Messaging: Basic Exchange, Processing and Transformation Models and Tools

Hong-Linh Truong
Faculty of Informatics, TU Wien

hong-linh.truong@tuwien.ac.at
http://www.infosys.tuwien.ac.at/staff/truong
Twitter: @linhsolar

Outline

- Overview of streaming message-oriented data programming
- Communication - Message-Oriented Middleware
  - Java Messaging Service (JMS), Advanced Message Queuing Protocol (AMQP), Message Queuing Telemetry Transport (MQTT)
- Integration - Enterprise Integration patterns
  - Message routing patterns
  - Message transformation patterns
- Processing - streaming data processing with Complex Event Processing

Topic complexity

Thousand of pages of documents, APIs, tutorials and code

Getting started with each topic of “complex *” in 10 minutes.

Further advanced topics will be covered in Lecture 5

Why messaging is so important for DST?

Overview

STREAMING MESSAGE-ORIENTED PROGRAMMING

Data stream programming

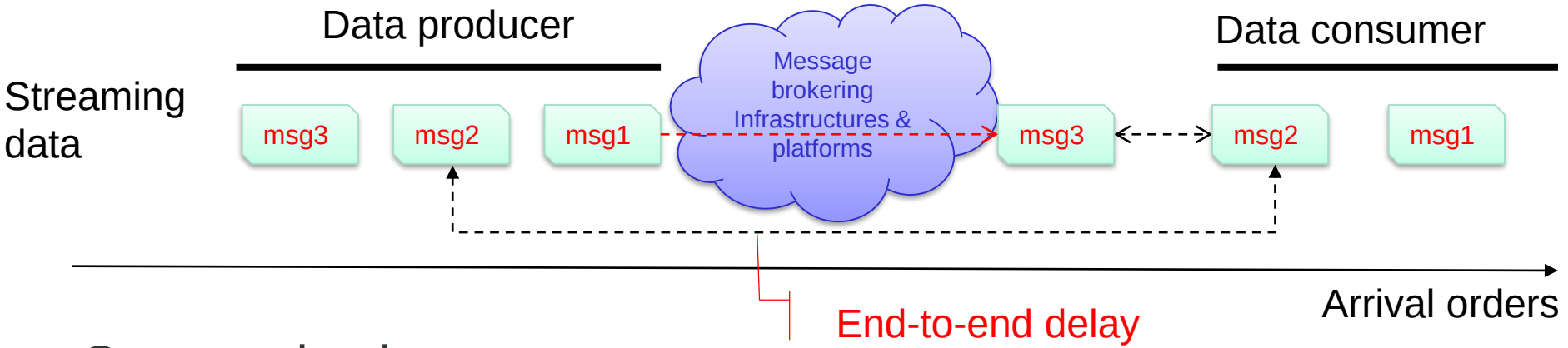
Data stream: a sequence/flow of data units

Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

Streaming data: produced by (near)realtime data sources as well as (big) static data sources

- Examples of data streams
  - Continuous media (e.g., video for video analytics)
  - Discrete media (e.g., stock market events, twitter events, system monitoring events, notifications)

Some key issues



- Communication
  - Many techniques are needed: sending/receiving, routing, storage, etc.
- Data processing
  - Within the brokering infrastructures and platforms
  - Within the producer and the consumer
  - Interoperability issues: message format, etc.
  - Performance issues: rates/throughput, intervals, delay/latency, processing time etc.

Message-oriented Middleware (MOM)

- Discrete media data units
  - Data units are structured messages (maybe ordered by timestamps)
- Well-supported in large-scale systems for
  - Persistent but asynchronous messages
  - Scalable message handling
- Message communication and transformation
  - publish/subscribe, routing, extraction, enrichment
- Several implementations

Amazon SQS

JMS

Apache Qpid™

stormmq

Apache Kafka

RabbitMQ
Messaging that just works

Message-oriented Persistent Communication

Exchange models

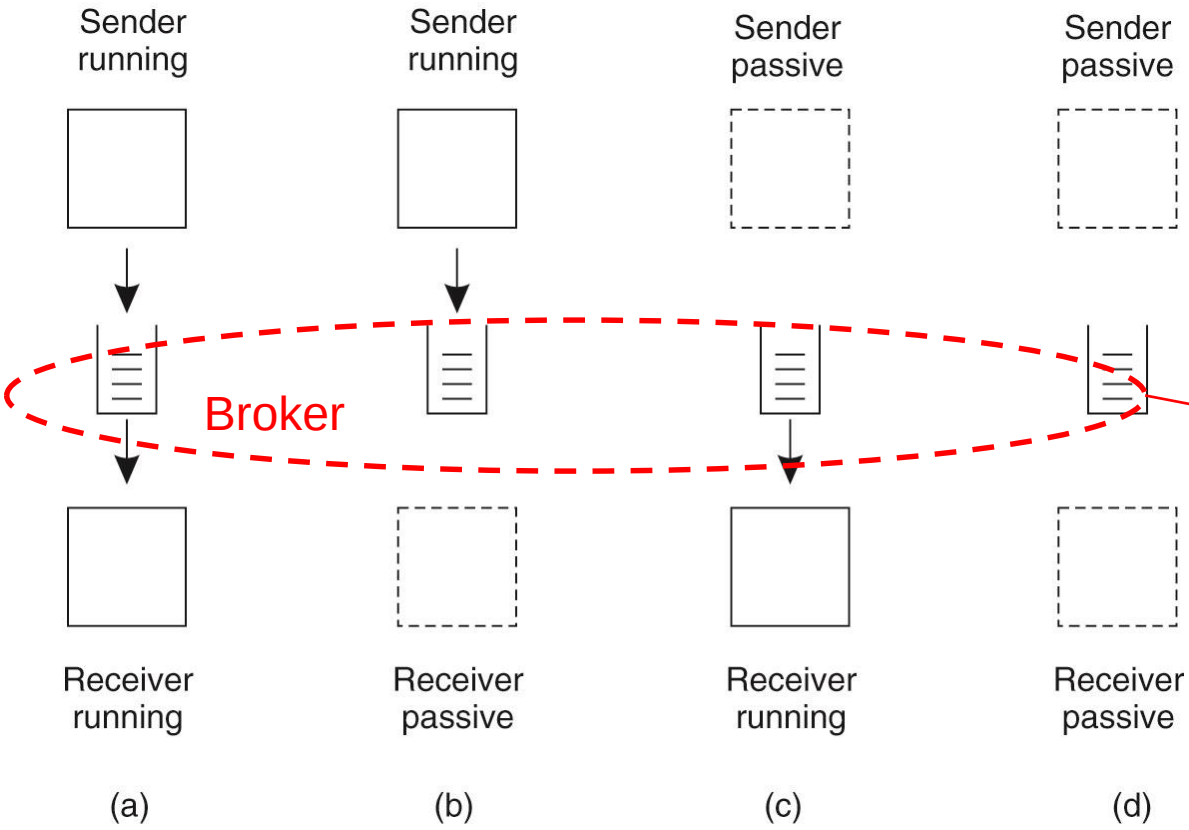
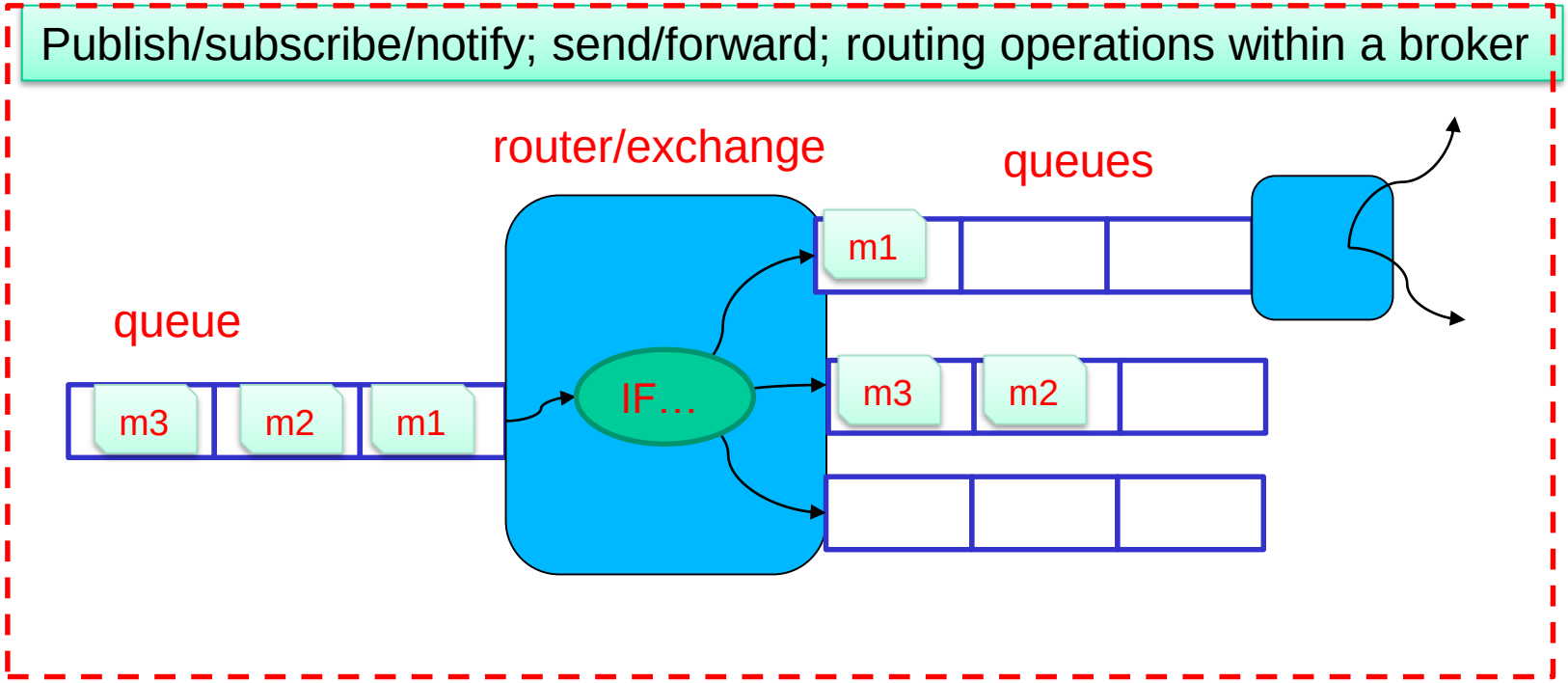


Fig source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

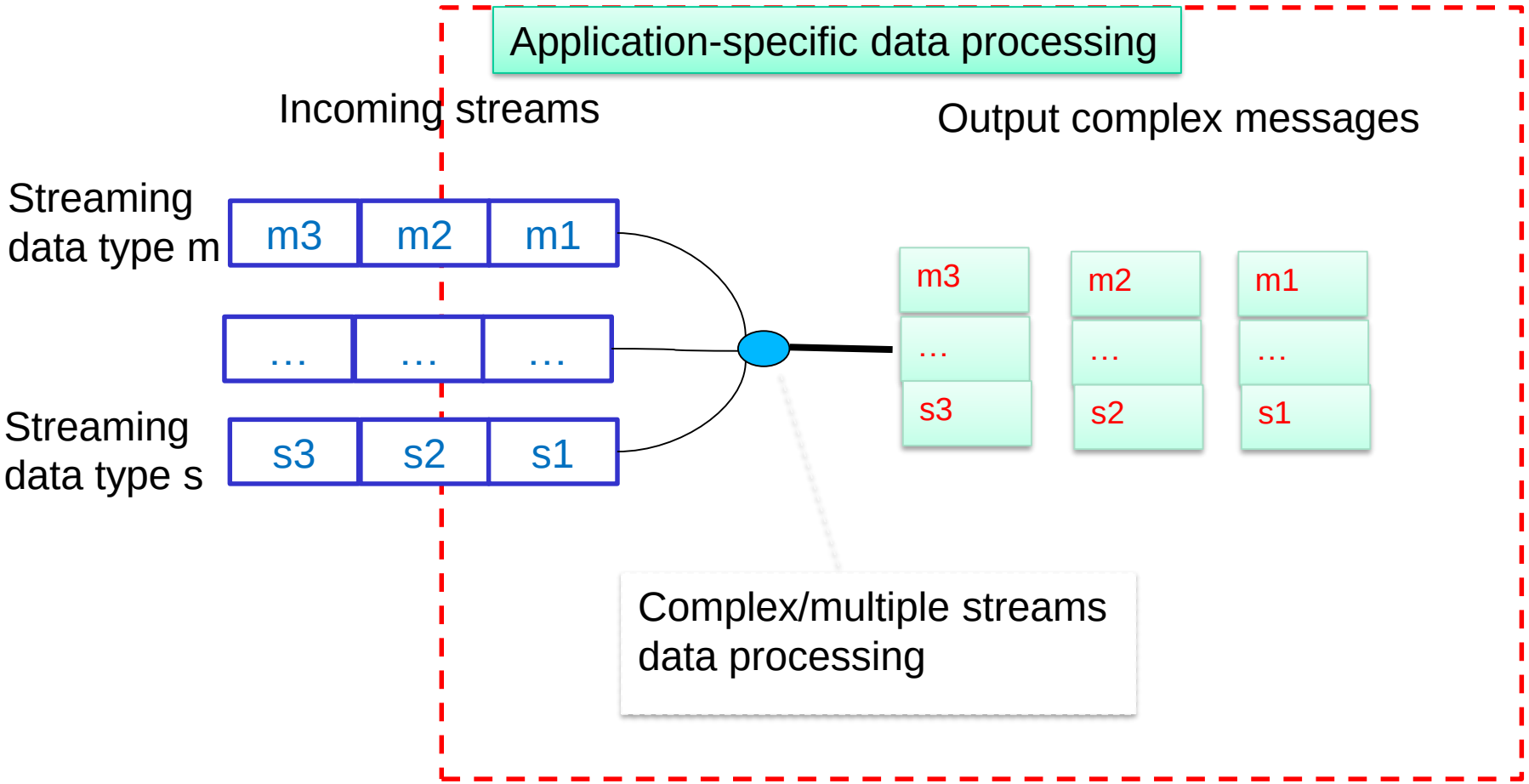
Operations
PUT/SEND/PUBLISH
GET/RECEIVE
POLL/SUBSCRIBE
NOTIFY/SEND

The receiver pulls the data from the broker or the broker pushes the data to the receiver?

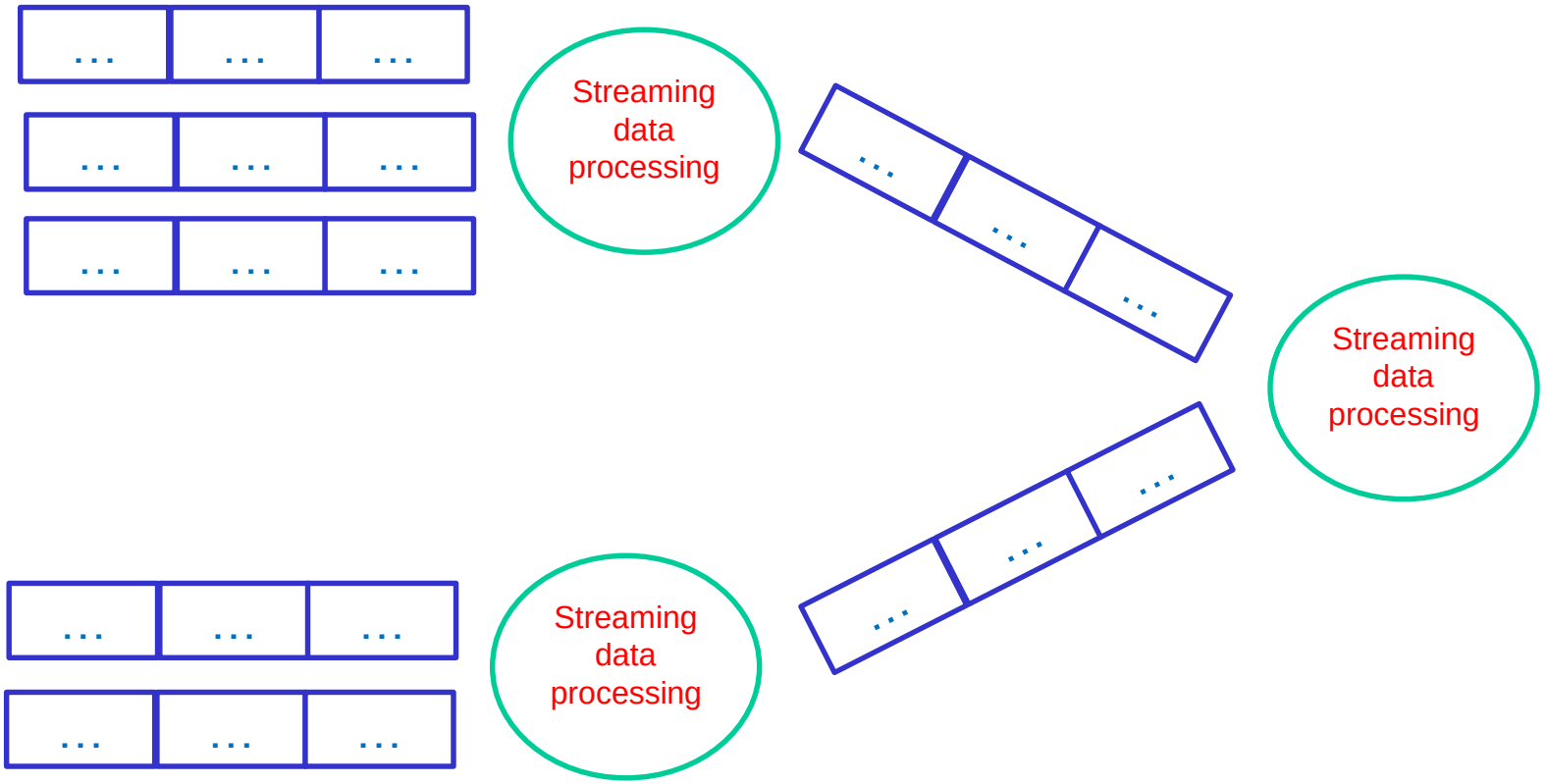
MOM – some message processing operations



Message processing within data consumer



Streaming data processing with a network of data processing elements

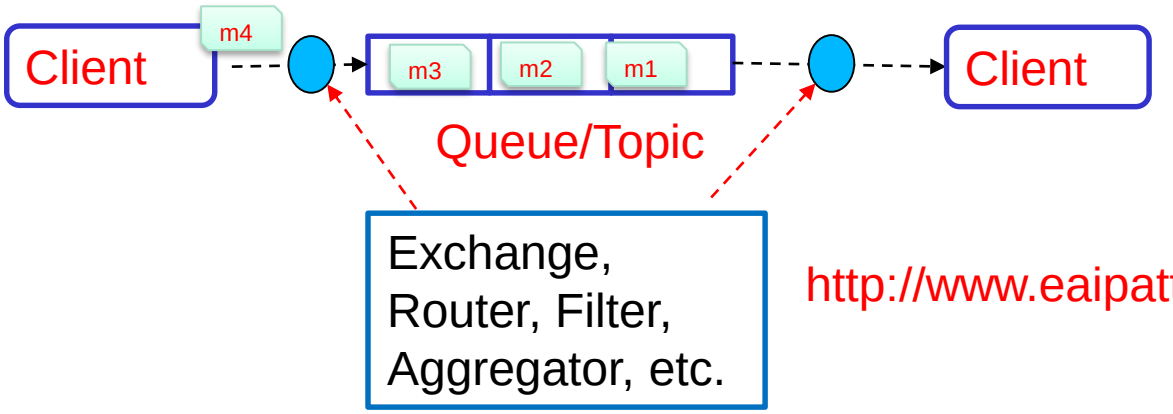


Message handling for service integration

- Messages handling concepts and patterns have been around for many years
  - Cross services/organizations integration
  - Enterprise integration pattern is well studied but mostly focused on business messages
  - http://www.enterpriseintegrationpatterns.com/

- Today distributed applications
  - not just enterprise integration patterns
  - also various types of measurements and log information integration

Filter, exchange, etc.



