

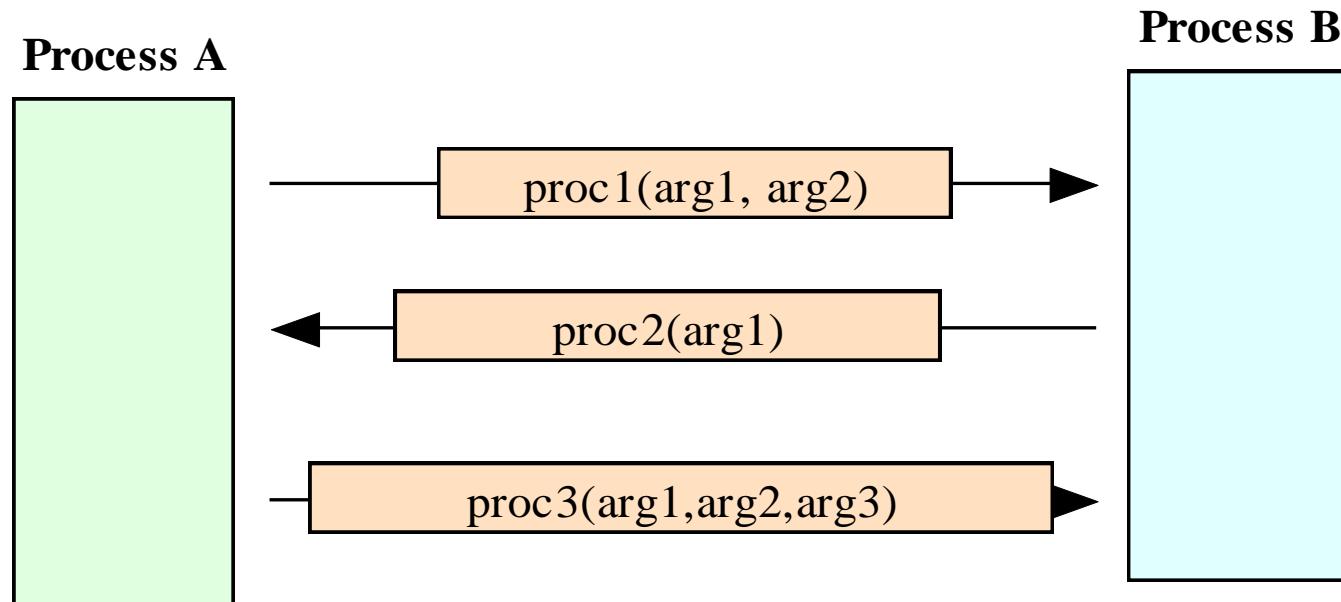
DISTRIBUTED COMPUTING SYSTEMS

RPC and RMI

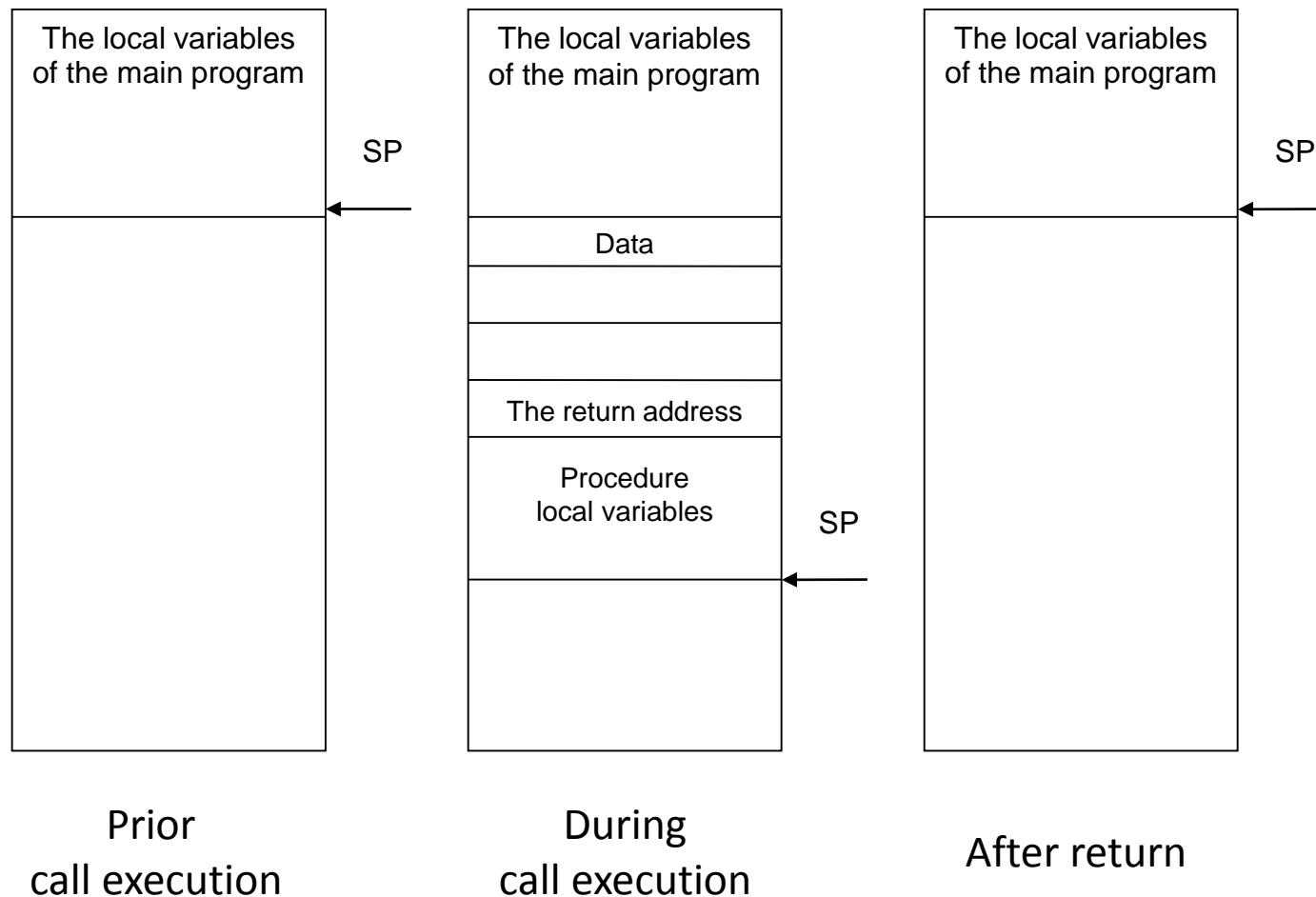
REMOTE PROCEDURE CALL
RPC-REMOTE PROCEDURE CALL
RMI-REMOTE METHOD INVOCATION

RPC TECHNOLOGY

- ◎ **Remote procedure call** is a technology that allows computer programs to call the function or procedure in a different address space.



