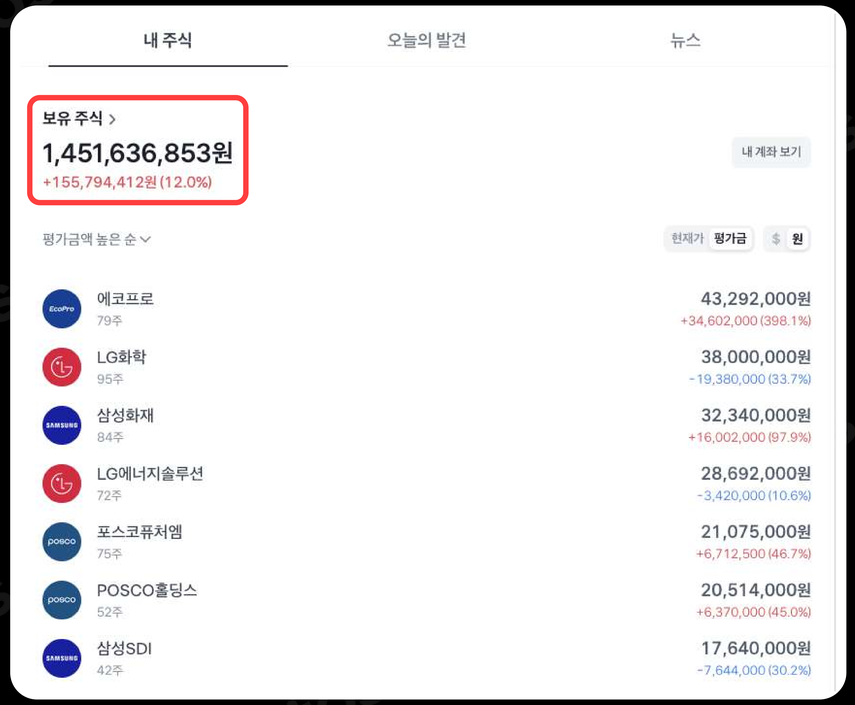
SLASH 24

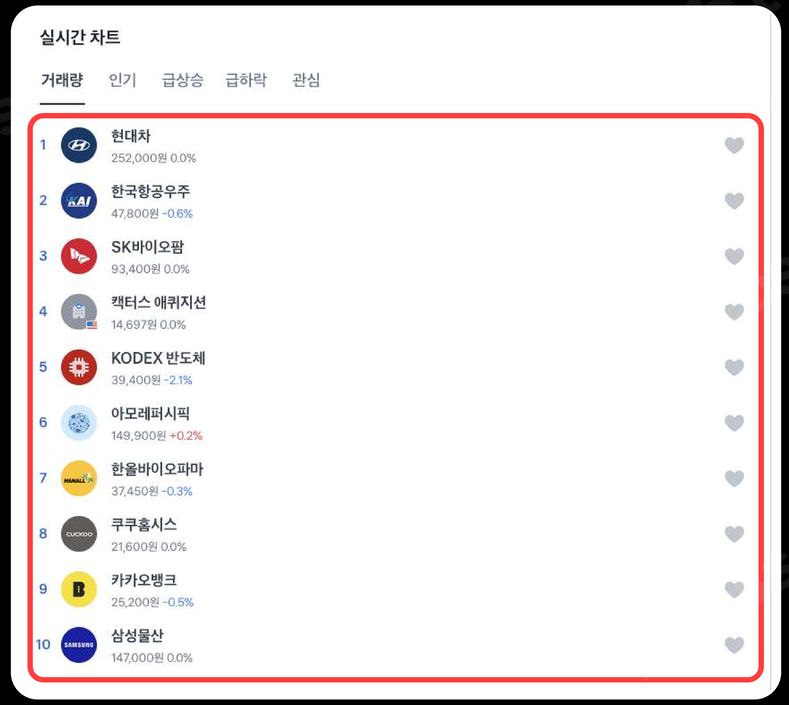
SSE 이벤트 푸쉬로 불필요한 Polling 제거하기

전연빈
Server Platform Team Leader

toss

1. 배경
2. Push server 구성
3. SSE란?
4. Client side로 message 전달 전략
5. Server side message broker 전략
6. 트러블 슈팅 및 모니터링


내 주식
오늘의 발견
뉴스
보유 주식 >
1,451,636,853원
+155,794,412원 (12.0%)
내 계좌 보기
평가금액 높은 순
현재가 평가금
$ 원
에코프로 79주 43,292,000원 +34,602,000 (398.1%)
LG화학 95주 38,000,000원 -19,380,000 (33.7%)
삼성화재 84주 32,340,000원 +16,002,000 (97.9%)
LG에너지솔루션 72주 28,692,000원 -3,420,000 (10.6%)
포스코퓨처엠 75주 21,075,000원 +6,712,500 (46.7%)
POSCO홀딩스 52주 20,514,000원 +6,370,000 (45.0%)
삼성SDI 42주 17,640,000원 -7,644,000 (30.2%)


실시간 차트
거래량
인기
급상승
급하락
관심
1 현대차 252,000원 0.0%
2 한국항공우주 47,800원 -0.6%
3 SK바이오팜 93,400원 0.0%
4 캑터스 애퀴지션 14,697원 0.0%
5 KODEX 반도체 39,400원 -2.1%
6 아모레퍼시픽 149,900원 +0.2%
7 한올바이오파마 37,450원 -0.3%
8 쿠쿠홈시스 21,600원 0.0%
9 카카오뱅크 25,200원 -0.5%
10 삼성물산 147,000원 0.0%

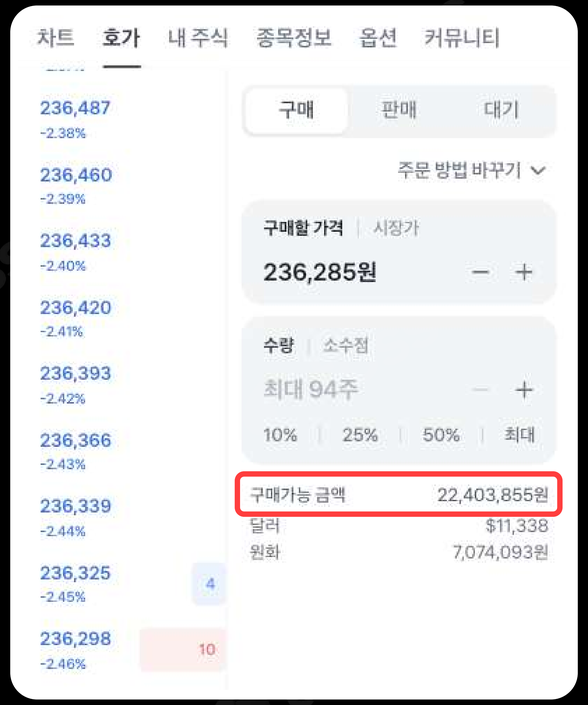
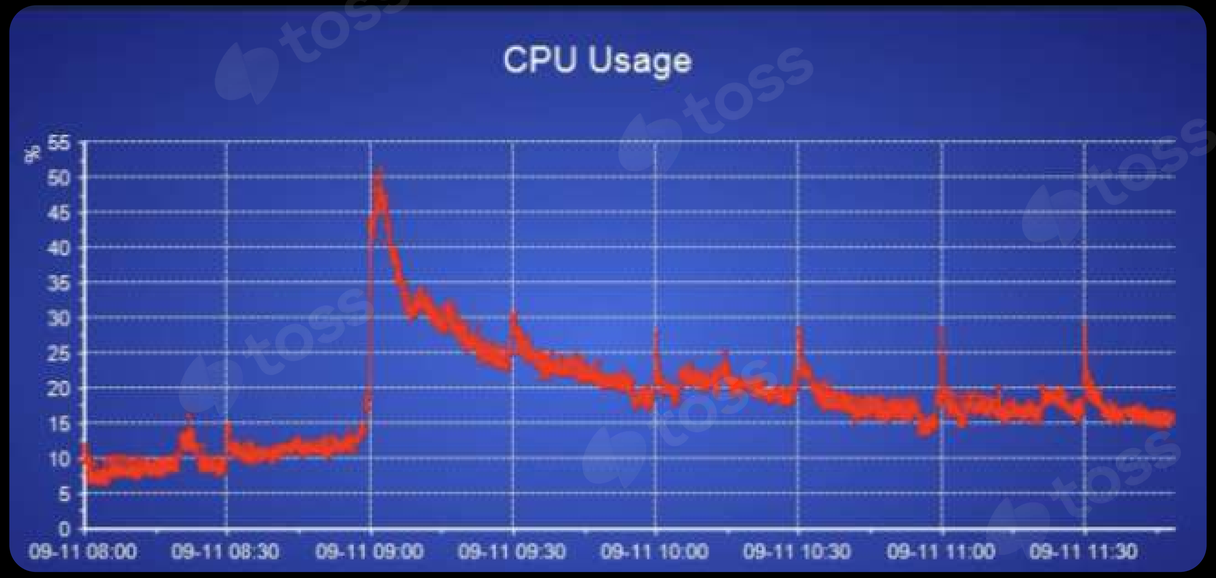
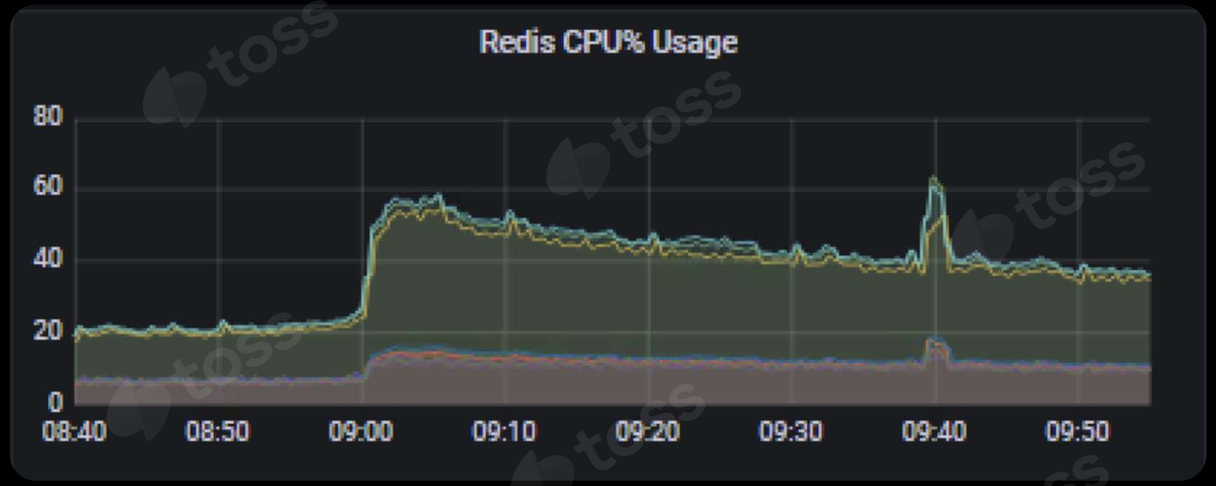
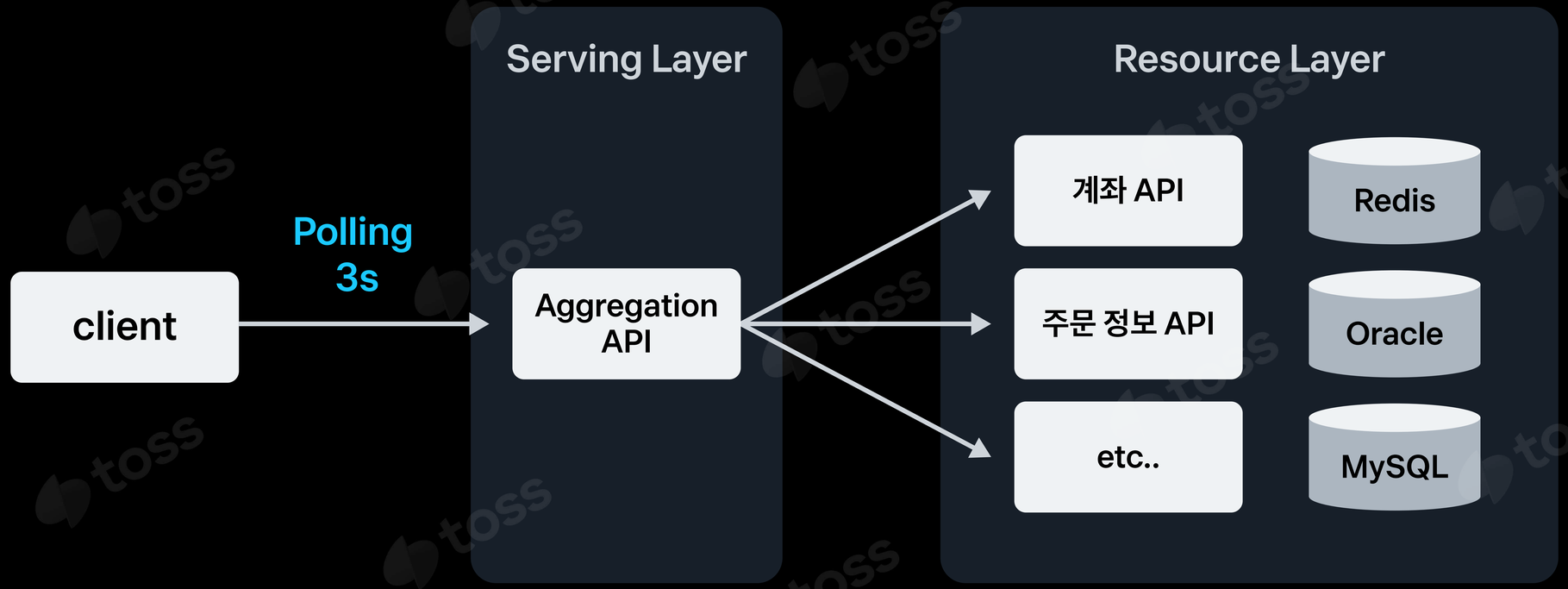
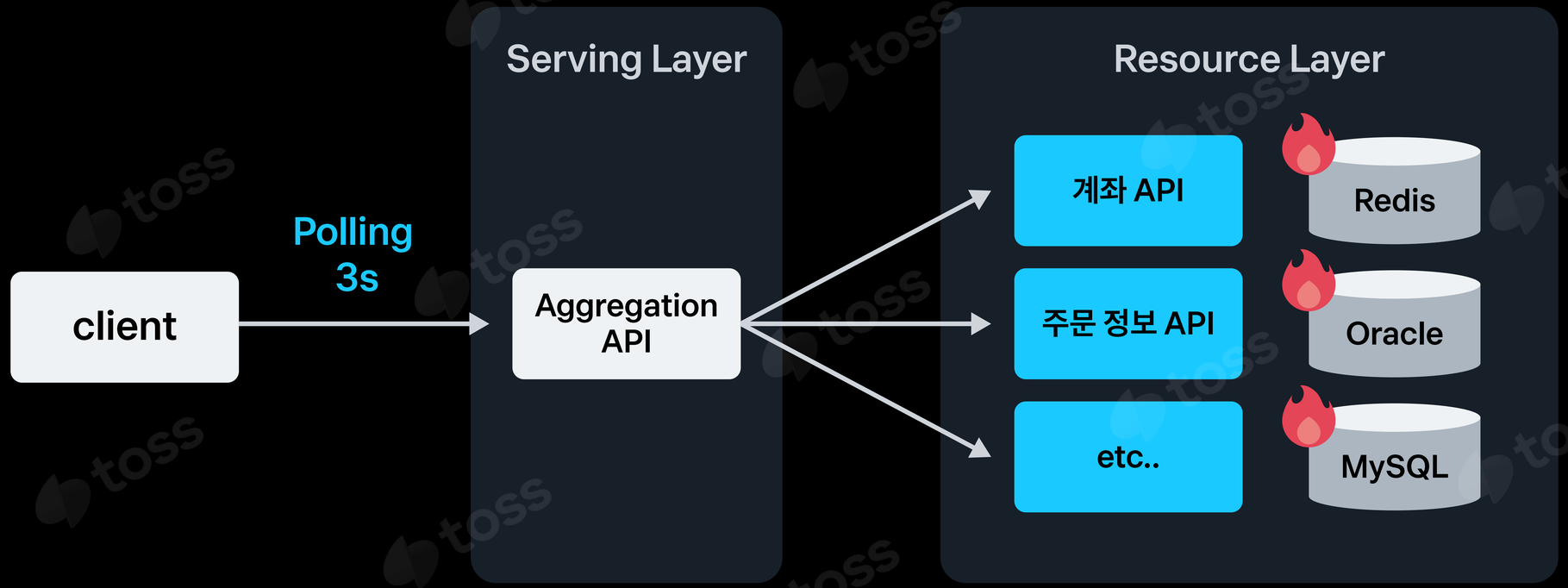


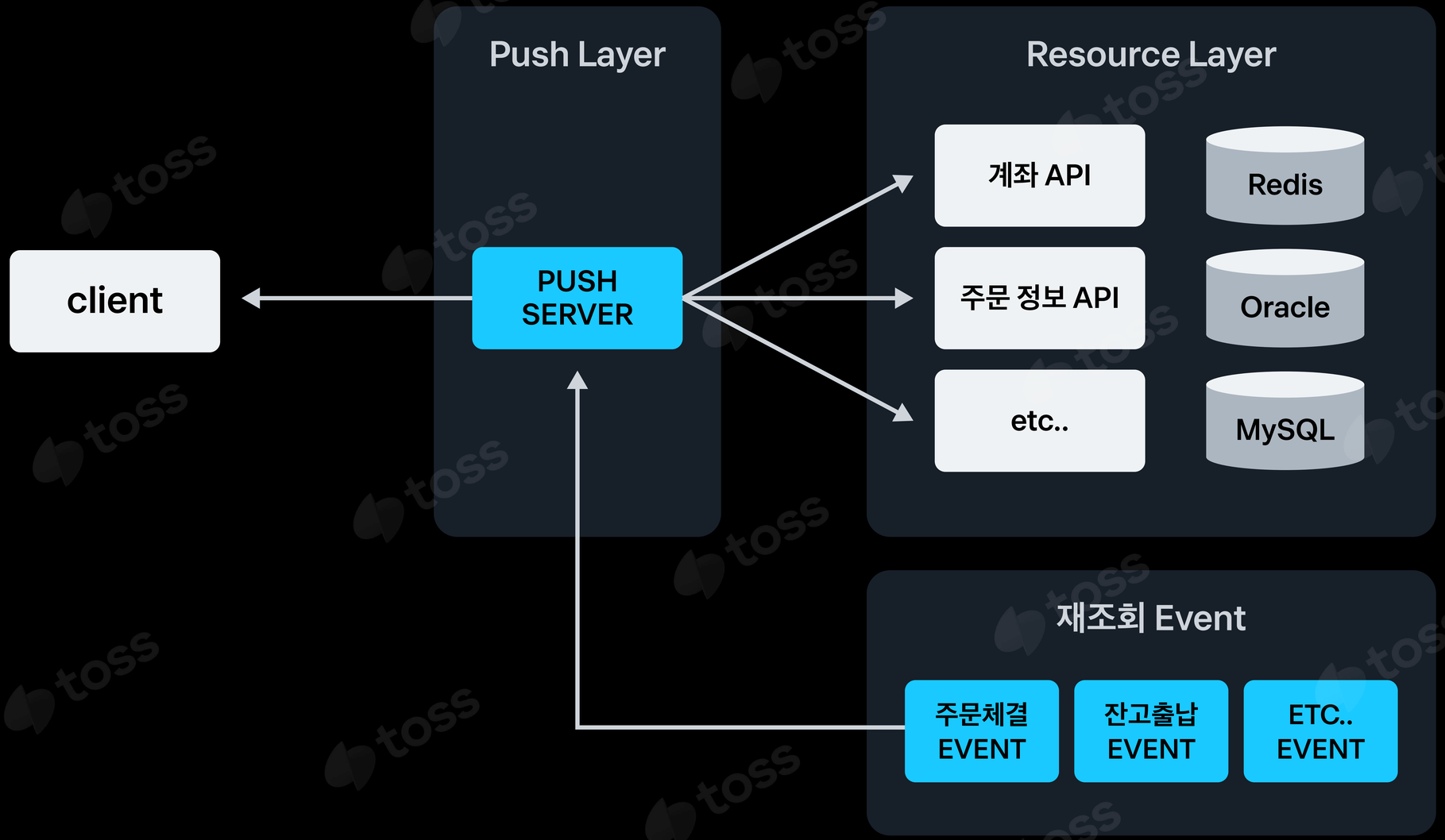
차트
호가
내 주식
종목정보
옵션
커뮤니티
236,487 -2.38%
236,460 -2.39%
236,433 -2.40%
236,420 -2.41%
236,393 -2.42%
236,366 -2.43%
236,339 -2.44%
236,325 -2.45%
4
236,298 -2.46%
10
구매
판매
대기
주문 방법 바꾸기
구매할 가격 | 시장가
236,285원
수량 | 소수점
최대 94주
10% | 25% | 50% | 최대
구매가능 금액 22,403,855원
달러 $11,338
원화 7,074,093원


CPU Usage
%
55
50
45
40
35
30
25
20
15
10
5
0
09-11 08:00
09-11 08:30
09-11 09:00
09-11 09:30
09-11 10:00
09-11 10:30
09-11 11:00
09-11 11:30


Redis CPU% Usage
80
60
40
20
0
08:40
08:50
09:00
09:10
09:20
09:30
09:40
09:50


Serving Layer
Resource Layer
client
Polling
3s
Aggregation
API
계좌 API
주문 정보 API
etc..
Redis
Oracle
MySQL


Serving Layer
Resource Layer
client
Polling
3s
Aggregation
API
계좌 API
주문 정보 API
etc..
Redis
Oracle
MySQL


Push Layer
Resource Layer
client
PUSH SERVER
계좌 API
주문 정보 API
etc..
Redis
Oracle
MySQL
재조회 Event
주문체결 EVENT
잔고출납 EVENT
ETC.. EVENT

PUSH SERVER 구성

WebSocket

ServerSentEvents

	SSE	WebSocket
Protocol	HTTP	Websocket
Direction	서버사이드 단방향	양방향
Auto Reconnect	Yes	No
Cookie	Yes	No
Response Type	Text	text or byte

그래서 증권에서 사용하는 방식은?

