Distributed Ada 95 with PolyORB
A schizophrenic middleware

ACM SIGAda'05 tutorial
Atlanta – Nov. 14th, 2005

Thomas Quinot, PhD
Senior Software Engineer
AdaCore

Laurent Pautet, PhD
Assistant Professor
École nationale supérieure des télécommunications

AdaCore
The GNAT Pro Company

www.adacore.com

Tutorial structure

- Motivation
- CORBA approach
- The Open Management Group
- The Open Management Architecture
- OMG IDL - standard Ada 95 mapping
- ORB core services
- Common Object Services
- The Ada 95 Distributed Systems Annex
- Ada 95 partitioning with GNATDIST
- Inteoperability with schizophrenic middleware

Motivation - Distribution models

- **Paradigms for describing and designing dist. apps**
- **Based on repartition facilities:**
  - Message passing
  - Remote subprograms
  - Distributed objects
  - Shared objects (distributed shared memory)
  - Transaction
- **Provides control over facilities**
  - Description language for interactions
  - APIs for distribution libraries
- **Implemented by *middleware***

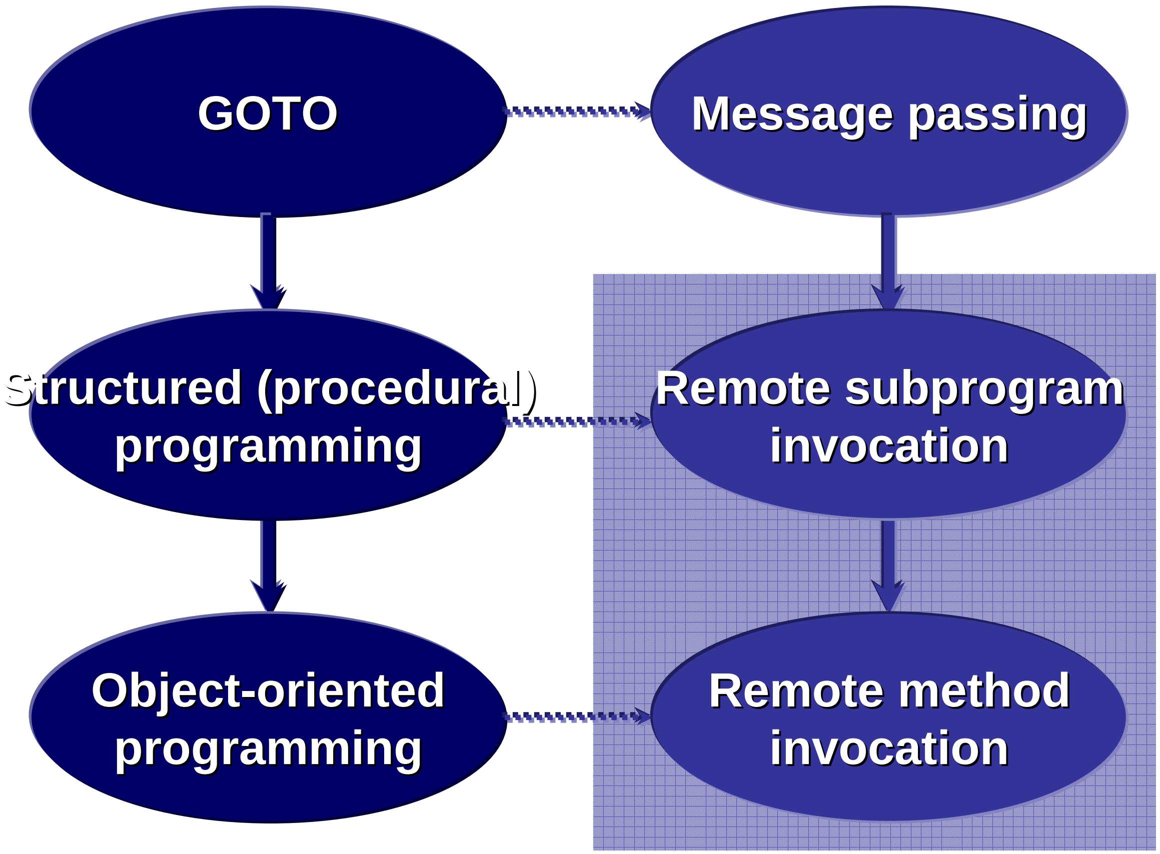
Motivation – Message passing

- **Low-level network interfaces**
  - UDP/TCP/ATM sockets
  - Advanced services: multicast, SSL/TLS, IPv6 anycast
- **PVM / MPI**
  - Massively parallel computing
  - Group communication
- **Java Messaging Serivce**
  - High-level interface
  - Message-oriented middleware (MOM)
  - Point-to-point
  - Point-to-multipoint: publish/subscribe

Motivation – Remote subprograms

- **Sun RPC**
  - Service location service: portmap/rpcbind
  - Rpcgen compiler producing stubs+skeletons
  - Part of DCE (Distributed Computing Environment)
  - Used for NFS, NIS...
- **Ada 95 DSA (RCI units)**
  - Bridges non-distributed and distributed software development
  - Transparent naming and location services
  - Remote call abortion
  - Dynamically bound RPC: remote access-to-subprogram types
- **SOAP, XML/RPC**
  - RPC based on XML+HTTP
  - Microsoft-backed alternative to CORBA

Motivation – Distributed models analogies



Motivation – Remote subprogram calls

- **Language-dependant solutions**
  - Modula-3
  - Java/RMI (Remote method invocation)
  - Ada 95 DSA (Distributed systems annex)
- **Language-independent solutions**
  - CORBA (OMG IDL)
  - DCOM (DCOM IDL)

Motivation - Heterogeneity

- **Remote invocation on objects independent of:**
  - Programming language
  - Operating system
  - Execution platform
  - Technology vendors
  - Communication protocols
  - Data formats
- **Consensus required for interoperability**

The Leader in Ada Technology

CORBA

CORBA approach - Objectives

- **Environment specified by open standards**
- **Modularity by object-oriented design**
- **Opacity of implementation details**
- **Definitions**
  - OMG – Open Management Group
  - OMA – Open Management Architecture
  - IDL – Interface Definition language
  - ORB – Object Request Broker
    - Communication subsystem for distributed application
    - “Software bus” analog to a hardware bus

CORBA approach – Fundamental principles

- **Location transparency**
  - Usage of a service is independent of its location
- **Access transparency**
  - Remote services are invoked just as local ones
- **Separation of interface and implementation**
  - Clients depend semantically on interfaces only
- **Typed interfaces**
  - Object references are typed
- **Support of multiple interface inheritance**
  - Allows extension, evolution, specialisation of services

CORBA approach – Fundamental CORBA notions

- **Interface – set of object operations and attributes**
  - Described in OMG IDL
  - Contract between client and server
  - Defines offered services
- **Implementation – code for operations**
- **Location – Physical host that contains an instance of an object**
- **Reference – structure (pointer) that designates a (possibly remote) object**

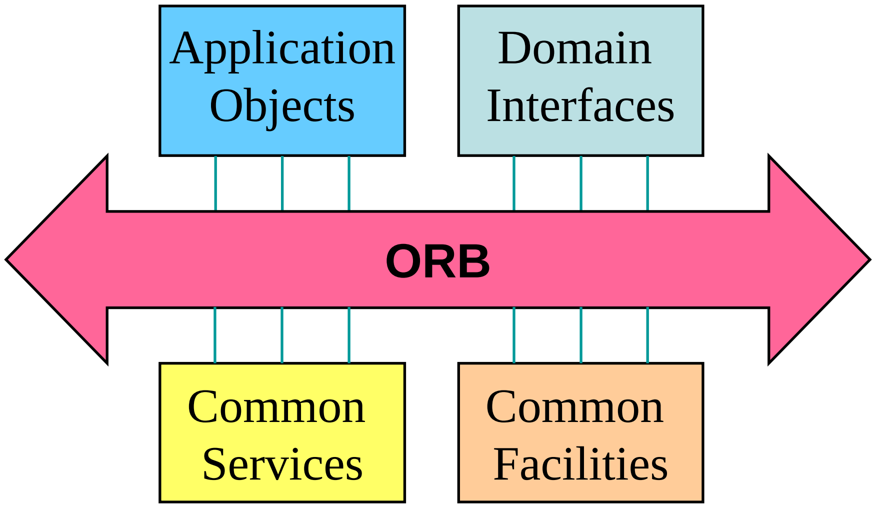
OMG – Organisation

- **Consortium incorporated in 1989, > 800 members**
  - Hardware vendors: Sun, HP, DEC, IBM...
  - OS vendors: Microsoft, Novell...
  - Tools vendors: Iona, Dorland...
  - Software vendors: Lotus, Oracle...
  - Industrial users: Boeing, Alcatel...
- **Missions**
  - To promote distributed objects technology
  - To provide an open, standard architecture for distributed applications
  - To ensure application interoperability and portability by offering:
    - Common terminology
    - Abstract object model
    - Reference architecture for the object model
    - Common APIs and protocols

OMG – Operations

- **RFPs (Request for proposals)**
  - As needs arise, OMG produces RFPs
  - A RFP defines objectives for a new specification
  - + a timeline
- **Proposals**
  - Any interested member may submit a proposal
  - A prototype implementation must be provided
  - Proposals are revised until consensus is reached
  - Final implementation must follow within one year

OMA - Architecture



- **Application objects**
- **ORB core**
  - Communication subsystem
- **Common object services**
  - Naming
  - Events
  - Transactions
  - ...
- **Common facilities**
  - User interface
  - Information mgmt
  - System mgmt
  - Task mgmt
  - ...
- **Domain interfaces**
  - Simulation
  - Banking
  - Telecommunications
  - ...

OMA – Application objects and Common services

- **Application objects**
  - IDL interface specifications
  - Defined by user application
  - Outside of OMG standardization scope
  - Opportunity to standardize emerging objects
- **Common services**
  - IDL interface specifications
  - May be extended or specialized through inheritance
  - Focus on application developer
  - Independent of application domain
  - Extend ORB core services
  - Examples:
    - Naming
    - Events
    - Transactions
  - "Horizontal interfaces"

OMA – Domain interfaces and Common facilities

- **Domain interfaces**
  - IDL interface specifications
  - OMG standards
  - May be extended or specialized through inheritance
  - Specific to an application domain:
    - Medical
    - Financial
    - Telecommunications
    - …
  - “Vertical interfaces”

- **Common facilities**
  - IDL interface specifications
  - OMG standards
  - May be extended or specialized through inheritance
  - Focus on end-user
  - Independent of application domain
  - Examples:
    - Printing
    - Online help
    - Error messages mgmt
  - “Horizontal interfaces”

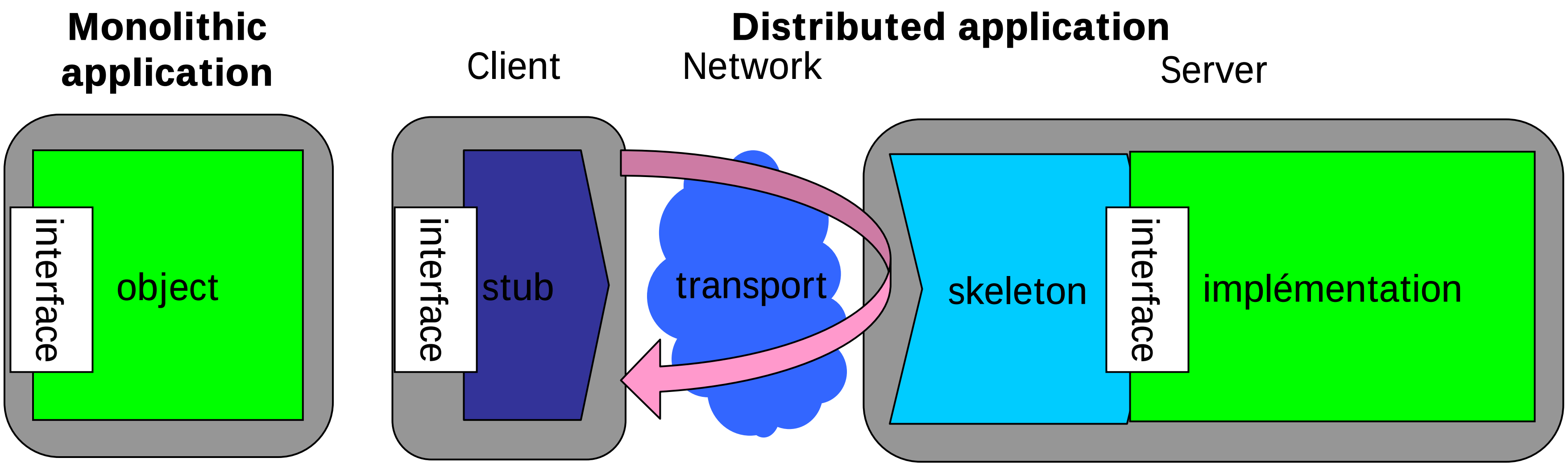
OMA – Definitions (1/2)

- **Client**
  - Any entity capable of sending requests to objects that provide a service
  - A client handles references to remote objects
- **Reference**
  - Value used by a client to invoke services on a remote (implementation) object
  - Acts as a local proxy for the remote object
- **Implementation object**
  - Server-side entity implementing the operations of an IDL interface

OMA – Definitions (2/2)

