Aalto University
School of Science

Big Data Ingestion, Transformation and Orchestration

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

Aalto University
School of Science

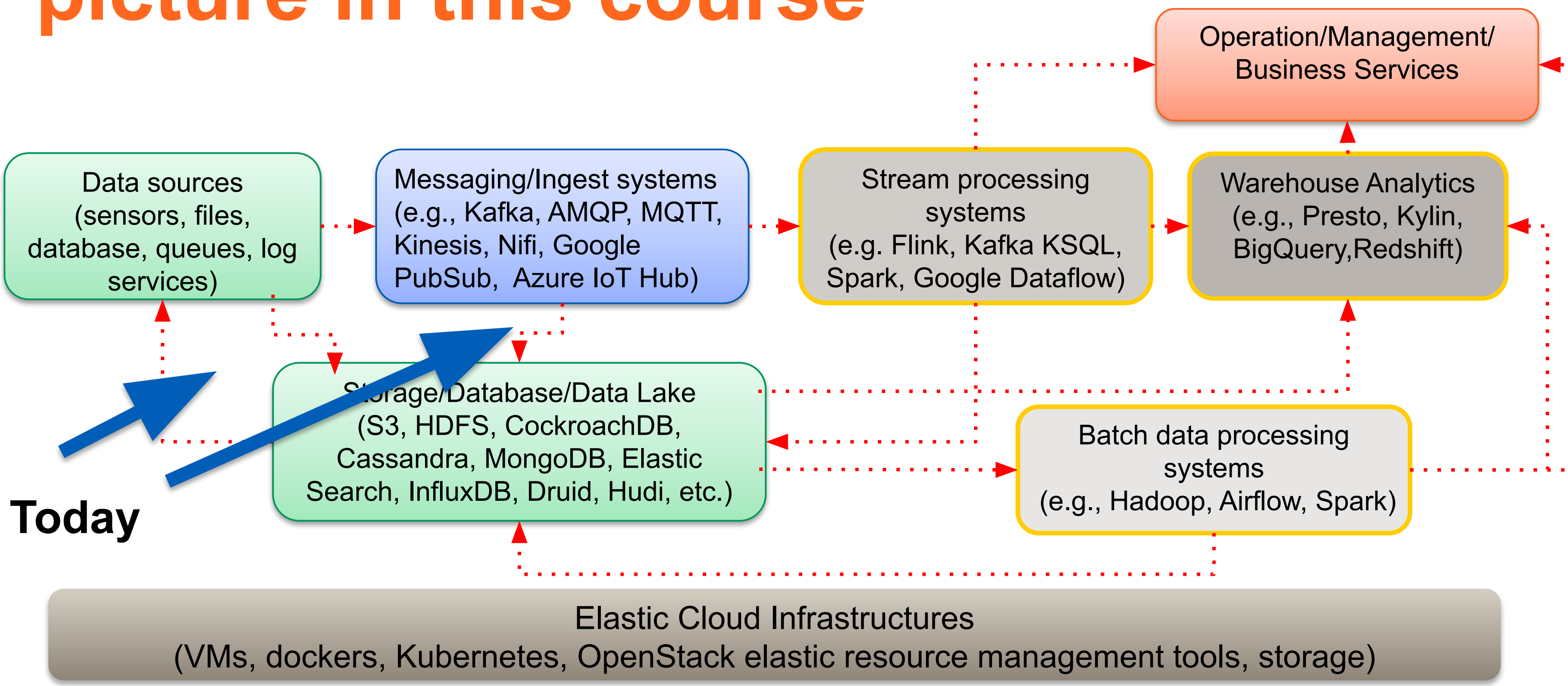
Bring the data to the right place!
where is the right place?

data store (database/storage)
target streams
running programs

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



Recall: Ingest big data into platforms

Data ingestion: move data from different sources into the big data platform (or selected destinations/sinks)

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!

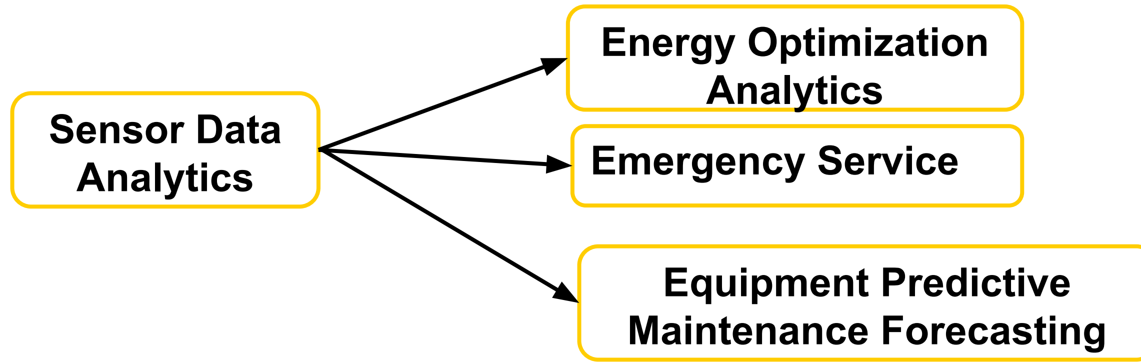
Transformation

- **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**
  - called ELT (load and then transformation)

Correctness and quality assurance are hard!

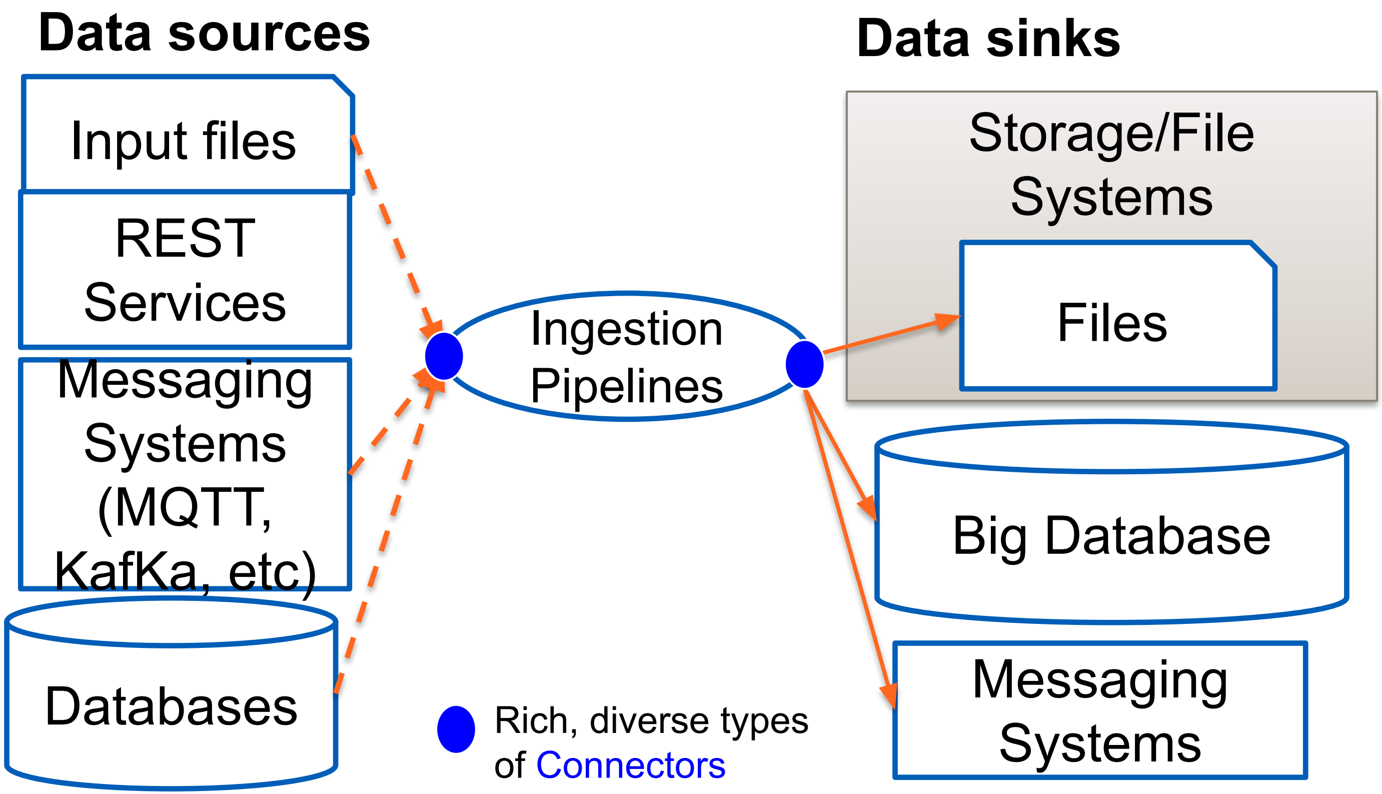
Orchestration

- **Remember**



- **Big data ingestion we have**
  - many tasks
  - multiple tenants/users
  - ad-hoc, on-demand vs scheduled pipelines
  - local vs cross data centers data movement
- **Complex orchestration techniques are used**

Data sources and sinks



Big data platform examples:

Hadoop File systems
Google Storage
Amazon Storage

Druid
Google BigQuery
Hive
MongoDB
ElasticSearch
Cassandra
InfluxDB
Hudi
Kafka, Pulsar

Diverse requirements from V* of big data

- **Requirements from access API and protocols**
  - REST API, ODBC, SFTP, specific client libs
  - MQTT, AMQP, CoAP, NATS, ...
- **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/masking
  - data storing
  - lineage and observability for quality assurance/governance (quality check)
- **Customer/user defined tasks vs platform tasks**
- **Other supports: compression, end-to-end security**

They are different for batch vs near real time ingestions

Data access and extraction tasks

- **Access**
  - obtaining/copy data from sources,
    - including change data capture (CDC)
  - often built based on common protocols and APIs
    - Rich connectors solutions! (strongly related to data storage/database/datalake sink/source)
  - 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 diverse data structures

