



Lecture 12

DATA ANALYTICS ON WEB SCALE

The data deluge



Source: The Economist,
February 25, 2010

The Data Deluge

“

EIGHTEEN months ago, Li & Fung, a firm that manages supply chains for retailers, saw 100 gigabytes of information flow through its network each day. Now the amount has increased tenfold. During 2009, American drone aircraft flying over Iraq and Afghanistan sent back around 24 years' worth of video footage. New models being deployed this year will produce ten times as many data streams as their predecessors, and those in 2011 will produce 30 times as many.

Source: The Economist,
February 25, 2010

The Data Deluge

“

Everywhere you look, the quantity of information in the world is soaring. According to one estimate, mankind created 150 exabytes (billion gigabytes) of data in 2005. This year, it will create 1,200 exabytes. Merely keeping up with this flood, and storing the bits that might be useful, is difficult enough. Analysing it, to spot patterns and extract useful information, is harder still.

1 gigabyte = 10^9 bytes

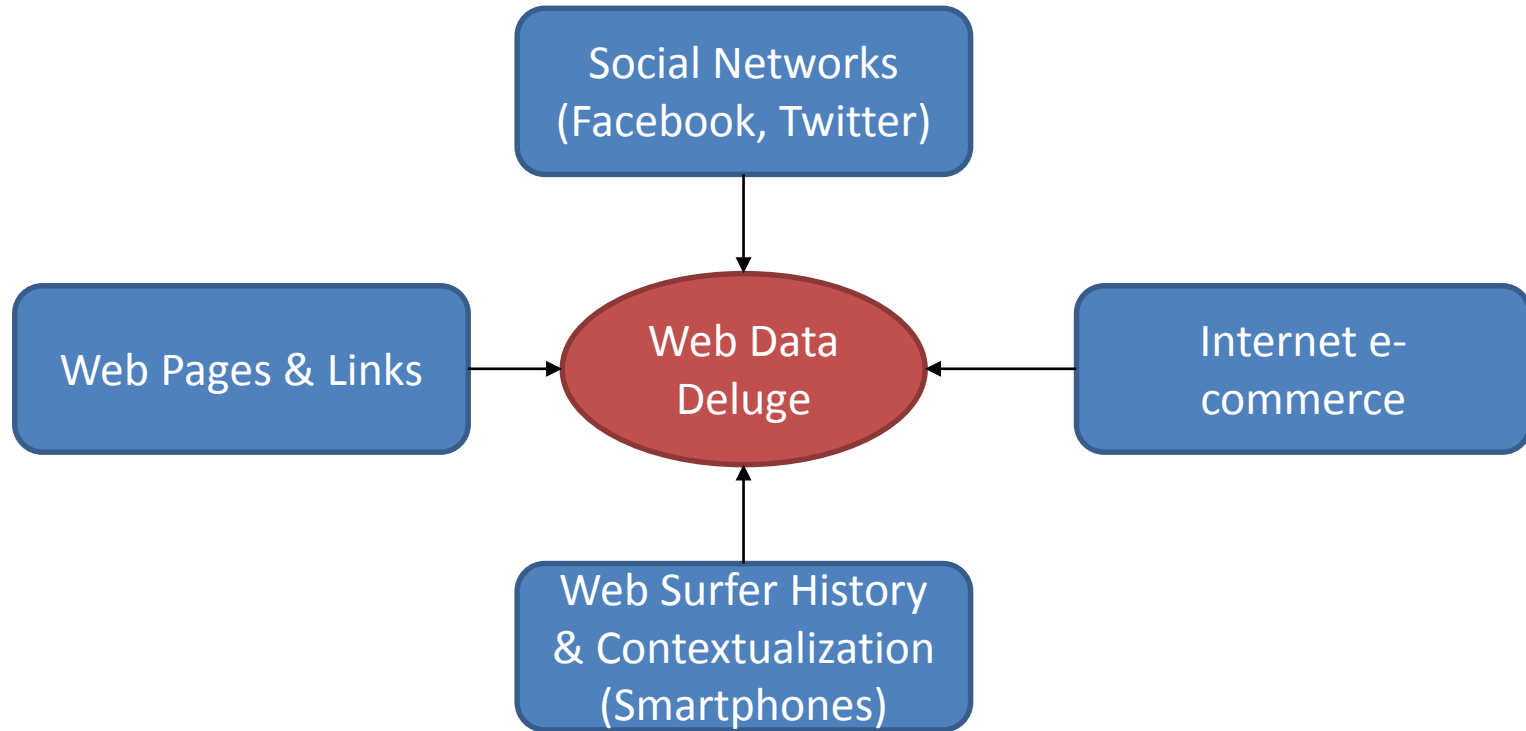
1 terabyte = 10^{12} bytes

1 petabyte = 10^{15} bytes

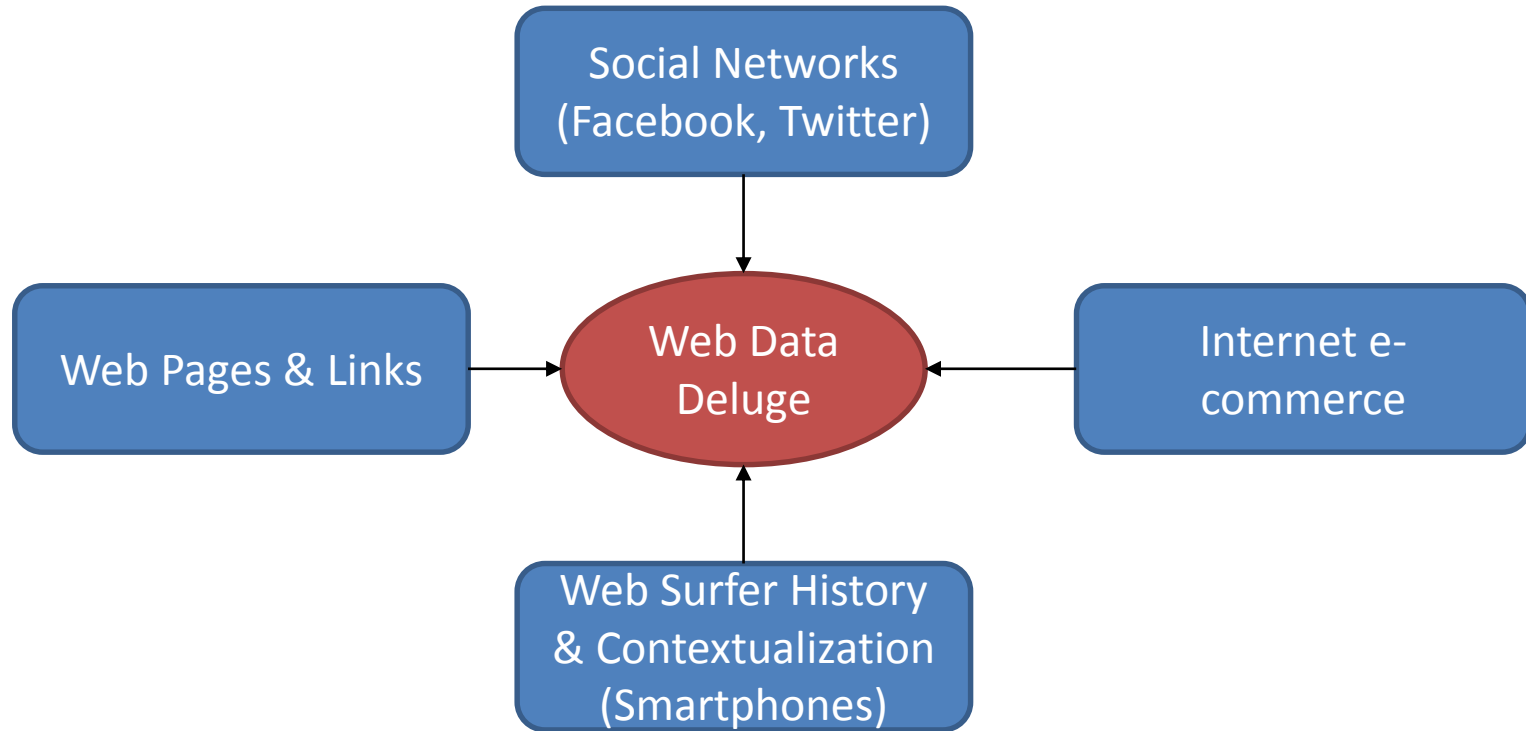
1 exabyte = 10^{18} bytes

Source: The Economist,
February 25, 2010

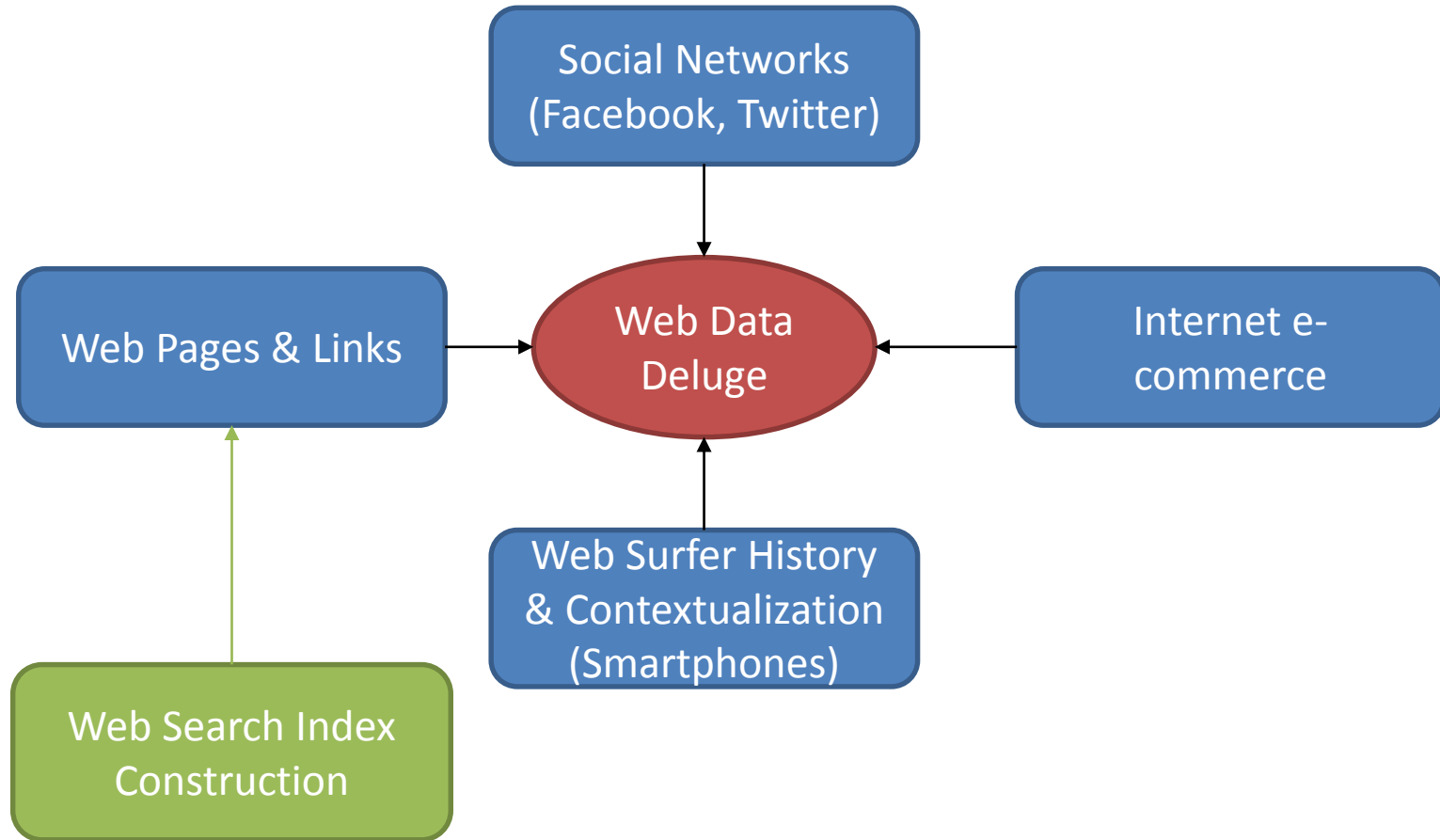
The Data Deluge on the Web



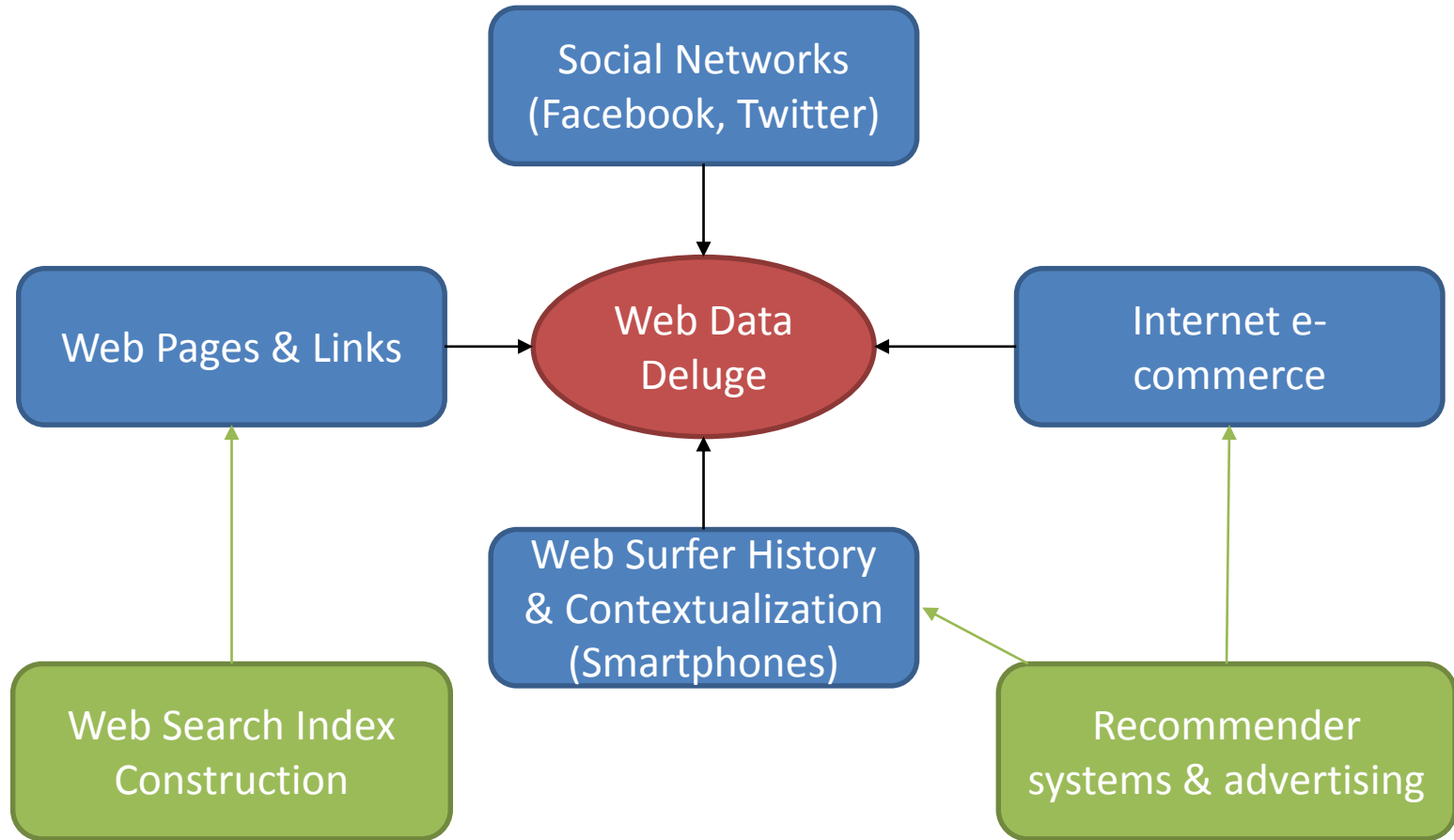
Why analyze this data?



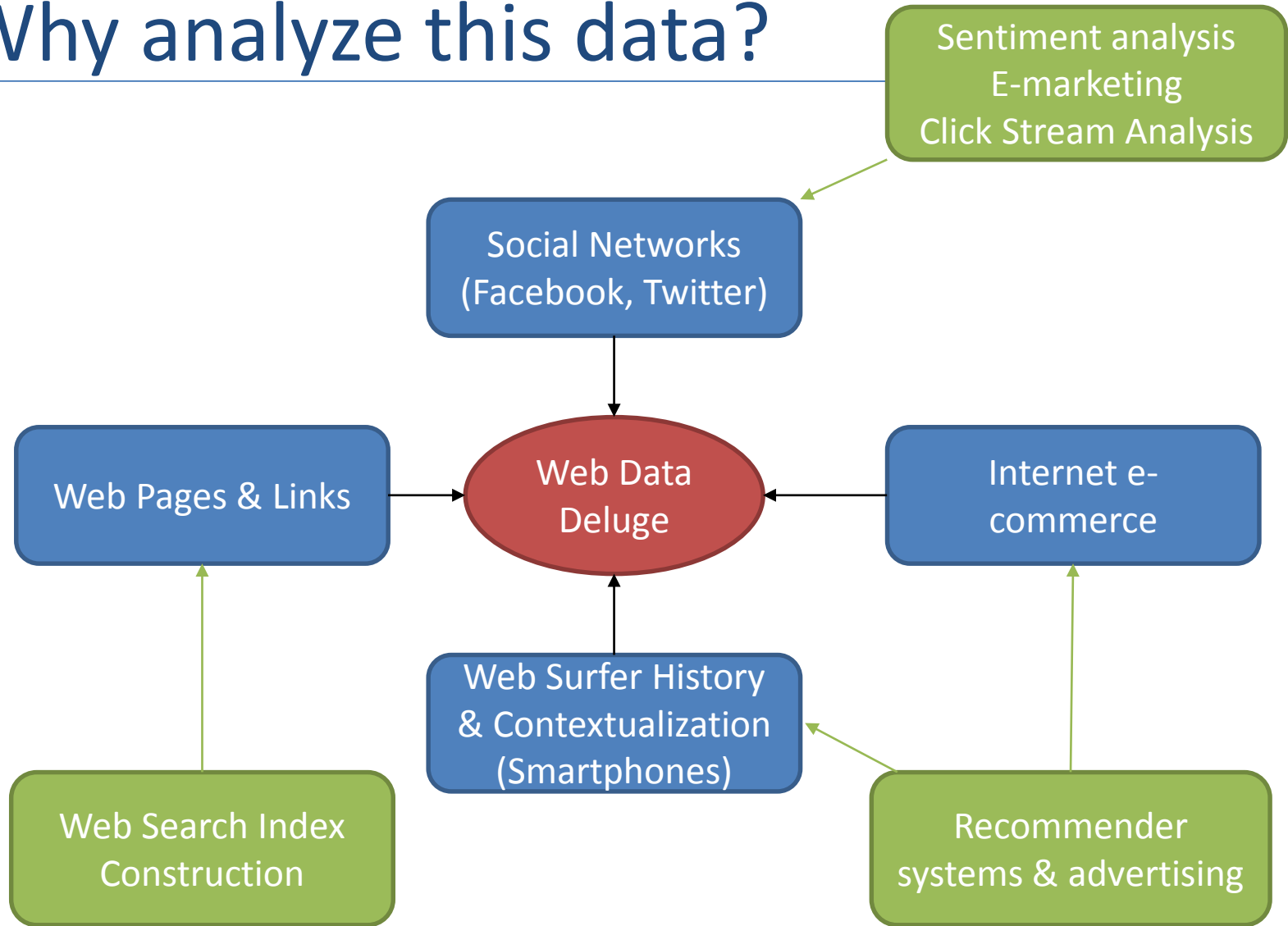
Why analyze this data?



Why analyze this data?



Why analyze this data?



Is this really problematic?

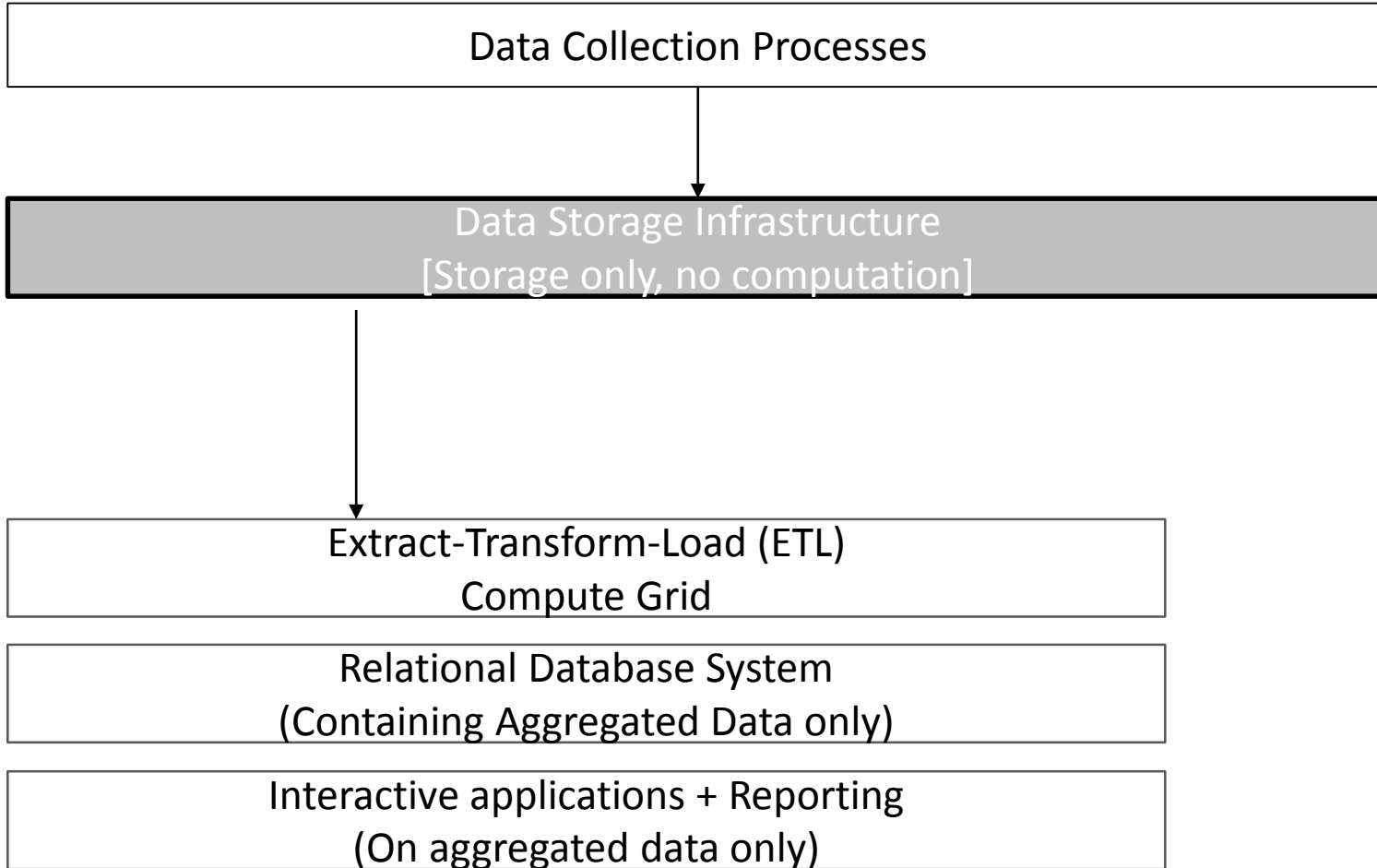
“

*Although it is hard to accurately determine the size of the Web at any point in time, it is safe to say that it consists of hundreds of billions of individual documents and that it continues to grow. If we assume the Web to contain 100 billion documents, with an average document size of 4 kB (after compression), the **Web is about 400 TB** ...*

