차세대 000 아키텍처 주제 #6 – RP 서비스화

송태국(tsong@nextree.co.kr)
넥스트리 소프트㈜
2012.1.4(수)

NEXTREE

RP와 FastTP는 차세대 000의 핵심이슈이므로

✓ RP 서비스화

를 아키텍처 주제로 하여 설계를 하고,

✓ RP 서비스화 RI

를 통해서 성능을 비롯한 이슈를 검증합니다.

RP 서비스화 - 목차

1. RP 서비스화 개요

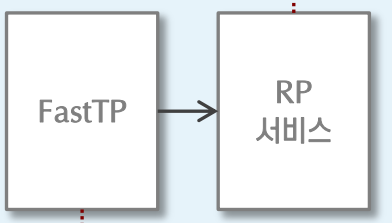
- ✓ RP 서비스는 000 서비스의 핵심 서비스이며, RP 서비스의 품질과 성능이 000 서비스의 품질과 성능으로 직결됨
- ✓ Fast TP는 이슈가 되고 있는 아키텍처 요소로써 차세대 000에서는 반드시 개선되어야 할 요소임
- ✓ RP 서비스화는 내부 개선보다는 RP 서비스로의 접근을 개선하여 보다 쉽고 안정적인 RP 서비스 사용에 초점을 두고있음

RP 서비스 영역의 핵심 모듈은 통신 인프라인 FastTP와 서비스 모듈인 RP 두 가지임

RP는 10년간의 경험이 직접되어 최고의 길찾기 효율을 보여주고 있으며, 수행성능향상 니즈가 존재함

FastTP 소스 제어권이 없어 운영리스크가 상존함

RP서비스의 핵심 모듈



RP 서비스는 길탐색 품질은 매우 높은 수준을 유지하고 있지만, 사용자 폭증과 실시간성 요구에 따른 성능향상 니즈가 있음.

성능향상은 알고리즘 개선이 아닌 SW 설계 기술, 특히 실시간 특성을 유지하면서 탐색결과 재사용, 동시처리 등을 통한 처리시간 단축이 가능할 것으로 보임 ← 아키텍처 컨설팅 범위에서 제외함

FastTP는 과거 장애를 겪었던 통신 인프라로 개발사의 폐쇠로 인해 유지보수가 어려우며 소스코드에 대한 소유 및 제어권을 확보하지 못하여 운영리스크가 상존하는 모듈임

차세대 000 시스템에서 보다 나은 모듈로 대체되거나(이 경우 Ownership 확보가 매우 중요함) 다른 설계방식으로 대체되어야 함

업무영역		LoC(라인)	파일개수(개)
RP-RSD	RP	66,663	203
	RSD	188,246	343
	LIB(RP+RSD)	150,701	521

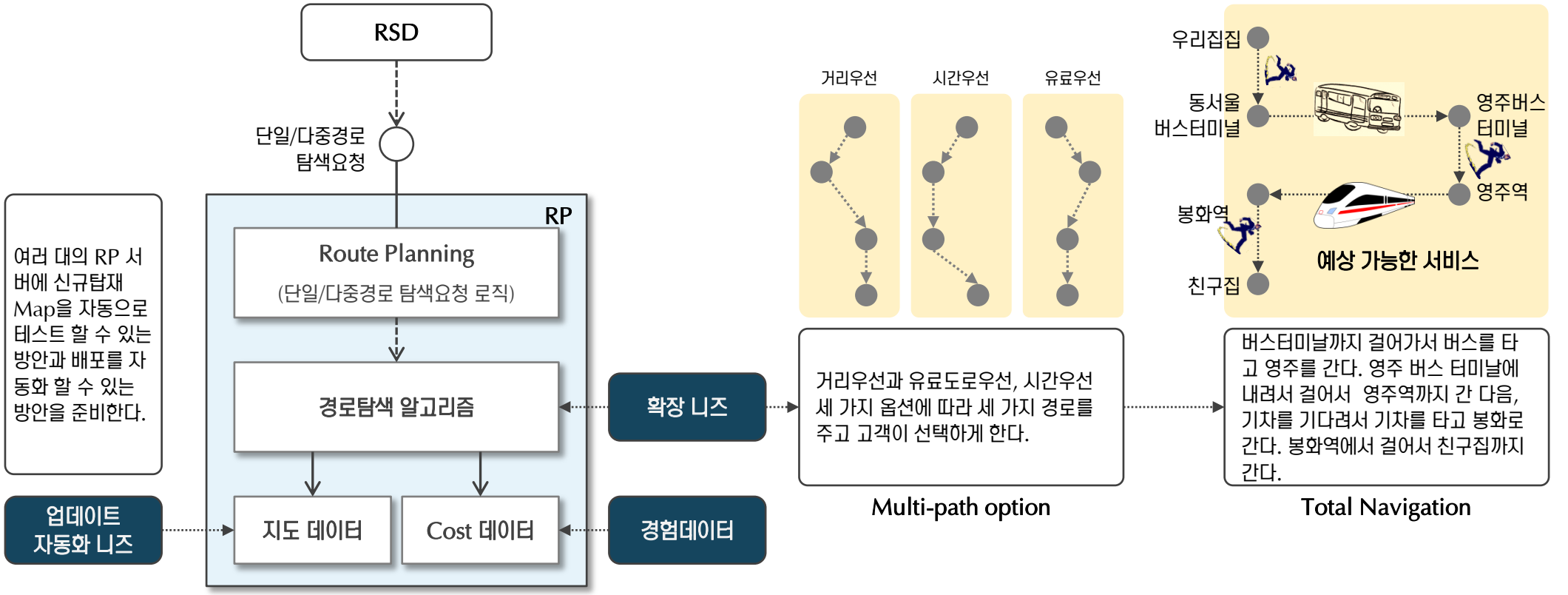
RP 소스코드 분석 결과 66,000 라인이며, 사용하는 라이브러리 RP + RSD 하여 15만 라인 포함하여 대략 20만 라인 정도로 구성됨

요약: RP 서비스화의 문제는 처리성능과 관련된 RP 내부 문제와 RP 서비스로의 접근을 위한 RP 접근 문제로 요약할 수 있음.

RP 서비스화 아키텍처 설계주제의 초점은 RP 내부 문제가 아니라 RP 서비스로 접근 방법 개선을 통한 편리하고 안정적인 RP 서비스 제공에 있음

2. RP 이해 (1/3) – 업무

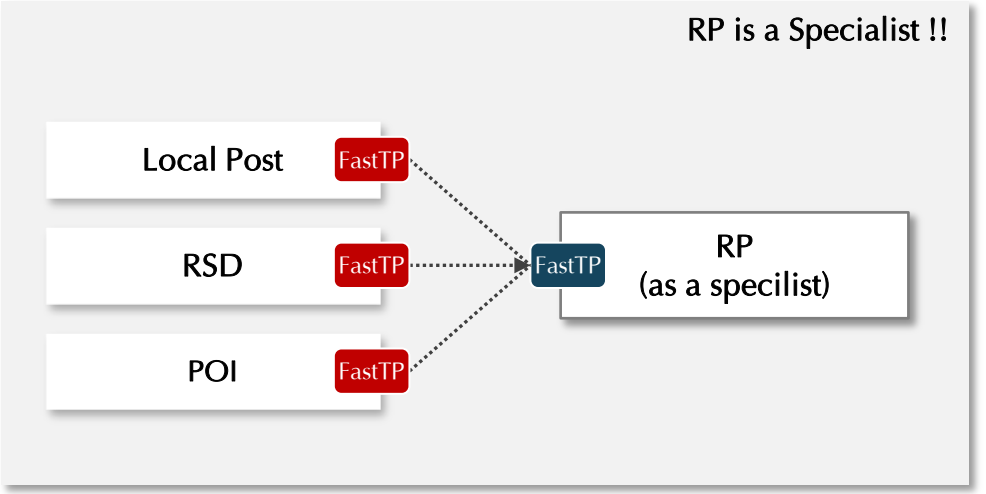
- ✓ 개선 포인트 1 : Map 배포 후 검증 자동화 및 효율화, 현행 서버가 늘어날수록 배포시간이 지속적으로 늘어남
- ✓ 개선 포인트 2 : 경로 탐색 확장, 다양한 이동수단(보행, 버스, 기차 등) 요구나 다수의 경로요구가 예상됨
- ✓ 개선 포인트 3 : 알고리즘 성능 향상, 현행 경로탐색 알고리즘을 계속 사용하면서 성능을 높일 수 있는 방안이 필요함

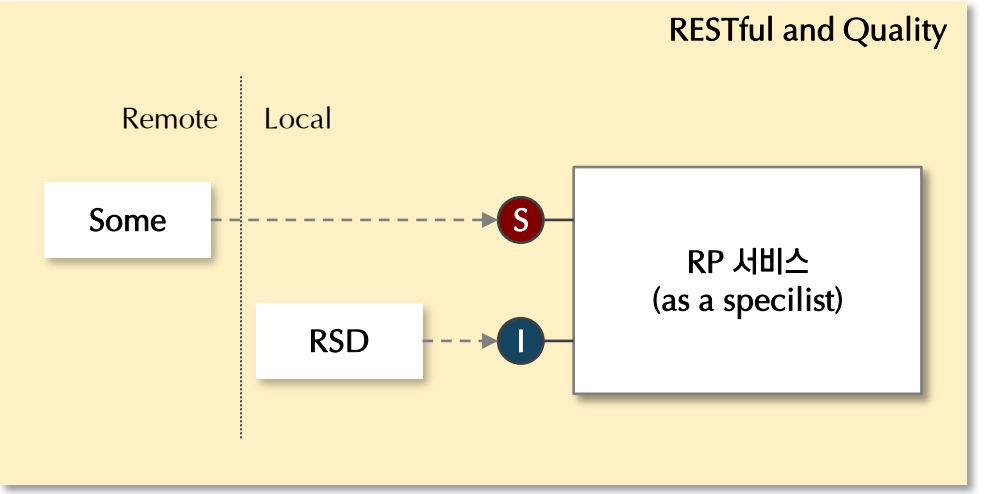


RP 업무처리는 컨설팅 범위가 아님

2. RP 이해 (2/3) – 연결구조

- ✓ RP(Route Planning) 서비스는 000 시스템의 핵심 서비스 자원이며, 기존의 투자와 경험을 보호하여야 함
- ✓ 차세대 000 시스템에서 RP서비스의 위상은 그대로 유지될 것이며, 기존의 불안한 요소(FastTP)를 제거하여야 함
- ✓ RP 서비스는 Local, Remote에서 모두 접속이 가능하여야 하고, 000 서비스의 품질요건을 충족해야 함





RP(route planning)는 경로탐색을 제공하는 서버모듈로써, 아키텍처 관점에서 특수한 기능을 수행하는 전문가(specialist)이다. 따라서, 타 서비스들이 쉽게 접근할 수 있도록 하는 것이 중요하다.

현행 RP 인터페이스는 TCP Socket을 이용하여 블럭메시지(Fixed size)를 전송하여주는 FastTP를 이용한다. 따라서 RP 클라이언트는 FastTP 클라이언트를 이용하여 접근하여야 한다.

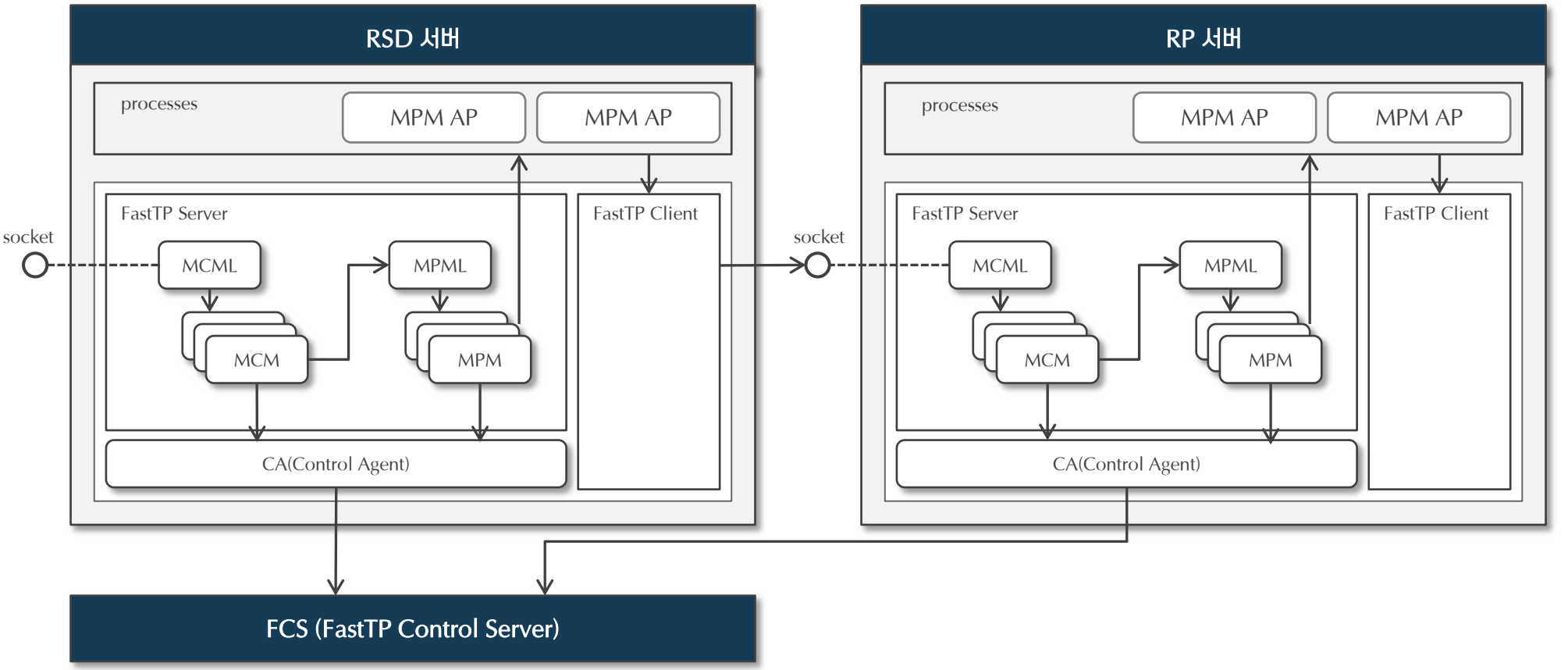
RESTful SOA를 위해서 RP를 서비스화하여야 한다. 즉, 다른 서비스 들이 쉽게 접근할 수 있는 서비스로 만들어야 한다.

