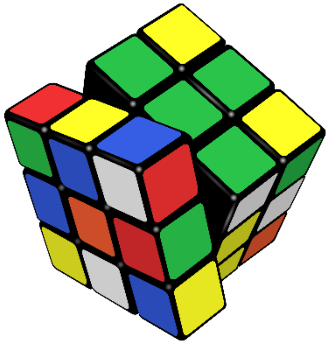
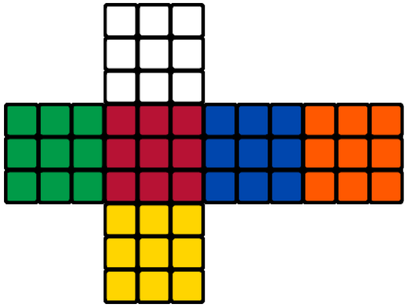


Big Data
Engineering
For Analytics

Big Data Ingestion - Tools and Practices

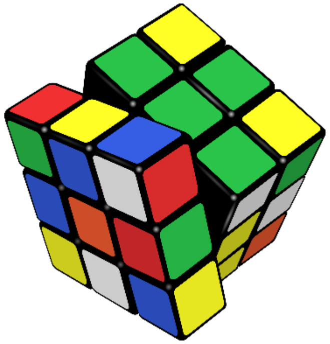
Suria R Asai
suria@nus.edu.sg
NUS-ISS

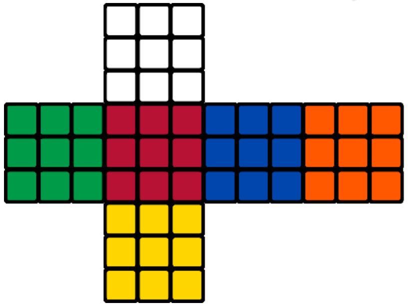


Total 72 Slides

Learning Objectives

- Design Ingestion Layer
- Understand and use Hadoop Distributed File System.
- Understand and use tool: Apache Sqoop
- Understand and use tool: Apache Flume





Big Data Engineering For Analytics

Introduction to Data Ingestion

Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes. Science is not about tools. It is about how we use them, and what we find out when we do.

Edsger Dijkstra

Definition

- Data ingestion can be done via manual, semi-automatic, or automatic methods.

  Data ingestion means the process of getting the data into the data system that we are building or using.

  1. How many data sources are there?
  2. How many large data items are available?
  3. Will the number of data sources grow over time?
  4. What is the rate at which data will be consumed?

- When it comes to data ingestion, developers like to create a bunch of policies, called **ingestion policies**, that guide the handling of errors during the data ingestion, as well as the data incompleteness, and so on.

Data sources

- **Data sensors**: These are thousands of sensors, producing data continuously.
- **Machine Data**: Produces data which should be processed in near real time for avoiding huge loss.
- **Telco Data**: CDR data and other telecom data generates high volume of data.
- **Healthcare system data**: Genes, images, ECR records are unstructured and complex to process.
- **Social Media**: Facebook, Twitter, Google Plus, YouTube, and others get a huge volume of data.
- **Geological Data**: Semiconductors and other geological data produce huge volumes of data.
- **Maps**: Maps have a huge volume of data, and processing data is a challenge in Maps.
- **Aerospace**: Flight details and runway management systems produce high-volume data and processing in real time.
- **Astronomy**: Planets and other objects produce heavy images, which have to be processed at a faster rate.
- **Mobile Data**: Mobile generates many events and a huge volume of data at a high velocity rate.

These are just some domains or data sources that produce data in Terabyte's or Exabyte's. **Data ingestion is critical and can make or break a system.**

Common Challenges In Ingesting

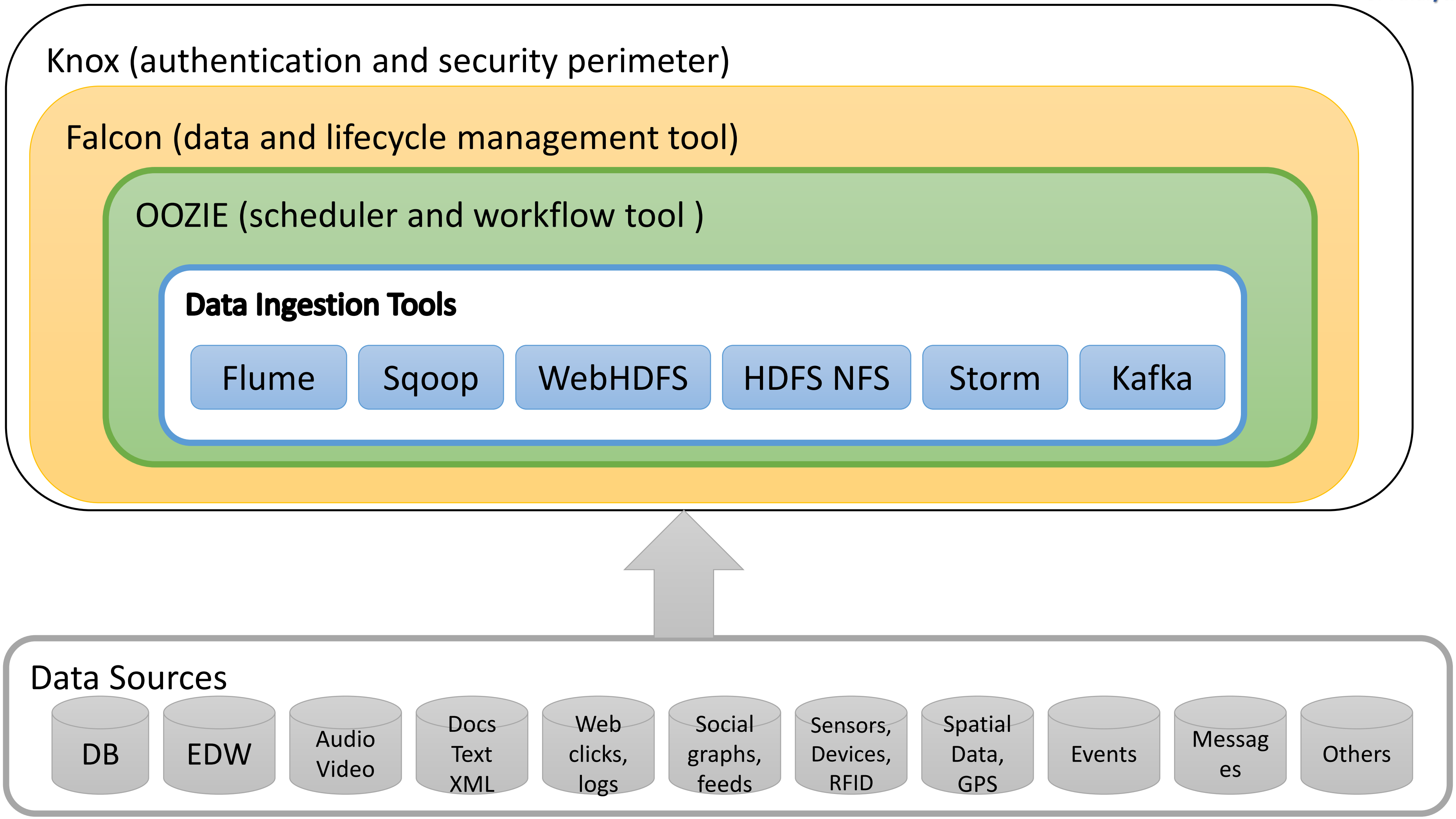
- Prioritizing each data source load
- Tagging and indexing ingested data.
- Validating and cleansing the ingested data.
- Transforming and compressing before ingestion.
- New data sources tend to deliver data at varying speed and frequencies; example: streaming and real time ingestion.
- With a wider range of data sources becoming relevant for the enterprise, the volume of data to be ingested has grown manifold over the years.
- Another challenge that applies to incremental data-ingestion processes is the detection and capture of changed data.

Stages In The Ingestion Process

Input → Filter → Enrich → Process → Segregate → Output

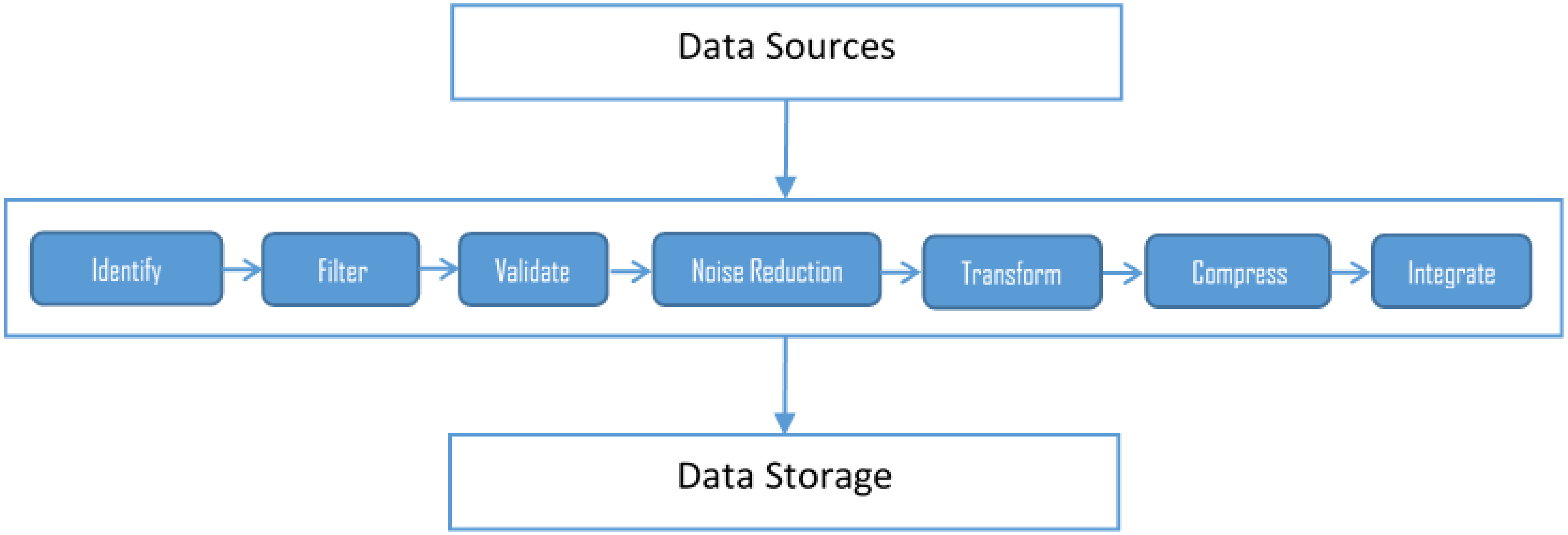
1. **Input:** Discover and fetch the data for ingestion. The discovery of data may be from File System, messaging queues, web services, sensors, databases or even the outputs of other ingestion apps.
2. **Filter:** Analyse the raw data and identify the interesting subset. The filter stage is typically used for quality control or to simply sample the dataset or parse the data.
3. **Enrich:** Plug in the missing pieces in the data. This stage often involves talking to external data sources to plug in the missing data attributes. Data may be transformed from a specific form into a form to make it suitable for downstream processes.
4. **Process** – This stage is meant to do some lightweight processing to either further enrich the event or transform the event from one form into another. The process stage usually computes using the existing attributes of the data and at times using external systems.
5. **Segregate** – Often times before the data is given to downstream systems, it makes sense to bundle similar data sets together. While this stage may not always be necessary for compaction, segregation does make sense most of the time.
6. **Output** – Outputs are almost always mirrors of inputs in terms of what they can do and are as essential as inputs. While the input stage requires fetching the data, the output stage requires resting the data – either on durable storage systems or other processing systems.

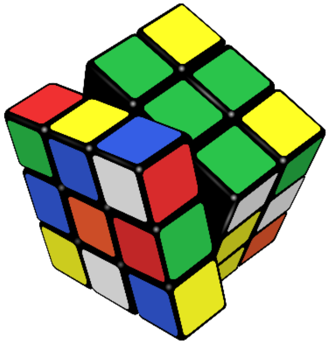
An Example of Data Ingestion Tools

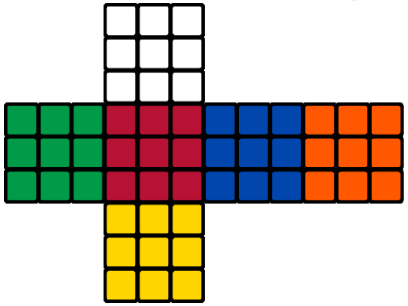


Common Challenges in Ingestion Layer

- Multiple data source load and prioritization
- Ingested data indexing and tagging
- Data validation and cleansing
- Data transformation and compression







Hadoop Distributed File System (HDFS)

"Talk is cheap. Show me the code."

— Linus Torvalds

Examples of Big Data Sources

- **Monitoring sensors**: Climate or ocean wave monitoring sensors generate data consistently and in a good size, and there would be more than millions of sensors that capture data.
- **Content(Posts to social media sites)**: Social media websites such as Facebook, Twitter, and others have a huge amount of data in petabytes.
- **Media(Digital pictures and videos posted online)**: Websites such as YouTube, Netflix, and others process a huge amount of digital videos and data that can be petabytes.
- **Transaction records of online purchases**: E-commerce sites such as eBay, Amazon, Flipkart, and others process thousands of transactions on a single time.
- **Server/application logs**: Applications generate log data that grows consistently, and analysis on these data becomes difficult.
- **CDR (call data records)**: Roaming data and cell phone GPS signals to name a few.
- **Scientific and Research Data**: Science, genomics, biogeochemical, biological, and other complex and/or interdisciplinary scientific research.

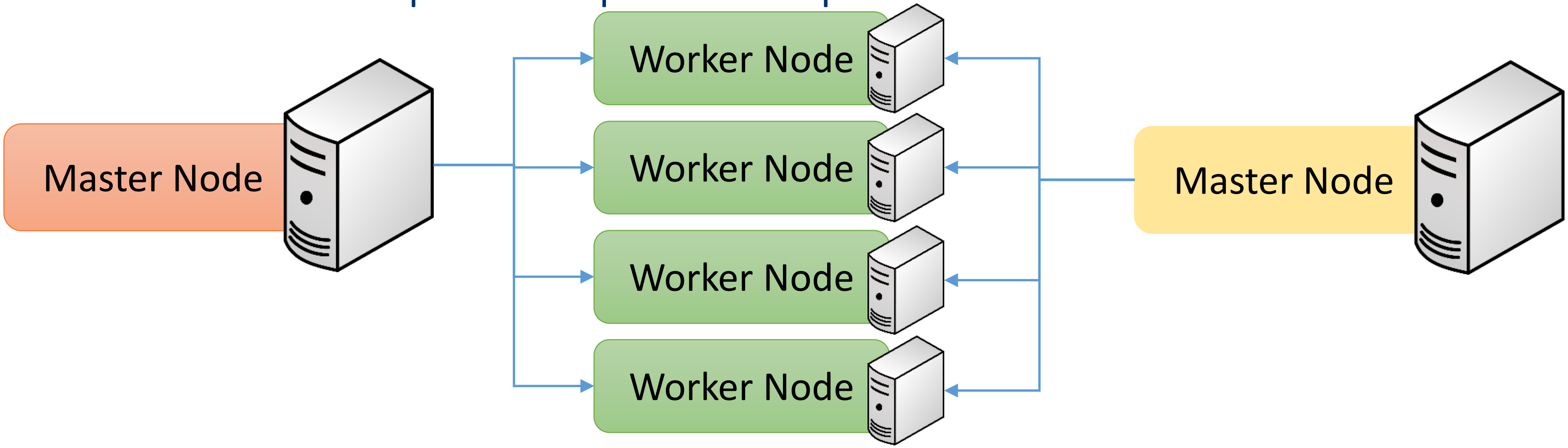
Big Data Engineering For Analytics

DFS Structure

- A **Service** – software entity running on one or more machines and providing a particular type of function to a priori unknown clients.
- A **Server** – service software running on a single machine.
- A **Client** – process that can invoke a service using a set of operations that forms its client interface.
- A **Client Interface** for a file service is formed by a set of primitive file operations (create, delete, read, write). Client interface of a DFS should be transparent, i.e., not distinguish between local and remote files.

