Object Request Broker

ORB

- ORB
  - provide communication and management services
  - enables objects to communicate over networks
    - request and receive responses
  - based on RPC and message queue

  - Two category
    - OMG: CORBA
    - Microsoft: DCOM (Distributed Component Object Model)

Services of ORB

- Basic services
  - hide network details from the object
    - providing network transparency
  - handle object request and the synchronization of the request and delivery of any response
  - handle the exception to the requesting object

- Extra services
  - provide administration and development services
    - to create and terminate objects
    - to debug and test an application
    - a form of directory service
  - provide extra run-time services
    - multi-threading, security, load balancing, and transaction recovery

What are ORBs? -1

- is middleware
- enables objects to communicate over networks with different communications protocols
- enables objects to reside on different hardware platforms
- enable inter-communication between objects
- enable objects to be moved about the network transparently
- provide mechanism by which objects make request and receive response
- responsible for managing and locating the objects
- support both inter-object communication and communication between objects and external services
  - these should be transparent to the application programmer

What are ORBs? -2

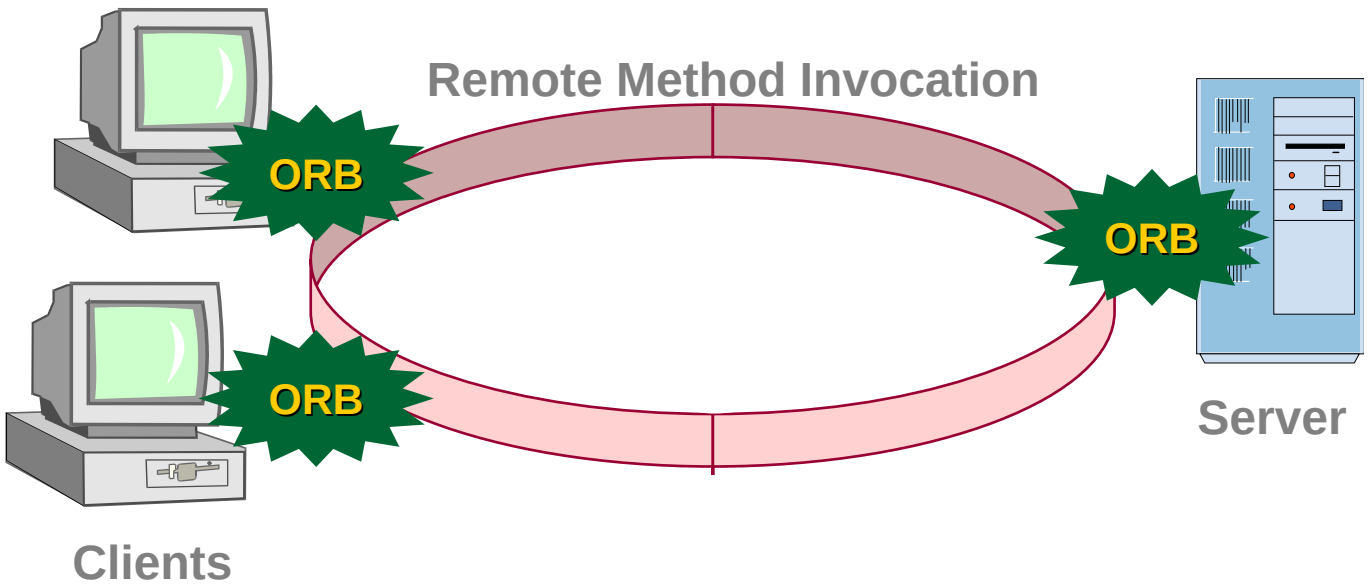
- objects interact by exchanging request and reply, either locally or over a network (About objects : application, process, object class, instance of class)
- act as the intermediary between objects
- locating the object on the network and communicating the request to object
- provide basic communication services
  - hiding network details from the object and providing network transparency
- handle object request
  - handle the synchronization of the request
  - delivery of any response or exception to the requesting object

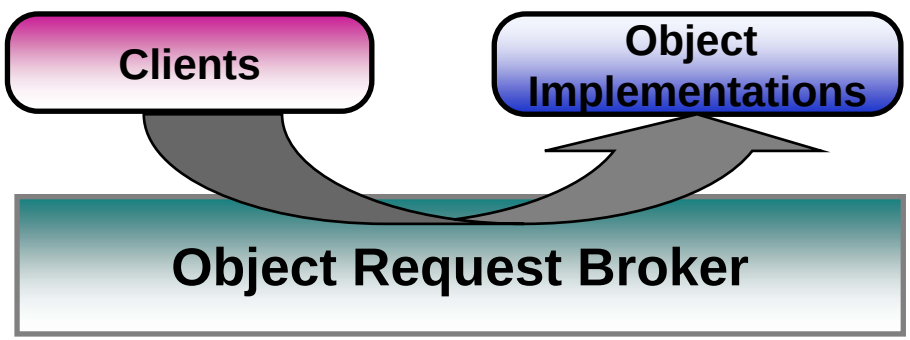
What are ORBs? -3

- provide extra services (run-time services)
  - provide administrative and development services
    - administrator to create and terminate objects, developer to debug and test an application
- services provided by ORBs differ between product implementation
  - multi-threading, security, a form of directory service, support for features such as load balancing and transaction recovery, location brokering

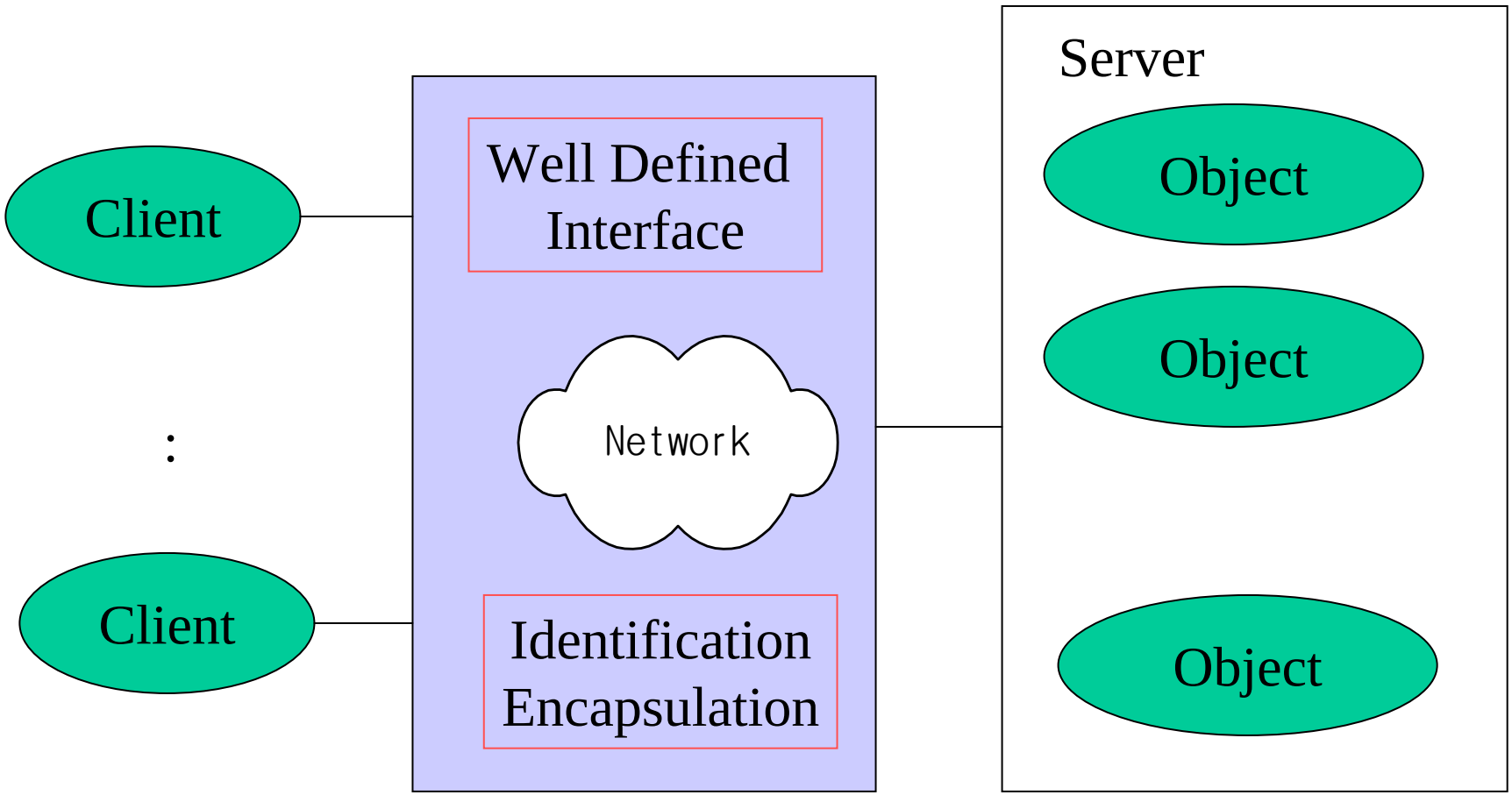
- is base of a communication layer provided by a low-level communication protocol, remote procedure calls or message-oriented middleware
- influence the style of communication supported
  - synchronous or asynchronous messaging

Distributed environment with ORB

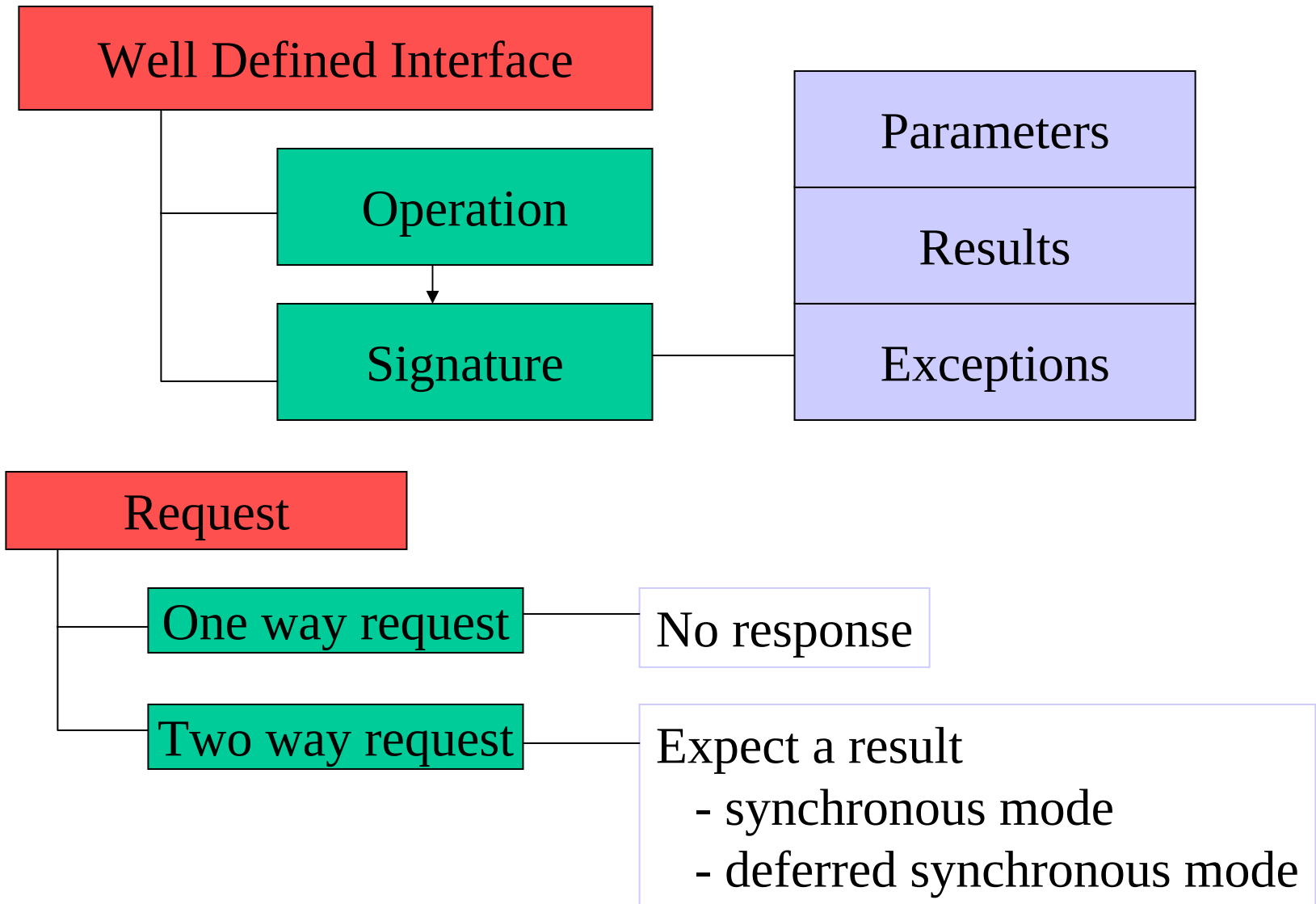




Basic Concept of ORB (1)



