



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

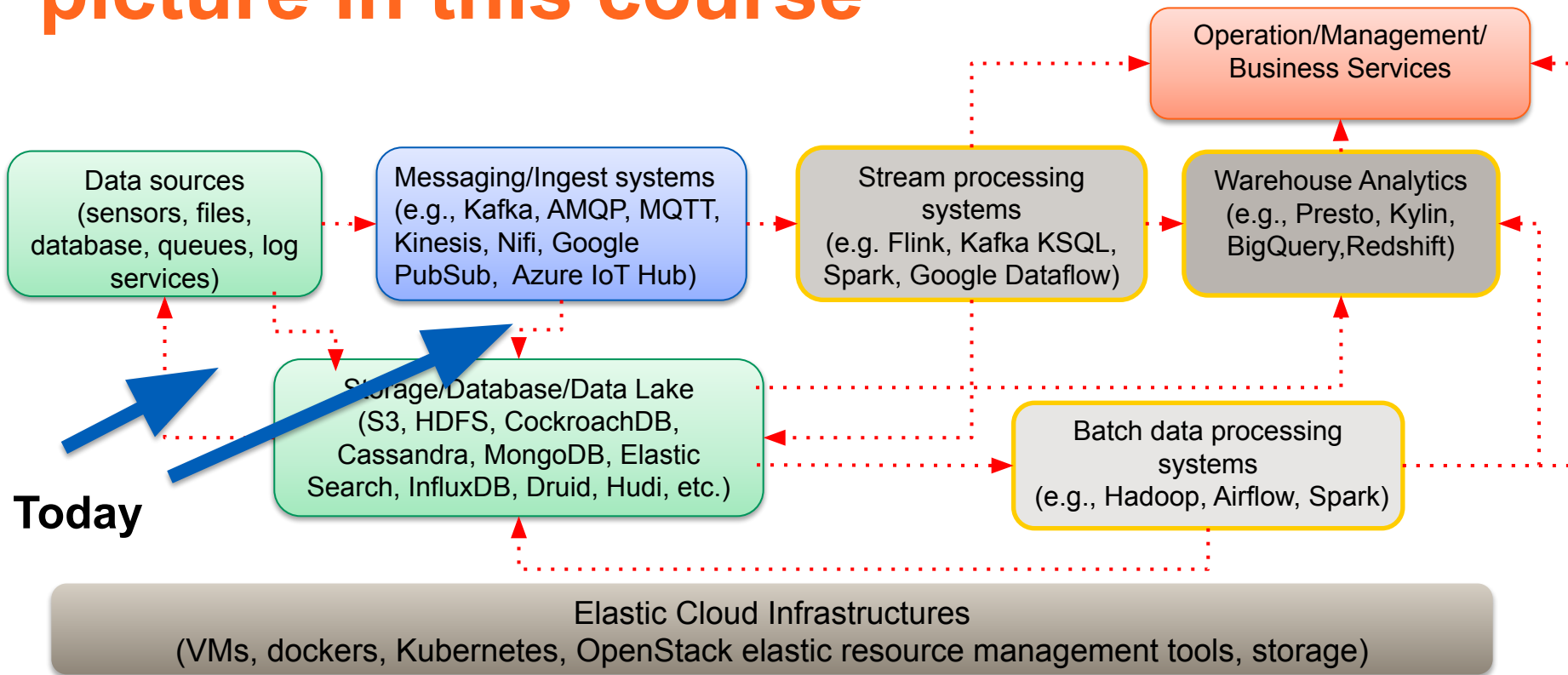
**Bring the data to the right place!
where is the right place?**

**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)

Data sources
(files,
databases,...)



Big data platform

e.g.

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

Two important aspects:

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

Reusability and extensibility

Data sources and sinks

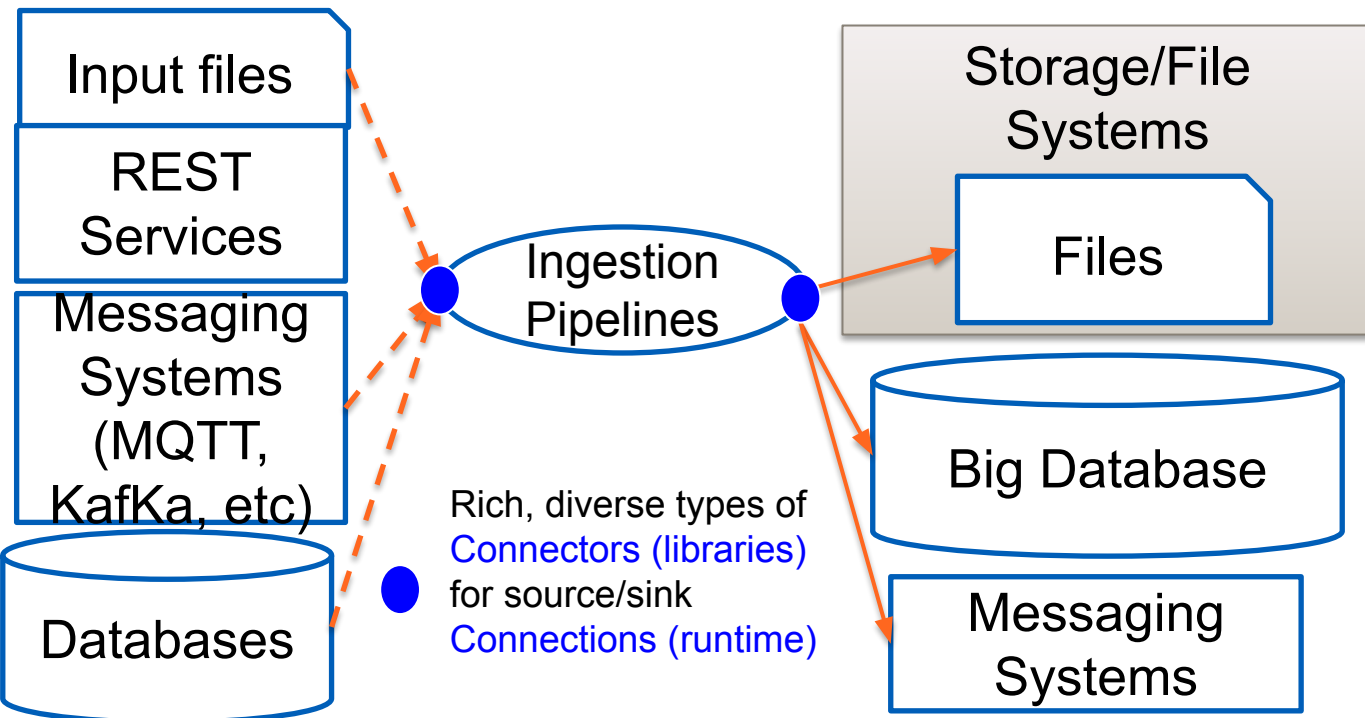
Data sources

Data 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

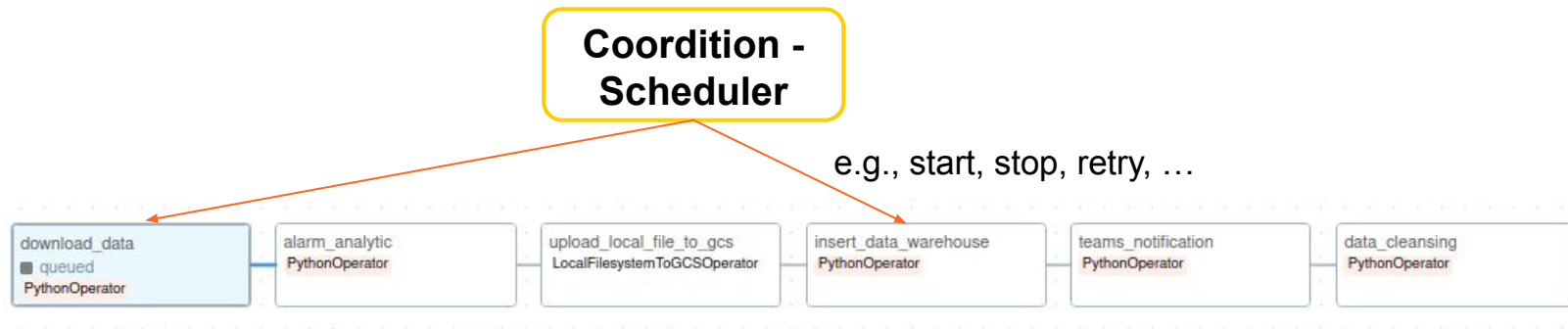
- **Interact with data sources: access APIs and protocols**
 - REST API, ODBC, SFTP, specific client libs
 - MQTT, AMQP, CoAP, NATS, Kafka,...
- **Requirements based on data characteristics**
 - structured, unstructured and semi-structured
 - speed, volume, accuracy, confidentiality, data regulation
- **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 we have**
 - 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/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