THE STACK WHEN CALLING LOCAL PROCEDURES



Prior
call execution

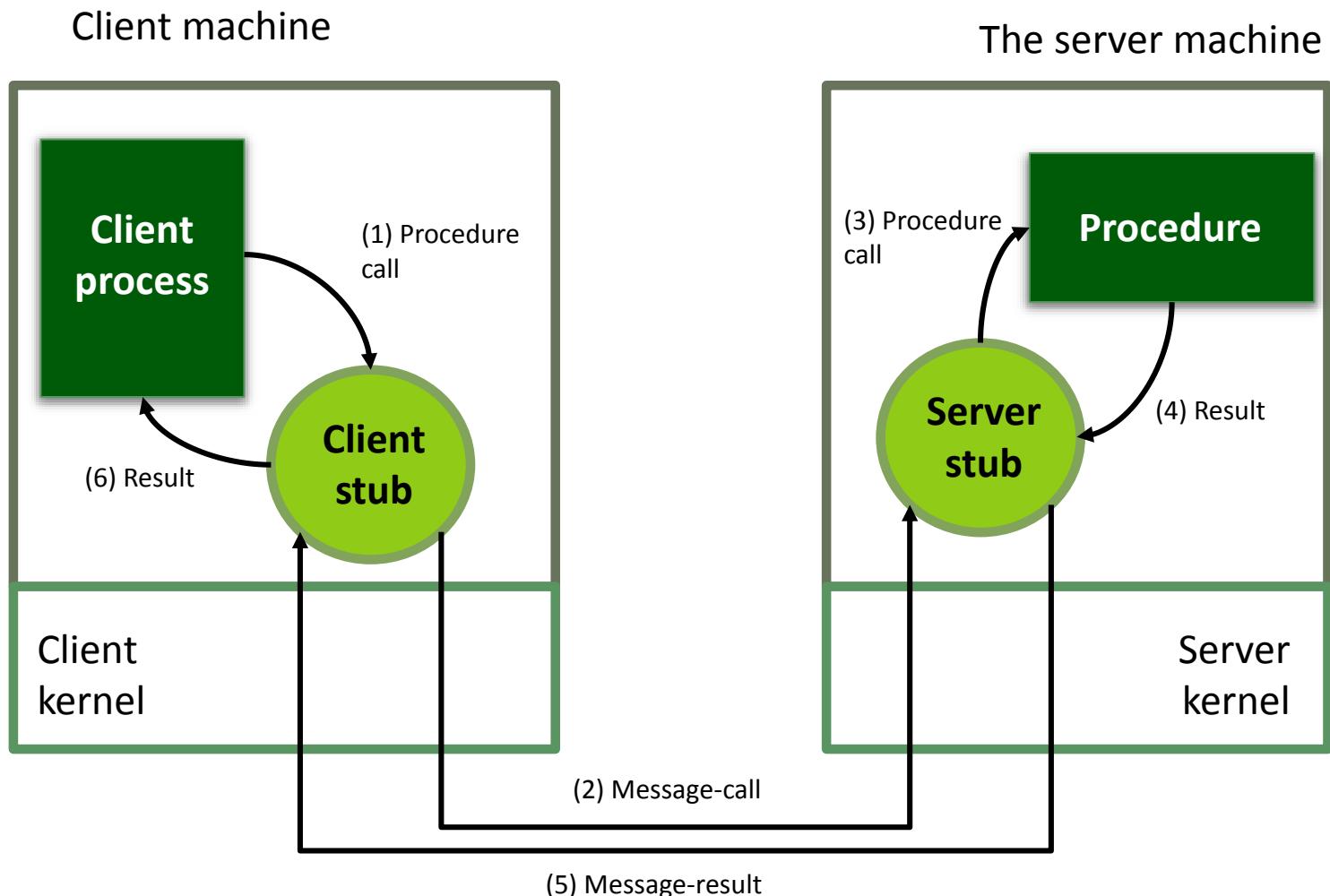
During
call execution

After return

RPC IMPLEMENTATION

- ◎ The idea: remote procedure call "transparent" for the local process
- ◎ Instead of the local procedure we use the “client stub”.
- ◎ It is called as a local procedure, but instead of execution it sends a message the remote machine.

REMOTE PROCEDURE CALL



RPC PSEUDO CODE

Client

```
main { ...  
    myType a = remoteProcedure (arg1, arg2);  
    ... }
```

(6) Result

(1) Procedure call

(2) Message-call

```
myType remoteProcedure (int arg1, int arg2) {  
    byte [] mess, response;  
    string name = "remoteProcedure";  
    string addr = "remote.host:1122";  
    mess =  
        encRemoteProcedure (arg1, arg2);  
    response =  
        callRemoteProcedure (addr, name, mess);  
    return  
        decRemoteProcedureResponse (response);  
}
```

(5) Message-response

Server

```
byte[] serverStab (string name, byte[] mess)  
{
```

switch name:

case "*remoteProcedure*":
 int a, b;

decRemoteProcedure (mess, &a, &b);

myType res = *remoteProcedure* (a, b);
byte [] response =

encRemoteProcedureResponse (res);

return response;

