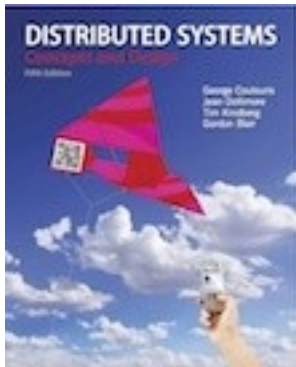


Slides for Chapter 5: Remote invocation



From **Coulouris, Dollimore, Kindberg and Blair**
**Distributed Systems:
Concepts and Design**

Edition 5, © Addison-Wesley 2012

Figure 5.1 Middleware layers

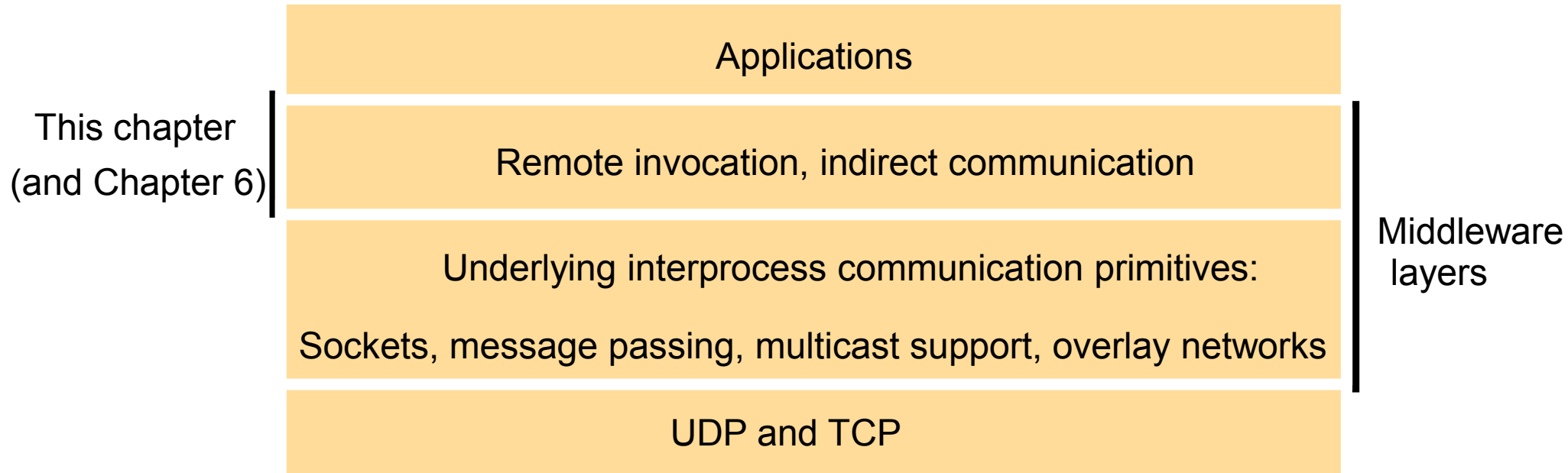


Figure 5.2

Request-reply communication

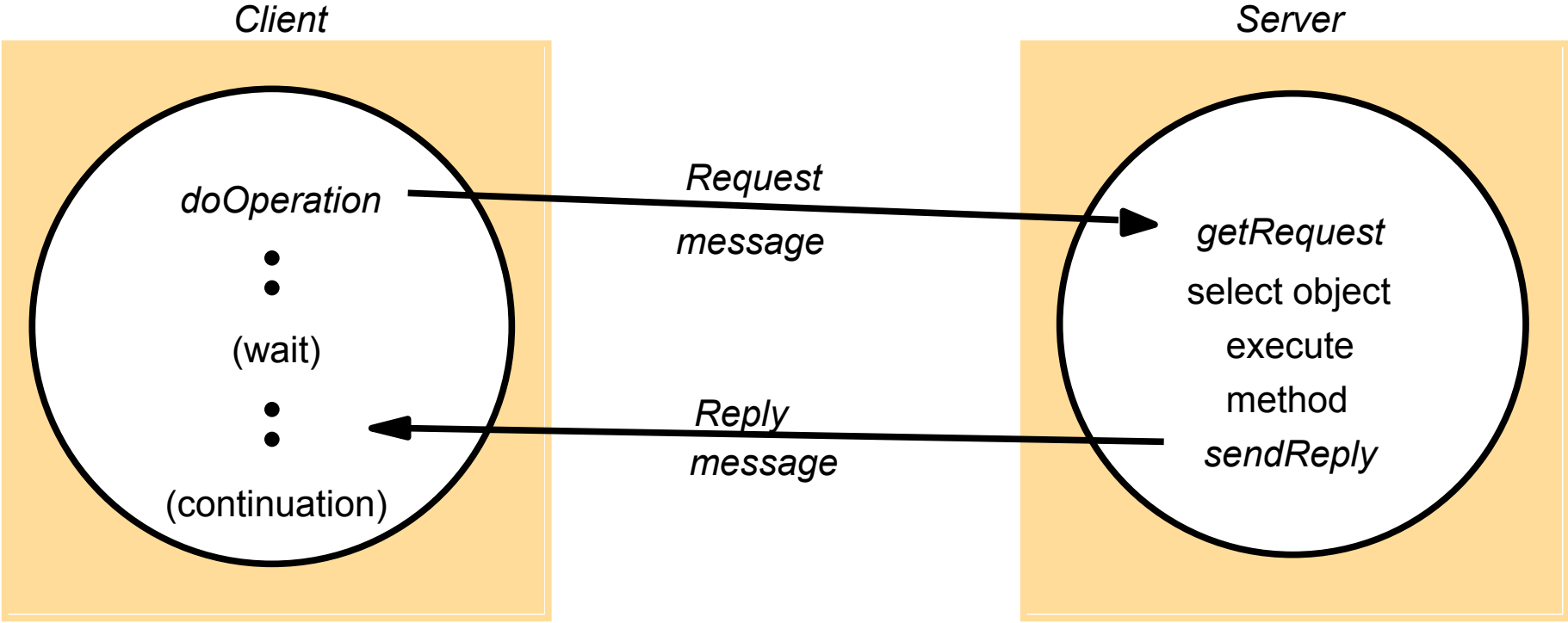


Figure 5.3

Operations of the request-reply protocol

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)

sends a request message to the remote server and returns the reply.

The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

public byte[] getRequest ();

acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

sends the reply message reply to the client at its Internet address and port.

Figure 5.4 Request-reply message structure

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

Figure 5.5 RPC exchange protocols

<i>Name</i>	<i>Messages sent by</i>		
	<i>Client</i>	<i>Server</i>	<i>Client</i>
R	<i>Request</i>		
RR	<i>Request</i>	<i>Reply</i>	
RRA	<i>Request</i>	<i>Reply</i>	<i>Acknowledge reply</i>

Figure 5.6 HTTP request message

<i>method</i>	<i>URL or pathname</i>	<i>HTTP version</i>	<i>headers</i>	<i>message body</i>
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

Figure 5.7
HTTP *Reply* message

<i>HTTP version</i>	<i>status code</i>	<i>reason</i>	<i>headers</i>	<i>message body</i>
HTTP/1.1	200	OK		resource data

Figure 5.8 CORBA IDL example

```
// In file Person.idl  
struct Person {  
    string name;  
    string place;  
    long year;  
};  
interface PersonList {  
    readonly attribute string listname;  
    void addPerson(in Person p) ;  
    void getPerson(in string name, out Person p);  
    long number();  
};
```

