Lecture 10 Middleware Architecture

- □ Middleware/Framework
- □ CORBA Implementation
- □ Case Study: ACE/TAO

中间件体系架构

Middleware Concept

A software layer resides between the O/S layer and application layer to hide platform heterogeneity and provide communication between different platforms or technologies.

- cross-platform
- component-based
- standard API
- communication architecture

Middleware Definition by IDG

A software layer that resides between the O/S or network protocol and the distributed application to hide platform discrepancy and to support interoperability.

A separate layer of system software or service program that resides on top on O/S and provides functions of resource management and network communication, by which the distributed systems may share resources among various platforms.

中间件体系架构

Major Middleware Types

- 远程过程调用 Remote Procedure Call
- 面向消息的中间件 Message-Oriented Middleware
- 对象请求代理 Object Request Brokers
- 数据访问界面 Unified Data Access
- 事务处理监控 Transaction Processing Monitor

Major Types of Distributed Object Middleware

ORB

Prompted by Object Management Group(OMG). It uses the Common Object Request Broker Architecture (CORBA) to register, publish and request for the resources in a distributed environment. CORBA is one of the major communication middleware that is widely used in industry for distributed application development across heterogeneous platforms

中间件体系架构

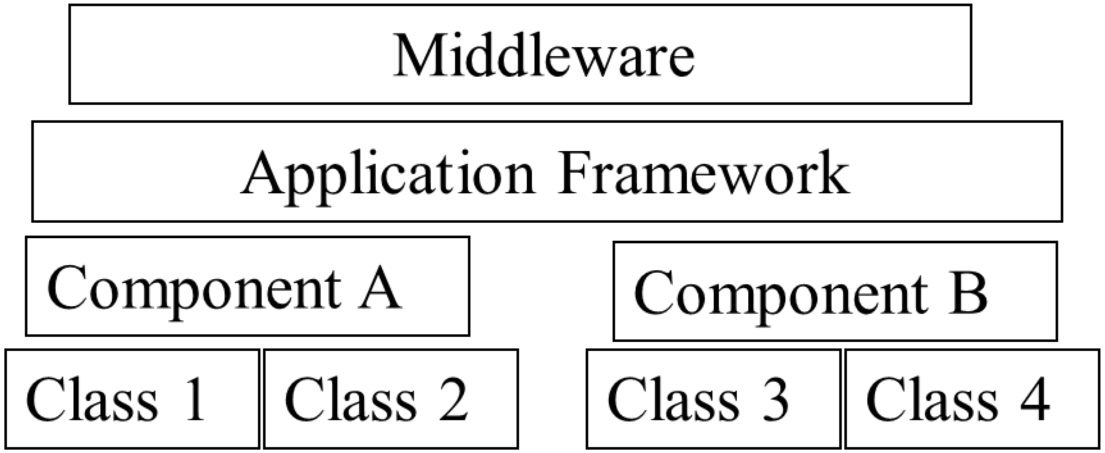
COM/DCOM → .NET Framework

Component Object Model (COM), now replaced by .NET framework is Microsoft’s solution for component middleware. COM defines and provides communication architecture for the software object , by which the client may directly communicate to the object without the interfere of COM, upon the connection is established. DCOM supports the distributed environment.

中间件体系架构

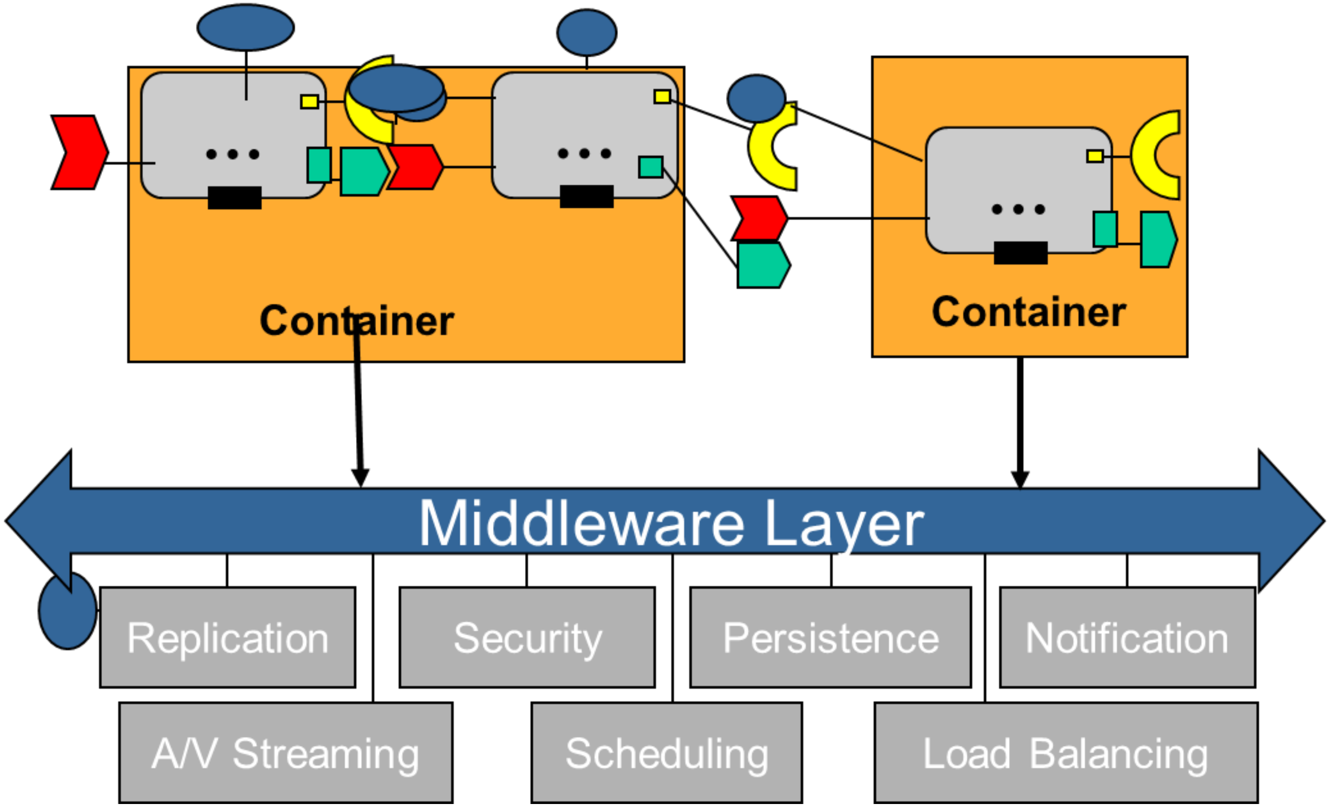
Middleware vs. Component

Middleware can be considered as a software library/subsystem/framework consists of components