RP 서비스는 Local/Remote에 대한 접근을 모두 지원하여야 한다. Local 접근일 경우, 성능/가용성/안정성이 중요하며, Remote접근일 경우, 오픈 기반 서비스 방식으로 접근하는 것이 중요하다.

RESTful SOA를 구현하기 위해서는 REST 방식의 서비스 호출이 바람직하다. 사용할 메시지는 경로데이터는 바이너리로 메타데이터는 문자열로 처리하여 기존의 데이터를 그대로 유지하는 것이 바람직하다.

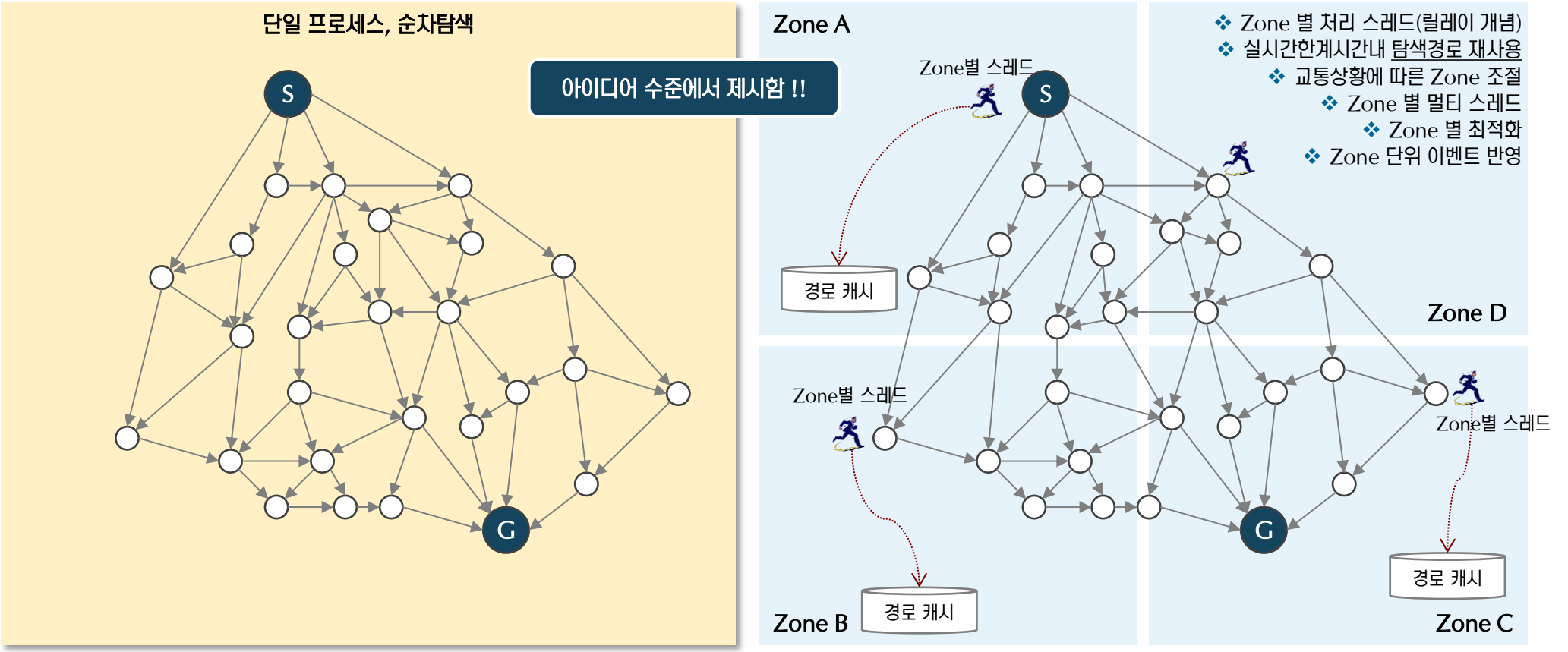
2. RP 이해 (3/3) – 요소기술: FastTP

- ✓ FastTP는 C로 개발하였으며, TCP 소켓을 통한 메시지 송수신을 지원하는 미들웨어임
- ✓ 개발사의 기술지원이 없으며, 운영 상의 지원 리스크가 상존하므로 반드시 대체 방안을 마련해야 함
- ✓ FastTP를 대체하는 방법은 여러 가지가 있으며, 아키텍처 컨설팅 과정에서 두 가지 방안을 탐색하기로 함



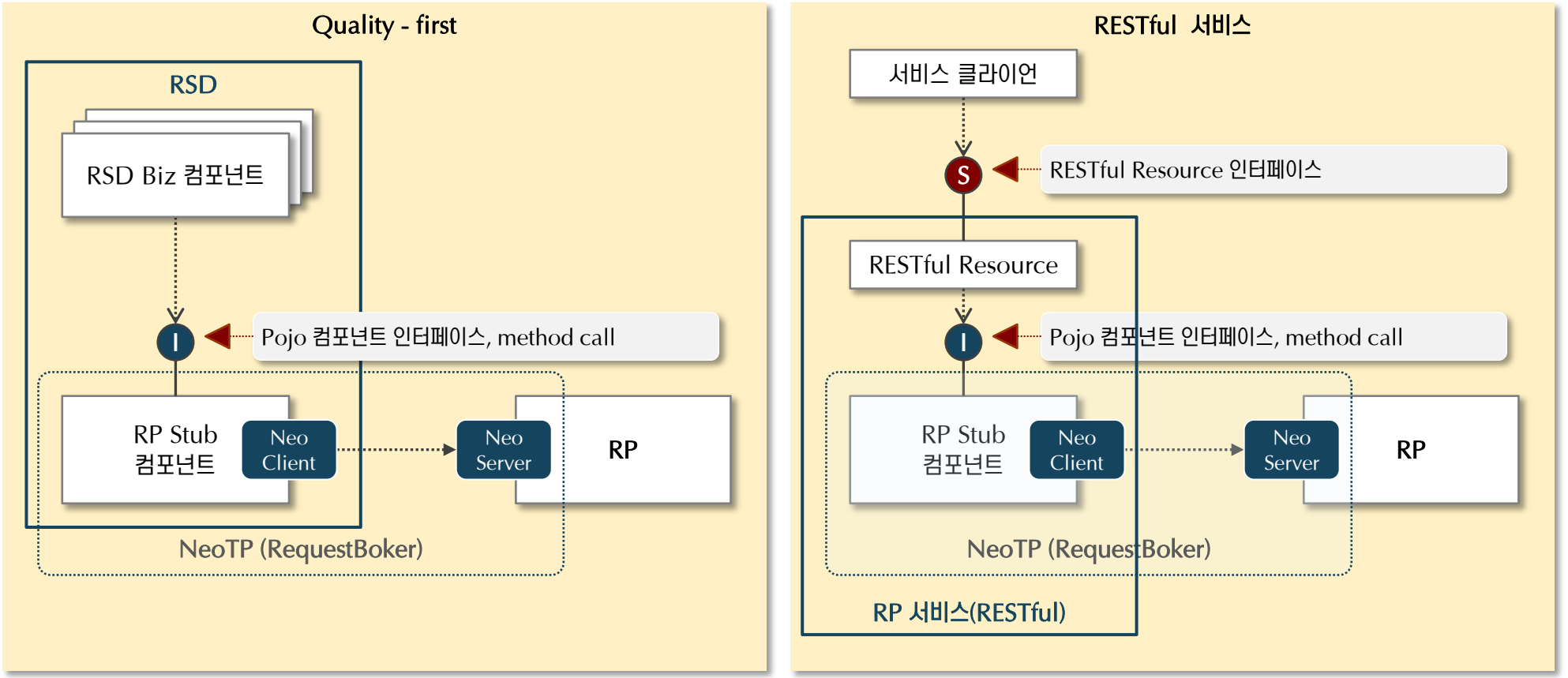
3. RP 탐색성능개선 아이디어

- ✓ AS-IS RP 길찾기 서비스는 단일 프로세스, 순차탐색 방식으로 성능을 최적화하여 사용하고 있음
- ✓ 영역(Zone)으로 나누고, 영역의 특성에 따른 최적화를 하며, 멀티 스레드를 이용하며, 탐색경로를 재사용하면,
- ✓ 기존의 알고리즘과 축적된 경험을 그대로 이어받으면서, 처리성능을 높일 수 있을 것으로 판단함



4. 설계안 두 가지

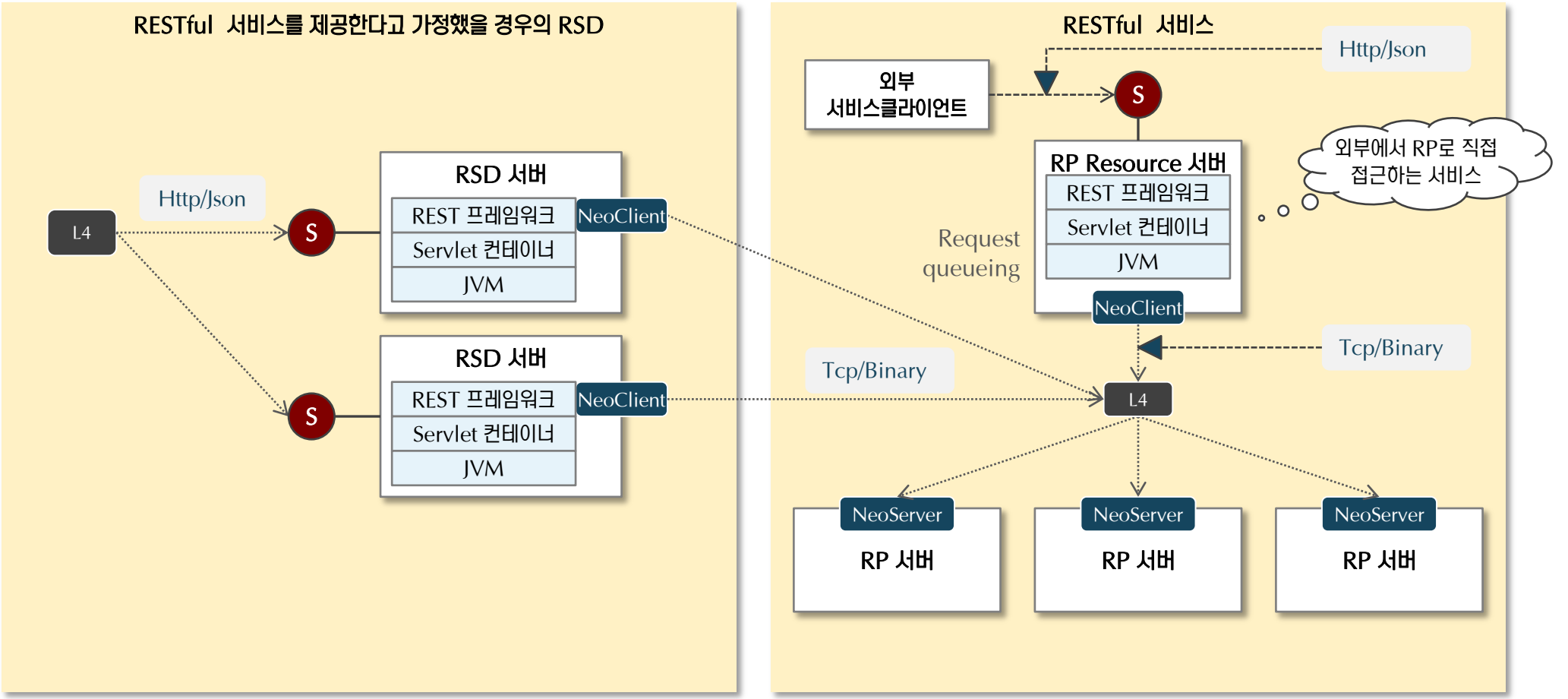
- ✓ RP(Route Planning) Local 접근은 NeoTP(가칭)를 하부에 깔고 있는 RP Stub 컴포넌트를 사용함
- ✓ RP Stub은 RP의 래퍼(Wrapper) 역할을 하며, RP의 API를 컴포넌트 인터페이스로 노출하는 역할을 함
- ✓ RP에 대한 RESTful 접근을 위해 RP 서버 외부에 RESTfule Resource로 래핑함



RESTful SOA 상세는 첨부참조 !!

