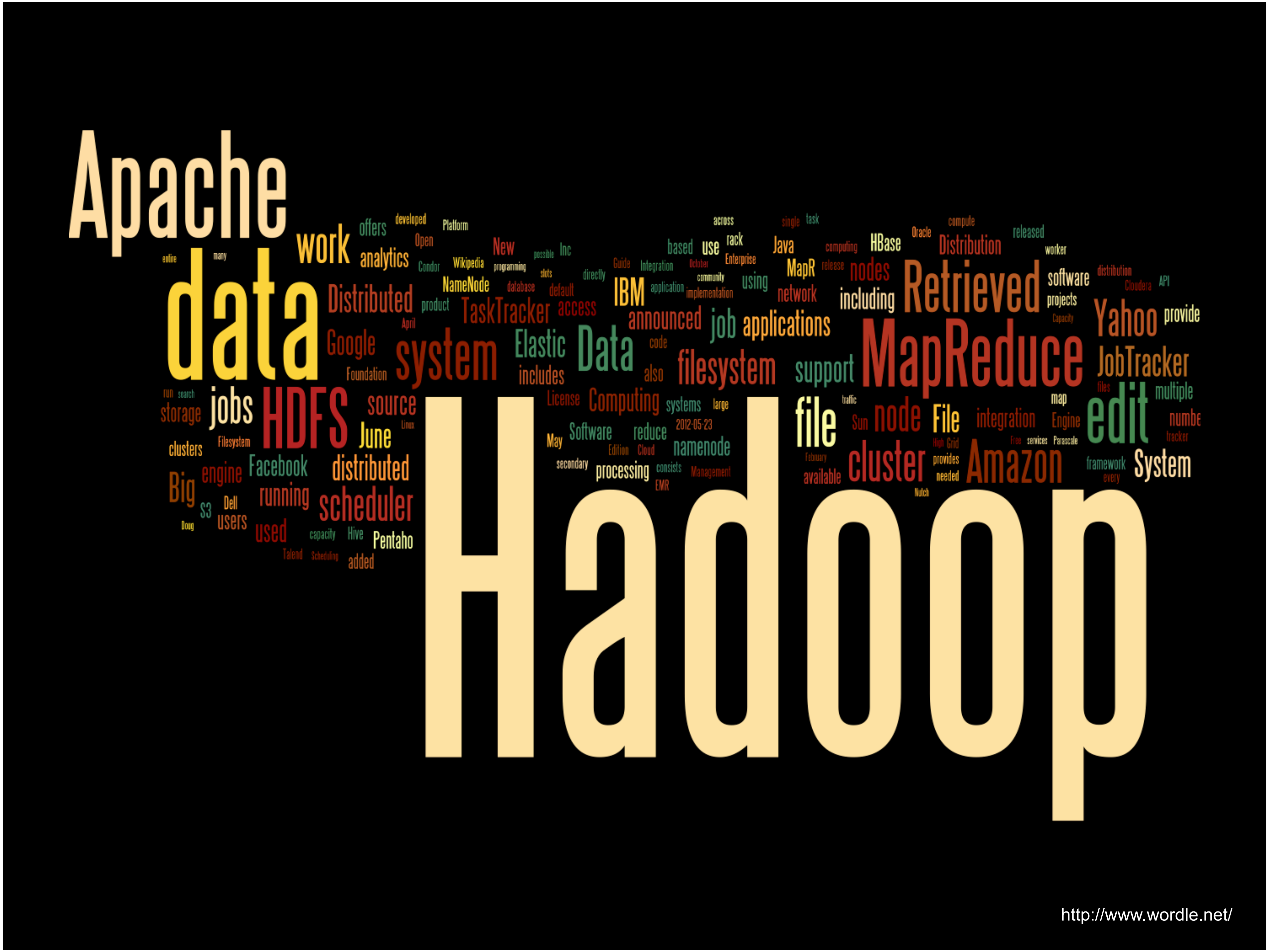
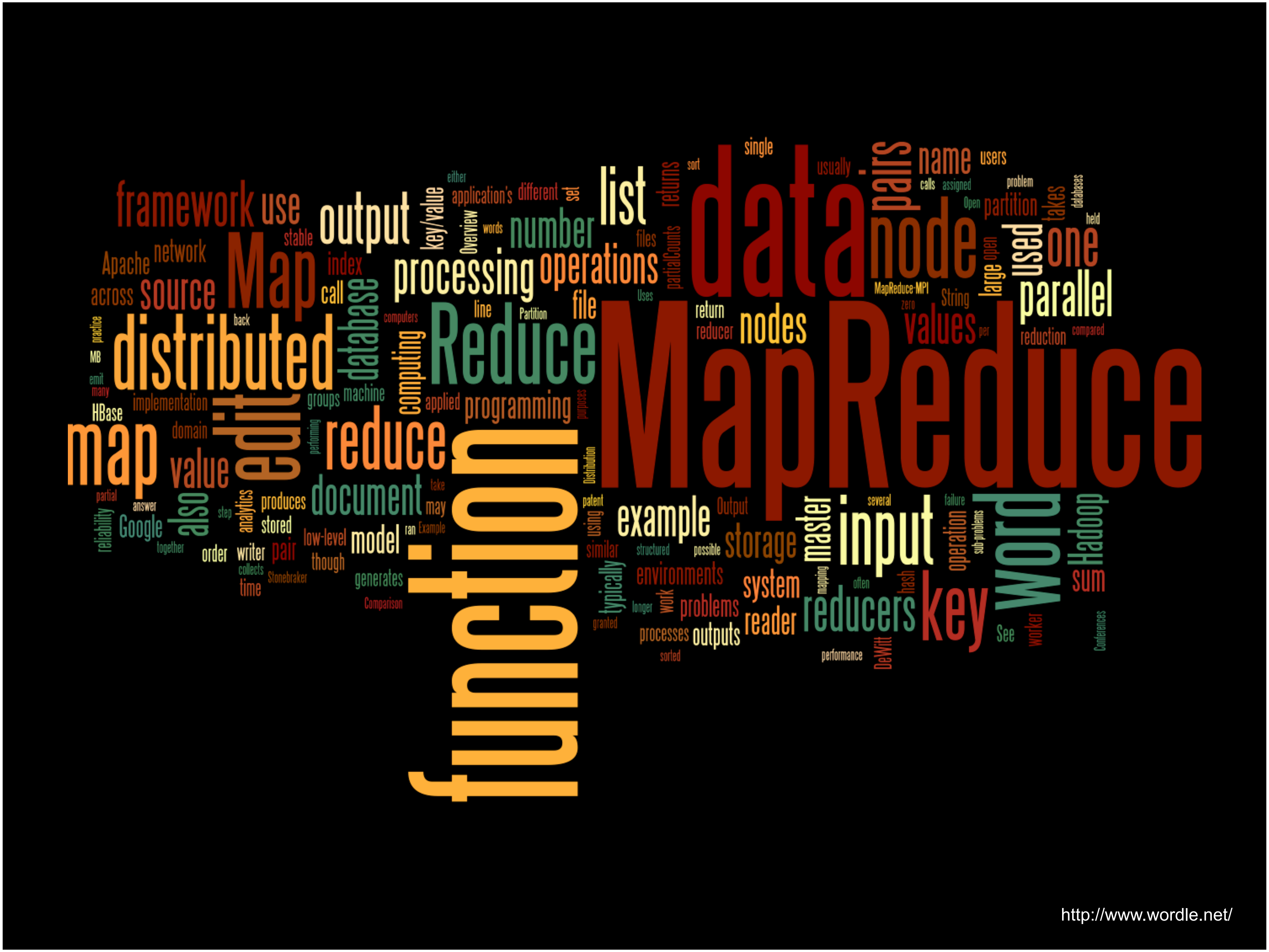
Hadoop & MapReduce


