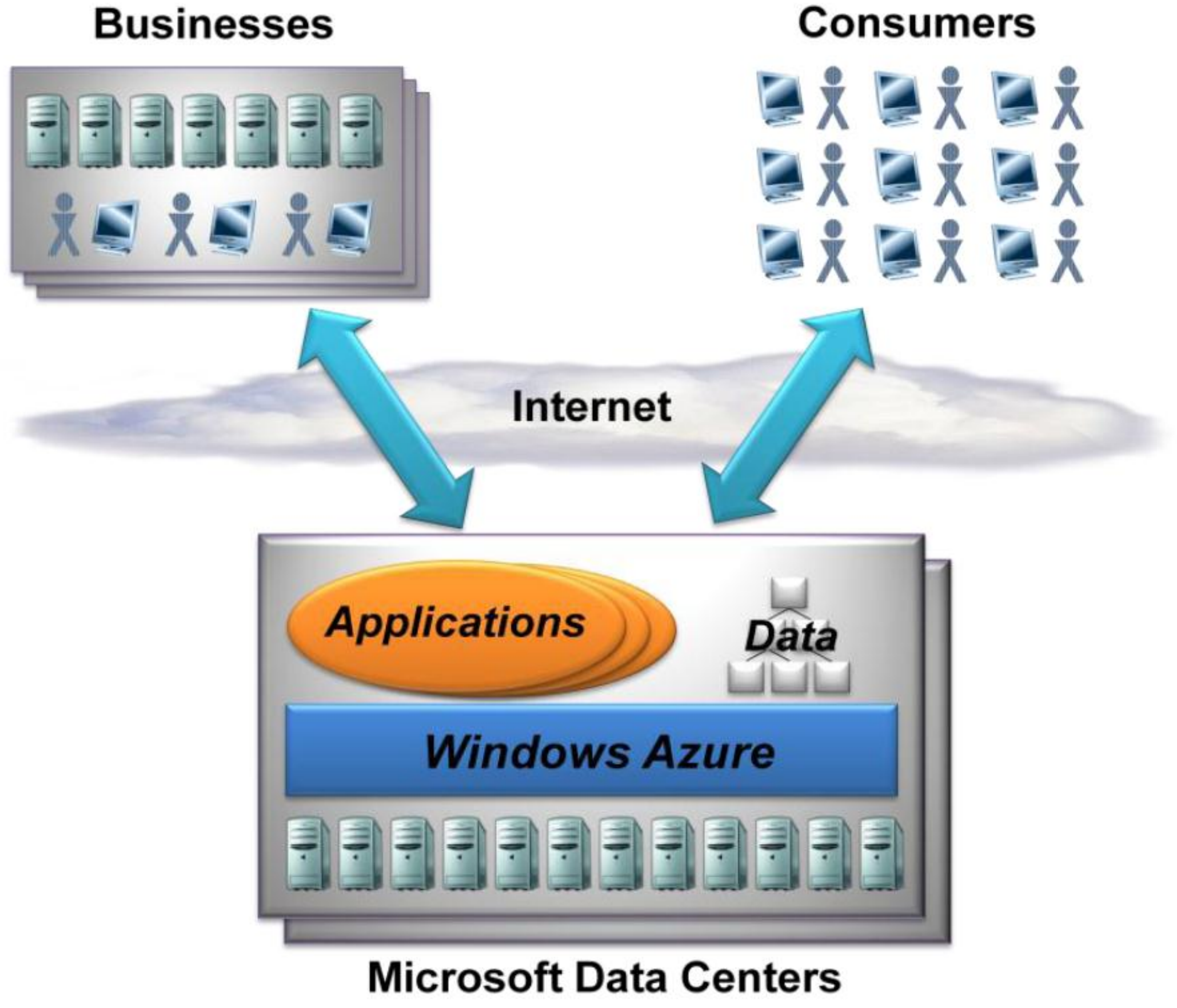
# Introduction to Cloud Computing and Social Network Analysis

高榮鴻 副教授
國立交通大學
電機工程學系/電信工程研究所
Email: runghung@mail.nctu.edu.tw

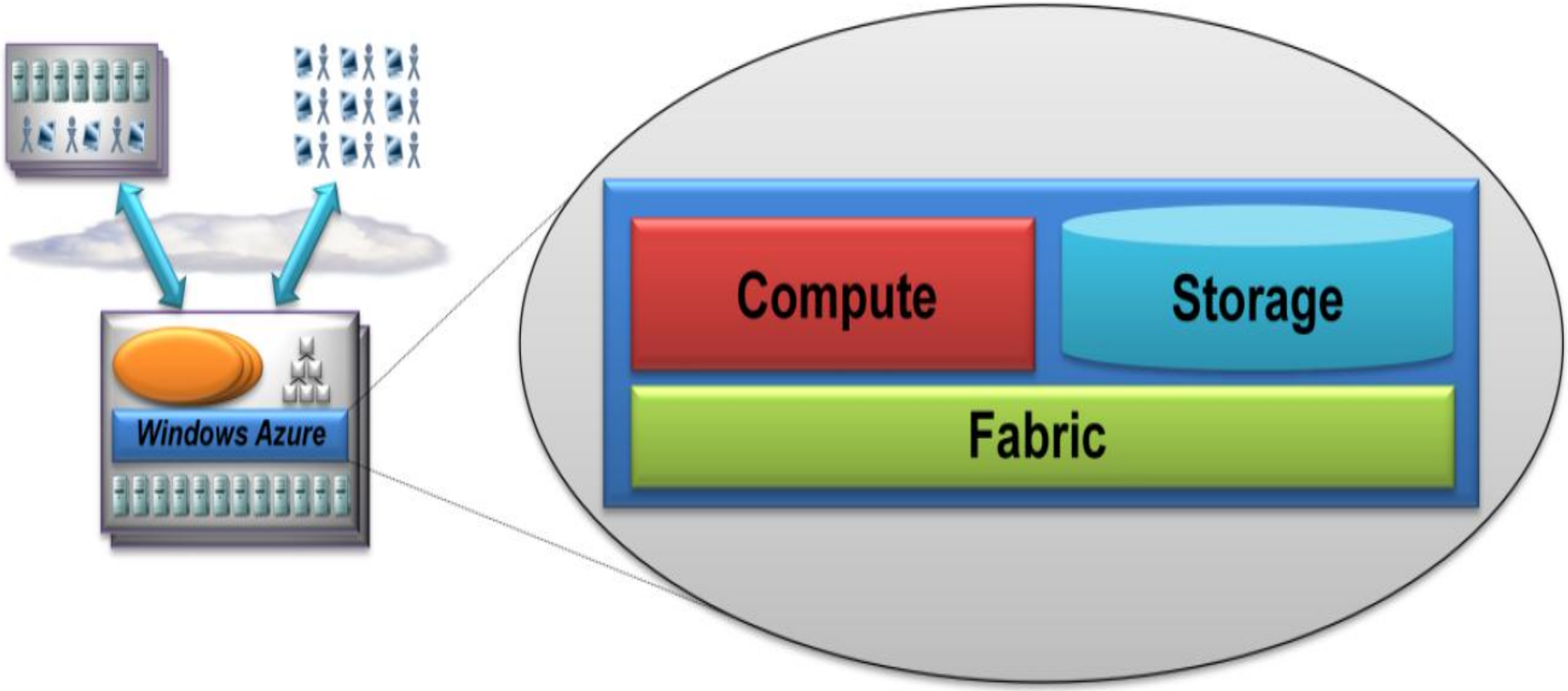
## Outline

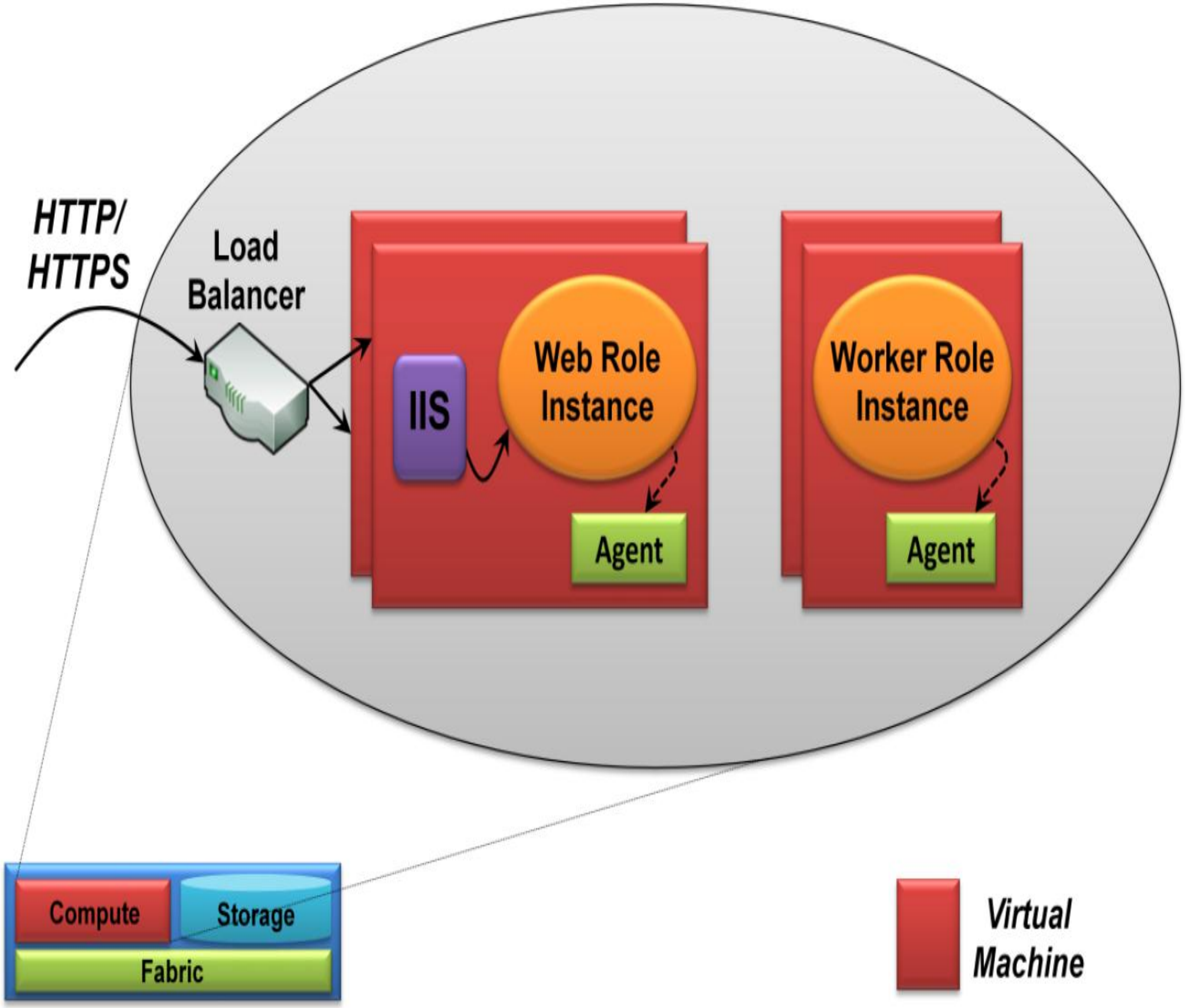
- Enabling Technologies for Cloud Computing
- Social Network Analysis based on Cloud Computing