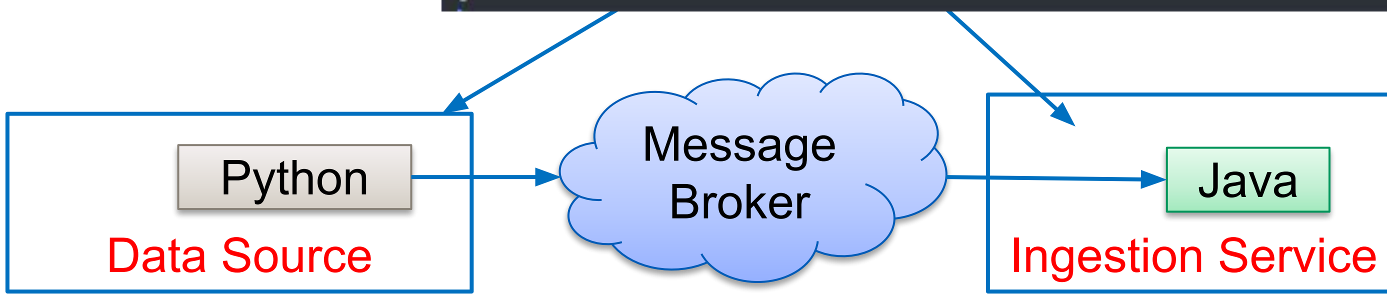
- **Remember that the data sender/producer and the receiver/consumer are diverse**
  - in many cases, they are not in the same organization, implemented with different languages and technologies
    - *need to guarantee the message syntax and semantics*
- **Solutions**
  - don't assume! agreed in advance □ 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

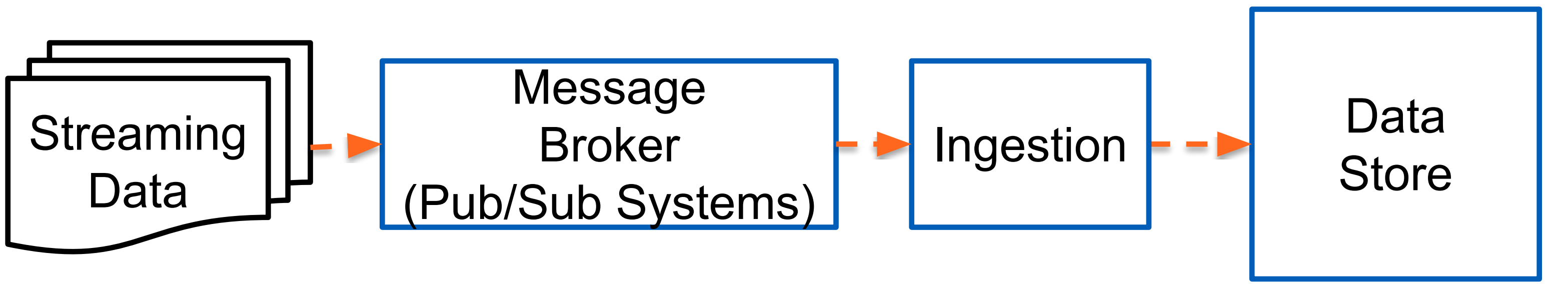
Syntax specification
https://avro.apache.org/

```
{
        "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"}

        ]
}
```

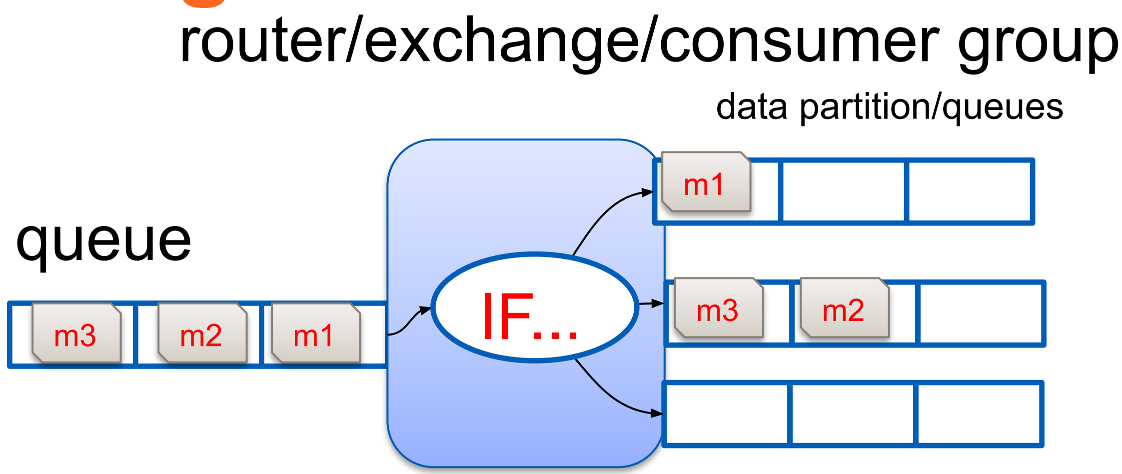


How do we move streaming data into big data databases/storage?



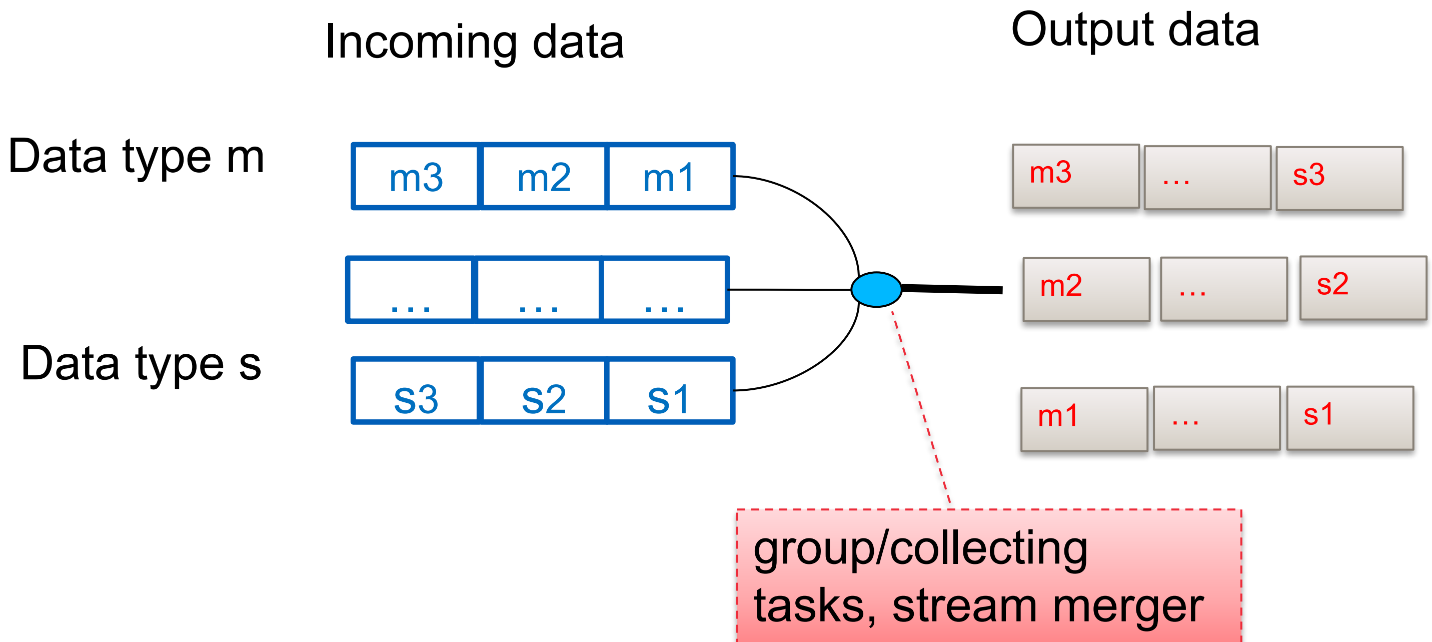
Protocol
Data format
Message structure
Basic streaming data processing techniques

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**
  - cleansing, 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?

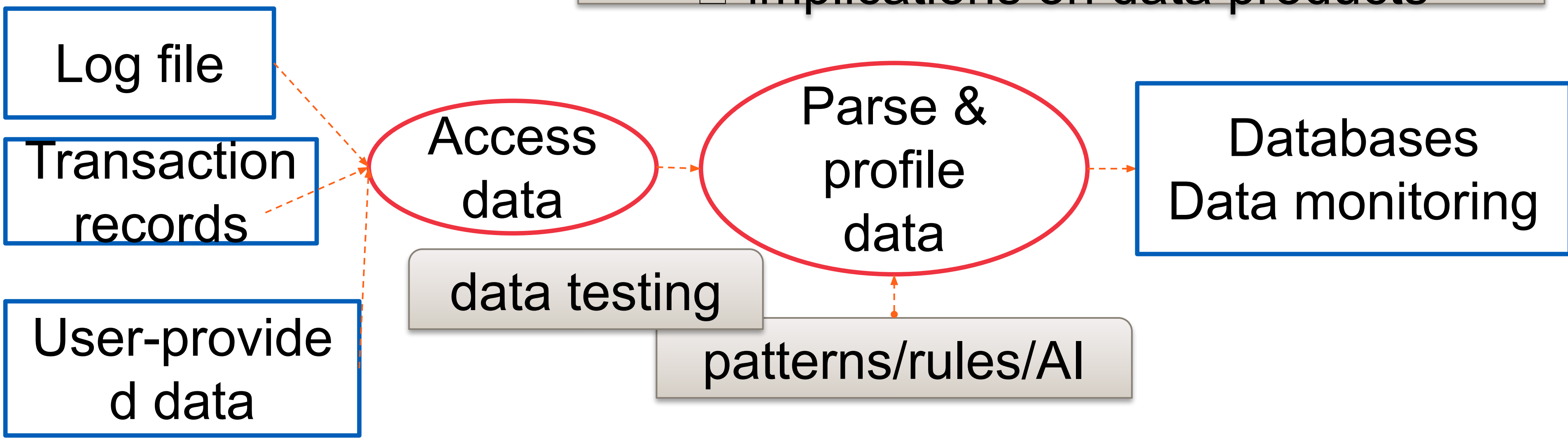
Data wrangling

- **In the context of big data platforms**
  - define or discover data schemas
  - 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 existing patterns:

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

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

Very common in handling logs, alarms, etc.

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/Dask and Dataframe?

