Sxternal Data Representation!→

Interprocess communication is important part in distributed computing where two process communicate with each other to share data.

The process can be in same running machine or different machines.

when communication happens wh b/w processes data type of ~~transmitted~~ data differs according to the platform the process is running.

↑

In This situation we need to implement ~~pat~~ external Data representation & marshalling.

So, " An agreed standard for the representation of data structures and primitive values is said to be an external Data representation"

information stored in running programs

↓

Represented as Data Structures

information stored in messages

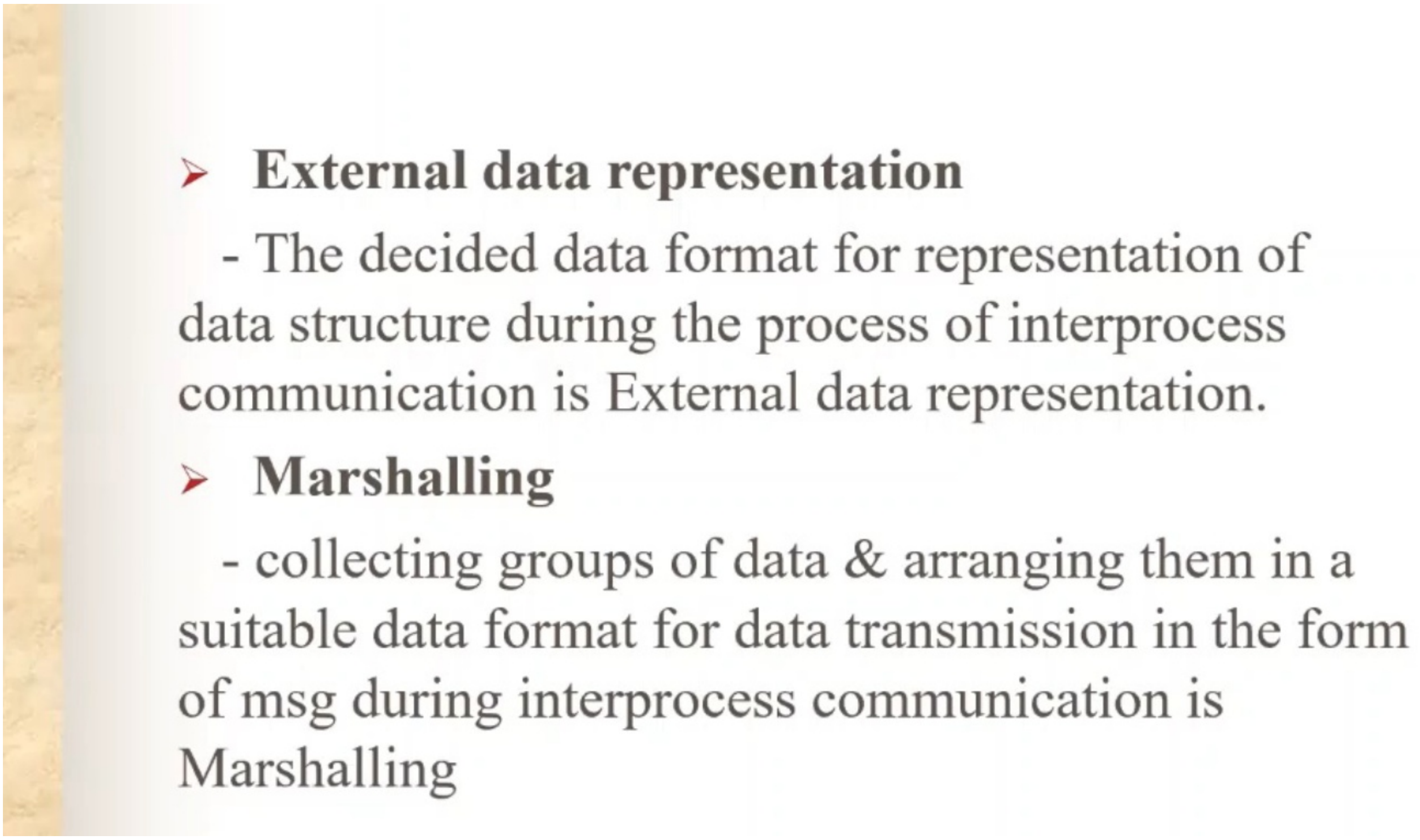
↓

consists of sequences of bytes

$$\begin{tikzcd}
\boxed{\text{Data structures must be converted}} \arrow[r] & \boxed{\text{Sequence of bytes}}
\end{tikzcd}$$

↑

Before transmission.

