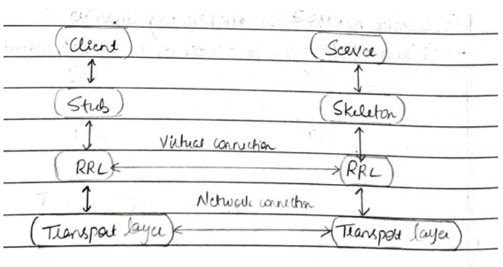
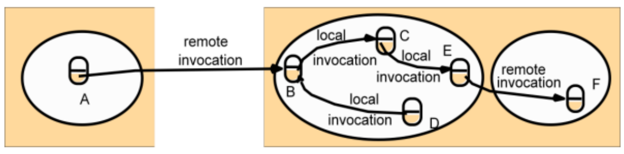
# Distribution Object and RMI

## 1. Explain communication between distributed objects by means of RMI.



Remote and local method invocations



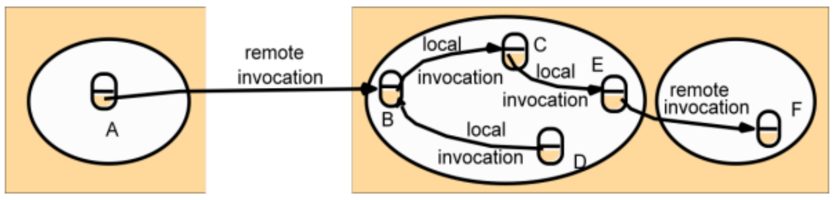
- The term distributed objects usually refer to software modules that are designed to work together but reside either in multiple computers connected via a network or in different processes inside the same computer.
- The state of an object consists of the values of its instance variables.
- Distributed object systems may adopt the client-server architecture. In this case, objects are managed by servers and their clients invoke their methods using Remote Method Invocation (RMI).
- In RMI, the client's request to invoke a method of an object is sent in a message to the server managing the object. The invocation is carried out by executing a method of the object at the server and the result is returned to the client in another message.
- To allow for chains of related invocation, objects in servers are allowed to become clients in other servers.
- **Transport Layer:** Connects client to the server.
- **Stub:** Representation (proxy) of the remote object at the client.
- **Skeleton:** Representation (proxy)  of the remote object at the client.
- **RRL (Remote Reference Layer):** Manages references made by the client to the remote object.

### Working of RMI Application

- When a client makes a request to the remote object, it is received by stub and eventually, the request is passed to RRL.
- When client-side RRL receives the request and it calls the method `invoke()` of the object remoteRef. It passes the request to RRL on the server-side.
- The RRL on the server-side passes the request to the skeleton which then finally executes the required object on the server.
- The result is passed all the way back to the client.

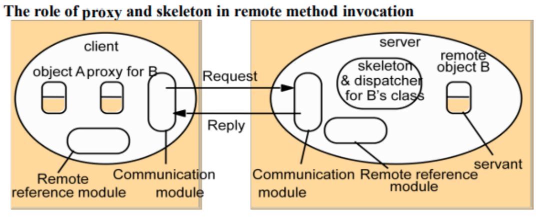
## 2. Explain remote and local invocation with neat diagrams.

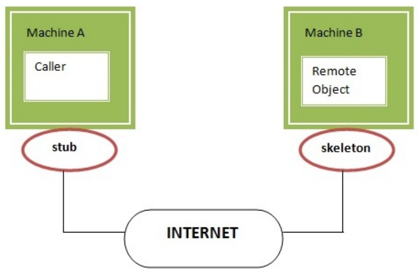
**Remote and local method invocations**



## 3. With a neat diagram explain the role of Proxy & Skeleton in RMI.

**The role of proxy and skeleton in remote method invocation**





RMI uses proxy and skeleton objects for communication with the remote object. A remote object is an object whose method can be invoked from another JVM.

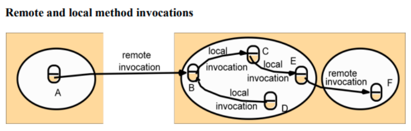
### Proxy

- The role of the proxy is to make remote method invocation transparent to clients by behaving like a local object to the invoker; but instead of executing an invocation, it forwards it in a message to the remote object.
- The proxy acts as a gateway for the client-side. All the outgoing requests are routed through it.
- When the caller invokes method on the stub object, it does the following tasks:
    - It initiates a connection with the remote Virtual Machine (JVM)
    - It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM)
    - It waits for the result
    - It reads (unmarshals) the return value or exception
    - And finally, it returns the value to the caller.
- There is one proxy for each remote object for which a process holds a remote object reference.

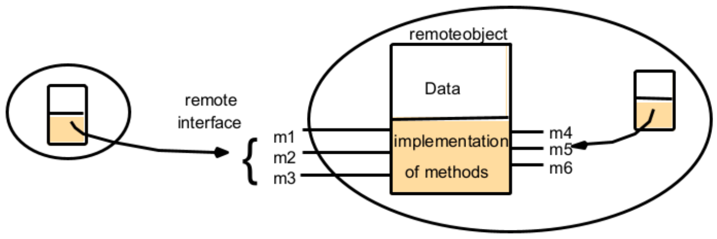
## Skeleton

- The skeleton acts as a gateway for the server-side object.
- All the incoming requests are routed through it.
- When the skeleton receives the incoming request, it does the following tasks:
    - It reads (unmarshals) the parameter for the remote method
    - It invokes the method on the actual remote object
    - It writes and transmits (marshals) the result to the caller.

