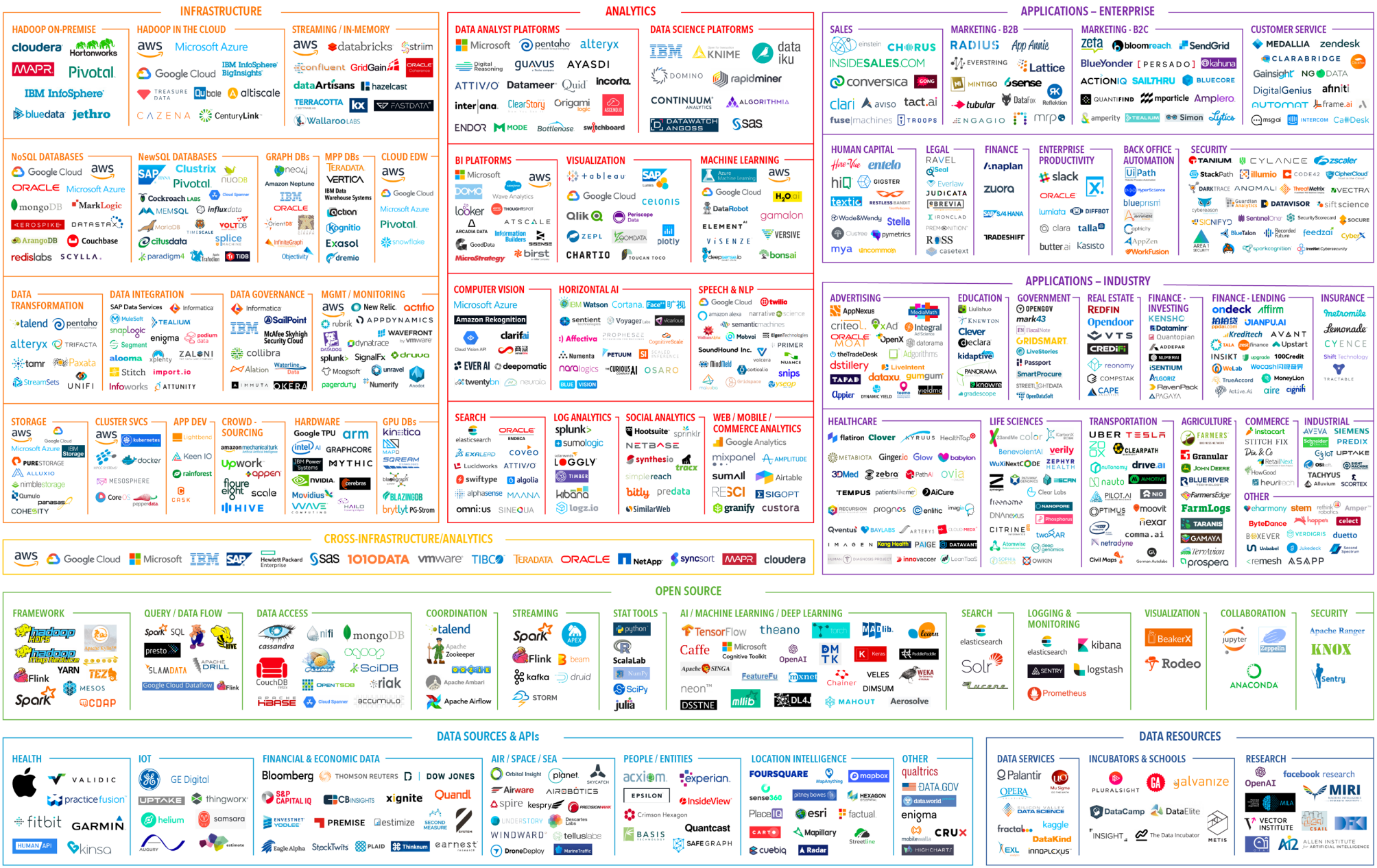


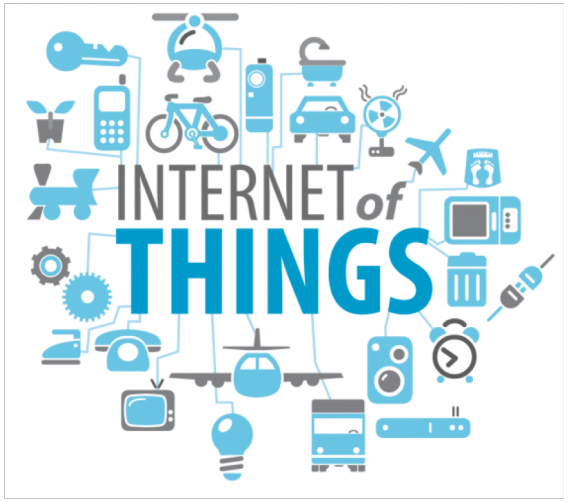

서울대학교
융합과학기술대학원
Seoul National University
Graduate School of Convergence
Science and Technology

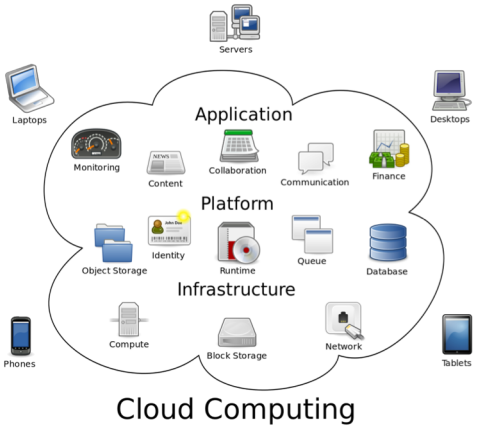
Hadoop & Spark

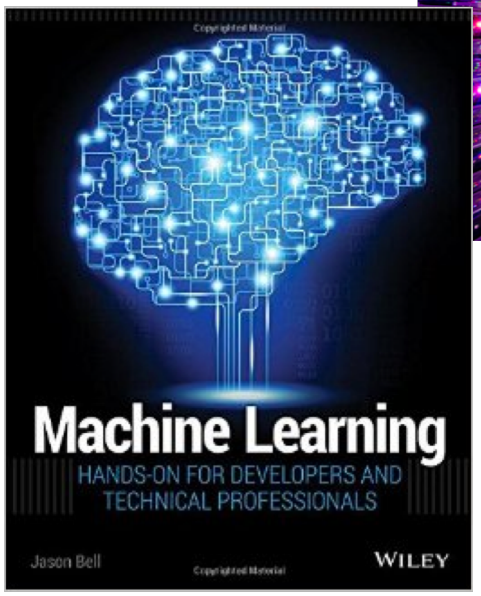
BIG DATA & AI LANDSCAPE 2018



빅데이터 Paradigm

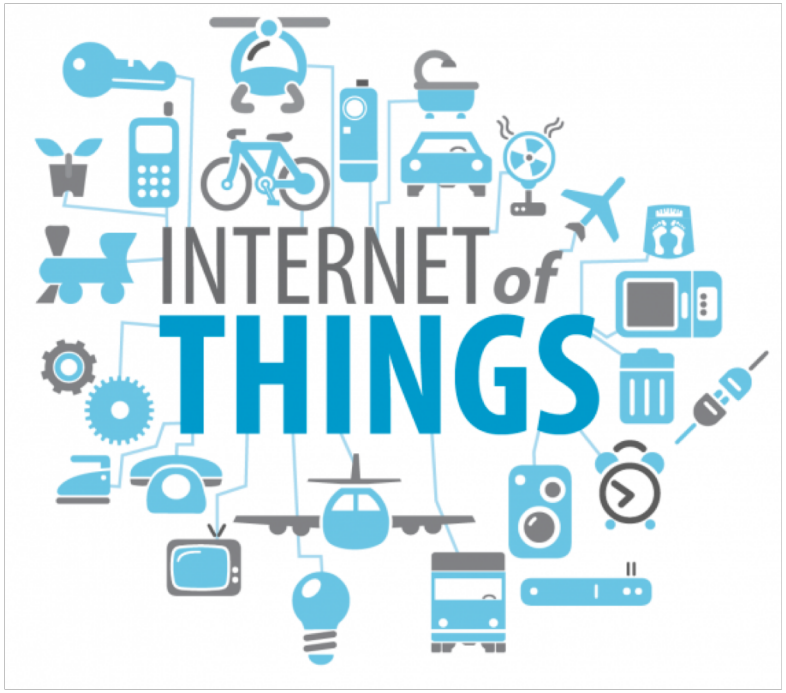




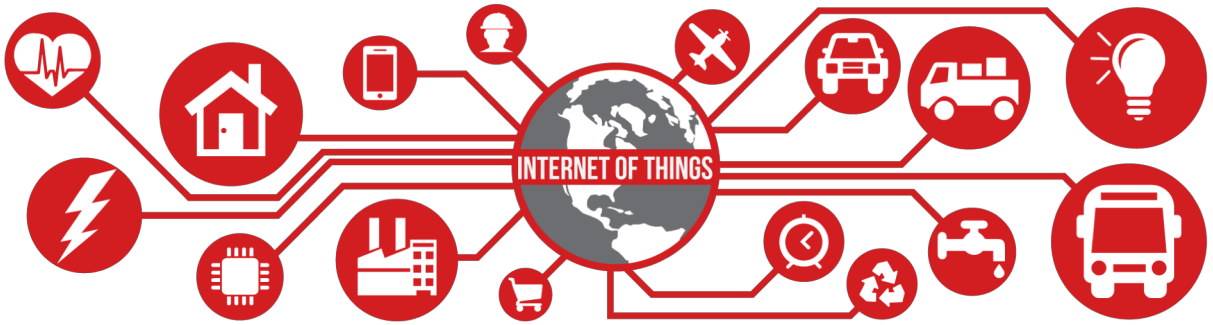


IoT (Internet of Things, 사물인터넷)

- 여러가지 물건이나 센서들이 네트워크로 연결되어서 기존에는 불가능한 서비스를 가능하게 하는것



- Wearable computing
  - 스마트와치, Google glass, Hologram
- Smart devices: iBeacon, 온도조절기
- 자동차, 스마트빌딩, 교실, 공장

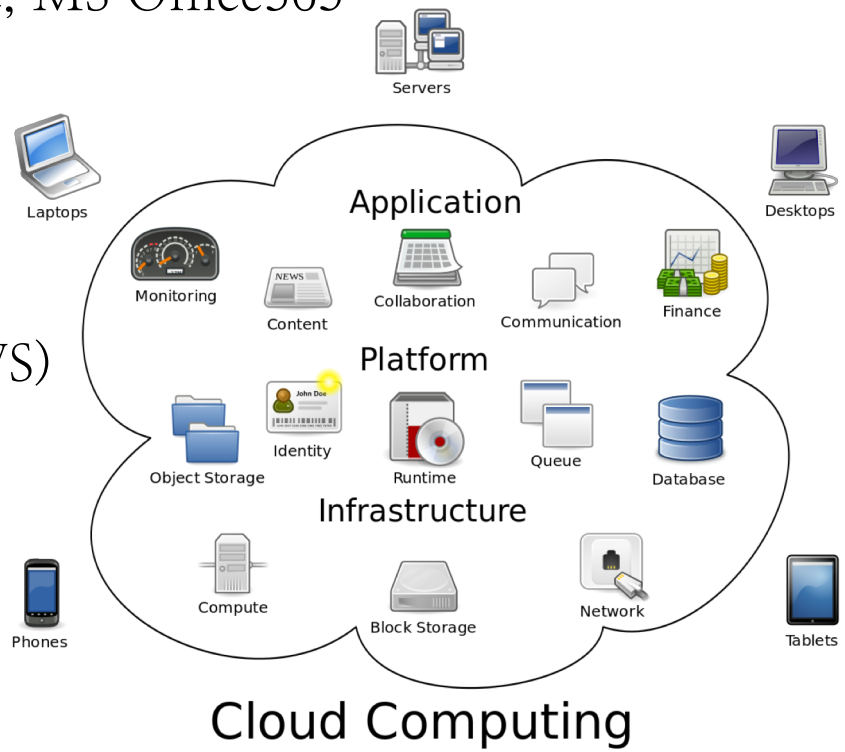


Cloud Computing

- 중요한 정보가 어디에 있는가?
  - 더이상 책상에 있는 컴퓨터, USB에 있지 않다.
  - 예) 온라인 계좌, 이메일, Google Drive, MS Office365