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)
```

Behind the scene: complex code & libraries hide low level distributed tasks

- **Code hides complex distributed and parallel tasks for ingestion**
- **Underlying, internal task models:**
  - MapReduce model
  - embarrassingly parallel model
  - full direct acyclic graph (DAG) task model

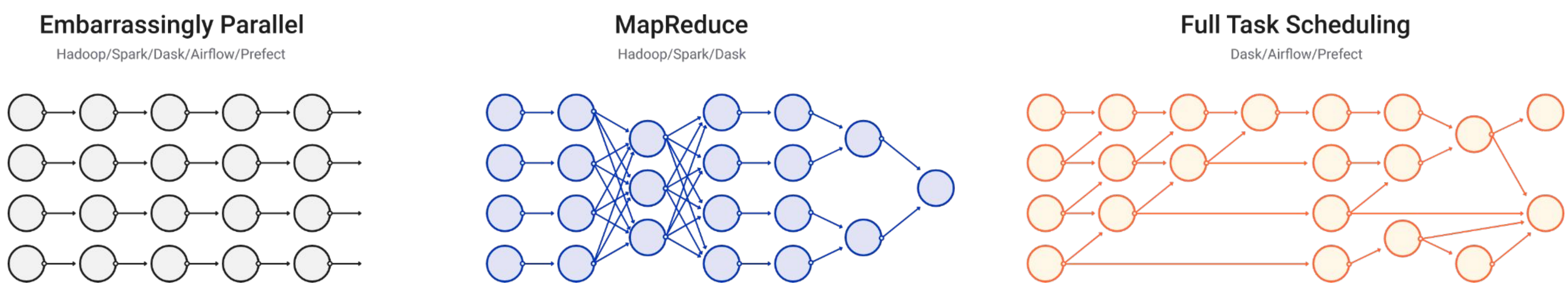
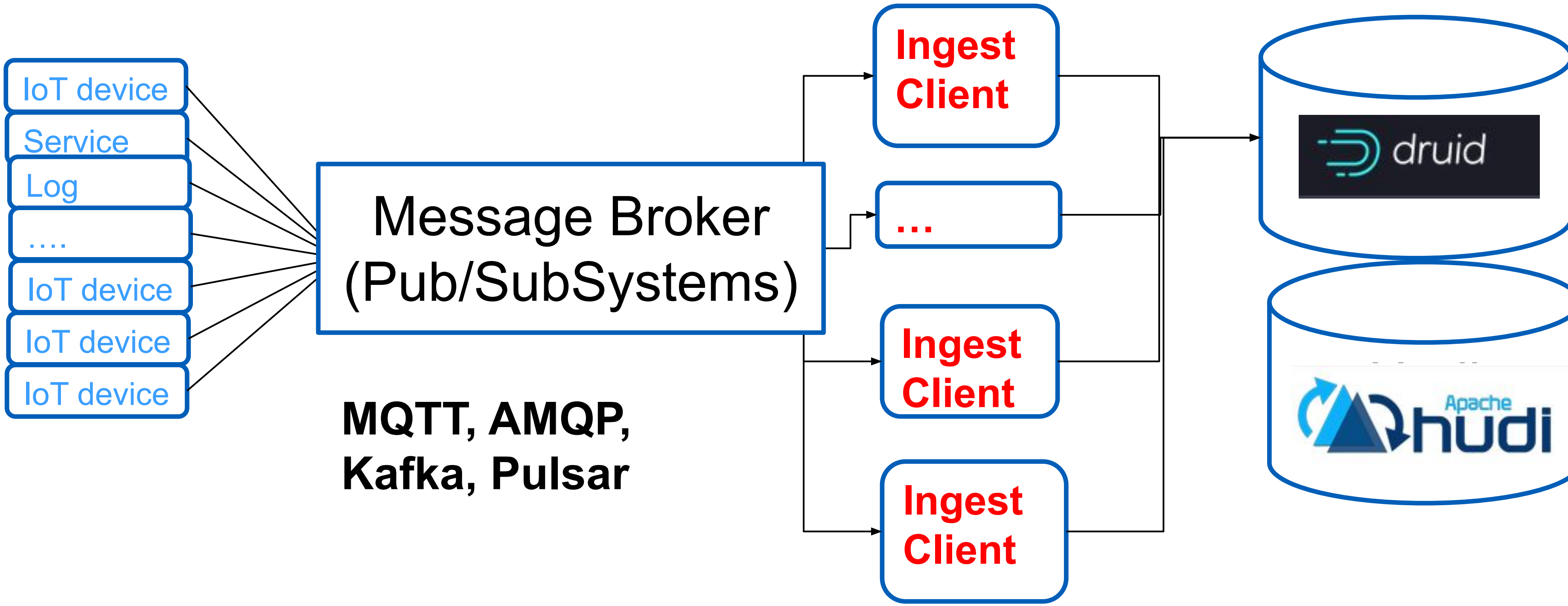
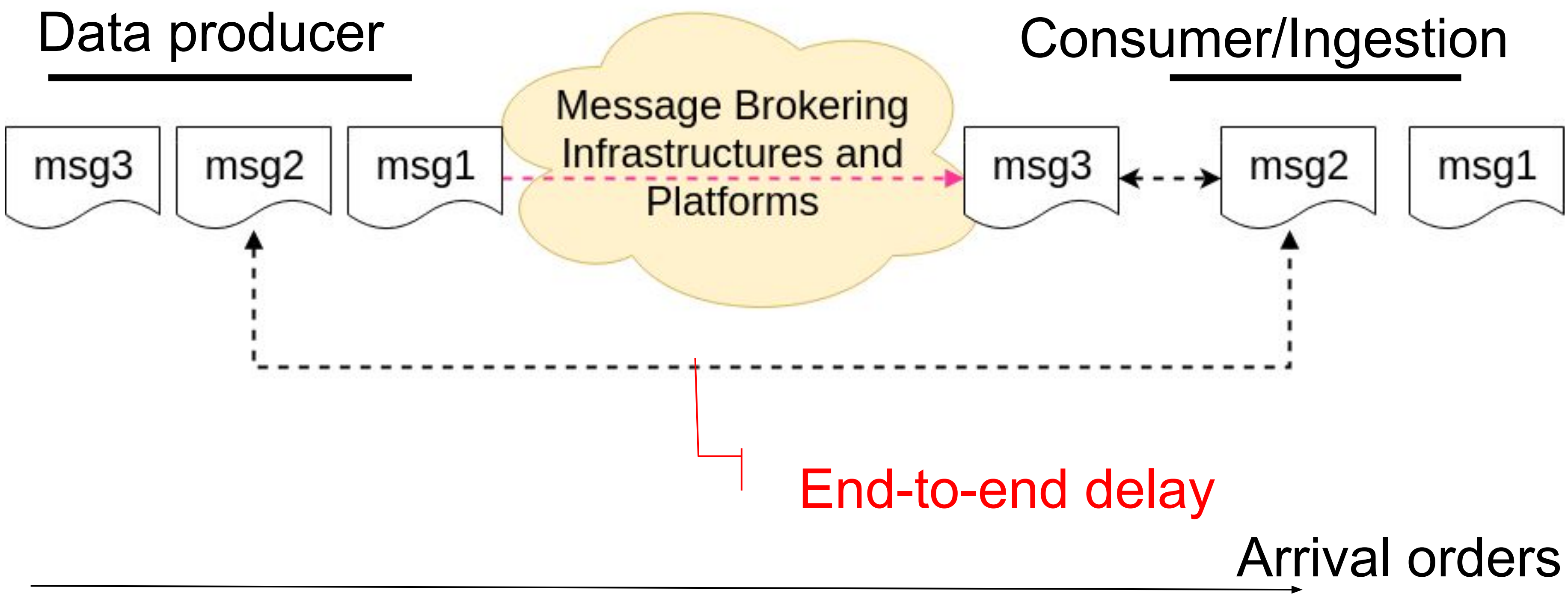


Figure source: https://docs.dask.org/en/stable/graphs.html

Near Real time ingestion



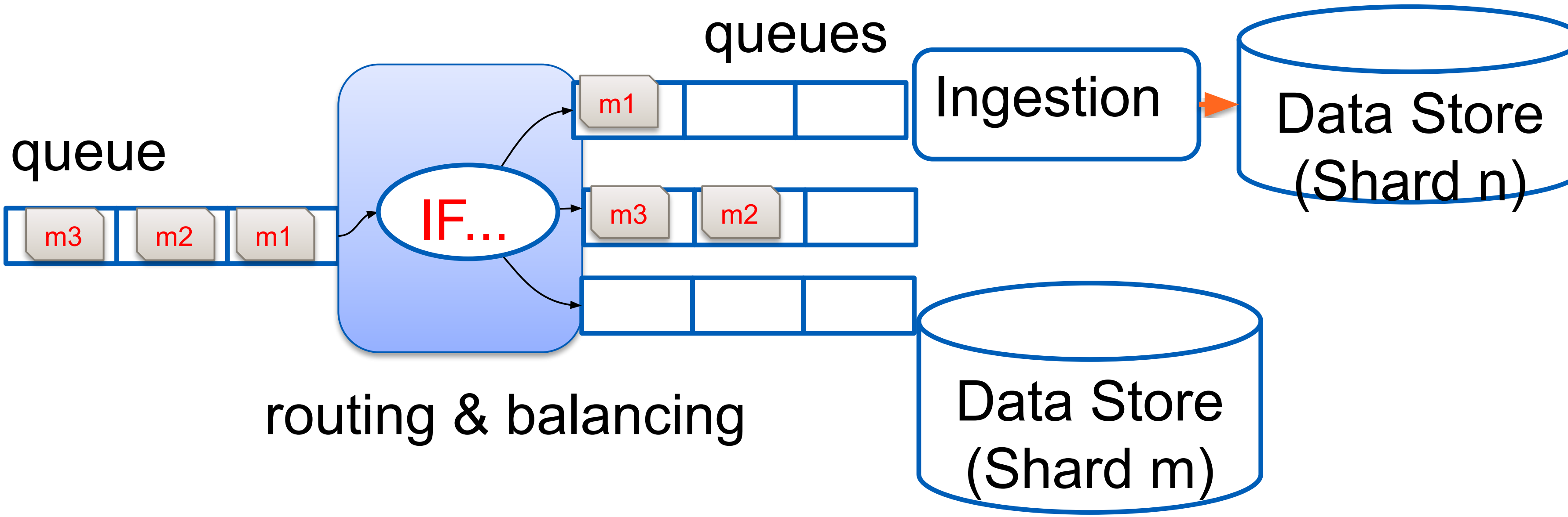
Key issues in streaming data ingestion



Some key issues for ingestion

- **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

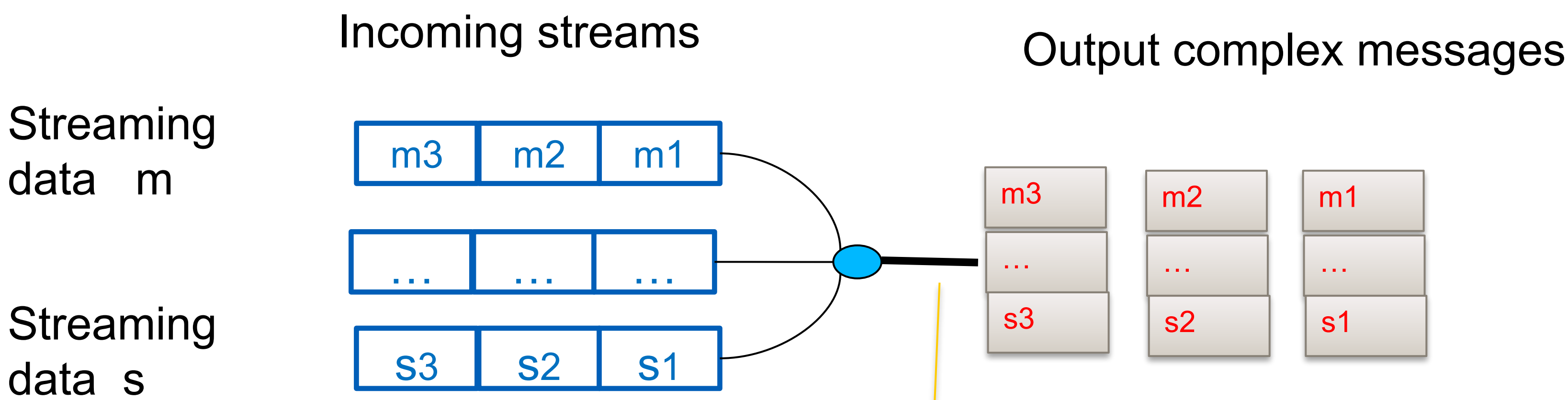
Split (pub/sub) and partition with ingestion



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?**

Procesing data before ingestion requires some streaming techniques



E.g., for data rollup/summarization

Ingestion tasks implemented as extensible, composable connectors

- **Basic tasks for big data ingestion can be used in different cases**
- **Support end-user goals**
