



Aalto University  
School of Science

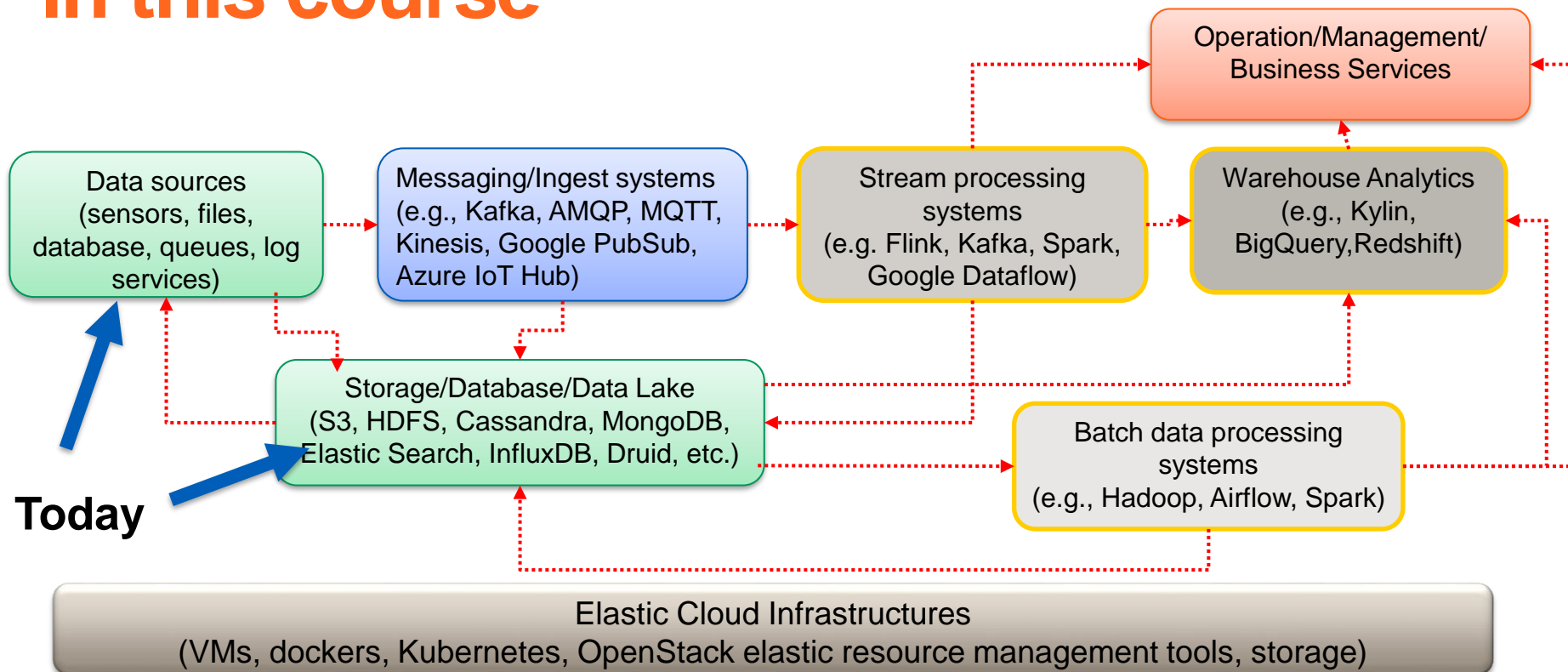
# Big Data Ingestion

*Hong-Linh Truong*  
*Department of Computer Science*  
*[linh.truong@aalto.fi](mailto:linh.truong@aalto.fi), <https://rdsea.github.io>*

# Learning objectives

- **Understand the overall design of data ingestion**
- **Study common tasks in data ingestions**
- **Understand and design 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



Big data platform

e.g.

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

**Two important aspects:**

- Requirements and tasks
- Architectures/Pipelines/Tools

**Reusability and extensibility are very important!**

# Big Data Ingestion

- **Data ingestion**
  - Move data from different sources into the big data platform
- **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 or not? (ELT)**

**Correctness and quality assurance are hard!**

# Fundamental ingestion models (1)

- **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.)

# Fundamental ingestion models (2)

- **(Near) real-time ingestion**
  - Data is encapsulated into messages
  - Ingest data as soon as the data is available
  - Message brokers are needed
- **Messages (unit of self-contained data)**
  - Text/CSV/JSON, ARVO
  - Application-specific designs

# Data source and sinks

## Data sources

Input files

REST  
Services

Messaging  
Systems  
(MQTT,  
KafKa, etc)

Databases

Ingestion  
Tasks

## Data sinks

Storage/File  
Systems

Output  
files

Big Database

Big data store systems

## Big data platform examples:

Hadoop File systems  
Google Storage  
Amazon Storage

Google BigQuery  
Hive  
MongoDB  
ElasticSearch  
Cassandra  
InfluxDB



# 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**

# Data access and extraction tasks

- **Access**

- Obtaining data from data sources
- Often built based on common protocols and APIs
- Reusability is important!

- **Encryption/masking**

- Might need to be done when accessing and extracting data
- Also during transfers of data
- data security requirements, personally identifiable information

# 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**
- **But semantics are domain/application-specific**

# Example: Arvo

## Syntax specification

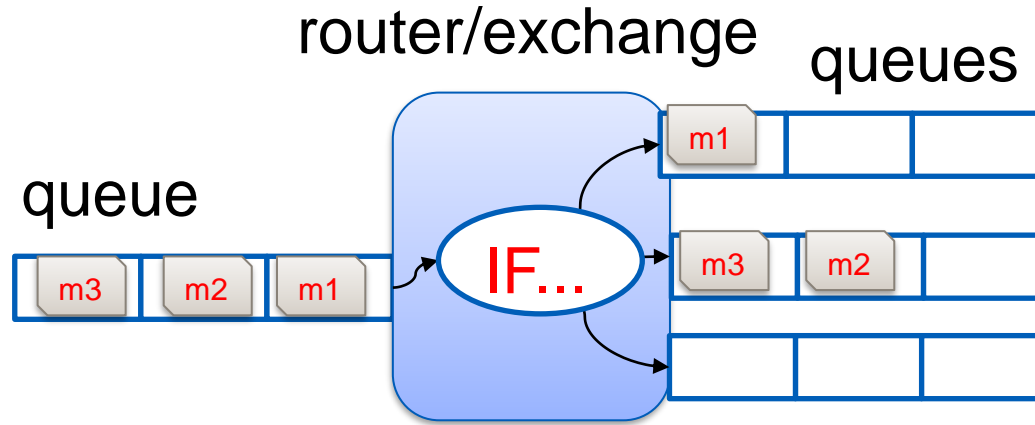
```
{  
  "namespace": "bdp.courses.aalto.fi",  
  "type": "record",  
  "name": "event",  
  "fields": [  
    {"name": "station_id", "type": "string"},  
    {"name": "datapoint_id", "type": "int"},  
    {"name": "alarm_id", "type": "int"},  
    {"name": "event_time", "type": "int"},  
    {"name":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



# Some other techniques

- **Protobuf, <https://github.com/google/protobuf>**
  - From Google, used by default in gRPC (gRPC.io)
  - Language-neutral, platform-neutral mechanism for serializing/deserializing structured data
- **Parquet, <https://parquet.apache.org/>**
  - Columnar storage
  - In Hadoop ecosystem
- **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

# Data routing: split tasks/distributor patterns

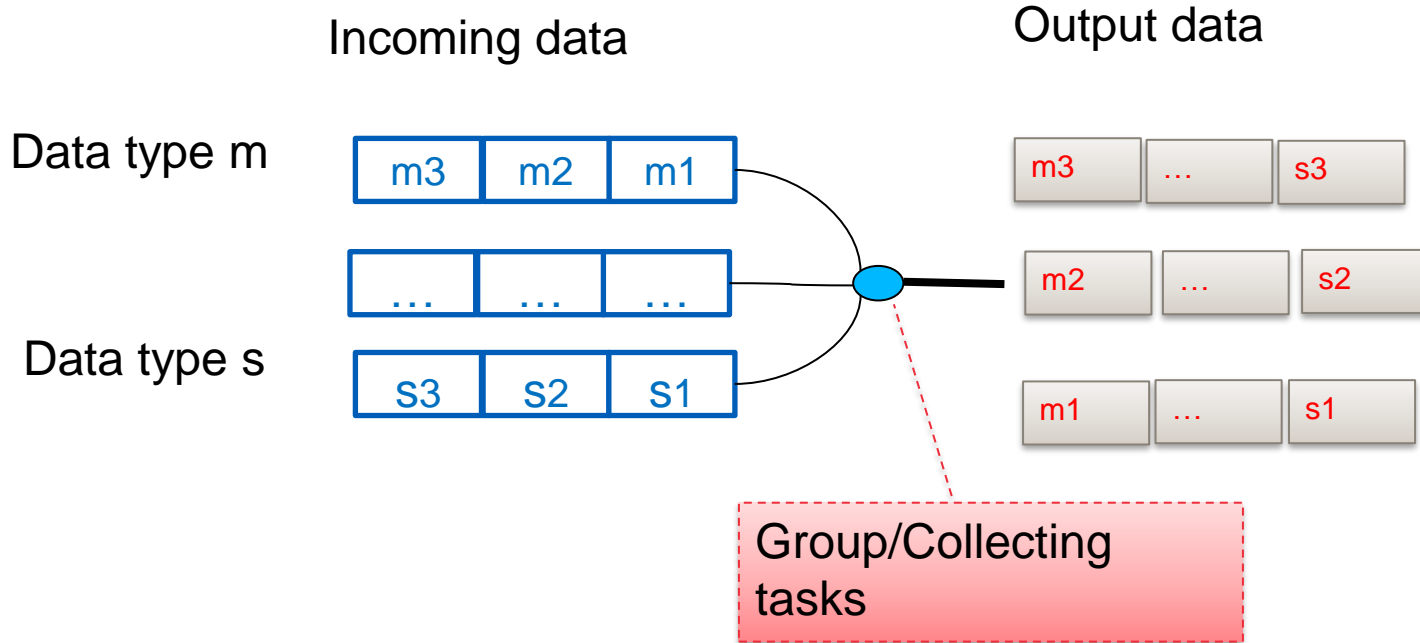


Read the famous book: Enterprise Integration Patterns

<https://www.enterpriseintegrationpatterns.com/patterns/messaging/>



# Data routing: grouping data/Collector pattern



# Data wrangling

- **Convert data from one form to another**
  - Cleaning, filtering, merging and reshaping data
- **Require access to the data!**
- **Key design choice: do you support it during the ingestion or after the ingestion?**

# 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 platforms to support debug, monitoring and handling exceptions
  - Runtime management for wrangling
- **Wrangling programs provided by platforms**
  - Constraints in dealing with customer data

# Examples

Write your  
own code with  
Pandas and  
Data frame?  
Similar with  
your code!

