정확하고, 우아하게!
Reactive를 품은 Kafka 메시지

이병찬
alchemistless@gmail.com

$ whoami

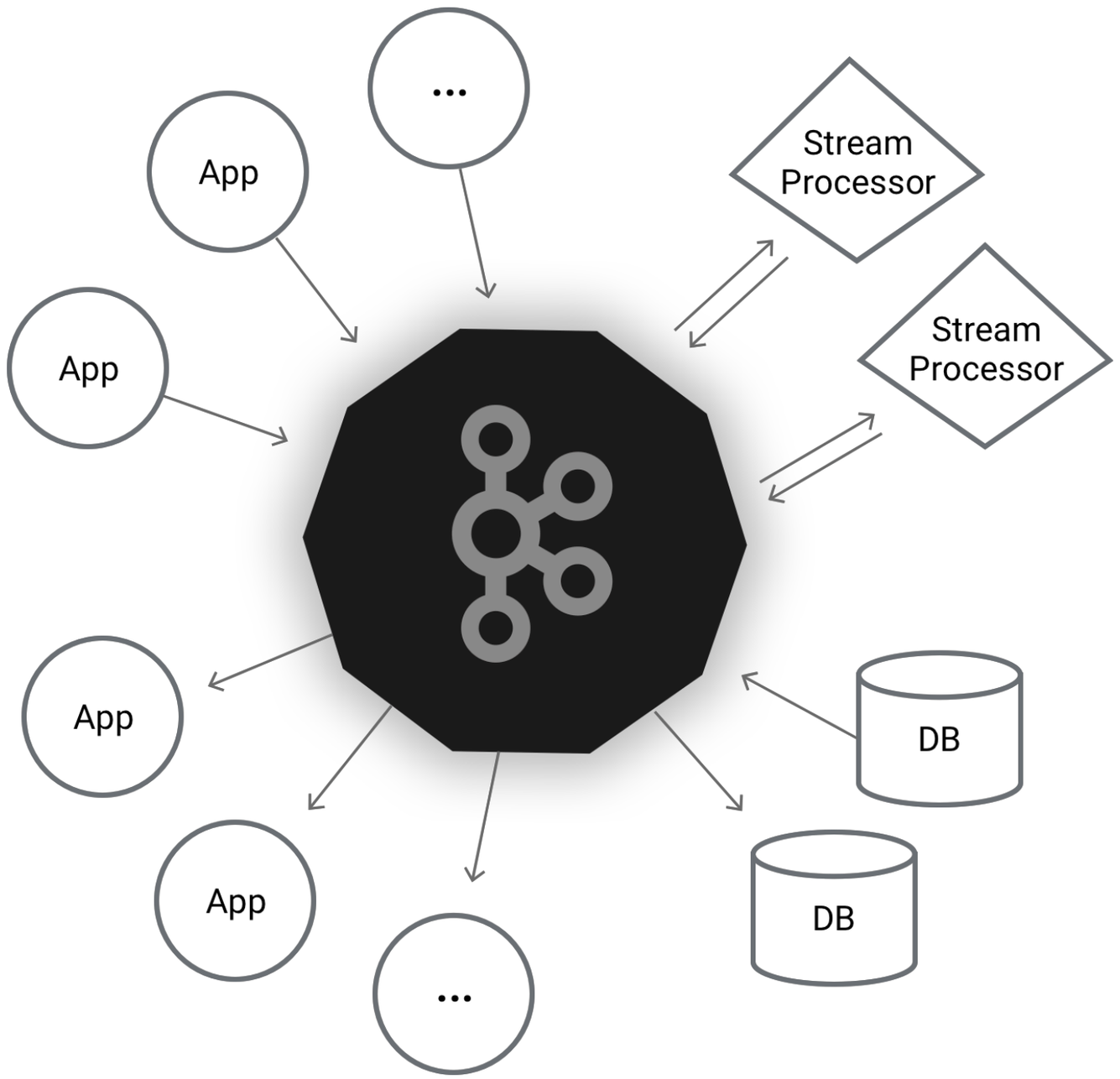
- NHN 모니터링플랫폼개발팀
  - NHN 서버 모니터링시스템 개발
  - TOAST 모니터링 상품 개발
- 아름다운 프로그램을 위하여.
- 집에서 즐기는 취미 코딩
- https://github.com/EleganceLESS


spring.camp
Korea
Spring
User
Group

Kafka 그리고 Spring

APACHE kafka®

A distributed streaming platform



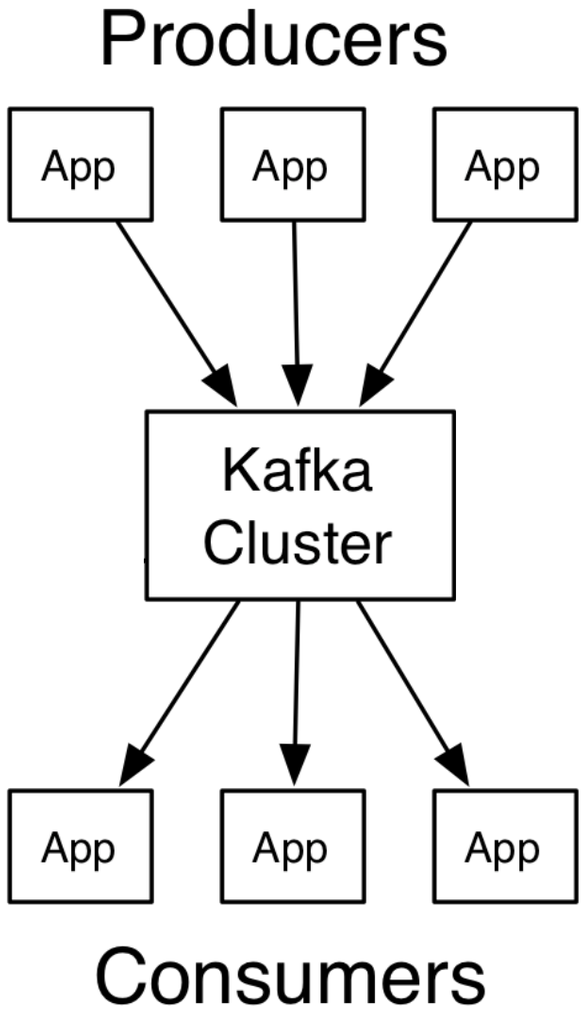
Spring에서 Kafka를 사용하는 방법

- **Spring Kafka**

Provides Familiar Spring Abstractions for Apache Kafka

```
@KafkaListener(topics = "myTopic")
public void listen(ConsumerRecord<?, ?> record) throws Exception {
    logger.info(record.toString());
}
```

Streaming Platform의 본질?



Streaming Platform의 본질?

Observable / Flux 의 본질?

Async<Stream<T>>

Streaming Platform의 본질?

Flux<T> !

Reactor Kafka!

```
<dependency>
    <groupId>io.projectreactor.kafka</groupId>
    <artifactId>reactor-kafka</artifactId>
    <version>1.1.0.RELEASE</version>
</dependency>


dependencies {
    compile "io.projectreactor.kafka:reactor-kafka:1.1.0.RELEASE"
}
```

Consumer


```
public Flux<ReceiverRecord<String, String>> consume() {
    return KafkaReceiver.create(RECEIVER_OPTIONS)
            .receive();
}
```

Producer


```
public Flux<SenderResult<String>> produce(Publisher<String> publisher) {
    return KafkaSender.create(SENDER_OPTIONS)
            .send(publisher.map(o -> SenderRecord.create(…)));
}
```

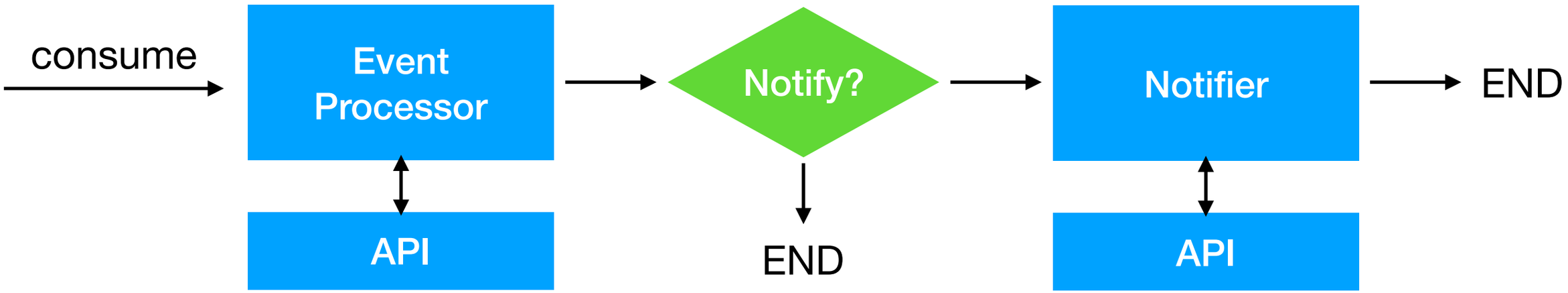
본격적으로 만들어 봅시다!

기본 기능 구현하기

이벤트가 감지되면 그 정보로 메시지를 생성하고 발행하자.

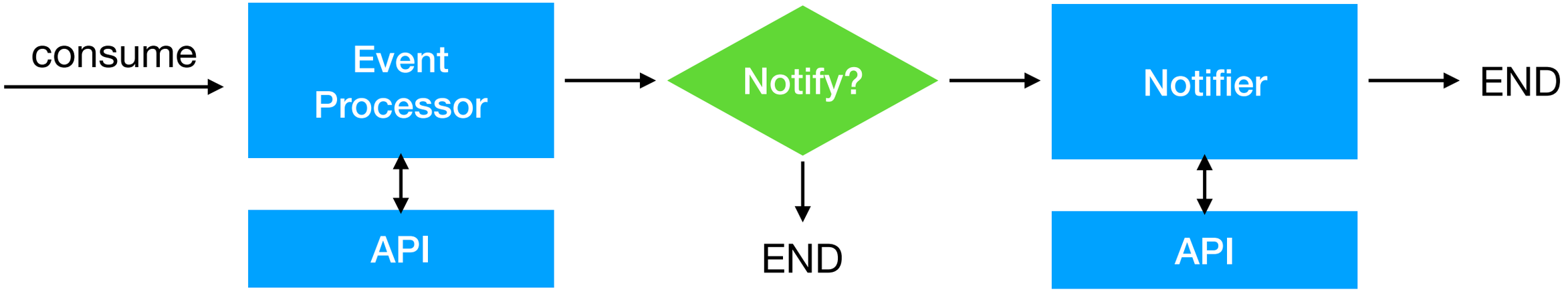
기본 기능 구현하기

처리기는 메시지를 읽고 이벤트 정보를 DB에 기록 후,
통보가 필요하면 발송 요청을 보내자.



기본 기능 구현하기

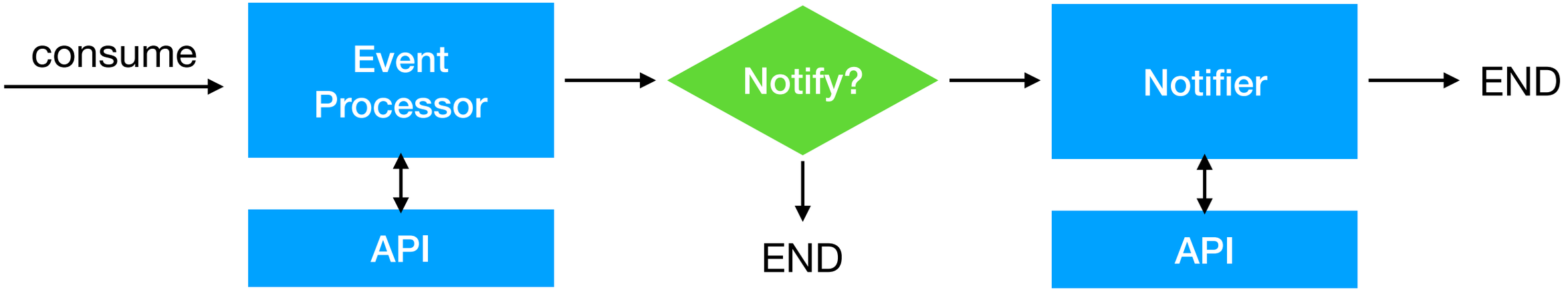
Step 1




```
public void process() {
    consume().flatMap(this::recordToEventObject)
             .flatMap(this::saveEvent)
             .flatMap(this::getReceivers)
             .flatMap(this::notify)
             .flatMap(this::saveResult)
             .subscribe();
}
```

기본 기능 구현하기

Step 1

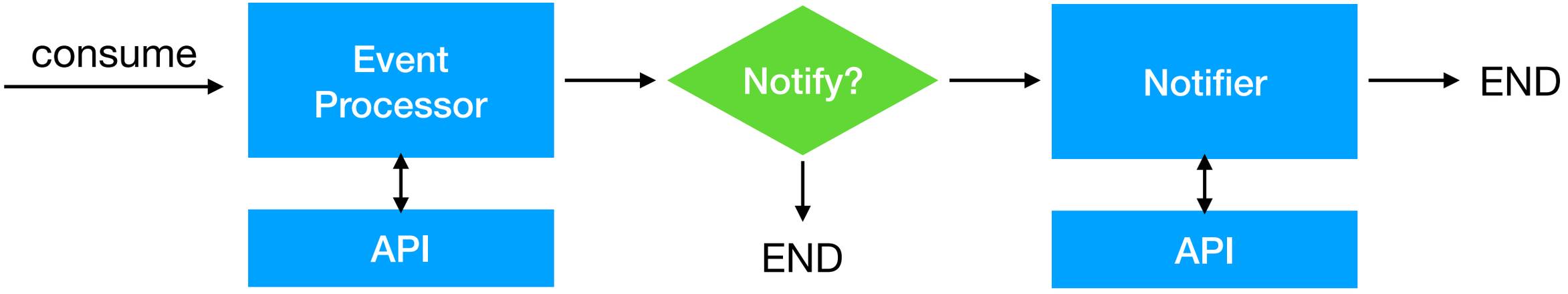



```
public void process() {
    consume().flatMap(this::recordToEventObject)
             .flatMap(this::saveEvent)
             .flatMap(this::getReceivers)
             .flatMap(this::notify)
             .flatMap(this::saveResult)
             .subscribe();
}
```

구현 끝!

