Aalto University
School of Science

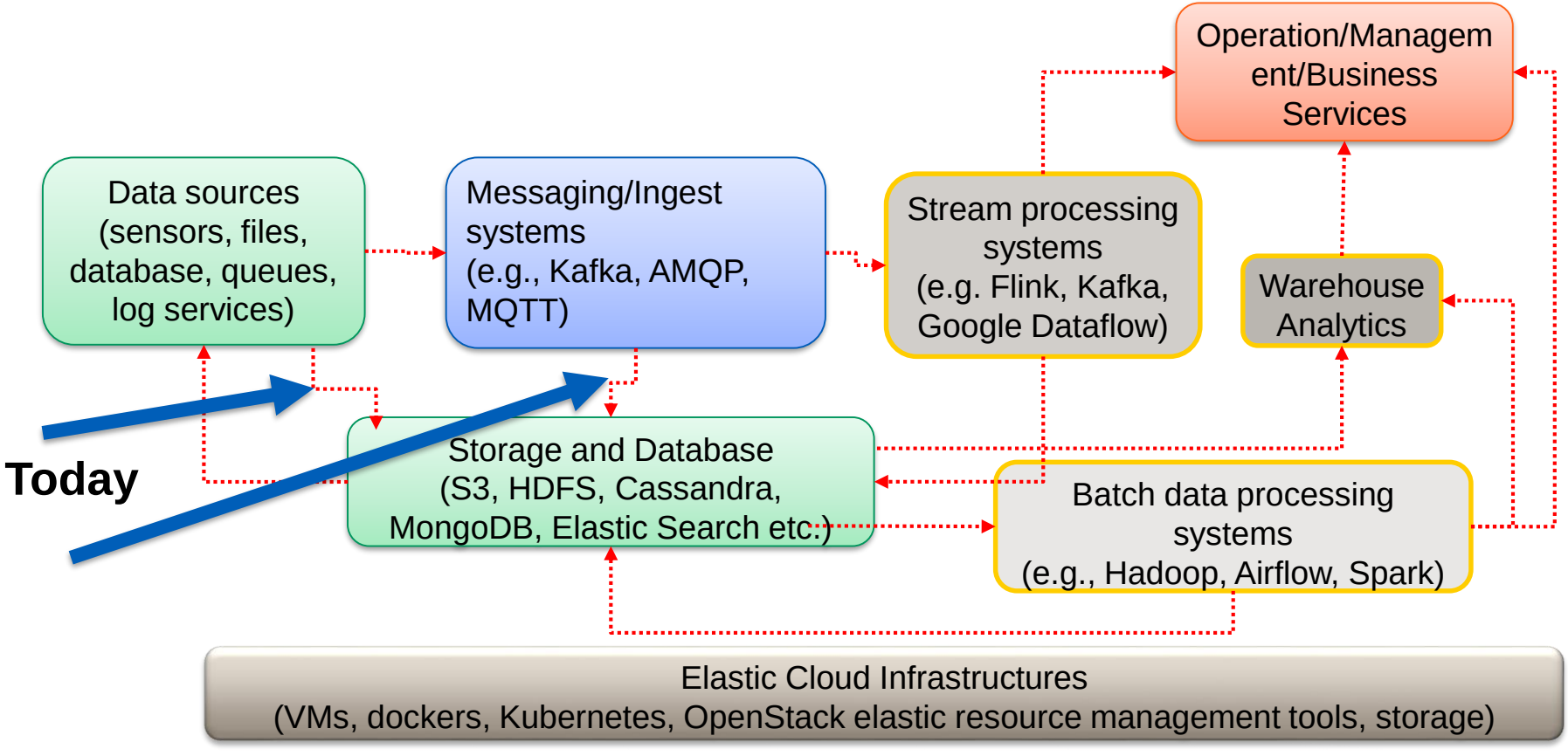
Big Data Ingestion

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

Schedule

- Overview
- Tasks in data ingestion
- Ingestion processes: architectural designs
- Examples of tools

Big data at large-scale



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**

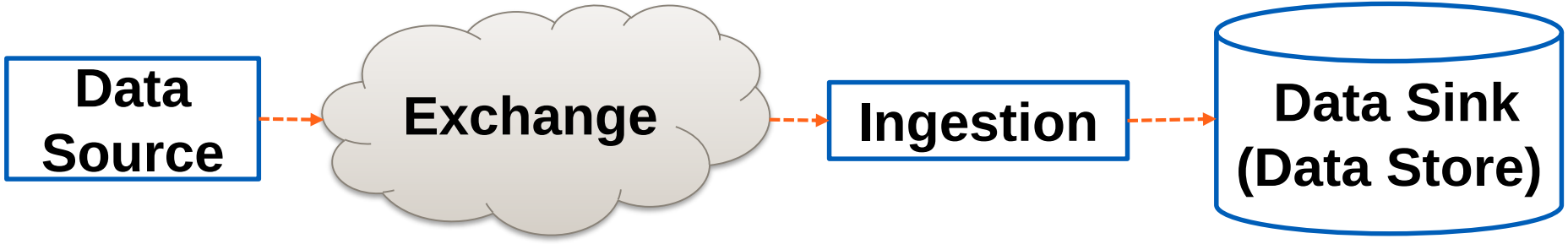
Platform versus end-user views

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!

Syntax and semantic problems



- Ingestion might not understand the semantics of the data
  - The same communication protocol does not mean that both sides understand the message well!

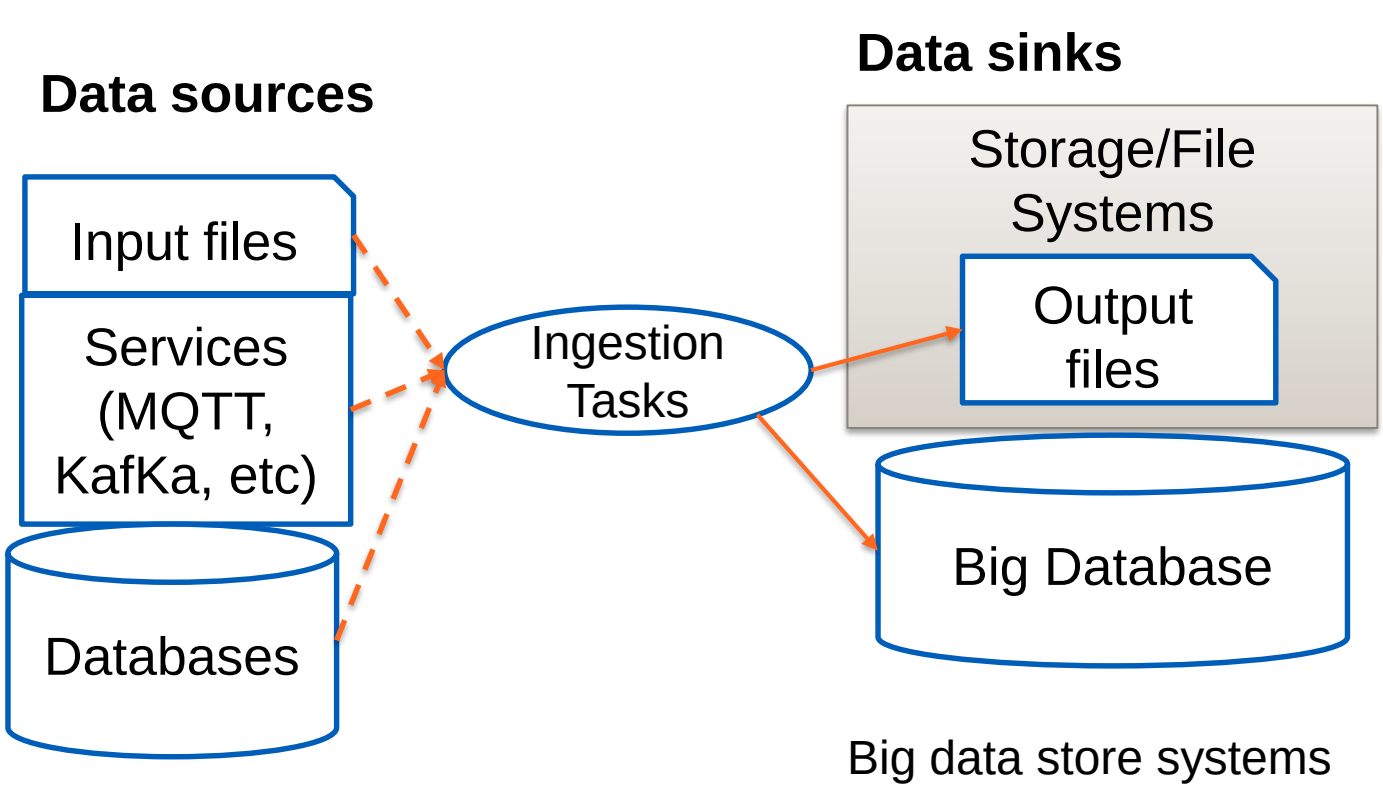
Fundamental ingestion models

- **Batch ingestion**
  - Data is in files
  - Ingestion can be done batches
- **(Near) real-time ingestion**
  - Data is encapsulated into messages
  - Ingest data as soon as the data is available
  - Message brokers are needed

Common Data Formats

- **Files**
  - CSV, Text, JSON, ARVO
  - Other typical formats
- **Messages (unit of information which is self-contained)**
  - Text/CSV/JSON
  - ARVO
  - Other forms

Data source and sinks



Big data platform examples:

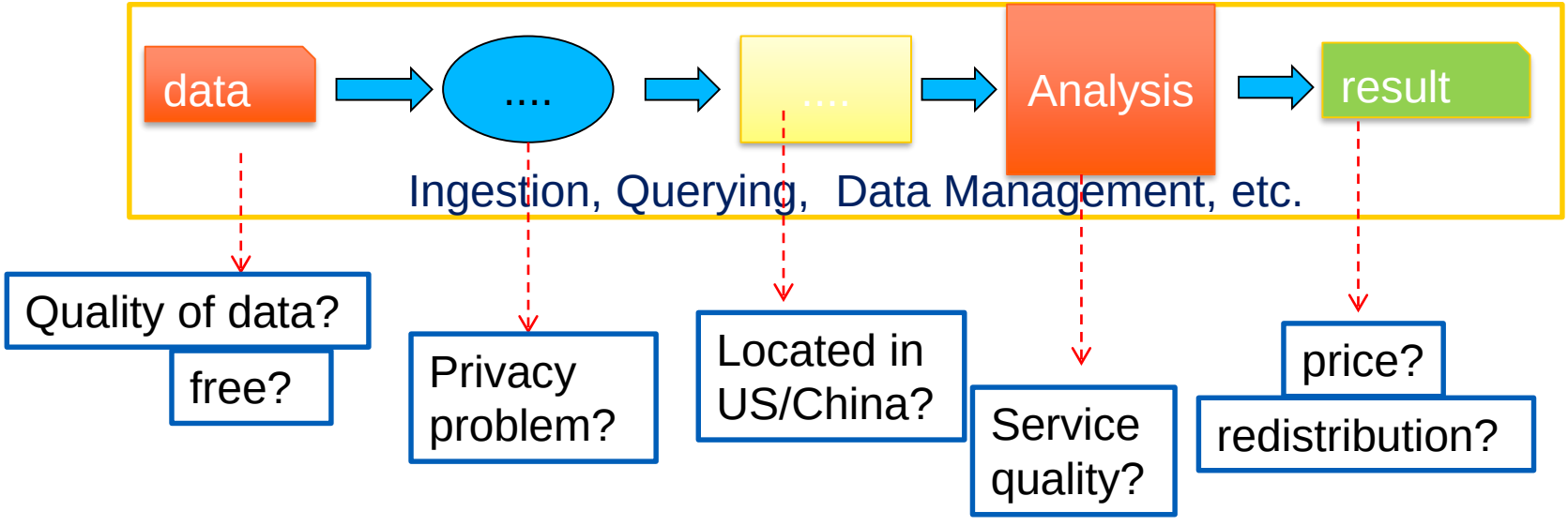
Hadoop File systems
Google Storage
Amazon Storage

