Aalto University
School of Science

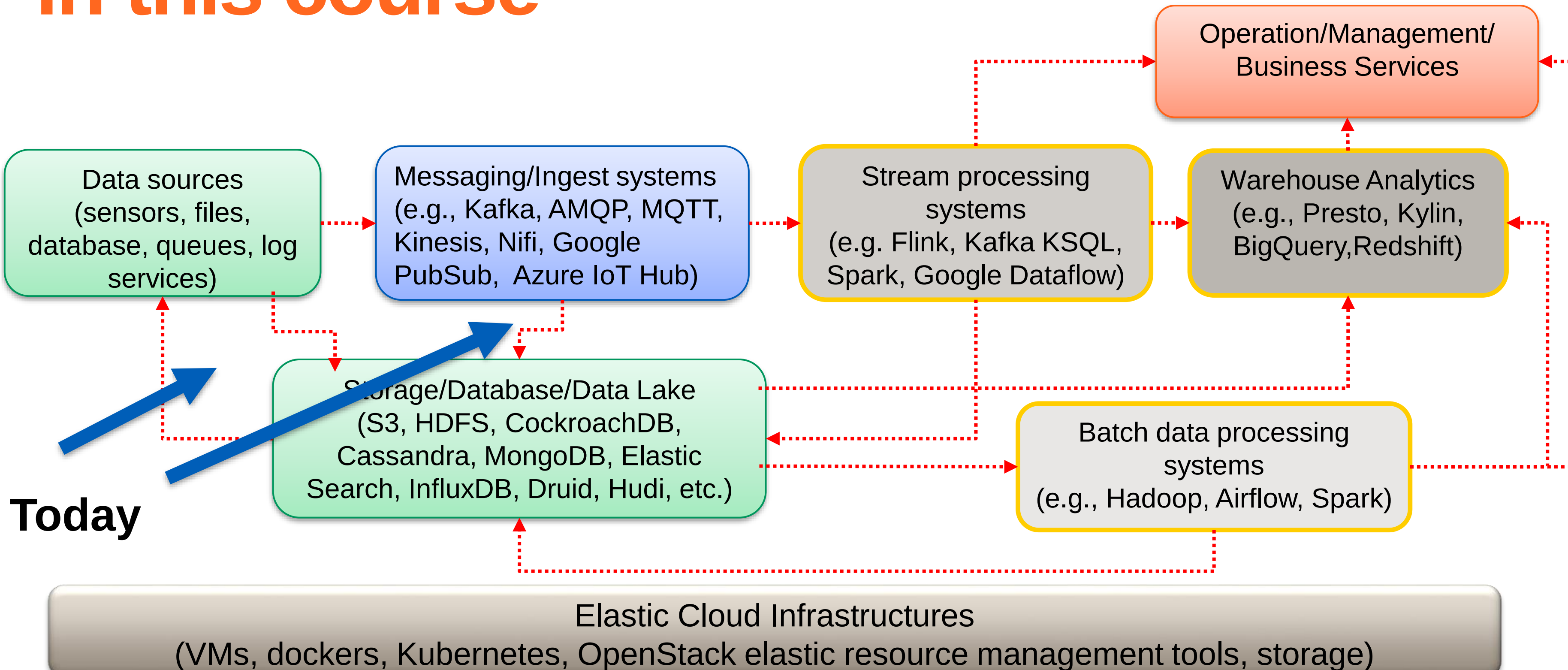
Big Data Ingestion

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



Ingest big data into platforms

Data ingestion: Move data from different sources into the 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, Load, Transform)**
  - 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 (ELT)**

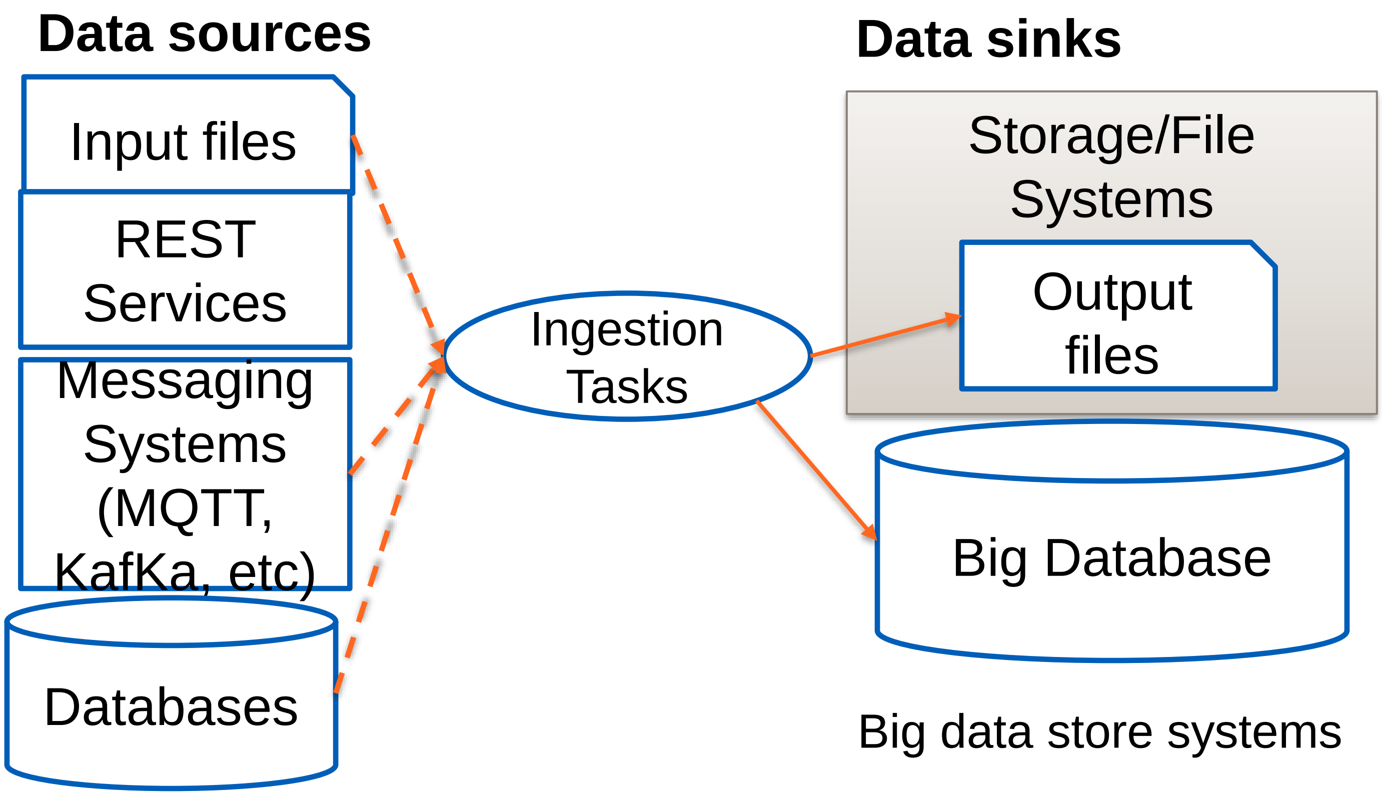
Correctness and quality assurance are hard!

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



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 ingestion 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

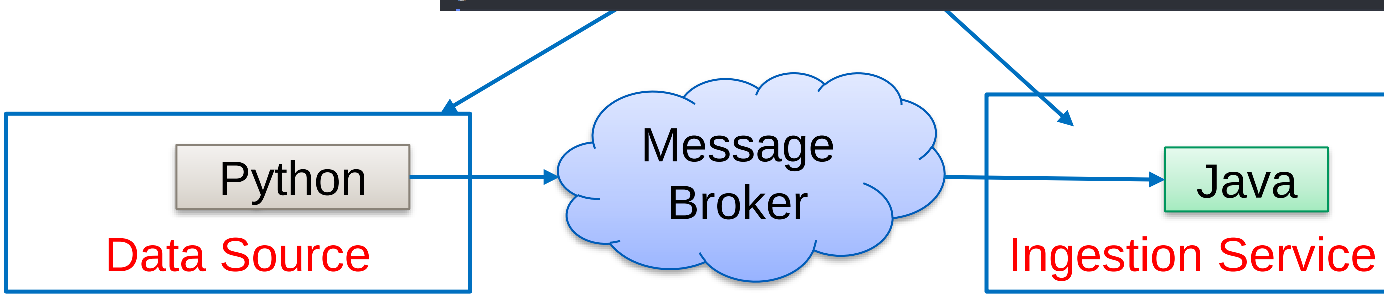
- **Remember that the data sender and the receiver are diverse**
  - In many cases, they are not in the same organization
    - *You need to guarantee the message syntax and semantics*
- **Solutions**
  - Agreed in advance → in the implementation or with a standard
  - Know and use tools to deal with 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"}

