Nerv 3.5 미들웨어

2020년 07월 07일

(주) 웨어그램

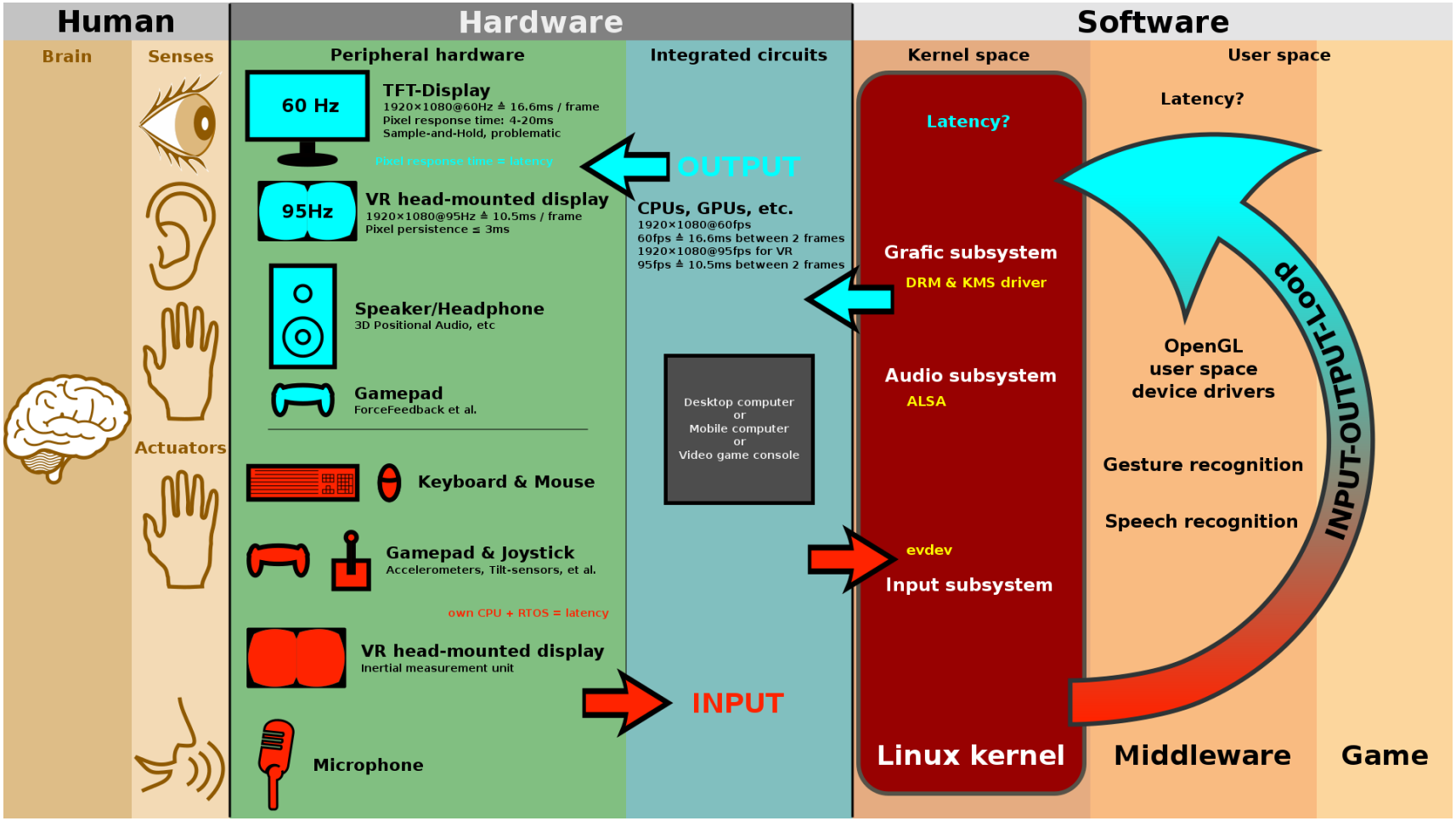
목차

목차

1-1. 미들웨어 개념

사람과 응용 소프트웨어와의 인터페이스 과정

그림출처: 위키



미들웨어 : 운영체제 커널과 사용자 응용프로그램 사이의 모든 소프트웨어

1-1. 미들웨어의 개념

- **미들웨어 장점**
  - 응용 공통 코드 일원화
  - 응용 공통 코드의 재사용성 향상
  - 응용간 미들웨어 인터페이스 표준화
  - 커널에서 지원하지 않는 API 라이브러리 표준화

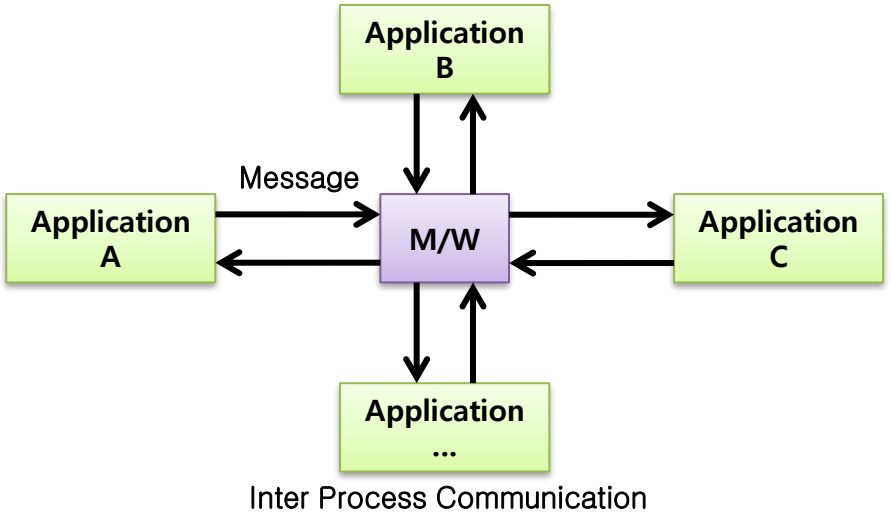
1-2. 미들웨어 종류

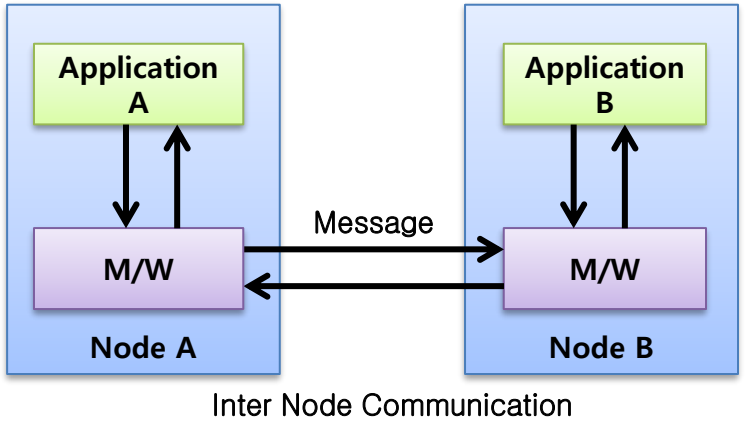
- **형식적 분류법**
  - Library Type
    - 응용의 라이브러리 타입으로 미들웨어 코드 수행
  - Background Process Type
    - 응용의 라이브러리 타입과 병용하여 별도의 프로세스로 동작하여 미들웨어 코드 수행
  - Kernel Driver Type
    - 커널의 디바이스 드라이버로 동작하여 코드 수행

- **기능적 분류법**
  - 분산 처리 미들웨어
    - 이기종 컴퓨터 및 시스템의 응용프로그램 간의 연동을 위한 미들웨어
  - 3D 그래픽 미들웨어
    - 3D 전시를 위한 응용 라이브러리
  - 인공지능 서포트 미들웨어
    - 인공지능 처리를 도와주는 응용 라이브러리

1-3. 분산 처리 미들웨어

- **분산 처리 미들웨어 장점**
  - 표준화된 인터페이스
  - 다양한 환경 지원
  - 이 체계간 상호연동
  - 분산 업무 동시 처리
  - 자료 일관성 유지
  - 부하의 분산





1-4. 분산 처리 미들웨어 분류

- Hurwitz(후르비츠) 분류법
  - Remote Procedure Call M/W
    - 클라이언트에서 원격 서버의 함수를 동기식 또는 비동기식으로 호출
  - Message Oriented M/W
    - 클라이언트에서 생성한 메시지를 저장소에 요청할 때 저장하면서, 다른 업무를 지속할 수 있도록 하는 비동기식 미들웨어
  - Object Request Broker M/W
    - 객체지향 시스템에서 객체 및 서비스를 요청할 수 있도록 지원하는 미들웨어
  - DB Connect M/W
    - 응용프로그램과 데이터베이스 서버를 연결해주는 미들웨어

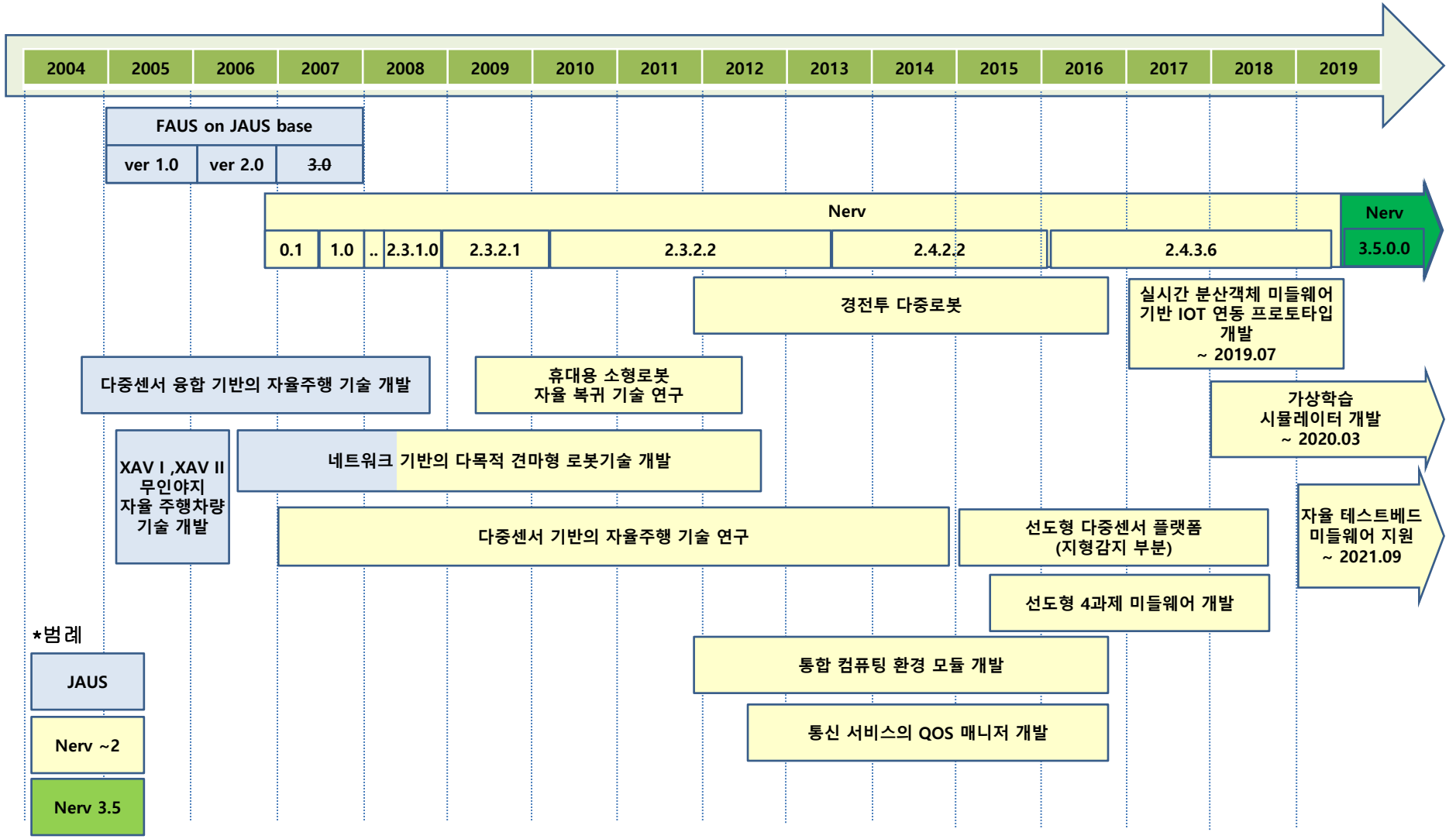
1-3. 분산 처리 미들웨어 분류

- **기타 분류법**
  - Transaction Processing Monitor
    - 분산시스템에서 트랜잭션이 온전하게 처리되고 있는지, 오류가 발생하면 조치를 취하는지에 대해 여러 개의 로컬과 원격 터미널 간의 데이터 전송을 감시하는 통제 프로그램
    - 로컬과 원격의 최소 처리단위인 트랜잭션을 감시하여 일관성을 유지하는 역할
  - Web Application Server
    - 웹 응용프로그램과 서버환경을 만들어 동작시키는 기능을 제공하는 소프트웨어 프레임워크이다.
    - 아파치 톰캣, 닷넷 애플리케이션 서버 등
  - Enterprise Service Bus
    - 메시지 기반으로 느슨한 결합 형태의 표준 인터페이스 통신을 지원하는 미들웨어
    - 서로 다른 미들웨어 간의 변환하는 트랜스

