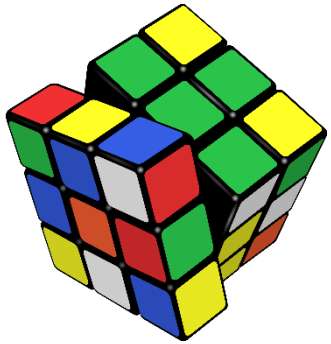


Big Data Engineering For Analytics

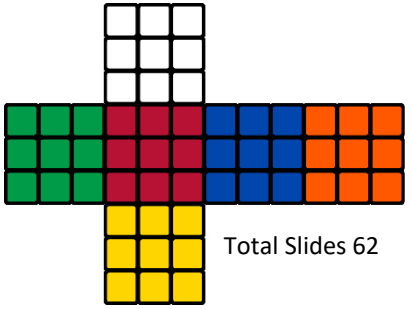
Design of Ingestion Layer and Related Tools

Suria R Asai

(suria@nus.edu,sg)

Institute of Systems Science

National University of Singapore



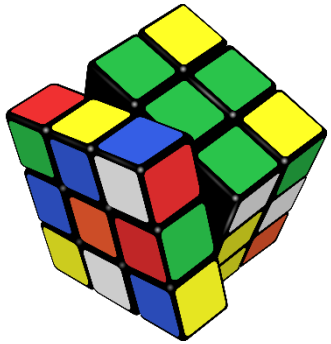
Total Slides 62

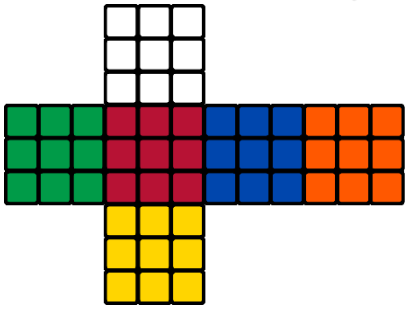
Learning Objectives

- Understand the fundamental concepts related to **Data Ingestion**.
- Prepare and manage data import/export efficiently from **multiple data sources**, with a reasonably good performance.
- Understand and use Data Ingestion Tools such as **Sqoop** and **Flume**

Agenda

- Introduction to Data Ingestion.
- Common Data Sources and Ingestion Layer Patterns
- Tool: Apache Sqoop
- Tool: Apache Flume
- Summary





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

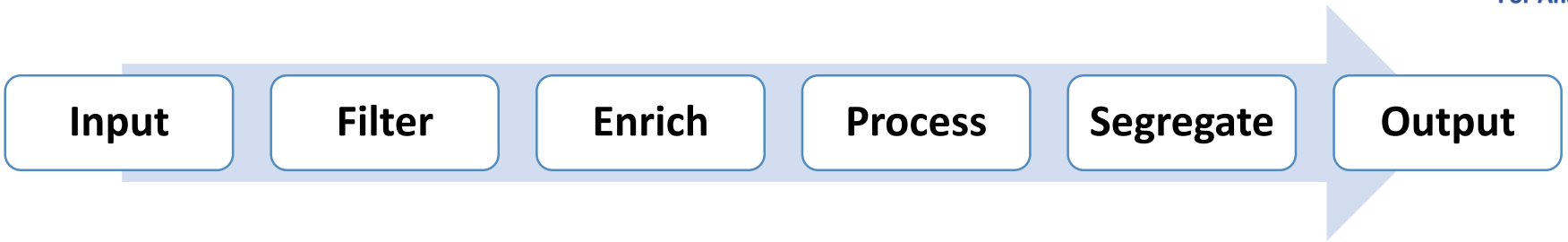
- **Data sensors**: These are thousands of sensors, producing data continuously.
- **Machine Data**: Produces data which should be processed in near real time for avoiding huge loss.
- **Telco Data**: CDR data and other telecom data generates high volume of data.
- **Healthcare system data**: Genes, 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

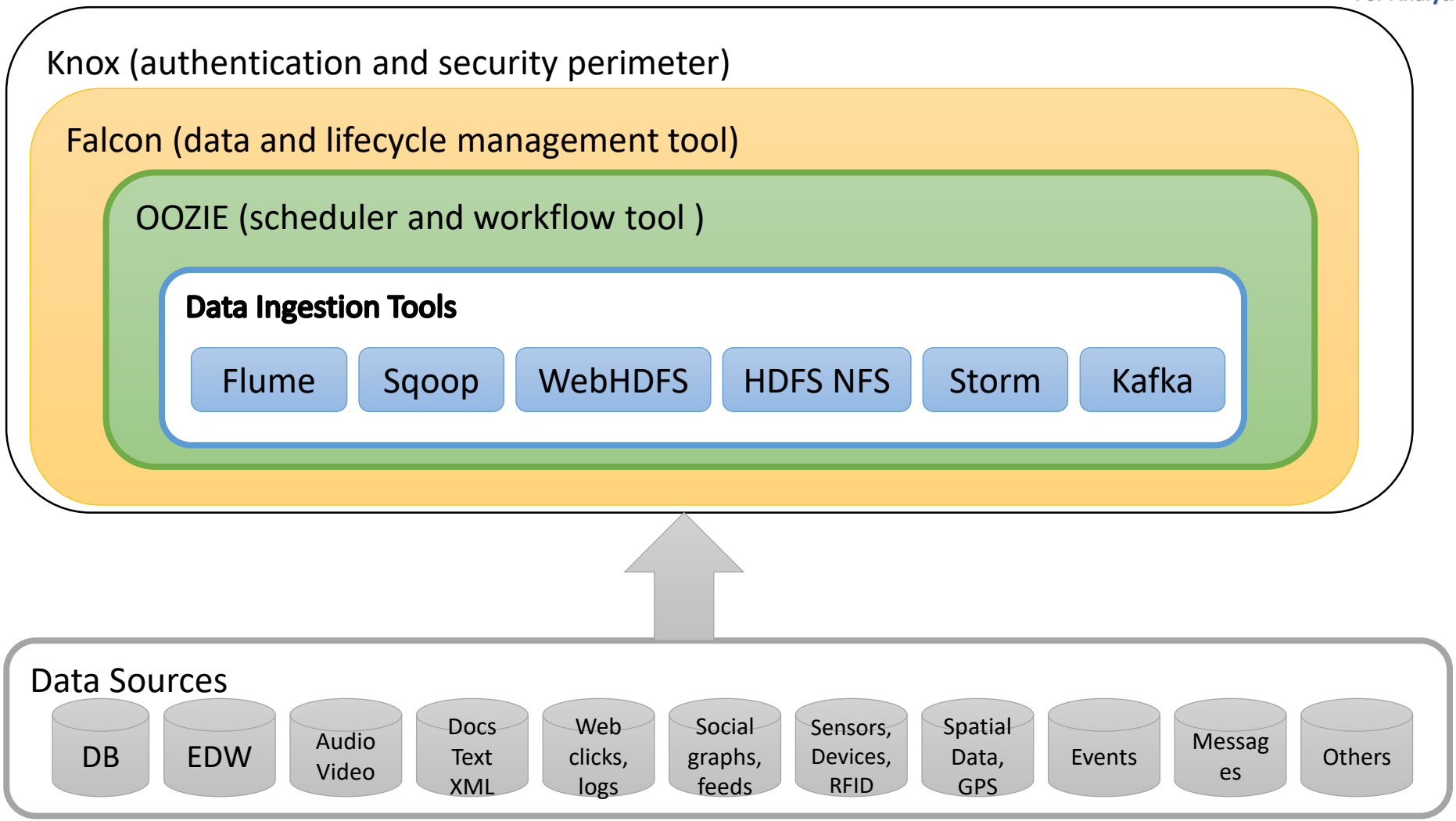
- Prioritizing each data source load
- Tagging and indexing ingested data.
- Validating and cleansing the ingested data.
- Transforming and compressing before ingestion.
- New data sources tend to deliver data at varying speed and frequencies; example: streaming and real time ingestion.
- With a wider 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



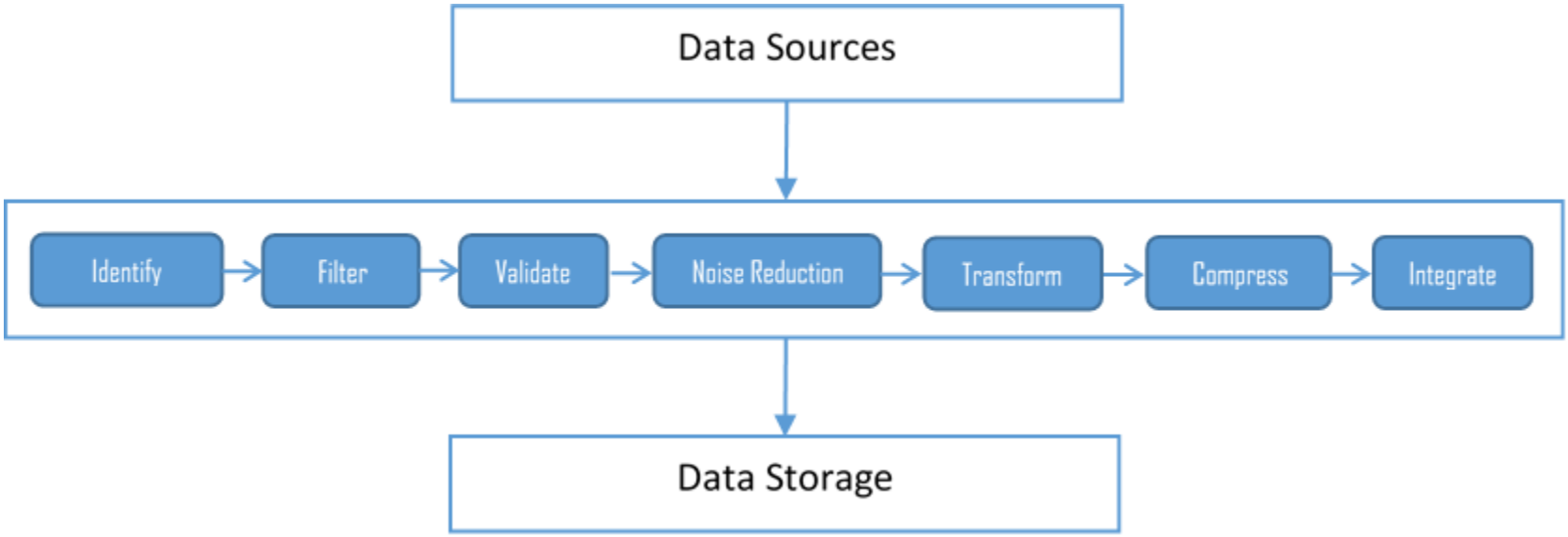
1. **Input:** Discover and fetch the data for ingestion. The discovery of data may be from File System, messaging queues, web services, sensors, databases or even the outputs of other ingestion apps.
2. **Filter:** Analyse the raw data and identify the interesting subset. The filter stage is typically 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