We need several features implemented by MOM, consumers, or external systems

Syntax and semantic problems

President: "Secretary! Nice to see you. What's happening?"
Secretary: "Sir, I have the report here about the new leader of China."

President: "Great. Lay it on me."
Secretary: "'Hu' is the new leader of China."

President: "That's what I want to know."
Secretary: "That's what I'm telling you."

President: "That's what I'm asking you. Who is the new leader of China?"
Secretary: "Yes."

President: "I mean the fellow's name."
Secretary: "Hu."

President: "The guy in China."
Secretary: "Hu."

President: "The new leader of China."
Secretary: "Hu."

President: "The Chinaman!"
Secretary: "Hu is leading China."

Source: http://www.smart-words.org/humor-jokes/language-humor/who-is-hu-china.html

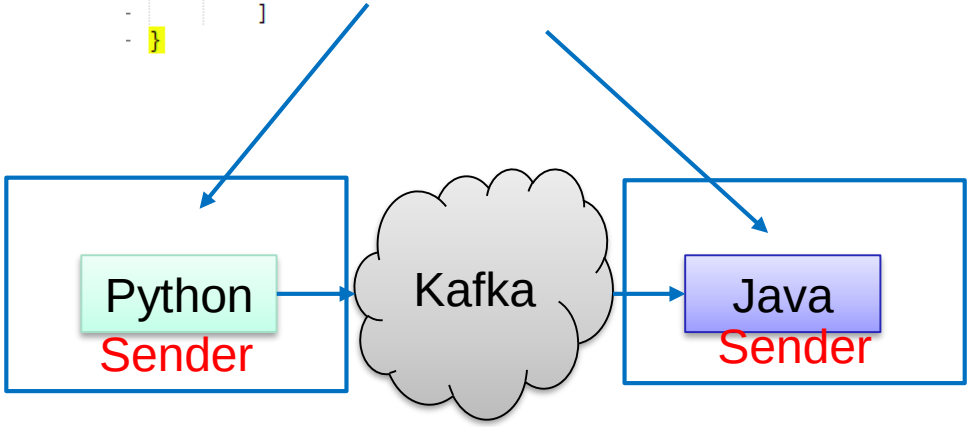
Message serialization and deserialization

- Remember that the 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

Arvo

- https://avro.apache.org/
- Support message description
- Serialize and deserialize libraries
- Work with different languages

```
{
    "namespace": "io.rdsea.training.kafka.storm",
    "type": "record",
    "name": "event",
    "fields": [
        {"name":"stationNetworkID", "type":"string"},
        {"name":"stationID", "type":"string"},
        {"name":"sensorName", "type":"string"},
        {"name":"metricName", "type":"string"},
        {"name":"metricValue", "type":"int"},
        {"name":"state", "type":"string"},
        {"name":"timeStamp", "type":"long"}
    ]
}
```



Why is it important?

Some other techniques

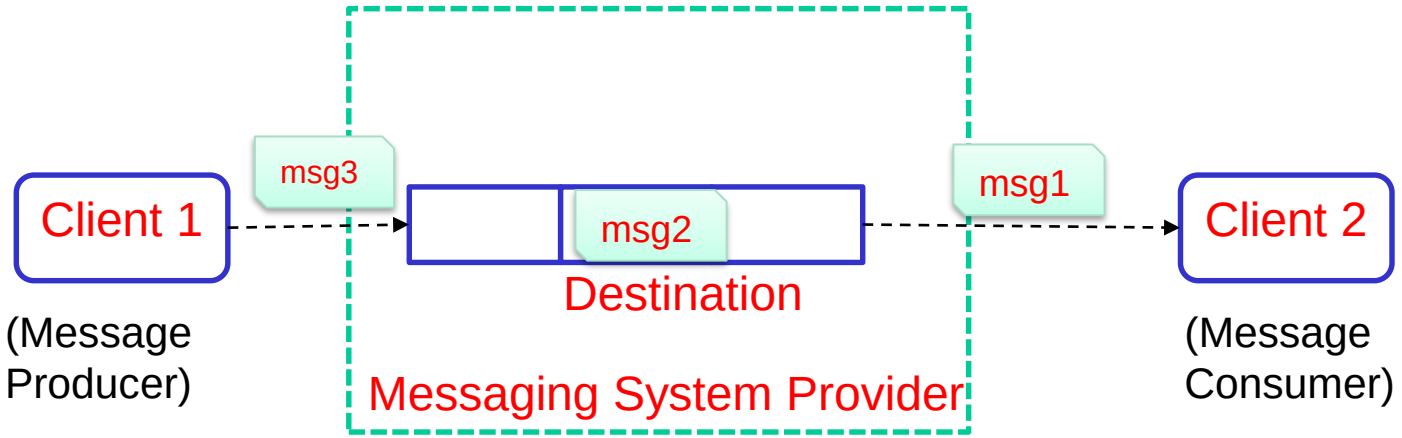
- Protobuf
  - From Google, used by default in gRPC (gRPC.io)
  - https://github.com/google/protobuf
  - Language-neutral, platform-neutral mechanism for serializing/deserializing structured data
- Thrift
  - https://thrift.apache.org
  - RPC style
  - Support also serializing and deserializing data)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

Communication

JAVA MESSAGING SERVICE

General concepts

- **Standard APIs** for Java platform



Message Structure

- Header: pre-defined system information (e.g., storage, routing and identification operations)
- Properties: application defined properties
- Body: application-defined
  - Java primitive types, Map (a set of tuples), Text, Serializable Object

- Types of messages (or what is a message for?)
  - Application-specific semantics
    - E.g., notify an event, send a document, or ask for the execution of a command

Delivery Patterns

Point-to-point

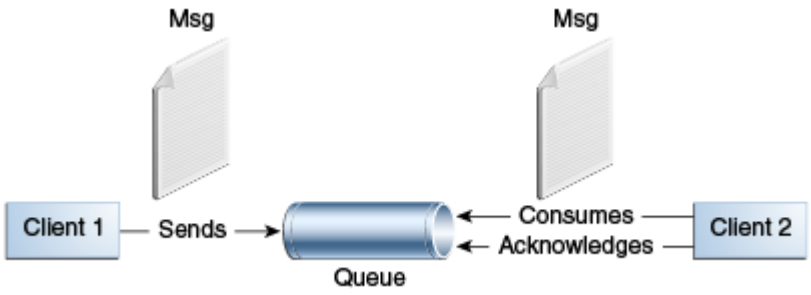


Fig source: http://docs.oracle.com/javaee/7/tutorial/doc/jms-concepts002.htm

Publish/Subscription (Pub/Sub)

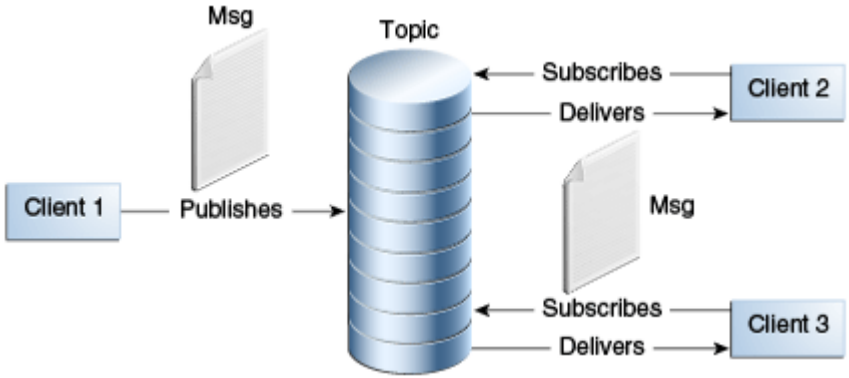


Fig source: http://docs.oracle.com/javaee/7/tutorial/doc/jms-concepts002.htm

Request-reply versus Request-only messages

- Request only
  - A sender does not expect a reply for a given request
- Request-reply
  - A sender expects, e.g., a system ack or an application-specific reply
- Some design principles
  - Need to uniquely identify a request message?

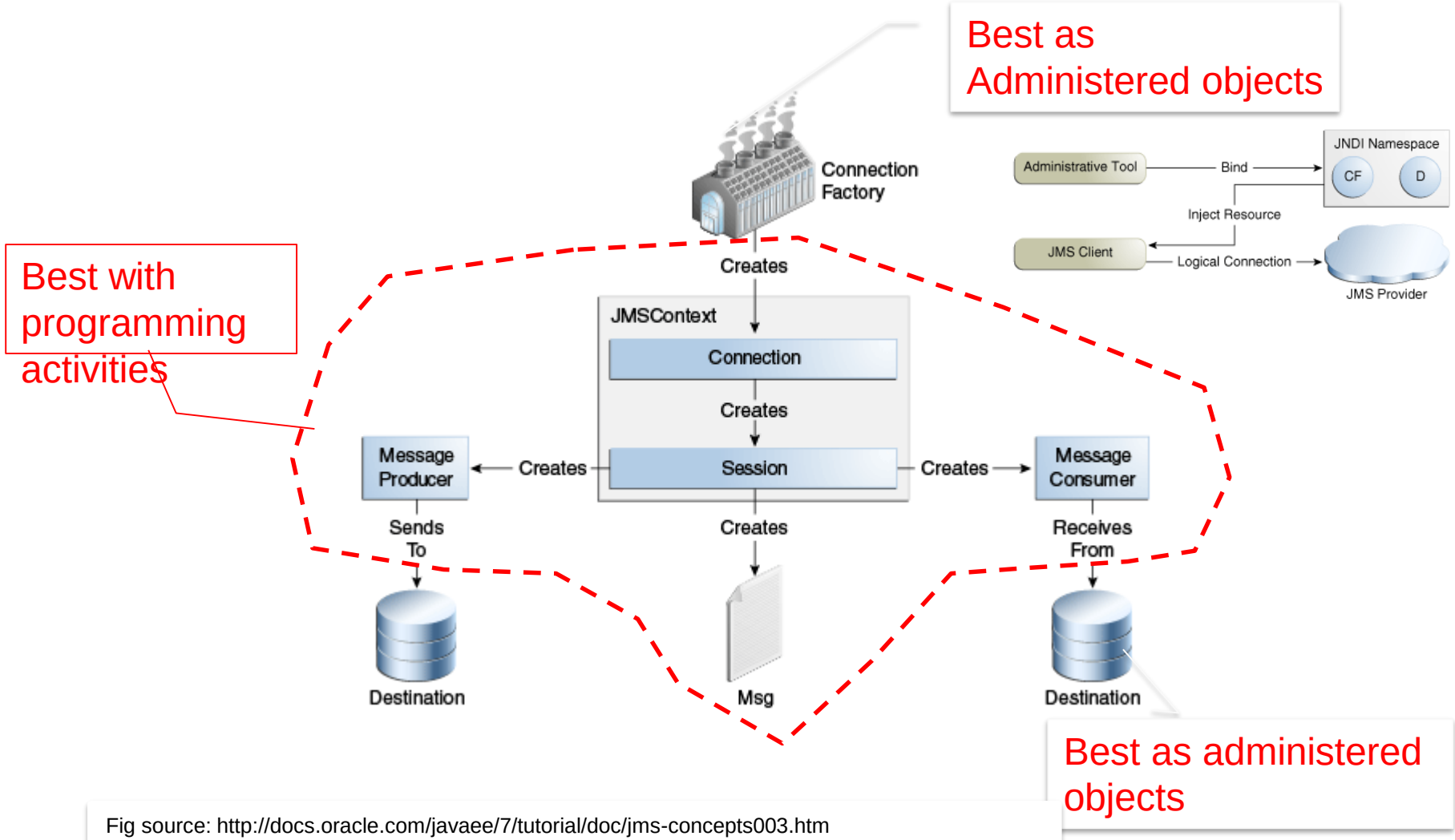
    → Use a unique identifier
  - Need a reply message from a request message

    → Where is the return address?

    → Correlation between the request and reply messages (using unique id), e.g., MessageType=REQUEST|REPLY & MessageID = ID

JMS programming versus administrative activities

Best as Administered objects



Best with programming activities

Best as administered objects

Fig source: http://docs.oracle.com/javaee/7/tutorial/doc/jms-concepts003.htm

Simple example from the Java tutorial

```
@Resource(lookup = "java:comp/DefaultJMSConnectionFactory")
 private static ConnectionFactory connectionFactory;
 @Resource(lookup = "jms/Queue")
  private static Queue dest;
....
try (JMSContext context = connectionFactory.createContext();) {
        int count = 0;
        for (int i = 0; i < NUM_MSGS; i++) {
                message = "This is message " + (i + 1) + " from producer";
                TextMessage msg = context.createTextMessage();
                msg.setText(message);
                 msg.setIntProperty("ID",count);
                if (((i+1) %2 )==0) {
                   msg.setStringProperty("msgType","EVEN");
                 }    else            msg.setStringProperty("msgType","ODD");
                 context.createProducer()
                            .setDeliveryMode(DeliveryMode.NON_PERSISTENT)
                            .send(dest, msg);
           count += 1;
        }
        System.out.println("Messages sent: " + count);
```

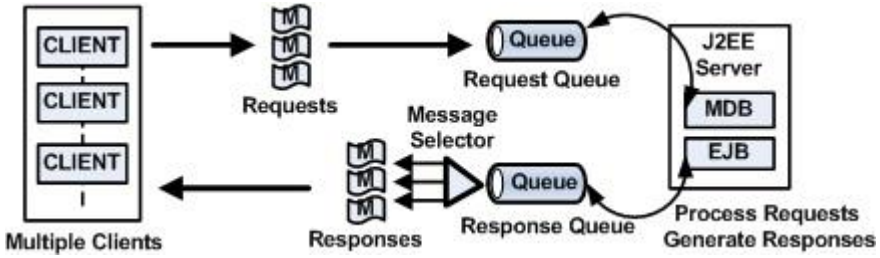
Some other JMS API features

- Control message acknowledgement
  - By JMS provider or by the client
- Message parameters
  - Persistent, priority, delay, and expiration
- Programming temporal destinations
- Nondurable versus durable subscription for subscribers
- Asynchronous sending

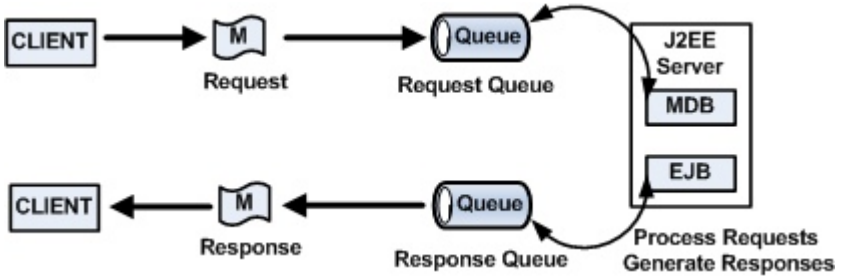
Generic question: how does the broker manage durable subscription?

Example of temporary queues for performance improvement

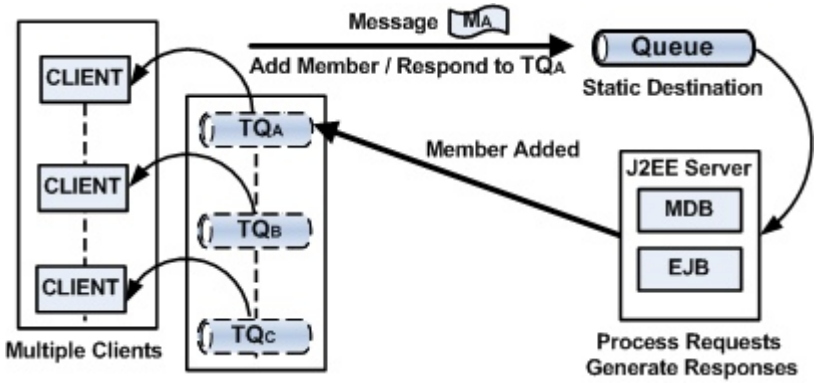
Common static queues for multiple clients



Separate static queues for multiple clients

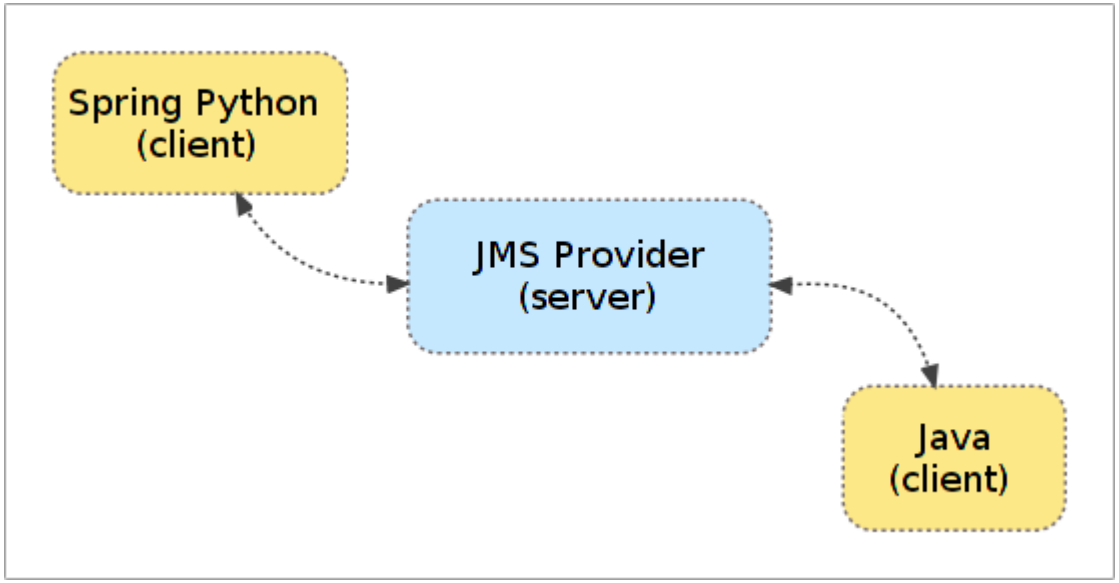


Temporary queues



Use cases and Figs source: http://www.onjava.com/2007/04/10/designing-messaging-applications-with-temporary-queues.html

Outside the java world?



Source: http://docs.spring.io/spring-python/1.2.x/sphinx/html/jms.html

Recall

Would you use a JMS topic or queue?

