Distributed Systems

Module 2
INTERPROCESS COMMUNICATION

Dr Vidya Raj C
Professor, CSE and Dean AA

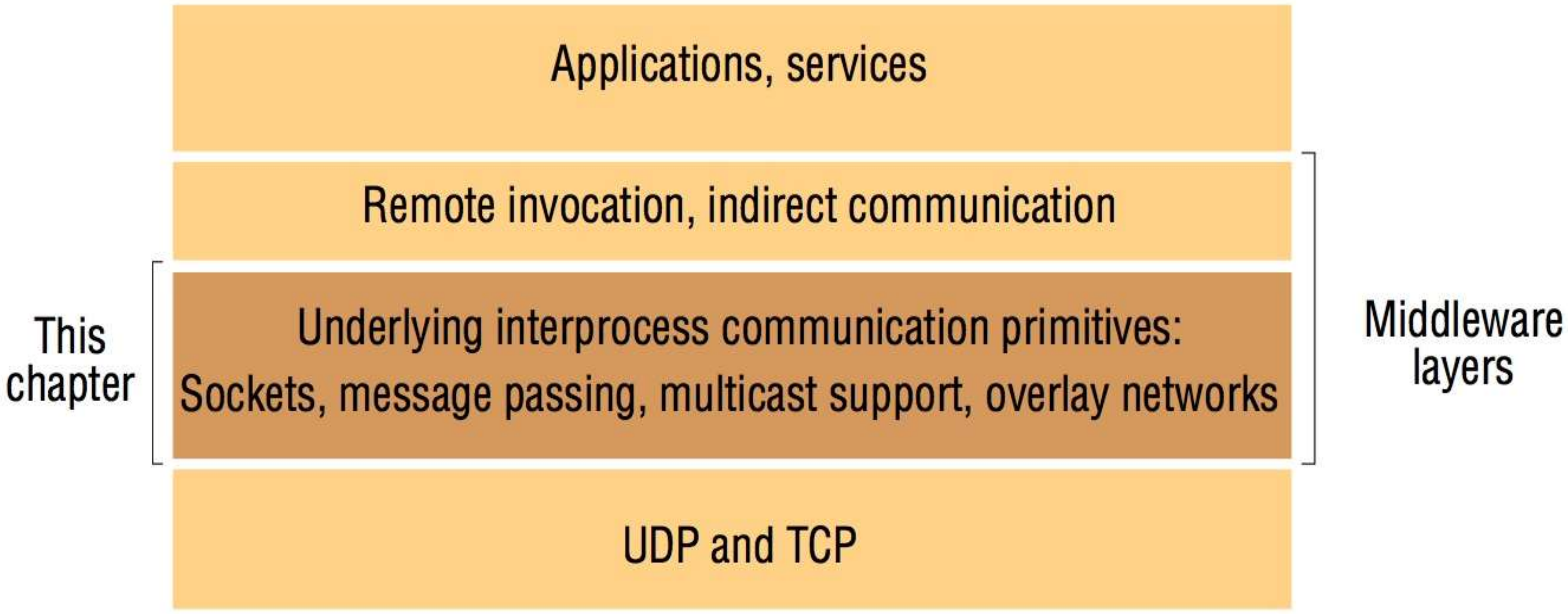
Textbook:

George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair: Distributed Systems – Concepts and Design, Fifth Edition, Pearson Publications, 2012.

Inter Process Communication

- Simplest form of IPC is Message passing
- Send and Receive are Message communication primitive
- Communication between sending and receiving processes – Synchronous and Asynchronous
- **Synchronous communication**
  – sending and receiving process are blocking operations
- **Asynchronous communication**
  – sending process is non-blocking and receiving process can be blocking or non-blocking

Figure 4.1
Middleware layers


Applications, services
Remote invocation, indirect communication
This chapter
Underlying interprocess communication primitives:
Sockets, message passing, multicast support, overlay networks
Middleware layers
UDP and TCP

