Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

Training

ANSAwise - The CORBA Object Management Architecture

Chris Mayers

Abstract

Having been introduced to the CORBA vision and architecture, organizations will wish to find out more about the structure of the CORBA architecture, its components, and interfaces (both internal and external).

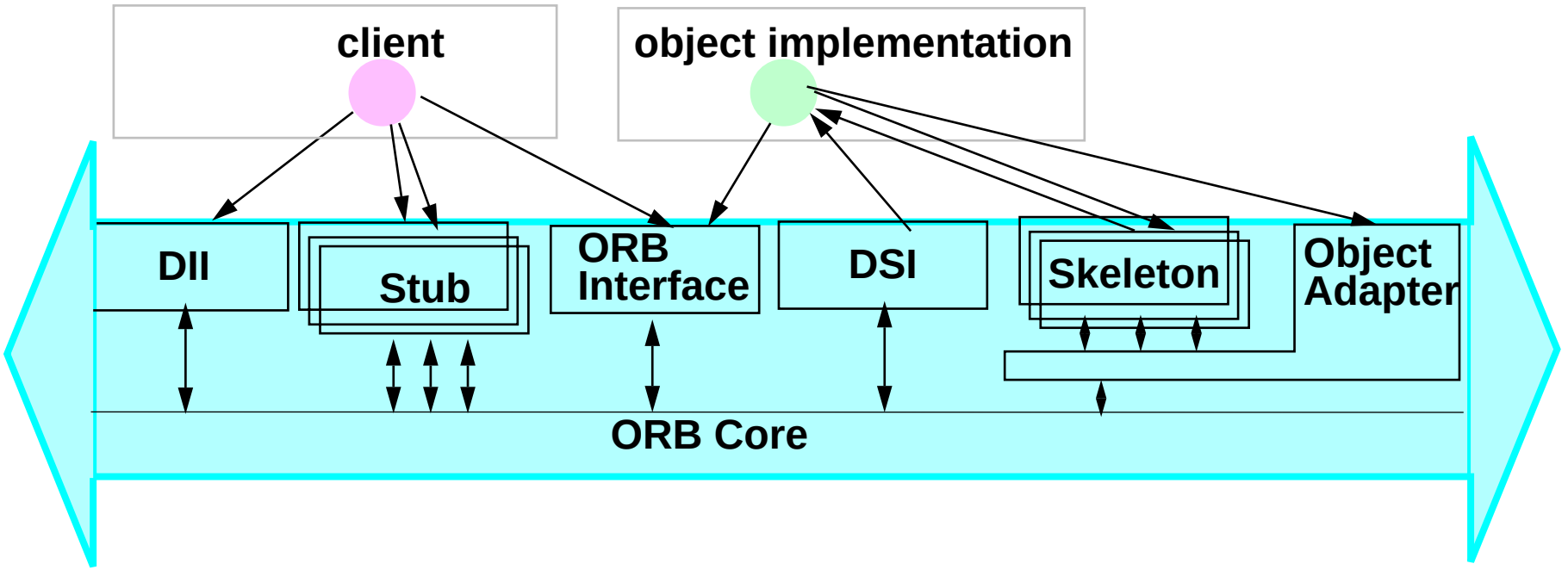
This module of the ANSAwise training programme examines the CORBA Object Management Architecture (in particular, the Common Object Request Broker Architecture within it). It explains the structure of an ORB, and the evolution of the ORB specification.

IDL is not discussed here.

APM.1345.03 **Approved** 13th May 1996
Briefing Note

Distribution:
Supersedes:
Superseded by:

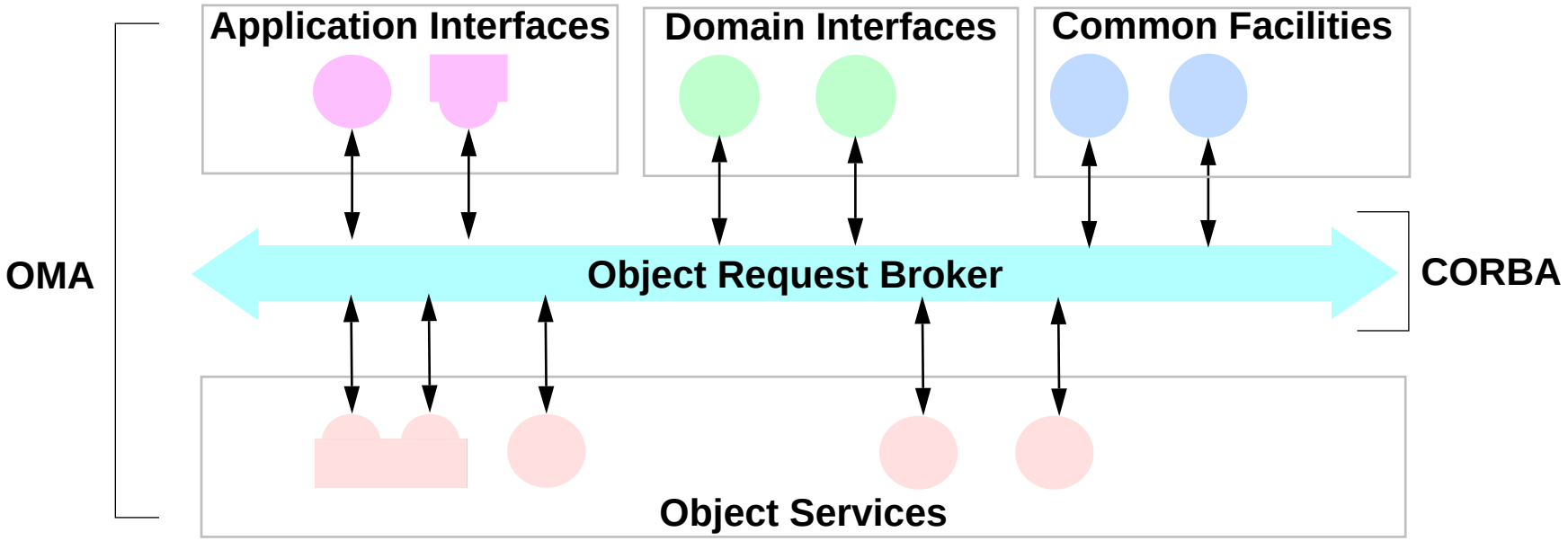
The CORBA Object Management Architecture



In this session

- **Explain the CORBA Object Management Architecture**
- **Explain the Core Object Model**
- **Explain how CORBA servers are implemented**

The Object Management Architecture



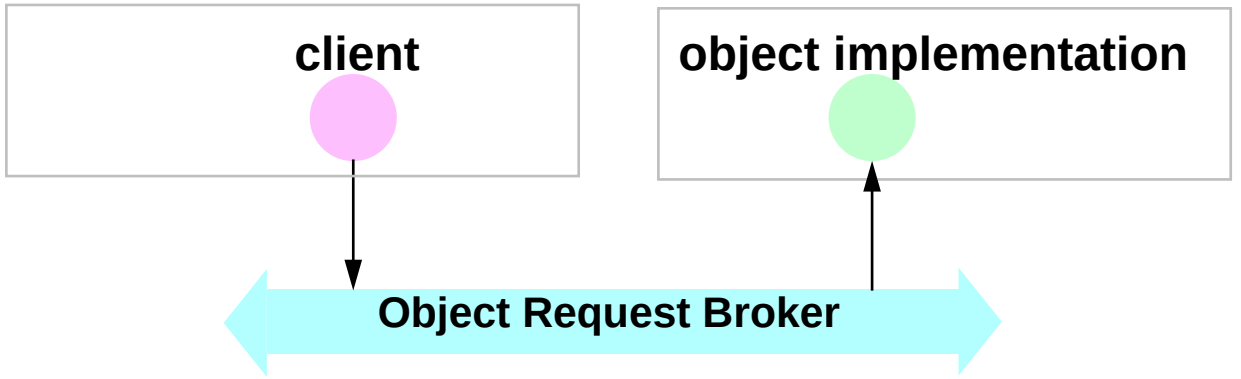
- Consists of the Object Request Broker (ORB), plus objects
  - Objects are Object Services, Common Facilities, Domain Interfaces, or Application Interfaces

