REBS #7: Event Processing

Hugo A. López
Software, Data, People and Society

UNIVERSITY OF COPENHAGEN

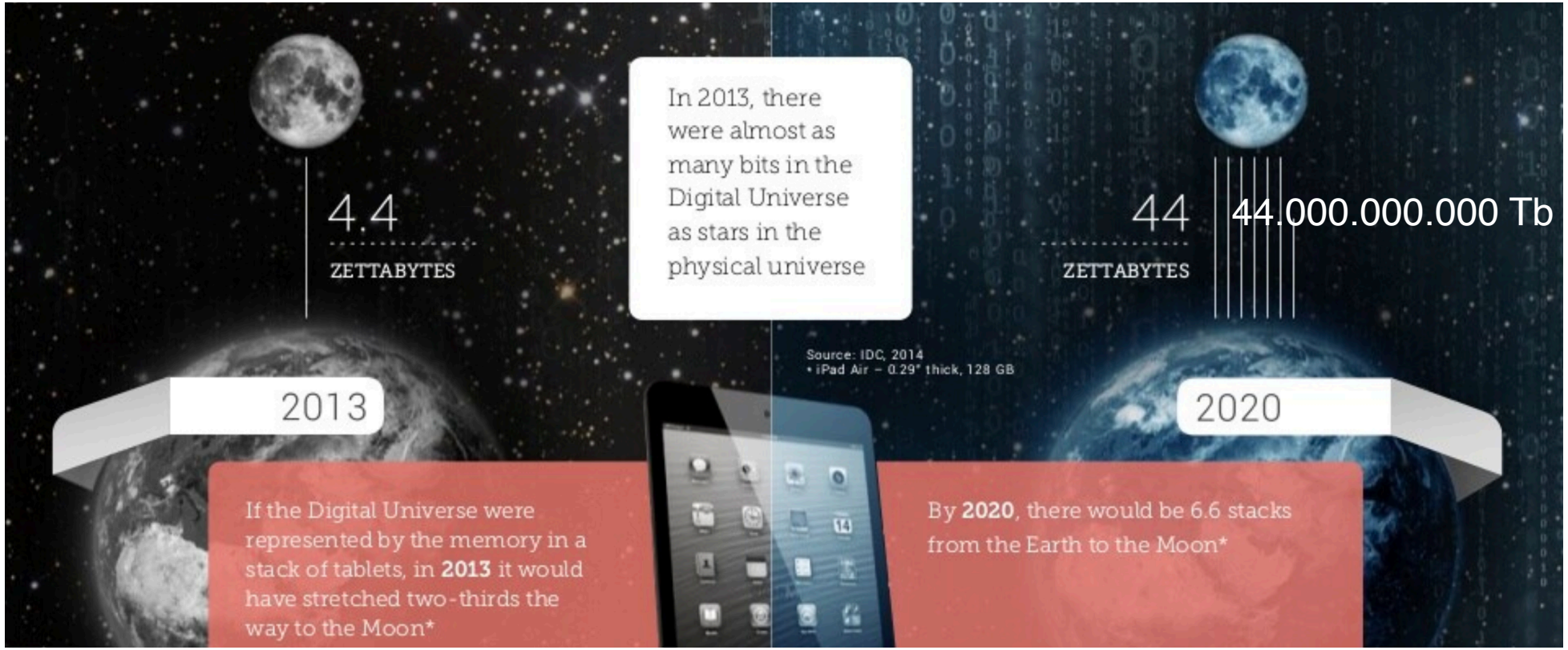
Intended Learning Outcomes

- Get an overview of the challenges of online process mining and the required infrastructure to deal with streaming data

An Internet of Events

- Data are collected about anything, at any time, and at any place. “Big Data” is often used to refer the expanding capabilities of information systems and other systems that depend on computing.

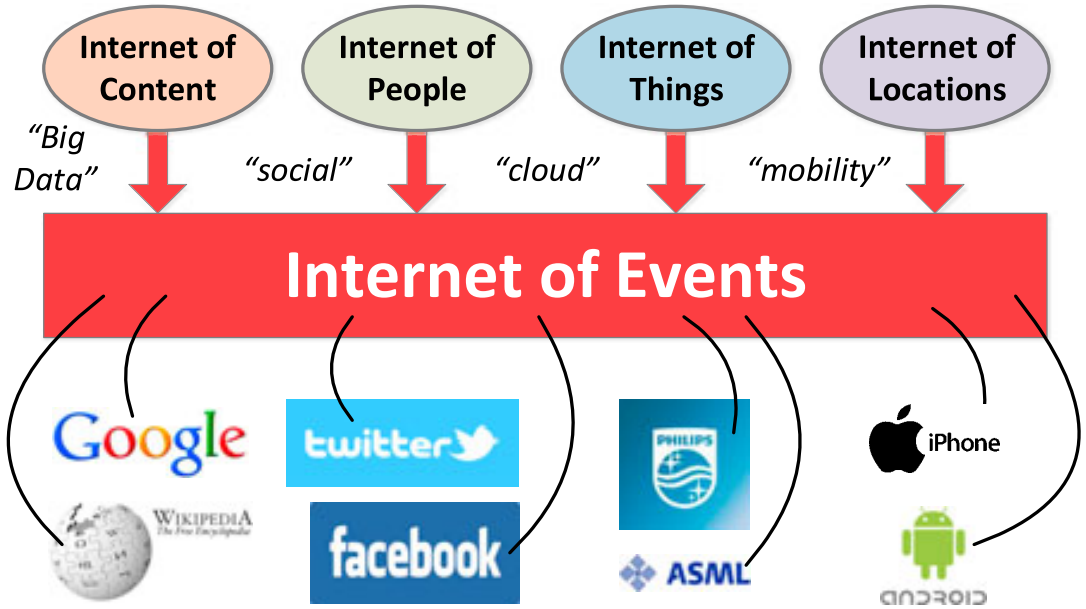
W. van der Aalst, *Process Mining*. *Springer, 2016*



https://www.slideshare.net/abdullah_nabulsi/idc-digitaluniverse2014-33810387

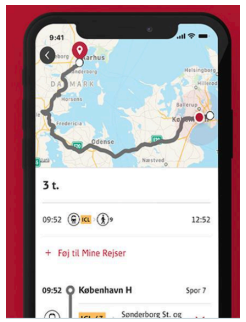
Making sense of the process

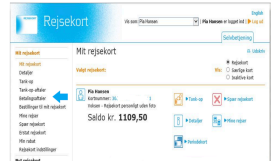
- Events capture the “observables” in a process. They do not come from a single source, but different event providers