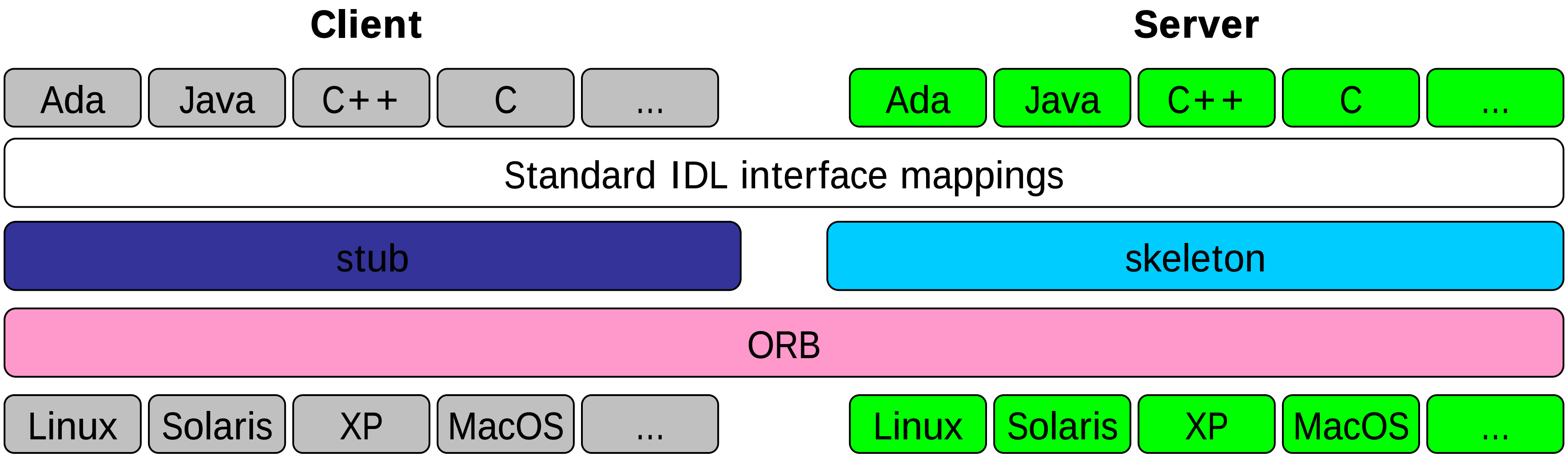
- **Request**
  - Message emitted by a client to request execution of an operation on a terget object
  - Contains operation identifier, target object identifier, and request arguments
- **Interface**
  - Description of a set of operations that an object may implement
  - Expressed in OMG IDL
- **Operation**
  - Named entity describing the signature of a service that can be requested from an object: types of the arguments and returned values.

CORBA - Transparency



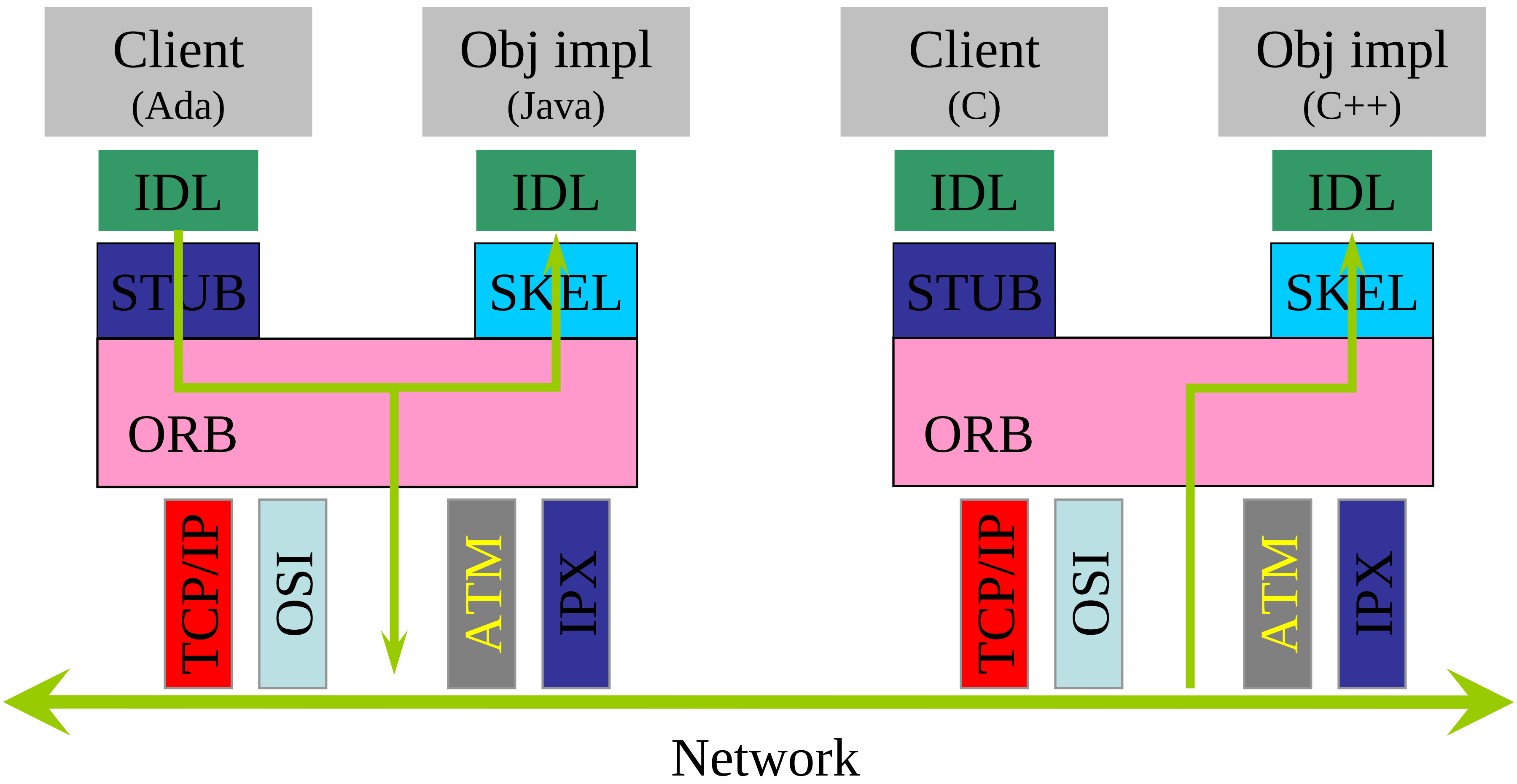
- **Stub interfaces client to client-side ORB**
  - Translates method call into request emission
  - Translates reply reception into method return
- **Skeleton interfaces server-side ORB to implementation**
  - Translates request reception into method call
  - Translates method return into reply emission
- **Distribution is hidden to client and implementation**
  - Marshalling/unmarshalling of arguments
  - Transport of request

CORBA – Heterogeneity and interoperability

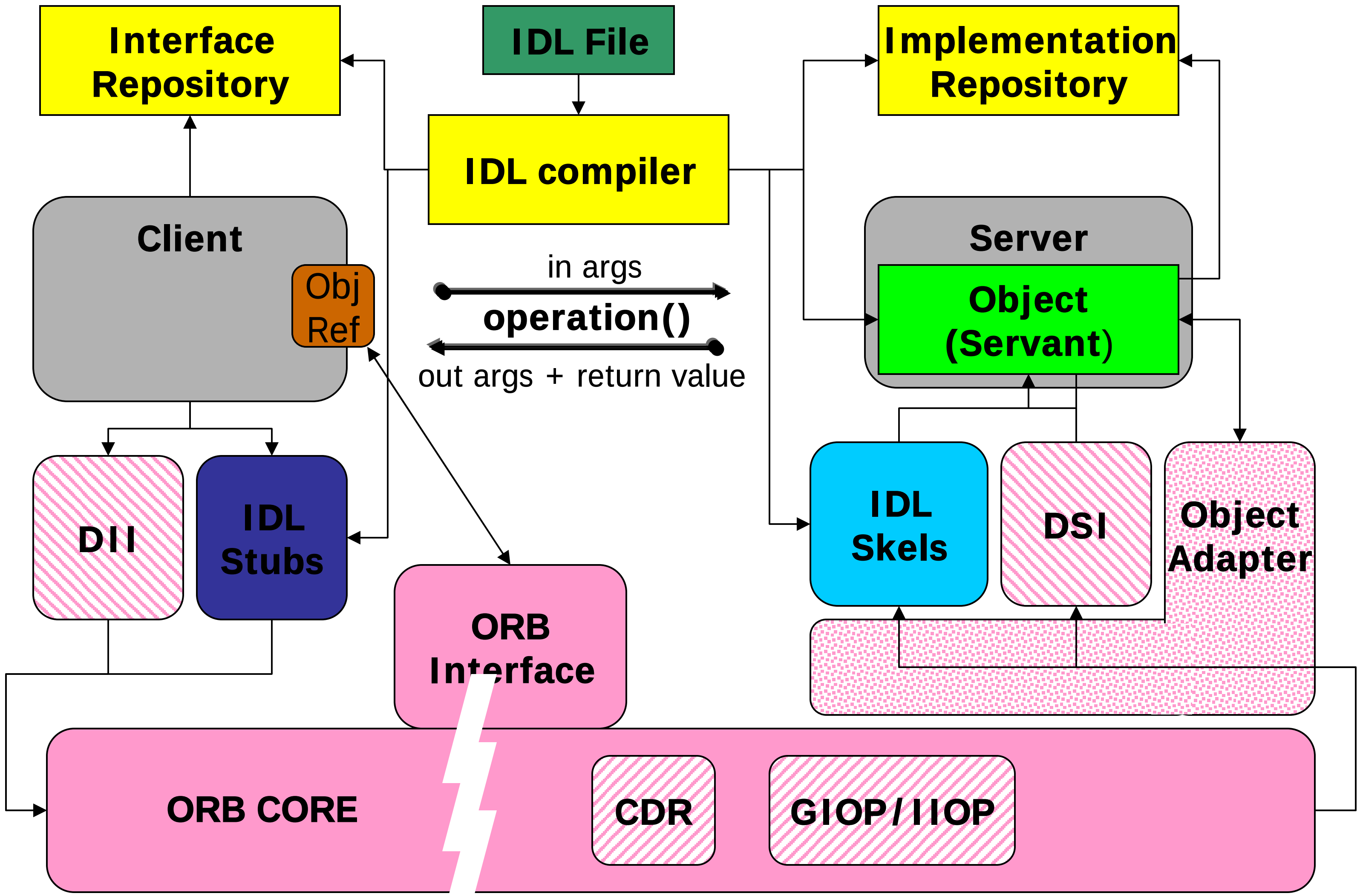


- **Stubs and skeletons generated automatically**
- **Interoperability masks OS/language/platform differences**
- **Compiler applies standard mapping of IDL to host language**
- **Client in L1 calls L1 stub**
- **L2 skeleton calls L2 implementation**

CORBA - Overview



CORBA –Architecture



CORBA – Object Request Broker

- **The ORB (Object Request Broker) manages:**
  - Object location and identification
  - Arguments marshalling/unmarshalling
  - Implementation upcalls and exception handling
  - Transport and procotols
  - Resources
- **ORB interface**
  - Makes ORB services available to application objects (clients and implementations)
  - Through standard APIs

CORBA – Static and dynamic stubs

- **Stubs**
  - Prepare ‘in’ arguments
  - Decode ‘out’ arguments and return value
- **Static stubs**
  - Specific interface
  - Compile-time generated by IDL compiler
  - Spec is defined by standard
- **Dynamic stubs**
  - Generic API (interface-independent)
  - Dynamically build requests for arbitrary interfaces
  - Allow invocation of services discovered at runtime

CORBA – Static and dynamic skeletons

- **Skeletons**
  - Symmetric to stubs on server side
  - Decode ‘in’ arguments
  - Prepare ‘out’ arguments and return value
- **Static skeletons**
  - Specific interface
  - Compile-time generated by IDL compiler
  - Depend upon standard spec provided by implementation
- **Dynamic skeletons**
  - Generic API (interface-independent)
  - Dynamically decode requests for arbitrary interfaces
  - Allow providing of services discovered at runtime

CORBA – Object adapters and references

- **Object adapters**
  - Provide a framework for implementation object instances
  - Interface between implementations and ORB core
  - Can manage instantiation of implementations
  - Supplies references designating implemented objects
  - Dispatch calls to implementations
  - Several OAs can coexist in distinct namespaces in the same server
- **Object references**
  - Unique identifier for an object within an ORB
  - Reference ::= <interface><profiles>
  - Profile ::= <network address><object key>
  - Object key identifies OA and object within OA

CORBA - Repositories

- **Interface repository**
  - Database of interface descriptions
  - Usually one per execution environment
  - Can be interconnected
- **Implementation repository**
  - Database of implementations
  - Usually one per host

OMG IDL – Interface Definition Language

- **Common specification language**
  - Declarative, strongly typed, object-oriented
  - Objects have operations (name+signature) and attributes (name+type)
  - Multiple inheritance
  - No overload
  - Encapsulation of implementations
- **Independent of programming languages**
  - Standard mappings to OO and non-OO languages:
    C++, Java, Ada 95, Smalltalk, C, COBOL...
    Python, Perl, Modula...
  - IDL compiler generates stubs, skeletons and implementation templates
  - IDL is “esperanto” between host languages