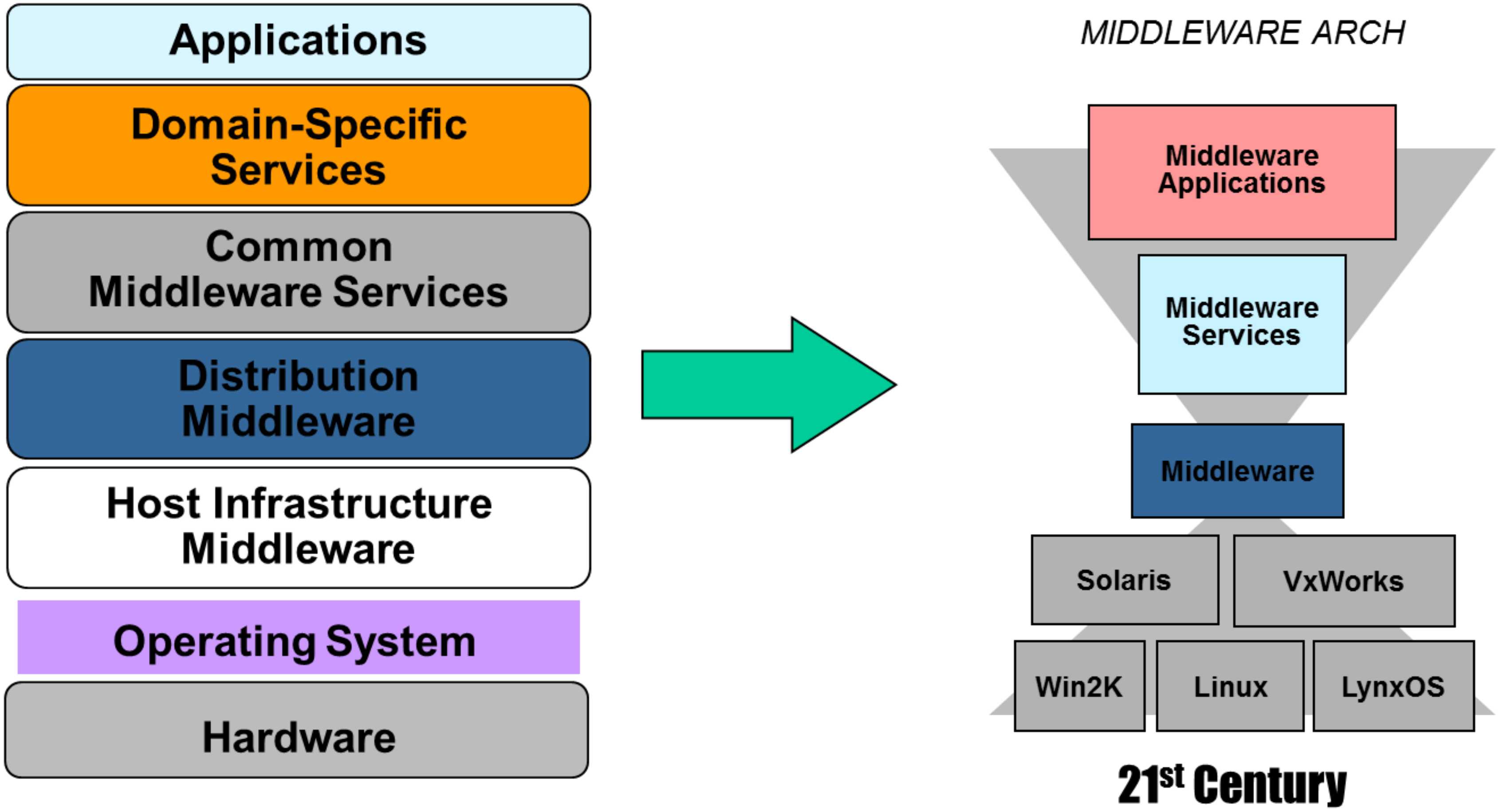
中间件体系架构

Component: encapsulates application business logics and services



中间件体系架构

Middleware layer makes underneath O/S platforms transparent



Applications

Domain-Specific Services

Common Middleware Services

Distribution Middleware

Host Infrastructure Middleware

Operating System

Hardware

MIDDLEWARE ARCH

Middleware Applications

Middleware Services

Middleware

Solaris

VxWorks

Win2K

Linux

LynxOS

21st Century

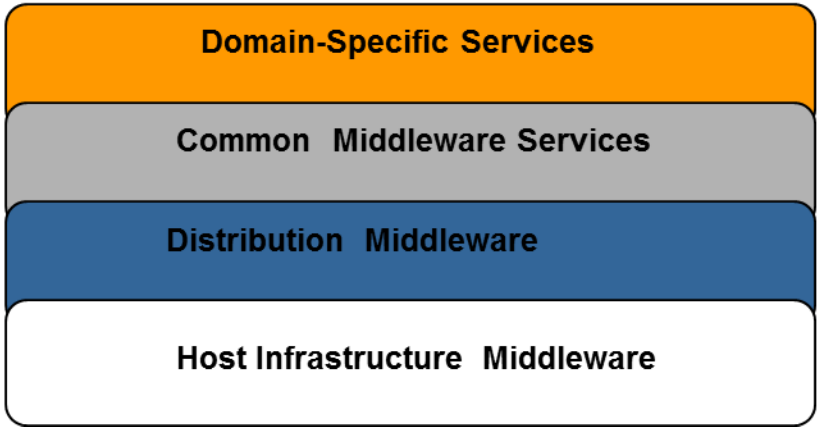
中间件体系架构

Host infrastructure middleware

Encapsulates & enhances native OS mechanisms to create reusable network programming components

Examples:

- Java Virtual Machine (JVM)
- Common Language Runtime (CLR)
- Adaptive Communication Environment (ACE)



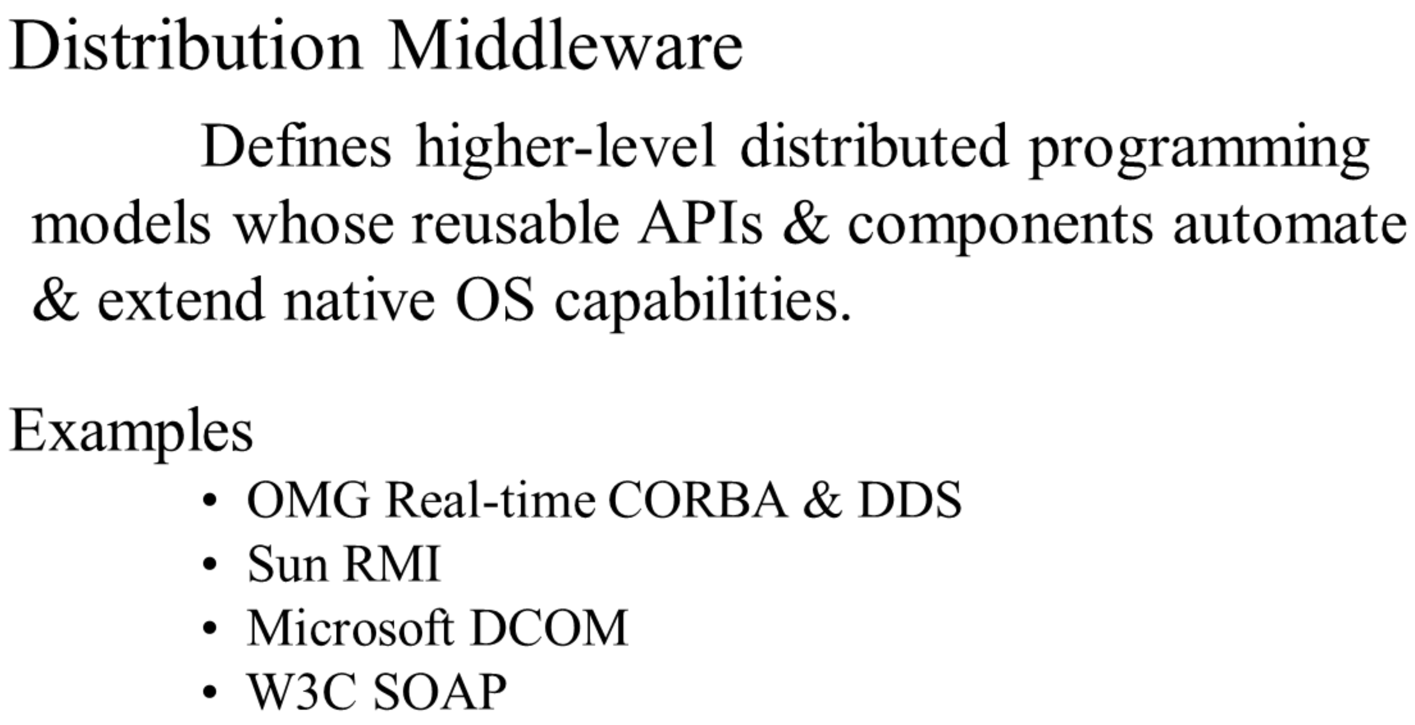
Domain-Specific Services

Common Middleware Services

Distribution Middleware

Host Infrastructure Middleware

中间件体系架构



Distribution Middleware

Defines higher-level distributed programming models whose reusable APIs & components automate & extend native OS capabilities.

Examples

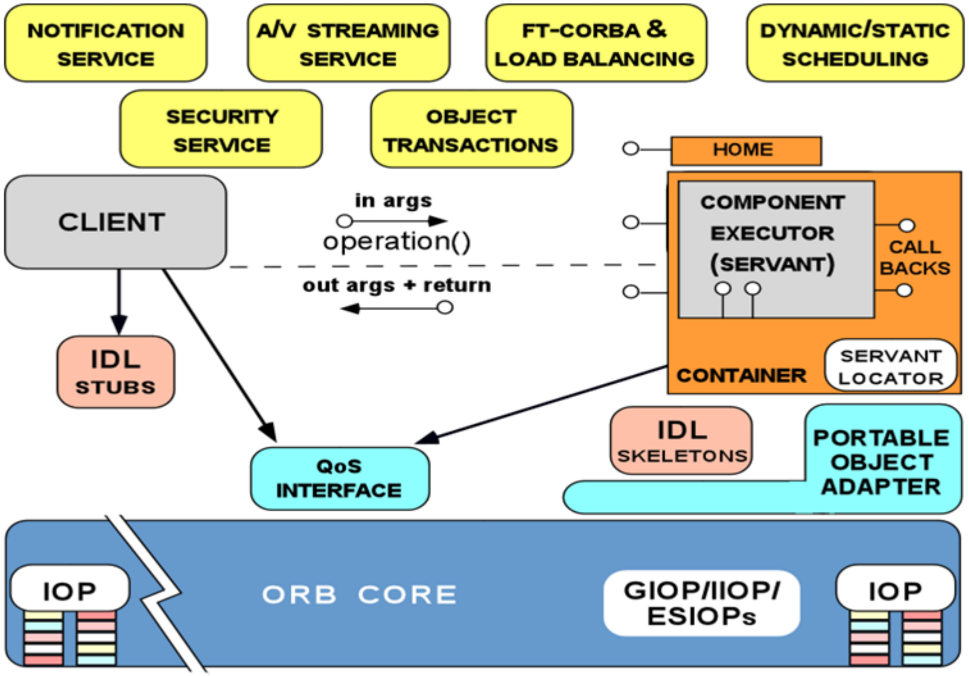
- OMG Real-time CORBA & DDS
- Sun RMI
- Microsoft DCOM
- W3C SOAP

中间件体系架构

Common Middleware

Augment distribution middleware by defining higher-level domain-independent services for “business logic”.

