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

EPFL Course September 1995: CORBA Specification

Mark Madsen

Abstract

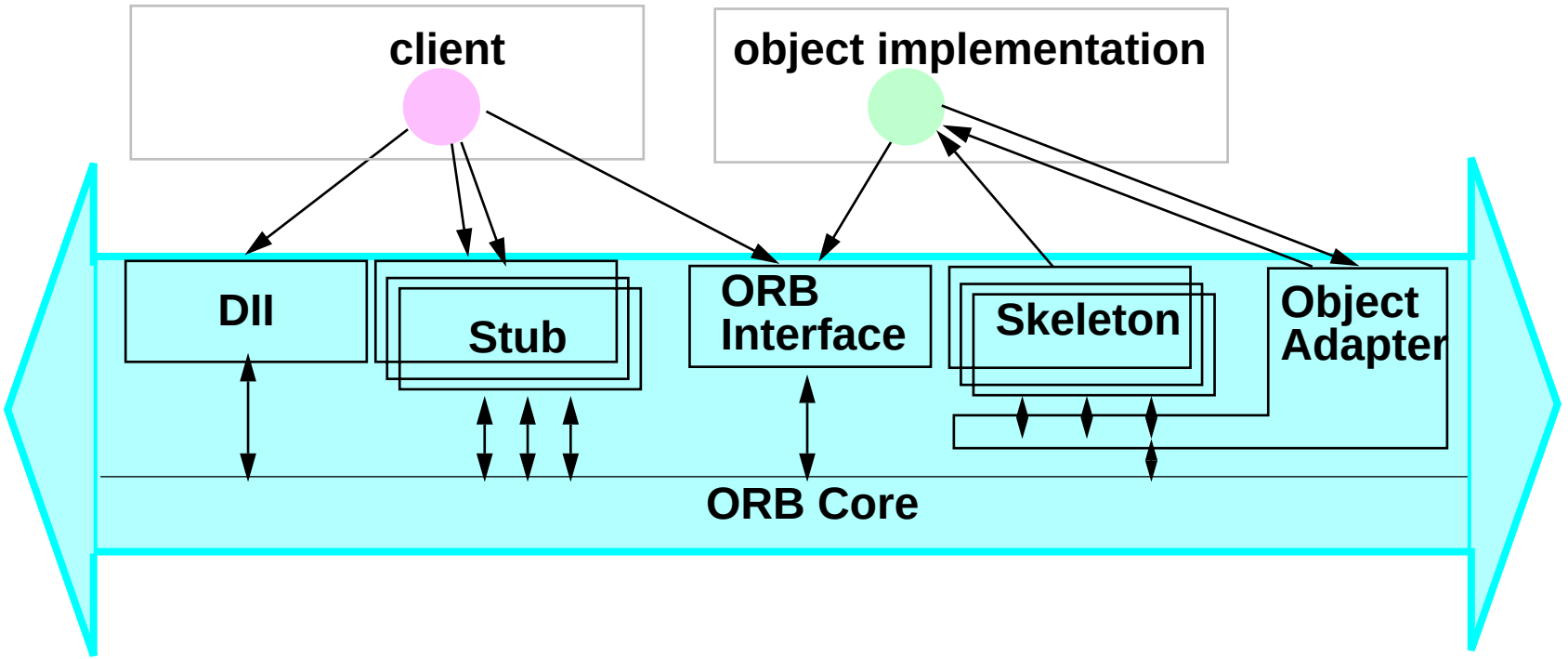
This presentation is based on APM.1345.01 “ANSAwise - The CORBA Object Management Architecture”. (The Appendix is adapted from APM.1534.01 “ANSAwise - Specifying Services in CORBA IDL”.)

It was adapted for presentation as part of Module M3 “Distributed Systems” of the course on Communication Networks given at Ecole Polytechnique Federale de Lausanne in September 1995.

APM.1566.01 **Approved** 8th September 1995
Briefing Note

Distribution:
Supersedes:
Superseded by:

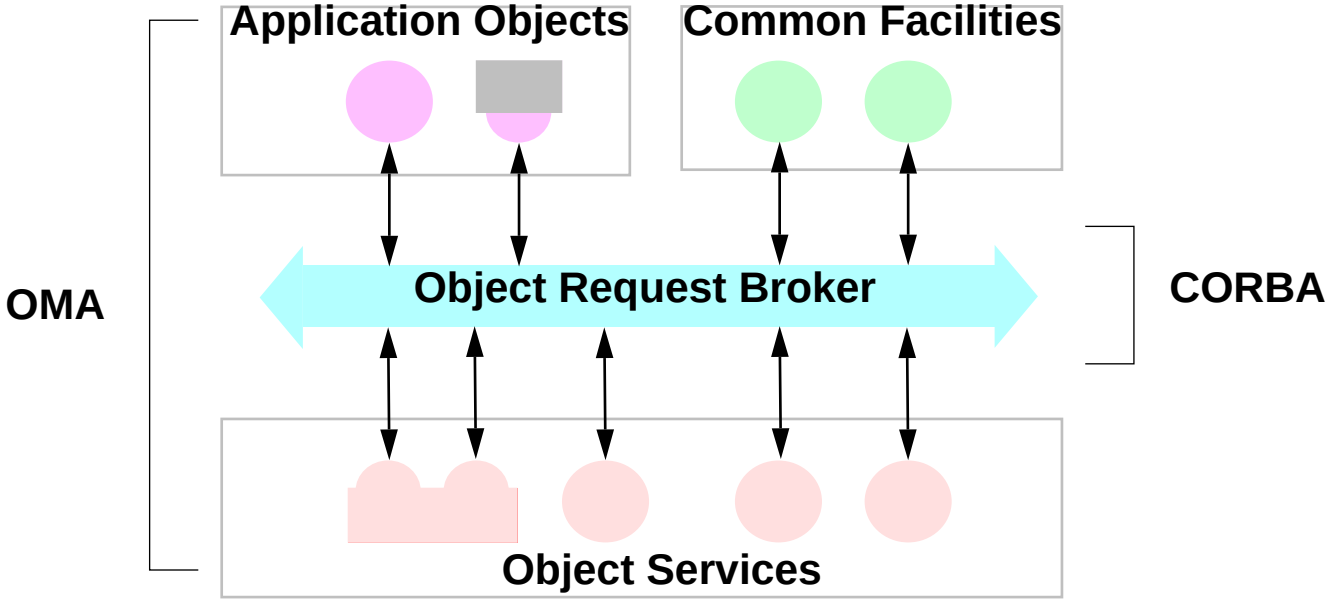
The CORBA Object Management Architecture