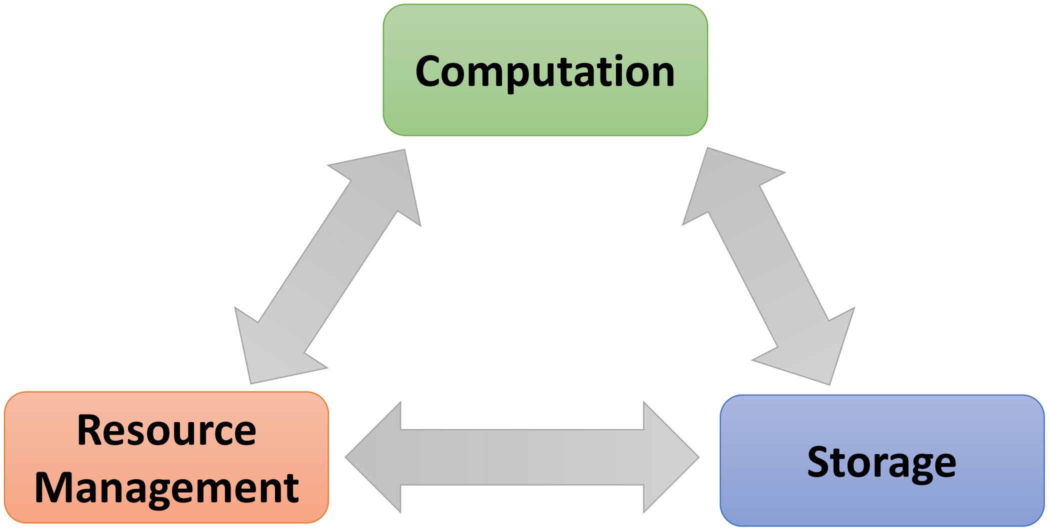
Hadoop Cluster Terminology

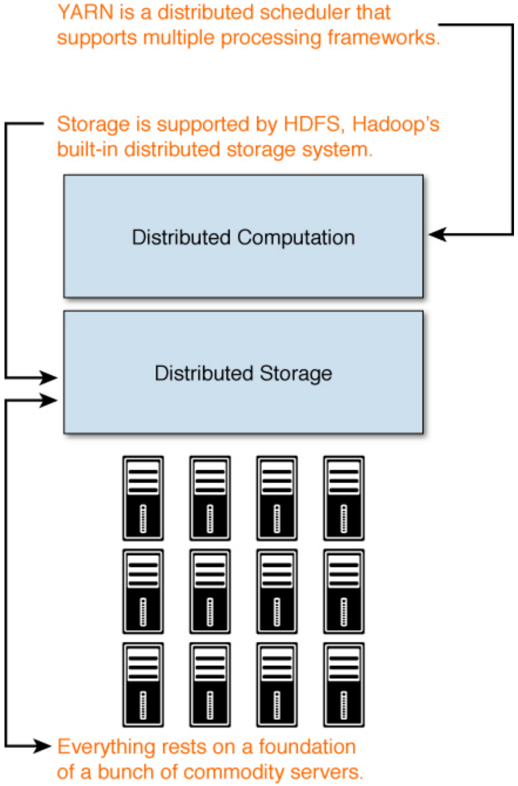
- **A *cluster* is a group of computers working together**
  - ➢Provides data storage, data processing, and resource management
- **A *node* is an individual computer in the cluster**
  - ➢*Master* nodes manage distribution of work and data to *worker* nodes
- **A *daemon* is a program running on a node**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

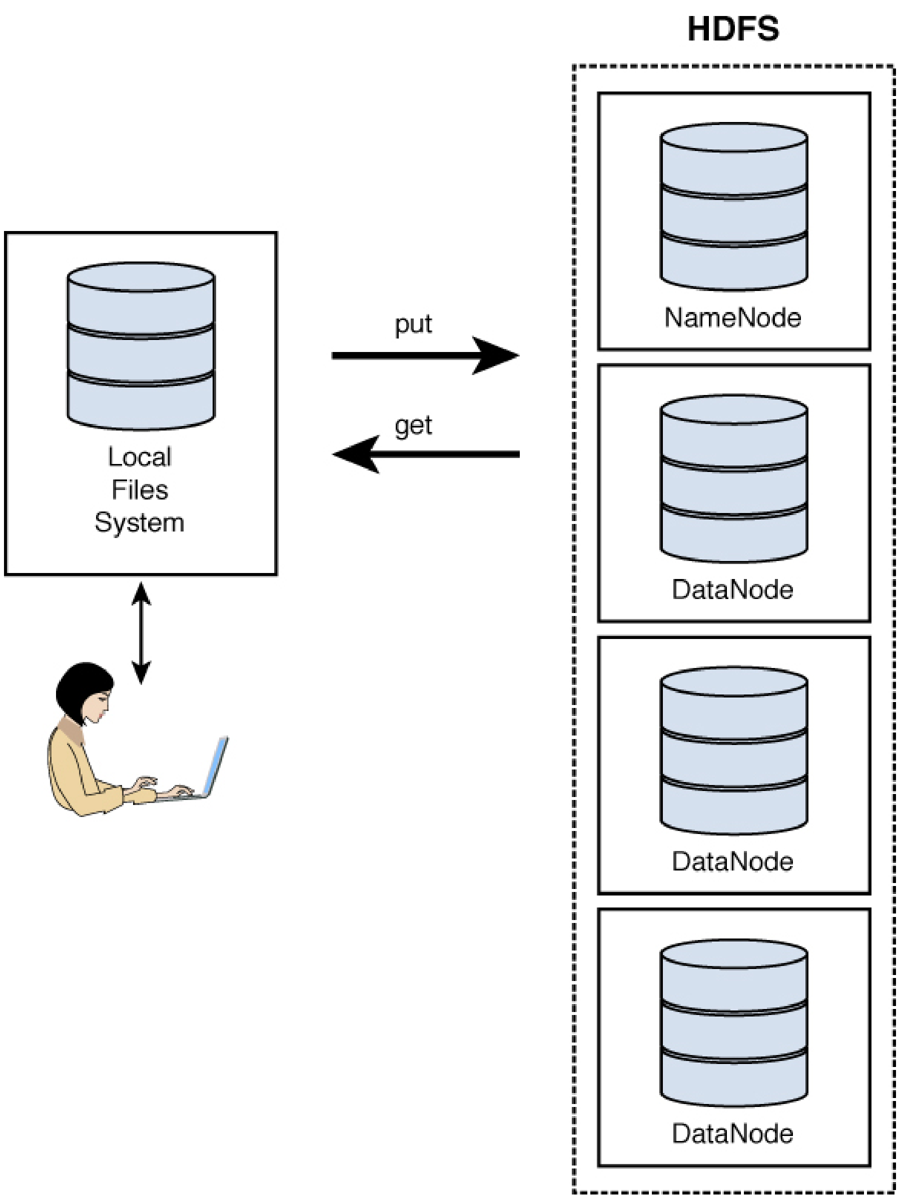


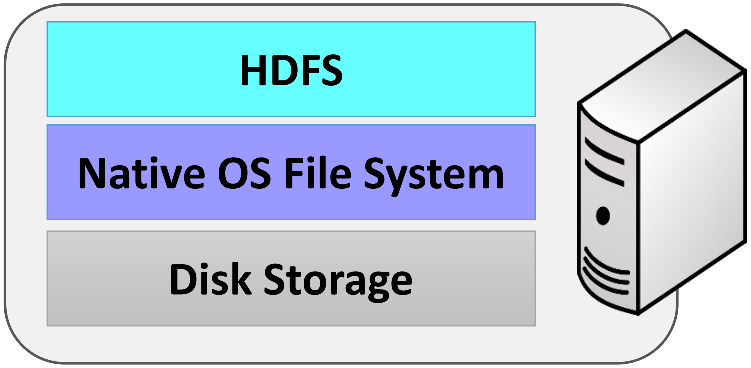
Hadoop Compute Cluster Components

Three main components of the Hadoop compute cluster works together to provide distributed data processing






HDFS
Local Files System
put
get
NameNode
DataNode
DataNode
DataNode


HDFS
Native OS File System
Disk Storage


Store
Process
Hadoop = HDFS + MapReduce

Features of HDFS

Big Data Engineering For Analytics

- **Scalability**:
  - ➢Scalable to petabytes or even more,
  - ➢Flexible enough to add or remove nodes to achieve scalability.
- **Reliability and Fault Tolerance**:
  - ➢replicates the data to a configurable parameter for reliability,
  - ➢increases the fault tolerance and data access.
- **Data Coherency**:
  - ➢**WORM** (**write once, read many**) model for data coherency and high throughput.
- **Hardware Failure Recovery**:
  - ➢has a good failure recovery processes even for commodity hardware,
  - ➢failover processes to recover the data and handle hardware failure recovery.
- **Portability**:
  - ➢portable on different hardware and software.
- **Computation closer to data**:
  - ➢Distributes data nearer to computation nodes - ideal for the MapReduce process.

HDFS Concepts

- HDFS is a file system written in Java
  - Based on Google's GFS
- Sits on top of a native file system
  - Such as ext3, ext4, or xfs
- Provides redundant storage for massive amounts of data
  - Using readily-available, industry-standard computers
- HDFS performs best with a 'modest' number of large files
  - Millions, rather than billions, of files
  - Each file typically 100MB or more
- Files in HDFS are 'write once'
  - No random writes to files are allowed
- HDFS is optimized for large, streaming reads of files
  - Rather than random reads

HDFS Architecture

- HDFS is managed by the daemon processes which are as follows:
  - **NameNode**: Master process
  - **DataNode**: Slave process
  - **Checkpoint NameNode or Secondary NameNode**: Checkpoint process
  - **BackupNode**: Backup NameNode

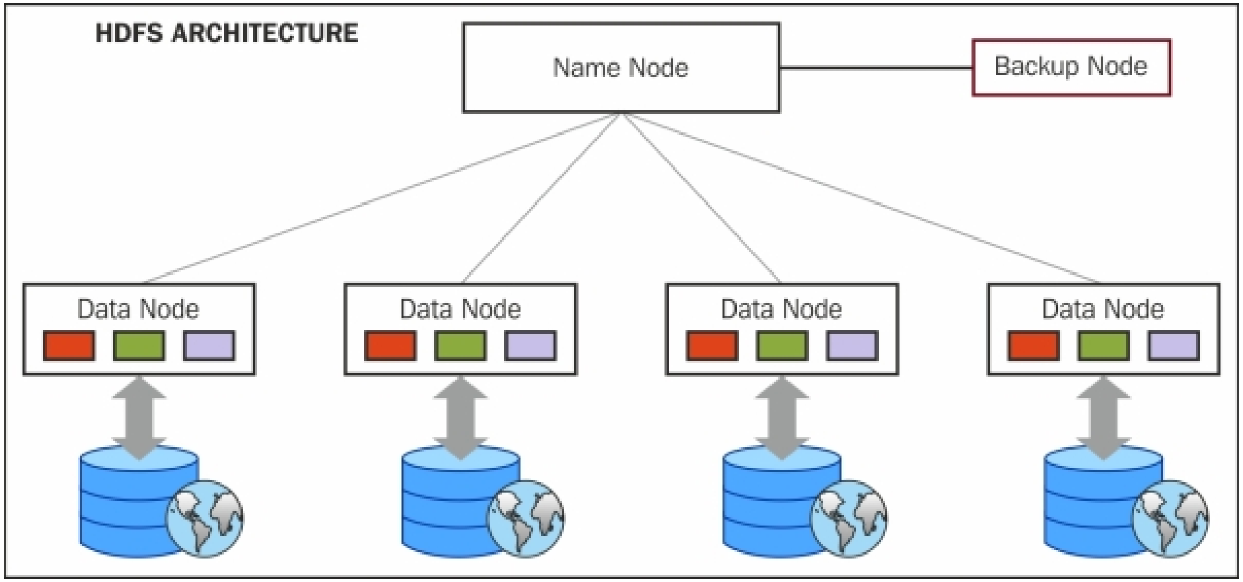


Image source: http://yoyoclouds.files.wordpress.com/2011/12/hadoop_arch.png

Roles of each component

- **Namenode**: coordinates all the operations related to storage; holds data regarding entire file system namespace and change logs to those files.
- **Datanode**: holds data and rresponsible for creating, deleting, and replicating data blocks, as assigned by Namenode.
- **Checkpoint** or secondary Namenode: maintains frequent data check points and manages failure.
- **Backupnode**: responsible for high availability.