```
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']];
                    severies[typeOfAlarm]=severies[typeOfAlarm]+1
                else:
                    severies =[row['RNW Object Name'],0,0,0,0,0,0]
                    severies[typeOfAlarm]=severies[typeOfAlarm]+1
                    Alarms[row['RNW Object Name']]=severies;

        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 file
#only for debugging
print dataframe
dataframe.to_csv(outputFile, index=False)
```

# 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

```
1 input {  
2   file {  
3     path => "/tmp/alarmtest2.txt"  
4     start_position => "beginning"  
5   }  
6 }  
7 filter {  
8   grok {  
9     match => {"message" => "%{NUMBER:AlarmID};%{DATESTAMP:Start};%{DATESTAMP:End};%{WORD:Severity};%{NOTSPACE:NetworkType};%{NOTSPACE:BSCName};%{NOTSPACE:Sta  
10  }  
11 }  
12 output {  
13   stdout {}  
14   csv {  
15     fields => ['AlarmID', 'Start', 'Stop', 'Severity', 'NetworkType', 'BSCName', 'StationName', 'CellName', 'AlarmInfo', 'Extra', 'AlarmStatus']  
16     path => "/tmp/test-%{+YYYY-MM-dd}.txt"  
17   }  
18 }
```

# Ingestion tasks as extensible, composable connectors

- **Basic tasks for big data ingestions that 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 processes: architectures and tools



# Architecture requirements

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

# Complex deployment and integration models

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

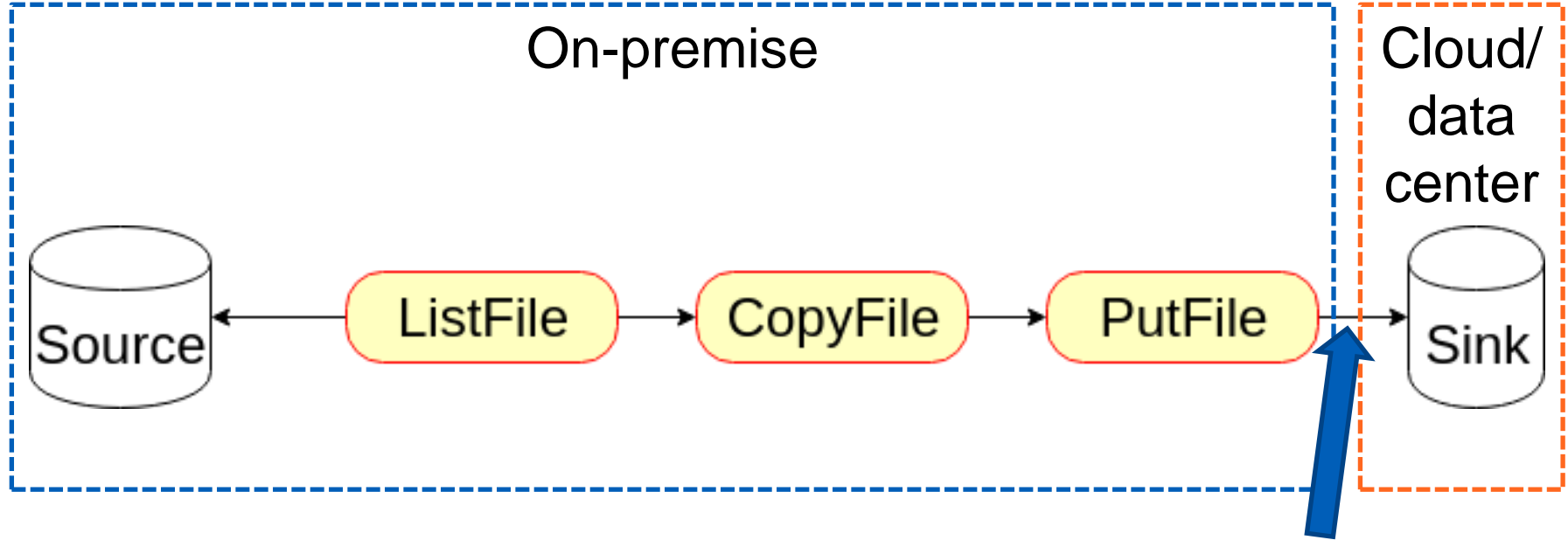


**Customer**

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

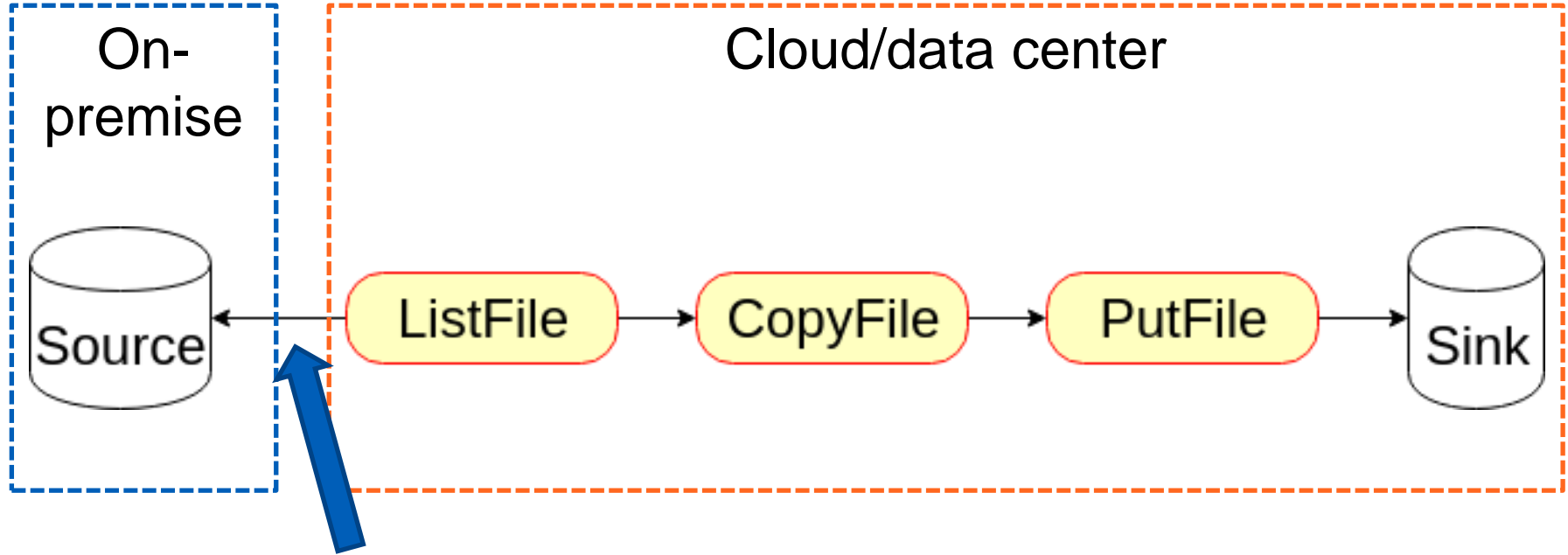
**Data  
store/platform  
provider**

