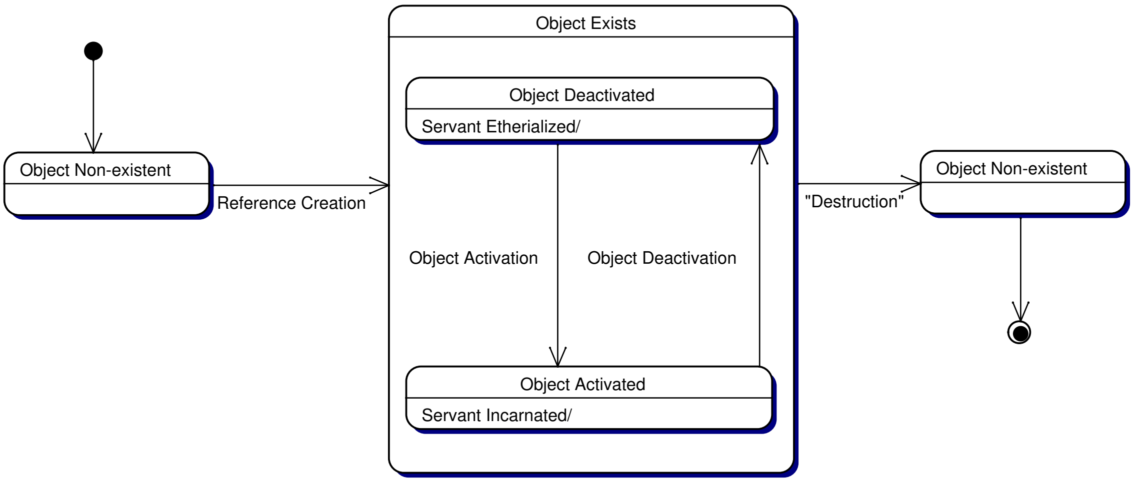
E.9 Portable Object Adaptor

- More complicated activation schemes
- POA interface
- Persistent references

1 CORBA Object Life Cycle

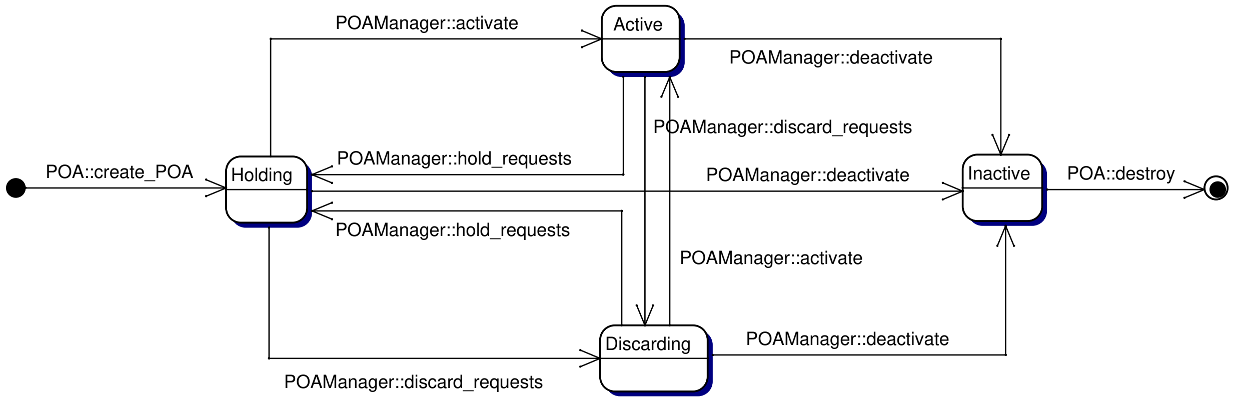
- State diagram for life cycle



◆ OA has to create servant and activate object when call arrives

2 POA Operation

- POA operation controlled by POA Manager
- POA Manager states



- One POA Manager for multiple POAs possible
- PIDL interface **`POAManager`**

3 POA Creation

- IDL interface:

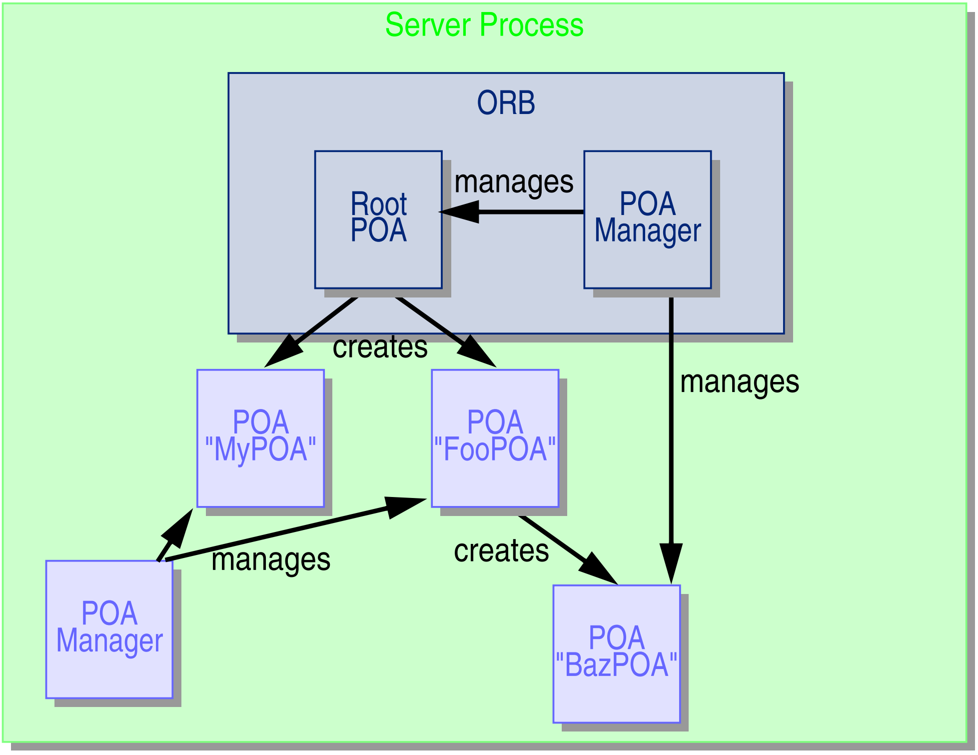
```
module PortableServer {
    interface POAManager;
    exception AdapterAlreadyExists {};
    exception InvalidPolicy { unsigned short index; };

    interface POA {
        POA create_POA(in string              adapter_name,
                       in POAManager          manager,
                       in CORBA::PolicyList   policies )
            raises(AdapterAlreadyExists, InvalidPolicy);
        ...
    };
};
```

- Policies influence POA operation
- Root POA already exists in ORB

```
org.omg.CORBA.Object o = orb.resolve_initial_references(
                                                  "RootPOA" );
org.omg.PortableServer.POA root_poa =
              org.omg.PortableServer.POAHelper.narrow( o );
```

3 POA Creation

- Hierarchy of POAs



4 POA Policies

- IDL:

```
module CORBA {
    typedef unsigned long PolicyType;

    interface Policy {
        readonly attribute PolicyType policy_type;

        Policy copy();
        void   destroy();
    };
    typedef sequence<Policy> PolicyList;
};
```

