DISTRIBUTED OBJECTS AND REMOTE COMMUNICATION

DILIP KUMAR SHRESTHA

OVERVIEW

- Introduction
- The API for the Internet Protocol
- External data representation and marshalling
- Client server communication
- Group communication

INTERPROCESS COMMUNICATION

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other. The IPC interfaces make this possible. Each IPC method has its own advantages and limitations so it is not unusual for a single program to use all of the IPC methods.

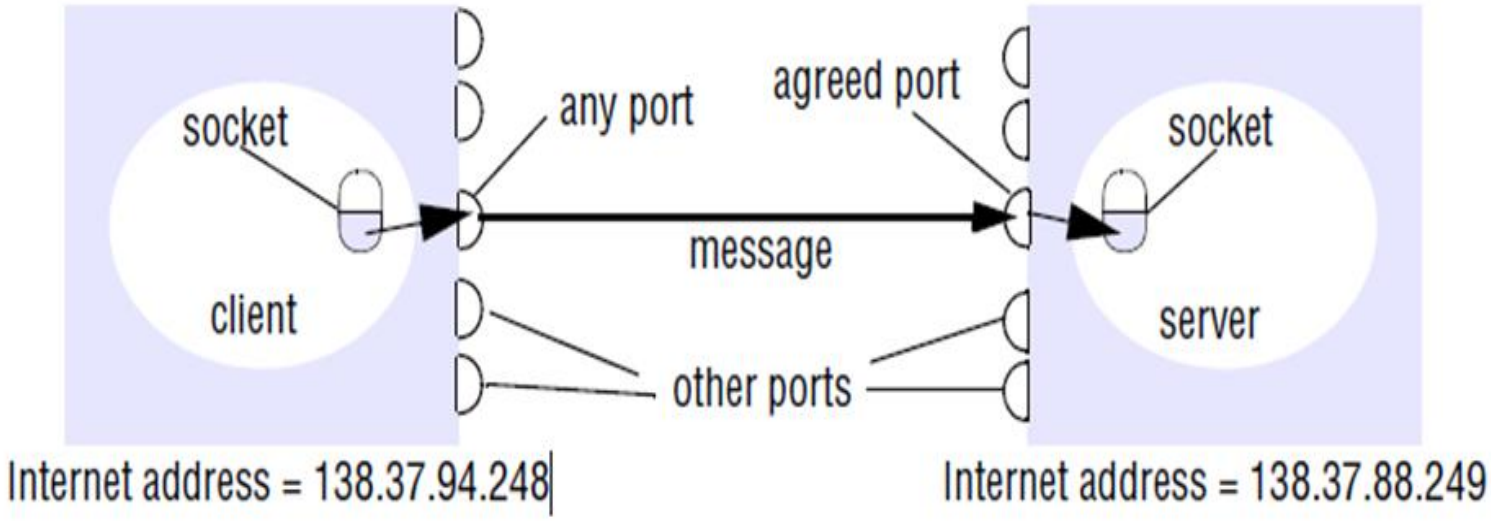
INTER-PROCESS COMMUNICATION METHODS

- Message queuing − one or more message queues sends messages between running processes and the OS kernel manages them.
- Pipes − information can only be sent in one direction and is buffered until received.
- Named pipes − a pipe has a certain name and can be used among processes that do not share a common origin.
- Shared memory − permits information exchange through a predefined area of memory and has to be allocated before data can gain access to the memory location.
- Semaphores − solves problems when synchronization or race conditions arise between processes.
- Socket − processes use these to communicate over a network via a client/server relationship.

THE API FOR THE INTERNET PROTOCOL

UDP

TCP



ISSUES

Datagram

- Packet size
- Blocking
- Timeouts
- Receive from any
- Failure model
  - Omission Failure
  - Ordering

Stream

- Message size
- Lost message
- Flow control
- Message duplication and ordering
- Message destinations
- Outstanding issues
  - Matching of data item(data format)
  - Blocking
  - Threads
- Failure Model
  - Ideal protocol

USAGES

Datagram

- DNS
- VOIP

Stream

- HTTP
- FTP
- Telnet
- SMTP

EXTERNAL DATA REPRESENTATION

- COBRA's Common Data Representation(CDR)
- Java Object Serialization
- Extensive Markup Language (XML)
- JSON (not in syllabus)