  - 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**

Aalto University
School of Science

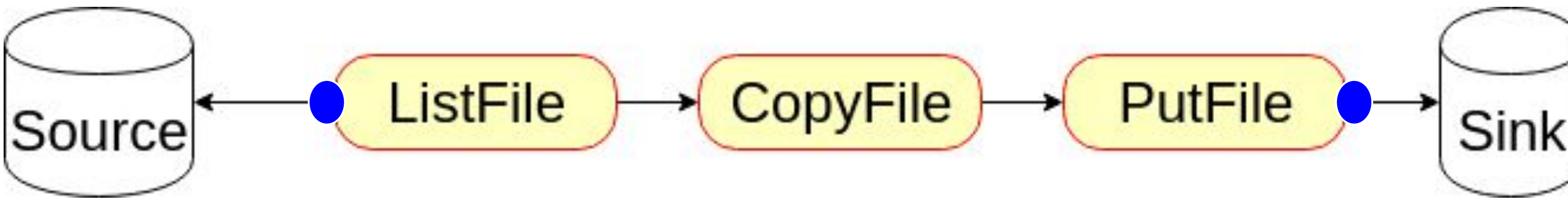
Pipeline designs and execution models

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**

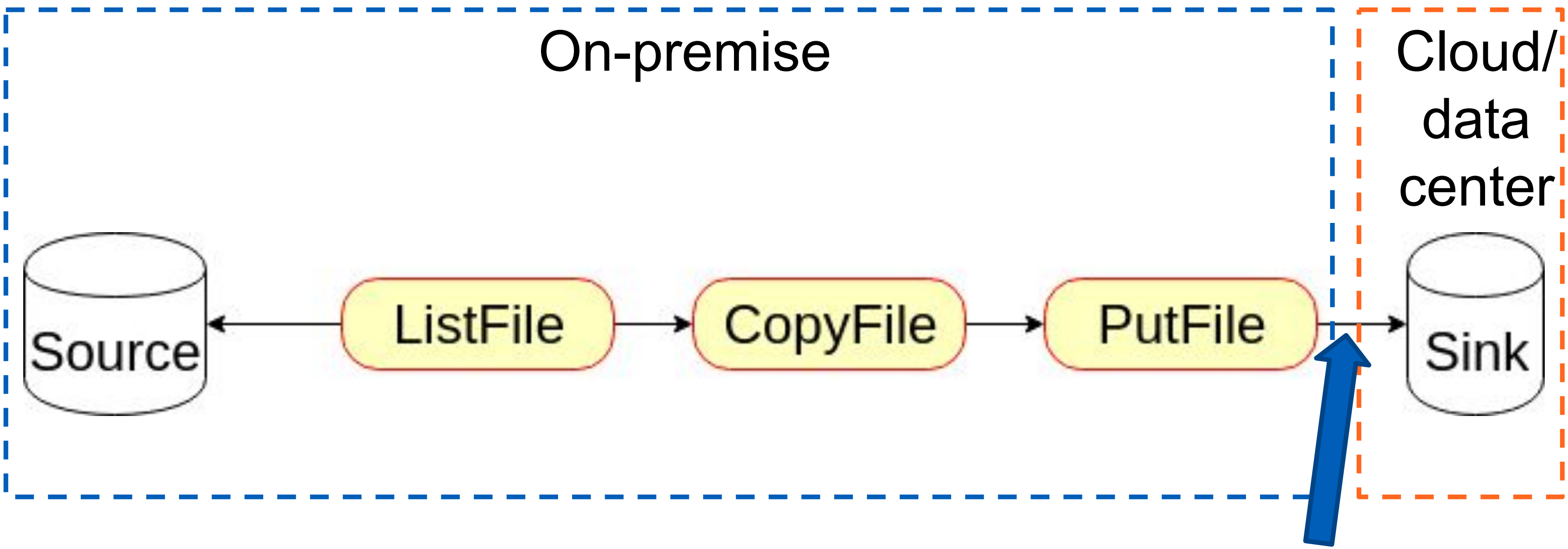


Tenant/ user

Ingestion pipeline developer (for whom?)

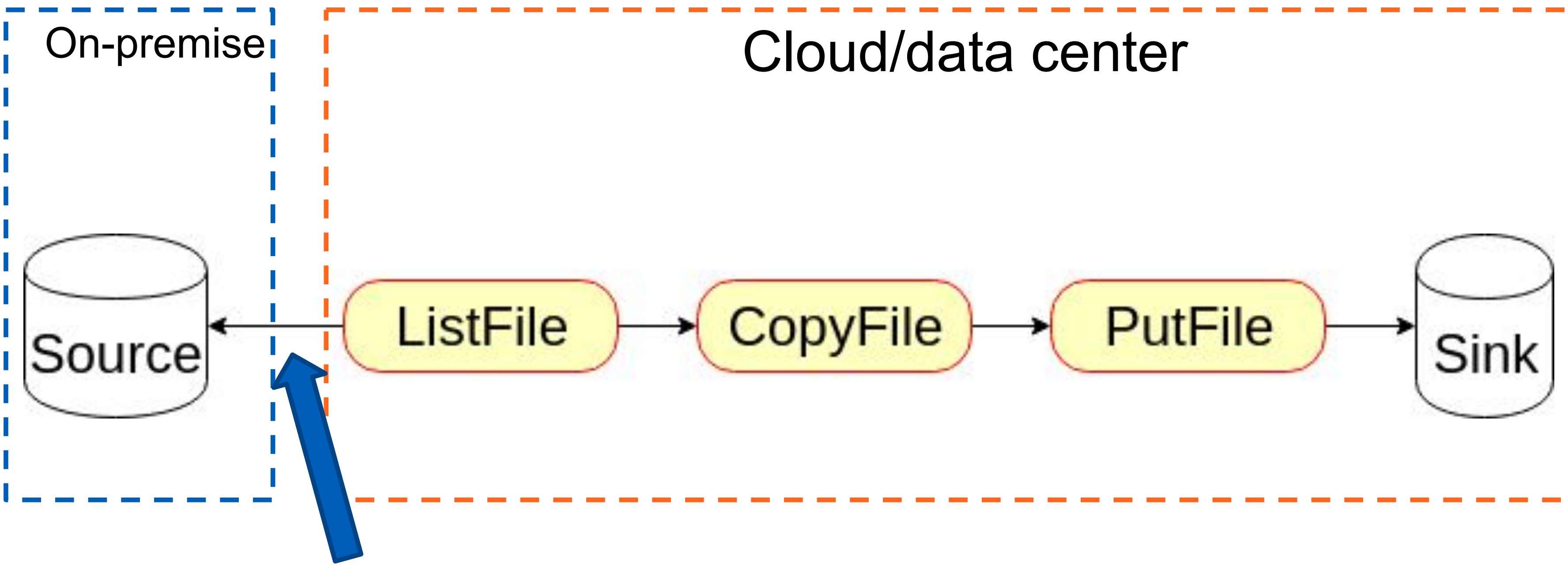
Data store/platform provider

Complex deployment and composition models



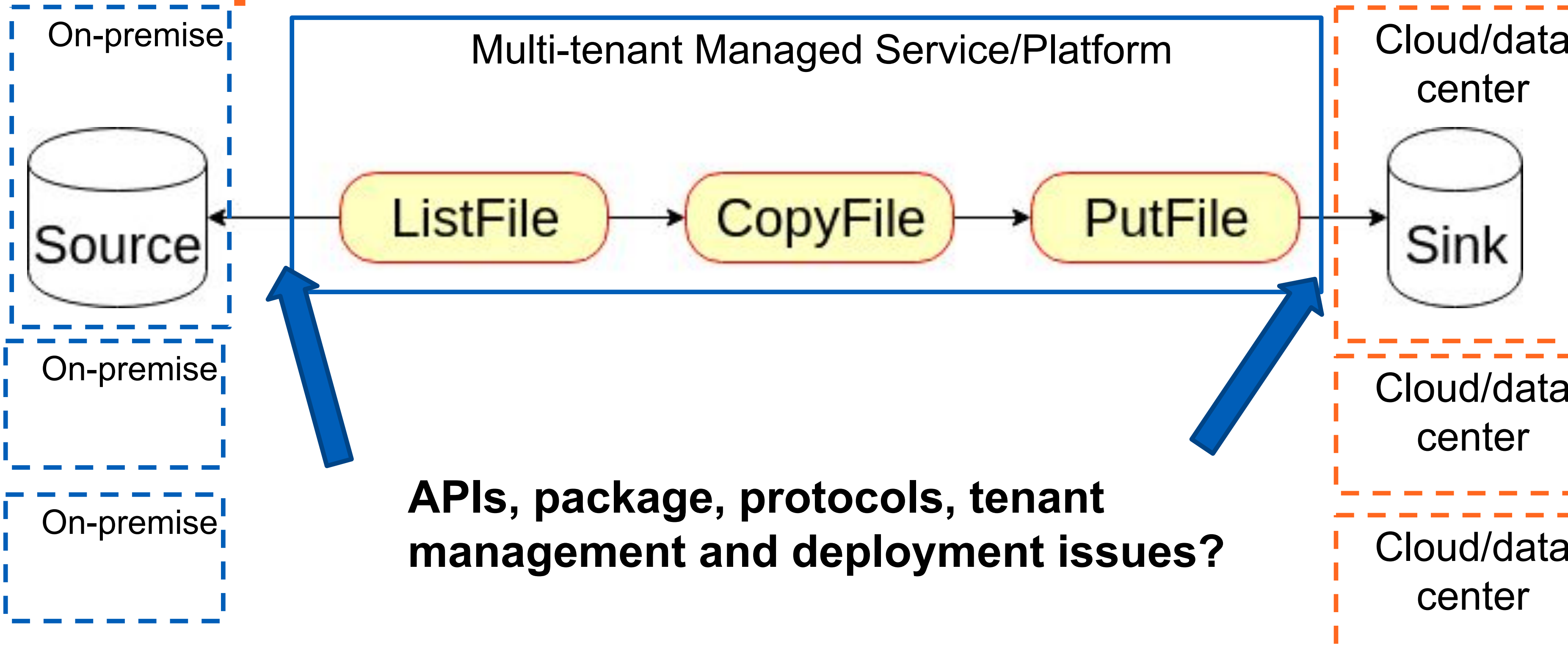
APIs, protocols and deployment issues?

Complex deployment and composition models



APIs, protocols and deployment issues?

Complex deployment and composition models



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

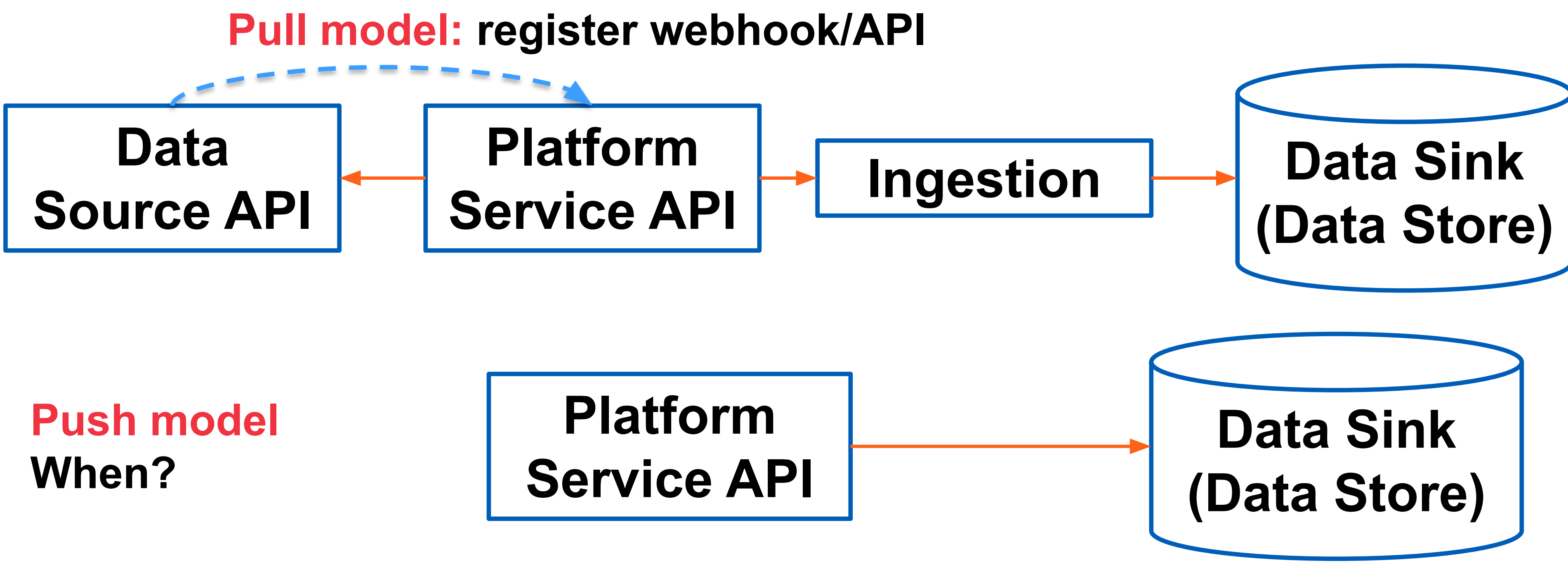
Architecture requirements

- **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**

Orchestrating batch ingestion processes

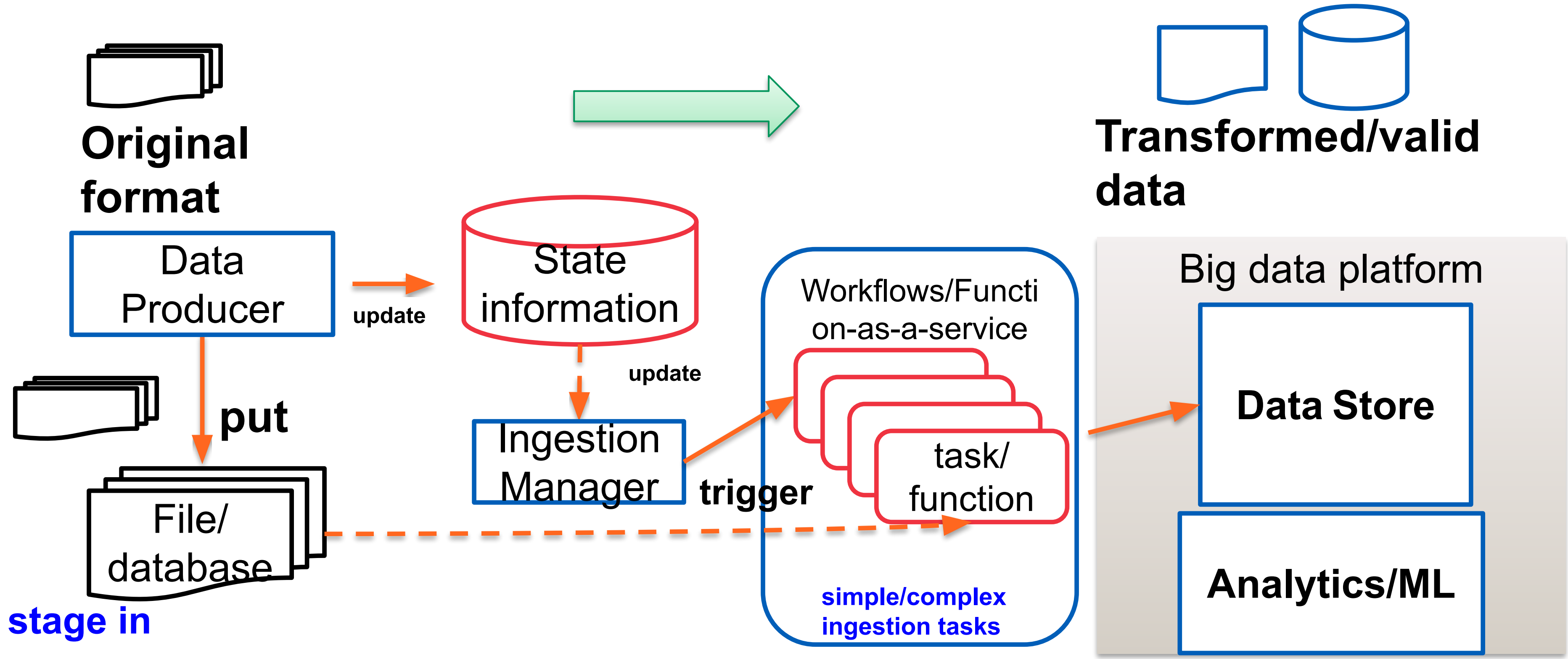
- **Data to be ingested is bounded**
  - files or messages are finite
- **Ingestion architectural styles**
  - (1) Direct APIs, (2) reactive pipelines, (3) workflows
- **Incremental ingestion**