기본 기능 구현하기

Step 1




```
public void process() {
    consume().flatMap(this::recordToEventObject)
             .flatMap(this::saveEvent)
             .flatMap(this::getReceivers)
             .flatMap(this::notify)
             .flatMap(this::saveResult)
             .subscribe();
}
```

구현 끝?

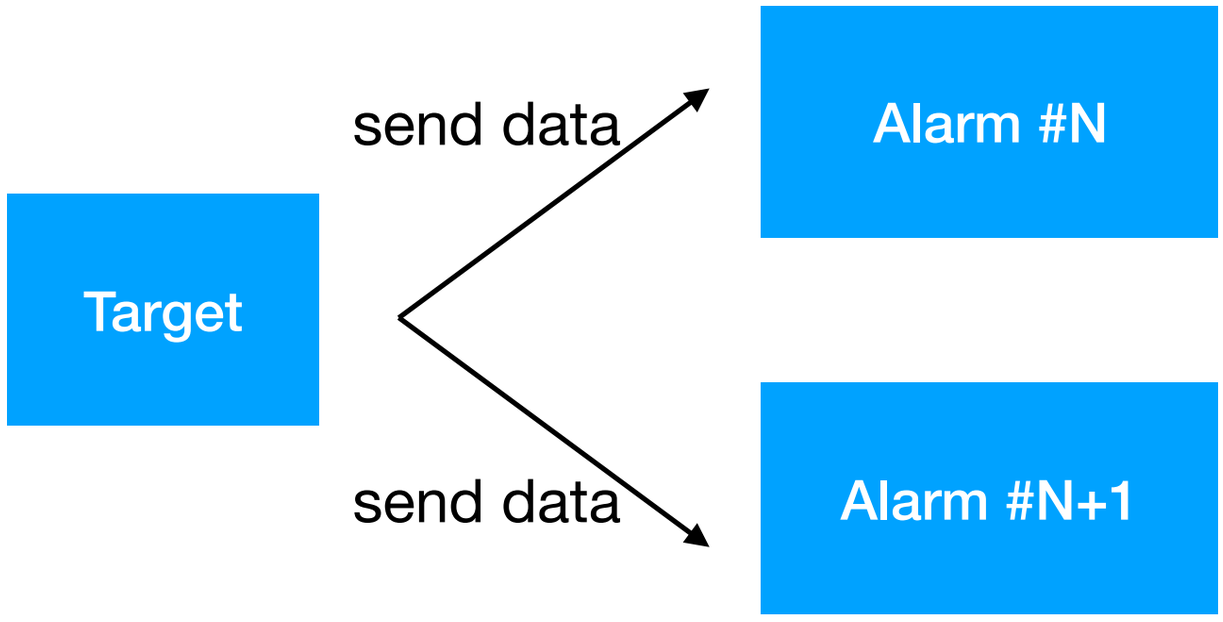
현실의 요구는 디테일에 있다

메시지 중복 제거하기

이 구조는 별 문제가 없습니다.

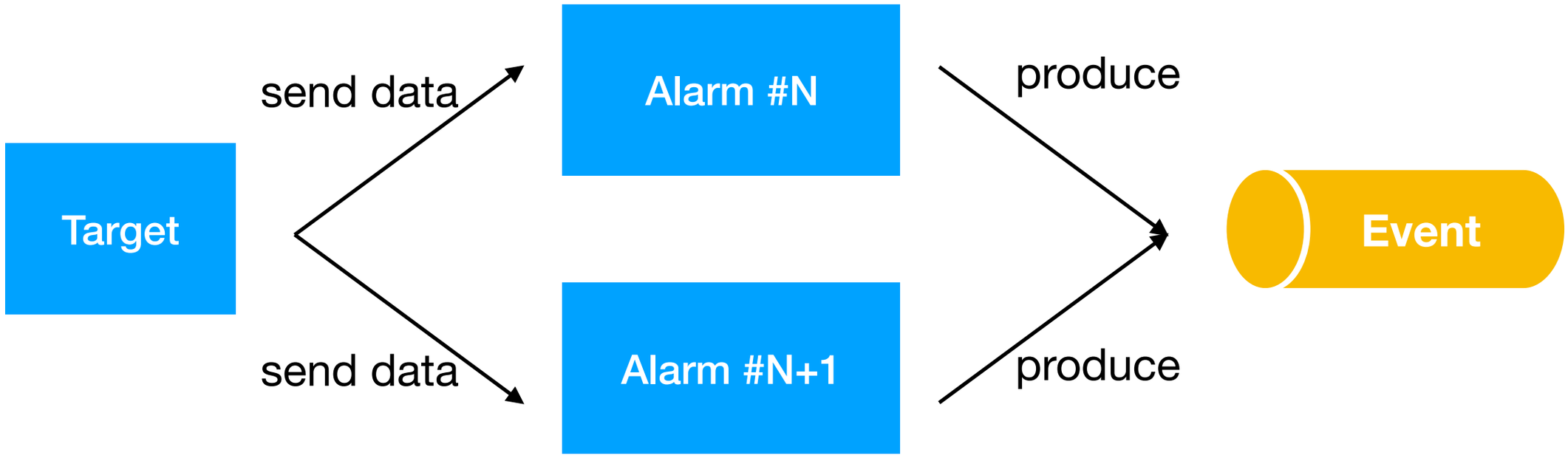
메시지 중복 제거하기

하지만 이벤트 메시지를 만드는 주체가 여럿이 된다면?



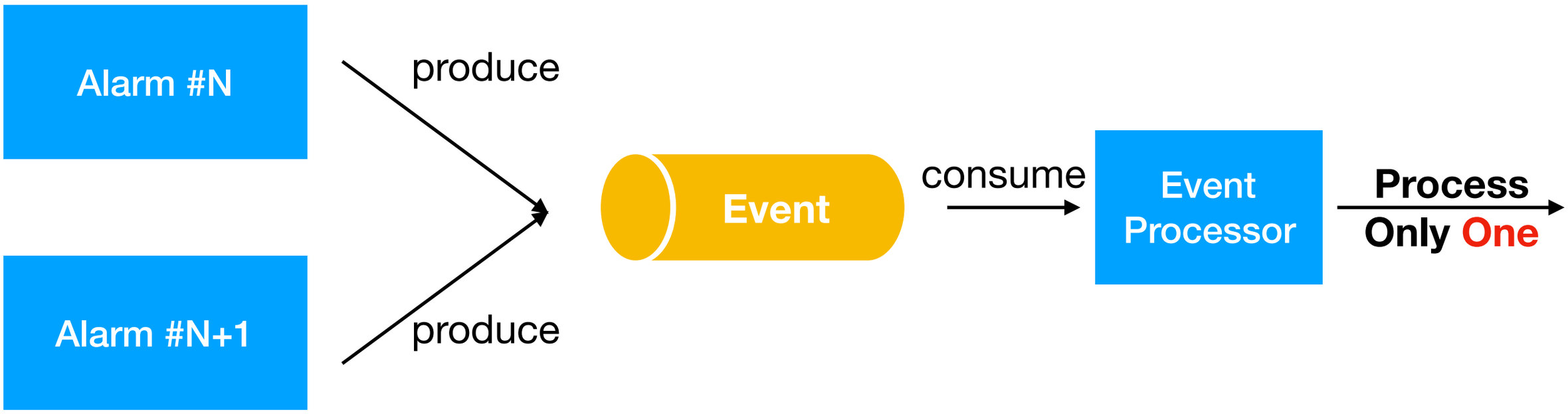
메시지 중복 제거하기

하지만 이벤트 메시지를 만드는 주체가 여럿이 된다면?