The scope of the Object Request Broker

- The document CORBA specifies CORBA 2.0.
- Core section covers
  - the Object Model: object semantics
  - the Common Object Request Architecture: how ORBs are structured
  - the CORBA Interface Definition Language (IDL): syntax and semantics
  - the Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI)
  - the fundamental services
- Interoperability section covers interaction between ORBs
- Other sections cover the C, C++, and Smalltalk Language Mappings

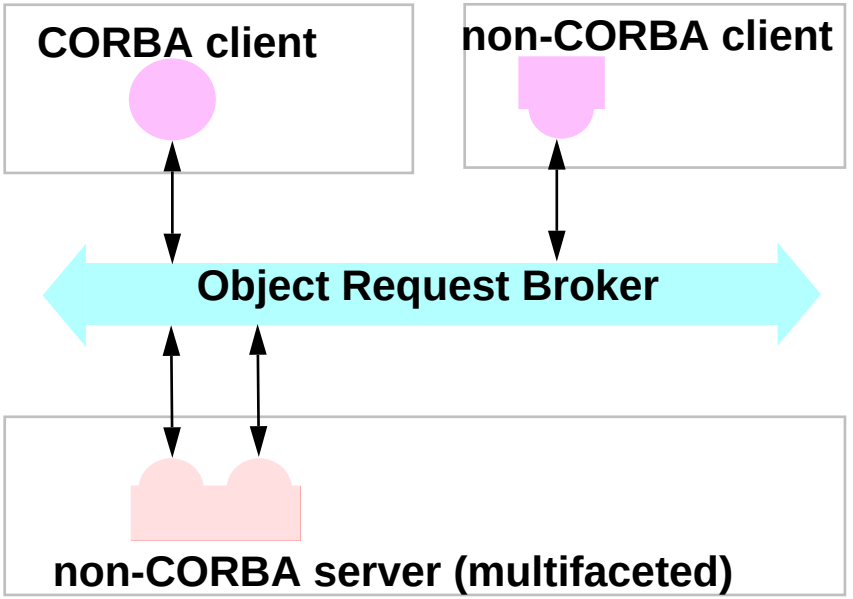
The Object Request Broker (ORB)

- **Objects request the services of other objects via the ORB**
  - **the ORB is responsible for locating the object implementation, and all the communications mechanisms that support the request**
  - **client and object implementation can be written in different languages, and run on different types of machines**



ORB Access to Existing Applications

- **The ORB can wrap existing non-OO applications...**
  - **... clients or servers**

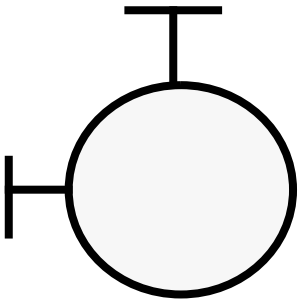


The Core Object Model

- **The key concepts are**
  - **Objects**
  - **Requests**
  - **Types**
  - **Interfaces**
  - **Operations**
  - **Attributes**

Encapsulation for objects

- **Objects are encapsulated**
  - **every object provides a service via interfaces**
  - **the interface is public; the implementation is private and hidden**
  - **encapsulation forms a boundary; the only access to an object is via its interfaces**

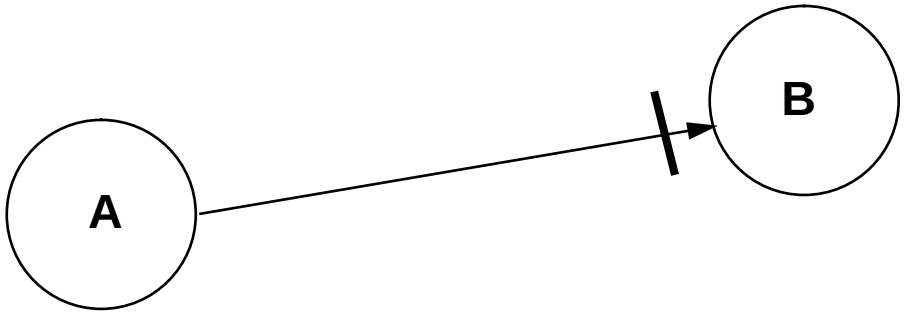


Encapsulation in programming languages

- Access only via explicit interfaces
  - C++ class
  - Smalltalk class
  - Ada package
- Hide the implementation
  - the data structures
  - the algorithms
  - the state
- Keep the data with the code

