MVC ~~vs~~ WebFlux

웹플럭스?

WebFlux = Spring Reactor

Reactor는 RxJava 2와 함께 Reactive Stream의 구현체

*** 쉽게 이해하시려면 java stream**
(netty base)

생명주기 = 조립 + 구독 + 런타임

마이크로서비스 기반 시스템에 적합

모바일 환경에 적합

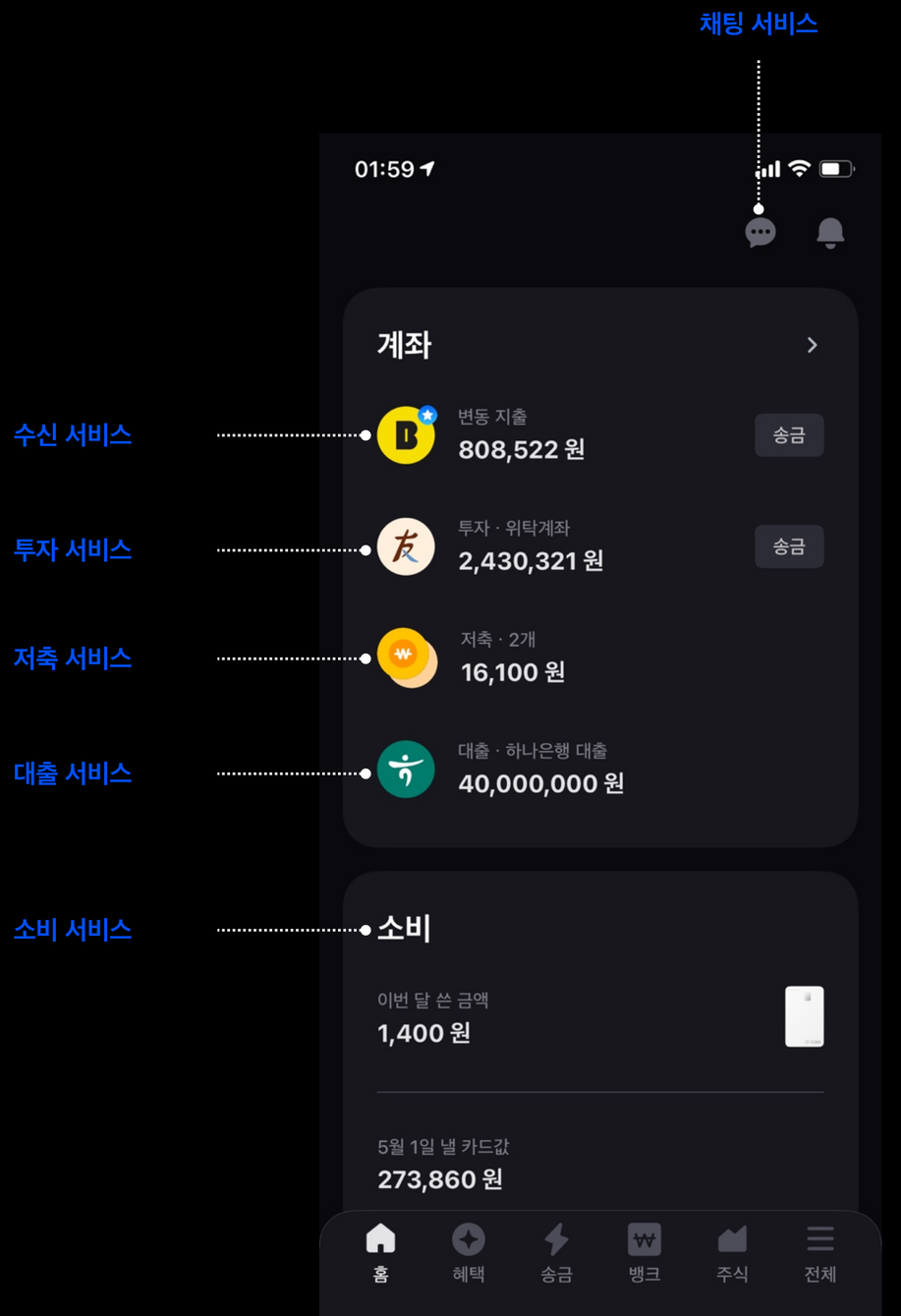
낮은 대기시간과 높은 처리량

확장성 및 고효율성

업/ 다운 스트리밍 및 백프레셔

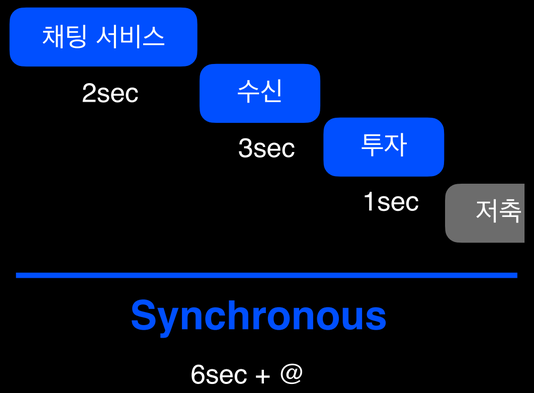
모노리스 방식과 비교해 마이크로 시스템이 가지는

핵심적인 특징은 많은 수의 I/O 통신

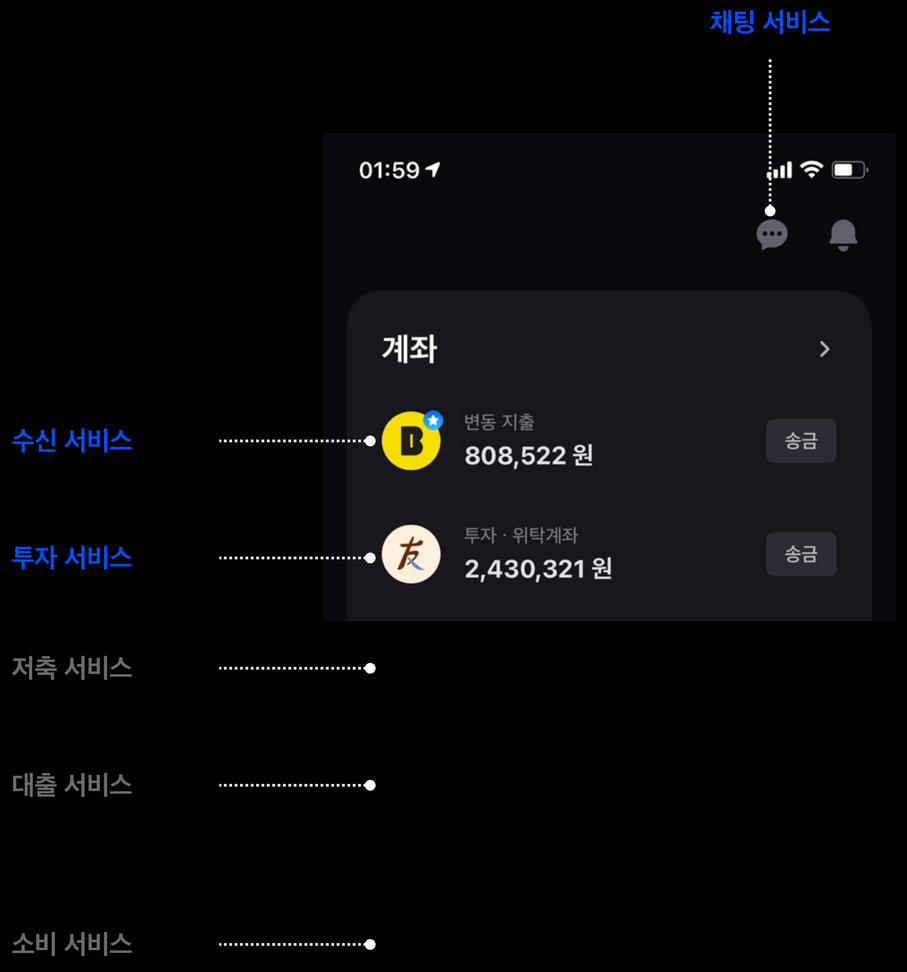


모노리스 방식과 비교해 마이크로 시스템이 가지는

핵심적인 특징은 많은 수의 I/O 통신



모든 요청에 걸리는 시간의 합



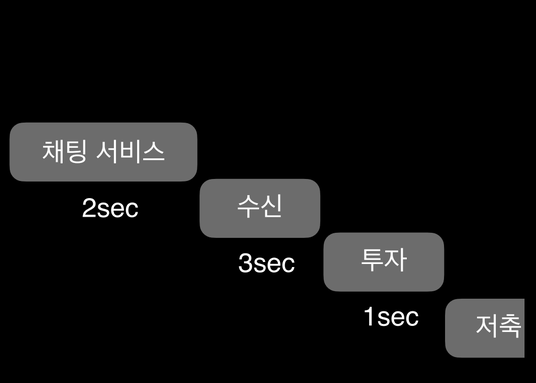
ex/

```
ex)
val 채팅_컨텐츠 = 채팅테이블.getContents()
val 수신_컨텐츠 = 수신테이블.getContents()
val 투자_컨텐츠 = 투자테이블.getContents()
val 저축_컨텐츠 = 저축테이블.getContents()
                ...
```

```
ex)
val 채팅_컨텐츠 = 채팅서버.getContents()
val 수신_컨텐츠 = 수신서버.getContents()
val 투자_컨텐츠 = 투자서버.getContents()
val 저축_컨텐츠 = 저축서버.getContents()
                ...
```