  - dealing with the same data source but the data in the source has been changed over the time (related to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

Simple, direct APIs for ingestion



Try to analyze pros and cons for your platform?

With functions/workflows/containers

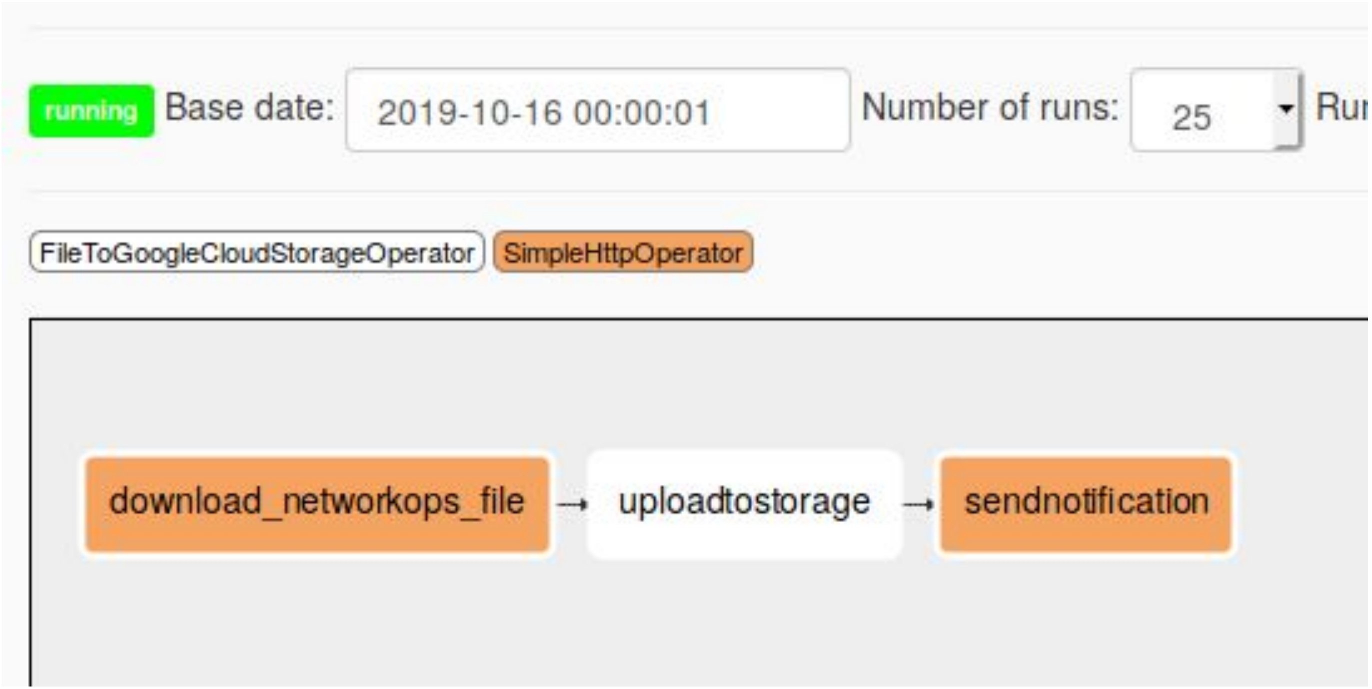


Who develops which components?

Orchestrating ingestion workflow

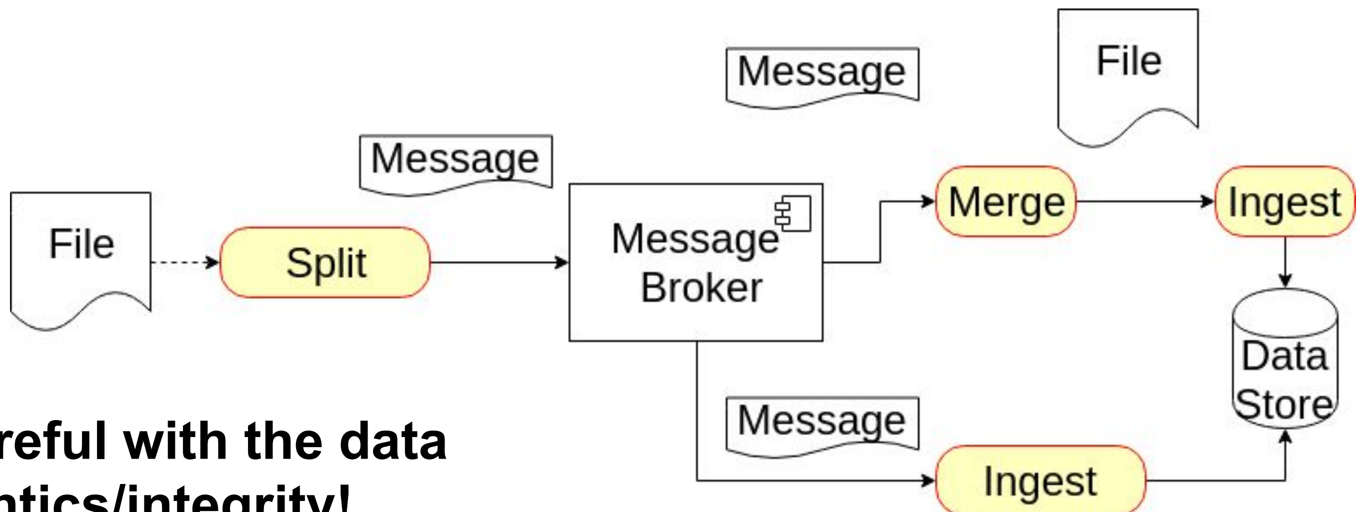
- **Different tasks for**
  - access and copy, extract, covert, quality check, and write data
  - tasks can be connected based on data or control flows
- **Workflows**
  - a set of connected tasks is executed by an engine
  - tasks can be scheduled and executed in different places
- **Bulk ingestion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

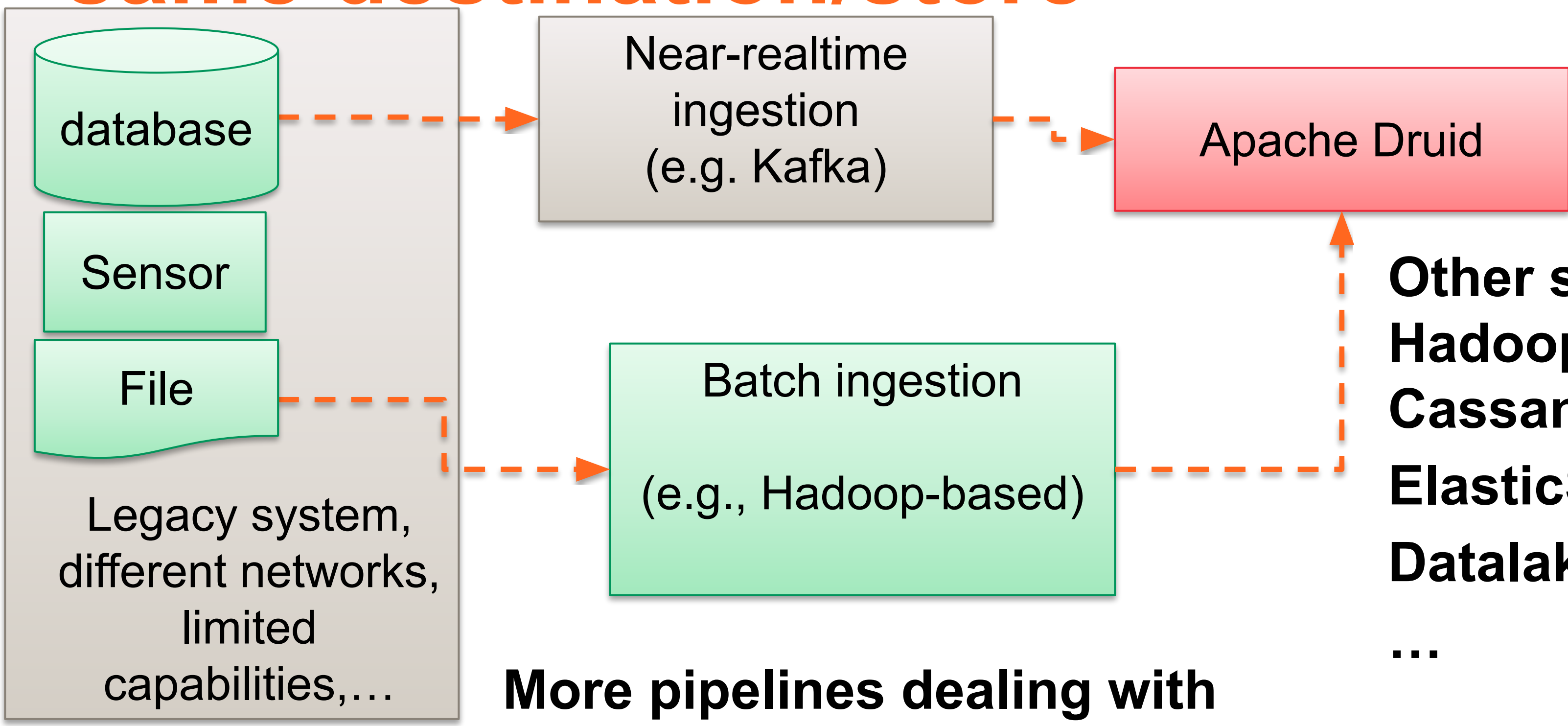
Complex ingestion pipelines in big data platforms

- **Multiple types of pipelines for multiple types of tenants/users**
- **A tenant/user 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

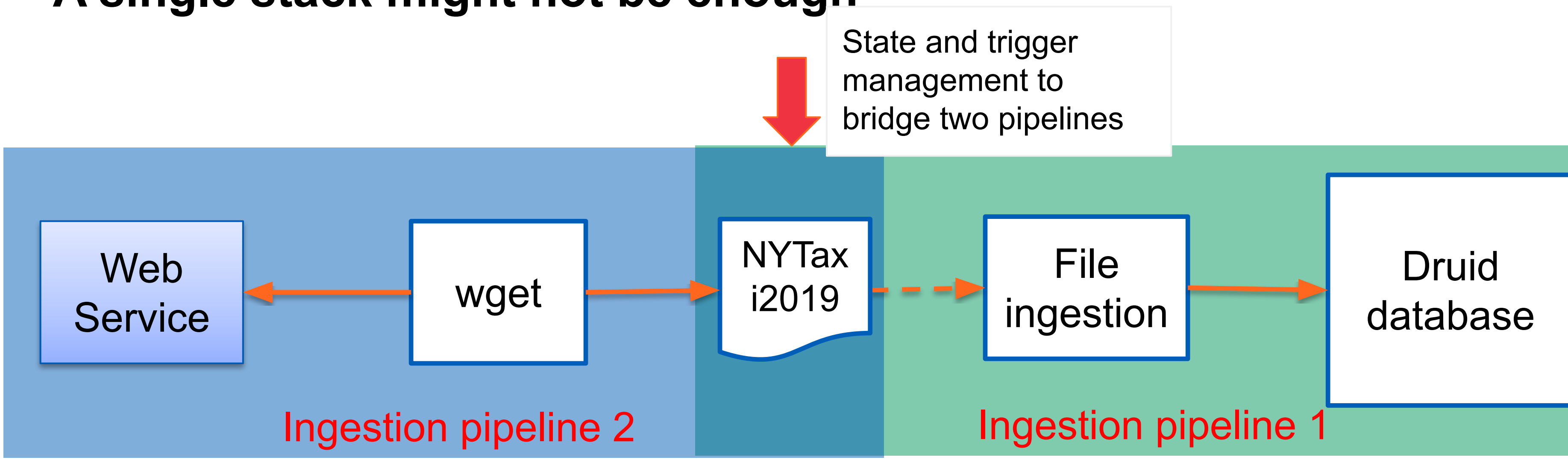


Other systems: Hadoop, Cassandra ElasticSearch Datalake …

More pipelines dealing with different sources

Connecting different ingestion pipelines

A single stack might not be enough

