Aalto University
School of Science

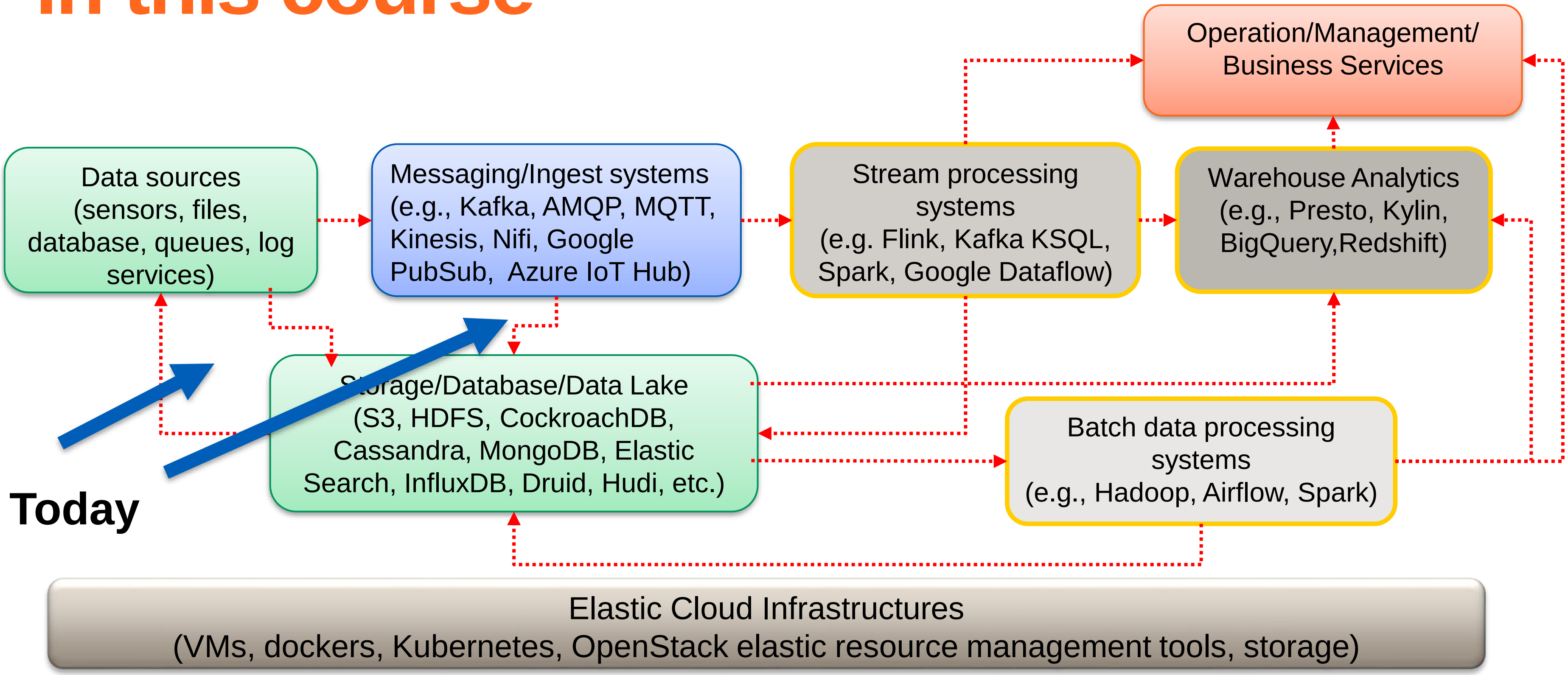
# Big Data Ingestion

*Hong-Linh Truong*
*Department of Computer Science*
*linh.truong@aalto.fi, https://rdsea.github.io*

# Learning objectives

- Understand the overall design of data ingestion
- Study common tasks in data ingestion
- Understand and design efficient, robust data ingestion pipelines/processes
- Learn existing technologies/frameworks for your own design

# Big data at large-scale: the big picture in this course



Operation/Management/ Business Services

Data sources (sensors, files, database, queues, log services)

Messaging/Ingest systems (e.g., Kafka, AMQP, MQTT, Kinesis, Nifi, Google PubSub, Azure IoT Hub)

Stream processing systems (e.g. Flink, Kafka KSQL, Spark, Google Dataflow)

Warehouse Analytics (e.g., Presto, Kylin, BigQuery,Redshift)

Storage/Database/Data Lake (S3, HDFS, CockroachDB, Cassandra, MongoDB, Elastic Search, InfluxDB, Druid, Hudi, etc.)

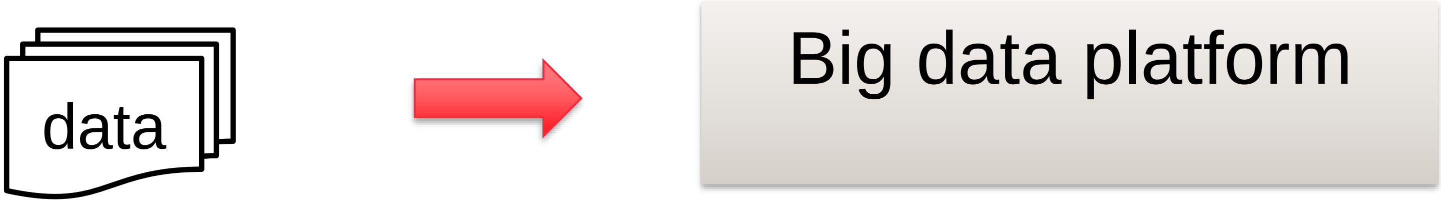
Batch data processing systems (e.g., Hadoop, Airflow, Spark)

Today

Elastic Cloud Infrastructures (VMs, dockers, Kubernetes, OpenStack elastic resource management tools, storage)

# Ingest big data  into platforms

Data ingestion: Move data from different sources into the big data platform



data

Big data platform

**e.g.**

- **logs of machines**
- **sell receipt transaction records**
- **IoT measurements**

**Two important aspects:**

- **requirements and tasks**
- **architectures, pipelines and service models**

**Reusability and extensibility are very important!**

# Big Data Ingestion

- **Relation with ETL (Extract, Transform, Load)**
  - During ingestion, some transformation tasks might be needed
  - ETL has many operations to deal with the semantics/syntax of data and the business of data
- **Transformation within ingestion**
- **Transformation done within the (target) platform**

**Correctness and quality assurance are hard!**

# Fundamental ingestion models

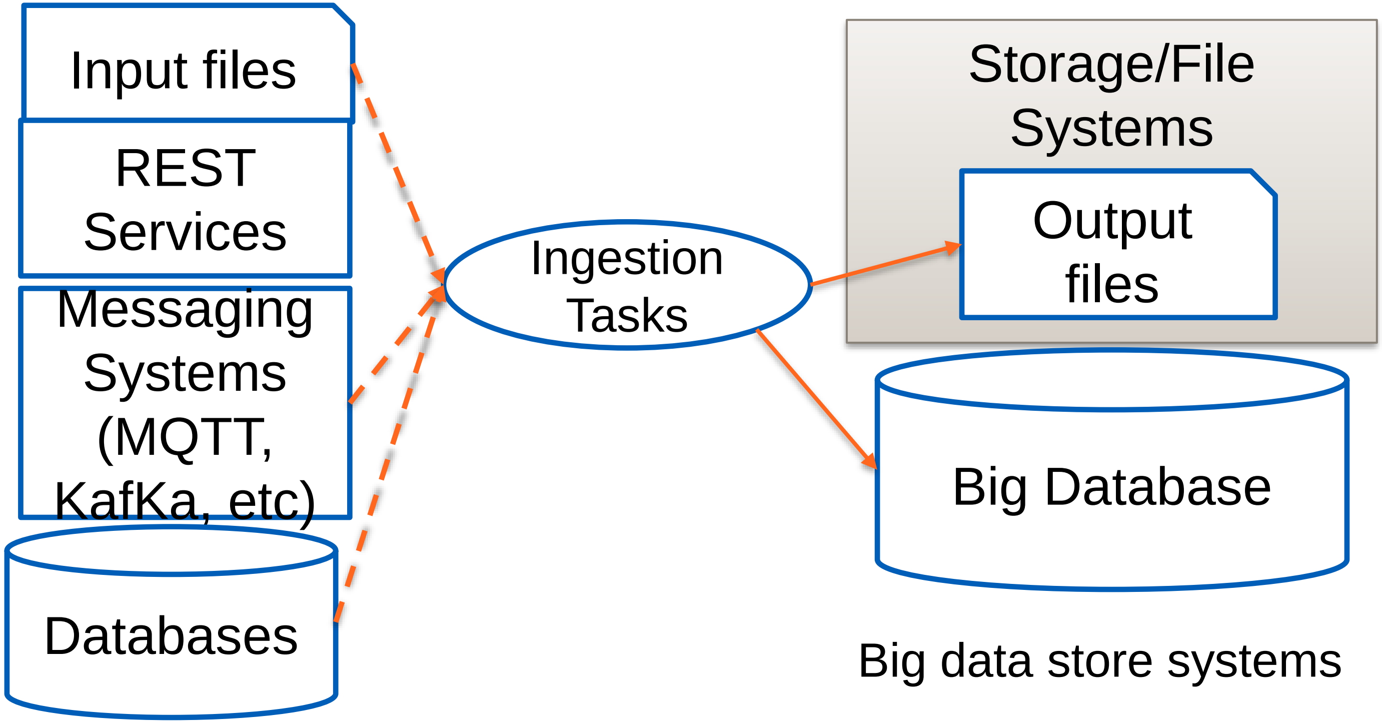
- **Batch ingestion**
  - Data is in files
  - Ingestion can be done in batches of files or batches of parts of files
- **Files**
  - CSV, Text, JSON, ARVO
  - Other typical formats (video, images, etc.)

- **(Near) real-time ingestion**
  - Data is encapsulated into messages
  - Ingest data as soon as the data is available
  - Messaging systems are needed
- **Messages**
  - Text/CSV/JSON, ARVO
  - Application-specific designs

# Data source and sinks

**Data sources**

**Data sinks**



Input files

REST Services

Messaging Systems (MQTT, KafKa, etc)

Databases

Ingestion Tasks

Storage/File Systems

Output files

Big Database

Big data store systems

**Big data platform examples:**

Hadoop File systems
Google Storage
Amazon Storage

Druid
Google BigQuery
Hive
MongoDB
ElasticSearch
Cassandra
InfluxDB
Hudi

# Requirements from V* of big data

- **Requirements from access API and protocols**
  - REST API, ODBC, SFTP, specific client libs
  - MQTT, AMQP, CoAP, HTTP, …
- **Requirements from data**
  - structured, unstructured and semi-structured
  - speed, volume, accuracy, confidentiality, data regulation
- **How deep a platform can support?**
  - able to go into inside of data elements (understanding the syntax and semantics of data)?

# Ingestion tasks: common tasks and requirements

# Main tasks in ingestion

- **Key categories of tasks**
  - Data access and extraction
  - Data routing
  - Data wrangling
  - Data storing
  - Quality assurance/governance (quality check, anonymizing data)
- **Customer/user tasks vs platform tasks**
- **Other supports: compression, end-to-end security**

# Data access  and extraction tasks

- **Access**
  - Obtaining/copy data from sources, change data capture (CDC)
  - Often built based on common protocols and APIs
  - Reusability is important!
- **Encryption, masking/anonymization**
  - Might need to be done when accessing and extracting data
  - Also during transfers of data
  - data security requirements, personally identifiable information

# Change data capture becomes important for big data ingestion

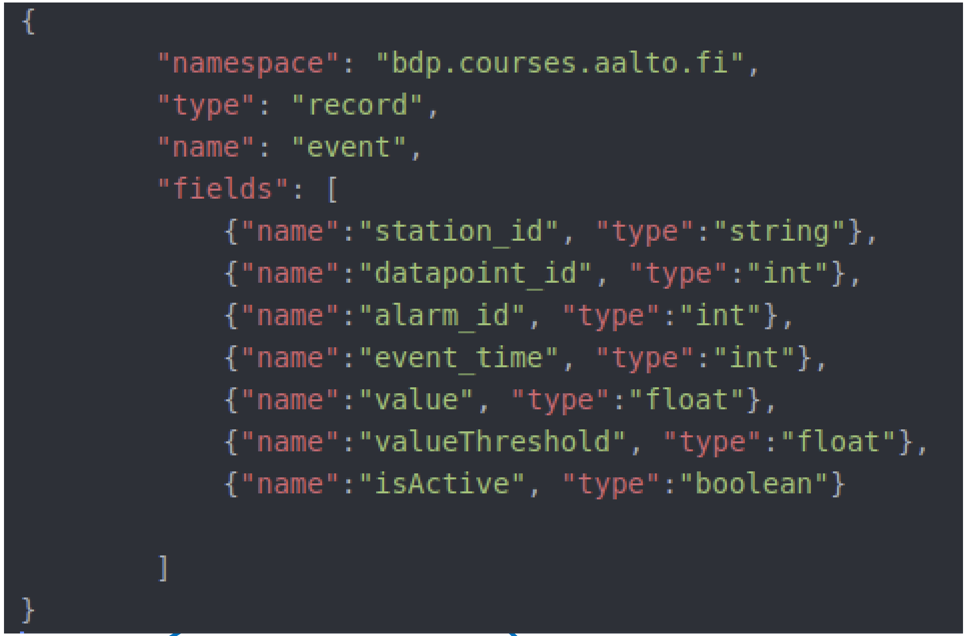
- **The principles:**
  - Capture and ingest only new data by listening data changes
  - Leverage many features of databases (update, query, insert operations), data stream offsets and status notification (e.g., the availability of new files)
- **You see implementation in different tools like Redhat Debezium, Hudi DeltaStreamer, Kafka connect**