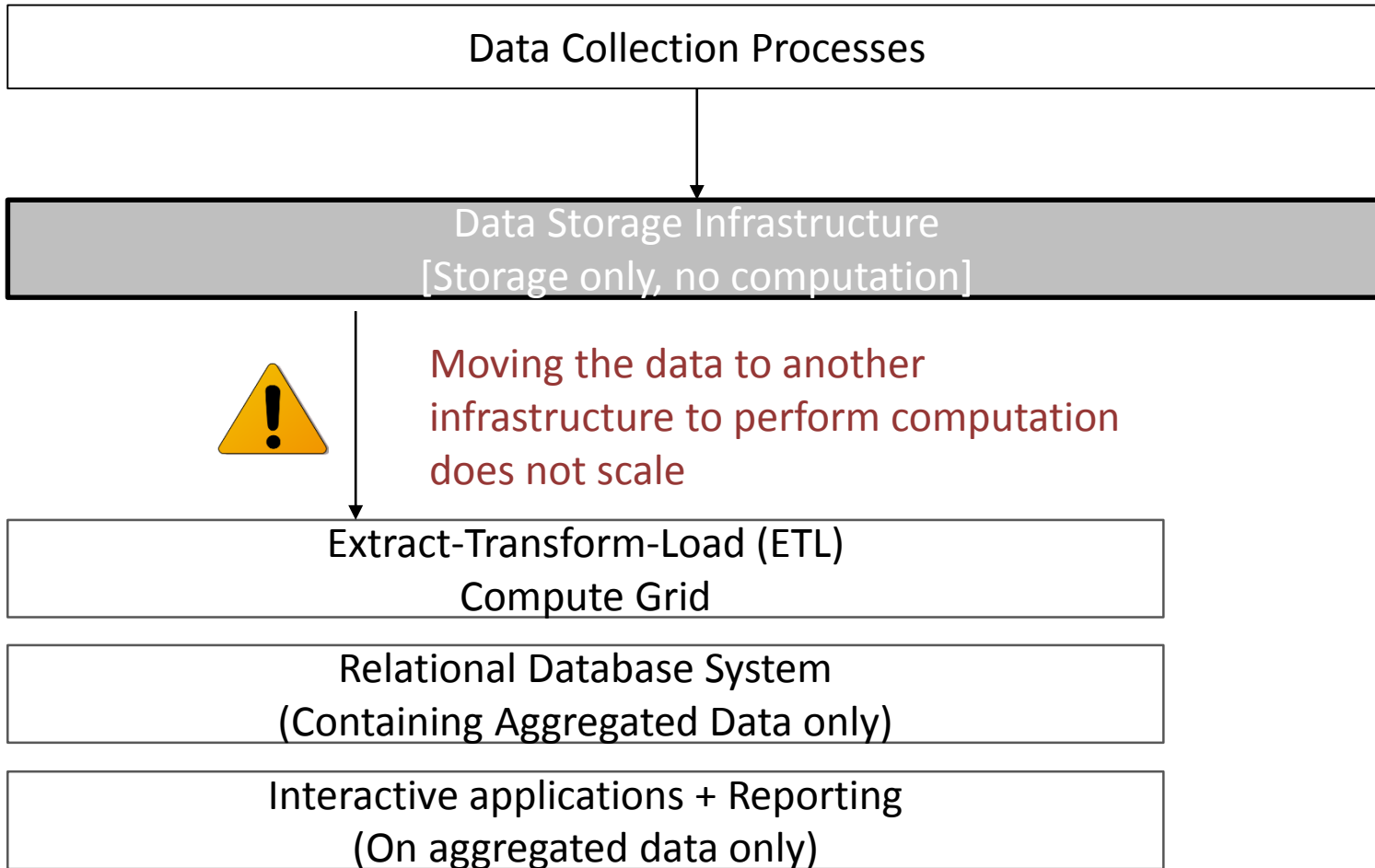
*... The size of **the resulting inverted (web search) index** depends on the specific implementation, but it **tends to be on the same order of magnitude** as the original repository.*

Source: The Datacenter as a Computer
Luiz André Barroso (Google)
Jimmy Clidaras (Google)
Urs Hölzle (Google)

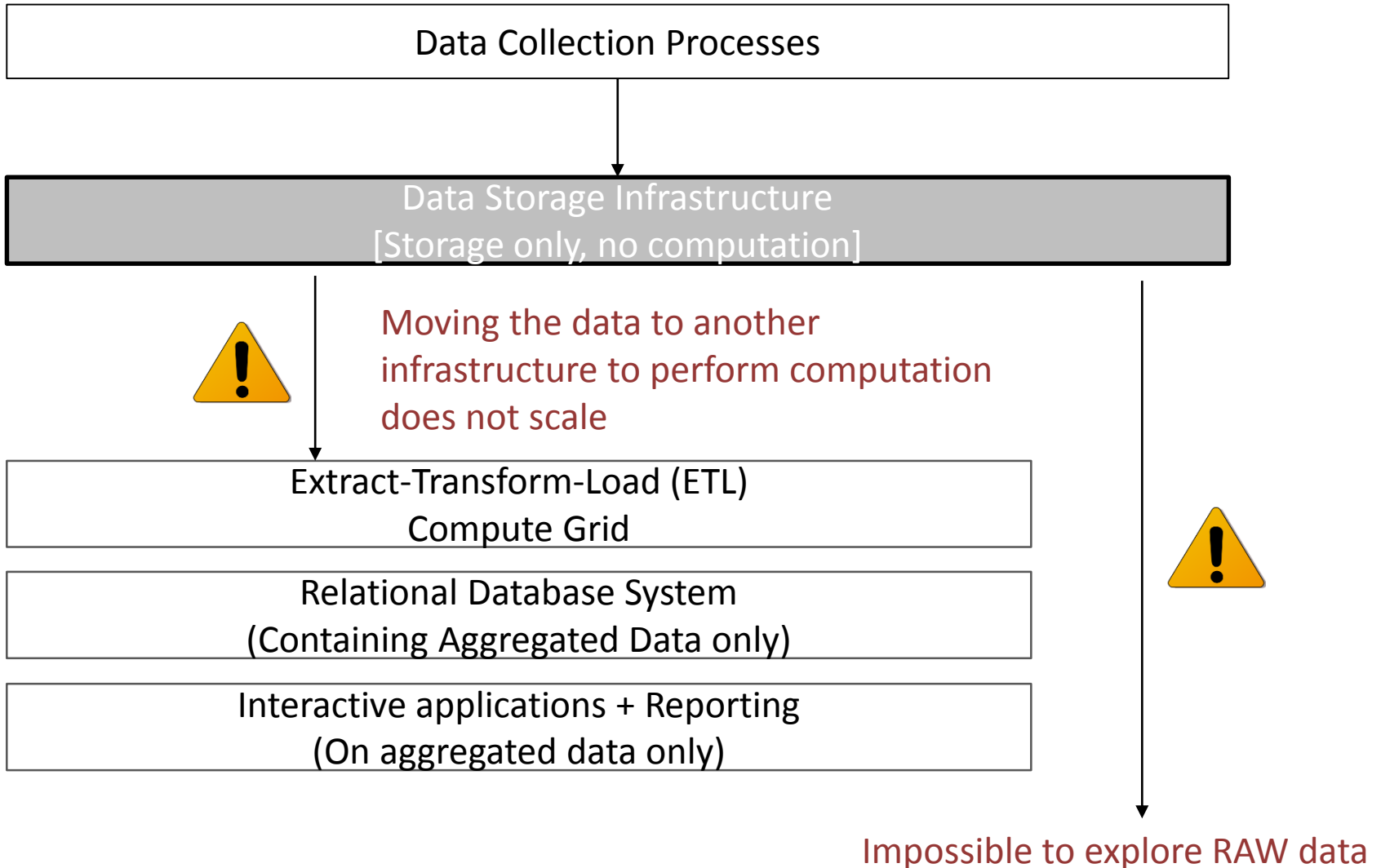
Limitations of Traditional Data Analytics Architecture



Limitations of Traditional Data Analytics Architecture



Limitations of Traditional Data Analytics Architecture



Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Name	Purpose
Google File System	A distributed file system for scalable storage and high-throughput retrieval
Map Reduce	A programming model + execution environment for general-purpose distributed batch processing
Pregel	A programming model + execution environment for analysis of graphs
Dremel	A query language for interactive SQL-like analysis of structured datasets

Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Name	Purpose
Google File System	A distributed file system for scalable storage and high-throughput retrieval
Map Reduce	A programming model + execution environment for general-purpose distributed batch processing
Pregel	A programming model + execution environment for analysis of graphs
Dremel	A query language for interactive SQL-like analysis of structured datasets

High-level design described in a series of papers; no implementation available

Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Name	Purpose
Google File System	A distributed file system for scalable storage and high-throughput retrieval
Map Reduce	A programming model + execution environment for general-purpose distributed batch processing
Pregel	A programming model + execution environment for analysis of graphs
Dremel	A query language for interactive SQL-like analysis of structured datasets

Open Source Impl



Apache Hadoop

- HDFS
- M/R

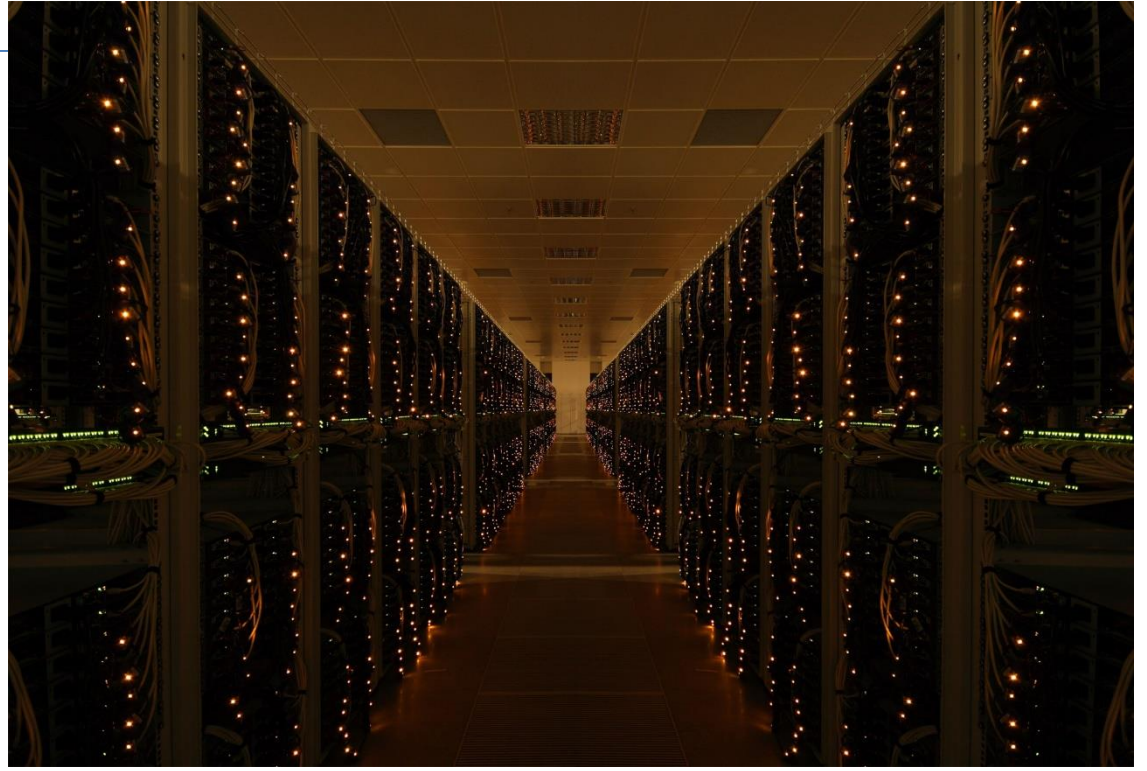
Apache Giraph

Apache Drill

High-level design described in a series of papers; no implementation available

Rest of this lecture

1. Execution hardware
2. HDFS: Hadoop Distributed File System
3. Map/Reduce



WAREHOUSE SCALE MACHINES

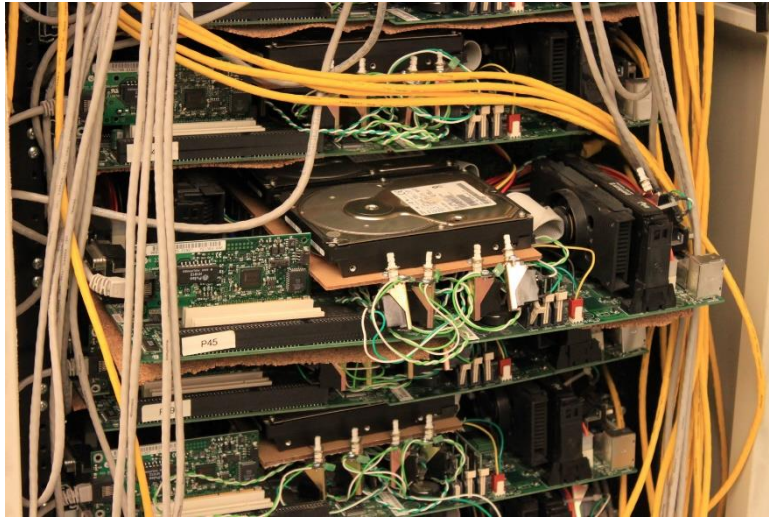
Execution environment - 1997

google.stanford.edu (circa 1997)