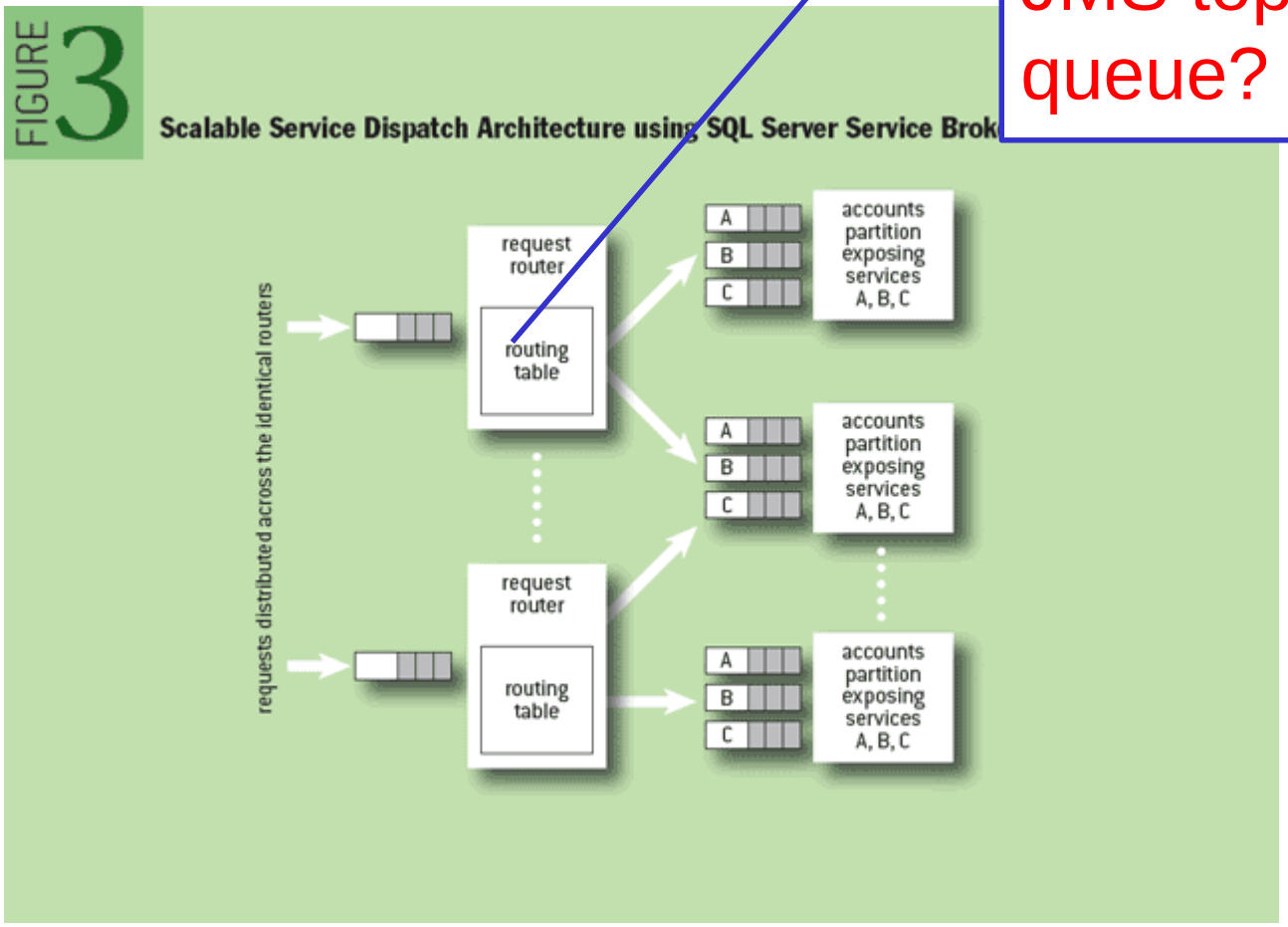


Figure source: http://queue.acm.org/detail.cfm?id=1971597

Communication

ADVANCED MESSAGE QUEUING PROTOCOL

Overview

- MOM, but not language- or platform- specific
  - For Java, C#, Python, ….
  - Solving message interoperability in heterogeneous environments of MOMs
- Binary wire-level protocol for message exchange, rather than APIs
  - It does not include broker behaviors/capabilities but they were in the standard before version 1.0
- http://www.amqp.org

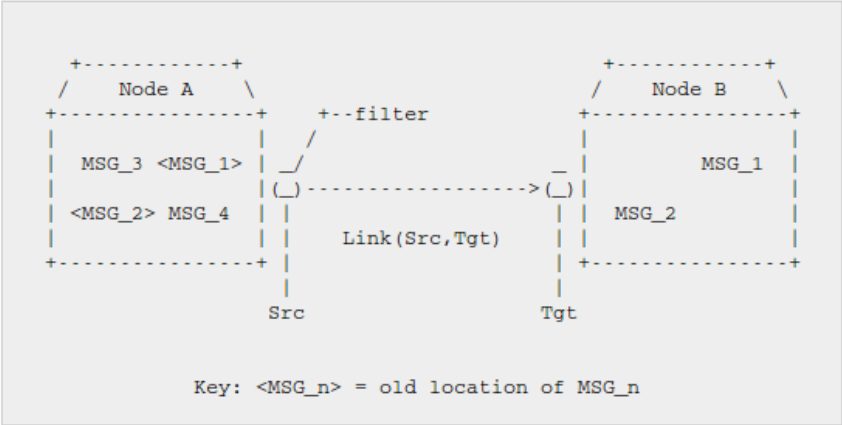
Apache Qpid™

stormmq

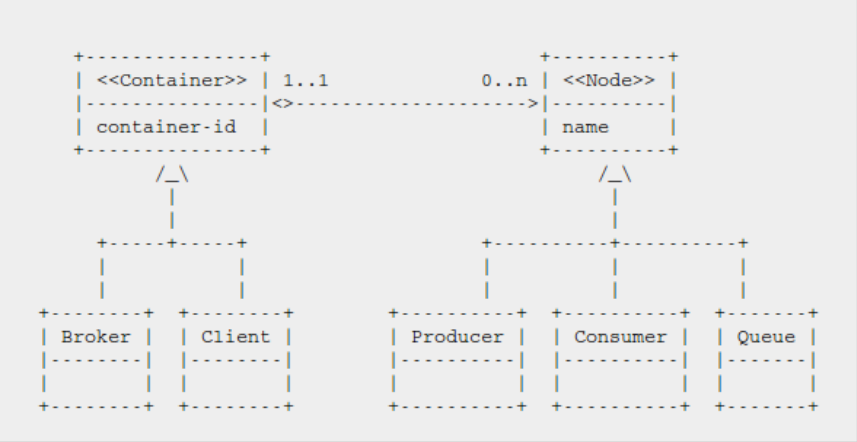
RabbitMQ
Messaging that just works

Core concepts – Message/Transport

- Message representation
  - Defined based on type systems for interoperability



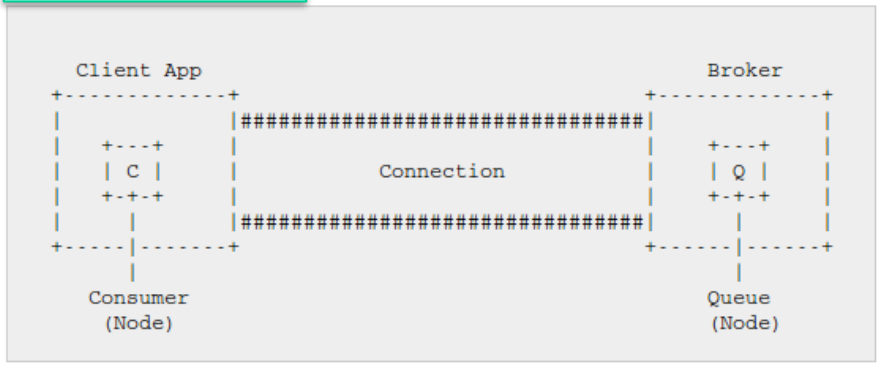
- Transport
  - A network of nodes connected via links
  - Node: message storage, delivery, relay, etc.
  - Container: includes nodes



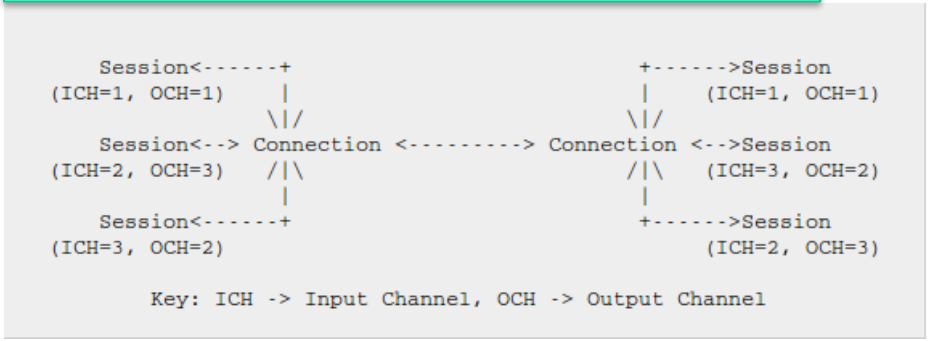
Figs source: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

Core concept -- Transport

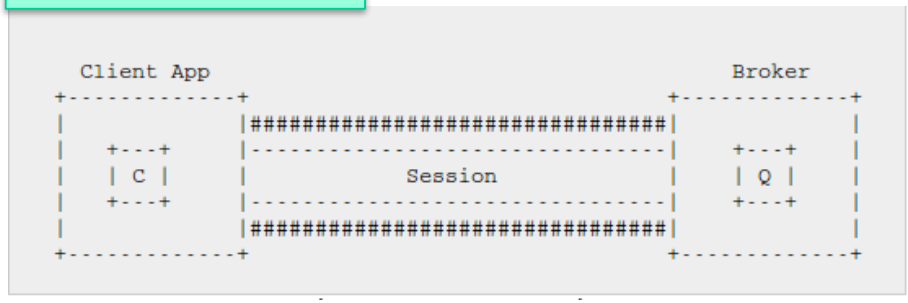
Connection



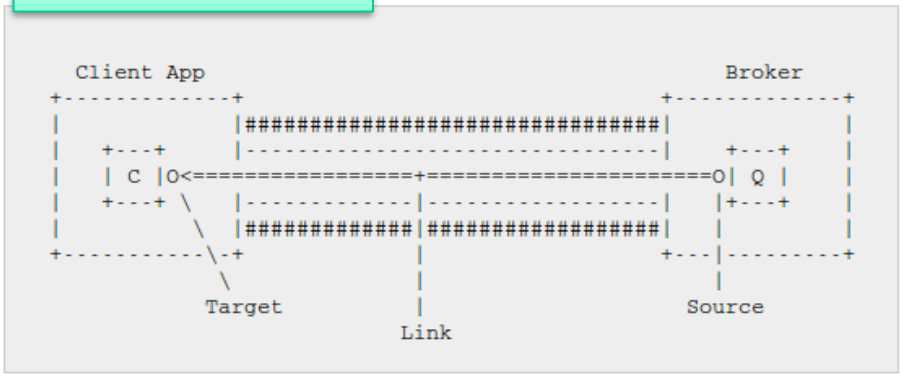
Session and Connection endpoints



Session



Links



Figs source: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

Example

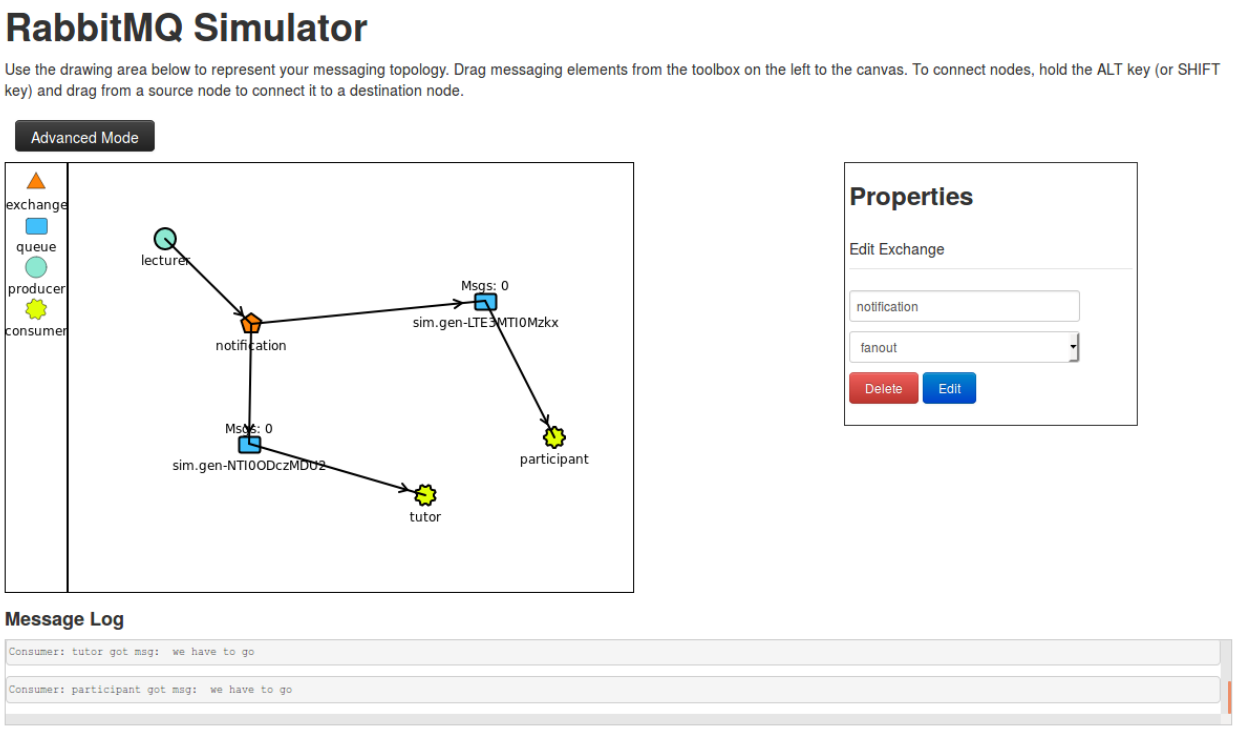
- Get a free instance of RabbitMQ from cloudamqp.com
- Get code from: https://github.com/cloudamqp/java-amqp-example
- First run the test sender, then run the receiver


```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    for (int i=0; i<100; i++) {
        String message = "Hello distributed systems guys:  "+i;
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
        System.out.println(" [x] Sent '" + message + "'");
        new Thread().sleep(5000);
    }
```

```
while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    String message = new String(delivery.getBody());
    System.out.println(" [x] Received '" + message + "'");
  }
```

Note: i modified the code a bit

Real code versus simulation



http://tryrabbitmq.com/

Performance

- “RabbitMQ Hits One Million Messages Per Second on Google Compute Engine”
  - https://blog.pivotal.io/pivotal/products/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine
  - https://cloudplatform.googleblog.com/2014/06/rabbitmq-on-google-compute-engine.html
  - Using 32 nodes
- RabbitMQ is widely used in big industries!

http://mqtt.org

MESSAGE QUEUING TELEMETRY TRANSPORT (MQTT)

MQTT Overview

- OASIS Standard
- ISO/IEC 20922:2016 (Message Queuing Telemetry Transport (MQTT) v3.1.1)
- M2M Connectivity Protocol atop TCP/IP
- MQTT brokers enable publish/subscribe messaging systems
  - Publisher can publish a messge within a topic that can be subscribed by many Subscribers
- Simple protocols
  - Suitable for constrained devices.

Protocol Features

- Lightweight protocol
  - Small message size
  - QoS
    - At most once, at least once and exactly once
  - Few commands/interactions: CONNECT, PUBLISH, SUBSCRIBE, UNSUBRIBE, DISCONNECT
    - Easy to implement
- Small foot-print libary
- Low bandwidth, high latency, data limits, and fragile connections
- Suitable for IoT (constrained devices/networks)

How QoS would impact the design of the subscriber?

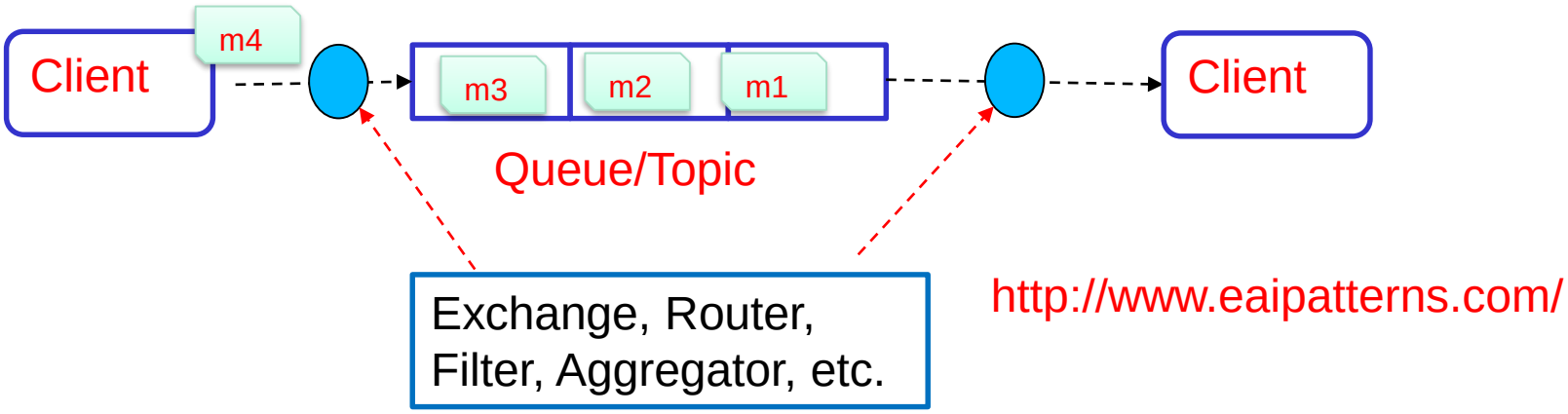
Model and Implementation

- Different programming languages for OS/devices
  - Including Anrduino, Nanode
- Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto)
- Paho: http://www.eclipse.org/paho/
- RabbitMQ
- Apache ActiveMQ
- Cloud providers:
  - http://cloudmqtt.com (get a free account to learn MQTT)

Integration

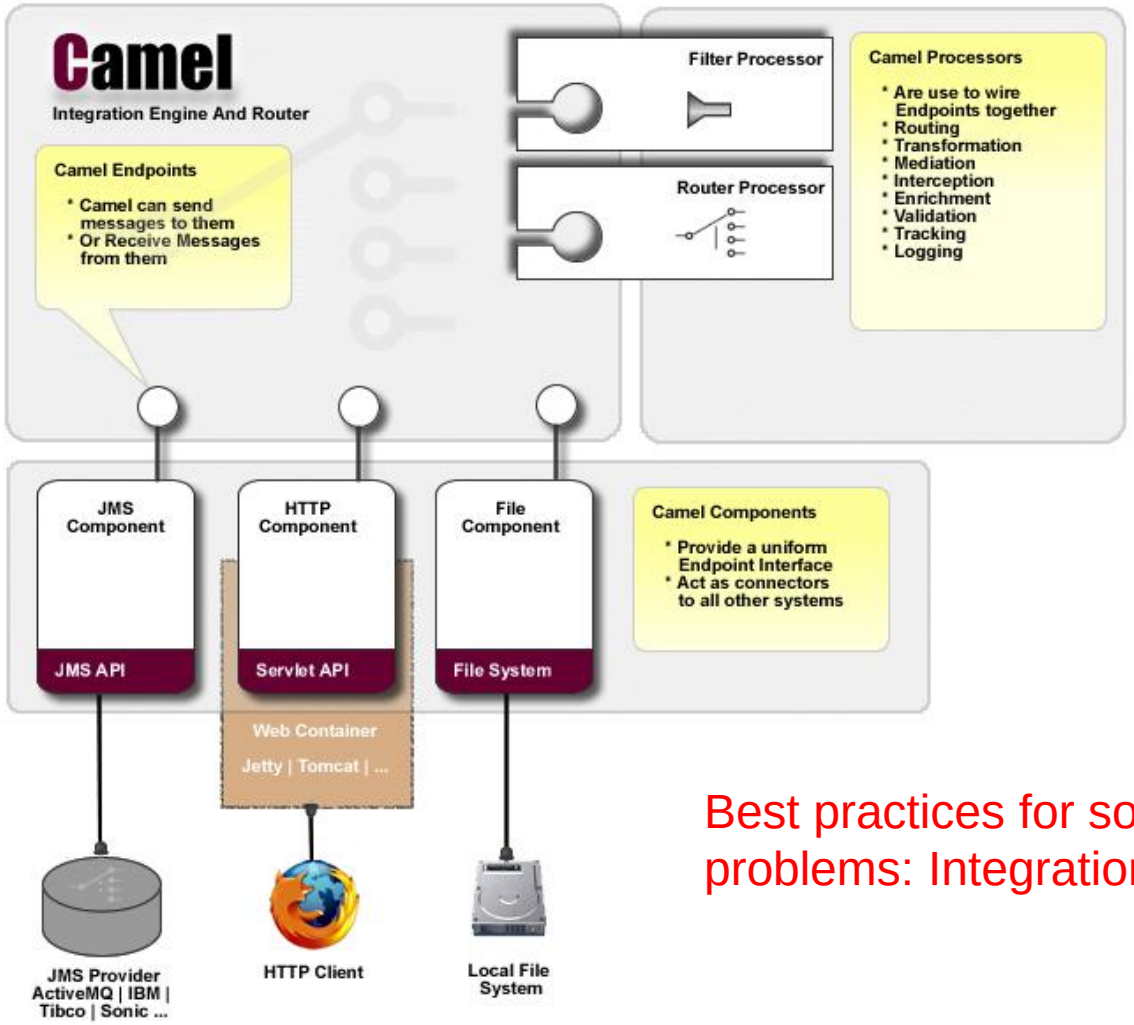
MESSAGE ROUTING PATTERNS

Integration Issues



- We need several features implemented by MOM, consumer, or external systems

Example of supporting technology



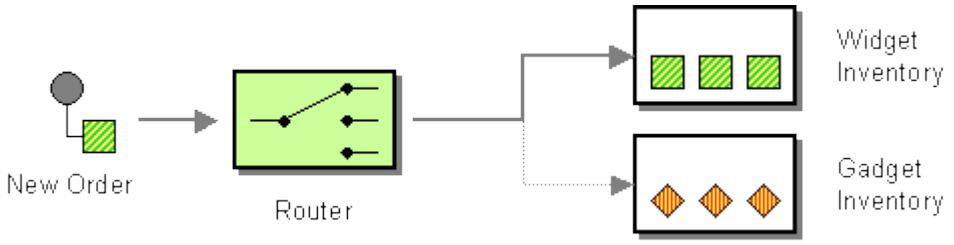
Best practices for solving common problems: Integration Patterns