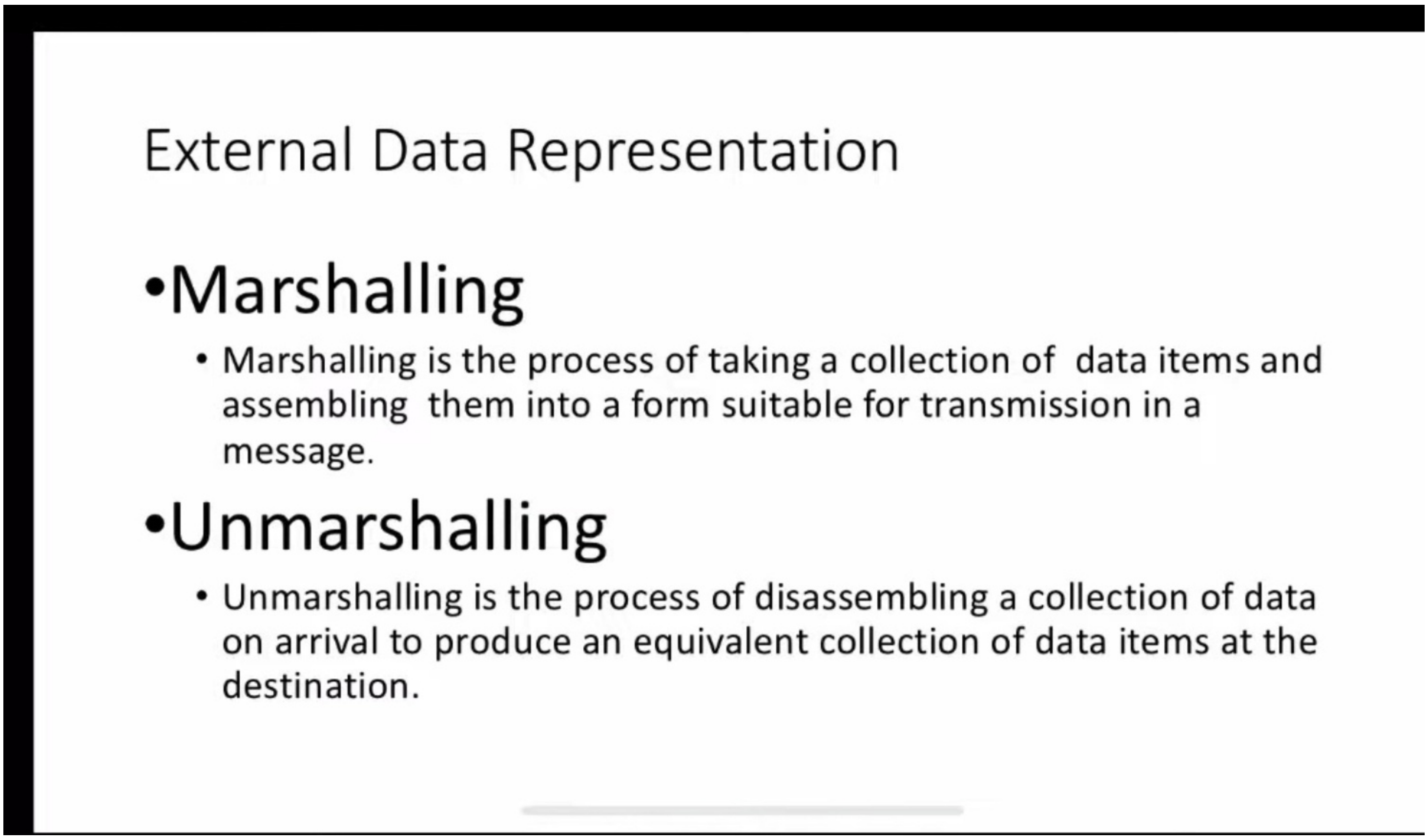


- **External data representation**

  - The decided data format for representation of data structure during the process of interprocess communication is External data representation.

