BlockingQueue 파헤치기

메시지 브로커(MMMQ) 개발기 1

모코

목차

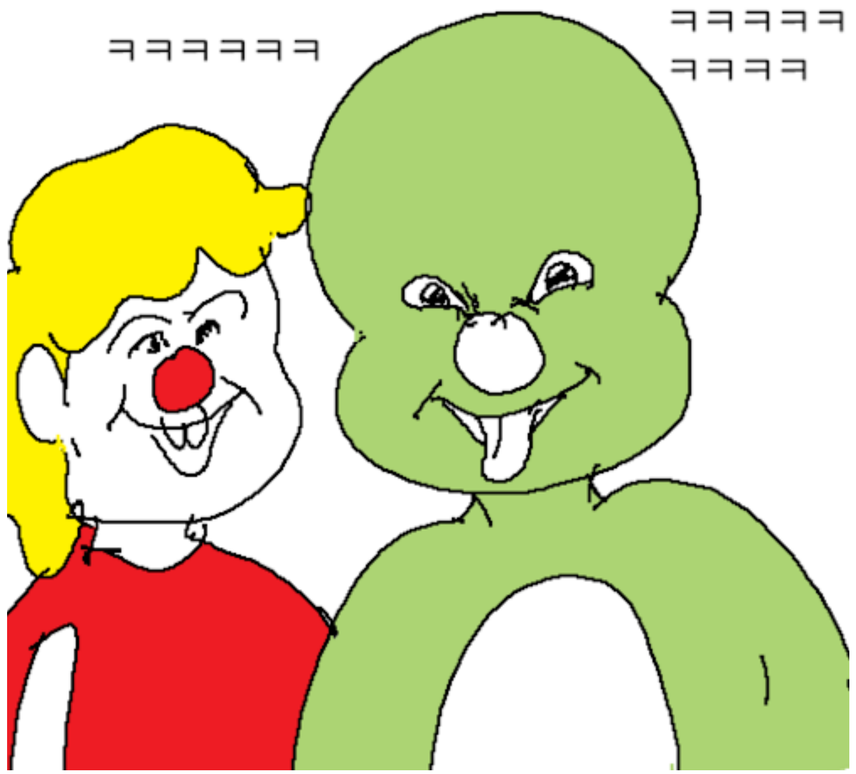
1부: MMMQ

왜 만들게 되었는가


우아한테크코스


어이 모코
머랭 어서오고.


메시지큐 만들어볼래?
재밌겠는데?


ㅋㅋㅋㅋㅋㅋ
ㅋㅋㅋㅋㅋ
ㅋㅋㅋㅋ

왜 만들게 되었는가

요즘 **MSA가 핫하다**던데...

우리 프로젝트는 MSA를 도입할 규모가 안 된단 말이지

근데 MSA에서 서비스 간 결합을 느슨하게 하는 **핵심은 메시지큐**잖아

그럼 그 핵심인 메시지큐를 직접 만들어보는 건 어때?

직접 만들어보면서 **메시지큐의 철학과 동작 원리를 깊이 이해**해보자.

1부: MMMQ

메시지큐 라이브러리

라이브러리 종류

라이브러리 종류

메시지 큐 표준 프로토콜 (법)

라이브러리 종류

메시지 큐 표준 프로토콜 (법)

- 대표적인 AMQP 구현체 (모범시민)
- 정석적인 메시지 큐
- AMQP 표준 스펙을 100% 따름
- 복잡한 라우팅을 지원함
- 확실한 메시지 전송을 보장함(ACK)

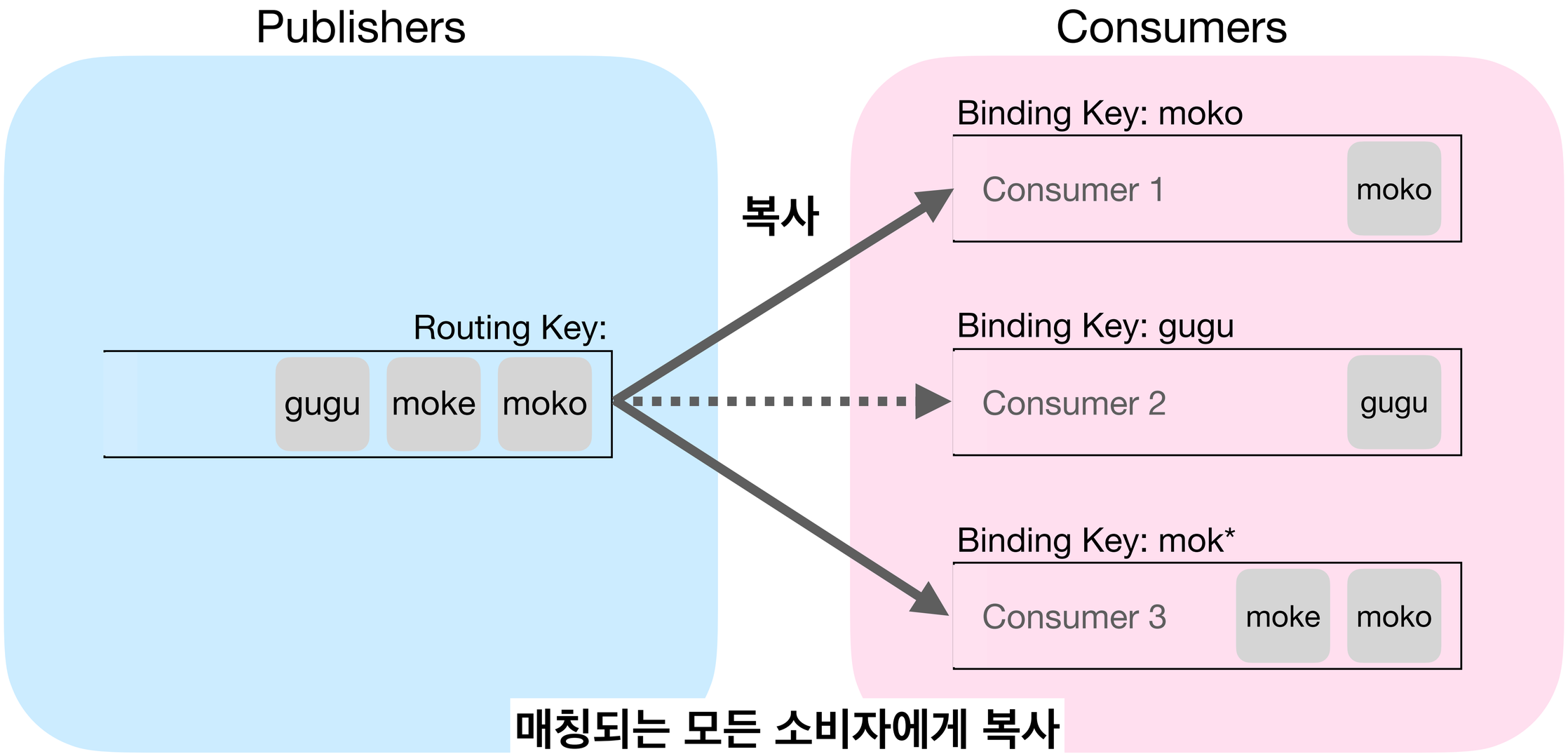
라이브러리 종류

메시지 큐 표준 프로토콜 (법)