WebSocket

양방향 통신이 필요함

송 / 수신량 데이터량이 많을때 사용

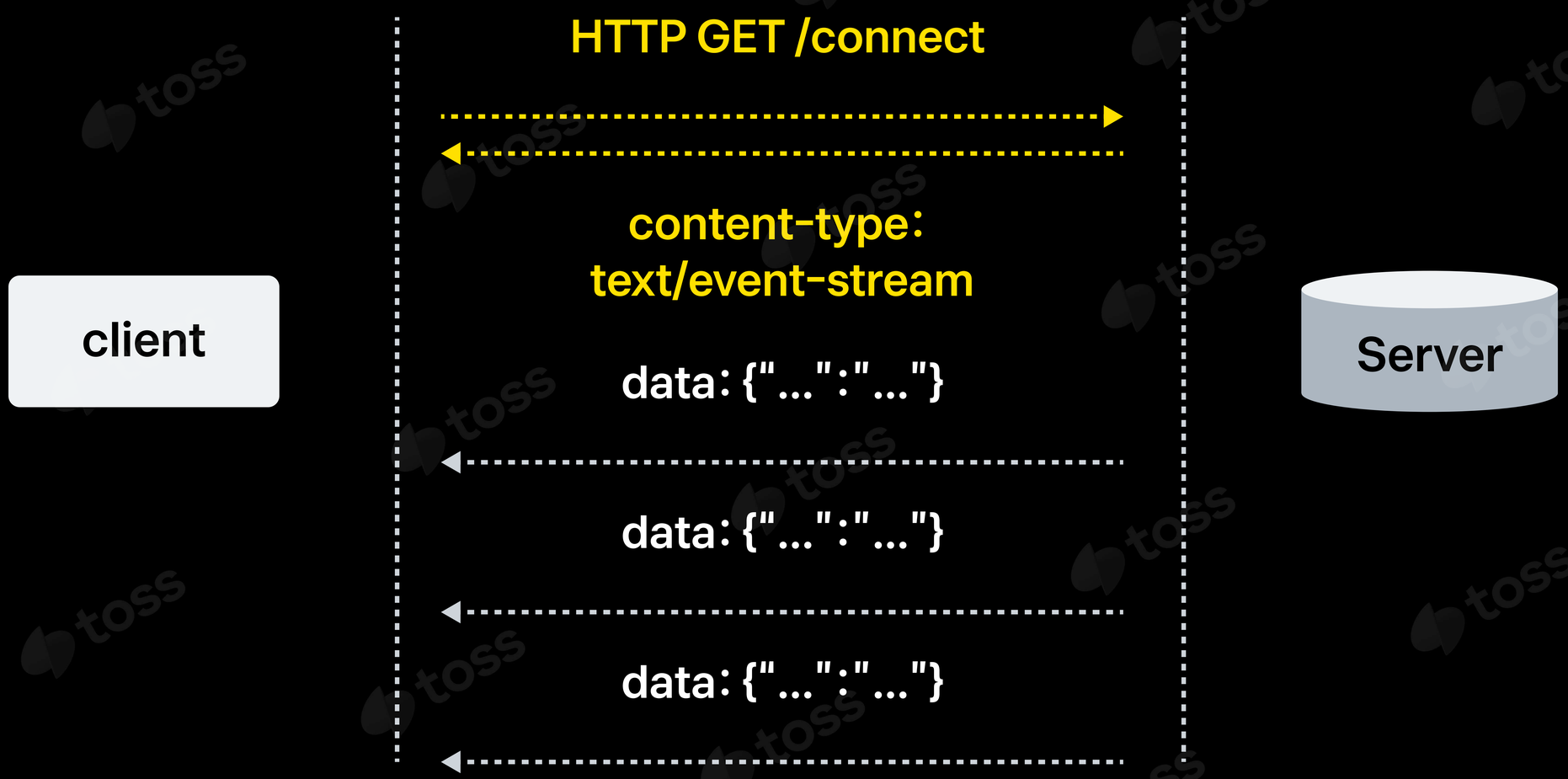
시세 위주 사용

ServerSentEvents

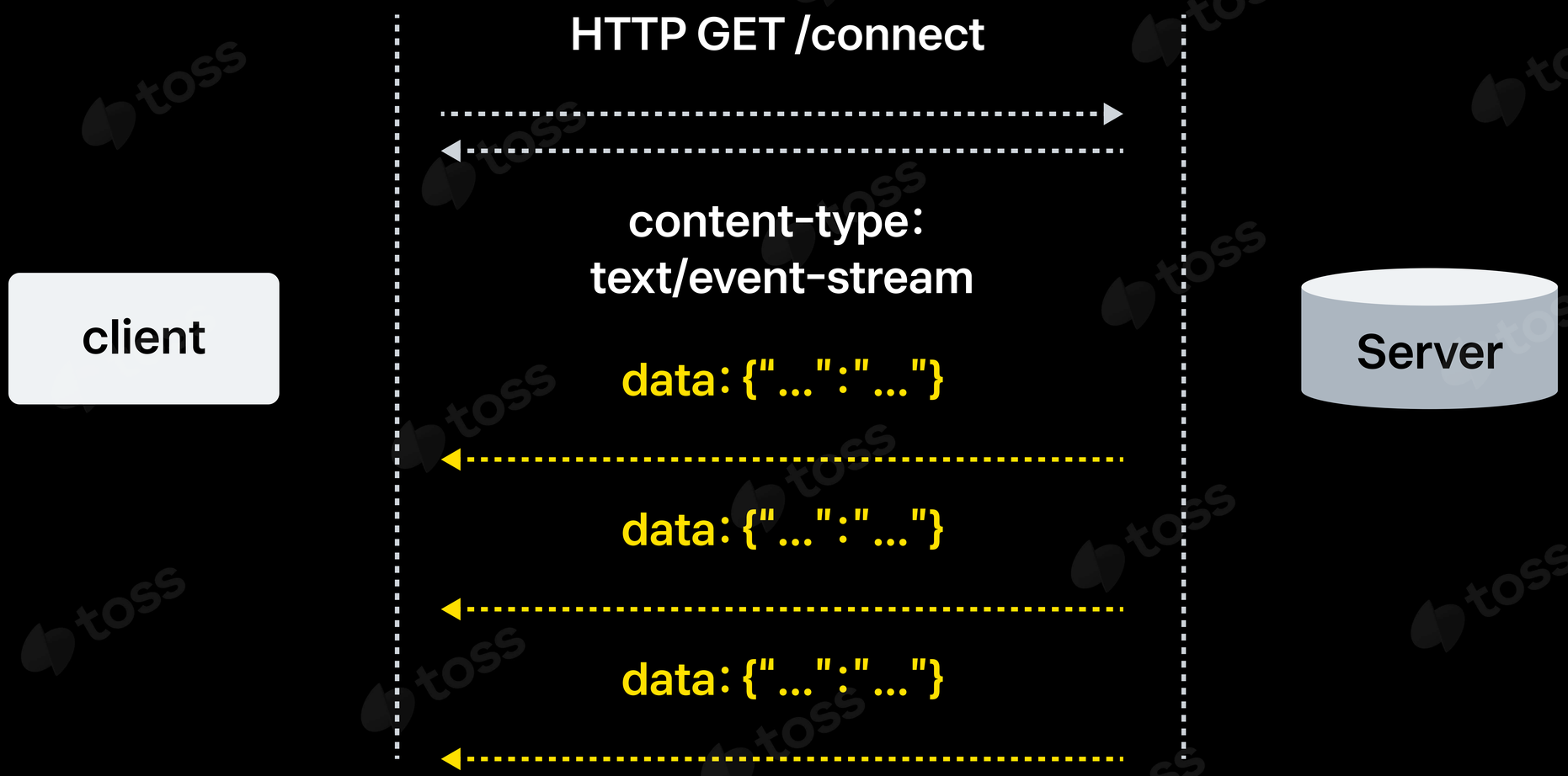
불필요한 polling 제거할때

개인화된 데이터를 이용한 이벤트 푸쉬

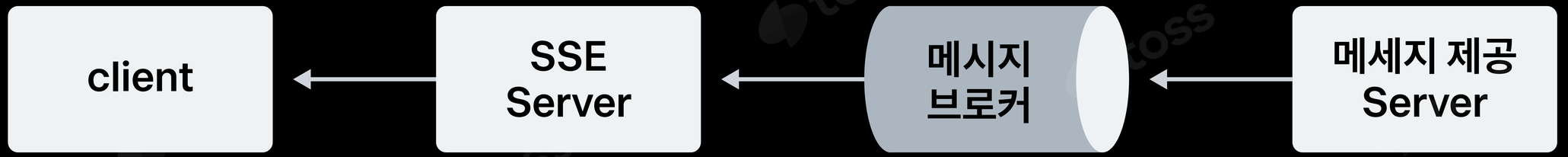
SSE


HTTP GET /connect
content-type:
text/event-stream
client
Server
data: {"..." : "..."}
data: {"..." : "..."}
data: {"..." : "..."}

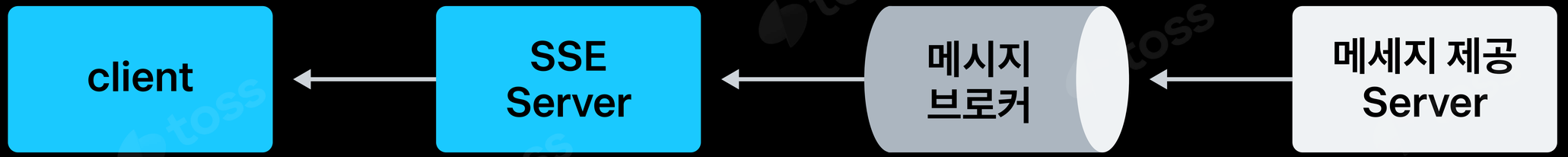
SSE


HTTP GET /connect
content-type:
text/event-stream
client
Server
data: {"..." : "..."}
data: {"..." : "..."}
data: {"..." : "..."}