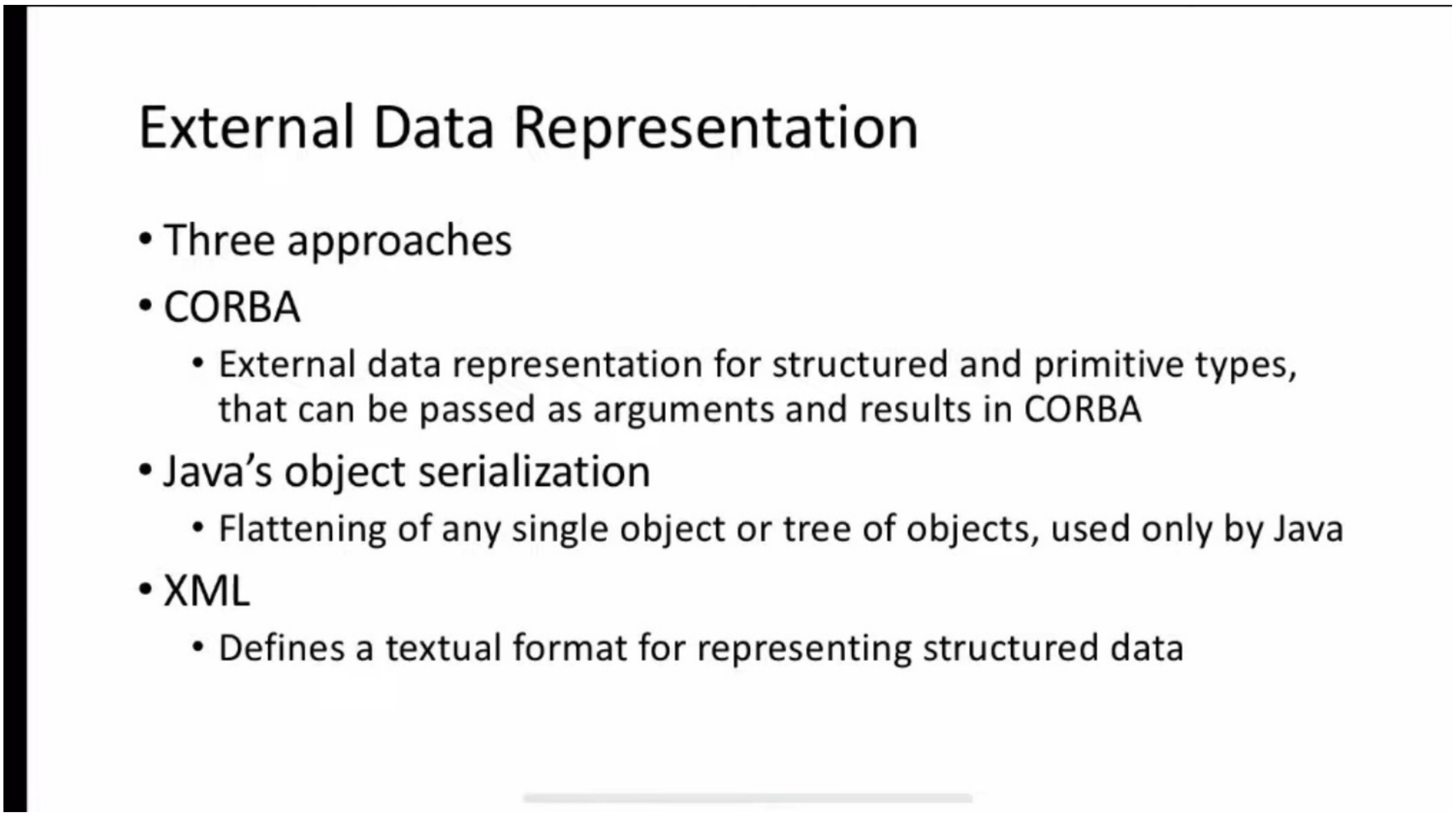
- **Marshalling**

  - collecting groups of data & arranging them in a suitable data format for data transmission in the form of msg during interprocess communication is Marshalling



