COMP90015 Distributed Systems
Semester 2, 2023
Topic: Remote Invocation

Dr Tawfiq Islam
School of Computing and Information Systems (CIS)
The University of Melbourne, Australia

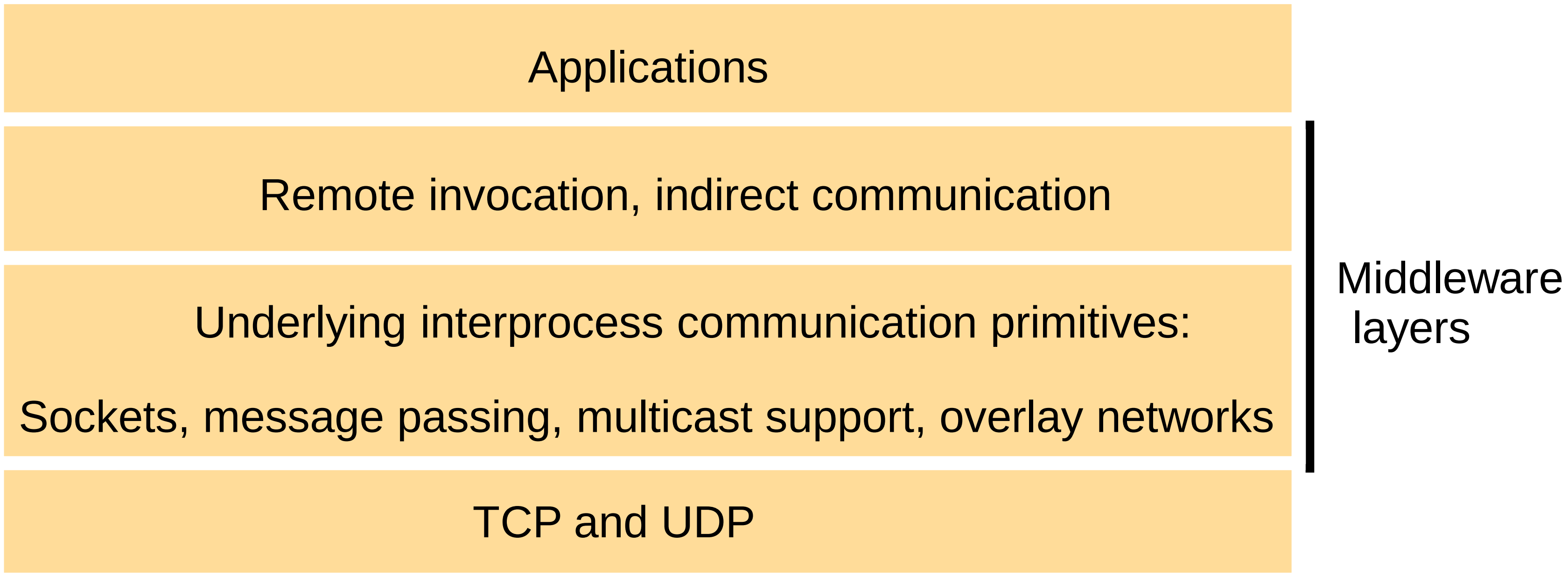
THE UNIVERSITY OF MELBOURNE

Learning Outcomes

- Request-Reply Protocol
- Invocation/Call Semantics
- Distributed Objects
- Remote Method Invocation (RMI) Architecture
- RMI Programming and a Sample Example:
  - ➢ Server-Side RMI programming
  - ➢ Client-Side RMI programming
- RPC and Summary

