Computer Science Program, The University of Texas, Dallas

Middleware

Distributed Objects and Components

Interoperability

Dynamic Linking

Middleware

NAS
OLE/COM
CORBA

Distributed Objects and Components

- **Stovepipe systems**
  A stovepipe system is a set of legacy applications
  that resists adaptation to user and organizational needs
  to create client/server solutions

  - monolithic, vertically integrated applications
    include in one tool all the services
    (e.g., for a desktop publishing, include word processing, spreadsheet, database, etc.)
    -> ever-growing tools

  - closed system and custom proprietary solution
    intolerant to outside services

  - little discernible software architecture
    system poorly understood by developers and maintainers
    -> not easily reusable or extensible
    -> slow development and deployment
    -> expensive maintenance and evolution

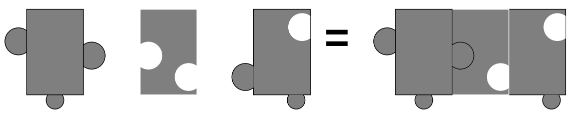
- **Architectural focus!**
  leave out details
  understand the system: components, interfaces, constraints, patterns, & rationale
  keep the size of components manageable and focus
- **Interoperable distributed components**

Distributed Objects and Components

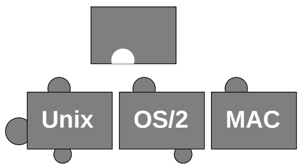
"An object is a living, breathing blob of intelligence that knows how to act in a given situation."
- Steve Jobs

- Client-server can be viewed as the precursor technology to distributed objects
  Objects work together across machine and network boundaries
  to create client/server solutions
- Distributed objects = independent software components
  A component is not bound to any particular
  OSs, HWs, network, tools, PLs, Applications, Vendors

Plug&Play



Portability

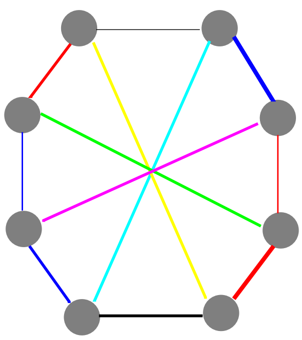


Coexistence (through object wrappers)

Interoperability
(communication (channel & protocol), (interpretable) request for fns & data, data interchange format, data semantics)

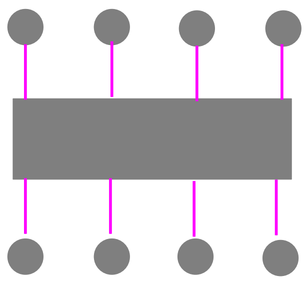
Interoperability

Custom Interface Solutions



O(N^2) Interfaces

Framework-based Solutions
(Good Software Architecture Practices)



O(N) Interfaces

- True database independence will not be possible without 3-tiered architectures
  (Meta Group, 1995)
- Interoperability between N vendor databases ends up eing an N*N problem
  (Mohsen Al-Ghosein, TP Architect, Microsoft, 1995)

Dynamic Linking

```
Module Menu;
   Import DisplayMenu, UserSelection;
   Procedure MenuStatus(...);
   Procedure ProcessMenu(...);
      begin
           DisplayMenu.MainMenu(...);
           MenuStatus(...);
           UserSelection.Accept(...);
           DisplayMenu.MinorMenu(...)
      end
end Menu.
```

```
Module DisplayMenu;
   Procedure MainMenu(...);
   Procedure MinorMenu(...);
end DisplayMenu.
```

```
Module UserSelection;
   Procedure Accept(...);
end UserSelection.
```

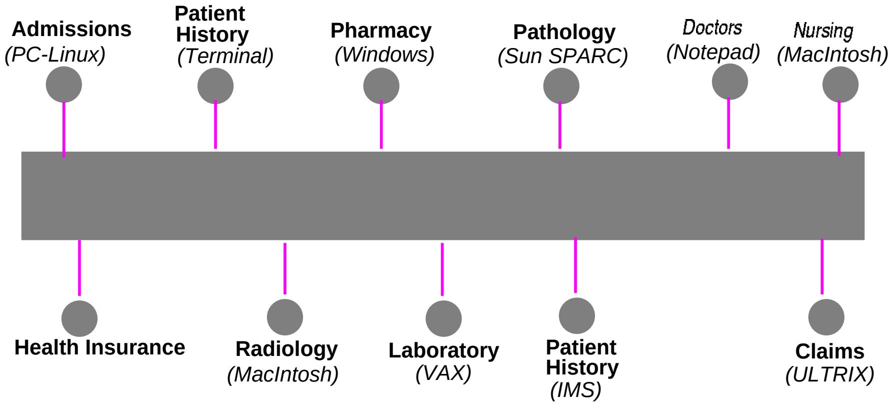
Object module Menu;

Entry table	Object code	Link table
entry0: MenuStatus@addr-1	*/* code for MenuStatus */*	link0: DisplayMenu.MainMenu
entry1: ProcessMenu@addr-2	addr-1: ...	link1: UserSelection.Accept
	/ code for ProcessMenu */*	link2: UserSelection.Accept
	addr-2:	
	addr-3: indirect call via link0	
	relative branch to addr-1	
	addr-4: indirect call vai link1	
	addr-4: indirect call vai link2	

+ *one copy, no recompilation of every component*

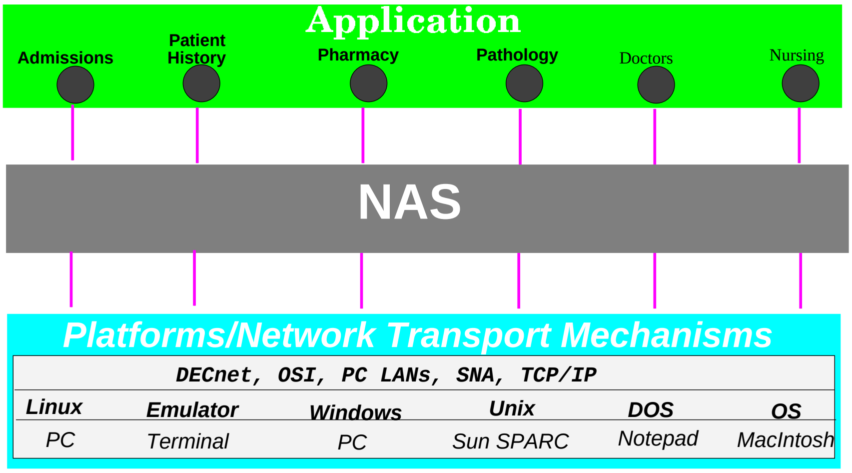
- *slower, type mismatch*

Middleware: An Illustration



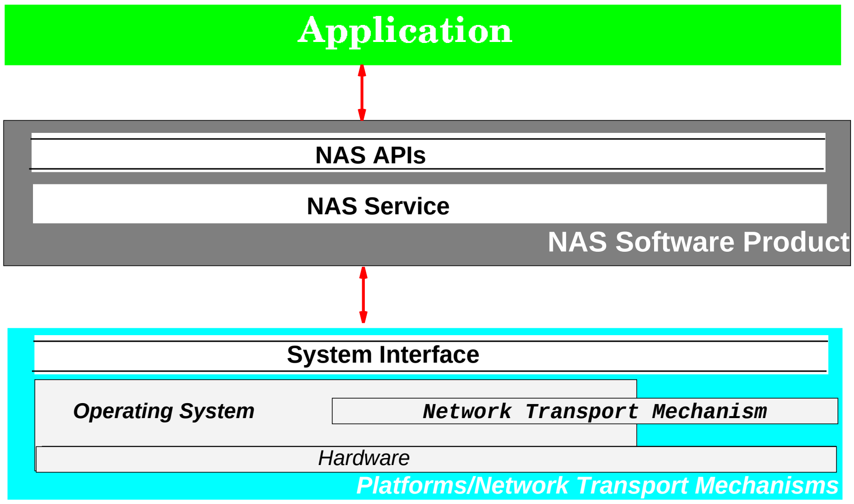
- A system integration
  from many functional areas, for their information processing needs
- Each platform can have a local DB
  E.g., Laboratory stores CAT scan results and other lab graphics;
- Many real benefits:
  reduction in cost
  substantial reduction in time delay between lab examination and doctor's evaluation
  [Fox Chase Cancer Center]
- Access to external services (e.g., Health Insurance) and
  access from external sites (e.g., doctors on trip, at home)

NAS (Network Application Support)



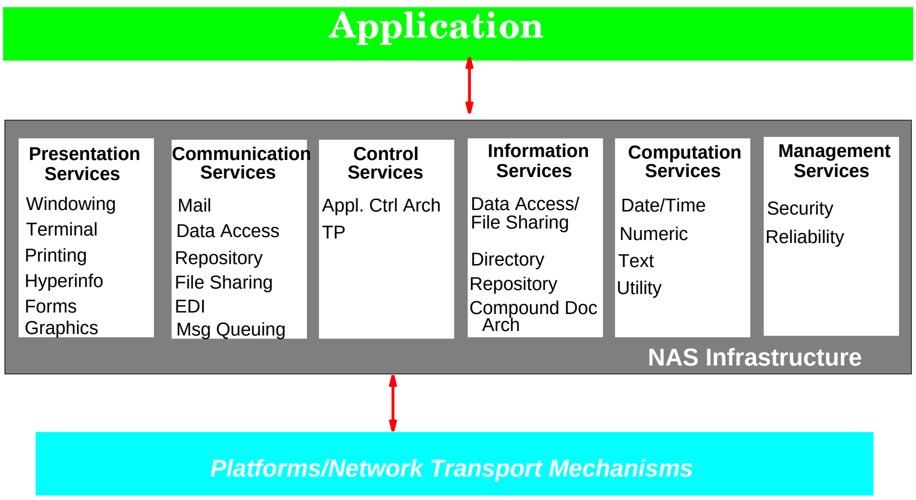
- by DEC towards open systems
  - *application portability*
  - *application interoperability*
  - *user interface portability*
- A (superset) implementation of industry middleware standards of DCE (Distributed Computing Environment) by OSF (The Open Software Foundation)
- Freedom in choosing network architecture, operating system architecture, and hw

NAS Computing Environment



- The NAS API (Application Programming Interface) consists of:
  - *PL Spec:* defines, for each PL, the syntax and semantics of the common requestable fns
  - *Environment Spec:*
    (API service profile) defines which NAS services are available on a particular platform
  - *API Service Spec:*
    defines the syntax and semantics of the functions provided by a specific NAS service
- The NAS system interface (SI) spec. defines how an NAS service accesses the services of OS/Network -> *enhances platform-independence*

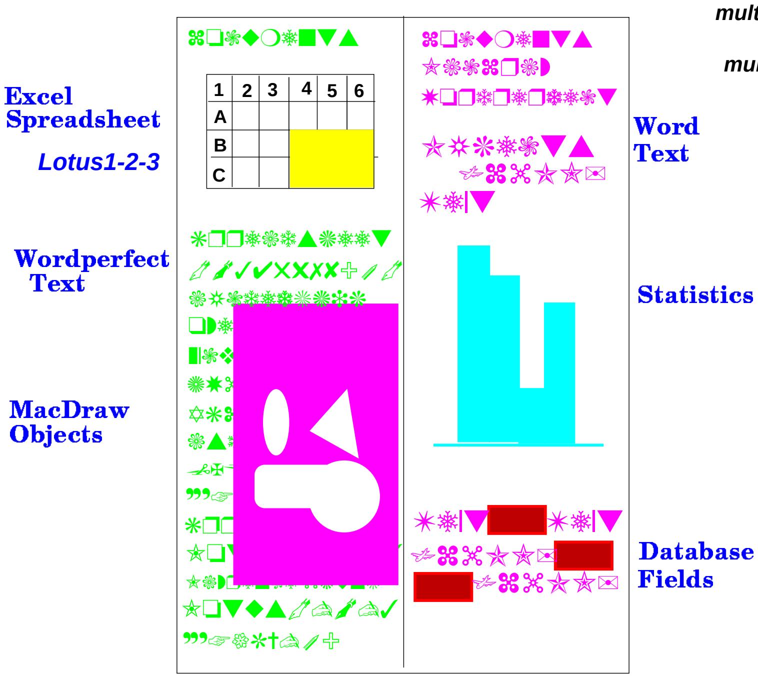
NAS Services



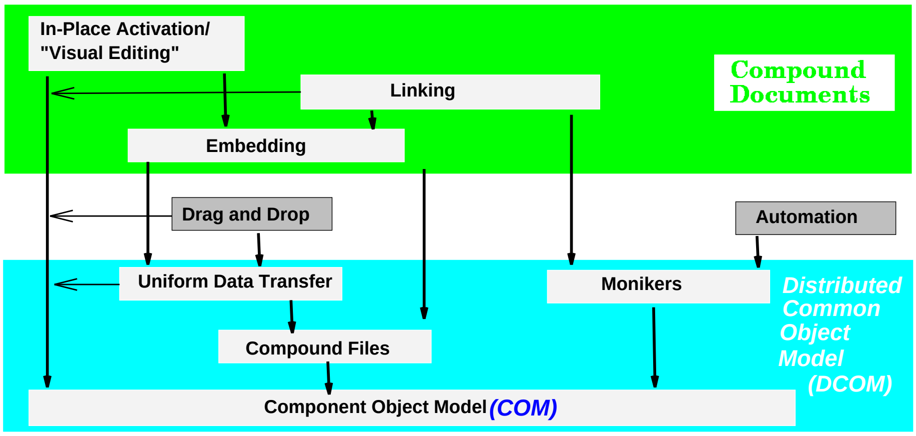
- The NAS API (Application Programming Interface) consists of:
  - *PL Spec:* defines, for each PL, the syntax and semantics of the common requestable fns
  - *Environment Spec:* (API service profile) defines which NAS services are available on a particular platform
  - *API Service Spec:* defines the syntax and semantics of the functions provided by a specific NAS service
- The NAS system interface (SI) spec. defines how an NAS service accesses the services of OS/Network -> *enhances platform-independence*

OLE/COM

Consider desktop publishing
multiple tools
multiple services
multiple documents
multiple objects



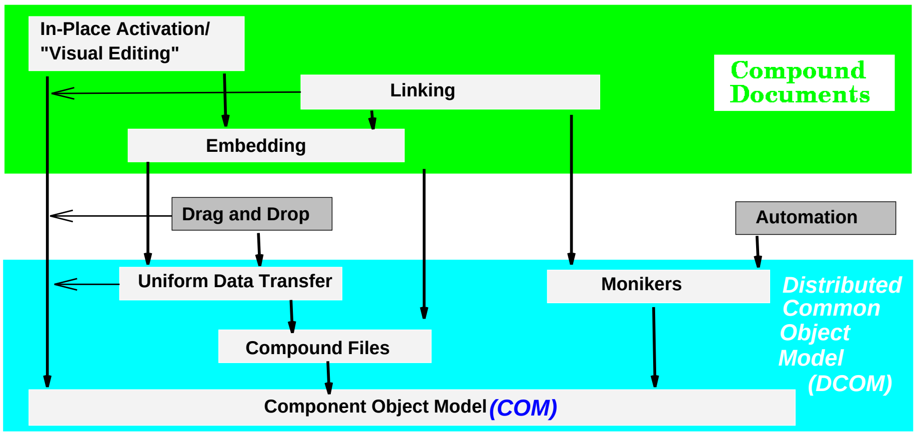
OLE/COM



- OLE2 (Object Linking and Embedding, Version2) is an object-based framework for desktop appln interoperability, but restricted to a single user&machine $\xrightarrow{OSF\ DCE}$ **OLE/DCOM**
- One of the 2 foundational APIs (WIN32 is the other), supported by Windows 3.1, NT, 95 & Cairo, Macintosh platforms

- *(Application) Embedding:*
  - for seamless application integration at the user interface thru "in-place activation";
  - a container application displays components objects from multiple applications (e.g., Word processor, spreadsheet, database, email, web browser, scheduler, multi-media)
  - the container's menus change, according to the user's selection of a component object, to allow the user to edit the object using the component application's operations

OLE/COM

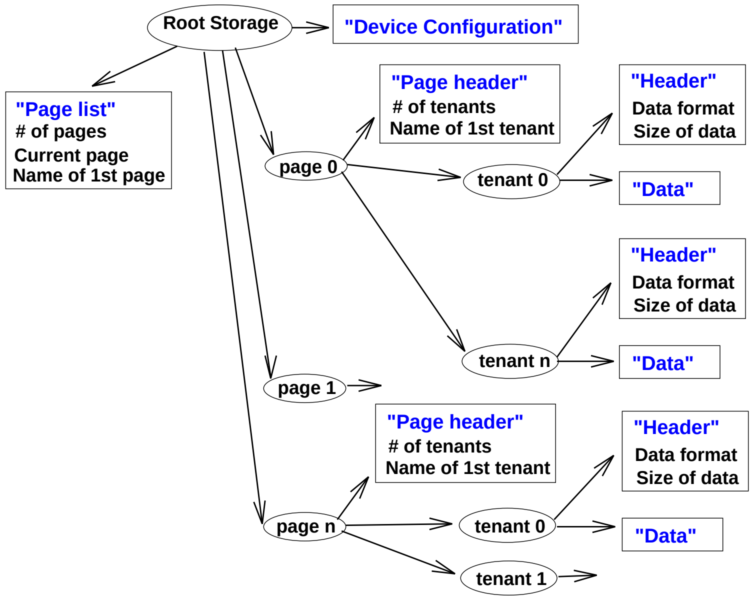


- *Embedding:* for editing objects within a container object without creating a separate appl window
- *Linking:* for the display of common data in multiple documents with updates
- *Drag and Drop:* for dragging & dropping subsets of docs between similarly enabled OLE2 applns
- *Uniform Data Transfer:* a clipboard facility for adding OLE2 data objects to the clipboard
- *Compound Files:* to persistently store multiple objects from multiple applns within a container object
- *COM:* defines the basic interface mechanisms for invoking OLE2 objects
- *Moniker:* a naming facility, supporting linking using file pathnames
- *Automation:* similar to Dynamic Invocation Interface, for controlling appln thru a dispatch fn

OLE/COM

Compound Files

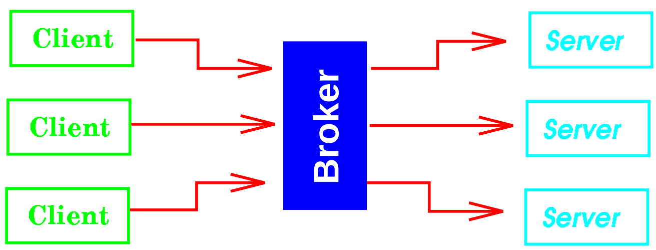
Structured Storage (Data Architecture Specification)



- enable storage & partitioning of complex data from multiple embedded objects into a common file
- created and and managed by OLE2 container objects

CORBA

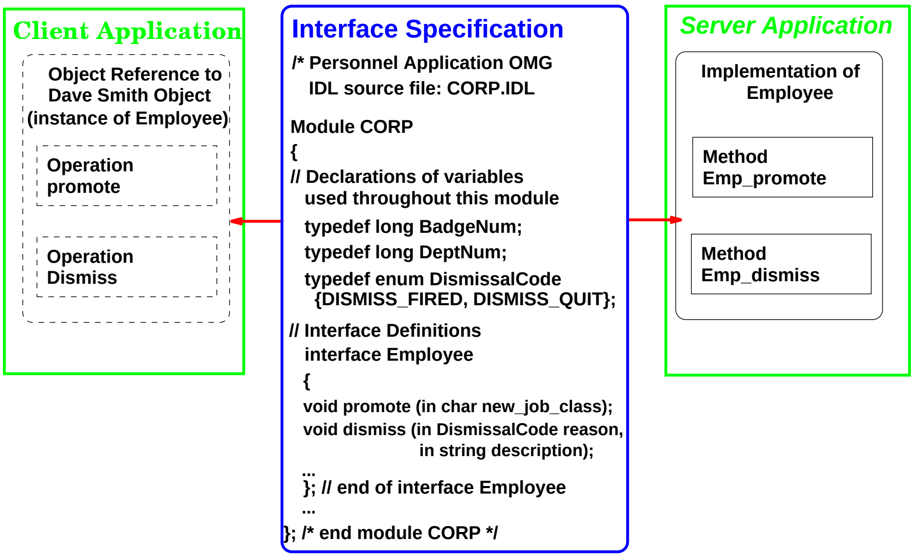
- Common Object Request Broker Architecture



- A specification for a standard OO architecture for applications
  - *not a low-level design/implementation*
  - *platform (OS, HW)-independence, PL-independence*
- defined by the Object Management Group (OMG) since Nov 1990
  currently >500 members
- CORBA clients and servers do not need direct knowledge of each other
  the broker knows the locations and capabilities of the servers on the network
- A client request can be fulfilled by several (competing) servers
  the broker should know who can provide the service fastest and cheapest
- An Object Model requires abstraction, encapsulation, inheritance & polymorphism

"The ability to create simplifying abstractions is a key innate talent of the software architect. Few individuals practicing in the software industry have this ability - perhaps as few as one in five software designers." [Coplien, '94]

OMG Interface Specification



- An OMG IDE file describes the data types, operations, and objects
  - *that the client can use to make a request and*
  - *that a server must provide for an implementation of a given object*
- OMG IDL and C code are very similar in appearance
- The internal representation of of an object reference might differ, but all object references have the same exterrnal representation for a given PL

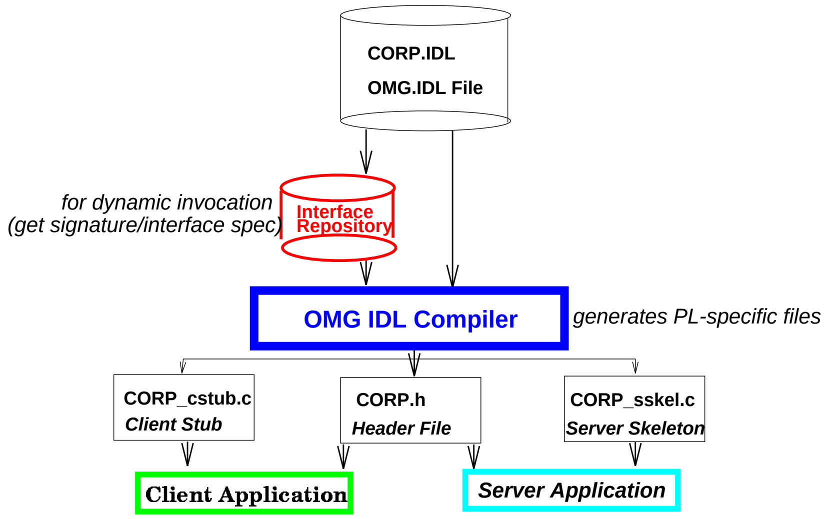
OMG Interface Specification

Interface Specification

```
/* Personnel Application OMG
   IDL source file: CORP.IDL

Module CORP
{
// Declarations of variables
   used throughout this module
   typedef long BadgeNum;
   typedef long DeptNum;
   typedef enum DismissalCode
     {DISMISS_FIRED, DISMISS_QUIT};
// Declarations of data types
   struct DeptInfo
        {
        DeptNum id;
        string name;
        };

// Interface Definitions
  // forward referencing
  interface Employee;
  interface Department
  {
       attribute DeptInfo DeptId;
  readonly attribute Employee manager_obj;
}; // end of interface Department

interface Employee
{      attribute EmpData personal_data;
 readonly attribute Department department_obj;
 void promote (in char new_job_class);
 void dismiss (in DismissalCode reason,
               in string description);
 void transfer (in Department new_dept_obj);
}; // end of interface Employee
interface Manager: Employee
{ void approve_transfer (in Employee employee_obj,
                  in Department current_department,
                  in Department new_department);
}; // end of interface Manager
interface Personnel: Employee
{ Employee hire (in Empdata employee_data,
                 in Department department_obj,
                 out BadgeNum new_employee_id);
}; // end of interface Personnel
interface PersonnelManager: Personnel, Manager
{ void arbitrate ();
}; //end of PersonnelManager
 ...
}; /* end module CORP */
 Module Engineering
   { interface EmployeeLocator
     void FindEngineer (in CORP::BadgeNum id,
                  out CORP::PersonalData info);};
 interface PersonnelManager: CORP::PersonnelManager
     { ... };}; //end module Engineering
```

- *complex data structure, forward referencing, readonly object attributes*
- *single inheritance, multiple inheritance within & across modules*