Ingestion tasks implemented as extensible, composable connectors

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



- Data compression and security must be considered

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
- 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;D
EF081;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 }
```

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, etc.
 - 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**

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

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']];
                    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)
```

Dealing with diverse data structures

- **Remember that the data sender/producer and the receiver/consumer are **diverse****
 - implemented with different languages and technologies
 - *need to guarantee the message syntax and semantics*
 - performance overhead due to 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 check**
- **But semantics are domain/application-specific**

Interoperability in data transfer: 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"}  
  ]  
}
```



Also: Protobuf, JSON Schema

Interoperability in data processing in ETL

Data sources and formats

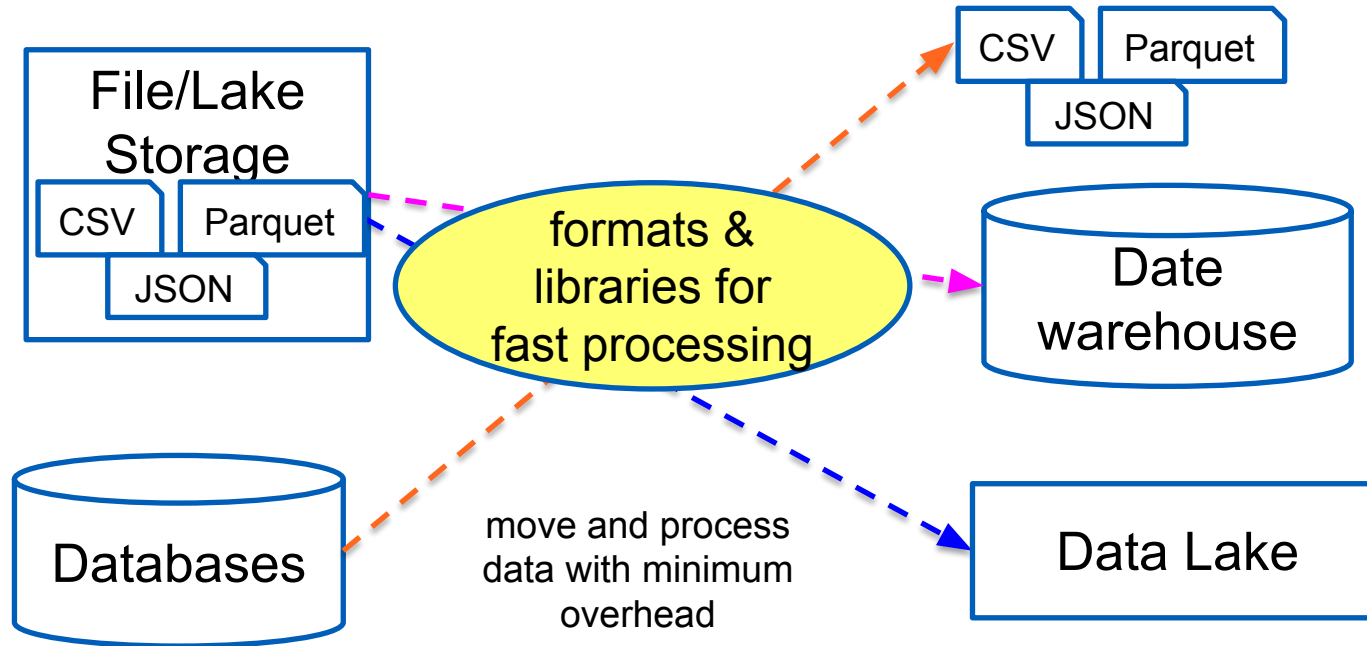
Data sinks and formats

Example: Apache Arrow

In-memory
columnar format

rich ecosystems