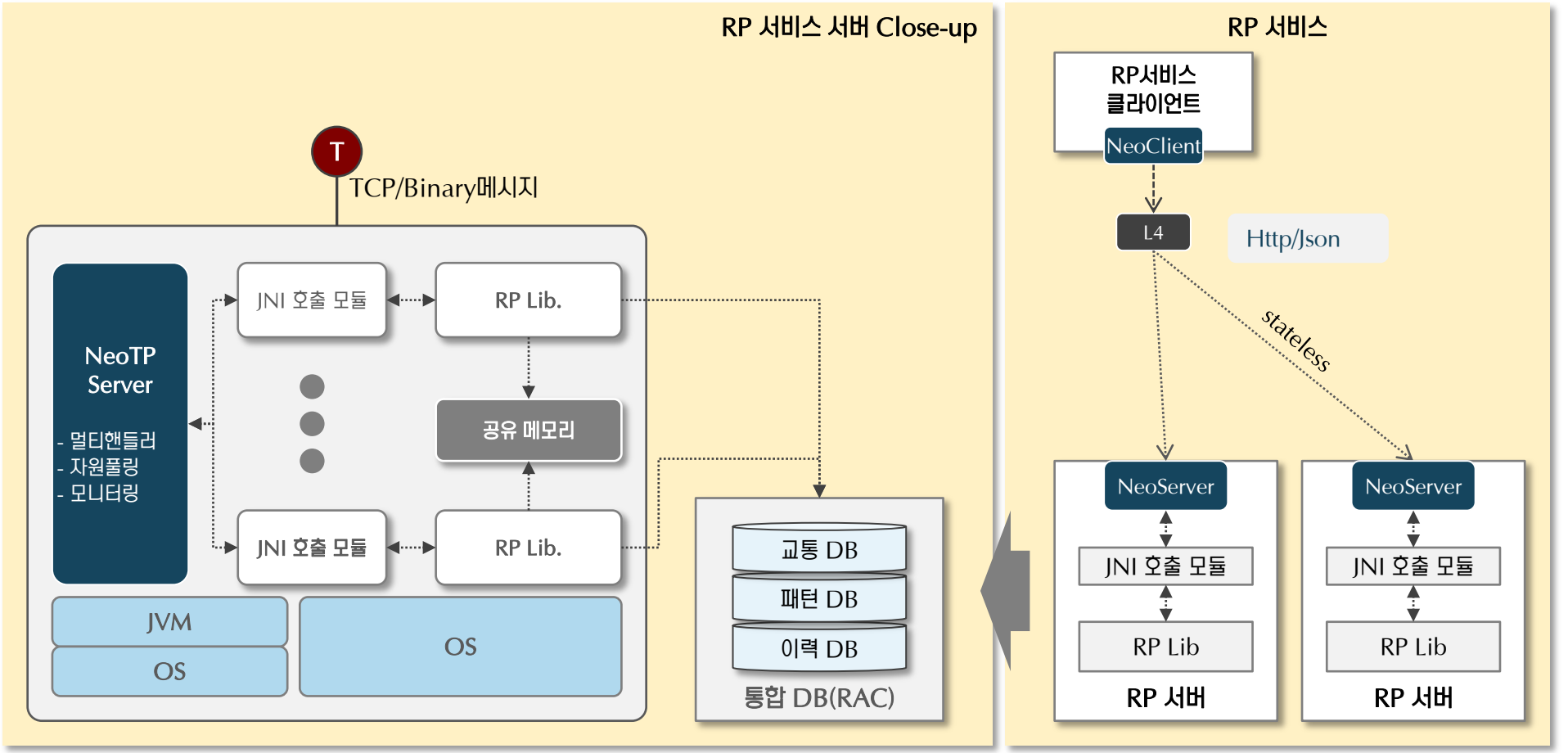
5. NeoTP 기반 서비스화 (1/2)

- ✓ [1안] 기존의 설계와 경험을 그대로 이어받으면서, Java로 FastTP를 새로 개발함(가칭:NeoTP)
- ✓ RSD에서 RP로의 연결 방식은 NeoTP 클라이언트로 대체되는 것 말고는 변화가 없음
- ✓ NeoTP는 Java로 구현되었으므로 C로 구현한 RP와는 JNI를 이용하여 연결함. RP는 라이브러리 형식으로 배포함



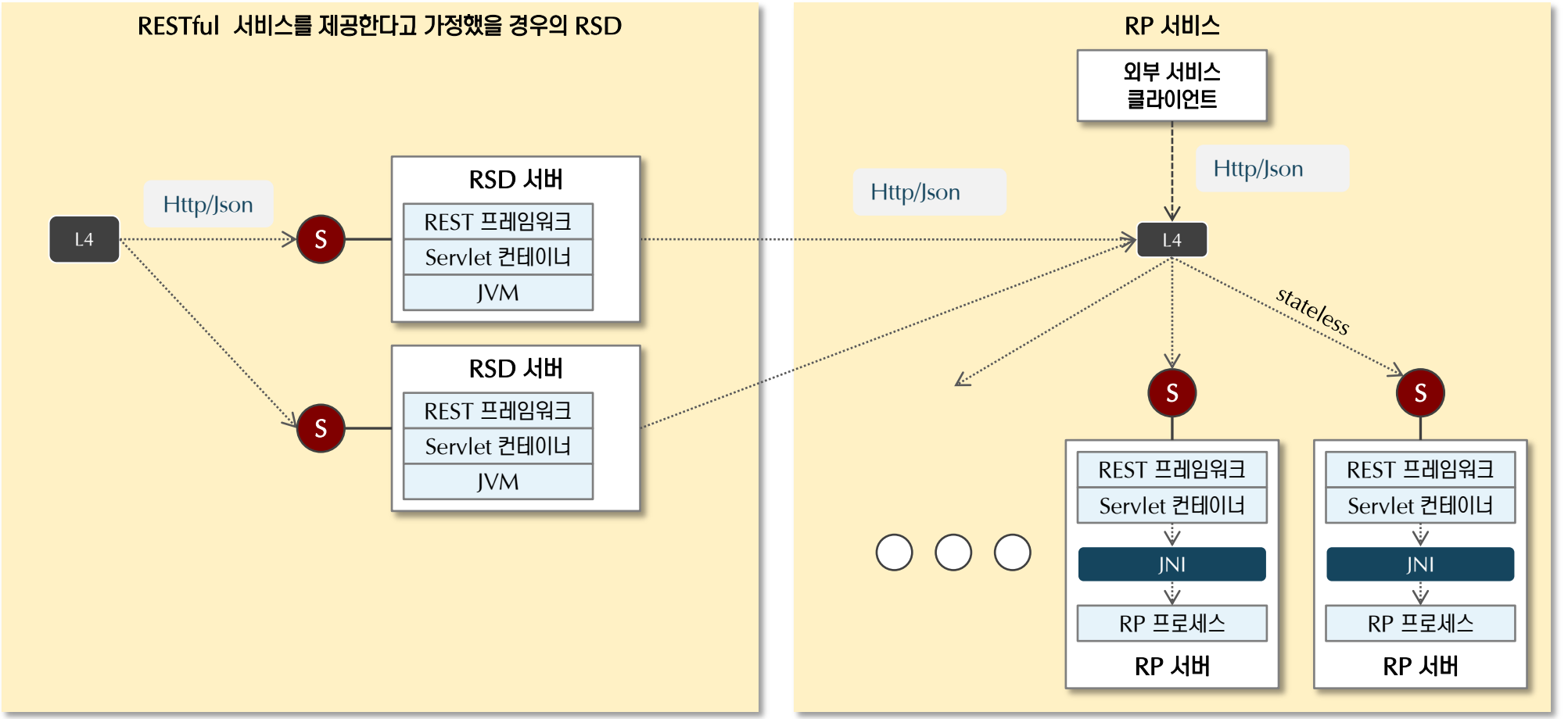
5. NeoTP 기반 서비스화 (2/2)

- ✓ NeoTP 서버는 멀티플랙싱(동시에 여러 건을 주고 받을 수 있음) 방식의 메시시 송수신 서버임
- ✓ 동시의 여러 건의 길찾기 요청을 처리할 수 있으며, 동시 최대처리에 대한 최적화 과정이 필요함
- ✓ NeoTP 서버는 모니터링 서버와 직접 연결되어 있으며, RP 서비스 제공 상황을 실시간으로 보고하도록 구성함



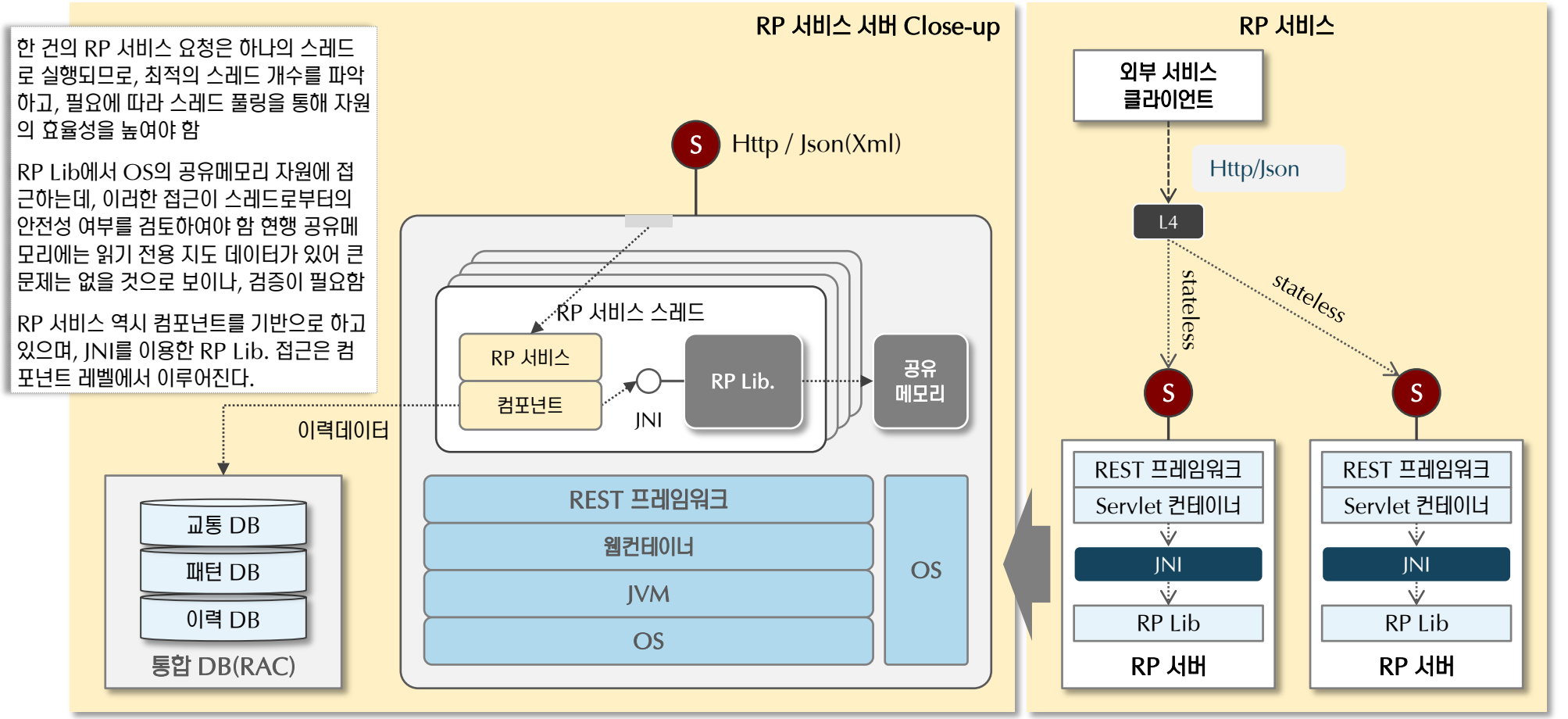
6. REST기반 서비스화 (1/2)

- ✓ [2안] RP를 완전한 RESTful 서비스로 설계함
- ✓ Local/Remote 어디에서도 동일한 방식으로, 그리고 OpenAPI 방식으로 RP 서비스에 접근할 수 있음
- ✓ RESTful 형태를 갖추더라도 메시지의 많은 부분은 기존의 Binary 포맷을 유지해야 함(예, TVAS 포맷)



