Messaging: Basic Exchange, Processing and Transformation Models and Tools

Hong-Linh Truong
Distributed Systems Group, TU Wien

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong
@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.

What You Know vs How much you know about it
What You Know
Everything
Nothing
Undergrad
Master's
Ph.D.
Oops! You overshot it!
A Little
A Lot
How much you know about it
JORGE CHAM © 2008
WWW.PHDCOMICS.COM

Further advanced topics will be covered in Lecture 5

Overview

STREAMING MESSAGE-ORIENTED PROGRAMMING

Data stream programming

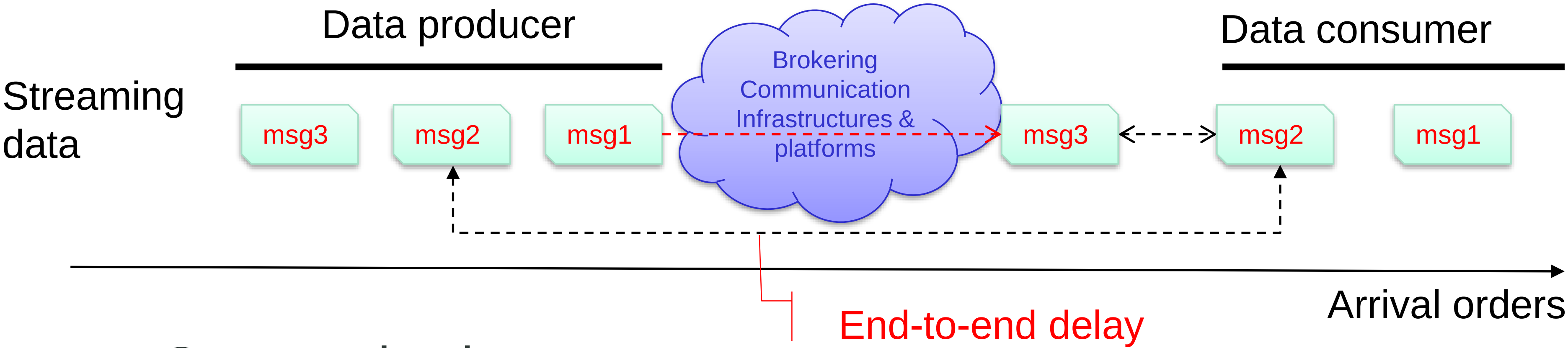
Data stream: a sequence/flow of data units

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)
  - Discrete media (e.g., stock market events, twitter events, system monitoring events, notifications)

Some key issues



- Communication
  - Which techniques can we use to support the communication (send, receive, route, storage, etc.)
- Data processing
  - Within the brokering communication infrastructures and platforms
  - Within the producer and the consumer
  - Interoperability issues: message format, etc.
  - Performance issues: rates, intervals, delay, etc.

Message-oriented Middleware (MOM)

- Discrete media data units
  - Data units are structured messages (maybe ordered by time stamps)
- 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

Communication models

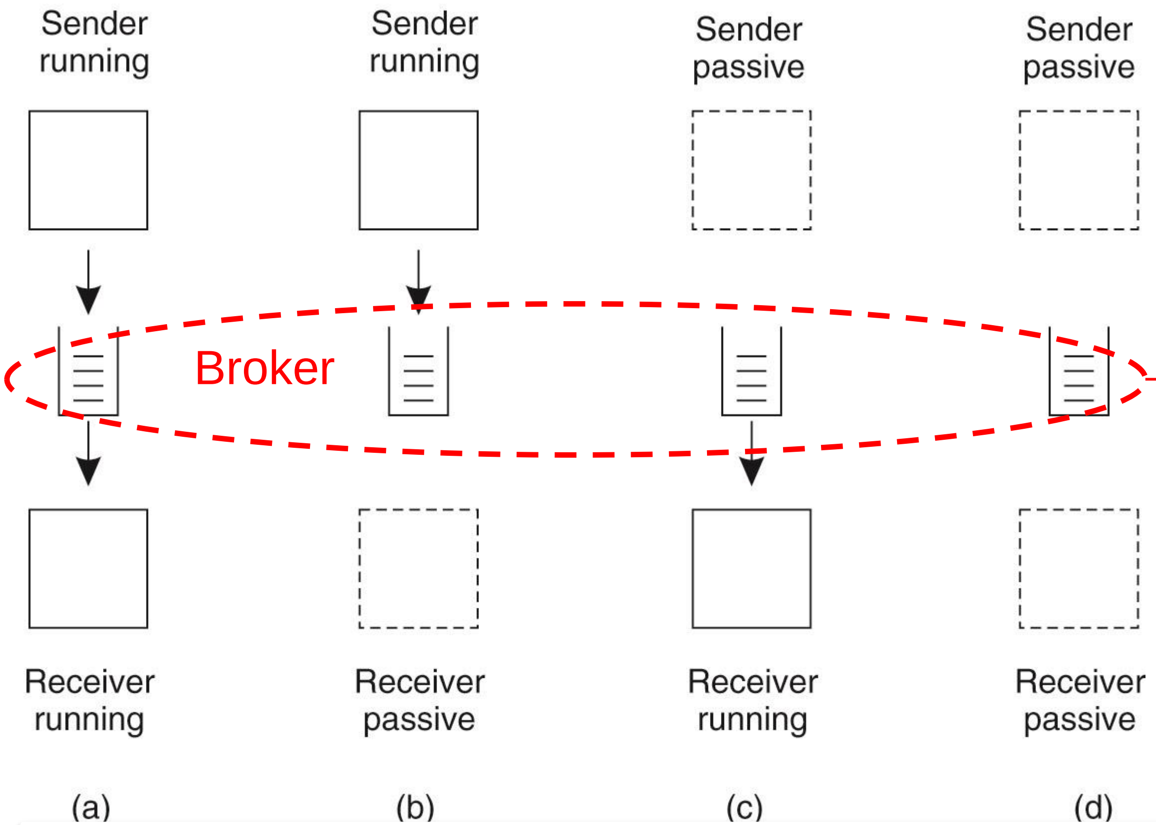
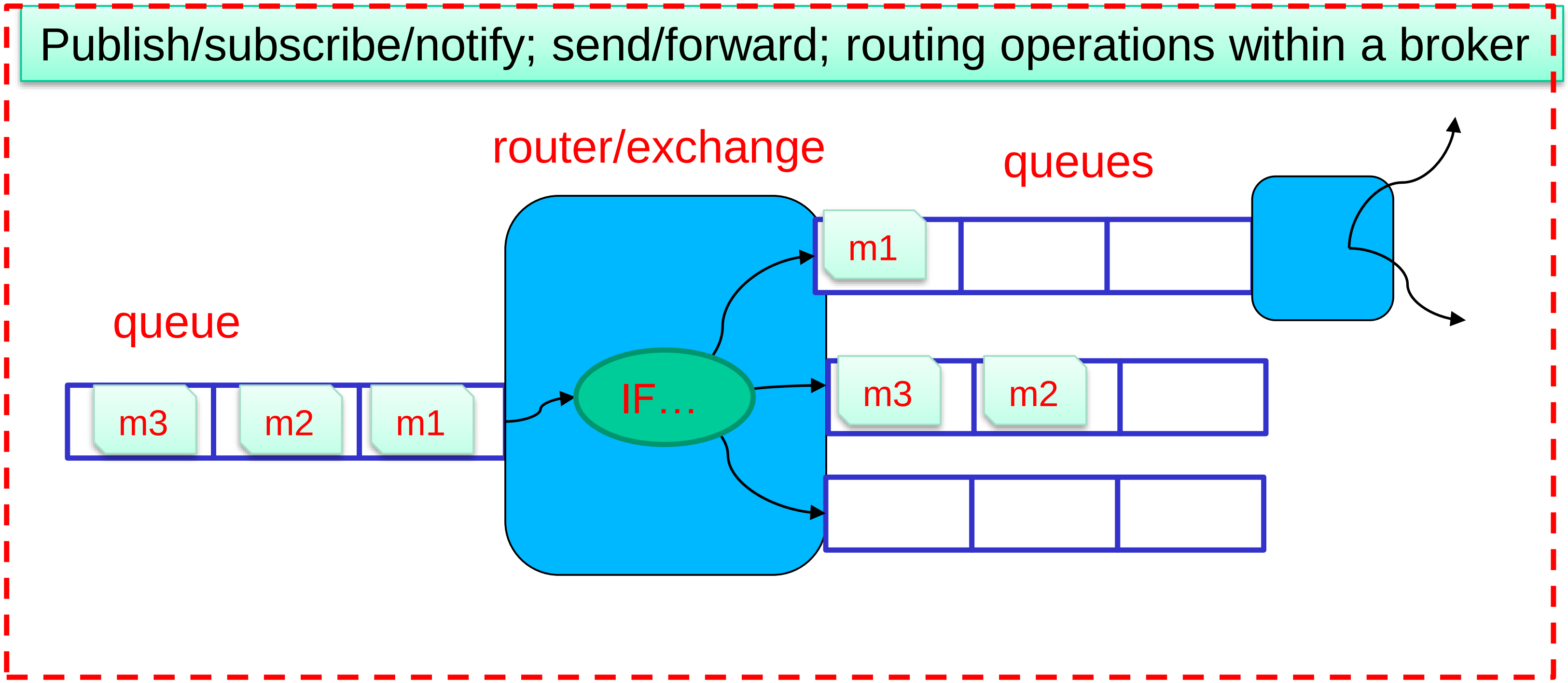


Fig source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

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

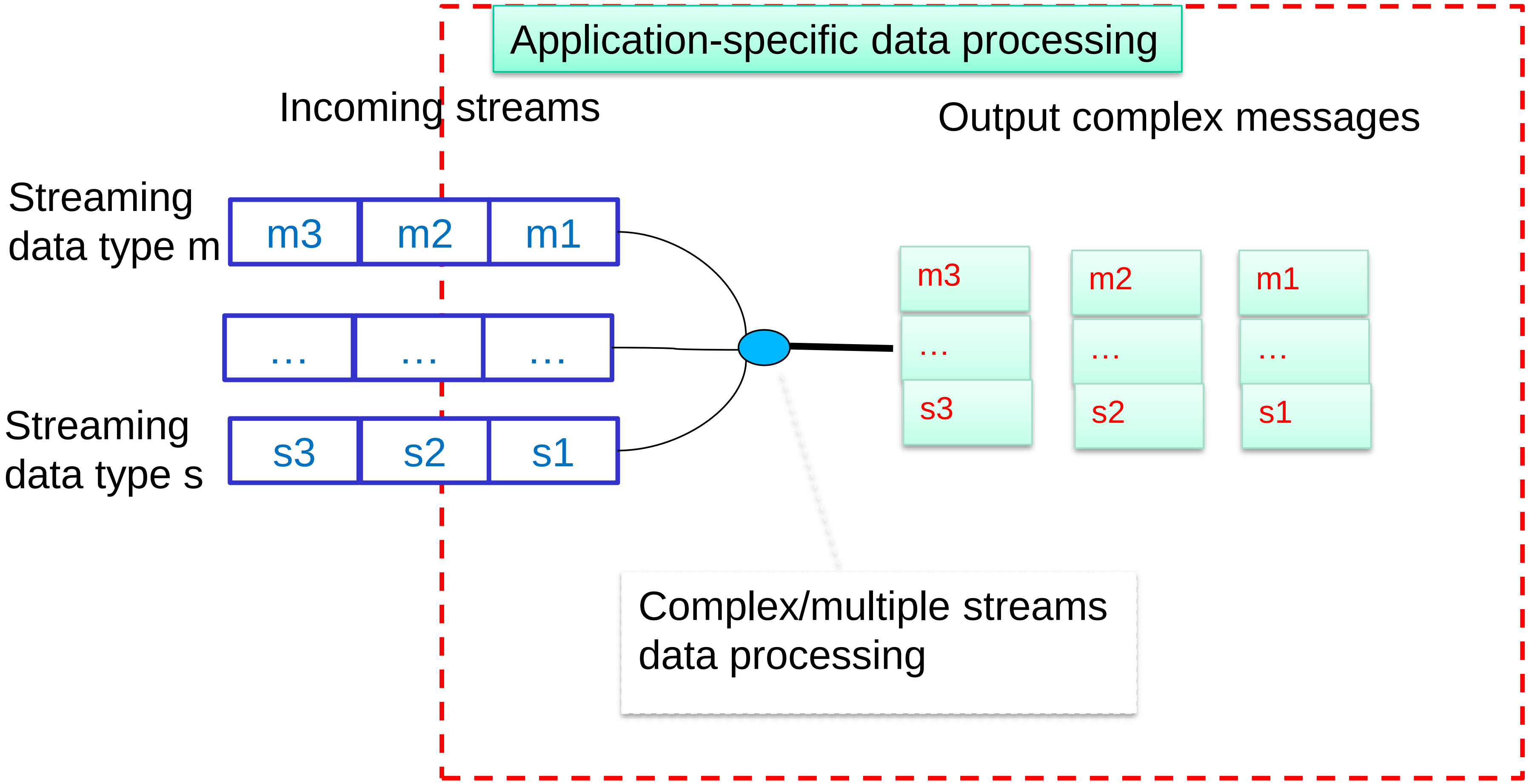
The receiver pulls the data from the broker or the broker pushes the data to the receiver?

MOM – some message processing operations

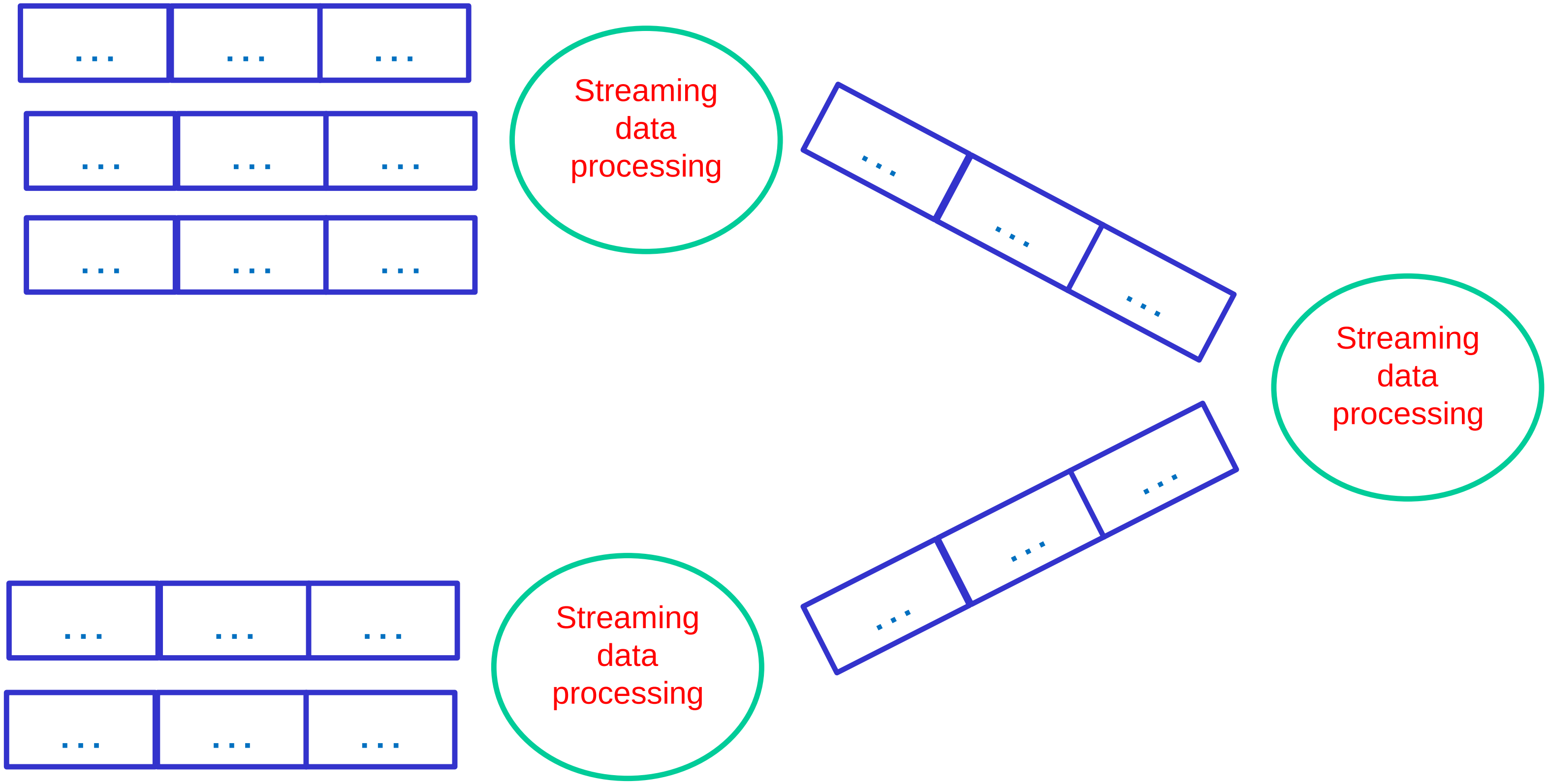


Publish/subscribe/notify; send/forward; routing operations within a broker

Message processing within data consumer



Streaming data processing with a network of data processing elements

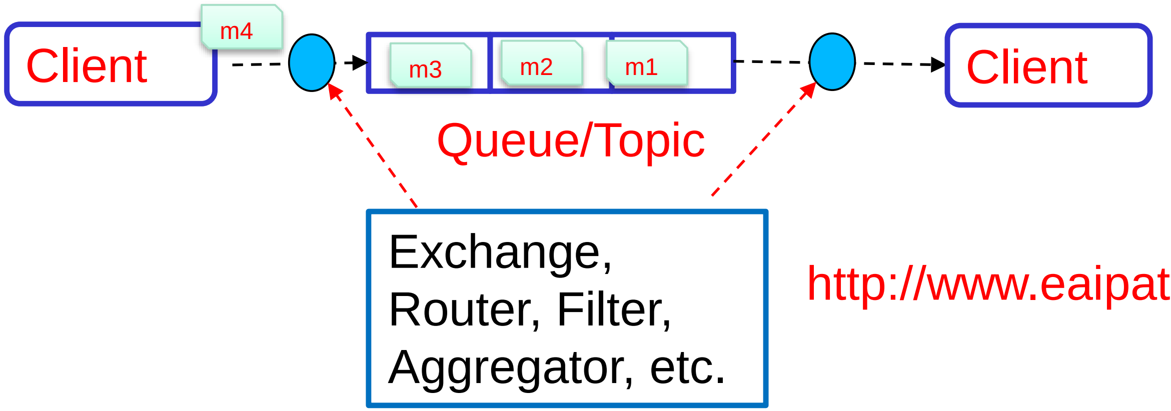

…
Streaming data processing
Streaming data processing
Streaming data processing

Message handling for enterprise integration

- Messages handling concepts and patterns have been around for many years, since we need to support cross services/organizations integration
  - Enterprise integration pattern is well studied but mostly focused on business message
  - http://www.enterpriseintegrationpatterns.com/

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

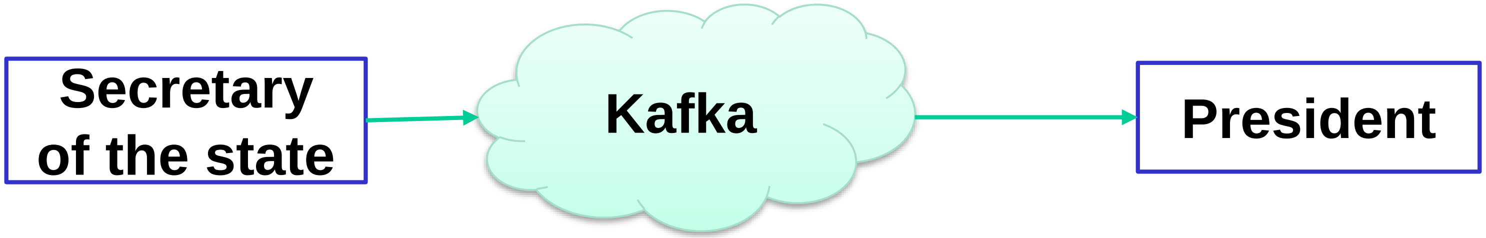
Filter, exchange, etc.

