Software Architecture

Distributed Architecture

Types of Distributed Architecture Style

- Client-server
- Multitier
- Proxy
- Dispatcher (Load Balancer)
- P2P
- Broker
- Service-oriented architecture
- Microservice architecture

Broker Architecture Style

Eunmi Choi
Kookmin University

RMI랑 비슷하다 느낄 수 있음.
RMI의 확장 형태
RMI - 안에 다 프록시로 연결

Broker가 하는 일이 더 많음. 클라이언트와 서버가 분리되는 형태

Broker Architecture Style

중개자.

- Synopsis

분산 컴퓨팅 환경에서 중간 과정이 있는 미들웨어 아키텍쳐. 서버들을 중재해줌. 서버와 클라이언트간에 요청에 따라 서비스를 제공.
서비스는 서버단에서 운영되는데 클라이어트가 원할때 제공해주는거.

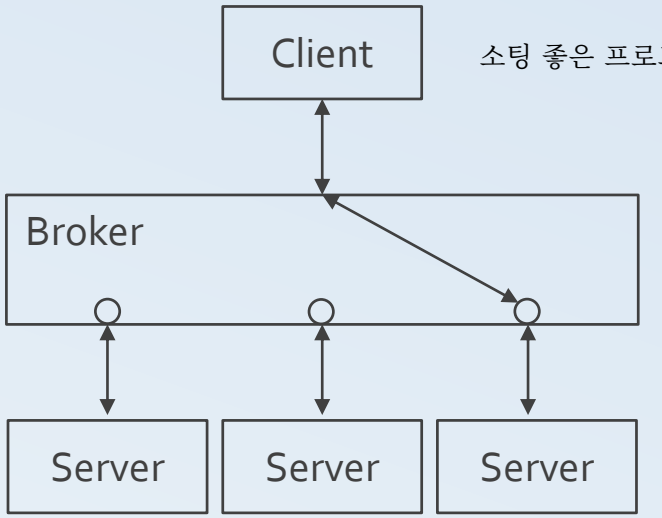
  - a middleware architecture used in distributed computing to coordinate and facilitate communication between registered servers and clients.

클라이언트가 리퀘스트를 보내는데 이 요청이 상황에 따라 브로커가 적절한 것을 찾아 제공을 해주는 것. 브로커는 중간에 중개자가 적절한 서비스를 찾아 서비스 해주는 형태.
브로커를 통해 원격 서버와 클라이언트가 디커플 되는 좋은 구조다.
전반적인 서비스에 대한 리퀘스트를 관장하고 중재해줌. 요청사항을 받아주고 클라이언트가 웒나ㅡㄴ 서비스를 하는 애에게 전달해주고 수행하고 그 결과값을 전달해줌. 서비스가 죽어
서비스가 앎된다면 그것도 에러로 줌..여러 익셉션도 전달해줄 수 있음.

  - The Broker architectural pattern can be used to structure distributed software systems with decoupled components that interact by remote service invocations.
  - A broker component is responsible for coordinating communication, such as forwarding requests, as well as for transmitting results and exceptions

Broker Architecture Style: Context

- A **distributed** and possibly **heterogeneous system** with **independent cooperating components**



소팅 좋은 프로그램을 쓰고 싶어

갖고 있는 서비스 중에 이 클라이언트에게 가장 맞는 서비스를 찾아 제공

이종 시스템을 연결해줄 수 있는 역할. 이 이종 시스템은 서로 독립적인 일을 수행함.(hetero genuous)

Broker Architecture Style

- Structure
  - A broker can be
    - either an invocation-oriented service to which clients send invocation requests for brokering,
    - or a document or message-oriented broker to which clients send a message (such as an XML document).
  - Better decoupling between clients and servers is one of the most important quality attributes
    - client and server components are decoupled through the use of a broker
  - A broker system is also called proxy-based system.

두가지 종류의 브로커가 있어~~~
인보케이션 관련 일, 메세지 관련 일을 처리해주는 거~
메소드를 부르듯이 , 객체를 인보케이션 해주듯,
이런 상태로 브로커가 인보케이션 하는 서비스를 찾아서 인보케이션을 해줌
메세지가 감. 메세지를 처리해주는 브로커. xml 다큐먼트(data description을 schema가 가지고 있음)를 제일 많이 씀.
주요한 일들을 처리해줌. quality attributes
브로커는내부 구조가 다 프로기로 이루어져 있음

Broker Architecture Style

- Structure
  - Servers make their services available to their clients by registering and publishing their interfaces with the broker.
  - Clients can request the services of servers from the broker statically or dynamically by lookup.
  - A broker component is responsible for coordinating communications.

서비스를 레지스트리 할 수 있도록 함. 인터페이스를 퍼블리쉬함.

서비스가 뭐가 있는지 테이블이 있음. 컴파일 시 아예 픽스된 형태도 있고 다이나믹한거도 있긔...
레이트 바인딩.. 9분 20초 ~40초

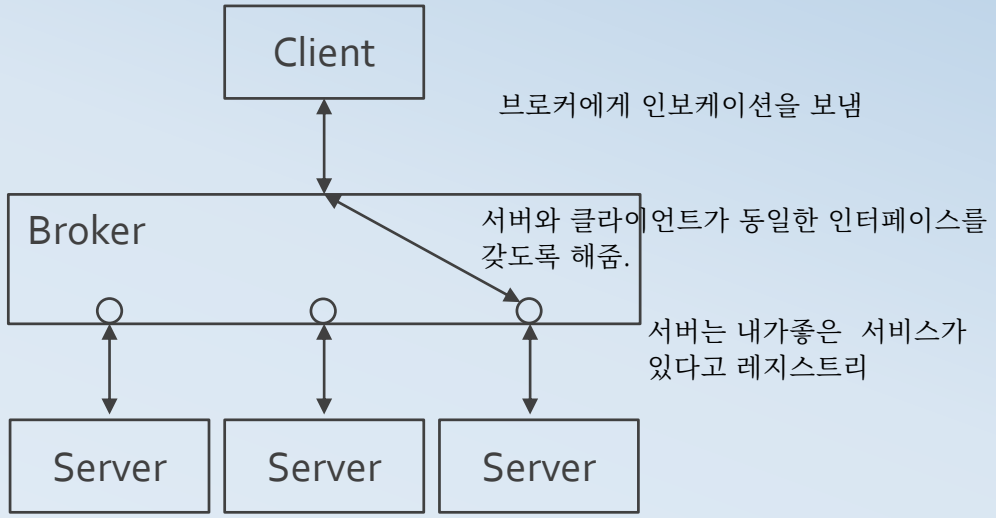
  - Ex) Common Object Request Broker Architecture (CORBA)

역할은 분산 오브젝트를 리퀘스트 하는 브로커.
외부의 분산오브젝을 요청할 수 있는 아키텍쳐
OO기반 서비스를 리퀘스트 할 수 있도록 함.
코바는 스펙구조임. 이걸 임플리멘테이션함.

요즘은 자바 라이브러리에 다 들어가 있음.