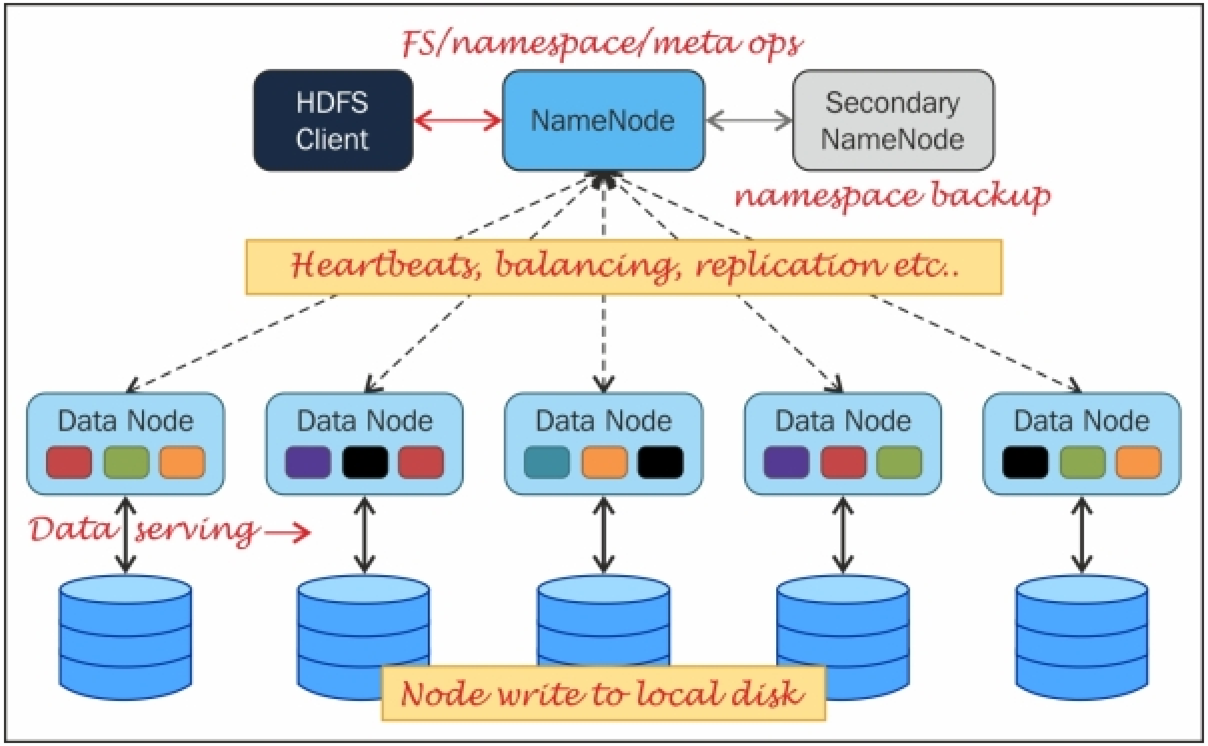
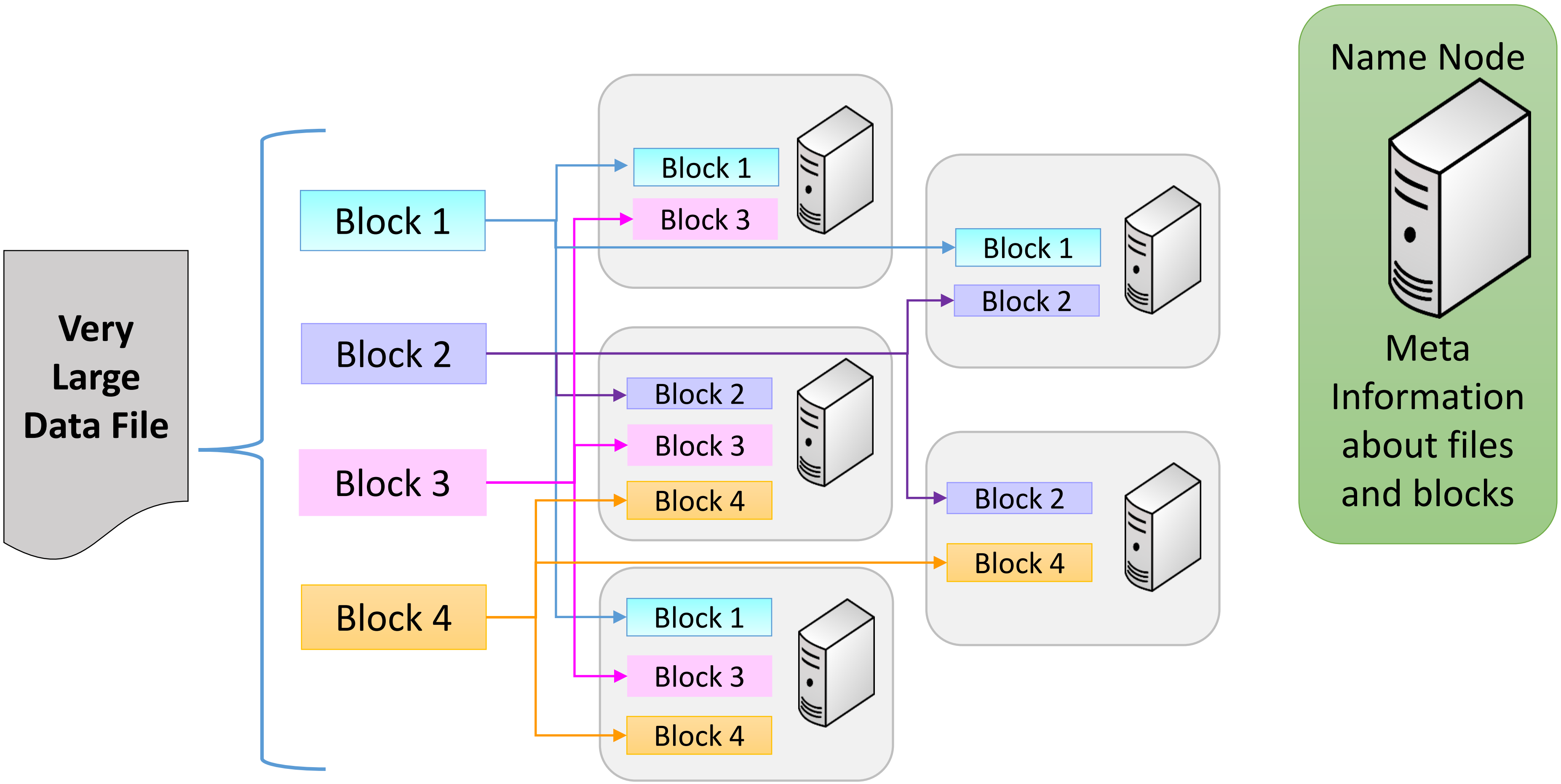


Image source: http://yoyoclouds.files.wordpress.com/2011/12/hadoop_arch.png

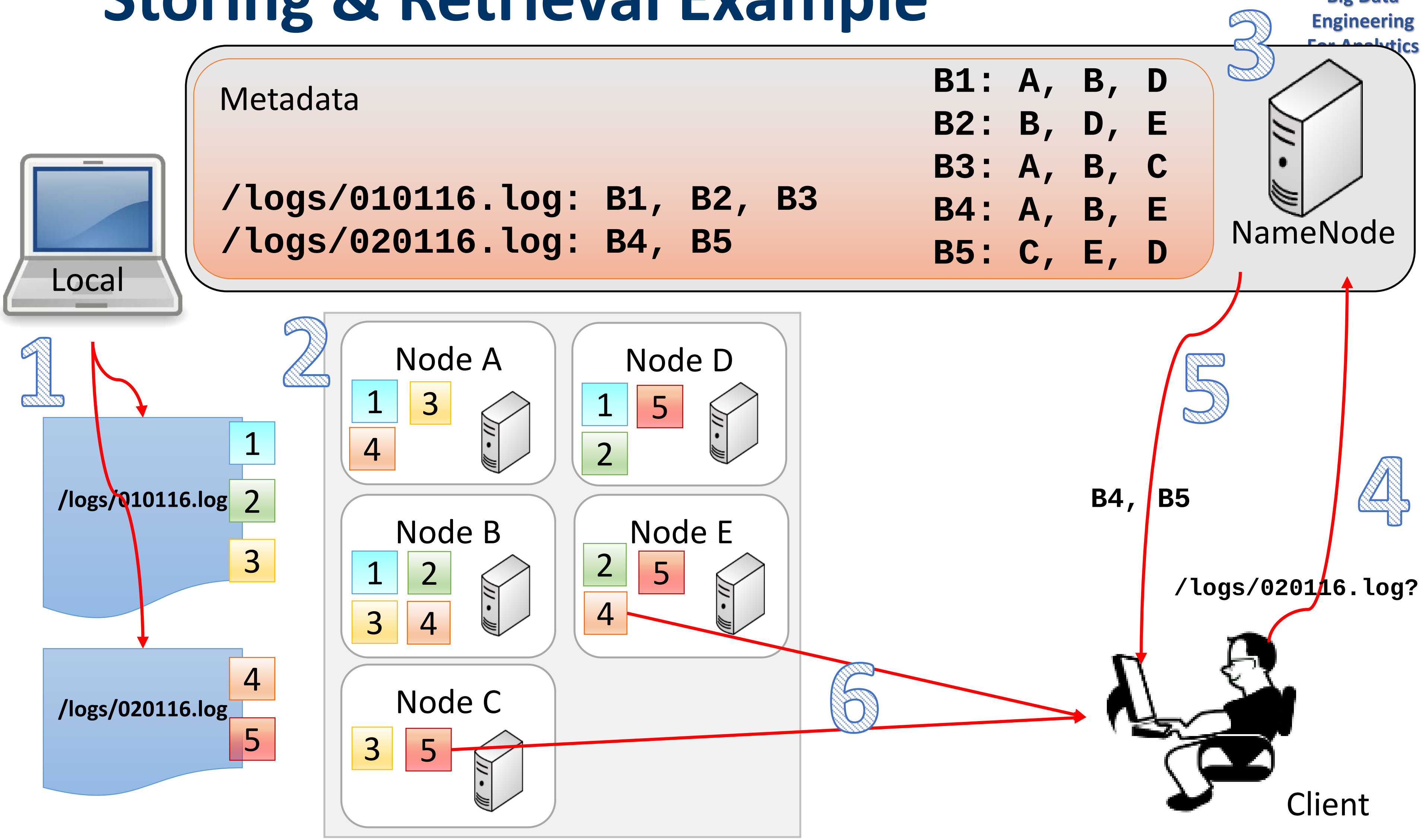
Storage Structure

- **Block**:
  - ➢ Files are divided in multiple blocks (configurable parameter & default block size is 128 MB)
  - ➢ Block size is high to minimize the cost of disk seek time (which is slower), leverage transfer rate (which can be high), and reduce the metadata size in Namenode for a file.
- **Replication**:
  - ➢ Each block of files divided earlier is stored in multiple Datanode (Configurable, default is 3)
  - ➢ Replication factor is the key to achieve fault tolerance.
  - ➢ The higher the number of the replication factor, the more the system is fault tolerant.
  - ➢ We have to balance the replication factor, not too high and not too low, so as to save storage space.

Storage Structure

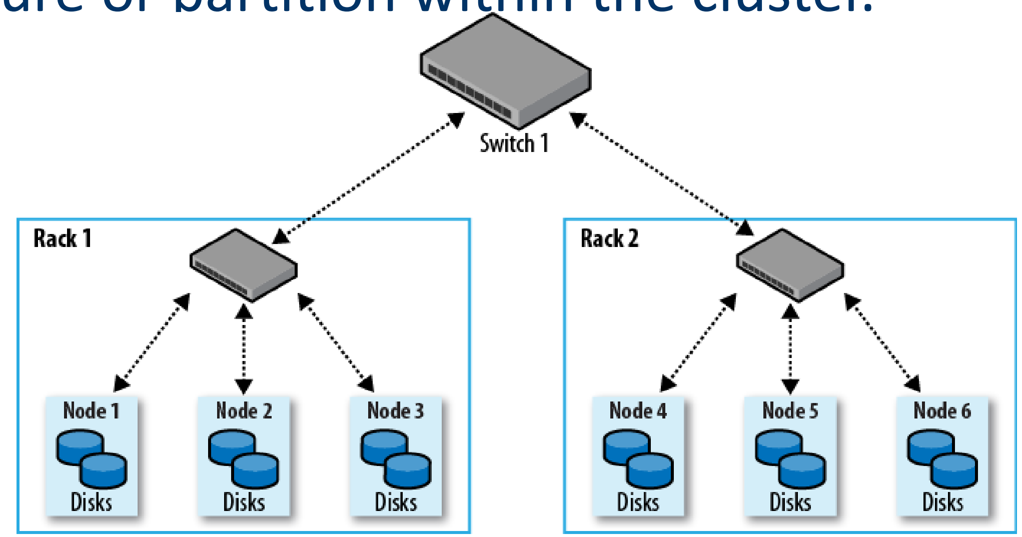


Storing & Retrieval Example



Rack Awareness

- Hadoop components are rack-aware.
- HDFS block placement will use rack awareness for fault tolerance by placing one block replica on a different rack.
- This provides data availability in the event of a network switch failure or partition within the cluster.



Access HDFS Commands

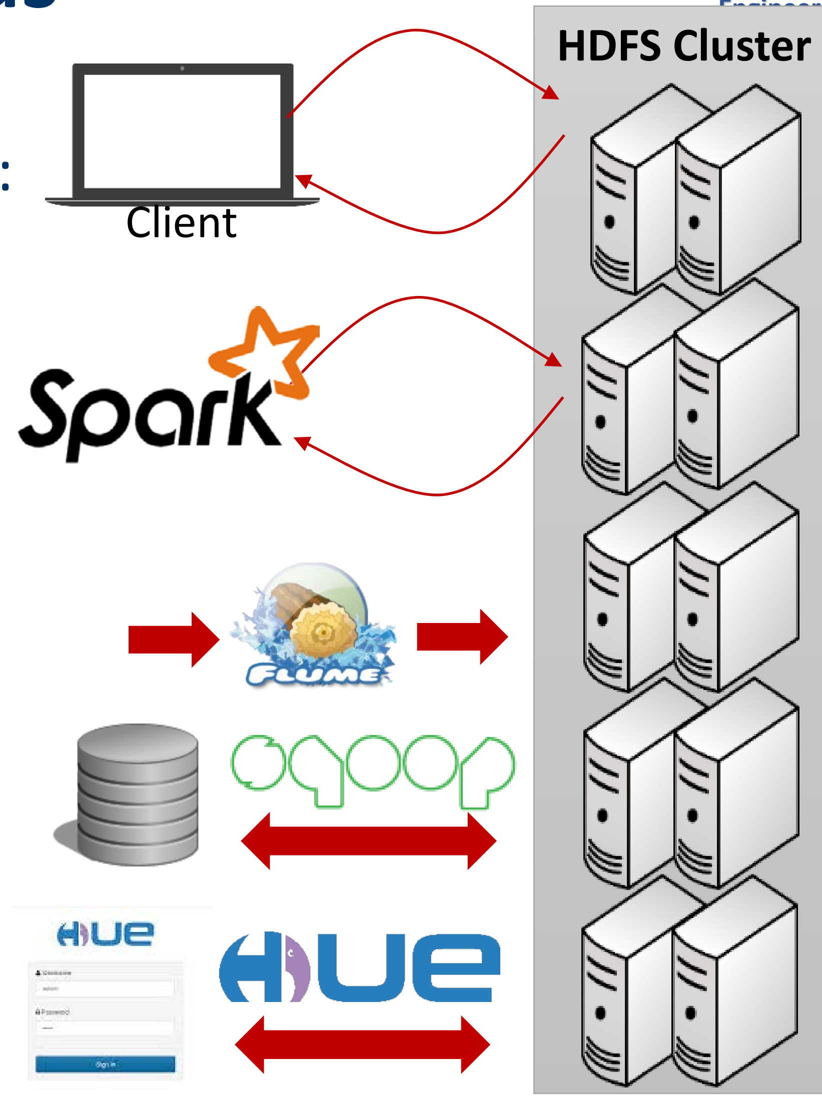
- From the command line – FsShell:

```
$ hdfs dfs
```

- In Spark – By URI,

```
hdfs://nnhost:port/file…
```

- Other programs
  - ➢Java API – Used by Hadoop MapReduce, Impala, Hue, Sqoop, Flume, etc.
  - ➢RESTful interface