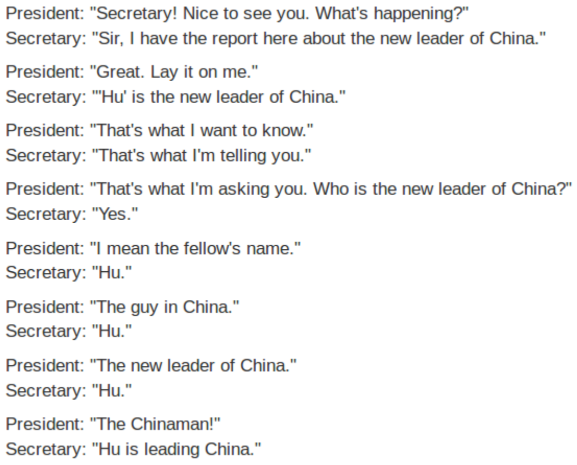


http://www.eaipatterns.com/

We need several features implemented by MOM, consumer, 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."

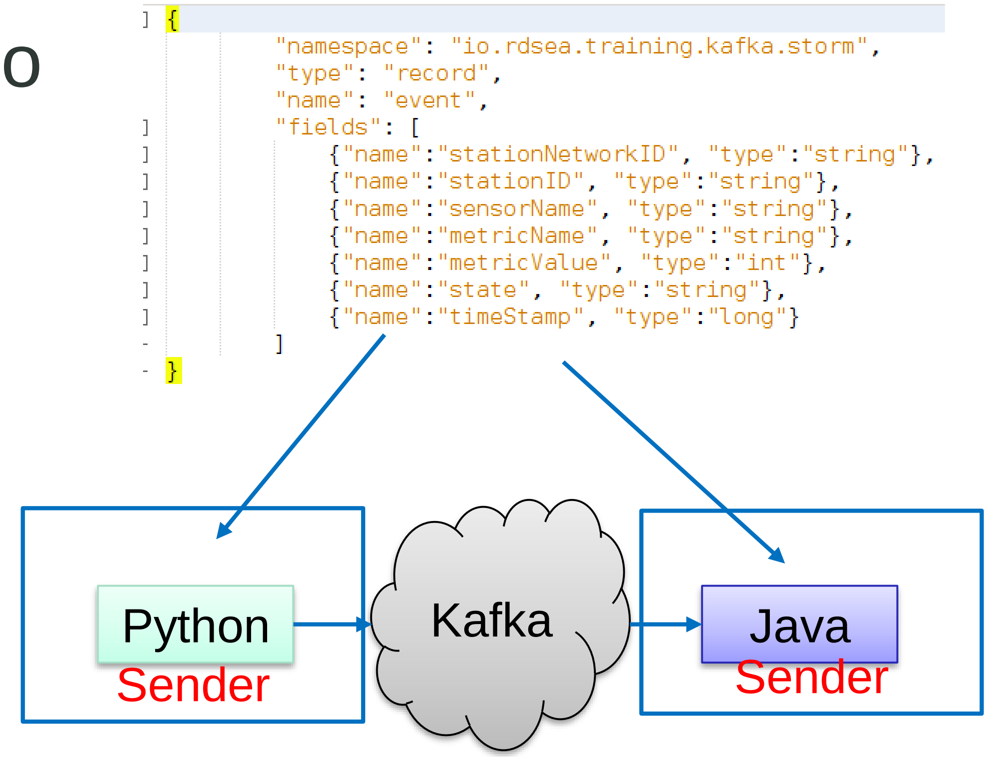
Source: http://www.smart-words.org/humor-jokes/language-humor/who-is-hu-china.html

Message serialization and deserialization

- Remember that the sender and the receiver are diverse
  - In many cases, they are not in the same organization
- Through communication you can send and receive the message
  - But 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-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"}
        ]
}
```

Some other techniques

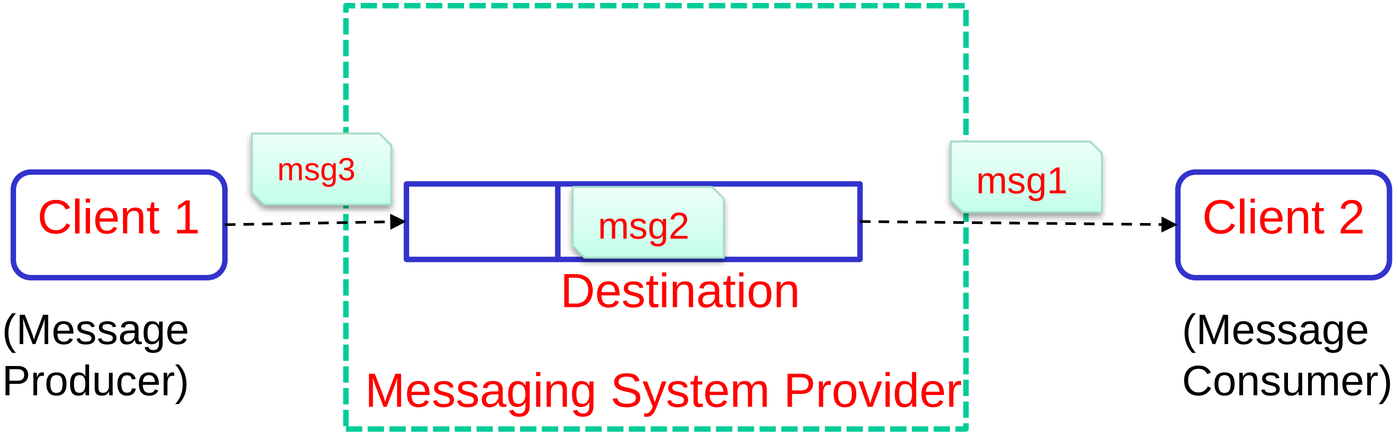
- Protobuf
  - From Google
  - https://github.com/google/protobuf
  - Language-neutral, platform-neutral mechanism for serializing/deserializing structured data
- Thrift
  - https://thrift.apache.org
  - Support also serializing and deserializing data)
  - Support cross-language services development
    - Specify services interfaces
    - Data exchange
    - Code generation

Communication

JAVA MESSAGING SERVICE

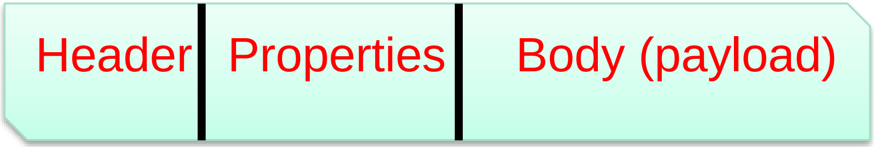
General concepts

- **Standard APIs** for Java platform



Message Structure

- Header: pre-defined system information (e.g., storage, routing and identification operations)
- Properties: application defined properties



Header

Properties

Body (payload)

- 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

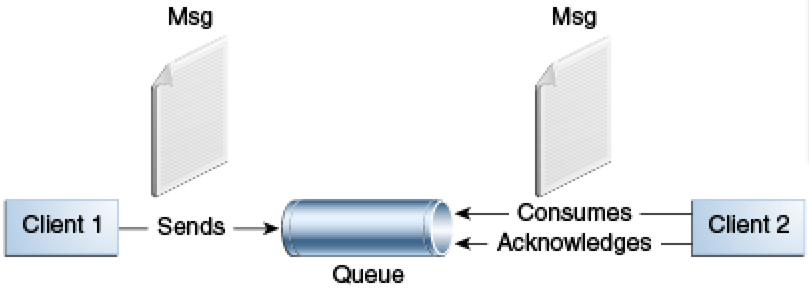


Fig source: http://docs.oracle.com/javaee/7/tutorial/doc/jms-concepts002.htm

Simple question: do we have multiple producers or a single producer per destination (queue/topic)?

Publish/Subscription

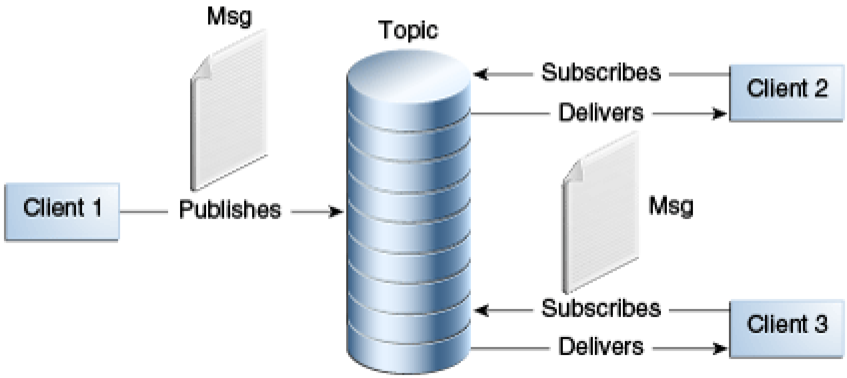


Fig source: http://docs.oracle.com/javaee/7/tutorial/doc/jms-concepts002.htm

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?

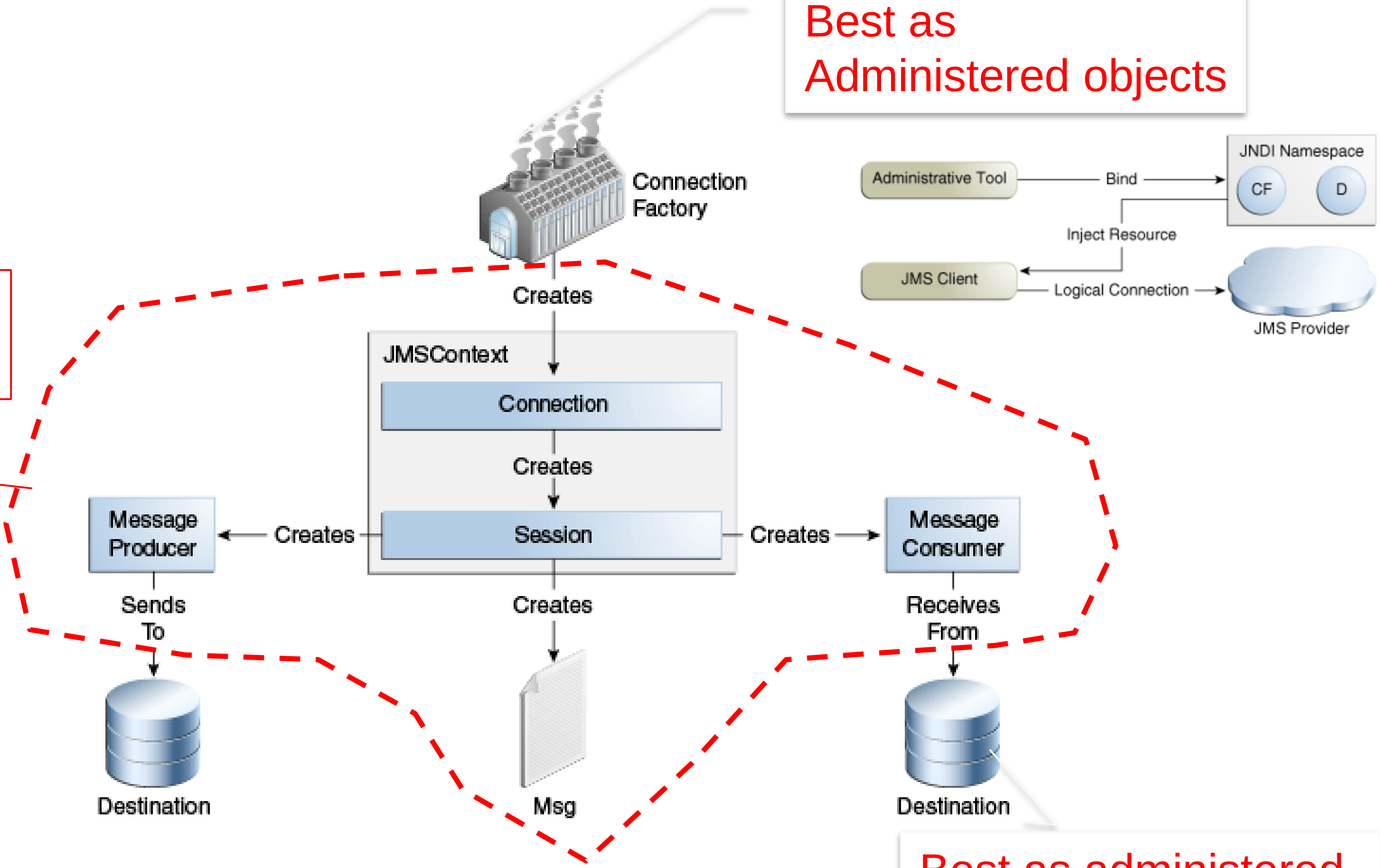
    → Use a unique identifier
  - Need a reply message from a request message

    → 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 with programming activities



Best as Administered objects

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);
```

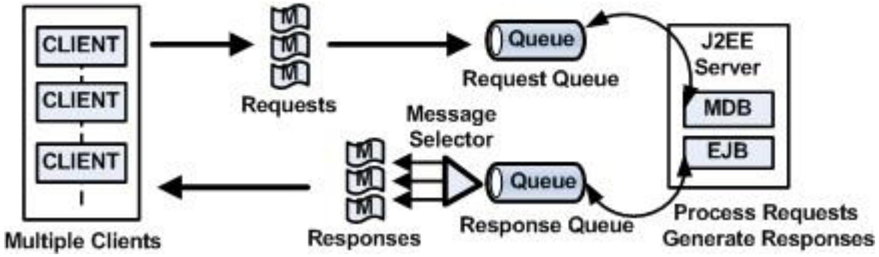
Some other JMS API features

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

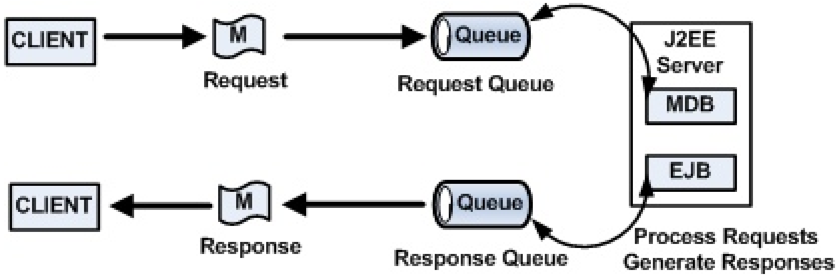
Generic question: how does the broker manage durable subscription?

Example of temporary queues for performance improvement

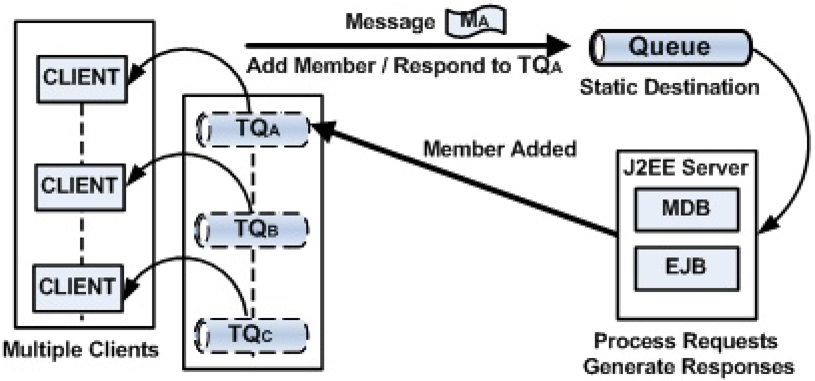
Common static queues for multiple clients