Basic Concept of ORB (2)



How ORBs work? -1

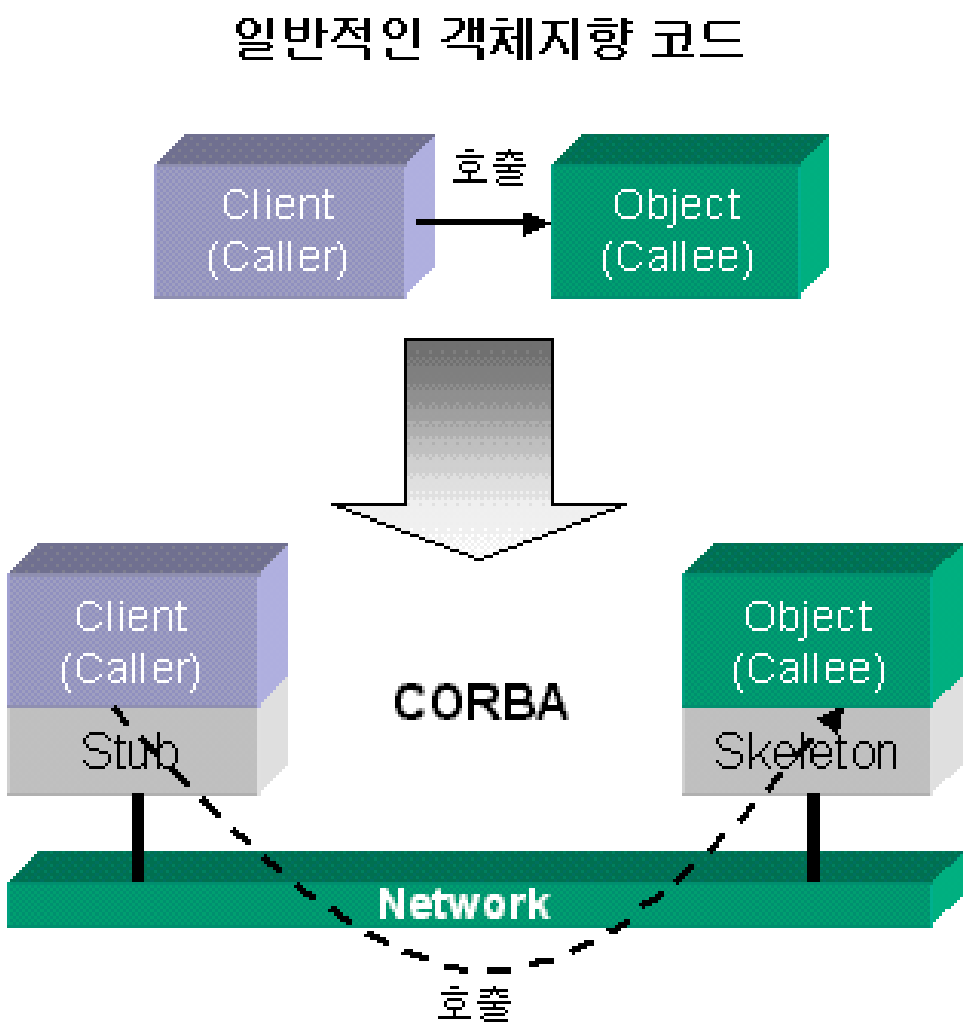
- based on the idea that applications are written around collections of objects that isolate the requester from provider (of the service) by a well defined interface

- interface to an object is fundamental to the way ORBs work
  - describes all the possible operations that a client may request of an object
  - is independent of where the object is located, what programming language it is defined, how its services are implemented
- each operation specified within the interface has a signature that describes the legitimate values of request parameters and returned results
  - parameters required, results, exceptions, additional contextual information

How ORBs work? -2

- Based on “classical object model”
  - where the client sends a message to the object and the object then interprets the message to decide what service to perform
  - at run-time, client issue messaging in the form of request (expect a result, require no response)
  - one-way request
    - has no need to synchronise with the target object’s execution
  - synchronous mode
    - client pause to wait for completion of the request
  - deferred synchronous mode
    - client does not wait for completion of the request, but does intend to accept results later

Distributed environment with ORB



ORBs?

- Static ORB
  - a similar way to RPC
  - an interface is decided at compile time
  - synchronous requesting
- Dynamic ORB
  - an interface is decided at run-time
  - deferred synchronous requesting

- ORB functions
  - finding the code and data that implement the object
    - implementation repository
  - communicating the data making up the request
  - invoking the object that is to receive the request
    - dynamic service activation
  - receiving from failure or error

Static ORBs

- the interface and function calls have been defined, the objects are compiled
  - similar way to RPC (restriction : request must be synchronous)
  - creating new threads at the client end
  - (in principle) object has to issue a message and wait for the reply
  - application which use static invocation must have all their interface definitions available at compile time
  - once executables are produced, the application structure is fixed
- less flexible to network changes
  - if objects are changed, the source code has to be altered and recompiled
- faster message passing
- simpler to program
- improve code quality by allowing compile-time type checking

Dynamic ORBs -1

- do not require the programmer to specify the interface to be used in the compiled code
- instead, code is written by the programmer that uses a separate specification of the interface, not held in the program

- determine dynamically what interface are available (reference to some file : interface repository - in CORBA)
  - provide a list of the interface available
  - arguments, parameters
  - enabling decisions to be made as late as possible

- manipulate the lists and puts them into standard form before routing them to the appropriate object implementation

Dynamic ORBs -2

- support deferred synchronous invocation
  - control returns to the client after the request has been made, and the client is free to continue executing code
  - after specific period, client may poll the ORB to retrieve the result
  - support multiple request to many objects
- more flexible to change
  - no need to recompile when change occurs to the interfaces
  - object has access to any object on the system and to all the interface
- require more code and effort to program and are likely to be slower than the approach used with a static ORB
  - errors being made in the code is increased as type checking has to be done at run time

ORB functions -1

- When client issues a request, ORB is responsible for
  - finding the code and data that implement the object
  - communicating the data making up the request
  - invoking the object that is to receive the request
  - recovering from failure or error

ORB functions -2

- **Locating the object**
  - may use an implementation repository
  - contains information about object
    - location
    - description of the data structure used to hold state information
    - operation and interface used by the object
    - (interface repository : detailed info about the interfaces)
  - approach used to create information about the objects depends on the standard to which the ORB conforms
  - most implementation-repository information is created at object-installation time, by the ORB administration s/w, by the object itself, by the administrator

ORB functions -3

- **Communicating the data**
  - assumed function of ORBs
    - conversion of the data required when communication is between heterogeneous machines
    - majority of ORBs provide this service, some do not
    - programmer will have to the conversion required if the communication services are weak
    - transport protocol is hidden from the programmer
    - ORB handles network protocols (ORB may support TCP/IP)

ORB functions -4

- **Invoking the object**
  - provide mechanism for object activation which allows them to be invoked only when they are required
  - when object is requested, ORB will create an instance of that object which is then used to process the request
    - subroutine is held in a library, and a copy of it is created in core ready for processing only when it is requested
  - ORB's responsibility
    - find the object
    - create object instances when they are needed
    - remove object instances when they are no longer

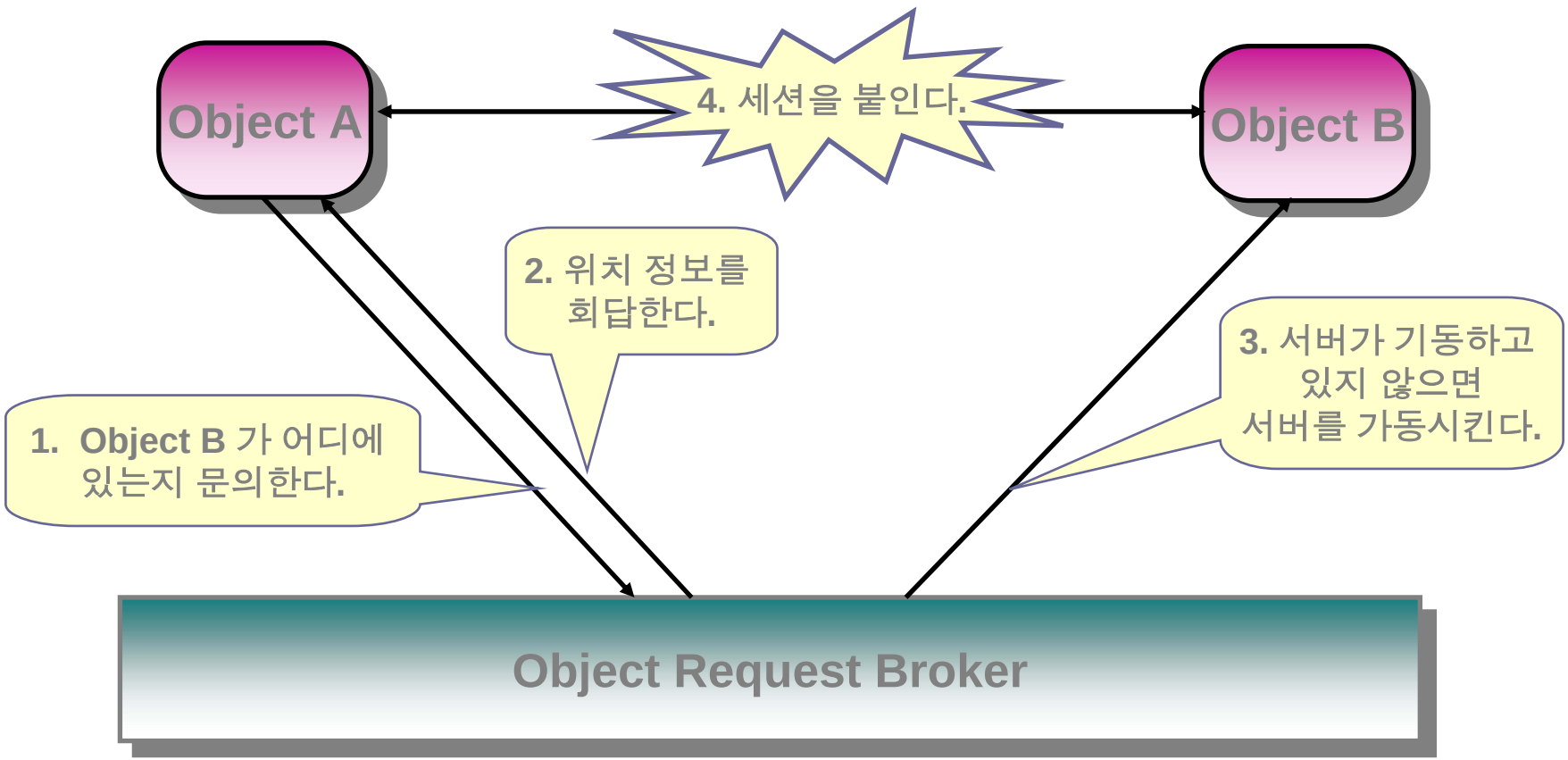
ORB functions -5

- **Invoking the object (cont.)**
  - dynamic service activation
    - when object invoked, it clones itself and the instance is loaded into the machine
    - once the object has completed its task, it may then be deactivated by the same service
  - the object interprets the message by selecting a method based on the operation requested (method is the code used to perform the service)
    - input parameters and data passed in the message are passed to the method so that it can use them to execute the request
    - request then results in the service being performed by the object on behalf of the client
    - data may be created or updated, results produced, and the internal state of the object changed.