In this session

- ***Explain the CORBA Object Management Architecture***
- ***Explain the Core Object Model***
- ***Explain the significance of ongoing developments***
  - **the differences between ORB 1.1 and ORB 2.0**

The Object Management Architecture



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

The scope of the Object Request Broker

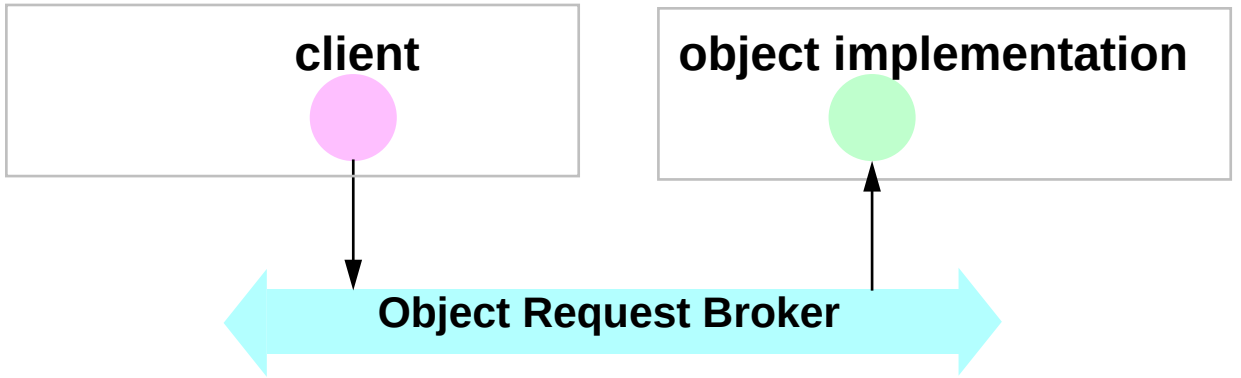
- *This is specified in the following document*
  - The Common Object Request Broker: Architecture and Specification (OMG and X/Open)
- *This document bundles together specifications for*
  - the core Object Model: object semantics
  - the Common Object Request Architecture: how ORBs are structured
  - the CORBA Interface Definition Language (IDL): syntax and semantics
  - the C Language Stub Mapping: how to request operations from C
  - the Dynamic Invocation Interface (DII)
  - the Interface Repository: how to find out what interfaces are present
  - the ORB Interface: operations on object references
  - the Basic Object Adapter: support for object implementations
  - Interoperability: between ORBs

Understanding the Object Request Broker

- *The specifications fall into three groups*
  - the ORB concepts and structure (Object Model, Architecture, and Interoperability)
  - built-in standard object services (Interface Repository, ORB Interface, and Basic Object Adapter)
  - portable programming interfaces for all objects (IDL, DII, and C Language Stub Mapping)
- *In this session we cover mainly the first of these*
  - IDL is covered in a separate session

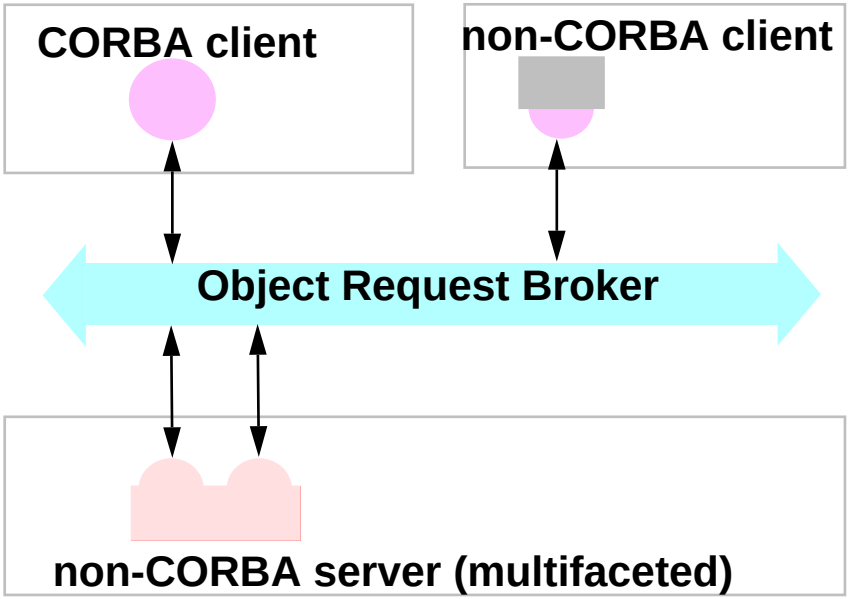
The Object Request Broker (ORB)

- *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



CORBA Services outside the ORB

- *Common Object Services are basic ‘system-level’ services*
  - for example, Events
- *Common Facilities are application-level objects that can be shared between applications*
  - for example, printing
- *Application Objects are application-specific*
  - provided by ISV or end-user

The Core Object Model

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

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’