Introduction

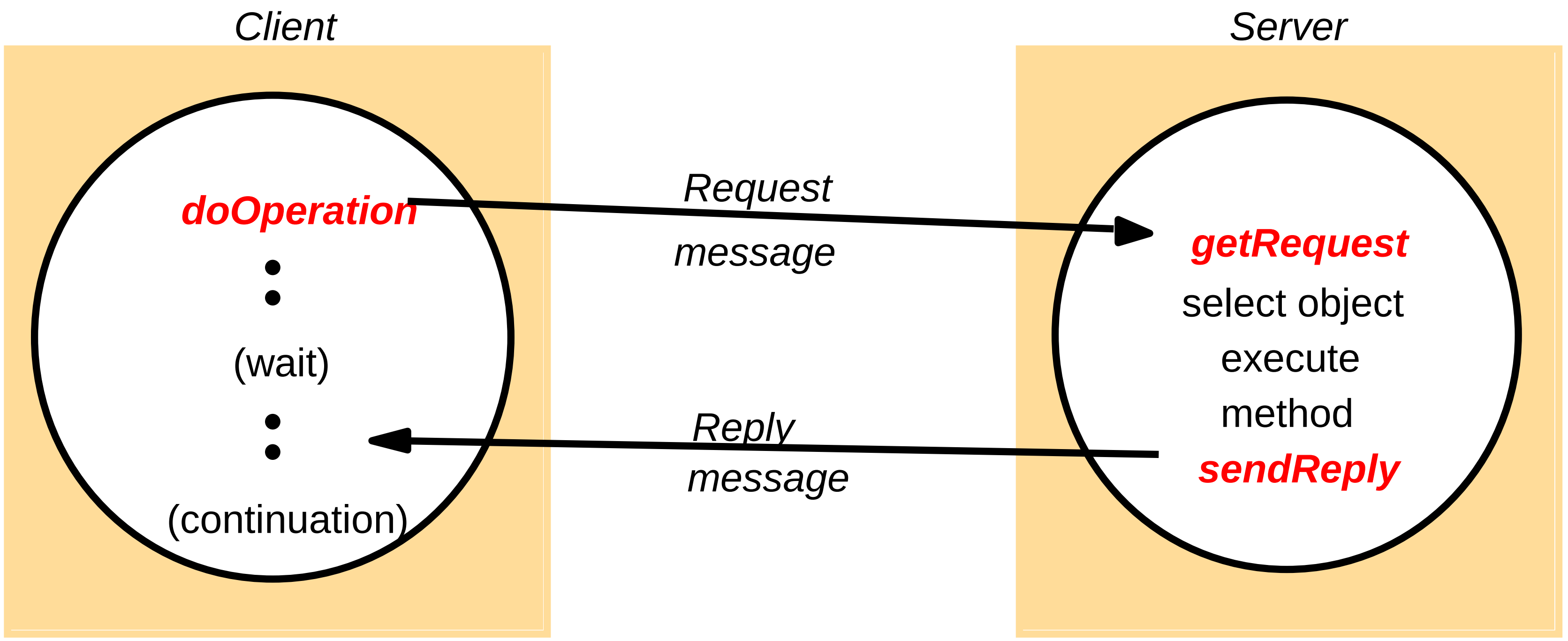
- We cover high-level programming models for distributed systems. Two widely used models are:
  - *Remote Procedure Call (RPC)* - an extension of the conventional procedure call model
  - *Remote Method Invocation (RMI)* - an extension of the object-oriented programming model.


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

THE UNIVERSITY OF MELBOURNE

Request-Reply Protocol

- Exchange protocol for the implementation of remote invocation in a distributed system.
- We discuss the protocol based on three abstract operations: doOperation, getRequest and sendReply



Request-Reply Operations

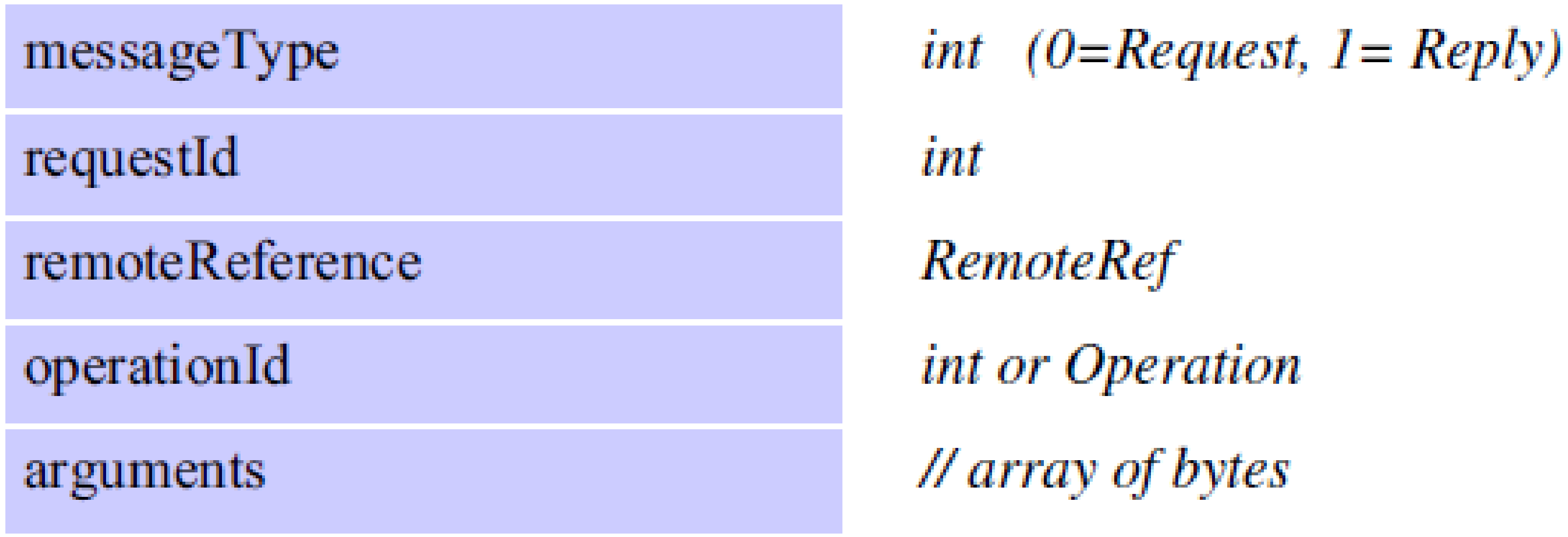
- **public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)**
  - Sends a request message to the remote server and returns the reply
  - The arguments specify the remote server, the operation to be invoked and the arguments of that operation