Hadoop
Apache
data
MapReduce
Retrieved
HDFS
system
Data
Elastic
filesystem
Distributed
TaskTracker
JobTracker
Yahoo
edit
Amazon
cluster
node
file
jobs
work
Google
scheduler
distributed
Facebook
running
used
users
Big
engine
storage
clusters
June
source
Pentaho
Hive
added
capacity
announced
IBM
job
applications
support
including
nodes
network
MapR
Java
HBase
Distribution
software
projects
provide
multiple
System
framework
integration
File
Engine
Computing
License
includes
Software
reduce
namenode
processing
systems
May
secondary
analytics
offers
developed
Platform
New
Open
NameNode
Wikipedia
product
access
default
Foundation
Inc
based
use
rack
across
Enterprise
using
Guide
Integration
compute
released
worker
distribution
API
also
code
large
Sun
available
provides
needed
Cloud
Edition
EMR
consists
Management
s3
Dell
Doug
Talend
Scheduling
Filesystem
Linux
search
run
April
Condor
programming
database
slots
directly
application
community
implementation
October
release
computing
single
task
Oracle
Cloudera
Capacity
files
map
number
tracker
every
services
Parascale
Free
Grid
High
February
Nutch
traffic
2012-05-23
entire
many
possible
http://www.wordle.net/


MapReduce
function
data
Reduce
distributed
Map
map
reduce
node
edit
word
key
input
output
processing
operations
list
number
framework
use
parallel
nodes
values
document
example
reducers
storage
master
Hadoop
programming
database
computing
file
source
value
also
Google
model
name
pairs
one
used
partition
network
Apache
across
index
stable
call
key/value
Overview
application's
different
set
returns
files
partialCounts
single
usually
users
problem
calls
assigned
databases
held
Open
takes
open
large
MapReduce-MPI
String
zero
per
reduction
compared
return
reducer
line
Partition
Uses
words
either
sort
back
computers
practice
MB
emit
many
implementation
groups
machine
applied
purposes
HBase
domain
performing
take
may
partial
answer
analytics
produces
stored
step
reliability
together
ran
Example
order
writer
pair
low-level
though
collects
generates
time
Stonebraker
Comparison
Distribution
patent
using
similar
typically
structured
environments
possible
Output
several
failure
operation
sub-problems
mapping
often
hash
sum
system
problems
reader
outputs
processes
longer
work
granted
sorted
performance
DeWitt
See
worker
Conferences
http://www.wordle.net/

Hadoop

is an open-source software framework (or platform) for…

Reliable — Storage/Computational unit Failures completely transparent to applications.

\+

Scalable — Large clusters of commodity hardware

\+

Distributed — Input data set will not fit on a single computer's hard drive

computing! — Built to process "web-scale" data on the order of petabytes.

Hadoop

Again, software for

Data intensive… … distributed processing.

Although not necessarily, typically think of two main components:

Hadoop DFS

- A distributed file system.
- Focuses on high-throughput access to application data.

Hadoop MapReduce

- System for parallel processing of large data sets.
- Executes programs adhering to a specific programming model: MapReduce