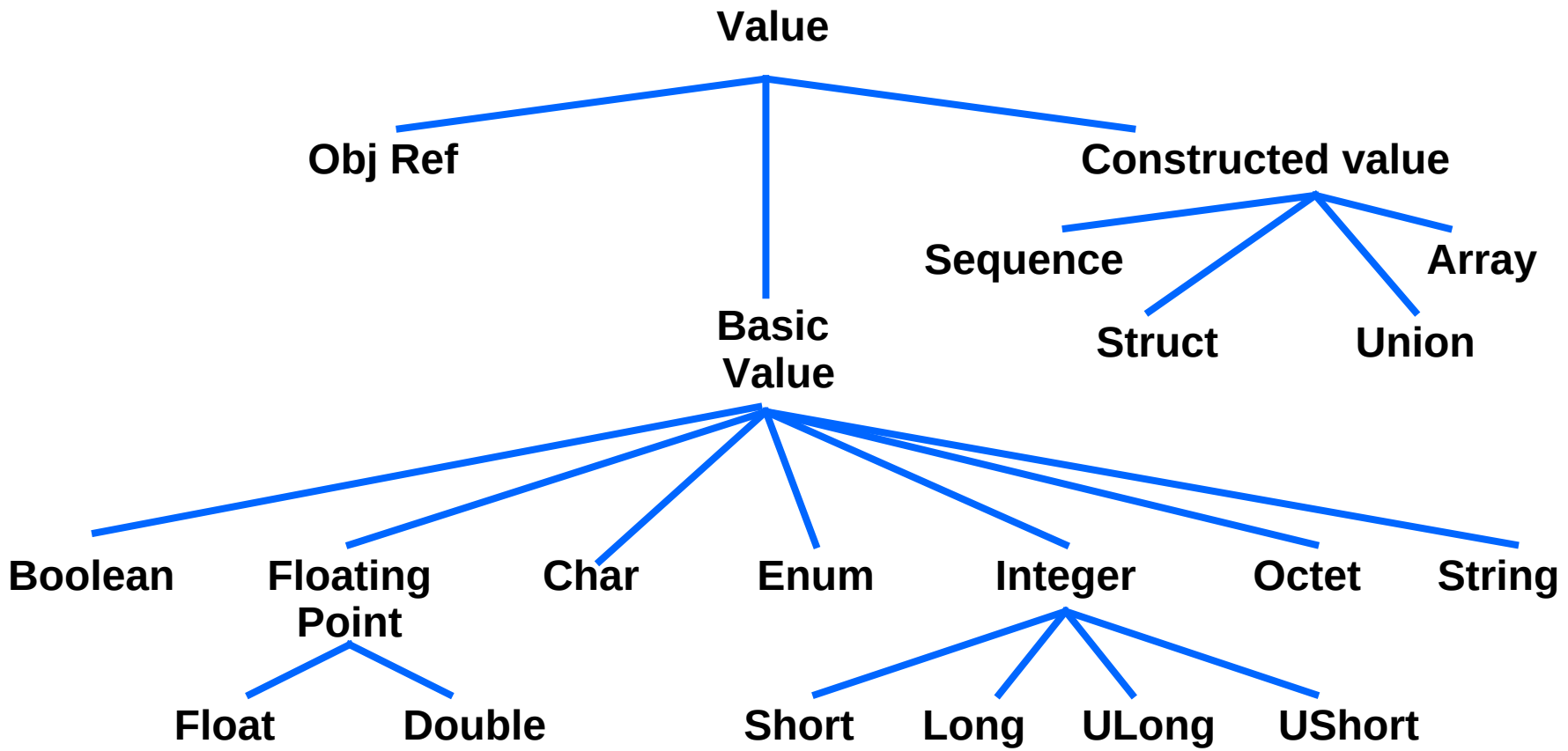
CORBA Operations

- ***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 Data Types



Why no pointers?

- ***In a distributed system, pointers are meaningless***
  - **they only make sense on the machine they were created**
- ***Use sequences instead***
  - **or arrays**

CORBA Requests

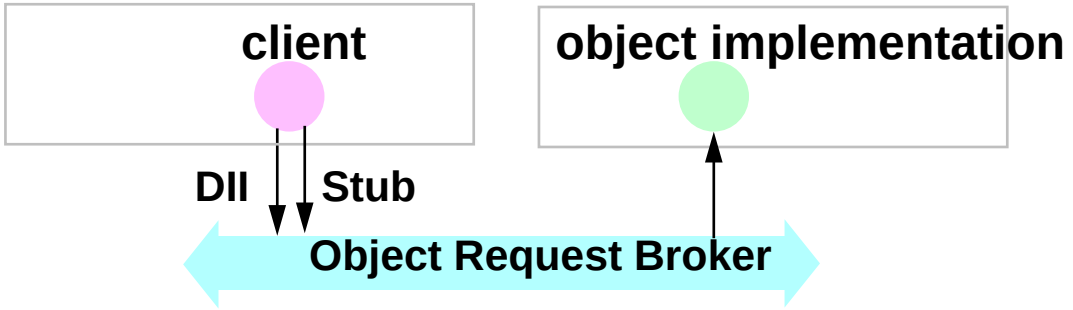
- *Clients invoke services by issuing requests for operations on objects*
- *A request consists of*
  - an operation name
  - a target object (identified by an object reference)
  - a list of zero or more (actual) parameters
  - a request context, providing additional information about the request