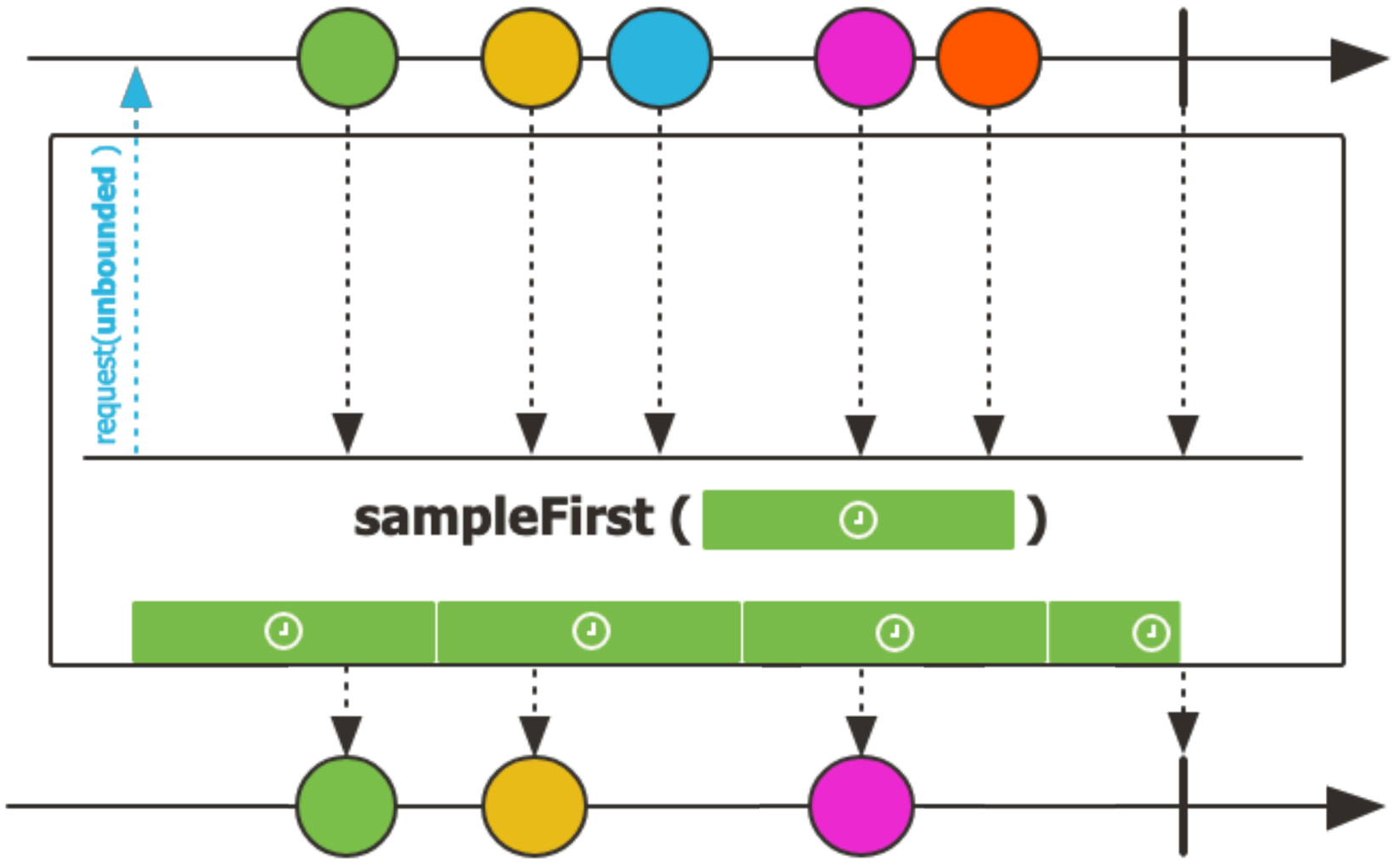


메시지 중복 제거하기

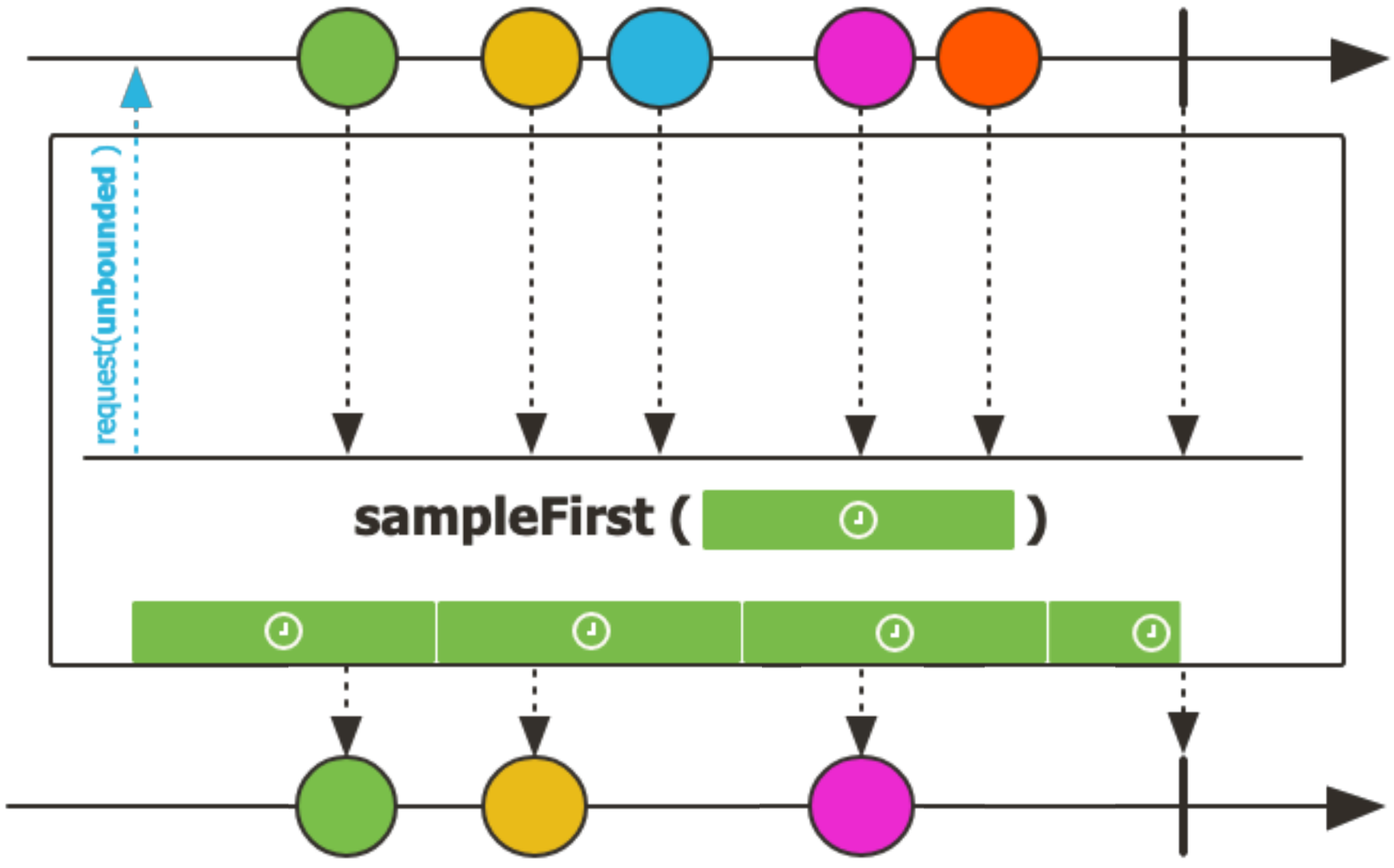
동일한 이벤트가 중복 감지되는 건은 무시하자.



Flux Operator : sampleFirst()

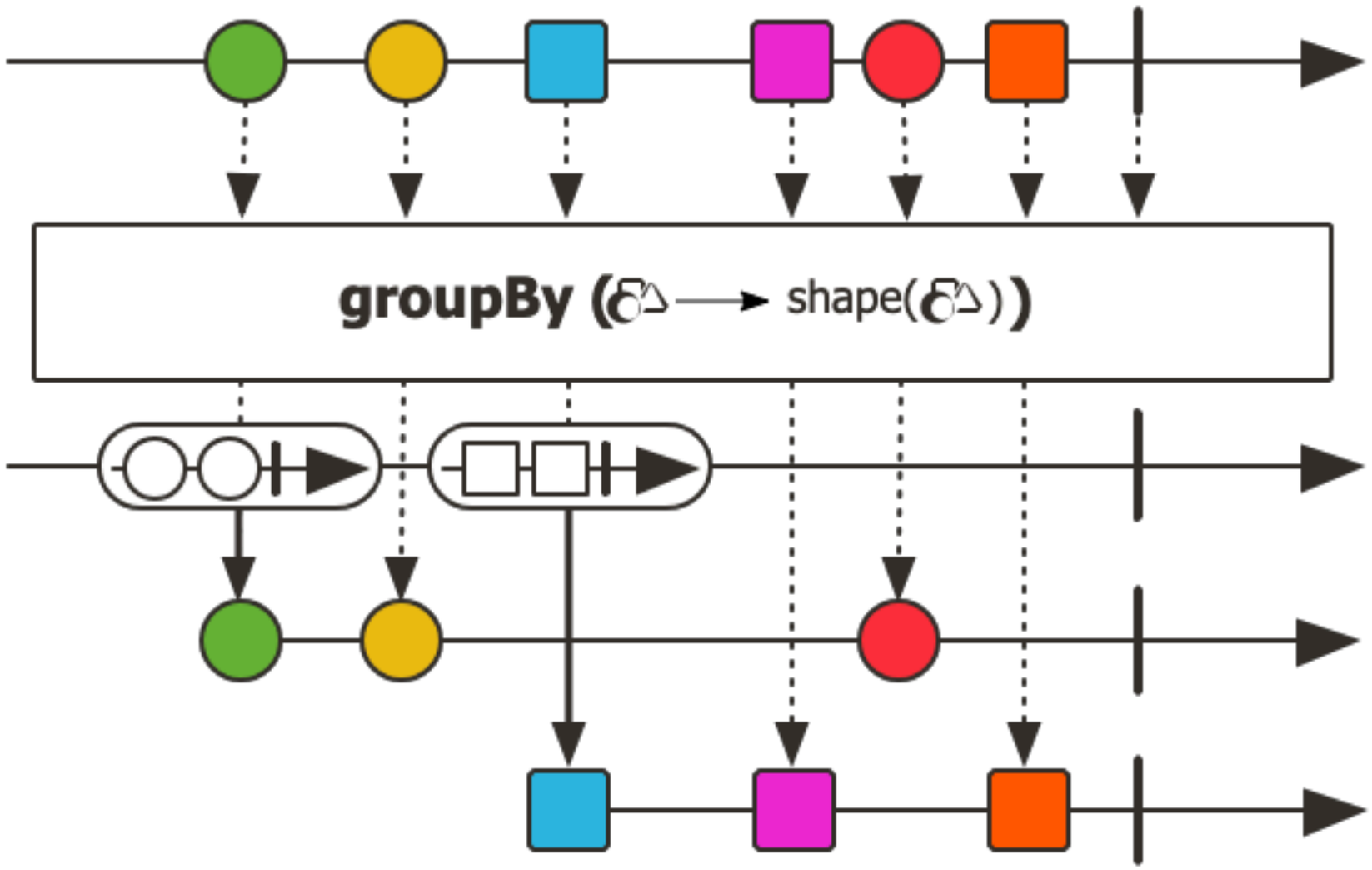


Flux Operator : sampleFirst()



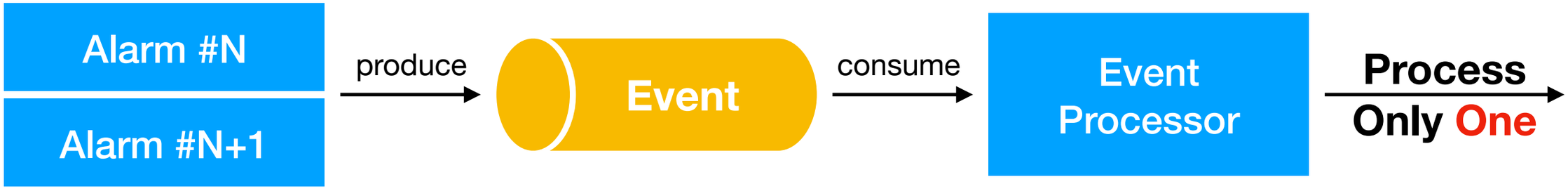
이걸로 해결이 될까요?

Flux Operator : groupBy()



메시지 중복 제거하기

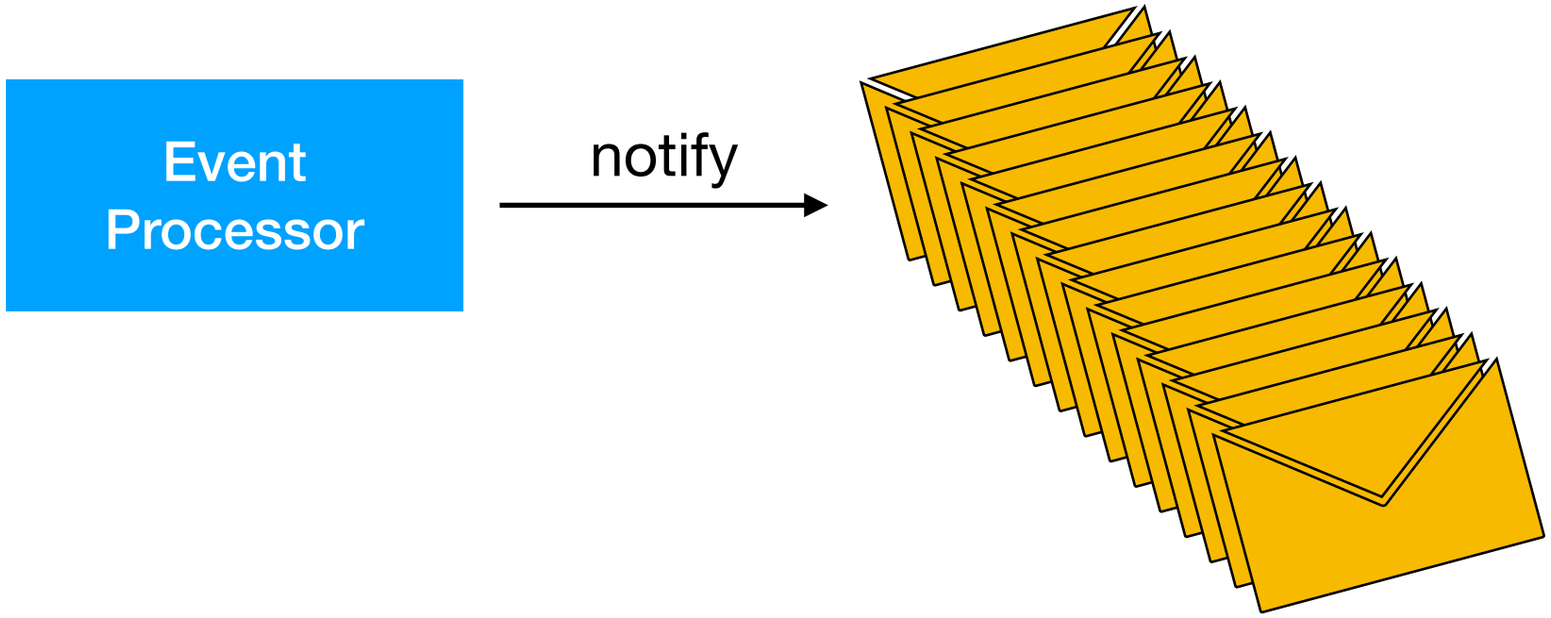
Step 2




```
public void process() {
    consume().flatMap(this::recordToEventObject)
             .groupBy(Message::key)
             .flatMap(flux -> flux.sampleFirst(Duration.ofSeconds(30)))
             .flatMap(this::saveEvent)
             .flatMap(this::getReceivers)
             .flatMap(this::notify)
             .flatMap(this::saveResult)
             .subscribe();
}
```

데이터 모아서 처리하기

한 순간에 메시지 발송 요청이 급격히 늘어나게 되면?