2. Nerv 미들웨어 소개

목차

2-1. Nerv 개발 및 적용 이력



2-2. Nerv 3.5 개발 동기

- **Nerv 의 개발 동기**
  - 로봇에 활용할 미들웨어 부재에 따라 로봇 플랫폼 미들웨어 개발
    - RPC / CORBA와 같은 비 실시간 미들웨어의 시대
    - DDS / ROS 등 부재 시대
  - 초창기 JAUS(Joint Architecture of Unmanned System) 프로토콜 기반으로 FAUS 미들웨어 개발
    - 매우 느린 프로토콜 Update 문제점
    - JAUS 프로토콜 개발 그룹의 버전 3.0 이후 프로토콜 비공개화
  - 로봇에 활용할 신기술 미들웨어의 원활한 공급을 위해 Nerv 개발

- **구버전 Nerv의 장점을 계승하면서 문제점을 해결하는 신기술 미들웨어**
  - 보안성이 있는 미들웨어
  - 트랜잭션의 옵션화
  - 속도 협상의 개선 신뢰성 QUIC 프로토콜 추가

2-3. Nerv 3.5 의 미들웨어 분류

- **형식적 분류**
  - 임베디드 시스템: Library Type
    - 구버전: background process type

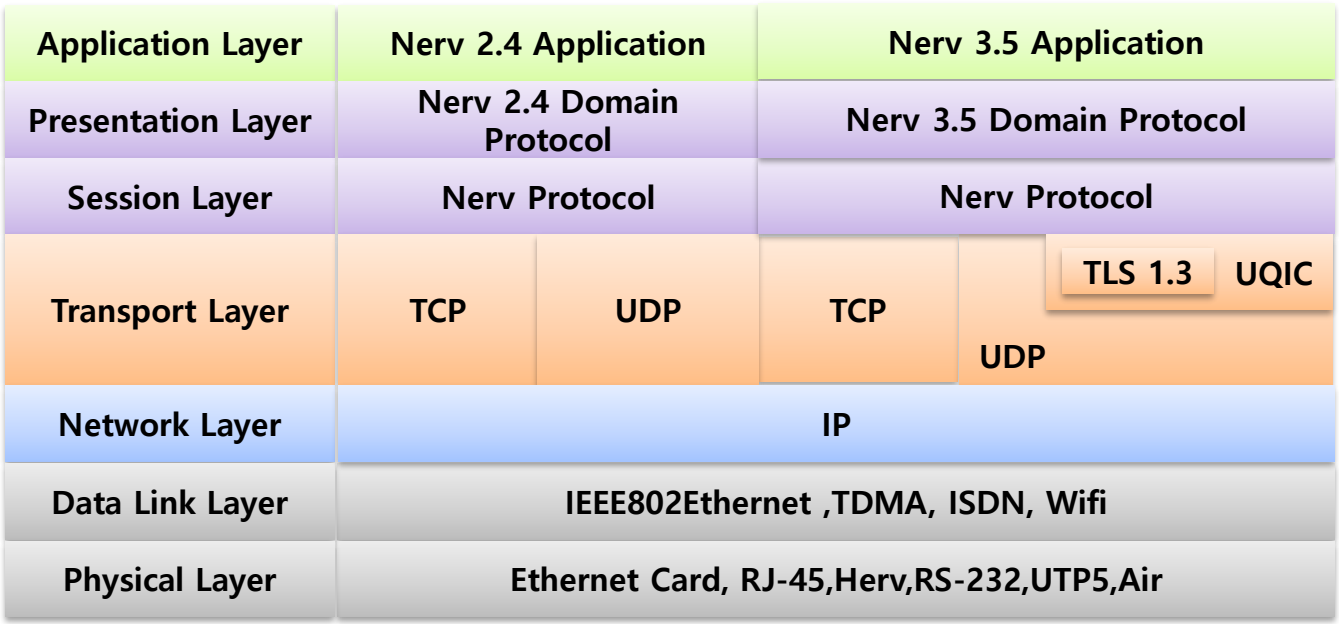
- **기능적 분류**
  - 분산처리 미들웨어
    - Hurwitz(후르비츠) 분류
      - Remote Procedure Call M/W
        - » Call/Signal/Event/Broadcast 트랜잭션
      - Message Oriented M/W
        - » Command/Information 트랜잭션
    - 기타 분류법
      - Transaction Processing Monitor
        - » Call/Signal/Event/Command/Information/Broadcast 트랜잭션 단위 처리

2-4. Nerv 3.5 미들웨어 특징

- **구현 특징**
  - QUIC 기반 구현
    - 신뢰성,정확성을 위한 QUIC 구현
    - 반복성,속도를 위한 QUIC 구현
    - 대역폭 최적화를 위한 QUIC Multicast & Broadcast 구현
  - QUIC/UDP/IP 기반의 OSI 7 계층에서 세션계층(Session Layer)에 대한 Nerv 3.5 Protocol 정의
  - 글로벌 컴퓨팅 개발 환경 도구를 통해서 표현계층(Presentation Layer)에 대한 Nerv 3.5 Domain Protocol 정의 및 관리하는 방안 제공
  - Binary 기반 프로토콜로서 Ascii 기반 대비 실시간성인 로봇 환경에 적합
  - Deadline 제약에 따른 조건부 실시간 예외처리 지원

2-4. Nerv 3.5 미들웨어 특징

- **OSI 7 Layer**



2-4. Nerv 3.5 미들웨어 특징

- **QUIC 기반의 특징**
  - 응용 계층에서 통신 속도 협상
    - 기존 TCP는 OS 계층에서 구식 장비에서 적용하던 통신 속도 협상 방식을 사용하므로 통신 방해 원인 해제이후에도 통신속도가 복구되는데 오래 걸림
    - QUIC는 응용계층에서 통신 속도를 협상하므로 현대 장비에 어울리는 통신 속도 협상 방식을 자유롭게 프로그래밍 가능해 졌음. (단, QUIC 라이브러리 제작사에서 이를 수행하지만 하고 수정가능한 부분임. MS의 Winsock의 TCP는 몇 십년동안 변경되지 아니함)
  - 보안성 추구
    - TLS 1.3을 기초로 하여 구현되므로 보안성이 향상됨.
    - TCP에 TLS 를 올리는 경우 QUIC보다 핸드쉐이크가 많이 발생하기 때문에 성능상의 이유로도 QUIC가 바람직함
  - 가변IP 환경에서 연결성 추구
    - TCP의 경우 모바일(수상정과 같은 모바일 플랫폼)의 IP주소가 변경되면 재할당 및 재연결의 필요성이 발생하고 서로 다른 연결로 인식함
    - QUIC의 경우 IP주소가 변경되어도 기존의 연결과 신규 연결을 동일한 연결로 인식하므로 무선 환경에서도 연결성이 강력함.
  - HTTP3 의 기반 프로토콜
    - HTTP1/1.2/2 는 TCP 프로토콜에 기반하는데 반하여
    - HTTP3 표준은 QUIC 프로토콜에 기반함.
  - 멀티플렉싱 기술 불필요
    - 응용계층에서 스트림 처리를 수행하므로 TCP와 같이 Windows는 IOCP/리눅스는 Epoll/유닉스는 K-queue와 같은 OS 기반의 멀티플렉싱 기술 불필요하므로 OS간 코드 호환성 향상

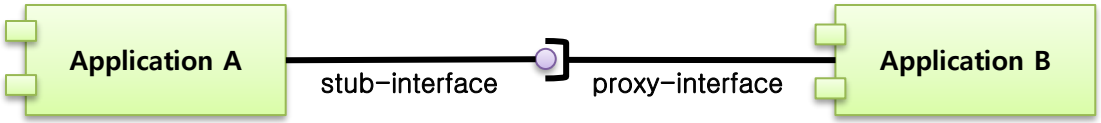
2-4. Nerv 3.5 미들웨어 특징

- **구 버전 Nerv 는 세션 공유 및 메모리 공유를 활용 함에 따라 객체 ID 존재**
- **Nerv 3.5는 공유메모리 사용하지 않음으로 주소체계 간결화 함.**

- **No Broker M/W**
  - Proxy-Stub 객체 동기화 처리
  - 객체 주소
    - 구버전: IP/Domain(Port)/Object ID
      - IP주소: 노드 식별
      - Domain 주소: 논리 영역 식별
      - Object ID: 단위 객체 식별
    - Nerv 3.5: IP/Port
      - IP주소: 노드 식별
      - Object ID: 단위 객체 식별
  - 응용프로그램의 컴포넌트화
    - Proxy-Stub 개념

Offset	1byte	1byte	1byte	1byte
0byte	Domain Addr		Object ID	
4byte	IP Address			

Offset	4byte	2byte
0byte	IP	Port



Component Diagram (UML)

    - 하나의 객체에 0개 이상의 메서드로 구성된 객체를 작성

2-4. Nerv 3.5 미들웨어 특징

- **Named 객체 생성 방법**
  - 구버전
    - Stub.create(domain_port, object_id);
    - Proxy.create(domain_port, stub_ip, stub_object_id);
  - 3.5
    - Stub.create(stub_name, port);
      - Ex> Stub.create(0, 10000); // 이름없는 객체
      - Ex> Stub.create("stub_name", 0); // 자동port할당하는 이름있는 객체
      - Ex> Stub.create("stub_name", 10000); // 10000 고정 port 사용하는 이름있는 객체
    - {stub_ip, stub_port} = Nerv35::get_address("stub_name");
    - Proxy.create({stub_ip, stub_port});
      - IP/port 주소를 직접 지정 가능
      - gethostbyname / getaddrinfo를 DNS서버로부터 IP 획득 및 고정 Port 사용
      - Nerv35::get_address 로부터 주소를 획득해서 사용가능
        - » 로컬 네트워크에서만 사용가능...

2-4. Nerv 3.5 미들웨어 특징

- **처리 개념의 명칭 변경**

언어 / 환경	Nerv 2.4	Nerv 3.5
객체 활성화	Create / Destroy (center 구동및 connect/listen/accept활동등을 하는 객체를 생성하고 파괴함)	Start / Stop (connect/listen/accept 활동을 시작하거나 중단함)
Proxy-Stub 객체간 연동	On_create / on_destroy	On_connect / on_disconnect
메서드 연결	Connect / disconnect /shutdown	Link / unlink
세션 연결	On_connected tcp / on_disconnected_tcp (기존 구현은 되어 있으나 대게 사용하지 않음)	세션간 연결이 객체간 연결과 1:1 대응으로 추가처리 안함.