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

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

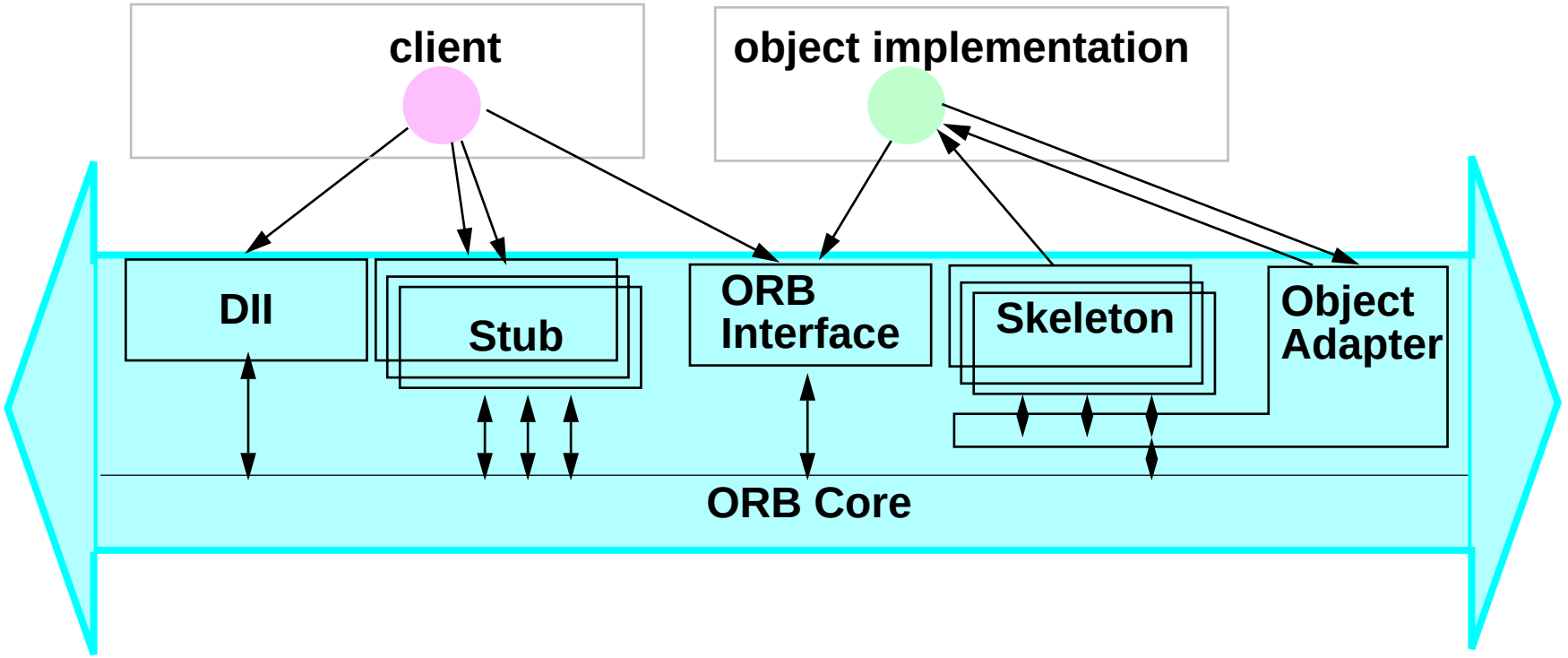
- *Application programmers must construct DII parameter lists by hand*
- *The DII API is large and complex*
- *There are no checks until run-time*
- *... You should therefore only use the DII for writing ORB tools*
  - for example, an object browser

Other DII factors

- *IDL Stubs only support synchronous requests*
  - they do not support ‘deferred synchronous’ requests (initiate/redeem, follow-up RPC)
- *The DII cannot be used on pseudo-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

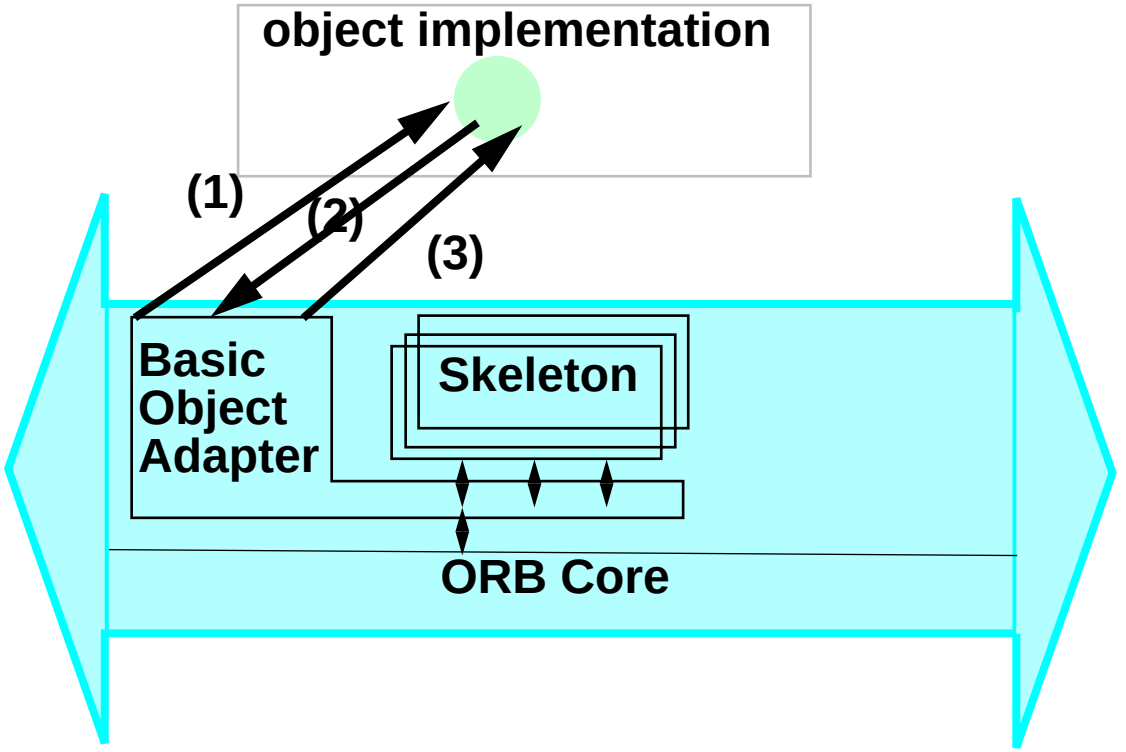
- ***Object implementations require general facilities***
  - **to establish and authenticate object identities**
  - **to create and destroy objects**
  - **to access ORB-dependent facilities**
- ***This is done via an Object Adapter***
  - **the object adapter is also used implicitly by skeletons**
- ***There is always a Basic Object Adapter***
  - **other object adapters may also be implemented 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*