- **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

THE UNIVERSITY OF MELBOURNE

Request-Reply Message Structure



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

THE UNIVERSITY OF MELBOURNE

Invocation/Call Semantics

- **Middleware that implements remote invocation generally provides a certain level of semantics:**
  - **Maybe**: The remote procedure call may be executed once or not at all. Unless the caller receives a result, it is unknown as to whether the remote procedure was called.
  - **At-least-once**: Either the remote procedure was executed at least once, and the caller received a response, or the caller received an exception to indicate the remote procedure was not executed at all.
  - **At-most-once**: The remote procedure call was either executed exactly once, in which case the caller received a response, or it was not executed at all, and the caller receives an exception.
- **Java RMI (Remote Method Invocation) supports at-most-once invocation.**
  - It is supported in various editions including J2EE.
- **Sun RPC (Remote Procedure Call) supports at-least-once semantics.**
  - Popularly used in Unix/C programming environments

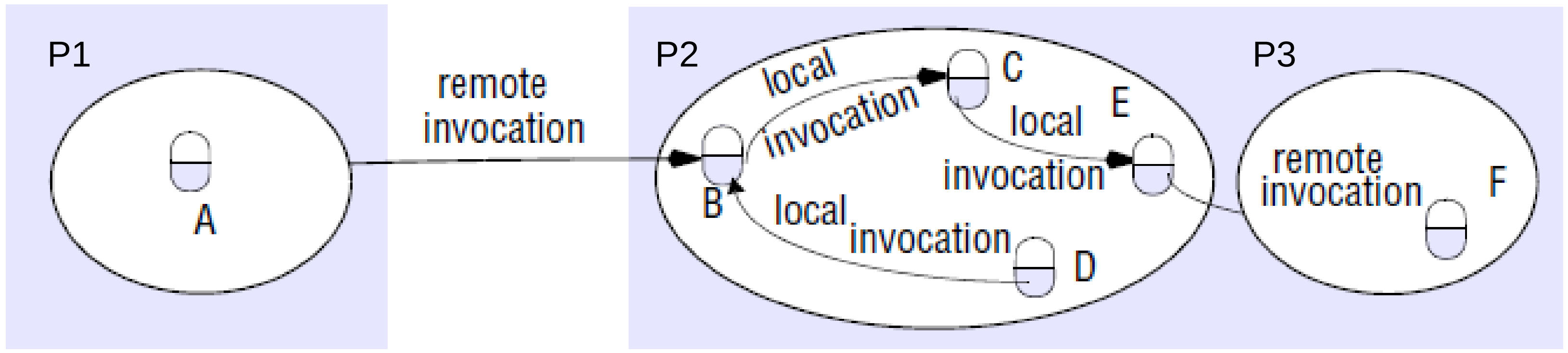
Invocation/Call Semantics

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*

Distributed Objects

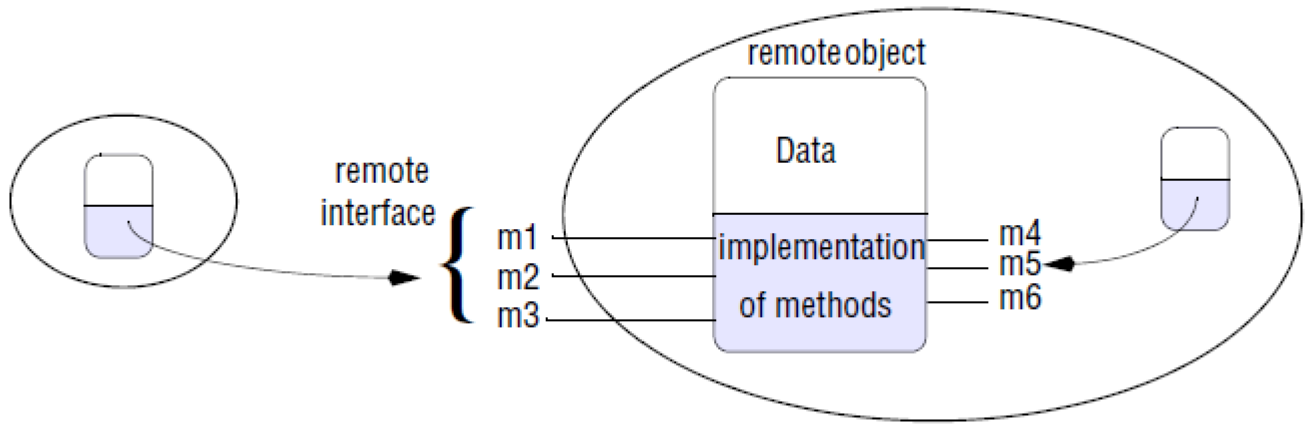
- A programming model based on Object-Oriented principles for distributed programming.
- Enables reuse of well-known programming abstractions (Objects, Interfaces, methods…), familiar languages (Java, C++, C#...), and design principles and tools (design patterns, UML…)
- Each process contains a collection of objects, some of which can receive both remote and local invocations:
  - ➢ Method invocations between objects in *different processes* are known as remote method invocation, *regardless the processes run in the same or different machines*.
- Distributed objects may adopt a client-server architecture, but other architectural models can be applied as well.

Distributed Object Model



- **Remote object references:** Other objects can invoke the methods of a remote object if they have access to its remote object reference. For example, a remote object reference for B must be available to A.
- **Remote interfaces:** Every remote object has a remote interface that specifies which of its methods can be invoked remotely. For example, the objects B and F must have remote interfaces.

Remote object and its remote interface



- Objects in other processes can invoke only the methods that belong to its remote interface.
- Local objects can invoke the methods in the remote interface as well as other methods implemented by a remote object.