Encapsulation in distributed system

- In a distributed system, encapsulation must be strict
  - no shared state between objects
  - no ‘back doors’
- Objects A and B could be on the same machine today...
- ... in different countries tomorrow

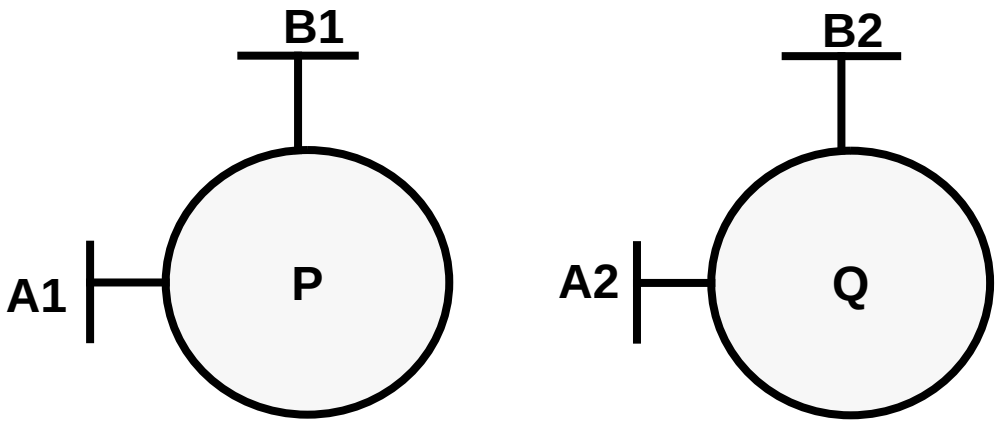


Polymorphism

- **The same operation can apply to different types of object**
  - **you can print a text file, a structured document, and a bitmap**
- **Each object can be expected to “know how to print itself”**
  - **it has a standard interface for printing**
  - **the object determines *how* to “print itself”**

Inheritance for objects

- Consider two similar objects



- They can have similar interfaces A and B
- They can have similar implementations P and Q
- ...these are quite separate properties

Views of inheritance

- In non-distributed object systems we may be interested in similar implementations
  - sharing or reusing some of the code...
  - ... for example the code for the A interface
- In a distributed system this is not relevant
  - sharing any code between distributed objects is impossible
- The important thing is compatibility of interfaces
  - type conformance (*substitutability)*
- Some programming languages try to use ‘class’ for both ideas

CORBA Objects

- An object is an identifiable encapsulated entity that provides one or more services that can be requested by a client
- An object can have one or more ‘handles’
  - these are known as *object references*
- Objects can be created and destroyed
  - from the client’s point of view, this happens automatically...
  - ... there is no special mechanism for creating and destroying objects
- An object reference is created when the object is created
  - it always refers to that same object...
  - ... object references are not ‘movable pointers’

CORBA Interfaces

- An interface is a description of a set of possible operations that a client may request
- Interfaces are specified in CORBA IDL
- Some special objects are implemented directly by the ORB
  - they have ordinary CORBA interfaces
  - ... but are specially handled by the ORB
  - ... these are called ‘pseudo objects’

CORBA Operations

- An operation denotes a service that can be requested
  - very roughly, a function prototype or method specification
- An operation has an operation identifier (a name), and a signature
- The signature specifies
  - (formal) parameters, with modes *in*, *out*, or *inout*
- An operation may return a single result
  - multiple results are not possible...
  - ... this is a concession to the language mappings; most programming languages do not support multiple results

CORBA Types

- CORBA supports the basic values you would expect...
  - ...Short, Long, Float, Boolean,...
- CORBA supports constructed values
  - ... Struct, Sequence, Union, and Array
- We discuss this in detail when covering IDL
- Notice one type that is deliberately not provided
  - no pointers!