<https://arrow.apache.org/>



Quality control/data regulation assurance

Data sources

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

Log file

Transaction records

User-provided data

Access data

data testing

Parse & profile data

patterns/rules/AI

Databases
Data monitoring

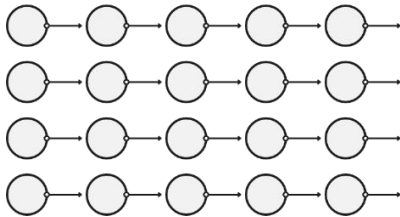
Hard issues: misinformation, GDPR, data quality, inappropriate content

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

- **Code hides 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

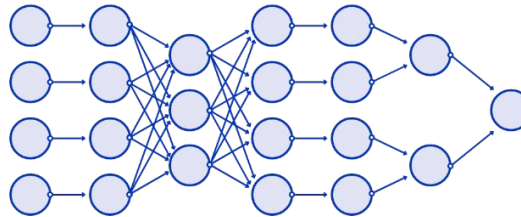
Embarrassingly Parallel

Hadoop/Spark/Dask/Airflow/Prefect



MapReduce

Hadoop/Spark/Dask



Full Task Scheduling

Dask/Airflow/Prefect

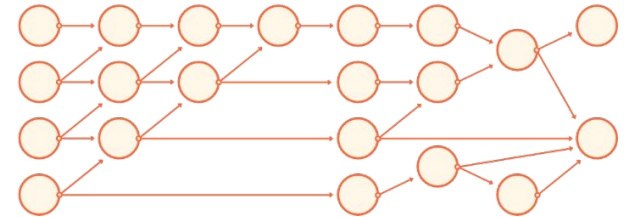
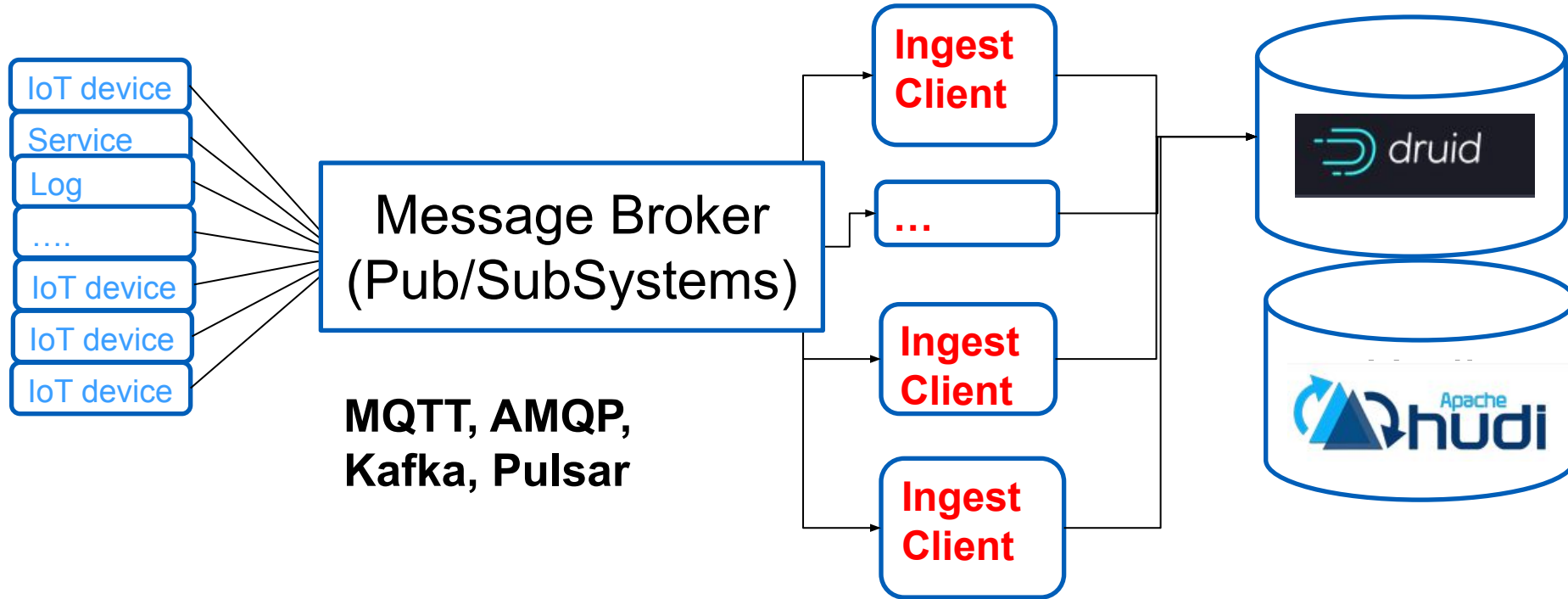
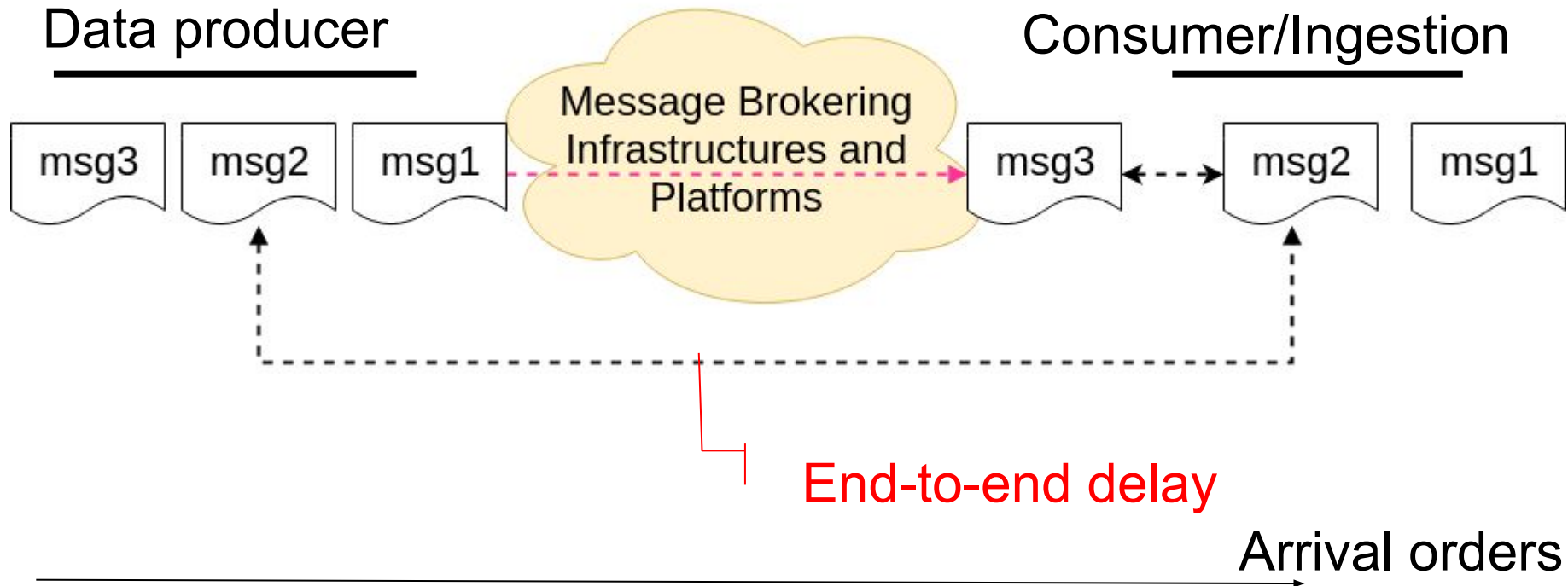


Figure source: <https://docs.dask.org/en/stable/graphs.html>

Near real time ingestion



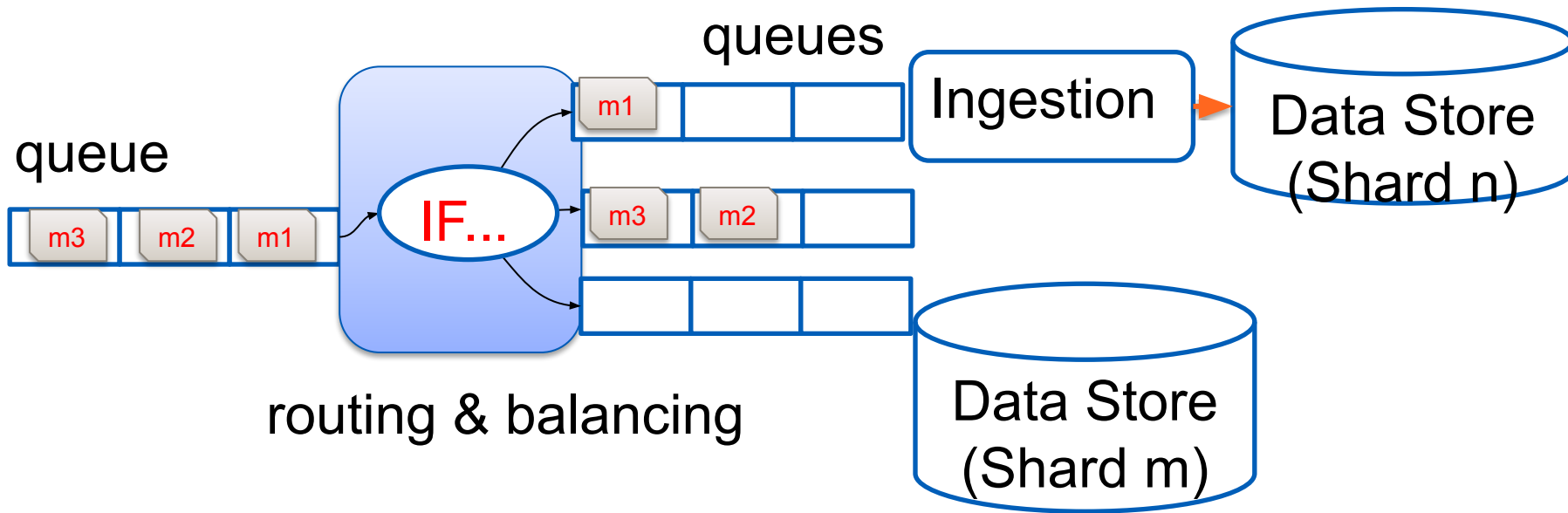
Key issues in streaming data ingestion



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
- **Data quality**

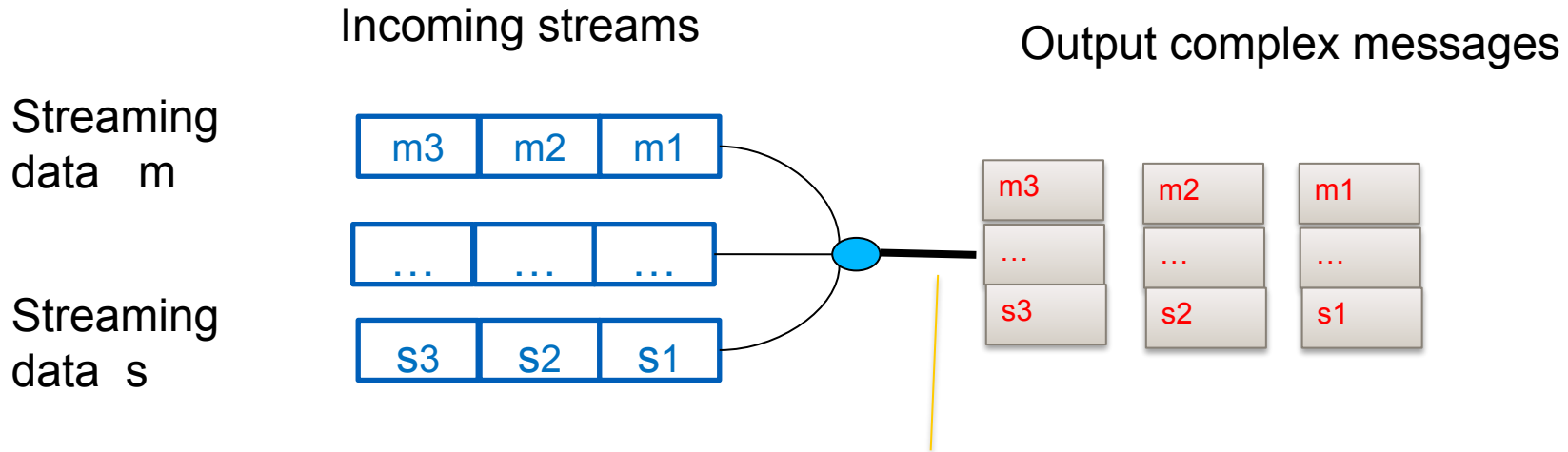
Split (pub/sub) and partition with ingestion



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?**

Processing data before ingestion requires some streaming techniques



E.g., for data rollup/summarization

Pipeline designs and execution models

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**