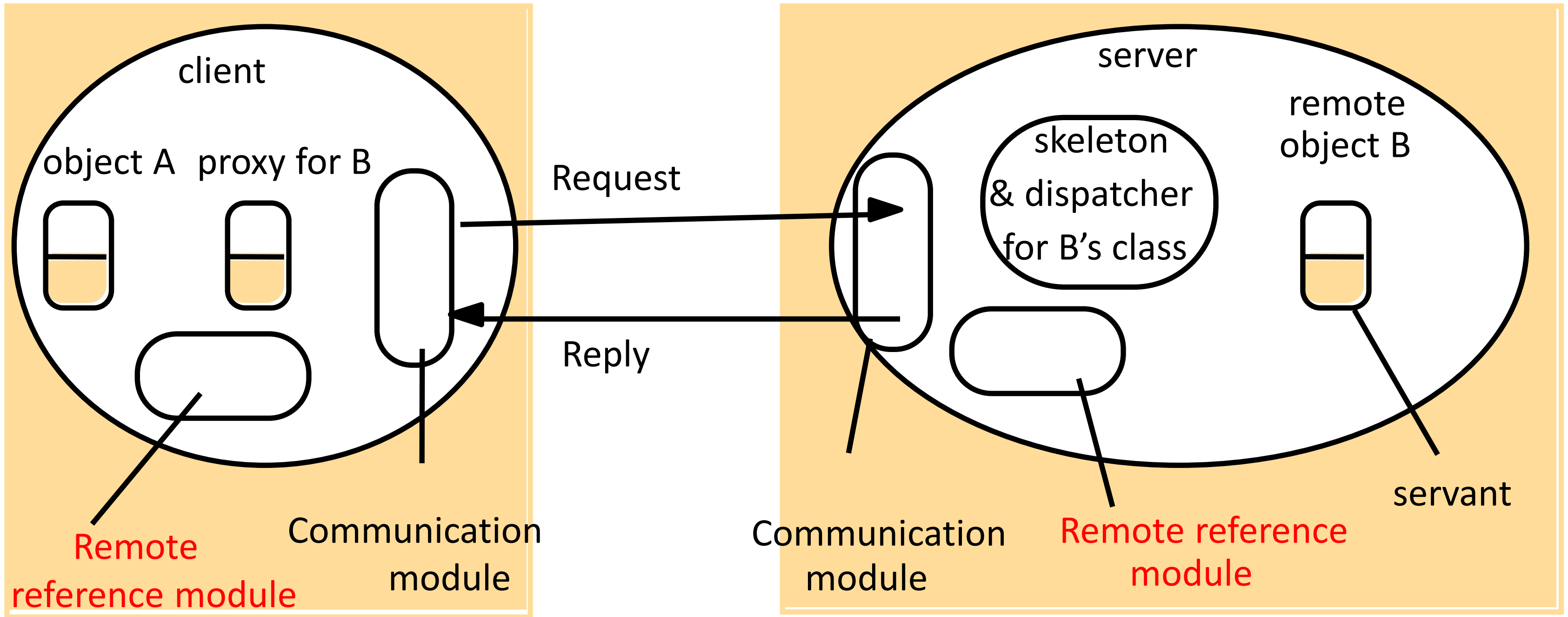
Java RMI

- Java Remote Method Invocation (Java RMI) is an extension of the Java object model to support distributed objects
  - ➢ methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts
- Single-language system with a proprietary transport protocol (JRMP, java remote method protocol)
  - ➢ Also supports IIOP (Internet Inter-Orb Protocol) from CORBA
- RMI uses object serialization to marshal and unmarshal
  - ➢ Any serializable object can be used as parameter or method return
- Releases of Java RMI
  - ➢ Java RMI is available for Java Standard Edition (JSE), Java Micro Edition (JME), and Java Enterprise Edition (Java EE)