# Complex deployment and integration models



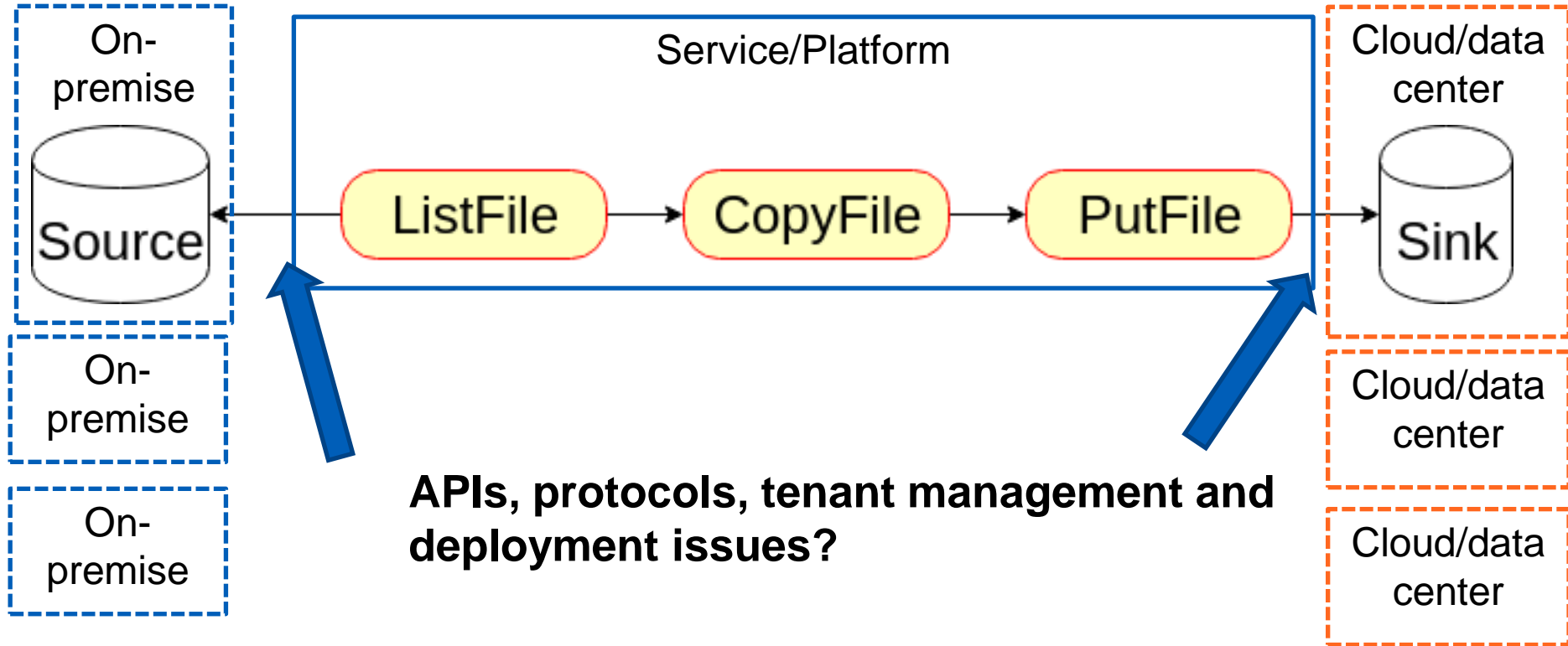
**APIs, protocols and deployment issues?**

# Complex deployment and integration models



**APIs, protocols and deployment issues?**

# Complex deployment and integration models

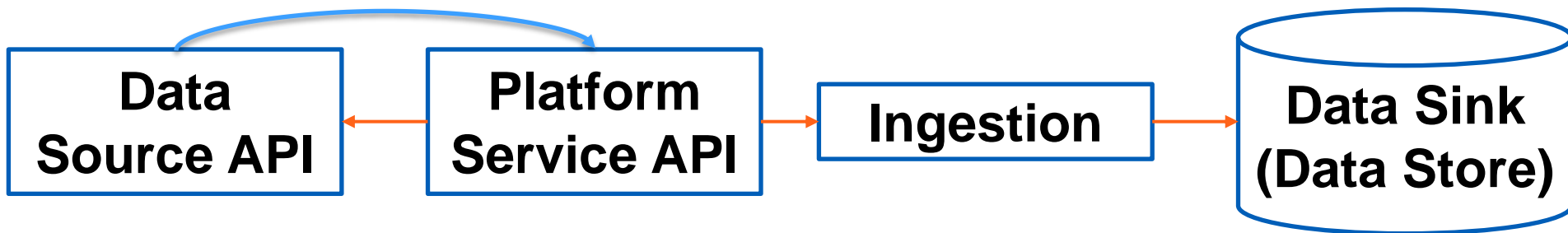


# Batch ingestion processes

- **Data to be ingested is bounded**
  - files or messages are finite
- **Ingestion architectural styles**
  - (1) Simple 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
- **Parallel and distributed execution**
  - Use workflows and distributed processing

# Simple APIs for ingestion

**Pull model:** register webhook/API

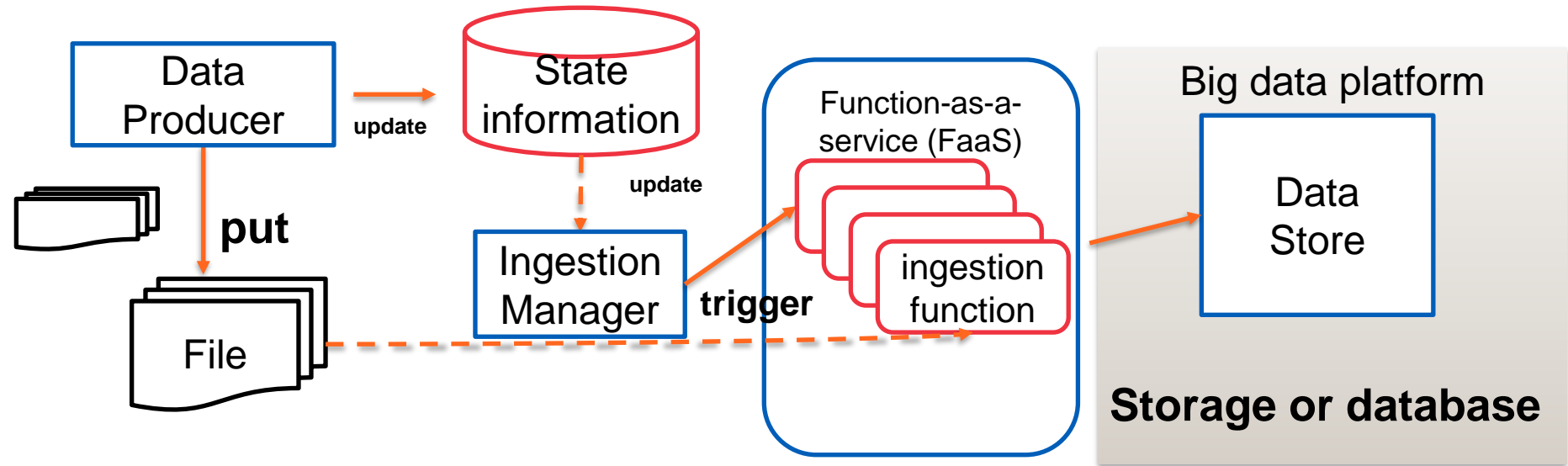


**Push model**  
When?



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

# Reactive with function-as-a-service



## Who develops which components?

Remember:

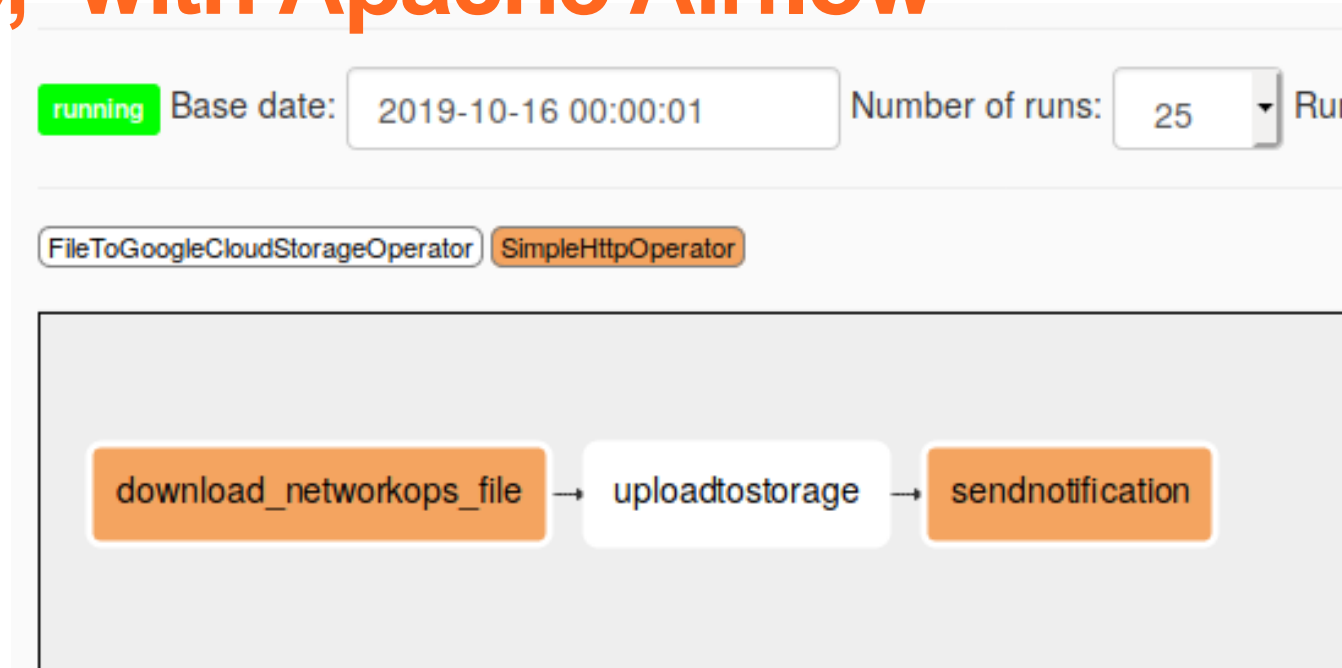
<https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/queuebaseddataingestion>



# 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**

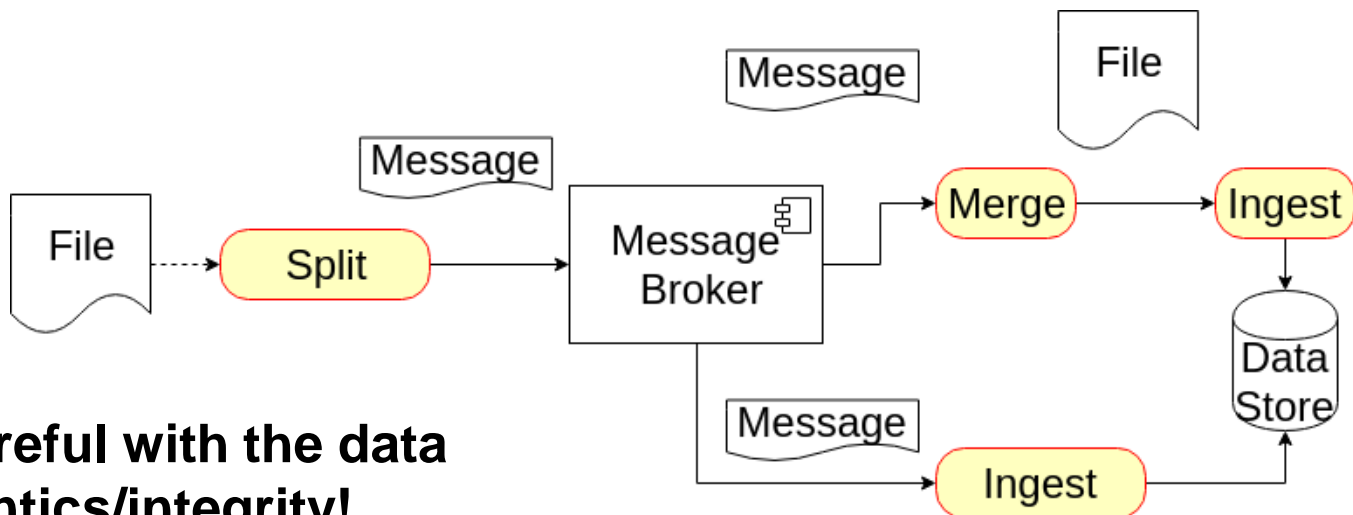
# E.g., workflow based on scheduled time, with Apache Airflow



**How would your design be changed, if a type of ingestion tasks or task cannot be handled by the engine?**

# Microbatching for ingestion

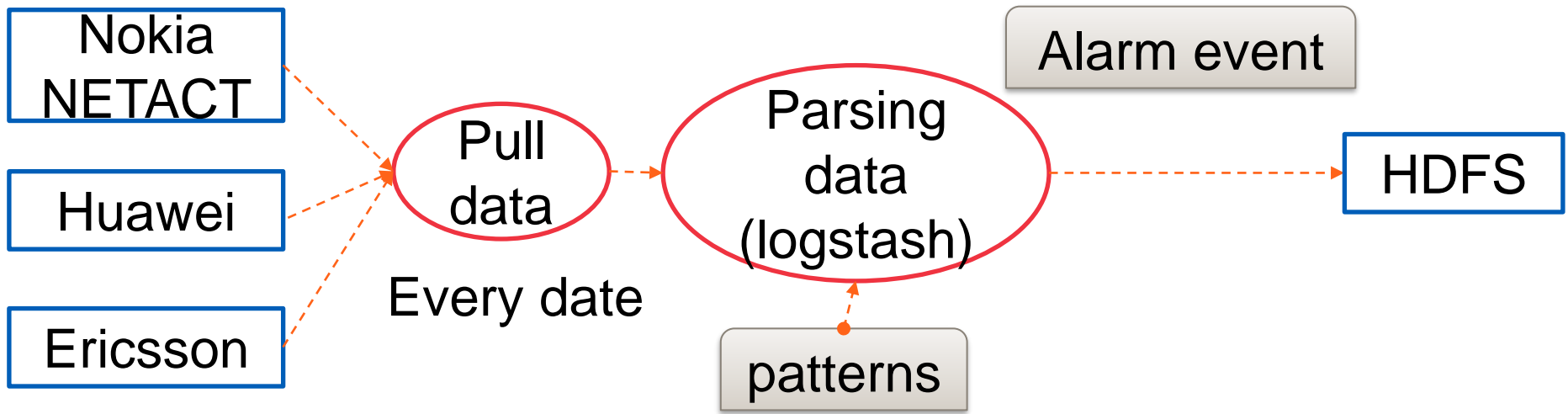
- **Data is split into different chunks ingested using a batch**
  - Using “streaming” to send chunks
  - Chunks are ingested into the system, or merged and then ingested



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

# Microbatching is useful for applying filter and quality control

## Example



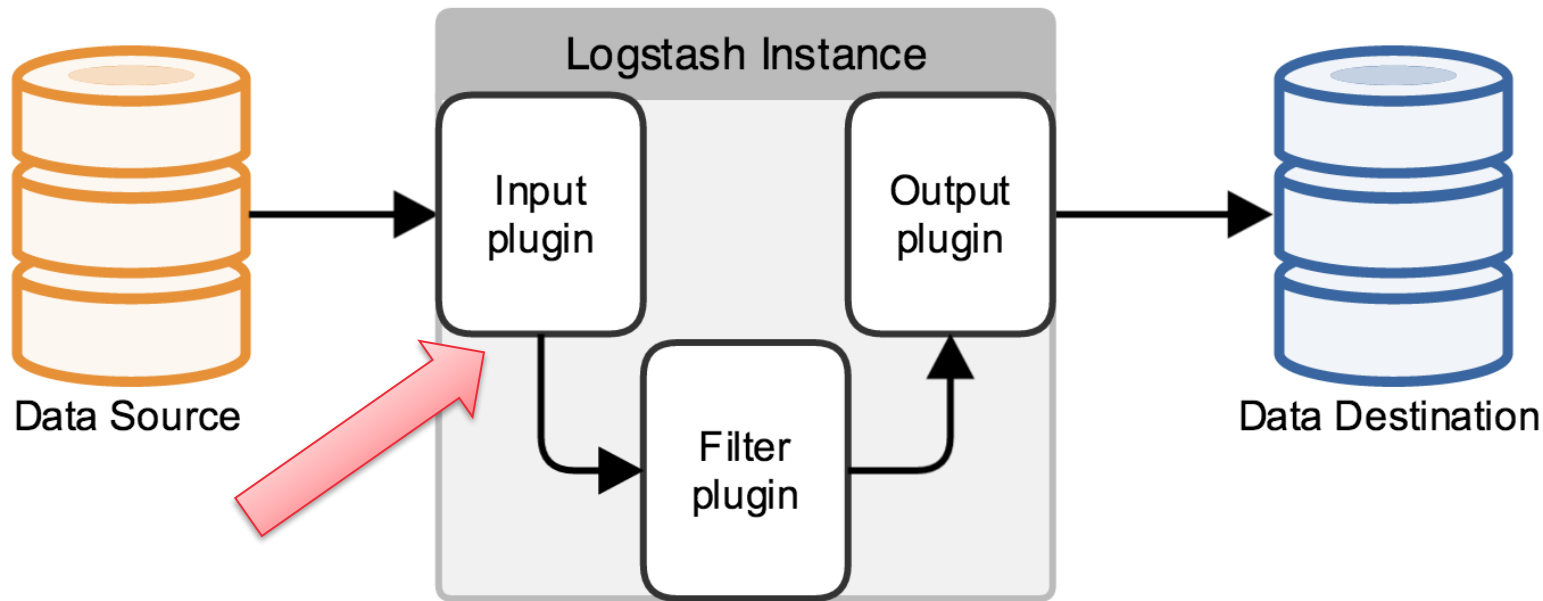