# Dealing with data structures

- **Remember that the data sender and the receiver are diverse**
  - In many cases, they are not in the same organization
    - *You need to guarantee the message syntax and semantics*
- **Solutions**
  - Agreed in advance → in the implementation or with a standard
  - Know and use tools to deal with syntax differences
- **Understanding the syntax allows some automatic transformations/quality check**
- **But semantics are domain/application-specific**

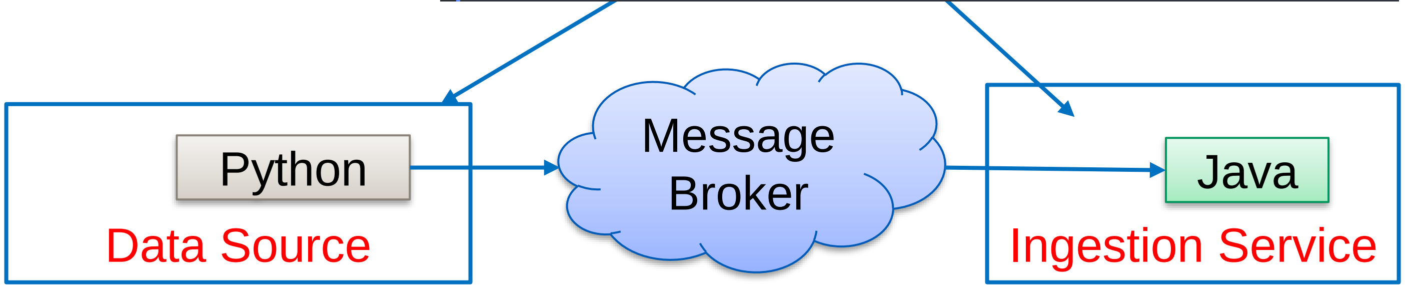
# Example: Arvo

**Syntax specification**
**https://avro.apache.org/**



```
{
        "namespace": "bdp.courses.aalto.fi",
        "type": "record",
        "name": "event",
        "fields": [
            {"name":"station_id", "type":"string"},
            {"name":"datapoint_id", "type":"int"},
            {"name":"alarm_id", "type":"int"},
            {"name":"event_time", "type":"int"},
            {"name":"value", "type":"float"},
            {"name":"valueThreshold", "type":"float"},
            {"name":"isActive", "type":"boolean"}

        ]
}
```

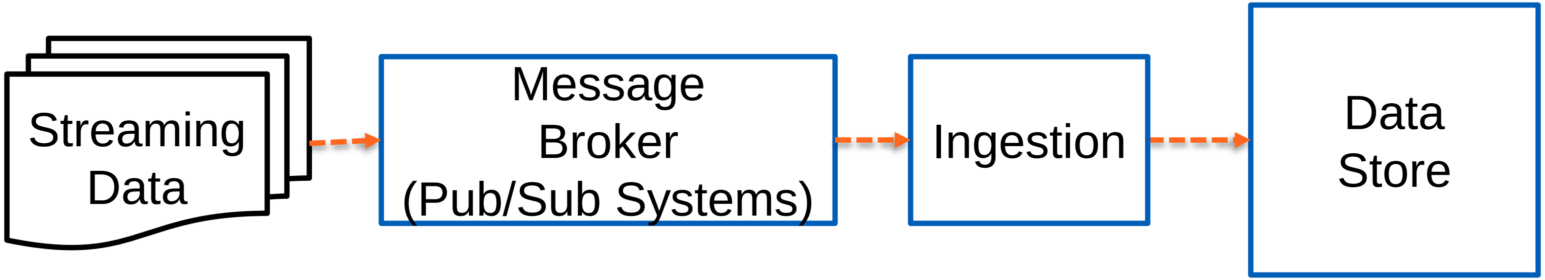


# Target file formats in big data storage

- **Parquet, https://parquet.apache.org/**
  - Columnar storage (optimizing for reading columns), big files, compression features
  - In Hadoop ecosystem/Spark (thus also available in Druid, Hudi), Azure, S3, etc.
- **ORC, https://orc.apache.org/**
  - Large-scale files, self-describing data and metadata, available in Hive, support ACID, multiple-level of indexes and complex types
- **Many big databases/storage and datalakes use them as the storage level**
  - Still allow SQL-style or other types of analytics

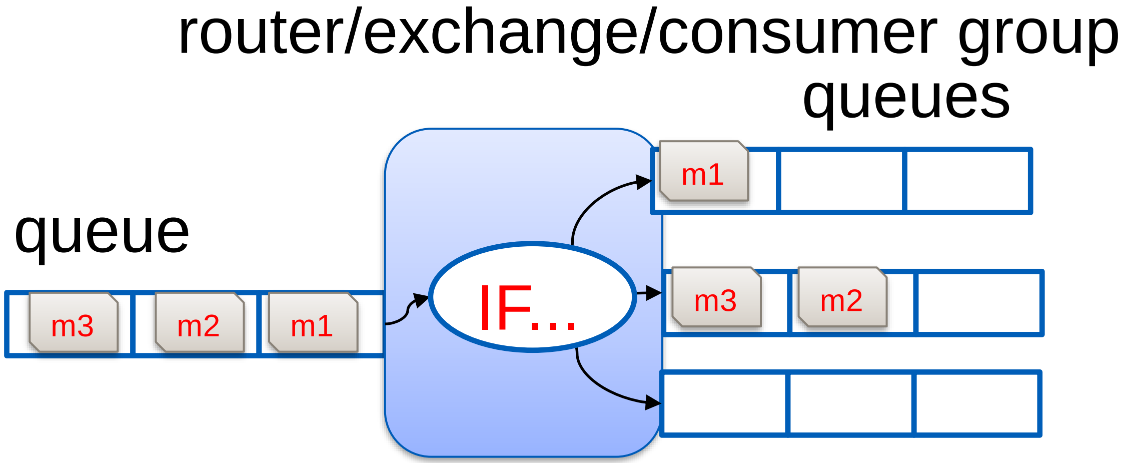
**Reading: an Uber blog bout file formats and performance: https://eng.uber.com/cost-efficiency-big-data/**

# How do we move streaming data into big data databases/storage?



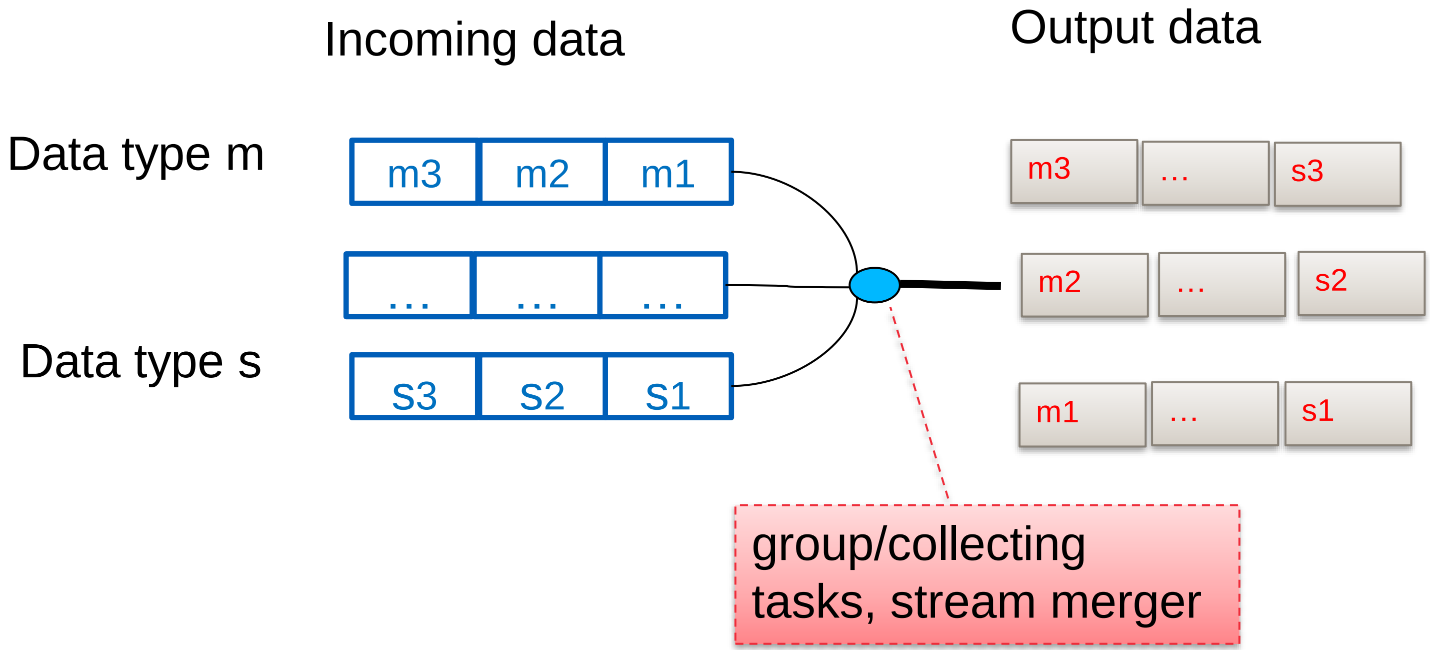
**Protocol**
**Data format**
**Message structure**
**Basic streaming data processing techniques**

# Use split tasks/distributor patterns to separate data for data parallelism processing



**Read the famous book: "Enterprise Integration Patterns" https://www.enterpriseintegrationpatterns.com/patterns/messaging/**

# Data routing: grouping data/collector pattern using stream merging/common topics



# Data wrangling

- **Convert data from one form to another**
  - Cleaning, filtering, merging and reshaping data
- **Require  access to the data!**
- **Key design choices:**
  - do you support it during the ingestion or after the ingestion?
  - as a platform provider: are you able to do this?

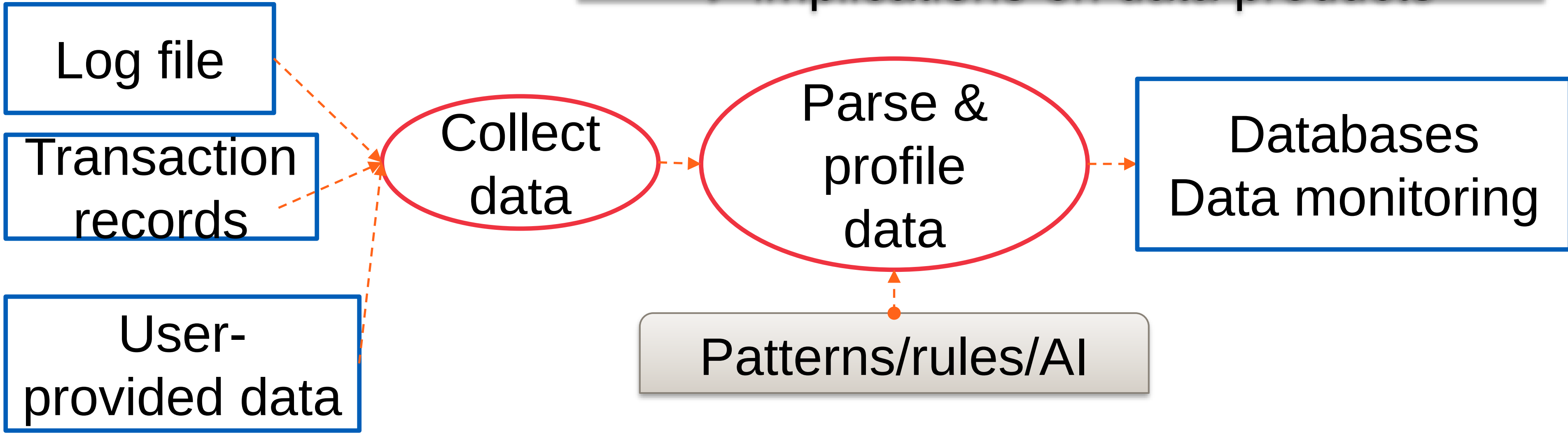
# Data wrangling

- **In the context of big data platforms**
  - Automatic data wrangling: write pipelines/programs which do the wrangling
- **Wrangling programs provided by customers**
  - Needs the platform to support debugging, monitoring and exception handling
  - Runtime management for wrangling
- **Wrangling programs provided by platforms**
  - Constraints in dealing with customer data

# Quality control/data regulation assurance

Responsible data: profiling, sampling, measuring quality and inspecting data → implications on data products

**Data sources**



Log file

Transaction records

User-provided data

Collect data

Parse & profile data

Patterns/rules/AI

Databases
Data monitoring

**Hot issues: misinformation, GDPR, data quality, inappropriate content**

# Examples: Logstash Grok – a kind of domain specific language?

Grok is for parsing unstructured log data

text patterns into something that matches your logs.