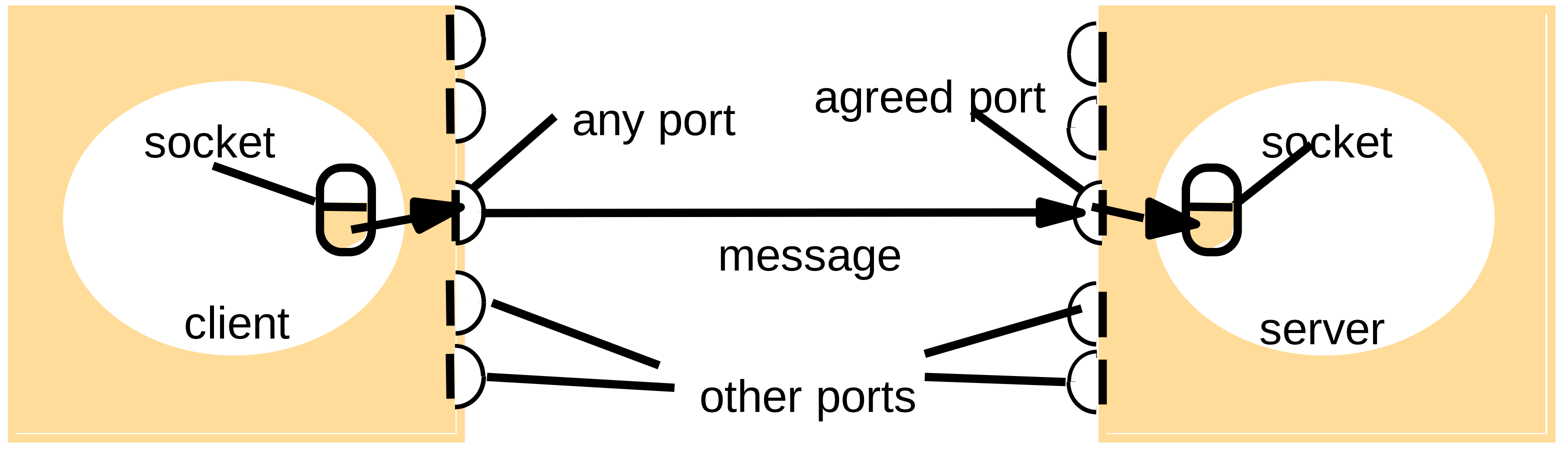
The API for the Internet protocols

The characteristics of interprocess communication:

1. Synchronous and asynchronous communication
   - Simplest form of IPC is Message passing
   - Message passing between a pair of processes is through Send and Receive operations
   - Communication between sending and receiving processes:
   - **Synchronous communication:** sending and receiving process are blocking operations
   - **Asynchronous communication:** sending process is non-blocking and receiving process can be blocking or non-blocking

Sockets and ports

•Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process
• Popular types of sockets: *Stream, Datagram, Raw*



Internet address = 138.37.94.248

Internet address = 138.37.88.249

Both forms of communication (UDP and TCP) use the *socket abstraction, which* provides an endpoint for communication between processes

UDP datagram communication

- Datagram - an independent packet containing a message
- Important issues relating to datagram communication:
  - Message size
  - Blocking
  - Timeouts
  - Receive from any

- Failure model for UDP datagram
  - Omission failure
  - Ordering

- Uses of UDP: DNS is implemented over UDP, Voice over IP

- JavaAPI provides datagram communication by means of 2 classes:
  - DatagramPacket
  - DatagramSocket

TCP stream communication

- Important characteristics:
  - Message size
  - Lost messages
  - Flow control
  - Message duplication and ordering
  - Message destinations
- Failure model
  - To satisfy integrity property
  - To satisfy validity property
- Uses of TCP: HTTP, FTP, Telnet, SMTP run over TCP
- The Java interface to TCP streams is provided in the classes *ServerSocket and Socket*

External Data Representation(XDR)

- **XDR** – an agreed standard for the representation of data structures and primitive values
- **Marshalling** – a process of taking a collection of data items and assembling them into a from suitable for transmission in a message
  - Translating of structured data items & primitive values into XDR
- **Unmarshalling** - is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination
  - Generation of primitive values from their XDR and rebuilding of the data structure

Approaches to XDR

1. **CORBA's common data representation (CDR**) , is concerned with an external representation for the structured and primitive types that can be passed as the arguments and results of remote method invocations in CORBA. It can be used by a variety of programming languages

