

Introduction & RMI Basics

CS3524 Distributed Systems

Lecture 01

Distributed Information Systems

- Distributed System:
 - A collection of autonomous computers linked by a network, with software to produce an integrated computing infrastructure
- Information:
 - Processed data – data that is organised, meaningful, and useful.
- Security:
 - How to prevent unauthorised access to information and services, and how to maintain availability of information and services to authorised users

Learning Outcomes and Assessment

- Learning outcome
 - Basic concepts of distributed object systems
 - Distributed database systems
 - Concurrency control
 - Security issues and secure programming
- Assessment
 - A 2.5hr examination (75%) (2 questions from 3, each 25 marks)
 - An in-course assessment on programming distributed object systems (25%), and

Reading

- Recommended reading:
 - G. Coulouris, J. Dollimore & T. Kindberg. *Distributed systems: Concepts and design*, Addison-Wesley, 4th Edition, 2005.
 - J. Farley, W. Crawford & D. Flanagan. *Java enterprise in a nutshell*, O'Reilly, 3rd Edition, 2005. Available free on-line through Safari (see information page).
- Many more books on the subject in the library including:
 - D. Ince. *Developing Distributed and E-Commerce Applications*, Addison-Wesley, 2nd Edition, 2004.
 - A. S. Tanenbaum & M. van Steen. *Distributed Systems: Principles and Paradigms*, 2nd Edition, 2007.

Distributed Systems

Software systems that consist of components that may reside on different networked hardware

- We will discuss in detail the **Client – Server Paradigm**
 - A server is a software application that provides access to services for clients
 - A client accesses a particular service provided by a server

Distributed Systems

- We will discuss technology for implementing client-server systems

- Interaction / access to services

- Remote Objects, Remote Method Invocation
- Socket Communication
- JDBC database access

- Concurrency of activities

- Processes, Threads and Scheduling
- Serialisation
- Transactions and Commit Protocols