Grok pattern syntax: %{SYNTAX:SEMANTIC}

Regular and custom patterns

A lot of exiting patterns:

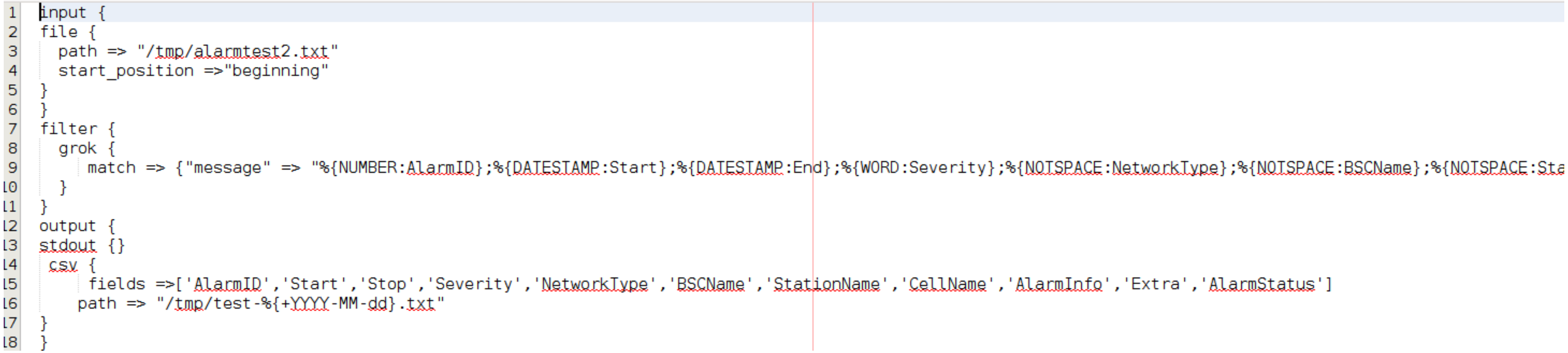
- https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

Debug Tools: http://grokdebug.herokuapp.com/

# Example with NETACT Log

29869;10/01/2017 00:57:56;;Major;PLMN-PLMN/BSC-xxxxxx/BCF-xxx/BTS-xxx;XYZ01N;ABC08;DEF081;BTS OPERATION DEGRADED;00 00 00 83 11 11;Processing

## Simple Grok

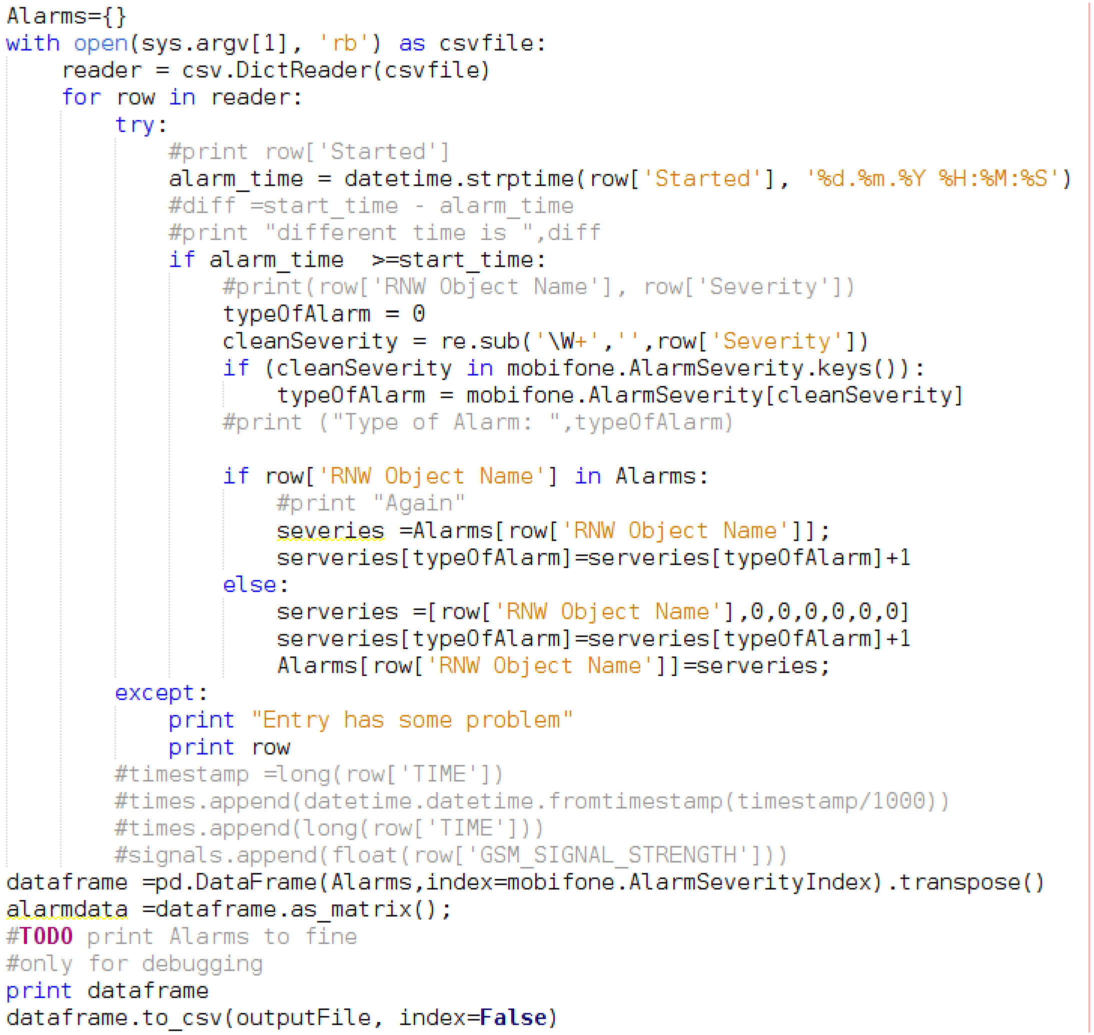


```
input {
file {
  path => "/tmp/alarmtest2.txt"
  start_position =>"beginning"
}
}
filter {
  grok {
     match => {"message" => "%{NUMBER:AlarmID};%{DATESTAMP:Start};%{DATESTAMP:End};%{WORD:Severity};%{NOTSPACE:NetworkType};%{NOTSPACE:BSCName};%{NOTSPACE:Sta
  }
}
output {
stdout {}
 csv {
     fields =>['AlarmID','Start','Stop','Severity','NetworkType','BSCName','StationName','CellName','AlarmInfo','Extra','AlarmStatus']
    path => "/tmp/test-%{+YYYY-MM-dd}.txt"
}
}
```

# Examples

**Write your own code with Pandas and Data frame?**

**Automatically generate code for wrangling?**



```
Alarms={}
with open(sys.argv[1], 'rb') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        try:
            #print row['Started']
            alarm_time = datetime.strptime(row['Started'], '%d.%m.%Y %H:%M:%S')
            #diff =start_time - alarm_time
            #print "different time is ",diff
            if alarm_time  >=start_time:
                #print(row['RNW Object Name'], row['Severity'])
                typeOfAlarm = 0
                cleanSeverity = re.sub('\W+','',row['Severity'])
                if (cleanSeverity in mobifone.AlarmSeverity.keys()):
                    typeOfAlarm = mobifone.AlarmSeverity[cleanSeverity]
                #print ("Type of Alarm: ",typeOfAlarm)

                if row['RNW Object Name'] in Alarms:
                    #print "Again"
                    severies =Alarms[row['RNW Object Name']];
                    serveries[typeOfAlarm]=serveries[typeOfAlarm]+1
                else:
                    serveries =[row['RNW Object Name'],0,0,0,0,0,0]
                    serveries[typeOfAlarm]=serveries[typeOfAlarm]+1
                    Alarms[row['RNW Object Name']]=serveries;
        except:
            print "Entry has some problem"
            print row
        #timestamp =long(row['TIME'])
        #times.append(datetime.datetime.fromtimestamp(timestamp/1000))
        #times.append(long(row['TIME']))
        #signals.append(float(row['GSM_SIGNAL_STRENGTH']))
dataframe =pd.DataFrame(Alarms,index=mobifone.AlarmSeverityIndex).transpose()
alarmdata =dataframe.as_matrix();
#TODO print Alarms to fine
#only for debugging
print dataframe
dataframe.to_csv(outputFile, index=False)
```

Aalto University
School of Science

# Ingestion tasks implemented as extensible, composable connectors

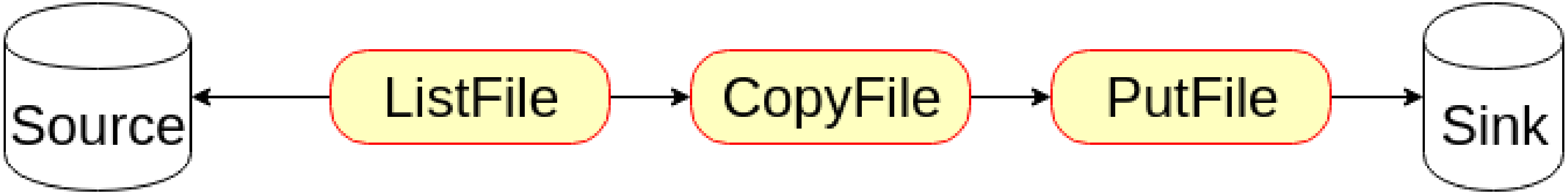
- **Basic tasks for big data ingestion can be used in different cases**
- **Support end-user tasks**
  - Platform enables the user to do many tasks through configurations
- **Enable pluggable approaches is important**
  - Input data plugin/component → filter/extract/convert → output data plugin/component
- **Data compression and security must be considered**

# Ingestion is not a single task!

# Ingestion pipelines/processes: architectures and tools

# Complex deployment and composition models

- **Understanding strong dependencies between protocols/APIs, security, performance and management**