OMG IDL – Syntax

- **Close to C++ but…**
- **Very different semantics**
- **New keywords:**
  module, interface, attribute, readonly, oneway, in, out
- **Identifiers**
  - Letters, digits, _
  - Conservative casing:
    - Two identifiers may not diff just by casing, but…
    - Usage occurrences within IDL must have proper casing

OMG IDL – Specifications and modules

```
<specification> ::= <definition>+
<definition> ::=
      <module> | <interface>
    | <type> | <constant> | <exception>
<module> ::= <definition>+
<interface> ::= <header> <body>
<body> ::= <export>*
<export> ::=
      <attribute> | <operation>
    | <type> | <constant>| <exception>
```

```
const long N = 100;
module Namespace {
  module Types {
    typedef string Name;
  };
  interface Group {
    ::Namespace::Types::Name Users[N];
  };
};
```

- **Specification / module contains**
  - Nested modules (providing namespaces)
  - Constants, types, exceptions
  - Interfaces
  - :: is scoping operator

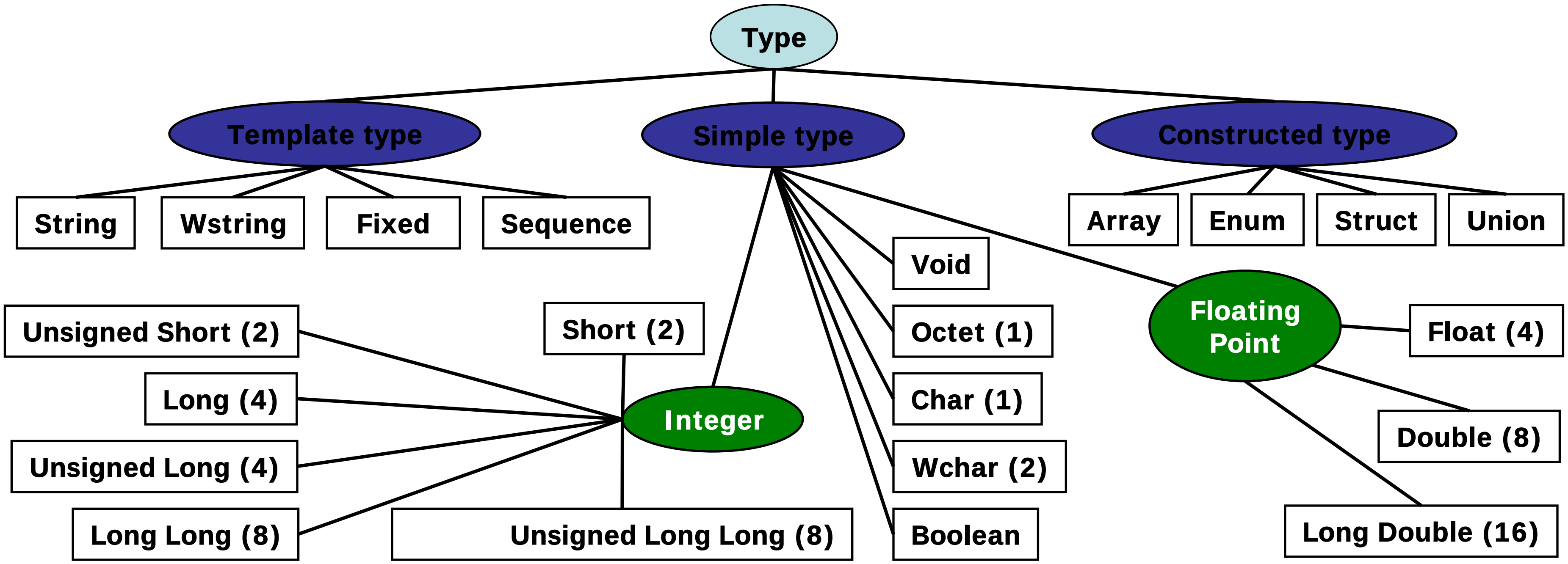
OMG IDL – Elementary types

```
<type> ::= <constructed_type>
  | <simple_type> | <template_type>
<constructed_type> ::= <union_type>
  | <struct_type> | <enum_type> | <array>
<simple_type> := <floating_point_type>
  | <integer_type>  | <object_type>
  | <any_type> | <octet_type> | char_type>
  | <wide_char_type> | <boolean_type>
<template_type> := <sequence_type>
  | <string_type> | <wide_string_type>
  | <fixed_point_type>
```

```
typedef string Name;
sequence <Name> Names;
struct Group {
  string Aliases[3];
  Names Users;
};
enum Sex {Male, Female};
union Status switch (Sex) {
  case Male: boolean Bearded;
  default:    long Children;
};
```

- **Typedef** define new type
- **Union** overlay (similar to C union) + discriminant
- **Enum** discrete type with enumerated literals
- **Array** constrained array of fixed size

OMG IDL – Complex types



- **Object** root type of all references
- **Any** typed container for arbitrary data

OMG IDL - Constants

<constante> ::= "**const**" <constante_type>
<identifier> "=" <expression>
<exp> ::= [<subexp>] <operator> <subexp>
<operator> ::= <unary_operator>
| <binary_operator>

- **Constant type must be of known type**
- **Value must be valid for the type**

const long N_Components = 150;
const long Component_Size = 8;
const long Component_Table_Size =
N_Components * Component_Size ;
const string Warning = "Beware !";
const Octet Invalid = 2 * 150 – 60;
// (2 * 150) – 60 = **300** – 60 = 240

Unary operator	
+ - ~	plus, minus, not
Binary operator	
| ^ &	or, xor, and
<< >>	shift left, shift right
* /	mul, div
+ - %	plus, minus, mod

OMG IDL - Exceptions

```
<exception> ::=   " exception" <identifier>
     <member>*
<member> ::= <type> <declarators>
<declarators> ::= <declarator>
                  {, >declarator}
```

exceptions prédéfinies:
OBJECT_NOT_EXIST
COMM_FAILURE
BAD_PARAM
...

```
exception No_Such_Name ;
exception Not_Unique {
   long Many;
};

long Lookup (in string Name)
  raises (No_Such_Name,
            Not_Unique);
```

- **Exceptions are valued (like a struct)**
- **Exceptions may be:**
  - System (standard-defined)
  - Vendor-defined
  - User-defined

OMG IDL - Interfaces

```
<interface> ::= <header><body>
<header> ::= "interface" <identifier>
   [: <inheritance>]
<inheritance>::= <interface_name>
     {, < interface_ name>}
<body> ::= <export> *
<export> ::= <attribute> | <operation>
  | <type> | <constant>  | <exception>
```

```
interface Chicken;
interface Egg;
interface Chicken {
   enum State {Sleeping, Awake};
   attribute State Alertness;
   Egg lay();
};
interface Egg {
   Chicken hatch();
};
```

- **Fundamental construct: structure of an exposed service**
- **Multiple inheritance, diamond inheritance**
- **All interfaces are derived implicitly from CORBA::Object**

OMG IDL - Operations

```
<operation> ::= ["oneway"] <type>
  <identifier> <parameters>
  [<raise>] [<context>]
<parameters> ::= <parameter>*
<mode> ::= "in" | "out" | "inout"
<parameter> ::= <mode> <type>
    <identifier>
<raise> ::= "raises" (<exception>,
  {, <exception>})
<context> ::= "context" (<string>,
  {, <string>})
```

```
interface Stack {
  exception Empty;
  readonly attribute long Length;
  oneway void Push (in long Value);
  long Pop () raises (Empty);
};
```

- An operation (methode) may raise exceptions
- Raised (user-defined) exceptions must be declared
- A oneway operation (no inout or out arguments, no exceptions, no returned value) is asynchronous

OMG IDL - Attributes

<attribute> ::= " **attribute** " [" **readonly** "]
<type> <declarators>
<declarators> ::= <declarator> {, declarator}

exemple de getter/setter en Java:
java.lang.string Title ();
void Title (java.lang.string value);
float Balance ();

```
interface Account {
  attribute string Title;
  readonly attribute float Balance;
};
struct Coord {
   unsigned long X, Y;
};
interface Triangle {
     attribute Coord Points[3];
};
```

- **Getter and setter operations**
- **No setter for readonly attributes**

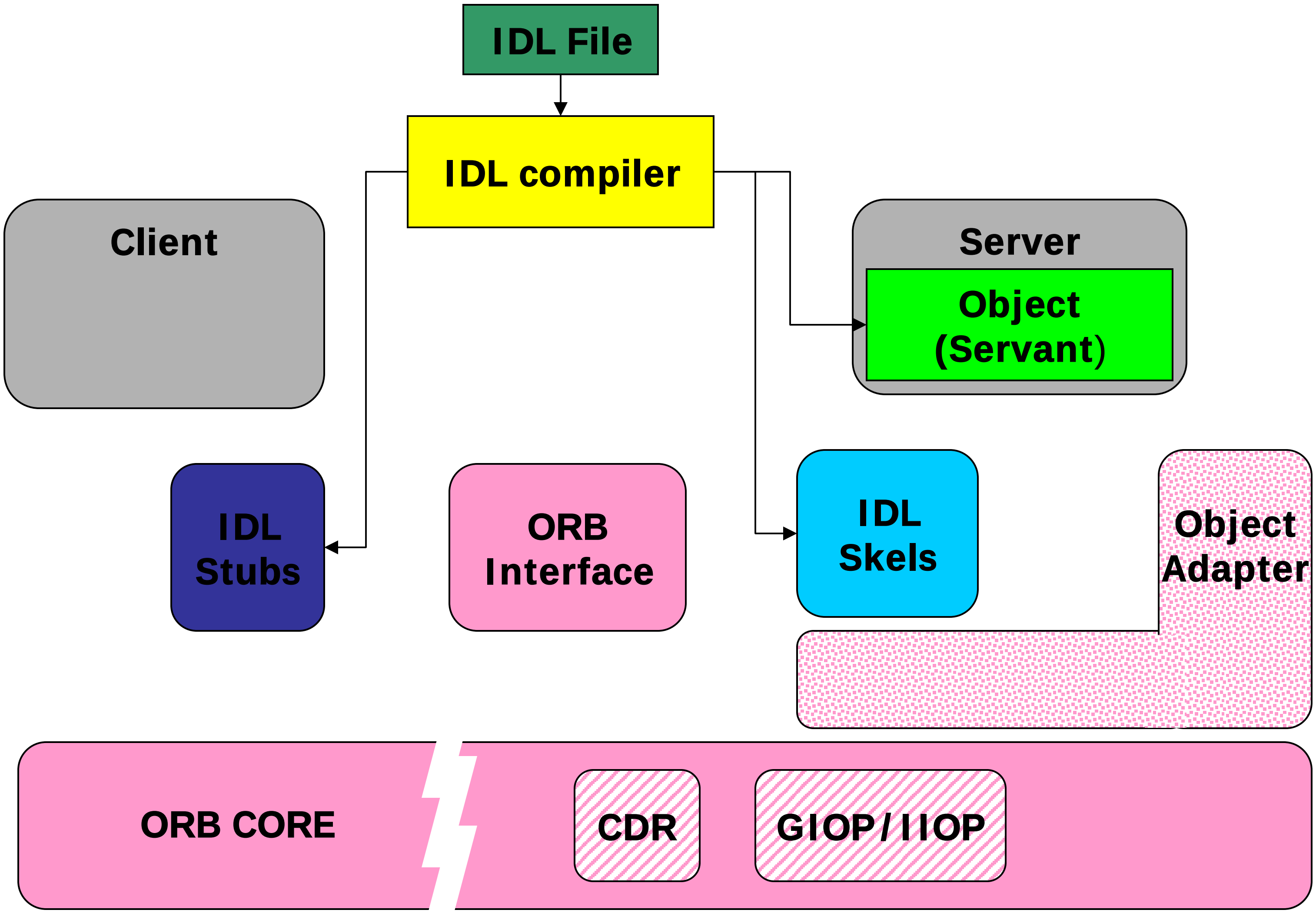
OMG IDL - Inheritance

<interface> ::= <header><body>
<header> ::= "**interface**" <identifier>
[: <inheritance>]
<inheritance>::= <interface_name>
{, < interface_ name>}
<body> ::= <export> *
<export> ::= <attribute> | <operation>
| <type> | <constant> | <exception>

```
interface Bird {
   void eat ();
};
interface Egg;
interface Chicken : Bird {
  Egg lay();
};
interface Egg {
  Chicken hatch();
};
```

- **Constants, types, exceptions, attributes, and operations are inherited from parents**
- **Types, constants, and exceptions may be overridden**
- **Attributes and operations cannot**
- **Diamond inheritance yields a single instance of the common ancestor**

CORBA development process



CORBA development process

- **OMG IDL programming**
  - Description of interfaces
  - Generation of host language stubs and skeletons by IDL compiler
  - Optionally, generation of implementation template
- **Host language programming**
  - Implementation of interfaces (from template)
  - Class tree in host language may differ from IDL class tree
  - Server partition must instantiate and register implementations
  - Client partition must obtain references to objects:
    - Through "magic" in the ORB (often for initial name service reference)
    - Well-known locations, text representations, configuration…
  - Learning curve may be steep, depending on mapping
    (Ada < Java < C++ << C)

