Remote Procedure Call (RPC)

Remote Procedure Call (RPC)

- RPC is a high-level model for client-server communication.
- It is the first successful distributed technology used in DS

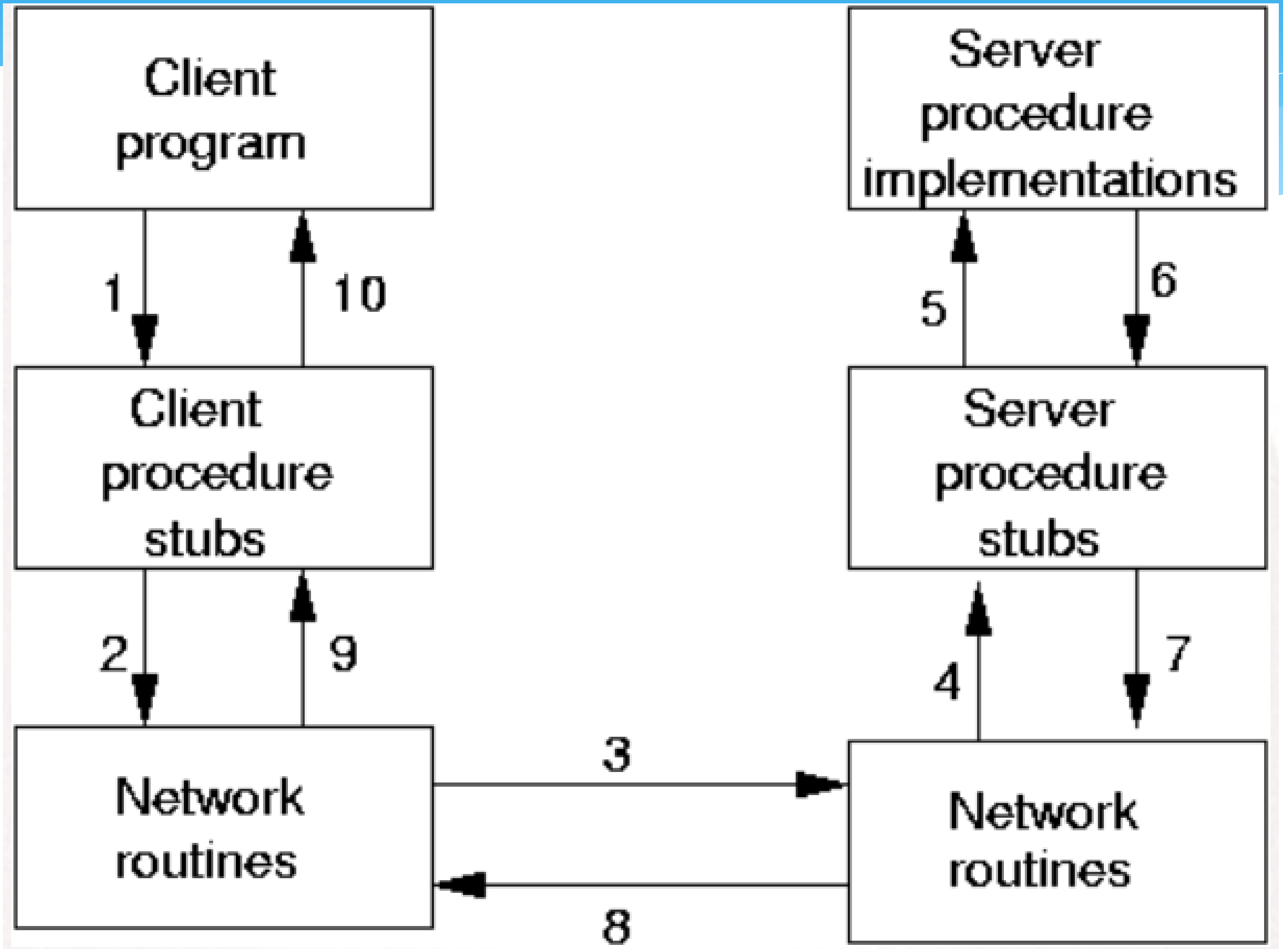
Importance of RPC

- The client needs an easy way to call the procedures of the server to get some services.
- Enables clients to communicate with servers by calling procedures in a similar way to the conventional use of procedure calls in high-level programming.
- Modeled on the local procedure call, but the called procedure is executed in a different process and usually a different computer.