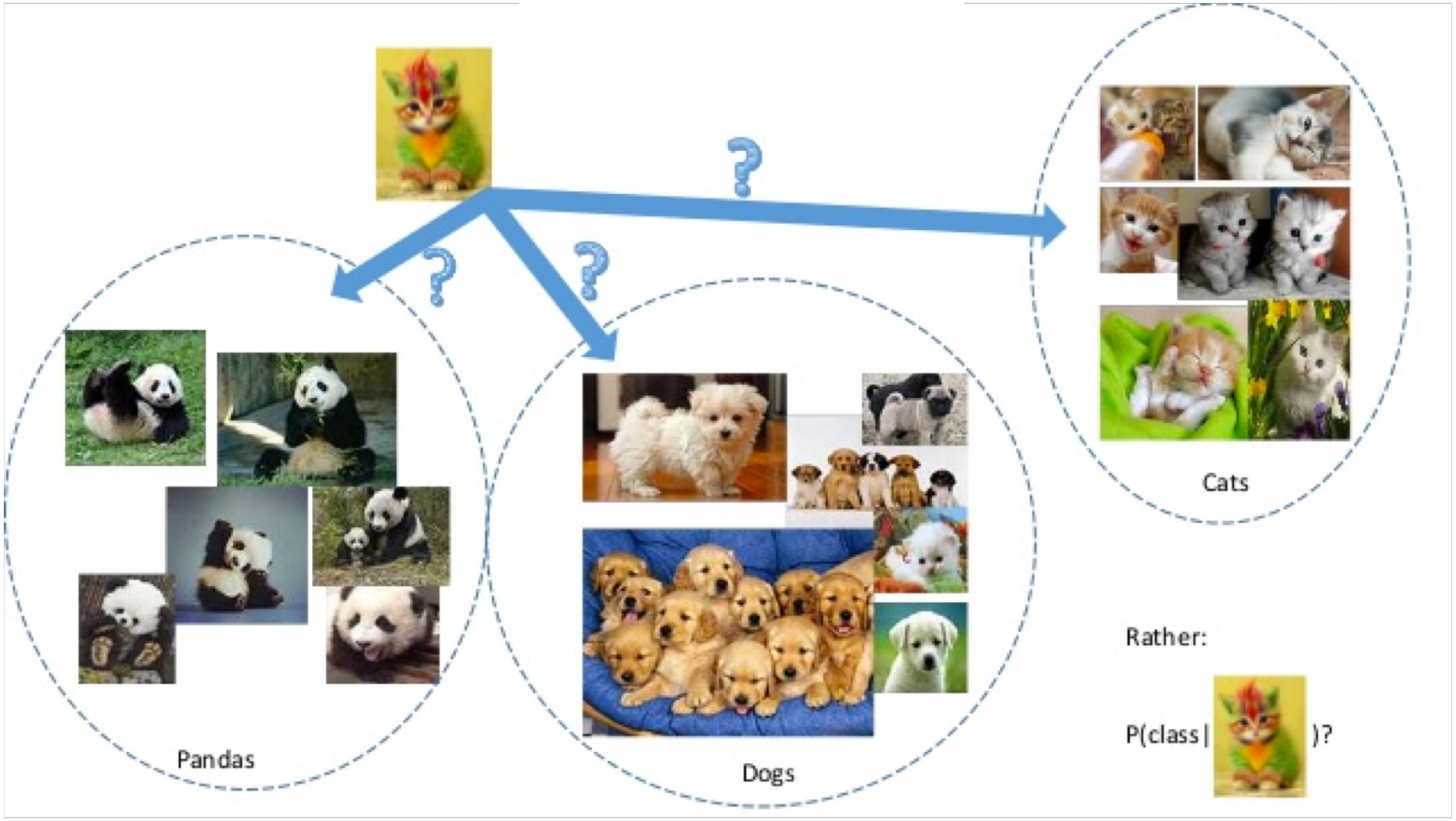
- 더이상 서버를 사지 않아도 된다
  - 필요할때 온라인으로 빌려서 사용
  - 노무관리의 예 (salesforce.com)
  - 서버 렌트 (Amazon Web Services, AWS)
  - 저장 장치 (NDrive, Dropbox)

- 보안 & Privacy 가 고려되어야 한다



Machine Learning

- 수집된 정보로 부터 insight를 얻는다.
- **<u>많은 정보가 수집</u>**되어 있어야 가능



http://biztribune.co.kr/n_news/news/view.html?no=29739

Harvard Business Review

DECISION MAKING

Good Data Won't Guarantee Good Decisions

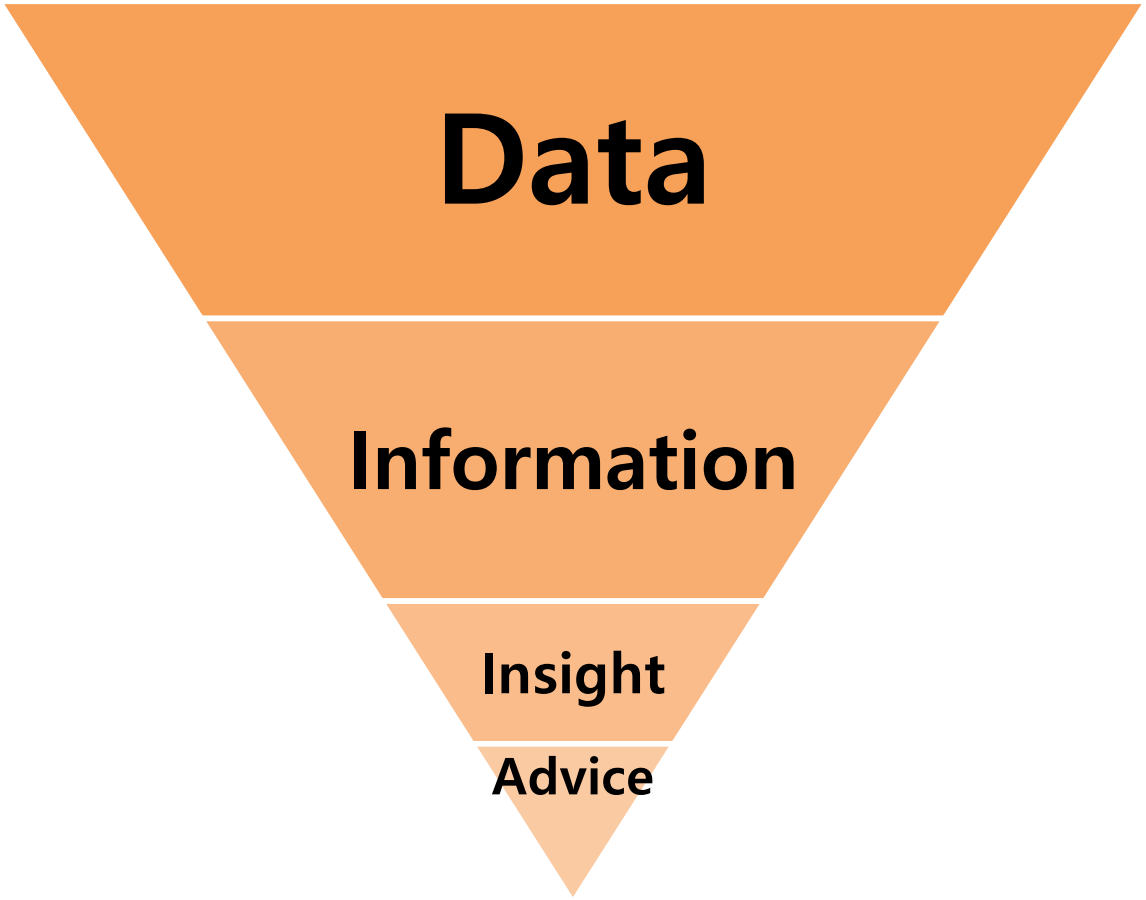
by Shvetank Shah, Andrew Horne, and Jaime Capellá

FROM THE APRIL 2012 ISSUE

Global businesses have entered a new era of decision making. The ability to gather, store, access, and analyze data has grown exponentially over the past decade, and companies now spend tens of millions of dollars to manage the information streaming in from suppliers and

목표: 인간이 사용할 수 있는 통찰발견

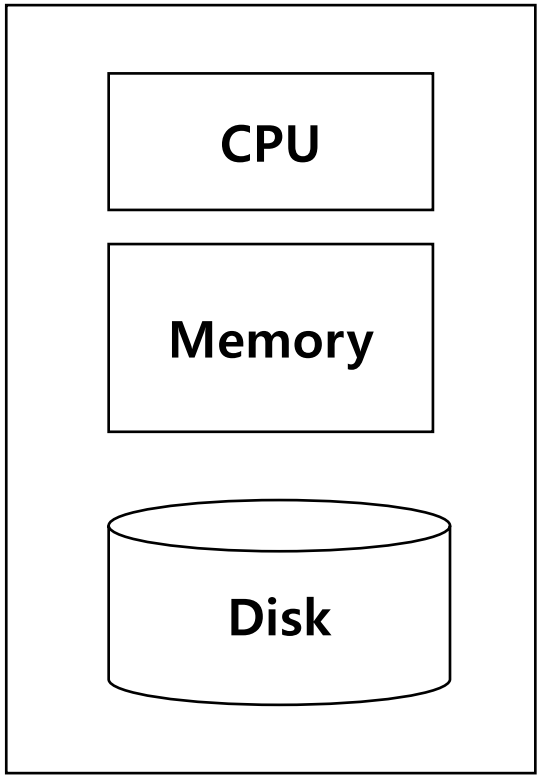
대규모 자료를 분석해 insight를 도출



What is Data Mining?

- Given lots of data
- Discover patterns and models that are:
  - Valid: hold on new data with some certainty
  - Useful: should be possible to act on the item
  - Unexpected: non-obvious to the system
  - Understandable: humans should be able to interpret the pattern

Single Node Architecture



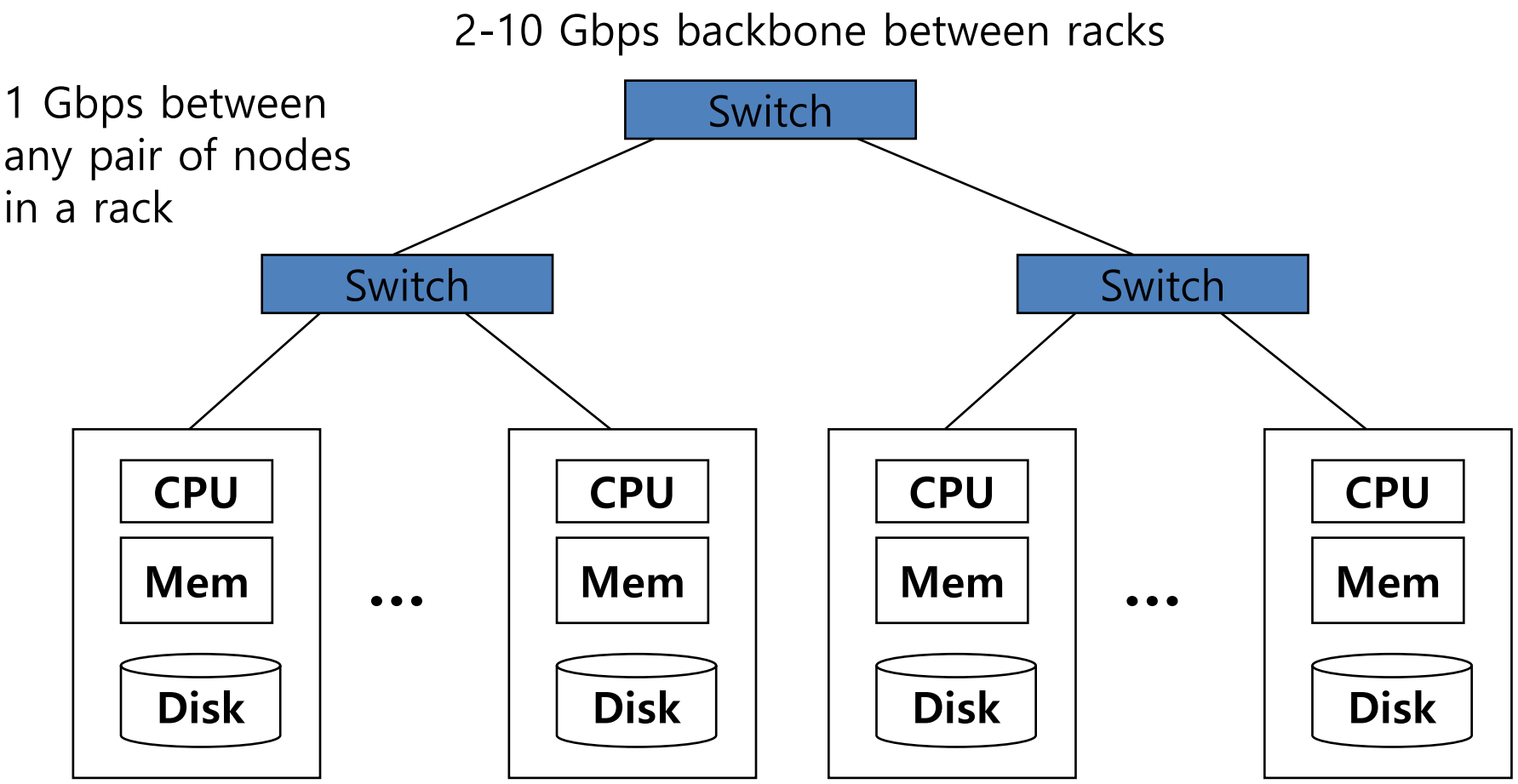
Machine Learning, Statistics

"Classical" Data Mining

Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- Today, a standard architecture for such problems is emerging:
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture



Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

Large-scale Computing

- Large-scale computing for data mining problems on commodity hardware
- Challenges:
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - Machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!

