DEVIEW
2019

핀포인트는 어떻게 observability 를 강화했는가

구태진
Observability Platform
NAVER

CONTENTS

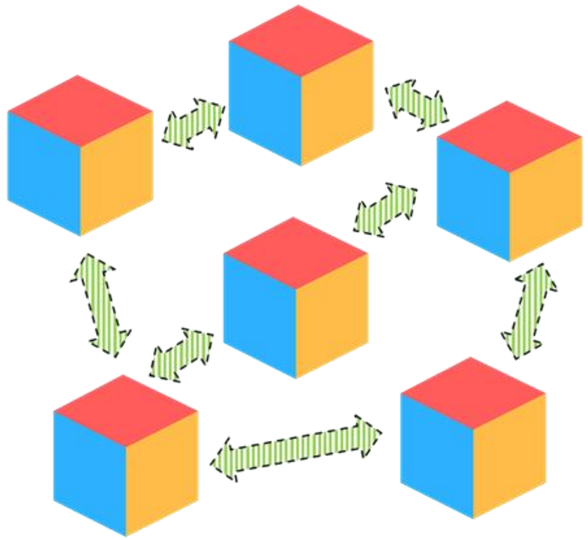
DEVIEW
2019

DEVIEW
2019

1.소개

What is

Observabilty?

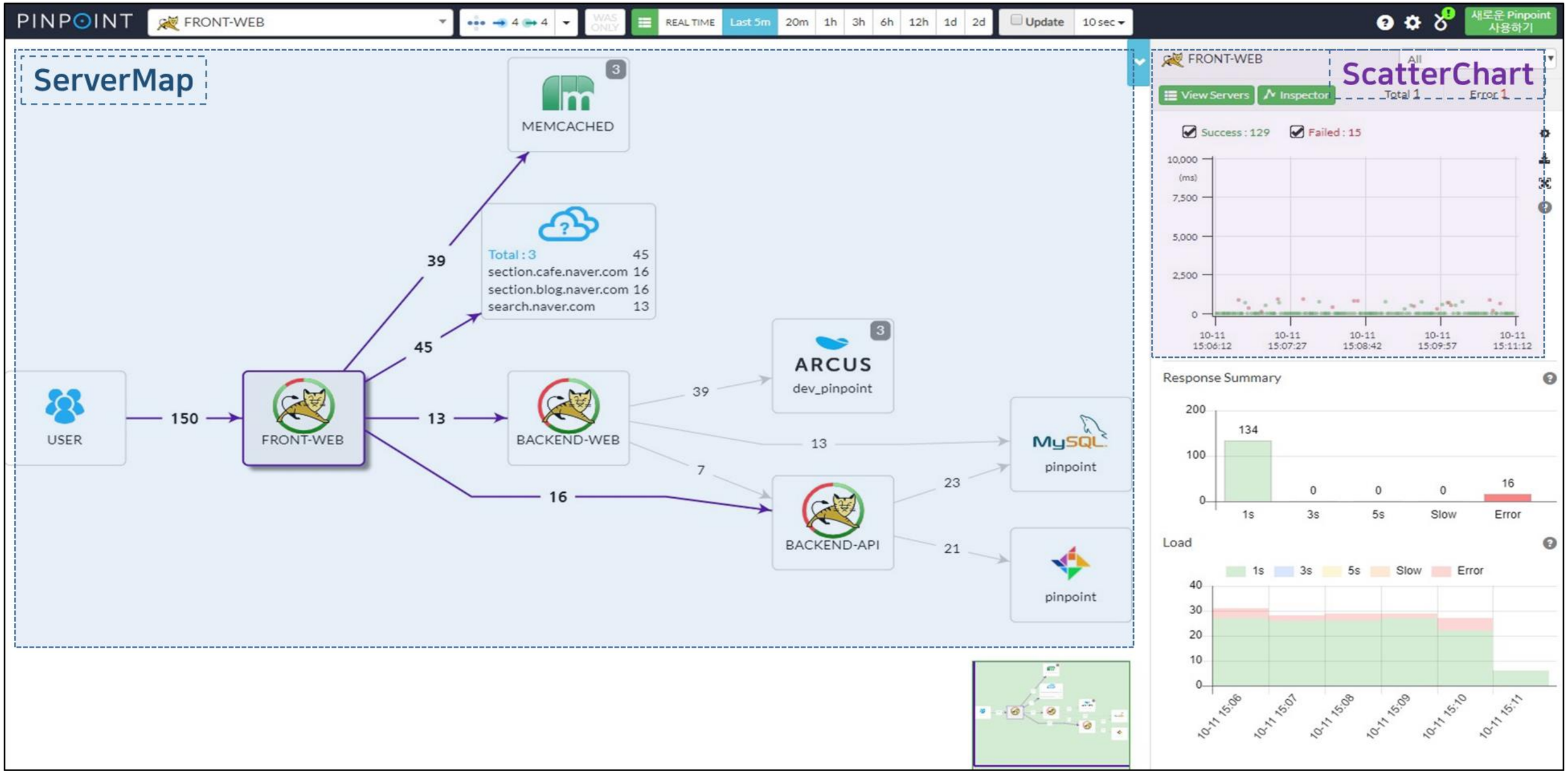
		
Server	Application	Distributed systems

Monitoring → Observability

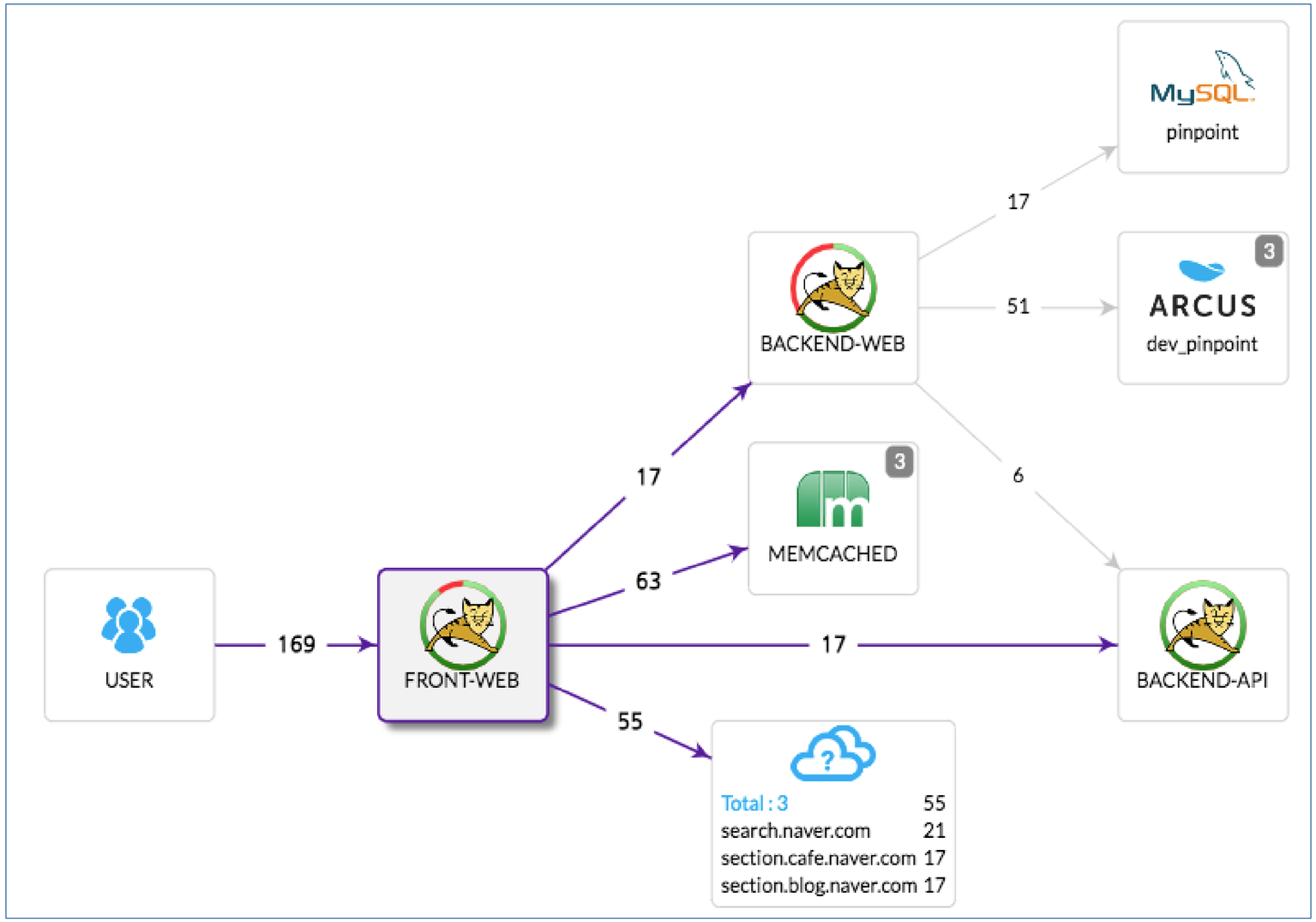
Pinpoint is APM tool for

large-scale distributed systems

```
JAVA_OPTS = “$JAVAOPTS
 -javaagent:$PINPOINT_HOME/pinpoint-bootstrap-1.9.0.jar
 -Dpinpoint.agentId=$AGENT_ID -Dpinpoint.applicationName=$APPLICATION_NAME”
```

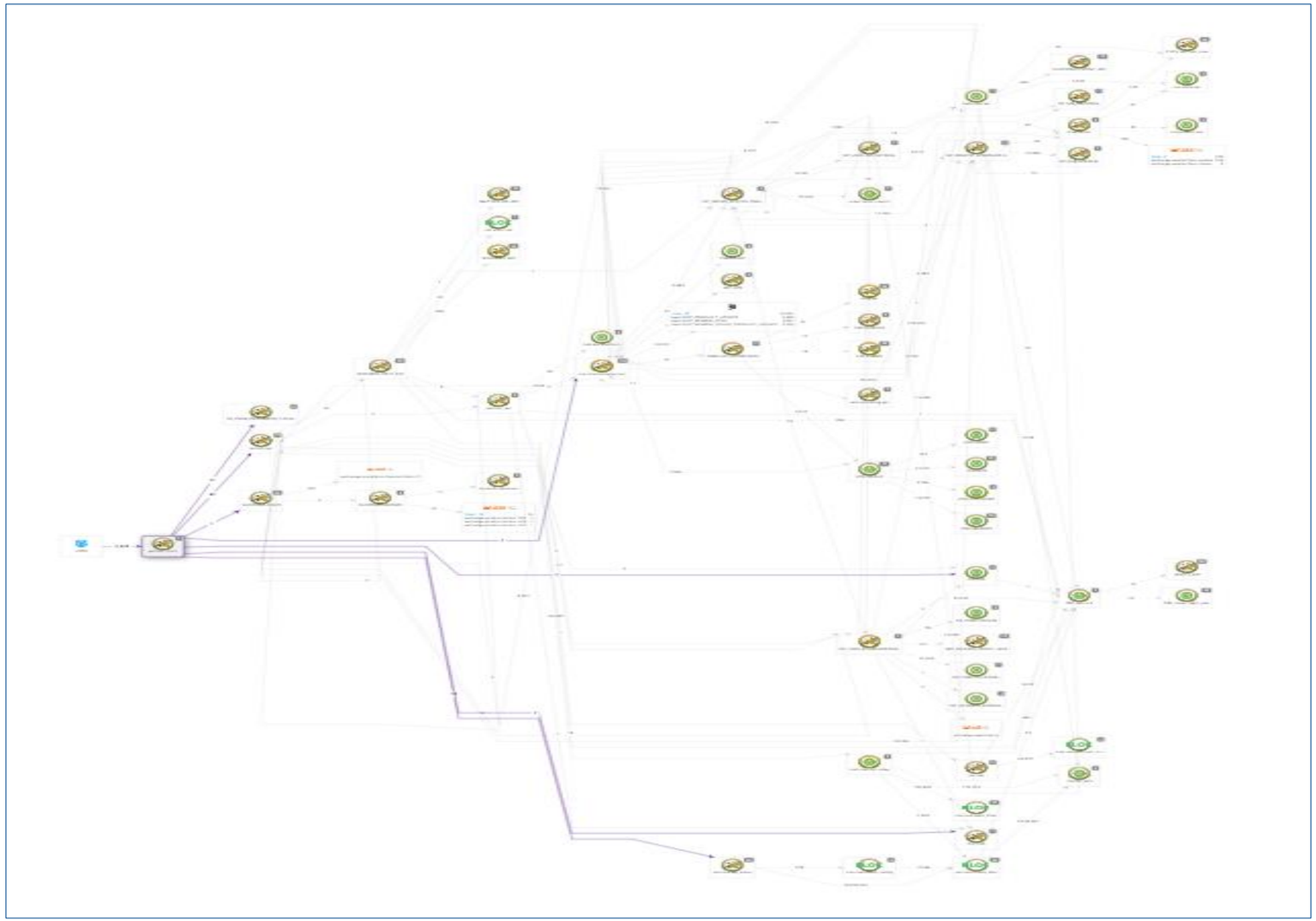

PINPOINT
FRONT-WEB
REAL TIME
Last 5m
20m
1h
3h
6h
12h
1d
2d
Update
10 sec
새로운 Pinpoint 사용하기
ServerMap
MEMCACHED
Total : 3 45
section.cafe.naver.com 16
section.blog.naver.com 16
search.naver.com 13
ARCUS
dev_pinpoint
USER
150
FRONT-WEB
39
45
13
16
BACKEND-WEB
39
13
7
BACKEND-API
23
21
MySQL
pinpoint
pinpoint
FRONT-WEB
ScatterChart
View Servers
Inspector
Total 1
Error 1
Success : 129
Failed : 15
10,000
(ms)
7,500
5,000
2,500
0
10-11 15:06:12
10-11 15:07:27
10-11 15:08:42
10-11 15:09:57
10-11 15:11:12
Response Summary
200
100
0
134
0
0
0
16
1s
3s
5s
Slow
Error
Load
1s
3s
5s
Slow
Error
40
30
20
10
0
10-11 15:06
10-11 15:07
10-11 15:08
10-11 15:09
10-11 15:10
10-11 15:11

ServerMap


MySQL
pinpoint
17
BACKEND-WEB
51
3
ARCUS
dev_pinpoint
6
3
MEMCACHED
17
63
169
USER
FRONT-WEB
17
BACKEND-API
55
Total : 3 55
search.naver.com 21
section.cafe.naver.com 17
section.blog.naver.com 17