Examples: CORBA Component Model & Object Services, Sun’s J2EE, Microsoft’s .NET, W3C Web Services



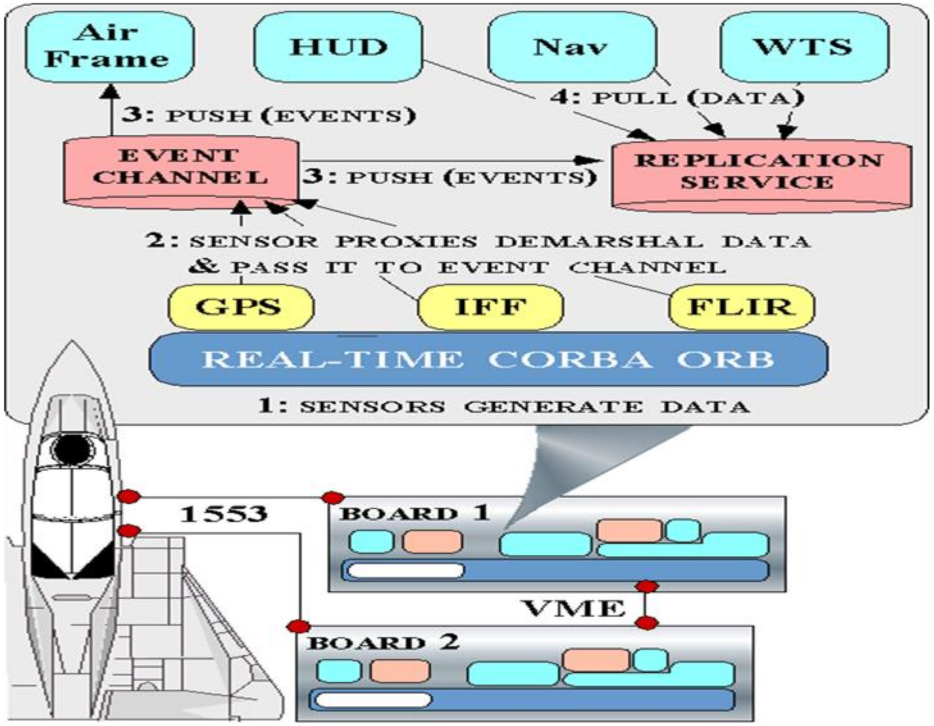
Common middleware services support many recurring distributed system capabilities, e.g.,

- Transactional behavior
- Authentication & authorization,
- Database connection pooling & concurrency control
- Active replication
- Dynamic resource management

中间件体系架构

Domain-specific Middleware

Tailored to the requirements of particular domains, such as telecom, e-commerce, health care, process automation, or aerospace.



Boeing Bold Stroke

Common software platform for Boeing avionics mission computing systems

Advantages of Middleware Solution

- Decrease development period
- Mitigate risk of project development

  - the risk of project failure for software development may reach a 90% probability without using a proved middleware product

  - self-development of middleware brings a too high cost
- Product quality and maintenance
- Better competitiveness on market

Advantages of Middleware Solution (cont’d)

- Transparency to platform interoperability
- Independent to execution platform
- Better scalability

ACE/TAO：A CORBA C++ Implementation

- Intro to CORBA Architecture/Protocol
- IDL, ORB and CORBA Objects
- Client Stub and Server Skeleton
- CORBA Naming Service
- Asynchronous Method Invocation (AMI)

What is CORBA ?

(Common Object Request Broker Architecture)

- An industry standard for distributed object systems (proposed and maintained by the Object Management Group, http://www.omg.org)
- A set of specs and protocols that defines the architecture (IDL, ORB, split of object interface from implementation) API (CORBA Core Spec v3.0.3), and protocols (GIOP, IIOP, Name Service, etc.)

Why bother CORBA ?

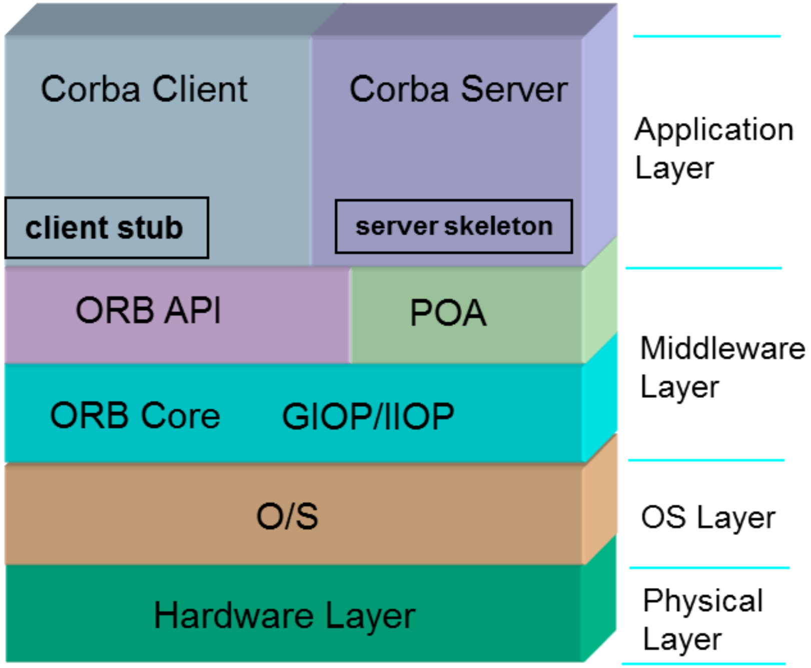
- Separate object interface and reference from object implementation, platform, and languages;
- An IDL-based standardized service interface that can be understood and supported by various vendors;
- CORBA architecture (orb/poa/stub/skeleton) provides a well-designed OO framework;
- IIOP (Internet Interoperable Orb Protocol) provides interoperability with other distributed object systems.

中间件体系架构

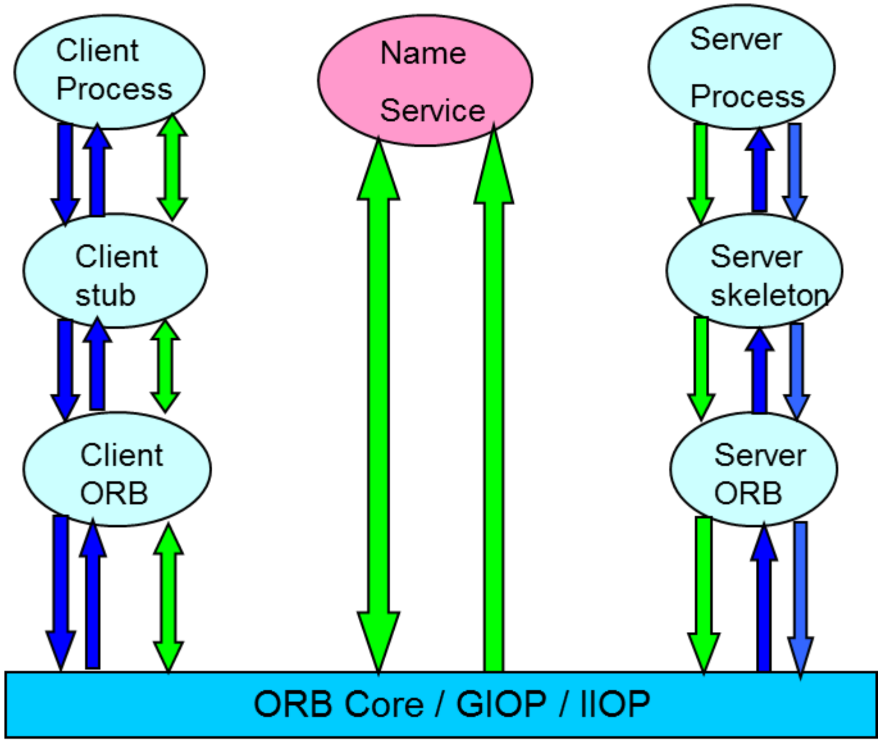
	Java/RMI	COM/DCOM	CORBA
Protocol	JRMP	LRPC/ORPC	GIOP, IIOP
Interface	Remote Interface	MIDL	IDL
Name/Directory	RMI Registry	Win Registry	Name Service
Language	Java	C++, Java, VB	C++, Java, Ada
Pro & Con	bind to Java, call blocking, scalability issue	tightly bind to Windows platform, rich tool set	language/platform- independent, fewer tools
Cost	free	MS charges you	open source