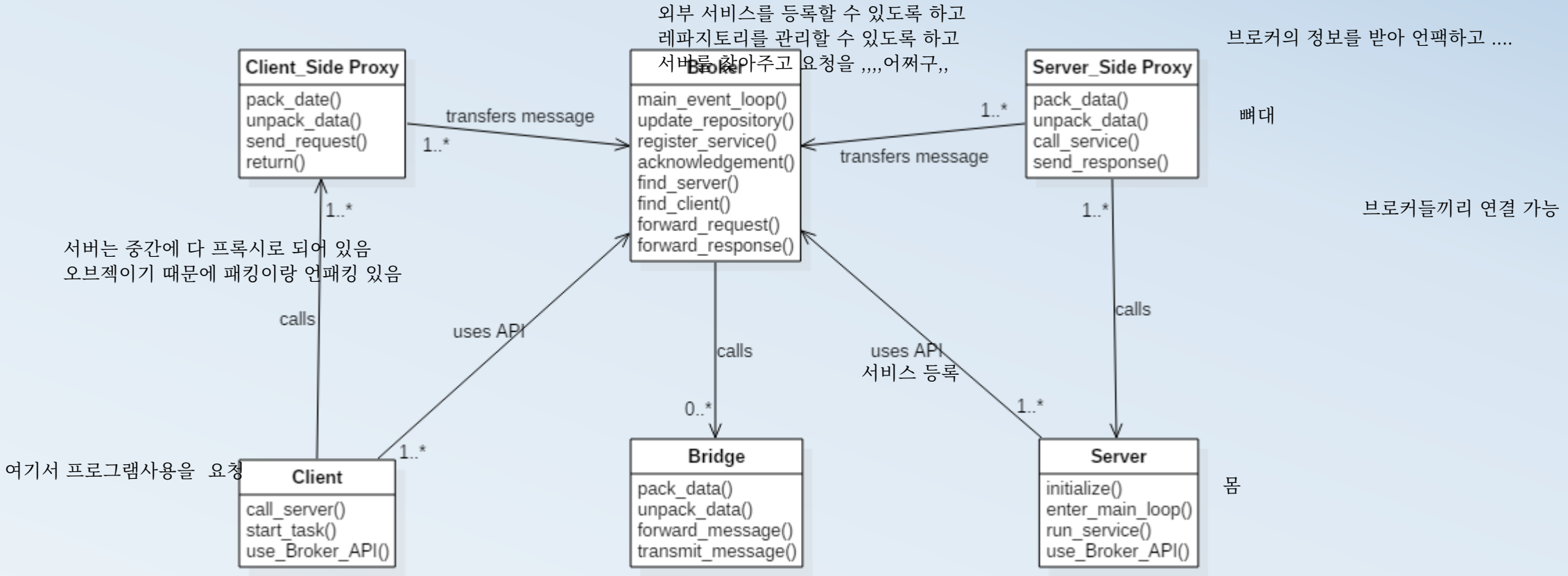
Broker: Solution - Structure

- Design broker component *to decouple clients from servers*
  - Servers
    - Register with broker
    - Present method interfaces to clients
  - Clients
    - Access server's methods via broker
    - Uses same form to call server's methods
  - Broker's Tasks
    - Locating appropriate server
    - Forwarding requests to server
    - Transmitting results and exceptions to client



Broker: Solution 1 - Static

- Six types of participating components: *clients*, *servers*, *brokers*, *bridges*, *client-side proxies* and *server-side proxies*



Broker Architecture Style

읽으심

- Decompose the broker architecture into subcomponents:
  - Broker : coordinates communications, passing on requests and returning replies.
    - The broker stares all servers' registration information, including their functionality and services as well as location information.
    - The broker provides APIs for clients to request, servers to respond, registering or unregistering server components, transferring messages, and locating servers.

나무 밑둥머리. 클라이언트가 있는 것처럼 사용. 이걸 통해 외부 서비스를 요청할 수 있음.

  - Stub (client-side proxy): mediates between the client and the broker and provides additional transparency between them.
    - To the client, a remote object appears like a local one.
    - The proxy hides the interprocess communication at protocol level, marshals parameter values, and unmarshals results from the server.
    - The stub is generated at the static compilation time and deployed to the client-side to be used as a proxy for the client.

마셜 : 팩을 해서 시리얼라이제이션 하는것
작은 상태로 함축해 씀.

Broker Architecture Style

읽으심. 리퀘스트를 받아 언팩해서 내용이 뭔지 언마셜 하고 적절한 것을 보내줌

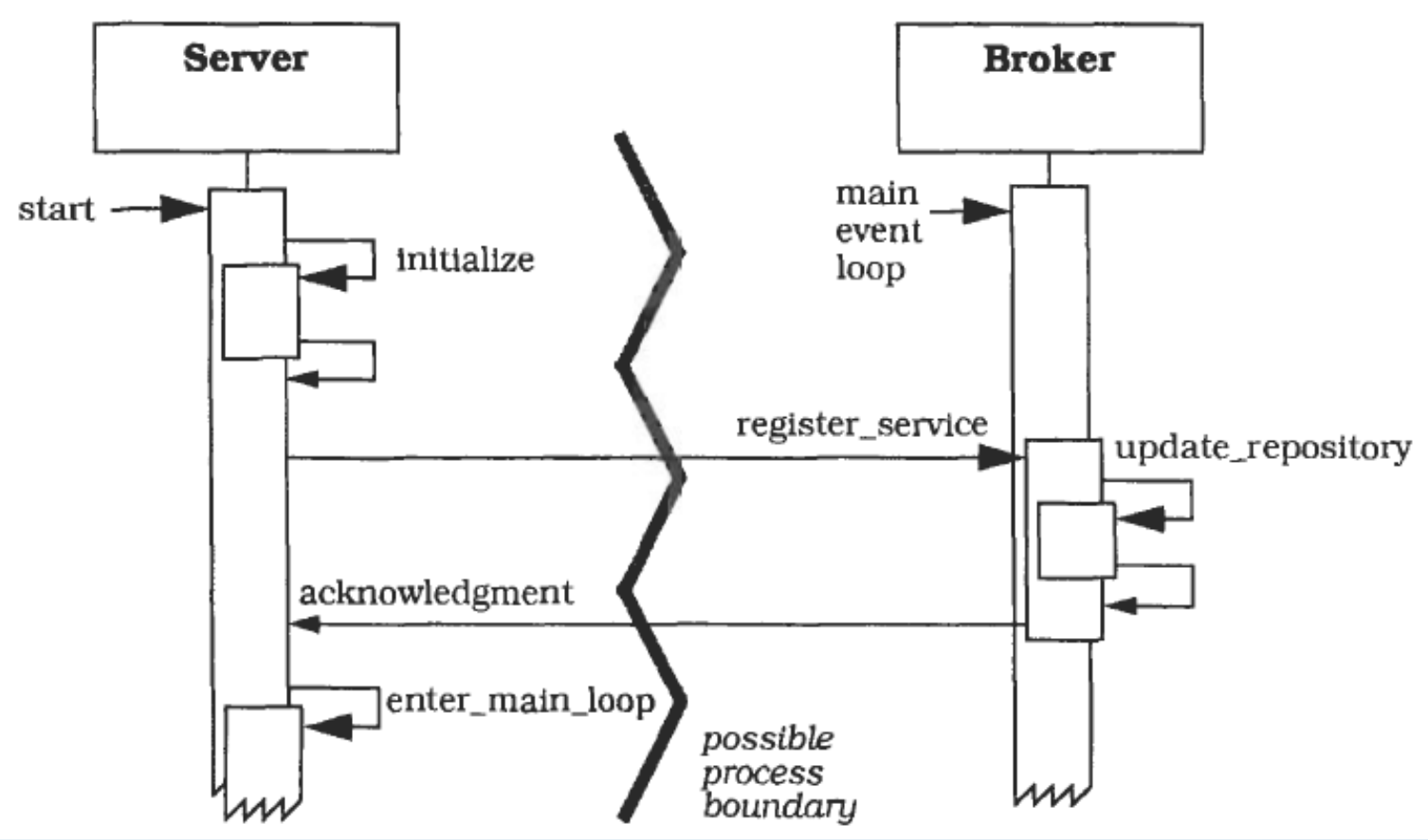
- Skeleton (server-side proxy): is also statically generated by the service interface compilation and then deployed to the server-side.
  - It encapsulates low-level system-specific networking functions like the client-proxy and provides high-level APIs to mediate between the server and the broker.
  - It receives and unpacks the requests, unmarshals the method arguments, and calls the appropriate service.
  - When it receives the result back from the server it also marshals the result before sending it back to the client.