CORBA Attributes

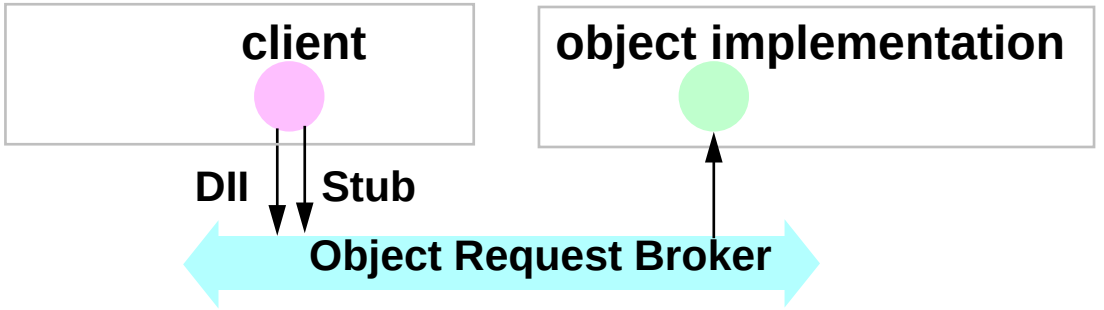
- An interface can have attributes
- Each attribute is equivalent to a pair of accessor functions
  - a get function to read the value
  - a set function to write the value
- The value can be a basic or constructed value (e.g. a sequence)
- A read-only attribute has only the get function
- The attribute accessor functions are not the same as operations
  - they may have a more convenient language mapping

CORBA Requests

- Clients invoke services by issuing requests for operations on objects
- A request consists of
  - an operation name
  - a target object (identified by an object reference)
  - a list of zero or more (actual) parameters
  - a request context, providing additional information about the request

Two forms of request

- Clients can make requests via IDL Stubs, or via the Dynamic Invocation Interface (DII)



- There is one IDL Stub per interface
  - but only one DII, which supports all interfaces
- Very roughly speaking
  - requests via IDL stubs are ‘compiled’
  - requests via the DII are ‘interpreted’

IDL Stubs or DII?

- The interfaces to IDL Stubs and the DII are the same for all ORBS
  - clients are portable, whether they use IDL Stubs or the DII
- But the interfaces to IDL Stubs and to the DII are very different in form
  - you need to write different source code for the two cases
- Clients can make requests via IDL Stubs for some interfaces, and the DII for others
- Servers support both IDL Stubs and the DII
  - they are not aware of the form of the client request

When you should use the DII

- Application programmers must construct DII parameter lists by hand
- The DII API is large and complex
- There are no checks until run-time
- ... You should therefore only use the DII for writing ORB tools
  - for example, an object browser

Dynamic Skeleton Interface (DSI)