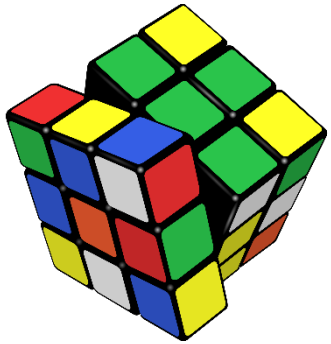
- Multiple data source load and prioritization
- Ingested data indexing and tagging
- Data validation and cleansing
- Data transformation and compression

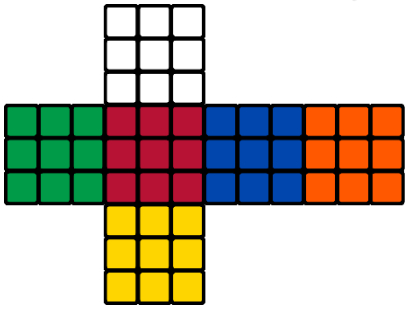


Five Representative Case Studies

Big Data Engineering For Analytics

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





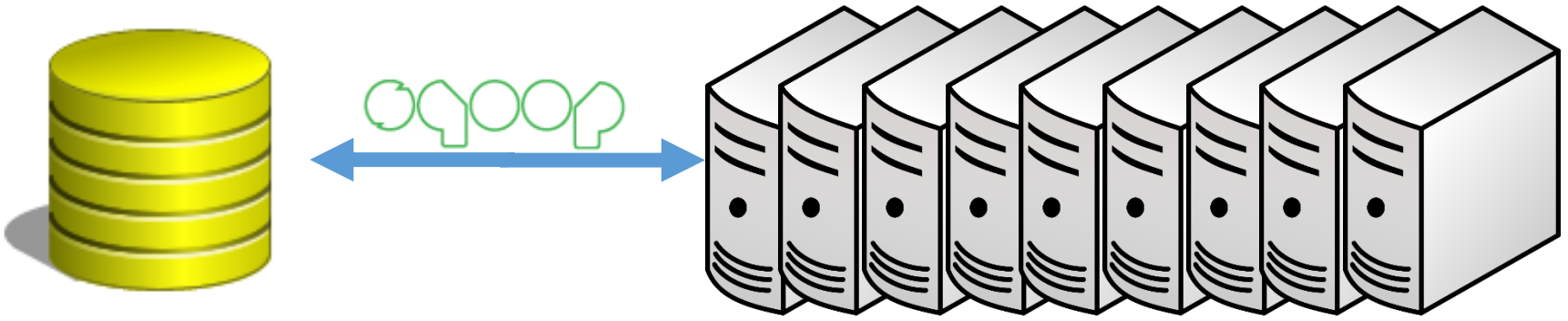
Apache Sqoop

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

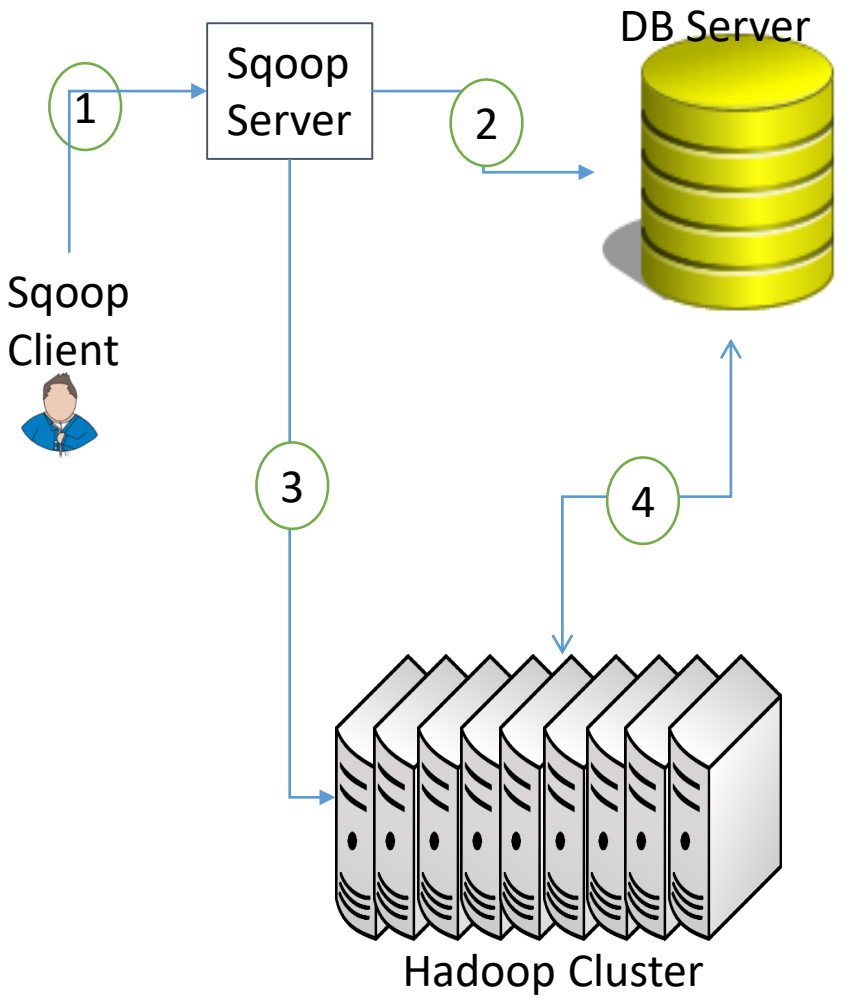
Apache Sqoop

- 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



Sqoop 2 Architecture

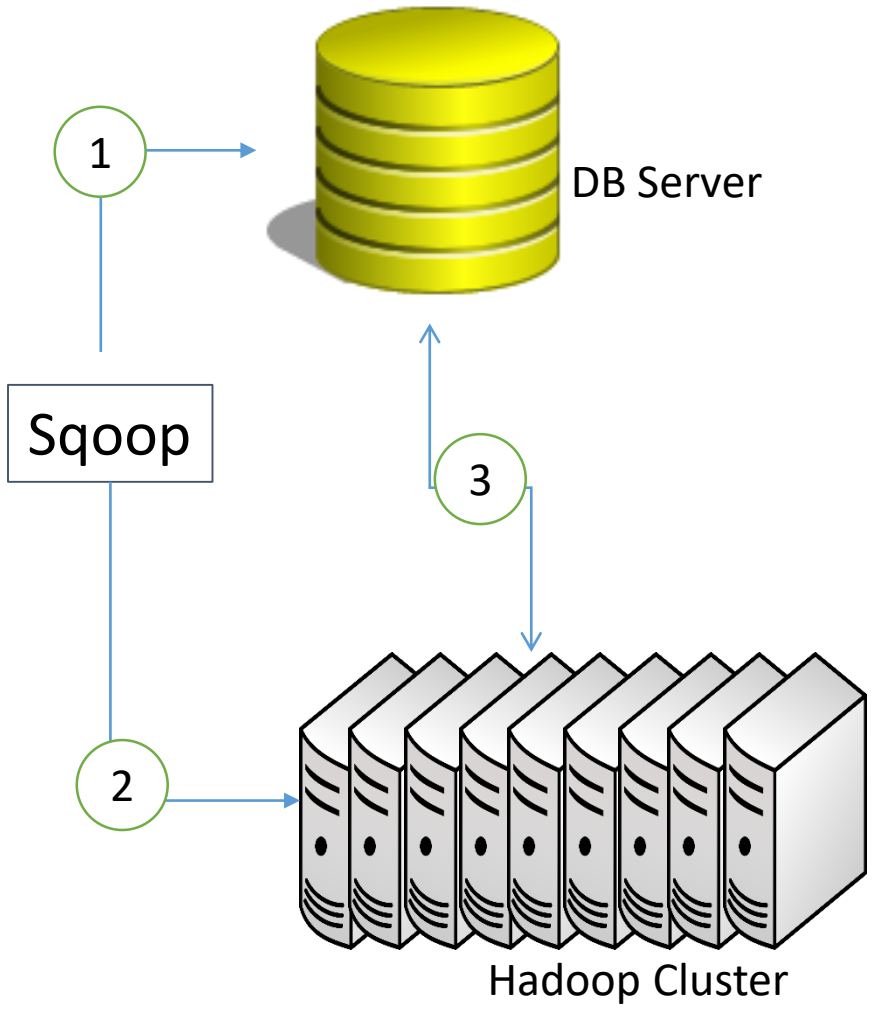
- Sqoop 2 is the next--generation version of Sqoop
  - Client--server design addresses limitations described earlier
  - API changes also simplify development of other Sqoop connectors
- Client requires connectivity only to the Sqoop server
  - DB connections are configured on the server by a system administrator
  - End users no longer need to possess database credentials
  - Centralized audit trail
  - Better resource management
  - Sqoop server is accessible via CLI, REST API, and Web UI
- Sqoop 2 is being actively developed
  - Implemented features are regarded as stable
  - Consider using Sqoop 1 unless you require a feature it lacks



How Does Sqoop Work?