        ]
}
```

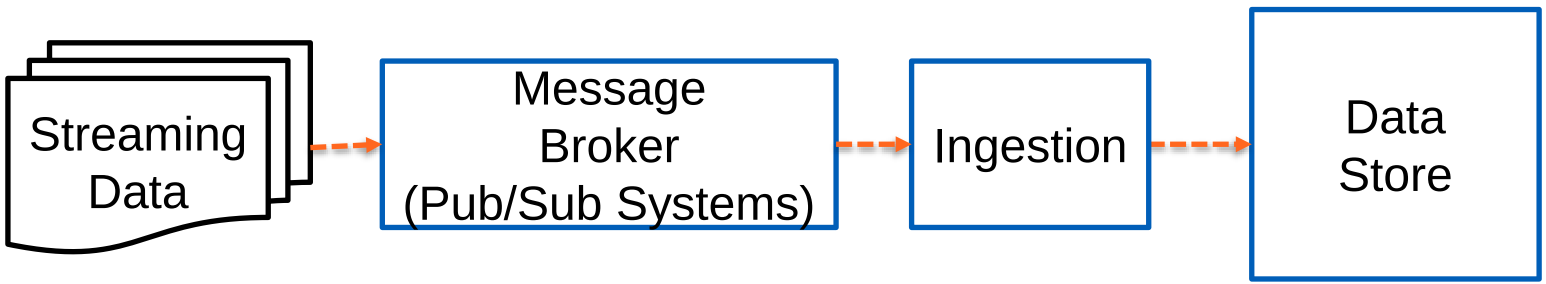


Target file formats in big data storage

- **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

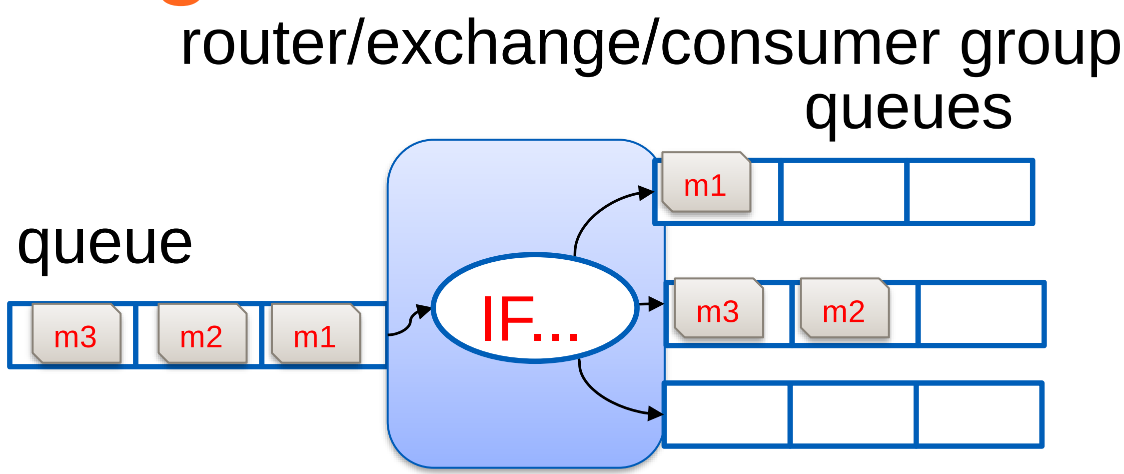
Reading: am Uber blog bout file formats and performance: https://eng.uber.com/cost-efficiency-big-data/

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



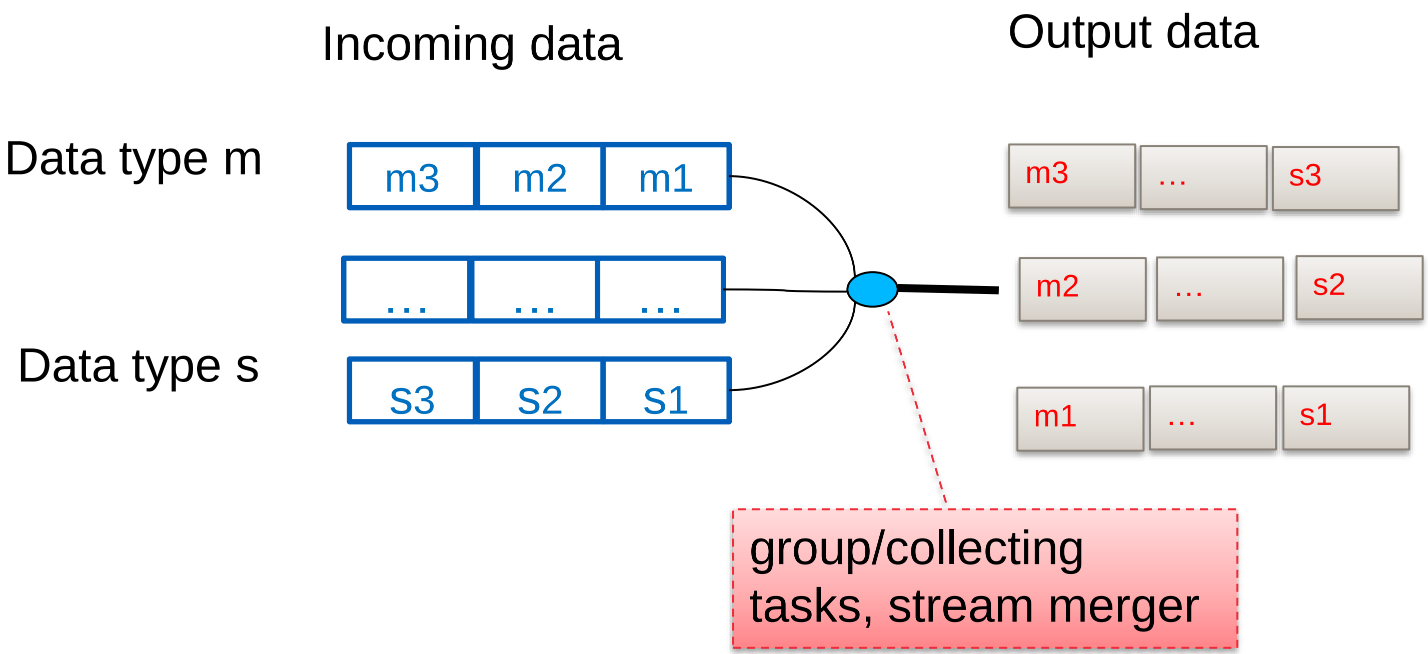
Protocol
Data format
Message structure
Basic streaming data processing techniques

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?

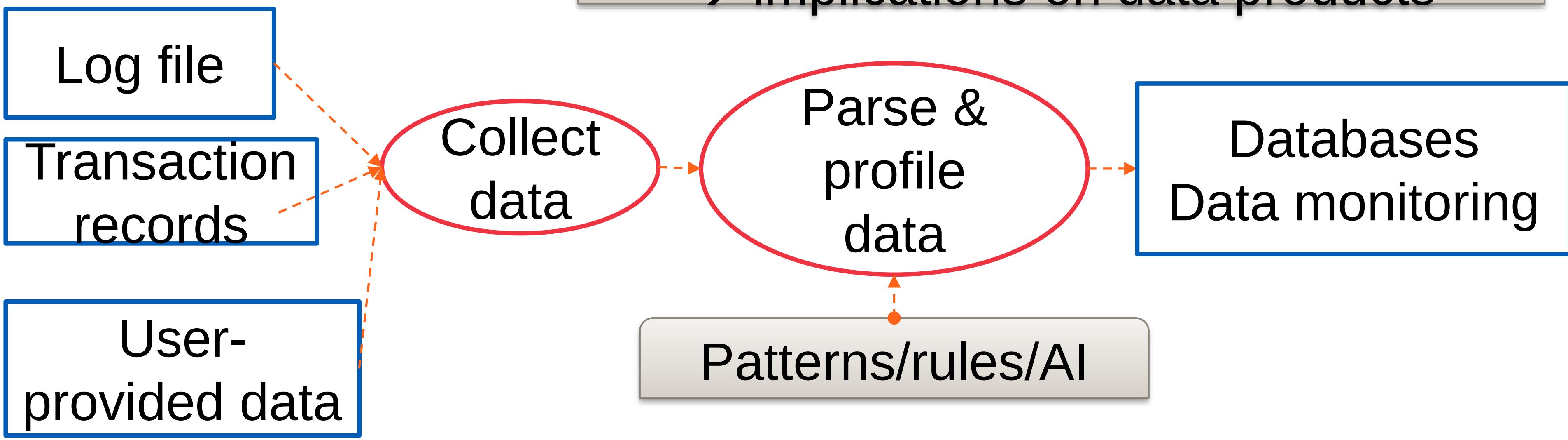
Data wrangling

- **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



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)
```

Ingestion tasks implemented as extensible, composable connectors

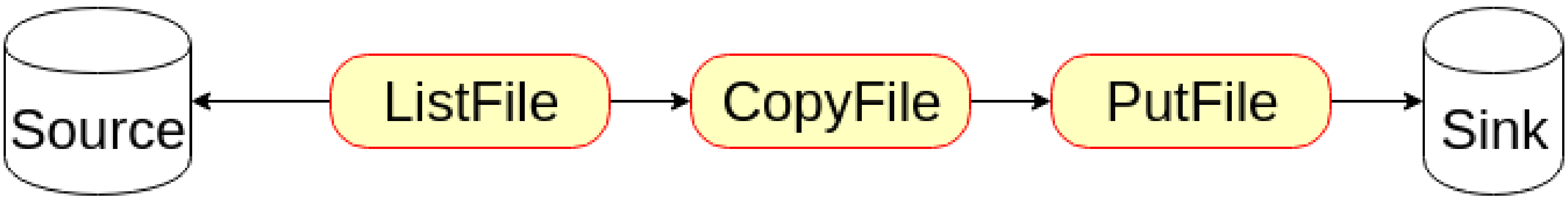
- **Basic tasks for big data ingestion can be used in different cases**
- **Support end-user tasks**
  - Platform enables the user to do many tasks through configurations
- **Enable pluggable approaches is important**
  - Input data plugin/component → filter/extract/convert → output data plugin/component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