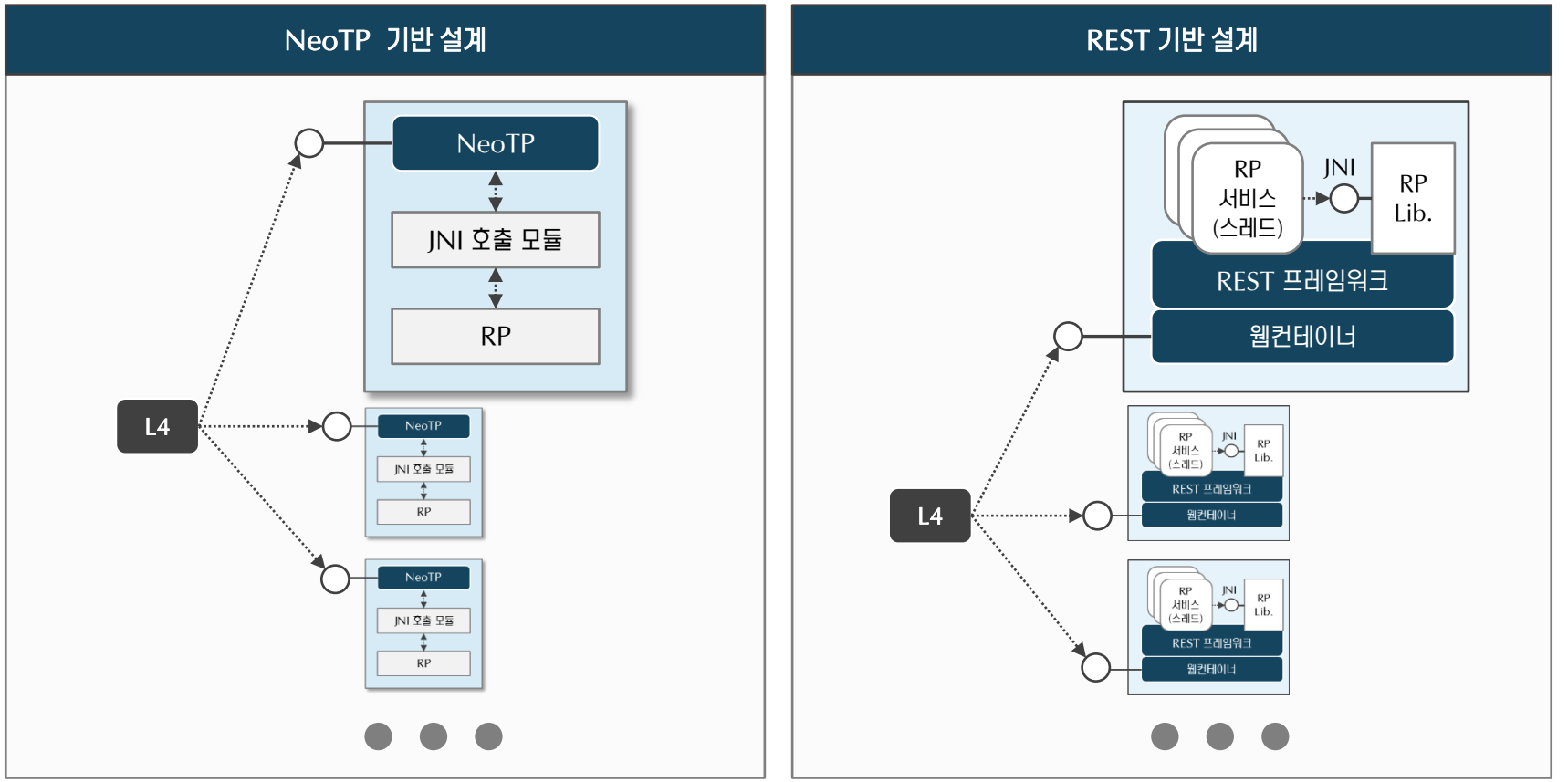
6. REST기반 서비스화 (2/2)

- ✓ RP 내에서 DB에 접근하는 일부 로직은 RP 컴포넌트 쪽으로 이동하고, RP는 탐색에만 집중하도록 함
- ✓ 메시지 포맷은 Json이나 Xml 중에서 택일 또는 두 가지 모두 발행(publishing)할 수 있도록 함
- ✓ RP 라이브러리에서 공유메모리 접근은 안정적이지만, 전역변수를 사용하는 부분은 검증이 필요함



7. 시스템 구성

- ✓ 어느 설계안을 선택하든 시스템 구성 상에는 변화가 없음, L4를 기반으로 부하균형을 실현함
- ✓ NeoTP 기반 설계의 경우, 서비스 요청을 TCP/Binary 전문 기반으로 받아서 처리함
- ✓ REST 기반 설계의 경우, RP 라이브러리는 Java의 스레드 안에서 실행되므로 Java에서 스레드 자원을 관리함



8. 설계안 비교분석

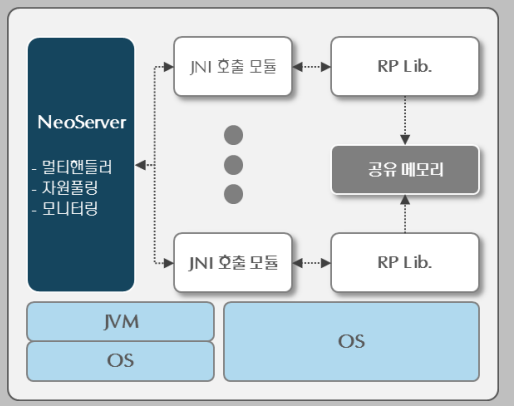
- ✓ 현재 선택할 수 있는 개선 방안은 RP 서비스화는 NeoTP 기반과 RESTful 서비스 두 가지가 있음
- ✓ NeoTP 기반은 매우 익숙한 구조이며, 많은 경험이 축적된 분야이고, 자체 개발 소스이므로 어떤 상황이든 대처할 수 있음
- ✓ REST 기반은 표준 기반이며, 대부분의 웹/Java 개발자에게 익숙한 기술이며, 오픈소스 프레임워크를 사용함

	NeoTP 기반 서비스화	RESTful 서비스	비고
서비스화 개요	00M&C 버전의 통신프레임워크 NeoTP를 사용함	REST 표준을 이용한 Open API 구성	NeoTP 기반일 경우, Open API를 위한 인프라가 별도 필요함
통신프로토콜	TCP Socket 사용	HTTP 사용 JAX-RS(Java API for RESTful Web Service)	
메시지 포맷	고정길이 바이너리 전문	JSON 또는 Xml, binary data 포함	
개발 용이성	클라이언트 측에 NeoTP 클라이언트를 감싸는 Stub을 컴포넌트 형태로 제공	클라이언트에서 RESTful 자원으로 직접 접근하거나, 컴포넌트로 요청을 Wrapping하여 간접 접근함	동등한 수준
RP 접근	Local: 직접 접근 Remote: 직접 TCP로 접근하기 어려움	Local: 직접 접근 Remote: 직접 접근	NeoTP의 경우, Remote 접근에 제약이 있음
확장성	NeoTP가 자체 개발이므로 확장에 어려움 없으나 모든 것을 직접 개발하여야 함	오픈소스 서비스 발행 프레임워크의 확장에 자연스럽게 동참하므로, 확장이 용이함	동등한 수준
유지보수성	자체 개발 소스이며, 소스코드가 1만 라인 내외여서 유지보수에 큰 어려움이 없음	오픈소스 서비스 발행 프레임워크에 대한 학습이 필요할 뿐, 특별한 어려움은 없음	동등한 수준
수행성능	NeoTP 방식보다는 빠르다고 할 수 있다. 하지만, 성능을 전체 자원의 효율적인 사용이라는 관점에서 볼 때, 많은 경험이 누적된 REST 방식이 유리할 수 있음. NeoTP 다양한 방식으로 자유롭게 튜닝이 가능함		관점에 따라 다름
요약	자체 개발이며 필요한 최소한의 기능을 탑재하고 있음, 필요할 경우 자유롭게 확장이 가능함	표준 기반으로 누가 오더라도 쉽게 유지관리가 가능하며, 향후 타 시스템으로도 확장될 것임	표준/비표준이 가장 큰 차이임

9. 요소기술 식별

- ✓ NeoTP 기반 설계는 Java와 Java를 이용한 TCP 소켓 프로그래밍 기술 만 필요함
- ✓ REST 기반 설계는 REST 프레임워크, 웹 컨테이너, Spring 프레임워크(DI) 등이 필요함
- ✓ NeoTP의 경우, 그 자체가 통신 프레임워크가 되므로 유사 개발경험이 충분한 팀이 개발을 해야 함

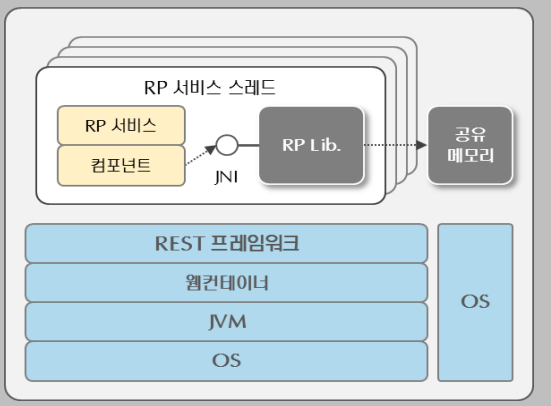
NeoTP 기반 서비스화



- ✓ 개발언어 - Java 1.6
- ✓ JVM - JRE 1.6
- ✓ NeoTP Server- 자체 개발(1만라인 내외로 3M/M 예상)
- ✓ RP 서비스 – AS-IS C 프로그램

*** NeoTP 서버는 모니터링 서버와 연계하여야 함*

RESTful 서비스

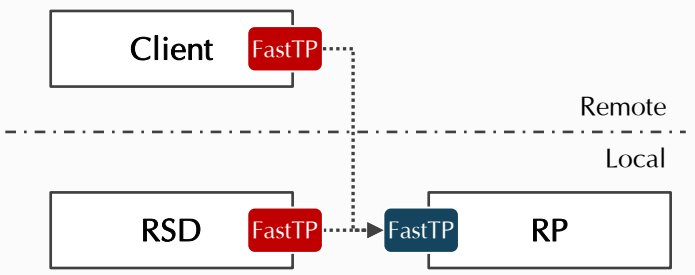


- ✓ 개발언어: Java 1.6
- ✓ JVM: JRE 1.6
- ✓ 웹컨테이너 – Tomcat 또는 상용 도구
- ✓ REST 프레임워크 – Jersey, Restlet 또는 상용도구
- ✓ RP 서비스, RP 컴포넌트(as a wrapper) – 자체 개발
- ✓ POJO 컨테이너 – Spring 3.1
- ✓ JNI 연계 – 자체 개발

10. 요약

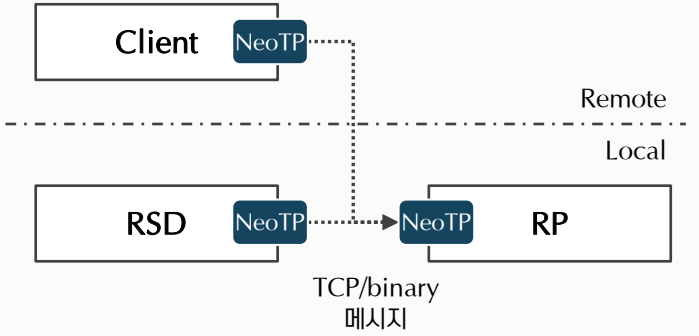
- ✓ 현행 서비스의 QoS를 해치지 않으면서 보다 나은 수준의 서비스를 제공할 수 있어야 함
- ✓ 향후 RP 서비스 자체 개선 시에도 유연하게 대응할 수 있어야 함
- ✓ Open API라는 시장의 요구에 부응하여야 하며, 엔지니어에게 편리하며, 또한 검증된 기술이어야 함

AS-IS RP 구조



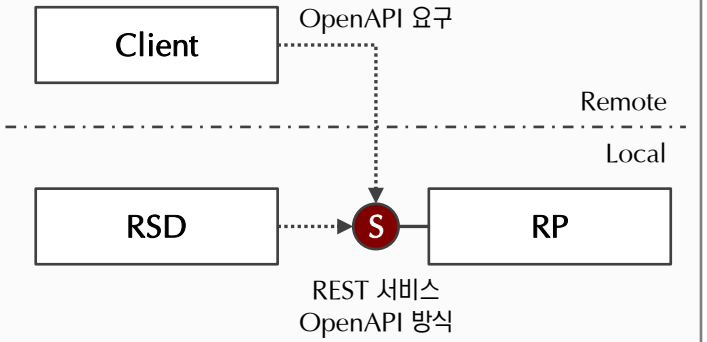
- ✓ 블랙박스(FastTP)로 인한 운영의 어려움과 리스크
- ✓ 서비스 운영 자원조절이 유연하지 않음
- ✓ FastTP 내부 상태 모니터링이 어려움
- ✓ C 개발자를 구하기 어려움
- ✓ FastTP는 새로운 것으로 대체되어야 함

TO-BE RP 접근구조 (NeoTP 기반)



- ✓ FastTP와 동일한 기술구조를 가짐
- ✓ Java로 개발하며, 성능과 모니터링에 최적화되어야 함
- ✓ FastTP 운영 경험을 그대로 NeoTP로 담을 수 있음
- ✓ 개발은 00M&C에서 리드하여야 하며, 솔루션은 배제함
- ✓ 소스코드에 대한 제어 및 소유권 이 매우 중요함