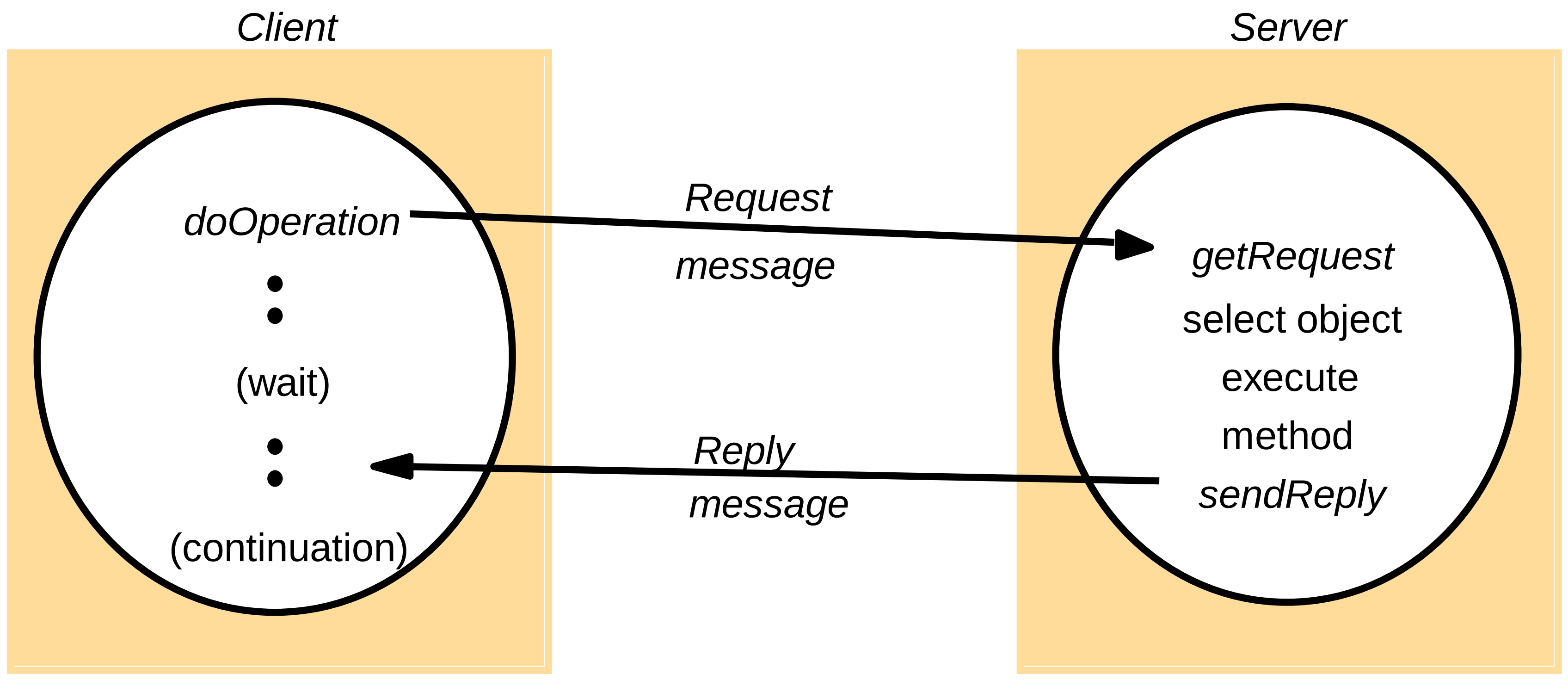
2. **Java Object serialization** is concerned with the flattening and external data representation of any single object or tree of objects that may need to be transmitted in a message or stored on a disk. It is for use only by Java.

3. **Extensible Markup Language** (XML), defines a textual format for representing structured data. Used to represent the data sent in messages exchanged by clients and servers in web services

Request-reply communication

- The basic form of communication for DS is Request Reply protocol (RR Protocol)
- Three communication primitives:
  - *doOperation*
  - *getRequest*
  - *sendReply*

Request-reply communication



The caller of *doOperation is blocked until the server performs the requested* operation and transmits a reply message to the client process.

***doOperation method**: is used by clients to invoke remote operations. Its* arguments specify the remote server and which operation to invoke, together with additional information (arguments) required by the operation.

***getRequest method** is used by a server process to acquire service requests*

***sendReply** is used by the server to send the reply message to the client at its Internet address and port*

Request-reply message structure

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*array of bytes*

- **Failure model of the request-reply protocol**
  - If the three primitives *doOperation*, *getRequest and sendReply are implemented over UDP datagrams, then they suffer from*
    - Omission failures
    - Messages are not guaranteed to be delivered in sender order
    - Failure of processes
- **Timeouts**
  - DoOperation performs timeout operation in order to break the infinite waiting time at the client. The action taken when a timeout occurs depends on the delivery guarantees

Delivery Guarantees

- Retry request message
- Discarding duplicate request messages
- Lost reply messages
- History