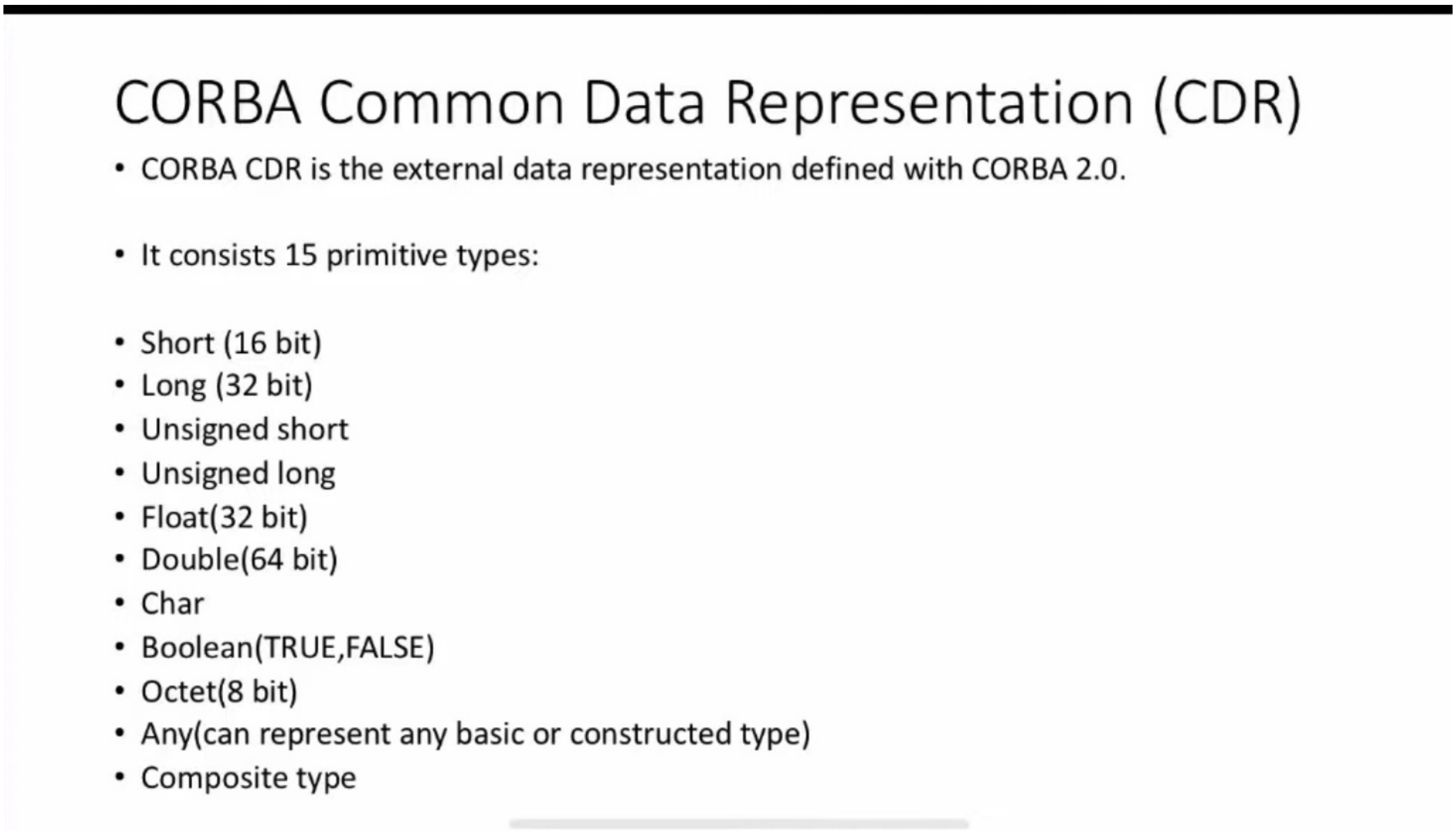
External Data Representation

- **Marshalling**
  - Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message.
- **Unmarshalling**
  - Unmarshalling is the process of disassembling a collection of data on arrival to produce an equivalent collection of data items at the destination.



External Data Representation

- Three approaches
- CORBA
  - External data representation for structured and primitive types, that can be passed as arguments and results in CORBA
- Java's object serialization
  - Flattening of any single object or tree of objects, used only by Java
- XML
  - Defines a textual format for representing structured data



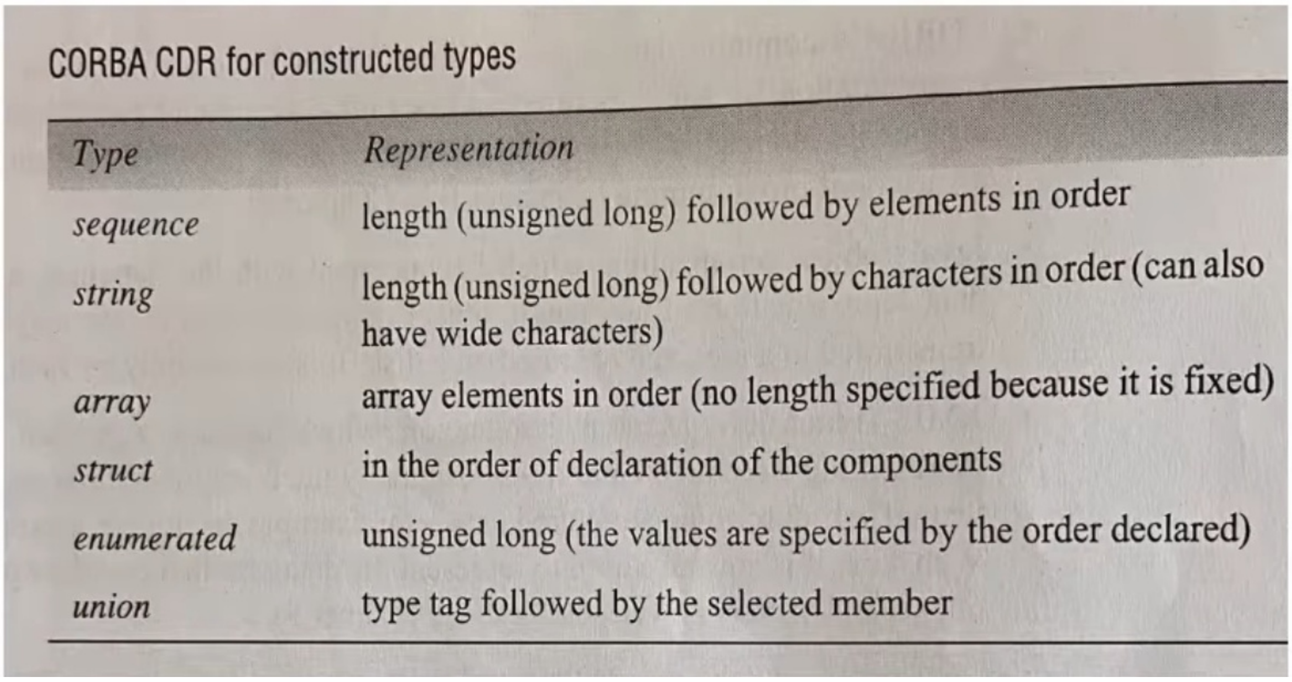
CORBA Common Data Representation (CDR)

- CORBA CDR is the external data representation defined with CORBA 2.0.