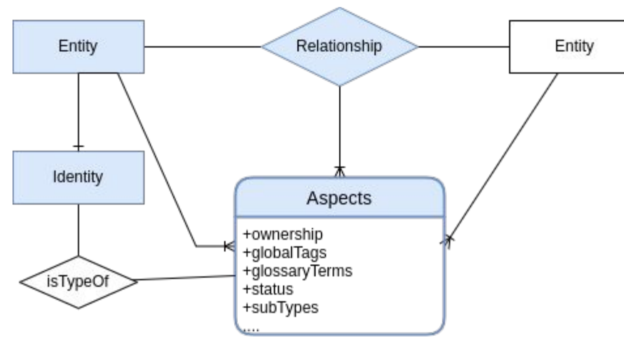


Real-world:

both pipelines and their connection are complex

Data Lineage and Observability

- **FAIR principles** (https://www.nature.com/articles/sdata201618)
  - findable, accessible, interoperable, and reusable
- **Lineage/Provenance**
  - capture relevant information for understanding how data has been moved, transferred, processed, etc.
  - metadata models: W3C Provenance Model, DataHub, etc.
- **Key issues**
  - which metadata must be captured?
  - based on existing tools or your own?
- **Instrumentation/logging processes and automated data lineage → performance overhead!**



High level view of datahub see https://datahubproject.io/docs/metadata-modeling/metadata-model

Lineage and Observability

- **Observability: the health about data**
  - near-real time metrics, offline checks and possible dashboards
  - similar to service observability, relying on traces, logs, metrics, etc.
- **Focus on data**
  - data metrics (volumes, data quality, schemas, lineage)
  - issues due to data problems
  - data ingestion processes/workflows
- **Some solutions**
  - validation of data against design schemas (e.g., Schema Registry in Kafka)
