Aalto University
School of Science

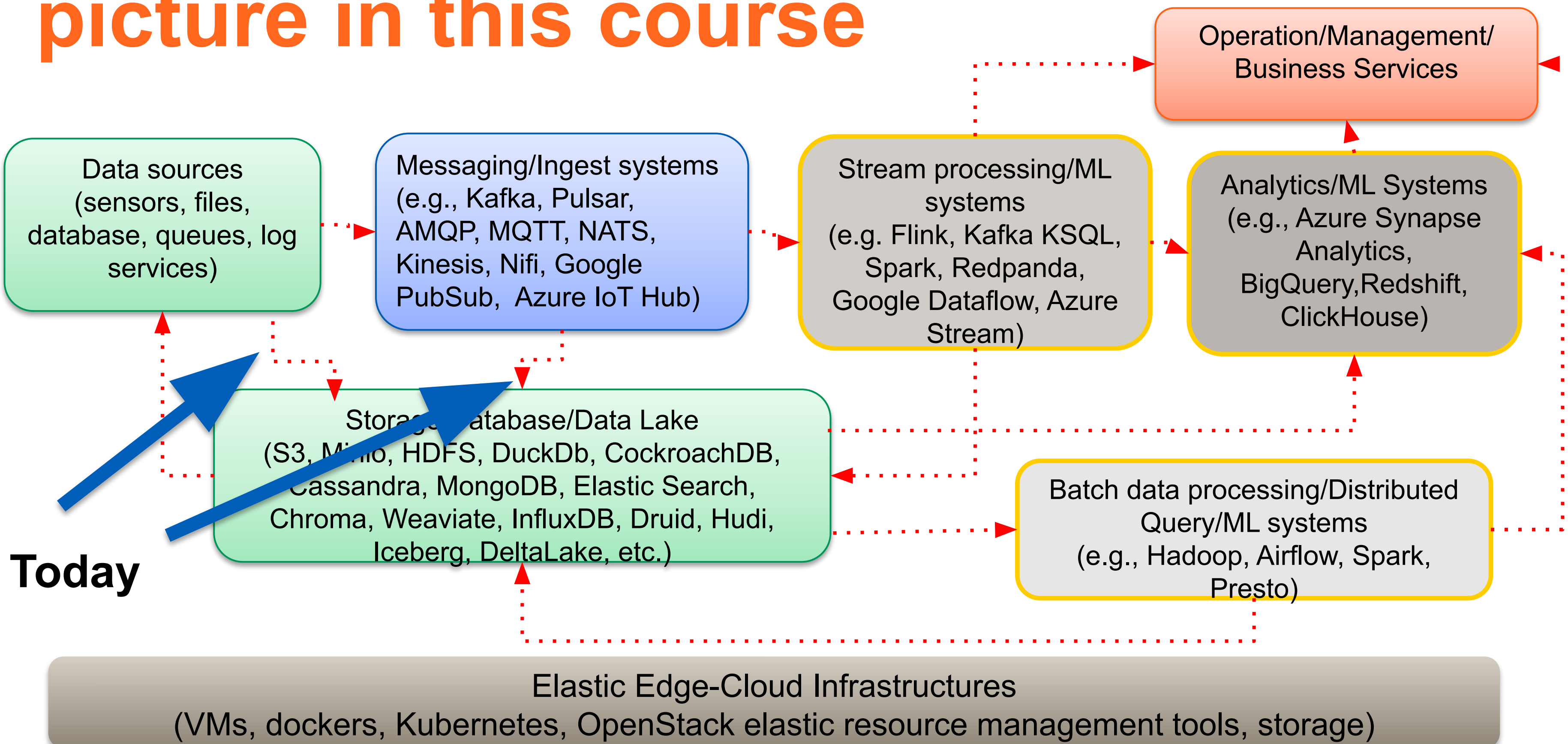
Big Data Ingestion, Transformation and Orchestration

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

Learning objectives

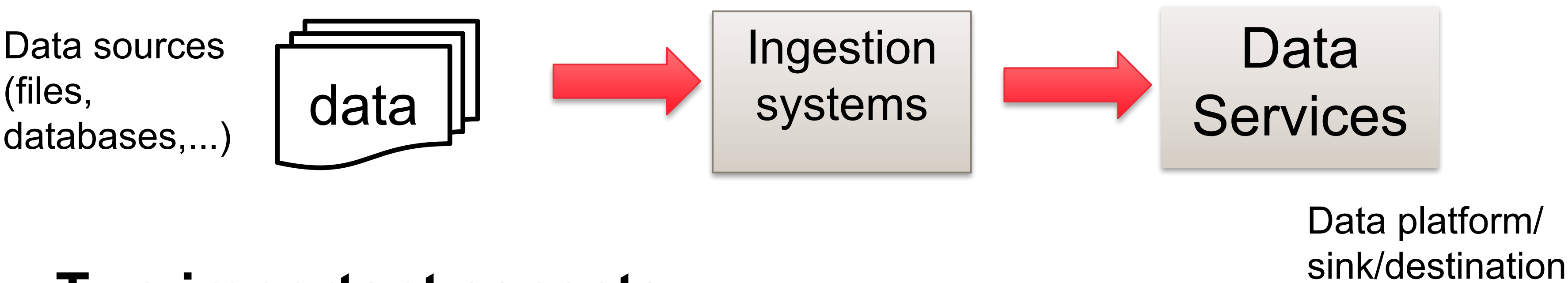
- **Understand the overall design of data ingestion**
- **Study common tasks in data ingestion**
- **Understand and design efficient, robust data ingestion pipelines/processes**
- **Learn existing technologies/frameworks for your own design**

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



Ingestion systems

Data ingestion: move data from <u>different sources</u> into data platforms or selected data sinks/destinations

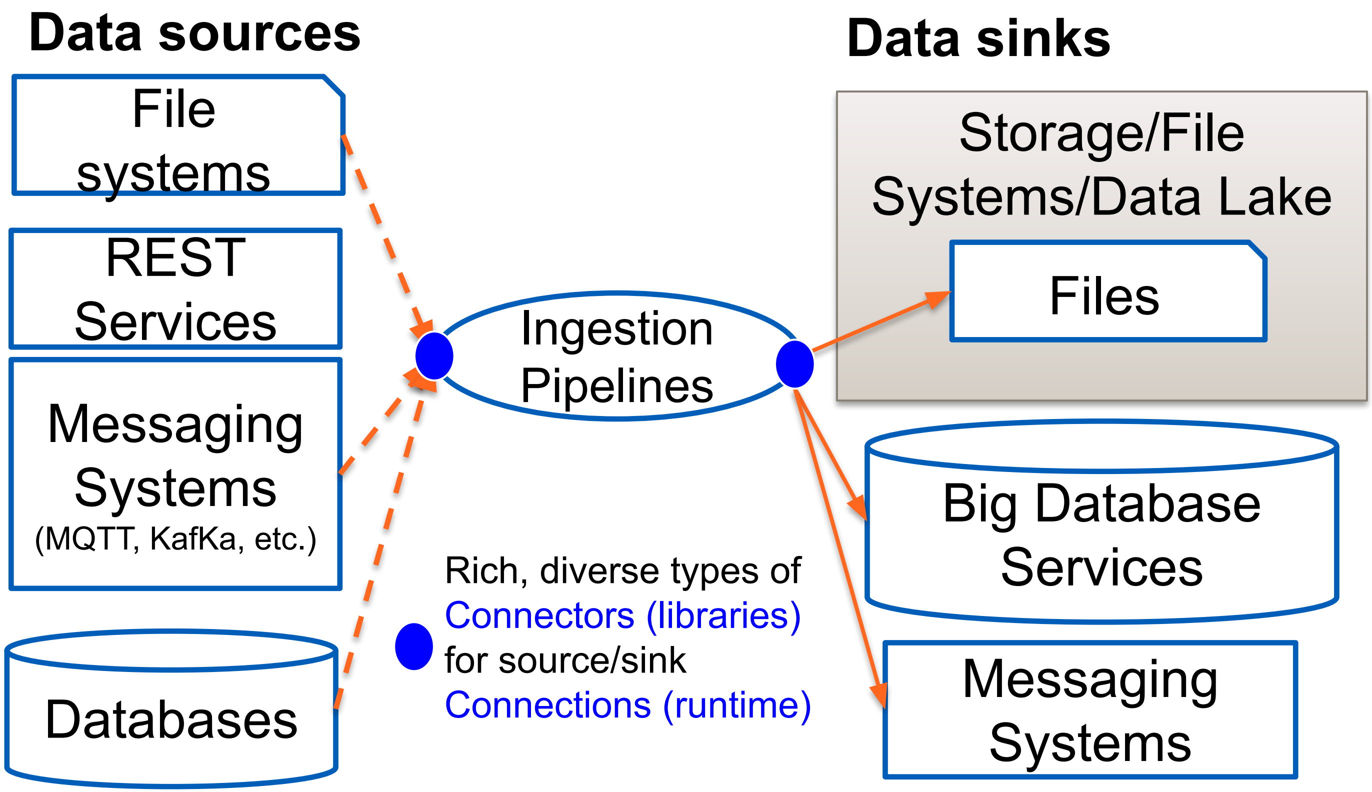


Two important aspects:

- tasks and non-functional requirements
- architectures, pipelines and service models

Reusability and extensibility

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

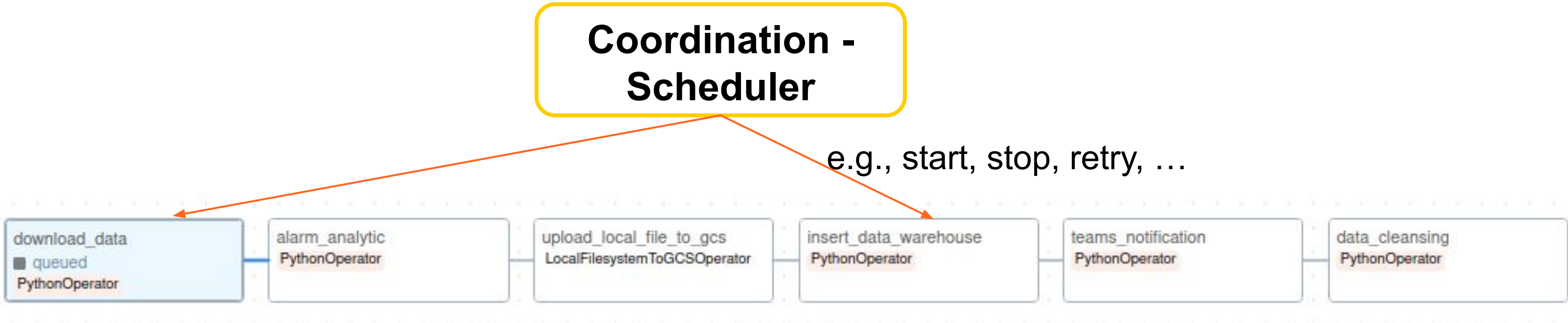
- **Requirements based on data characteristics**
  - structured, unstructured and semi-structured
  - speed, volume, accuracy, confidentiality, data regulation
- **Interact with data sources:**
  - Access APIs and protocols
    - *REST API, ODBC, SFTP, specific client libs*
    - *MQTT, AMQP, CoAP, NATS, Kafka,...*
  - Connection management:
    - *Performance, reliability and security*
- **How deep can a platform support complex requirements?**
  - e.g., able to go into inside of data elements (understanding the syntax and semantics of data)?