TO-BE RP 접근구조(REST 기반)



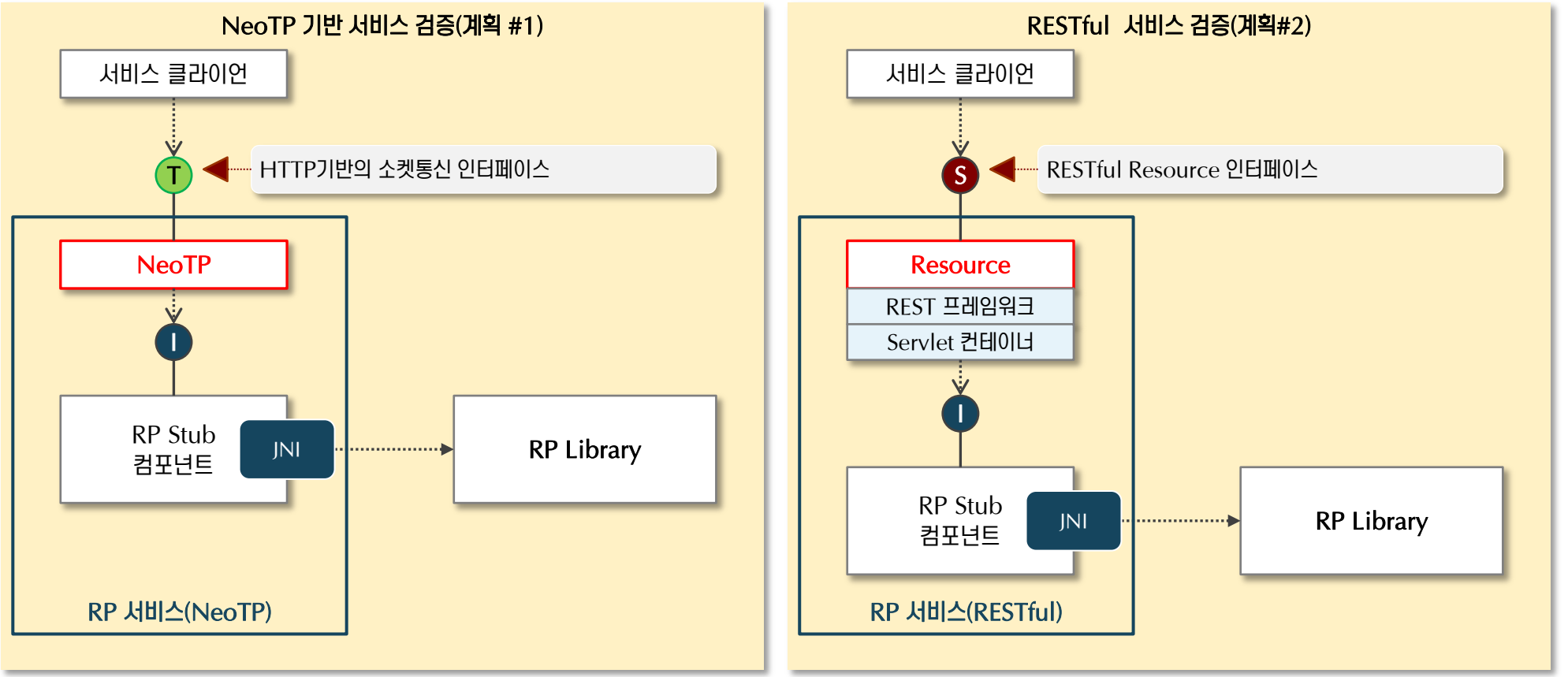
- ✓ Open API 제공
- ✓ JNI를 통한 C 라이브러리 접근은 검증된 방법임
- ✓ Java 기반 오픈소스 활용으로 엔지니어 확보 용이함
- ✓ 모니터링 장치 추가 용이함(서비스 요청의 필터, 사전,사후 처리)
- ✓ REST 기술은Http를 기반으로 하므로 검증된 기술임

RP 서비스화 RI - 목차

RI:Reference Implementation

1. RI 계획 – RP 서비스화

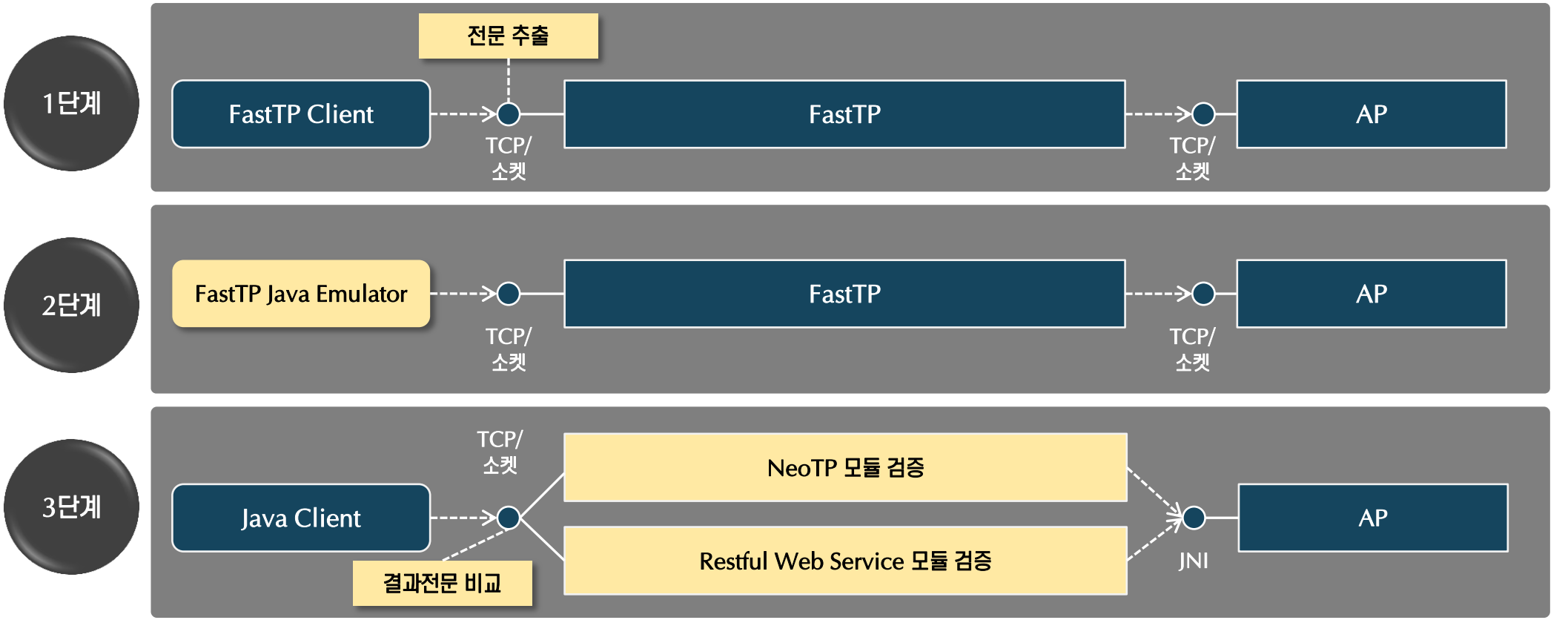
- ✓ 기존 FastTP 기반의 RP 서비스를 자바 기반의 미들웨어(Middleware)로 변경하는 과정에 대한 RI를 진행함
- ✓ 가장 빈번히 발생하는 RP 서비스를 추출하여 해당 서비스를 자바기반으로 변경하며, 두 가지 방향을 검증
- ✓ 계획 1 : FastTP를 대체하는 새로운 자바 기반의 미들웨어 (NeoTP)를 도입하고 구조를 검증함
- ✓ 계획 2 : FastTP를 REST기반 프레임워크로 교체하고 구조를 검증함



2. RP 서비스화 검증 방안

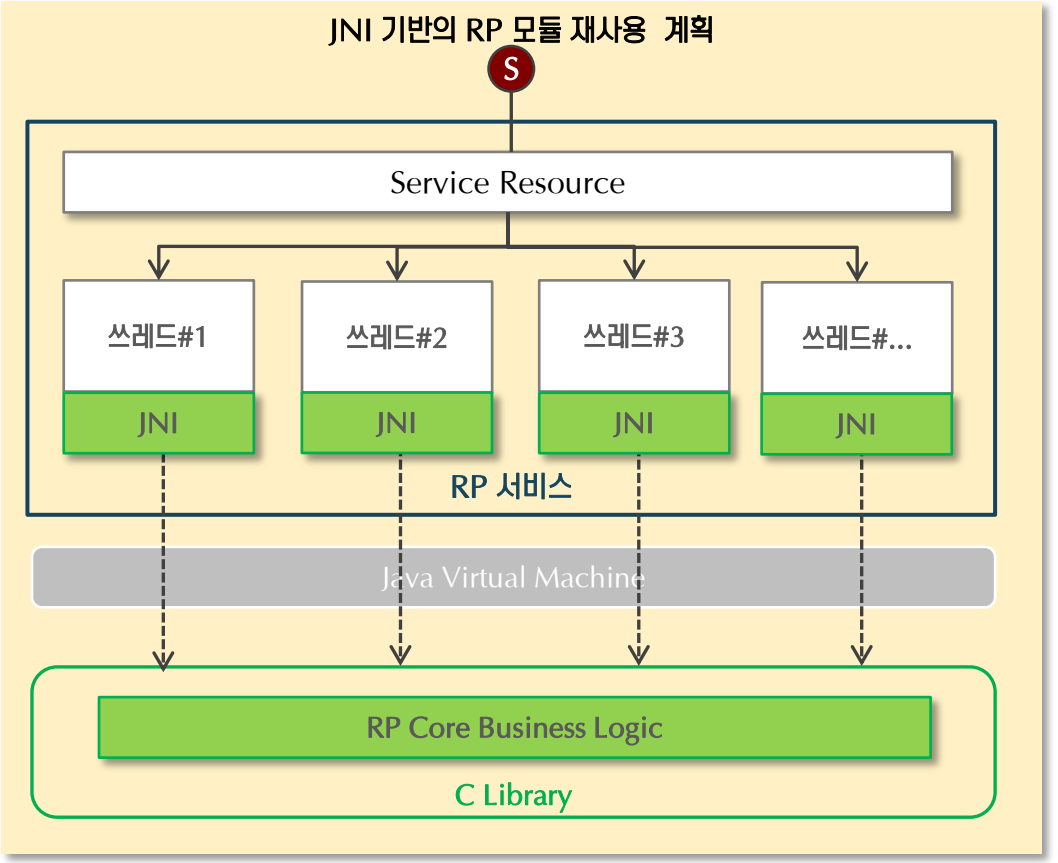
✓ 3단계 접근 방식을 통한 RI 검증

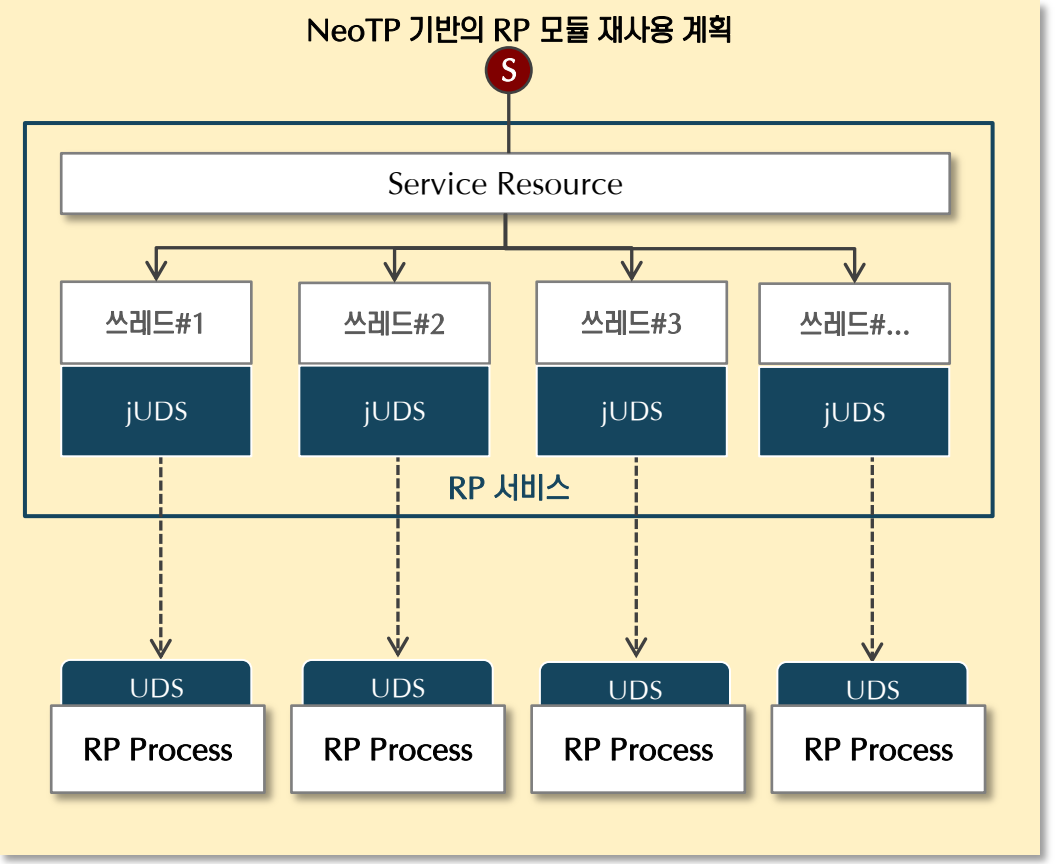
- ✓ 1단계 – FastTP 클라이언트를 이용한 기존 시스템 전문 검증 및 전문 추출
- ✓ 2단계 – FastTP 클라이언트의 Java 버전 개발 및 기본 전문 검증
- ✓ 3단계 – FastTP의 교체 및 JNI를 통한 RP 모듈 호출 및 전문 검증