모노리스 방식과 비교해 마이크로 시스템이 가지는

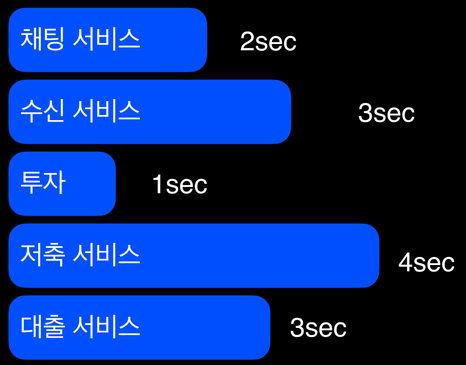
핵심적인 특징은 많은 수의 I/O 통신



Synchronous

6sec + @

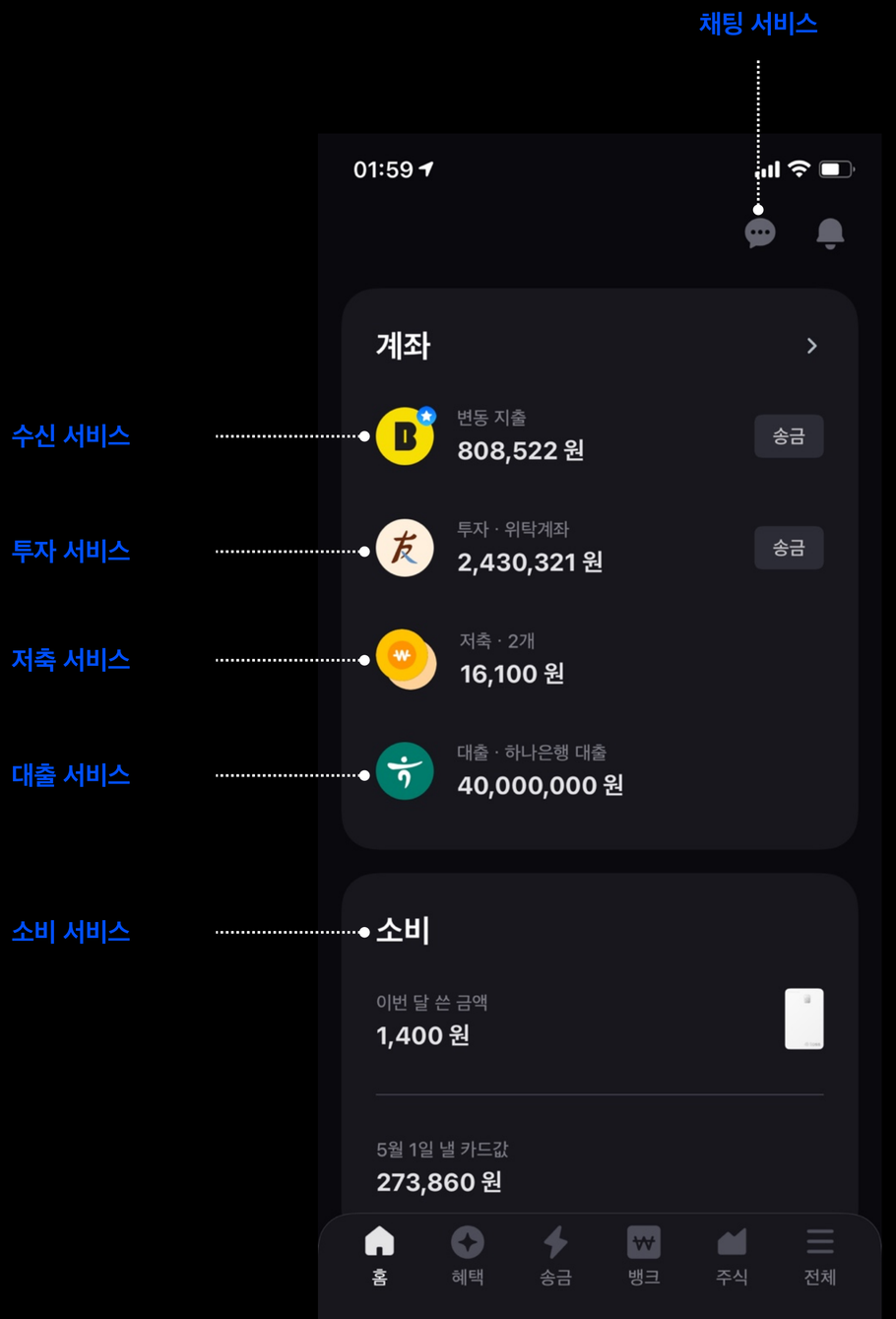
모든 요청에 걸리는 시간의 합



Asynchronous

4sec

요청에 걸리는 시간 중 최대 값

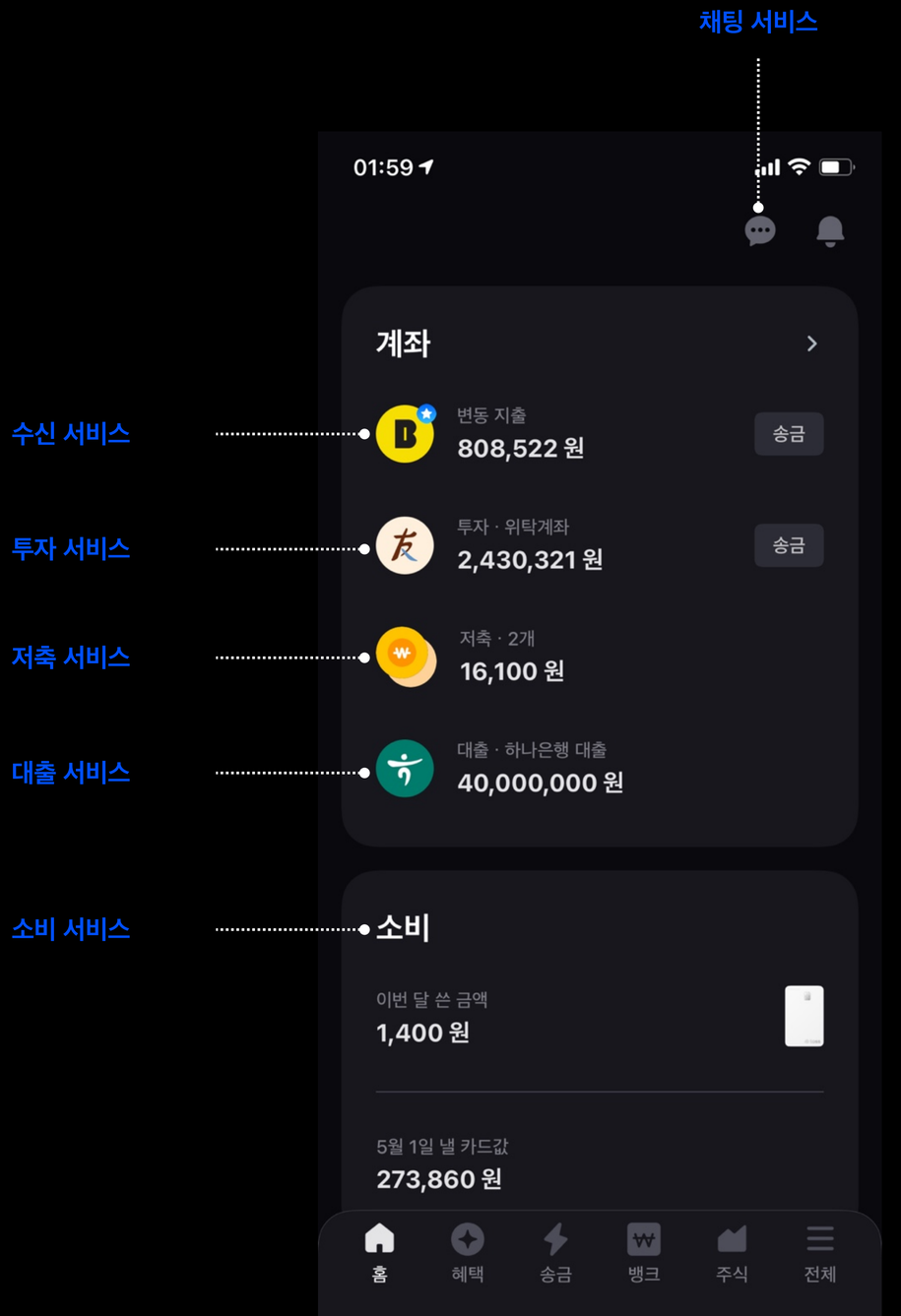


모노리스 방식과 비교해 마이크로 시스템이 가지는

핵심적인 특징은 많은 수의 I/O 통신

즉, 서비스 간 호출이 많은 시스템의 경우

웹플럭스는 가장 효율적인 솔루션 중 하나



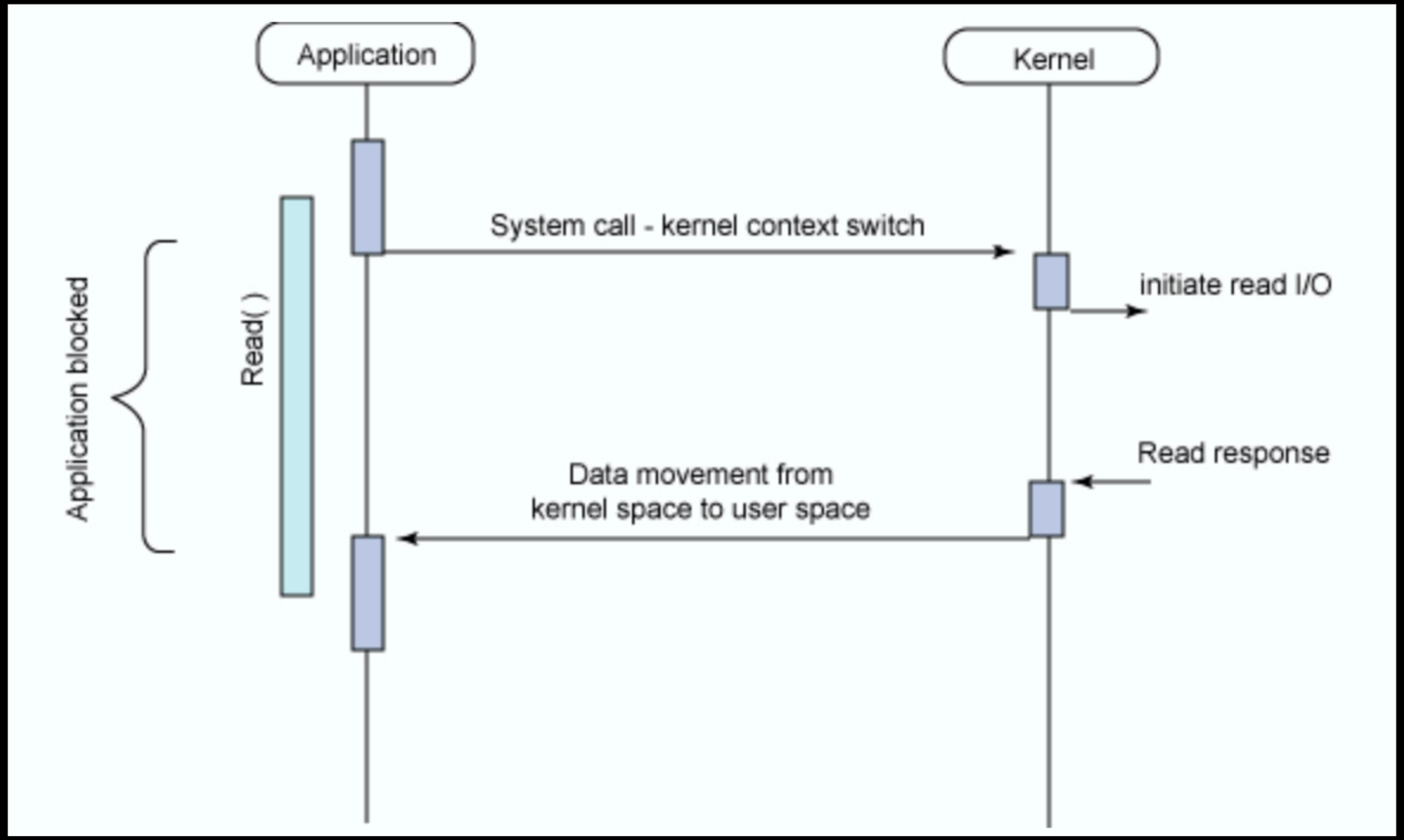
왜 이런 차이를 보이는걸까?

Synchronous Blocking I/O

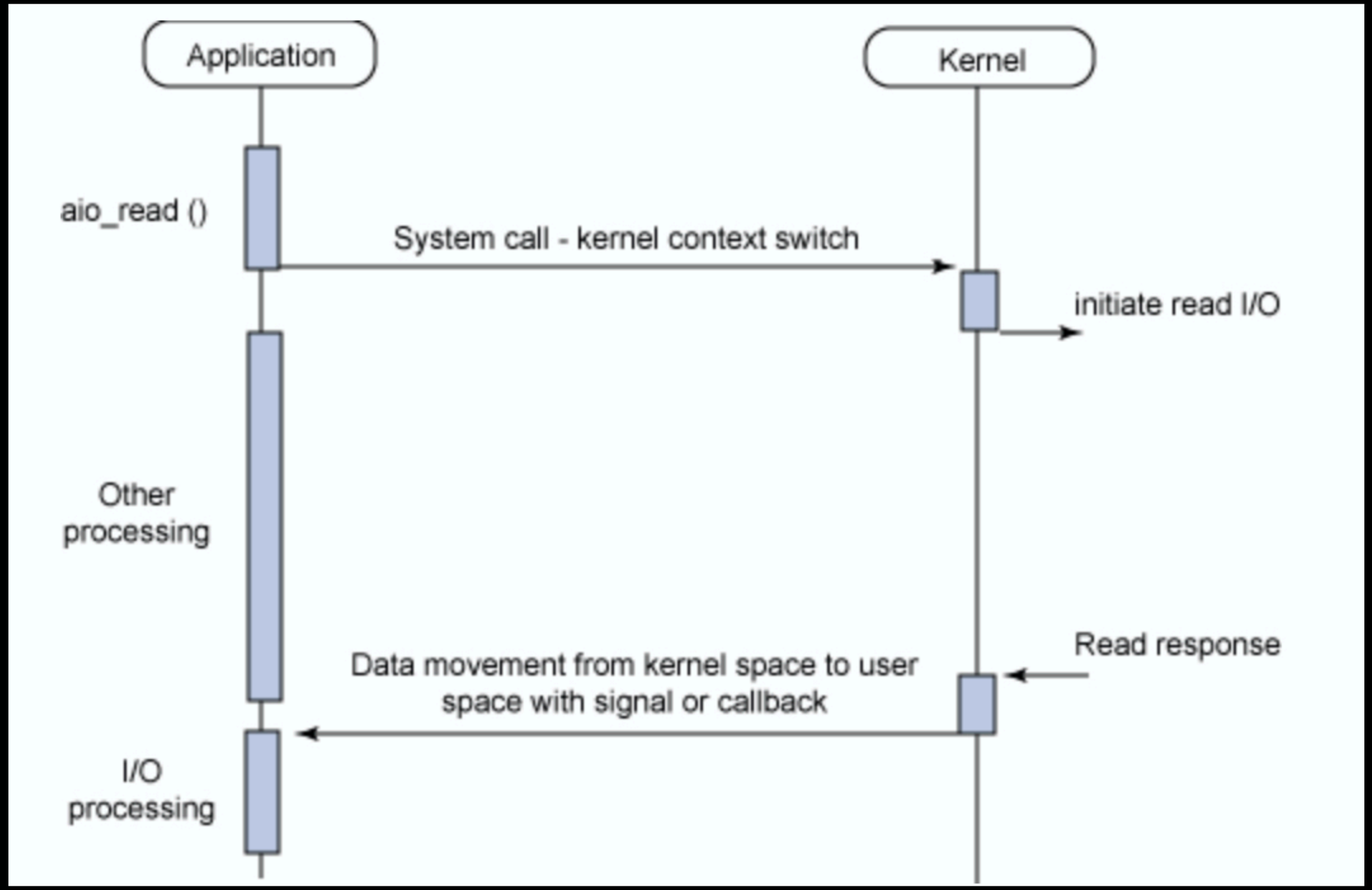
vs

Asynchronous Non-Blocking I/O (AIO)

Synchronous Blocking I/O



Asynchronous Non-Blocking I/O (AIO)



thread per request

savings 👨🏻‍✈️ /credentials → thread 192

savings 👨🏻‍🍳 /accounts → thread 16

user 👩🏿 /user →

loans 👨🏻‍🎨 /applications →

....

thread 44, thread 79

default thread = 200

thread per request

savings 👨🏻‍✈️ /credentials → thread 192

savings 👨🏻‍🍳 /accounts → thread 16

user 👩🏿 /user → thread 44

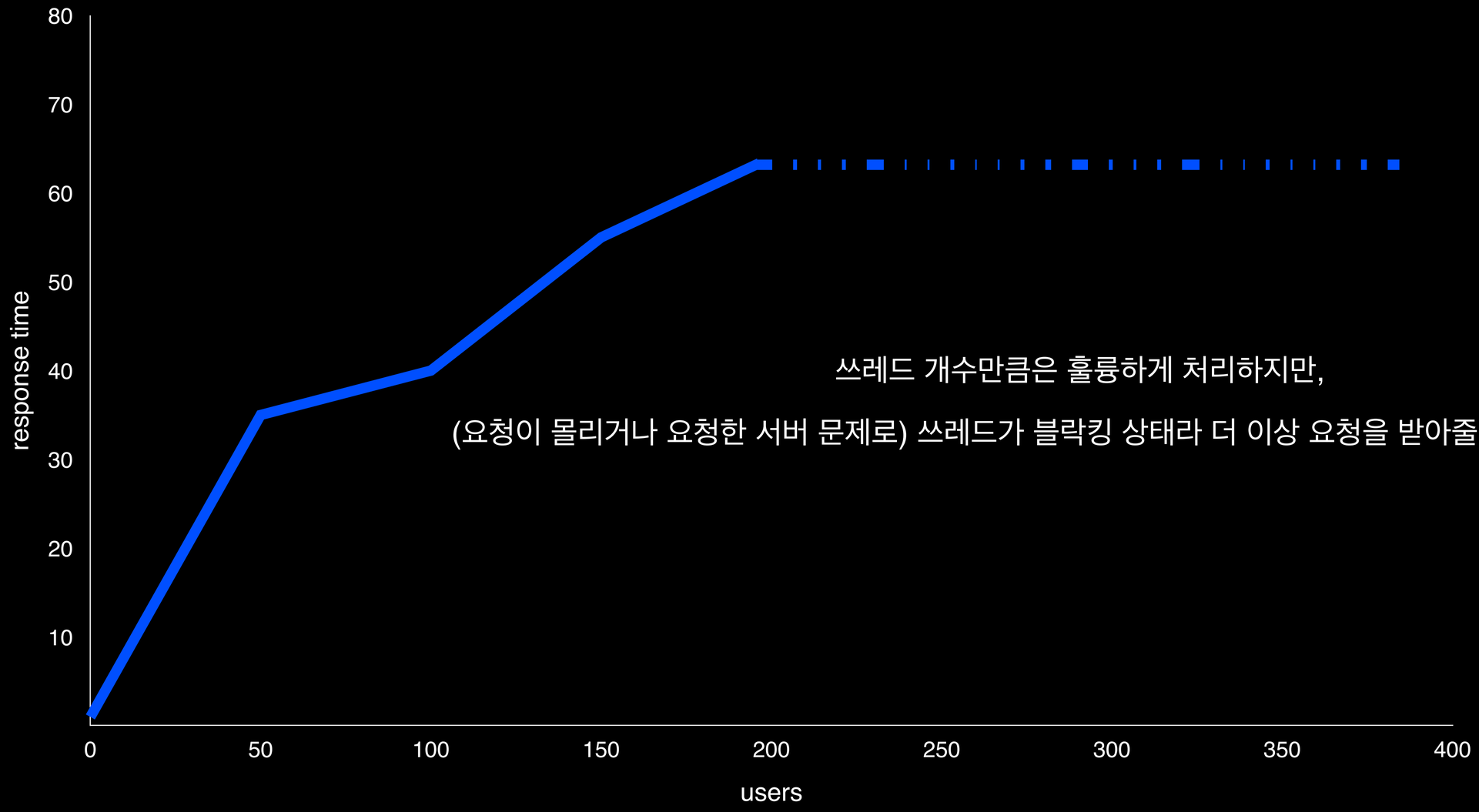
loans 👨🏻‍🎨 /applications → thread 79

....