  브로컹와 브로커를 연결
- Bridges : are optional components used to hide implementation details when two brokers interoperate.
  - Bridges encapsulate underlying network detail implementation and mediate different brokers such as DCOM, .NET Remote, and Java CORBA brokers.
  - They can translate requests and parameters from one format to another.
  - A bridge can connect two different networks based on different communication protocols.
- Network : connects components using designated protocol standards such as TCP/IP, OIIP, or SOAP.
  - The request carries data in a message document or method invocation format.

Broker: Solution 1 - Dynamics

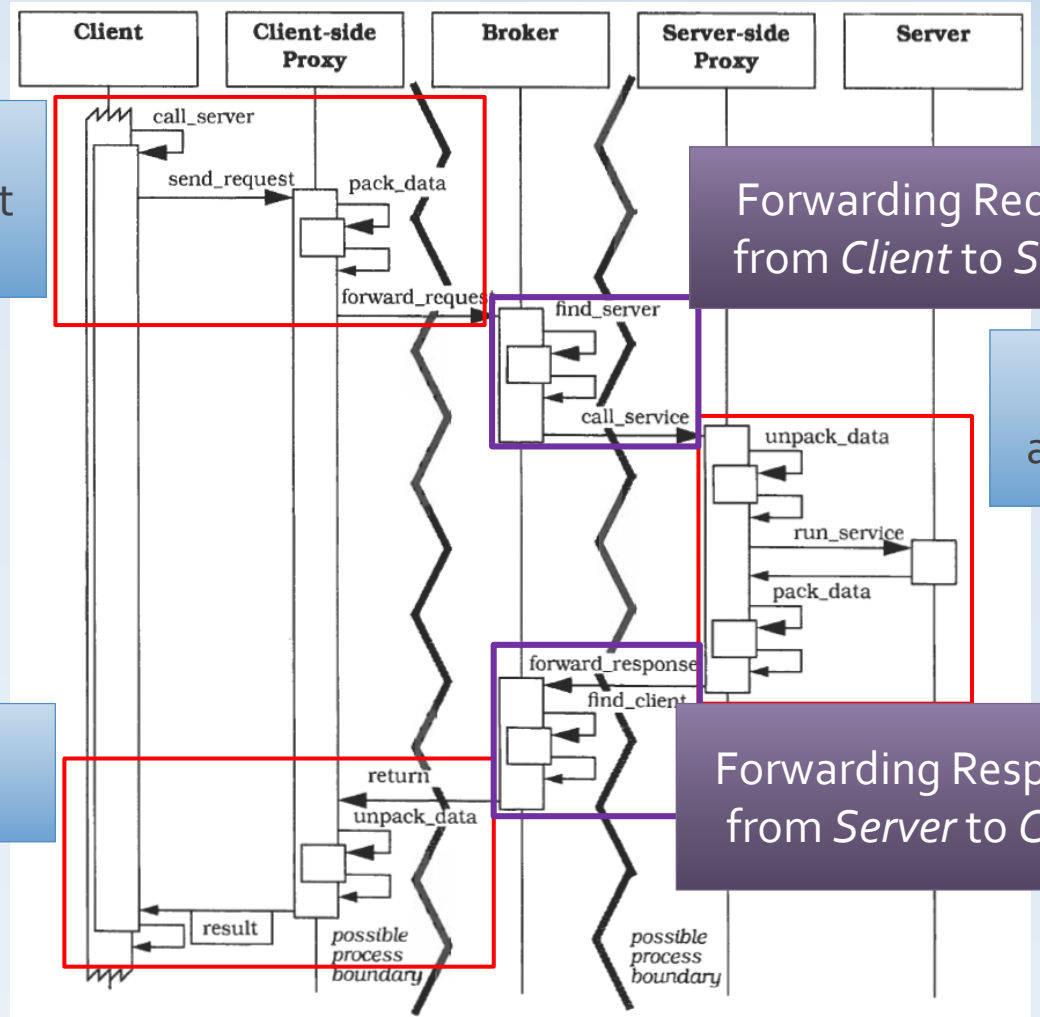
봄

- **Scenario 1** : Server Registration



Broker: Solution 1 - Dynamics

- **Scenario 2** : Service Process



Sending request

브로커에게

Forwarding Request from *Client* to *Server*

리퀘스트가 뭔지 받아서 서버에게 보냄

Receiving Request and Doing Response

받아서 서비스를 하도록 하고

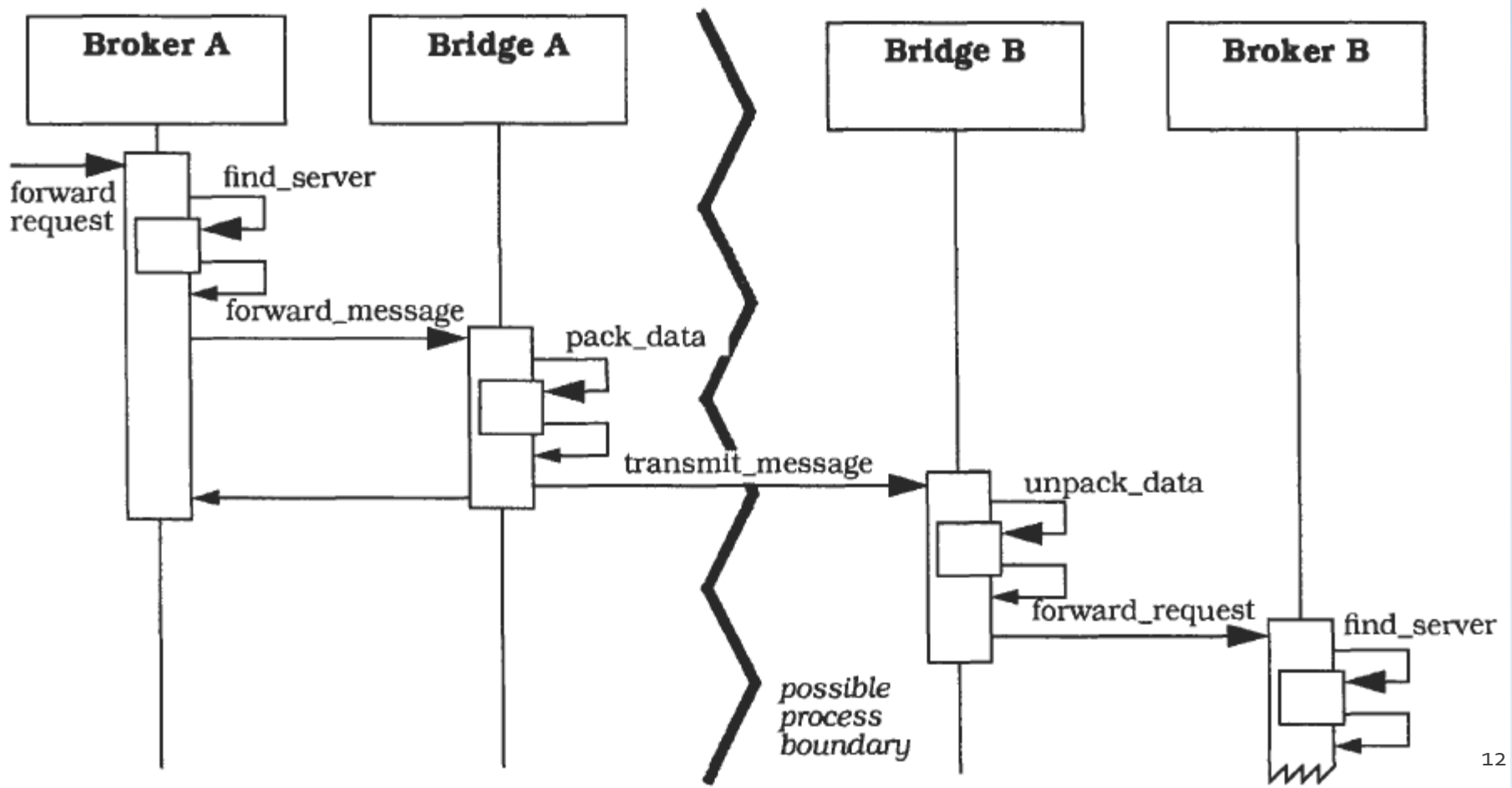
Getting Response

Forwarding Response from *Server* to *Client*