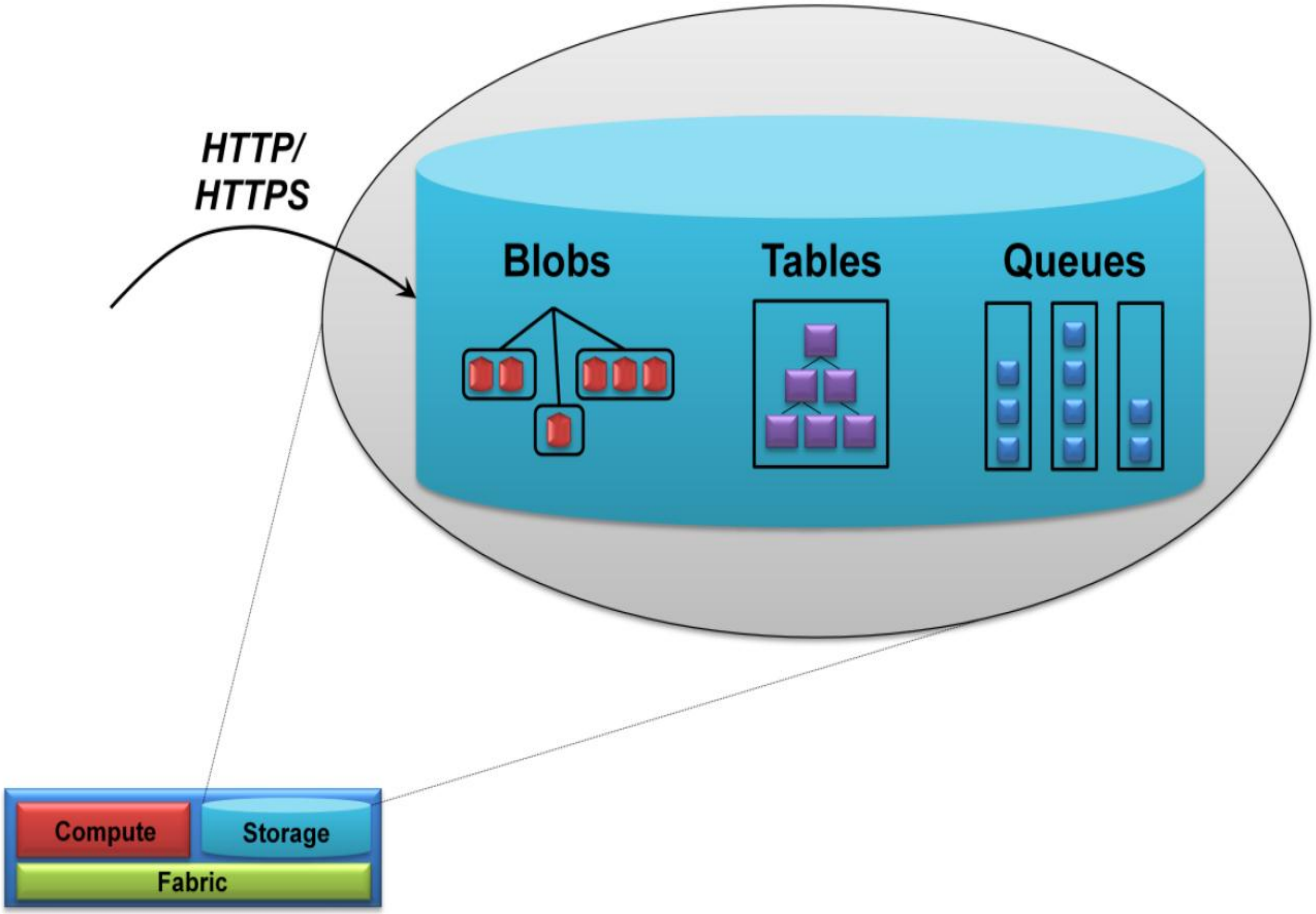

Businesses
Consumers
Internet
Applications
Data
Windows Azure
Microsoft Data Centers

# Three Enabling Technologies for Cloud Computing

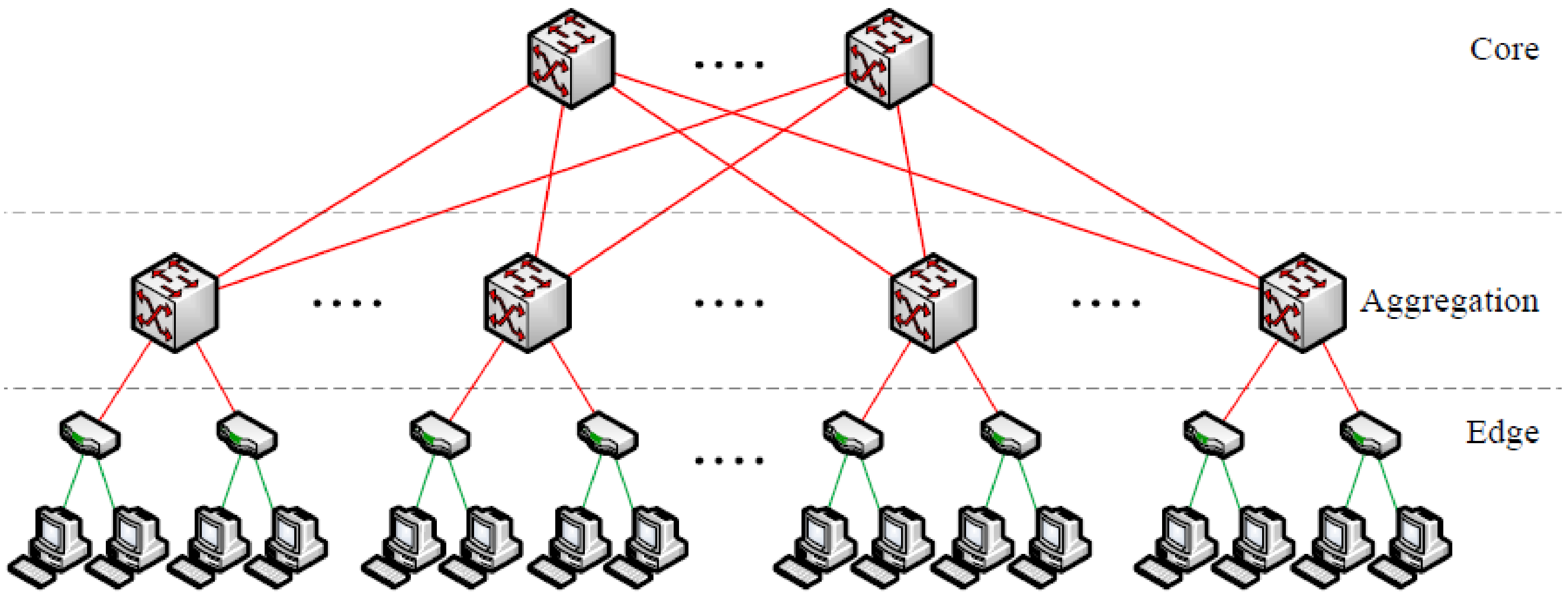
- Networking
  - Data center networks
- (Mass) Storage
  - BigTable
- (Parallel and Distributed) Computing
  - MapReduce
  - Google Application Engine
  - Windows Azure


Windows Azure
Compute
Storage
Fabric

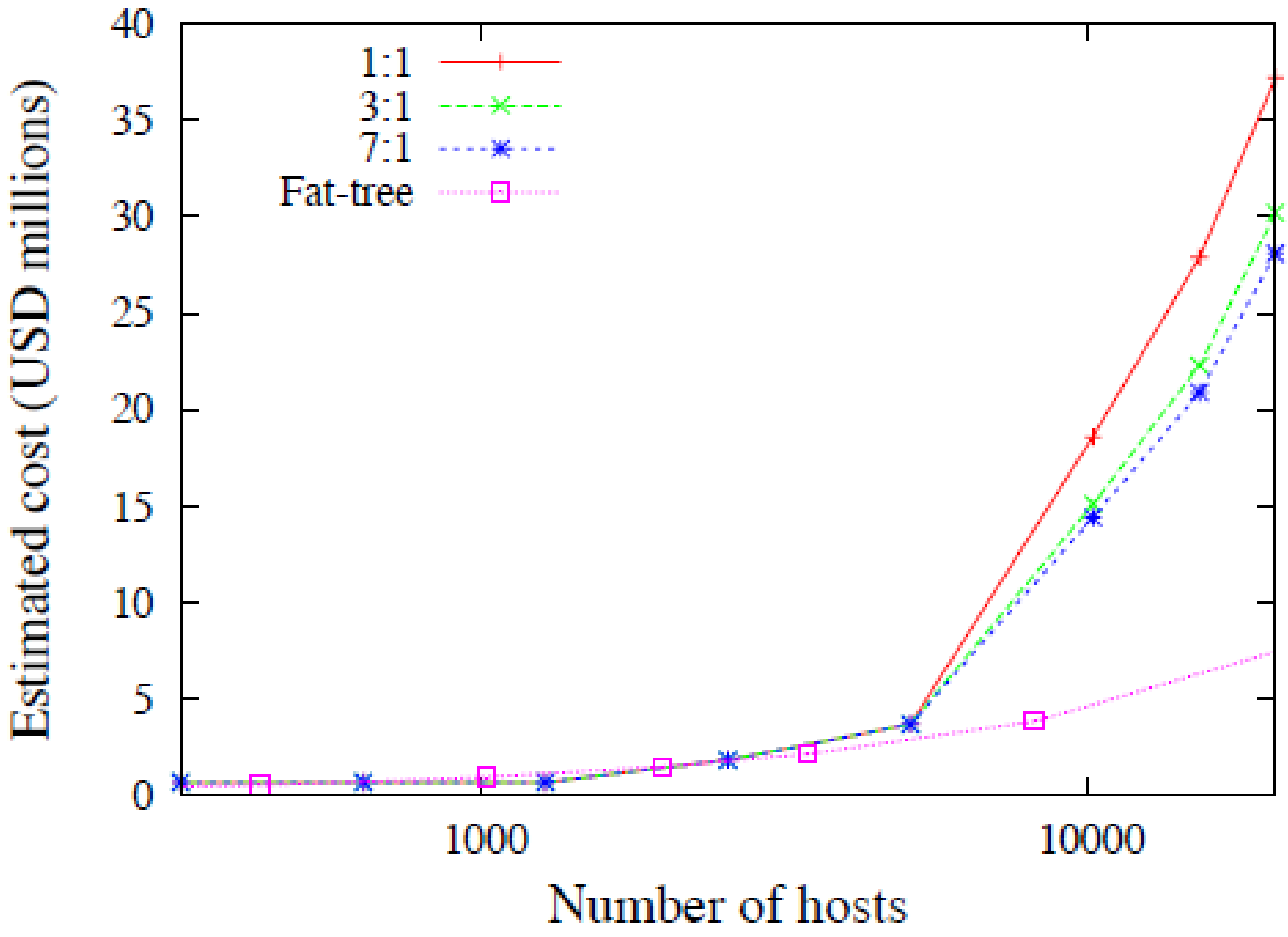

HTTP/
HTTPS
Load
Balancer
IIS
Web Role
Instance
Agent
Worker Role
Instance
Agent
Compute
Storage
Fabric
Virtual
Machine