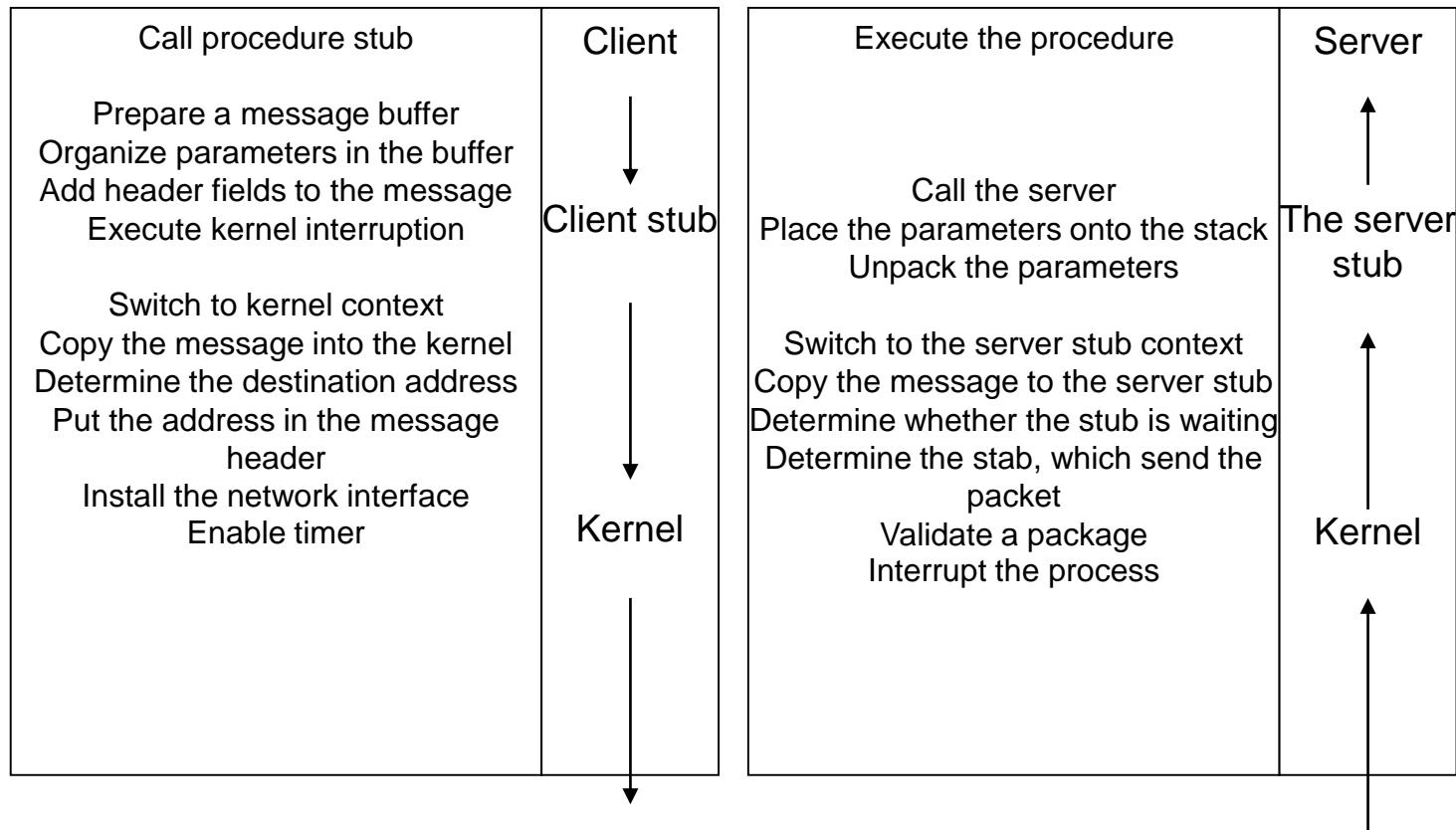
case ...

(4) Response

(3) Procedure call

```
myType remoteProcedure (arg1, arg2) { ...  
    return process(arg1, arg2);  
    ... }
```

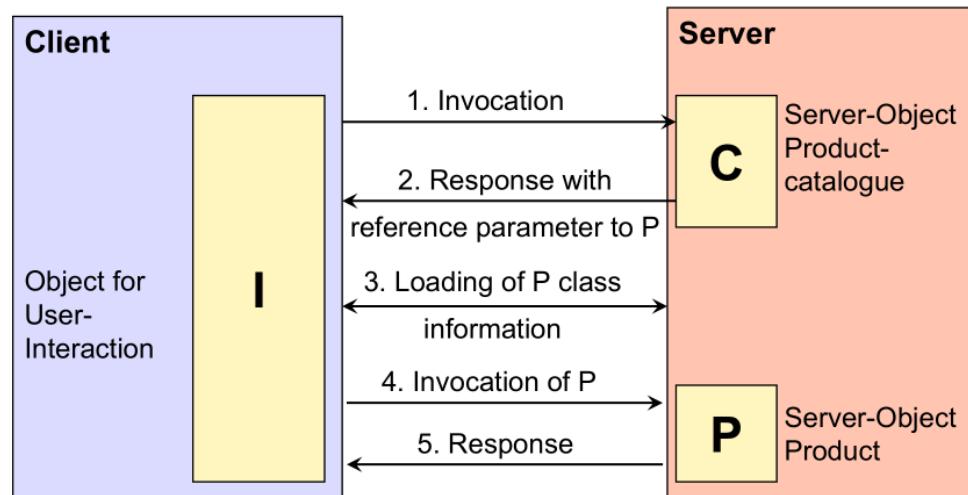
STAGES OF THE RPC



REMOTE METHOD INVOCATION

In terms of OOP the Remote Method Invocation (RMI) concept was implemented.