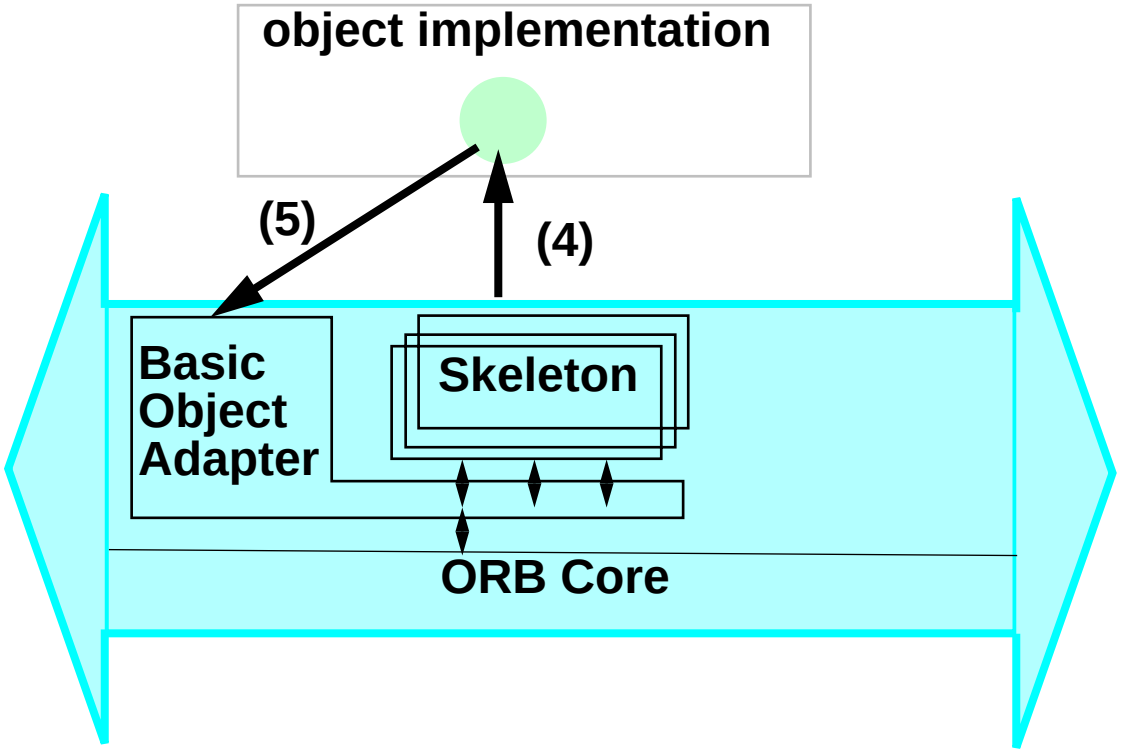
How the Basic Object Adapter works - activation

- *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

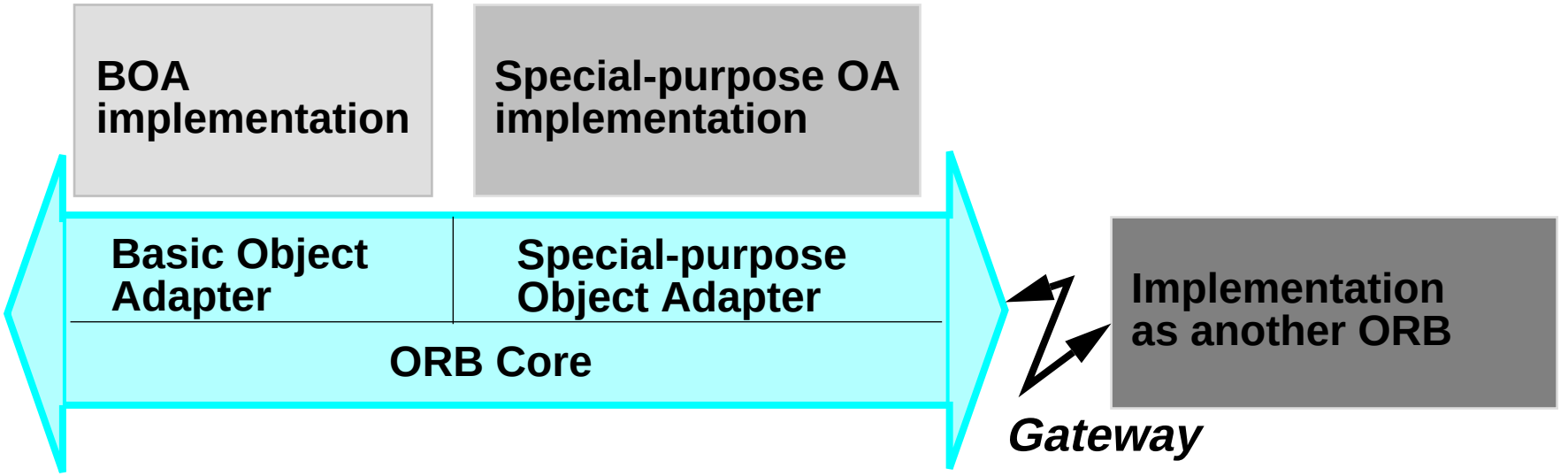
- ***The CORBA Architecture is intended to allow a variety of ORB implementations, for example***
  - **static libraries linked into clients and servers**
  - **dynamic-linked libraries bound to clients and servers at run time**
  - **as a centralized server**
  - **built into the underlying operating 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*



History of ORB 1.1

- *OMG's first RFP cycle*
  - RFI closed Aug. 1990 (8 responses), RFP closed Dec. 1990 (10 LOIs)
- *Seven proposals presented March 1991*
  - APM, Bull, DEC, DSET, HP/Sun, Hyperdesk, NCR/ODI
- *Two merged submissions by demonstrations in May 1991*
  - HP/Sun/NCR/ODI & Hyperdesk/DEC
- *"90 day" team formed, presented merged proposal (CORBA) in September 1991*
- *Proposal accepted October 1991*

ORB 1.x and ORB 2.0 Specification

- *So far, we've covered the ORB 1.1 specification*
  - The ORB 1.2 specification closes a few loopholes, but is almost identical
- *ORB 2.0 will cover*
  - Changes to Interoperability and Initialization
  - Changes to Interface Repository

Summary

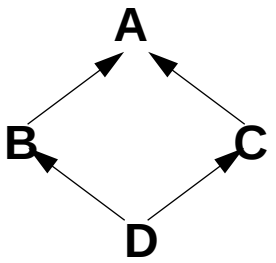
- *The ORB is underpinned by a core Object Model*
- *Clients make requests on object implementations via the ORB*
  - via IDL Stubs (preferred), or via the DII
- *Object Implementations use an Object Adapter*
- *For more on this topic*
  - see *The Common Object Request Broker: Architecture and Specification (OMG and X/Open)*