HDFS

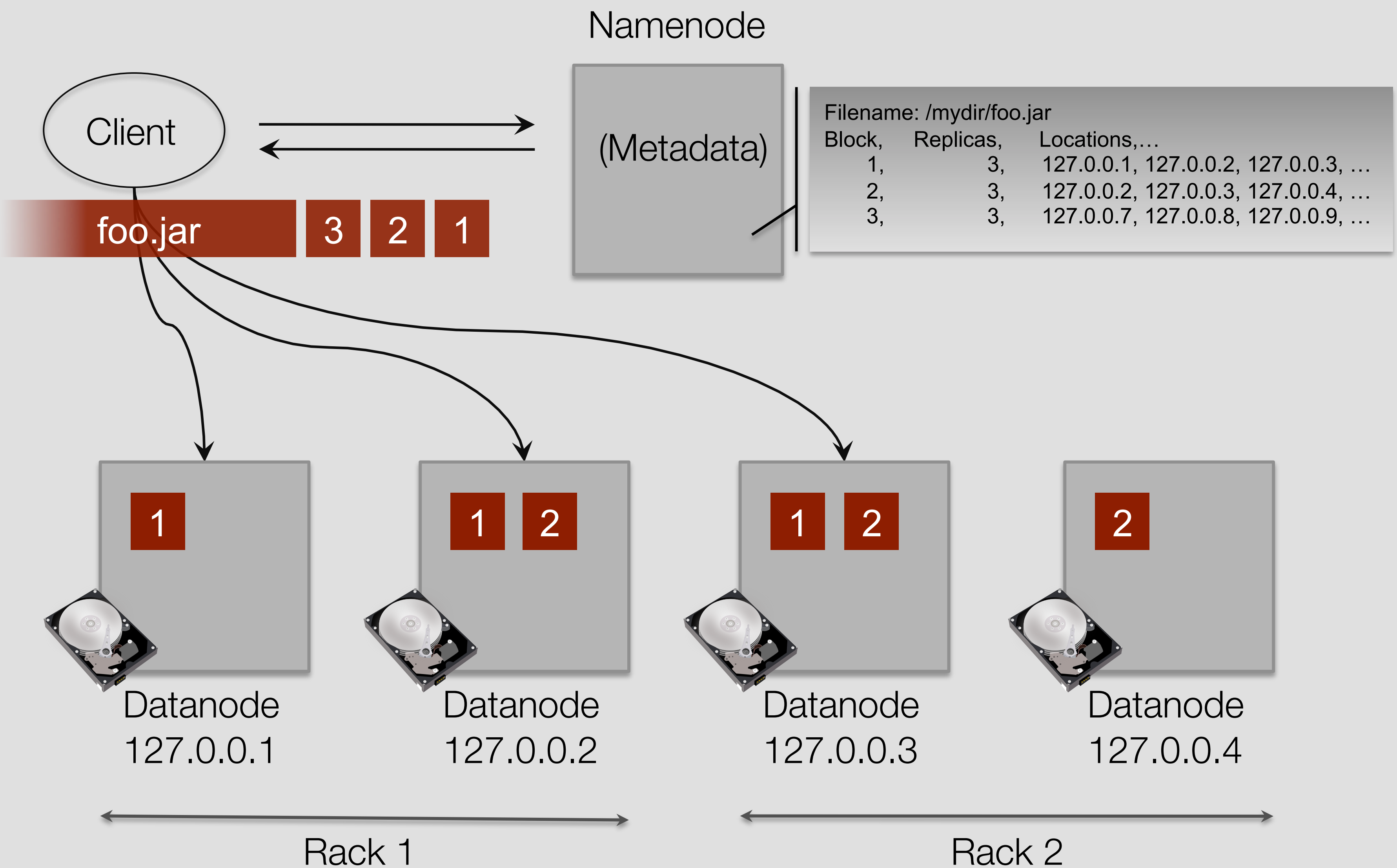
Hadoop Distributed File System

- Breaks up input data into block of fixed length
- Sends the blocks to several machines in the cluster
- A block is stored several times to prevent losses.

Quite simple, Master-Slave architecture…

- One machine remembers what is where: **Namenode**
- All the other machines just store blocks: **Datanodes**

HDFS


Namenode
Client
(Metadata)
Filename: /mydir/foo.jar
Block, Replicas, Locations,…
1, 3, 127.0.0.1, 127.0.0.2, 127.0.0.3, …
2, 3, 127.0.0.2, 127.0.0.3, 127.0.0.4, …
3, 3, 127.0.0.7, 127.0.0.8, 127.0.0.9, …
foo.jar
3
2
1
1
1 2
1 2
2
Datanode
127.0.0.1
Datanode
127.0.0.2
Datanode
127.0.0.3
Datanode
127.0.0.4
Rack 1
Rack 2

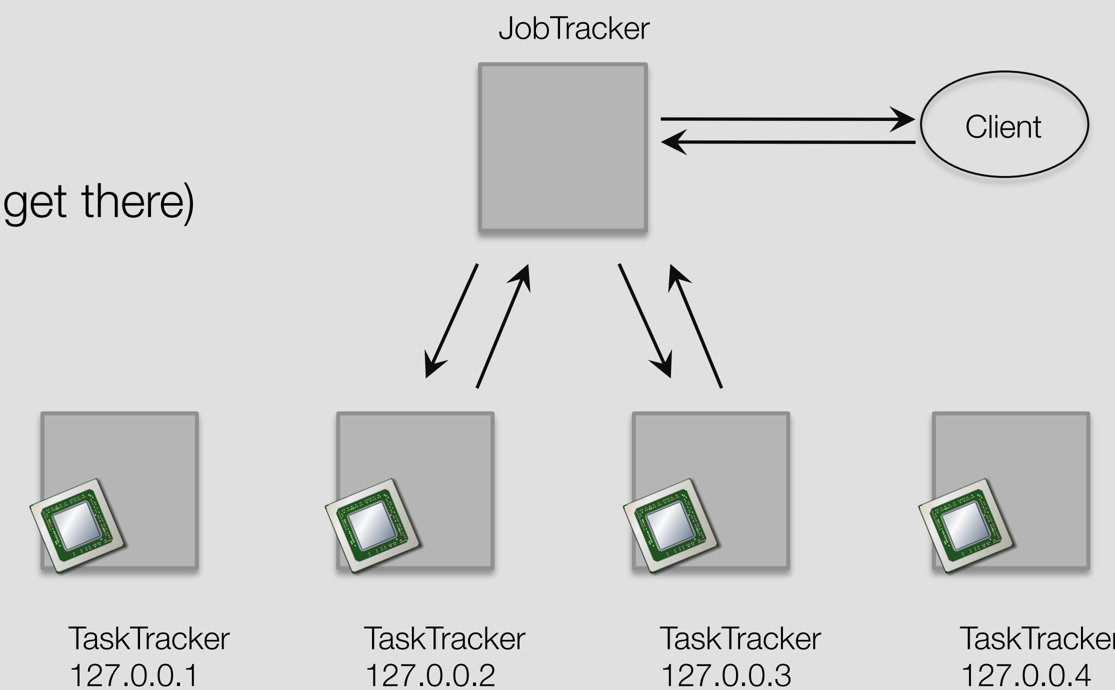
Hadoop MapReduce Engine

The computational platform

- Also a simple Master – Slave architecture.
- Clients send jobs to the **JobTracker**
- The JobTracker splits the work into small tasks and assigns them to the **TaskTrackers.**

What kind of jobs?