Data transformation

- **Relation with ETL (Extract, Transform, Load)**
  - during ingestion, data transformation tasks might be needed
  - ETL has many operations to deal with the semantics/syntax of data and the business of data
- **Data transformation within ingestion**
- **Data transformation done after, within the (target) platform**
  - called ELT (load and then transformation)

Performance, correctness and quality assurance

Ingestion needs task coordination



- **Big data ingestion involves**
  - many tasks
  - multiple tenants/users
  - ad-hoc, on-demand vs scheduled task pipelines
  - data movement in single vs cross data centers
- **Complex coordination techniques are used**
  - tasks are not in the same machine (executors) → data exchange among tasks (data dependencies)

Ingestion tasks: common tasks and requirements

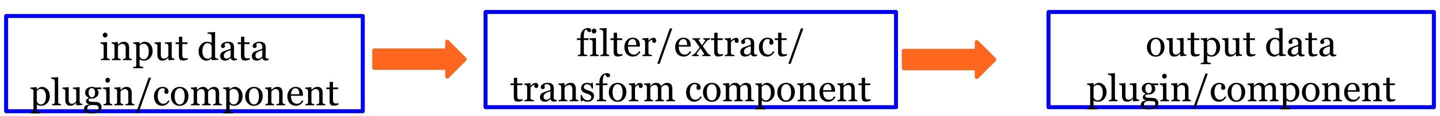
Main tasks in ingestion

- **Key categories of tasks**
  - data access and extraction
  - data routing
  - data wrangling
  - data storing
  - lineage and observability for quality assurance/governance (quality check)
- **Consumer/user defined tasks vs platform tasks**
- **Other supports: compression, end-to-end security**

They are different for batch vs near real time ingestions

Ingestion tasks implemented as extensible, composable connectors

- **Basic tasks for big data ingestion can be (re)used in different cases**
- **Support end-user goals**
  - enables the user to do many tasks through configurations and extensions
- **Enable pluggable approaches is important**



Data access and extraction tasks

- **Access**
  - obtaining/copy data from sources
    - including change data capture (CDC)
  - often built based on common protocols and APIs
    - connector library: strongly related to data storage/database/datalake sink/source)
    - runtime connection management: maintain list of connections created from connectors
  - 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

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"
}
}
```

Change data capture (CDC)

- **The principles:**
  - capture and ingest only new data by listening data changes
    - "new": application-specific, e.g., based on time, value, and version.
  - leverage many features of databases (update, query, insert operations), data stream offsets and status notification (e.g., the availability of new files)
- **Implementation in different tools like Redhat Debezium, Hudi DeltaStreamer, Kafka Connect**

Data wrangling

- **Convert/transform data from one form to another**
  - cleansing, filtering, merging, enriching, infering, and reshaping data
- **Require access to the data content!**
- **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

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)
```

Data Wrangling: complex data transformation and processing

- **More complex data processing**
  - extract only important data
    - *feature engineering*
  - enrich data on the fly with external sources

Example: extract vectors from images

```
class TowheeExtractor(BaseExtractor):
    def __init__(self):
        self.towhee_feature_extractor = (
            pipe.input('file_path')
                .map('file_path', 'img', ops.image_decode.cv2_rgb())
                .map('img', 'embedding', ops.image_embedding.timm(model_name='resnet50'))
                .map('embedding', 'embedding', ops.towhee.np_normalize())
                .output('embedding')
            )

    def get_model_name(self):
        return "resnet50"

    def feature_extractor(self, image_file):
        ##currently only a single figure so we have to get the first element
        embedding=self.towhee_feature_extractor(image_file).get()[0]
        return embedding
```

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

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

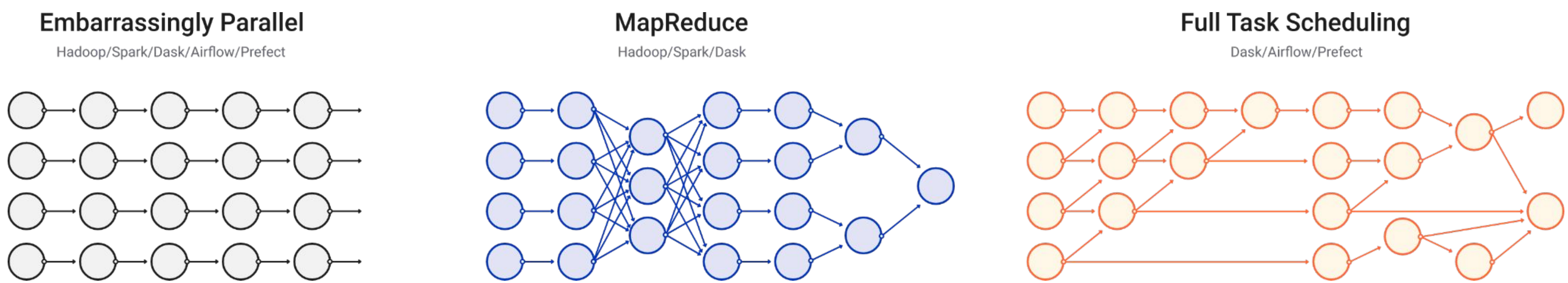
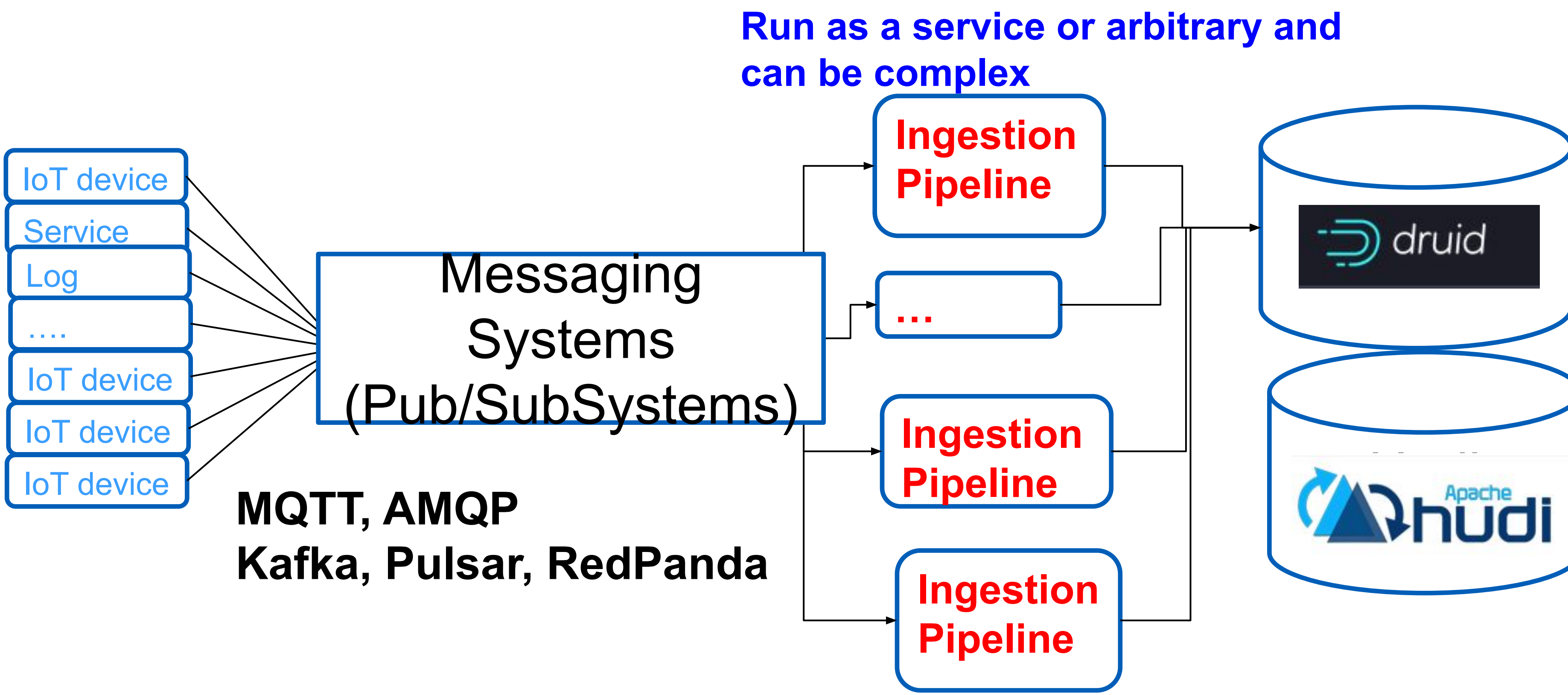
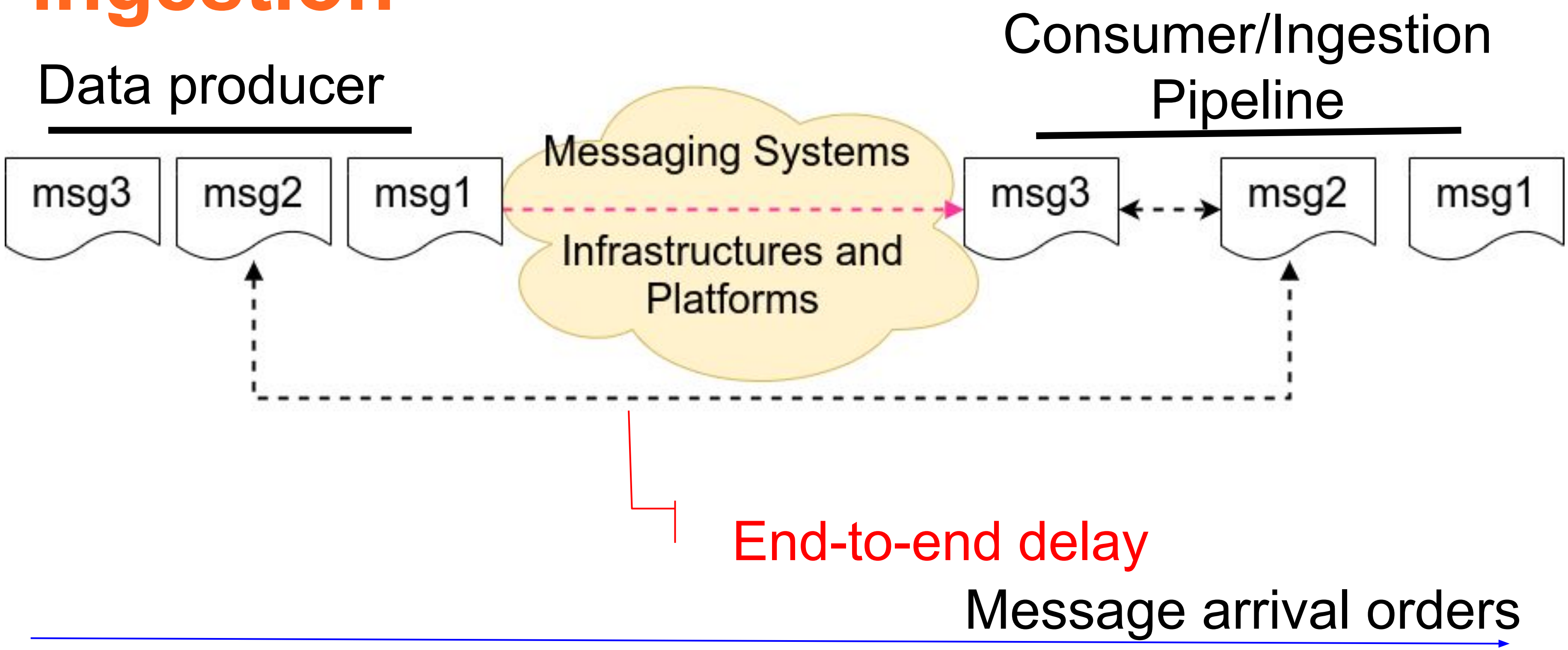