Google BigQuery
Hive
MongoDB
ElasticSearch

Requirements from V* of big data

- **Requirements from access and protocols**
  - *REST API, Databases, other big data platforms, File, SFTP, …*
  - *Access APIs: http, file, ftp, etc.*
- **Requirements from data**
  - Structured, unstructured and semi-structured
  - speed, volume, accuracy, confidentiality, data regulation
- **How deep you can support?**
  - Are you able to go into inside of data elements (understanding the syntax and semantics of data)

Recall: data concerns



- Ethical consequence?
- Regulation-compliant platforms: e.g., GDPR

Main tasks in ingestion

- **Some key tasks**
  - Data access and extraction
  - Data routing
  - Data wrangling
- **The key point is to distinguish which tasks the user has to do, which will be done by the platform**

“Poor” platforms: only give “here is the API”

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

Dealing with message structures

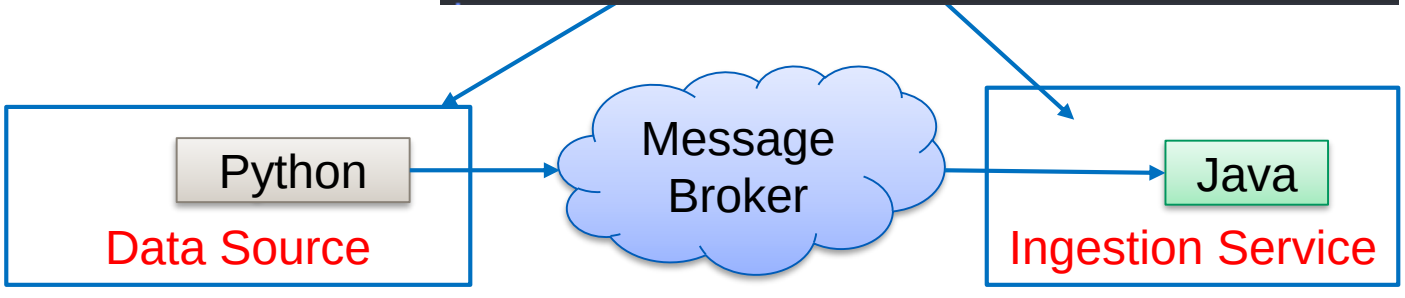
- **Remember that the data sender and the receiver are diverse**
  - In many cases, they are not in the same organization
    - *You need to guarantee the message syntax and semantics*
- **Solutions**
  - Agreed in advance → in the implementation or with a standard
  - Know and use tools to deal with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

Design question: does the platform need to impose message structures if it just provides "broker"

Example: Arvo

Syntax specification

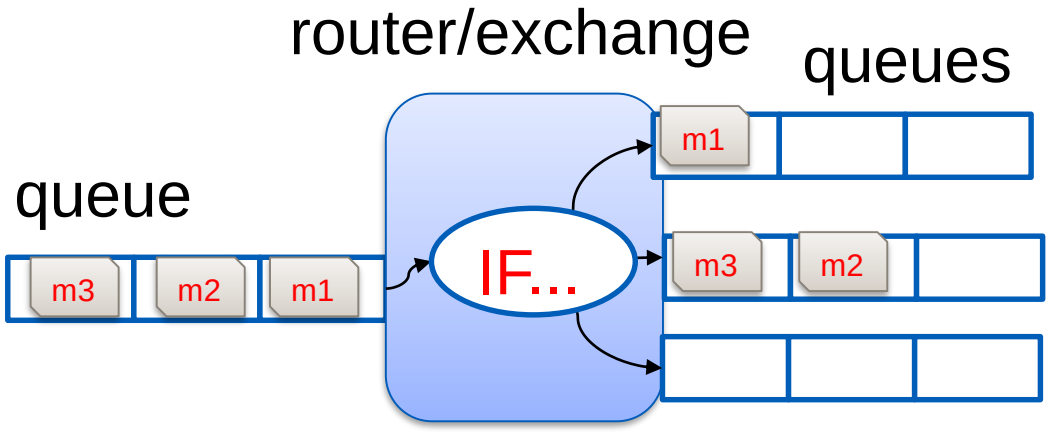
```
{
        "namespace": "bdp.courses.aalto.fi",
        "type": "record",
        "name": "event",
        "fields": [
            {"name":"station_id", "type":"string"},
            {"name":"datapoint_id", "type":"int"},
            {"name":"alarm_id", "type":"int"},
            {"name":"event_time",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

        ]
}
```



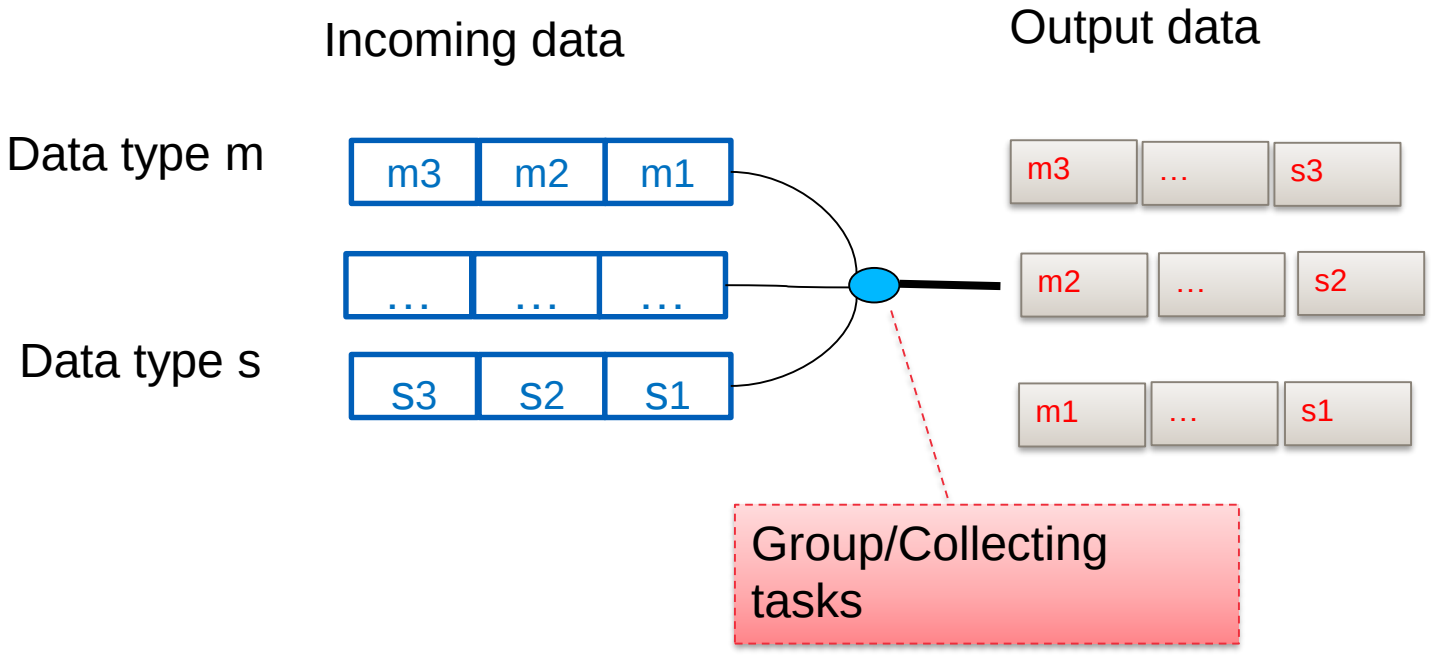
Some other techniques

- **Protobuf**
  - From Google, used by default in gRPC (gRPC.io)
  - https://github.com/google/protobuf
  - Language-neutral, platform-neutral mechanism for serializing/deserializing structured data
- **Thrift**
  - https://thrift.apache.org
  - RPC style
  - Support also serializing and deserializing data)
  - Support cross-language services development
    - *Specify services interfaces*
    - *Data exchange*
    - *Code generation*
  - **Flatbuffers**
    - https://github.com/google/flatbuffers

Data routing: split tasks/distributor pattern



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']];
                    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)
```

Examples: Logstash Grok – a kind of domain specific language?

Grok is for parsing unstructured log data

text patterns into something that matches your logs.

Grok pattern syntax: %{SYNTAX:SEMANTIC}

Regular and custom patterns

A lot of exiting patterns:

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

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

Example with NETACT Log

29869;10/01/2017 00:57:56;;Major;PLMN-PLMN/BSC-xxxxxx/BCF-xxx/BTS-xxx;XYZ01N;ABC08;DEF081;BTS OPERATION DEGRADED;00 00 00 83 11 11;Processing

Simple Grok