Involved Process

- Marshalling
- Unmarshalling

COBRA'S COMMON DATA REPRESENTATION(CDR)

Primitive Types

Type	Size(bit)
Short	16
Long	32
Unsigned short	16
Unsigned long	32
Float	32
Double	64
Char	1
Boolean	1
Octet	8

Composite/Constructed Types

Type	Representation
Sequence	Length(unsigned long) followed by elements in order
String	Length(unsigned long) followed by characters in order
Array	Array elements in order(fixed length)
Struct	In the order of declaration of the components
Enumerated	Unsigned long(the values are specified by the order of declaration)

EXAMPLE

```
struct Person{
        string name;
        string place;
        unsigned long year;
};
```

Let us consider 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*

JAVA OBJECT SERIALIZATION

```
Public class Person implements
Serializable{
        private String name;
        private String place;
        private int year;

        //constructors
};
```

Let us consider Person struct with value:

Person p = new Person("Smith", "London", 1984).

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name	java.lang.String place	*number, type and name of instance variables*
1984	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers; h0 and h1 are handles.

EXTENSIVE MARKUP LANGUAGE

XML

```
<person pers:id="123456789" xmlns:pers = "http://www.cdk5.net/person">
    <pers:name> Smith </pers:name>
    <pers:place> London </pers:place >
    <pers:year> 1984 </pers:year>
</person>
```