Disadvantages of RPC

- RPC is not, by design, object oriented
- RPC supports a limited set of data types, therefore it is not suitable for passing and returning Java Objects
- RPC requires the programmer to learn a special interface definition language (IDL) to describe the functions that can be invoked remotely

RPC


Client program
Client procedure stubs
Network routines
Server procedure implementations
Server procedure stubs
Network routines
1
2
3
4
5
6
7
8
9
10

Java RMI

Client-Server Programming Review

- What usually happens in a CS application (e.g. recording data)?

SERVER	CLIENT
-Listens	-Connect to Server
-Accepts connection	-Collect data
-Receive data from client	-Send data to server
-Responds to client	-Receives reply

Distributed Systems

"A collection of independent computers that appear to its users as one computer." – Andrew Tanenbaum

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." – Leslie Lamport

Distributed Systems

- Examples:
  - Web search
  - Massively multiplayer online games
  - Financial trading
    - Real time access and processing of a wide range of information resources.

Importance of Distributed Systems

- Resource sharing
  - Resource managers
  - Resource sharing model
    - Client-server resource model
    - Object-based resource model
      - Message passing

Importance of Distributed Systems

- Scalability
  - Scaling techniques
    - Distribution
    - Replication
    - Hiding communication latencies

Importance of Distributed Systems

- Fault tolerance
  - Techniques for handing failures
    - Detecting failures
    - Masking failures
    - Tolerating failures
    - Recovery from failures
    - Redundancy
- Allow heterogeneity

DS Transparency

- Access
- Location
- Migration
- Relocation
- Replication
- Concurrency
- Failure

Pitfalls DS Development

- False assumptions made by first time developer:
  - Network is reliable
  - Network is secure
  - Network is homogenous
  - Topology does not change
  - Latency is zero
  - Bandwidth is infinite
  - Transport cost is zero
  - One administrator

Quality of Service (QoS)

- Non-functional properties of the system:
  - Reliability
  - Security
  - Performance
- Adaptability

Roles of MW in DS

- Layer of software offering a single-system view
- Offers transparencies
- Simplifies development of distributed applications and services

Types of DS

- Distributed computing systems
  - Cluster and cloud computing systems
  - Grid computing systems
- Distributed information systems
  - Transaction processing systems
  - EAI

Types of DS

- Distributed pervasive (or ubiquitous) systems
  - Mobile and embedded systems
  - Home systems
  - Sensor networks

Remote Method Invocation

JAVA RMI

RMI

- Is an API that provides a mechanism to create distributed application in Java.
- Allows an object to invoke methods on an object running in another JVM
- Provides communication between the applications using two objects **stub** and **skeleton**

TWO OBJECTS

- Stub
  - An object, acts as a gateway for the client side.
- Skeleton
  - An object, acts as a gateway for the server side.

Participating processes

- Client
- Server
- Object Registry
  - Name server that associates objects with names
  - Objects are registered
  - URL namespace
    - rmi://hostname:port/pathname

Requirements for distributed applications

- If any application performs these tasks, it can be distributed application.
  - The application need to locate the remote method
  - It needs to provide the communication with the remote objects, and
  - The application needs to load the class definitions for the objects