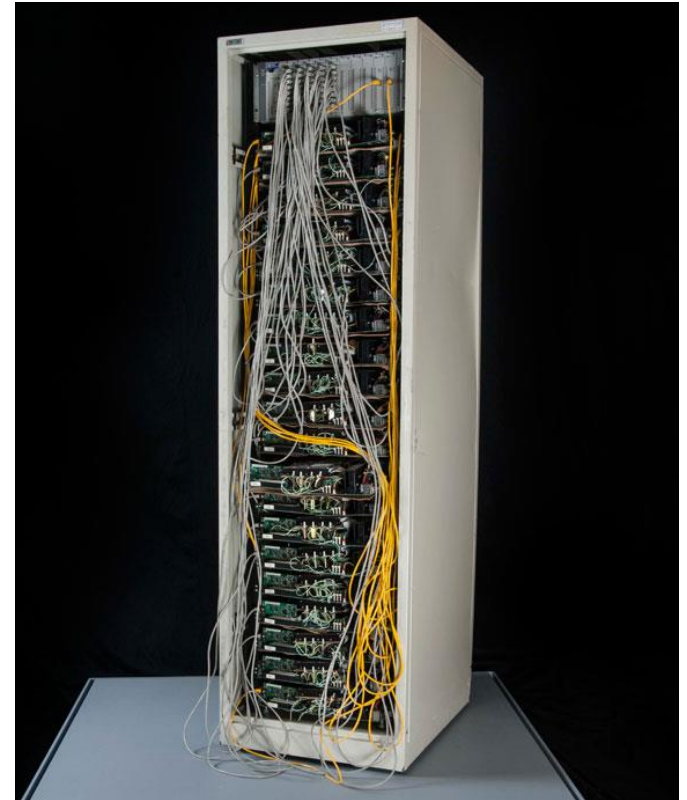


- Just a bunch of computers.
- The web index fit on a single computer
- Redundancy to ensure availability

Execution environment - 1999



- **Compute servers** consist of multiple CPUs (possibly with multiple cores per CPU), and attached hard disks,
- Servers are collected in **racks**. High-speed Ethernet connections (at least 1Gbps) connect servers in a rack together



The Google “Corckboard”
rack

Execution environment - currently



An image of the google datacenter in Mons (Belgium)

- Racks are connected together to central network switches using multi-Gbps redundant links.
- Access of data from other racks is slower than data on same computer/same rack
- Many racks together form a **data center**

Discussion

- Each compute node is kept simple by design:
 - Mid-range computers are much cheaper than powerful high-range computers ...
 - ... and consume less energy
 - ... but google has lots of them!