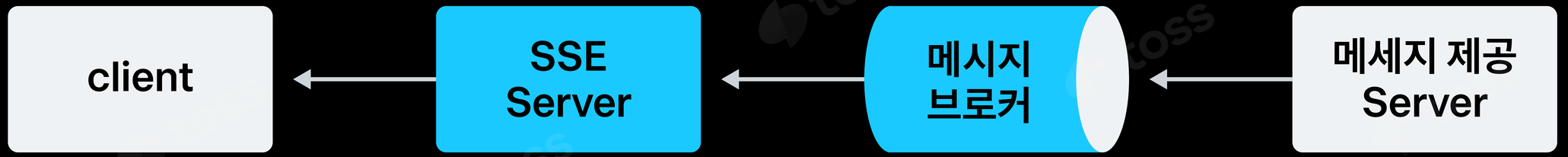
SSE



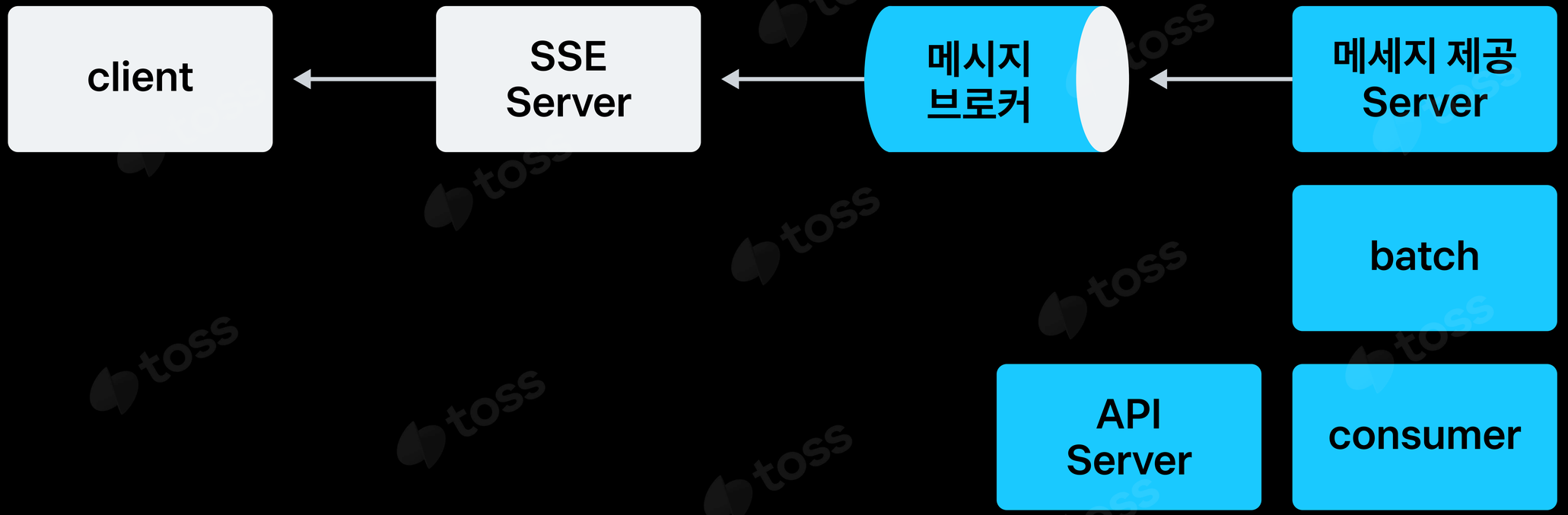
SSE



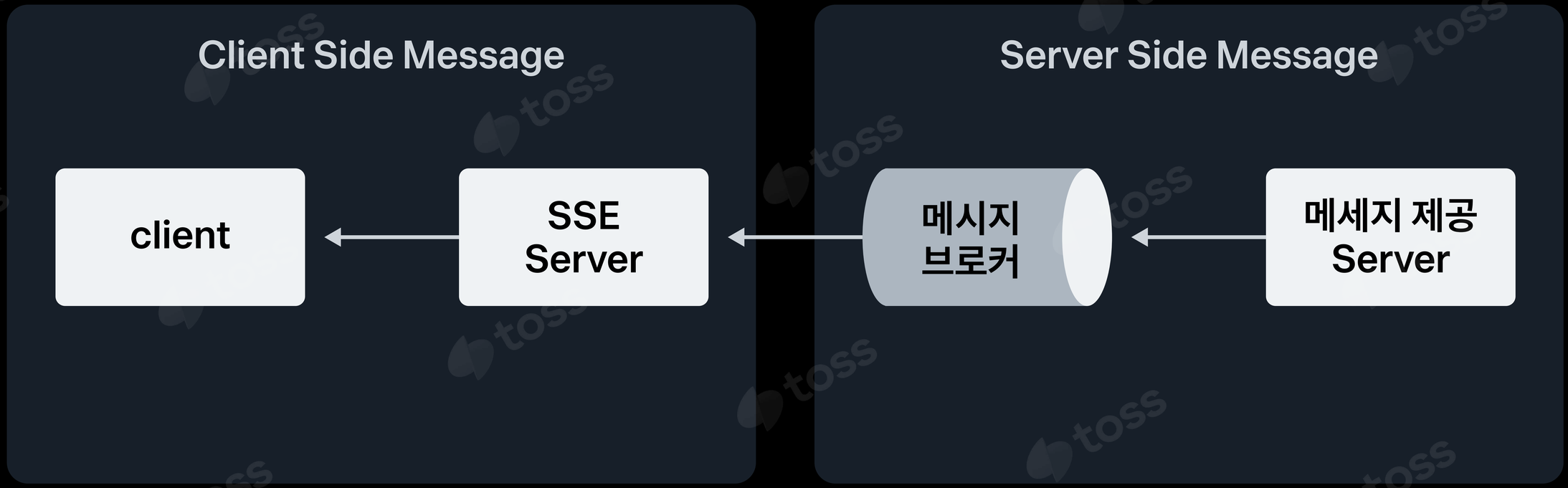
SSE



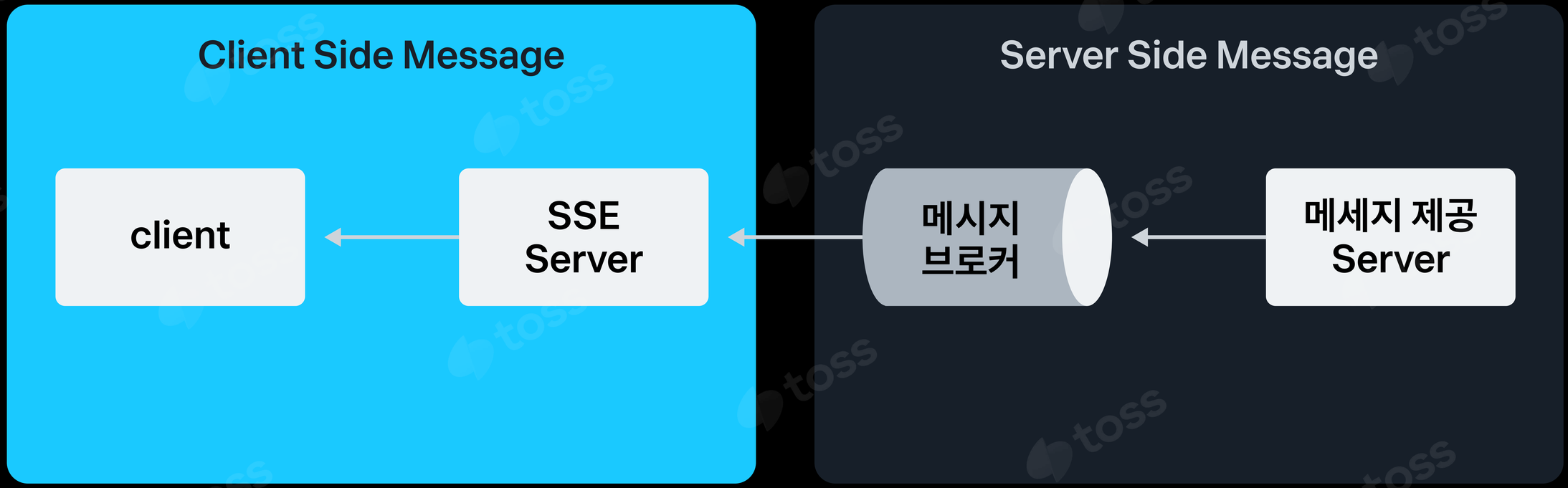
SSE


client
SSE
Server
메시지
브로커
메세지 제공
Server
batch
API
Server
consumer

SSE


Client Side Message
Server Side Message
client
SSE
Server
메시지
브로커
메세지 제공
Server

SSE

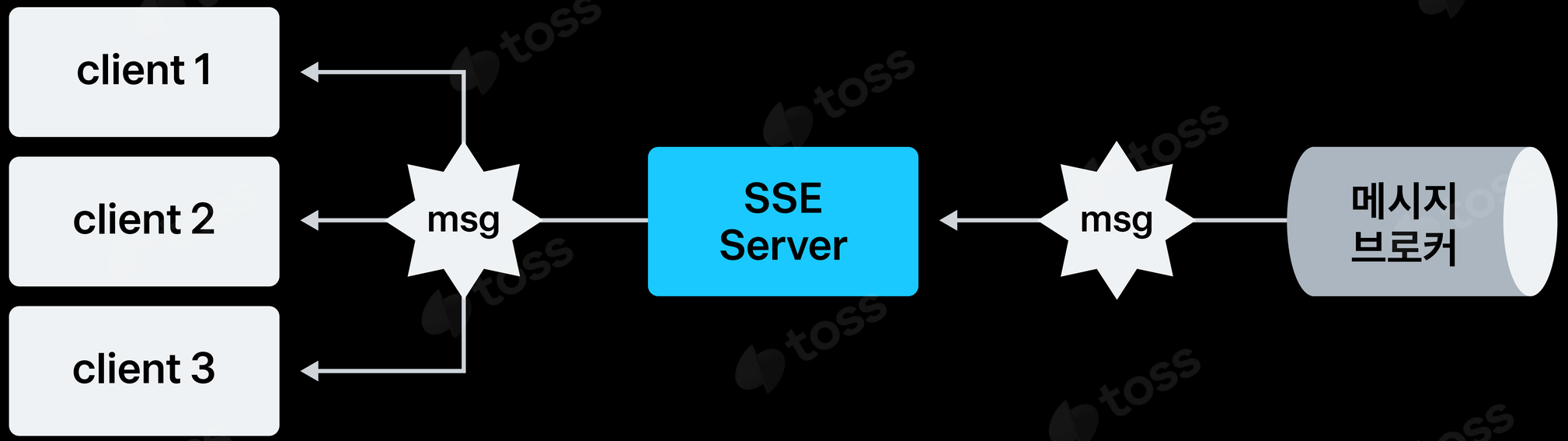

Client Side Message
Server Side Message
client
SSE
Server
메시지
브로커
메세지 제공
Server

Client side message 전략

BroadCasting

Unicasting

broadcasting



```
private val channel: Sinks.Many<ServerSentEvent<T>> = Sinks.many().multicast().directAllOrNothing()

override val doProcess: Consumer<ReactiveSubscription.Message<String, String>>
    get() = Consumer { msg ->
        val sseMsg = JsonUtil.fromJson(msg.message, object: TypeReference<SseMessage<T>>() {})
        channelMeterRegistry.recordMessageSub(sseMsg.header.sentTime)

        // send to client
        super.send(
            channel,
            ServerSentEvent.builder<T>()
                .data(sseMsg.msg)
                .id(msg.channel)
                .build()
        )
    }

override fun afterPropertiesSet() {
    // subscribe inbound message
    super.onMessage(CHANNEL_NAME)
        .subscribe()
}

fun connect(): Flux<ServerSentEvent<T>> {
    return Flux.merge(
        super.connect(channel) {},
    )
}
```

```
private val channel: Sinks.Many<ServerSentEvent<T>> = Sinks.many().multicast().directAllOrNothing()

override val doProcess: Consumer<ReactiveSubscription.Message<String, String>>
    get() = Consumer { msg ->
        val sseMsg = JsonUtil.fromJson(msg.message, object: TypeReference<SseMessage<T>>() {})
        channelMeterRegistry.recordMessageSub(sseMsg.header.sentTime)

        // send to client
        super.send(
            channel,
            ServerSentEvent.builder<T>()
                .data(sseMsg.msg)
                .id(msg.channel)
                .build()
        )
    }

override fun afterPropertiesSet() {
    // subscribe inbound message
    super.onMessage(CHANNEL_NAME)
        .subscribe()
}

fun connect(): Flux<ServerSentEvent<T>> {
    return Flux.merge(
        super.connect(channel) {},
    )
}
```

```
private val channel: Sinks.Many<ServerSentEvent<T>> = Sinks.many().multicast().directAllOrNothing()

override val doProcess: Consumer<ReactiveSubscription.Message<String, String>>
    get() = Consumer { msg ->
        val sseMsg = JsonUtil.fromJson(msg.message, object: TypeReference<SseMessage<T>>() {})
        channelMeterRegistry.recordMessageSub(sseMsg.header.sentTime)

        // send to client
        super.send(
            channel,
            ServerSentEvent.builder<T>()
                .data(sseMsg.msg)
                .id(msg.channel)
                .build()
        )
    }

override fun afterPropertiesSet() {
    // subscribe inbound message
    super.onMessage(CHANNEL_NAME)
        .subscribe()
}

fun connect(): Flux<ServerSentEvent<T>> {
    return Flux.merge(
        super.connect(channel) {},
    )
}
```

```
private val channel: Sinks.Many<ServerSentEvent<T>> = Sinks.many().multicast().directAllOrNothing()

override val doProcess: Consumer<ReactiveSubscription.Message<String, String>>
    get() = Consumer { msg ->
        val sseMsg = JsonUtil.fromJson(msg.message, object: TypeReference<SseMessage<T>>() {})
        channelMeterRegistry.recordMessageSub(sseMsg.header.sentTime)

        // send to client
        super.send(
            channel,
            ServerSentEvent.builder<T>()
                .data(sseMsg.msg)
                .id(msg.channel)
                .build()
        )
    }

override fun afterPropertiesSet() {
    // subscribe inbound message
    super.onMessage(CHANNEL_NAME)
        .subscribe()
}

fun connect(): Flux<ServerSentEvent<T>> {
    return Flux.merge(
        super.connect(channel) {},
    )
}
```

```kotlin
@RestApiController
class BroadcastEventController(
    @Qualifier(ChannelConfiguration.BROADCAST_CHANNEL)
    private val broadcastChannelHandler: BroadcastChannelHandler<String>
) {
    @GetMapping(value = ["/api/v1/live-chat"], produces = [MediaType.TEXT_EVENT_STREAM_VALUE])
    fun eventConnect(): Flux<ServerSentEvent<String>> {
        return broadcastChannelHandler.connect()
    }
```

broadcasting 실제 사용 사례

이벤트성 단일 채팅방

거래량 1 TQQQ +3.9%

실시간 의견

31.6만명 참여

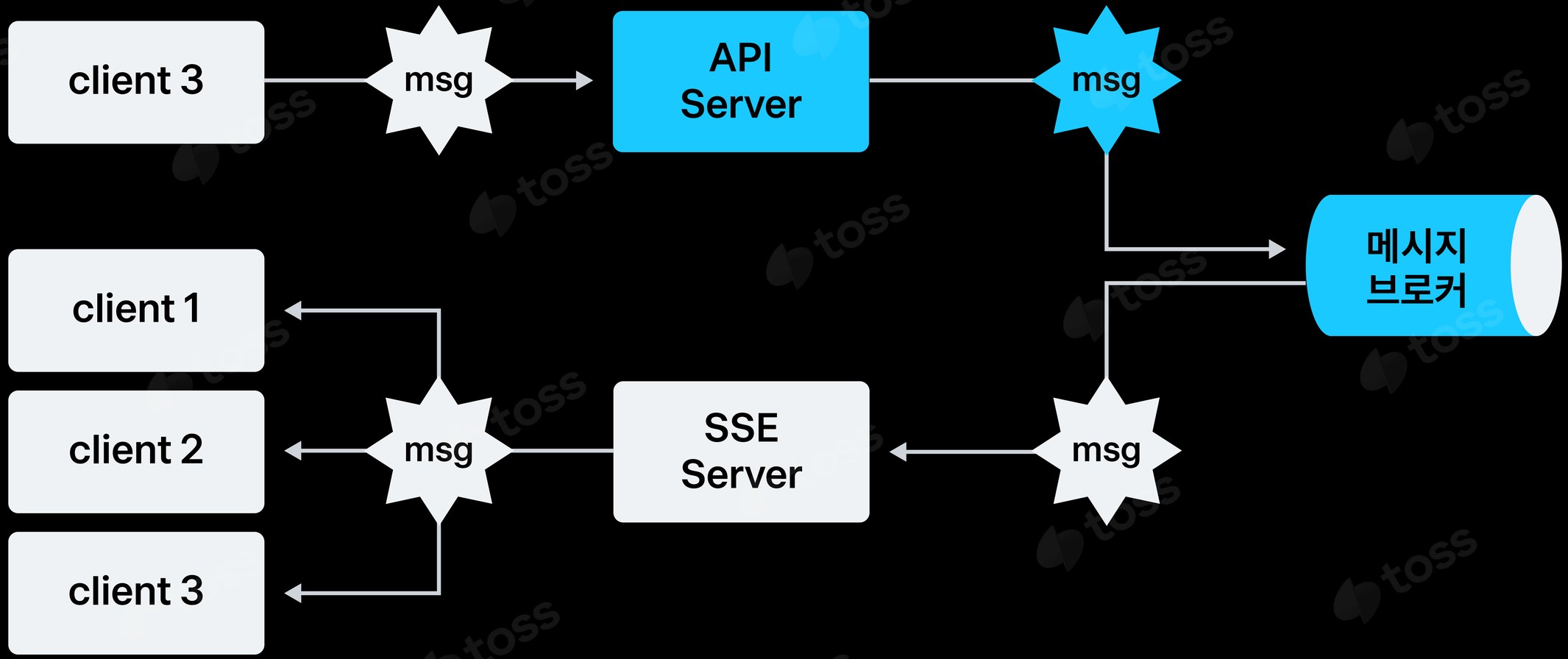
빠지면더진다
인상하는 건 사실상 예견된 일이었고, 중요한 건
앞으로 금리를 어떻게 하느냐인데...

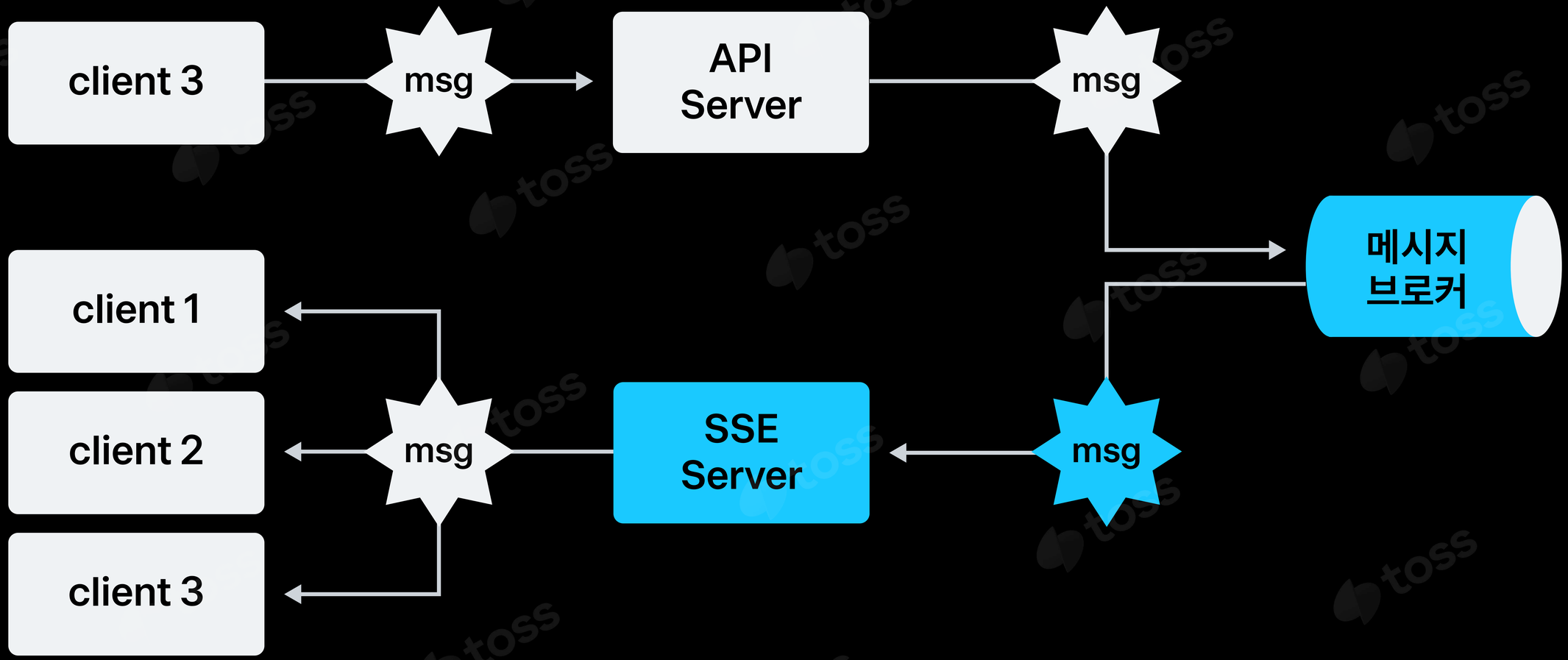
연남동서뛰지마라
매번 느끼는거지만, 파월 형님 수트빨 잘 받네ㅋㅋㅋ

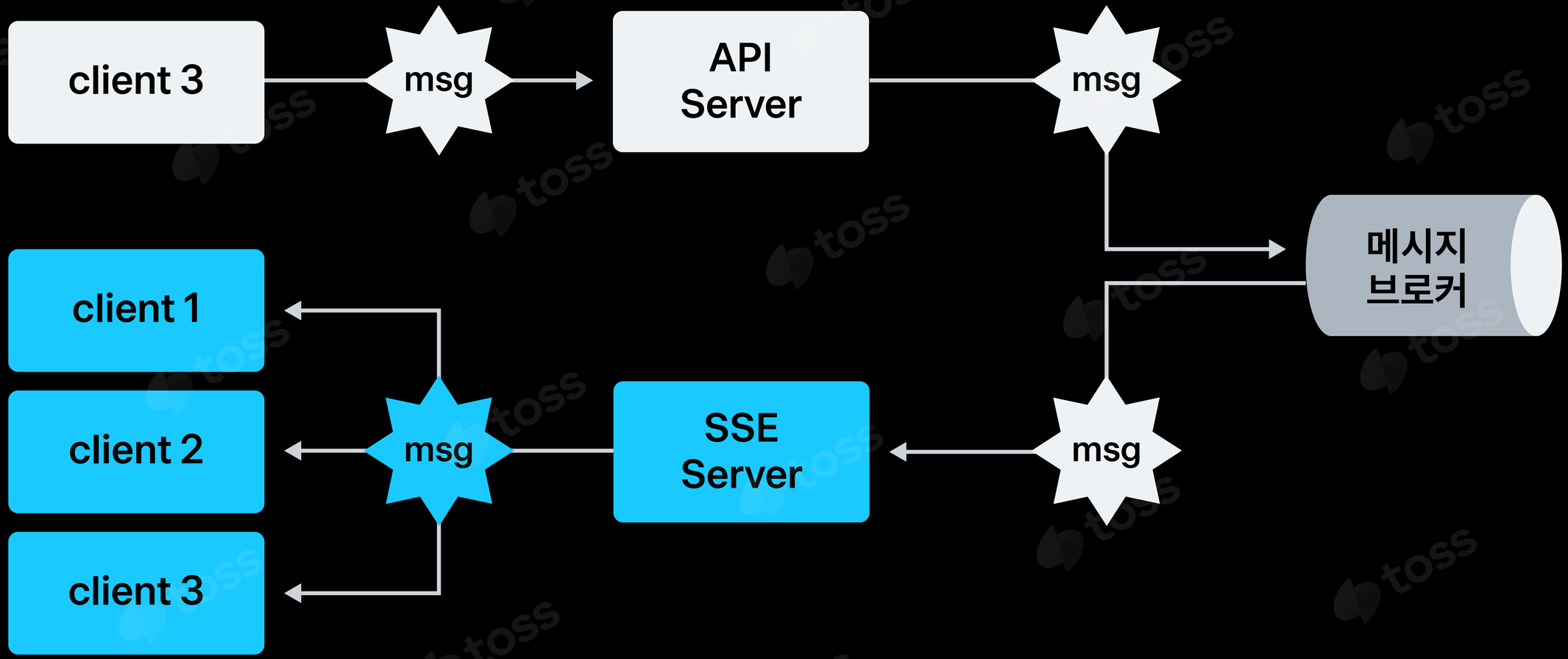
충성충성
TQQQ 0.042892주 구매
이렇게 좋은날 안 사고 못 버티지~~~ 가즈아!!

뚱인데요
그래서 오늘 뭐사면 돼요??? TQ? SQ?

의견 남기기


client 3
msg
API Server
msg
메시지 브로커
client 1
client 2
client 3
msg
SSE Server
msg

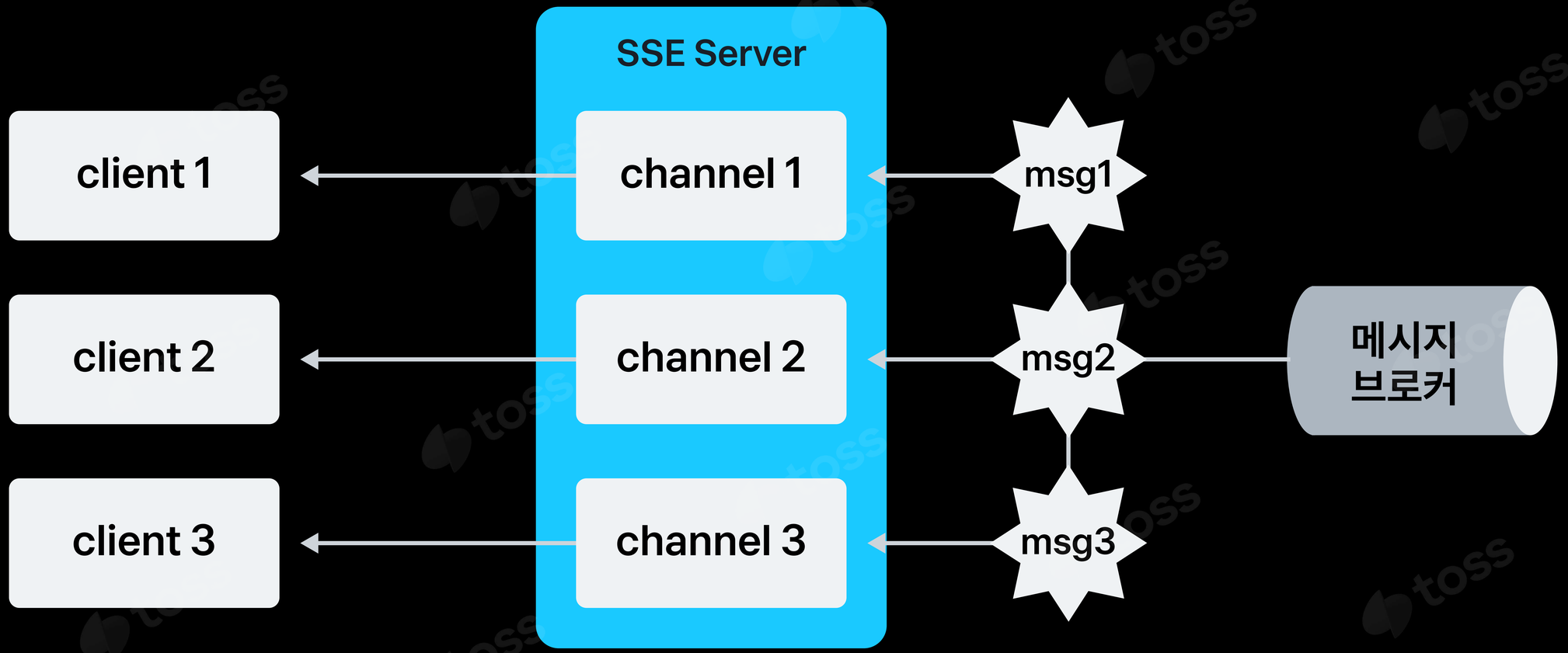

client 3
msg
API Server
msg
메시지 브로커
client 1
client 2
client 3
msg
SSE Server
msg


client 3
msg
API Server
msg
메시지 브로커
client 1
client 2
client 3
msg
SSE Server
msg

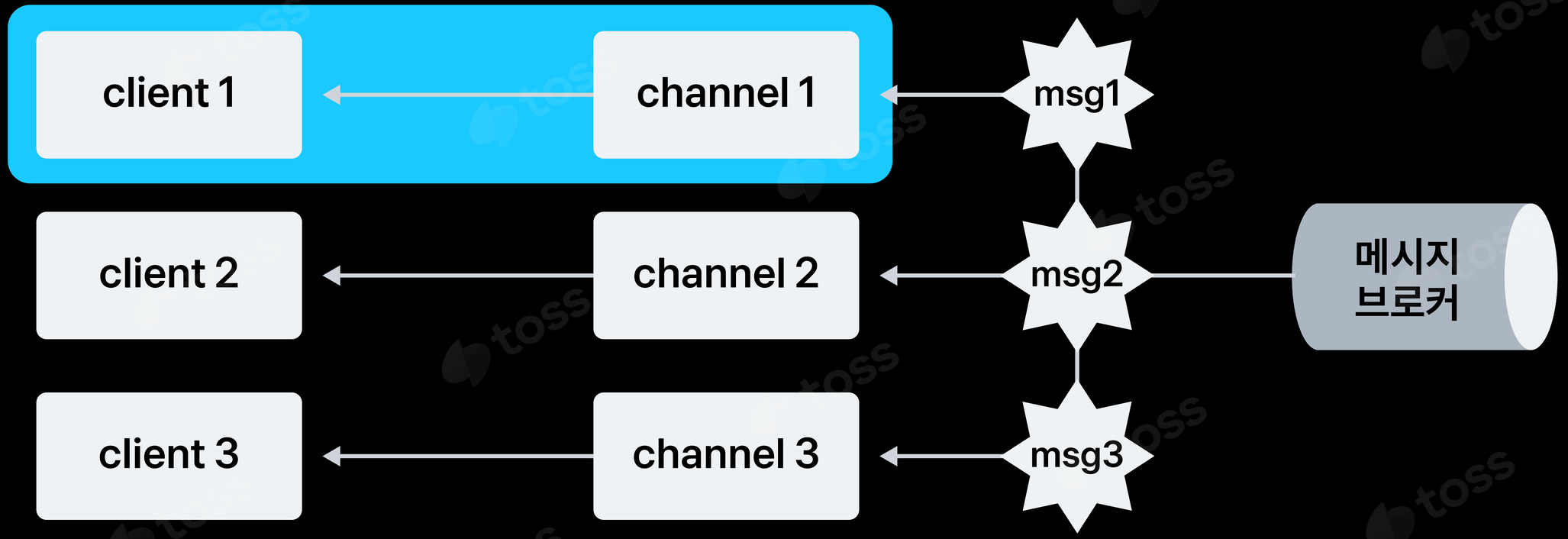
하지만 유저별은 어떻게..?