ORB functions -6

- **Coping with failure**
  - ORBs do not necessary provide services to manage transaction failure
    - (these may have to be handled by application program)

ORB 호출 방법


Object A
4. 세션을 붙인다.
Object B
2. 위치 정보를
회답한다.
3. 서버가 기동하고
있지 않으면
서버를 가동시킨다.
1. Object B 가 어디에
있는지 문의한다.
Object Request Broker

Ways of implementing ORBs

- generally defined by standards and so depend on the particular product implementation
  - location of the ORB s/w
  - language it is written in
  - protocol used for communicating

- ORB can reside on both the client and server
  - with location services in ORB to establish communication between client and server
- all clients communicate with one server that contains the routing mechanism for the ORB

Additional Services of ORBs

- Event service
- Multi-threading
- Resilience and Fault-tolerance services
- Communication-optimization services
  - dynamic routing, load balancing, performance monitoring, and
  - dynamic loading
- Transaction services
- Location brokering and object-broker services
- Other services
  - memory management
  - versioning and change management
  - security services, timing services, error handling
- Etc
  - Administrative services and Development Services

Additional services of ORBs -1

- **Event service**

- extend the capability of ORBs to support various communication approaches
  - one-way, synchronous, deferred synchronous, asynchronous, broadcast, multicast

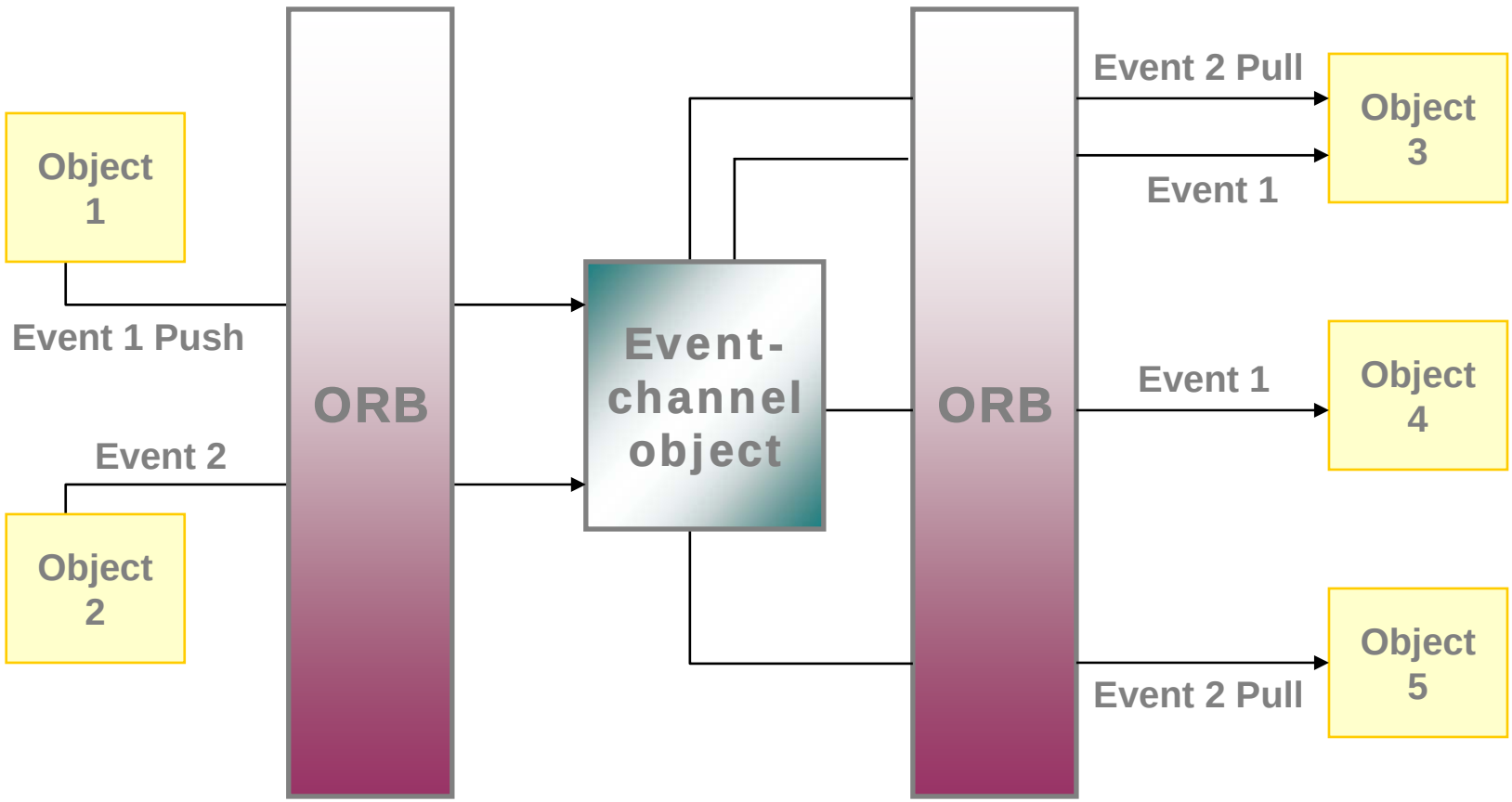
  - basic ORB implementation require
    - the client issuing the request and the object satisfying the request have to be up and running in order for the request to be successful
    - if a request fails because the server object is unavailable, the ORB sends the client an exception which it must then handle

Additional services of ORBs -2

- **Event service (cont.)**
  - in order to obtain this de-coupling of consumer and supplier, most ORBs use intermidiate objects that act like a sort of post-box for storing events received from multiple supplier and sending off events to multiple consumers asynchronously
    - in CORBA, terms these objects event channels
  - event-channel object's responsibility to determine how events are propagated
    - broadcast, multicast
  - ORB terminology use event filtering instead multicast
    - all event filtering is achieved via event channels which can select events based on criteria in the message or by the message or by criteria defined in the channel itself

Additional services of ORBs -3

- **Example use of event-channel object**



Additional services of ORBs -4

- **Multi-threading**
  - ORB can create new client threads each time when they bind with an object's implementation
  - ORB support for thread management and scheduling
  - whenever a shared resource is accessed, locks are applied by the ORB and released upon completion of the thread
  - ORB can also be multi-threaded
    - with various degrees of multi-threading support
    - the number of threads is limited by the virtual memory available

Additional services of ORBs -5

- **Resilience and fault-tolerance services**
  - provide facility that automatically cope with network failure by
    - for example
    - re-connecting a client to a copy of the object the ORB creates and clones
    - client has no need to know of the failure
  - object replication
    - ORB use fully replicated ORB cores to provide continuous back-up
    - ORB prevent loss of service due to network or host failure

Additional services of ORBs -6

- **Communication-optimisation service**
  - optimise the communication and number of calls made by avoiding unnecessary calls
    - for example
    - if the client and server object reside on the same machine, the ORB and network may be by-passed altogether
  - dynamic routing - of messages to objects
  - load balancing - with the facility to query the loads on hosts to determine the appropriate host to use
  - altering - a performance monitor bridge to systems management tools
  - dynamic loading - the object is loaded at run-time from anywhere on the network to an appropriate host when it is needed and where it is best placed

Additional services of ORBs -7

- **Transaction services**
  - include the ability to log transaction
    - if the network fails to recover from the error, roll back all the resources involved to their prior state
  - record the state of participating objects in a cache or log
    - with the ability to roll back to prior state

Additional services of ORBs -8

- **Location brokering and object-broker services**
  - enable an application to locate objects access multiple ORBs
  - the location broker creates a network-services request and connects the application to the ORB
  - if a “trader” service is included, each server can register the services it provides
  - broker can then substitude other servers if a server fails or is unavailable

Additional services of ORBs -9

- **Other services**

- **memory management** - memory allocation may be controlled by the ORB in all the communicating processors with memory management
- **versioning and cache management** - object versions can be kept and managed via the directory or information repository with the relationship between object versions
- **security services** - many of product implementations already use various forms of security checking
- **time services** - some product provide a bridge to DCE services
- **error handling** - error at the client and server may be logged, analysed and queried

Administrative services of ORBs -1

- ORB usually provide the following sorts of service for administering objects

- **object naming service**
  - employ some sort of naming scheme and services to handle naming conversions, so that objects are bound with names that are not implementation-dependent
- **object life cycle service**
  - basic services to enable objects to create, move, delete, copy
- **object property service**
  - sort of properties the developer can may include version, the date created, time created, the person creating the object, synonyms of the object name

Administrative services of ORBs -2

- **Object relationship service**
  - objects are related to one another by inheritance or by being contained by other objects
  - dependency between objects are known as object relationships
  - the services provided in this context ensure that information about the connections is kept
  - used to refine the life cycle services
- **object query service**
  - services to find out what objects exist, what their properties and relationships are and what they do
- **object licensing service**
  - way of charging for objects and licensing objects to users or applications
  - the services are needed to handle the charging and billing when objects are used, either on a one-off basis or one a per-use basis

Development services

- Use a purpose-built language for defining the interface to an object and for constructing message
  - CORBA-conformant products use IDL, OpenBase use CDL, Netsmiths use OIL
  - advanced visualisation tools
  - compiler
  - debugger
  - test tools
  - object browsers

Two Category

CORBA

DCOM

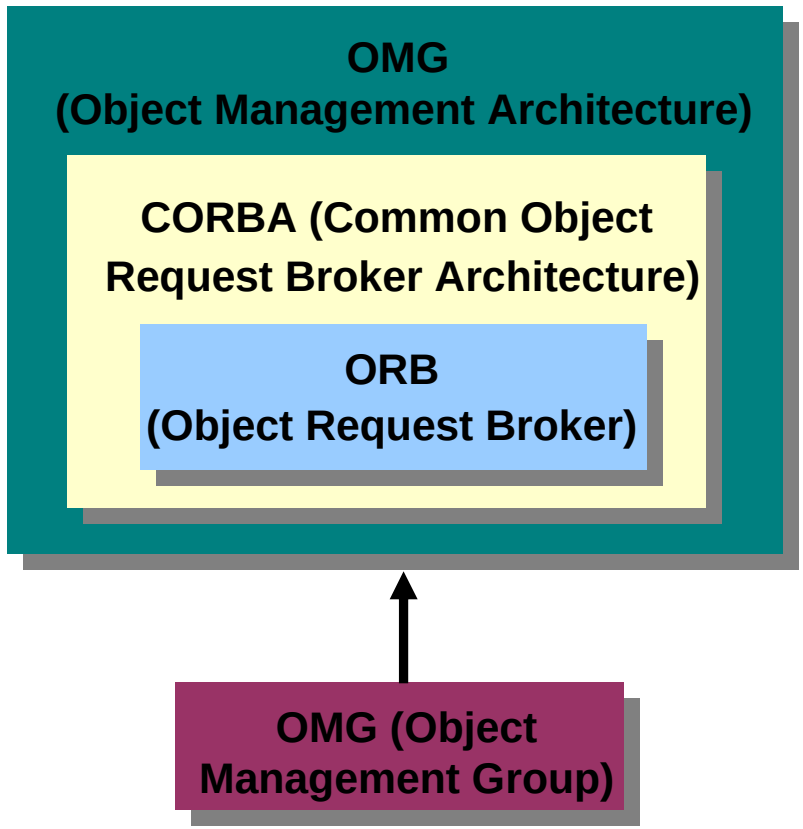
CORBA-based ORBs
Introduction to CORBA -1

- Common Object Request Broker Architecture and Specification (CORBA)
  - standard by the OMG
  - the most influential standard in the ORB

- ORB
  - heart of the OMA
  - communication mechanism
    - enabling object to send and receive messages in distributed, heterogeneous environment
  - various services
    - provided by classes and objects which are themselves invoked via object interface

CORBA-based ORBs
Introduction to CORBA -2

- OMG(Object Management Group)
  - 객체지향 표준을 제정하는 컨소시엄
- OMA(Object Management Architecture)
  - 이종의 분산된 환경 하에서 응용 프로그램들이 서로 통합하고, 상호 연동할 수 있도록 하는 표준 기술
- CORBA (Common Object Request Broker Architecture)
  - 이종의 네트워크 시스템에서 객체간의 통신을 가능하게 해주는 아키텍처
- ORB(Object Request Broker)
  - 객체간에 클라이언트/서버 환경을 구축해 주는 미들웨어