中间件体系架构

CORBA Basic Architecture

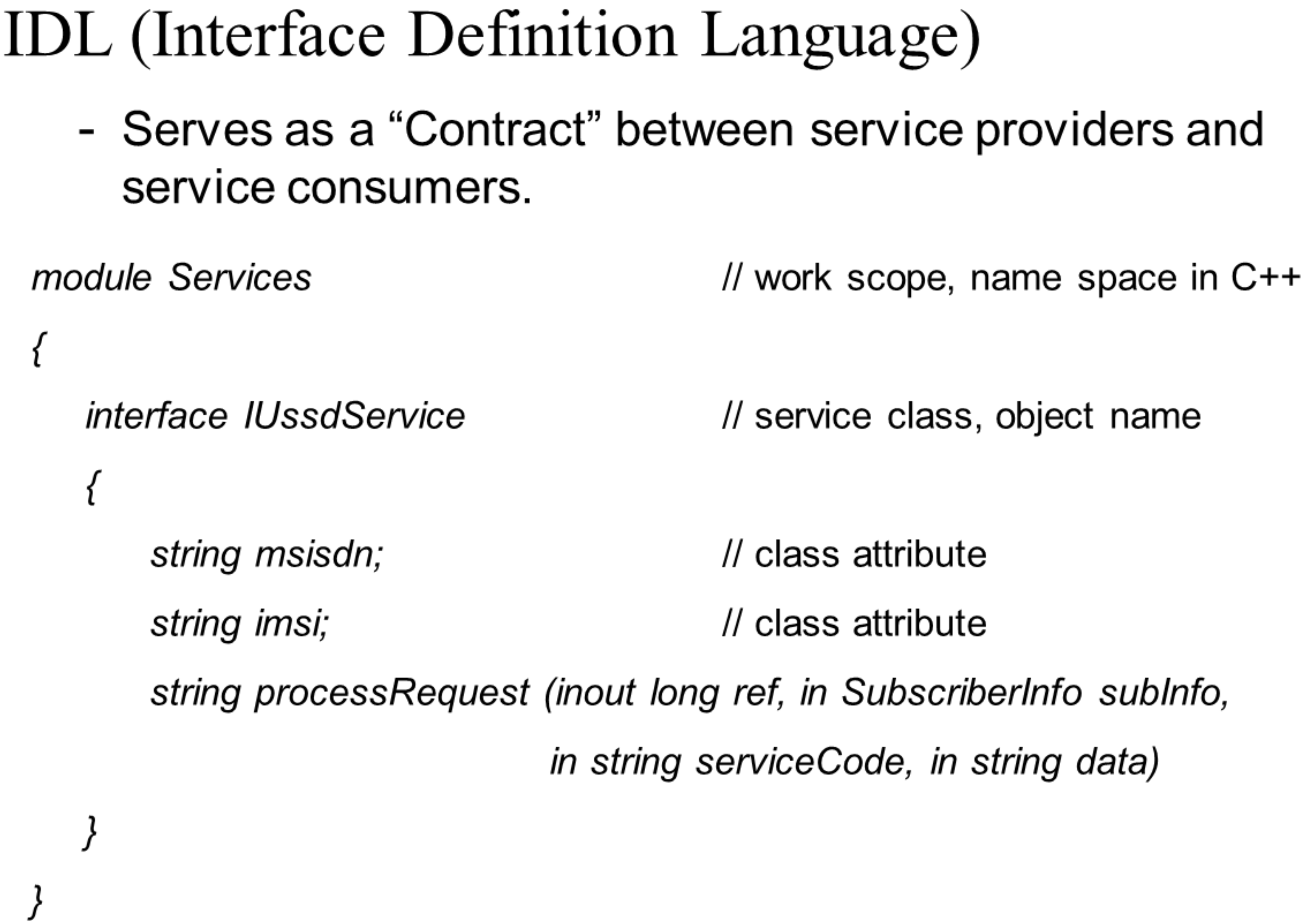


CORBA Layered Model



CORBA Data Flow

中间件体系架构



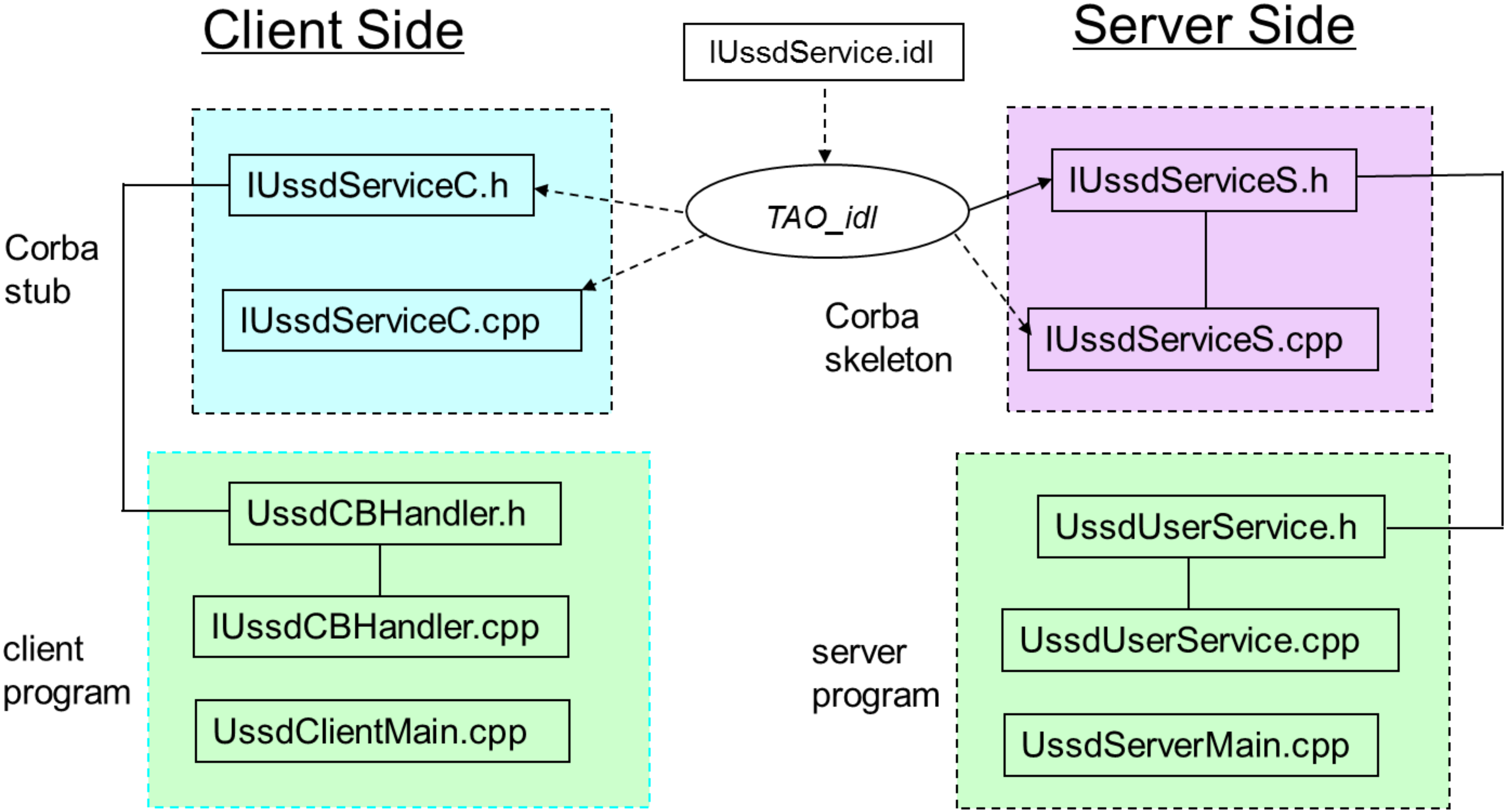
IDL (Interface Definition Language)

- Serves as a “Contract” between service providers and service consumers.