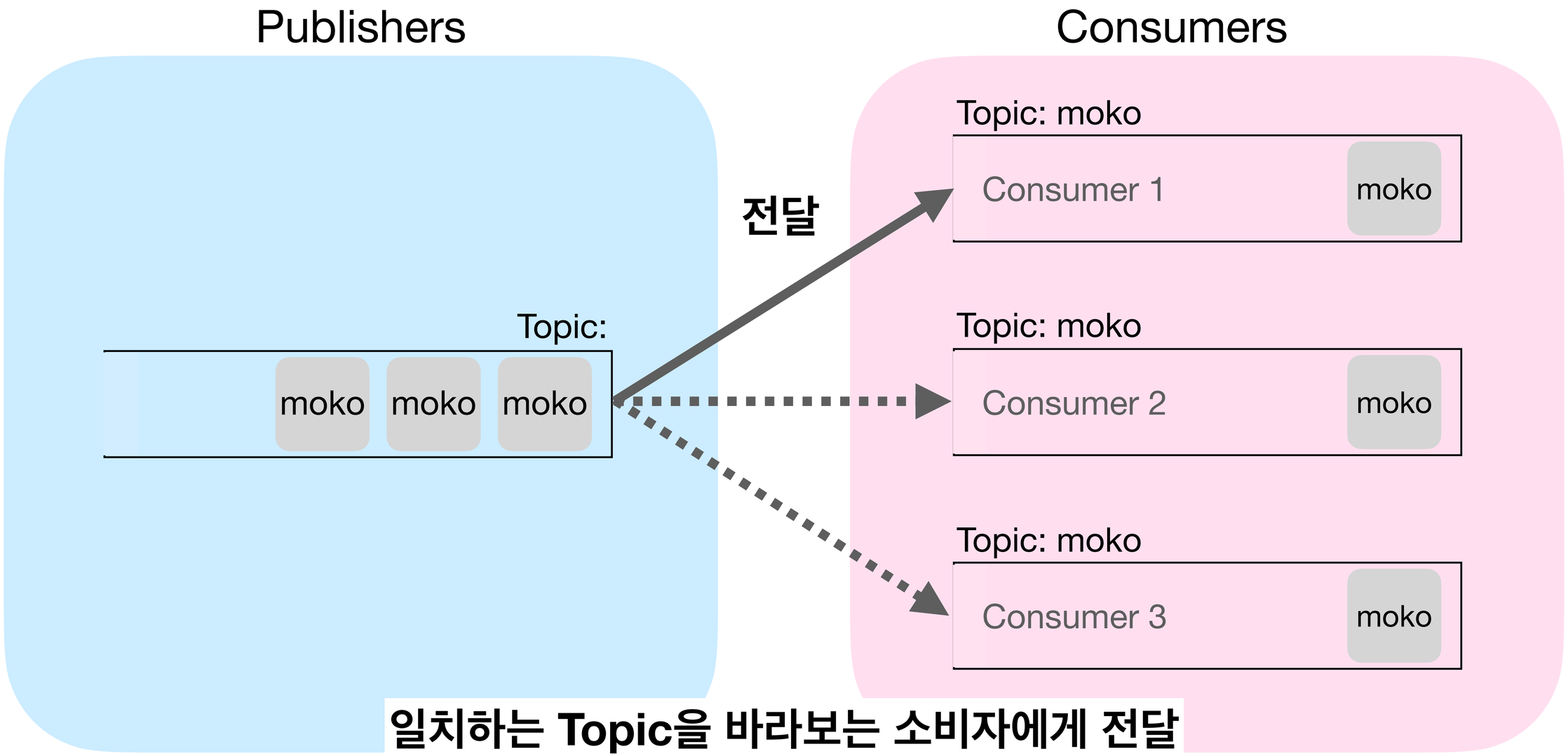
- 대표적인 AMQP 구현체 (모범시민)
- 정석적인 메시지 큐
- AMQP 표준 스펙을 100% 따름
- 복잡한 라우팅을 지원함
- 확실한 메시지 전송을 보장함(ACK)

- 독자 프로토콜 (탈세 부자)
- 엇나가서 오히려 좋은 메시지 큐
- AMQP 표준 스펙을 따르지 않고 독자적 스펙을 고수함
- 영속화 제공(누락 방지, Replay)
- 압도적인 처리량 제공