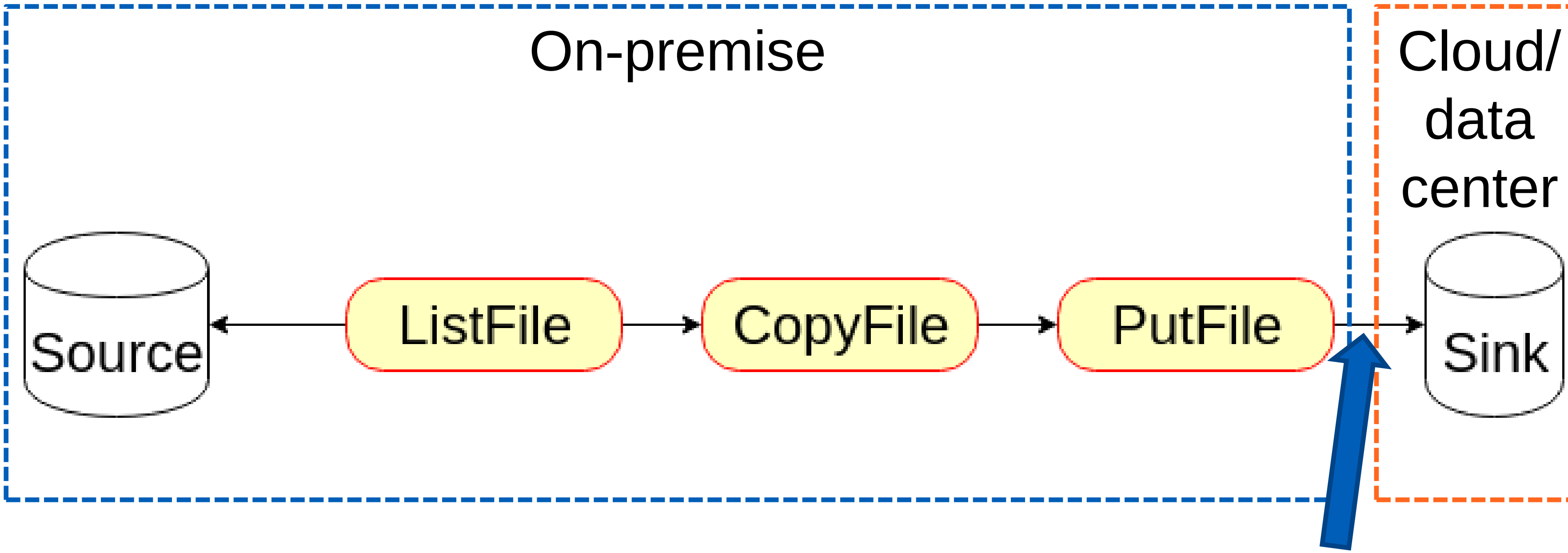


Customer

Ingestion pipeline developer (for whom?)

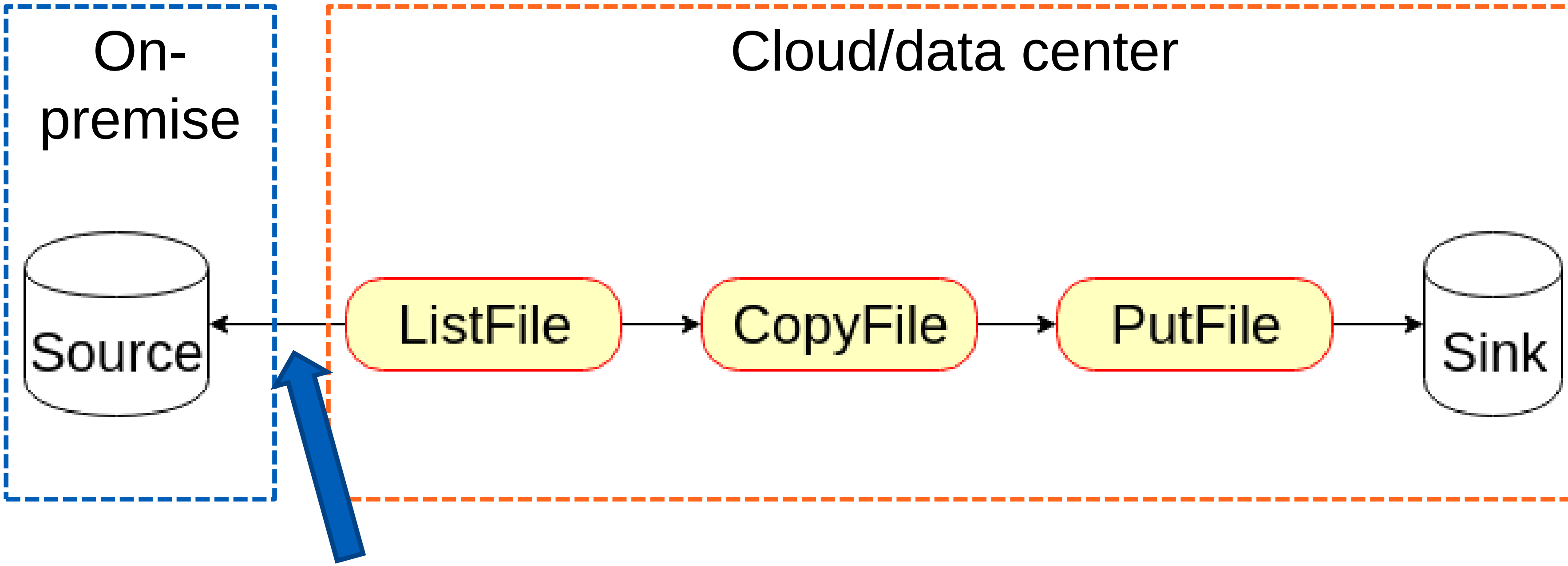
Data store/platform provider

Complex deployment and composition models



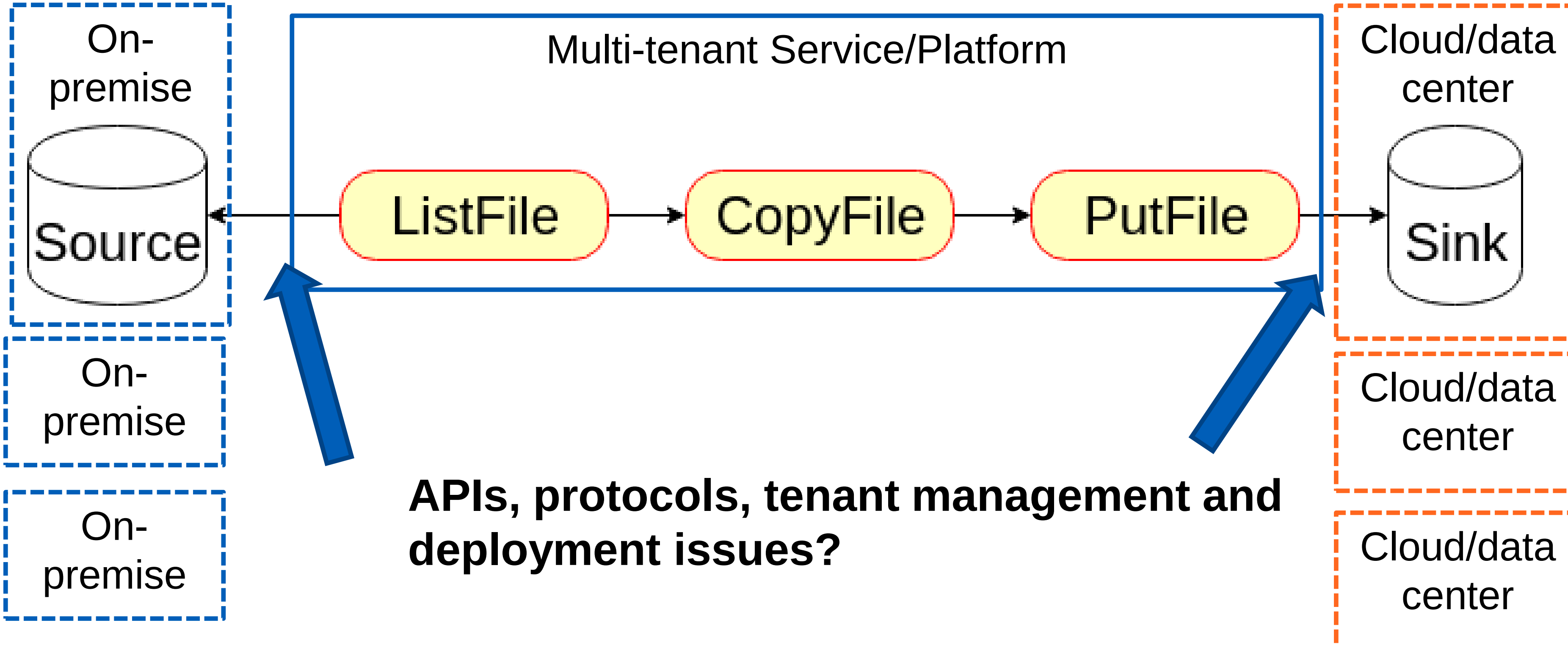
APIs, protocols and deployment issues?