2-5. Nerv 3.5 미들웨어 지원 환경

환경 \ 언어	C++	C#	Python	비고
x86 32/64bit Windows 7 이상	VS 2019 QT Creator + vc/mingw-gcc	20년 7월	지원 중	
x86 64bit Linux 우분투 16.04	Eclipse/QT-Creator + gcc	20년 7월	지원 중	

3. 미들웨어 비교

목차

3.1 Nerv 3.5 & Nerv 차이점

비교사항	Nerv 3.5	Nerv 2.4
프로토콜	QUIC 기반	TCP/UDP
보안	TLS 1.3	없음
트랜잭션	메서드에 트랜잭션 옵션 Ex> ins 메서드 선언하고 Call/Signal/Event/ 서브 트랜잭션 메서드 중 선택사용	트랜잭션에 메서드 식별 Ex> ins 메서드를 Information 트랜잭션으로 선언 다른 트랜잭션으로 사용하기 위해서 각각 선언필요
가변IP대응	가능	불가능
이름있는 객체	있음	없음
백그라운드	없음. QUIC 통신 기반	있음. 공유메모리 기반 IPC 통신
	로컬 컴포넌트끼리 통신시 공유메모리 기반으로 통신이 빠르지만, 공유메모리 기반 통신은 응용프로그램끼리 메모리 버그 간섭이 심하여 디버깅이 어려운 문제가 있으므로 Nerv 3.5는 이를 배제함. 공유메모리 기반 IPC 통신의 성능이 소켓기반 통신의 성능보다 10배 빠르다 하여도 현시대의 CPU성능에 와서 전체 CPU 사용률 관점에서는 미미한 차이이기 때문에 공유메모리 기반의 IPC 통신의 성능향상이 시스템 성능 향상으로 기대하기 어려움. 따라서 Nerv 3.5는 로컬 컴포넌트간 디버깅시 잘잘못을 따지기 쉬운 소켓기반으로 설계됨.	
통신속도협상	Application – 빠른 속도회복	OS – 느린 속도 회복
	무선 상황에서 통신 방해요인(거리,재밍 등)으로 인해 통신속도가 저하되었다가 다시 회복되었을때 TCP 보다 QUIC가 빠르게 본래의 통신속도로 회복할수 있음. TCP가 오래된 프로토콜이기 때문에 시대 변화에 대한 대응에 실패한 프로토콜될 가능성이 있음.	

3.1 Nerv 3.5 & DDS 차이점

비교사항	Nerv 3.5	DDS
프로토콜	QUIC 기반	UDP
보안	TLS 1.3	없음. 최신프로토콜은
개념	원격 객체를 로컬 객체처럼 사용	원격 데이터를 로컬 데이터처럼 사용
	DDS는 기본적으로 UDP에 데이터를 실어서 브로드캐스트하여 원격의 다른 장치가 잘 읽어 들이는 것에서 출발함. 때문에 원격함수 호출과 같은 기능을 위해서는 데이터 동기화를 이용하여 함수호출을 구현하는 기형적 구현을 하거나 Corba 와 같은 별도의 미들웨어를 추가로 사용하여야 함. Nerv 3.5는 기본적으로 객체를 동기화 하는데 개념을 가지기 때문에 단순히 원격의 C++의 개체를 로컬의 C++ 객체처럼 사용하자라는 것에서 개념이 출발함. 때문에 원격함수 호출과 원격 데이터 동기화의 기능이 모두 내포하고 있음.	
가변IP대응	가능	불가능
이름 있는	객체	토픽
백그라운드	없음. QUIC 통신 기반	없음. UDP 기반 통신
특허	Open Source	US Patent.
	DDS관리 기구인 OMG는 IETF와 같은 표준화 기구가 아니기 때문에 표준이라 하기 어려움. DDS는 이미 비대해진 Corba를 대신하여 데이터동기화를 위해 나왔으나 OMG의 Cobar가 몰락한 것 처럼 OMG의 DDS 또한 같은 길을 갈 것으로 판단됨.	

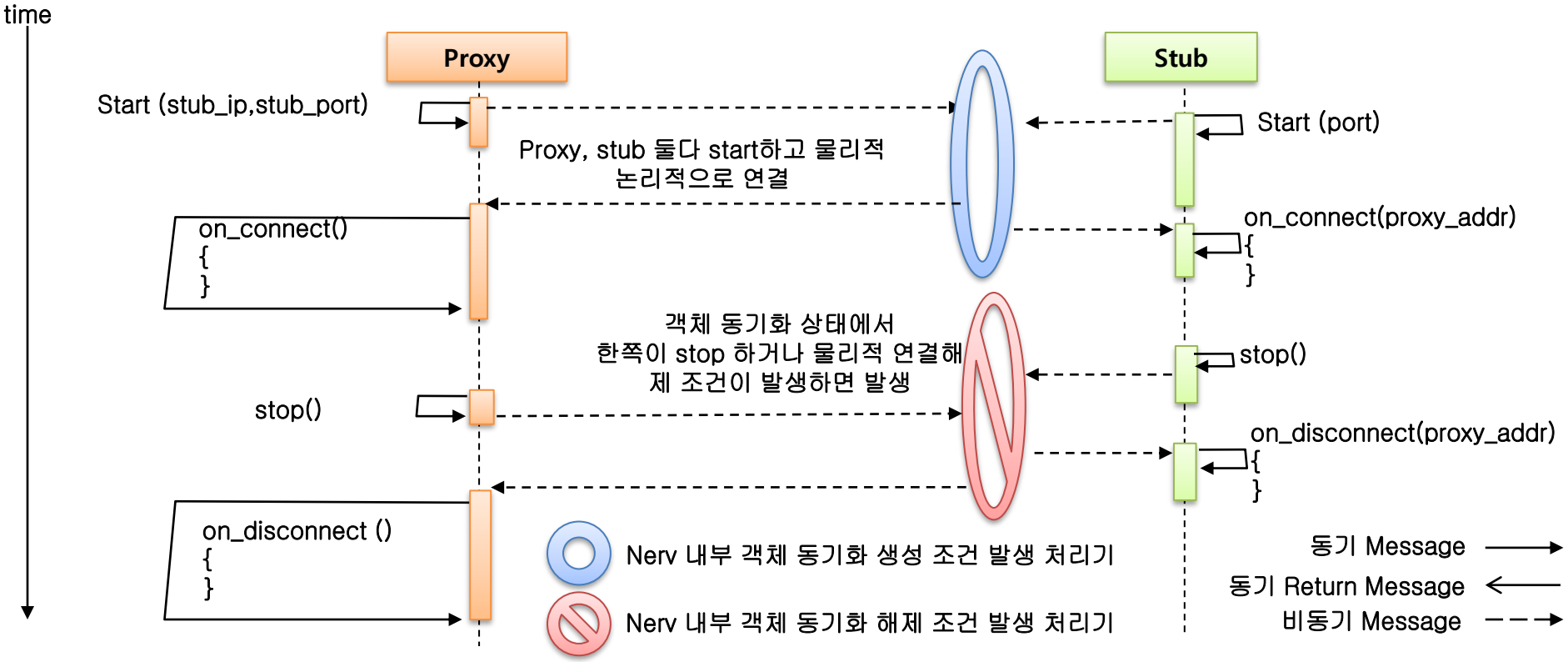
4. Nerv 3.5 미들웨어 동작 개념

목차

4-1. 객체간 연결 처리

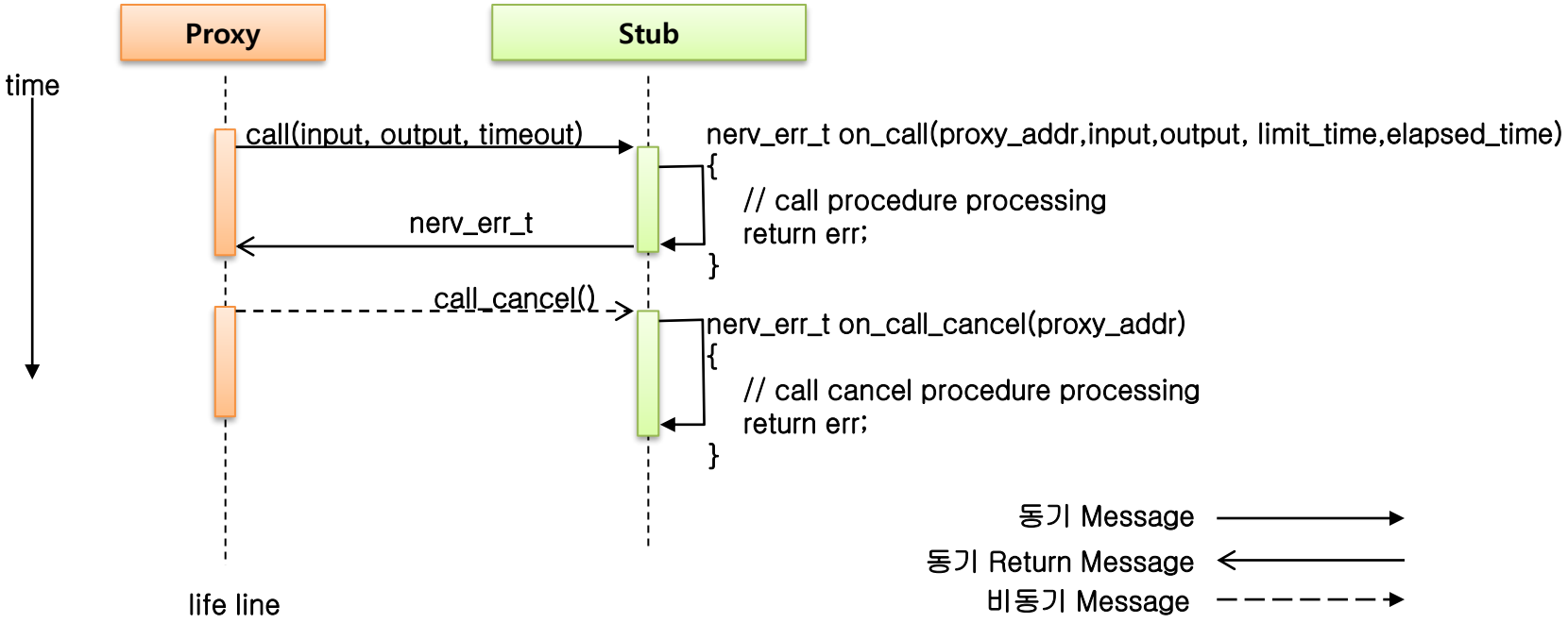
- Call/Signal/Event/Command/Information의 트랜잭션은 connect 상태가 이루어져야 이용 가능함

- 객체 지향 컨셉
- 스트림(TCP/QUIC) 통신으로 구현
- 1:1 단위 동기화 처리
- N:1 객체 동기화 가능



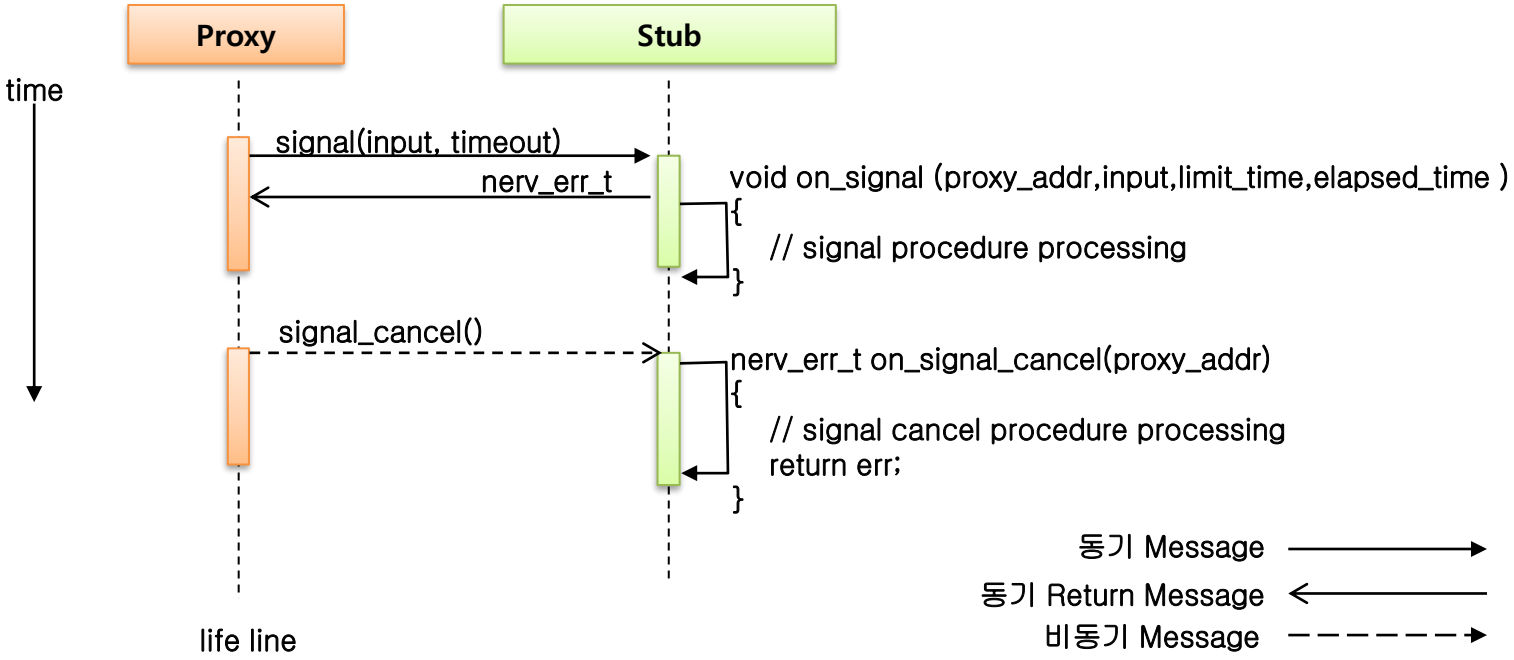
4-2. Call 트랜잭션 처리

- RPC M/W 분류 근거
- Data Input/Out
- 함수 호출 컨셉
- 동기식 병렬 처리 컨셉 활용 가능
- Timeout Deadline
- Synchronized Remote Procedure Call
- 스트림(TCP/QUIC) 통신으로 구현
- 1:1 단위 트랜잭션 처리