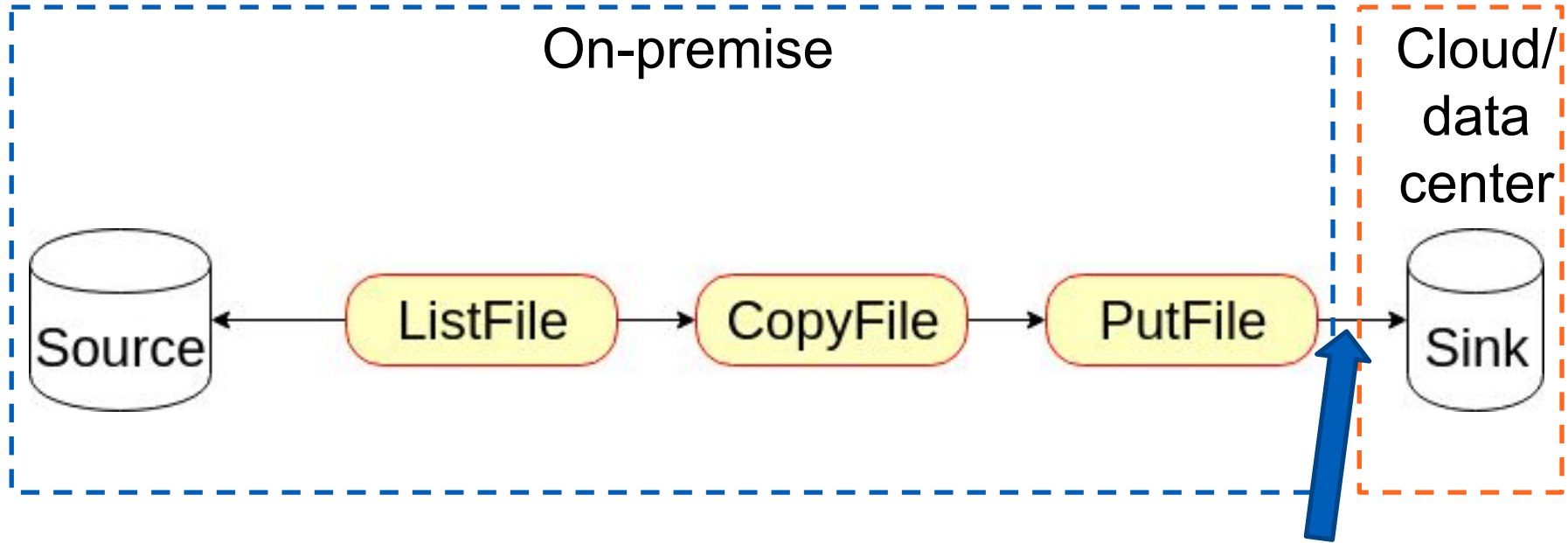


**Tenant/
user**

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

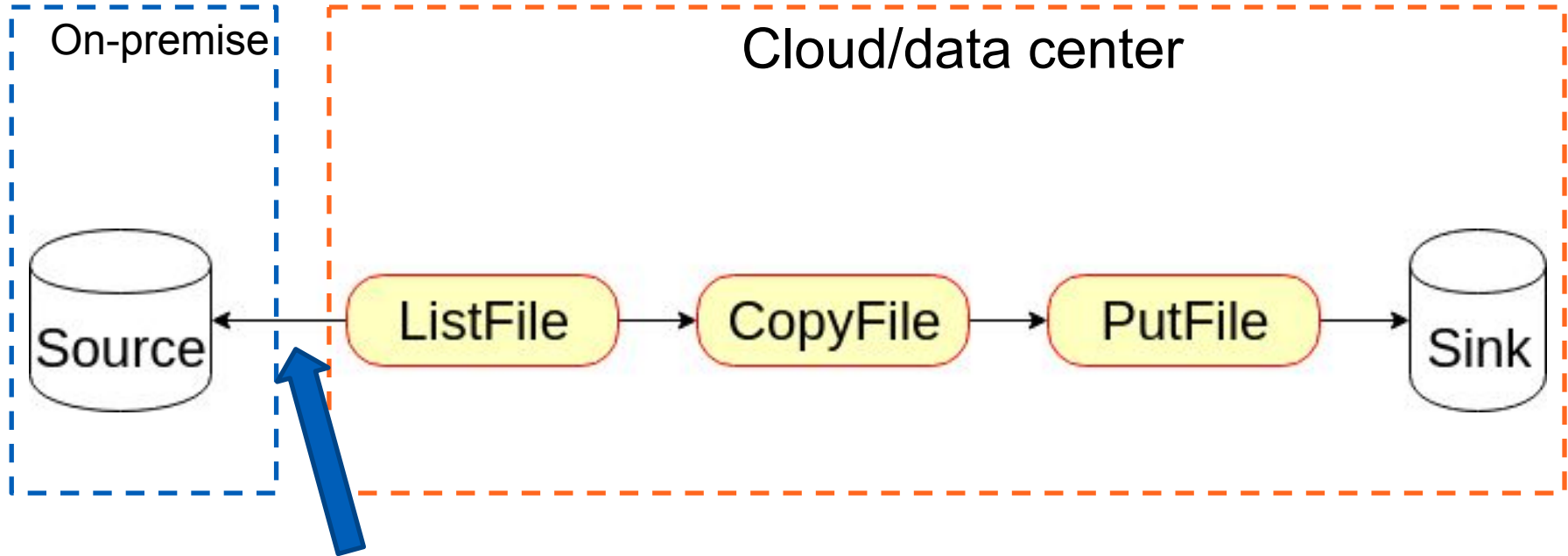
**Data
store/platform
provider**

Complex deployment and composition models



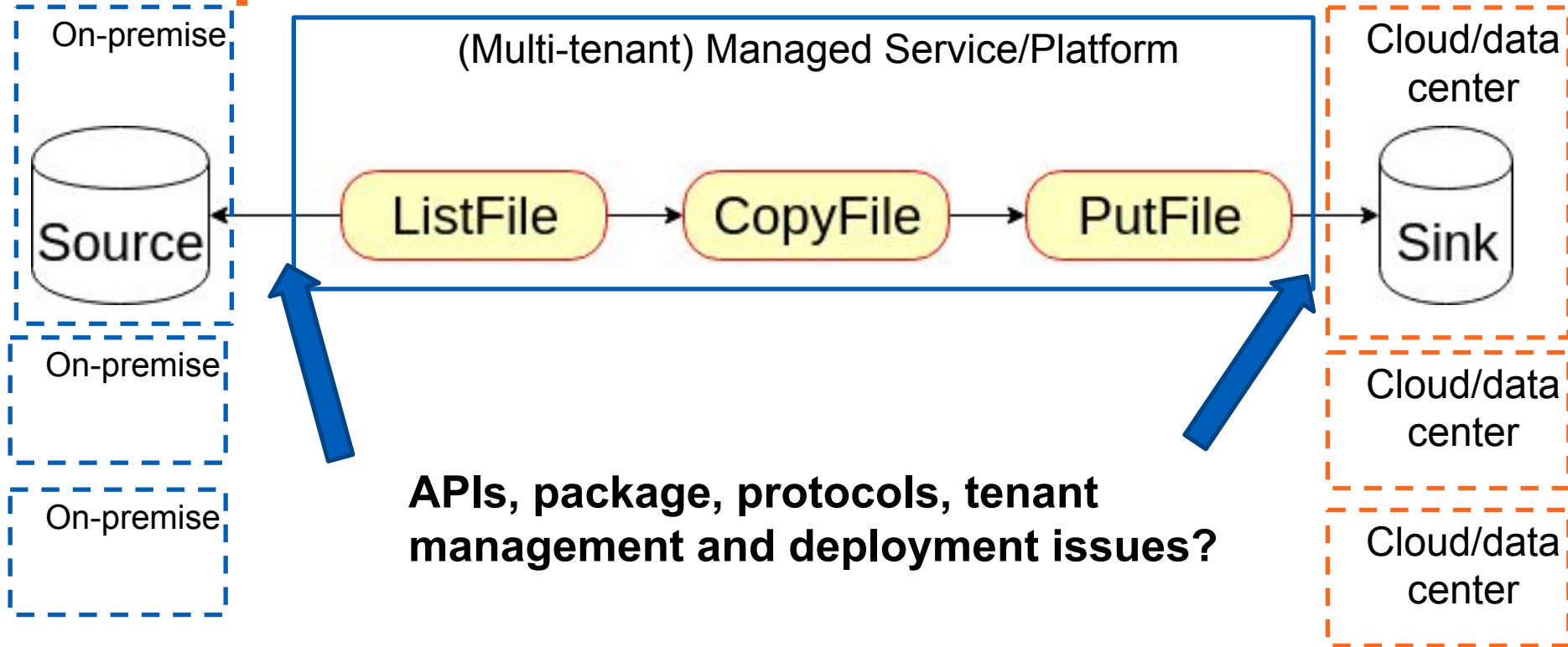
APIs, protocols and deployment issues?

Complex deployment and composition models



APIs, protocols and deployment issues?

Complex deployment and composition models



Architecture requirements

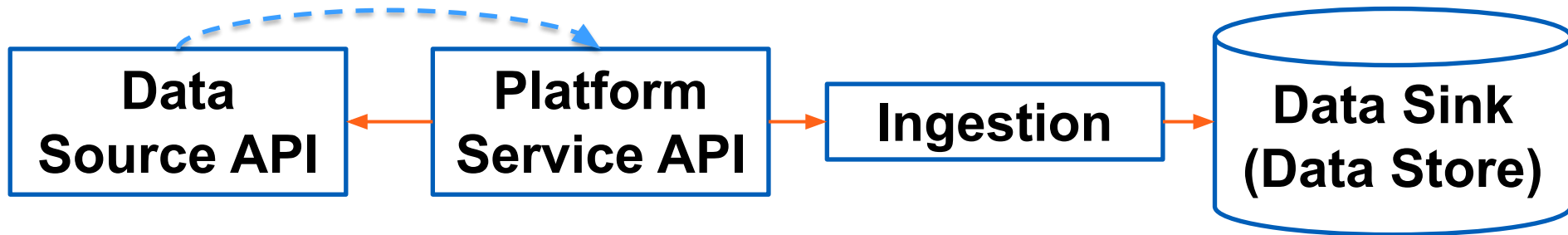
- **Data source integration**
 - the richness and extensibility of data sources and data sinks
- **Batch ingestion and near real-time ingestion requirements**
- **Integration between different ingestion processes across distributed places**
- **The architecture addresses “big data” properties**

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) task pipelines/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

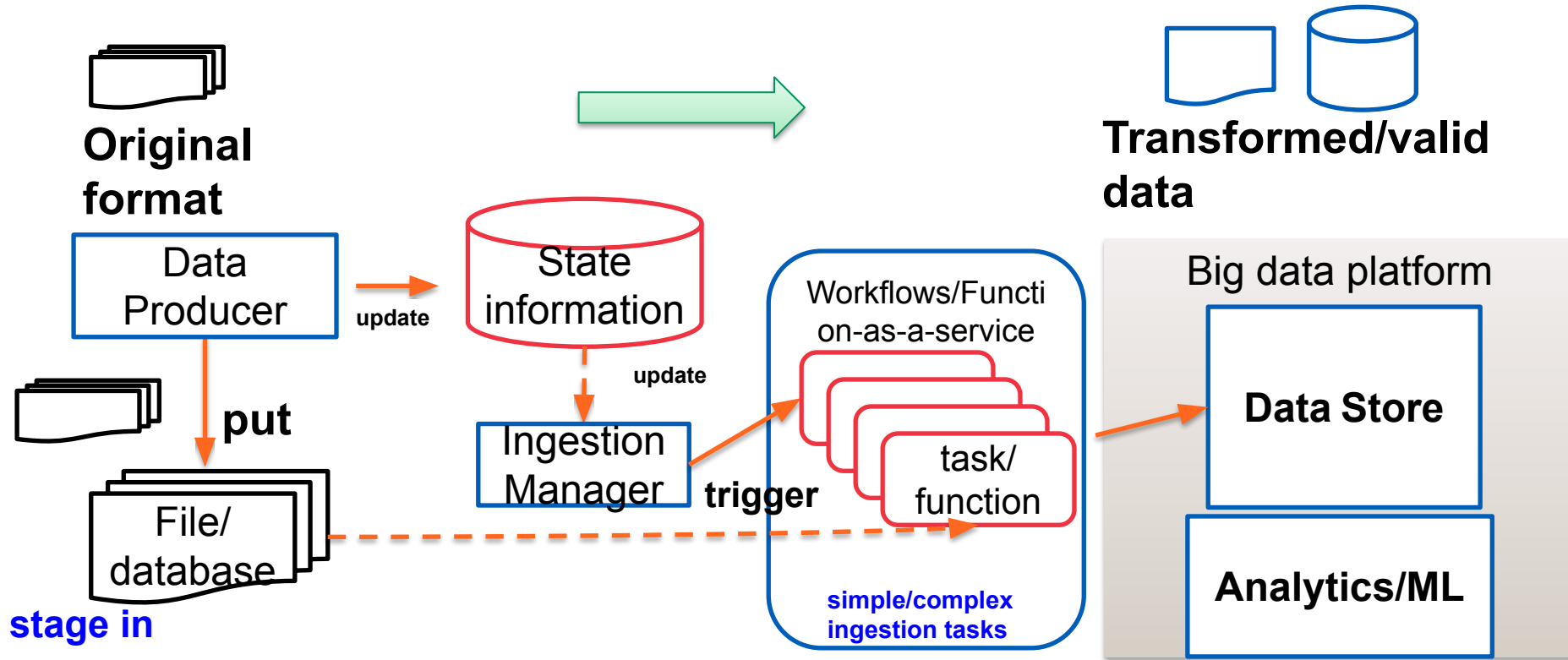


Push model
When?



Try to analyze pros and cons for your platform!

With functions/workflows/containers

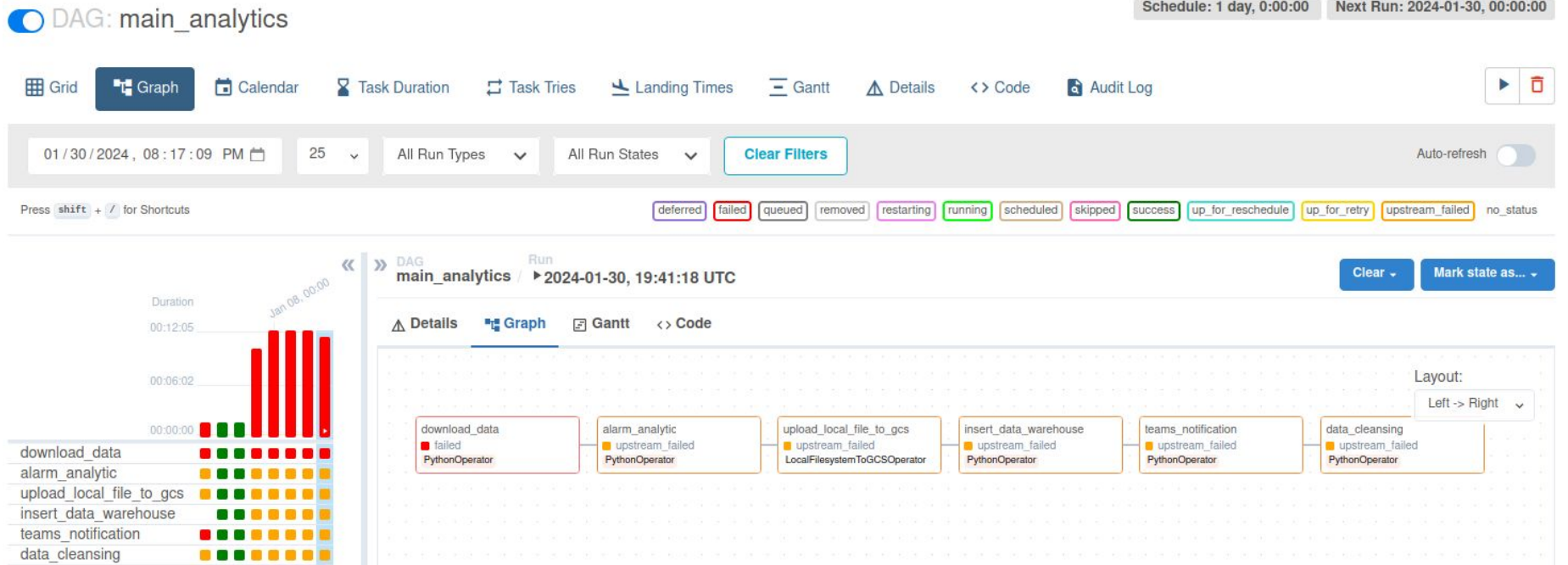


Who develops which components?

Orchestrating ingestion workflow

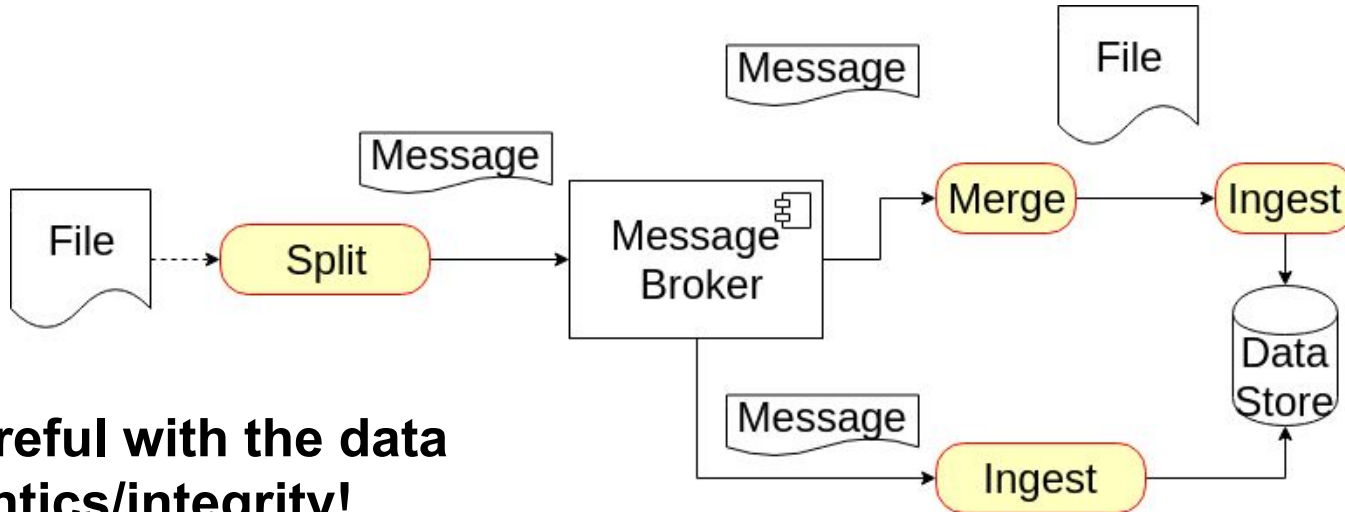
- **Different tasks for**
 - access and copy, extract, covert, quality check, and write data
 - tasks are connected based on data or control flows