Figure source: https://docs.dask.org/en/stable/graphs.html

Near real time ingestion



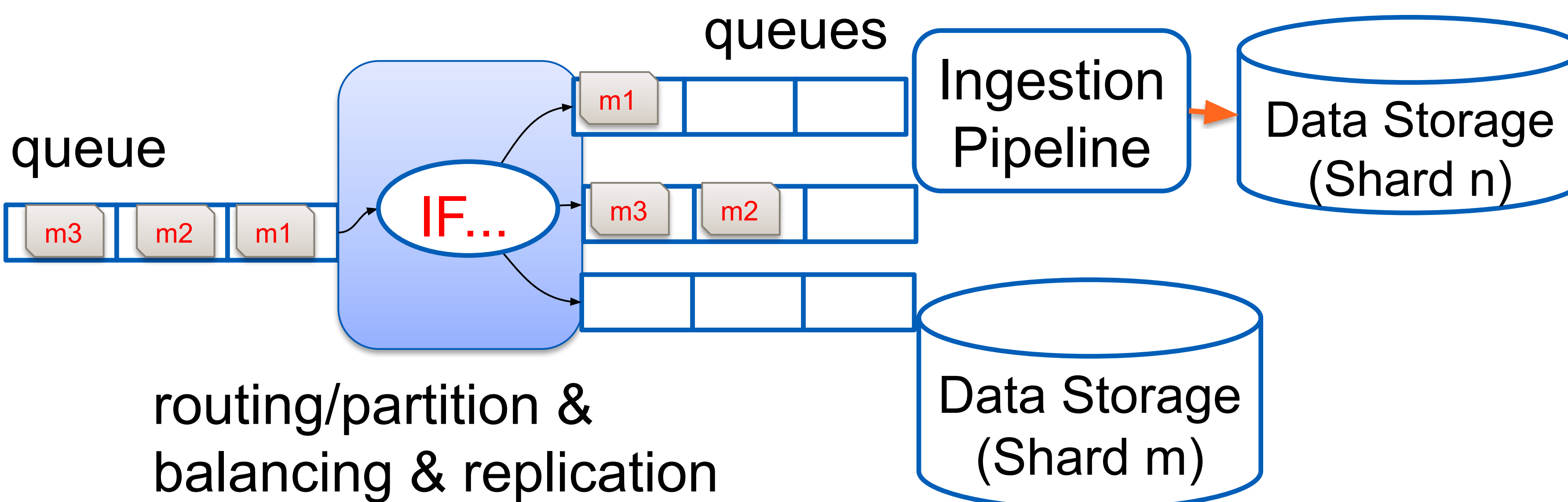
Key issues in streaming data ingestion



Some key issues for ingestion of streaming data

- **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
- **Data quality**
  - fast processing, overhead

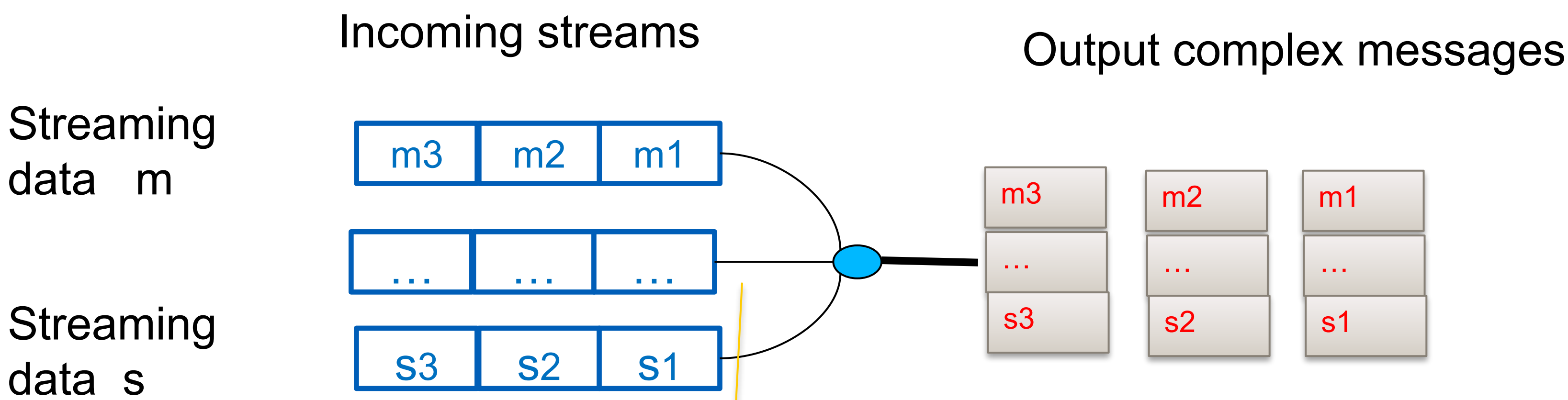
Split (pub/sub) and partition with ingestion



Some key issues

- **Multiple topics/streams of data**
  - speed and volume of data per topic vary
  - should not have duplicate data in data store
- **How to distribute topic/data to ingestion pipelines?**
  - application logics and performance
- **Where should we run the messaging system?**
- **Where and when should the elasticity be applied?**
  - computing resources + data resources (streams)

Procesing data before ingestion requires some streaming techniques



E.g., basic processing for data rollup/summarization

Aalto University
School of Science

Pipeline designs and execution models

Dealing with diverse data structures

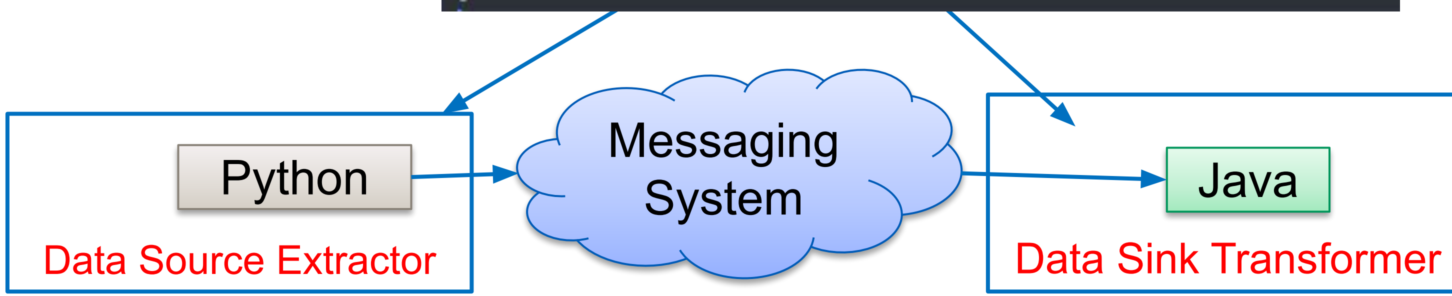
- **the data sender/producer and the receiver/consumer are diverse**
  - implemented with different languages and software technologies
    - *need to guarantee the message syntax and semantics*
  - performance overhead due to data format conversion
- **Solutions:** don't assume! agreed in advance
  - agreed in advance $\Rightarrow$ within the implementation or with a standard
  - know and use tools to deal with syntax differences
- **Understanding the syntax allows some automatic transformations/quality checks**
- **But semantics are domain/application-specific**

Example of interoperability in data transfer: Arvo

```
{
        "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"}

        ]
}
```