개인화된 데이터를 이용한 이벤트 푸쉬는..?

unicasting


SSE Server
client 1
client 2
client 3
channel 1
channel 2
channel 3
msg1
msg2
msg3
메시지
브로커

unicasting


client 1
channel 1
msg1
client 2
channel 2
msg2
메시지
브로커
client 3
channel 3
msg3

```
class UnicastChannelHandler<T : Any>(
    private val channelMeterRegistry: ChannelMeterRegistry,
    private val inboundStream: Flux<String>,
) : ChannelOutBoundHandler<ServerSentEvent<T>> {

    private val CHANNEL = Sinks.many().unicast().onBackpressureBuffer<ServerSentEvent<T>>()
    private val TYPE_REFERENCE = object : TypeReference<SseMessage<T>>() {}

    fun connect(): Flux<ServerSentEvent<T>> {
        return Flux.merge(
            // channel connect
            super.connect(CHANNEL) {
            },

            inboundStream
                .map { JsonUtil.fromJson(it, TYPE_REFERENCE) }
                .map { sseMsg ->
                    channelMeterRegistry.recordMessageSub(sseMsg.header.sentTime)
                    ServerSentEvent.builder<T>()
                        .data(sseMsg.msg)
                        .id(sseMsg.header.key)
                        .build()
                }
        )
    }
}
```

```kotlin
class UnicastChannelHandler<T : Any>(
    private val channelMeterRegistry: ChannelMeterRegistry,
    private val inboundStream: Flux<String>,
) : ChannelOutBoundHandler<ServerSentEvent<T>> {

    private val CHANNEL = Sinks.many().unicast().onBackpressureBuffer<ServerSentEvent<T>>()
    private val TYPE_REFERENCE = object : TypeReference<SseMessage<T>>() {}

    fun connect(): Flux<ServerSentEvent<T>> {
        return Flux.merge(
            // channel connect
            super.connect(CHANNEL) {
            },

            inboundStream
                .map { JsonUtil.fromJson(it, TYPE_REFERENCE) }
                .map { sseMsg ->
                    channelMeterRegistry.recordMessageSub(sseMsg.header.sentTime)
                    ServerSentEvent.builder<T>()
                        .data(sseMsg.msg)
                        .id(sseMsg.header.key)
                        .build()
                }
        )
    }
}
```

```kotlin
class UnicastChannelHandler<T : Any>(
    private val channelMeterRegistry: ChannelMeterRegistry,
    private val inboundStream: Flux<String>,
) : ChannelOutBoundHandler<ServerSentEvent<T>> {

    private val CHANNEL = Sinks.many().unicast().onBackpressureBuffer<ServerSentEvent<T>>()
    private val TYPE_REFERENCE = object : TypeReference<SseMessage<T>>() {}

    fun connect(): Flux<ServerSentEvent<T>> {
        return Flux.merge(
            // channel connect
            super.connect(CHANNEL) {
            },

            inboundStream
                .map { JsonUtil.fromJson(it, TYPE_REFERENCE) }
                .map { sseMsg ->
                    channelMeterRegistry.recordMessageSub(sseMsg.header.sentTime)
                    ServerSentEvent.builder<T>()
                        .data(sseMsg.msg)
                        .id(sseMsg.header.key)
                        .build()
                }
        )
    }
}
```

개인용 data stream은 어떻게 만들지?

```kotlin
fun createInboundStreamFromRedisPubSub(
    connection: StatefulRedisClusterPubSubConnection<String, String>,
    channel: String
): Flux<String> {
    return Flux.create { sink: FluxSink<String> ->
        val pubSubListener = object : RedisPubSubAdapter<String, String>() {
            override fun message(messageChannel: String, message: String?) {
                if (channel == messageChannel && message != null) {
                    sink.next(message)
                }
            }
        }

        val (_, subscriber) =
            shardConnection(connection.async().upstream(), channel)
                ?: return@create sink.error(UnreachableCodeException("redis connection error"))

        subscriber.statefulConnection.addListener(pubSubListener)
        subscriber.subscribe(channel)

        sink.onDispose {
            subscriber.unsubscribe(channel)
            subscriber.statefulConnection.removeListener(pubSubListener)
        }
    }
}
```

```kotlin
fun createInboundStreamFromRedisPubSub(
    connection: StatefulRedisClusterPubSubConnection<String, String>,
    channel: String
): Flux<String> {
    return Flux.create { sink: FluxSink<String> ->
        val pubSubListener = object : RedisPubSubAdapter<String, String>() {
            override fun message(messageChannel: String, message: String?) {
                if (channel == messageChannel && message != null) {
                    sink.next(message)
                }
            }
        }

        val (_, subscriber) =
            shardConnection(connection.async().upstream(), channel)
                ?: return@create sink.error(UnreachableCodeException("redis connection error"))

        subscriber.statefulConnection.addListener(pubSubListener)
        subscriber.subscribe(channel)

        sink.onDispose {
            subscriber.unsubscribe(channel)
            subscriber.statefulConnection.removeListener(pubSubListener)
        }
    }
}
```

```kotlin
fun createInboundStreamFromRedisPubSub(
    connection: StatefulRedisClusterPubSubConnection<String, String>,
    channel: String
): Flux<String> {
    return Flux.create { sink: FluxSink<String> ->
        val pubSubListener = object : RedisPubSubAdapter<String, String>() {
            override fun message(messageChannel: String, message: String?) {
                if (channel == messageChannel && message != null) {
                    sink.next(message)
                }
            }
        }

        val (_, subscriber) =
            shardConnection(connection.async().upstream(), channel)
                ?: return@create sink.error(UnreachableCodeException("redis connection error"))

        subscriber.statefulConnection.addListener(pubSubListener)
        subscriber.subscribe(channel)

        sink.onDispose {
            subscriber.unsubscribe(channel)
            subscriber.statefulConnection.removeListener(pubSubListener)
        }
    }
}
```

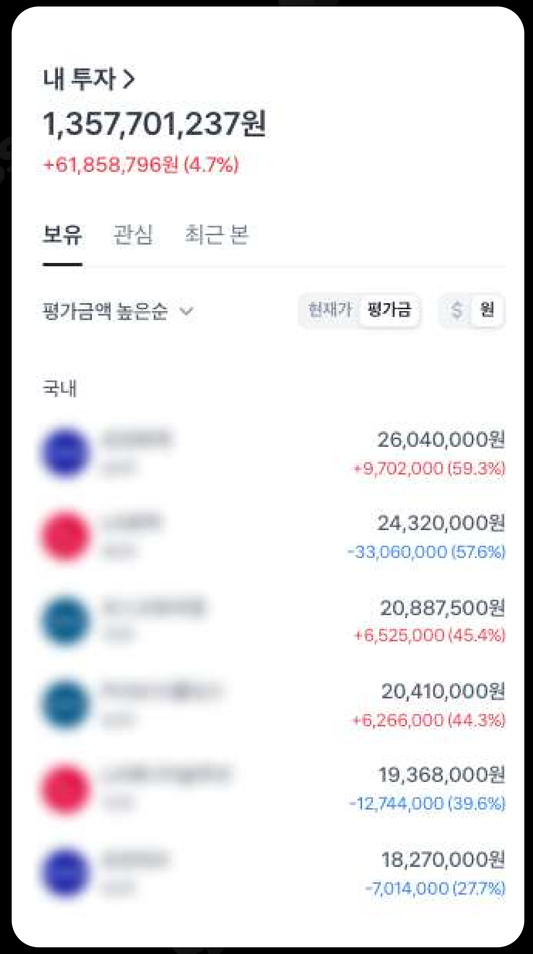
```kotlin
fun createInboundStreamFromRedisPubSub(
    connection: StatefulRedisClusterPubSubConnection<String, String>,
    channel: String
): Flux<String> {
    return Flux.create { sink: FluxSink<String> ->
        val pubSubListener = object : RedisPubSubAdapter<String, String>() {
            override fun message(messageChannel: String, message: String?) {
                if (channel == messageChannel && message != null) {
                    sink.next(message)
                }
            }
        }

        val (_, subscriber) =
            shardConnection(connection.async().upstream(), channel)
                ?: return@create sink.error(UnreachableCodeException("redis connection error"))

        subscriber.statefulConnection.addListener(pubSubListener)
        subscriber.subscribe(channel)

        sink.onDispose {
            subscriber.unsubscribe(channel)
            subscriber.statefulConnection.removeListener(pubSubListener)
        }
    }
}
```