- Secure Programming

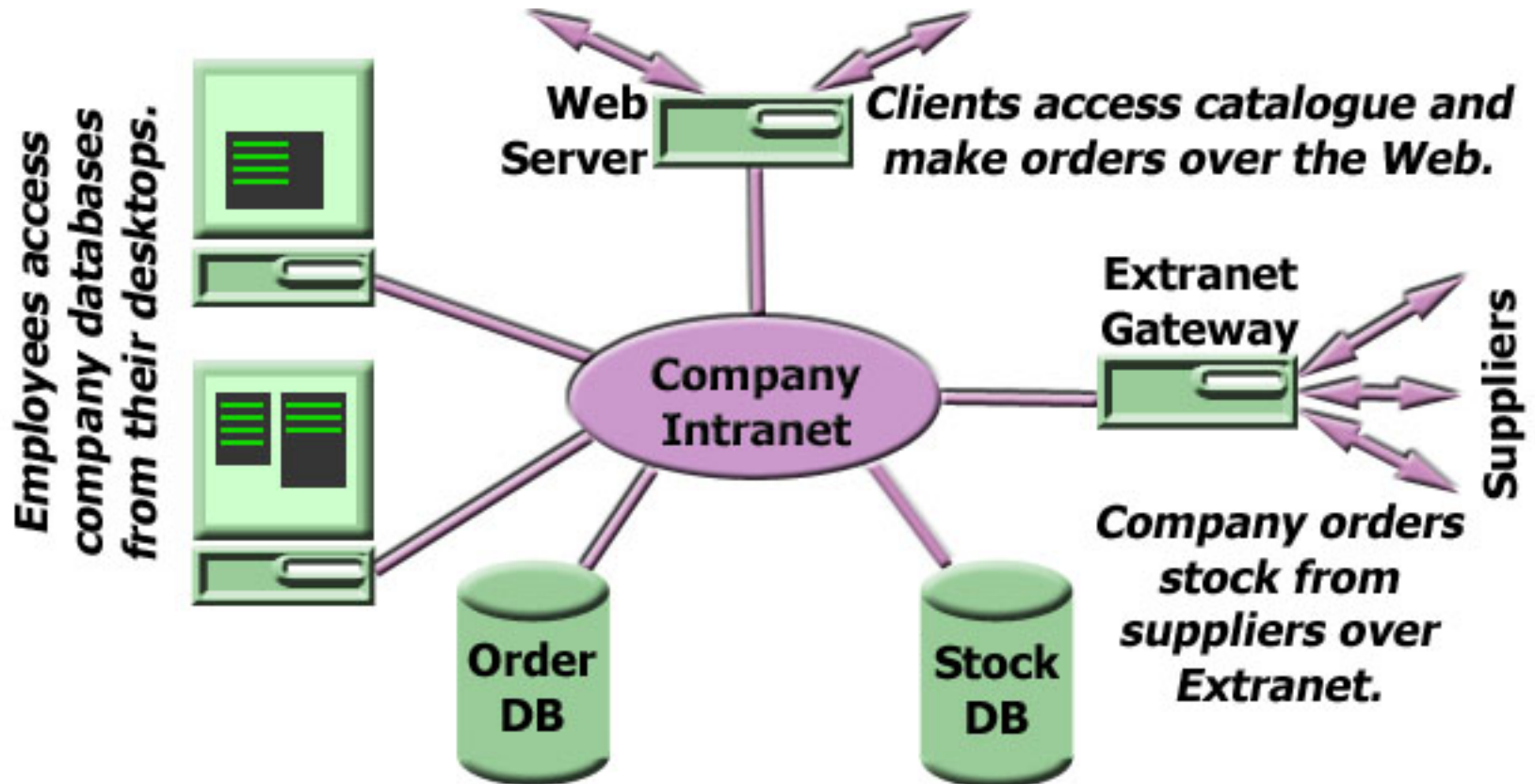
Distributed Systems

- We will discuss Security Issues in Distributed Systems
 - Threat analysis: how to counteract attacks of online systems
 - Network security: attack analysis and defence
 - Cryptology
 - Digital Signature
 - Authentication and Access Control

The Use of Java

- Implementing distributed systems in Java
 - Easy-to-use **socket** interface for stream & datagram sockets
 - Object-based interaction using **Java RMI**
 - Database access via java database connectivity (**JDBC**)
- We will see some of these technologies