4-3. Signal 트랜잭션 처리

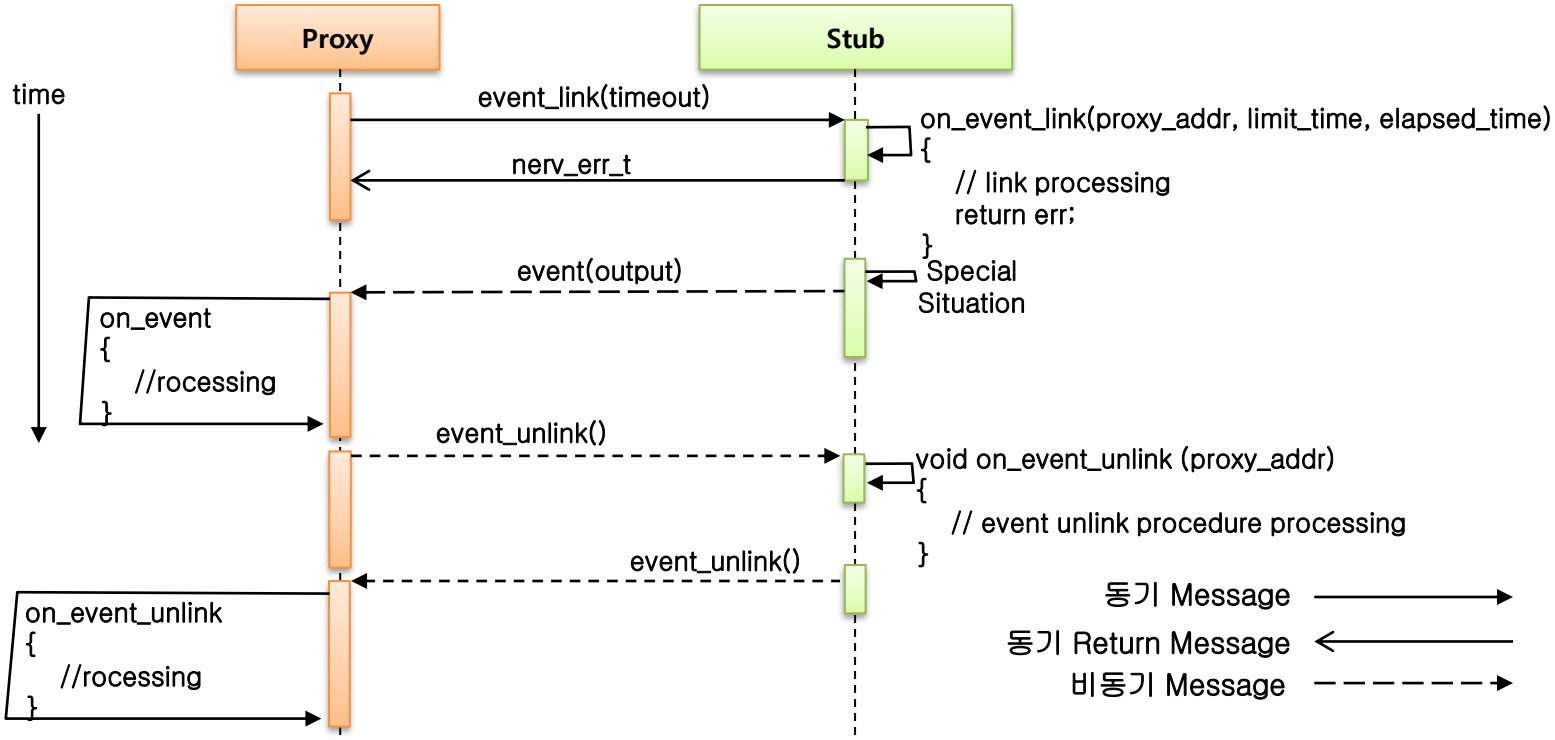
- RPC M/W 분류 근거
- Data Input
- 시그널 실행 컨셉
- 비동기식 병렬 처리 컨셉
- Timeout Deadline
- Asynchronous Remote Procedure Call
- 스트림(TCP/QUIC) 통신으로 구현
- 1:1 단위 트랜잭션 처리



4-4. Event 트랜잭션 처리

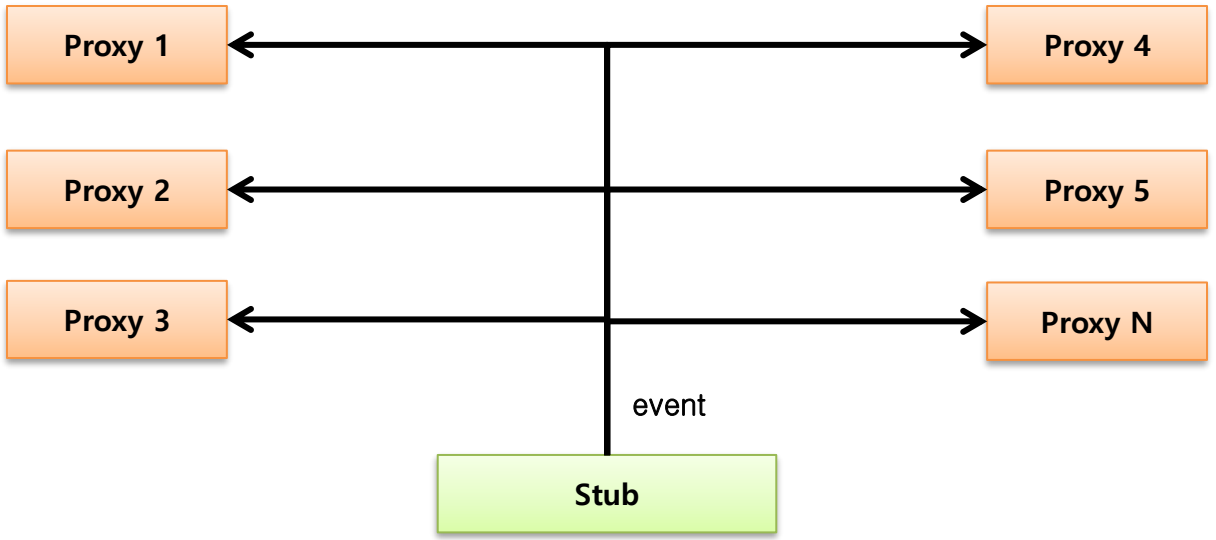
- RPC M/W 분류 근거
- Link 과정 필요
- Data Output
- 콜백함수나 인터럽트 실행 컨셉

- Reverse Asynchronous Remote Procedure Call
- 스트림(TCP/QUIC) 통신으로 구현
- N:1 단위 트랜잭션 처리