OMG Interface Specification



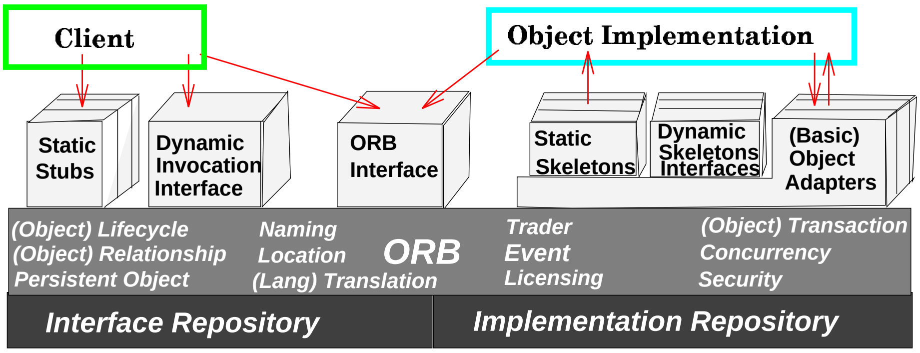
- ***Client Stub***

  one PL-specific routine for each operation (e.g., "hire") in cource code format; compiled & linked into the client application
- ***Header File***

  define data types (structures and constants) to be included into the client/server appln source code
- ***Server skeleton***

  for mapping client operations to methods in the implementation; generated in source code format; compiled and linked into the server application

CORBA Interfaces



- ***Client Stub***

  one PL-specific routine for each operation (e.g., "hire") in cource code format; compiled & linked into the client application
- ***Header File***

  define data types (structures and constants) to be included into the client/server appln source code
- ***Dynamic Invocation Interface***

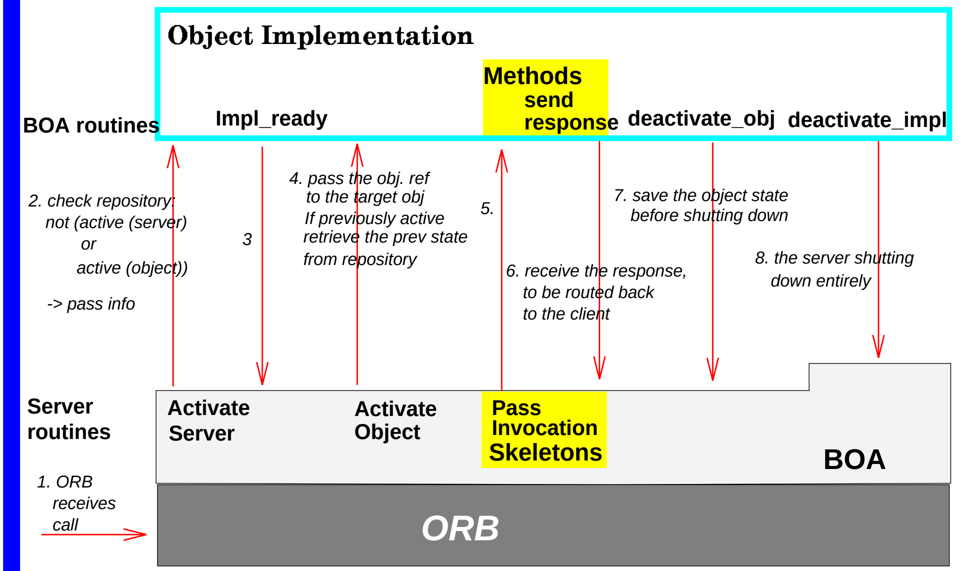
  Unlike Static Invocation Interface (Static Client Stub), DII lets you postpone selection of object type and operation until run time + asynchrous modes
- ***Server skeleton***

  for mapping client operations to methods in the implementation; generated in source code format; compiled and linked into the server application
- ***Dynamic Skeleton Interfaces***

  communicate with remote ORBs
- ***Object Adapters***

  activating and invoking a server (object implementation)

Basic Object Adapters



- *Shared server policy: a single server running a number of objects, one per interface*
  - *+ reduced process-initiation overhead*
  - *- hard to enforce memory and security isolation of one object from another*
- *Unshared server policy: only 1 object of a given impl at a time can be active on a server*
  appropriate when exclusive resource (e.g., a printer)
- *Server-per-method policy: a separate server for each method invocation*
  good for load balancing or isolation of servers from each other (security/administration)

So,

All systems have subsystems and
all systems are parts of larger systems.

The value added by a system must come from
the relationships between the parts,
not from the parts per se.
- The art of systems architecting

Architectural design can, literally, make it or break it.