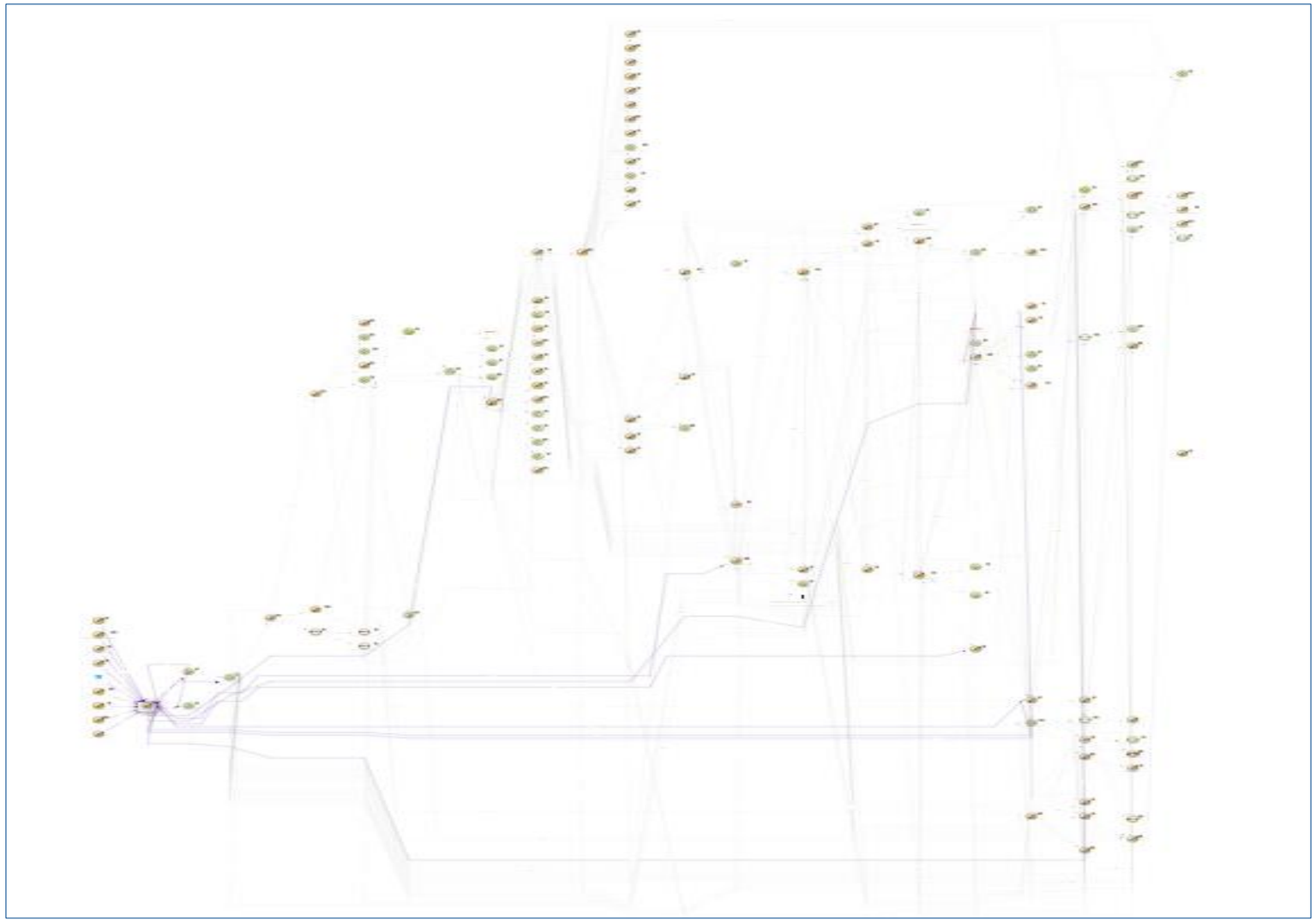
ServerMap

DEVIEW
2019

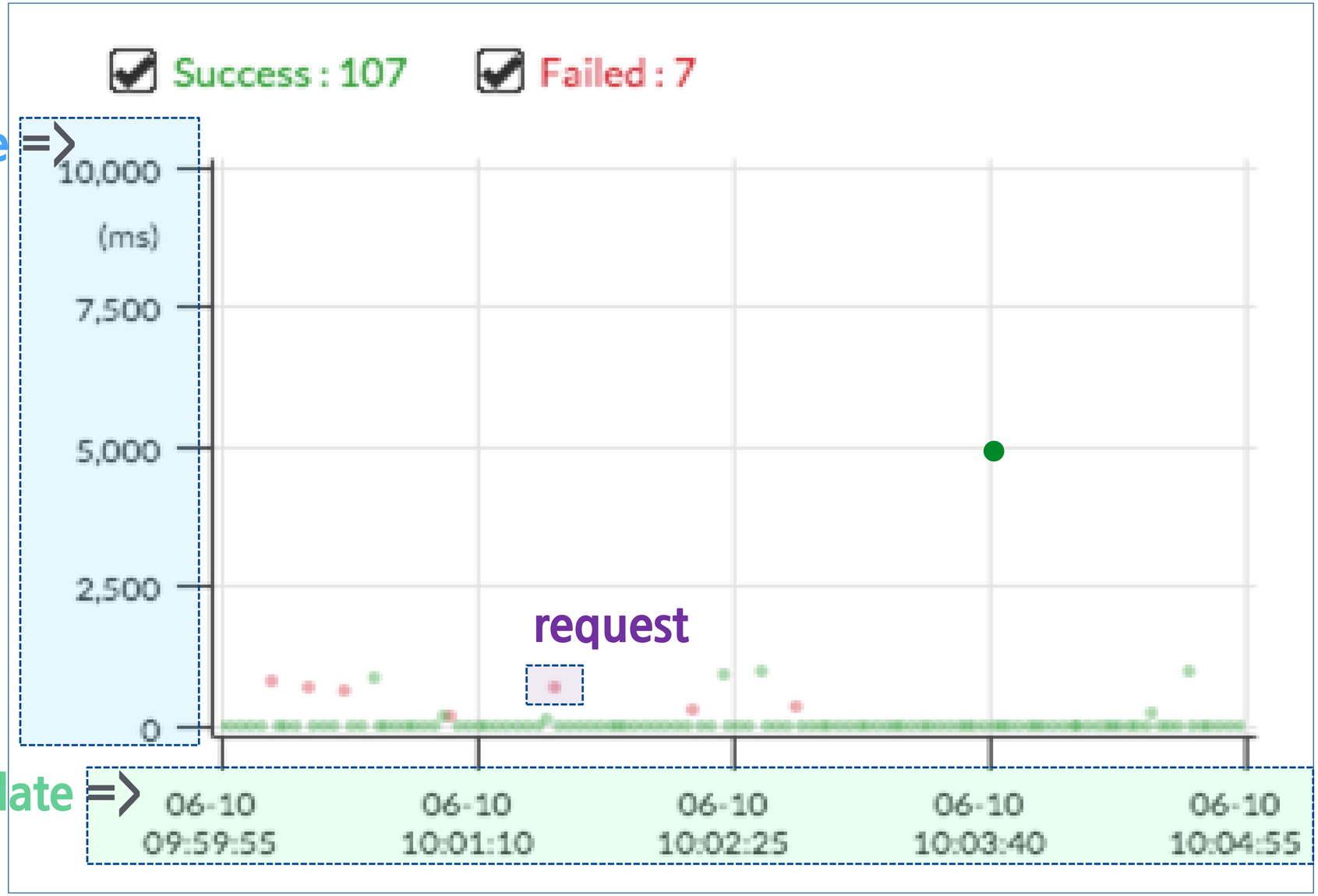


ServerMap

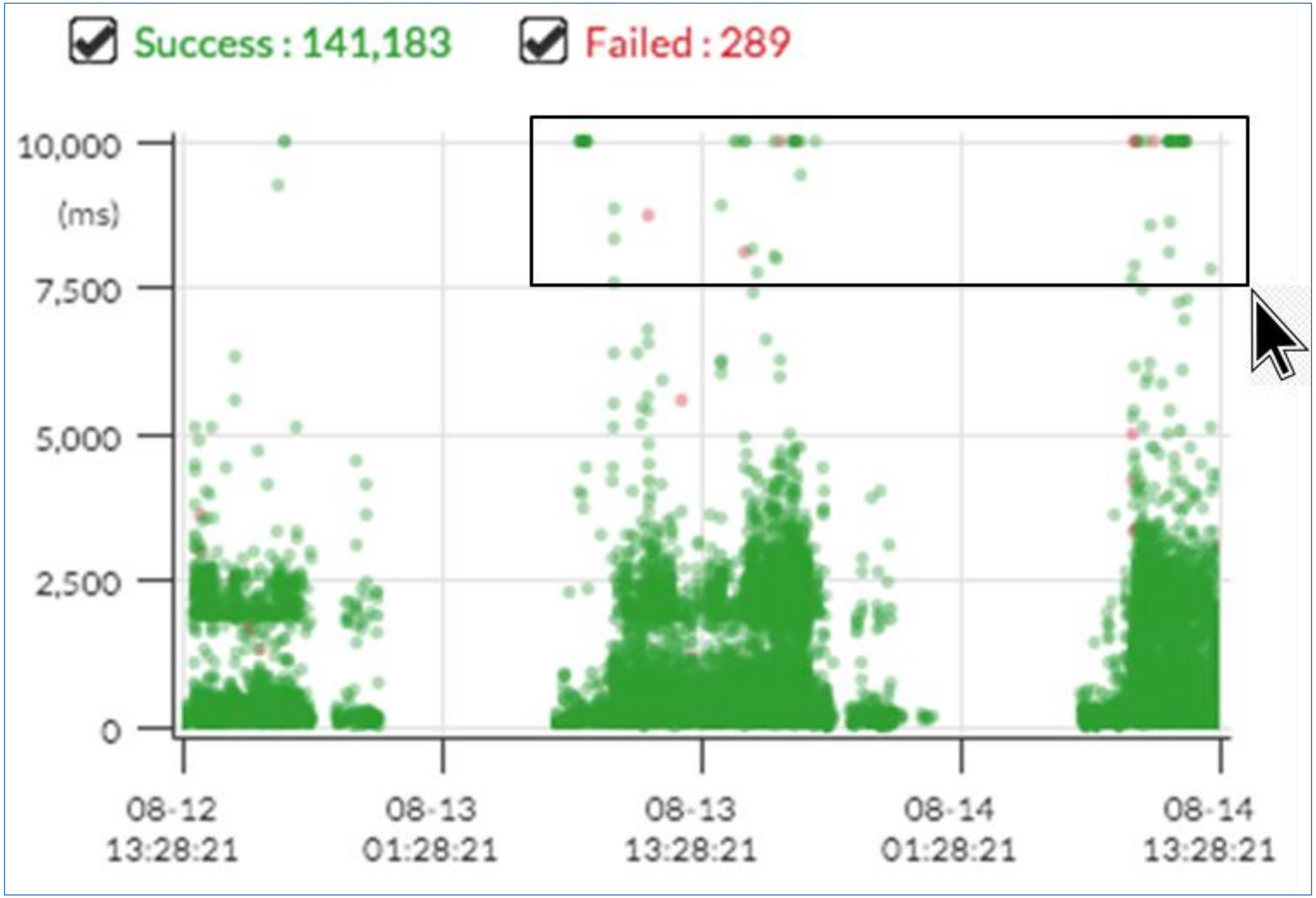
DEVIEW
2019



ScatterChart



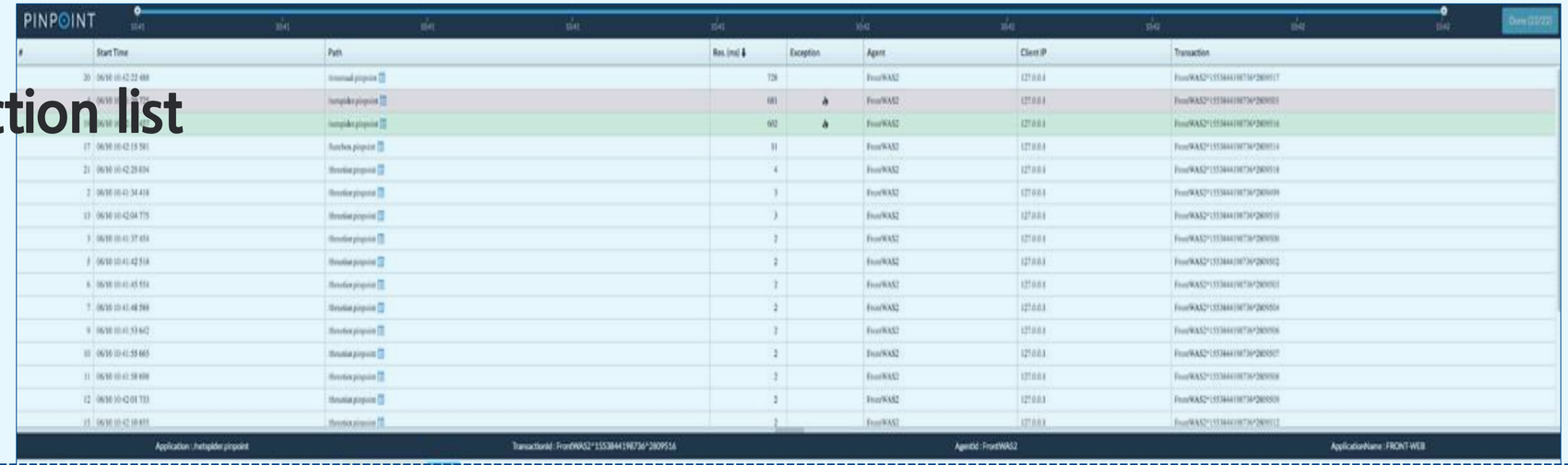
ScatterChart



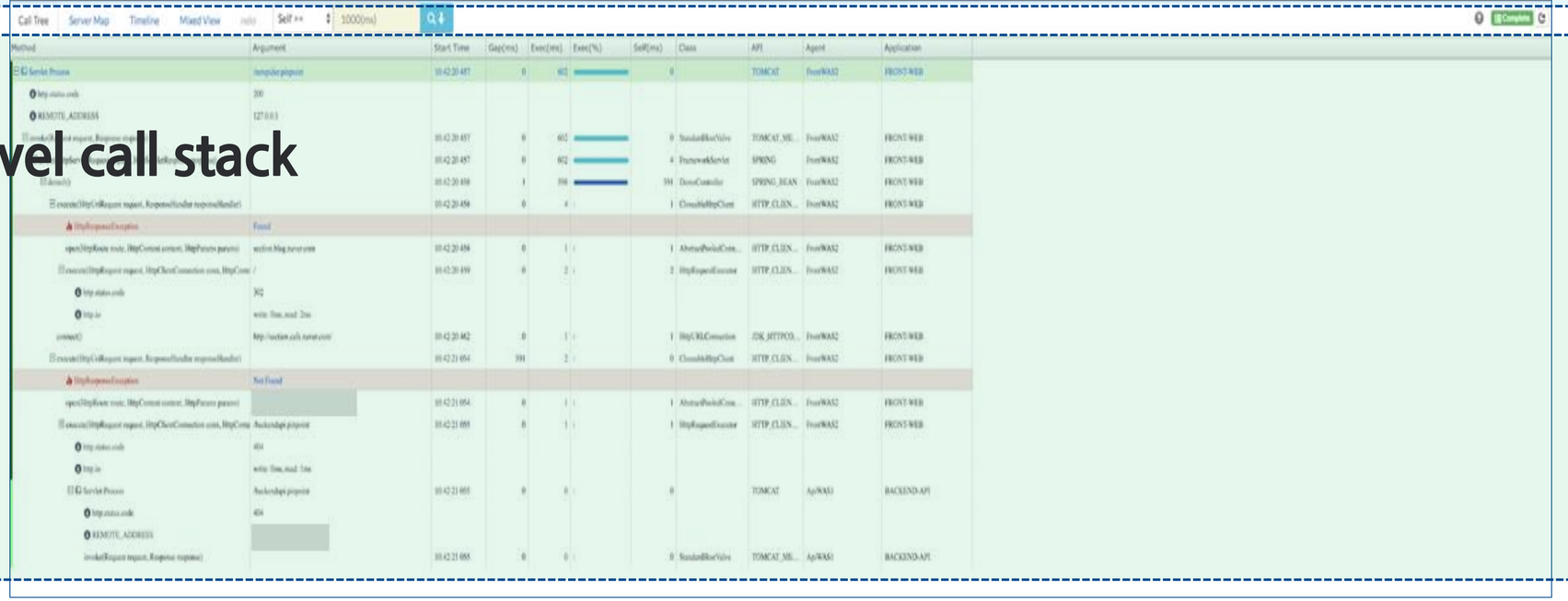
Distributed Call tree

DEVIEW 2019

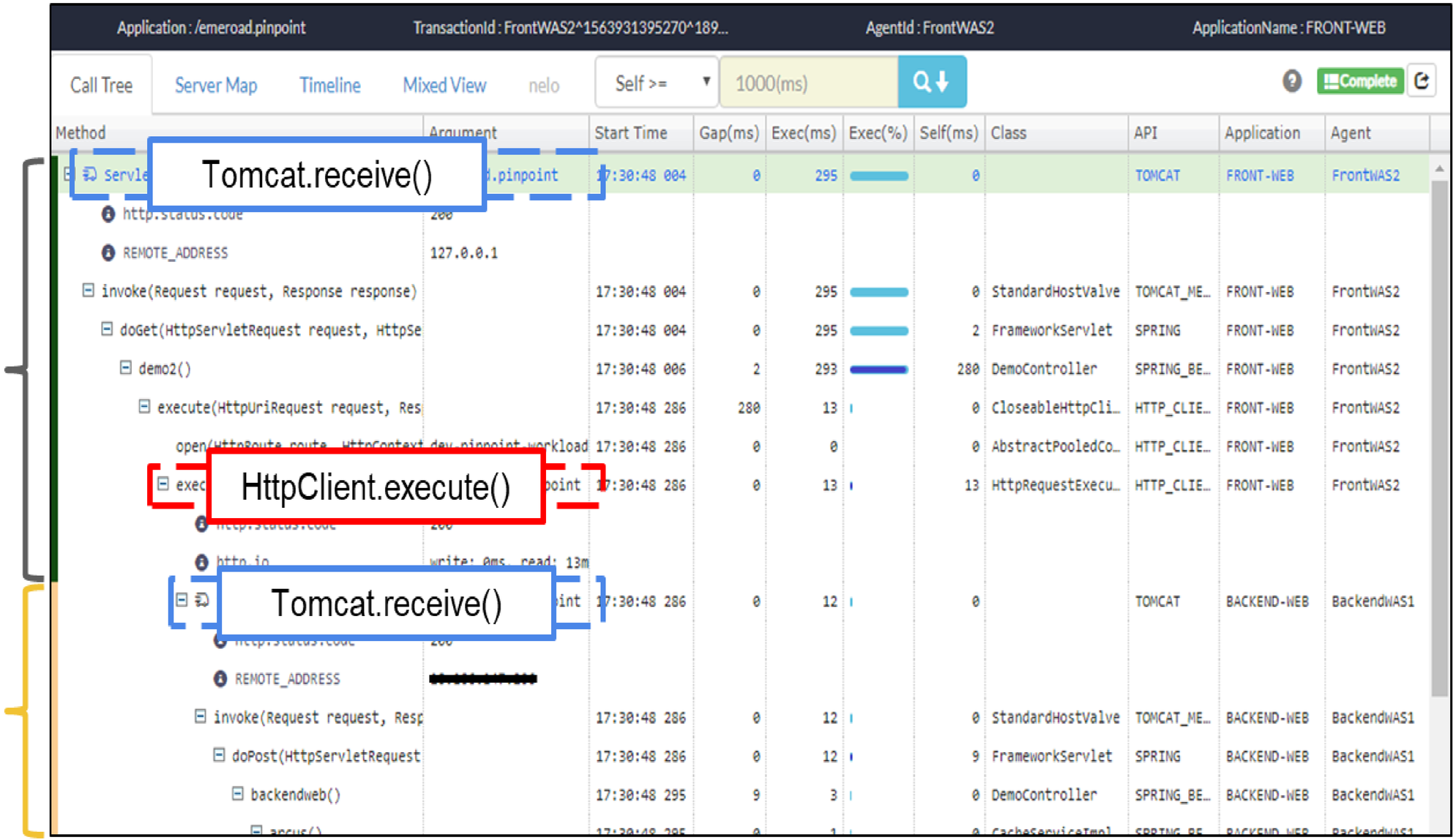
Transaction list



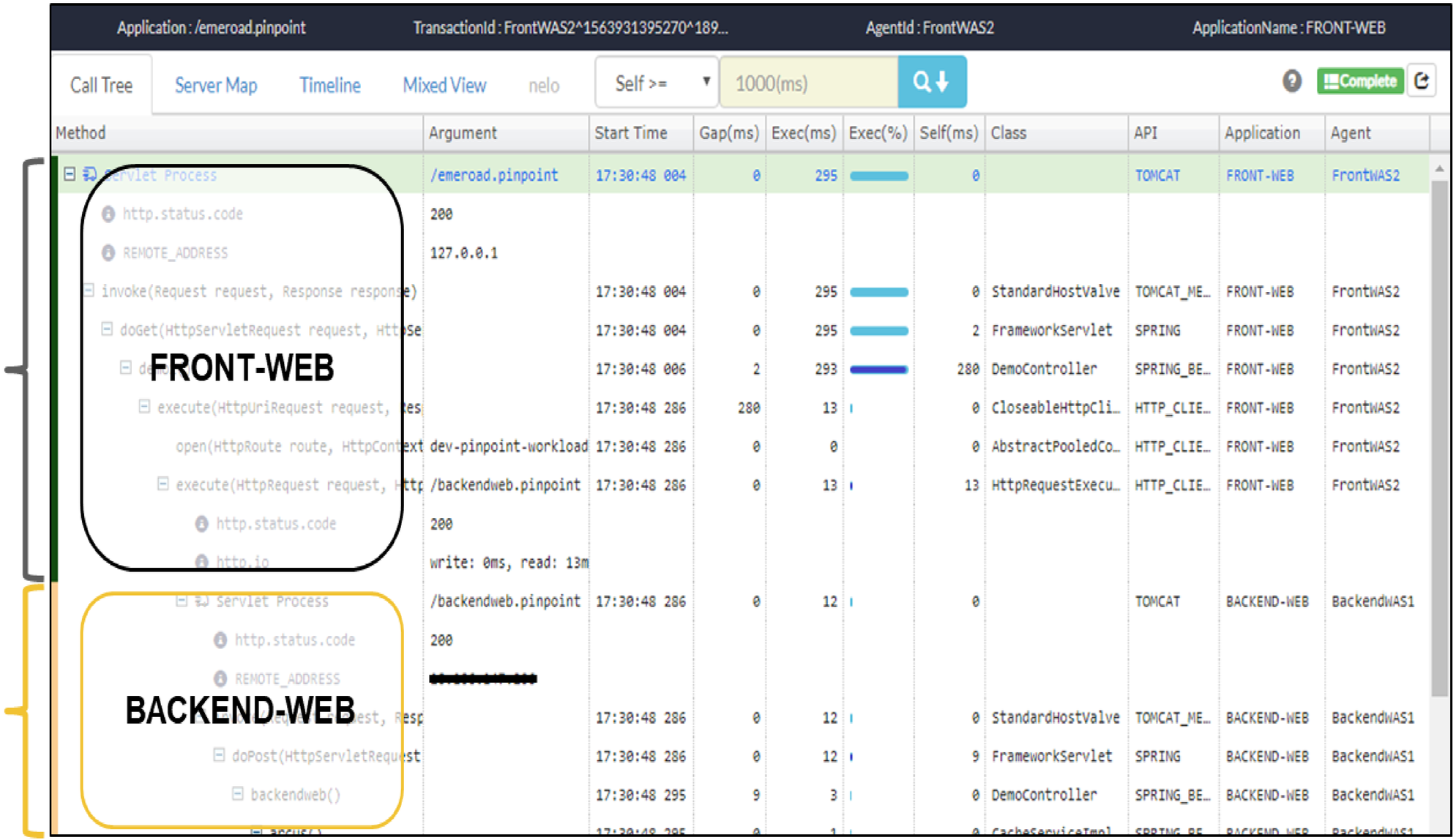
Code level call stack



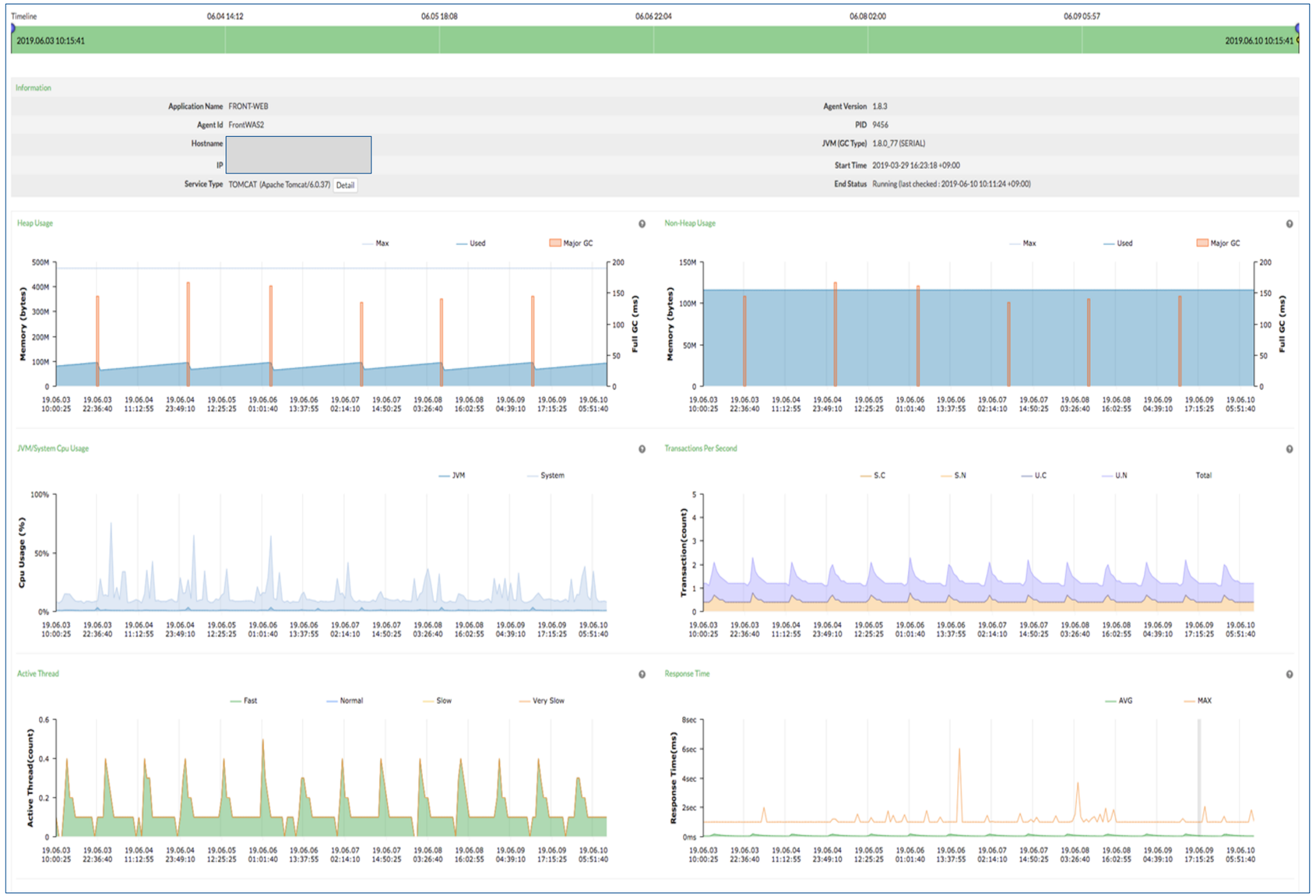
Distributed Call tree



Distributed Call tree



Inspector



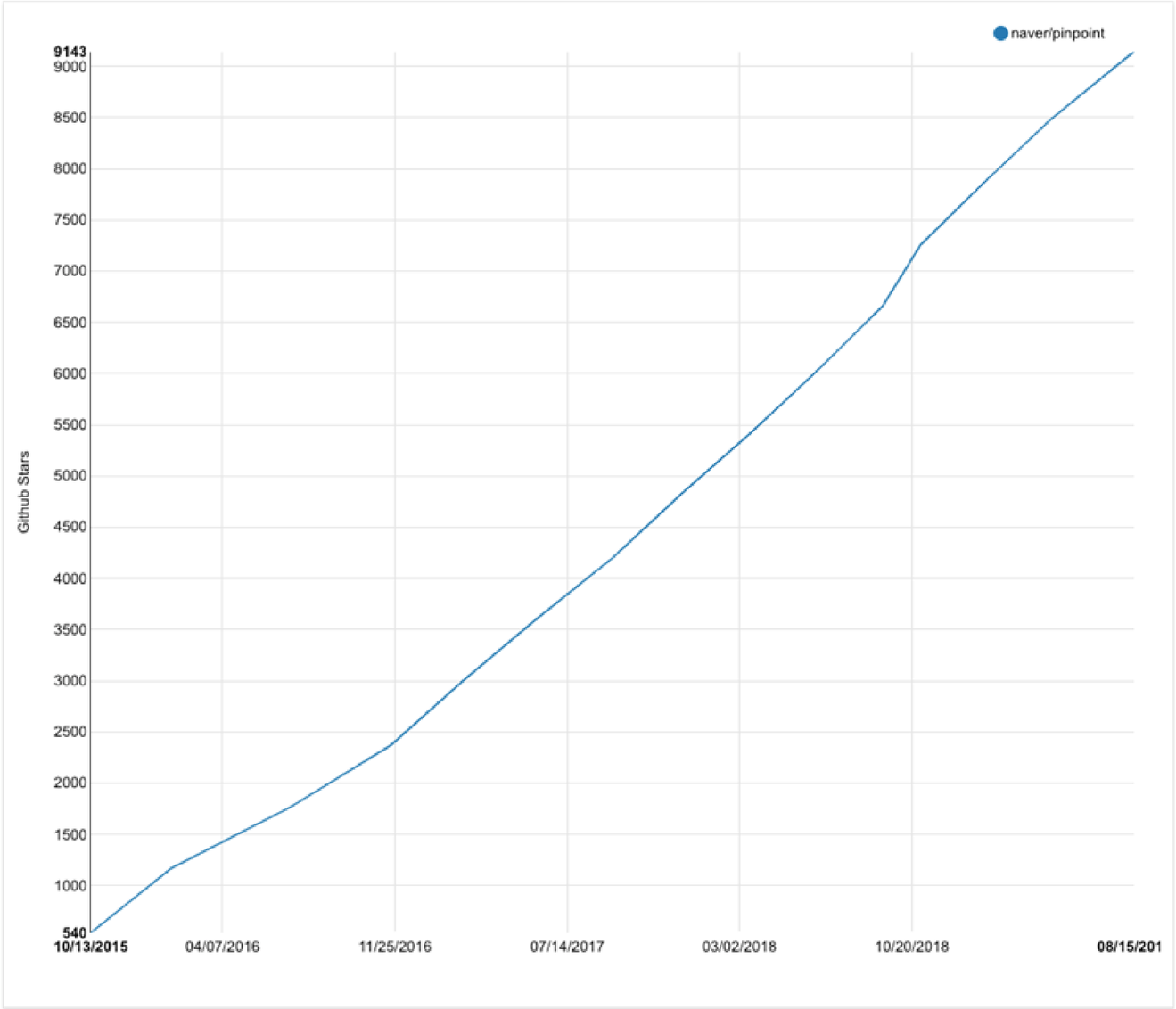
- Timeline
- Memory
- CPU
- JVM GC
- TPS
- Active Thread
- Response Time
- File Descriptor
- Buffer Usage
- Data Source

Pinpoint is

Opensource

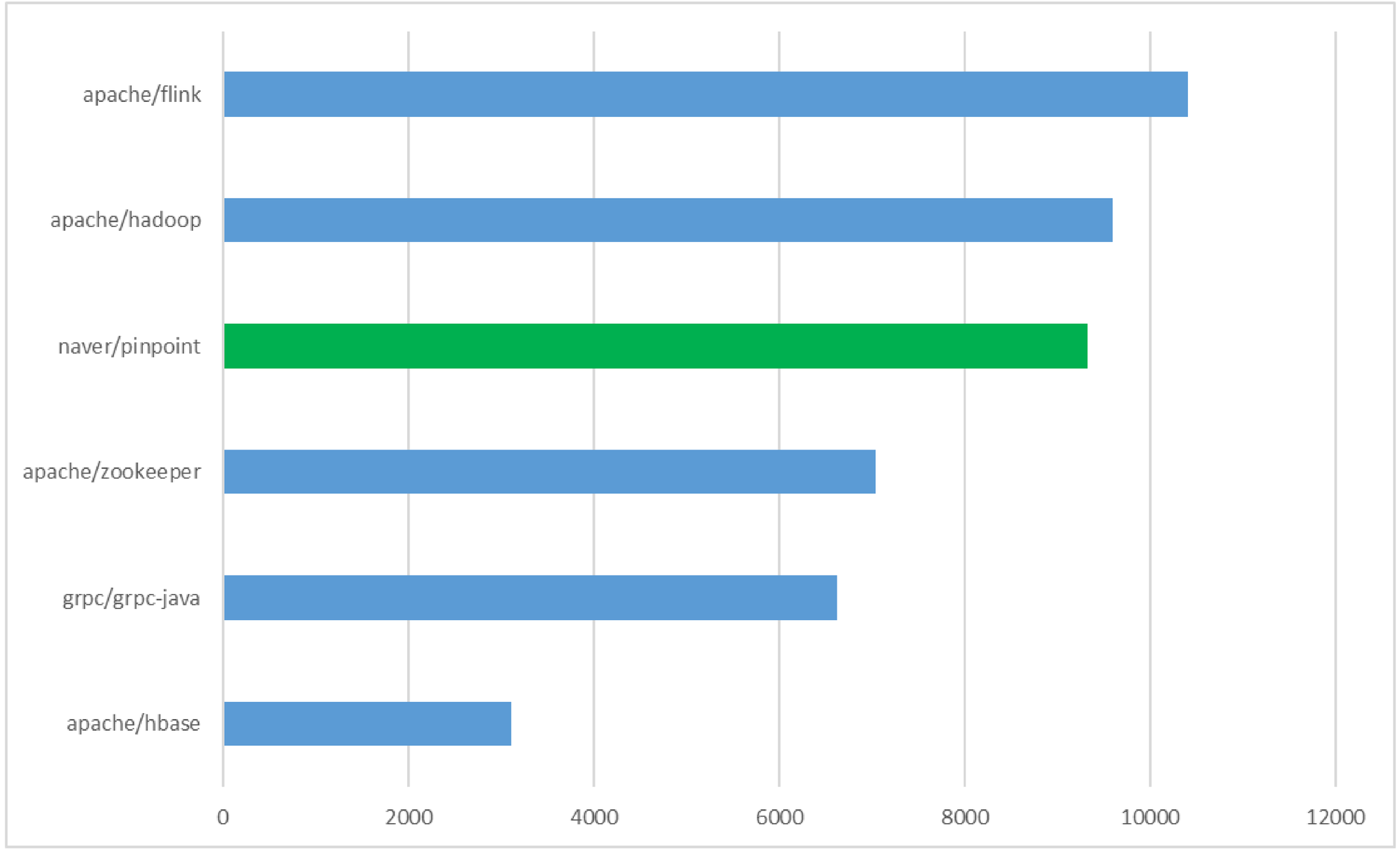
Github star

DEVIEW
2019



Github star

DEVIEW 2019



Active usage

1.	China	73.34%
2.	South Korea	6.97%
3.	Hong Kong	5.10%
4.	United States	4.15%
5.	India	1.82%

2019. 1. 1. - 2019. 9. 22.

사용자

- Lost of Top 10 IT companies in China
- Global enterprises in Korea
- Various IT companies in USA
- Companies in Financial Industry

DEVIEW
2019

Pinpoint 코어가

궁금하다면 데뷰 2015

다시보기

DEVIEW
2019

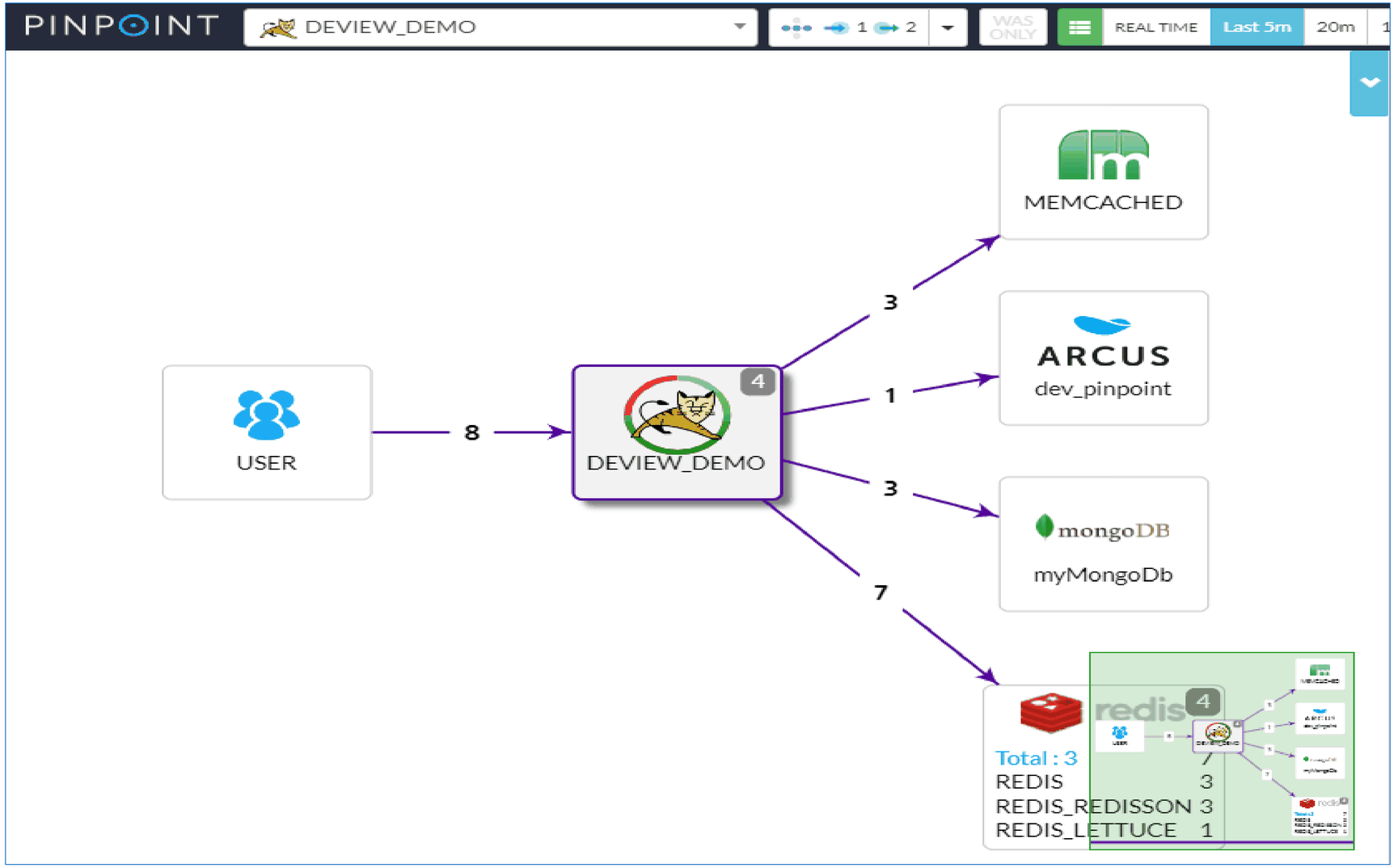
2. 실시간 observability 확보하기

해결해야 하는 문제

현재 각 에이전트가 처리하고 있는 요청은 얼마나 되는지?

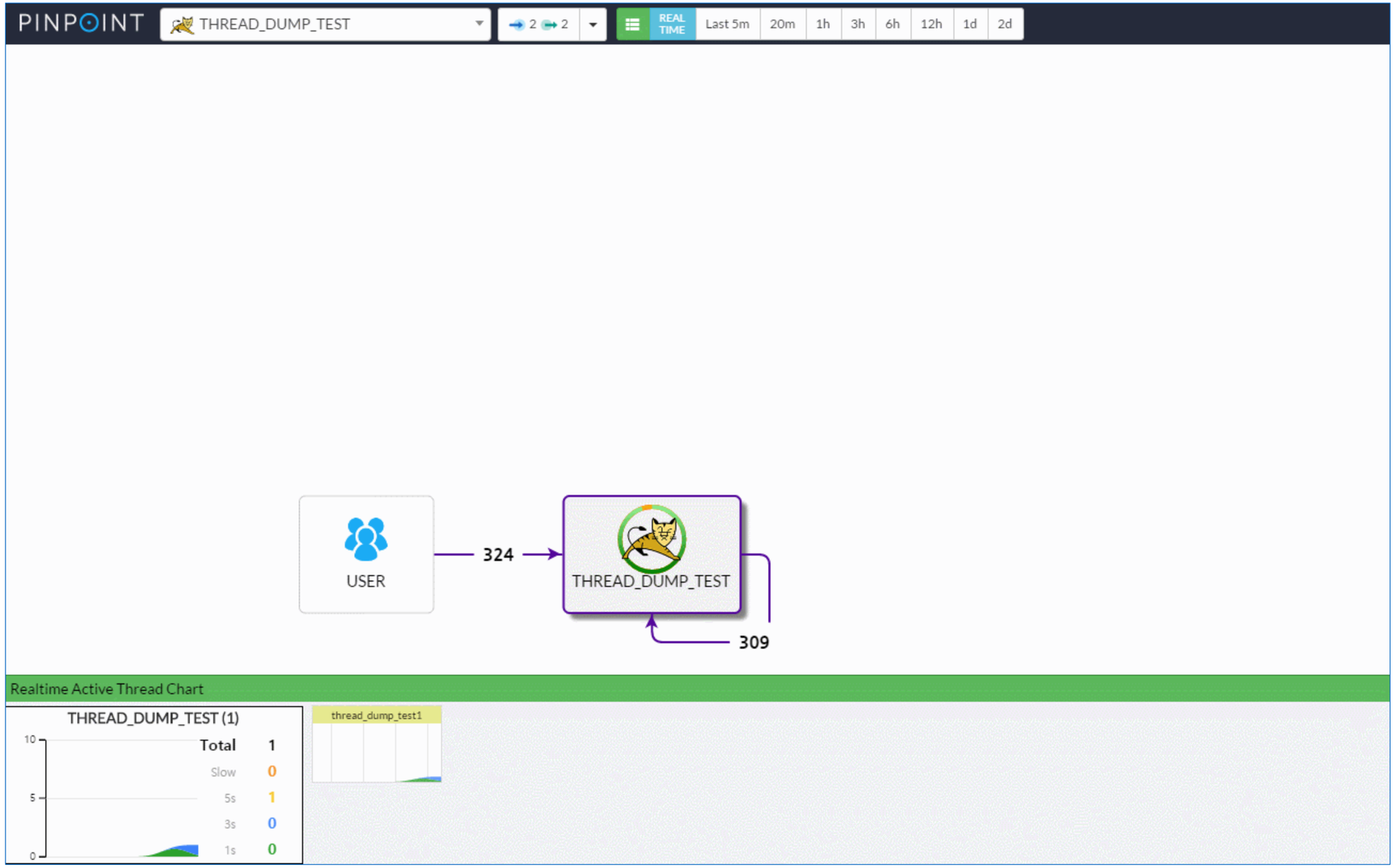
예상하지 못한 문제로 끝나지 않는 요청이 있다면 어떻게 확인하는지?

해결 (리얼타임기능)

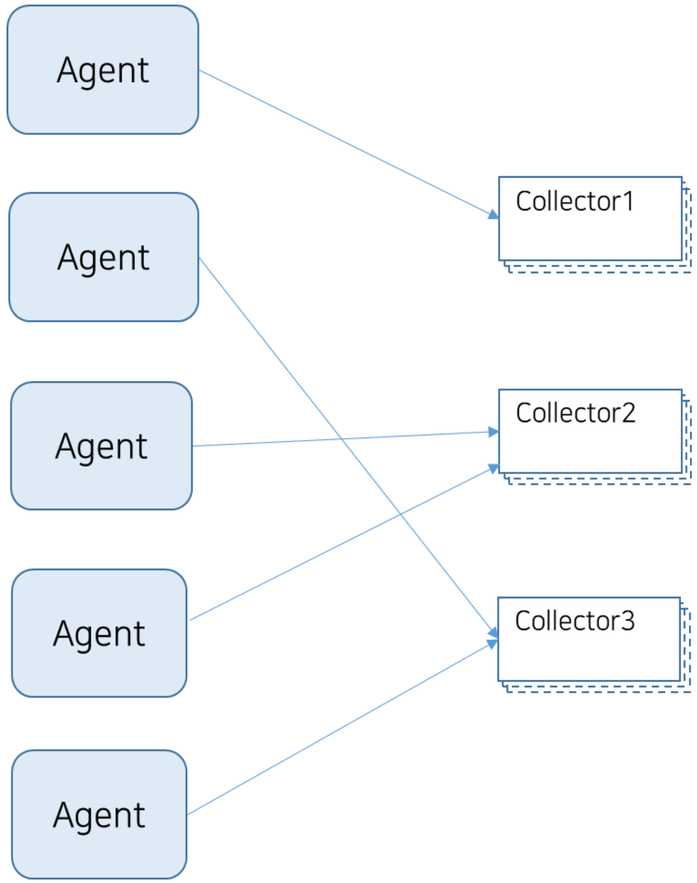

PINPOINT
DEVIEW_DEMO
1
2
WAS ONLY
REAL TIME
Last 5m
20m
MEMCACHED
ARCUS
dev_pinpoint
mongoDB
myMongoDb
USER
8
4
DEVIEW_DEMO
3
1
3
7
redis
4
Total : 3
REDIS 3
REDIS_REDISSON 3
REDIS_LETTUCE 1

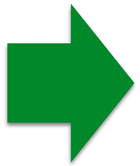
해결 (실시간 스레드덤프 기능)

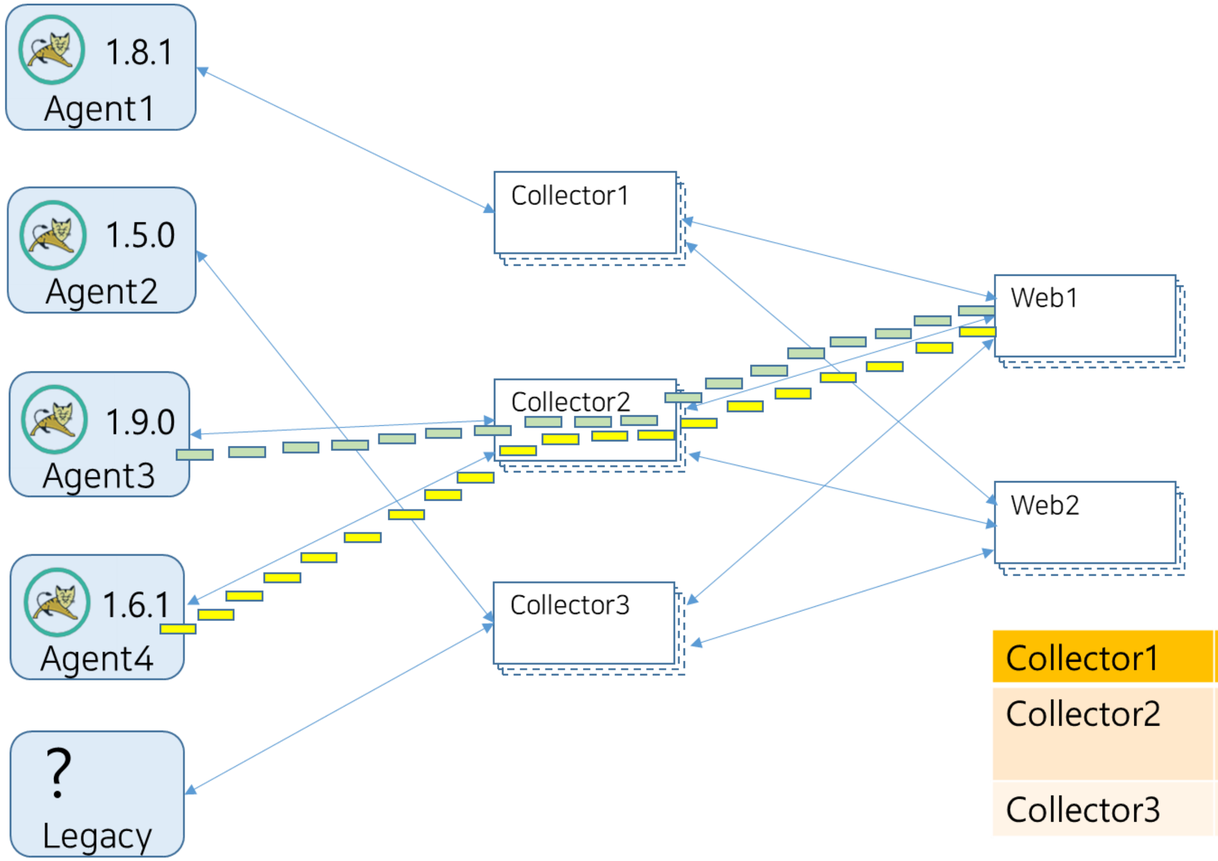
DEVIEW 2019



해결 방법 : 구조 변경

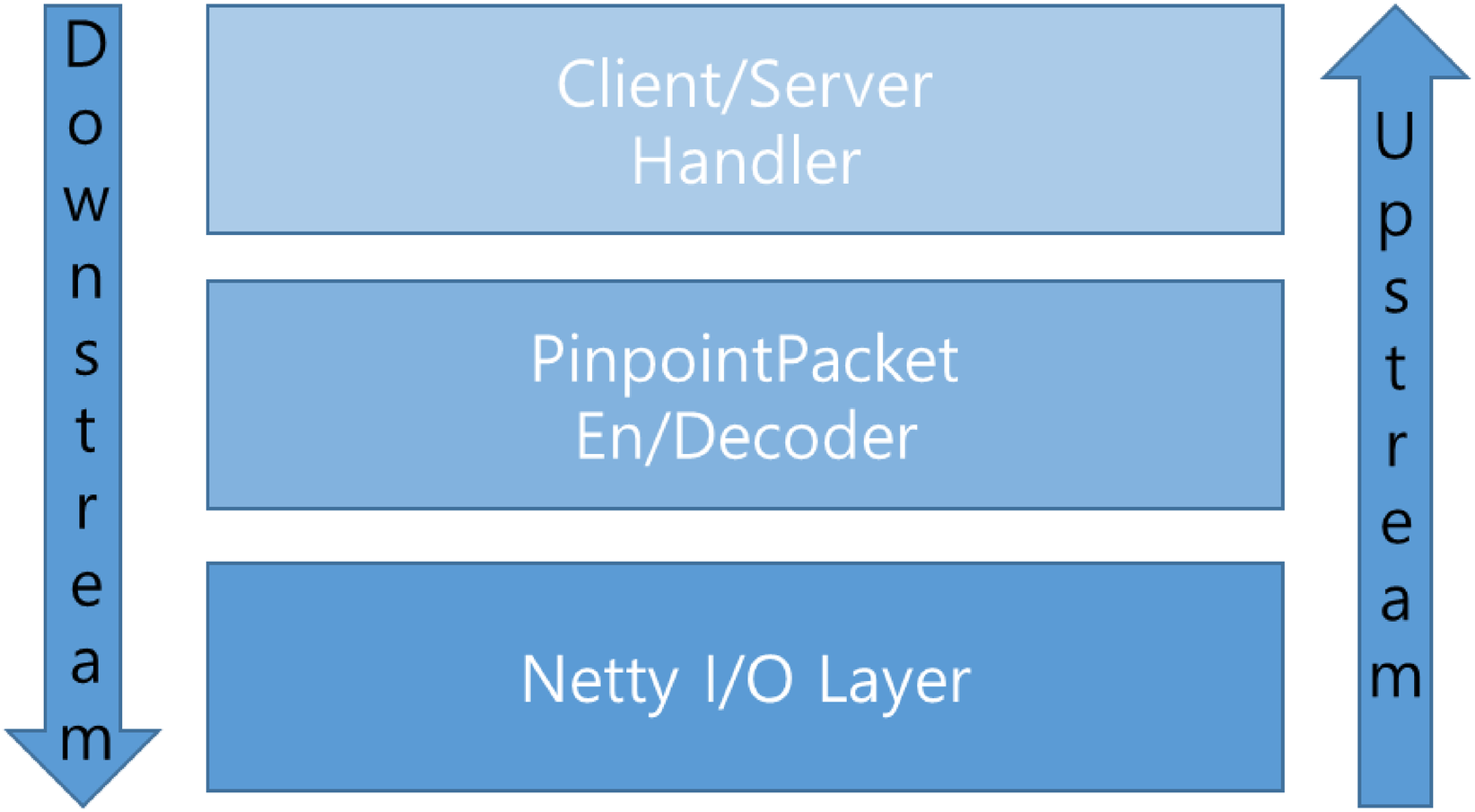






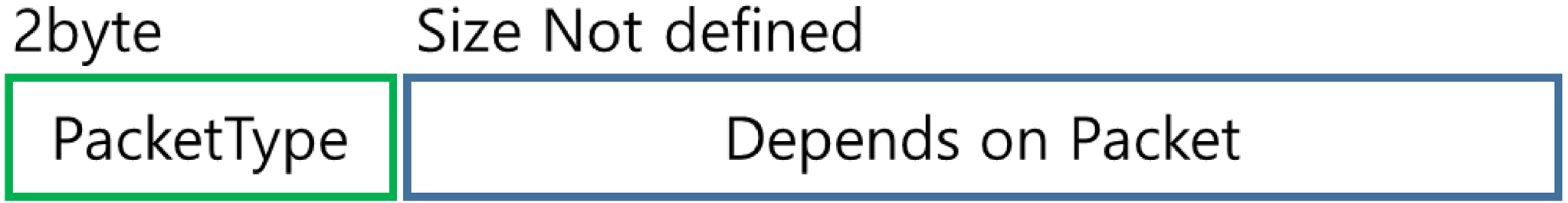
Collector1	Agent1:1.8.1
Collector2	Agent3:1.9.0 Agenr4:1.6.1
Collector3	Agent2:1.5.0