(AMQP)패턴 매칭 라우팅

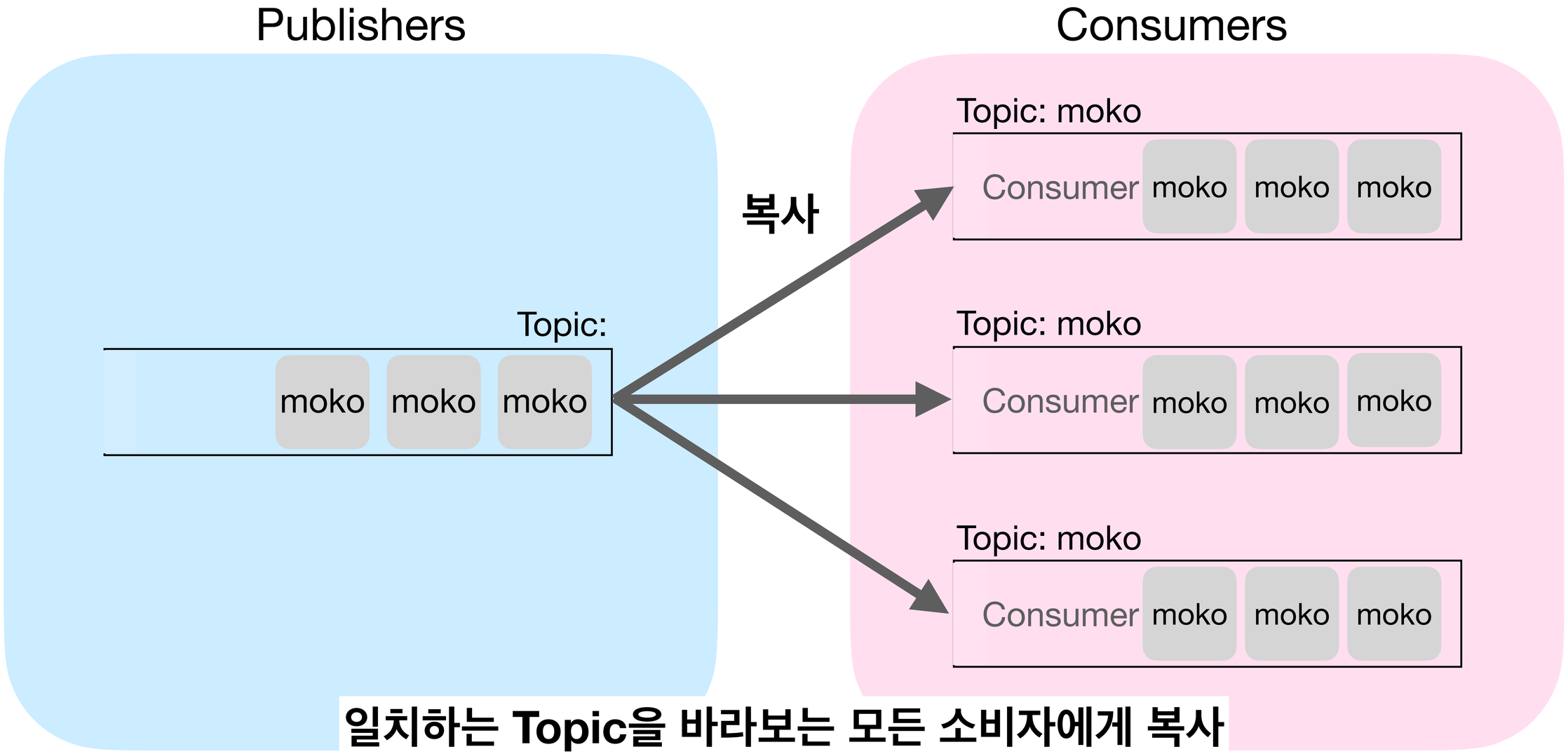


(Kafka)대기열 방식



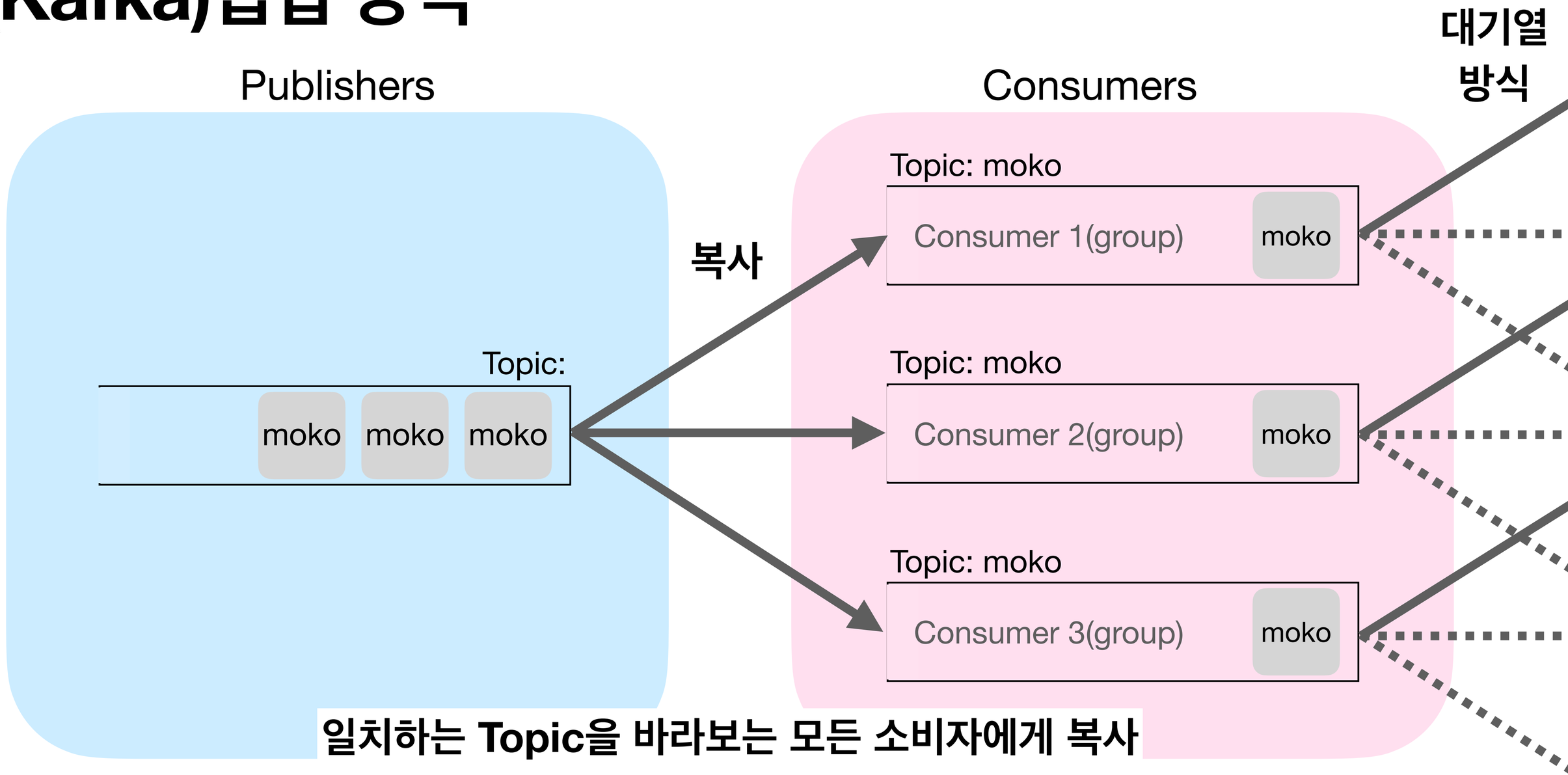
일치하는 Topic을 바라보는 소비자에게 전달

(Kafka)펍섭 방식



일치하는 Topic을 바라보는 모든 소비자에게 복사

(Kafka)펍섭 방식



일치하는 Topic을 바라보는 모든 소비자에게 복사

1부: MMMQ

MMMQ

MMMQ - 메시지 브로커

2025.10 ~ 진행 중(2개월)

2인 페어 프로그래밍 | Github Readme | Release

Java Spring Boot Jitpack

메시지 큐의 내부 동작 원리를 깊이 이해하기 위해 밑바닥부터 구현 중인 **메시지 브로커 프로젝트**입니다.
토픽 기반 Pub/Sub 구조로 토픽별 메시지 라우팅을 수행합니다.

https://github.com/moko-meringue/mmmq

MMMQ - 메시지 브로커

2025.10 ~ 진행 중(2개월)

2인 페어 프로그래밍 | Github Readme | Release

Java Spring Boot Jitpack

메시지 큐의 내부 동작 원리를 깊이 이해하기 위해 밑바닥부터 구현 중인 메시지 브로커 프로젝트입니다.
토픽 기반 Pub/Sub 구조로 토픽별 메시지 라우팅을 수행합니다.

https://github.com/moko-meringue/mmmq

MMMQ - 메시지 브로커

2025.10 ~ 진행 중(2개월)

2인 페어 프로그래밍 | Github Readme | Release

Java Spring Boot Jitpack

메시지 큐의 내부 동작 원리를 깊이 이해하기 위해 밑바닥부터 구현 중인 **메시지 브로커 프로젝트**입니다.
토픽 기반 Pub/Sub 구조로 토픽별 메시지 라우팅을 수행합니다.

메시지큐의 철학은 무엇일까?

https://github.com/moko-meringue/mmmq

MMMQ - 메시지 브로커

2025.10 ~ 진행 중(2개월)

2인 페어 프로그래밍 | Github Readme | Release

Java Spring Boot Jitpack

메시지 큐의 내부 동작 원리를 깊이 이해하기 위해 밑바닥부터 구현 중인 **메시지 브로커 프로젝트**입니다.
토픽 기반 Pub/Sub 구조로 토픽별 메시지 라우팅을 수행합니다.

메시지큐의 철학은 무엇일까?

생산자는 소비자를 몰라야 한다(디커플링)

https://github.com/moko-meringue/mmmq

MMMQ - 메시지 브로커

2025.10 ~ 진행 중(2개월)

2인 페어 프로그래밍 | Github Readme | Release

Java Spring Boot Jitpack

메시지 큐의 내부 동작 원리를 깊이 이해하기 위해 밑바닥부터 구현 중인 **메시지 브로커 프로젝트**입니다.
토픽 기반 Pub/Sub 구조로 토픽별 메시지 라우팅을 수행합니다.

메시지큐의 철학은 무엇일까?

생산자는 소비자를 몰라야 한다(디커플링)

소비자가 몇 명인지조차 몰라야겠네?(1명인지, n명인지)

https://github.com/moko-meringue/mmmq

MMMQ - 메시지 브로커

2025.10 ~ 진행 중(2개월)

2인 페어 프로그래밍 | Github Readme | Release

Java Spring Boot Jitpack

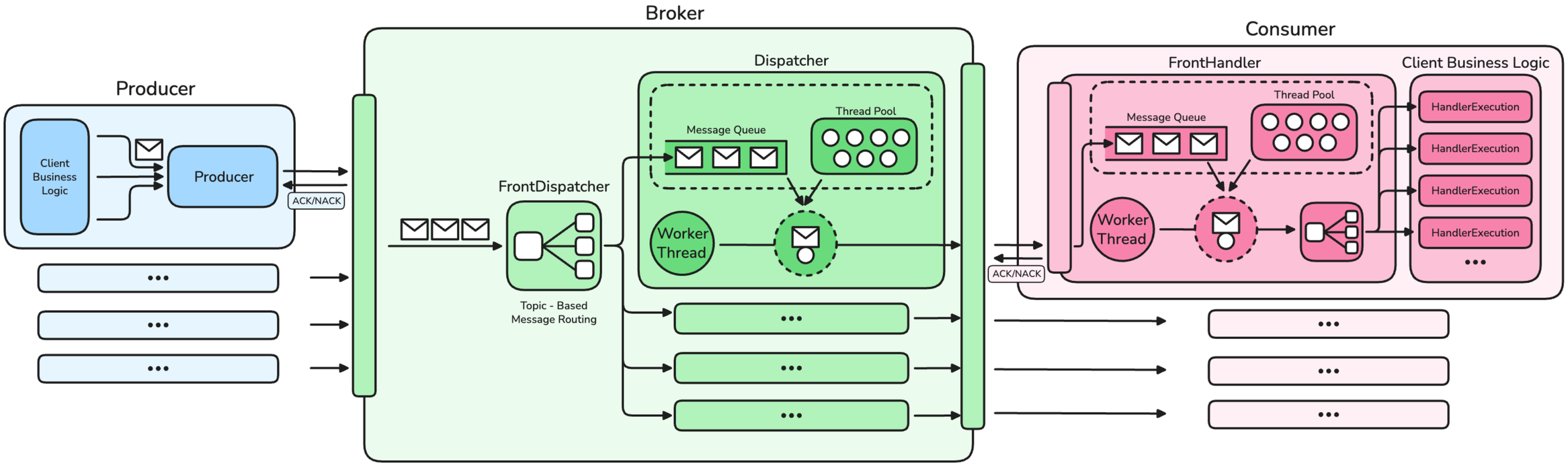
메시지 큐의 내부 동작 원리를 깊이 이해하기 위해 밑바닥부터 구현 중인 **메시지 브로커 프로젝트**입니다.
토픽 기반 Pub/Sub 구조로 토픽별 메시지 라우팅을 수행합니다.

메시지큐의 철학은 무엇일까?

생산자는 소비자를 몰라야 한다(디커플링)

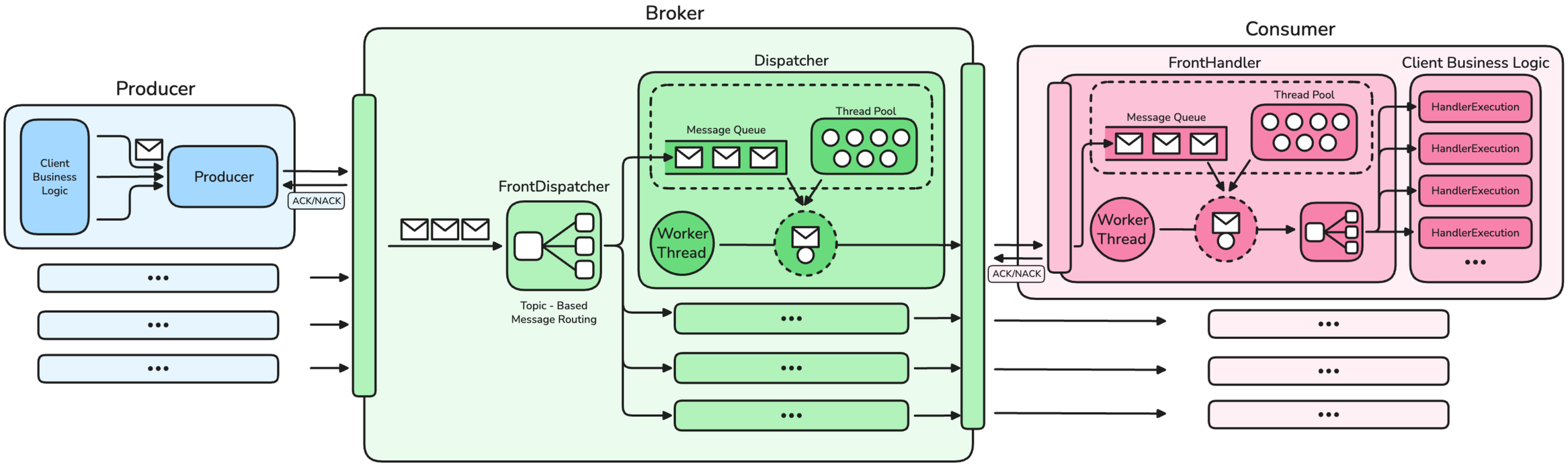
소비자가 몇 명인지조차 몰라야겠네?(1명인지, n명인지)