Idea and Solution

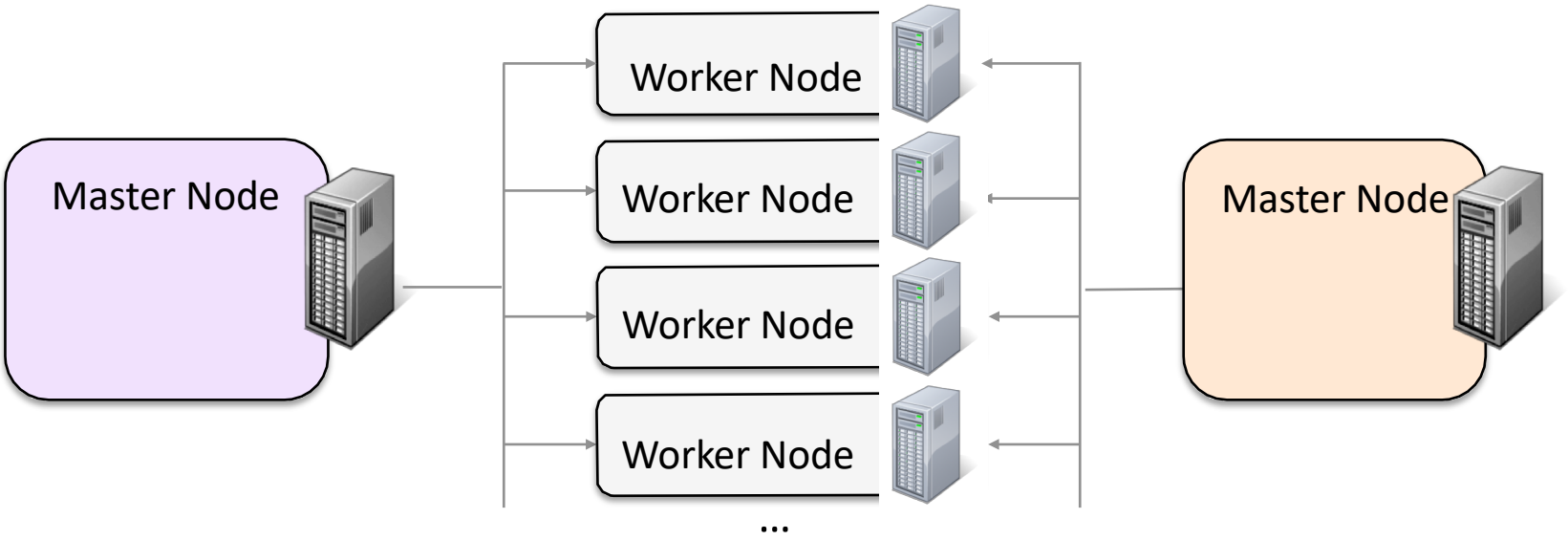
- Issue: Copying data over a network takes time
- Idea:
  - Bring computation close to the data
  - Store files multiple times for reliability
- Map-reduce addresses these problems
  - Google's computational/data manipulation model
  - Elegant way to work with big data
  - Storage Infrastructure – File system
    - Google: GFS. Hadoop: HDFS
  - Programming model
    - Map-Reduce

Storage Infrastructure

- Problem:
  - If nodes fail, how to store data persistently?
- Answer:
  - Distributed File System:
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- Typical usage pattern
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

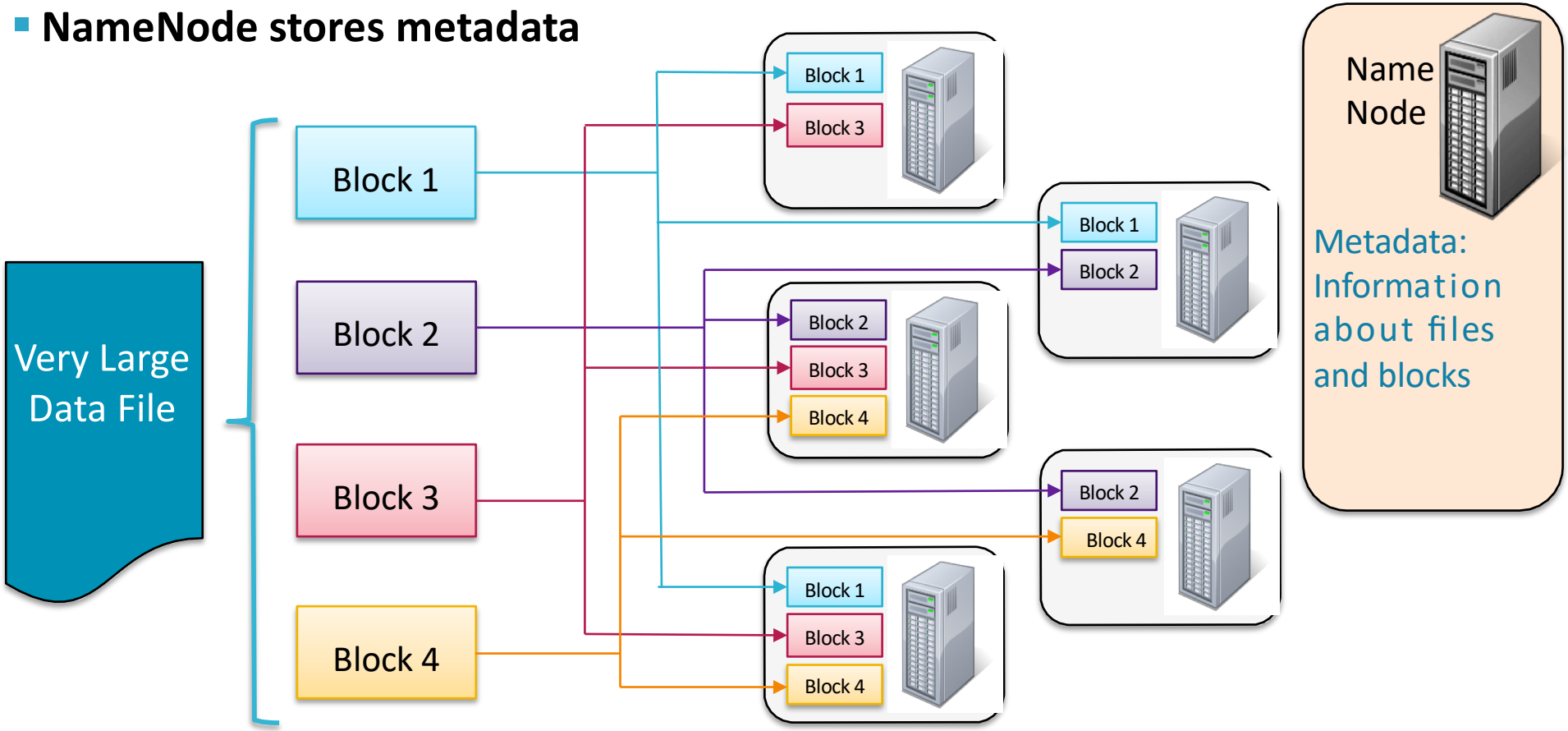
Hadoop Cluster Terminology

- **A *cluster* is a group of computers working together**
  - Provides data storage, data processing, and resource management
- **A *node* is an individual computer in the cluster**
  - *Master* nodes manage distribution of work and data to *worker* nodes
- **A *daemon* is a program running on a node**
  - Each Hadoop daemon performs a specific function in the cluster

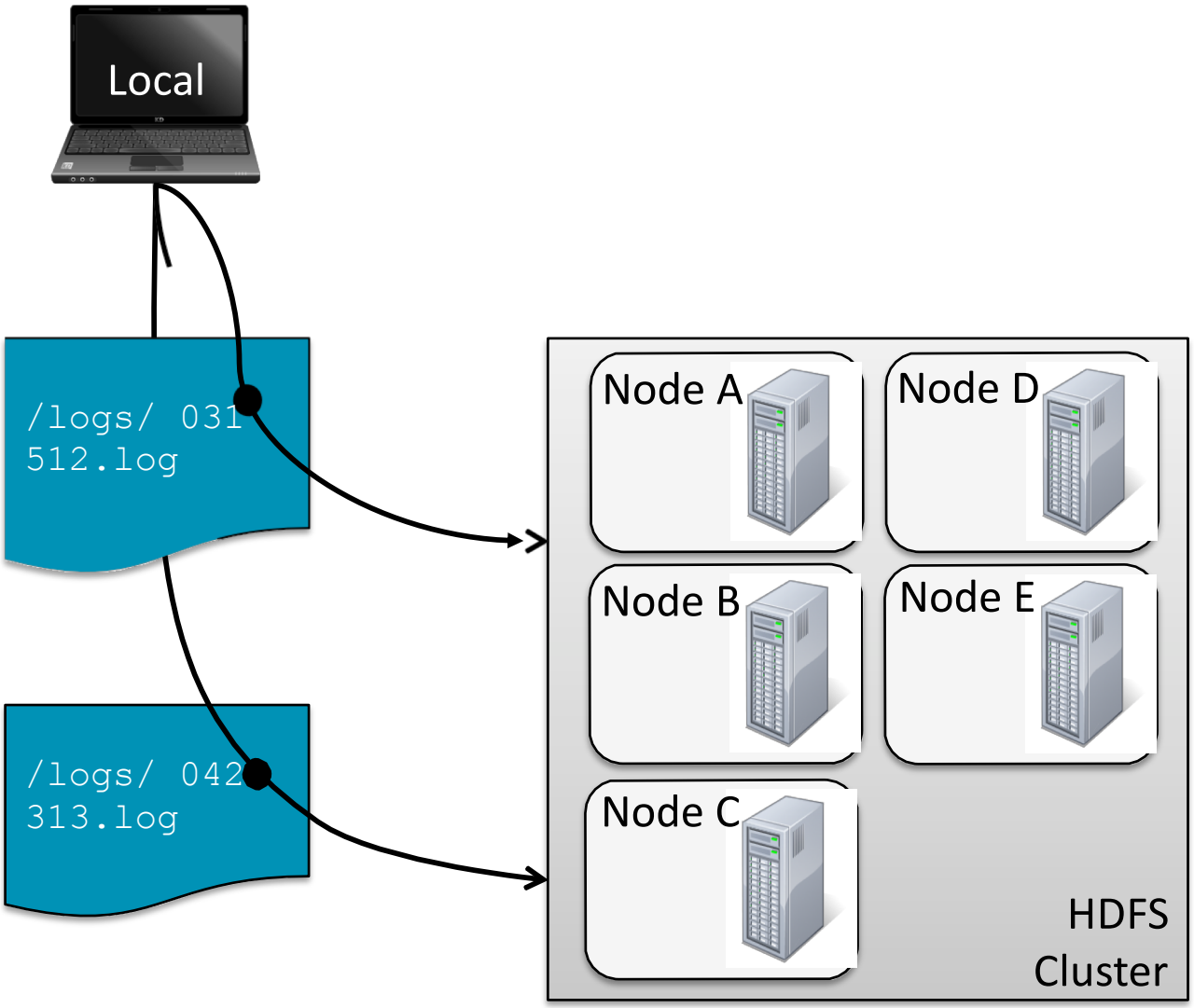


HDFS: How Files Are Stored

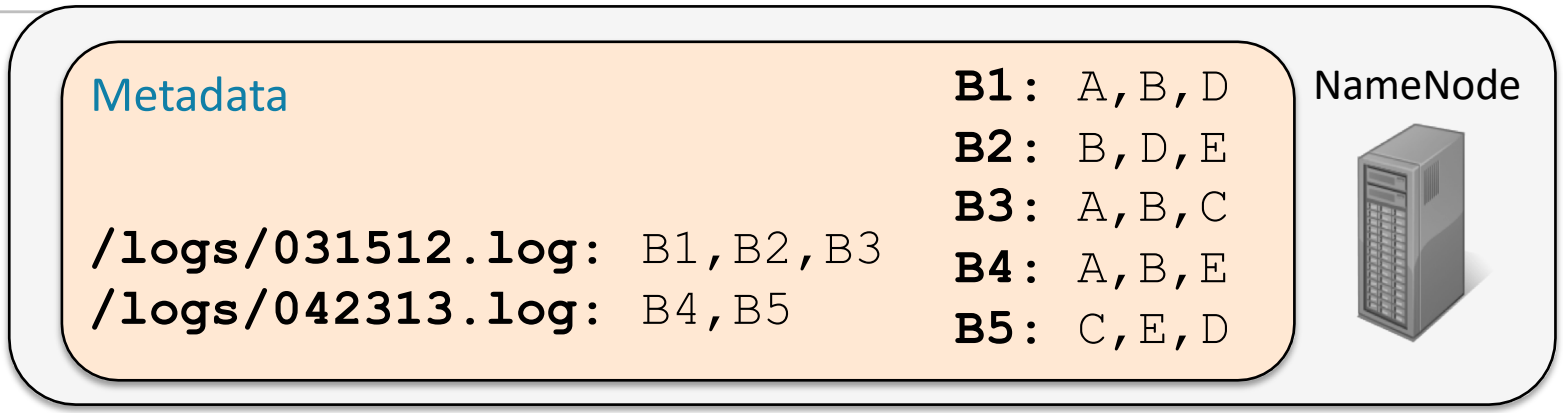
- **Data files are split into 128MB blocks which are distributed at load time**
- **Each block is replicated on multiple data nodes (default 3x)**
- **NameNode stores metadata**

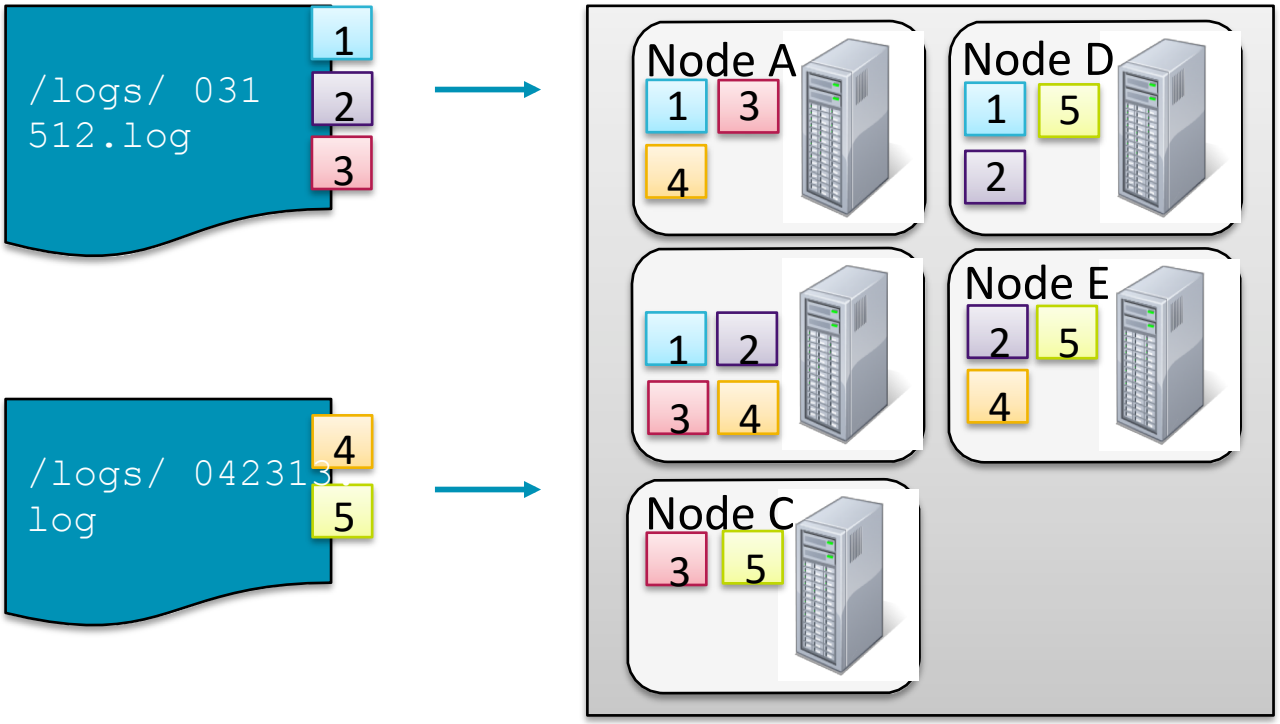


Example: Storing and Retrieving Files (1)