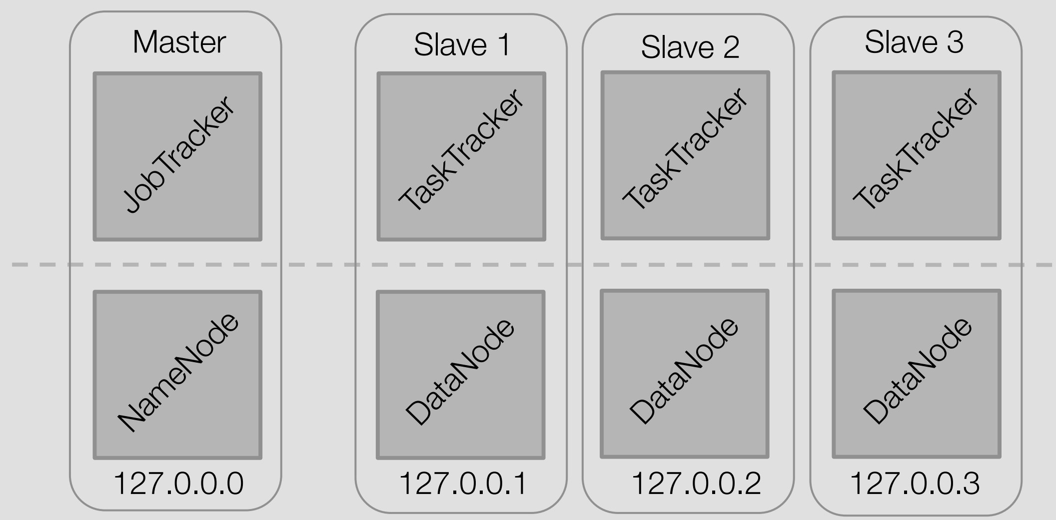
- MapReduce jobs. (We will get there)



Hadoop Summary

Software for Reliable Scalable Distributed Computing

- Master – Slave organized cluster
- MapReduce Layer
- HDFS Layer
- Many more…



MapReduce

A **programming paradigm**

- for processing large datasets, typically in a cluster of computers

"MapReduce: Simplified Data Processing on Large Clusters"

Jeffrey Dean and Sanjay Ghemawat of Google

Appeared in:
OSDI'04: Sixth Symposium on Operating System Design and Implementation,
San Francisco, CA, December, 2004.