- Locality-constrained objects

5 CORBA Object Life Span Policy

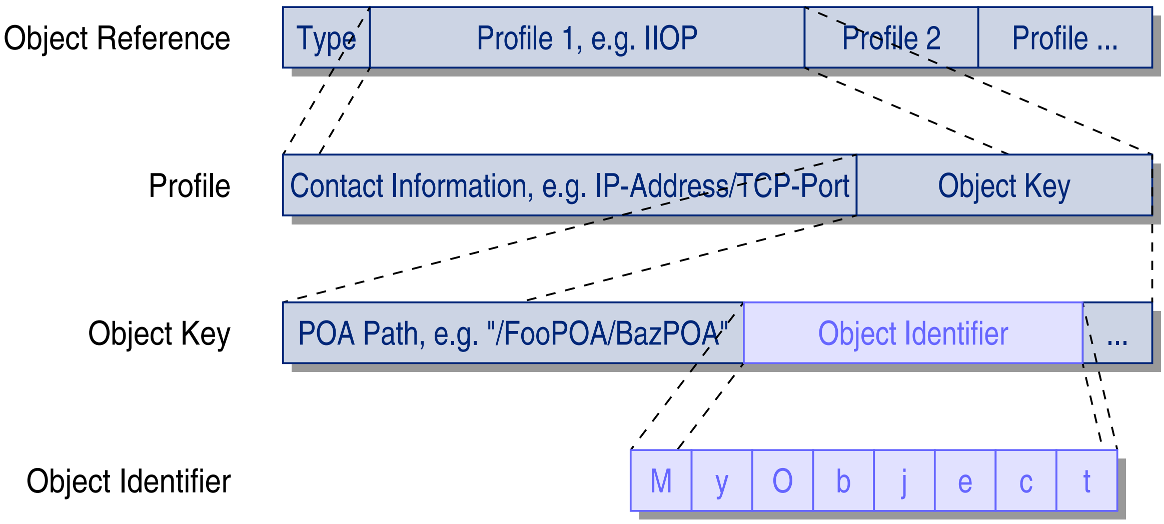
- IDL:

```
module PortableServer {
    enum LifespanPolicyValue {
        TRANSIENT, PERSISTENT
    };
    interface LifespanPolicy : CORBA::Policy {
        readonly attribute LifespanPolicyValue value;
    };
};
```

- Single POA can either support persistent or transient objects, not both
- Persistent objects
  - ◆ ORB and Implementation Repository must keep track of these objects
  - ◆ Additional information for re-activation is stored
- Default Value: **`TRANSIENT`**

6 Object Identifiers

- Objects are identified via object references



- Object IDs either chosen by:
  - ◆ the POA (**`SYSTEM_ID`**)
  - ◆ the application (**`USER_ID`**), e.g. when objects are mapped to a database

6 Object Identifiers

- IDL:

```
module PortableServer {
    enum IdAssignmentPolicyValue {
        SYSTEM_ID, USER_ID
    };
    interface IdAssignmentPolicy : CORBA::Policy {
        readonly attribute IdAssignmentPolicyValue value;
    };
};
```

- Default Value: **`SYSTEM_ID`**

7 Mapping Objects to Servants

- Relation between Object IDs and Servants

- IDL:

```
module PortableServer {
    enum IdUniquenessPolicyValue {
        UNIQUE_ID, MULTIPLE_ID
    };
    interface IdUniquenessPolicy : CORBA::Policy {
        readonly attribute IdUniquenessPolicyValue value;
    };
};
```

- **`UNIQUE_ID`**: One-to-one relation between objects and servants

- **`MULTIPLE_ID`**: There may be several CORBA objects (and therefore several Object IDs) that are implemented by the same Servant

- Default Value: **`UNIQUE_ID`**

8 Implicit Activation

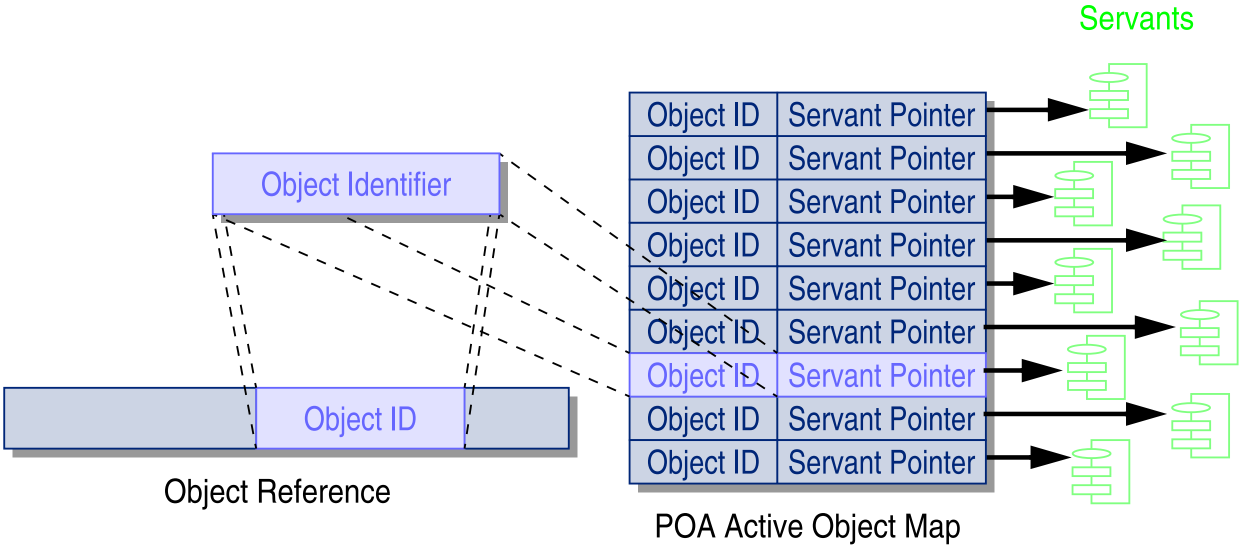
- Activation via some special Skeleton method (in Java **`_this()`**)
- IDL:

```
module PortableServer {
    enum ImplicitActivationPolicyValue {
        IMPLICIT_ACTIVATION, NO_IMPLICIT_ACTIVATION
    };
    interface ImplicitActivationPolicy : CORBA::Policy {
        readonly attribute ImplicitActivationPolicyValue
            value;
    };
};
```

- Default Value: **`NO_IMPLICIT_ACTIVATION`**
- Exception: RootPOA has **`IMPLICIT_ACTIVATION`**

9 Matching Requests to Servants

- POA can store active Servants in an *Active Object Map*



◆ Only objects that are registered in Active Object Map exist (**`USE_ACTIVE_OBJECT_MAP_ONLY`**)

◆ POA throws **`CORBA::OBJECT_NOT_EXIST`** otherwise

9 Matching Requests to Servants

- For dynamic activation application supplies a *Servant Manager* to the POA
  - Servant Manager is contacted if Object ID is not found in Active Object Map
  - Servant Manager may either return a Servant for the object or throw a **CORBA::OBJECT_NOT_EXIST**
  - Mode is **USE_SERVANT_MANAGER**
- The application supplies a Default Servant – there is no Active Object Map
  - All requests are sent to the Default Servant
  - It uses the Dynamic Skeleton Interface (DSI) to process the request
  - Mode is **USE_DEFAULT_SERVANT**