THE UNIVERSITY OF MELBOURNE

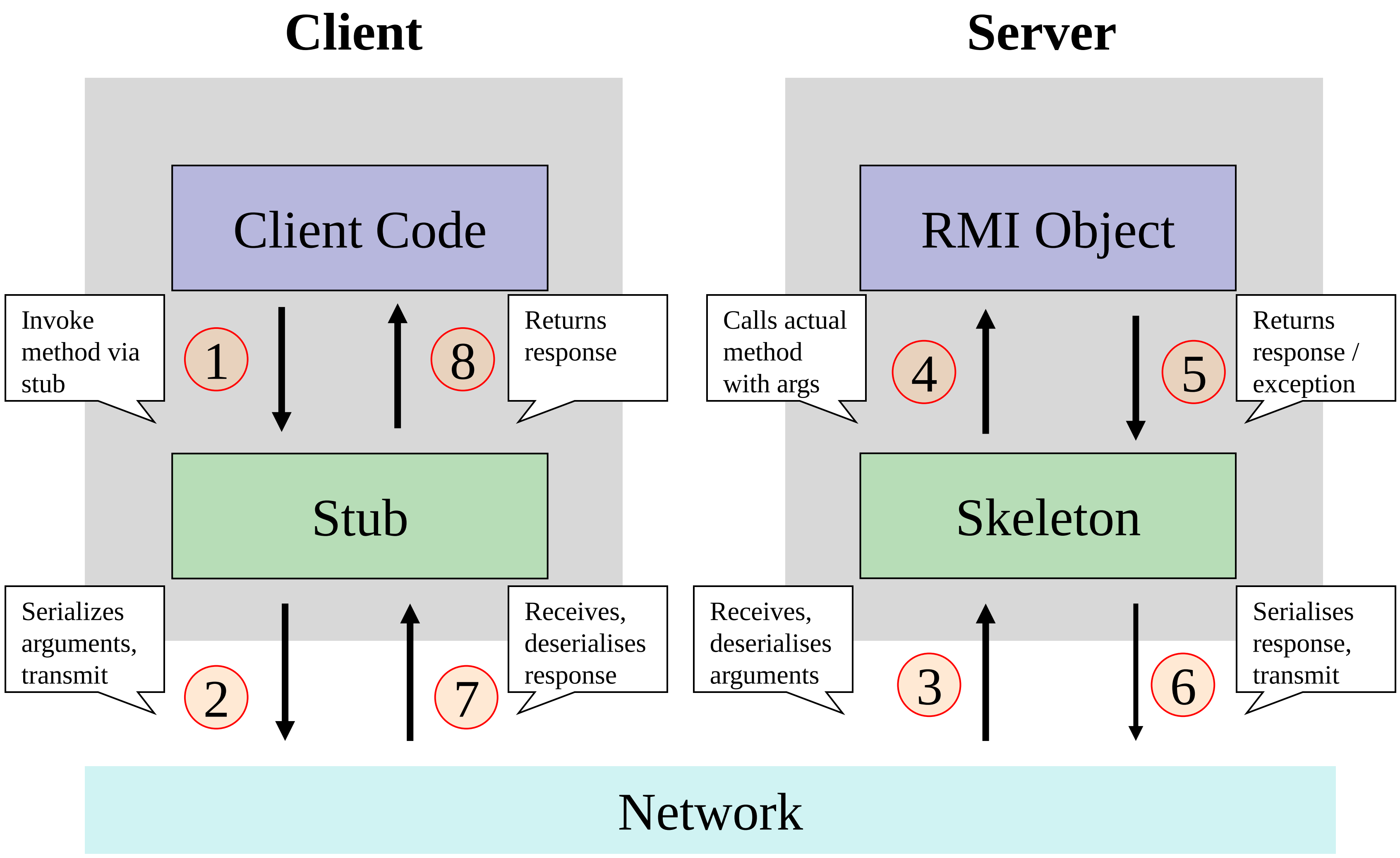
RMI Architecture and Components

- Remote reference module (at client & server) is responsible for providing addressing to the proxy (stub) object
- Proxy is used to implement a stub and provide transparency to the client. It is invoked directly by the client (as if the proxy itself was the remote object), and then marshal the invocation into a request
- Communication module is responsible for networking
- Dispatcher selects the proper skeleton and forward message to it
- Skeleton un-marshals the request and calls the remote object



THE UNIVERSITY OF MELBOURNE

Invocation Lifecycle


Client
Server
Client Code
RMI Object
Invoke method via stub