- Sqoop is a client-side application that imports data using Hadoop MapReduce
- A basic import involves three steps orchestrated by Sqoop

1. Examine table details
2. Create and submit job to cluster
3. Fetch records from table and write this data to HDFS



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

- 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

- 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

- 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

- 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

- 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

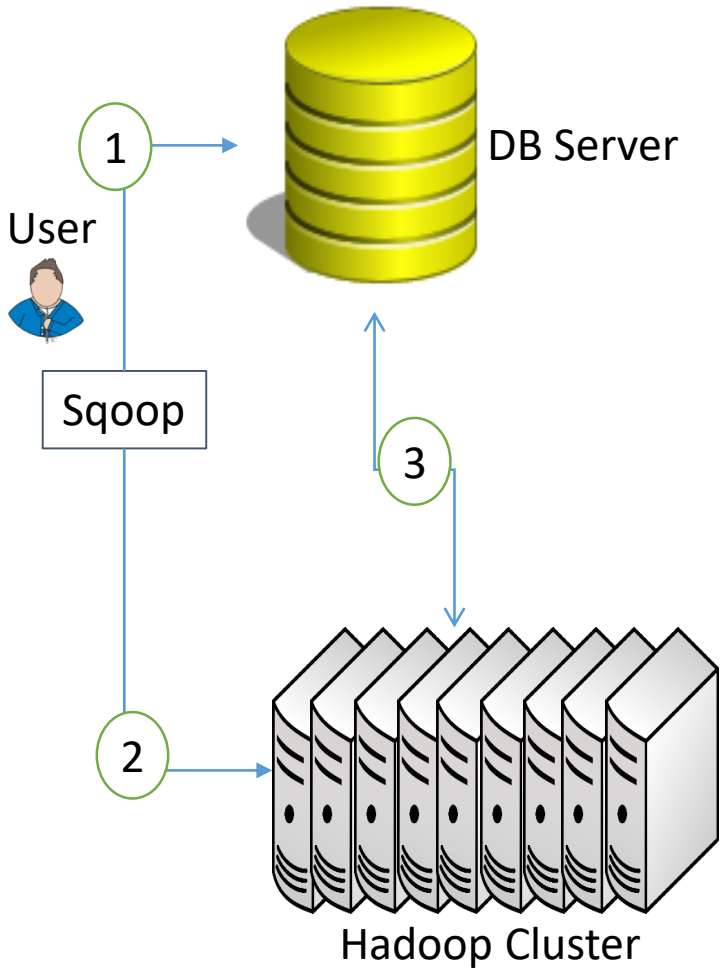
- By default, Sqoop typically imports data using four parallel tasks (called mappers)
  - Increasing the number of tasks might improve import speed
  - Caution: Each task adds load to your database server
- You can influence the number of tasks using the `-m` option
  - Sqoop views this only as a hint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

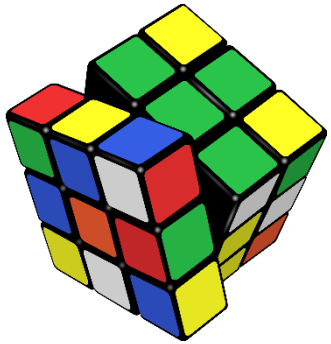
```
$ sqoop import --table accounts \
--connect jdbc:mysql://dbhost/loudacre \
--username dbuser --password pw \
-m 8
```

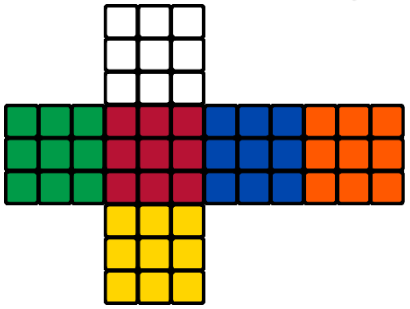
- Sqoop assumes all tables have an evenly-distributed numeric primary key
  - Sqoop uses this column to divide work among the tasks
  - You can use a different column with the `--split-by` option