XML Schema

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions >
    <xsd:element name= "person" type ="personType" />
        <xsd:complexType name="personType">
            <xsd:sequence>
                <xsd:element name = "name"  type="xs:string"/>
                <xsd:element name = "place"  type="xs:string"/>
                <xsd:element name = "year"  type="xs:positiveInteger"/>
            </xsd:sequence>
            <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
        </xsd:complexType>
</xsd:schema>
```

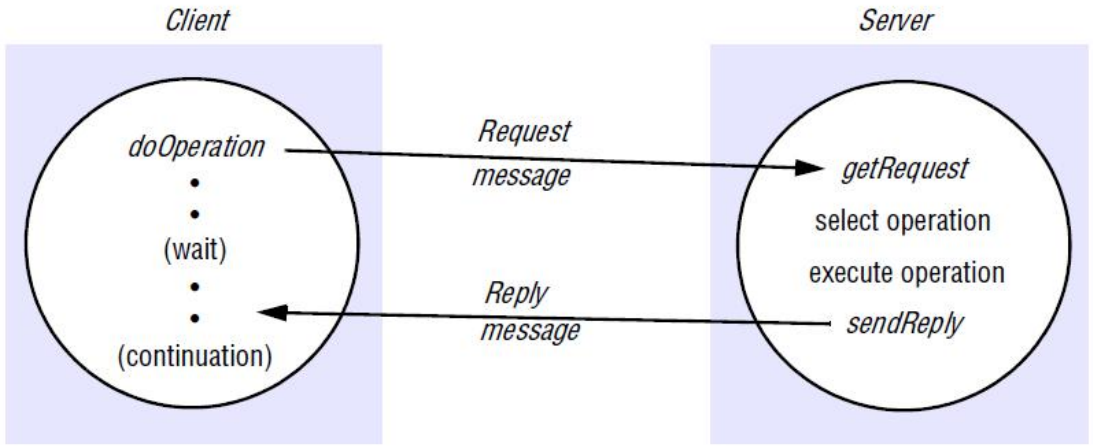
REMOTE OBJECT REFERENCE

A *remote object reference* is an identifier for a remote object that is valid throughout a distributed system. A remote object reference is passed in the invocation message to specify which object is to be invoked.

Internet Address	Port number	Time	Object number	Interface of remote object
32 bits	32 bits	32 bits	32 bits	