## Telco devices

# 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**

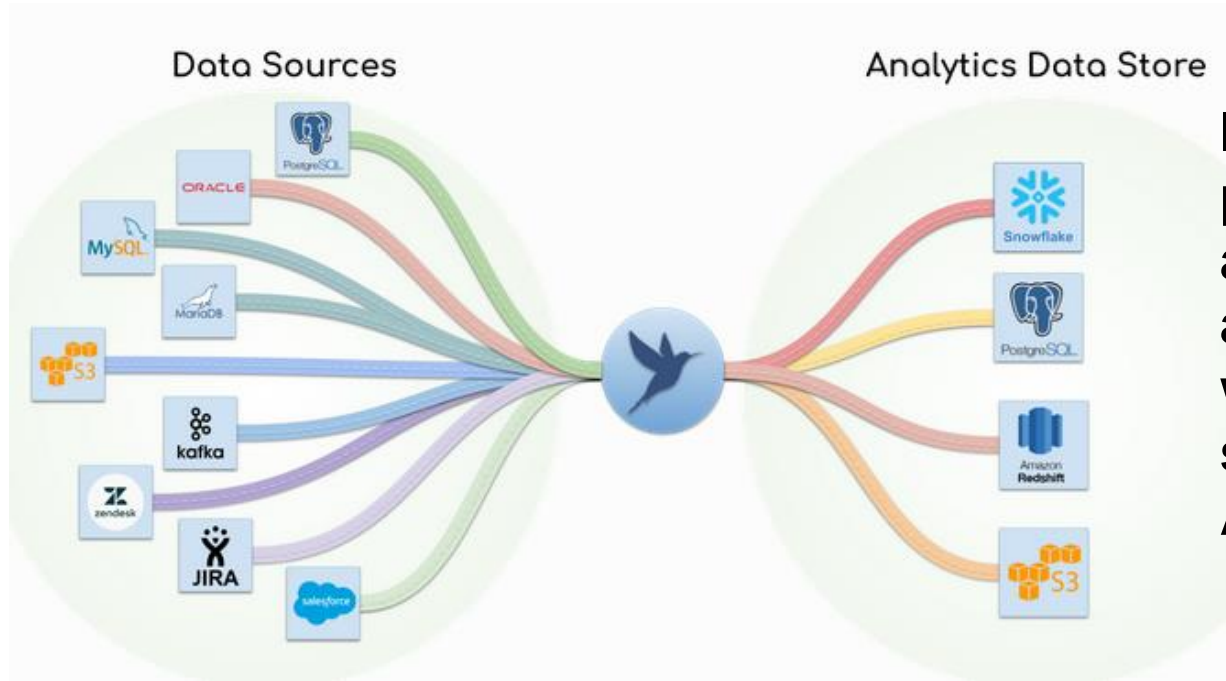
# Tools for ingestion processes : Logstash



## Pluggable approaches

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

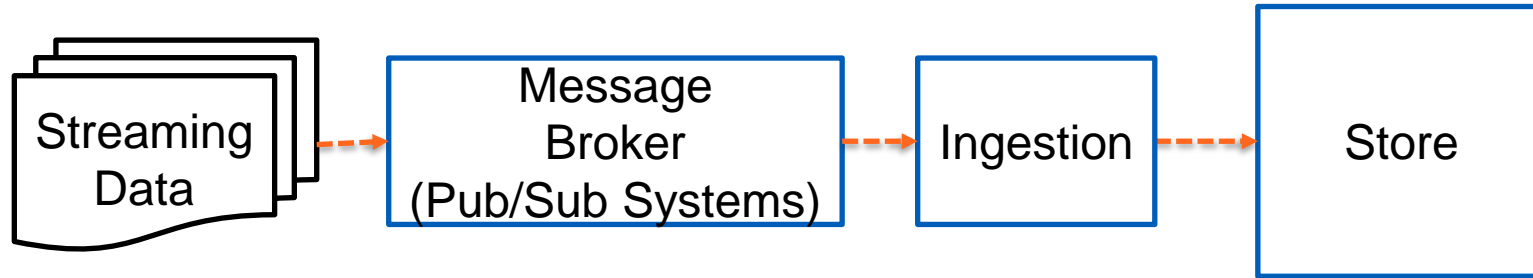
# Tools for ingestion processes: PipelineWise



**Ingestion pipelines/processes are described in YAML and can be scheduled with different schedulers (e.g. Cron or Apache Airflow)**

Figure source: <https://transferwise.github.io/pipelinelwise/>

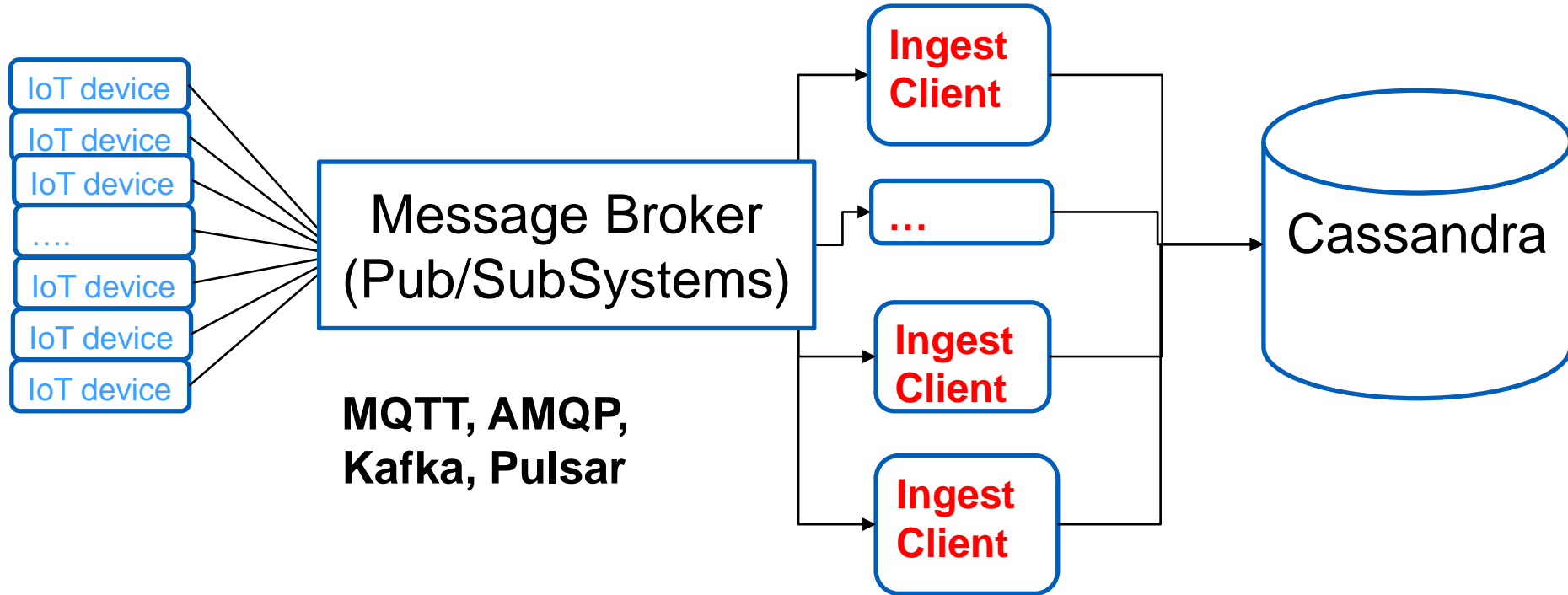
# Near-real time ingestion processes



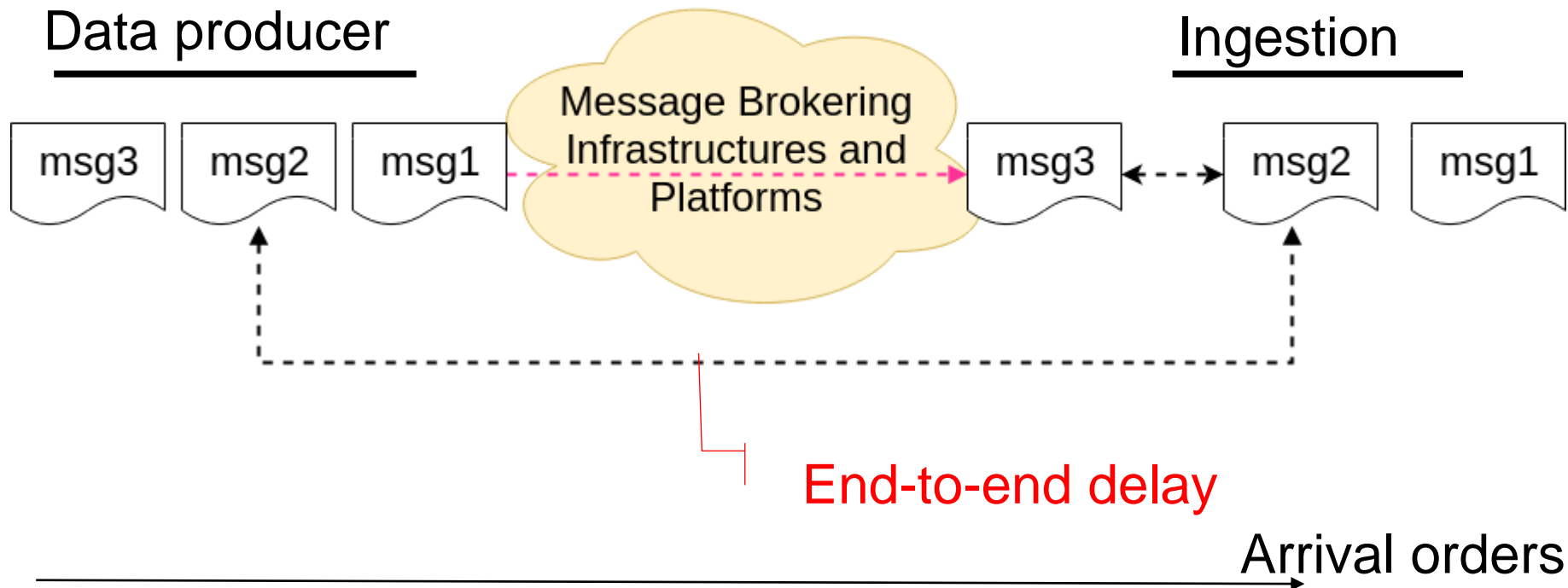
- **Moving streaming data**
- **Unbounded data, amount of data varies, fast ingestion**



# Example



# Key issues in streaming data ingestion



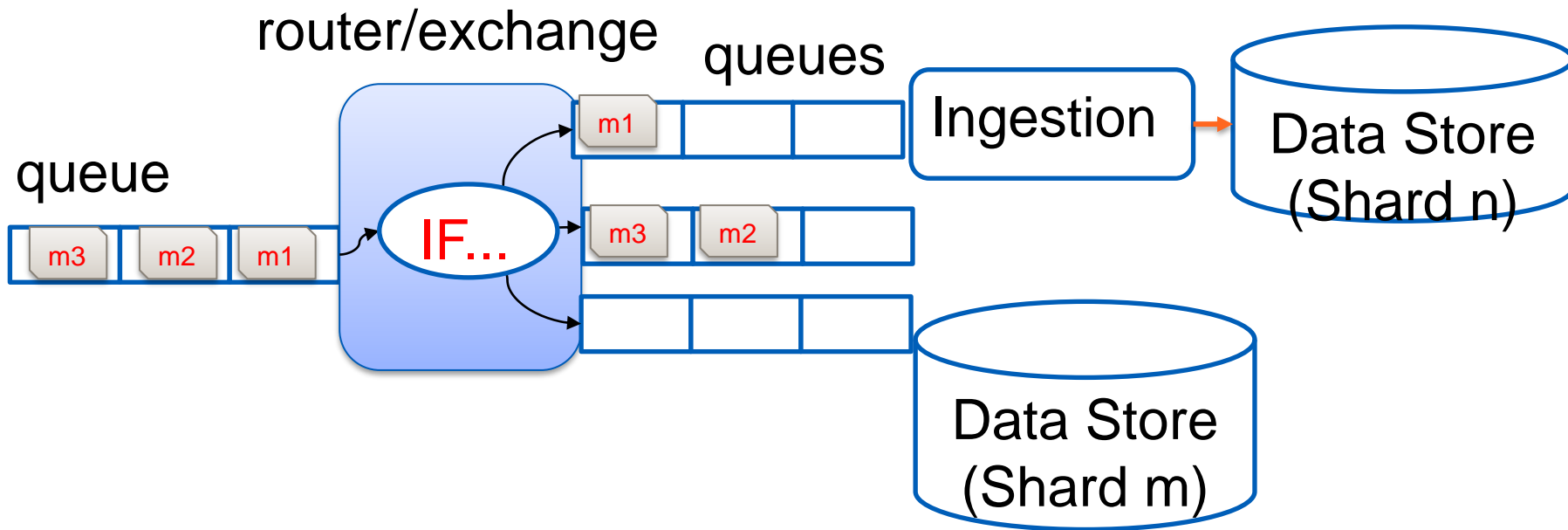
# Some key issues

- **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