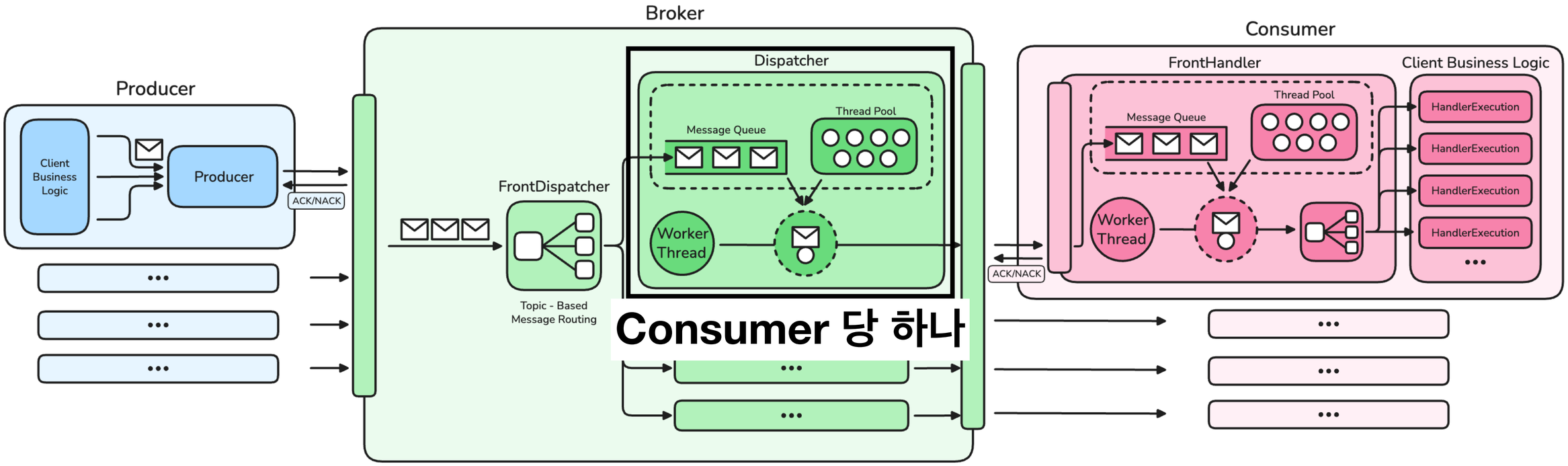
https://github.com/moko-meringue/mmmq

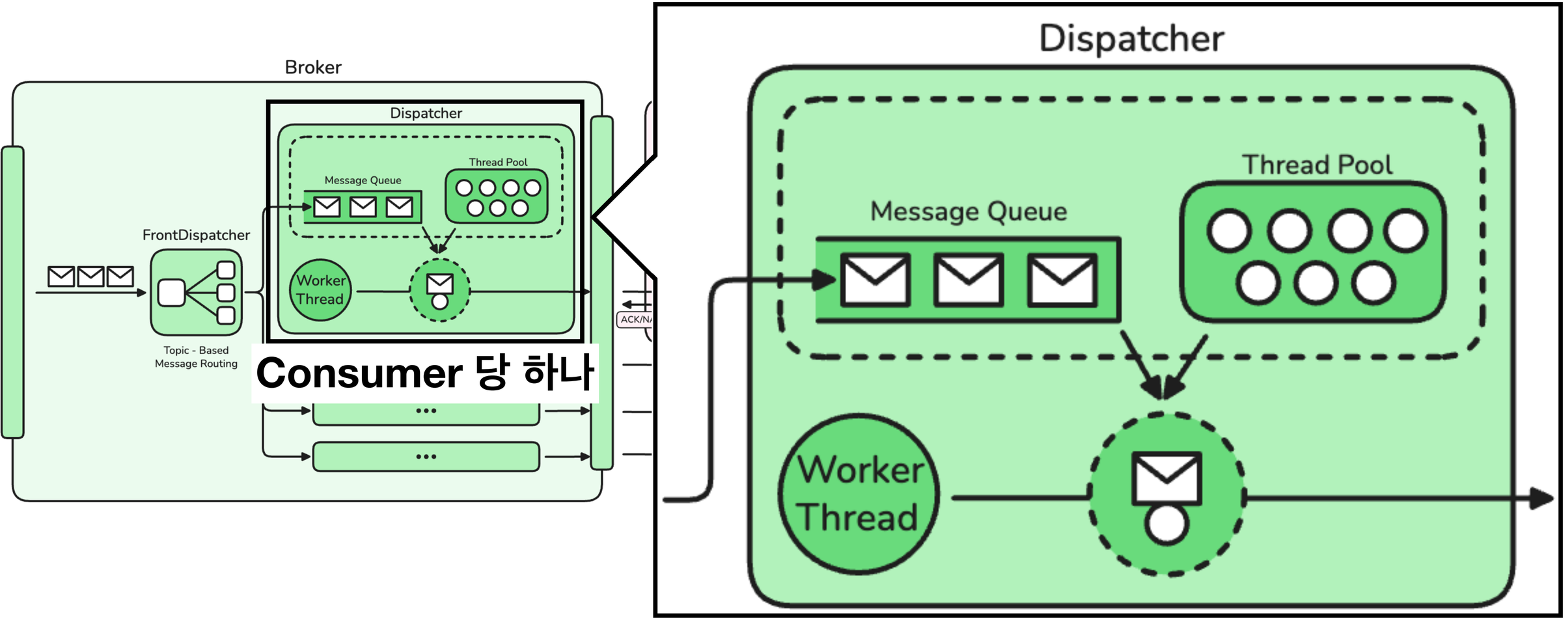

Broker
Producer
Client Business Logic
Producer
ACK/NACK
FrontDispatcher
Topic - Based Message Routing
Dispatcher
Message Queue
Thread Pool
Worker Thread
Consumer
FrontHandler
Client Business Logic
Message Queue
Thread Pool
Worker Thread
HandlerExecution
HandlerExecution
HandlerExecution
HandlerExecution
ACK/NACK

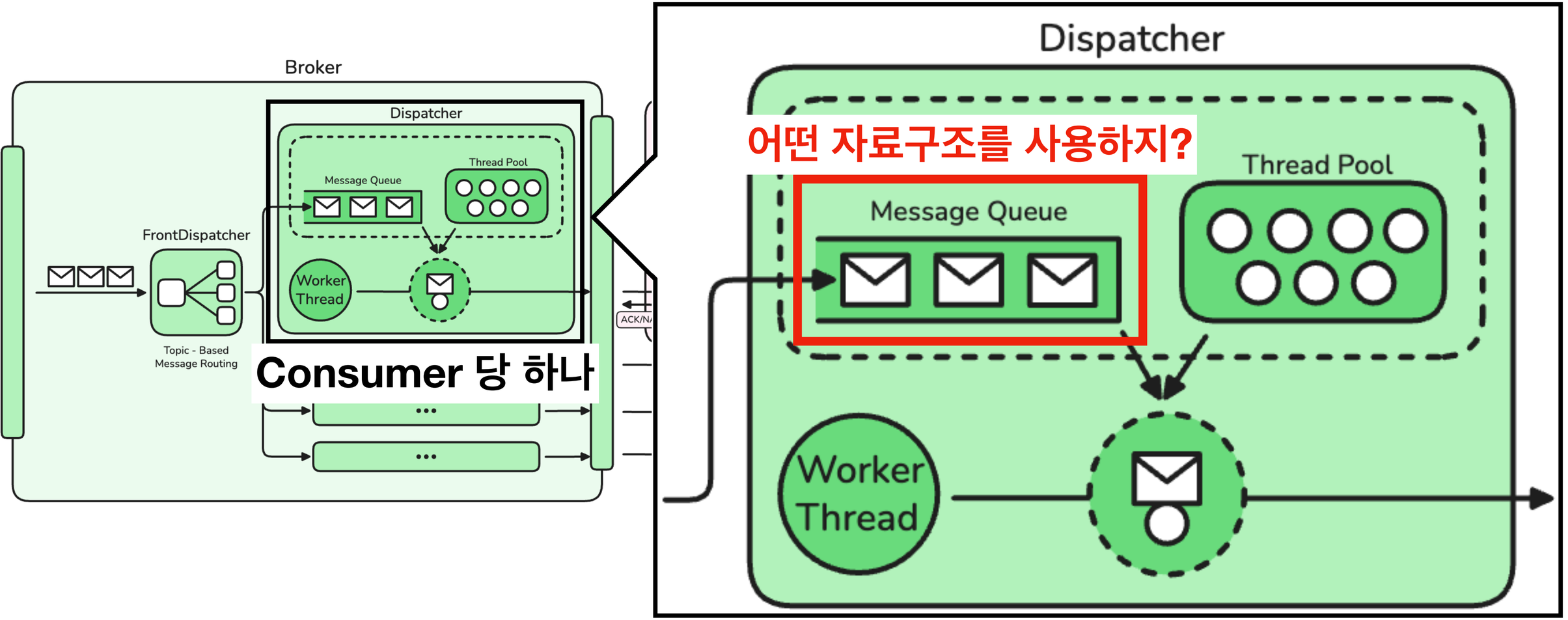
2부: 메시지 큐 자료구조

자료구조 선정


Broker
Producer
Client Business Logic
Producer
ACK/NACK
FrontDispatcher
Topic - Based Message Routing
Dispatcher
Message Queue
Thread Pool
Worker Thread
Consumer
FrontHandler
Client Business Logic
Message Queue
Thread Pool
Worker Thread
HandlerExecution
HandlerExecution
HandlerExecution
HandlerExecution
ACK/NACK