- **Workflows**
 - a set of connected tasks is executed by an engine
 - tasks can be scheduled and executed in different places
 - flexible designs

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



Microbatching in data 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

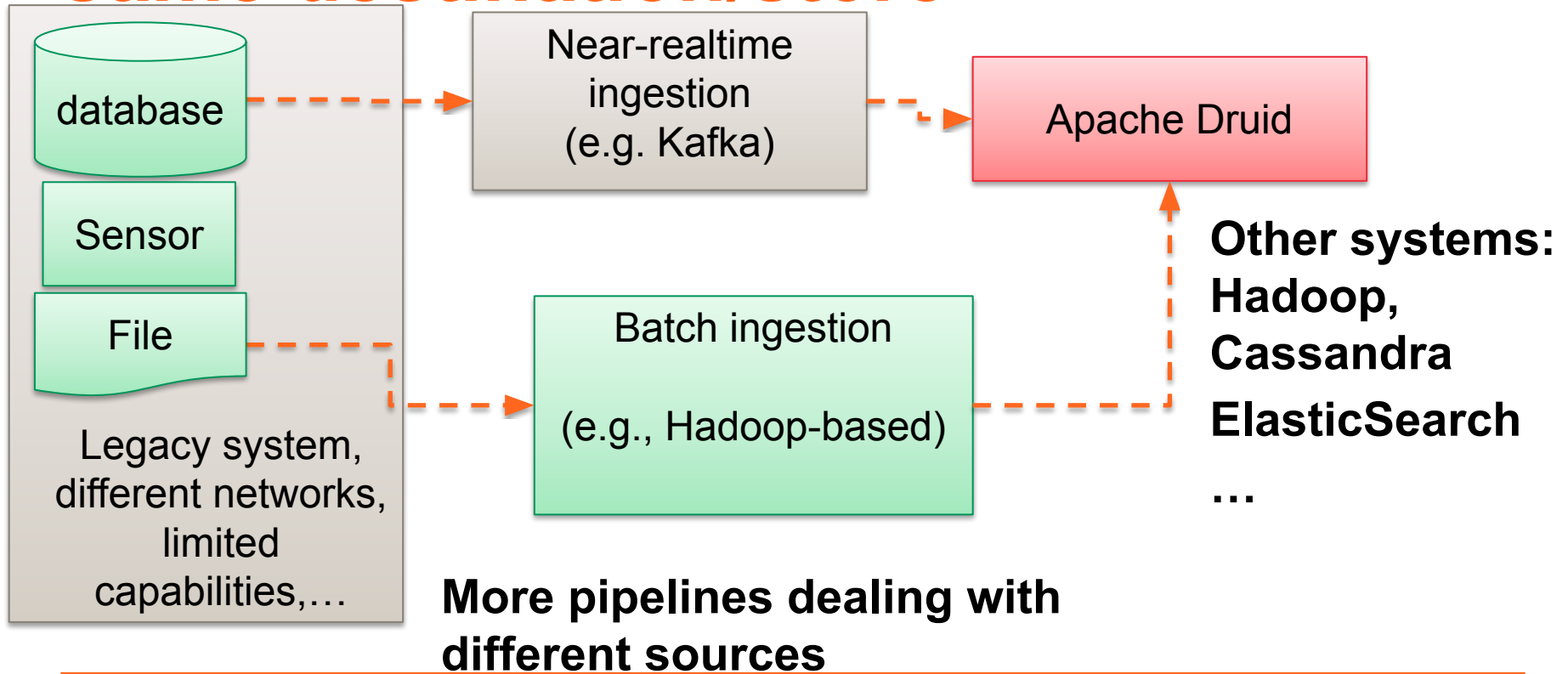


Be careful with the data semantics/integrity!

Complex ingestion pipelines in big data platforms

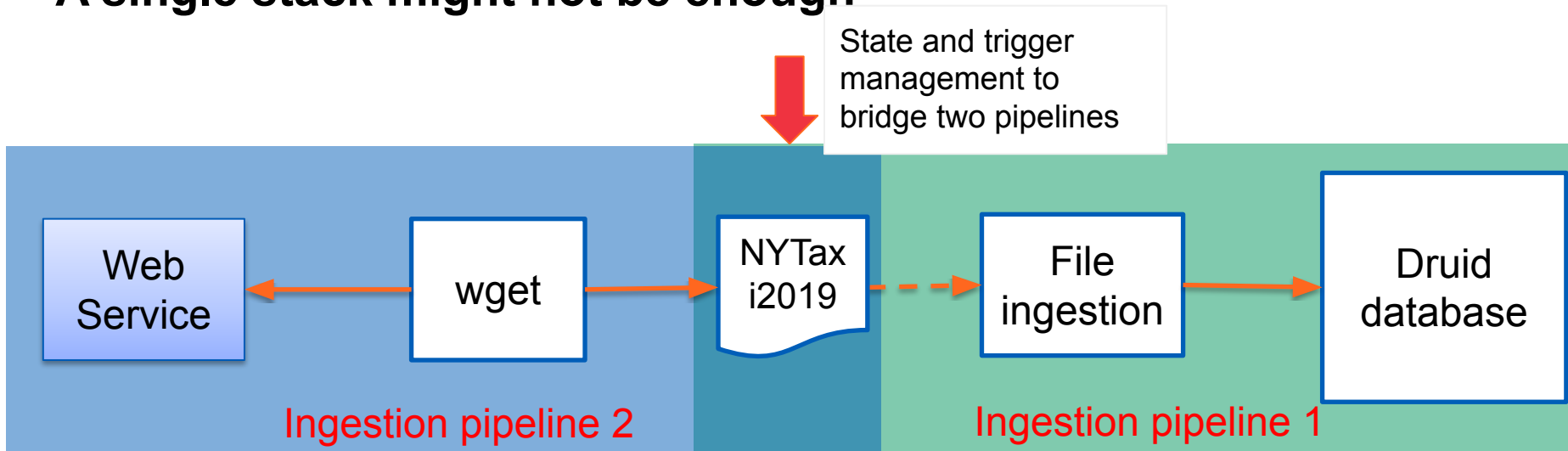
- **Multiple types of pipelines for multiple types of tenants/users**
 - **A tenant/user might need different integrated pipelines**
- ⇒ **Both batch and near-realtime ingestion are supported**
- **Complex architectural designs**
 - ingestion pipeline-to-pipeline needs “bridges”

Multiple types of pipelines for the same destination/store



Connecting different ingestion pipelines

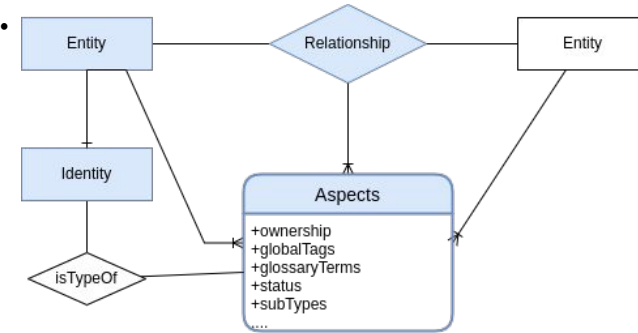
A single stack might not be enough



Real-world:
both pipelines and their connections 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

- **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:



Microsoft Presidio

YData Profiling



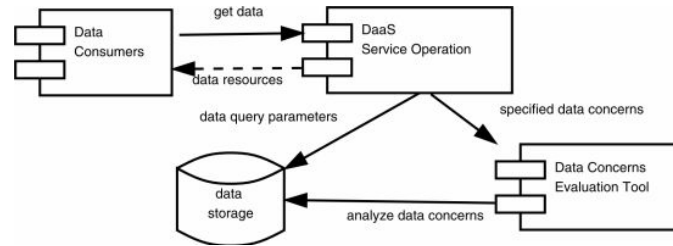
Aalto University
School of Science

CS-E4640 Big Data Platforms, Spring 2024, Hong-Linh Truong
31/01/2024

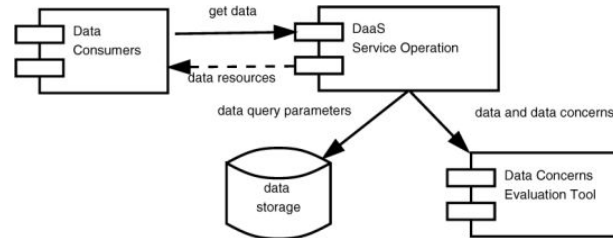
Quality control/data regulation assurance (1)