1. RPC 구조

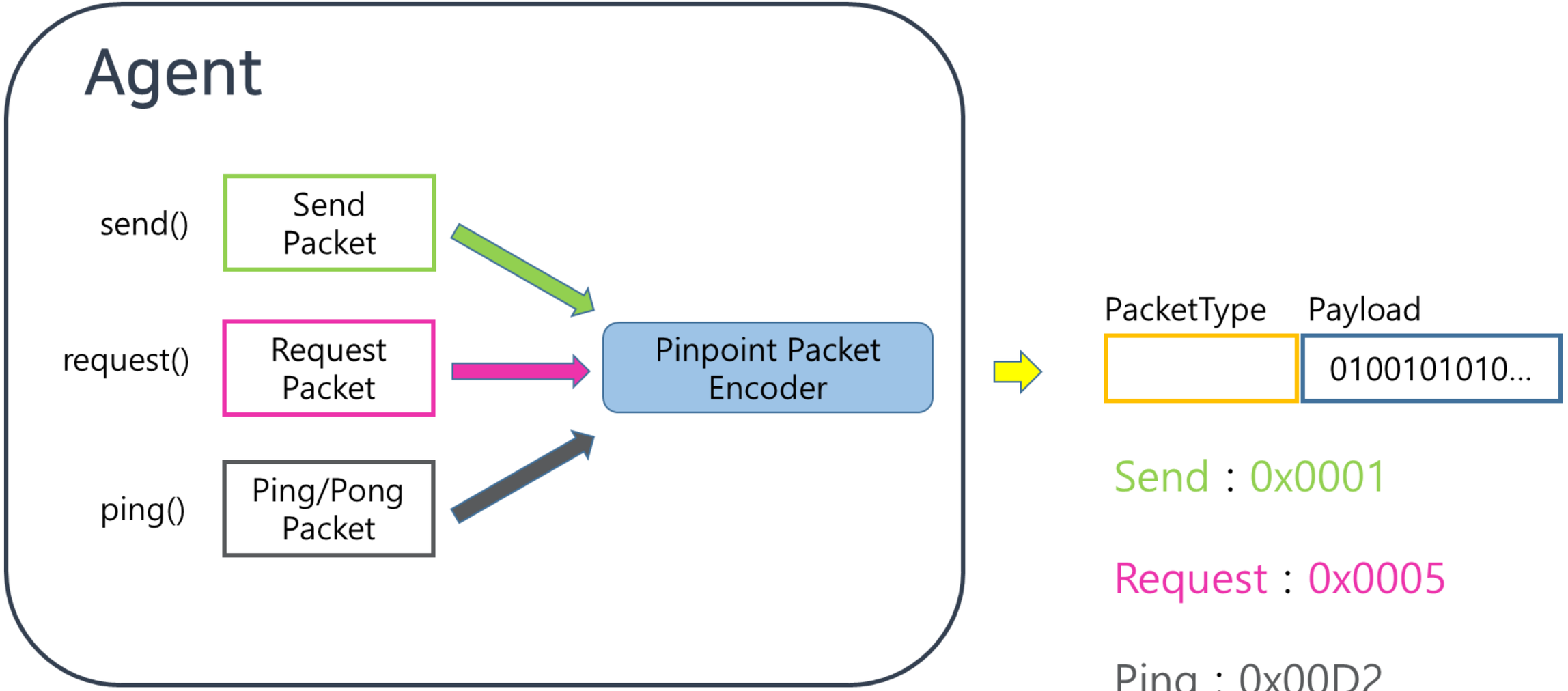

Downstream
Client/Server
Handler
PinpointPacket
En/Decoder
Netty I/O Layer
Upstream

1. PinpointPacket 이란?

- 핀포인트내에서 데이터를 주고 받는 최소 단위
- 각 패킷은 역할을 가짐 (Send,Request, ... 등)
- Packet Format (Binary Message)

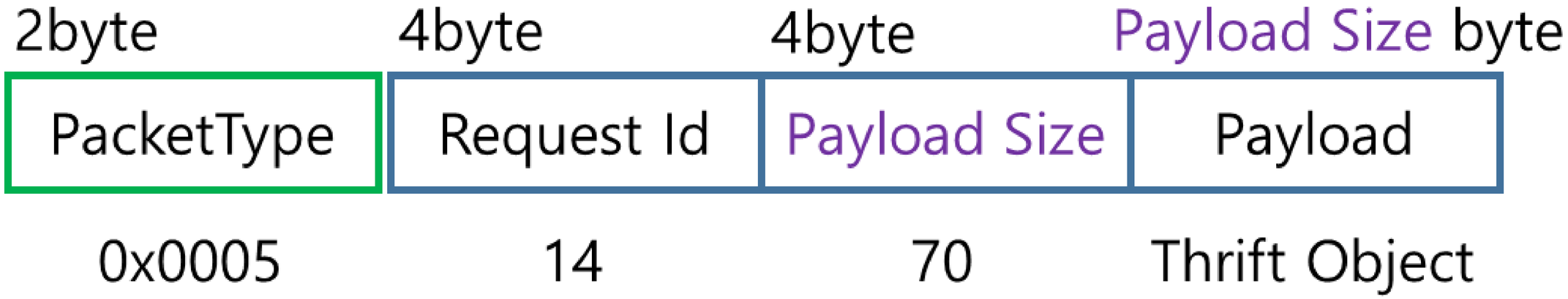


1. 에이전트에서 Packet 송신

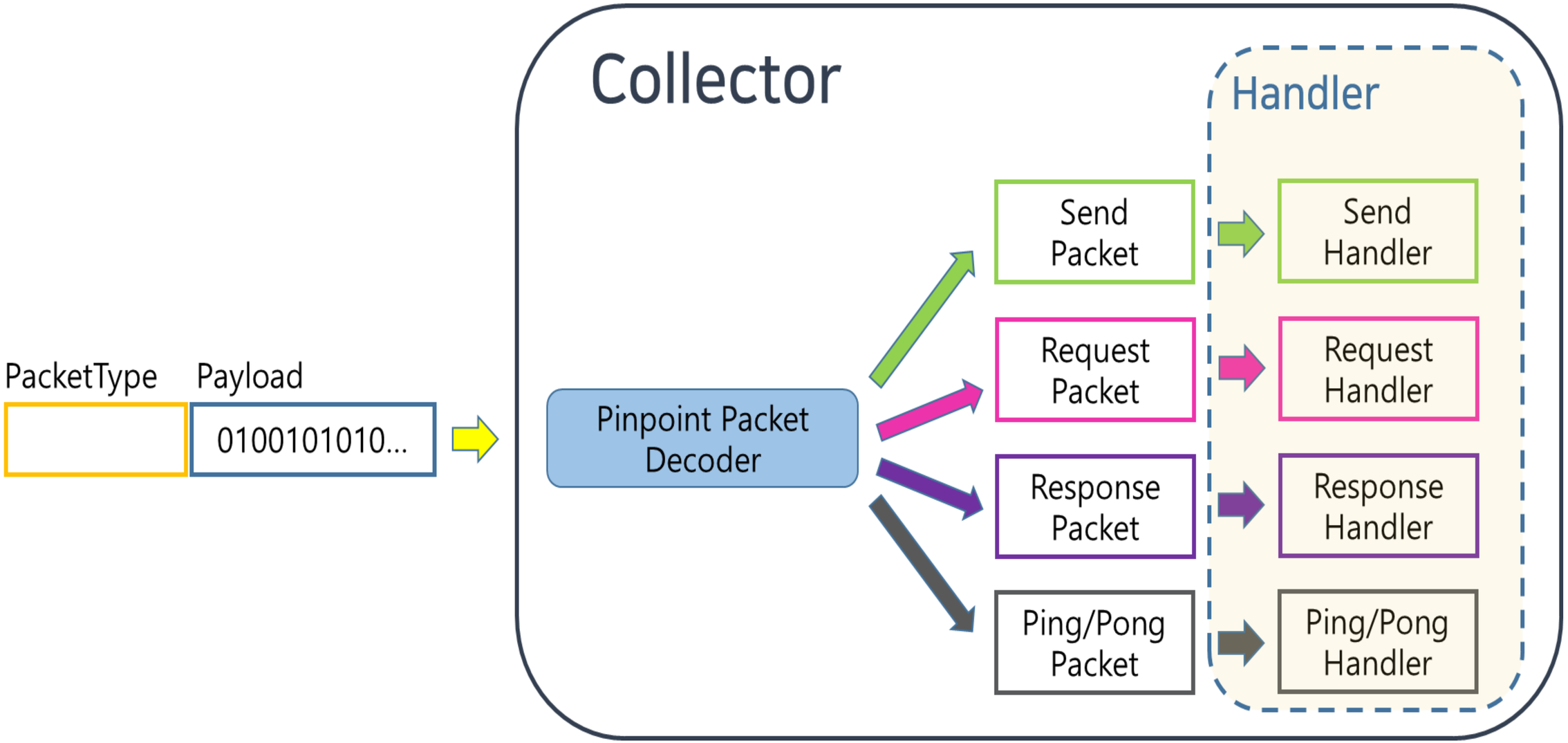


1. RequestPacket 구조

DEVIEW 2019

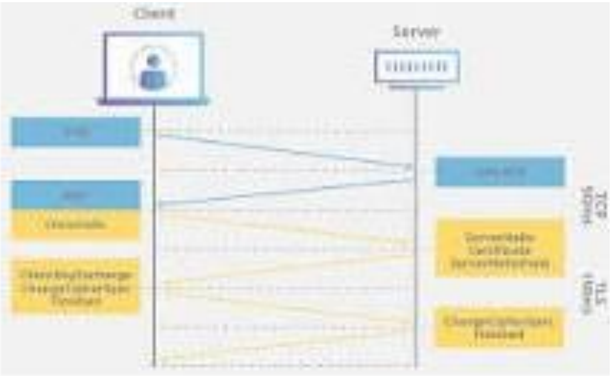


1. 콜렉터에서 Packet 수신



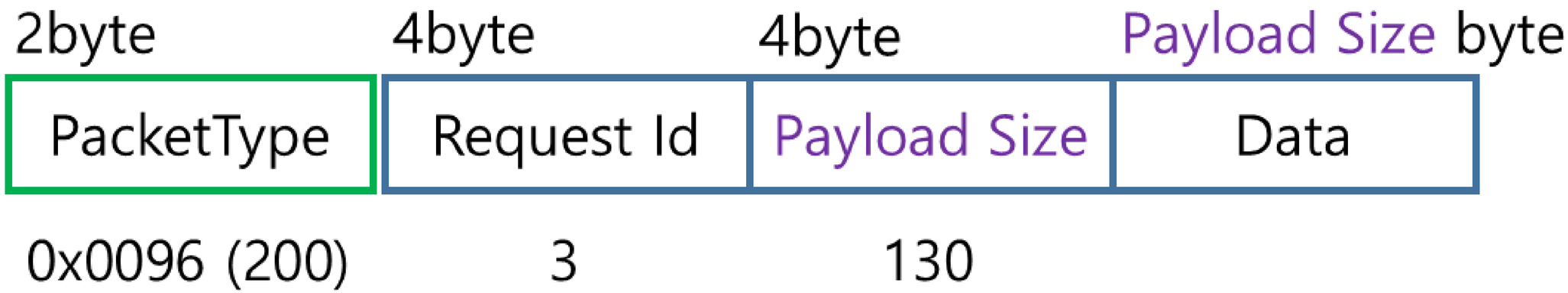
2. Handshake

A **TLS handshake** is the process that kicks off a communication session that uses **TLS** encryption. During a **TLS handshake**, the two communicating sides exchange messages to acknowledge each other, verify each other, establish the encryption algorithms they will use, and agree on session keys.



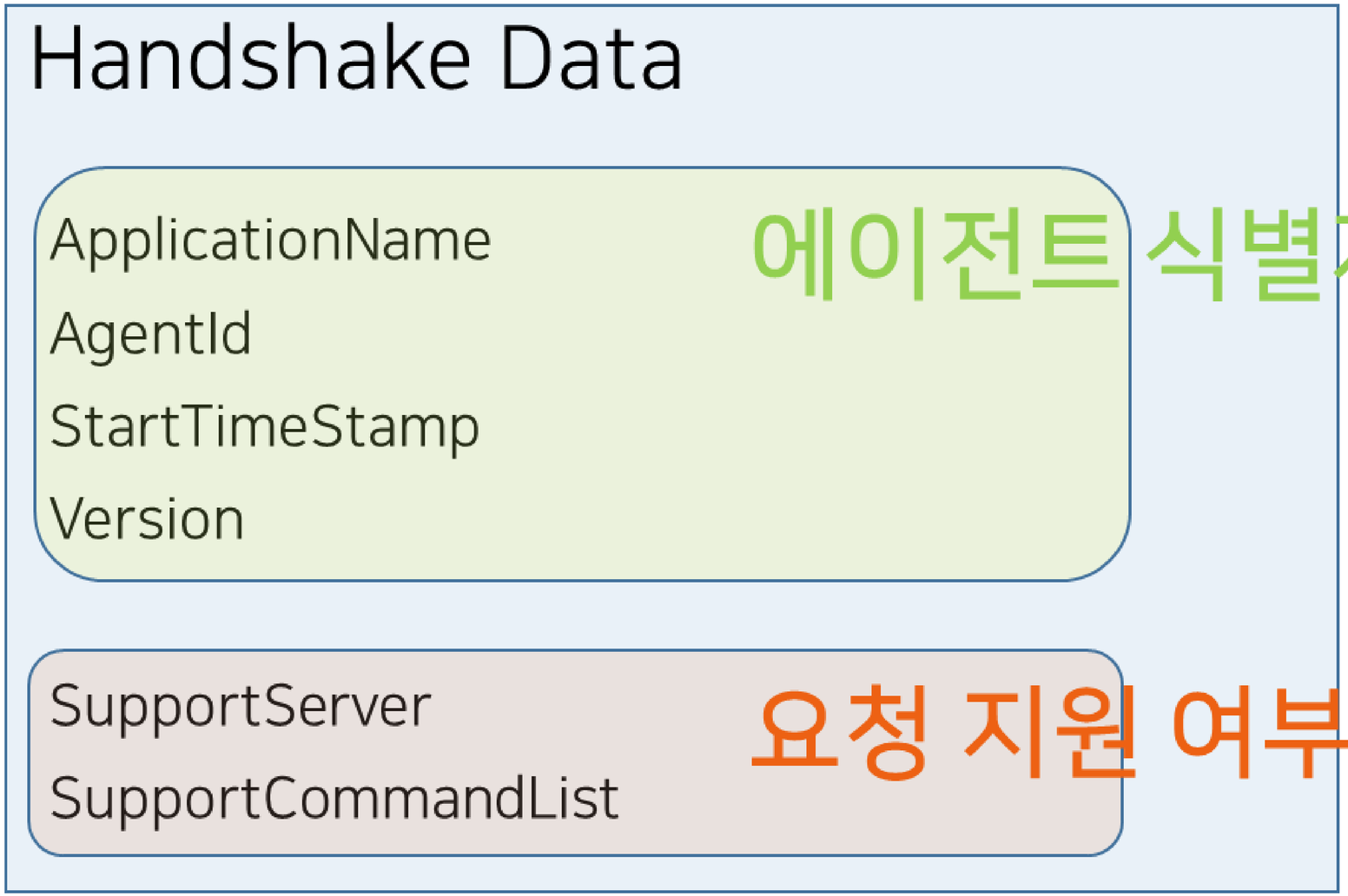
2. HandshakePacket 추가

- ControlHandshakePacket

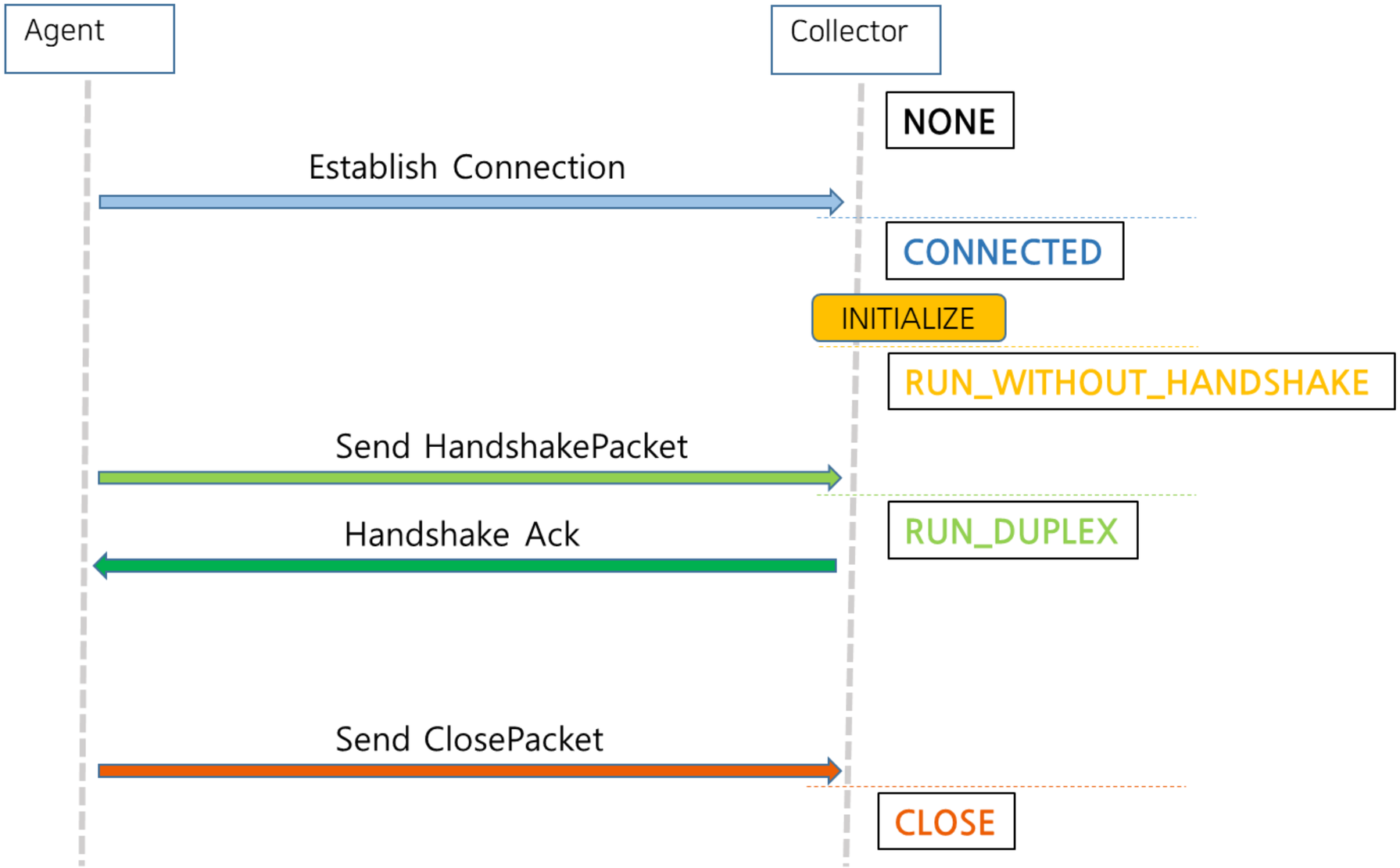


- ControlHandshakeResponsePacket

2. Handshake Data



2. 상태값 생성 및 변경

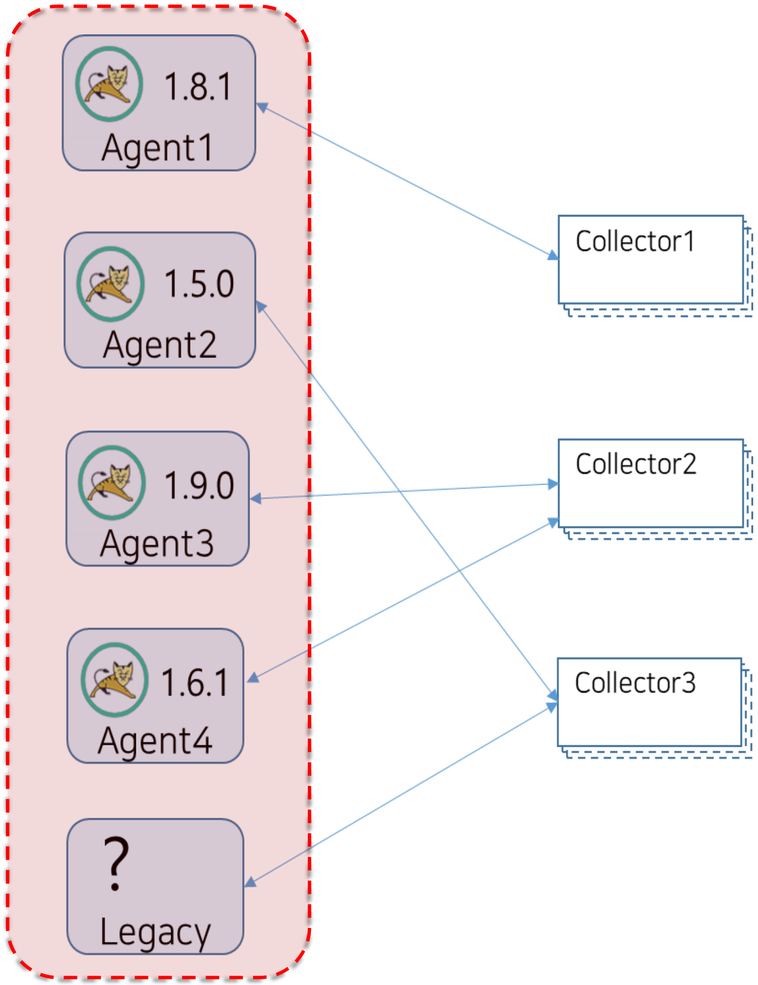

Agent
Collector
NONE
Establish Connection
CONNECTED
INITIALIZE
RUN_WITHOUT_HANDSHAKE
Send HandshakePacket
RUN_DUPLEX
Handshake Ack
Send ClosePacket
CLOSE

2. 상태 처리 리스너 생성

- StateChangeEventListener 생성

```
public interface StateChangeEventListener<S extends PinpointSocket> {

    void stateUpdated(S pinpointSocket, SocketStateCode updatedStateCode) throws Exception;

}
```

2. Handshake 결과



3. 에이전트 연결 정보 공유

DEVIEW
2019

What is ZooKeeper?

Apache ZooKeeper™

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they

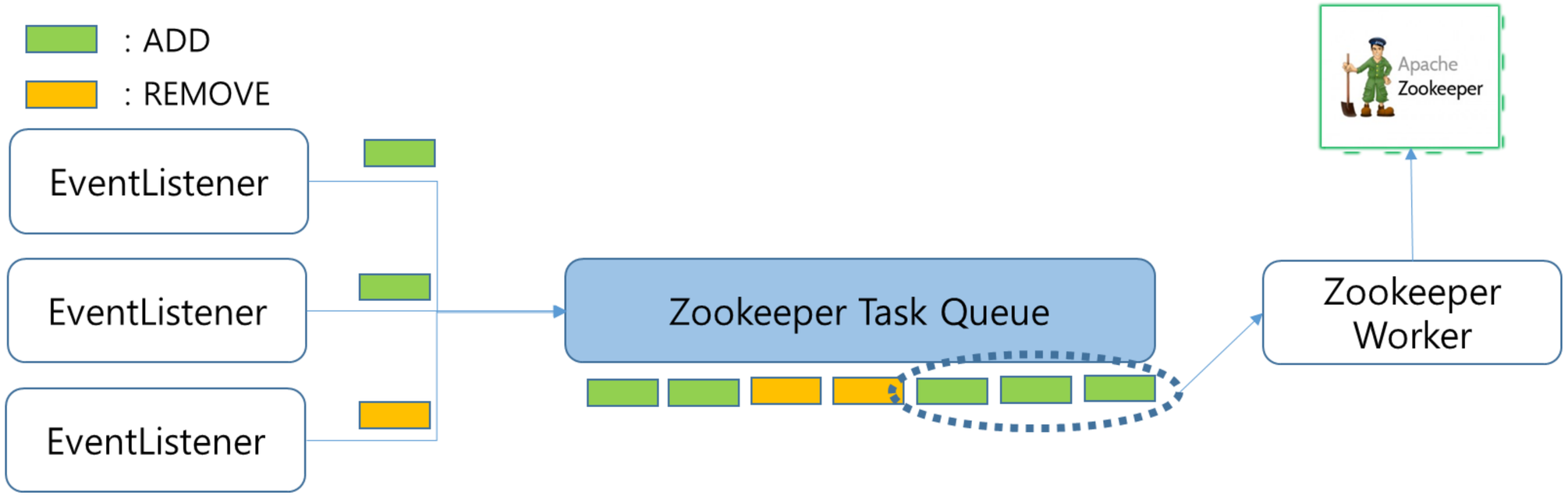
3. 상태 처리 리스너 구현


```
@Override
public void stateUpdated(PinpointServer pinpointServer, SocketStateCode updatedStateCode) {
    logger.info("stateUpdated() started. (PinpointServer={}, updatedStateCode={})", pinpointServer, updatedStateCode);

    Map<Object, Object> channelPropertiesMap = pinpointServer.getChannelProperties();
    ChannelProperties channelProperties = channelPropertiesFactory.newChannelProperties(channelPropertiesMap);

    if (SocketStateCode.RUN_DUPLEX == updatedStateCode) {
        ClusterPoint<byte[]> pinpointServerClusterPoint = newClusterPoint(pinpointServer, channelProperties);
        profilerClusterManager.register(pinpointServerClusterPoint);
    } else if (SocketStateCode.isClosed(updatedStateCode)) {
        ClusterPoint<byte[]> pinpointServerClusterPoint = newClusterPoint(pinpointServer, channelProperties);
        profilerClusterManager.unregister(pinpointServerClusterPoint);
    }
}
```

3. 쥬키퍼 이벤트 저장

- Worker Queue 형태

  쥬키퍼 부하를 줄이기 위해서 같은 형태의 이벤트는 함께 처리



3. 콜렉터 노드 생성

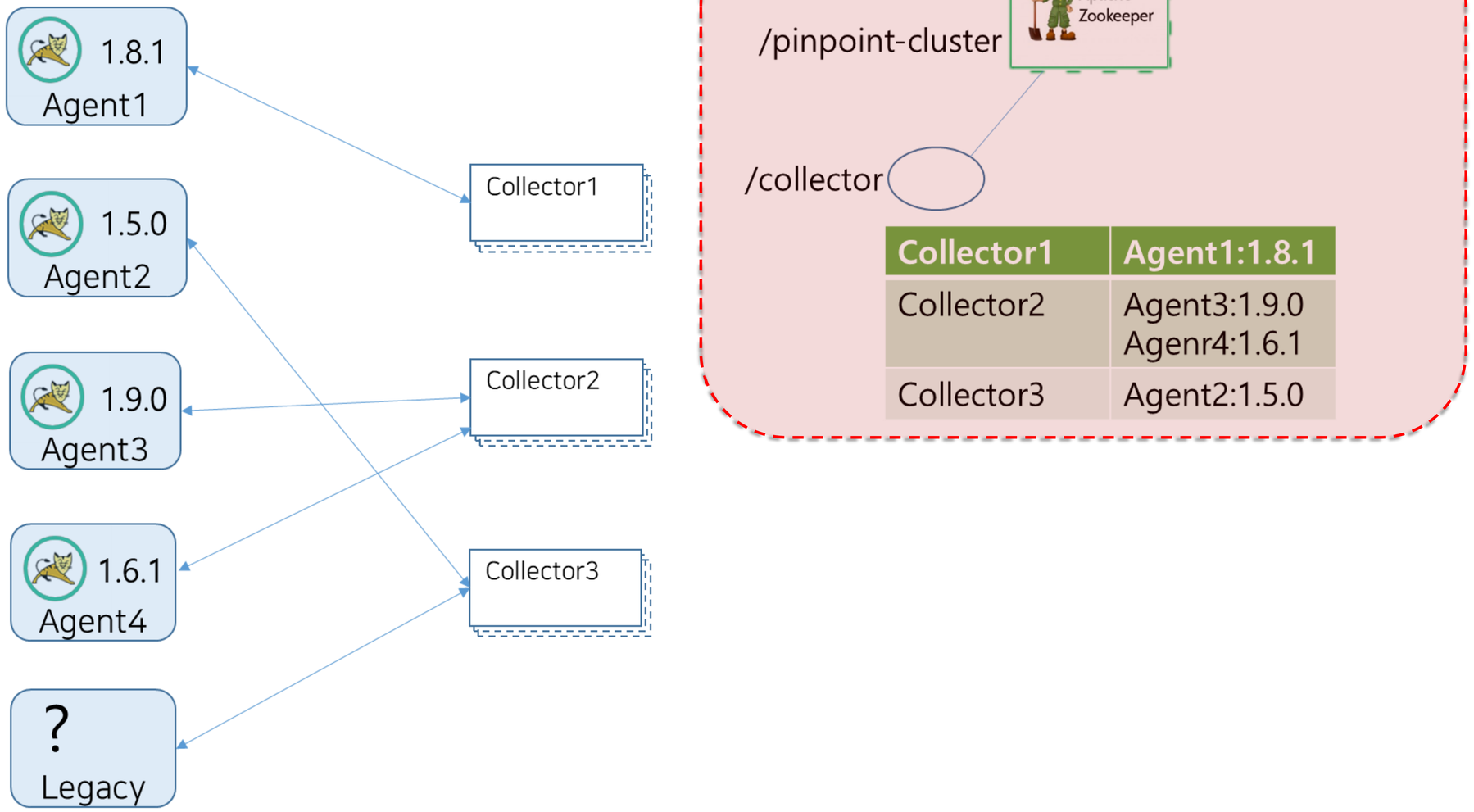
- 연결되어 있는 에이전트의 정보를 저장하기 위한 노드 생성

```
/pinpoint-cluster/collector
     - $serverIdentifier (ephemeral)
                  - $applicationName:$agentId:$startTimeStamp
                  - $applicationName:$agentId:$startTimeStamp
                  - $applicationName:$agentId:$startTimeStamp
     - $serverIdentifier (ephemeral)
                  - $applicationName:$agentId:$startTimeStamp
                  - $applicationName:$agentId:$startTimeStamp
                  - $applicationName:$agentId:$startTimeStamp
```

에이전트 식별자

- 노드 이름 “${pid}@{HostName}”
- 쥬키퍼의 ephemeral 노드 생성
- 노드에 콜렉터와 연결되어있는 모든 에이전트 정보 저장

3. 에이전트 연결 정보 공유 결과



4. 웹 콜렉터 연동

Apache ZooKeeper™

What is ZooKeeper?