Figure 5.9
Call semantics

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

Figure 5.10
Role of client and server stub procedures in RPC

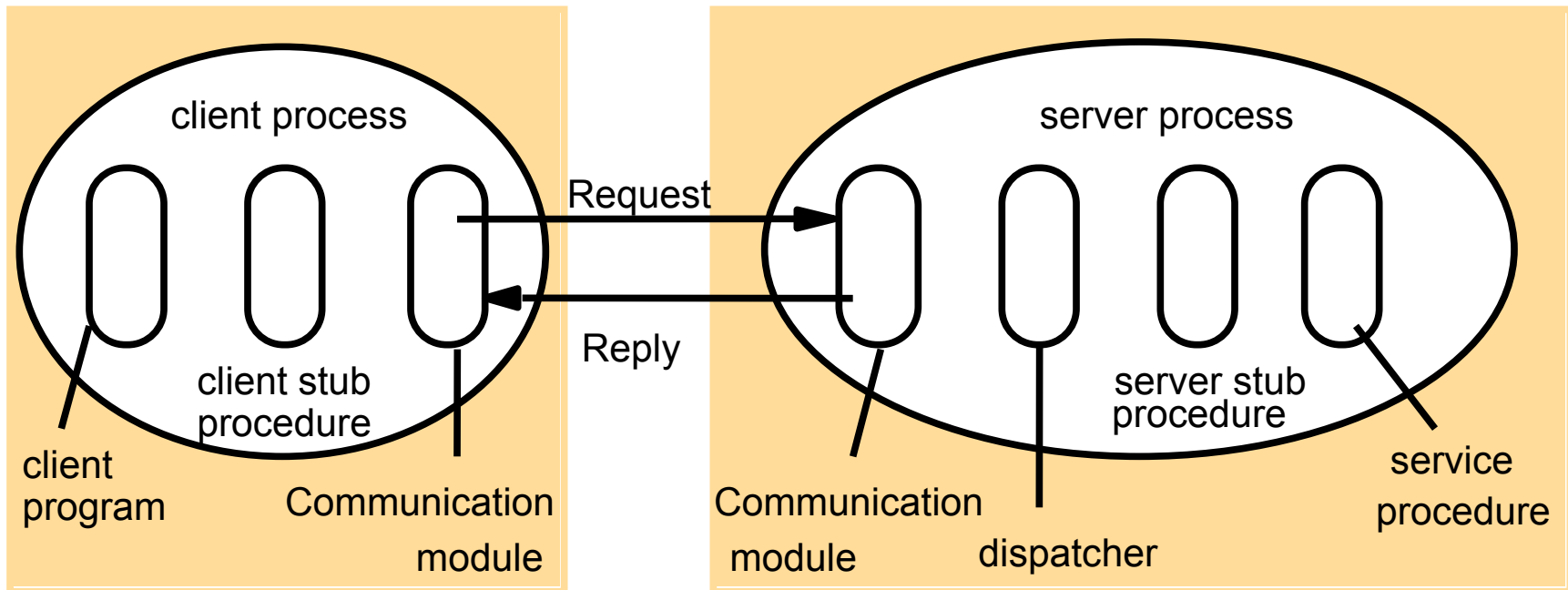


Figure 5.11 Files interface in Sun XDR

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
```

```
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
    version VERSION {
        void WRITE(writeargs)=1; 1
        Data READ(readargs)=2; 2
        }=2;
    } = 9999;
```

Figure 5.12
Remote and local method invocations

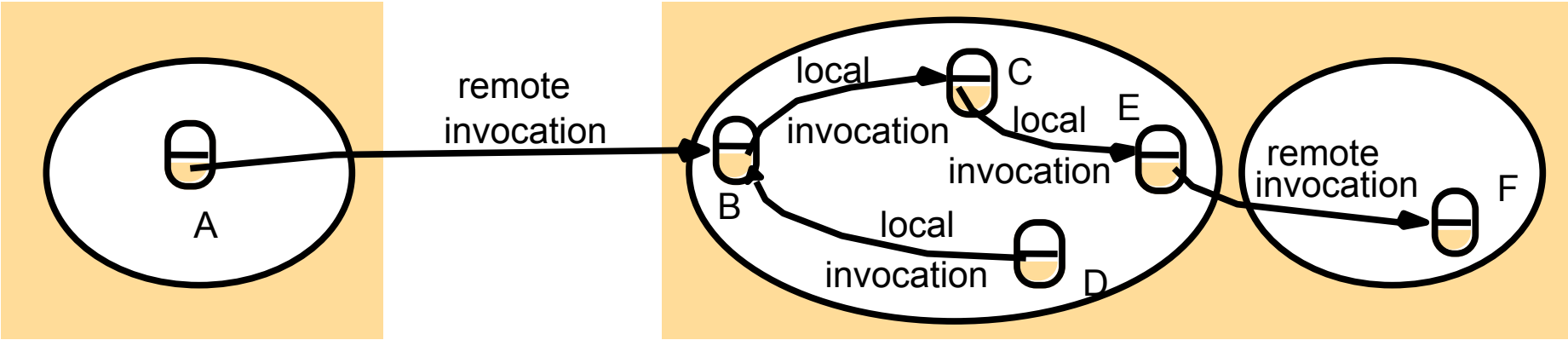


Figure 5.13
A remote object and its remote interface

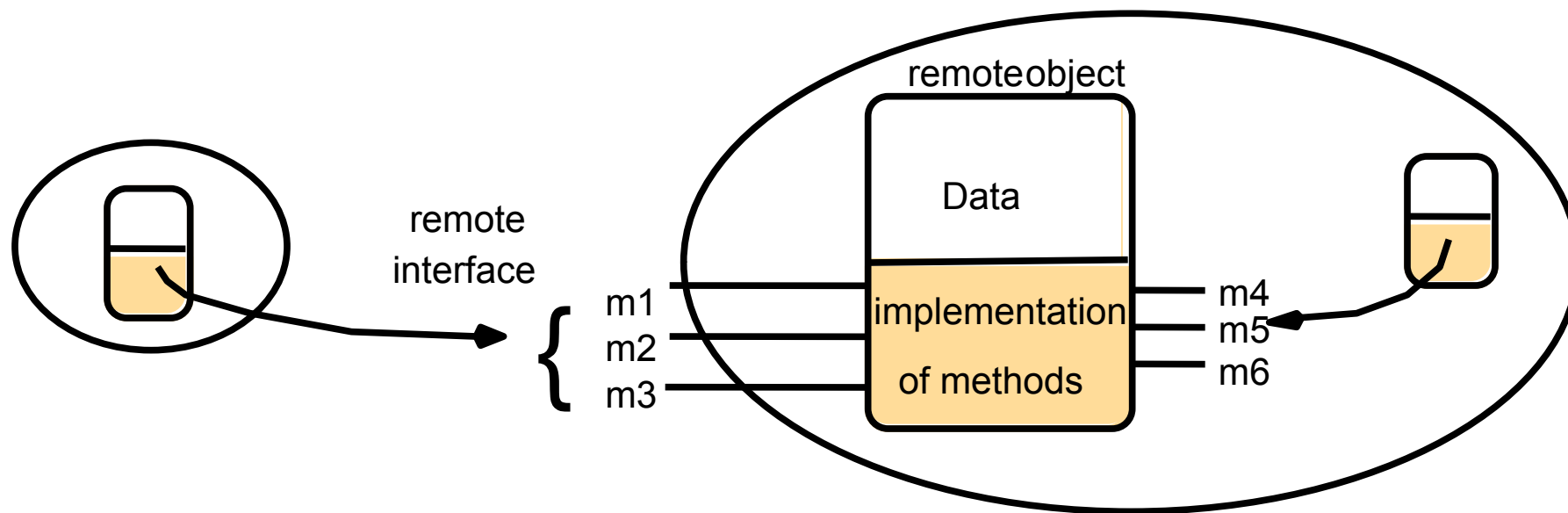


Figure 5.14
Instantiation of remote objects

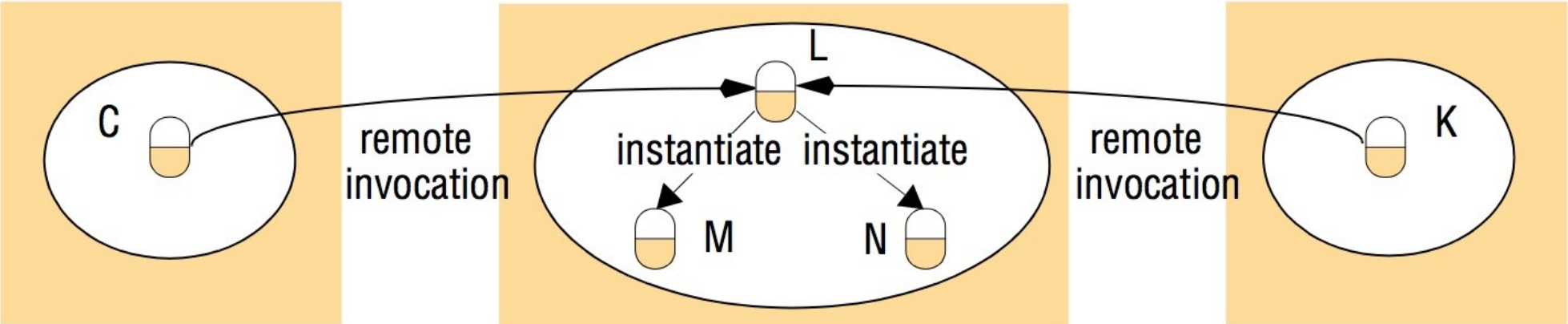


Figure 5.15

The role of proxy and skeleton in remote method invocation

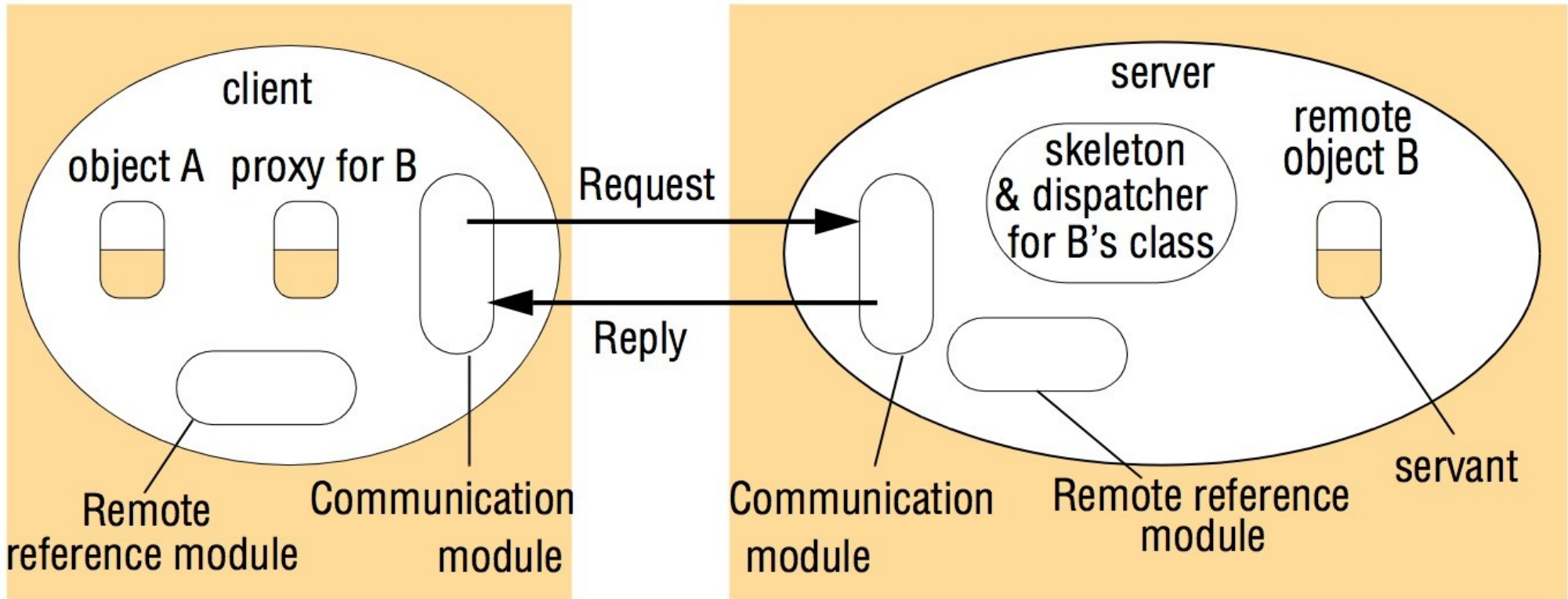


Figure 5.16

Java Remote interfaces *Shape* and *ShapeList*

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject getAllState() throws RemoteException; 1
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException; 2