```
input {
file {
  path => "/tmp/alarmtest2.txt"
  start_position =>"beginning"
}
}
filter {
  grok {
     match => {"message" => "%{NUMBER:AlarmID};%{DATESTAMP:Start};%{DATESTAMP:End};%{WORD:Severity};%{NOTSPACE:NetworkType};%{NOTSPACE:BSCName};%{NOTSPACE:Sta
  }
}
output {
stdout {}
 csv {
     fields =>['AlarmID','Start','Stop','Severity','NetworkType','BSCName','StationName','CellName','AlarmInfo','Extra','AlarmStatus']
    path => "/tmp/test-%{+YYYY-MM-dd}.txt"
}
}
```

Short summary of ingestion tasks

- **Basic tasks but in the context of big data**
- **Distinguish between platform tasks and end-user tasks**
  - Platform enables the user to do many tasks
  - Will a platform act on-behalf of the user?
- **Enable pluggable approaches is important**
  - Input data plugin/component → filter/extract/convert → output data plugin/component
- **Both programming and configuration are needed for building piplelines**

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

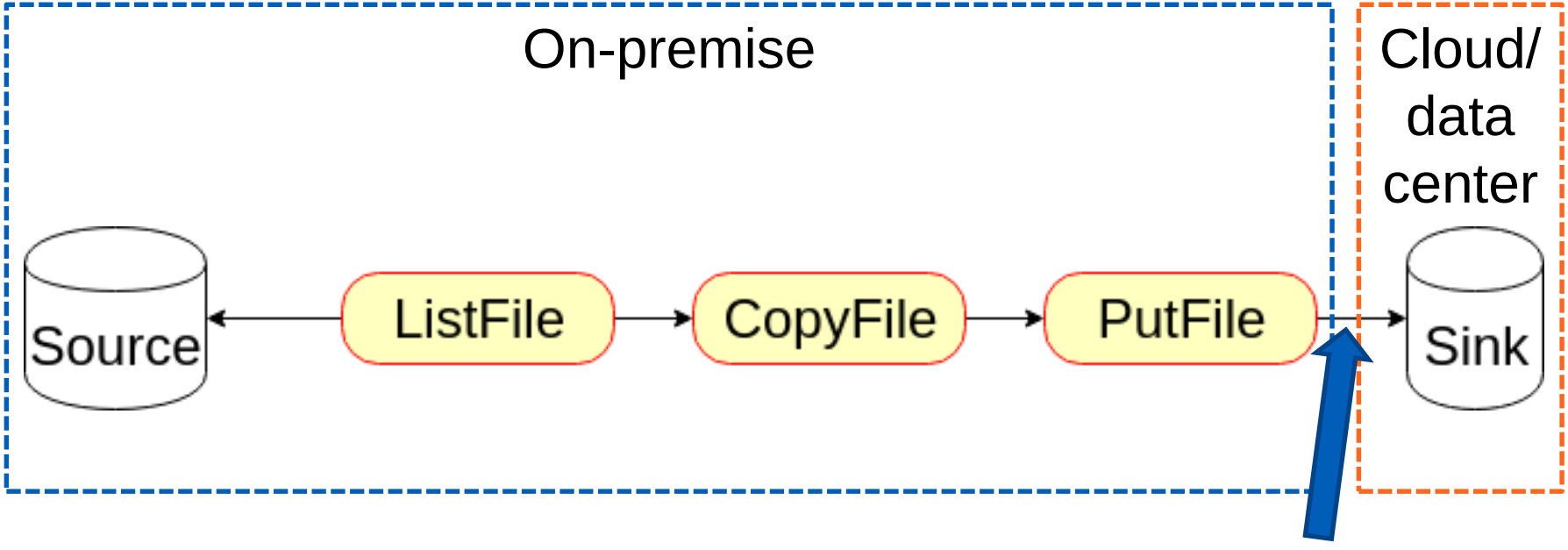
- **Understanding strong dependencies between protocols/APIs, security, deployment and managed services**

Customer

Ingestion pipeline developer (for whom?)

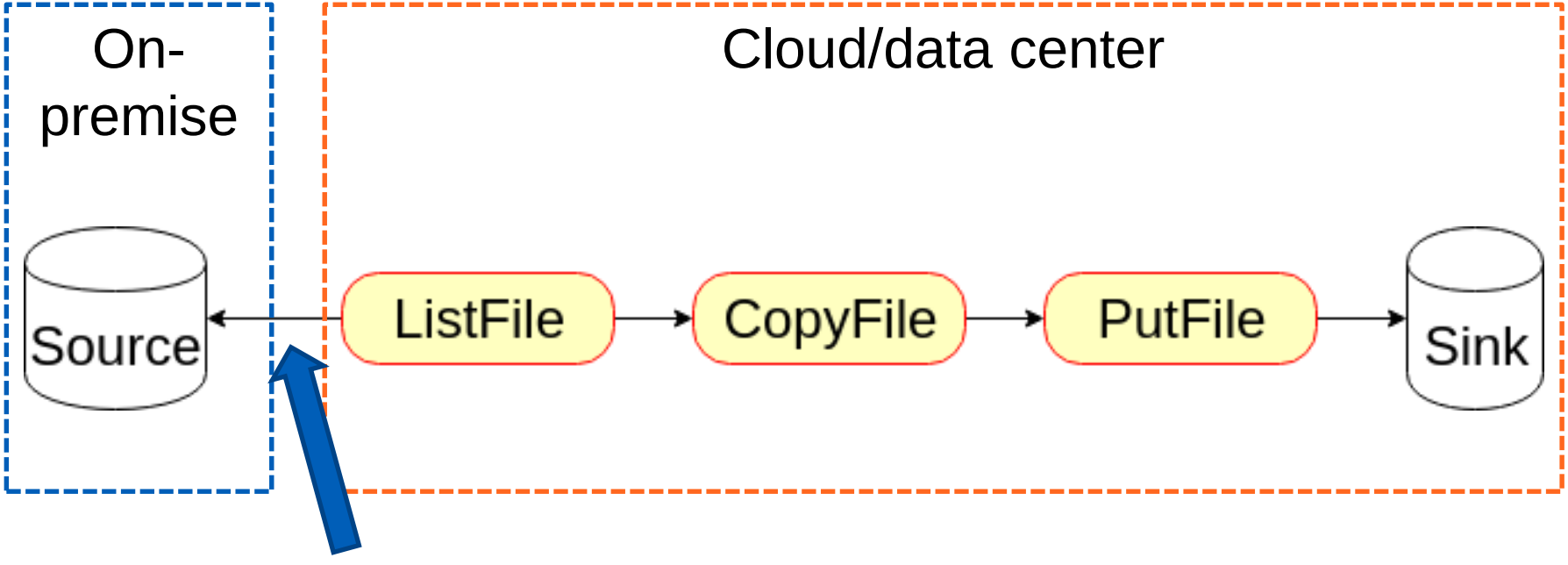
Data store/platform provider

Complex deployment and integration models



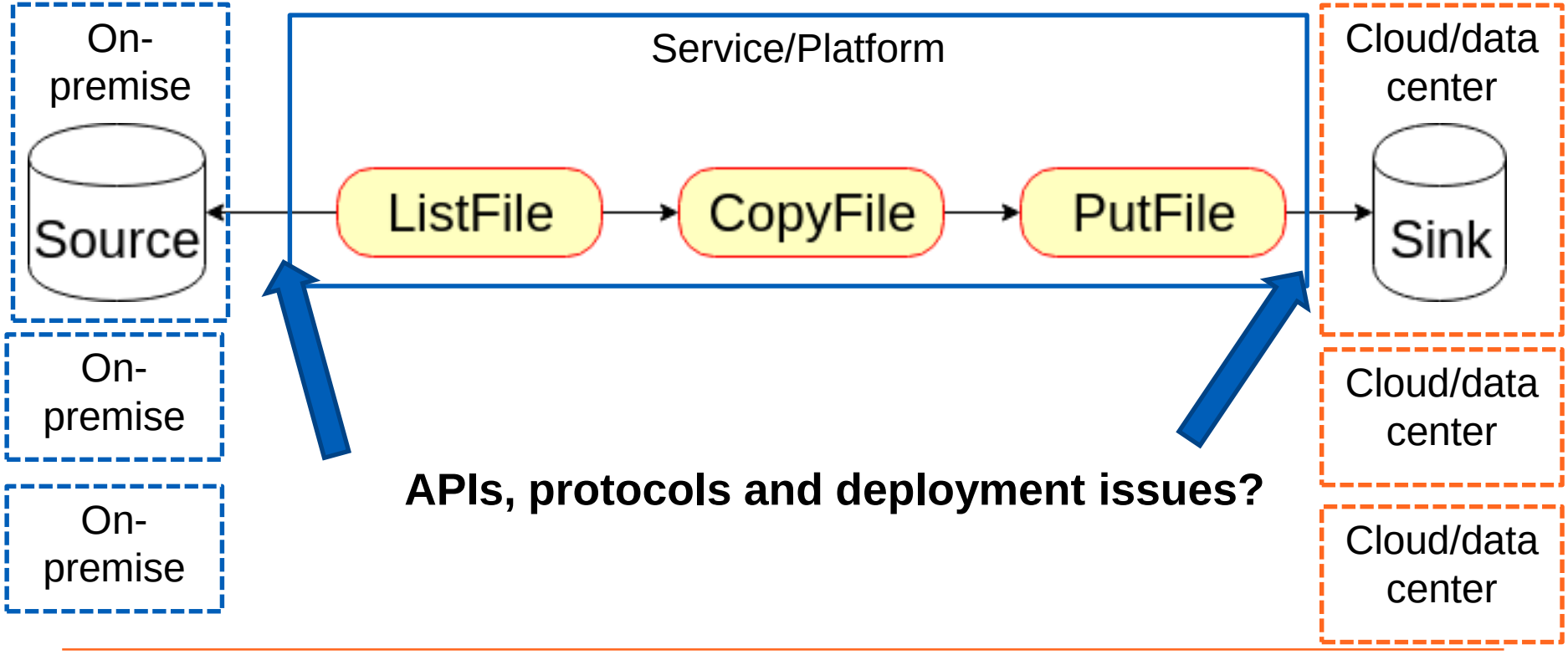
APIs, protocols and deployment issues?

Complex deployment and integration models



APIs, protocols and deployment issues?

Complex deployment and integration models



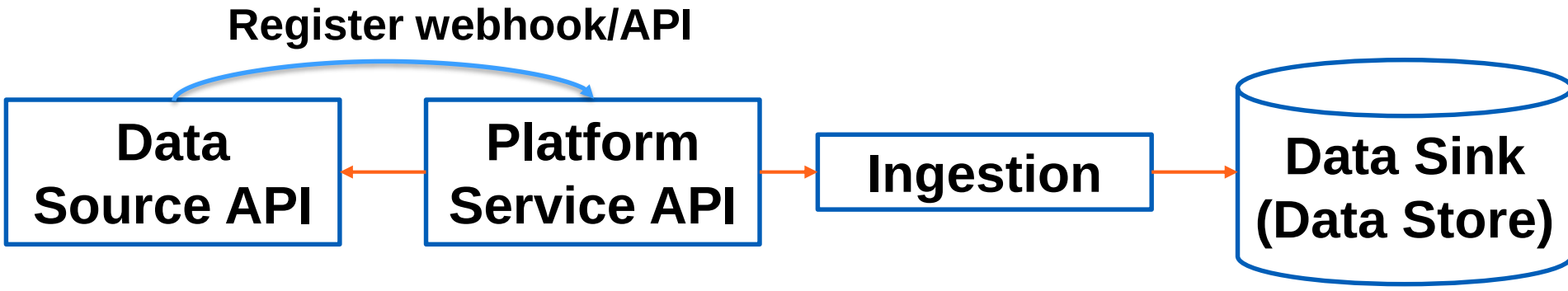
Remember:

All the issues discussed in previous slides are valid for batch and near-realtime ingestion

Batch ingestion

- **Data to be ingested is bounded**
  - files or messages are finite
- **Ingestion architectural styles**
  - Simple APIs versus reactive pipelines versus 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

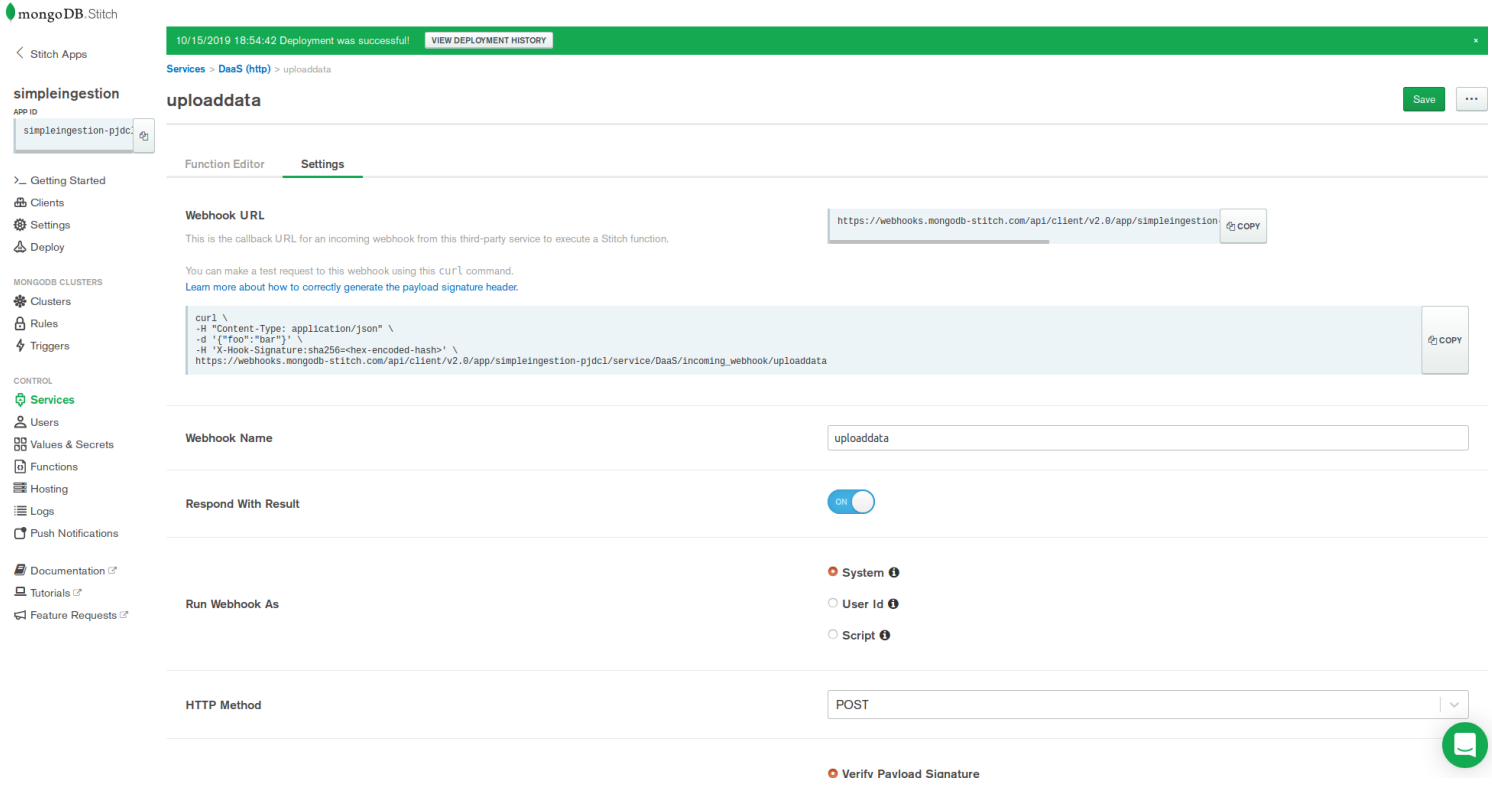
Simple APIs for ingestion



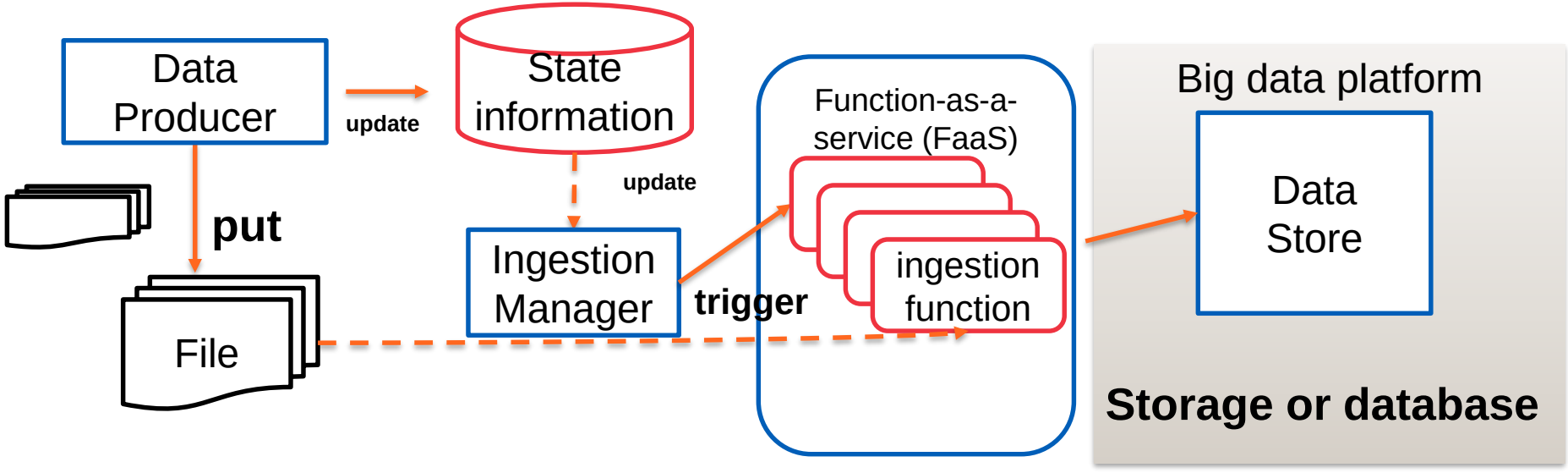