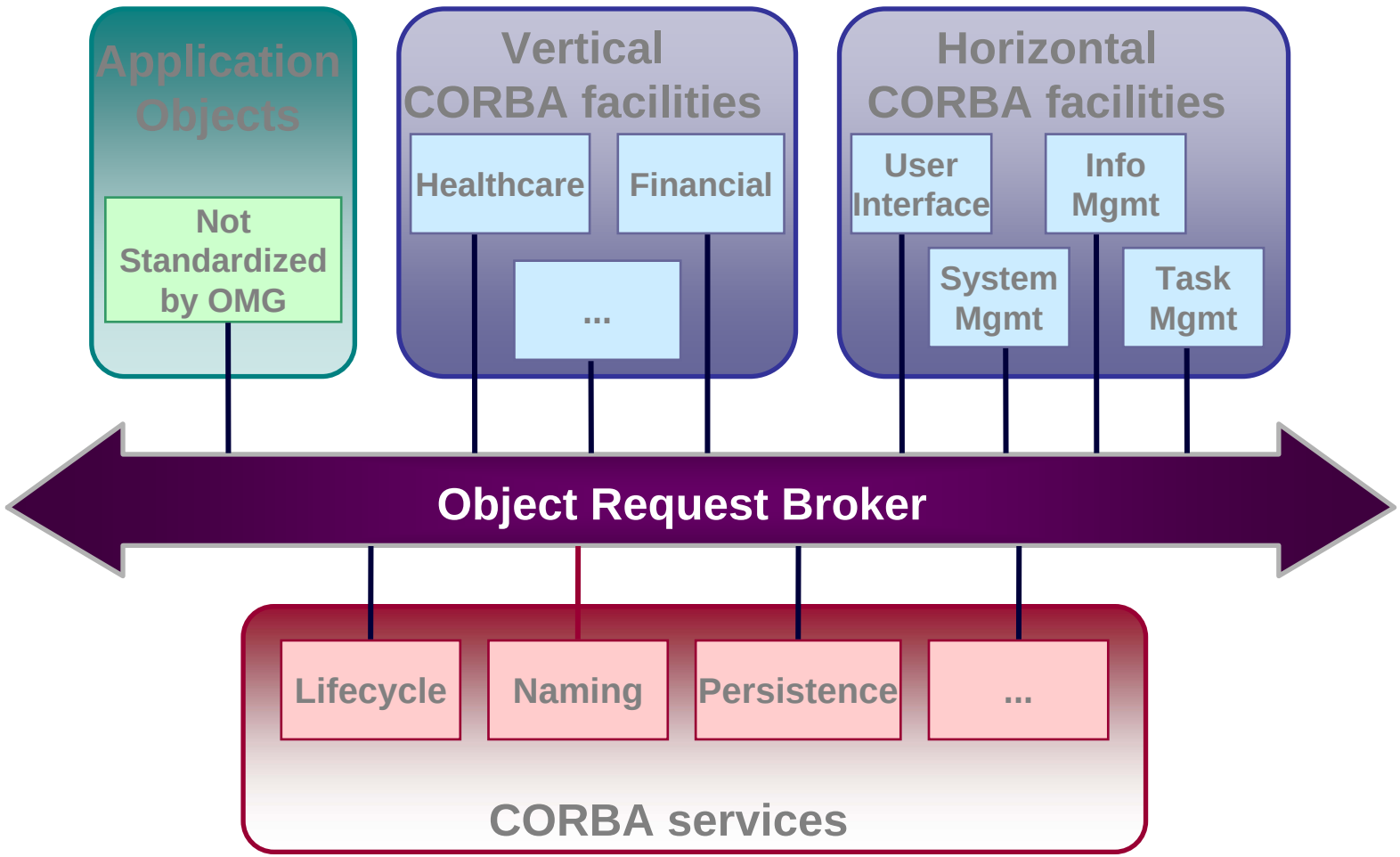
CORBA-based ORBs
ORB's services -1

- **Object services**
  - provide system-level services for objects.
  - basic functions needed in all object-based environments
  - specification and description of these services are defined in the Common Object Service Specification (COSS)
    - object naming, object events, object life cycle, persistent object
    - object relationship, object externalisation, object transactions, object currency control, object licensing, object property, objet query

CORBA-based ORBs
ORB's services -2

- **Common facilities**
  - described in the Common Facilities Specification (CFS)
    - cover such thing as security and time
    - cover less fundamental services than those provided by COSS
    - Horizontal facilities
      - are used by virtually every business (Enterprise-wide)
    - Vertical facilities
      - specialized to particular industry groups (Domain-specific)
- **Application services**
  - provide services for applications.
  - more straightforward
  - cover general-purpose objects which developers might find useful in building applications (checker, graphing objet, mailer, spreadsheet …)

CORBA-based ORBs
Object Management Architecture



How does CORBA work?

- provide basic run-time service
  - Basic CORBA implementation may only provide simple communication services
- does not mandate the location of ORB process
  - ORB functions could be on a separate host, client, server
- many CORBA implementation use an approach which has some of the ORB process on the client and some on the server
  - central component does not used, object requests are passed directly from client code to the object implementation
- use an implementation repository
  - information about the objects (where they are), a description of the data structure (used to hold state information), activation information, the methods, the interface used by the object
  - enables CORBA to send request across the network and find the objects requested

Services of CORBA -1

- **Event service**
  - extend the basic capability of the ORB to support simple synchronous communication
    - provide support for synchronous request
    - bringing the ORB services closer to that used by messaging software
  - broadcast of events to many objects
  - the receipt by an object of multiple events
  - push and pull delivery models
  - reliable event delivery through what OMG calls “appropriate event channel implementations”
  - generation of events without needing to know the customer
  - receipt of events without needing to know the sender
  - multicast events via event filtering

Services of CORBA -2

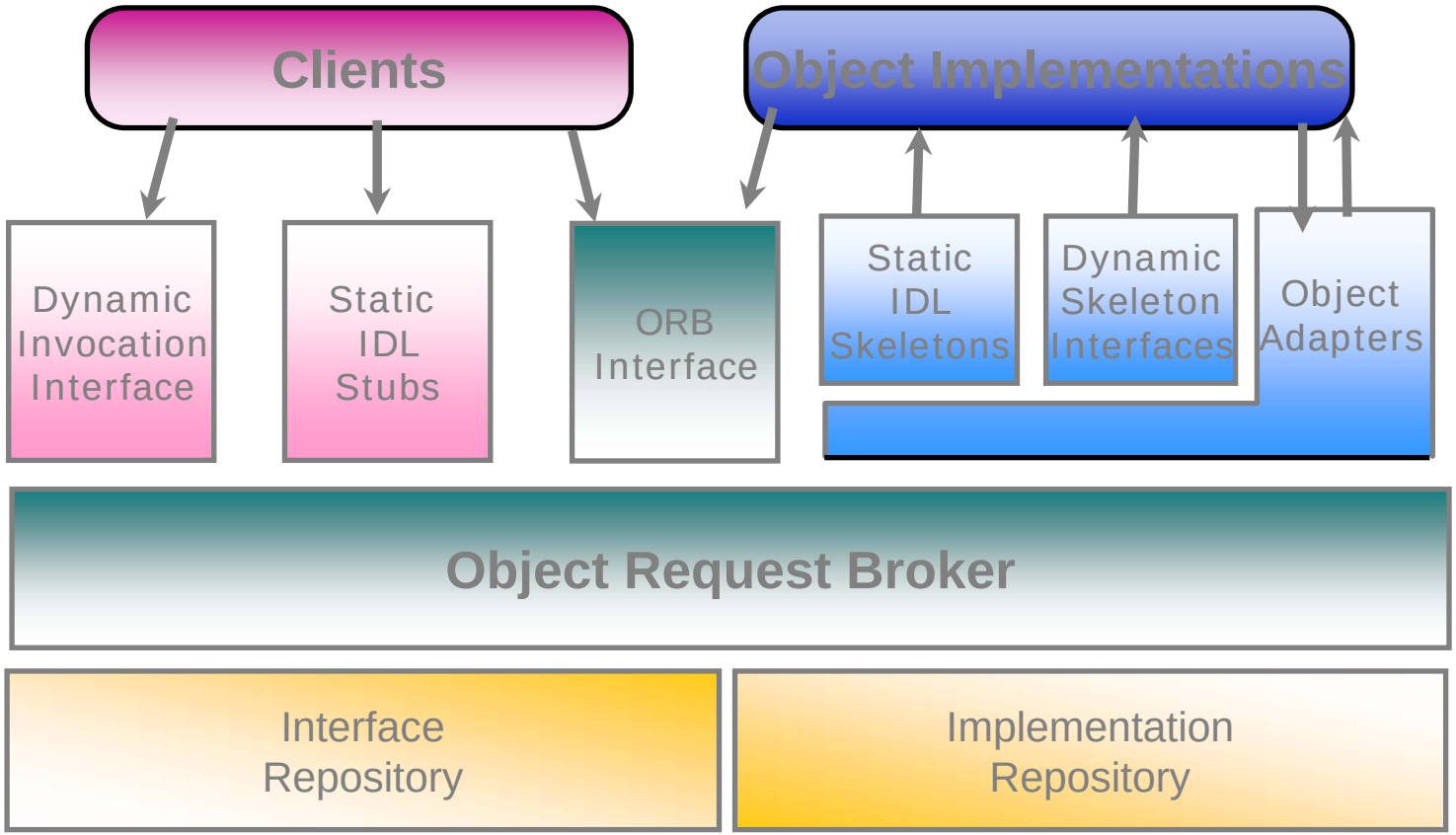
- **Life-cycle service**

- responsible for create, delete, copy, move distributed objects in multiple locations
- “factory”
  - collection of objects geared towards the task of functions such as creation and deletion
  - through factory, objects are designed to help the user manage application objects, accessed via IDL interfaces

Services of CORBA -3

- **Naming service**
  - provide the ability to bind a name to an object relative to a naming context
  - give objects an identifier that is independent of computer language, h/w environment, location and international language
  - based on creating, removing, listing and generally managing names and contexts and so on resolving names and binding objects to names

CORBA-based application



DCOM vs. CORBA

목 차

1 COM/DCOM

2 CORBA

3 DCOM과 CORBA의 구조비교

COM/DCOM

COM/DCOM의 등장

- DCOM의 근간은 COM
- PRC를 이용한 원격함수 호출을 분산 응용 프로그램 개발도구로 사용

COM이란 무엇인가

- 소프트웨어의 생산성 향상 개념도입
- 객체 지향 프로그래밍이 해결책으로 등장
  - 비용최소화
  - 적시의 UpGrade
  - 재사용
- 객체지향 프로그래밍의 문제점
  - 소스코드 재사용에 의한 문제
  - 개발언어에 대한 의존
  - 버전관리의 불편함

COM이란 무엇인가

- 컴포넌트라는 개념으로 해결
- 객체간의 표준 통신방법을 정하기 위한 MS의 노력
  - OLE 버전 1 (객체간의 DDE 통신방법)
- 범용성 있는 통신방법으로 COM개념 도입
  - OLE 버전 2
  - Component Object Model

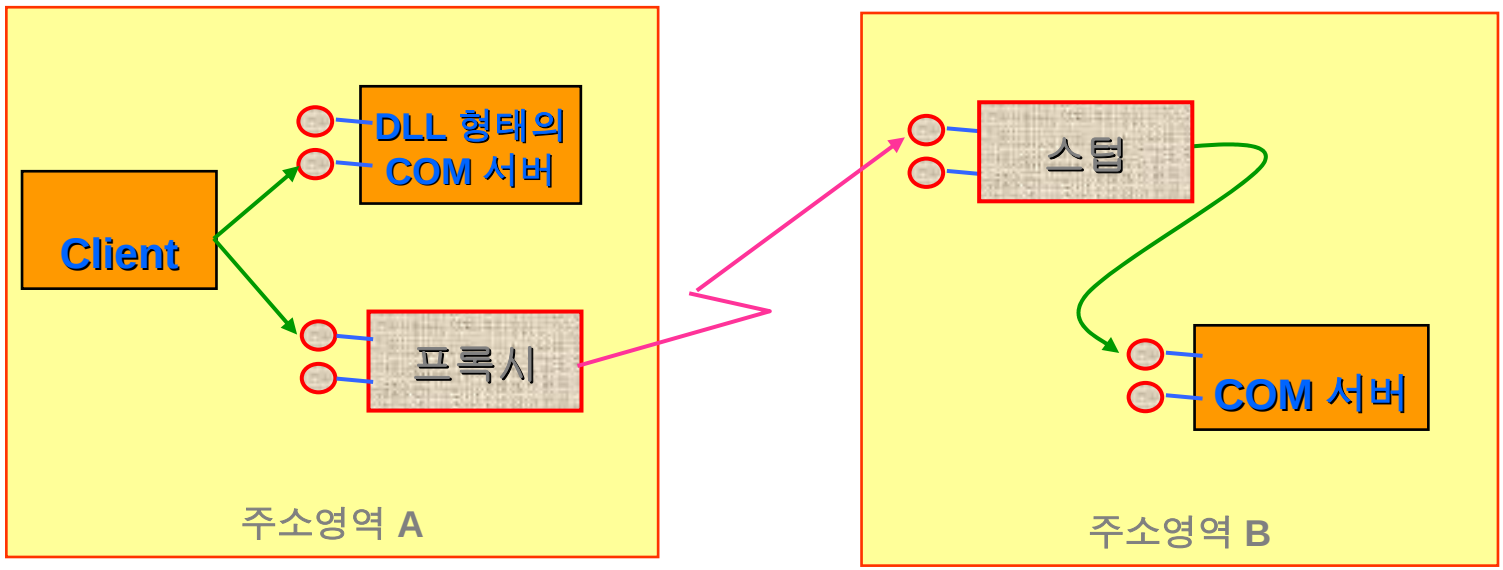
COM의 구성

- COM 서버
  - 자신의 서비스를 하나 이상의 인터페이스를 통해 외부에 공개한다.
- 프록시와 스텁
  - MIDL을 통해 생성한다.
- IID와 CLSID
  - 클라이언트에게 자신이 원하는 COM서버에 접근하기 위해 사용하려는 인터페이스가 무엇인지, 어떤 클래스인지 알려주기위해 COM에서 제공한다.

COM의 구성(계속…)

- COM 서버 종류
  - DLL, Active X 형태
    - 클라이언트와 동일한 주소영역에서 동작
    - 클라이언트에서 직접 COM서버의 인터페이스를 사용해 COM서버의 서비스를 호출한다.
  - EXE 화일 형태
    - 클라이언트와 분리된 주소영역에서 동작
    - 프록시와 스텁간의 관계로 호출이 이루어진다.