HDFS Commands - 1

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

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

- Delete the directory input_old and all its contents

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

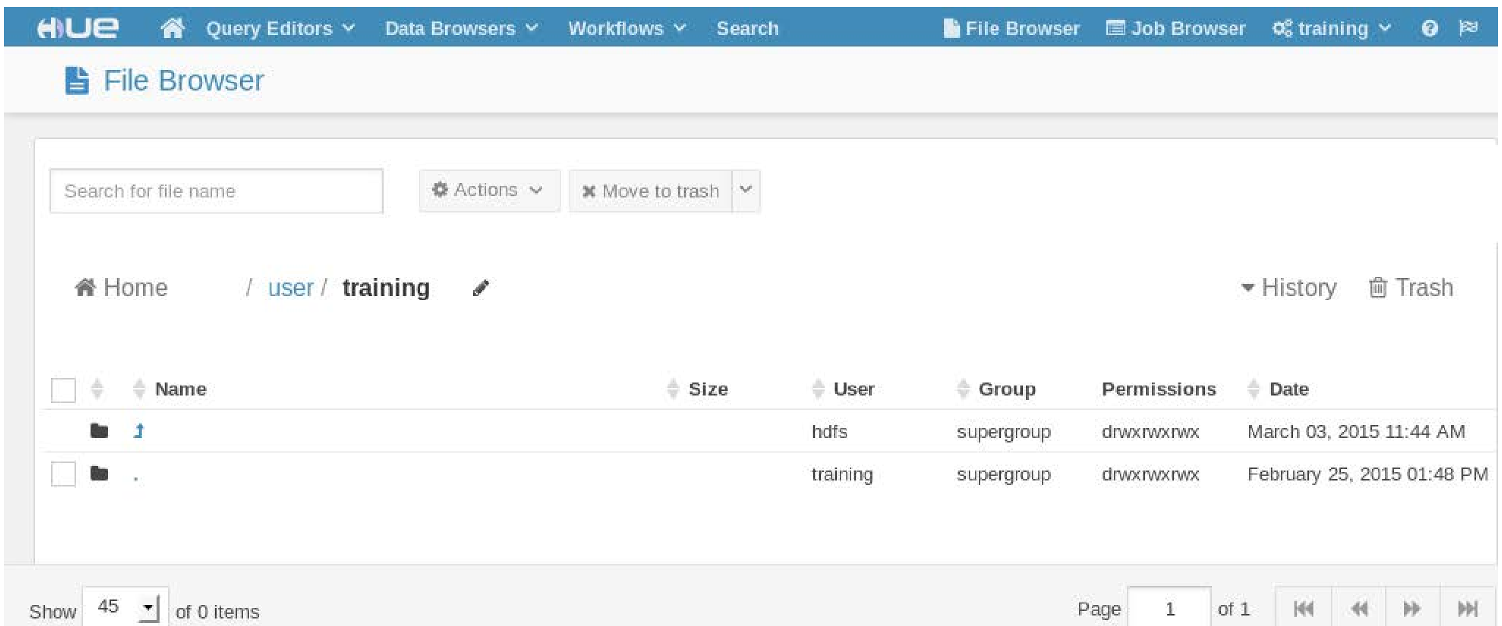
HDFS Recommendations

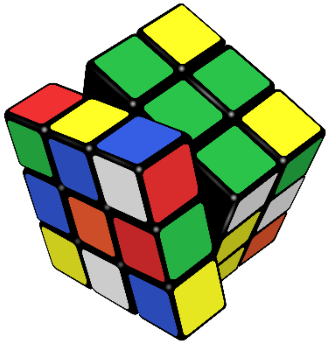
Big Data Engineering For Analytics

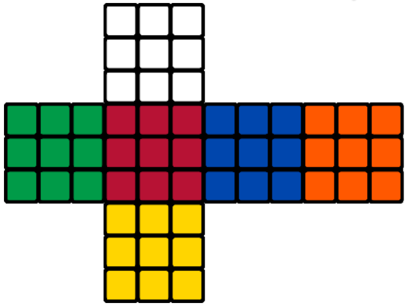
- HDFS is a repository for all stored data
  - Structure and organize carefully!
- Best practices include
  - Defining a standard directory structure
  - Including separate locations for staging data
- Example organization
  - /user/… → data and configuration belonging only to a single user
  - /etl → Work in progress in Extract/Transform/Load stage
  - /tmp → Temporary generated data shared between users
  - /data → Data sets that are processed and available across the organization for analysis
  - /app → Non-data files such as configuration, JAR files, SQL files, etc.

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







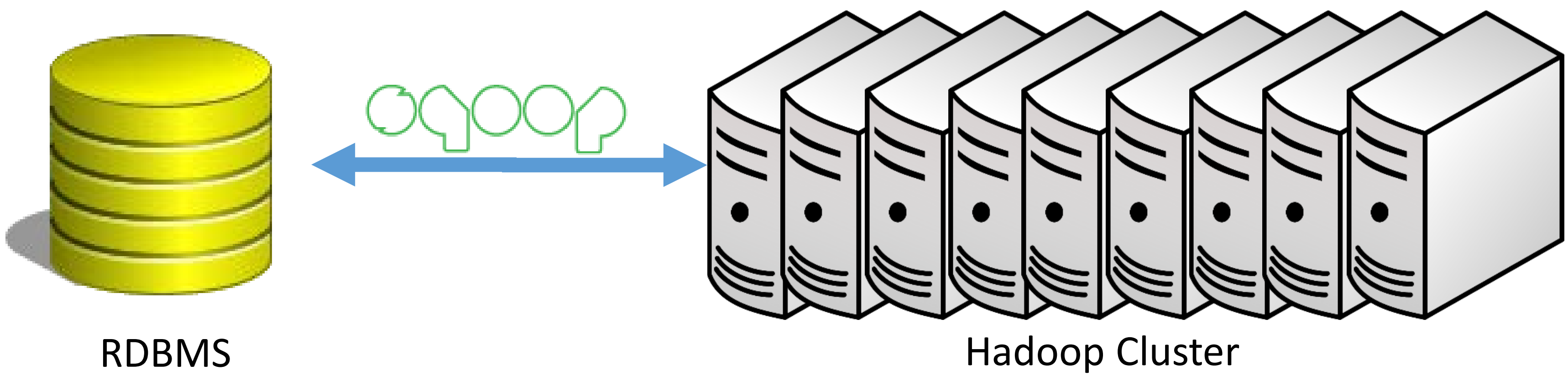
Apache Sqoop

We are all shaped by the tools we use, in particular: the formalisms we use shape our thinking habits, for better or for worse, and that means that we have to be very careful in the choice of what we learn and teach, for unlearning is not really possible.

Edsger Dijkstra

Apache Sqoop

- Open source Apache project originally developed by Cloudera
  - ➢ The name is a contraction of “SQL-to-Hadoop”
- Sqoop exchanges data between a database and HDFS
  - ➢ Can import all tables, a single table, or a partial table into HDFS
  - ➢ Data can be imported a variety of formats
  - ➢ Sqoop can also export data from HDFS to a database



Basic Syntax

- Sqoop is a command-line utility with several subcommands, called tools
  - There are tools for import, export, listing database contents, and more
  - Run `sqoop help` to see a list of all tools
  - Run `sqoop help tool -name` for help on using a specific tool
- Basic syntax of a Sqoop invocation

```
$ sqoop tool-name [tool-options]
```

- This command will list all tables in the `loudacre` database in MySQL

```
$sqoop      list-tables \
--connect   jdbc:mysql://dbhost/loudacre  \
--username  dbuser      \
--password  pw
```

Overview of Import Process

Big Data Engineering For Analytics

- Imports are performed using Hadoop MapReduce jobs
- Sqoop begins by examining the table to be imported
  - ➢ Determines the primary key, if possible
  - ➢ Runs a ***boundary query*** to see how many records will be imported
  - ➢ Divides result of boundary query by the number of tasks (mappers)
    - Uses this to configure tasks so that they will have equal loads
- Sqoop also generates a Java source file for each table being imported
  - ➢ It compiles and uses this during the import process
  - ➢ The file remains after import, but can be safely deleted

Importing an Entire Database Through Sqoop

Big Data Engineering For Analytics

- The `import-all-tables` tool imports an entire database
  - Stored as comma-delimited files
  - Default base location is your HDFS home directory
  - Data will be in subdirectories corresponding to name of each table

```
$ sqoop  import-all-tables    \
--connect   jdbc:mysql://dbhost/loudacre  \
--username  dbuser       --password  pw
```

- Use the `--warehouse-dir` option to specify a different base directory

```
$sqoop import-all-tables       \
--connect   jdbc:mysql://dbhost/loudacre  \
--username  dbuser       --password  pw    \
--warehouse-dir    /loudacre
```

Importing a Single Table with Sqoop

- The import tool imports a single table
- This example imports the accounts table.
  - ➢It stores the data in HDFS as comma-delimited fields

```
$ sqoop import --table accounts \
--connect jdbc:mysql://dbhost/loudacre \
--username dbuser --password pw
```

This variation writes tab-delimited fields instead

```
$ sqoop import --table accounts \
--connect jdbc:mysql://dbhost/loudacre \
--username dbuser --password pw \
--fields-terminated-by "\t"
```

Incremental Imports - 1

Big Data Engineering For Analytics

- What if records have changed since last import?
  - Could re-import all records, but this is inefficient
- Sqoop's incremental `lastmodified` mode imports new and modified records
  - Based on a timestamp in a specified column
  - You must ensure timestamps are updated when records are added or changed in the database

```
$ sqoop     import --table      invoices      \
--connect    jdbc:mysql://dbhost/loudacre     \
--username   dbuser --password    pw     \
--incremental lastmodified \
--check-column      mod_dt \
--last-value '2015-09-30  16:00:00'
```

Incremental Imports – 2.

- Or use Sqoop's incremental append mode to import only new records
  - Based on value of last record in specified column

```
$ sqoop      import --table      invoices      \
--connect    jdbc:mysql://dbhost/loudacre      \
--username   dbuser --password   pw     \
--incremental append \
--check-column      id     \
--last-value  9478306
```

Exporting Data from Hadoop to RDBMS with Sqoop

Big Data Engineering For Analytics

- Sqoop's import tool pulls records from an RDBMS into HDFS
- It is sometimes necessary to push data in HDFS back to an RDBMS
  - Good solution when you must do batch processing on large data sets
  - Export results to a relational database for access by other systems
- Sqoop supports this via the export tool
  - The RDBMS table must already exist prior to export

```
$ sqoop         export \
--connect       jdbc:mysql://dbhost/loudacre       \
--username      dbuser --password    pw      \
--export-dir    /loudacre/recommender_output       \
--update-mode   allowinsert    \
--table         product_recommendations
```

Importing Partial Tables with Sqoop

- Import only specified columns from accounts table

```
$ sqoop     import        --table     accounts    \
--connect   jdbc:mysql://dbhost/loudacre  \
--username  dbuser        --password  pw    \
--columns   "id, first_name, last_name, state"
```

- Import only matching rows from accounts table

```
$ sqoop     import        --table     accounts    \
--connect   jdbc:mysql://dbhost/loudacre  \
--username  dbuser        --password  pw    \
--where     "state='CA'"
```

Using Free Form Query

- You can also import the results of a query, rather than a single table
- Supply a complete SQL query using the `--query` option
  - You must add the literal `WHERE $CONDITIONS` token
  - Use `--split-by` to identify field used to divide work among mappers
  - The `--target-dir` option is required for free-form queries

```
$ sqoop import  \
--connect      jdbc:mysql://dbhost/loudacre  \
--username     dbuser  --password     pw     \
--target-dir   /data/loudacre/payable \
--split-by     accounts.id     \
--query 'SELECT accounts.id,   first_name, last_name, bill_amount     \
FROM    accounts        JOIN    invoices        ON \
(accounts.id    =       invoices.cust_id)       WHERE   $CONDITIONS'
```

Using a Free-Form Query with WHERE Criteria

- The --where option is ignored in a free-form query
  - You must specify your criteria using AND following the WHERE clause

```
$ sqoop import  \
--connect      jdbc:mysql://dbhost/loudacre  \
--username     dbuser  --password    pw      \
--target-dir   /data/loudacre/payable \
--split-by     accounts.id    \
--query 'SELECT accounts.id,    first_name, last_name, bill_amount
       FROM    accounts         JOIN    invoices        ON
       (accounts.id   =         invoices.cust_id)       WHERE
       $CONDITIONS    AND bill_amount         >=        40'
```

Options for Database Connectivity

Big Data Engineering For Analytics

- Generic (JDBC)
  - ➢Compatible with nearly any database
  - ➢Overhead imposed by JDBC can limit performance
- Direct Mode
  - ➢Can improve performance through use of database-specific utilities
  - ➢ Currently supports MySQL and Postgres (use `--direct` option)
  - ➢Not all Sqoop features are available in direct mode
- Cloudera and partners offer high--performance Sqoop connectors
  - ➢These use native database protocols rather than JDBC
  - ➢Connectors available for Netezza, Teradata, and Oracle
    - Download these from Cloudera’s Web site
    - Not open source due to licensing issues, but free to use

Controlling Parallelism

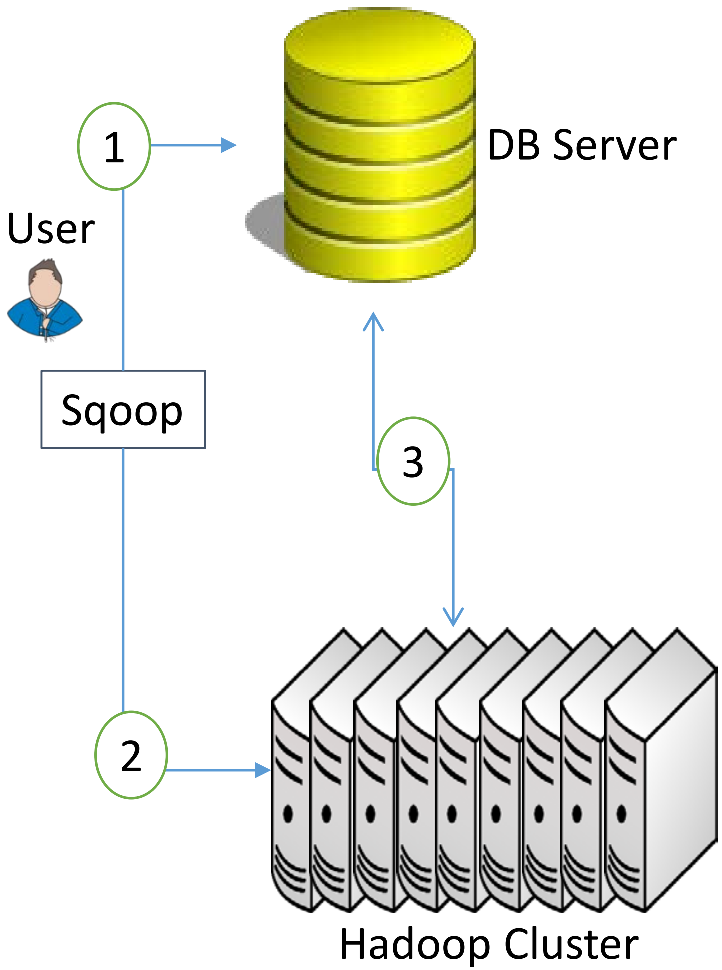
- By default, Sqoop typically imports data using four parallel tasks (called mappers)
  - Increasing the number of tasks might improve import speed
  - Caution: Each task adds load to your database server
- You can influence the number of tasks using the `-m` option
  - Sqoop views this only as a hint and might not honor it