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they

쥬키퍼에 웹 장비의 정보를 저장하여 연결정보 공유

4. 웹 노드 생성

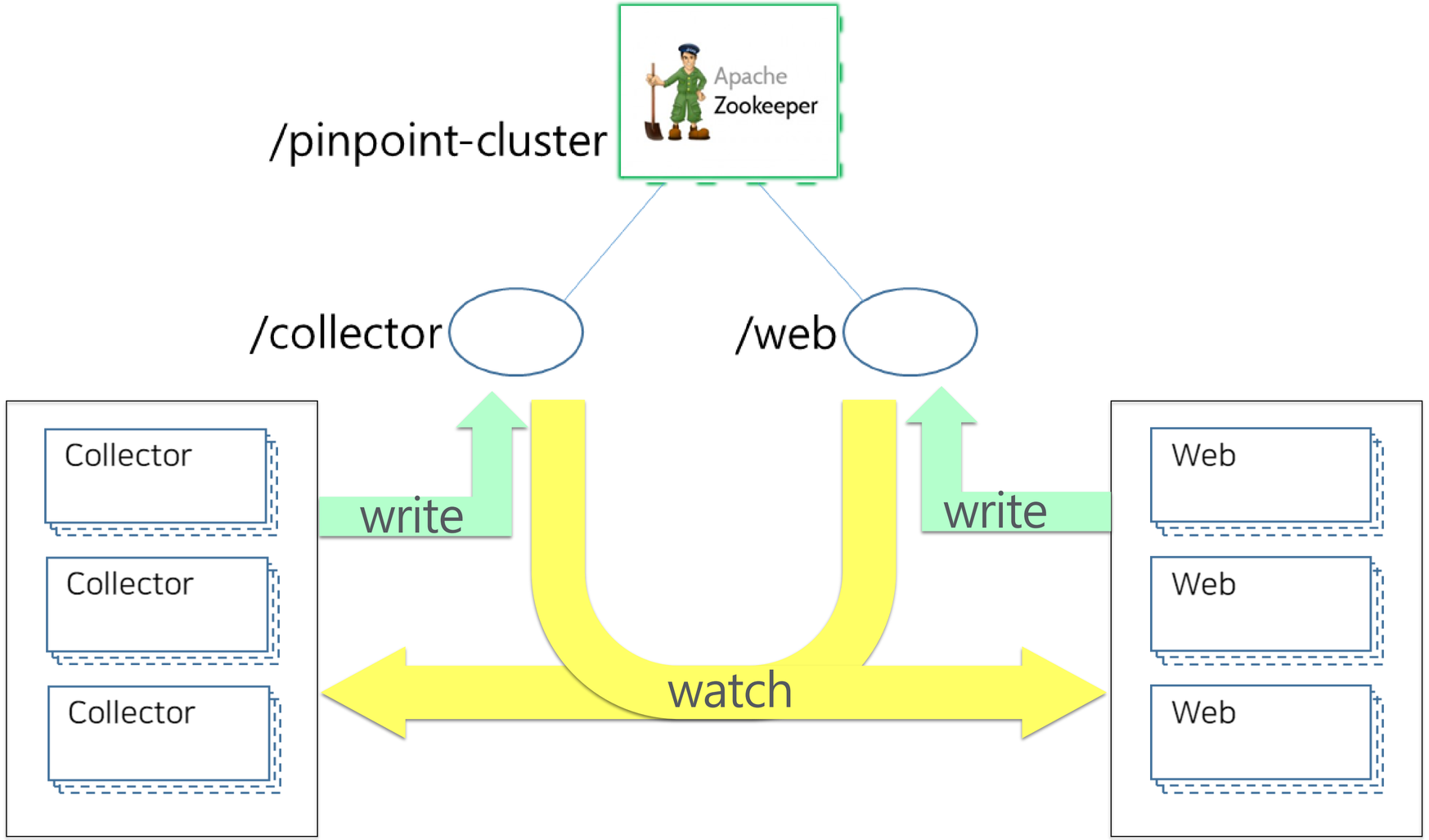
- 웹 모듈의 네트워크 정보 저장을 위한 노드 생성

```
/pinpoint-cluster/web
        -${represtationIp}:port (ephemeral)
                    - $serverAddress1
                    - $serverAddress2
                    - $serverAddress3
                    - ...
        -${represtationIp}:port (ephemeral)
                    - $serverAddress1
                    - $serverAddress2
                    - $serverAddress3
                    - ...
```

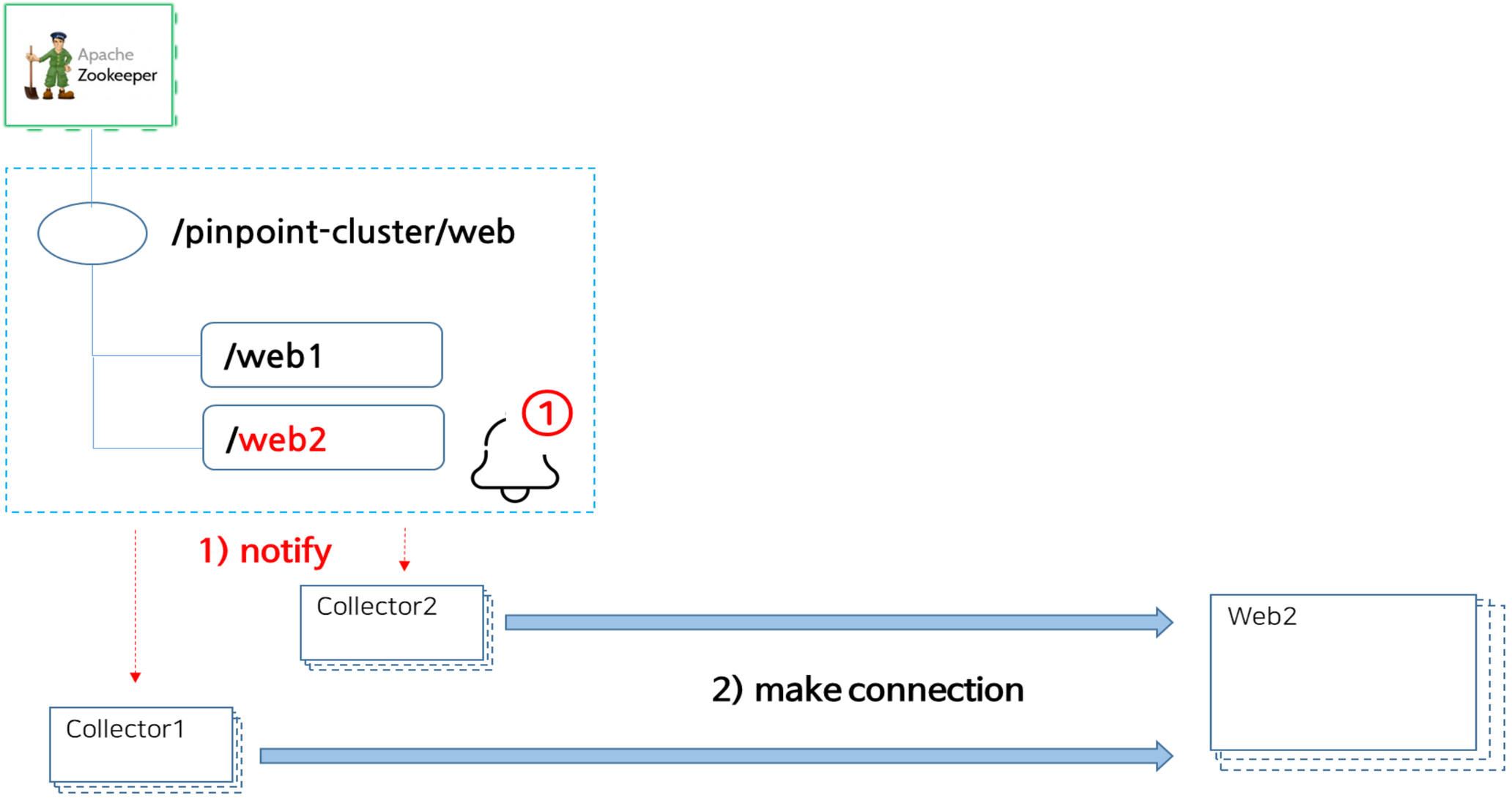
모든 네트워 주소목록

- 노드 이름 “${대표IP}:{Port}”
- 쥬키퍼의 ephemeral 노드 생성
- 내용으로 웹 장비의 모든 네트워 주소 정보 저장

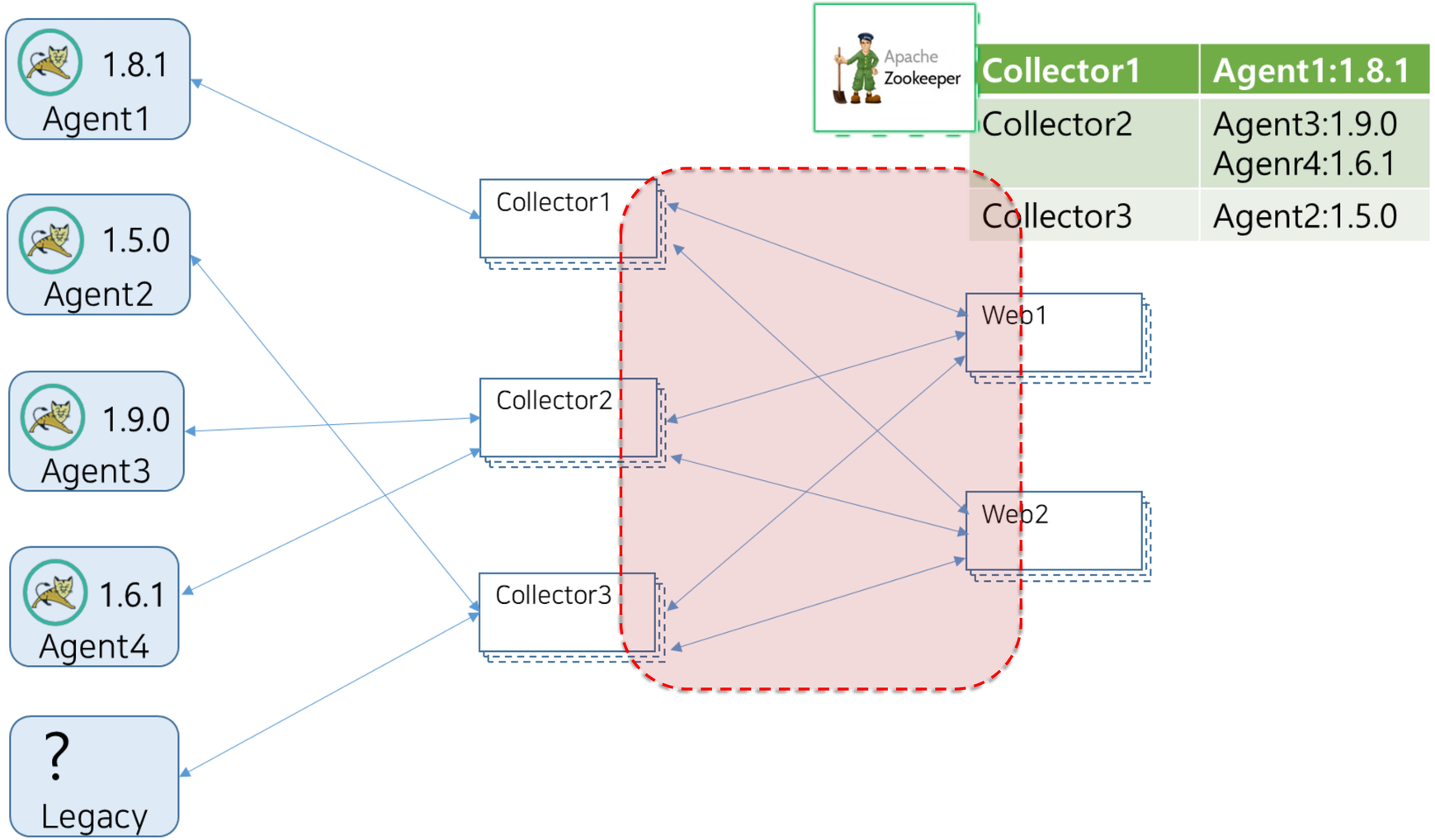
4. 웹 콜렉터간 노드 정보 공유



4. 콜렉터에서 웹으로 연결 구조



4. 웹 콜렉터 연동 결과



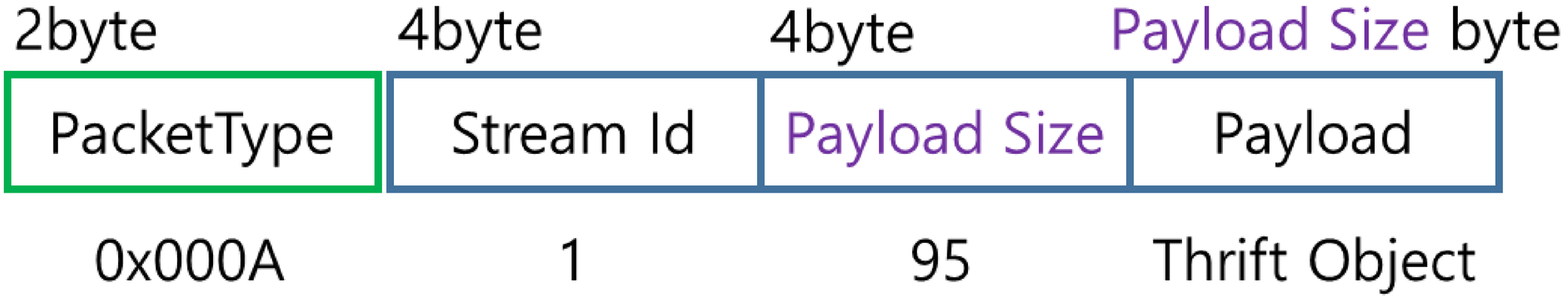
Collector1	Agent1:1.8.1
Collector2	Agent3:1.9.0 Agenr4:1.6.1
Collector3	Agent2:1.5.0

5. Req/Res 한계점

- 현재의 구조에서 ActiveThread를 가져오려면

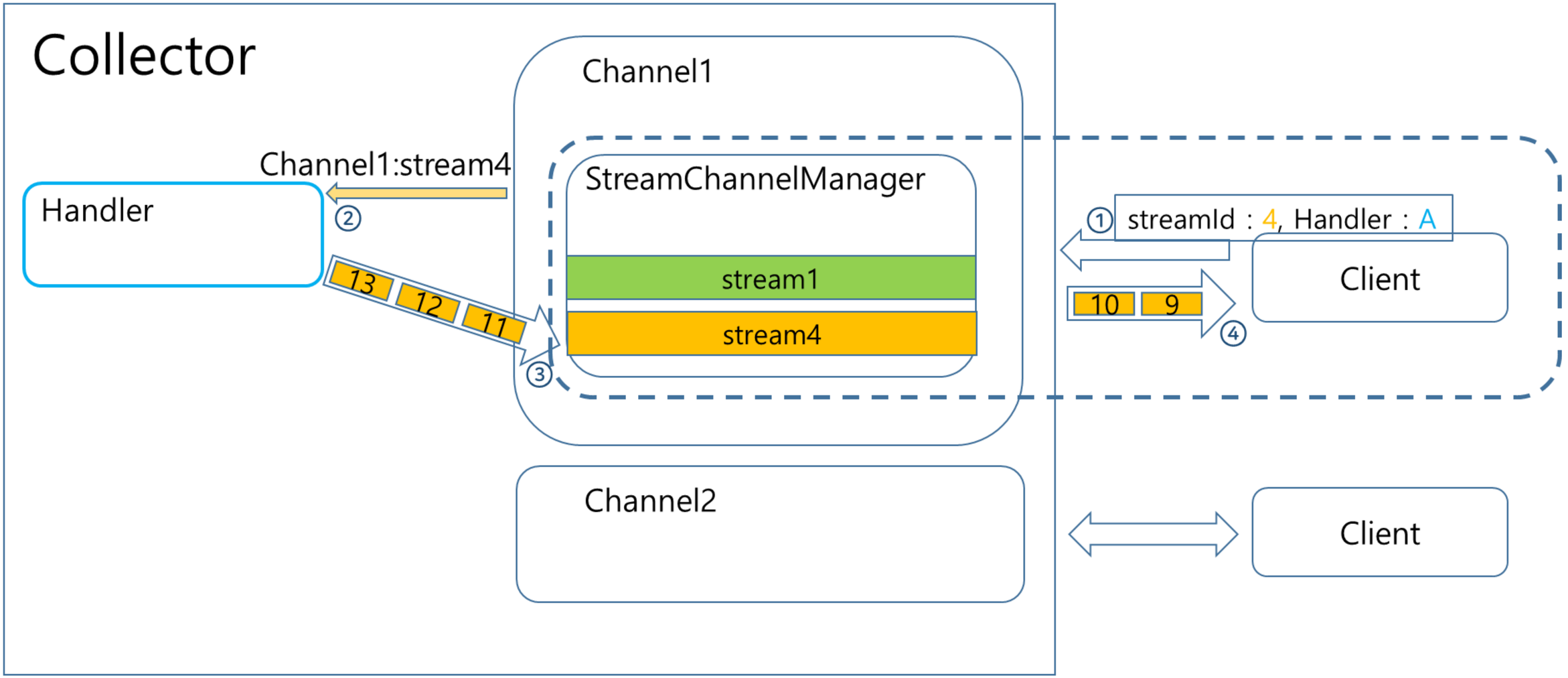
5. StreamPacket 추가

- StreamCreatePacket



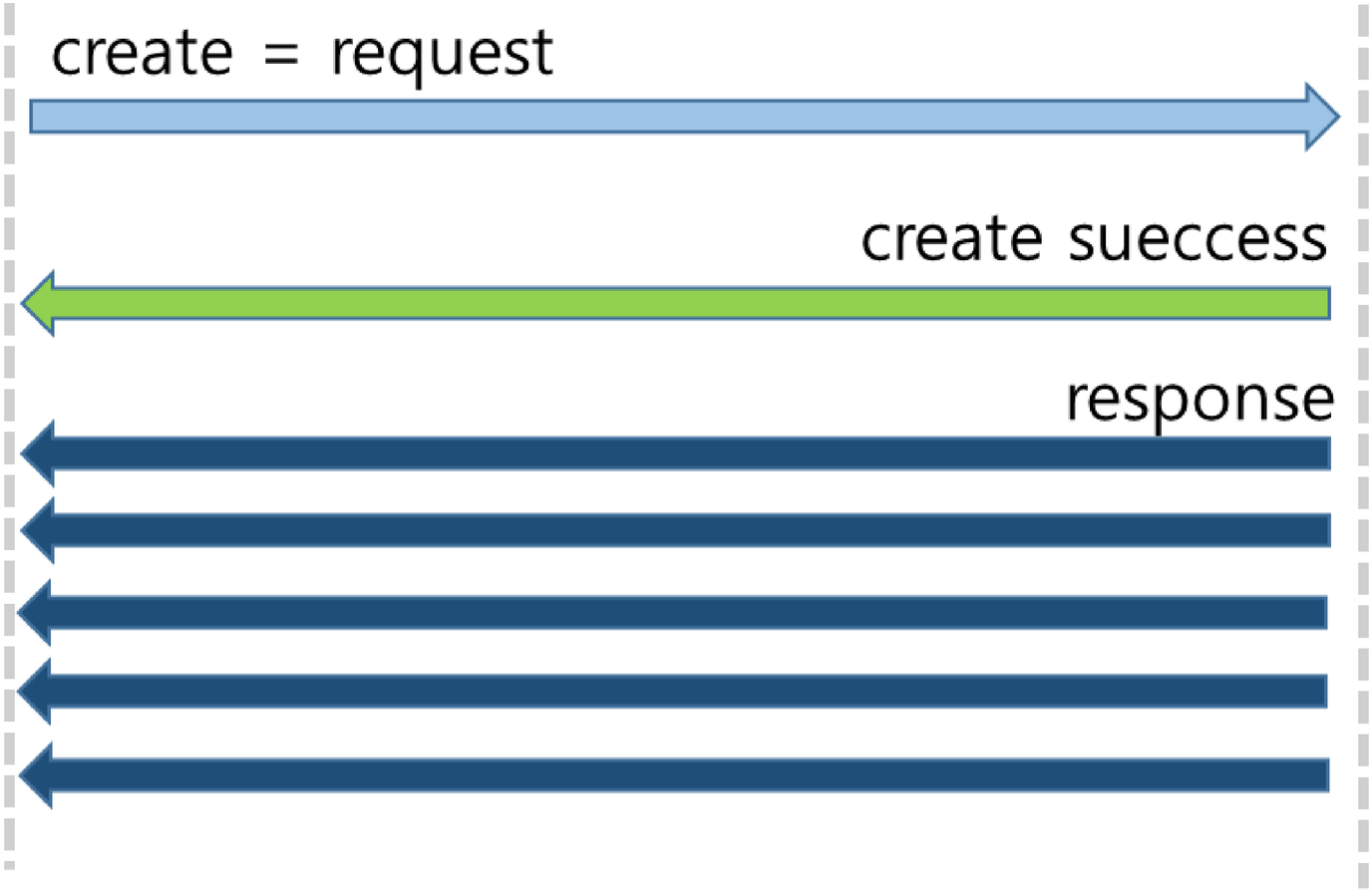
- StreamCreateSuccessPacket
- StreamCreateFailPacket
- StreamResponsePacket

5. Stream 동작 구조

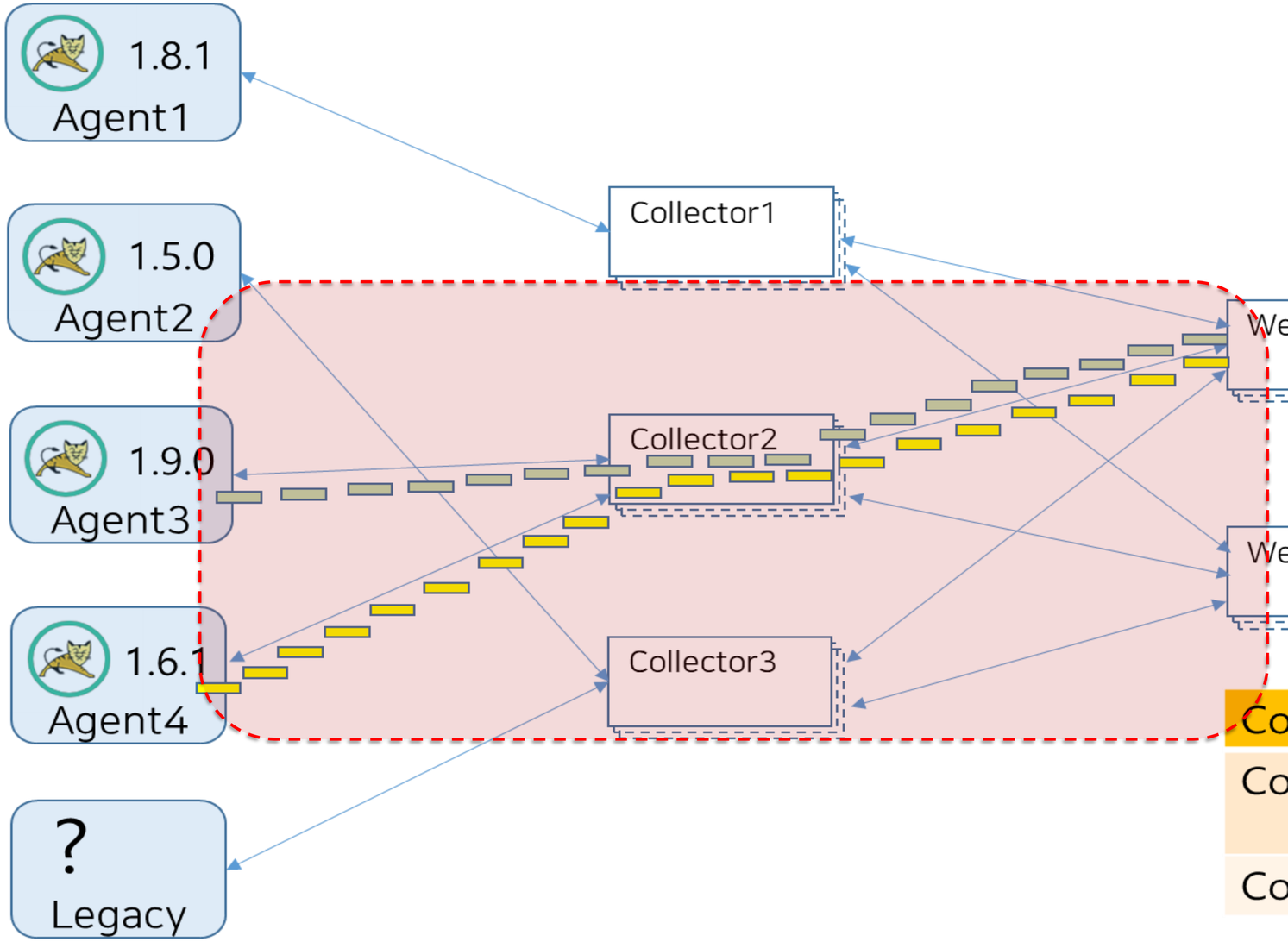


5. Stream 기능 지원 결과

- 에이전트의 부담이 줄어든 1 Req / Multi Res



결과



Collector1	Agent1:1.8.1
Collector2	Agent3:1.9.0 Agenr4:1.6.1
Collector3	Agent2:1.5.0

DEVIEW
2019

3. 효율적으로 observability 정보 저장하기

해결해야 하는 문제

사용자 늘어나

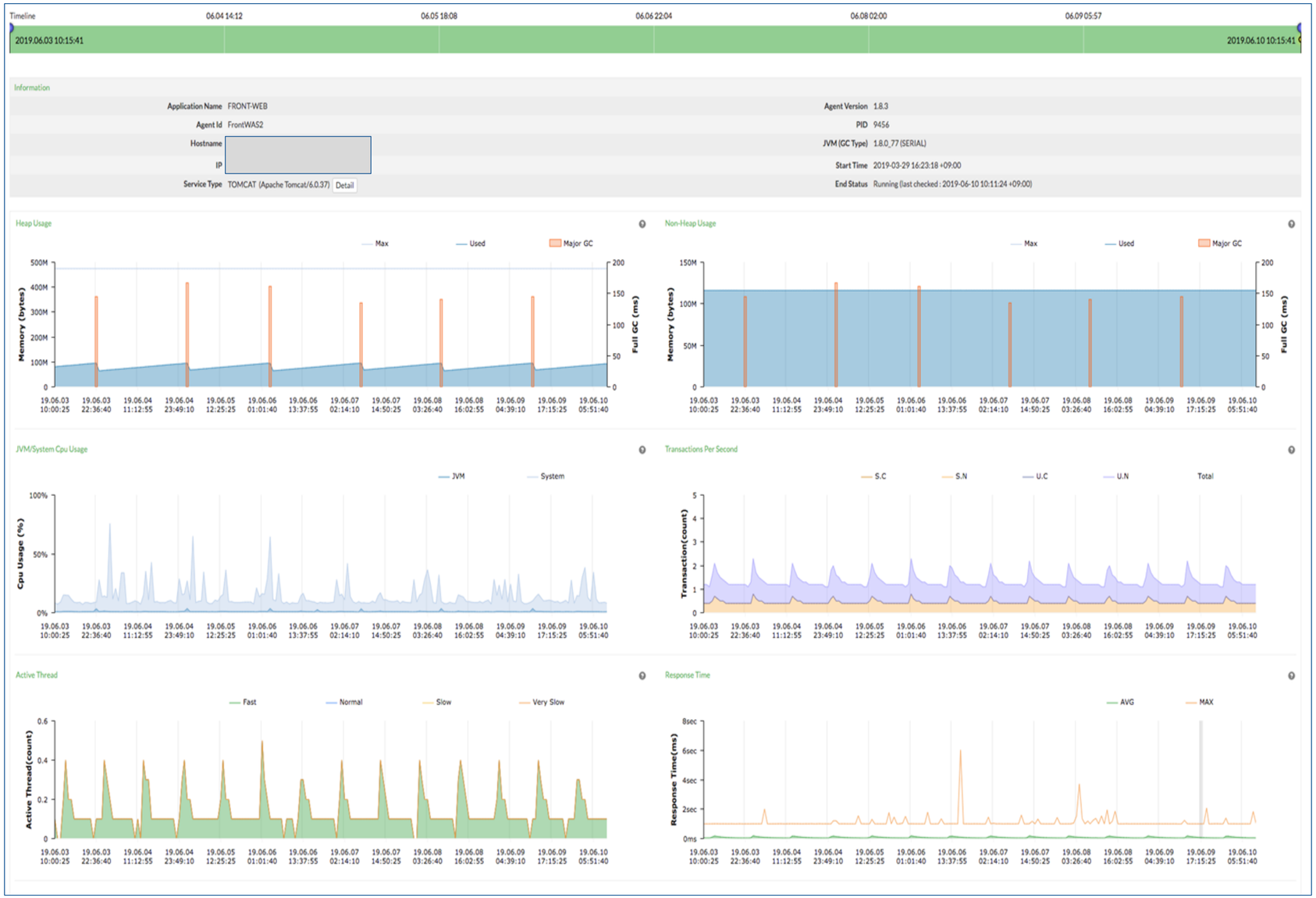
⇒저장공간 부족

⇒네트워크 대역 부족

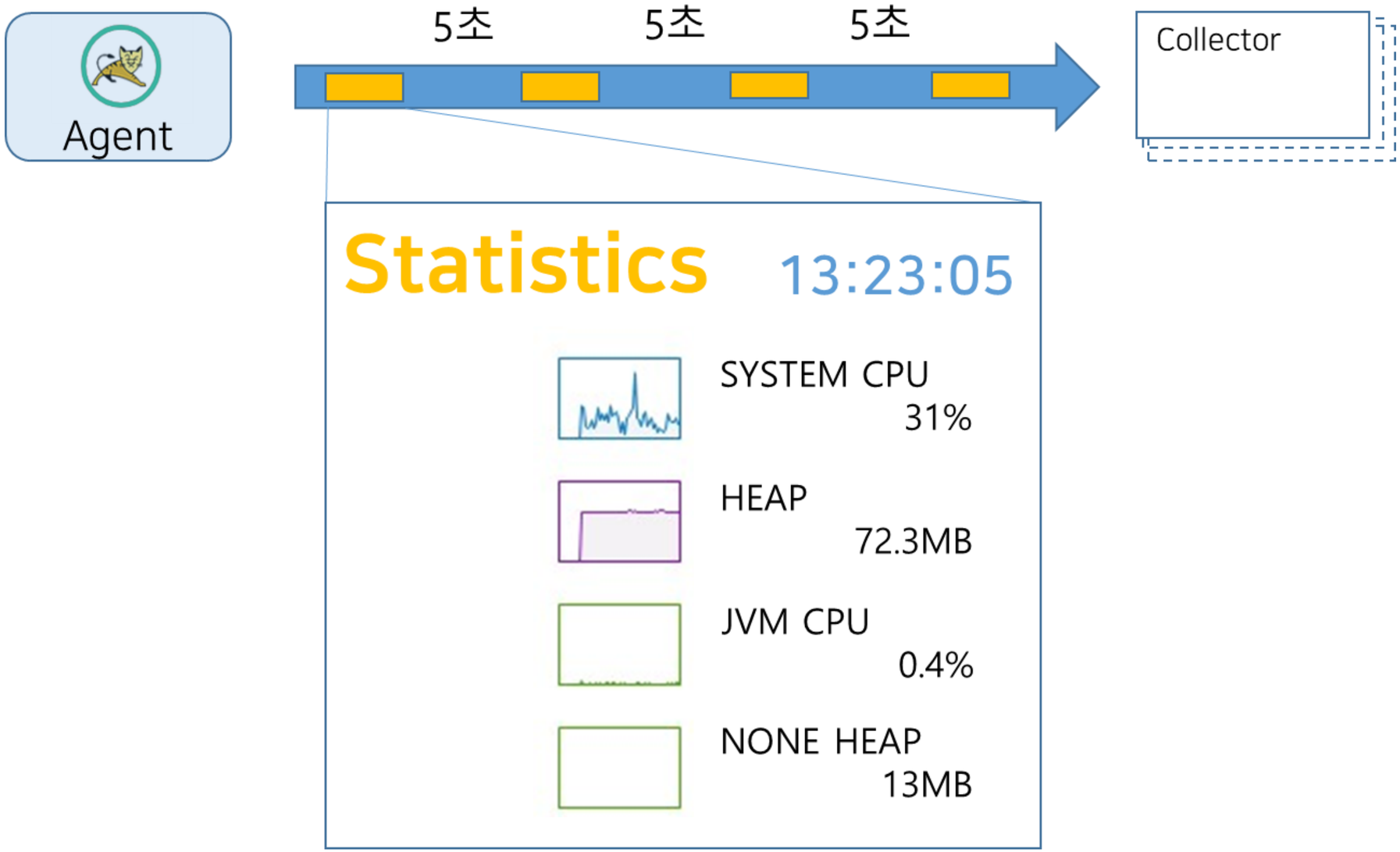
⇒조회 시간 증가

必 개선 필요

1. Inspector 개선



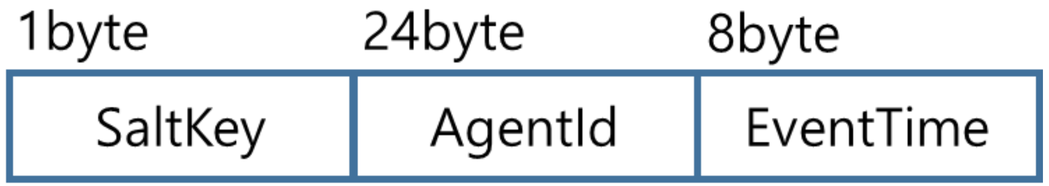
1. 데이터 전달 구조


Agent
5초
5초
5초
Collector
Statistics
13:23:05
SYSTEM CPU
31%
HEAP
72.3MB
JVM CPU
0.4%
NONE HEAP
13MB

1. 기존 데이터 저장방식

- 5초 단위로 데이터를 받고 이를 즉시 Hbase에 저장

rowkey :

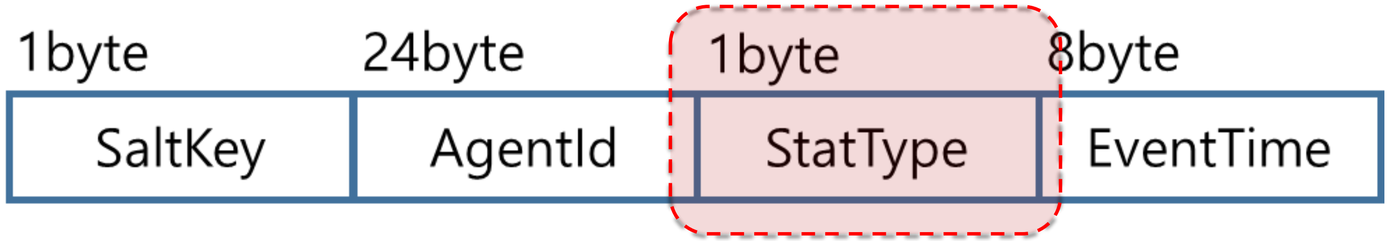


value :

1. 데이터 저장방식 변경

- 5분 단위 저장

- 각각의 메트릭 정보를 나누어서 저장 (메트릭 6개)



1. 데이터 저장방식 변경 중간 결과

DEVIEW 2019

- rowkey 갯수 (5분 기준, 메트릭 6개)

  60 -> 6

- 조회속도

  parallel 요청 가능 (성능개선)

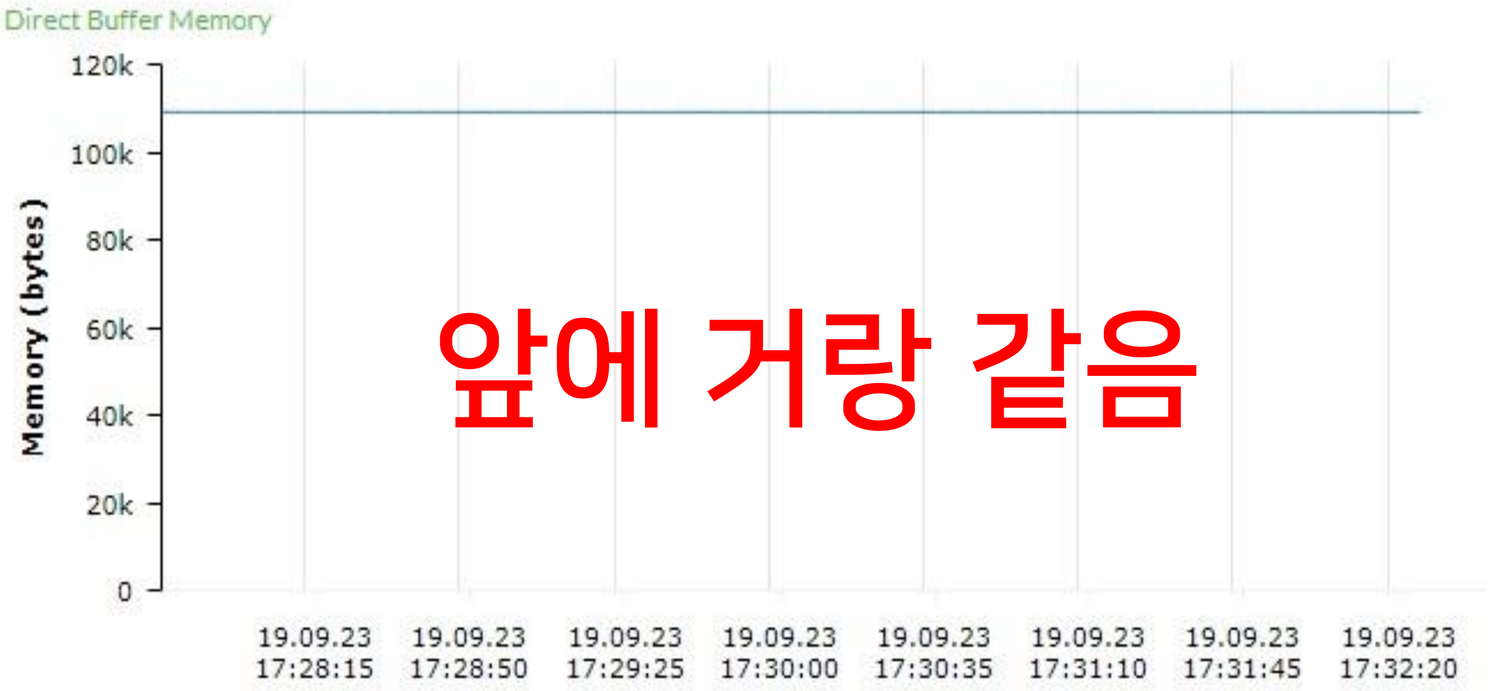
Type을 나누고 시간을 뭉쳐보니

1. 데이터를 가만히 바라보니

DEVIEW
2019

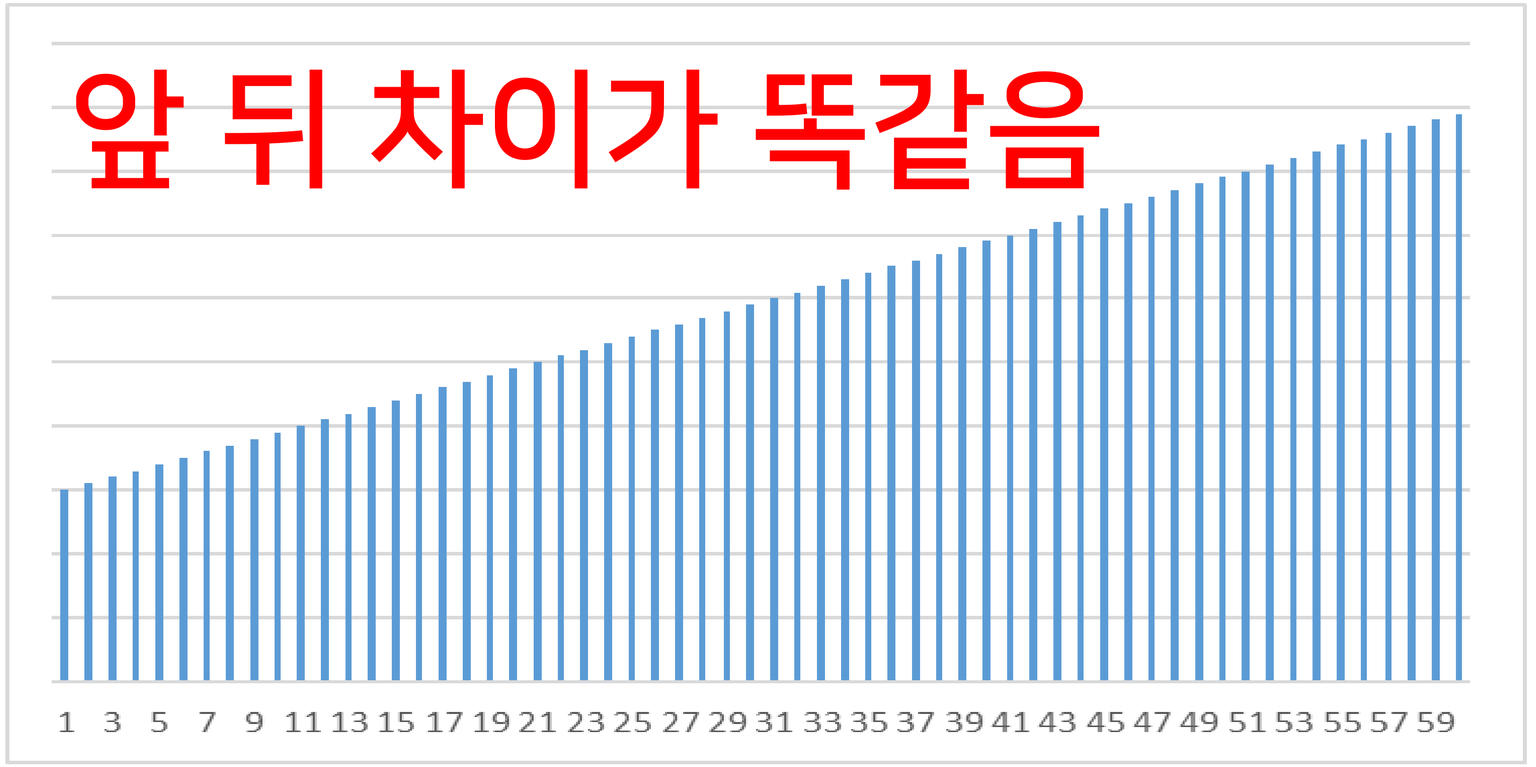
1. 데이터를 가만히 바라보니

DEVIEW
2019



1. 데이터를 가만히 바라보니

DEVIEW
2019



1. Dynamic Encoding 지원

- Delta
- Delta of Delta
- Repeat
- SAME
- RAW
- ...