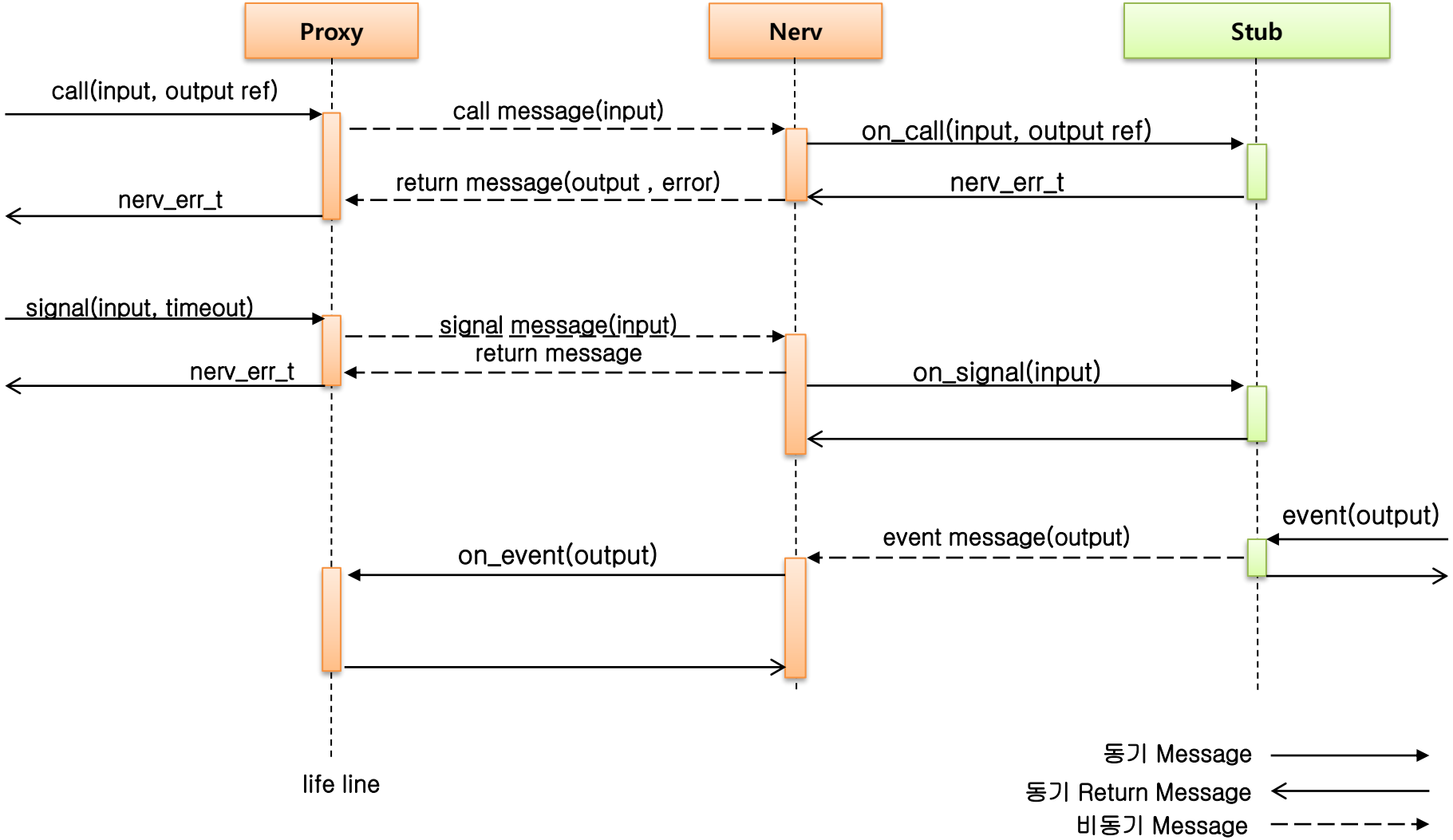


4-4. Event 트랜잭션 처리

– Connect 상태인 모든 Proxy에게 동시에 이벤트 처리 함수를 실행 하도록 함



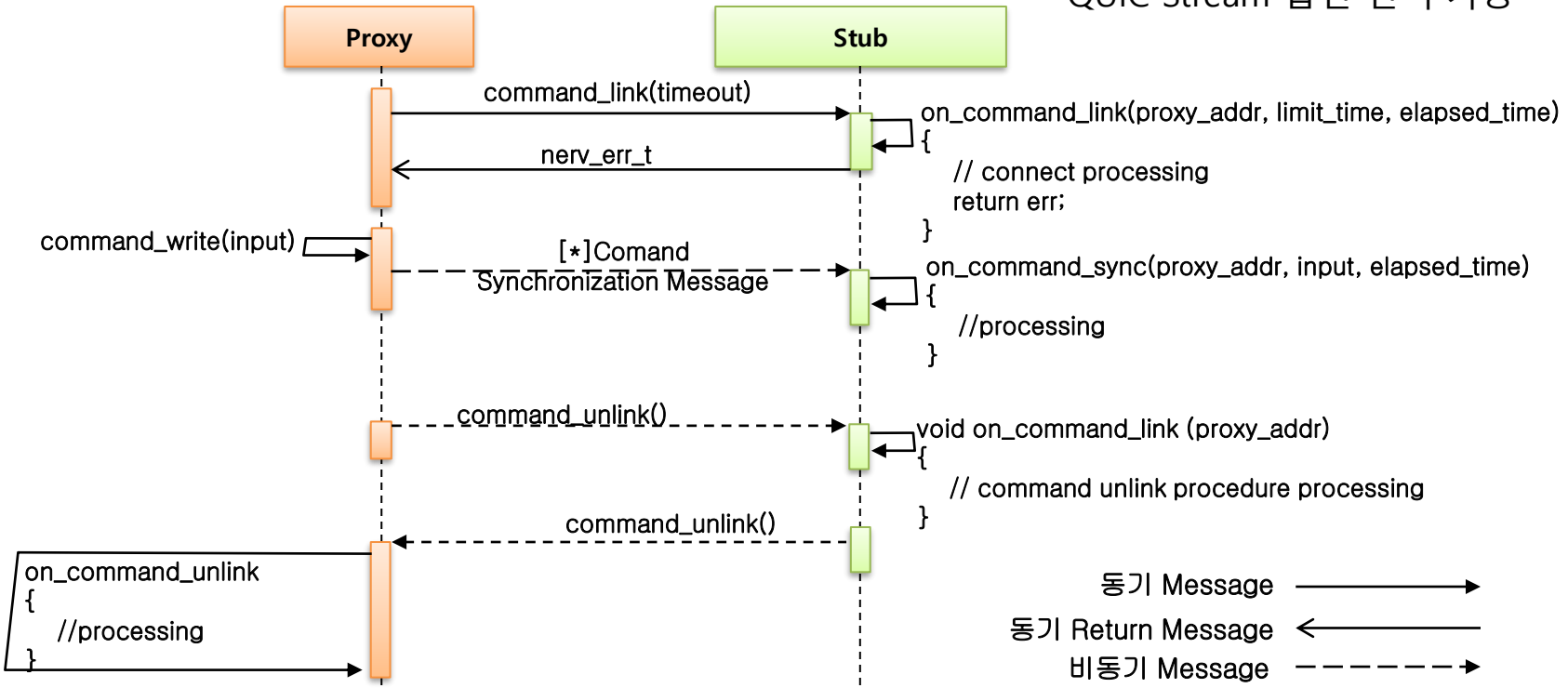
4-8. Call, Signal, Event 정리



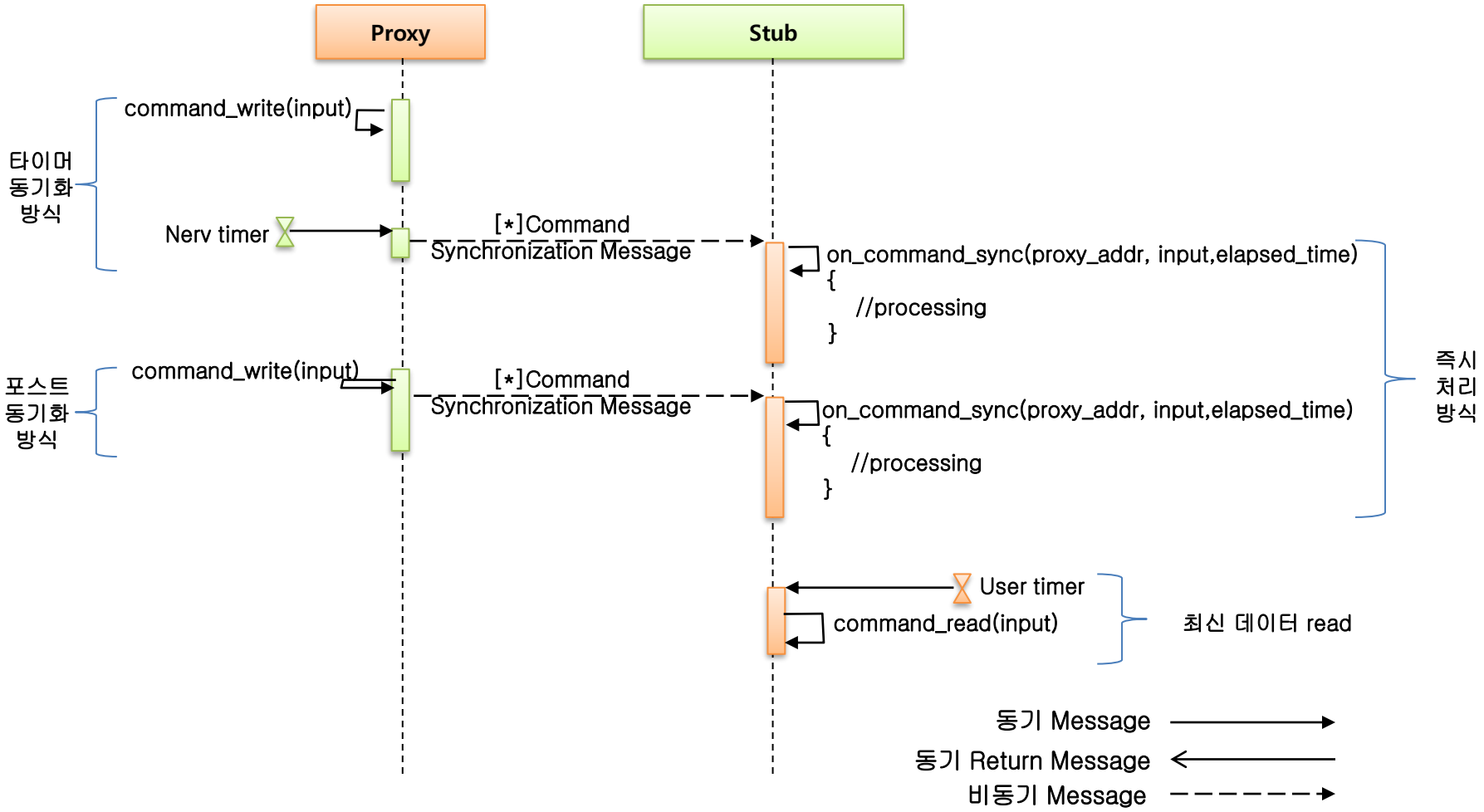
4-5. Command 트랜잭션 처리

- Link 과정 필요
- Data Input
- 객체의 쓰기 멤버 데이터 컨셉
- N:1 단위 트랜잭션 처리

- Message Oriented M/W 분류 근거
- QUIC Datagram 통신으로 구현
  - 65000byte이상 output 데이터는 실제로 QUIC Stream 전송
  - UDP일때 Unicast
  - QUIC Stream 옵션 선택 가능