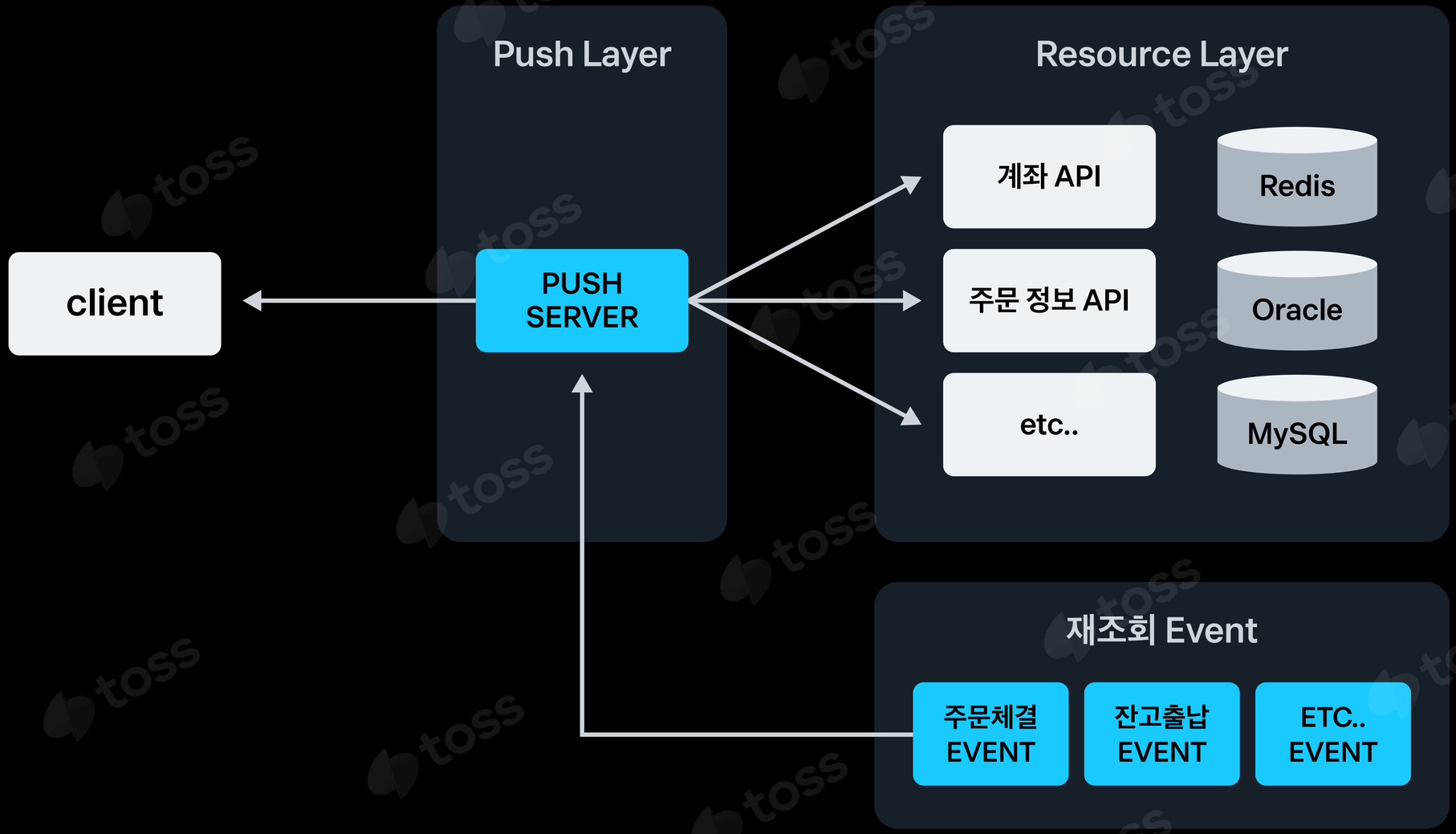
unicasting 실제 사용 사례

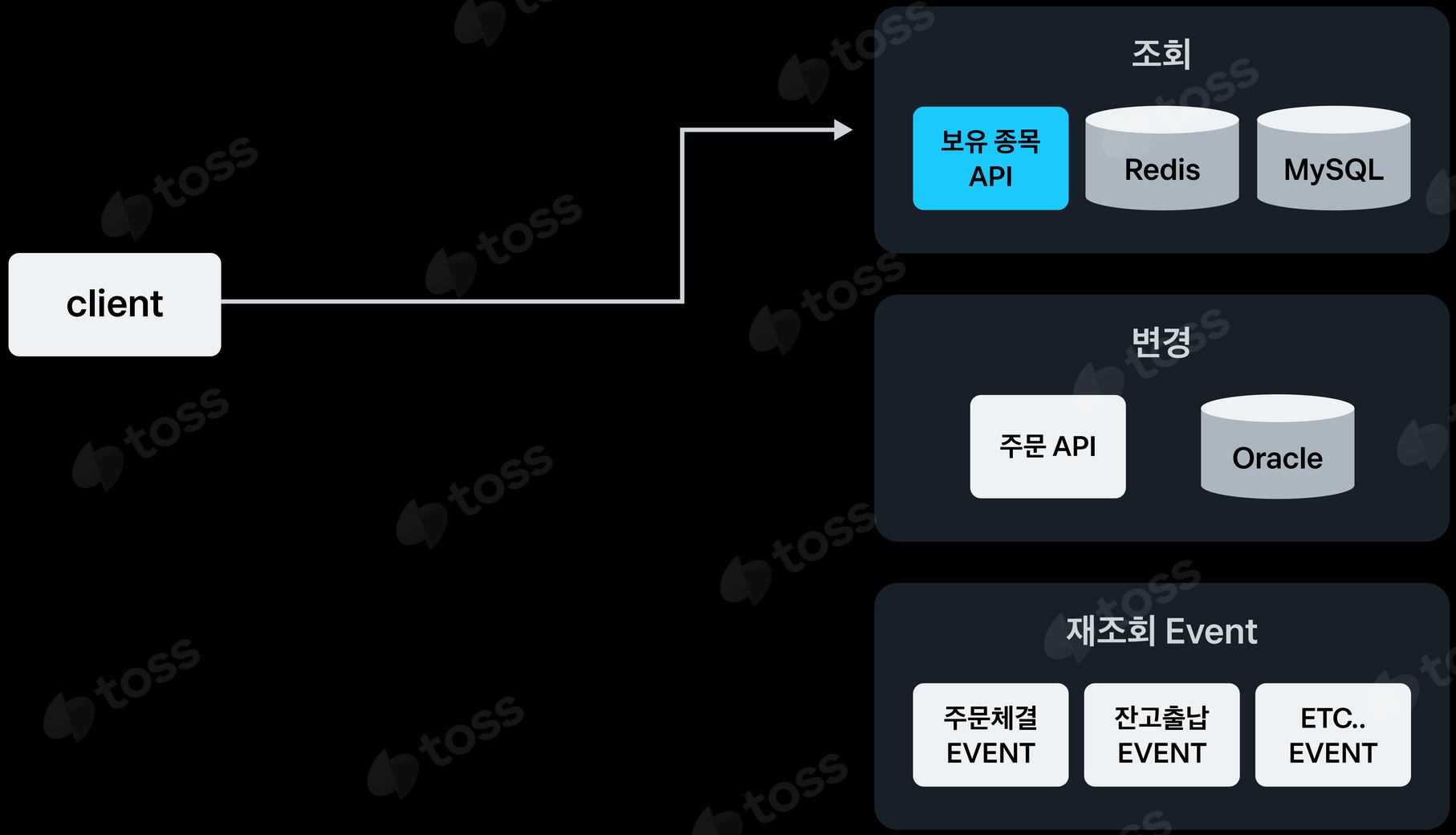
보유자산 polling 제거

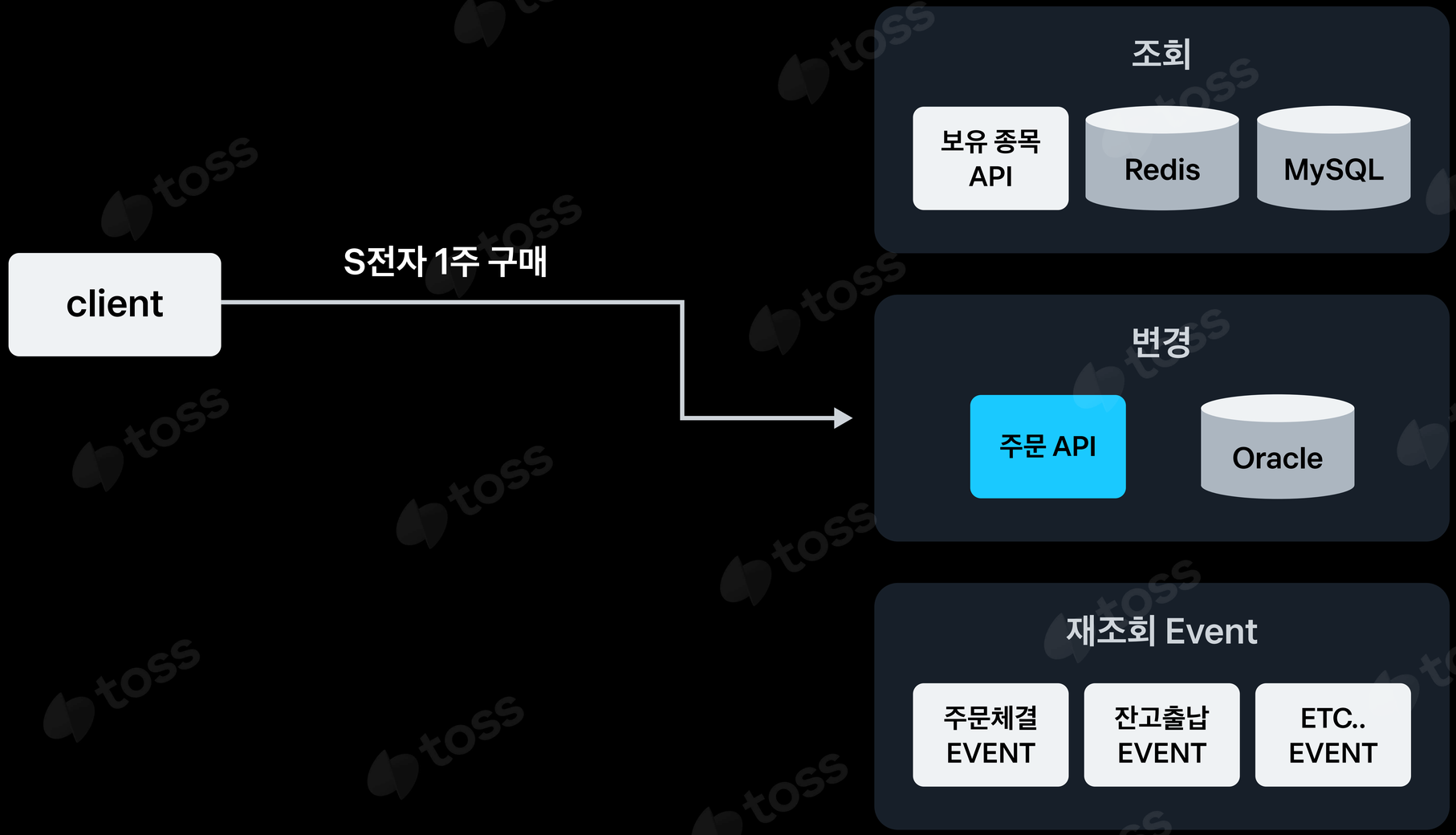
안드로이드 푸쉬알람 대체

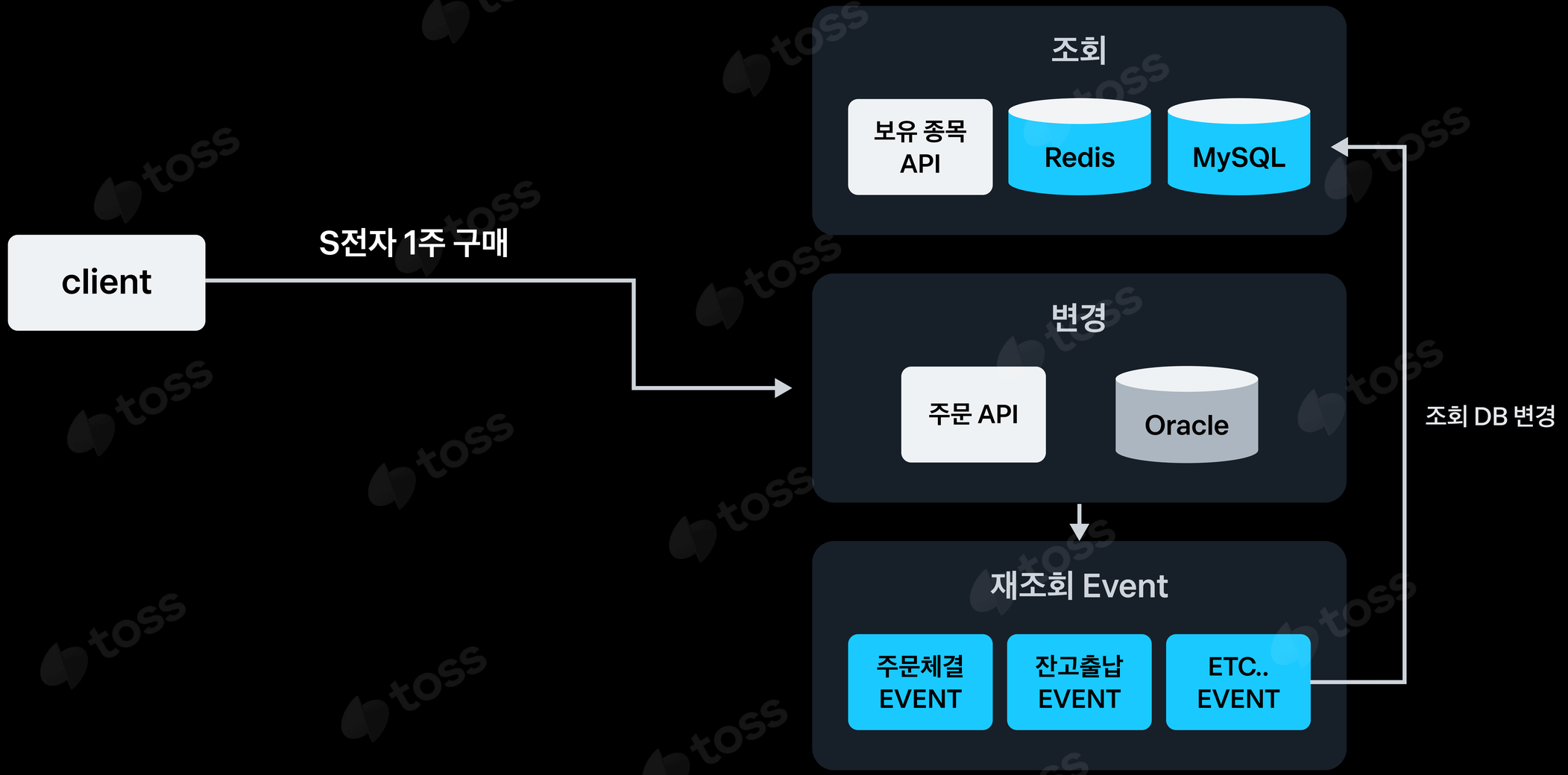
보유 자산 polling 제거

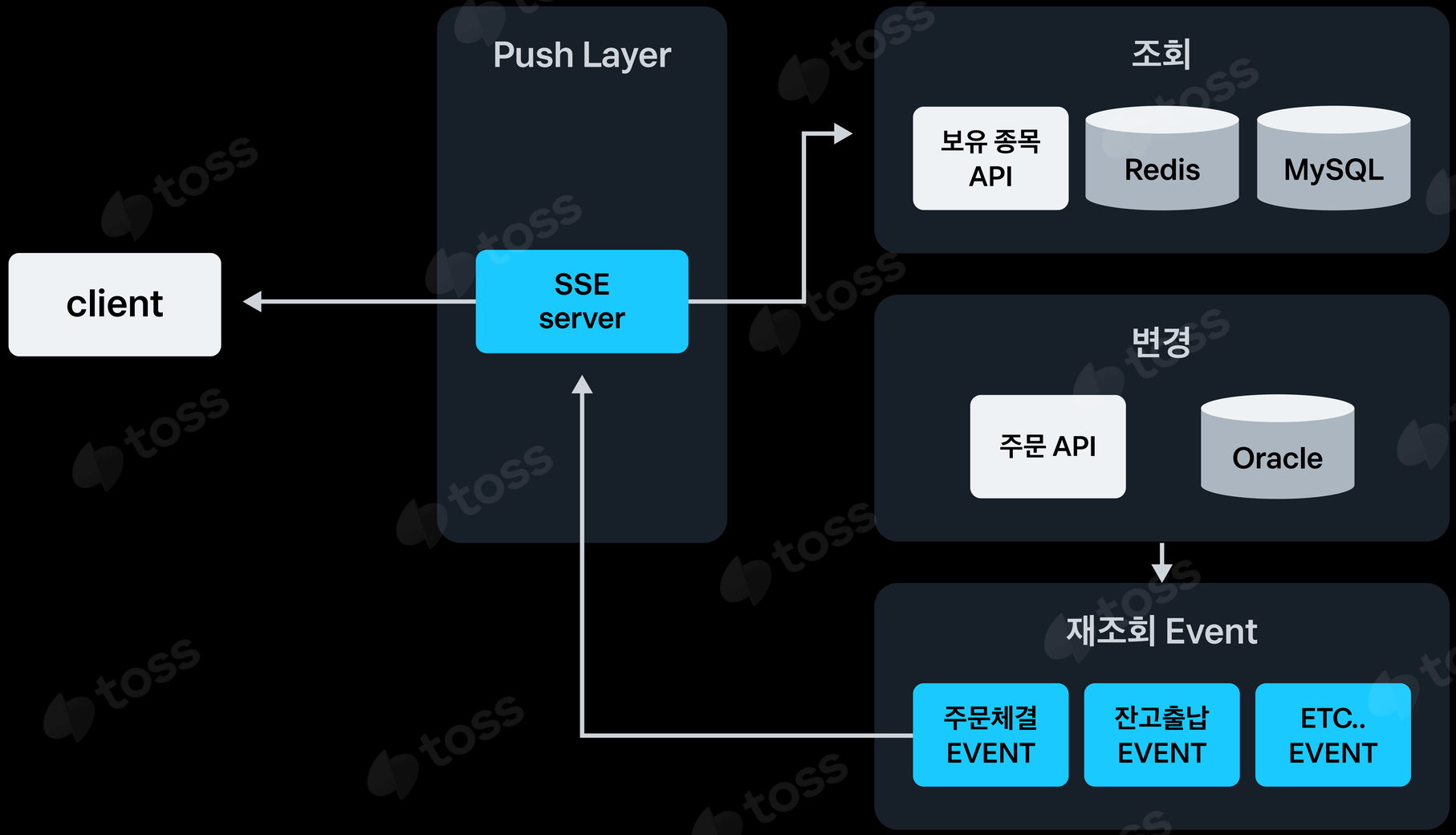


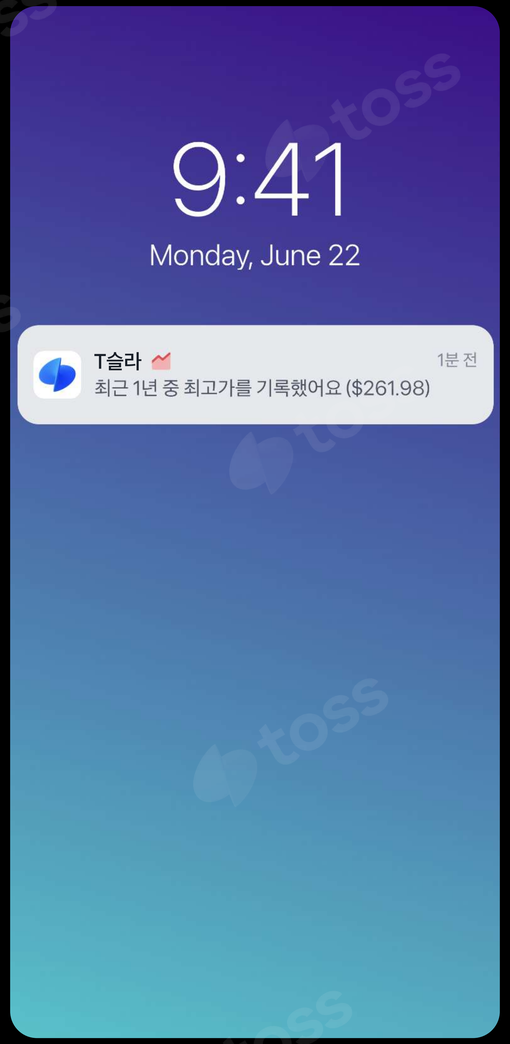

Push Layer
Resource Layer
client
PUSH SERVER
계좌 API
주문 정보 API
etc..
Redis
Oracle
MySQL
재조회 Event
주문체결 EVENT
잔고출납 EVENT
ETC.. EVENT


client
조회
보유 종목 API
Redis
MySQL
변경
주문 API
Oracle
재조회 Event
주문체결 EVENT
잔고출납 EVENT
ETC.. EVENT


client
S전자 1주 구매
조회
보유 종목 API
Redis
MySQL
변경
주문 API
Oracle
재조회 Event
주문체결 EVENT
잔고출납 EVENT
ETC.. EVENT

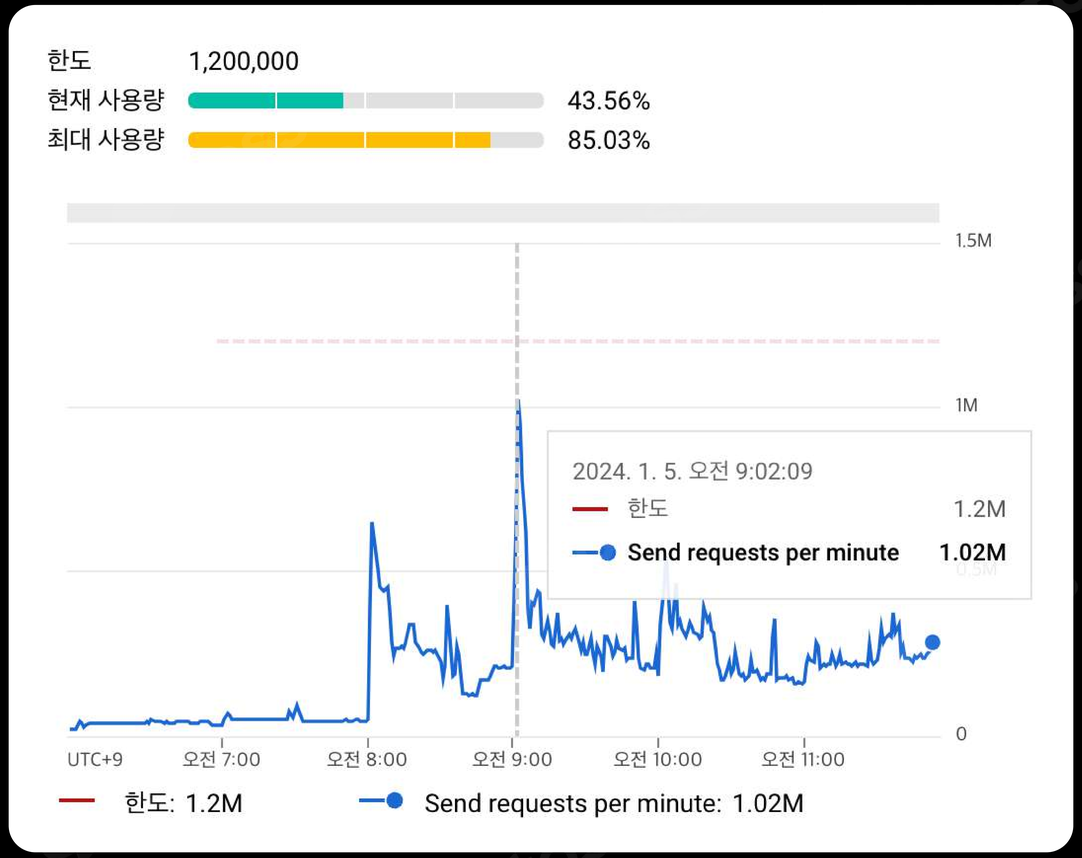

조회
보유 종목 API
Redis
MySQL
client
S전자 1주 구매
변경
주문 API
Oracle
조회 DB 변경
재조회 Event
주문체결 EVENT
잔고출납 EVENT
ETC.. EVENT

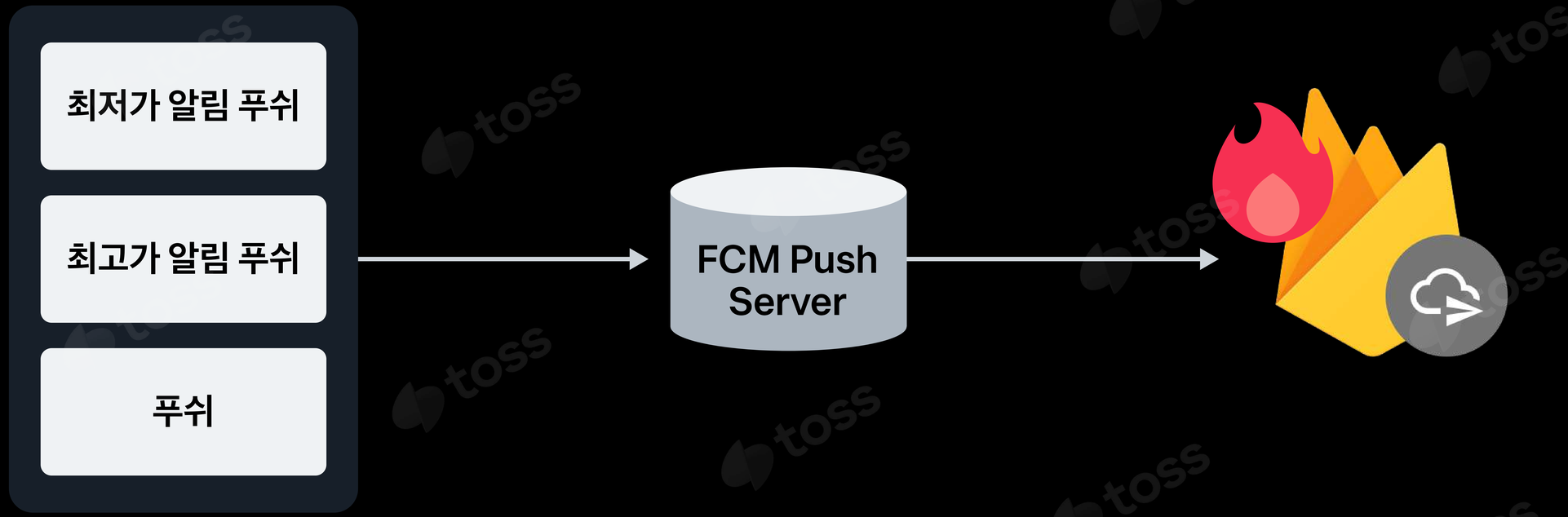

Push Layer
client
SSE server
조회
보유 종목 API
Redis
MySQL
변경
주문 API
Oracle
재조회 Event
주문체결 EVENT
잔고출납 EVENT
ETC.. EVENT

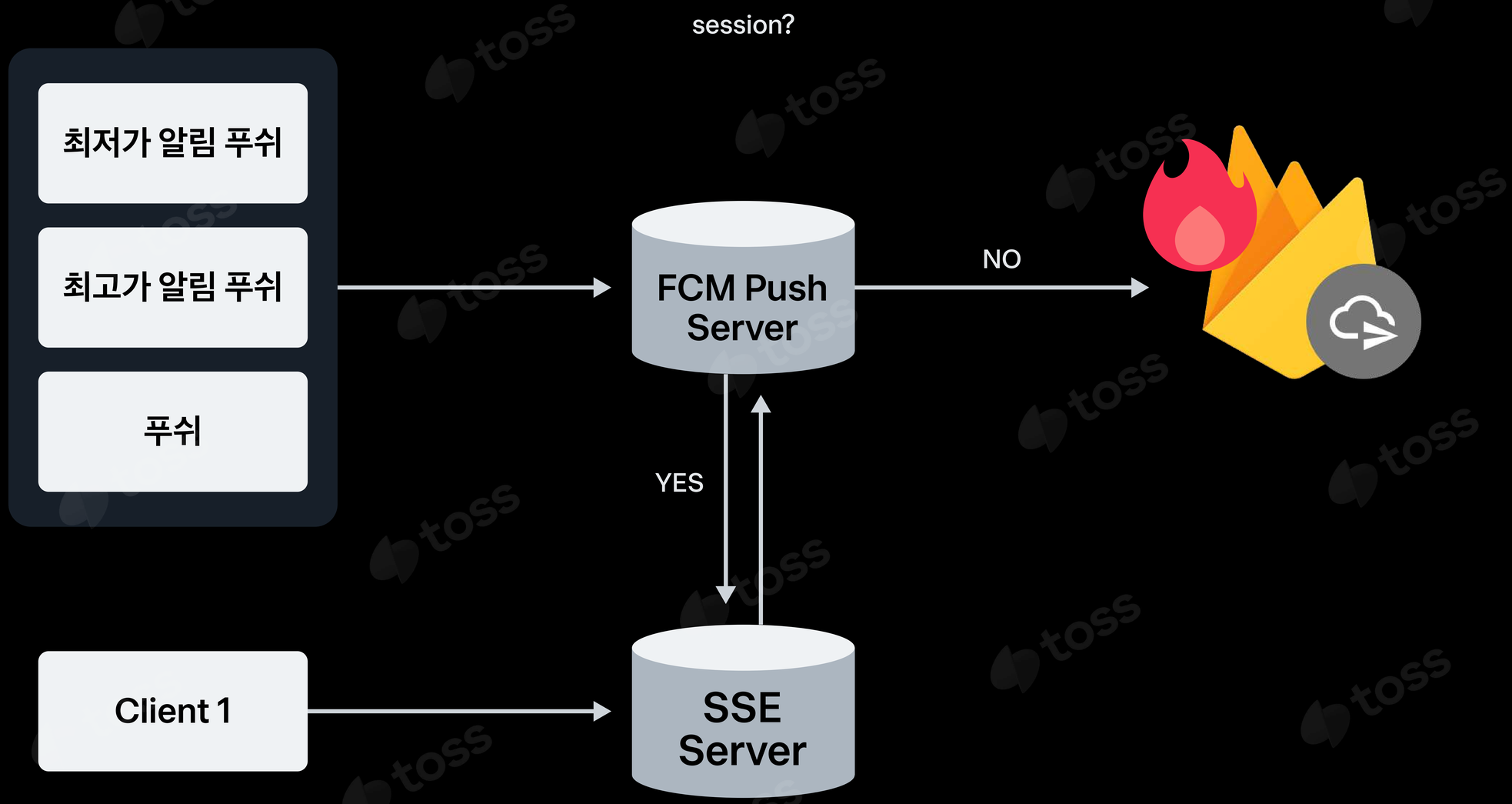


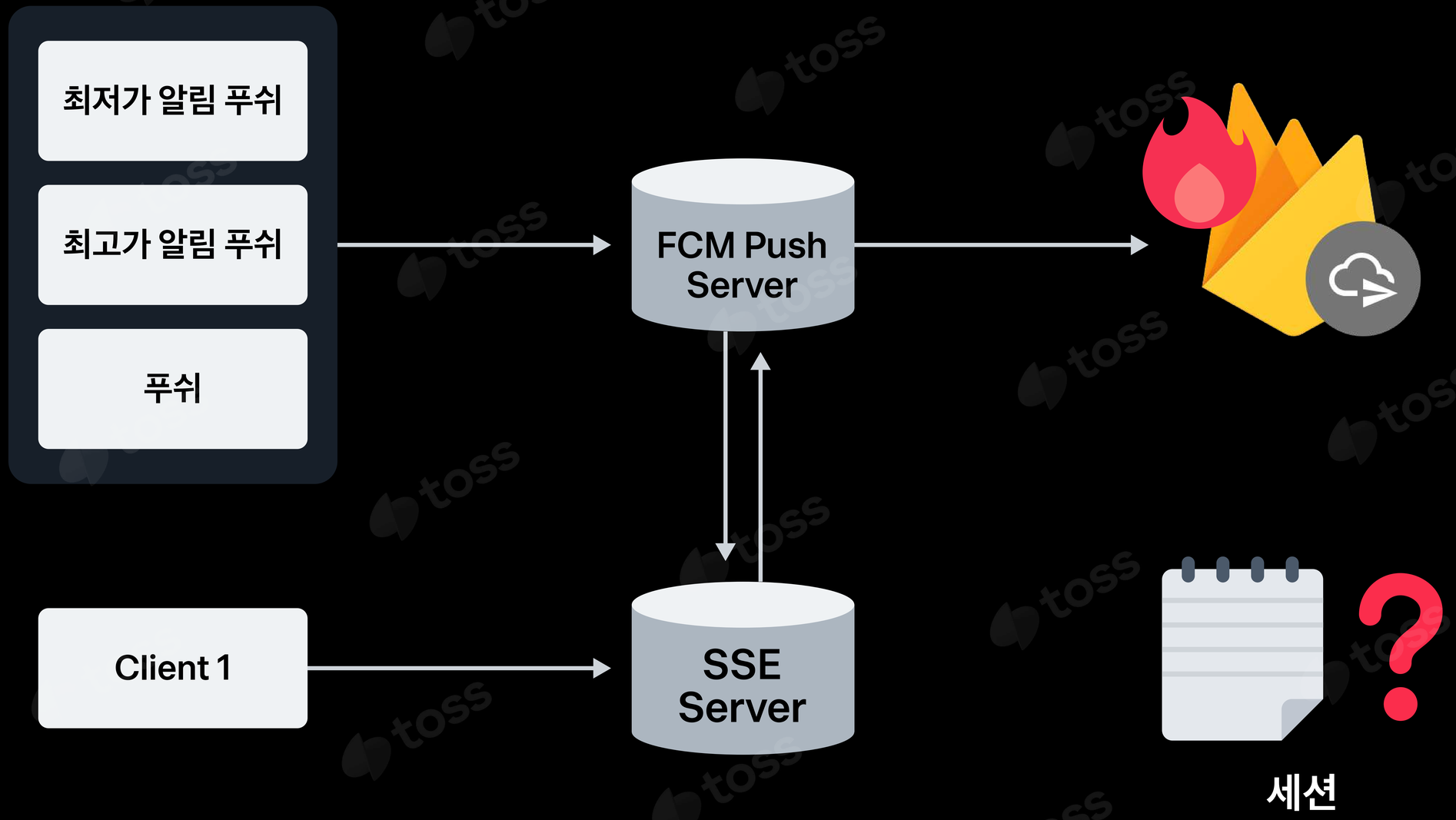
*이미지 내 종목은 단순 참고용일뿐 투자 권유나 종목 추천이 아닙니다.