no more thread!

default thread = 200

Synchronous Blocking I/O



쓰레드 개수만큼은 훌륭하게 처리하지만,

(요청이 몰리거나 요청한 서버 문제로) 쓰레드가 블락킹 상태라 더 이상 요청을 받아줄 수 없는 상태라면?

event loop

default thread = core * 2

savings 👨🏻‍✈️ /credentials

savings 👨🏻‍🍳 /accounts

user 👩🏿‍💼 /user

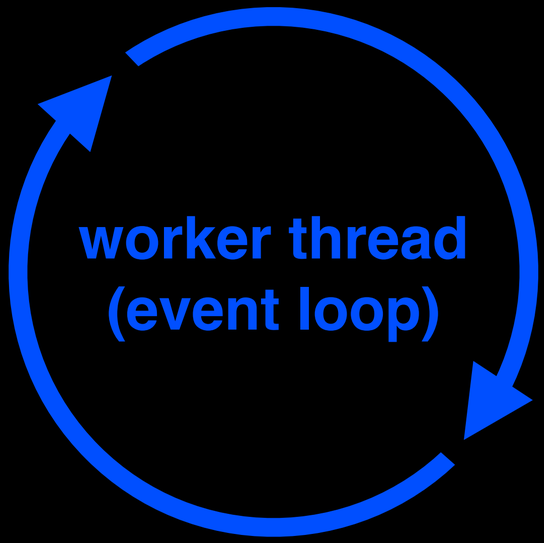
loans 👨🏽‍🎨 /applications

bank-fds 👩🏻‍⚕️ /verify

card 👷🏻‍♂️ /payment

document 👲🏻 /download

....



request
(register callback event)

callback

OPERATION

👨🏻‍🍳 event

👩🏿‍💼 event

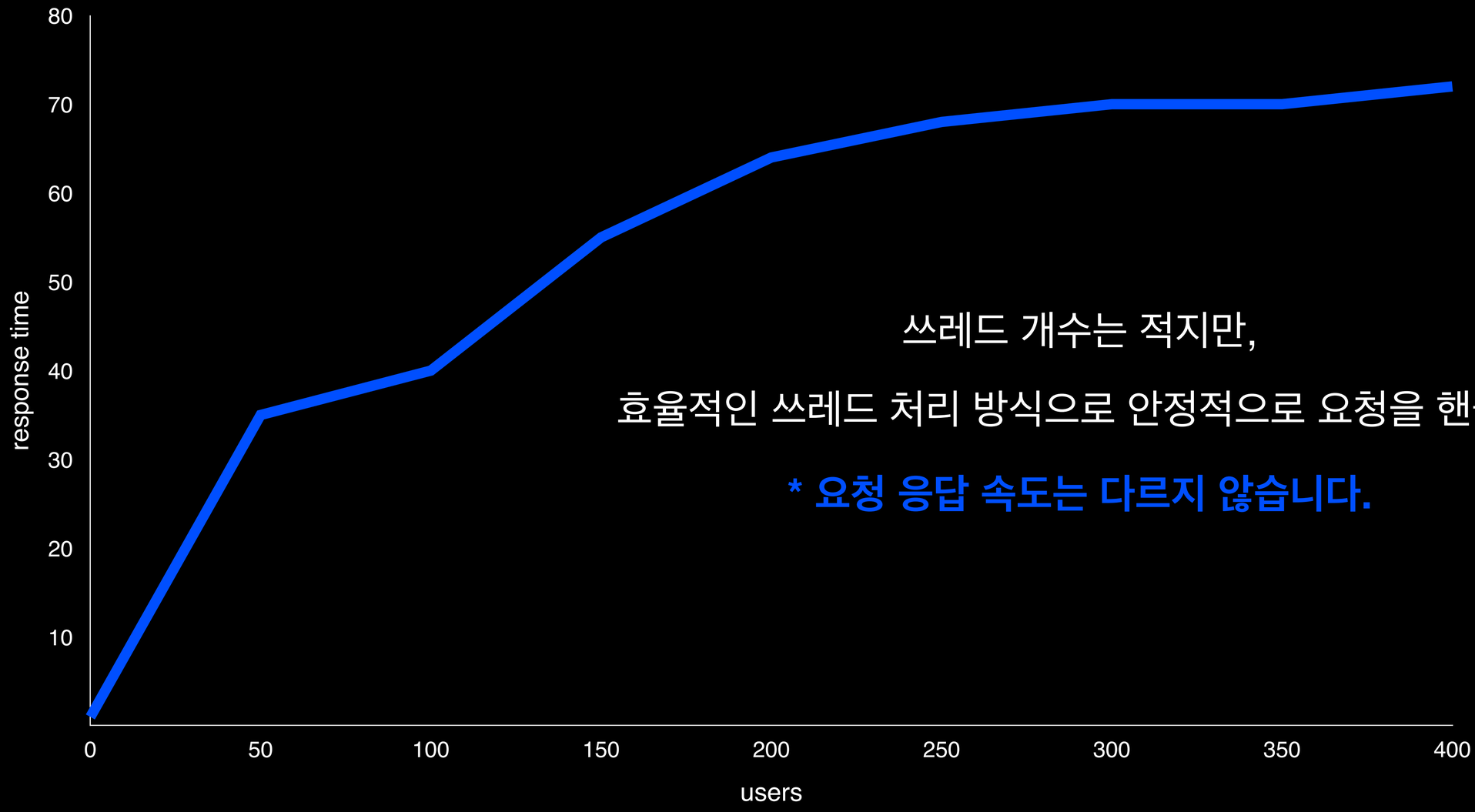
👨🏻‍✈️ event

👨🏽‍🎨 event

👲🏻 event

....

Asynchronous Non-Blocking I/O (AIO)