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

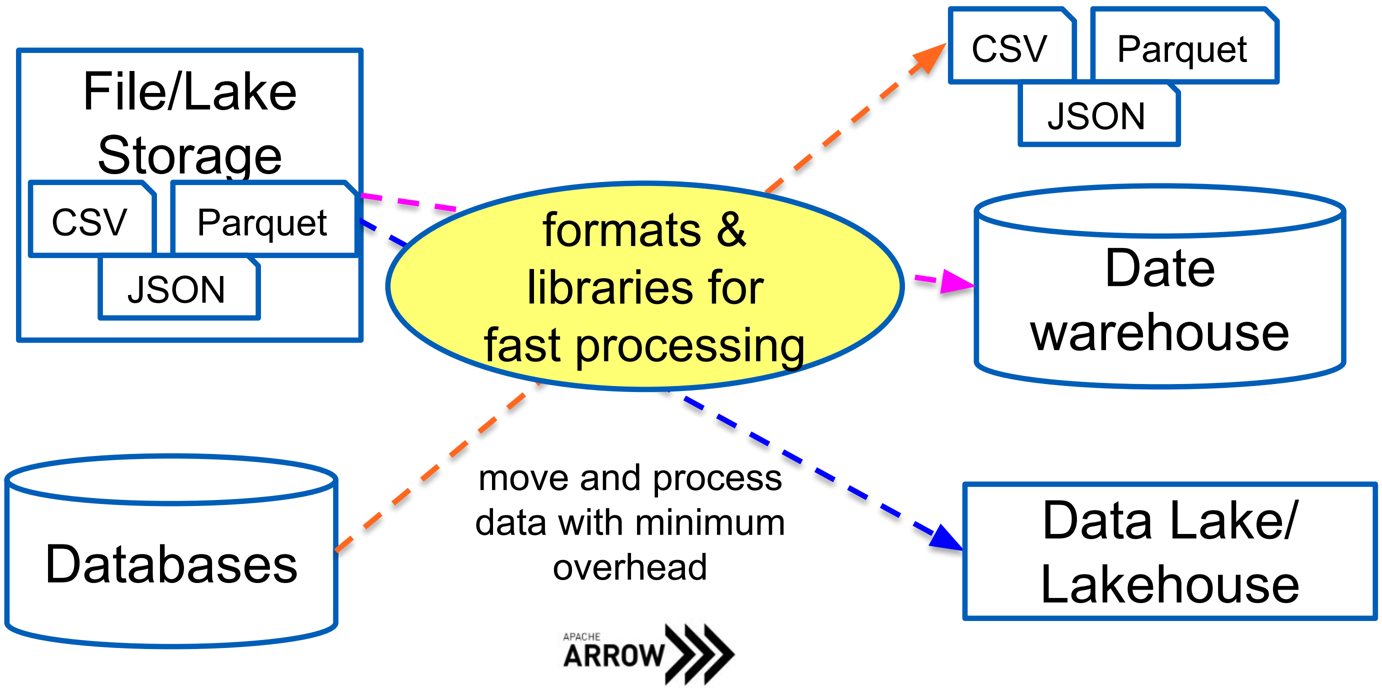


Also: Protobuf, JSON Schema

Interoperability in data processing in ETL

Data sources and formats

Data sinks and formats



Example: Apache Arrow

In-memory columnar format

rich ecosystems

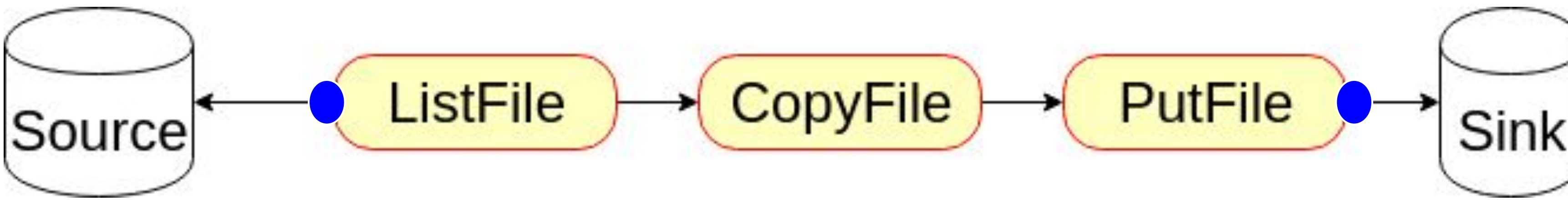
https://arrow.apache.org/

Ingestion is not a single task!

Ingestion pipelines/processes: architectures, composition, coordination, and tools

Complex deployment and composition models

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

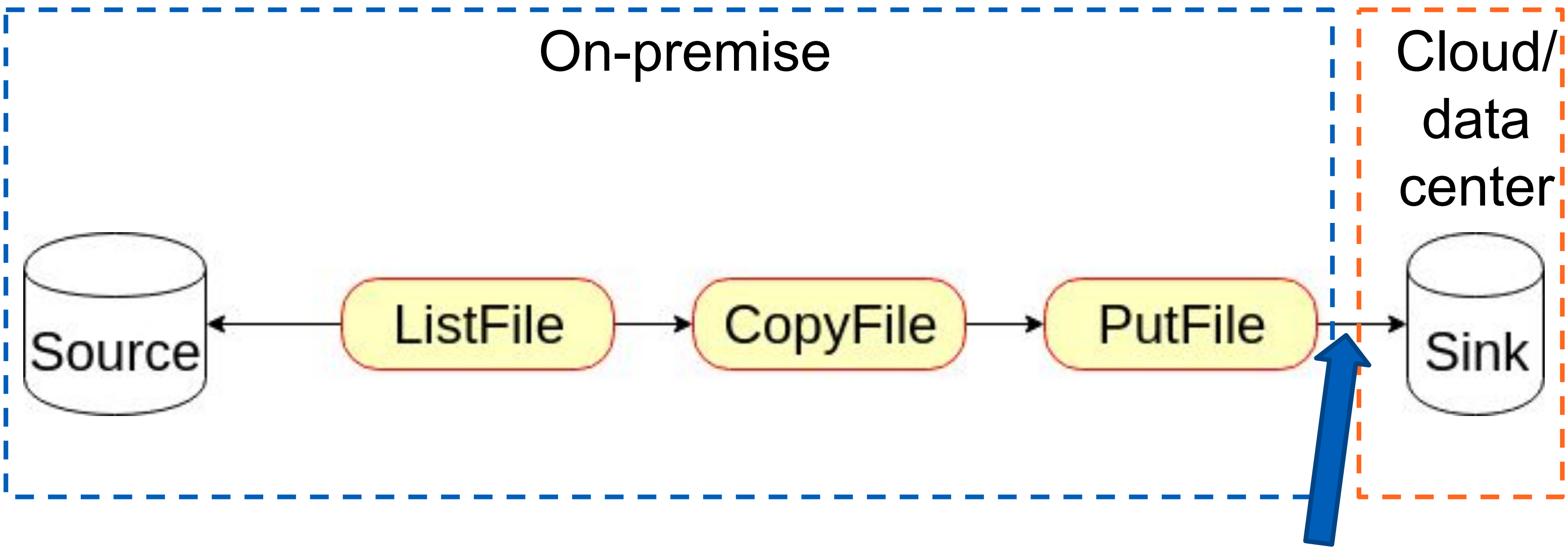


Tenant/ user

Ingestion pipeline developer (for whom?)