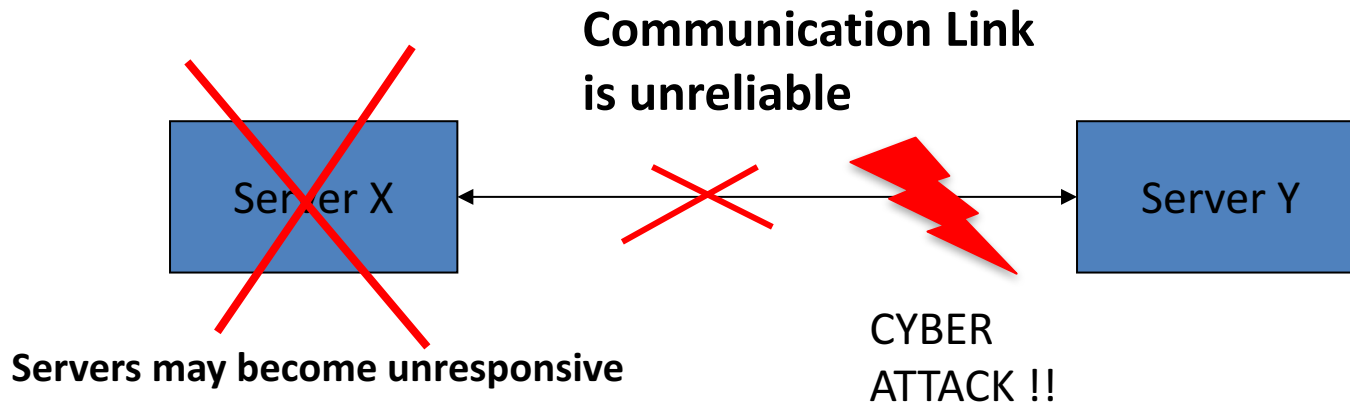
Big Picture: The Online Business



Scenario: The Online Business

- Infrastructure for Online Business
 - Provides access to databases, e.g. Orders, provides information about stocked goods etc.
 - Dedicated applications for staff to access and manipulate this information (internal)
 - Web servers to allow clients / customers to order products, browse product catalogues etc
 - Integration with external systems of business partners, suppliers etc.
- Activities within this distributed system
 - Internal: communication via the company intranet, access to database resources
 - External: access to resources via web servers, via gateways
- Security Issues:
 - Internal/external: different access rights to data
 - Guarantee confidentiality for customer data
 - Save-guard sensitive company data against unauthorised access

Basic Problems



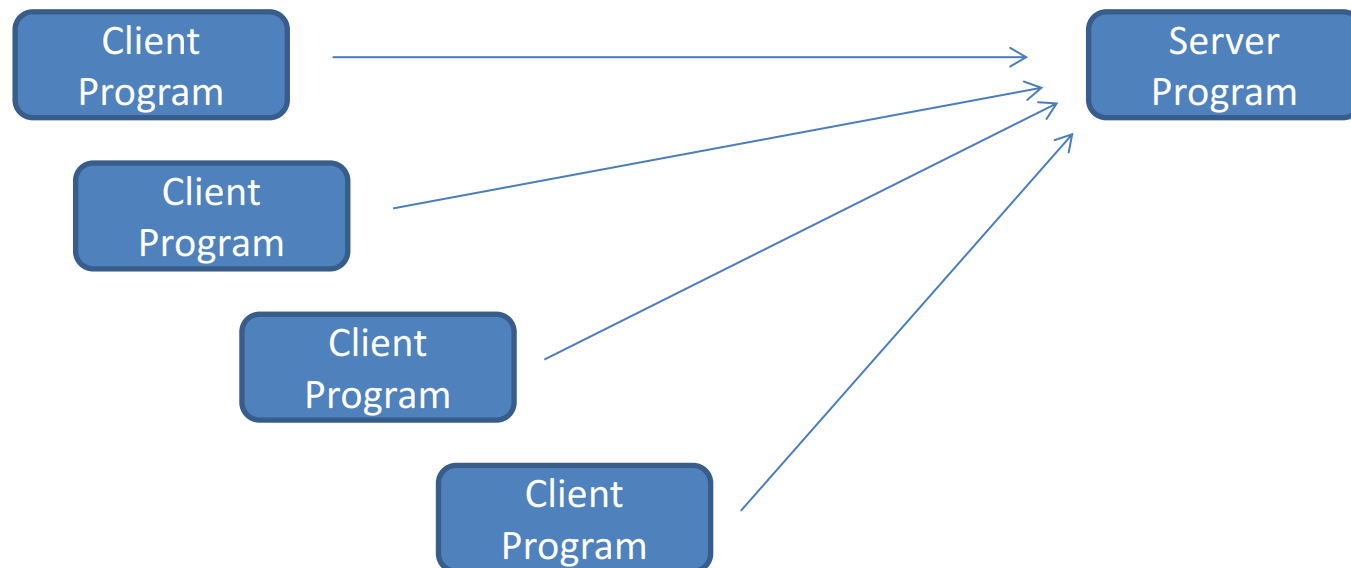
- How can components of a distributed system – “clients” and “servers” – interact reliably?
- How can we guarantee reliable communication?
- How can we guarantee that no data is lost?
- How can we guarantee that the system is save?

Client – Server Paradigm

Programming Distributed Systems

Client – Server Interaction

- Client applications access services on a server application
- How to implement such a distributed application:
 - How to establish a communication between a client program and a server program?
 - Identification: how can a client “find” a particular server?
 - Data exchange: how can data be exchanged between client and server?