ANSA Sponsors offering CORBA products

- *Hewlett-Packard's ORB Plus*
- *ICL's DAIS*
- *Iona's Orbix*

APPENDIX

Specifying Services in CORBA IDL



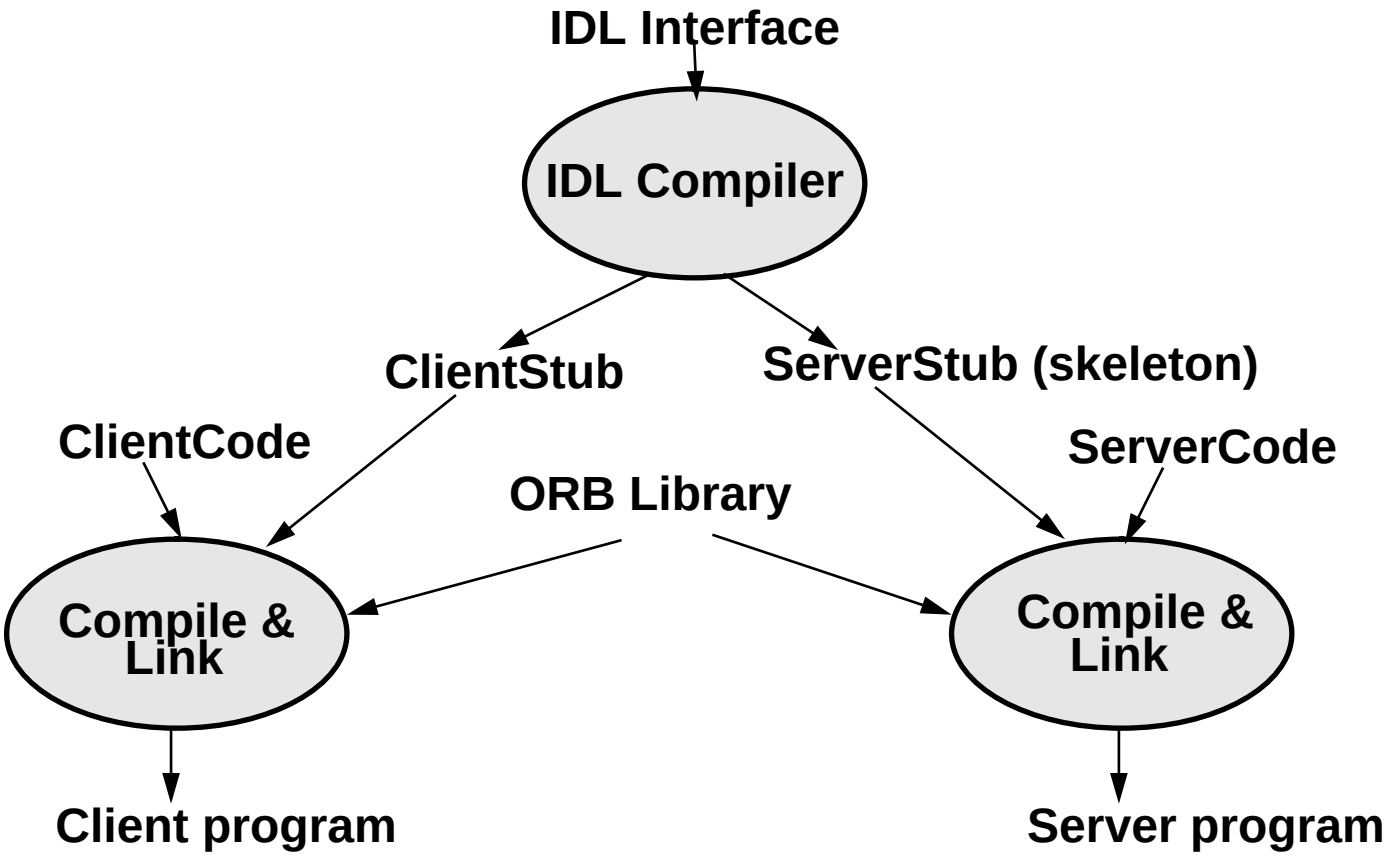
In this appendix

- *Explain how to write service specifications in CORBA IDL (Interface Definition Language)*
- *Explore the basic constructs of CORBA IDL*
- *Explain the significance of interface inheritance in CORBA*

Interface Definitions

- ***Are a 'contract' between service provider and service user***
- ***Contain only definitions and declarations***
  - **no statements**
  - **...they cannot be executed**
  - **... they are 'sophisticated header files'**
  - **... they *are* compiled**
- ***Support many programming languages (C, C++, Smalltalk,Ada,...) from one interface definition***
  - **any programming language for which there is a *language mapping***

CORBA IDL in a simple build process



CORBA applications and interfaces

- ***One CORBA application can***
  - **use many IDL interfaces (as clients)**
  - **implement many IDL interfaces (as a server - an *object implementation*)**
- ***One CORBA interface can***
  - **be used by many applications**
  - **be implemented by many applications**

A simple service in CORBA IDL

- *This looks rather like C++*

```
interface Echo {
// Comment lines start with two slashes

   string Echo (in string Src);

   string Reverse (in string Src);

};
```