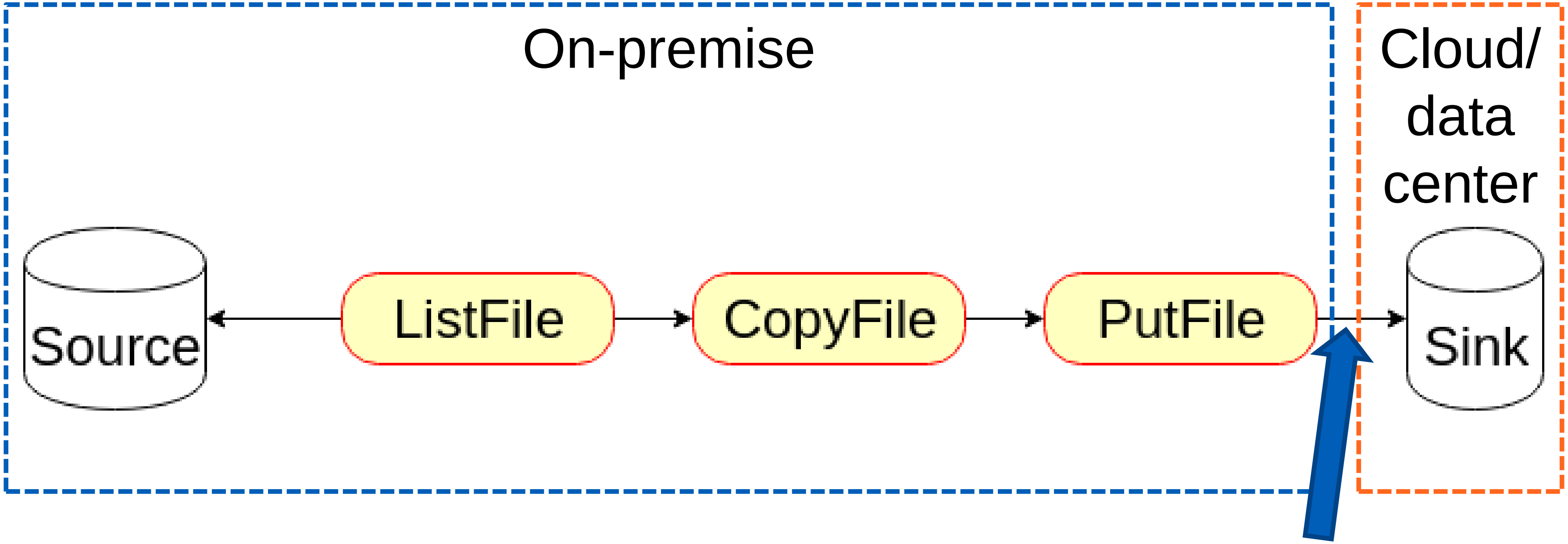


**Customer**

**Ingestion pipeline developer ( for whom?)**

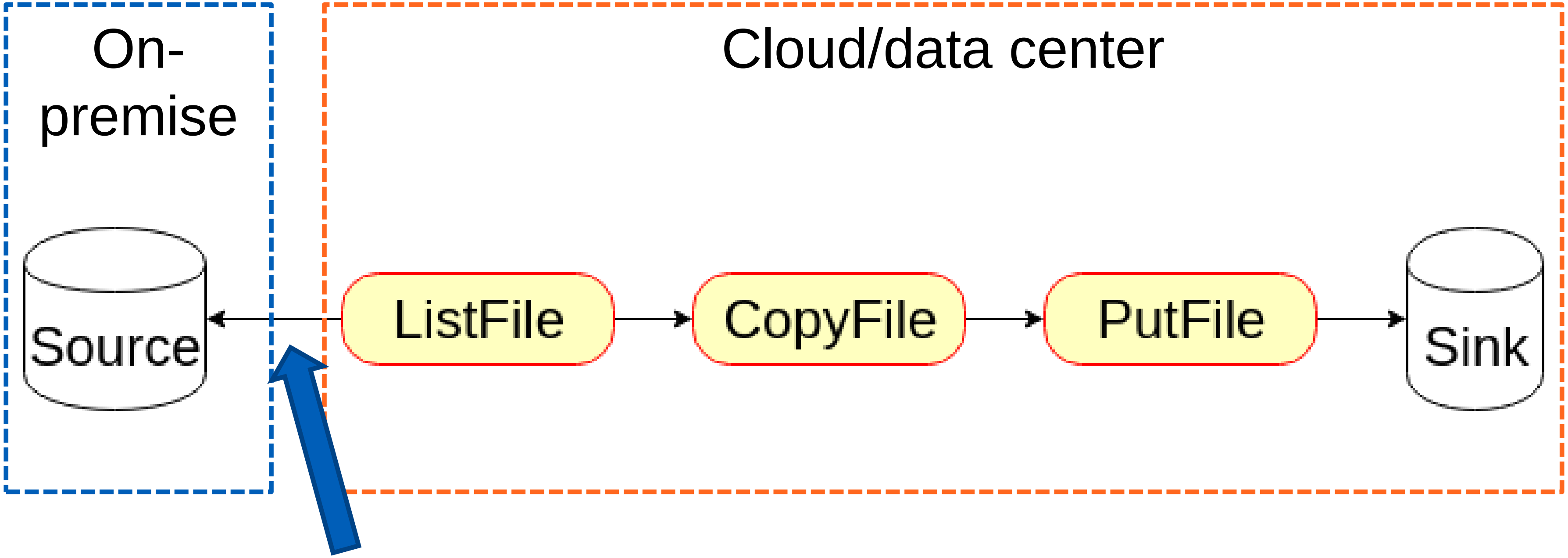
**Data store/platform provider**

# Complex deployment and composition models



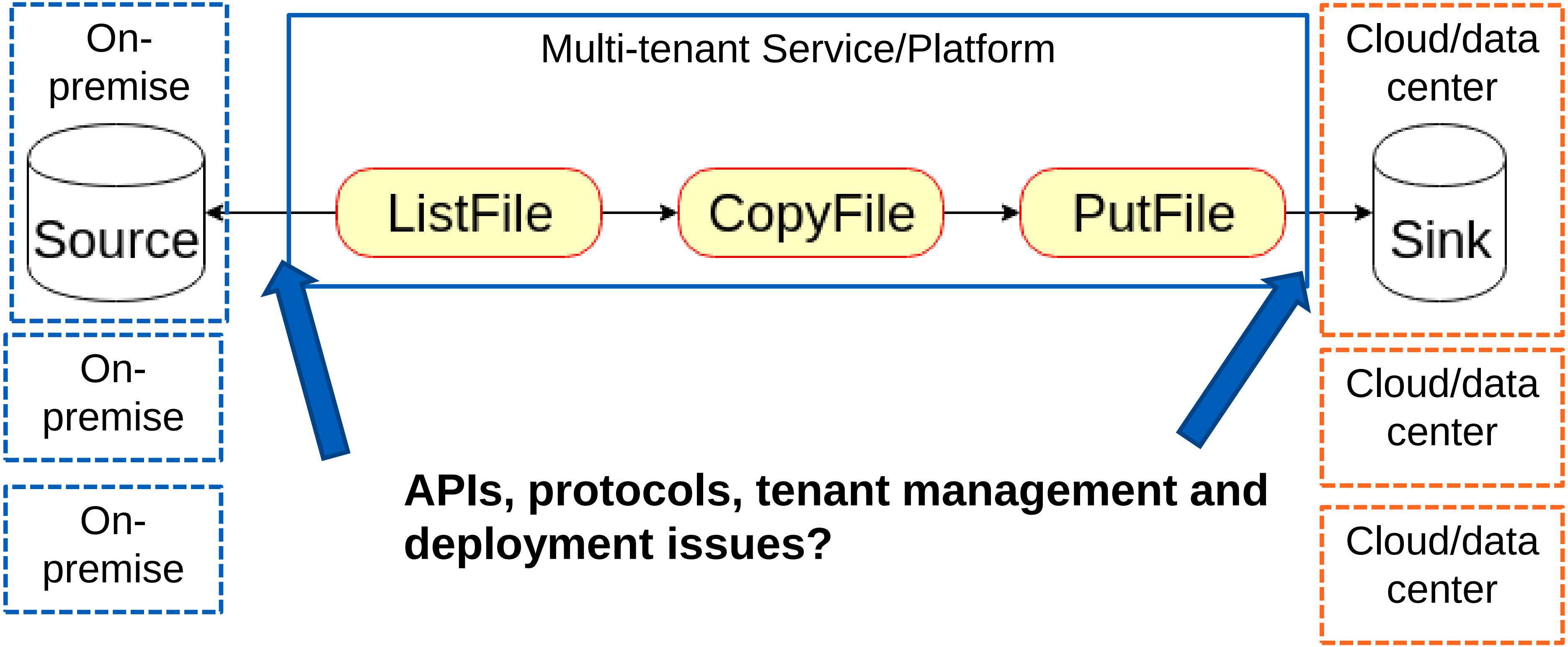
**APIs, protocols and deployment issues?**

# Complex deployment and composition models



**APIs, protocols and deployment issues?**

# Complex deployment and composition models



**APIs, protocols, tenant management and deployment issues?**

Aalto University
School of Science

# Pipeline designs and execution models

# Architecture requirements

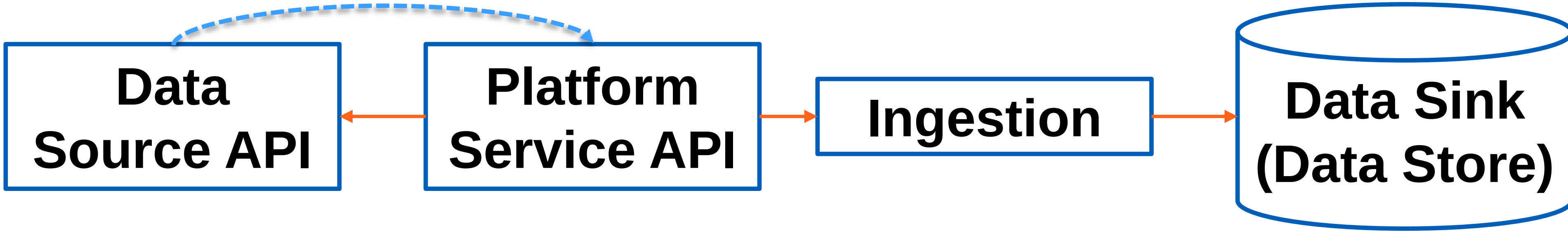
- **Data source integration**
  - The richness and extensibility of data sources and data sinks
- **Batch ingestion and near real-time ingestion requirements**
- **Integration between different ingestion processes across distributed places**
- **The architecture addresses “big data” properties**

# Batch ingestion processes

- **Data to be ingested is bounded**
  - files or messages are finite
- **Ingestion architectural styles**
  - (1) Direct APIs, (2) reactive pipelines, (3) workflows
- **Incremental ingestion**
  - Dealing with the same data source but the data in the source has been changed over the time (related to change data capture)
- **Parallel and distributed execution**
  - Use workflows and distributed processing