HTTP/
HTTPS
Blobs
Tables
Queues
Compute
Storage
Fabric

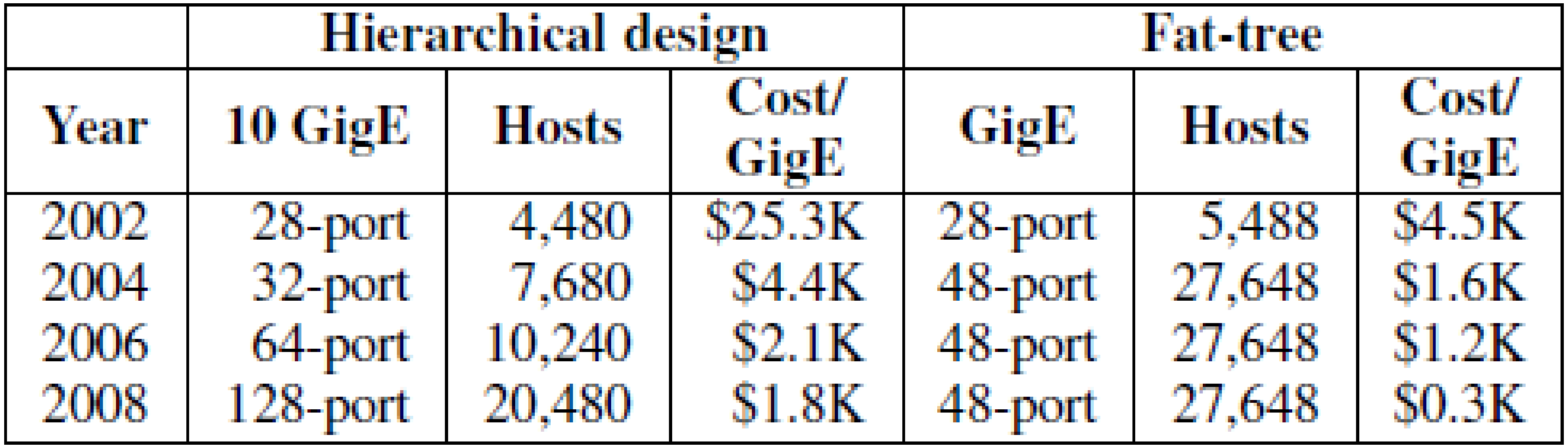
## Networking for Cloud Computing



**Common data center interconnect topology. Host to switch links are GigE and links between switches are 10 GigE.**

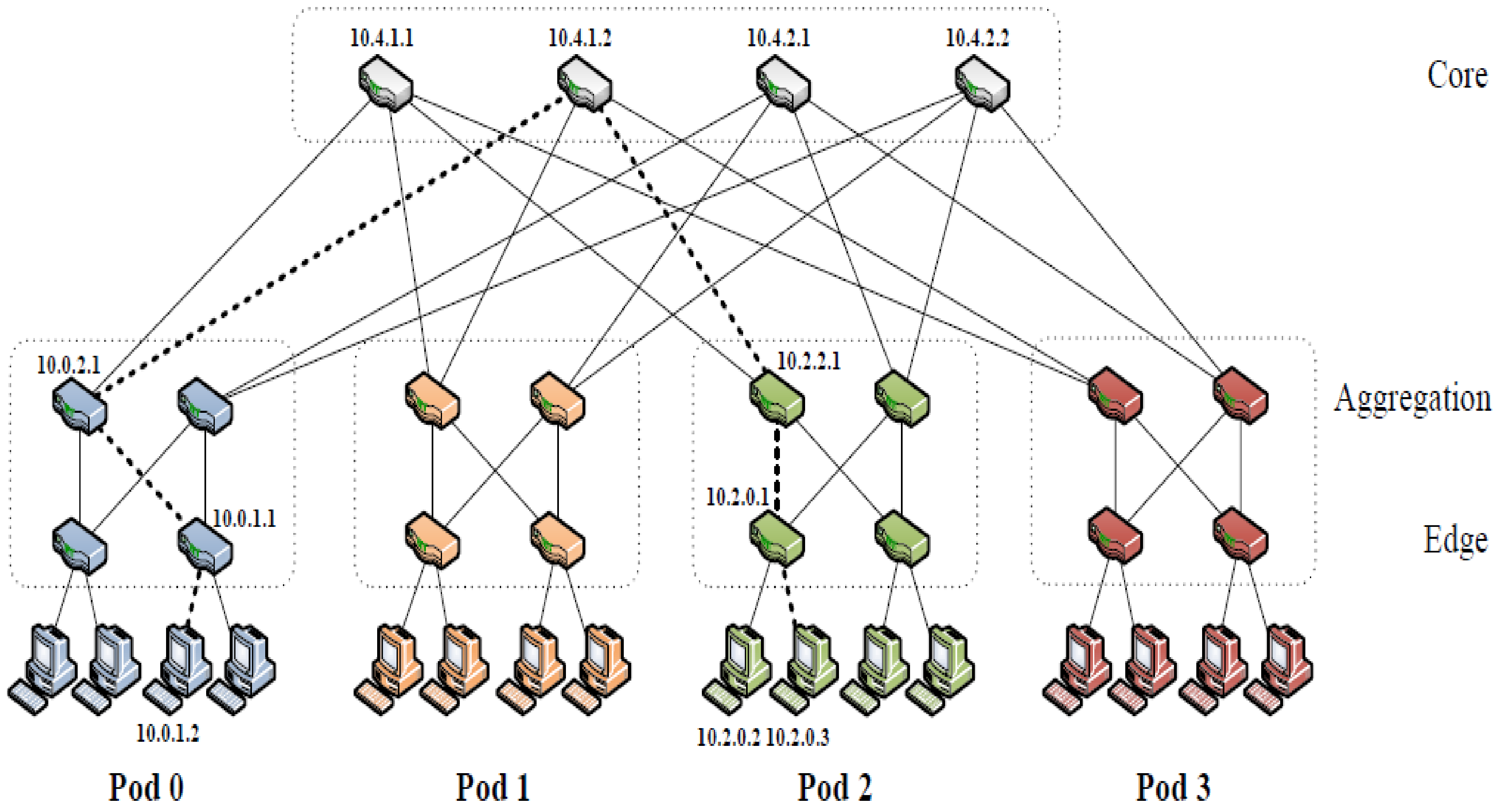


**Current cost estimate vs. maximum possible number of hosts for different oversubscription ratios.**



| | Hierarchical design | | | Fat-tree | | |
|---|---|---|---|---|---|---|
| Year | 10 GigE | Hosts | Cost/ GigE | GigE | Hosts | Cost/ GigE |
| 2002 | 28-port | 4,480 | $25.3K | 28-port | 5,488 | $4.5K |
| 2004 | 32-port | 7,680 | $4.4K | 48-port | 27,648 | $1.6K |
| 2006 | 64-port | 10,240 | $2.1K | 48-port | 27,648 | $1.2K |
| 2008 | 128-port | 20,480 | $1.8K | 48-port | 27,648 | $0.3K |

**The maximum possible cluster size with an oversubscription ratio of 1:1 for different years.**



Simple fat-tree topology. Using the two-level routing tables described in Section 3.3, packets from source 10*.0.1.2 to* destination 10*.2.0.3 would take the dashed path.*

## Fat Trees

- There are *k pods, each* containing two layers of *k/2 switches.*
- *Each k-port switch in the* lower layer is directly connected to *k/2 hosts.*
- *Each of the remaining k/2 ports is connected to k/2 of the k ports in the aggregation* layer of the hierarchy.

## Fat Trees

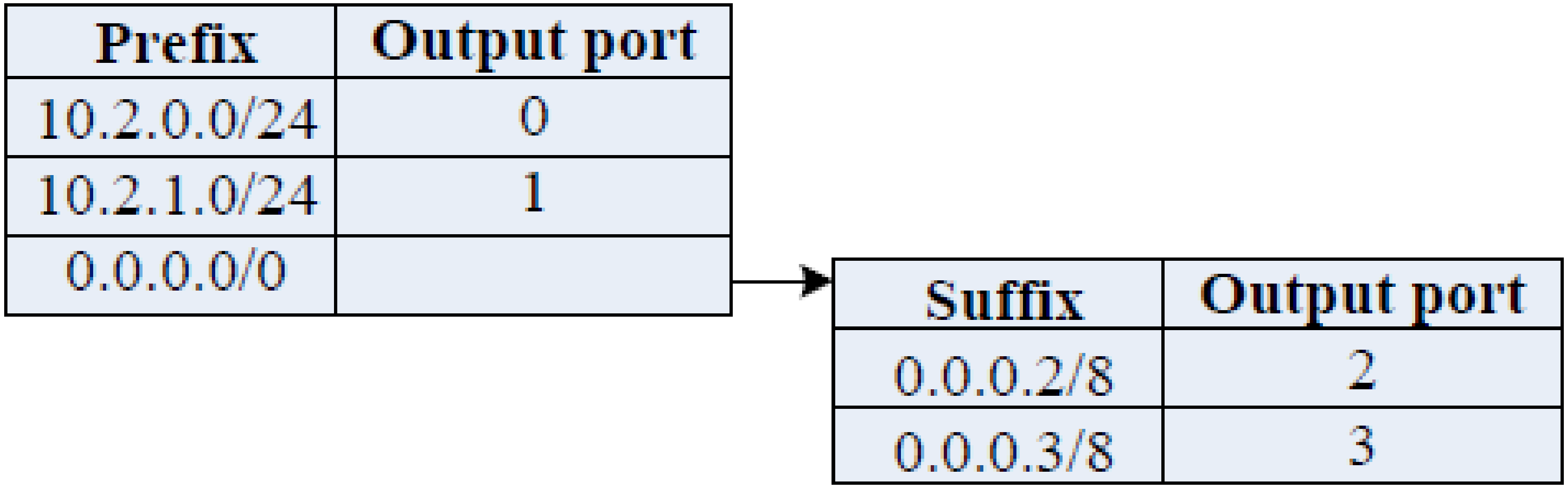
- There are (*k/2)^2 k-port core switches. Each core switch has one* port connected to each of *k pods.*
- The *i-th port of any core switch* is connected to pod *i such that consecutive ports in the aggregation* layer of each pod switch are connected to core switches on (*k/2)* strides.
- In general, a fat-tree built with *k-port switches supports (k^3)/4 hosts.*

## An Example of Fat-Tree

- A fat-tree built from 48-port GigE switches would consist of 48 pods, each containing an edge layer and an aggregation layer with 24 switches each.
- The edge switches in every pod are assigned 24 hosts each. The network supports 27,648 hosts, made up of 1,152 subnets with 24 hosts each.