1. 데이터가 들어오면


이벤트 시간
힙 사용량
최대 힙용량
Collector
6시 5분 35초 : 300m : 2g
6시 5분 40초 : 301m : 2g
6시 5분 45초 : 303m : 2g
6시 5분 50초 : 290m : 2g
6시 5분 55초 : 291m : 2g
6시 6분 00초 : 300m : 2g

1. 데이터가 들어오면 (기존)

rowkey	heapUsed	heapMax	
salt(1)agentId(24) 1497029500000(8)	hpU(3):300,000,000(5)	hpM(3):2,000,000,000(5)	49 bytes
salt(1)agentId(24) 1497029505000(8)	hpU(3):301,000,000(5)	hpM(3):2,000,000,000(5)	49 bytes
salt(1)agentId(24) 1497029510000(8)	hpU(3):303,000,000(5)	hpM(3):2,000,000,000(5)	49 bytes
salt(1)agentId(24) 1497029515000(8)	hpU(3):290,000,000(5)	hpM(3):2,000,000,000(5)	49 bytes
salt(1)agentId(24) 1497029520000(8)	hpU(3):291,000,000(5)	hpM(3):2,000,000,000(5)	49 bytes
salt(1)agentId(24) 1497029525000(8)	hpU(3):300,000,000(5)	hpM(3):2,000,000,000(5)	49 bytes

Before : 294 bytes

이벤트시간 <=

힙사용량 <=

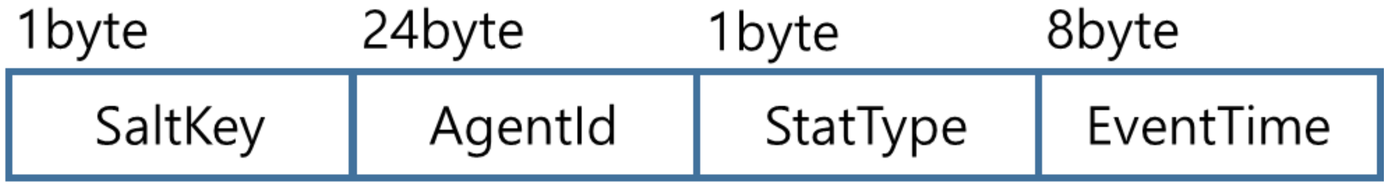
힙맥스값 <=

294B

1. 데이터가 들어오면 (변경)

DEVIEW 2019

- Rowkey :



34 bytes

- 이벤트 시간 : DELTA_OF_DELTA(1) 5000(2) 0000(4) 7 bytes

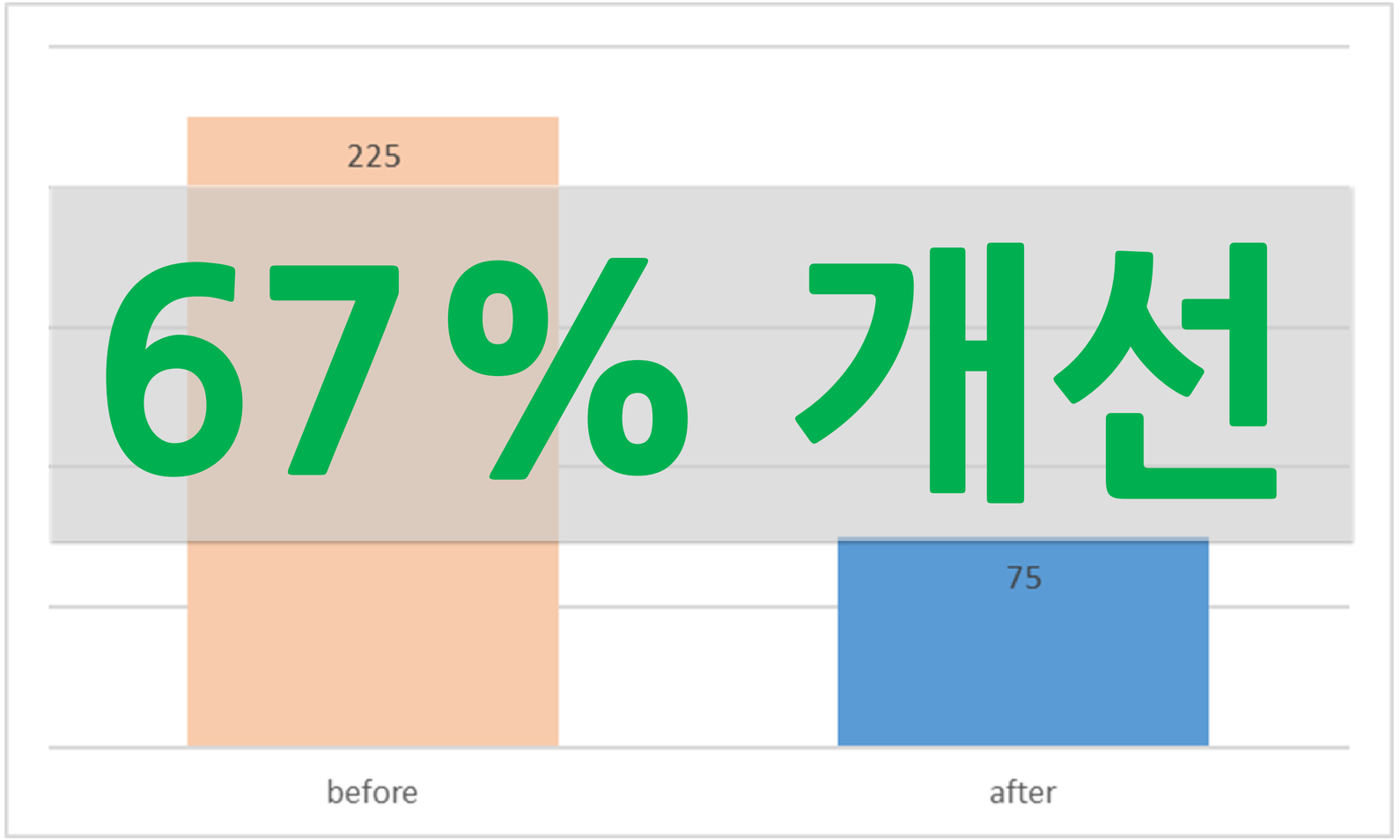
- 힙사용량 : DELTA(1) 300m(5) 1m(3) 2m(4) 24 bytes
  -13m(4) 1m(3) 9m(4)

- 최대힙량 : REPEAT(1) 2g(5) 6(1) 7 bytes

72B

1. 데이터가 들어오면 (결과)

1. 저장용량 개선



일부 에이전트 데이터 1주일간 DualWrite로 저장 후 비교

1. 조회시간 개선

DEVIEW
2019

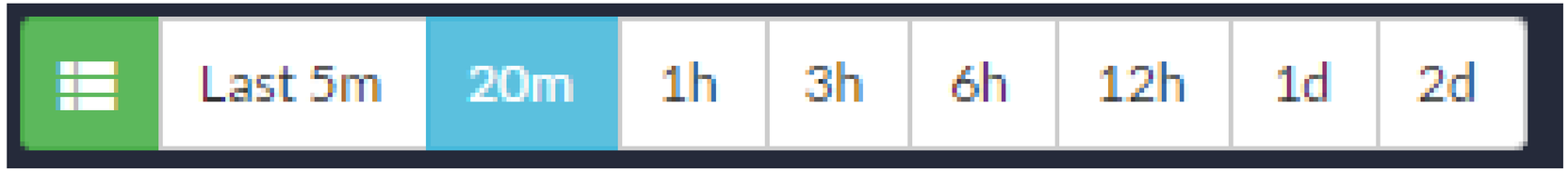
- 로컬 기준 2일치 (기존 Max값)

  3.5 sec -> 350 ms

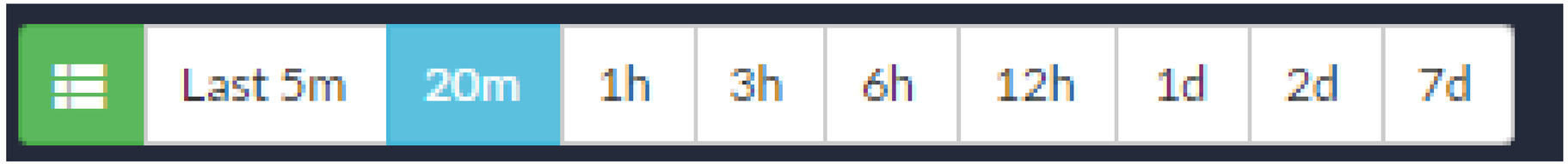
90% 개선

1. 더 긴 조회시간 추가

DEVIEW
2019

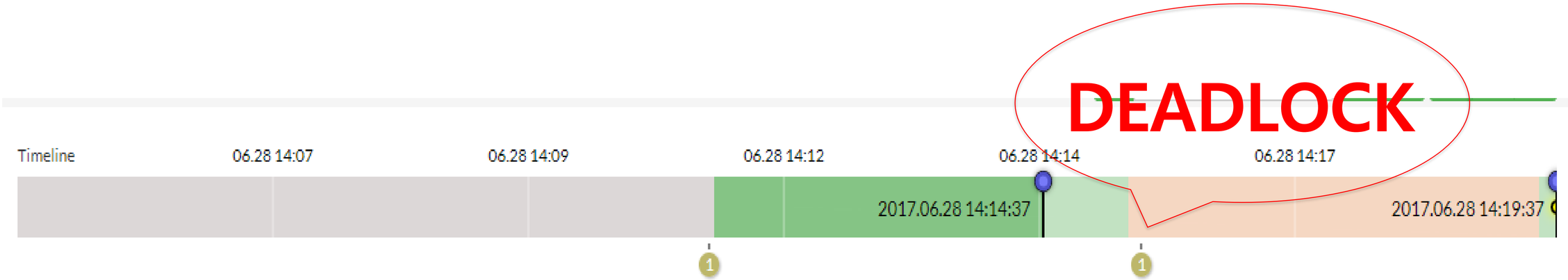


+ 7d



1. 서버 타임라인 이벤트 추가

DEVIEW 2019



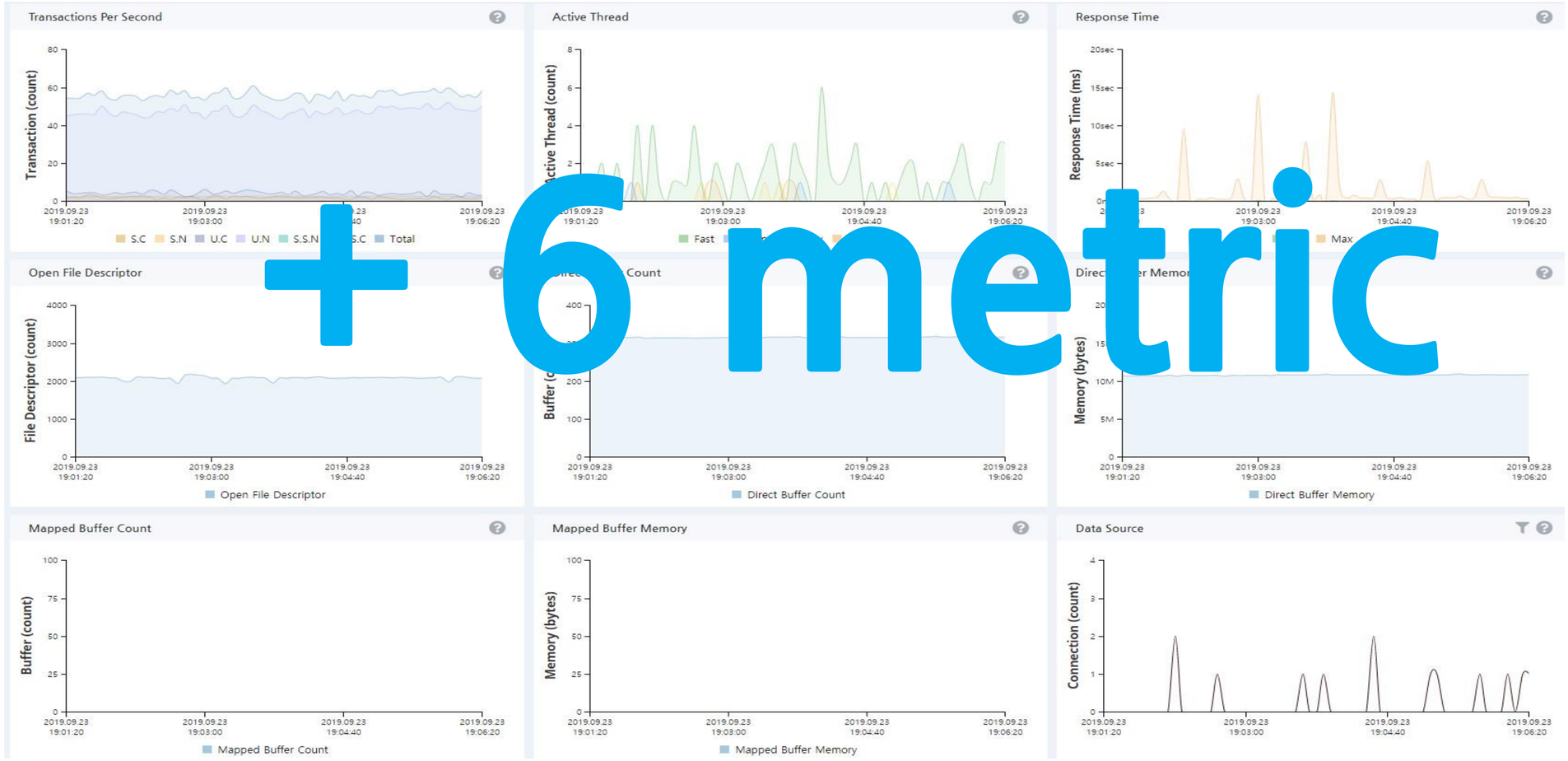
Event - 1

Time	Description	Message
2017-06-28 14:15:27 +09:00	Agent deadlock detected	"DEADLOCK THREAD-1" Id=0x9c BLOCKED on java.lang.Object@1dbc0490 owned by "DEADLOCK THREAD-2" Id=157 at com.navercorp.test.pinpoint.testweb.controller.ThreadDeadLockController$Thread1.run(ThreadDeadLockController.java:85) - blocked on java.lang.Object@1dbc0490 "DEADLOCK THREAD-2" Id=0x9d BLOCKED on java.lang.Object@5246ea93 owned by "DEADLOCK THREAD-1" Id=156 at com.navercorp.test.pinpoint.testweb.controller.ThreadDeadLockController$Thread1.run(ThreadDeadLockController.java:85) - blocked on java.lang.Object@5246ea93

1. Inspector 개선 결과

DEVIEW 2019

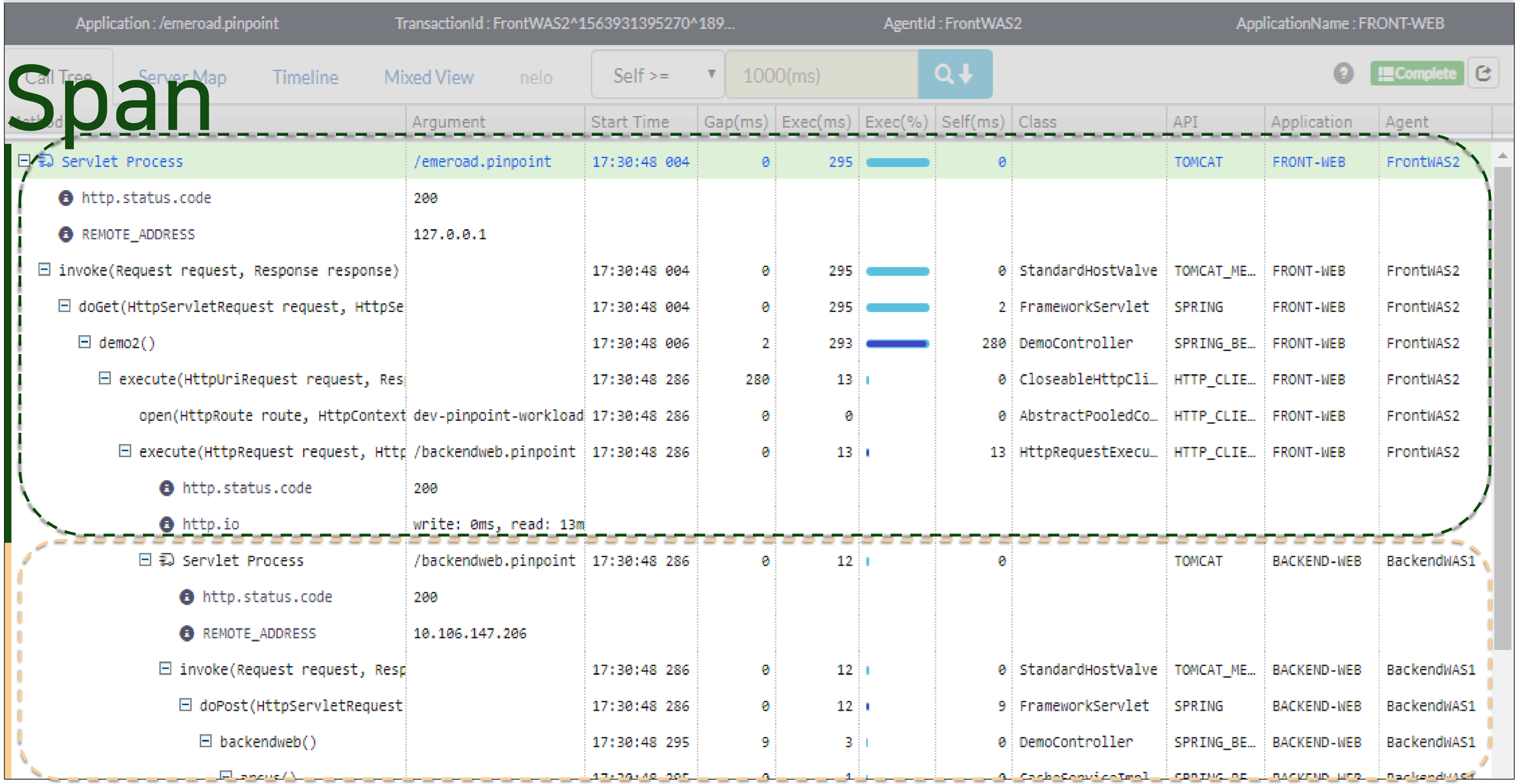
- DataSource, ResponseTime, OpenFD, ByteBuffer등 추가



2. Trace 개선

DEVIEW 2019

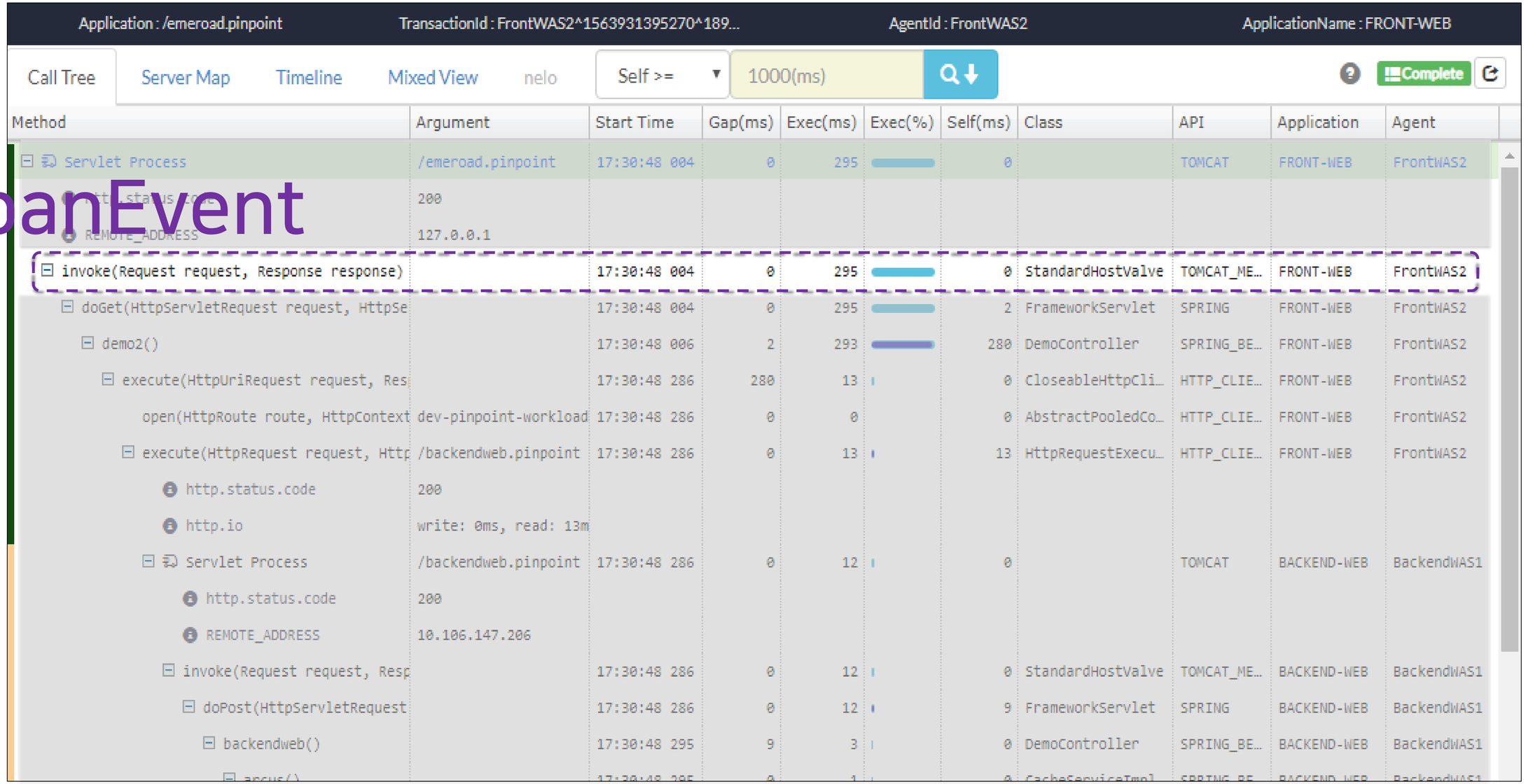
Span



2. SpanEvent란?

DEVIEW 2019

SpanEvent



2. 저장 단위 (기존)

DEVIEW 2019

- 기존 : SpanEvent 단위로 저장

	Argument	Start Time	Gap(ms)	Exec(m...	Class	API	Agent	Application
Servlet Process	/backendapi.pinpoint	11:00:17 752	0	5		TOMCAT	ApiWAS1	BACKEND-API
http.status.code	200							
REMOTE_ADDRESS								
invoke(Request request, Response response)		11:00:17 752	0	5	StandardHostValve	TOMCAT_METHOD	ApiWAS1	BACKEND-API
doPost(HttpServletRequest request, HttpServletRe		11:00:17 752	0	5	FrameworkServlet	SPRING	ApiWAS1	BACKEND-API
backendapi()		11:00:17 753	1	3	DemoController	SPRING_BEAN	ApiWAS1	BACKEND-API
getConnection()		11:00:17 753	0	0	BasicDataSource	DBCP2	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	false	11:00:17 753	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberServiceImpl	SPRING_BEAN	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberDaoJdbc	SPRING_BEAN	ApiWAS1	BACKEND-API
executeQuery(String sql)		11:00:17 753	0	1	StatementImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
SQL	/* testquery */ select * from member							
commit()		11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	true	11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
close()		11:00:17 754	0	0	PoolingDataSource...	DBCP2	ApiWAS1	BACKEND-API

2. 저장 단위 (변경)

DEVIEW 2019

- SpanChunk(임의의 SpanEvent 묶음) 단위로 저장

	Argument	Start Time	Gap(ms)	Exec(m...	Class	API	Agent	Application
Servlet Process	/backendapi.pinpoint	11:00:17 752	0	5		TOMCAT	ApiWAS1	BACKEND-API
http.status.code	200							
REMOTE_ADDRESS								
invoke(Request request, Response response)		11:00:17 752	0	5	StandardHostValve	TOMCAT_METHOD	ApiWAS1	BACKEND-API
doPost(HttpServletRequest request, HttpServletRe		11:00:17 752	0	5	FrameworkServlet	SPRING	ApiWAS1	BACKEND-API
backendapi()		11:00:17 753	1	3	DemoController	SPRING_BEAN	ApiWAS1	BACKEND-API
getConnection()		11:00:17 753	0	0	BasicDataSource	DBCP2	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	false	11:00:17 753	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberServiceImpl	SPRING_BEAN	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberDaoJdbc	SPRING_BEAN	ApiWAS1	BACKEND-API
executeQuery(String sql)		11:00:17 753	0	1	StatementImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
SQL	/* testquery */ select * from member							
commit()		11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	true	11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
close()		11:00:17 754	0	0	PoolingDataSource...	DBCP2	ApiWAS1	BACKEND-API

- rowkey 갯수 : 10 -> 1

가까운 SpanEvent를 뭉쳐보니

2. 데이터를 또 가만히 바라보니

DEVIEW 2019

시작시간이 같거나 비슷

	Argument	Start Time	Gap(ms)	Exec(m...	Class	API	Agent	Application
Servlet Process	/backendapi.pinpoint	11:00:17 752	0	5		TOMCAT	ApiWAS1	BACKEND-API
http.status.code	200							
REMOTE_ADDRESS								
invoke(Request request, Response response)		11:00:17 752	0	5	StandardHostValve	TOMCAT_METHOD	ApiWAS1	BACKEND-API
doPost(HttpServletRequest request, HttpServletRe		11:00:17 752	0	5	FrameworkServlet	SPRING	ApiWAS1	BACKEND-API
backendapi()		11:00:17 753	1	3	DemoController	SPRING_BEAN	ApiWAS1	BACKEND-API
getConnection()		11:00:17 753	0	0	BasicDataSource	DBCP2	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	false	11:00:17 753	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberServiceImpl	SPRING_BEAN	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberDaoJdbc	SPRING_BEAN	ApiWAS1	BACKEND-API
executeQuery(String sql)		11:00:17 753	0	1	StatementImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
SQL	/* testquery */ select * from member							
commit()		11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	true	11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
close()		11:00:17 754	0	0	PoolingDataSource...	DBCP2	ApiWAS1	BACKEND-API

2. 데이터를 또 가만히 바라보니

DEVIEW 2019

	Argument	Start Time	Gap(ms)	Exec(m...	Class	API	Agent	Application
Servlet Process	/backendapi.pinpoint	11:00:17 752	0	5		TOMCAT	ApiWAS1	BACKEND-API
http.status.code	200							
REMOTE_ADDRESS								
invoke(Request request, Response response)		11:00:17 752	0	5	StandardHostValve	TOMCAT_METHOD	ApiWAS1	BACKEND-API
doPost(HttpServletRequest request, HttpServletRe		11:00:17 752	0	5	FrameworkServlet	SPRING	ApiWAS1	BACKEND-API
backendapi()		11:00:17 753	1	3	DemoController	SPRING_BEAN	ApiWAS1	BACKEND-API
getConnection()		11:00:17 753	0	0	BasicDataSource	DBCP2	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	false	11:00:17 753	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberServiceImpl	SPRING_BEAN	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberDaoJdbc	SPRING_BEAN	ApiWAS1	BACKEND-API
executeQuery(String sql)		11:00:17 753	0	1	StatementImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
SQL	/* testQuery */ select * from member							
commit()		11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	true	11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
close()		11:00:17 754	0	0	PoolingDataSource...	DBCP2	ApiWAS1	BACKEND-API

API 타입이 앞에 것과 같은 경우 많음

2. 데이터를 또 가만히 바라보니

DEVIEW 2019

	Argument	Start Time	Gap(ms)	Exec(m...	Class	API	Agent	Application
Servlet Process	/backendapi.pinpoint	11:00:17 752	0	5		TOMCAT	ApiWAS1	BACKEND-API
http.status.code	200							
REMOTE_ADDRESS								
invoke(Request request, Response response)		11:00:17 752	0	5	StandardHostValve	TOMCAT_METHOD	ApiWAS1	BACKEND-API
doPost(HttpServletRequest request, HttpServletRe		11:00:17 752	0	5	FrameworkServlet	SPRING	ApiWAS1	BACKEND-API
backendapi()		11:00:17 753	1	3	emController	SPRING_BEAN	ApiWAS1	BACKEND-API
getConnection()		11:00:17 753	0	0	BasicDataSource	DBCP2	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	false	11:00:17 753	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
list()		11:00:17 753	0		MemberServiceImpl	SPRING_BEAN	ApiWAS1	BACKEND-API
list()		11:00:17 753	0		MemberDaoJdbc	SPRING_BEAN	ApiWAS1	BACKEND-API
executeQuery(String sql)		11:00:17 753	0	1	StatementImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
SQL	/* testquery */ select * from member							
commit()		11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	true	11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
close()		11:00:17 754	0	0	PoolingDataSource...	DBCP2	ApiWAS1	BACKEND-API

Argument가 없는 경우 많음

2. 데이터를 또 가만히 바라보니

DEVIEW 2019

	Argument	Start Time	Gap(ms)	Exec(m...	Class	API	Agent	Application
Servlet Process	/backendapi.pinpoint	11:00:17 752	0	5		TOMCAT	ApiWAS1	BACKEND-API
http.status.code	200							
REMOTE_ADDRESS								
invoke(Request request, Response response)		11:00:17 752	0	5	StandardHostValve	TOMCAT_METHOD	ApiWAS1	BACKEND-API
doPost(HttpServletRequest request, HttpServletRe		11:00:17 752	0	5	FrameworkServlet	SPRING	ApiWAS1	BACKEND-API
backendapi()		11:00:17 753	1	3	DemoController	SPRING_BEAN	ApiWAS1	BACKEND-API
getConnection()		11:00:17 753	0	0	BasicDataSource	DBCP2	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	false	11:00:17 753	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberServiceImpl	SPRING_BEAN	ApiWAS1	BACKEND-API
list()		11:00:17 753	0	1	MemberDaoJdbc	SPRING_BEAN	ApiWAS1	BACKEND-API
executeQuery(String sql)		11:00:17 753	0	1	StatementImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
SQL	/* testquery */ select * from member							
commit()		11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
setAutoCommit(boolean autoCommitFlag)	true	11:00:17 754	0	0	ConnectionImpl	MYSQL(pinpoi...	ApiWAS1	BACKEND-API
close()		11:00:17 754	0	0	PoolingDataSource...	DBCP2	ApiWAS1	BACKEND-API

depth가 같은 경우가 많음

2. 필드별 자주 사용하는 규칙 적용

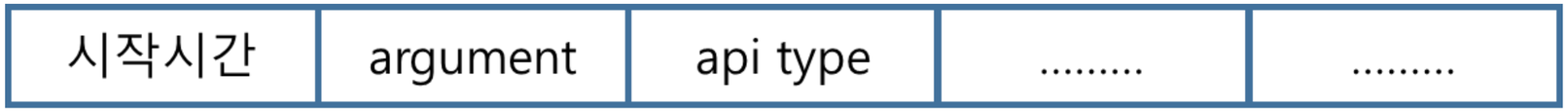
DEVIEW
2019

- 시작시간 (Equal or Delta)
- Argument (Non exist or Raw)
- Api Type (Equal or Raw)
- Depth (Equal or Raw)
- ...