Ada mapping

IDL	Ada
[unsigned] short	*[Unsigned_]Short
[unsigned] long	*[Unsigned_]Long
[unsigned] long long	*[Unsigned_]Long_Long
float	*Float
double	*Double
long double	*Long_Double
char, wchar	*Char, Wchar
string, wstring	*String, Wide_String
boolean	boolean
octet	*Octet
any	*Any
void	<*space*>

IDL	Ada
module	package
interface	package + tagged type
attribute	Ada methods getter/setter
operation	Ada methods
struct	record type
enum	enumerated type
union	discriminated record type
fixed	decimal type
array	array
sequence	*generic sequence
const	constante
exception	exception

* From package CORBA

Ada mapping – Syntax and modules

- OMG IDL identifiers → Ada identifiers
- In case of conflicts (with Ada reserved words or identifiers defined by the standard mapping), prefixed with "U_"

- OMG IDL modules → Ada nested packages

```
// OMG IDL
module Namespace {
  ...
};

-- Ada
package Namespace is
  ...
end NameSpace;
```

Ada mapping – Struct, enum, typedef, array

- **OMG IDL struct → Ada record type**

```
// OMG IDL
struct Point {
  long X, Y;
};
-- Ada
type Point is record
  X, Y : CORBA.Long;
end record;
```

- **OMG IDL enum → Ada enumerated type**

```
// OMG IDL
enum Color {Red, Green, Blue};
-- Ada
type Color is (Red, Green, Blue);
```

- **OMG IDL typedef → Ada derived type**

```
// OMG IDL
typedef long Duration;
--  Ada
type Duration is new
    CORBA.Long;
```

Ada mapping – Union, constant, sequence

- **OMG IDL union → Ada discriminated record**

```
// OMG IDL
union Status (boolean) {
  case TRUE:  boolean Bearded;
  case FALSE: long Children;
};
-- Ada
type Status(Switch : Boolean := True) is
  record
    case Switch is
      when True =>
        Bearded : Boolean;
      when False =>
        Children : CORBA.Long;
    end case;
  end record;
```

- **OMG IDL constant → Ada constant**

```
// OMG IDL
const double Pi = 3.14;
-- Ada
Pi : constante CORBA.Double := 3.14;
```

- **OMG IDL sequence → Ada generic instantiation**

```
// OMG IDL
typedef sequence<long>  Vector;
-- Ada
with CORBA.Sequence;
package IDL_Sequence_long is new
   CORBA.Sequence (CORBA.Long);
type Vector is
   new IDL_Sequence_long.Sequence;
```

Ada mapping – Interface, code structure

- OMG IDL interface → Ada package + tagged type

```
// OMG IDL
interface Stack {
  void Pop (out long v) raises Empty;
  void Push (in long v);
};
-- Ada
package Stack is
  type Ref is new CORBA.Object.Ref
    with null record;
  procedure Pop
    (Self : Ref; v : out CORBA.Long);
  procedure Push
    (Self : Ref; v in CORBA.Long);
end Stack;
```

For each interface <I>

- **Stubs pkg <I>**
- **Skel pkg <I>.Skel**
- **Impl pkg <I>.Impl**
- **Utility pkg <I>.Helper**

Ada mapping – Operation, attribute

- **OMG IDL operation → Ada primitive subprog.**

```
// OMG IDL
interface I {
  long M (in T x, inout T y, out T
    z);
};

-- Ada
package I is
  type Ref is new
    CORBA.Object.Ref
     with null record;
  procedure M
    (x : in T;
     y : in out T;
     z : out T;
     Returns : out CORBA.Long);
end I;
```

- **OMG IDL attribute → Ada getter / setter**

```
// OMG IDL
interface Account {
  readonly long Balance;
  string Name;
};

-- Ada
package Account is
  type Ref is new
   CORBA.Object.Ref
    with null record;
  function Balance (Self : Ref)
    return CORBA.Long;
  procedure Set_Name
    (Self : Ref; To : CORBA.String);
  function Get_Name (Self : Ref)
    return CORBA.String;
end Account;
```

Ada mapping – Exception, helper

- **OMG IDL exception → Ada exception + tagged type**

```
// OMG IDL
exception Error {
   long Code;
};
-- Ada
Error : exception;
type Error_Members is
   new
     CORBA.IDL_Exception_Members
   with record
      Code : CORBA.Long;
   end record;
procedure Get_Members
   (From : Exception_Occurrence;
    To   : in out Error_Members);
```

- **Every type has helpers for conversion to ORB representations**
  - Found in CORBA pkg for base types
  - Generated for user types

```
-- Ada
package TYPE.Helper is
   function From_Any (Item : in
     Any)
      return TYPE;
   function To_Any (Item : in
     TYPE)
       return Any;
   function To_Ref
      (The_Ref : in
     CORBA.Object.Ref'Class)
        return Ref;
end TYPE.Helper;
```

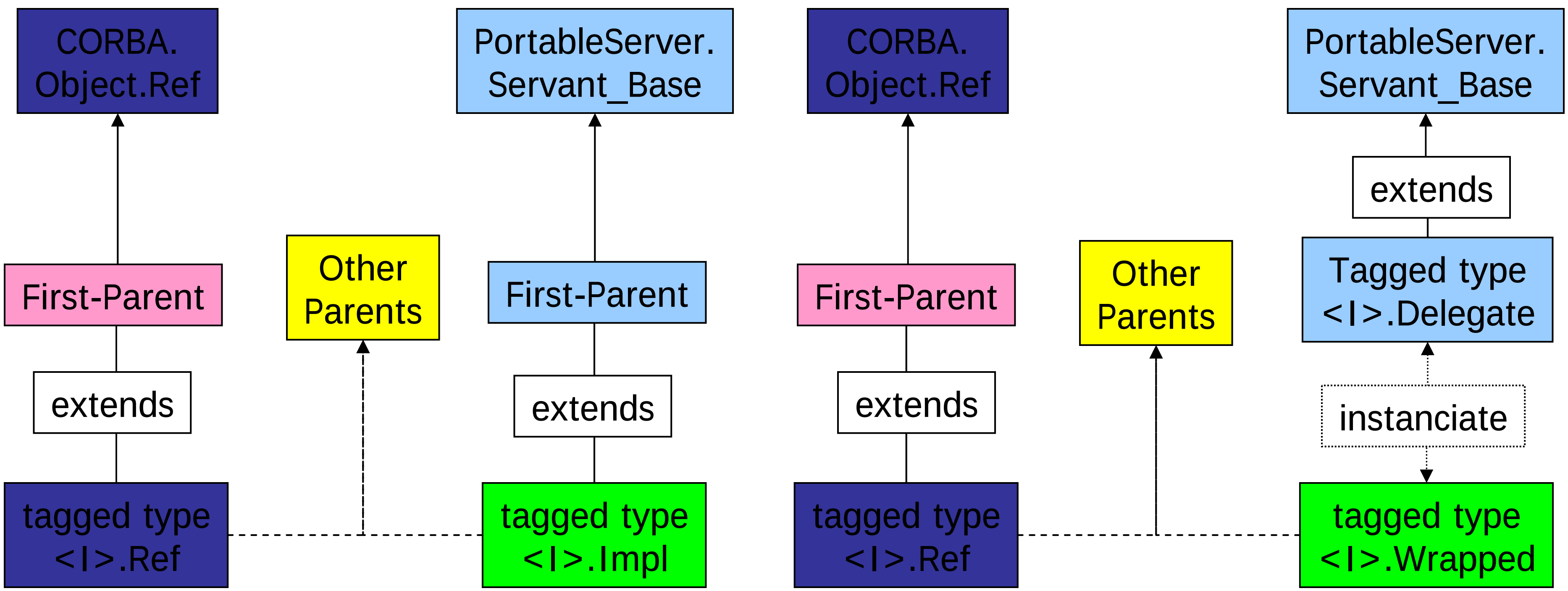
Ada – Client and server main subprograms

```
with myInterface, CORBA;
use myInterface, CORBA;
procedure Client is
   myObject : myInterface.Ref;
begin
   ORB.Initialize ("ORB");
   ORB.String_To_Object
      (To_CORBA_String (Arg), myObject);
   myOperation (myObject);
end Client ;
with PortableServer, Ada.Text_IO, CORBA,
 CORBA.Impl, myInterface,  myInterface.Impl;
use PortableServer, myInterface, CORBA;
procedure Server is
   Root_POA : POA.Ref;
   Ref : CORBA.Object.Ref;
   Obj : CORBA.Impl.Object_Ptr;
```

```
procedure Server is
   Root_POA : POA.Ref;
   Ref : CORBA.Object.Ref;
   Obj : CORBA.Impl.Object_Ptr;
begin
   ORB.Initialize ("ORB");
   Obj := new myInterface.Impl.Object;
   Root_POA := POA.To_Ref
     (ORB.Resolve_Initial_References("RootPOA"));
   POAManager.Activate
     (POA.Get_The_POAManager (Root_POA));
   Ref := POA.Servant_To_Reference
     (Root_POA, Servant (Obj));
   Ada.Text_IO .Put_Line (To_Standard_String
     (Object.Object_To_String (Ref)));
   CORBA.ORB.Run;
end Server;
```

Ada mapping – Implementations

- Implementation inherits from first parent
- Deletage impl provided as generic wrapper



ORB – Interoperable object reference

Type identifier **(*in interface repo.*)**	*Protocol* *Host, port*	*Object key* (***OA + object ID***)

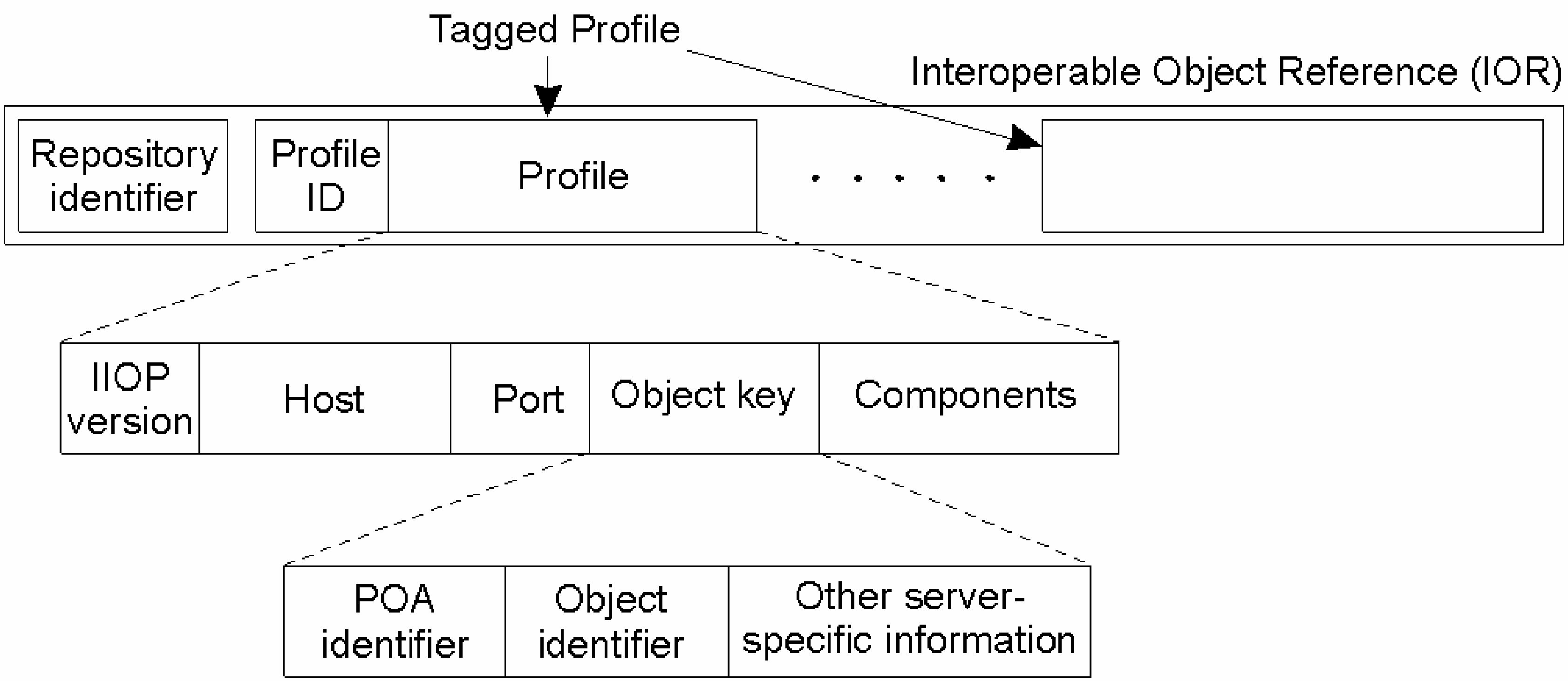
Protocol info **ex**: *IIOP v1.0, somehost.example.net:5555*