Example C++ Interface Mapping

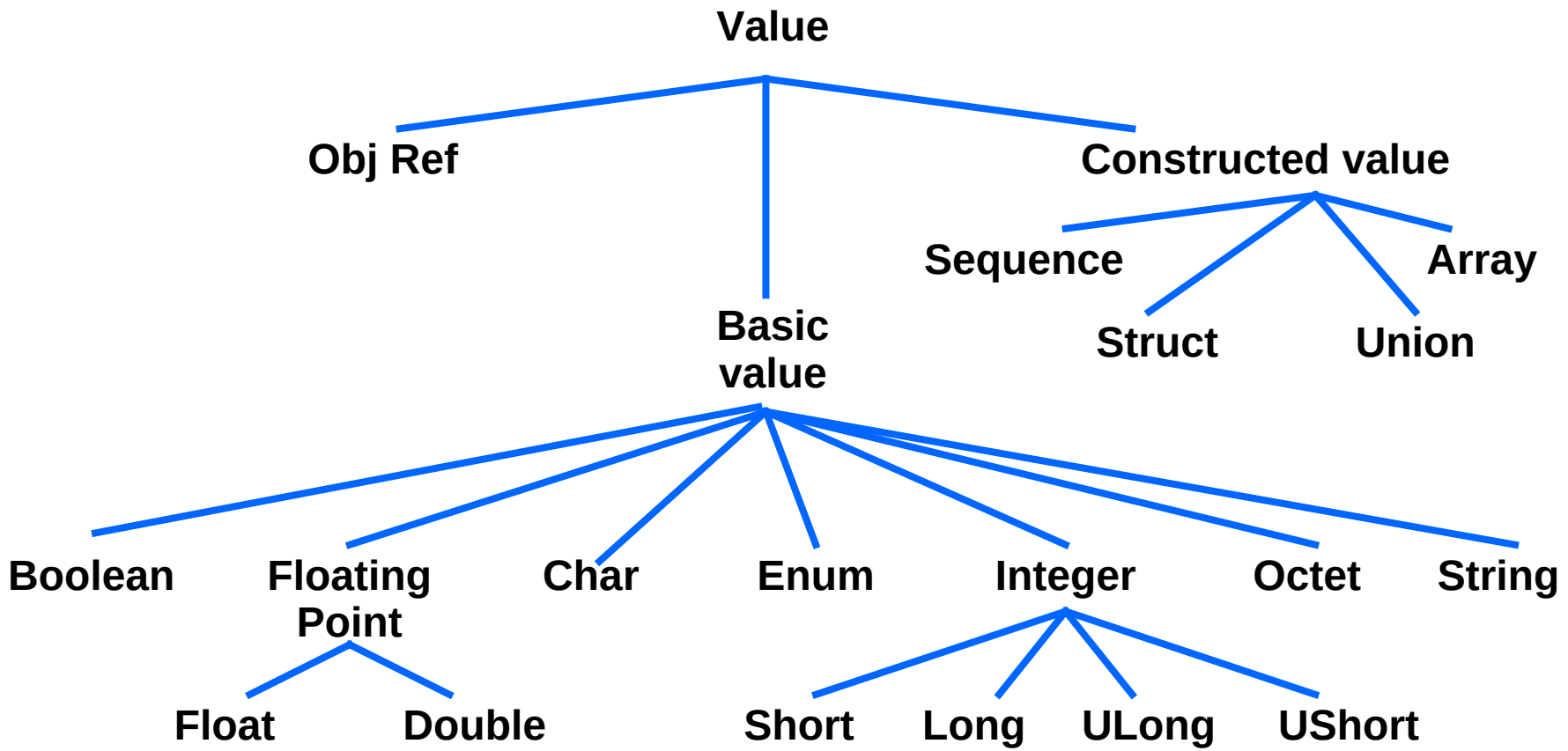
- *This mapping is typical*

```
class Echo;
typedef Echo *Echo_ptr;
typedef Echo_ptr Echoref;
class A : public virtual Object
{
  // Inherits from the C++ class Object defined by CORBA
  public:
  ...
  virtual char *Echo (const char *Src) = 0;
  virtual char *Reverse (const char *Src) = 0;
  ...
};
...
```


CORBA IDL for specification

- ***CORBA IDL is close to a pure specification language***
  - **special cases are included for efficiency of language mappings**
  - **... there are few arbitrary restrictions**

CORBA Data Types



CORBA Types Have Specified Values

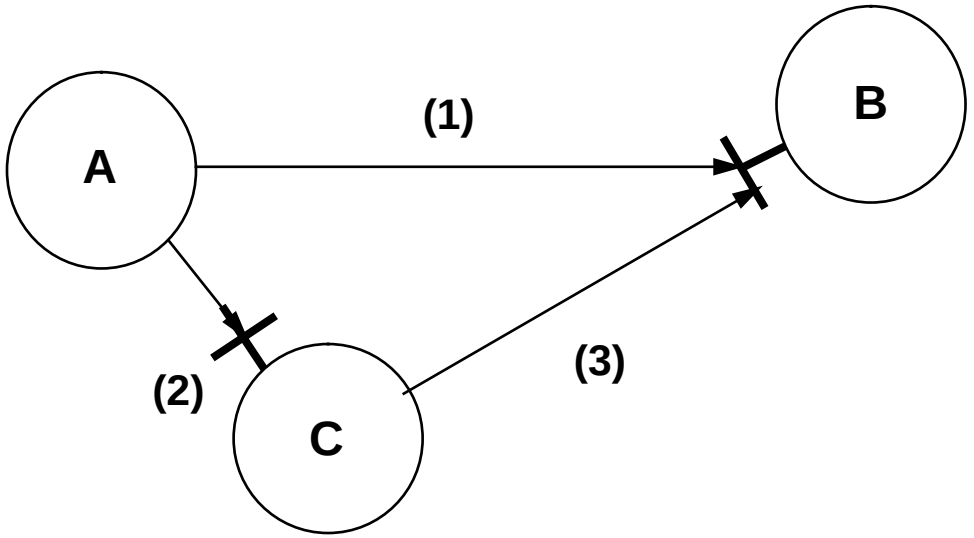
- *Unlike some programming languages, CORBA types have a specified set of values*
  - for example, short has exactly the range -2^{15} .. $2^{15}-1$
  - ... no more, no less
- *What about 64-bit integers?*
  - Sorry, no 64-bit integers - long is the longest: -2^{32} .. $2^{32}-1$
  - ... and there's no 8-bit integer either (although there is an octet type)

CORBA Types Do Not Have Specified Representations

- ***The representation of the values is not specified***
  - **it will differ between machines (e.g byte ordering)**
  - **... the stubs resolve this (when marshalling/unmarshalling)**
  - **... the application programmer does not have to worry**