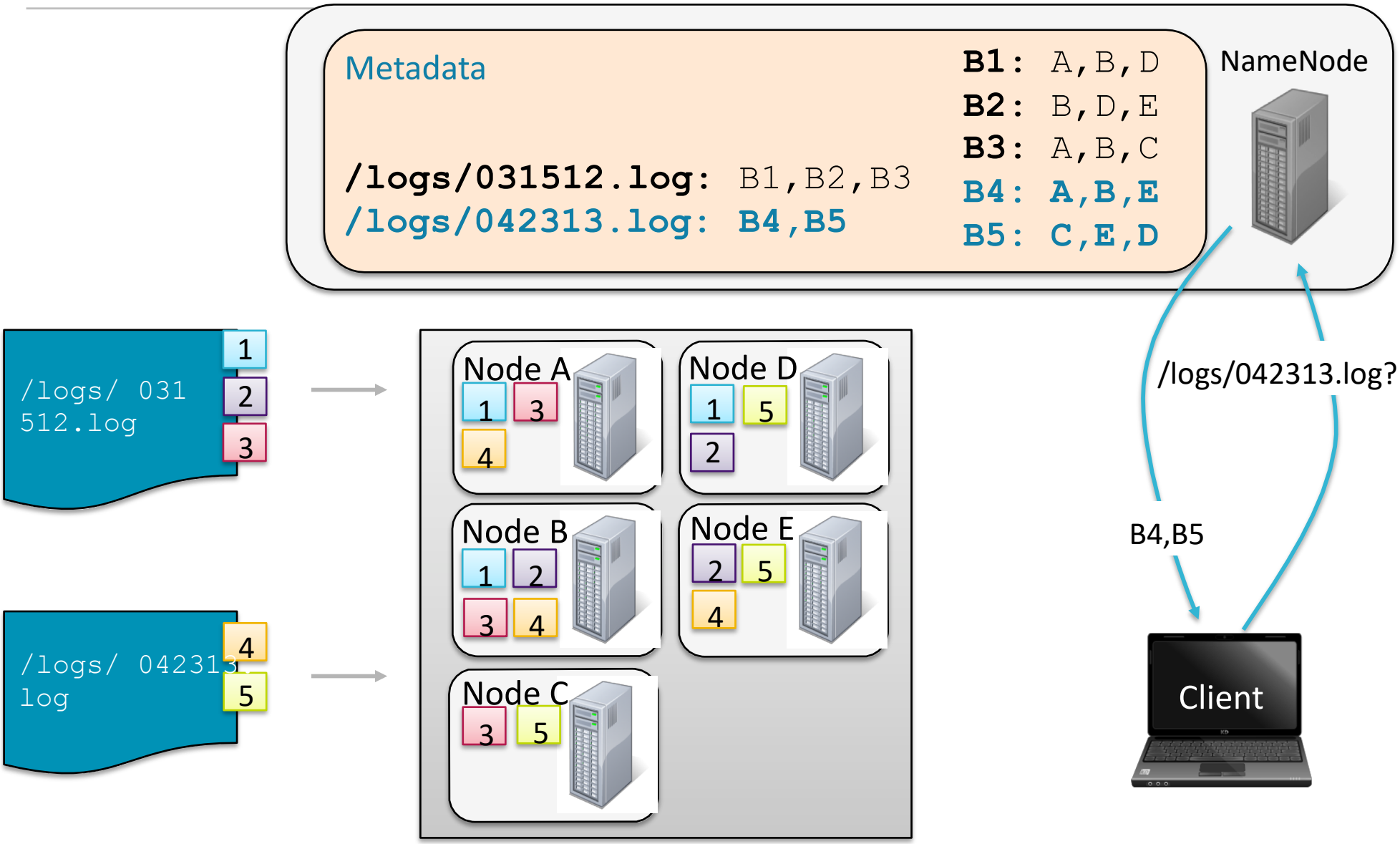


Example: Storing and Retrieving Files (2)

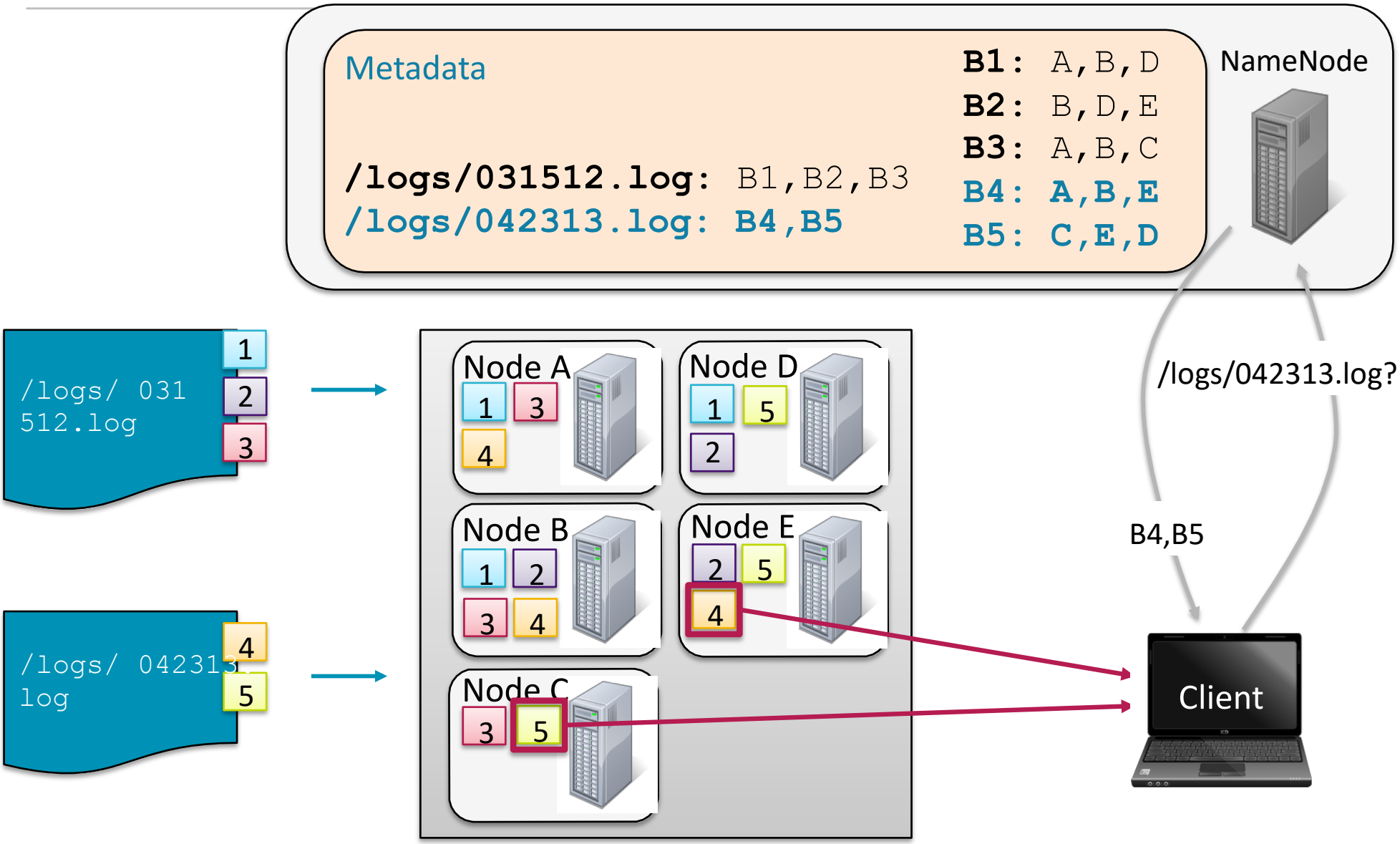




Example: Storing and Retrieving Files (3)



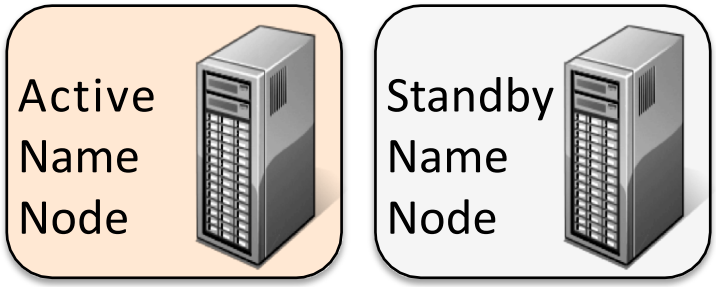
Example: Storing and Retrieving Files (4)



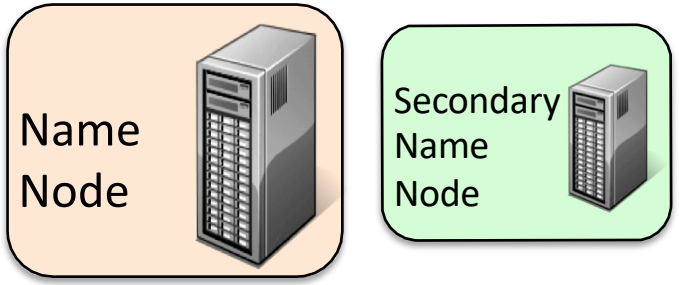
HDFS NameNode Availability

- **The NameNode daemon must be running at all times**
  - If the NameNode stops, the cluster becomes inaccessible

- **HDFS is typically set up for High Availability**
  - Two NameNodes: Active and Standby



- **Small clusters may use 'Classic mode'**
  - One NameNode
  - One "helper" node called
    the Secondary NameNode
    - Bookkeeping, not backup



HDFS Command Line Examples (1)

- **Copy file `foo.txt` from local disk to the user's directory in HDFS**

```
$ hdfs dfs -put foo.txt foo.txt
```

  - This will copy the file to `/user/username/foo.txt`

- **Get a directory listing of the user's home directory in HDFS**

```
$ hdfs dfs -ls
```

- **Get a directory listing of the HDFS root directory**

```
$ hdfs dfs -ls /
```

HDFS Command Line Examples (2)

- **Display the contents of the HDFS file `/user/fred/bar.txt`**

```
$ hdfs dfs -cat /user/fred/bar.txt
```

- **Copy that file to the local disk, named as `baz.txt`**

```
$ hdfs dfs -get /user/fred/bar.txt baz.txt
```

- **Create a directory called `input` under the user's home directory**

```
$ hdfs dfs -mkdir input
```

Note: `copyFromLocal` is a synonym for `put`; `copyToLocal` is a synonym for `get`

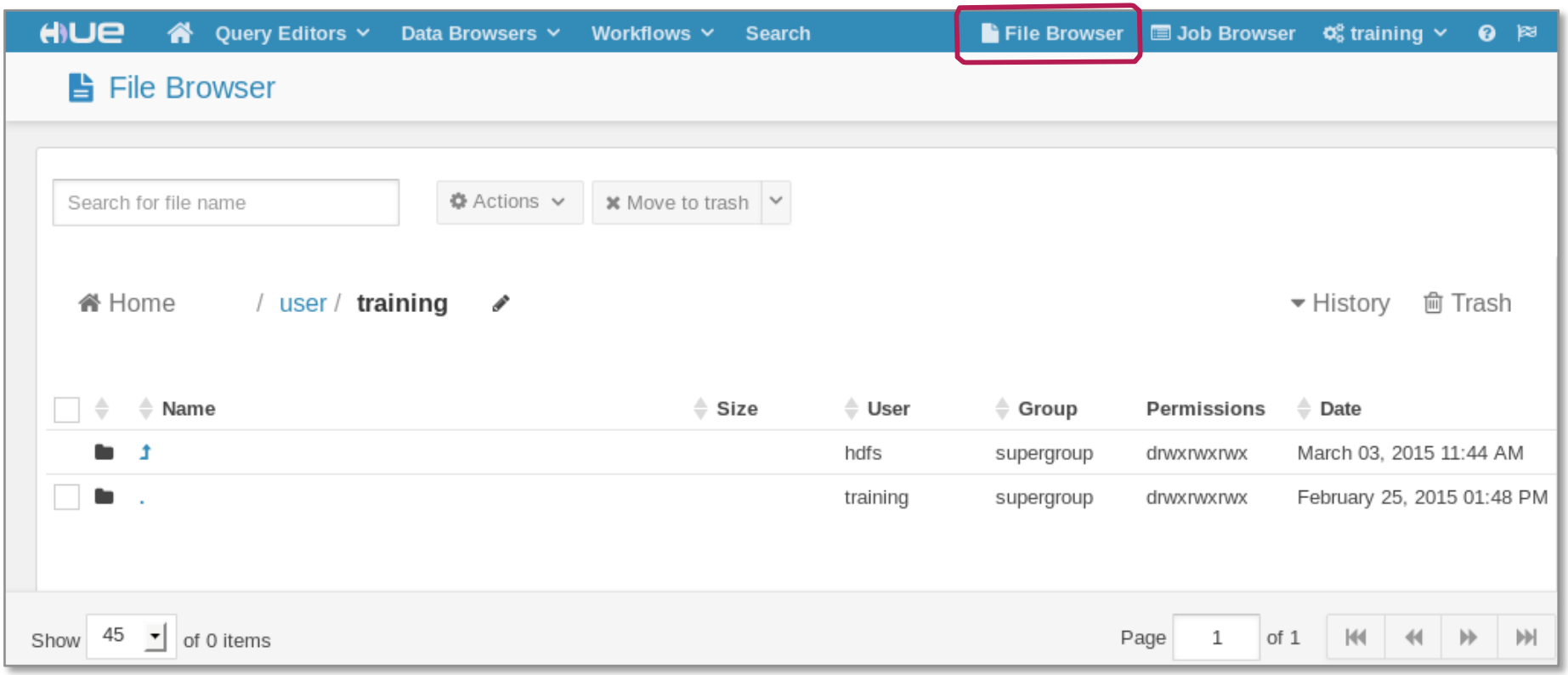
HDFS Command Line Examples (3)