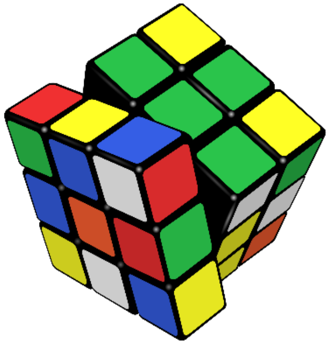
```
$ sqoop import  --table accounts        \
--connect       jdbc:mysql://dbhost/loudacre  \
--username      dbuser  --password     pw      \
-m      8
```

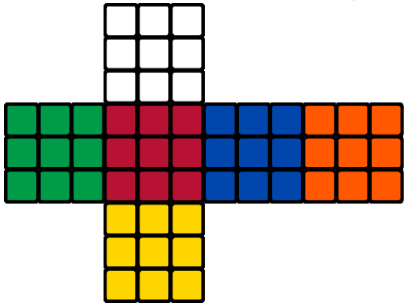
- Sqoop assumes all tables have an evenly-distributed numeric primary key
  - Sqoop uses this column to divide work among the tasks
  - You can use a different column with the `--split-by` option

Limitations of Sqoop

- Sqoop is stable and has been used successfully in production for years
- However, its client-side architecture does impose some limitations
  - ➢Requires connectivity to RDBMS from the client (client must have JDBC drivers installed)
  - ➢Requires connectivity to cluster from the client
  - ➢Requires user to specify RDBMS username and password
  - ➢Difficult to integrate a CLI within external applications
- Also tightly coupled to JDBC semantics
  - ➢A problem for NoSQL databases







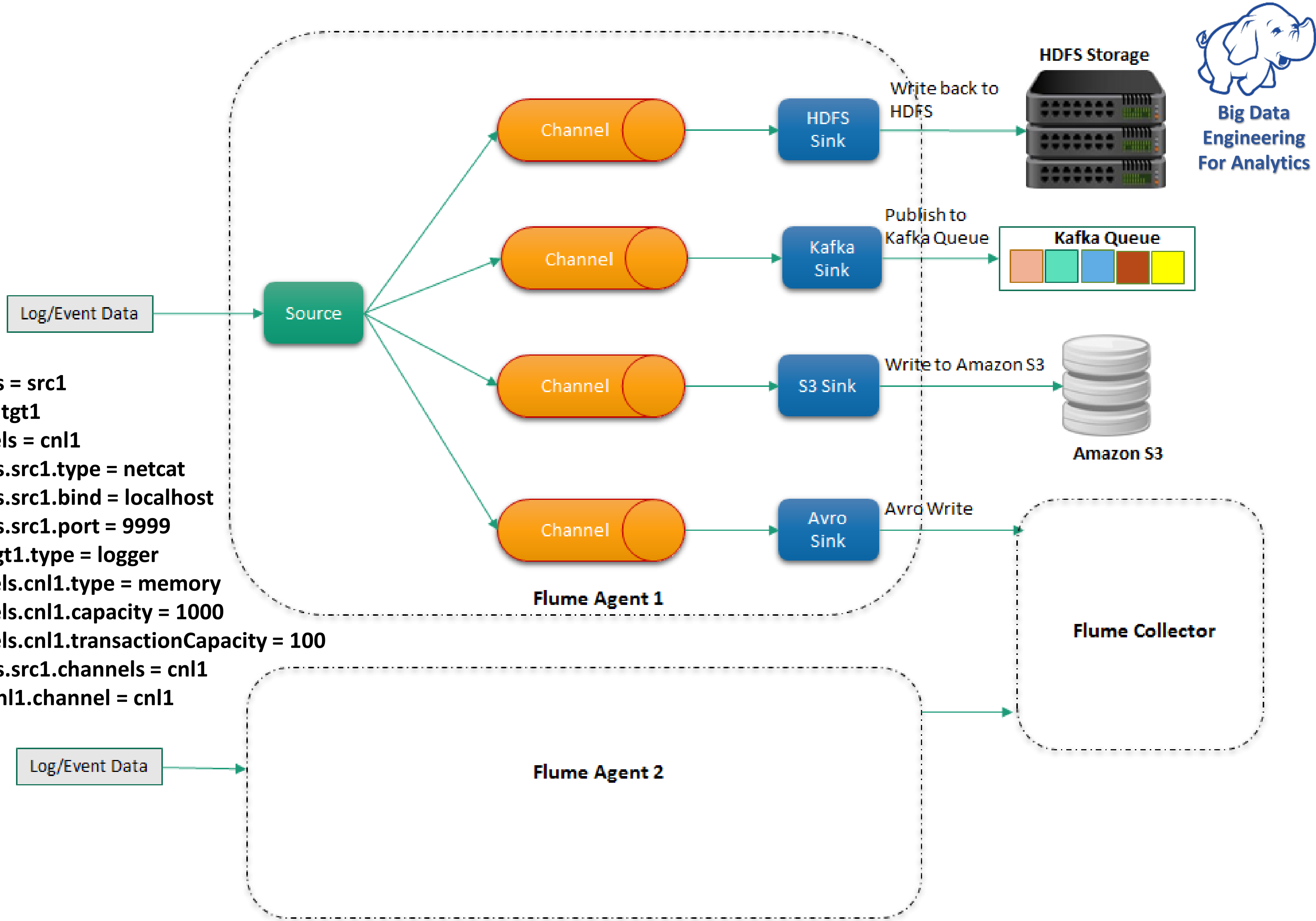
Apache Flume

Elegance is not a dispensable luxury but a factor that decides between success and failure.

Edsger Dijkstra

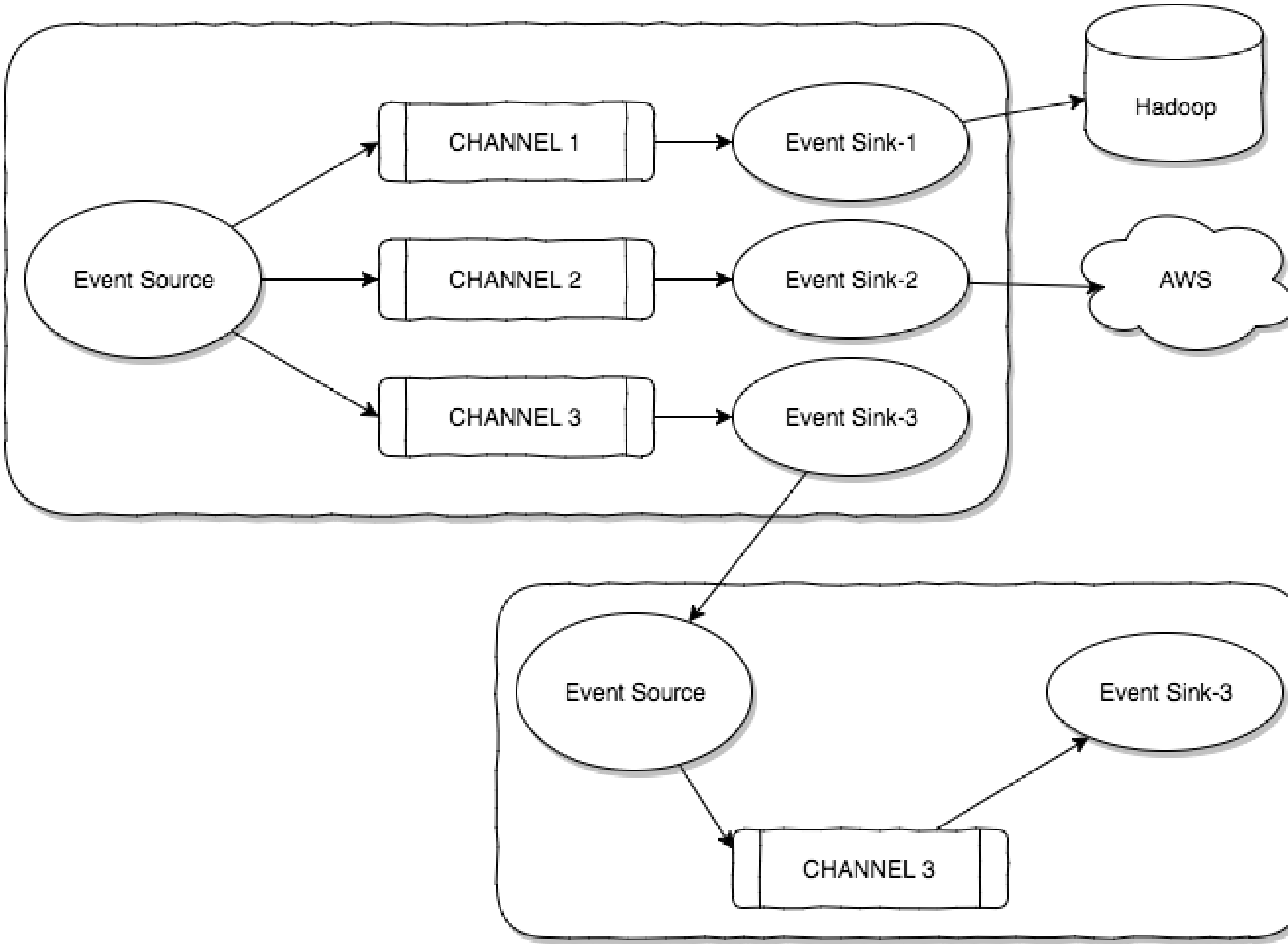
Apache Flume

- **Apache Flume is a high-‐performance system for data collection**
  - Name derives from original use case of near-‐real time log data ingestion
  - Now widely used for collection of any streaming event data
  - Supports aggregating data from many sources into HDFS
- **Originally developed by Cloudera**
  - Donated to Apache Software Foundation in 2011
  - Became a top-‐level Apache project in 2012
  - Flume OG gave way to Flume NG (Next Generation)
- **Benefits of Flume**
  - Horizontally-‐scalable
  - Extensible
  - Reliable




```
a1.sources = src1
a1.sinks = tgt1
a1.channels = cnl1
a1.sources.src1.type = netcat
a1.sources.src1.bind = localhost
a1.sources.src1.port = 9999
a1.sinks.tgt1.type = logger
a1.channels.cnl1.type = memory
a1.channels.cnl1.capacity = 1000
a1.channels.cnl1.transactionCapacity = 100
a1.sources.src1.channels = cnl1
a1.sinks.cnl1.channel = cnl1
```