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

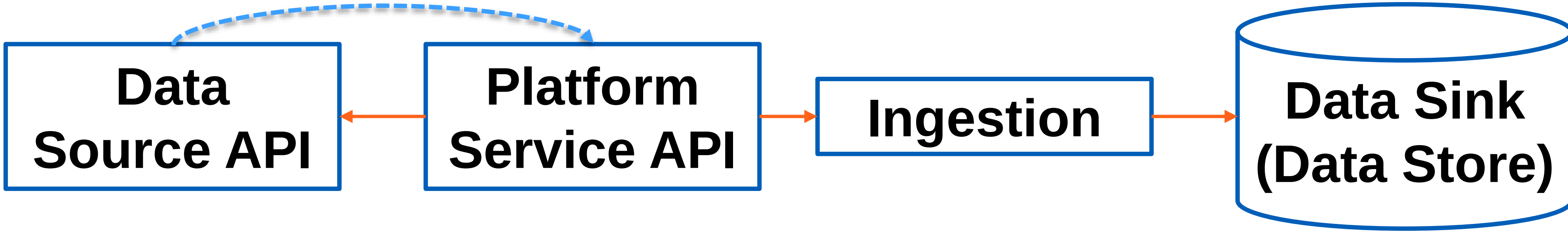
- **Data source integration**
  - The richness and extensibility of data sources and data sinks
- **Batch ingestion and near real-time ingestion requirements**
- **Integration between different ingestion processes across distributed places**
- **The architecture addresses “big data” properties**

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

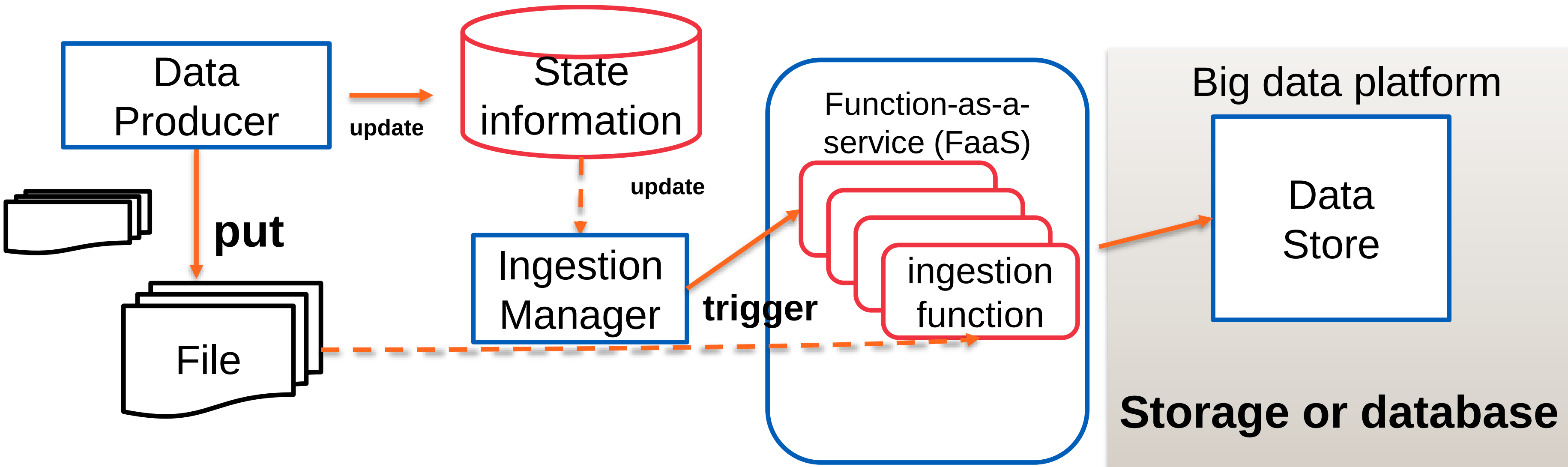
Pull model: register webhook/API



Push model
When?

Try to analyze pros and cons for your platform?

Reactive with function-as-a-service



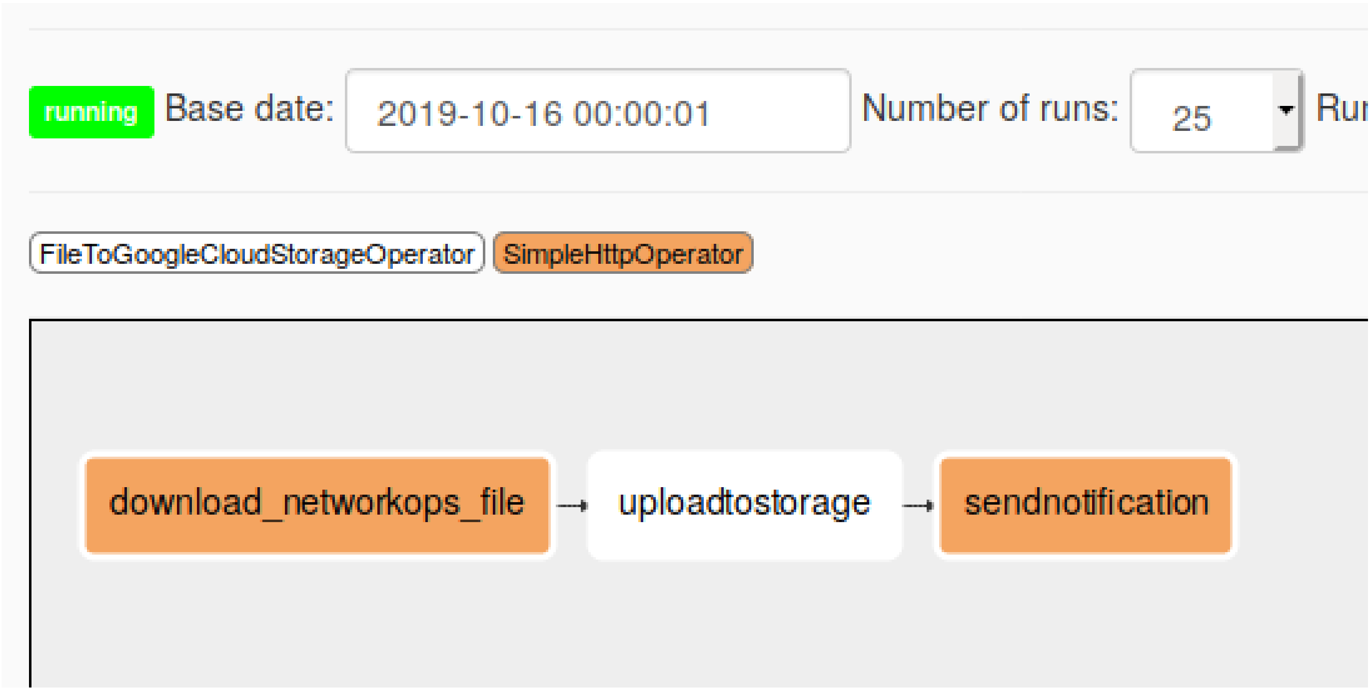
Who develops which components?

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

Orchestrating ingestion workflow

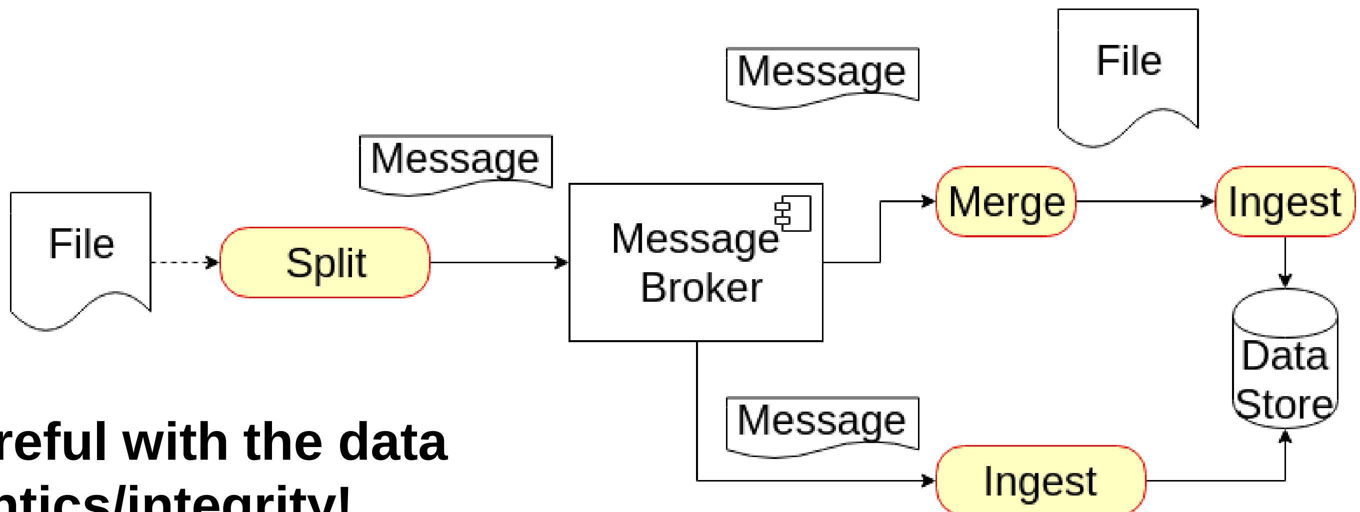
- **Different tasks for**
  - Access and copy, extract, covert, quality check, and write data
  - Tasks can be connected based on data or control flows
- **Workflows**
  - A set of connected tasks is executed by an engine
  - Tasks can be scheduled and executed in different places
- **Bulk ingestion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