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







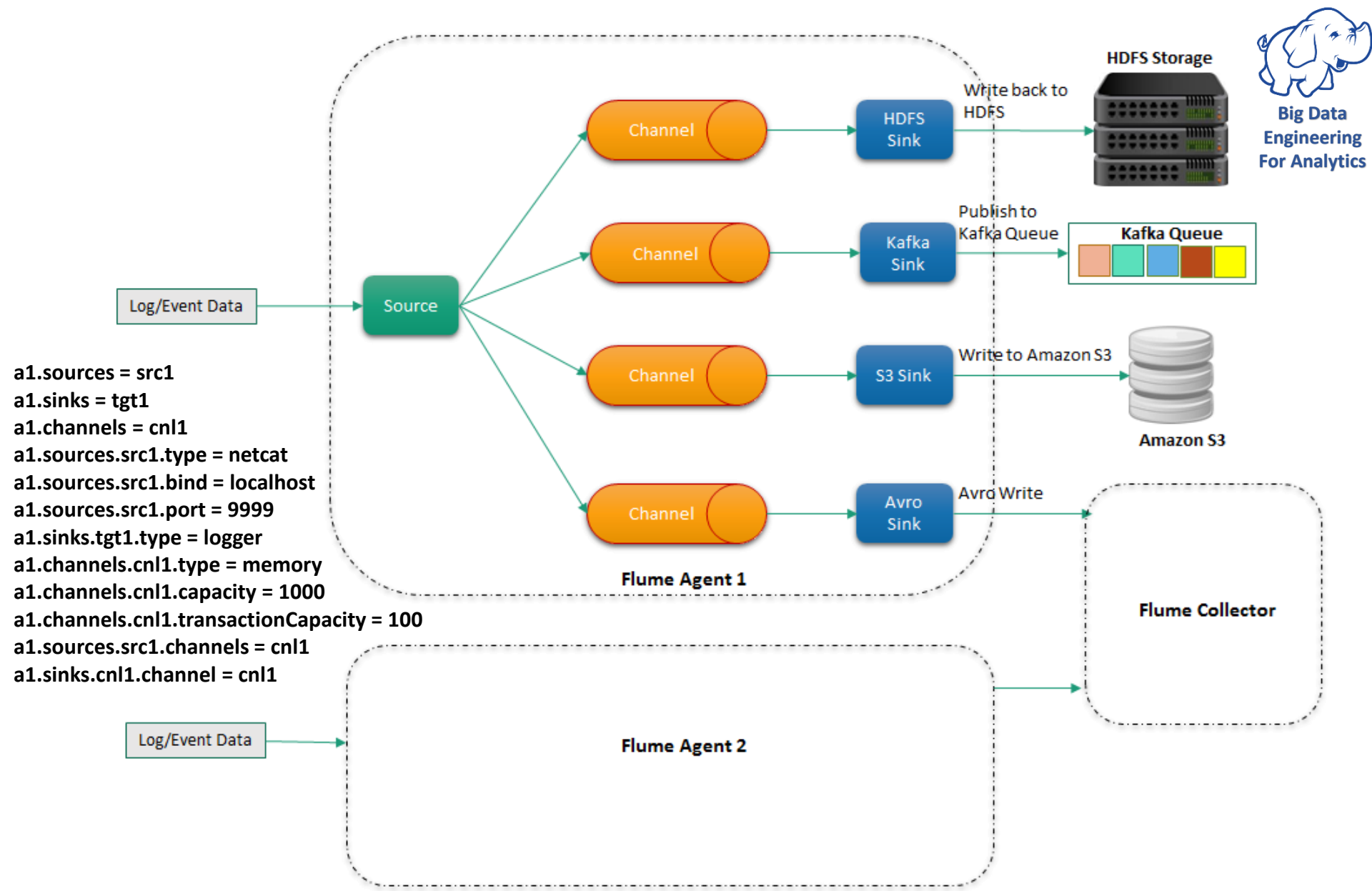
Apache Flume

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

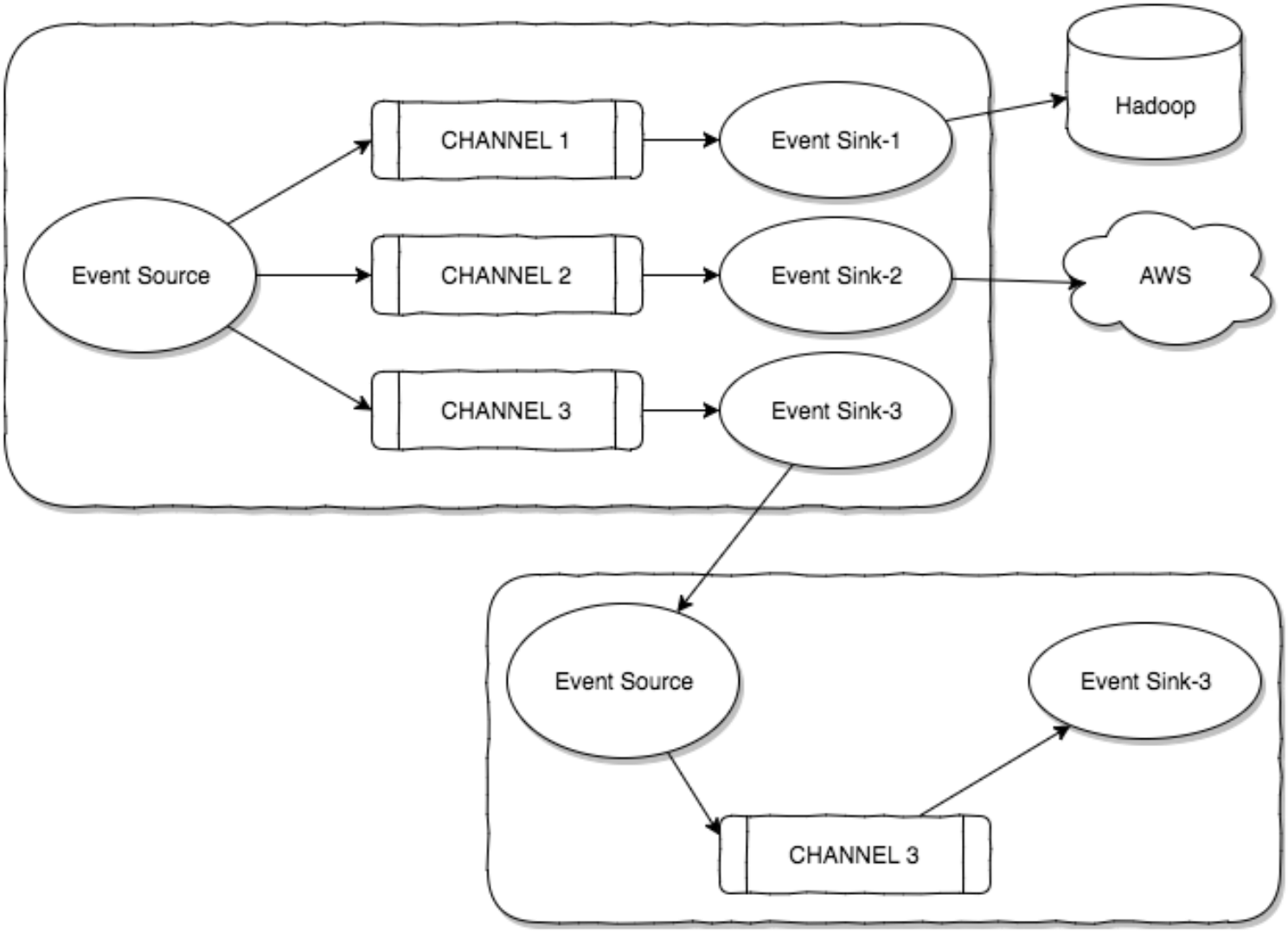
Edsger Dijkstra

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



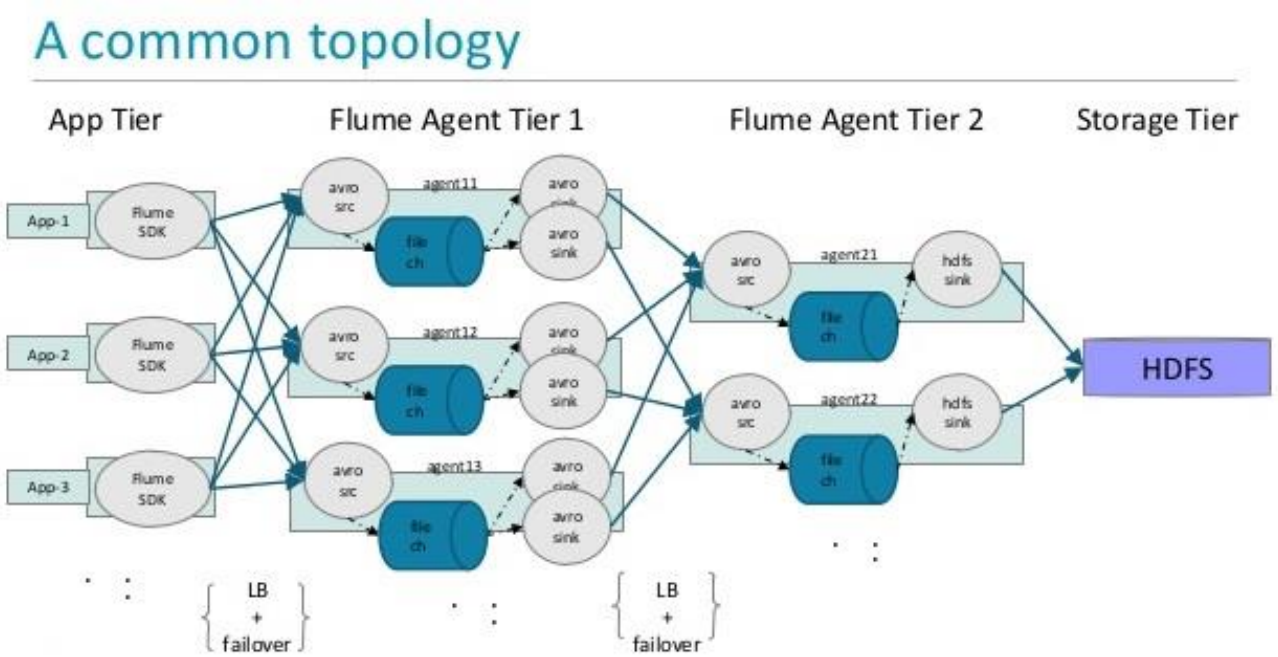
Flow Multiplexer



Flume Topology

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

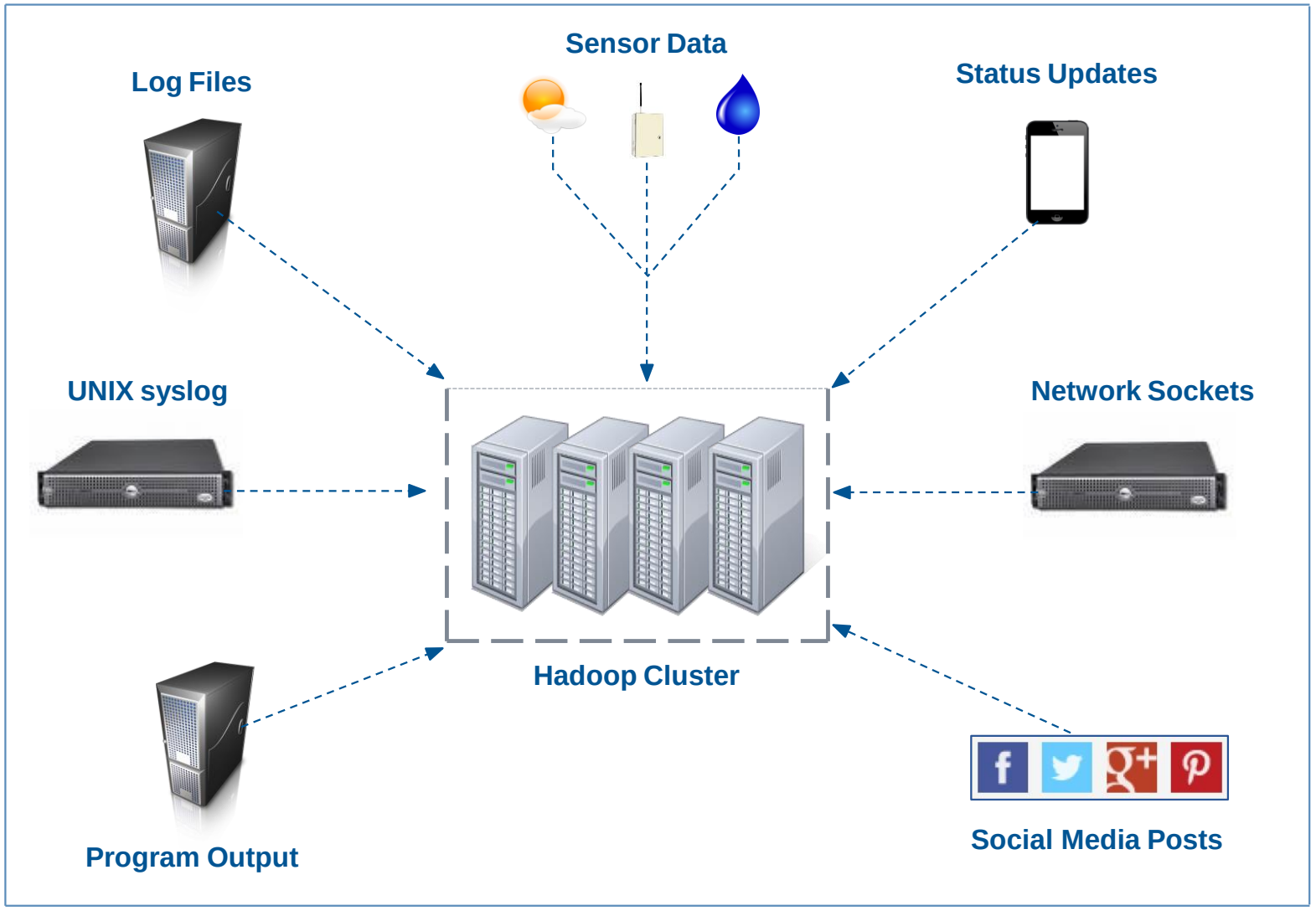


Flume's Design Goals

Big Data Engineering For Analytics

- **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

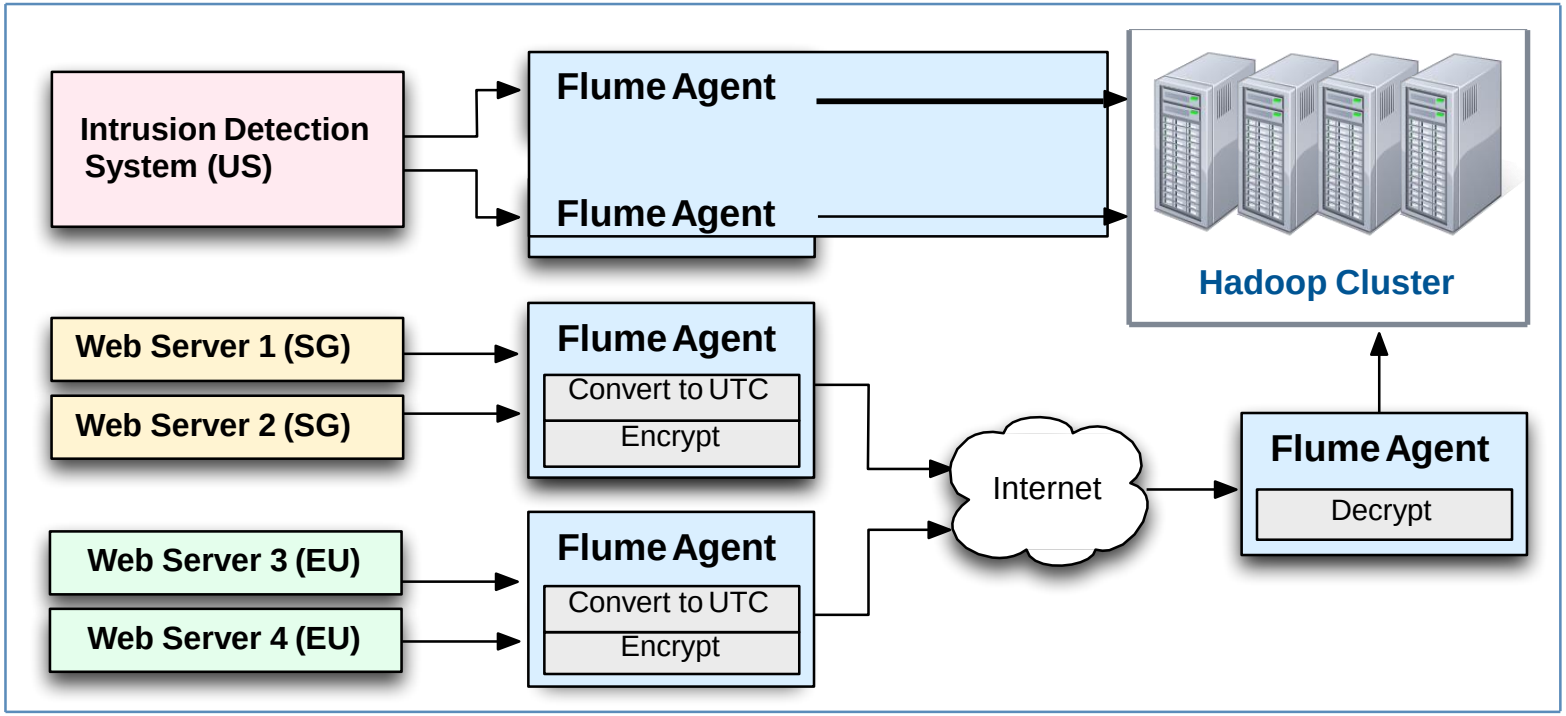
Common Data Sources



Large-Scale Deployment Example

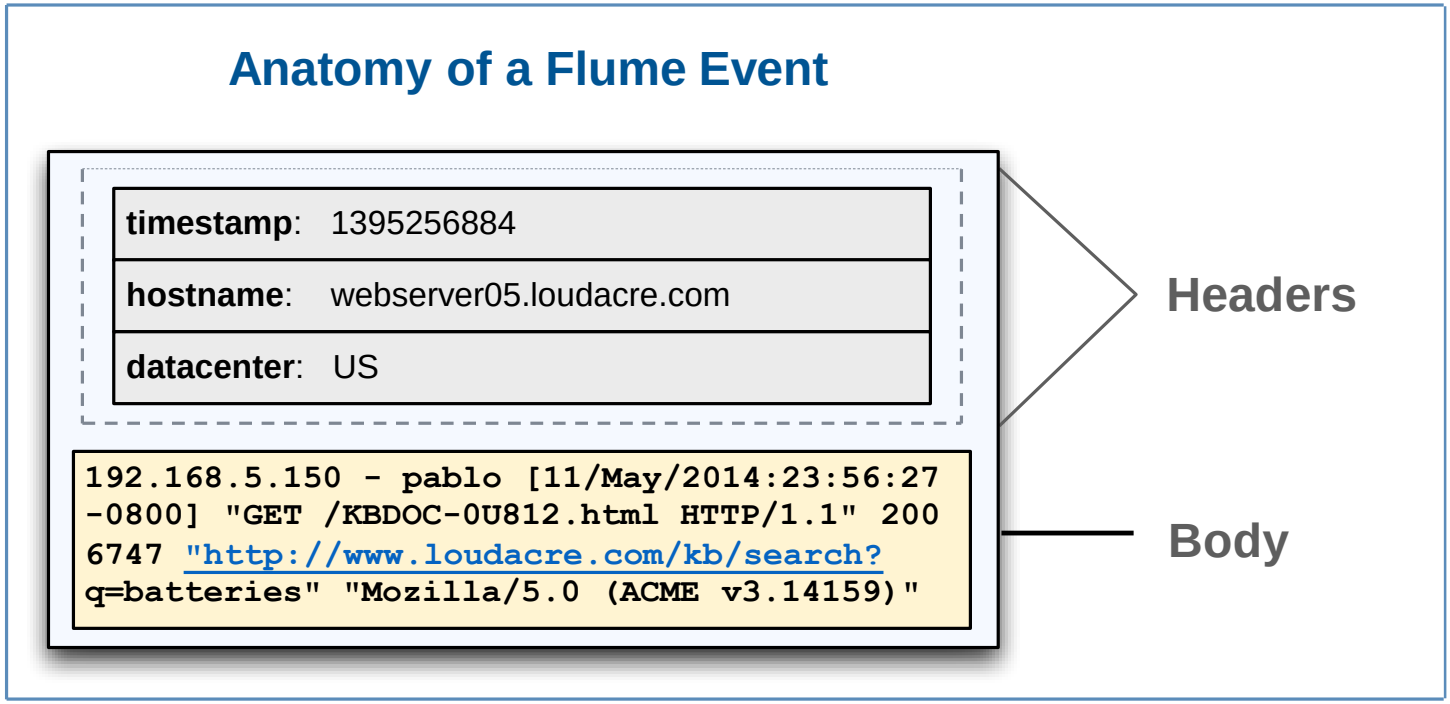
Flume collects data using configurable "agents"

- 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