1
8
Returns response
Calls actual method with args
4
5
Returns response / exception
Stub
Skeleton
Serializes arguments, transmit
2
7
Receives, deserialises response
Receives, deserialises arguments
3
6
Serialises response, transmit
Network

THE UNIVERSITY OF MELBOURNE

Steps for implementing an RMI application

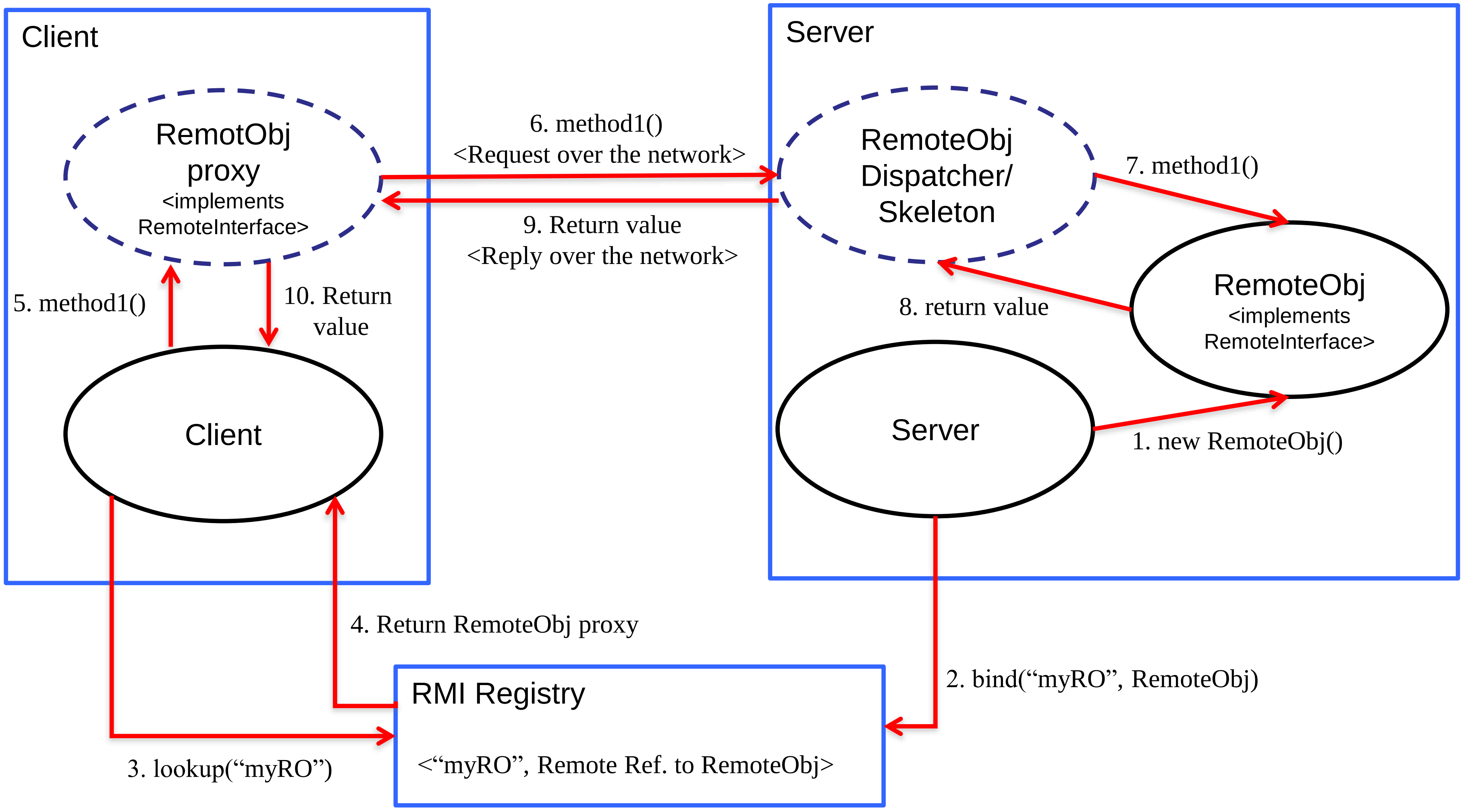
- Design and implement the components of your distributed application
  - Remote interface
  - Servant program
  - Server program
  - Client program
- Compile source code and generate stubs
  - Client proxy stub
  - Server dispatcher and skeleton
- Make classes network accessible
  - Distribute the application on server side
- Start the application