Cited: About 3660 times

MapReduce

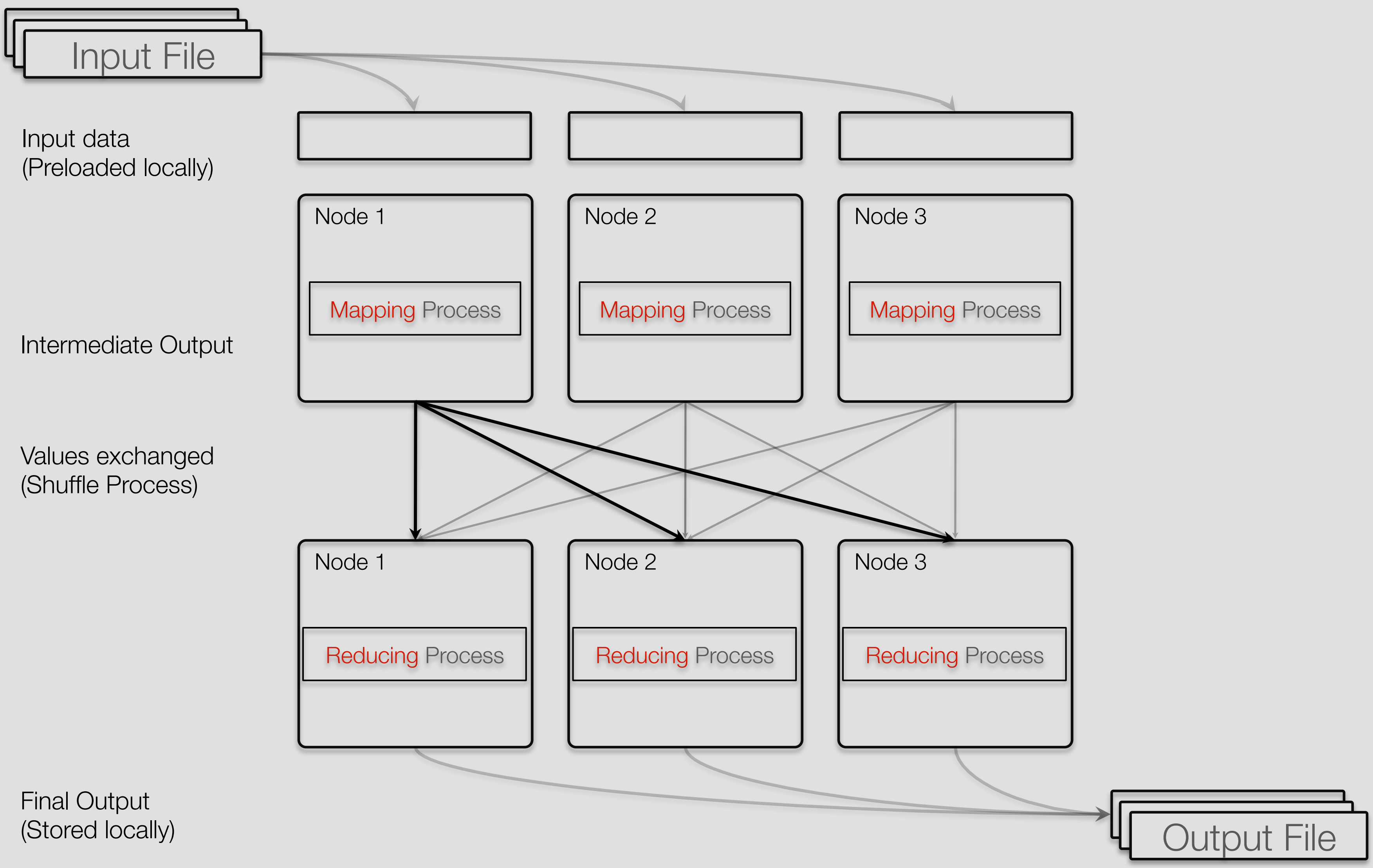
A **programming paradigm**

- for processing large datasets, typically in a cluster of computers.
- Basic Idea: minimal data transfer - send the code to the data and execute,
- in 2 steps:
  - **Map step**: parts of the file are processed in parallel and produce some intermediate output.
  - **Reduce step**: intermediate output of all individual parts is combined to create the final output.

Visualization coming up…

- Need not worry about fault tolerance, parallelization, status and monitoring tools. All taken care by the system.
- Only worry about your algorithm: Map and Reduce steps.

MapReduce

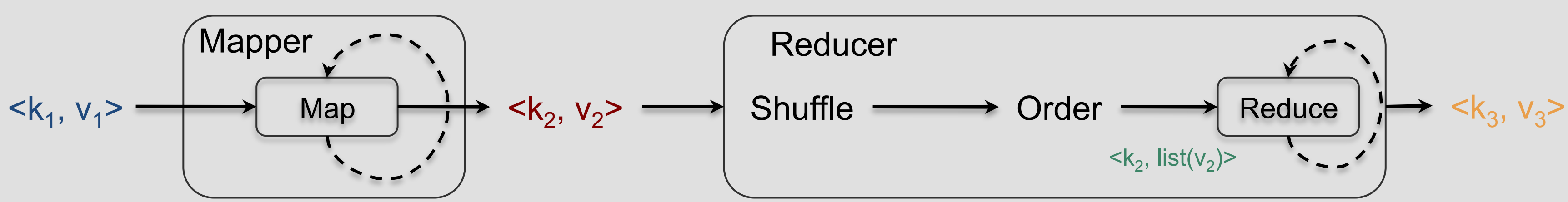


Input File
Input data
(Preloaded locally)
Node 1
Mapping Process
Node 2
Mapping Process
Node 3
Mapping Process
Intermediate Output
Values exchanged
(Shuffle Process)
Node 1
Reducing Process
Node 2
Reducing Process
Node 3
Reducing Process
Final Output
(Stored locally)
Output File