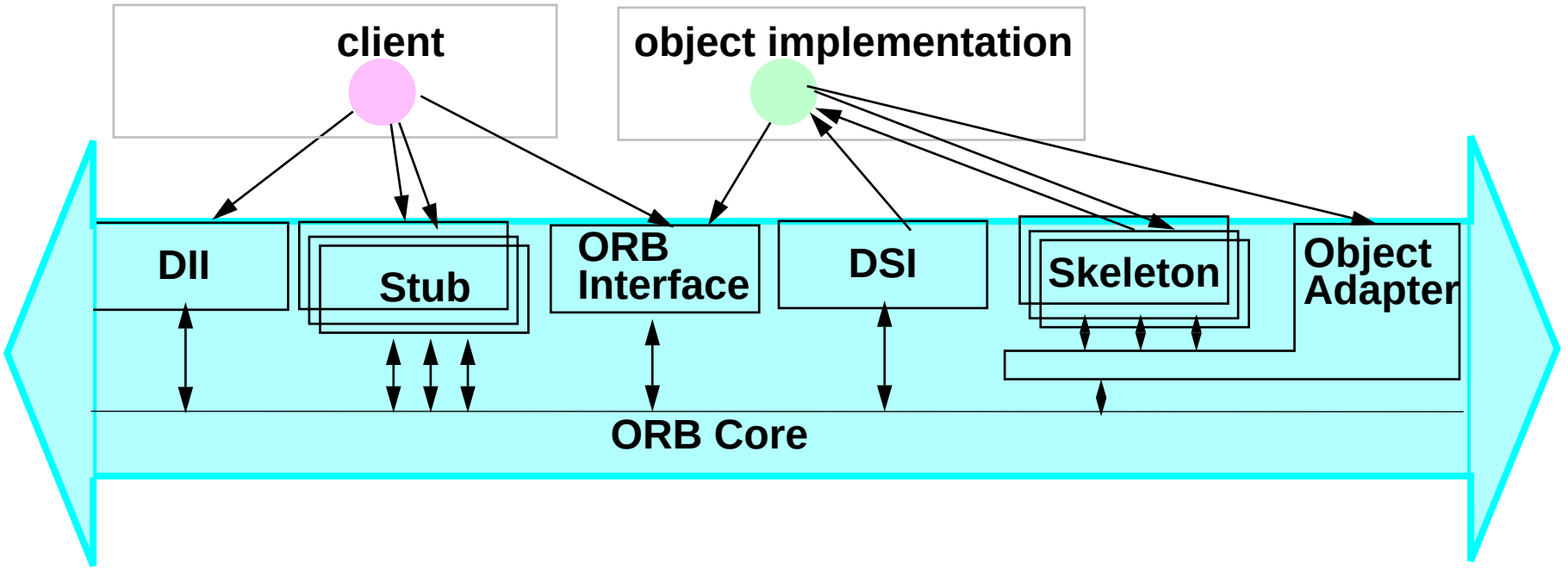
- The server-side equivalent to the DII
- The same considerations apply
- ... You would only use the DSI for writing bridges and gateways

Other ORB Services

- **Interface Repository: retrieval of interface specifications**
- **Implementation Repository: retrieval of implementation information**
- **ORB Interface: operations common to all objects**

Inside the ORB

- **The ORB is not a monolith...**



The structure of an ORB

- One IDL stub and one IDL skeleton per interface
  - automatically generated from IDL
- One DII and one ORB Interface per ORB
- One or more Object Adapters per ORB
- Interfaces with the ORB Core are ORB-dependent

Object Adapters

- **Object implementations require general facilities**
  - **to establish and authenticate object identities**
  - **to create and destroy objects**
  - **to access ORB-dependent facilities**
- **This is done via an Object Adapter**
  - **the object adapter is also used implicitly by skeletons**

Special Object Adaptors

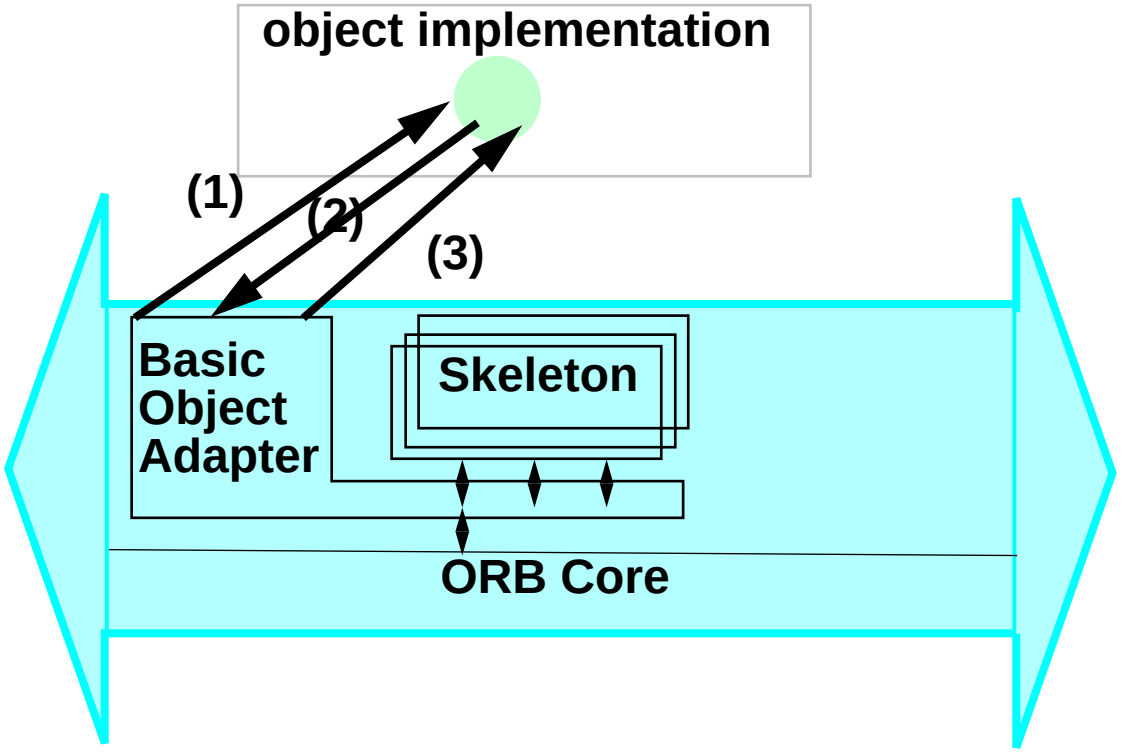
- There is always a Basic Object Adapter; there may be others
  - for example, for legacy systems integration, or for high-performance implementations (as libraries, or as OO databases)
  - the object implementation can choose which Object Adapter to use