## An Example of Fat-Tree

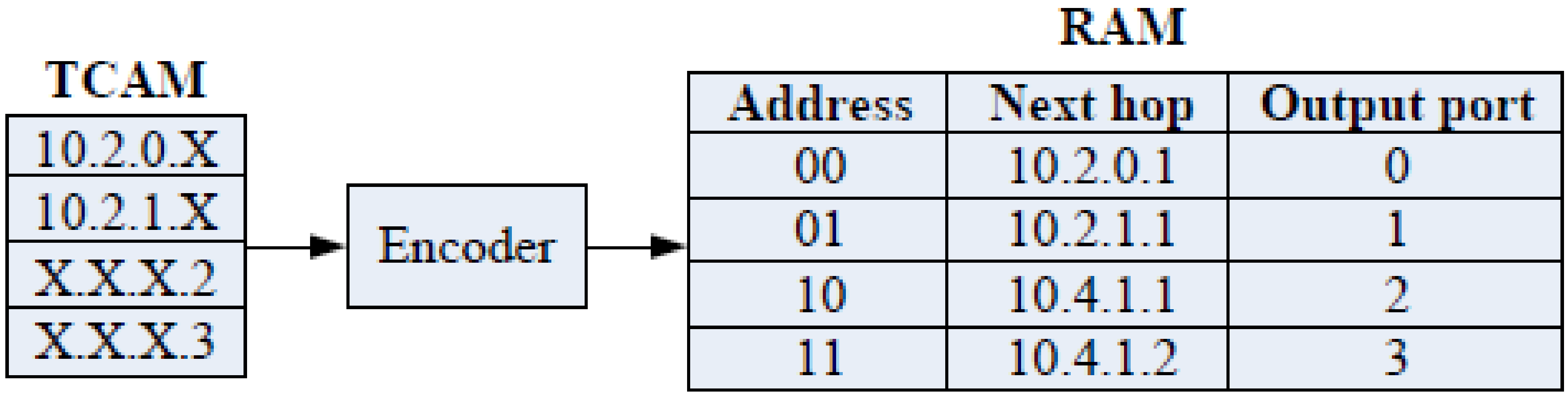
- There are 576 equal-cost paths between any given pair of hosts in different pods.
- The cost of deploying such a network architecture would be $8.*64M, compared to $37M for the traditional* techniques described earlier.



| Prefix | Output port |
|---|---|
| 10.2.0.0/24 | 0 |
| 10.2.1.0/24 | 1 |
| 0.0.0.0/0 | |

| Suffix | Output port |
|---|---|
| 0.0.0.2/8 | 2 |
| 0.0.0.3/8 | 3 |

**Two-level table example. This is the table at switch** 10.*2.2.1****. An incoming packet with destination IP address*** 10.*2.1.2* ***is forwarded on port 1, whereas a packet with destination*** **IP address 10*****.3.0.3 is forwarded on port 3.***



| Address | Next hop | Output port |
|---|---|---|
| 00 | 10.2.0.1 | 0 |
| 01 | 10.2.1.1 | 1 |
| 10 | 10.4.1.1 | 2 |
| 11 | 10.4.1.2 | 3 |

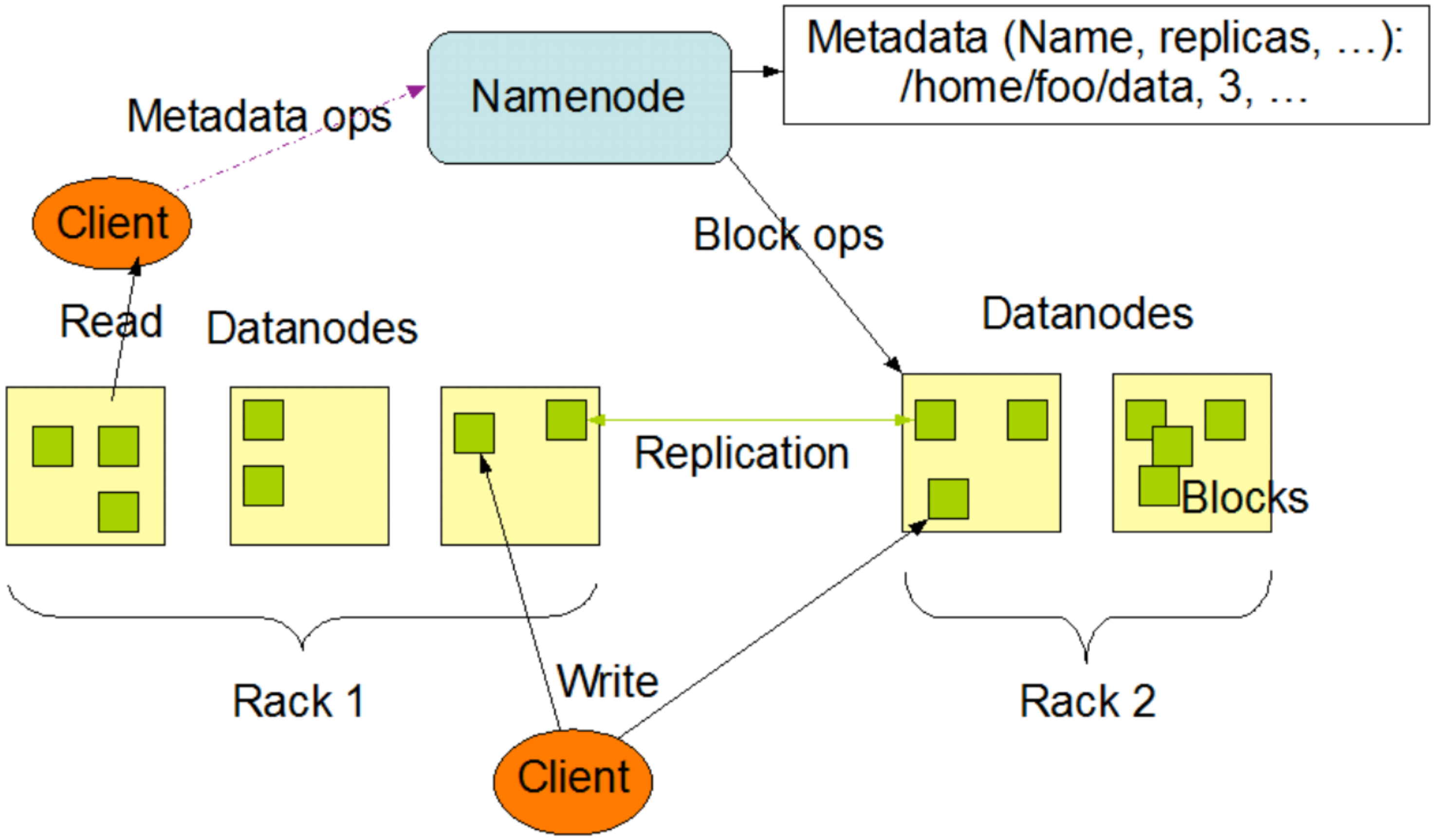
**TCAM two-level routing table implementation.**

## TCAM Implementation

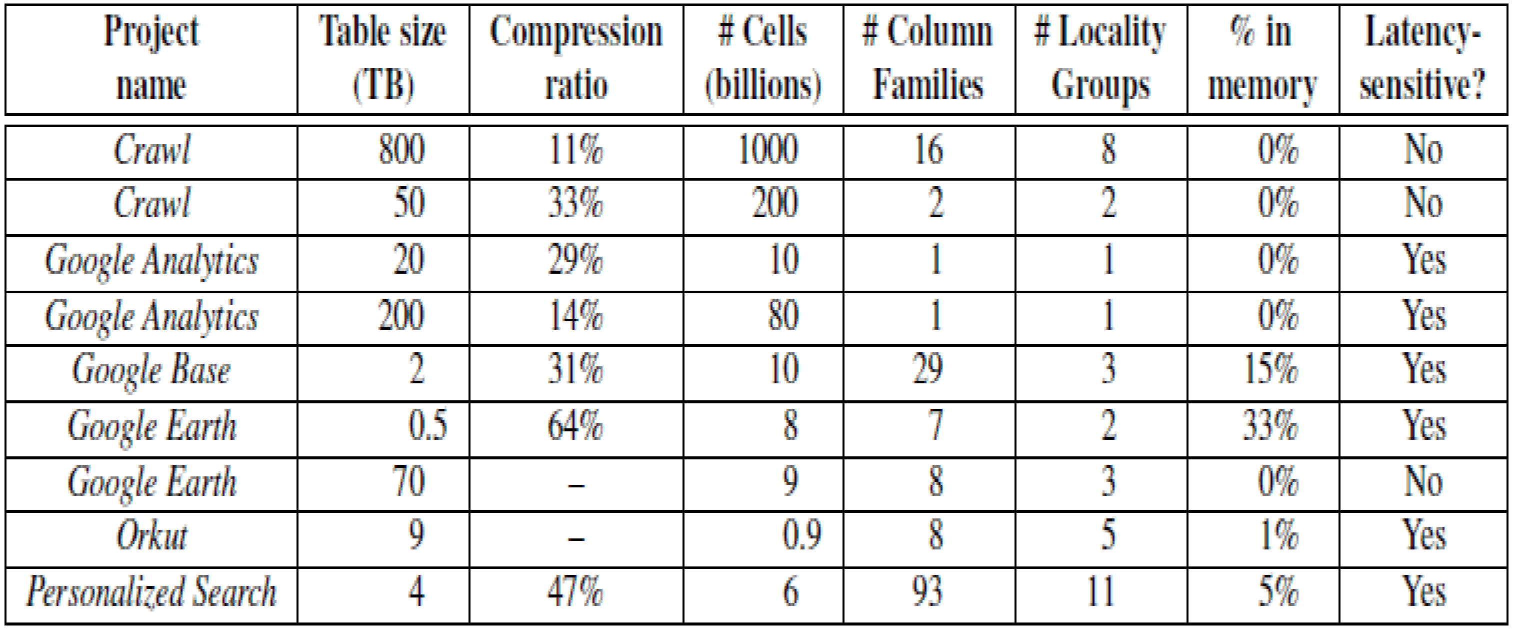
- The above figure shows our proposed implementation of the two-level lookup engine.
- A TCAM stores address prefixes and suffixes, which in turn indexes a RAM that stores the IP address of the next hop and the output port.