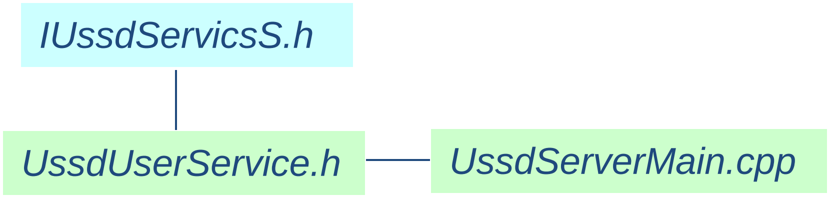
```
module Services                                   // work scope, name space in C++
{
    interface IUssdService                        // service class, object name
    {
        string msisdn;                            // class attribute
        string imsi;                              // class attribute
        string processRequest (inout long ref, in SubscriberInfo subInfo,
                               in string serviceCode, in string data)
    }
}
```

中间件体系架构

CORBA Implementation



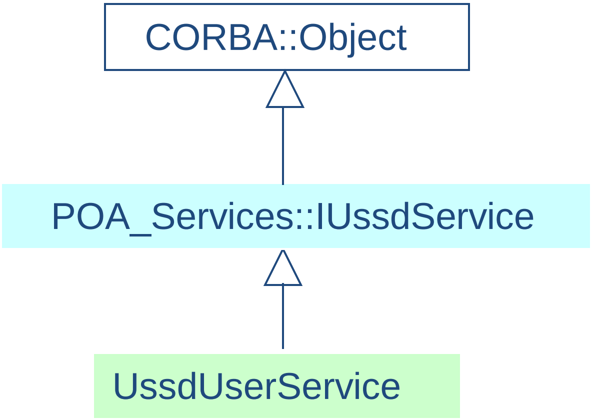
中间件体系架构



IUssdServicsS.h

UssdUserService.h

UssdServerMain.cpp



CORBA::Object

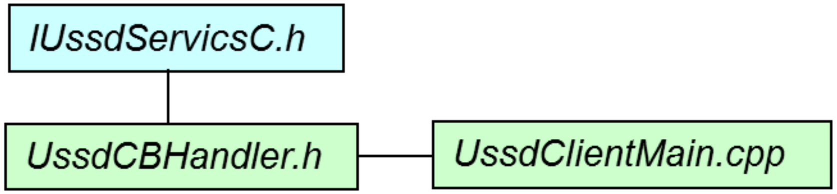
POA_Services::IUssdService

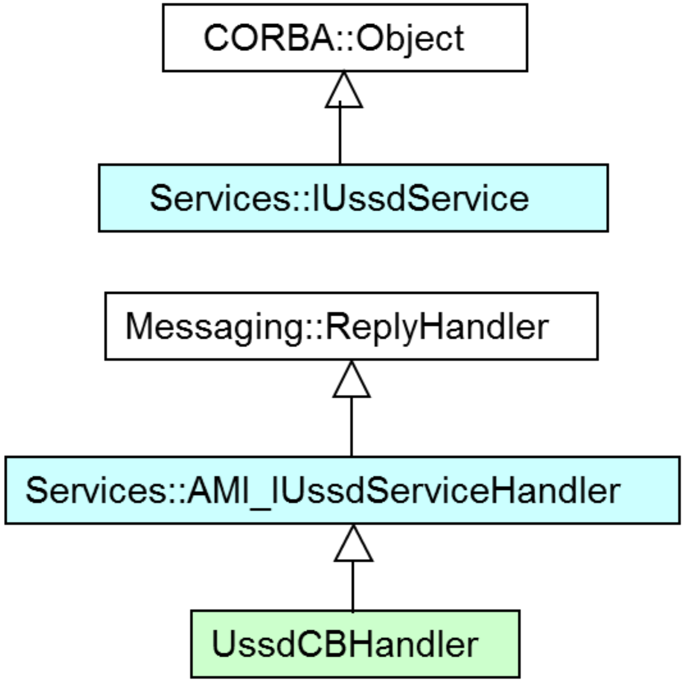
UssdUserService

```
char* UssdUserService::processRequest (::CORBA::Long& ref,
                                        const & subscrInfo,
                                        const char* serviceCode,
                                        const char* data)
        throw(CORBA::SystemException)
{
     // provide service
     return (CORBA::string_dup(returnStr.c_str()));
}

main (int argc, argv*[])

{  // initialize client side ORB and POA …

   // register service with naming service

   obj=orbRef->resolve_initial_references("NameService");

   rootNC = CosNaming::NamingContext::_narrow(obj.in());

   // create servant object
   Services::IUssdService_var servant = servantObj._this();
   // publish service (use rebind to allow stop and start again)
   rootNC->rebind(“MyService”, servant.in());
   // run server ORB
   orbRef->run();
   // upon finishing, delete ORB
   orbRef->destroy();
}
```

中间件体系架构

Corba Client





```
main (int argc, argv*[])
{  // initialize client side ORB and POA
   orbRef = CORBA::ORB_init(argc, argv);
   orbRef->resolve_initial_references("RootPOA");
   poaMgr->activate();
   // resolve name service
   obj=orbRef->resolve_initial_references("NameService");
   rootNC=CosNaming::NamingContext::_narrow(obj2.in());
   // retrieve object reference from naming service
   servName[0].id   = CORBA::string_dup("MyService");
   servName[0].kind = CORBA::string_dup("");
   objRef = rootNC->resolve(servName);
   // create callback handler
   UssdCBHandler cbHandler;
   Services::AMI_IUssdServiceHandler_var cbRef = &cbHandler;
   // make AMI call to request service
   ObjRef->sendc_processRequest(cbRef.in(), reference, subInfo,
                                serviceCode, serviceName);
   // run client ORB
   orbRef->perform_work();
    // upon finishing, delete ORB
   orbRef->destroy();
}
```

Directory/Name Service

Directory Service

A software application that organizes and stores information and provides look-up service.

Name Service --- service publish/service discovery

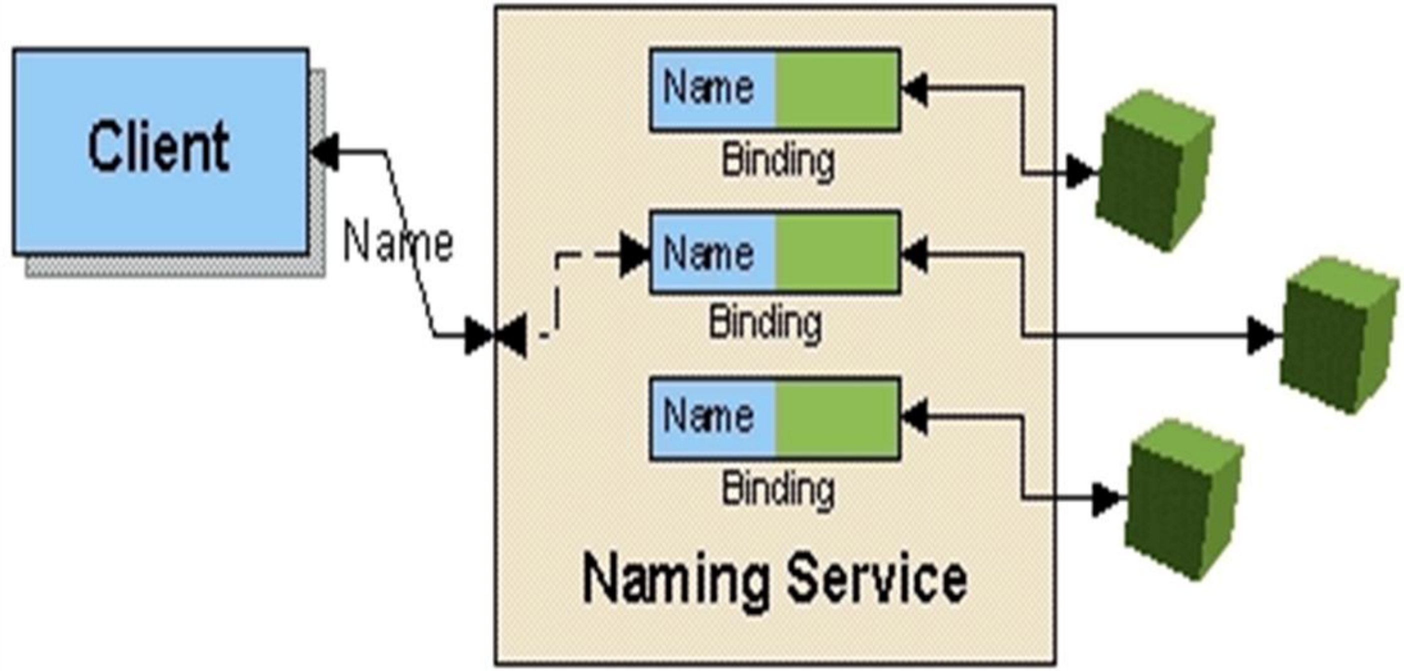
A simple directory service that maps the name of service to the service handle (object reference, codeword, location).

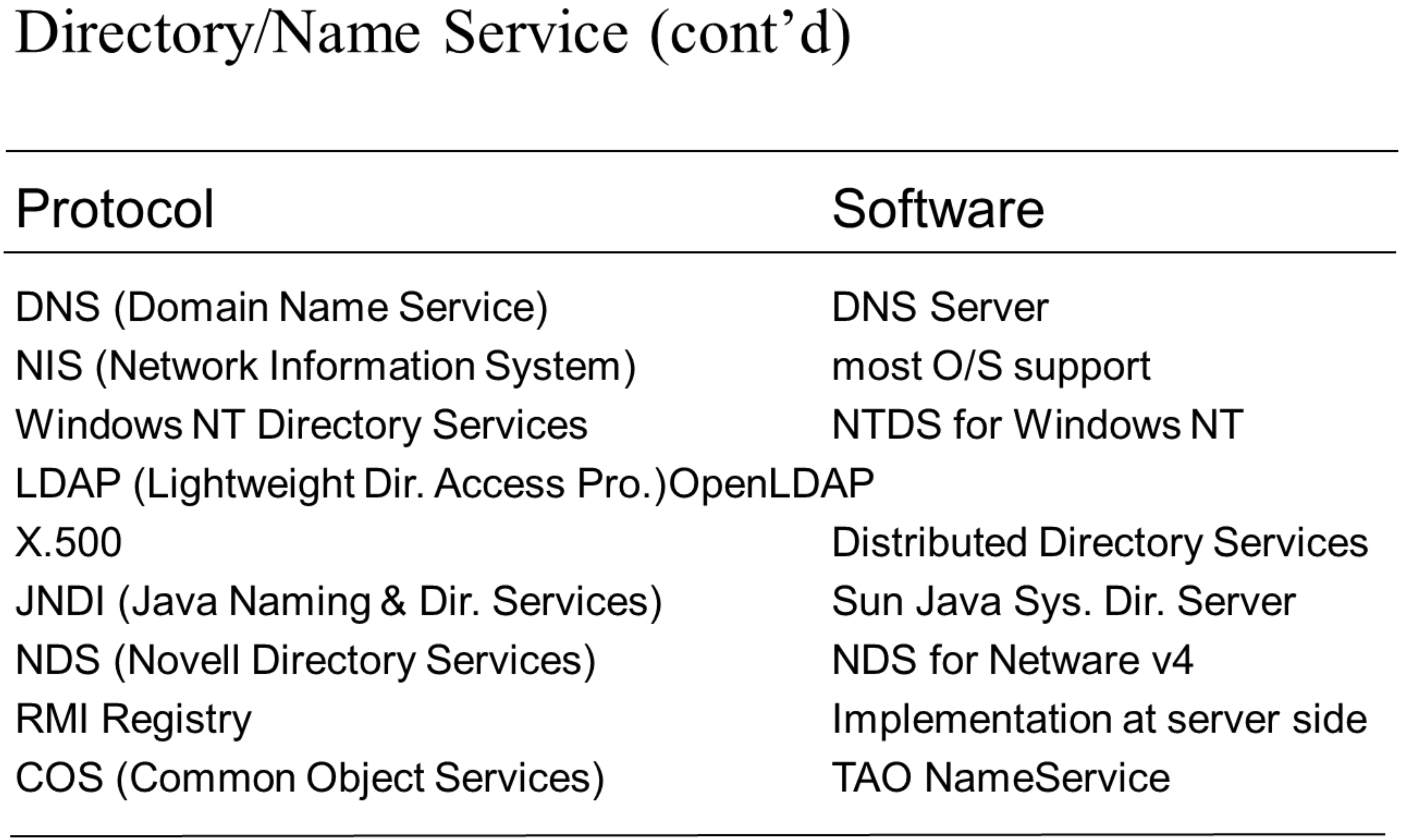
Protocol and Software Implementation

Each type of naming service has protocol/spec to describe its schema, and a software implementation to deliver the service.

中间件体系架构

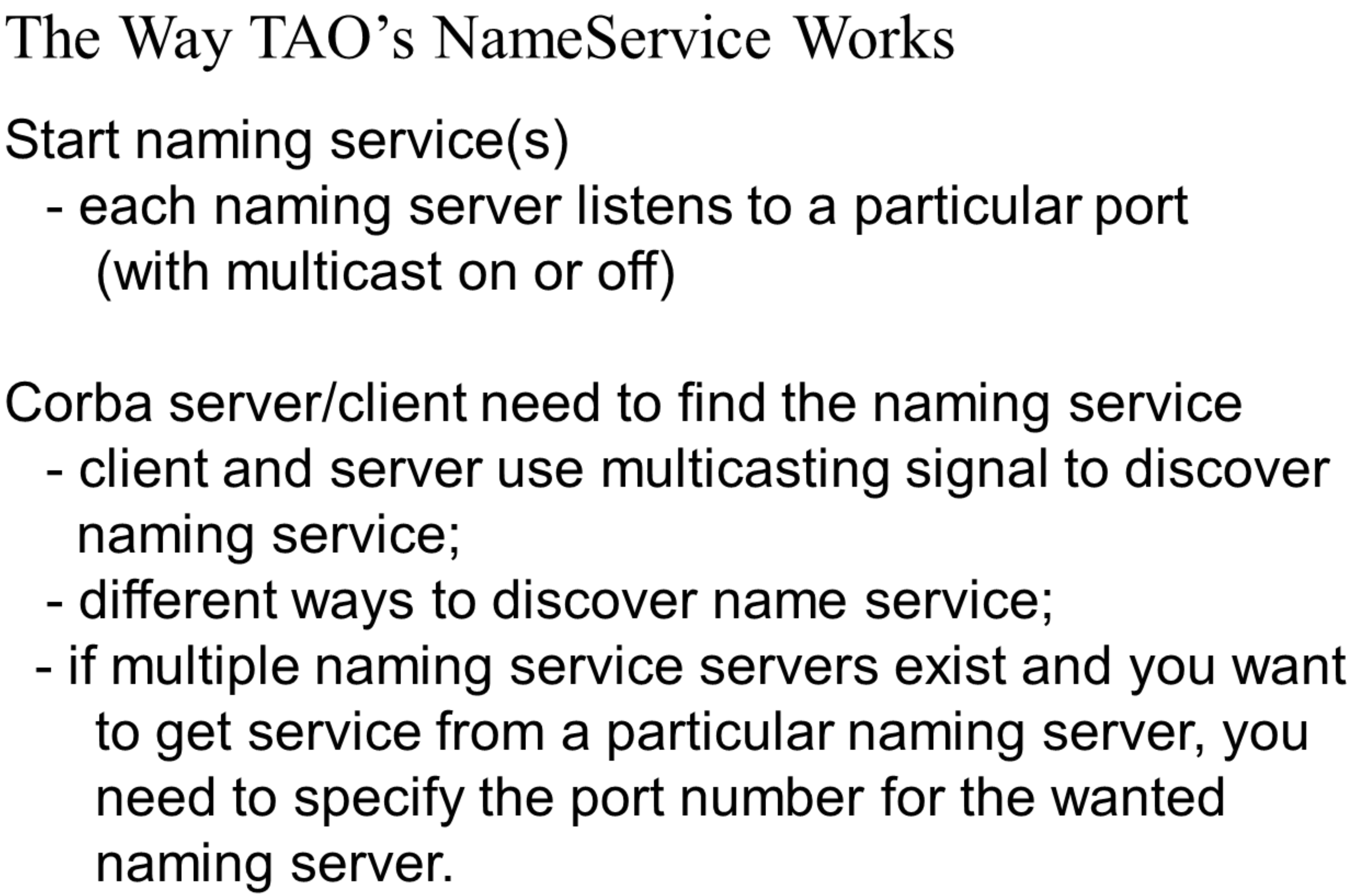
Directory/Name Service (cont’d)





Directory/Name Service (cont’d)