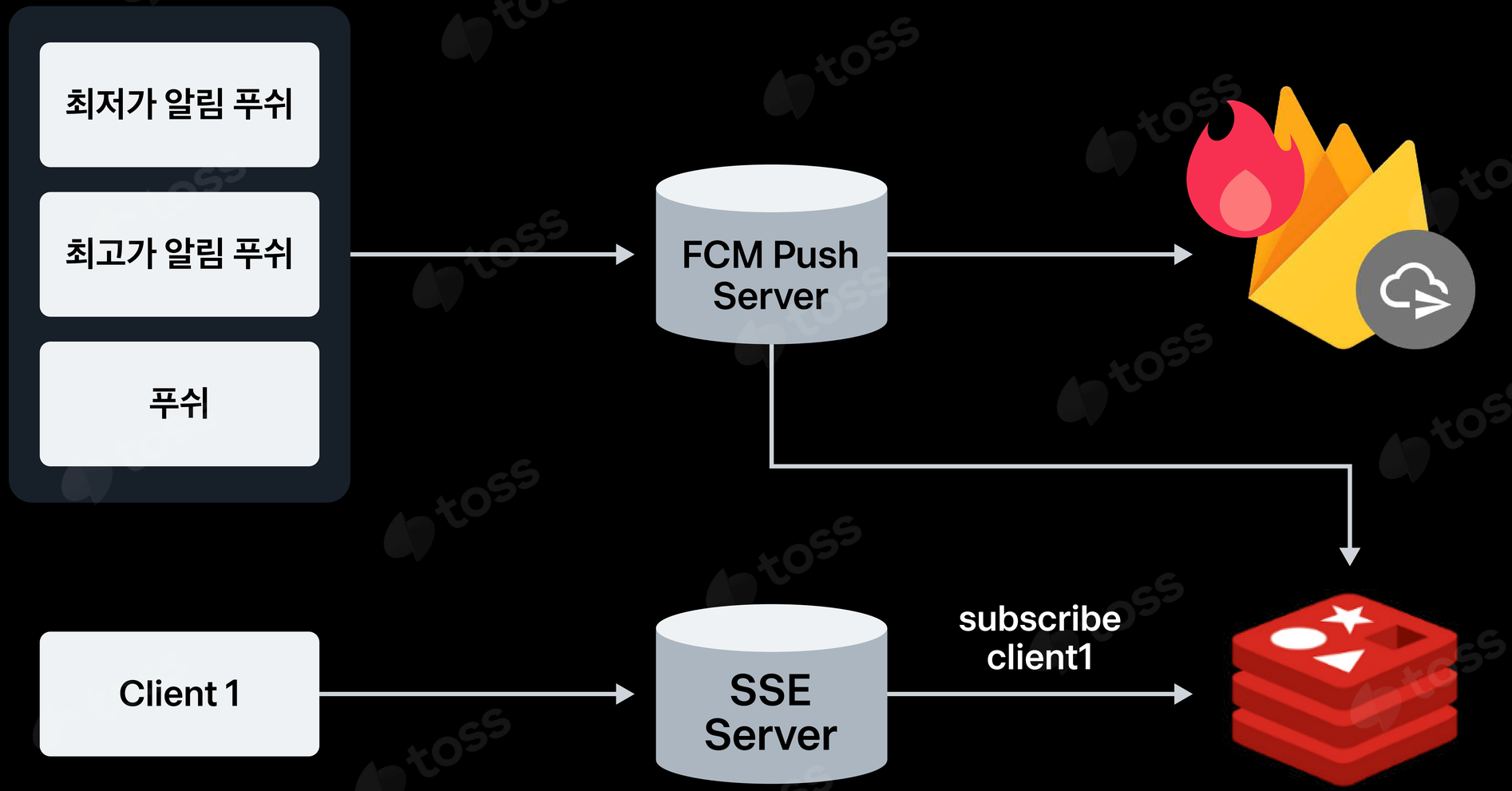
FCM message push

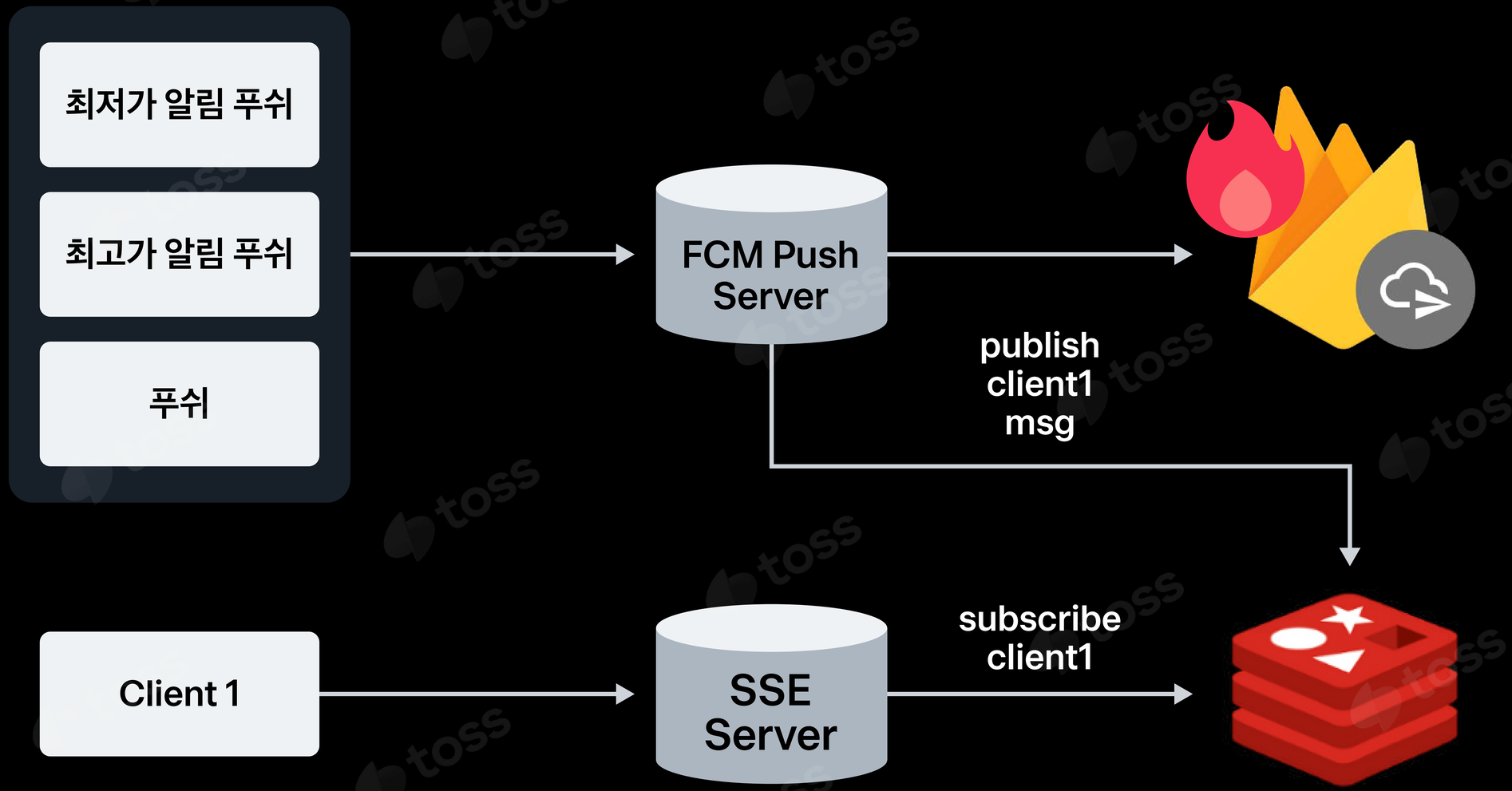


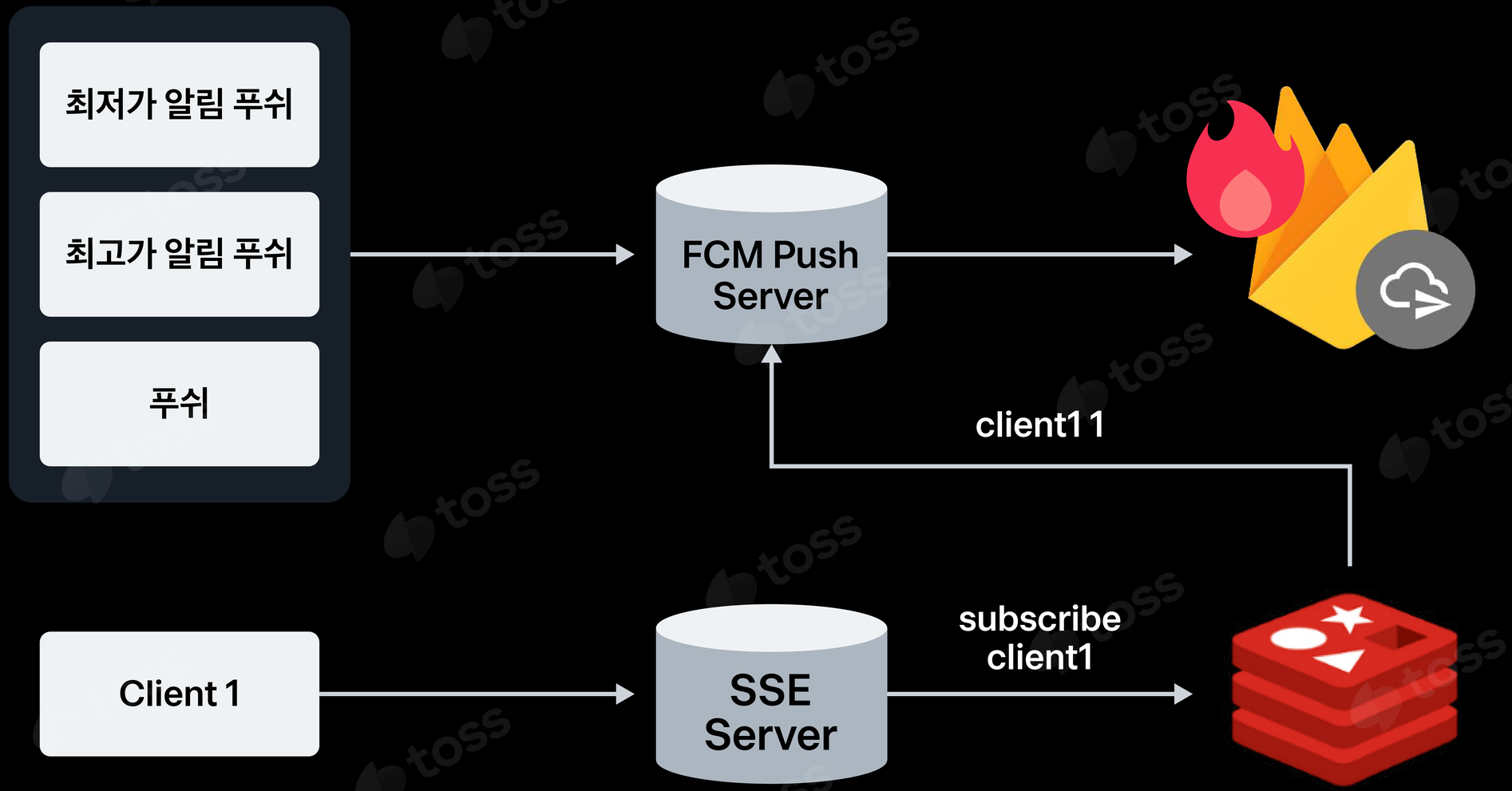

최저가 알림 푸쉬
최고가 알림 푸쉬
푸쉬
FCM Push Server


session?
최저가 알림 푸쉬
최고가 알림 푸쉬
푸쉬
FCM Push Server
NO
YES
Client 1
SSE Server

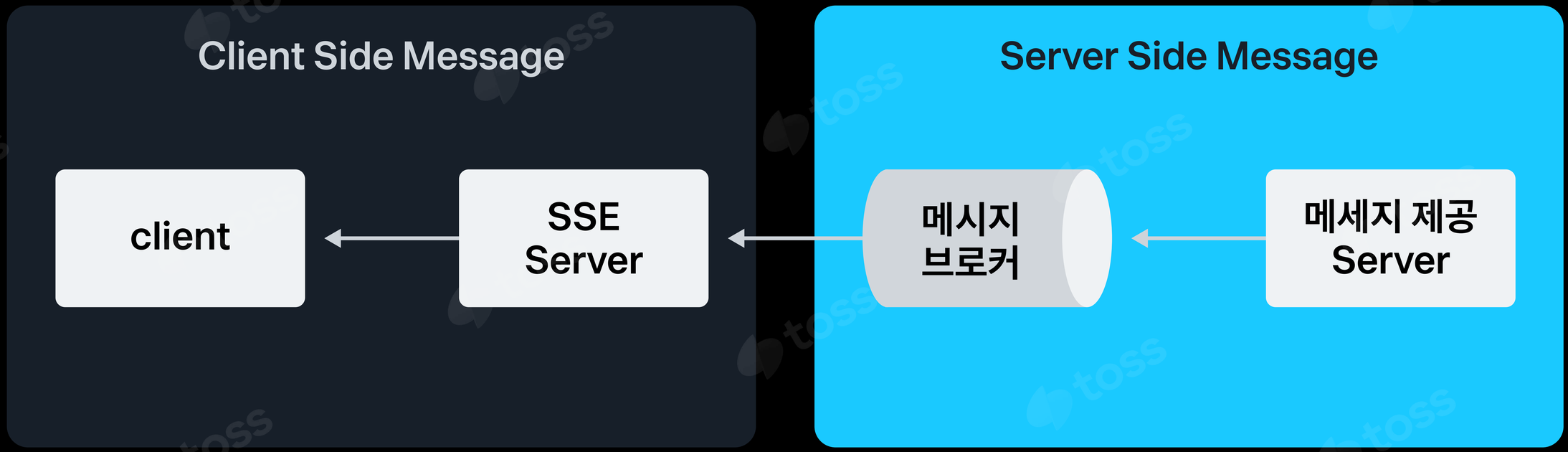

최저가 알림 푸쉬
최고가 알림 푸쉬
푸쉬
FCM Push Server
Client 1
SSE Server
세션


최저가 알림 푸쉬
최고가 알림 푸쉬
푸쉬
FCM Push Server
Client 1
SSE Server
subscribe client1


최저가 알림 푸쉬
최고가 알림 푸쉬
푸쉬
FCM Push Server
publish client1 msg
Client 1
SSE Server
subscribe client1


최저가 알림 푸쉬
최고가 알림 푸쉬
푸쉬
FCM Push Server
client1 1
Client 1
SSE Server
subscribe client1