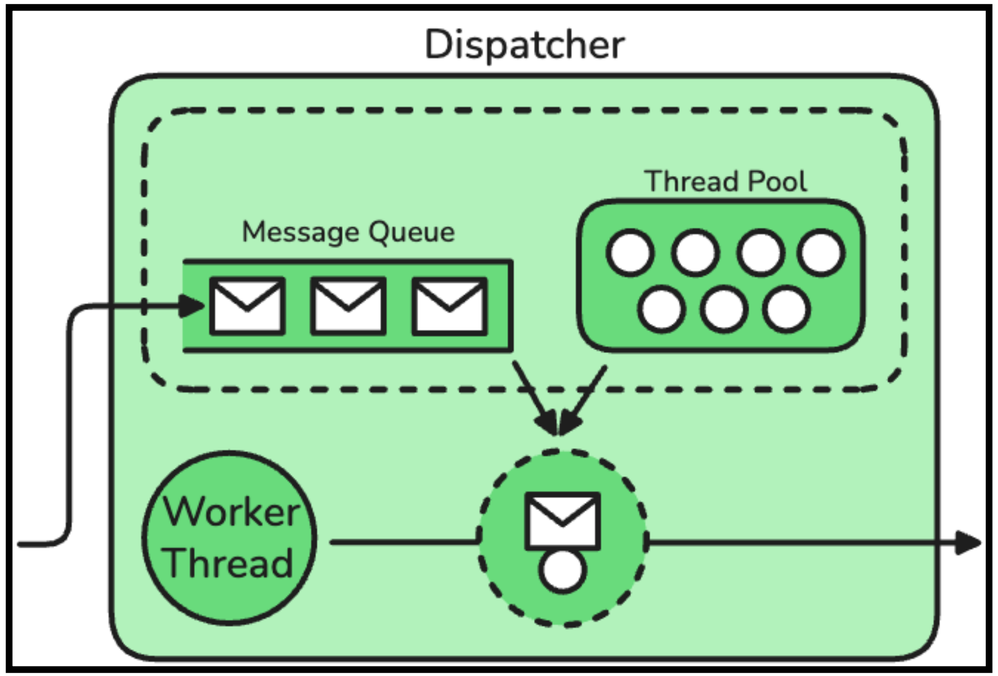

Producer
Client Business Logic
Producer
ACK/NACK
Broker
FrontDispatcher
Topic - Based Message Routing
Dispatcher
Message Queue
Thread Pool
Worker Thread
Consumer 당 하나
Consumer
FrontHandler
Client Business Logic
Message Queue
Thread Pool
Worker Thread
HandlerExecution
HandlerExecution
HandlerExecution
HandlerExecution
ACK/NACK


Broker
Dispatcher
Thread Pool
Message Queue
FrontDispatcher
Worker Thread
Topic - Based Message Routing
Consumer 당 하나
Dispatcher
Thread Pool
Message Queue
Worker Thread


Broker
Dispatcher
Thread Pool
Message Queue
FrontDispatcher
Worker Thread
Topic - Based Message Routing
Consumer 당 하나
Dispatcher
어떤 자료구조를 사용하지?
Thread Pool
Message Queue
Worker Thread

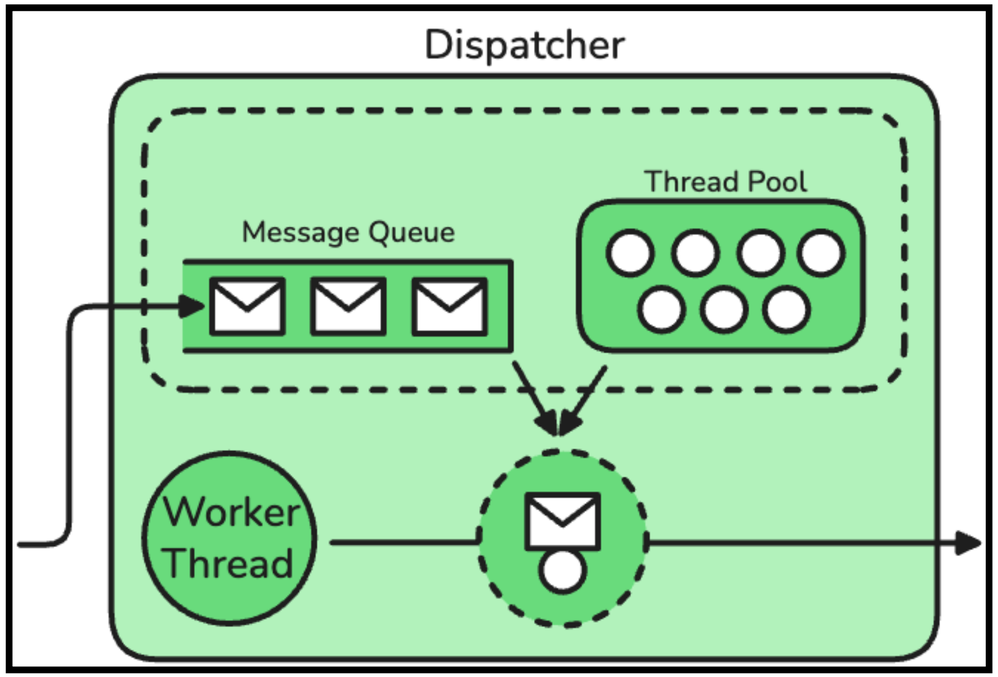
메시지 큐 자료구조

- 동시에 많은 메시지가 들어올 수 있다.
- 워커 스레드가 메시지를 소비한다.
  - 워커 스레드는 큐가 비어있는지 확인해야 한다.



메시지 큐 자료구조

- 동시에 많은 메시지가 들어올 수 있다.
- 워커 스레드가 메시지를 소비한다.
  - 워커 스레드는 큐가 비어있는지 확인해야 한다.




```
while(true) {
    if (messageQueue.size() > 0) {
        Message message = messageQueue.poll();
        // Process the message
    }
}
```

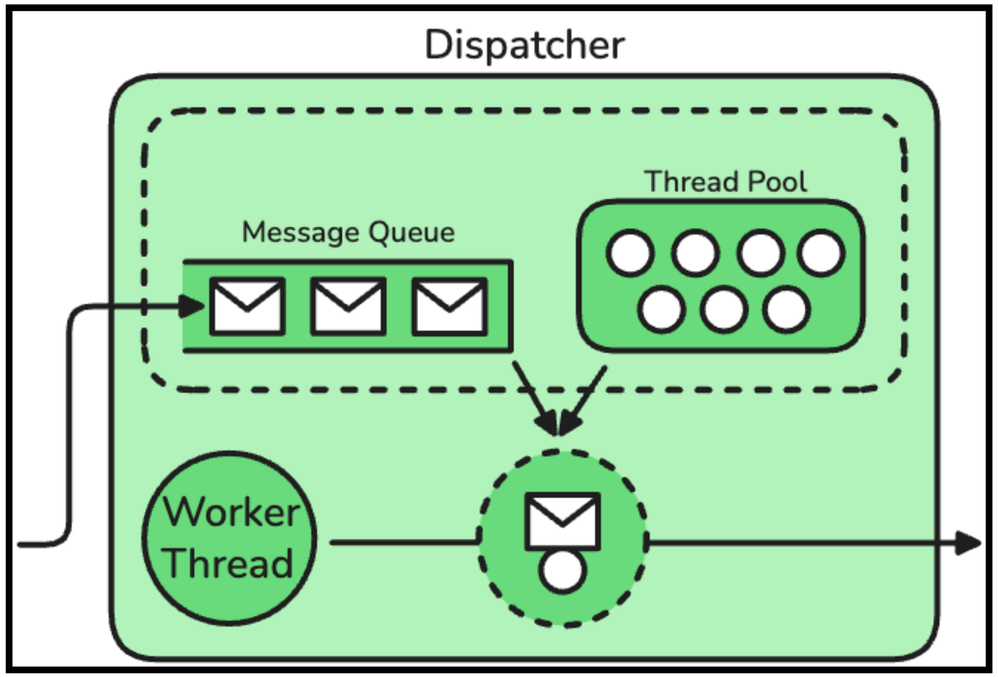
일반 큐: Busy Waiting 발생(CPU 낭비)

```
while(true) {
    Message message = messageQueue.take();
    // Process the message
}
```