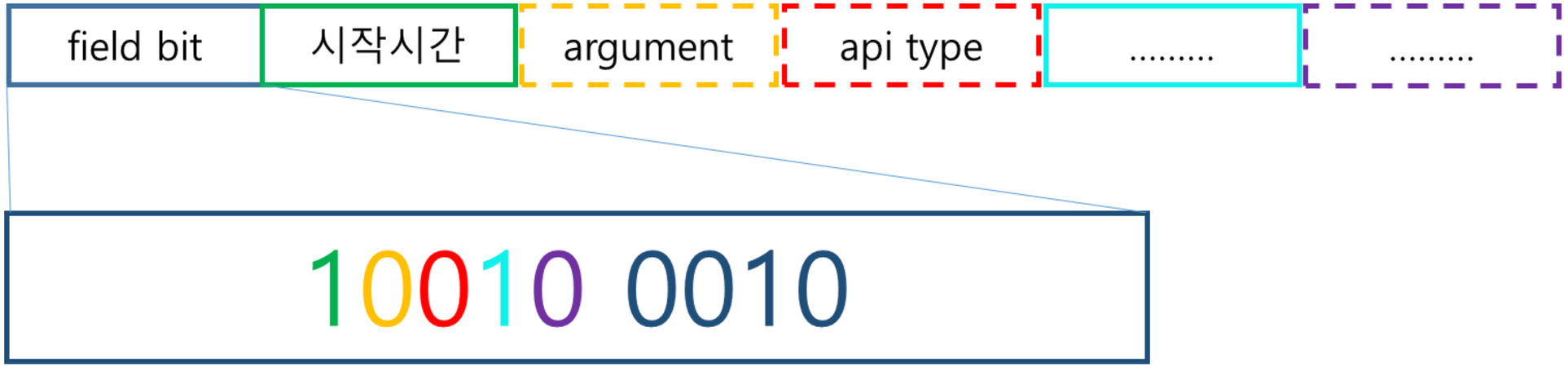
2. 규칙 + 비트패킹 적용

DEVIEW 2019

- 기존 : 모든 필드 정보 기록

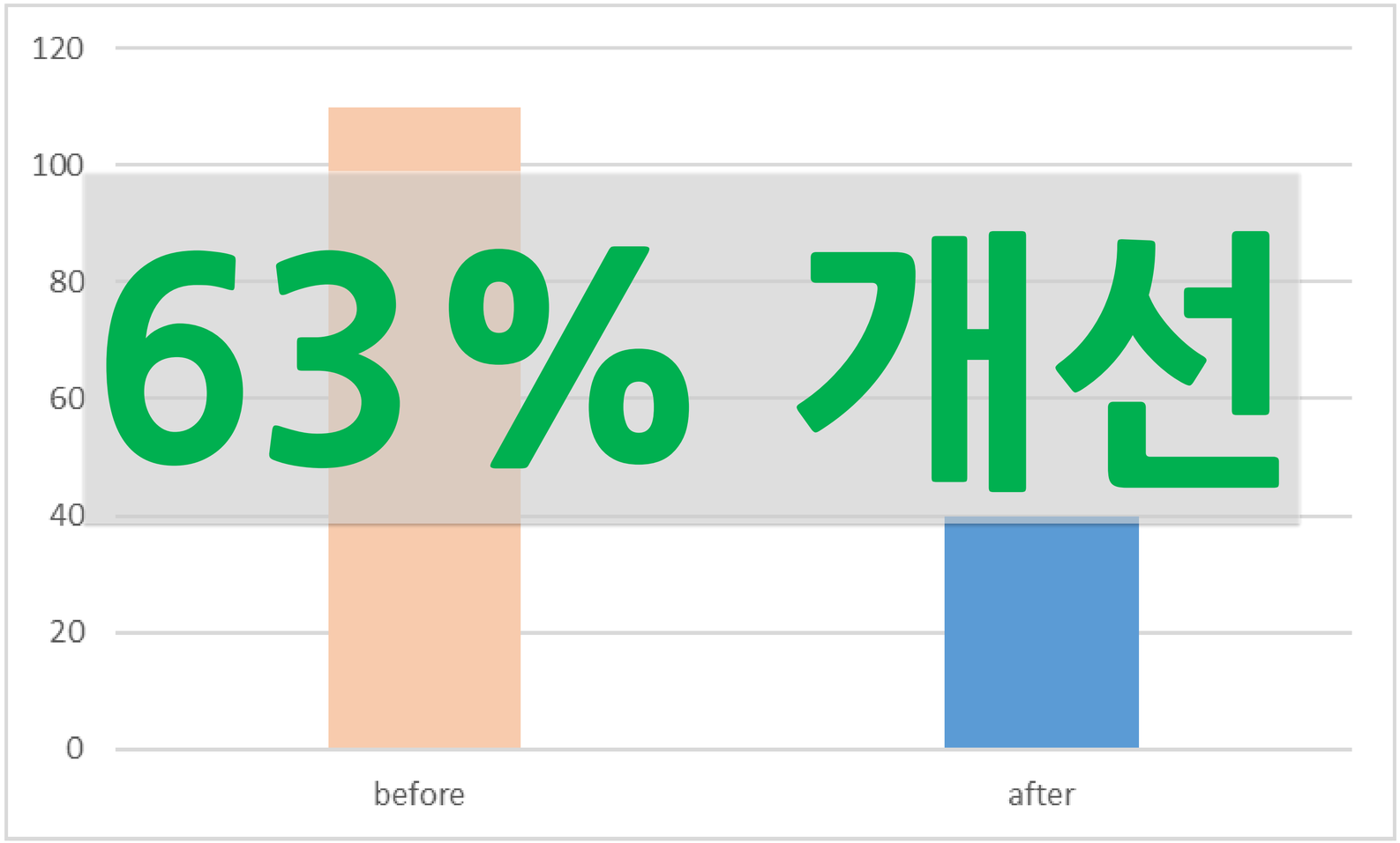


- 변경 : 미리 정해진 규칙으로 저장되는 경우 1bit만으로 기록



2. Trace 개선 결과

DEVIEW 2019



일부 에이전트 데이터 1주일간 DualWrite로 비교시

3. Hbase Parallel Scanner 개발

- salting for rowkey

37. Rowkey Design

37.1. Hotspotting

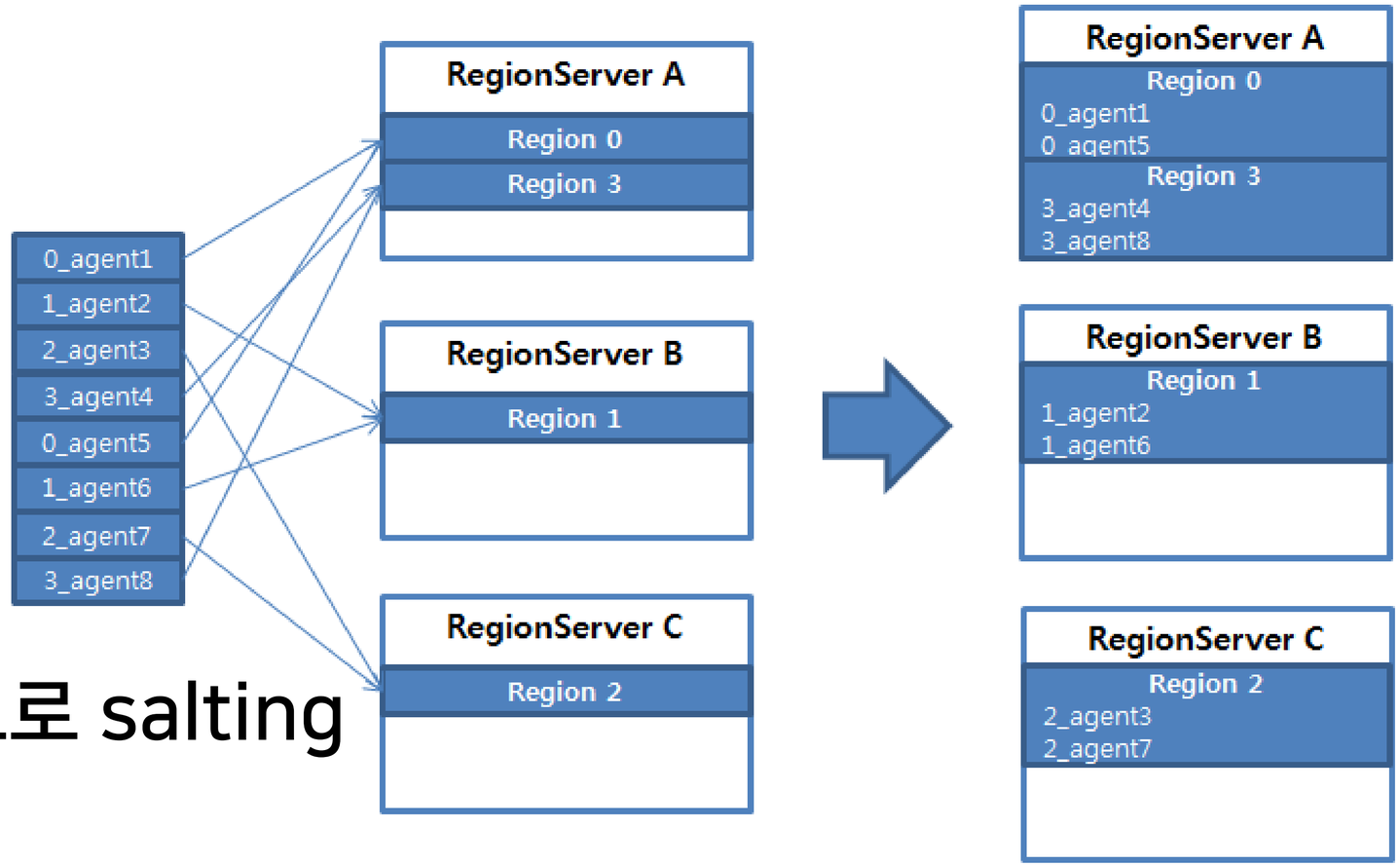
https://hbase.apache.org/book.html#rowkey.design

Rows in HBase are sorted lexicographically by row key. This design optimizes for scans, allowing you to store related rows, or rows that will be read together, near each other. However, poorly designed row keys are a common source of *hotspotting*. Hotspotting occurs when a large amount of client traffic is directed at one node, or only a few nodes, of a cluster. This traffic may represent reads, writes, or other operations. The traffic overwhelms the single machine responsible for hosting that region, causing performance degradation and potentially leading to region unavailability. This can also have adverse effects on other regions hosted by the same region server as that host is unable to service the requested load. It is important to design data access patterns such that the cluster is fully and evenly utilized.

To prevent hotspotting on writes, design your row keys such that rows that truly do need to be in the same region are, but in the bigger picture, data is being written to multiple regions across the cluster, rather than one at a time. Some common techniques for avoiding hotspotting are described below, along with some of their advantages and drawbacks.

Salting

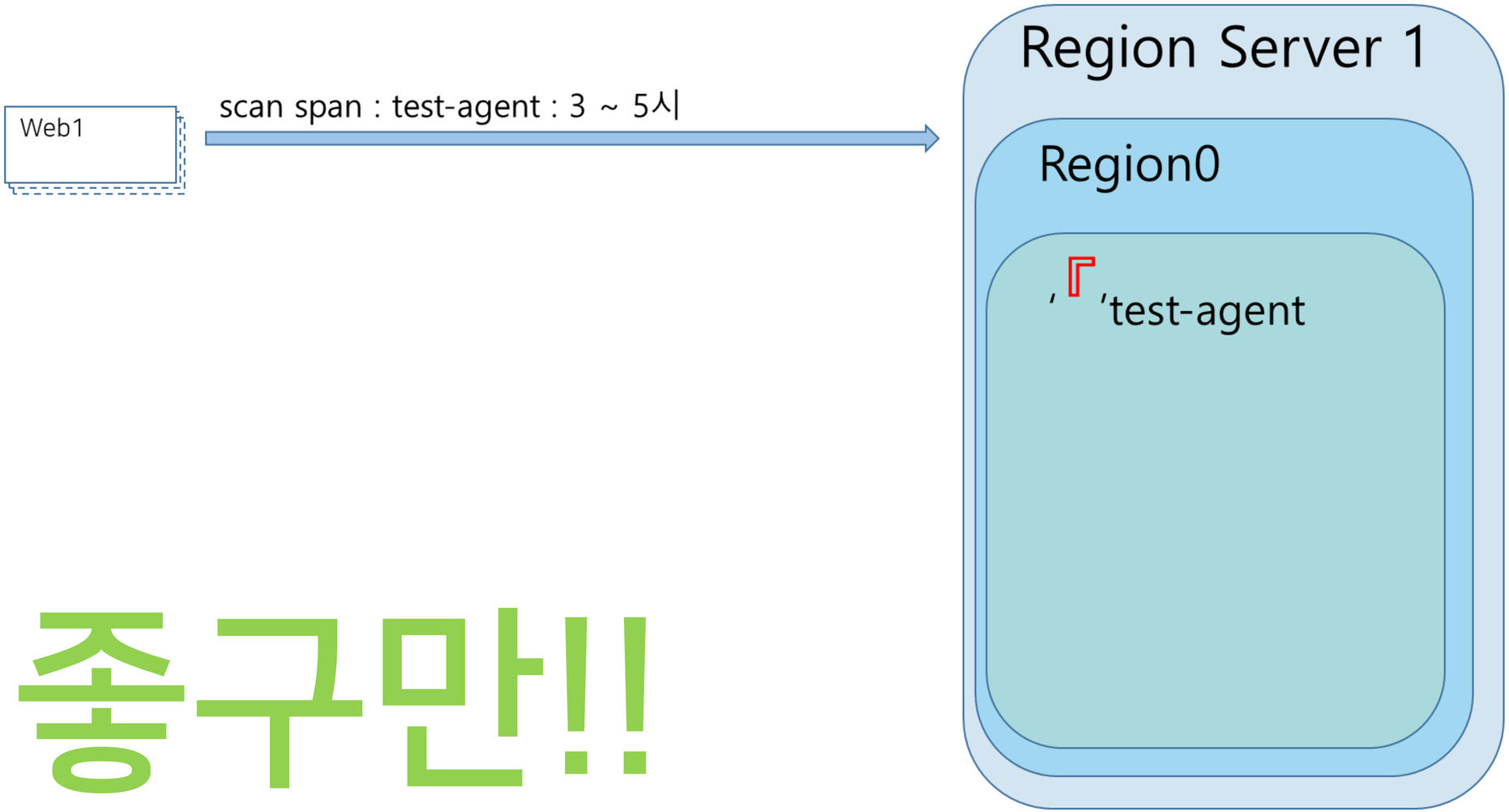
3. Hbase 저장 (기존)



agent 이름으로 salting

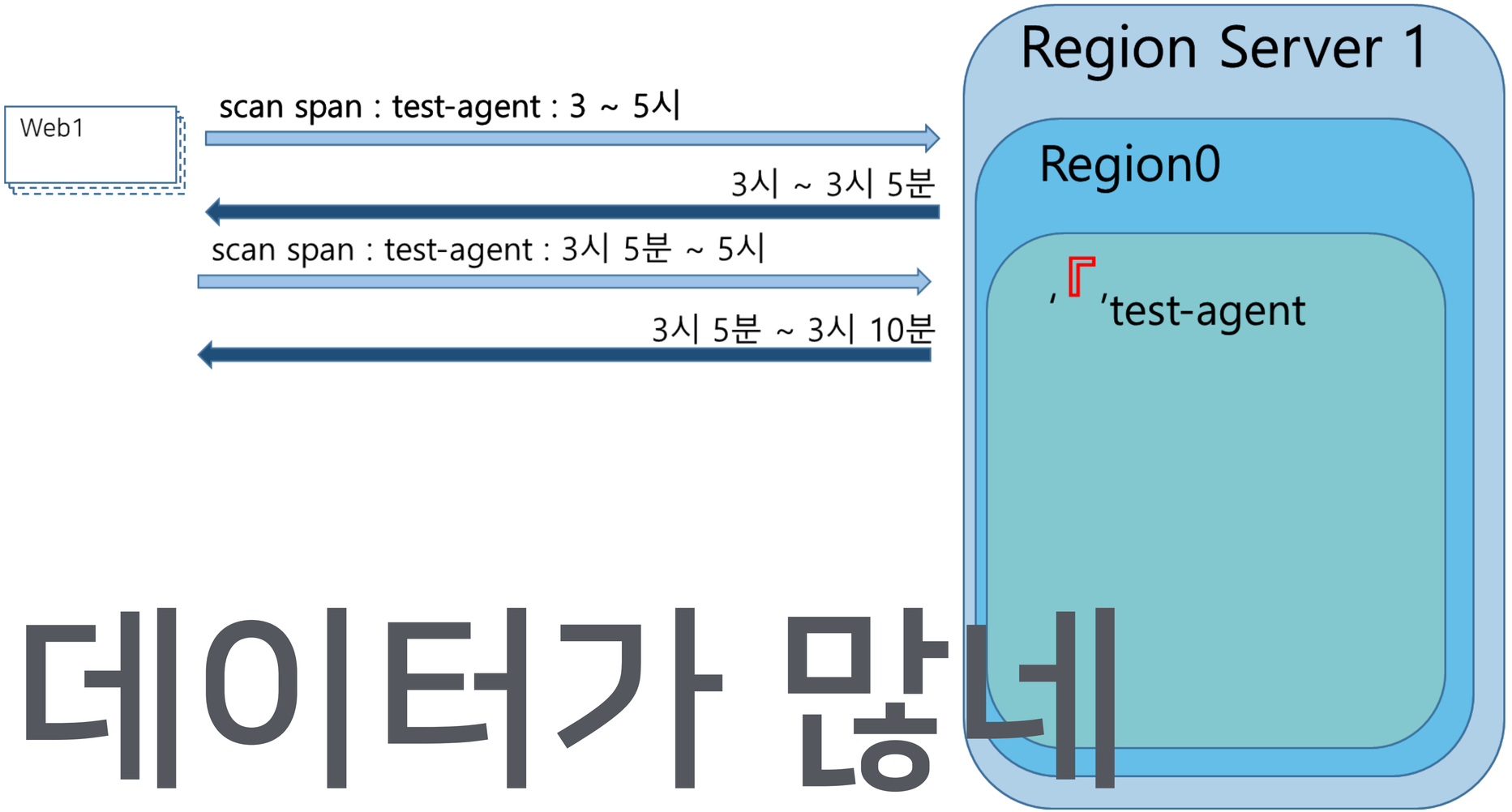
Hotspotting은 피하고, 연관성 있는 데이터는 모으고

3. Trace 데이터 조회 (기존)



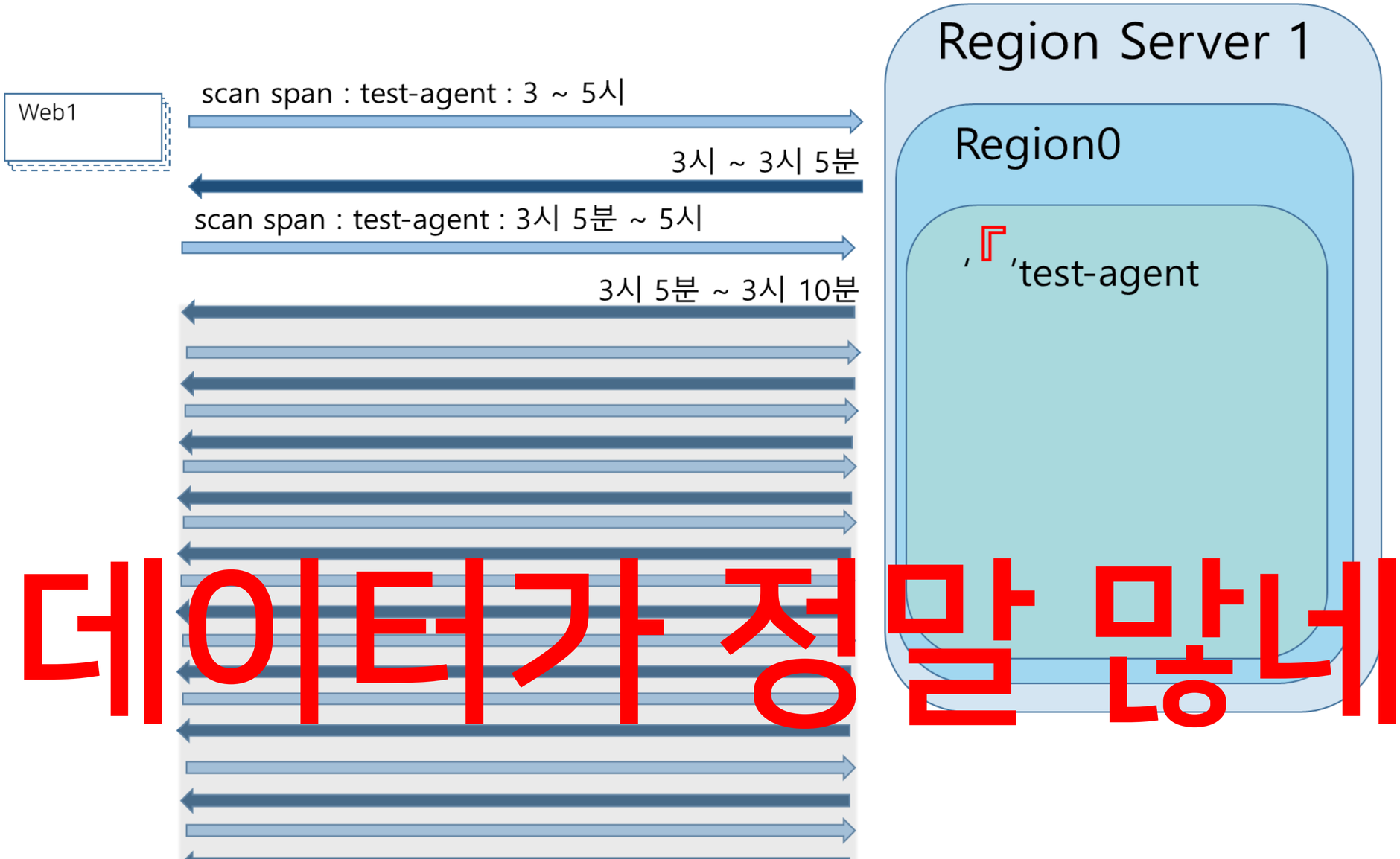
종구만!!

3. Trace 데이터 조회 (기존)



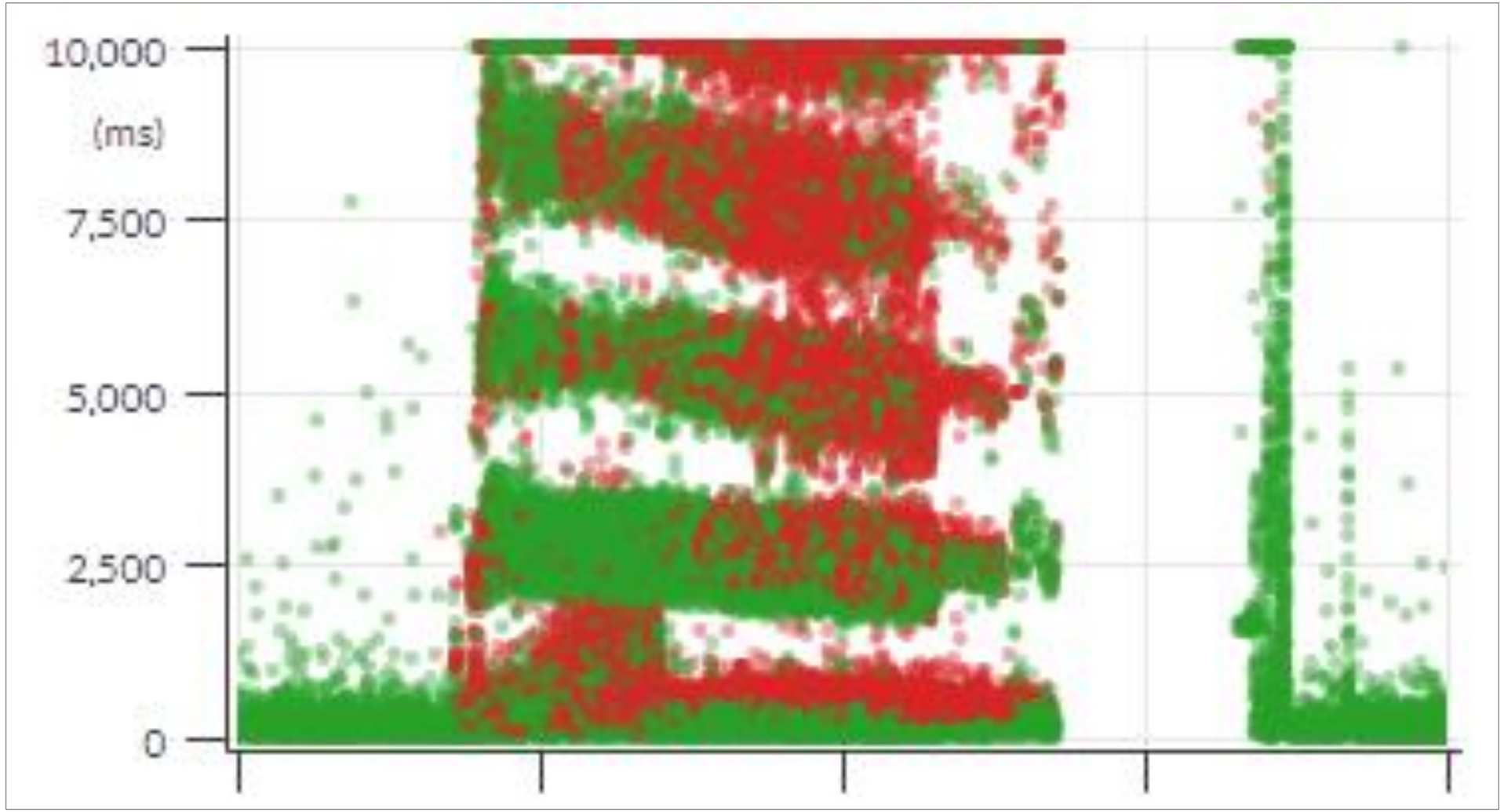
데이터가 많네

3. Trace 데이터 조회 (기존)



3. Trace 데이터는 정말 많았다

DEVIEW
2019



3. Trace 데이터는 정말 많았다

DEVIEW
2019

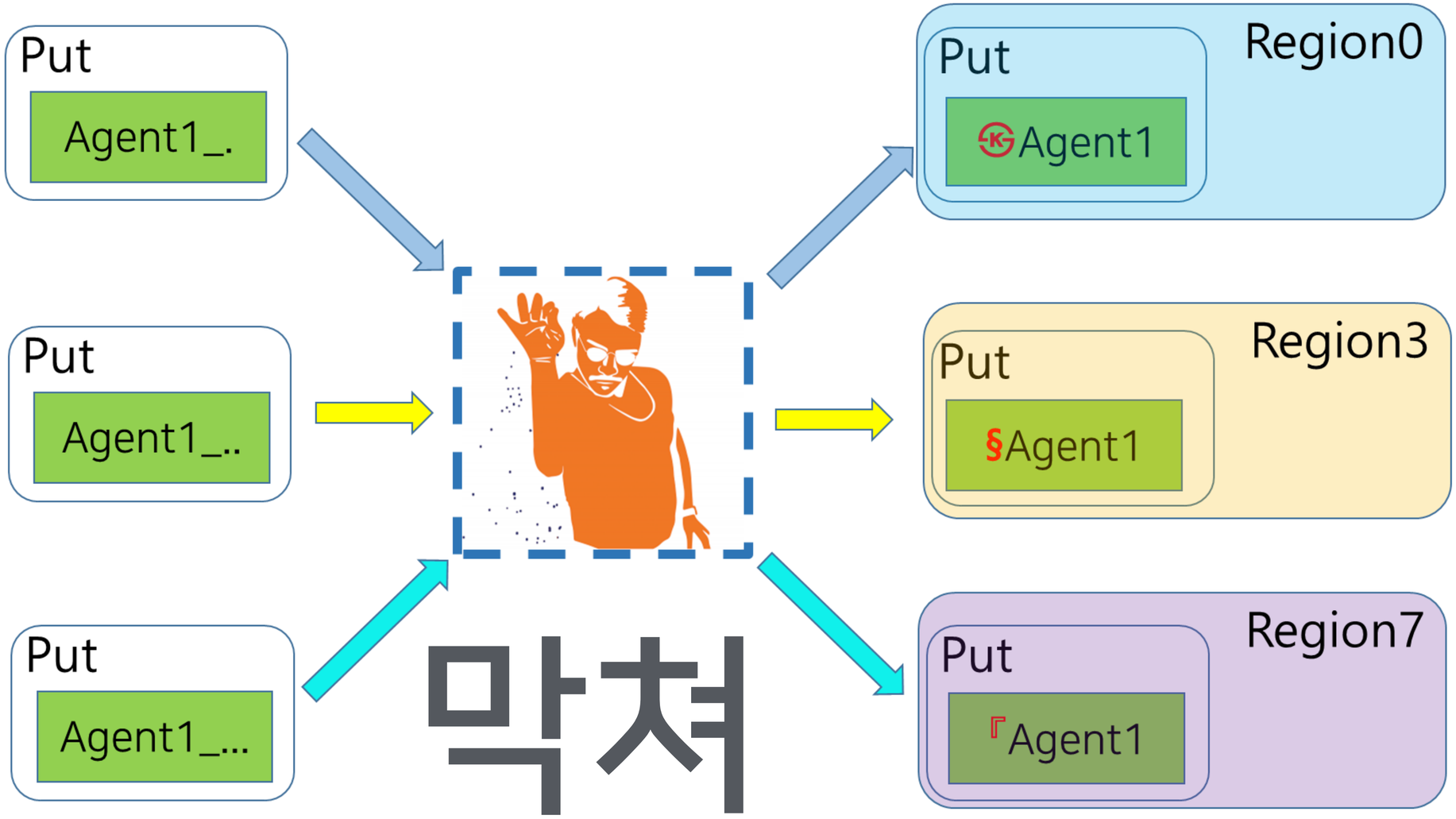
3. 이게 우리한테 맞는건가?