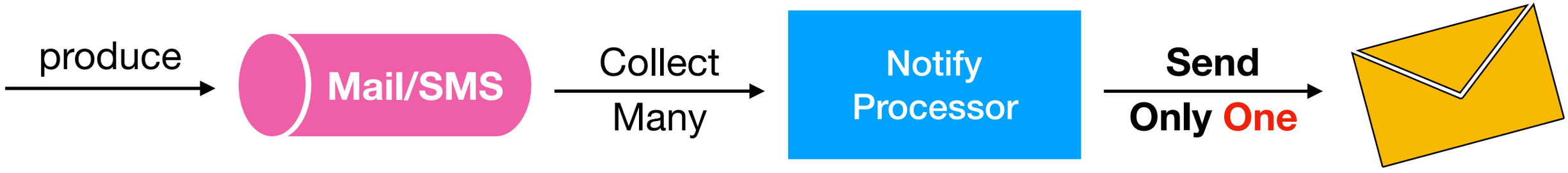


데이터 모아서 처리하기

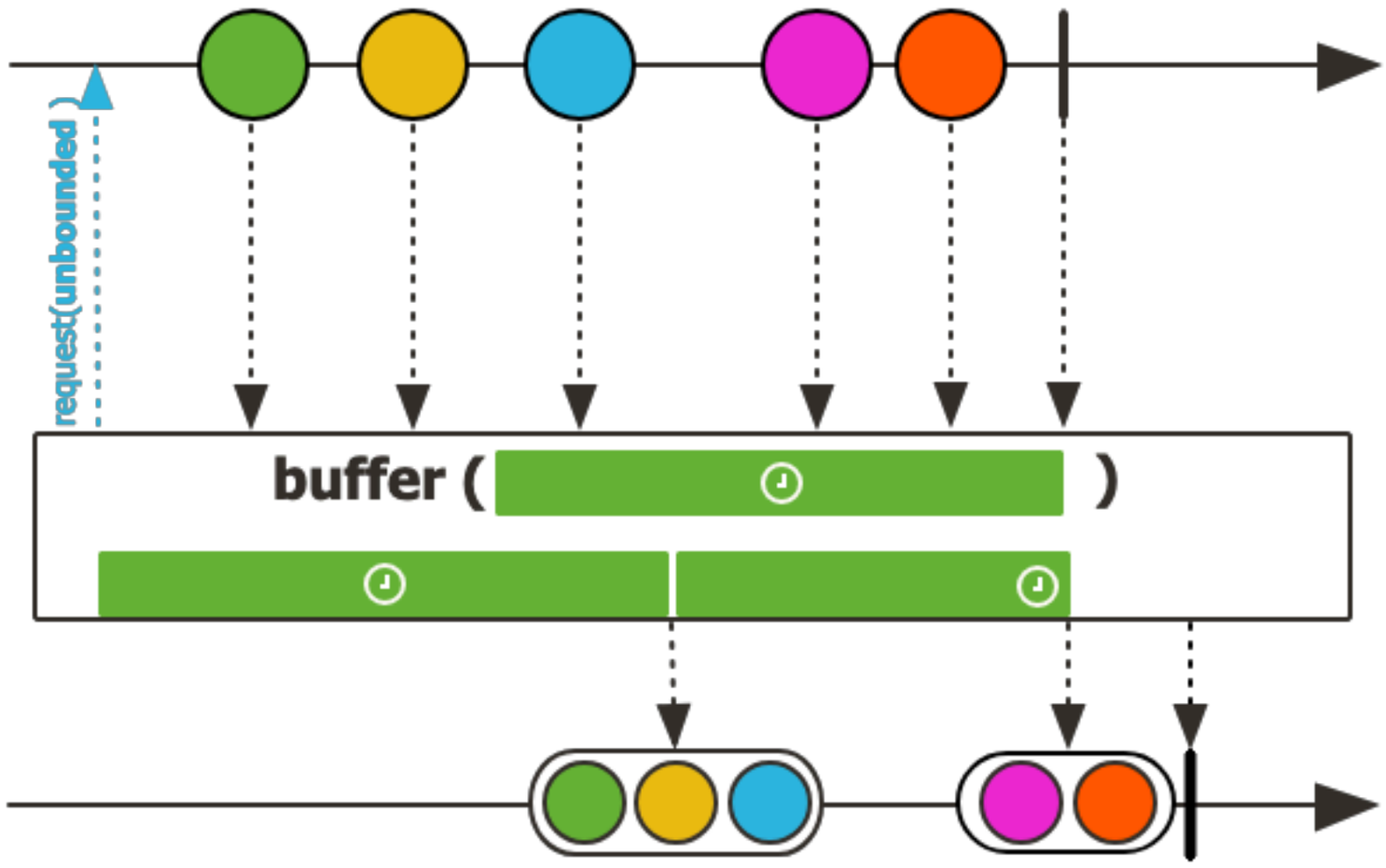
기준 시간동안 발생한 여러 이벤트는 하나의 메시지로 모아서 통지하자.

데이터 모아서 처리하기

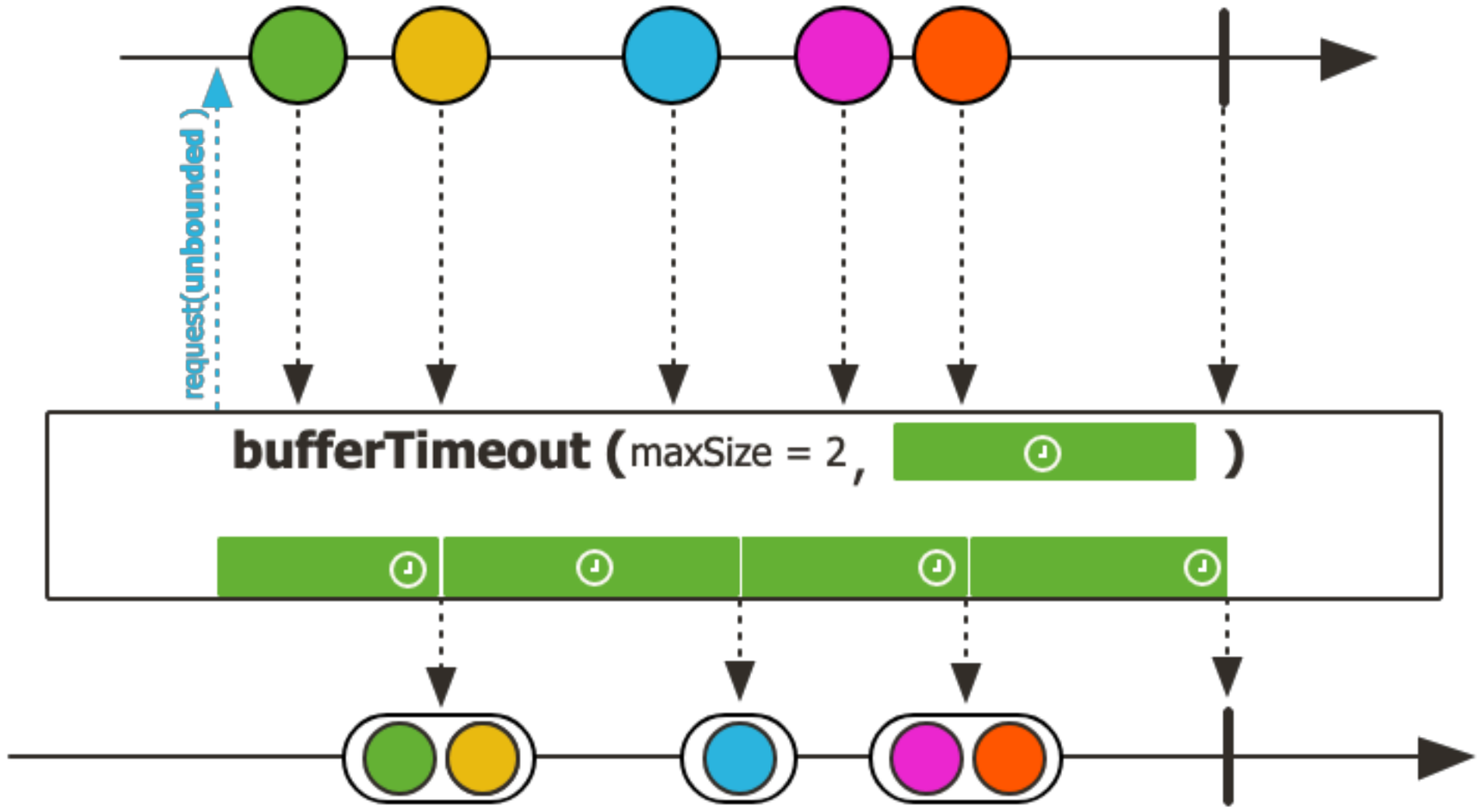
기준 시간동안 발생한 여러 이벤트는 하나의 메시지로 모아서 통지하자.



Flux Operator : buffer()

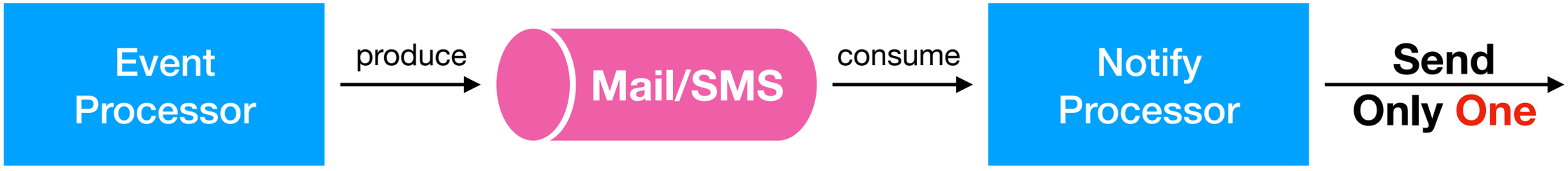


Flux Operator : bufferTimeout()