BlockingQueue: 메시지 인입까지 스레드 대기

메시지 큐 자료구조

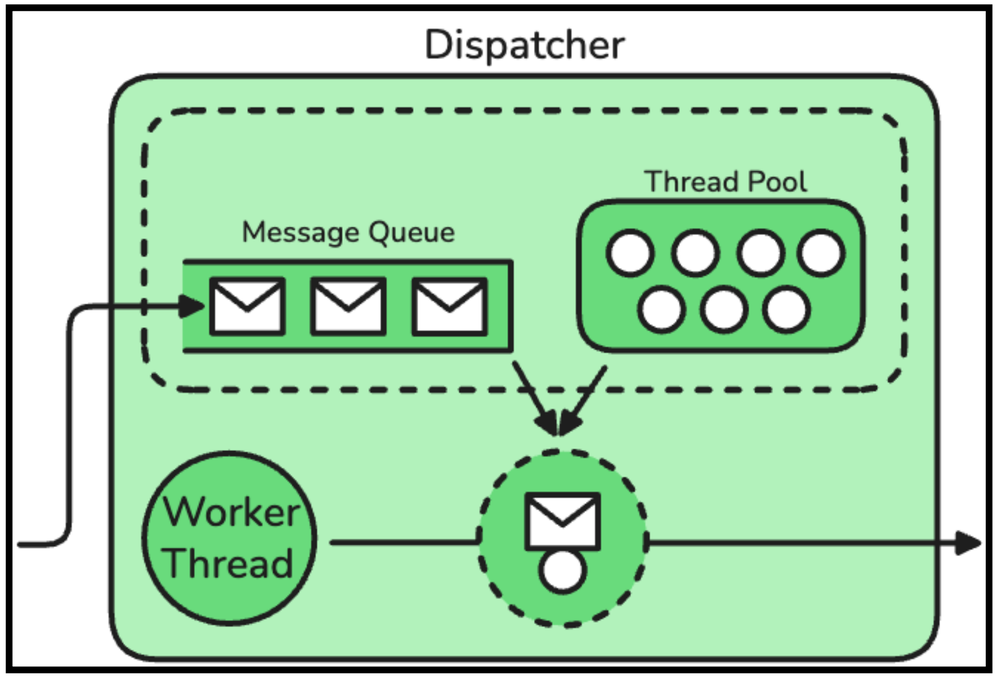
- 동시에 많은 메시지가 들어올 수 있다.
- 워커 스레드가 메시지를 소비한다.
  - 워커 스레드는 큐가 비어있는지 확인해야 한다.



	ArrayBlockingQueue	LinkedBlockingQueue
큐 최대 크기	최초 생성 시 할당 후 수정 불가	사실상 무한 (기본 21억)
동시성	안좋음 (삽입/삭제 단일 락 구조)	좋음 (삽입/삭제 이중 락 구조)

메시지 큐 자료구조

- 동시에 많은 메시지가 들어올 수 있다.
- 워커 스레드가 메시지를 소비한다.
  - 워커 스레드는 큐가 비어있는지 확인해야 한다.



	ArrayBlockingQueue	LinkedBlockingQueue
큐 최대 크기	최초 생성 시 할당 후 수정 불가	사실상 무한 (기본 21억)
동시성	안좋음 (삽입/삭제 단일 락 구조)	좋음 (삽입/삭제 이중 락 구조)

