* 1. **RPC** is a *program* (i.e. application) level abstraction where you can use data structures whereas **IPC** is at the *operating system* level where user data structures won’t be visible.
  2. **IPC** is message based off of a byte-stream that is passed as data.
  3. Because **RPC** is implemented using **IPC** at the system level, so you have to convert whatever data structure you have to byte streams, and marshalling/unmarshalling is the mechanism to do exactly that.
  4. Strictly speaking, **IPC** handles only *messages*, so you won’t be doing any marshalling/unmarshalling at that level. However, marshalling/unmarshalling still must be performed (i.e. whether by the application framework or the programmer himself).
     1. **RPC** is a call and return protocol. The call is implemented using a request message and return is implemented using the reply message. These messages must be properly synchronized.
     2. **IPC** requires *one message only*.
     3. In client-server interaction a request is sent and a reply is expected, so this request and reply scheme should be properly synchronized, so the middleware takes care of this requirement, making the programmer’s life easier. The client-server interaction is application level and it requires handling higher level data structures and therefore requires marshalling/unmarshalling and the middleware takes care of this, taking this responsibility from the programmer.
     4. Using the middleware, the application makes a method call the same way, regardless of it being local or remote. So the middleware makes the remote call appear to be local and abstracts this notion of ***access* *transparency***. The middleware also achieves ***concurrency* *transparency*** (#funFact).
     5. Server-server interactions are implemented by server programmers who are typically much more skilled, knowledgeable, environment (like, what?). So the goal is ***performance*** rather than ***transparency***.
     6. Because it is up to the server programmers to implement the marshalling and unmarshalling to gain optimization/performance benefits.
     7. Client-server communication is implemented as RPC calls which require a request and reply that is in order and synchronized. TCP/IP provides a virtual connection to simplify the sending and receiving of the request and reply.
     8. There is an overhead in setting up the virtual connection and therefore there are losses in performance.
     9. Typically there is a larger programming effort to implement the marshalling/unmarshalling logic.
     10. Object adapters translate the language neutral request and reply to the language specific request and reply in CORBA.
     11. Yes, because all CORBA systems the request reply are language neutral in the middleware.
     12. The implementation repository, as you know, contains the host address and port number which the object adapter can use to download the server implementation in different programming languages. Since CORBA supports multiple languages, should have this construct.
     13. Obviously not. If the object adapter already knows all the implementation and supports only one language then it wont need an implementation repository.
     14. The interface repository contains the interface definitions of the server methods so that clients can invoke these methods properly. Particularly, using dynamic invocation, that is how interface repositories are used. Since CORBA is a dynamic system designed to operate on Wide Area Networks where new servers can be added freely, CORBA has more reason to make dynamic invocations than Java RMI clients do. That is why they use interface repositories.
     15. Obviously, no. If the client already knows all the server methods then there may be no need to use an interface repository.
     16. 3 marshalling 4 unmarshalling (the first server marshalls its request once and sends it to each server for the remaining 2 servers, of which will unmarshall the request, perform the operation, marshall the response to send back to the calling server, which unmarshalls 2 separate requests now.).