Design: different evaluation modes

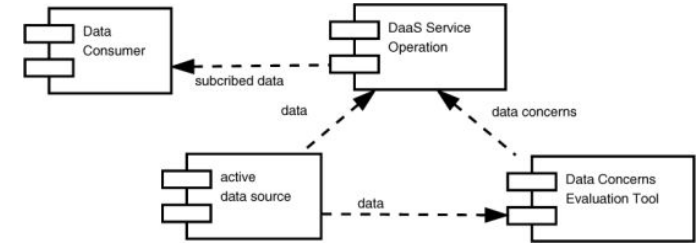
Pull,
pass-by-reference
model for
evaluating data
concerns



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



Push model for evaluating data
concerns of active data sources



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 program
- **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/>

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: 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

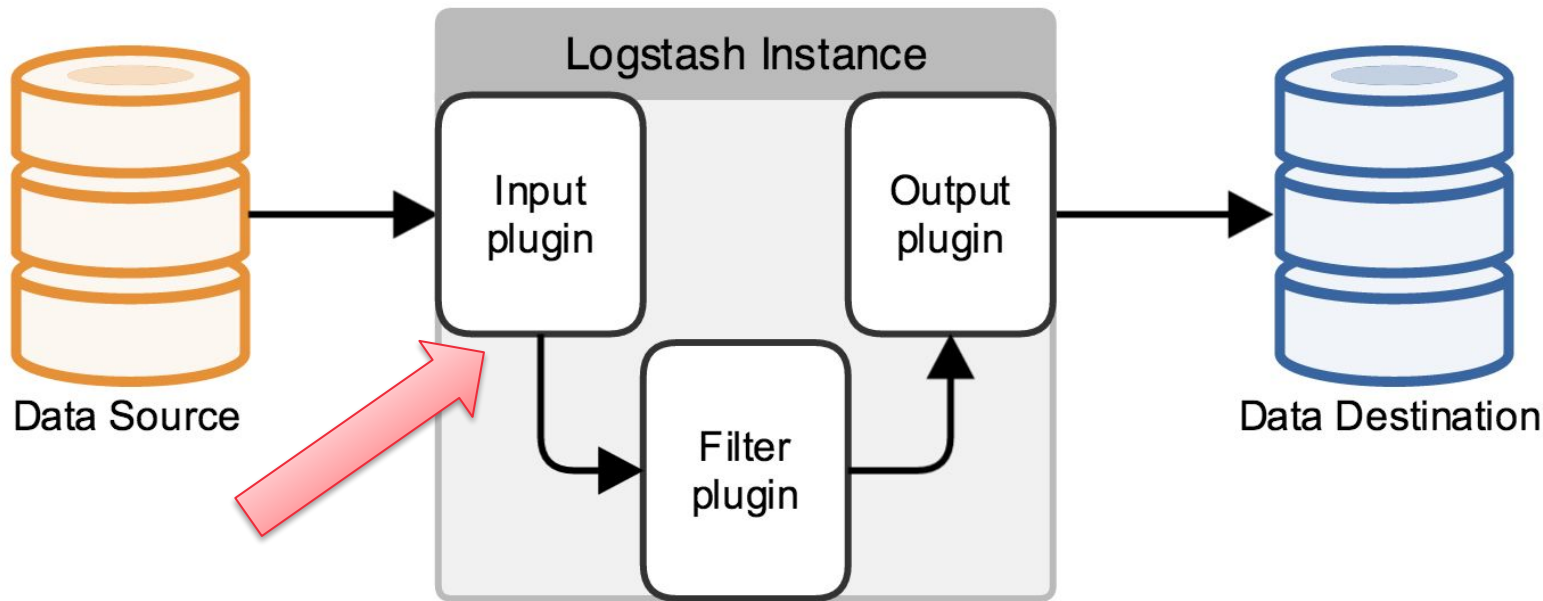


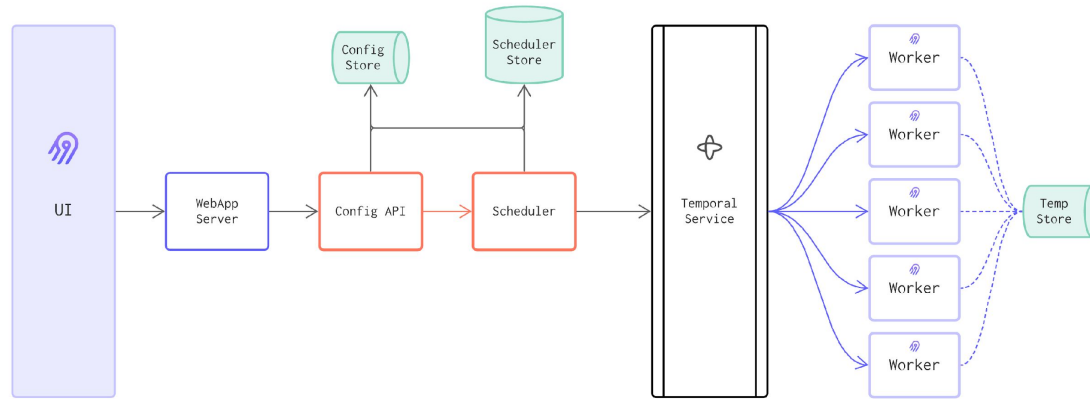
Figure source:

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

Pluggable approaches

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

Connect and parse raw data

Transform data and configure schema

Tune parameters

Verify and submit

Start

Connect

Parse data

Parse time

Transform

Filter

Configure schema

Partition

Tune

Publish

Edit spec

VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,payment_type,
2,11/04/2084 12:32:24 PM,11/04/2084 12:47:41 PM,1,1.34,1,N,238,236,2,10,0,0,5,0,0,0,3,10.8
2,11/04/2084 12:25:53 PM,11/04/2084 12:29:00 PM,1,0.32,1,N,238,238,2,4,0,0,5,0,0,0,3,4.8
2,11/04/2084 12:08:33 PM,11/04/2084 12:22:24 PM,1,1.85,1,N,236,238,2,10,0,0,5,0,0,0,3,10.8
2,11/04/2084 11:41:35 AM,11/04/2084 11:59:41 AM,1,1.65,1,N,68,237,2,12,5,0,0,5,0,0,0,3,13.3