정말 많은 데이터를
1Region에 모으고
순차적으로 조회

VS

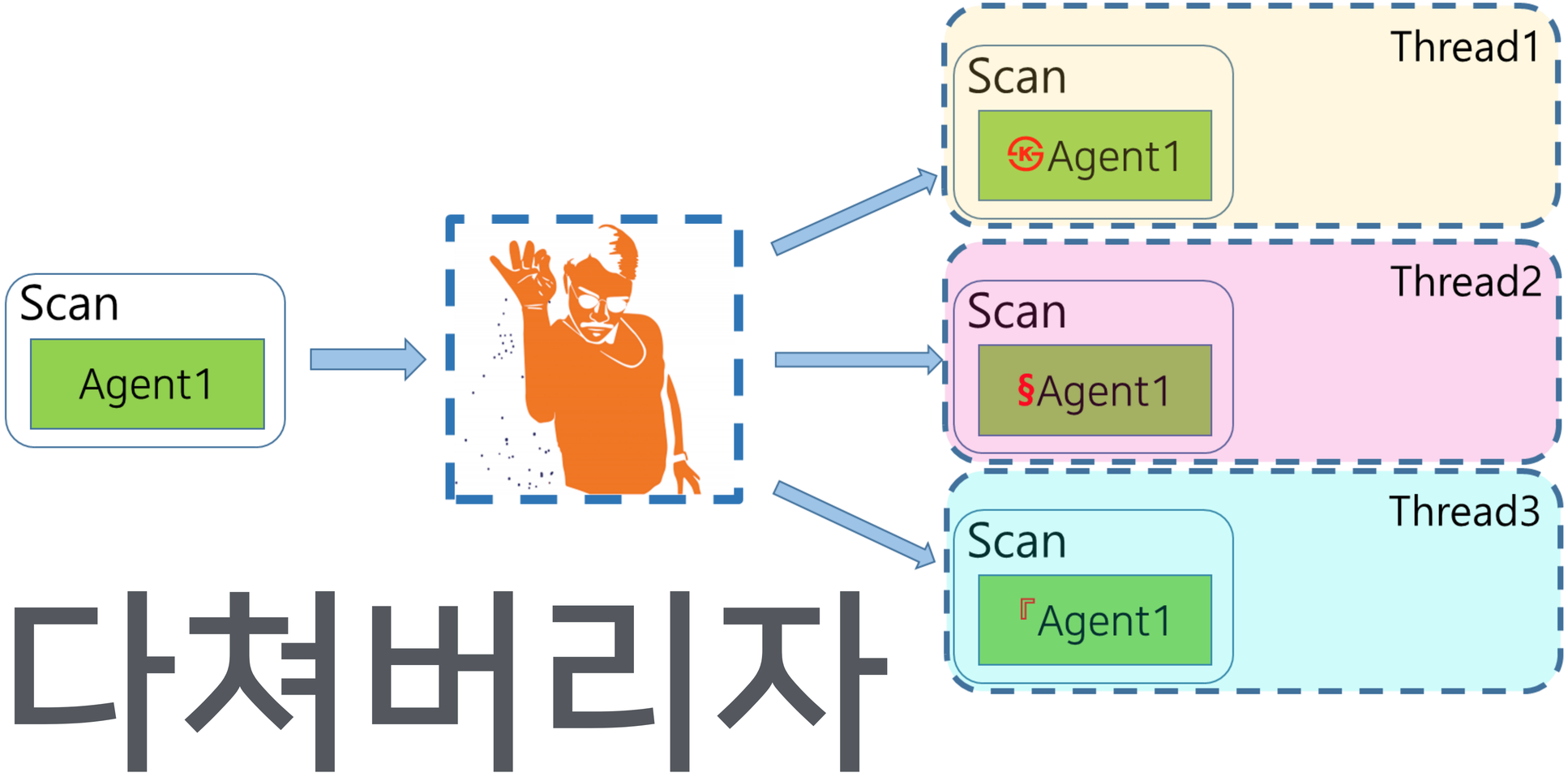
정말 많은 데이터를
여러 Region 넣고
병렬적으로 조회

3. 넣을때 무작위로


Put
Agent1_.
Put
Agent1_..
Put
Agent1_...
막쳐
Region0
Put
Agent1
Region3
Put
§Agent1
Region7
Put
『Agent1

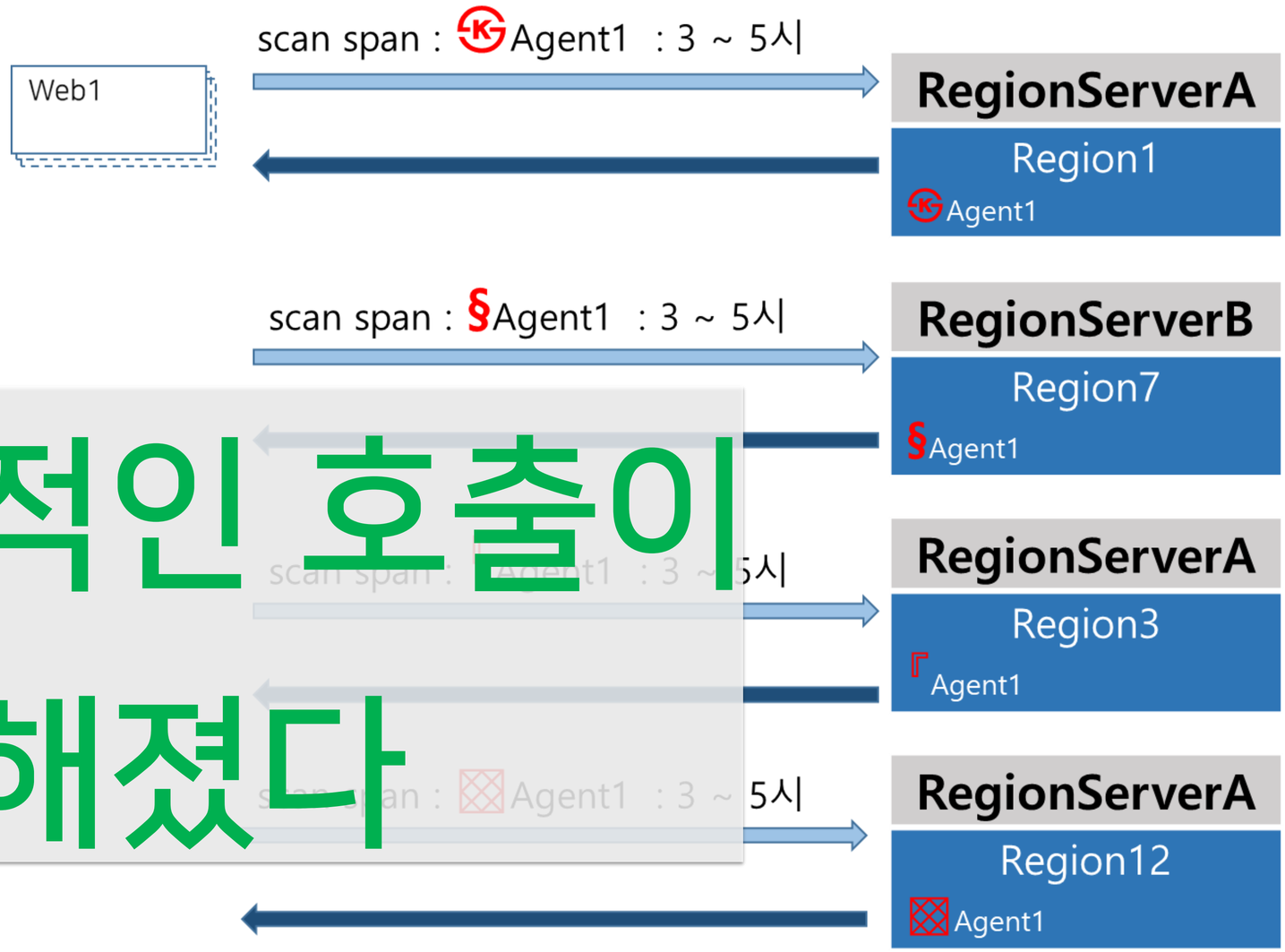
3. 가지고 올때는 다 가져오고

DEVIEW 2019



다쳐버리자

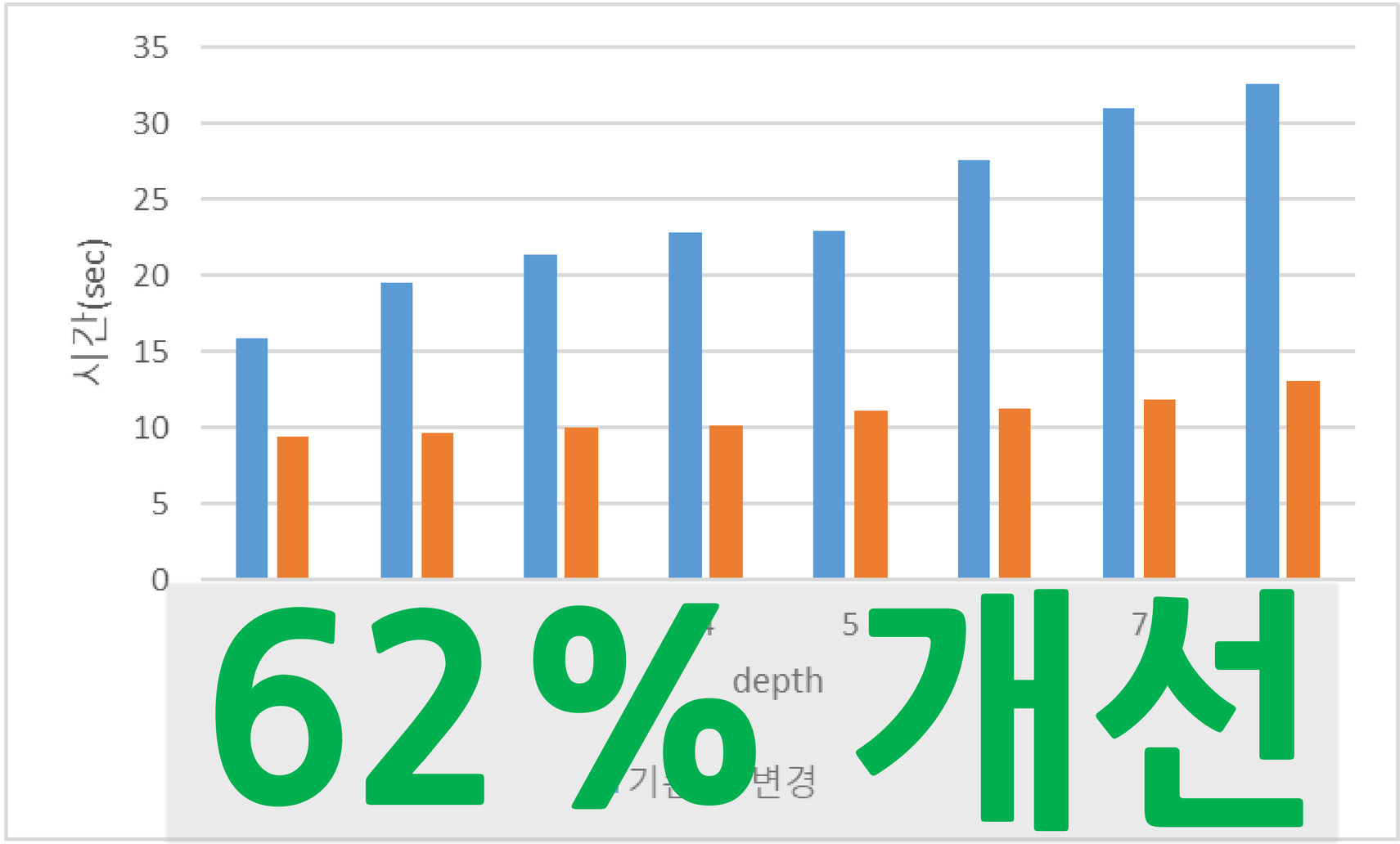
3. 무작위 salting 결과



병렬적인 호출이 가능해졌다

3. 무작위 salting 결과

DEVIEW 2019



3. Hbase Parallel Scanner 회고

- Salting Key를 이 처럼 쓰는건 매우 데이터가 많이 들어오는 경우에만 권장합니다.

- Hbase client 2.0 부터는 유사한 기능을 지원하고 있습니다.

DEVIEW
2019

4. 다양한 컴포넌트에 대한 observability 확보하기

해결해야 하는 문제

- 많고 다양한 플러그인을 지원하려면

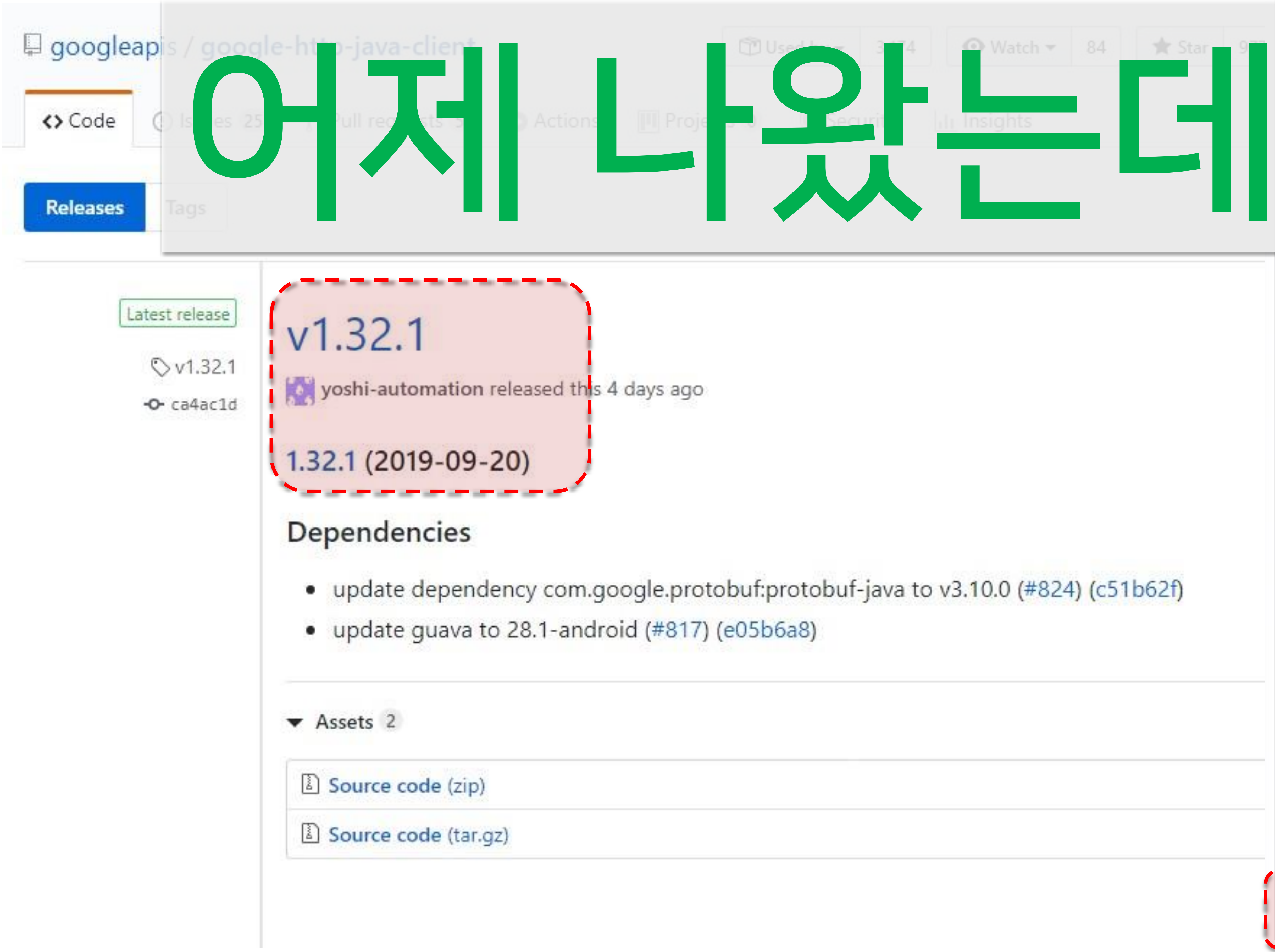
- 지속적으로 릴리즈 되는 라이브러리의 플러그인을 지원하려면

해결

DEVIEW
2019

doc/modules.md 확인가능

어제 나왔는데 오늘 지원함!



Title	Instrumented Library	Min	Max	Comment
Tomcat		6.x	9.x	
Jetty		8.x	9.x	
JBoss		6.x	7.x	
Resin		4.x	4.x	
Websphere		6.x	8.x	
Vertx		3.3	3.5	
Weblogic		[illegible]	[illegible]	
Undert[illegible]				
Jasper				
OpenWhisk	whisk.core			
SpringMVC Framework	spring-webmvc	3.0.7	5.1.9	
Spring Web	spring-web	4.1.2	4.3.25	
Spring RabbitMQ	spring-rabbit	1.3.3	2.1.10	
Spring IBatis	spring-ibatis	2.0.7	2.0.8	
Spring MyBatis	mybatis-spring	1.1.0	1.3.3	
Spring Boot	spring-boot-autoconfigure			
MyBatis	mybatis	3.0.3	3.3.1	
Hystrix	hystrix-core	1.4.0	1.5.18	
JDKHTTP				
Httpclient3	commons-httpclient	3.0	3.1	
Httpclient4	httpclient	4.0	4.5.4	
Thrift	libthrift	0.9.1	0.12.0	
Google HTTP Client	google-http-client	1.19.0	1.32.1	

必

플러그인 테스트

1. 플러그인 어떻게 테스트 할까?

플러그인 테스트는

최대한 실상황에서

⇒그럴려면 에이전트를

활성화 해야 하는데

⇒그럴려면 프로세스를

실행 해야 하는데

1. Custom TestRunner 생성


```
public class PinpointTestRunner extends BlockJUnit4ClassRunner {

    public PinpointTestRunner(Class<?> klass) throws InitializationError {
        super(klass);
    }


    @Override
    public void run(RunNotifier notifier) {
        // 여기다 에이전트를 활성화 하는 프로세스를 실행해서 결과값을 받으면;;;;
    }

}
```

1. 테스트에서 외부 프로세스를 실행

- ProcessBuilder

```
ProcessBuilder builder = new ProcessBuilder();
// 실행할 커맨드를 넣고
// builder.command("rm", "-rf", "/");
builder.command(getCommands());  <= "java ${runClass} ${args} "
// 커맨드를 실행
Process process = builder.start();
// 실행된 커맨드의 결과값을 획득
InputStream inputStream =
process.getInputStream();
```

1. 자바 프로세스는 main부터

- ${runClass} // main에서 테스트를 실행하는 클래스 생성

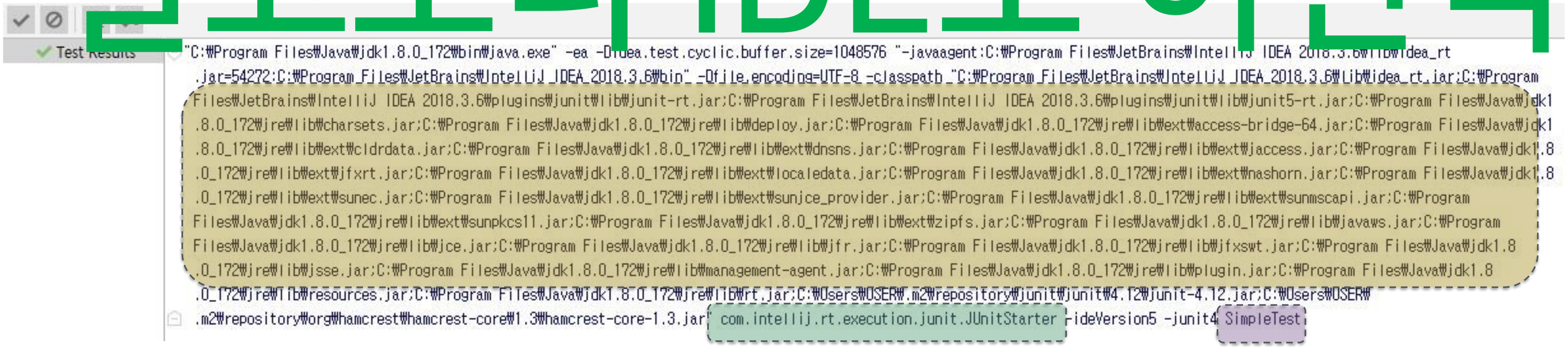
```
public static void main(String[] args) throws ClassNotFoundException, InitializationError {
    // 인자로 Test할 ClassName을 받아
    String testClazzName = args[1];

    // 클래스 생성후
    ClassLoader classLoader = RunTestMain.class.getClassLoader();
    Class<?> testClazz = classLoader.loadClass(testClazzName);

    Runner runner = new BlockJUnit4ClassRunner(testClazz);
    // junit 실행
    JUnitCore junit = new JUnitCore();
    junit.run(runner);
}
```

1. 테스트에서 외부 프로세스를 실행

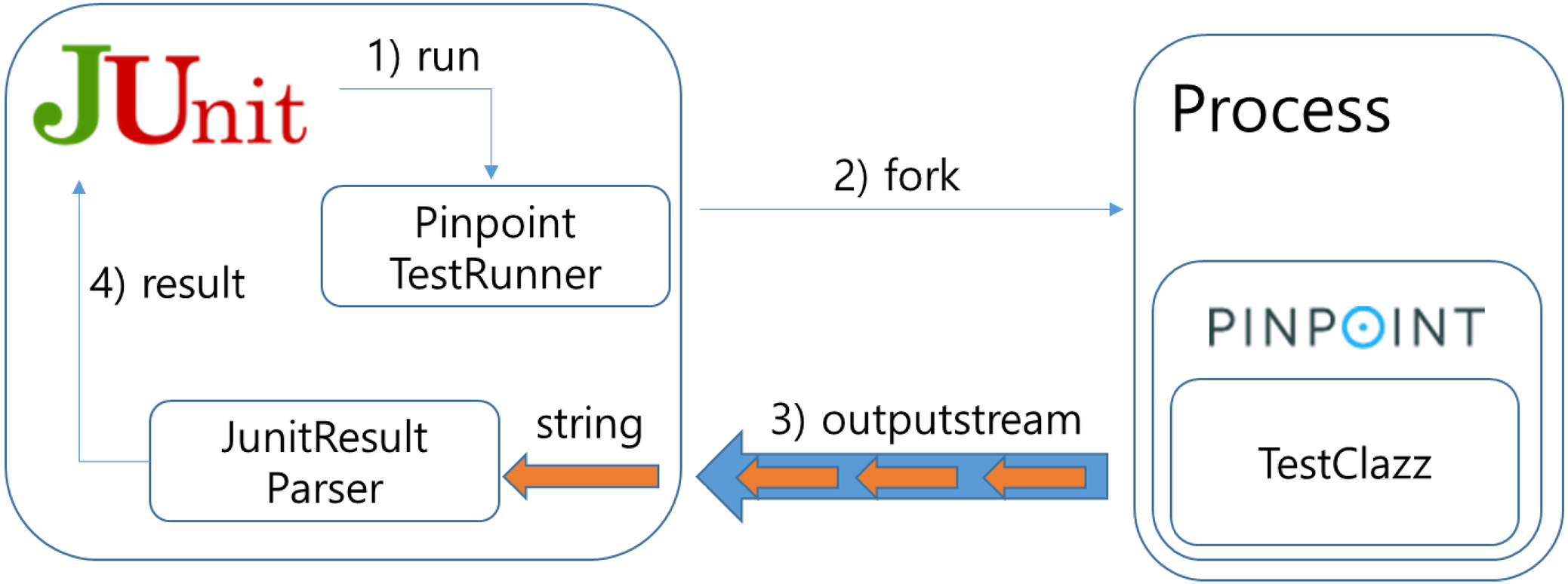
DEVIEW 2019

```
java -cp ${classpath}
        ${PINPOINT_OPTS}
        RunMainTest test.JacksonPluginIT
```

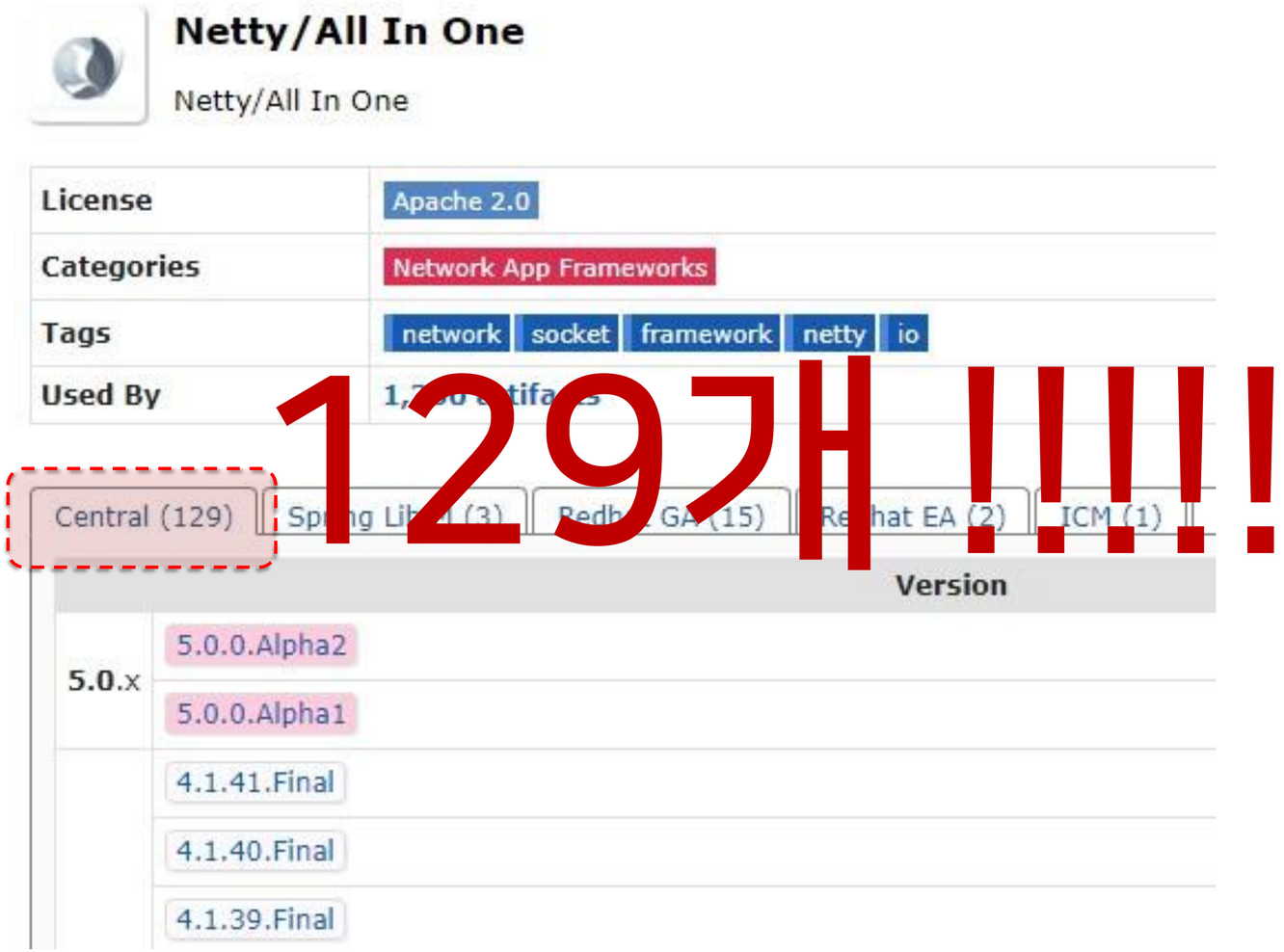
알고보니 IDE도 이런식



1. 플러그인 테스트 구조


JUnit
1) run
Pinpoint
TestRunner
2) fork
Process
4) result
PINPOINT
JunitResult
Parser
string
3) outputstream
TestClazz

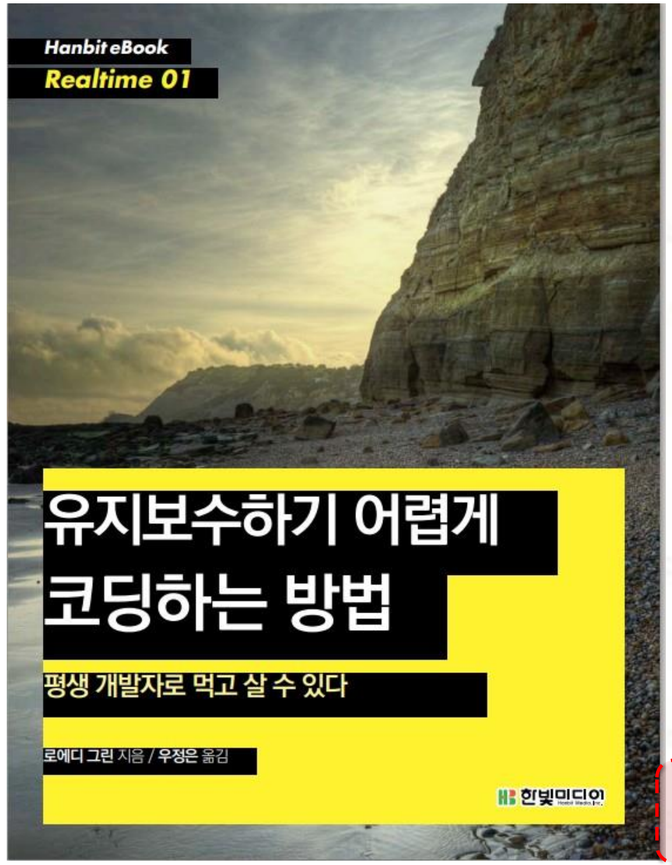
2. 다양한 버전 테스트



129개 !!!!!

2. 겁쟁이가 되기 싫어졌다

- netty 버전 테스트를 하려면 pom.xml을 129번 바꾸면 되긴하는데.

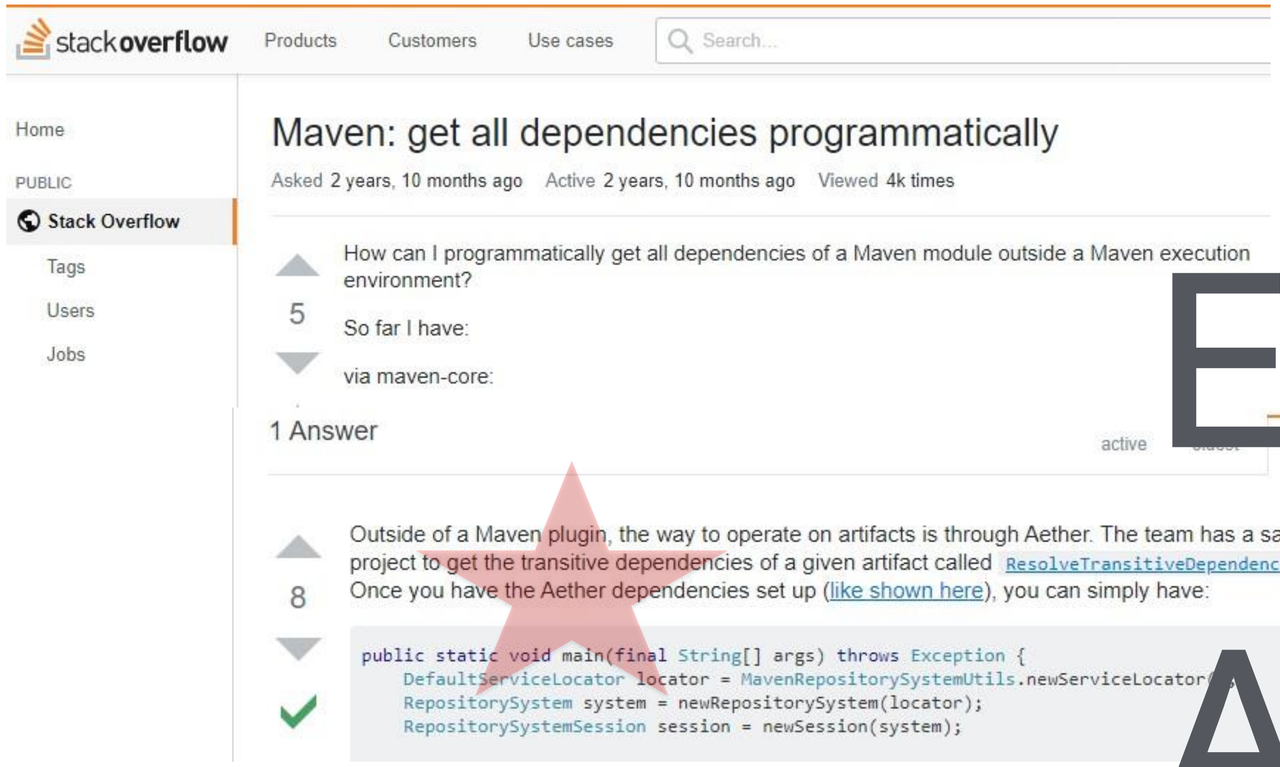


겁쟁이 전용 테스트

용감한 코더는 이 단계를 건너뛴다. 상사를 무서워하고, 직장을 잃을까
고, 고객의 불평 메일을 받거나 고소당하는 일을 걱정하는 프로그래머
다. 이러한 두려움 때문에 행동에 제약을 받고, 생산성이 떨어진다는 것
아는 사실이다. 연구에 따르면 테스트 단계를 없애는 것이 관리자로 하여
을 미리 결정할 수 있게 하는 등 계획 단계부터 많은 도움이 된다. 두려움
다면 혁신과 실험정신이 창궐할 수 있다. 프로그래머는 코드를 생산하는
수 있고, 헬프 데스크와 기존 유지보수 그룹이 협력해서 디버깅을 담당할
우리의 코딩 능력을 온전히 믿는다면 테스트 따위는 더 이상 불필요하다.

2. Aether

DEVIEW 2019



Eclipse Aether

2. Version Scheme

- 특정 버전 명시 : [1.3.2] => 1.3.2
- 특정 버전 여러 개 명시 : [1.3.2], [1.3.5] => 1.3.2, 1.3.5
- 버전 범위 : [1.3.2,1.3.10] => 1.3.2 ~ 1.3.10
- 버전 범위 : [1.3.2,1.3.10) => 1.3.2 ~ 1.3.9
- 버전 범위 : (1.3.2,1.3.10] => 1.3.3 ~ 1.3.10
- 버전 범위 : (1.3.2,1.3.10) => 1.3.3 ~ 1.3.9

2. Version Scheme

- 버전 범위 : [1.3.2,1.3.max] => 1.3.2부터 1.3.x 최신 버전 까지
  1.3.2 ~ 1.3.33
- 버전 범위 : [1.3.min,1.3.max] => 1.3.0 ~ 1.3.33 (1.3.가장최신)
- 버전 범위 : [1.3.min,) => 1.3.0 부터 최신 버전 까지
  1.3.0 ~ 2.0.27

2. artifact로 버전 가져오기


```
@Test
public void getPinpointVersionTest() throws VersionRangeResolutionException {
    // 메이븐 아티팩트 생성
    DefaultArtifact artifact = new DefaultArtifact("com.navercorp.pinpoint", "pinpoint", "jar", "[1.min,)");

    // Request 파라미터로 아티팩트와 메이븐 리파지토리 포함
    VersionRangeRequest rangeRequest = new VersionRangeRequest();
    rangeRequest.setArtifact(artifact);
    rangeRequest.setRepositories(getLocalRepository());

    // 호출
    VersionRangeResult rangeResult = getSystem().resolveVersionRange(getSession(), rangeRequest);
    List<Version> versionList = rangeResult.getVersions();

    System.out.println(versionList);
}
```

```
[1.6.0, 1.6.1, 1.6.2, 1.7.0, 1.7.1, 1.7.2, 1.7.3-p1, 1.8.0-RC1, 1.8.0-SNAPSHOT, 1.8.0, 1.8.1-RC1,

 1.8.1-SNAPSHOT, 1.8.1, 1.8.2-SNAPSHOT, 1.8.3-SNAPSHOT, 1.8.3, 1.8.4-SNAPSHOT, 1.8.5-SNAPSHOT, 1.9.0-SNAPSHOT]
```

2. 의존 라이브러리 가져오기

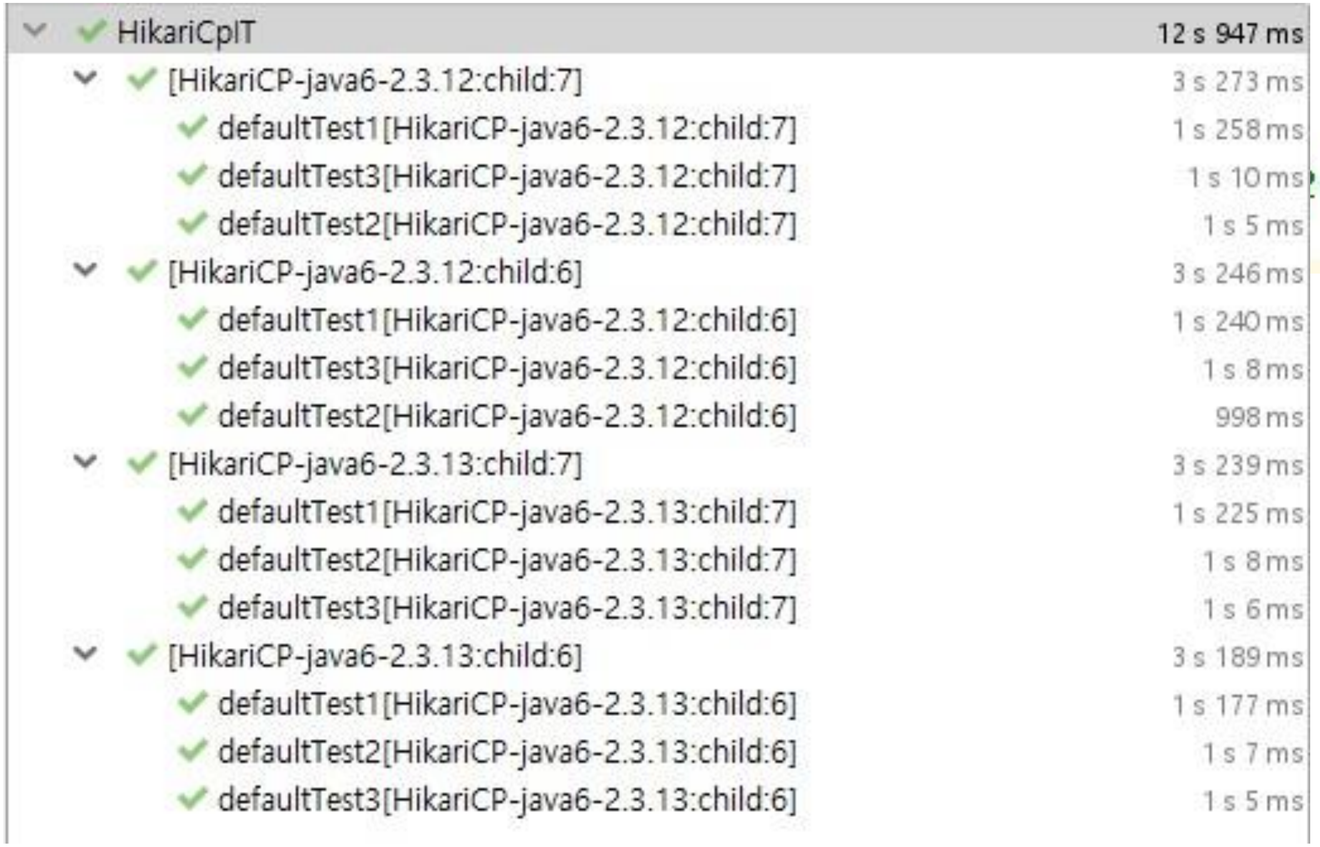

```
@Test
public void getPinpointDependencyTest() throws Exception{
    // 메이븐 아티팩트 생성
    DefaultArtifact artifact = new DefaultArtifact("com.navercorp.pinpoint", "pinpoint-rpc", "jar", "1.8.3");

    // 디펜던시 요청 메이븐 RUNTIME 범위만 가져오게함 ,
    Dependency dependency = new Dependency(artifact, JavaScopes.RUNTIME);
    DependencyFilter classpathFilter = DependencyFilterUtils.classpathFilter(JavaScopes.RUNTIME);

    // 파라미터로 디펜던시, 리파지토리
    CollectRequest collectRequest = new CollectRequest((Dependency)null, Arrays.asList(dependency), getLocalRepository());
    DependencyRequest dependencyRequest = new DependencyRequest(collectRequest, classpathFilter);

    // 호출 (로컬에 라이브러리가 없는 경우 로컬 리파지토리에 저장) & 결과값 출력
    DependencyResult result = getSystem().resolveDependencies(getSession(), dependencyRequest);
    System.out.println(getFiles(result));
}
```

```
[C:₩Users₩USER₩.m2₩repository₩com₩navercorp₩pinpoint₩pinpoint-rpc₩1.8.3₩pinpoint-rpc-1.8.3.jar,
C:₩Users₩USER₩.m2₩repository₩com₩navercorp₩pinpoint₩pinpoint-annotations₩1.8.3₩pinpoint-annotations-1.8.3.jar,
C:₩Users₩USER₩.m2₩repository₩com₩navercorp₩pinpoint₩pinpoint-commons₩1.8.3₩pinpoint-commons-1.8.3.jar,
C:₩Users₩USER₩.m2₩repository₩io₩netty₩netty₩3.10.6.Final₩netty-3.10.6.Final.jar]
```