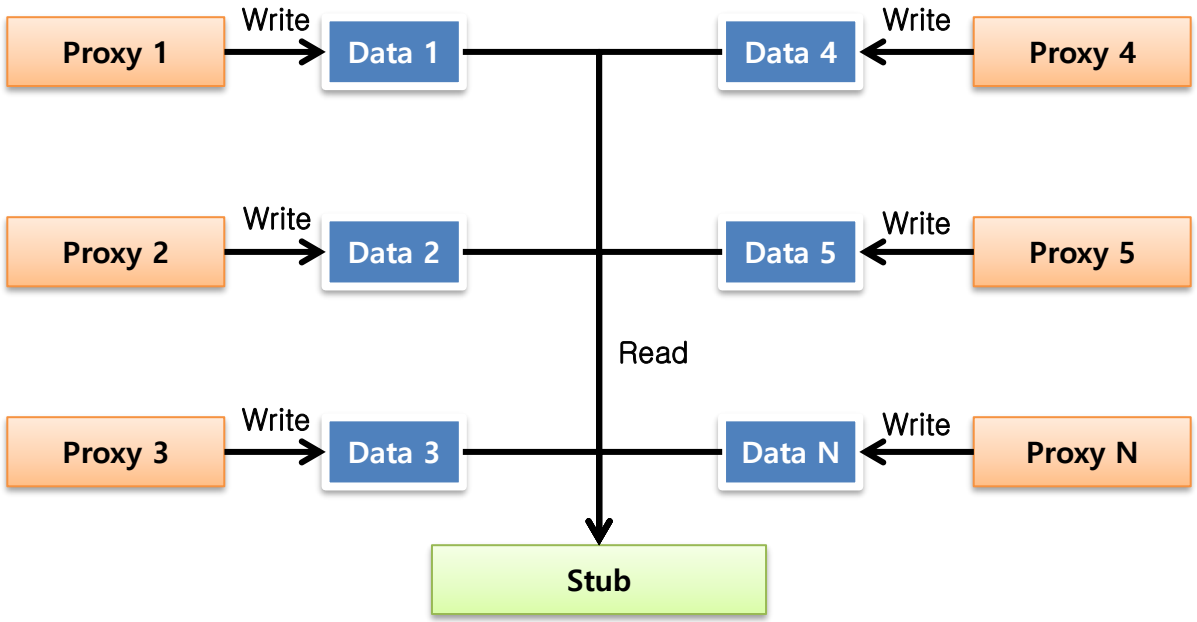


4-5. Command 트랜잭션 처리



4-5. Command 트랜잭션 처리

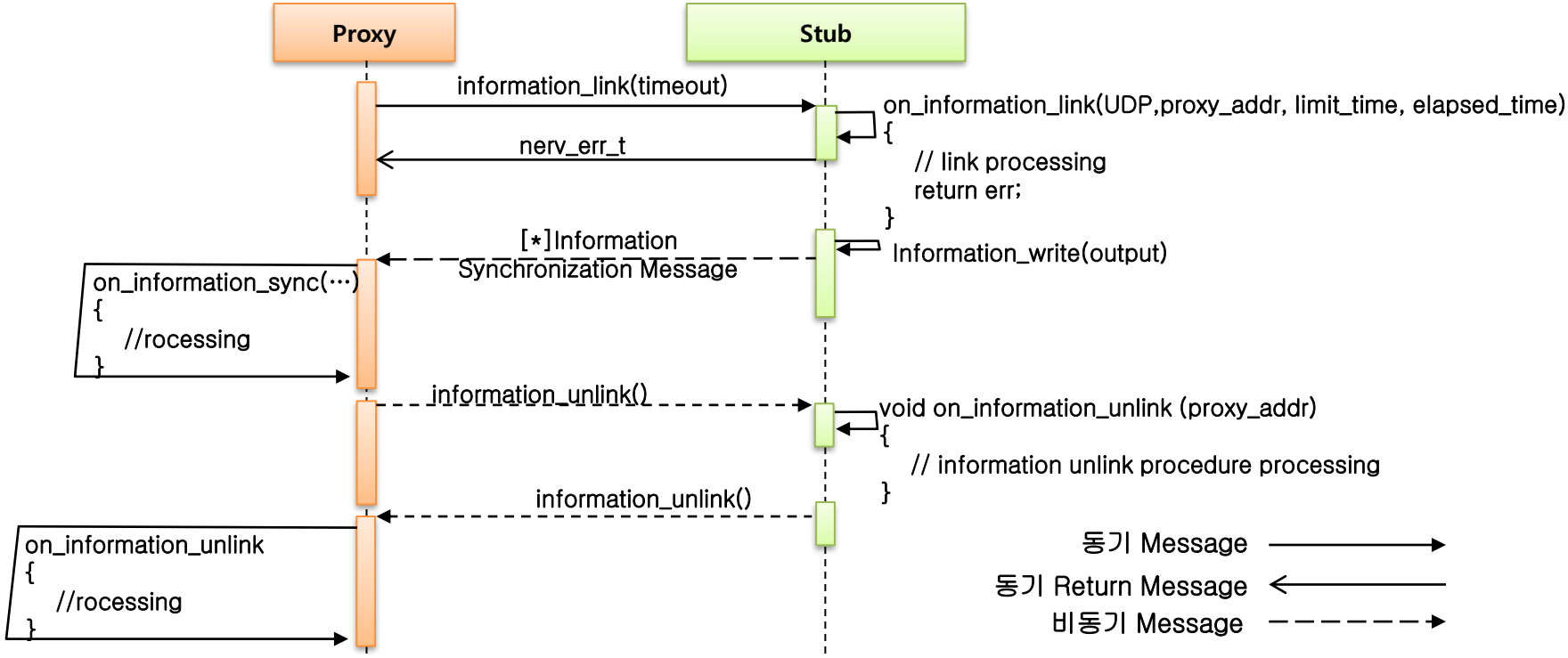
– N개의 Proxy에서 N개의 Data가 1개의 Stub에 전송



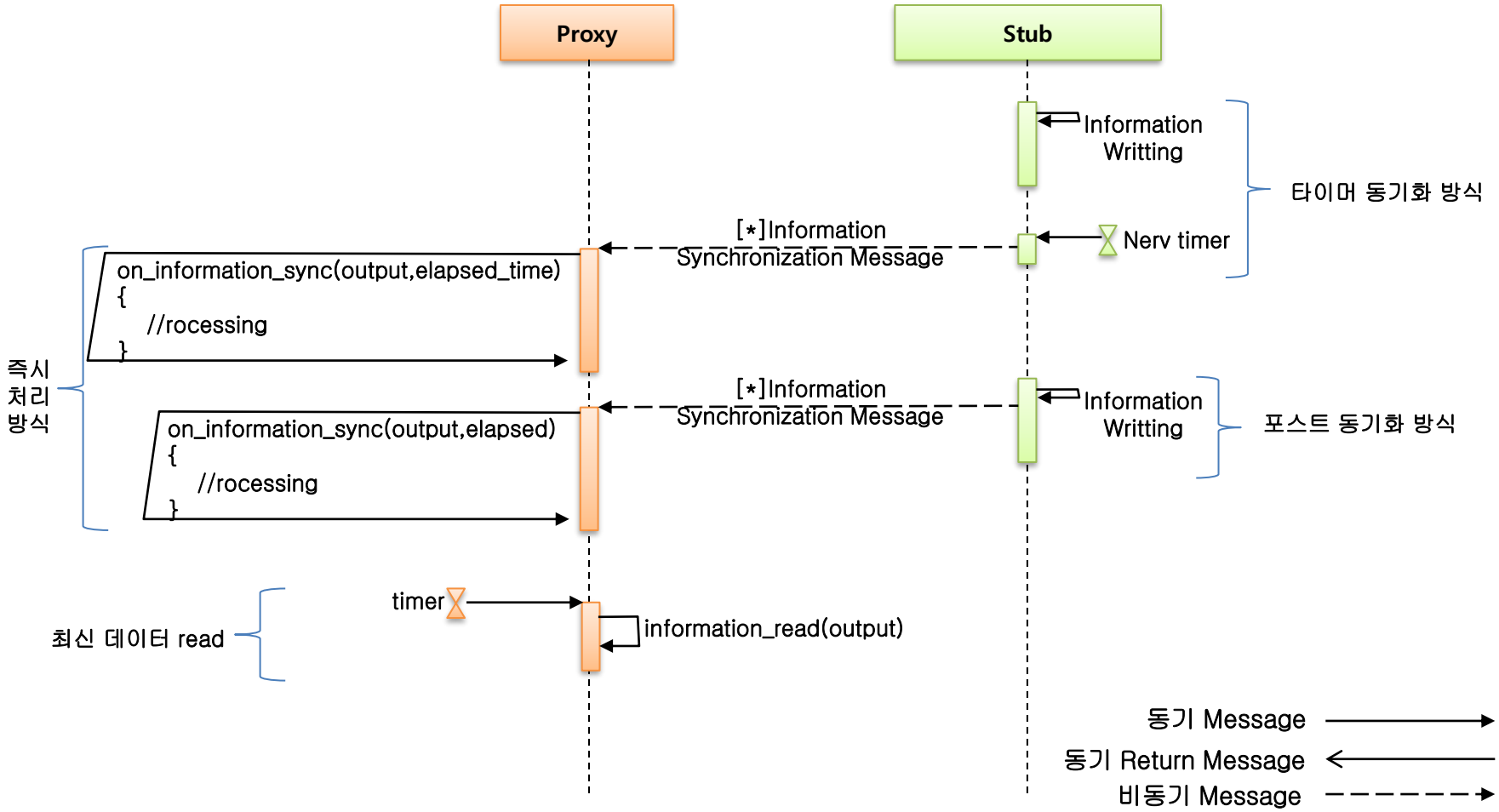
4-6. Information 트랜잭션 처리

- Link 과정 필요
- Data Output
- 객체의 읽기 멤버 데이터 컨셉
- N:1 단위 트랜잭션 처리

- Message Oriented M/W 분류 근거
- QUIC Datagram 통신으로 구현
  - 65000byte이상 output 데이터는 실제로 QUIC Stream 전송
  - UDP Multicast 옵션 선택 가능
  - QUIC Stream 옵션 선택 가능

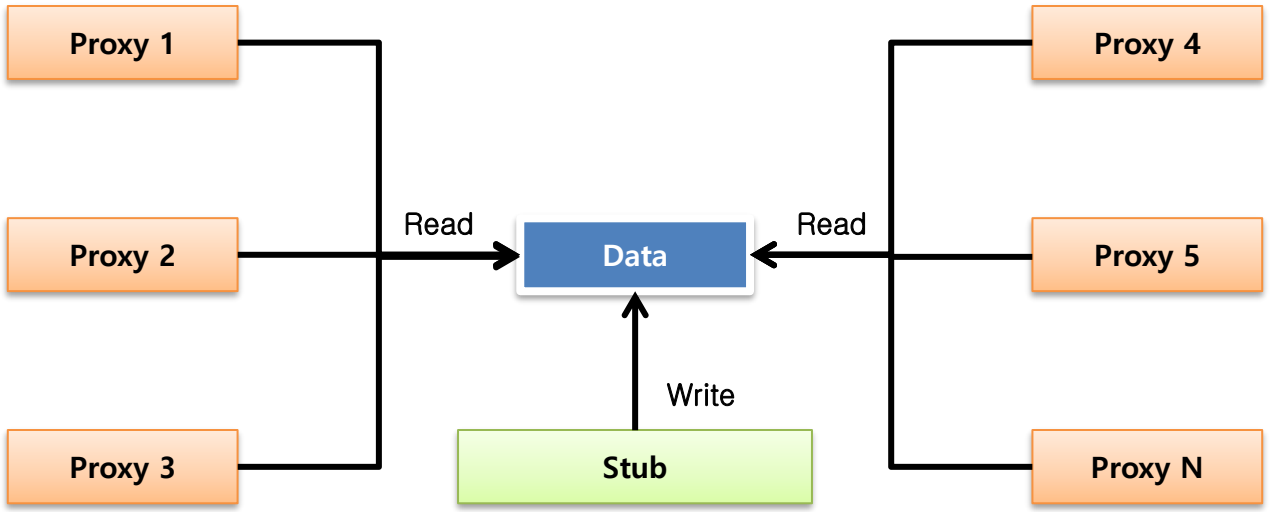


4-6. Information 트랜잭션 처리



4-6. Information 트랜잭션 처리

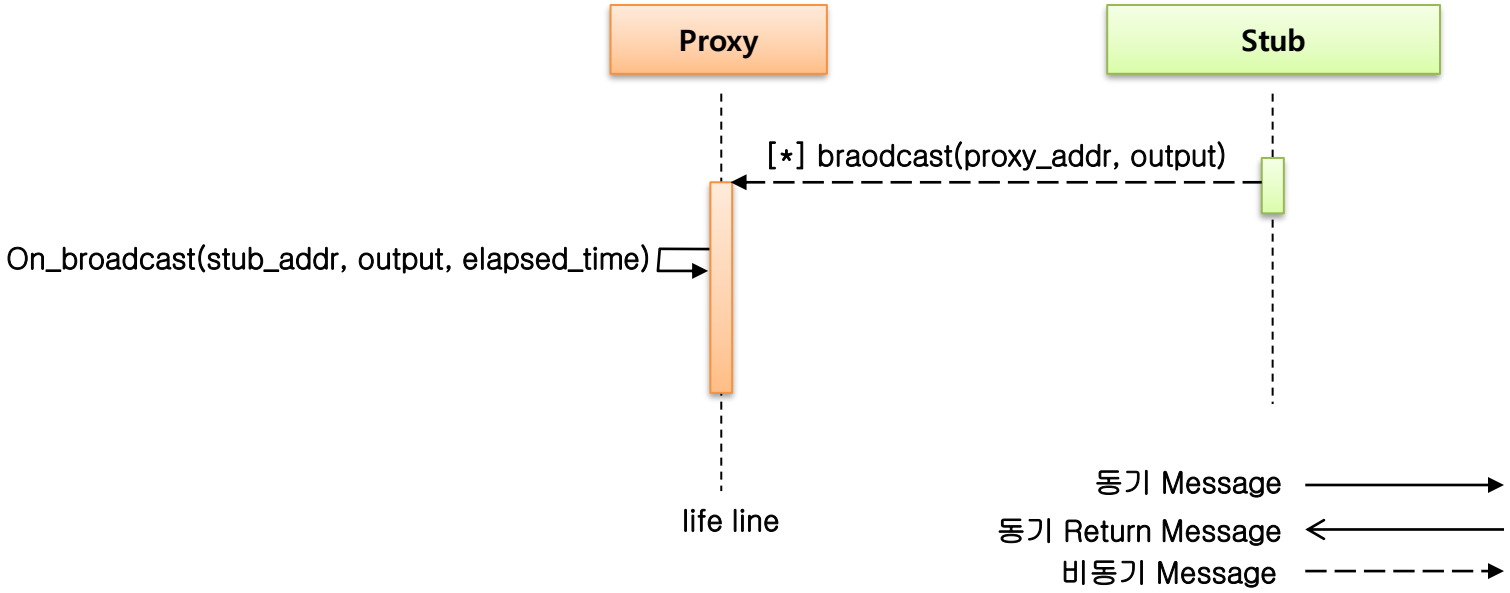
– Stub의 1개의 데이터를 N개의 Proxy에서 공유하여 사용



4-7. Broadcast 트랜잭션 처리

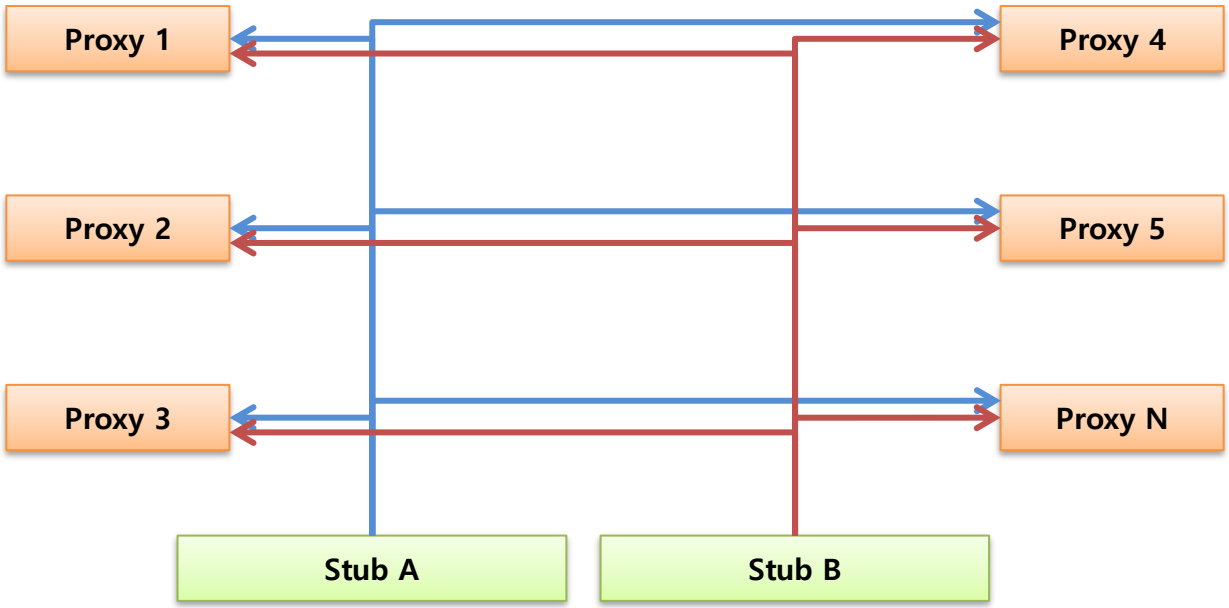
- 객체 동기화 불필요
- Plug & Play 기능 구현 가능
- N:M 단위 트랜잭션 처리
- UDP로만 구현
  - 65000byte 크기 제약
  - UDP Unicast/Broadcast로 구현

- 구번전에서도 잘 쓰이지 않음.
- 3.5부터 Named object 기능의 추가로 응용 구현 없이 plug & play 기능이 가능함에 따라 실제 활용성 떨어짐
- Stub port를 같게 운용하는 특수한 환경에서 활용성 있음.