- **Delete the directory `input_old` and all its contents**

```
$ hdfs dfs -rm -r input_old
```

The Hue HDFS File Browser

- The File Browser in Hue lets you view and manage your HDFS directories and files
  - Create, move, rename, modify, upload, download and delete directories and files
  - View file contents



Programming Model: MapReduce

Warm-up task:

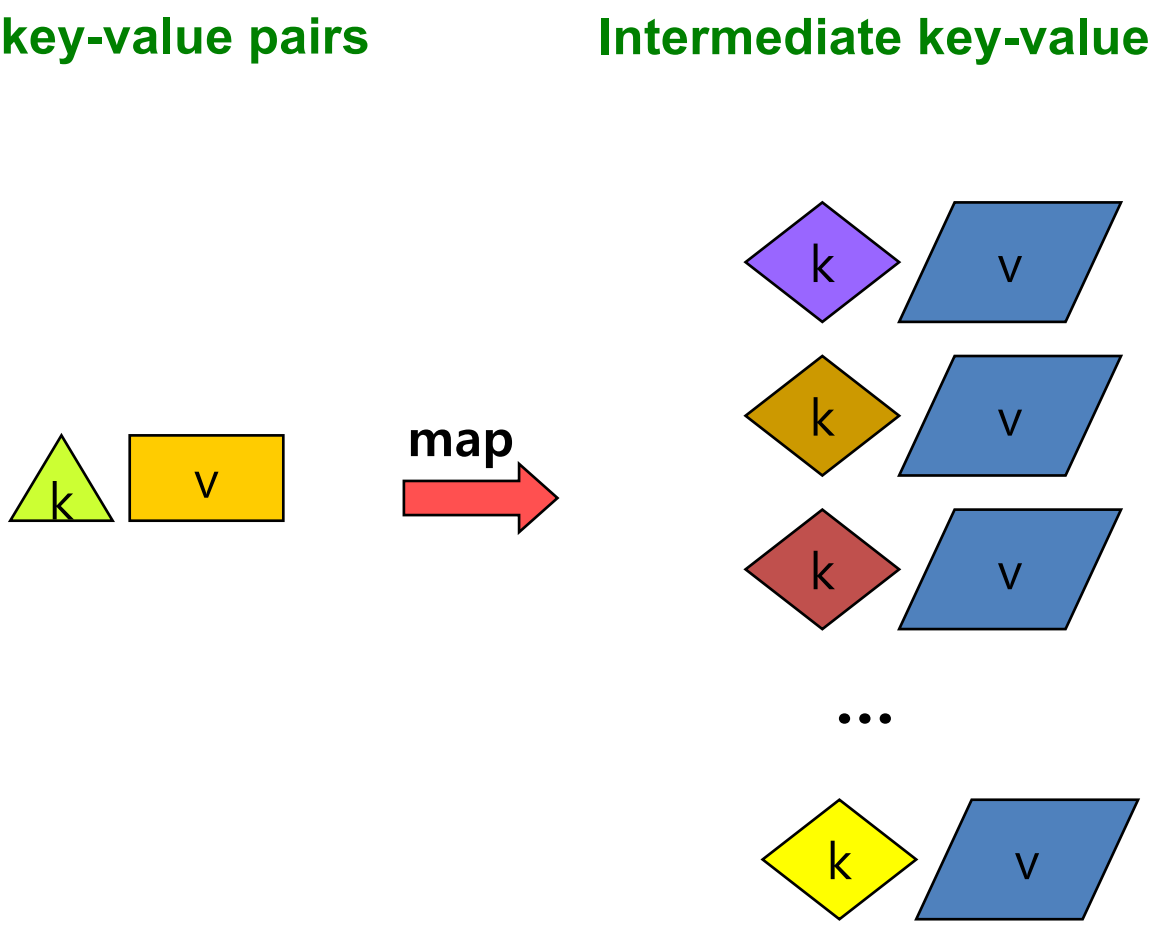
- We have a huge text document
- Count the number of times each distinct word appears in the file
- Sample application:
  - Analyze web server logs to find popular URLs

MapReduce: Overview

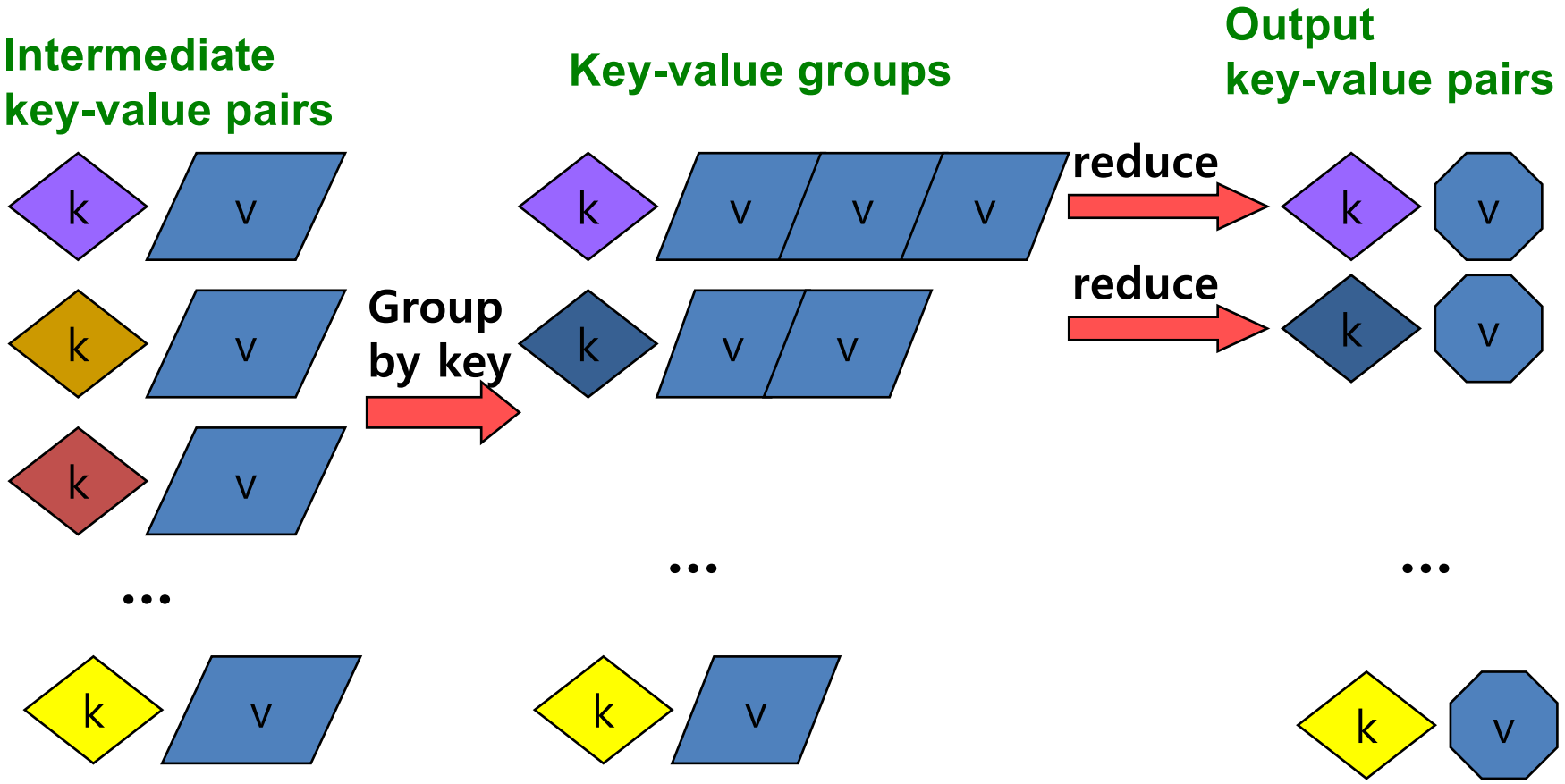
- Sequentially read a lot of data
- Map:
  - Extract something you care about
- **Group by key:** Sort and Shuffle
- Reduce:
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