COM의 구성(계속...)

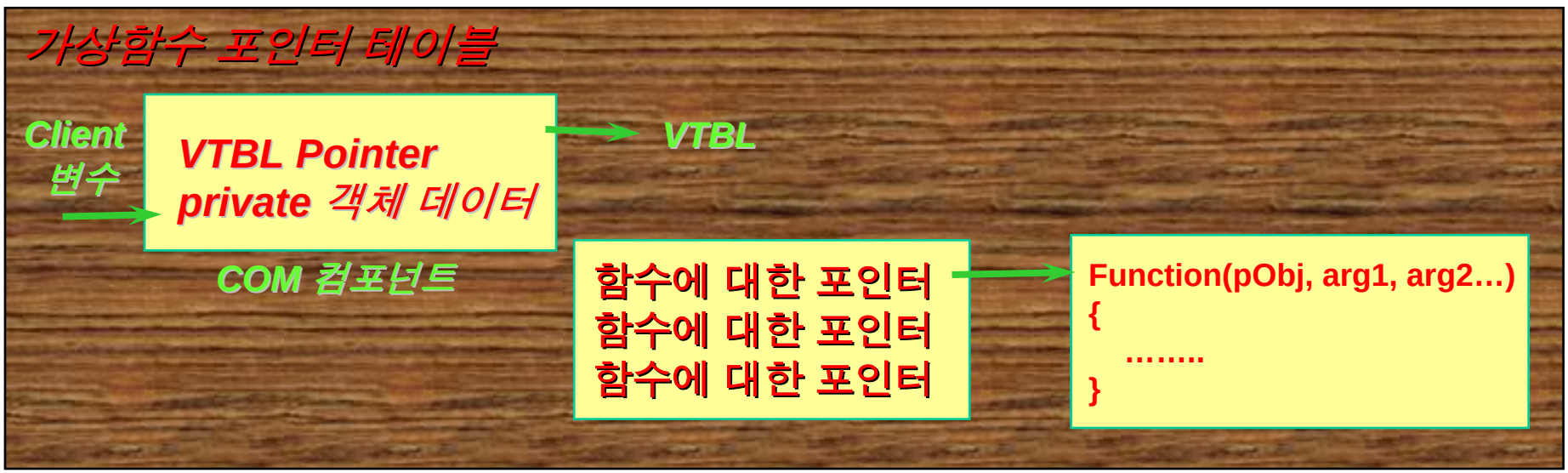


COM의 특성

- Binary 재사용 규약
  - 다른 사람이 만든 바이너리 코드를 소스 없이 가져다 사용
  - 코드 재사용 문제 해결
  - 가상함수 포인터 테이블 사용
    - (Virtual Function Pointer table : vtable)
    - 모든 COM 객체는 반드시 함수 포인터를 담고 있는 메모리 블록을 가져야 하고, 이 메모리 블록 안에있는 함수 포인터만을 통해 클라이언트는 COM객체안에 있는 함수를 사용한다.
    - 모든 COM 컴포넌트는 Iunknown클래스를 상속해야 한다.

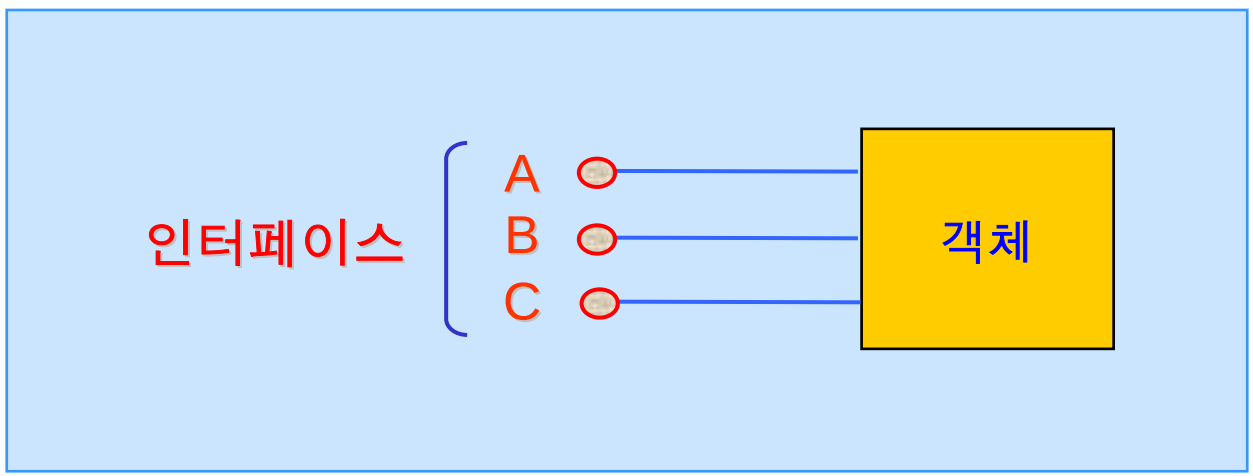
COM의 특성(계속…)


```
Class Iunknown
{
   public:
        virtual HRESULT QueryInterface(REFIID iid, void** ppvObj) = 0;
        virtual ULONG AddRef() = 0;
        virtual ULONG Release() = 0;
};
```



COM의 특성(계속...)

- 인터페이스
  - QueryInterface
    - COM 객체에서 재사용 가능하도록 내놓은 함수 포인터를 리턴해 준다.
    - 클라이언트는 이 함수 포인터를 사용해 COM객체내의 함수를 사용할 수 있다.



COM의 특성(계속...)

- 레퍼런스 카운트 기법
  - AddRef
  - Release
  - COM객체의 생성과 소멸
    - COM객체는 스스로 생성하고 스스로 소멸된다.

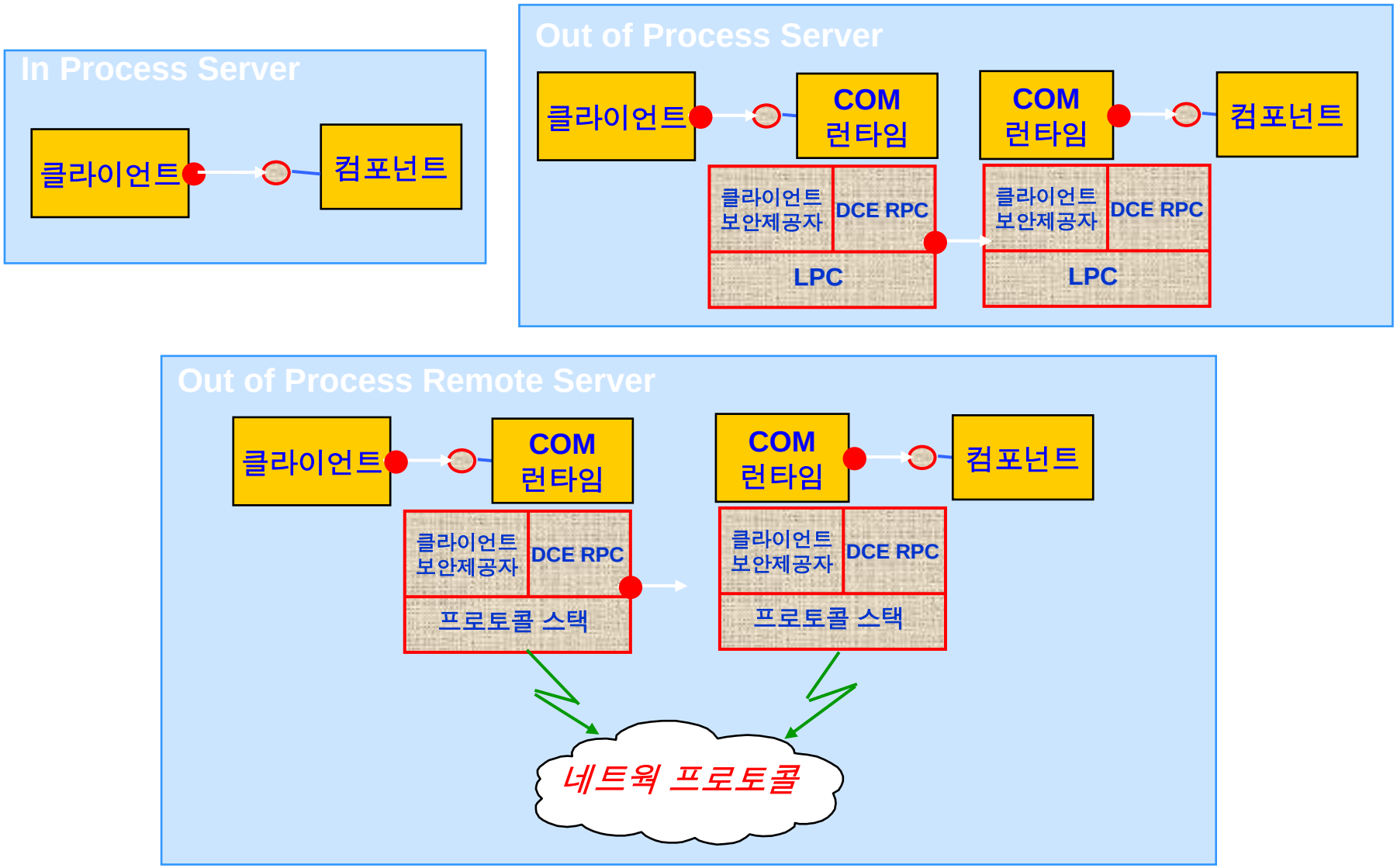
```
ULONG CMyComponent::AddRef()
{
    return ++m_cRef;
};
```

```
ULONG CMyComponent::Release()
{
    if (--m_Ref == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
};
```

DCOM의 구조

- COM의 확장
  - 네트웍으로 연결된 두 컴퓨터 안에 있는 컴포넌트간의 통신 구현
- 클라이언트 기준에서 COM의 위치에 따른 세가지 구조
  - In Process Server
  - Out of Process Server
  - Out of Process Remote Server

DCOM의 구조(계속...)


In Process Server
클라이언트
컴포넌트
Out of Process Server
클라이언트
COM
런타임
COM
런타임
컴포넌트
클라이언트
보안제공자
DCE RPC
LPC
클라이언트
보안제공자
DCE RPC
LPC
Out of Process Remote Server
클라이언트
COM
런타임
COM
런타임
컴포넌트
클라이언트
보안제공자
DCE RPC
프로토콜 스택
클라이언트
보안제공자
DCE RPC
프로토콜 스택
네트웍 프로토콜

DCOM의 특성

- 컴포넌트 재사용 가능
  - 개발비용 및 개발 시간 단축
- 위치 투명성 제공
  - 소스 코드의 수정 없이 컴포넌트를 적절히 배치해 컴포넌트간 통신 부담 최적화
  - 다양한 성능 최적화 가능
- 개발 언어 제약에서 탈피
  - 두 객체간 통신방법으로 개발언어의 특성과는 무관한 바이너리 인터페이스 제공

DCOM의 특성(계속...)