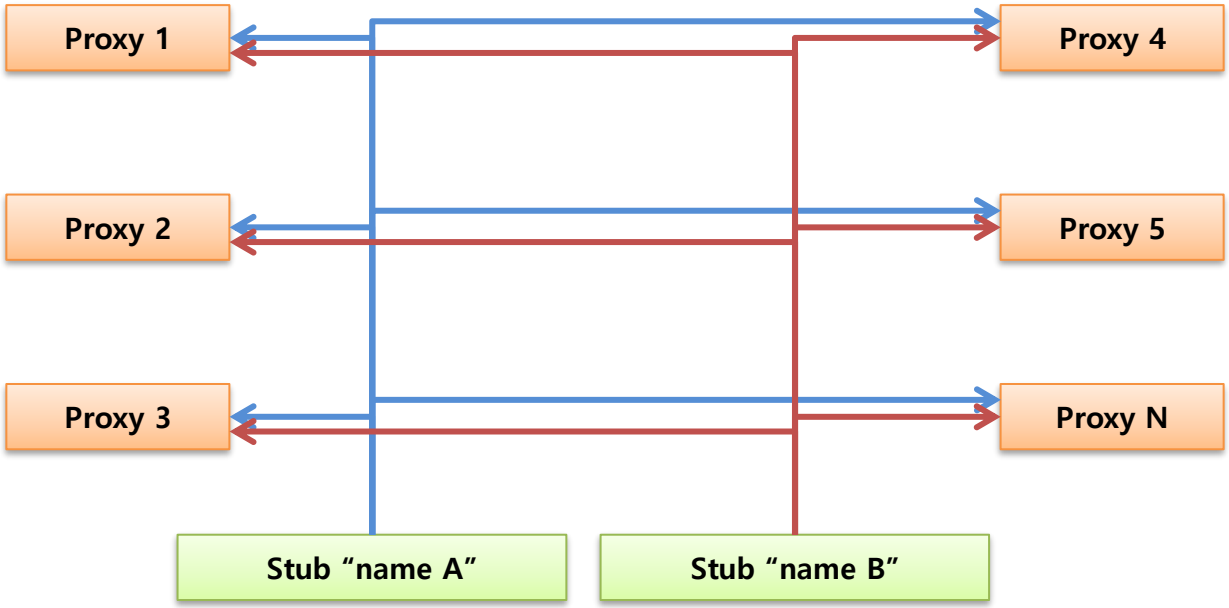
3-7. Broadcast 트랜잭션 처리

- 다른 트랜잭션의 경우 proxy:stub = N : 1
- Broadcast 트랜잭션은 proxy:stub = N : M



4-7. Named Stub Object

- Stub 객체에 이름을 주는 경우 1초에 한번씩 Broadcast로 객체정보를 송신한다.
- 수신측은 8초간 카운트다운하여 객체정보를 유지하다 삭제한다.
- 신규정보를 수신하는 경우 항상 8초 카운트다운으로 리셋 갱신된다.
- Proxy는 get_address함수를 이용하여 구축된 named tables을 이용하여 원하는 stub객체의 주소를 획득하여 연결할 수 있다.



목차

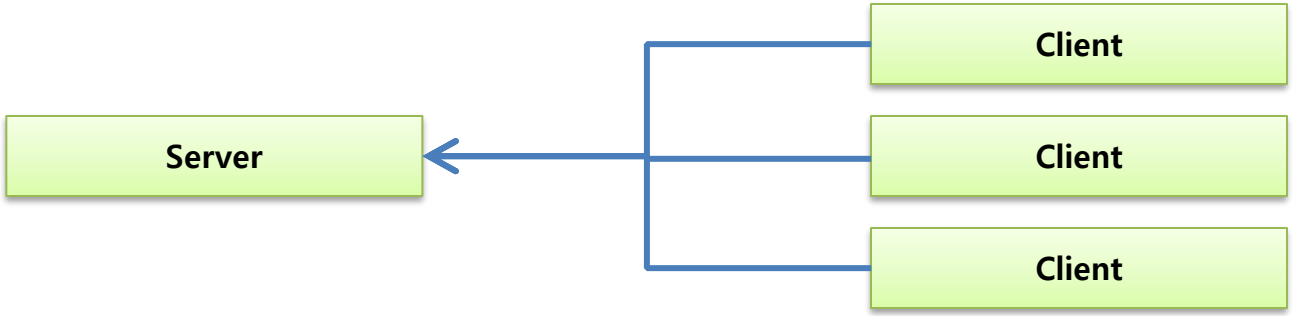
5-1. 헤더파일

헤더파일	설명
nerv_define.hh	호환성 코드 유지를 위한 define
nerv_error.hh	에러 상수 값 정의
nerv_history.hh	nerv_history 클래스 및 history 관련 템플릿 정의
nerv_object_broadcast.hh	nerv_object_broadcast 객체 메서드 클래스 정의
nerv_object_call.hh	nerv_object_call 객체 메서드 클래스 정의
nerv_object_command.hh	nerv_object_command 객체 메서드 클래스 정의
nerv_object_event.hh	nerv_object_event 객체 메서드 클래스 정의
nerv_object_information.hh	nerv_object_information 객체 메서드 클래스 정의
nerv_object_signal.hh	nerv_object_signal 객체 메서드 클래스 정의
nerv_object.hh	nerv_object 객체 클래스 정의
nerv_pre_define.hh	클래스간 의존성을 위한 전방 선언
nerv_proxy_object_broadcast.hh	nerv_proxy_object_broadcast 대리객체 클래스 정의
nerv_proxy_object_call.hh	nerv_proxy_object_call 대리객체 클래스 정의

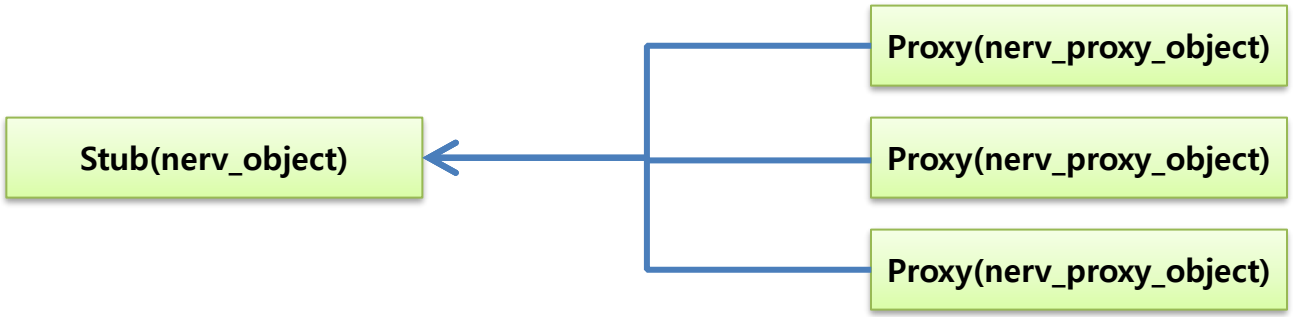
5-1. 헤더파일

헤더파일	설명
nerv_proxy_object_command.hh	nerv_proxy_object_command 대리객체 클래스 정의
nerv_proxy_object_event.hh	nerv_proxy_object_event 대리객체 클래스 정의
nerv_proxy_object_information.hh	nerv_proxy_object_information 대리객체 클래스 정의
nerv_proxy_object_signal.hh	nerv_proxy_object_signal 대리객체 클래스 정의
nerv_proxy_object.hh	nerv_proxy_object 대리객체 클래스 정의
nerv_scaled_integer.hh	scaled integer 템플릿 정의
nerv_serial_paraller.hh	메시지 직렬/병렬 처리 제어 함수 정의
nerv_typedef.hh	기본 타입 및 구조체 정의
nerv_utility.hh	유틸리티 함수 정의
Nerv.hh	통합 Include 정의

5-2. Proxy-Stub 모델



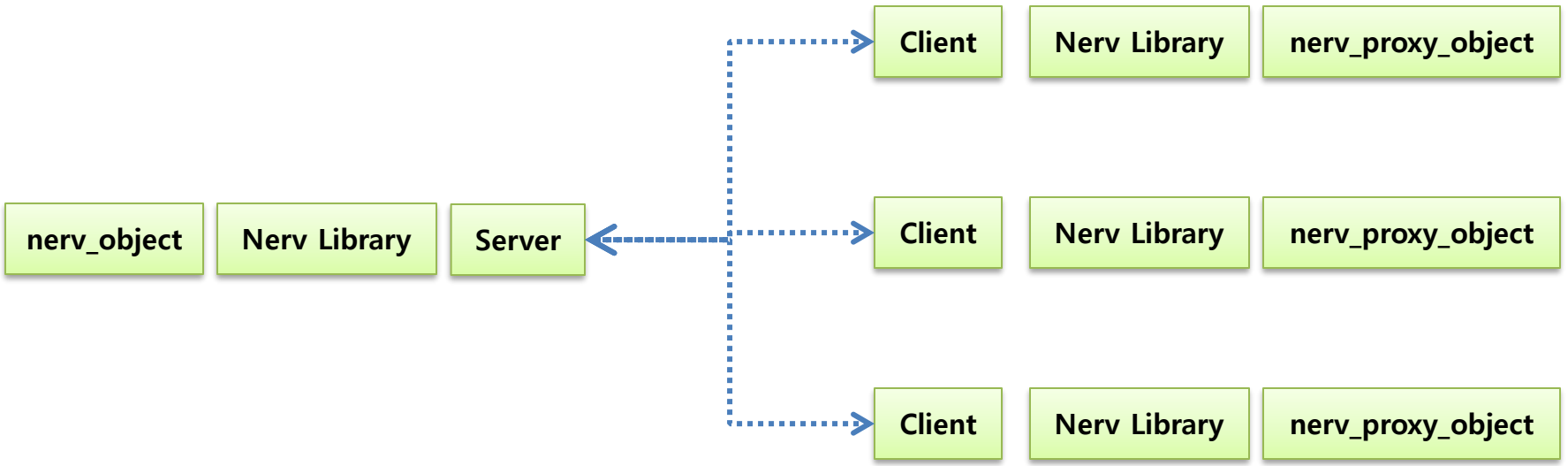
클라이언트-서버 모델 : 스트림이나 데이터그램을 송수신하기 위한 소켓 구조



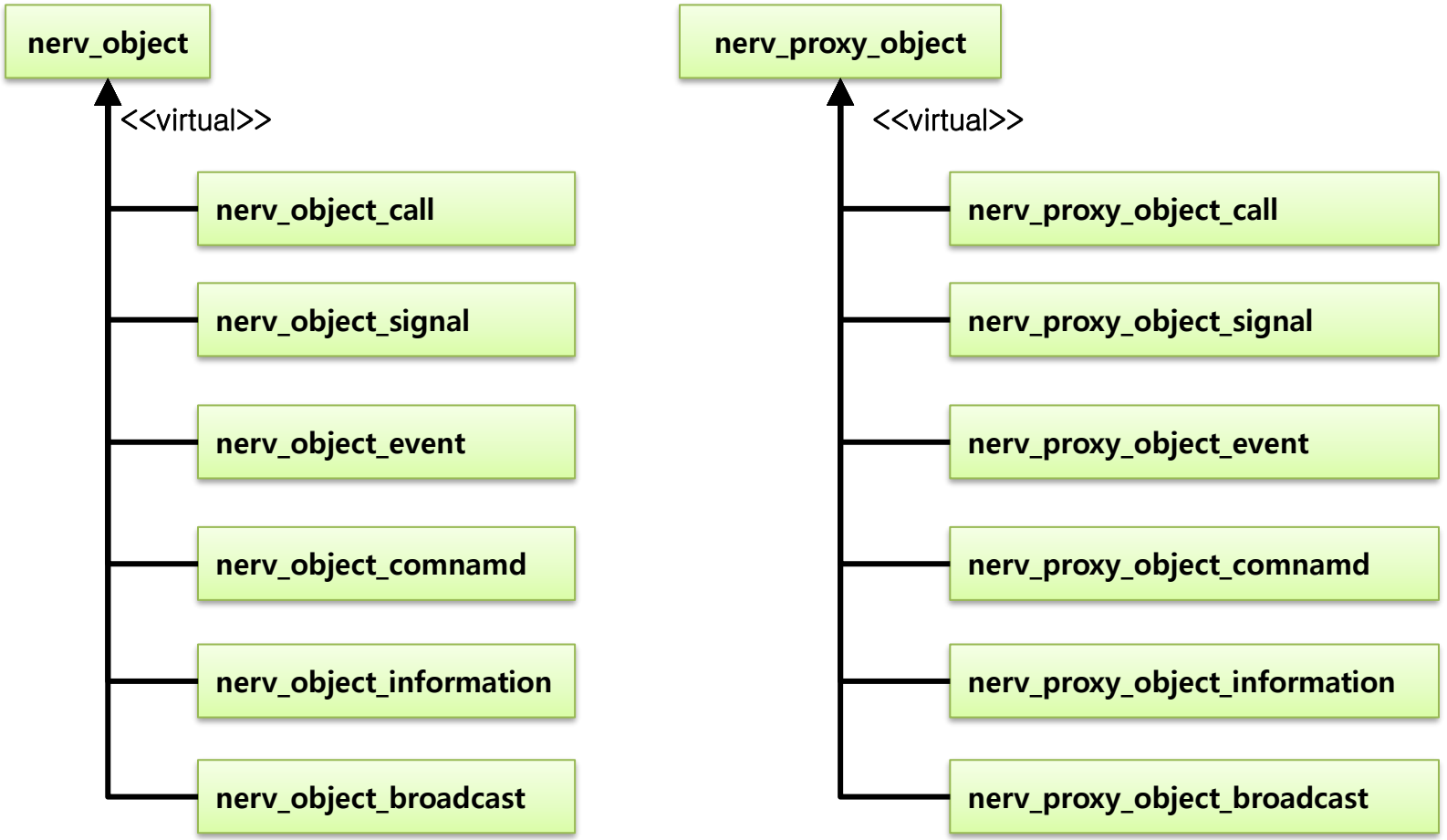
프록시-스텁 모델 : Proxy Object를 조작하면 연동되어 Stub Object가 조작되는 구조

Nerv는 내부적으로 클라이언트-서버 모델을 사용하여 외부적으로 프록시 스텁 모델로 표현한다.