- ◎ RMI allows to provide transparent access to the methods of remote objects, providing
 - ◎ delivery of parameters of the invoked method,
 - ◎ message to the remote object to execute the method
 - ◎ and the transfer of a return values back to the client



REMOTE OBJECT

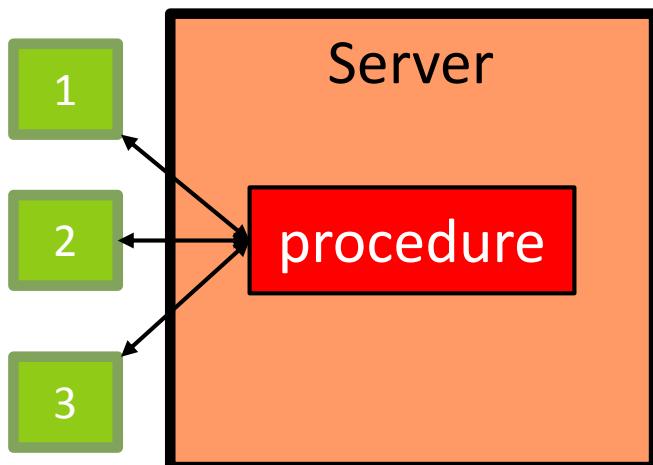
- ◎ The remote object is a collection of some data that determine its State. This State can be changed by calling some of his methods.
- ◎ Methods and fields of an object that can be used via remote calls, are available through the **external interface** of the objects class.