Separate static queues for multiple clients

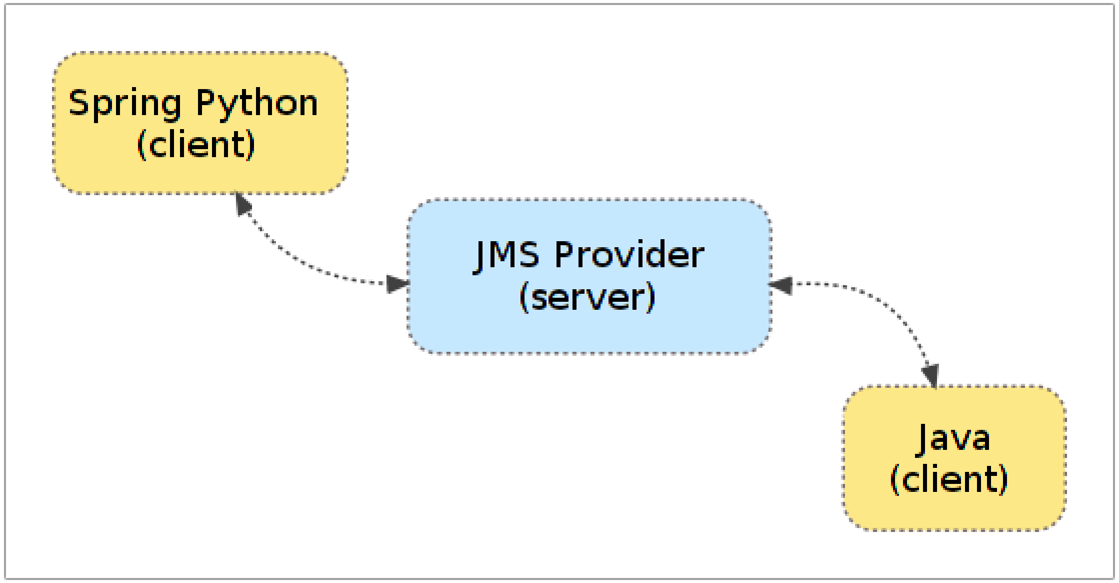


Temporary queues



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

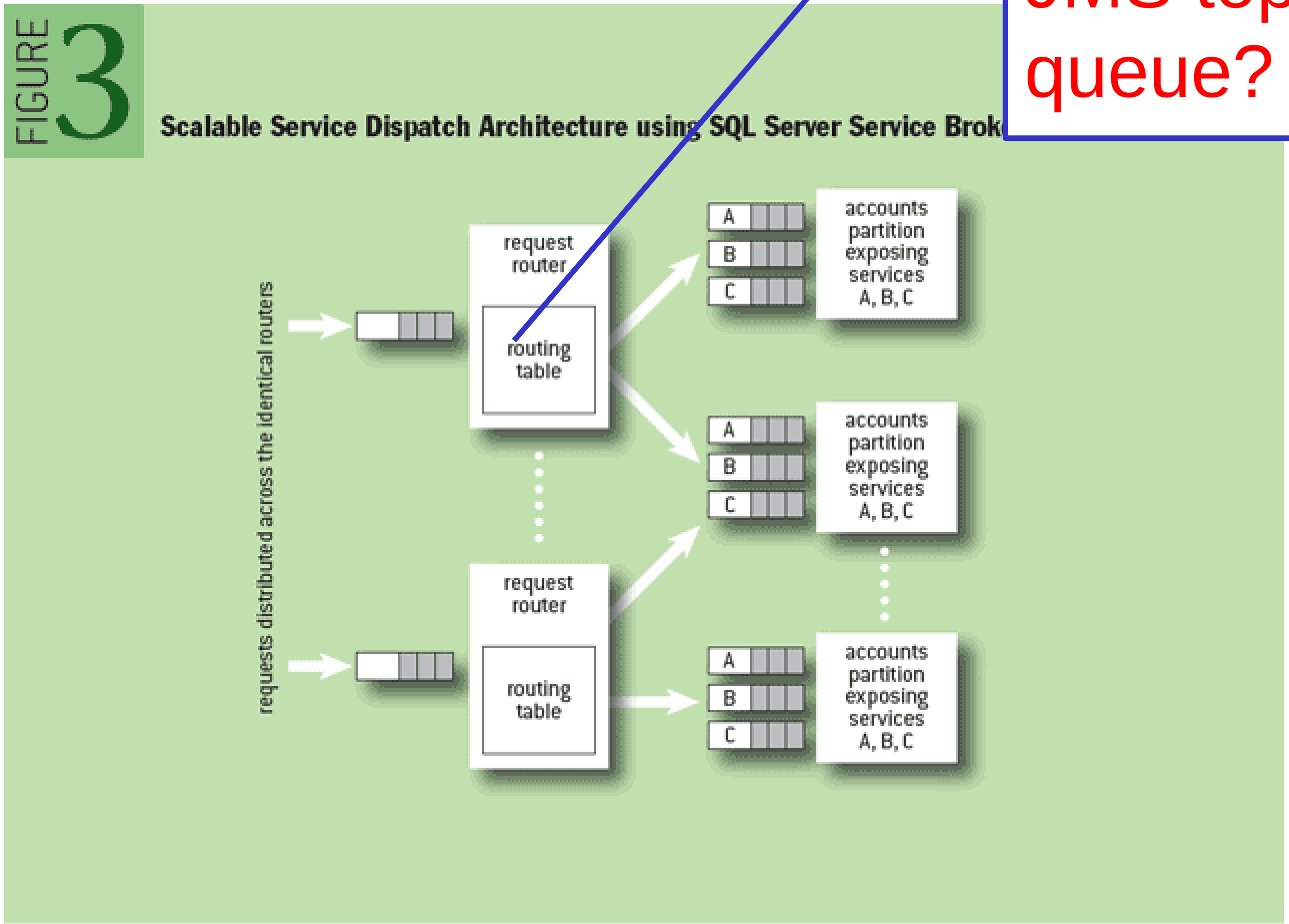


FIGURE 3
Scalable Service Dispatch Architecture using SQL Server Service Brok
requests distributed across the identical routers
request router
routing table
request router
routing table
A
B
C
accounts partition exposing services A, B, C
A
B
C
accounts partition exposing services A, B, C
A
B
C
accounts partition exposing services A, B, C

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

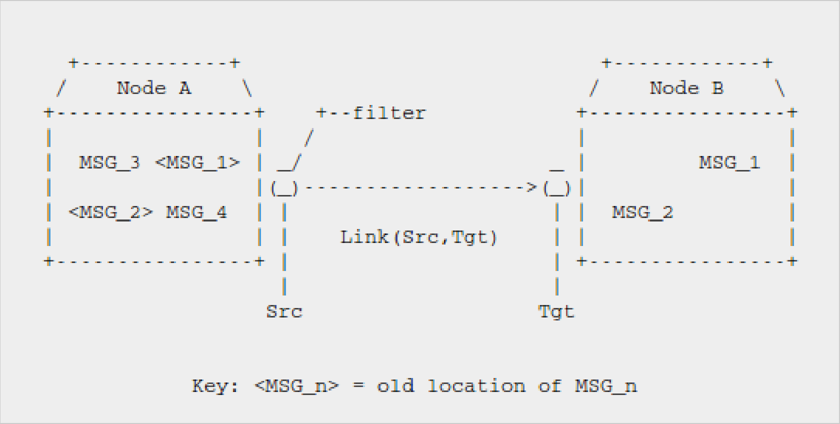
RabbitMQ

Messaging that just works

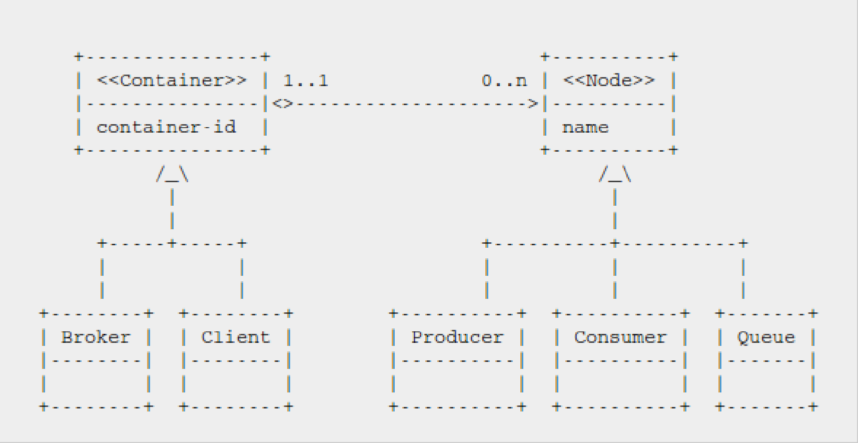
Core concepts – Message/Transport

- 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



```
  +------------+                              +------------+
 /    Node A    \                            /    Node B    \
+----------------+    +--filter             +----------------+
|                |   /                      |                |
|  MSG_3 <MSG_1> | _/                     _ |         MSG_1  |
|                |(_)------------------->(_)|                |
| <MSG_2> MSG_4  | |                      | |  MSG_2         |
|                | |    Link(Src,Tgt)     | |                |
+----------------+ |                      | +----------------+
                   |                      |
                  Src                    Tgt


            Key: <MSG_n> = old location of MSG_n
```

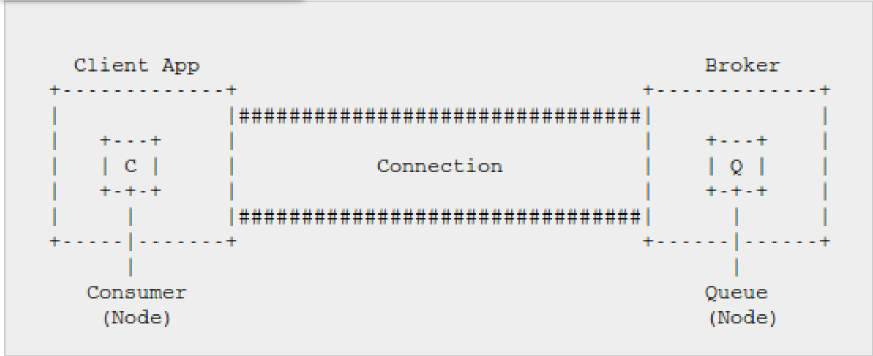


```
   +---------------+                        +----------+
   | <<Container>> | 1..1              0..n | <<Node>> |
   |---------------|<>--------------------->|----------|
   | container-id  |                        | name     |
   +---------------+                        +----------+
          /_\                                    /_\
           |                                      |
           |                                      |
     +-----+-----+                     +----------+----------+
     |           |                     |          |          |
     |           |                     |          |          |
+--------+  +--------+         +----------+  +----------+  +-------+
| Broker |  | Client |         | Producer |  | Consumer |  | Queue |
|--------|  |--------|         |----------|  |----------|  |-------|
|        |  |        |         |          |  |          |  |       |
+--------+  +--------+         +----------+  +----------+  +-------+
```

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

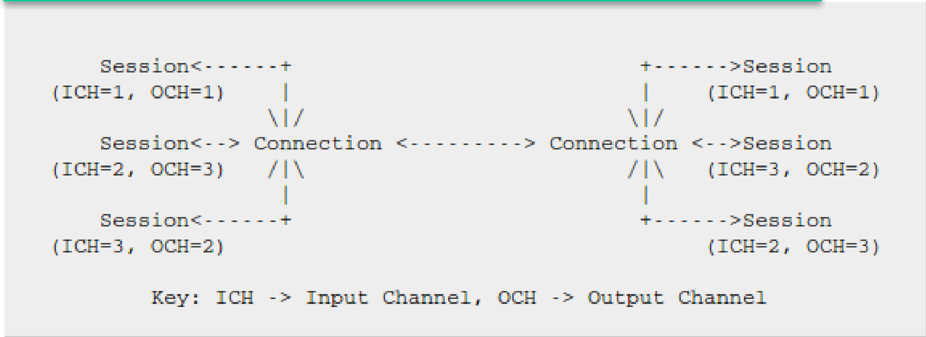
Core concept -- Transport

Connection



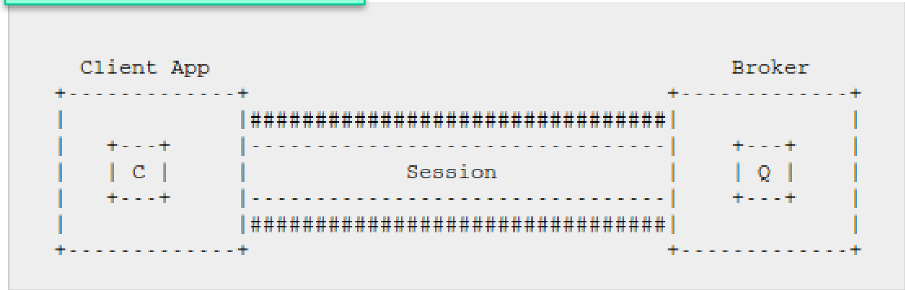
```
  Client App                                         Broker
+--------------+                                +--------------+
|              |################################|              |
|   +---+      |                                |    +---+     |
|   | C |      |          Connection            |    | Q |     |
|   +-+-+      |                                |    +-+-+     |
|     |        |################################|      |       |
+-----|--------+                                +------|-------+
      |                                                |
   Consumer                                          Queue
    (Node)                                           (Node)
```

Session and Connection endpoints



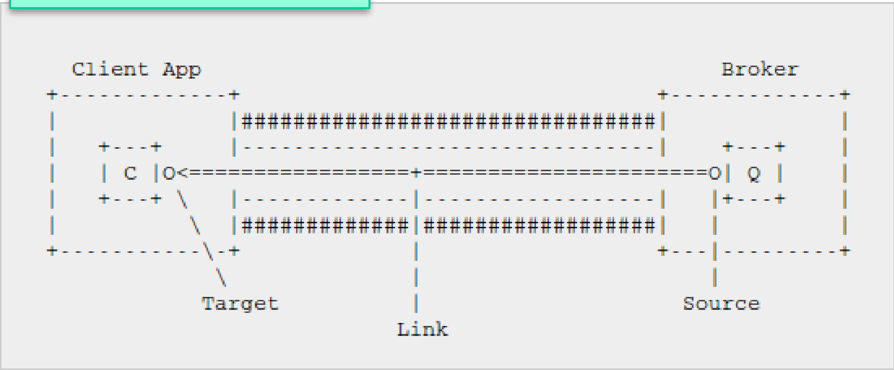
```
    Session<------+                            +------>Session
(ICH=1, OCH=1)    |                            |    (ICH=1, OCH=1)
                 \|/                          \|/
    Session<--> Connection <---------> Connection <-->Session
(ICH=2, OCH=3)   /|\                          /|\   (ICH=3, OCH=2)
                  |                            |
    Session<------+                            +------>Session
(ICH=3, OCH=2)                                      (ICH=2, OCH=3)

        Key: ICH -> Input Channel, OCH -> Output Channel
```

Session



```
  Client App                                         Broker
+--------------+                                +--------------+
|              |################################|              |
|   +---+      |--------------------------------|    +---+     |
|   | C |      |            Session             |    | Q |     |
|   +---+      |--------------------------------|    +---+     |
|              |################################|              |
+--------------+                                +--------------+
```

Links

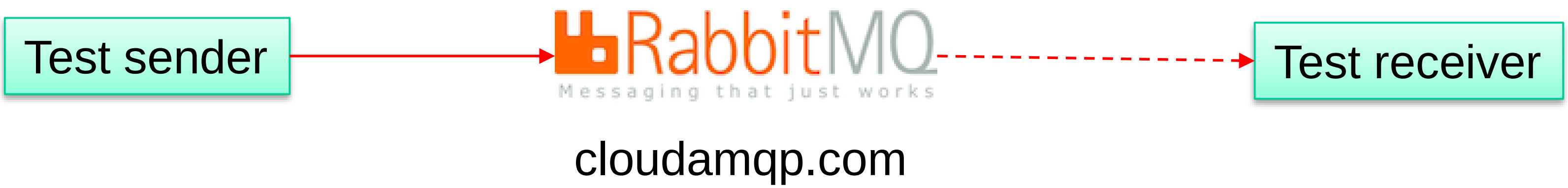


```
  Client App                                           Broker
+--------------+                                  +--------------+
|              |##################################|              |
|   +---+      |----------------------------------|    +---+     |
|   | C |O<===================+=======================O| Q |     |
|   +---+  \   |--------------|-------------------|   |+---+     |
|           \  |##############|###################|   |          |
+------------\-+              |                   +---|----------+
              \               |                       |
            Target            |                    Source
                            Link
```

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

Example: AMQP

```
ConnectionFactory factory = new ConnectionFactory();
   factory.setUri(uri);
   Connection connection = factory.newConnection();
   Channel channel = connection.createChannel();


   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);
   }


   channel.close();
   connection.close();
```