Characteristics

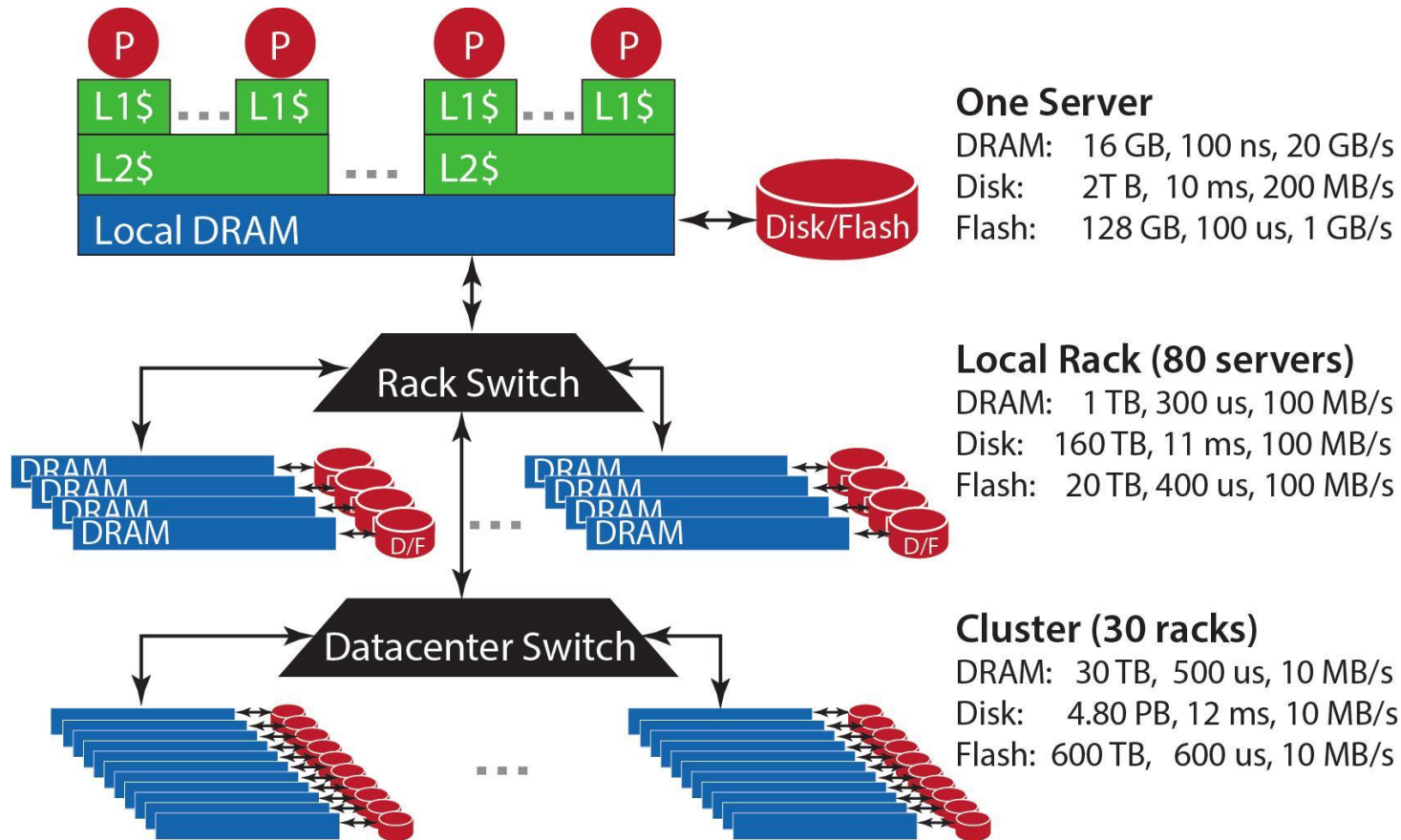


Figure Source: The Datacenter as a Computer

Characteristics

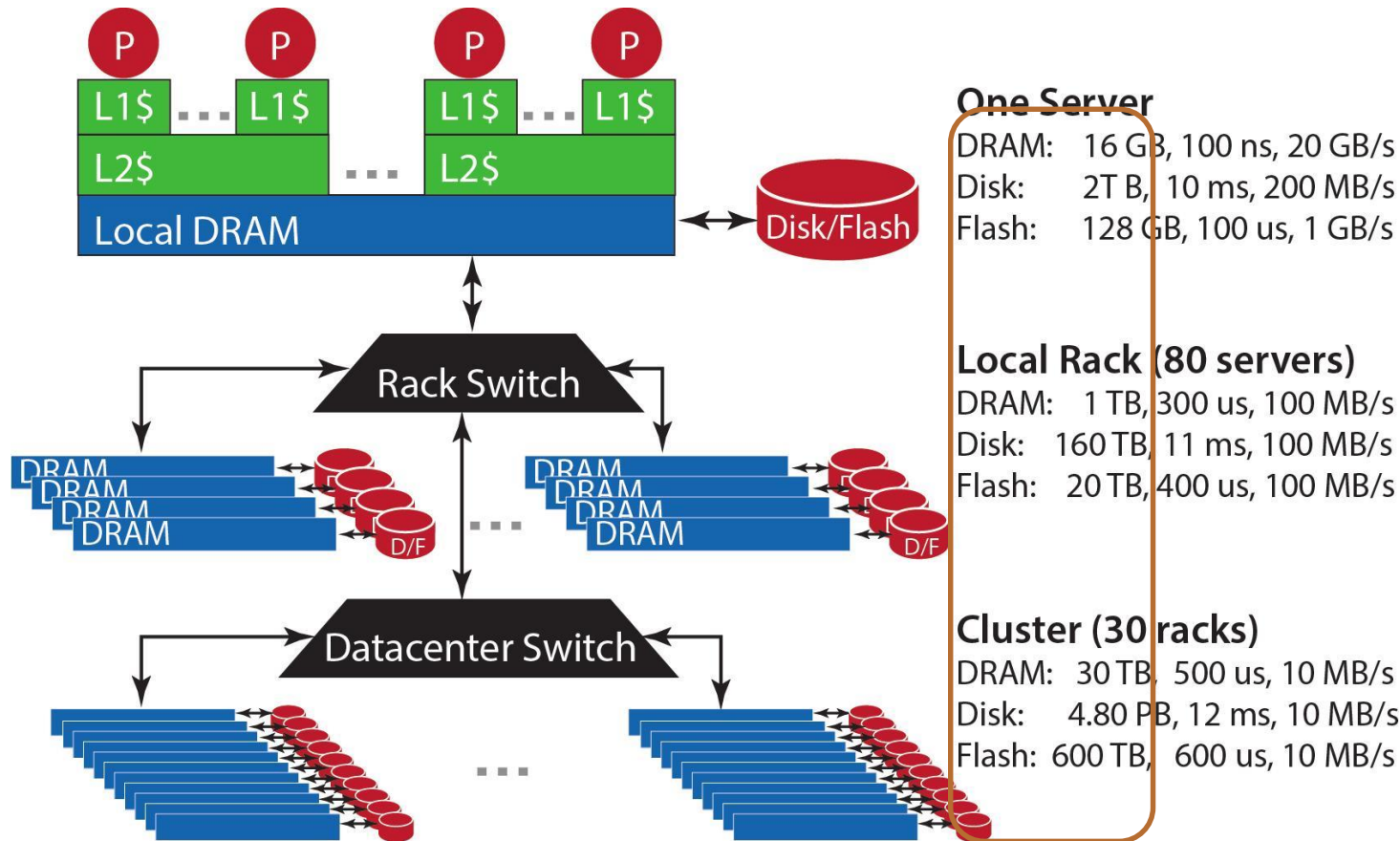


Figure Source: The Datacenter as a Computer

Capacity: The amount of data we can store per server/rack/datacenter

Characteristics

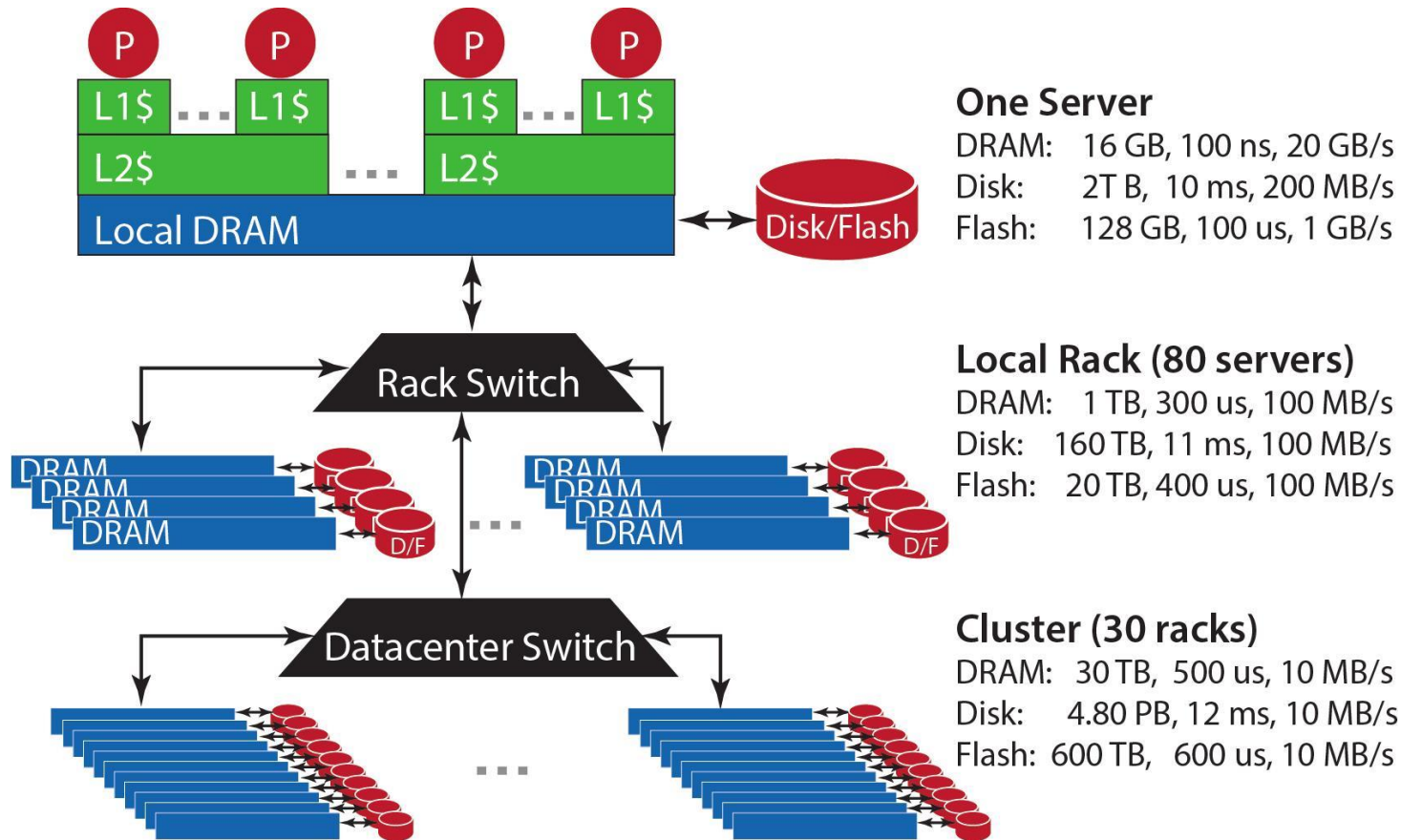


Figure Source: The Datacenter as a Computer

Characteristics

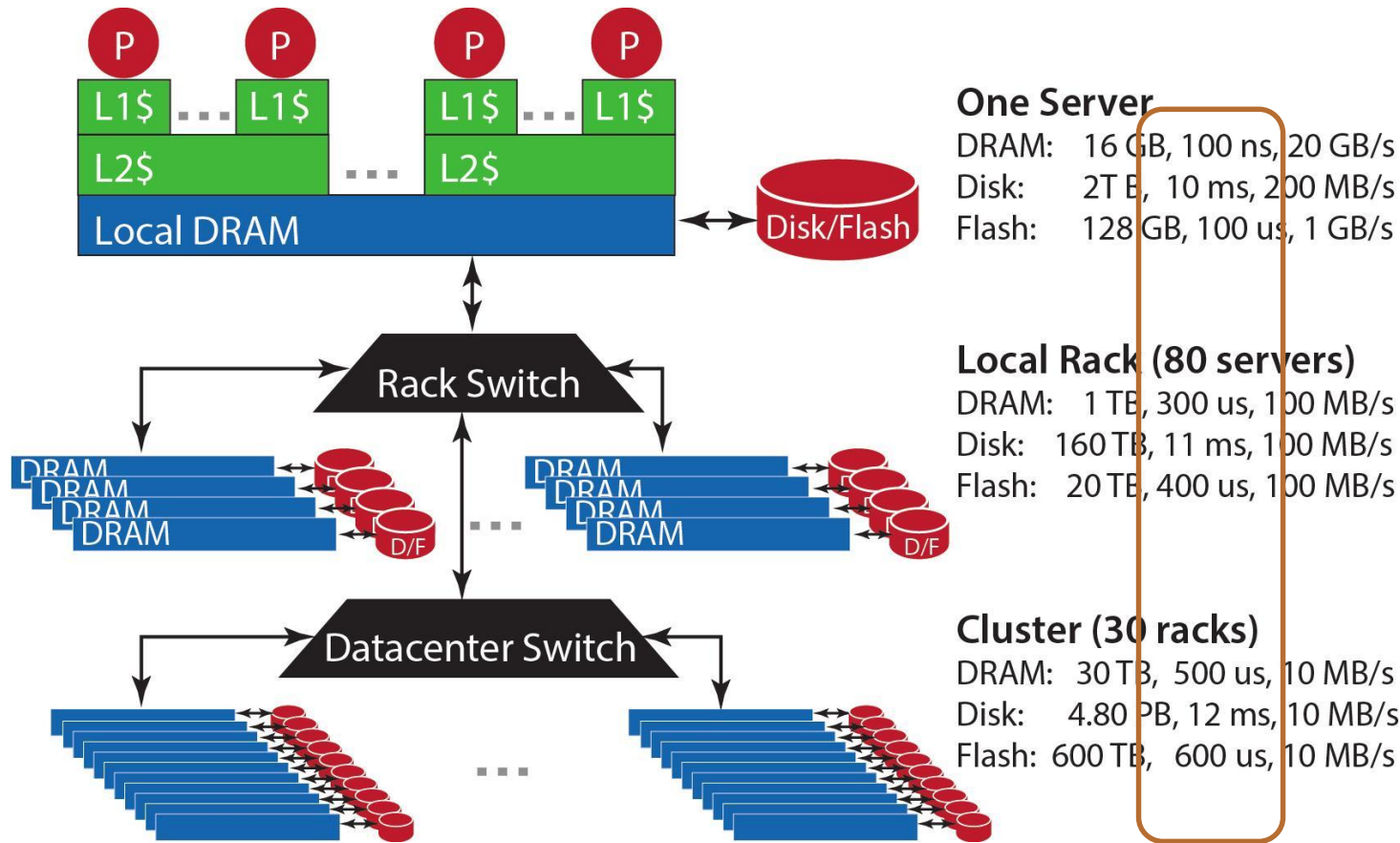


Figure Source: The Datacenter as a Computer

Latency: The time it takes to fetch a data item, when asked on local machine/another server on the same rack/another server on a different rack

Characteristics

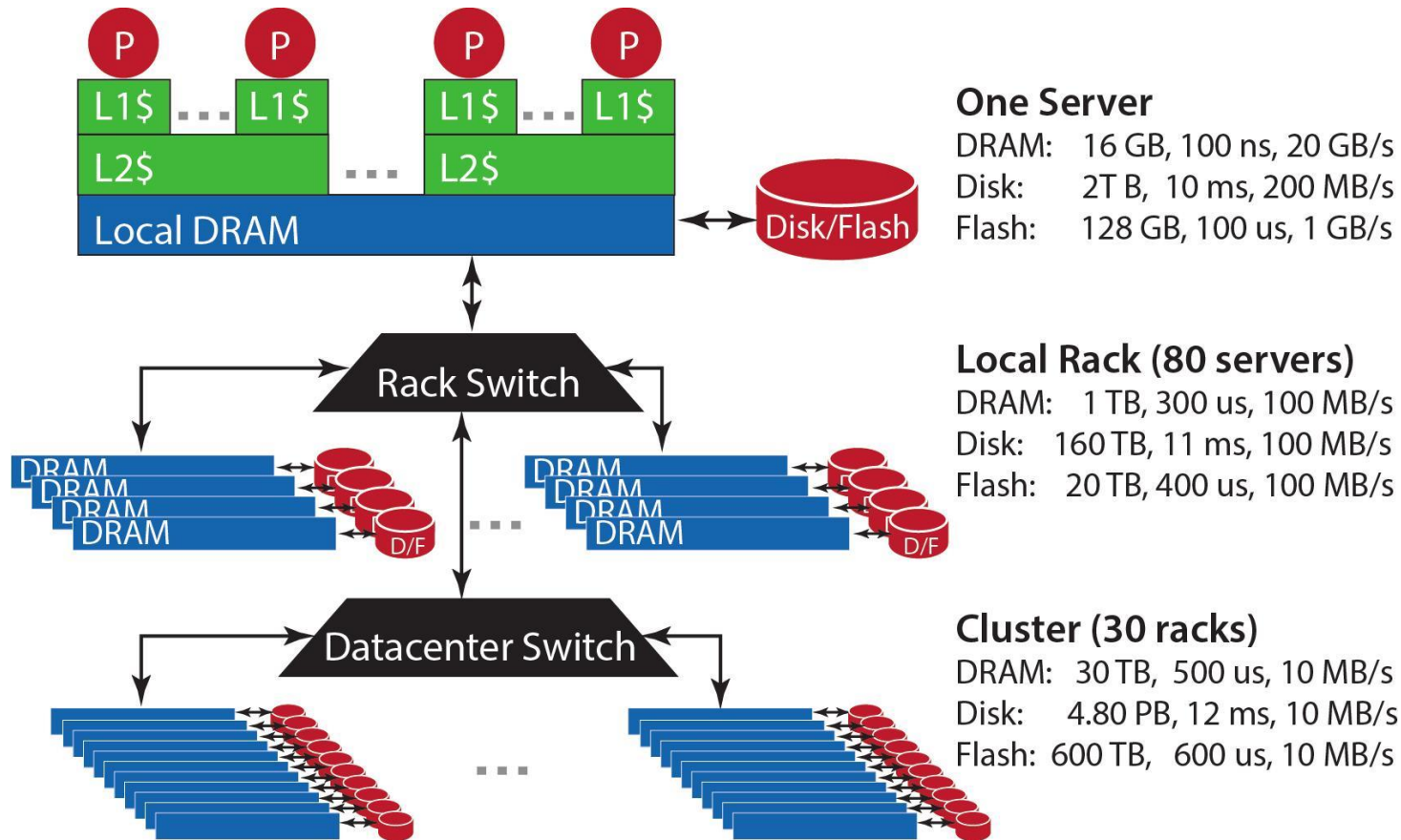


Figure Source: The Datacenter as a Computer

Characteristics

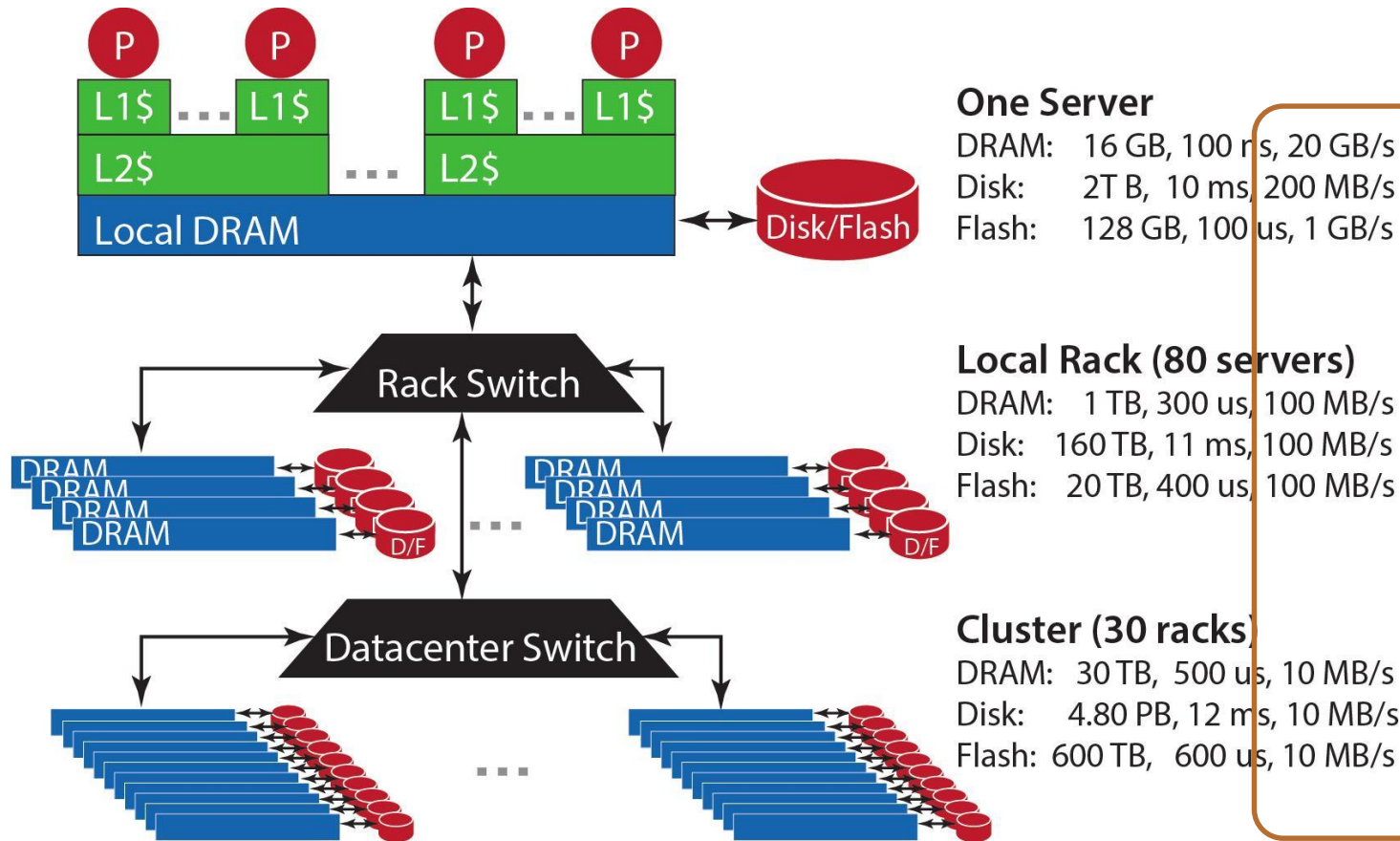


Figure Source: The Datacenter as a Computer

Bandwidth: the speed at which data can be transferred to the same machine/another server on the same rack/another server on a different rack