쓰레드 개수는 적지만,

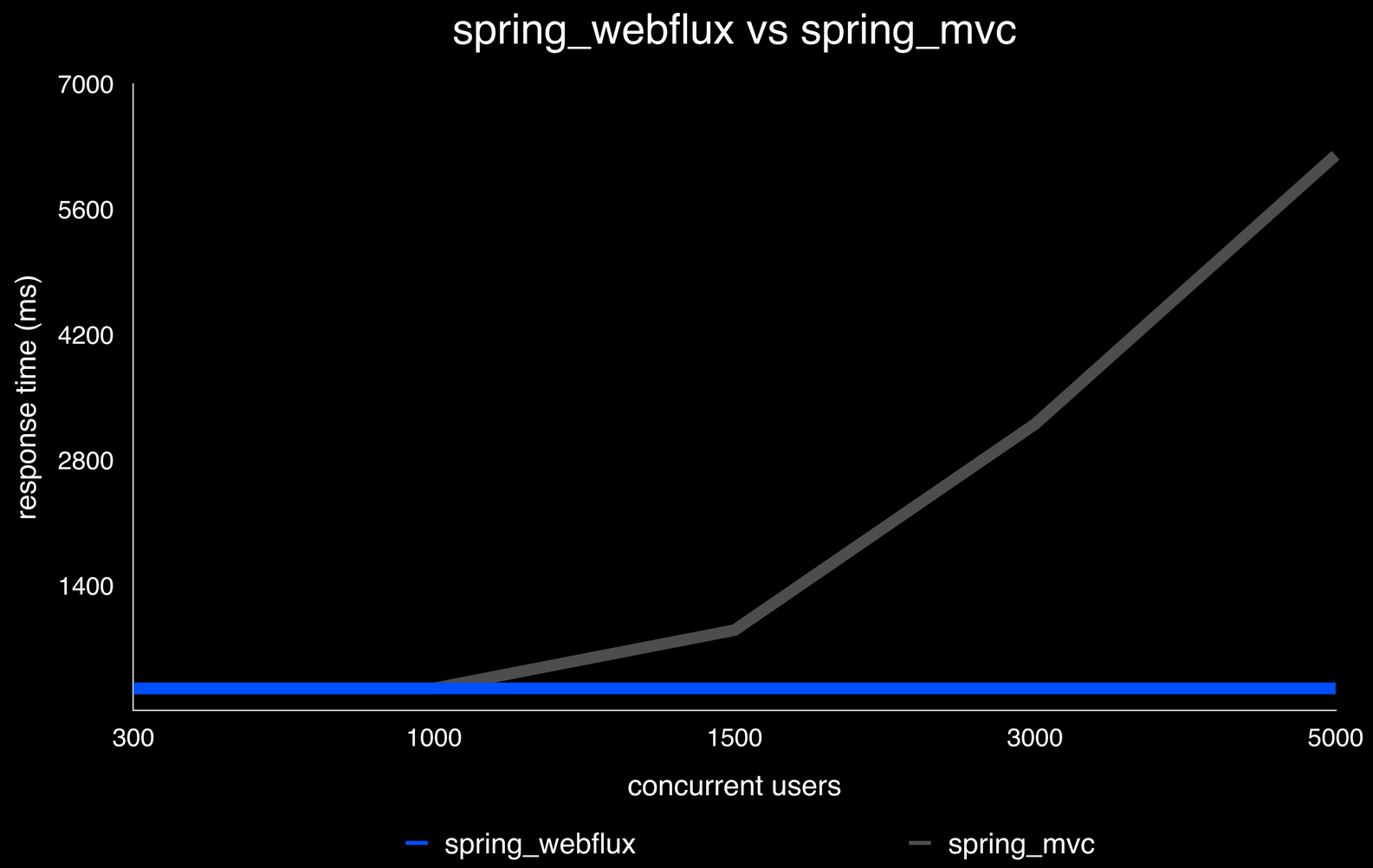
효율적인 쓰레드 처리 방식으로 안정적으로 요청을 핸들링

*** 요청 응답 속도는 다르지 않습니다.**

ex/ MVC = Scv, WebFlux = Probe

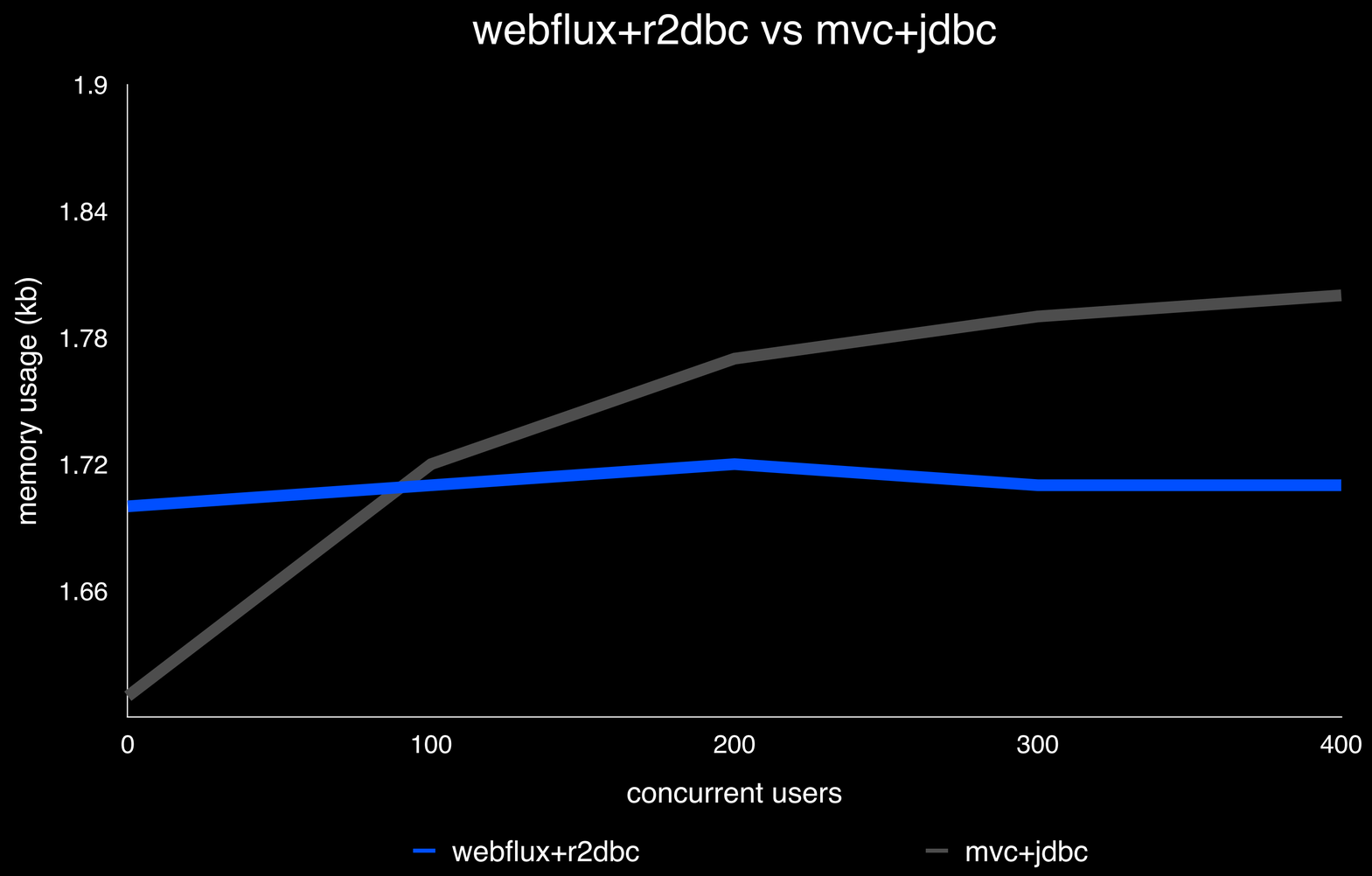
WebFlux Performance

WebFlux Performance : IO-heavy workloads



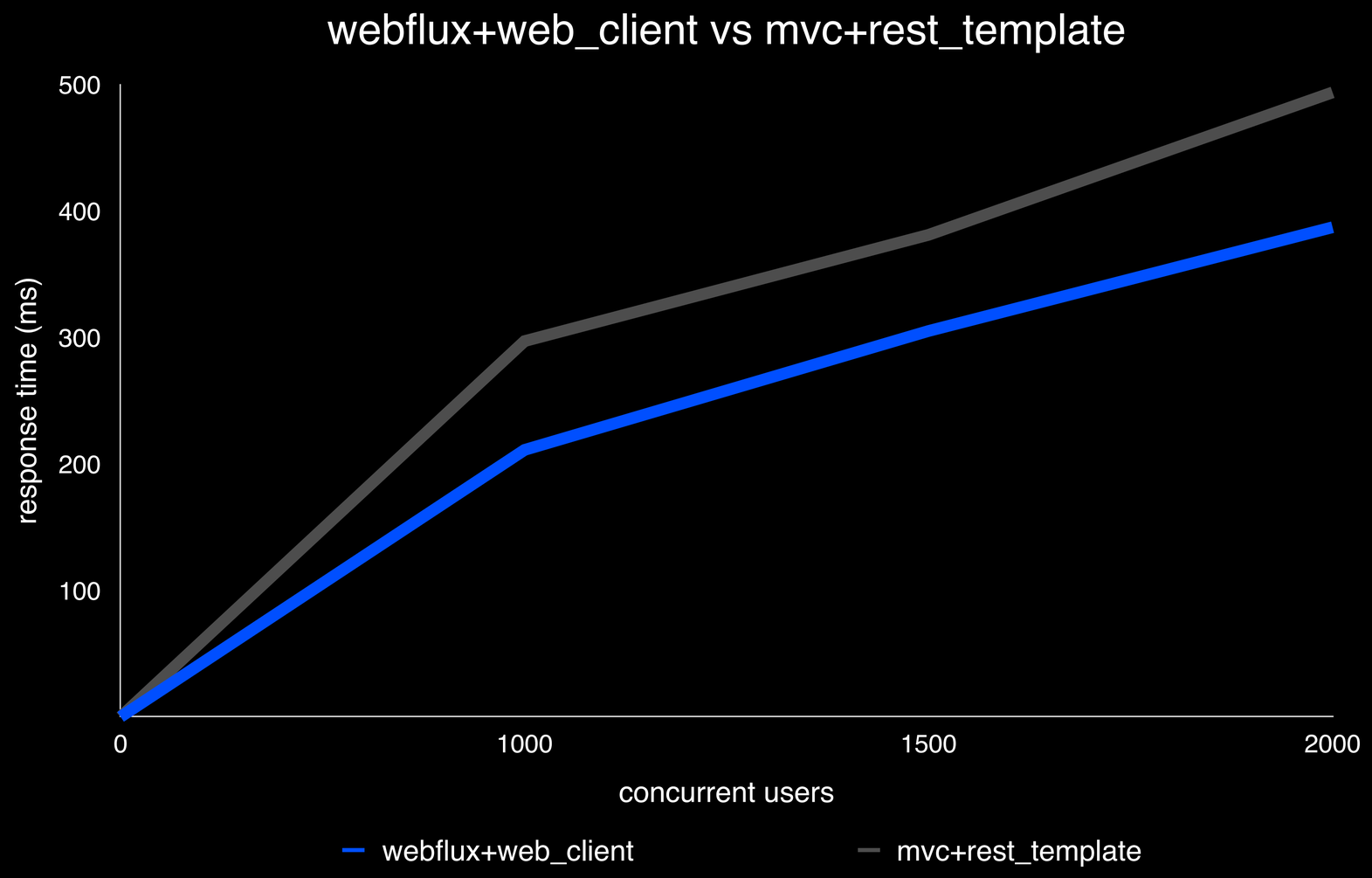
참고: https://dzone.com/articles/raw-performance-numbers-spring-boot-2-webflux-vs-s

WebFlux Performance : Memory Usage



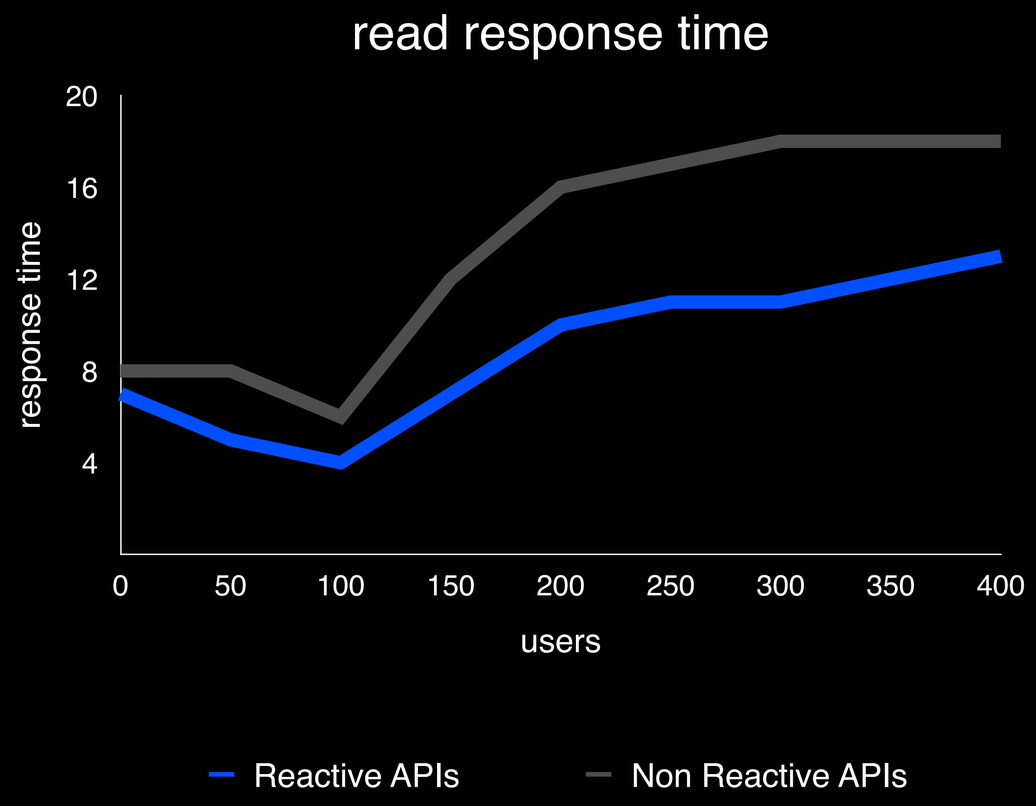
참고: https://technology.amis.nl/software-development/performance-and-tuning/spring-blocking-vs-non-blocking-r2dbc-vs-jdbc-and-webflux-vs-web-mvc/

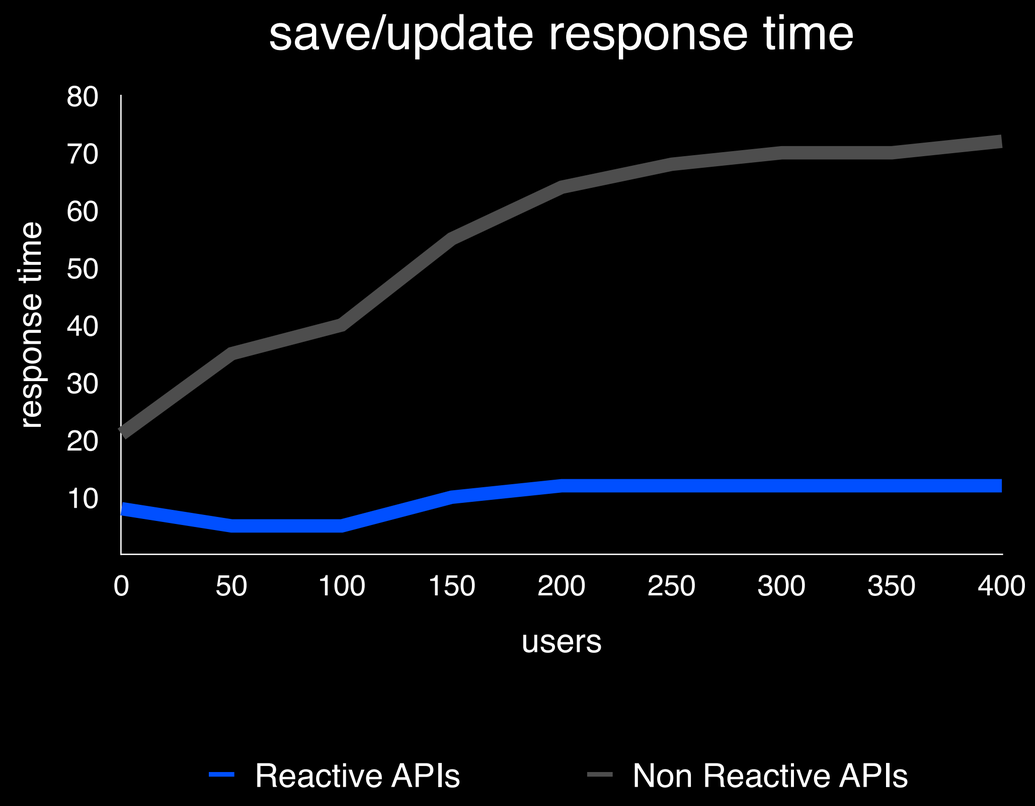
WebFlux Performance : Http Client



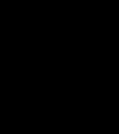
참고: http://gjchoi.github.io/etc/%EC%9B%B9%ED%94%8C%EB%9F%AD%EC%8A%A4_%EC%84%B1%EB%8A%A5%ED%85%8C%EC%8A%A4%ED%8A%B8/