Object adapter path and object ID

ex: *"OA007/OA009/obj001"*

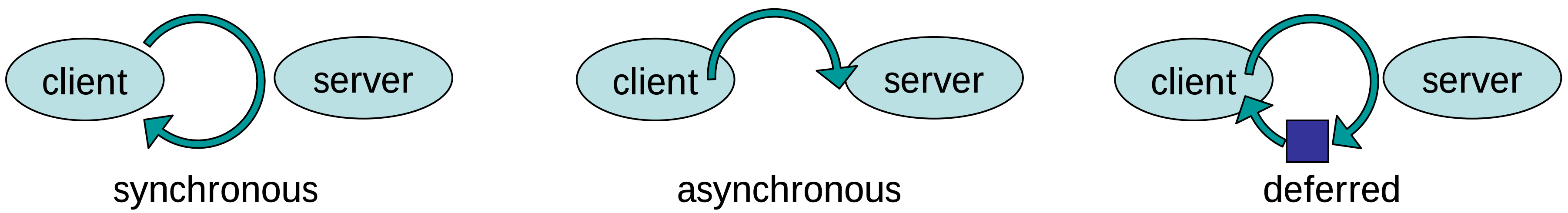
Repository ID **ex**: *"IDL:myObject:1.0"*

ORB – Reference, profile, key



ORB - GIOP

- **Operation invocation modes**



- **GIOP – General Inter-ORB Protocol**
- **Must be mapped onto a transport mechanism:**
  - GIOP over TCP → IIOP
  - GIOP over UDP multicast → MIOP
- **Associated to reresentation syntax**
  - CDR (Common Data Representation)

ORB - GIOP

- **Protocol to convey request across ORB boundaries**
  - Request
  - Reply
  - Cancel Request
  - Locate Request
  - Locate Reply
  - CloseConnection
  - MessageError
  - Fragment
- **Mappings over transport**
  - Usually connected (typical: IIOP)...
  - But not always (with restrictions): MIOP

ORB - CDR

- **Data syntax for GIOP**
  - Supports both big-endian and little-endian representations
  - Sender may use its own representation
  - Differs from XDR (which forces big-endian order)
- **Specifies data aligment**
  - Structures have maximal alignment for their members:

Alignement	Type
1	char, octet, boolean
2	(unsigned) short
4	(unsigned) long, float, enum
8	(unsigned) long long, (long) double

ORB – Object adapters

- **Implementation objects register with the OA**
- **OA generates and interprets object references**
- **Locates implementation from reference info**
- **Activates and deactivates implementations**
- **Performs upcalls**
- **Several adapter classes with different customizable policies:**
  - BOA – Basic Object Adapter (4 activation modes)
  - POA – Portable Object Adapter (7 customizable steps, each with 2/3/4 operation modes)

ORB – Basic Object Adapter

- First OA defined
- Flat namespace
- Server activation policies:
  - **Shared server**
    - Multiple active objects share the same server. The server services requests from multiple clients. The server remains active until it is deactivated or exits.
  - **Un-shared server**
    - Only one object is active in the server. The server exits when the client that caused its activation exits.
  - **Per-method server**
    - Each request results in the creation of a server. The server exits when the method completes.
  - **Persistent server**
    - The server is started by an entity other than the BOA (application developer, operating services, etc.). Multiple active objects share the

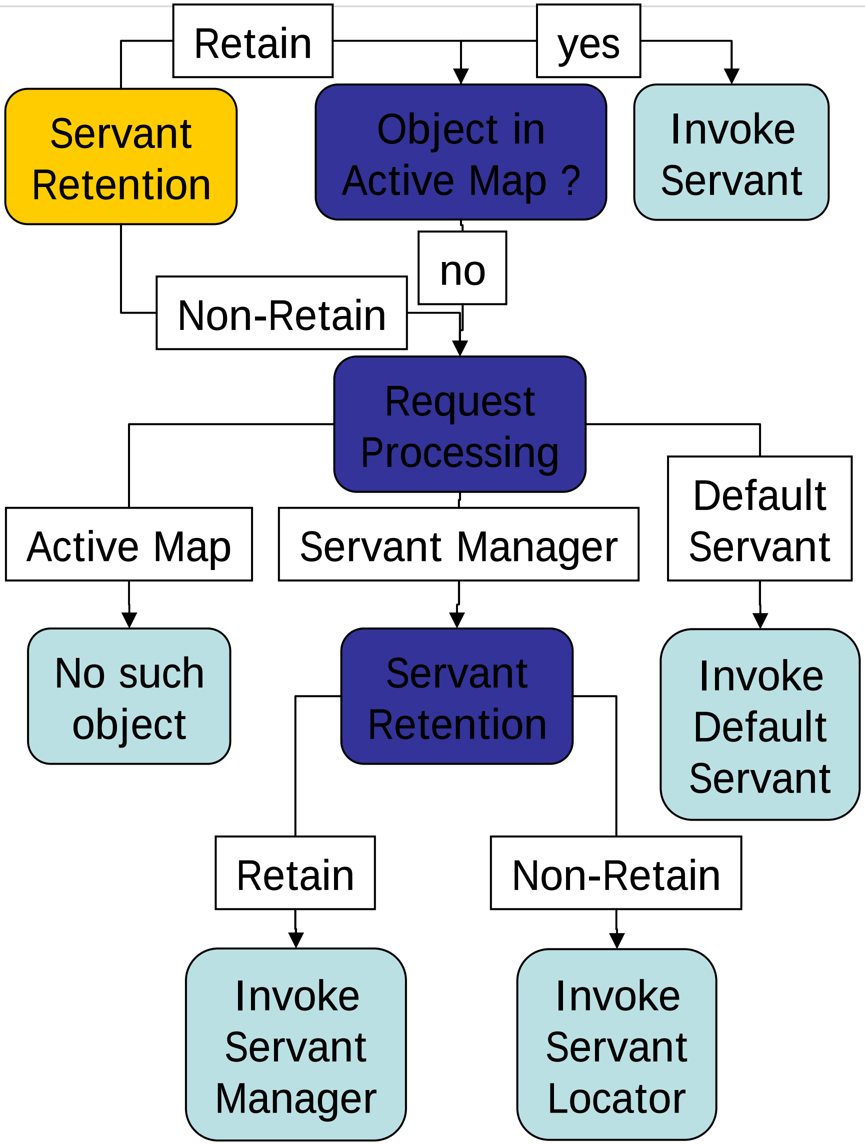
AdaCore
The GNAT Pro Company

ORB – Portable Object Adapter

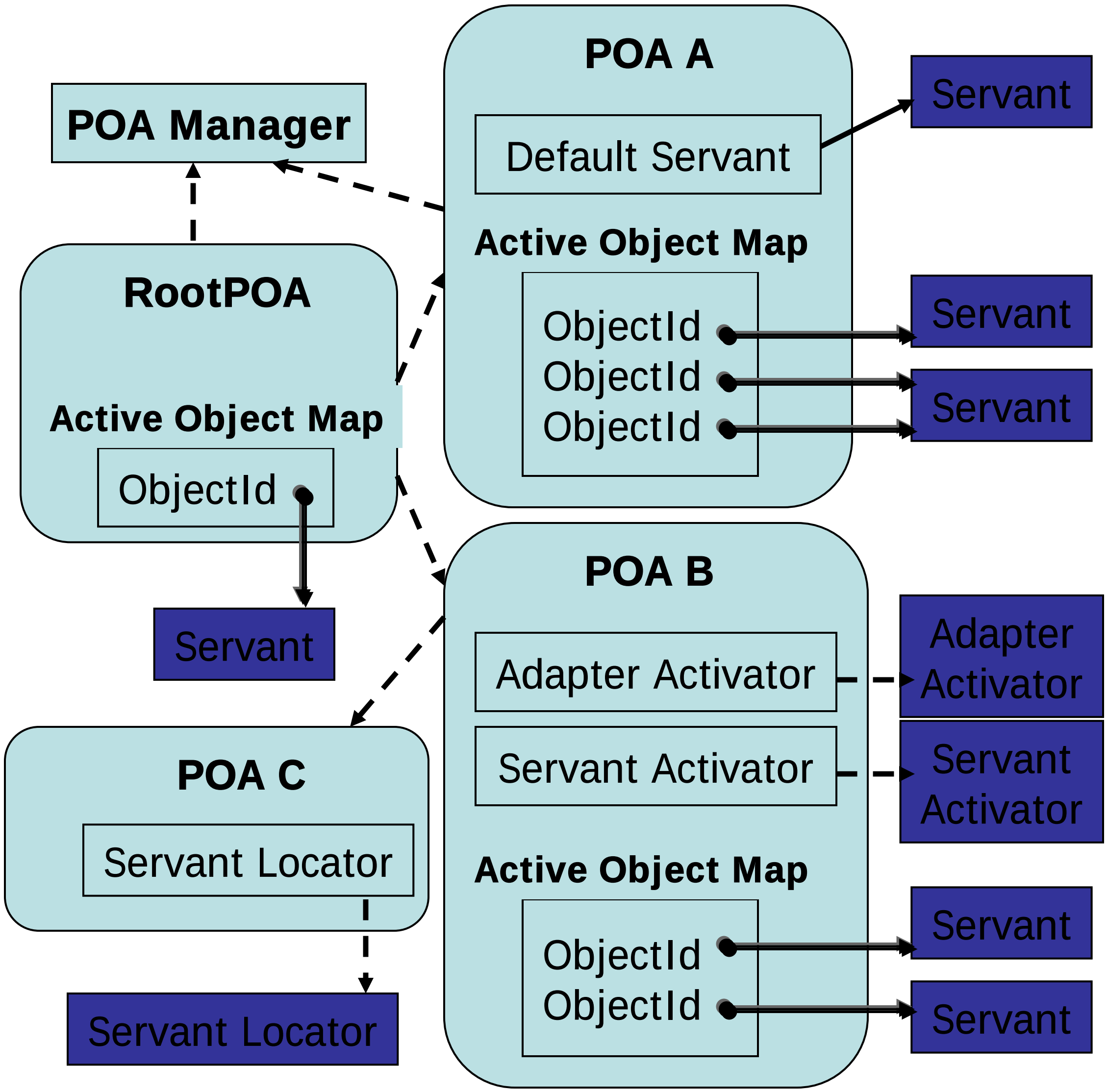
- Enriches and refines initial OA specification
- Hierarchical namespace (tree of POAs)
- Object key contains “tree path” + object id within tree node
- Dynamic, flexible association between object references and implementation entities (*servants*) (unlike BOA)
- Implementation instantiation and reference creation may be decorrelated
- Association can be dynamically controlled by ServantLocator or ServantActivator

ORB – POA policies

- **Thread policy**
  - Creation and management of execution threads
- **Lifespan**
  - Life cycle of implementation object
- **Oid uniqueness**
  - One servant may implement multiple objects
- **Servant retention policy**
  - Caching of reference/servant association
- **Request processing policy**
  - Creation of missing servants
- **Implicit activation policy**
  - Activation of servant upon creation of a reference



ORB – POA operation



- POA manager controls POA states
- Adapter activator creates missing POA nodes
- Default servant acts as fallback
- Servant manager dynamically associates servant
  - ServantLocator finds servant
  - ServantActivator

CORBA – Common object services

- COS Naming, LifeCycle, Events
- COS Transactions, Concurrency, Externalization, Relationship
- COS Security, Time
- COS Property, License, Query
- COS Trading, Object Collections

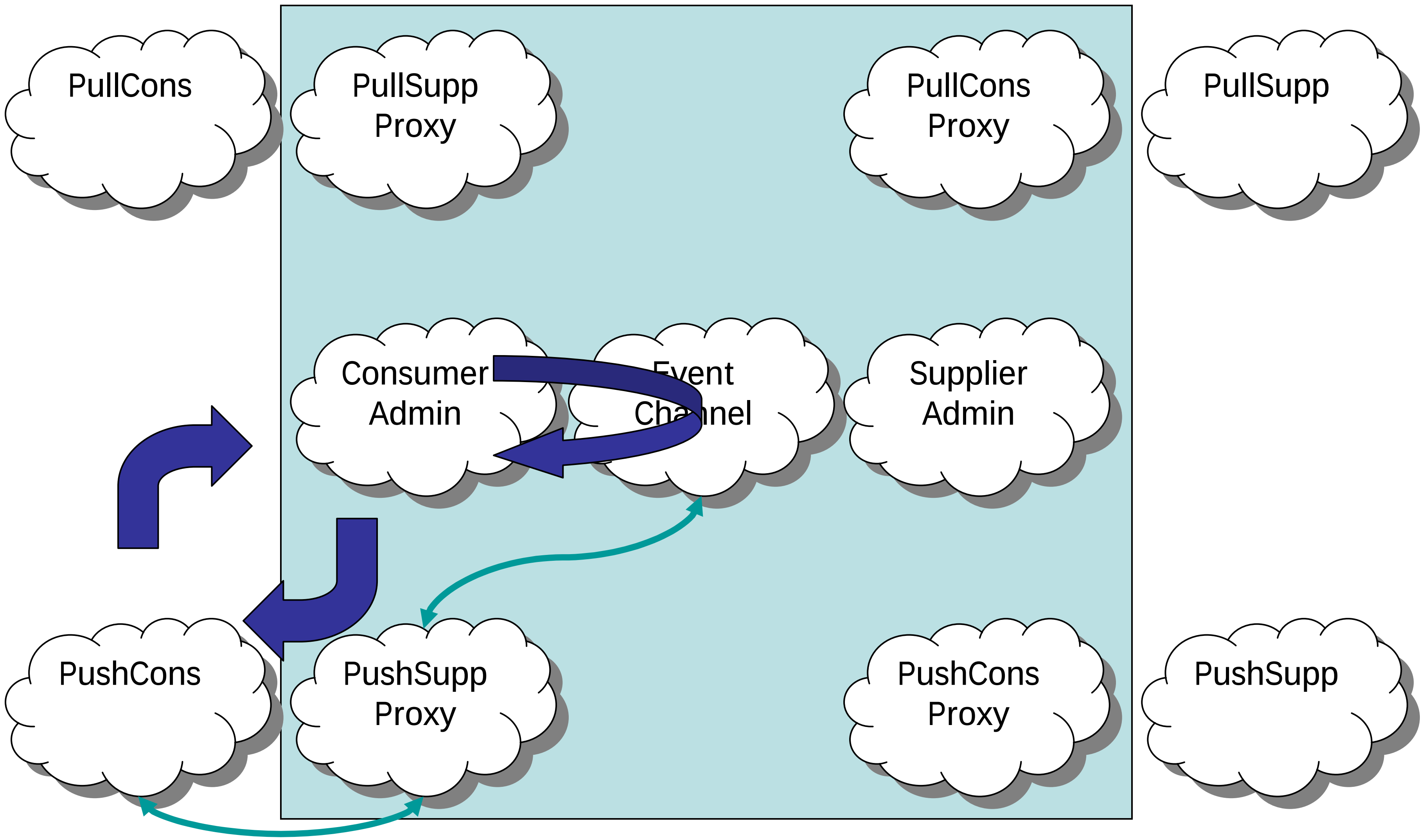
CORBA – COS Naming

- **COS Naming provides**
  - Association between name and object reference
  - Contexts containing associations and (references to) other contexts
  - Accessors for associations and contexts
- **Organisation**
  - Similar to directory tree
    - Contexts ↔ directories
    - Associations ↔ files
  - But more general
    - Cycles are allowed

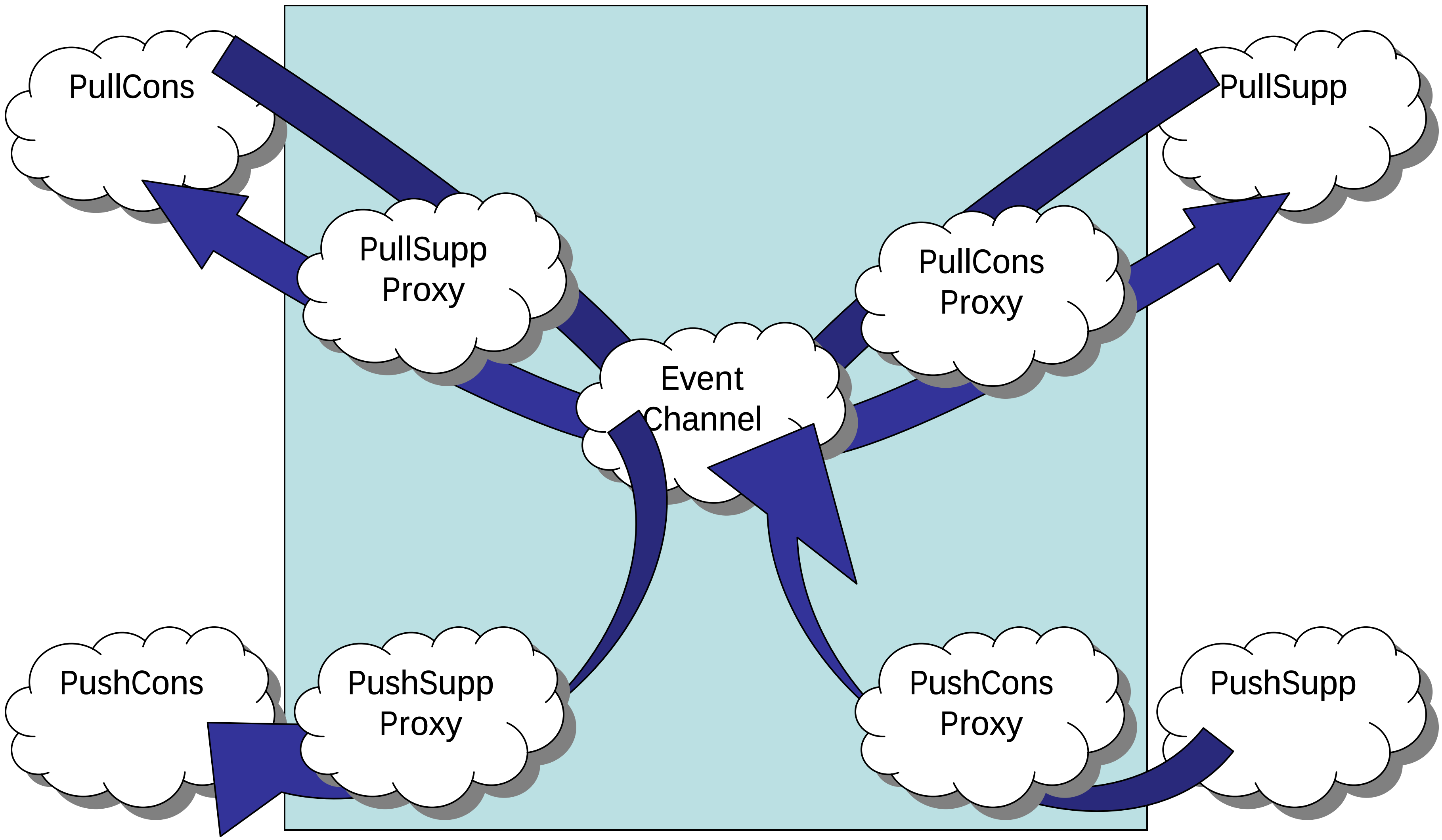
CORBA – COS Events principles

- **Publish / subscribe communication between:**
  - Event producers
  - Event consumers
- **Two connection models:**
  - Push
  - Pull
- **Allows decoupling of communication:**
  - Client and server do not know each other
  - Client and server need not be active simultaneously (asynchronous delivery)

CORBA – COS Events architecture



CORBA – COS Events operation



CORBA – COS Events roles

Rôle	Action	Description
PushSupplier *active*	Invokes push on proxy	Producer sends event to channel
PullSupplier *passive*	Provides pull to proxy	Proxy waits for producer to send event, and publishes it on channel
PushConsumer *passive*	Provides to proxy	Proxy invokes push on consumer upon event arrival
PullConsumer *active*	Invokes pull on proxy	Proxy unblocks consumer pull call upon event arrival

Consumer	Supplier	
	Push	Pull
Push	Notifier	Agent
Pull	Queue	Procurer

CORBA – PolyORB technology

- **PolyORB neutral core**
- **CORBA application personality library**
  - Implements standard CORBA ORB API
- **GIOP protocol personality library**
  - Implements IIOP, MIOP
- **IDLAC IDL to Ada compiler**
  - Implements standard Ada 95 mapping
- **CORBA COS servers:**
  - Naming
  - Events
  - Time
- **Interface repository**

CORBA - Summary

- **CORBA facilitates distribution**
  - Provides tools to simplify implementation of distributed apps
  - Offers system and language interoperability
  - Leverages existing technology
- **Drawbacks**
  - Complexity of standards
  - Steep learning curve
- **Moving target**
  - CORBA 3.0 specifications include component model
- **Competition**
  - Message-oriented middlewares (JMS)
  - Distributed langiuages (Java RMI, Ada 95 DSA)

Ada 95 Distributed systems annex

DSA approach - Objectives

- **Seamless integration of distribution in the Ada 95 language**
- **Modularity using language-defined constructs for spec/implementation separation**
  - Packages
  - Tagged types
- **Allow testing of application in non-distributed mode**
- **Standard optional annex of ISO 8652 (annex E)**
- **GNAT provides a conformant implementation (GLADE)**

DSA approach – Fundamental principles

- **Location transparency**
  - Usage of a service is independent of its location
- **Access transparency**
  - Remote services are invoked just as local ones
  - Application can be built in non-distributed mode with no source changes
- **Separation of interface and implementation**
  - Enforced at the usual package boundaries
- **Typed interfaces**
  - All typing properties of the language are strictly preserved

DSA approach – Fundamental DSA notions

- **Categorization pragmas**
  - **Pure > Remote_Types > Remote_Call_Interface**
  - Shared passive
- **Stream-oriented attributes**
  - Default implementations for base types
  - Need to be user-defined for limited and access types
- **PCS**
- **Partitioning facility**

DSA – Declared pure units

- Only data definitions
- No state
- No code
- May appear in the closure of any unit

DSA – Remote types units

- **Transportable data types**
- **Distributed objects**
  - Tagged limited private types
  - General access to classwide types $\rightarrow$ remote access to classwide types
  - Extend dispatching property: the node is determined at run time
- **Dynamic binding**
  - Remote access to subprogram types
- **May appear in the closure of other RT units and of RCI units**
- **May be assigned to several partitions**

DSA – Remote call interfaces

- Remote subprograms (static RPC)
- Can be used to give value to RAS (dynamic RPC)
- Must be assigned to only one partition
- All calls to unit from other partitions are remote
- Procedures or RAS may be declared *asynchronous*
- Local calls may go through the PCS as well (pragma All_Calls_Remote)

AdaCore
The GNAT Pro Company

DSA – Shared passive units

- Shared data repository
- Assigned on a single partition
- No code – may be assigned on “passive” partition
- Accessed transparently by multiple partitions
- PCS provides a consistent view of shared variables

DSA – Types supporting external streaming

- **Data exchanged between partitions must have a global meaning**
- **Embodied by stream attributes: ’Read / ’Write / ’Input / ’Output**
- **Nonlimited types have default implementations**
- **Access types issue:**
  - Semantic is local to the partition
  - Default stream attributes do not provide “external” semantics
- **User can always specify external streaming**
  - Attribute definition clauses:

```
type T is limited private;
procedure W (St : access Root_Stream_Type'Class; X : T);
for T'Write use W;
```

DSA – Example situation

```
package Types is
    type  Sensor        is ...;
    type  Temperature   is ...;
end Types;
```

Node 1

```
with Types; use Types;
with Device;
procedure Client is
    S : Sensor := ...;
    T : Temperature := Device.Get_T (S);
    ...
end Client;
```

Node 2

```
with Types; use Types;
package Device is
    function Get_T (S : Sensor)
        return Temperature;
end Device;
```

```
package body Device is
    function Get_T (S : Sensor)
        return Temperature
    is
        ...
    end Get_T;
end Device;
```

DSA – Example implementation

```
package Types is
    pragma Pure
    type  Sensor         is ...;
    type  Temperature  is ...;
end Types;
```

Client

unchanged

Server

```
with Types; use Types;
package Device is
    pragma Remote_Call_Interface;
    function Get_T (S : Sensor)
        return Temperature;
end Device;
```

unchanged

DSA – Development model

- Distribution boundaries ↔ standard Ada abstraction boundaries
- Developers can write Ada code as though non-distributed
- Identify remote/shared entities, add *categorization pragmas*
- Application can still be built and tested as *monolithic system*
- Partitioning using gnatdist, distributed test and deployment

DSA – The GNATDIST language

- **Describes partititions that constitute an application:**
  - Boot partition
  - RCI assignments
  - Main subprogram
  - Operational parameters
- **Gnatdist driver calls compiler for each partition**
- **All code generation goes on behind the scene**
  - Stubs and skeletons for RCI units and remote objects
  - Partition main subprogram

DSA – Gnatdist example

```
configuration Testbed is
   pragma Starter (None);
   ServerP : Partition := (Rci);
   ClientP : Partition := (SP);
   for ClientP'Termination use Local_Termination;

   procedure Noproc is in ServerP;
   procedure Client;
   for ClientP'Main use Client;
end Testbed;
```

DSA – PCS

- **Partition Communication Subsystem**
  - Distribution runtime library
  - Called transparently by compiler-generated code
  - Performs the same role as the CORBA ORB
  - + location of RCI units (akin to naming service)
- **Compiler / PCS interface spec was mandatory in ISO 8652:1995…**
  - Theoretically allowed 3rd party PCS replacement
  - Actually impractical and inflexible
  - Clause changed to "implementation advice" in TC1