- It consists 15 primitive types:

- Short (16 bit)
- Long (32 bit)
- Unsigned short
- Unsigned long
- Float(32 bit)
- Double(64 bit)
- Char
- Boolean(TRUE,FALSE)
- Octet(8 bit)
- Any(can represent any basic or constructed type)
- Composite type

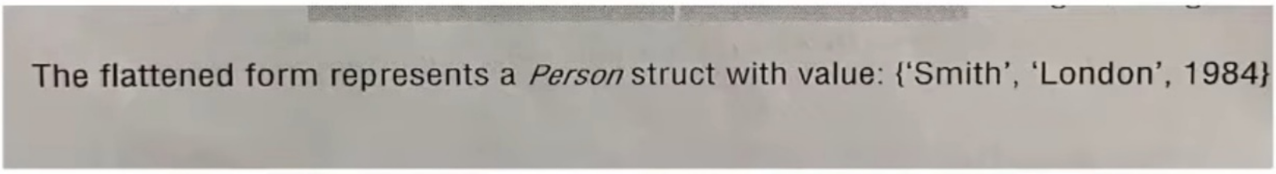
CORBA Common Data Representation (CDR)



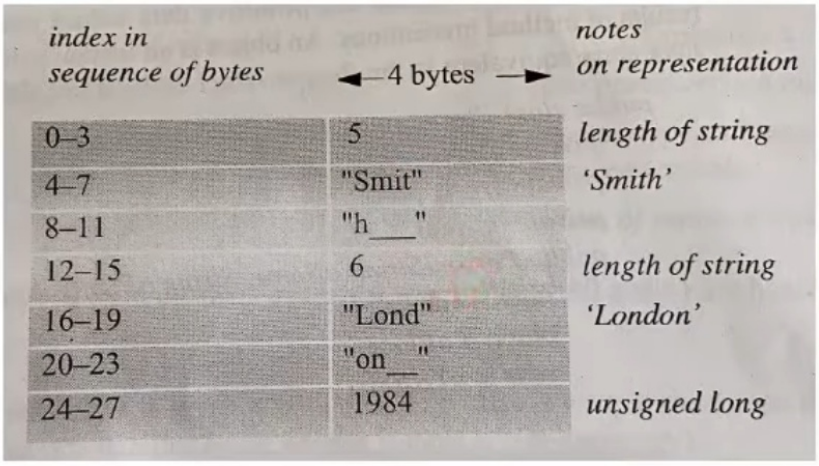
CORBA CDR for constructed types

Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

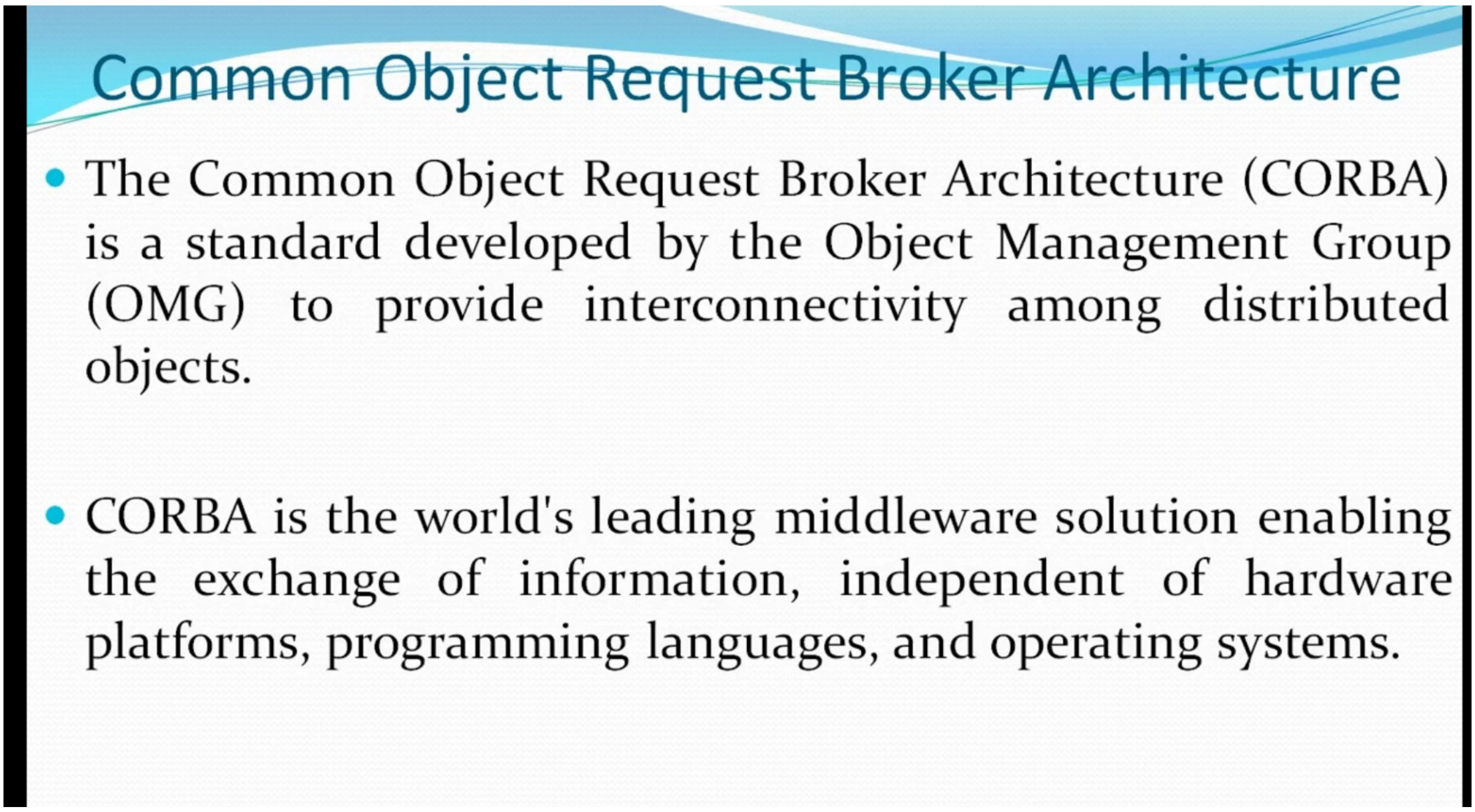
CORBA Common Data Representation (CDR)