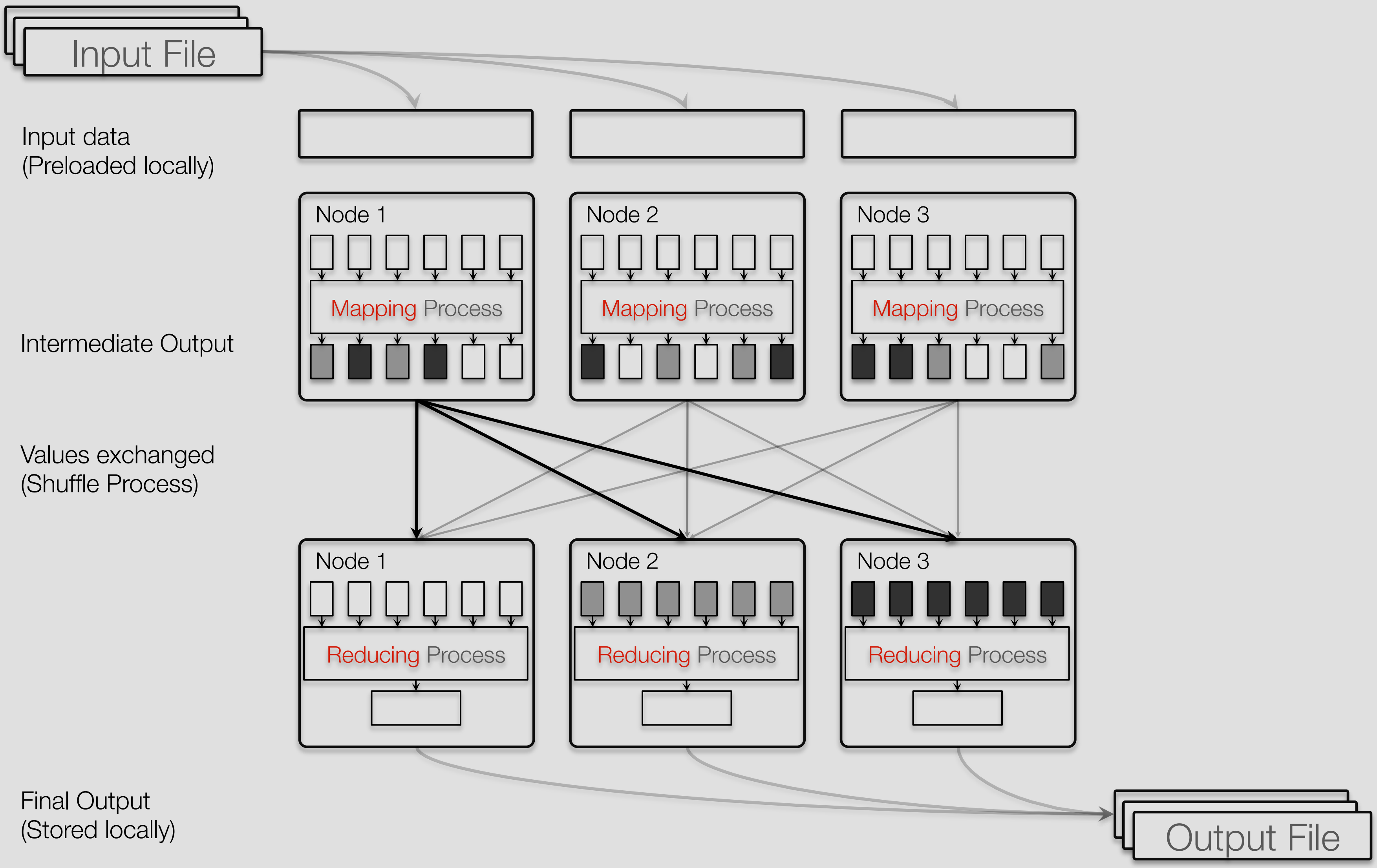
MapReduce

A little more detail…

- A file consists of Records (e.g. in a text file, a line can be a "Record").
- Each Record is fed to a map functions as a $<k_1, v_1>$ pair and processed independently.
- Mappers emit other $<k_2, v_2>$ pairs.
- Mapper outputs are hashed by key so that all $<k_2, v_2>$ pairs go to the same Reducer and grouped by key to produce $<k_2, list(v_2)>$.
- Finally, Reducers emit $<k_3, v_3>$ pairs that form the final output.



MapReduce


Input File
Input data
(Preloaded locally)
Node 1
Mapping Process
Node 2
Mapping Process
Node 3
Mapping Process
Intermediate Output
Values exchanged
(Shuffle Process)
Node 1
Reducing Process
Node 2
Reducing Process
Node 3
Reducing Process
Final Output
(Stored locally)
Output File

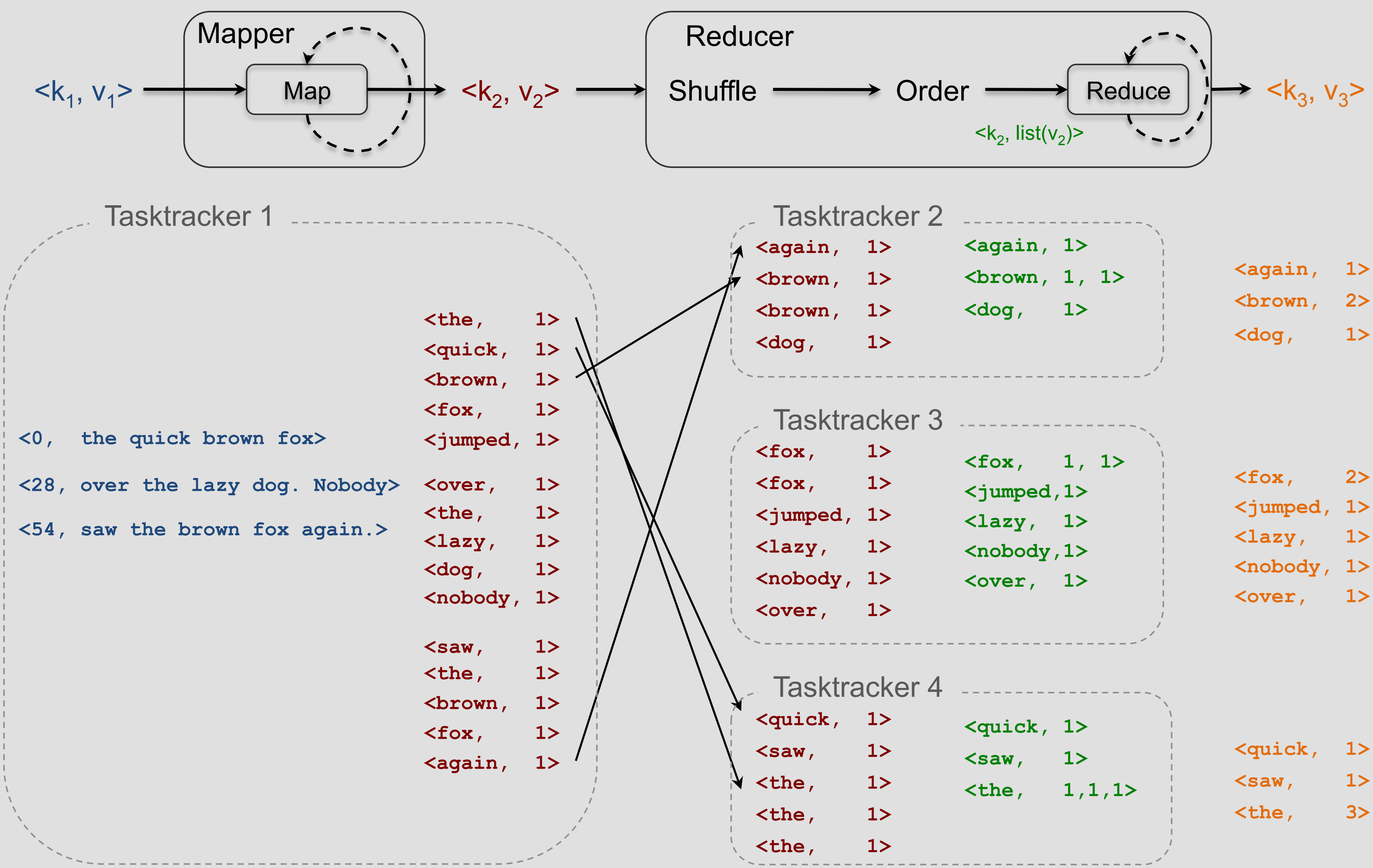
The Word Count Example

```
The quick brown fox jumped
over the lazy dog. Nobody
saw the brown fox again.
```

“Hello World”

```
again,  1
brown,  2
dog,    1
fox,    2
jumped, 1
lazy,   1
nobody, 1
over,   1
quick,  1
saw,    1
the,    3
```

The Word Count Example



MapReduce

Can any problem be solved in MapReduce?