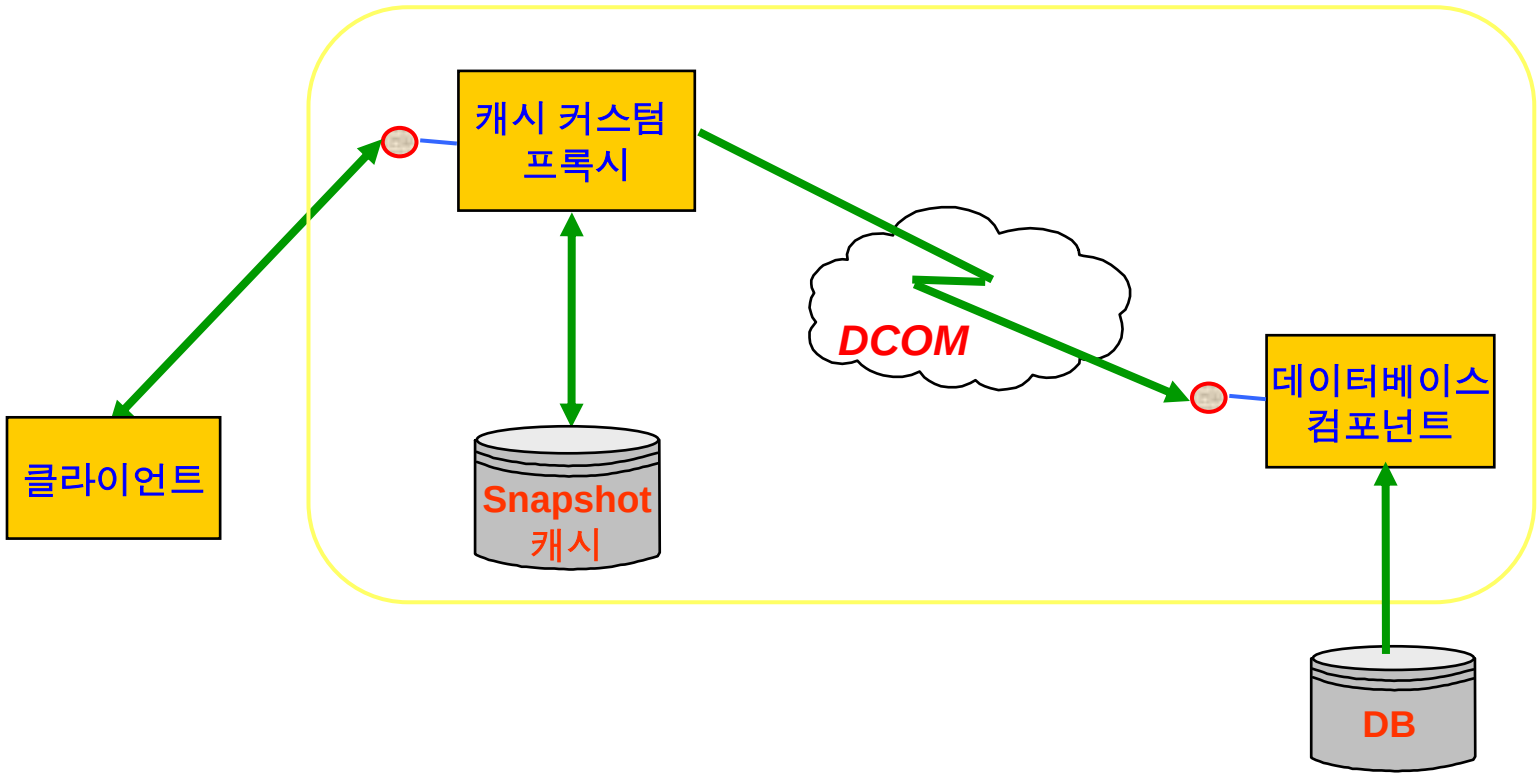
- 효율적인 연결관리
  - Ping 프로토콜 & 레퍼런스 카운트 기법
- 다양한 성능 증대기법 제공
    - 얼마나 많은 사용자를 감당할 수 있는가
    - 얼마나 많은 데이터를 처리할 수 있는가
    - 얼마나 많은 기능을 수행할 수 있는가
  - 대칭형 멀티 프로세스 시스템
  - 위치 투명성을 이용한 서버 컴포넌트 재배치
  - 동일한 서버 컴포넌트에 대해 다른 인터페이스를 통해 추가 기능 제공 가능

DCOM의 성능

- 통신 대역폭과 지연시간의 최적화
  - 단 한번의 Ping 메시지 전송으로 모든 서버 컴포넌트와의 연결 검증
  - 프록시 객체 지원
    - 네트웍 회귀 회수를 줄이기 위한 일괄 처리 방식의 대안
- 다양한 보안 기법 제공
  - NT보안 체계 사용 (확장 가능)
    - 윈도우 NT NLTM 인증 프로토톨
    - 커버로스 버전 5 인증 프로토콜

DCOM의 성능(계속...)

클라이언트 측 캐싱 : 프록시 객체



DCOM의 성능(계속...)

  - 분산 암호 인증
  - 안전 채널 보안 서비스
  - DCE 호환 보안 체계
- DCOM이 컴포넌트의 위치를 숨겨 주듯이 편리한 컴포넌트 설치방법 제공

- 편리한 컴포넌트 설치방법 제공
  - 위치 투명성
  - 다양한 사용자 인터페이스 지원
  - 편리한 클라이언트/서버 업그레이드

DCOM의 성능(계속...)

- 다양한 프로토콜과 플랫폼 지원
  - 프로토콜 투명성/프로토콜의 보안 체계 지원
  - Win95/WinNT
  - UNIX용 DCOM - 소프트웨어 AG와 디지털, MS 공동 개발
  - 자바/DCOM 연동 - IE3.0 부터 자바가상머신 제공

DCOM의 성능(계속...)

- 인터넷과의 연동 기능 제공
  - 가상 사설 네트웍상의 DCOM
  - 방화벽을 통해서도 완전한 동작

DCOM의 기능 정리

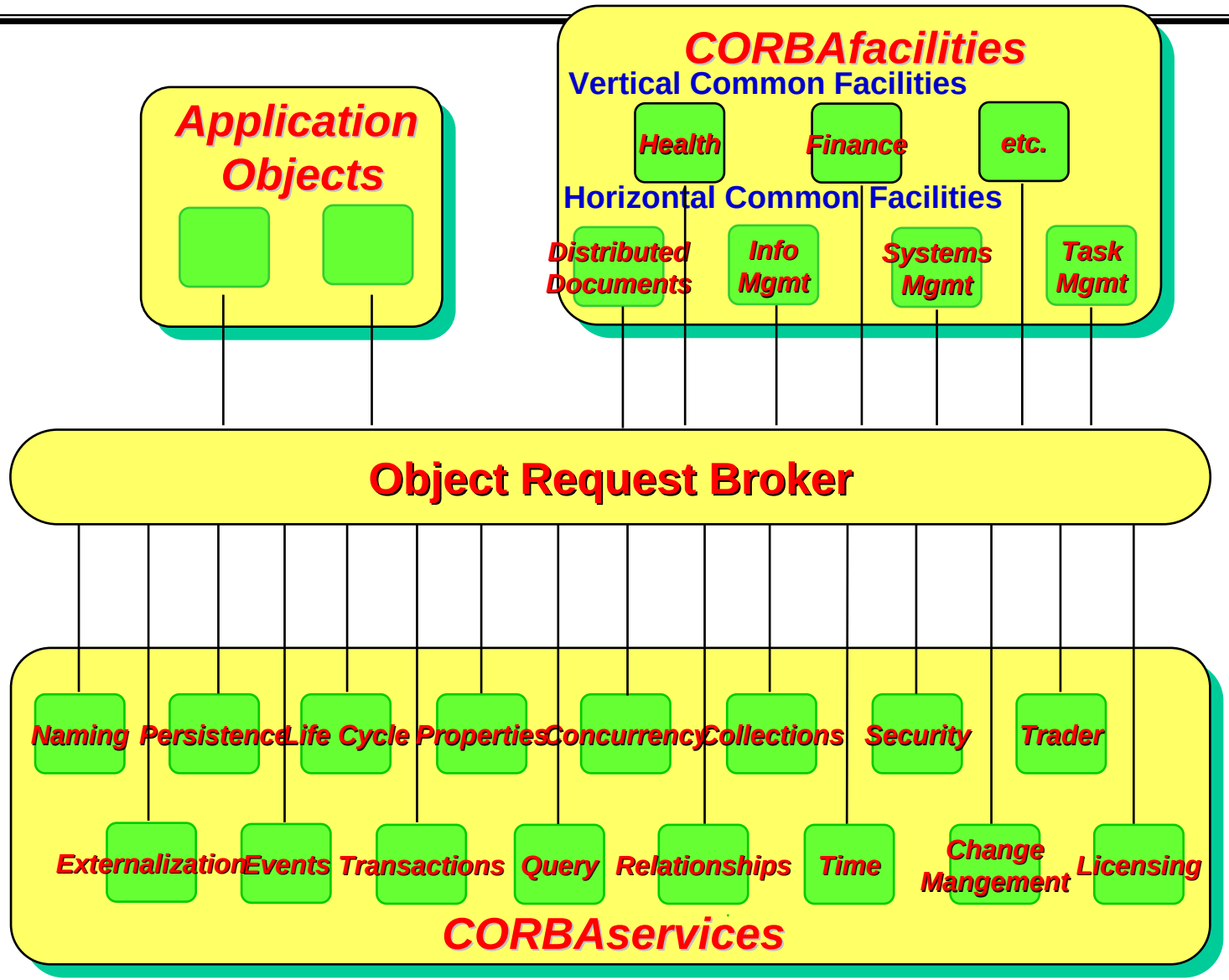
- 컴포넌트 재사용으로 인해 개발 비용 및 개발시간 단축
- 위치 투명성 기능 제공으로 다양한 성능 최적화 가능
- 클라이언트와 컴포넌트 간의 효율적인 연결관리
- 적절한 배치를 통한 성능 향상
- 편리한 버전 관리로 분산 응용 프로그램 관리가 편리
- 다양하게 구비된 보안체계로 안정성 향상
- 다양한 프로토콜 지원
- 플랫폼과 무관

2

CORBA

CORBA Overview

- OMA



3

DCOM/CORBA 구조 비교

DCOM

- COM에 분산기능을 확장 시킨 것으로 원격 객체를 지원하기 위해 DCE-RPC위에 ORPC(Object Remote Procedure Call)계층을 만든다.
- ORPC는 표준 RPC 패킷에 DCOM의 인터페이스 포인터를 구별하는 IPID, 버전정보 등을 담아 전달해 함수를 호출한다
  - IPID는 서버에 위치한 특정 객체의 특정 인터페이스를 가리키게 된다.

DCOM(계속…)

- 클라이언트는 인터페이스를 얻어 원격 컴포넌트가 마치 자신의 주소 공간에 있는 것으로 간주하고 메소드를 호출한다.
- 호출을 받은 컴포넌트는 주어진 일을 마치고 결과를 다시 RPC계층에 전달한다
- ORPC계층에서 호출한 컴포넌트를 찾아 응답을 보내는 일을 담당한다.

CORBA

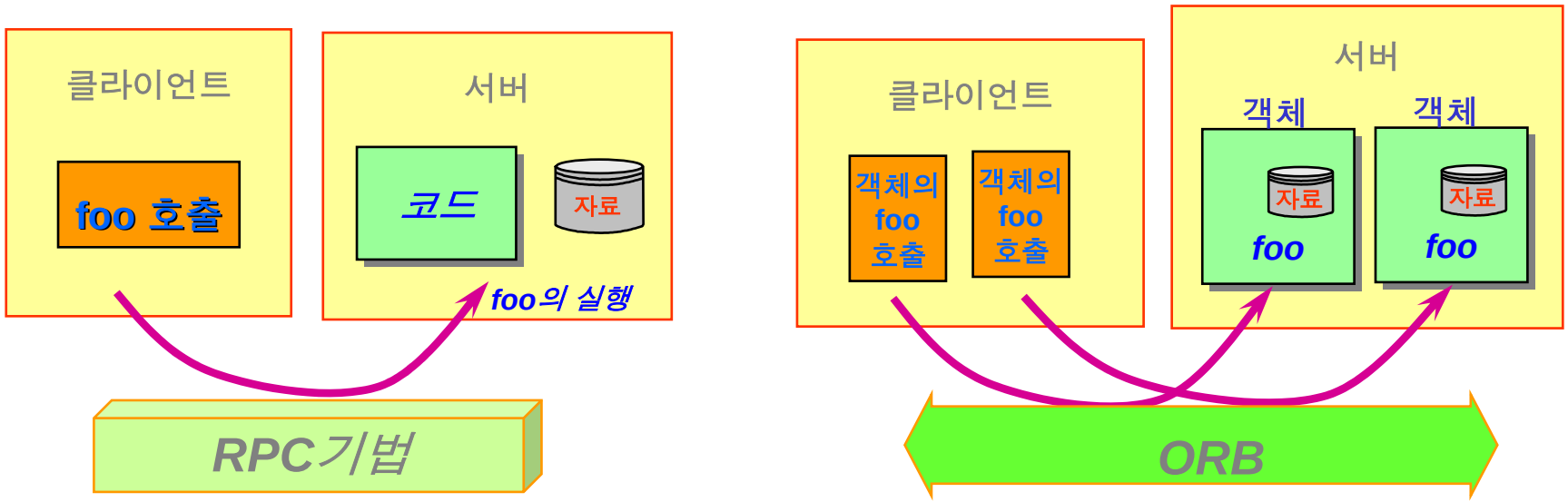
ORB

- CORBA의 핵심은 객체간의 클라이언트/서버 관계를 성립시켜주는 ORB라는 미들웨어이다.
- 하나의 컴포넌트가 다른 컴포넌트를 호출하면, 클라이언트는 IDL 스텁이나 동적호출 인터페이스를 통해 메소드 호출을 ORB로 보낸다.