RMI Programming and Examples

- Key RMI components
  - Remote Interface
    - Exposes the set of methods and properties available
    - Defines the contract between the client and the server
    - Constitutes the root for both stub and skeleton
  - Servant component
    - Represents the remote object (skeleton)
    - Implements the remote interface
  - Server component
    - Main driver that makes available the servant
    - It usually registers with the naming service
  - Client component

THE UNIVERSITY OF MELBOURNE

Java RMI Workflow



Client
RemotObj proxy <implements RemoteInterface>
5. method1()
10. Return value
Client
6. method1() <Request over the network>
9. Return value <Reply over the network>
Server
RemoteObj Dispatcher/ Skeleton
7. method1()
8. return value
RemoteObj <implements RemoteInterface>
Server
1. new RemoteObj()
2. bind(“myRO”, RemoteObj)
4. Return RemoteObj proxy
RMI Registry
<“myRO”, Remote Ref. to RemoteObj>
3. lookup(“myRO”)

THE UNIVERSITY OF MELBOURNE

Example RMI Application – Hello World

- **Server side**
  - Create a HelloWorld interface
  - Implement HelloWorld interface with methods
  - Create a main method to register the HelloWorld service in the RMI Name Registry
  - Generate Stubs and Start RMI registry
  - Start Server
- **Client side**
  - Write a simple Client with main to lookup HelloWorld Service and invoke the methods

1. Define Interface of remote method

```
//file: Hello.java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloWorld extends Remote {
   public String sayHello(String who) throws RemoteException;
}
```