The flattened form represents a *Person* struct with value: {'Smith', 'London', 1984}



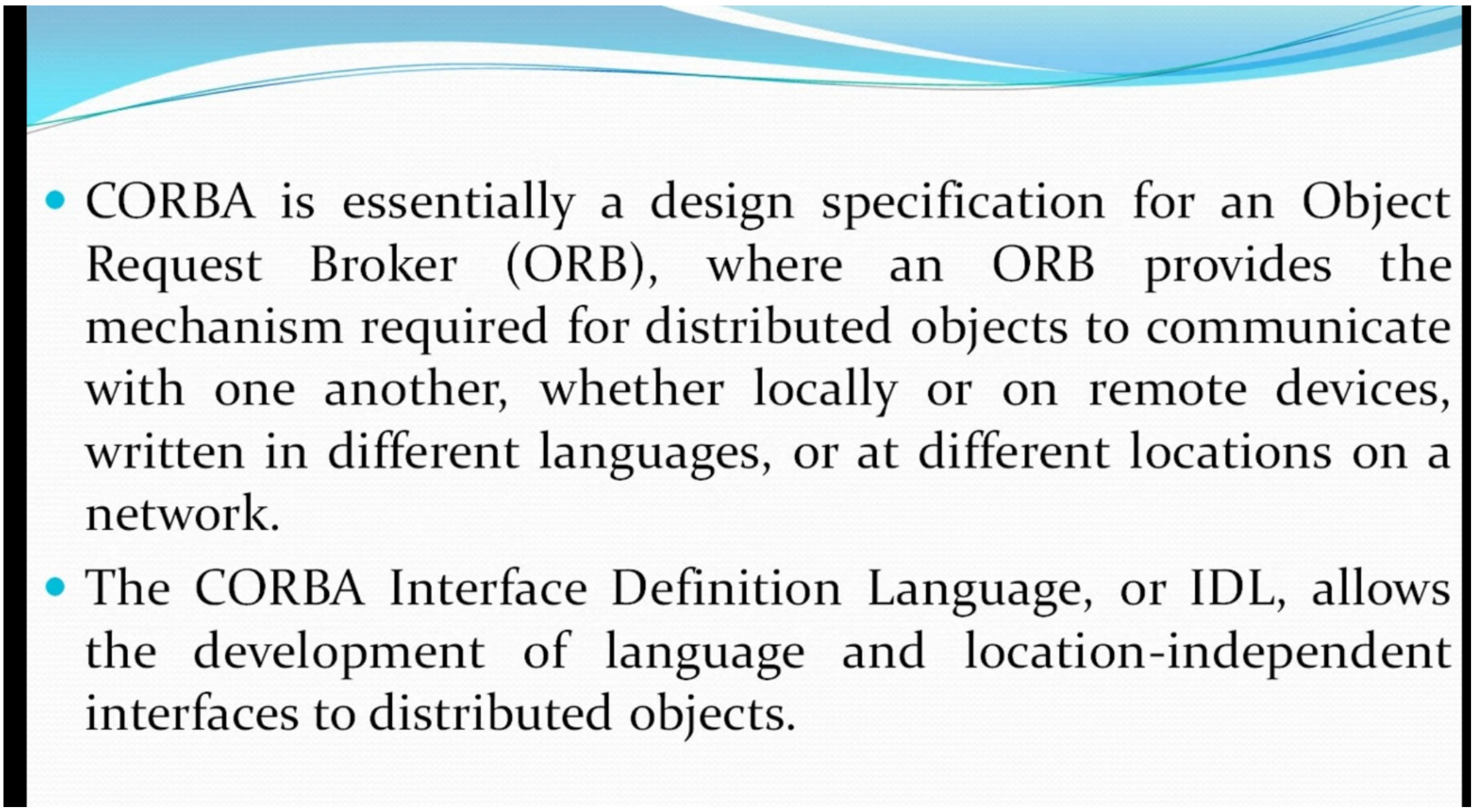
index in sequence of bytes	◄— 4 bytes —►	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20–23	"on__"	
24–27	1984	*unsigned long*