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



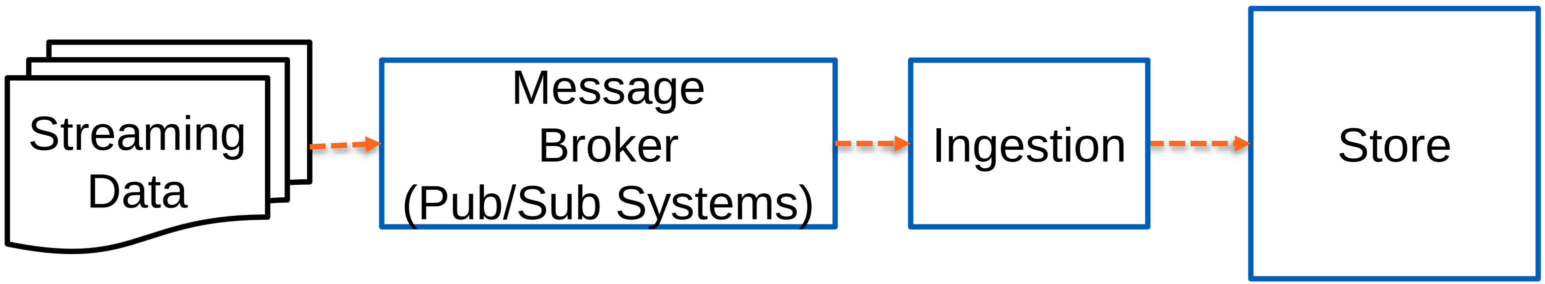
Microbatching for ingestion

- **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



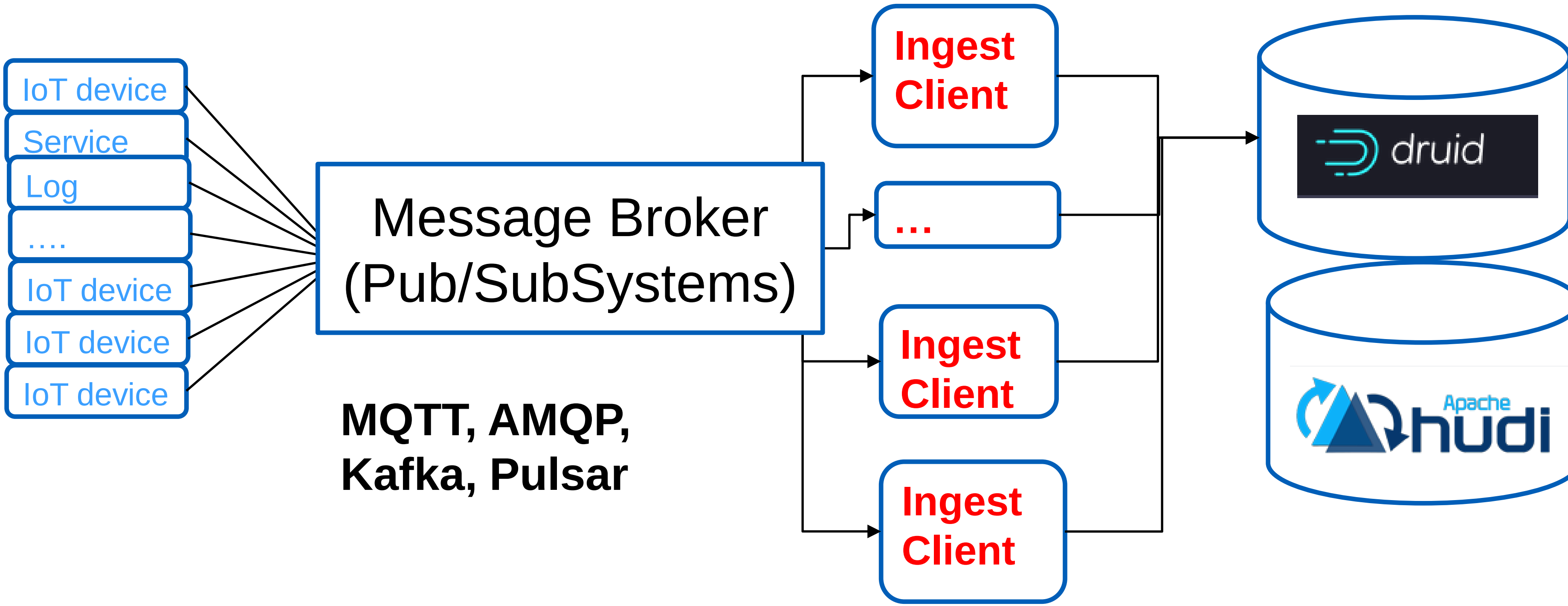
Be careful with the data semantics/integrity!

Near-real time ingestion processes



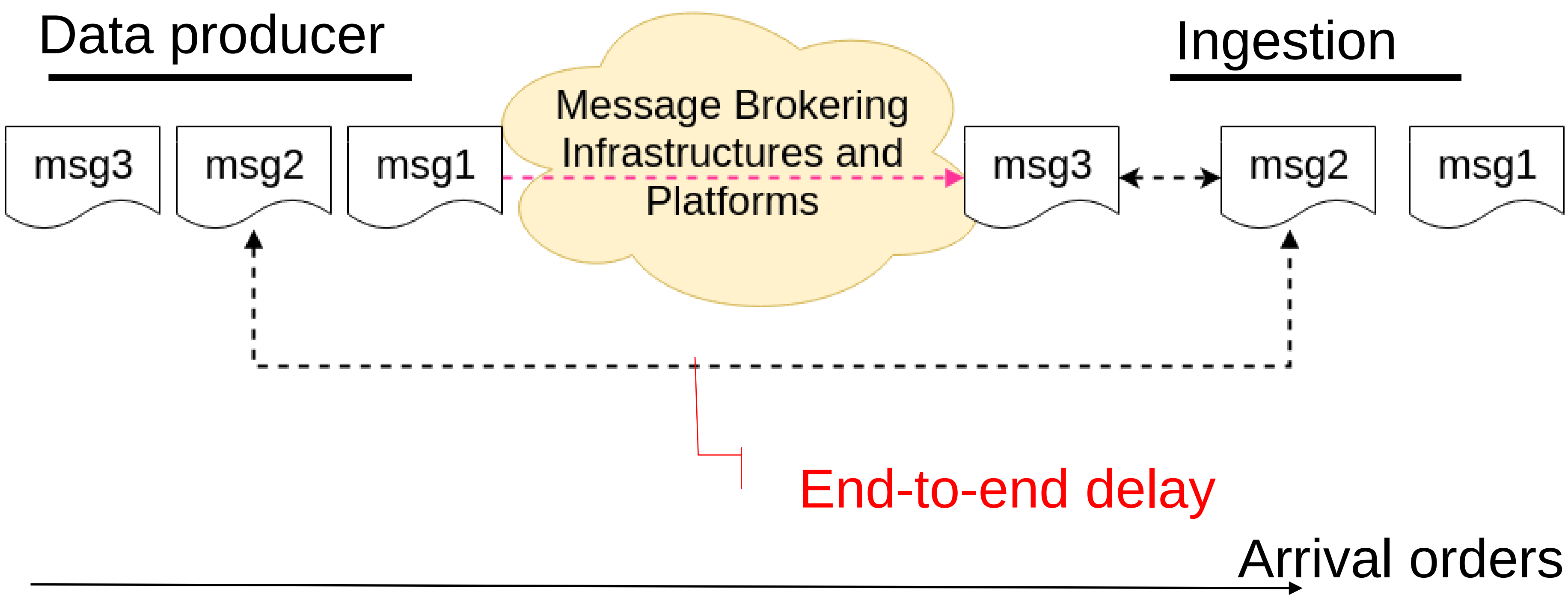
- **Moving streaming data**
- **Unbounded data, amount of data varies, fast ingestion**

Example



IoT device
Service
Log
....
IoT device
IoT device
IoT device
Message Broker
(Pub/SubSystems)
MQTT, AMQP,
Kafka, Pulsar
Ingest
Client
...
Ingest
Client
Ingest
Client
druid
Apache
hudi