Also check: http://projects.spring.io/spring-integration/

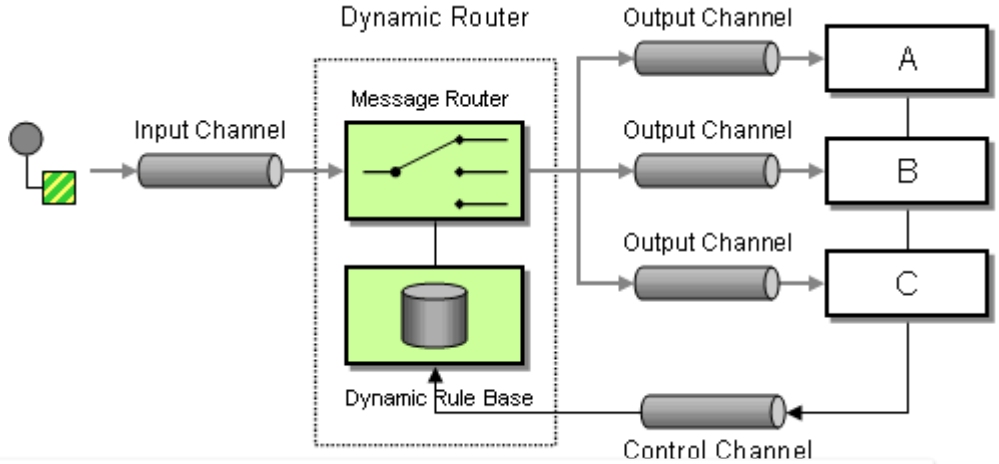
Content-Based Message Routing: Camel/EIP

Content-Based Router: can be used to decide the right destination queue for a given message based on the message content

Dynamic Router: can self-configure based on processing messages

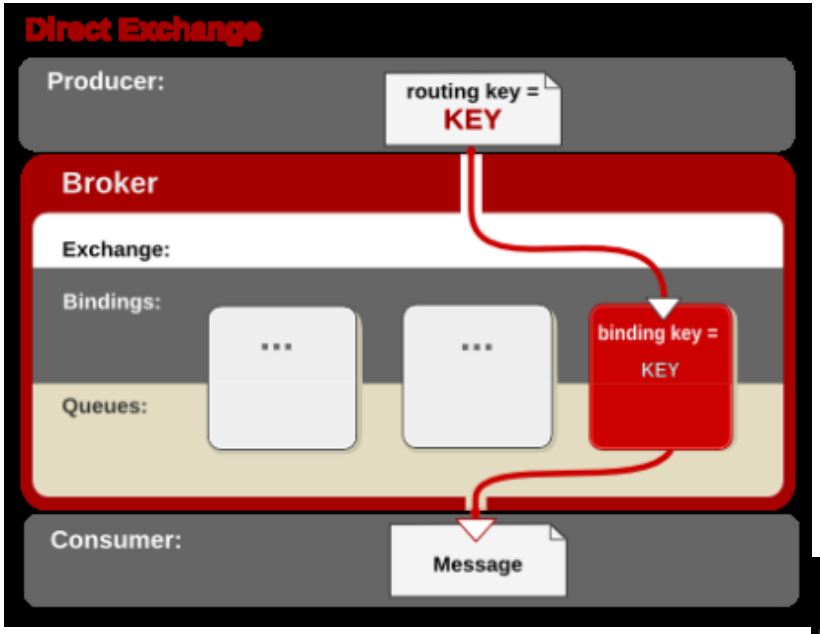


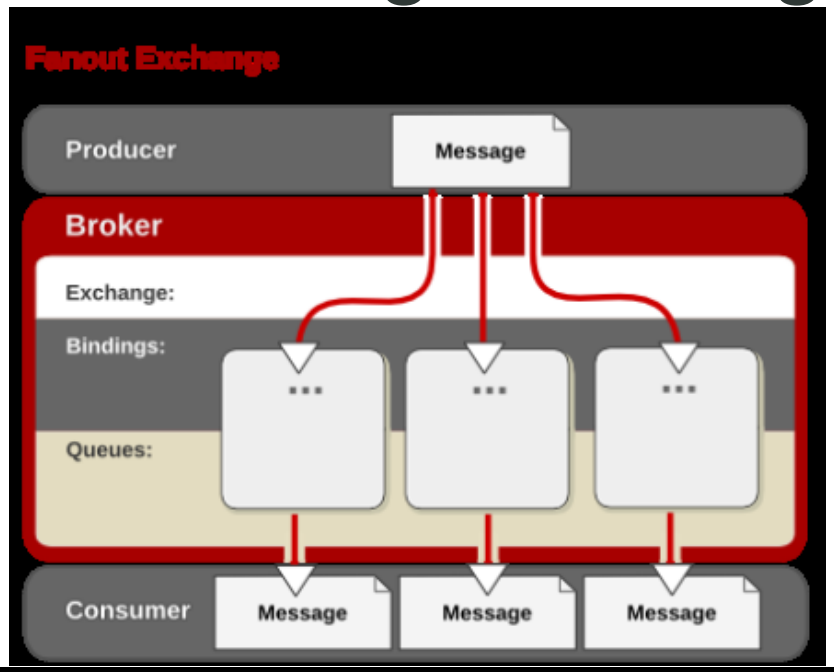
Source: https://camel.apache.org/content-based-router.html

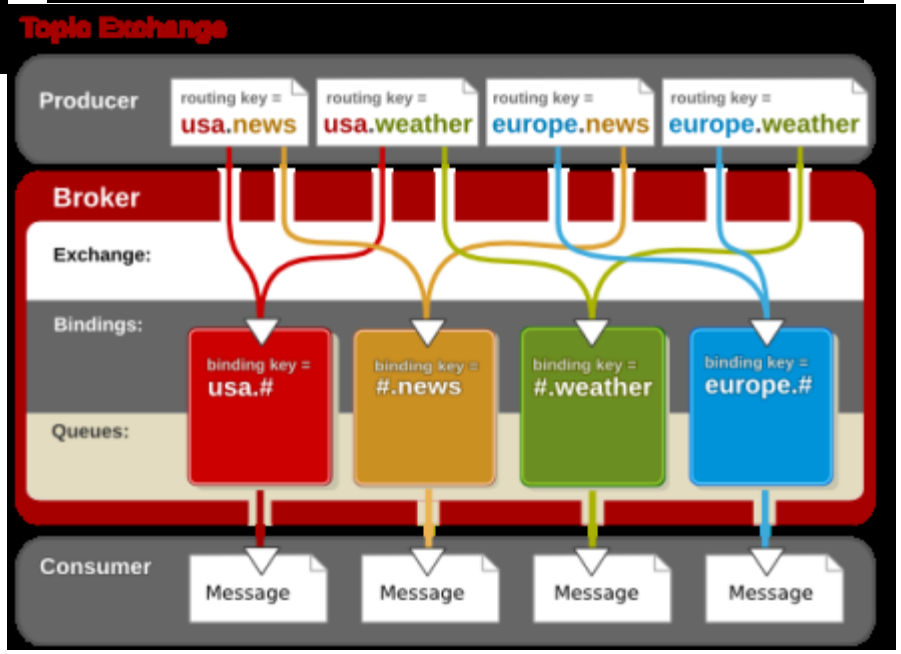


Source: https://camel.apache.org/dynamic-router.html

Content-Based Message Routing: AMQP







Note: defined in AMQP 0-10
But not in AMQP 1.0

Figs source: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_MRG/1.1/html/Messaging_User_Guide/chap-Messaging_User_Guide-Exchanges.html

Some code example with RabbitMQ

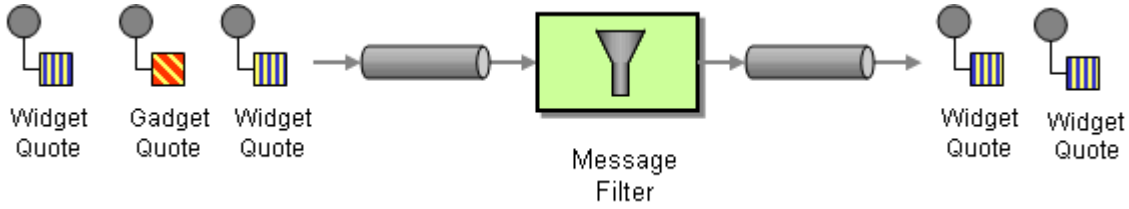
Message Filter/Selector

JMS: selector based on message header and properties

Message Selector or Message Filter: filter unneeded messages

```
TextMessage msg = context.createTextMessage();
msg.setText(message);
msg.setIntProperty("ID",count);
if ((count % 2 )==0) {
        msg.setStringProperty("msgType",“EVEN");
}
else
        msg.setStringProperty("msgType",“ODD");

JMSConsumer consumer = context.createConsumer(dest,"msgType
=‘EVEN’”);
```

CAMEL/EIP: Message Filter



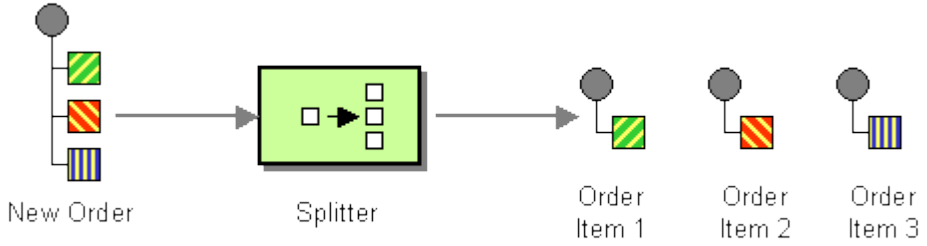
https://camel.apache.org/message-filter.html

Integration

TRANSFORMATION PATTERNS AND TOOLS

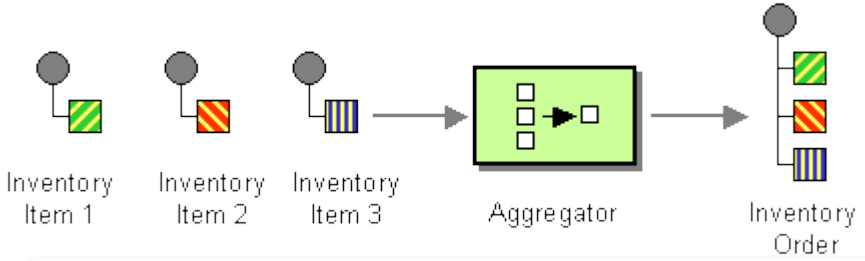
Splitter and Aggregator

Splitter: decompose a composite message into different messages



https://camel.apache.org/splitter.html

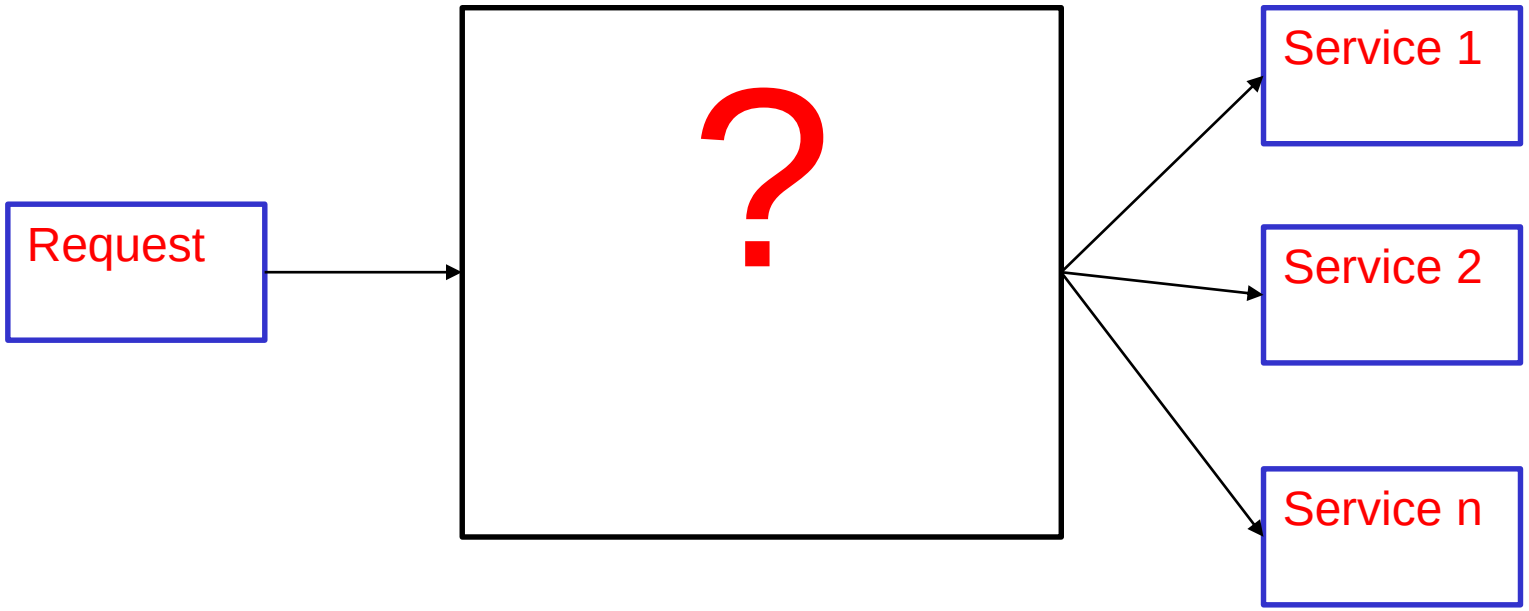
Aggregator: gather all correlated messages for a specific purpose then build a new composite message



https://camel.apache.org/aggregator2.html

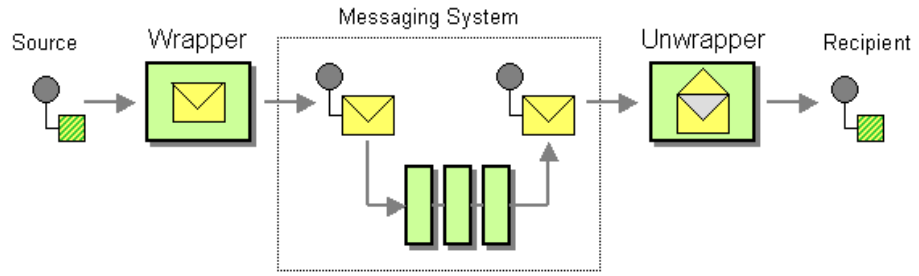
Questions: for which scenarios/use cases we can use the above-mentioned patterns

How would you use splitter and aggregator with a set of microservices for a request



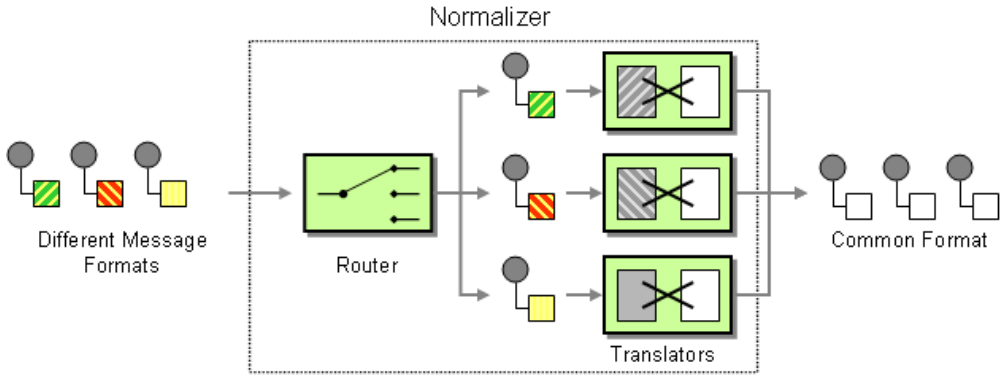
Envelope Wrapper and Normalizer

Envelope wrapper: wrap a message before sending it into a messaging system and unwrap it after the wrapped message leaves the messaging system



http://www.eaipatterns.com/EnvelopeWrapper.html

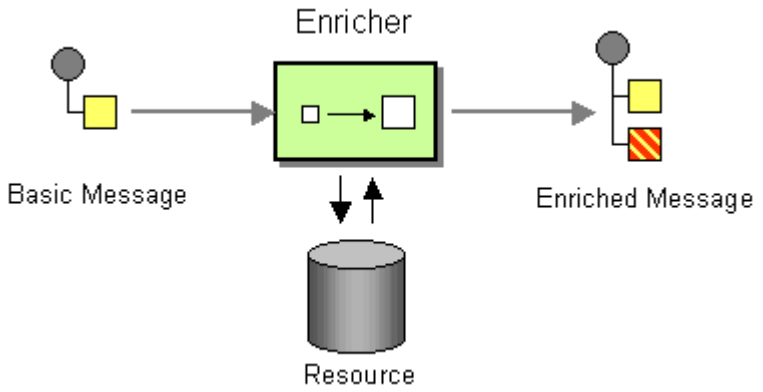
Normalizer: route all messages of a given type to a suitable Message Translator which transforms the message to the common format.



https://camel.apache.org/normalizer.html

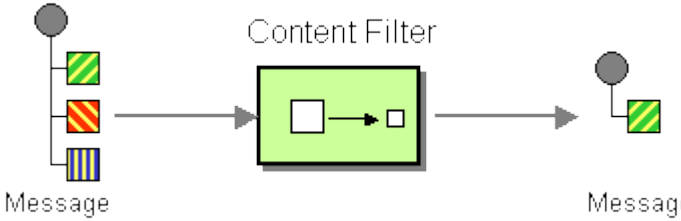
Content Enricher & Extracter

Content Enricher: obtain required/missing data then enrich the message with the newly obtained data



https://camel.apache.org/content-enricher.html

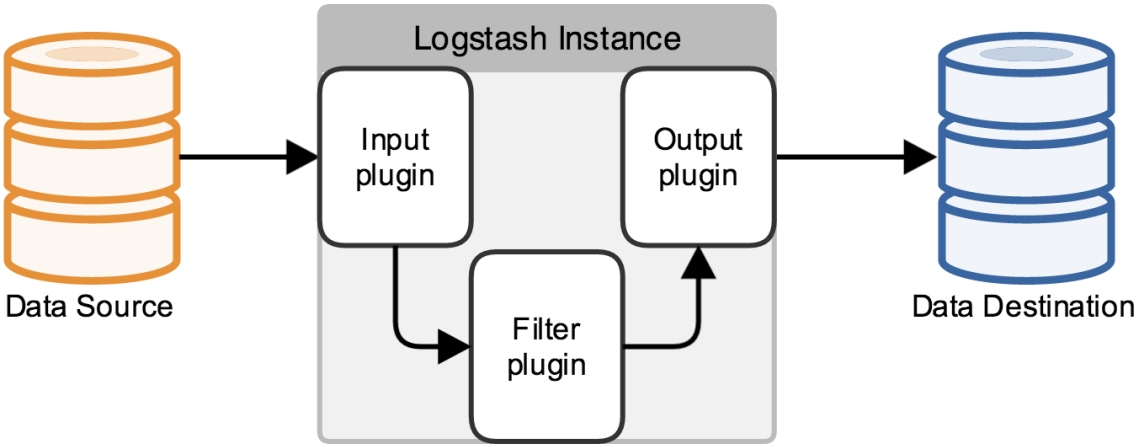
Content Filter: remove unimportant data items from a message or extract only needed information.



https://camel.apache.org/content-filter.html

Question: is it possible to send the to-be-enriched message to an external service to enrich it or to send the message to an external extraction service?