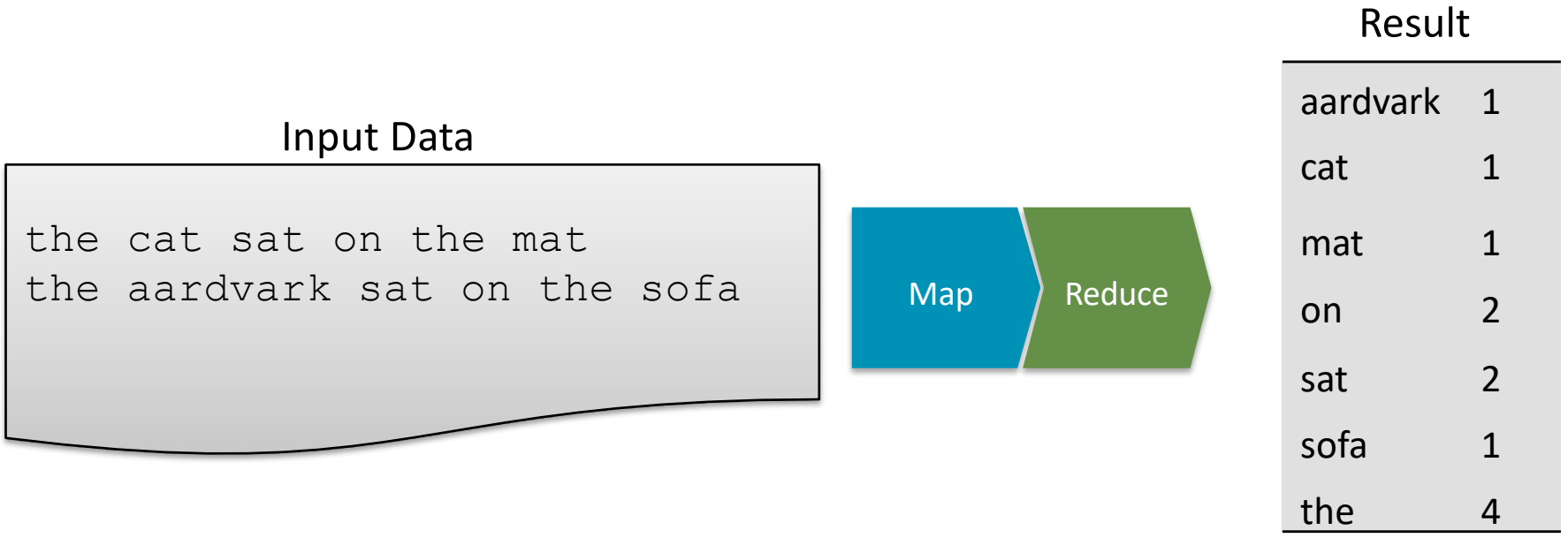
MapReduce: The Map Step



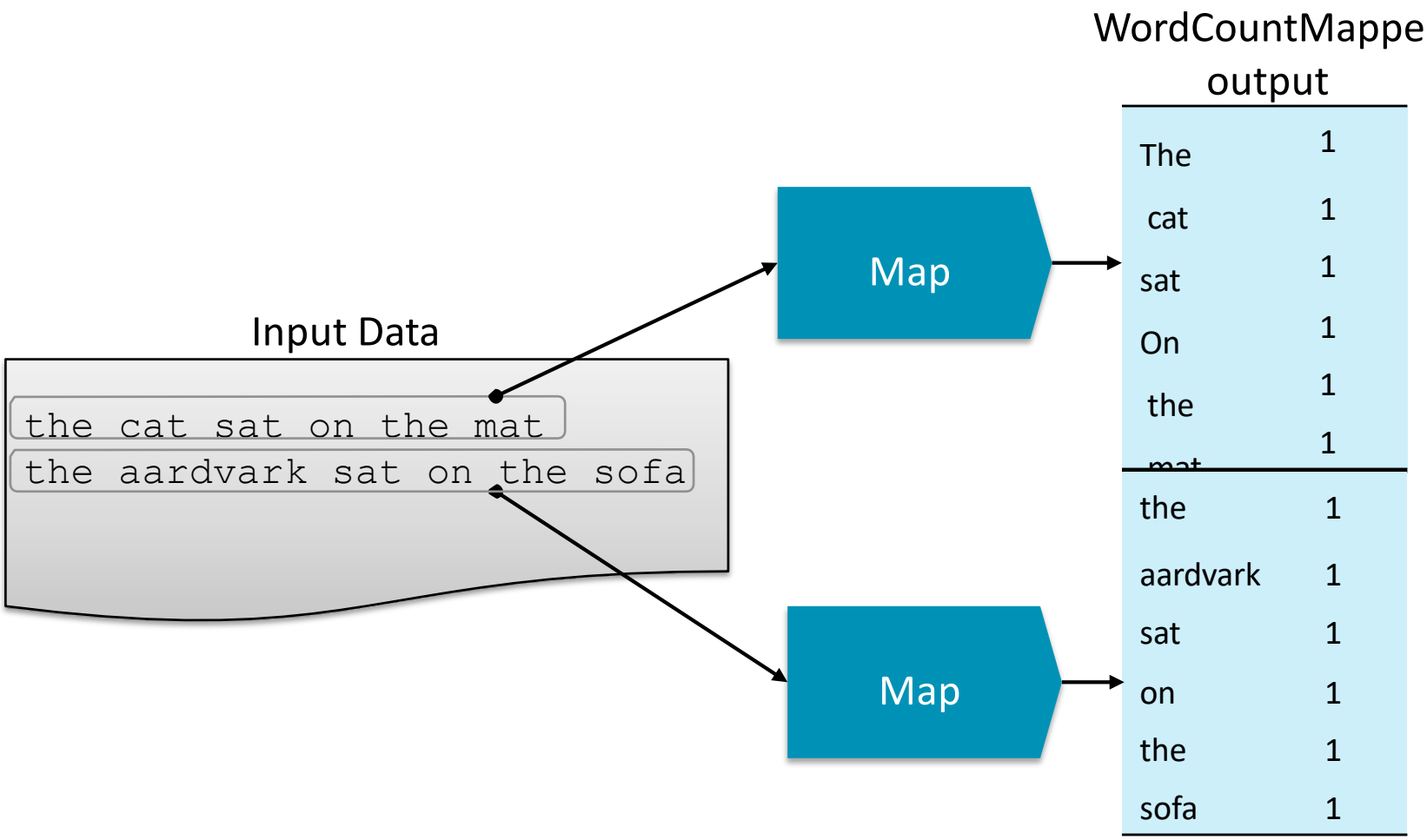
MapReduce: The Reduce Step



Example: Word Count



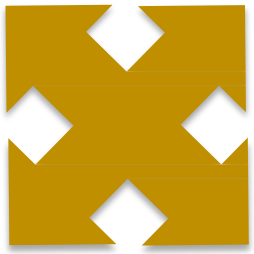
Example: The WordCount Mapper



Example: Shuffle & Sort

Mapper Output

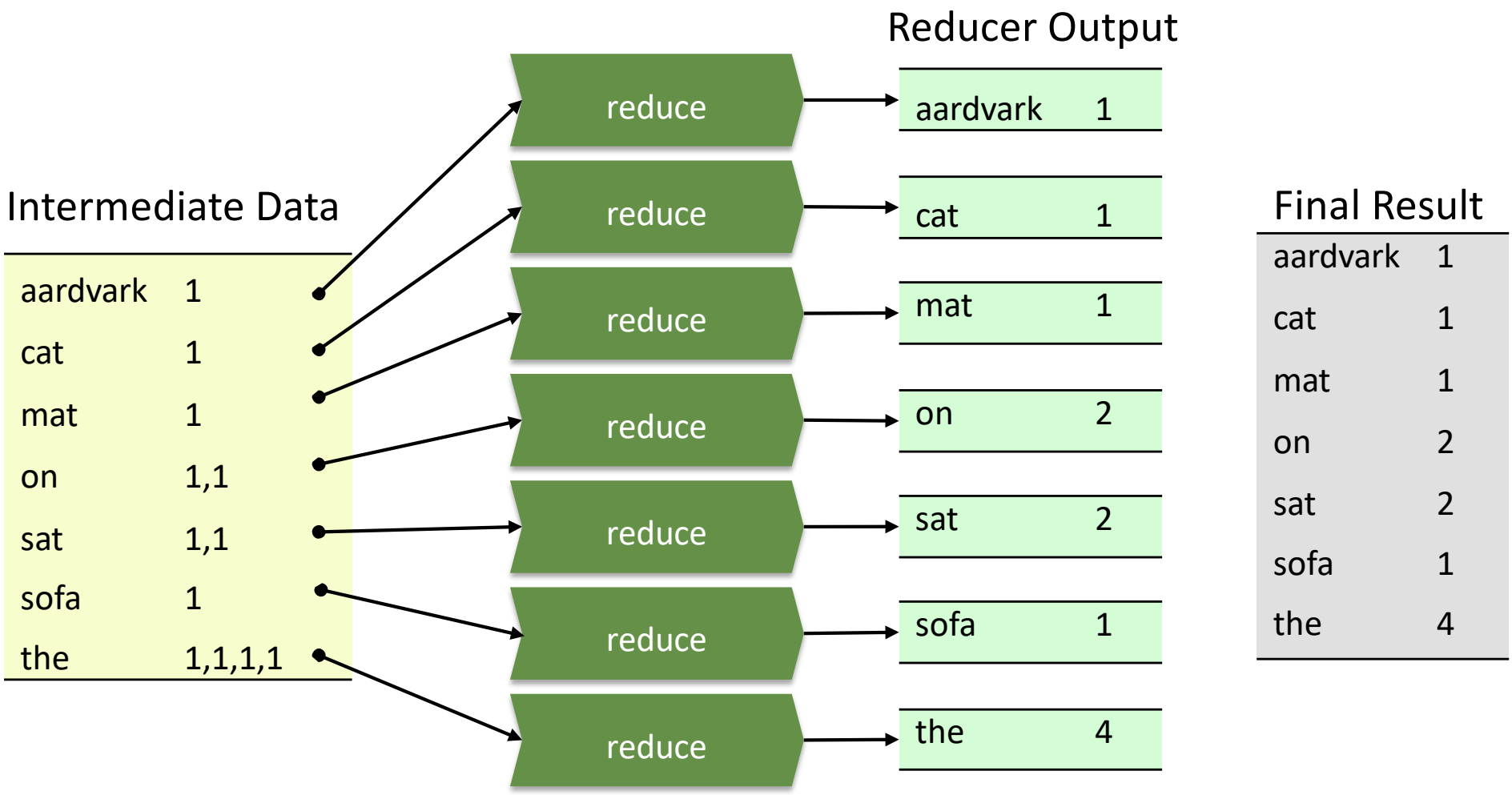
the	1
cat	1
sat	1
on	1
the	1
mat	1
the	1
aardvark	1
sat	1
on	1
the	1
sofa	1



Intermediate Data

aardvark	1
cat	1
mat	1
on	1,1
sat	1,1
sofa	1
the	1,1,1,1

Example: SumReducer



Mappers Run in Parallel

- **Hadoop runs Map tasks on the slave node where the block is stored (when possible)**
  - Many Mappers can run in parallel
  - Minimizes network traffic

Input Data

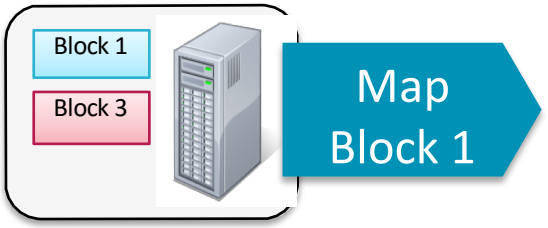
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa.
Vestibulum lacinia arcu eget nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc.
Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem…

HDFS Blocks

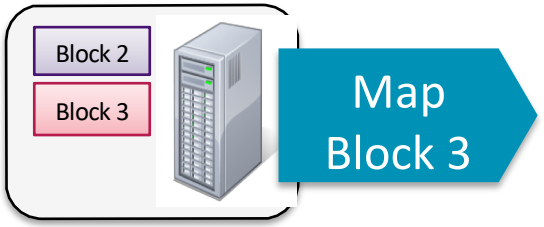
Lorem ipsum dolor sit amet, consec tetur adipisc ing elit. Integer…

Sed pretium blandit orci. Ut eu diam at pede susci pit soda les…

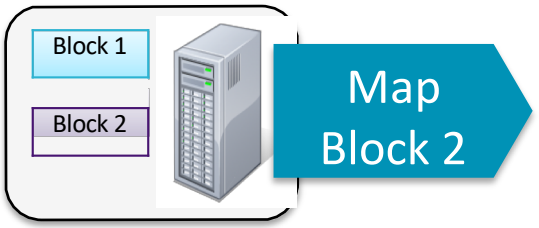
Aenean quam. In sc elerisque sem at d olor. Maecenas mat tis. Sed con…



a	1,1,1,…
ac	1,1,1,1,1….
accumsan	1,1
ad	1,1,1,1….
adipiscing	1,1,1,1 al
iquet	1,1,1,1….
…	



a	1,1,1,1….
ac	1,1,1,1,1…
ad	1,1,1,1,1…
aliquam	1,1,1
aliquet	1
auctor	1,1,1,1…
…	



a	1,1,1,…
ad	1,1,1,1….
aliquam	1.1,1
amet	1,1,1,1
blandit	1,1,1
…	

More Specifically

- Input: a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** → <k', v'>*
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every *(k,v)* pair
  - **Reduce(k', <v'>*)** → <k', v''>*
    - All values *v'* with same key *k'* are reduced together and processed in *v'* order
    - There is one Reduce function call per unique key *k'*

Word Count Using MapReduce

```
map(key, value):
// key: document name; value: text of the document
   for each word w in value:
       emit(w, 1)

reduce(key, values):
// key: a word; value: an iterator over counts
       result = 0
       for each count v in values:
             result += v
       emit(key, result)
```

Map-Reduce: Environment

Map-Reduce environment takes care of:

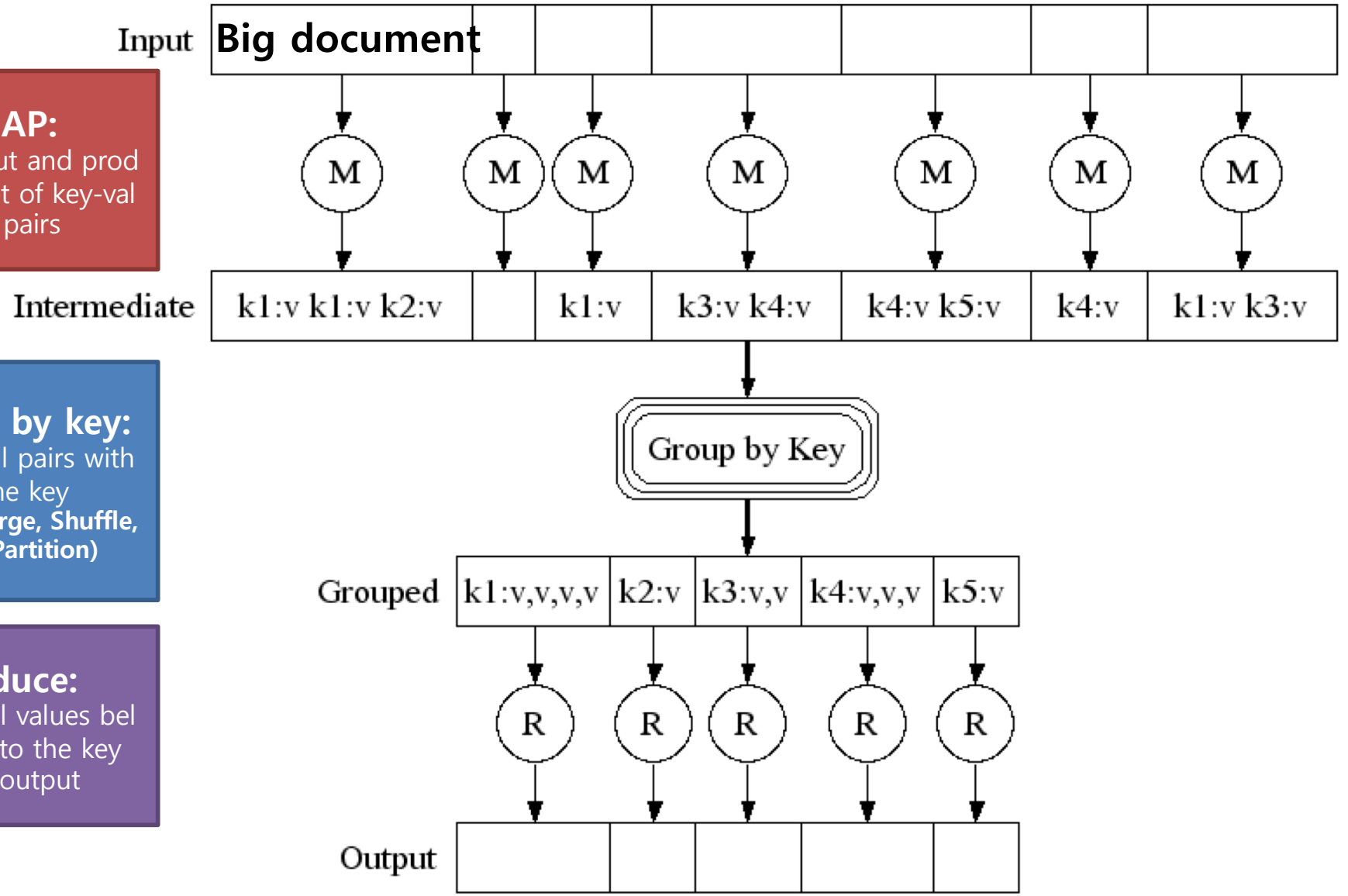
- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the group by key step
- Handling machine failures
- Managing required inter-machine communication