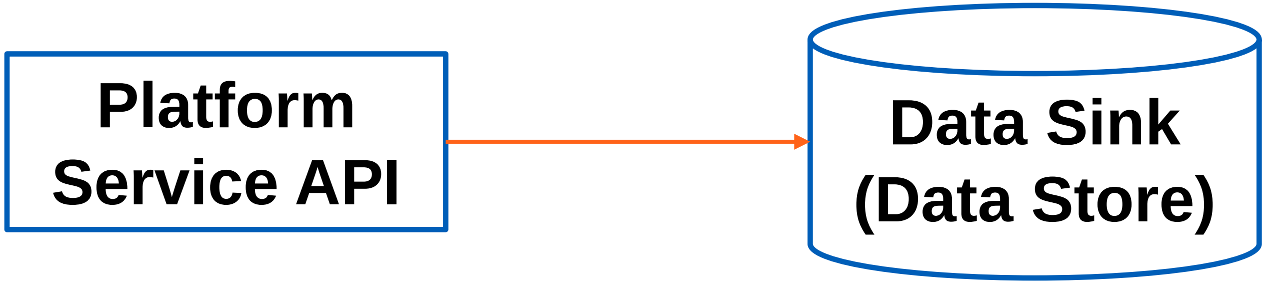
# Simple, direct APIs for ingestion

**Pull model:** register webhook/API



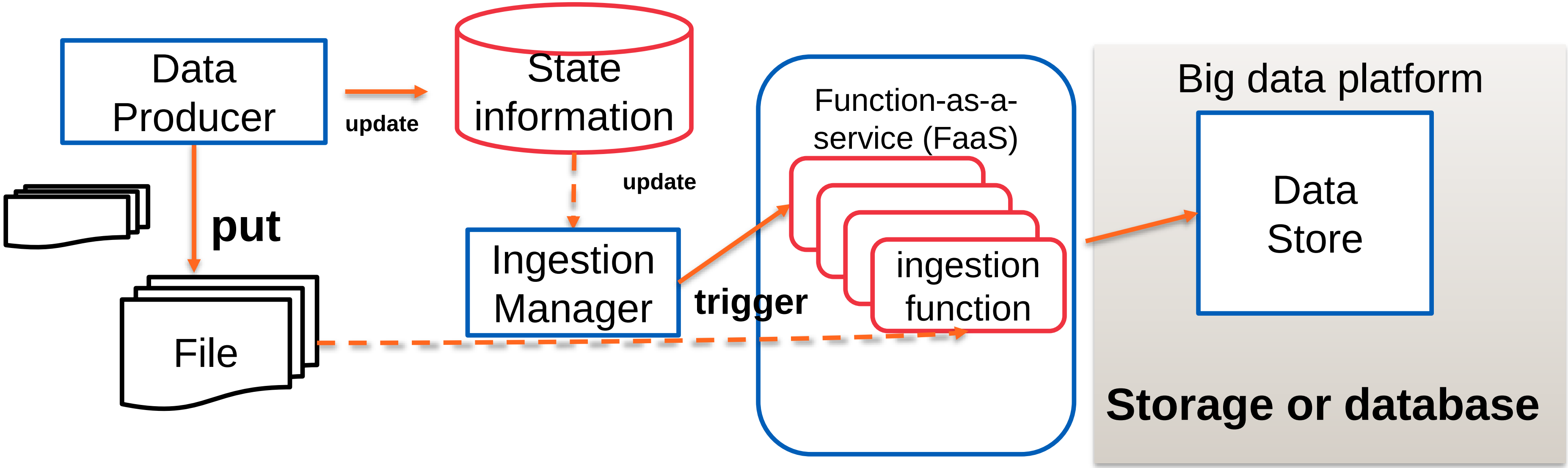
**Push model**

**When?**



**Try to analyze pros and cons for your platform?**

# Reactive with function-as-a-service



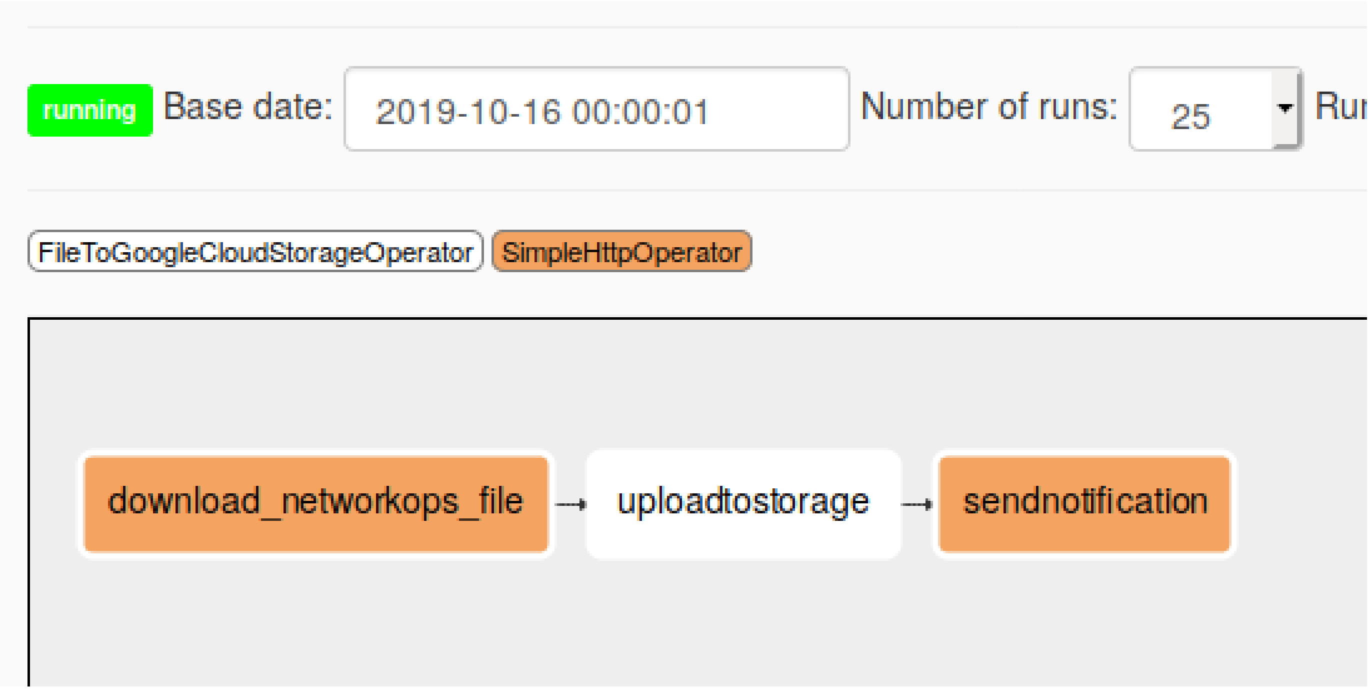
**Who develops which components?**

**Remember?: https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/queuebaseddataingestion**

# Orchestrating ingestion workflow

- **Different tasks for**
  - Access and copy, extract, covert, quality check, and write data
  - Tasks can be connected based on data or control flows
- **Workflows**
  - A set of connected tasks is executed by an engine
  - Tasks can be scheduled and executed in different places
- **Bulk ingestion can be done using workflows**

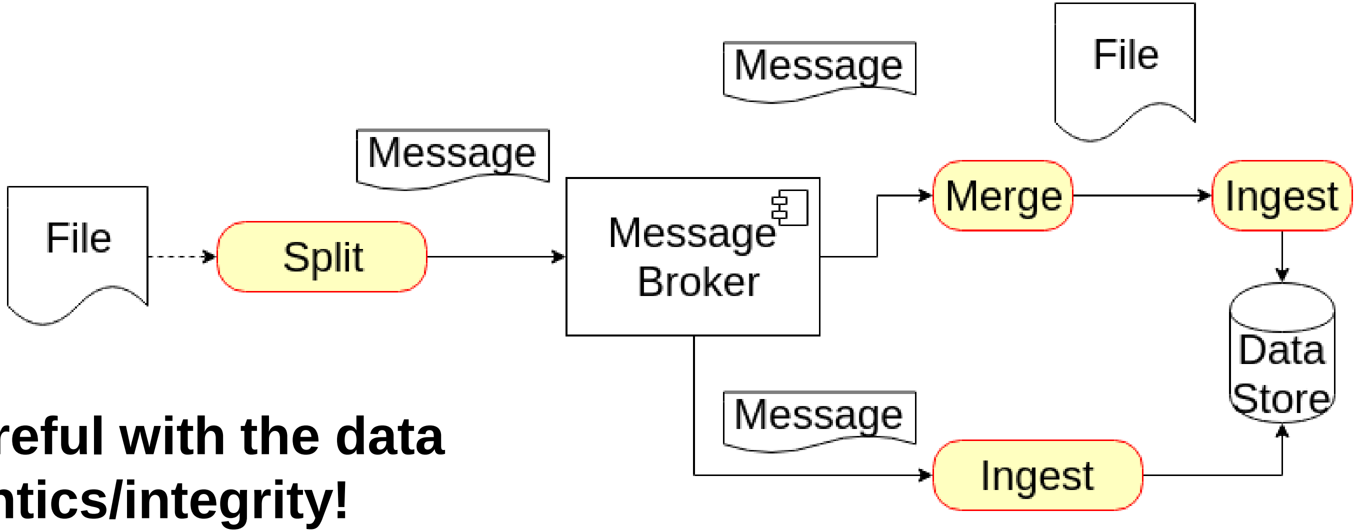
# E.g., workflow based on scheduled time, with Apache Airflow



running Base date: 2019-10-16 00:00:01 Number of runs: 25 Ru

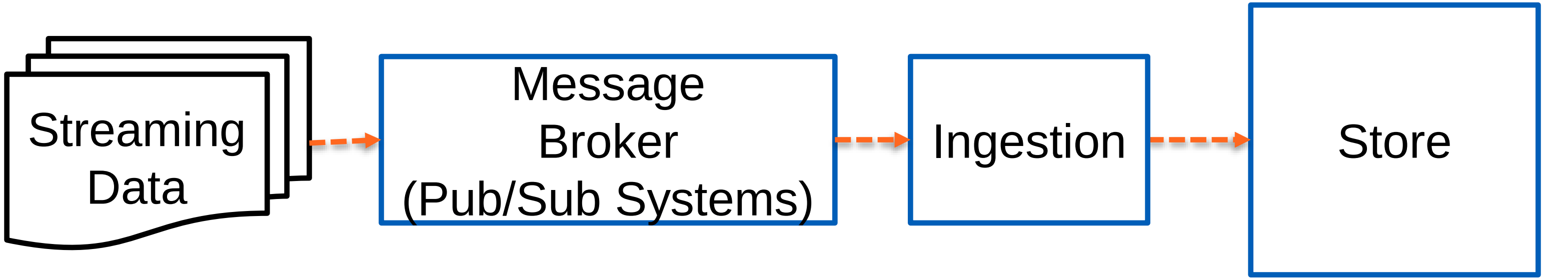
# Microbatching for ingestion

- **Data is split into different chunks ingested using  a batch**
  - Using “streaming” to send chunks
  - Chunks are ingested into the system, or merged and then ingested



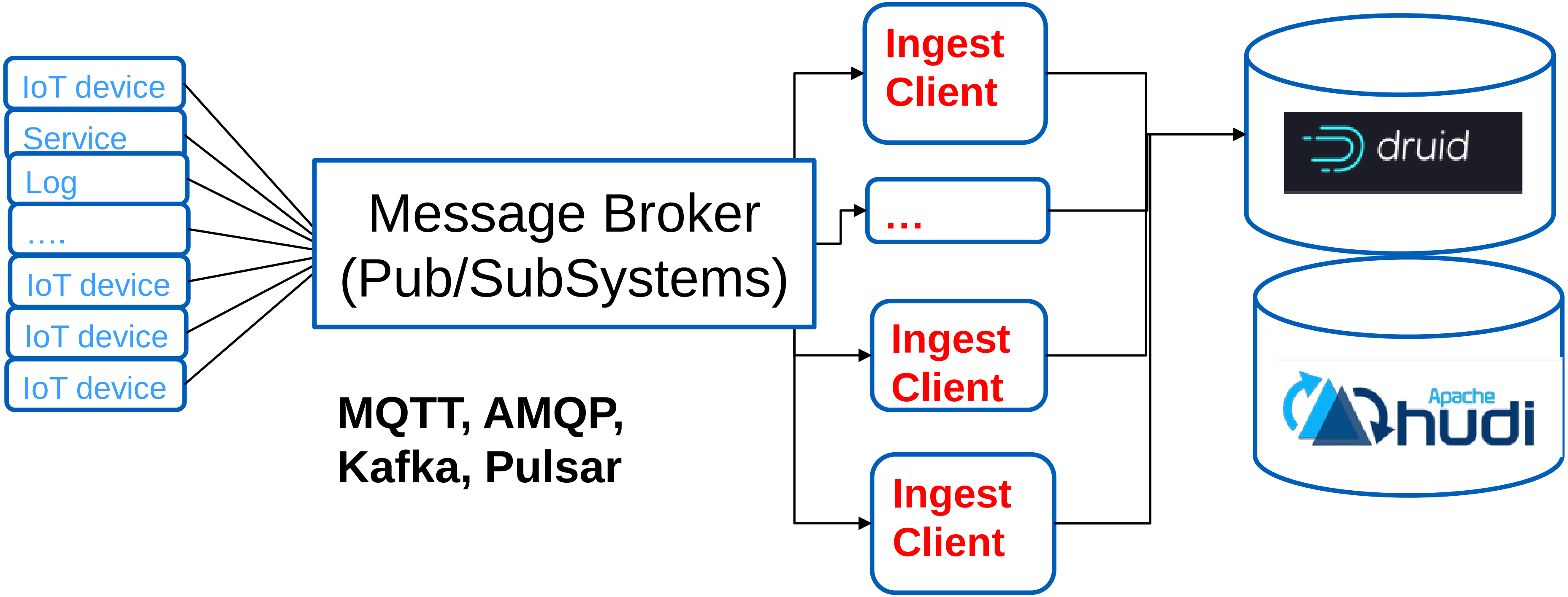
**Be careful with the data semantics/integrity!**

# Near-real time ingestion processes



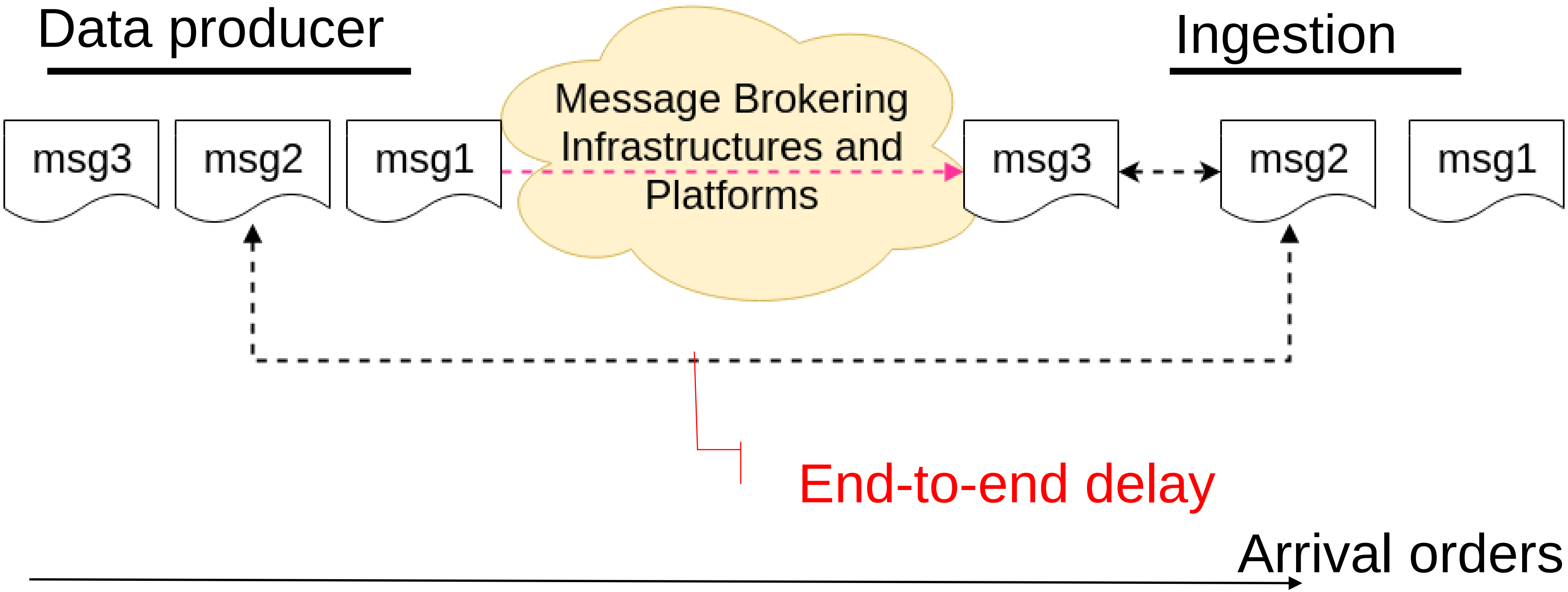
- **Moving streaming data**
- **Unbounded data, amount of data  varies, fast ingestion**