Logstash



- Codecs: stream filters within inputs or outputs that change data representation
- E.g.: multilines → a single event

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

Plug-ins

beats	Receives events from the Elastic Beats framework	logstash-input-beats
cloudwatch	Pulls events from the Amazon Web Services CloudWatch API	logstash-input-cloudwatch
couchdb_changes	Streams events from CouchDB's `_changes` URI	logstash-input-couchdb_changes
drupal_dblog	Retrieves watchdog log events from Drupal installations with DBLog enabled	logstash-input-drupal_dblog
elasticsearch	Reads query results from an Elasticsearch cluster	logstash-input-elasticsearch
eventlog	Pulls events from the Windows Event Log	logstash-input-eventlog
exec	Captures the output of a shell command as an event	logstash-input-exec
file	Streams events from files	logstash-input-file
ganglia	Reads Ganglia packets over UDP	logstash-input-ganglia
gelf	Reads GELF-format messages from Graylog2 as events	logstash-input-gelf
gemfire	Pushes events to a GemFire region	logstash-input-gemfire
generator	Generates random log events for test purposes	logstash-input-generator
github	Reads events from a GitHub webhook	logstash-input-github
graphite	Reads metrics from the `graphite` tool	logstash-input-graphite
heartbeat	Generates heartbeat events for testing	logstash-input-heartbeat

aggregate	Aggregates information from several events originating with a single task	logstash-filter-aggregate
alter	Performs general alterations to fields that the `mutate` filter does not handle	logstash-filter-alter
anonymize	Replaces field values with a consistent hash	logstash-filter-anonymize
cidr	Checks IP addresses against a list of network blocks	logstash-filter-cidr
cipher	Applies or removes a cipher to an event	logstash-filter-cipher
clone	Duplicates events	logstash-filter-clone
collate	Collates events by time or count	logstash-filter-collate
csv	Parses comma-separated value data into individual fields	logstash-filter-csv
date	Parses dates from fields to use as the Logstash timestamp for an event	logstash-filter-date
de_dot	Computationally expensive filter that removes dots from a field name	logstash-filter-de_dot
dissect	Extracts unstructured event data into fields using delimiters	logstash-filter-dissect
dns	Performs a standard or reverse DNS lookup	logstash-filter-dns
drop	Drops all events	logstash-filter-drop
elapsed	Calculates the elapsed time between a pair of events	logstash-filter-elapsed

cloudwatch	Aggregates and sends metric data to AWS CloudWatch	logstash-output-cloudwatch
csv	Writes events to disk in a delimited format	logstash-output-csv
datadog	Sends events to DataDogHQ based on Logstash events	logstash-output-datadog
datadog_metrics	Sends metrics to DataDogHQ based on Logstash events	logstash-output-datadog_metrics
elasticsearch	Stores logs in Elasticsearch	logstash-output-elasticsearch
email	Sends email to a specified address when output is received	logstash-output-email
exec	Runs a command for a matching event	logstash-output-exec
file	Writes events to files on disk	logstash-output-file
ganglia	Writes metrics to Ganglia's `gmond`	logstash-output-ganglia
gelf	Generates GELF formatted output for Graylog2	logstash-output-gelf
google_bigquery	Writes events to Google BigQuery	logstash-output-google_bigquery
google_cloud_storage	Writes events to Google Cloud Storage	logstash-output-google_cloud_storage
graphite	Writes metrics to Graphite	logstash-output-graphite
graphtastic	Sends metric data on Windows	logstash-output-graphtastic
hipchat	Writes events to HipChat	logstash-output-hipchat
http	Sends events to a generic HTTP or HTTPS endpoint	logstash-output-http

54

Logstash Grok

Grok is for parsing unstructured log data

text patterns into something that matches your logs.

Grok 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-401441/BCF-137/BTS-403;XYZ01N;ABC08;DEF081;BTS OPERATION DEGRADED;00 00 00 83 11 11;Processing

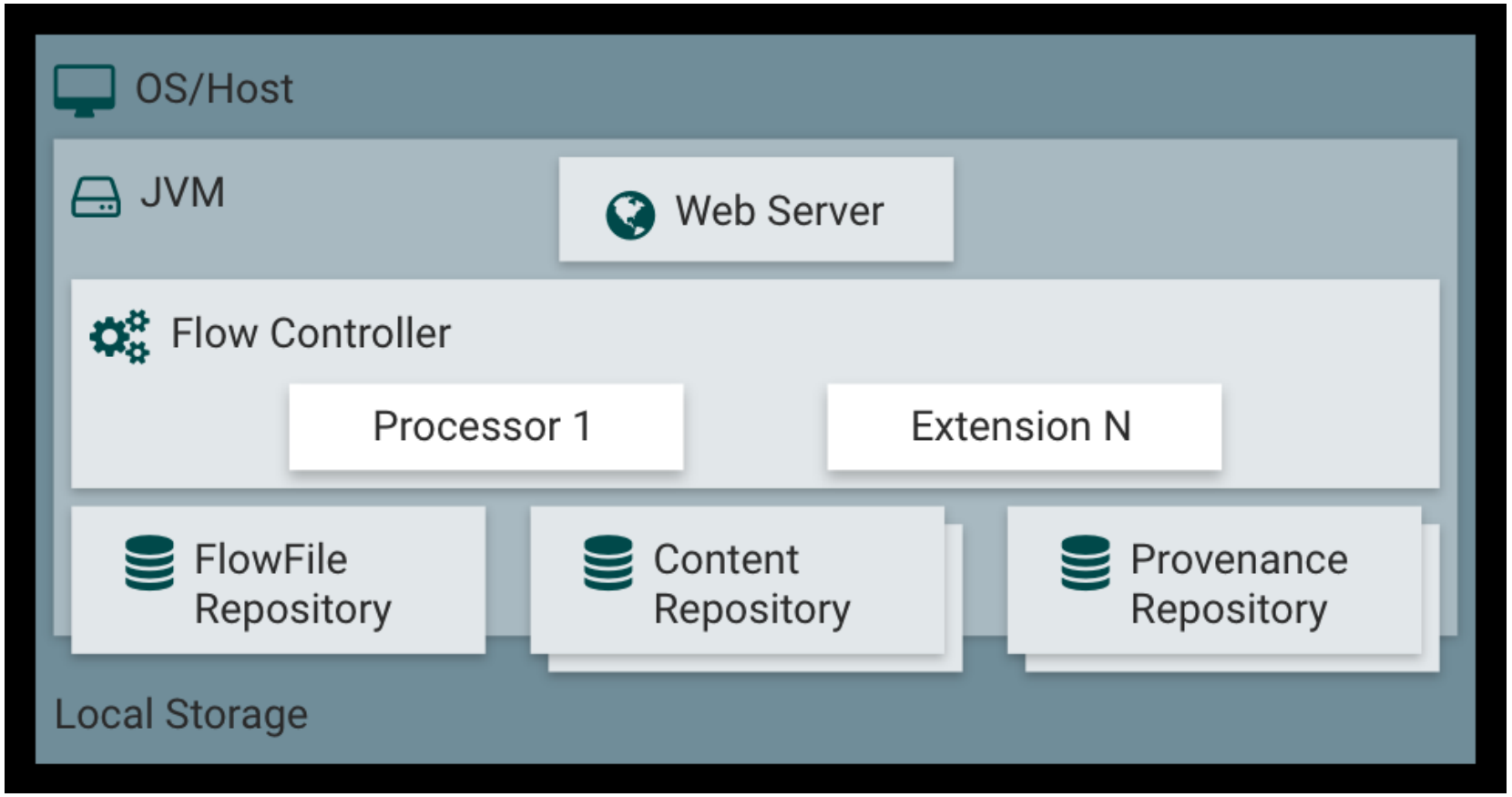
Simple Grok

```
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"
}
}
```

Apache Nifi

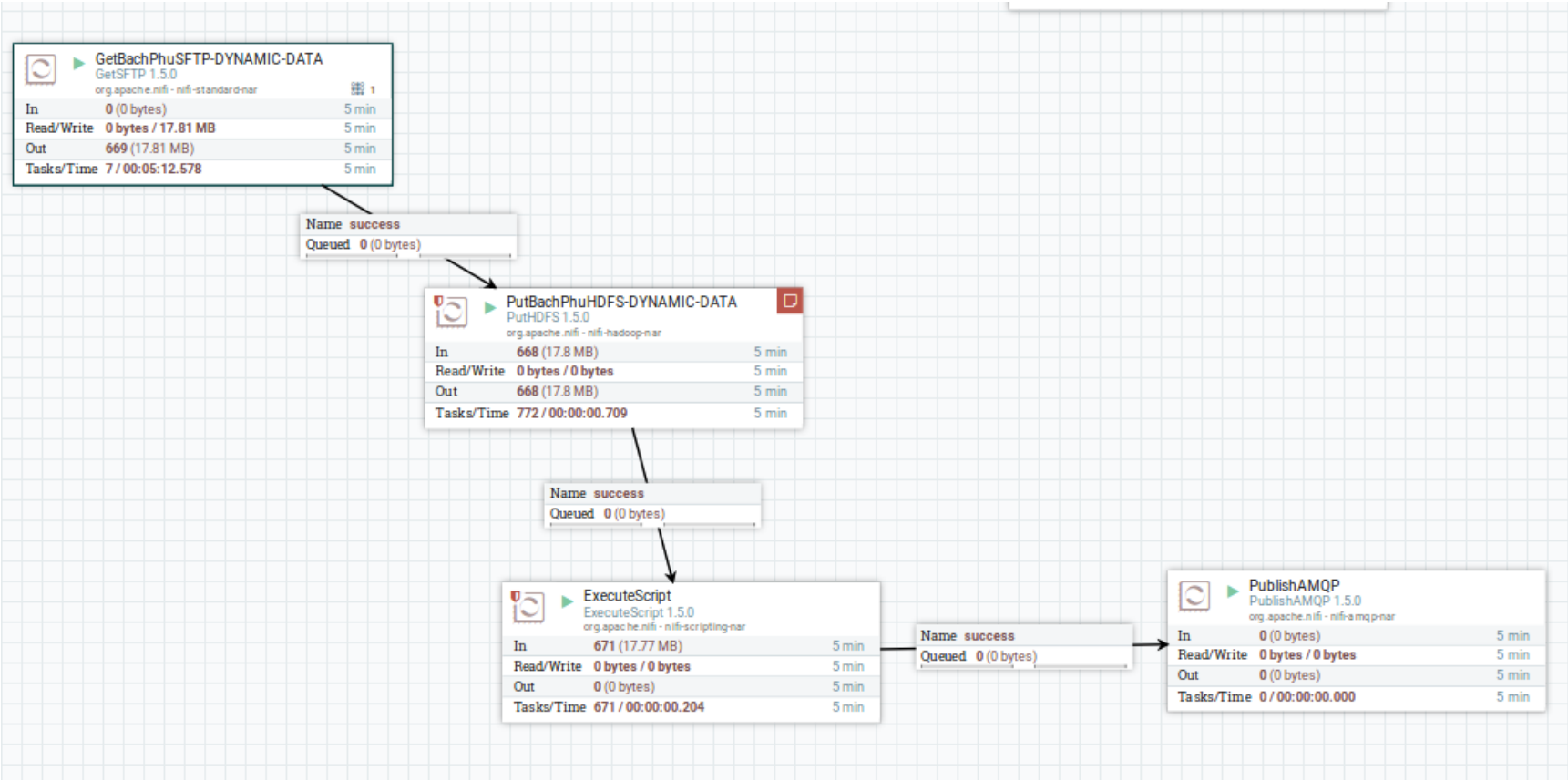
- From NSA
- http://nifi.apache.org/
- Main concepts:
  - Processor: components to handle data, such as download, store, transform, etc.
  - FlowFile: describes how different components are composed to create pipelines for data ingestion
  - Provenance (for data governance): see all usage records in detail

Apache Nifi



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

Example


GetBachPhuSFTP-DYNAMIC-DATA
GetSFTP 1.5.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 17.81 MB 5 min
Out 669 (17.81 MB) 5 min
Tasks/Time 7 / 00:05:12.578 5 min
Name success
Queued 0 (0 bytes)
PutBachPhuHDFS-DYNAMIC-DATA
PutHDFS 1.5.0
org.apache.nifi - nifi-hadoop-nar
In 668 (17.8 MB) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 668 (17.8 MB) 5 min
Tasks/Time 772 / 00:00:00.709 5 min
Name success
Queued 0 (0 bytes)
ExecuteScript
ExecuteScript 1.5.0
org.apache.nifi - nifi-scripting-nar
In 671 (17.77 MB) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 671 / 00:00:00.204 5 min
Name success
Queued 0 (0 bytes)
PublishAMQP
PublishAMQP 1.5.0
org.apache.nifi - nifi-amqp-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Processing

COMPLEX EVENT PROCESSING

Dataflow programming and streaming processing

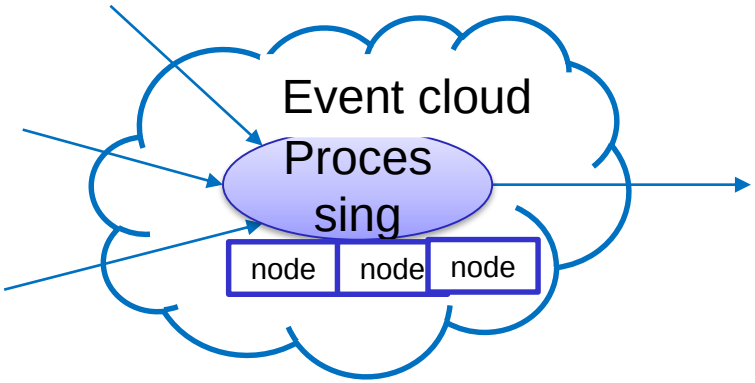
- Data exchange between tasks
  - Links in task graphs reflect data flows
- Streaming processing
  - Centralized or distributed (in terms of computing resources)
  - Various applications
  - CEP is just one type of applications of streaming processing

- Note: we will go further some advanced streaming processing in Lecture 5

Centralized versus distributed processing topology

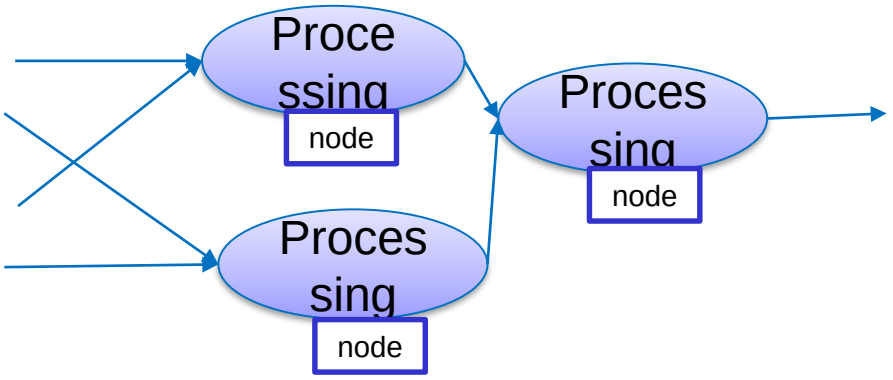
Two views: streams of events or cloud of events

Centralized processing



Usually only queries/patterns are written

Distributed processing

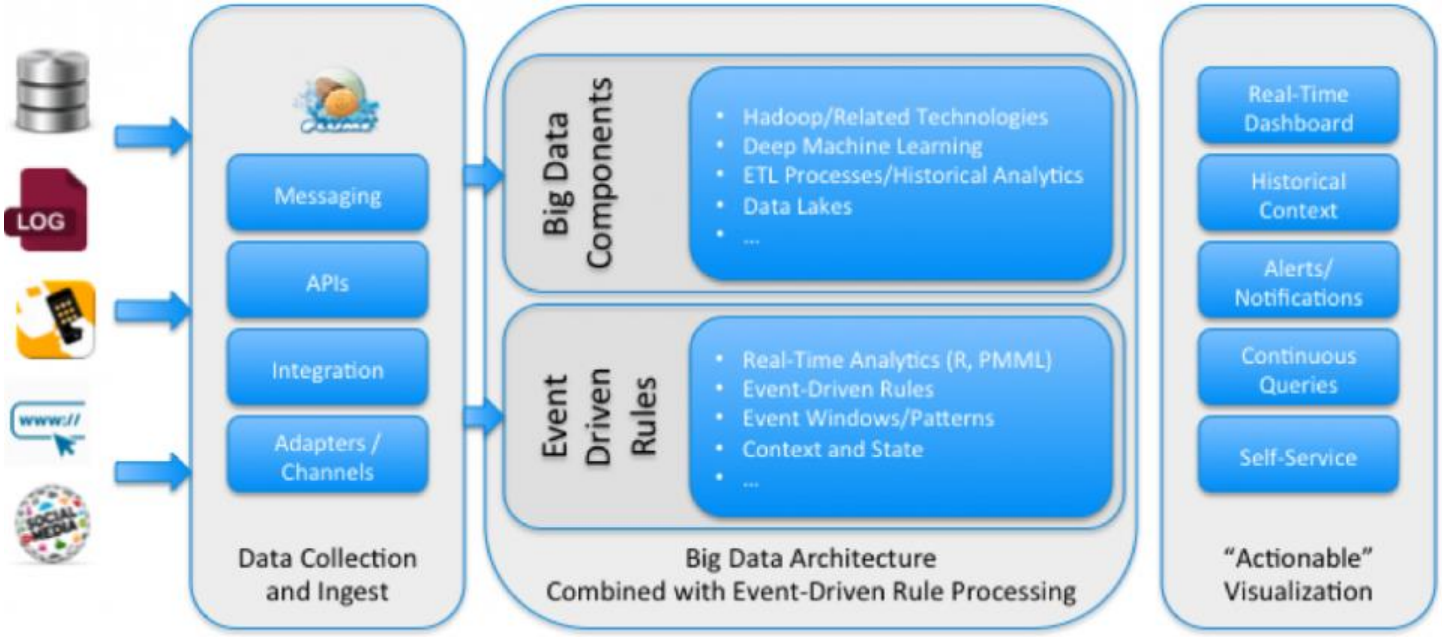


Code processing events and topologies need to be written

Goals of complex event processing

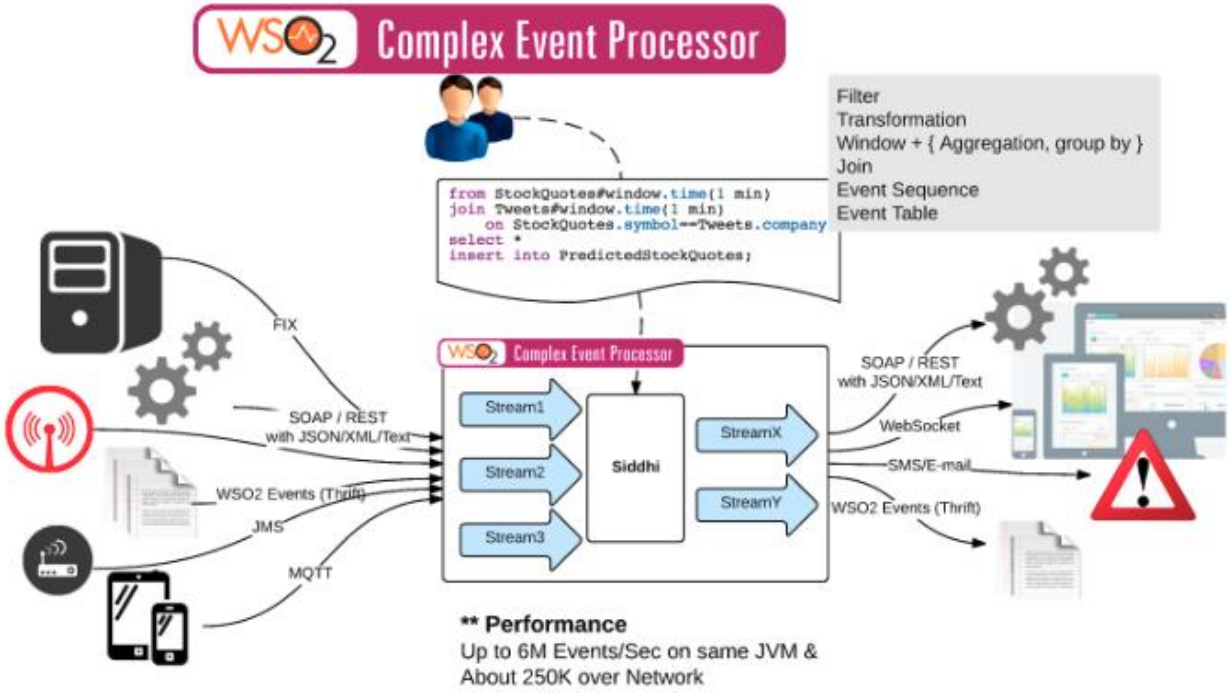
- Group and process events in a specific time (time) and space (size) constraints
  - Detect special situations
  - Finding correlation among events
  - Aggregation results
- Special case of streaming processing

TIBCO Systems



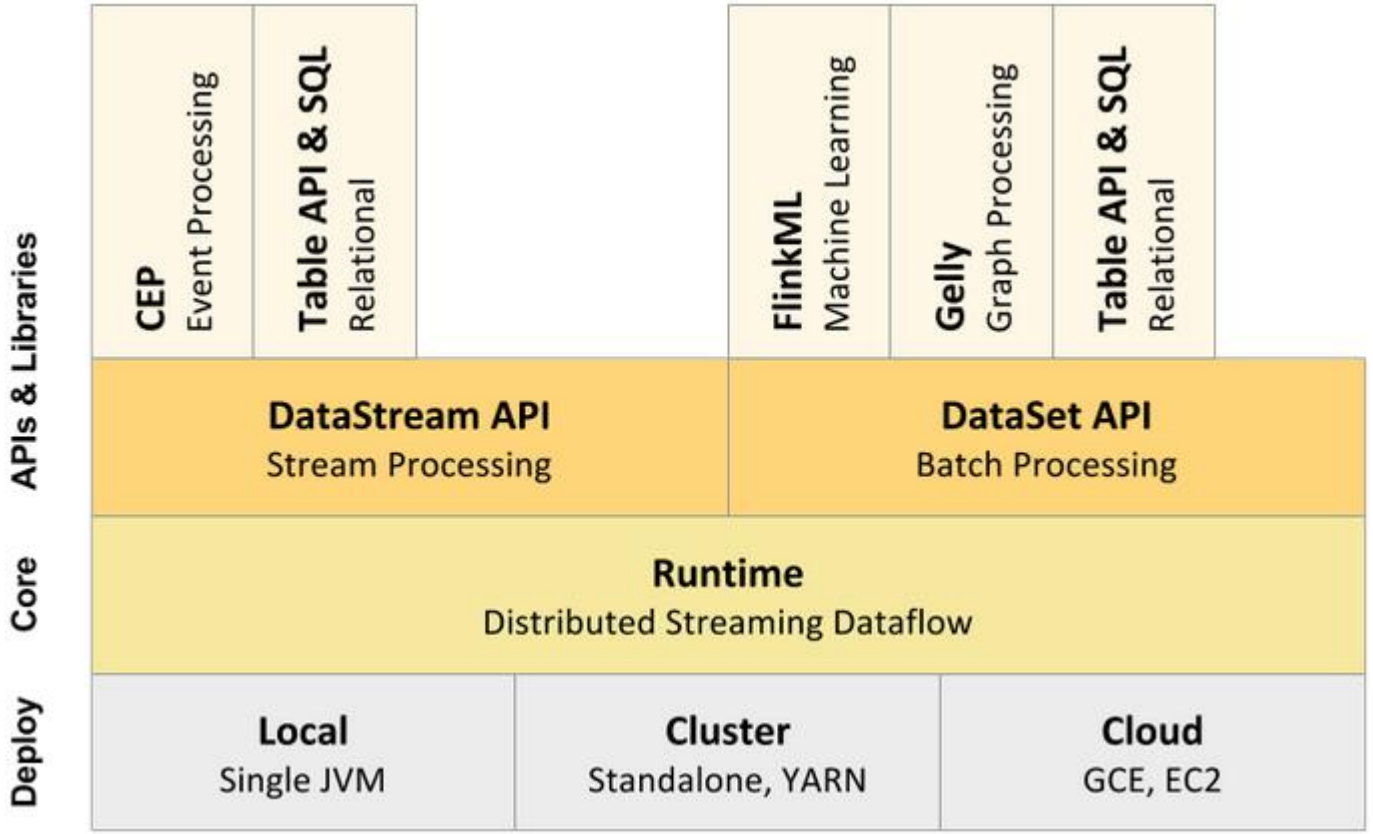
Source: http://www.tibco.com/blog/2015/10/05/how-to-extend-big-data-architectures-with-rules-and-visualization/

WSO2 Carbon CEP/Siddhi



Source: https://docs.wso2.com/display/CEP420

Apache Flink

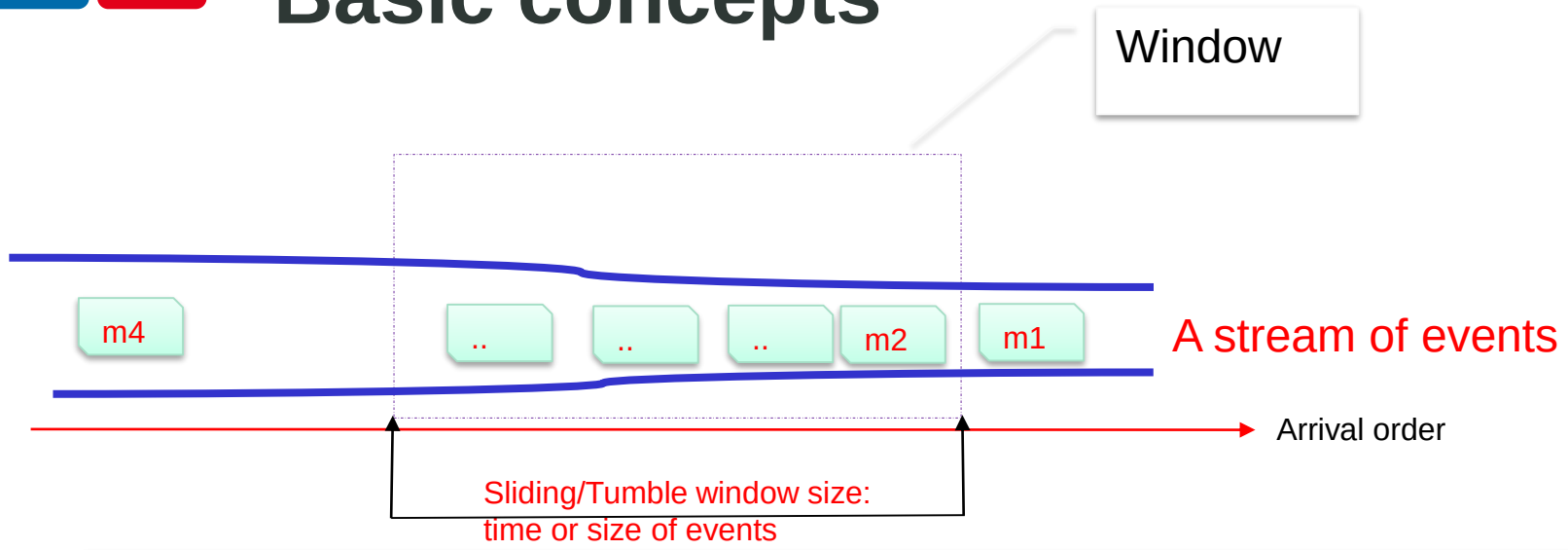


Source: https://ci.apache.org/projects/flink/flink-docs-release-1.4/internals/components.html

Common concept in these systems

- **The way to connect data streams and obtain events**
  - Focusing very much on connector concepts and well-defined event structures (e.g., can be described in XML, JSON, POJO)
  - Assume that existing systems push the data
- **The way to specify “analytics”**
  - Statements and how they are glued together to process flows of events
  - High-level, easy to use
- **The engine to process analytics requests**
  - Centralized in the view of the user → so the user does not have to program complex distributed applications
  - Underlying it might be complex (for scalability purposes)
- **The way to push results to external components**

Basic concepts



If we

- specify a set of conditions for the window and events within the window

then we can

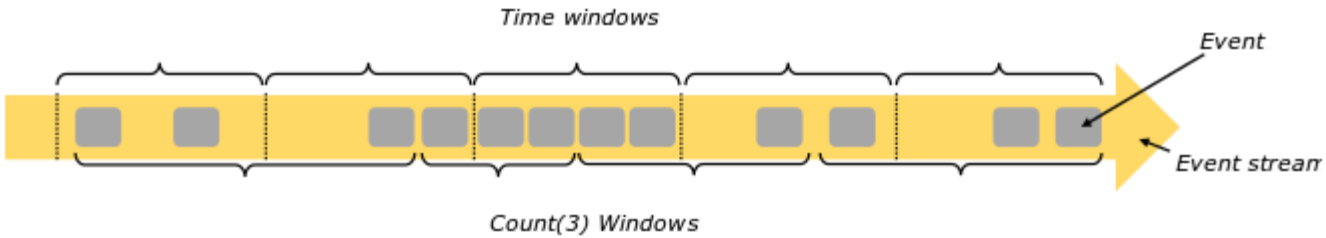
- get a set of events filtered from the window that match these conditions

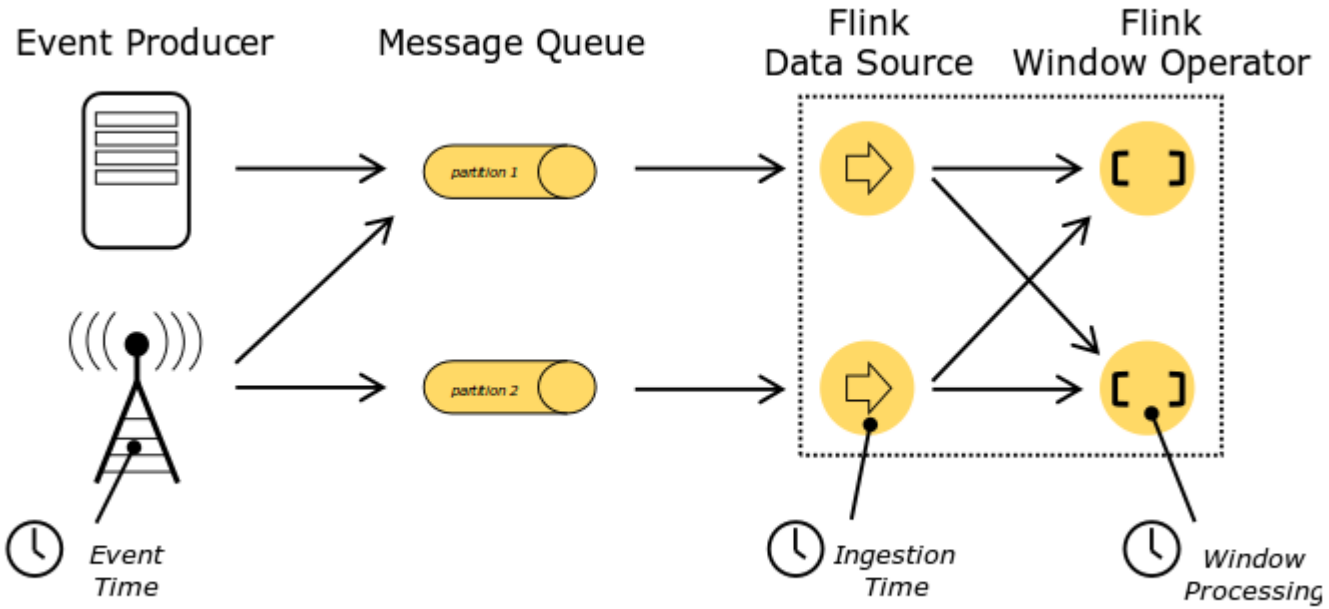
Conditions: can be specified using an SQL-alike language or pre-defined functions

Event Representation, Streams and Views

- Event sources: via MOM, files, different IO adapters/connectors, etc.
- Event representation & views
  - POJO (Plain Old Java Object), Map, Object-array, XML
  - SQL-alike tables
- Event Stream
  - Events ordered based on their arrival times
- Event Sink
  - A component receiving events via its listener that declares some statements on interesting events

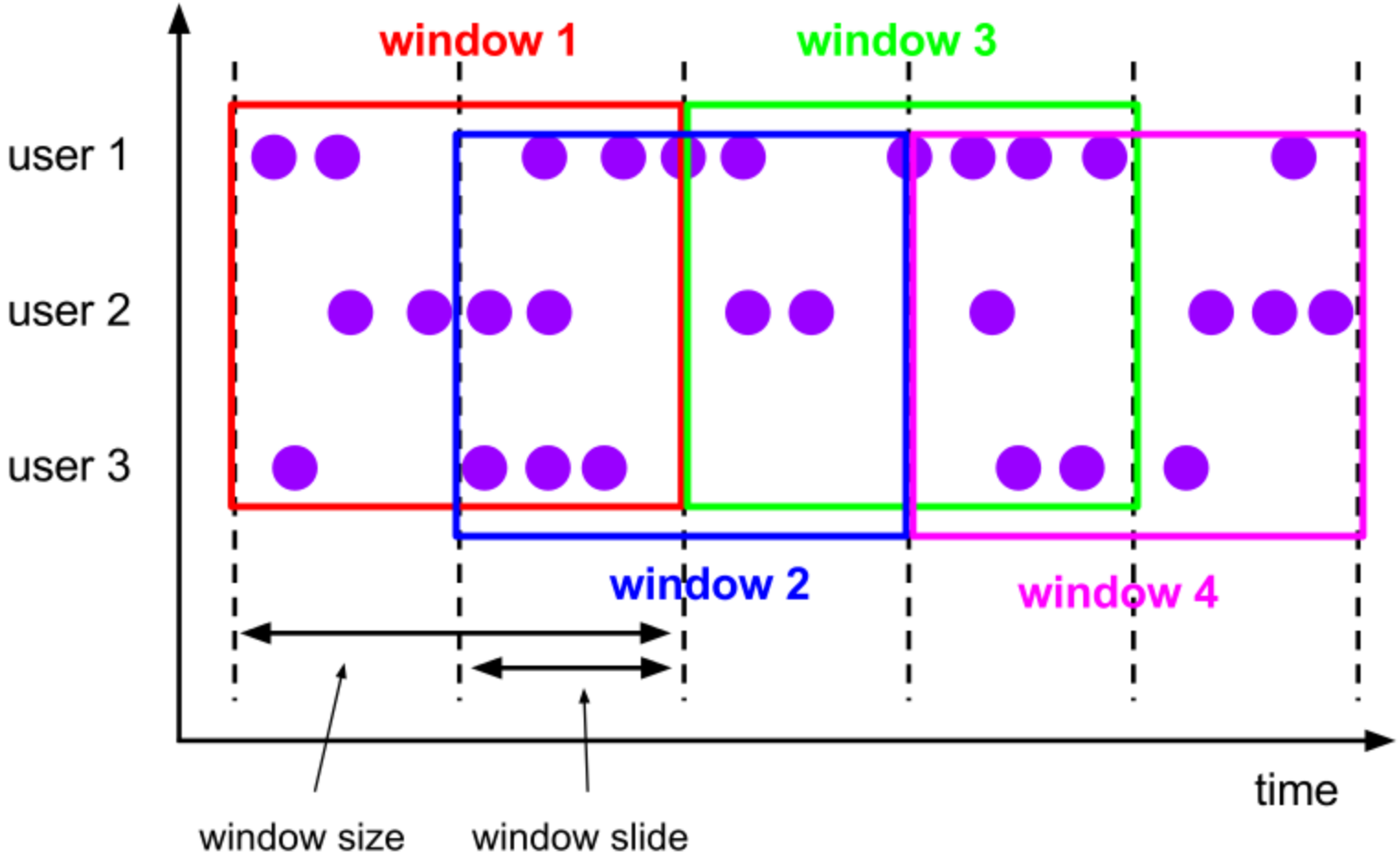
Windows and Times





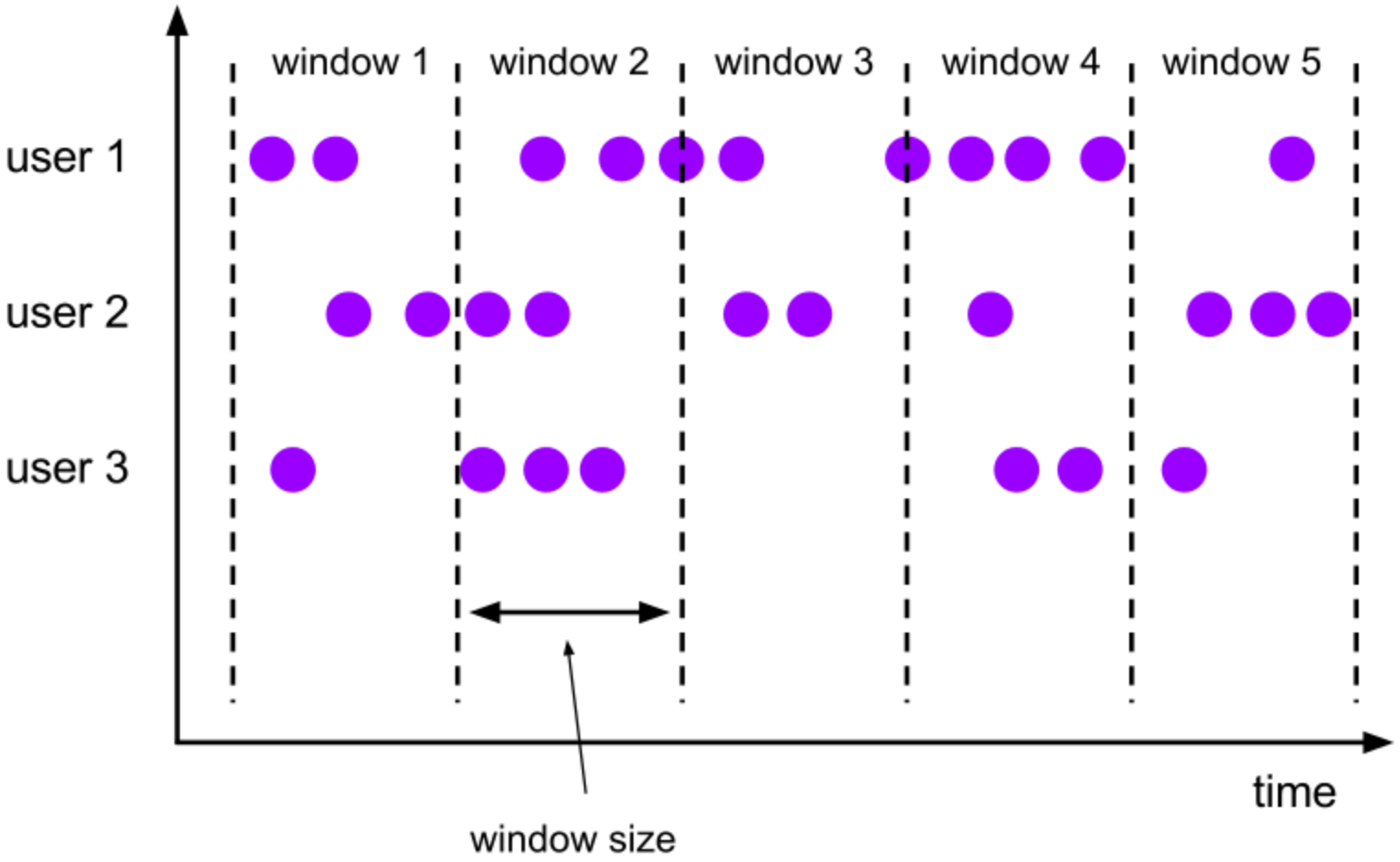
Source: https://ci.apache.org/projects/flink/flink-docs-release-1.4/concepts/programming-model.html