  - checks of realtime and offline data quality → integrate with data ingestion processes or batch analytics for data quality (e.g., offline data profiling)
  - integrated data quality tests in pipelines (e.g., data testing)

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

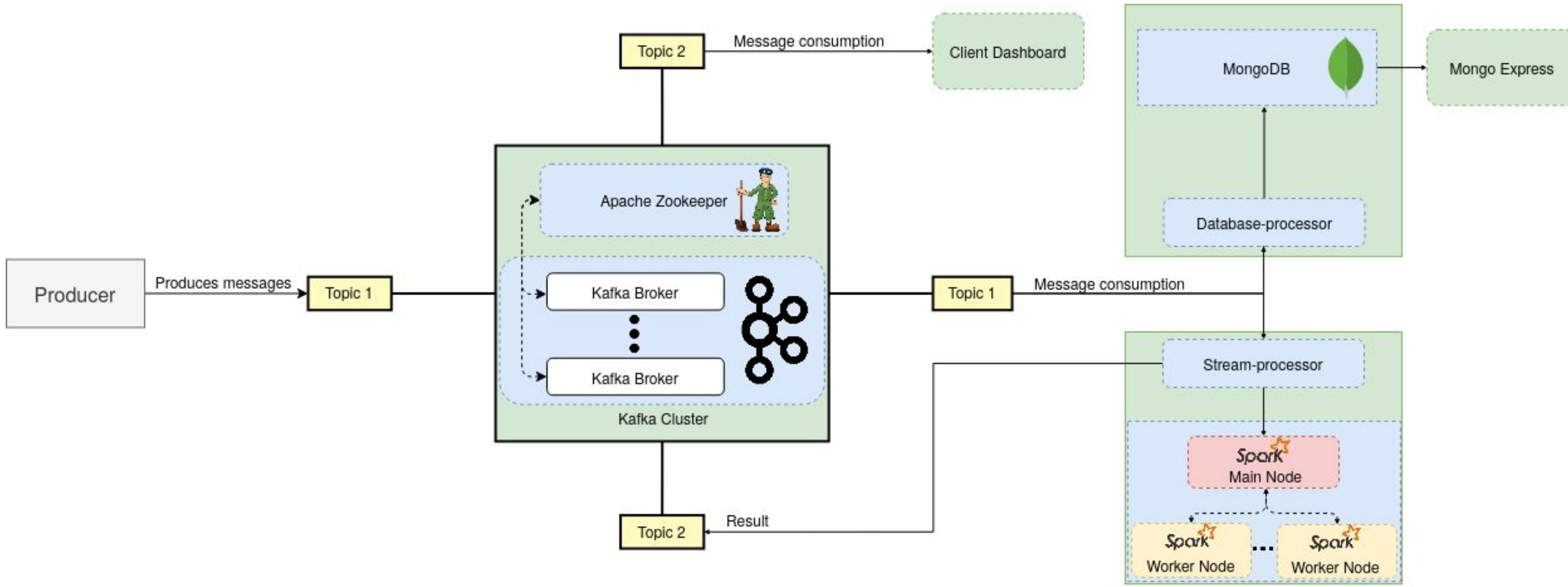
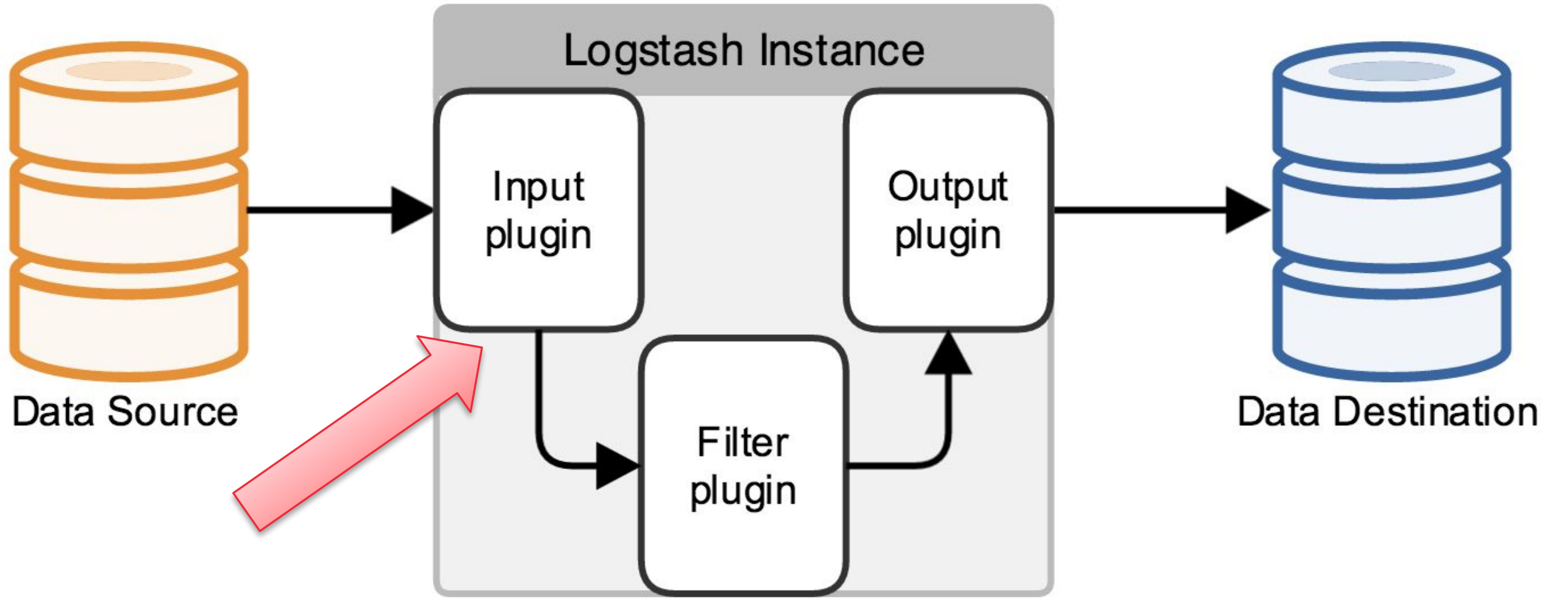


Figure source:
https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/cloud-data-pipeline

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

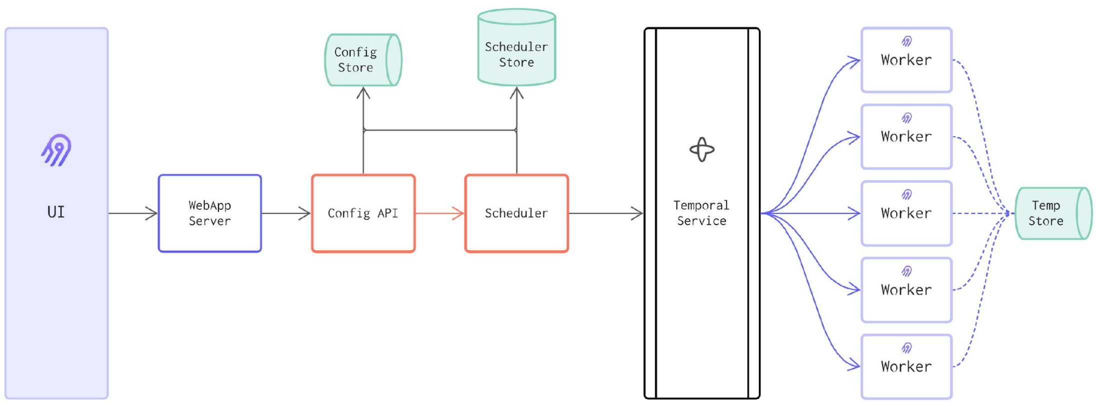


Pluggable approaches

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



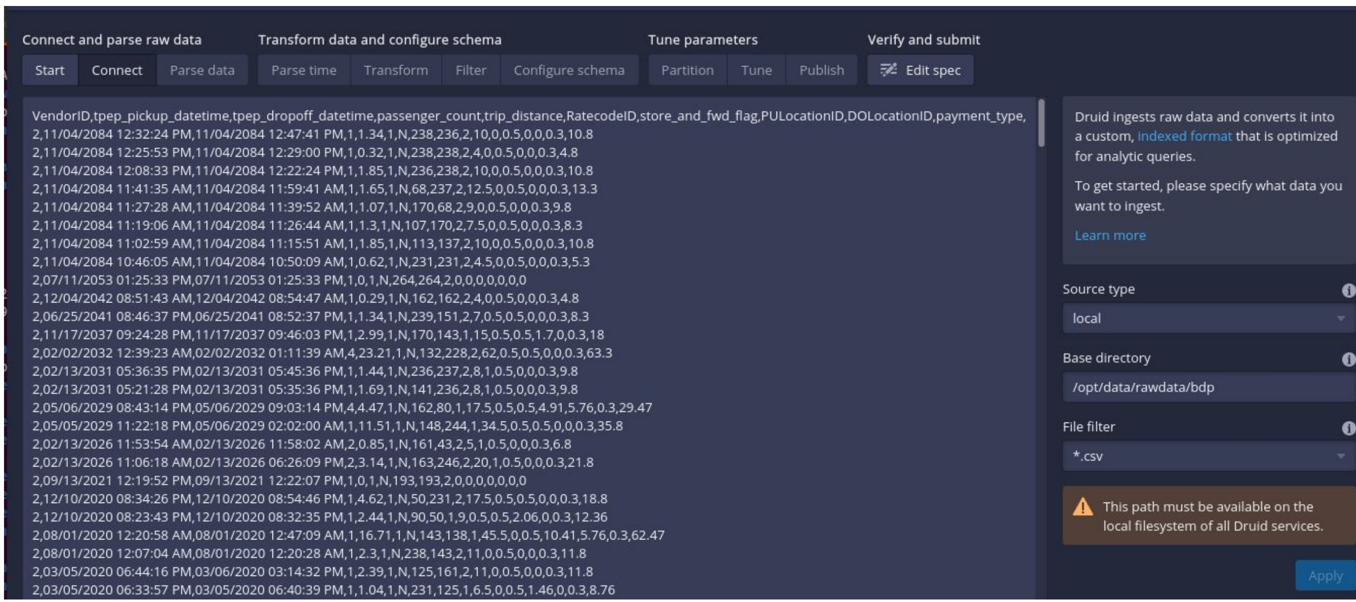
Connectors for multiple data sources

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

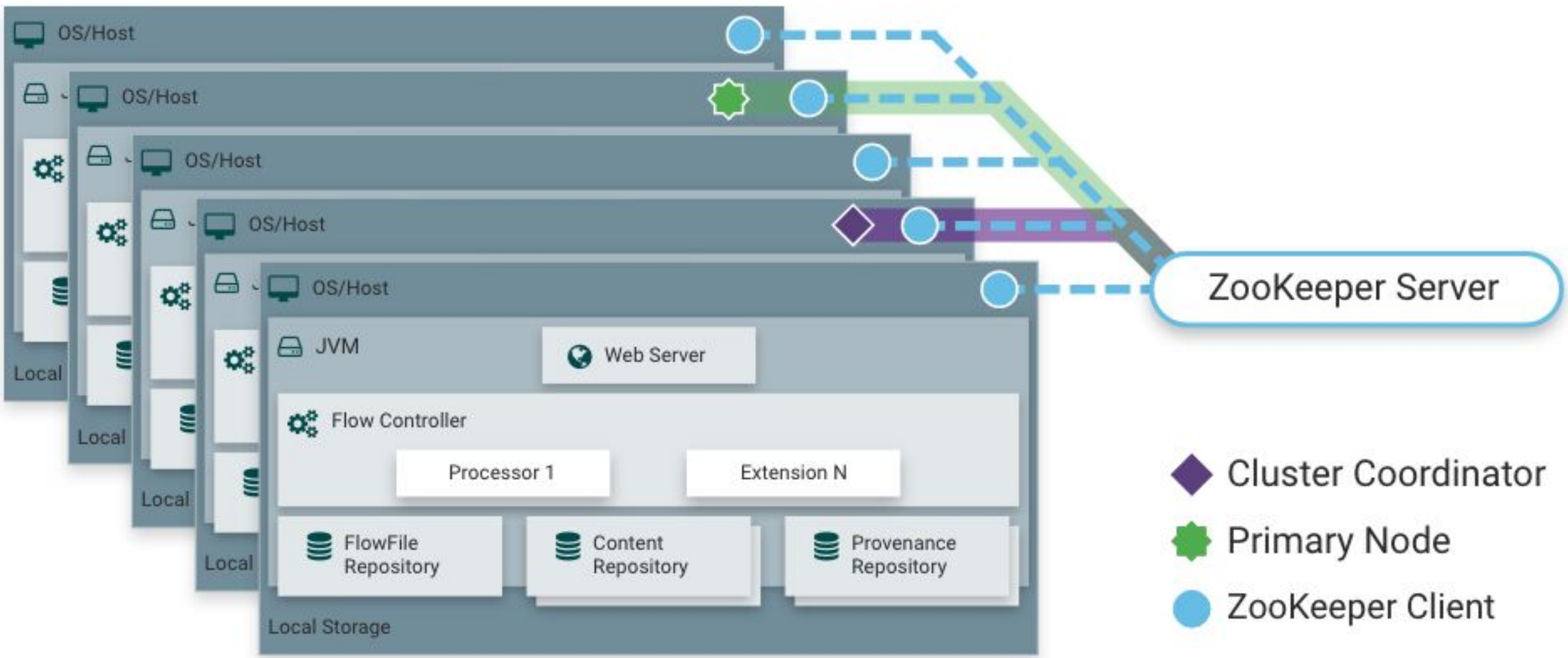
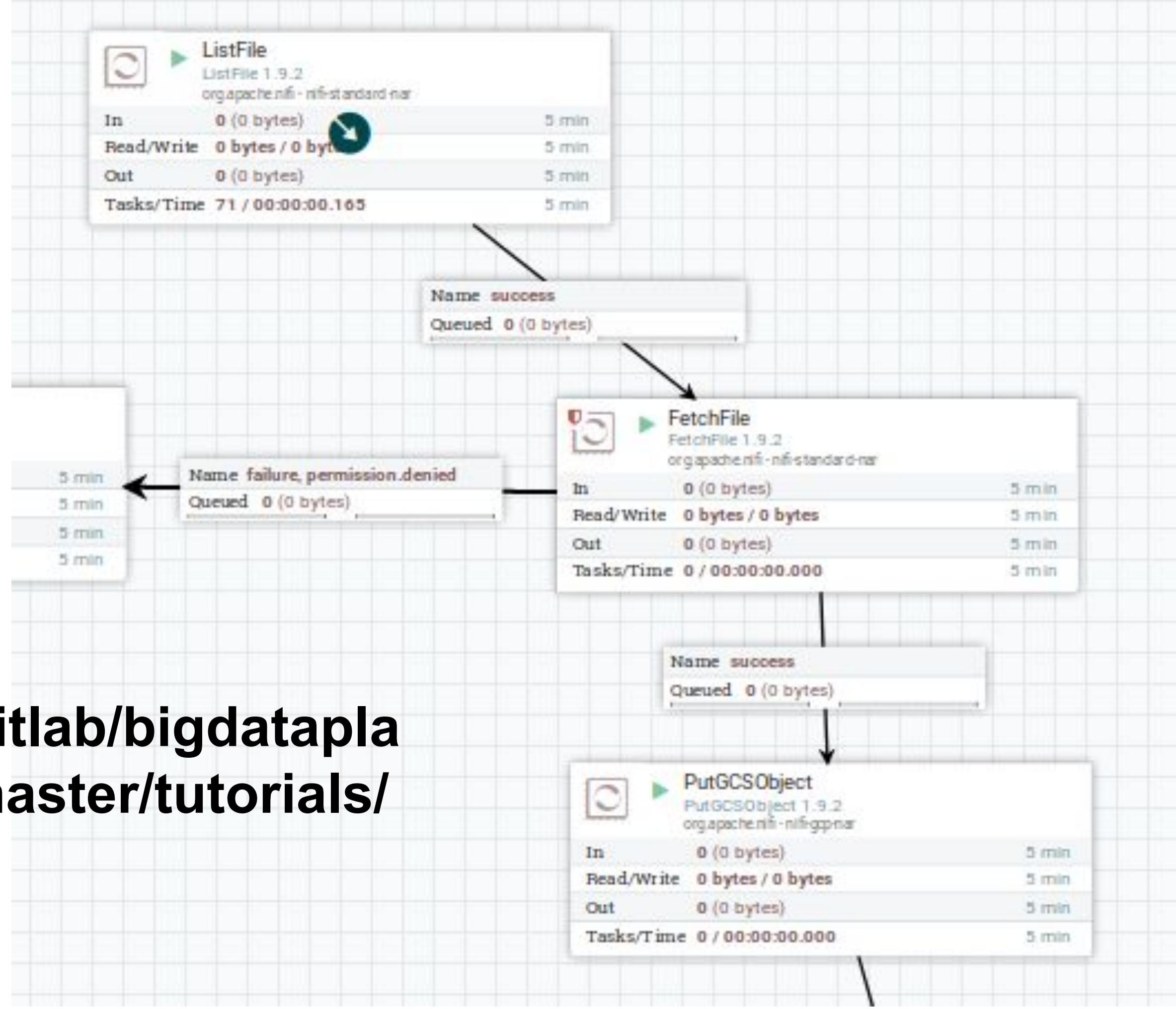


Figure source: https://nifi.apache.org/docs.html

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

Other related tools/stacks

- **Data Form**
  - https://dataform.co
- **DBT**
  - https://www.getdbt.com/
- **Key aspects**
  - data engineering and governance tasks centered around SQL-style