## 4. Explain the fundamental concepts of the distributed object model.



Remote Method Invocation



Remote Interface`

- Each process contains a collection of objects, some of which can receive both local and remote invocations, whereas the other objects can receive only local invocations.
- Method invocations between objects in different processes, whether in the same computer or not, are known as remote method invocations.
- Method invocations between objects in the same process of the computer are local method invocations and objects that can receive remote invocations are called remote objects.
- There are two fundamental concepts in distributed object model:
    - Remote Object Reference
    - Remote Interfaces
- **Remote Object References:**
  A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object. Other objects can invoke the methods of a remote object if they have access to its remote object reference. For example, a remote object reference for B in the figure must be available to A.
    - The remote object receiving a remote method invocation is specified by the invoker as a remote object reference.
    - Remote object references may be passed as an argument.
- **Remote Interfaces:**
    - The class of a remote object implements the methods of its remote interface.
    - It specifies which methods can be invoked remotely.
    - Objects in other processes can invoke only the methods that belong to its remote interface.
    - Remote interfaces, like all interfaces, do not have constructors.
    - The CORBA system provides an **Interface Definition Language (IDL),** which is used for defining remote interfaces.

## 5. Discuss RMI invocation semantics and tabulate failure handling mechanism for

## each.

## The RMI invocation semantics are as follows:

### Maybe Semantics

- With *maybe* semantics, the remote procedure call may be executed once or not at all.
- Maybe semantics arises when no fault-tolerance measures are applied and can suffer from the following types of failures:
  - Omission failures if the request or result message is lost.
  - Crash failures when the server containing the remote operation fails.
- If the result message has not been received after a timeout and there are no retries, it is uncertain whether the procedure has been executed.

### At-least-once Semantics

- With *at-least-once* semantics, the invoker receives wither a result, in which case the invoker knows that the procedure was executed at least once, or an exception informing it that no result was received.
- *At-least-once* semantics can be achieved by the retransmission of request messages.
- *At-lease-once* semantics can suffer from the following types of failures:
  - Crash failures when the server containing the remote procedure fails.
  - Arbitrary failures - in cases when the request message is retransmitted, the remote server may receive it and execute the procedure more than once, possibly causing wrong values to be stored and returned.

### At-most-once Semantics

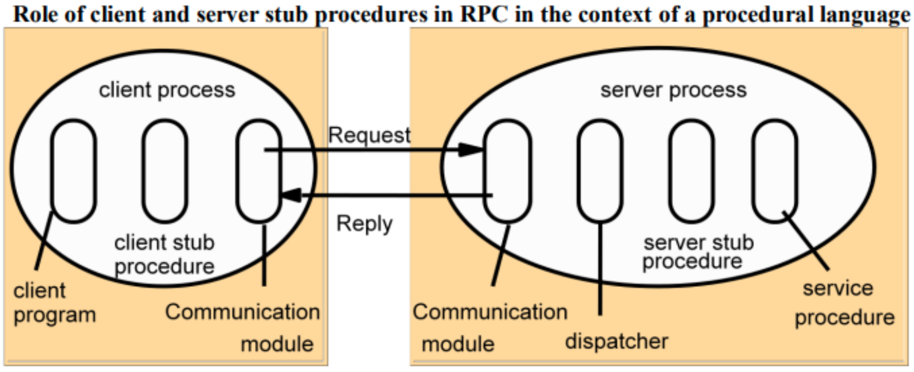
- With *at-most-once* semantics, the caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received, in which case the procedure will have been executed either once or not at all.

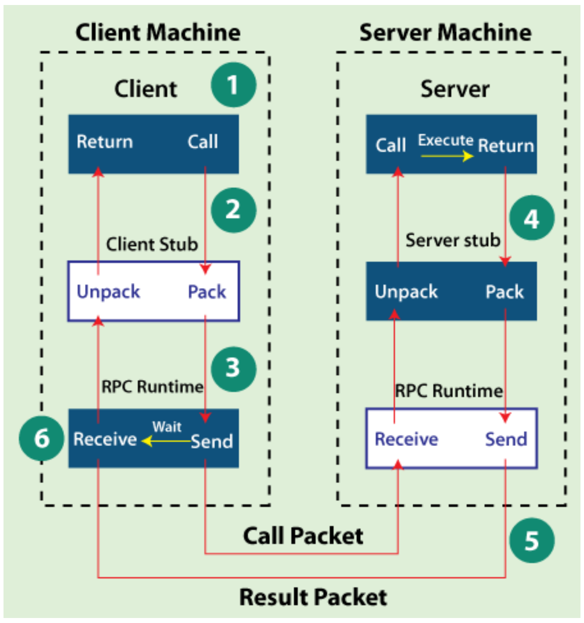
### Failure handling mechanism

| Invocation Semantics | Fault tolerance measures | Fault tolerance measures | Fault tolerance measures |
|---|---|---|---|
| | Retransmit request message | Duplicate filtering | Re-execute procedure or retransmit reply |
| Maybe | No | Not applicable | Not applicable |
| At-least-once | Yes | No | Re-execute procedure |
| At-most-once | Yes | Yes | Retransmit reply |

## 6. Define RPC and with a neat diagram explain its implementation.

Role of client and server stub procedures in RPC in the context of a procedural language





- The *Remote Procedure Call (RPC)* is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

- An RPC is analogous to a function call. Like a function call, when RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure.

## Implementation

- The software components required to implement RPC are shown in the figure.
- The client calls the client stub. The call is a local procedure call, with parameters pushed onto the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called Marshalling.
- The client's local operating system sends the message from the client machine to the server machine.
- The local operating system on the server machine passes the incoming packets to the server stub.
- The server stub unpacks the parameters from the message. Unpacking the parameters is called Unmarshalling.
- Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.