9 Matching Requests to Servants

- IDL:

```
module PortableServer {
    enum RequestProcessingPolicyValue {
        USE_ACTIVE_OBJECT_MAP_ONLY,
        USE_DEFAULT_SERVANT,
        USE_SERVANT_MANAGER
    };
    interface RequestProcessingPolicy : CORBA::Policy {
        readonly attribute RequestProcessingPolicyValue
            value;
    };
};
```

- Default Value: **USE_ACTIVE_OBJECT_MAP_ONLY**

10 Retention of Object ID to Servant Associations

- Should the POA remember Servants and store them in the Active Object Map?
- Yes (**RETAIN**): POA searches Active Object Map
- No (**NON_RETAIN**): POA relies on Default Servant or Servant Manager to provide the association
- IDL:

```
module PortableServer {
    enum ServantRetentionPolicyValue {
        RETAIN, NON_RETAIN
    };
    interface ServantRetentionPolicy : CORBA::Policy {
        readonly attribute ServantRetentionPolicyValue
            value;
    };
};
```

- Default Value: **RETAIN**

11 Multithreading

- How are requests allocated to threads?
- IDL:

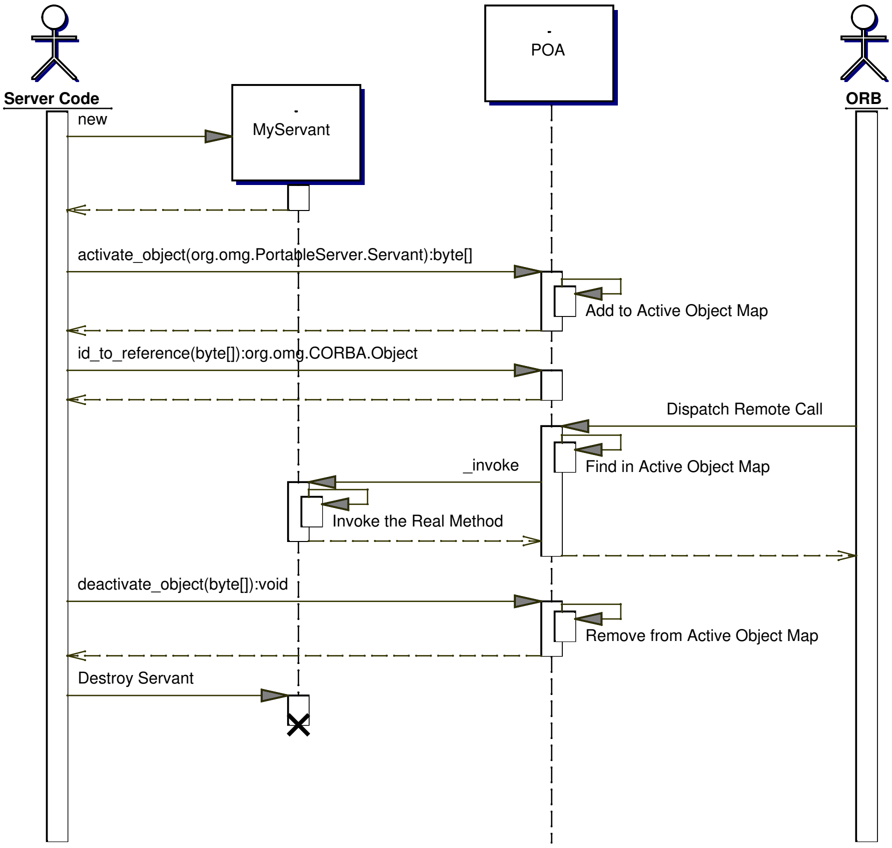
```
module PortableServer {
    enum ThreadPolicyValue {
        ORB_CTRL_MODEL, SINGLE_THREAD_MODEL
    };
    interface ThreadPolicy : CORBA::Policy {
        readonly attribute ThreadPolicyValue value;
    };
};
```

- Default Value: **ORB_CTRL_MODEL**

12 Useful Policy Combinations

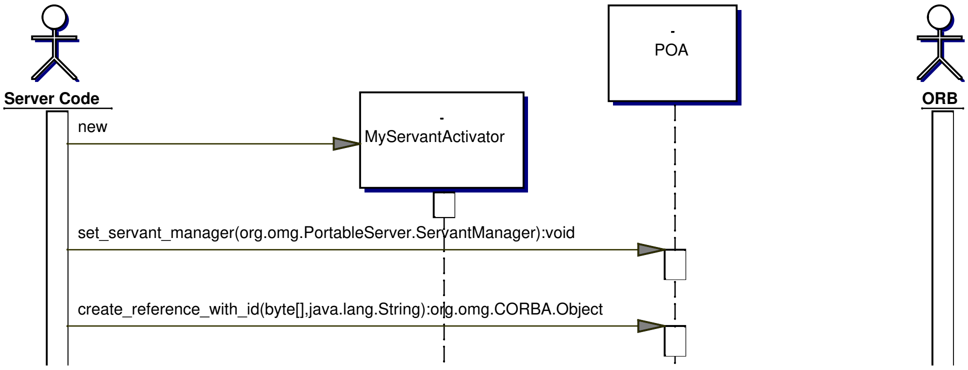
- Some combinations don't make sense and creation of such a POA fails
- **PERSISTENT** often used with **USER_ID**
  - ◆ Easier to re-create servant if ObjectID contains a key to find the servant data
- **IMPLICIT_ACTIVATION** requires **SYSTEM_ID**
  - ◆ Where should the ObjectID come from?

13 Simple POA without Implicit Activation

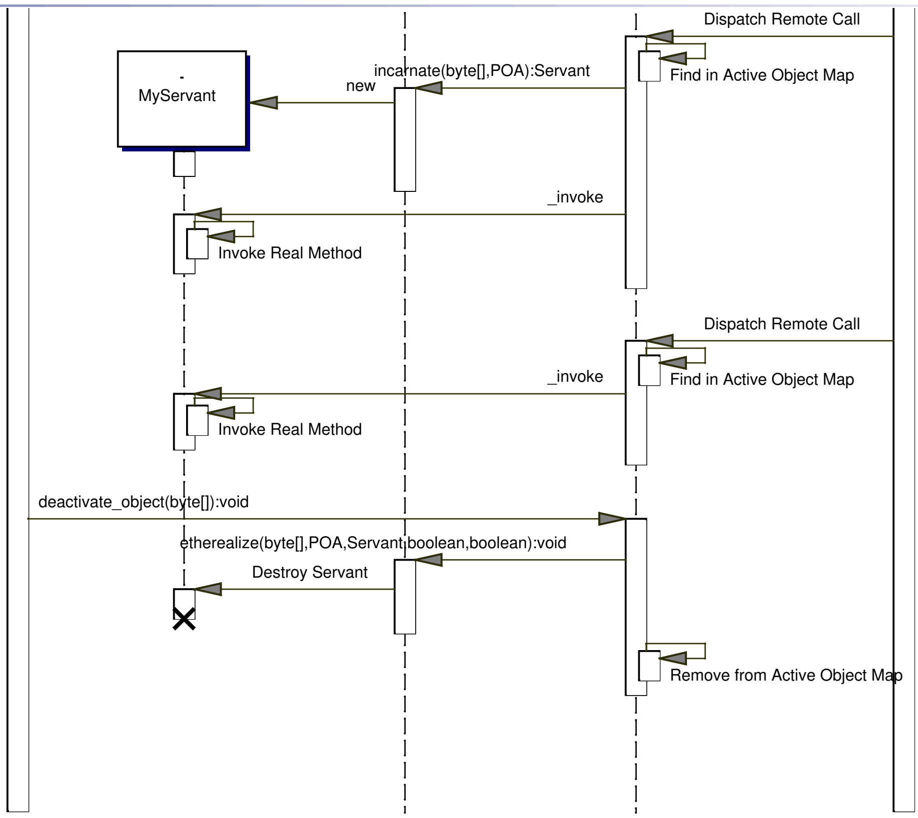


14 POA with Servant Activator

- Policies:
  - ◆ **USE_SERVANT_MANAGER**
  - ◆ **RETAIN**
- Application activates objects on demand via Servant Manager
- Application can deactivate objects using its own strategy