2부: 메시지 큐 자료구조

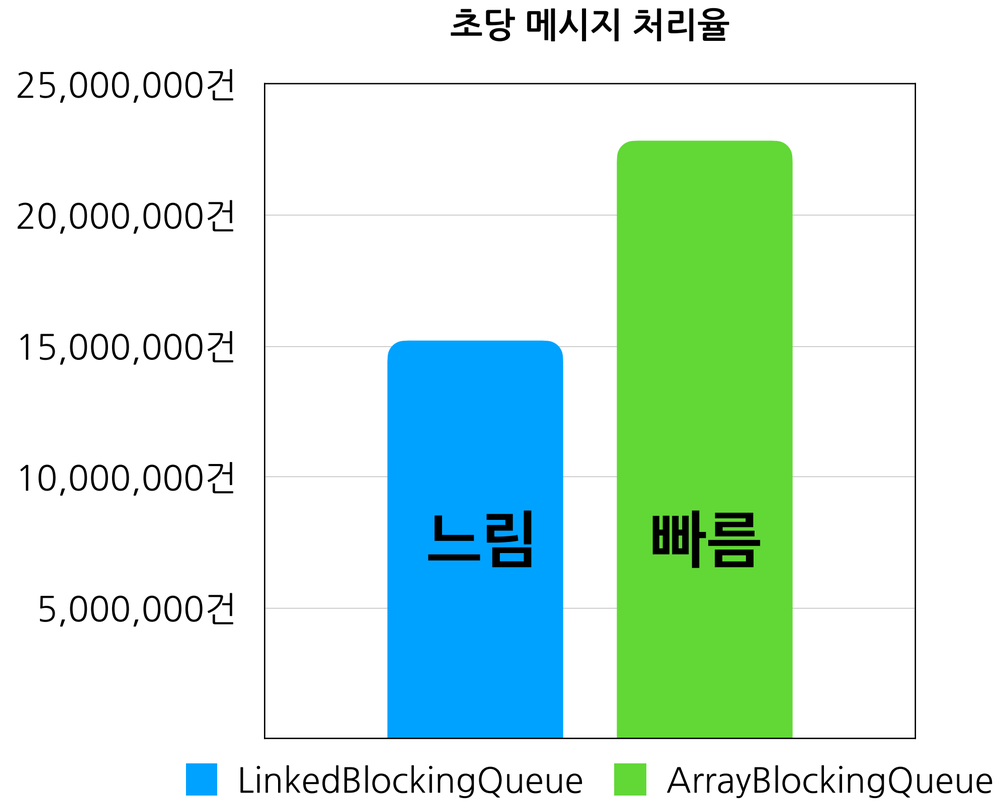
부하 테스트

부하 테스트 진행

MMMQ 환경을 재현하여 테스트

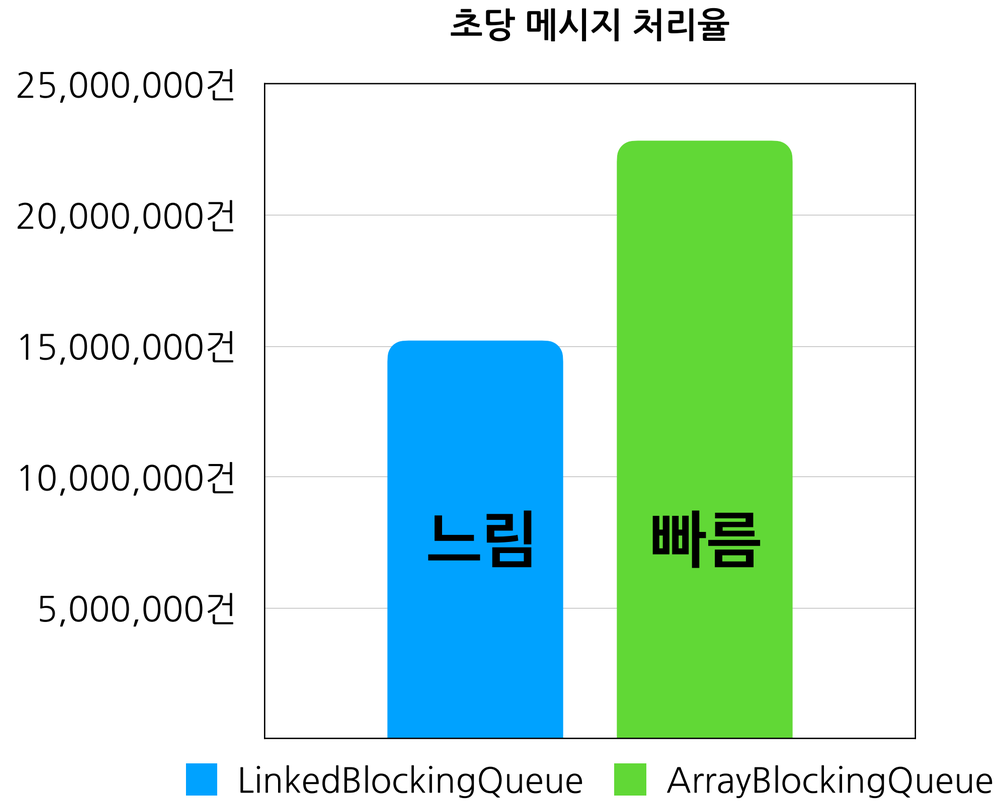
부하 테스트 진행

MMMQ 환경을 재현하여 테스트



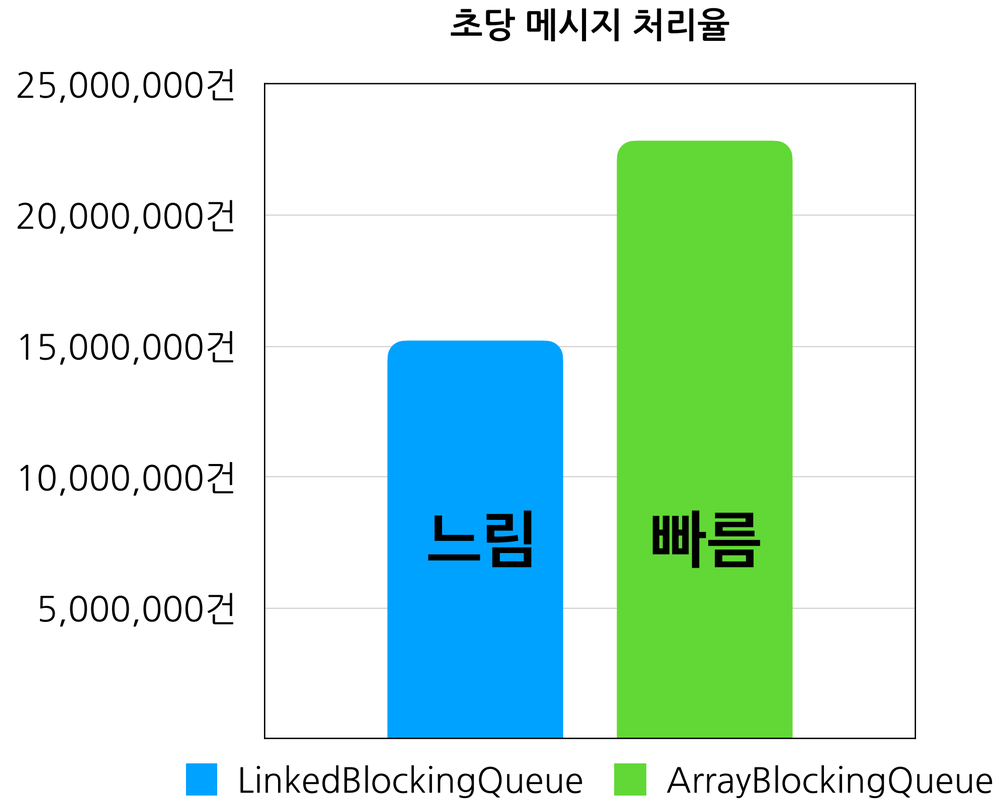
부하 테스트 진행

MMMQ 환경을 재현하여 테스트



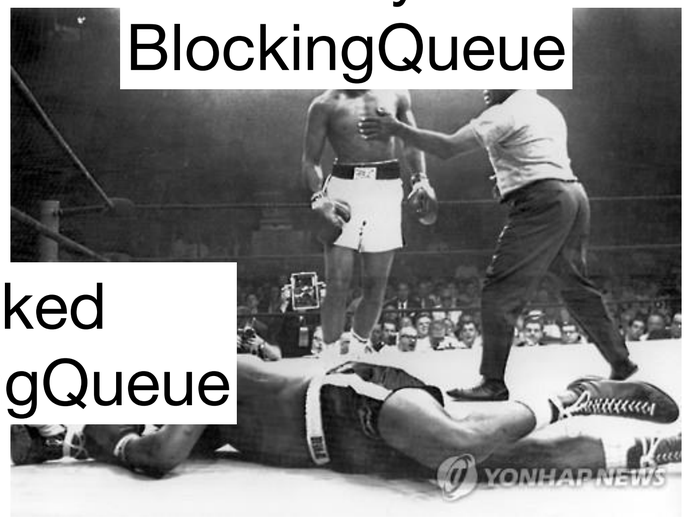
부하 테스트 진행

MMMQ 환경을 재현하여 테스트



ArrayBlockingQueue가 더 빠르게 처리했다?

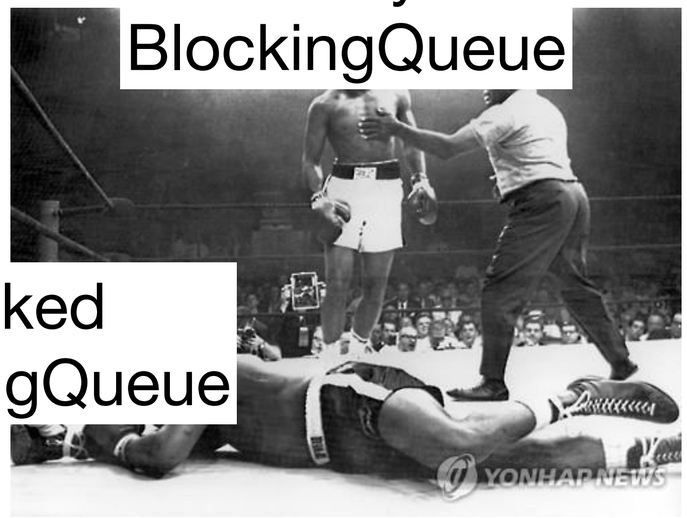
Array
BlockingQueue



Linked
BlockingQueue

ArrayBlockingQueue가 더 빠르게 처리했다?

Array
BlockingQueue



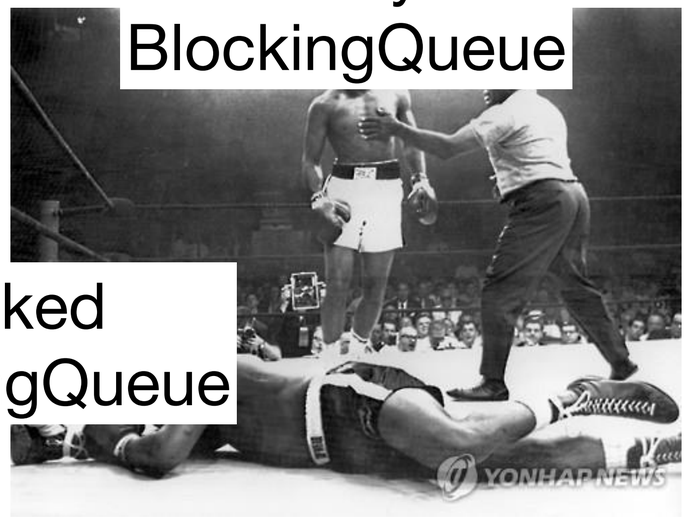
Linked
BlockingQueue

ArrayBlockingQueue가 더 빠르게 처리했다?

Linked가 락 분리로 속도도 빠르고

큐 용량도 제한이 없어서 유연할텐데?

Array
BlockingQueue



Linked
BlockingQueue

ArrayBlockingQueue가 더 빠르게 처리했다?

Linked가 락 분리로 속도도 빠르고

큐 용량도 제한이 없어서 유연할텐데?

캐시 지역성 **경합 확률**

캐시 지역성

경합 확률

캐시 지역성

ArrayBlockingQueue

: 좋다(배열 기반)

LinkedBlockingQueue

: 안좋다(연결 리스트 기반)

캐시 지역성: Array > Linked

경합 확률

캐시 지역성

ArrayBlockingQueue

: 좋다(배열 기반)

LinkedBlockingQueue

: 안좋다(연결 리스트 기반)

캐시 지역성: Array > Linked

경합 확률

락 분리로 인한 성능 우위

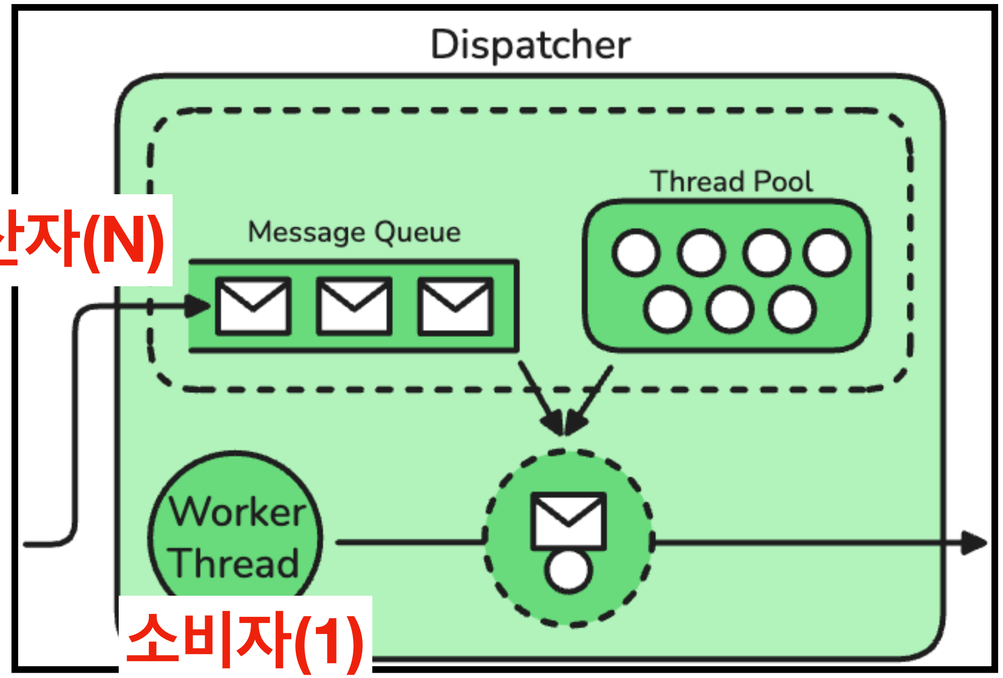
: Linked > Array

락 분리로 인한 성능 영향

: 미미(소비자 경합이 없음)