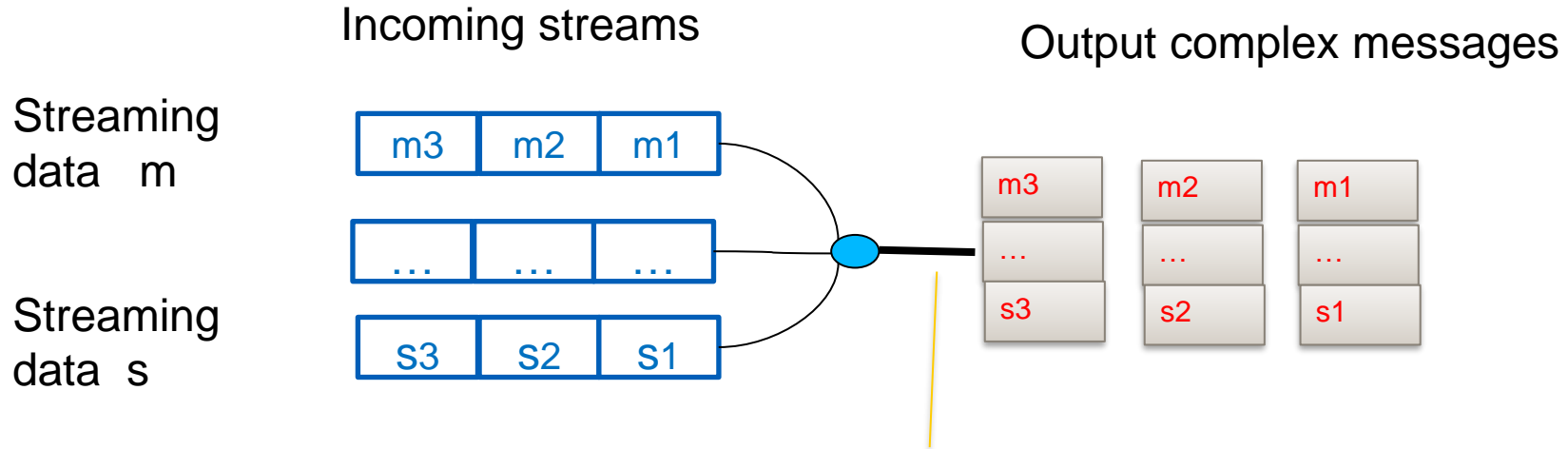
# Some key issues

- **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?**

# Split (pub/sub) and partition with ingestion

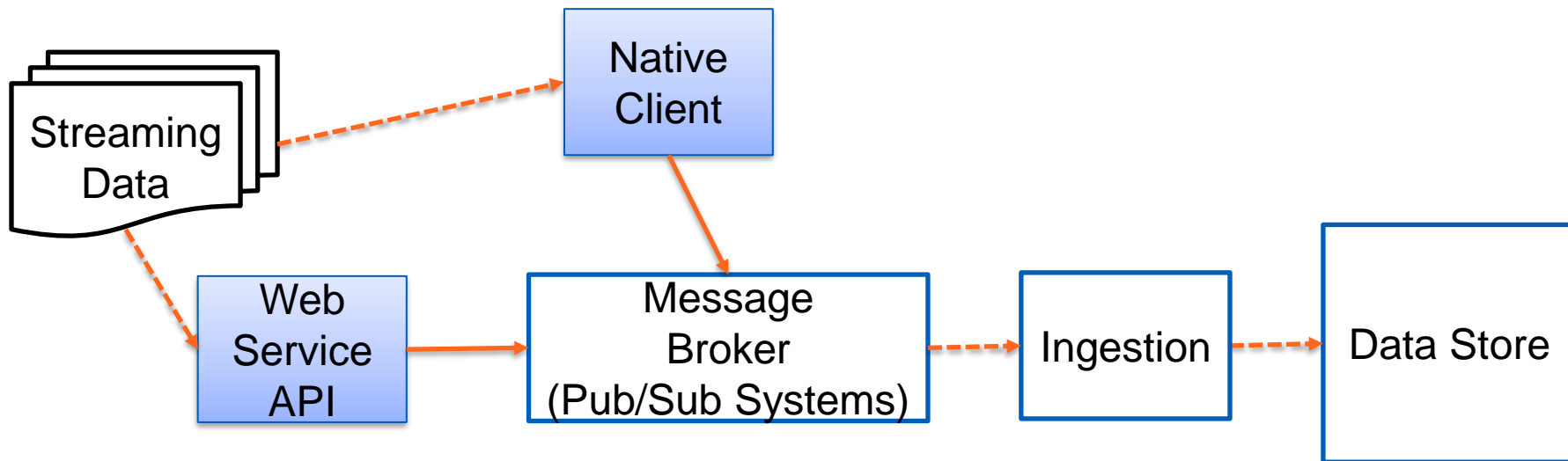


# Do we have to merge data before ingestion



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

# Which types of APIs for integration?



## Pros and cons?

# Tools: Apache Kafka

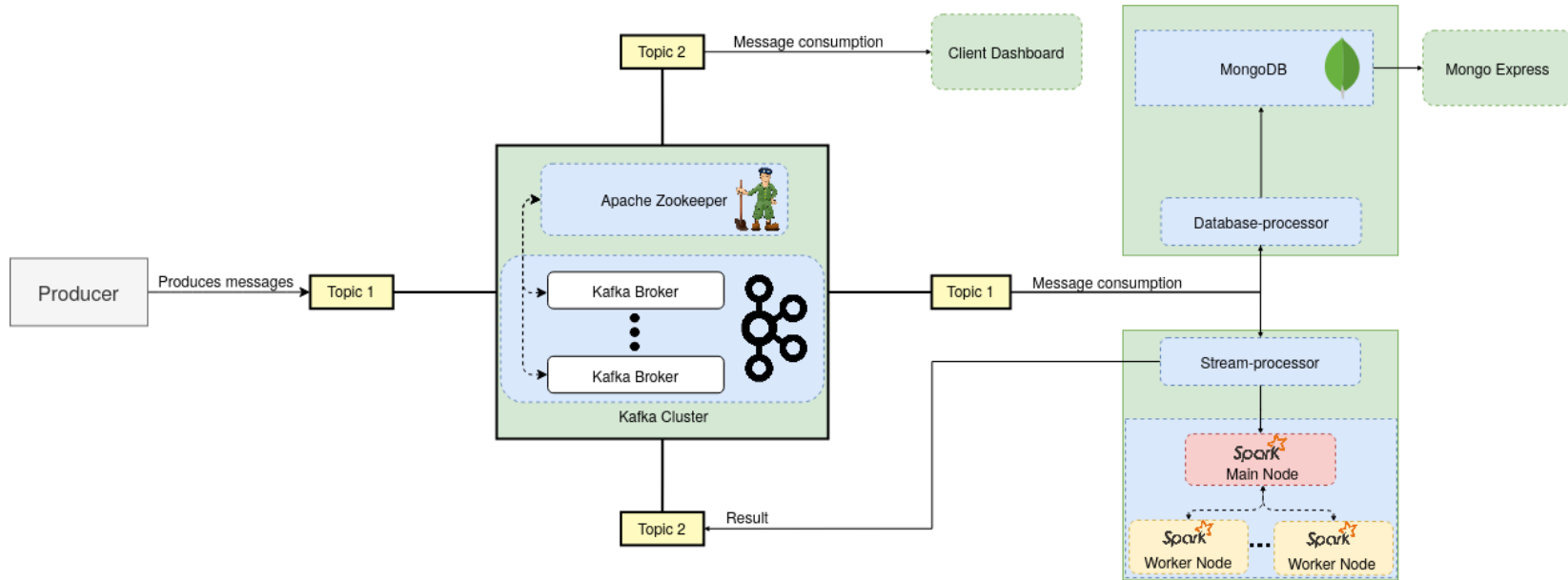


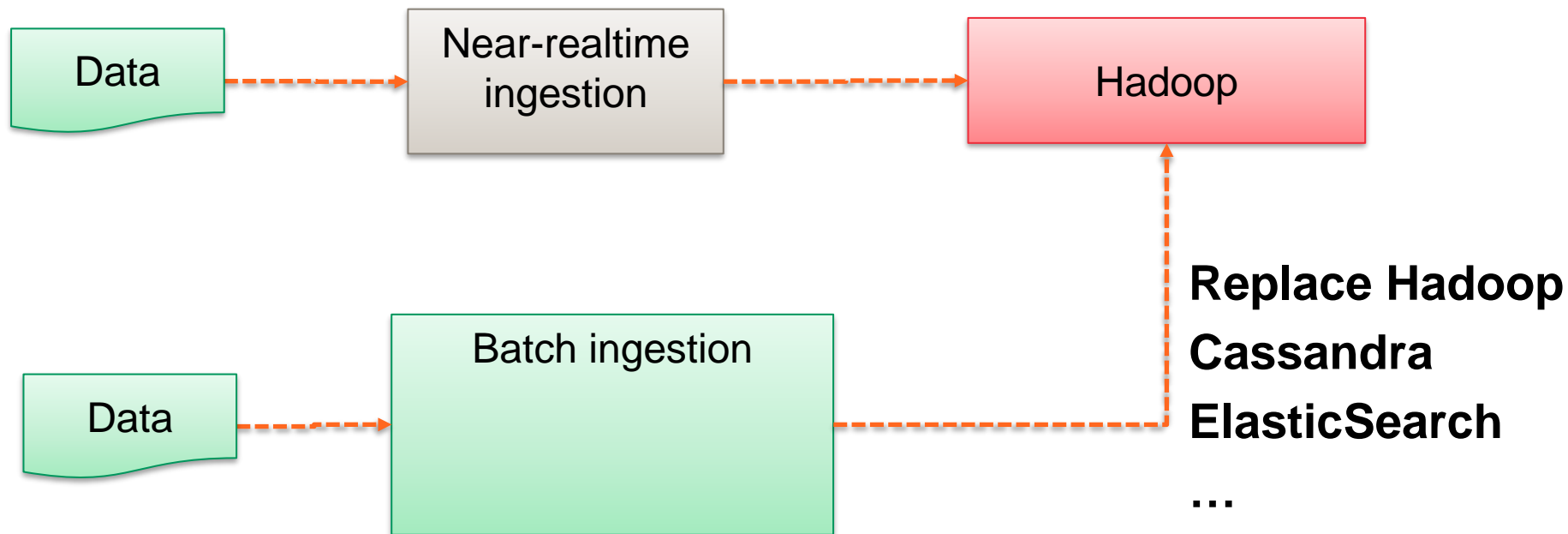
Figure source: <https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/cloud-data-pipeline>



# Complex ingestion pipelines in big data platforms

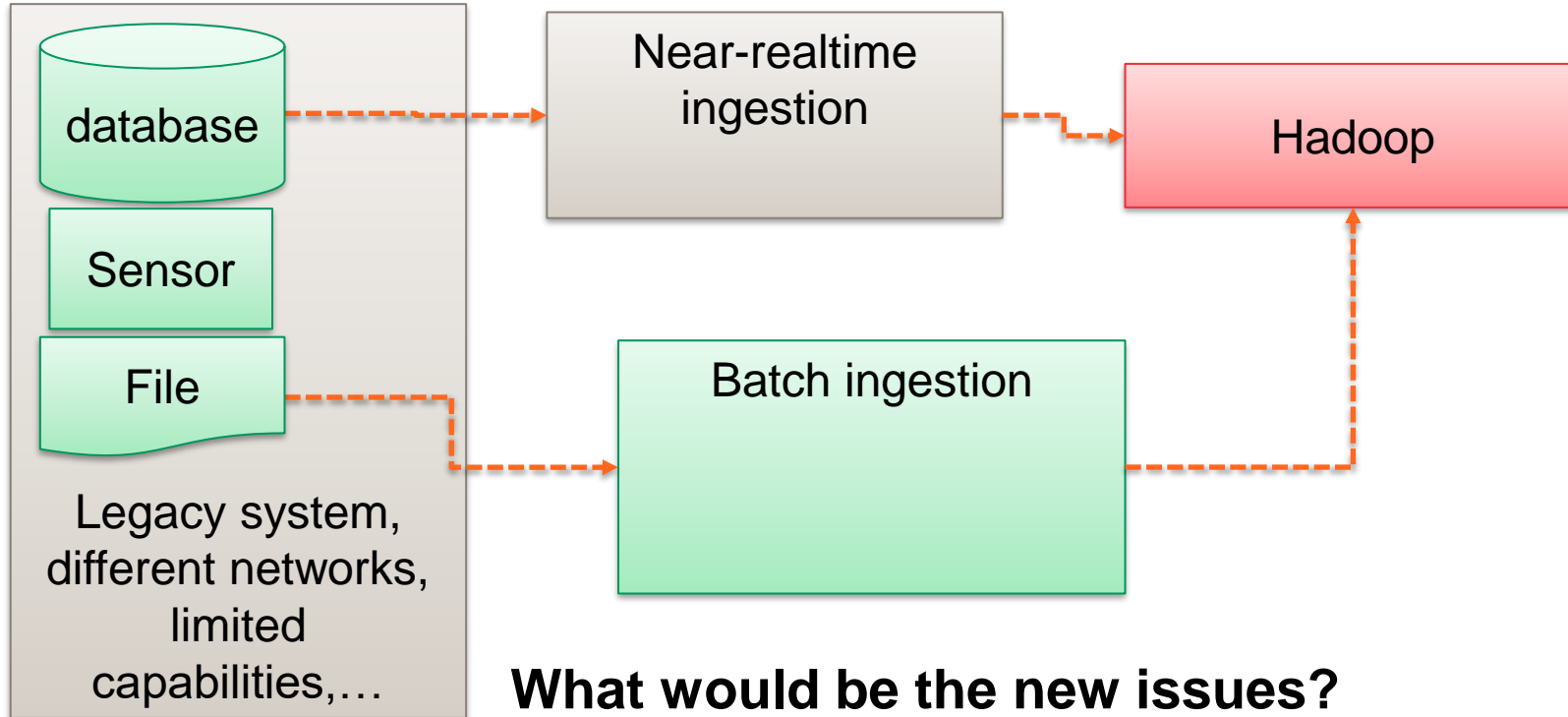
- Multiple types of pipelines for multiple types of customers
  - A customer might need different integrated pipelines
- Both batch and near-realtime ingestion are supported