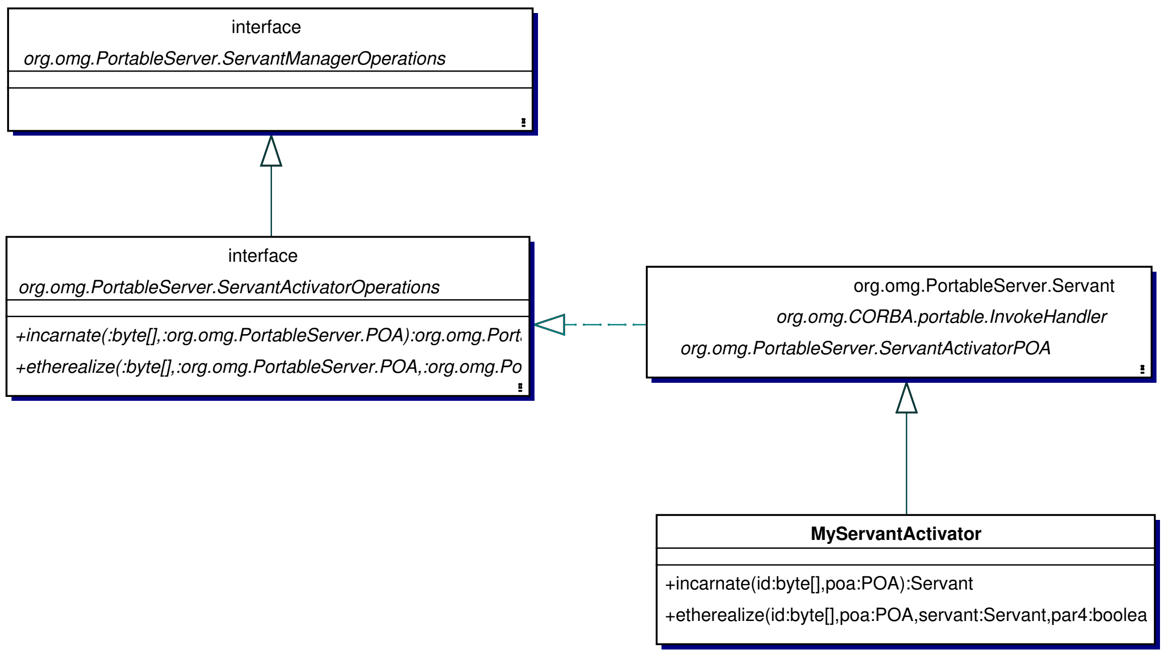


14 POA with Servant Activator



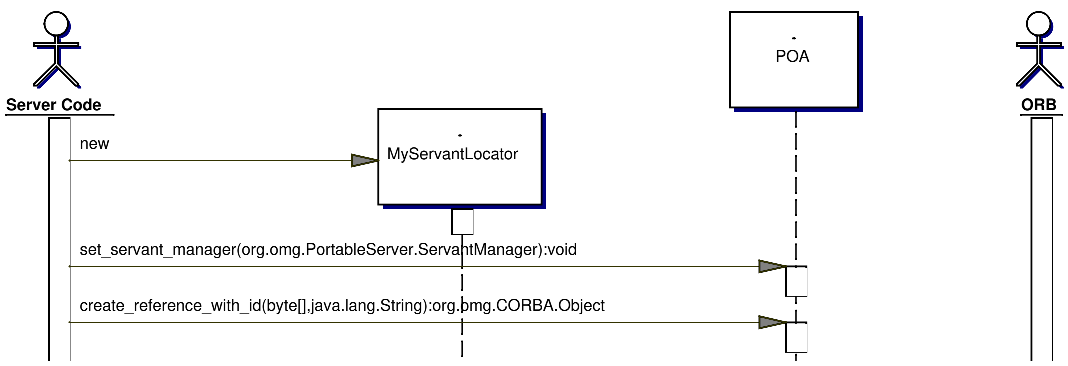
14 POA with Servant Activator

- Servant Manager is a *Servant Activator* (derived interface)

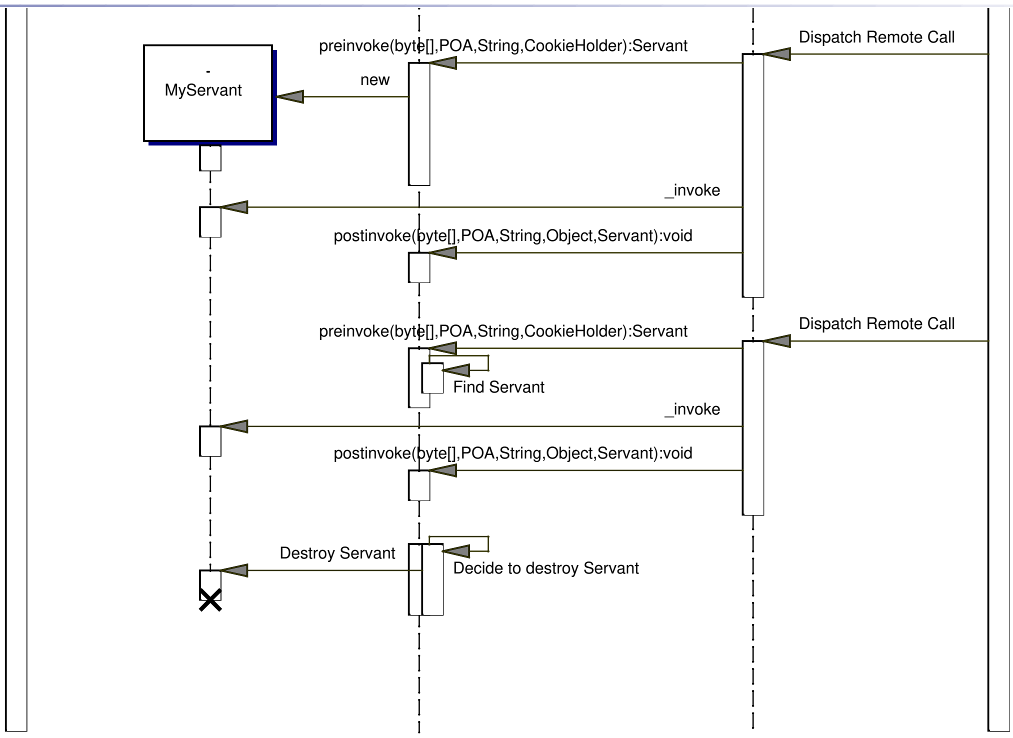


15 POA with Servant Locator

- Policies:
  - ◆ **USE_SERVANT_MANAGER**
  - ◆ **NON_RETAIN**
- POA has no Active Object Map
- Servant Locator is consulted before and after each request
- Servant Locator implements own Active Object Map and eviction strategy

