed worker
  - Heartbeat mechanism
- Reschedule failed task

Stragglers

- Tasks that take long time to execute
- Might be bug, flaky hardware, or poor partitioning
- When done with most tasks, reschedule any remaining executing tasks
  - Keep track of redundant executions
  - Significantly reduces overall run time

Map/Reduce Fault Tolerance



Data Integrity

- Store multiple copies of each file
- Including intermediate results of each Map / Reduce
  - Continuous checkpointing

Recovering from Failure

- Simply recompute lost result
  - Localized effect
- Dynamic scheduler keeps all processors busy

Map/Reduce Summary

Typical Map/Reduce Applications

- Sequence of steps, each requiring map & reduce
- Series of data transformations
- Iterating until reach convergence

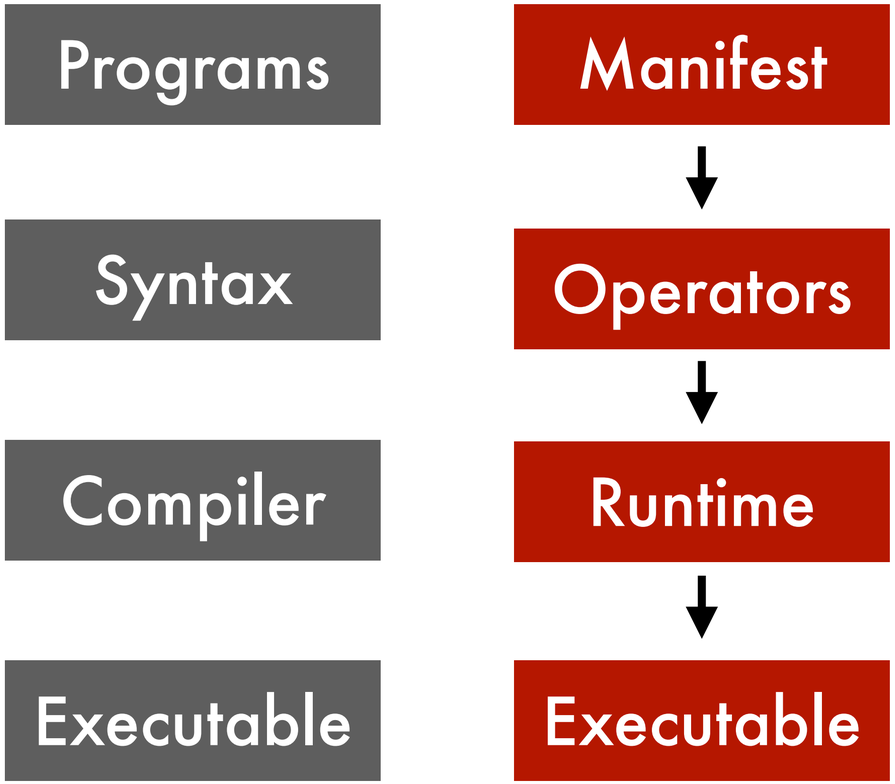
Strengths of Map/Reduce

- User writes simple functions, system manages complexities of mapping, synchronization, fault tolerance
- Very general
- Good for large-scale data analysis

Limitations

- No locality of data or activity
- Each map/reduce step must complete before next begins

MPF - Modular Privacy Flows



A fixed set of operators

A trusted runtime with a small set of pre-loaded implementations

Comparing Hadoop to Distributed Databases

- Unix + Shell
- Hadoop + MapReduce
  - Distributed Unix + Shell
  - More structured processing.
  - Constrained data models; less utility functions.
- Massively parallel processing Distributed version of Unix;
  - Database indexes, column storages?

Data models

- Hadoop: byte sequences
- MapReduce: key-value pairs
- Databases data models: relational, documents.
- Evolutions
  - A purist's point of view:
    - Database approaches are better. Careful reflections
  - Reality:
    - Dump all the data. Defer the schema design.
    - Schema-on-read + Data warehouse + ETL