3. RI 계획 – RP 모듈 연계

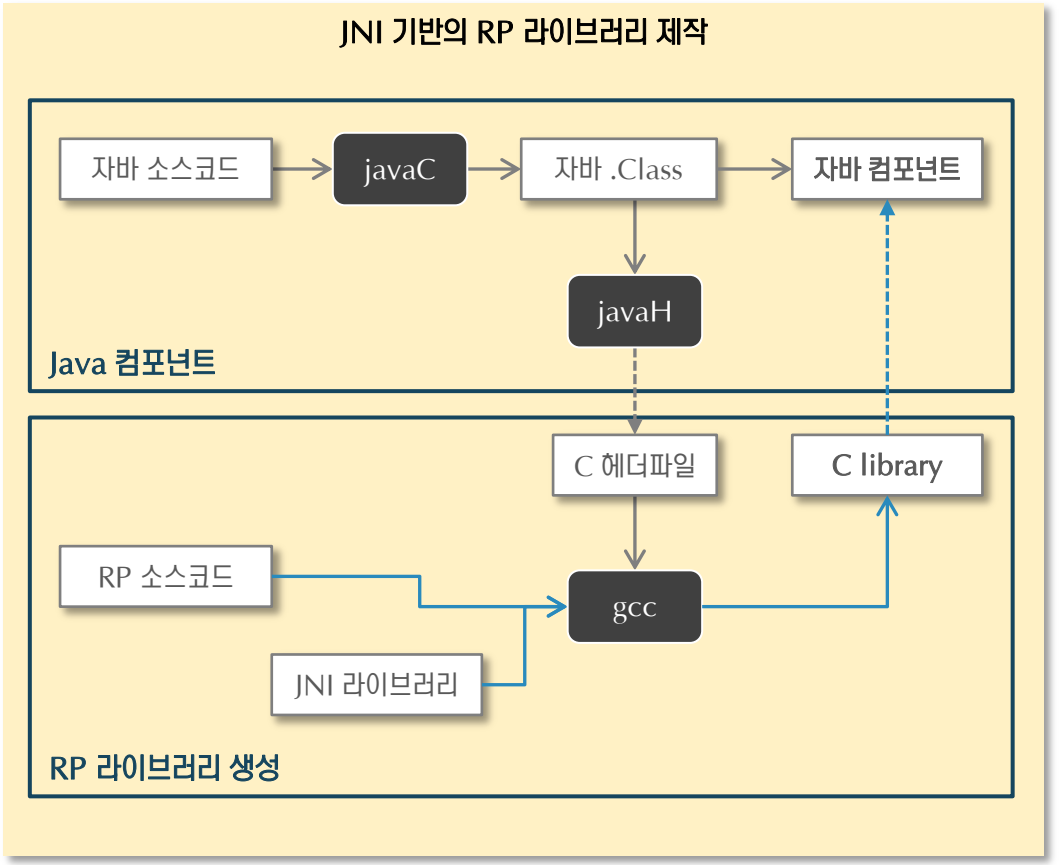
- ✓ 재개발 시 C 기반의 RP 모듈과 자바 모듈의 연계를 위한 최적의 방법을 탐색
- ✓ 계획 1 : UDS기반의 C 모듈을 라이브러리로 변경하고 JNI를 통해 접근하는 방법을 검증함
- ✓ 계획 2 : UDS기반의 C 모듈을 자바 UDS통신 기반으로 변경하고 방법을 검증함

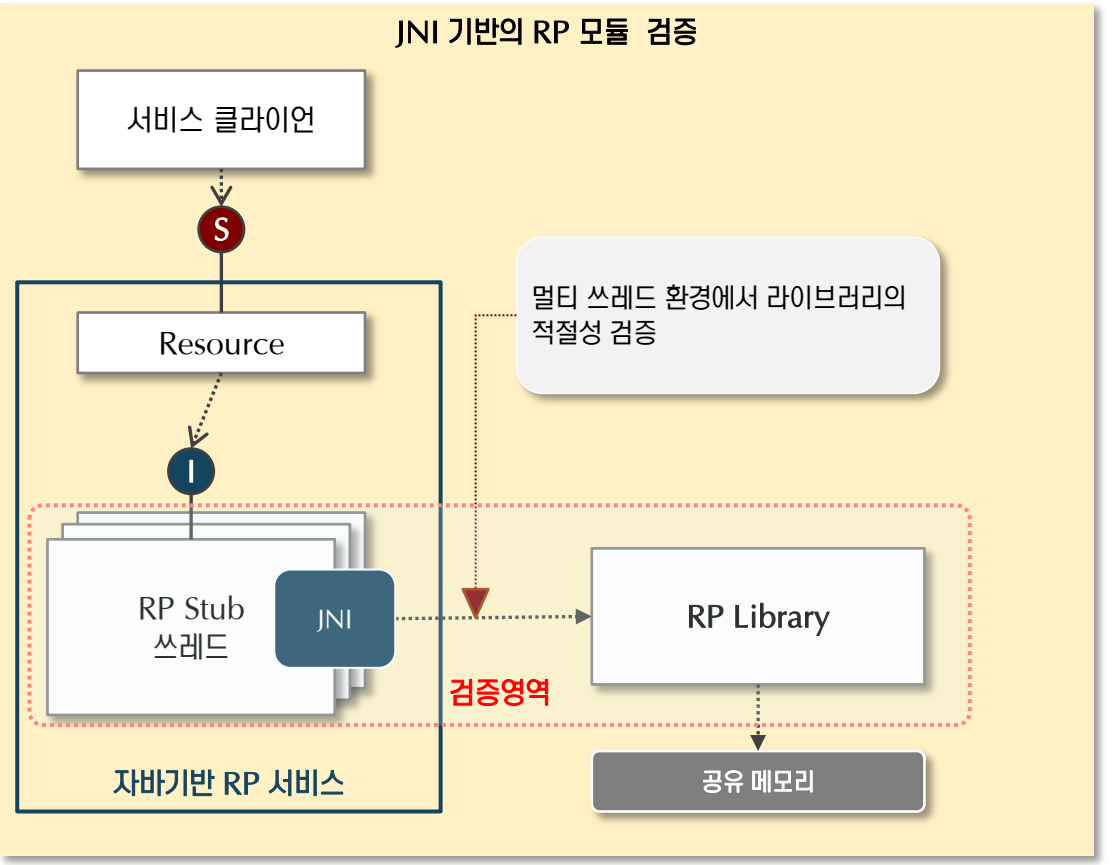




4. RP 모듈 연계 검증 방안 – I

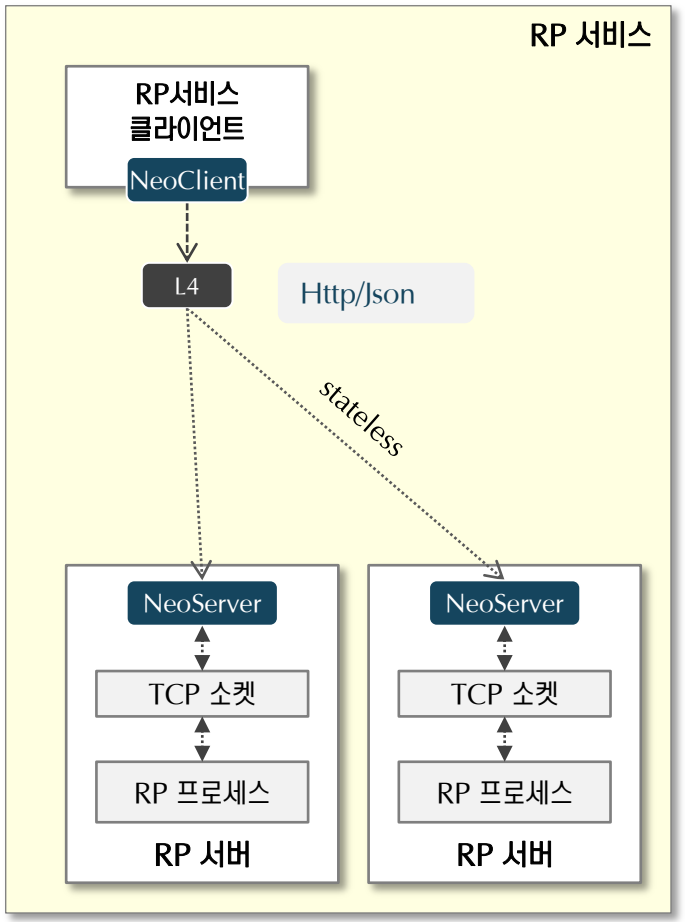
- ✓ 독립된 프로세스로 구성된 AP 를 라이브러리 형태로 변환
- ✓ 자바의 네이티브(Native) 인터페이스인 JNI을 통해 라이브러리로 변경된 AP에 접근하여 기존 RP 로직 수행
- ✓ 멀티쓰레드(Multi-Thread) 환경에서 RP 라이브러리의 올바른 동작 검증