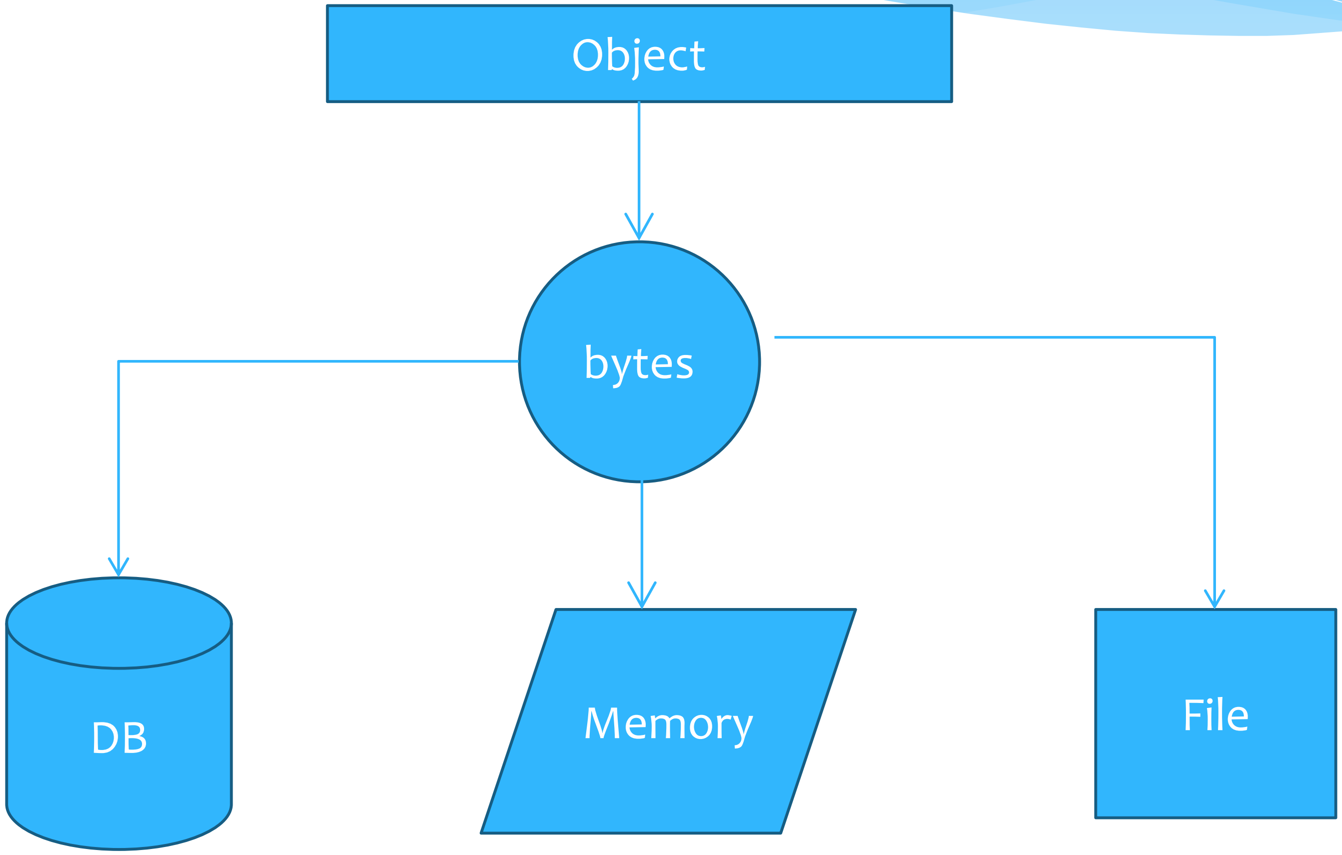
Steps to write RMI

- Create the remote interface
- Provide implementation for the remote interface
- Compile the implementation class and create the stub and skeleton objects using the rmic tool
- Start the registry service by rmiregistry tool
- Create and start the remote application
- Create and start the client application

Serialization

Java RMI

Serialization



Why serialize?

- Store data
- Transmit data
- Clone an object without overriding clone

Serialize object -> File

* To store a serialized object in a file:
  * Create an ObjectOutputStream with a FileOutputStream

FileOutputStream – writes binary data/sequence of bytes into a file
ObjectOutputStream – converts a java object to sequence of bytes and writes it into an outputstream

Deserialize Object <- File

- To re-create an object from a file:
  - Create an ObjectInputStream with a FileInputStream

FileInputStream – reads binary data/ sequence of bytes from a file
ObjectInputStream – reads a sequence of bytes from an inputstream and reconstructs the java object

Interface Serializable

- Required, if a class wants to have its state serialized/deserialized
- No methods, no fields
- If a class implements Serializable every instance field has to be serializable or declared transient