Source code: https://github.com/cloudamqp/java-amqp-example

```
ConnectionFactory factory = new ConnectionFactory();
  factory.setUri(uri);
  Connection connection = factory.newConnection();
  Channel channel = connection.createChannel();


  channel.queueDeclare(QUEUE_NAME, false, false,
     false, null);
  System.out.println(" [*] Waiting for messages");


  QueueingConsumer consumer = new
     QueueingConsumer(channel);
  channel.basicConsume(QUEUE_NAME, true,
     consumer);


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

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)

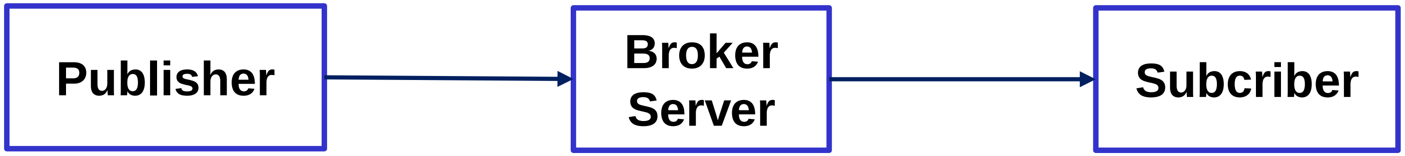
MQTT Overview

- 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)

Model and Implementation

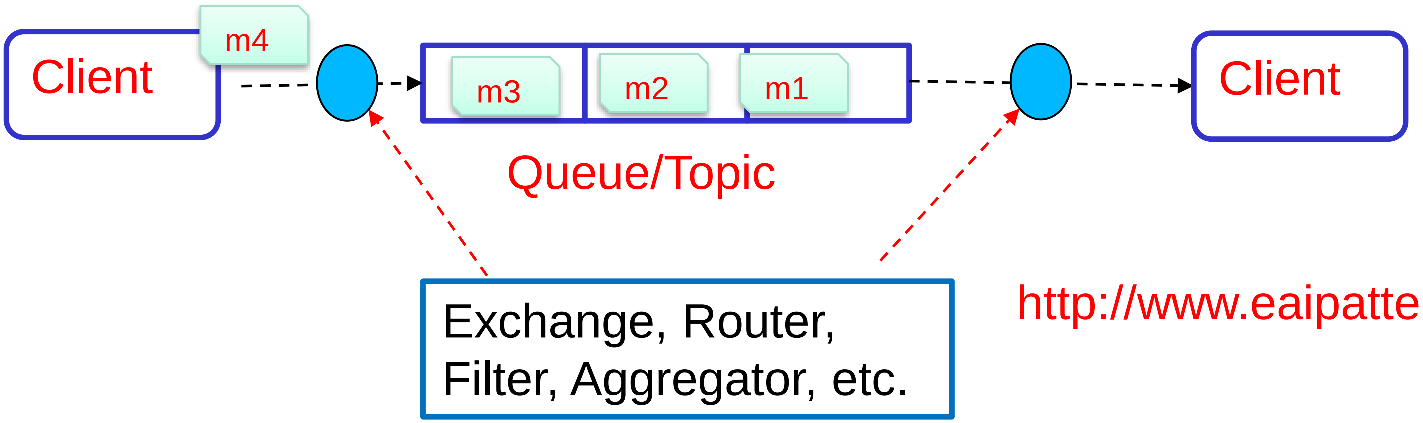


- Different programming languages for OS/devices
  - Including Anrduino, Nanode
- Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto)
- Paho: http://www.eclipse.org/paho/
- RabbitMQ
- Apache ActiveMQ
- Cloud providers:
  - http://cloudmqtt.com (get a free account to learn MQTT)

Integration

MESSAGE ROUTING PATTERNS

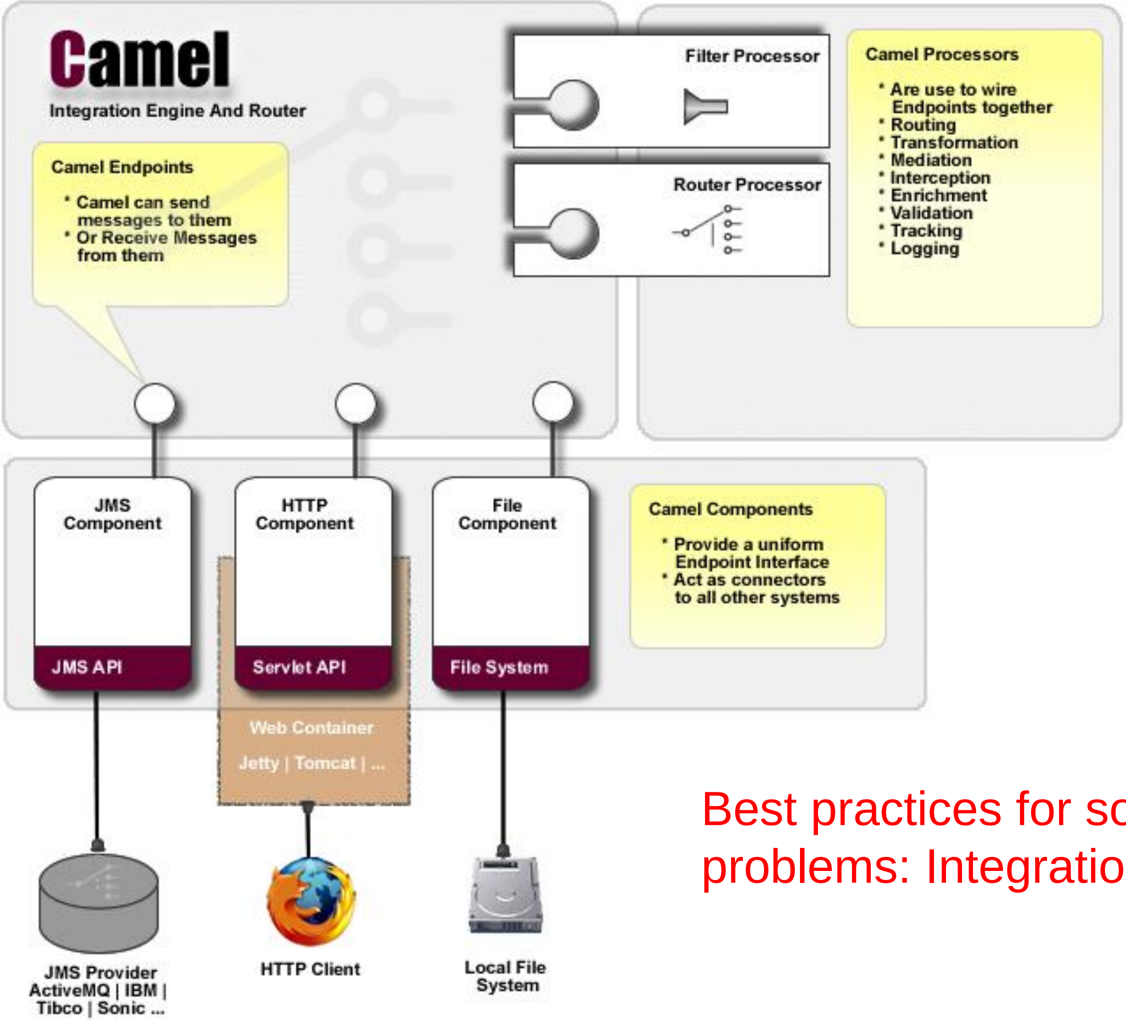
Integration Issues



http://www.eaipatterns.com/

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

Example of supporting technology



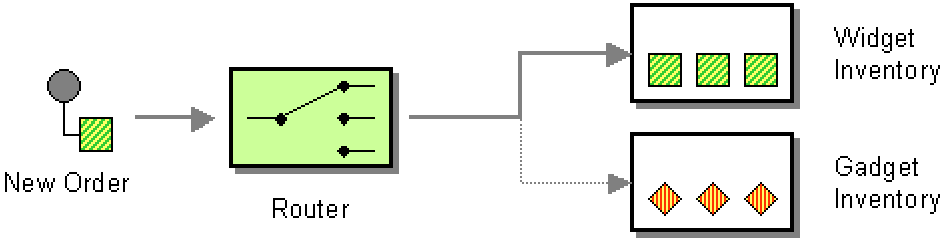
Best practices for solving common problems: Integration Patterns

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

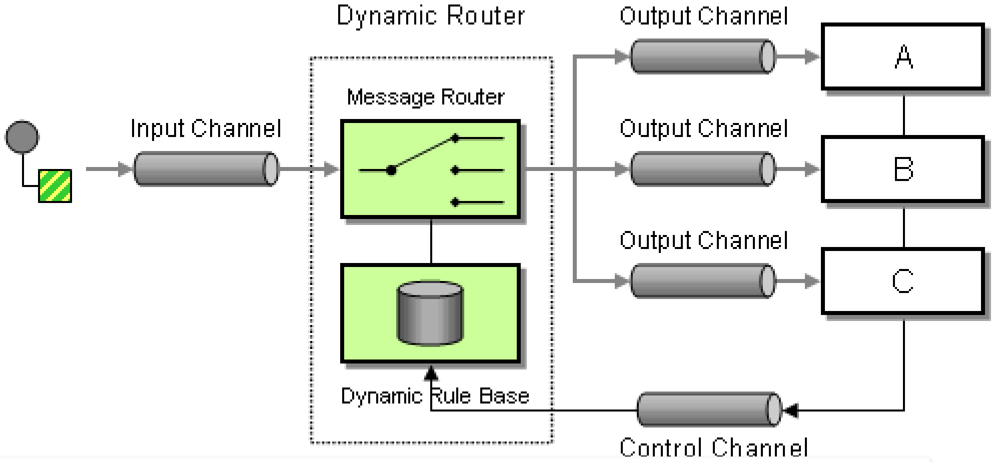
Content-Based Message Routing: Camel/EIP

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

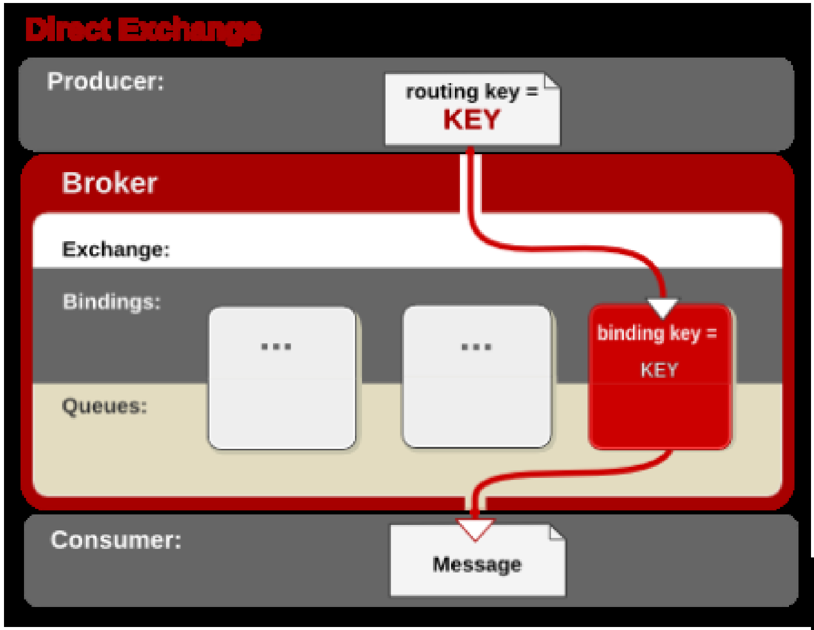


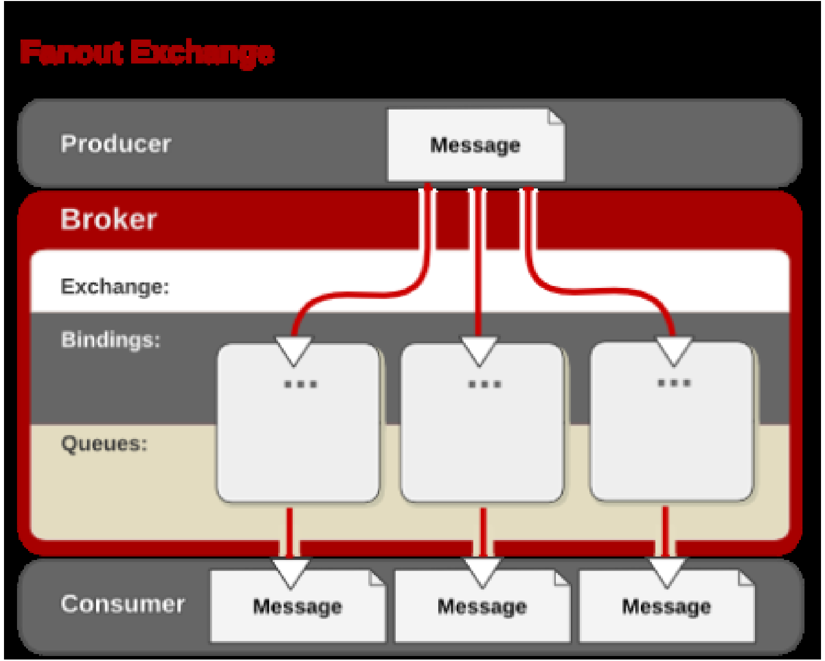
Source: https://camel.apache.org/content-based-router.html

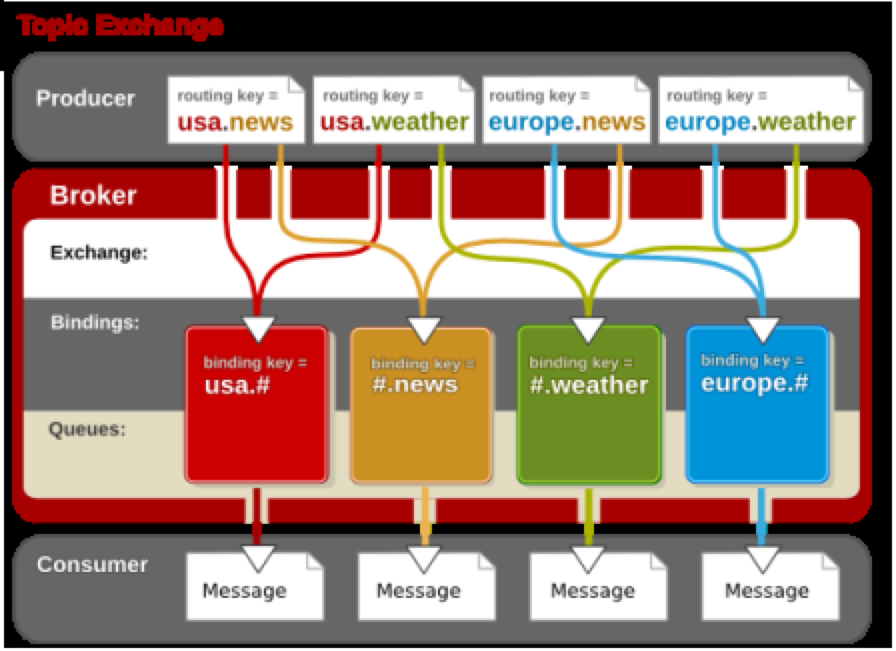


Source: https://camel.apache.org/dynamic-router.html

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

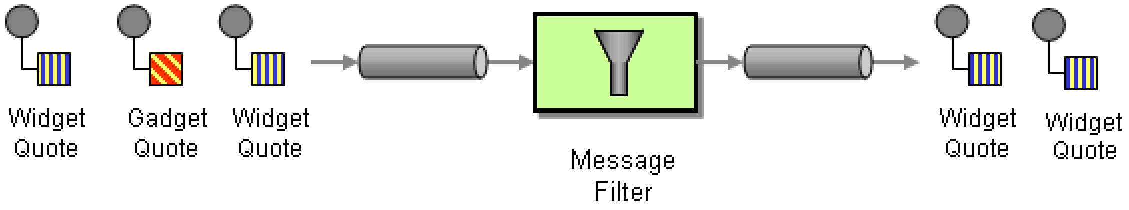
Message Filter/Selector

JMS: selector based on message header and properties

```
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’”);
```

Message Selector or Message Filter: filter unneeded messages

CAMEL/EIP: Message Filter



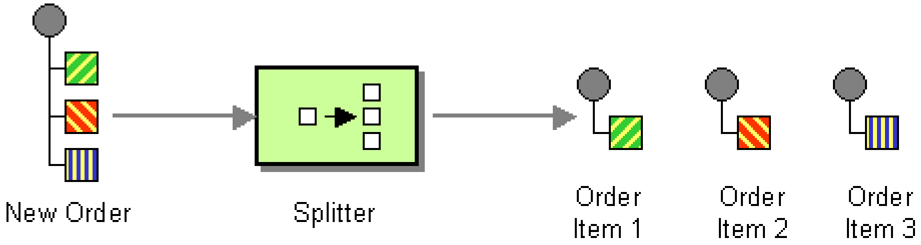
https://camel.apache.org/message-filter.html

Integration

TRANSFORMATION PATTERNS AND TOOLS

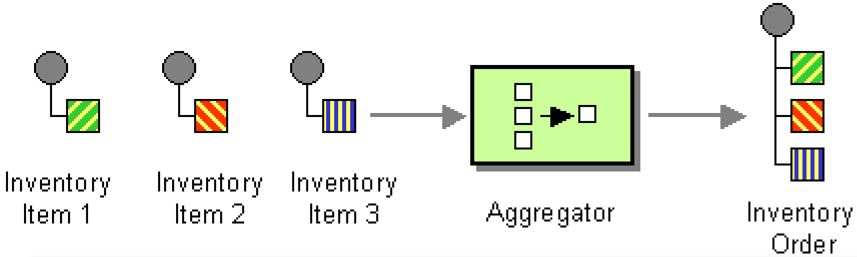
Splitter and Aggregator

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

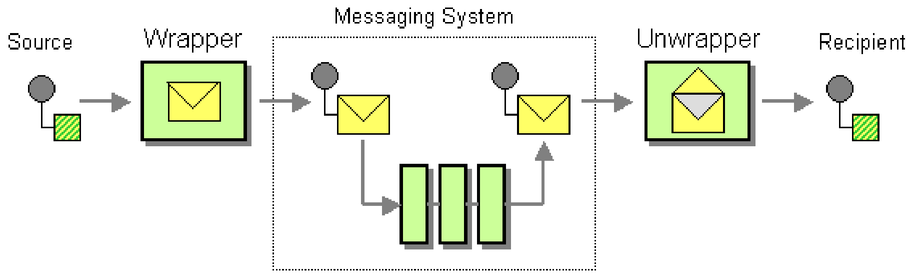


https://camel.apache.org/aggregator2.html

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

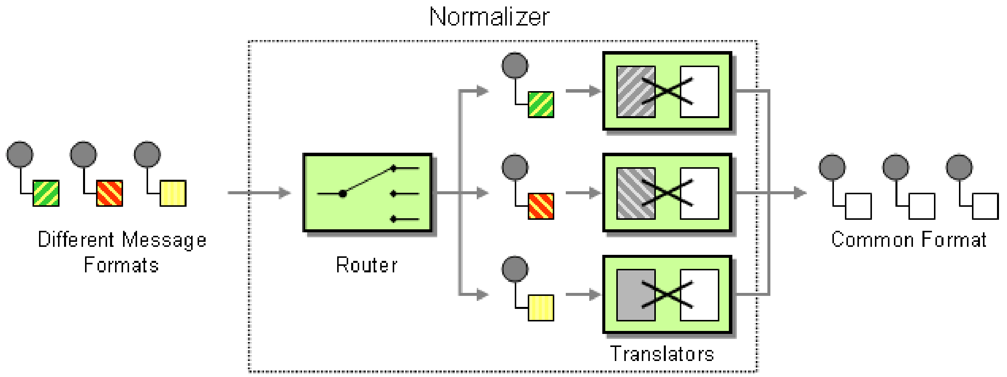
Envelope Wrapper and Normalizer

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

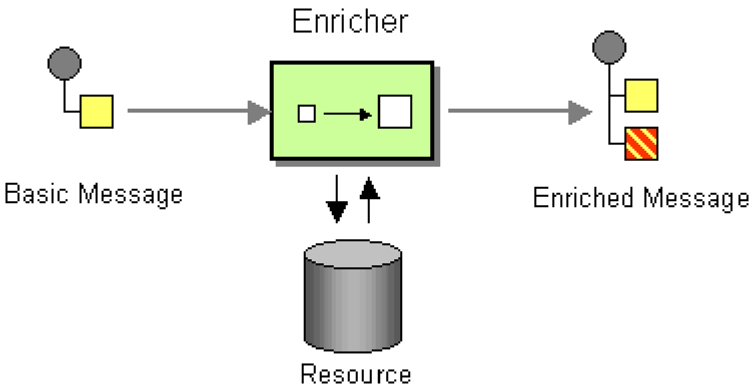
Normalizer: route all messages of a given type to a suitable Message Translator which transforms the message to the common format.



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

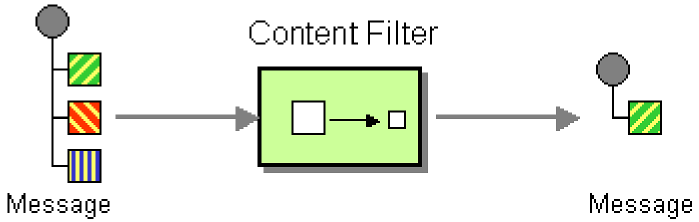
Content Enricher & Extracter

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



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

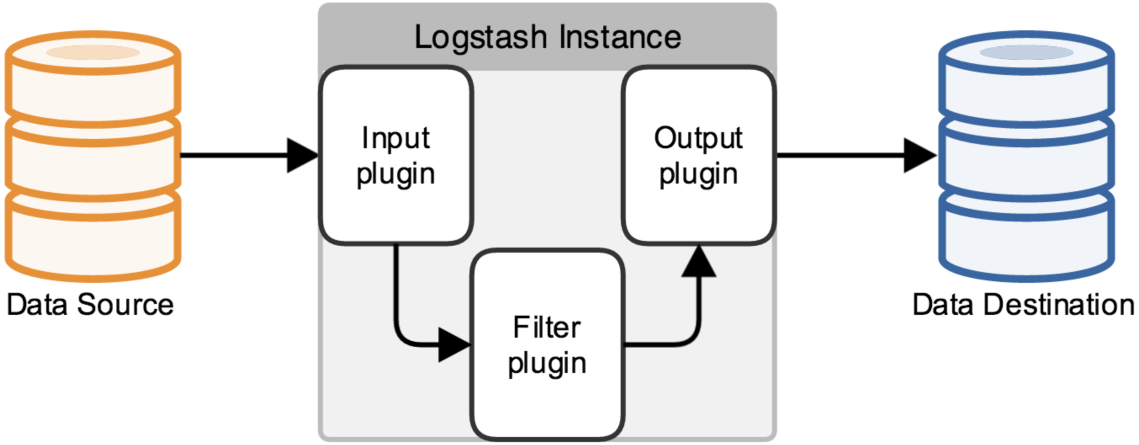
Content Filter: remove unimportant data items from a message or extract only needed information.



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?

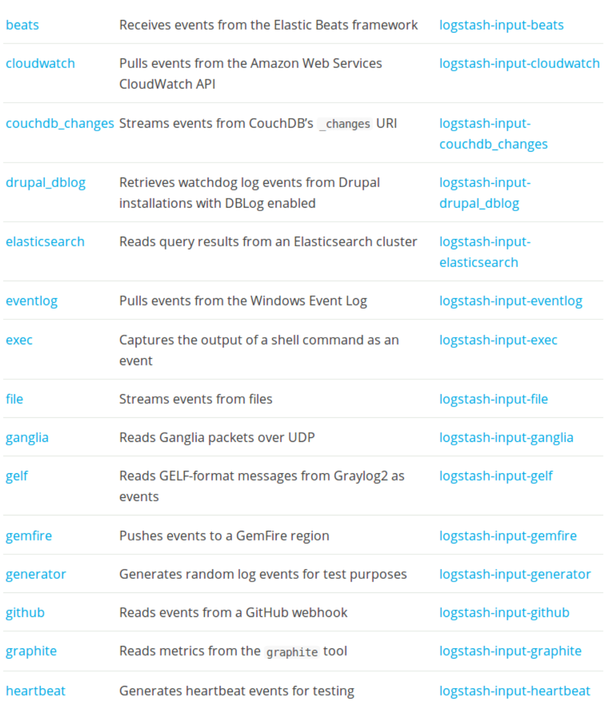
Logstash



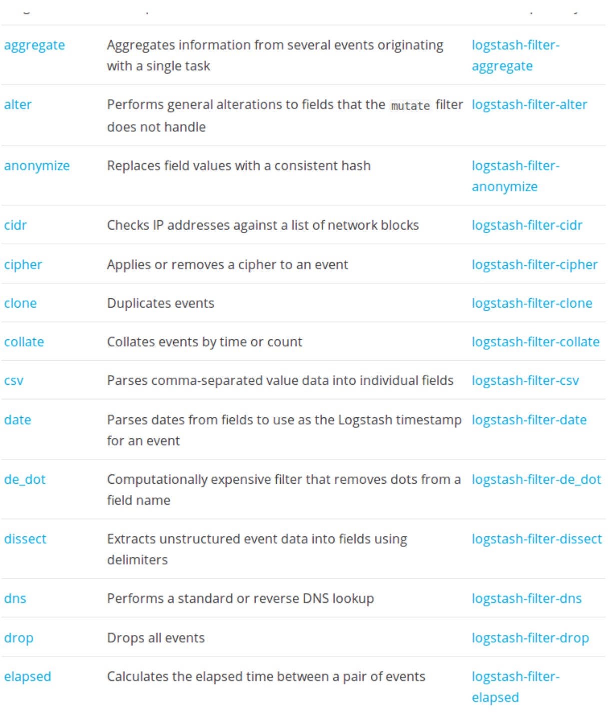
- Codecs: stream filters within inputs or outputs that change data representation
- E.g.: multilines → a single event

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

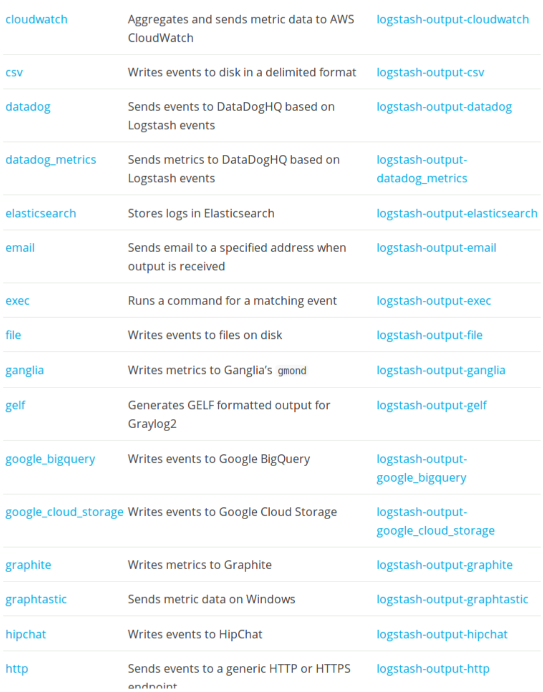
Plug-ins



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

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:

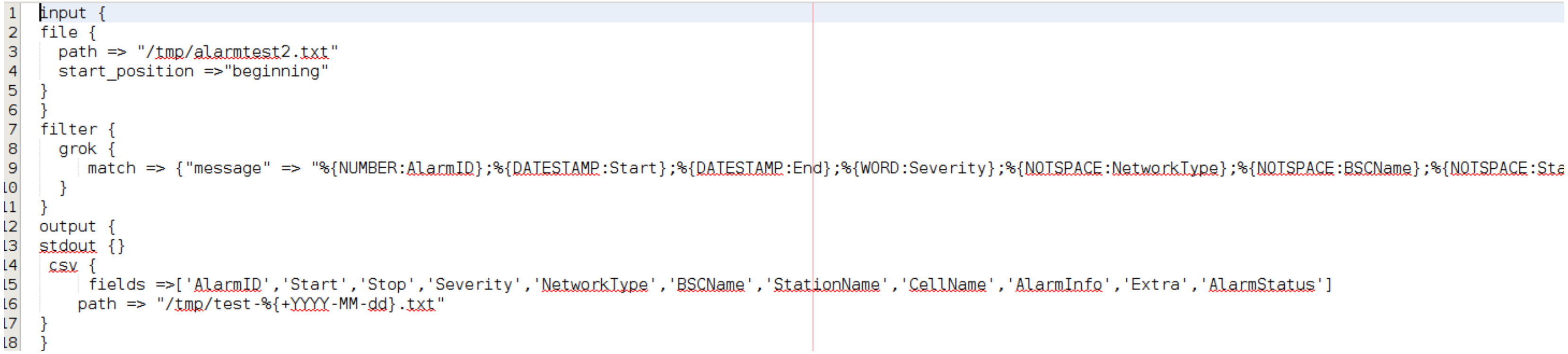
- https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

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

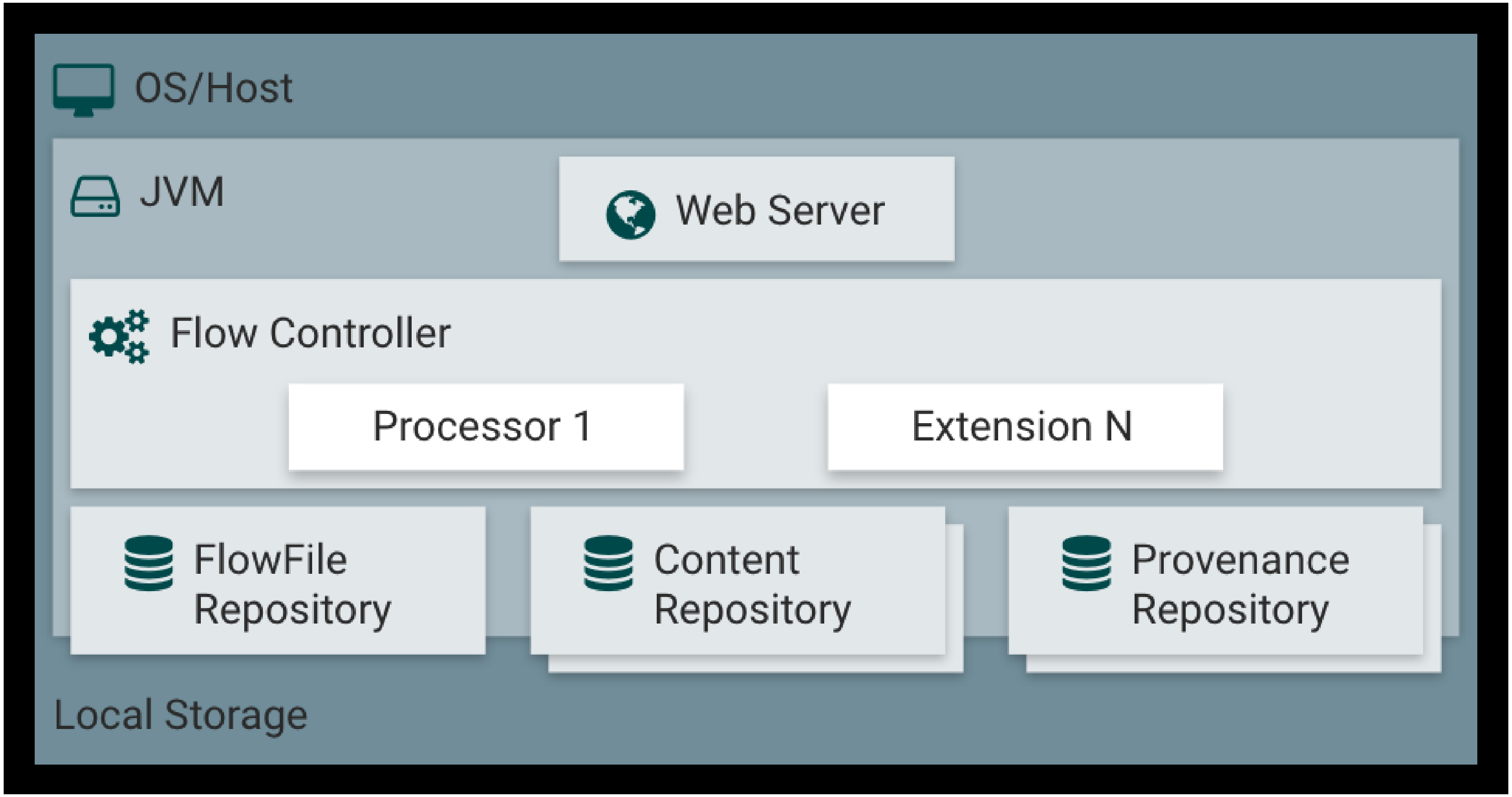