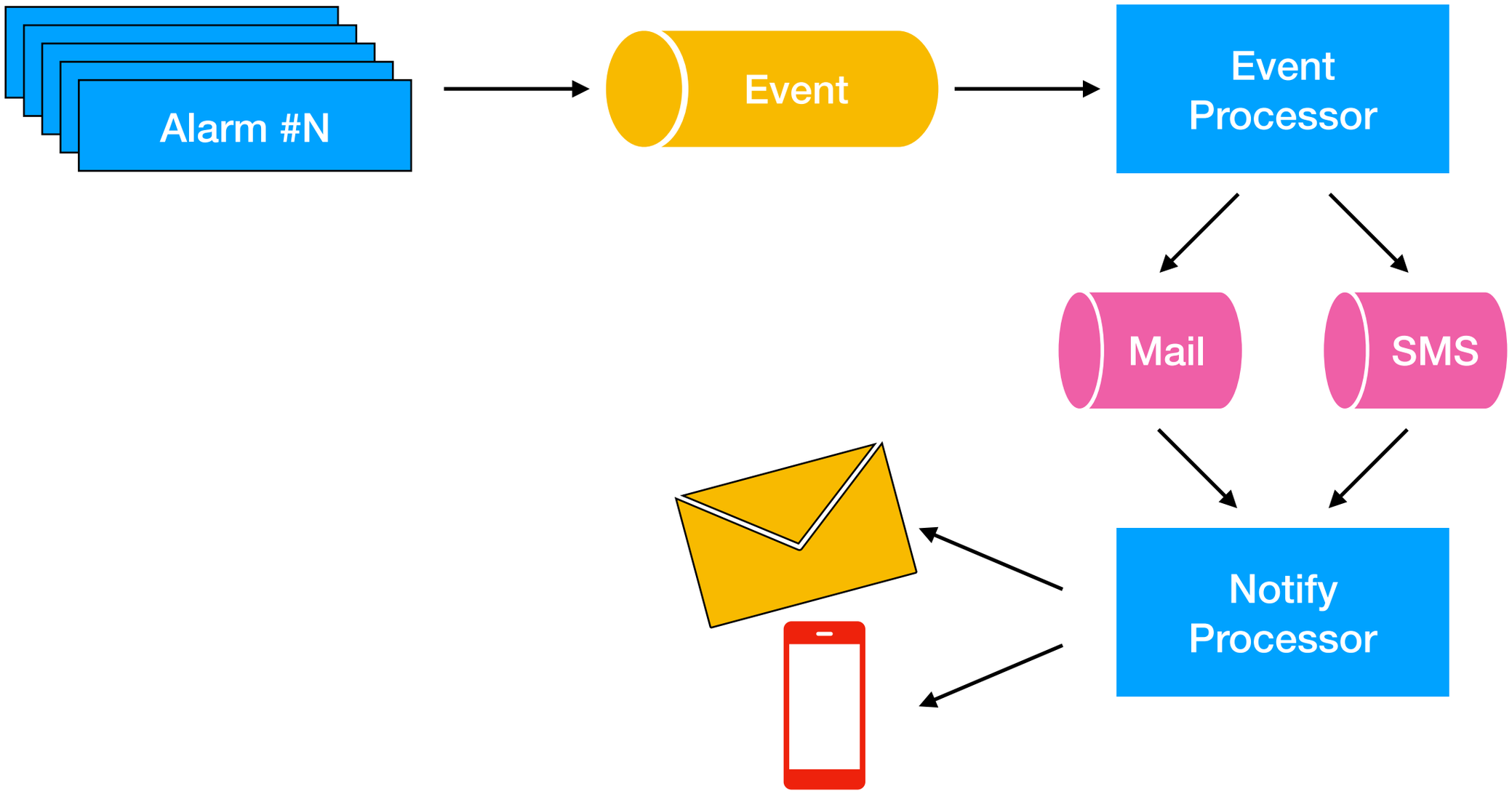
데이터 모아서 처리하기

Step 3

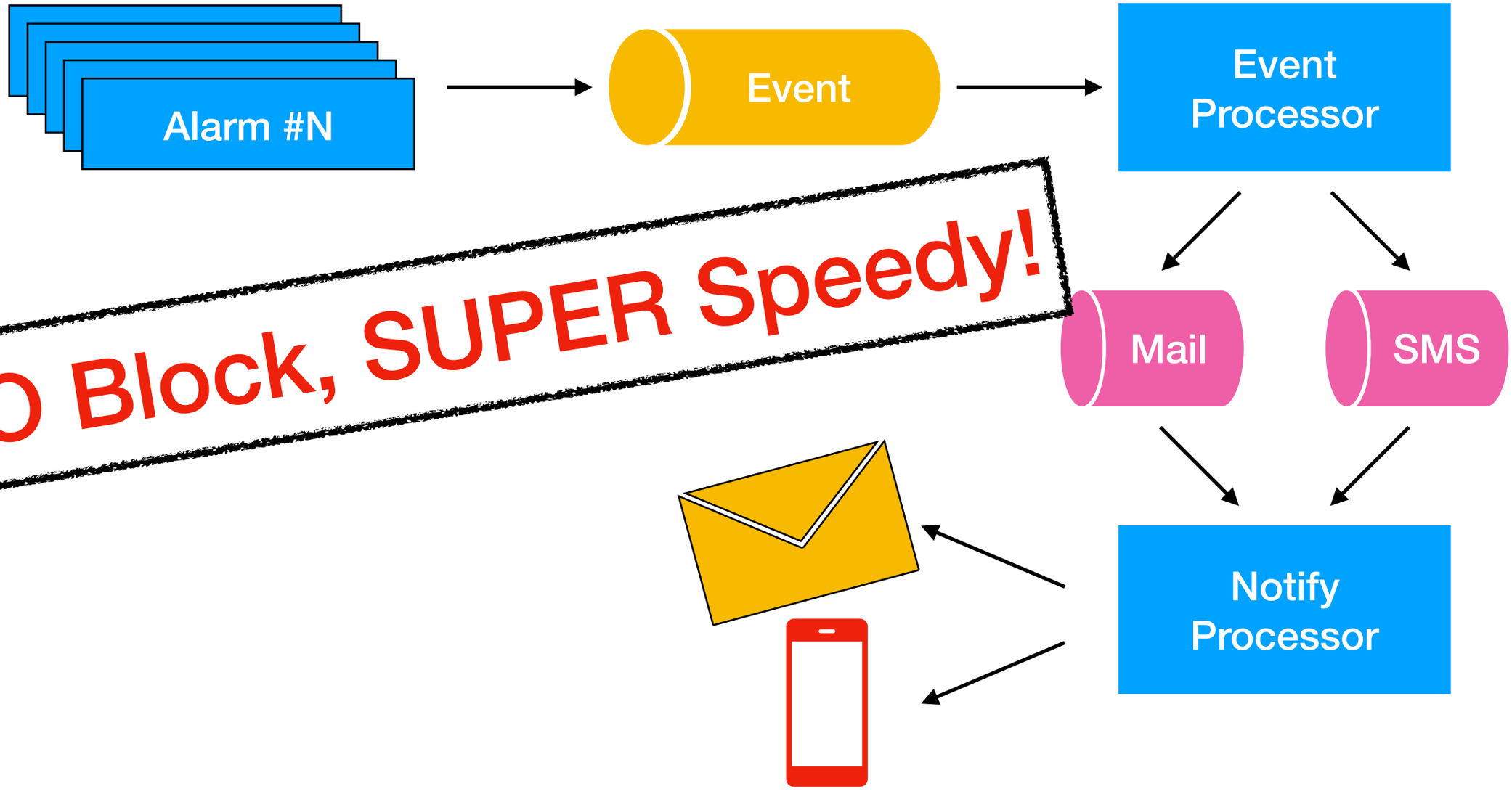



```
public void process() {
    consume().flatMap(this::recordToNotifyObject)
             .groupBy(Message::key)
             .flatMap(flux -> flux.buffer(Duration.ofSeconds(30)))
             .flatMap(this::notify)
             .flatMap(this::saveNotifyResult)
             .subscribe();
}
```

메시지가 흘러가는 큰 그림!

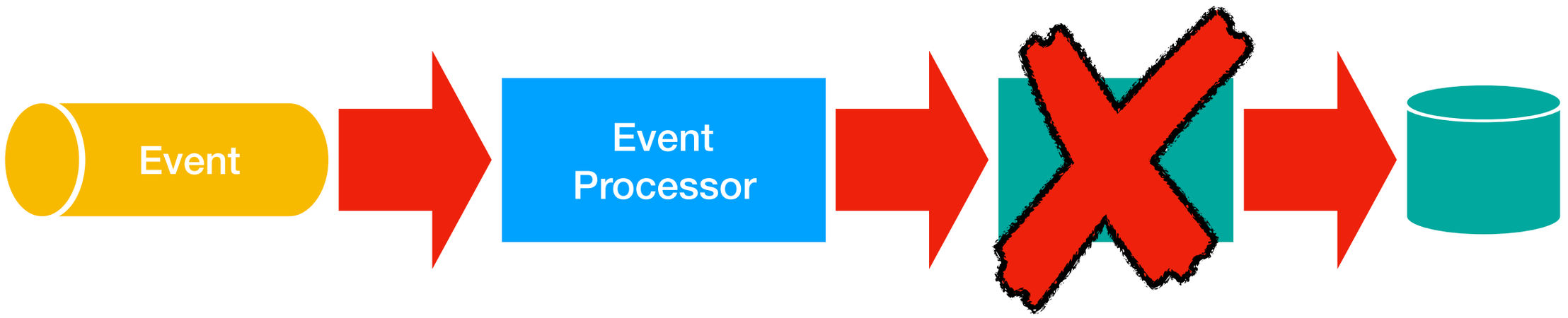


메시지가 흘러가는 큰 그림!

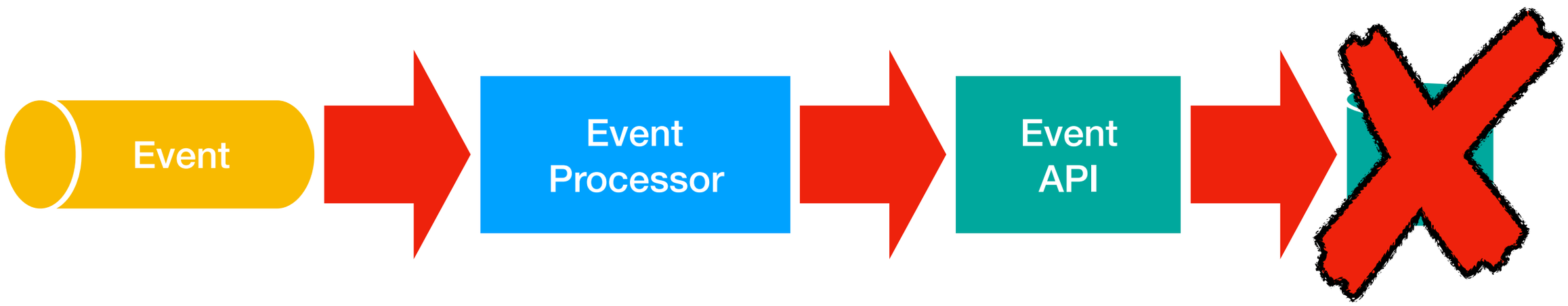


Many Many Many Events ?

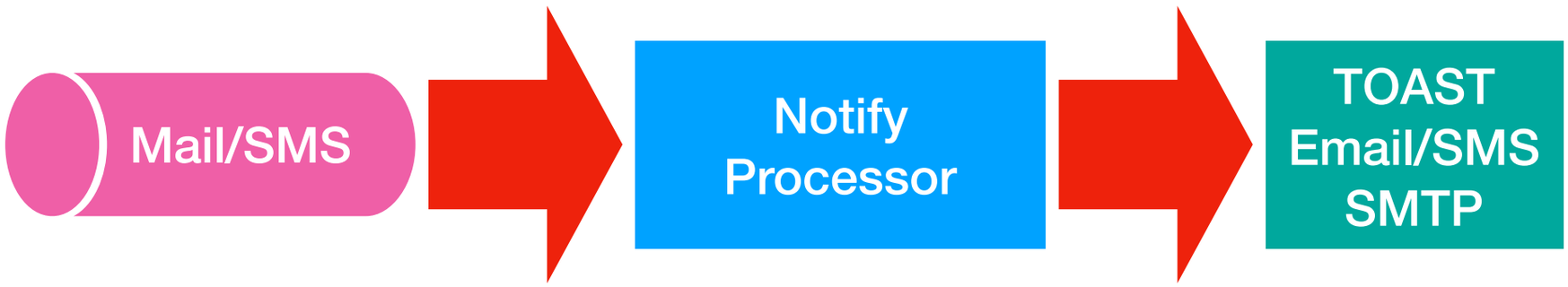
Many Many Many Events ?



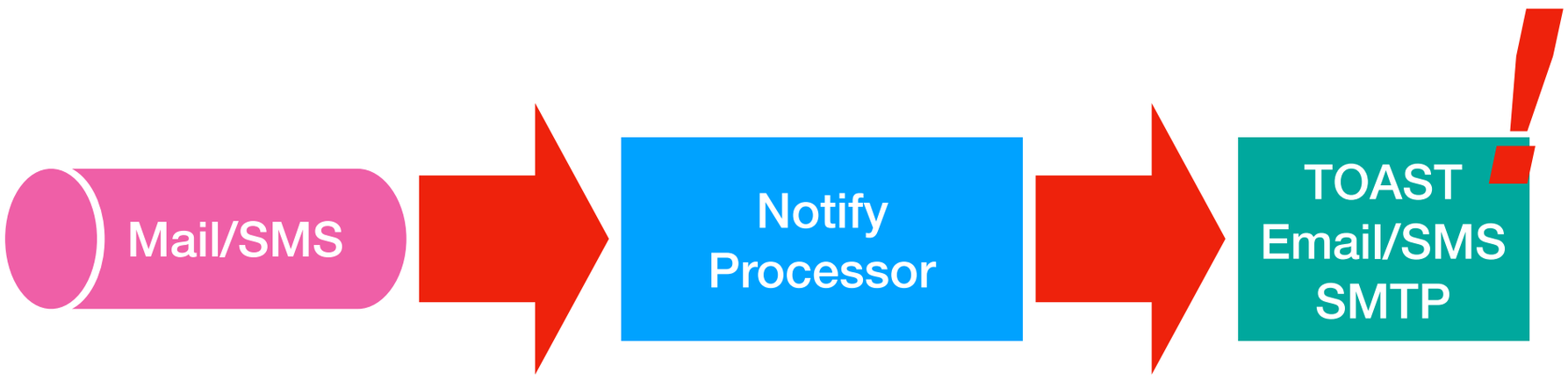
Many Many Many Events ?