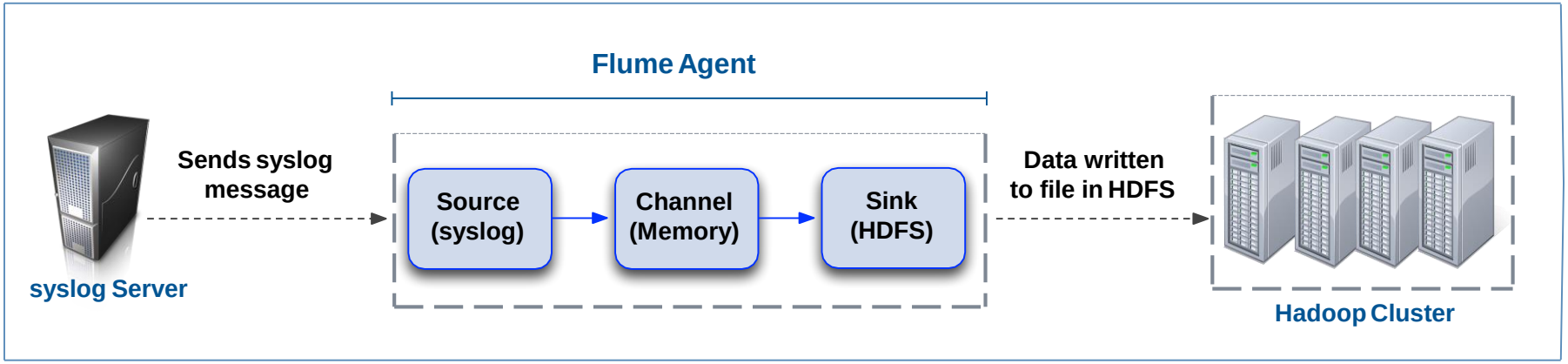
- An event is the fundamental unit of data in Flume
- Consists of a body (payload) and a collection of headers (metadata)
- Headers consist of name-value pairs
- Headers are mainly used for directing output



Flume Data Flow

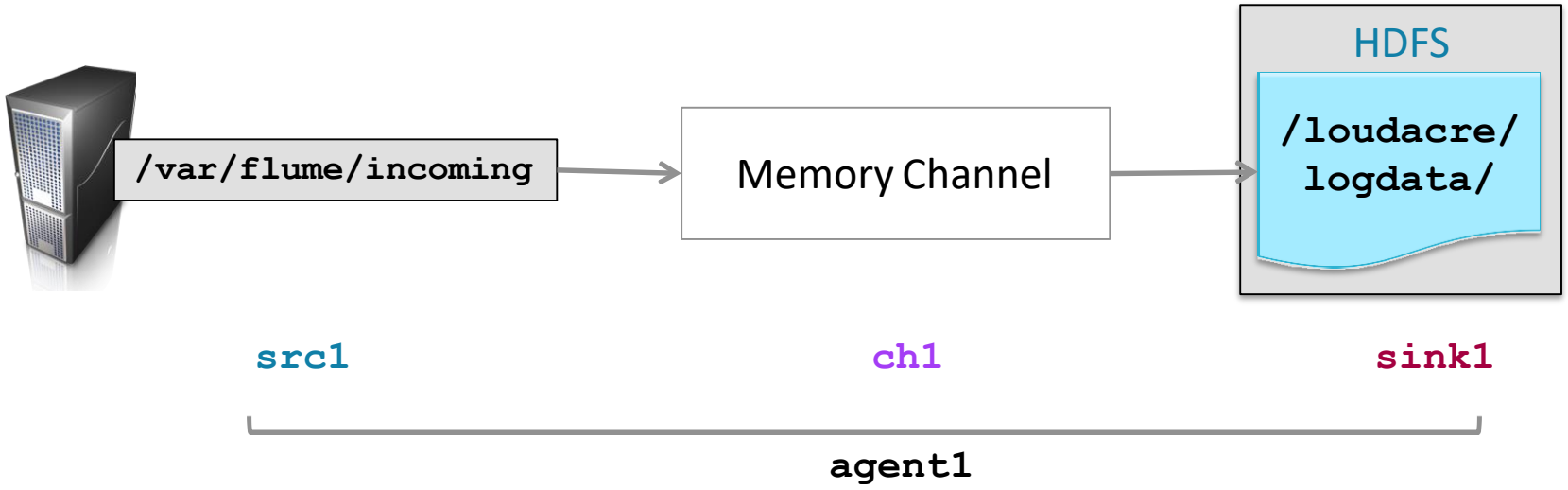
This diagram illustrates how syslog data might be captured to HDFS

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



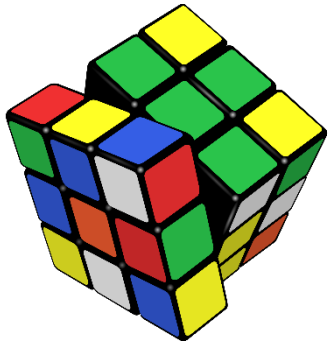
Flume Sources & Sinks

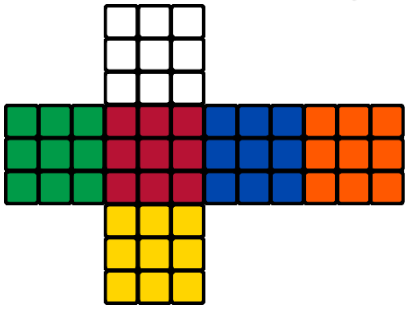
Big Data Engineering For Analytics

- 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

The art of programming is the art of organizing complexity. Probably I am very naive, but I also think I prefer to remain so, at least for the time being and perhaps for the rest of my life.