Map-Reduce: A diagram

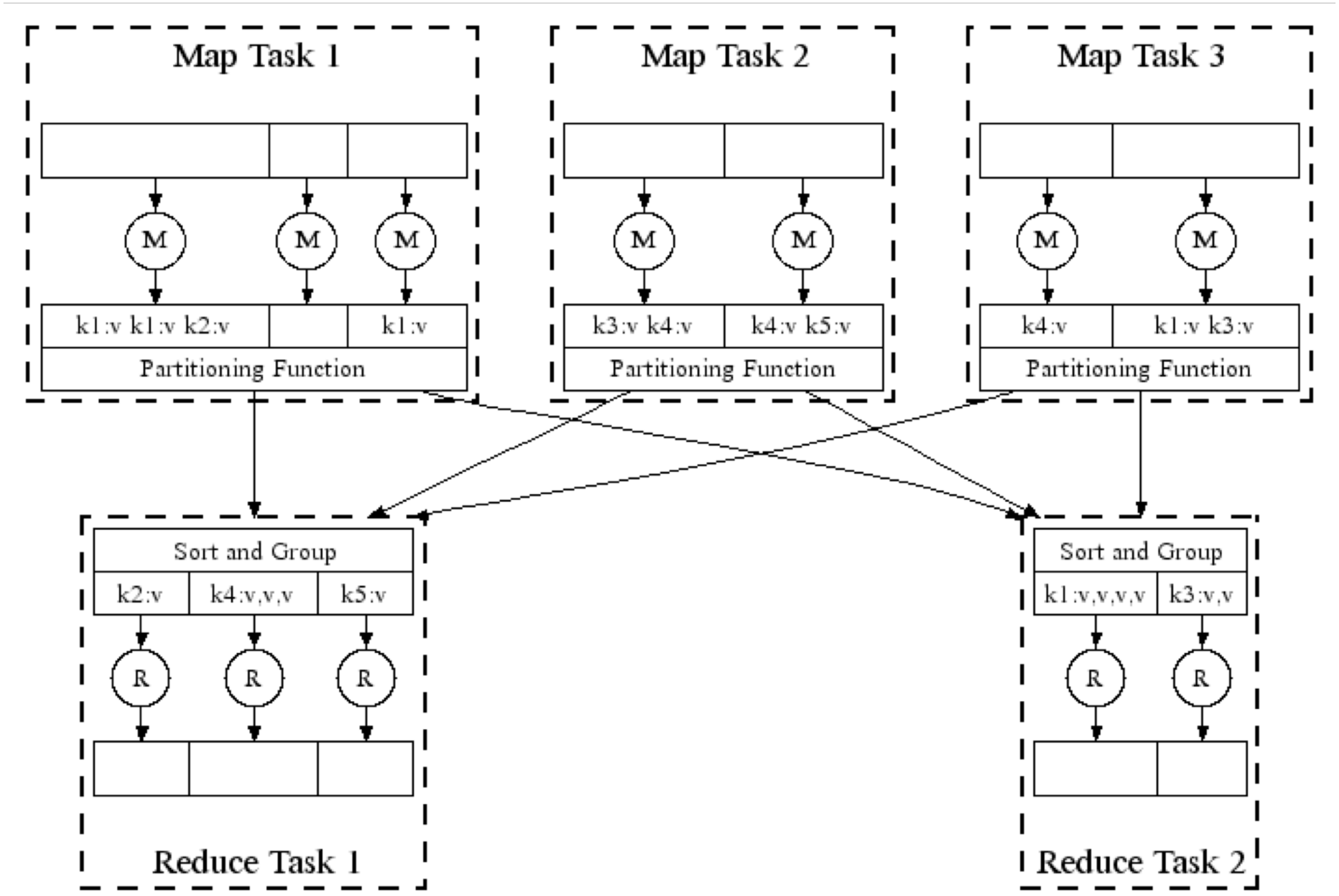
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



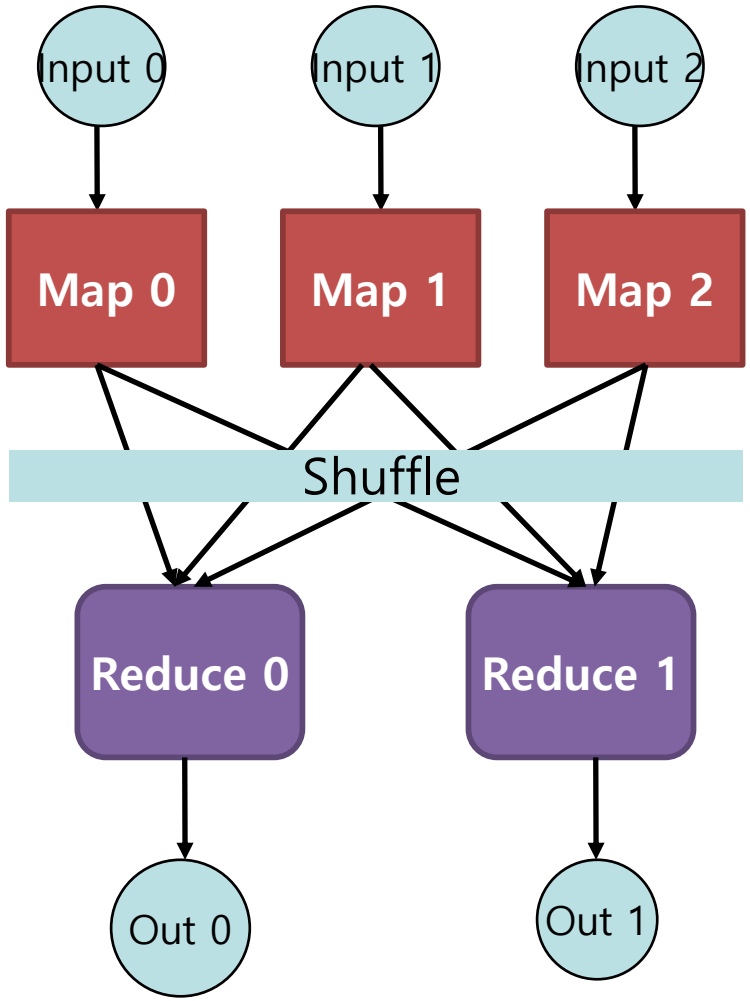
Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

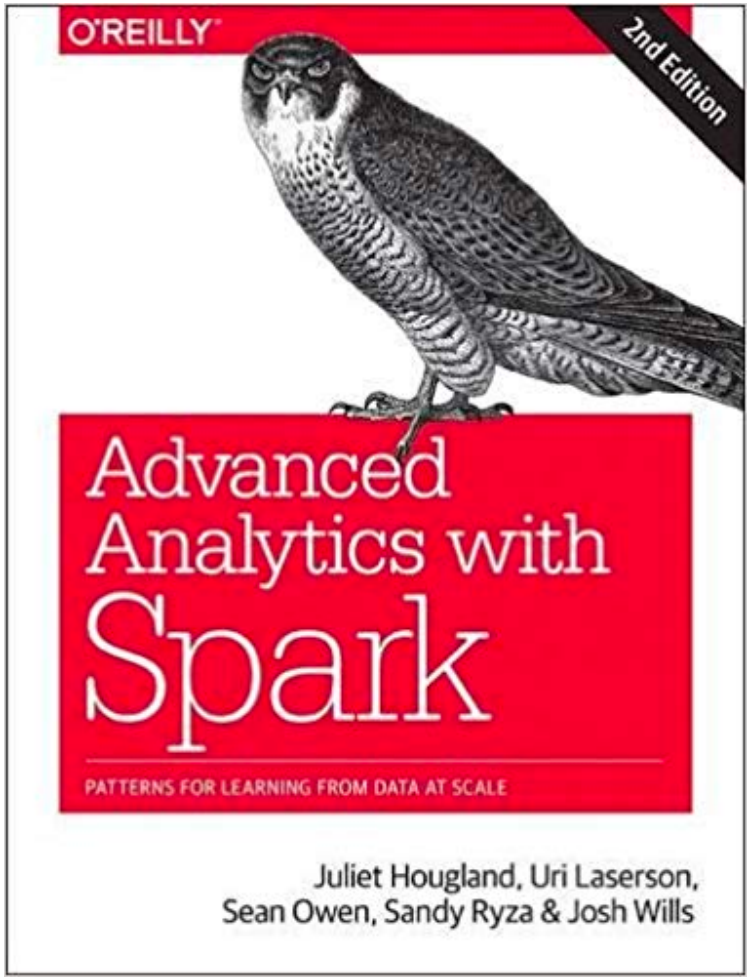
Map-Reduce

- Programmer specifies:
  - Map and Reduce and input files
- Workflow:
  - Read inputs as a set of key-value-pairs
  - Map transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k' are sent to the same reduce
  - Reduce processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work



교재

- Advanced Analytics Spark 2nd Edition
- https://www.amazon.com/Advanced-Analytics-Spark-Patterns-Learning/dp/1491972955/