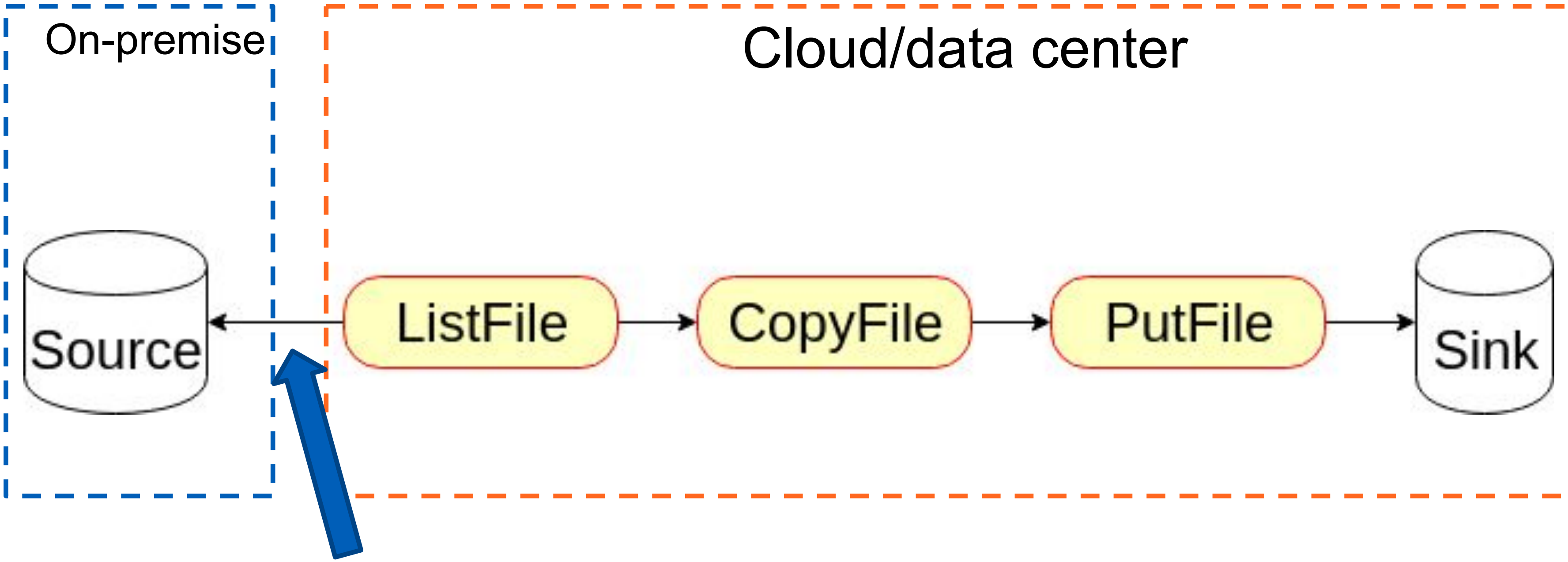
Data store/platform provider

Complex deployment and composition models



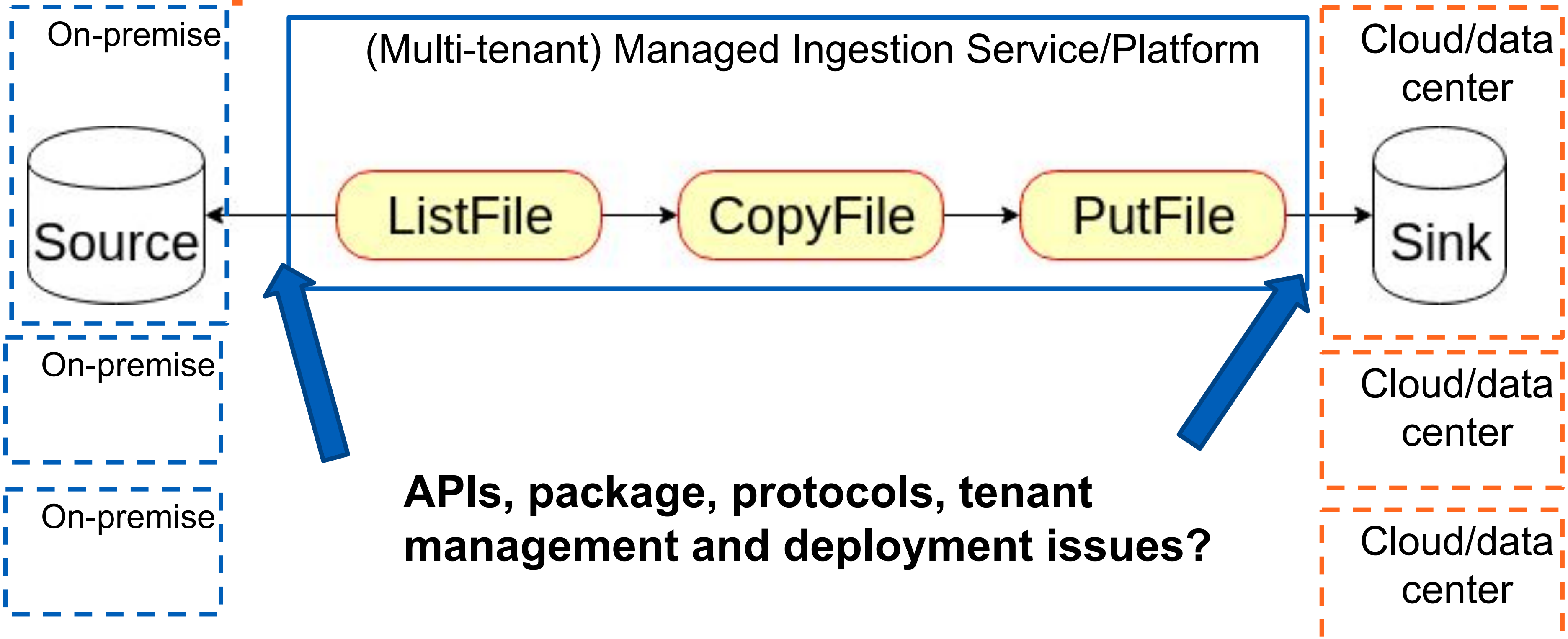
APIs, protocols and deployment issues?

Complex deployment and composition models



APIs, protocols and deployment issues?

Complex deployment and composition models

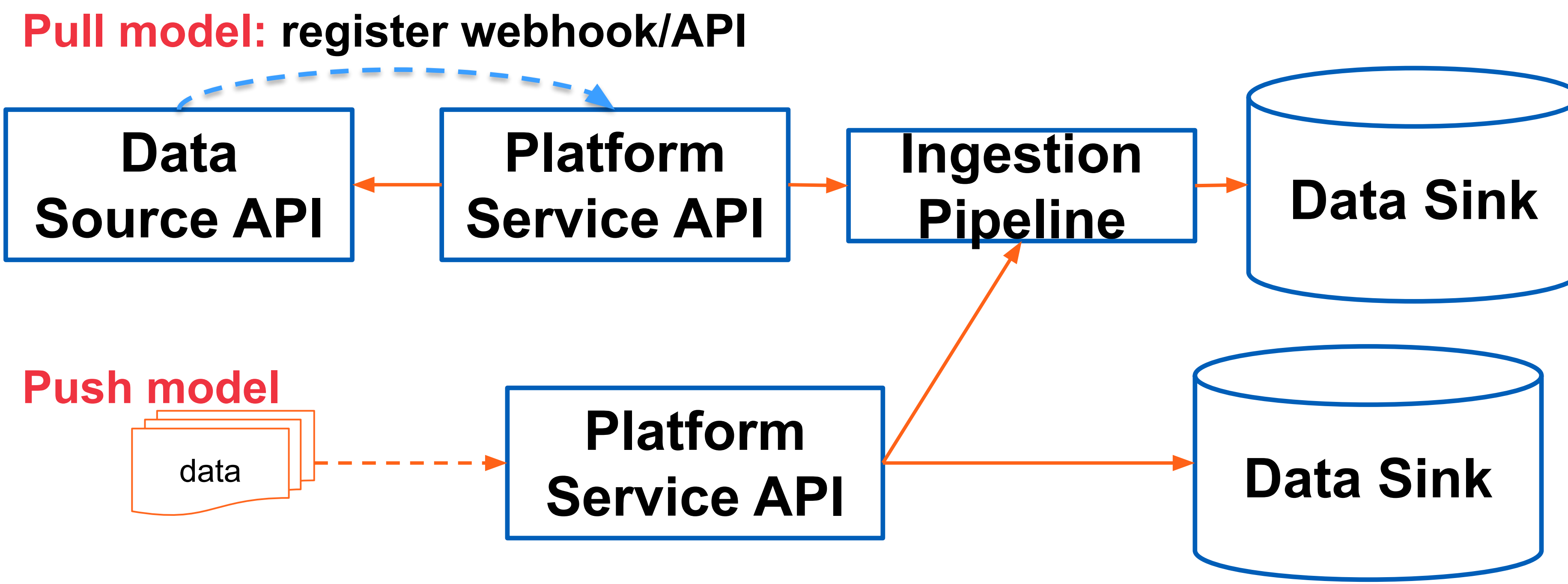


APIs, package, protocols, tenant management and deployment issues?

Orchestrating batch ingestion pipelines

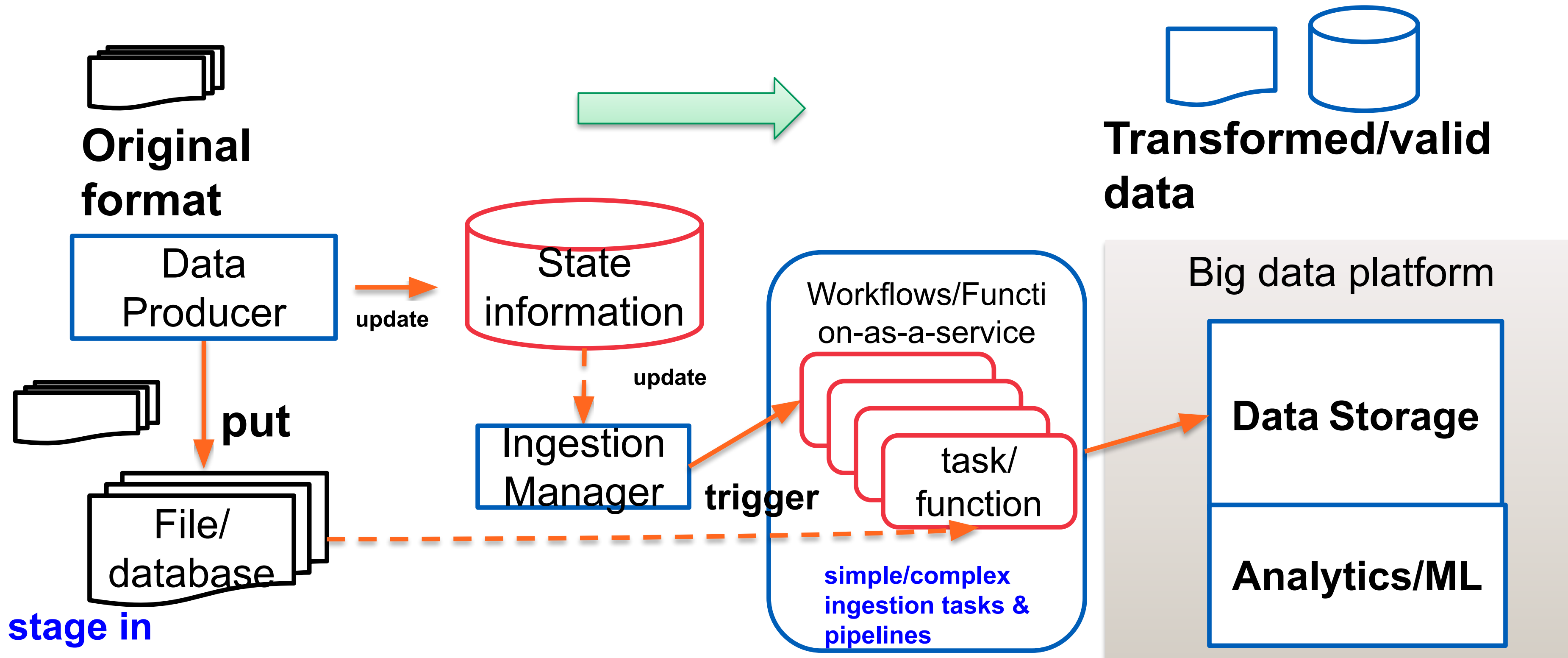
- **Data to be ingested is bounded**
  - files or messages are finite
- **Ingestion architectural styles**
  - (1) Direct APIs, (2) reactive pipelines, (3) tasks/workflows
- **Incremental ingestion**
  - dealing with the same data source but the data in the source has been changed over the time (related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

Simple, direct APIs for ingestion



Try to analyze pros and cons for your platform!