SSE


Client Side Message
client
SSE
Server
Server Side Message
메시지
브로커
메세지 제공
Server

Message broker 선택 전략

kafka

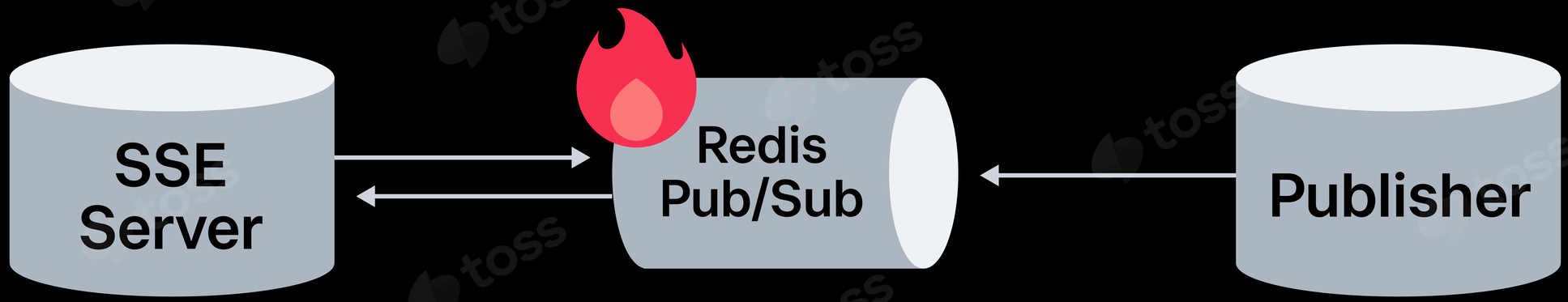
redis pub/sub

nats

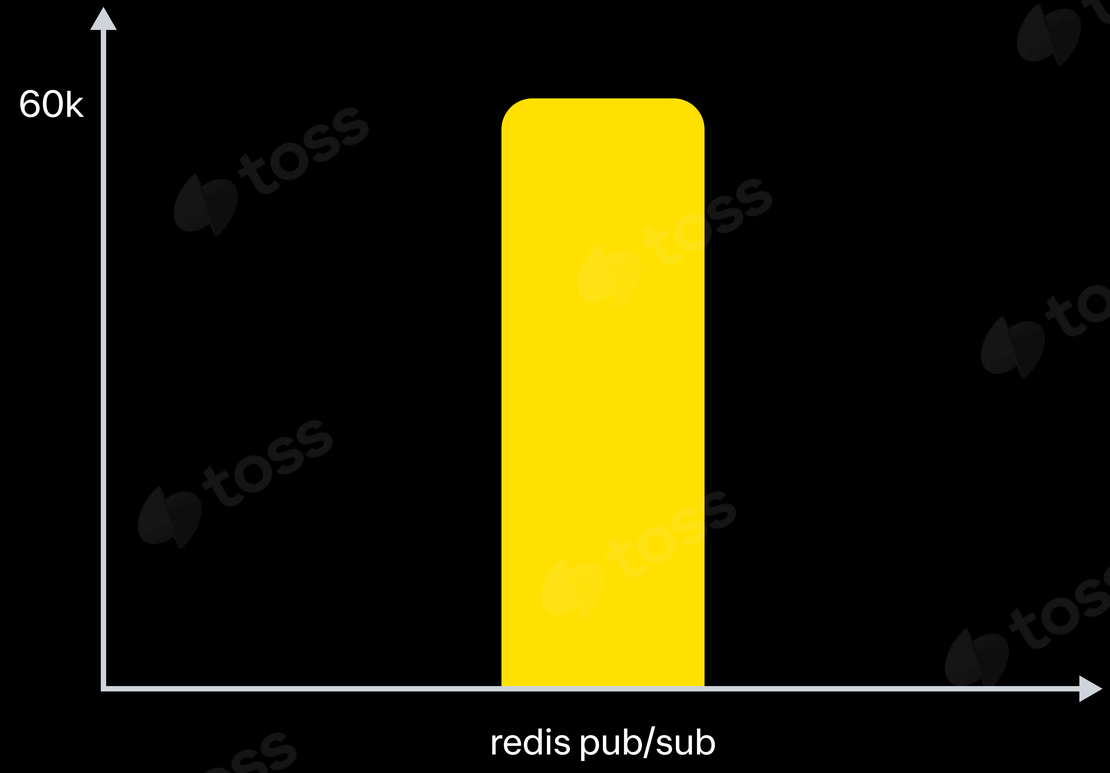

kafka


SSE
Server
접속
사용자
접속
사용자
접속X
접속X
Kafka

Redis pub/sub



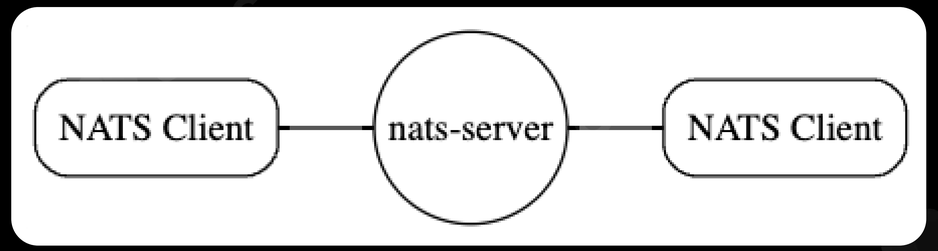
성능 테스트

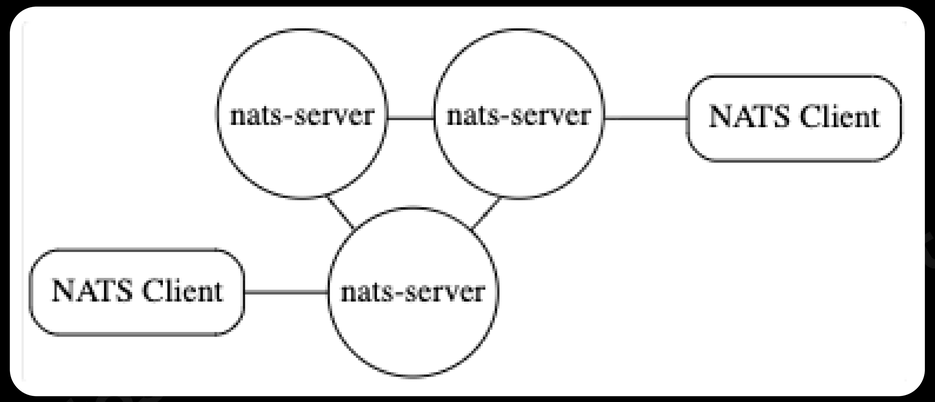


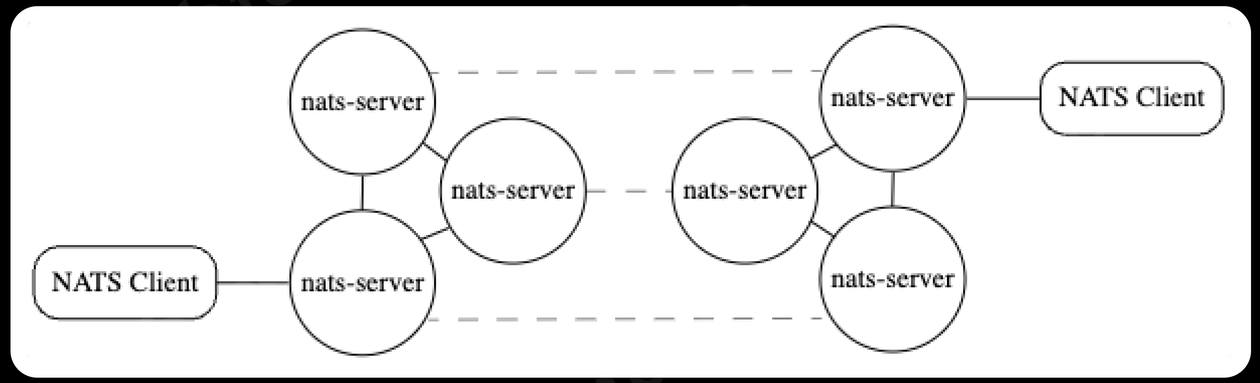
Brokered Throughput

SLASH 24

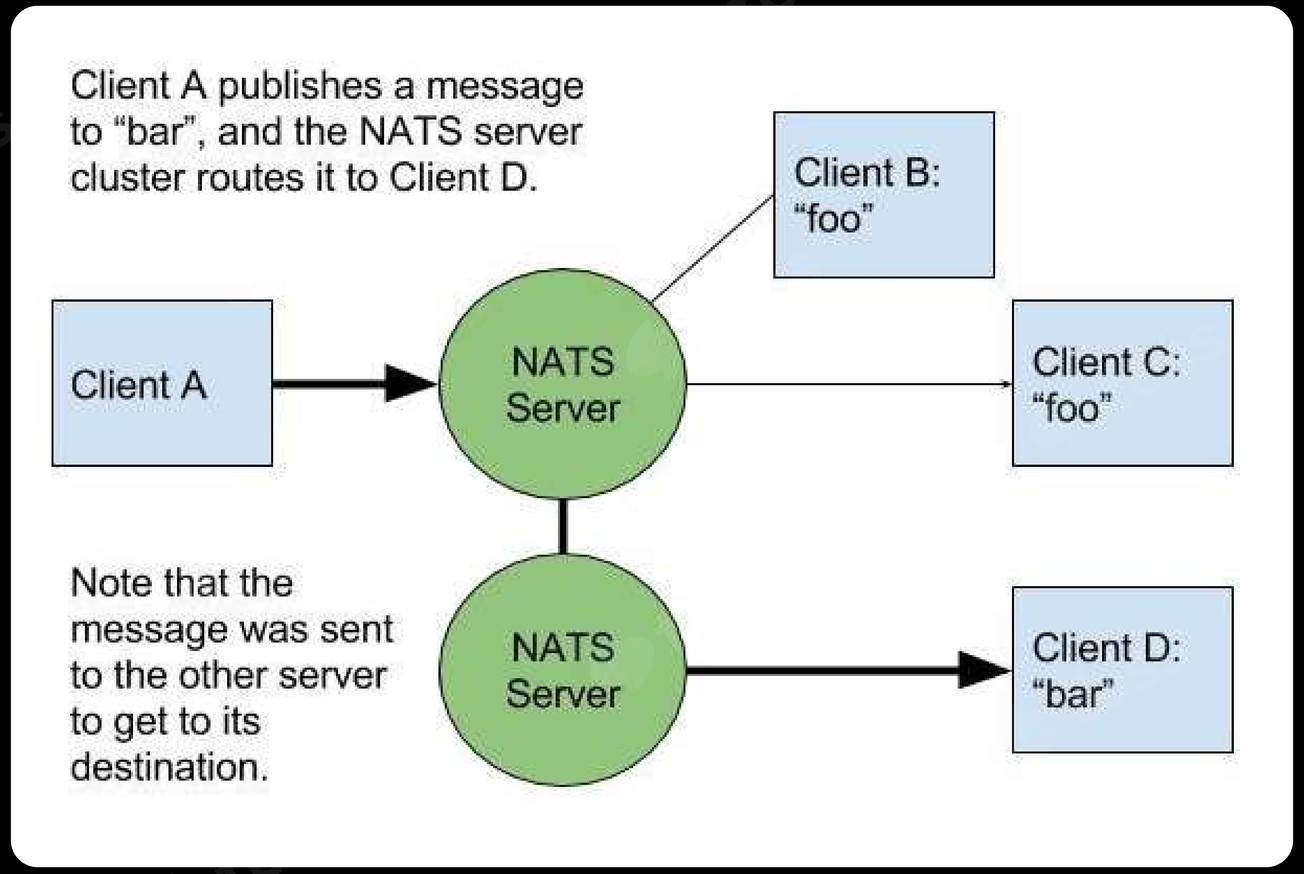
NATS



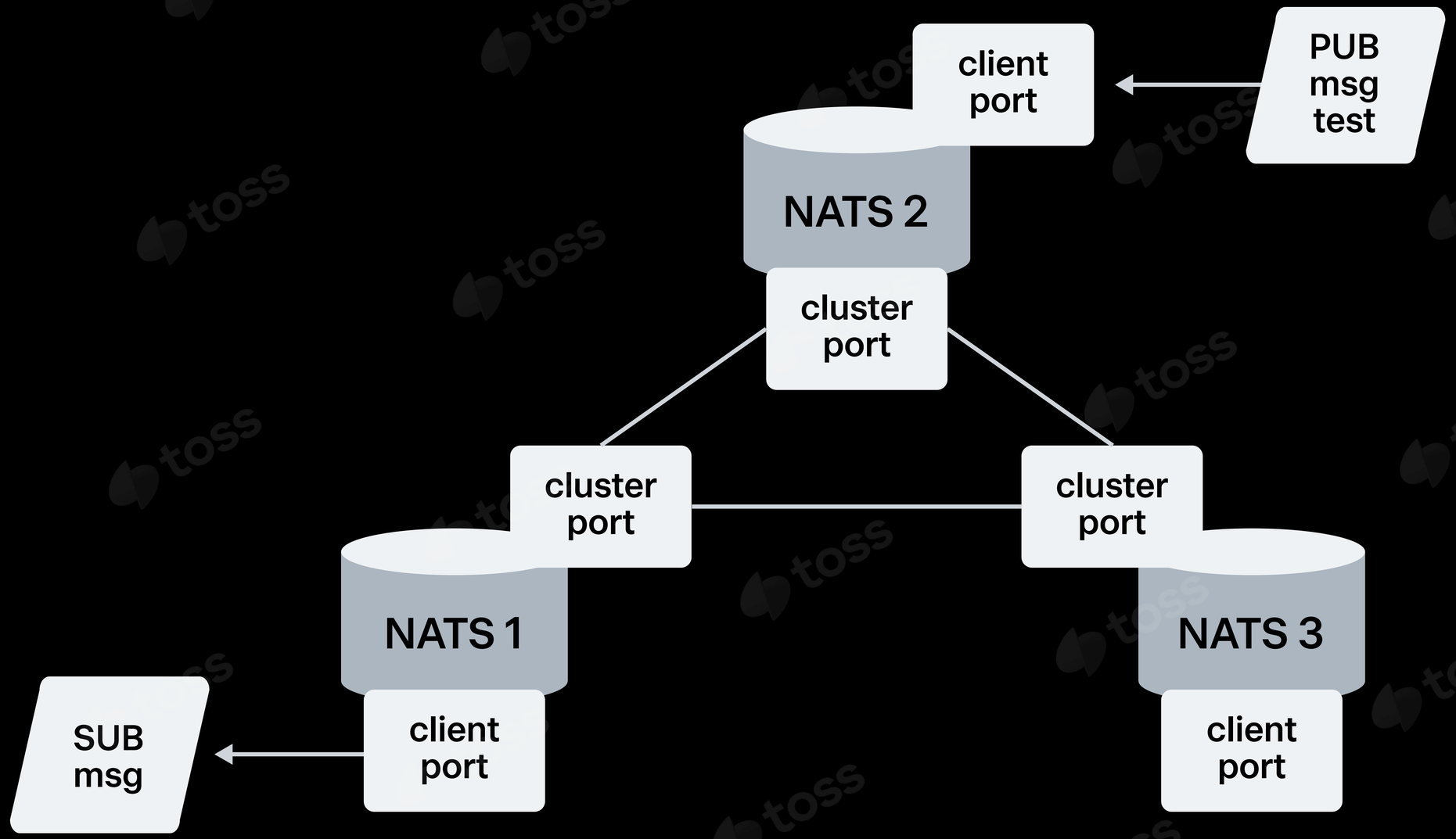


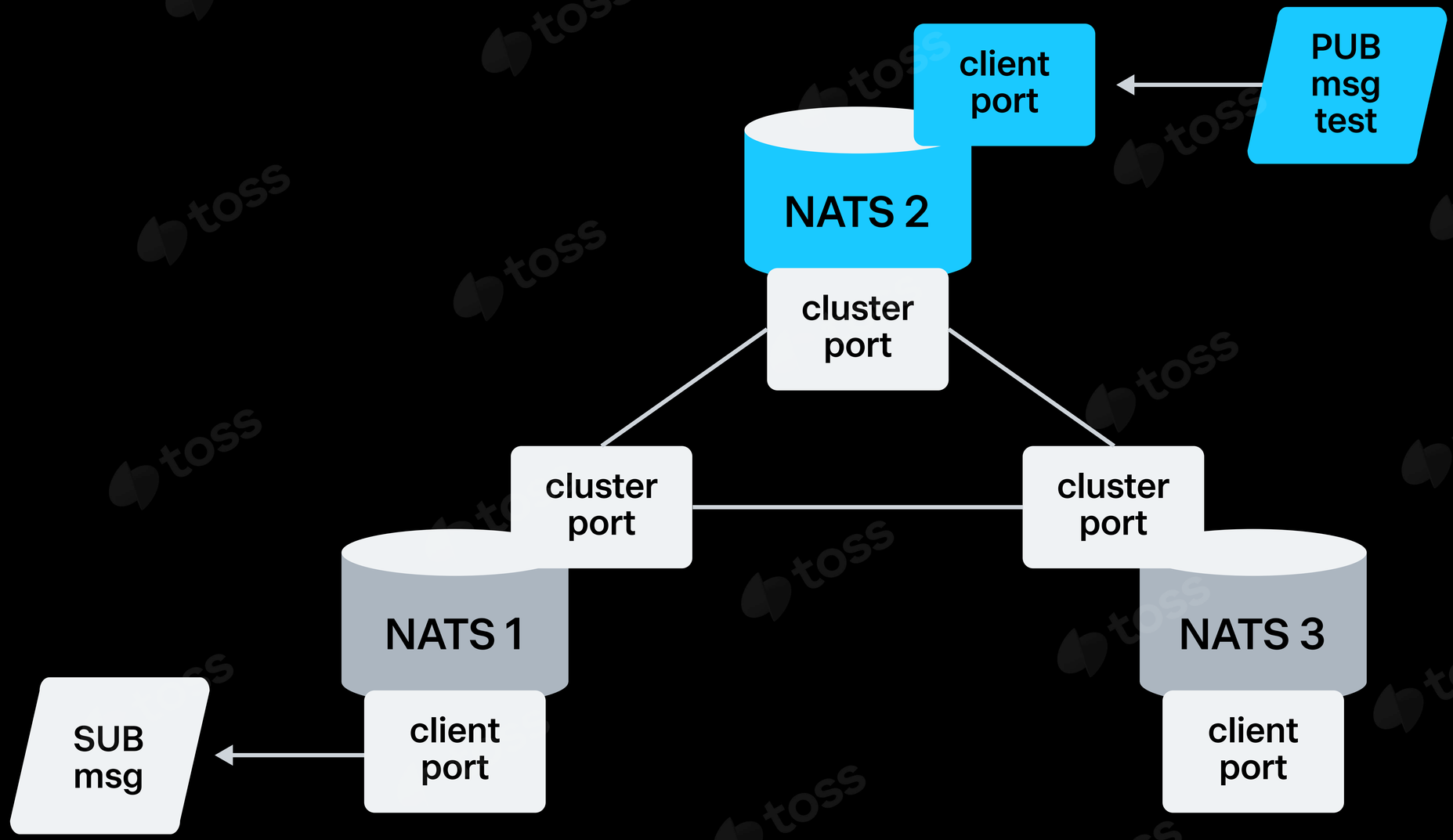


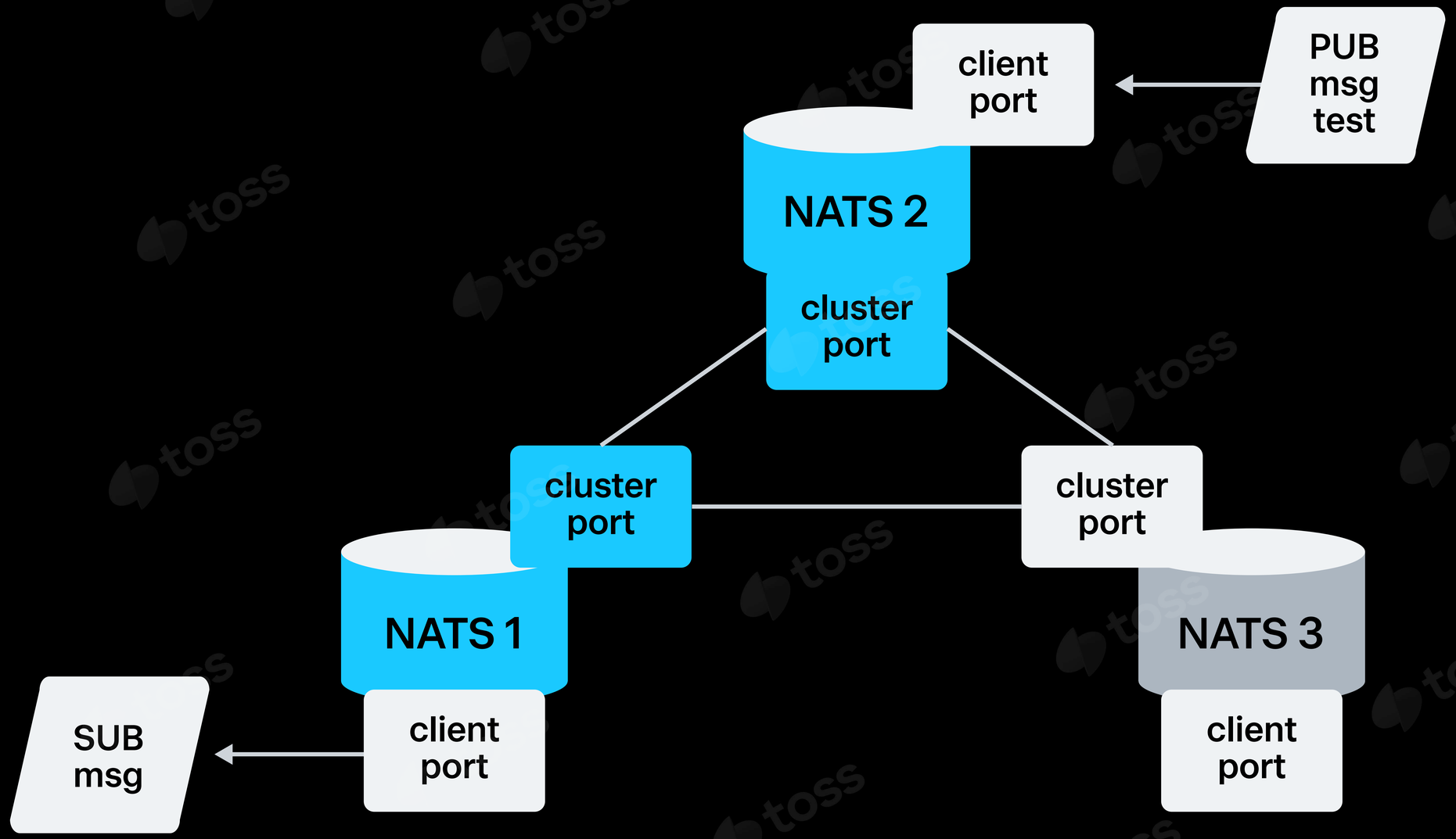
출처 : https://docs.nats.io/nats-concepts/service_infrastructure/adaptive_edge_deployment