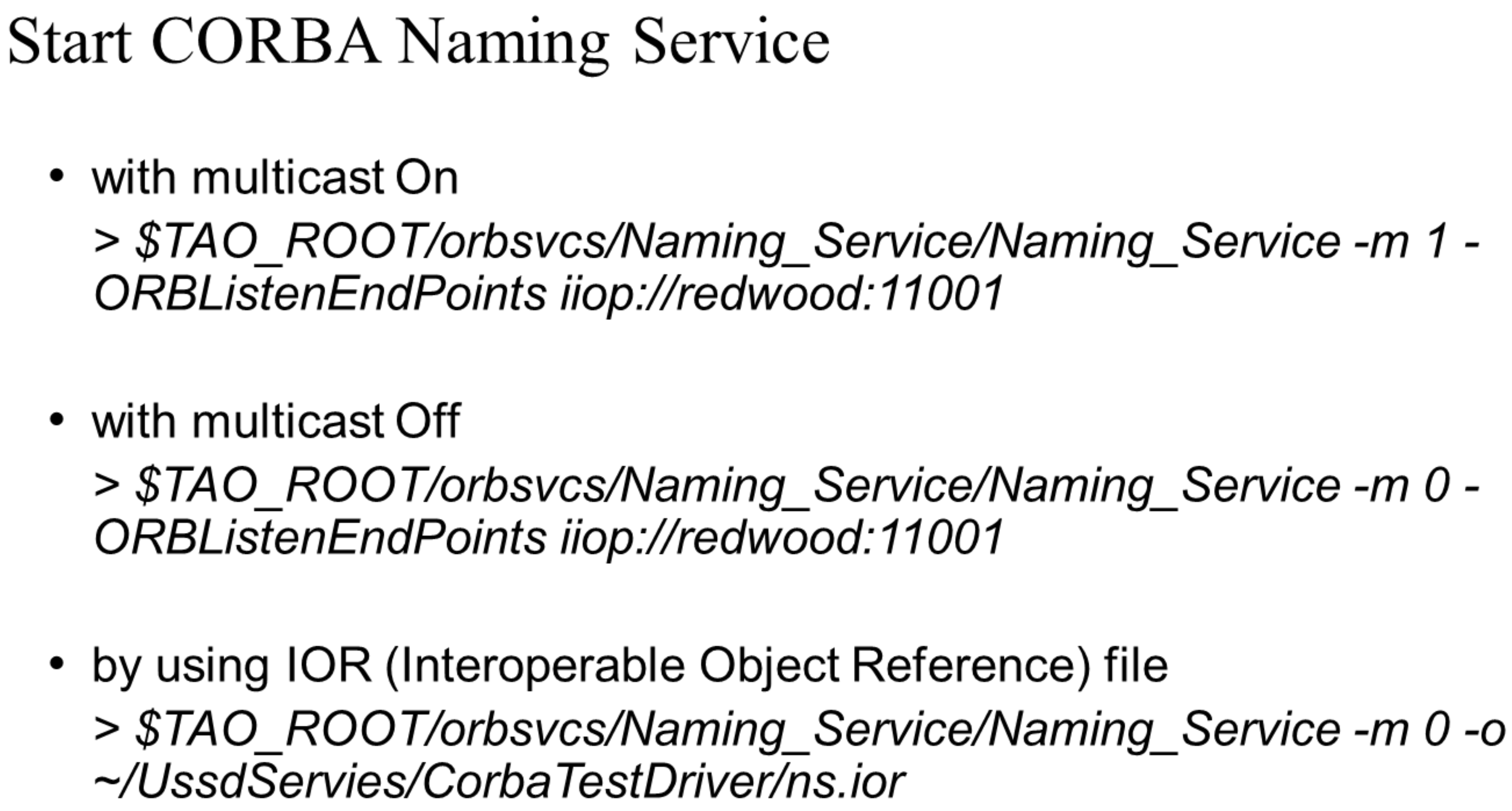
Protocol	Software
DNS (Domain Name Service)	DNS Server
NIS (Network Information System)	most O/S support
Windows NT Directory Services	NTDS for Windows NT
LDAP (Lightweight Dir. Access Pro.)	OpenLDAP
X.500	Distributed Directory Services
JNDI (Java Naming & Dir. Services)	Sun Java Sys. Dir. Server
NDS (Novell Directory Services)	NDS for Netware v4
RMI Registry	Implementation at server side
COS (Common Object Services)	TAO NameService



The Way TAO's NameService Works

Start naming service(s)
- each naming server listens to a particular port (with multicast on or off)

Corba server/client need to find the naming service
- client and server use multicasting signal to discover naming service;
- different ways to discover name service;
- if multiple naming service servers exist and you want to get service from a particular naming server, you need to specify the port number for the wanted naming server.



Start CORBA Naming Service

- with multicast On

```
> $TAO_ROOT/orbsvcs/Naming_Service/Naming_Service -m 1 -ORBListenEndPoints iiop://redwood:11001
```

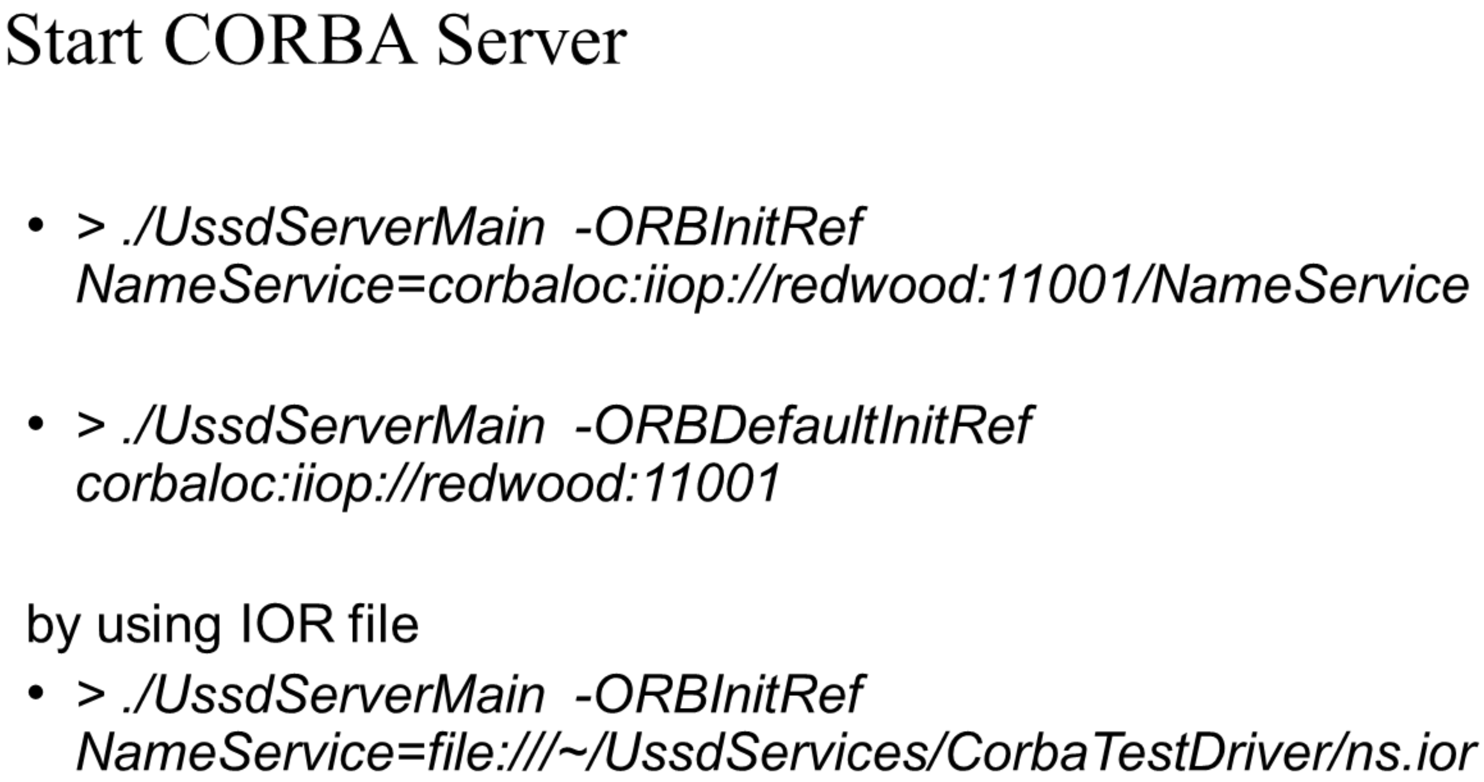
- with multicast Off

```
> $TAO_ROOT/orbsvcs/Naming_Service/Naming_Service -m 0 -ORBListenEndPoints iiop://redwood:11001
```

- by using IOR (Interoperable Object Reference) file

```
> $TAO_ROOT/orbsvcs/Naming_Service/Naming_Service -m 0 -o ~/UssdServies/CorbaTestDriver/ns.ior
```

中间件体系架构



Start CORBA Server

-
  ```
  > ./UssdServerMain  -ORBInitRef NameService=corbaloc:iiop://redwood:11001/NameService
  ```

-
  ```
  > ./UssdServerMain  -ORBDefaultInitRef corbaloc:iiop://redwood:11001
  ```

by using IOR file

-
  ```
  > ./UssdServerMain  -ORBInitRef NameService=file:///~/UssdServices/CorbaTestDriver/ns.ior
  ```

Start CORBA Client

- Put the follow line in “Hlrservices.cfg” file.

  ```
  ORB  -ORBInitRef  NameService=corbaloc:iiop://redwood:11001/NameService
  ```

  (for IOR file)