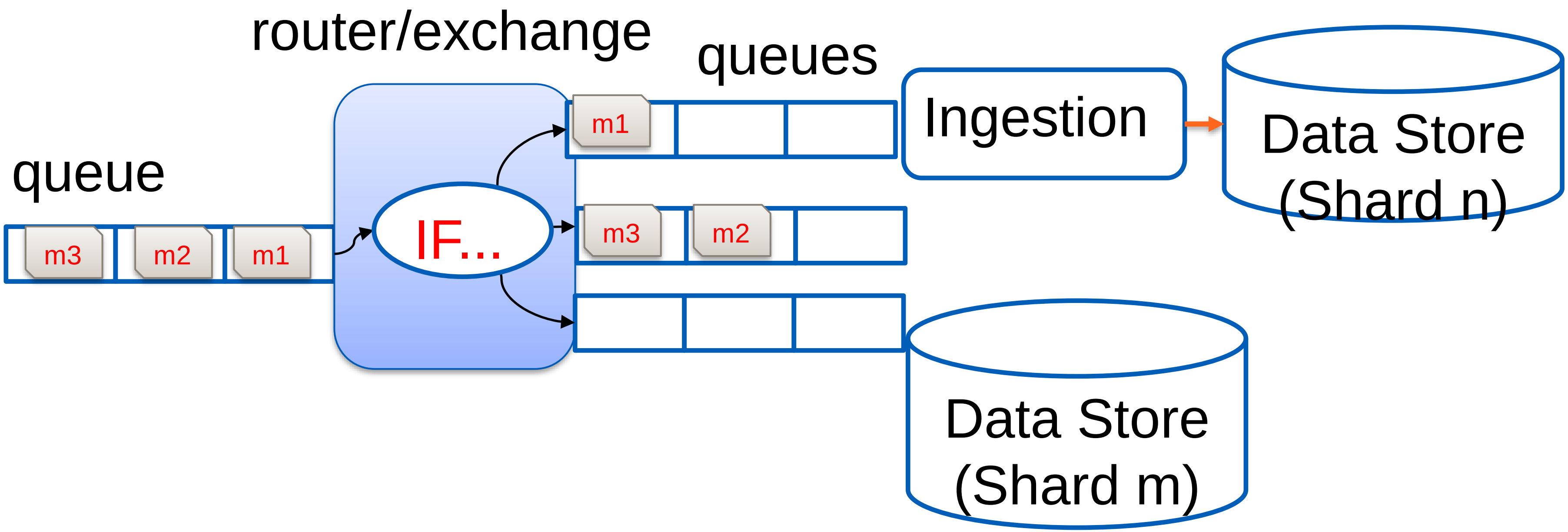
Key issues in streaming data ingestion



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

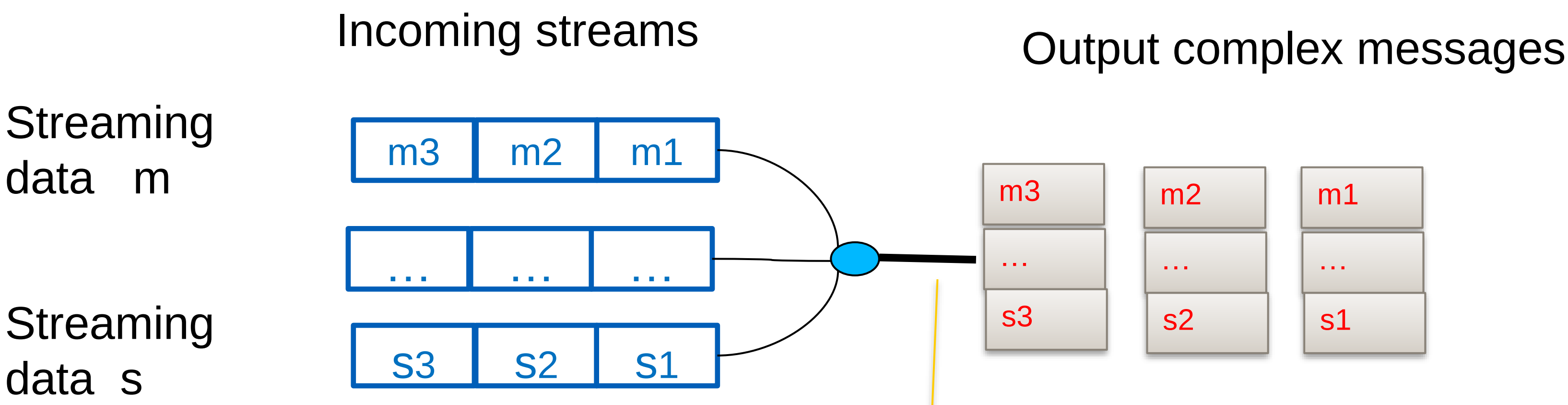
Split (pub/sub) and partition with ingestion



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

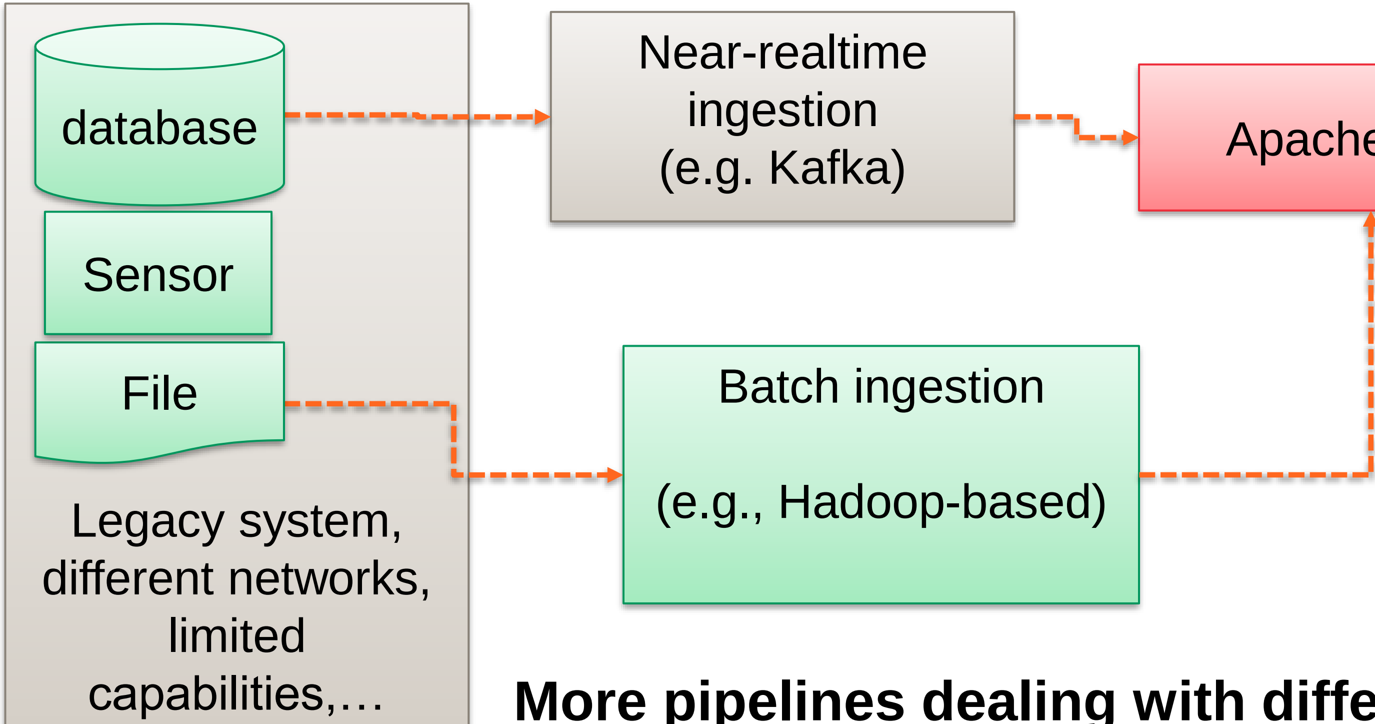
Complex ingestion pipelines in big data platforms

- **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



Other systems: Hadoop, Cassandra

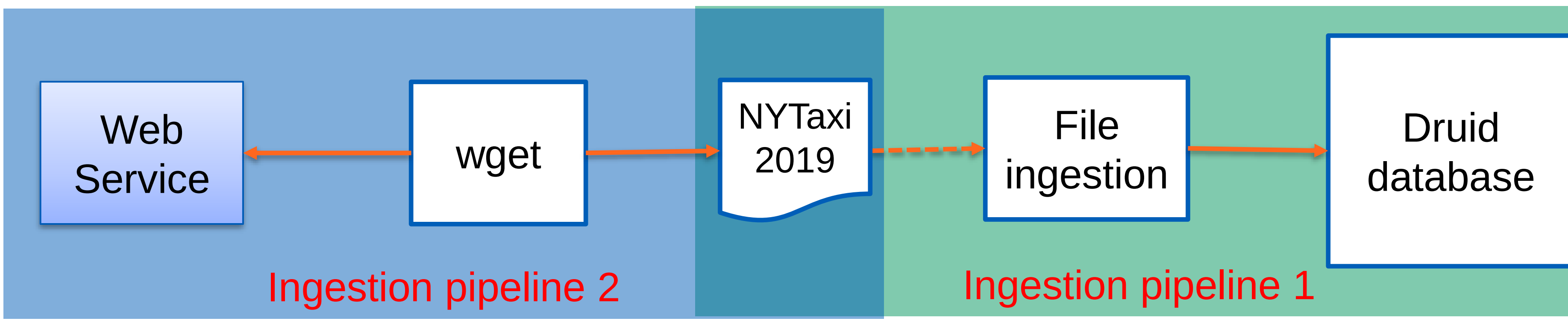
ElasticSearch

…

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/)