# Example



**MQTT, AMQP, Kafka, Pulsar**

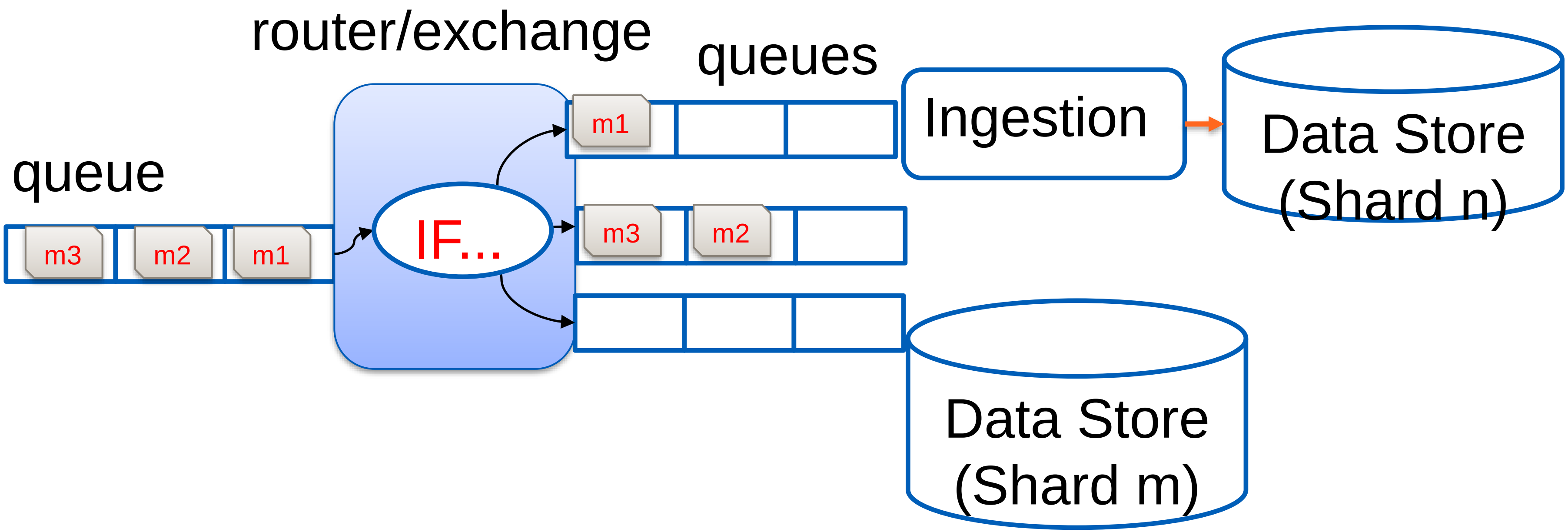
# Key issues in streaming data ingestion



# Some key issues

- **Late data, data out of order?**
- **Exactly once?**
- **Back pressure and retention**
  - for individual components or the whole pipelines
- **Scalability and elasticity**
  - changes in data streams can be unpredictable

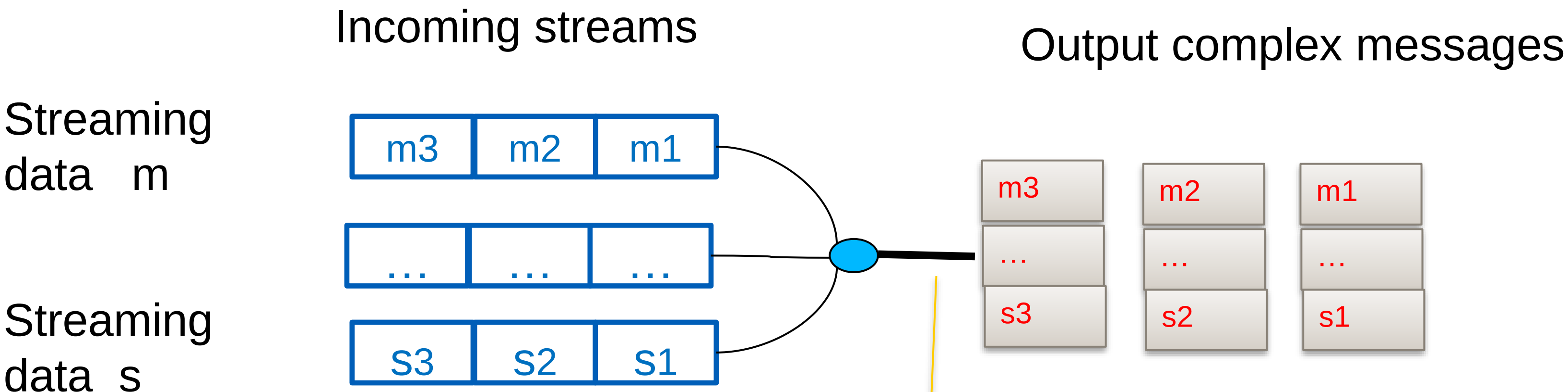
# Split (pub/sub) and partition with ingestion



# Some key issues

- **Multiple topics/streams of data**
  - amount of data per topic varies
  - should not have duplicate data in data store
- **How to distribute topic/data  to ingestion clients?**
- **Where should we run the message broker?**
- **Where should the elasticity be applied?**

# Do we have to merge data before ingestion



Why? e.g., for data rollup/summarization

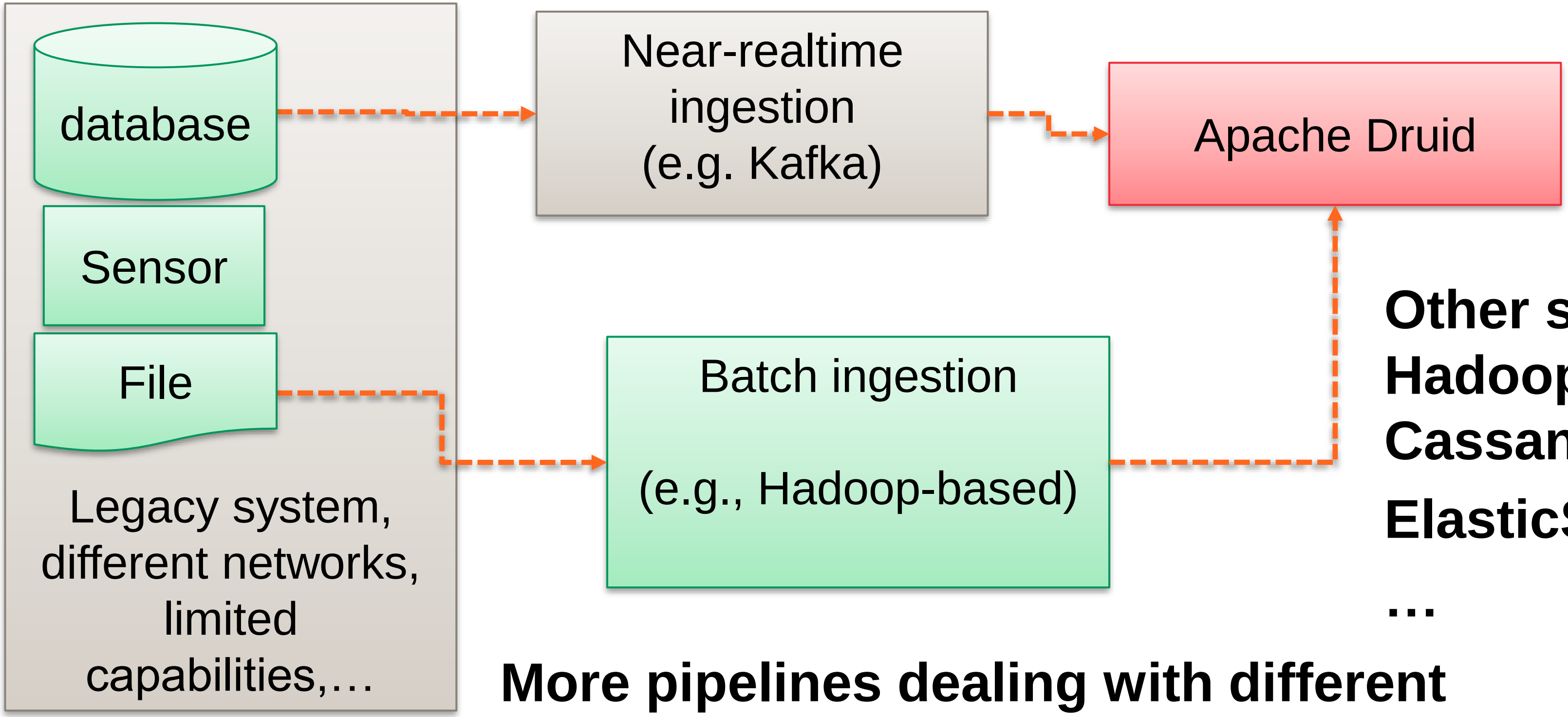
# Complex ingestion pipelines in big data platforms

- **Multiple types of pipelines for multiple types of customers**
- **A customer might need different integrated pipelines**

**→ Both batch and near-realtime ingestion are supported**

- **Complex architectural designs**
  - Ingestion pipeline-to-pipeline needs “bridges”

# Multiple types of pipelines for the same destination/store



database
Sensor
File
Legacy system, different networks, limited capabilities,…
Near-realtime ingestion (e.g. Kafka)
Apache Druid
Batch ingestion
(e.g., Hadoop-based)

**Other systems: Hadoop, Cassandra**

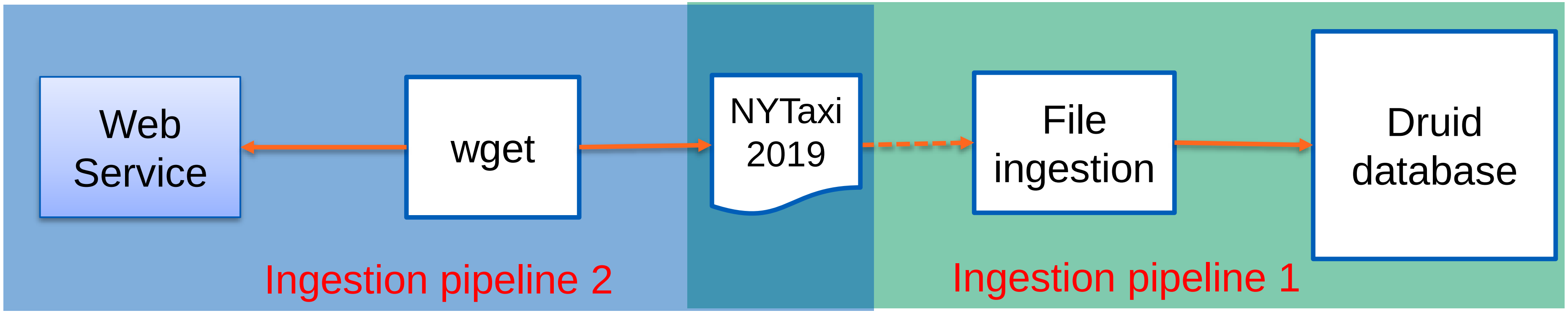
**ElasticSearch**

**…**

**More pipelines dealing with different sources**

# Connecting different ingestion pipelines

**A single tool might not be enough**



**Real-world:**

**both pipelines and their connection are complex**

# Data ingestion with (emerging) Data Lake

**Data Lake provides single store for multiple types of data → reduce effort in building ingestion pipelines**

**Example with Delta Lake** (https://delta.io/)

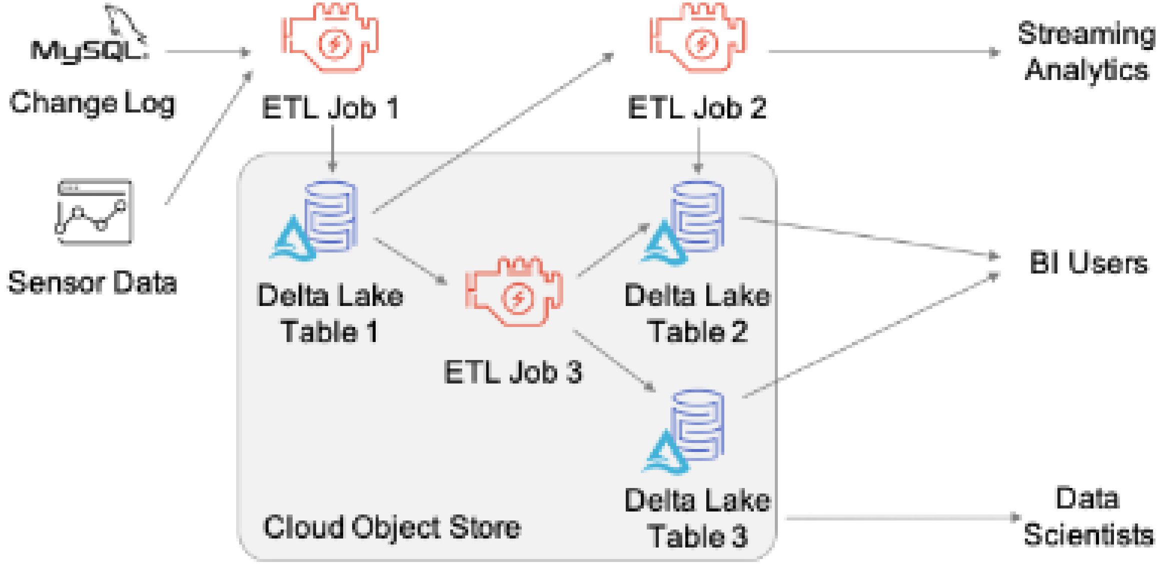


Figure source: “Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores”, https://databricks.com/wp-content/uploads/2020/08/p975-armbrust.pdf