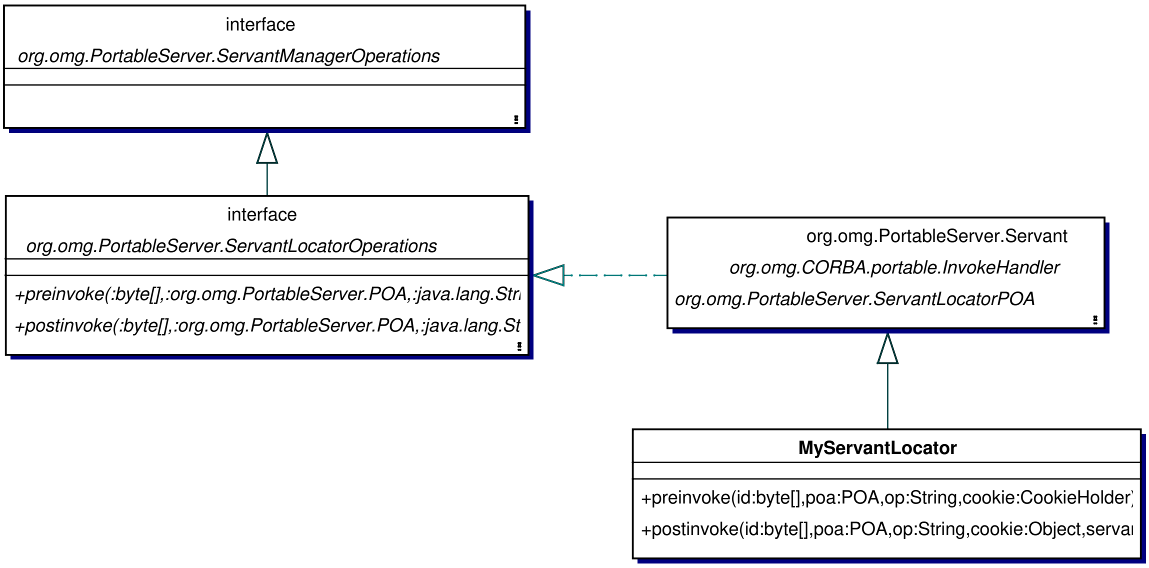


15 POA with Servant Locator



15 POA with Servant Locator

- Servant Manager is a *Servant Locator* (derived interface)



16 POA with Default Servant

- Policies:
  - ◆ **USE_DEFAULT_SERVANT**
  - ◆ **NON_RETAIN**
- POA has no Active Object Map
- Application supplies Default Servant
- Default Servant gets each request of that POA and processes it using the Dynamic Skeleton Interface (DSI)

17 POA Interface

- POA attributes

```
readonly attribute string the_name;
readonly attribute POA the_parent;
readonly attribute POAList the_children;
readonly attribute POAManager the_POAManager;
attribute AdapterActivator the_activator;
```

- POA creation

```
POA create_POA(in string adapter_name,
               in POAManager a_POAManager,
               in CORBA::PolicyList policies)
    raises (AdapterAlreadyExists, InvalidPolicy);
POA find_POA( in string adapter_name,
               in boolean activate_it)
    raises (AdapterNonExistent);
void destroy( in boolean etherealize_objects,
               in boolean wait_for_completion);
```

17 POA Interface

- Policy factory operations

```
ThreadPolicy create_thread_policy(
    in ThreadPolicyValue value);
LifespanPolicy create_lifespan_policy(
    in LifespanPolicyValue value);
IdUniquenessPolicy create_id_uniqueness_policy(
    in IdUniquenessPolicyValue value);
IdAssignmentPolicy create_id_assignment_policy(
    in IdAssignmentPolicyValue value);
ImplicitActivationPolicy create_implicit_activation_policy(
    in ImplicitActivationPolicyValue value);
ServantRetentionPolicy create_servant_retention_policy(
    in ServantRetentionPolicyValue value);
RequestProcessingPolicy create_request_processing_policy(
    in RequestProcessingPolicyValue value);
```


17 POA Interface

- Servant Manager operations

```
ServantManager get_servant_manager()
    raises (WrongPolicy);
void set_servant_manager(in ServantManager imgr)
    raises (WrongPolicy);
```

- Default Servant operations

```
Servant get_servant()
    raises (NoServant, WrongPolicy);
void set_servant(in Servant p_servant)
    raises (WrongPolicy);
```

- Object activation and deactivation operations

```
ObjectId activate_object(in Servant p_servant)
    raises (ServantAlreadyActive, WrongPolicy);
void activate_object_with_id(in ObjectId id,
                             in Servant p_servant)
    raises (ServantAlreadyActive, ObjectAlreadyActive,
            WrongPolicy);
void deactivate_object(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);
```

17 POA Interface

- Identity mapping operations

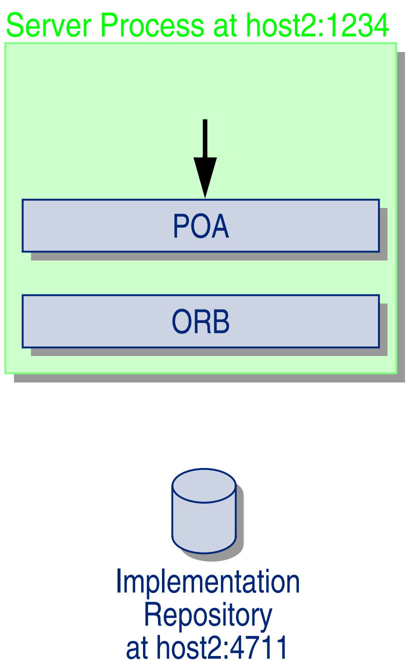
```
ObjectId servant_to_id(in Servant p_servant)
    raises (ServantNotActive, WrongPolicy);
Object servant_to_reference(in Servant p_servant)
    raises (ServantNotActive, WrongPolicy);

Servant reference_to_servant(in Object reference)
    raises(ObjectNotActive, WrongPolicy);
ObjectId reference_to_id(in Object reference)
    raises (WrongAdapter, WrongPolicy);

Servant id_to_servant(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);
Object id_to_reference(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);
```

- Reference creation operations

```
Object create_reference(in CORBA::RepositoryId intf)
    raises (WrongPolicy);
Object create_reference_with_id(in ObjectId oid,
                                in CORBA::RepositoryId intf)
    raises (WrongPolicy);
```