result를 받아서 보냄

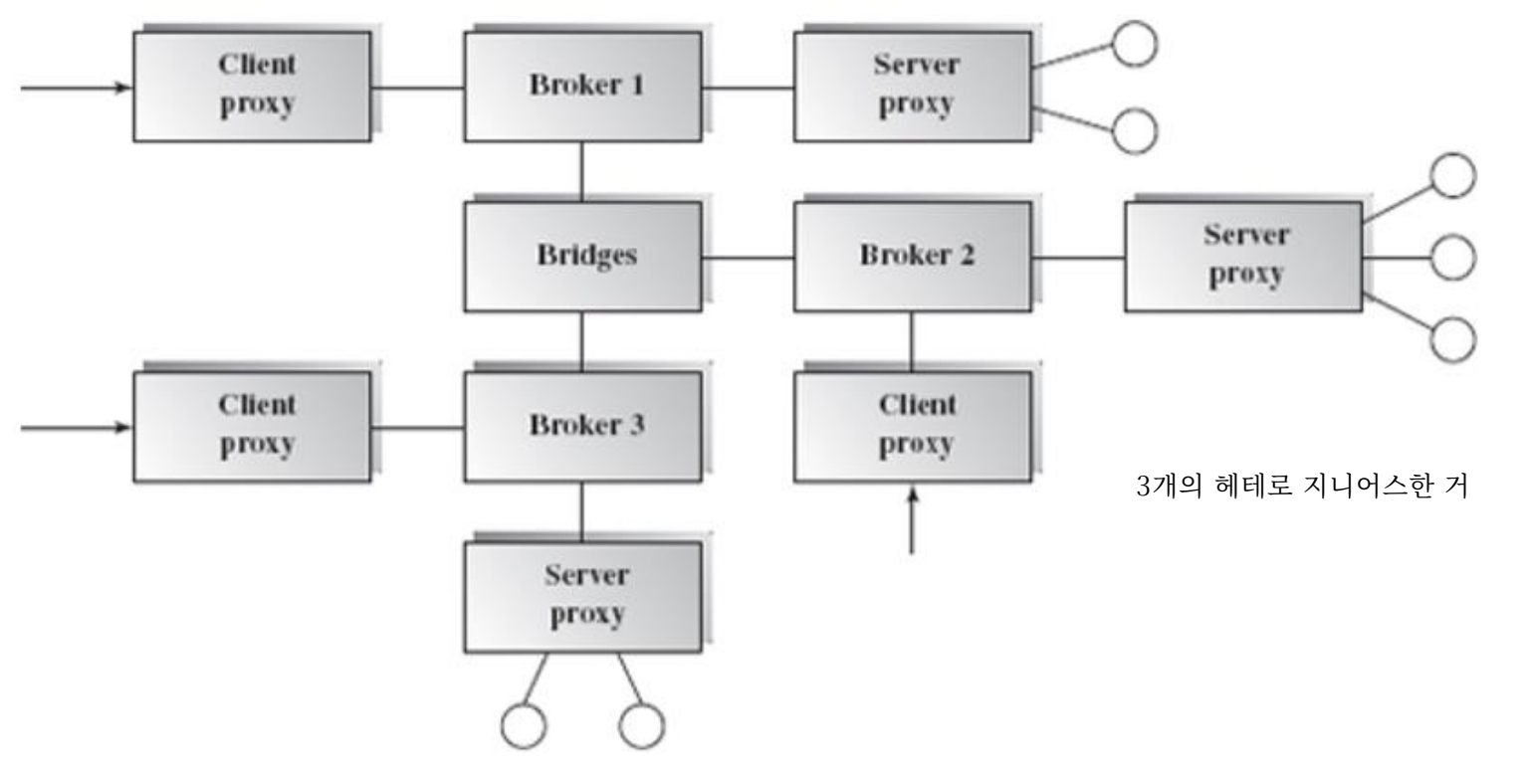
Broker: Solution 1 - Dynamics

- **Scenario 3** : Service Process via Bridge



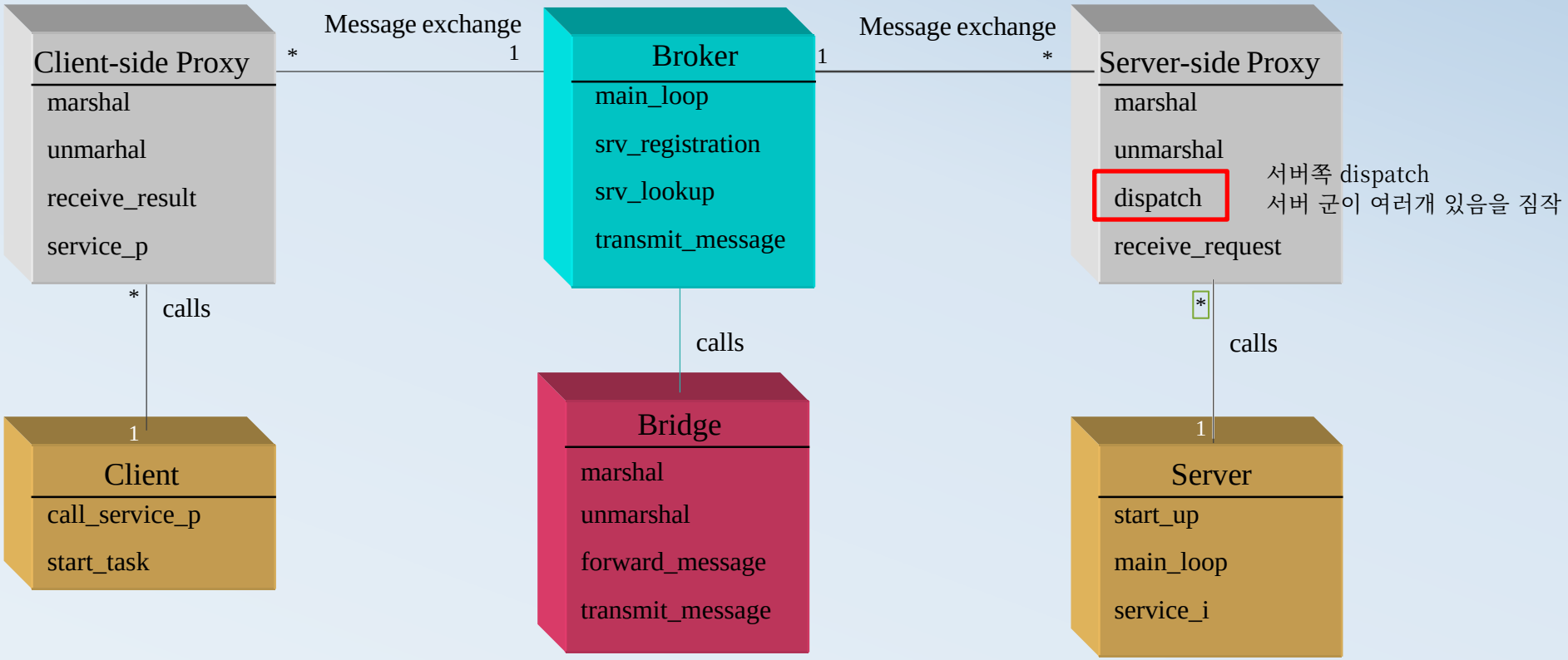
12

Broker: Solution 1 - Overall

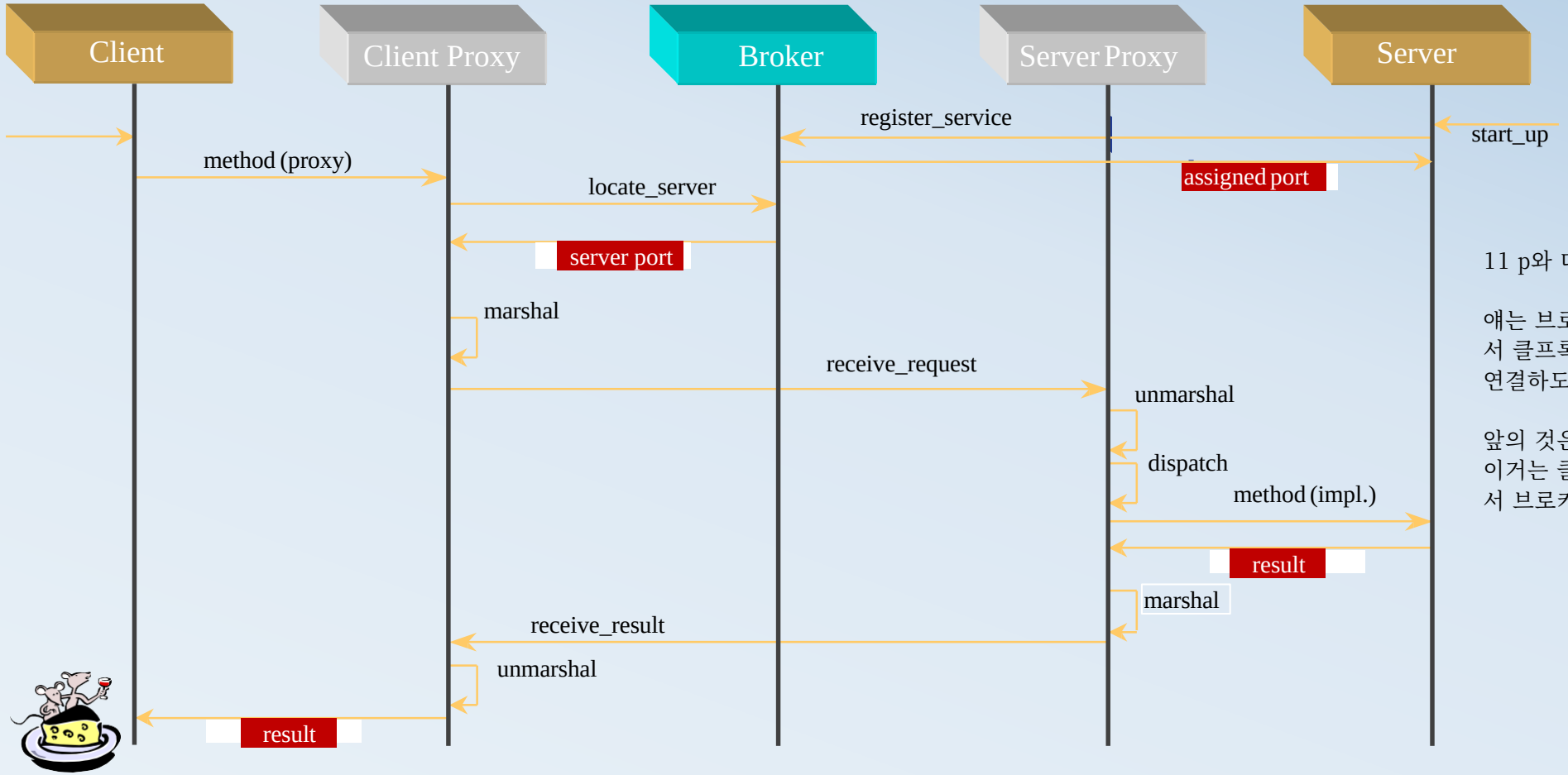


3개의 헤테로 지니어스한 거

Broker: Solution 2 – Static (with Dispatcher)



Broker: Solution 2 - Dynamics



11 p와 다른점?

얘는 브로커가 위치정보를 알려줘서 클프록시가 서버프록시를 직접 연결하도록 해준다.

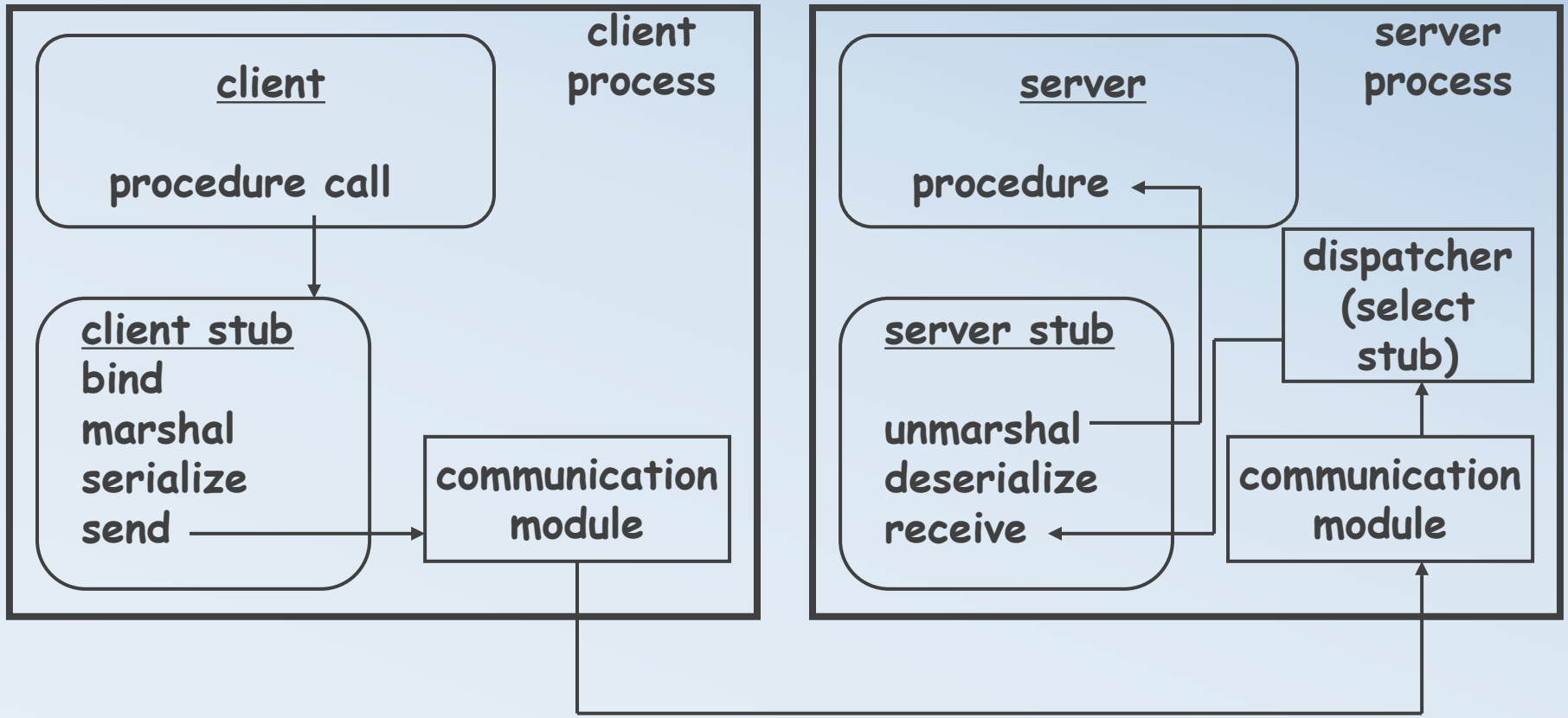
앞의 것은 세션을 유지하는걸 잘 함
이거는 클 - 서버 연결점이 강해져서 브로커의 로드를 줄여줌.