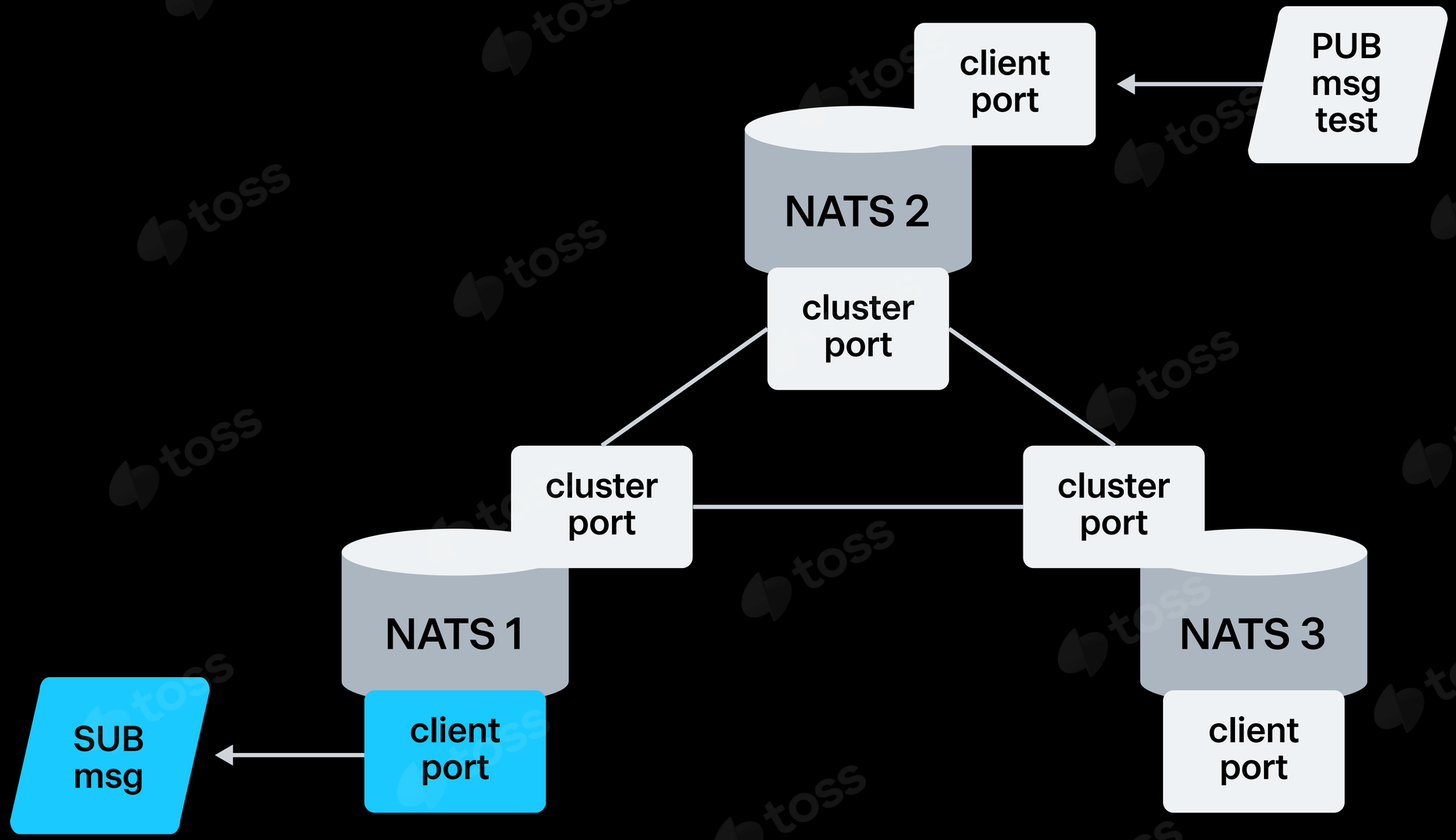


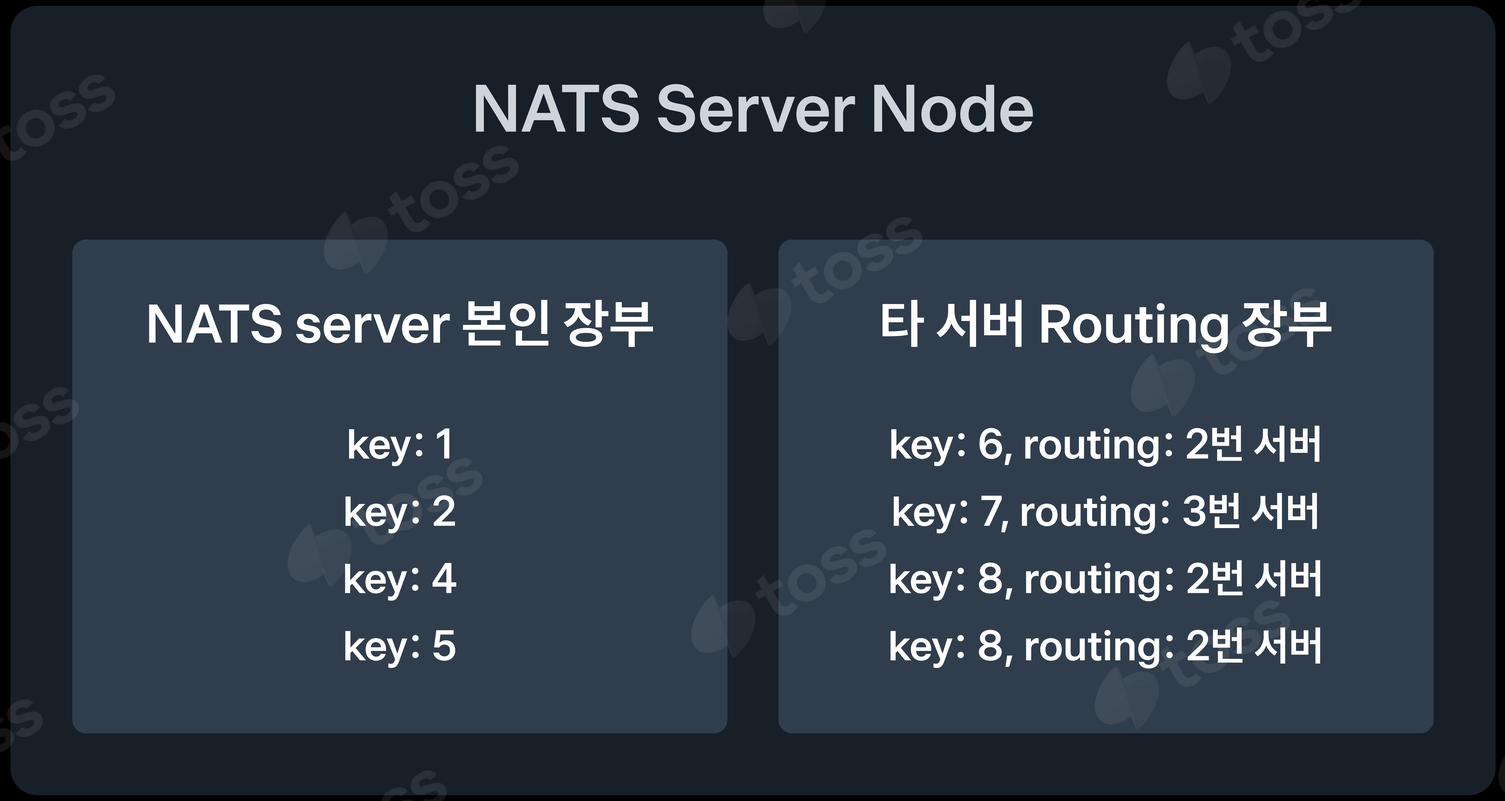
출처 : https://github.com/nats-io/nats-general/blob/main/architecture/ARCHITECTURE.md#subscriptions-and-routing

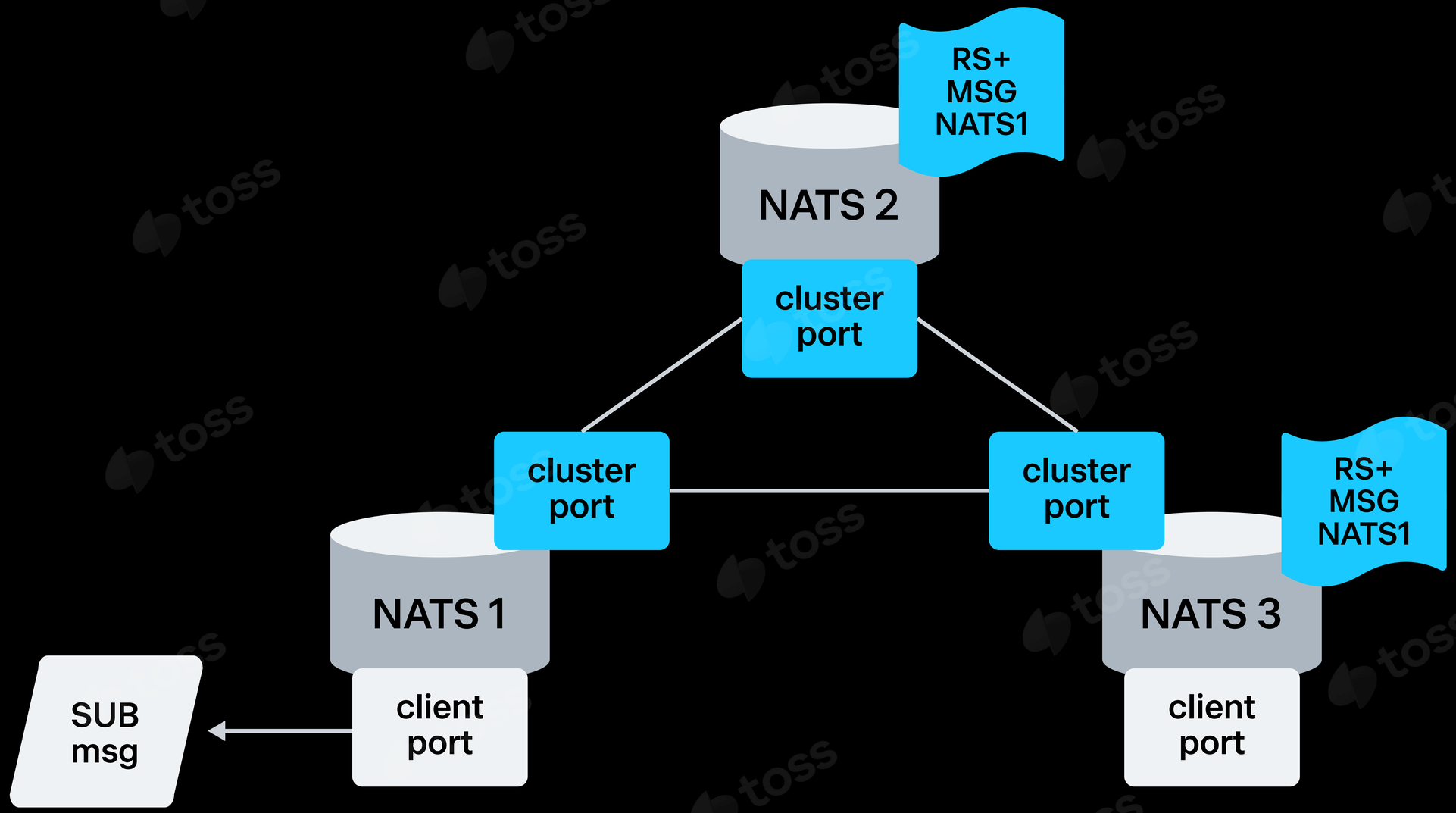

client port
PUB msg test
NATS 2
cluster port
cluster port
cluster port
NATS 1
NATS 3
SUB msg
client port
client port


client
port
PUB
msg
test
NATS 2
cluster
port
cluster
port
cluster
port
NATS 1
NATS 3
SUB
msg
client
port
client
port

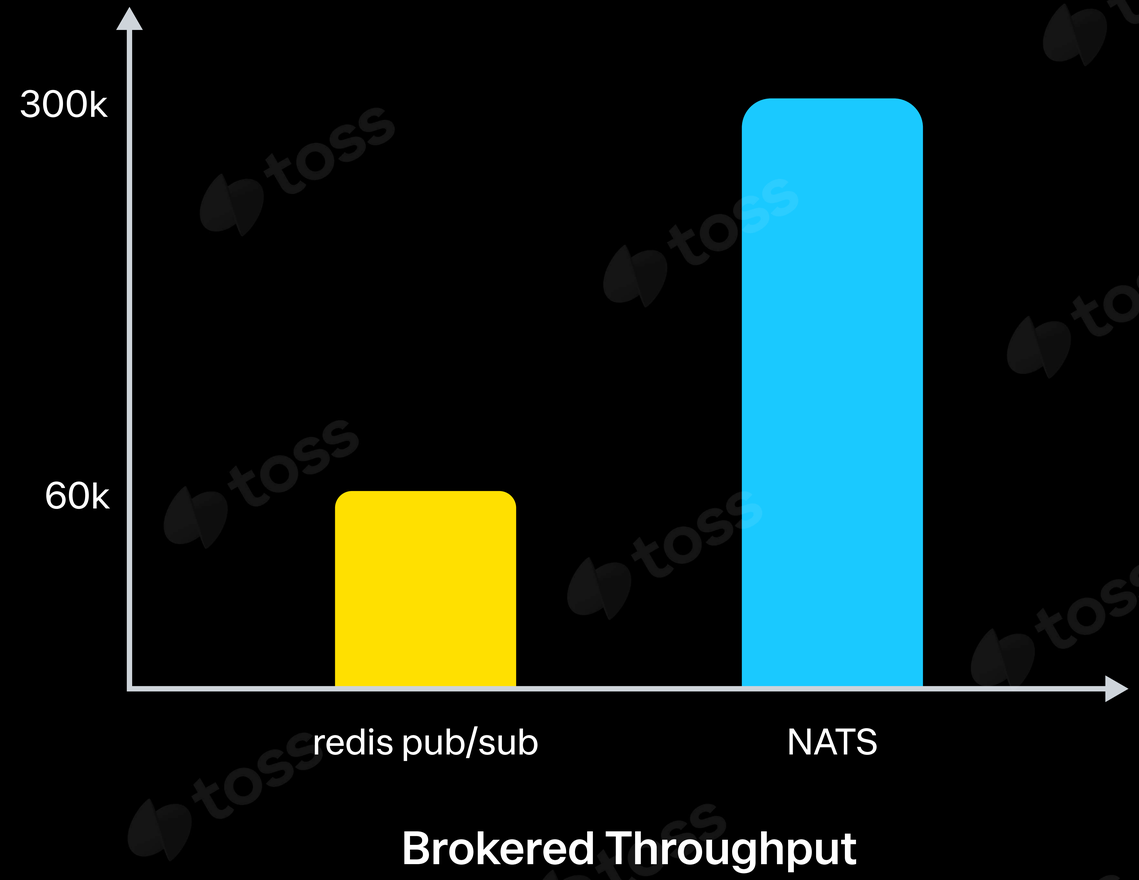

client
port
PUB
msg
test
NATS 2
cluster
port
cluster
port
cluster
port
NATS 1
NATS 3
SUB
msg
client
port
client
port


client
port
PUB
msg
test
NATS 2
cluster
port
cluster
port
cluster
port
NATS 1
NATS 3
SUB
msg
client
port
client
port

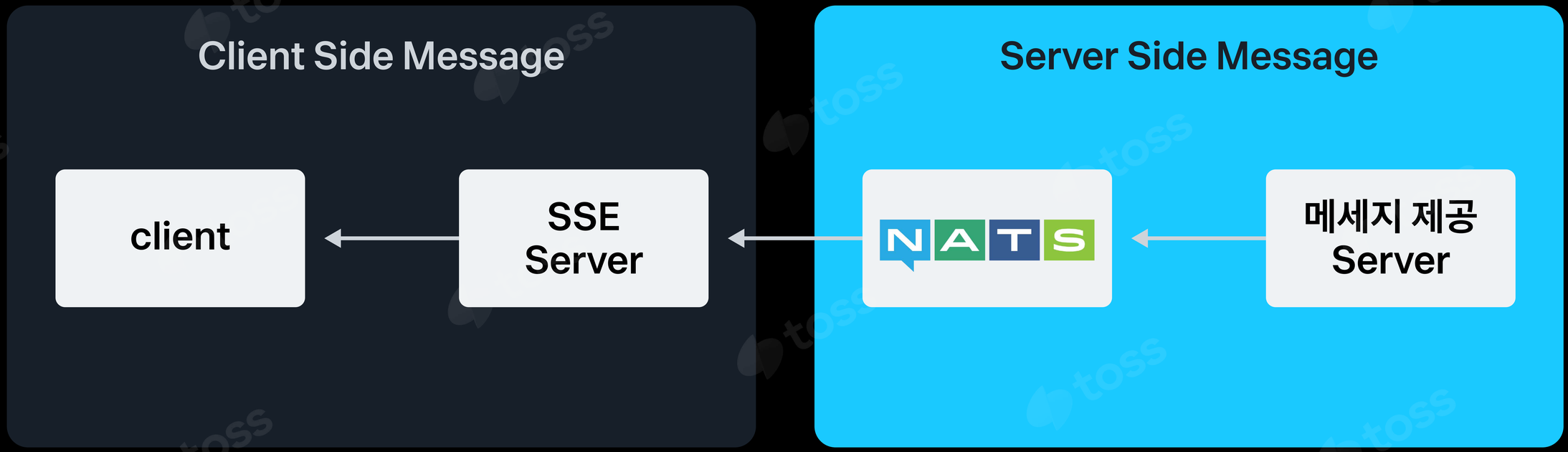

NATS Server Node
NATS server 본인 장부
key: 1
key: 2
key: 4
key: 5
타 서버 Routing 장부
key: 6, routing: 2번 서버
key: 7, routing: 3번 서버
key: 8, routing: 2번 서버
key: 8, routing: 2번 서버


RS+
MSG
NATS1
NATS 2
cluster
port
cluster
port
cluster
port
RS+
MSG
NATS1
NATS 1
NATS 3
SUB
msg
client
port
client
port

성능 테스트



SSE


Client Side Message
client
SSE
Server
Server Side Message
NATS
메세지 제공
Server

SLASH 24

트러블 슈팅

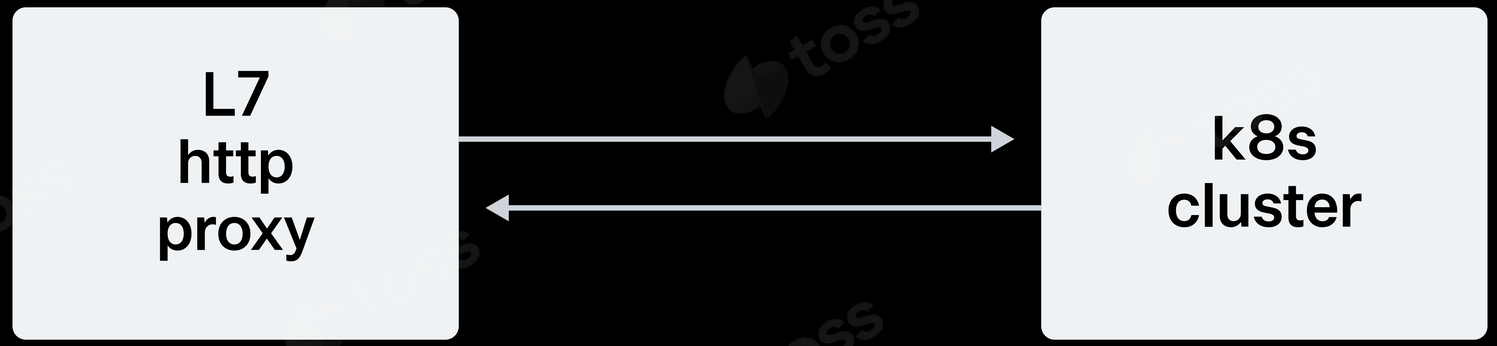
heartbeat

```
private fun heartbeat(): Flux<ServerSentEvent<T>> =
    Flux.interval(Duration.ofSeconds(2)) // heartbeat
        .map {
            ServerSentEvent.builder<T>()
                .comment("heartbeat")
                .retry(Duration.ofHours(1))
                .build()
        }
```

```
GET /api/v1/sse -> 200 took 10000ms
```

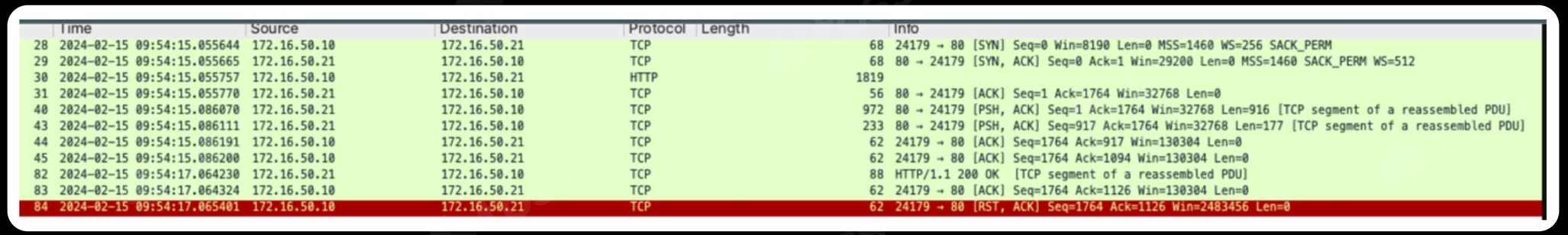
2 * n 초?

AS-IS



현상

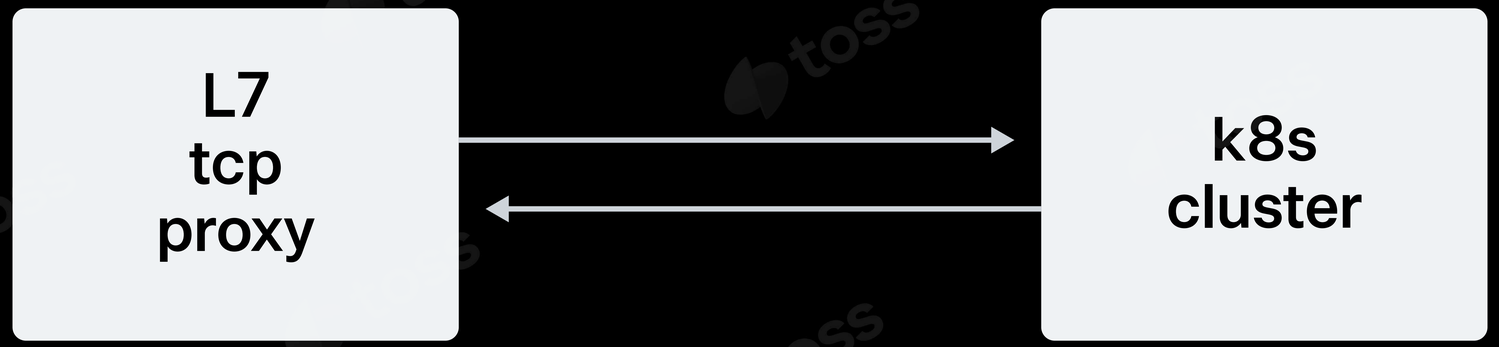
heartbeat



	Time	Source	Destination	Protocol	Length	Info
28	2024-02-15 09:54:15.055644	172.16.50.10	172.16.50.21	TCP	68	24179 → 80 [SYN] Seq=0 Win=8190 Len=0 MSS=1460 WS=256 SACK_PERM
29	2024-02-15 09:54:15.055665	172.16.50.21	172.16.50.10	TCP	68	80 → 24179 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=512
30	2024-02-15 09:54:15.055757	172.16.50.10	172.16.50.21	HTTP	1819	
31	2024-02-15 09:54:15.055770	172.16.50.21	172.16.50.10	TCP	56	80 → 24179 [ACK] Seq=1 Ack=1764 Win=32768 Len=0
40	2024-02-15 09:54:15.086070	172.16.50.21	172.16.50.10	TCP	972	80 → 24179 [PSH, ACK] Seq=1 Ack=1764 Win=32768 Len=916 [TCP segment of a reassembled PDU]
43	2024-02-15 09:54:15.086111	172.16.50.21	172.16.50.10	TCP	233	80 → 24179 [PSH, ACK] Seq=917 Ack=1764 Win=32768 Len=177 [TCP segment of a reassembled PDU]
44	2024-02-15 09:54:15.086191	172.16.50.10	172.16.50.21	TCP	62	24179 → 80 [ACK] Seq=1764 Ack=917 Win=130304 Len=0
45	2024-02-15 09:54:15.086200	172.16.50.10	172.16.50.21	TCP	62	24179 → 80 [ACK] Seq=1764 Ack=1094 Win=130304 Len=0
82	2024-02-15 09:54:17.064230	172.16.50.21	172.16.50.10	TCP	88	HTTP/1.1 200 OK [TCP segment of a reassembled PDU]
83	2024-02-15 09:54:17.064324	172.16.50.10	172.16.50.21	TCP	62	24179 → 80 [ACK] Seq=1764 Ack=1126 Win=130304 Len=0
84	2024-02-15 09:54:17.065401	172.16.50.10	172.16.50.21	TCP	62	24179 → 80 [RST, ACK] Seq=1764 Ack=1126 Win=2483456 Len=0

이미 끊어진 connection에 ACK를 응답하다보니 RST 날라옴

TO-BE



정리

1. polling 대체제?
2. message broker 전략을 어떻게 가져갈지?
3. SSE connection이 정상 close가 됬는지 모니터링 도구

SLASH 24