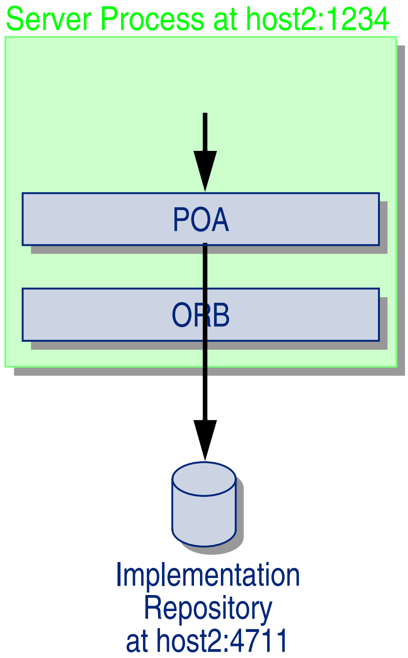

18 Persistent References

- Server Process starts and asks POA to create a reference



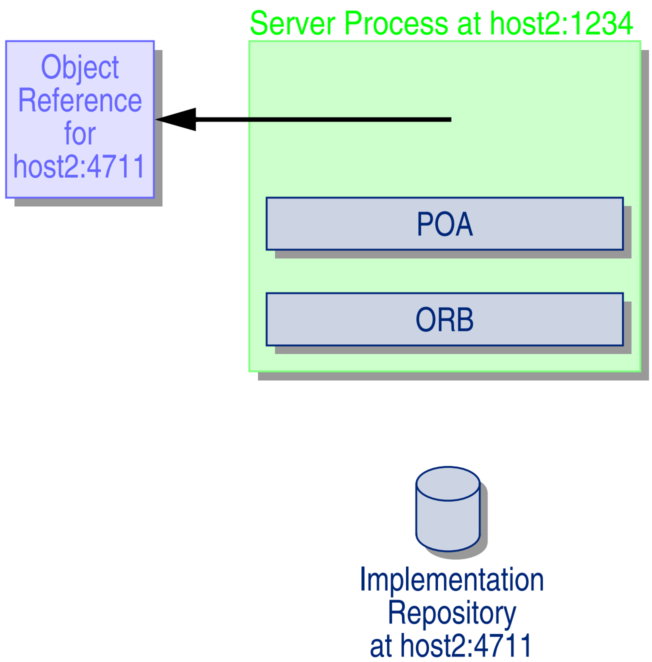
18 Persistent References

- Persistent POA registers server with Implementation Repository



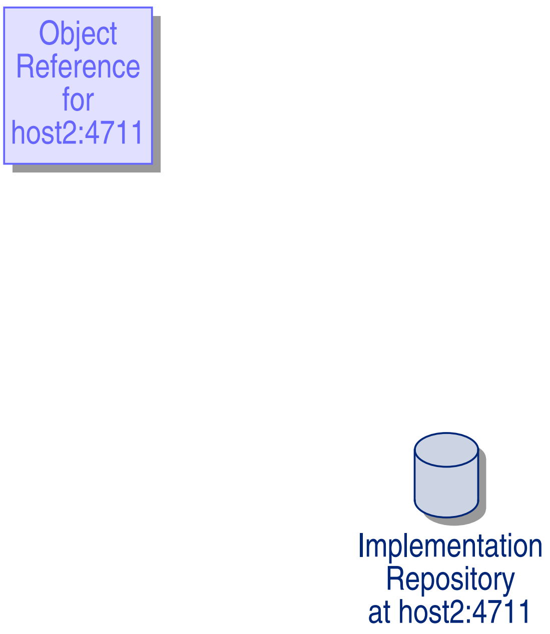
18 Persistent References

- Server saves Persistent Object Reference returned by POA



18 Persistent References

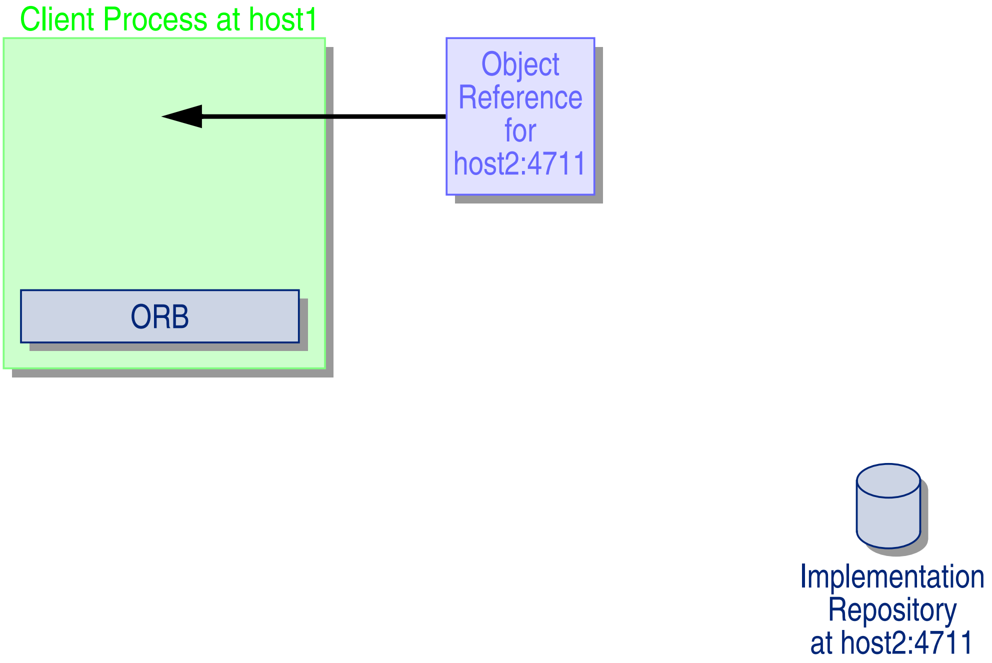
- Server terminates



E.9 Portable Object Adaptor

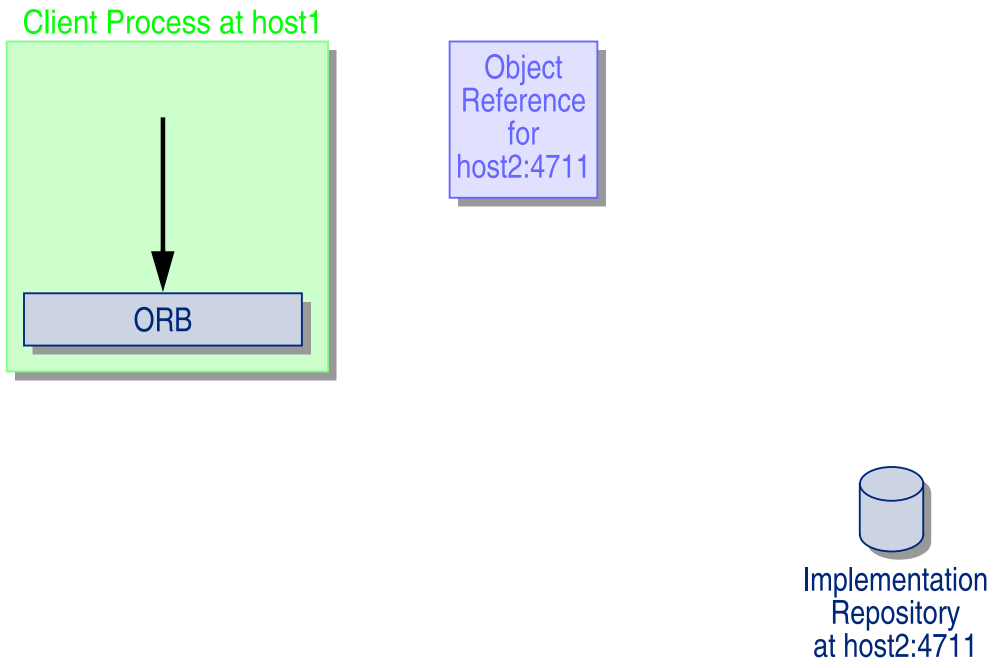