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```

Figure 5.17

The *Naming* class of Java RMIregistry

void rebind (String name, Remote obj)

This method is used by a server to register the identifier of a remote object by name, as shown in Figure 15.18, line 3.

void bind (String name, Remote obj)

This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)

This method removes a binding.

Remote lookup (String name)

This method is used by clients to look up a remote object by name, as shown in Figure 5.20 line 1. A remote object reference is returned.

String [] list()

This method returns an array of Strings containing the names bound in the registry.

Figure 5.18

Java class *ShapeListServer* with *main* method

```
import java.rmi.*;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        try{
            ShapeList aShapeList = new ShapeListServant();           1
            Naming.rebind("Shape List", aShapeList );                2
            System.out.println("ShapeList server ready");
        }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());}
        }
    }
```

Figure 5.19

Java class *ShapeListServant* implements interface *ShapeList*

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeList {
    private Vector theList;           // contains the list of Shapes
    private int version;
    public ShapeListServant() throws RemoteException {...}
    public Shape newShape(GraphicalObject g) throws RemoteException {      1
        version++;
        Shape s = new ShapeServant( g, version);                             2
        theList.addElement(s);
        return s;
    }
    public Vector allShapes() throws RemoteException {...}
    public int getVersion() throws RemoteException { ... }
}
```

Figure 5.20

Java client of *ShapeList*

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        ShapeList aShapeList = null;
        try{
            aShapeList = (ShapeList) Naming.lookup("//bruno.ShapeList");
            Vector sList = aShapeList.allShapes();
        } catch(RemoteException e) {System.out.println(e.getMessage());}
        }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
    }
}
```

1
2

Figure 5.21
Classes supporting Java RMI