```
input {
file {
  path => "/tmp/alarmtest2.txt"
  start_position =>"beginning"
}
}
filter {
  grok {
     match => {"message" => "%{NUMBER:AlarmID};%{DATESTAMP:Start};%{DATESTAMP:End};%{WORD:Severity};%{NOTSPACE:NetworkType};%{NOTSPACE:BSCName};%{NOTSPACE:Sta
  }
}
output {
stdout {}
 csv {
     fields =>['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

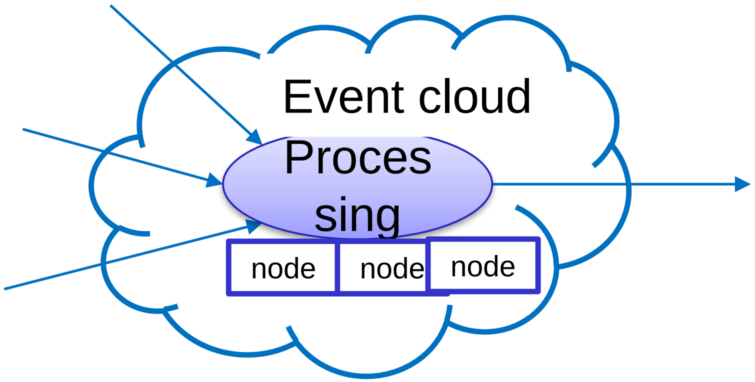
Processing

COMPLEX EVENT PROCESSING

Centralized versus distributed processing topology

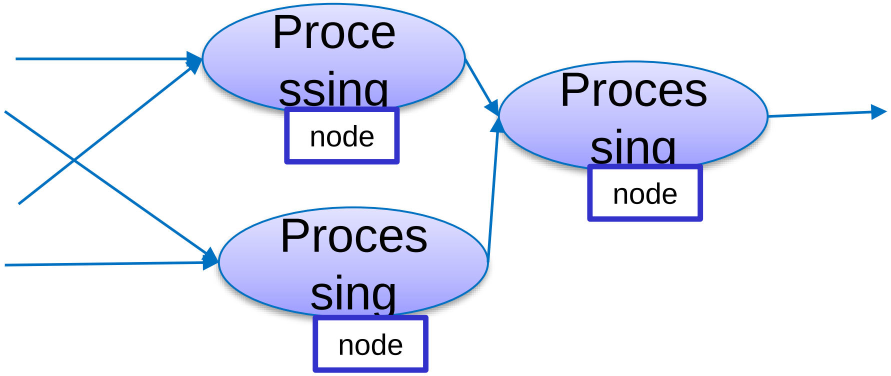
Two views: streams of events or cloud of events

Complex Event Processing
(centralized processing)



Usually only
queries/patterns are written

Streaming Data Processing
(distributed processing)

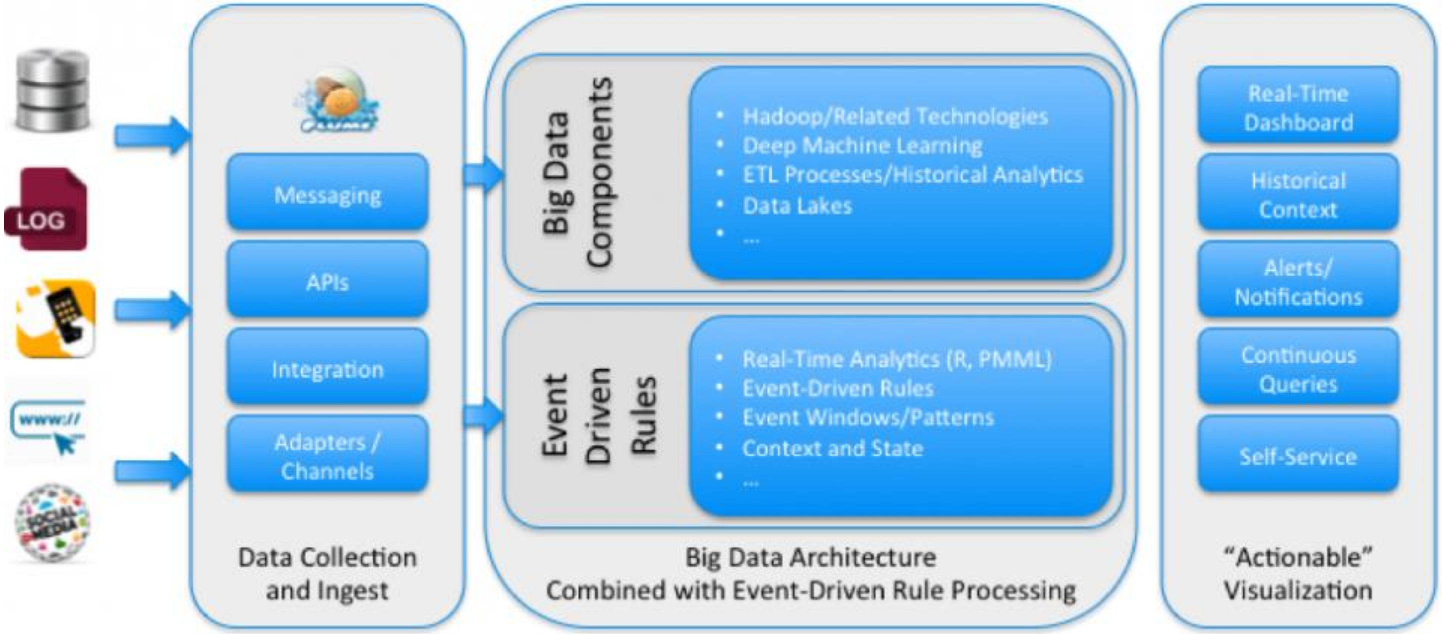


Code processing events and
topologies need to be
written

Goals of complex event processing

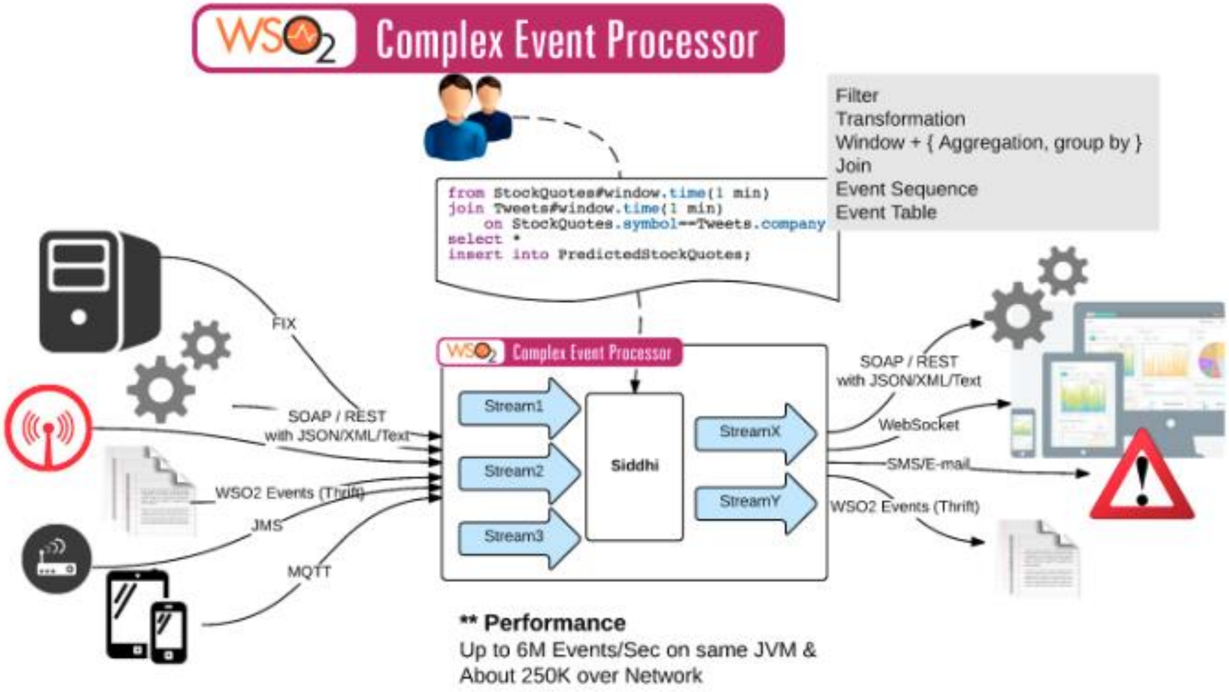
- Group and process events in a specific time (how long?) and space (size) constraints
  - Detect special events
  - Finding correlation and causality
  - Aggregation events
  - Queries for period time

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

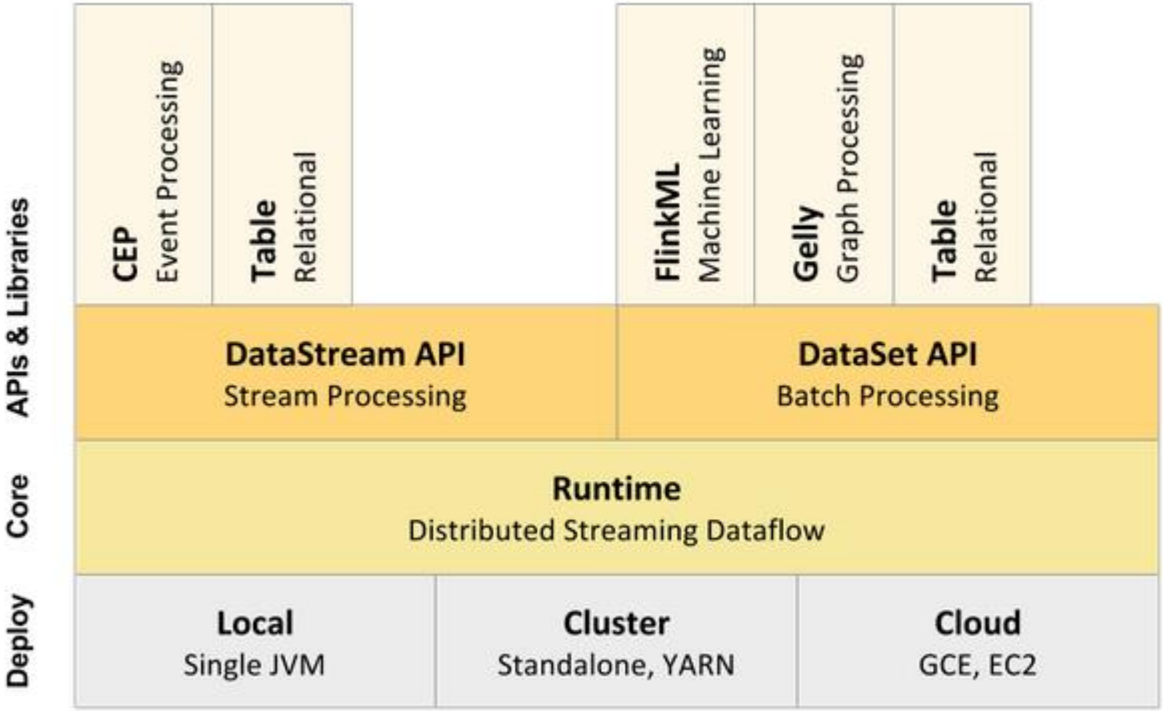


```
from StockQuotes#window.time(1 min)
join Tweets#window.time(1 min)
    on StockQuotes.symbol==Tweets.company
select *
insert into PredictedStockQuotes;
```

**** Performance**
Up to 6M Events/Sec on same JVM &
About 250K over Network

Source: https://docs.wso2.com/display/CEP420/WSO2+Complex+Event+Processor+Documentation