Characteristics

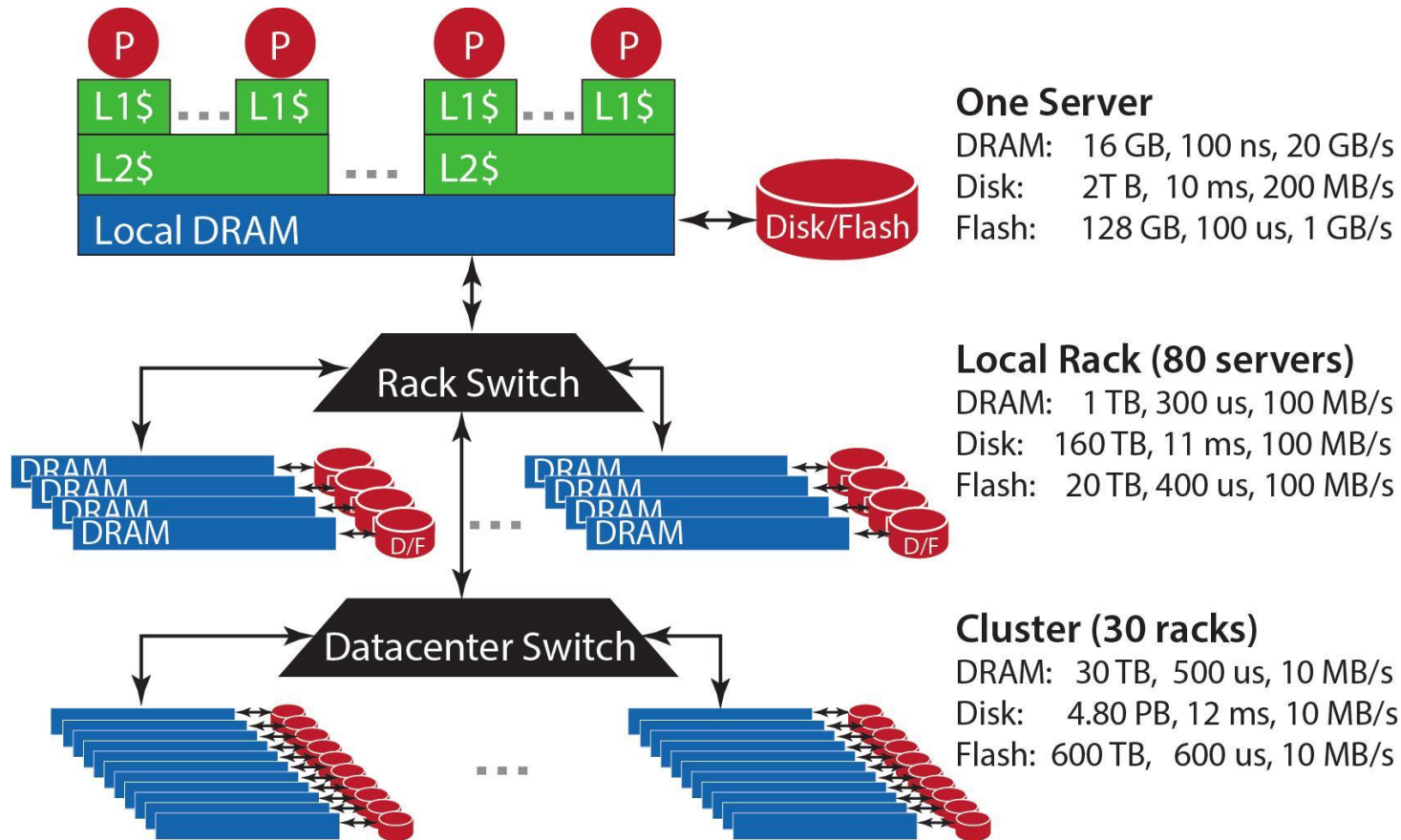


Figure Source: The Datacenter as a Computer

Characteristics

Conclusion:

- Huge storage capacity
- Latency between racks
 - = 1/10 latency on rack level
 - ≈ 1/10 latency on server level
- Bandwidth between racks
 - = 1/10 bandwidth on rack level
 - = ½ to 1/10 bandwidth on server level

One Server

DRAM: 16 GB, 100 ns, 20 GB/s

Disk: 2T B, 10 ms, 200 MB/s

Flash: 128 GB, 100 us, 1 GB/s

Local Rack (80 servers)

DRAM: 1 TB, 300 us, 100 MB/s

Disk: 160 TB, 11 ms, 100 MB/s

Flash: 20 TB, 400 us, 100 MB/s

Cluster (30 racks)

DRAM: 30 TB, 500 us, 10 MB/s

Disk: 4.80 PB, 12 ms, 10 MB/s

Flash: 600 TB, 600 us, 10 MB/s

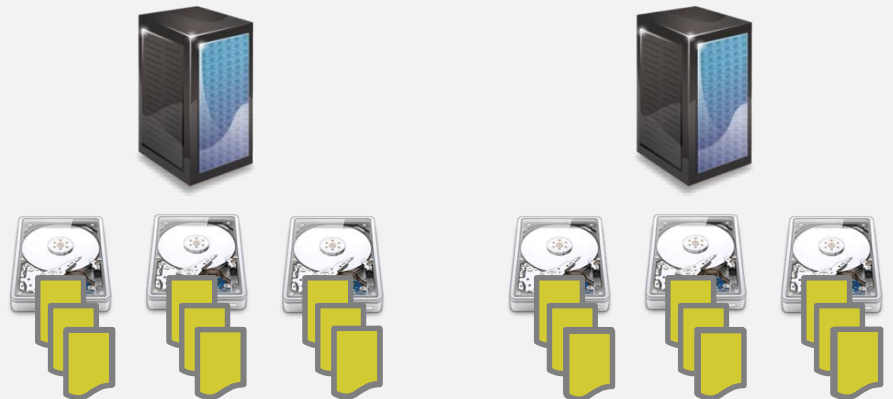


What do we gain? Parallelism!

- Let us consider the **maximal aggregate bandwidth**: the speed by which we can analyze data in parallel assuming ideal data distribution over servers & disks

What do we gain? Parallelism!

- Let us consider the **maximal aggregate bandwidth**: the speed by which we can analyze data in parallel assuming ideal data distribution over servers & disks
- “Embarassingly parallel” example: count the number of times the word “Belgium” appears in documents on the Web.



What do we gain? Parallelism!

- Let us consider the **maximal aggregate bandwidth**: the speed by which we can analyze data in parallel assuming ideal data distribution over servers & disks
- “Embarassingly parallel” example: count the number of times the word “Belgium” appears in documents on the Web.

- Each server has multiple CPUs and can read from multiple disks in parallel. As such, each server can analyze many documents in parallel.



What do we gain? Parallelism!

- Let us consider the **maximal aggregate bandwidth**: the speed by which we can analyze data in parallel assuming ideal data distribution over servers & disks
- “Embarassingly parallel” example: count the number of times the word “Belgium” appears in documents on the Web.

- Each server has multiple CPUs and can read from multiple disks in parallel. As such, each server can analyze many documents in parallel.



What do we gain? Parallelism!

- Let us consider the **maximal aggregate bandwidth**: the speed by which we can analyze data in parallel assuming ideal data distribution over servers & disks
- “Embarassingly parallel” example: count the number of times the word “Belgium” appears in documents on the Web.

- Each server has multiple CPUs and can read from multiple disks in parallel. As such, each server can analyze many documents in parallel.
- At the end, sum the per-server counters (which can be done very fast)



What do we gain? Parallelism!

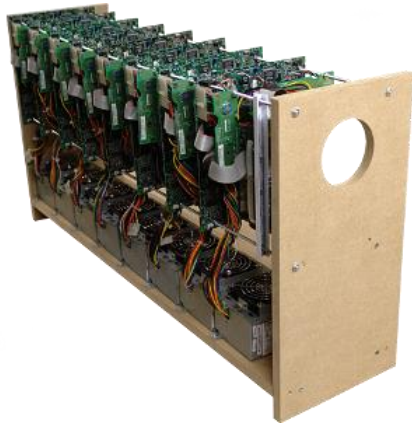
- Let us consider the **maximal aggregate bandwidth**: the speed by which we can analyze data in parallel assuming ideal data distribution over servers & disks

Component		Max Aggr Bandwidth
1 Hard Disk		100 MB/sec (\approx 1 Gbps)
Server	= 12 Hard Disks	1.2 GB/sec (\approx 12 Gbps)
Rack	= 80 servers	96 GB/sec (\approx 768 Gbps)
Cluster/datacenter	= 30 racks	2.88 TB/sec (\approx 23 Tbps)

- Scanning 400TB hence takes 138 secs \approx 2,3 minutes
- Scanning 400TB *sequentially* at 100 MB/sec takes \approx 46,29 days

The challenge

- **Scalable** software development: allow growth without requiring re-architecting algorithm/applications



Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Name	Purpose
Google File System	A distributed file system for scalable storage and high-throughput retrieval
Map Reduce	A programming model + execution environment for general-purpose distributed batch processing
Pregel	A programming model + execution environment for analysis of graphs
Dremel	A query language for interactive SQL-like analysis of structured datasets

Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Name	Purpose
Google File System	A distributed file system for scalable storage and high-throughput retrieval
Map Reduce	A programming model + execution environment for general-purpose distributed batch processing
Pregel	A programming model + execution environment for analysis of graphs
Dremel	A query language for interactive SQL-like analysis of structured datasets

High-level design described in a series of papers; no implementation available

Google's solutions

- New programming models and frameworks for distributed and scalable data analysis

Name	Purpose
Google File System	A distributed file system for scalable storage and high-throughput retrieval
Map Reduce	A programming model + execution environment for general-purpose distributed batch processing
Pregel	A programming model + execution environment for analysis of graphs
Dremel	A query language for interactive SQL-like analysis of structured datasets

Open Source Impl



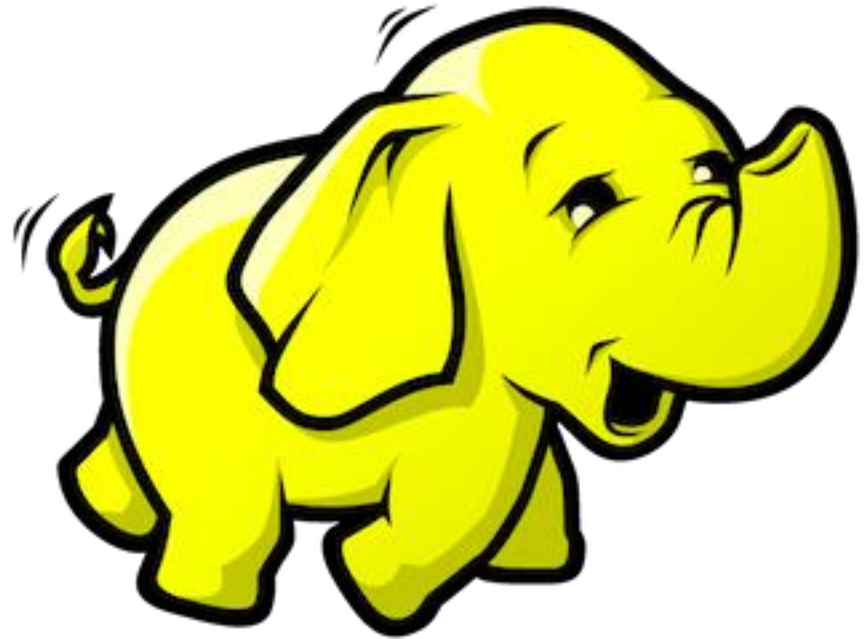
Apache Hadoop

- HDFS
- M/R

Apache Giraph

Apache Drill

High-level design described in a series of papers; no implementation available



HDFS: **THE HADOOP DISTRIBUTED FILE SYTEM**

HDFS Architecture

- HDFS has a master/slave architecture
- Master = **NameNode (NN)** manages the file system and regulates access to files by clients.
- Slaves = **DataNodes (DN)**, usually one per server in the cluster, manage storage attached to the server that they run on.