락 성능: Array ≈ Linked

캐시 지역성



캐시 지역성: Array > Linked

경합 확률

락 분리로 인한 성능 우위

: Linked > Array

락 분리로 인한 성능 영향

: 미미(소비자 경합이 없음)

락 성능: Array ≈ Linked

캐시 지역성

ArrayBlockingQueue

: 좋다(배열 기반)

LinkedBlockingQueue

: 안좋다(연결 리스트 기반)

캐시 지역성: Array > Linked

경합 확률

락 분리로 인한 성능 우위

: Linked > Array

락 분리로 인한 성능 영향

: 미미(소비자 경합이 없음)

락 성능: Array ≈ Linked

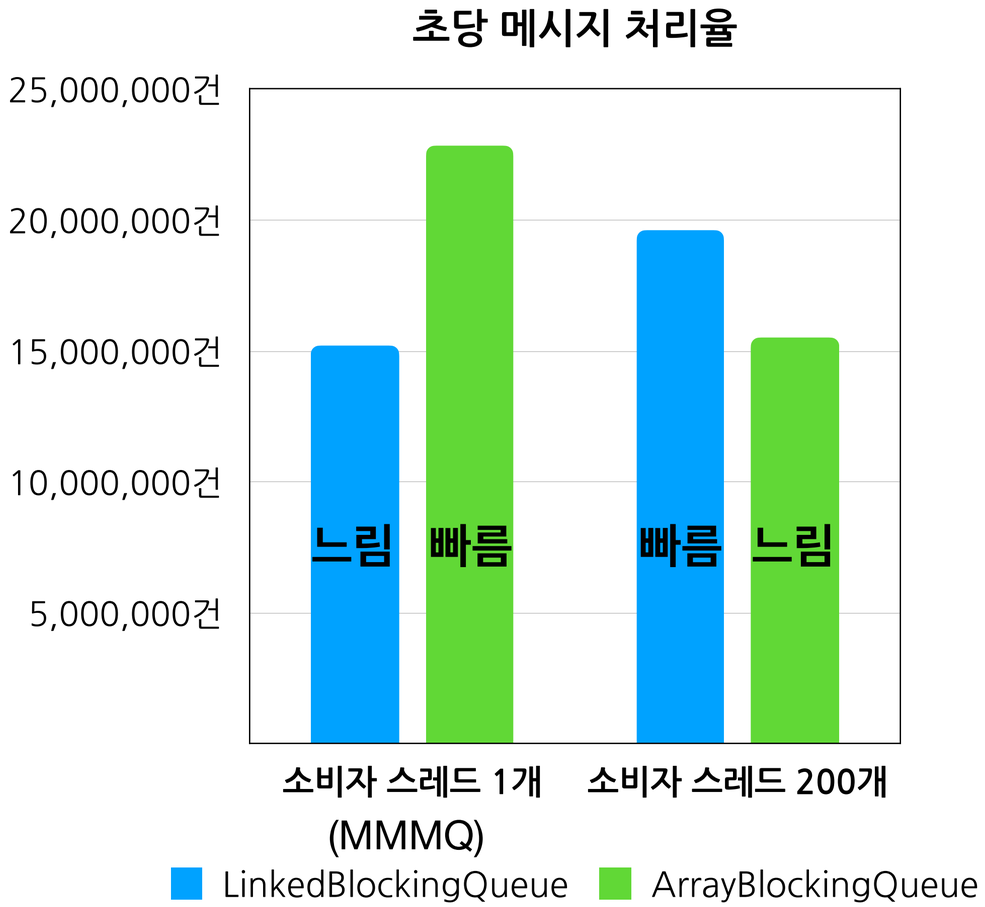
경합 확률이 낮아 락 성능 차이가 미미한 시점에서,

캐시 지역성이 우수한 ArrayBlockingQueue가 더 뛰어난 성능을 보여준다.

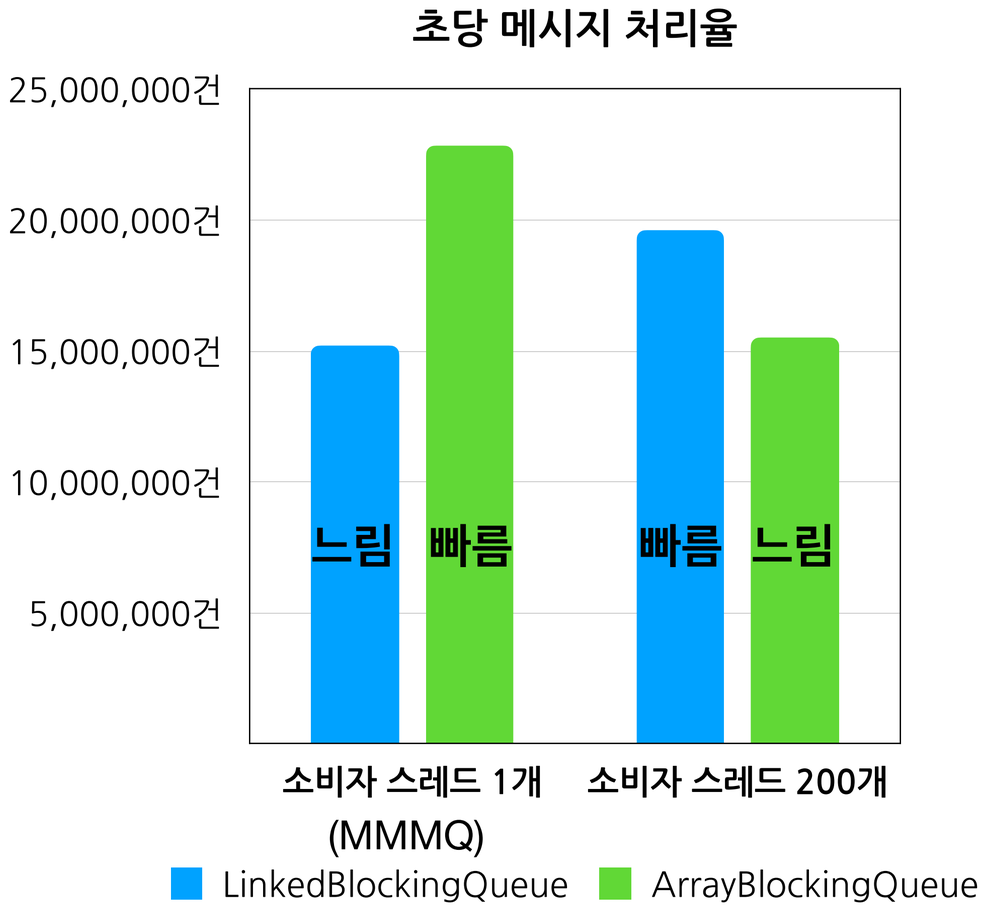
2부: 메시지 큐 자료구조

리테스트

리테스트


초당 메시지 처리율
25,000,000건
20,000,000건
15,000,000건
10,000,000건
5,000,000건
느림
빠름
빠름
느림
소비자 스레드 1개
(MMMQ)
소비자 스레드 200개
LinkedBlockingQueue
ArrayBlockingQueue

리테스트



경합이 잦을수록 LinkedBlockingQueue의 성능이 높아진다

2부: 메시지 큐 자료구조

결론

자료구조 선택은 환경에 맞게

- 메시지 큐 자료구조는 ArrayBlockingQueue로 교체
  -> 처리 성능 최적화

- DLQ 자료구조는 LinkedBlockingQueue로 유지
  -> 갑작스러운 장애에 안정성 보장

모든 상황에 정답인 자료구조는 없다.

이론만으로 성능을 예측하는 것은 위험할 수 있다.

코드 레벨 뿐만 아니라 Low-level까지 고민해보자.

감사합니다

MMMQ: https://github.com/moko-meringue/mmmq
부하테스트: https://github.com/moko-meringue/blocking-queue-analysis