Aalto University
School of Science

# Tooling and examples

# Tooling

- **Given different ingestion models, how do you deliver your ingestion tools/services?**
- **(Traditional) ways of REST API/specific client libraries**
  - Upload using put/get operations
- **Workflows**
  - Self-developed workflows vs automatically generated workflows
- **Pipelines are bundled into containers**
  - Self-developed vs generic pipelines based on user configurations

# Design tools for ingestion processes: Apache Kafka + various data sinks

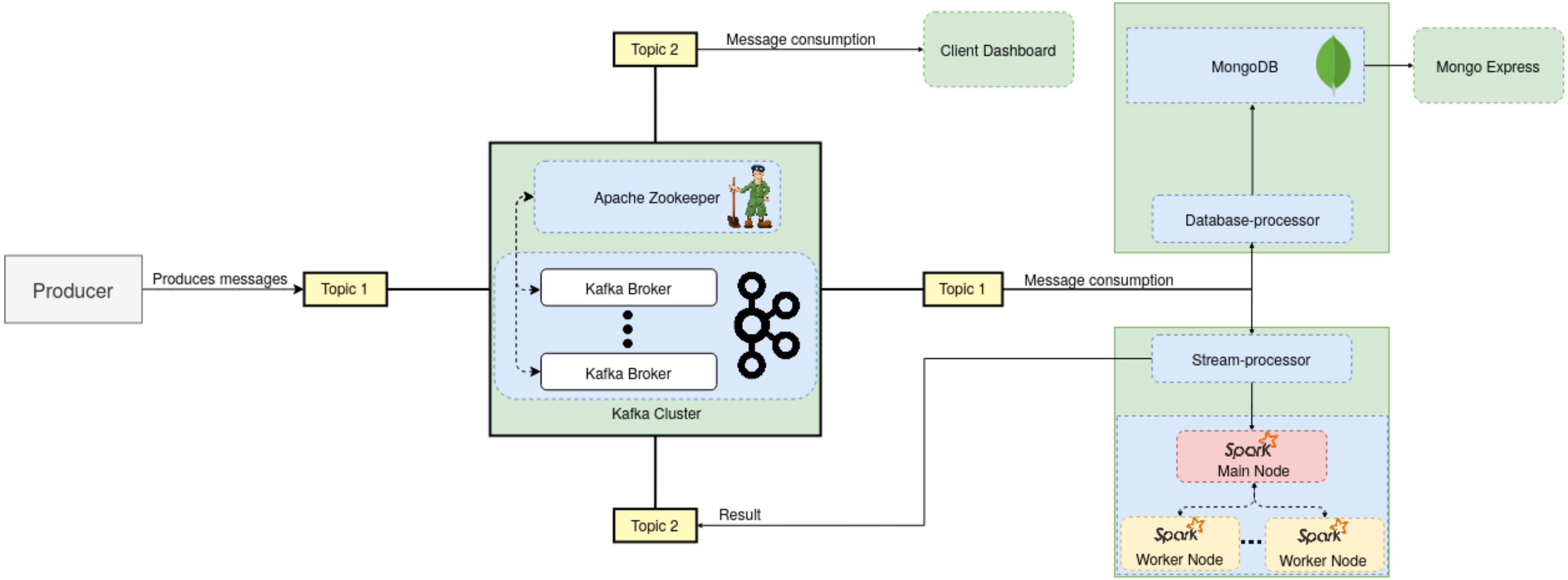
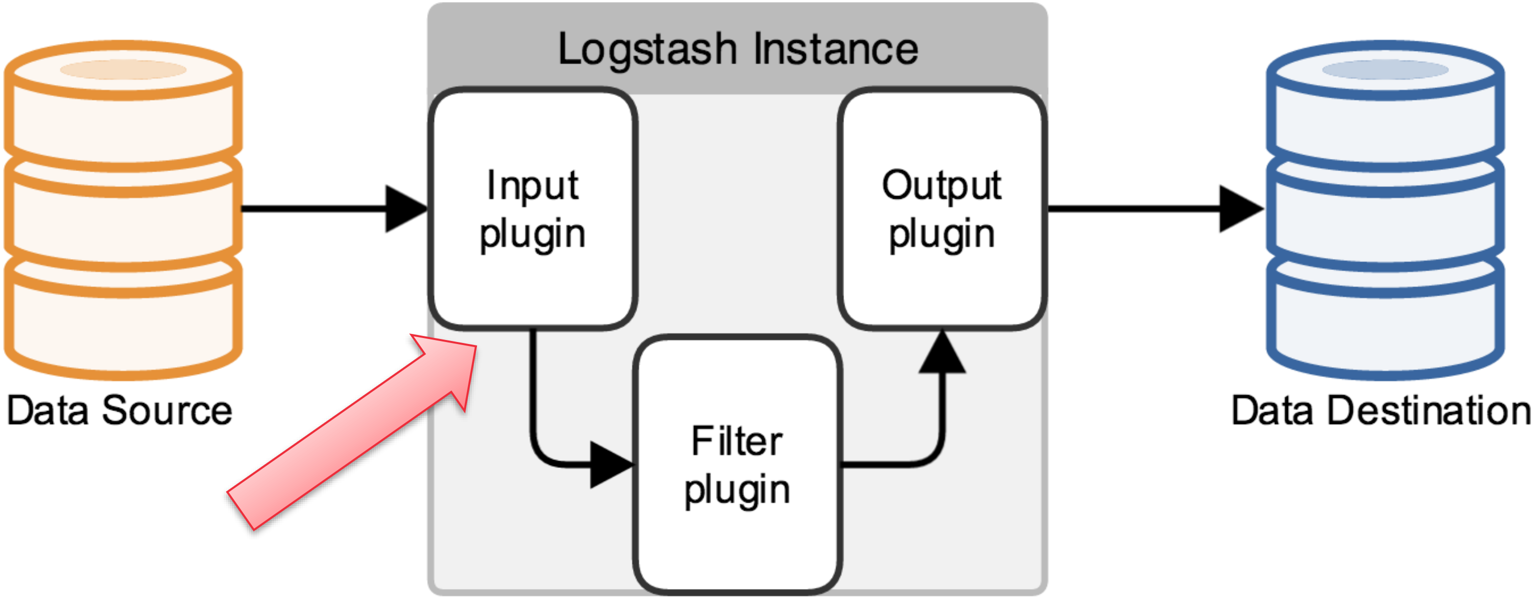


**Figure source: https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/cloud-data-pipeline**

# Design tools for ingestion processes: Logstash

- **For managing logs and events**
  - Collect data from various connectors
  - And parse and store the results through various connectors
- **Programming**
  - Focus on making  pipelines of pluggable components
  - Both programming and configuration deployment needed
- **Deployment**
  - Individual deployment or pipelines
- **Work very well with ElasticSearch**

# Design tools for ingestion processes: Logstash

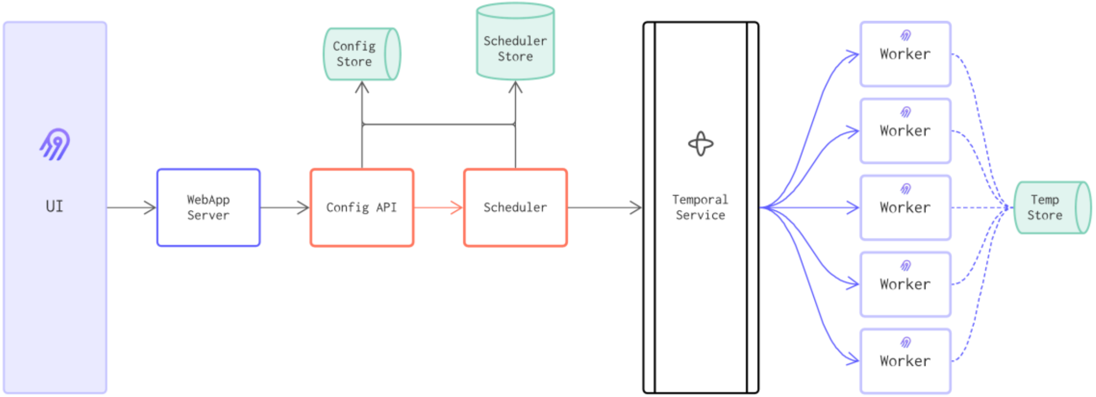


**Pluggable approaches**

Figure source: https://www.elastic.co/guide/en/logstash/current/advanced-pipeline.html

# Design tools for ingestion processes: Airbyte

Allow the user defines input and output configuration then create and deploy containers including ingestion code



**Connectors for multiple data sources**

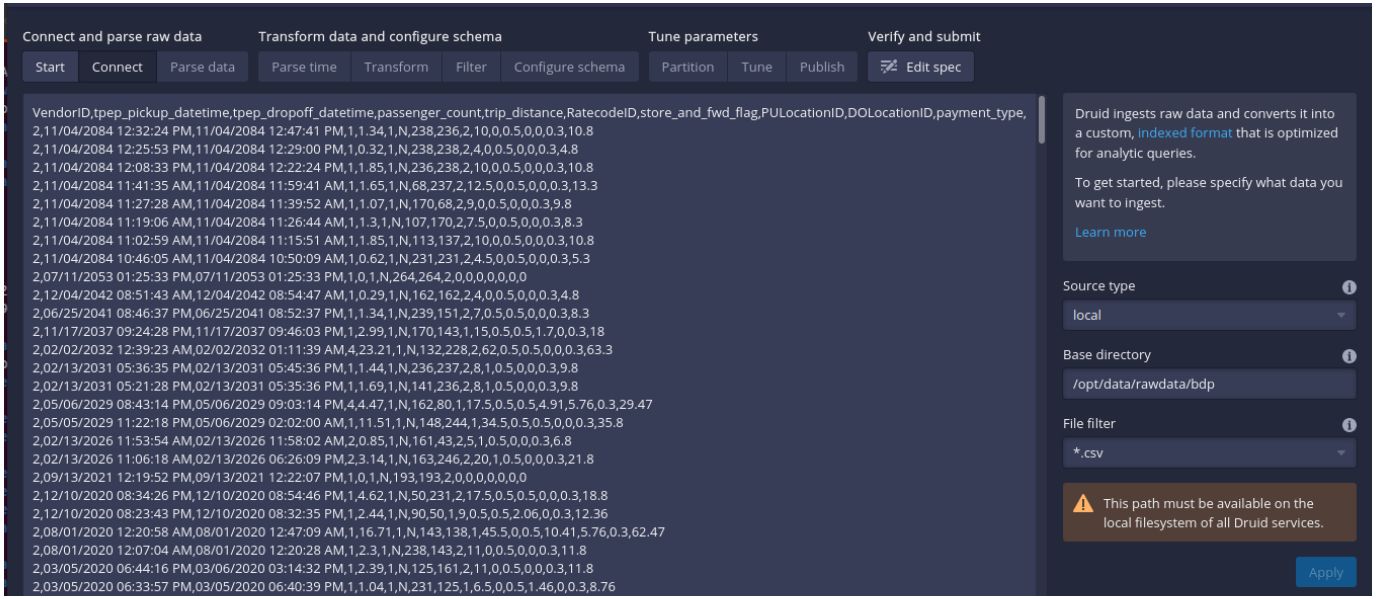
**Platform with scheduler, jobs, workers for data ingestion**