HDFS Architecture

- HDFS has a master/slave architecture
- Master = **NameNode (NN)** manages the file system and regulates access to files by clients.
- Slaves = **DataNodes (DN)**, usually one per server in the cluster, manage storage attached to the server that they run on.

NameNode



DataNodes



HDFS Architecture

- HDFS has a master/slave architecture
- Master = **NameNode (NN)** manages the file system and regulates access to files by clients.
- Slaves = **DataNodes (DN)**, usually one per server in the cluster, manage storage attached to the server that they run on.

- Files are transparently broken down into **blocks** (default 64 MB).
- Blocks are **replicated** across datanodes. Replication is **rack-aware**.

NameNode



DataNodes



HDFS Architecture

- HDFS has a master/slave architecture
- Master = **NameNode (NN)** manages the file system and regulates access to files by clients.
- Slaves = **DataNodes (DN)**, usually one per server in the cluster, manage storage attached to the server that they run on.

- Files are transparently broken down into **blocks** (default 64 MB).
- Blocks are **replicated** across datanodes. Replication is **rack-aware**.

dat0.txt 1 3

dat1.txt 2 4 5

NameNode



DataNodes



HDFS Architecture

- HDFS has a master/slave architecture
- Master = **NameNode (NN)** manages the file system and regulates access to files by clients.
- Slaves = **DataNodes (DN)**, usually one per server in the cluster, manage storage attached to the server that they run on.

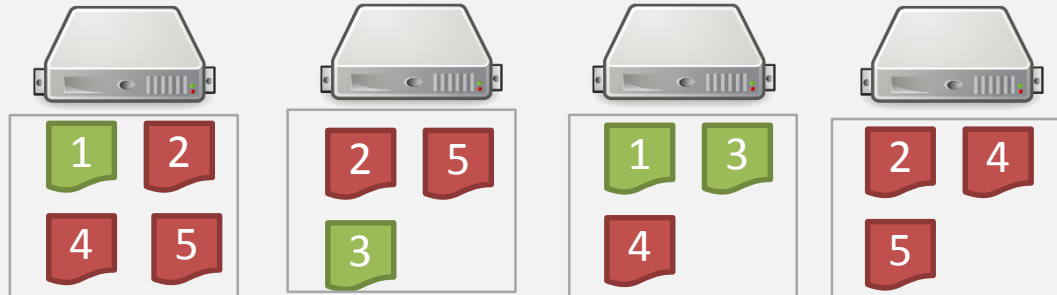
- Files are transparently broken down into **blocks** (default 64 MB).
- Blocks are **replicated** across datanodes. Replication is **rack-aware**.



NameNode



DataNodes



HDFS Architecture

- HDFS has a master/slave architecture
- Master = **NameNode (NN)** manages the file system and regulates access to files by clients.
- Slaves = **DataNodes (DN)**, usually one per server in the cluster, manage storage attached to the server that they run on.

- Files are transparently broken down into **blocks** (default 64 MB).
- Blocks are **replicated** across datanodes. Replication is **rack-aware**.



NameNode



MetaData(FileName, Replication Factor, Block Ids)

```
/users/sv/dat0.txt r:2 {1,3}
/users/sv/dat1.txt r:3 {2, 4, 5}
```

DataNodes



HDFS Architecture

- HDFS has a master/slave architecture
- Master = **NameNode (NN)** manages the file system and regulates access to files by clients.
- Slaves = **DataNodes (DN)**, usually one per server in the cluster, manage storage attached to the server that they run on.

- Optimized for:
 - Large files
 - Read throughput
 - Appending writes
- Replication ensures:
 - Durability
 - Availability
 - Throughput

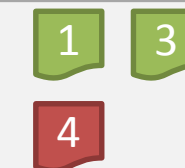
NameNode



MetaData(FileName, Replication Factor, Block Ids)

```
/users/sv/dat0.txt r:2 {1,3}
/users/sv/dat1.txt r:3 {2, 4, 5}
```

DataNodes



HDFS Implementation

“

*The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). **HDFS is built using the Java language**; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines.*

Source: Hadoop documentation

- This implies that clients only need to install a JAR file to access the HDFS

Typical HDFS commands

```
bin/hadoop fs -ls
```

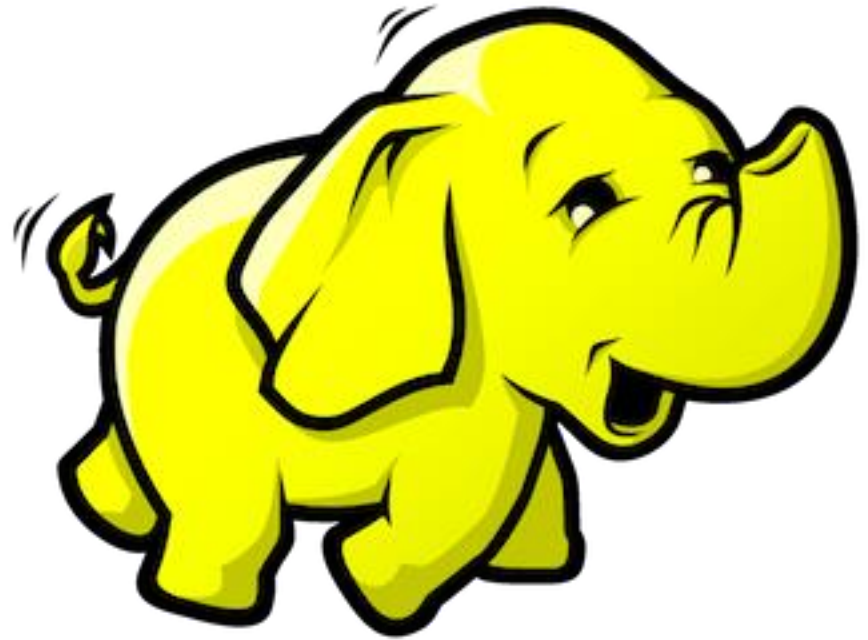
```
bin/hadoop fs -mkdir
```

```
bin/hadoop fs -copyFromLocal
```

```
bin/hadoop fs -copyToLocal
```

```
bin/hadoop fs -moveToLocal
```

```
bin/hadoop fs -rm
```



MAP/REDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS

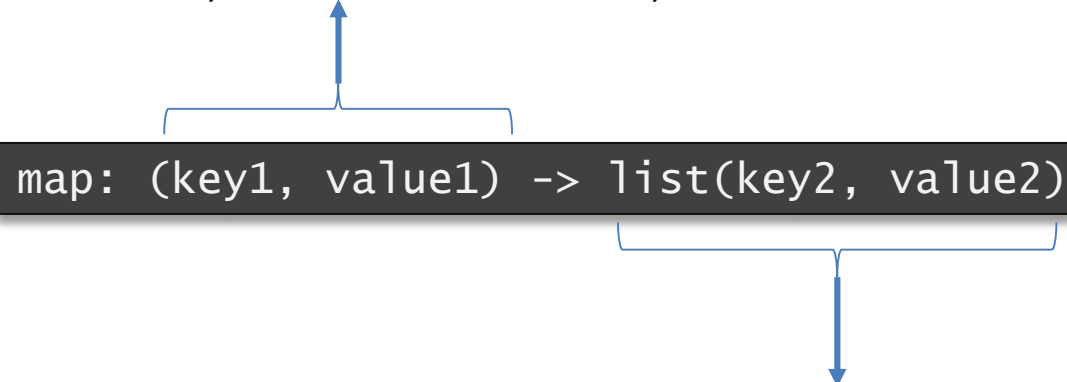
M/R Computational Model

- A M/R program (or **job**) is specified by two functions: **map** and **reduce**

M/R Computational Model

- A M/R program (or **job**) is specified by two functions: **map** and **reduce**

The input key/value pair represents a logical record in the input data source. In the case of a file this could be a line, or if the input source is a database table, this could be a record,



The diagram shows the map function signature: `map: (key1, value1) -> list(key2, value2)`. A blue bracket is positioned above the input parameters `(key1, value1)`, with an arrow pointing upwards to the text 'The input key/value pair represents a logical record in the input data source. In the case of a file this could be a line, or if the input source is a database table, this could be a record,'. Another blue bracket is positioned below the output parameter `list(key2, value2)`, with an arrow pointing downwards to the text 'A single input key/value pair may result in zero or more output key/value pairs.'

```
map: (key1, value1) -> list(key2, value2)
```

A single input key/value pair may result in zero or more output key/value pairs.

M/R Computational Model

- A M/R program (or **job**) is specified by two functions: **map** and **reduce**

The reduce function is called once per unique map output key, and receives a list of all values emitted for that key



```
reduce: (key2, list(value2)) -> list(key3, value3)
```

Like the map function, reduce can output zero to many key/value pairs.

M/R Computational Model

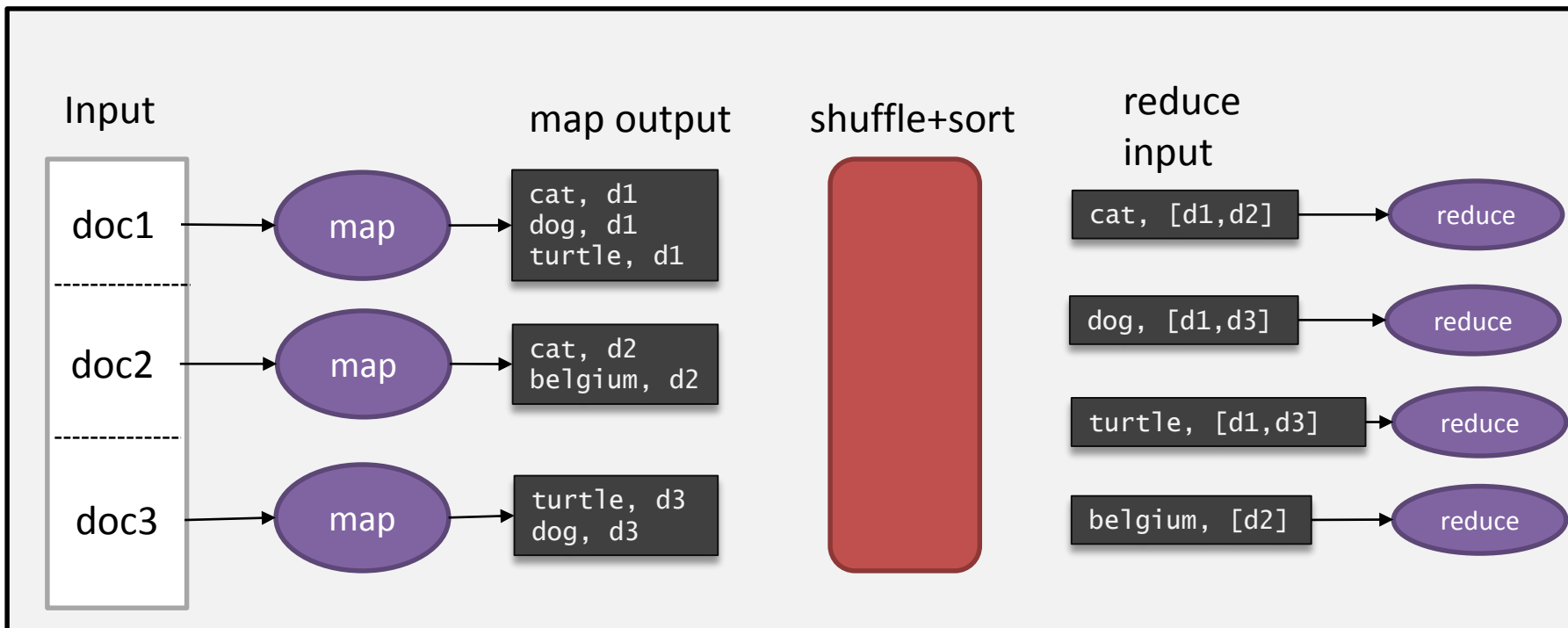
- For each word occurring in a document on the Web, count the number of occurrences across all documents.

```
def map(docid, line):  
    for each word in line:  
        yield (word, docid)
```

```
def reduce(word, list-of-docids):  
    yield (word, sizeof(list-of-docids))
```