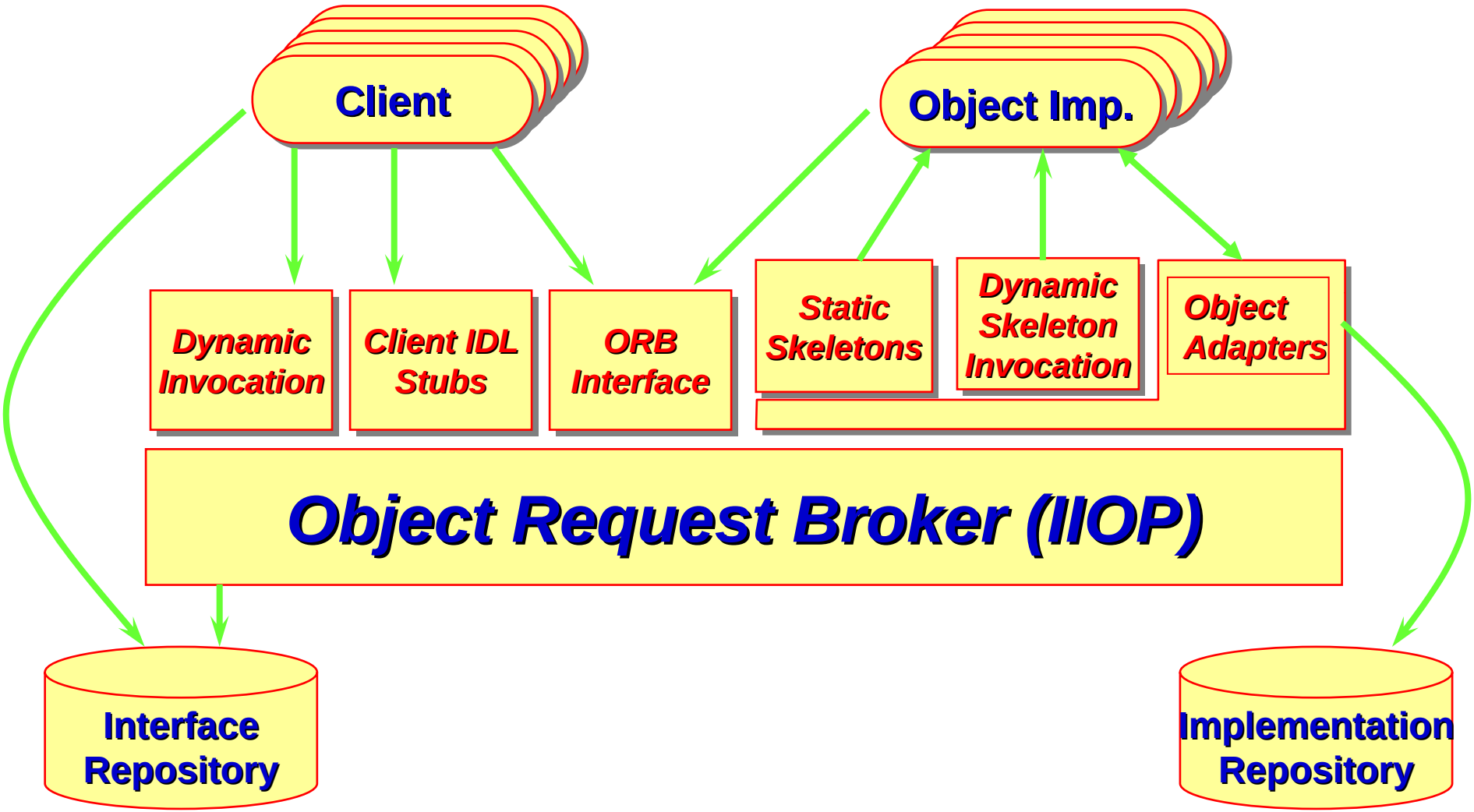
CORBA(계속...)

- 스텁은 객체 서비스에 정적인 인터페이스를 제공한다.
  - 클라이언트는 ORB에서 일어나는 서버객체와의 통신방법이나 활성화 방법, 저장밥법 등은 몰라도 되며, ORB가 다 알아서 해 준다.
- 활성화된 객체는 객체 러퍼런스를 통해 구별되고 클라이언트는 이 객체 레퍼런스를 얻어 메소드를 호출하기 위한 핸들로 사용한다.

ORB 와 RPC

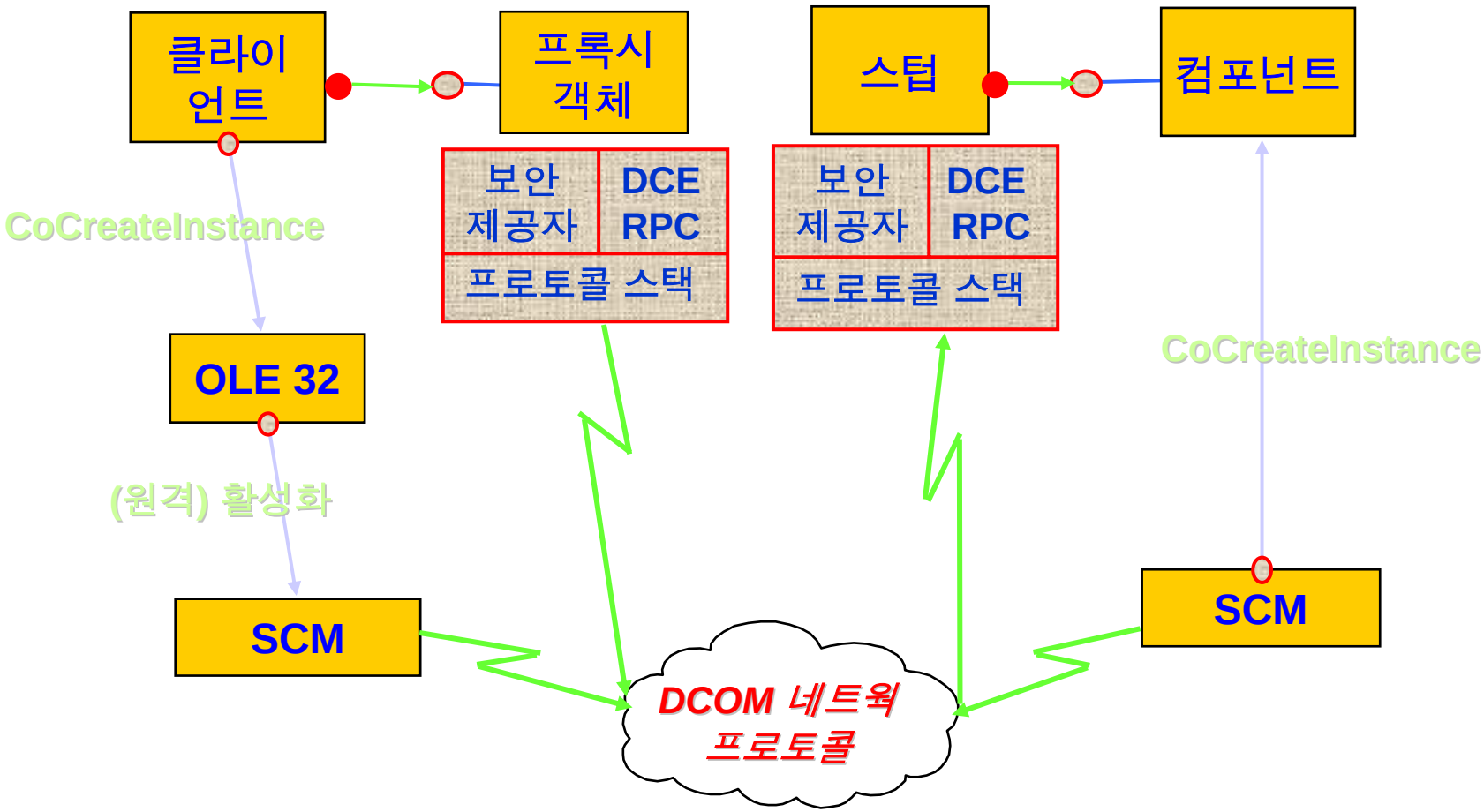

클라이언트
foo 호출
서버
코드
자료
foo의 실행
RPC기법
클라이언트
객체의 foo 호출
객체의 foo 호출
서버
객체
객체
자료
자료
foo
foo
ORB