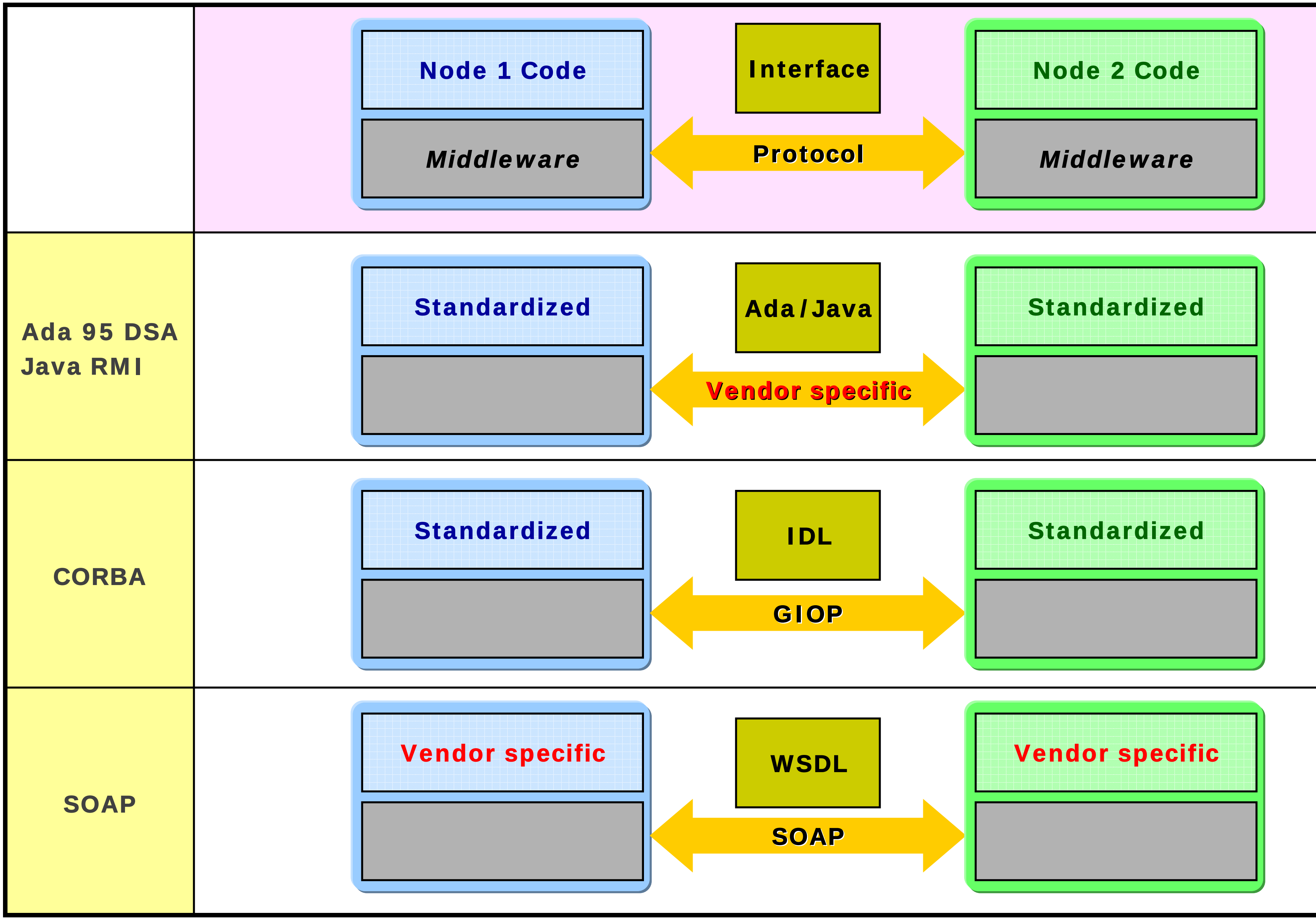
DSA – The PolyORB PCS

- **Legacy GARLIC PCS was closed to outside world**
- **PolyORB**
  + DSA application personality
  + new code gen backend
  → opens distributed Ada to foreign applications
- **Mapping of distributed Ada constructs in a general distributed objects model**
  - RACWs are distributed objects
  - RCIs are distributed *singleton* objects
  - Singleton property is enforced by registration with a name server

AdaCore
The GNAT Pro Company

Getting the best of both worlds

	Language independent	Vendor independent *from the programmer point of view*	Partitioning *distributing services on machines*	IDL	IDL to language mappings	Transport Protocol
Ada 95 DSA Java RMI	NO	YES	Transparent in the code	The language itself	Identity	Vendor dependent
CORBA	YES	YES	Explicit in the code	OMG IDL	Standardized	Standardized
SOAP	YES	NO	Explicit in the code	WSDL	Vendor dependent	Standardized


Node 1 Code
Middleware
Interface
Protocol
Node 2 Code
Middleware
Ada 95 DSA
Java RMI
Standardized
Ada / Java
Vendor specific
Standardized
CORBA
Standardized
IDL
GIOP
Standardized
SOAP
Vendor specific
WSDL
SOAP
Vendor specific

	PROS	CONS
Ada 95 DSA Java RMI	Programmers have a very small learning curve Distribution not hardcoded	Same language for all the nodes Stuck with 1 vendor
CORBA	Many things standardized Language independence Vendor independence	Very steep learning curve for programmers Distribution hard coded
SOAP	Reuses web servers infrastructure (firewalls, web servers, ...)	Learning curve for programmers Strong web service orientation Stuck with 1 vendor

Sectional cover- http

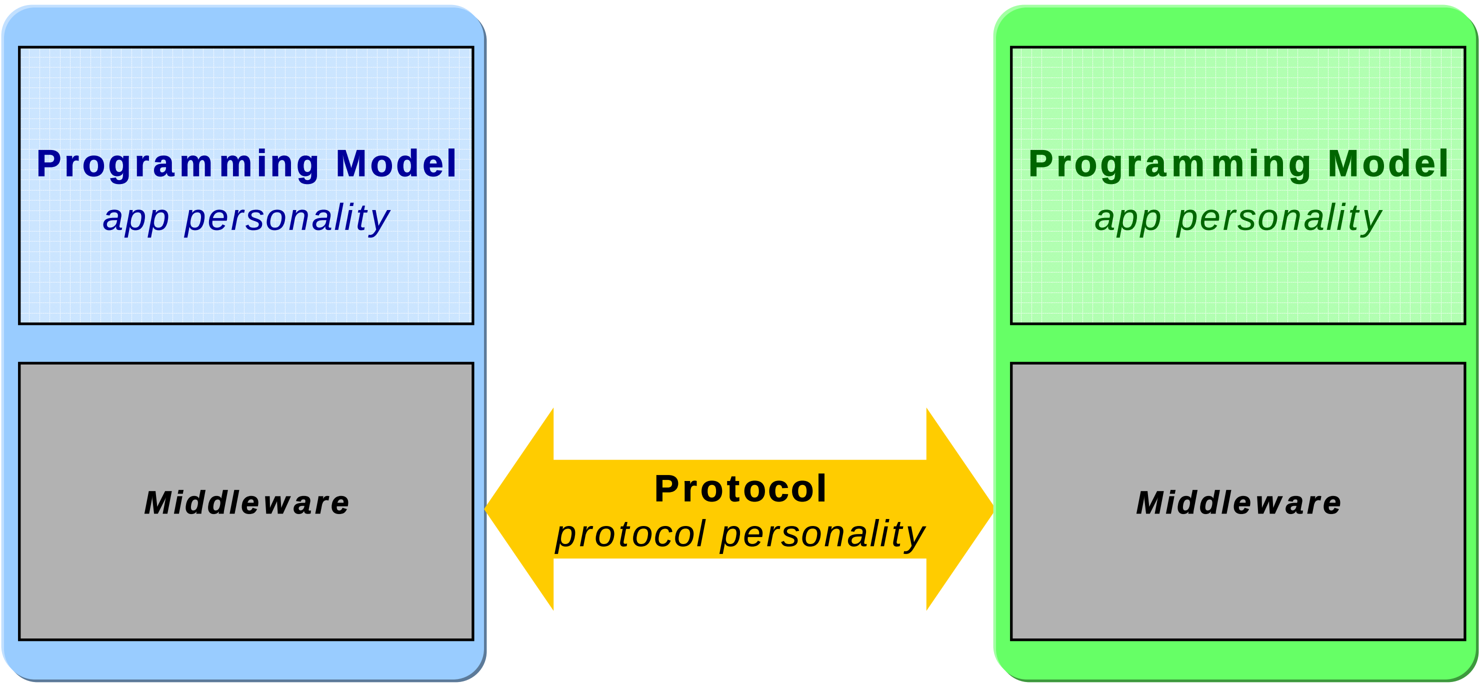
AdaCore
The GNAT Pro Company

Schizophrenic Middleware

Application and protocol personalities

Can we mix programming models and protocols?