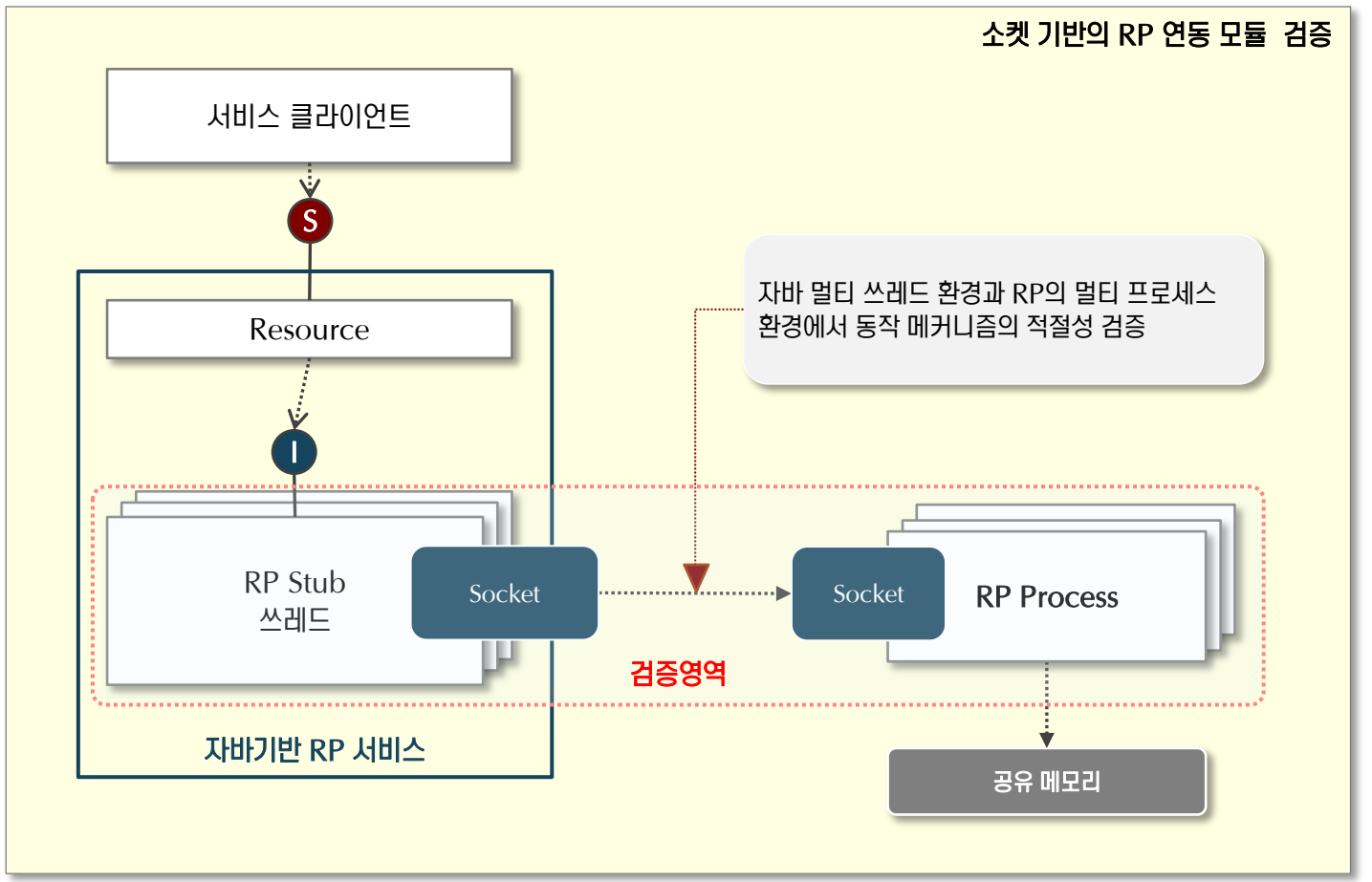




4. RP 모듈 연계 검증 방안 – II

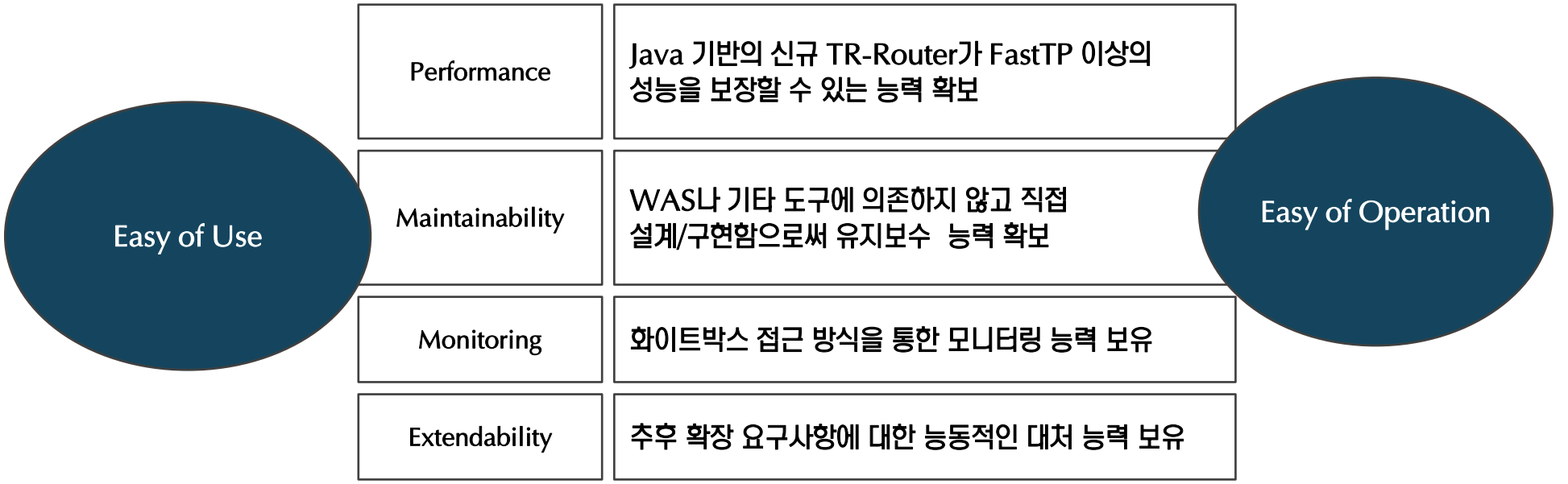
- ✓ PR 모듈을 독립 프로세스로 구성한 후 소켓통신을 통해 RP와 연계하는 방안에 대한 검증을 수행함
- ✓ 기존의 UDS 메커니즘과 유사하여, 기존 로직을 수정하지 않고 UDS를 통해 통신하는 방법으로 검증
- ✓ 자바의 멀티쓰레드(Multithread)와 RP의 멀티 프로세스 환경에서 적절한 동작 메커니즘을 검증





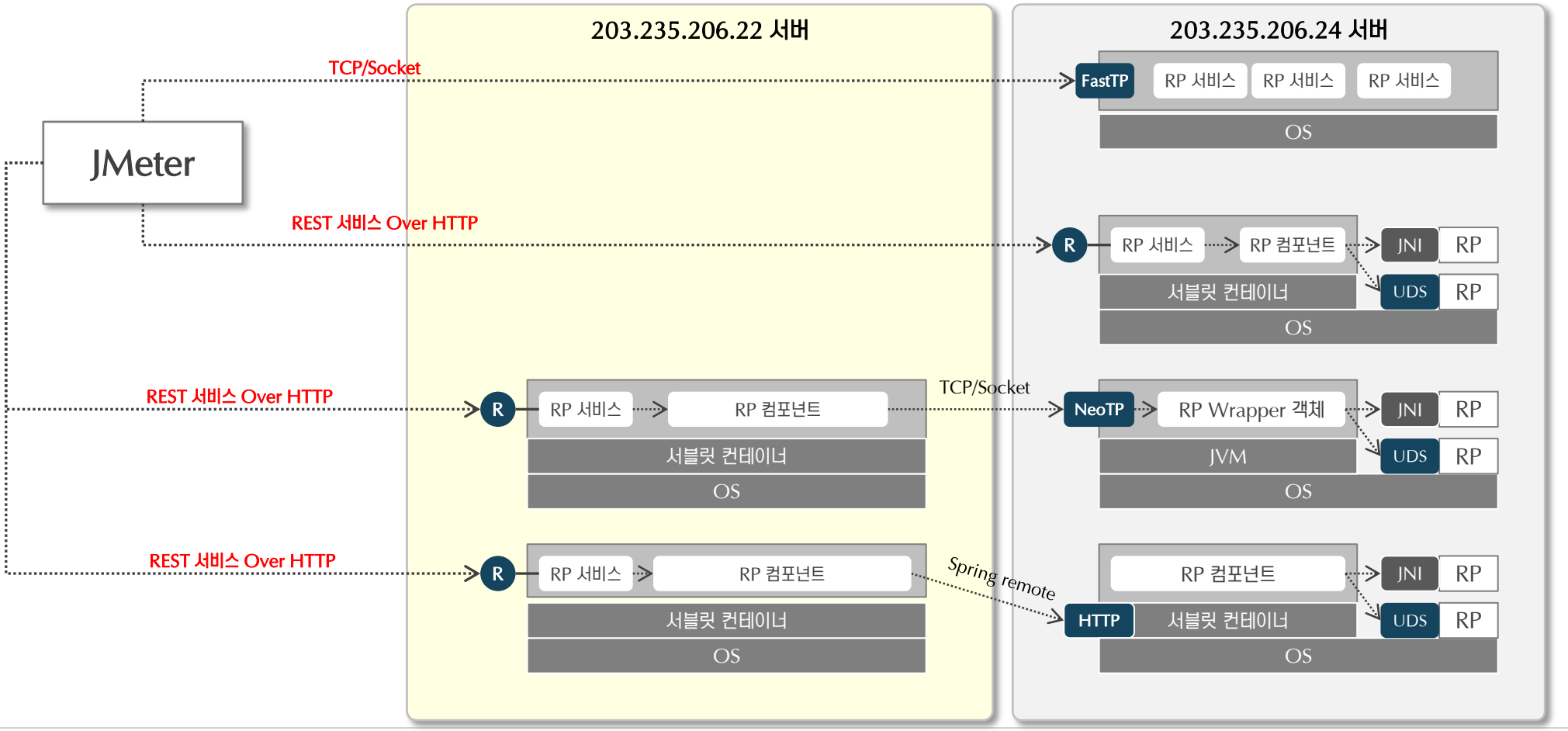
5. RI 기대 효과

- ✓ FastTP의 교체 및 개선에 대한 다양한 방안의 적절성 검증
- ✓ 재개발로 구축되는 자바기반 시스템과 기존 C로 모듈간의 연동 방법에 검증
- ✓ 최종적으로 도출된 개선방향과 변화에 대한 선 검증을 통해 아키텍처의 적정성을 판단



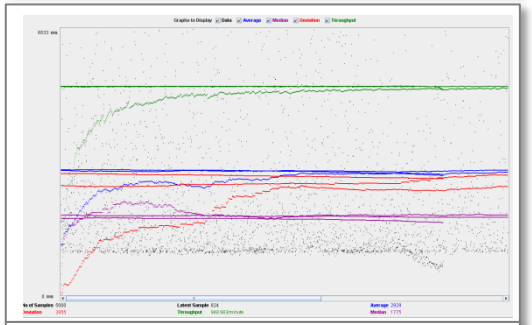
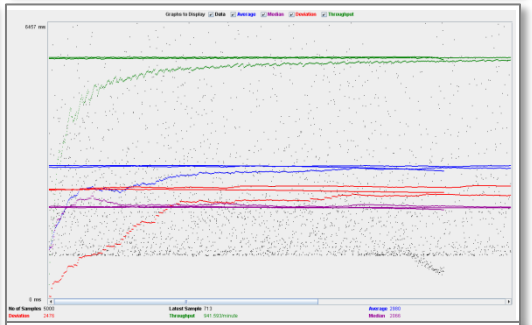
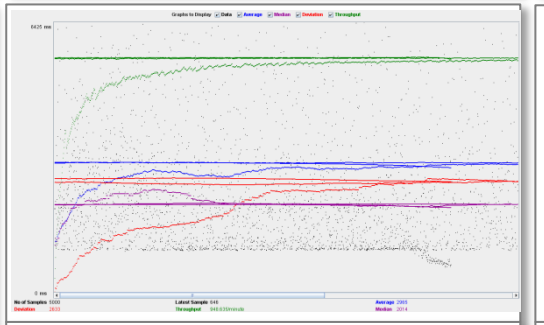
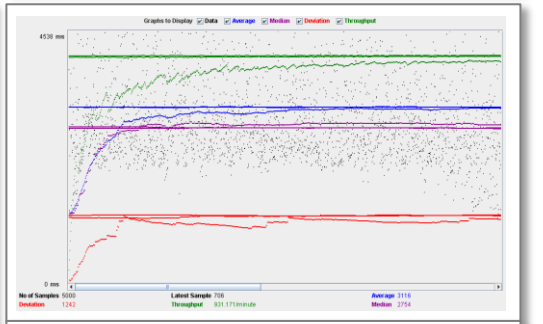
5. RI 결과

- ✓ 자바 기반의 성능 테스트 도구인 jMeter를 이용해 기존 방식과 새로운 방식에 대한 부하 측정
- ✓ 50 User로 각 100회의 부하를 가했을 때 결과를 측정 (ramp-up period: 1sec)
- ✓ RP001만을 대상으로 수행하였으며 FastTP와의 상대 비교를 위한 목적이므로 반응 속도에 대한 의미는 없음



5. RI 결과

- ✓ 현재 RP 서비스의 성능은 대부분 경로 탐색의 성능과 직결됨을 알 수 있음
- ✓ WAS + REST + Remoting 나 NeoTP 구간에서 성능에 대한 영향은 거의 없는 것으로 파악됨
- ✓ JNI 테스트를 통해 RP를 라이브러리 변경 시에 성능 영향도를 분석해야 함

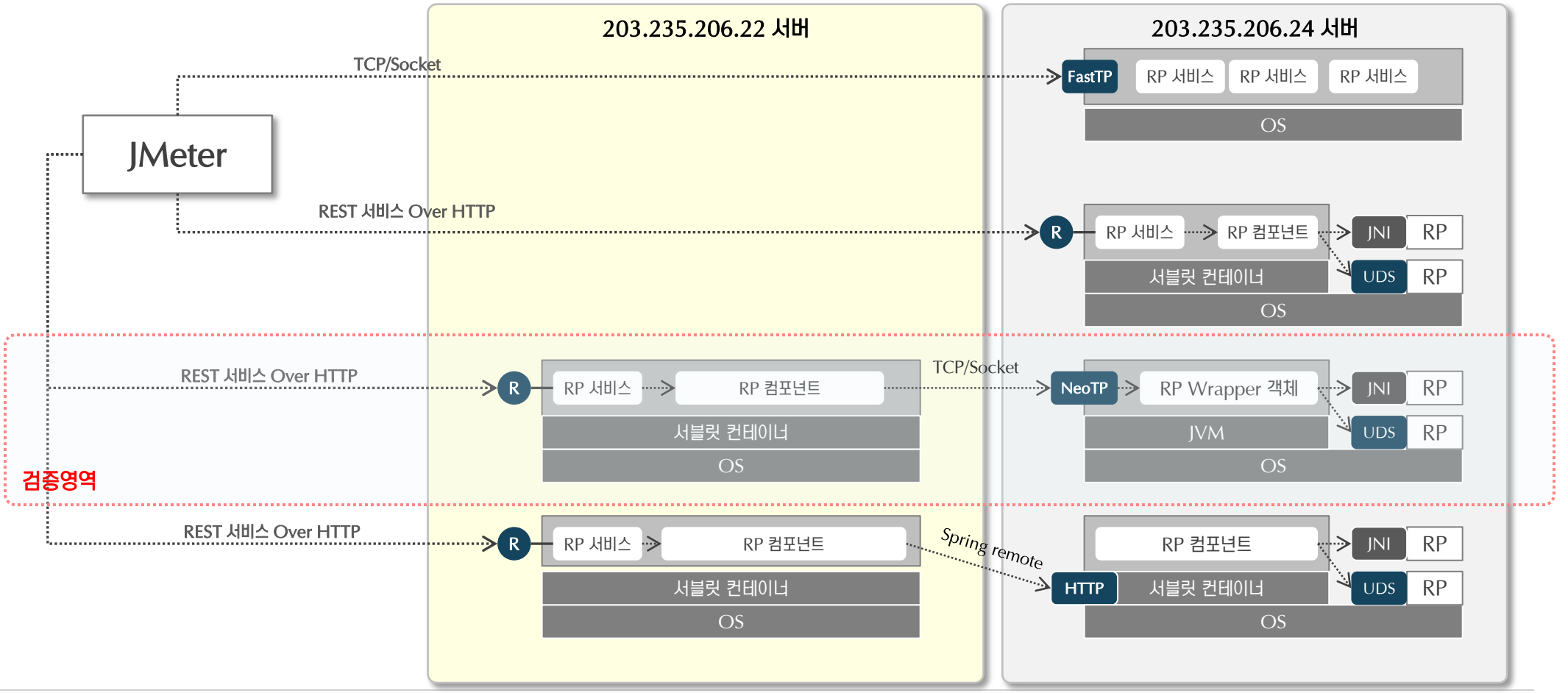
FastTp	Rest	Rest + RMI	Rest + NeoTp
			
✓ Throughput : 948.983/minutes(15tps) ✓ Average : 2929ms ✓ Deviation : 2855ms ✓ Median : 1775ms	✓ Throughput : 941.593/minutes(15tps) ✓ Average : 2980ms ✓ Deviation : 2470ms ✓ Median : 2066ms	✓ Throughput : 948.635/min (15tps) ✓ Average : 2965ms ✓ Deviation : 2633ms ✓ Median : 2014ms	✓ Throughput : 931.171/min (15tps) ✓ Average : 3116ms ✓ Deviation : 1242ms ✓ Median : 2754ms