WebFlux Performance : Redis

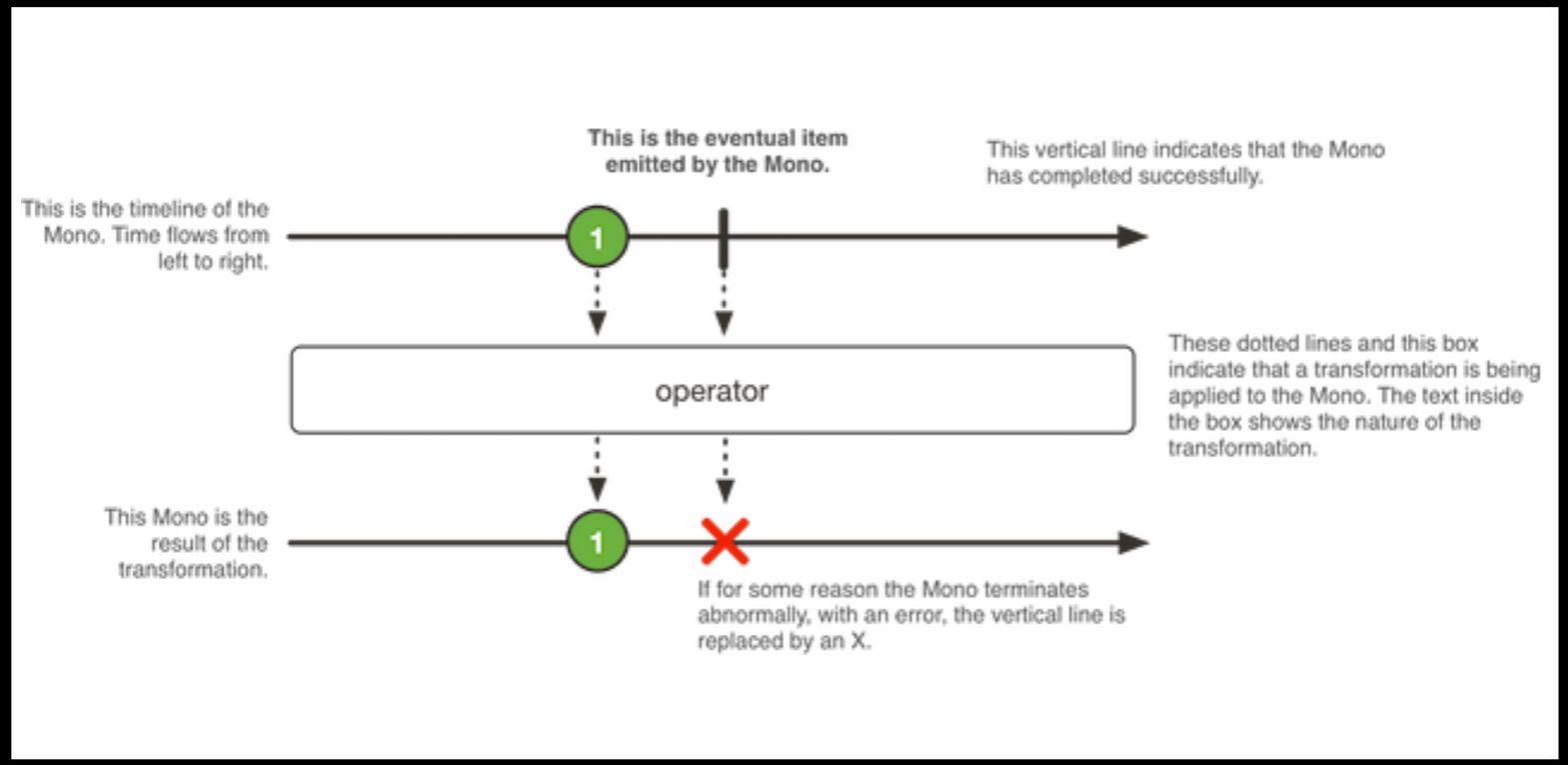




참고: https://dzone.com/articles/springboot-performance-war

WebFlux 

Mono an Asynchronous 0, 1 Result



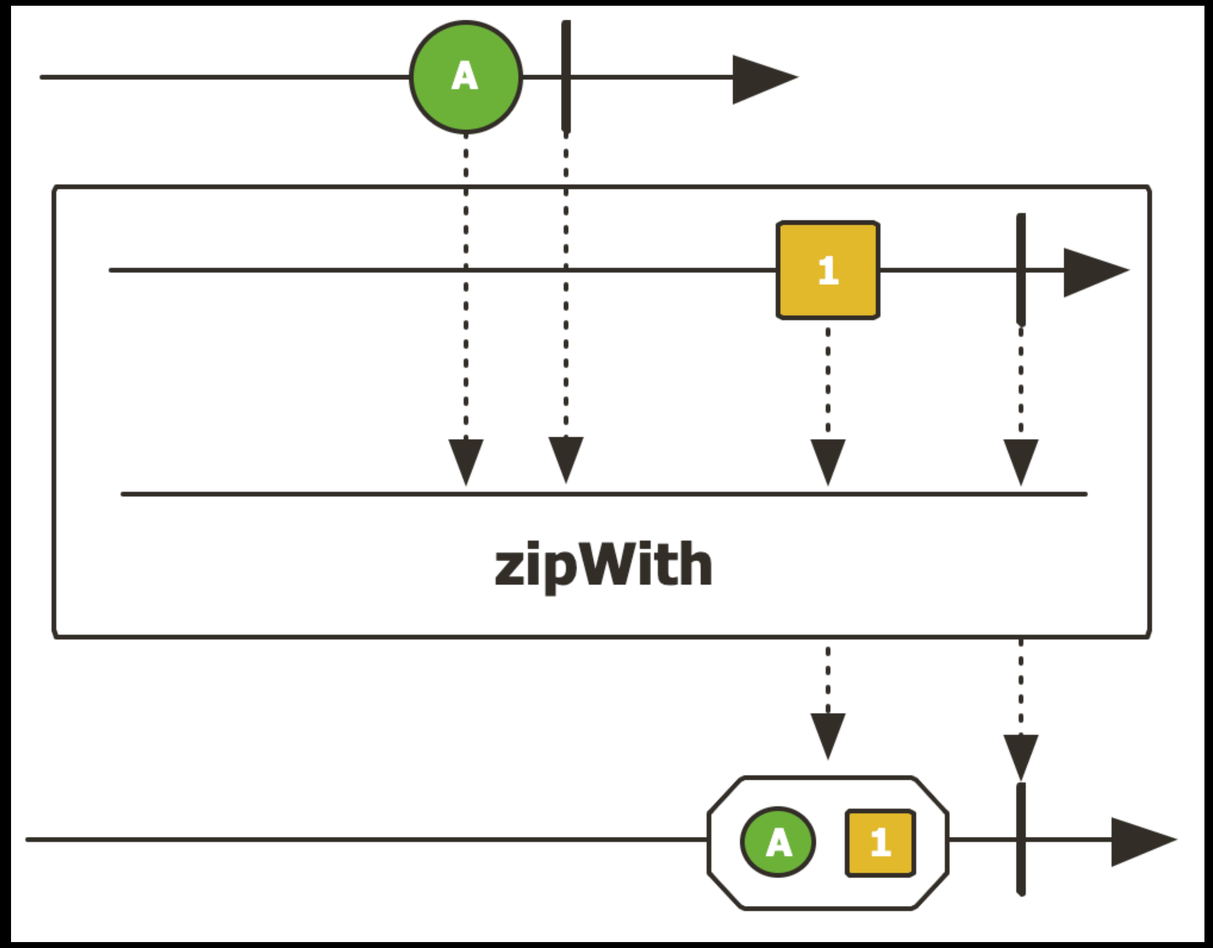
Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```

Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```

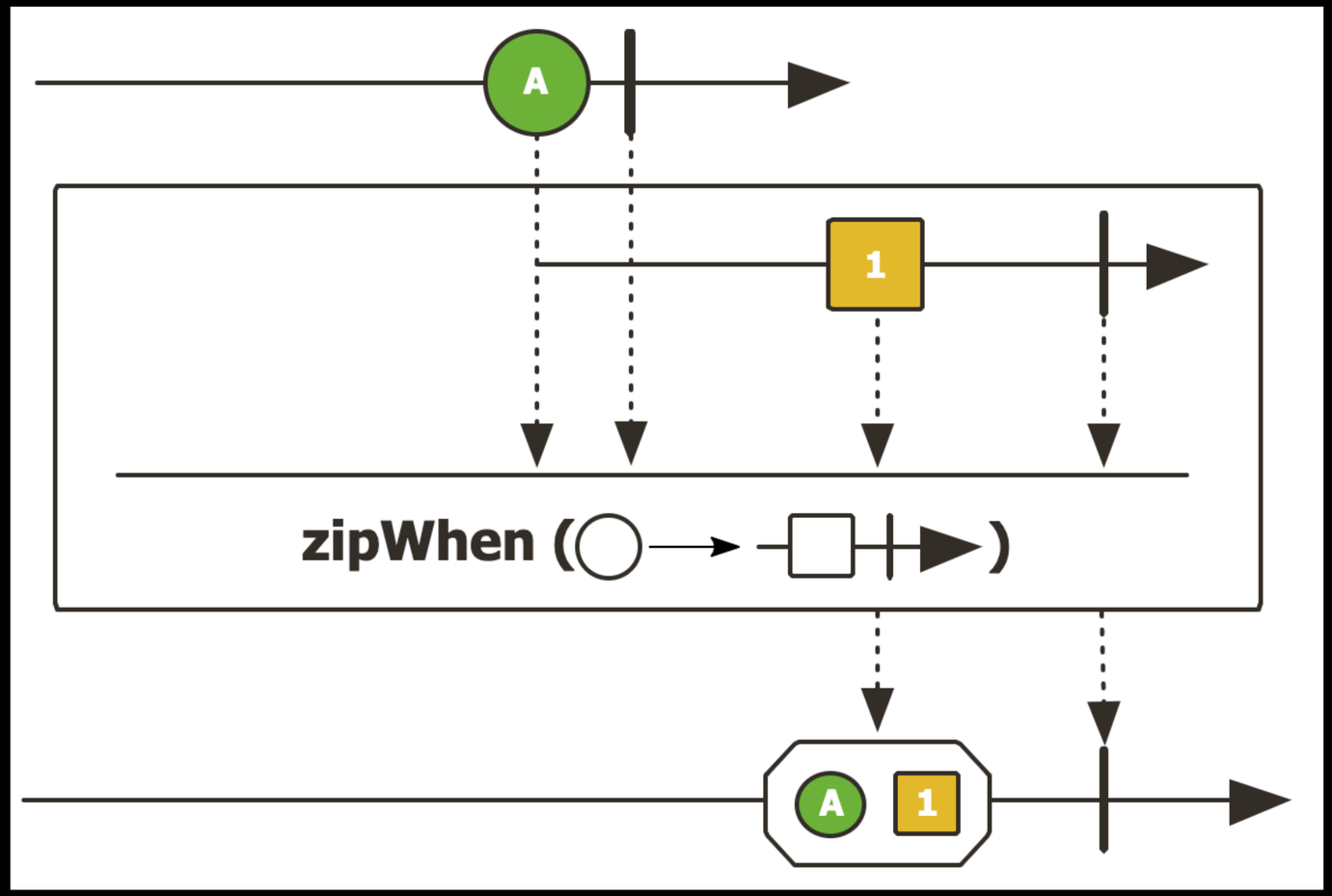
ex/ KCB + NICE + @ 데이터를 한번에 요청하고 받아서 저장 (async)



Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```

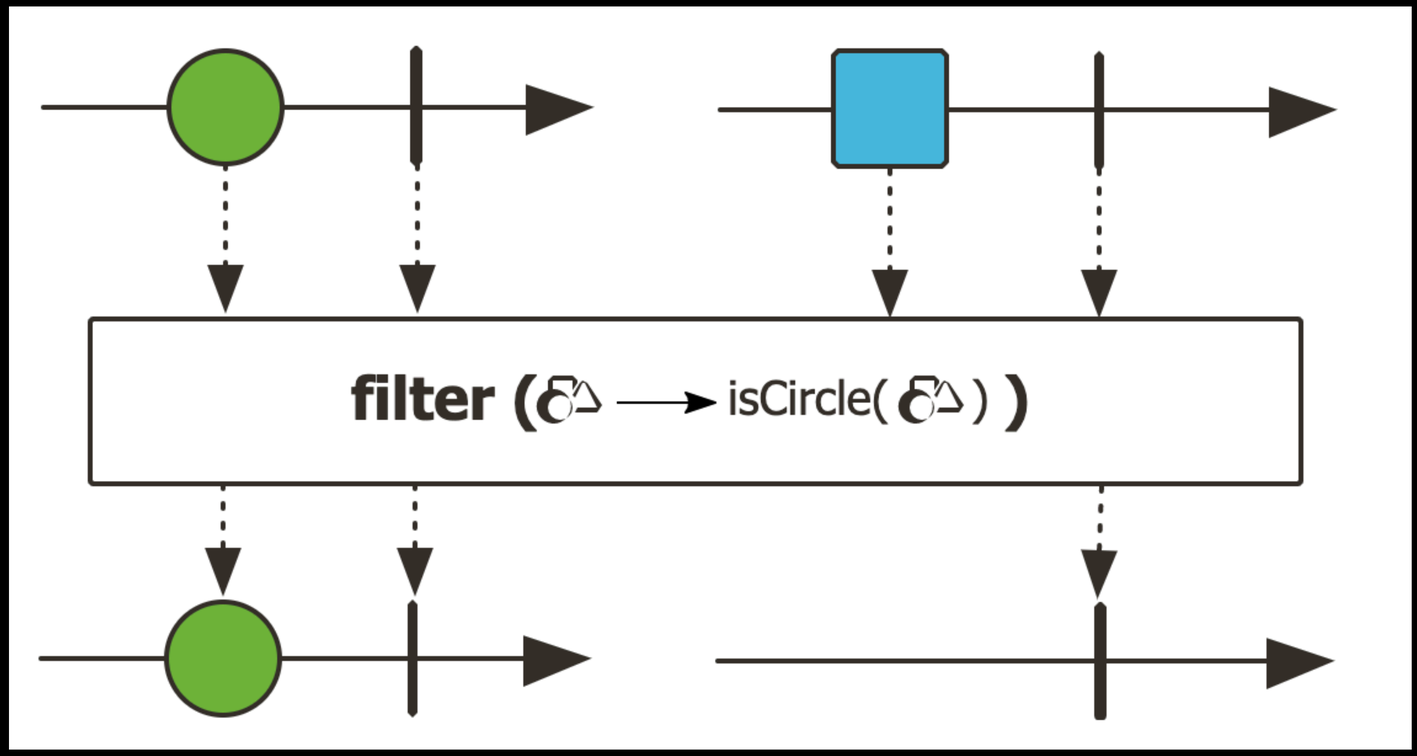
ex/ 사용자 대출 상태를 먼저 조회하고 대출 가능 여부에 따라 KCB + NICE + @ 데이터를 요청



Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```

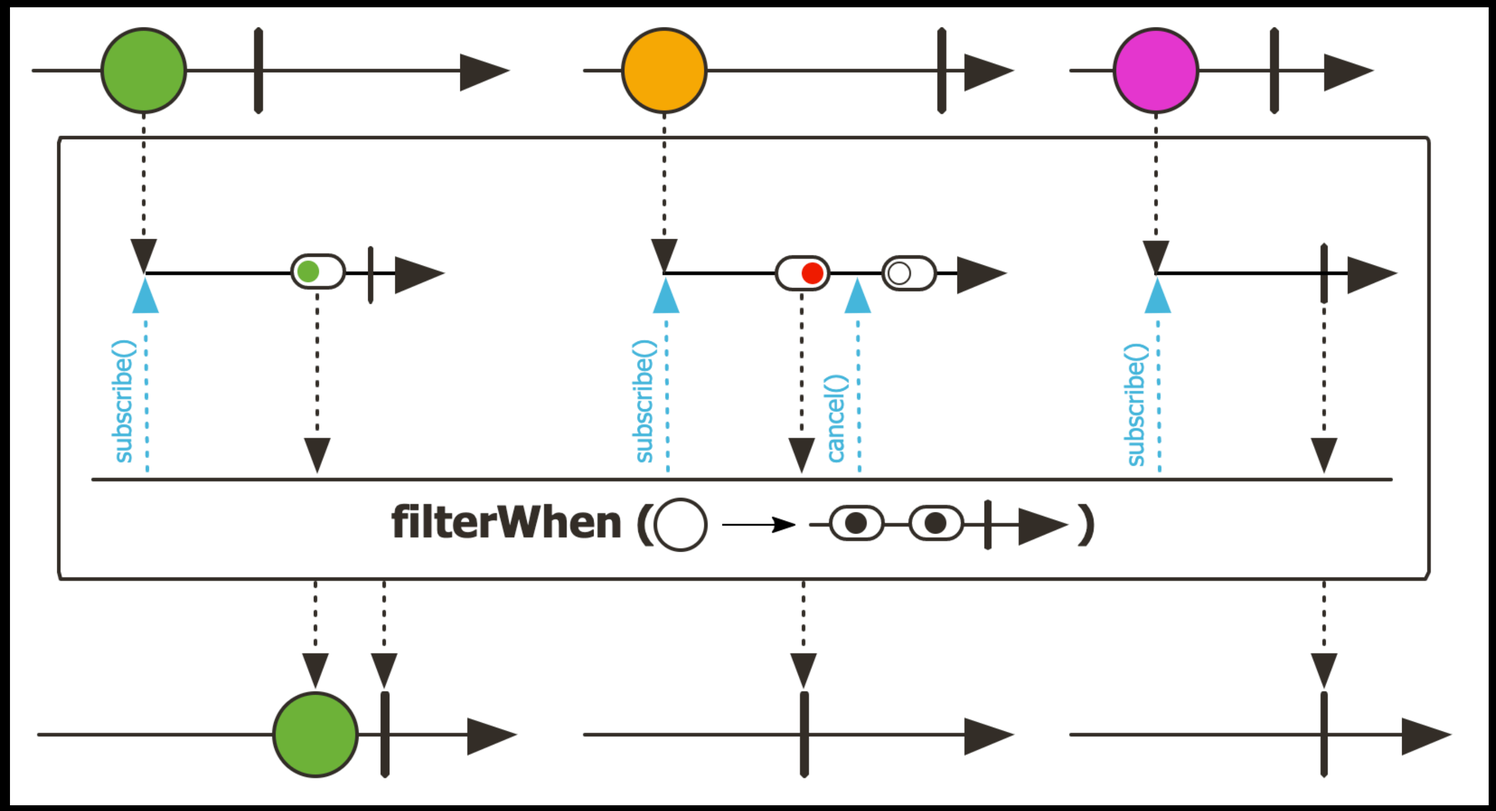
ex/ 조회한 회원의 데이터에서 총 대출금액이 2천만원 이하일 경우에만 진행가능 (sync)



Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```

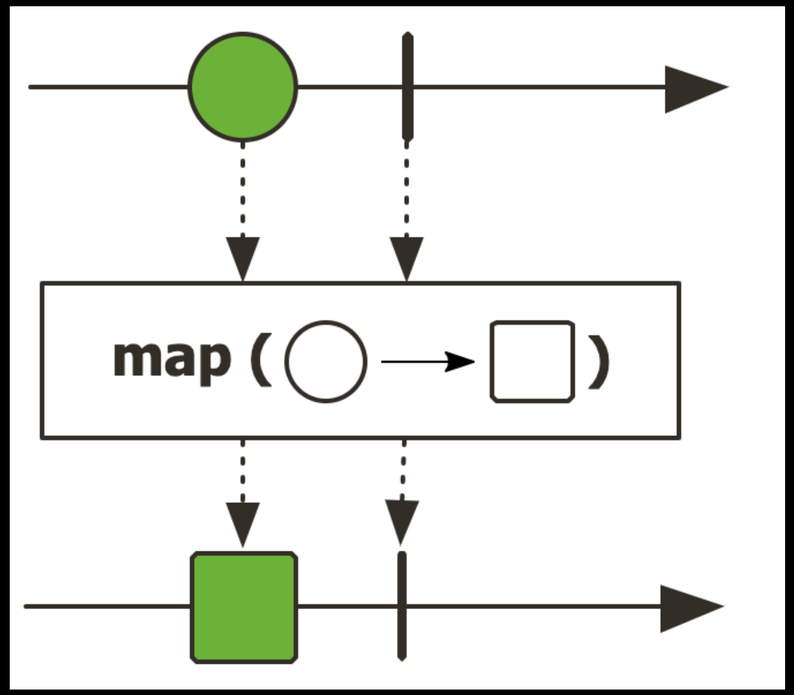
ex/ 회원이 타 기관에서 대출이 가능한지 외부 서버 API/전문 통신 조회 (async)



Mono an Asynchronous 0, 1 Result

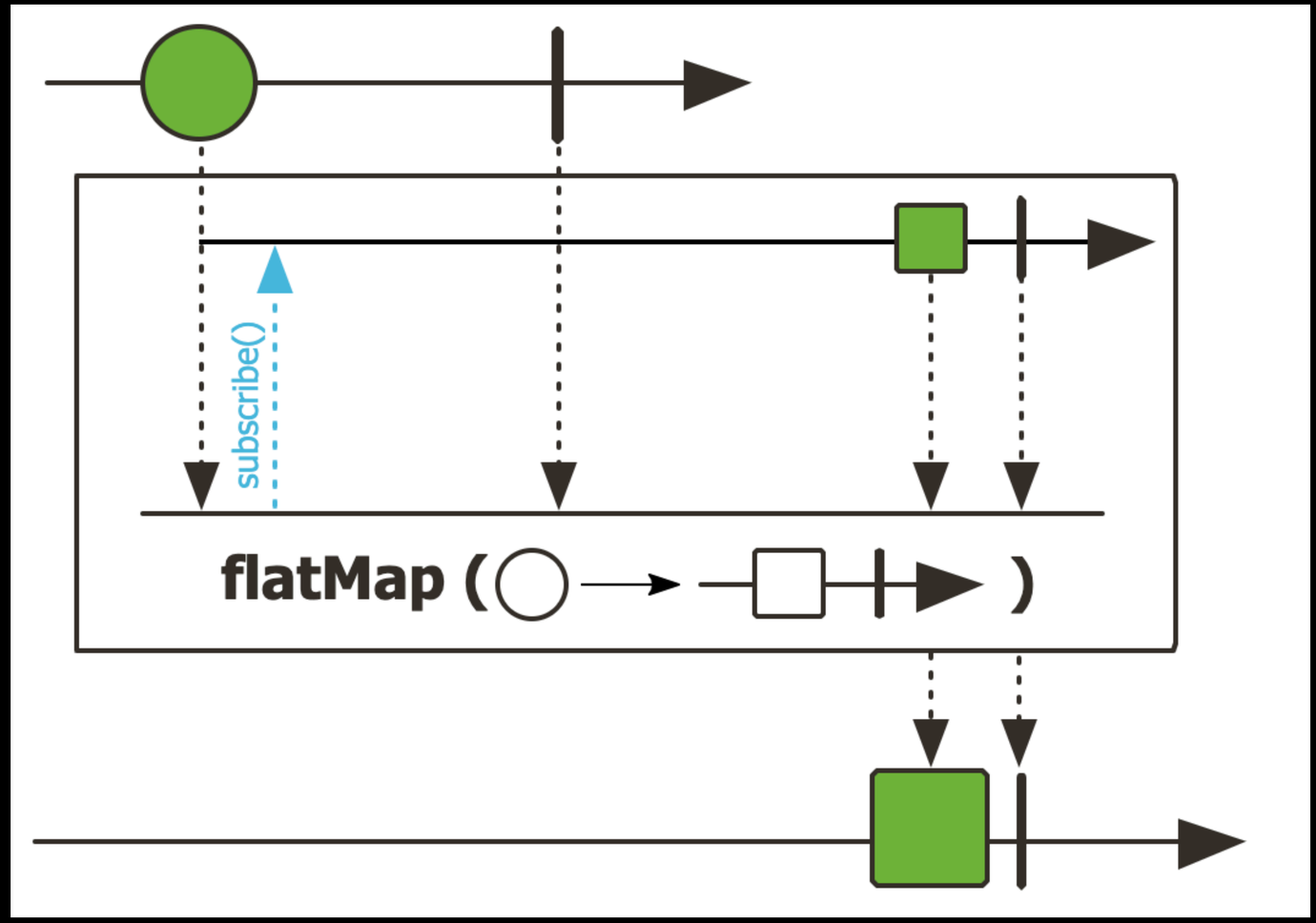
```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```

ex/ 스트림 시퀀스의 값을 변경하고 싶을 때 (sync)
회원의 전체 수신 데이터 중 자동이체 데이터만 빼서 다음 플로우를 진행시키고 싶을 때



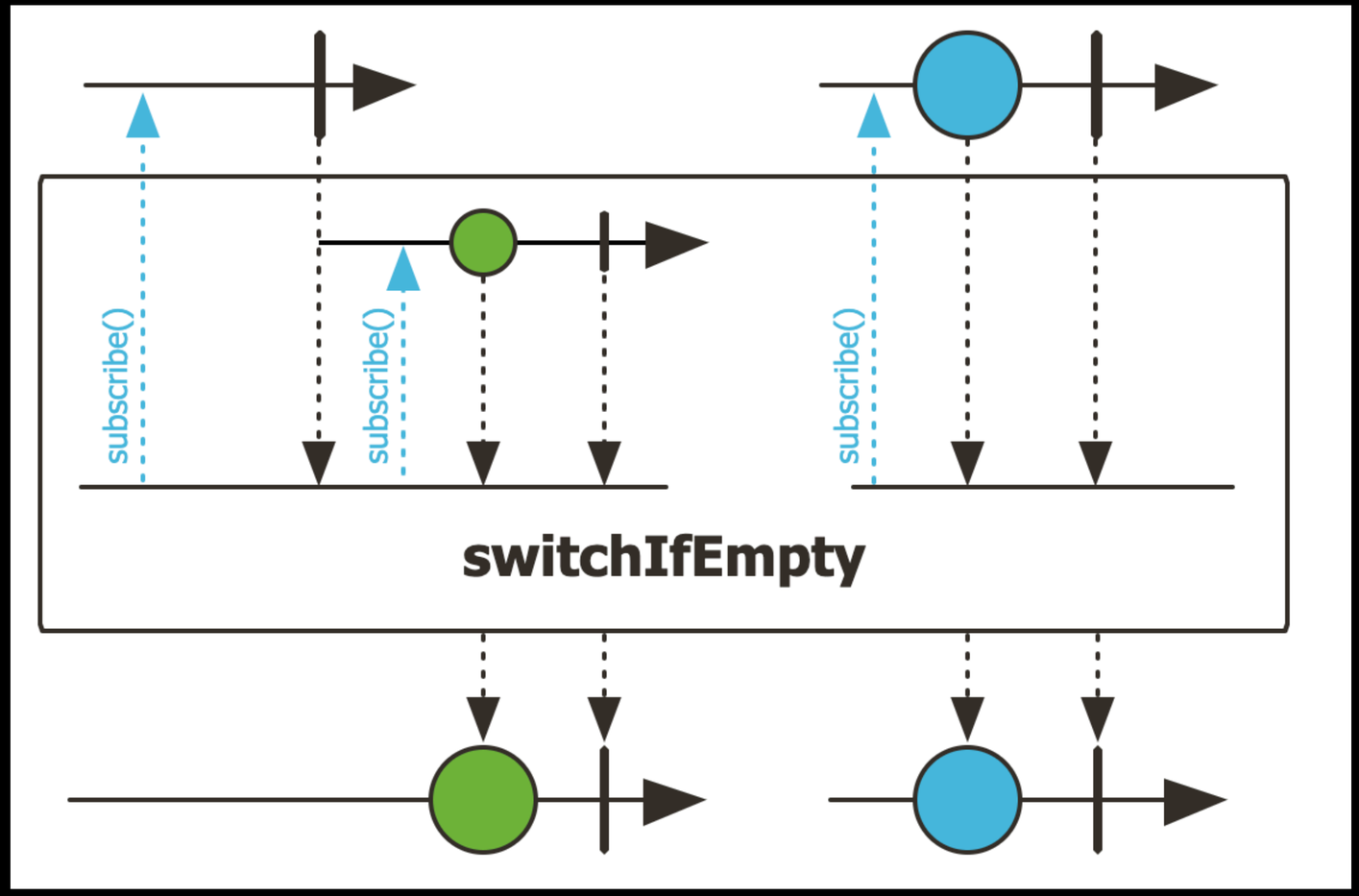
Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```



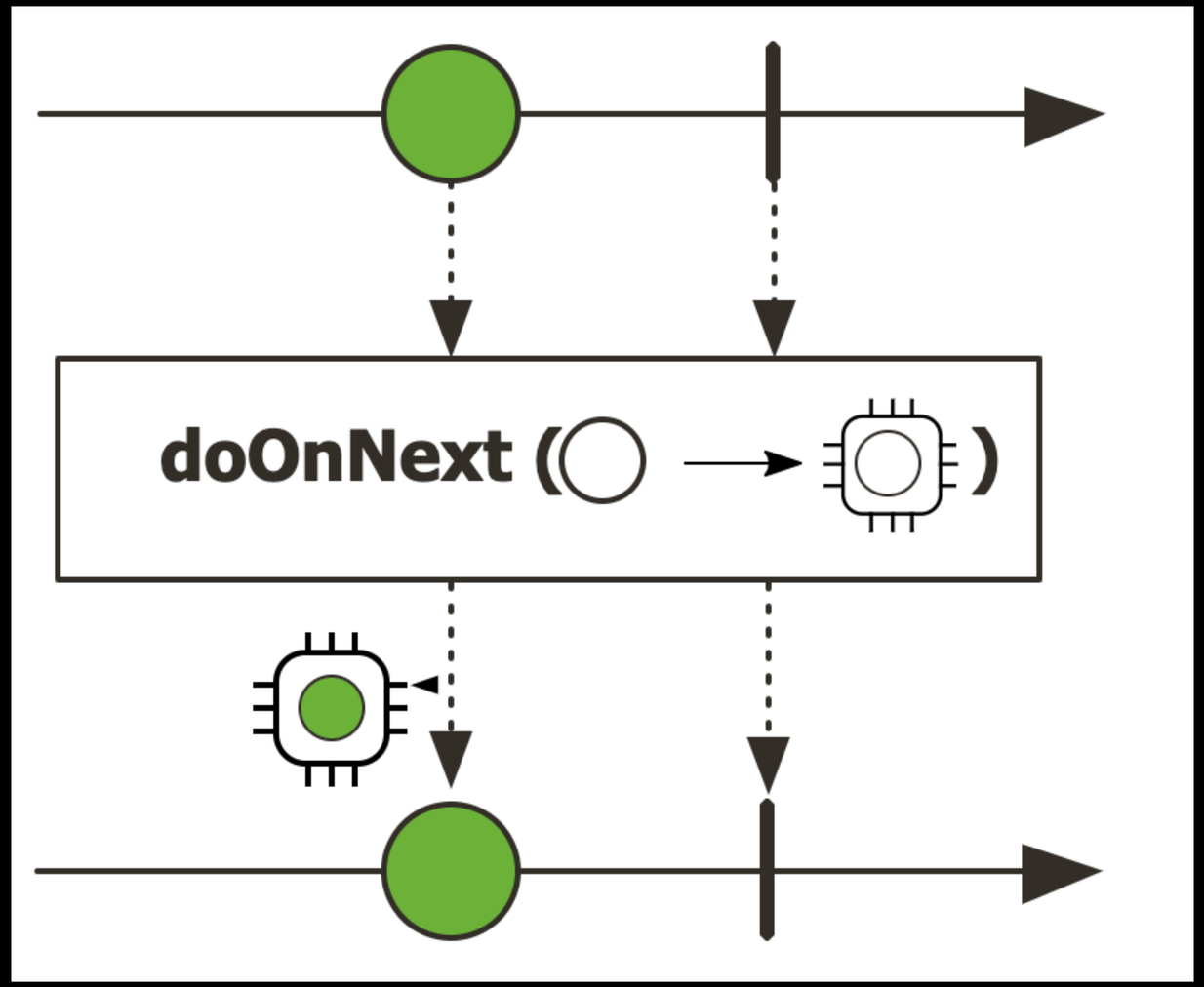
Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .zipWith()
    .zipWhen {    }
    .filter {    }
    .filterWhen {    }
    .map {    }
    .flatMap {    }
    .switchIfEmpty {    }
    .defaultIfEmpty {    }
```