W. van der Aalst, *Process Mining*. *Springer, 2016*

Example: Customer Journey





Anne

1 How to travel to Amsterdam?

2 Visit website DSB

3 Visit website OV-chipcard

4 Purchase OV-chipcard

5 Check schedule

6 Share schedule with friends

7 Load credit onto card

8 Check-in using card (train)

9 Install app

10 Tweet problem

11 Get delay notification

12 Check-out using card (train)

13 Check credit

14 Request refund

15 Check-in using card (bus)

16 Check-out using card (bus)

17 Get refund

18 Plan next trip

awareness of product or brand

orientation

planning/ shopping

purchase or booking

(wait for) delivery

consume, use, experience

after sales, follow-up, complaints handling

social media

website

e-mail

TV, radio

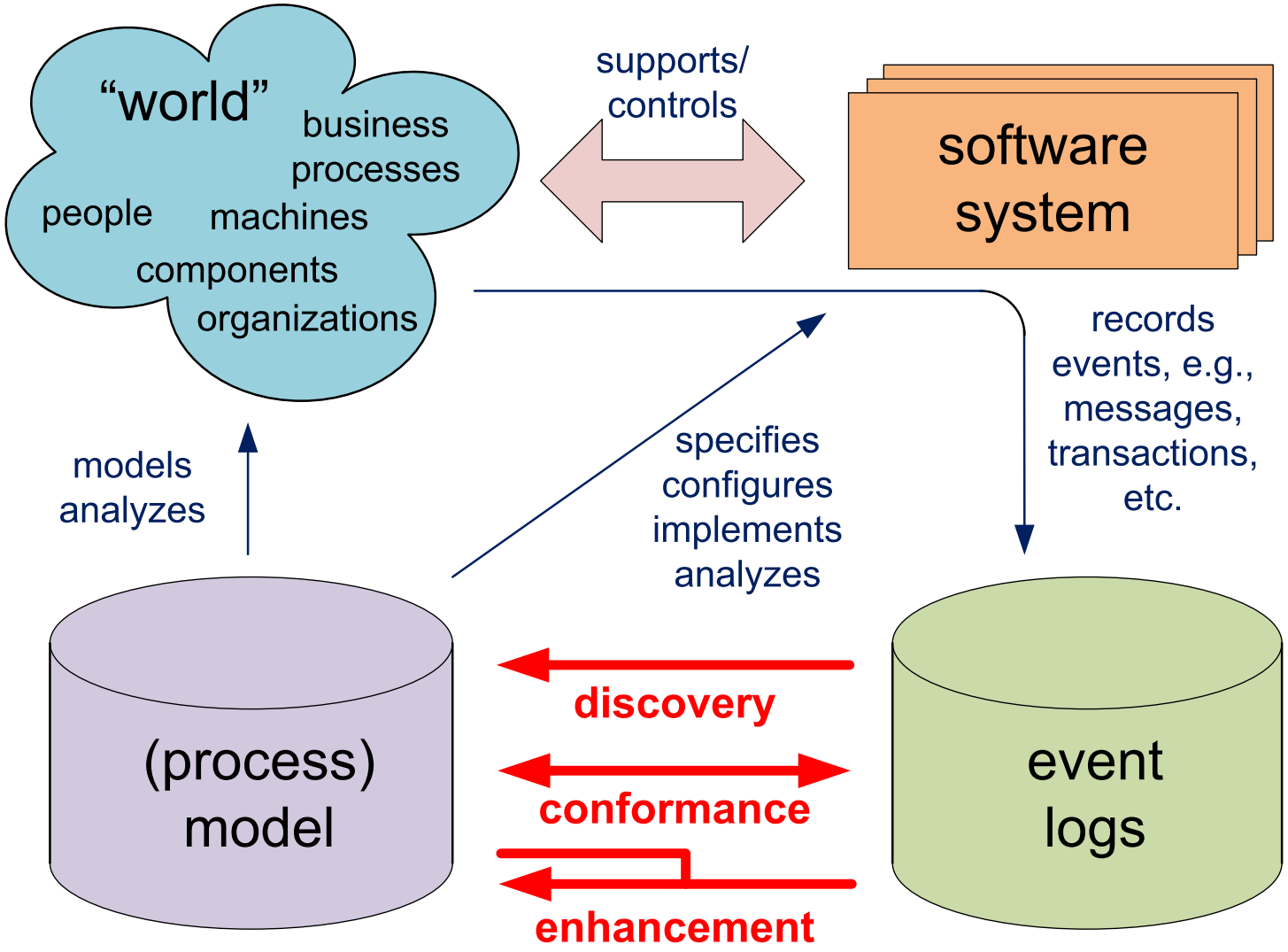
call center

face-to-face

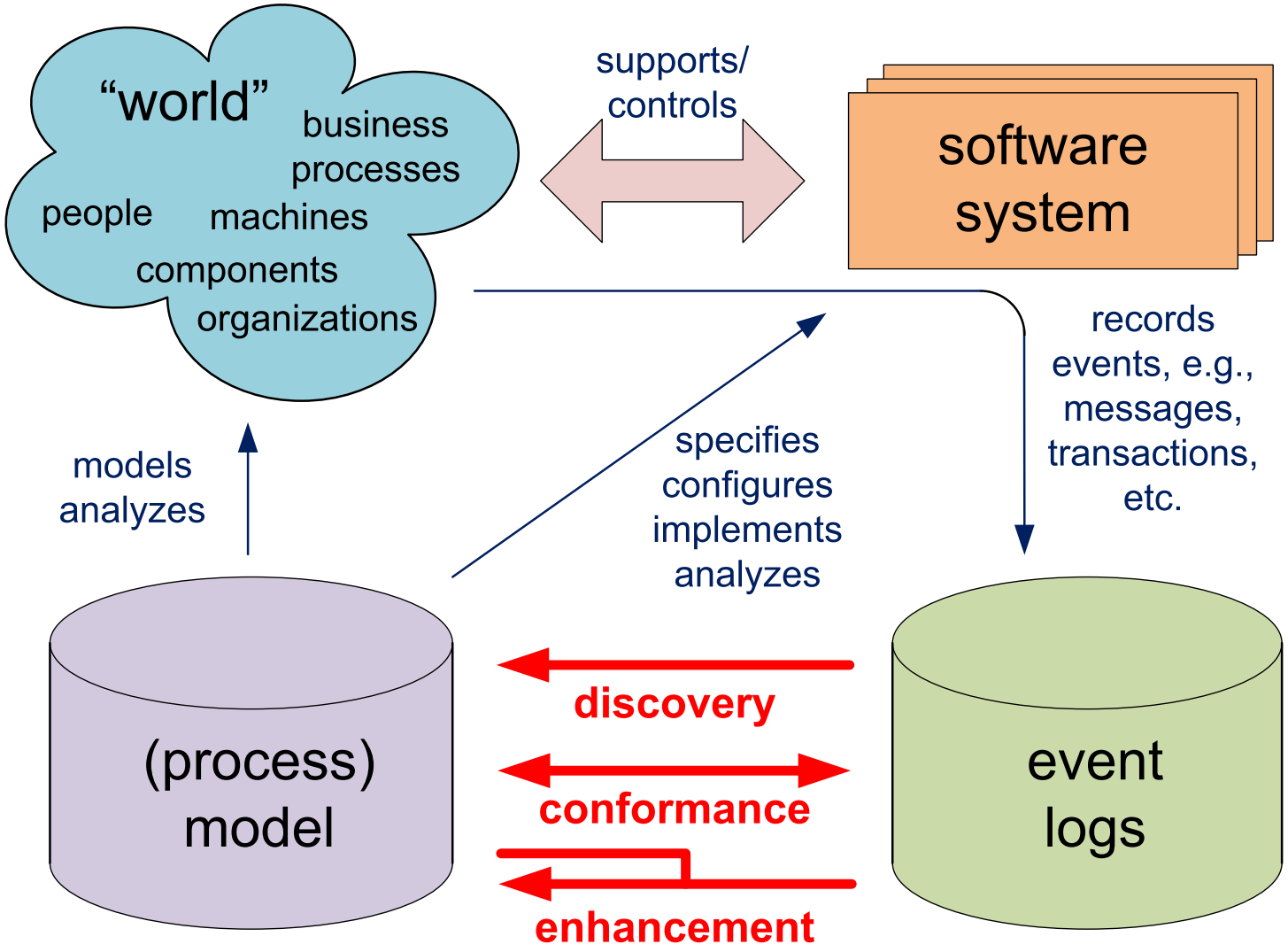
other

W. van der Aalst, *Process Mining*. *Springer, 2016*

Process Mining



Process Mining



What if our observations change over time? New events coming, new cases spanned, etc?

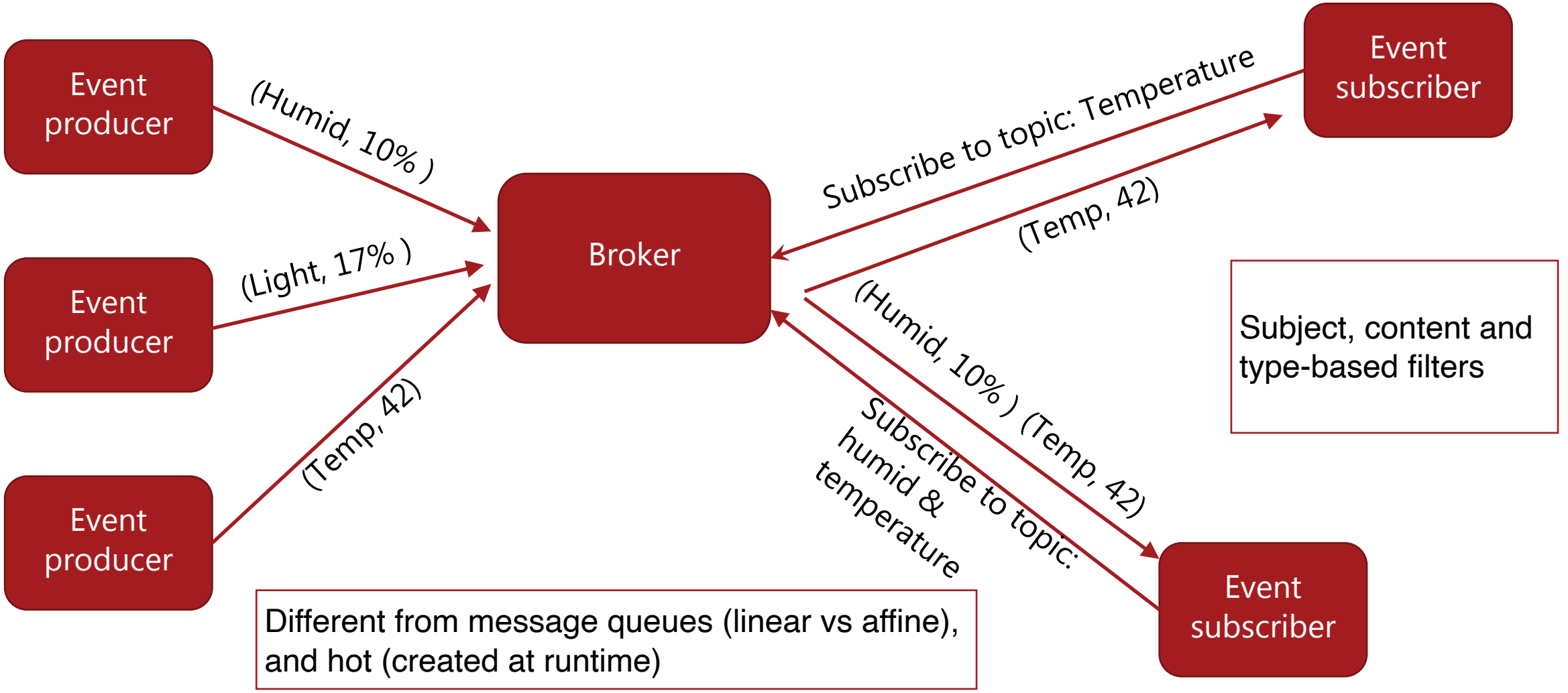
Communicating Event-Based Architectures

- Event streams, message queues, brokers and the publish/subscribe architecture pattern are key elements of event-based systems