2. Define RMI Server Program

```
// file: HelloWorldServer.java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloServer extends UnicastRemoteObject implements Hello {
    public HelloServer() throws RemoteException {
        super();
    }

    @Override
    public String sayHello() throws RemoteException {
        return "Hello, World!";
    }

    public static void main(String[] args) {
        try {
            java.rmi.registry.LocateRegistry.createRegistry(1099);
            HelloServer obj = new HelloServer();
            java.rmi.Naming.rebind("HelloServer", obj);
            System.out.println("HelloServer bound in registry.");
        } catch (Exception e) {
            System.err.println("HelloServer exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

THE UNIVERSITY OF MELBOURNE

3. Define Client Program

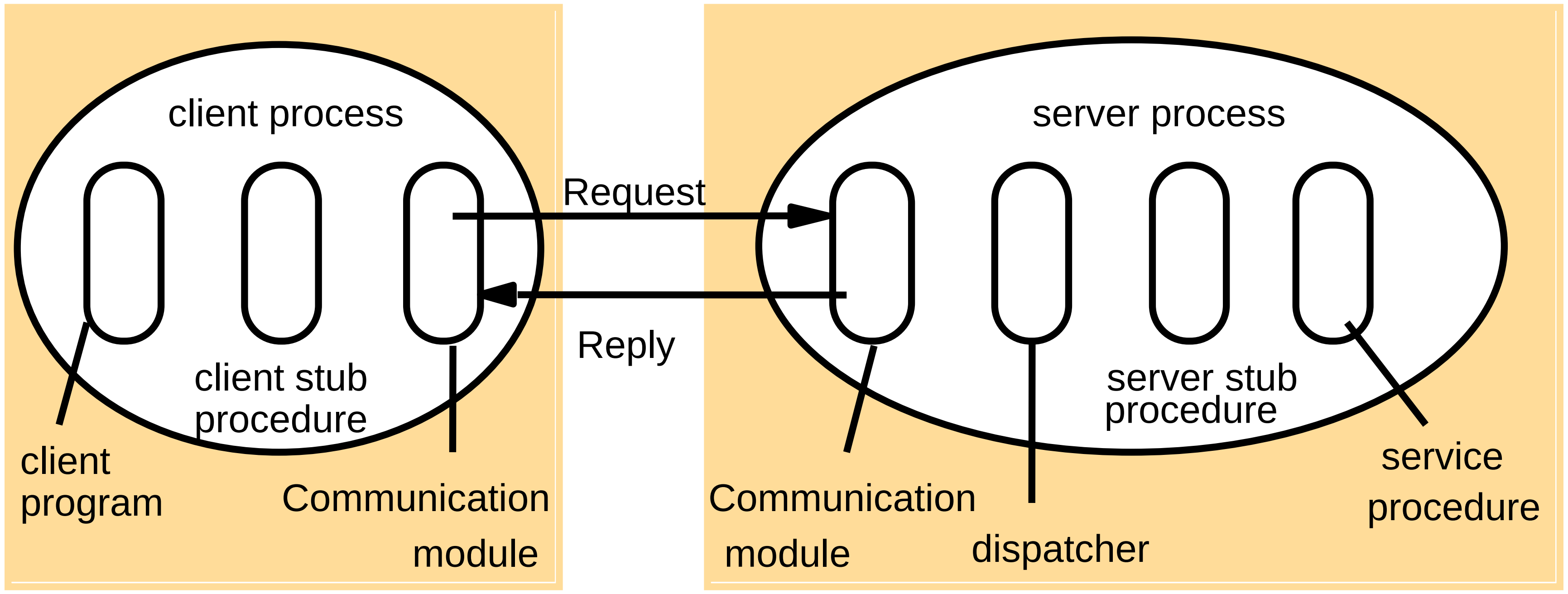
```
// file:  RMIClient.java

import java.rmi.Naming;

public class HelloClient {
    public static void main(String[] args) {
        try {
            Hello obj = (Hello) Naming.lookup("rmi://localhost/HelloServer");
            String message = obj.sayHello();
            System.out.println("Message from server: " + message);
        } catch (Exception e) {
            System.err.println("HelloClient exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Running the Java RMI Example

- **Running the Server and Client**
  - Compile Client and Server classes
  - Develop a security policy file (e.g., HelloPolicy)
    - grant { permission java.security.AllPermission "", ""; };
  - VM options for server: -Djava.rmi.server.hostname=localhost
  - VM options for client: -Djava.security.policy=security.policy
  - Run Server
  - Run Client

Remote Procedure Call (RPC) – used in C

- RPCs enable clients to execute procedures in server processes based on a defined service interface.



THE UNIVERSITY OF MELBOURNE

Remote Procedure Call (RPC)

- **Communication Module**
  - Implements the desired design choices in terms of retransmission of requests, dealing with duplicates and retransmission of results
- **Client Stub Procedure**
  - Behaves like a local procedure to the client. Marshals the procedure identifiers and arguments which is handed to the communication module
  - Unmarshalls the results in the reply
- **Dispatcher**
  - Selects the server stub based on the procedure identifier and forwards the request to the server stub
- **Server stub procedure**
  - Unmarshalls the arguments in the request message and forwards it to the service procedure
  - Marshalls the arguments in the result message and returns it to the client

THE UNIVERSITY OF MELBOURNE

Summary: RMI Programming

- RMI greatly simplifies creation of distributed applications (e.g., compare RMI code with socket-based apps)
- Server Side
    - Define interface that extend java.rmi.Remote
    - Servant class both implements the interface and extends java.rmi.server.UnicastRemoteObject
    - Register the remote object into RMI registry
    - Ensure both rmiregistry and the server is running
- Client Side
    - No restriction on client implementation, both thin and rich client can be used. (Console, Swing, or Web client such as servlet and JSP)