REMOTE PROCEDURE VS REMOTE OBJECT

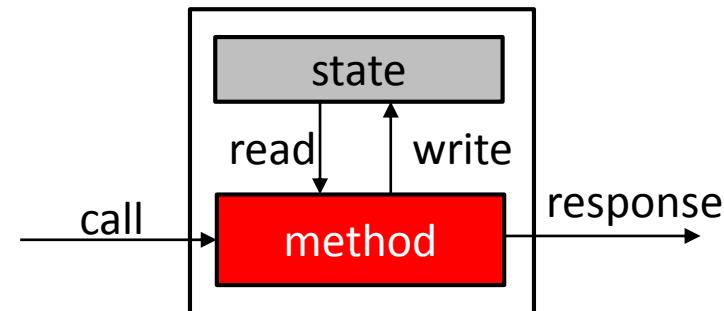
Remote Procedure



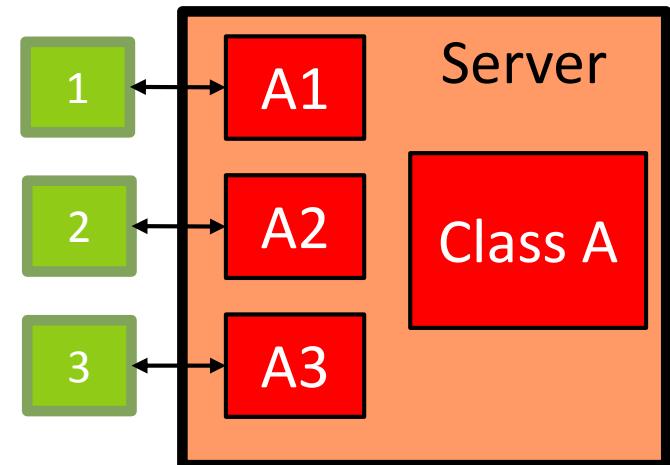
Clients



Remote Object



Clients

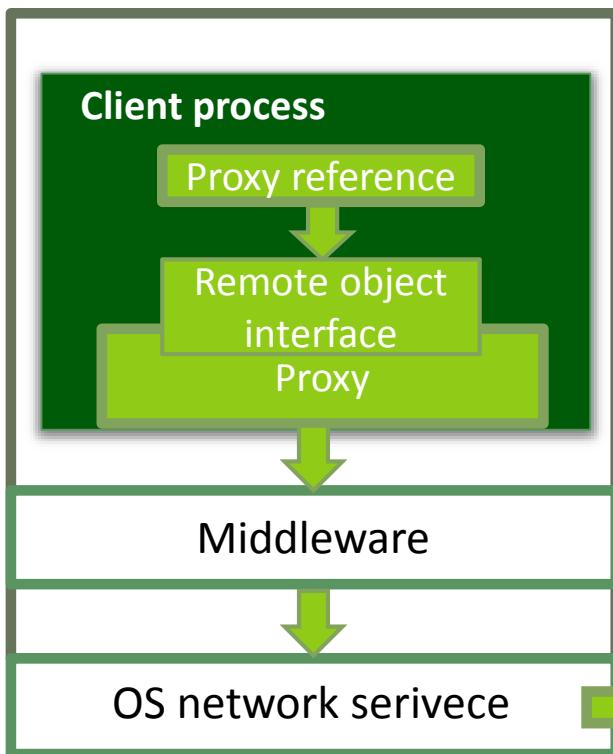


A PROXY AND A SCELETON

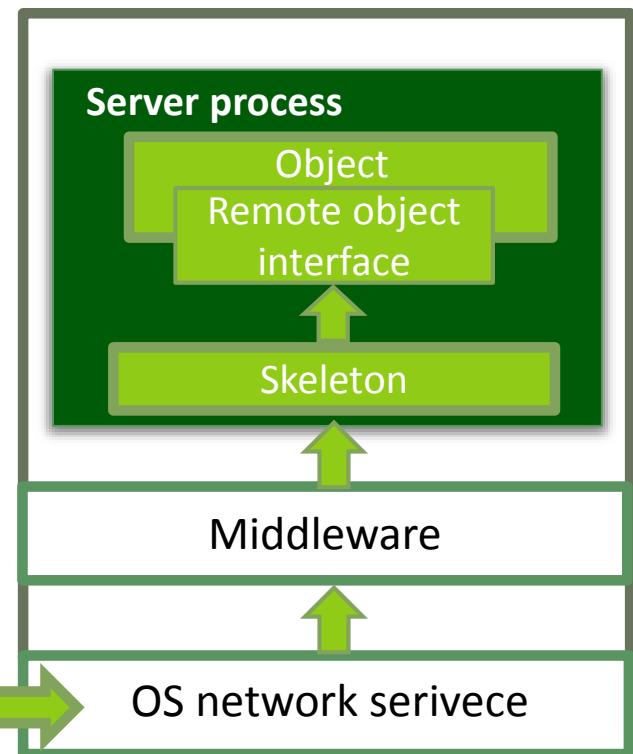
- ◎ A client stub that invokes a remote object is called **the proxy**.
- ◎ **Proxy** implements the same interface as the remote object.
- ◎ The server-side stub is called **the skeleton** (in Java RMI)
- ◎ The skeleton is associated with a specific instance of the remote object and invokes the method with the desired settings

REMOTE OBJECT USAGE

The client machine



The server machine



Data transfer channel

REMOTE METHOD INVOCATION RMI

Interface:

```
public interface ProductCatalogue extends java.rmi.Remote
{
    ProductDescription[] searchProduct(String productType) throws java.rmi.RemoteException;
    Product provideProduct(ProductDescription d) throws java.rmi.RemoteException;
    int deleteProduct(ProductDescription d) throws java.rmi.RemoteException;
    int updateProduct(Product p) throws java.rmi.RemoteException;
    ...
}
```

Server – interface realization:

```
public class ProductCatalogueImpl extends java.rmi.server.UnicastRemoteObject
implements ProductCatalogue
{
    public ProductCatalogueImpl() throws java.rmi.RemoteException
    {   super();   }

    public ProductDescription[] searchProduct(String productType)
        throws java.rmi.RemoteException
    {
        ProductDescription[] desc = ProductCatalogue.getDescriptionByType(productType);
        return desc;
    }
    ...
}
```

REMOTE METHOD INVOCATION RMI

Server realization:

```
public class ProductCatalogueServer {
    public ProductCatalogueServer() {
        try {
            ProductCatalogue c = new ProductCatalogueImpl();
            Naming.rebind("rmi://localhost:1099/ProductCatalogueService", c);
        }
        catch (Exception e) {...}
    }
    public static void main(String args[]) {
        new ProductCatalogueServer();
    }
}
```

Client Realization:

```
public class ProductCatalogueClient {
    public static void main(String[] args)
    {
        try {
            ProductCatalogue c= (ProductCatalogue)Naming.lookup(
                "rmi://hostname/ProductCatalogueService");
            System.out.println( c.searchProduct("book"));
        }
        catch (Exception e) {...}
    }
}
```