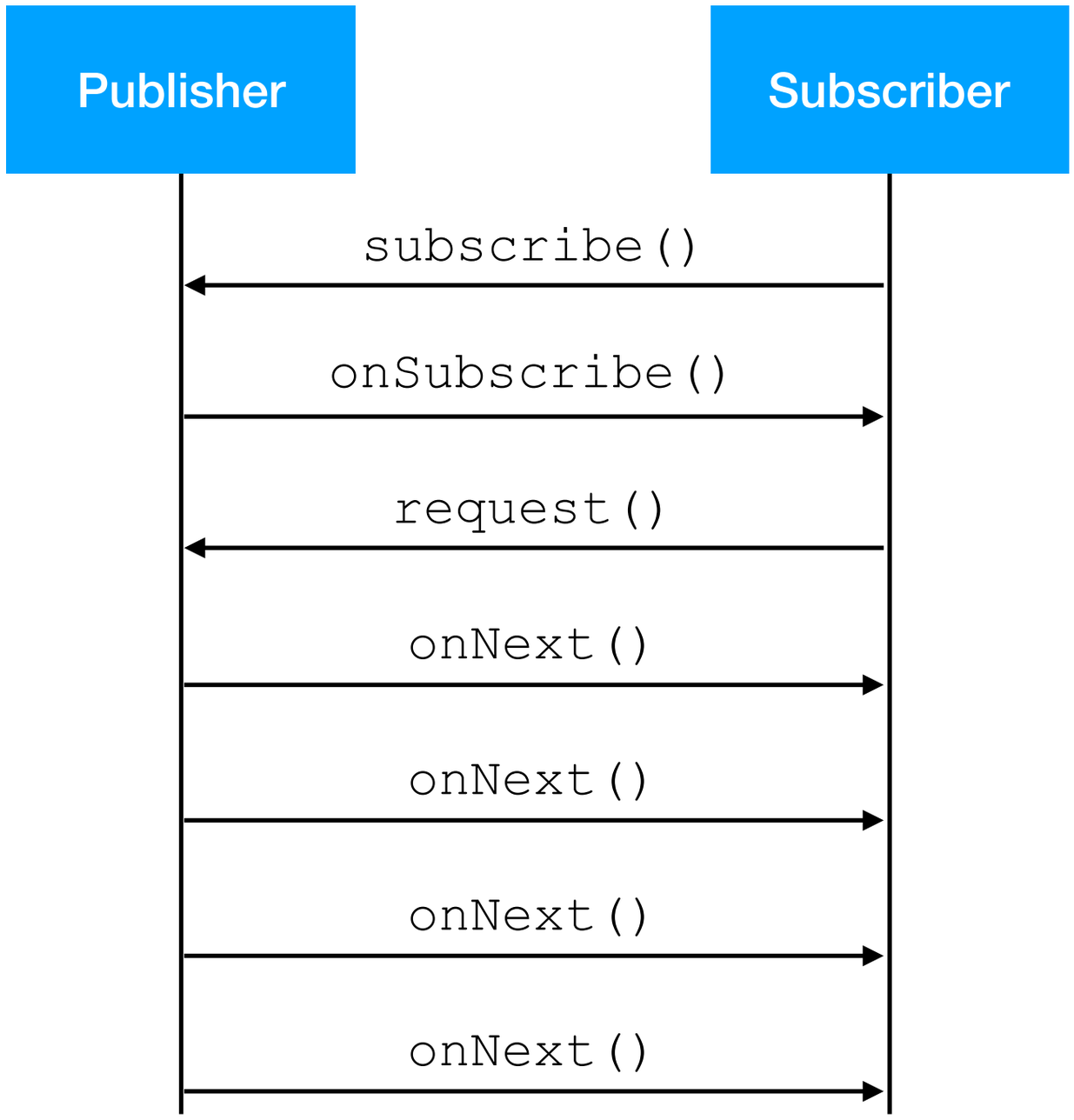
Many Many Many Events ?



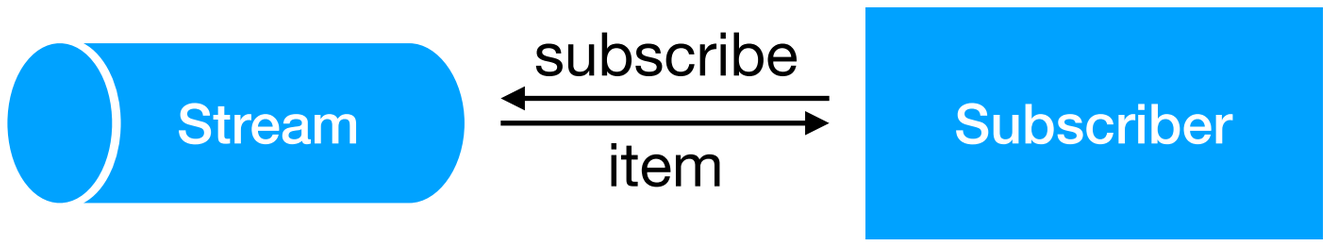
Many Many Many Events ?



Flux에서 데이터가 흐르는 과정

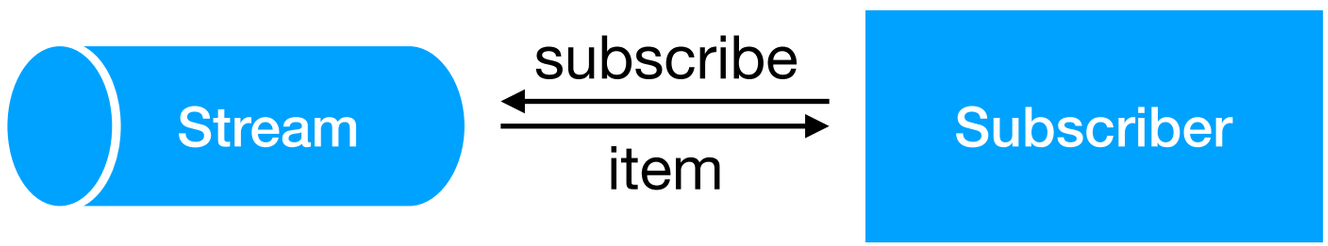


Flux에서 데이터가 흐르는 과정




```
.subscribe(this::doSomething);
```

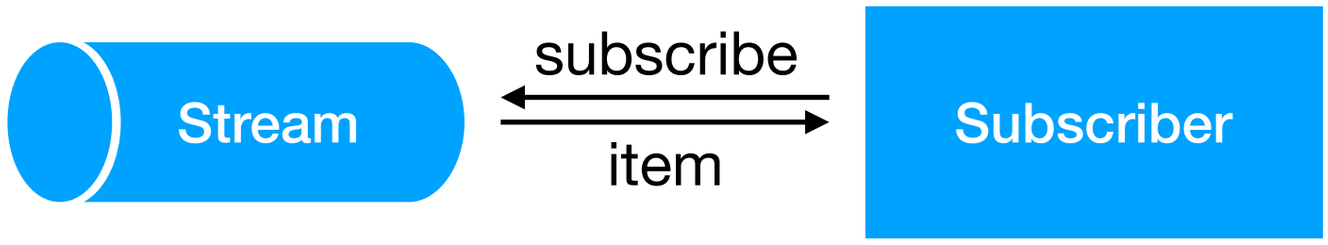
Flux에서 데이터가 흐르는 과정




```
.subscribe(this::doSomething);
```

```
reactor.core.publisher.Flux :
  public final Disposable subscribe(Consumer<? super T> consumer)
  public abstract void subscribe(CoreSubscriber<? super T> actual)
```

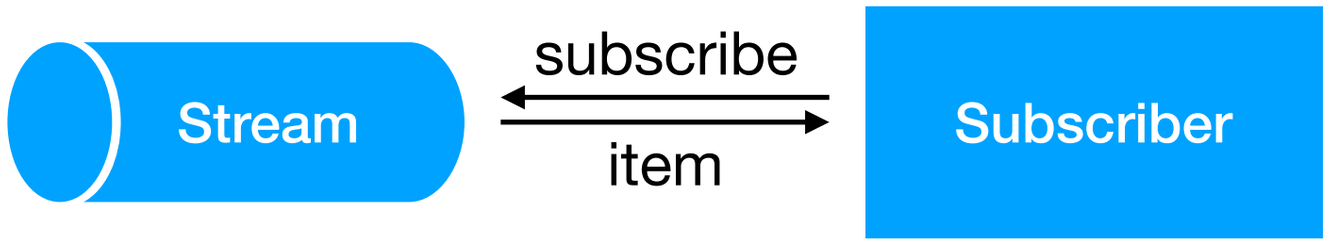
Flux에서 데이터가 흐르는 과정




```
.subscribe(this::doSomething);
```

```
reactor.core.publisher.LambdaSubscriber :
  public final void onSubscribe(Subscription s) {
    ...
    s.request(Long.MAX_VALUE);
    ...
  }
```

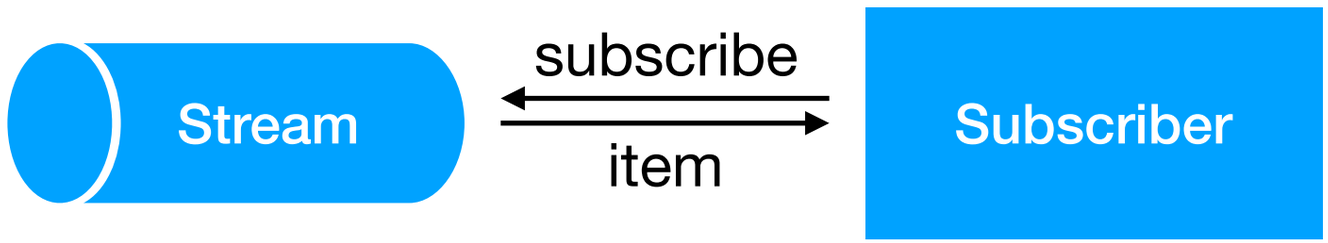
Flux에서 데이터가 흐르는 과정




```
.subscribe(this::doSomething);


org.reactivestreams.Subscription :
  public interface Subscription {
    public void request(n);
                                // implement with onNext(), onComplete()
  }
```

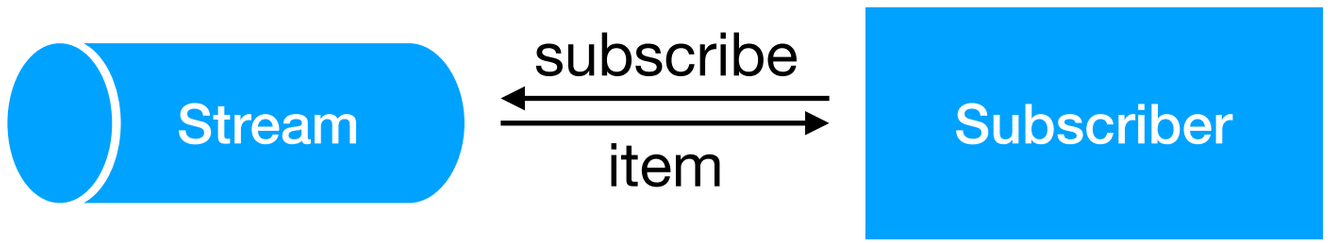
정해진 양 만큼만 처리하기




```
public class CustomSubscriber extends BaseSubscriber<~> {

    @Override
    protected void hookOnSubscribe(Subscription subscription) {
        request(1);
    }
}
```

정해진 양 만큼만 처리하기

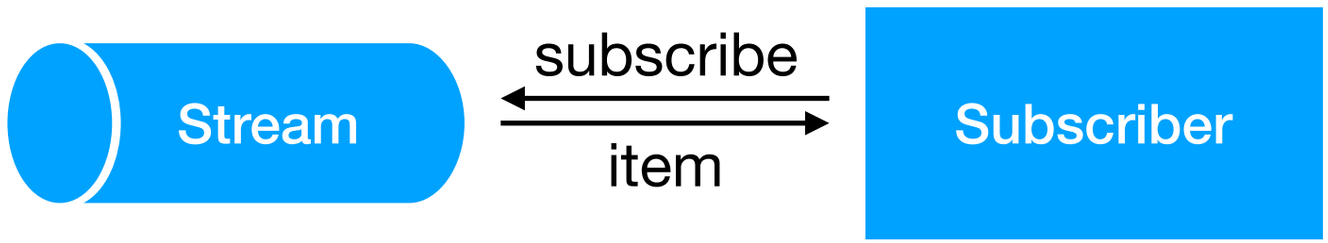



```
public class CustomSubscriber extends BaseSubscriber<~> {

    @Override
    protected void hookOnNext(ReceiverRecord<String, String> record) {
        Mono.just(record)
                .flatMap(/* Do Something */)
                .subscribe(r -> {
                    record.receiverOffset().acknowledge();
                    request(1);
                });
    }
}
```

비동기가 비동기가 아니게 되어버려…

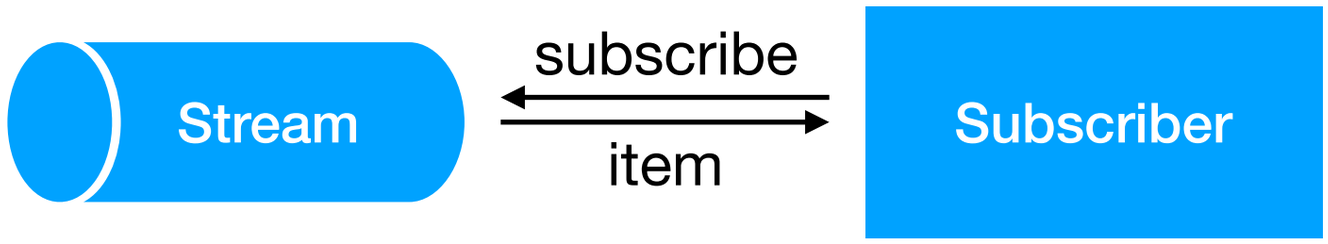
정해진 양 만큼만 처리하기




```
public class CustomSubscriber extends BaseSubscriber<~> {

    @Override
    protected void hookOnSubscribe(Subscription subscription) {
        request(10);
    }
}
```

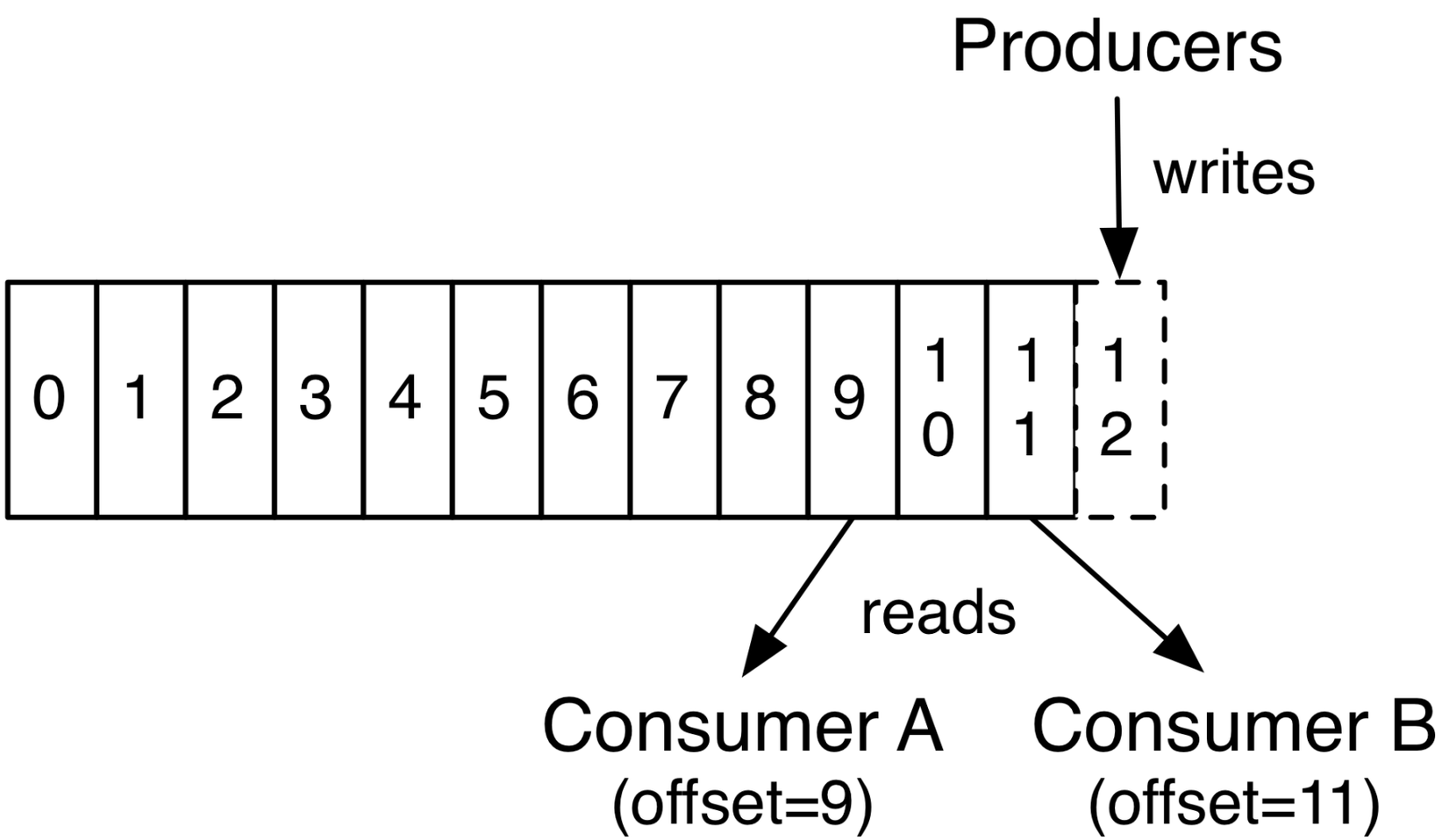
정해진 양 만큼만 처리하기



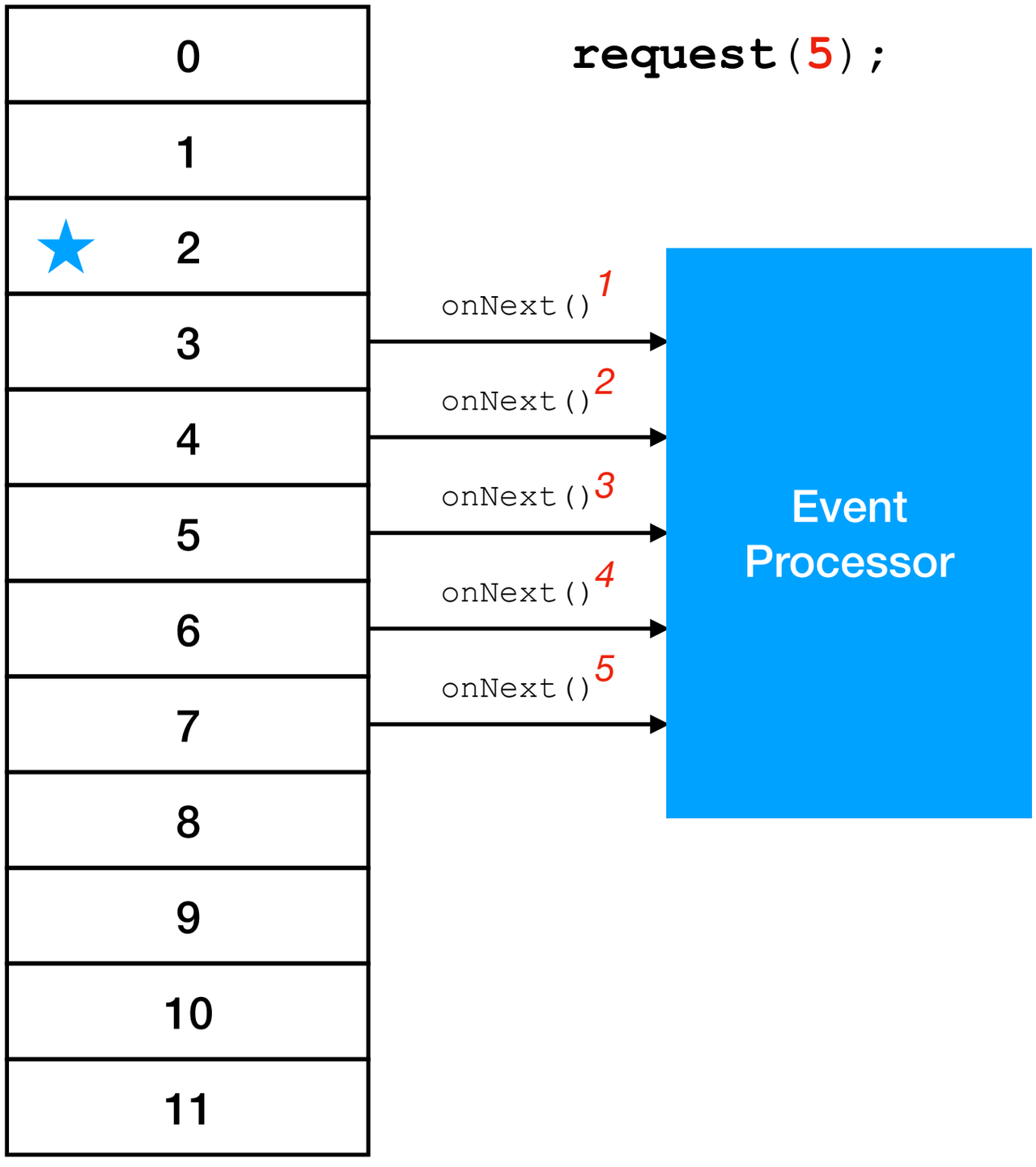

```
kafkaManager.consume(topicName)
    .subscribe(new CustomSubscriber(this::doSomething));
```

시간을 달리는 메시지

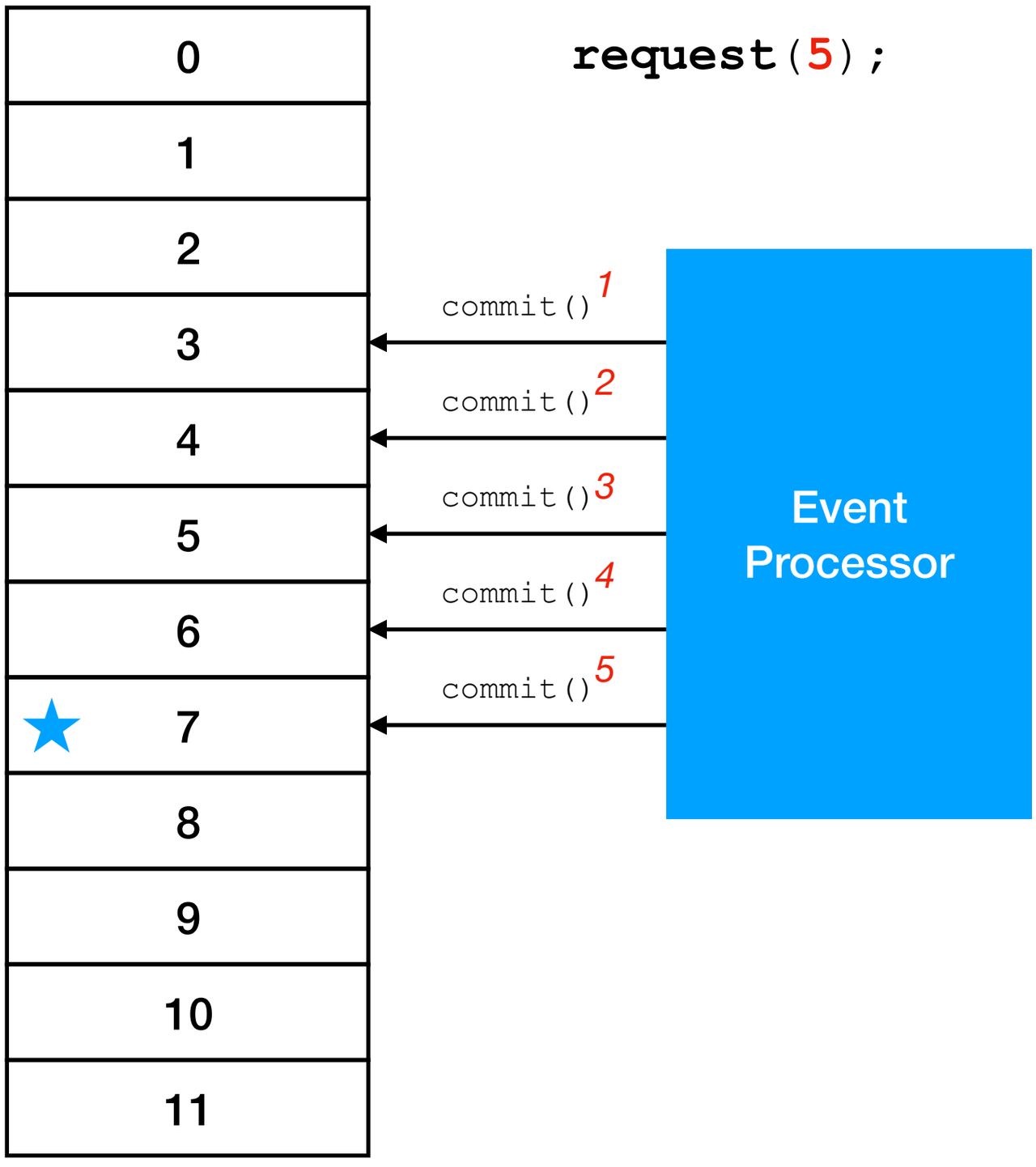

Alarm #N
produce
Stream
consume
commit
Event Processor


Producers
writes
0
1
2
3
4
5
6
7
8
9
10
11
12
reads
Consumer A
(offset=9)
Consumer B
(offset=11)

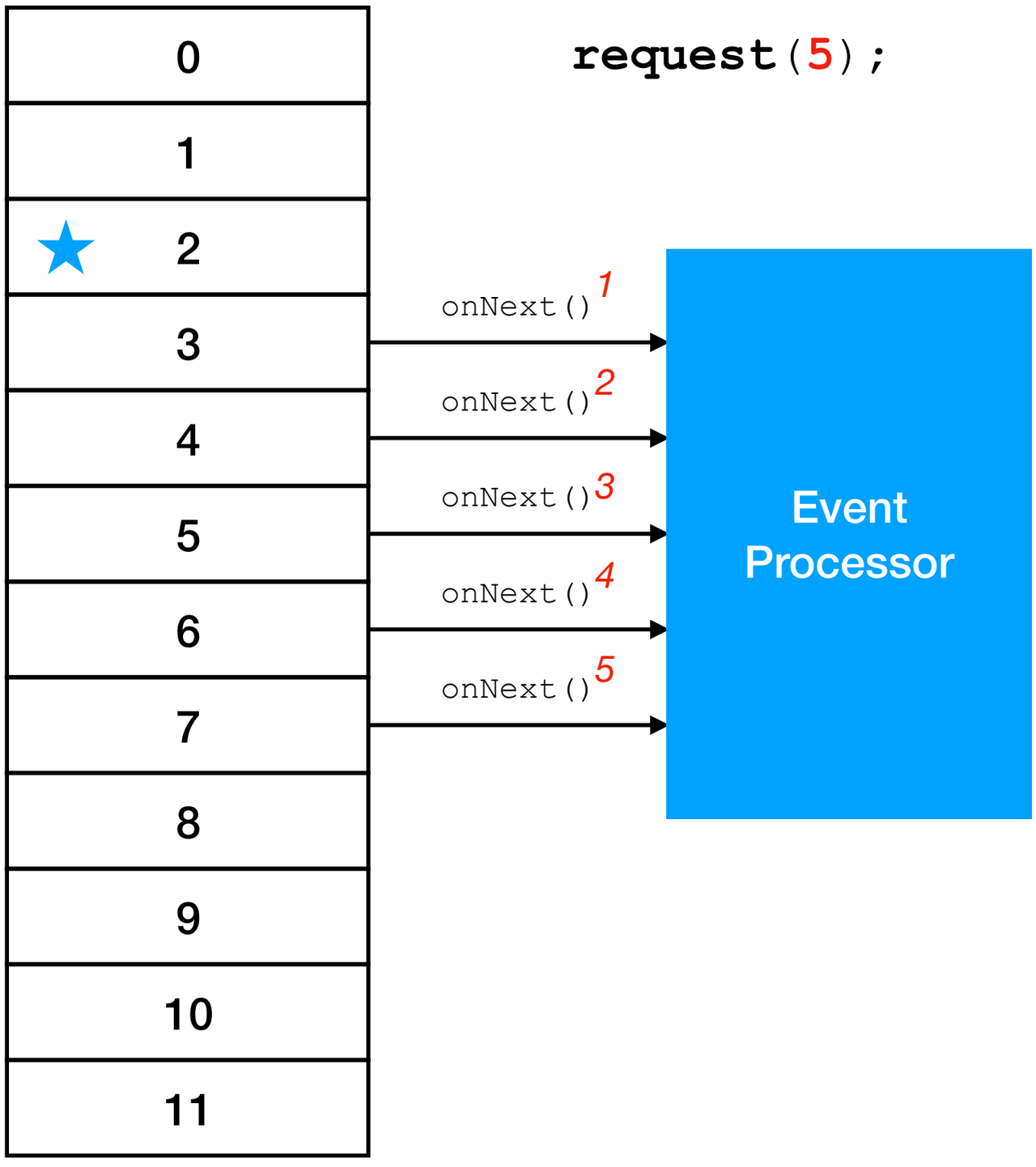
시간을 달리는 메시지


request(5);
0
1
2
3
4
5
6
7
8
9
10
11
onNext()1
onNext()2
onNext()3
onNext()4
onNext()5
Event
Processor

시간을 달리는 메시지

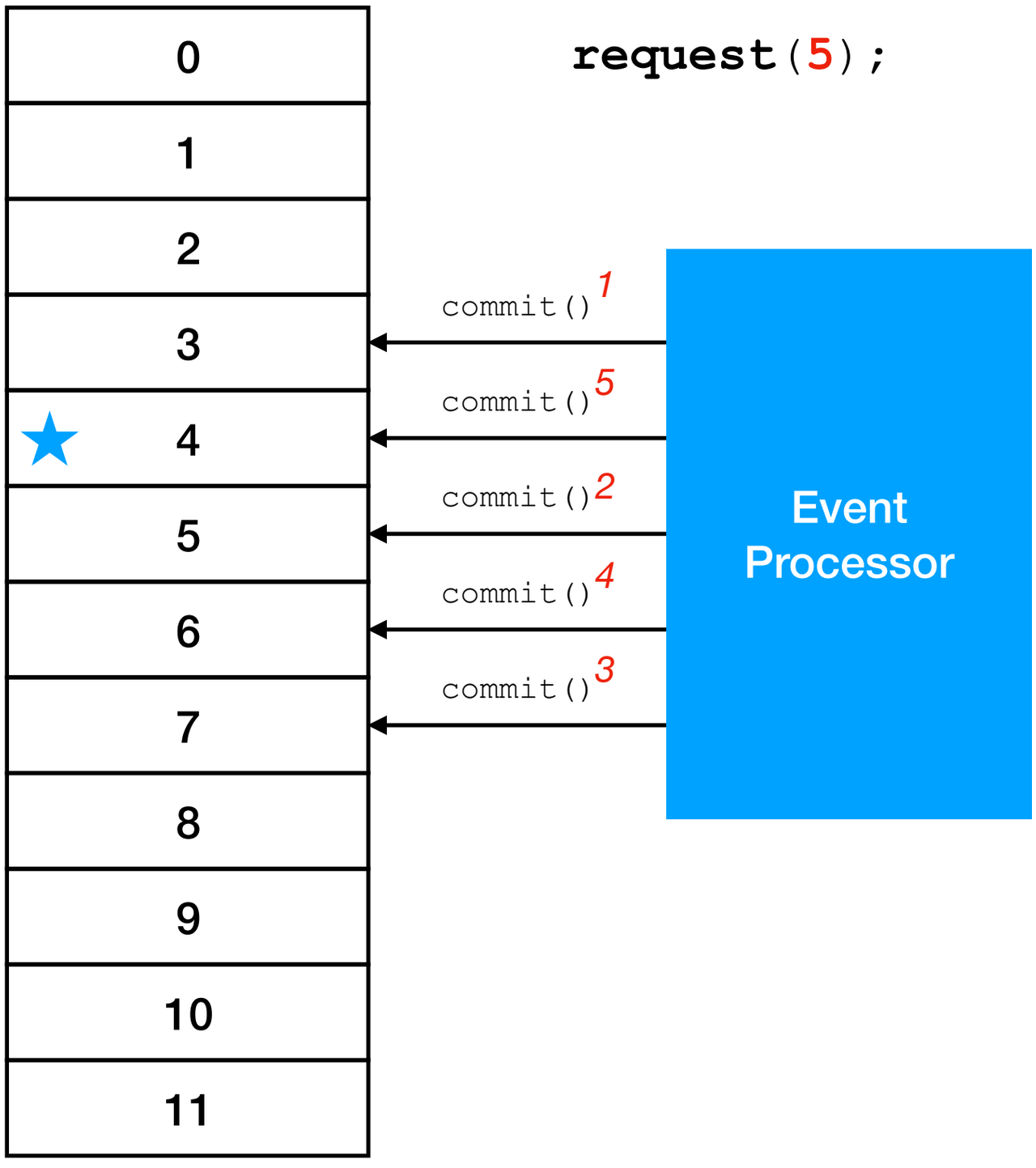

request(5);
0
1
2
3
4
5
6
7
8
9
10
11
commit() 1
commit() 2
commit() 3
commit() 4
commit() 5
Event Processor

시간을 달리는 메시지



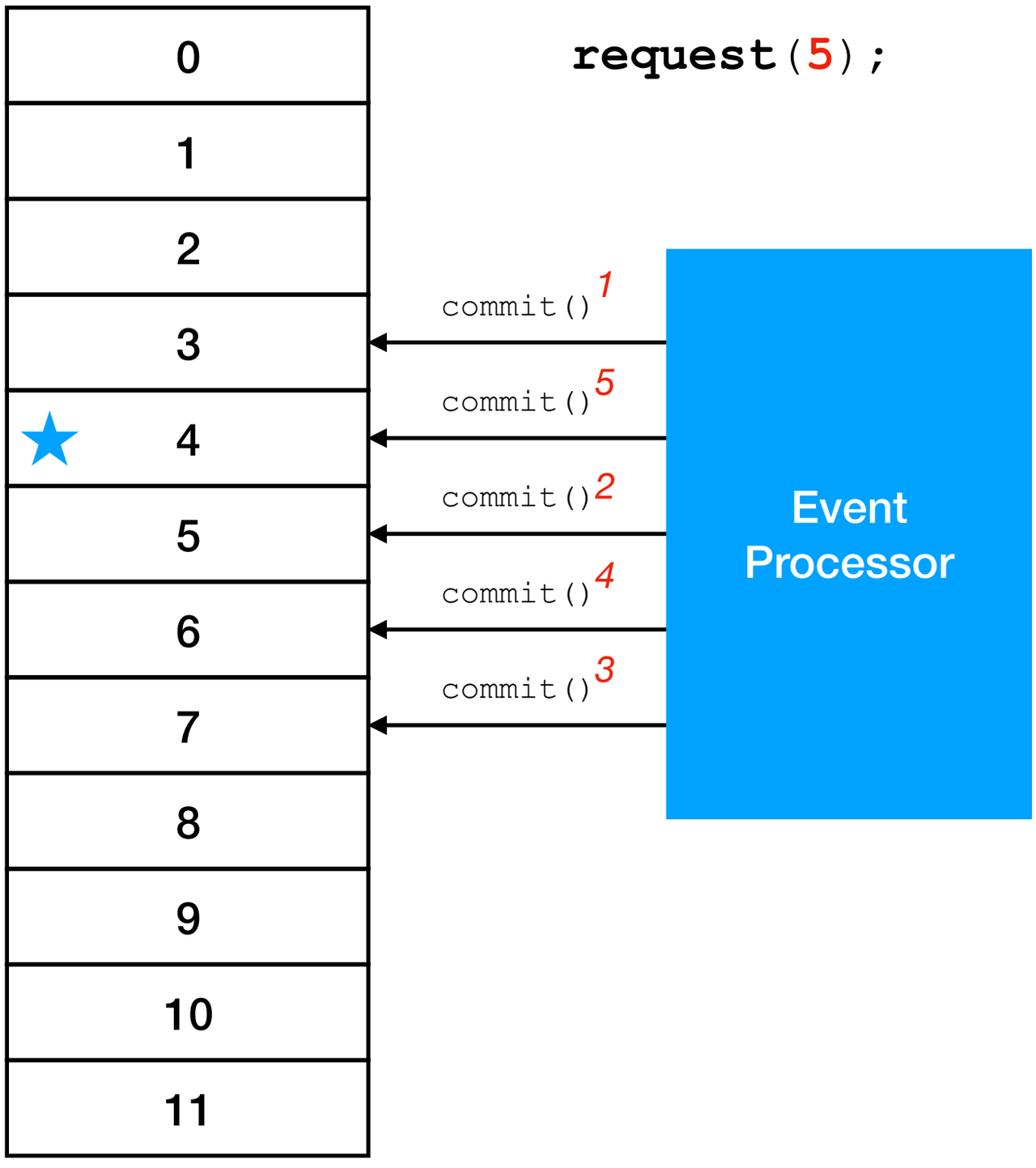
시작 할 때는
차례대로지만,

시간을 달리는 메시지



끝날 때는
아니란다!

시간을 달리는 메시지



이 시점에서
프로세스를
재시작 한다면?

시간을 달리는 메시지

Offset이 증가하는 경우에만 commit을 하자.

```
public class CustomSubscriber extends BaseSubscriber<~> {

    private UnicastProcessor<~> offsetProcessor =
                                        UnicastProcessor.create();

    private FluxSink<~> offsetSink = offsetProcessor.sink();

}
```

시간을 달리는 메시지

Offset이 증가하는 경우에만 commit을 하자.

```
public class CustomSubscriber extends BaseSubscriber<~> {

    @Override
    protected void hookOnNext(ReceiverRecord<String, String> record) {
        Mono.just(record)
            .flatMap(/* Do Something */)
            .subscribe(r -> {
                offsetSink.next(record);
                request(1);
            });
    }
}
```

시간을 달리는 메시지

Offset이 증가하는 경우에만 commit을 하자.

```
public class CustomSubscriber extends BaseSubscriber<~> {

    public CustomSubscriber(Function<~> func) {
        this.offsetProcessor.publish()
            .autoConnect()
            .reduce(-1L, (last, r) -> last < r.offset()
                                    ? commit(r)
                                    : last
                    )
            .subscribe();
    }
}
```

DEMO

https://github.com/EleganceLESS/spring-camp-2019

http://t0a.st/lXHi

몇 가지 결론

구슬이 서 말이라도
꿰어야 보배라

More Abstraction, Less Code!

The NO Free Lunch, NO Silver Bullet.

Bye-bye, Spring Kafka?

Spring Kafka 2.3.0(M1)

```
public class ReactiveKafkaProducerTemplate<K, V>

public class ReactiveKafkaConsumerTemplate<K, V>
```

https://github.com/spring-projects/spring-kafka/issues/43

Based on Reactor Kafka!

Q -> Flux<A>

https://github.com/EleganceLESS/spring-camp-2019

http://t0a.st/lXHi

We are Hiring!!

https://recruit.nhn.com/

References & Materials

- ReactiveX
  http://reactivex.io/

- Reactor 3 Reference Guide
  https://projectreactor.io/docs/core/release/reference/

- Reactor Kafka Reference Guide
  https://projectreactor.io/docs/kafka/release/reference/

- Web on Reactive Stack
  https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html

- Your mouse is a database
  http://queue.acm.org/detail.cfm?id=2169076

- Python Asynchronous Programming with RxPY
  https://github.com/kstreee/kstreee.github.io/blob/master/talk/pycon2017-skim.pdf

Thank you,
Happy Reactive Coding!