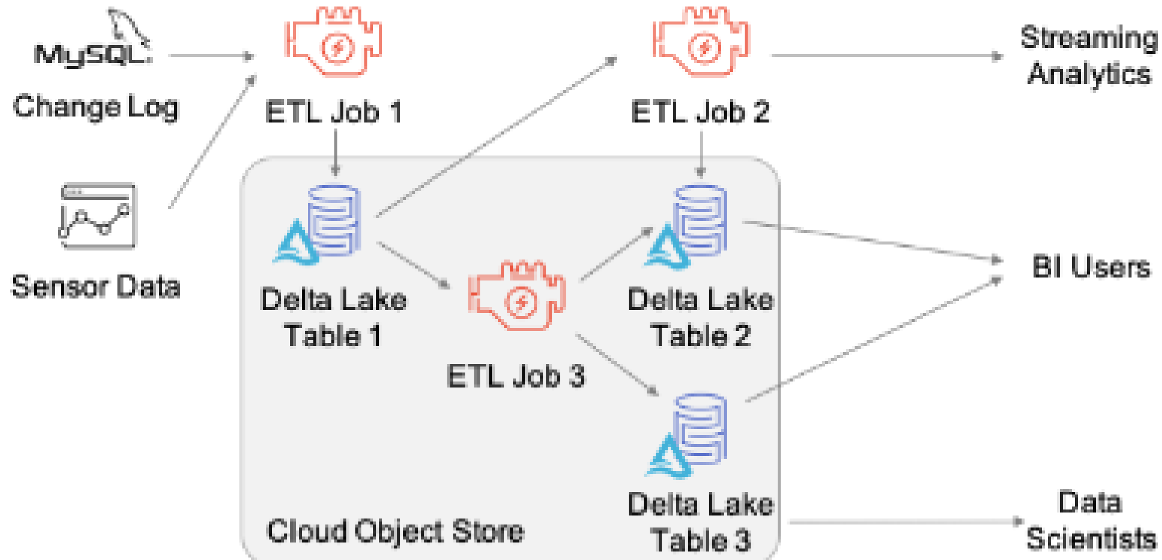


Figure source: "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores", https://databricks.com/wp-content/uploads/2020/08/p975-armbrust.pdf

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

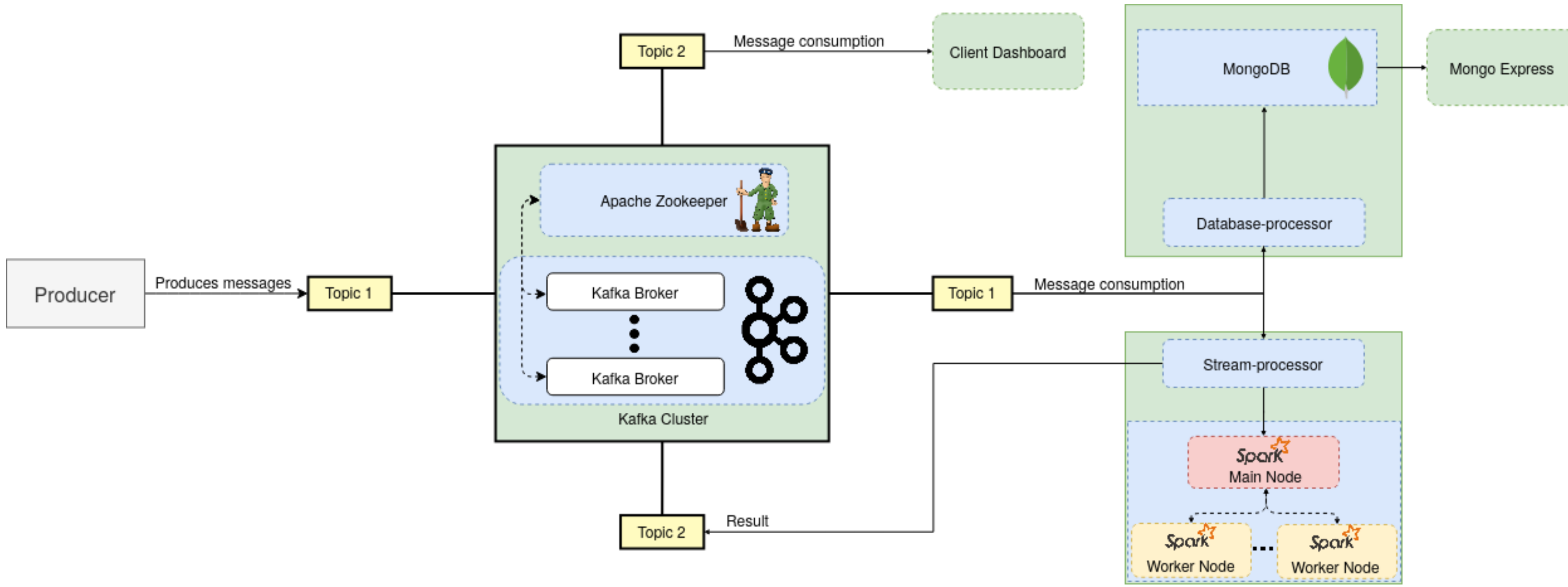
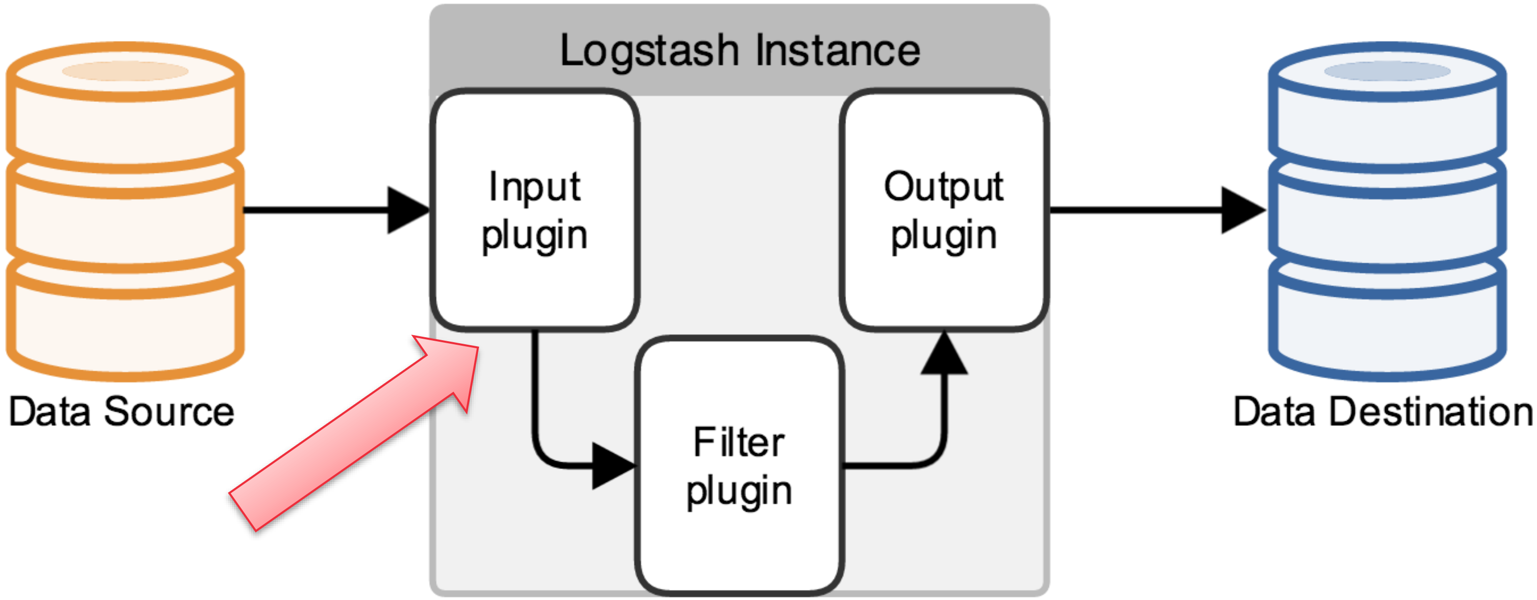


Figure source: https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/cloud-data-pipeline