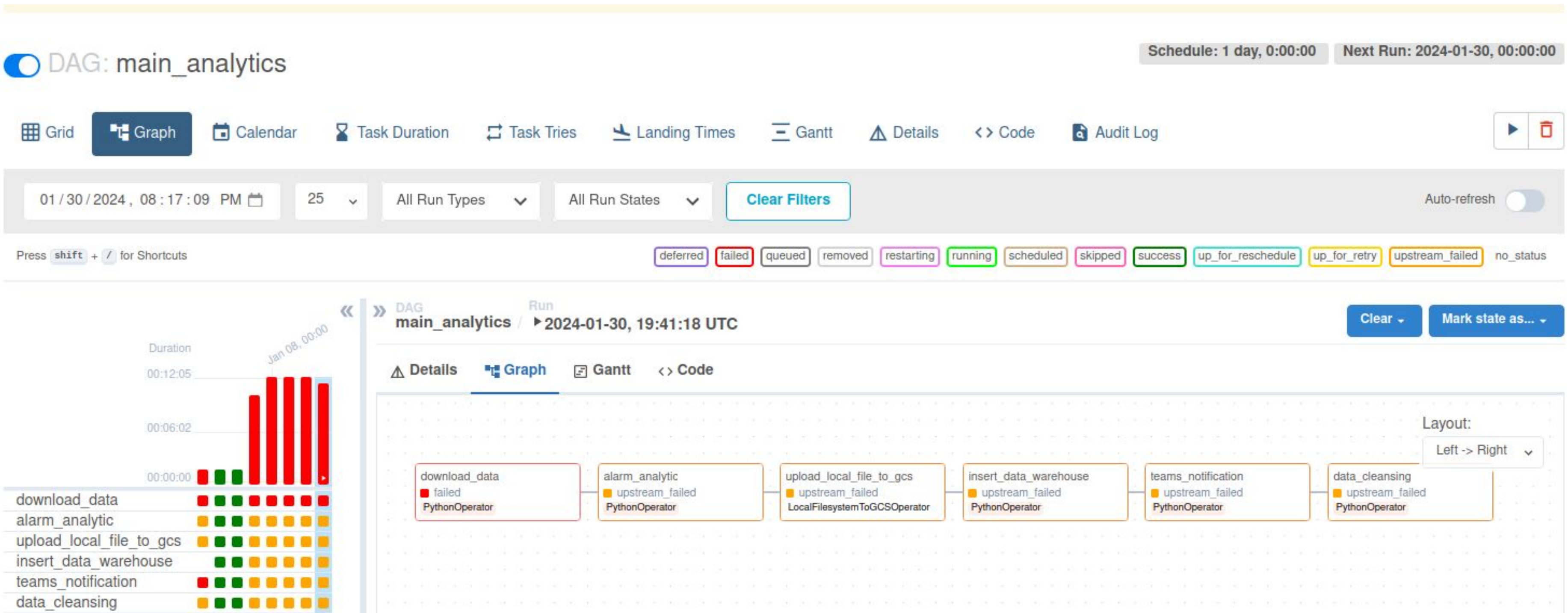
Reactive pipelines with functions/workflows/containers



Ingestion workflows orchestration

- **Different tasks**
  - access and copy, extract, covert, quality check, and write data
  - tasks are 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
  - flexible designs
- **Different tenants have different service level agreements**
  - Performance, reliability, and cost.

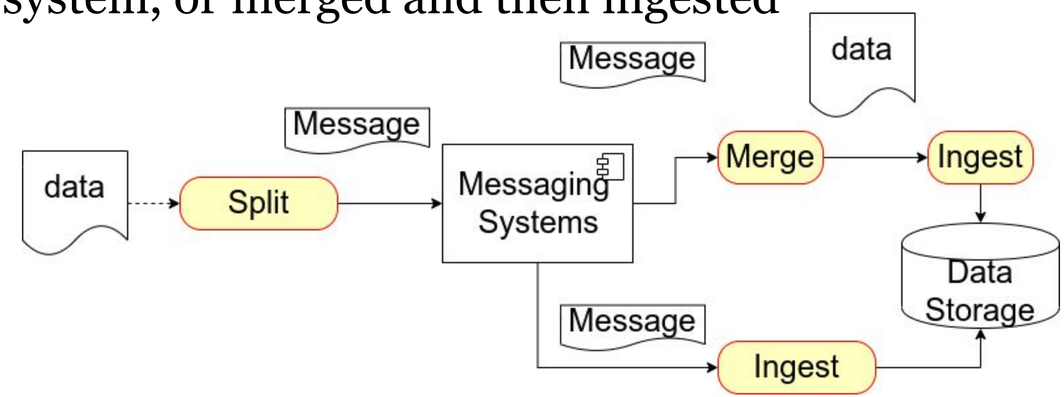
E.g., workflow based on scheduled time, with Apache Airflow



Microbatching in data ingestion

- **Microbatching: we mean the application/tenant strategy to deal with big dataset using batches of data (small chunks)**
  - not necessary the same as using batch systems to transfer small data in near realtime
- **Data is split into different chunks for ingestion**
  - using streaming or batch systems to transfer data chunks
  - chunks are ingested into the system, or merged and then ingested

Example: with streaming system



Be careful with the data semantics/integrity!

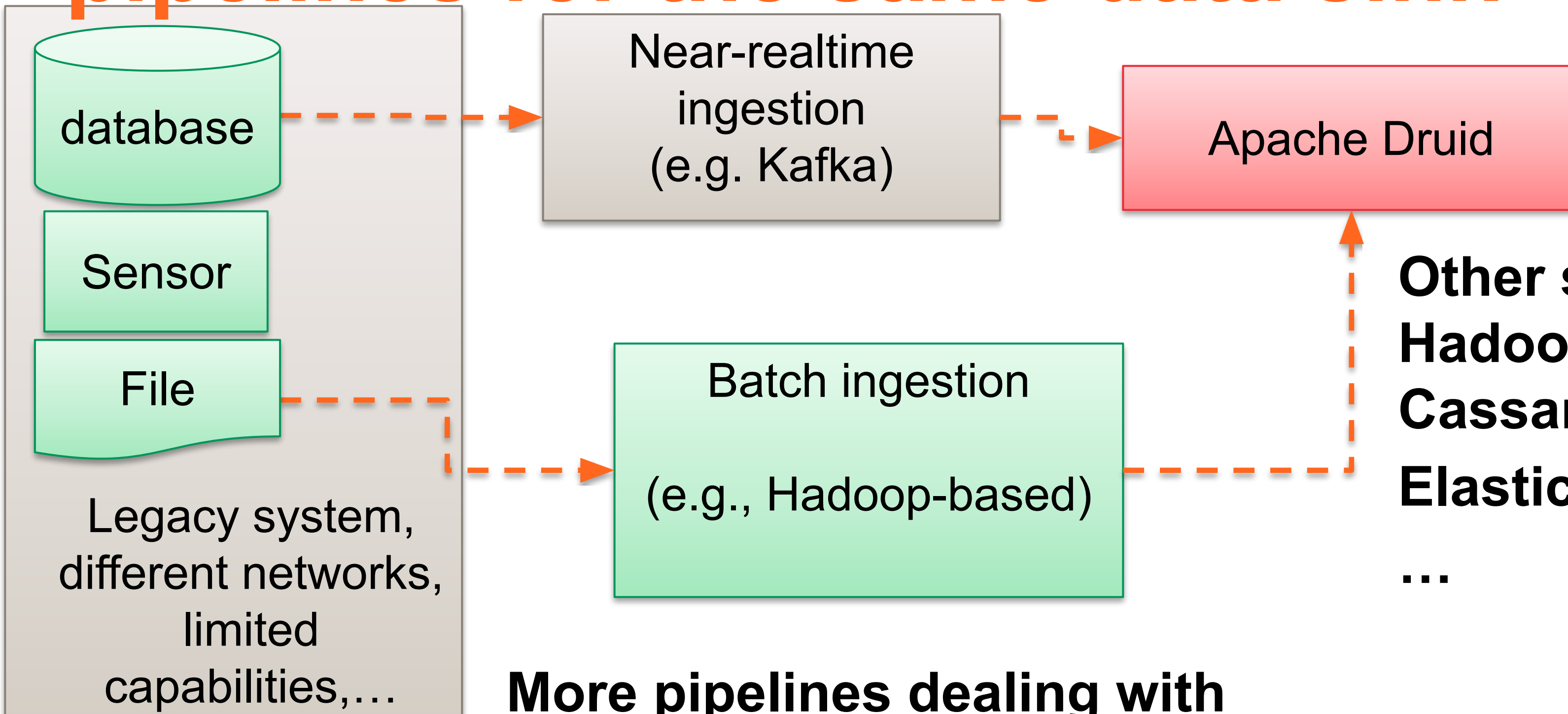
Combining ingestion pipelines in big data platforms

- **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”