CLIENT SERVER COMMUNICATION

Request-Reply protocol primitives



Request-reply message structure

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

IMPLEMENTATION

Datagram

- Message identifiers
- Failure model of request reply
- Timeouts (for do operation)
- Discarding duplicate request message
- Lost reply message
- History
- RPC exchange protocols

Stream

- Message size
- Failure model
- Timeouts
- Authentication
- Content validity
- Content Negotiation

GROUP COMMUNICATION

Unicast

- Unicast is the term used to describe communication where a piece of information is sent from one point to another point. In this case there is just one sender, and one receiver.

Broadcast

- Broadcast is the term used to describe communication where a piece of information is sent from one point to all other points. In this case there is just one sender, but the information is sent to all connected receivers.

Multicast

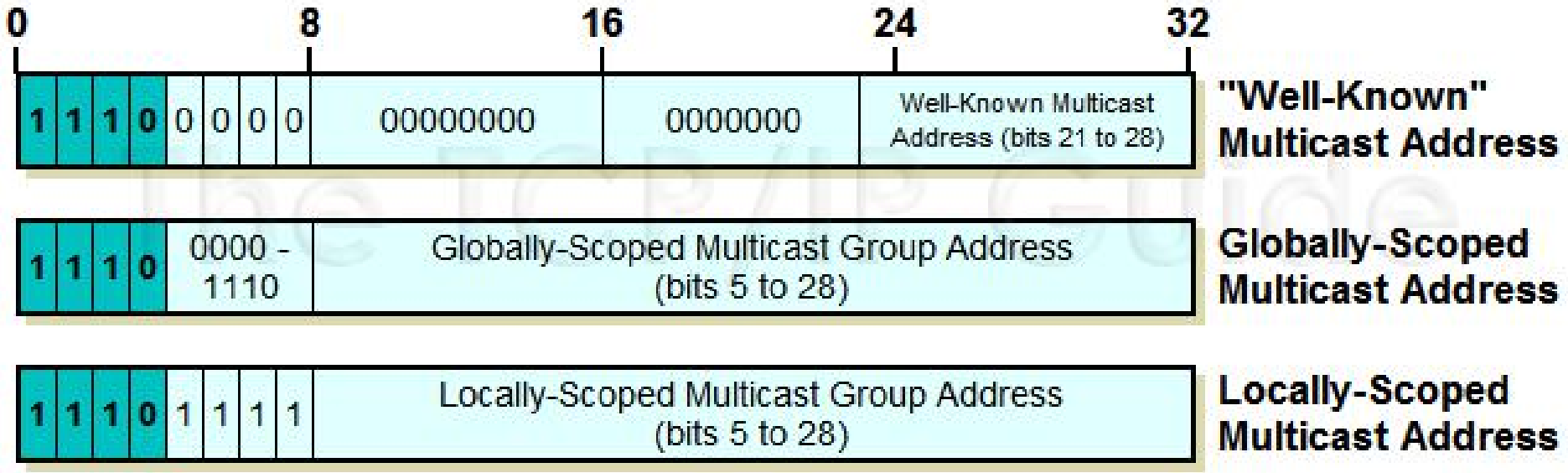
- Multicast is the term used to describe communication where a piece of information is sent from one or more points to a set of other points. In this case there is may be one or more senders, and the information is distributed to a set of receivers (theer may be no receivers, or any other number of receivers).

USAGE OF GROUP COMMUNICATION

- Fault tolerance based on replicated services
- Finding the discovery services in spontaneous network
- Better performance through replicated data
- Propagation of event notifications

MULTICAST IP RANGE

Range Start Address	Range End Address	Description
224.0.0.0	**224.0.0.255**	Reserved for special “well-known” multicast addresses.
224.0.1.0	**238.255.255.255**	Globally-scoped (Internet-wide) multicast addresses.
239.0.0.0	**239.255.255.255**	Administratively-scoped (local) multicast addresses.



COMMUNICATION BETWEEN DISTRIBUTED OBJECTS