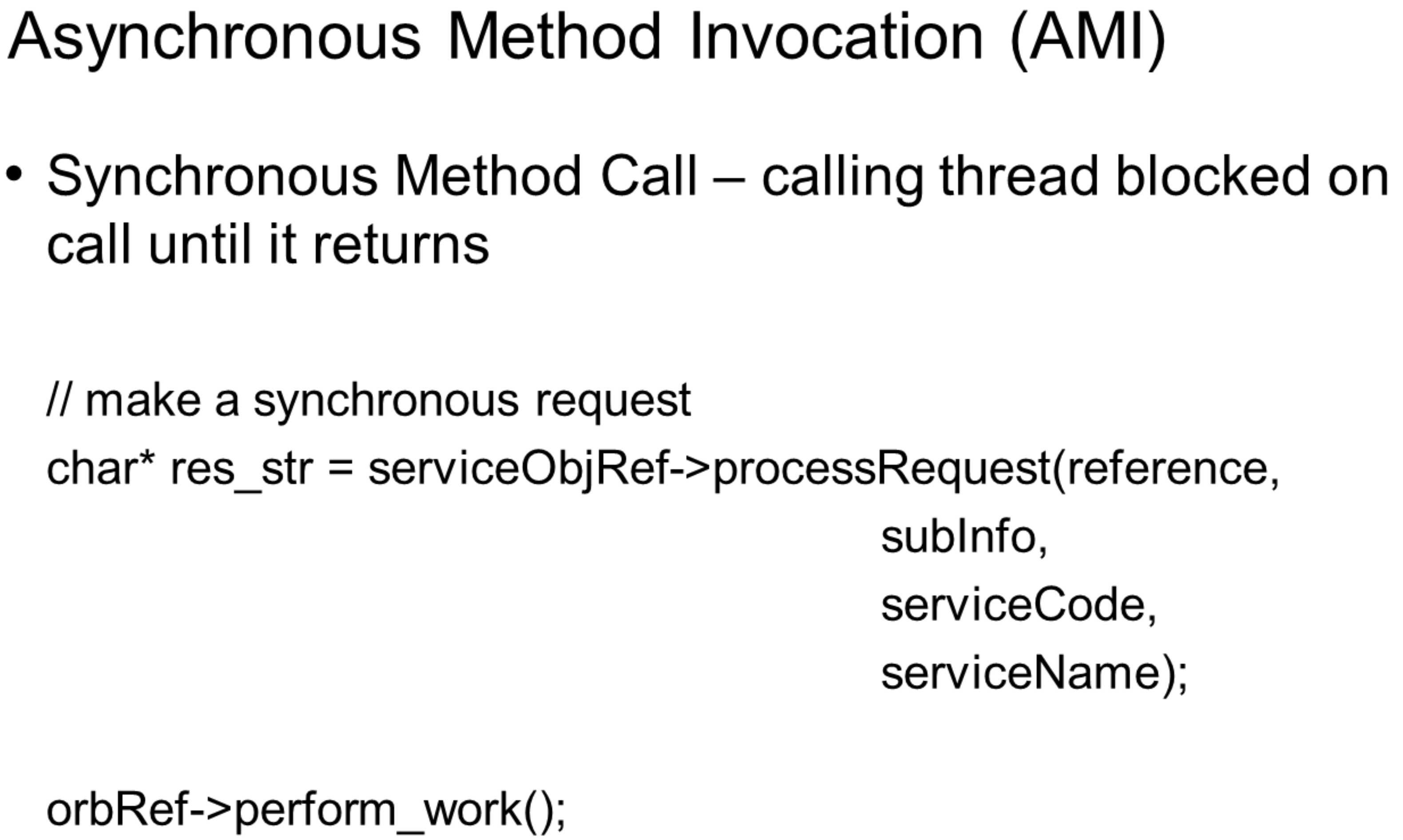
  ```
  ORB  -ORBInitRef  NameService=file://filepath/ns.ior
  ```

- Set NameServiceIOR environment variable

  ```
  > export NameServiceIOR=corbaloc:iiop:redwood:11001/NameService
  ```

- Start a standalone client

  ```
  > ./UssdClientMain  -ORBInitRef
  NameService=corbaloc:iiop://redwood:11001/NameService
  ```



Asynchronous Method Invocation (AMI)

- Synchronous Method Call – calling thread blocked on call until it returns

```
// make a synchronous request
char* res_str = serviceObjRef->processRequest(reference,
                                      subInfo,
                                      serviceCode,
                                      serviceName);

orbRef->perform_work();
```

中间件体系架构

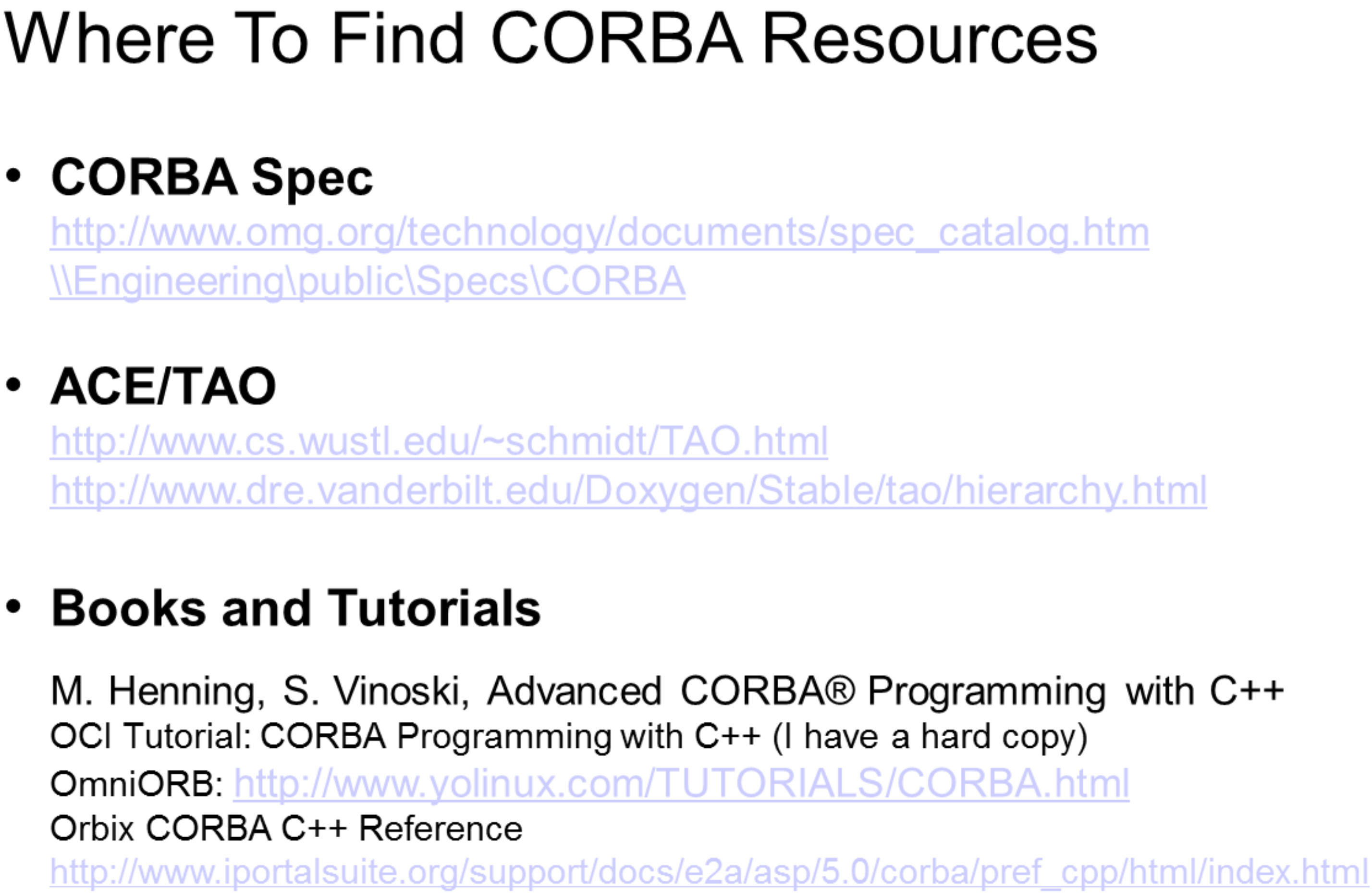
- AMI call – provide a mechanism to run service request call in multithread mode and do not block the calling thread

- create a Callback handler class

```
class UssdCBHandler  : public Services::AMI_IUssdServiceHandler
{
    public:
            virtual  void  handleCallback(const  int reference,  const  char*  result);
            virtual  void  processRequest(const  char*  ami_return_val,  CORBA::Long  ref)
            virtual  void
processRequest_excep(Services::AMI_IUssdServiceExceptionHolder*,  …)
};
```

中间件体系架构

```
UssdCBHandler cbHandler;
Services::AMI_IUssdServiceHandler_var cbHandlerRef = &cbHandler;

// make an AMI request
serviceObjRef -> sendc_processRequest (cbHandlerRef.in(),
                                       reference,
                                       subInfo,
                                       serviceCode,
                                       serviceName);
// start ORB in a separate thead
if ( fork() == 0 )        // in a child process
{
   // let ORB do its work
   orbRef->perform_work();

   // when finish, exits
   exit(0);
}
```



Where To Find CORBA Resources

- **CORBA Spec**
  http://www.omg.org/technology/documents/spec_catalog.htm
  \\Engineering\public\Specs\CORBA
- **ACE/TAO**
  http://www.cs.wustl.edu/~schmidt/TAO.html
  http://www.dre.vanderbilt.edu/Doxygen/Stable/tao/hierarchy.html
- **Books and Tutorials**

  M. Henning, S. Vinoski, Advanced CORBA® Programming with C++
  OCI Tutorial: CORBA Programming with C++ (I have a hard copy)
  OmniORB: http://www.yolinux.com/TUTORIALS/CORBA.html
  Orbix CORBA C++ Reference
  http://www.iportalsuite.org/support/docs/e2a/asp/5.0/corba/pref_cpp/html/index.html

中间件体系架构

End of Lecture

谢谢！