Pros and cons?

But also When?

Example with Stitch in MongoDB



Reactive with function-as-a-service

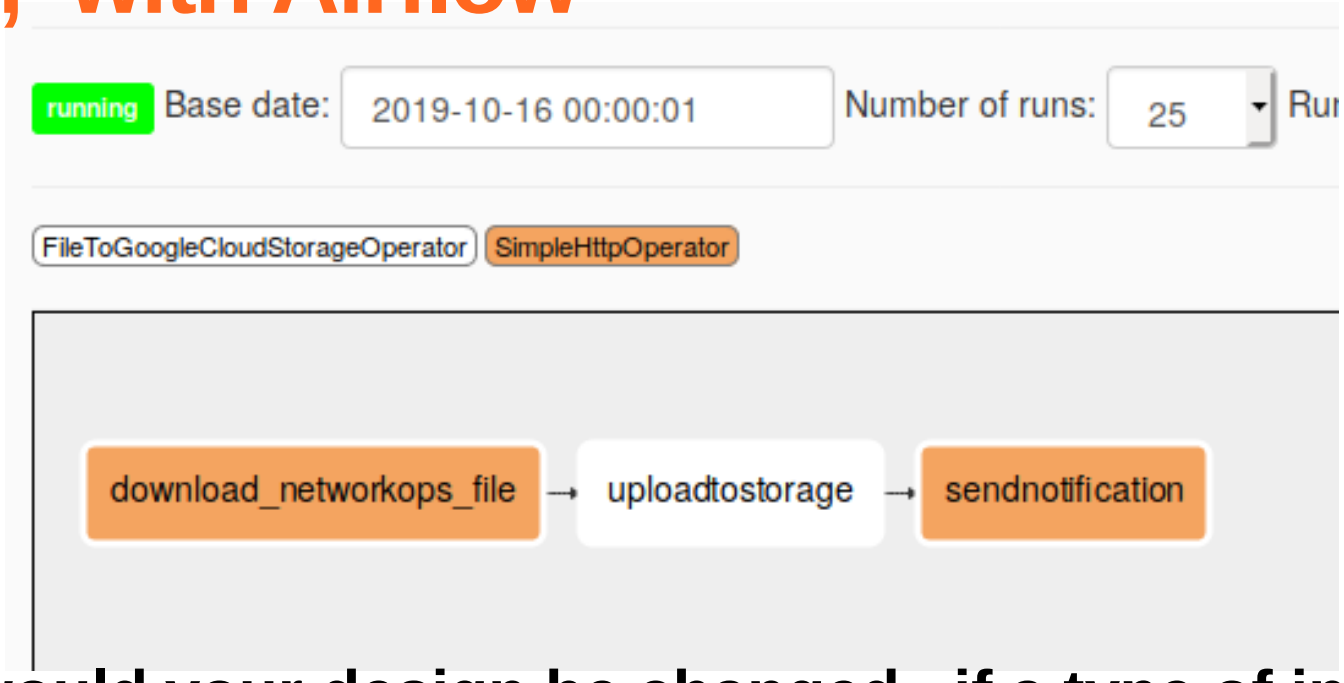


Who develops which components?

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**

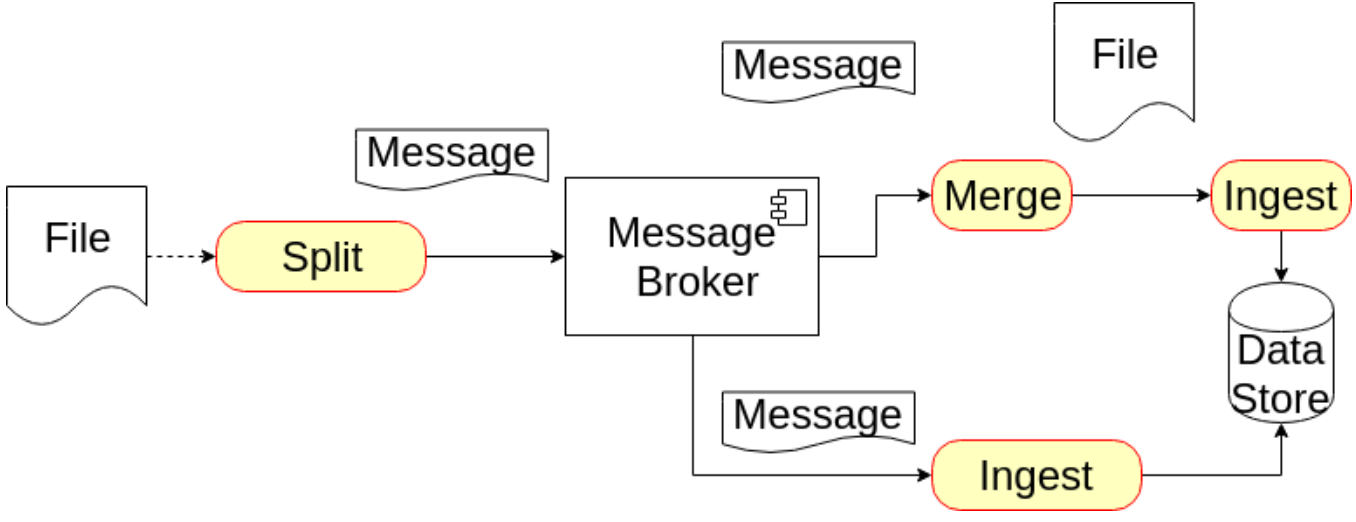
E.g., workflow based on scheduled time, with Airflow



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

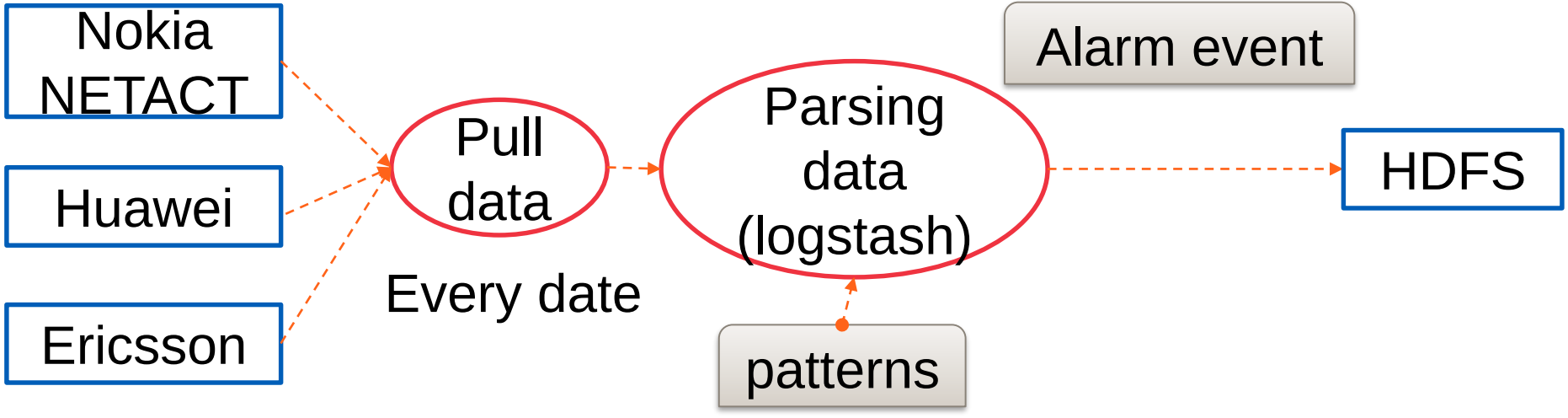
Microbatching for ingestion

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



Microbatching is useful for applying filter and quality control

Example



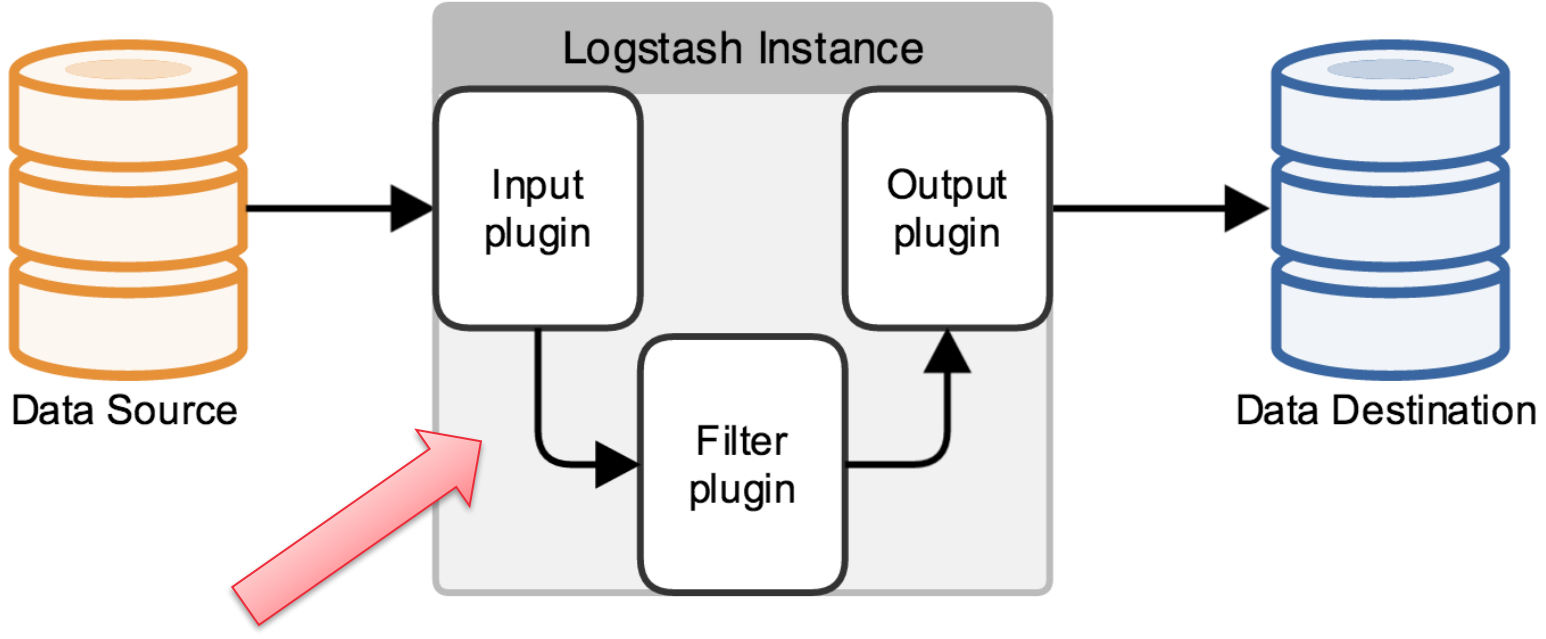
Telco devices

Tools: 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: Logstash

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



Pluggable approaches

Apache Nifi

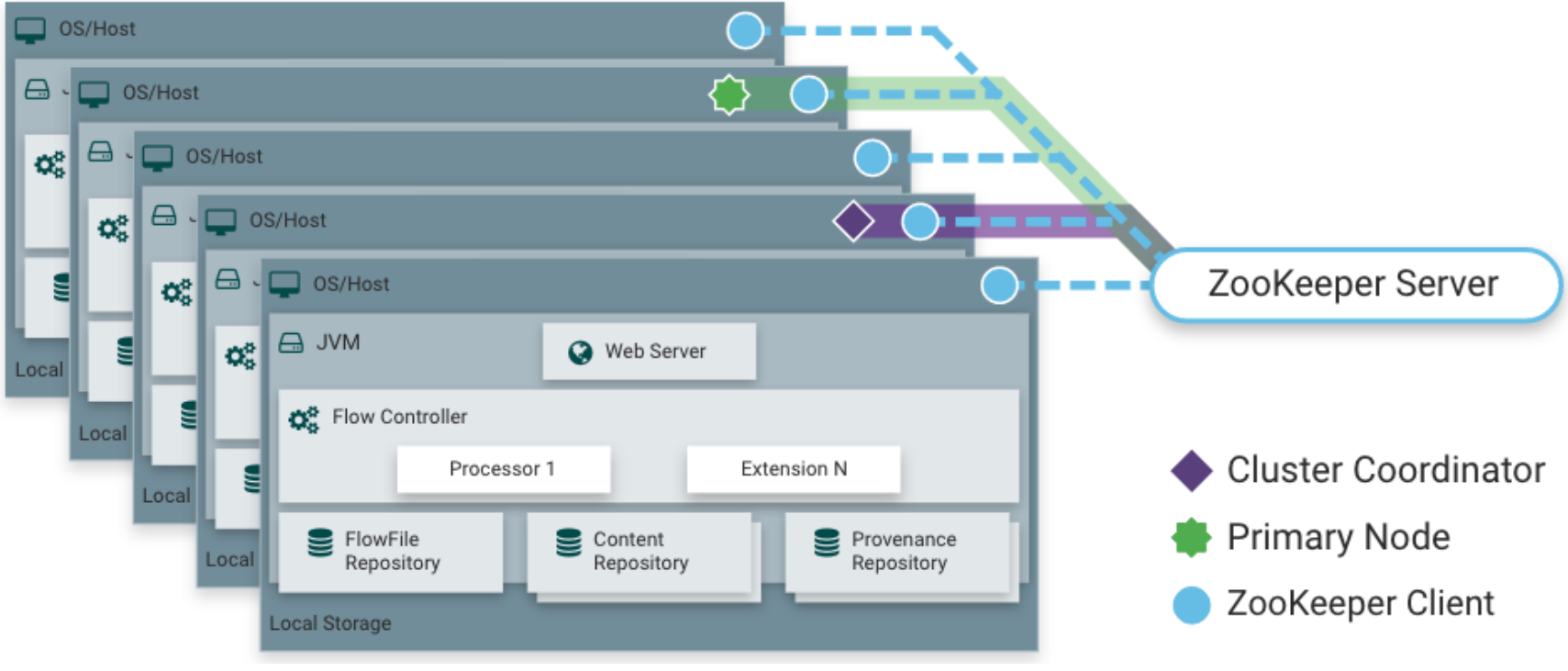


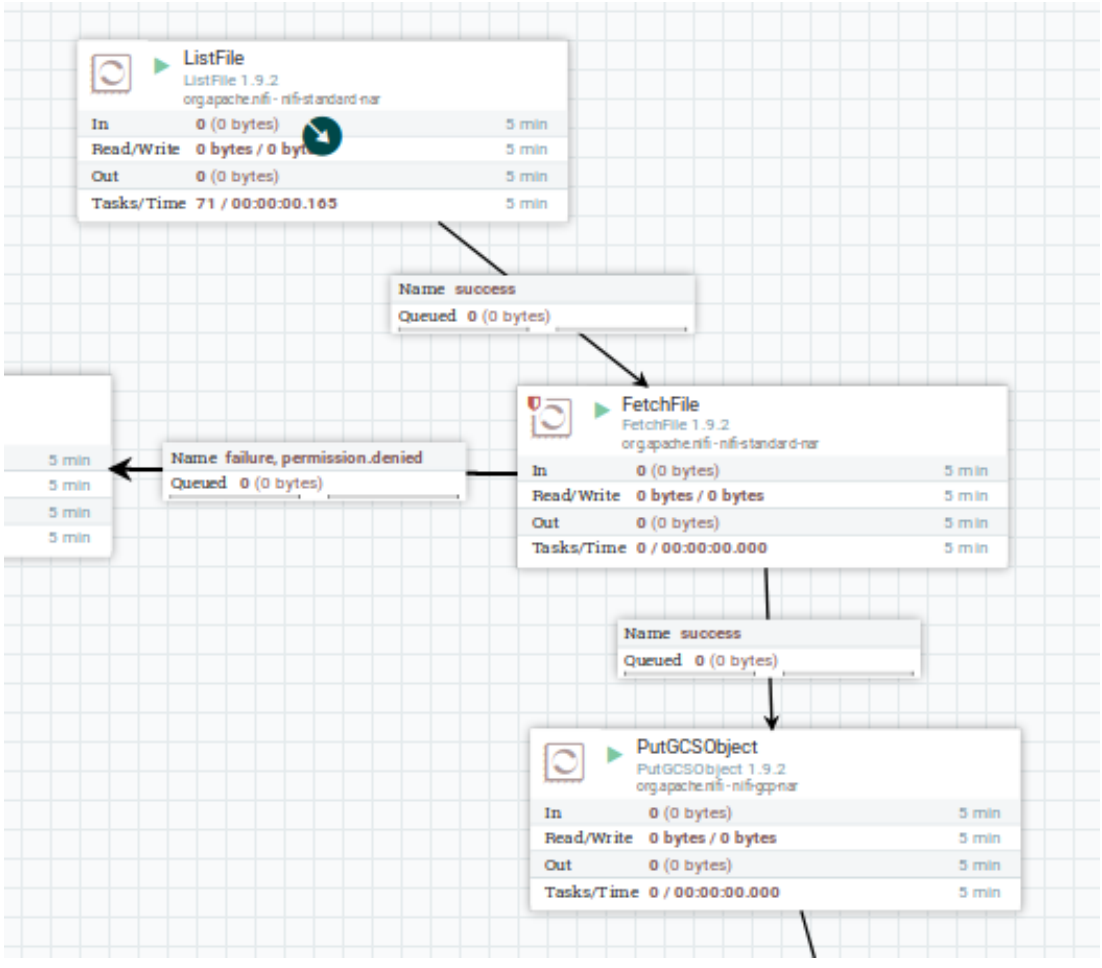
Figure source: https://nifi.apache.org/docs.html

Apache Nifi - key concept

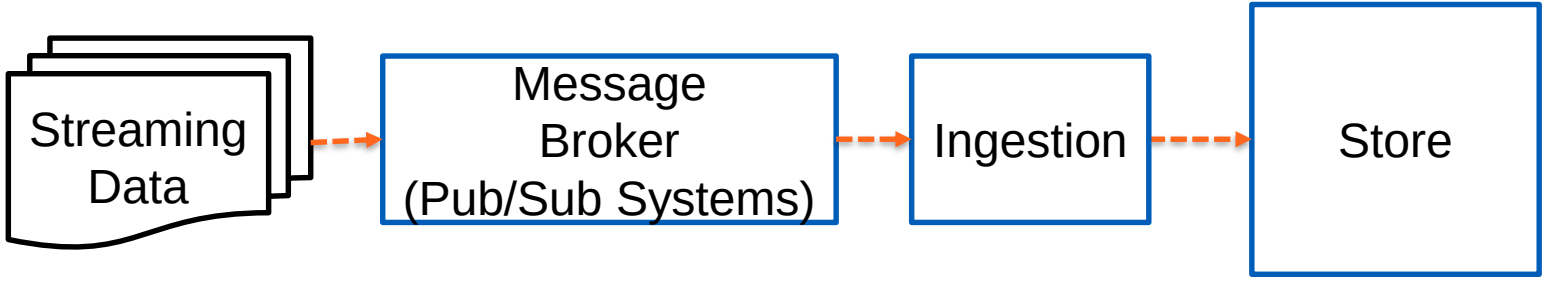