Apache Flink



APIs & Libraries

CEP
Event Processing

Table
Relational

FlinkML
Machine Learning

Gelly
Graph Processing

Table
Relational

DataStream API
Stream Processing

DataSet API
Batch Processing

Core

Runtime
Distributed Streaming Dataflow

Deploy

Local
Single JVM

Cluster
Standalone, YARN

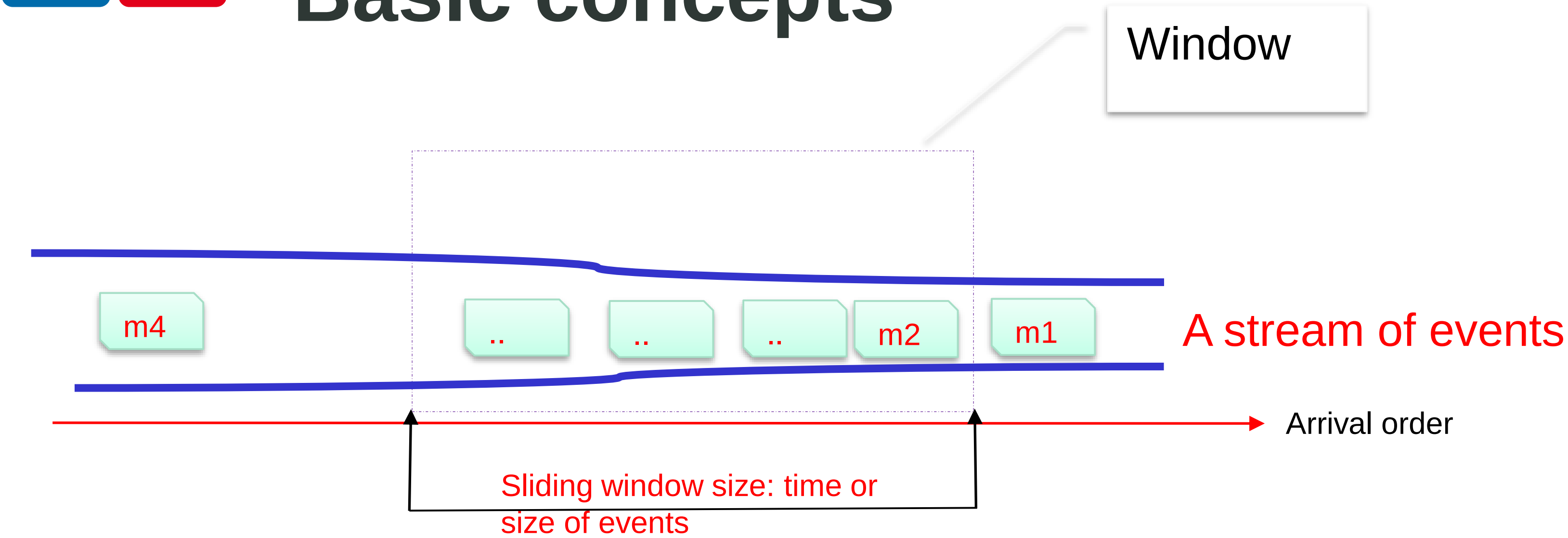
Cloud
GCE, EC2

Source: https://flink.apache.org/introduction.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

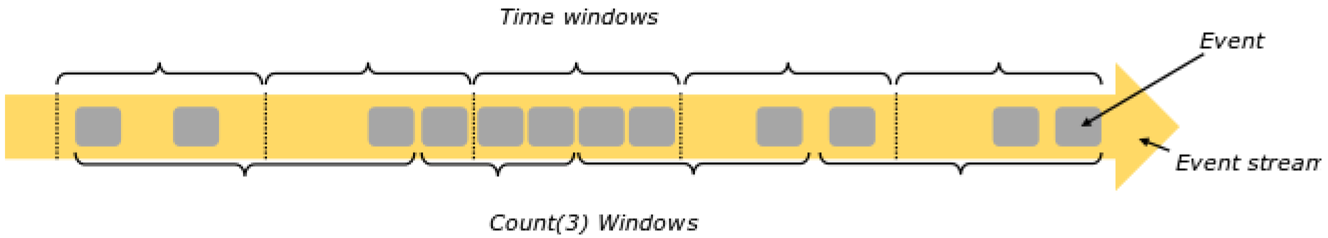
- get a set of events filtered from the window that match these conditions

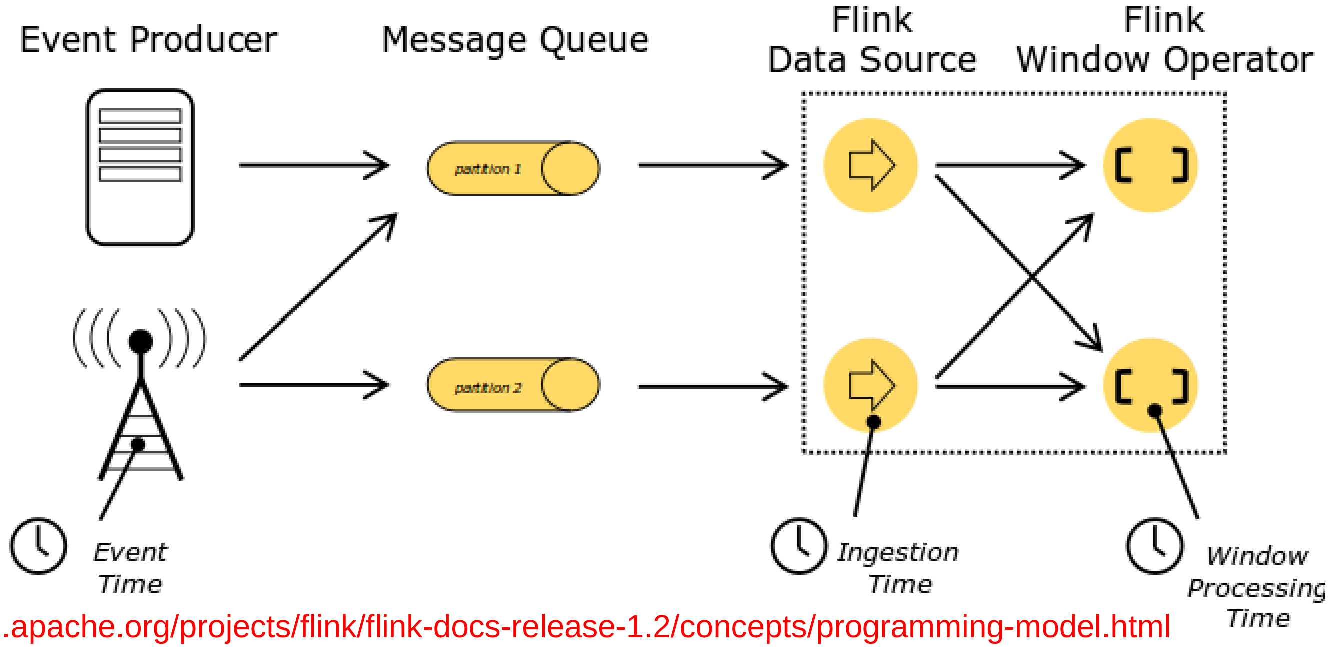
Conditions: can be specified using an SQL-alike language or pre-defined functions

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

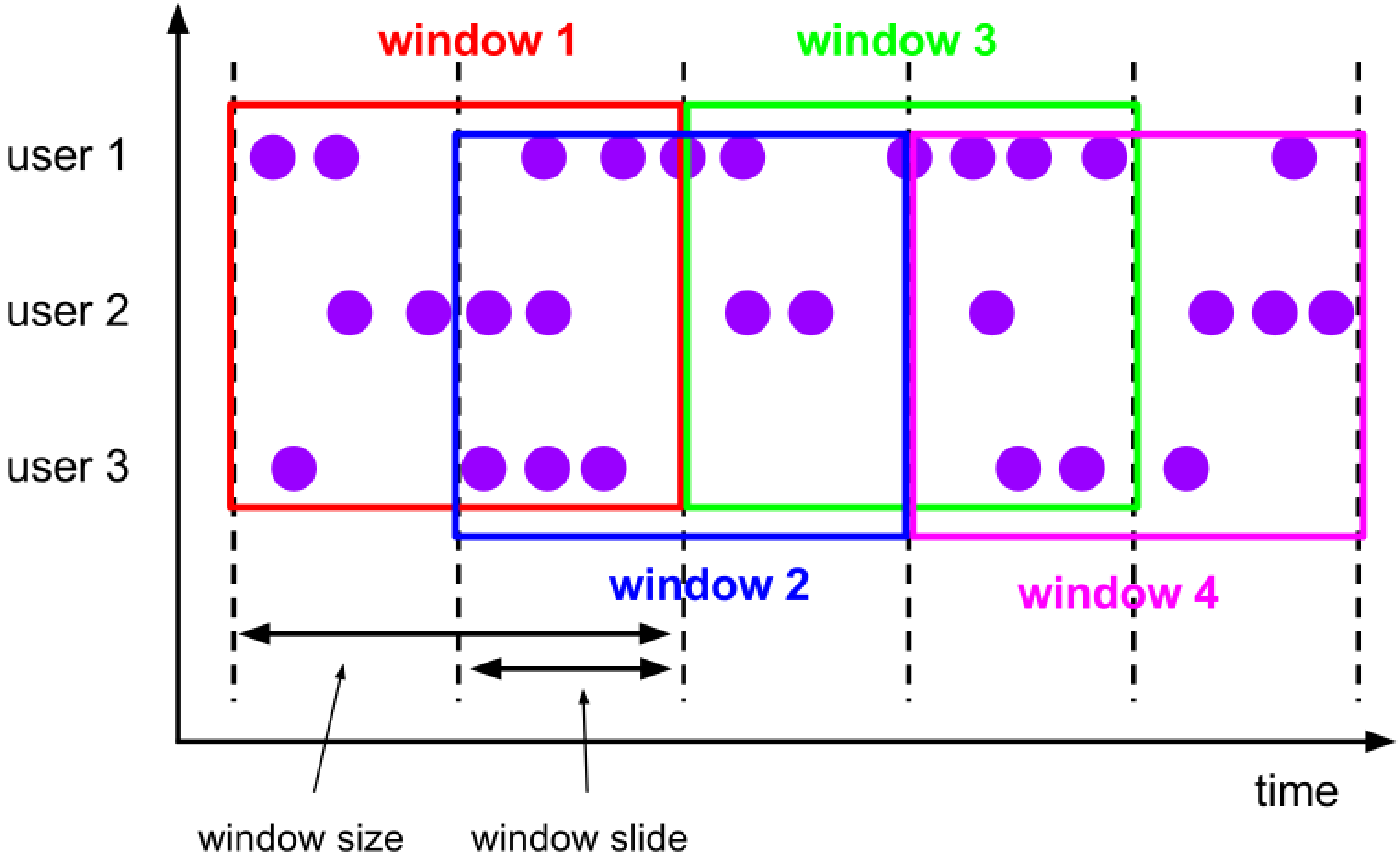
Windows and Times





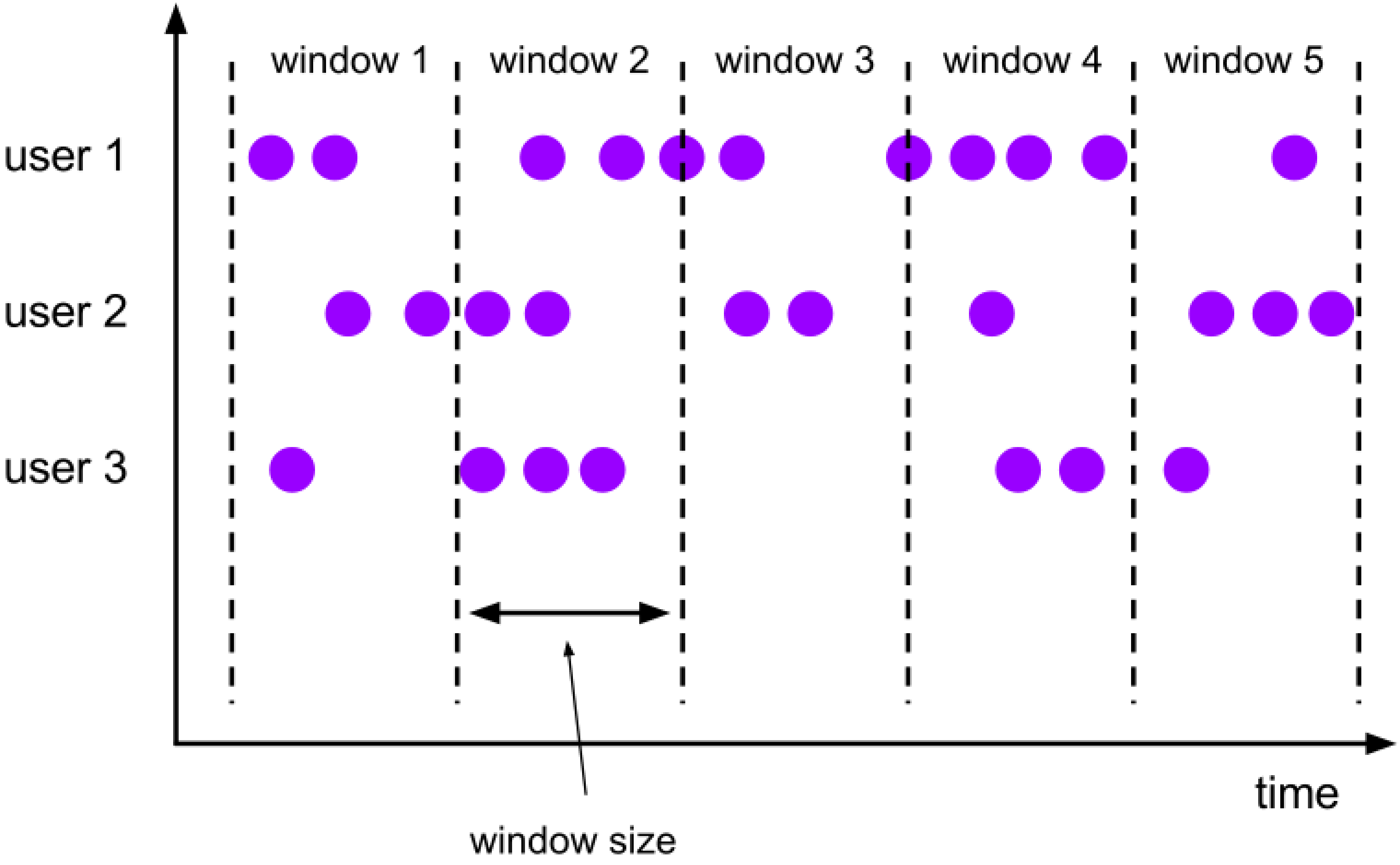
Source: https://ci.apache.org/projects/flink/flink-docs-release-1.2/concepts/programming-model.html

Window size and slide



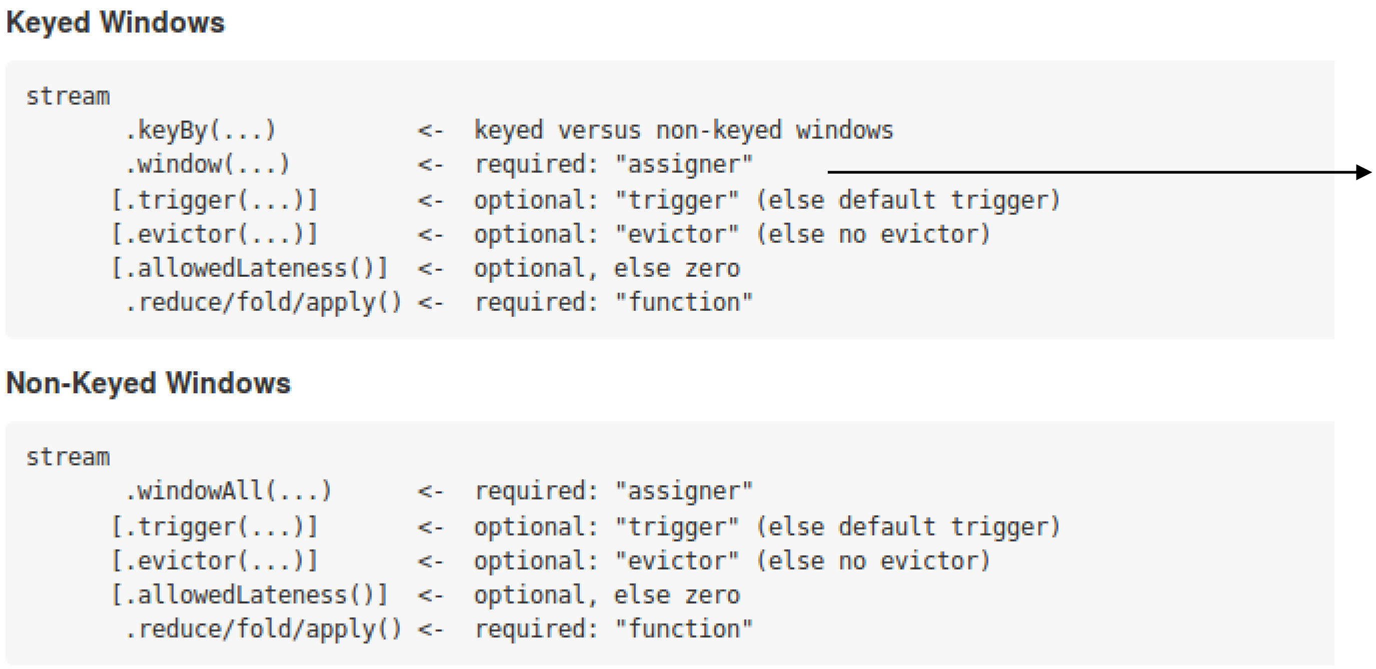
Source: https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/windows.html

Batch/Tumbling Windows



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

Flink Window processing



Keyed Windows

```
stream
       .keyBy(...)          <-  keyed versus non-keyed windows
       .window(...)         <-  required: "assigner"
      [.trigger(...)]       <-  optional: "trigger" (else default trigger)
      [.evictor(...)]       <-  optional: "evictor" (else no evictor)
      [.allowedLateness()]  <-  optional, else zero
       .reduce/fold/apply() <-  required: "function"
```