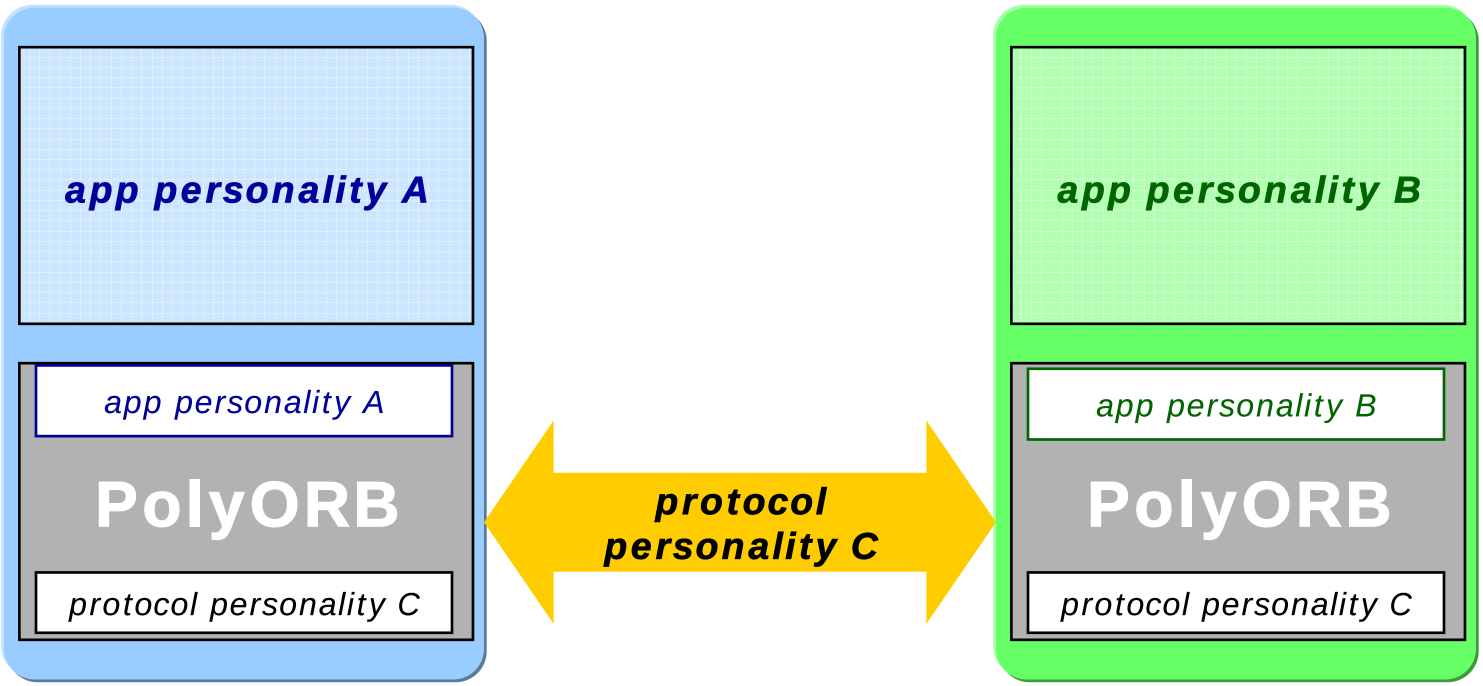
- **Yes with schizophrenic middleware**



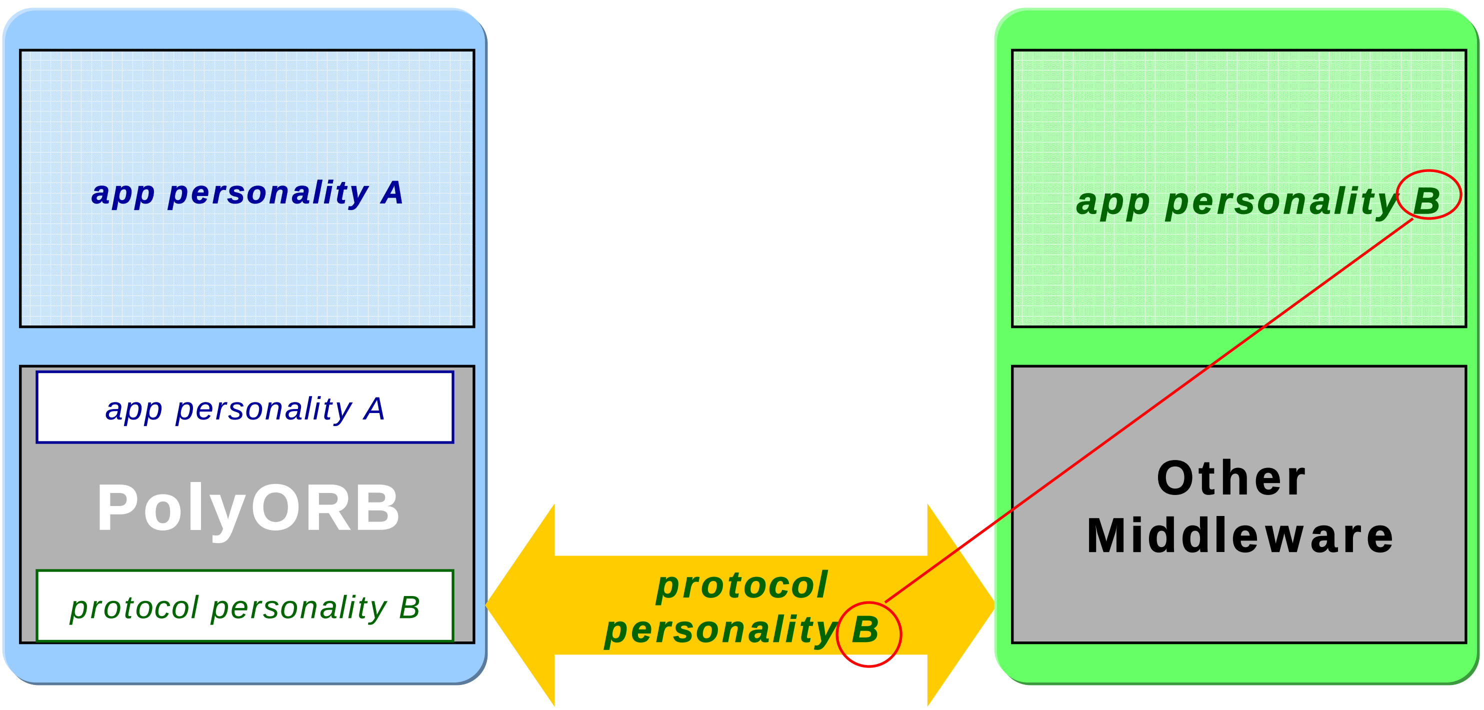
PolyORB – Schizophrenic middleware

With PolyORB you can mix and match

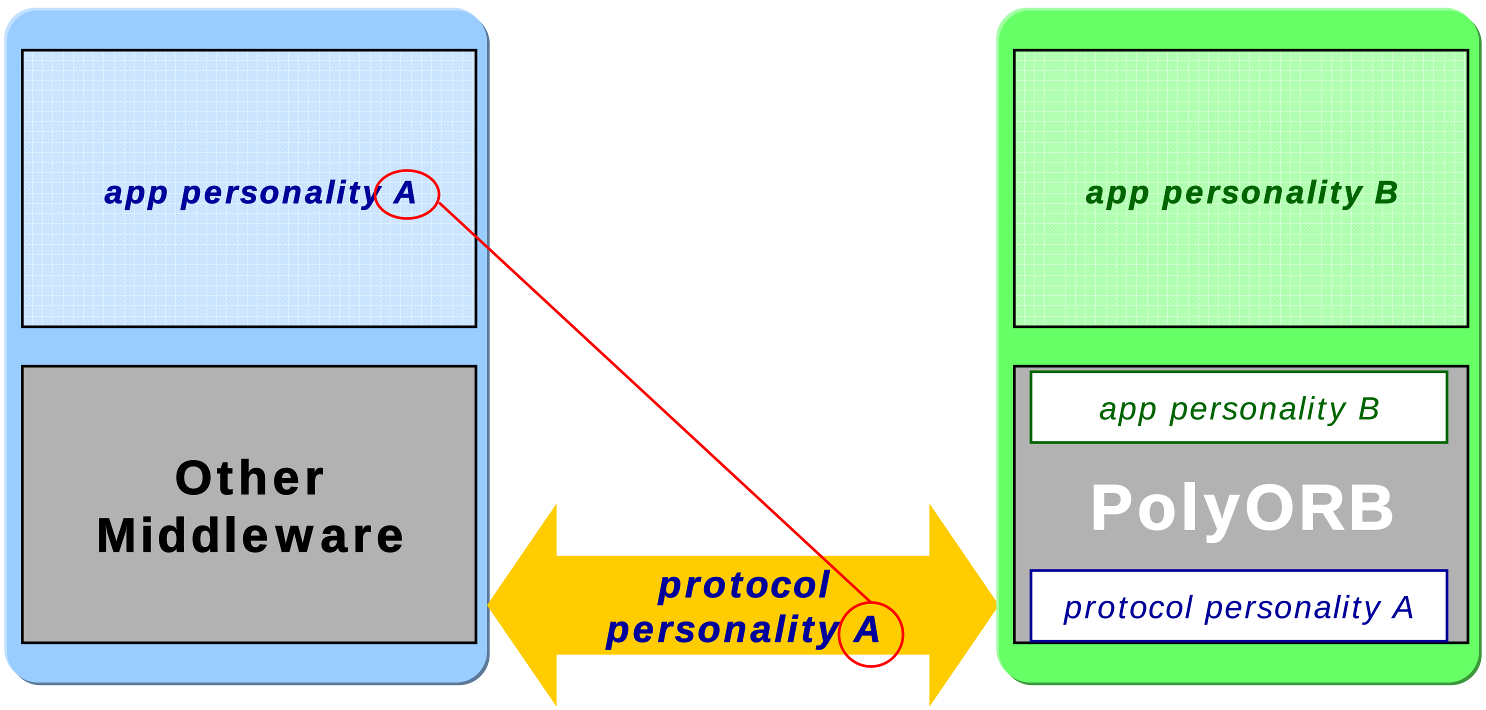
- **Application and protocol personalities**



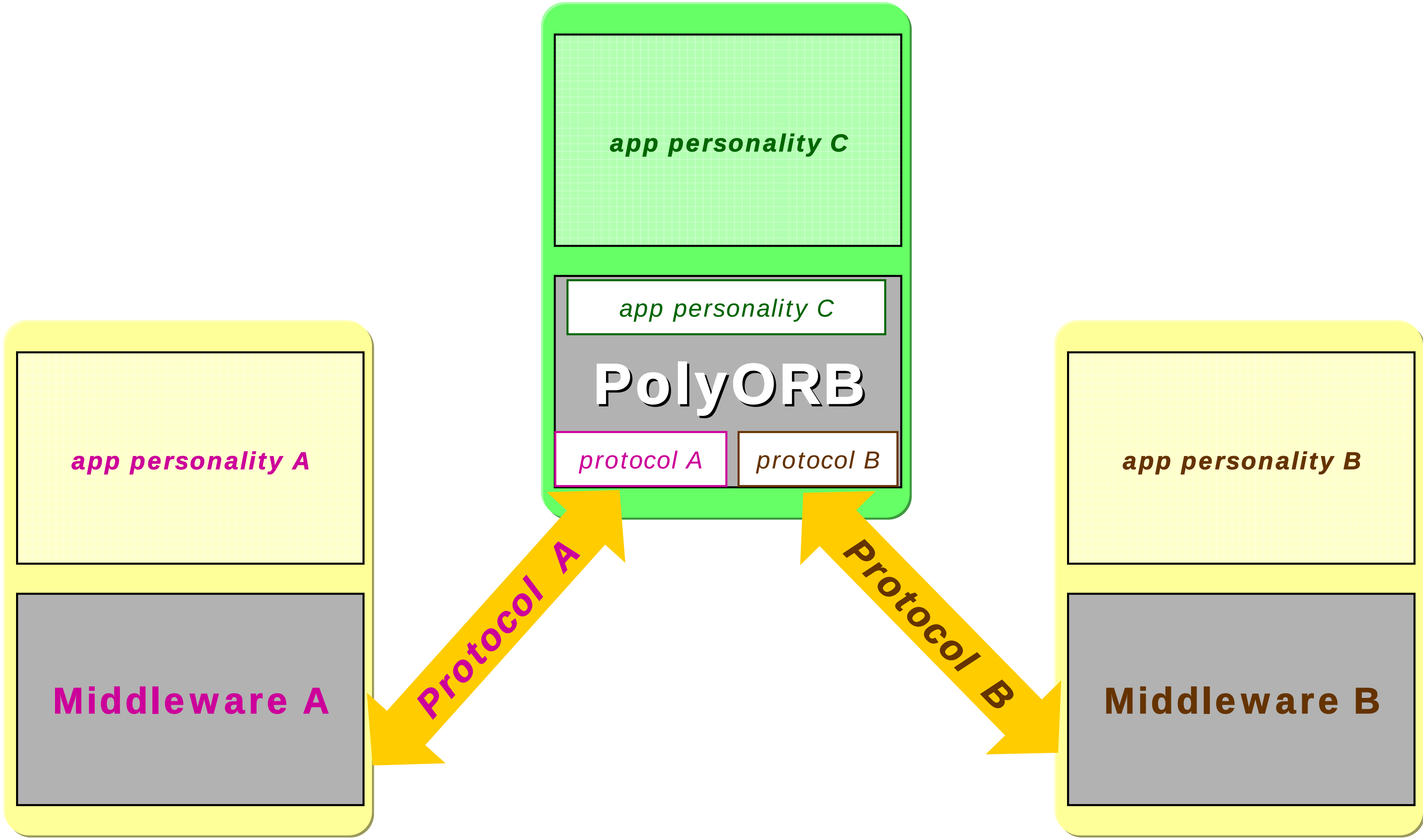
PolyORB - Interoperability (1 of 2)



PolyORB - Interoperability (2 of 2)

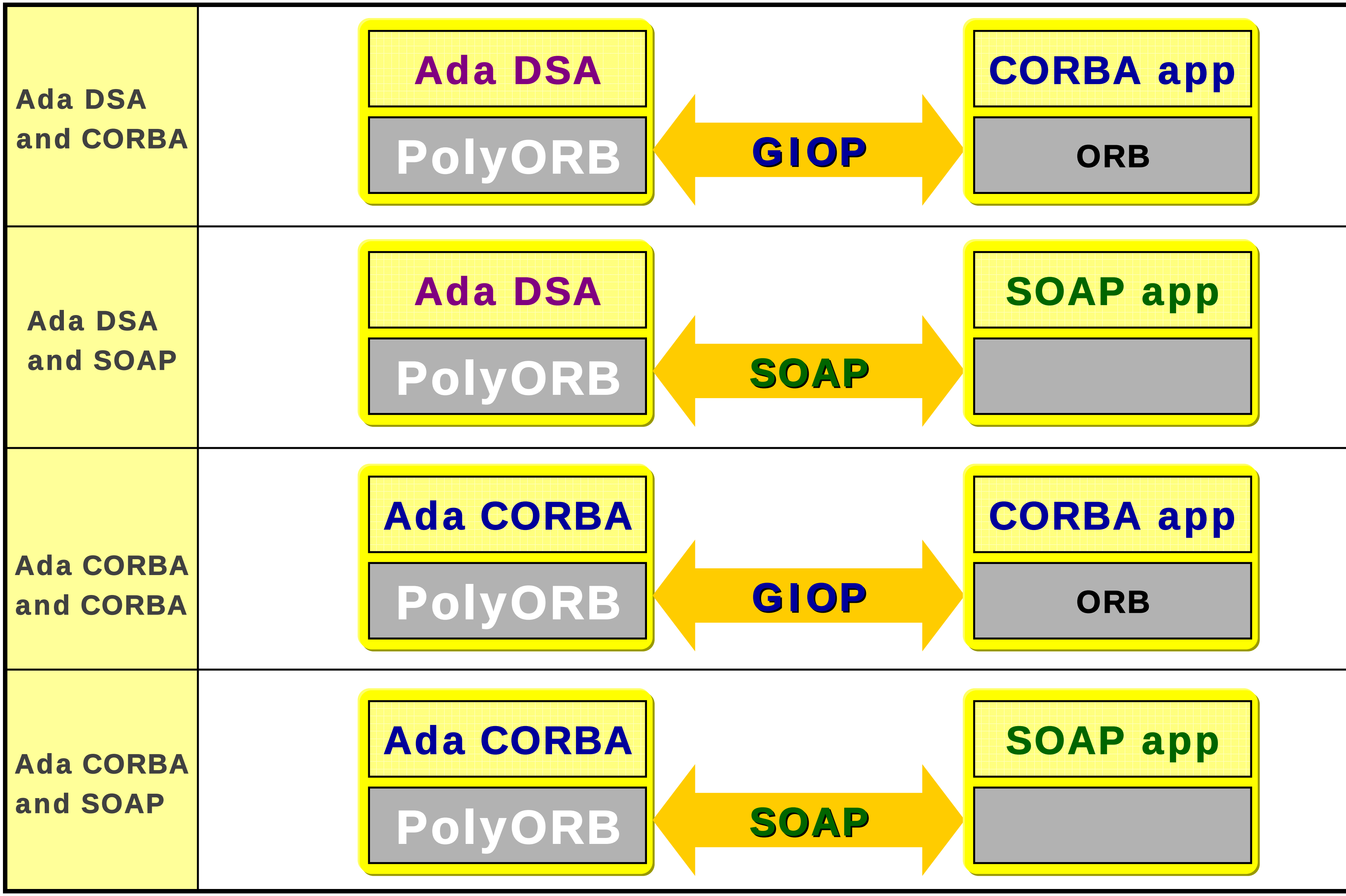


PolyORB – Bridge configuration



Standardized Code and Protocol Personalities

- **Standardized Ada code personalities**
  - Ada 95 DSA (Distributed System Annex)
  - Ada 95 CORBA
- **Standardized protocols**
  - CORBA GIOP
  - SOAP
- **PolyORB supports all 4 combinations**


Ada DSA and CORBA
Ada DSA
PolyORB
GIOP
CORBA app
ORB
Ada DSA and SOAP
Ada DSA
PolyORB
SOAP
SOAP app
Ada CORBA and CORBA
Ada CORBA
PolyORB
GIOP
CORBA app
ORB
Ada CORBA and SOAP
Ada CORBA
PolyORB
SOAP
SOAP app

AdaCore
The Leader in Ada Technology

Conclusion

Distribution technology with Ada

- **CORBA**
  - Integration with multi-language, legacy systems
  - Distribution explicit in source code
  - Steep learning curve
- **Ada DSA**
  - Seamless integration in Ada application
  - Standard does not mandate interoperability
- **Schizophrenic middleware**
  - Decouples application and protocol interface
  - Allows interoperation between the two models
  - Generic, configurable, interoperable
  - Designed to be extensible and customizable
  - *Make the most of both worlds*

Check out PolyORB technology on www.adacore.com