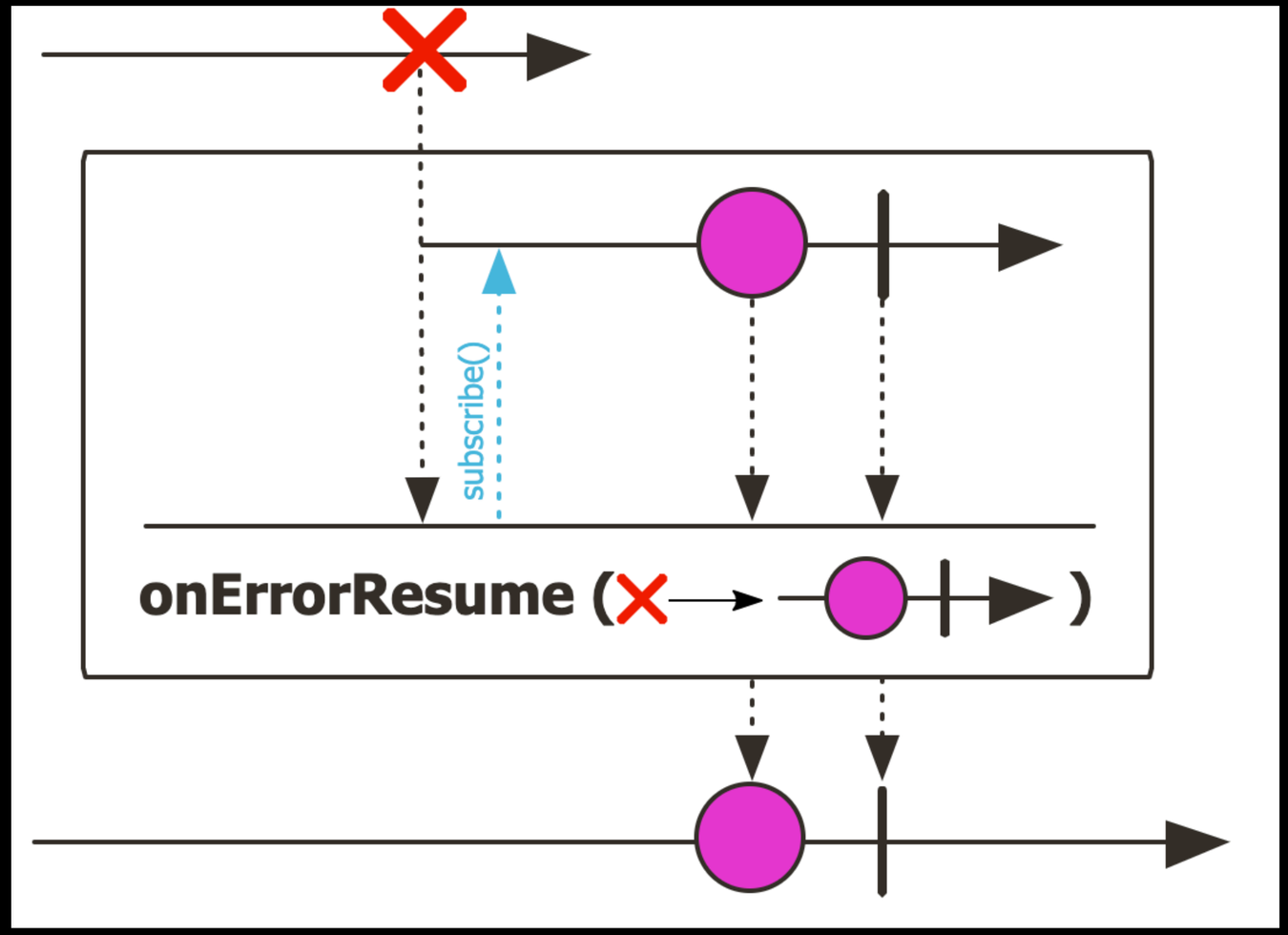
Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .doOnNext {    }
    .doOnComplete {    }
    .doOnError {    }
    .doOnSubscribe {    }
    .onErrorResume {    }
    .onErrorReturn {    }
    .onErrorStop {    }
    .onErrorContinue {    }
```



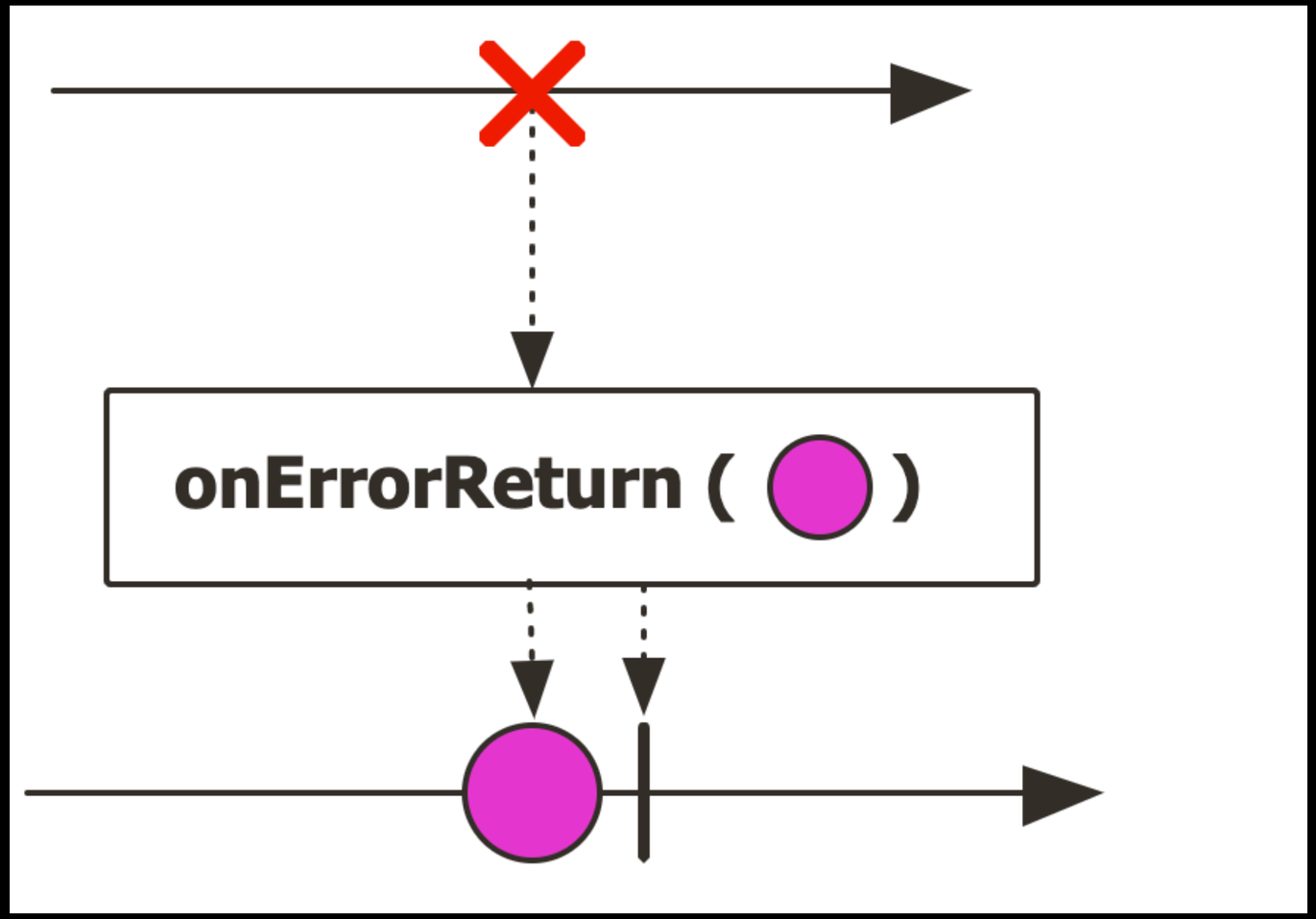
Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .doOnNext {    }
    .doOnComplete {    }
    .doOnError {    }
    .doOnSubscribe {    }
    .onErrorResume {    }
    .onErrorReturn {    }
    .onErrorStop {    }
    .onErrorContinue {    }
```

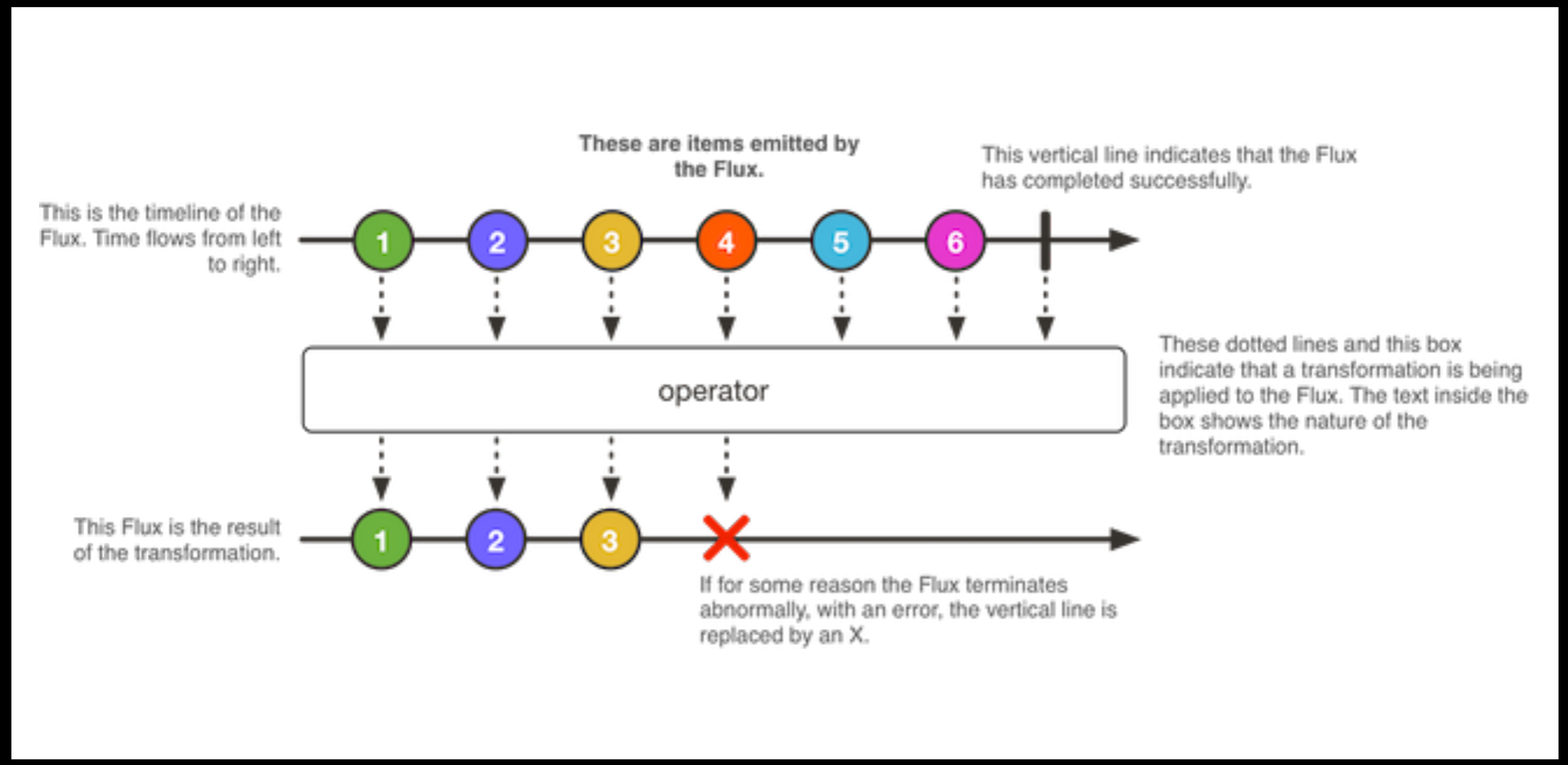


Mono an Asynchronous 0, 1 Result

```
Mono.just("객체가 들어가요")
    .doOnNext {    }
    .doOnComplete {    }
    .doOnError {    }
    .doOnSubscribe {    }
    .onErrorResume {    }
    .onErrorReturn {    }
    .onErrorStop {    }
    .onErrorContinue {    }
```



Flux an Asynchronous Sequence of 0, 1, N Items



Flux an Asynchronous Sequence of 0, 1, N Items

```
Flux.fromIterater("객체 리스트가 들어가요")
    .buffer()
    .cache()
    .concat()
    .merge()
    .firstWithSignal()
    .sampleFirst()
    .limitRate()
    .onBackpressureBuffer()
    .repeat()
    .retry()
    .collectList()
```

Flux an Asynchronous Sequence of 0, 1, N Items

Flux.fromIterater("객체 리스트가 들어가요")
.buffer()
.cache()
.concat()
.merge()
.firstWith
.sampleF
.limitRate
.onBackp
.repeat()
.retry()
.collectLi

```
override fun isVerified(
    userId: Long,
    query: BankAccountTransactionVerifyQuery,
): Mono<VerifyResult> {
    logger.debug("1시간 이내 입금 내역이 3회 이상 존재하는지 체크")
    return bankAccountDepositTransactionRepository
        .findAllByUserIdAndUpdatedAtAfter(userId, LocalDateTime.now().minusHours(1))
        .collectList()
        .map { if (it.size >= 3) VerifyResult.DETECT else VerifyResult.PASS }
        .switchIfEmpty(Mono.just(VerifyResult.PASS))
        .transformDeferred(CircuitBreakerOperator.of(mongoCircuitBreaker))
        .onErrorResume(CallNotPermittedException::class.java) { Mono.just(VerifyResult.PASS) }
        .onErrorResume(Exception::class.java) {
            logger.error("$this ${it.message} user_id: $userId, query: $query", it)
            Mono.just(VerifyResult.PASS)
        }
}
```

Flux an Asynchronous Sequence of 0, 1, N Items

```
Flux.fromIterater("객체 리스트가 들어가요")
    .subscribeOn(Schedulers.parallel())
    .subscribeOn(Schedulers.elastic())
    .subscribeOn(Schedulers.single())
    .subscribeOn(Schedulers.immediate())
```

주의할 점

하나라도 blocking 되는 코드가 있다면 느리다.