Basic Object Adapter

- This has an IDL specification of its own
  - interface BOA
- This is likely to be hard-wired and pre-linked into the ORB

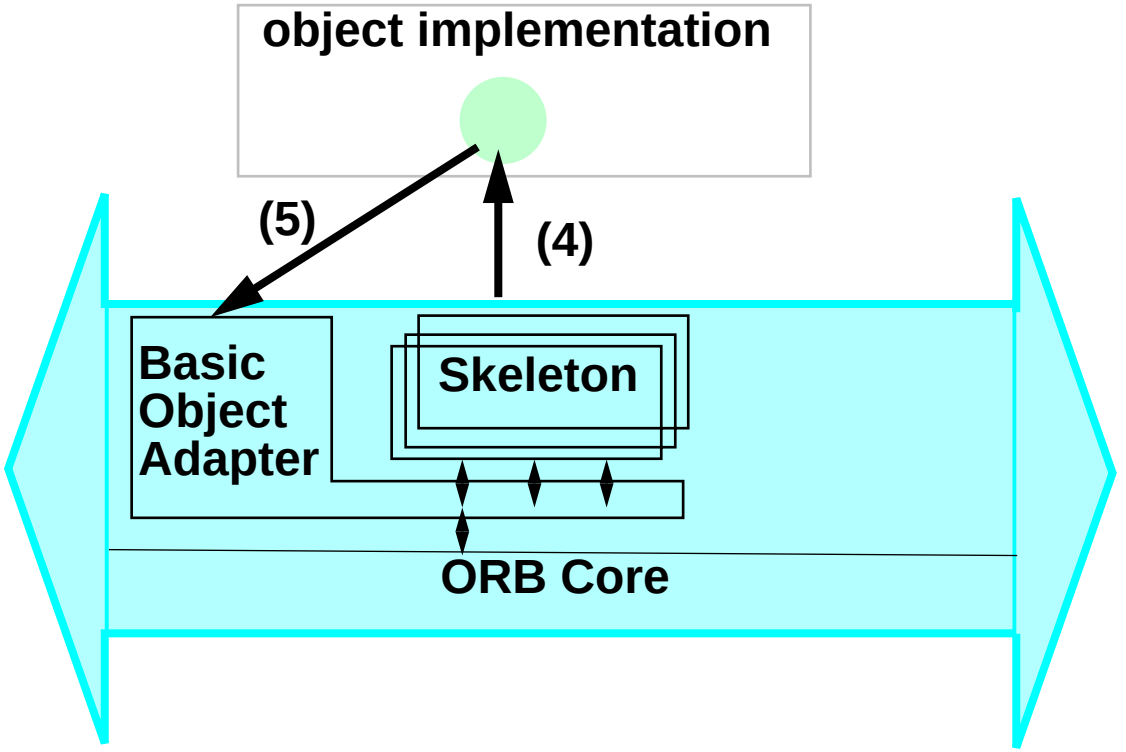
How the Basic Object Adapter works - activation