  - data transformation and ingestion code managed similar to software code
    - *templates, configuration files, variables, CLI supports*
    - *git style, documentation for code management*
  - deployment using other workflows/resource scheduling systems

Summary

- **Different designs of data ingestion for batch and streaming**
- **Ingestion is a complex pipeline**
  - many different sub tasks
  - complex requirements w.r.t performance, scale, failure handling, etc.
- **Different tools/stacks/services available**
  - share composable design principles, but different software models and deployments
  - explore them for your work
- **Do real-world designs**
  - hands-ons
  - complex designs but we do not need to "reinvent the wheel" → stay with core concepts and requirements to find the right tools first!

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io

All kind of data ingestion with 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/)

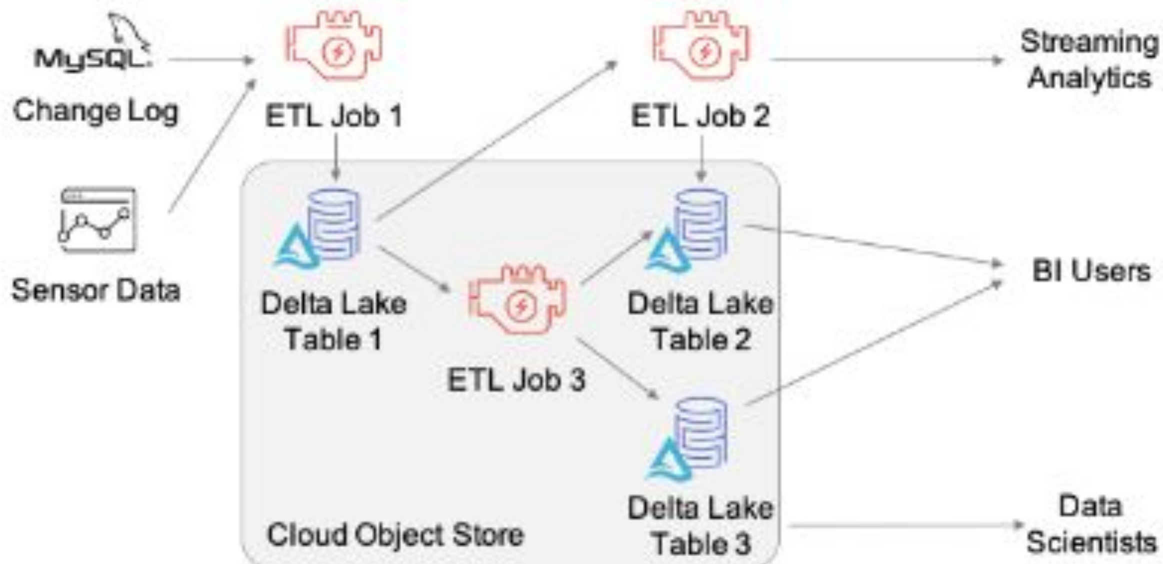


Figure source: "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores", https://databricks.com/wp-content/uploads/2020/08/p975-armbrust.pdf