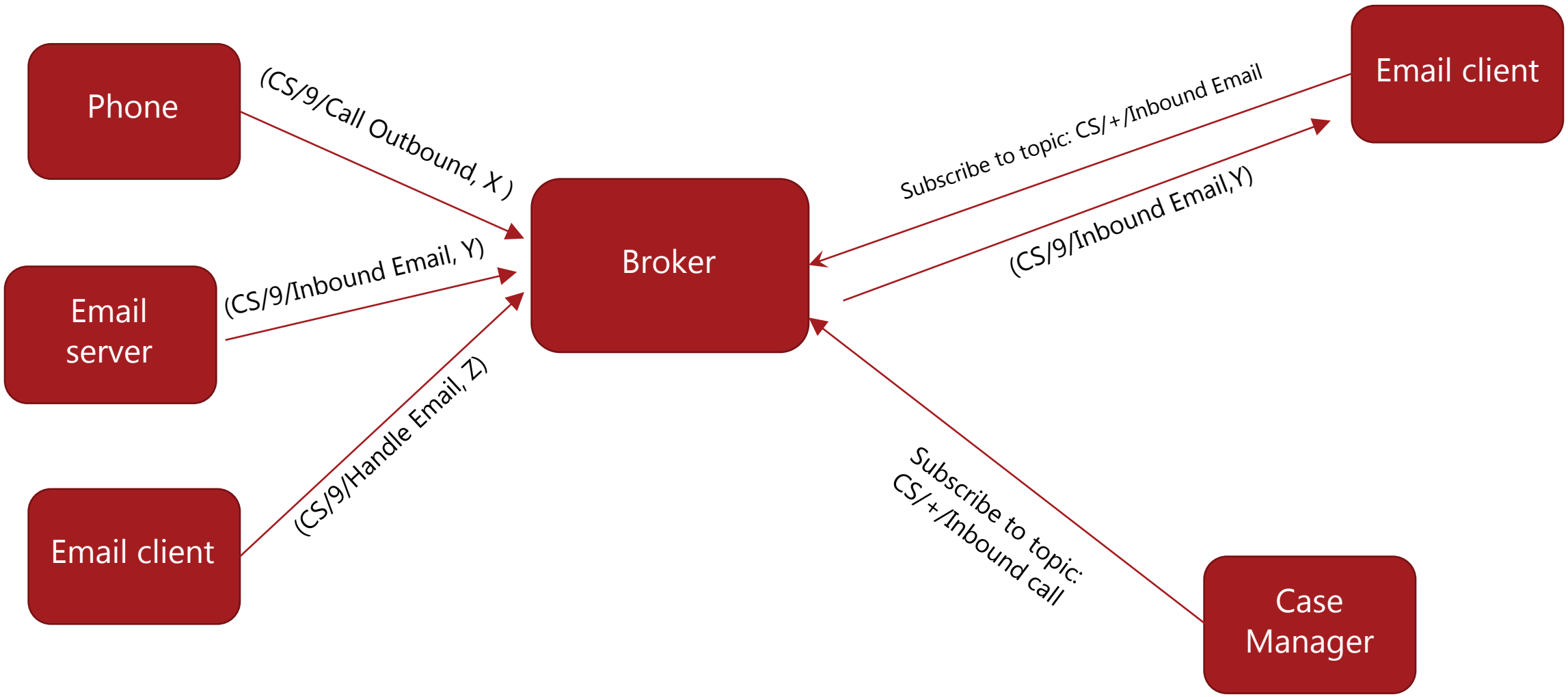
- Internet of Things (IoT) are particular event-based systems that require lightweight protocols.

Event streams, broker and publish/subscribe architecture

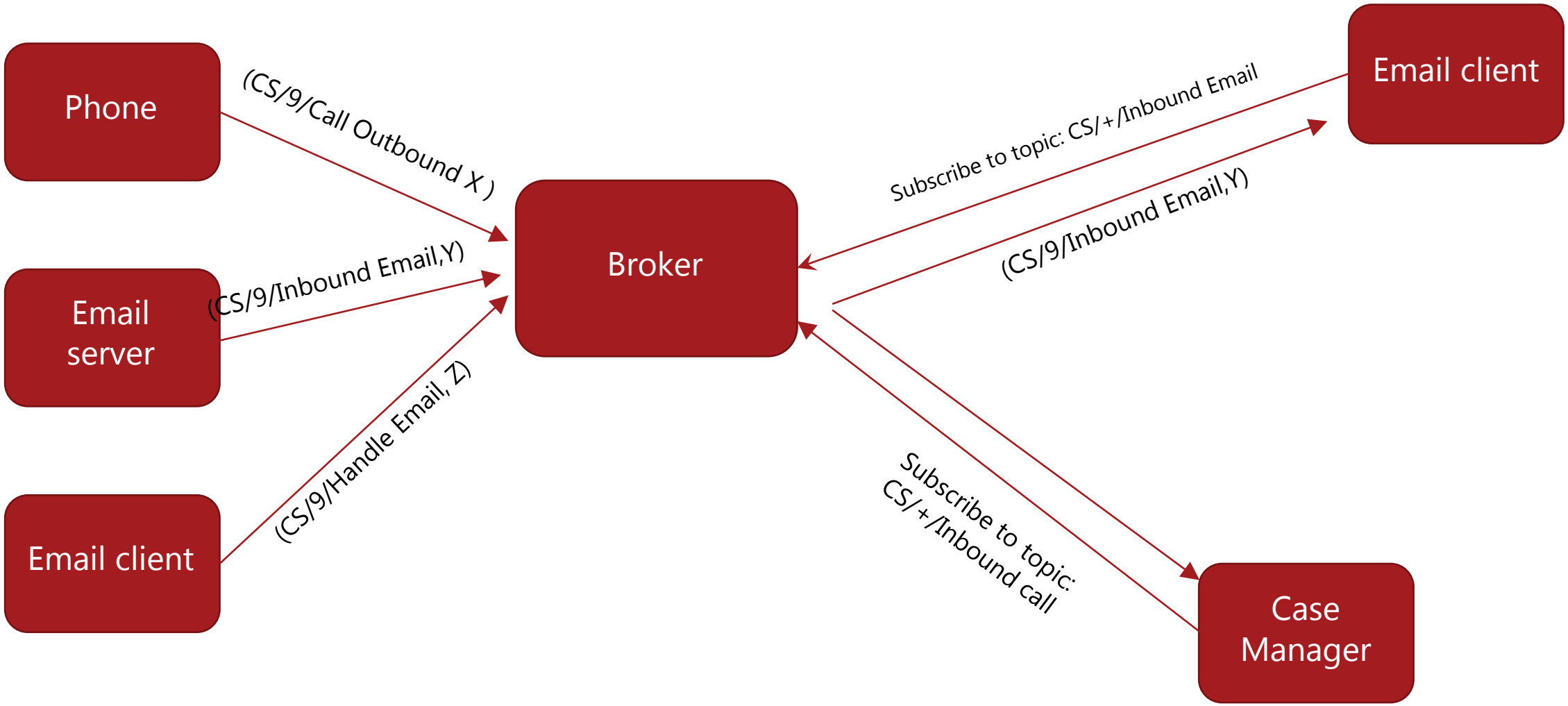
Space, time and synchronisation decoupling



Example: Case Management at call center



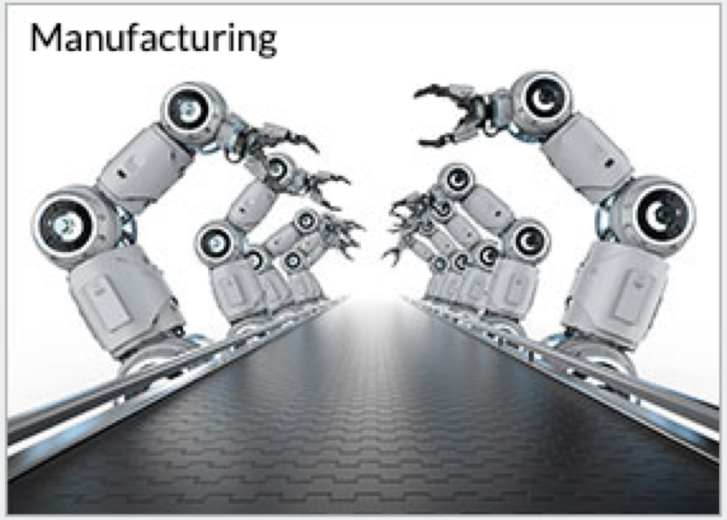
Example: Case Management at call center



Message Queuing Telemetry Transport MQTT Protocol (MQTT.org)

- Invented at IBM & Arcom in 1999 (Communication from oil pipes to satelites)
- Released as open protocol in 2010. The Paho eclipse project
- OASIS standard (2014) publish/subscribe messaging transport protocol
- Lightweight, Reliable, Bi-directional, Scalable, Security enabled

Application Areas

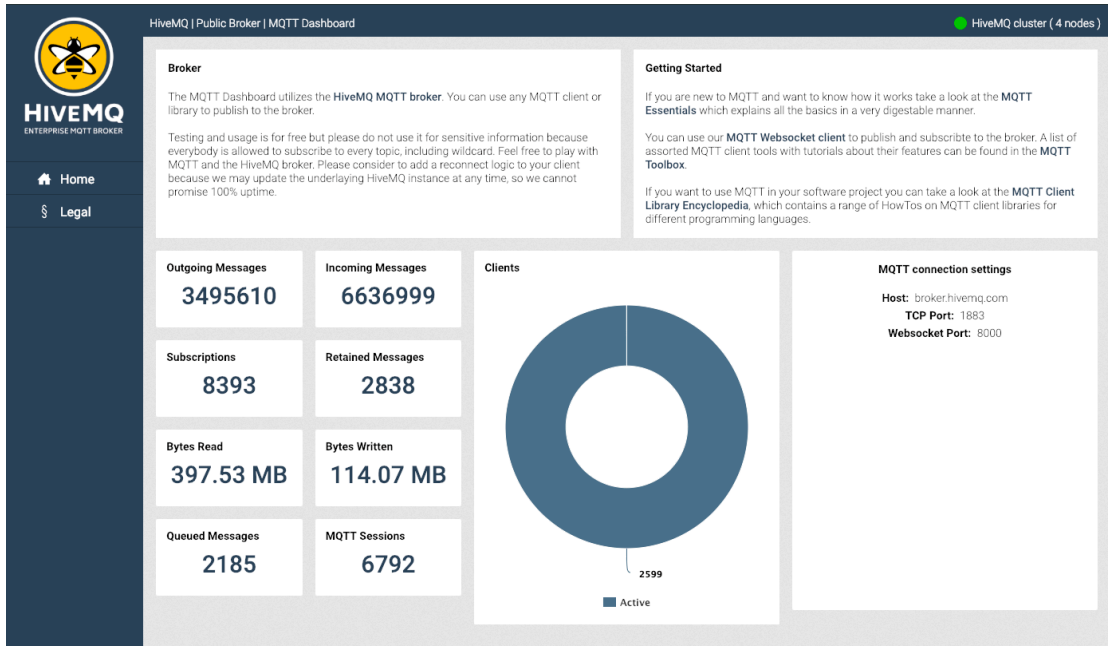


Try MQTT via HyveMQ

https://www.hivemq.com/public-mqtt-broker/

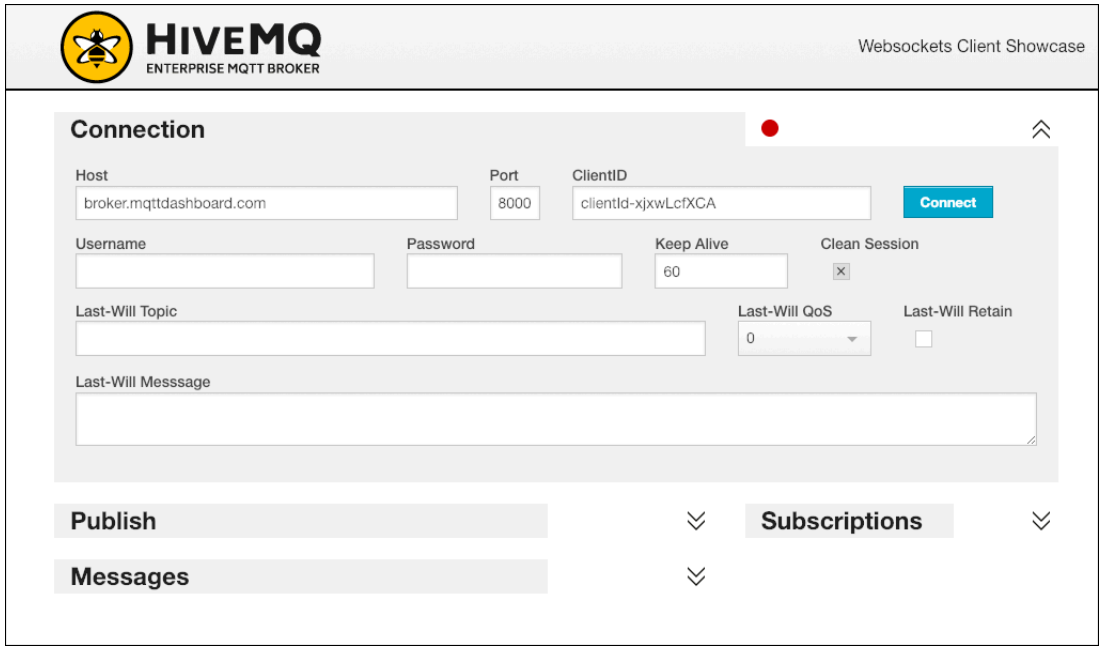
Public MQTT Broker

http://www.mqtt-dashboard.com/



MQTT Browser Client

http://www.hivemq.com/demos/websocket-client/



Broker: **`broker.hivemq.com`**

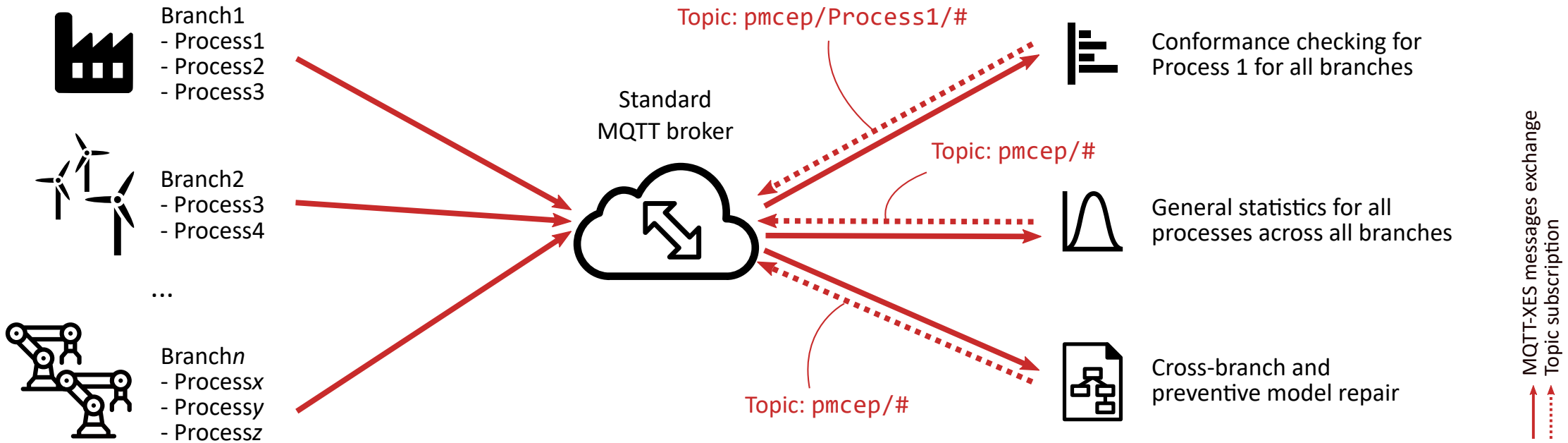
TCP Port: **`1883`**

Websocket Port: **`8000`**

Example: Streaming Business Process Events

MQTT-XES: Real-time Telemetry for Process Event Data

Andrea Burattin[1], Martin Eigenmann[2], Ronny Seiger[2], and Barbara Weber[2]

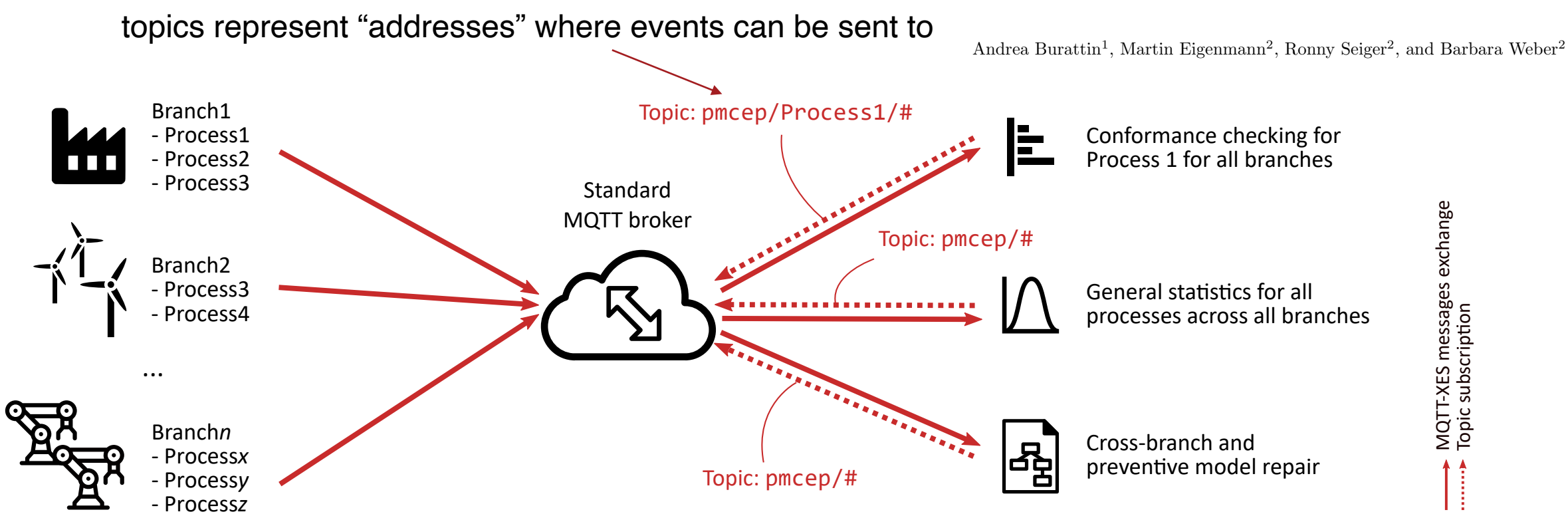


- Rationale: to have timely snapshots of the process, reducing the delay between the observations and the decisions made

Example: Streaming Business Process Events

MQTT-XES: Real-time Telemetry for Process Event Data

Andrea Burattin[1], Martin Eigenmann[2], Ronny Seiger[2], and Barbara Weber[2]



- Rationale: to have timely snapshots of the process, reducing the delay between the observations and the decisions made

Anatomy of a streamed event

- Hierarchies of topics:

BASE/[SOURCE ID]/[PROCESS INSTANCE ID]/[ACTIVITY NAME]

  - Base: all event part of a monitoring initiative
  - Source ID: Source of the event (f.x. process model that generates it)
  - Process Instance ID: Case
  - Activity Name: name of activity executed
  - Result

{"event":
{"Role":"recover","concept:name":"DL_WFCaseTypeStep|485","EventType
":"Task","Title":"Round approved","EventName":"Round
approved","lifecycle:transition":"complete"},"trace":
{"case:variant":"15","concept:name":"14b-330_0"}}

Existence of the event in a topic indicates its occurrence (control flow)

Between logs and streams

- A log is a static data structure, and allows different analysis than a stream:

Data-Based Management Systems	Stream-based Management Systems
Persistent Relations	Transient Relations
One-time queries	Continuous Queries
Random Access	Sequential Access
Access Plan determined by query processor and physical DB design	Unpredictable data characteristics and arrival patterns

Event #	Activity	Originator	Time	
Case id: C1				
1	A	U1	2017-09-01	...
2	B	U1	2017-09-02	...
3	C	U2	2017-09-03	...
4	E	U2	2017-09-04	...
Case id: C2				
1	A	U1	2017-09-02	...
2	B	U1	2017-09-03	...
3	D	U3	2017-09-04	...
4	E	U3	2017-09-05	...

Typical event log

Time	Case id	Activity	Originator	
2017-09-01	C1	A	U1	...
2017-09-02	C2	A	U1	...
2017-09-02	C1	B	U1	...
2017-09-03	C1	C	U2	...
2017-09-03	C2	B	U1	...
2017-09-04	C1	E	U2	...
2017-09-04	C2	D	U3	...
2017-09-05	C2	E	U3	...
...				

Typical stream

Between logs and streams

- A log is a static data structure, and allows different analysis than a stream:

Data-Based Management Systems	Stream-based Management Systems
Persistent Relations	Transient Relations
One-time queries	Continuous Queries
Random Access	Sequential Access
Access Plan determined by query processor and physical DB design	Unpredictable data characteristics and arrival patterns

Event #	Activity	Originator	Time	...
Case id: C1				
1	A	U1	2017-09-01	...
2	B	U1	2017-09-02	...
3	C	U2	2017-09-03	...
4	E	U2	2017-09-04	...
Case id: C2				
1	A	U1	2017-09-02	...
2	B	U1	2017-09-03	...
3	D	U3	2017-09-04	...
4	E	U3	2017-09-05	...

Typical event log

Time	Case id	Activity	Originator	...
2017-09-01	C1	A	U1	...
2017-09-02	C2	A	U1	...
2017-09-02	C1	B	U1	...
2017-09-03	C1	C	U2	...
2017-09-03	C2	B	U1	...
2017-09-04	C1	E	U2	...
2017-09-04	C2	D	U3	...
2017-09-05	C2	E	U3	...
...				

Typical stream

Between logs and streams

- A log is a static data structure, and allows different analysis than a stream:

Data-Based Management Systems	Stream-based Management Systems
Persistent Relations	Transient Relations
One-time queries	Continuous Queries
Random Access	Sequential Access
Access Plan determined by query processor and physical DB design	Unpredictable data characteristics and arrival patterns

Event #	Activity	Originator	Time	
	Case id: C1			
1	A	U1	2017-09-01	...
2	B	U1	2017-09-02	...
3	C	U2	2017-09-03	...
4	E	U2	2017-09-04	...
	Case id: C2			
1	A	U1	2017-09-02	...
2	B	U1	2017-09-03	...
3	D	U3	2017-09-04	...
4	E	U3	2017-09-05	...

Typical event log

Time	Case id	Activity	Originator	
2017-09-01	C1	A	U1	...
2017-09-02	C2	A	U1	...
2017-09-02	C1	B	U1	...
2017-09-03	C1	C	U2	...
2017-09-03	C2	B	U1	...
2017-09-04	C1	E	U2	...
2017-09-04	C2	D	U3	...
2017-09-05	C2	E	U3	...
		...		

Typical stream

Characteristics of Process mining streams

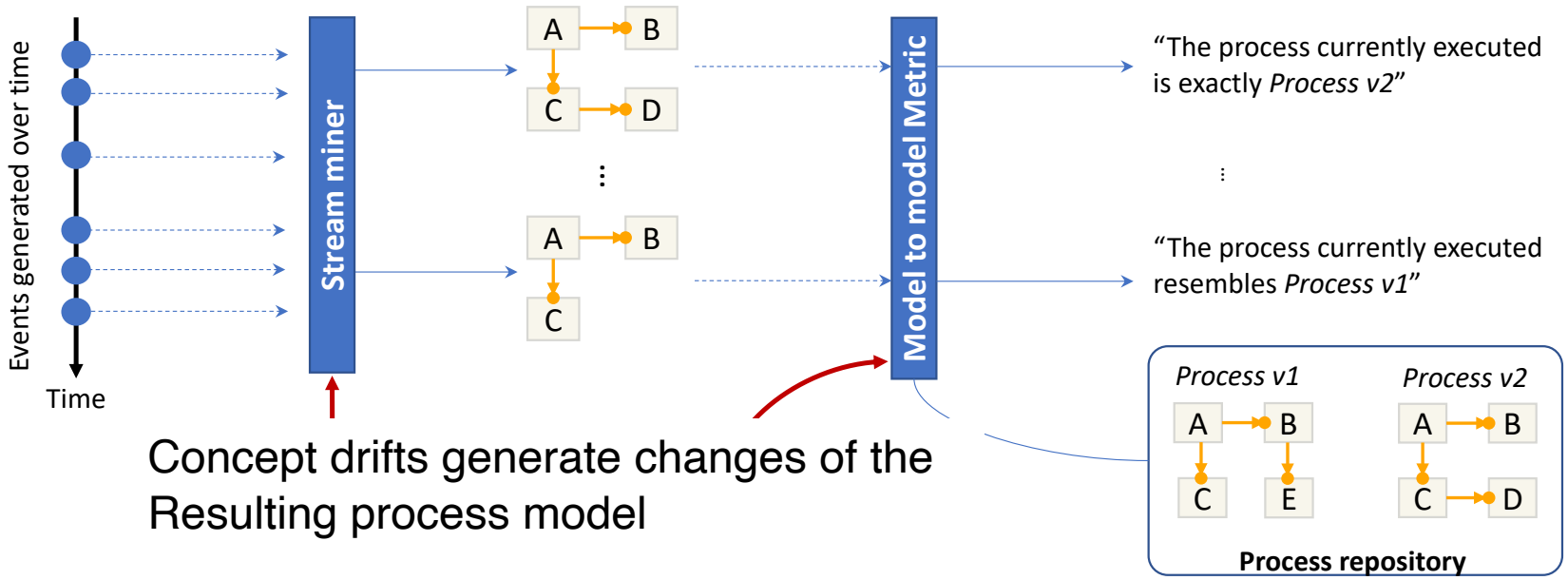
1. Impossible to store the entire stream (possibly infinite)
2. Backtracking not feasible (as we cannot store). Need to design constant-time operations to scale linearly wrt # processed events
3. Needs to adapt with unusual data values (concept drifts)

Example: Process Discovery of DCR graphs

Uncovering and Quantifying Change:
A Streaming Approach for Declarative Processes

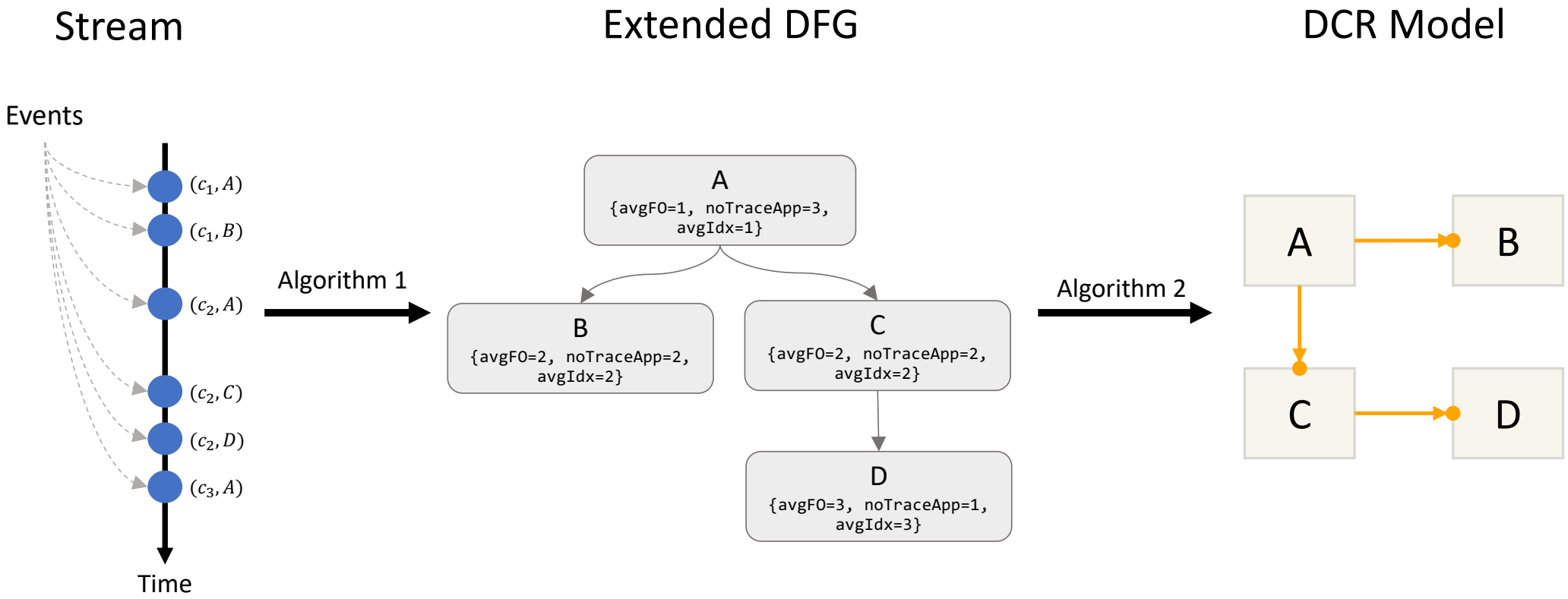
Lasse Starklit[1], Hugo A. López[2], and Andrea Burattin[1]

https://github.com/beamline/dcr-miners



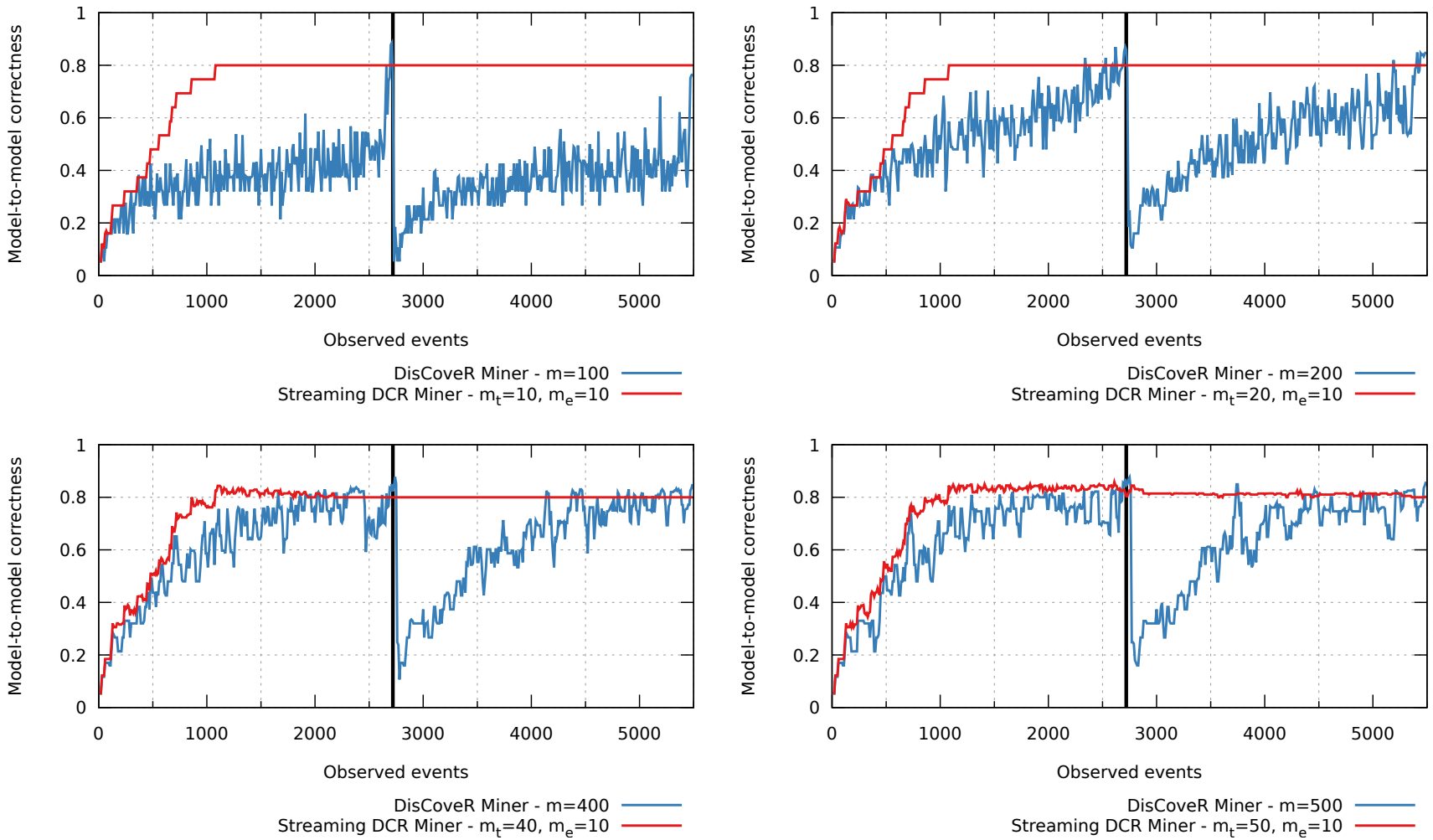
Concept drifts generate changes of the Resulting process model

Streaming DCR Miner



- The intuition is to
- First, identify an extended Direct-Follows graph from the stream
- Then, identify DCR patterns from the DFG

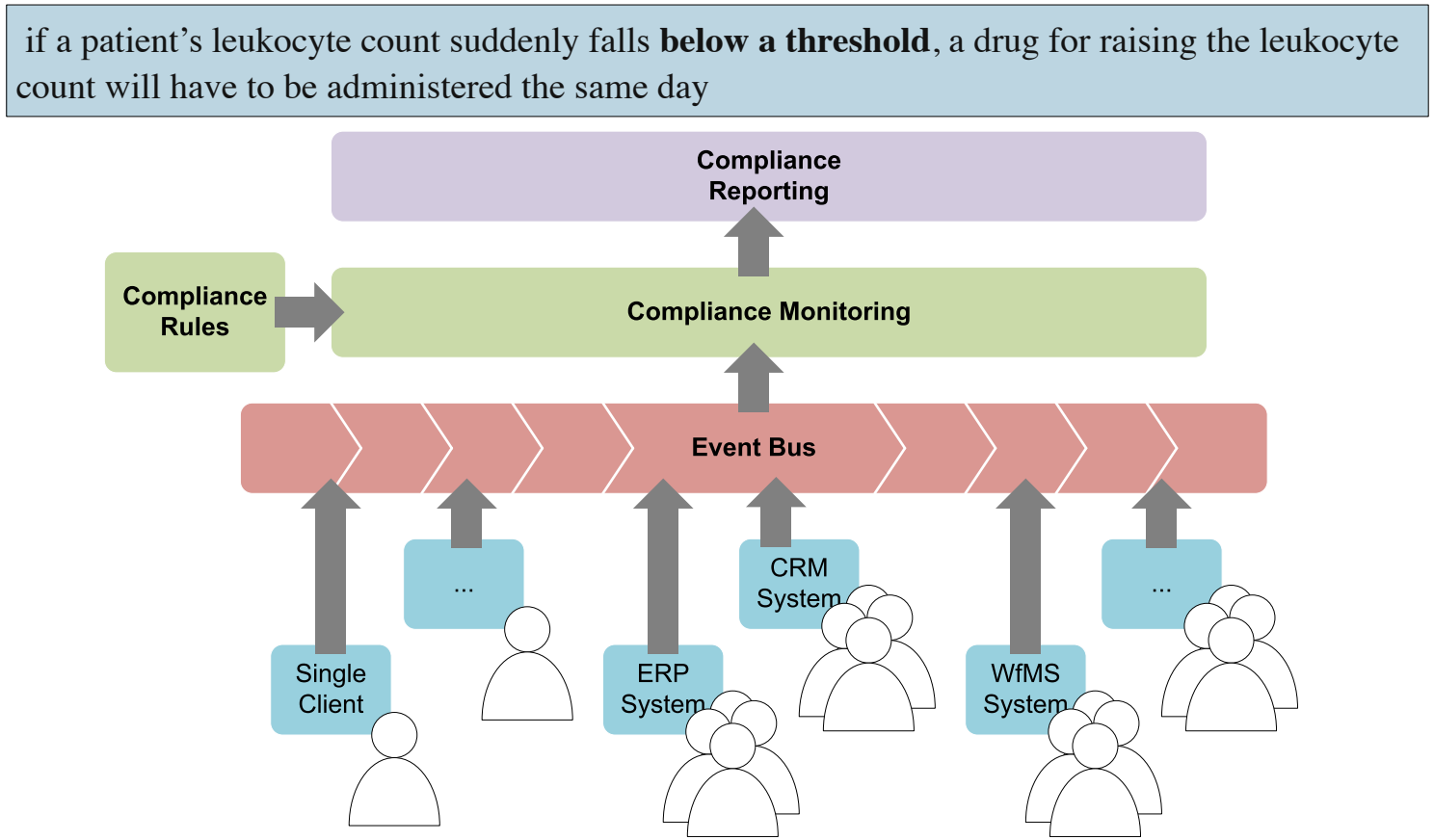
Drift Sensitivity



- The use case for online discovery is that of identifying change. Given that is it an approximation of the full log, some loss in precision is expected, yet, it should be able to remain accurate when changes in the observations occur

Another use case: Runtime compliance

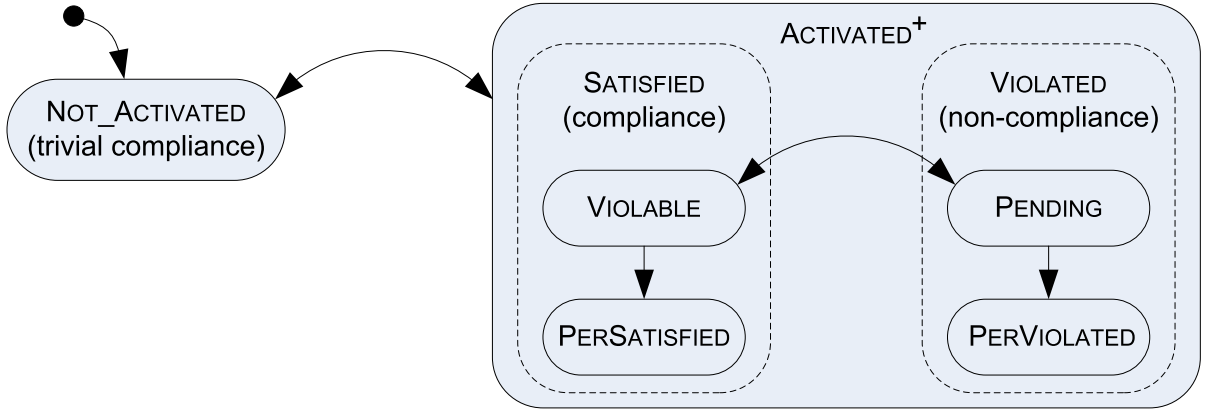
- Not always feasible to check all the constraints at compile time:



[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- A **monitor** keeps track of the state of each compliance rule:
- It does not apply to the running process, or **Not_Activated.**
- It is **Activated** for the running process, and it can be:
  - **Satisfied**:
    - But it is still **violable**
    - Or permanently satisfied.
  - **Violated**:
    - But healable (**pending**)
    - Or permanently violated.



[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- State(c1) =

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
⋮					
77	**2/7/2013**	**15:43**	**start**	**234**	**SolvencyCheck (Mrs. Brown)**
78	2/7/2013	15:43	read	234	customer = Mr.Smith
79	2/7/2013	15:54	write	234	rating= high
80	**2/7/2013**	**15:55**	**end**	**234**	***SolvencyCheck***
⋮					
91	**2/7/2013**	**18:13**	**start**	**453**	**Approval (Mr. Muller)**
92	2/7/2013	18:14	read	453	customer = Mr.Smith
93	2/7/2013	18:14	read	453	rating = high
94	2/7/2013	18:17	write	453	result= granted
95	**2/7/2013**	**18:18**	**end**	**453**	***Approval***
⋮					

Compliance rules

c_1 When a ***request item*** with an amount ***greater than 10,000*** is ***received from an agent***, the request must not be approved unless the solvency of the respective customer is checked. The latter task must be started ***at maximum three days*** after the receipt. Furthermore, task ***approval*** and task ***solvency check*** must be performed ***by different staff members***.

c_2 After approval of a ***request item***, the agent must be informed about the result ***within one day***.

c_3 After starting the production related to a particular ***order*** the latter may only be changed ***by the head of production***.

State(c2) = ?

State(c3) = ?

[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- State(c1) =

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
⋮					
77	**2/7/2013**	**15:43**	**start**	**234**	**SolvencyCheck (Mrs. Brown)**
78	2/7/2013	15:43	read	234	customer = Mr.Smith
79	2/7/2013	15:54	write	234	rating= high
80	**2/7/2013**	**15:55**	**end**	**234**	***SolvencyCheck***
⋮					
91	**2/7/2013**	**18:13**	**start**	**453**	**Approval (Mr. Muller)**
92	2/7/2013	18:14	read	453	customer = Mr.Smith
93	2/7/2013	18:14	read	453	rating = high
94	2/7/2013	18:17	write	453	result= granted
95	**2/7/2013**	**18:18**	**end**	**453**	***Approval***
⋮					

Compliance rules

c_1 When a ***request item*** with an amount ***greater than 10,000*** is ***received from an agent***, the request must not be approved unless the solvency of the respective customer is checked. The latter task must be started ***at maximum three days*** after the receipt. Furthermore, task ***approval*** and task ***solvency check*** must be performed ***by different staff members***.

c_2 After approval of a ***request item***, the agent must be informed about the result ***within one day***.

c_3 After starting the production related to a particular ***order*** the latter may only be changed ***by the head of production***.

Violable

State(c2) = ?

State(c3) = ?

[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- State(c1) =

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
⋮					
77	**2/7/2013**	**15:43**	**start**	**234**	**SolvencyCheck (Mrs. Brown)**
78	2/7/2013	15:43	read	234	customer = Mr.Smith
79	2/7/2013	15:54	write	234	rating= high
80	**2/7/2013**	**15:55**	**end**	**234**	***SolvencyCheck***
⋮					
91	**2/7/2013**	**18:13**	**start**	**453**	**Approval (Mr. Muller)**
92	2/7/2013	18:14	read	453	customer = Mr.Smith
93	2/7/2013	18:14	read	453	rating = high
94	2/7/2013	18:17	write	453	result= granted
95	**2/7/2013**	**18:18**	**end**	**453**	***Approval***
⋮					

Compliance rules

c_1 When a ***request item*** with an amount ***greater than 10,000*** is ***received from an agent***, the request must not be approved unless the solvency of the respective customer is checked. The latter task must be started ***at maximum three days*** after the receipt. Furthermore, task ***approval*** and task ***solvency check*** must be performed ***by different staff members***.

c_2 After approval of a ***request item***, the agent must be informed about the result ***within one day***.

c_3 After starting the production related to a particular ***order*** the latter may only be changed ***by the head of production***.

Violable

State(c2) = ?

State(c3) = ?

[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- State(c1) =

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
⋮					
77	**2/7/2013**	**15:43**	**start**	**234**	**SolvencyCheck (Mrs. Brown)**
78	2/7/2013	15:43	read	234	customer = Mr.Smith
79	2/7/2013	15:54	write	234	rating= high
80	**2/7/2013**	**15:55**	**end**	**234**	***SolvencyCheck***
⋮					
91	**2/7/2013**	**18:13**	**start**	**453**	**Approval (Mr. Muller)**
92	2/7/2013	18:14	read	453	customer = Mr.Smith
93	2/7/2013	18:14	read	453	rating = high
94	2/7/2013	18:17	write	453	result= granted
95	**2/7/2013**	**18:18**	**end**	**453**	***Approval***
⋮					

Compliance rules

c_1 When a ***request item*** with an amount ***greater than 10,000*** is ***received from an agent***, the request must not be approved unless the solvency of the respective customer is checked. The latter task must be started ***at maximum three days*** after the receipt. Furthermore, task ***approval*** and task ***solvency check*** must be performed ***by different staff members***.

c_2 After approval of a ***request item***, the agent must be informed about the result ***within one day***.

c_3 After starting the production related to a particular ***order*** the latter may only be changed ***by the head of production***.

PerSatisfied

State(c2) = ?

State(c3) = ?

[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- State(c1) =

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
55	**1/7/2013**	**18:03**	**receive**	**592**	**Request**
56	1/7/2013	18:03	write	592	customer = Mrs.John
57	1/7/2013	18:03	write	592	amount = 27.000€
58	**1/7/2013**	**18:03**	**end**	**592**	***Request***
⋮					
⋮					
96	**2/7/2013**	**18:19**	**start**	**642**	**Approval (Mrs. Brown)**
97	2/7/2013	18:20	read	642	customer = Mrs.John
98	2/7/2013	18:23	write	642	result = granted
99	**2/7/2013**	**18:23**	**end**	**642**	***Approval***
⋮					

Compliance rules

c_1 When a ***request item*** with an amount ***greater than 10,000*** is ***received from an agent***, the request must not be approved unless the solvency of the respective customer is checked. The latter task must be started ***at maximum three days*** after the receipt. Furthermore, task ***approval*** and task ***solvency check*** must be performed ***by different staff members***.

c_2 After approval of a ***request item***, the agent must be informed about the result ***within one day***.

c_3 After starting the production related to a particular ***order*** the latter may only be changed ***by the head of production***.

Violable

State(c2) = ?

State(c3) = ?

[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- State(c1) =

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
55	**1/7/2013**	**18:03**	**receive**	**592**	**Request**
56	1/7/2013	18:03	write	592	customer = Mrs.John
57	1/7/2013	18:03	write	592	amount = 27.000€
58	**1/7/2013**	**18:03**	**end**	**592**	***Request***
⋮					
77	**2/7/2013**	**15:43**	**start**	**234**	**SolvencyCheck (Mrs. Brown)**
78	2/7/2013	15:43	read	234	customer = Mr.Smith
79	2/7/2013	15:54	write	234	rating= high
80	**2/7/2013**	**15:55**	**end**	**234**	***SolvencyCheck***
⋮					
96	**2/7/2013**	**18:19**	**start**	**642**	**Approval (Mrs. Brown)**
97	2/7/2013	18:20	read	642	customer = Mrs.John
98	2/7/2013	18:23	write	642	result = granted
99	**2/7/2013**	**18:23**	**end**	**642**	***Approval***
⋮					

Compliance rules

c_1 When a ***request item*** with an amount ***greater than 10,000*** is ***received from an agent***, the request must not be approved unless the solvency of the respective customer is checked. The latter task must be started ***at maximum three days*** after the receipt. Furthermore, task ***approval*** and task ***solvency check*** must be performed ***by different staff members***.

c_2 After approval of a ***request item***, the agent must be informed about the result ***within one day***.

c_3 After starting the production related to a particular ***order*** the latter may only be changed ***by the head of production***.

Violable
PerViolated

State(c2) = ?

State(c3) = ?

[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

Compliance monitoring

- State(c1) =

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
55	**1/7/2013**	**18:03**	**receive**	**592**	**Request**
56	1/7/2013	18:03	write	592	customer = Mrs.John
57	1/7/2013	18:03	write	592	amount = 27.000€
58	**1/7/2013**	**18:03**	**end**	**592**	***Request***
⋮					
77	**2/7/2013**	**15:43**	**start**	**234**	**SolvencyCheck (Mrs. Brown)**
78	2/7/2013	15:43	read	234	customer = Mr.Smith
79	2/7/2013	15:54	write	234	rating= high
80	**2/7/2013**	**15:55**	**end**	**234**	***SolvencyCheck***
⋮					
96	**2/7/2013**	**18:19**	**start**	**642**	**Approval (Mrs. Brown)**
97	2/7/2013	18:20	read	642	customer = Mrs.John
98	2/7/2013	18:23	write	642	result = granted
99	**2/7/2013**	**18:23**	**end**	**642**	***Approval***
⋮					

Compliance rules

c_1 When a ***request item*** with an amount ***greater than 10,000*** is ***received from an agent***, the request must not be approved unless the solvency of the respective customer is checked. The latter task must be started ***at maximum three days*** after the receipt. Furthermore, task ***approval*** and task ***solvency check*** must be performed ***by different staff members***.

c_2 After approval of a ***request item***, the agent must be informed about the result ***within one day***.

c_3 After starting the production related to a particular ***order*** the latter may only be changed ***by the head of production***.

PerViolated

State(c2) = ?

State(c3) = ?

[1]
D. Knuplesch, M. Reichert, and A. Kumar, "Visually Monitoring Multiple Perspectives of Business Process Compliance," in *Business Process Management*, 2015, pp. 263–279.