Supporting multiple types of pipelines for the same data sink

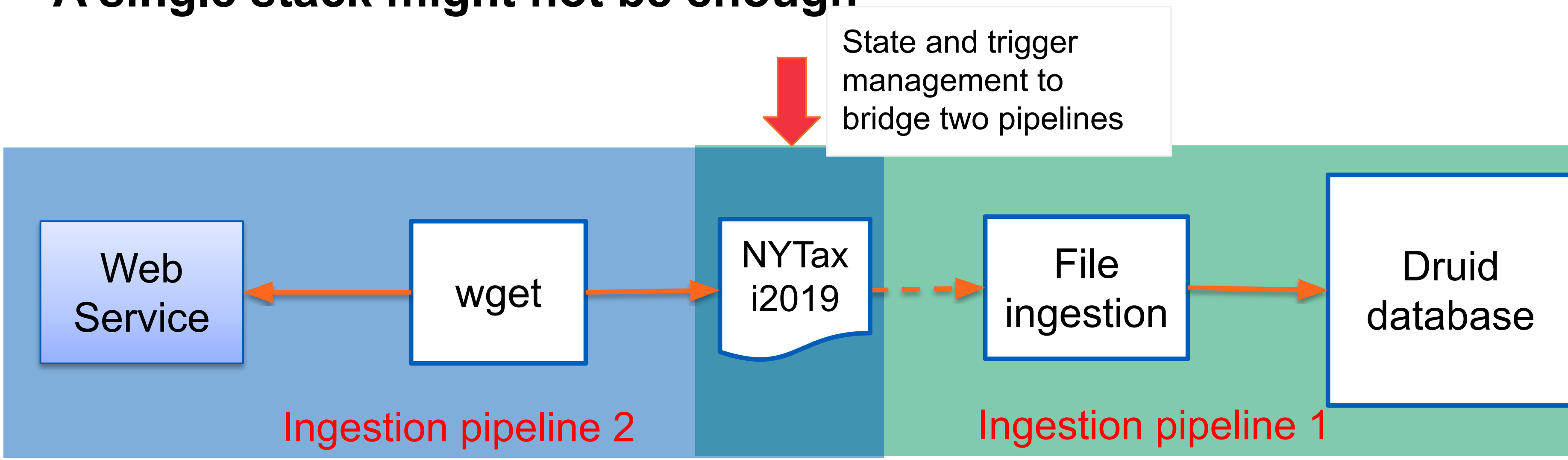


Other systems: Hadoop, Cassandra ElasticSearch …

More pipelines dealing with different sources

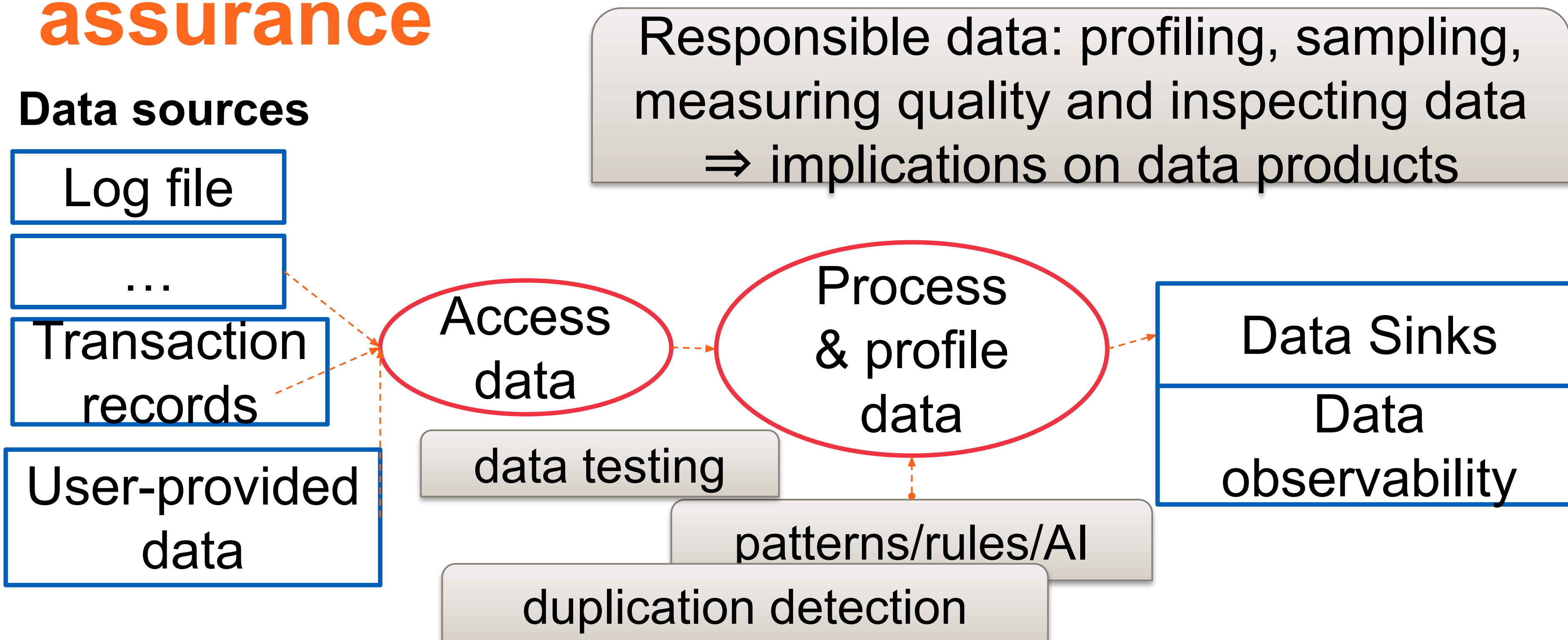
Connecting different ingestion pipelines

A single stack might not be enough



Real-world:
both pipelines and their connections are complex

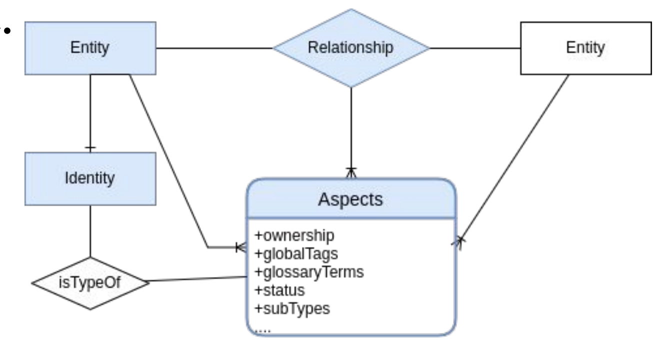
Quality control/data regulation assurance



Challenging issues: misinformation, GDPR, data quality, inappropriate content

Data lineage and observability (1)

- **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

Data lineage and observability (2)

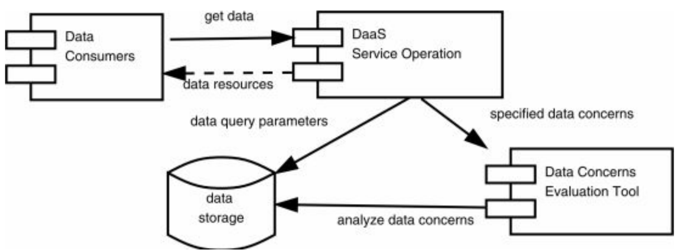
- **Data observability: the health about data**
  - near-real time metrics, offline checks and possible dashboards
  - similar to service observability, relying on traces, logs, metrics, etc.
- **Focus on data**
  - data metrics (volumes, data quality, schemas, lineage)
  - issues due to data problems
  - 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 attributes → integrate with data ingestion processes or offline data profiling
  - integrated data quality tests in pipelines (e.g., data testing)

Tools: great expectations, OpenLineage, Microsoft Presidio, YData Profiling, aws deequ Public

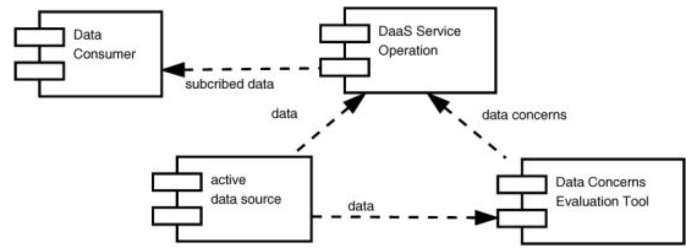
Quality control/data regulation assurance (1)

Design: dififerent evaluation modes

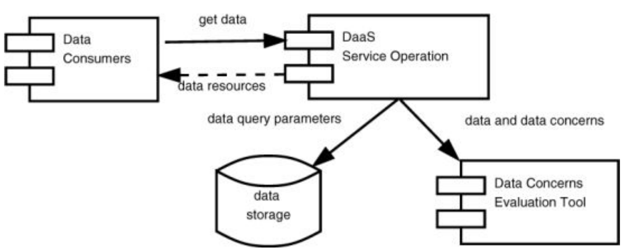
Pull, pass-by-reference model for evaluating data concerns



Push model for evaluating data concerns of active data sources



Pull, pass-by-value model for evaluating data concerns



Source: H. -L. Truong and S. Dustdar, "On Evaluating and Publishing Data Concerns for Data as a Service," *2010 IEEE Asia-Pacific Services Computing Conference*, doi: 10.1109/APSCC.2010.54.

Quality control/data regulation assurance (2)

- **Before, after or during the ingestion/transformation**
- **In-process vs out-process**
  - in process: using libraries doing data quality very fast
  - out-process: a separate task in the workflow or external programs/services
- **Profiling, sampling, ML techniques for data quality**
- **Examples:**
  - Using a separate program like pydeequ Spark to check quality
    - *https://github.com/rdsea/bigdataplatforms/tree/master/tutorials/dataquality*
  - Anonymizing data
    - *https://microsoft.github.io/presidio/anonymizer/*

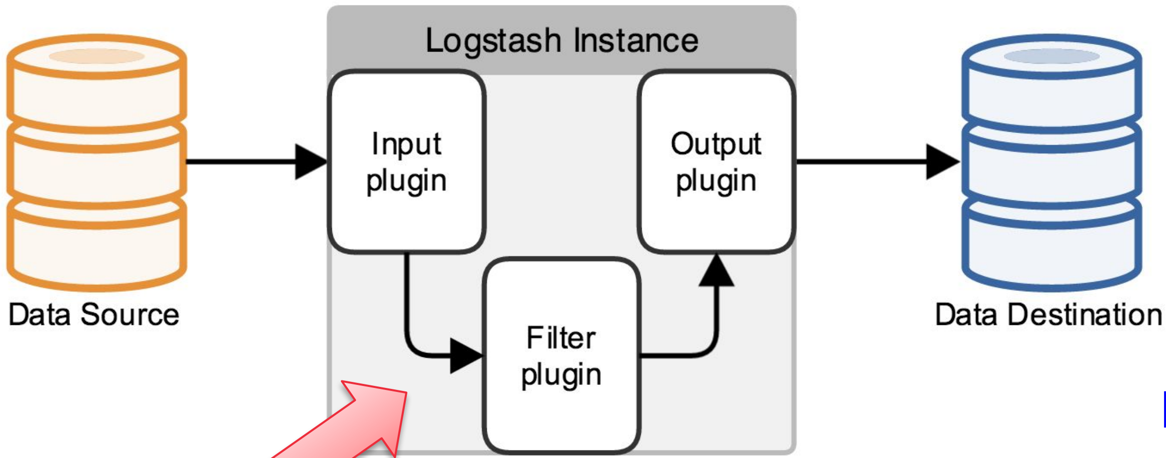
Aalto University
School of Science

Tooling for ingestion pipelines

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: Logstash




```
input {
  file {
    path => "${MY_INPUT_DIR}/bts-data-alarm-2017.csv"
    start_position => "beginning"
  }
}
filter {
  csv {
    separator => ","
    columns => ["station_id","datapoint_id","alarm_id",
  }

}

output {
    stdout {}
    #...
}
```