Edsger Dijkstra

Essential Points

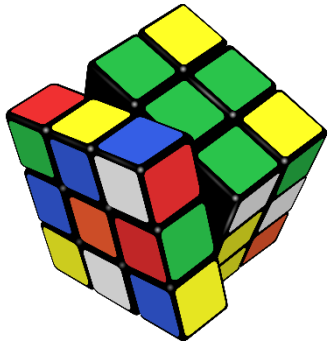
- Data Ingestion is critical and should be emphasized on all big data projects
  - With the huge volume of data coming into enterprises from various data sources, different challenges are encountered that can be solved using the patterns mentioned.
  - These patterns provide topologies to address multiple types of data formats and protocols, as well provide guidance to process, transform etc.,

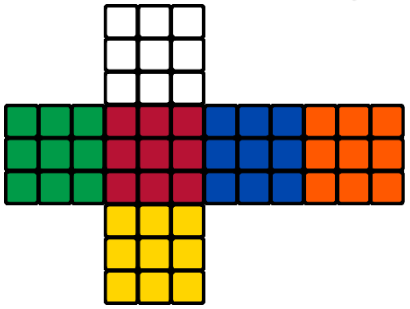
Essential Points

- Sqoop exchanges data between a database and the Hadoop cluster
  - Provides subcommands (tools) for importing, exporting, and more
- Tables are imported using MapReduce jobs
  - These are written as comma-delimited text by default
  - You can specify alternate delimiters or file formats
  - Uncompressed by default, but you can specify a codec to use
- Sqoop provides many options to control imports
  - You can select only certain columns or limit rows
  - Supports using joins in free-form queries
- Sqoop 2 is the next-generation version of Sqoop
  - Client-server design improves administration and resource management

Essential Points

- **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





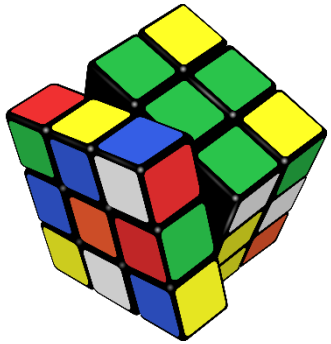
Reference

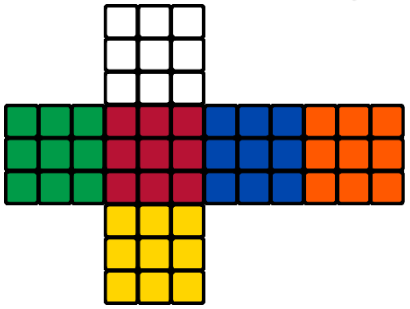
If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.

Edsger Dijkstra

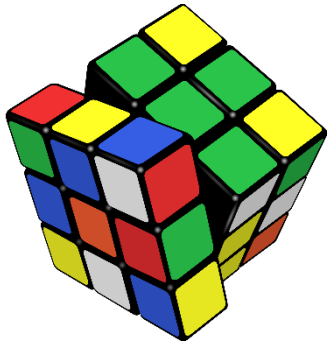
References

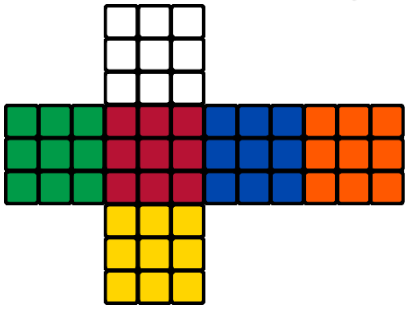
- Hadoop Essentials by Swizec Teller, Published by Packt Publishing, 2015.
- Sqoop User Guide
  - http://tiny.cloudera.com/sqoopuserguide
- Apache Sqoop Cookbook (published by O'Reilly)
  - http://tiny.cloudera.com/sqoopcookbook
- New Generation of Data Transfer Tools for Hadoop: Sqoop 2
  - http://tiny.cloudera.com/adcc05c
- Instant Apache Sqoop, by Ankit Jain, Published by Packt Publishing, 2013





Appendix – Data Ingestion Patterns





Common Data Sources and Ingestion Layer Patterns

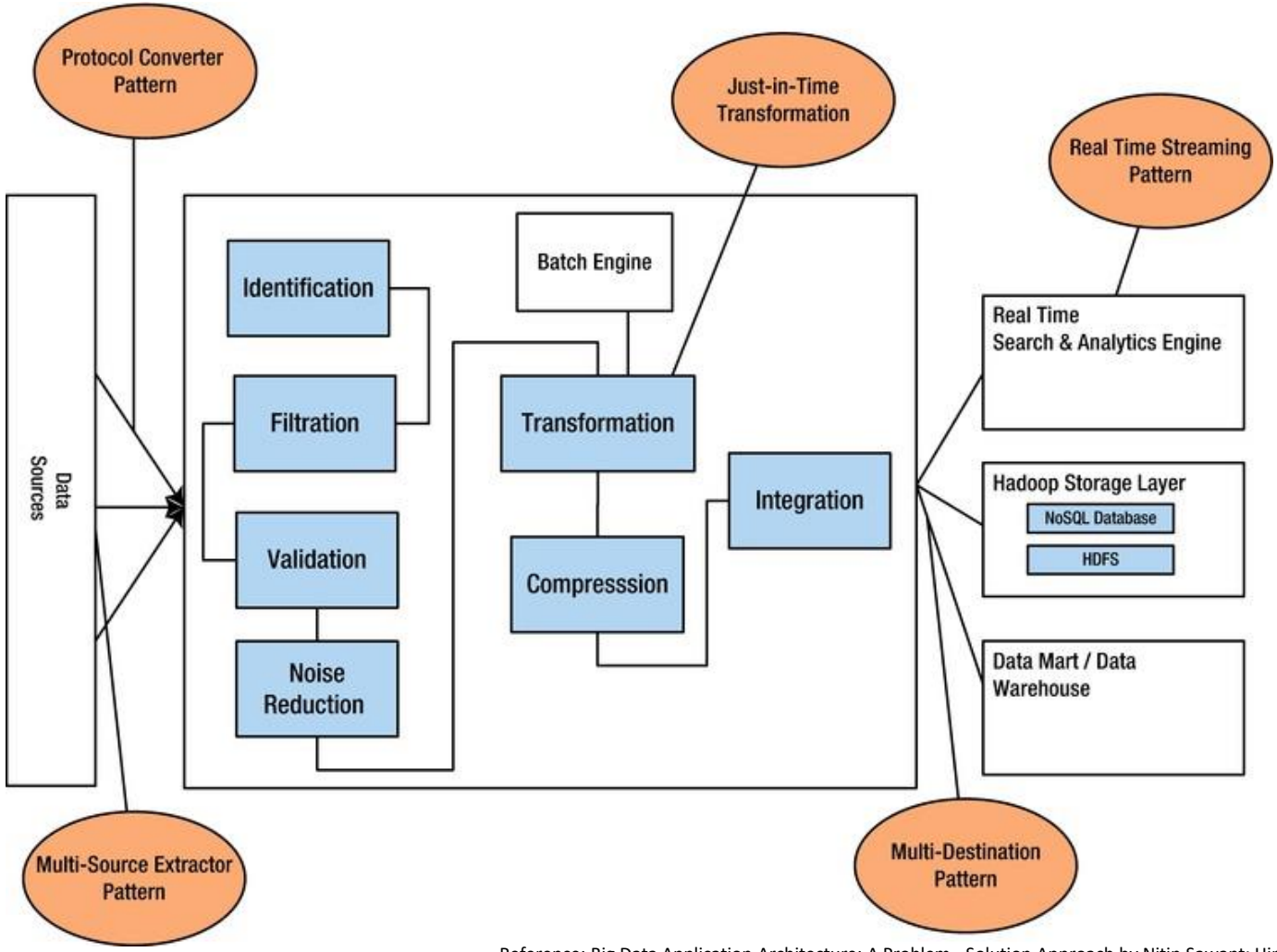
Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated.

Edsger Dijkstra

Common Interaction Patterns

- Request/response pattern
  - It's used when the client must have an immediate response or wants the service to complete a task without delay.
- Request/acknowledge pattern
  - The request/acknowledge usually sends a request and collects back an acknowledgment. This acknowledgement can be used to make subsequent requests, perhaps to check the status of the initial request or get a final response.
- Publish/subscribe pattern
  - This is a common pattern with message-based data systems; The messages are often sent to a topic, which is a logical grouping for messages. Next, the message is sent to all the consumers subscribing to that topic.
- One-way pattern
  - This interaction pattern is commonly found in cases where the system making the request doesn't need a response.
- Stream pattern
  - This pattern is very interesting and powerful: ingesting a stream of data and producing another stream.