## Distributed Storage for Cloud Computing

# Hadoop Distributed File System

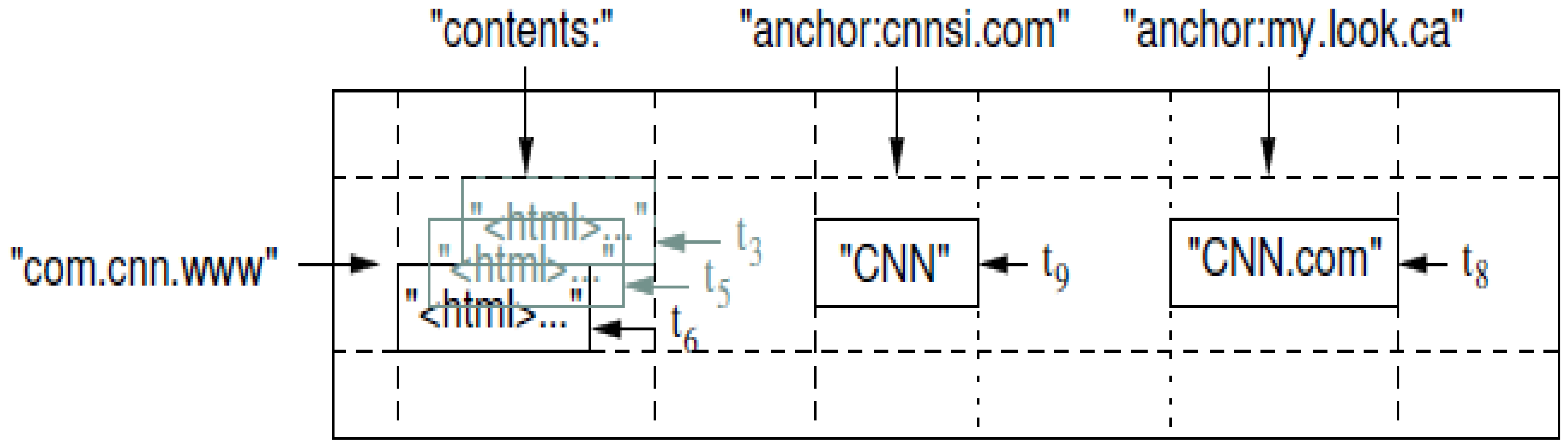


# Storage-Intensive Applications based on BigTable (in 2006)



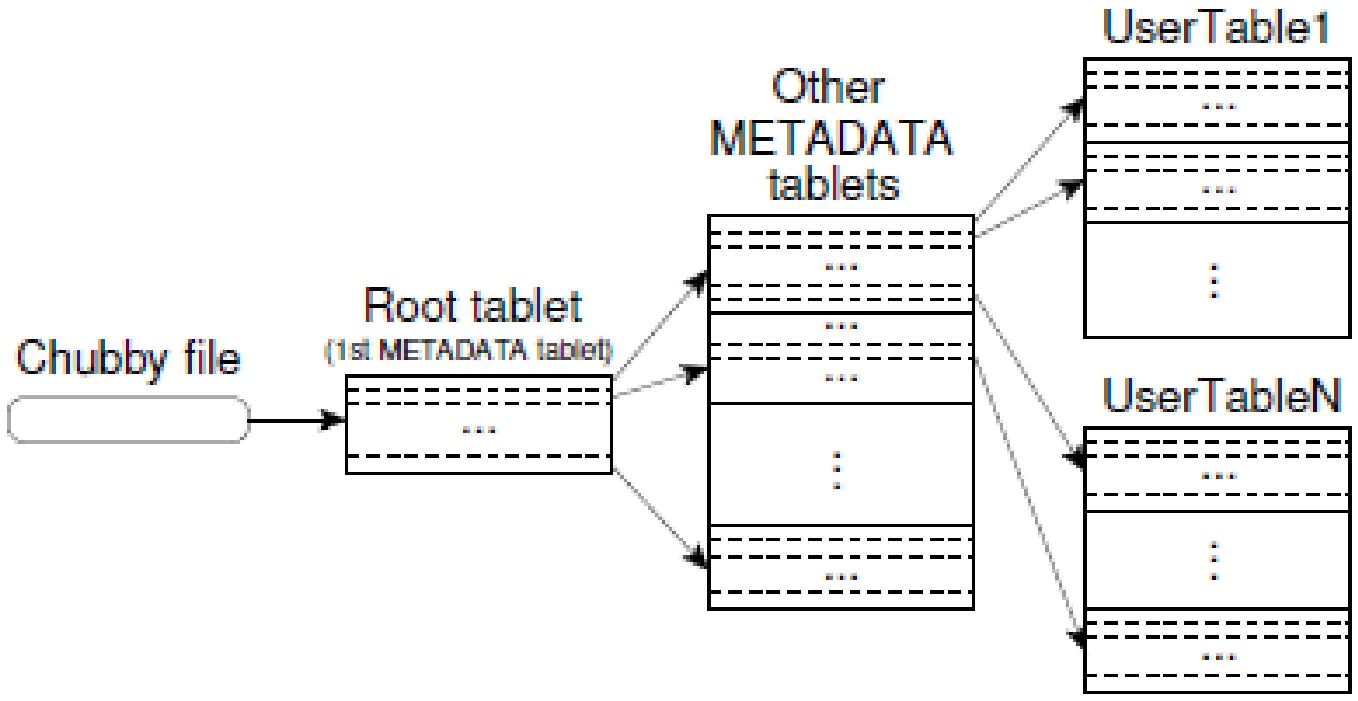
| Project name | Table size (TB) | Compression ratio | # Cells (billions) | # Column Families | # Locality Groups | % in memory | Latency-sensitive? |
|---|---|---|---|---|---|---|---|
| *Crawl* | 800 | 11% | 1000 | 16 | 8 | 0% | No |
| *Crawl* | 50 | 33% | 200 | 2 | 2 | 0% | No |
| *Google Analytics* | 20 | 29% | 10 | 1 | 1 | 0% | Yes |
| *Google Analytics* | 200 | 14% | 80 | 1 | 1 | 0% | Yes |
| *Google Base* | 2 | 31% | 10 | 29 | 3 | 15% | Yes |
| *Google Earth* | 0.5 | 64% | 8 | 7 | 2 | 33% | Yes |
| *Google Earth* | 70 | – | 9 | 8 | 3 | 0% | No |
| *Orkut* | 9 | – | 0.9 | 8 | 5 | 1% | Yes |
| *Personalized Search* | 4 | 47% | 6 | 93 | 11 | 5% | Yes |

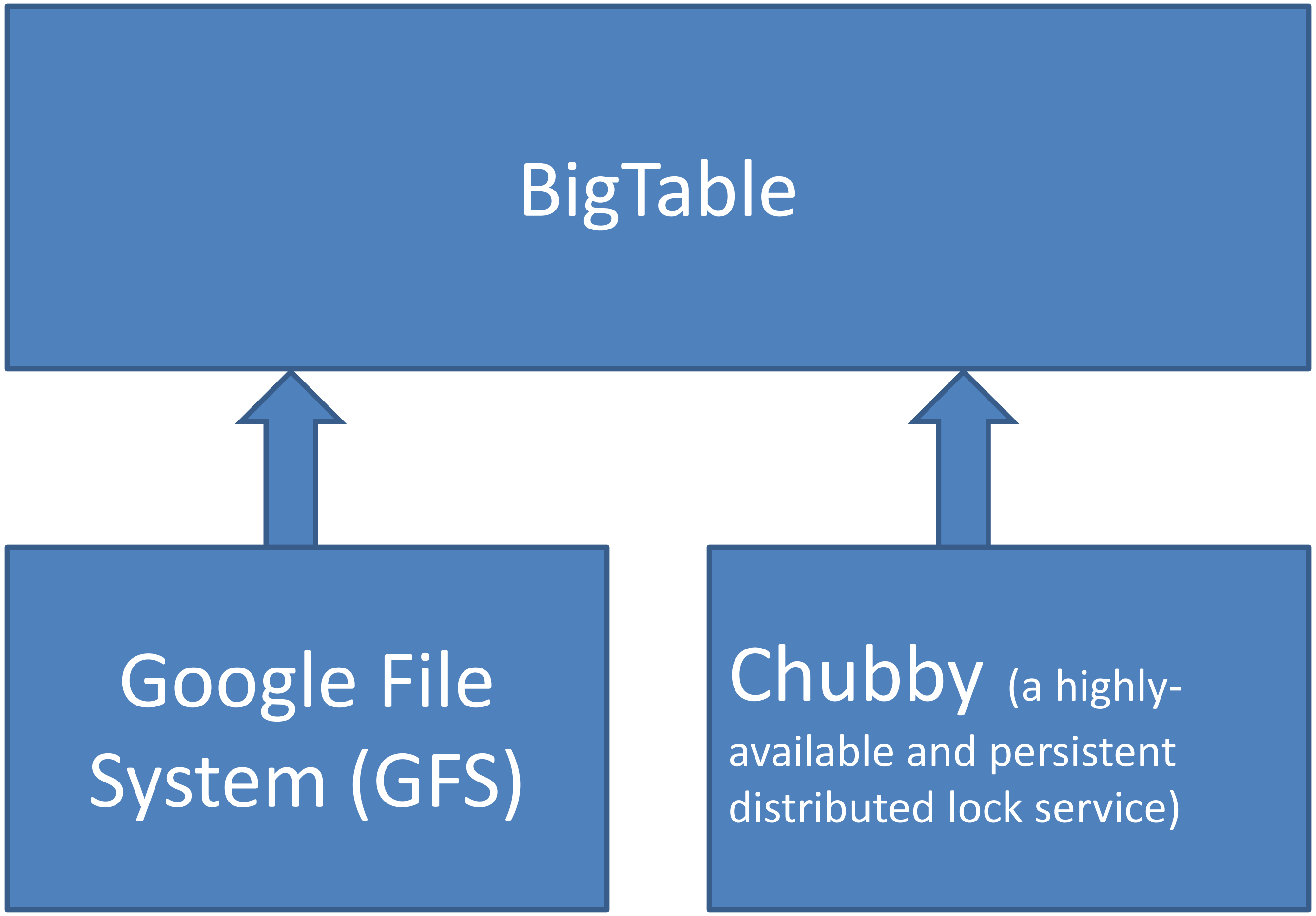
# BigTable: A Distributed Storage System for Structured Data



A slice of an example table that stores Web pages. The row name is a reversed URL. The contents column family contains the page contents, and the anchor column family contains the text of any anchors that reference the page. CNN's home page is referenced by both the Sports Illustrated and the MY-look home pages, so the row contains columns named anchor:cnnsi.com and anchor:my.look.ca. Each anchor cell has one version; the contents column has three versions, at timestamps t3, t5, and t6.