18 Persistent References

- Client starts and reads Object Reference



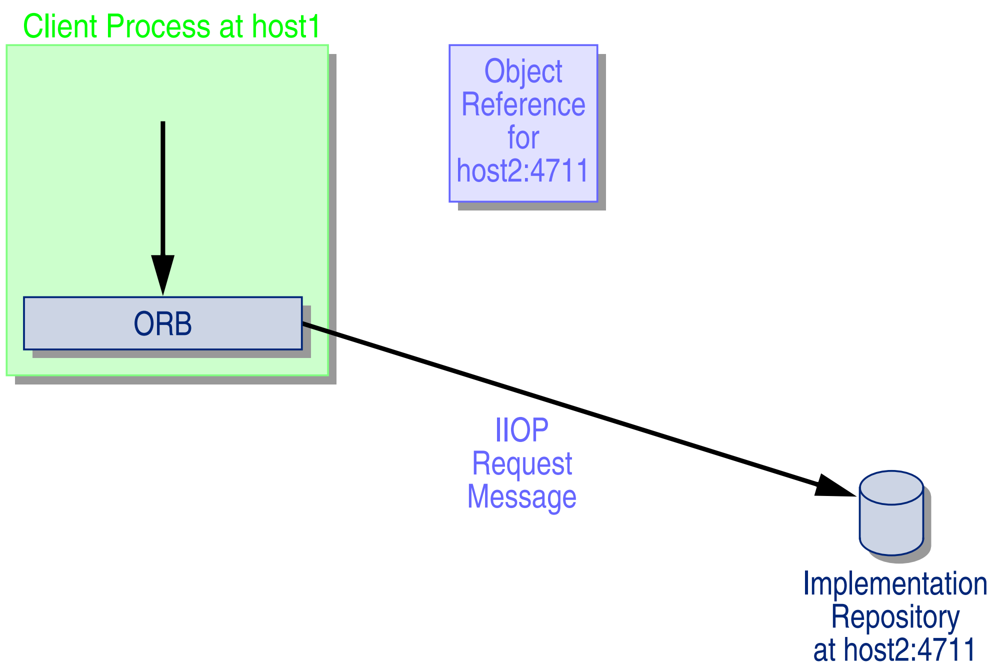
18 Persistent References

- Client invokes operation



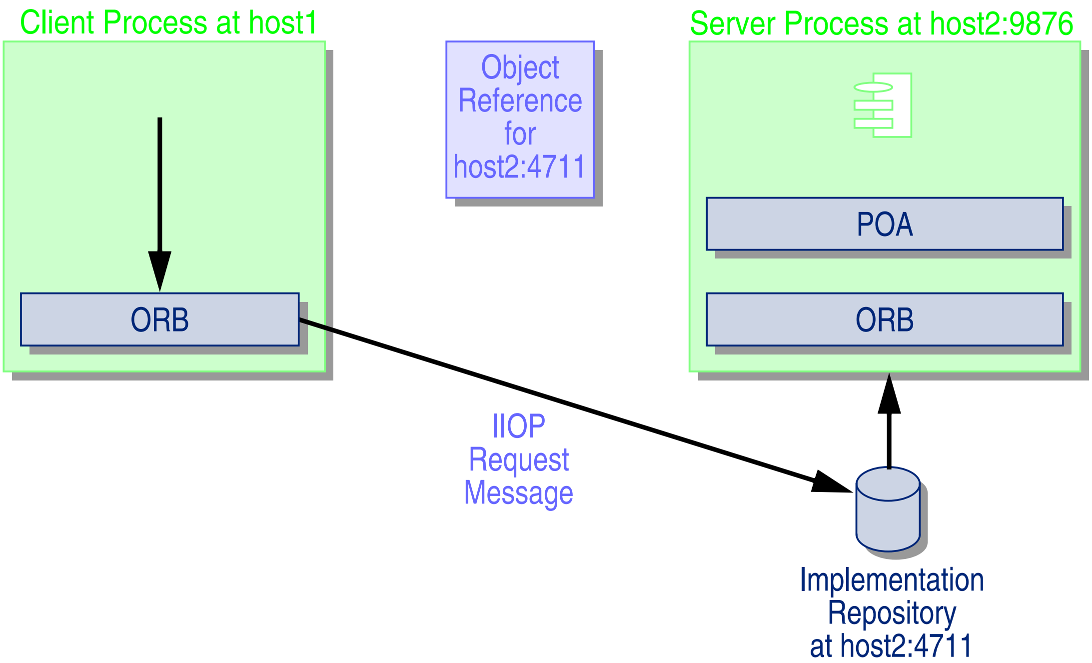
18 Persistent References

- Invocation request is sent to contact address, i.e. IR



18 Persistent References

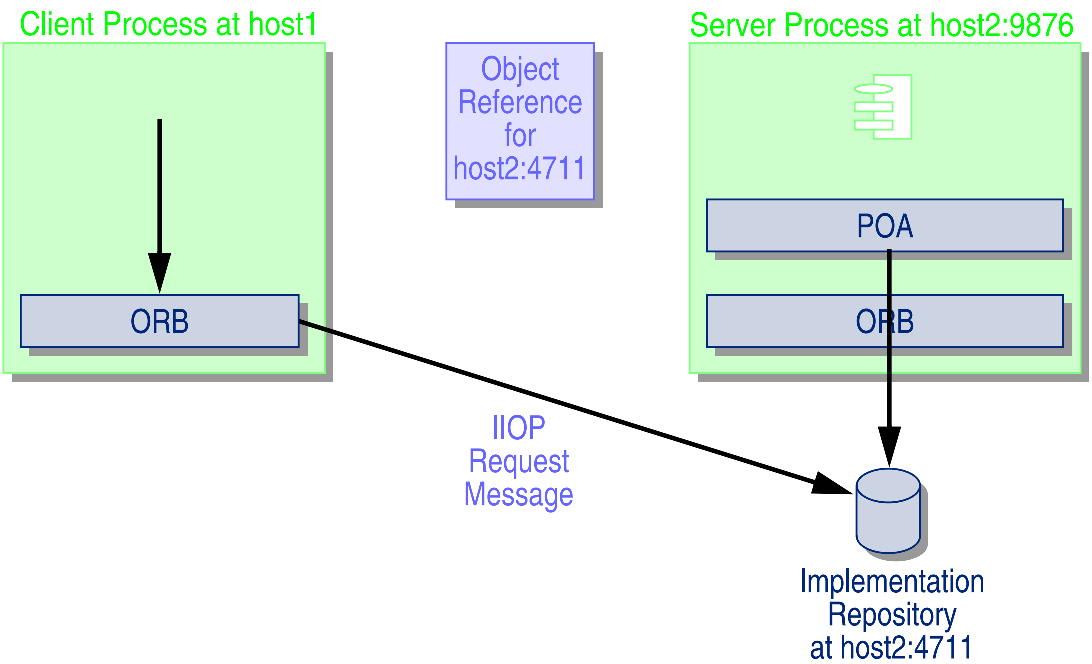
- IR starts Server Process



E.9 Portable Object Adaptor

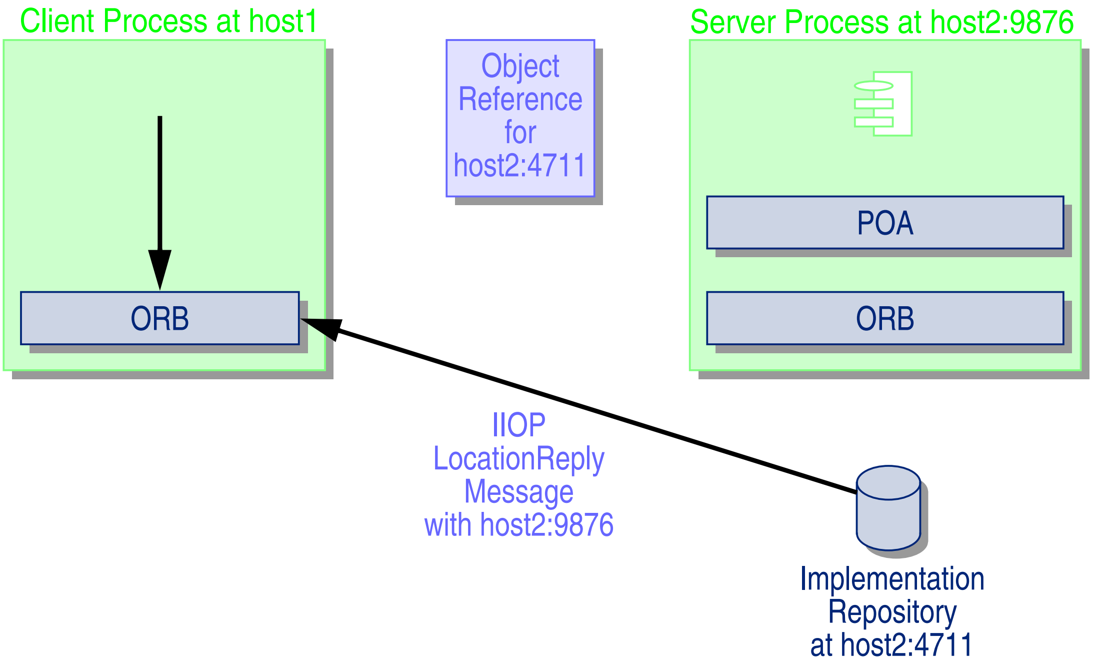
18 Persistent References

- POA registers new contact address with IR