Flow Multiplexer

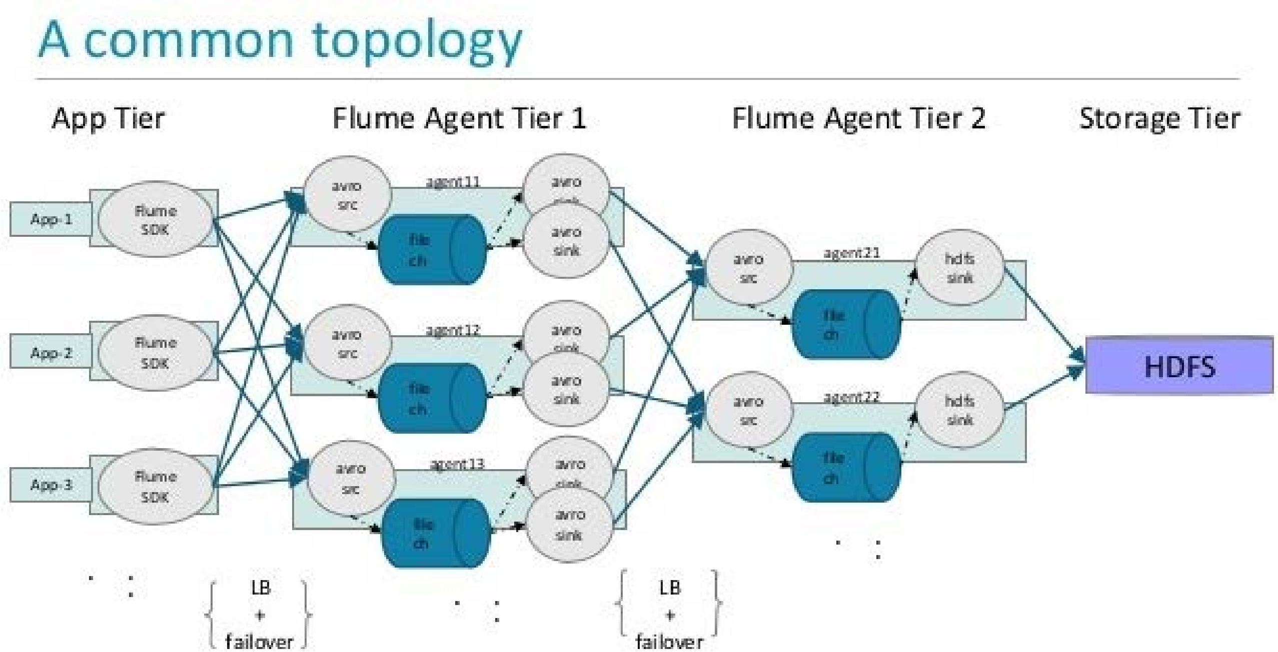


Flume Topology

Big Data Engineering For Analytics

Flume Use Cases

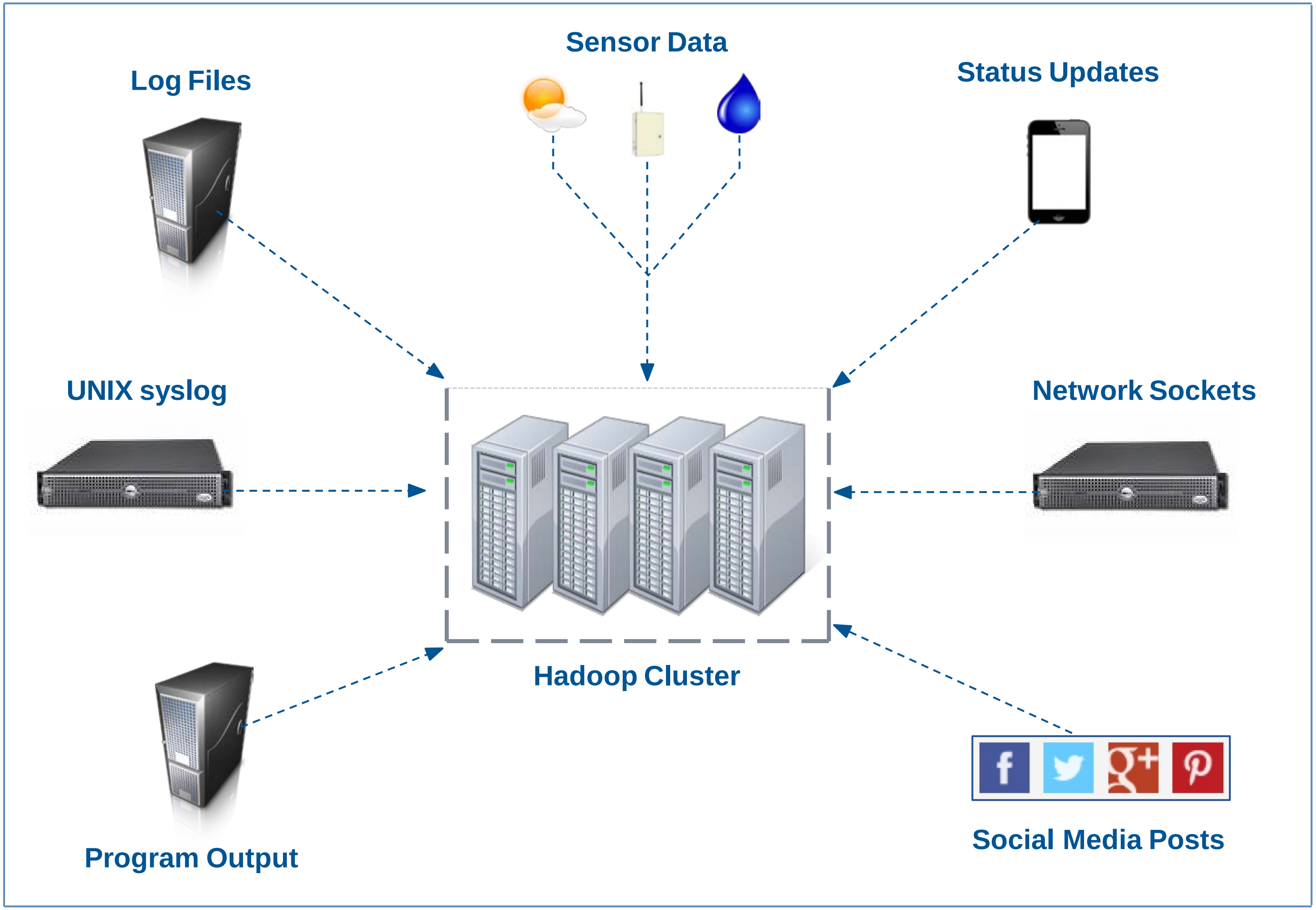
1. Collect network traffic data (or any structured data)
2. Collect social media data (or semi structured to unstructured data)
3. Email messages
4. Website scraped data



Flume's Design Goals

- **Channels provide Flume's Reliability**
  - **Memory Channel -** Data will be lost if power is lost
  - **Disk–based Channel -** Disk–based queue guarantees durability of data in face of a power loss
  - **Data transfer between Agents and Channels is transactional -** A failed data transfer to a downstream agent rolls back and retries
  - **Can configure multiple Agents with the same task -** For example, 2 Agents doing the job of 1 'collector' – if one agent fails then upstream agents would fail over
- **Scalability**
  - The ability to increase system performance linearly – or better – by adding more resources to the system; Flume scales horizontally; As load increases, more machines can be added to the configuration
- **Extensibility** The ability to add new functionality to a system
  - **Flume can be extended by adding Sources and Sinks to existing storage layers or data platforms**
    - General Sources include data from files, syslog, and standard output from any Linux process
    - General Sinks include files on the local filesystem or HDFS
    - Developers can write their own Sources or Sinks

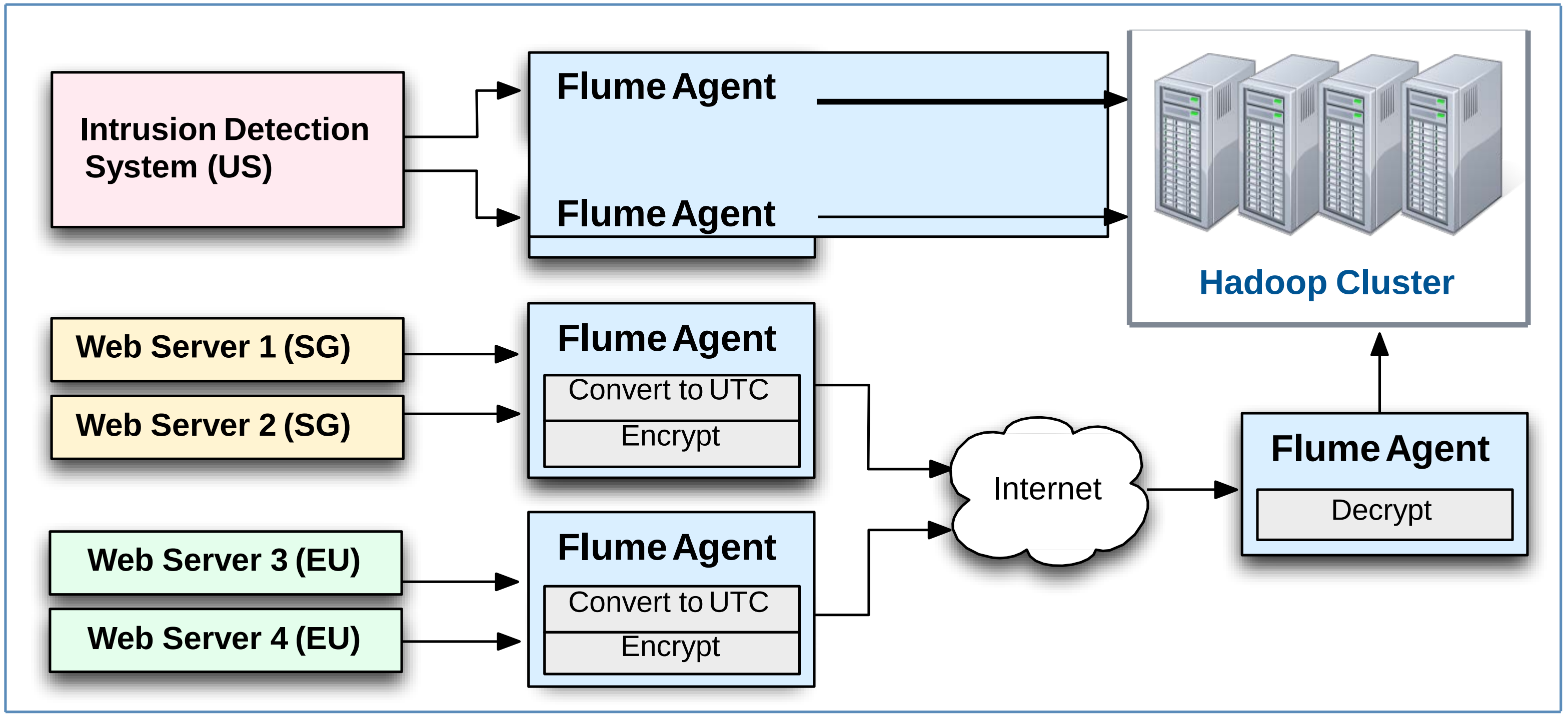
Common Data Sources



Large-Scale Deployment Example

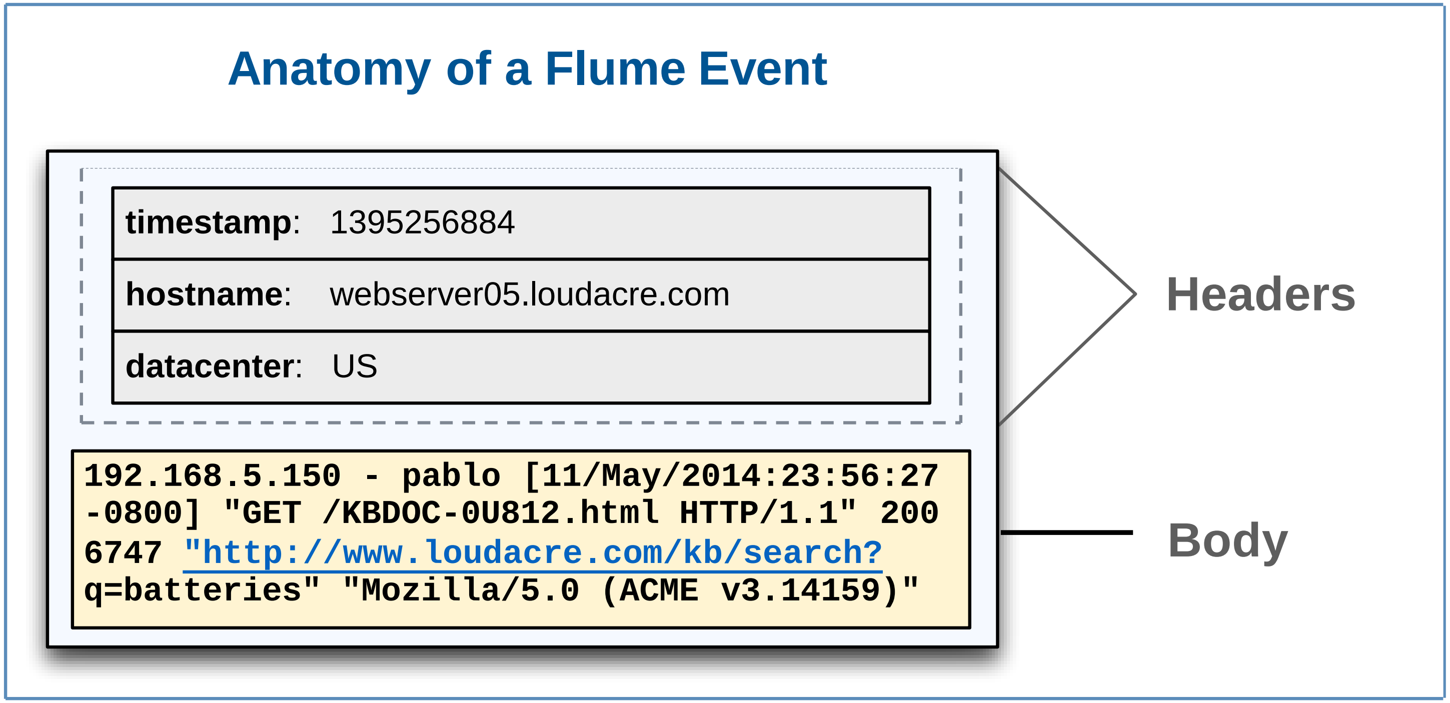
Flume collects data using configurable "agents"

- Agents can receive data from many sources, including other agents
- Large-scale deployments use multiple tiers for scalability and reliability
- Flume supports inspection and modification of in--flight data



Flume Events

- An event is the fundamental unit of data in Flume
- Consists of a body (payload) and a collection of headers (metadata)
- Headers consist of name-value pairs
- Headers are mainly used for directing output

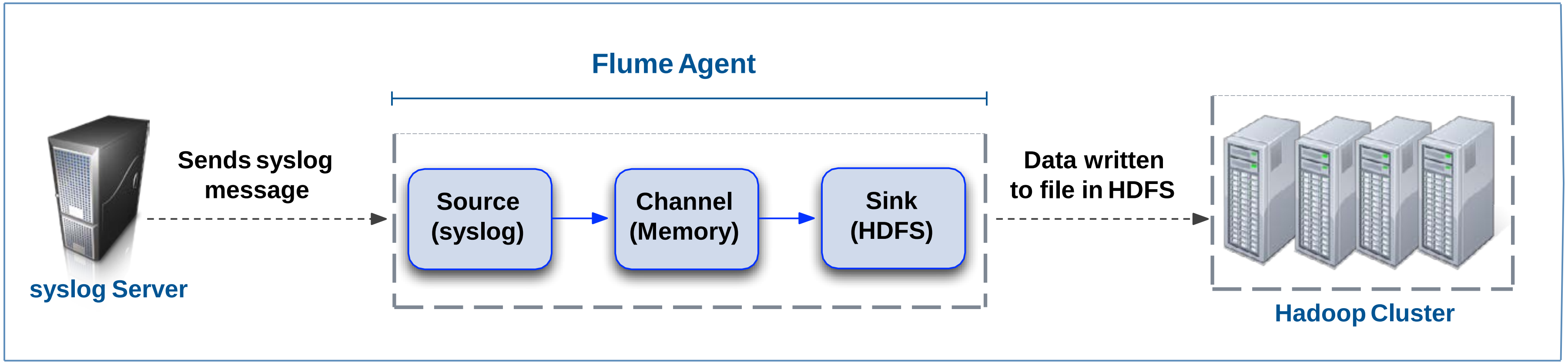


Flume Data Flow

Big Data Engineering For Analytics

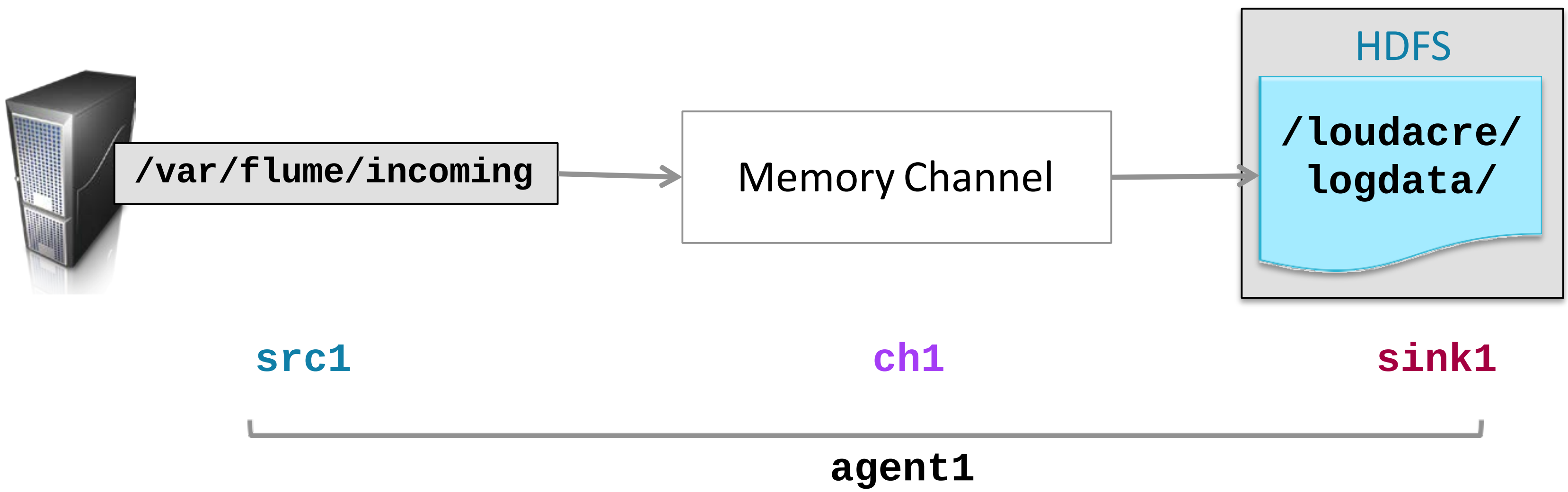
This diagram illustrates how syslog data might be captured to HDFS

1. Message is logged on a server running a syslog daemon
2. Flume agent configured with syslog source receives event
3. Source pushes event to the channel, where it is buffered in memory
4. Sink pulls data from the channel and writes it to HDFS



Example Component Configuration

Example: Configure a Flume Agent to collect data from remote spool directories and save to HDFS

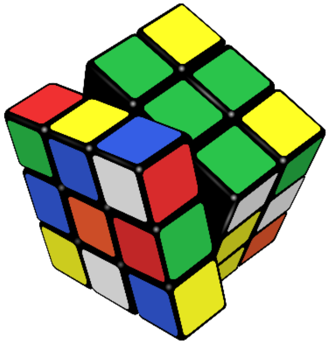


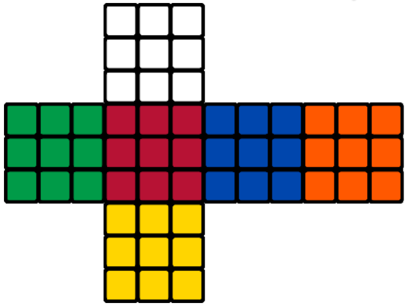
Flume Sources & Sinks

- Syslog
  - ➢Captures messages from UNIX syslog daemon over the network
- Netcat
  - ➢Captures any data written to a socket on an arbitrary TCP port
- Exec
  - ➢Executes a UNIX program and reads events from standard output *
- Spooldir
  - ➢Extracts events from files appearing in a specified (local) directory
- HTTP Source
  - ➢Receives events from HTTP requests
- Null
  - ➢Discards all events (Flume equivalent of /dev/null)
- Logger
  - ➢Logs event to INFO level using SLF4J
- IRC
  - ➢Sends event to a specified Internet Relay Chat channel
- HDFS
  - ➢Writes event to a file in the specified directory in HDFS
- HBaseSink
  - ➢Stores event in HBase

Flume Channels

- **Memory**
  - Stores events in the machine's RAM
  - Extremely fast, but not reliable (memory is volatile)
- **File**
  - Stores events on the machine's local disk
  - Slower than RAM, but more reliable (data is written to disk)
- **JDBC**
  - Stores events in a database table using JDBC
  - Slower than file channel





Summary

Those that can, do. Those that can't, complain.

Linus Torvalds

Key Points

- HDFS is the default storage in Hadoop, which is distributed, considerably simple in design, extremely scalable, flexible, and with high fault tolerance capability.
- HDFS architecture has a master-slave pattern due to which the slave nodes can be better managed and utilized.
  - Chunks data into blocks and distributes them across the cluster when data is stored
  - Slave nodes run DataNode daemons, managed by a single NameNode on a master node
- Access HDFS using Hue, the hdfs command or via the HDFS API
- One key assumption in HDFS is *Moving Computation is Cheaper than Moving Data*.

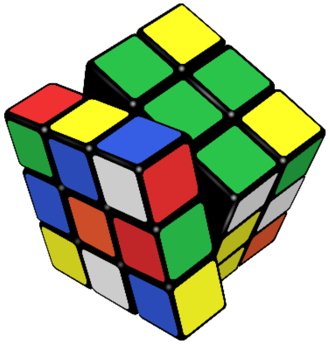
Key Points

Big Data Engineering For Analytics

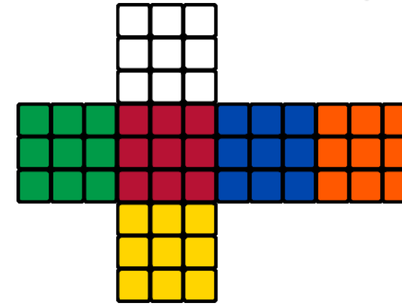
- Sqoop exchanges data between a database and the Hadoop cluster
  - ➢ Provides subcommands (tools) for importing, exporting, and more
  - ➢ You can select only certain columns or limit rows
  - ➢ Supports using joins in free-form queries
- **Apache Flume is a high-performance system for data collection**
  - Scalable, extensible, and reliable
- **A Flume agent manages the source, channels, and sink**
  - Source receives event data from its origin
  - Sink sends the event to its destination
  - Channel buffers events between the source and sink
- **The Flume agent is configured using a properties file**
  - Each component is given a user-defined ID
  - This ID is used to define properties of that component

Five Representative Case Studies

Objective	Solution Characteristics	Tools	Details
Handling large data volume	Data extraction with load-balancing using a distributed solution or a cluster of nodes.	Apache Flume, Apache Storm, Apache Spark	Apache Flume is useful in processing log-data. Apache Storm is desirable for operations monitoring and Apache Spark for streaming data, graph processing and machine-learning.
Messaging for distributed ingestion	Messaging system should ensure scalable and reliable communication across nodes involved in data-ingestion.	Apache Kafka	LinkedIn makes use of Apache Kafka to achieve fast communication between the cluster-nodes.
Real-time or near real-time ingestion	Data-ingestion process should be able to handle high-frequency of incoming or streaming data.	Apache Storm, Apache Spark	
Batch-mode ingestion	Ability to ingest data in bulk-mode.	Apache Sqoop, Apache Kafka, Apache Chukwa	Apache Chukwa process data in batch-mode and are useful when data needs to be ingested at an interval of few minutes/hours/days.
Detecting incremental data	Ability to handle structured and unstructured data, low-latency.	DataBus, Infosphere and Goldengate	Databus from LinkedIn is a distributed solution that provides a timeline-consistent stream of change capture events for a database. – Infosphere and Goldengate are Data Integrators



Big Data Engineering For Analytics



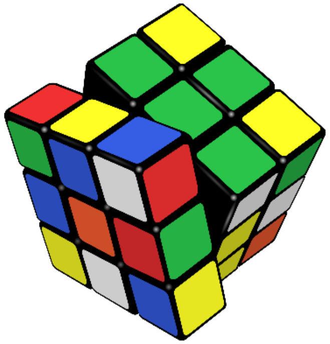
Reference

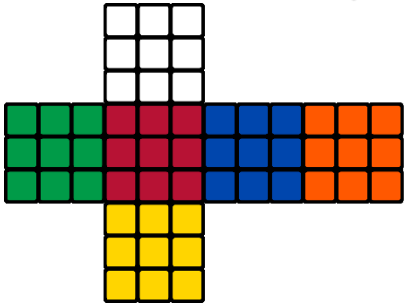
An individual developer like me cares about writing the new code and making it as interesting and efficient as possible. But very few people want to do the testing.

Linus Torvalds

Reference

- Tanenbaum, Andrew S., and Maarten Van Steen. *Distributed systems*. Prentice-Hall, 2007.
- Shvachko, Konstantin, et al. "The hadoop distributed file system." *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE, 2010.
- Borthakur, Dhruba. "The hadoop distributed file system: Architecture and design." *Hadoop Project Website* 11.2007 (2007): 21.



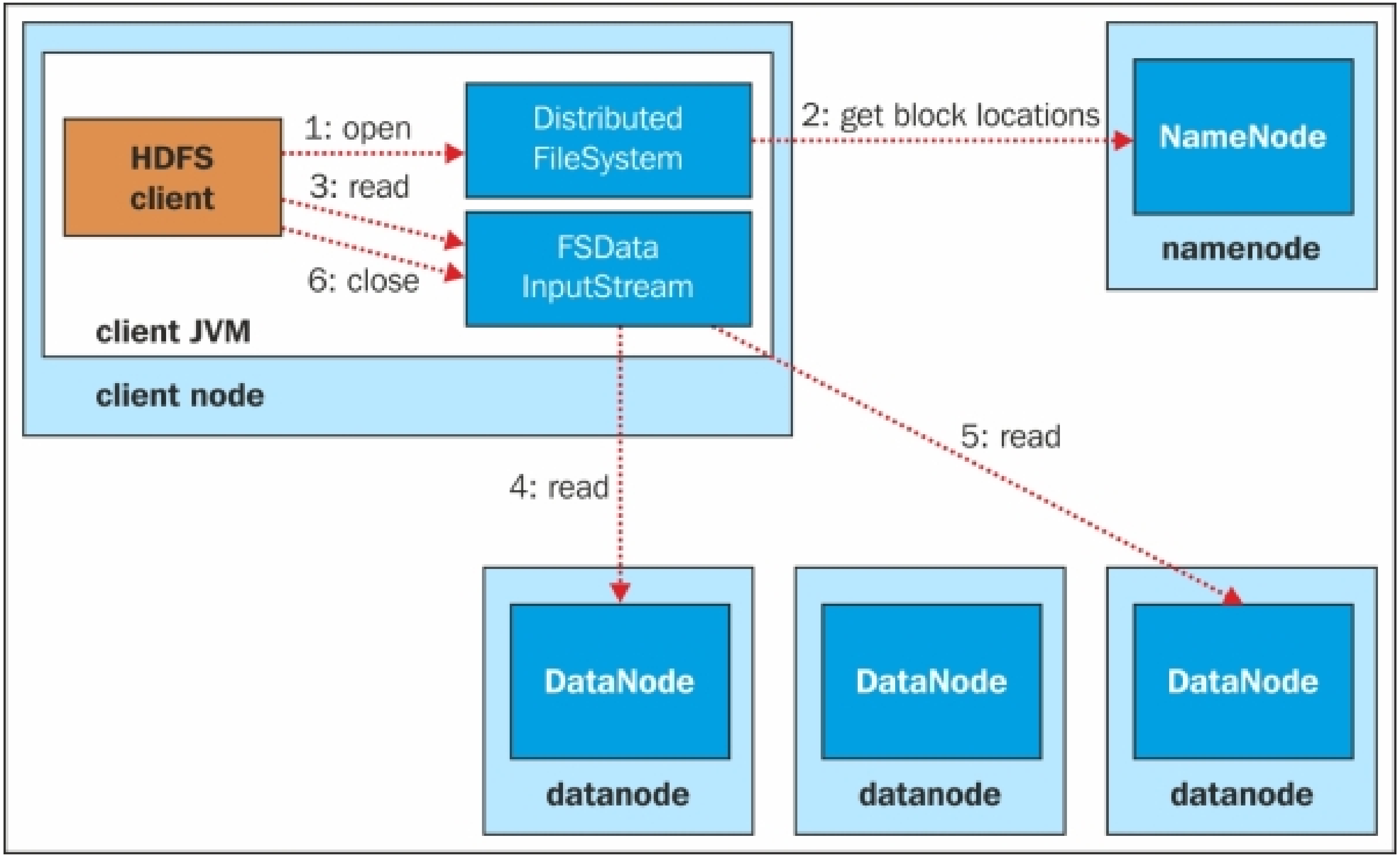


Appendix

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live"

— John Woods

Read Pipeline



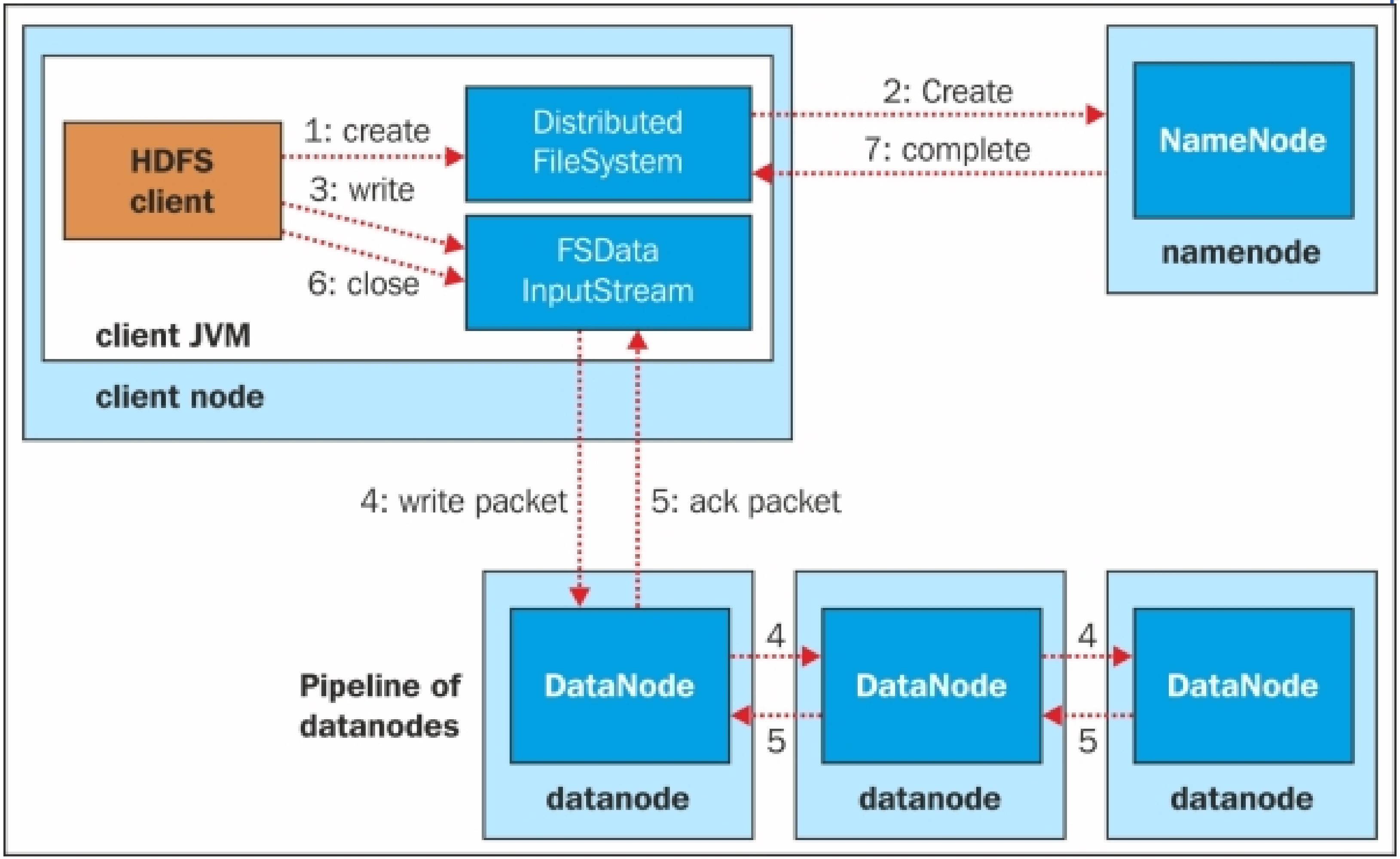
Read Pipeline

Big Data Engineering For Analytics

The HDFS read process involves the following six steps:

1. The client using a **Distributed FileSystem** object of Hadoop client API calls open() which initiate the read request.
2. **Distributed FileSystem** connects with **NameNode**. **NameNode** identifies the block locations of the file to be read and in which **DataNodes** the block is located. **NameNode** then sends the list of **DataNodes** in order of nearest DataNodes from the client.
3. **Distributed FileSystem** then creates **FSDataInputStream** objects, which, in turn, wrap a **DFSInputStream**, which can connect to the **DataNodes** selected and get the block, and return to the client. The client initiates the transfer by calling the read() of **FSDataInputStream**.
4. **FSDataInputStream** repeatedly calls the read() method to get the block data.
5. When the end of the block is reached, **DFSInputStream** closes the connection from the DataNode and identifies the best DataNode for the next block.
6. When the client has finished reading, it will call close() on **FSDataInputStream** to close the connection.