# B+ Tree for Storing Tablet Location Information




BigTable
Google File System (GFS)
Chubby (a highly-available and persistent distributed lock service)

## GFS and Chubby for BigTable

- Bigtable uses the distributed Google File System (GFS) to store
  - log files and
  - data files
- Bigtable uses Chubby for a variety of tasks:
  - to ensure that there is at most one active master at any time
  - to store the bootstrap location of BigTable data

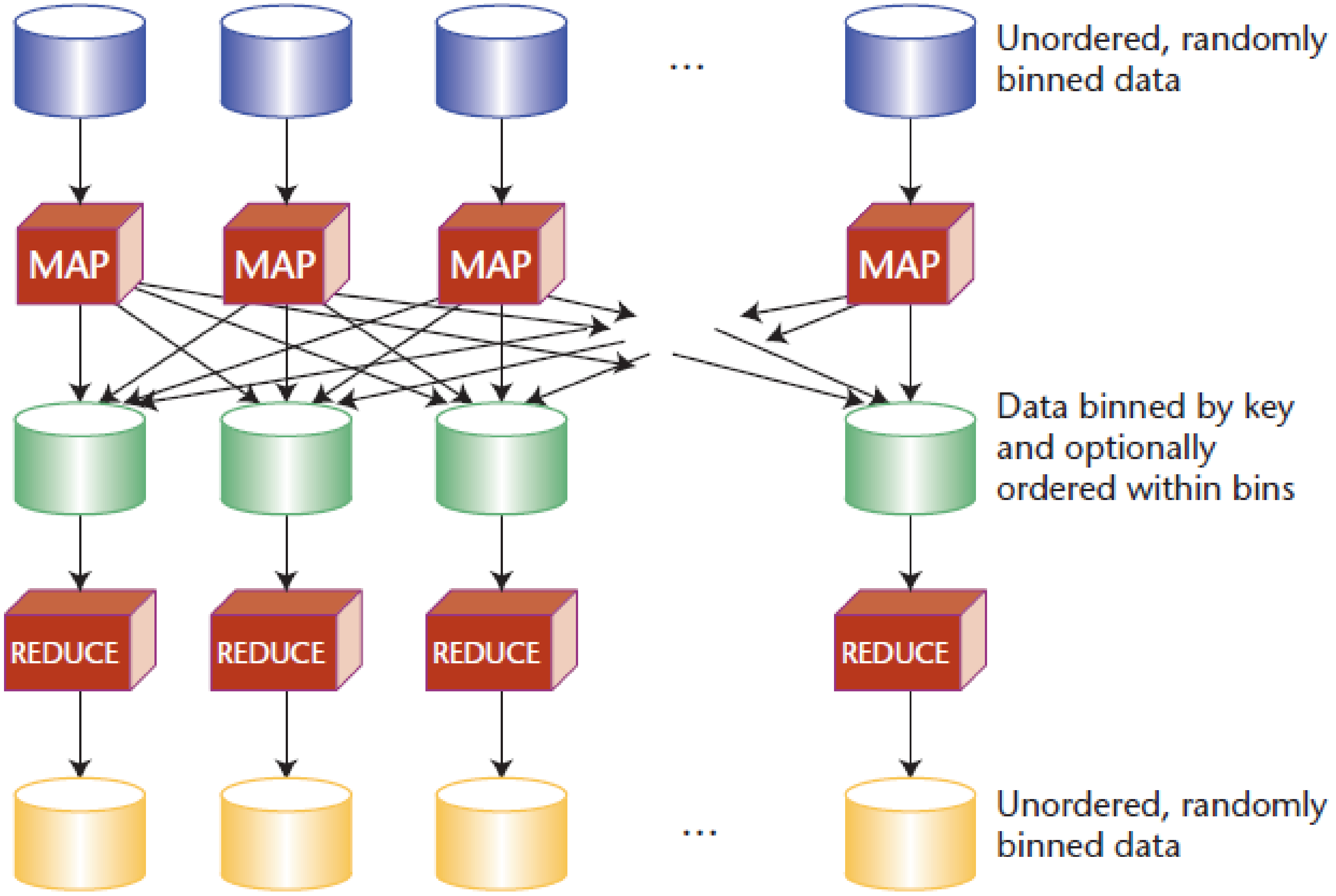
## GFS and Chubby for BigTable

- to discover tablet servers and finalize tablet server deaths
- to store Bigtable schema information (the column family information for each table);
- and to store access control lists.

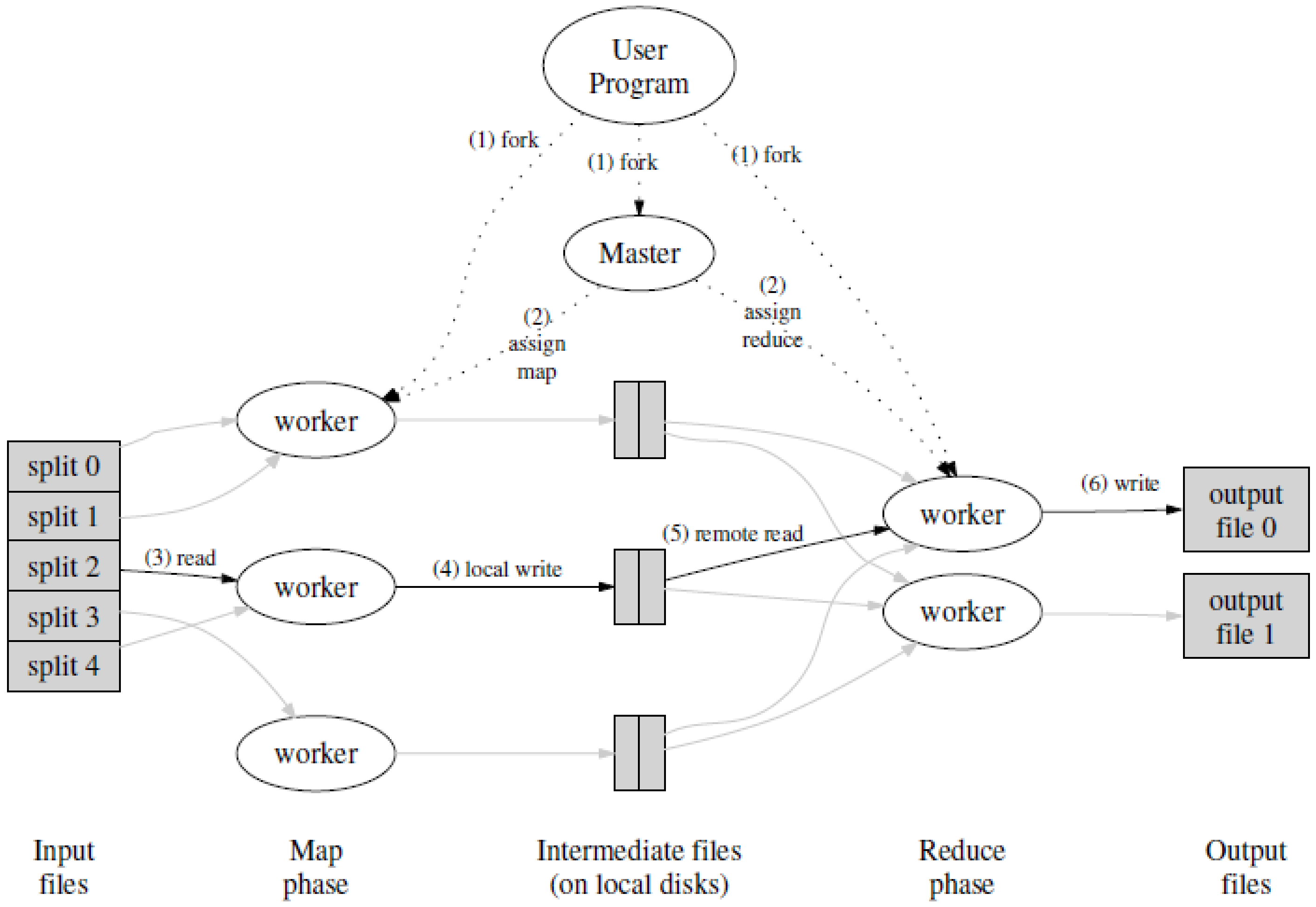
## Distributed/Parallel Computing for Cloud Computing

# MapReduce

- Hadoop Map/Reduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.


MAP
MAP
MAP
MAP
REDUCE
REDUCE
REDUCE
REDUCE
...
...
Unordered, randomly binned data
Data binned by key and optionally ordered within bins
Unordered, randomly binned data

# Map Reduce


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

# Input and Output types of a Map/Reduce job

- (input) <k1, v1>
- -> **map** -> <k2, v2>
- -> **combine** -> <k2, v2>
- -> **reduce** -> <k3, v3> (output)

## Map Reduce Operations (1)

- The MapReduce library in the user program first shards the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.
- One of the copies of the program is special: the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.

## Map Reduce Operations (2)

- A worker who is assigned a map task reads the contents of the corresponding input shard. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.
- Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

## Map Reduce Operations (3)

- When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. If the amount of intermediate data is too large to fit in memory, an external sort is used.