Data Ingestion Layer And Associated Patterns

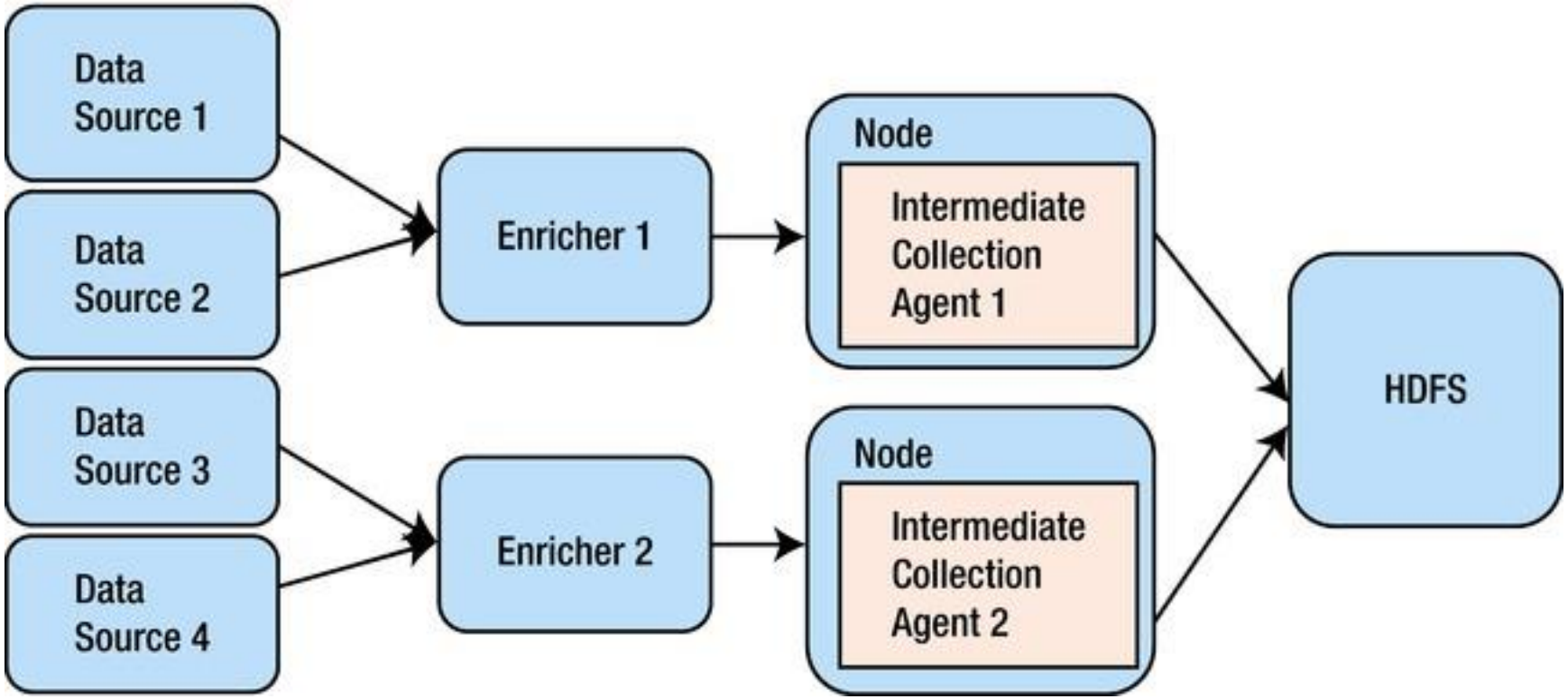


Reference: Big Data Application Architecture: A Problem - Solution Approach by Nitin Sawant; Himanshu Shah, 2013

Common Data-ingestion And Streaming Patterns

- **Multisource Extractor Pattern:**
  - An approach to ingest multiple data source types in an efficient manner.
- **Protocol Converter Pattern:**
  - Employs a protocol mediator to provide abstraction for the incoming data from the different protocol layers.
- **Multi-destination Pattern**:
  - Used in a scenario where the ingestion layer has to transport the data to multiple storage components like DFS, data marts, or real-time analytics engines.
- **Just-in-Time Transformation Pattern:**
  - Large quantities of unstructured data can be uploaded in a batch mode using traditional ETL (extract, transfer and load) tools and methods.
  - Data is transformed only when required to save compute time.
- **Real-Time Streaming patterns:**
  - Real-time ingestion and analysis of the in-streaming data is required for instant analysis of data.

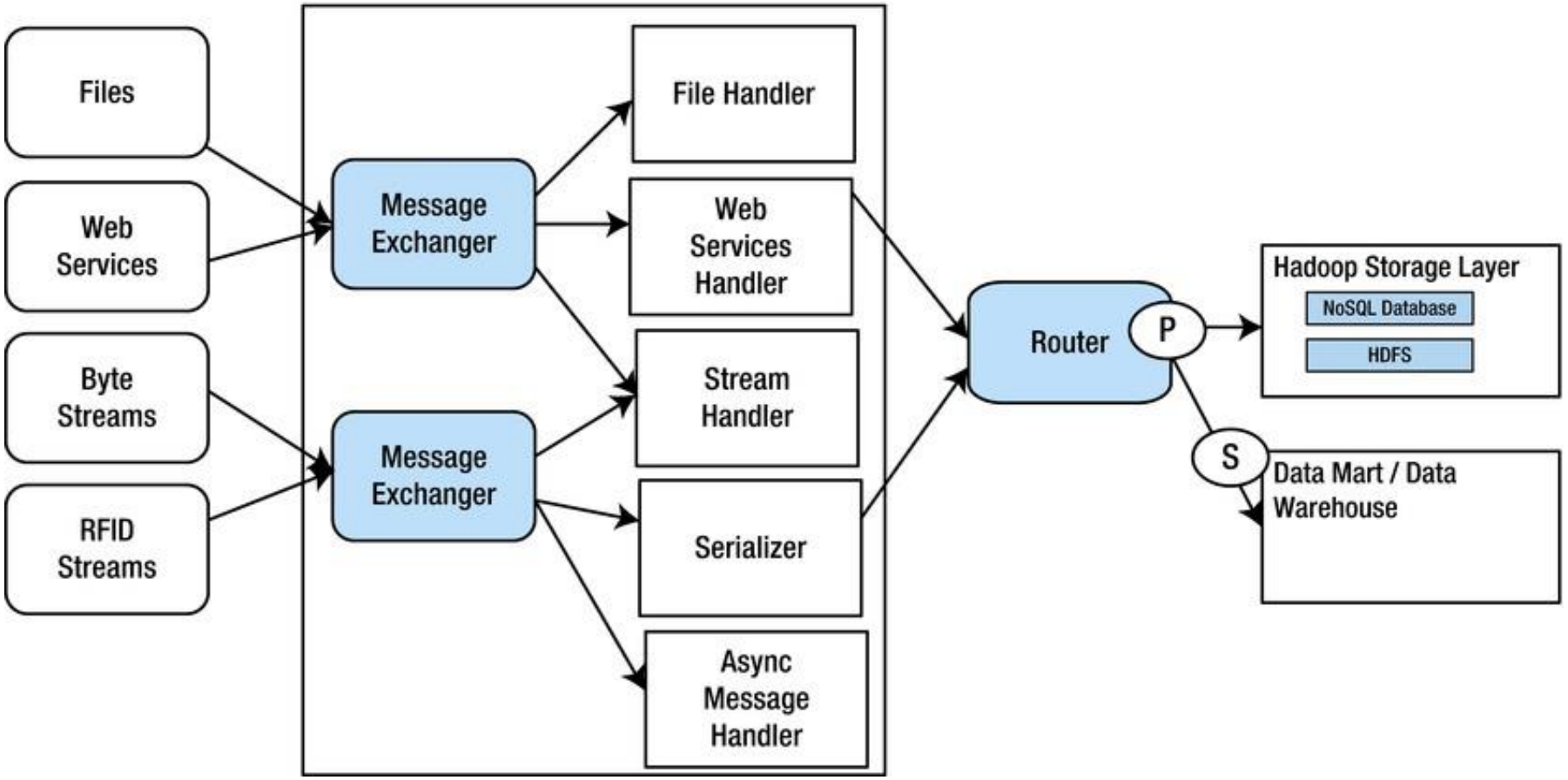
Multisource Extractor Pattern



Multisource Extractor Pattern

- Used in enterprises that have large collections of unstructured data need to investigate these disparate datasets and non-relational databases (for example, NoSQL, Cassandra, and so forth);
  - Big data operators employ ***enrichers*** to do initial data aggregation and cleansing.
  - An *enricher* reliably transfers files, validates them, reduces noise, compresses and transforms from a native format to an easily interpreted representation
  - Once the files are processed by enrichers, they are transferred to a cluster of ***intermediate collectors*** for final processing and loading to destination systems.