Transfer of Object References

- ***Recall that in a distributed system, we need to be able to transfer object references***
  - **Object A is using Object B**
  - **Suppose it needs to tell C to use the same interface**



  - **It must be possible to pass a reference to B's interface between A and C**

CORBA Object References

- *For an object reference, just use the name of the interface*

```
interface B {
   ...
};
interface C {
   ...
   void Op (in B my_B);
   ...
};
```

  - A calls C's Op operation passing a B parameter; the implementation of C can now call operations of B

- *Object references can be used freely in CORBA IDL*
  - not just as parameters - also in structs, sequences, unions, arrays

CORBA Enumerations

- *Enumerations are ordered values*
  - ...for those languages that support ordered enums
  - ...so declare them in ascending order

```
enum boolean {FALSE, TRUE}
```

- *Enumerations can have up to 2^{32} enumerators*

CORBA Structures

- *Structures are records with fields*

- *For example*

```
struct position_t {
   float x, y;
};
```


CORBA Unions

- *CORBA unions are 'discriminated'*
  - they must have an explicit typed discriminator (tag field)
  - you'll probably want to use an enum (or perhaps boolean) for the discriminator, but you can use integer types or even char
- *For example*

```
union Example switch (long) {
   case 1: long x;
   case 2: float y;
   default: char z;
};
```

- *Note that the discriminator has a type (here, it is long), but no name*
  - language mappings give it a standard name

CORBA Sequences

- *Sequences are one-dimensional, with*
  - a maximum size (fixed at compile time)
  - a length (determined at run time)
- *Sequences can be specified as bounded or unbounded (with or without a maximum size)*
- *For example, a bounded sequence of longs...*

```
sequence<long,10>
```

- *or an unbounded sequence of sequences*

```
sequence<sequence<long> >
```

CORBA Strings

- ***Strings are sequences of char***
  - **except the NUL character, of course**
- ***Strings can be specified as bounded or unbounded***

  `string<10>`
  - **the bound is the maximum length...**
  - **... in some language mappings, you'll have to allocate extra storage space for a NUL terminator (11 bytes for C)**
- ***Strings are not that special***
  - **they are provided because their language mappings may be more convenient and efficient**

CORBA Arrays

- ***Arrays are multi-dimensional, and fixed size***
  - **indexed by integers**
  - **language mapping determines how the array indices are used (for example, 0..size-1 for C)**

The absence of pointers

- *CORBA IDL deliberately does not support pointers*
  - they are not meaningful in a distributed system
- *Why do we normally use pointers?*
  -
  -
  -
  -
  -

Sensible alternatives to pointers

- ***How should these pointers be described in CORBA IDL instead?***
  -
  -
  -
  -
  -

CORBA Constants

- *CORBA interfaces can declare constants of basic types*
- *For example*

```
const long L =4;
```

- *This is handy for declaring common bounds for arrays, sequences, and strings*
- *Constant expressions are also allowed, using C-style operators*

CORBA Operations

- *Recall that operations are much like function prototypes...*

```
Status create_request (
   in Context        ctx,
   in Identifier     operation,
   in NVList         arg-list,
   inout NamedValue  result,
   out Request       request,
   in Flags          req_flags
);
```

- *... note the use of modes in, out, and inout*
  - and a result (here, of type Status)

CORBA Exceptions

- *An exception is like a named 'error status'*
  - but it can have additional information...
  - ... it is like a struct
- *An operation must declare which exceptions it raises*
- *You can declare your own exceptions*

```
interface A {
     exception E {
        long L;
     };
     void f() raises(E);
};
```

CORBA standard exceptions

- *There is a standard set of exceptions that can occur during any request*
  - these are predefined in module StExcept
- *CORBA exceptions are not necessarily implemented as programming language exceptions*
  - this depends on the language mapping

Inheritance in CORBA IDL

- *Inheritance in CORBA IDL is solely about extension of specifications*
  - it is nothing to do with object implementations
- *An interface can be derived from another interface*

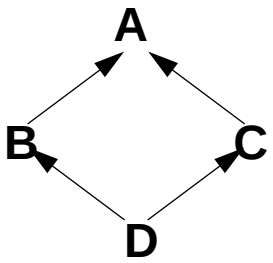
```
interface A {...};
interface B: A {...};
```

  - here, A is the base interface of B
  - all the definitions of A implicitly appear in B
- *Redefinitions are possible too*

Inheritance rules

- *Multiple inheritance is possible*

```
interface A {...};
interface B: A {...};
interface C: A {...};
interface D: B, C {...};
```



- *There are various rules to avoid strange interactions when names are redefined or names clash due to inheritance*
  - ambiguities can be resolved by explicit qualification

The effect of inheritance in CORBA

- *Inheritance simply avoids the effort of writing down definitions a second time...*

```
interface A {void f (in float x)}
interface B {long g (in long x)}
interface C: B, A {void h (in long x)}
```

- *... interface C is completely equivalent to:*

```
interface C:{void f {in float x}
             long g {in long x}
             void h {in long x}}
```

- *The important thing is that an object with interface C may be substituted wherever clients require A or B*
  - it makes no difference which way you write down C

Is IDL really necessary?

- *"Why can't I just write in my ordinary programming language?"*
  - if we can convert between IDLs, why not directly from C++ headers?
- *One day, this may be practical*
  - ...perhaps using special C++ pragmas embedded in header files
  - ...but for the foreseeable future, IDL is going to be the way to specify interfaces

General guidance for using CORBA IDL

- *Interfaces don't have to contain any operations*
  - you can use them to collect together type definitions, constants,...
- *Watch out for incomplete CORBA products*
  - some products don't yet support full CORBA IDL
- *Beware of mode inout*
  - when an unbounded string or sequence is passed as an inout parameter, the returned value cannot be longer than the input value...
  - if an exception is raised, the value is undefined (it is not necessarily the input value)
- *Think carefully before declaring new exceptions*

Summary

- ***IDL defines interfaces, not implementations***
  - **clients and servers can be implemented in any supported programming language**
- ***For more on CORBA IDL***
  - **see *The Common Object Request Broker: Architecture and Specification* (OMG and X/Open)**