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

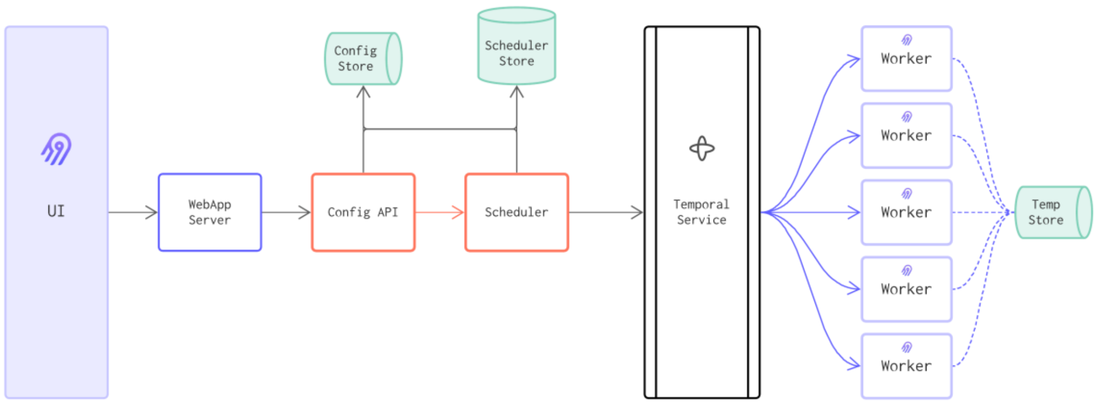


Pluggable approaches

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



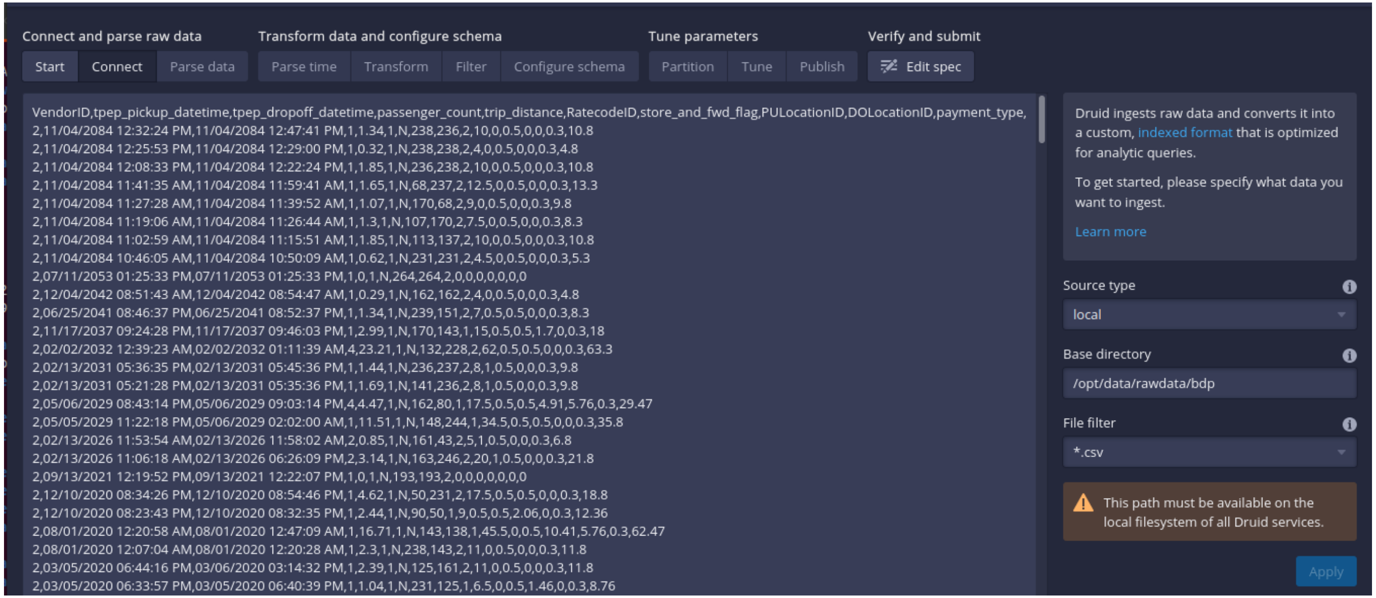
Connectors for multiple data sources

Platform with scheduler, jobs, workers for data ingestion

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

Design tools for ingestion processes: Apache Druid

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



Design tools for ingestion processes: Apache Nifi

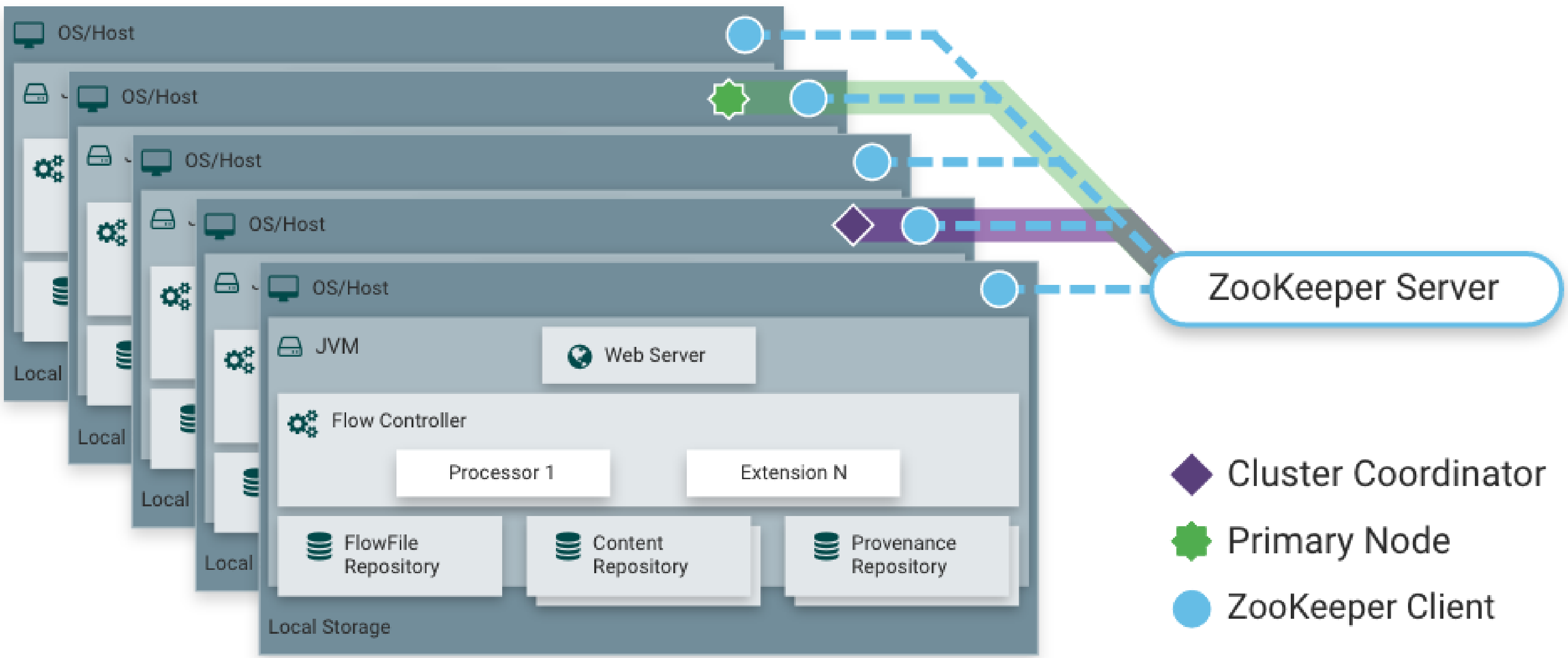
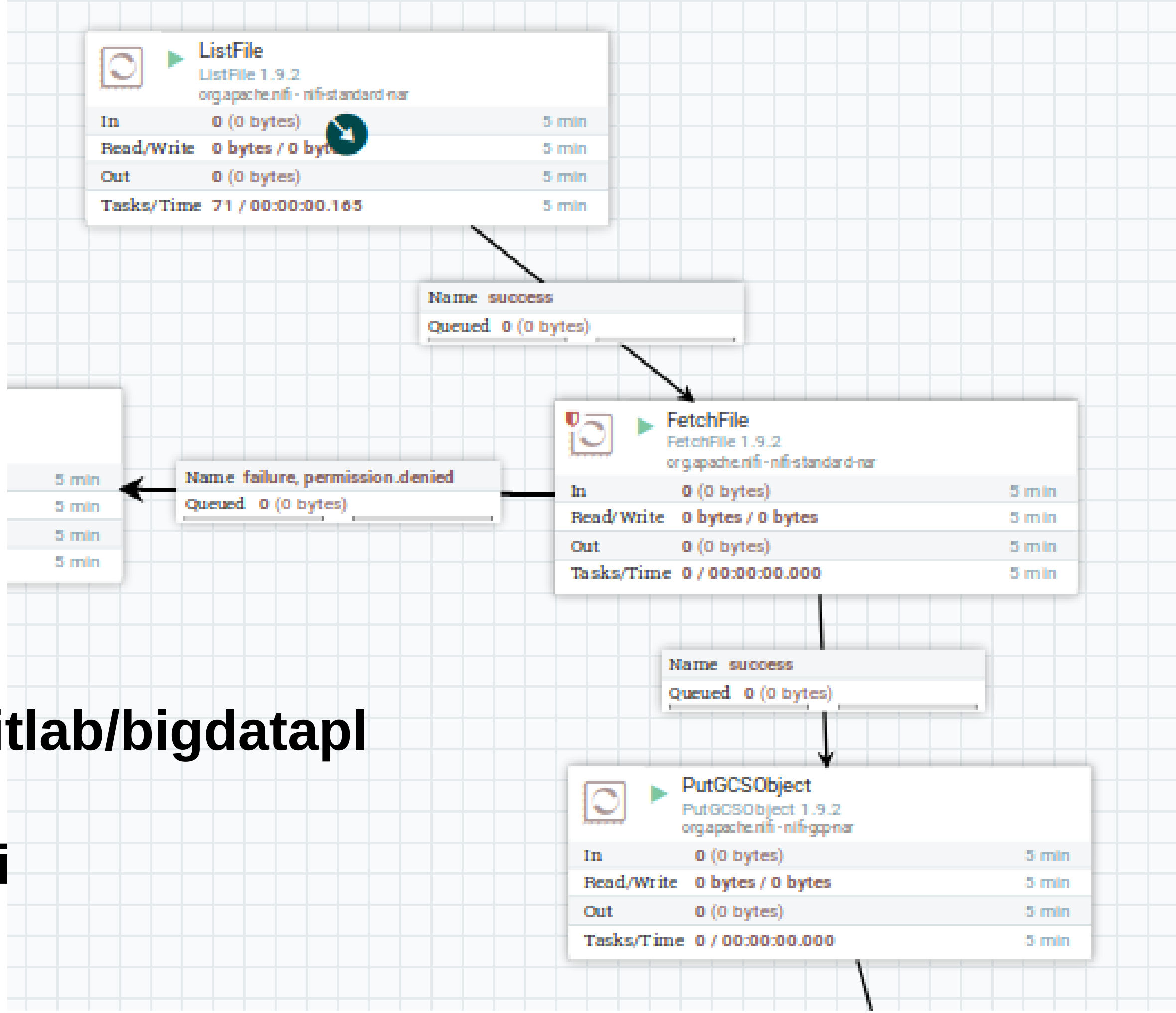


Figure source: https://nifi.apache.org/docs.html

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



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