- Complex architectural designs
    - Ingestion pipeline-to-pipeline needs “bridges”

# Multiple types of pipelines for the same destination



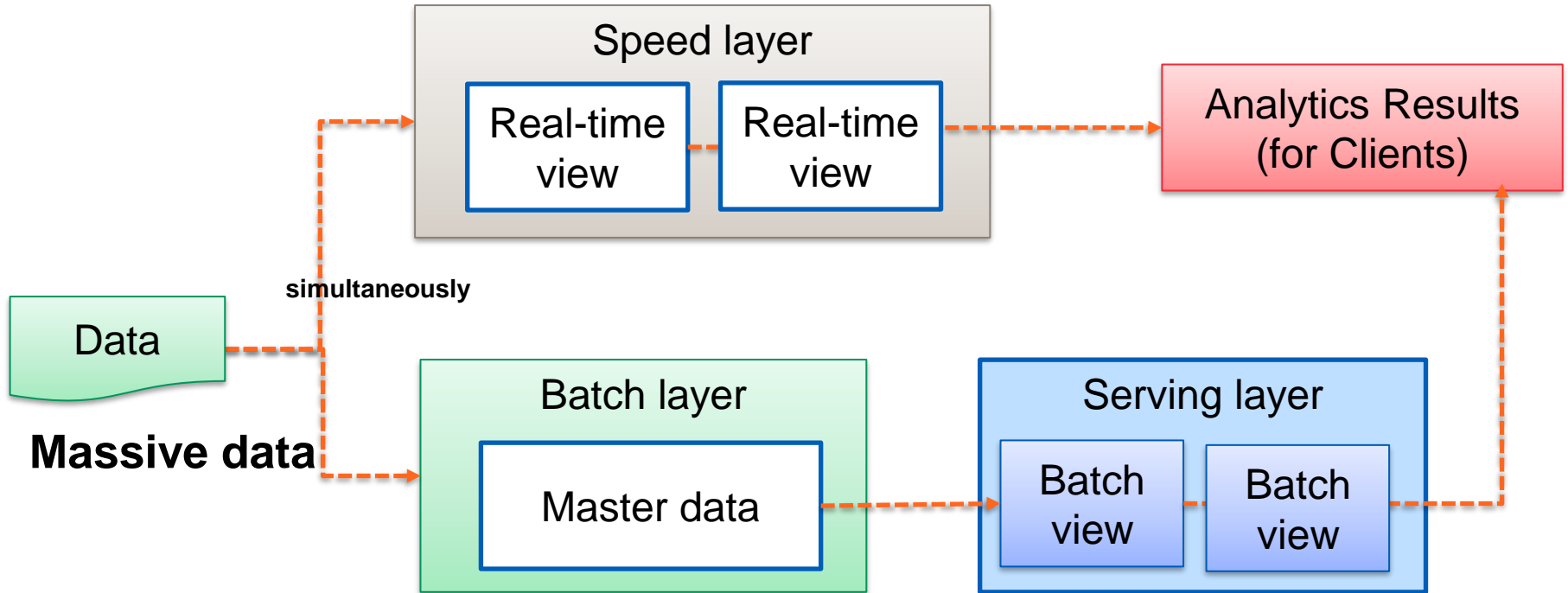
**How much code can we reuse?**

# Multiple types of pipelines for the same destination



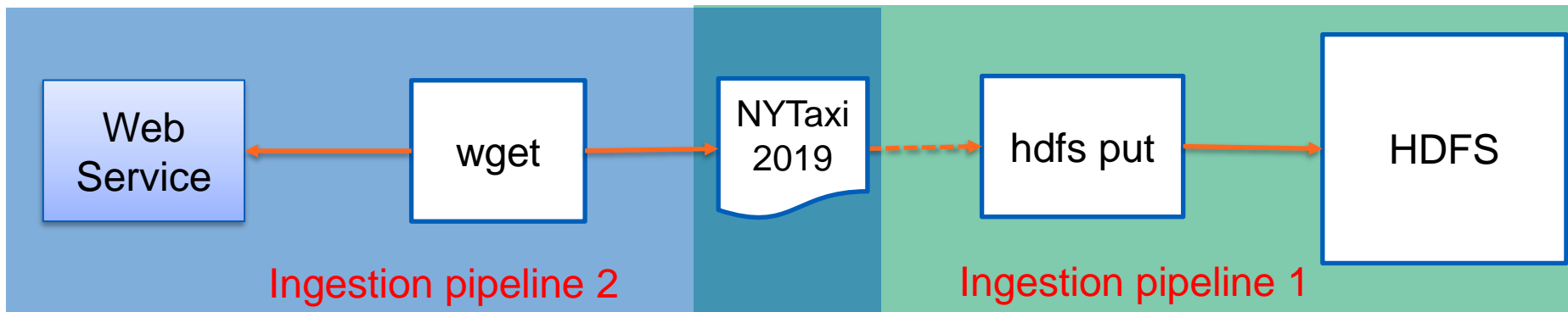
**What would be the new issues?**

# Recall: Lambda (see previous lectures)



# Connecting different ingestion pipelines

A single tool might not be enough



**Real-world:**

**Both pipelines and their connection are complex**

# Tools: Apache Nifi

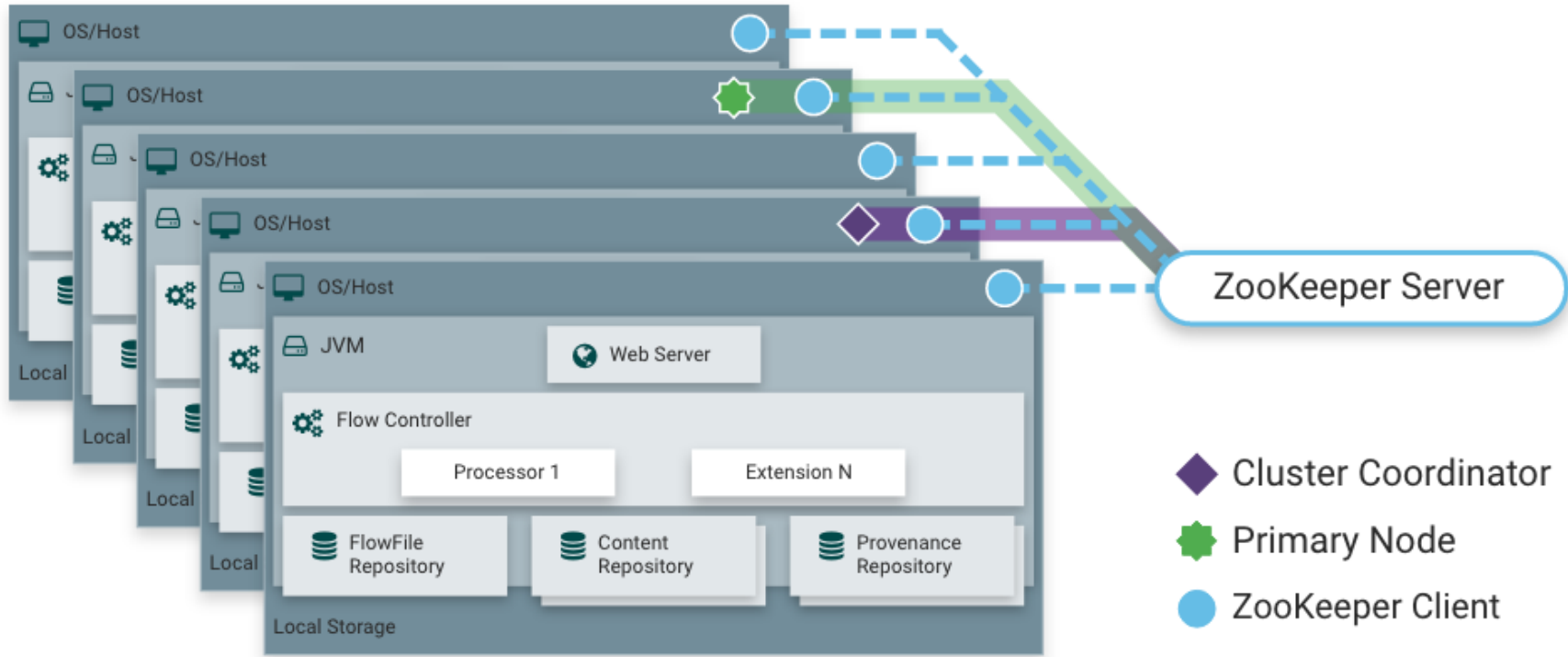
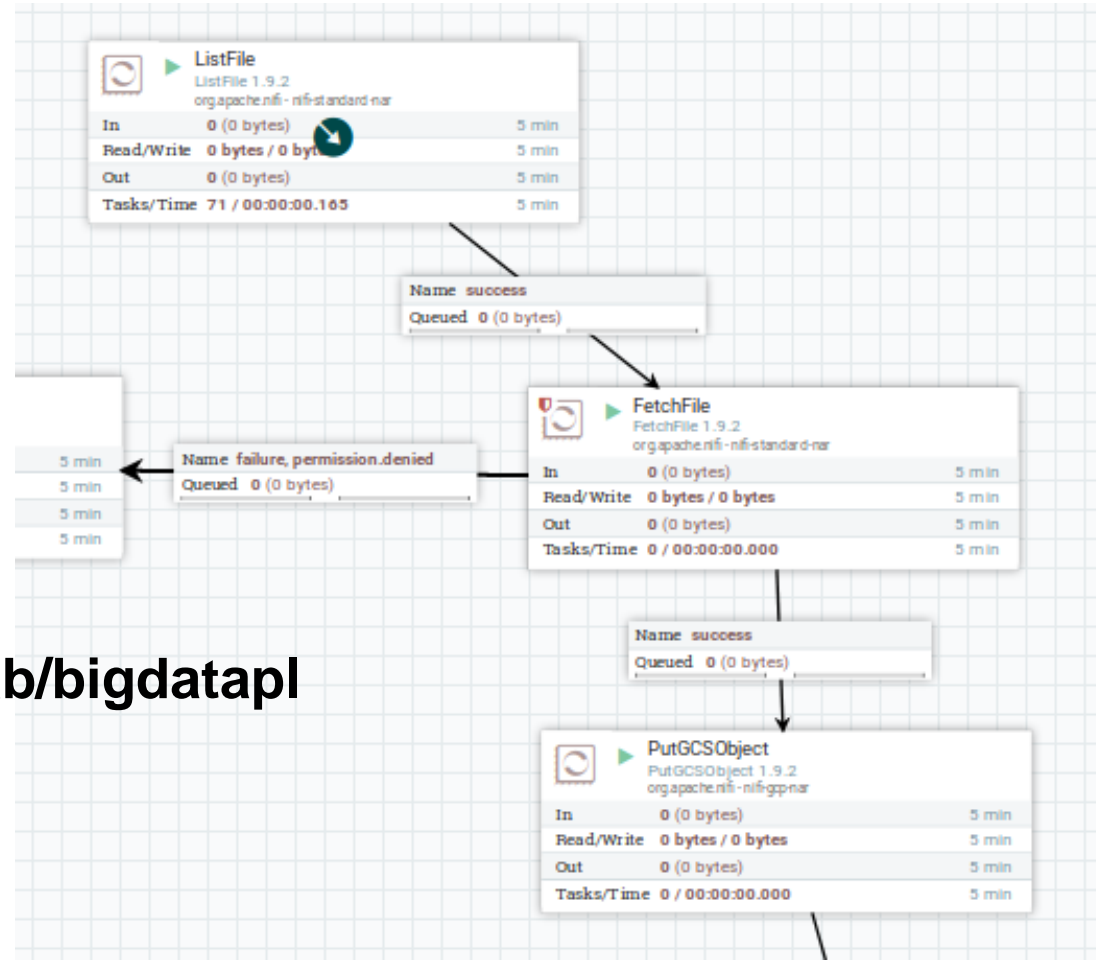


Figure source: <https://nifi.apache.org/docs.html>

# Tools: 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

# Example



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**