Figure source: https://docs.airbyte.com/understanding-airbyte/high-level-view

Aalto University
School of Science

# Design tools for ingestion processes: Apache Druid

## Allow the user to build the plan: select tasks, configuration, etc. and then generate ingestion pipelines



```
VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,payment_type,
2,11/04/2084 12:32:24 PM,11/04/2084 12:47:41 PM,1,1.34,1,N,238,236,2,10,0,0.5,0,0,0.3,10.8
2,11/04/2084 12:25:53 PM,11/04/2084 12:29:00 PM,1,0.32,1,N,238,238,2,4,0,0.5,0,0,0.3,4.8
2,11/04/2084 12:08:33 PM,11/04/2084 12:22:24 PM,1,1.85,1,N,236,238,2,10,0,0.5,0,0,0.3,10.8
2,11/04/2084 11:41:35 AM,11/04/2084 11:59:41 AM,1,1.65,1,N,68,237,2,12.5,0,0.5,0,0,0.3,13.3
2,11/04/2084 11:27:28 AM,11/04/2084 11:39:52 AM,1,1.07,1,N,170,68,2,9,0,0.5,0,0,0.3,9.8
2,11/04/2084 11:19:06 AM,11/04/2084 11:26:44 AM,1,1.3,1,N,107,170,2,7.5,0,0.5,0,0,0.3,8.3
2,11/04/2084 11:02:59 AM,11/04/2084 11:15:51 AM,1,1.85,1,N,113,137,2,10,0,0.5,0,0,0.3,10.8
2,11/04/2084 10:46:05 AM,11/04/2084 10:50:09 AM,1,0.62,1,N,231,231,2,4.5,0,0.5,0,0,0.3,5.3
2,07/11/2053 01:25:33 PM,07/11/2053 01:25:33 PM,1,0,1,N,264,264,2,0,0,0,0,0,0,0
2,12/04/2042 08:51:43 AM,12/04/2042 08:54:47 AM,1,0.29,1,N,162,162,2,4,0,0.5,0,0,0.3,4.8
2,06/25/2041 08:46:37 PM,06/25/2041 08:52:37 PM,1,1.34,1,N,239,151,2,7,0.5,0.5,0,0,0.3,8.3
2,11/17/2037 09:24:28 PM,11/17/2037 09:46:03 PM,1,2.99,1,N,170,143,1,15,0.5,0.5,1.7,0,0.3,18
2,02/02/2032 12:39:23 AM,02/02/2032 01:11:39 AM,4,23.21,1,N,132,228,2,62,0.5,0.5,0,0,0.3,63.3
2,02/13/2031 05:36:35 PM,02/13/2031 05:45:36 PM,1,1.44,1,N,236,237,2,8,1,0.5,0,0,0.3,9.8
2,02/13/2031 05:21:28 PM,02/13/2031 05:35:36 PM,1,1.69,1,N,141,236,2,8,1,0.5,0,0,0.3,9.8
2,05/06/2029 08:43:14 PM,05/06/2029 09:03:14 PM,4,4.47,1,N,162,80,1,17.5,0.5,0.5,4.91,5.76,0.3,29.47
2,05/05/2029 11:22:18 PM,05/06/2029 02:02:00 AM,1,11.51,1,N,148,244,1,34.5,0.5,0.5,0,0,0.3,35.8
2,02/13/2026 11:53:54 AM,02/13/2026 11:58:02 AM,2,0.85,1,N,161,43,2,5,1,0.5,0,0,0.3,6.8
2,02/13/2026 11:06:18 AM,02/13/2026 06:26:09 PM,2,3.14,1,N,163,246,2,20,1,0.5,0,0,0.3,21.8
2,09/13/2021 12:19:52 PM,09/13/2021 12:22:07 PM,1,0,1,N,193,193,2,0,0,0,0,0,0,0
2,12/10/2020 08:34:26 PM,12/10/2020 08:54:46 PM,1,4.62,1,N,50,231,2,17.5,0.5,0.5,0,0,0.3,18.8
2,12/10/2020 08:23:43 PM,12/10/2020 08:32:35 PM,1,2.44,1,N,90,50,1,9,0.5,0.5,2.06,0,0.3,12.36
2,08/01/2020 12:20:58 AM,08/01/2020 12:47:09 AM,1,16.71,1,N,143,138,1,45.5,0,0.5,10.41,5.76,0.3,62.47
2,08/01/2020 12:07:04 AM,08/01/2020 12:20:28 AM,1,2.3,1,N,238,143,2,11,0,0.5,0,0,0.3,11.8
2,03/05/2020 06:44:16 PM,03/06/2020 03:14:32 PM,1,2.39,1,N,125,161,2,11,0,0.5,0,0,0.3,11.8
2,03/05/2020 06:33:57 PM,03/05/2020 06:40:39 PM,1,1.04,1,N,231,125,1,6.5,0,0.5,1.46,0,0.3,8.76
```

Druid ingests raw data and converts it into a custom, indexed format that is optimized for analytic queries.

To get started, please specify what data you want to ingest.

Learn more

Source type: local

Base directory: /opt/data/rawdata/bdp

File filter: *.csv

⚠ This path must be available on the local filesystem of all Druid services.

# Design tools for ingestion processes: Apache Nifi

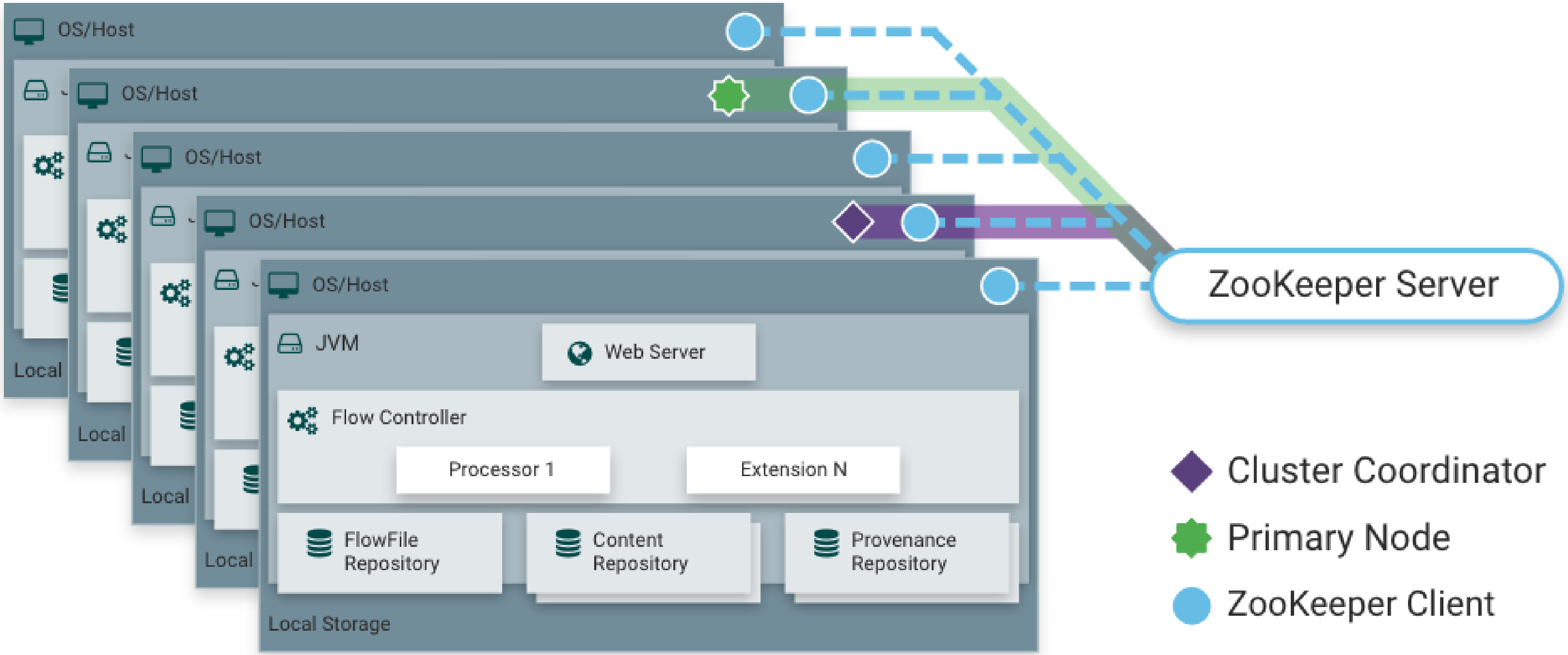
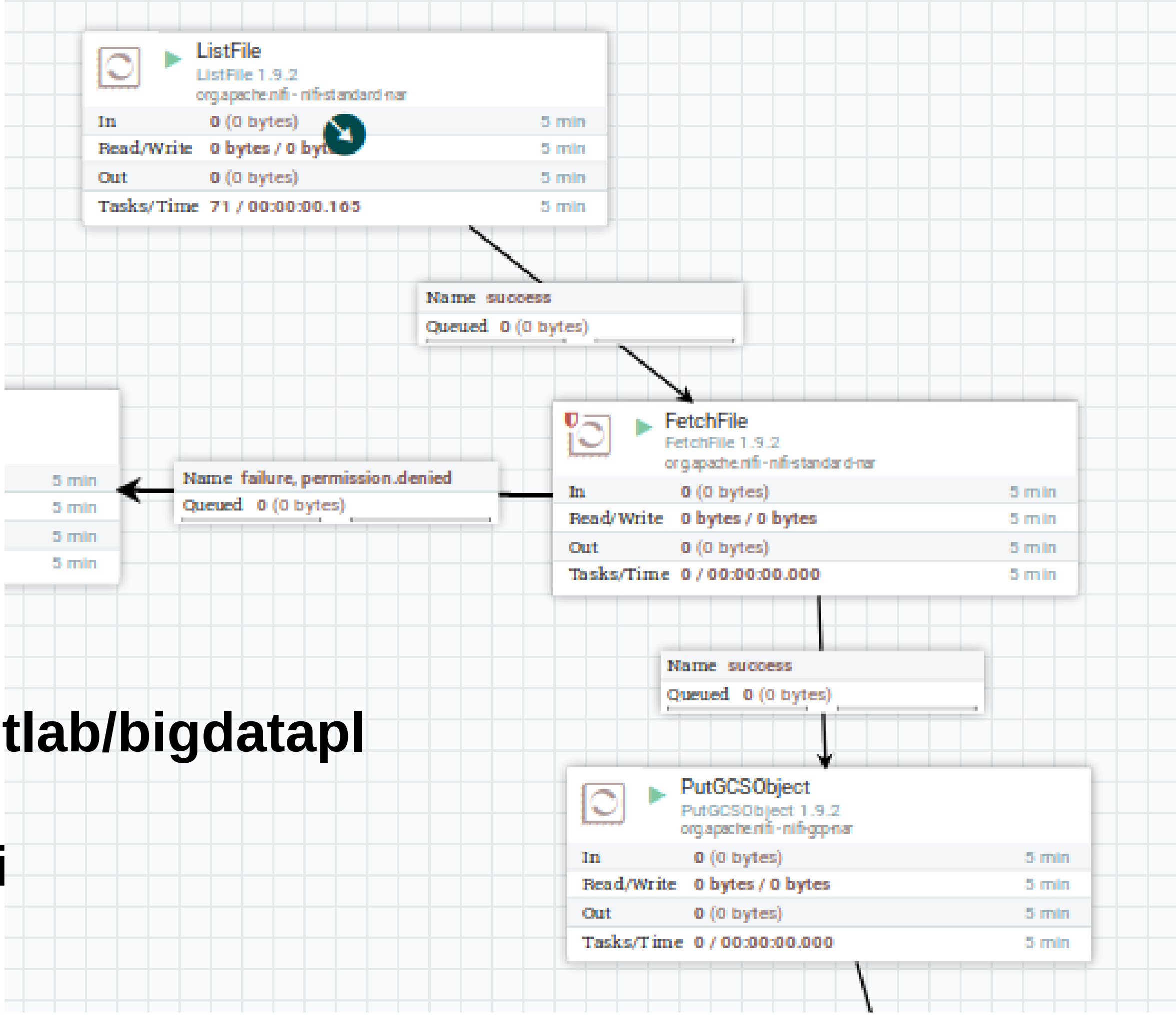


Figure source: https://nifi.apache.org/docs.html

# Design tools for ingestion processes: Apache Nifi - key concept

- Data is encapsulated into “FlowFile”
- Processor (Component) performs tasks
- Processor handle FlowFile and has different states
  - Each state indicates the results of processing that can be used for establishing relationships to other components
- Processors are connected by Connection
- Connection can have many relationships based on states of upstream Processors

# Design tools for ingestion processes: Apache Nifi



ListFile
ListFile 1.9.2
org.apache.nifi - nifi-standard-nar

| | | |
|---|---|---|
| In | 0 (0 bytes) | 5 min |
| Read/Write | 0 bytes / 0 bytes | 5 min |
| Out | 0 (0 bytes) | 5 min |
| Tasks/Time | 71 / 00:00:00.165 | 5 min |

Name success
Queued 0 (0 bytes)

FetchFile
FetchFile 1.9.2
org.apache.nifi - nifi-standard-nar

| | | |
|---|---|---|
| In | 0 (0 bytes) | 5 min |
| Read/Write | 0 bytes / 0 bytes | 5 min |
| Out | 0 (0 bytes) | 5 min |
| Tasks/Time | 0 / 00:00:00.000 | 5 min |

Name failure, permission.denied
Queued 0 (0 bytes)

5 min
5 min
5 min
5 min

Name success
Queued 0 (0 bytes)

PutGCSObject
PutGCSObject 1.9.2
org.apache.nifi - nifi-gcp-nar

| | | |
|---|---|---|
| In | 0 (0 bytes) | 5 min |
| Read/Write | 0 bytes / 0 bytes | 5 min |
| Out | 0 (0 bytes) | 5 min |
| Tasks/Time | 0 / 00:00:00.000 | 5 min |

See the tutorial: https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/nifi

# Thanks!

## Hong-Linh Truong
## Department of Computer Science

## rdsea.github.io