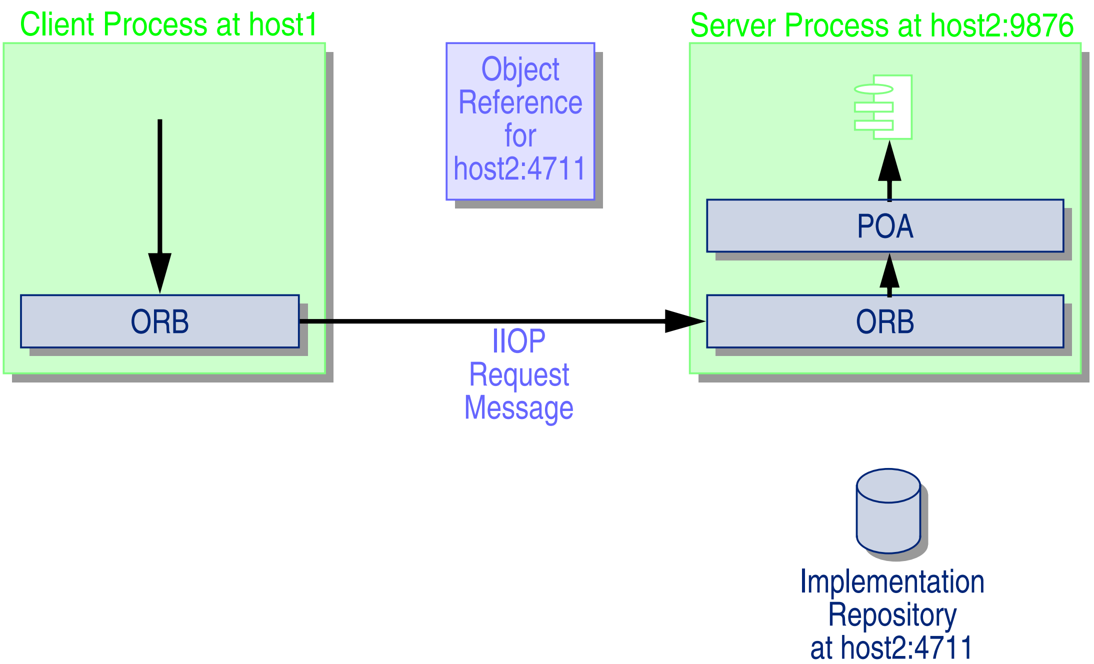
18 Persistent References

- IR returns location forward message with new contact address



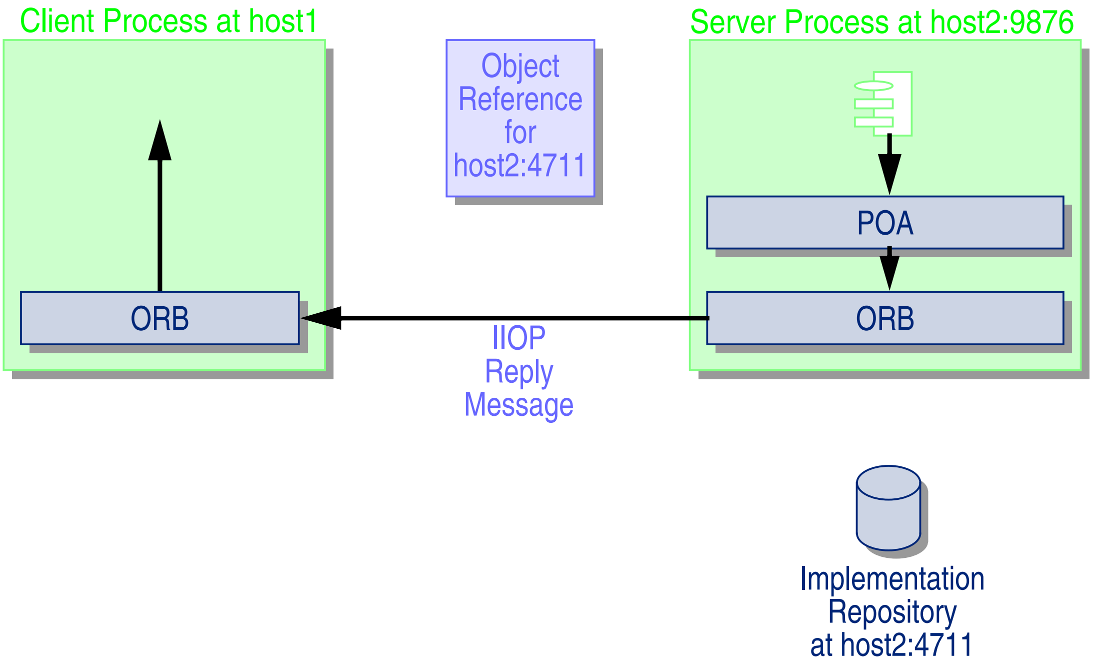
18 Persistent References

- Invocation request is re-sent to new contact address and executed



18 Persistent References

- Reply is returned to the client



19 POA Summary

- Hierarchy of POAs
- Many different policies
- All sorts of request processing and servant management strategies possible
- Persistent references via Implementation Repository