Protocol Converter Pattern



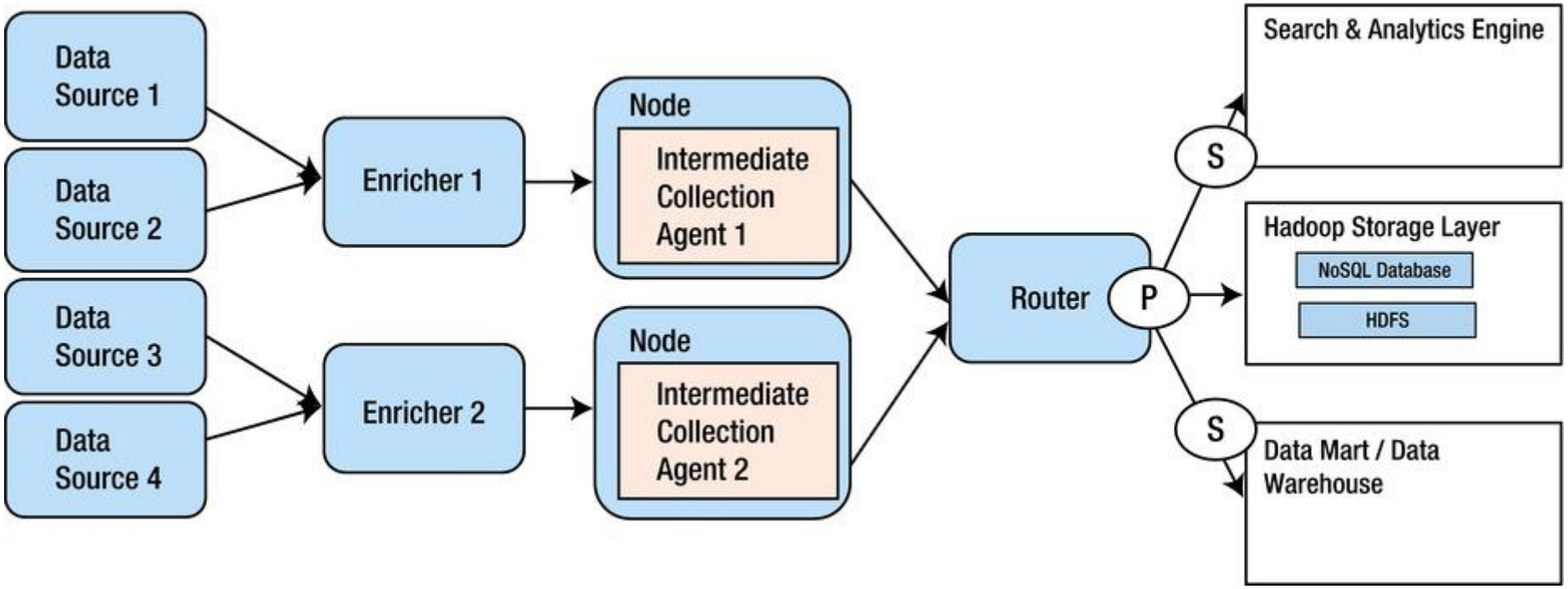
Protocol Converter Pattern Services

- **Message Exchanger:** The messages could be synchronous or asynchronous depending on the protocol used for transport. A typical example is a web application information exchange over HTPP and the JMS-like message oriented communication that is usually asynchronous.
- **Stream Handler:** This component recognizes and transforms data being sent as byte streams or object streams—for example, bytes of image data, PDFs, and so forth.
- **File handler:** This component recognizes and loads data being sent as files - for example, FTP.
- **Web Services Handler:** This component defines the manner of data population and parsing and translation of the incoming data into the agreed-upon format—for example, REST WS, SOAP-based WS, and so forth.
- **Async Handler:** This component defines the system used to handle asynchronous events—for example, MQ, Async HTTP, and so forth.
- **Serializer:** The serializer handles incoming data as *Objects* or complex types over RMI (remote method invocation)—for example, EJB components. The object state is stored in databases or *flat files*.

Protocol Converter Pattern Steps

1. Identifies multiple channels of incoming event.
2. Identifies polydata structures.
3. Provides services to mediate multiple protocols into suitable sinks.
4. Provides services to interface binding of external system containing several sets of messaging patterns into a common platform.
5. Provides services to handle various request types.
6. Provides services to abstract incoming data from various protocol layers.
7. Provides a unifying platform for the next layer to process the incoming data

Multi-destination Pattern



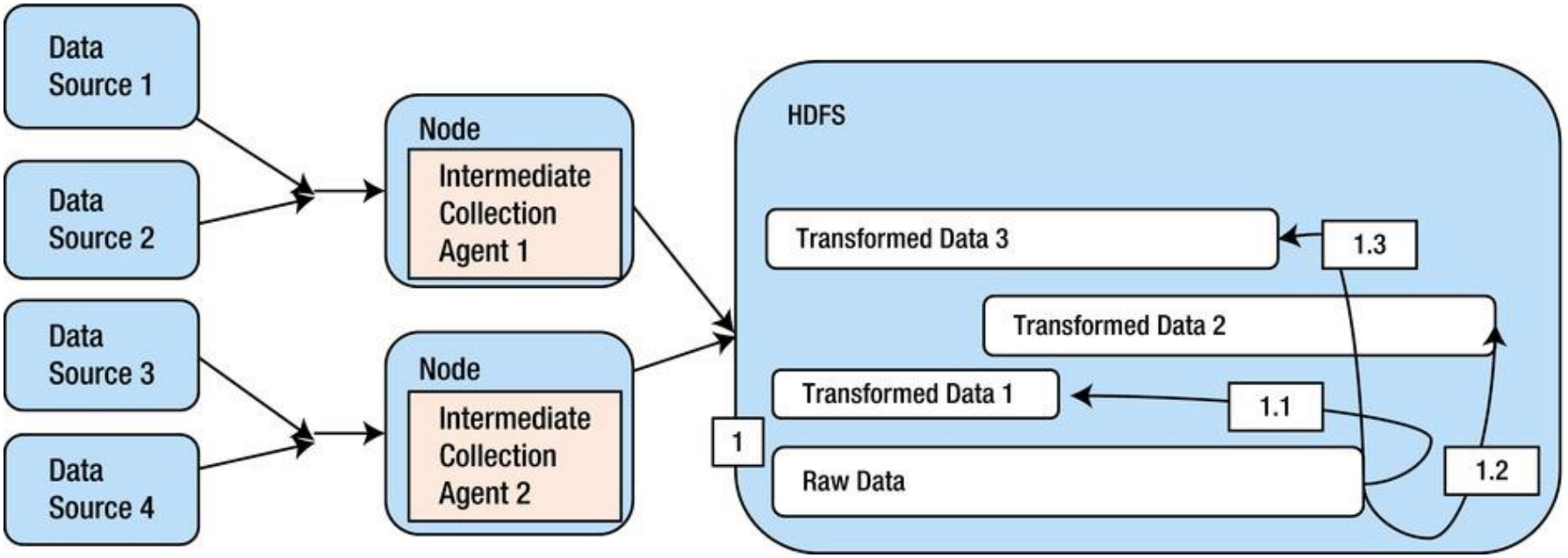
Multi-destination Pattern

This pattern solves some of the problems of ingesting and storing huge volumes of data:

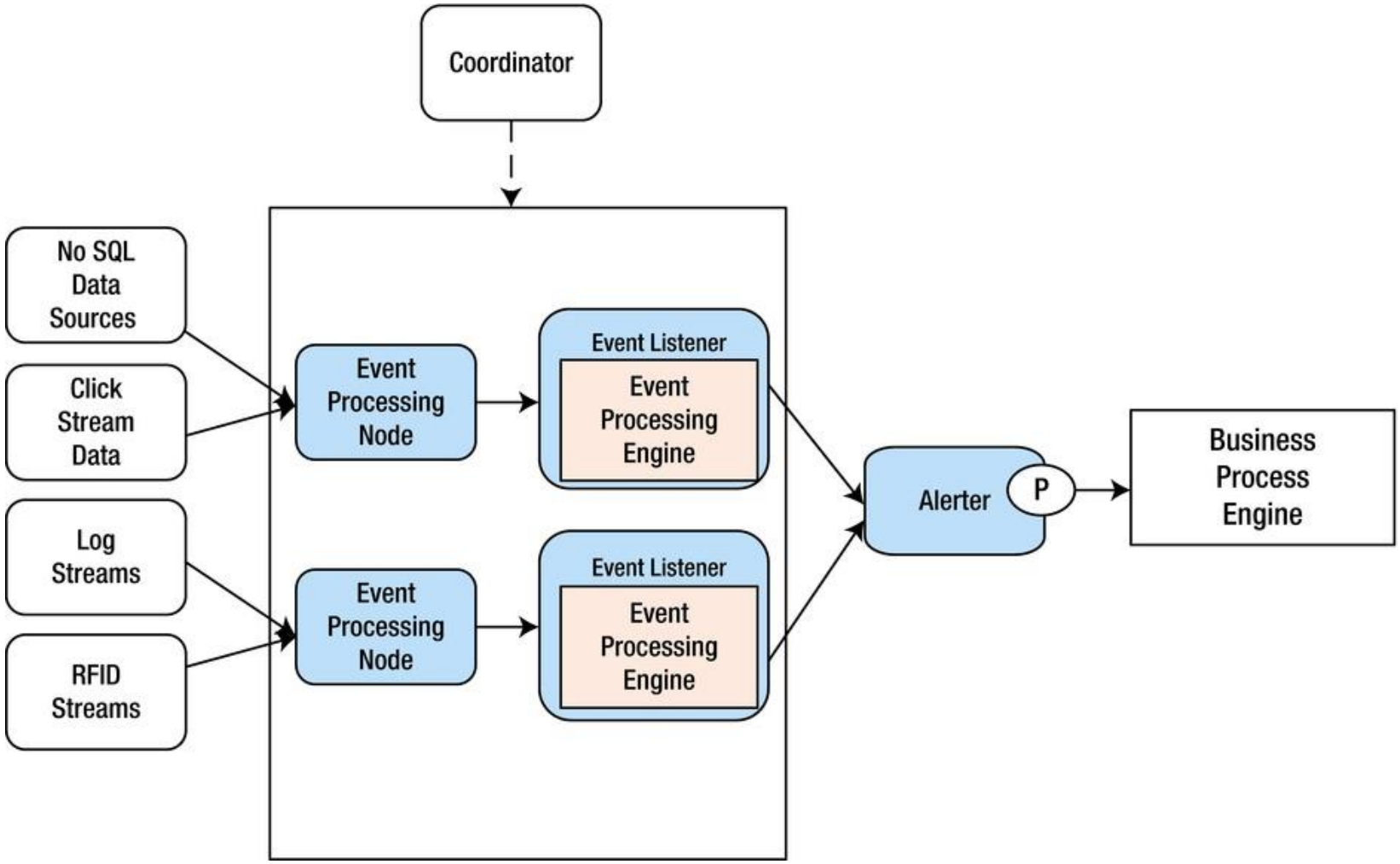
- Splits the cost of storage by dividing stored data among traditional storage systems and HDFS.
- Provides the ability to partition the data for flexible access and processing in a decentralized fashion.
- Due to replication on the HDFS nodes, there is no “data regret.”
- Because each node is self-sufficient, it’s easy to add more nodes and storage without delays.
- Decentralized computation at the data nodes without extraction of data to other tools.
- Allows use of simple query languages like Hive and Pig alongside the giants of traditional analytics.

Just-in-Time Transformation Pattern

- Ingest all the raw data into HDFS and then run dependent preprocessing; Basic validations can be performed as part of preprocessing on data being ingesting batch jobs.



Real-Time Streaming Pattern



Real-Time Streaming Pattern

- It should be self-sufficient and use local memory in each processing node to minimize latency.
- It should have a share-nothing architecture—that is, all nodes should have atomic responsibilities and should not be dependent on each other. .
- It should provide a simple API for parsing the real time information quickly.
- The atomicity of each of the components should be such that the system can scale across clusters using commodity hardware.
- There should be no centralized master node. All nodes should be deployable with a uniform script.