~25:25

외부에 있는 behavior를 리모트하게 인보크 하겠다 라는 것.

Broker : Example 1 – Remote Procedure Call

RMI의 전신.



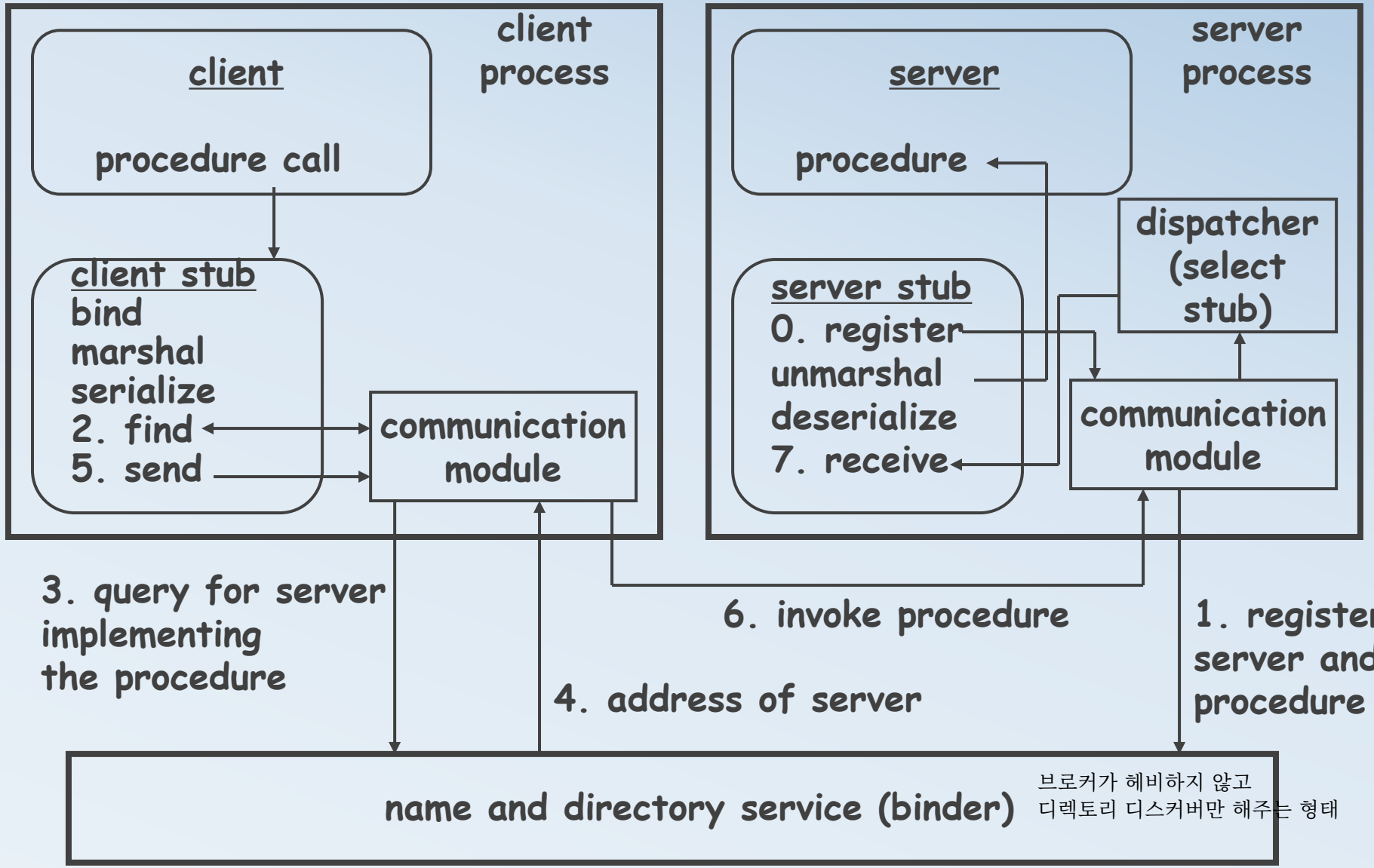
계속적으로 있었던 테크놀로지

~27:18

Basic functioning of RPC

Dynamic Binding in RPC Architecture

execution time



두가지 케이스 중 두번째
클 프록시와 섭 프록시가
서로 연결

브로커가 헤비하지 않고
디렉토리 디스커버만 해주는 형태

Dynamic binding

Broker : Example 2 – CORBA (Common Object Request Broker Architecture)

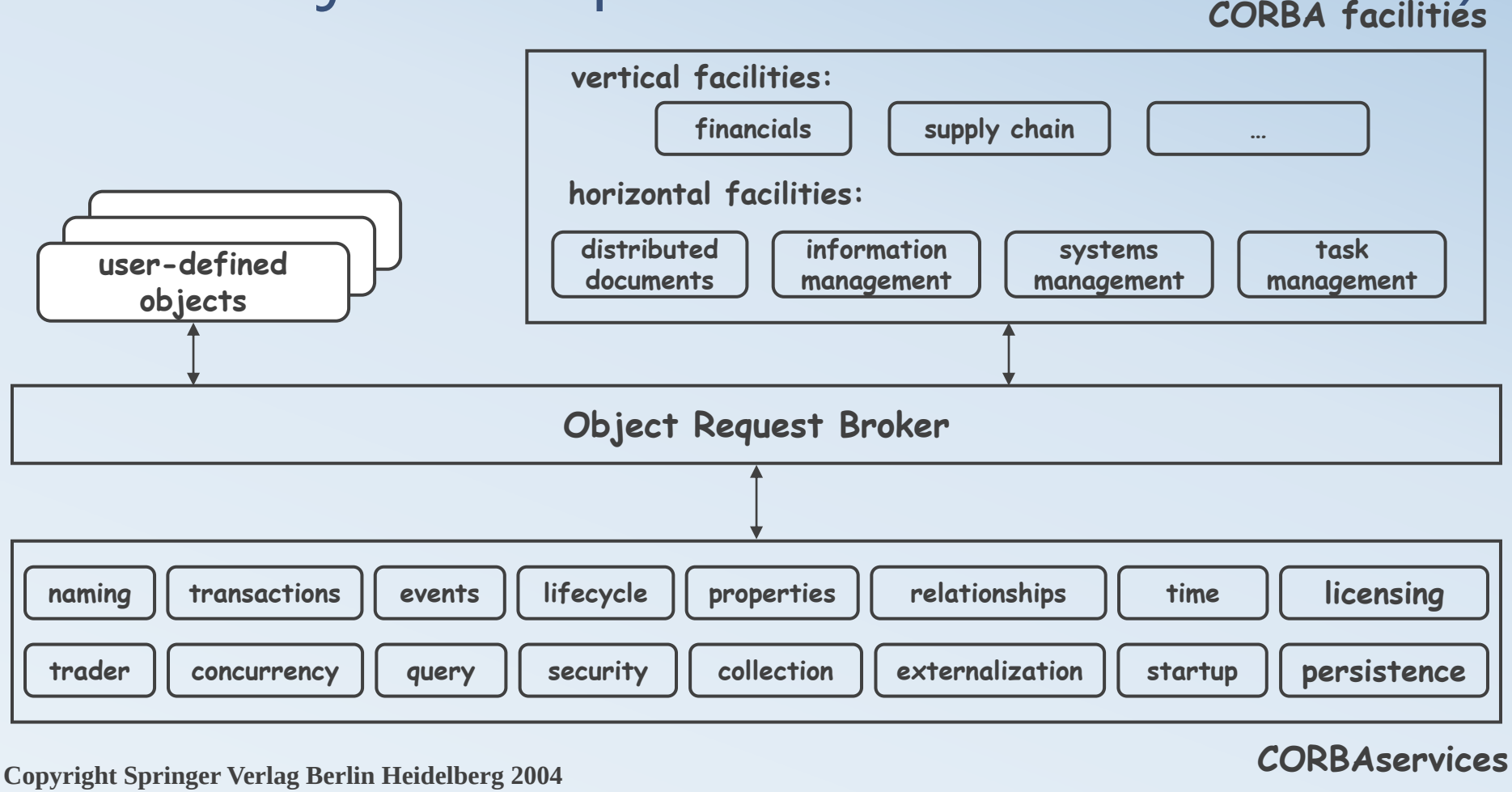
~29:

~30

- CORBA is an open standard for distributed, remote-method invocation computing schemes.
  - It provides platform and language-independent middleware to integrate distributed applications, including legacy code.
  - The services and operations that a CORBA object provides are specified in interfaces using the Interface Definition Language (IDL) so that clients can make requests without knowing the detailed implementation and location of the CORBA objects.

Broker : Example 2 – CORBA (Common Object Request Broker Architecture)

코바 안에 라이브러리가 있음
요청을 보내면 세션을 그때부터 브로커가 가지고 있음
~31:30



~33:

~34:22

Broker: Advantages & Disadvantages

- Advantages:
  - Server component implementation and location transparency
  - Changeability and extensibility
  - Simplicity for clients to access server and server portability
  - Interoperability via broker bridges
  - Reusability
  - Feasibility of runtime changes of server components (add or remove server components on the fly) 필요에 따라 레지스트리하고 위치를 바꾸므로
- Disadvantages:
  - Inefficiency due to the overhead of proxies 프록시를 너무 많이 씀
  - Low fault-tolerance 응답이 없으면 브로커가 문젠지 서버에서 서비스를 문제가 생긴건지, 간접적으로 계속 거쳐 서비스를 하기 때문에 컨트롤이 잘 안됨....
  - Difficulty in testing due to the amount of proxies 운영과 테스팅 추적이 어려움..

Software Architecture

Distributed Architecture

Types of Distributed Architecture Style

- Client-server
- Multitier
- Proxy
- Dispatcher (Load Balancer)
- P2P
- Broker
- Service-oriented architecture
- Microservice architecture

Message Broker Architecture Style

Eunmi Choi
Kookmin University

Message Broker Architecture

메세지 브로커는 맘에다 브로커 기능을 +알파 해준 것
버퍼 베이스한 메세지 오리엔티드 하게 인보케이션 하는 아키텍쳐

- A message is a packaged or formatted information
  - such as an XML document sent and received between applications.
- A message broker is a Message-Oriented Middleware (MOM) server in a message-oriented distributed system.
  - It performs message routing, message transformation, message invocation, security checking, and exception handling.

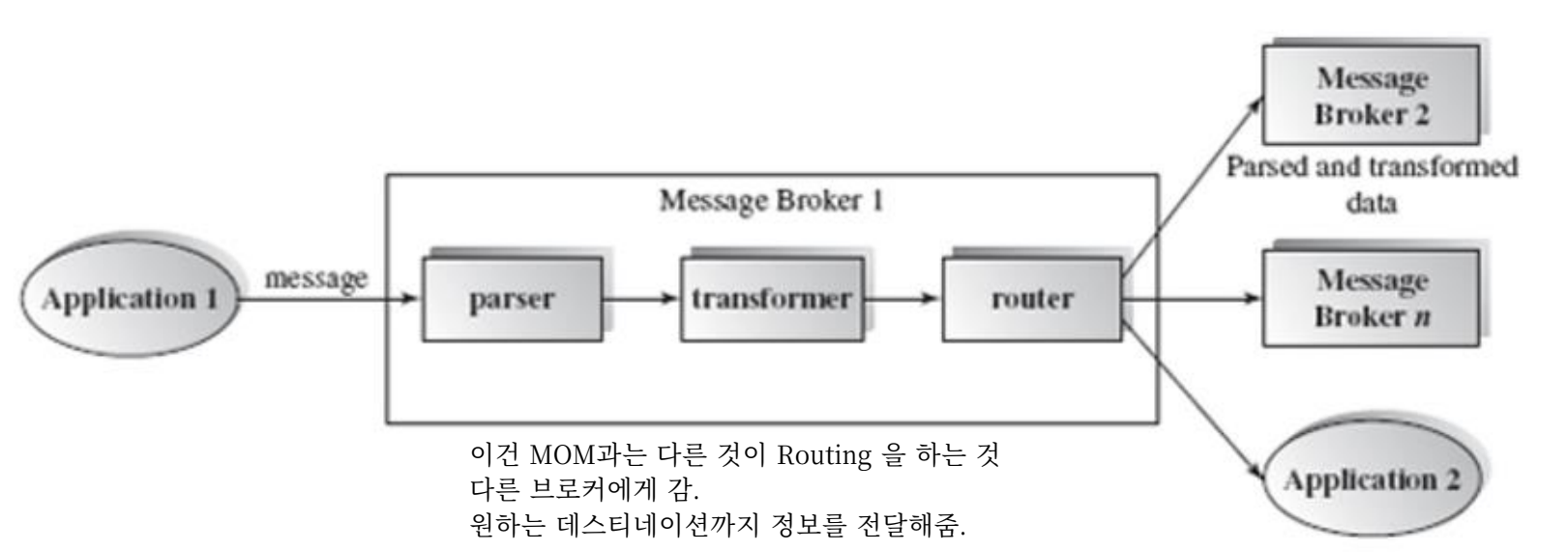
이건 중간에 버퍼를 주는 것.
요청을 이 버퍼에 두고 갈수 있게 하는 것

~36:20

Message Broker Architecture

- A message broker must be able to route messages from a sender to a receiver based on the content of the message.
  - A broker can transform a message from one format to another to meet the different requirements of the sender and the recipient.
  - A broker may also transform a message by modifying, inserting, merging, or removing some data elements in the message.

Message Broker Architecture



이건 MOM과는 다른 것이 Routing 을 하는 것
다른 브로커에게 감.
원하는 데스티네이션까지 정보를 전달해줌.

