Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

Indirect Communication (Messaging)

Chun-Feng Liao

廖峻鋒

Dept. of Computer Science

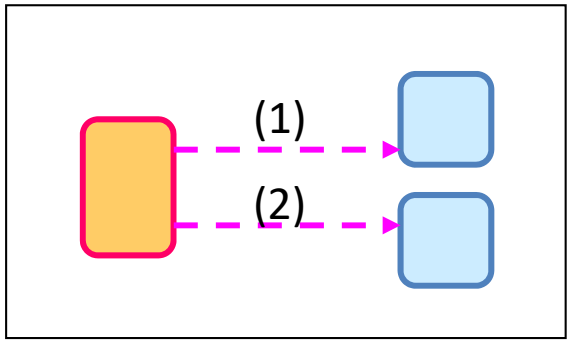
National Chengchi University

Basic Remoting Styles

- Two major styles
  - Direct communication (RPC)
  - Indirect communication (Messaging)

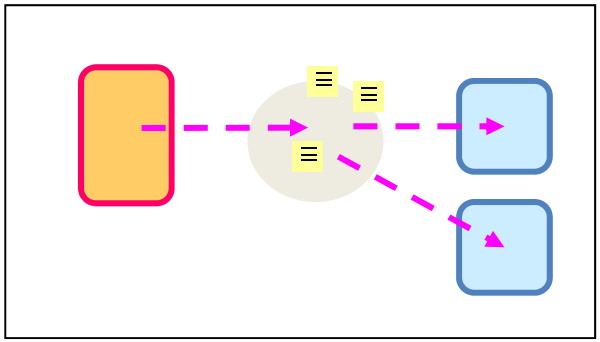
decouple

呼叫的時候都要存在(時間綁定)
需要知道彼此 IP (空間綁定)
還要按照次序



簡單、直覺

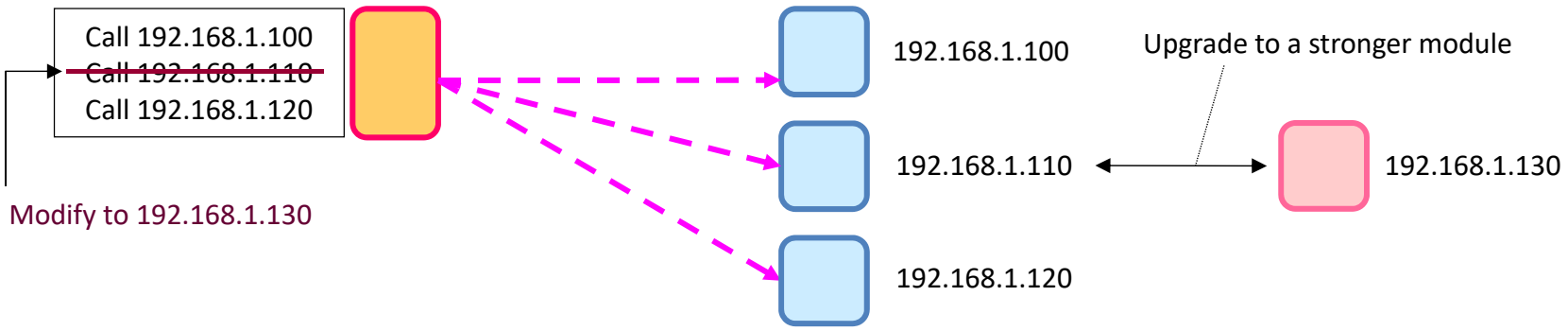
只要知道中間的IP
兩端不需知道彼此 IP(空間解耦)
訊息可暫存中間，兩端可不用同時存在(時間解耦)



中間的 broker 就會變得不彈性

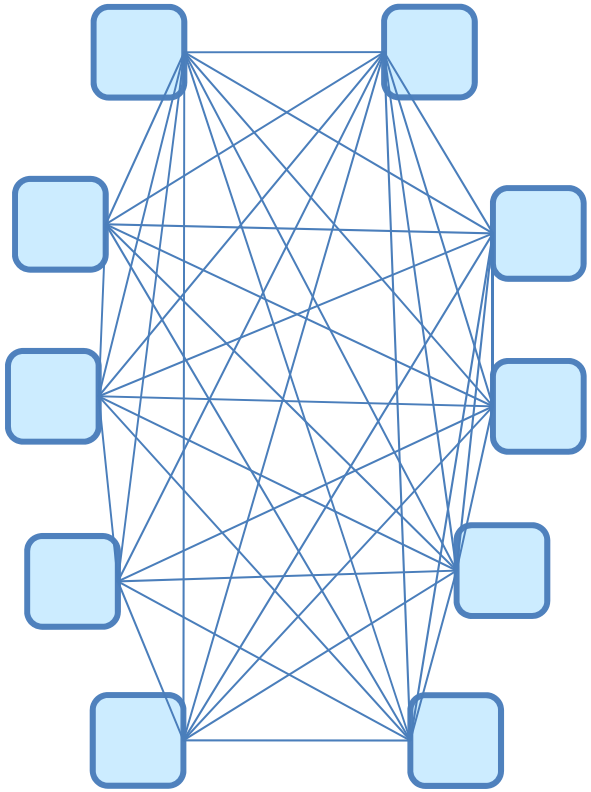
Direct Communication

- Benefits
  - Simplified application development
- Drawbacks
  - Tightly coupled on space (reference) and time (synchronous)
  - Fragile and hard to recover



Integrating Systems One-by-One

(n-1)!

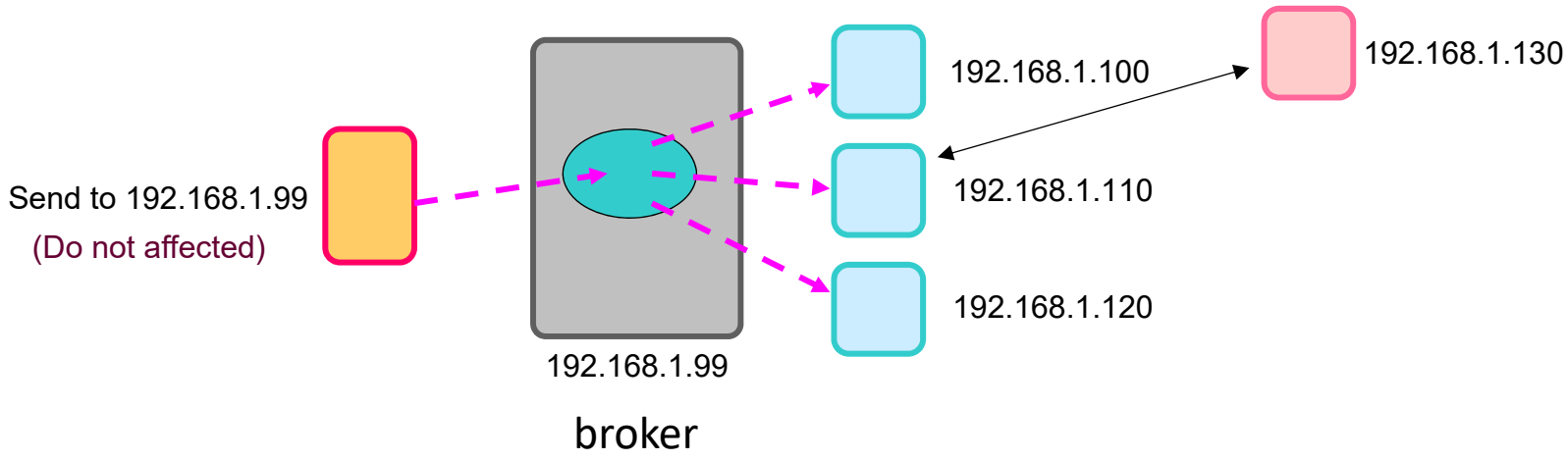


Indirect Communication

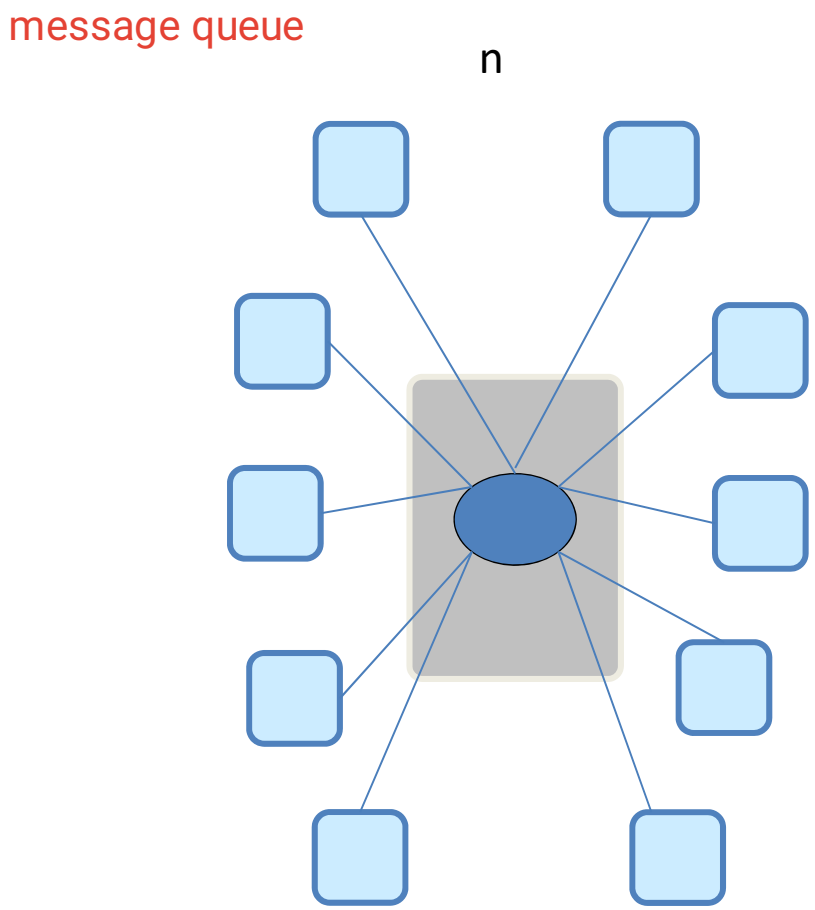
- Benefits
  - Decoupling in both space and time

空間時間解耦
為什麼空間可以解耦 => 不需直接經對方的 IP
為什麼時間可以解耦 => 不需要同時存在

- Drawbacks
  - Single point of failure can be alleviated by clustering
  - Address binding of the broker can be alleviated by broker discovery



Integrating Systems by Using MOM

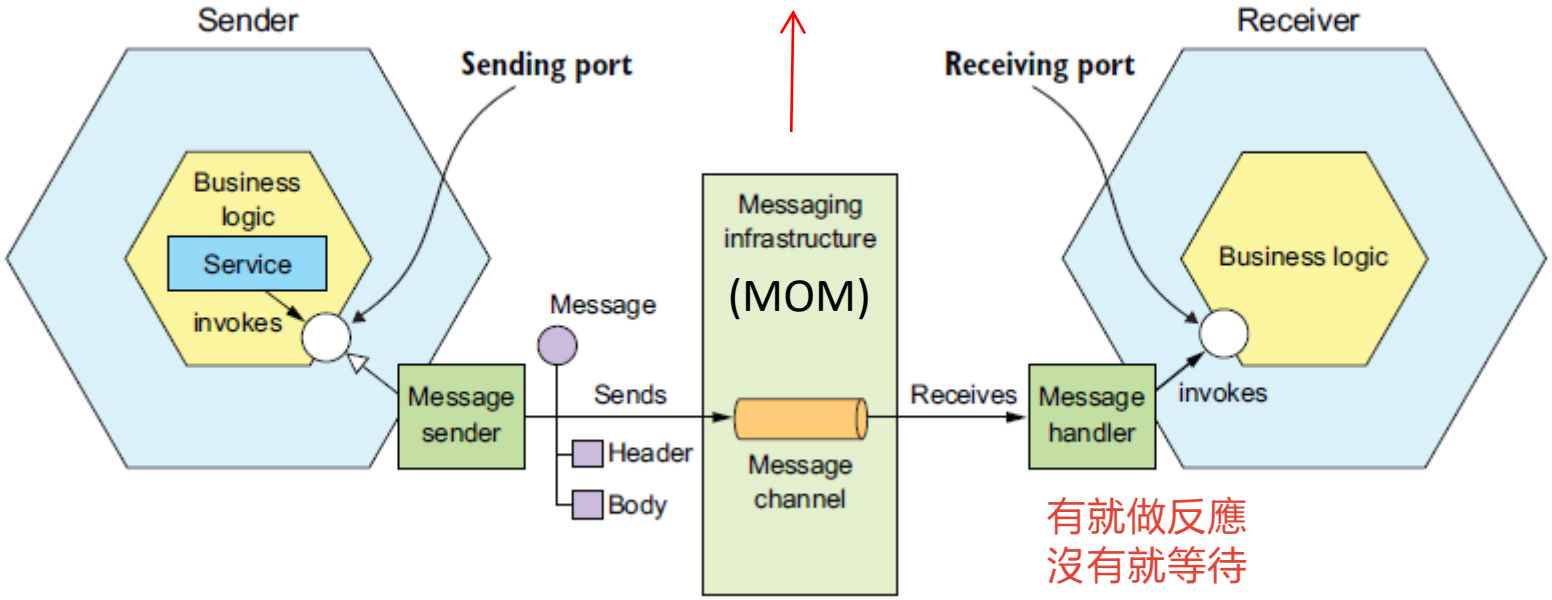


Architectural Overview

Queue => 適合工作分配，有人先做，別人就不能再做
Topic => 每個訂閱者都要做

傳送時要告訴 middleware要做的是哪個

message oriented middleware



有就做反應
沒有就等待

publisher (message) broker subscriber

根據訊息內容的 header 繞送至目標

broker 會有很多 Topic 和 Queue
Publisher 會發布到 Topic/Queue
Subcriber 會去訂閱 Topic/Queue

Message Semantics

MOM 的結構都會有 Header & Body

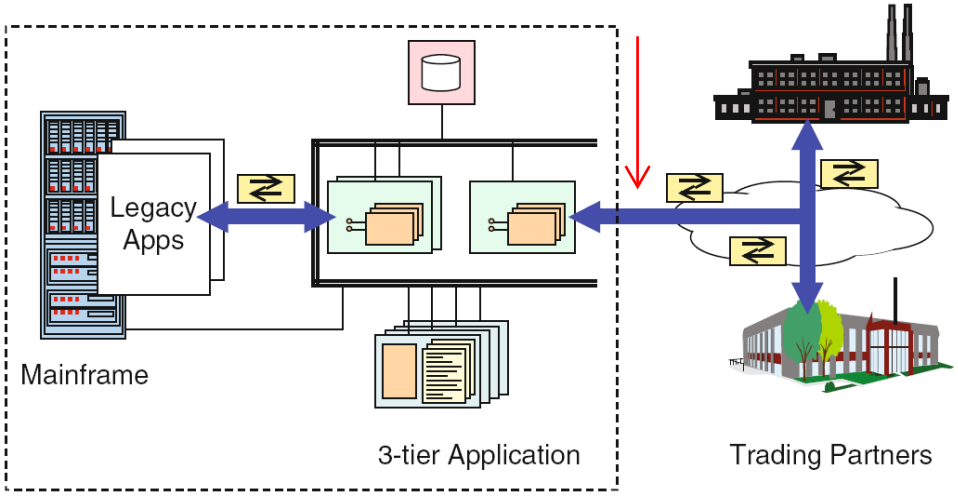
- Header **key-value pair，描述 Body 承載的東西**
  - Name-value pairs to annotate the message
  - Ex: Metadata, message id, replying channel name
- Body (Payload) 分三類
  - Document
    - Ex: return data
  - Command
    - Ex: to simulate RPC
  - Event 有什麼事件發生
    - Indicating that something notable has occurred
    - Domain event: the state change of a domain object

Message-Oriented Middleware (MOM)

金融常用

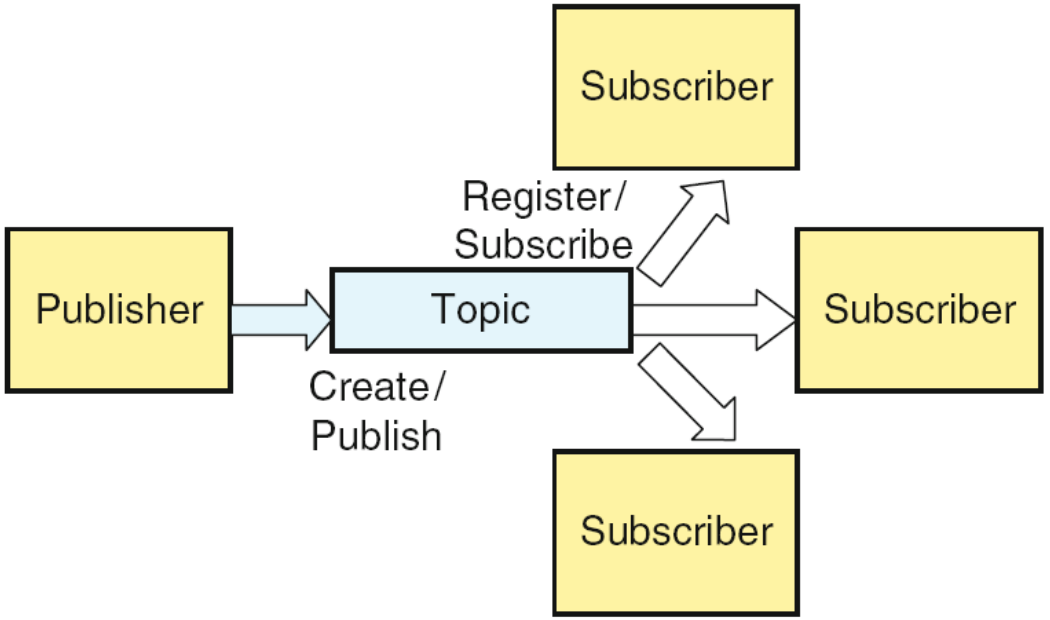
- Key technology for building large-scale enterprise systems
  - It is the glue that binds together other independent and autonomous applications and turns them into a single, integrated system
  - Achieved by placing a queue/hub between senders and receivers, providing a level of indirection during communications

這裡會有 MOM
做資料的交換



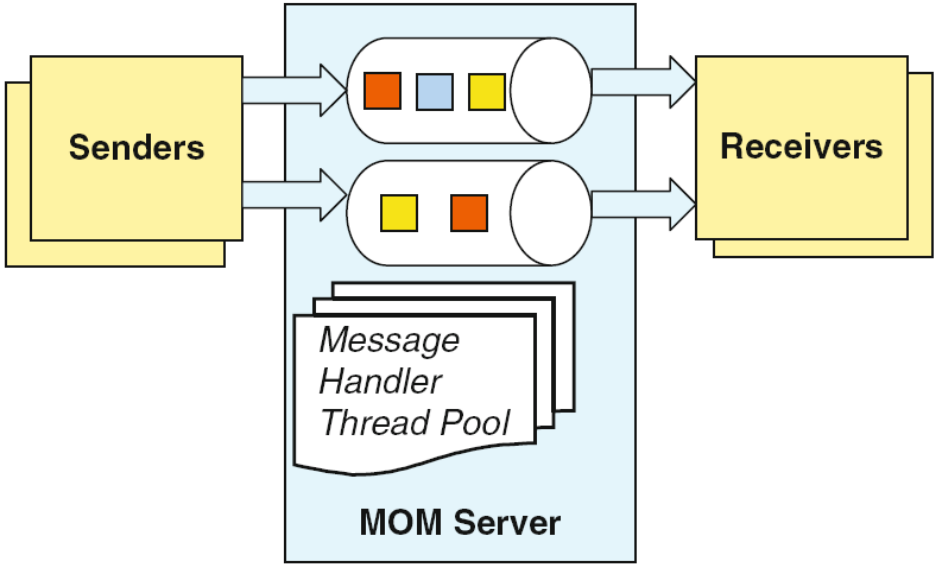
Publish-Subscribe

- Publishers
  - Send a message to a named topic
- Subscribers
  - listen for messages that are sent to topics that interest them



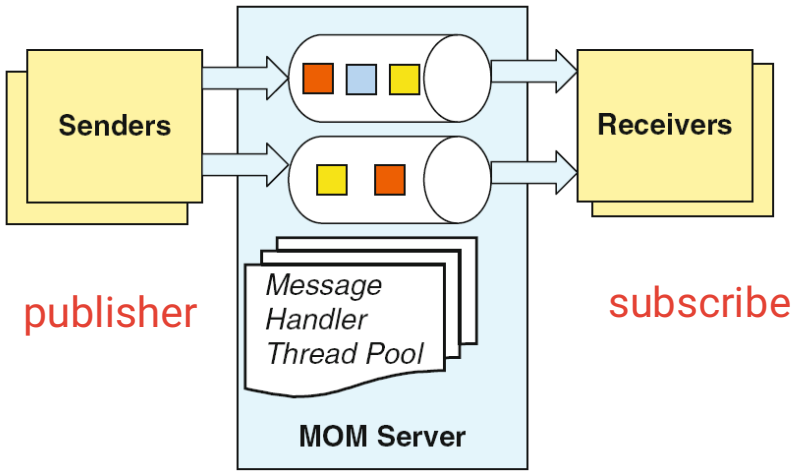
MOM Server (Message Brokers)

- Mechanisms
  - Create and manage multiple messages queues (topics)
  - Handle multiple messages being sent from queues simultaneously using threads organized in a thread pool

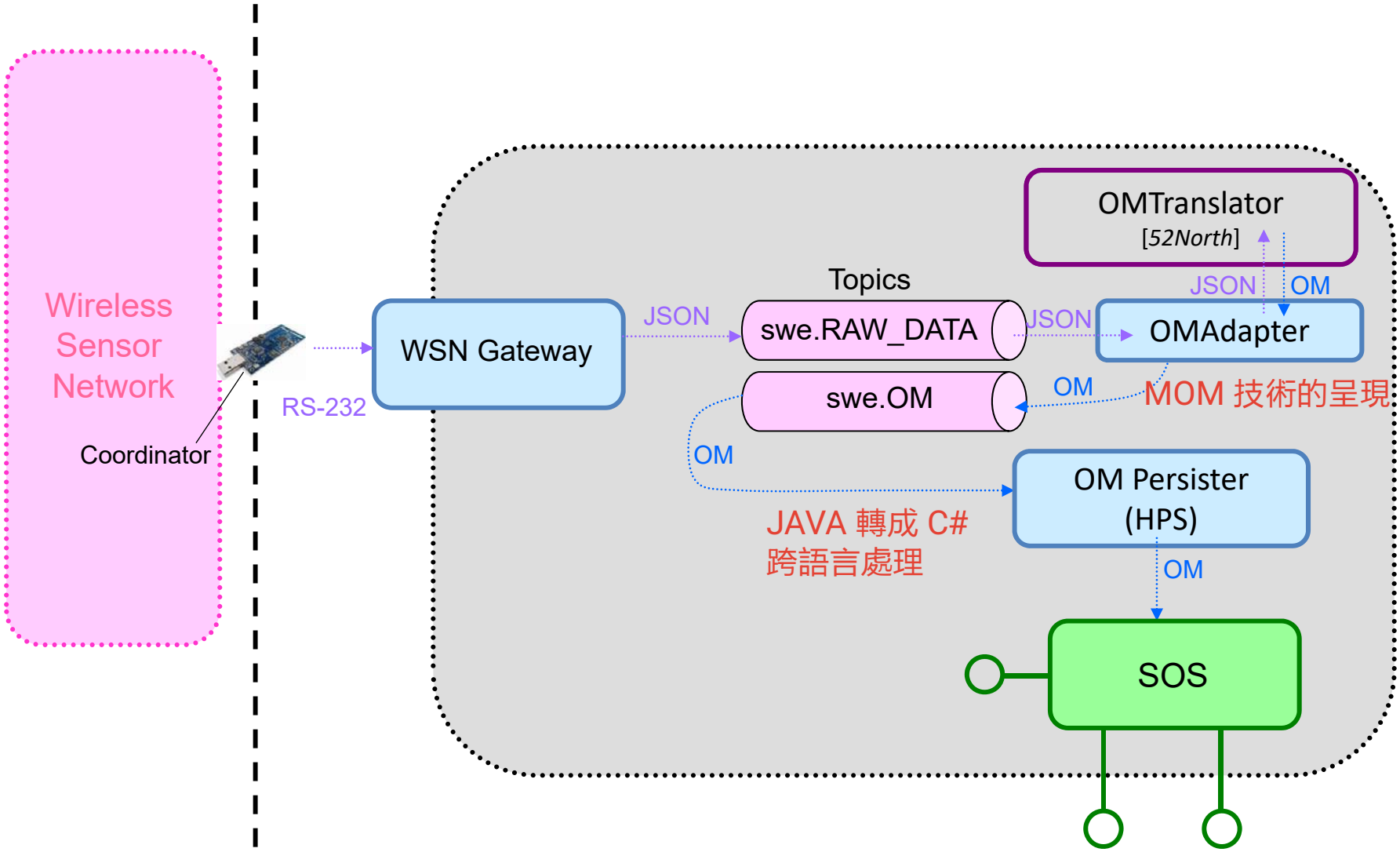


MOM Server (Message Brokers)

- Message transmitting
  - Accept/Ack messages from the senders
  - Place the messages at the end of the queue (topic)
  - Hold messages for an extended period of time



Case:
Sensor Observation Service (OGC SWE)

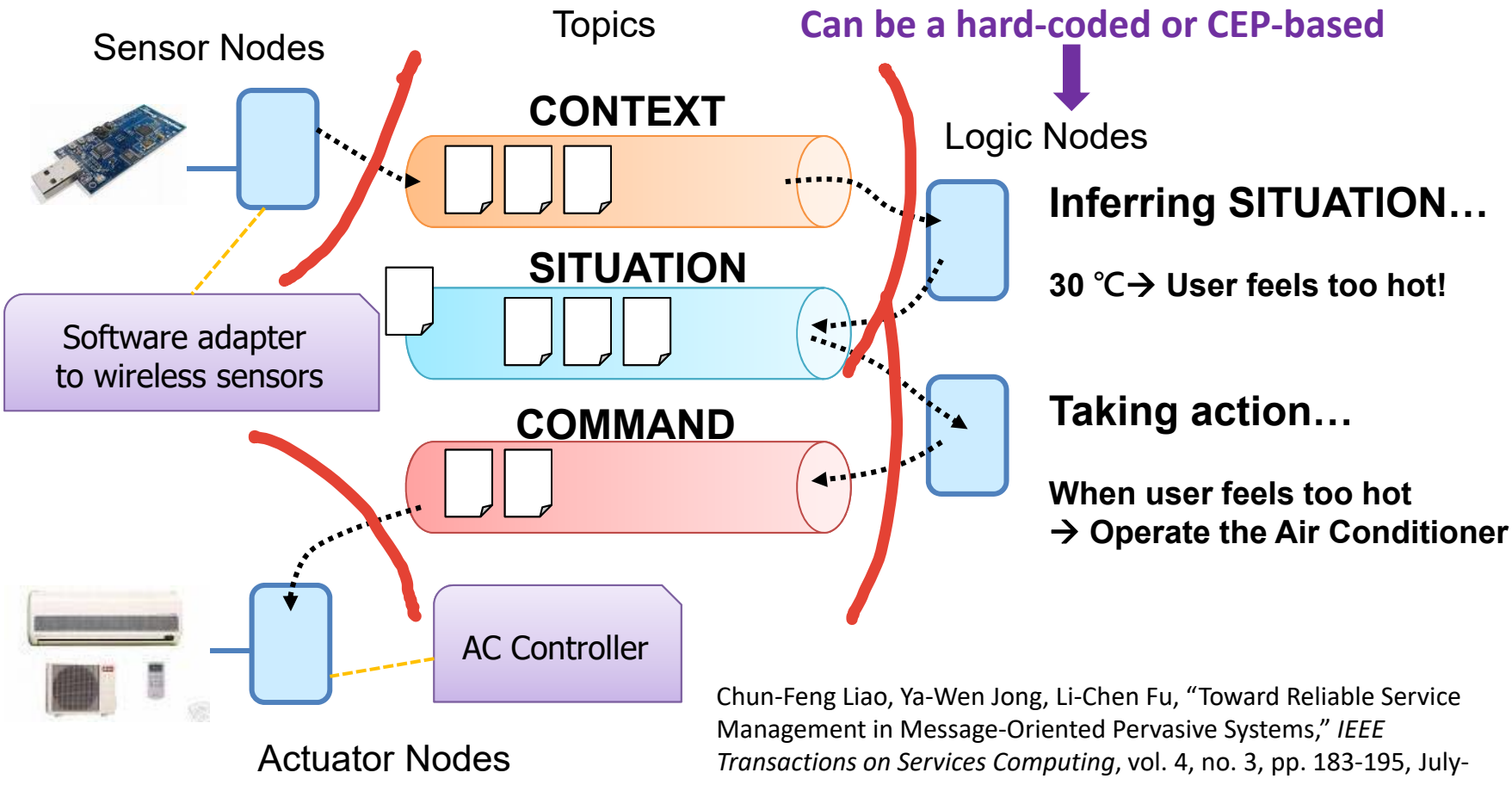


Case: MQTT

- Message Queuing Telemetry Transport
  - Proposed by IBM and Eurotech 後台的概念轉到物聯網
  - Based on TCP/IP (MQTT-SN can use UDP)
- Compact
  - Designed for IoT
    - The most popular MOM for IoT
    - Low bandwidth and computing capability requirement
- Know uses
  - Facebook Messenger, Amazon IoT, OGC, Azure IoT Hub

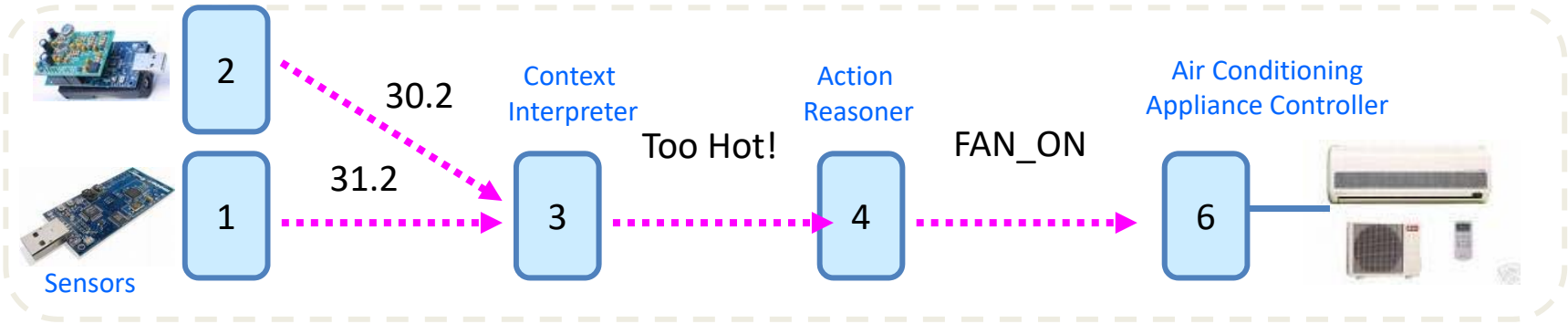
Case: 物聯網系統整合

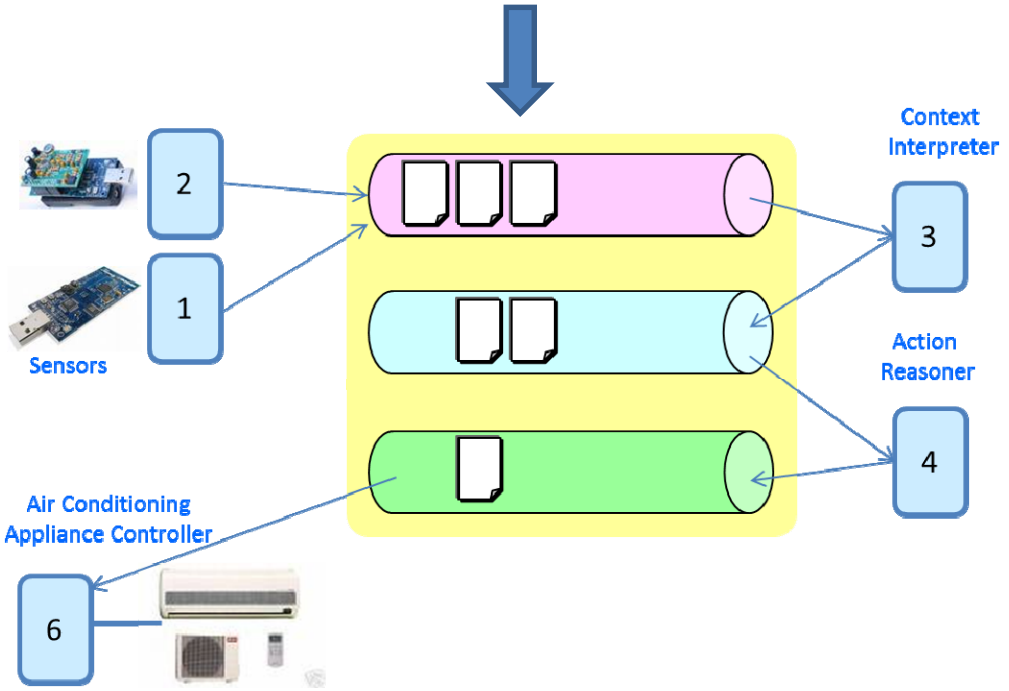
每個區塊都可以直接測試



Chun-Feng Liao, Ya-Wen Jong, Li-Chen Fu, "Toward Reliable Service Management in Message-Oriented Pervasive Systems," *IEEE Transactions on Services Computing*, vol. 4, no. 3, pp. 183-195, July-Sept. 2011.

Case: MQTT 物聯網系統整合





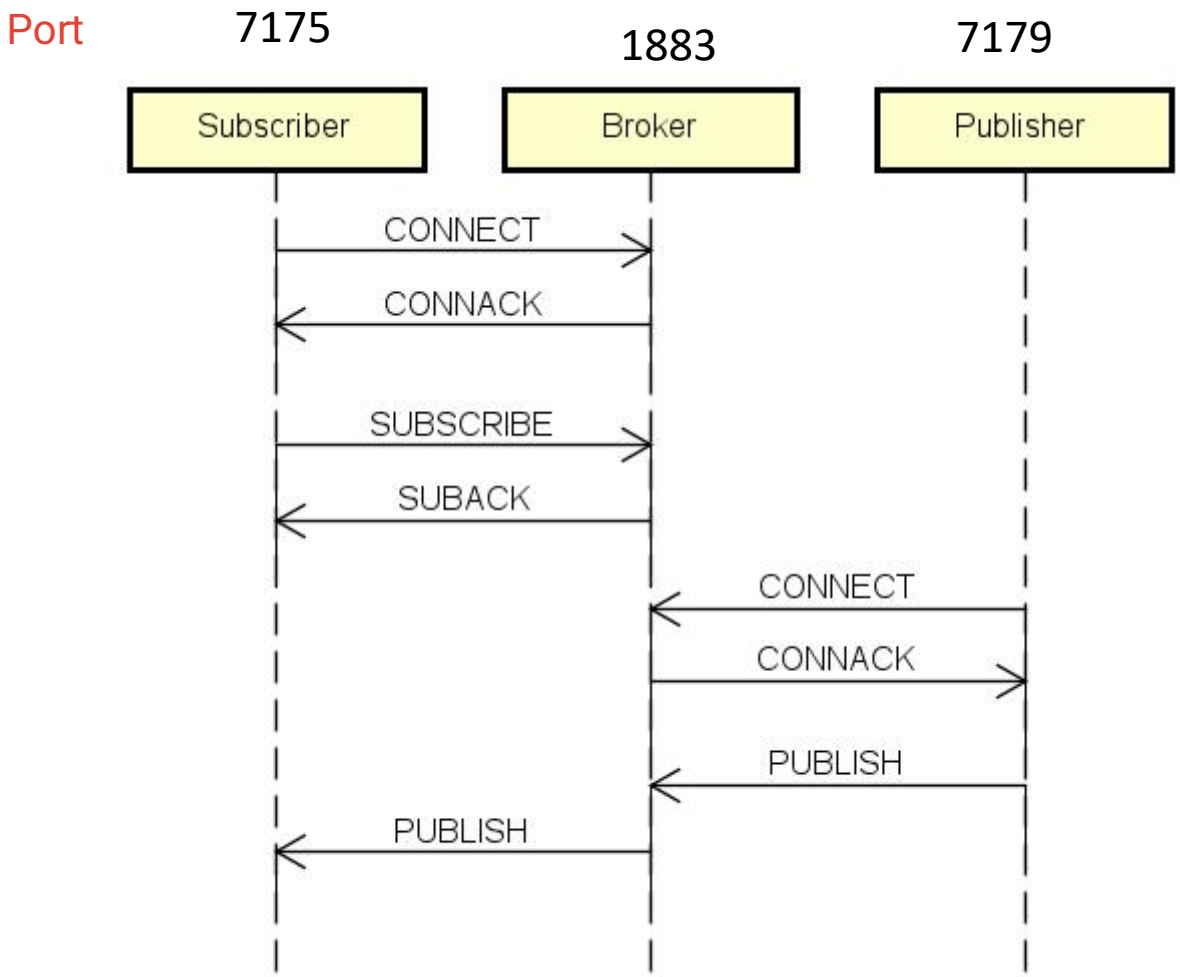
MQTT Basic Protocol Elements

- 連線管理
  - CONNECT/CONNACK
  - DISCONNECT
- 訂閱
  - SUBSCRIBE/ SUBACK
- 發送
  - QoS0: PUBLISH
  - QoS1: PUBACK
  - QoS2: PUBREC/PUBREL/PUBCOMP

重要的訊息比較緊急，用較多的封包發送，也較耗時間
不重要的反之

根據需求指定，越高級代很越緊急

訊息傳送基本流程

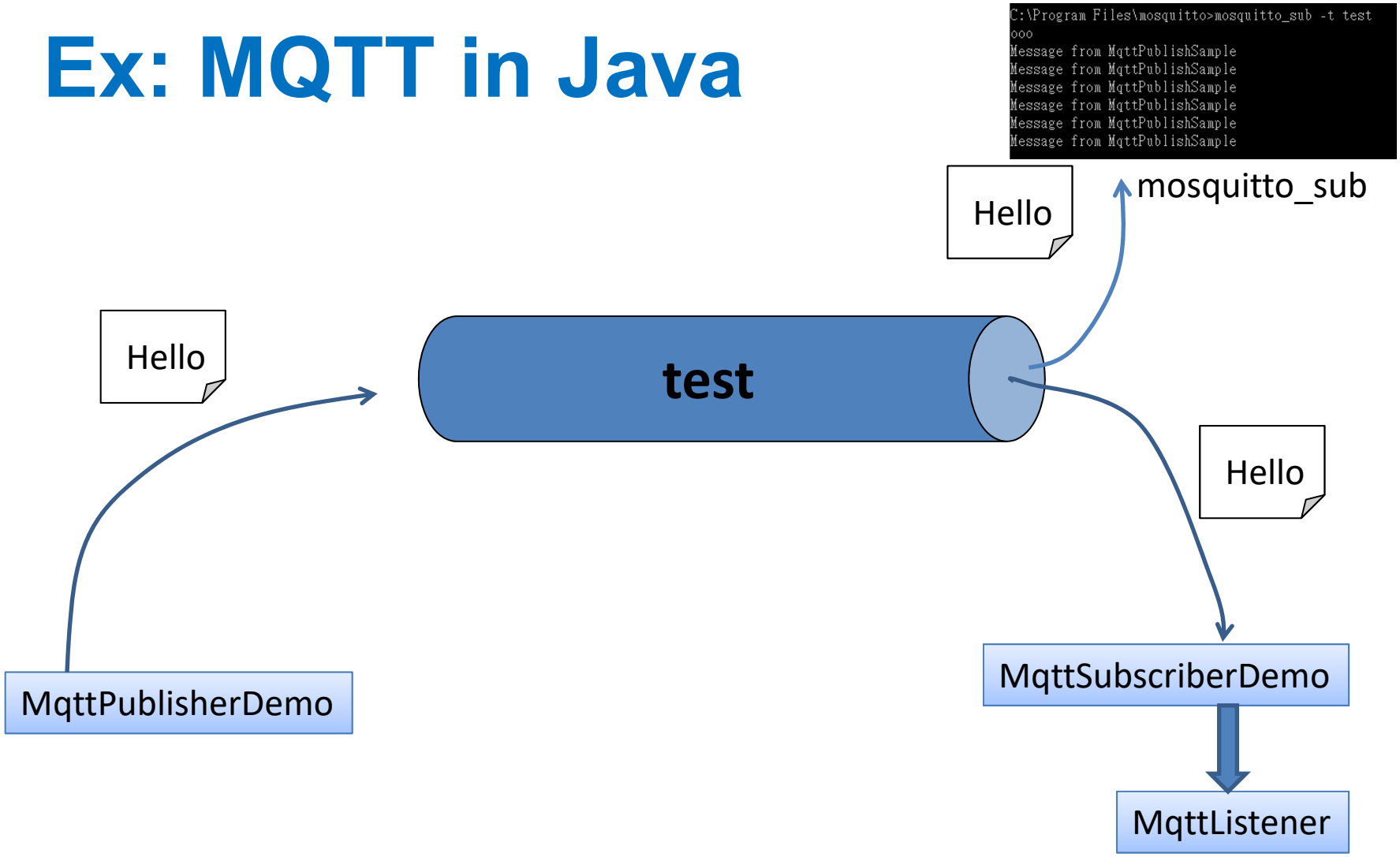

Port
7175
1883
7179
Subscriber
Broker
Publisher
CONNECT
CONNACK
SUBSCRIBE
SUBACK
CONNECT
CONNACK
PUBLISH
PUBLISH

封包觀察

- See wireshark

封包解讀 https://github.com/mqttjs/mqtt-packet

Ex: MQTT in Java



Java MQTT Publisher

- Connect

```
String clientId = "cfliao-pub";
MemoryPersistence persistence = new MemoryPersistence();
MqttClient sampleClient = new MqttClient("tcp://localhost:1883", clientId, persistence);
sampleClient.connect(connOpts);
```

- Publish

```
String content = “My Message";
String topic = “test”;
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(0);
sampleClient.publish(topic, message);
```

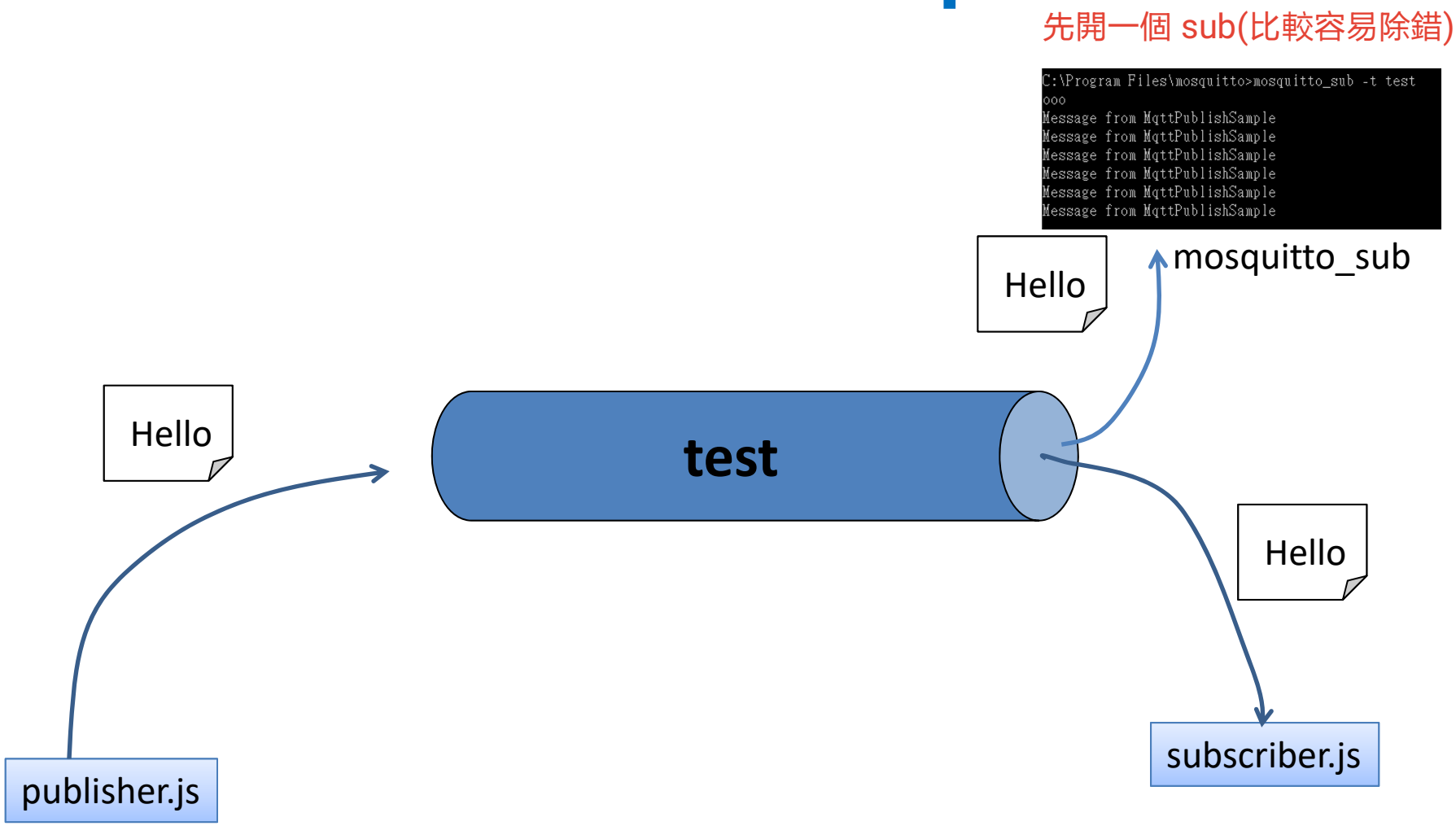
Java MQTT Subscriber

```
String clientId = "cfliao-sub";
MemoryPersistence persistence = new MemoryPersistence();
MqttAsyncClient sampleClient =
                    new MqttAsyncClient("tcp://localhost:1883", clientId, persistence);
sampleClient.setCallback(new MqttListener());
IMqttToken conToken = sampleClient.connect();
conToken.waitForCompletion();

String topicName = "test";
IMqttToken subToken = sampleClient.subscribe(topicName, 0);
subToken.waitForCompletion();
```

```
public class MqttListener implements MqttCallback {
        …
   @Override
   public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
        System.out.println(mqttMessage.toString());
   }

}
```

Ex: MQTT in JavaScript

先開一個 sub(比較容易除錯)



JavaScript (Node.js)

publisher

```
const mqtt = require('mqtt');
const client = mqtt.connect();
client.on('connect', function ()  {
   client.publish('MY_TOPIC', 'hello from js');
   client.end();
});
```

subscriber

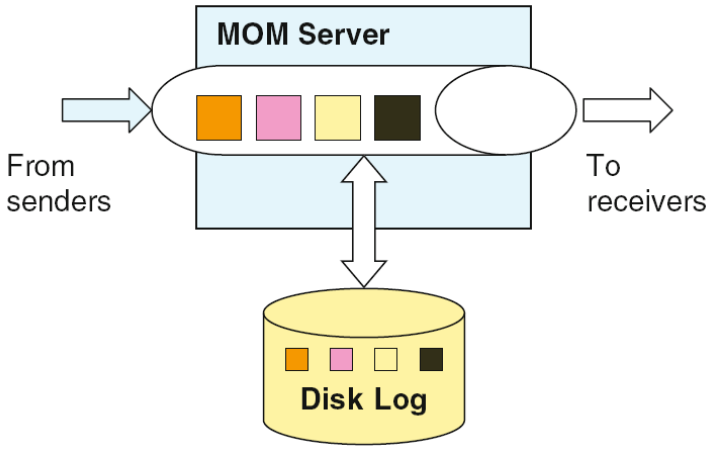
```
const mqtt = require('mqtt');
const client = mqtt.connect();
client.subscribe('MY_TOPIC');
client.on('message', function (topic, message) {
   console.log(message.toString());
   client.end();
});
```

Reliable Message Delivery

- Reliability comes with the price of performance
- 三種常見的broker支援reliable message方式
  - Best effort　　盡力而為
  - Persistent　　未送出就先儲存訊息
  - Transactional　　最嚴重，沒傳送成功就會Error

Reliable Message Delivery

- Best effort 送出就不管，射後不理
  - Undelivered messages are only kept in memory and can be lost if a system fails
- Persistent 存進資料庫
  - Undelivered messages are logged to disk
  - Can be recovered and delivered after a system failure

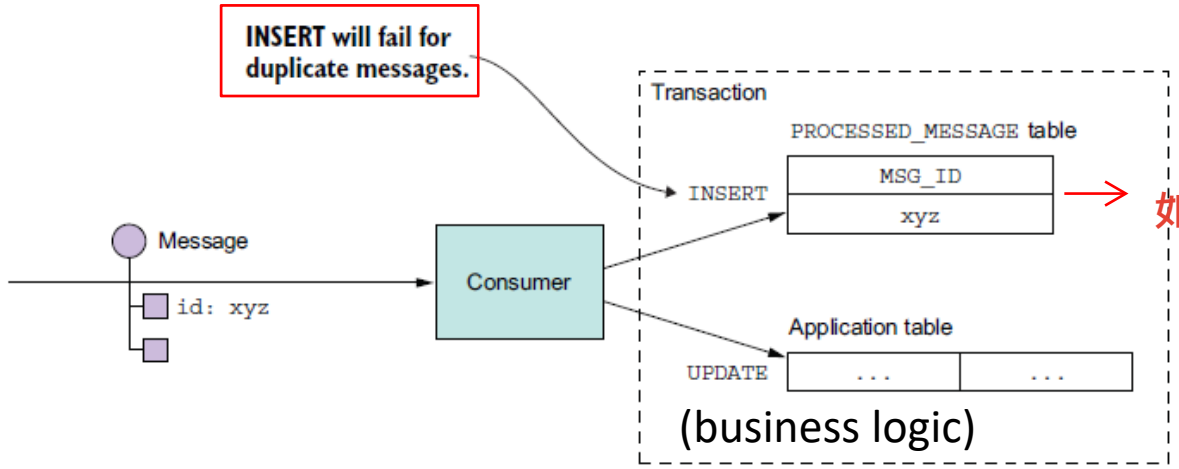


Handling duplicate messages

- Two options
  - Idempotent message handler 重覆訊息不影響邏輯正確性
  - Track messages and discard duplicates

如果 non-idempotent

- A simple implementation
  - Insert message_id of each message into a local database
  - Message_id as a primary key: error on duplication



primary key
如果訊息重覆就會報錯

下方邏輯也會 Error

WIRESHARK => Header flag 裡面有寫號碼，也有英文

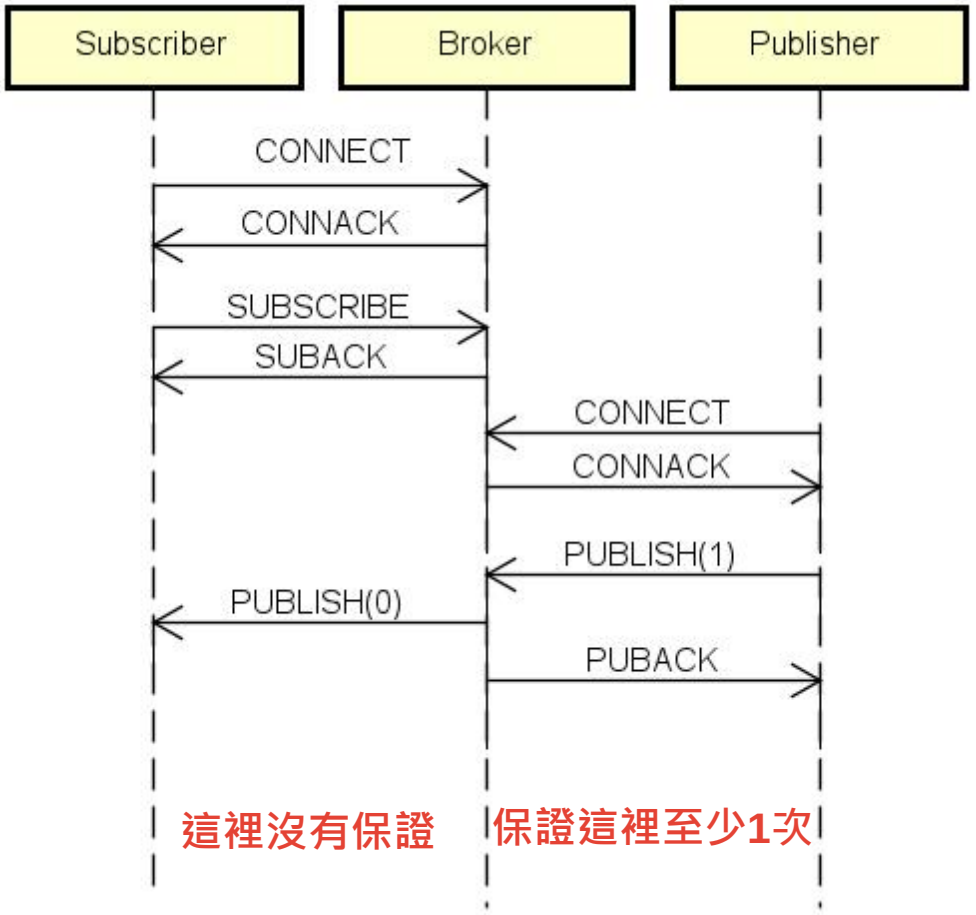
Case: MQTT QoS and Last Will

- QoS 0 “At most once”，最多一次
  - 訊息遺失或是重複發送的狀況可能會發生
  - Ex: 溫度感測
- QoS 1 "At least once"，至少一次
  - 採用ACK進行訊息確認送達，有問題則重送
  - 若遇到non-idempotent 邏輯，訊息重複會造成錯誤
- QoS 2 "Exactly once"，確定一次
  - 確認訊息只會送到一次
  - 耗費較多資源與網路頻寬
- Last Will
  - 連線後可事先設定Last Will，敘明當異常斷線發生時的處理方式

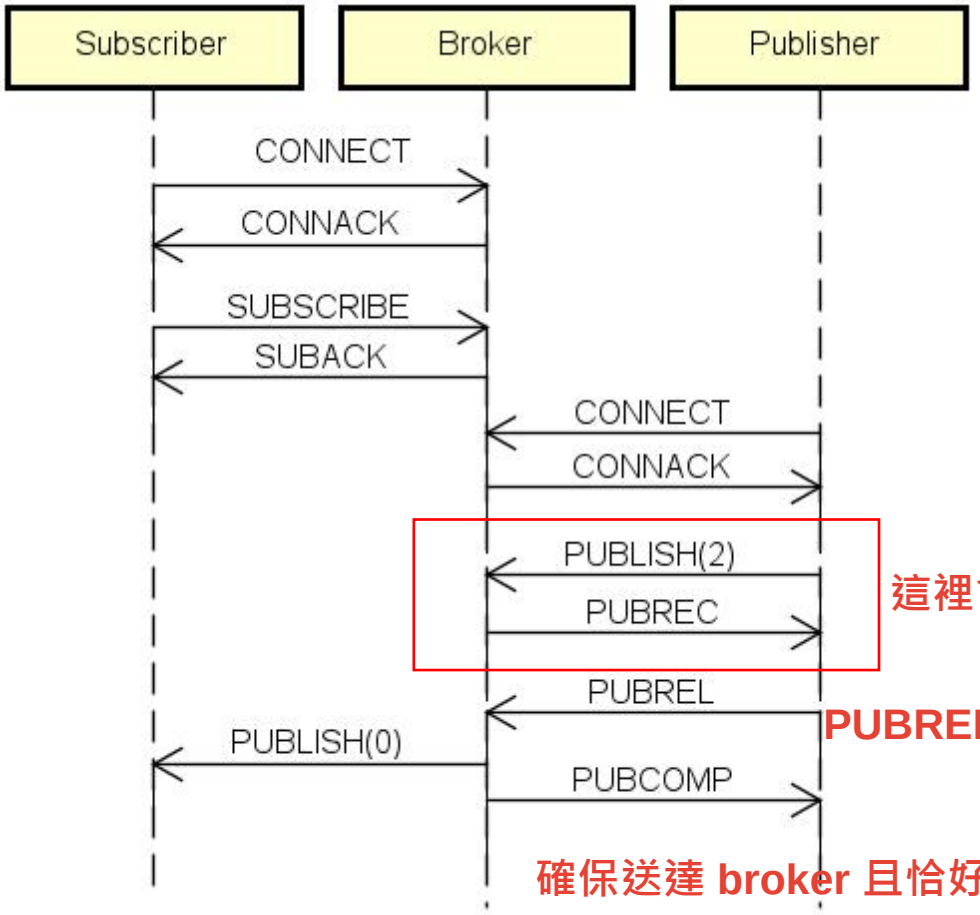
idempotent => 做一次和做很多次結果相同

確保 non-idempotent

Case: MQTT QoS 1 (Publisher-side only)



Case: MQTT QoS 2 (Publisher-side only)



這裡會送很多次，直到 **PUBREL** 送出

PUBREL 只能選一個送出，不然也會出錯

確保送達 **broker** 且恰好一次

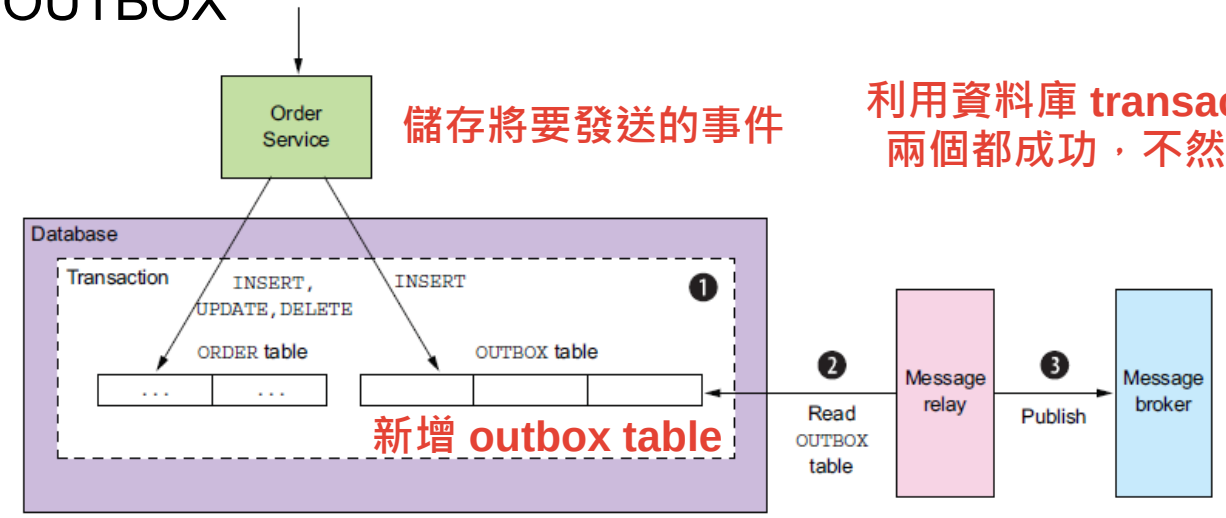
希望每筆交易和訂單都要讓大家知道
有些時候修改完沒來得及傳出就死了，就會掉訊息

Transactional Messaging

- Motivation and problem
  - A message endpoint need to publish an event when it updates its states
    - Update → Publish
  - Update without notification can occur
    - Update → (endpoint crashes)

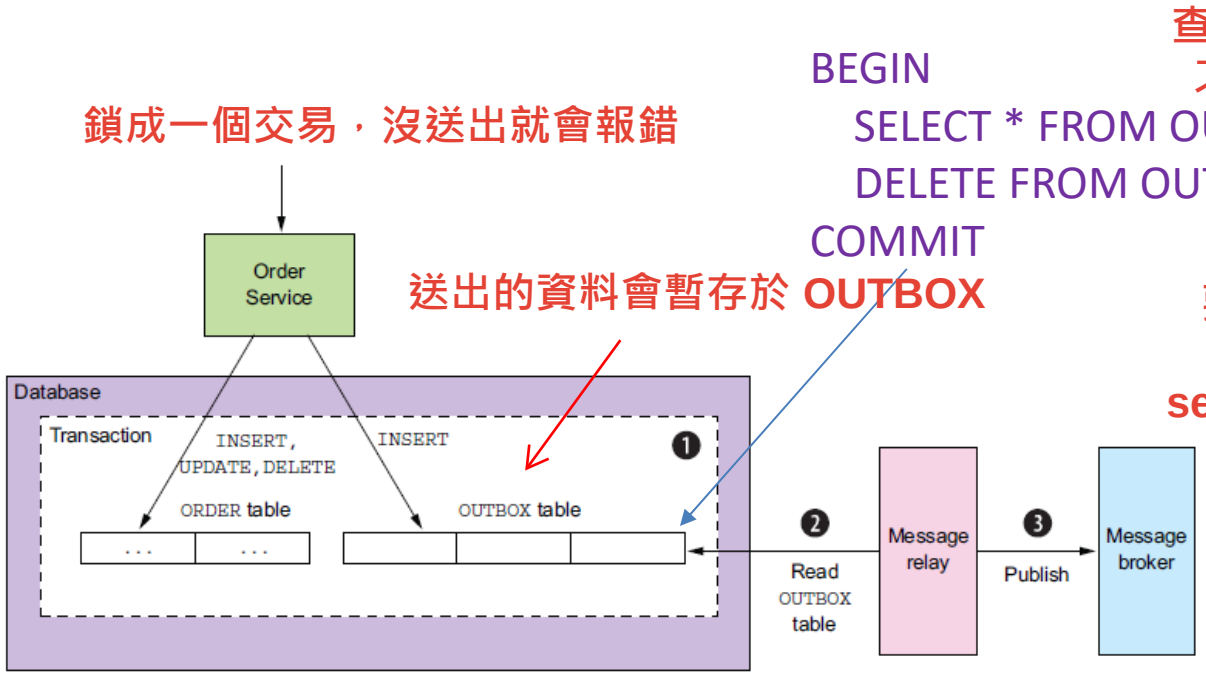
Transactional Outbox

- Solution
  - Write the message to be published into a OUTBOX table
  - Updates to domain tables and the OUTBOX table are bundled into a Transaction
    - 完成ORDER修改→ Crash → OUTBOX還沒改所以rollback
  - Using a message relay to constantly poll-publish-delete the records in OUTBOX



Realizing Message Relay

- Polling publisher
  - The message relay constantly query the OUTBOX, publish events, and then clear OUTBOX in a transaction
  - Cons: Frequently polling the database can be expensive



底層操作，是用 tail，有 system code，不會對資料庫造成負擔

Realizing Message Relay

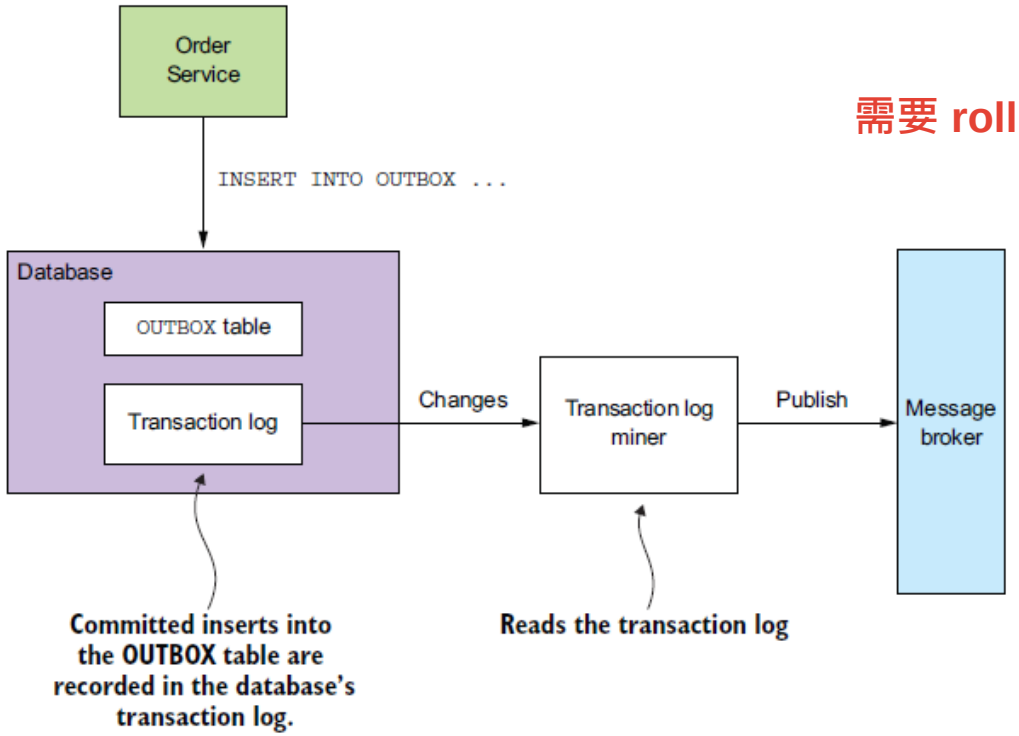
為了解決上方效能的問題

- Transaction Log Tailing

從 log(資料庫底層)裡去撈，就不用與 2 競爭
但是滿 hack 的(要parse log)，因為不同資料庫的 log 都不相同，不好寫，但其實現有些 Open Source 的專案可以做

  - Use a transaction log miner to read the low-level transaction logs of the DB and publish each change to the message broker

業界很多人這樣做
可以有不錯的效能
需要 rollback 的 event 也能確保知道



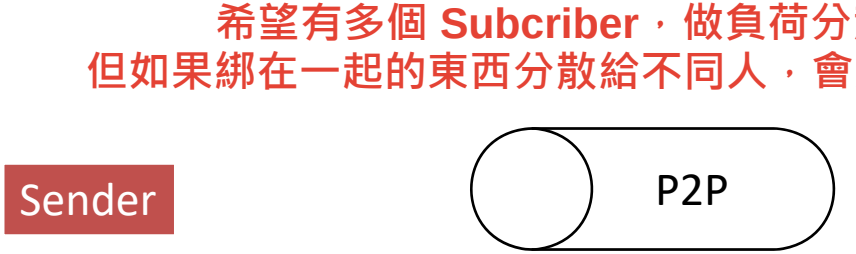
Realizing Message Relay

- Transaction Log Tailing Tools
  - Debezium
    - publishes database changes to the Apache Kafka message broker
  - Eventuate Tram (by Chris Richardson)
    - An enhanced version of Debezium
  - LinkedIn Databus
    - mines the Oracle transaction log
  - DynamoDB streams
    - DynamoDB built-in service

Message Ordering Problem

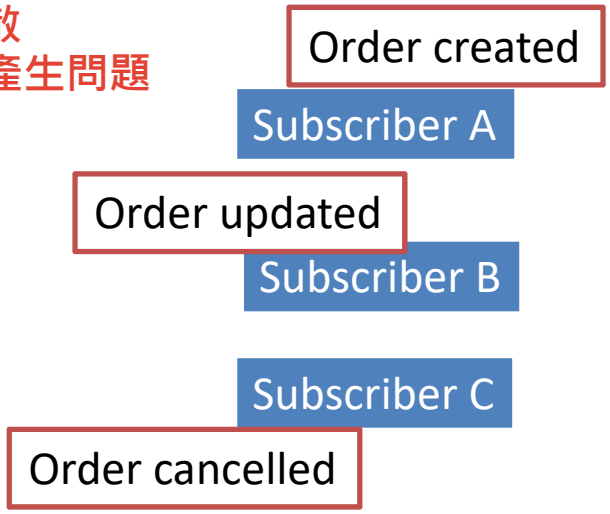
- Multiple subscribers handling messages from the same topic
  - how to ensure the message is processed once and in order?
  - Example: (order created, order updated, order cancelled)

希望有多個 **Subcriber**，做負荷分散
但如果綁在一起的東西分散給不同人，會產生問題



Created → updated → cancelled

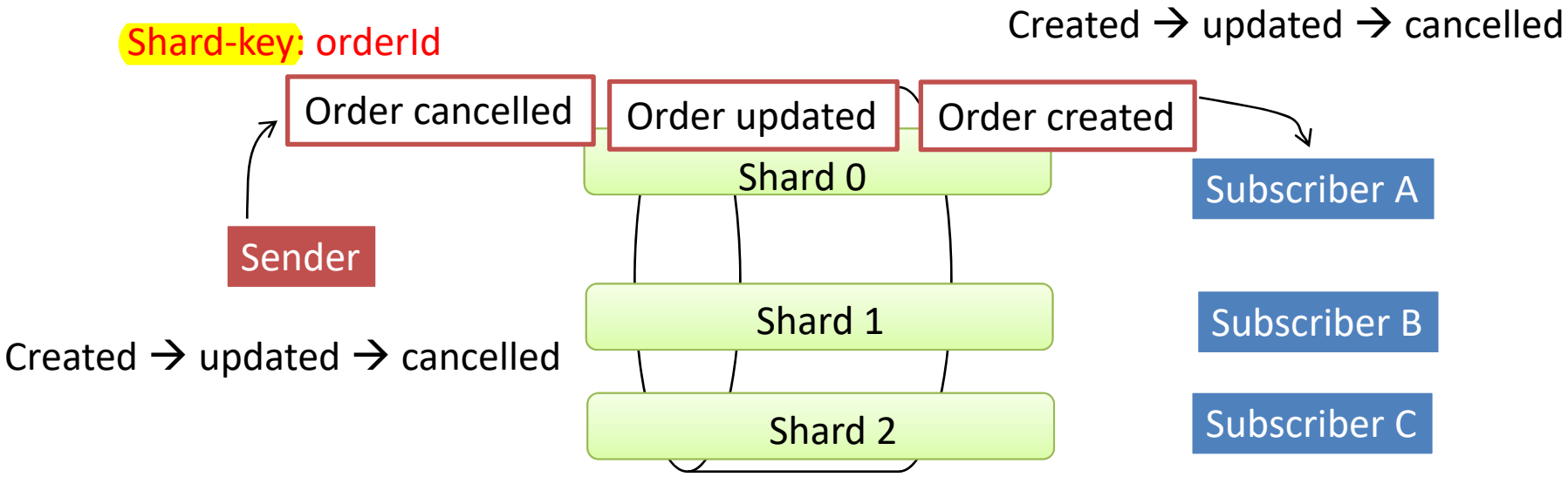
希望讓同一個人處理，避免順序被打亂的問題



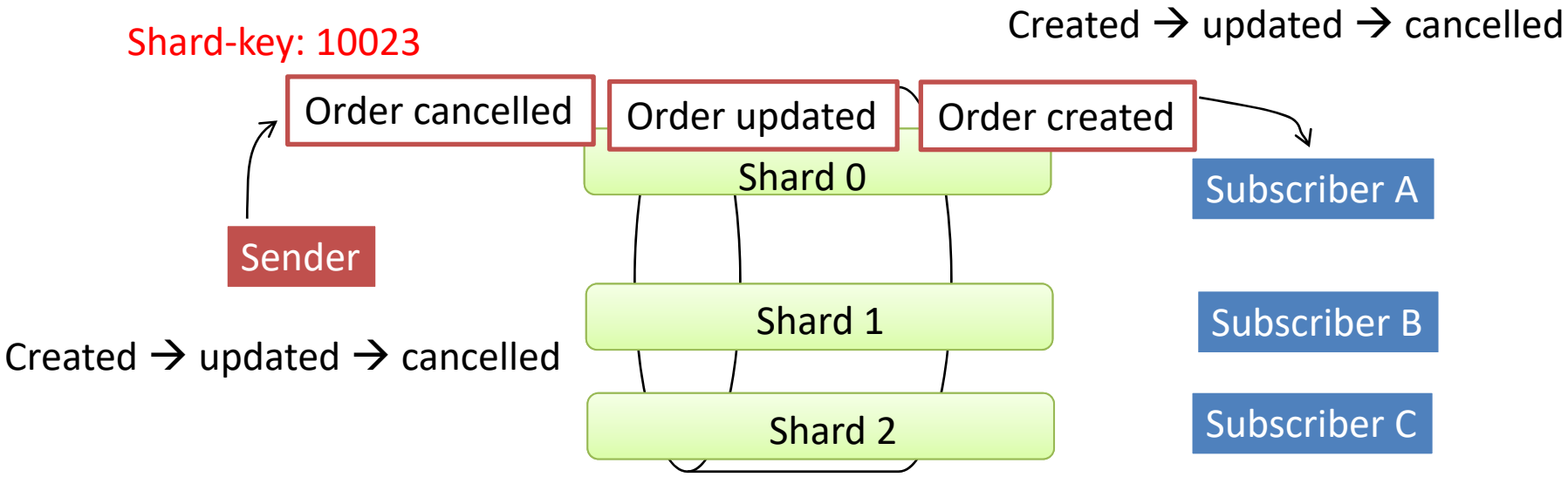
cancelled → created → updated

Channel Sharding

可以分散負荷，也可以按照次序傳入(因為同個 Shard)

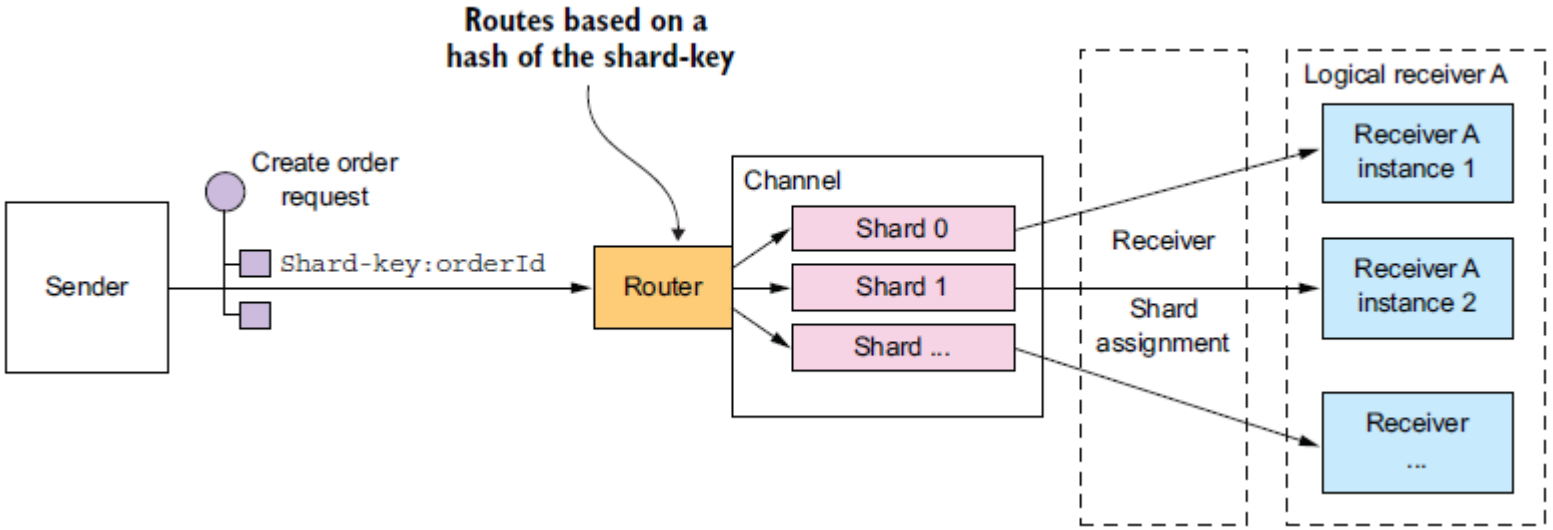


將需要具有次序的一群訊息，以shard-key group起來
MOM會將它們綁定同一個shard queue中


Shard-key: 10023
Created → updated → cancelled
Order cancelled
Order updated
Order created
Shard 0
Subscriber A
Sender
Shard 1
Subscriber B
Created → updated → cancelled
Shard 2
Subscriber C

Channel Sharding

- Mechanism
  - A group of subscribers process the same channel
  - Broker splits a channel into two or more shards, each of which behaves like a channel
  - Sender specifies a shard key in header
  - Broker uses the shard key to assign the message to particular shard 同一個shard key會送往同一個shard



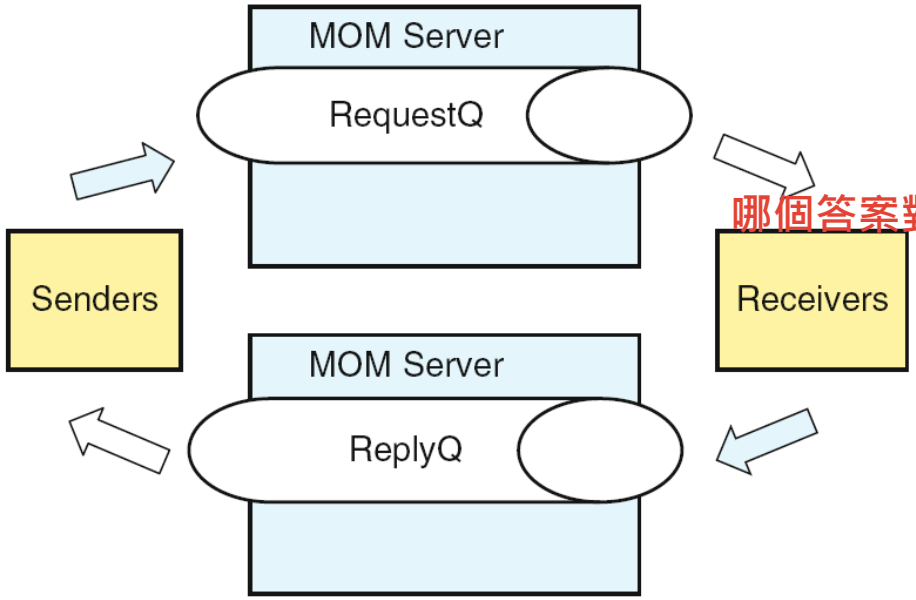
Simulating RPC

MOM 希望模擬 remoting(request, respond)的行為

MOM 是訂閱，發布

透過一些技巧，希望實現遠端呼叫(計算之類的)

- MOM can also be used for synchronous communications
- Frequently used in enterprise systems to replace conventional synchronous technology



如果兩邊都不知道對方
不知道要送往哪裡(如果預先沒設定)
Sender & Receivers怎麼把問題和答案兜起來

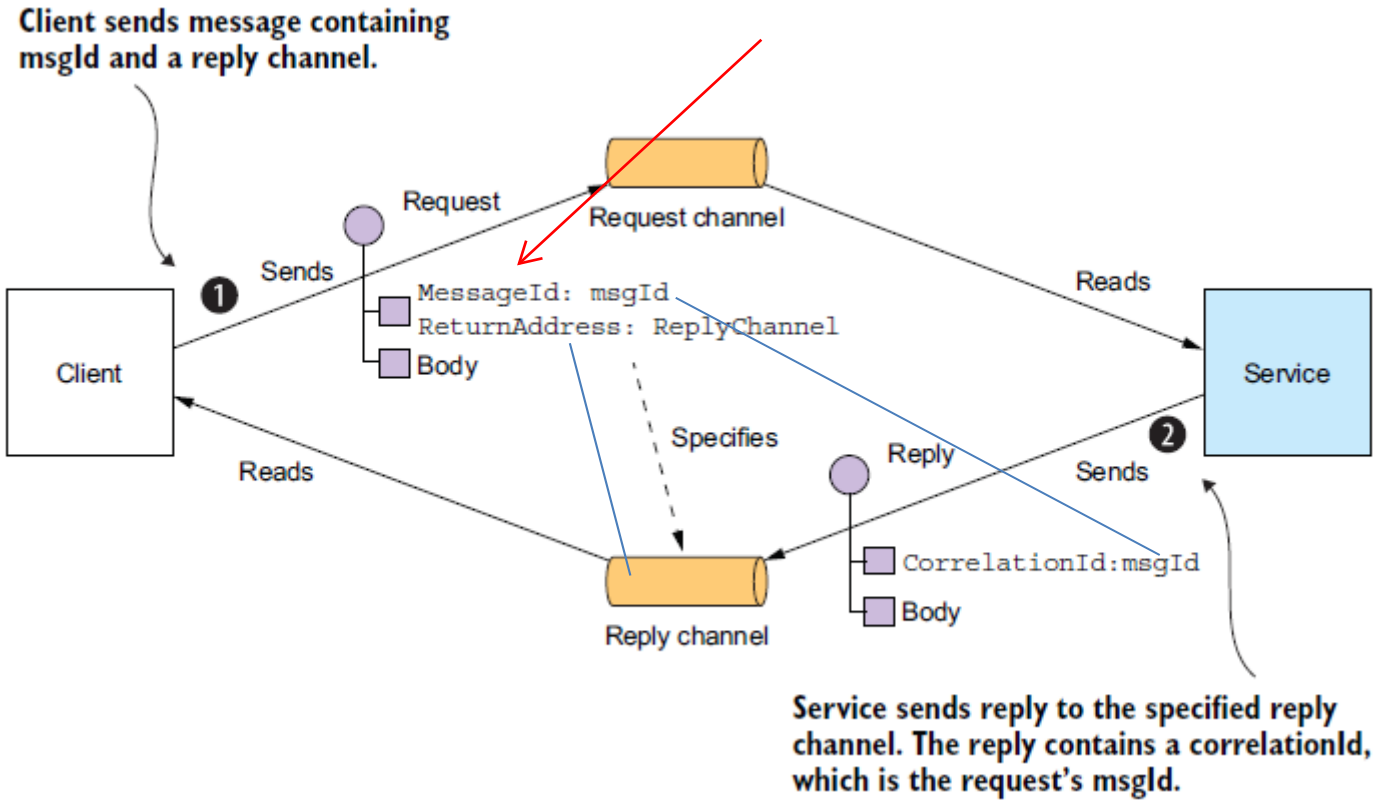
哪個答案對應到哪個問題(如果沒按照發送順序算出答案)

1. 如何協調回傳值的資訊
2. message 怎麼對得起來

透過 message 的 header 就可以解決

Simulating a Call

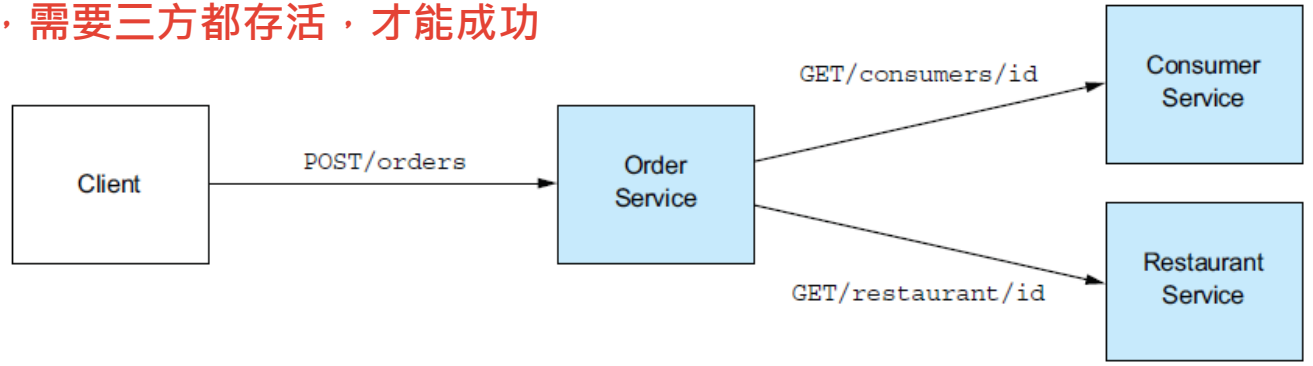
header 新增欄位
1. reply channel 的名字放在 value 裡
2. 告訴他 msgId



Case:提升RESTful WS的availability

- 如下圖中的同步RPC，呼叫時，三個服務必須同時available

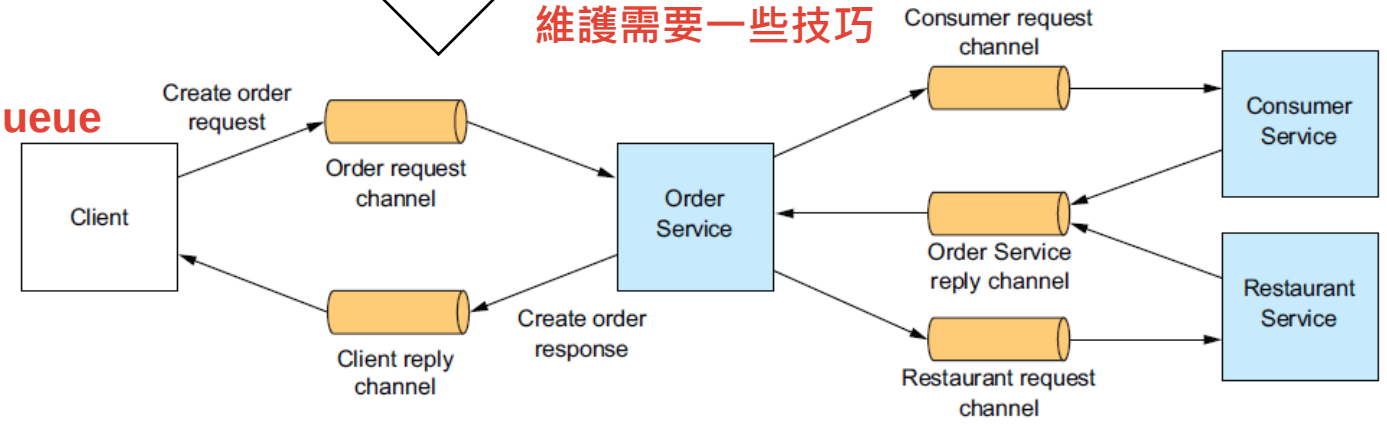
傳統呼叫，需要三方都存活，才能成功



變成非同步之後，若服務不available，訊息會先存在MQ中

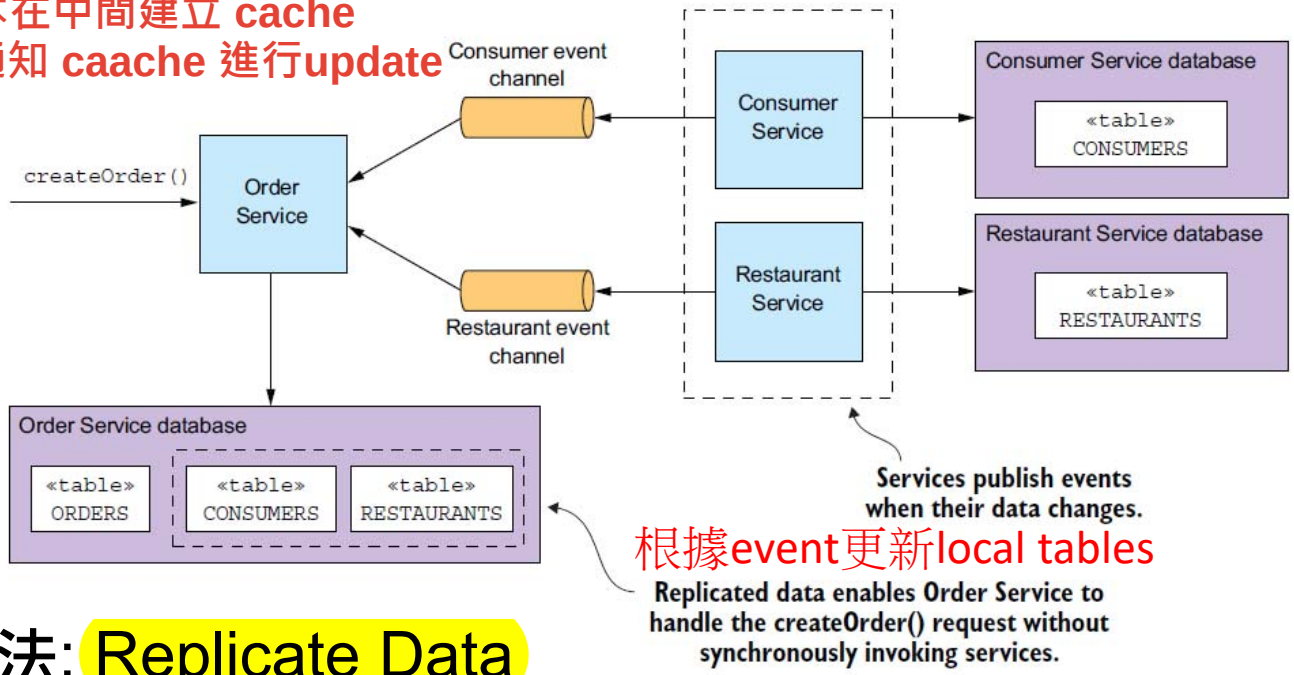
業界幾乎用此維護需要一些技巧

MOM 放在中間
中間留個 message queue



複製後兩個的副本在中間建立 cache
如果真的被改了，再通知 caache 進行update

有更新時發出 event

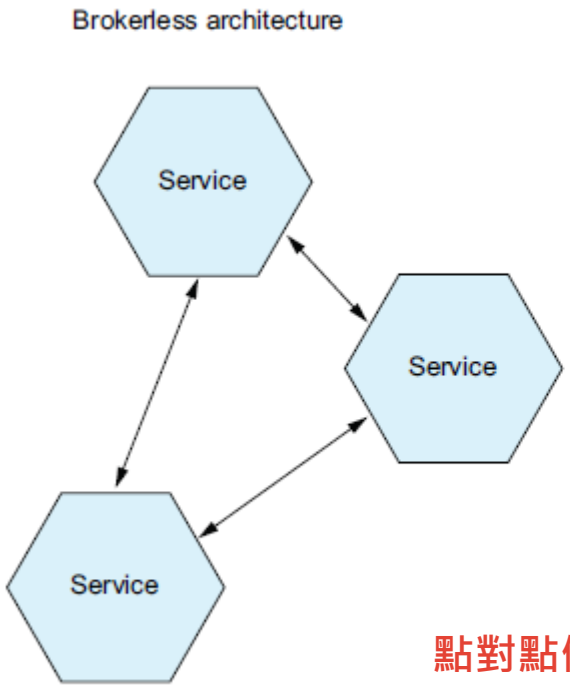


- 另一種做法: Replicate Data
  - Order對Consumer與Restaurant都只是查詢
  - 在Order中維護一份Consumer與Restaurant的資料庫副本
  - Order中的資料表透過Domain Event，同步更新
  - 如此一來，Order不需要查詢另二個服務(少二次RPC)就可完成
  - 缺點: 只適用於查詢
    - 有寫入動作的話會產生資料同步問題

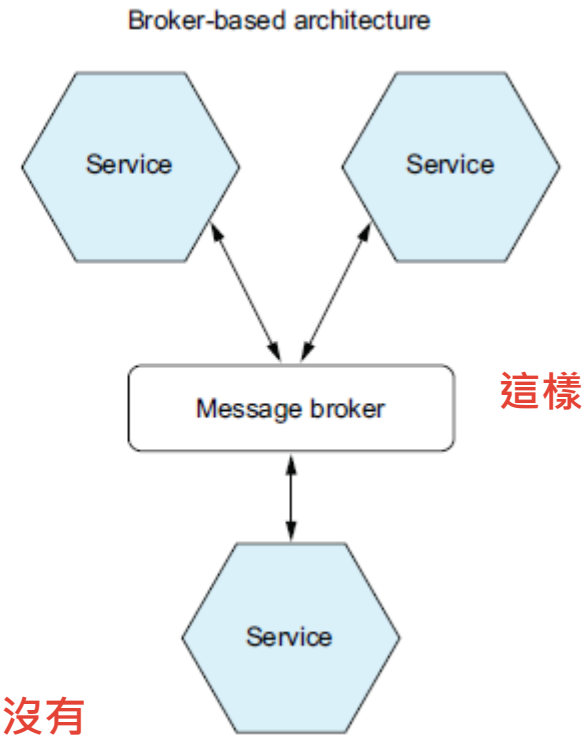
犧牲 consistency 換取 availability

如果是寫入頻繁的會比 RPC 慢

Two Types of Message Brokers



Vs.



這樣架構最怕 broker 死掉

點對點傳送

只有邏輯上有 broker，實際上沒有

In the broker-less architecture, each endpoint sees the “logical” address (or URI) of the virtual broker

Brokerless Architecture

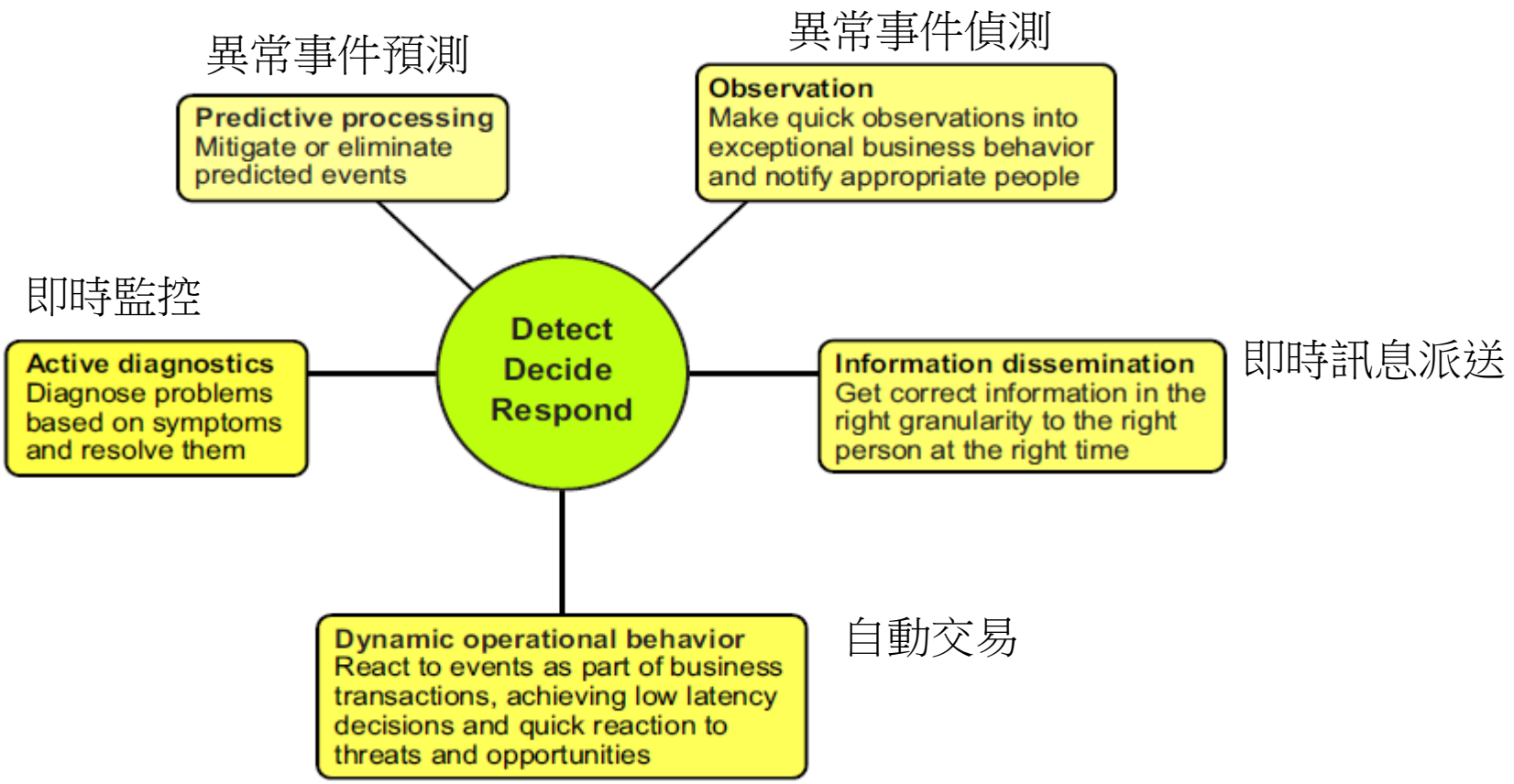
- Pros
  - Light network traffic and better latency
    - Messages go directly from the sender to the receiver
  - Prevent performance bottleneck or a single point of failure
  - Less operational complexity: no broker to setup and maintain
- Cons
  - Need to know about each other's locations 需要知道彼此 IP
    - Use one of the discovery mechanisms
  - Offer reduced availability 犧牲 availability，maintain consistency
    - Both the sender and receiver of a message must be available while the message is being exchanged (傳送過程中，沒有保存訊息之處)
  - Hard to implement guaranteed delivery
- Example
  - ZMQ、multicast

this chapter only to this

Event as a Message

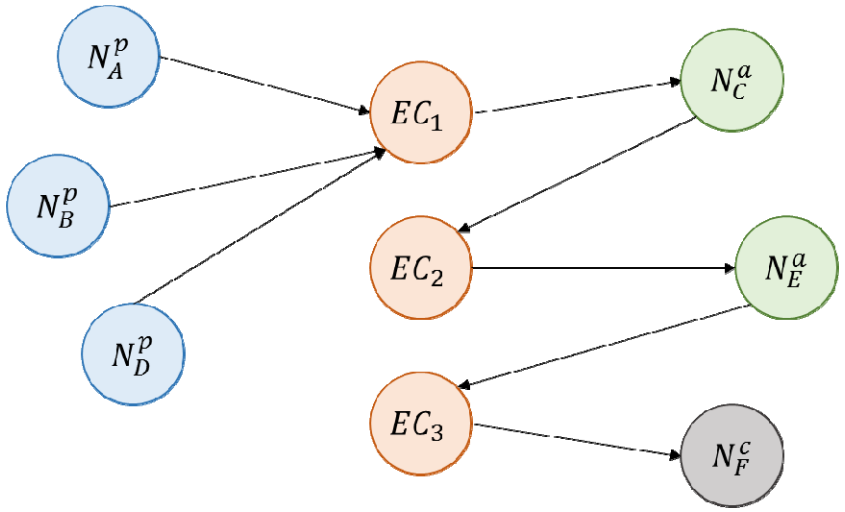
- Event
  - A digital indication of something has happened
  - Typically requires a reaction (computer processing)
- Suitable for
  - IoT systems
  - Robot systems
  - Interactive systems

EDA Applications



Channel-style Event Processing Network

- 由Sharon & Etzion提出，由四個部分組成
  - Event Producer(N^p)
  - Event Consumer(N^c)
  - Event Processing Agent(EPA) (N^a)
    - 可對Event Producer 或其他EPA發出之事件做進一步處理
  - Event Channel(EC)



Complex Event

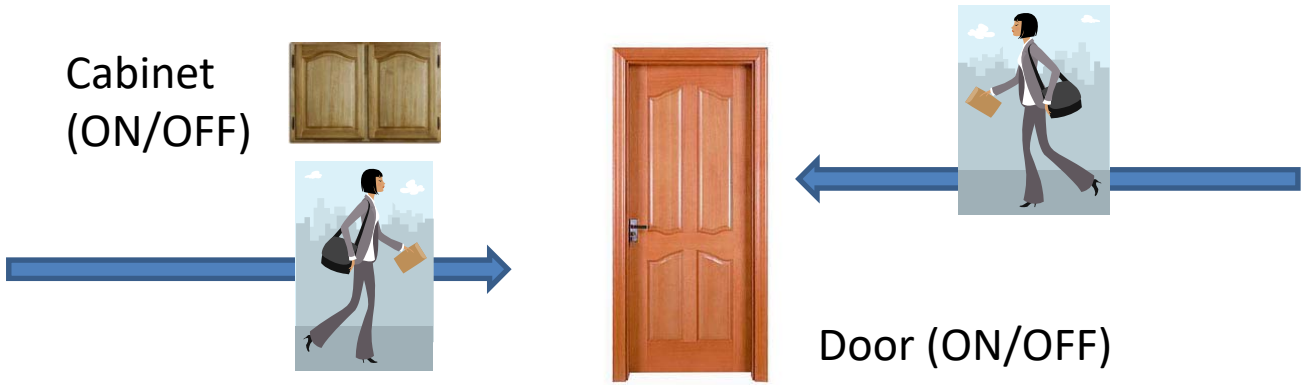
- An event is complex if $|C(e)| > 1$

$C(e) = (\varepsilon_1, \varepsilon_2, ...\varepsilon_n)$, where ε_i is the i th event that causes e to happen

e: event

C(leaveHome)=(Cabinet_ON, Cabinet_OFF, Door_ON, Door_OFF)

C(comeHome)=(Door_ON, Door_OFF, Cabinet_ON, Cabinet_OFF)

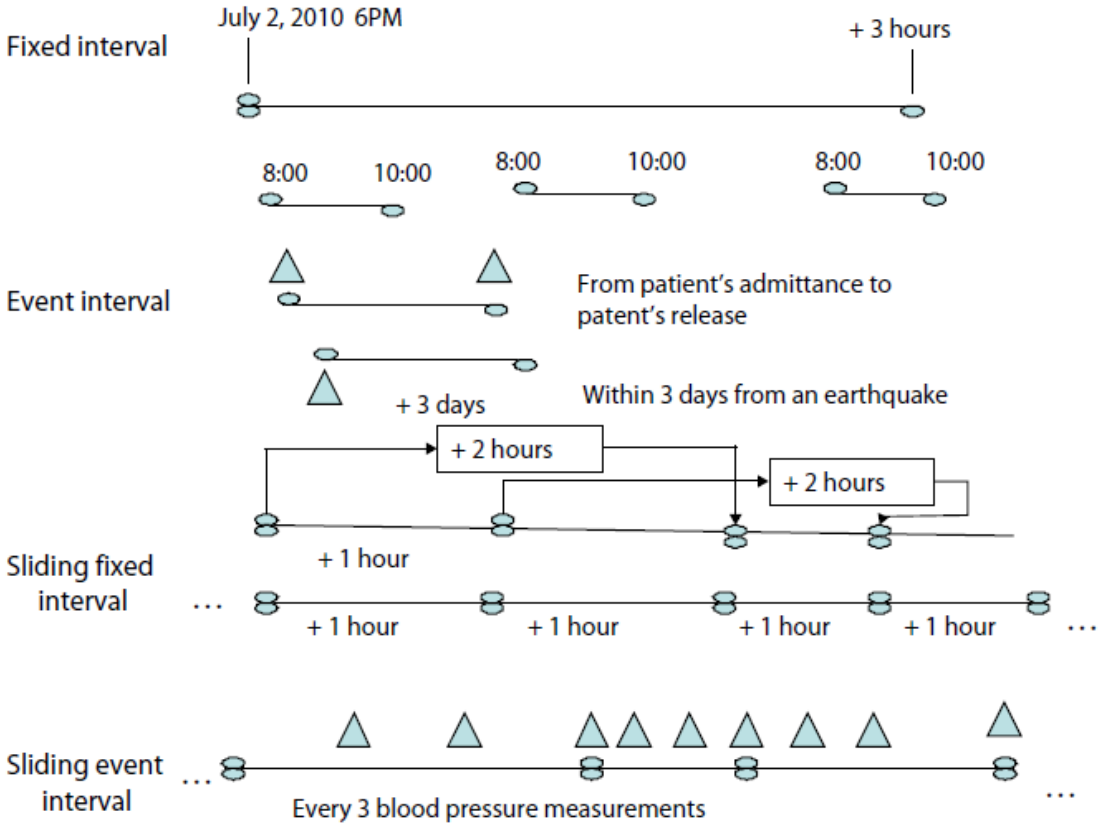


事件前提

- 事件前提(Context)
  - 若不符合此前提則EPA將不予處理
  - 透過事件處理前提的過濾機制，EPA可取得並處理特定事件

Temporal context

- 由一個或多個時間間隔組合成，有些會重疊發生



以絕對時間區分

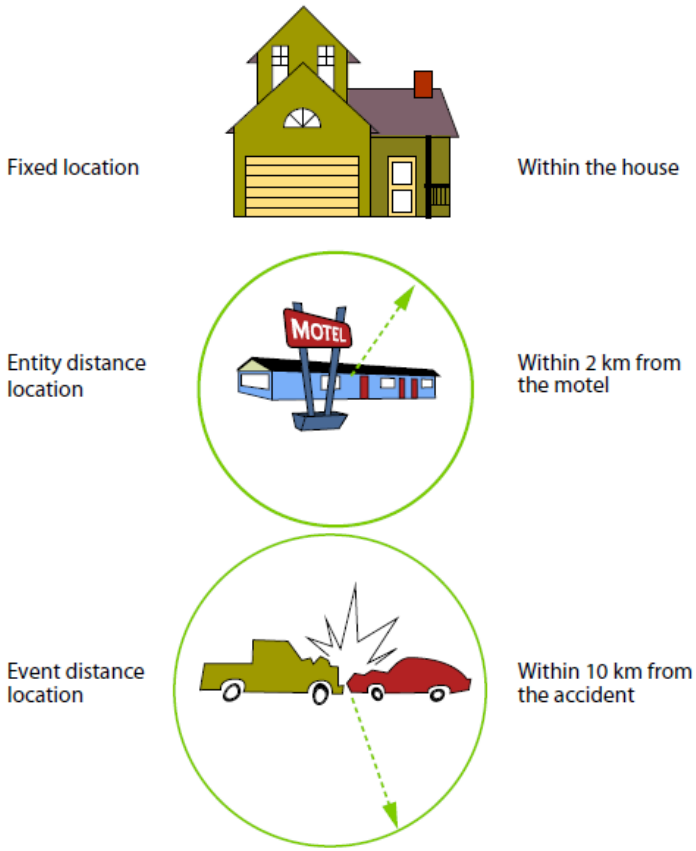
以事件發生區分

以相對時間區分

以事件發生次數區分

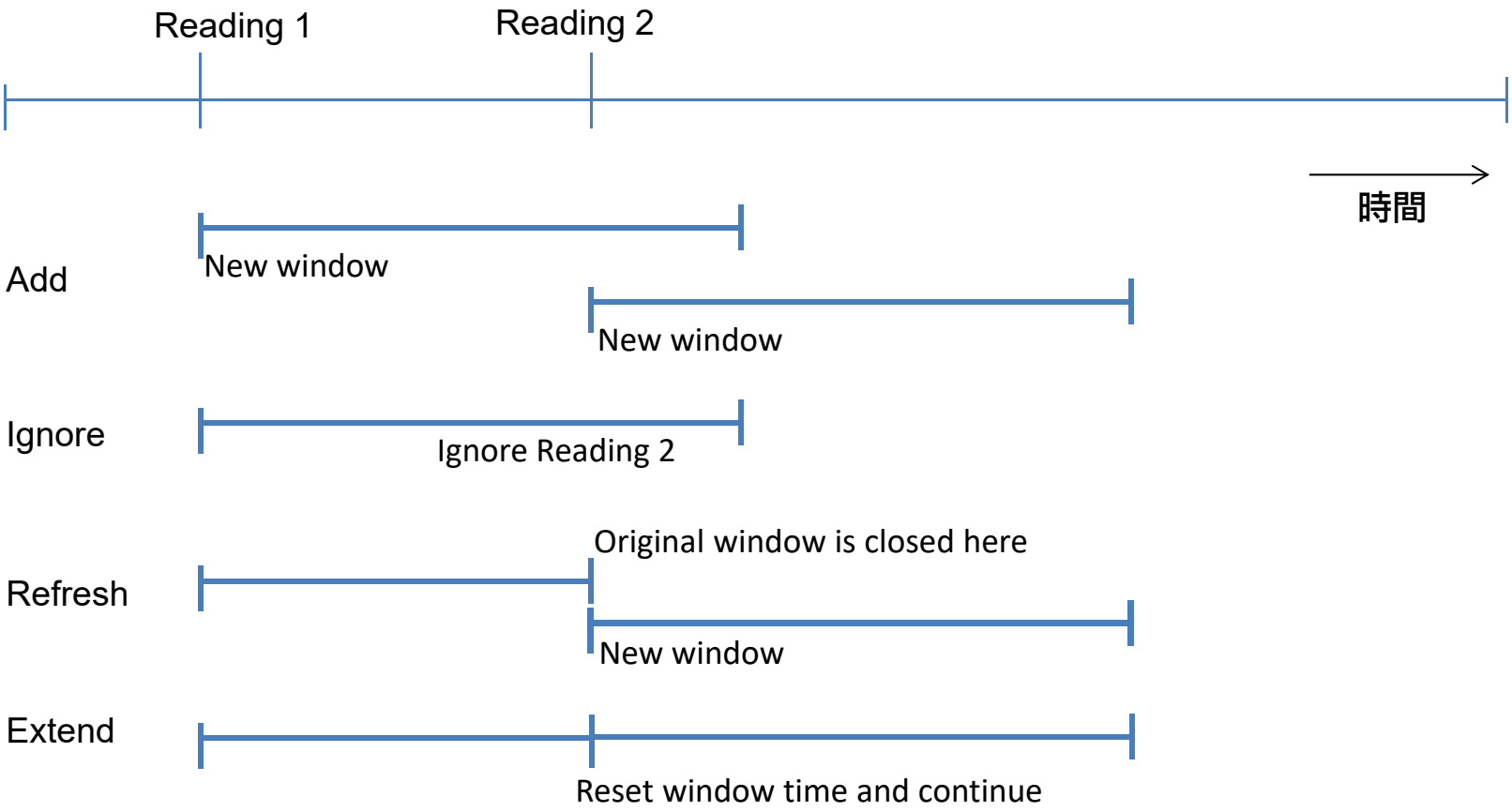
Spatial Context

- The location attribute can take two forms:
  - 座標(經緯度)
  - 空間位置名稱(大仁樓)
- Examples of spatial context:
  - 固定地點(Fixed location)
  - 實體距離 (Entity distance location)
  - 事件距離 (Event distance location)



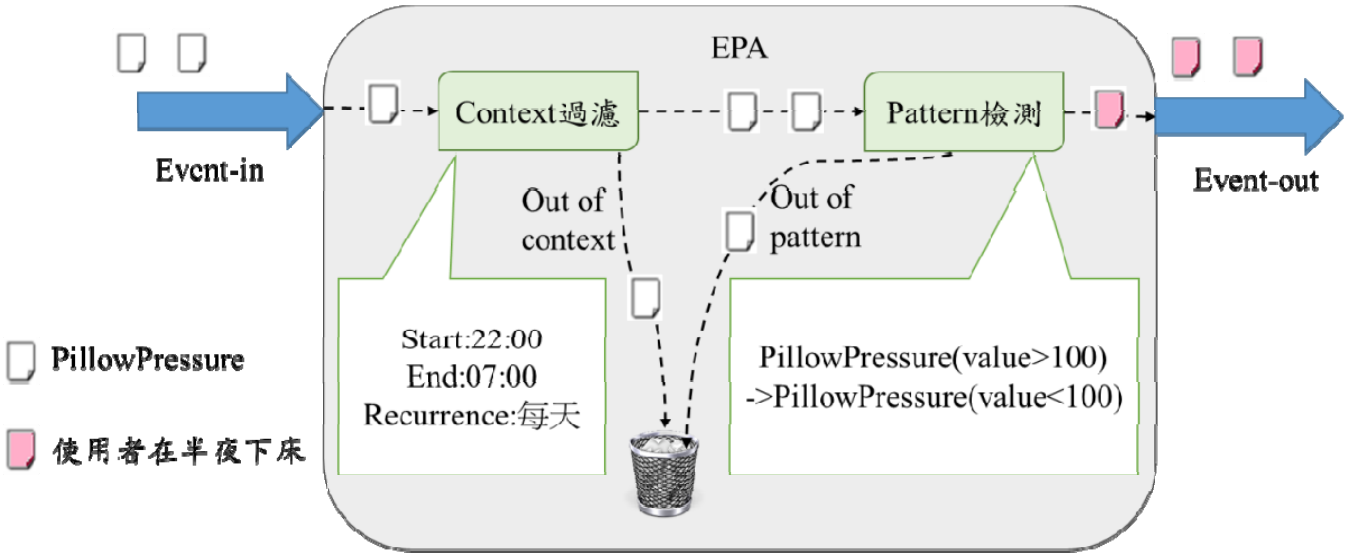
Context Initiator Policy

連續二個相同類型event進來時該如何處理?



範例1

- 當使用者在晚上10:00至早上7:00時下床，則開啟房間與浴室燈
  - 定義Context為重複的固定時間間隔
    - CREATE CONTEXT NightContext start 22:00 end 07:00
  - 定義Pattern為床的壓力感測器數據由>100轉變至<100
    - PillowPressure(value>100) -> PillowPressure(value<100)



範例2

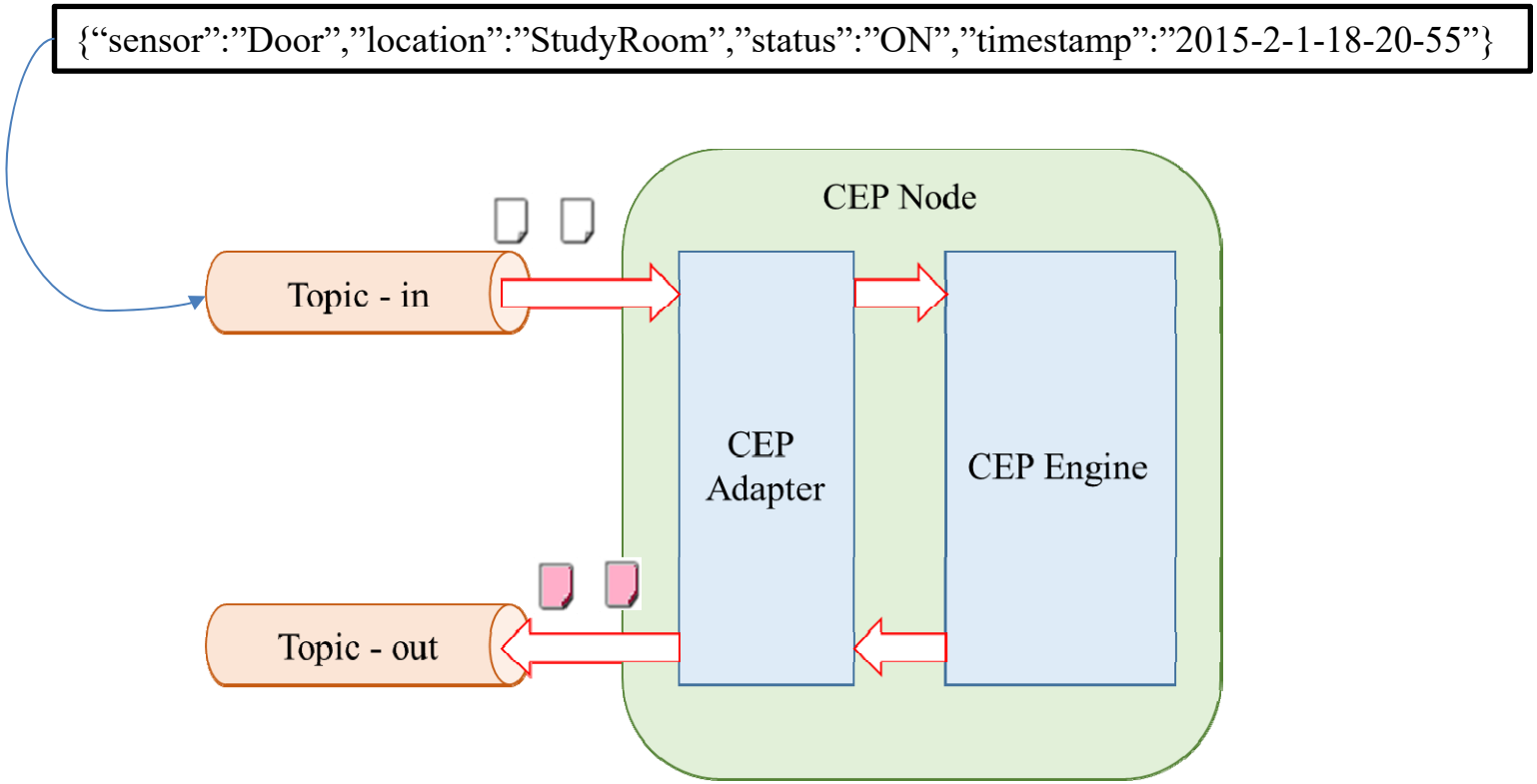
- 當使用者在晚上10:00至早上7:00時進入浴室後20分鐘未回到床上時，發出警告訊息
  - 定義Context為重複的固定時間間隔
    - CREATE CONTEXT NightContext start 22:00 end 07:00
  - 定義Pattern為浴室門感測器狀態由ON轉變至OFF，並且20分鐘內未發生床的壓力感測器數據>100
    - BathroomDoor(status = ‘ON’) -> BathroomDoor(status=’OFF’)

    ->(timer:interval(20 min) and not PillowPressure(value>100))

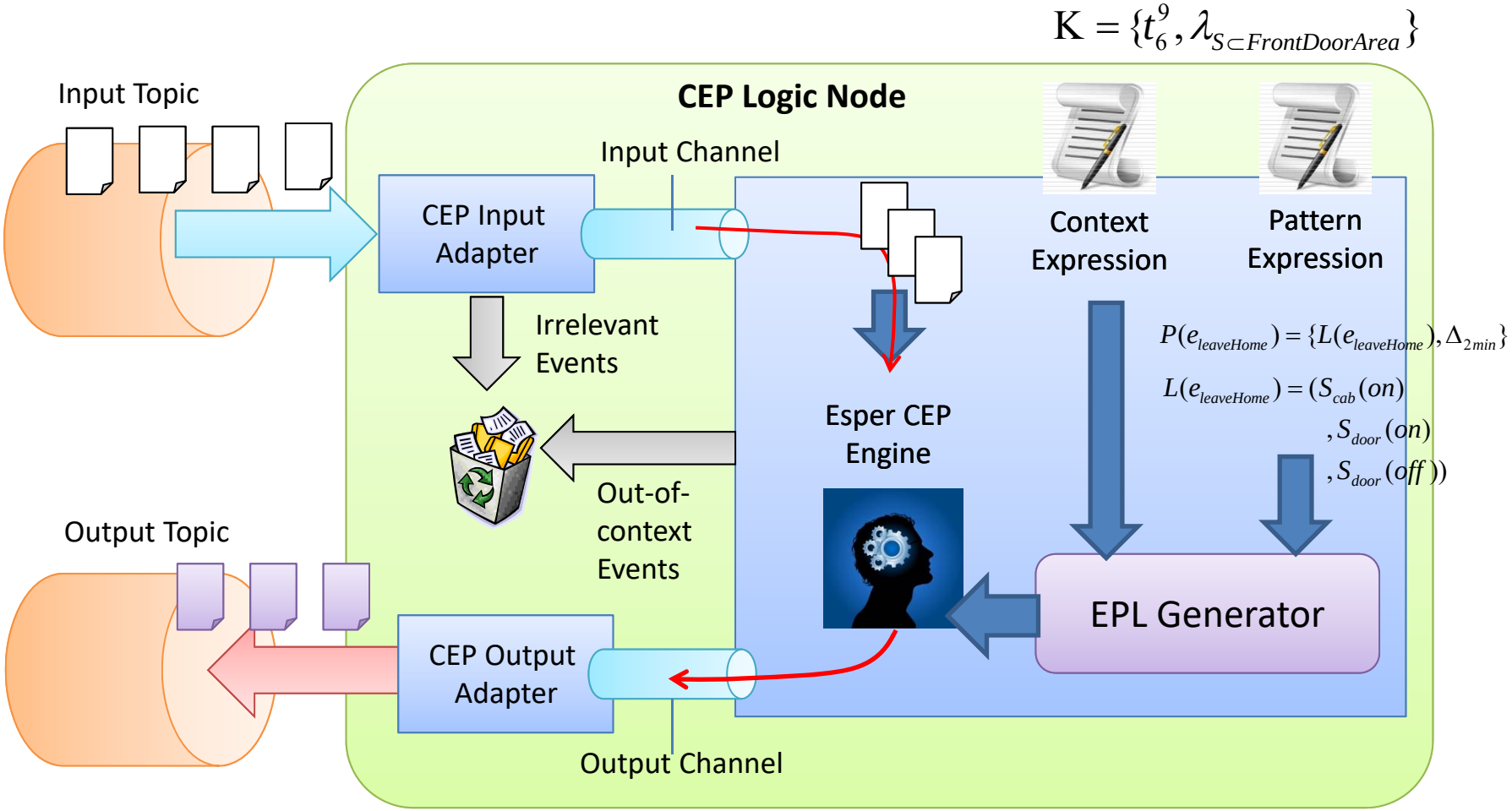
去廁所超過20分鐘還沒回來

CEP+MOM

- 感測器數據透過CEP Adapter轉換成CEP引擎所接受之事件格式



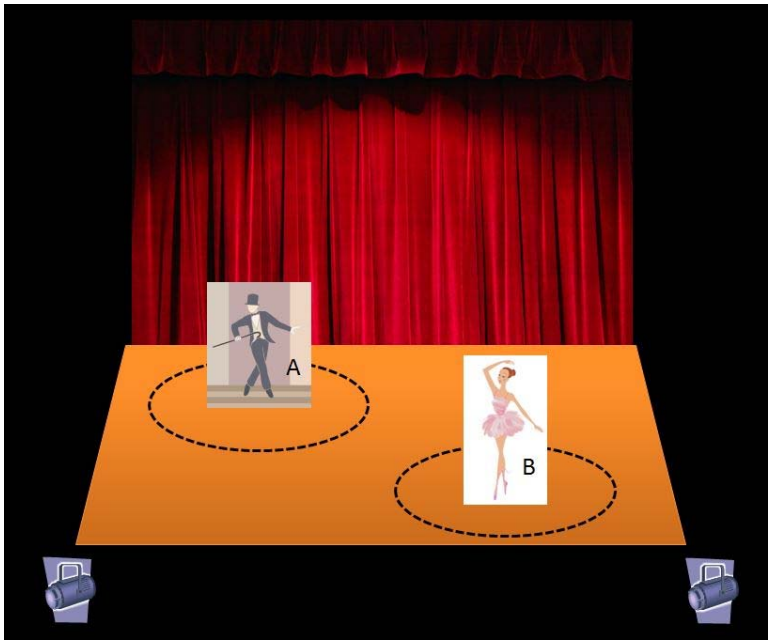
Architecture of a CEP + MOM



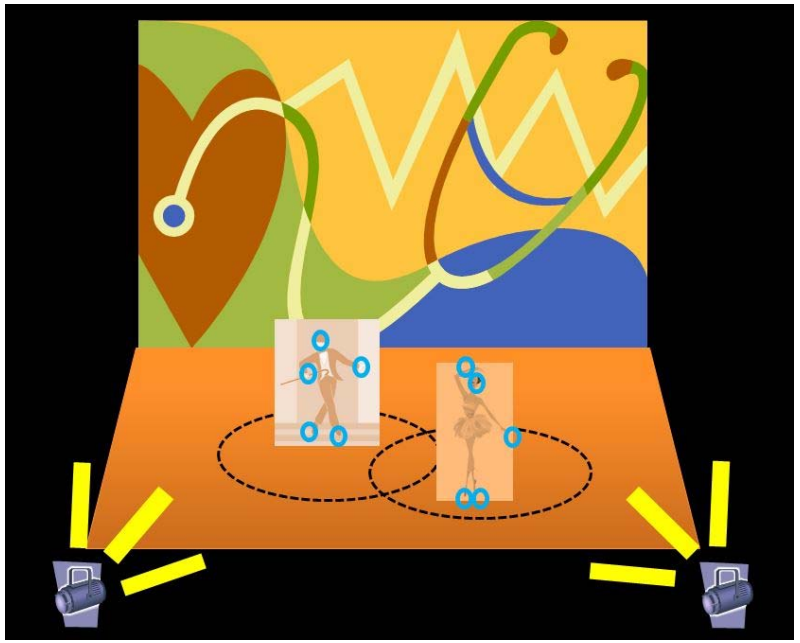
https://youtu.be/SARQhz3LMmQ?t=12

Case: Interactive Performance

- Motivating scenario



Two performers on the stage

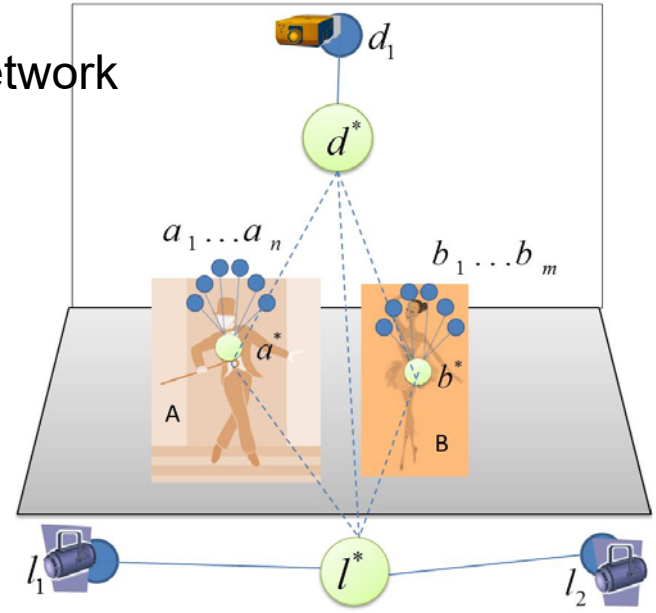


When performers are near, the stage lights are turned on and the animations is projected

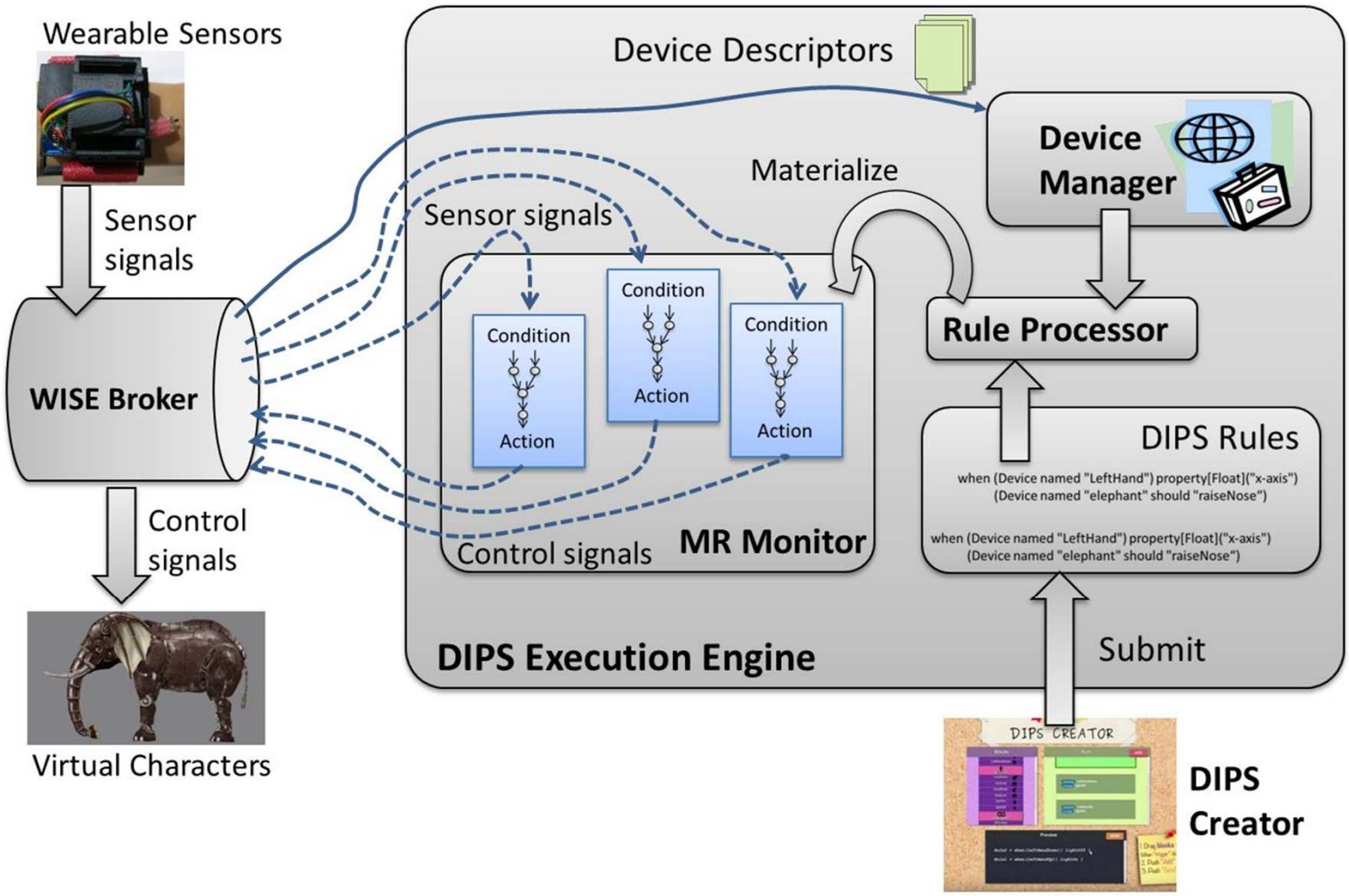
The WISE Platform

- System model
  - WISE Item ($a_1 \ldots a_n, b_1 \ldots b_m$)
    - Wearable devices such as mo-cap or remote controllable LED array
  - WISE Coordinator a^*, b^*
    - Protocol gateways between BAN and IP network
  - WISE Assembly $\alpha = (c^\alpha, I^\alpha) \in C \times 2^I$
    - One coordinator plus a set of items
    - Ex: A is a WISE Assembly where

$$A = (a^*, [a_1 \ldots a_n])$$

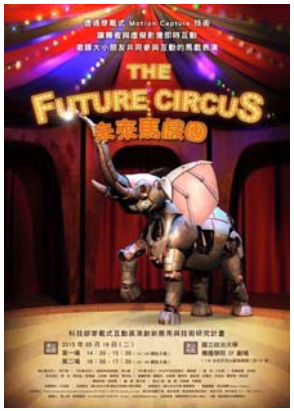


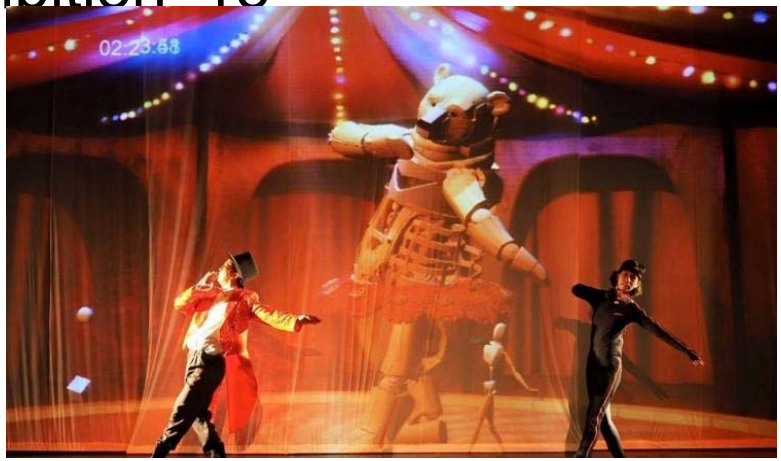
Approach


Wearable Sensors
Sensor signals
WISE Broker
Control signals
Virtual Characters
Device Descriptors
Device Manager
Materialize
Sensor signals
Condition
Action
Condition
Action
Condition
Action
Control signals
MR Monitor
Rule Processor
DIPS Rules
when (Device named "LeftHand") property[Float]("x-axis")
(Device named "elephant" should "raiseNose")
when (Device named "LeftHand") property[Float]("x-axis")
(Device named "elephant" should "raiseNose")
DIPS Execution Engine
Submit
DIPS CREATOR
DIPS Creator

Cases and Applications

- Future Circus
  - A proof of concept performance realized based on WISE
  - Publicly performed or demonstrated in Kaohsiung Exhibition Center, Digital Art Festival Taipei'15, and ACM UbiComp/ISWC Design Exhibition '15

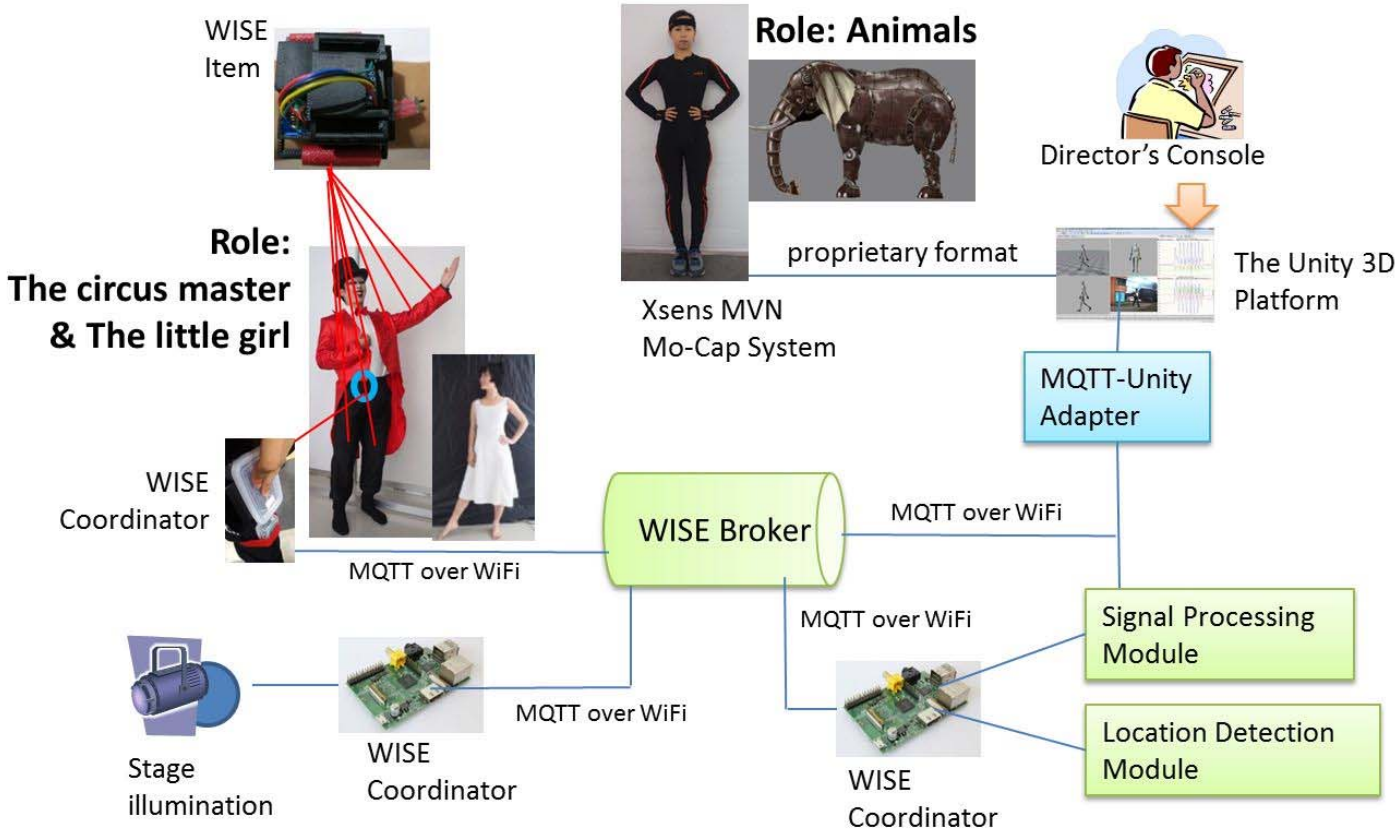




Hsin Huang, Hsin-Chien Huang, Chun-Feng Liao, Ying-Chun Li, Tzu-Chieh Tsai, Li-jia Teng, and Shih Wei Wang, "Future Circus: A Performer-Guided Mixed-reality Performance Art, " in Adjunct Proc. International Symposium on Wearable Computers (ISWC), Design Exhibition track, Osaka, Japan, 2015.

Future Circus

- System architecture

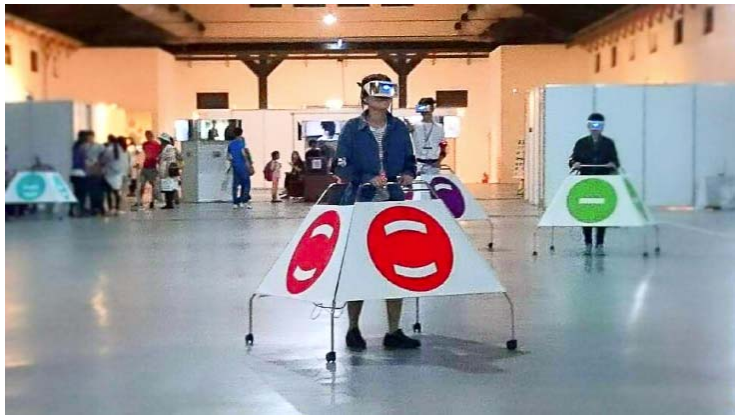


Cases and Applications

Step In and Out of the Dreams

Ya-Lun Tao, Chun-Feng Liao, Hsuan Huang, Ya-Wen Su, Yo-Ja Lin, Pin-Hsin Chen, Tzu-Chieh Tsai, “Step In and Out of the Dreams: Toward an Immersive and Interactive Virtual Experience of Dreams, ” in Adjunct Proc. International Symposium on Wearable Computers (ISWC), Design Exhibition track, Heidelberg, Germany, 2016.

2015/10/17~10/22
Songshan Cultural and Creative Park
松山文創園區1號倉庫



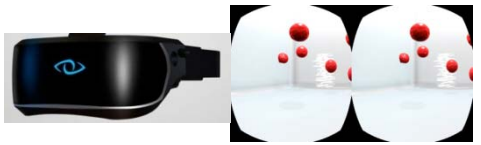
Dreaming Device Design

- Supporting car
  - Laptop and batteries
- Wearable devices
  - VR glasses
  - WISE Items
- Global positioning
  - Camera
  - Label

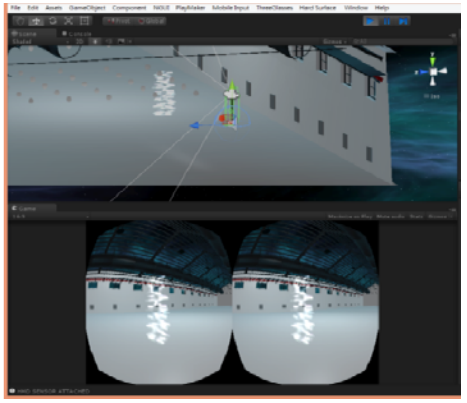
Implementation: Step in and out of the Dream

VR glasses

Global positioning
- Camera-based
- Also calculate relative positions



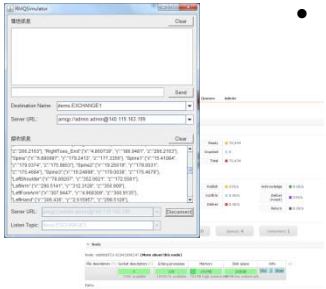
Rendering
- Rendering results to the VR glasses



Mo-cap
- Sense motions
- One-click calibration
- 30 FPS

WISE Broker

WISE
- Monitoring and broking messages



Virtual world (Unity)
- Multi-user navigation
- Reactions and effects control

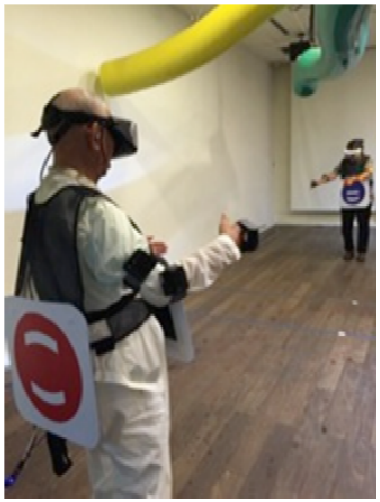
Cases and Applications: Electronic Skin

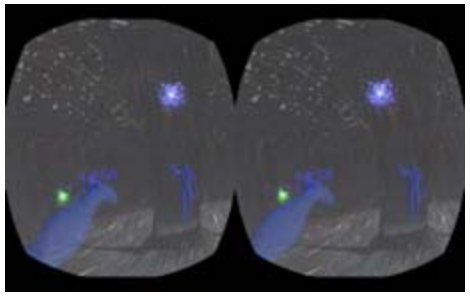
2016/5/7~5/15
Museum of Contemporary Art, Taipei
台北當代藝術館

https://www.youtube.com/watch?v=Z2Zv0S68XII

Electronic Skin

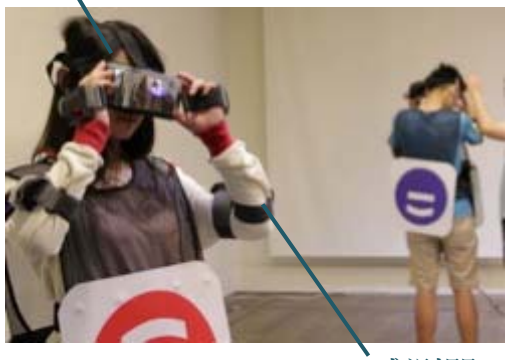
Global positioning
- Camera-based
- Also calculate relative positions





Rendering
- Rendering results to the VR glasses
- Also renders motions of hands

VR glasses

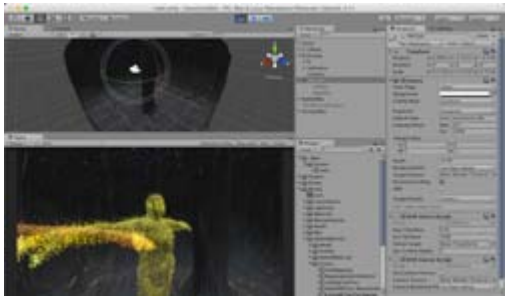


感測器

Mo-cap
- Sense motions
- One-click calibration
- 60 FPS

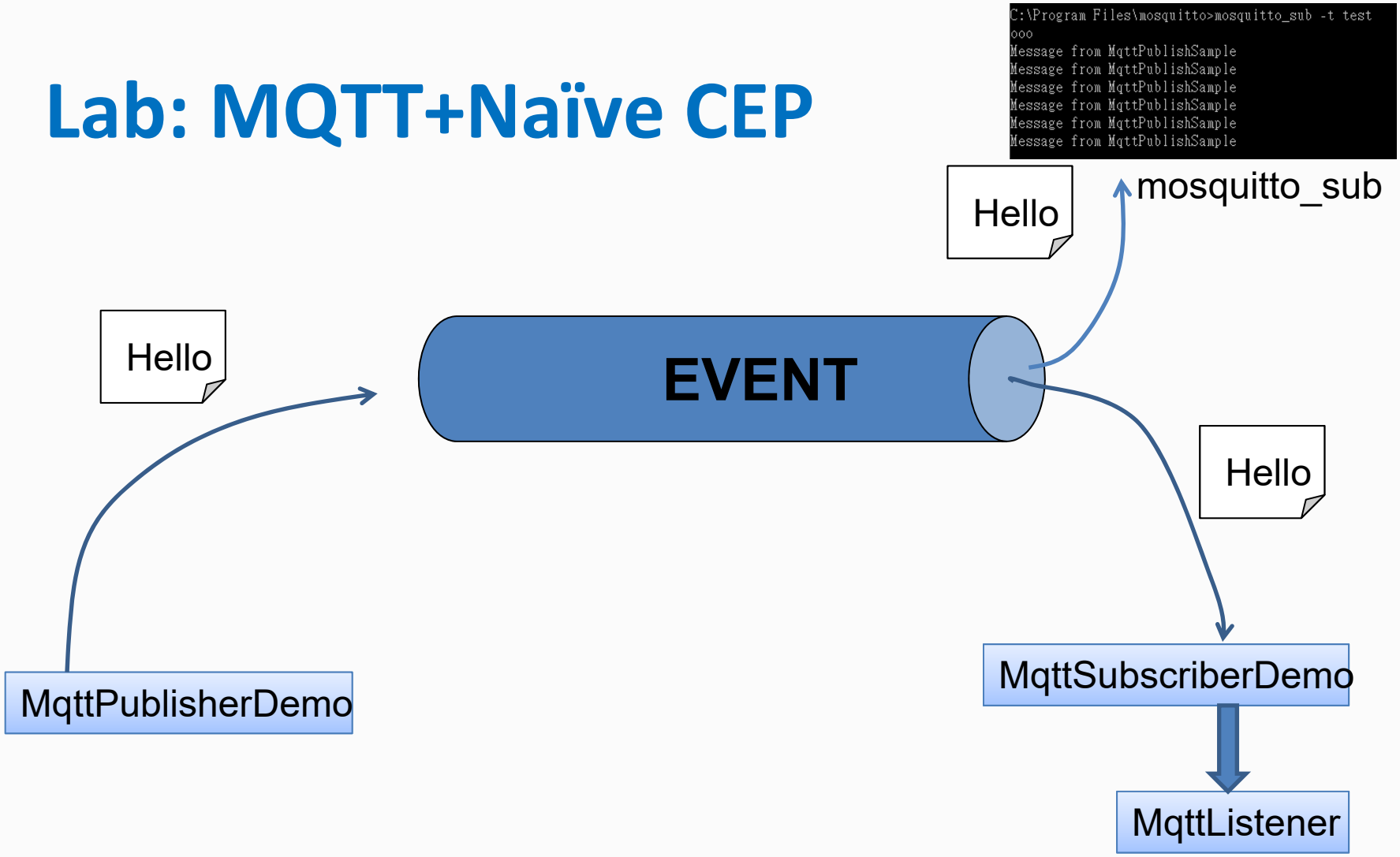
WISE Platform

WISE
- Monitoring and broking messages



Virtual world (Unity)
- Multi-user navigation
- Reactions and effects control

Lab: MQTT+Naïve CEP



Q and A