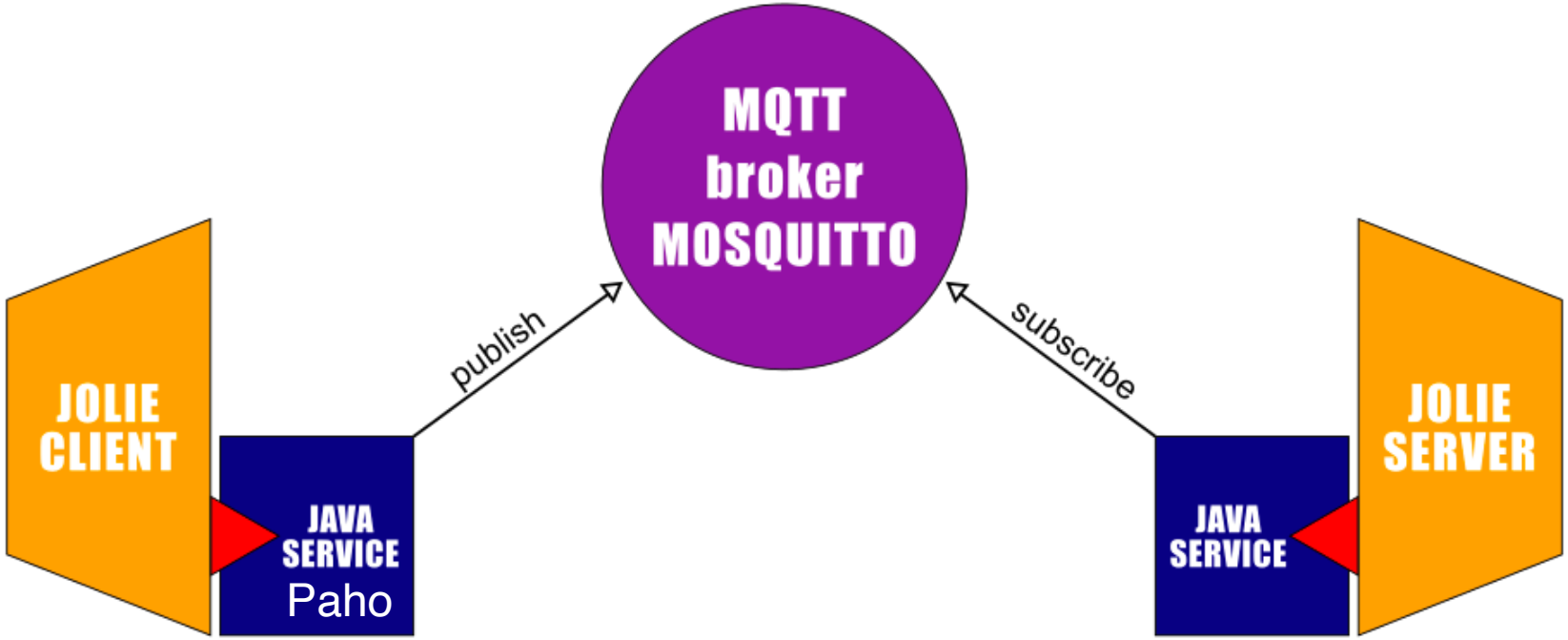
BREAKOUT ROOMS

- 1. Model compliance rules c_1, c_2 & c_3 in a DCR-graph (use additional activities to model data/time checks).

- 2. Calculate what is the final state of rules c_2 & c_3 after execution of event 99.You can use the simulator for this

#	date	time	type	id	details
⋮					
37	**1/7/2013**	**15:27**	**receive**	**124**	**Request**
38	1/7/2013	15:27	write	124	customer = Mr.Smith
39	1/7/2013	15:27	write	124	amount = 15.000€
40	**1/7/2013**	**15:27**	**end**	**124**	***Request***
⋮					
55	**1/7/2013**	**18:03**	**receive**	**592**	**Request**
56	1/7/2013	18:03	write	592	customer = Mrs.John
57	1/7/2013	18:03	write	592	amount = 27.000€
58	**1/7/2013**	**18:03**	**end**	**592**	***Request***
⋮					
77	**2/7/2013**	**15:43**	**start**	**234**	**SolvencyCheck (Mrs. Brown)**
78	2/7/2013	15:43	read	234	customer = Mr.Smith
79	2/7/2013	15:54	write	234	rating= high
80	**2/7/2013**	**15:55**	**end**	**234**	***SolvencyCheck***
⋮					
91	**2/7/2013**	**18:13**	**start**	**453**	**Approval (Mr. Muller)**
92	2/7/2013	18:14	read	453	customer = Mr.Smith
93	2/7/2013	18:14	read	453	rating = high
94	2/7/2013	18:17	write	453	result= granted
95	**2/7/2013**	**18:18**	**end**	**453**	***Approval***
96	**2/7/2013**	**18:19**	**start**	**642**	**Approval (Mrs. Brown)**
97	2/7/2013	18:20	read	642	customer = Mrs.John
98	2/7/2013	18:23	write	642	result = granted
99	**2/7/2013**	**18:23**	**end**	**642**	***Approval***
⋮					

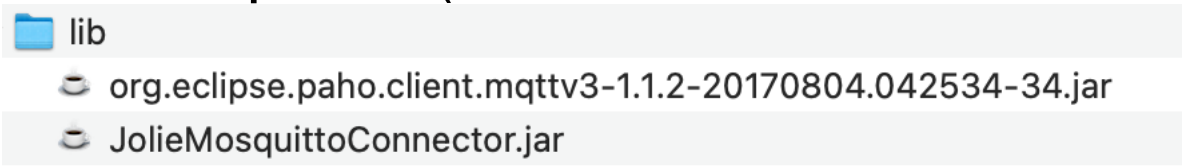
Programming an event subscriber



- Mosquitto is a lightweight MQTT broker
- Eclipse Paho provides open source client-side implementations of MQTT
- The Jolie-mosquitto connector provides a connector that allows programmers to choose MQTT as a data transmission protocol in Jolie

Requirements

- Files required (See in Absalon under /Files/Code/Lecture7/lib)



- JolieMosquittoConnector contains both the connector and the interfaces necessary for the Jolie service to communicate qith the Mosquitto broker. Here is the official source
- org.eclipse.paho.client contains the dependency to the Paho library that Java services use to create publishers-subscribers. Here is the official source

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

Simple Operation Data Exchange Prot.

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

Simple Operation Data Exchange Prot.
One way receiver interface

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

Simple Operation Data Exchange Prot.
One way receiver interface

Request-response interf.

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

Simple Operation Data Exchange Prot.
One way receiver interface

Request-response interf.

Embedding mosquitto jar connector

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

Simple Operation Data Exchange Prot.
One way receiver interface

Request-response interf.

Embedding mosquitto jar connector

Executes at service initialization

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

Simple Operation Data Exchange Prot.
One way receiver interface

Request-response interf.

Embedding mosquitto jar connector

Executes at service initialization

Implements MosquittoInterface

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Subscriber in Jolie

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"
include "json_utils.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService"
in Mosquitto
}

init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/DreyersFond/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}
```

Simple Operation Data Exchange Prot.
One way receiver interface

Request-response interf.

Embedding mosquitto jar connector

Executes at service initialization

Implements MosquittoInterface

Depends on execution modality

```
main {
    receive (request)
    println@Console("topic :
"+request.topic)()

getJsonValue@JsonUtils( request.
message )( response )
    println@Console("message :
"+response.event.Activity)()
}
```

Third Assignment (Part B)

UNIVERSITY OF COPENHAGEN

Part B

- In this part you will use Jolie to implement subscriber to a MQTT-XES broker in HiveMQ

- You can try to run your exercises with several logs (for all the broker url is broker.hivemq.com):

  - pmcep/REBS/#
  - pmcep/Hospital log/#
  - pmcep/Disco Example Log/#
  - pmcep/Hospital Billing - Event Log/#
  - pmcep/DreyersFond/#

Exercise B.1

Below is an implementation of a server process subscribing to the call center log (called Disco Example Log) published at the HIVEMQ broker.

```
include "mosquitto/interfaces/MosquittoInterface.iol"
include "console.iol"

execution {concurrent}

inputPort Server {
    Location: "local"
    Protocol: sodep
    Interfaces: MosquittoReceiverInteface
}

outputPort Mosquitto {
    Interfaces: MosquittoInterface
}

embedded {
    Java:
        "org.jolielang.connector.mosquitto.MosquittoConnectorJavaService" in Mosquitto
}
```

B.1 (continued)

```
init {
    request << {
        brokerURL = "tcp://broker.hivemq.com",
        subscribe << {
            topic = "pmcep/Disco Example Log/#"
        }
        // I can set all the options available from the Paho library
    }
    setMosquitto@Mosquitto (request)()
}

main {
    receive (request)
    println@Console("topic :     "+request.topic)()
    println@Console("message :   "+request.message)()
}
```

- using the jar files JolieMosquittoConnector.jar and org.eclipse.paho.client.mqttv3-1.1.2-20170804.042534-34.jar which should both be placed in a folder called lib in the same location as the subscriber source file.

Exercise B.1.1

- Run the subscriber. It should (after some warnings) receive events from the broker on the call center example with topics at the format pmcep/Disco Example Log/CaseX/Activity, where X is a number denoting the ID of the case and Activity is the activity, e.g. Inbound Call.

- 2.1 Run the subscriber. It should (after some warnings) receive events from the broker on the call center example with topics at the format pmcep/Disco Example Log/CaseX/Activity, where X is a number denoting the ID of the case and Activity is the activity, e.g. Inbound Call.
- 2.2 Run the subscriber in two shells. Explain what happens and how this is different than for a message queuing system and actors.
- 2.3 Implement two new subscribers that uses the topic filters to only listens to respectively Inbound Email and the Inbound call activities as the Email client and the call center example above.
- 2.4 Try to implement a subscriber that counts how many time each kind of activity happens.

Formalities

- Deadline: 18 January 11:00 am
- Includes Part A (last week) and Part B (this week)
- Include a PDF with the report and if there is any functional code, include sources and/or links to repositories.
  - If there are any additional packages, provide a way to automate installation (maven/ant/build scripts)