MultipleExceptions 처리를 잘하자!

empty 체크를 잘하자.

logging 레벨을 잘 조절해서 I/O 통신을 관리하자.

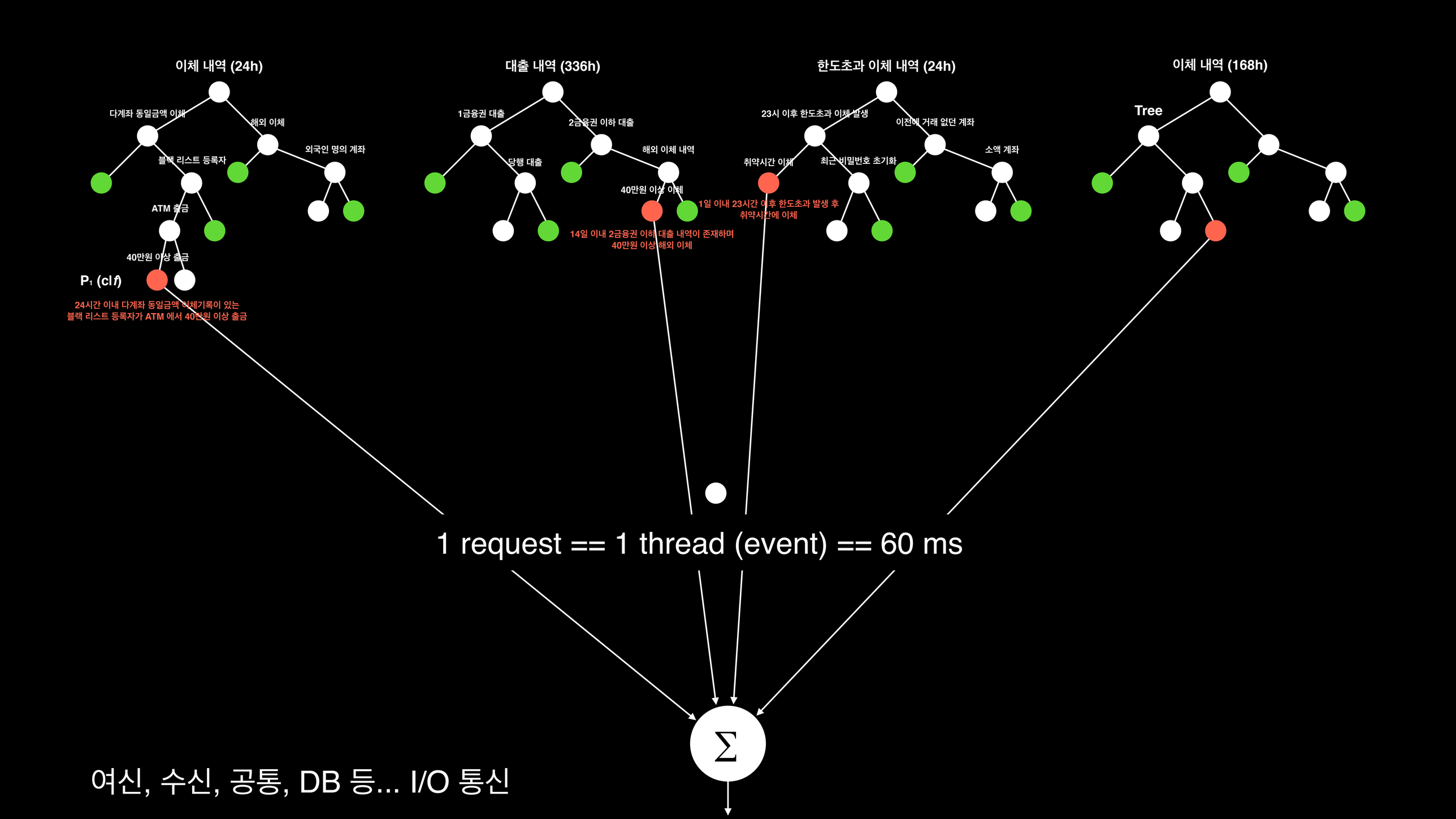
jdbc 를 사용하지 말자.

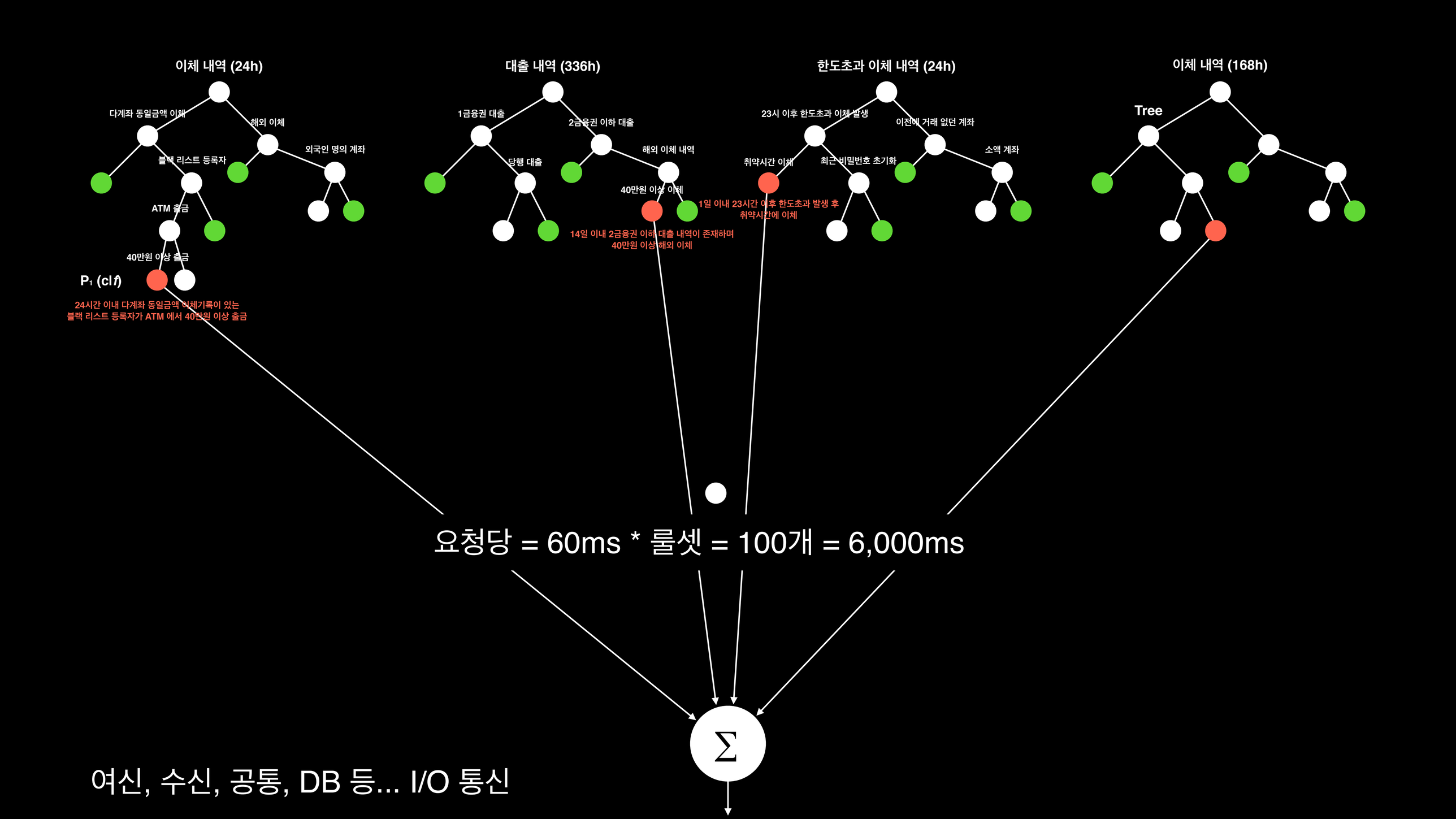
@EnableWebMvc, @EnableWebFlux 를 같이 쓰지 말자.

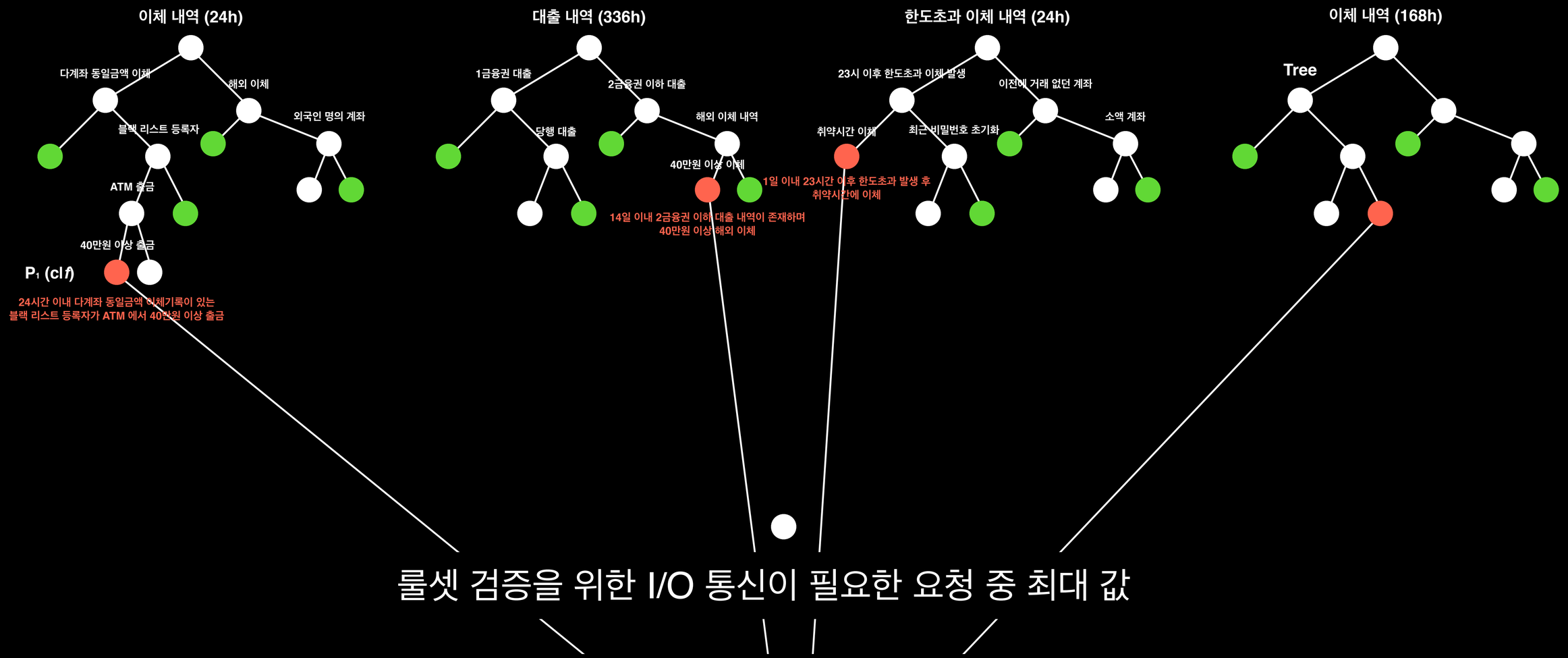
사용하는 라이브러리에 Mvc 의존성(?) 이 있는지 확인하자.

등등...

활용 사례 


이체 내역 (24h)
다계좌 동일금액 이체
해외 이체
블랙 리스트 등록자
외국인 명의 계좌
ATM 출금
40만원 이상 출금
P₁ (clf)
24시간 이내 다계좌 동일금액 이체기록이 있는
블랙 리스트 등록자가 ATM 에서 40만원 이상 출금
대출 내역 (336h)
1금융권 대출
2금융권 이하 대출
당행 대출
해외 이체 내역
40만원 이상 이체
14일 이내 2금융권 이하 대출 내역이 존재하며
40만원 이상 해외 이체
한도초과 이체 내역 (24h)
23시 이후 한도초과 이체 발생
이전에 거래 없던 계좌
취약시간 이체
최근 비밀번호 초기화
소액 계좌
1일 이내 23시간 이후 한도초과 발생 후
취약시간에 이체
이체 내역 (168h)
Tree
1 request == 1 thread (event) == 60 ms
Σ
여신, 수신, 공통, DB 등... I/O 통신


이체 내역 (24h)
다계좌 동일금액 이체
해외 이체
블랙 리스트 등록자
외국인 명의 계좌
ATM 출금
40만원 이상 출금
P₁ (clf)
24시간 이내 다계좌 동일금액 이체기록이 있는
블랙 리스트 등록자가 ATM 에서 40만원 이상 출금
대출 내역 (336h)
1금융권 대출
2금융권 이하 대출
당행 대출
해외 이체 내역
40만원 이상 이체
14일 이내 2금융권 이하 대출 내역이 존재하며
40만원 이상 해외 이체
한도초과 이체 내역 (24h)
23시 이후 한도초과 이체 발생
이전에 거래 없던 계좌
취약시간 이체
최근 비밀번호 초기화
소액 계좌
1일 이내 23시간 이후 한도초과 발생 후
취약시간에 이체
이체 내역 (168h)
Tree
요청당 = 60ms * 룰셋 = 100개 = 6,000ms
Σ
여신, 수신, 공통, DB 등... I/O 통신



룰셋 검증을 위한 I/O 통신이 필요한 요청 중 최대 값

```
2021-04-16 14:26:55,566 INFO  lettuce-kqueueEventLoop-7-10 (c.t.f.i.a.e.i.v.s.c.v.ExPostVerifier) [-:-] rule_count: 79 verified
(elapsed_time: 76ms)
```

그럼 무조건 웹플럭스가 좋은거에요?

놉! 

(구조상 반드시) JDBC 를 사용해야하며 대용량 트래픽 처리보다는
빠른 구현을 필요로 하는 프로젝트를 해야한다면? MVC

다양한 MSA 서비스를 호출하거나 많은 트래픽을 핸들링 해야하며
동시에 빠른 응답속도를 위한 프로젝트를 해야한다면? WebFlux

MVC ~~vs~~ WebFlux