Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

Indirect Communication (Messaging)

Chun-Feng Liao

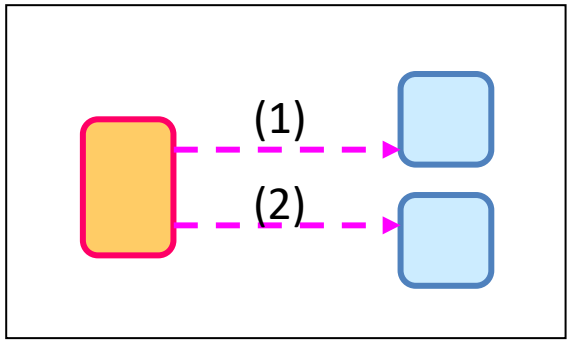
廖峻鋒

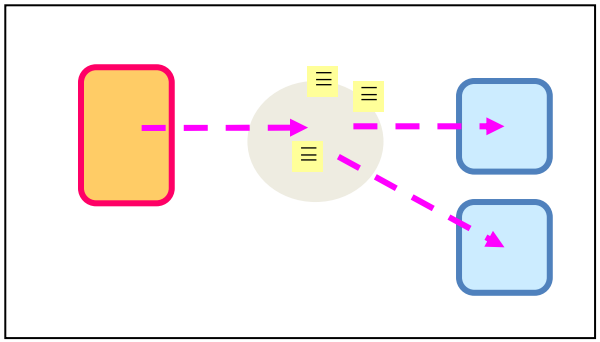
Dept. of Computer Science

National Chengchi University

Basic Remoting Styles

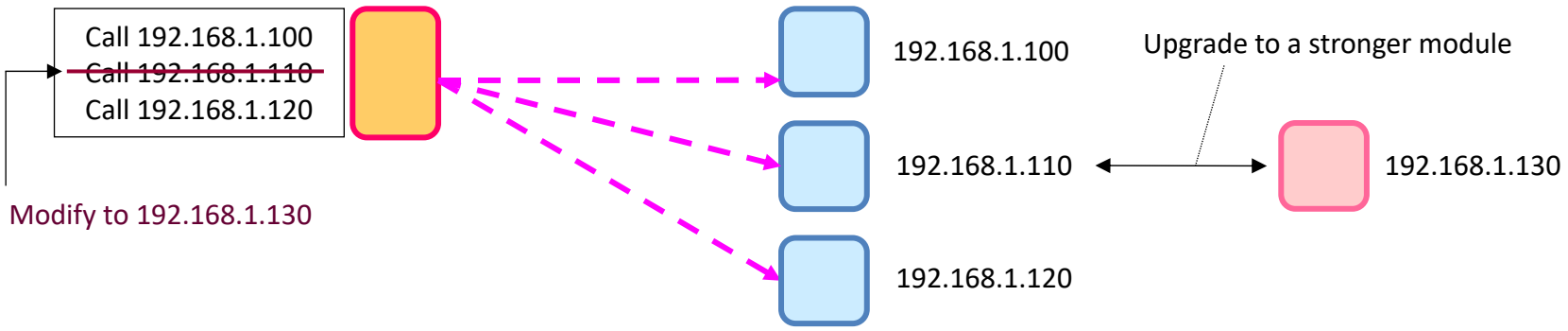
- Two major styles
  - Direct communication (RPC)
  - Indirect communication (Messaging)





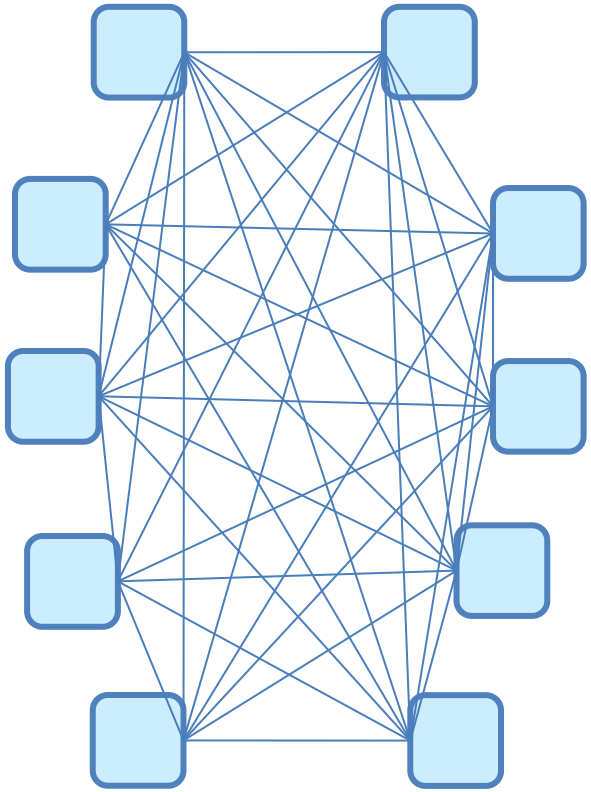
Direct Communication

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



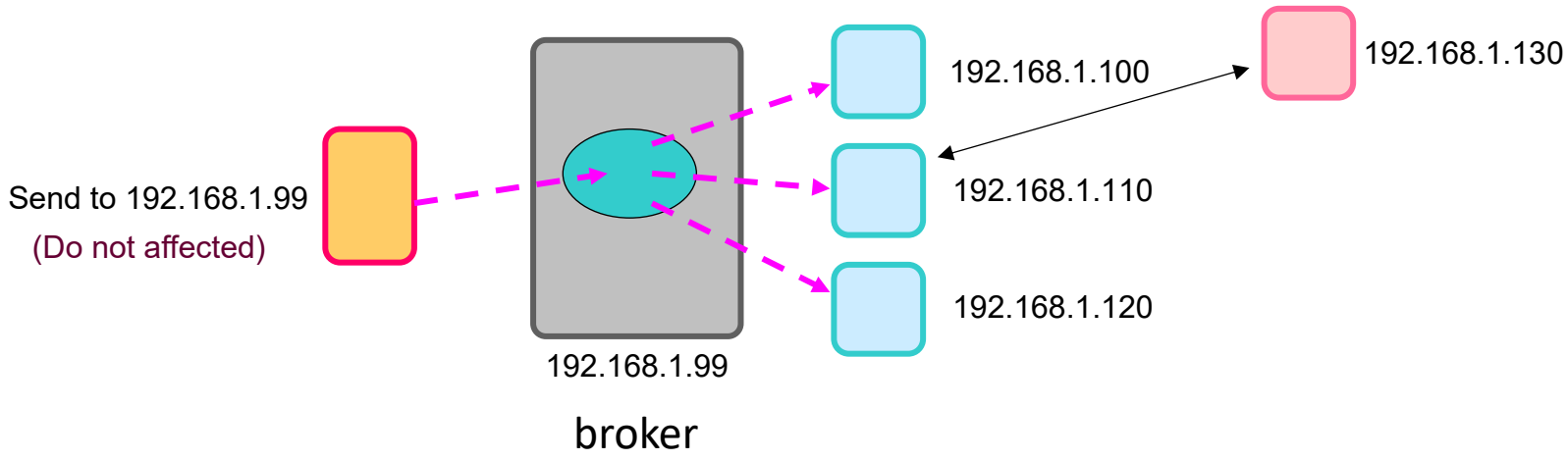
Integrating Systems One-by-One

(n-1)!

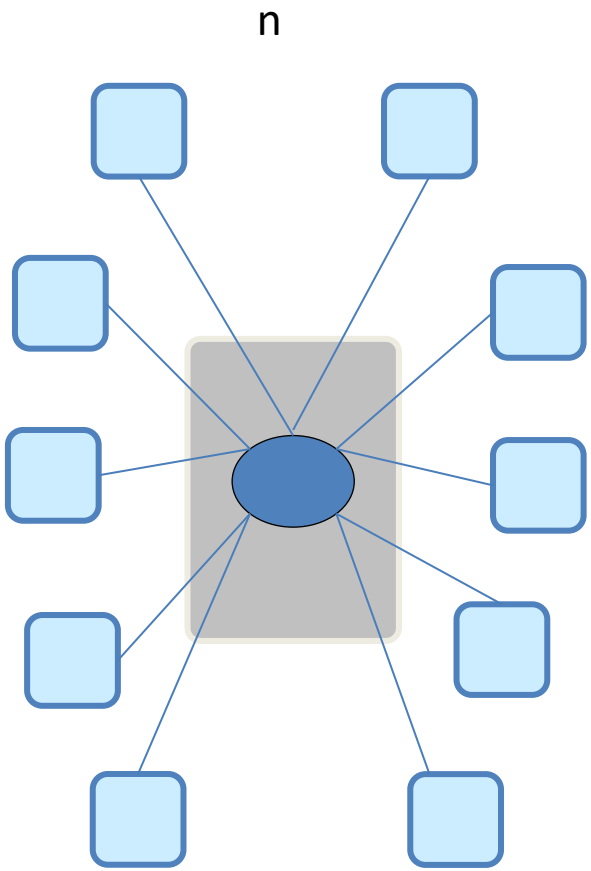


Indirect Communication

- Benefits
  - Decoupling in both space and time
- Drawbacks
  - Single point of failure can be alleviated by clustering
  - Address binding of the broker can be alleviated by broker discovery
    亦可使用brokerless架構處理(後面會介紹)

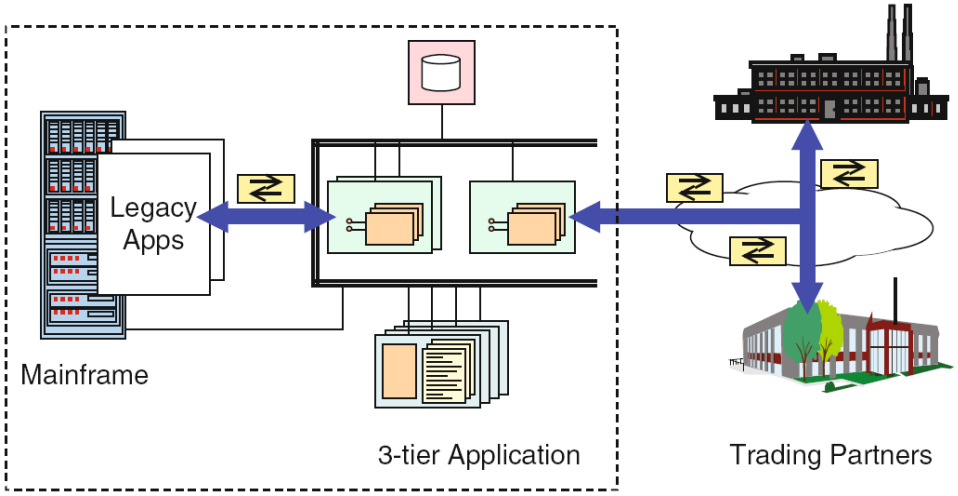


Integrating Systems by Using MOM



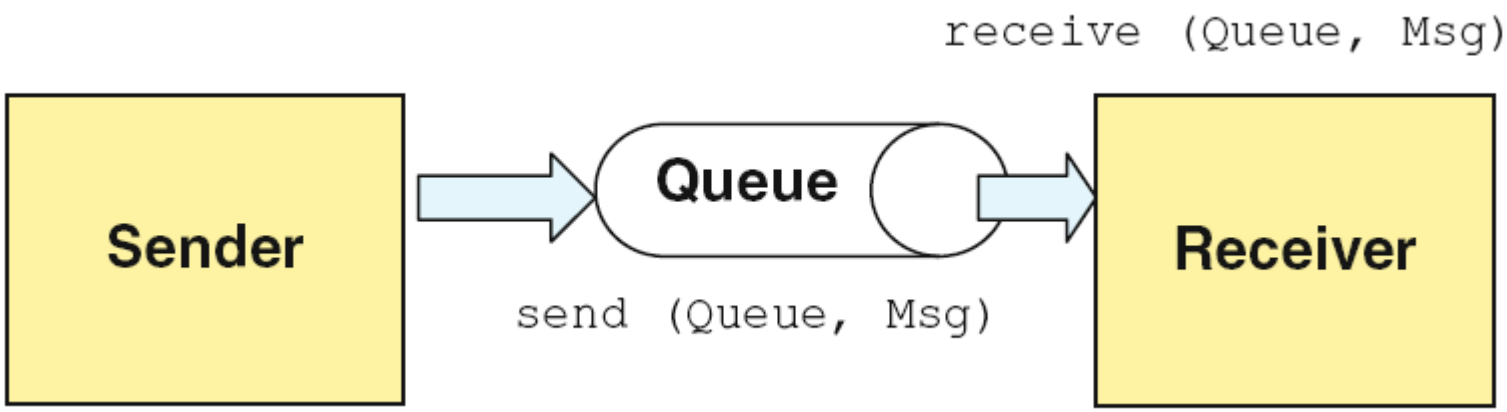
Message-Oriented Middleware (MOM)

- 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

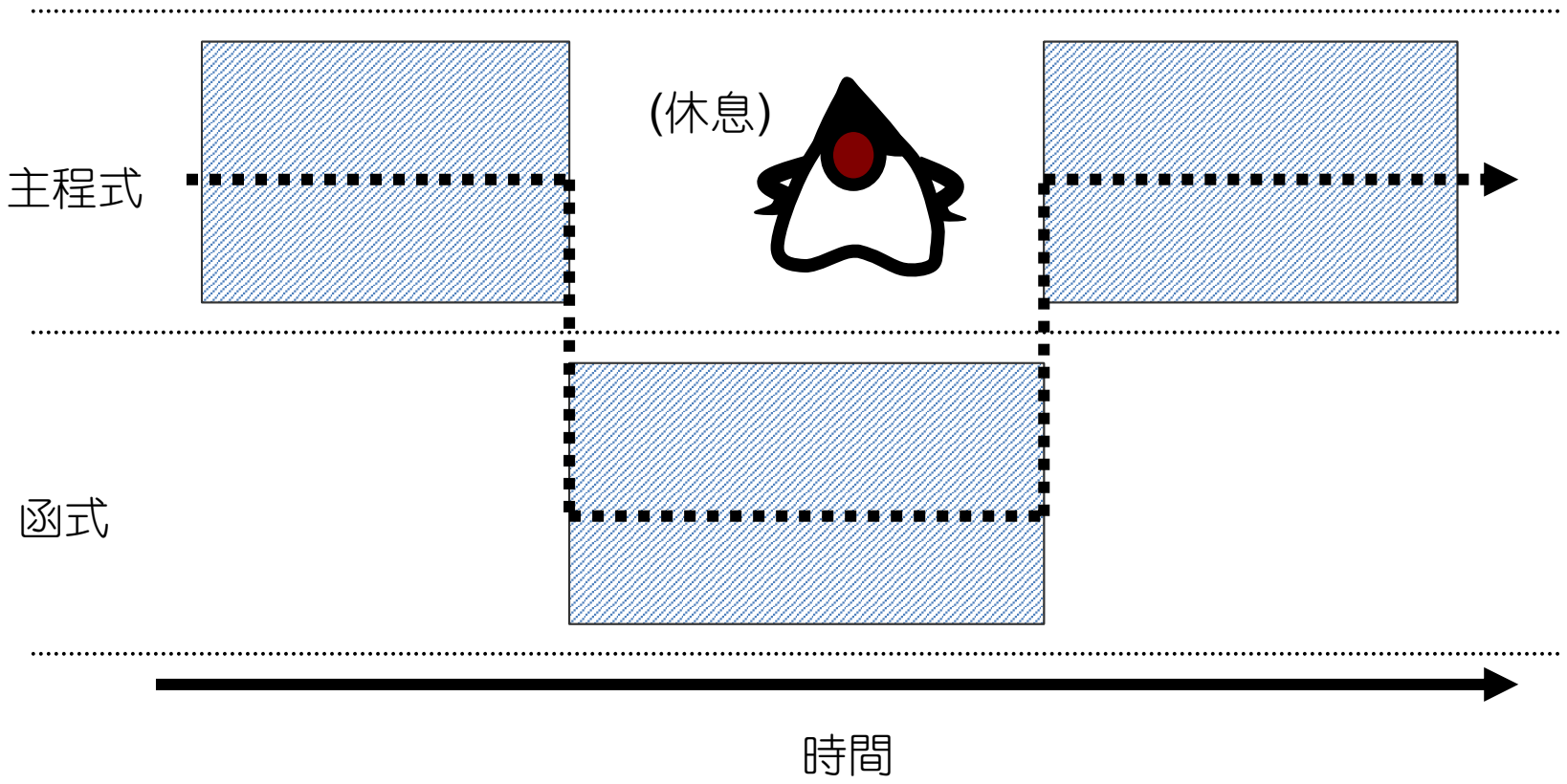


MOM Basics

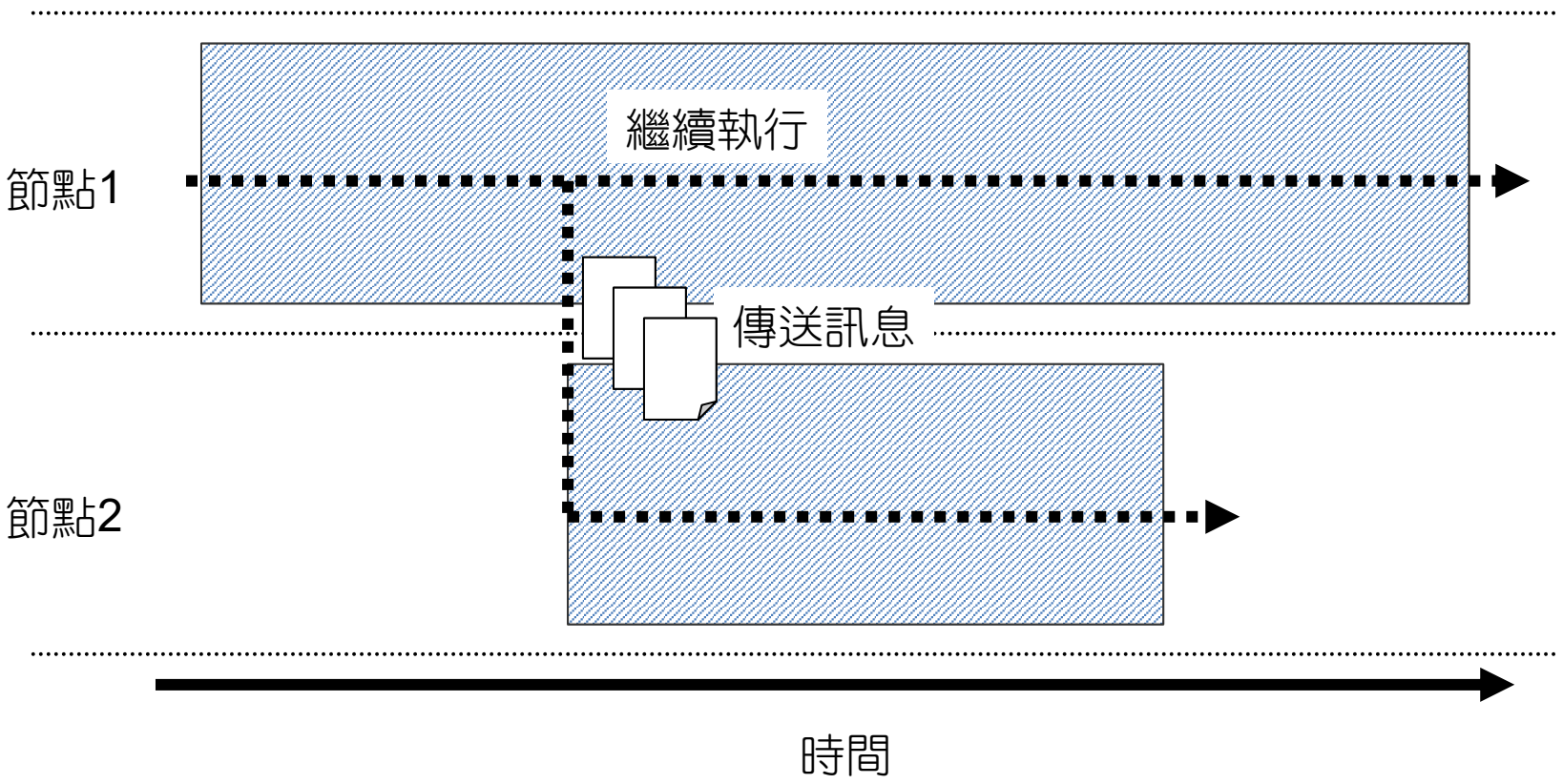
- MOM is inherently loosely coupled and asynchronous
  - MOM decouples senders and receivers using an intermediate queue
  - The sender does not know the actual location of receiver



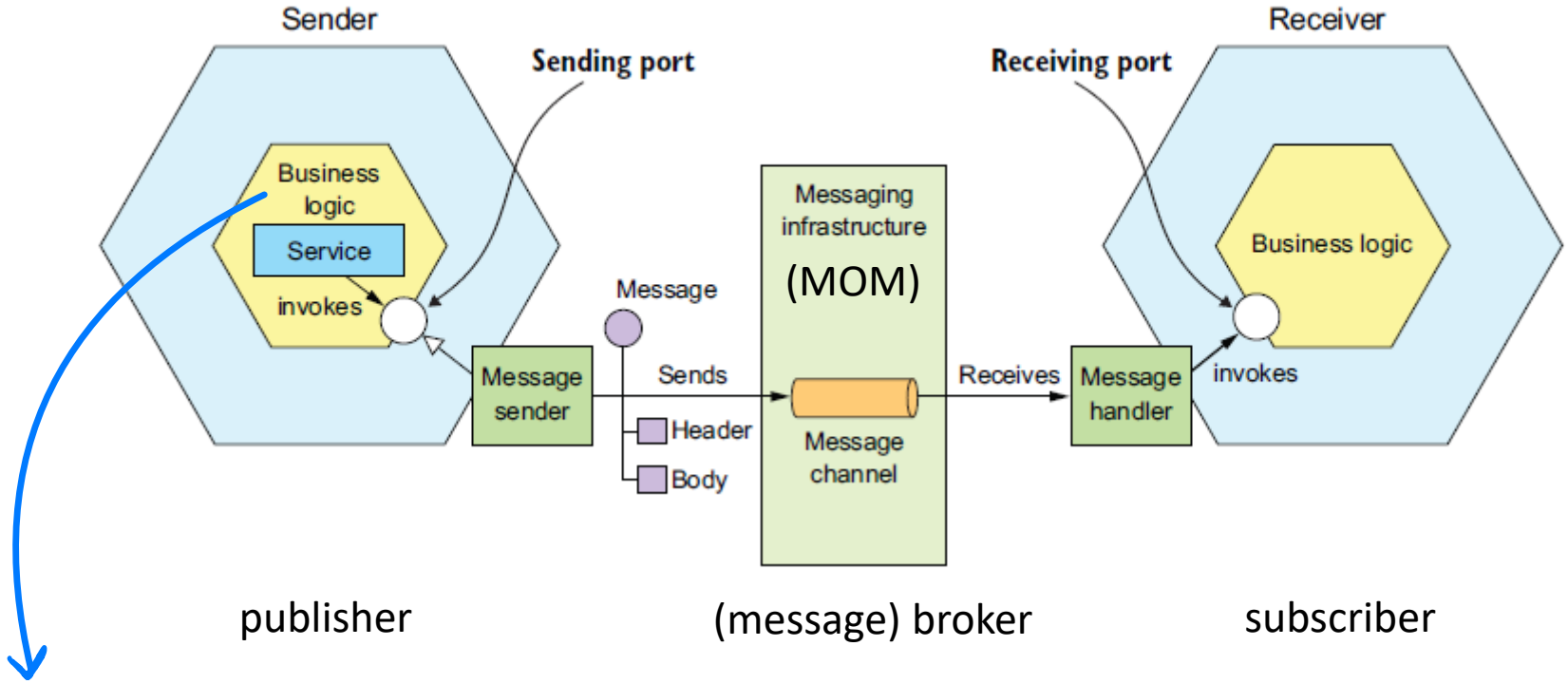
Synchronous Model



Asynchronous Model



Detailed View of MOM



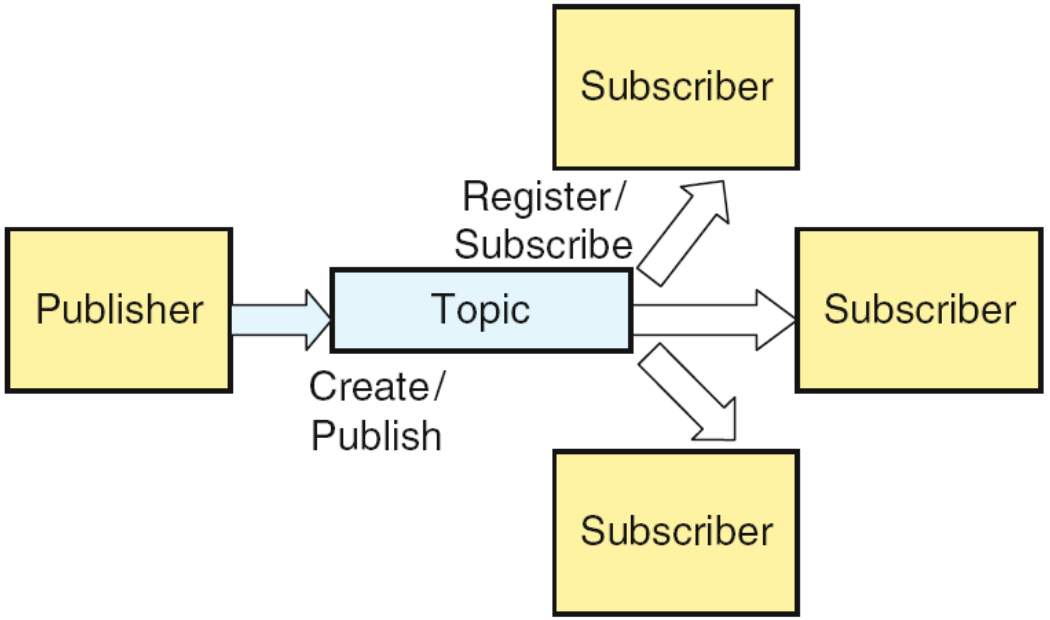
In computer software, business logic or domain logic is the part of the program that encodes the real-world business rulesthat determine how data can be created, stored, and changed. It is contrasted with the remainder of the software that might be concerned with lower-level details of managing a database or displaying the user interface, system infrastructure, or generally connecting various parts of the program.

Message Semantics

- Header
  - Name-value pairs to annotate the message
  - Ex: Metadata, message id, replying channel name
- Body (Payload)
  - Document
    - Ex: return data
  - Command
    - Ex: to simulate RPC
  - Event
    - Indicating that something notable has occurred at a specific time
    - The state change of a domain object at a specific time

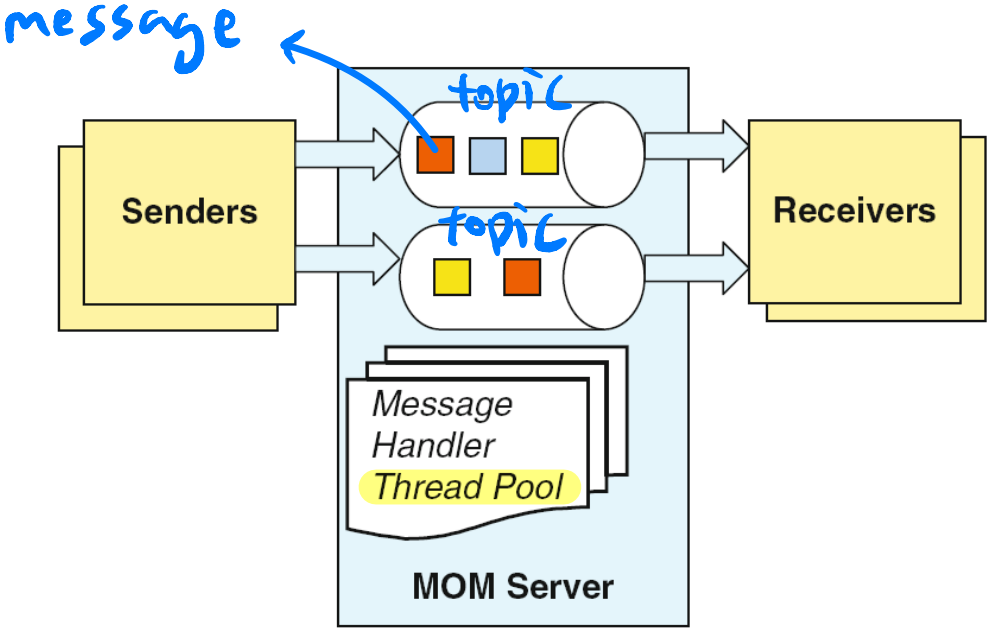
Publish-Subscribe

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



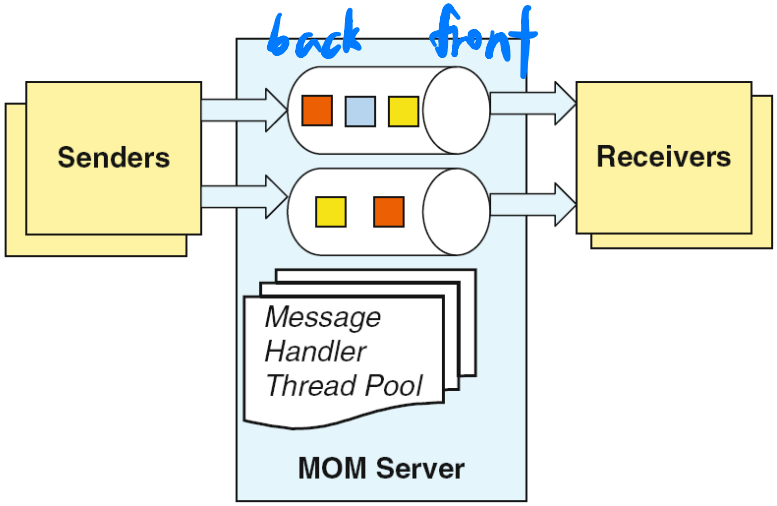
MOM Server (Message Brokers)

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

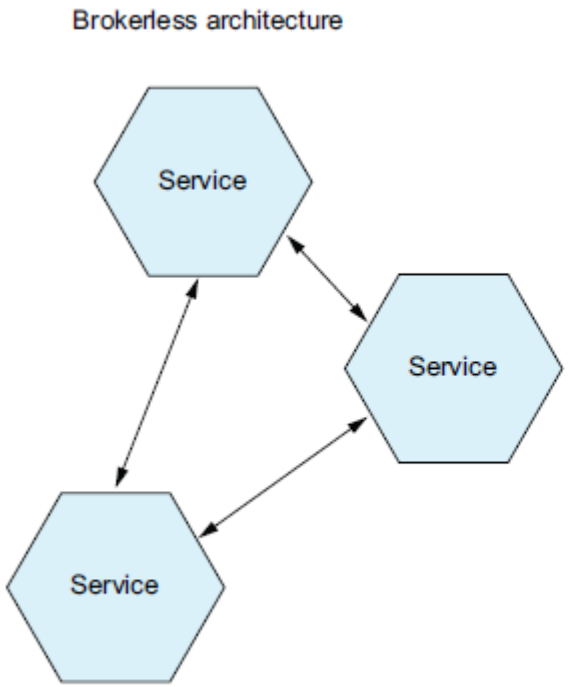


MOM Server (Message Brokers)

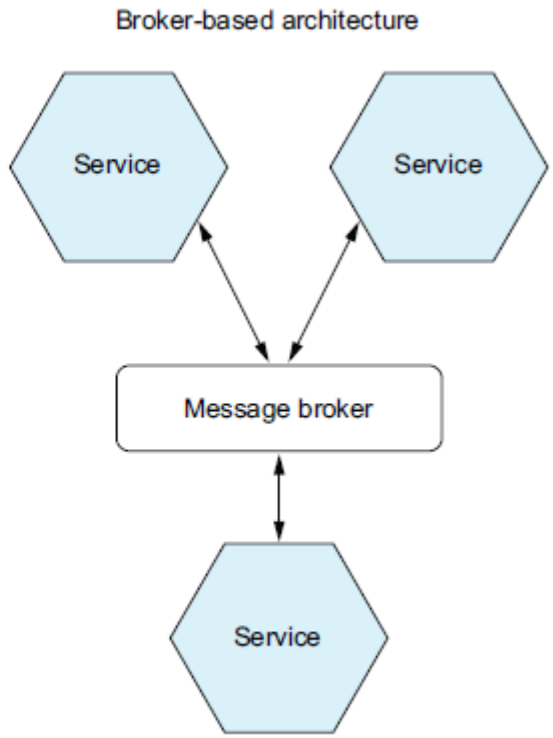
- Message transmitting
  - Accept/Ack messages from the senders
  - Place the messages at the end of the queue (topic)
  - Hold messages for an extended period of time



Two Types of Message Brokers



Vs.

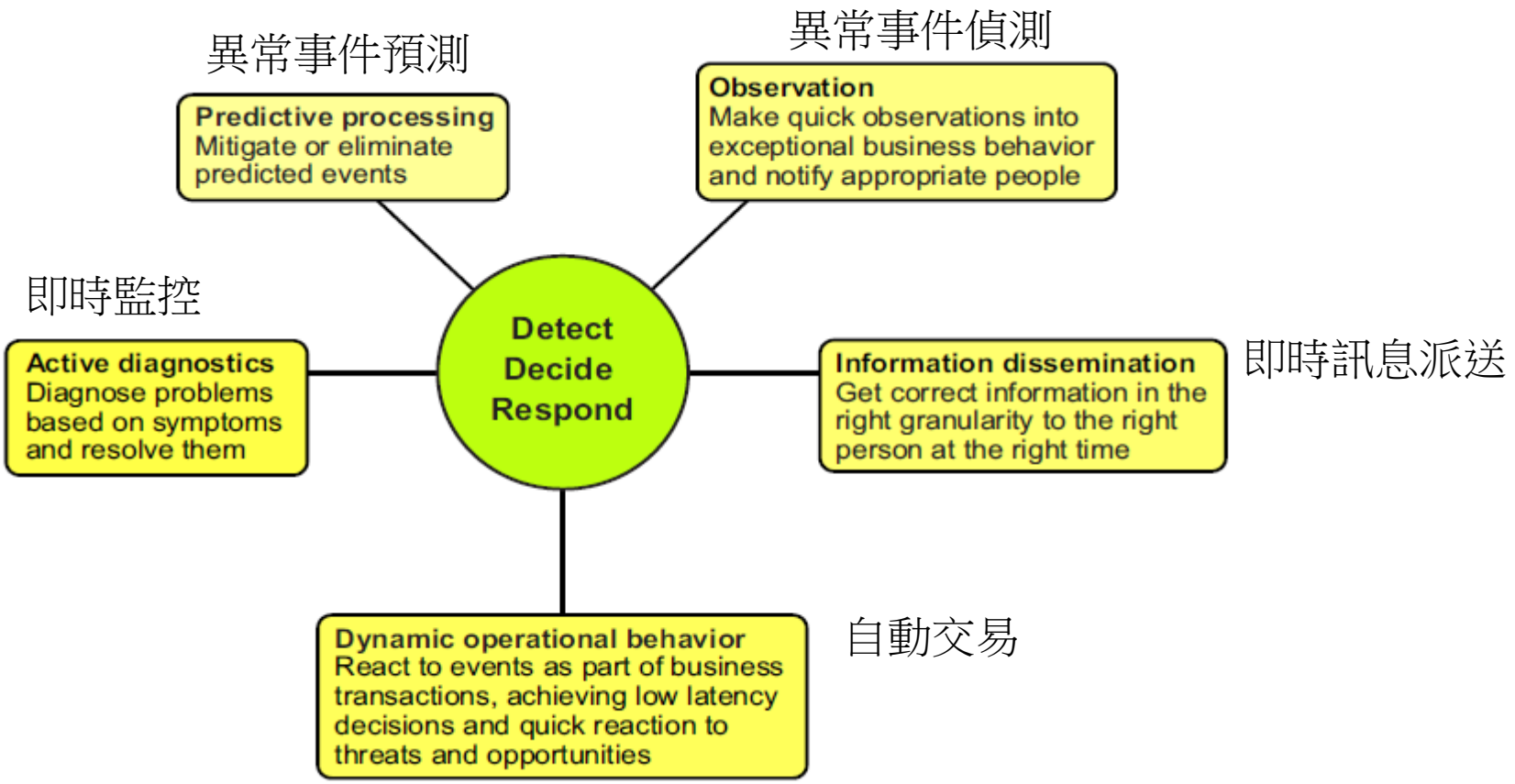


In the broker-less architecture, each endpoint sees the “logical” address (or URI) of the virtual broker

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
    - Use one of the discovery mechanisms
  - Offer reduced availability
    - 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、NSQ、multicast

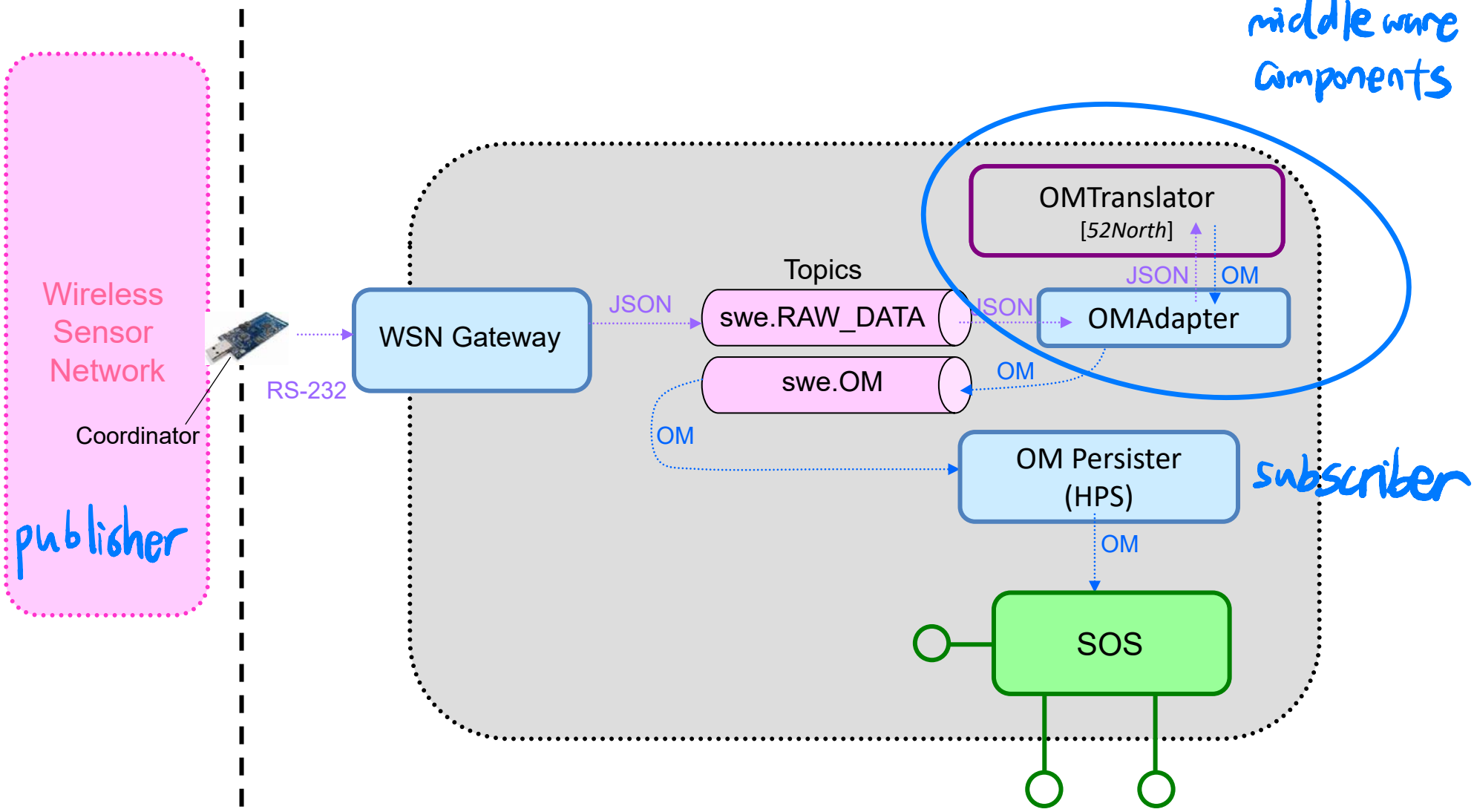
Event Processing Network using MOM



Case:
Sensor Observation Service (OGC SWE)

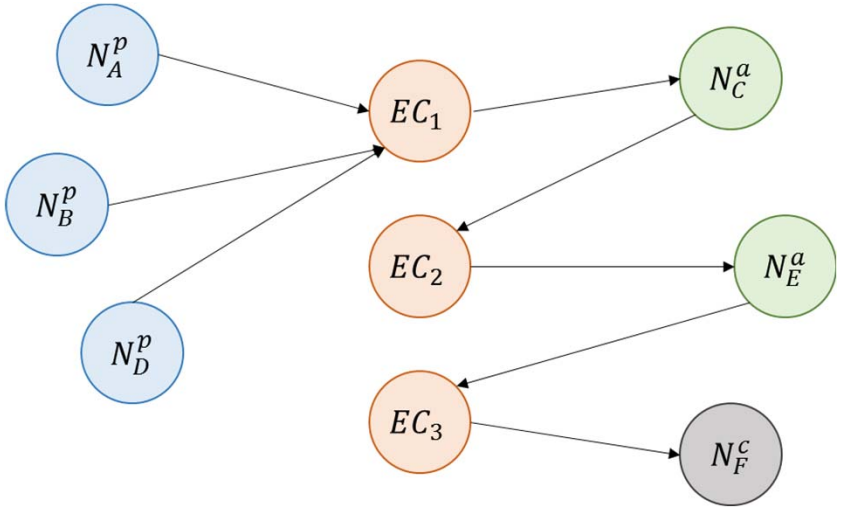
Open Geospatial Consortium

Sensor Web Enablement



Channel-style Event Processing Network

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

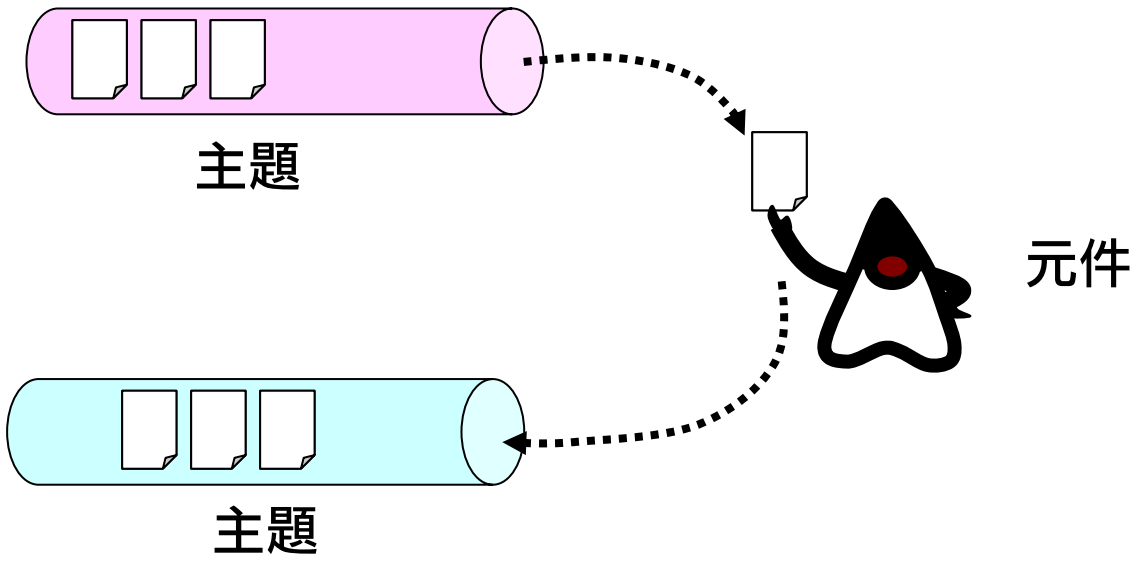


Choreography

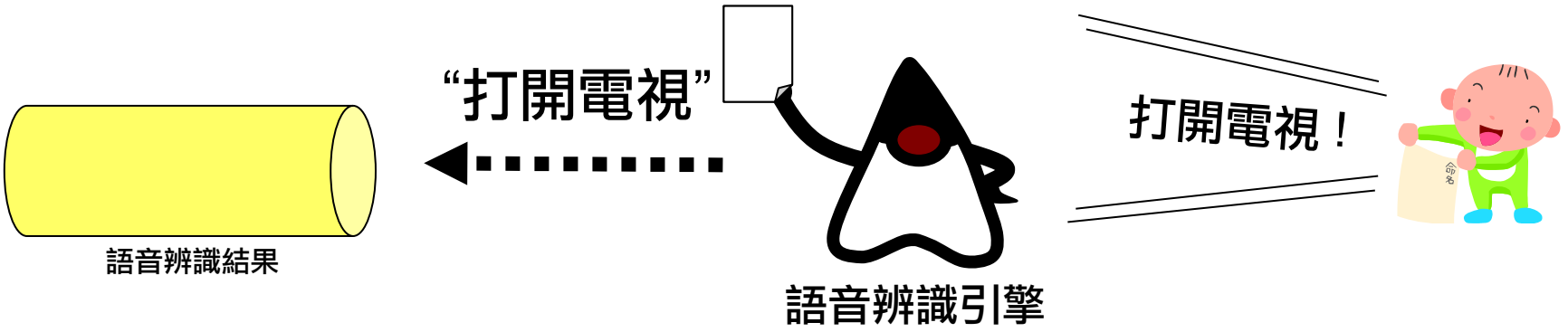
所有元件都只會二個動作:

從一至多個主題(Topic)接收並處理訊息

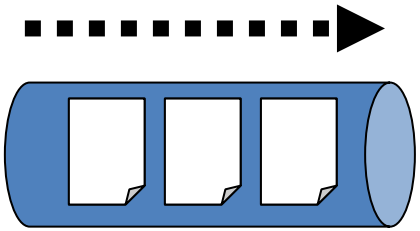
向一至多個主題丟出訊息



語音系統

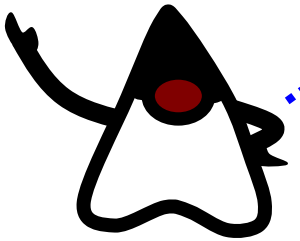


指令對應

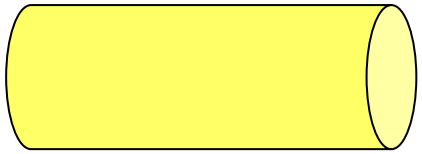


語音辨識結果

打開客廳燈



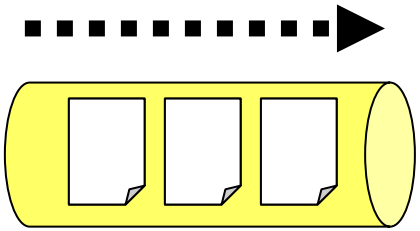
指令對應服務



下達控制命令

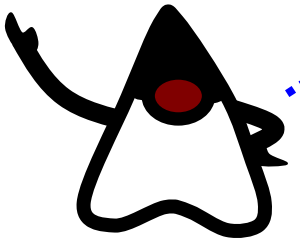
打開電視	TV_ON
關電視	TV_OFF
打開客廳燈	LIVIN DOM- LIGH N
關客廳燈	LIVINGROOM- LIGHT_OFF
…	…

智慧家電及環境控制

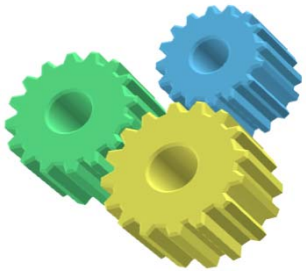


控制命令

LIVINGROOM-LIGHT_ON

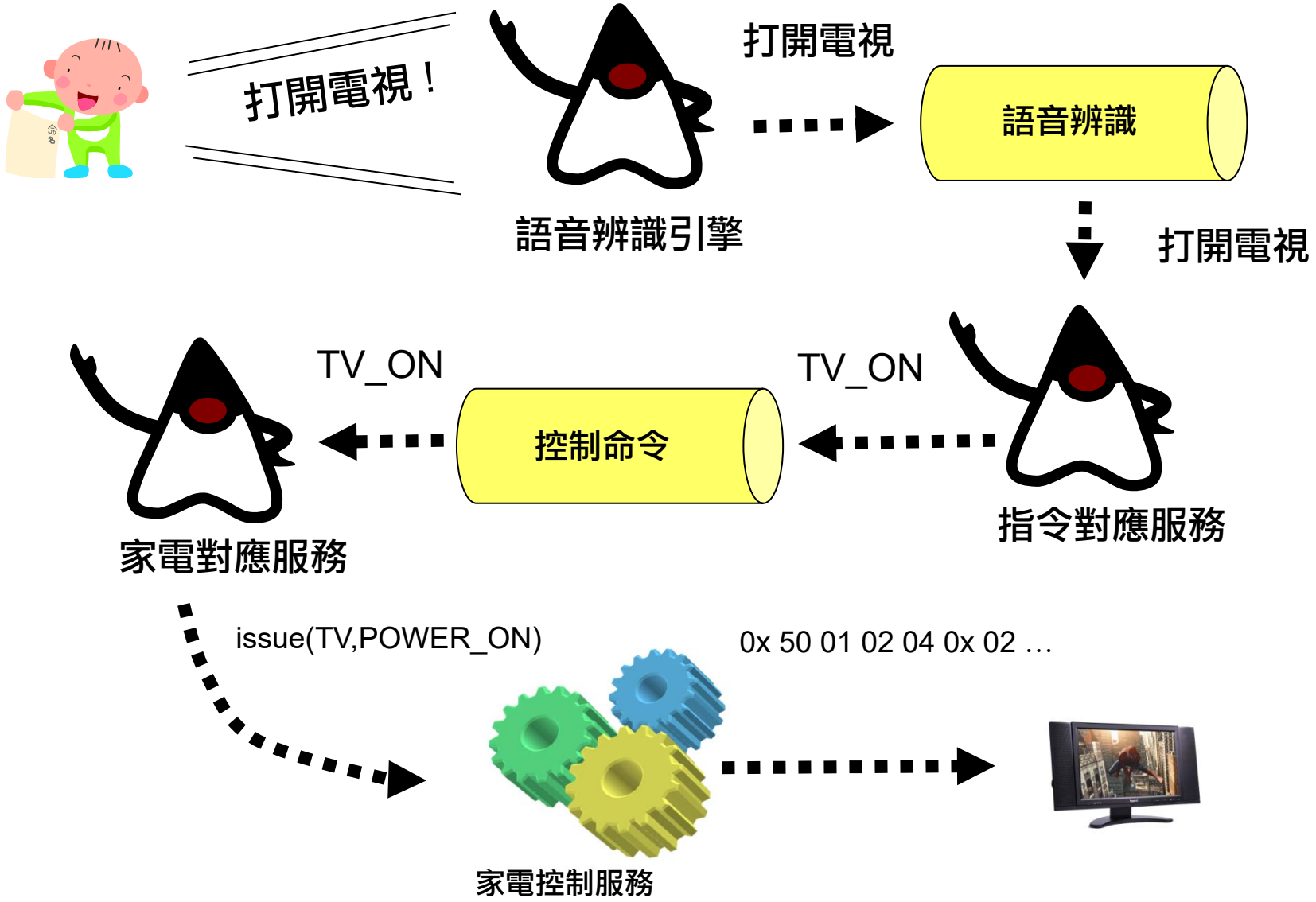


家電對應



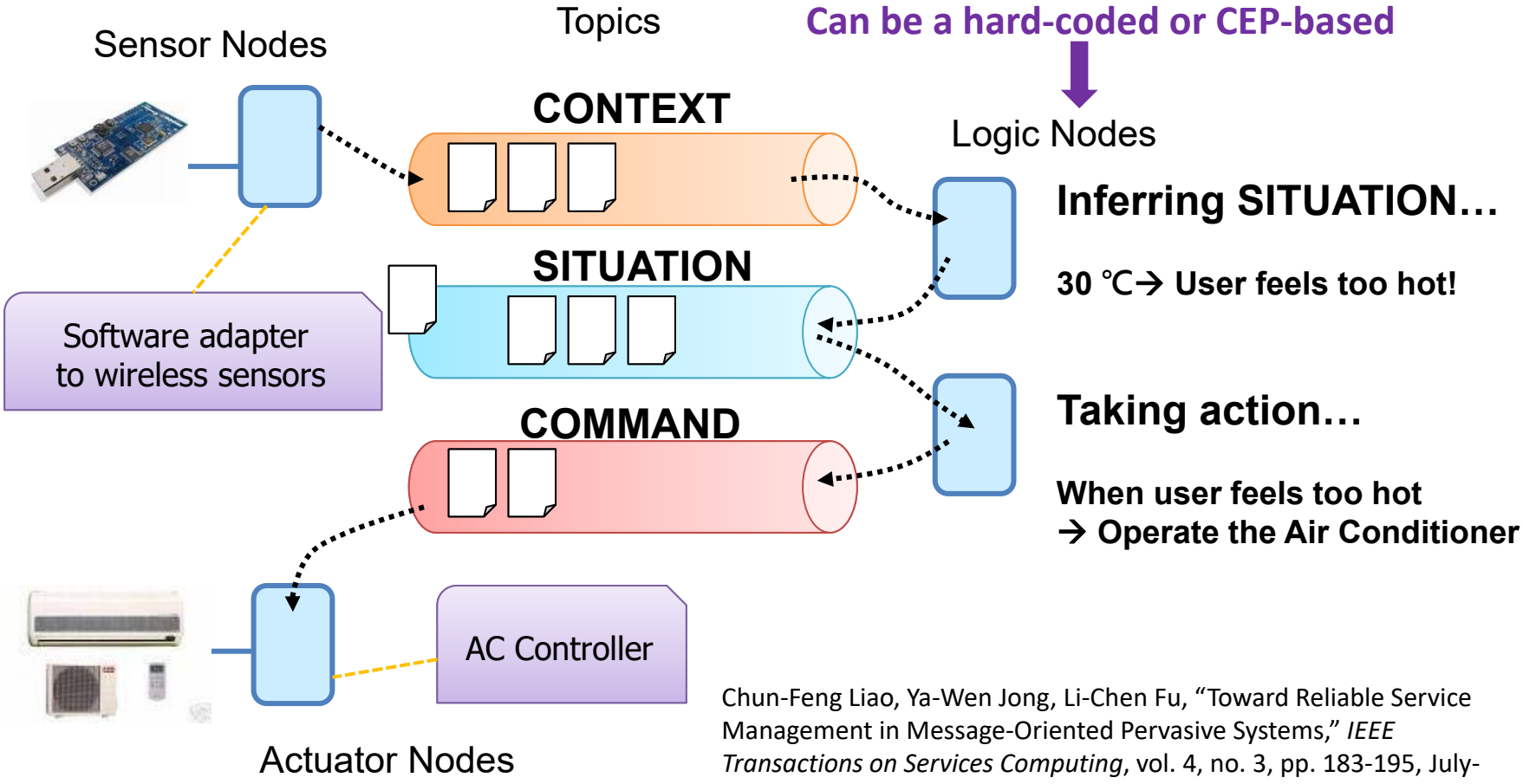
家電控制服務

LIVINGROOM-LIGHT_ON	Hc1[illegible]aryCommand ID=[illegible] 1-6 CMD=ON
LIVINGROOM-LIGHT_OFF	Hc12BinaryCommand ID=1-3, 1-6 CMD=OFF
TV_ON	Hc12TvCommand ID=TV FUN=POWER
…	…


打開電視！
語音辨識引擎
打開電視
語音辨識
打開電視
指令對應服務
TV_ON
TV_ON
控制命令
家電對應服務
issue(TV,POWER_ON)
0x 50 01 02 04 0x 02 …
家電控制服務

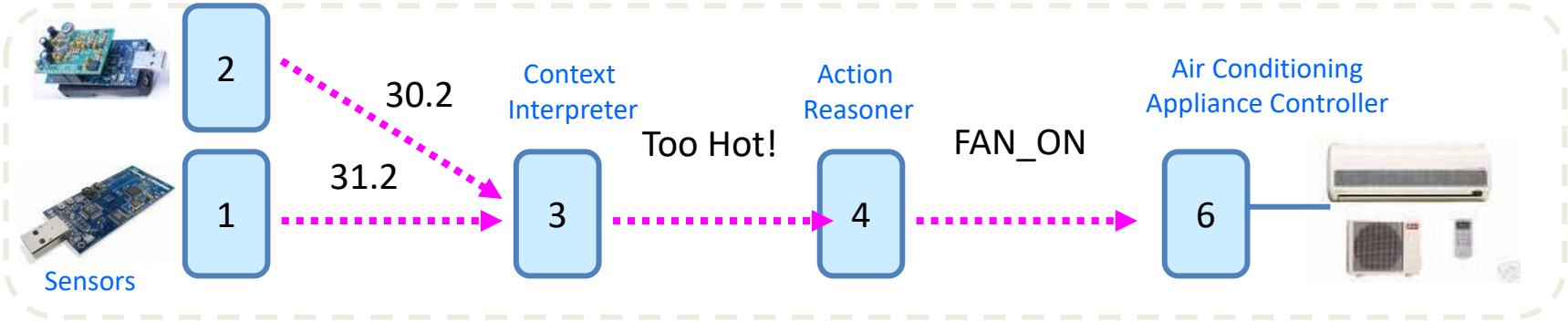
Case: 物聯網系統整合

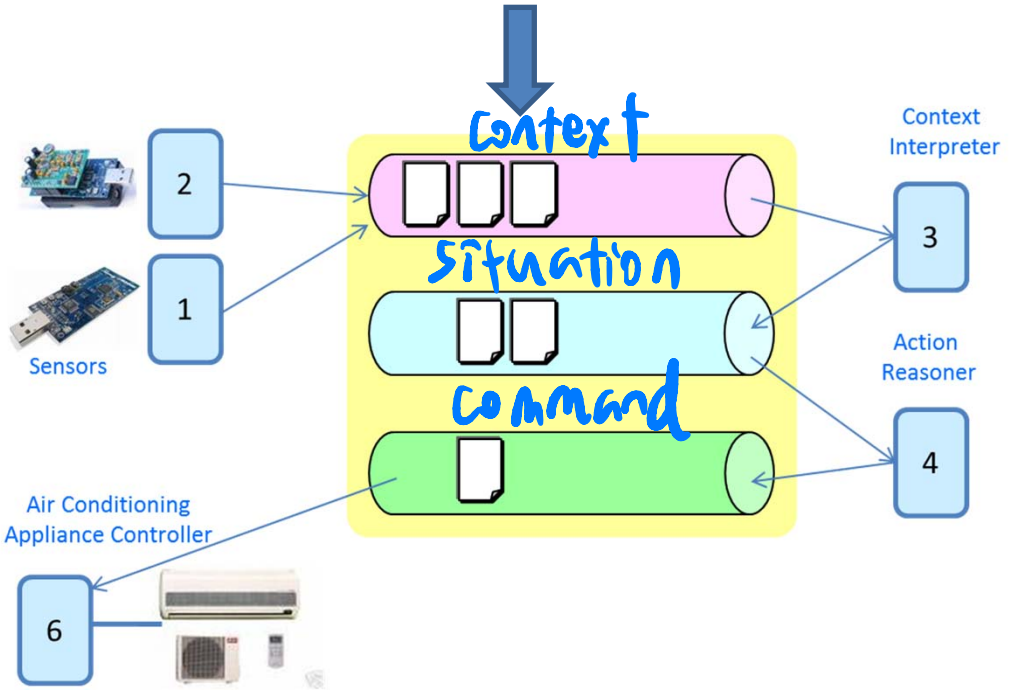
Complex Event Processing



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: 物聯網系統整合

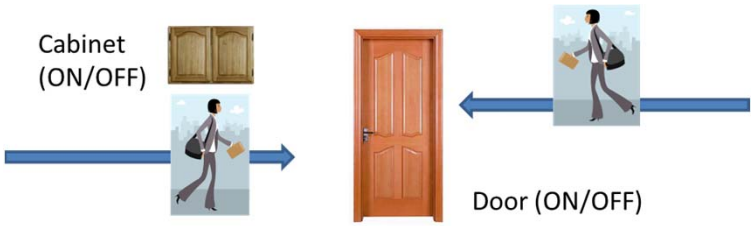




Complex Event Processing (CEP)

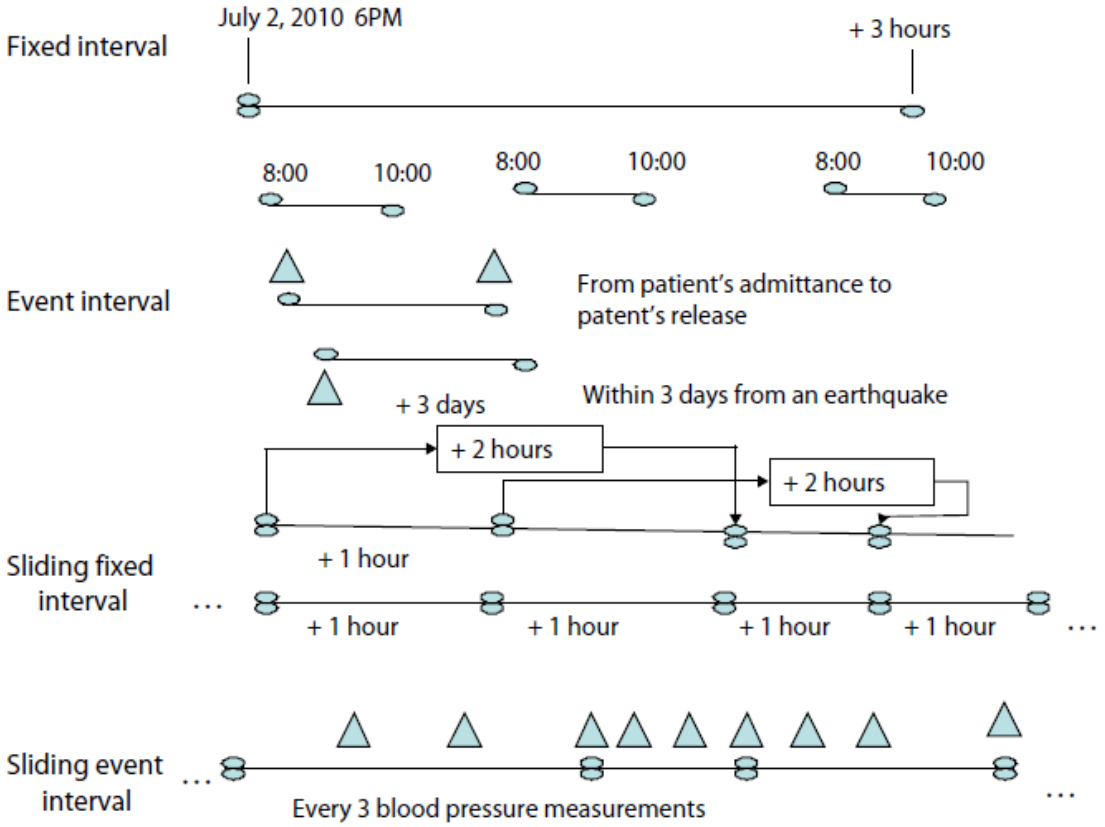
- 定義事件前提(Context)規則
  - 事件符合或不符合某些外在環境條件
- 目的
  - 忽略
    - 若不符合此前提則EPA將不予處理
  - 處理
    - 透過事件處理前提的過濾機制，EPA可取得並處理特定事件
  - 衍生
    - 符合特定條件後，衍生更多事件

Event Processing Agent



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

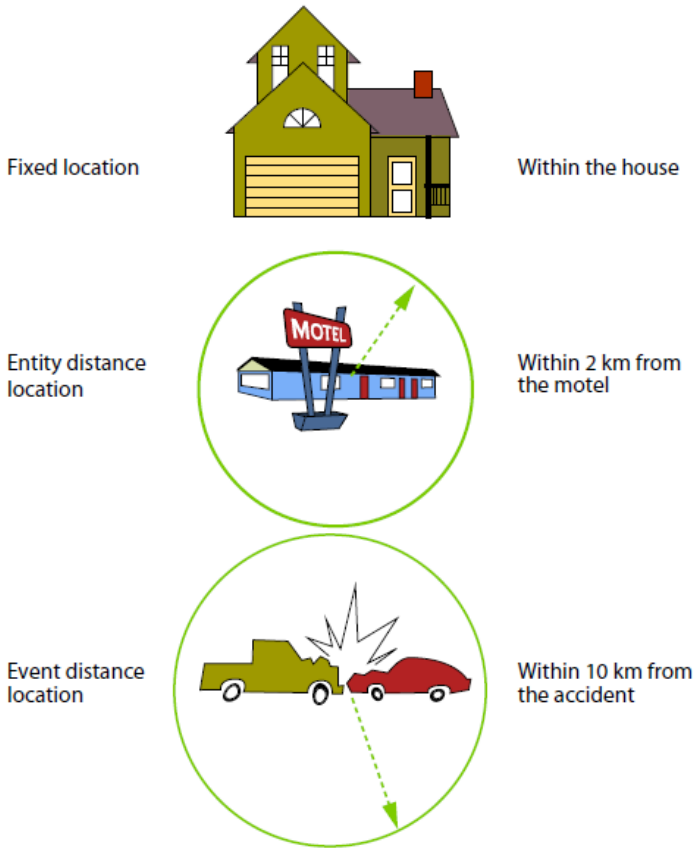
以事件發生區分

以相對時間區分

以事件發生次數區分

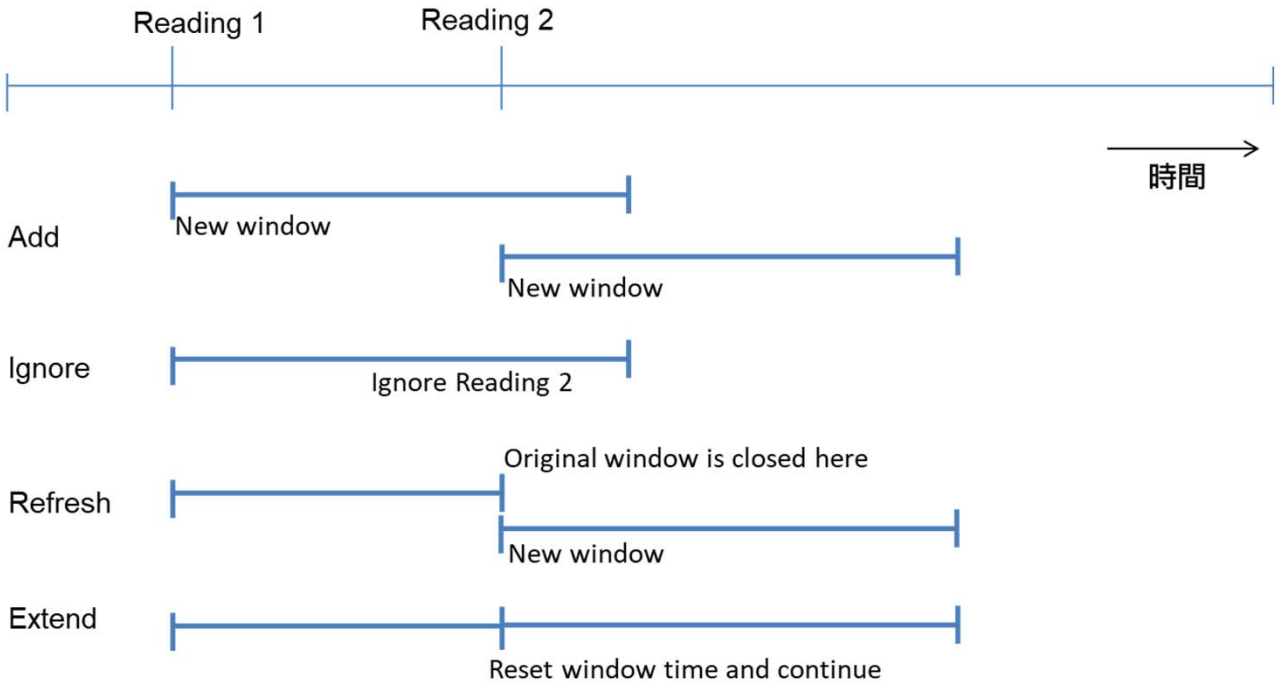
Spatial Context

- 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

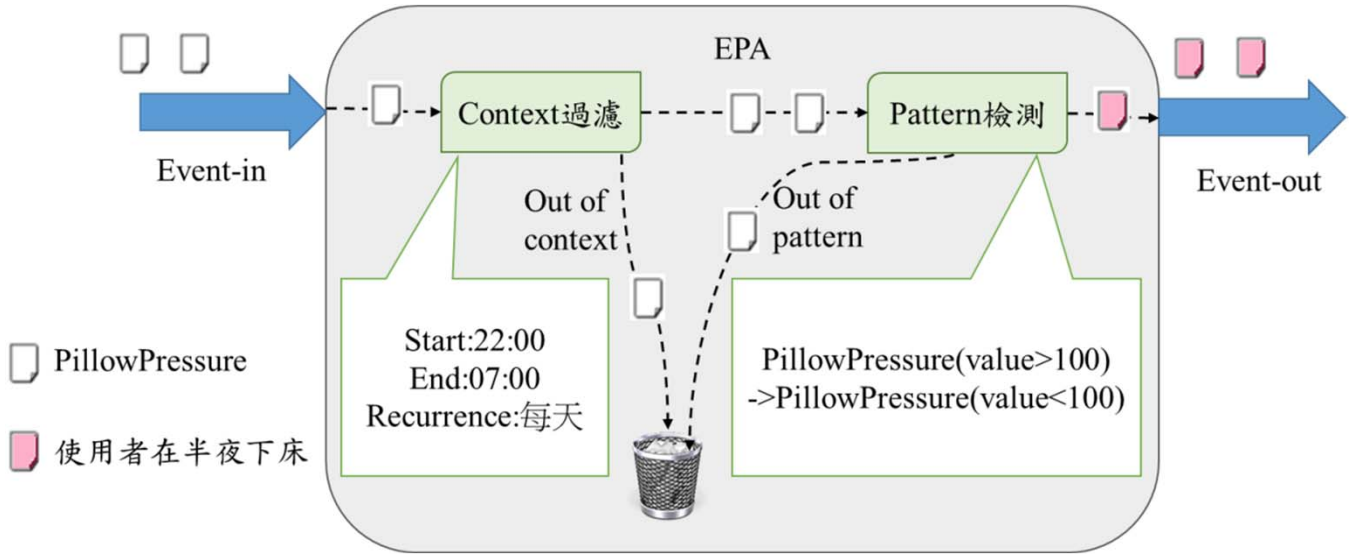
連續二個相同類型event進來時該如何處理?



例如，室溫大於28度→很熱所以開冷氣1小時，如果10秒間取得二個讀值28度以上，如何解讀?
Add: 開二次
Ignore: 只有第一次會開
Refresh: 先關再開
Extend: 從第二次到時，原來計算的時限(1小時)重設

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

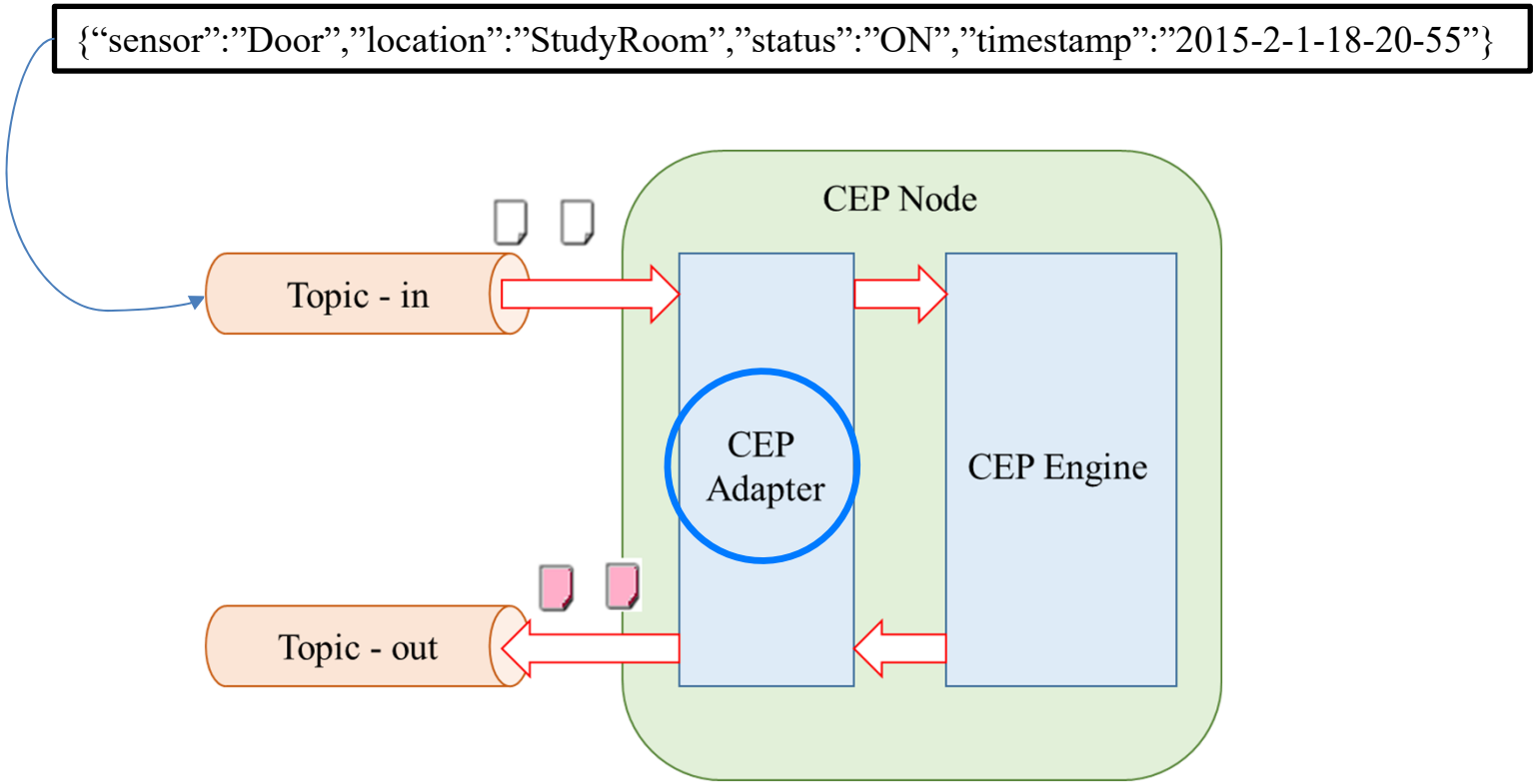
- 當使用者在晚上10:00至早上7:00時進入浴室後20分鐘未回到床上時，發出警告訊息
  - 定義Context為重複的固定時間間隔
    - CREATE CONTEXT NightContext start 22:00 end 07:00
  - 定義Pattern為浴室門感測器狀態由ON轉變至OFF，並且20分鐘內未發生床的壓力感測器數據>100
    - BathroomDoor(status = 'ON') -> BathroomDoor(status='OFF')

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

去廁所超過20分鐘還沒回來

CEP+MOM

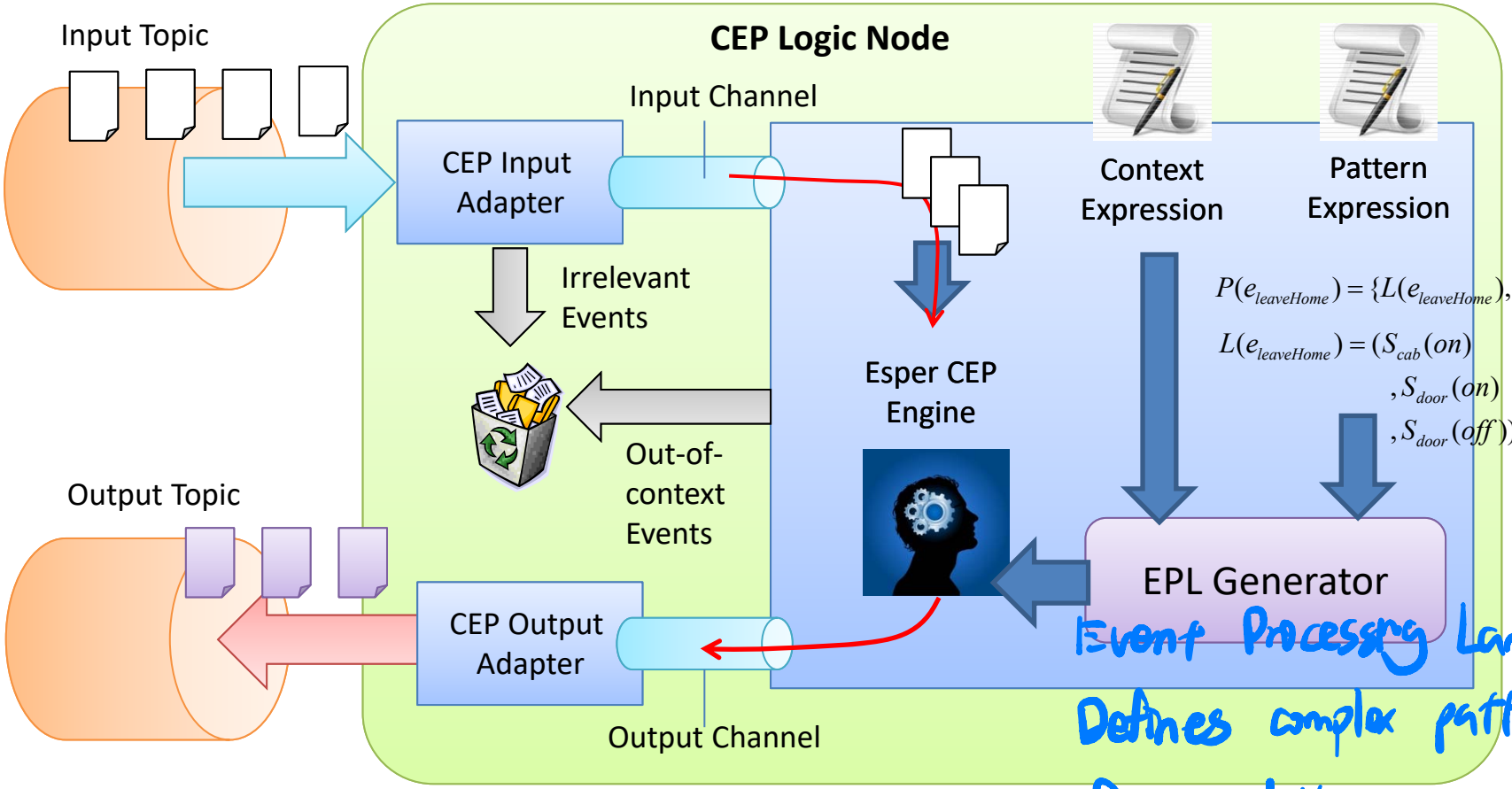
- 感測器數據透過CEP Adapter轉換成CEP引擎所接受之事件格式



Architecture of a CEP + MOM

defines the context under certain patterns should be matched

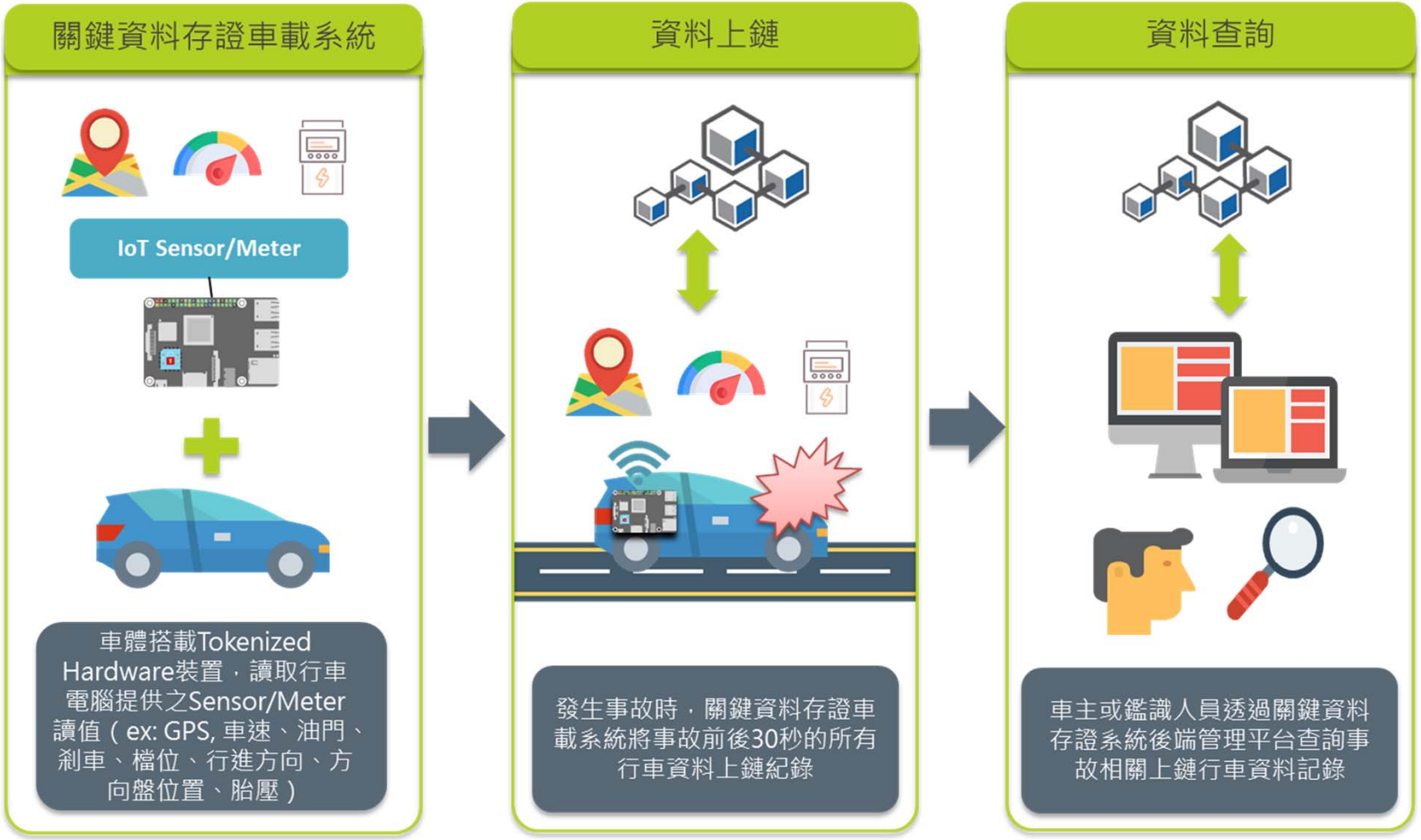
$$K = \{t_6^9, \lambda_{S \subset FrontDoorArea}\}$$



defines patterns of events to match

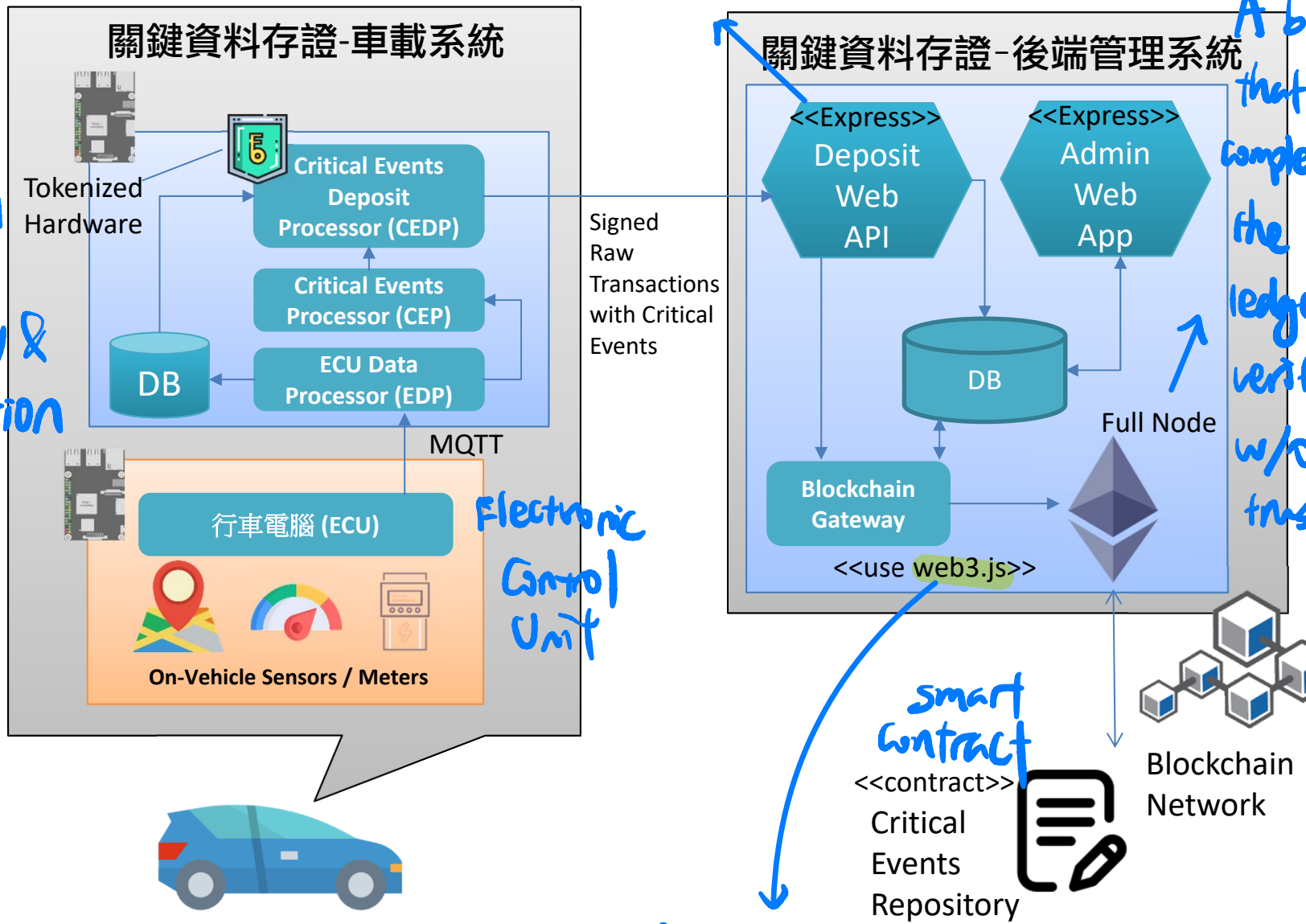
Event Processing Language: Defines complex patterns & conditions

案例: 事故存證



工研院，基於區塊鏈技術的車聯網關鍵資料存證服務規劃與開發 (2021)

A web application framework for Node.js ⇒ build web apps & APIs



authentication e.g. ensures data integrity & non-repudiation

Electronic Control Unit

A blockchain node that contains a complete copy of the blockchain ledger and can verify transaction w/o needing to trust other nodes.

smart contract

A collection of libraries that allow you to interact with local or remote Ethereum node, to communicate with the blockchain network.

常見的開源CEP與MOM

- CEP Complex Event Processing
  - EsperTech
    - https://github.com/espertechinc/esper
  - Apache Flink
    - https://flink.apache.org/
- MOM Message-Oriented Middleware.
  - MQTT (協定)
    - Mosquitto、moquette
  - AMQP (協定)
    - RabbitMQ、Apache Qpid
  - Kafka
    - https://kafka.apache.org/

MQTT

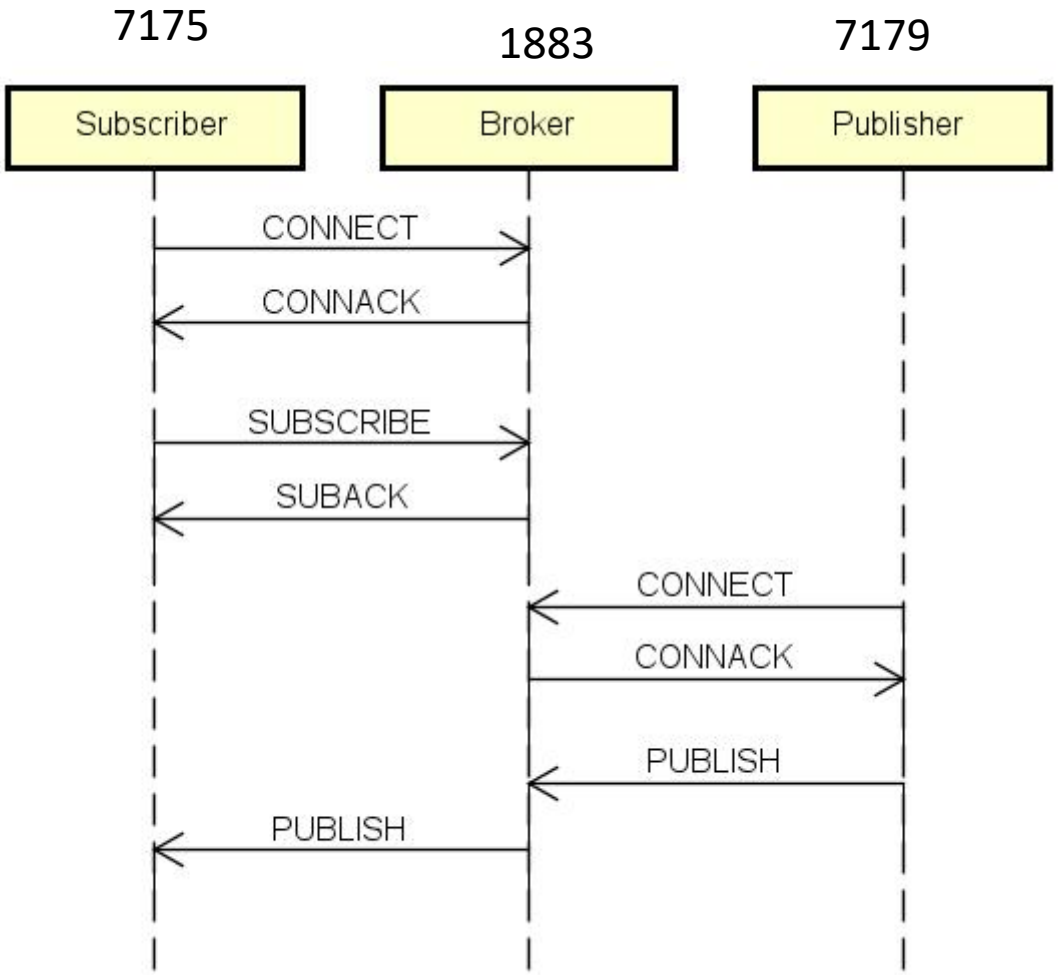
- Message Queuing Telemetry Transport
  - Proposed by IBM and Eurotech
  - Based on TCP/IP (MQTT-SN can use UDP)
- Compact
  - Designed for IoT & M2M communication : lightweight
    - The most popular MOM for IoT
    - Low bandwidth and computing capability requirement
- Know uses
  - Facebook Messenger, Amazon IoT, OGC, Azure IoT Hub

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

PUBREC is "Publish Received" and is sent by the receiver to acknowledge that it has received the message.
PUBREL is "Publish Release" and is sent by the sender after receiving PUBREC, to ask the receiver to release the message to the application.
PUBCOMP is "Publish Complete" and is sent by the receiver to confirm that the message was released to the application, completing the QoS 2 protocol exchange.

訊息傳送基本流程 (QoS0)

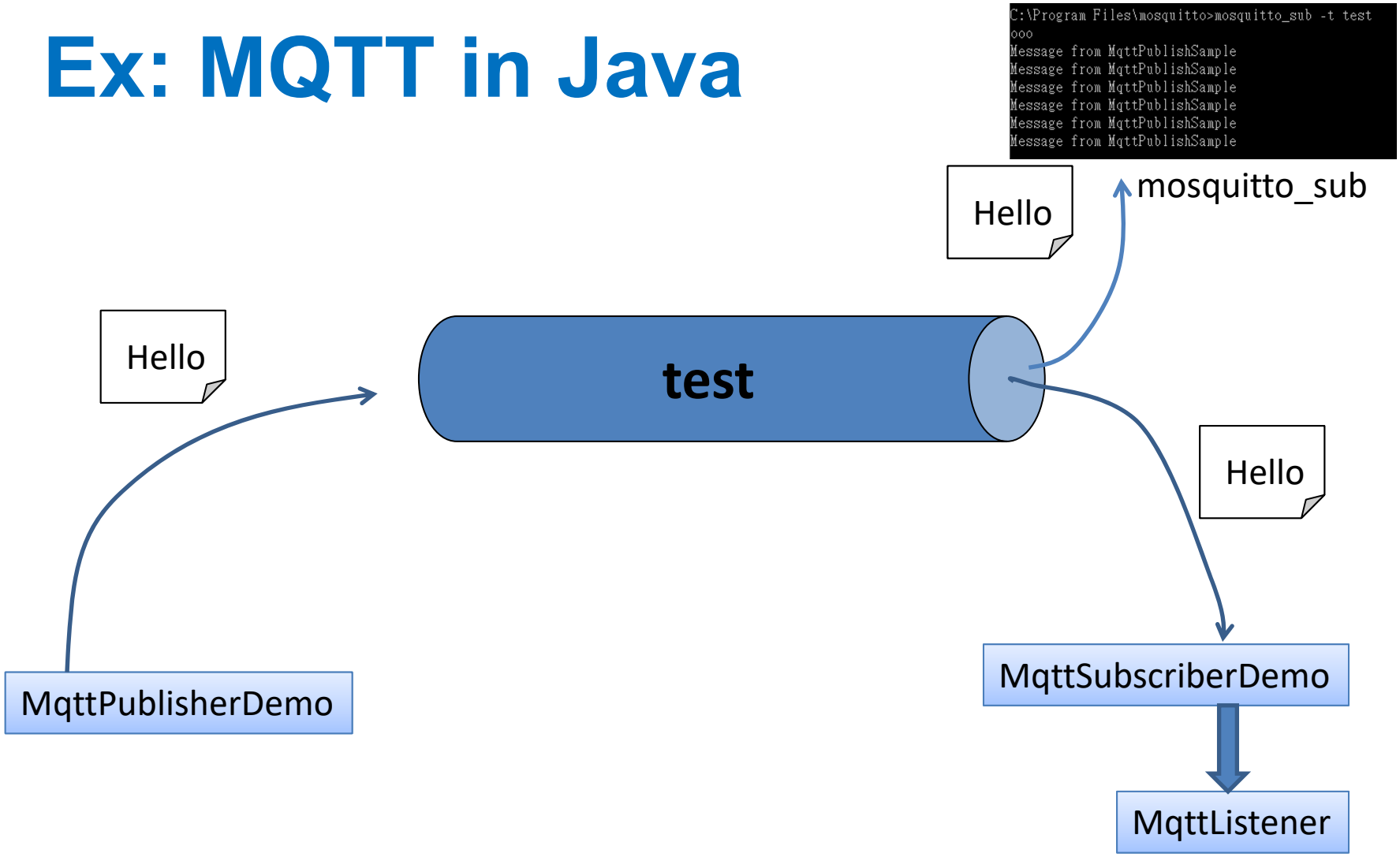


封包觀察

- See wireshark

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

Ex: MQTT in Java



Java MQTT Publisher

- Connect

used to persist message until they are succesfully published

```
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);
```

Set Quality of Service

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();
```

see the next slide

) wait for connection to complete

) wait for subscription to complete

MQTT clients use this to handle events triggered by the MQTT broker, e.g. when a message arrives

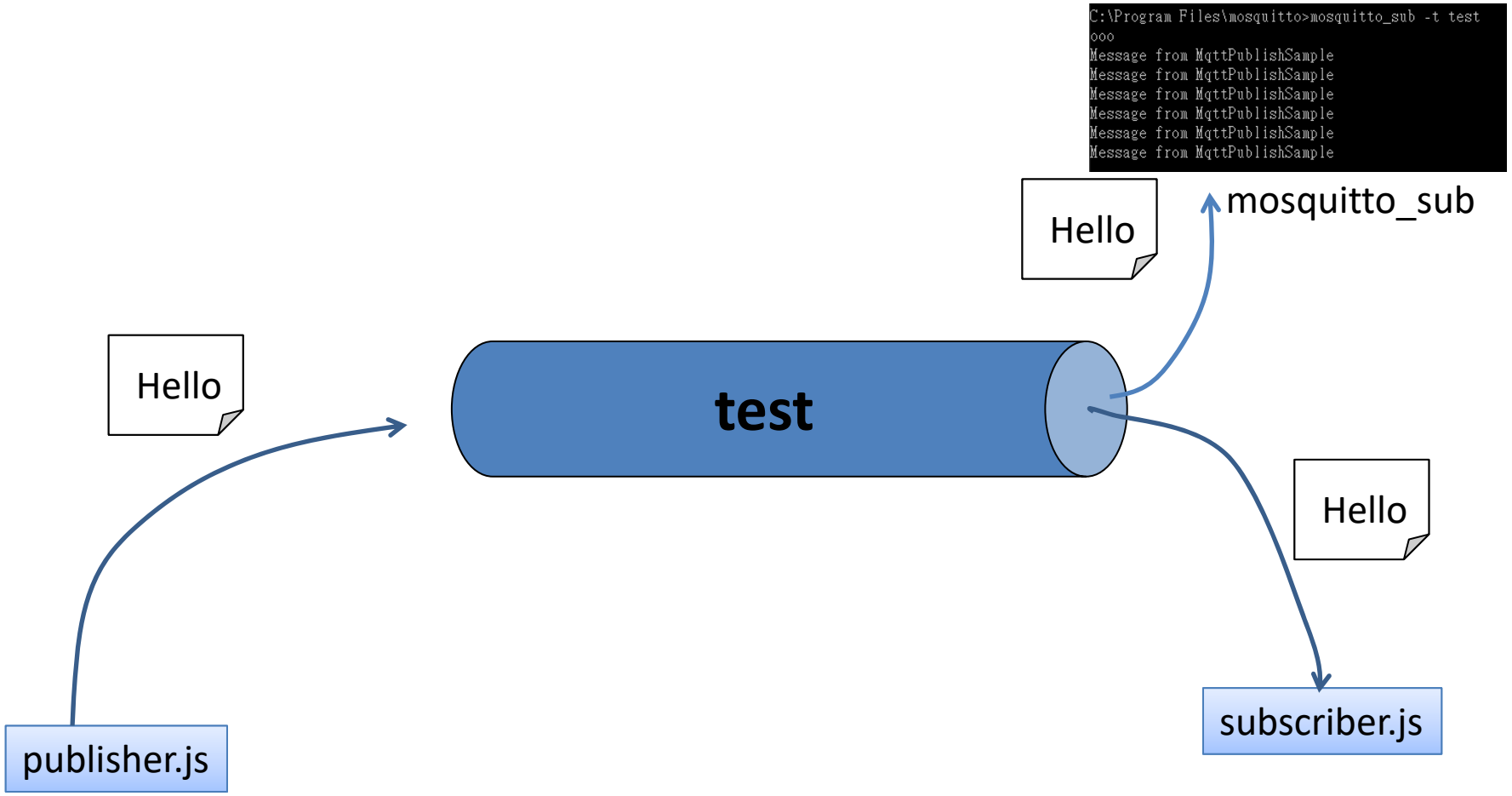
```
public class MqttListener implements MqttCallback {
        ...
    @Override
    public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
        System.out.println(mqttMessage.toString());
    }

}
```

Interface

print the received message to the console

Ex: MQTT in JavaScript



JavaScript (Node.js)

publisher

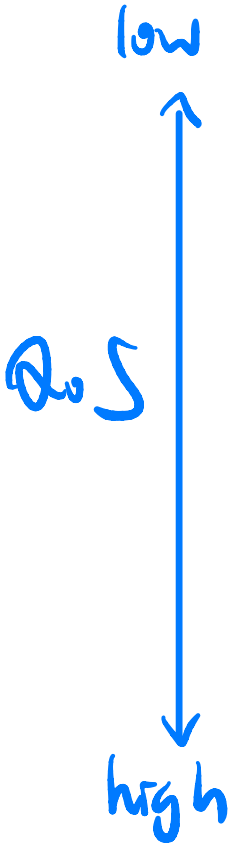
```
const mqtt = require('mqtt');
const client = mqtt.connect();
client.on('connect', function () {
   client.publish('MY_TOPIC', 'hello from js');
   client.end();
});
```

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();
});
```

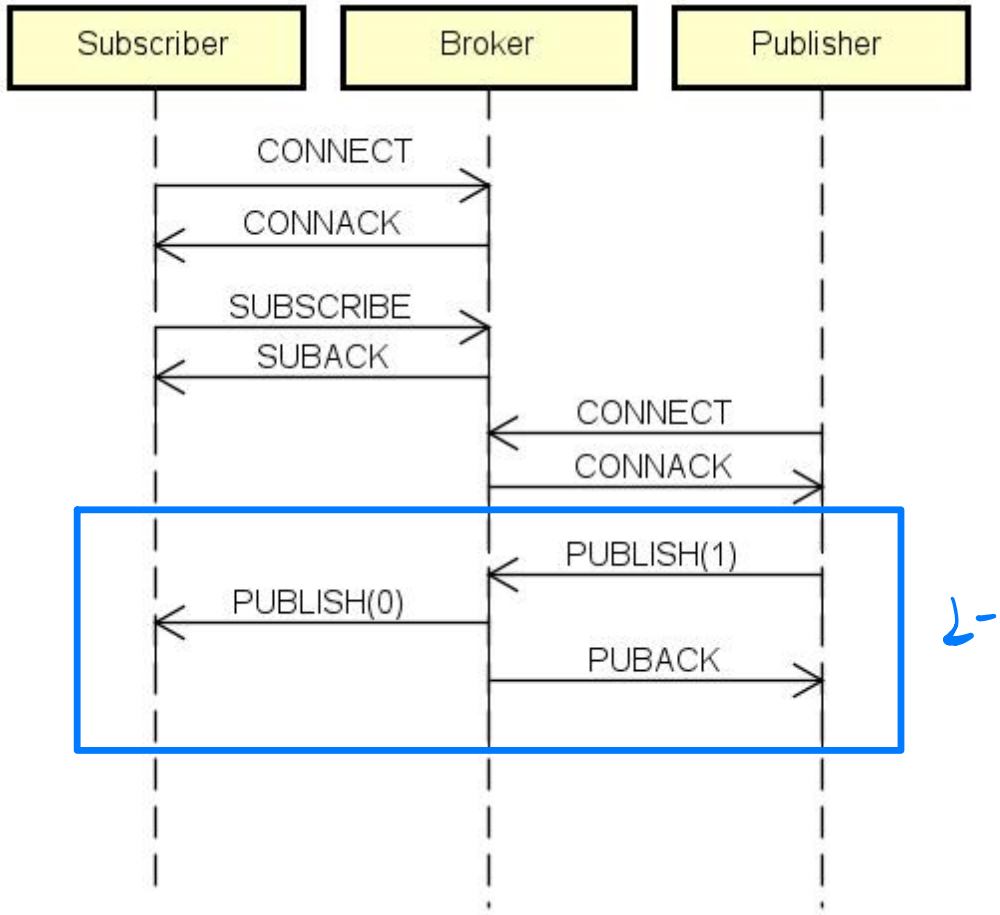
deliver messages with varying level of assurance

Case: MQTT QoS and Last Will



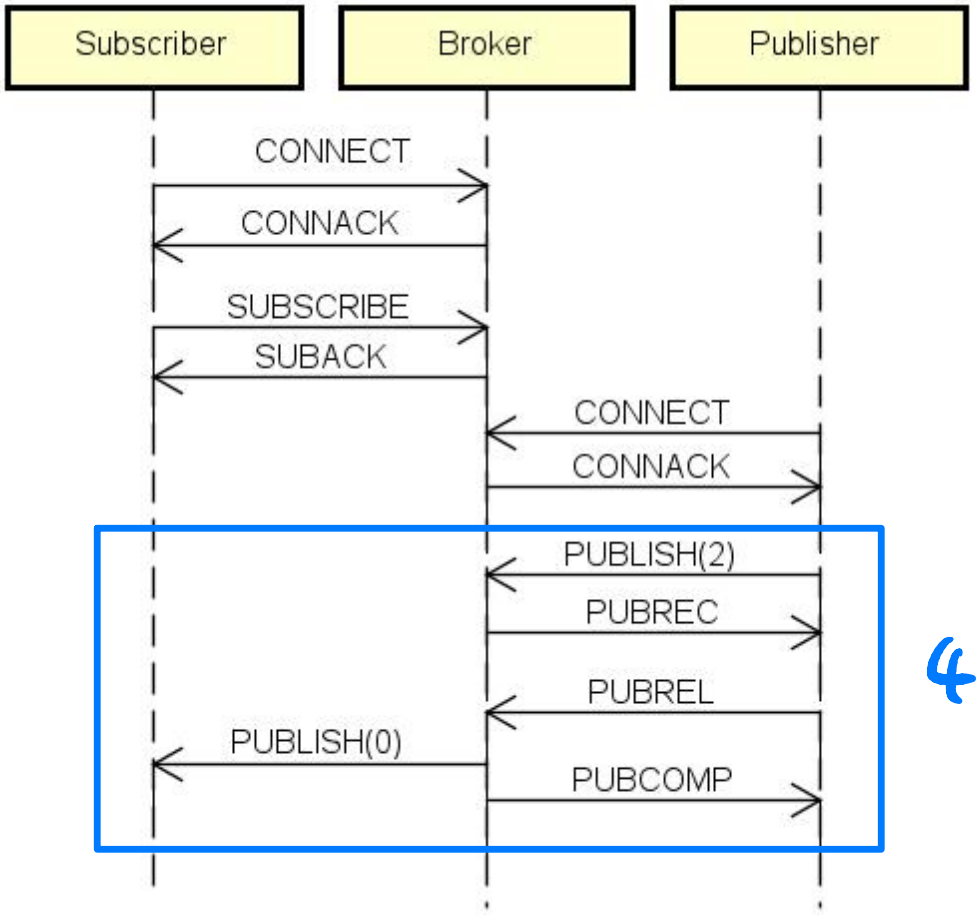
- QoS 0 “At most once”，最多一次 (fire-and-forgot)
  - 訊息遺失或是重複發送的狀況可能會發生
  - Ex: 溫度感測
- QoS 1 "At least once"，至少一次
  - 採用ACK進行訊息確認送達，有問題則重送
  - 若遇到non-idempotent 邏輯，訊息重複會造成錯誤
- QoS 2 "Exactly once"，確定一次
  - 確認訊息只會送到一次
  - 耗費較多資源與網路頻寬
- Last Will
  - 連線後可事先設定Last Will，敘明當異常斷線發生時的處理方式

Case: MQTT QoS 1 (Publisher-side only)



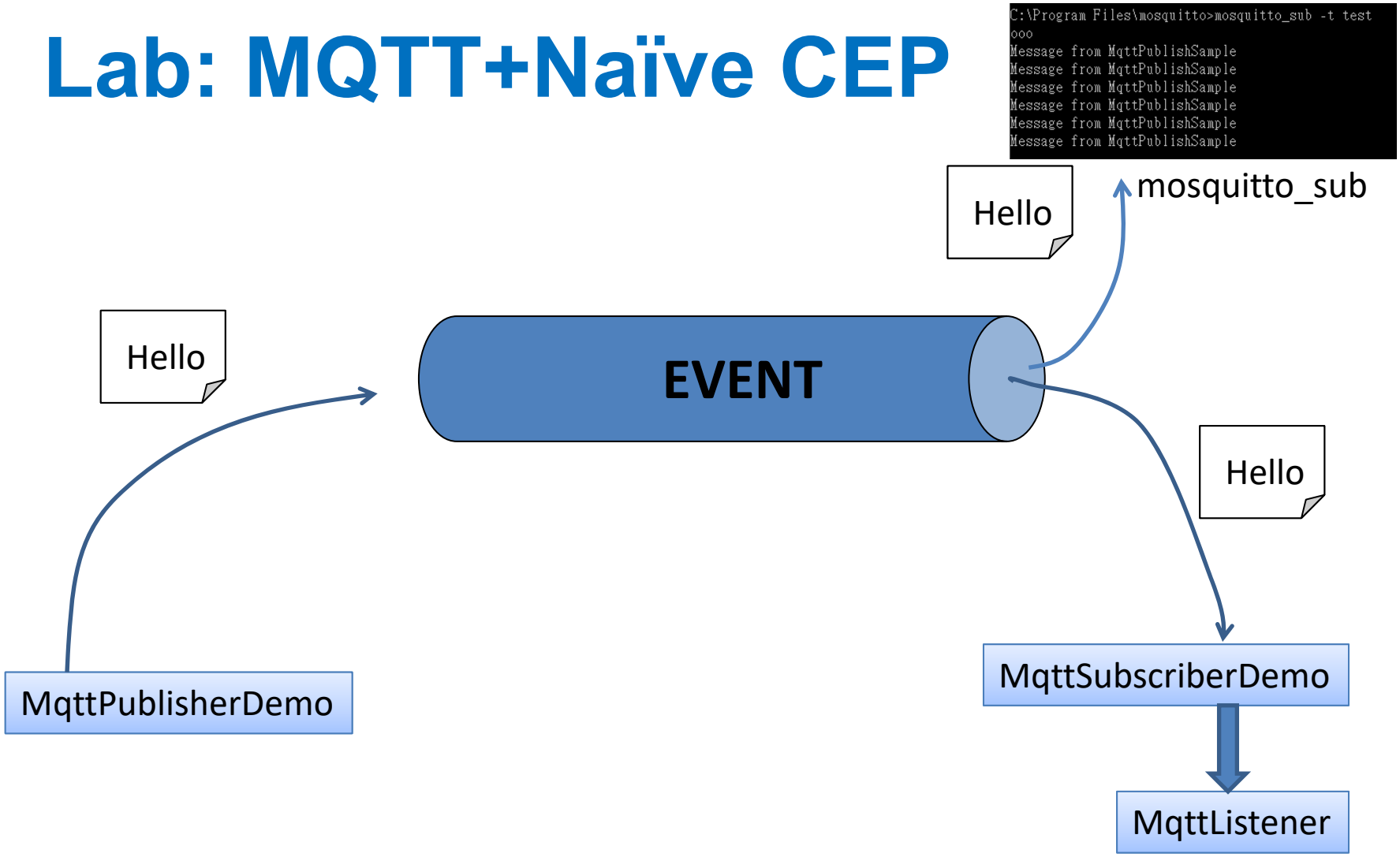
2-step handshake

Case: MQTT QoS 2 (Publisher-side only)



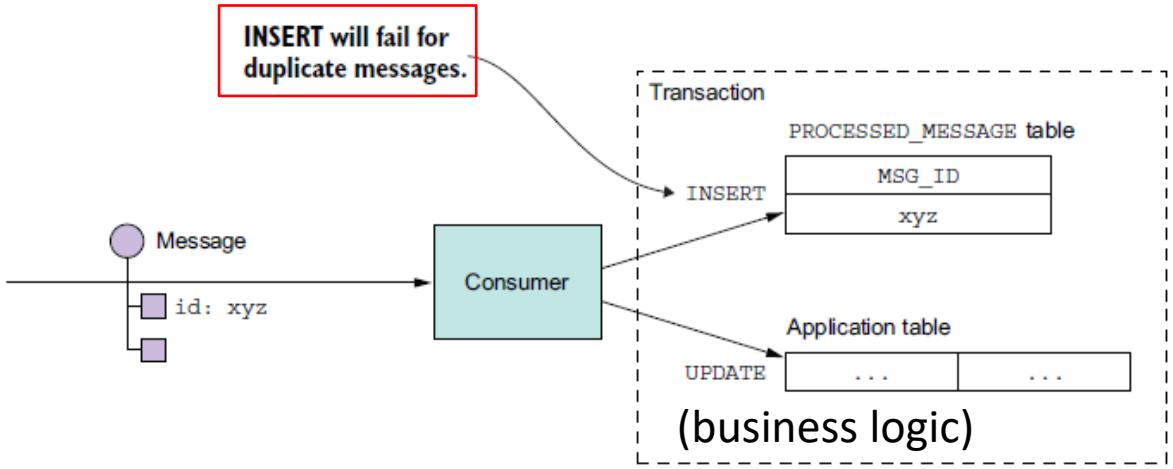
4-step handshake.

Lab: MQTT+Naïve CEP



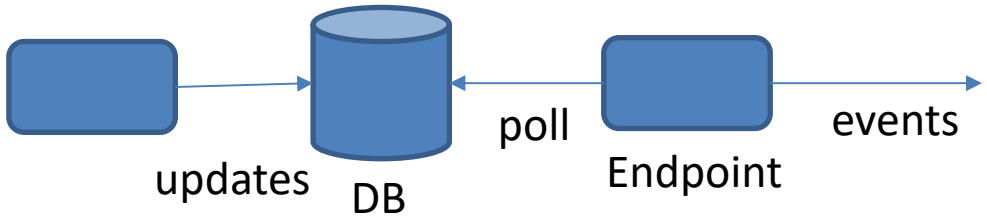
端點: Handling duplicate messages

- Two options
  - Idempotent (重覆訊息不影響邏輯正確性) message handler
  - Track messages and discard duplicates
    - A simple implementation (using DB transaciton)
      1. Insert message_id of each message into a local database
      2. Message_id as a primary key: key重覆的話，此交易就失敗



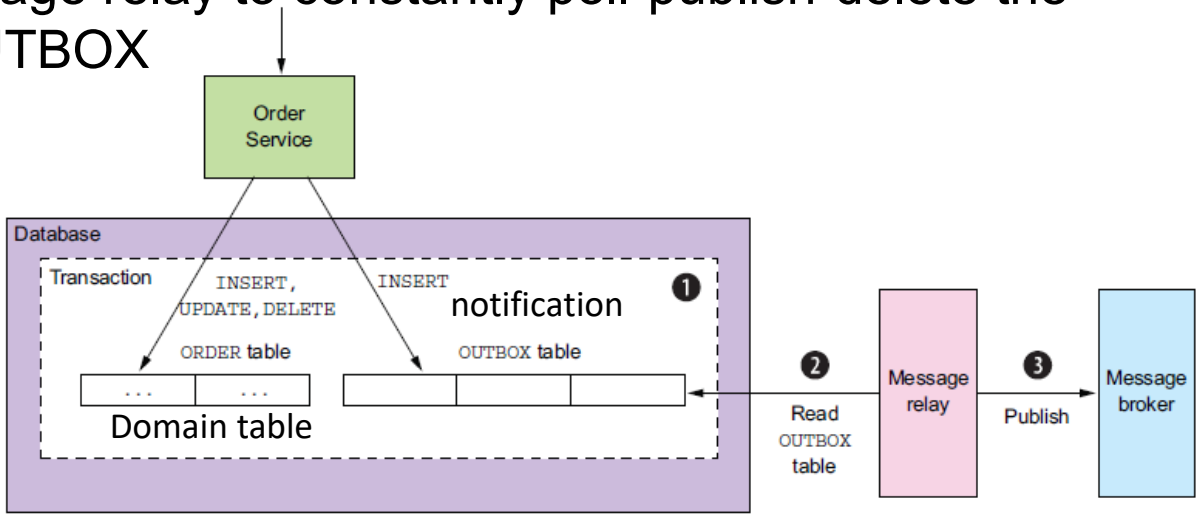
端點: Transactional Messaging

- 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)



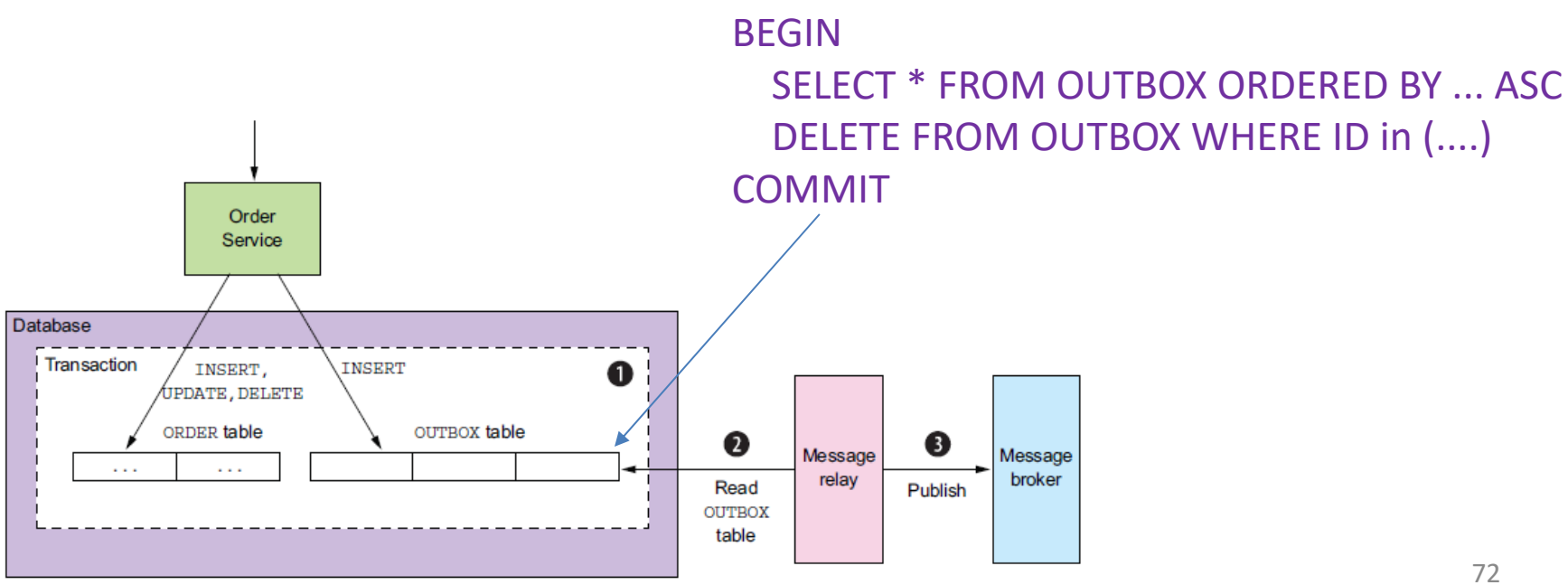
Transactional Outbox

- Solution: 確保state修改, notification一定會送出
  - Write the message to be published into a OUTBOX table
  - Updates to domain tables and the OUTBOX table are bundled into a Transaction
    - 確保「更改domain table」與「notification」一同被完成
      - 例: 完成ORDER修改→ service crash → OUTBOX還沒改所以rollback
  - Using a message relay to constantly poll-publish-delete the records in OUTBOX



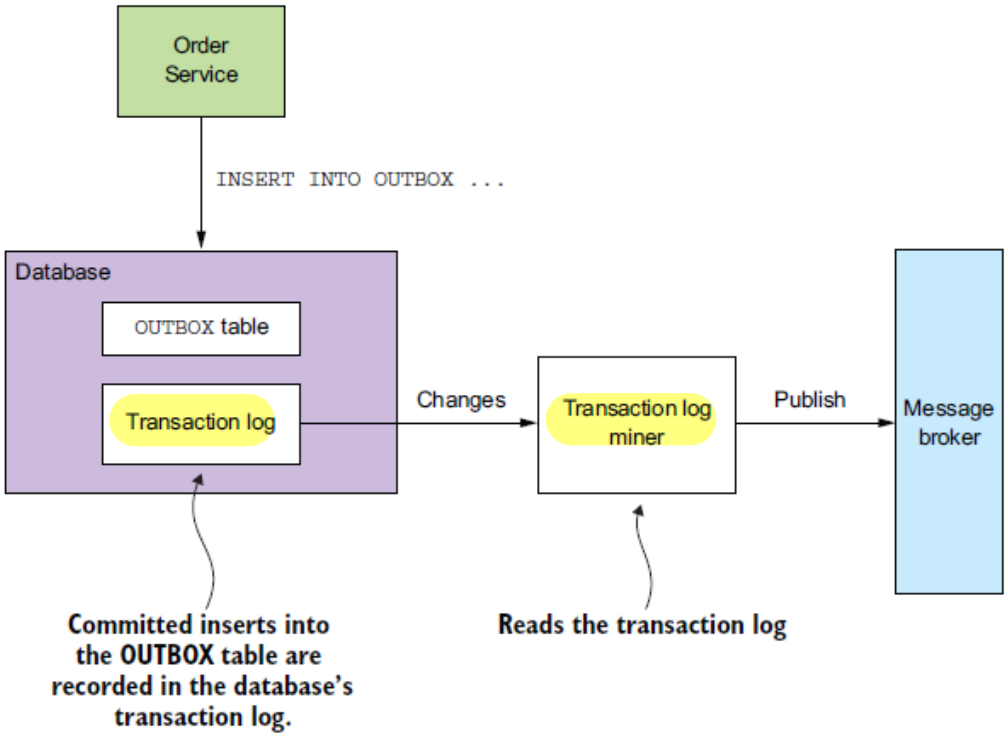
Realizing Message Relay

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



Realizing Message Relay

- Transaction Log Tailing (又稱CDC, Change Data Capture)
  - Use a transaction log miner to read the low-level transaction logs of the DB and publish each change to the message broker

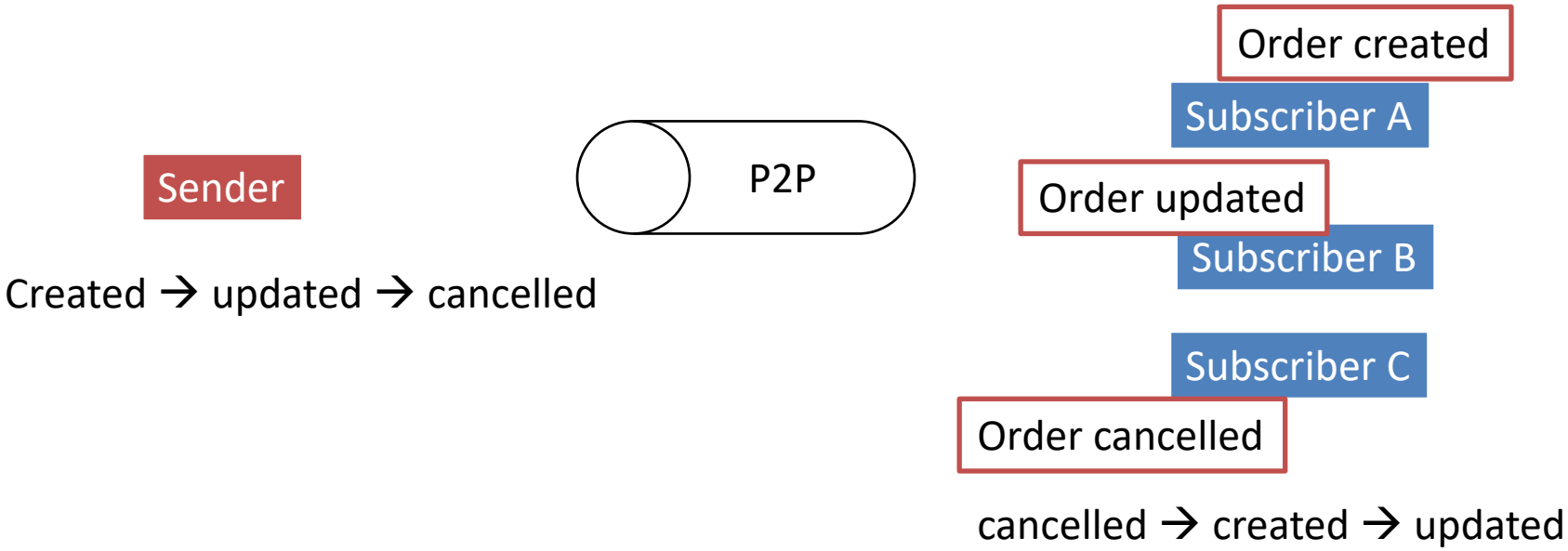


Realizing Message Relay

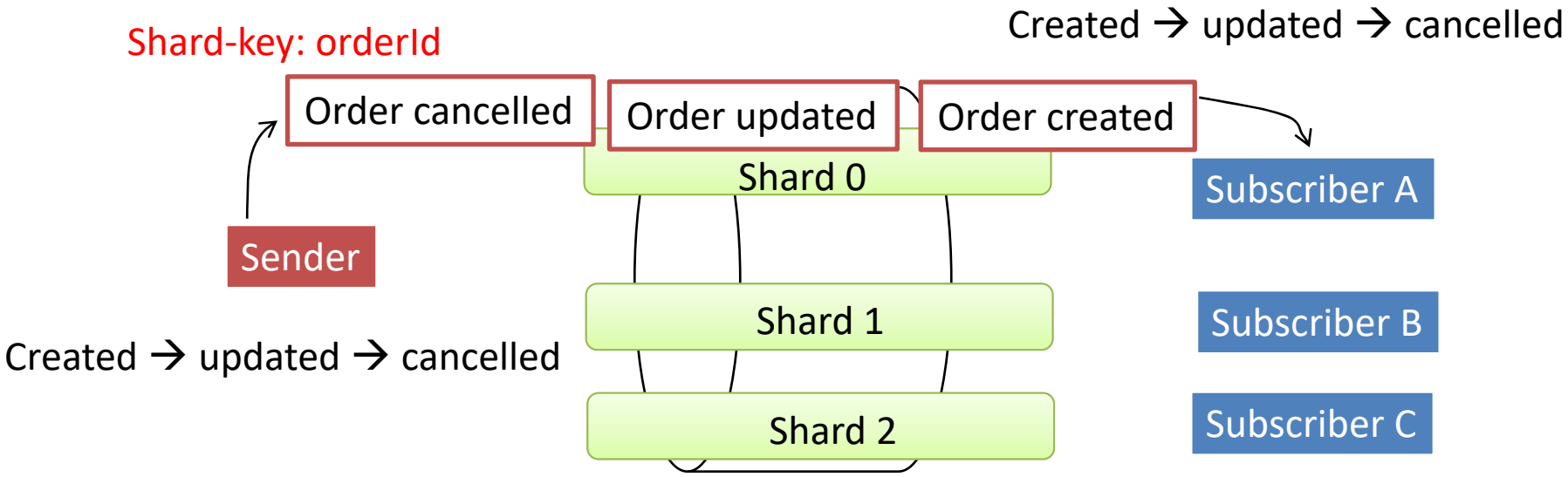
- Transaction Log Tailing Tools
  - Debezium
    - publishes database changes to the Apache Kafka message broker
  - Eventuate Tram (by Chris Richardson)
    - An enhanced version of Debezium
  - LinkedIn Databus
    - mines the Oracle transaction log
  - DynamoDB streams
    - DynamoDB built-in service

Message Ordering Problem

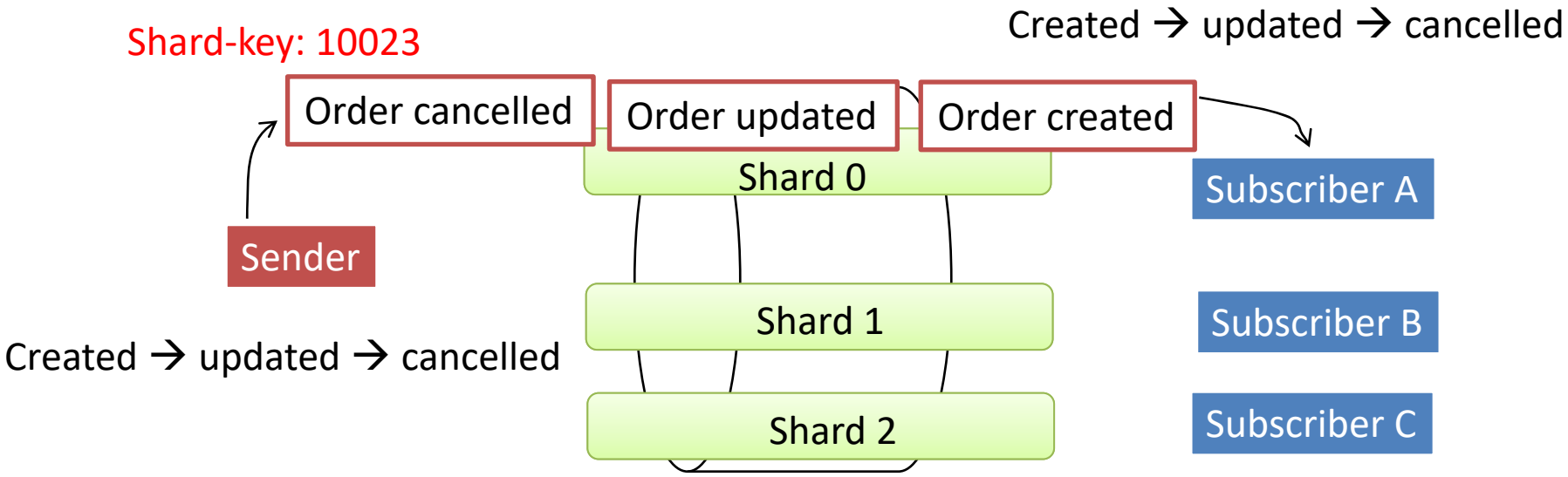
- Multiple subscribers use the same topic
  - 希望做load balance，但又希望某些訊息依次序接收
  - Example: (order created, order updated, order cancelled)



Channel Sharding

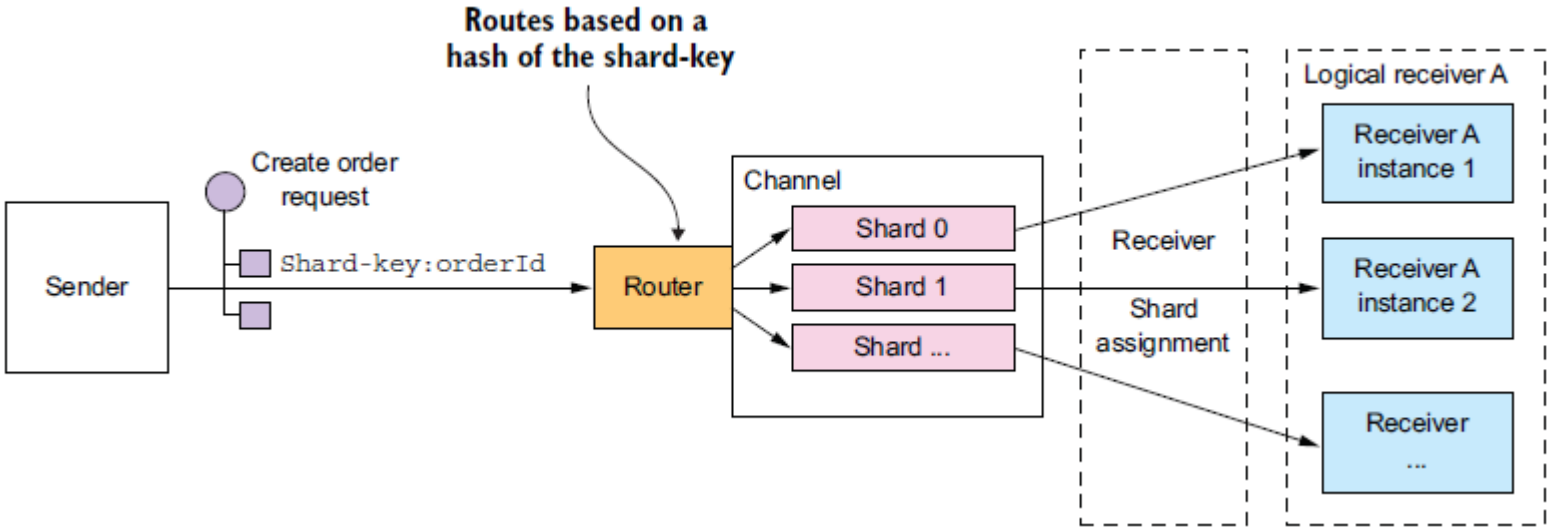


將需要具有次序的一群訊息，以shard-key group起來
MOM會將它們綁定同一個shard queue中; 被同一個subscriber處理


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

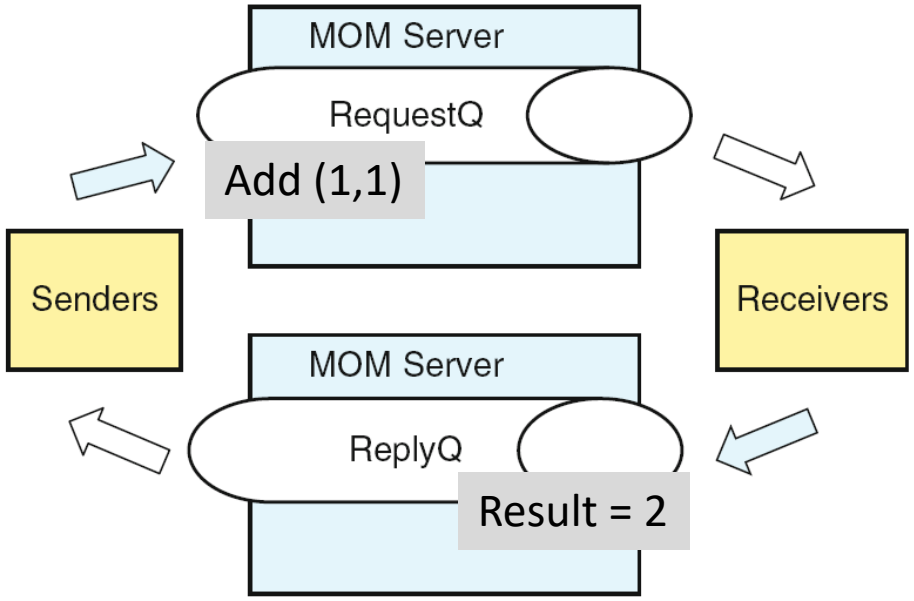
Channel Sharding

- 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

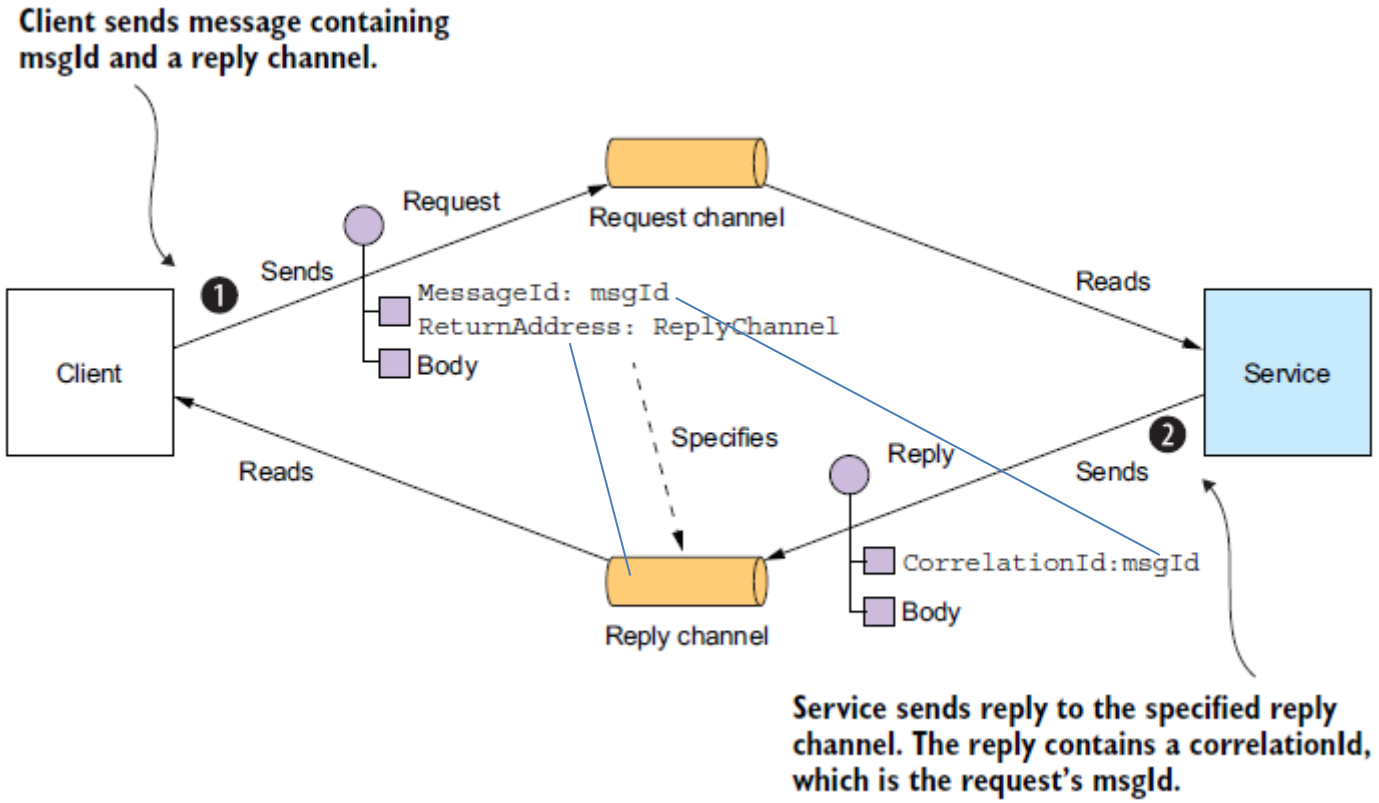


Simulating RPC

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

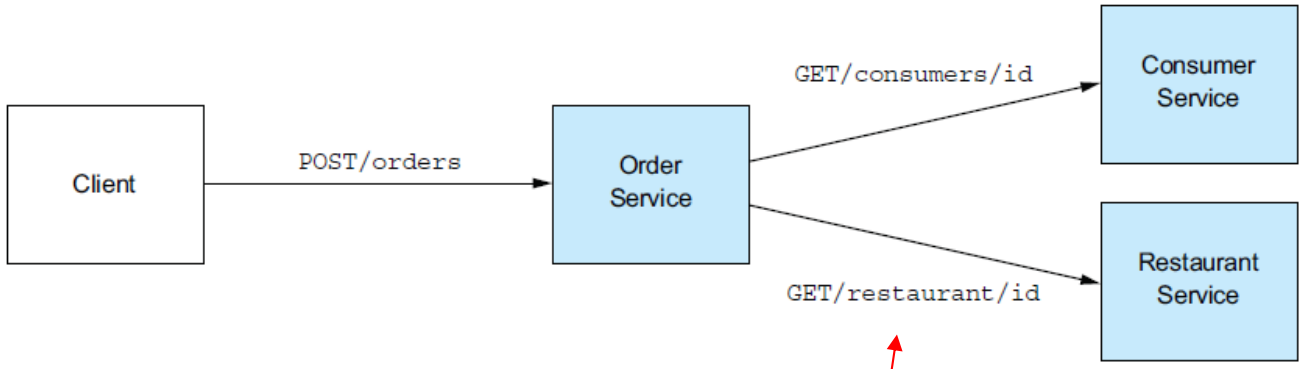


Simulating a Call



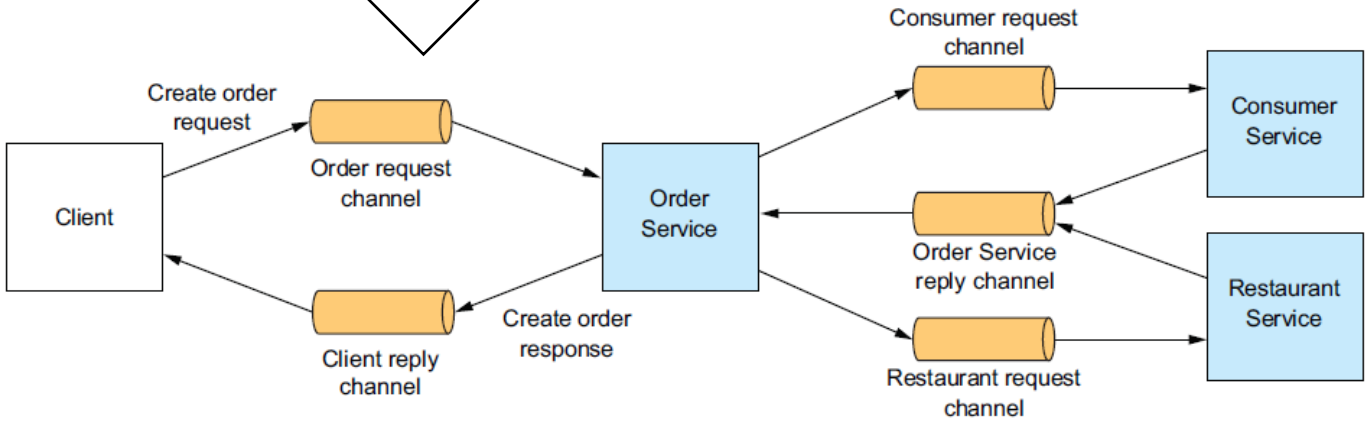
Case:提升RESTful WS的availability

- 如下圖中的同步RPC，呼叫時，三個服務必須同時available

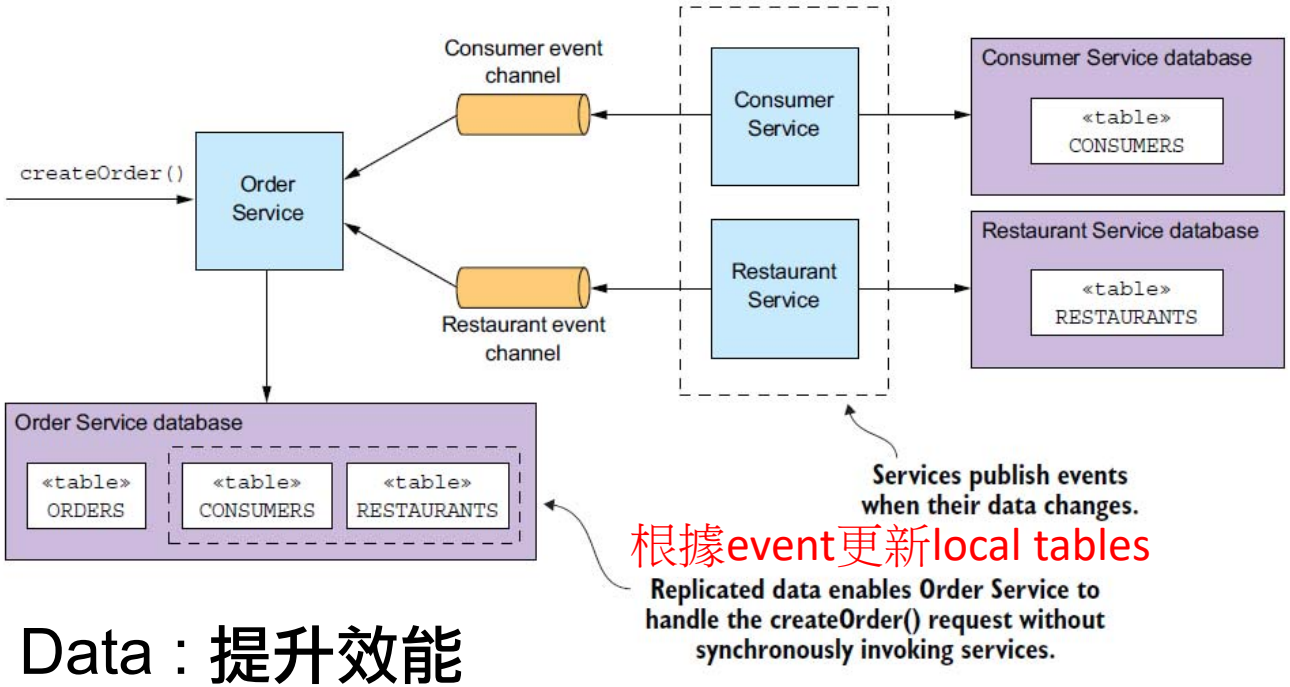


觀察: 只涉及讀取邏輯

變成非同步之後，若服務不available，訊息會先存在MQ中



有更新時發出 event



- Replicate Data：**提升效能**
  - Order對Consumer與Restaurant都只涉及查詢
    - 在Order中維護一份Consumer與Restaurant的資料庫副本
    - Order中副本接收更新的Domain Event，有更新時才更新
    - 如此一來，原來2個查詢式RPC要求直接變成local db查詢
  - **缺點: 只適用於查詢**
    - Order Service無法直接對真正的table內容進行修改

類似 cache