Non-Keyed Windows

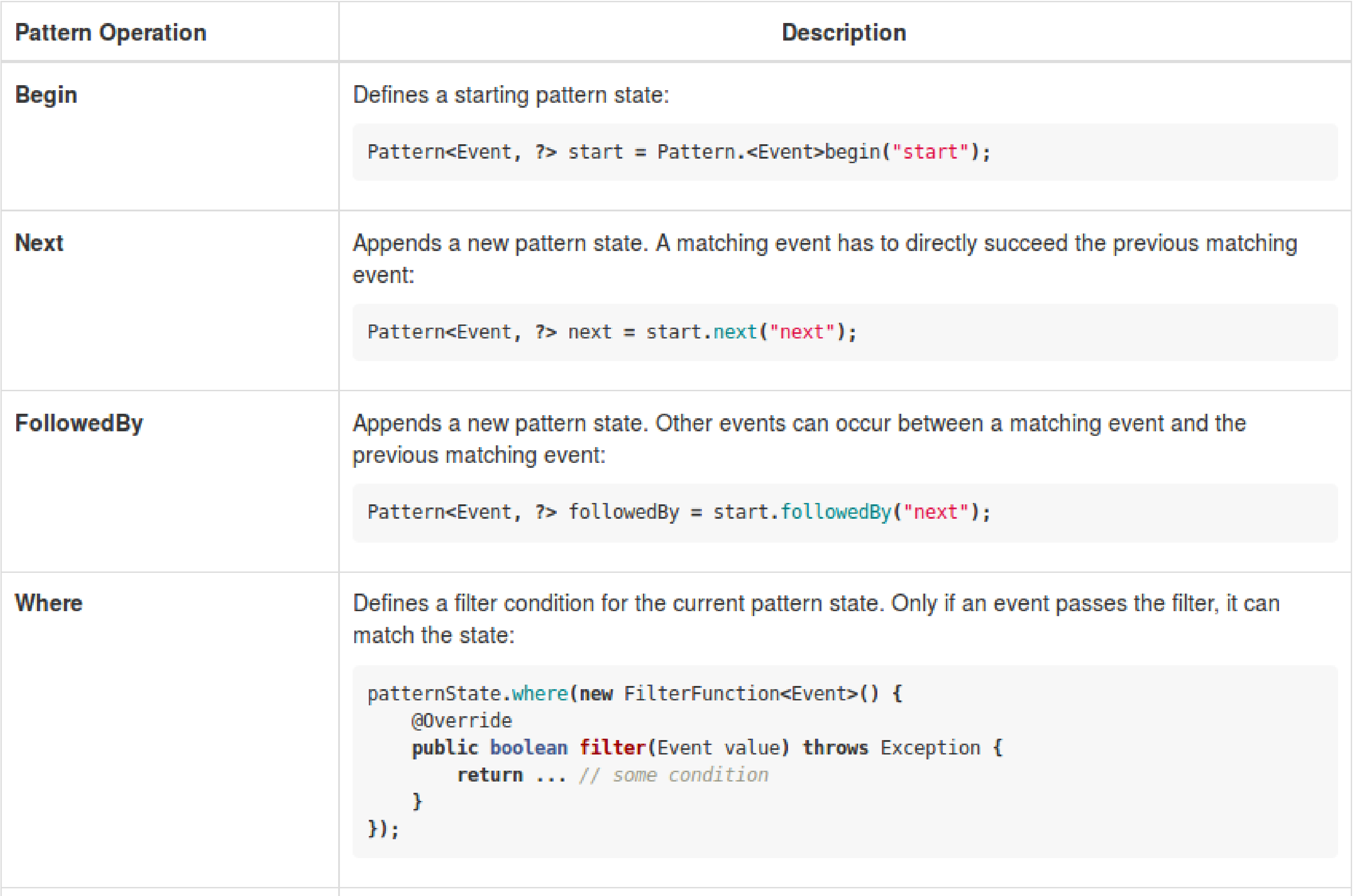
```
stream
       .windowAll(...)      <-  required: "assigner"
      [.trigger(...)]       <-  optional: "trigger" (else default trigger)
      [.evictor(...)]       <-  optional: "evictor" (else no evictor)
      [.allowedLateness()]  <-  optional, else zero
       .reduce/fold/apply() <-  required: "function"
```

Type of windows

- Trigger: send the results
- Evictor: remove elements from a window in certain conditions
- Lateness: allow late time-based events
- Windows function: computation applied to windows

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

Flink CEP Patterns

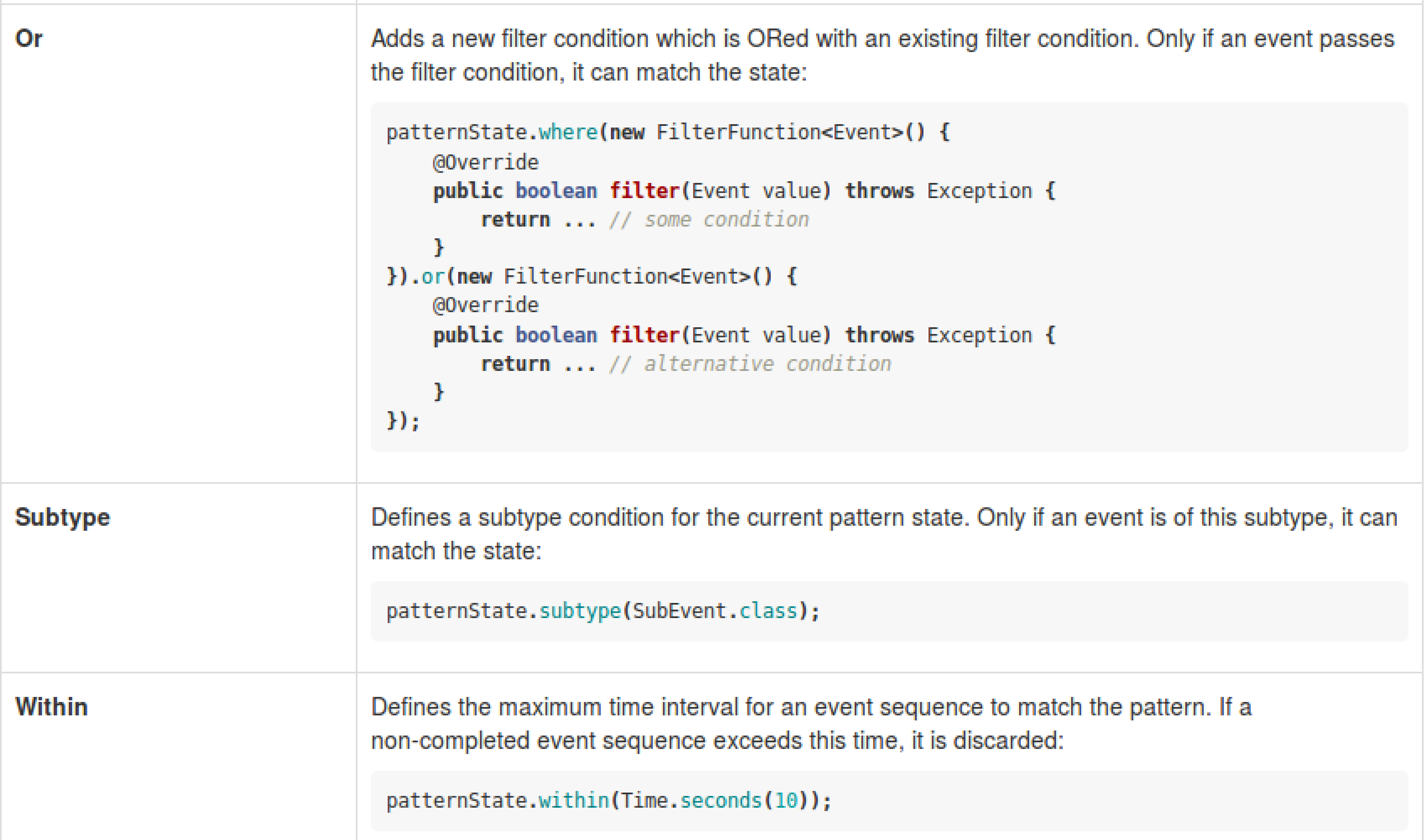


Pattern Operation	Description
Begin	Defines a starting pattern state: `Pattern<Event, ?> start = Pattern.<Event>begin("start");`
Next	Appends a new pattern state. A matching event has to directly succeed the previous matching event: `Pattern<Event, ?> next = start.next("next");`
FollowedBy	Appends a new pattern state. Other events can occur between a matching event and the previous matching event: `Pattern<Event, ?> followedBy = start.followedBy("next");`

Where	Defines a filter condition for the current pattern state. Only if an event passes the filter, it can match the state:
	`patternState.where(new FilterFunction<Event>() {`
	`    @Override`
	`    public boolean filter(Event value) throws Exception {`
	`        return ... // some condition`
	`    }`
	`});`

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

Flink CEP Patterns



Or

Adds a new filter condition which is ORed with an existing filter condition. Only if an event passes the filter condition, it can match the state:

```
patternState.where(new FilterFunction<Event>() {
    @Override
    public boolean filter(Event value) throws Exception {
        return ... // some condition
    }
}).or(new FilterFunction<Event>() {
    @Override
    public boolean filter(Event value) throws Exception {
        return ... // alternative condition
    }
});
```

Subtype

Defines a subtype condition for the current pattern state. Only if an event is of this subtype, it can match the state:

```
patternState.subtype(SubEvent.class);
```

Within

Defines the maximum time interval for an event sequence to match the pattern. If a non-completed event sequence exceeds this time, it is discarded:

```
patternState.within(Time.seconds(10));
```

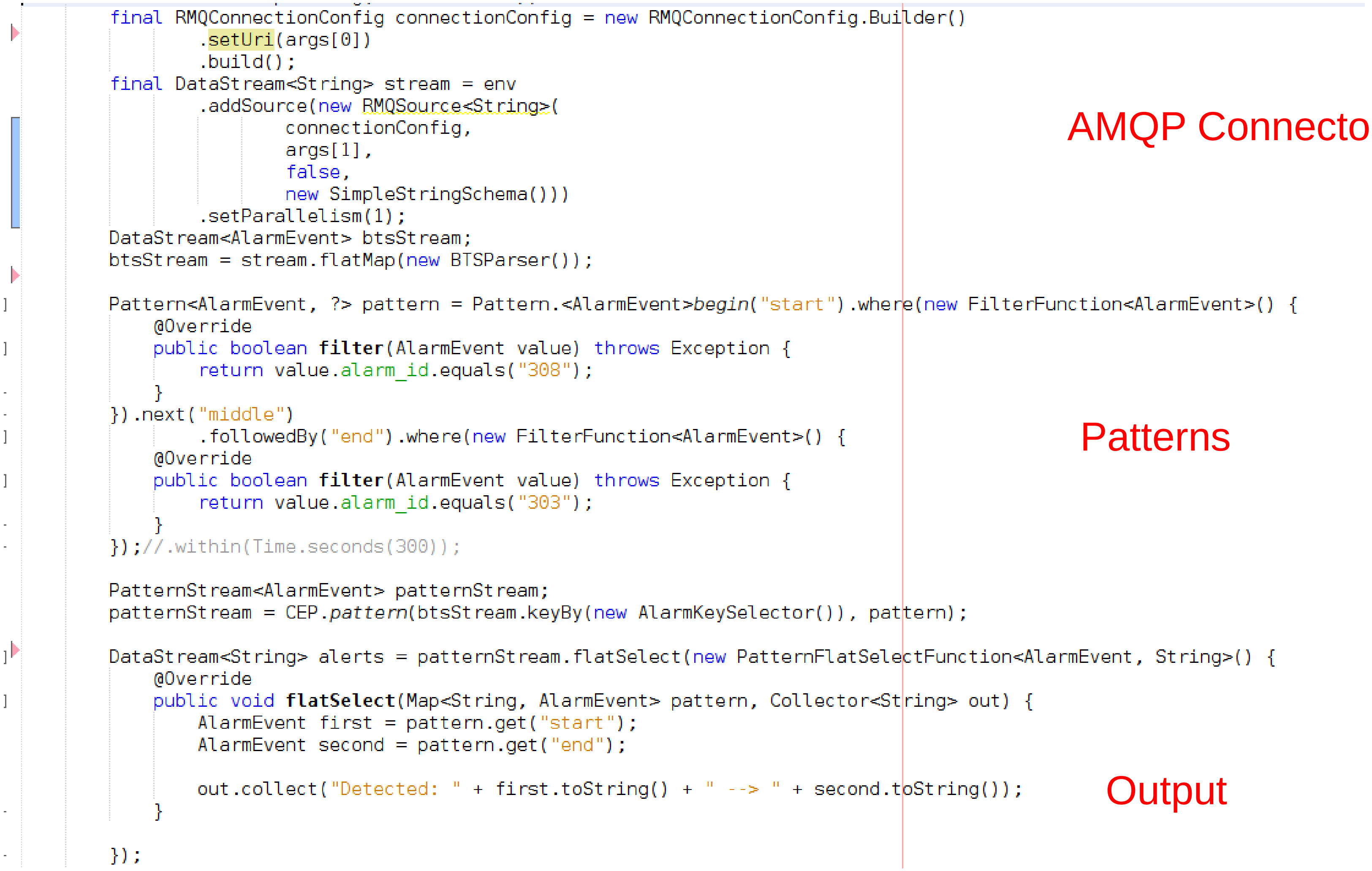
Source: https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/libs/cep.html

Example with Base Transceiver Station

Data

```
station_id,datapoint_id,alarm_id,event_time,value,valueThreshold
1161115016,121,308,2017-02-18 18:28:05 UTC,240,240
1161114050,143,312,2017-02-18 18:56:20 UTC,28.5,28
1161115040,141,312,2017-02-18 18:22:03 UTC,56.5,56
1161114008,121,308,2017-02-18 18:34:09 UTC,240,240
1161115040,141,312,2017-02-18 18:20:49 UTC,56,56
1161114050,143,312,2017-02-18 18:47:40 UTC,28.5,28
1161115016,121,308,2017-02-18 19:01:14 UTC,241,240
1161114061,121,301,2017-02-18 18:59:03 UTC,76,80
1161114011,121,308,2017-02-18 18:51:09 UTC,241,240
```

Simple example



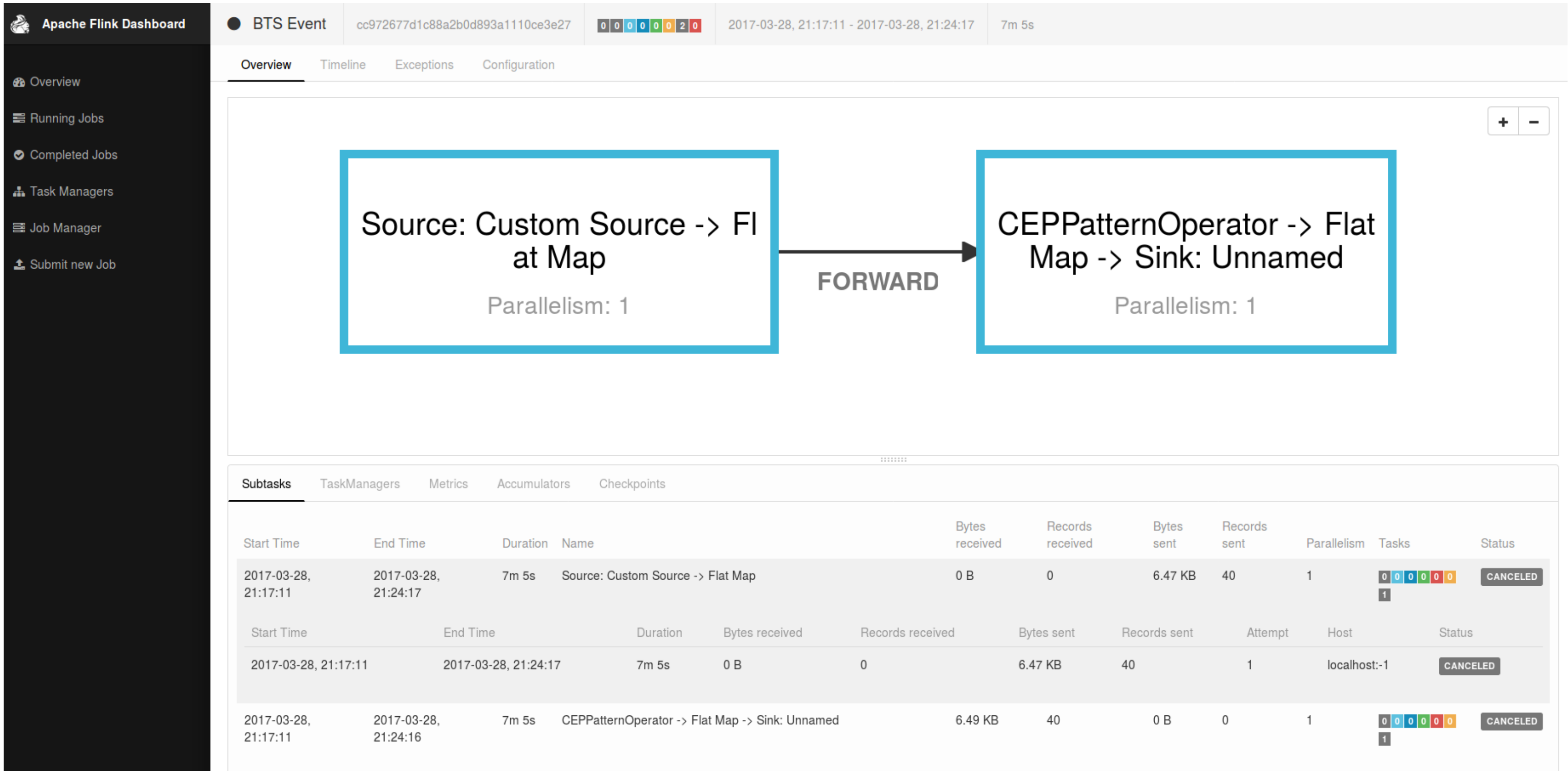
```
final RMQConnectionConfig connectionConfig = new RMQConnectionConfig.Builder()
        .setUri(args[0])
        .build();
final DataStream<String> stream = env
        .addSource(new RMQSource<String>(
                connectionConfig,
                args[1],
                false,
            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

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

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

DataStream<String> alerts = patternStream.flatSelect(new PatternFlatSelectFunction<AlarmEvent, String>() {
    @Override
    public void flatSelect(Map<String, AlarmEvent> pattern, Collector<String> out) {
        AlarmEvent first = pattern.get("start");
        AlarmEvent second = pattern.get("end");

        out.collect("Detected: " + first.toString() + " --> " + second.toString());
    }

});
```

AMQP Connector

Patterns

Output

Monitoring



● BTS Event | cc972677d1c88a2b0d893a1110ce3e27 | 0 0 0 0 0 0 2 0 | 2017-03-28, 21:17:11 - 2017-03-28, 21:24:17 | 7m 5s

Source: Custom Source -> Fl at Map

Parallelism: 1

CEPPatternOperator -> Flat Map -> Sink: Unnamed

Parallelism: 1

Start Time	End Time	Duration	Name	Bytes received	Records received	Bytes sent	Records sent	Parallelism	Tasks	Status
2017-03-28, 21:17:11	2017-03-28, 21:24:17	7m 5s	Source: Custom Source -> Flat Map	0 B	0	6.47 KB	40	1	0 0 0 0 0 0 1	CANCELED
2017-03-28, 21:17:11	2017-03-28, 21:24:16	7m 5s	CEPPatternOperator -> Flat Map -> Sink: Unnamed	6.49 KB	40	0 B	0	1	0 0 0 0 0 0 1	CANCELED

Start Time	End Time	Duration	Bytes received	Records received	Bytes sent	Records sent	Attempt	Host	Status
2017-03-28, 21:17:11	2017-03-28, 21:24:17	7m 5s	0 B	0	6.47 KB	40	1	localhost:-1	CANCELED