Window size and slide



Source: https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/operators/windows

Batch/Tumbling Windows



Source: https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/operators/windows.html

Flink CEP Patterns

Pattern Operation	Description
begin(#name)	Defines a starting pattern: `Pattern<Event, ?> start = Pattern.<Event>begin("start");`

begin(#pattern_sequence)	Defines a starting pattern:
	`Pattern<Event, ?> start = Pattern.<Event>begin(`
	`  Pattern.<Event>begin("start").where(...).followedBy("middle").where(...)`
	`);`

next(#name)	Appends a new pattern. A matching event has to directly succeed the previous matching event (strict contiguity): `Pattern<Event, ?> next = start.next("middle");`

next(#pattern_sequence)	Appends a new pattern. A sequence of matching events have to directly succeed the previous matching event (strict contiguity):
	`Pattern<Event, ?> next = start.next(`
	`  Pattern.<Event>begin("start").where(...).followedBy("middle").where(...)`
	`);`

followedBy(#name)	Appends a new pattern. Other events can occur between a matching event and the previous matching event (relaxed contiguity): `Pattern<Event, ?> followedBy = start.followedBy("middle");`

followedBy(#pattern_sequence)	Appends a new pattern. Other events can occur between a sequence of matching events and the previous matching event (relaxed contiguity):
	`Pattern<Event, ?> followedBy = start.followedBy(`
	`  Pattern.<Event>begin("start").where(...).followedBy("middle").where(...)`
	`);`

Source: https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/cep.html

Flink CEP Patterns

Pattern Operation	Description

where(condition)	Defines a condition for the current pattern. To match the pattern, an event must satisfy the condition. Multiple consecutive where() clauses lead to their conditions being ANDed:
	`pattern.where(new IterativeCondition<Event>() {`
	`  @Override`
	`  public boolean filter(Event value, Context ctx) throws Exception {`
	`  return ... // some condition`
	`  }`
	`});`

or(condition)	Adds a new condition which is ORed with an existing one. An event can match the pattern only if it passes at least one of the conditions:
	`pattern.where(new IterativeCondition<Event>() {`
	`  @Override`
	`  public boolean filter(Event value, Context ctx) throws Exception {`
	`  return ... // some condition`
	`  }`
	`}).or(new IterativeCondition<Event>() {`
	`  @Override`
	`  public boolean filter(Event value, Context ctx) throws Exception {`
	`  return ... // alternative condition`
	`  }`
	`});`

until(condition)	Specifies a stop condition for a looping pattern. Meaning if event matching the given condition occurs, no more events will be accepted into the pattern.
	Applicable only in conjunction with `oneOrMore()`
	NOTE: It allows for cleaning state for corresponding pattern on event-based condition.
	`pattern.oneOrMore().until(new IterativeCondition<Event>() {`
	`  @Override`
	`  public boolean filter(Event value, Context ctx) throws Exception {`
	`  return ... // alternative condition`
	`  }`
	`});`

Source: https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/libs/cep.html

Example with Base Transceiver Station

Data

station_id,datapoint_id,alarm_id,event_time,value,valueThreshold,
isActive,storedtime
1161115016,121,308,2017-02-18 18:28:05 UTC,240,240,false,
1161114050,143,312,2017-02-18 18:56:20 UTC,28.5,28,true,
1161115040,141,312,2017-02-18 18:22:03 UTC,56.5,56,true,
1161114008,121,308,2017-02-18 18:34:09 UTC,240,240,false,
1161115040,141,312,2017-02-18 18:20:49 UTC,56,56,false,

Simple example

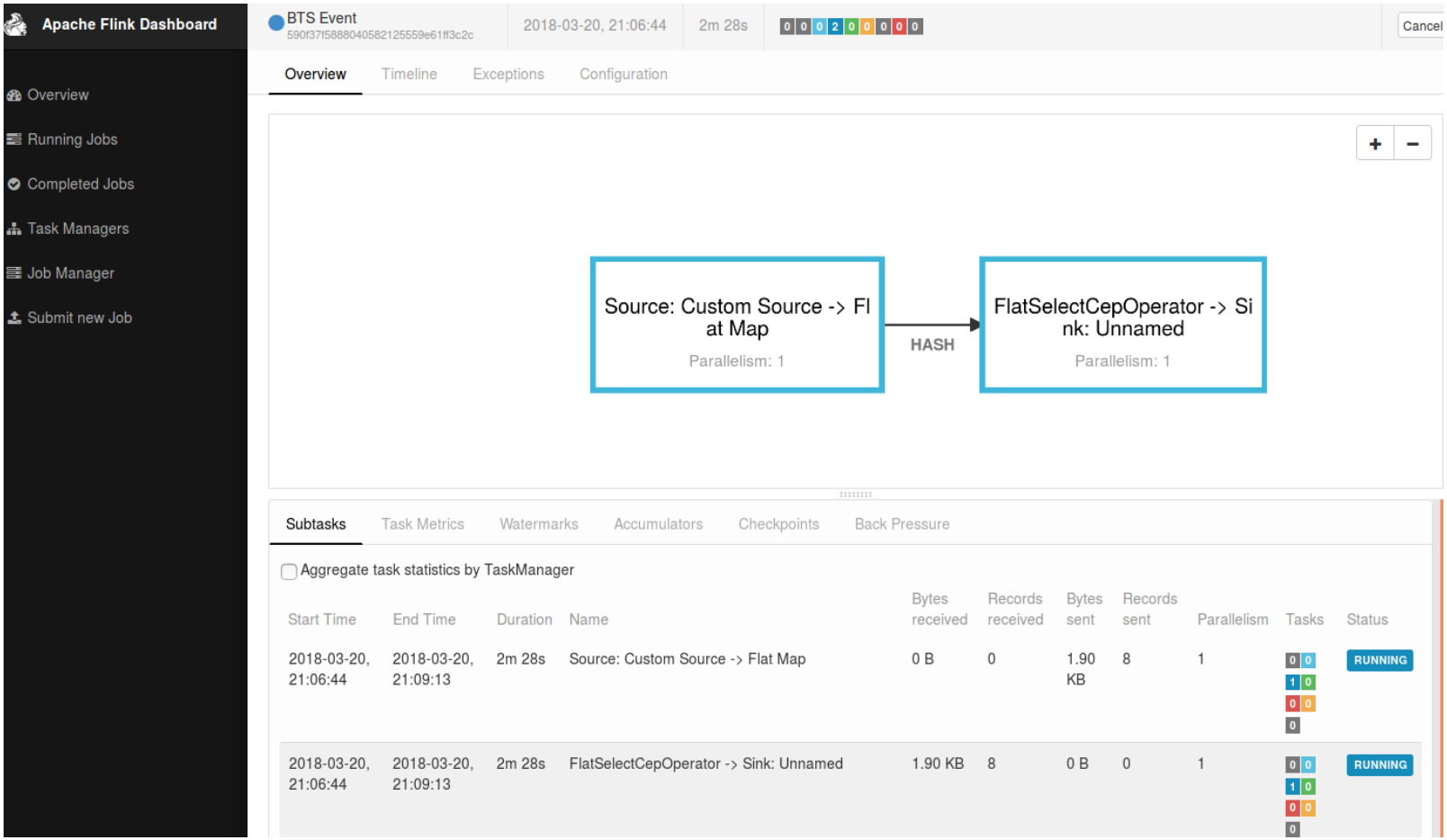
```
final RMQConnectionConfig connectionConfig = new RMQConnectionConfig.Builder()
        .setUri(args[0])
        .build();
final DataStream<String> stream = env
        .addSource(new RMQSource<String>(
                connectionConfig,
                args[1],
                false,
                new SimpleStringSchema()))
        .setParallelism(1);

DataStream<AlarmEvent> btsStream;
btsStream = stream.flatMap(new BTSParser());

Pattern<AlarmEvent, ?> pattern = Pattern.<AlarmEvent>begin("start").where(new SimpleCondition<AlarmEvent>() {
    @Override
    public boolean filter(AlarmEvent value) throws Exception {
        System.out.println("Start event");
        return value.alarm_id.equals("308");
    }
}).next("middle")
        .followedBy("end").where(new SimpleCondition<AlarmEvent>() {
    @Override
    public boolean filter(AlarmEvent value) throws Exception {
        System.out.println("End event");
        return value.alarm_id.equals("303");
    }
});//.within(Time.seconds(300));

PatternStream<AlarmEvent> patternStream;
patternStream = CEP.pattern(btsStream.keyBy(new AlarmKeySelector()), pattern);

final DataStream<String> alerts = patternStream.select(new PatternSelectFunction<AlarmEvent, String>() {
    @Override
    public String select(Map<String, List<AlarmEvent>> pattern) {
        AlarmEvent first = pattern.get("start").get(0);
        AlarmEvent second = pattern.get("end").get(0);
        final String result ="Detected: " + first.toString() + " --> " + second.toString();
        System.out.println("FOUND: "+result);
        return result;
    }

});
RMQSink<String> sink =new RMQSink<String>(
                connectionConfig,
                args[2],
                new SimpleStringSchema());

alerts.addSink(sink);
```

AMQP Connector

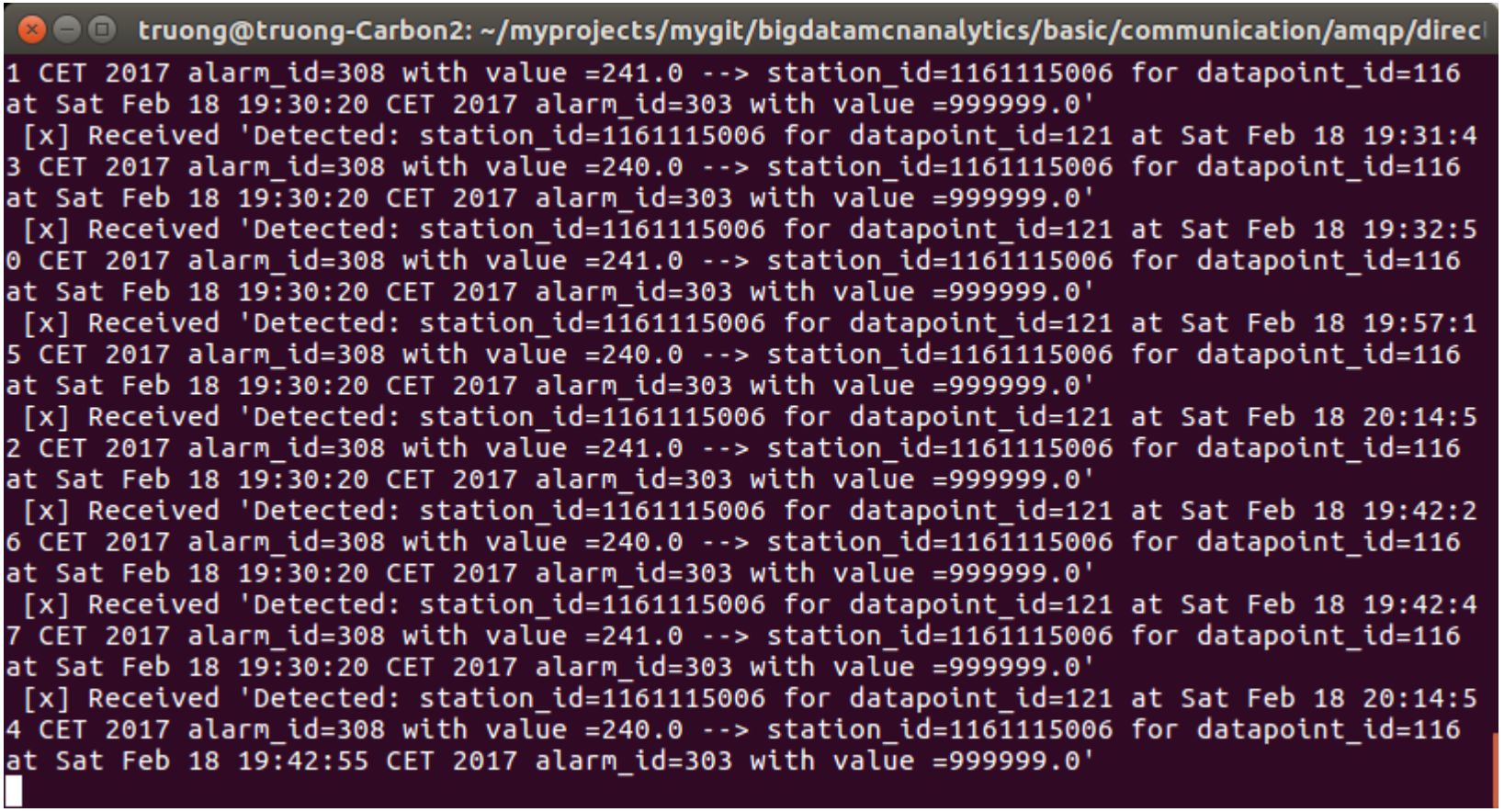
Patterns

Output

Monitoring


Apache Flink Dashboard
BTS Event
590f37f588804058212559e61ff3c2c
2018-03-20, 21:06:44
2m 28s
Cancel
Overview
Timeline
Exceptions
Configuration
Overview
Running Jobs
Completed Jobs
Task Managers
Job Manager
Submit new Job
Source: Custom Source -> Fl at Map
Parallelism: 1
HASH
FlatSelectCepOperator -> Si nk: Unnamed
Parallelism: 1
Subtasks
Task Metrics
Watermarks
Accumulators
Checkpoints
Back Pressure
Aggregate task statistics by TaskManager
Start Time
End Time
Duration
Name
Bytes received
Records received
Bytes sent
Records sent
Parallelism
Tasks
Status
2018-03-20, 21:06:44
2018-03-20, 21:09:13
2m 28s
Source: Custom Source -> Flat Map
0 B
0
1.90 KB
8
1
RUNNING
2018-03-20, 21:06:44
2018-03-20, 21:09:13
2m 28s
FlatSelectCepOperator -> Sink: Unnamed
1.90 KB
8
0 B
0
1
RUNNING

Results


truong@truong-Carbon2: ~/myprojects/mygit/bigdatamcnanalytics/basic/communication/amqp/direc
1 CET 2017 alarm_id=308 with value =241.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:30:20 CET 2017 alarm_id=303 with value =999999.0'
[x] Received 'Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 19:31:4
3 CET 2017 alarm_id=308 with value =240.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:30:20 CET 2017 alarm_id=303 with value =999999.0'
[x] Received 'Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 19:32:5
0 CET 2017 alarm_id=308 with value =241.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:30:20 CET 2017 alarm_id=303 with value =999999.0'
[x] Received 'Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 19:57:1
5 CET 2017 alarm_id=308 with value =240.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:30:20 CET 2017 alarm_id=303 with value =999999.0'
[x] Received 'Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 20:14:5
2 CET 2017 alarm_id=308 with value =241.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:30:20 CET 2017 alarm_id=303 with value =999999.0'
[x] Received 'Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 19:42:2
6 CET 2017 alarm_id=308 with value =240.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:30:20 CET 2017 alarm_id=303 with value =999999.0'
[x] Received 'Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 19:42:4
7 CET 2017 alarm_id=308 with value =241.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:30:20 CET 2017 alarm_id=303 with value =999999.0'
[x] Received 'Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 20:14:5
4 CET 2017 alarm_id=308 with value =240.0 --> station_id=1161115006 for datapoint_id=116
at Sat Feb 18 19:42:55 CET 2017 alarm_id=303 with value =999999.0'

SQL-alike CEP

- We can register/view stream as a table (like SQL)
- Then apply SQL-alike statements with windows for detecting events and patterns
- Tools: Esper, WSO2, and certain streaming databases

Example of WSO2 Siddhi

Pass-through

```
from <stream-name>
select ( {<attribute-name>}| '*'|)
insert into <stream-name>
```

Filters

```
from <stream-name> {<conditions>}
select ( {<attribute-name>}| '*'|)
insert into <stream-name>
```

Windows

```
from <stream-name> {<conditions>}#window.<window-name>(<parameters>)
select ( {<attribute-name>} | '*' |)
insert [<output-type>] into <stream-name>
```