Write Pipeline



Write Pipeline - 1

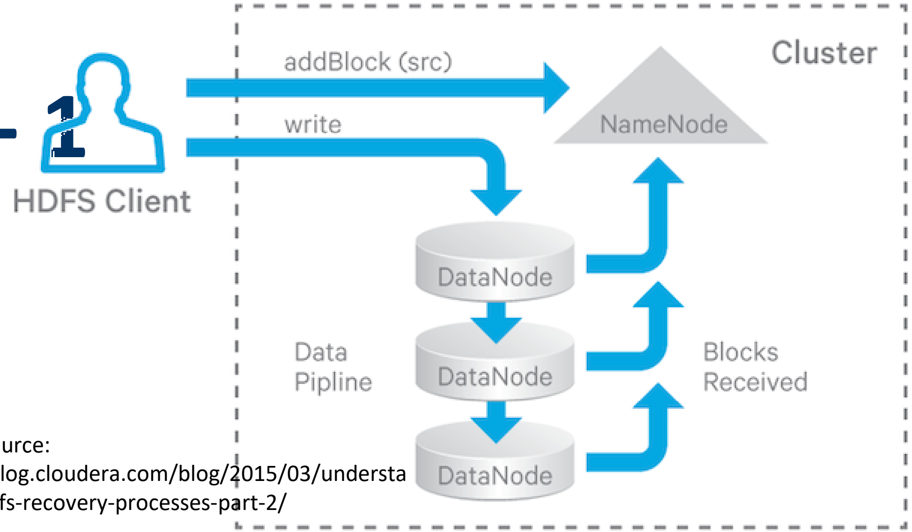


Image Source:
https://blog.cloudera.com/blog/2015/03/understanding-hdfs-recovery-processes-part-2/

The HDFS write pipeline process flow is described in the following seven steps:

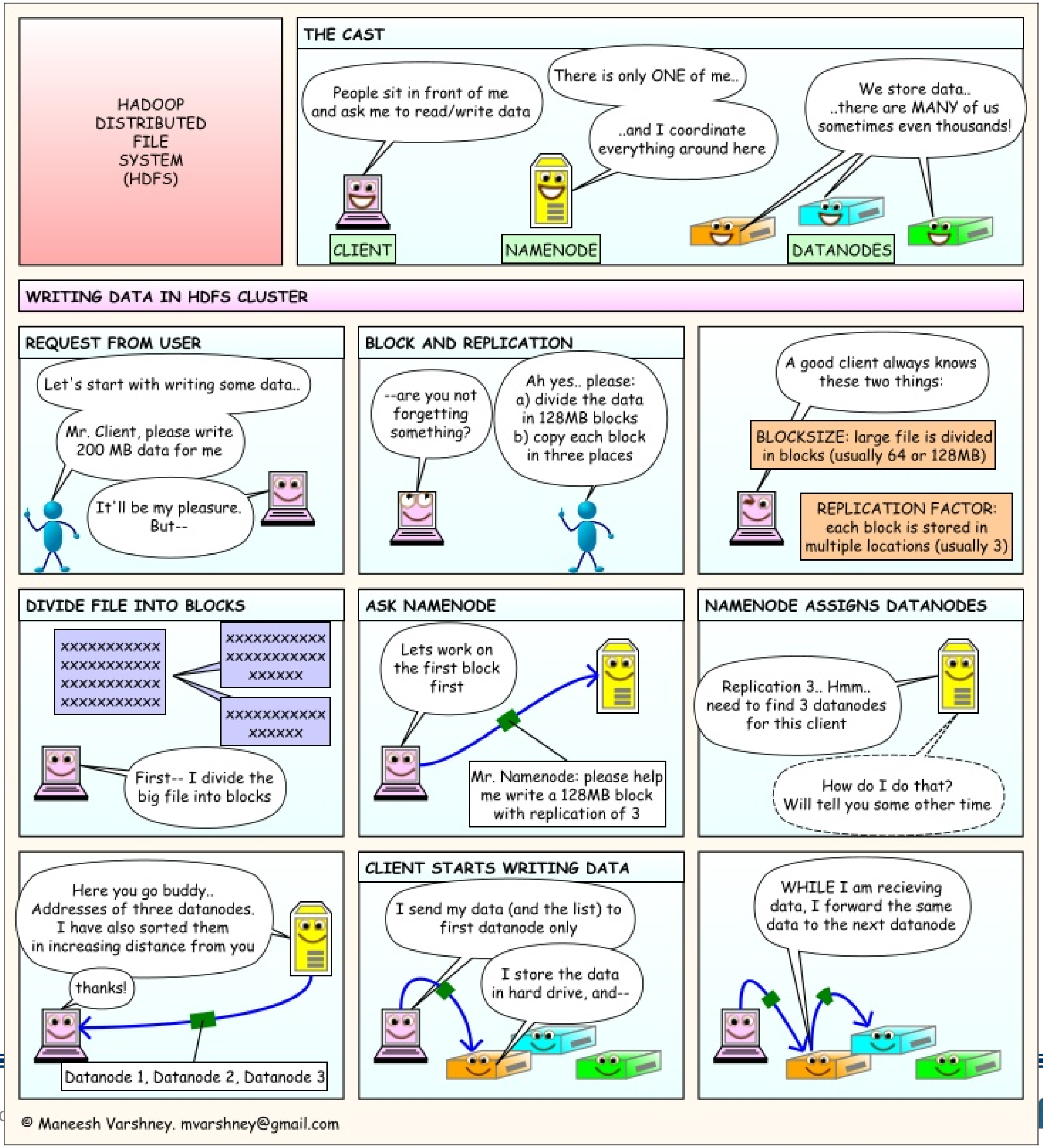
1. The client, using a **Distributed FileSystem** object of Hadoop client API, calls create(), which initiates the write request.
2. **Distributed FileSystem** connects with **NameNode**. **NameNode** initiates a new file creation, and creates a new record in metadata and initiates an output stream of type **FSDataOutputStream**, which wraps **DFSOutputStream** and returns it to the client. Before initiating the file creation,**NameNode** checks if a file already exists and whether the client has permissions to create a new file and if any of the condition is true then an IOException is thrown to the client.
3. The client uses the **FSDataOutputStream** object to write the data and calls the write() method. The **FSDataOutputStream** object, which is DFSOutputStream, handles the communication with the DataNodes and **NameNode**.

Write Pipeline – 2.

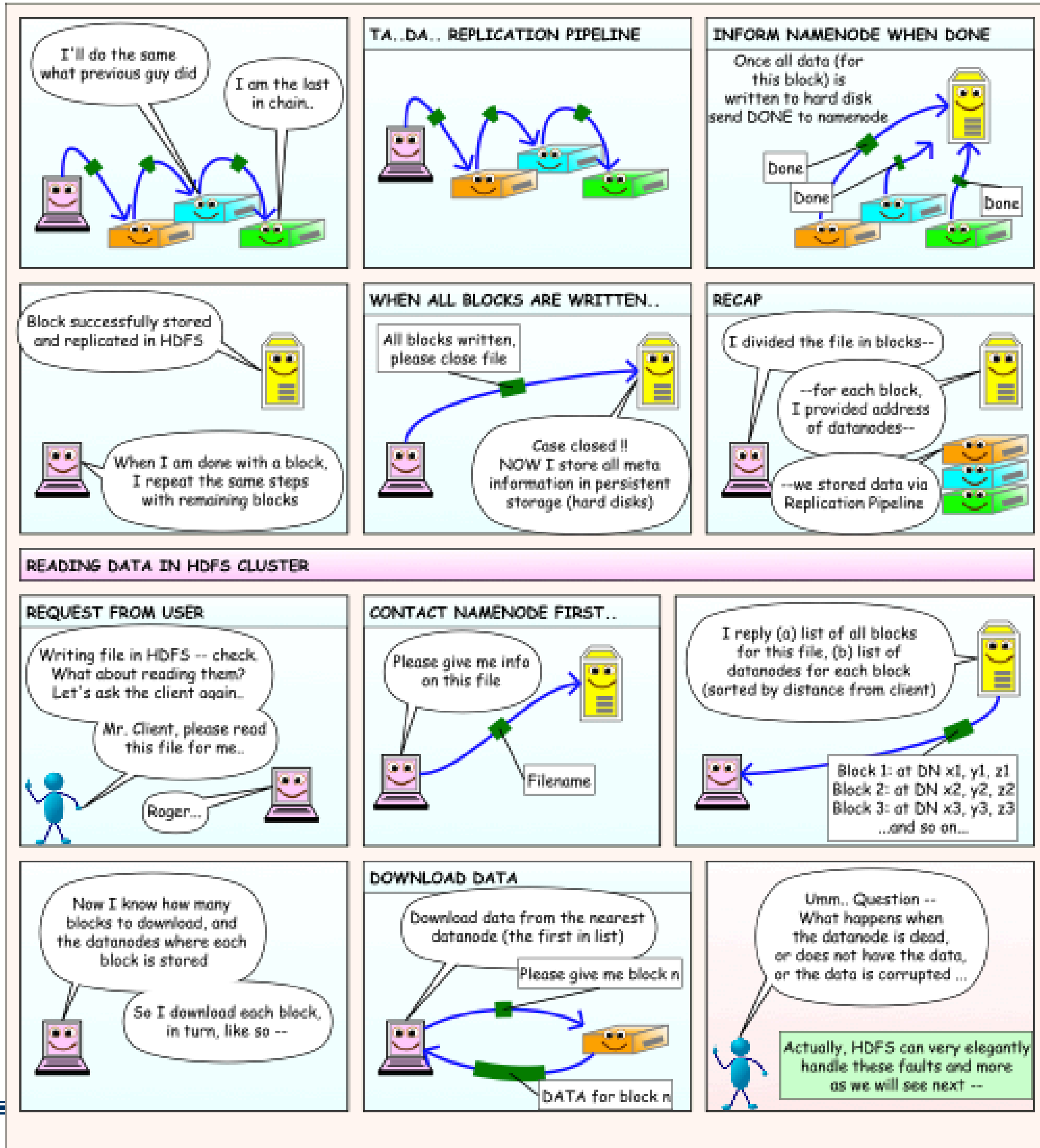
4. DFSOutputStream splits files to blocks and coordinates with **NameNode** to identify the **DataNode**and the replica DataNodes. The number of the replication factor will be the number of DataNodes identified. Data will be sent to a **DataNode** in packets, and that **DataNode** will send the same packet to the second **DataNode**, the second **DataNode** will send it to the third, and so on, until the number of DataNodes is identified.
5. When all the packets are received and written, DataNodes send an acknowledgement packet to the sender **DataNode**, to the client. DFSOutputStream maintains a queue internally to check if the packets are successfully written by **DataNode**. DFSOutputStream also handles if the acknowledgment is not received or **DataNode** fails while writing.
6. If all the packets have been successfully written, then the client closes the stream.
7. If the process is completed, then the **Distributed FileSystem** object notifies the **NameNode** of the status.

Source: Maneesh - https://drive.google.com/file/d/0B-zw6KHOtbT4MmRkZWJjYzEtYjI3Ni00NTFjLWE0OGItYTU5OGMxYjc0N2M1/view

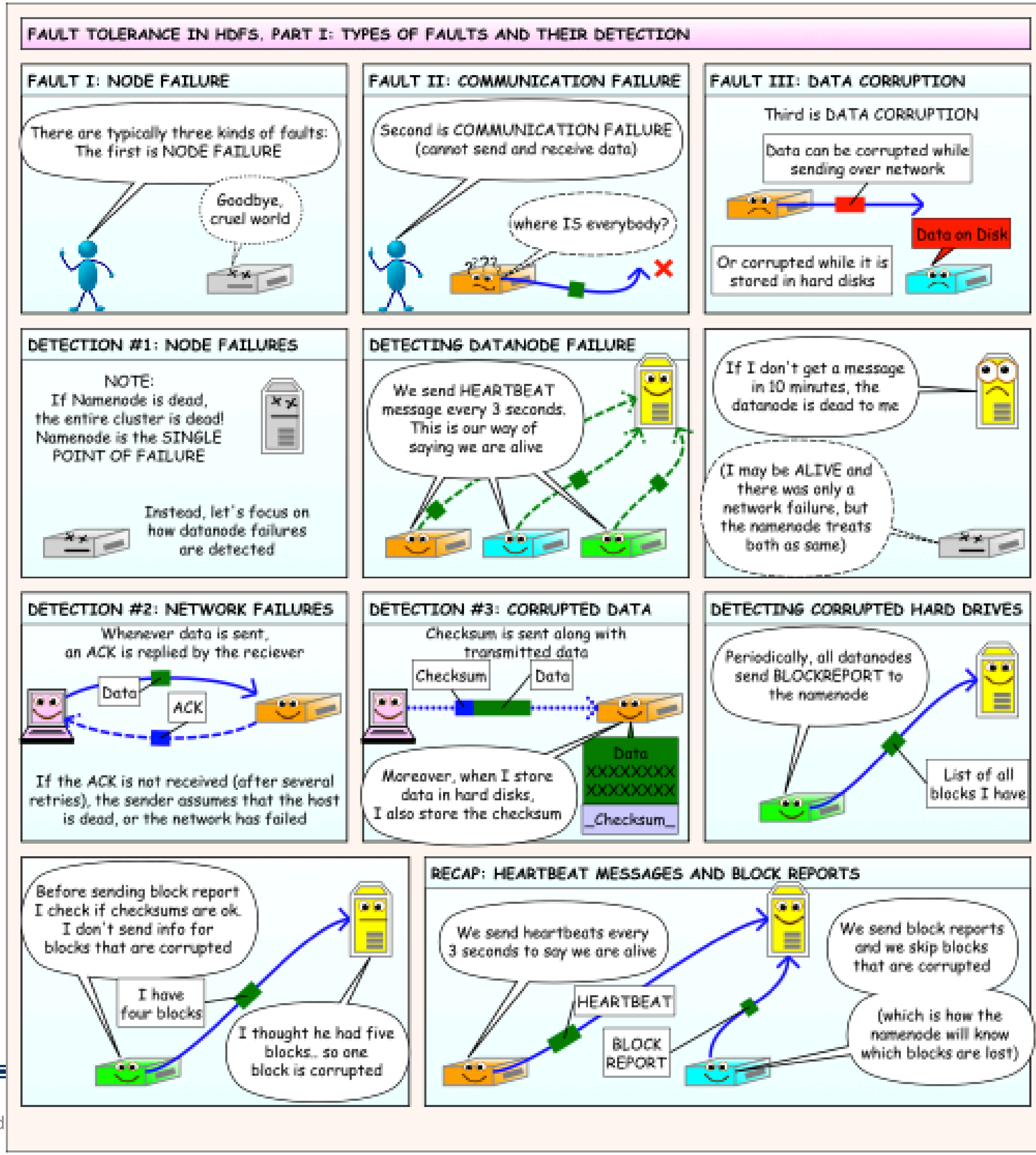
Big Data Engineering For Analytics



Big Data Engineering For Analytics



Big Data Engineering For Analytics



Big Data Engineering For Analytics