CORBA 2.0 ORB의 구조

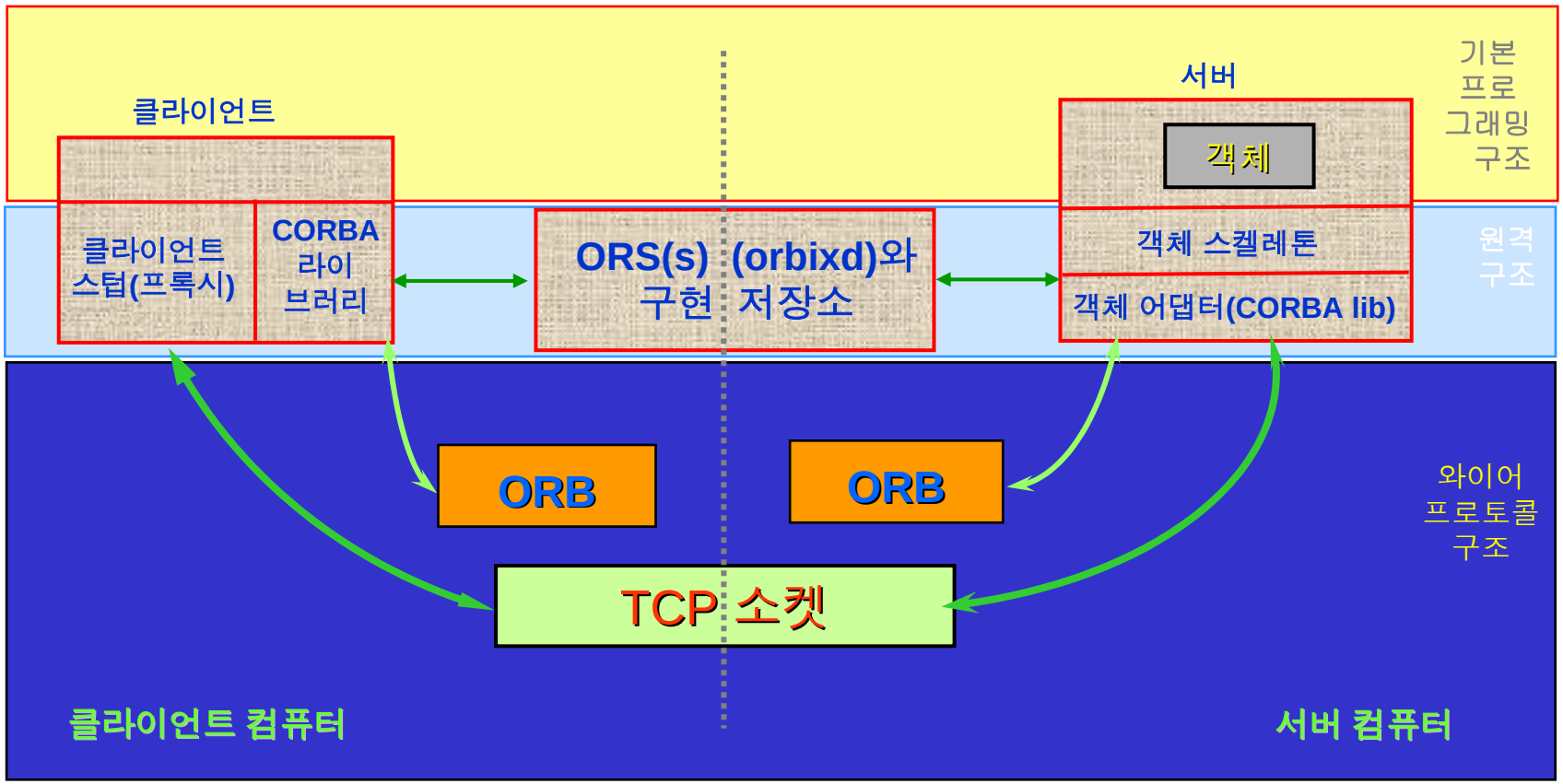


DCOM의 구조

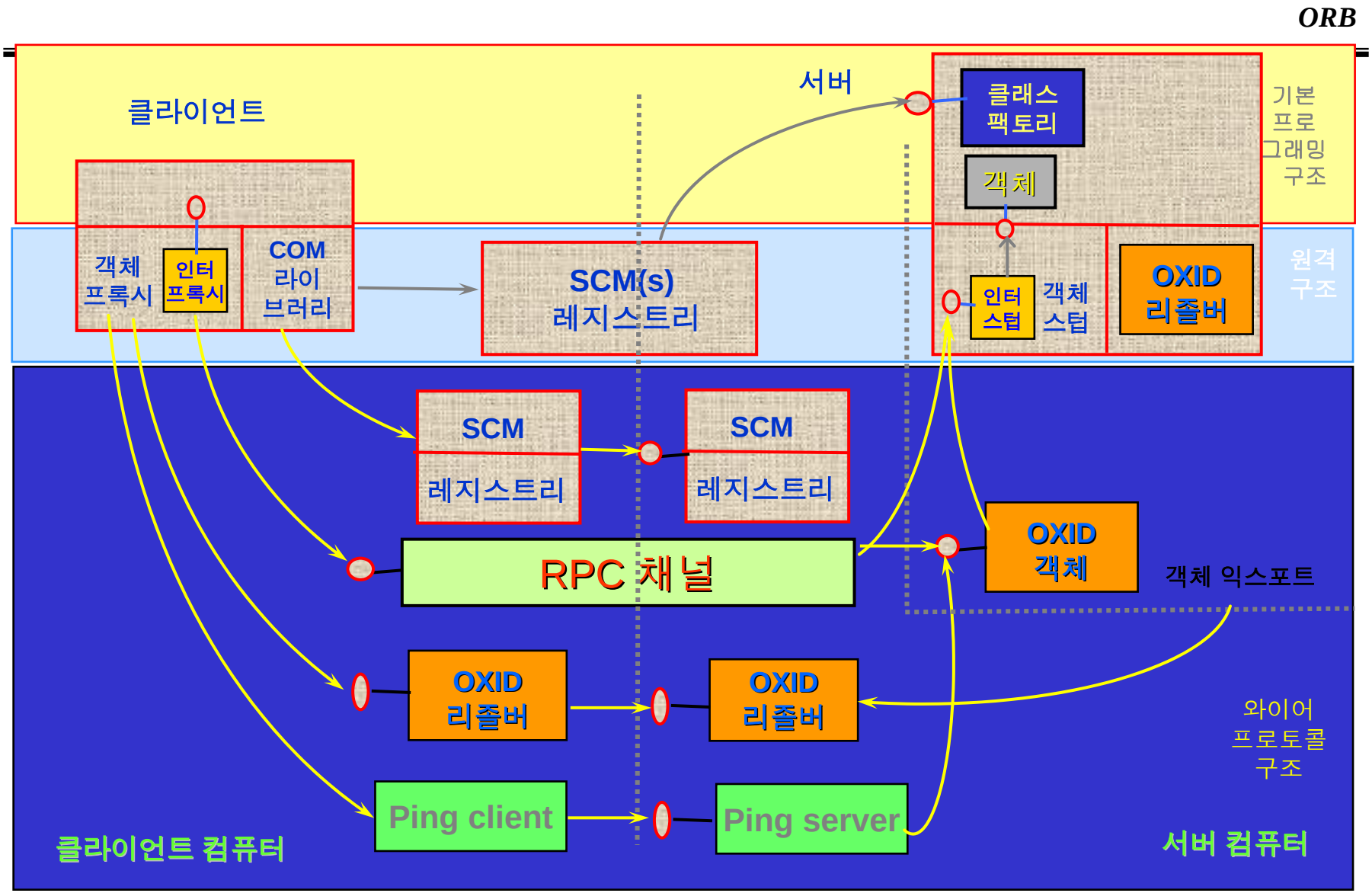
ORB


클라이언트
프록시 객체
스텁
컴포넌트
보안 제공자
DCE RPC
프로토콜 스택
보안 제공자
DCE RPC
프로토콜 스택
CoCreateInstance
OLE 32
(원격) 활성화
CoCreateInstance
SCM
SCM
DCOM 네트웍 프로토콜

CORBA의 전체 구조


기본
프로
그래밍
구조
클라이언트
서버
객체
클라이언트
스텁(프록시)
CORBA
라이
브러리
ORS(s) (orbixd)와
구현 저장소
객체 스켈레톤
객체 어댑터(CORBA lib)
원격
구조
ORB
ORB
와이어
프로토콜
구조
TCP 소켓
클라이언트 컴퓨터
서버 컴퓨터

DCOM의 전체 구조


ORB
클라이언트
서버
클래스 팩토리
기본 프로그래밍 구조
객체
객체 프록시
인터 프록시
COM 라이브러리
SCM(s) 레지스트리
인터 스텁
객체 스텁
OXID 리졸버
원격 구조
SCM 레지스트리
SCM 레지스트리
RPC 채널
OXID 객체
객체 익스포트
OXID 리졸버
OXID 리졸버
와이어 프로토콜 구조
Ping client
Ping server
클라이언트 컴퓨터
서버 컴퓨터

상위 계층

- 기본 프로그래밍 구조
  - 프로그래머의 관점
- 차이점
  - 클라이언트의 객체를 요청과 메소드 호출 방법
  - 서버의 객체 생성과 초기화 방법

상위 계층(계속...)

- DCOM
  - 클라이언트는 COM라이브러리의 CoCreateInstance()를 호출한다.
  - COM의 하부조직은 객체 서버를 시작한다.
  - 서버는 지원되는 모든 CLSID마다 클래스 팩토리를 생성하고 등록한다. 서버블럭은 더 이상 필요없다는 신호가 올 때까지 기다린다. 클라이언트로부터 들어오는 요청은 다른 쓰레드가 처리한다.
  - COM은 클래스 팩토리 포인터를 얻어, CreateInstance()를 호출한다.
  - CreateInstance()에서 서버는 객체를 생성하고, 인터페이스 포인터를 얻기위해 QueryInterface()를 호출한다.
  - COM은 인터페이스 포인터를 클라이언트에게 되돌려준다.

상위 계층(계속…)

- CORBA
  - 클라이언트는 클라이언트 스텁의 정적 함수 ::bind()를 호출한다.
  - ORB는 지원하는 객체를 포함하는 서버를 시작힌다.
  - 서버는 지원되는 모든 객체를 초기 활성화 시킨다. (각 생성자에서 객체 레퍼런스를 생성하고 등록될 수 있게 호출된다.)
  - 서버는 CORBA::BOA::implisready()를 호출해 ORB에게 클라이언트 요청을 받을 수 있다는 것을 알린다.
  - ORB는 객체 레퍼런스를 클라이언트에게 돌려준다.

중간 계층

- 원격 구조
  - 클라이언트와 서버가 마치 같은 주소 공간에 위치한 것처럼 보이게 하는 하부구조
- 차이점
  - 서버객체 등록 방법
  - 프록시/스텁/스켈레톤 인스턴스 생성 시점

중간 계층(계속...)

- DCOM
  - CoCreateInstance()를 호출을 받으면 COM 라이브러리는 SCM(Service Control Manager)으로 넘긴다.
  - SCM은 클래스 팩토리가 등록되어있는지 확인한다. 등록돼 있지 않으면 SCM은 레지스트리를 뒤져서 해당 클래스의 서버 경로 이름을 찾아 서버를 활성화 한다.
  - 서버는 클래스 객체 테이블에 있는 모든 지원되는 클래스 팩토리를 등록한다.
  - SCM은 테이블에서 포인터를 얻고, 그 클래스의 CreateInstance()를 호출한다.
  - CreateInstance()가 포인터를 돌려주면, COM은 새로 생성되는 객체 인스턴스를 위한 스텁을 만든다.

중간 계층(계속...)

- DCOM
  - 객체 스텁은 인터페이스 포인터를 마샬링하고 레지스트리를 탐색해 인터페이스 스텁을 만들고, 서버객체의 실제 인터페이스와 연결한다.
  - SCM이 마샬링된 포인터를 클라이언트쪽으로 보내면, COM은 그 객체 인스턴스를 위한 객체 프록시를 만든다.
  - 객체 프록시는 포인터를 언마샬링하고 레지스트리를 살펴 인터페이스 포록시를 만들고 스텁에 연결돼 있는 RPC채널객체에 연결한다.
  - COM 라이브러리는 클라이언트로 인터페이스 포록시에 대한 포인터를 돌려준다.

중간 계층(계속...)

- CORBA
  - ::bind()호출을 받으면, 클라이언트 스텁은 작업을 ORB에 위임한다.
  - ORB는 구현 저장소에 서버경로 이름을 묻고, 서버를 활성화한다.
  - 서버는 지원되는 모든 객체를 초기에 활성화한다. CORBA::Object의 생성자는 유일한 레퍼런스 ID로 BOS::create()를 호출해 객체 레퍼런스를 얻는다. 그 다음 obj_is_ready()를 호출해 객체 레퍼런스를 등록한다.
  - 클래스의 생성자는 스켈레톤 클래스의 인스턴스를 생성한다.
  - ORB가 객체 레퍼런스를 클라이언트쪽에 보내면, ORB는 프록시 클래스의 인터페이스를 생성하고 해당 객체 레퍼런스와 함께 프록시 객체 테이블에 등록한다.
  - 클라이언트 스텁은 클라이언트에 객체 레퍼런스를 돌려준다.

하위 계층

- 와이어 프로토콜 구조
  - 서버와 클라이언트가 서로 다른 컴퓨터에 위치하고 있을 경우를 지원
- CORBA
  - ORB간의 통신은 전적으로 구현한 업체에 달려있다.
    - GIOP (다른 업체의 ORB 통신을 위해)
    - IIOP (TCP/IP연결 지원)

하위 계층(계속...)

- DOCM
  - OXID객체
    - IRemUnknowm 인터메이스 지원
      - RemQueryInterface()
      - RemAddRef()
      - RemRelease()
    - 같은 객체 익스포터로 가는 여러 개의 원격 Iunknown 메소드 호출을 묶어주어 성능 개선
  - DCE RPC 명세에 기준
    - IRemUnknown 친터페이스, Pinging 프로토콜 추가

하위 계층(계속...)

- 차이점
  - 원격 인터페이스 포인터나 객체 레퍼런스가 서버측의 정보를 클라이언트에게 전달하는 방법
  - 다양한 환경에서의 전송을 위해 자료가 마샬링된 표준 포맷

하위 계층(계속...)

- DCOM
  - 위임된 CoCreateInstance()요청을 받았을 때, 클라이언트측의 SCM이 레지스트리를 찾아 그리드 객체가 다른 컴퓨터 상의 서버에 있으면 IRemoteActivation RPC 인터페이스의 메소드를 호출한다.
  - 서버측의 SCM이 서버를 활성화 시키면 객체의 익스포터와 연결되고, OXID를 받는다. 서버에 접근하기 위한 RPC 바인딩과 OXID 사이의 맵핑은 서버측의 OXID 리졸버를 통해 등록된다.
  - CreateInstance()가 돌려주는 포인터를 객체스텁이 마샬링할 때, 포인터는 서버에서 유일한 인터페이스 포인터 지시자(IPID)가 할당된다. 또 포인터를 나타내는 객체 레퍼런스가 생성된다. 객체 레퍼런스에는 IPID, OXID, 프로토콜당 하나의 OXID리졸버주소 등의 내용이 담겨 있다.

하위 계층(계속...)

- DCOM
  - 서버측의 SCM이나 클라이언트 측의 SCM을 통해 마샬링된 인터페이스 포인터가 돌아오면, 객체 프록시는 객체 레퍼런그에서 OXID와 OXID 리졸버 주소를 추출하고 지역 OXID리졸버의 IOXIDResolve::ResolveOxid() 메소드를 호출한다.
  - 클라이언트측 OXID 리졸버는 OXID의 캐싱된 맵핑이 있는지 알아보고, 없다면 서버측 OXID리졸버의 IOXIDresolve::ResolveOxid() 메소드를 호출한다. 이 메소드는 등록된 RPC바인딩을 되돌려준다.
  - 클라이언트측 리졸버는 맵핑을 캐시하고 객체 프록시에 RPC 바인딩을 돌려준다. 이렇게 하면 객체 프록시는 자신과 새로 생성된 인터페이스 프록시들을 객체 인터페이스에 연결된 RPC채널이 연결한다.

하위 계층(계속…)

- CORBA
  - 위임된 ::bind() 요청이 들어오면, 클라이언트측 ORB는 지원하는 서버 기계를 위치 파일에 물어 찾는다. 그 다음 TCP/IP를 통해 서버측 ORB에 요청을 전달한다.
  - 서버측의 ORB가 서버를 활성화 시키면, 객체는 서버에 의해 활성화 되고, COEBA::Object 생성자가 호출된다. 그 다음 BOA::Create()안에서 BOA는 단말 소켓을 만들고, 객체는 객체 ID를 얻는다. 그리고 인터페이스와 구현 이름, 레퍼런스 ID, 단말 주소를 갖는 객체 레퍼런스도 만들어진다. IIOP프로토콜을 알고 있는 클라이언트를 위해 서버는 기계의 이름, TCP/IP포트번호, object_key를 포함하는 상호 작동 가능한 객체 레퍼런스를 만든다. BOA는 ORB에 객체 레퍼런스를 등록한다.
  - 객체 레퍼런스가 클라이언트측으로 돌아오면, 프록시는 단말 주소를 찾아내 서버에 소켓 연결을 한다.

결론

- DCOM/CORBA의 구조는 매우 흡사하다
  - 원격 객체 활성화 와 투명한 접근을 허용하는 분산 객체 시스템의 하부구조
- 주요 차이점
  - DCOM의 다중 인터페이스 상속 (QueryInterface() 지원)
    - 객체가 원격에서 다중 인터페이스를 동적으로 로드할 수 있다.
  - CORBA의 모든 인터페이스는 CORBA::object를 상속받는다.
    - 객체의 등록, 레퍼런스 생성, 스켈레톤 초기활성화등 모든 객체에 공통적인 임무를 수행한다.
    - DCOM에서는 서버가 담당하거나 DCOM 런타임 시스템이 동적으로 수행
  - 와이어 프로토콜 구조(RPC/ORB)