Short answer: No!

Even fewer in a single MapReduce cycle…

MapReduce

But, using…

- a few iterations, or
- chains of programs, and
- smart choice of <key, value> pairs…

… quite a lot can be done!

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

A simple, non-trivial example

Counting Triangles
and the Curse of the Last Reducer

Siddharth Suri, Sergei Vassilvitskii, Yahoo! Research

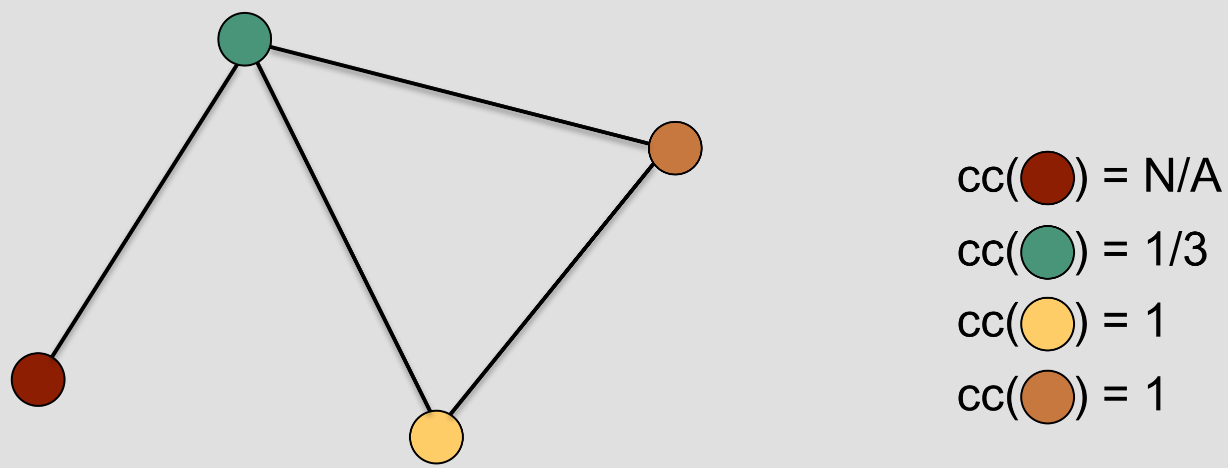
* The sequel contains extended parts of Sergei's presentation.

Counting Triangles

Why? Clustering Coefficient:

Given a graph $G = (V, E)$ the Clustering Coefficient cc(v) of a vertex v is the fraction of pairs of neighbors of v that are also neighbors:

$$cc(v) = \frac{|\{(u,w) \in E | u, w \in N(v)\}|}{\binom{\deg(v)}{2}}$$

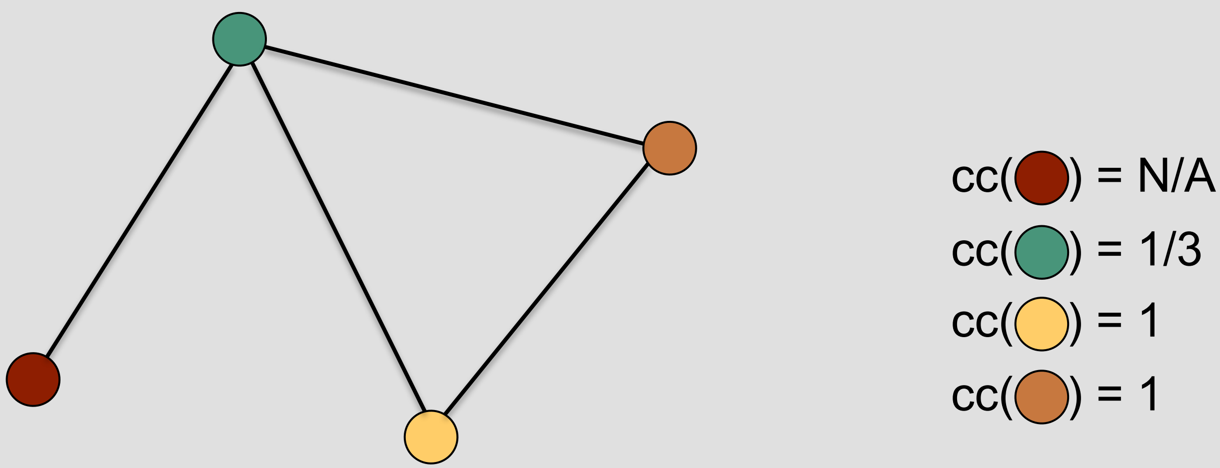


Counting Triangles

Why? Clustering Coefficient:

Given a graph $G = (V, E)$ the Clustering Coefficient cc(v) of a vertex v is the fraction of pairs of neighbors of v that are also neighbors:

$$cc(v) = \frac{\#\Delta' s \text{ incident to } v}{\binom{\deg(v)}{2}}$$

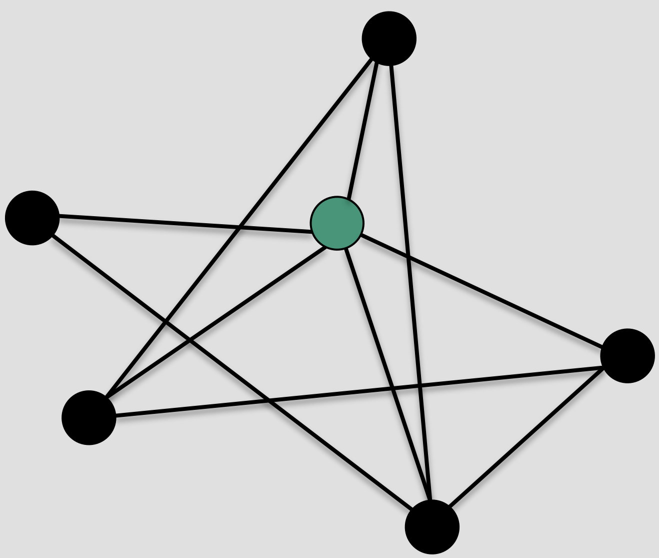


Counting Triangles

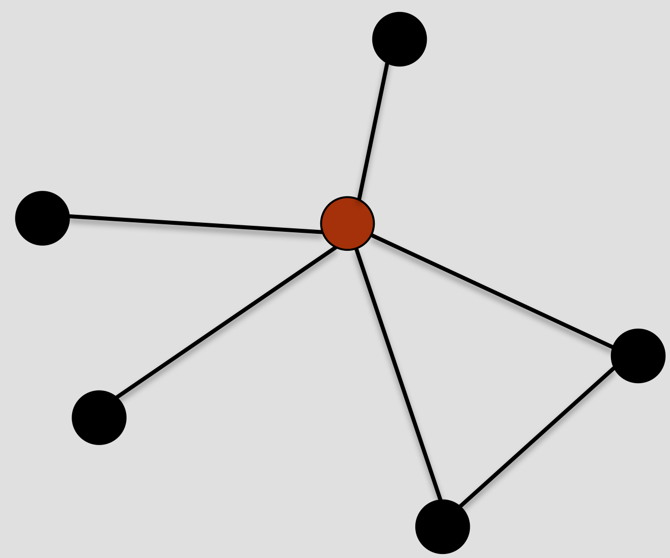
Why Clustering Coefficient ?

The Clustering Coefficient captures how tight a network is around a node.

cc(●) = 1/2



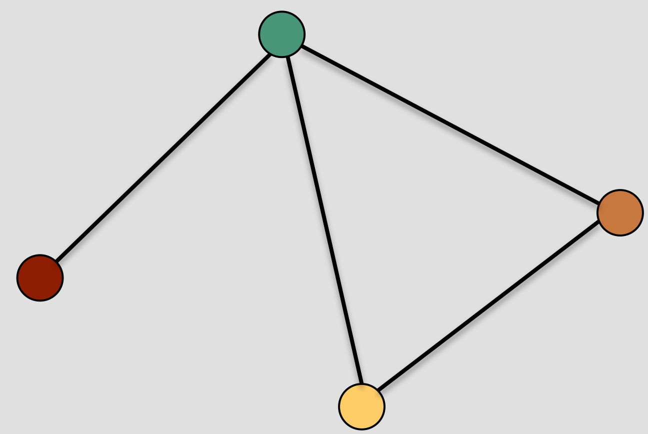
cc(●) = 1/10



Counting Triangles

Sequential Algorithm

```
T = 0;
foreach v in V do
    foreach u,w pair in N(v) do
        if (u, w) in E then
            T = T + 1;
return T;
```



- Running time $\sum_{v \in V} \deg(v)^2$
- Even for sparse graphs can be quadratic in the number of edges if a vertex has high degree.
- It happens in natural graphs.

Counting Triangles

Sequential Algorithm 2 (Schank '07)

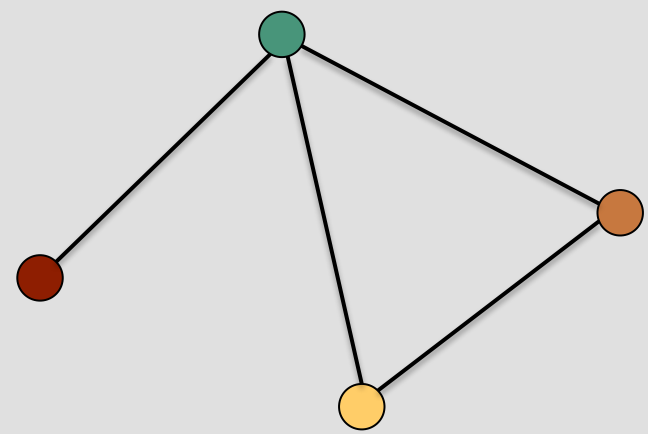
```
T = 0;

foreach v in V do

    foreach u,w pair in N(v) do

        if deg(u) > deg(v) &&
         deg(w) > deg(v), then
          if (u, w) in E then
           T = T + 1;

return T;
```



- Running time $O(m^{3/2})$
- There exists graph for which we cannot do better.

Counting Triangles in M/R

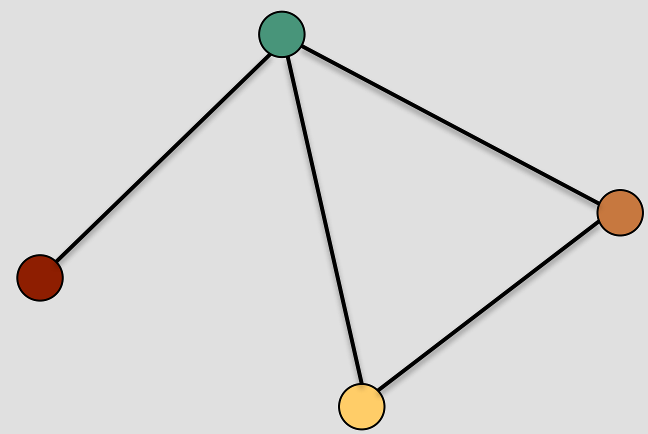
```
Map 1:    Input <(u,v); 0>
          if deg(v) > deg(u), then
             emit <u;v>

Reduce 1: Input <v; S subset of N(v)>
          for (u,w) : u,w in S, do
               emit <v;(u,w)>

Map 2:    if Input of type <v;(u,w)> then
             emit <(u,w); v>

          if Input of type <(u,v);0> then
             emit <(u,v); $>

Reduce 2: Input <(u,w); S subset of Union(V, {$})>
          if $ appears in S then
             for v in S except $ do
                emit <v;1>    //emit <u;1>, emit <w;1>
```



Counting Triangles in M/R

Analysis

- How much main memory per reduce call? (sublinear in the input).
- Total memory used for the computation on the cluster (not to exceed n^2)
- Number of rounds.