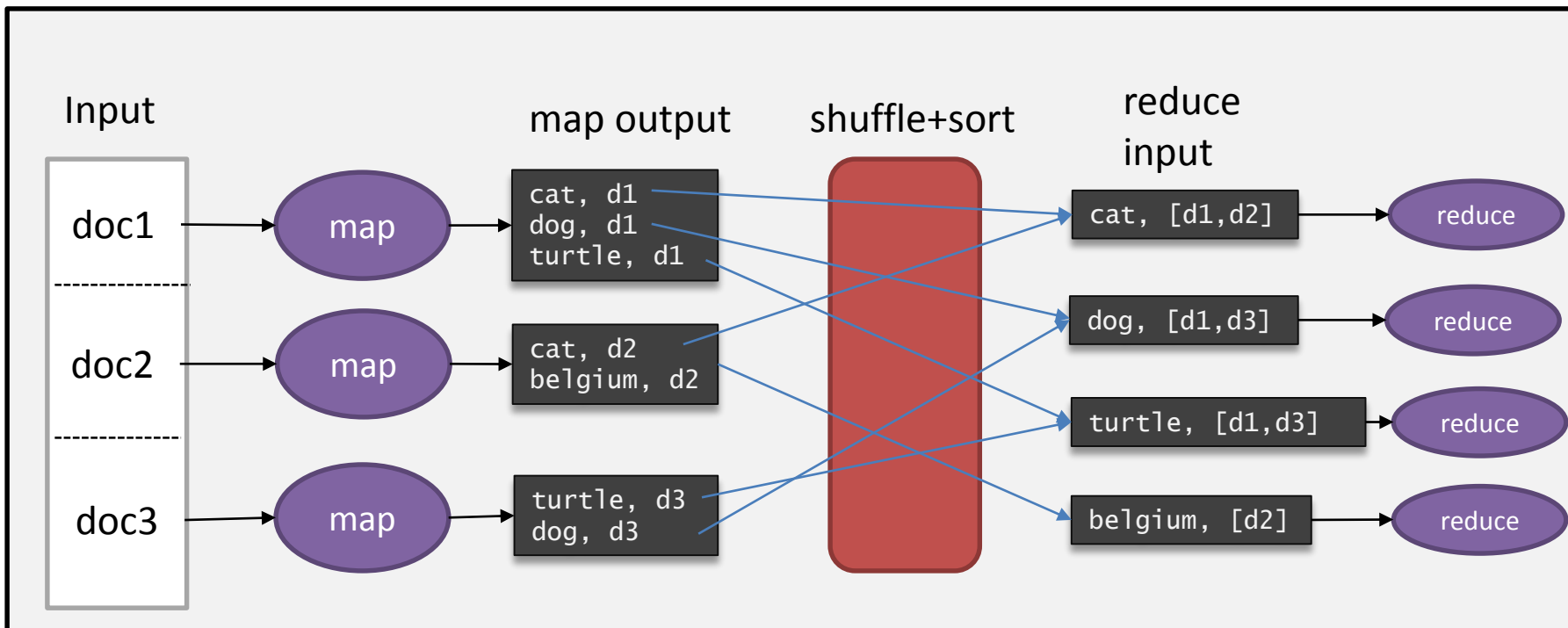
M/R: opportunity for parallelism

- We can **spawn multiple copies** of the map function (called **map tasks**) in parallel (at most one for each key/value pair).
- Likewise, we can **spawn multiple copies** of the reduce function (called **reduce tasks**) in parallel (at most one for each unique key output by the map).



M/R: opportunity for parallelism

- We can **spawn multiple copies** of the map function (called **map tasks**) in parallel (at most one for each key/value pair).
- Likewise, we can **spawn multiple copies** of the reduce function (called **reduce tasks**) in parallel (at most one for each unique key output by the map).



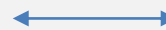
M/R Execution: Architecture

- Master = **JobTracker** – accepts jobs, decomposes them into map and reduce tasks, and schedules them for remote execution on child nodes.
- Slave = **TaskTracker** – accepts tasks from Jobtracker and spawns child processes to do the actual work.
- Idea: **ship computation to data**

- The JobTracker accepts M/R jobs.
- If the input is a HDFS file, a map task is created for each block and sent to the node holding that block for execution.
- Map output is written to local disk.

NameNode

JobTracker



DataNodes = TaskTrackers



Job1 Map task 1
Map task 2
Reduce task 1
Reduce task 2

Job2 ...

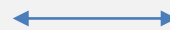
M/R Execution: Architecture

- Master = **JobTracker** – accepts jobs, decomposes them into map and reduce tasks, and schedules them for remote execution on child nodes.
- Slave = **TaskTracker** – accepts tasks from Jobtracker and spawns child processes to do the actual work.
- Idea: **ship computation to data**

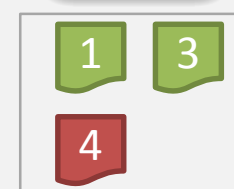
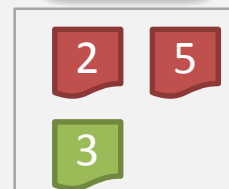
- The JobTracker creates reduce tasks intelligently
- Reduce tasks read the map outputs over the network and write their output back to HDFS.

NameNode

JobTracker



DataNodes = TaskTrackers



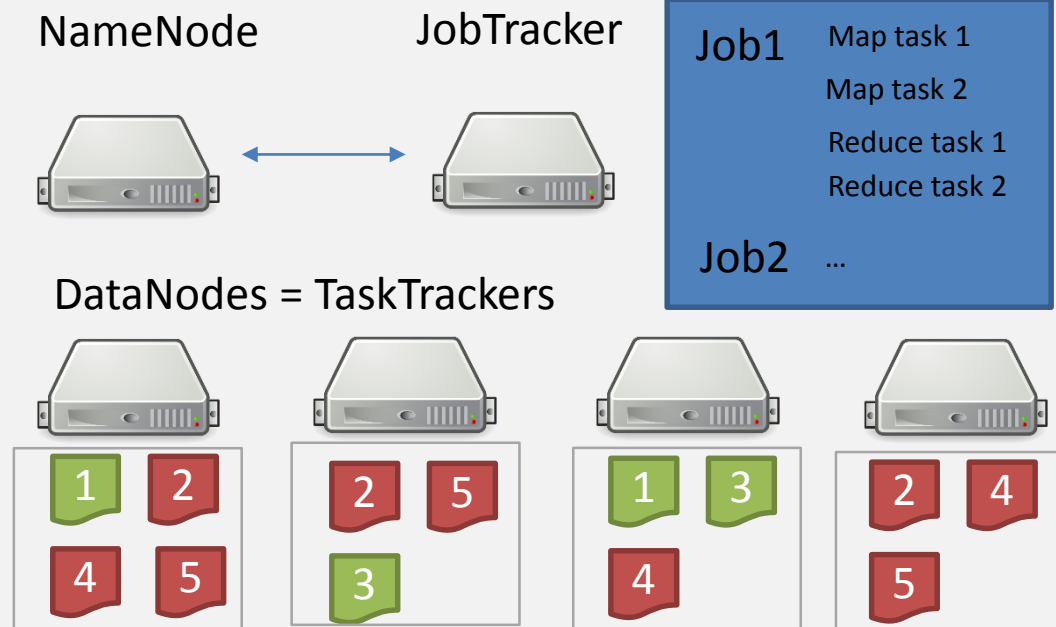
Job1 Map task 1
Map task 2
Reduce task 1
Reduce task 2

Job2 ...

M/R Execution: Architecture

- Master = **JobTracker** – accepts jobs, decomposes them into map and reduce tasks, and schedules them for remote execution on child nodes.
- Slave = **TaskTracker** – accepts tasks from Jobtracker and spawns child processes to do the actual work.
- Idea: **ship computation to data**

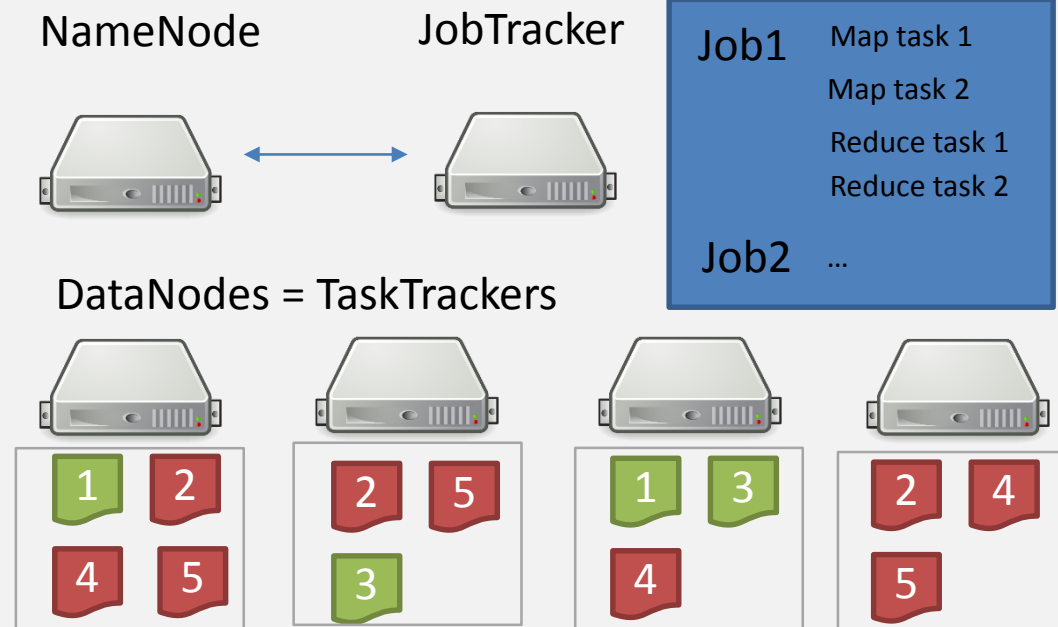
- **Load balancing:** The JobTracker monitors for stragglers and may spawn additional map on datanodes that hold a block replica. Whichever node completes first is allowed to proceed. The other(s) is/are killed.



M/R Execution: Architecture

- Master = **JobTracker** – accepts jobs, decomposes them into map and reduce tasks, and schedules them for remote execution on child nodes.
- Slave = **TaskTracker** – accepts tasks from Jobtracker and spawns child processes to do the actual work.
- Idea: **ship computation to data**

- **Failures:** In clusters with 1000s of nodes, hardware failures occur frequently. The same mechanism as for load balancing allows to cope with such failures



References

- S. Ghemawate, H. Gobioff H.-T. Leung
The Google File System
<http://research.google.com/archive/gfs-sosp2003.pdf>
- HDFS architecture: <http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>
- Jeffrey Dean and Sanjay Ghemawat
MapReduce: Simplified Data Processing on Large Clusters
Communications of the ACM 2008
- G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski .
Pregel: a system for large scale graph processing
http://kowshik.github.io/JPregel/pregel_paper.pdf
- L. A. Barroso, J. Clidaras, U. Hölzle
The datacenter as a computer.
<http://www.morganclaypool.com/doi/abs/10.2200/S00516ED2V01Y201306CAC024>