Styles of exchange protocols

- the *request (R) protocol;*
- the *request-reply (RR) protocol;*
- the *request-reply-acknowledge reply (RRA) protocol*

R protocol

- may be used when there is no value to be returned from the remote operation and the client requires no confirmation that the operation has been executed.
- The client may proceed immediately after the request message is sent as there is no need to wait for a reply message.
- This protocol is implemented over UDP datagrams and therefore suffers from the same communication failures

RR Protocol

- Is useful for most client-server exchanges because it is based on the request-reply protocol.
- Special acknowledgement messages are not required, because a server’s reply message is regarded as an acknowledgement of the client’s request message

The RRA protocol

- Is based on the exchange of three messages: request-reply-acknowledge reply.
- The Acknowledge reply message contains the *requestId* from the reply message being acknowledged. This will enable the server to discard entries from its history.

RPC exchange protocols

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

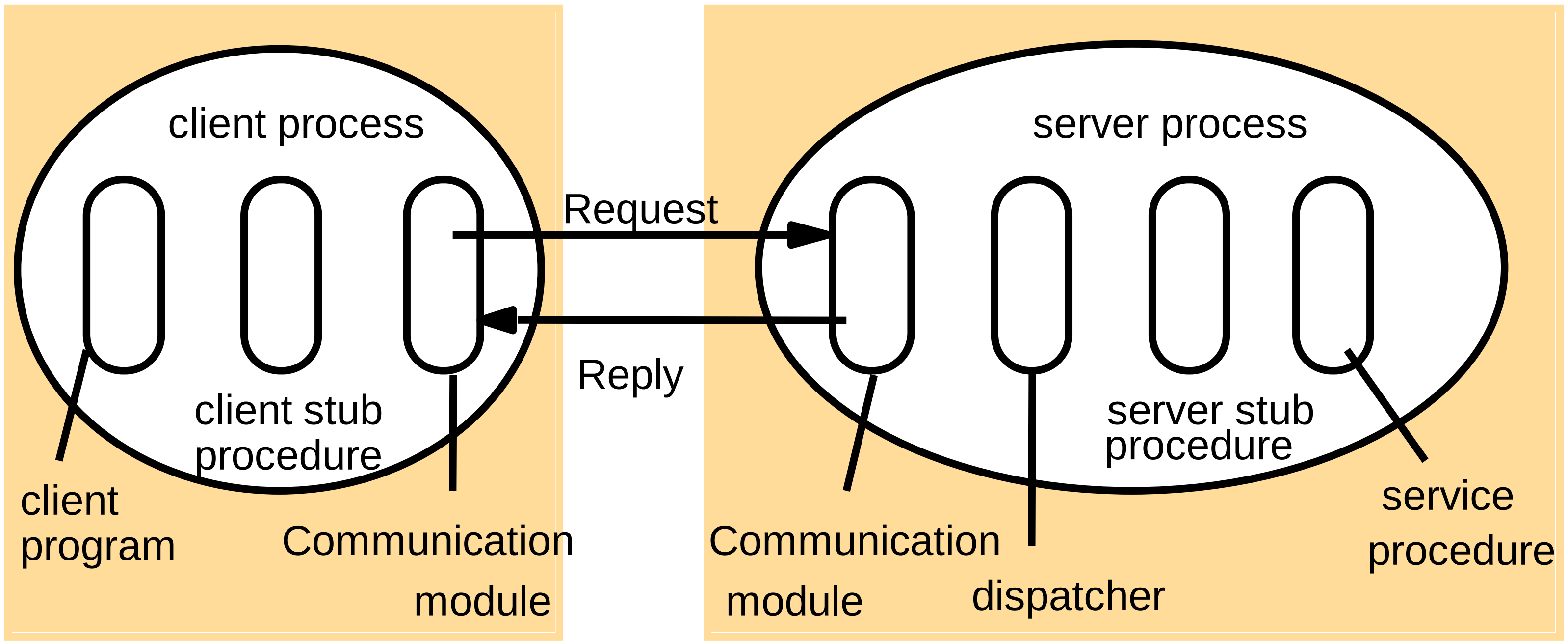
Call semantics

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

Remote Procedure Call (RPC)

- The remote procedure call (RPC) approach extends the common programming abstraction of the procedure call to distributed environments, allowing a calling process to call a procedure in a remote node as if it is local
- RPC is generally implemented over a request-reply protocol
- Design issues for RPC – IDL, Call semantics, Transparency