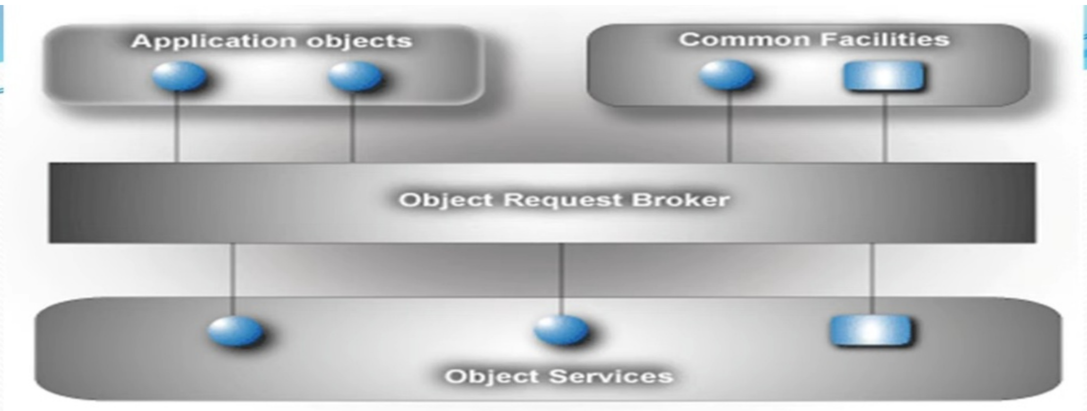
Common Object Request Broker Architecture

- The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interconnectivity among distributed objects.

- CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems.



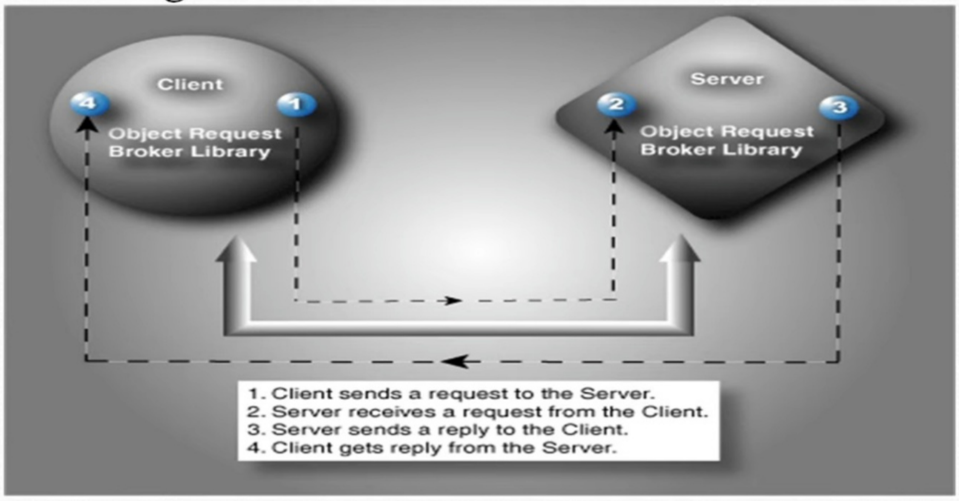
- CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.
- The CORBA Interface Definition Language, or IDL, allows the development of language and location-independent interfaces to distributed objects.