2. 어노테이션으로 기능 구현

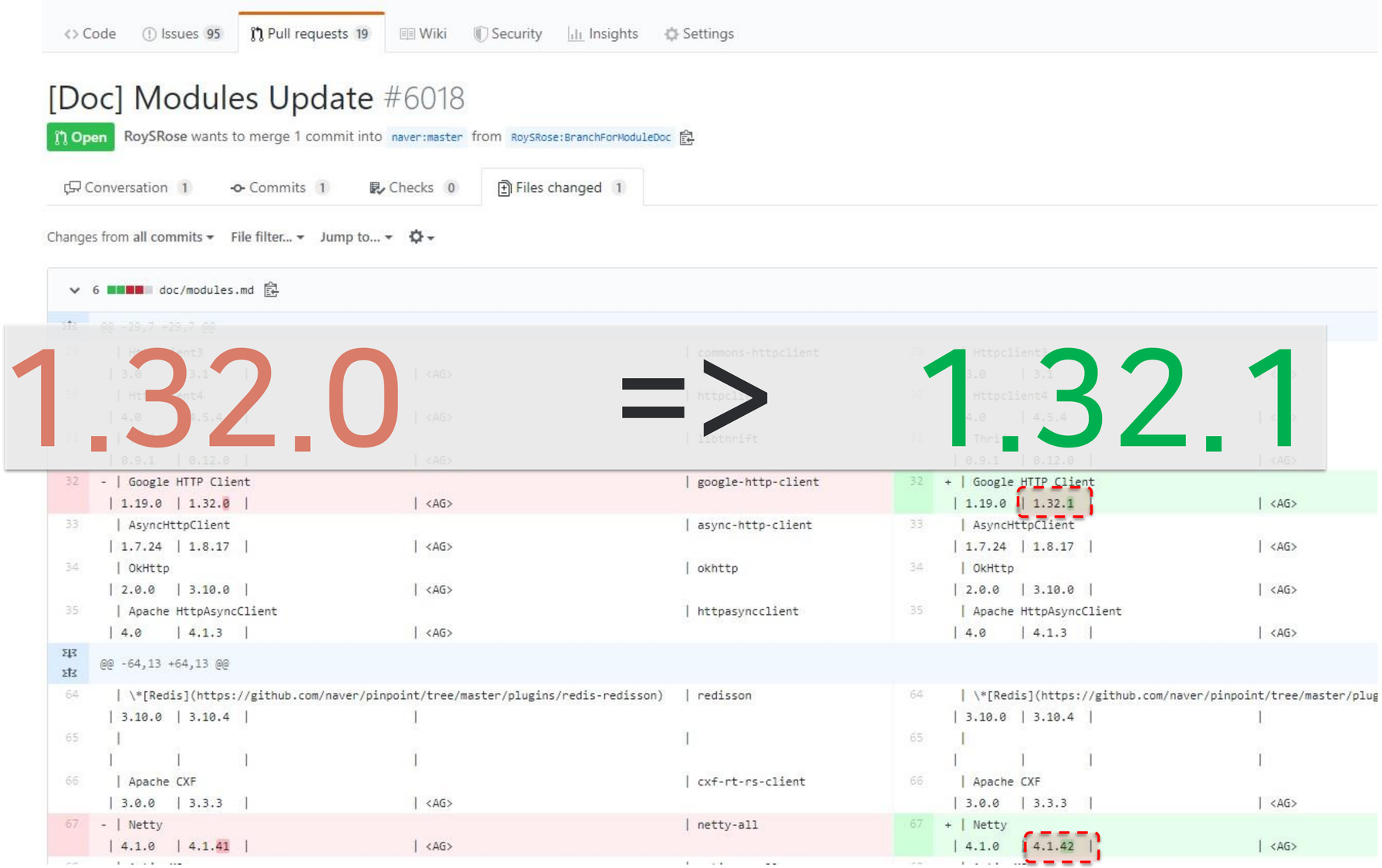

```
@RunWith(PinpointPluginTestSuite.class)
@PinpointAgent(AgentPath.PATH)
@JvmVersion({6, 7})
@Dependency({"com.zaxxer:HikariCP-java6:(2.3.11,]", "com.h2database:h2:1.4.191"})
public class HikariCpIT {
```

2. 다양한 버전 테스트 결과

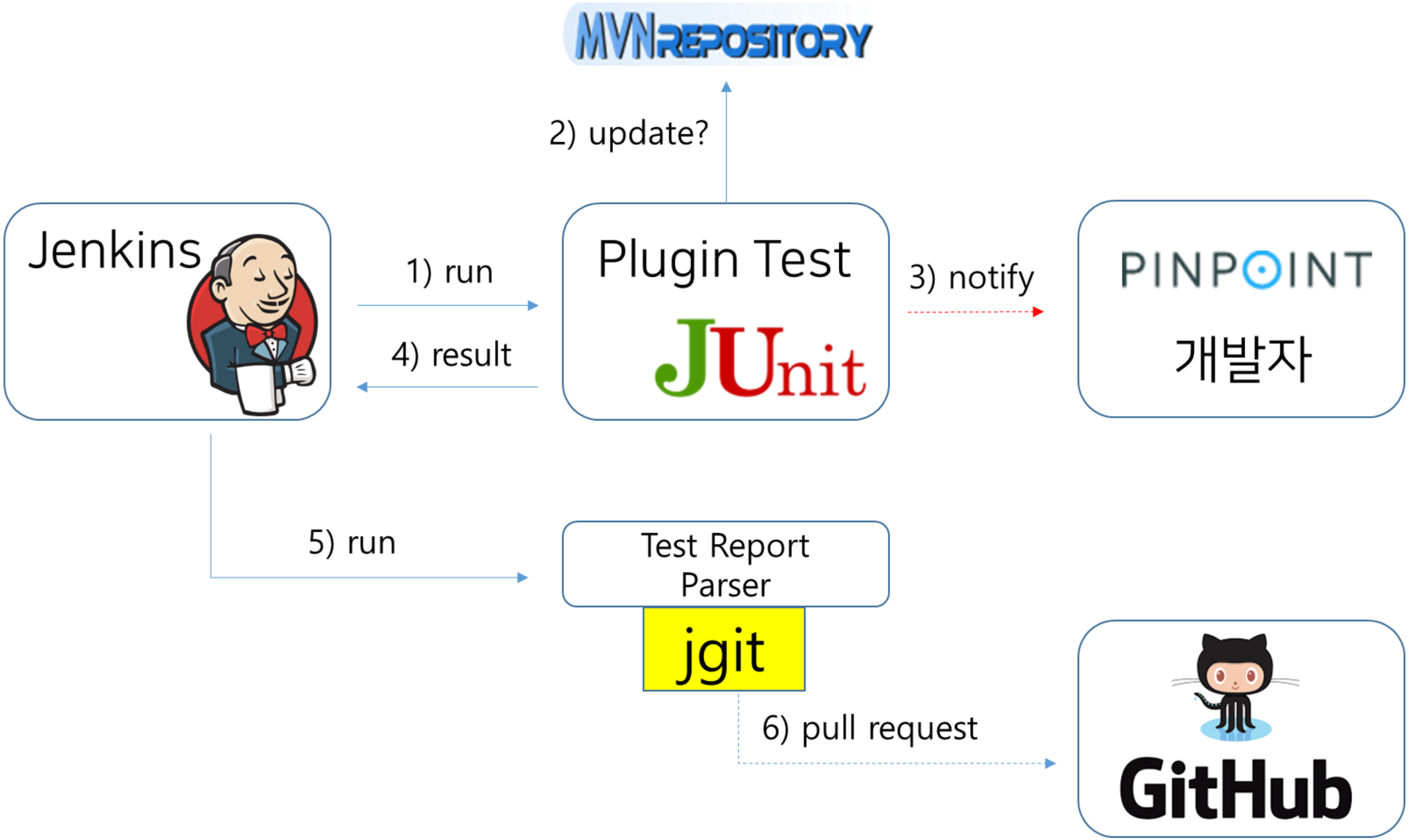


잘된다.!

3. github으로 pull request



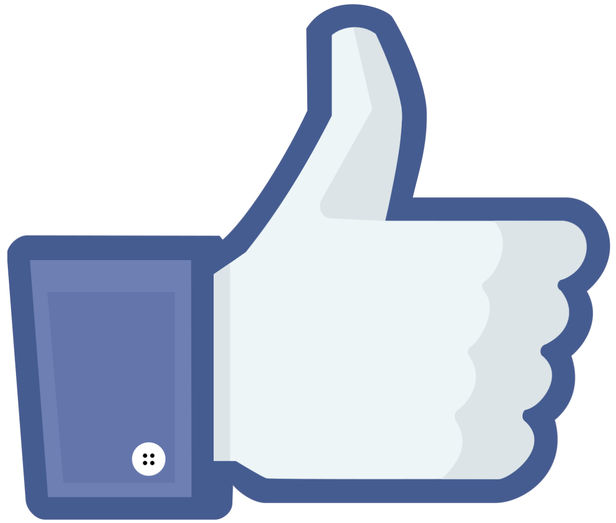
3. 지원 라이브러리 업데이트 구조



3. 테스트의 자가증식

DEVIEW 2019

- 코드는 그대론데 테스트가 계속 늘어난다.



테스트 작성 252개

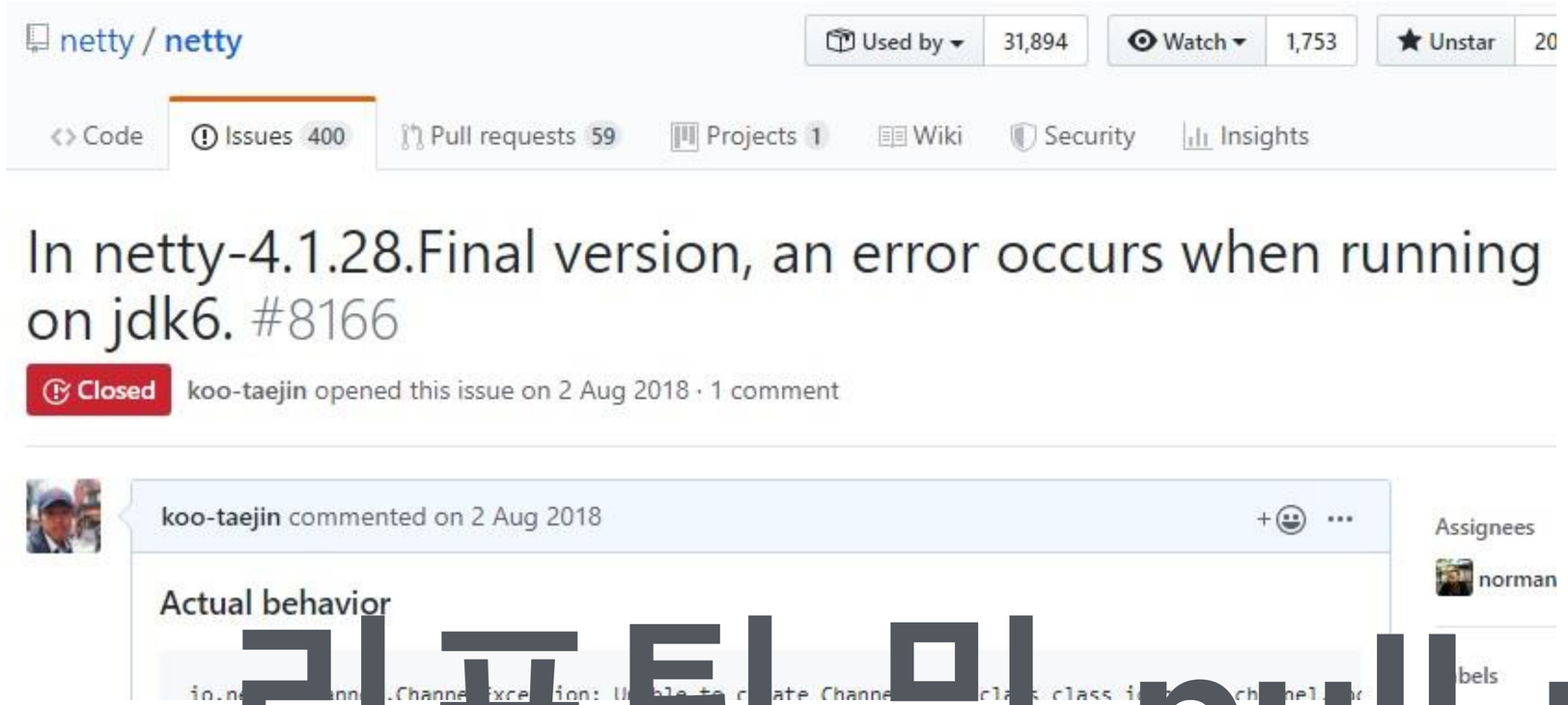
-> 처음 1300개

-> 지금 4512개

3. 핀포인트 테스트를 넘어서

DEVIEW 2019

- netty 플러그인 테스트 에러가 났는데 아무리 봐도 이상함



리포팅 및 pull request

3. 핀포인트 테스트를 넘어서

DEVIEW 2019

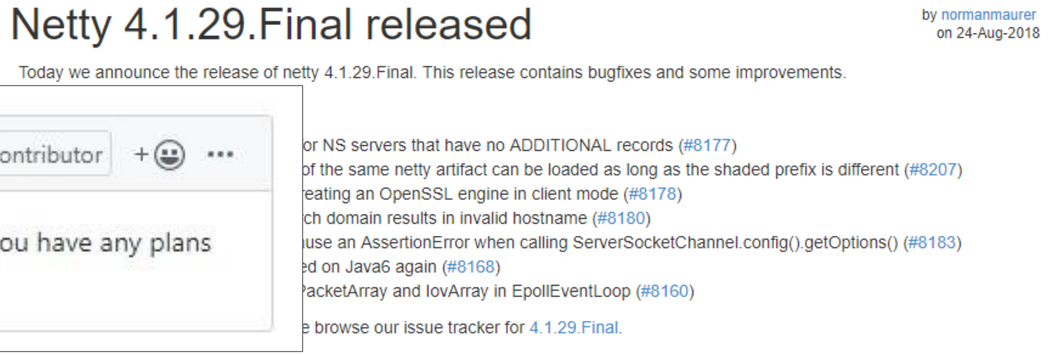
- netty 플러그인 테스트 에러가 났는데 아무리 봐도 이상함

버그는 맞는데

직접 고친다고함

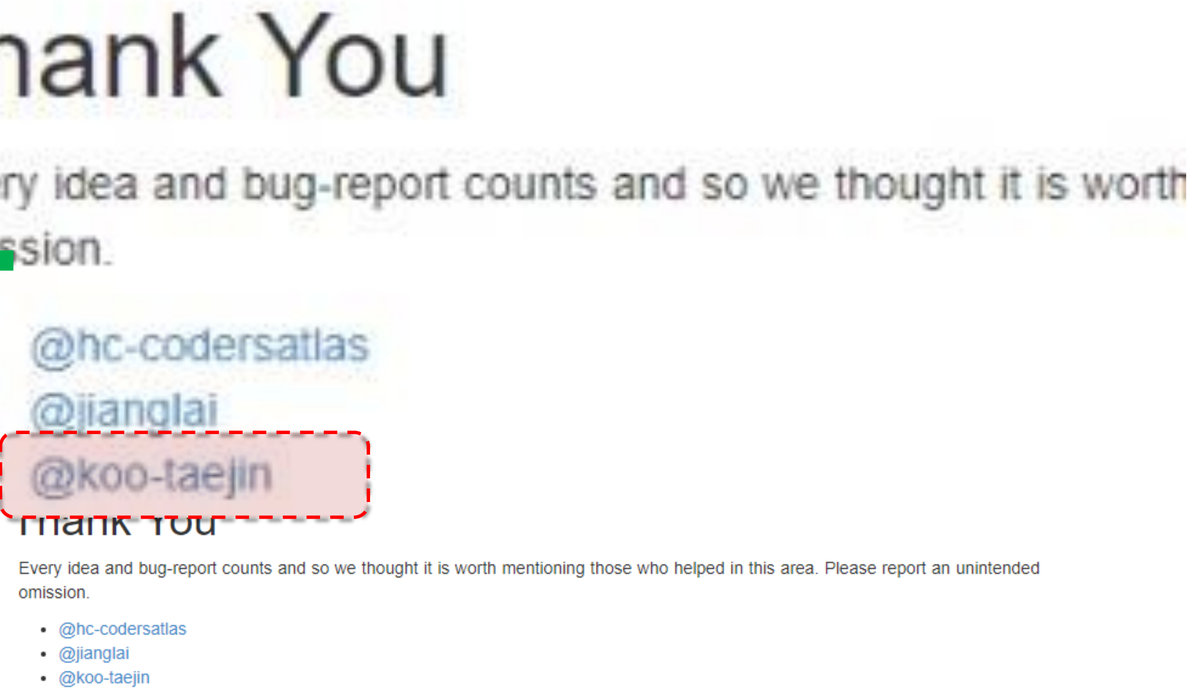
3. 핀포인트 테스트를 넘어서

DEVIEW 2019



intellij 개발자도 빨리 고쳐주세요라고

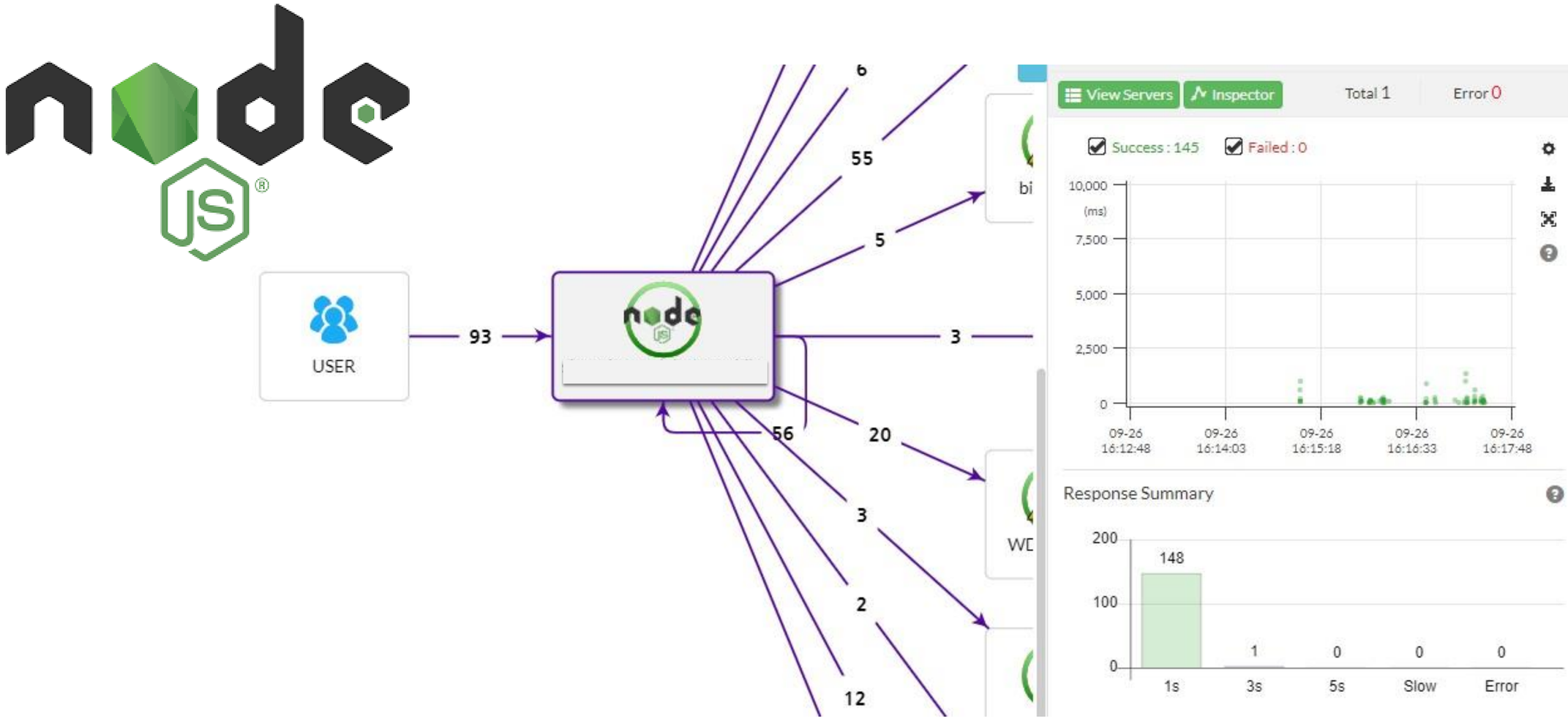
기여 & 뿌듯



DEVIEW
2019

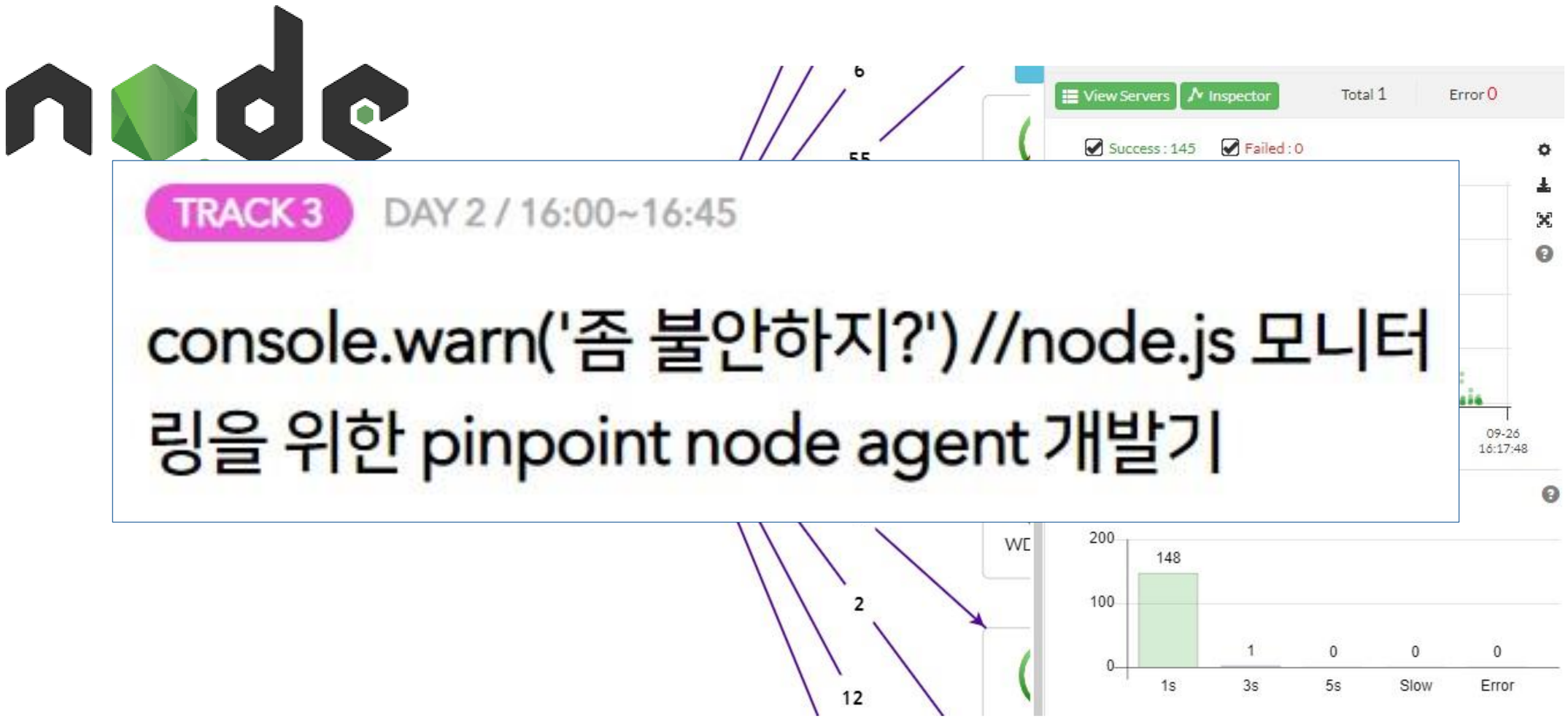
5. What’s next

Multi language support



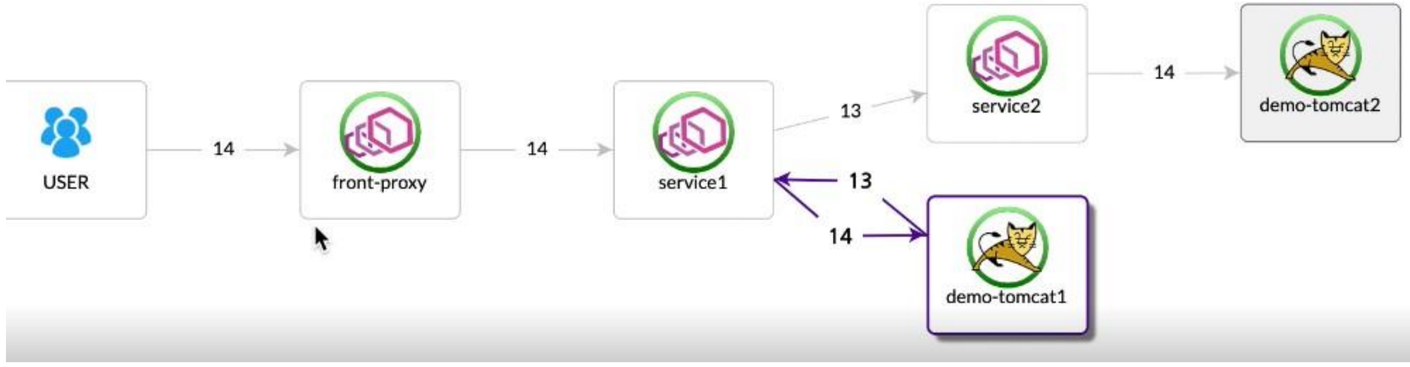
Multi language support

DEVIEW 2019



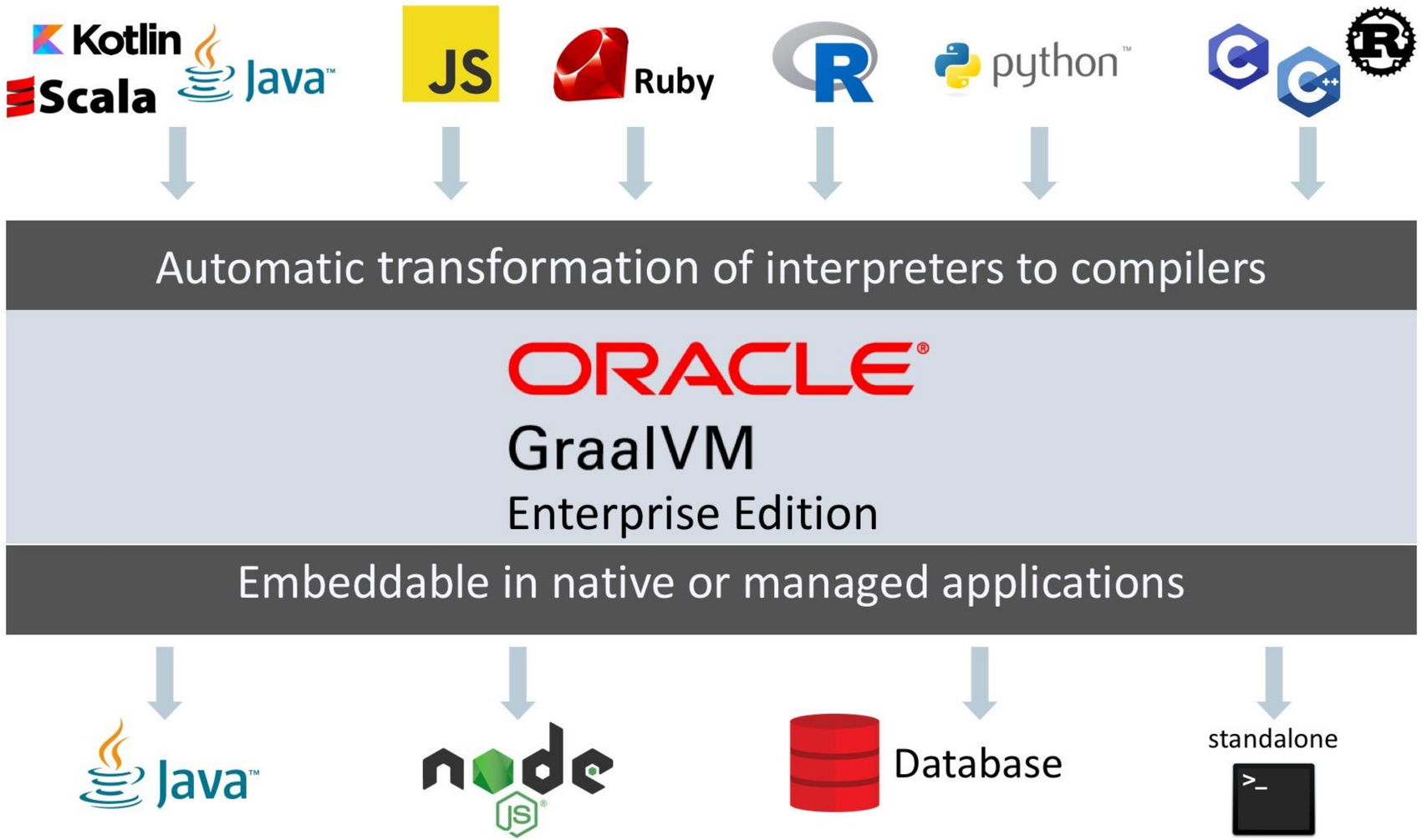
Multi language support (gRpc)

Multi language support (gRpc)



Multi language support (GraalVM)

Multi language support (GraalVM)

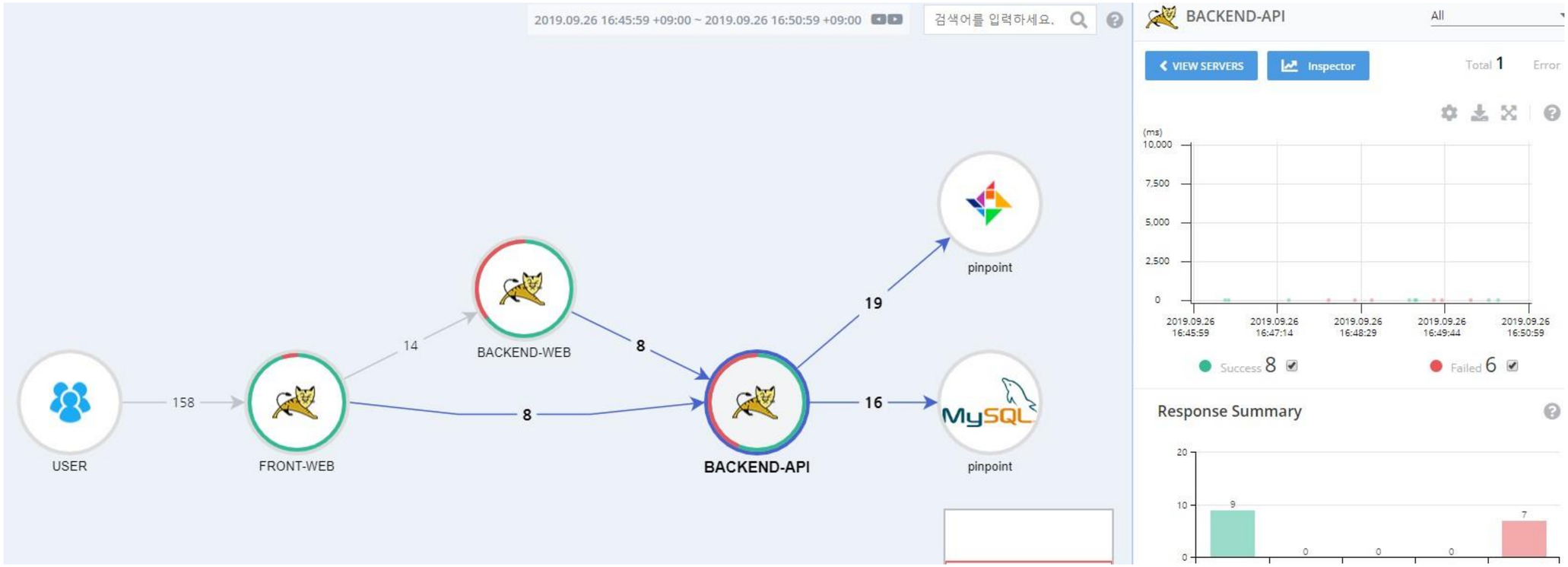


Integration with k8s

DEVIEW
2019

New UI

DEVIEW 2019



서버맵

New UI

DEVIEW 2019

인스펙터

Community

DEVIEW
2019

홍콩, 대만

오픈소스 컨퍼런스

참가 및

세션 진행

Community

DEVIEW 2019

글로벌

APM 리더

초대 및 세미나

진행

@강남

Community

DEVIEW
2019

APM 주요 이슈
공유 및 토론

@ 그린팩토리

Community

글로벌

APM 리더

한국 문화

공유 및

실습

DEVIEW
2019

Q & A

DEVIEW
2019

Thank You