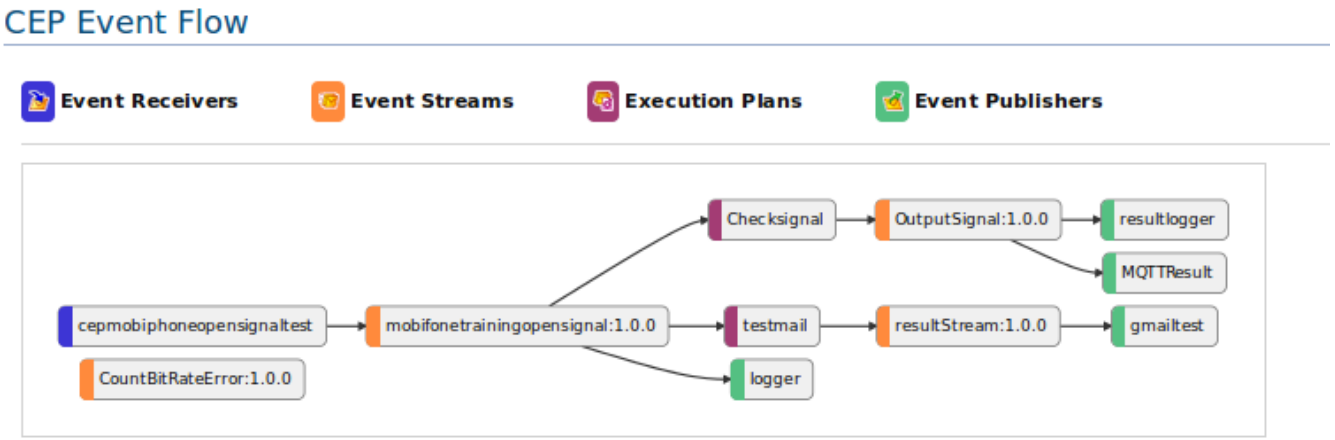
Source: https://docs.wso2.com/display/CEP420/SiddhiQL+Guide+3.1

SQL-alike conditions

```
@Import('mobifonetrainingopensignal:1.0.0')
define stream inStream (meta_USERPHONE int, meta_TIME long, correlation_lat float,
    correlation_lon float, GSM_BIT_ERROR_RATE float, GSM_SIGNAL_STRENGTH float,
    LOC_ACCURACY float, LOC_SPEED float);



@Export('OutputSignal:1.0.0')
define stream OutputSignal (avgSignalStrength double, avgBitRateError double);


from inStream#window.lengthBatch(5)
select avg(GSM_SIGNAL_STRENGTH) as avgSignalStrength, avg(GSM_BIT_ERROR_RATE) as
    avgBitRateError
insert  into OutputSignal;
```

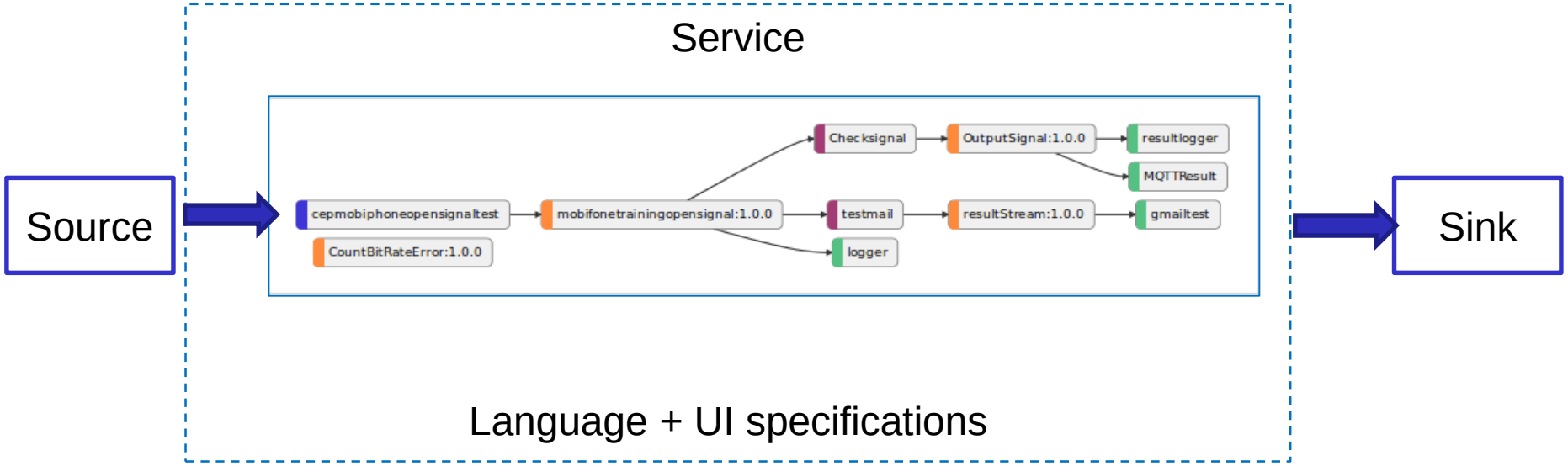
Put things together

A data pipeline of stream receivers → event processor → event publishers



82

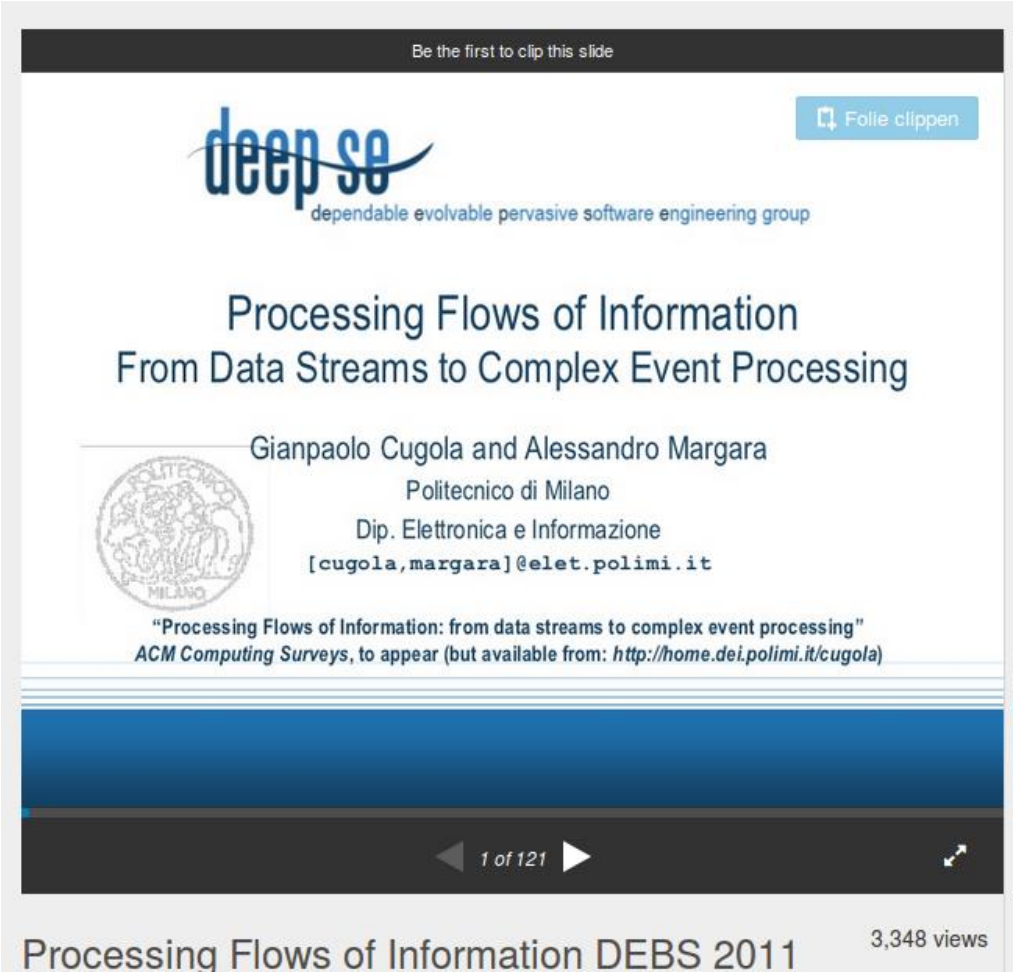
Example with WSO2 Carbon CEP



83

Get a high-level view

Check:
http://de.slideshare.net/alessandro_margara/processing-flows-of-information-debs-2011



Partially covered in Lecture 5

BEYOND BASIC MESSAGE PROCESSING

Datalake with messaging

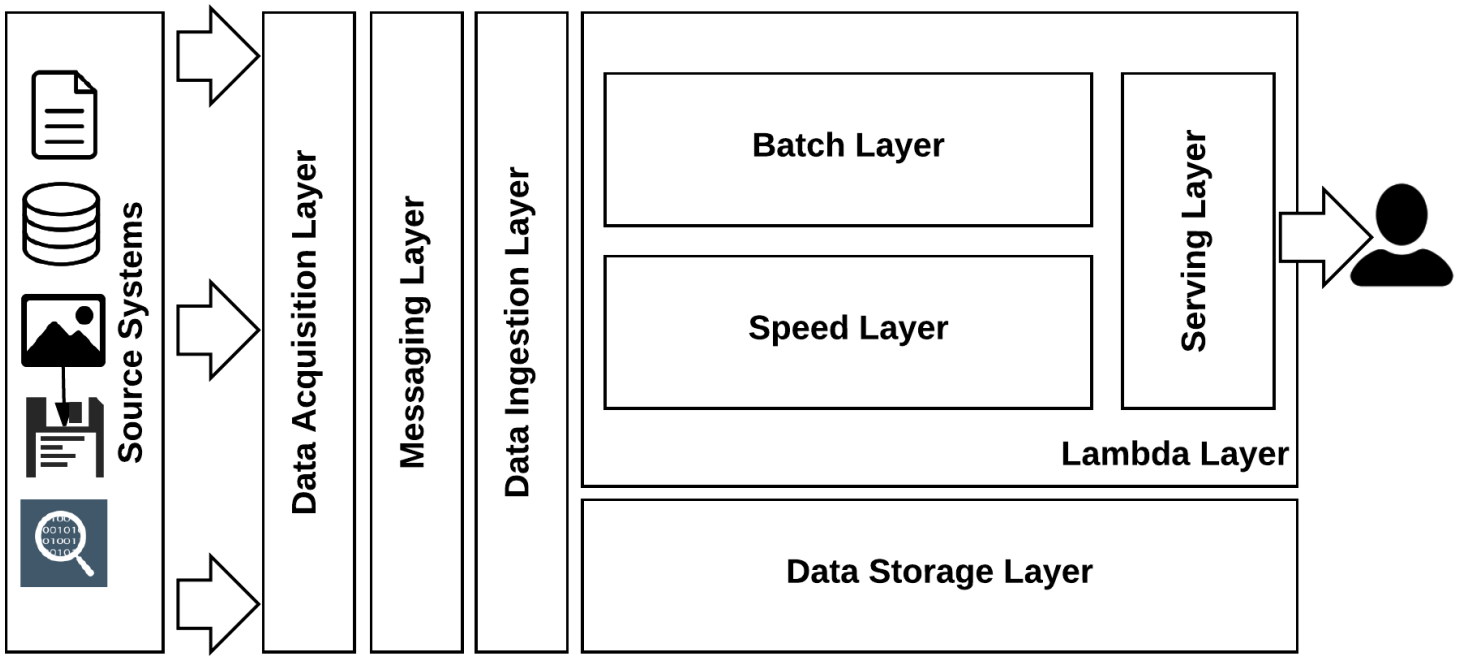
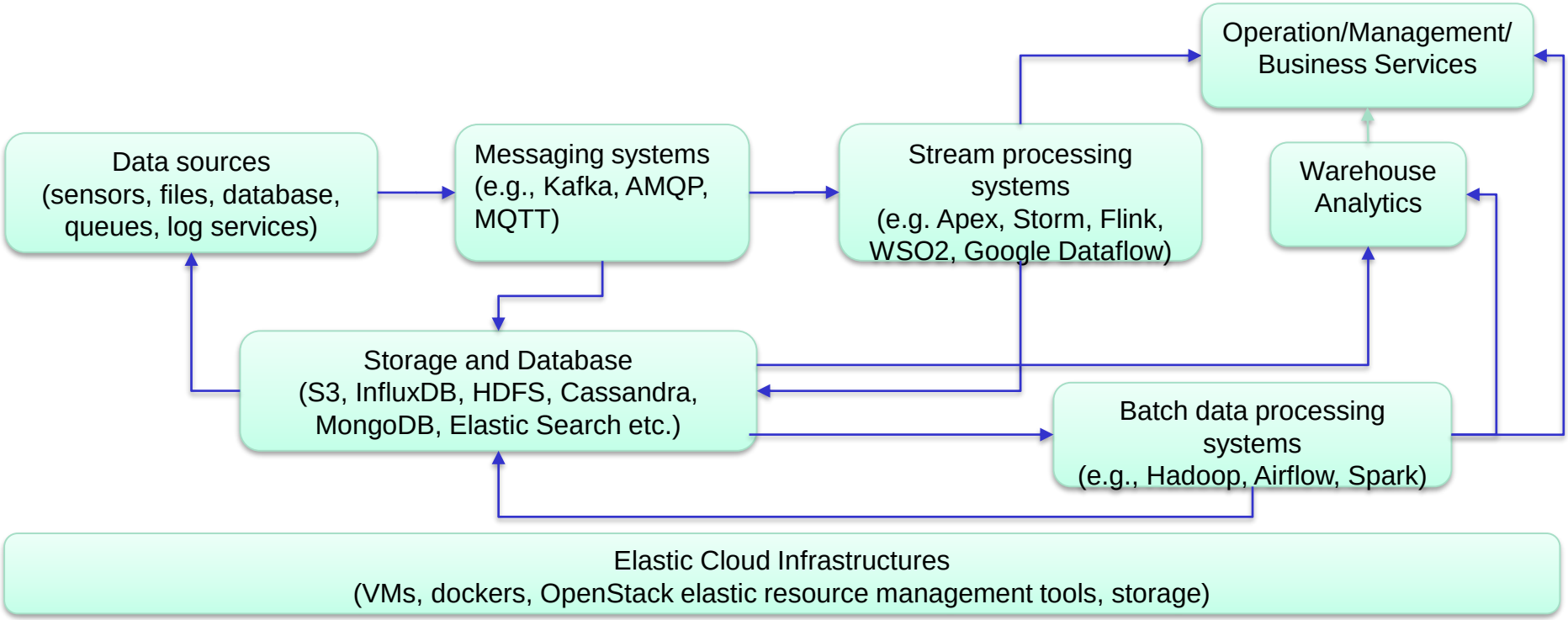


Figure source: Data Lake for Enterprises by Pankaj Misra; Tomcy John Published by Packt Publishing, 2017

Cloud services and big data analytics

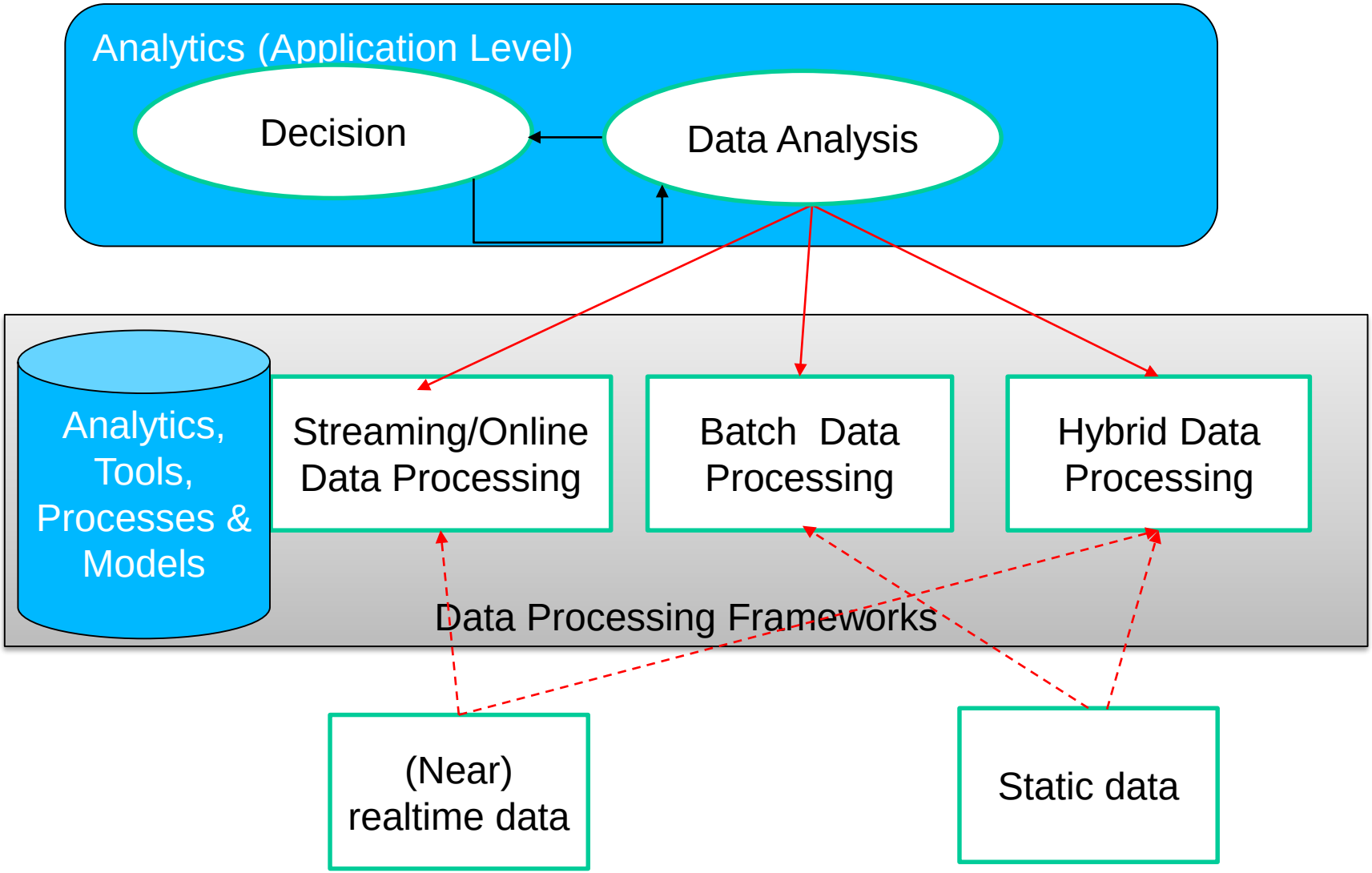


Data Processing Framework

- Batch processing
  - Mapreduce/Hadoop
  - Scientific workflows
- (Near) realtime streaming processing
  - Flink, Apex, Kafka SQL, Storm
- Hybrid data processing
  - Summingbird, Apache Kylin
  - Impala, Storm-YARN
  - Apache Spark

Take a short read: http://www.infoq.com/articles/stream-processing-hadoop

Conceptual View



Recap

So how can you use messaging techniques for complex distributed applications/systems?

- Reactive patterns
- Asynchronous communications
- Large-scale integration
- Big data
- ?

Further materials

- https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_MRG/1.1/html/Messaging_User_Guide/sect-Messaging_User_Guide-Introduction_to_RHM-The_AMQP_0_10_Model.html
- Java Message Service: http://www.oracle.com/technetwork/java/index-jsp-142945.html
- Java Message Service specification, version 2.0, available from: http://jcp.org/en/jsr/detail?id=343
- http://kafka.apache.org
- https://camel.apache.org/enterprise-integration-patterns.html
- http://www.eaipatterns.com
- http://docs.oracle.com/javaee/7/tutorial/doc/home.htm
- http://docs.oracle.com/cd/E13157_01/wlevs/docs30/epl_guide/index.html
- http://www.espertech.com/esper/documentation.php
- Miyuru Dayarathna and Toyotaro Suzumura. 2013. A performance analysis of system s, s4, and esper via two level benchmarking. In Proceedings of the 10th international conference on Quantitative Evaluation of Systems (QEST'13), Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio (Eds.). Springer-Verlag, Berlin, Heidelberg, 225-240. DOI=10.1007/978-3-642-40196-1_19 http://dx.doi.org/10.1007/978-3-642-40196-1_19
- https://code.facebook.com/posts/872547912839369/improving-facebook-s-performance-on-android-with-flatbuffers/

Thanks for your attention

Hong-Linh Truong
Faculty of Informatics, TU Wien
hong-linh.truong@tuwien.ac.at
http://www.infosys.tuwien.ac.at/staff/truong