## Map Reduce Operations (4)

- The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.
- When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

## MapReduce Java Code Example: WordCount v1.0

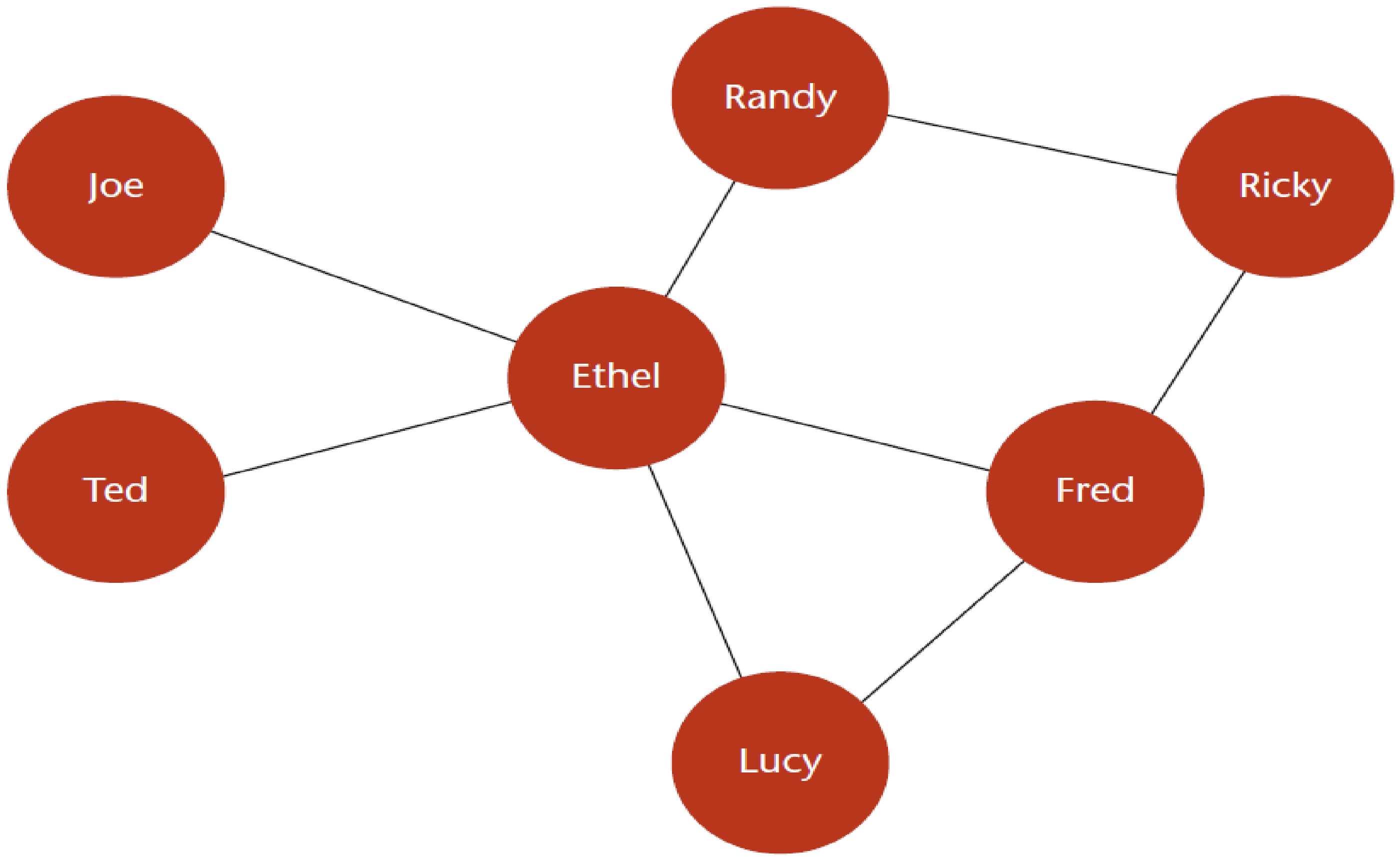
```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

```

```
public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  // set type arguments for generic classes
  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      output.collect(word, one);
    }
    }
  }

  public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }
```

```java

public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);
```

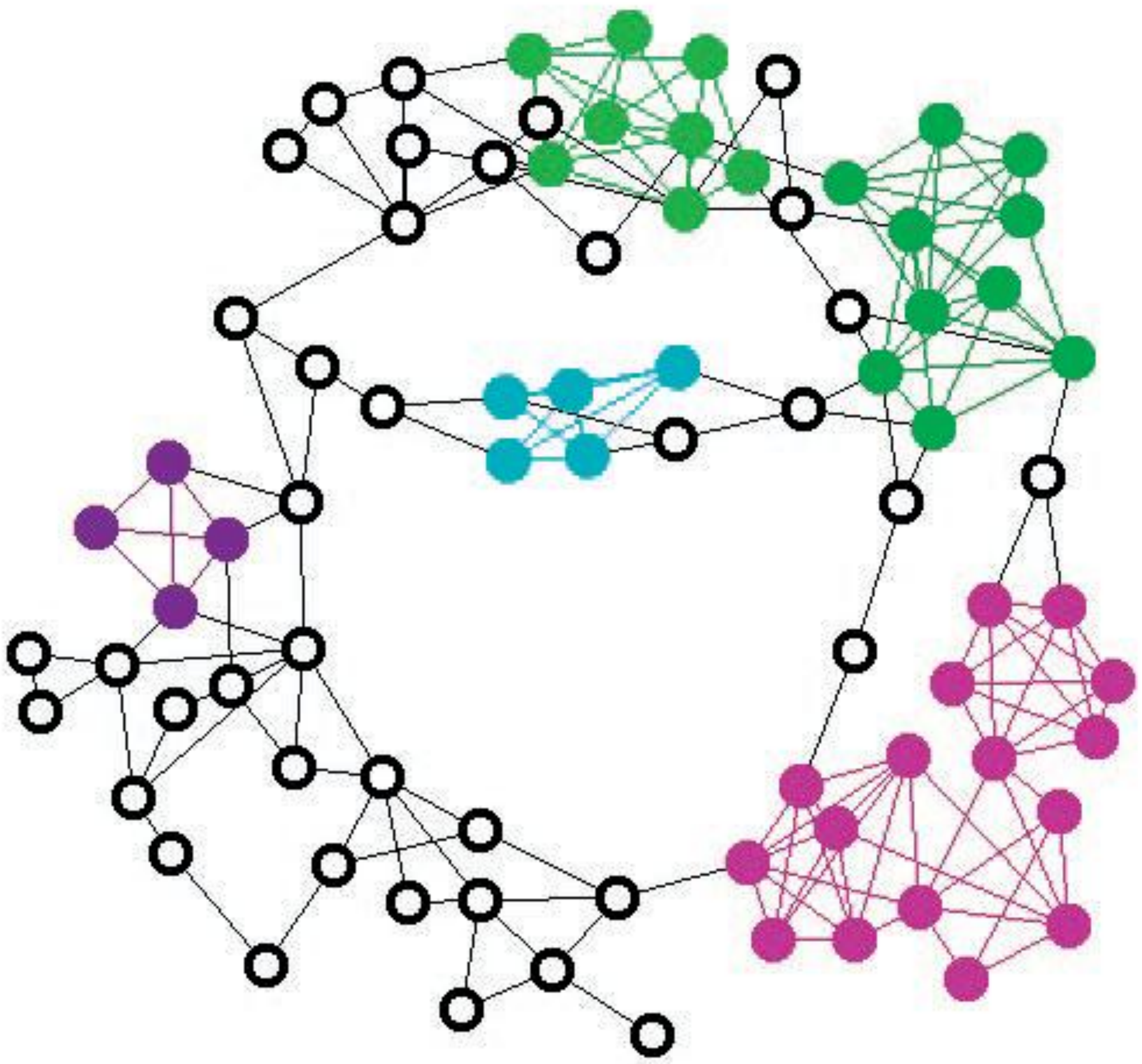
```

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }
}

```

## Modeling Online Social Network



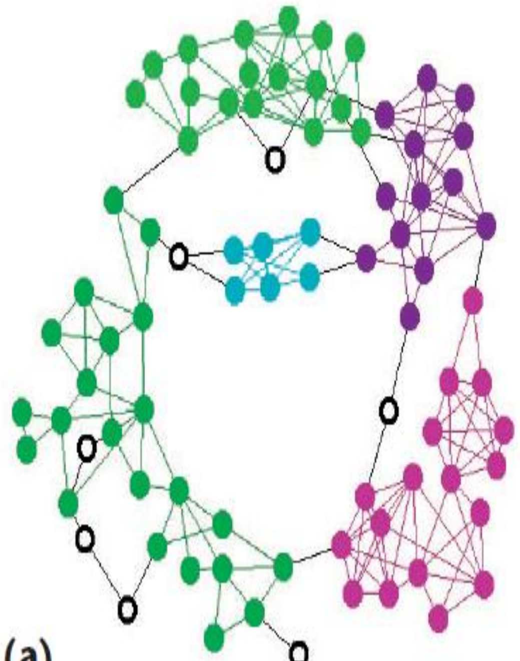
## Why trusses are important?

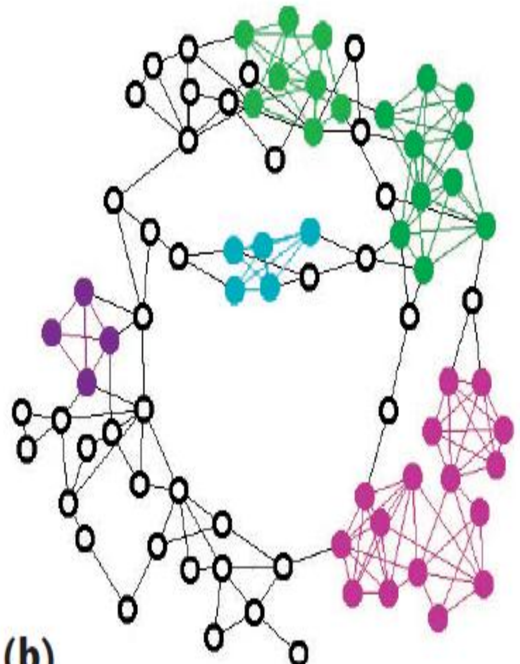
- Trusses are subgraphs of high connectivity, suitable for recognizing clusters of tight interaction in social networks.
- They’re a relaxation of cliques and capture cliques’ intent without their many shortcomings.
  - Cliques are computationally intractable intractable.
  - Cliques are unlikely to be found in naturally occurring graphs, particularly when only sampled information is available, and intersect in ways that make their interpretation difficult.

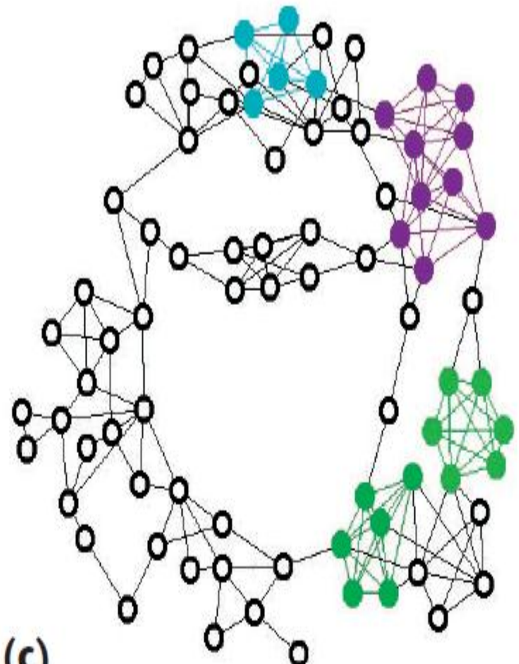


## Definition of Trusses

- Specifically, a *k-truss is a relaxation of a k-member* clique and is a nontrivial, single-component maximal subgraph, such that every edge is contained in at least *k - 2 triangles in the subgraph.*
- The use of “nontrivial” here is meant to exclude a subgraph that consists only of a single vertex.
- A clique of *k vertices is an example of a k-truss.*


(a)


(b)