- **Data is encapsulated into "FlowFile"**
- **Processor (Component) performs tasks**
- **Processor handle FlowFile and has different states**
  - Each state indicates the results of processing that can be used for establishing relationships to other components
- **Processors are connected by Connection**
- **Connection can have many relationships based on states of upstream Processors**

Example

Let us do some practice tomorrow!

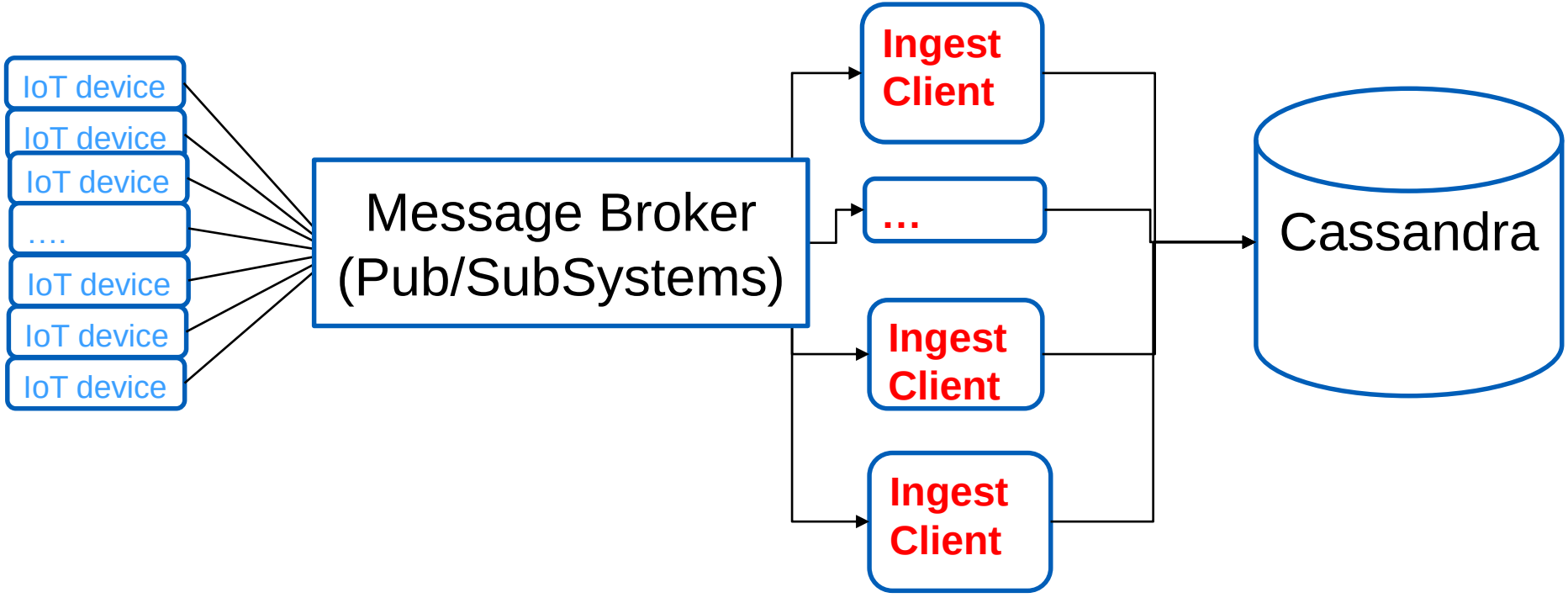


Near-real time ingestion: how do I move streaming data into the cloud?

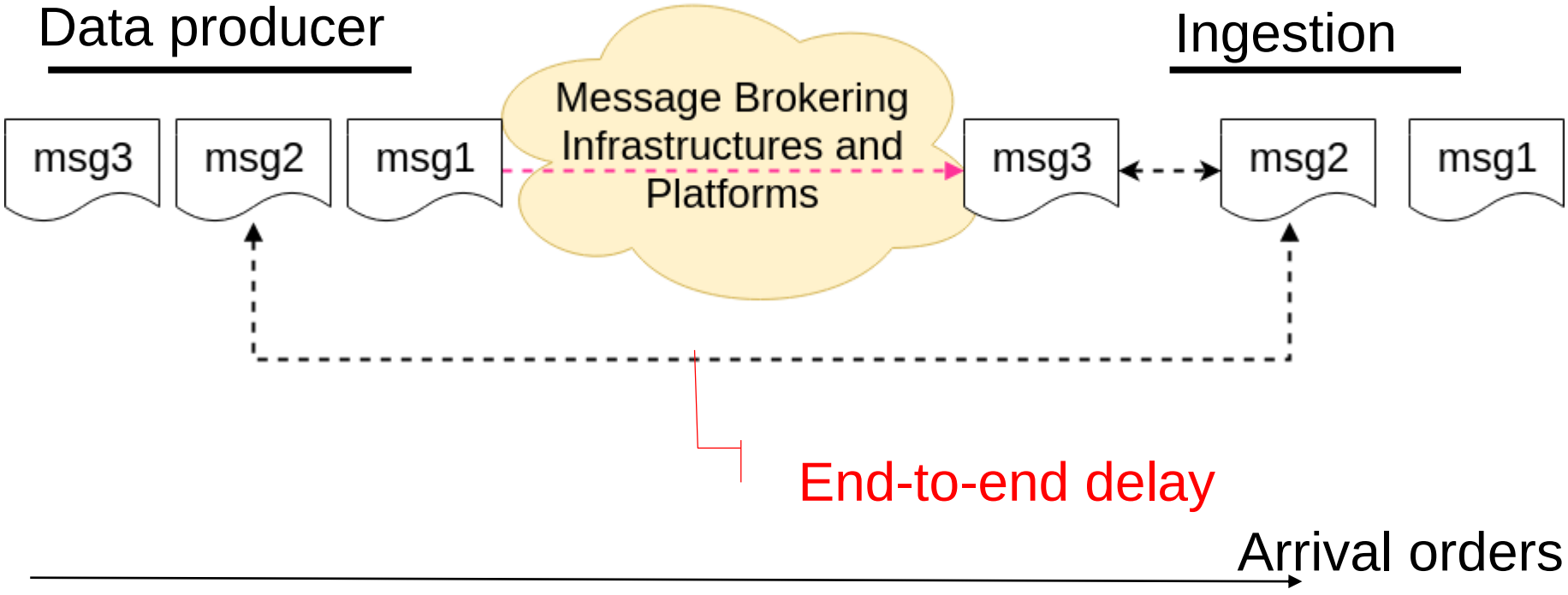


- **Unbounded data, amount of data varies, fast ingestion**

Example


IoT device
IoT device
IoT device
....
IoT device
IoT device
IoT device
Message Broker
(Pub/SubSystems)
Ingest
Client
...
Ingest
Client
Ingest
Client
Cassandra

Key issues in streaming data ingestion



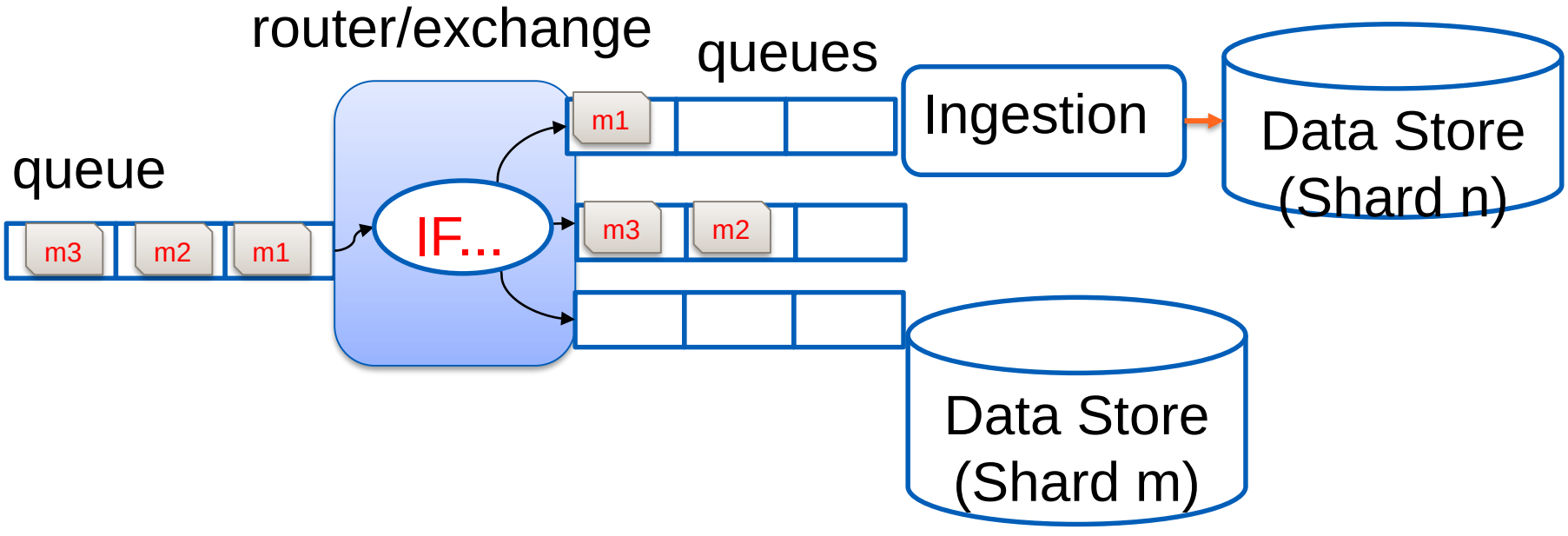
Some key issues

- **Late data, data out of order?**
- **Exactly once?**
- **Back pressure and retention**
  - For individual components or the whole pipelines
- **Scalability and elasticity**
  - Changes in stream data can be unpredictable

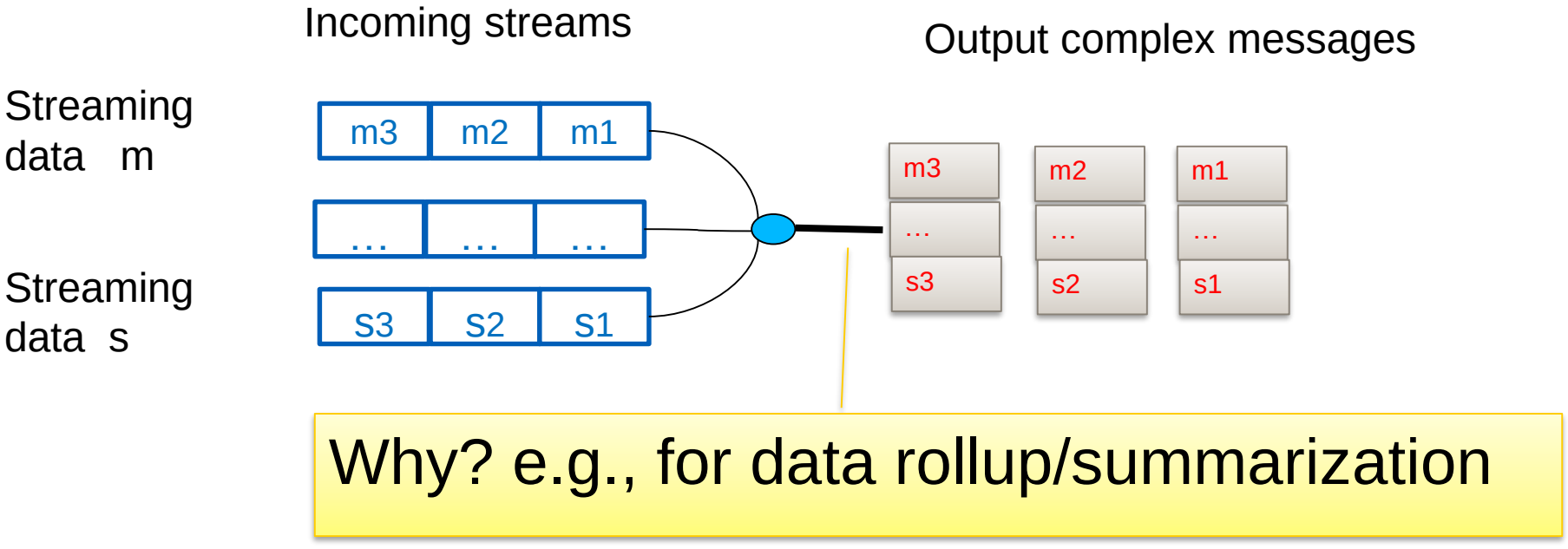
Some key issues

- **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 elasticity can be applied?**

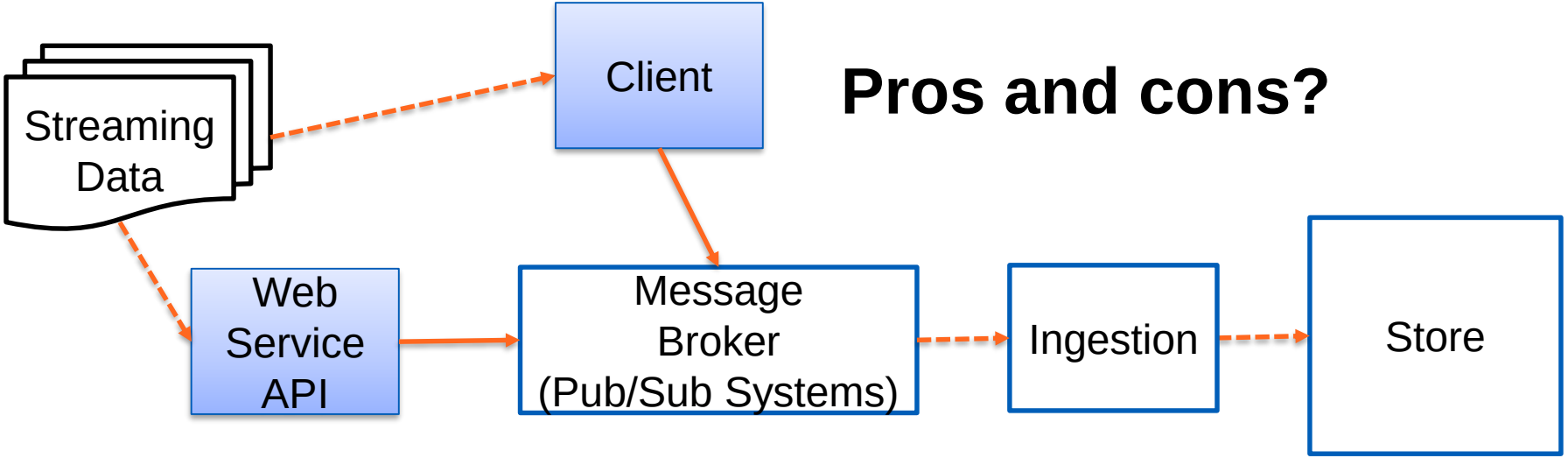
Split (pub/sub) and partition with ingestion



Do we have to merge data before ingestion



Which types of APIs for integration?



Pros and cons?

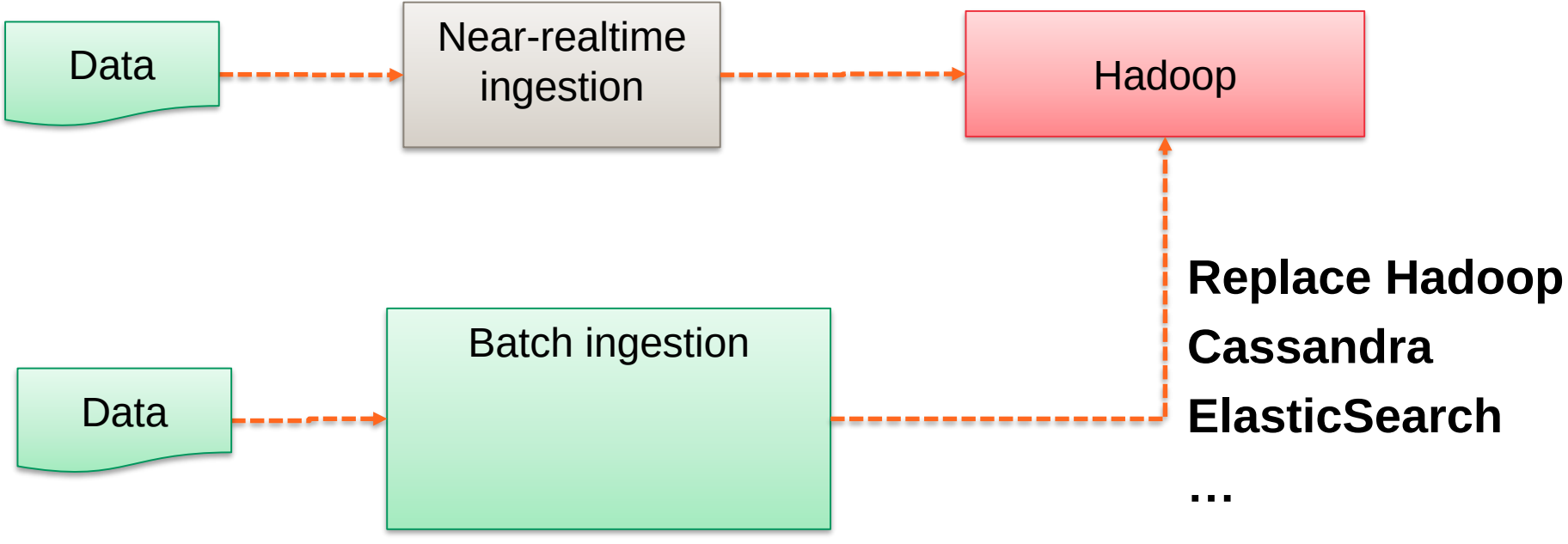
Complex ingestion pipelines in big data platforms

- **Multiple types of pipelines for multiple types of customers**
- **A customer might need different integrated pipelines**

→ **Both batch and near-realtime ingestion are supported**

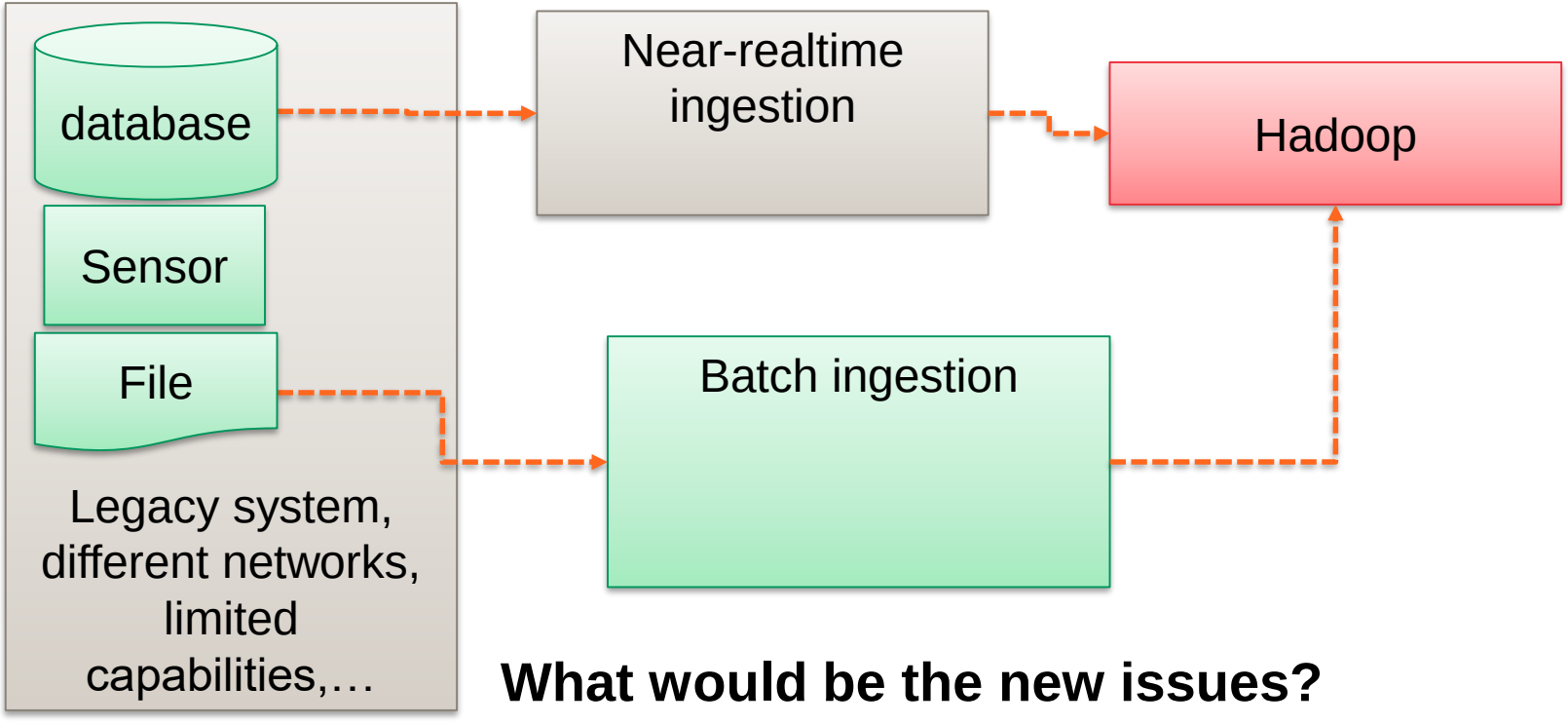
- **Complex architectural designs**
  - Ingestion pipeline-to-pipeline needs “bridges”

Multiple types of pipelines for the same destination



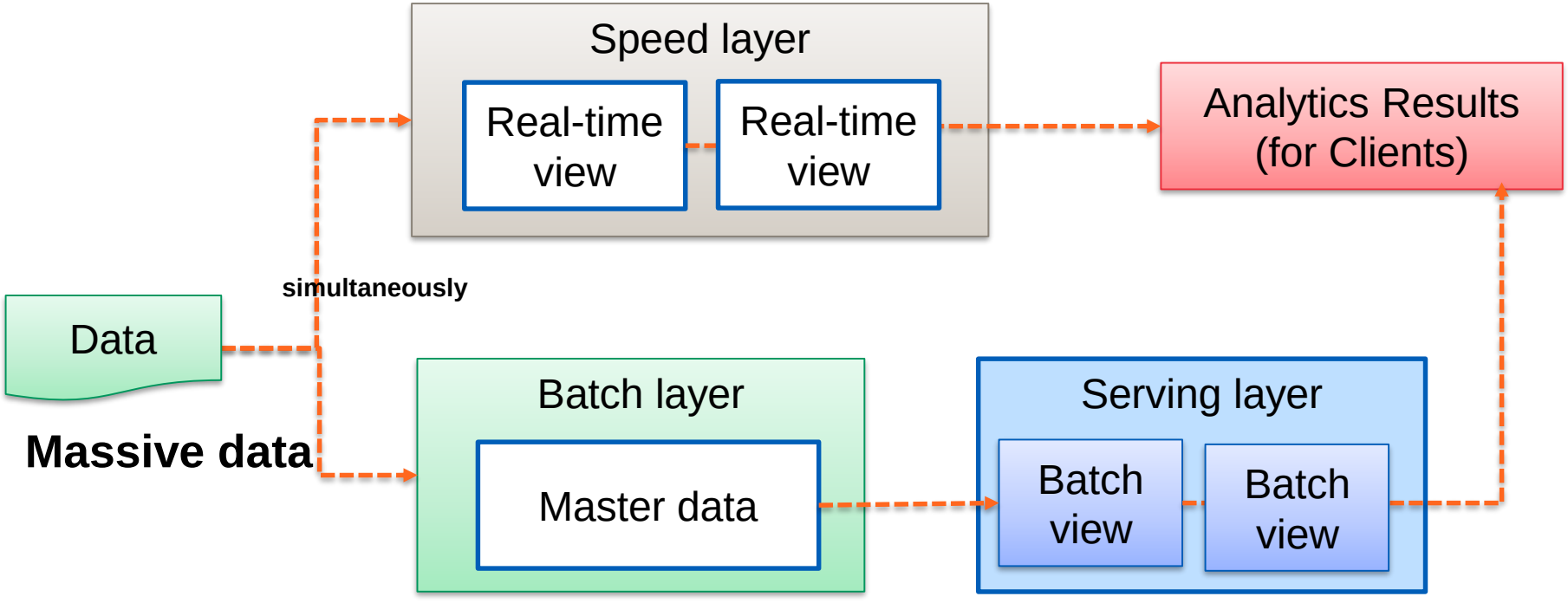
How much code can we reuse?

Multiple types of pipelines for the same destination



What would be the new issues?

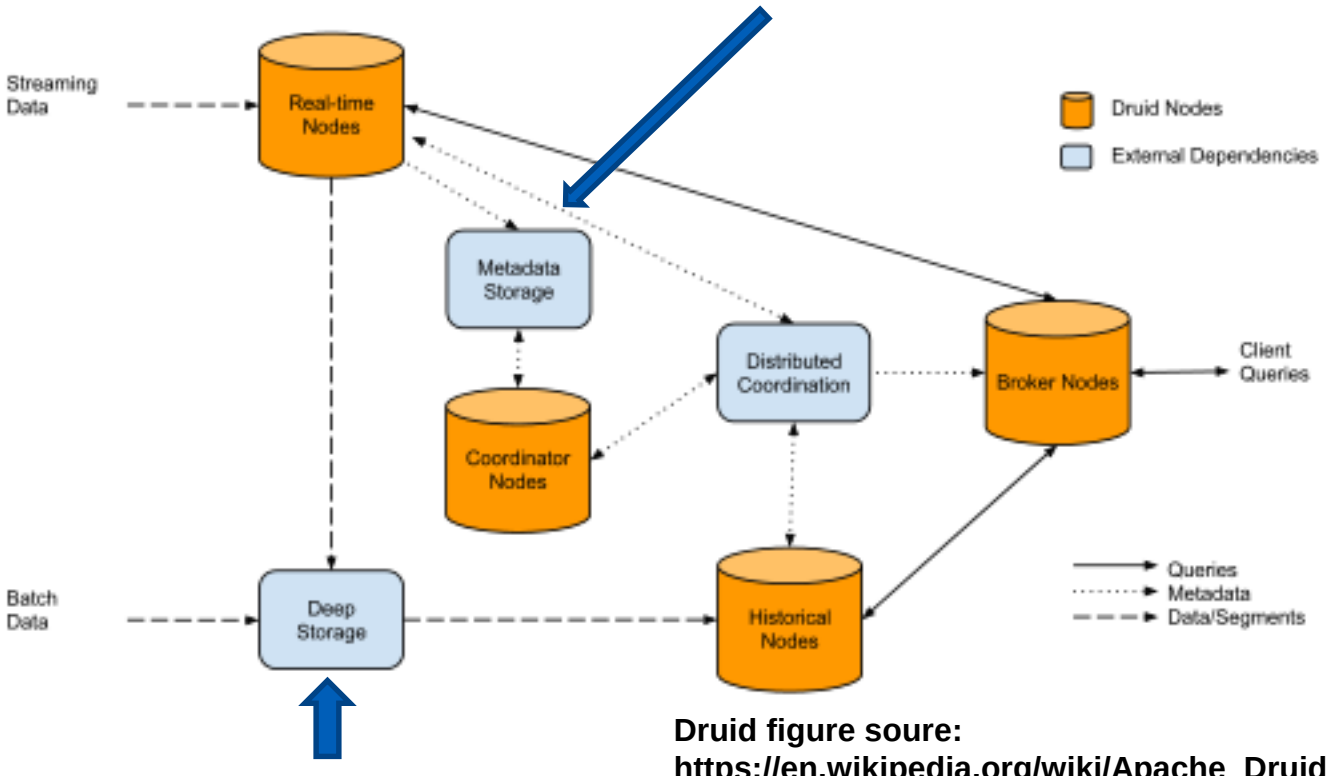
Recall: Lambda (see previous lectures)



Ingestion for different sinks

?

?

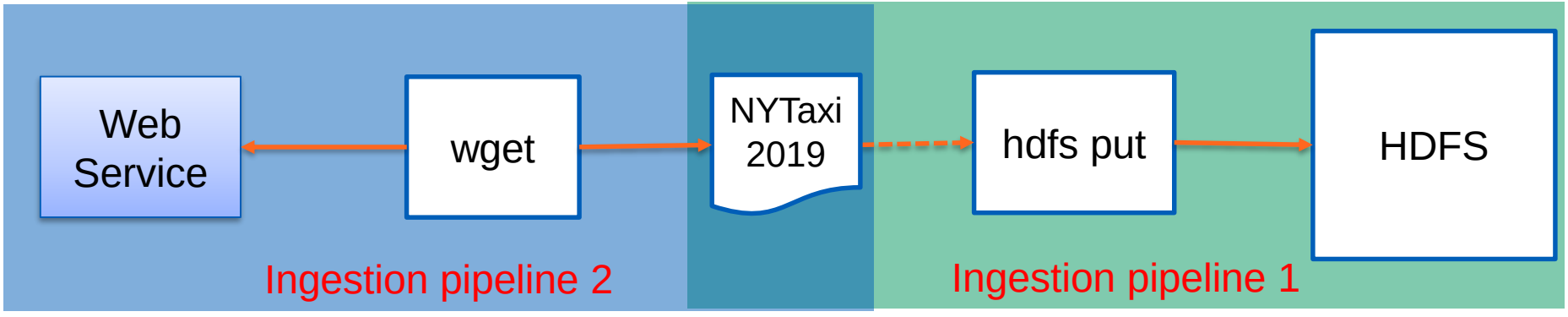


Druid figure soure: https://en.wikipedia.org/wiki/Apache_Druid

Connecting different ingestion pipelines

- **Do you still remember the Hadoop File Systems exercise?**

  hdfs dfs -put nytaxi2019 /user/mybdp/



Real-world:

Both pipelines and their connection are complex

Summary: “both configuration and programming are needed!”

- Many different architectural patterns and tools, suitable for different types of systems and data ingestion
- Ingestion pipeline-to-pipeline is common
- Learn existing designs for using and extending existing tools for your own design
- Implementation of ingestion of data might look easy until you ensure the correctness, quality control, etc.

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io