Message Broker Architecture

메세지를 받을 때는 MOM과 동일하게
서브스크라이빙

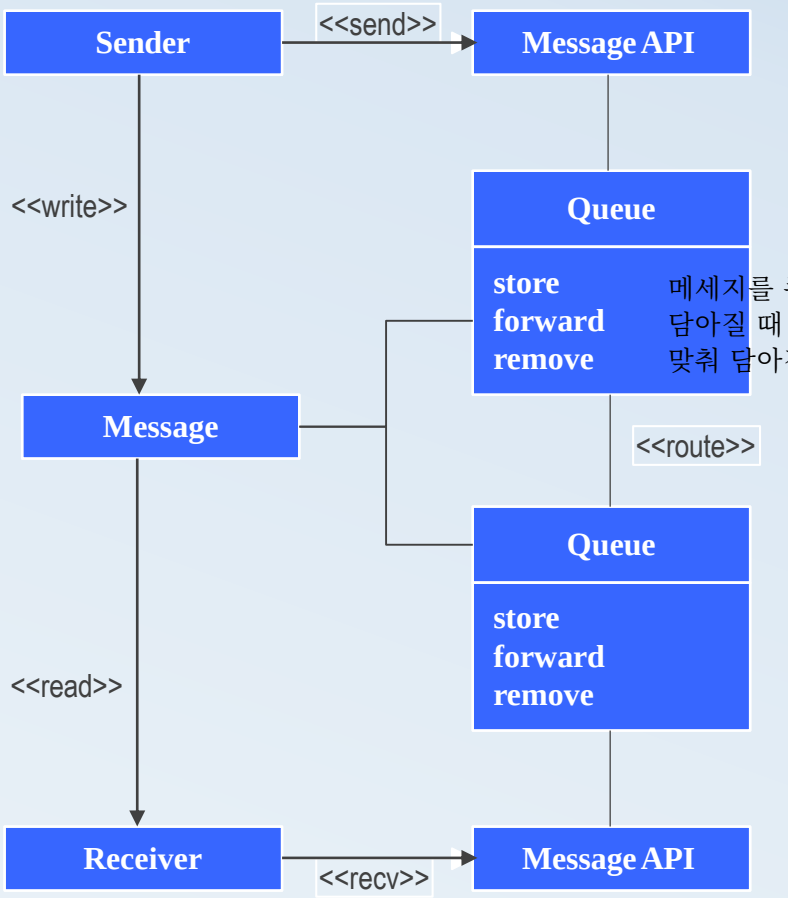
- Handle message distribution by publishing and subscribing.
- An application can publish on topics, and many applications can subscribe them on the broker.
- commercial message broker systems
  - WebSphere message broker, IBM MQSeries, Sun JMS, etc.

얘네가 메세지 브로커를 담당함

버퍼를 가진
메세지 큐 시리즈~

- ex) email processing

Solution



- Introduce intermediary queues between senders and receivers :
  - *Queues* are used to store messages persistently. They cooperate with other queues for message routing.
  - *Messages* are objects send from a sender to a receiver.
  - A sender sends messages, while a *receiver* receives them.
  - A *Message API* is provided for senders and receivers to send/recv messages.

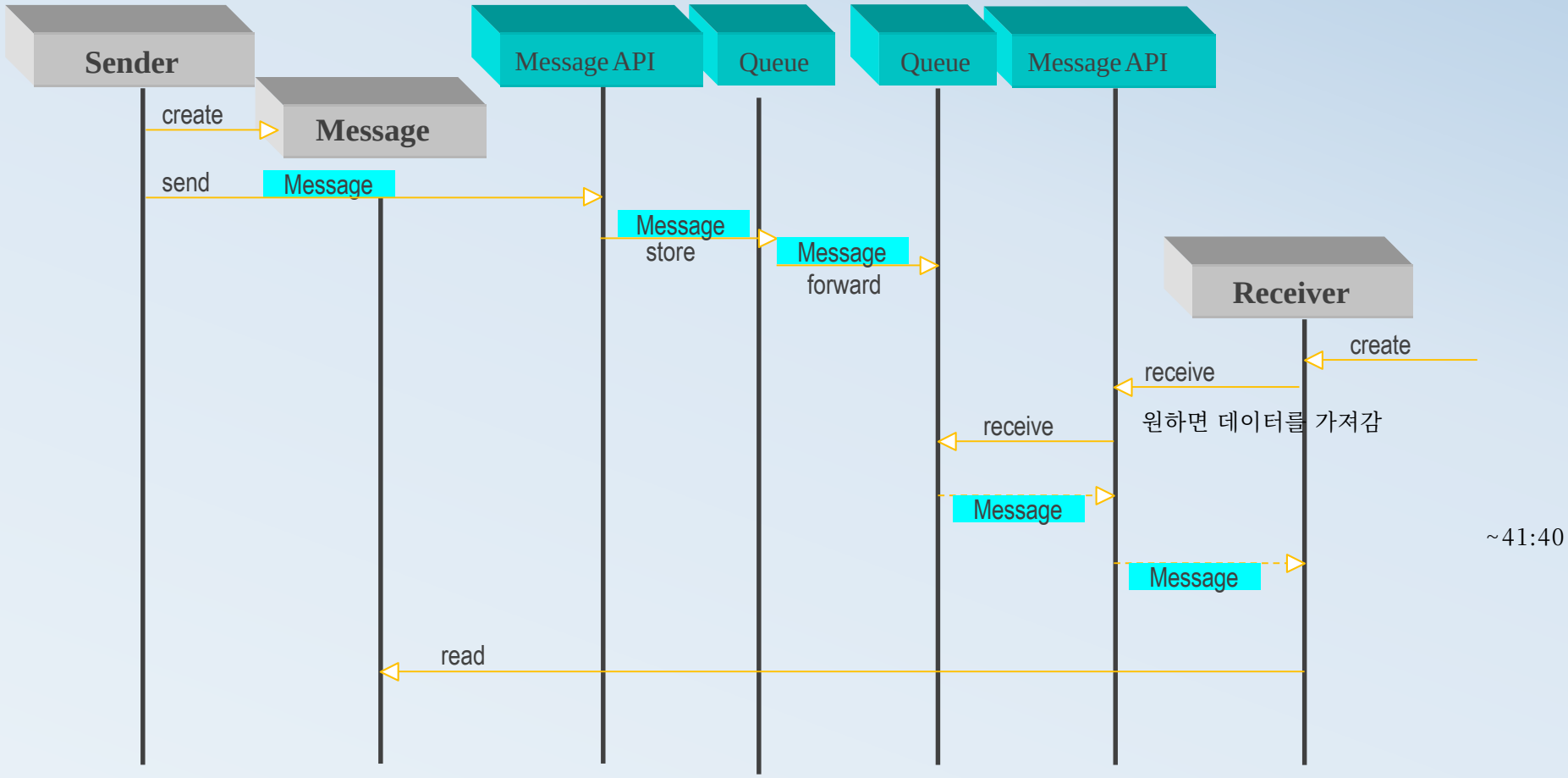
메세지를 큐에 담음
담아질 때 메세지 포맷에 맞춰 담아짐

큐는 퍼시스턴스한.
인티그레이티드를 유지함.

메세지는 큐 안의 오브젝트

~40:20

Dynamics



~41:40

Message Broker: Advantages & Disadvantages

- Advantages:
  - Reusability and maintainability: Loose coupling between the client and server component leads to easy maintenance and extension on both sides.
  - Flexibility : Invocation-oriented or document-oriented messaging; message heading and body can be altered for specific purposes.
- Disadvantages:
  - Overhead, indirection complexity, and difficulty in debugging and testing due to the new protocol stack added

중간에 MOM으로연결이 다 되어 있음(프록시 x)
계속적으로 다른 큐를 붙여나가니 간접적 레벨이 늘어남.. 오버헤드가 계속 일어남.
디버깅,테스팅이 힘듦.

RMI는 로컬 세그먼트에서하는게 좋음. 세션을 물고 있기 때문에. 서버단 구성시 이거가 좋음

밖으로 나갈 때는 브로커가 좋음..! WAN 구조로 갈 때. 큰 덩치의 서비스를 다닐 때 좋음.