- **Activate Implementation, Register Implementation, Activate Object...**



How the Basic Object Adapter works - use

- ... Invoke Method, Access BOA Service



Object Implementations

- **Object Implementations do not have to be traditional programs**
  - **they can be scripts, loadable modules, or any other kind of executable software**
- **Object Implementations can be connected to the ORB in different ways**
  - **this is implementation-dependent**

ORB Implementations

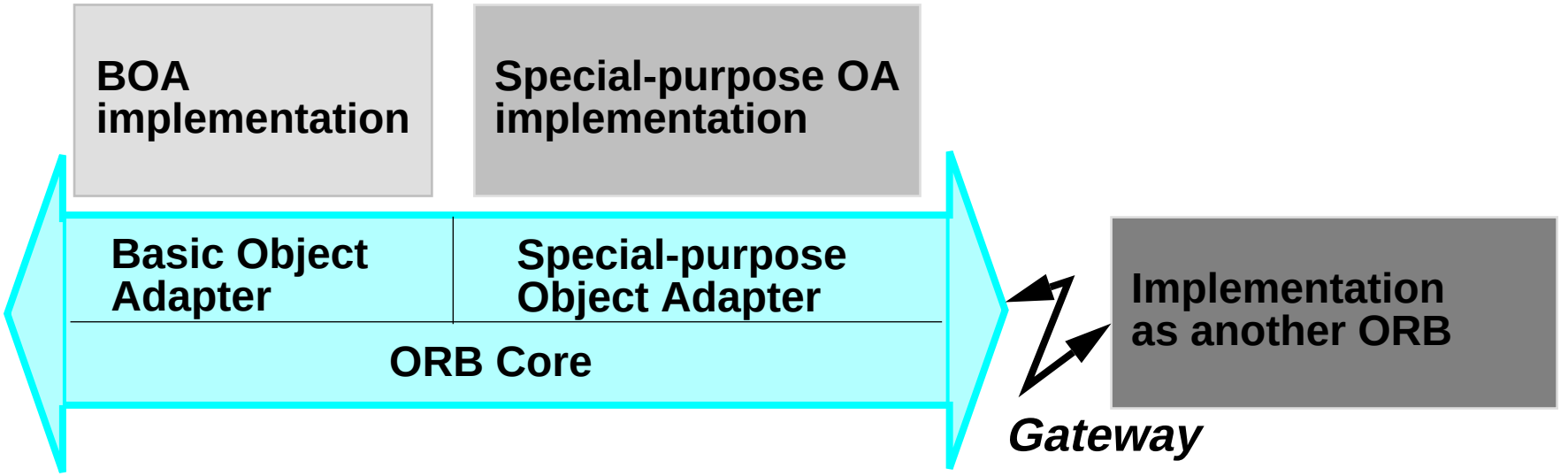
- The CORBA Architecture is intended to allow a variety of ORB implementations, for example
  - static libraries linked into clients and servers
  - dynamic-linked libraries bound to clients and servers at run time
  - as a centralized server
  - built into the underlying operating system
- The ORB implementations affect the way applications are built and installed...
  - ... but applications are still source-code portable and interoperable

Multiple ORB Implementations

- An ORB installation may consist of several interoperating ORBs
  - supplied by different vendors
  - implemented in different ways
- These ORB implementations may use
  - different representations for object references
  - different means for performing invocations
- These differences are transparent to the application
  - the ORB implementations must be capable of resolving these differences automatically

ORB Integration with other object systems

- As well as wrapping non-OO applications, an ORB can integrate with non-CORBA object systems
- This can be done in these ways



Summary

- The ORB is underpinned by a core Object Model
- Clients make requests on object implementations via the ORB
  - via IDL Stubs (preferred), or via the DII
- Object Implementations use an Object Adapter
- For more on this topic
  - see *CORBA* (Object Management Group Inc.)