Role of client and server stub procedures in RPC

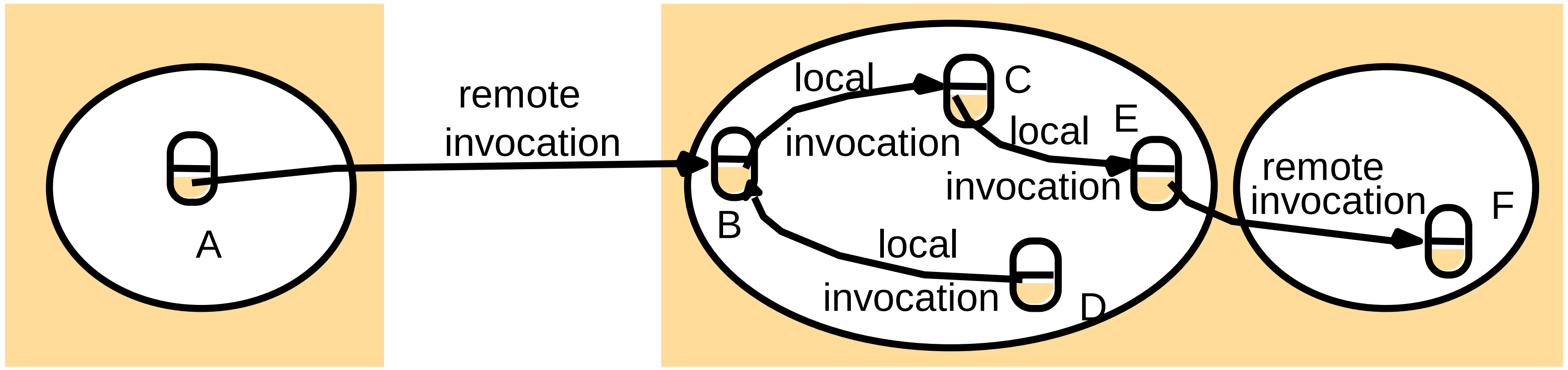


Remote Method Invocation (RMI)

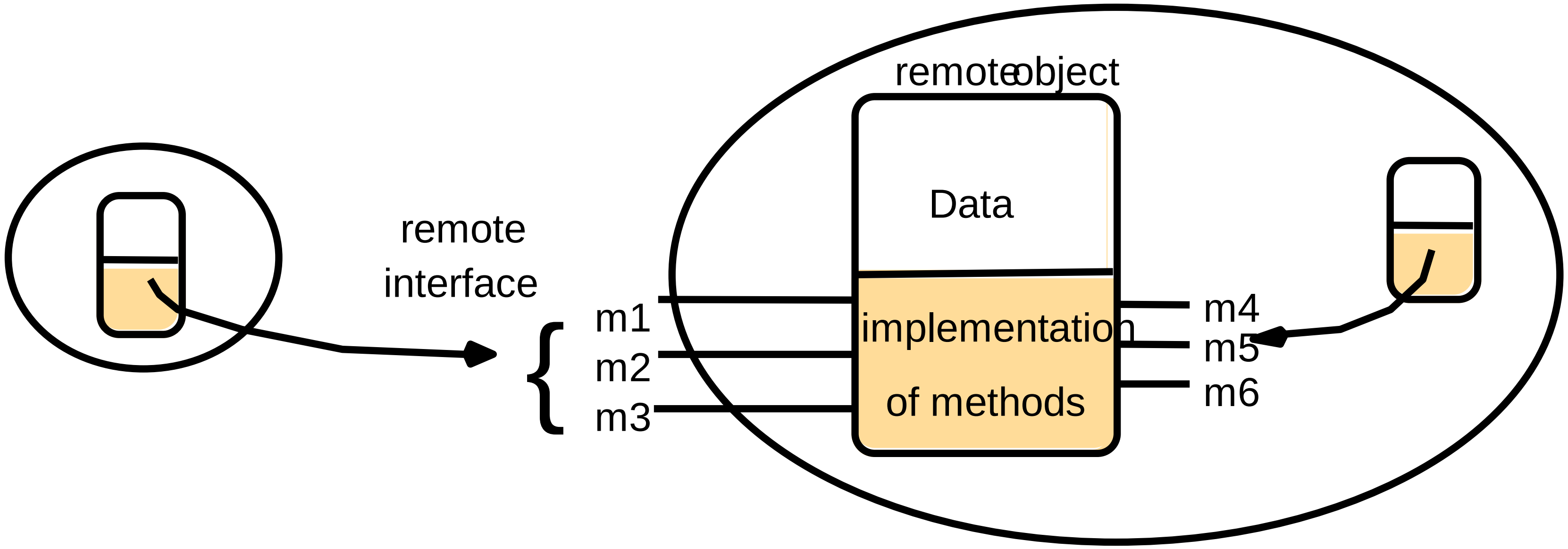
- is similar to RPC but for distributed objects, with added benefits in terms of using object-oriented programming concepts in distributed systems and also extending the concept of an object reference to the global distributed environments, and allowing the use of object references as parameters in remote invocations.

- Design issues for RMI –Object model, distributed objects, Remote object references, Remote interfaces, Garbage collection in a distributed-object system, Exceptions

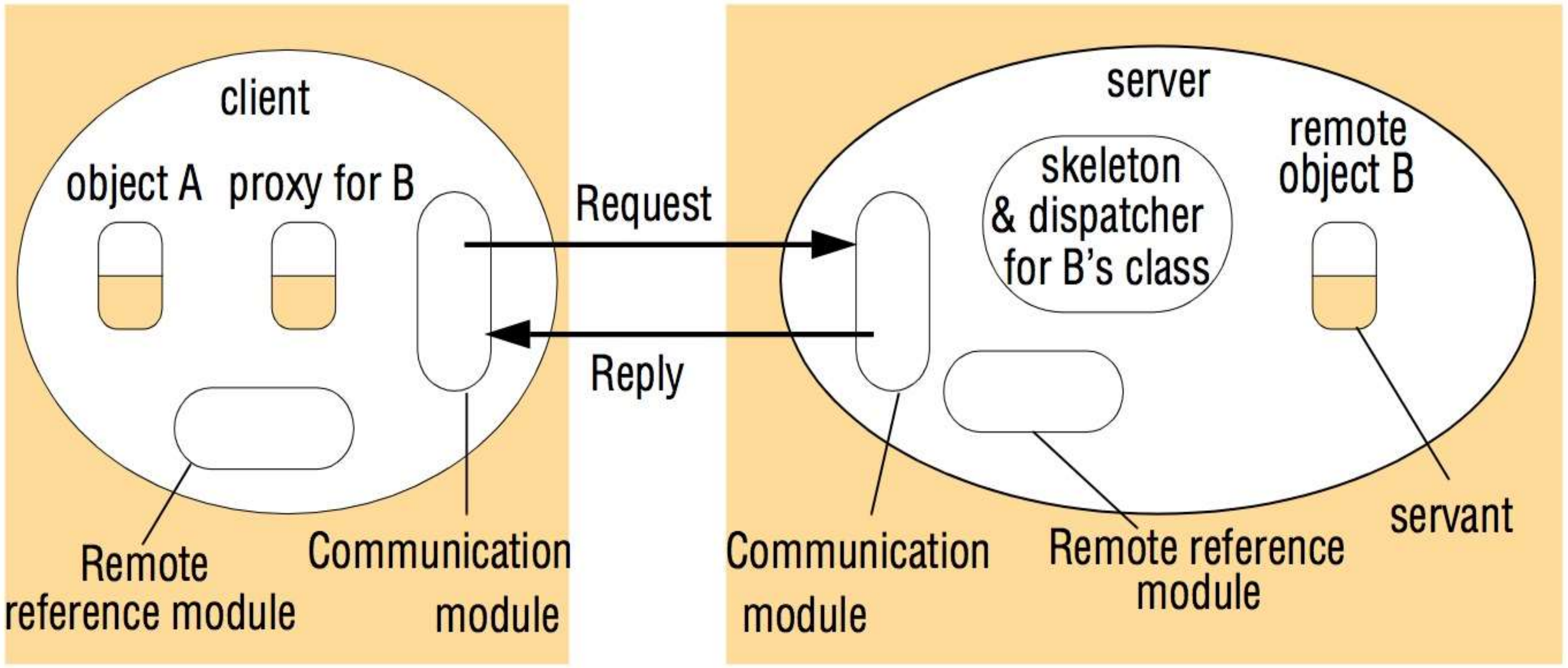
Remote and local method invocations



A remote object and its remote interface



RMI Implementation:
The role of proxy and skeleton in remote method invocation



client
object A
proxy for B
Request
Reply
server
skeleton & dispatcher for B's class
remote object B
Remote reference module
Communication module
Communication module
Remote reference module
servant