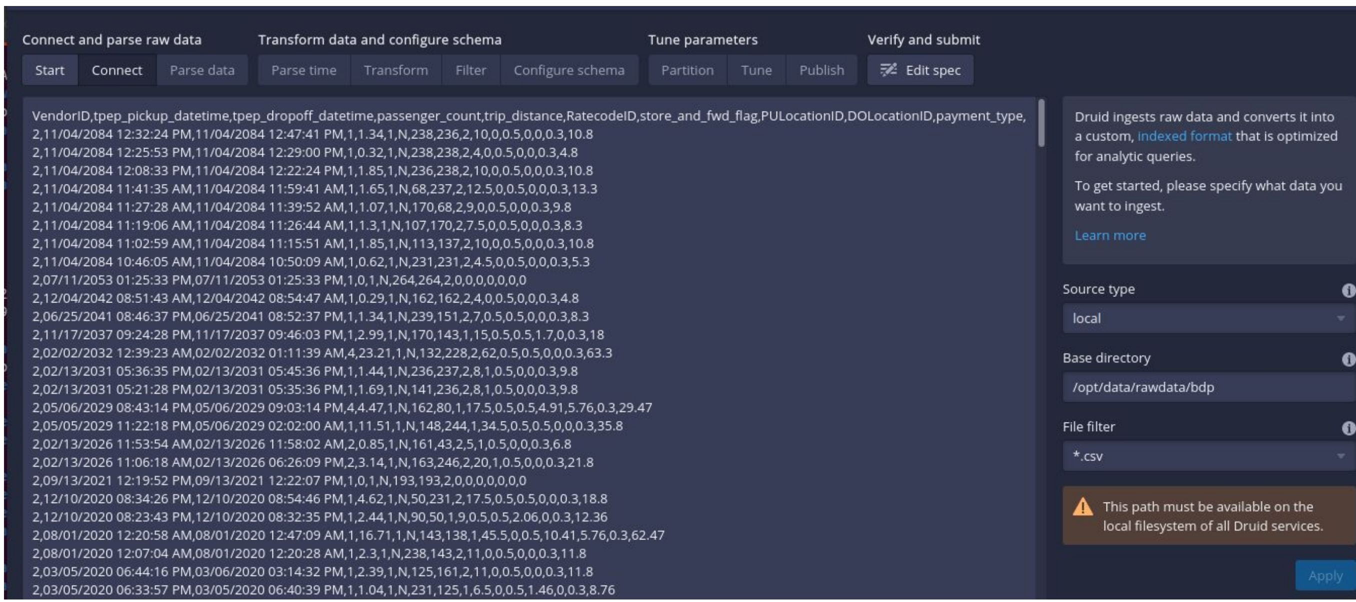
Pipeline is defined in a configuration file

Pluggable approaches

Figure source: https://www.elastic.co/guide/en/logstash/current/getting-started-with-logstash.html (from the previous version of Logstash)

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

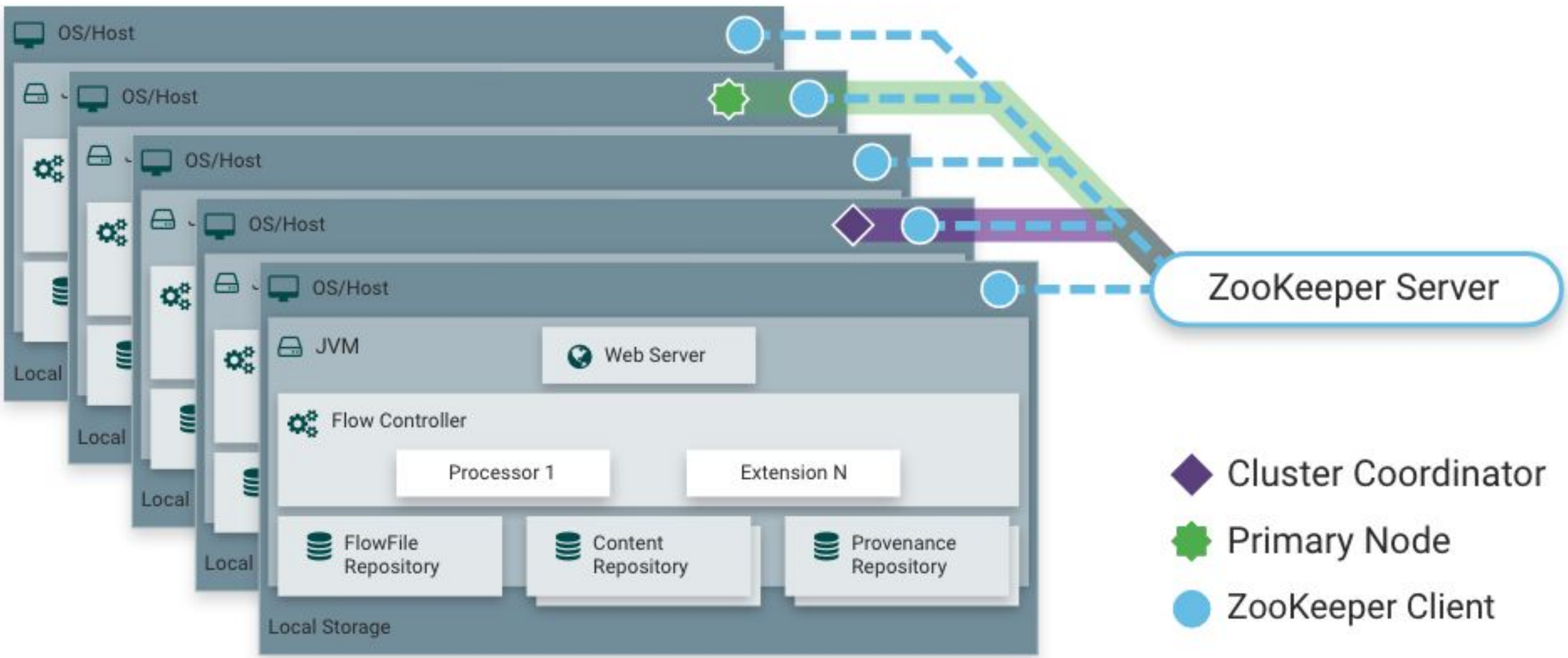
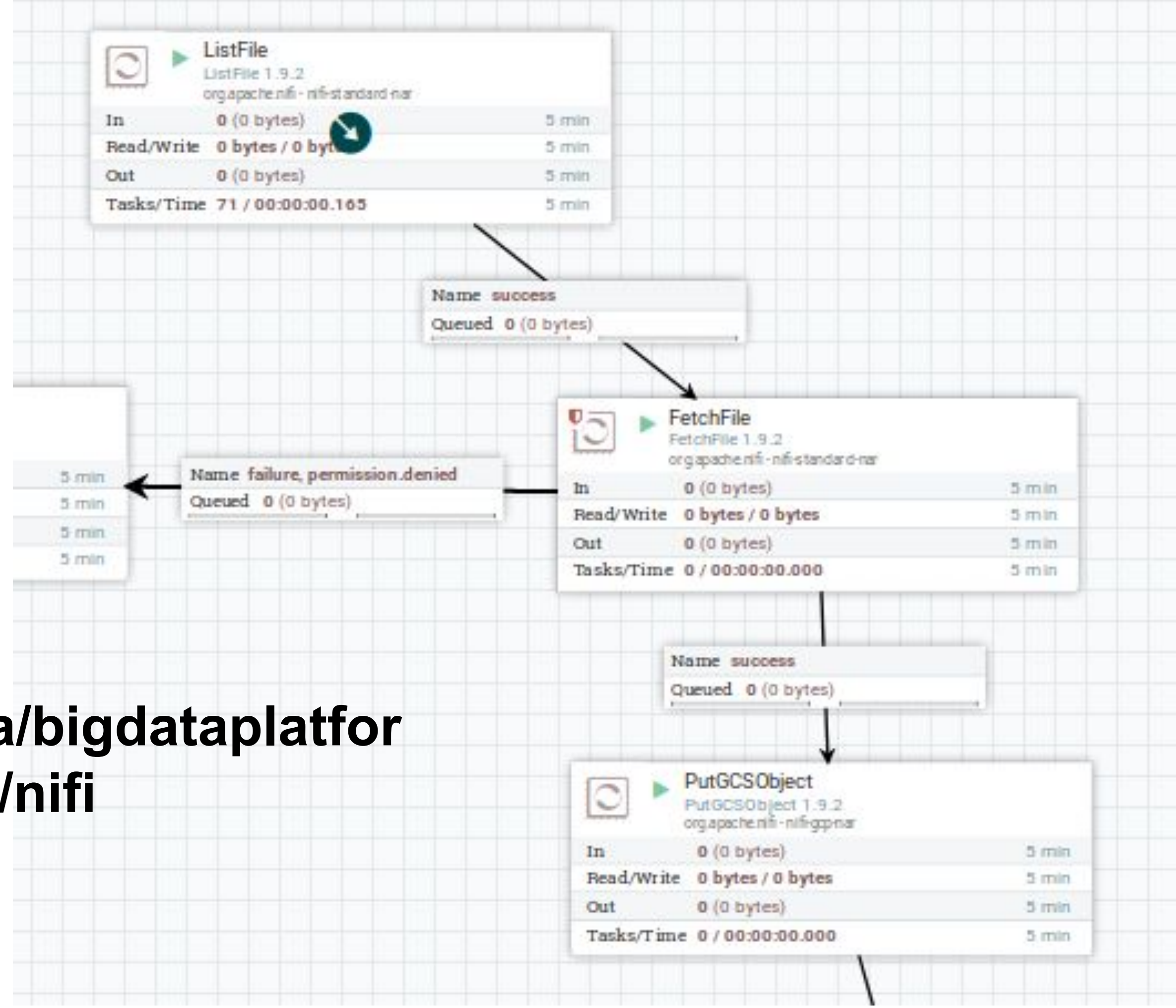


Figure source: https://nifi.apache.org/docs.html

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://github.com/rdsea/bigdataplatforms/tree/master/tutorials/nifi

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
- **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!

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io