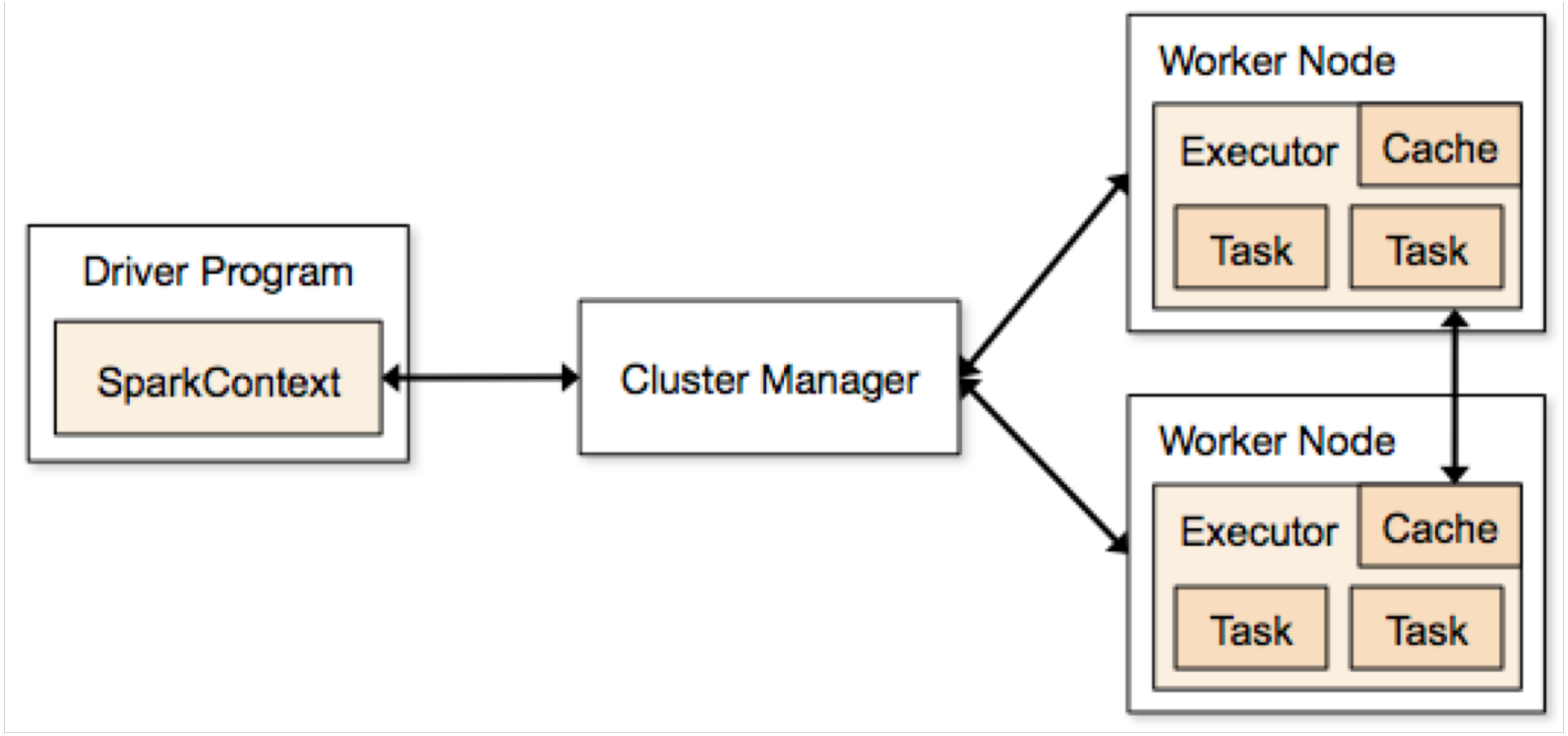


Spark

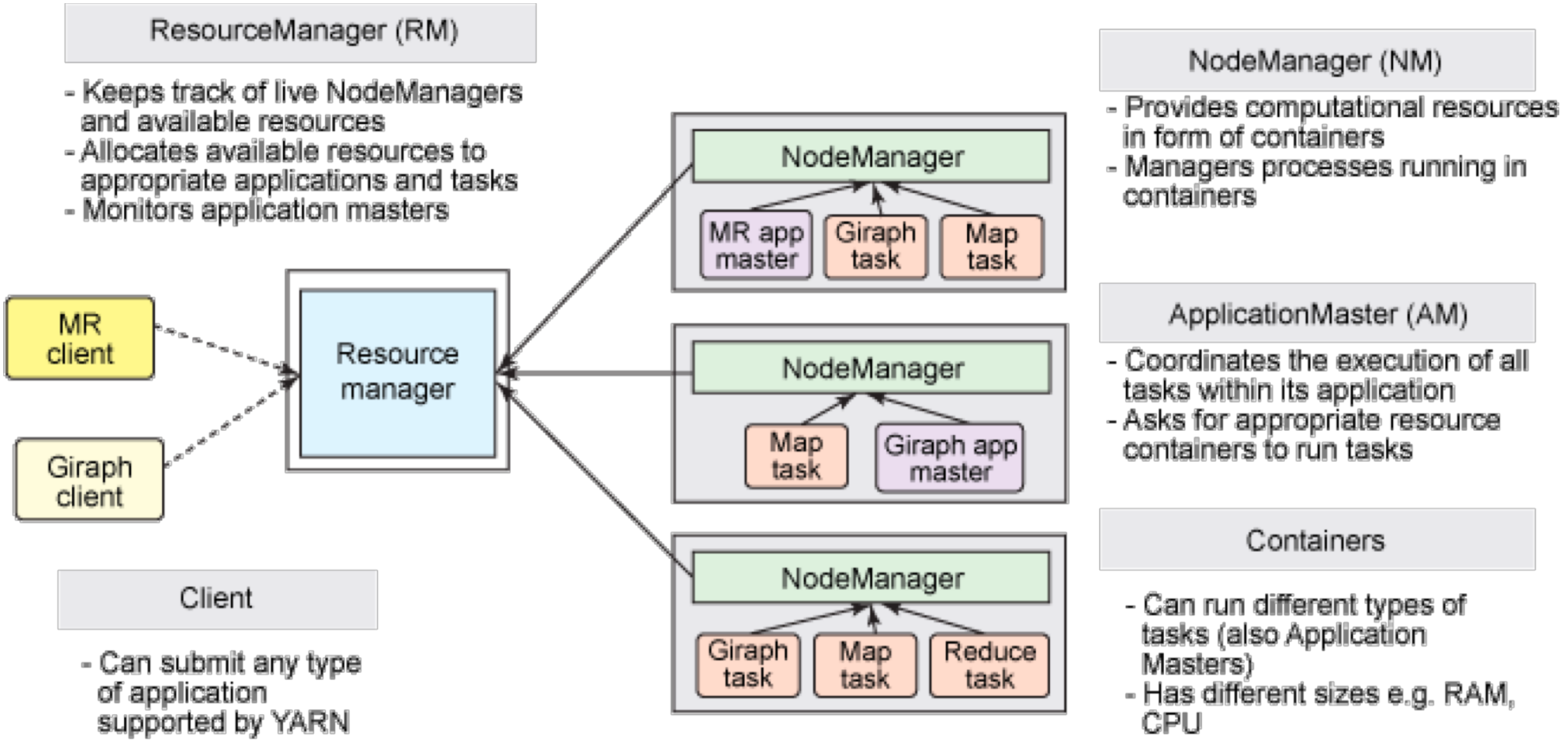
- Another way to express computations for a cluster.

- In contrast to MapReduce:
  - Number of stages is up to the programmer (not just "map to key/value pairs" then "reduce using keys").
  - Spark really likes keeping things in memory.
  - Runs on YARN or locally (or standalone or on Mesos or EC2).

Spark Architecture



Spark Architecture



https://www.ibm.com/developerworks/library/bd-yarn-intro/

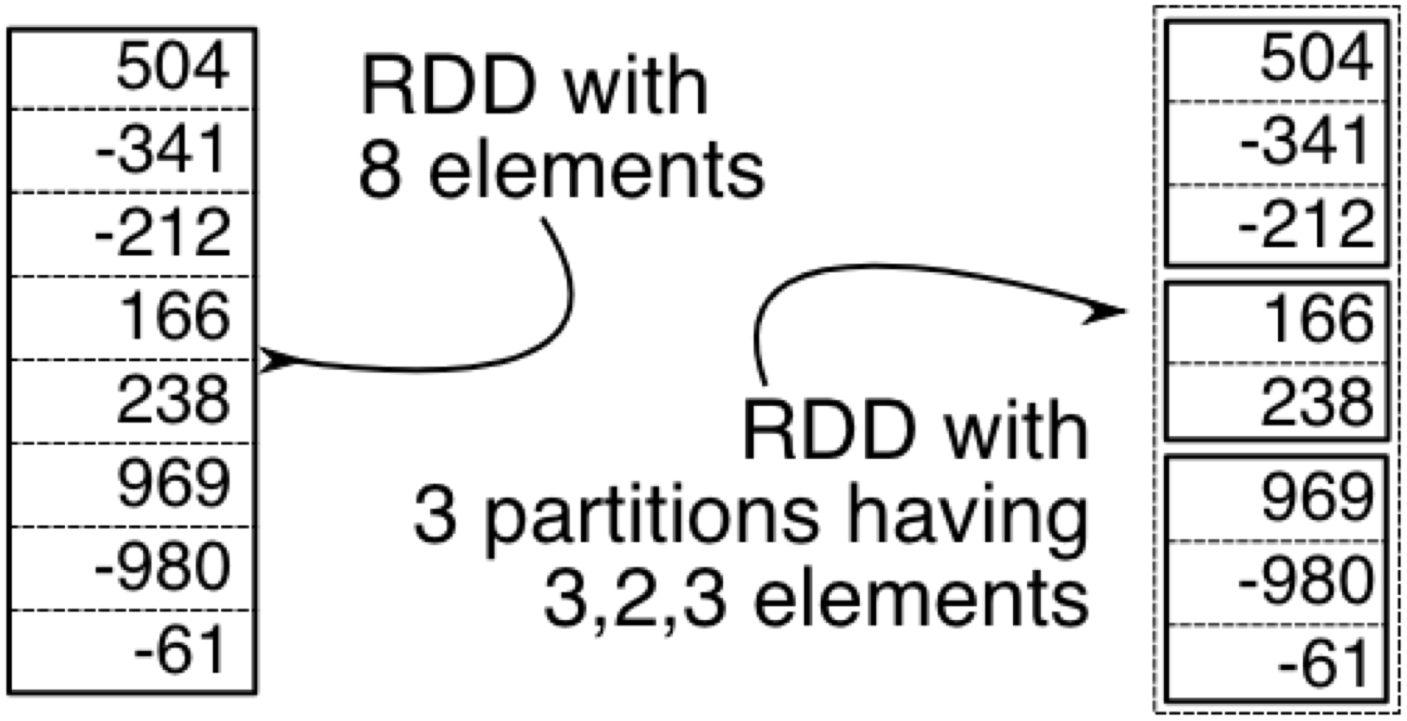
RDD

- The basic class that represents data in Spark is the Resilient Distributed Dataset. Basically, a collection of data (rows, elements, or however you think of them).

- Can be partitioned across multiple nodes/processes. Operations can be done on partitions in parallel.

- Are immutable: values in a particular RDD can't change (but can be used to compute a new RDD).

RDD


504
-341
-212
166
238
969
-980
-61
RDD with
8 elements
RDD with
3 partitions having
3,2,3 elements
504
-341
-212
166
238
969
-980
-61

RDD Operations

- Operations that return a new RDD: (transformations)
  - .map(f): result of applying f to each element.
  - .filter(f): elements where f returns True.

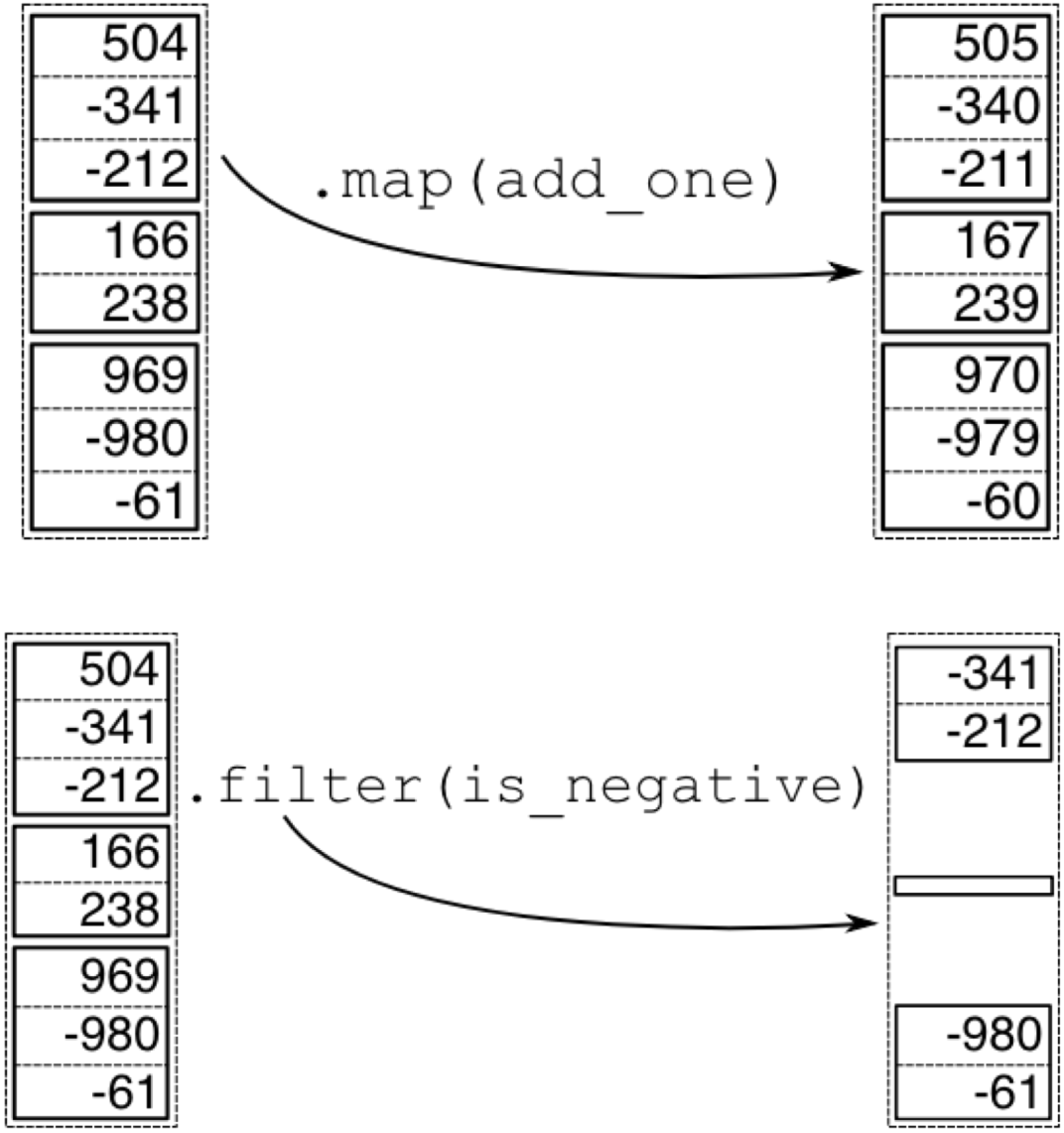
- Operations that return a Python value: (actions)
  - .first
  - .take(10)
  - .collect()

- .max()/.min()/.mean()/.count(): largest/smallest/average/number of values.

- Operations that do something: (actions)
  - .saveAsTextFile(path): write one element per line.

- https://spark.apache.org/docs/2.3.1/api/python/pyspark.html#pyspark.RDD

.map & .filter


504
-341
-212
166
238
969
-980
-61
.map(add_one)
505
-340
-211
167
239
970
-979
-60
504
-341
-212
166
238
969
-980
-61
.filter(is_negative)
-341
-212
-980
-61

Lazy Evaluation

- Operations on RDDs are lazily evaluated.

- Think of RDD objects in Python as a plan for calculations that can be done in the future.
- As you build an RDD (from input or previous RDDs), the plan (actually a "directed acyclic graph, DAG, of calculations") is constructed.

- The plan isn't evaluated until you actually do something with the results…

- Do something with the results could be:
  - some_pos_nums.collect().collect() turns the RDD into a Python list.
  - some_pos_nums.saveAsTextFile('/tmp/output')
  - s = some_pos_nums.sum()

Chaining Calculation

```
lines = sc.textFile('/tmp/inputs')
numbers = lines.map(int)
pos_nums = numbers.filter(lambda n: n > 0)
some_pos_nums = pos_nums.sample(fraction=0.01,
        withReplacement=False)
print(some_pos_nums.take(5))
```

```
print(sc.textFile('/tmp/inputs')
    .map(int).filter(lambda n: n > 0)
    .sample(fraction=0.01, withReplacement=False)
    .take(5))
```

Word Count

```
>>> wordcounts = sc.textFile('hdfs://ubuntu1:54310/user/dev/gutenberg') \
        .map( lambda x: x.replace(',',' ').replace('.',' ').replace('-',' ').lower()) \
        .flatMap(lambda x: x.split()) \
        .map(lambda x: (x, 1)) \
        .reduceByKey(lambda x,y:x+y) \
        .map(lambda x:(x[1],x[0])) \
        .sortByKey(False)
>>> wordcounts.take(10)
[(500662, u'the'), (331864, u'and'), (289323, u'of'), (196741, u'to'),
 (149380, u'a'), (132609, u'in'), (100711, u'that'), (92052, u'i'),
 (77469, u'he'), (72301, u'for')]
```

Data Frame

- Python 의 Pandas, R의 data.frame 과 같이 구조적인 정보도 같이 가지고 있다
- SQL과 유사한 기능 수행

```
parsed = spark.read.option("header", "true") \
                   .option("nullValue", "?") \
                   .option("inferSchema", "true") \
                   .csv('linkage_csv')
```

```
parsed.groupBy("is_match") \
    .count() \
    .orderBy("count", ascending=False) \
    .show()
```

```
+--------+-------+
|is_match|  count|
+--------+-------+
|   false|5728201|
|    true|  20931|
+--------+-------+
```