Results

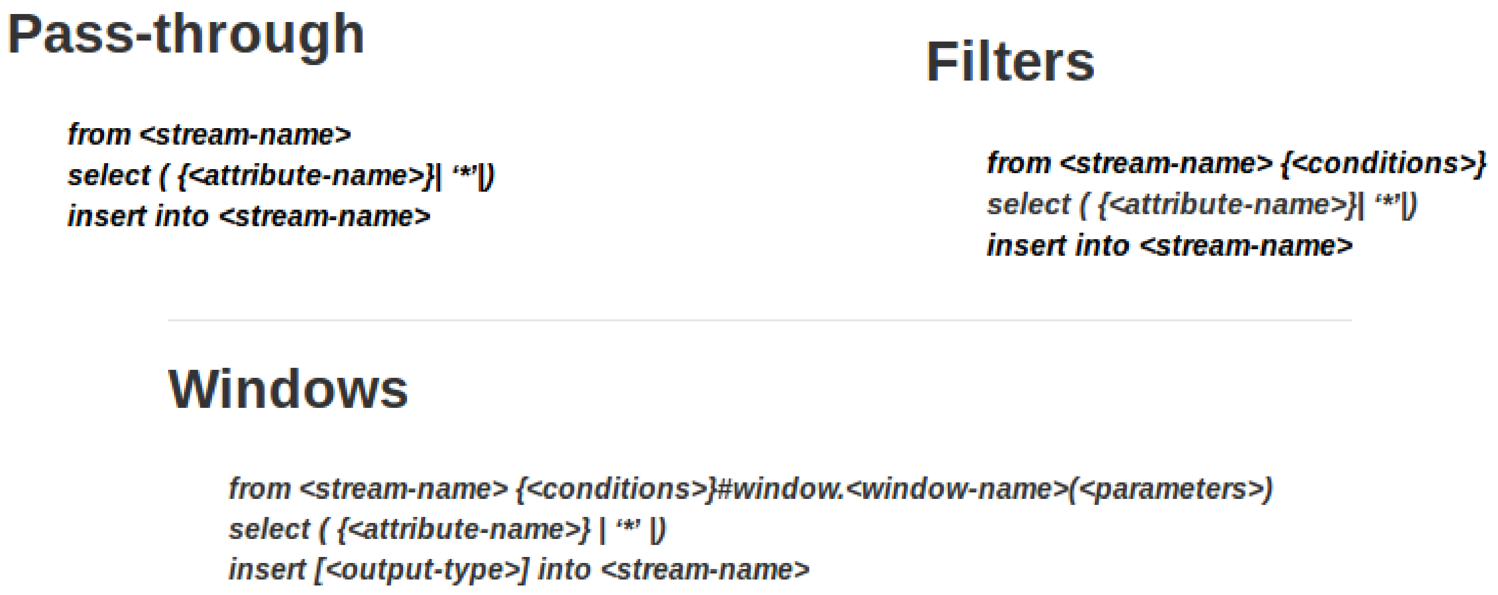
```
Detected: station_id=1161115006 for datapoint_id=121 at Sat Feb 18 21:54:30 CET
2017 alarm_id=308 with value =240.0 --> station_id=1161115006 for
datapoint_id=116 at Sun Feb 19 02:20:22 CET 2017 alarm_id=303 with value
=999999.0
```

```
Detected: station_id=1161114011 for datapoint_id=121 at Sat Feb 18 20:57:34 CET
2017 alarm_id=308 with value =241.0 --> station_id=1161114011 for
datapoint_id=116 at Sun Feb 19 00:59:18 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>
```

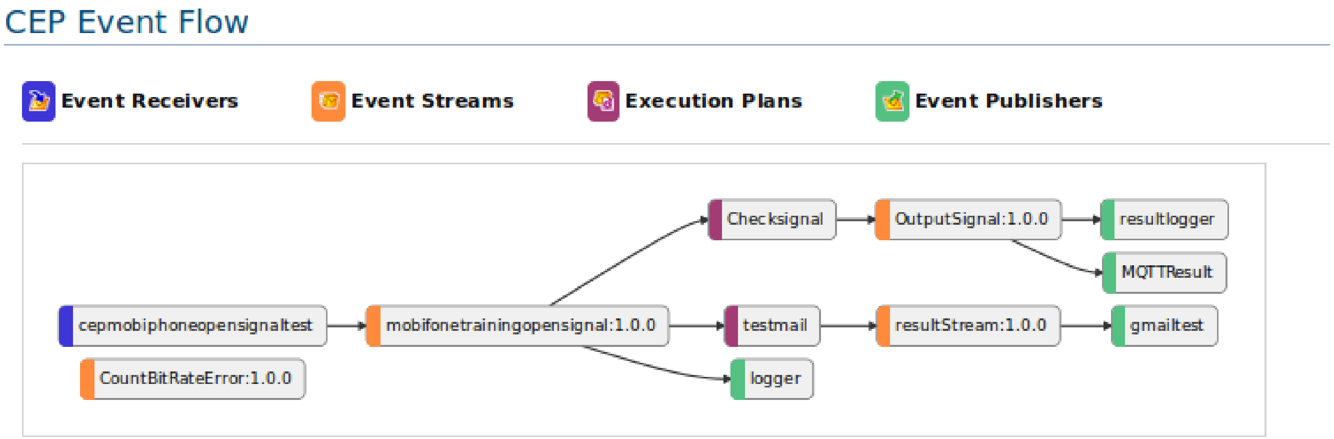
Source: https://docs.wso2.com/display/CEP420/SiddhiQL+Guide+3.1

SQL-alike conditions

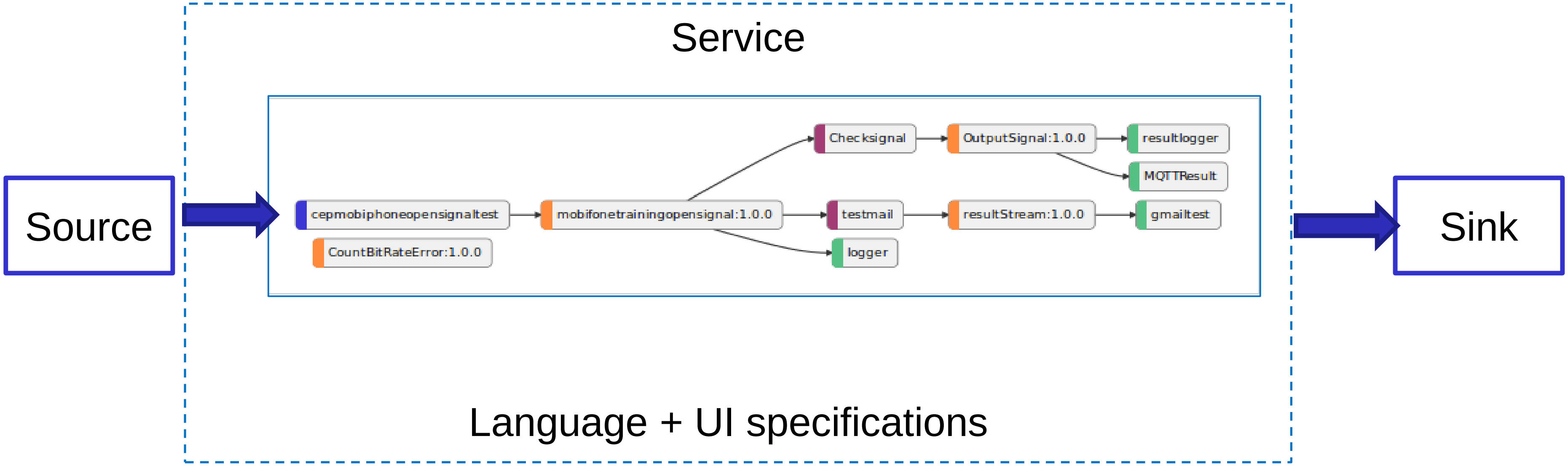
```
@Import('mobifonetrainingopensignal:1.0.0')
define stream inStream (meta_USERPHONE int, meta_TIME long, correlation_lat float,
    correlation_lon float, GSM_BIT_ERROR_RATE float, GSM_SIGNAL_STRENGTH float,
    LOC_ACCURACY float, LOC_SPEED float);



@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



Example with WSO2 Carbon CEP

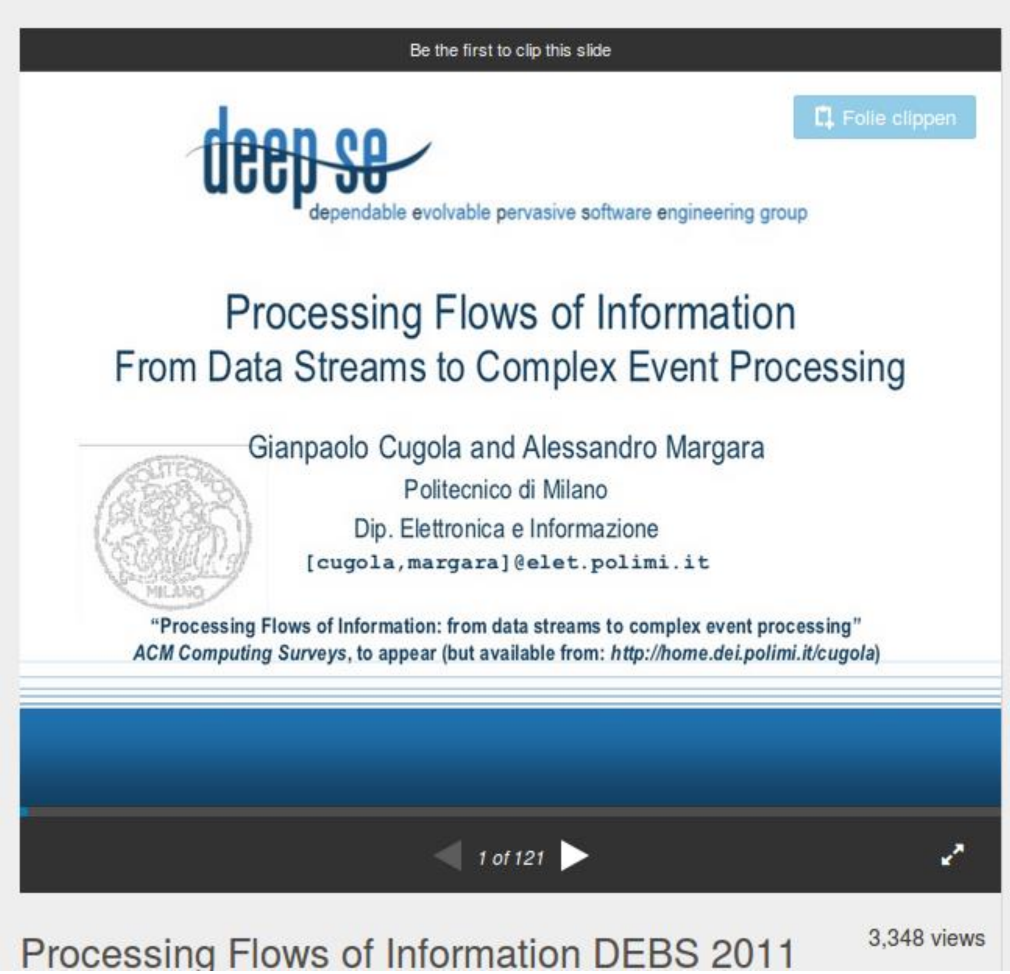


Service

Language + UI specifications

Get a high-level view

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



Be the first to clip this slide

Processing Flows of Information
From Data Streams to Complex Event Processing

Gianpaolo Cugola and Alessandro Margara
Politecnico di Milano
Dip. Elettronica e Informazione
[cugola,margara]@elet.polimi.it

"Processing Flows of Information: from data streams to complex event processing"
ACM Computing Surveys, to appear (but available from: *http://home.dei.polimi.it/cugola*)

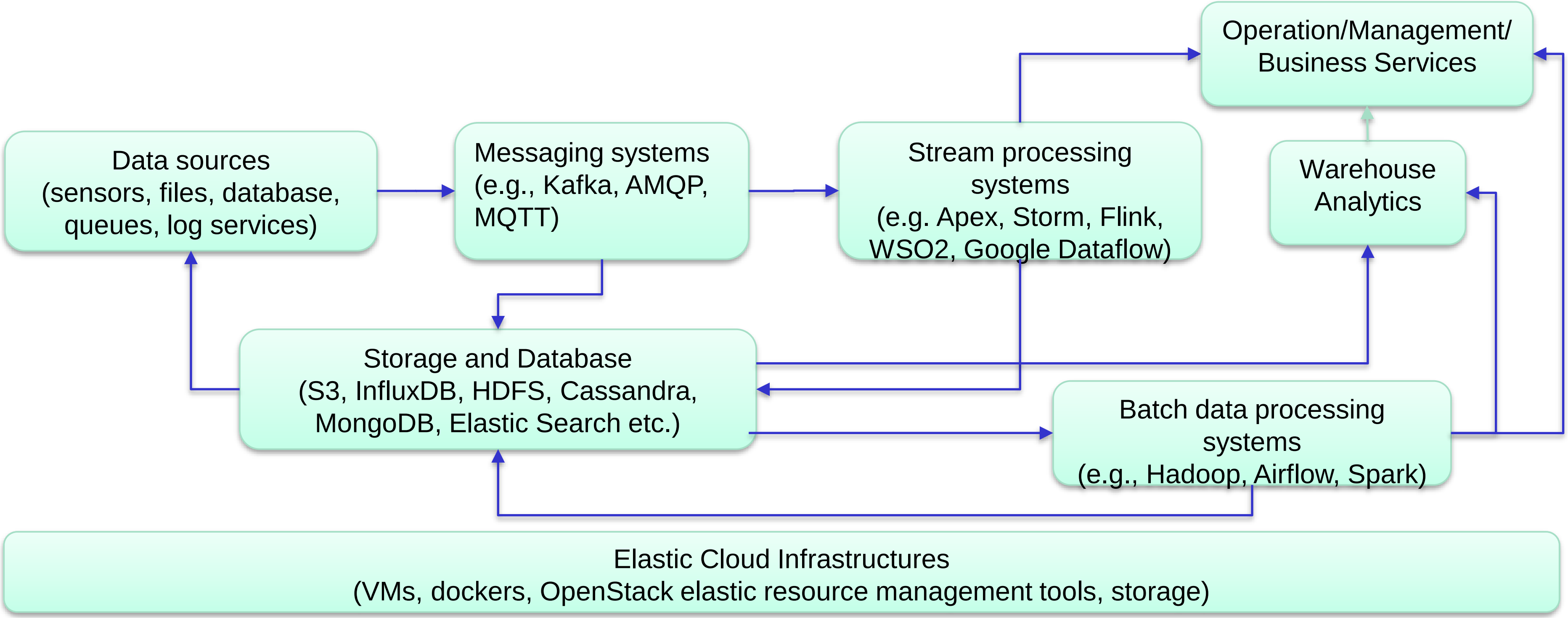
Processing Flows of Information DEBS 2011

3,348 views

Partially covered in Lecture 5

BEYOND BASIC MESSAGE PROCESSING

Cloud services and big data analytics



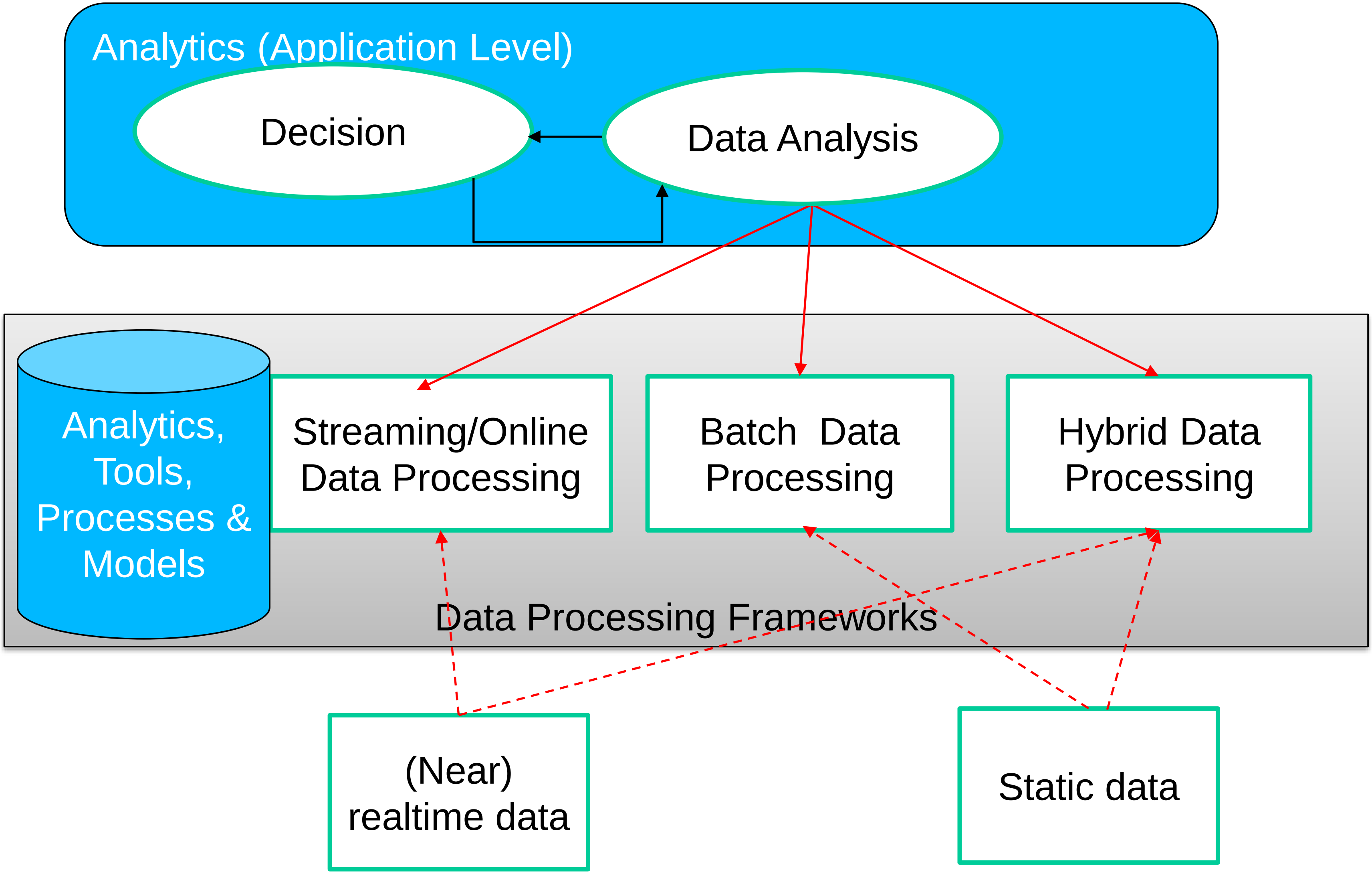
Operation/Management/ Business Services
Data sources (sensors, files, database, queues, log services)
Messaging systems (e.g., Kafka, AMQP, MQTT)
Stream processing systems (e.g. Apex, Storm, Flink, WSO2, Google Dataflow)
Warehouse Analytics
Storage and Database (S3, InfluxDB, HDFS, Cassandra, MongoDB, Elastic Search etc.)
Batch data processing systems (e.g., Hadoop, Airflow, Spark)
Elastic Cloud Infrastructures (VMs, dockers, OpenStack elastic resource management tools, storage)

Data Processing Framework

- Batch processing
  - Mapreduce/Hadoop
  - Scientific workflows
- (Near) realtime streaming processing
  - Flink, Apex, 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


Analytics (Application Level)
Decision
Data Analysis
Analytics, Tools, Processes & Models
Streaming/Online Data Processing
Batch Data Processing
Hybrid Data Processing
Data Processing Frameworks
(Near) realtime data
Static 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

Thanks for your attention

Hong-Linh Truong
Distributed Systems Group, TU Wien
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong