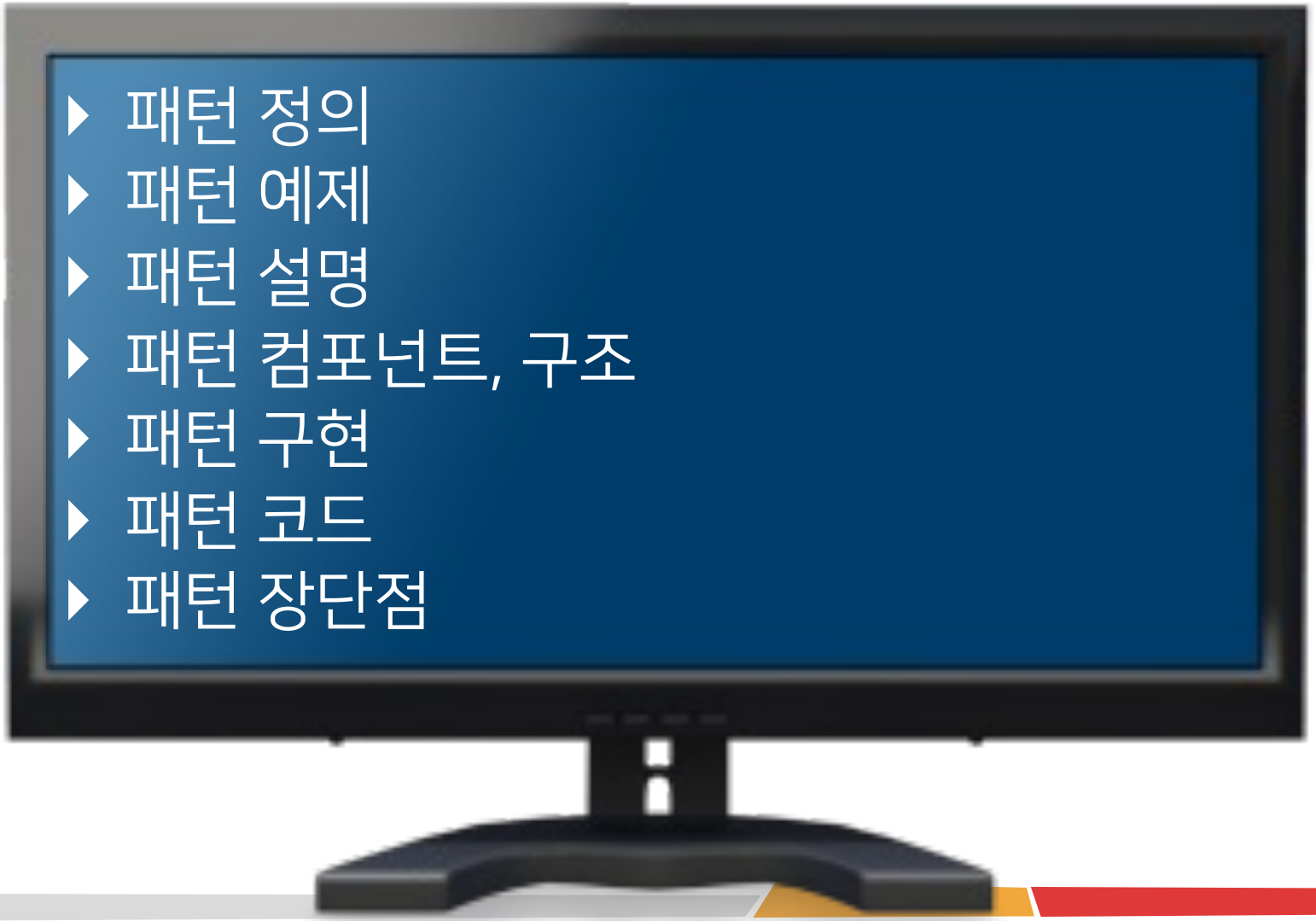
Spring 2023

소프트웨어 아키텍처 패턴: Publisher-Subscriber

Seonah Lee

Gyeongsang National University

Publisher-Subscriber 패턴

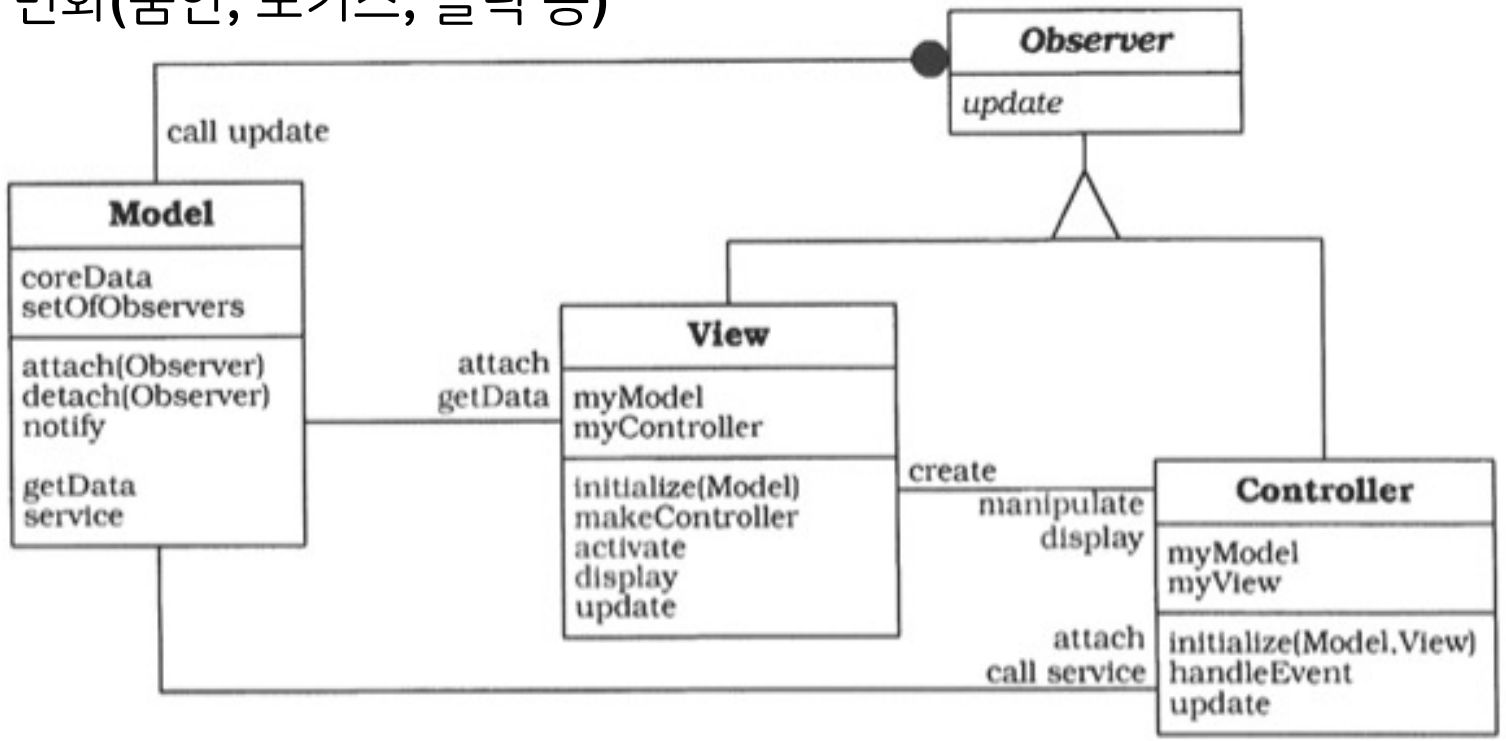


Publisher-Subscriber Pattern: Definition

- 정의
  - 하나의 **Publisher**가 다수의 **Subscriber**에게 상태가 변경되었음을 단방향 전파로 통지하는 패턴
  - 협력 컴포넌트들의 상태를 동기화하는데 유용
  - 다음 패턴으로 응용
    - **Observer** 패턴
    - **Dependents** 패턴
    - **Event** 패턴

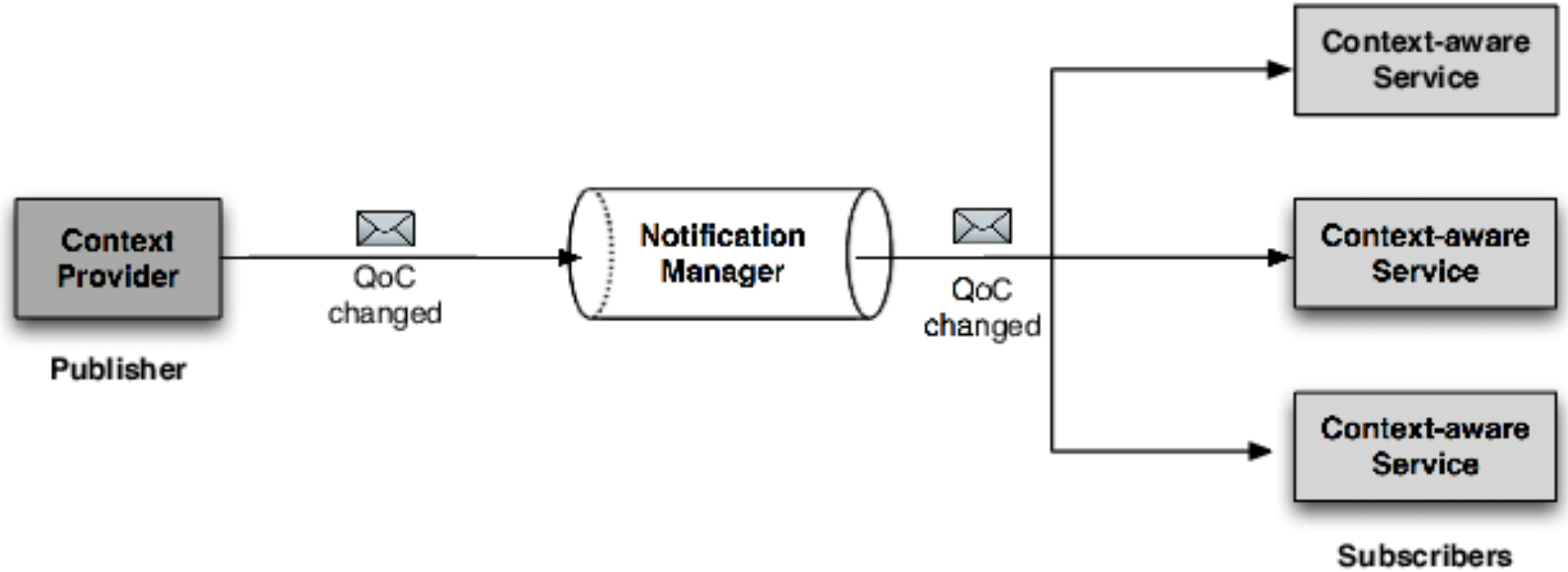
Publisher-Subscriber Pattern: Example

- 예제
  - **GUI** 애플리케이션
    - 사용자의 요청에 따른 화면의 변화(줌인, 포커스, 클릭 등)
  - **MVC** 패턴 애플리케이션



Publisher-Subscriber Pattern: Description

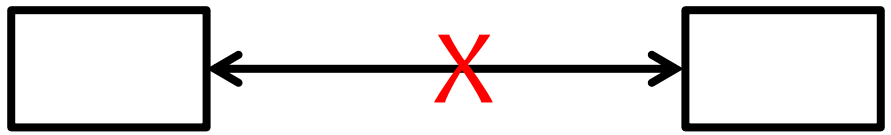
- 정황(Context)
  - 한번의 호출로 다수의 협력 컴포넌트의 상태를 변경해야 하는 경우



https://www.researchgate.net/figure/Topic-based-publish-subscribe-system-Fig-3-depicts-this-model-QoC-offerings-of-context_fig2_279532489

Publisher-Subscriber Pattern: Description

- 문제(**Problem**)
  - 특정 컴포넌트에서 발생하는 상태 변경 정보를 하나 이상의 컴포넌트에 통지
    - 종속 컴포넌트들의 개수 및 신원에 대해서는 미리 알 필요도 없으며, 언제든 변경되어도 무관
    - 종속 컴포넌트들이 새로운 정보를 받기 위해 명시적 **Polling** 불가
  - 정보 게시자(**Publisher**)와 종속자(**Subscriber**)는 서로 **tightly coupled** 되어서는 안됨



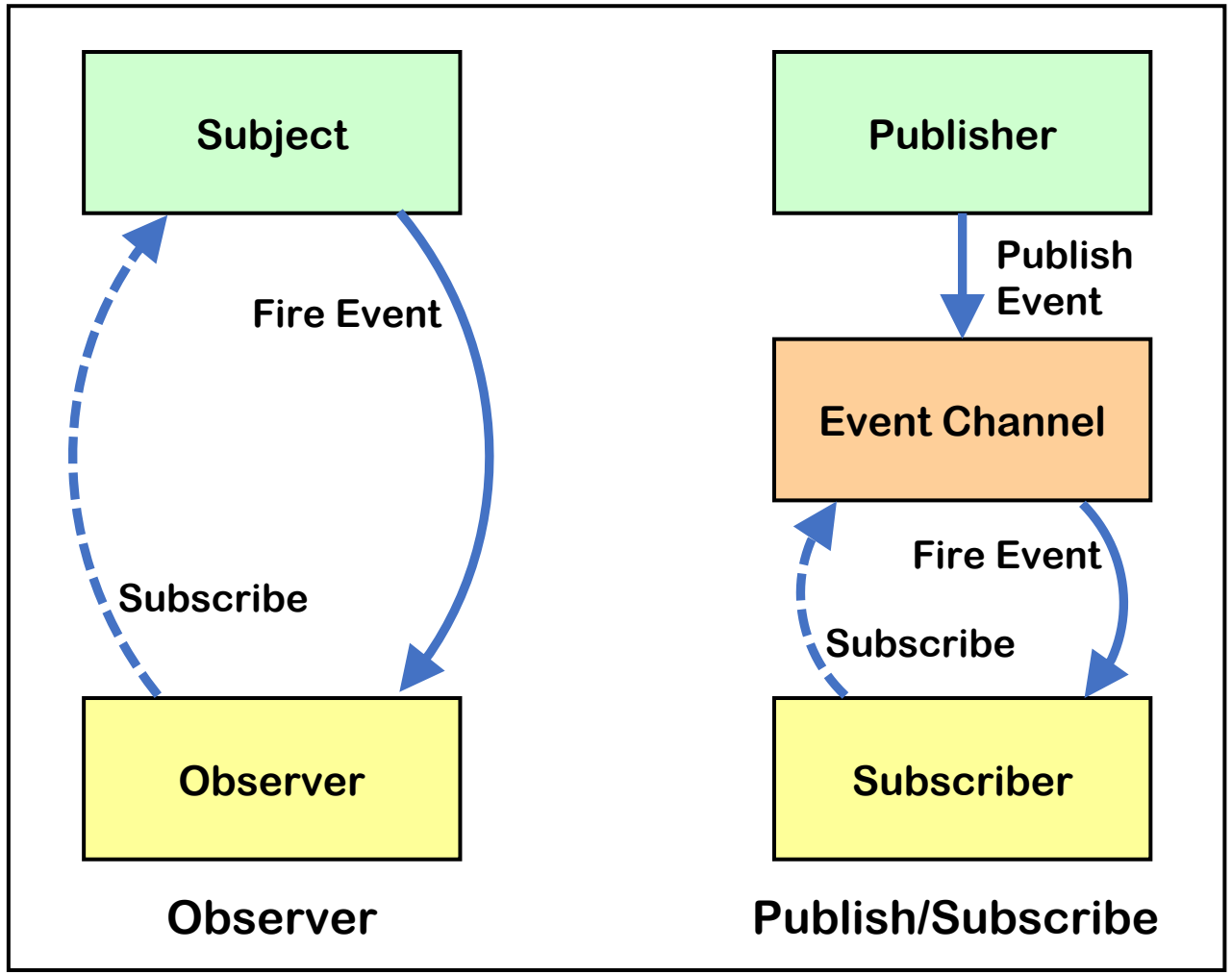
Publisher-Subscriber Pattern: Description

- 해법(Solution)
  - 하나의 컴포넌트가 게시자(**Publisher**)의 역할을 수행
    (One dedicated component takes the role of the publisher.)
  - 게시자에서 일어나는 상태 변경을 받을 컴포넌트들은 그 게시자의 구독자 **(subscriber)**에 해당

게시자(Publisher)	현재 구독된 컴포넌트들의 레지스트리를 저장
구독자(Subscriber)	게시자(Publisher)가 제공한 인터페이스를 통한 등록 및 구독 인터페이스를 사용한 구독 취소 가능

  - 게시자**(Publisher)**의 상태가 변경될 때마다 게시자**(Publisher)**는 등록된 모든 구독자**(Subscriber)**에게 변경 상태 전송

Observer Pattern vs. Pub-Sub Pattern



https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c

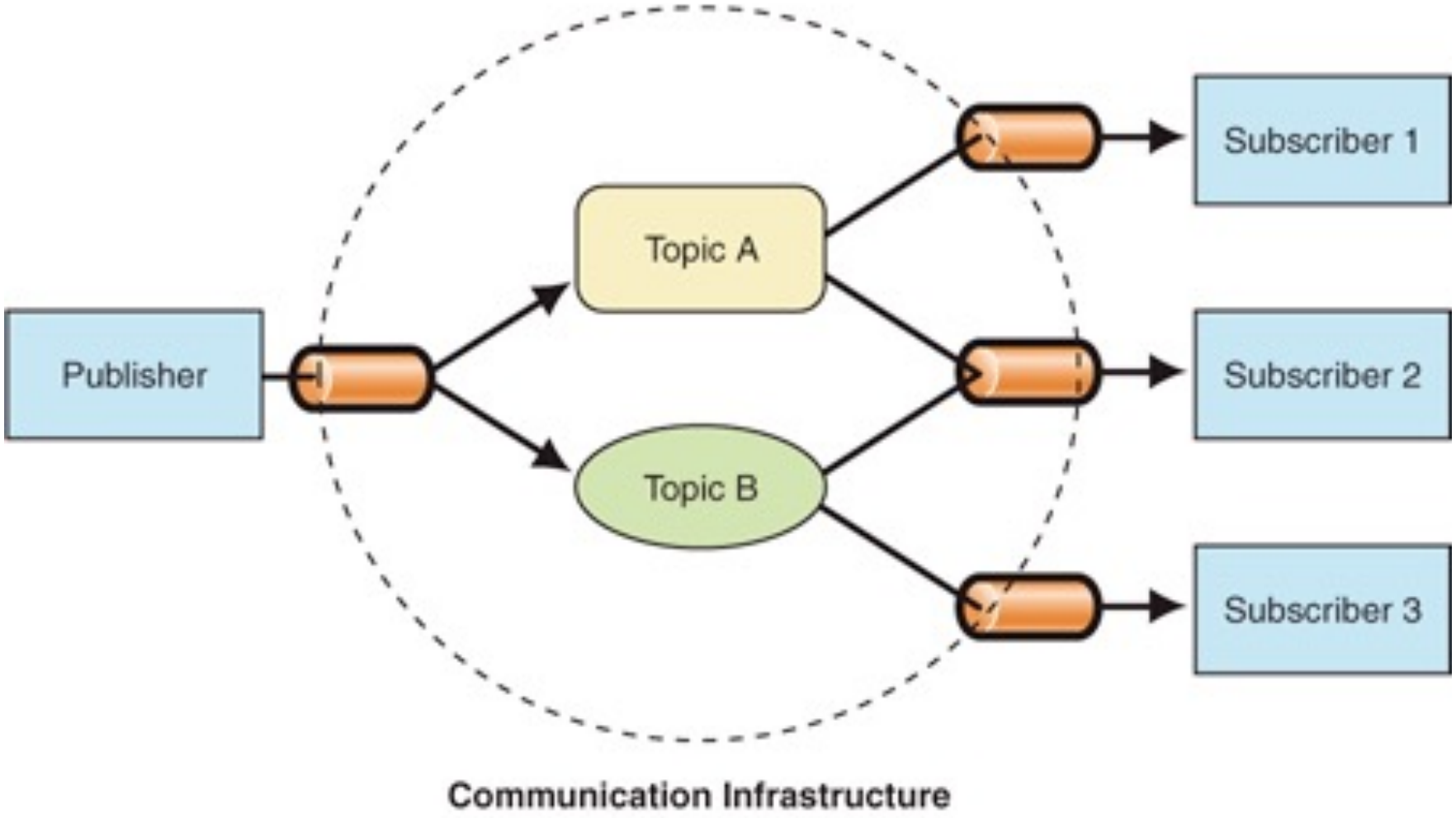
Publisher-Subscriber Pattern: Components

Publisher
• 특정 주제에 대한 Message를 만들어 메시지 서비스를 전송 • Observer Pattern의 Subject와 같은 역할

Subscriber
• 지정한 Subscription에 대한 Message를 받음 • Observer Pattern의 Observer와 같은 역할

Event Channel
• Publisher로 부터 Event를 받아 필터링 후 Subscriber에 송신 • 중간 컴포넌트(Broker 또는 Message Bus) 역할

Publisher-Subscriber Pattern: Structure



https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c

Publisher-Subscriber Pattern: Realization

- 구현
  - 이벤트 기반으로 **Publisher-Subscriber** 패턴 구현
  - 이벤트 기반으로 구현 시 시스템 변경이 쉬움
  - 이벤트가 명확히 전송되었는지 알 수가 없음 **(Non-Deterministic)**
  - 응답 시간을 명확히 예측할 수 없음 **(Non-Deterministic)**

Publisher-Subscriber Pattern: Realization

- 프로그래밍 언어
  - **C** 언어로 구현하기 어려움
  - 자바 언어를 사용할 경우 **Observer, Observable**을 사용하여 구현할 수 있음
    - ***Java 9 deprecated Observer, Observable***

- 기본 구현
  - 게시자**(Publisher)**: 메시지 생성
  - 구독자**(Subscriber)**: 메시지 구독
  - 중간매개체**(EventChannel)**: 메시지 중재

https://stackoverflow.com/questions/46380073/observer-is-deprecated-in-java-9-what-should-we-use-instead-of-it

Publisher-Subscriber Pattern: Implementation

```
public class Publisher {
    ...
    public Publisher(Topic t) {
        this.topic = t;
    }


    public void publish(Message m) {
        EventChannel.getInstance().sendMessage(this.topic, m);
    }
}
```

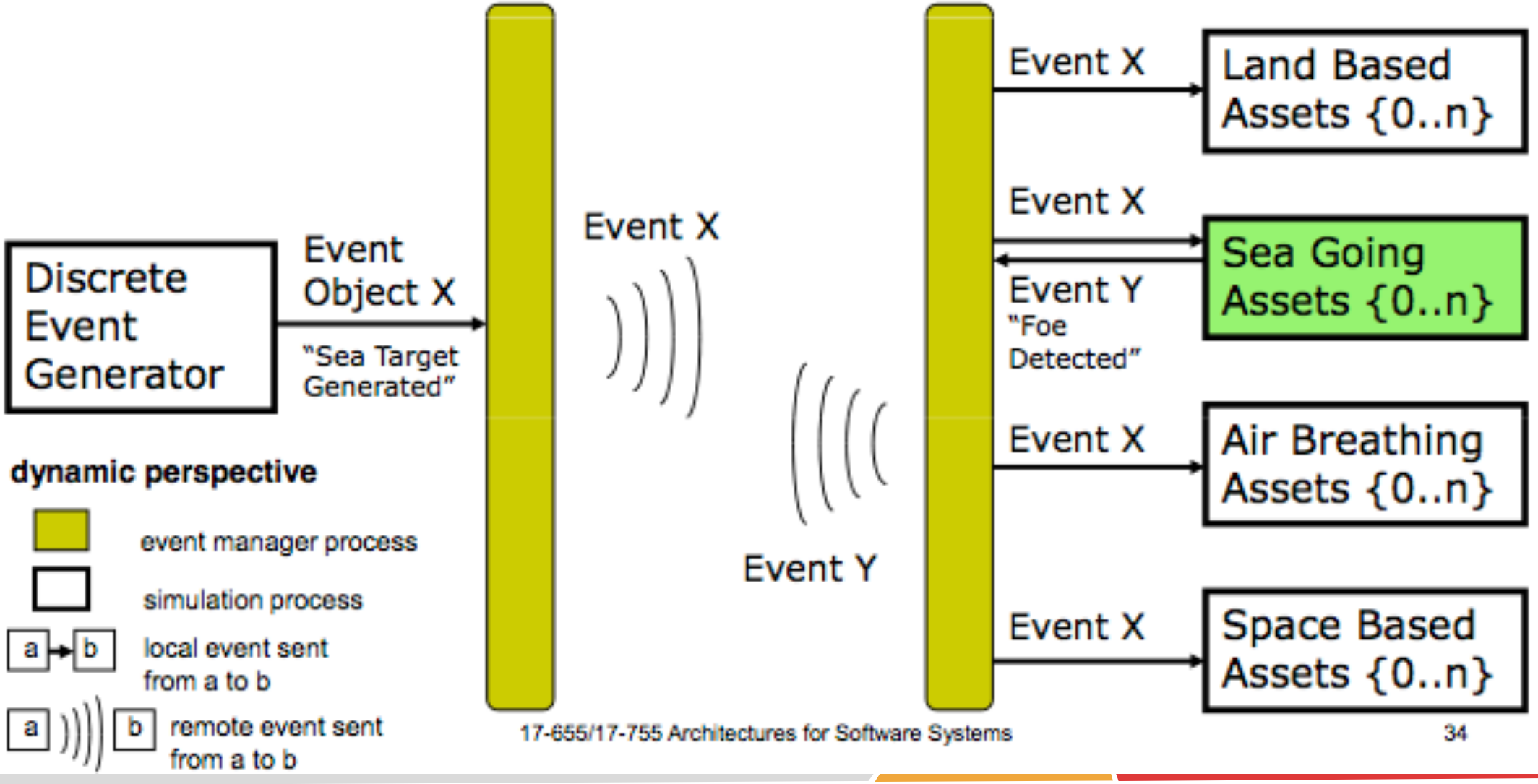
Publisher-Subscriber Pattern: Implementation

```
public class EventChannel {
    private Hashtable<Topic, List<Subscriber>> subscriberLists;
    private static EventChannel serverInstance;

    public static EventChannel getInstance() {
        if (serverInstance == null) {
            serverInstance = new EventChannel();
        }
        return serverInstance;
    }
    private EventChannel() {
        this.subscriberLists = new Hashtable<>();
    }
    public sendMessage(Topic t, Message m) {
        List<Subscriber> subs = subscriberLists.get(t);
        for (Subscriber s : subs) {
            s.receivedMessage(t, m);
        }
    }
    public void registerSubscriber(Subscriber s, Topic t) {
        subscriberLists.get(t).add(s);
    }
}
```

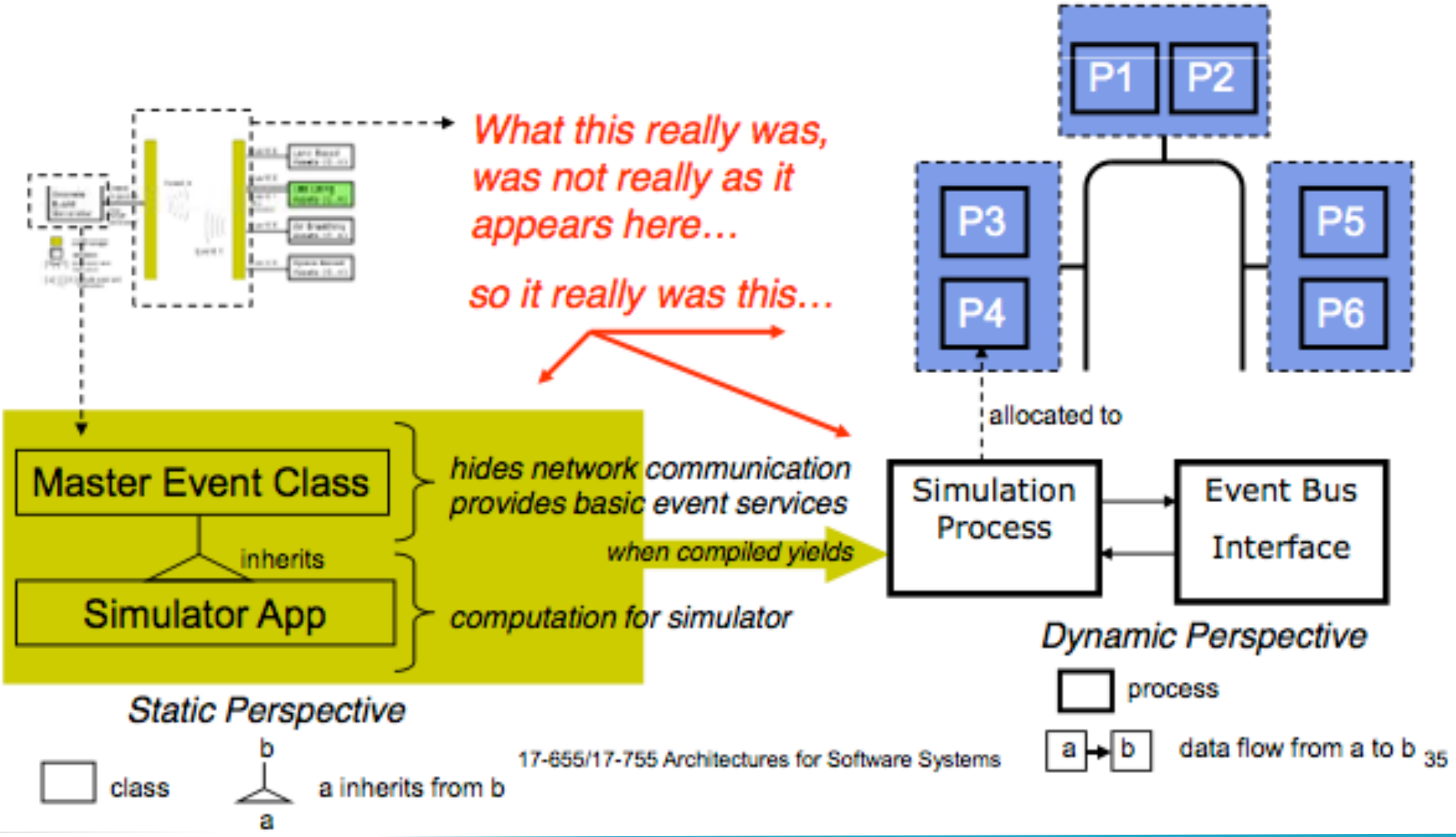
Publisher-Subscriber Pattern: Implementation

```
public class Subscriber {
    public Subscriber(Topic...topics) {
        for (Topic t : topics) {
            EventChannel.getInstance().registerSubscriber(this, t);
        }
    }

    public void receivedMessage(Topic t, Message m) {
        switch(t) {
            ...
        }
    }
}
```

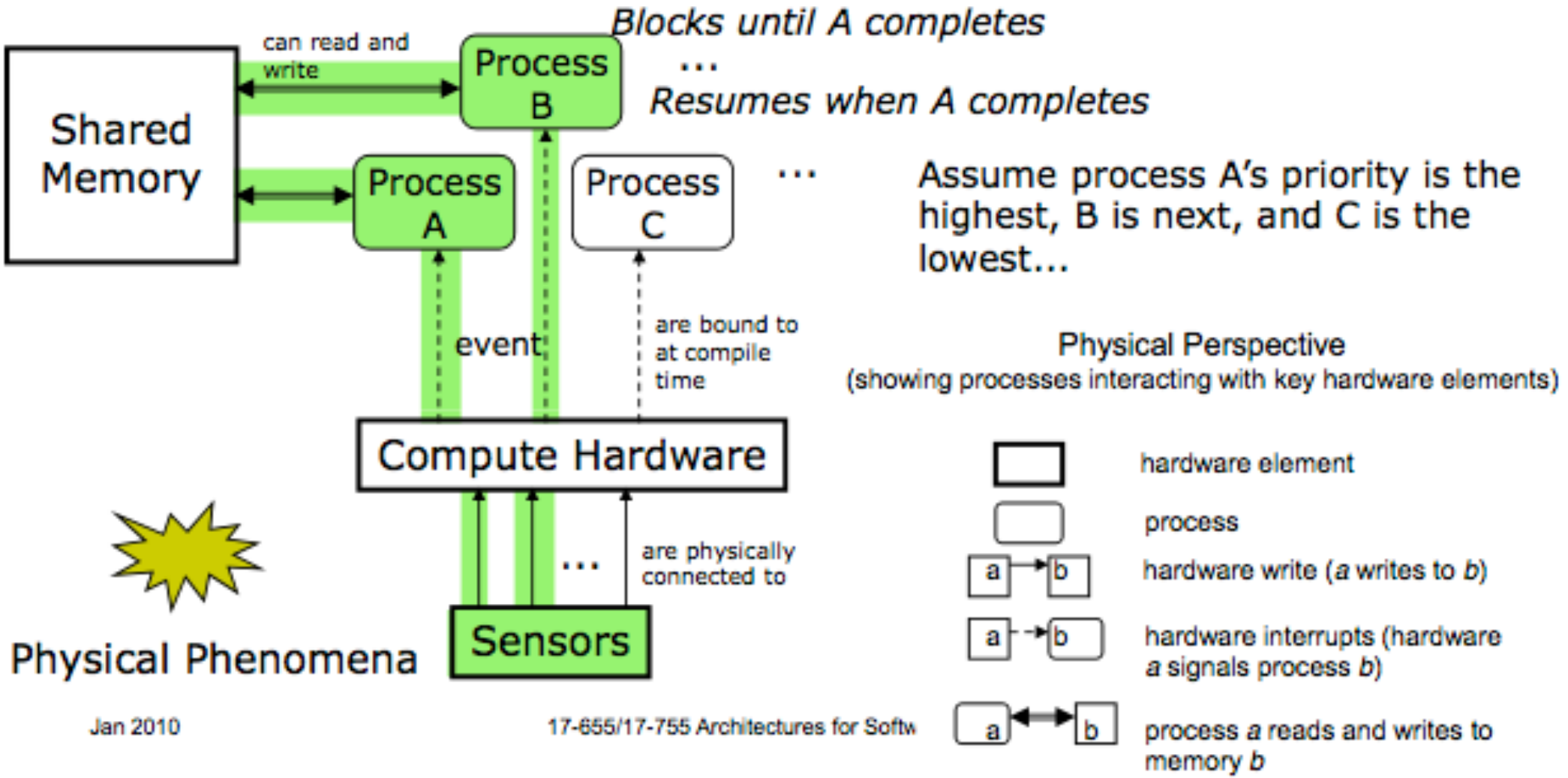
Publisher-Subscriber Pattern: Case Studies



Publisher-Subscriber Pattern: Case Studies

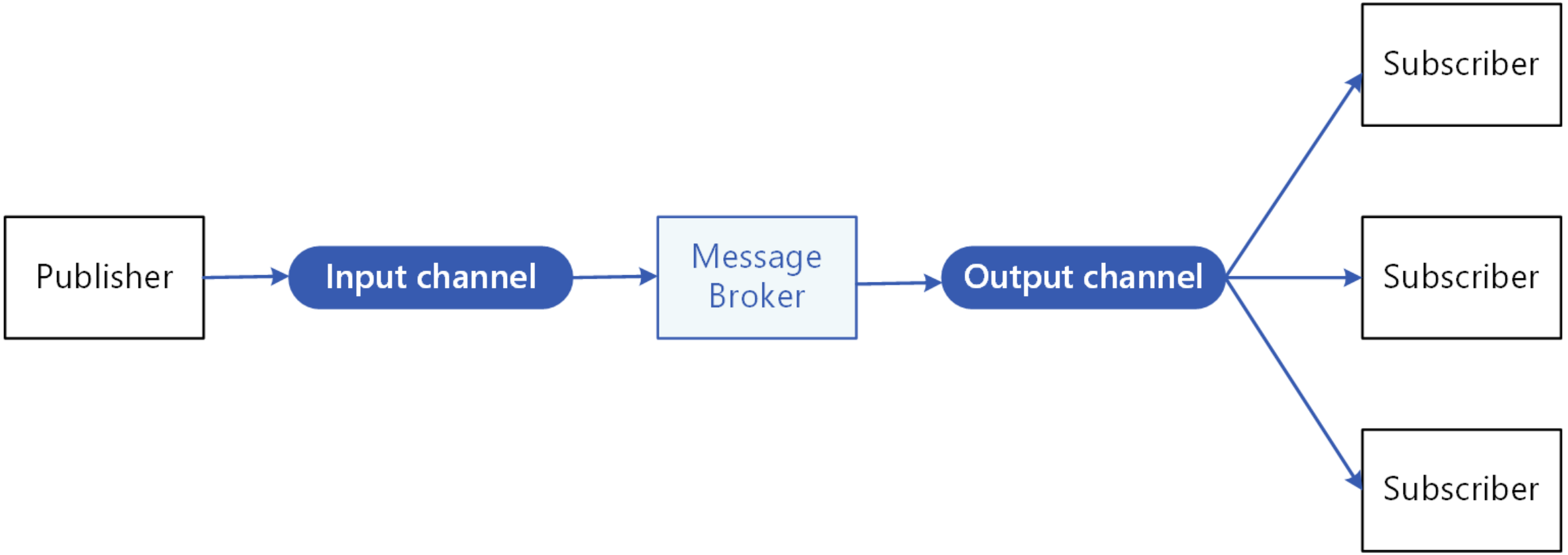


Publisher-Subscriber Pattern: Case Studies

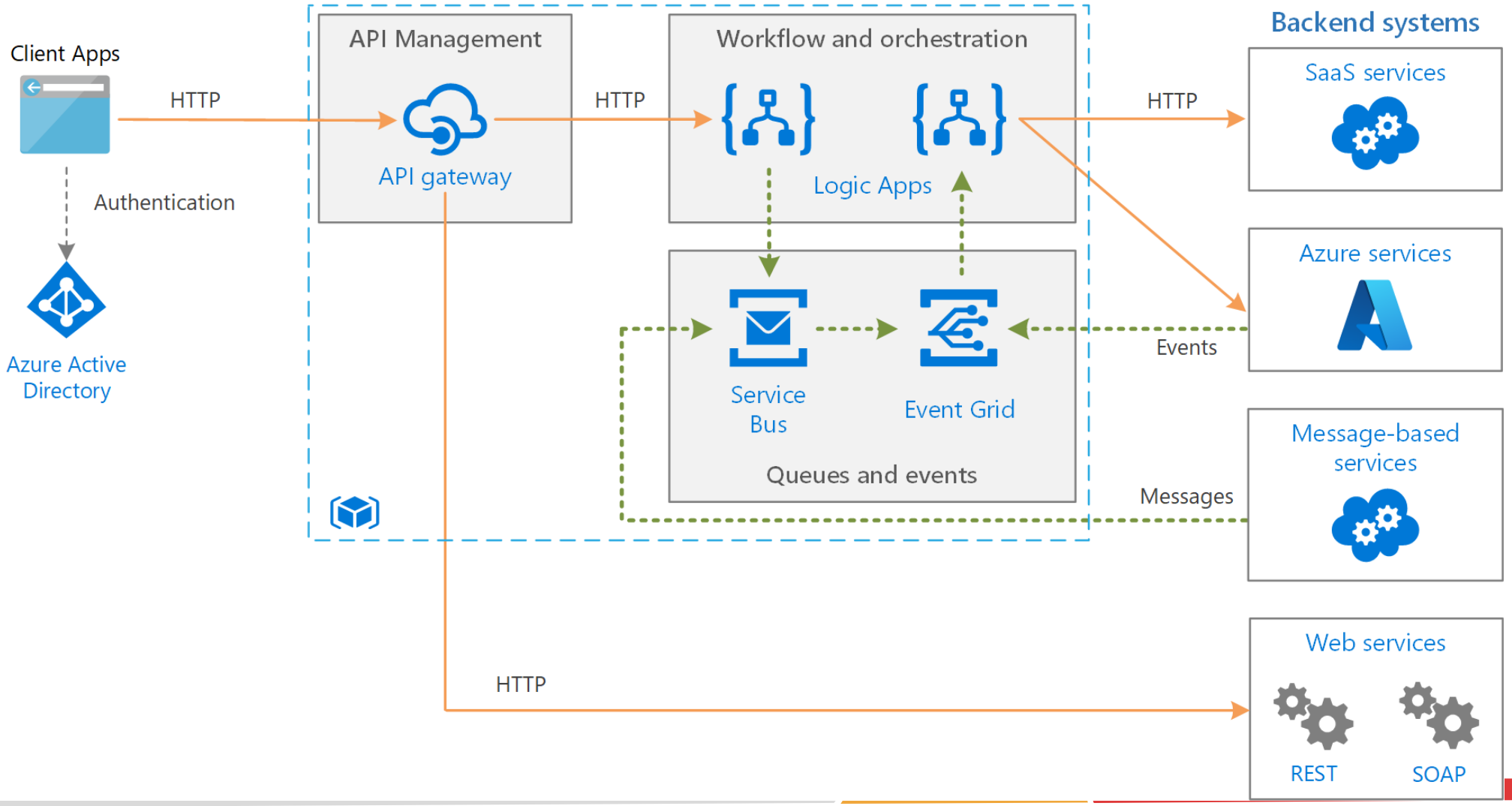


Referenced by : David Garlan, Lecture notes of "17-655/17-755 Architecture for software system", Institute for software research, CMU, 2011.

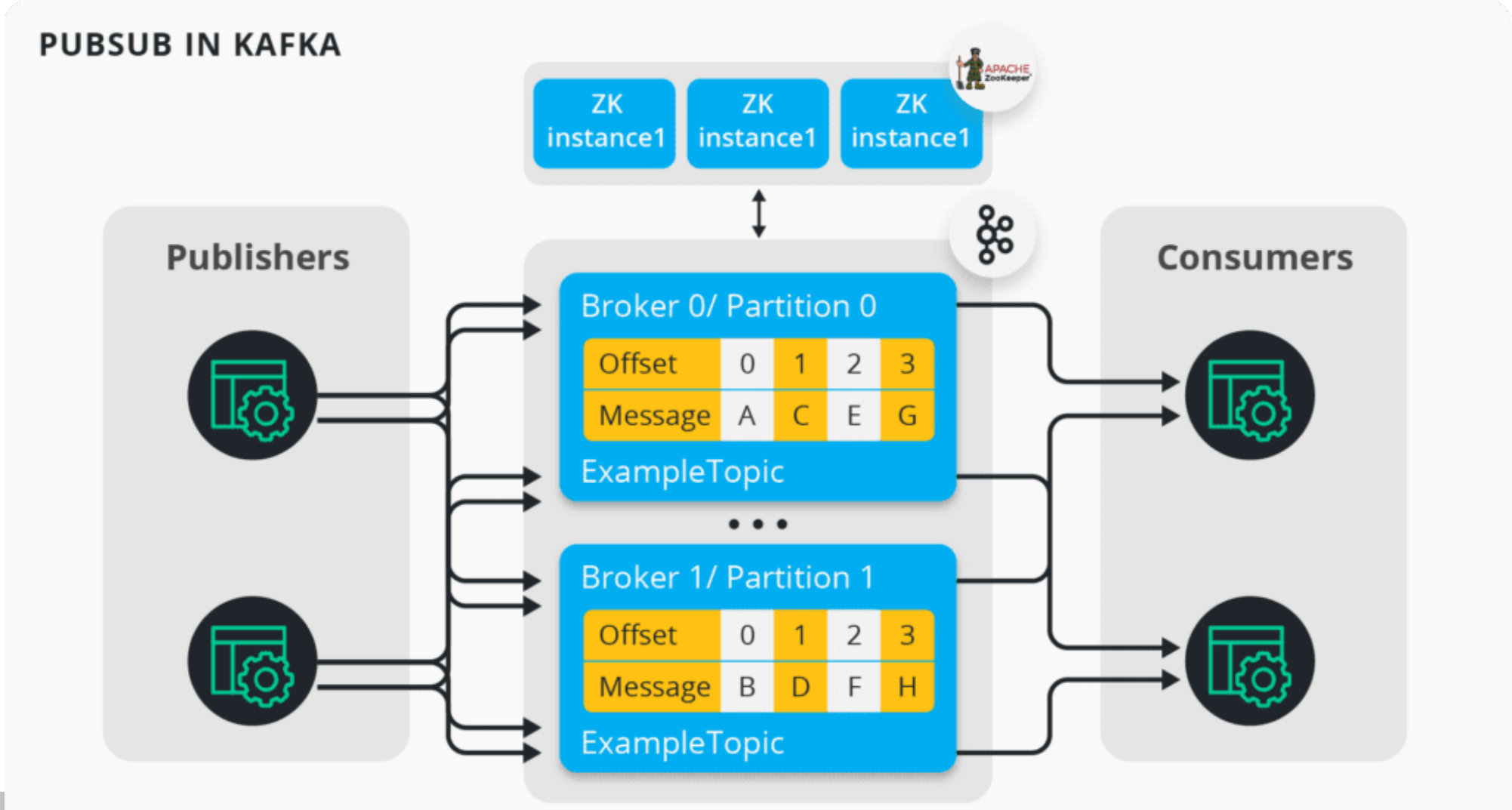
Publisher-Subscriber Pattern: Case Studies



Publisher-Subscriber Pattern: Case Studies



Publisher-Subscriber Pattern: Case Studies



https://solace.com/blog/solace-pubsub-vs-kafka-comparison-publish-subscribe-messaging-pattern/

Publisher-Subscriber Pattern: Benefits

- 다수의 컴포넌트에게 동시에 변경 공지를 할 수 있음
  - 하나의 게시자**(publisher)**가 다수의 구독자**(subscriber)**에게 상태 변경 통지
- **GUI** 인터페이스를 쉽게 만들 수 있음
  - **GUI** 빌더나 프레임워크를 쉽게 만들 수 있음
- 확장성**(Scailability)**과 신뢰성**(Reliability)** 증대
  - 게시자**(publisher)**와 구독자**(subscriber)**를 필요 시 추가할 수 있음

https://www.bmc.com/blogs/pub-sub-publish-subscribe/

Publisher-Subscriber Pattern: Liabilities

- **Non-deterministic** 한 문제
  - 미들웨어를 통하기 때문에 의도한대로 전달하지 못할 수도 있음
  - 흐름이 예측 되지 않을 가능성이 있고, 디버깅에 어려움이 있음
- 이벤트 핸들러와 프로세스 로직이 **Tightly coupled** 되어 있음
- 오디오 및 비디오과 같은 동기적인 스트림 처리에는 적절하지 않음

Question?

Seonah Lee

saleese@gmail.com