2,11/04/2084 11:27:28 AM,11/04/2084 11:39:52 AM,1,1.07,1,N,170,68,2,9,0,0,5,0,0,0,3,9.8
2,11/04/2084 11:19:06 AM,11/04/2084 11:26:44 AM,1,1.3,1,N,107,170,2,7,5,0,0,5,0,0,0,3,8.3
2,11/04/2084 11:02:59 AM,11/04/2084 11:15:51 AM,1,1.85,1,N,113,137,2,10,0,0,5,0,0,0,3,10.8
2,11/04/2084 10:46:05 AM,11/04/2084 10:50:09 AM,1,0.62,1,N,231,231,2,4,5,0,0,5,0,0,0,3,5.3
2,07/11/2053 01:25:33 PM,07/11/2053 01:25:33 PM,1,0,1,N,264,264,2,0,0,0,0,0,0,0,0
2,12/04/2042 08:51:43 AM,12/04/2042 08:54:47 AM,1,0.29,1,N,162,162,2,4,0,0,5,0,0,0,3,4.8
2,06/25/2041 08:46:37 PM,06/25/2041 08:52:37 PM,1,1.34,1,N,239,151,2,7,0,5,0,5,0,0,0,3,8.3
2,11/17/2037 09:24:28 PM,11/17/2037 09:46:03 PM,1,2.99,1,N,170,143,1,15,0,5,0,5,1,7,0,0,3,18
2,02/02/2032 12:39:23 PM,02/02/2032 01:11:39 AM,4,23,21,1,N,132,228,2,62,0,5,0,5,0,0,0,3,63.3
2,02/13/2031 05:36:35 PM,02/13/2031 05:45:36 PM,1,1.44,1,N,236,237,2,8,1,0,5,0,0,0,3,9.8
2,02/13/2031 05:21:28 PM,02/13/2031 05:35:36 PM,1,1.69,1,N,141,236,2,8,1,0,5,0,0,0,3,9.8
2,05/06/2029 08:43:14 PM,05/06/2029 09:03:14 PM,4,4,47,1,N,162,80,1,17,5,0,5,0,5,4,91,5,76,0,3,29.47
2,05/05/2029 11:22:18 PM,05/06/2029 02:02:00 AM,1,11,51,1,N,148,244,1,34,5,0,5,0,5,0,0,3,35.8
2,02/13/2026 11:53:54 AM,02/13/2026 11:58:02 AM,2,0.85,1,N,161,43,2,5,1,0,5,0,0,0,3,6.8
2,02/13/2026 11:06:18 AM,02/13/2026 06:26:09 PM,2,3,14,1,N,163,246,2,20,1,0,5,0,0,0,3,21.8
2,09/13/2021 12:19:52 PM,09/13/2021 12:22:07 PM,1,0,1,N,193,193,2,0,0,0,0,0,0,0,0