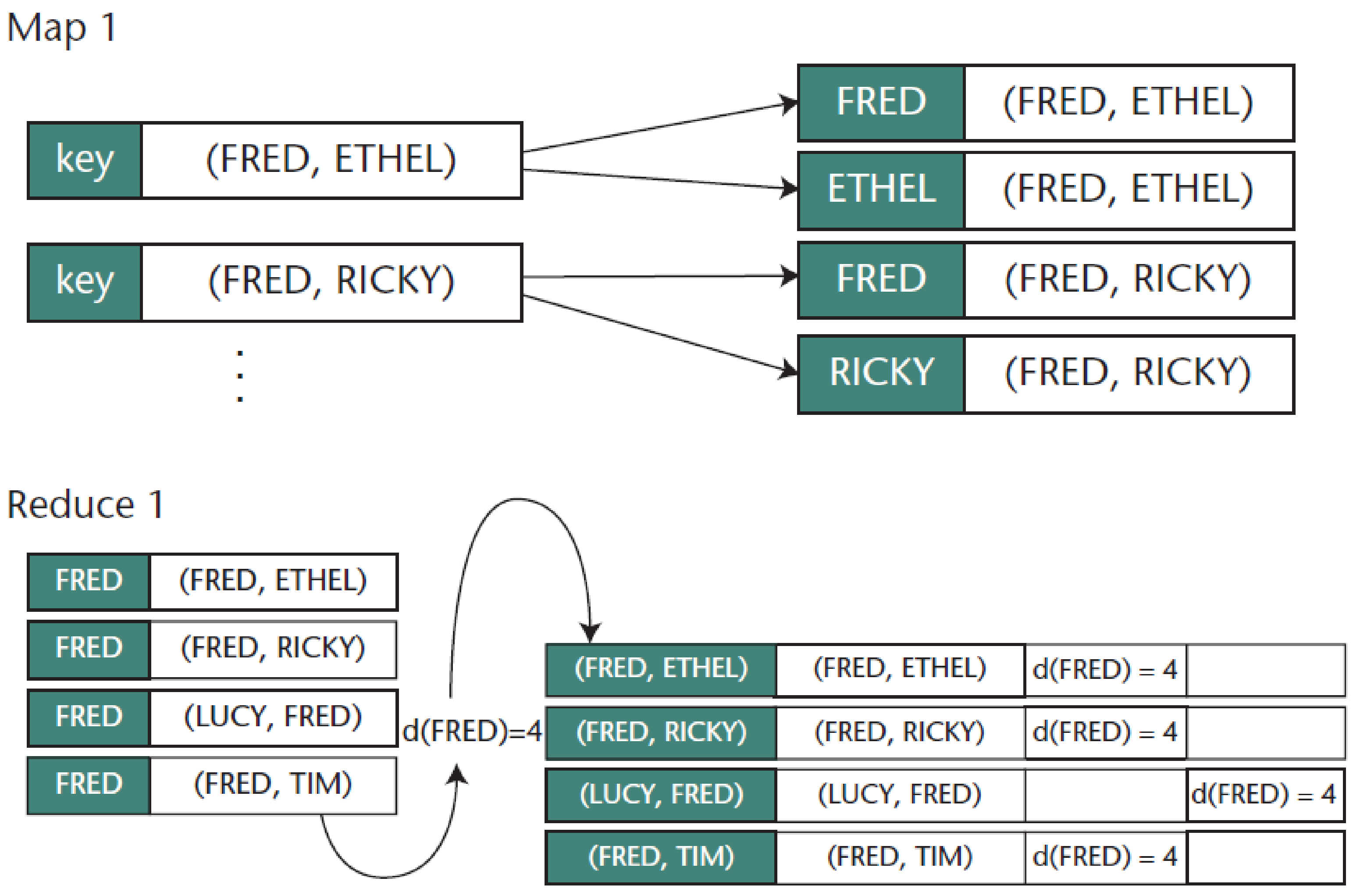

(c)

## Challenges of Designing Cloud-Scale Graph Algorithms

- Some graph-theoretic problems are simple when the graph is small
- They become non-trivial when the graph representation takes more than 4GB RAM
- Recall that hard disks are too slow for large-scale computing

## Vertex Degree Counting

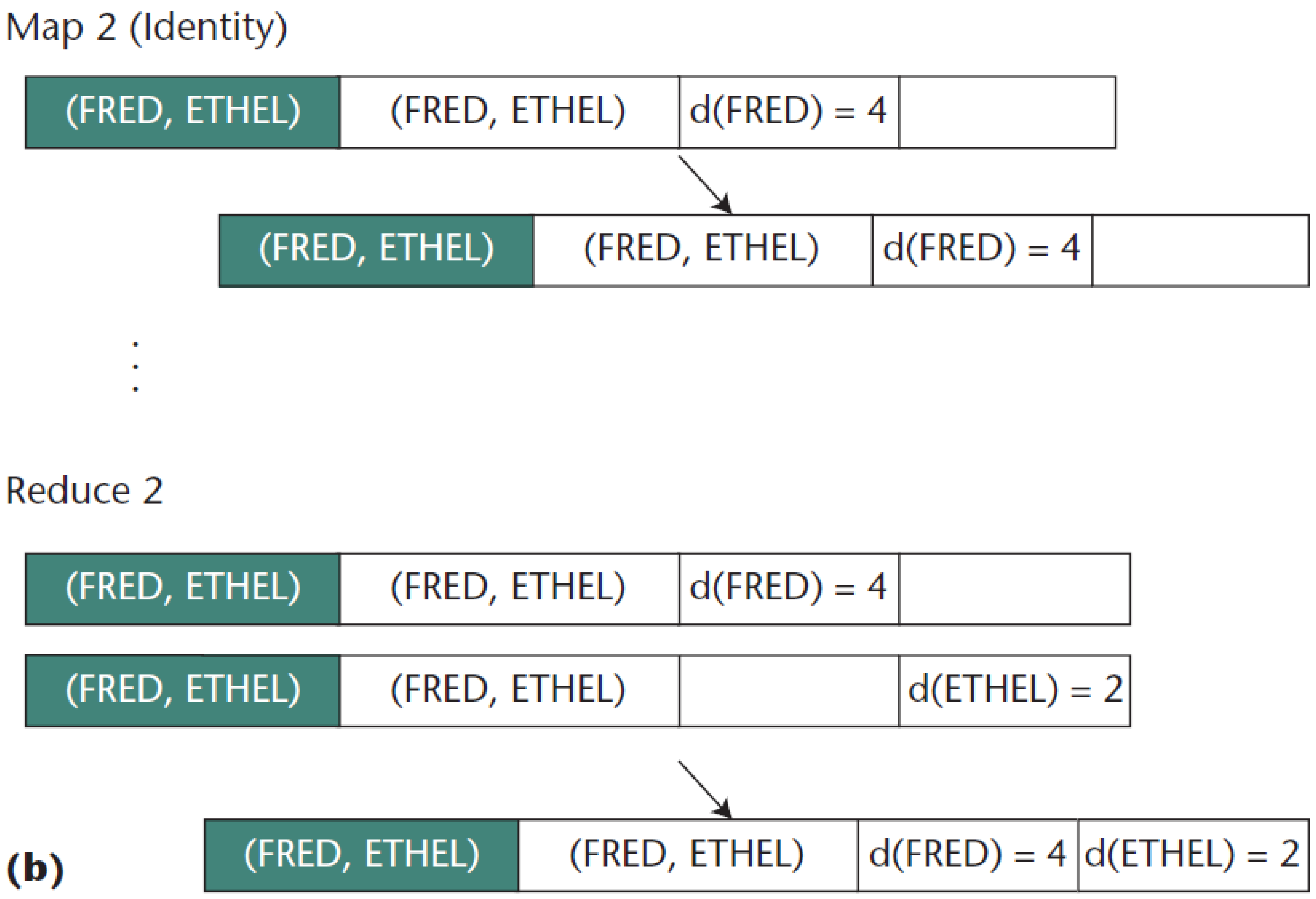
- Composed of Two MapReduce jobs
- In the first MapReduce job, for each input record, the map creates two output records, one keyed under each vertex in the edge.
- The reduce takes all edges mapped to a single vertex, counts them to obtain the degree, and emits a record for each input record, each keyed under the edge it represents.


Map 1
key
(FRED, ETHEL)
key
(FRED, RICKY)
FRED
(FRED, ETHEL)
ETHEL
(FRED, ETHEL)
FRED
(FRED, RICKY)
RICKY
(FRED, RICKY)
Reduce 1
FRED
(FRED, ETHEL)
FRED
(FRED, RICKY)
FRED
(LUCY, FRED)
FRED
(FRED, TIM)
d(FRED)=4
(FRED, ETHEL)
(FRED, ETHEL)
d(FRED) = 4
(FRED, RICKY)
(FRED, RICKY)
d(FRED) = 4
(LUCY, FRED)
(LUCY, FRED)
d(FRED) = 4
(FRED, TIM)
(FRED, TIM)
d(FRED) = 4

**(a)**

## Vertex Counting

- In the second MapReduce job, the identity mapper preserves the records unchanged, so the records are binned by the edges they represent.
- The reducer combines the partial-degree information to produce a complete record, which it exports.


Map 2 (Identity)
(FRED, ETHEL)
(FRED, ETHEL)
d(FRED) = 4
(FRED, ETHEL)
(FRED, ETHEL)
d(FRED) = 4
Reduce 2
(FRED, ETHEL)
(FRED, ETHEL)
d(FRED) = 4
(FRED, ETHEL)
(FRED, ETHEL)
d(ETHEL) = 2
(FRED, ETHEL)
(FRED, ETHEL)
d(FRED) = 4
d(ETHEL) = 2

(b)

## Open Triads and Triangles

- Enumerating triangles is essentially a two-step approach: enumerate open triads (pairs of edges of the form {(A, B), (B, C)}) and recognize when an edge closes those triads to form triangles.

## An Algorithm for Finding Trusses

1. Augment the edges with vertex valences.
2. Enumerate triangles.
3. For each edge, record the number of triangles containing that edge.
4. Keep only the edges with sufficient support.
5. If step 4 dropped any edges, return to step 1.
6. Find the remaining graph's components; each is a truss.

## Reference

- Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. The Sixth Symposium on Operating System Design and Implementation*, 2004.
- J. Lin and C. Dyer, *Data-Intensive Text Processing with MapReduce*. 2010, Morgan and Claypool Publishers.
- http://hadoop.apache.org/mapreduce/
- http://www.windowsazure.com/zh-tw/

## Reference

- Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. ACM SIGCOMM 2008*.
- Jonathan Cohen, "Graph Twiddling in a MapReduce World," IEEE Computing in Science and Engineering, pp. 29-41, July 2009.
- Rung-Hung Gau, Tzu-Chiang Hsieh, Sheng-Wen Tsai, and Ching-Pei Cheng, "An Implementation Framework of MapReduce Email Social Network Analysis," in *Proc. The 7th ACM Workshop on Wireless Multimedia Networking and Computing*, pp. 67-69, October 2011.