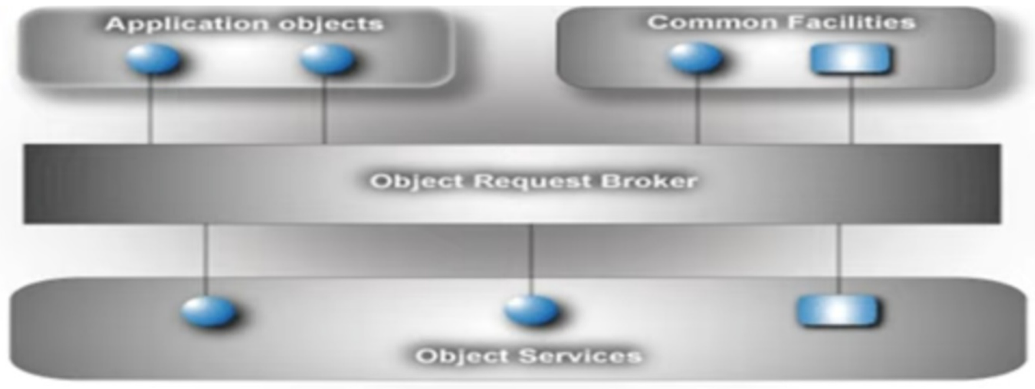
❑Data communication from client to server is accomplished through a well-defined object-oriented interface.
❑The Object Request Broker (ORB) determines the location of the target object, sends a request to that object, and returns any response back to the caller.

❑Through this object-oriented technology, developers can take advantage of features such as inheritance, encapsulation, polymorphism, and runtime dynamic binding.

- These features allow applications to be changed, modified and re-used with minimal changes to the parent interface. For Example below identifies how a client sends a request to a server through the ORB:



1. Client sends a request to the Server.
2. Server receives a request from the Client.
3. Server sends a reply to the Client.
4. Client gets reply from the Server.

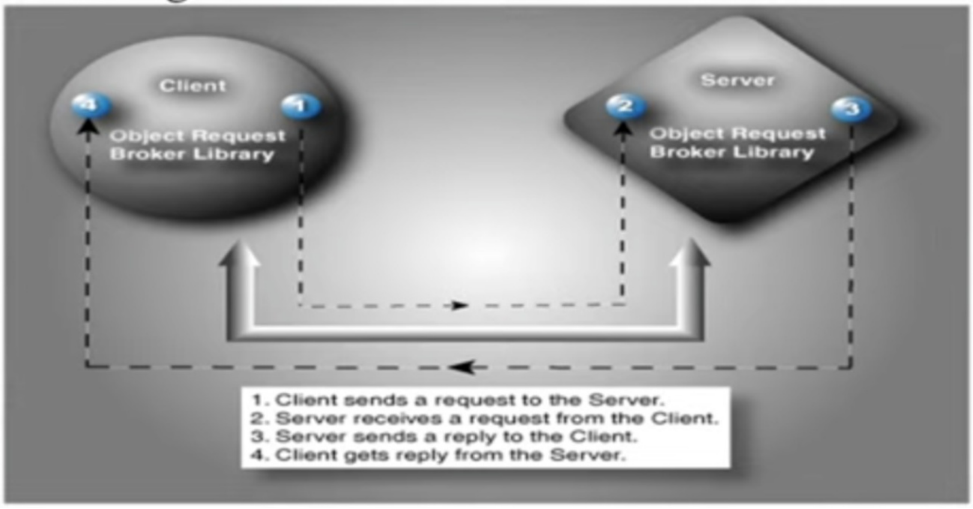


❑Data communication from client to server is accomplished through a well-defined object-oriented interface.

❑The Object Request Broker (ORB) determines the location of the target object, sends a request to that object, and returns any response back to the caller.

❑Through this object-oriented technology, developers can take advantage of features such as inheritance, encapsulation, polymorphism, and runtime dynamic binding.

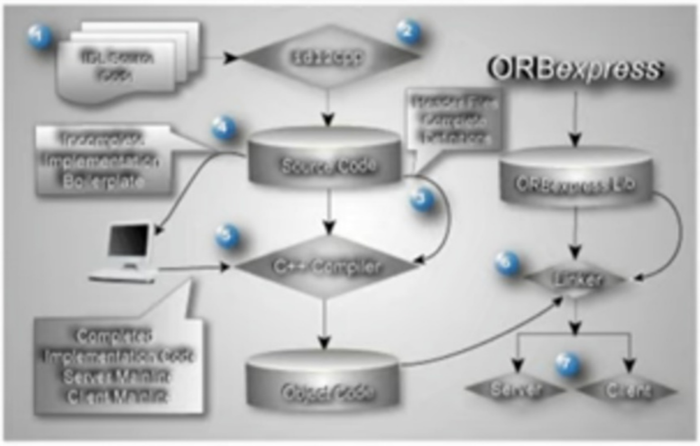
- These features allow applications to be changed, modified and re-used with minimal changes to the parent interface. For Example below identifies how a client sends a request to a server through the ORB:



1. Client sends a request to the Server.
2. Server receives a request from the Client.
3. Server sends a reply to the Client.
4. Client gets reply from the Server.

The basic steps for CORBA development can be seen in the Following Example

- An overview of how the IDL is translated to the corresponding language (in this example, C++), mapped to the source code, compiled, and then linked with the ORB library, resulting in the client and server implementation.



1. **Create the IDL to Define the Application Interfaces**
2. **Translate the IDL**
3. **Compile the Interface Files**
4. **Complete the Implementation**
5. **Compile the Implementation**
6. **Link the Application**
7. **Run the Client and Server**