2,12/10/2020 08:34:26 PM,12/10/2020 08:54:46 PM,1,4.62,1,N,50,231,2,17,5,0,5,0,5,0,0,3,18.8
2,12/10/2020 08:23:43 PM,12/10/2020 08:32:35 PM,1,2.44,1,N,90,50,1,9,0,5,0,5,2,06,0,0,3,12.36
2,08/01/2020 12:20:58 AM,08/01/2020 12:47:09 AM,1,16.71,1,N,143,138,1,45,5,0,5,0,5,10,41,5,76,0,3,62.47
2,08/01/2020 12:07:04 AM,08/01/2020 12:20:28 AM,1,2,3,1,N,238,143,2,11,0,0,5,0,0,0,3,11.8
2,03/05/2020 06:44:16 PM,03/06/2020 03:14:32 PM,1,2,39,1,N,125,161,2,11,0,0,5,0,0,0,3,11.8
2,03/05/2020 06:33:57 PM,03/05/2020 06:40:39 PM,1,1,04,1,N,231,125,1,6,5,0,5,1,46,0,0,3,8.76

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

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

[Learn more](#)

Source type

local

Base directory

/opt/data/rawdata/bdp

File filter

*.csv

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

Apply

A Aalto University
School of Science

CS-E4640 Big Data Platforms, Spring 2024, Hong-Linh Truong
31/01/2024

55

Design tools for ingestion processes: Apache Nifi

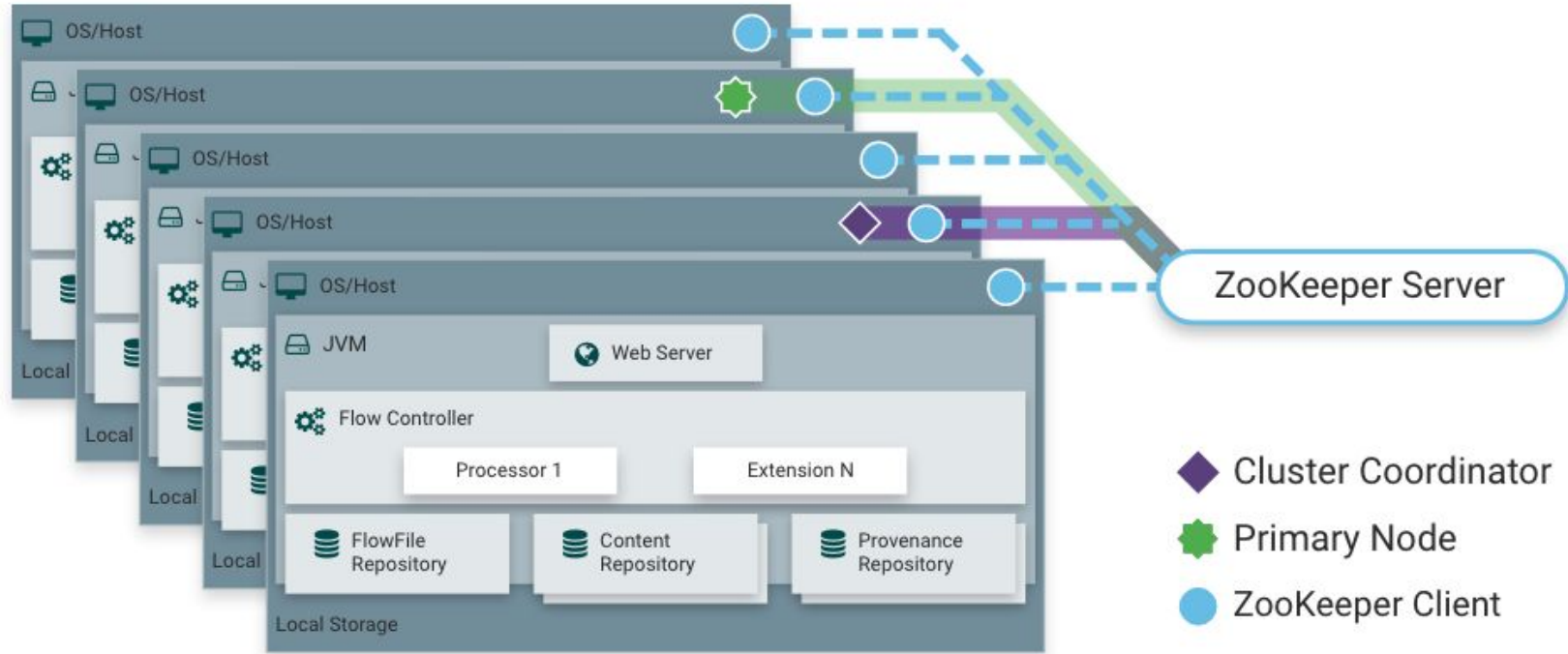


Figure source: <https://nifi.apache.org/docs.html>

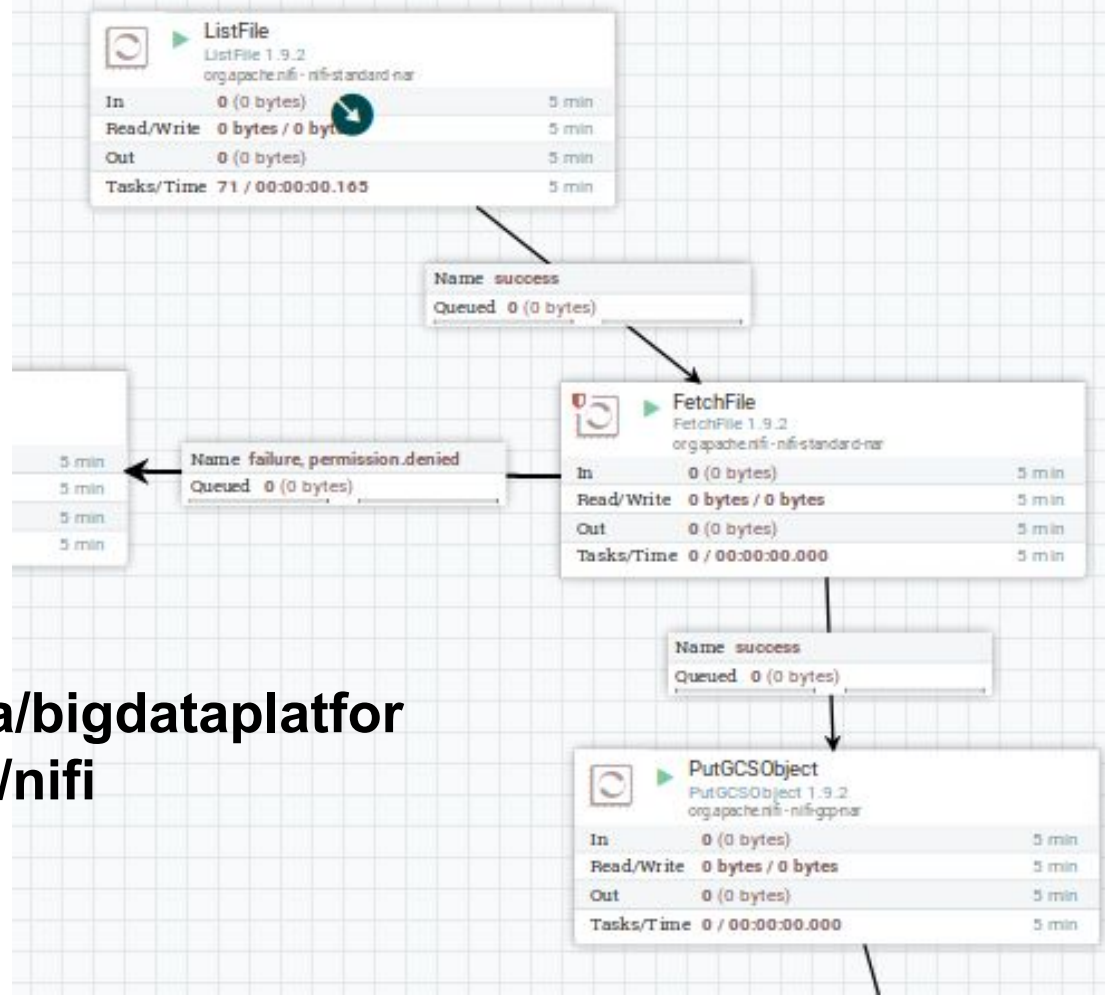
Design tools for ingestion processes: Apache Nifi - key concept

- 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/bigdataplatfor.ms/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
- **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