5. RI 검토

- ✓ Filter 이용한 로깅 기법 소개
- ✓ Reflection을 이용한 동적 필드 매핑 기법 소개
- ✓ Restful Web Service를 이용한 서비스 표출 기법 소개
- ✓ JAXB, JSON을 이용한 객체 전문 변환 기법 소개
- ✓ Spring Remoting을 이용한 원격 메소드 호출 기법 소개
- ✓ Spring을 이용한 Dependency Injection 소개

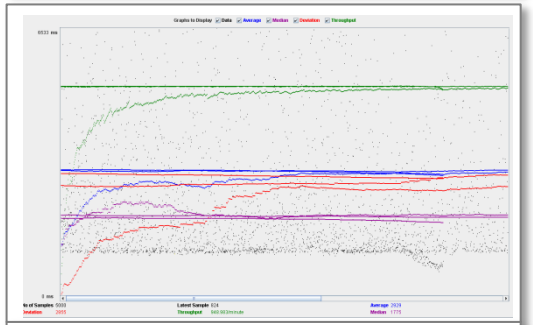
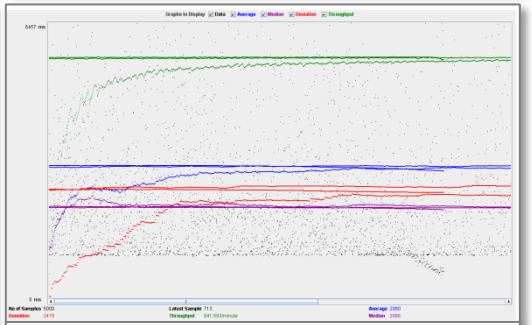
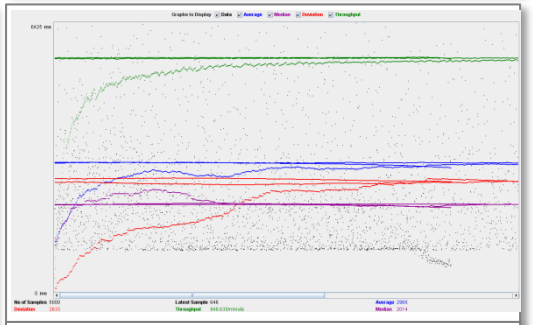
6. RI#1 시나리오

- ✓ 자바 기반의 성능 테스트 도구인 jMeter를 이용해 기존 방식과 새로운 방식에 대한 부하 측정
- ✓ 50 User로 각 100회의 부하를 가했을 때 결과를 측정 (ramp-up period: 1sec)
- ✓ RP001만을 대상으로 수행하였으며 FastTP와의 상대 비교를 위한 목적임으로 반응 속도에 대한 의미는 없음



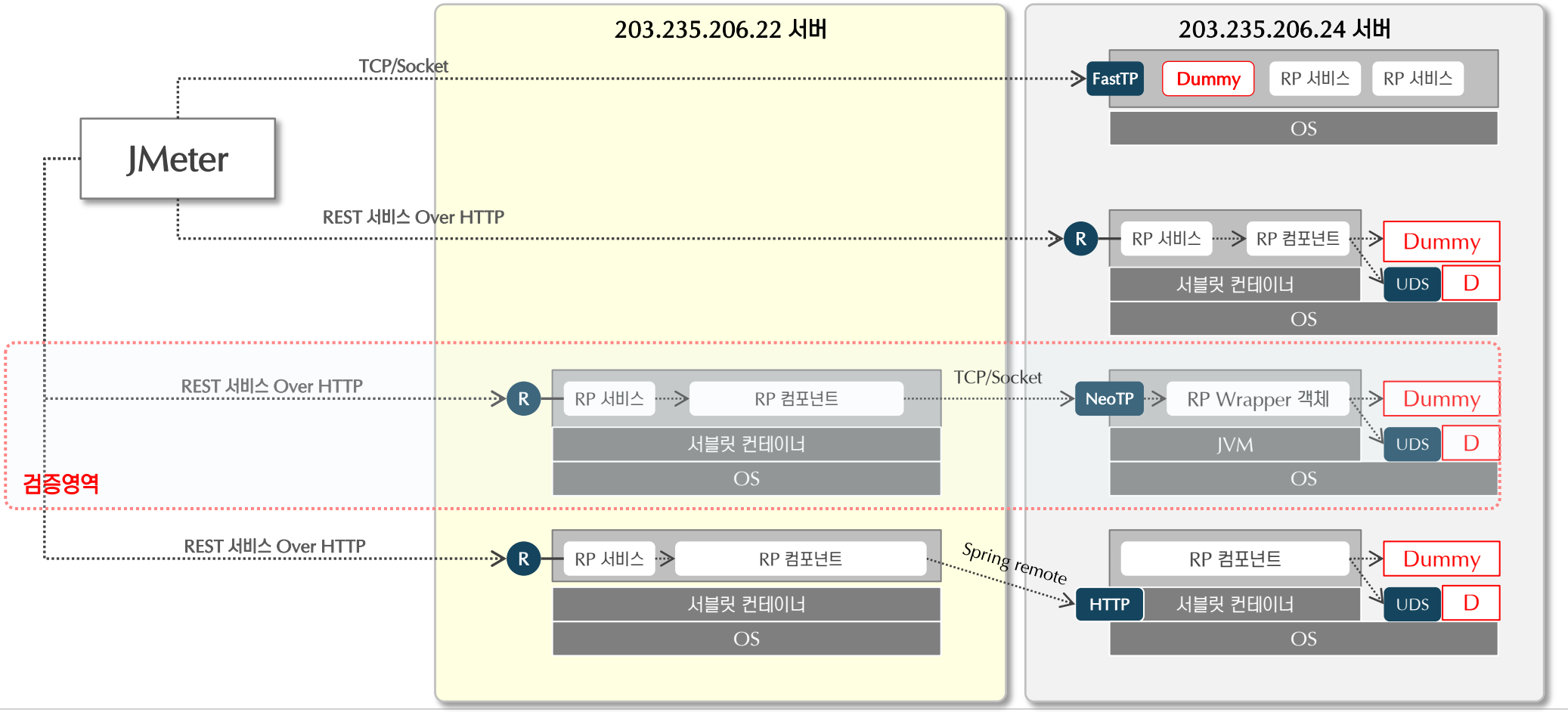
6. RI#1 결과

- ✓ 현재 RP 서비스의 성능은 대부분 경로 탐색의 성능과 직결됨을 알 수 있음
- ✓ WAS + REST + Remoting 나 NeoTP 구간에서 성능에 대한 영향은 거의 없는 것으로 파악됨
- ✓ JNI 테스트를 통해 RP를 라이브러리 변경 시에 성능 영향도를 분석해야 함

FastTp	Rest	Rest + RMI	Rest + NeoTp
			
✓ Throughput : 948.983/minutes(15tps) ✓ Average : 2929ms ✓ Deviation : 2855ms ✓ Median : 1775ms	✓ Throughput : 941.593/minutes(15tps) ✓ Average : 2980ms ✓ Deviation : 2470ms ✓ Median : 2066ms	✓ Throughput : 948.635/min (15tps) ✓ Average : 2965ms ✓ Deviation : 2633ms ✓ Median : 2014ms	✓ Throughput : 931.171/min (15tps) ✓ Average : 3116ms ✓ Deviation : 1242ms ✓ Median : 2754ms

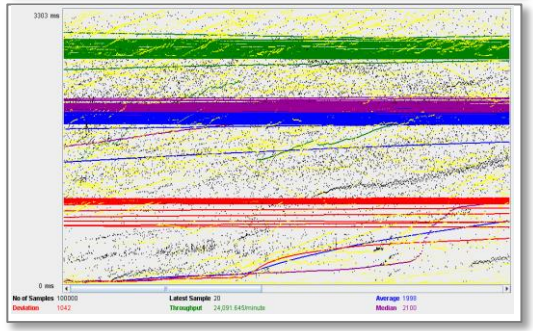
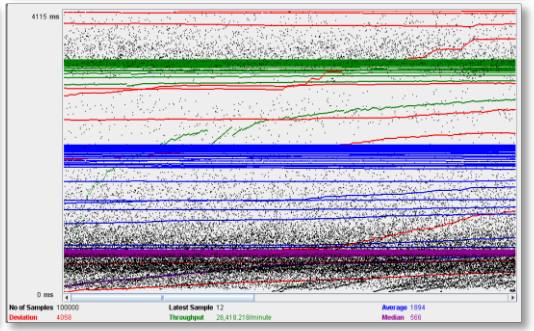
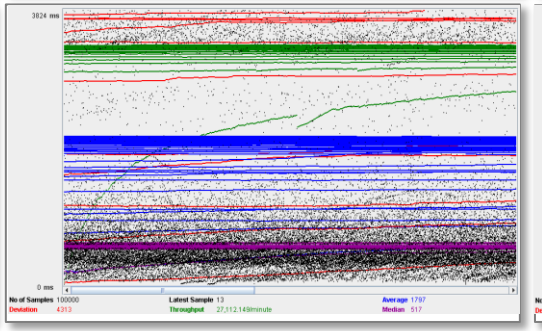
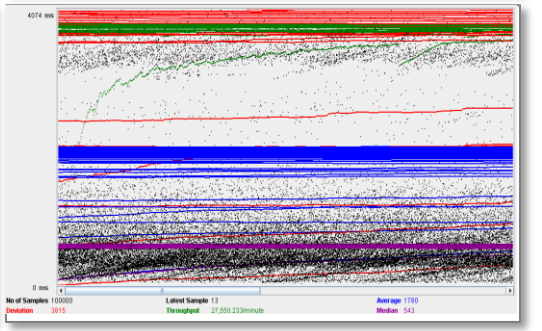
7. RI#2 시나리오

- ✓ RP 계산 로직을 수행하지 않고 제안 아키텍처의 상의 구간 성능을 탐색하기 위한 목적으로 수행
- ✓ FastTP의 경우도 Dummy로직을 수행하도록 변경하여 FastTP 전송 구간상의 성능을 측정
- ✓ 1000 User로 각 100번 씩 수행하면 더미 로직에서는 100회의 랜덤 수를 생성함



7. RI#2 결과

- ✓ FastTP로 직접 부하를 가할 경우 테스트환경에서 FastTP 프로세스가 더 이상 부하를 받지 못하는 현상을 보임
- ✓ WAS + DummyAp에 기반한 방식의 경우 모든 부하를 끝까지 받으며 400tps의 결과를 보임
- ✓ DummyAp에 접근하지 않고 자바 쓰레드에서 직접 랜덤 함수를 실행할 경우 대략 800~900tps 정도의 결과를 보임

FastTP	Rest	Rest + RMI	Rest + NeoTP
			
✓ Throughput : 24,091.645/min (400tps) ✓ Average : 1998ms ✓ Deviation : 1042ms ✓ Median : 2100ms ✓ 2~3회 정도 hanging ✓ 다수의 에러 발생으로 정확한 tps측정이 어려움	✓ Throughput : 26,418.218/min (440tps) ✓ Average : 1894ms ✓ Deviation : 4058ms ✓ Median : 566ms ✓ 자바 랜덤 로직을 사용할 경우 약 50,000/min (830tps) 를 나타냄	✓ Throughput : 27,112.149/min (451tps) ✓ Average : 1797ms ✓ Deviation : 4313ms ✓ Median : 517ms ✓ 자바 랜덤 로직을 사용할 경우 약 50,000/min (830tps) 를 나타냄	✓ Throughput : 27,550.233/min (459tps) ✓ Average : 1780ms ✓ Deviation : 3815ms ✓ Median : 543ms ✓ 자바 랜덤 로직을 사용할 경우 약 59,000/min (980tps)를 나타냄