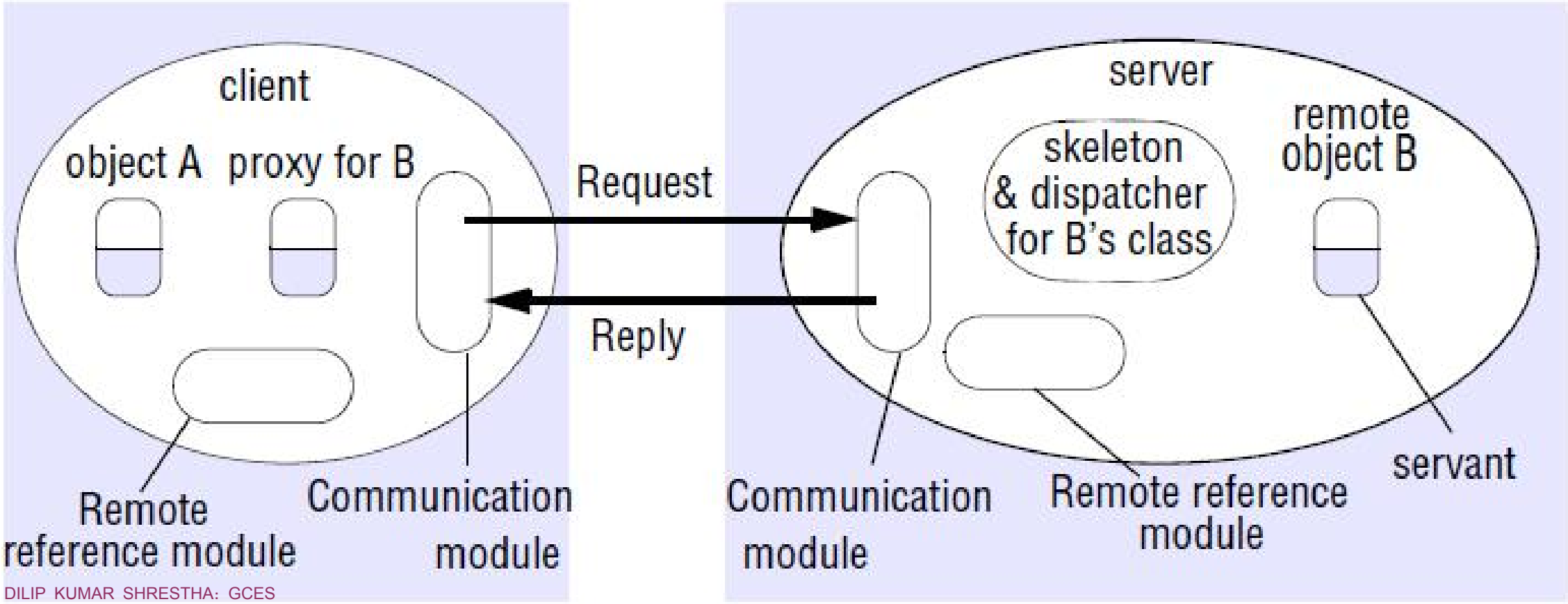
Object Model	Remote Object Model
Object reference	Remote Object reference
Interface	Remote Interface
Actions	Actions
Exception	Exception
Garbage collection	Garbage collection

REMOTE METHOD INVOCATION

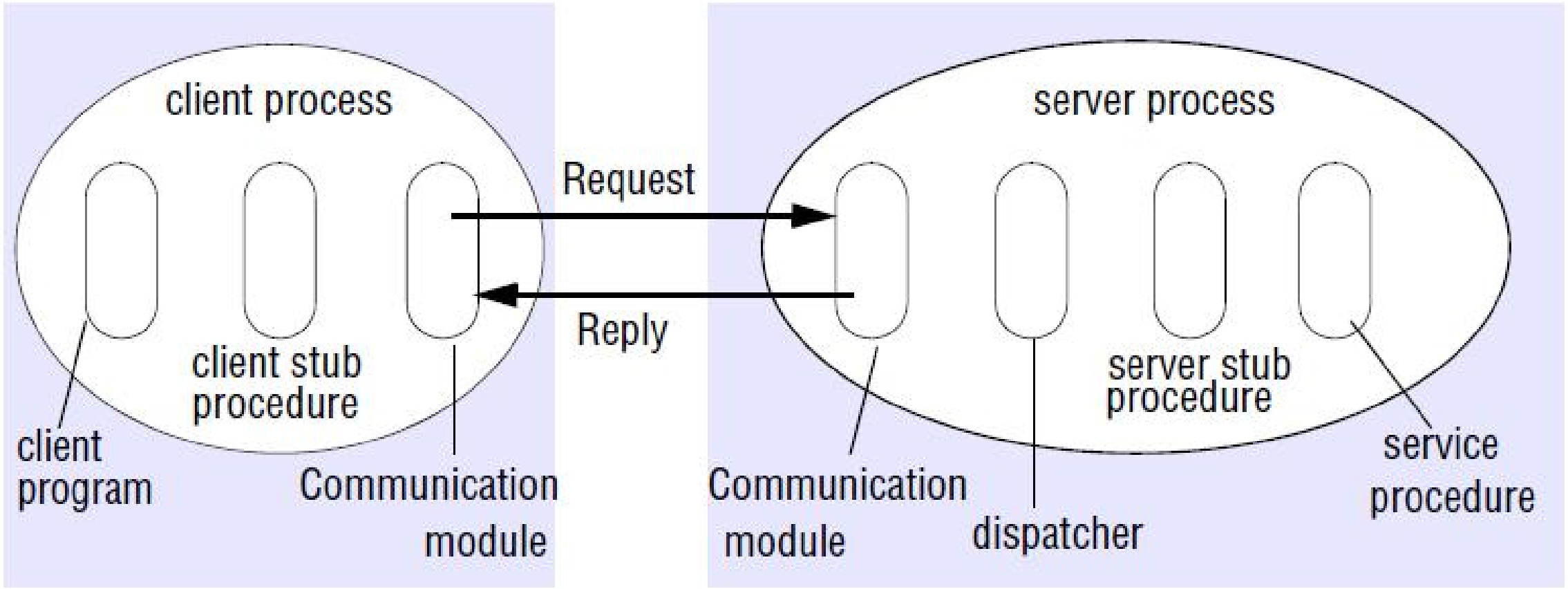
Design Issues

- RMI Invocation Semantics
  - Maybe
  - At-least-once
  - At-most-once
- Transparency
  - Syntax and semantics
  - Latency
  - Exception

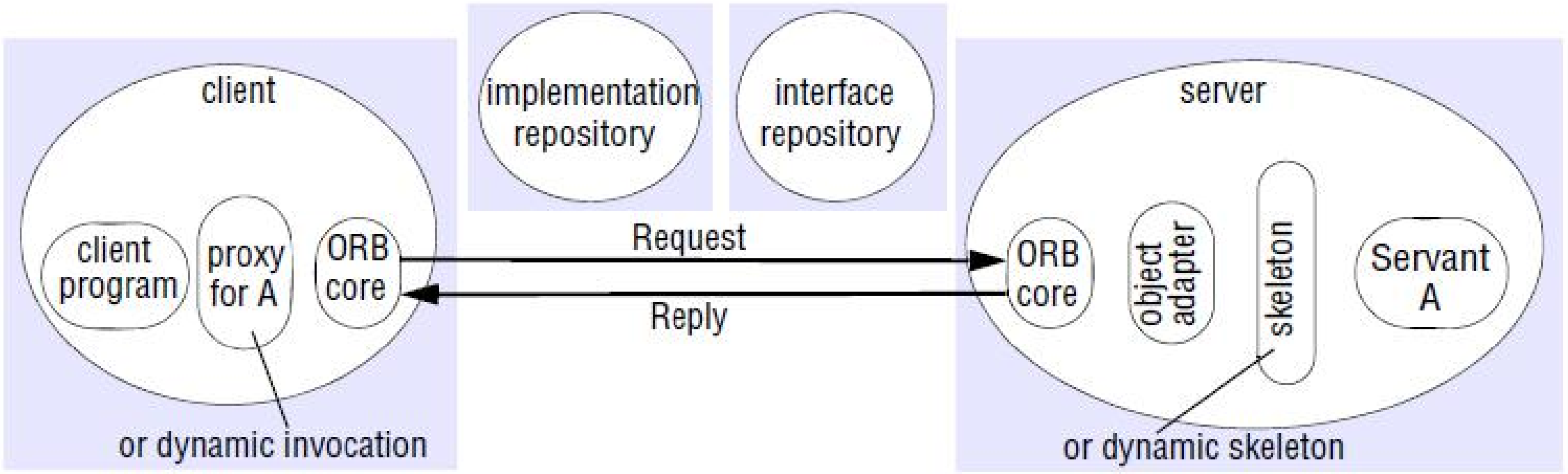
RMI IMPLEMENTATION



RPC IMPLEMENTATIONS



CORBA IMPLEMENTATION



EVENTS AND NOTIFICATIONS

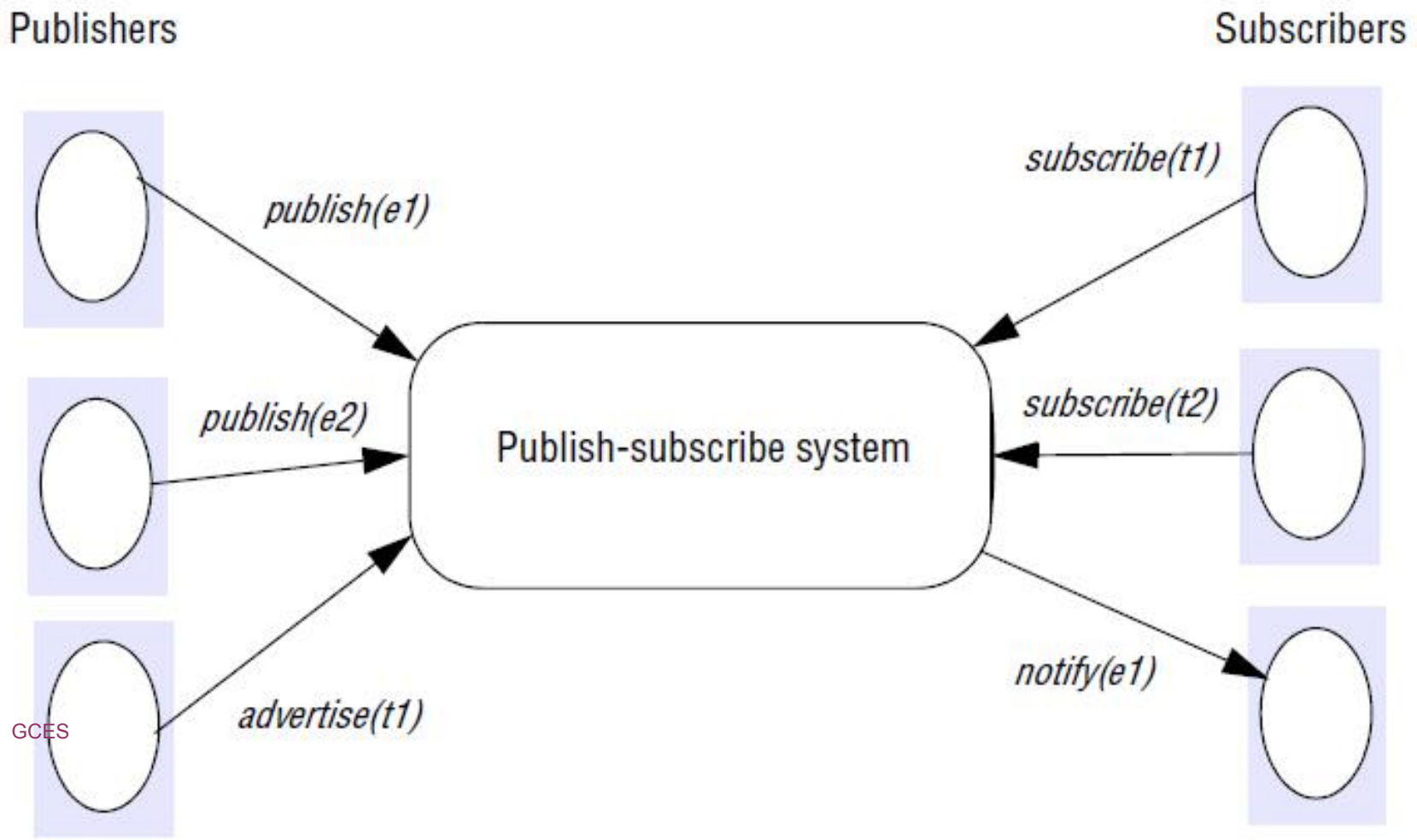
Characteristics

- Heterogenous
- Asynchronous

Participants

- The object of interest
- Event
- Notification
- Subscriber
- Observer objects
- publisher

EVENT NOTIFICATION PARADIGM



ROLES OF OBSERVERS

- Forwarding
- Filtering of notifications
- Patterns of events
- Notification mailboxes

ASSIGNMENT