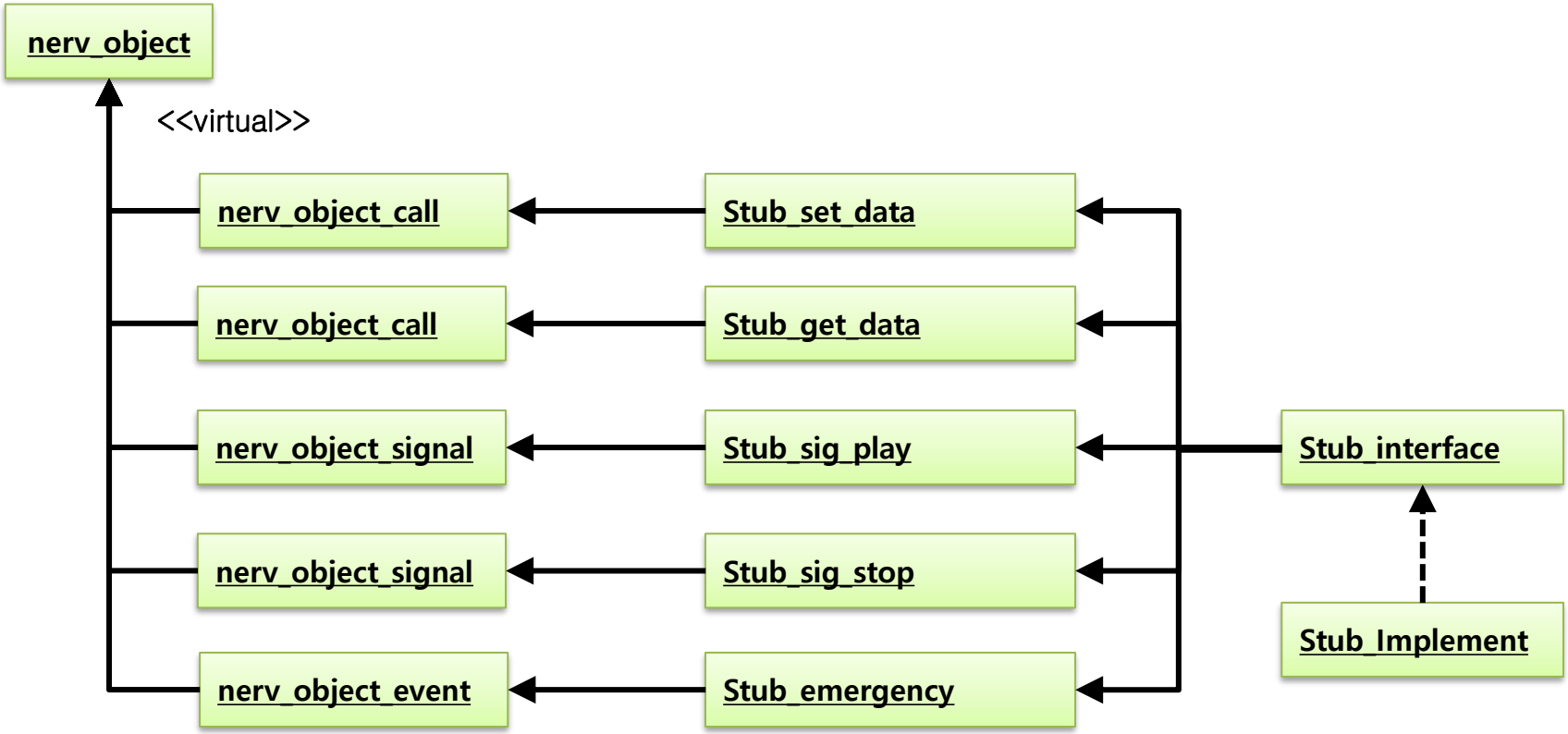
5-3. Nerv의 Proxy-Stub 구조



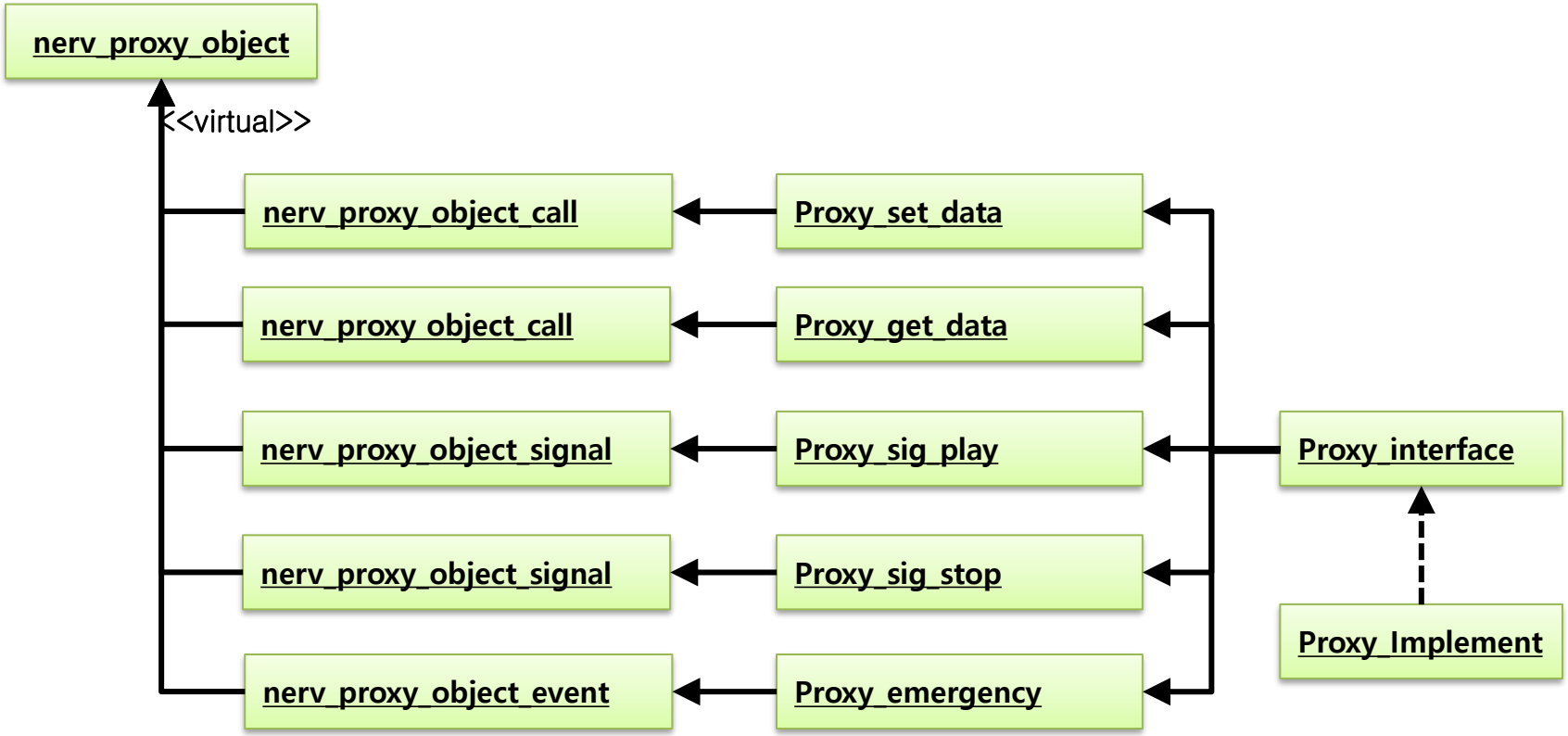
5-4. Nerv 주요 Class 상속 구조



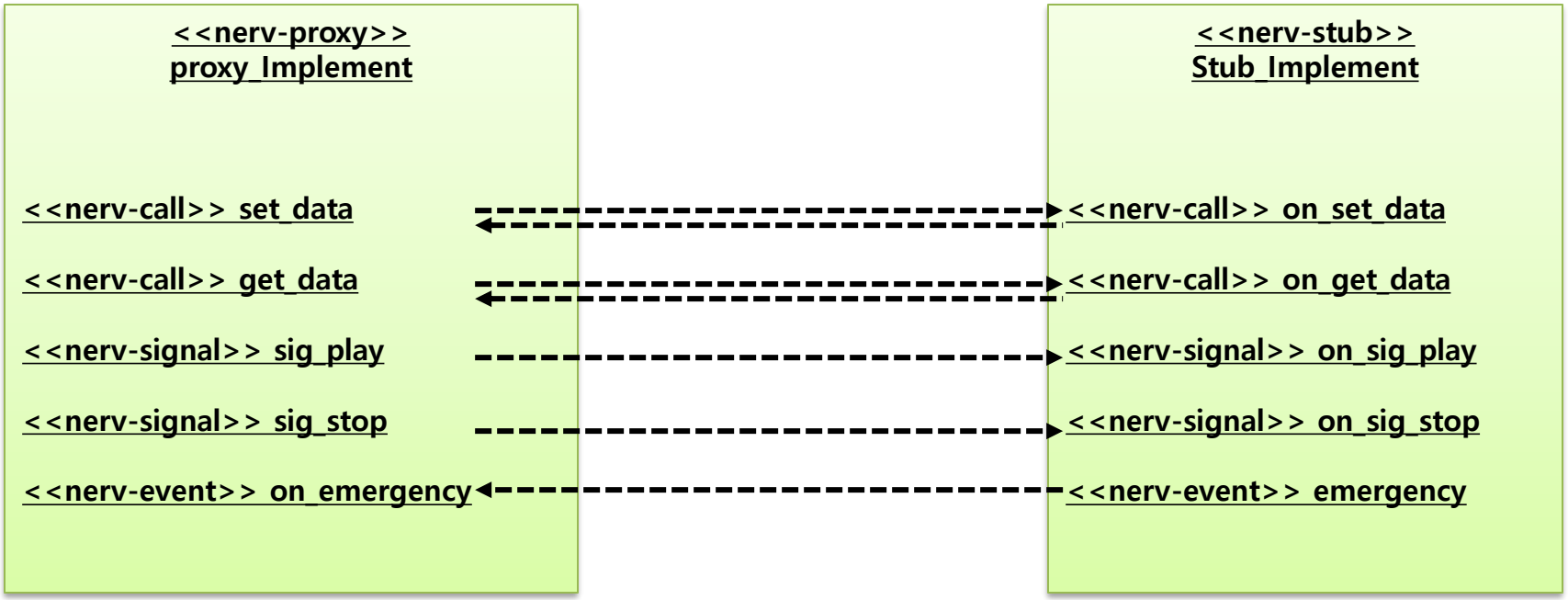
5-5. Proxy - Stub 객체 활용 예



5-5. Proxy - Stub 객체 활용 예



5-5. Proxy - Stub 객체 활용 예



6. 질의 응답

감사합니다.