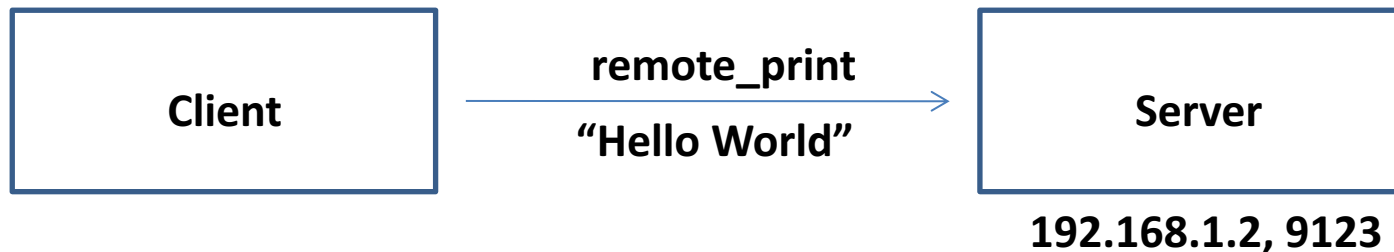


Programming Distributed Systems

- We want a programmer to use a very simple way to use services on a server – network concepts and distribution should be hidden
 - Simple method invocation

```
remote_print ( server, "Hello World" ) ;
```

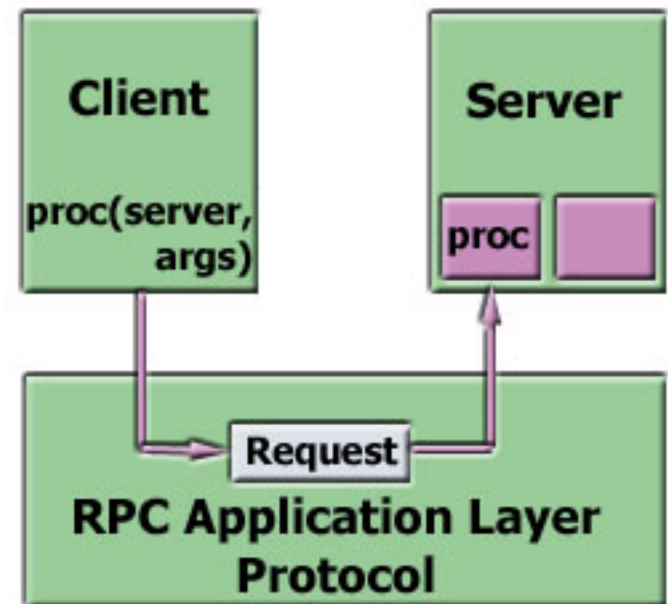

Server handle



History: Client – Server Connection

Remote Procedure Calls (RPCs)

- In the 1980s and early 1990s, the procedural programming paradigm was dominant
 - Client – Server systems were built using Remote Procedure Calls (RPCs)
 - Such a procedure call on a client takes a reference to a server as an extra argument – a so-called *server handle*



Remote Procedure Calls

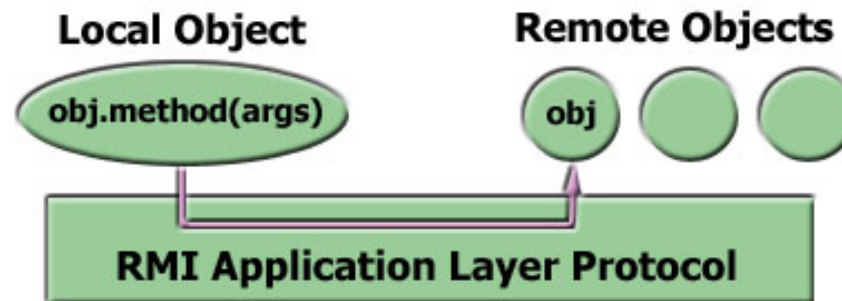
- Remote Procedure Calls are programming means that hide aspects of identifying and finding remote procedures and the data exchange between client and server:
- Identification
 - IP (Internet Protocol) address, e.g. 192.168.1.2, and ports (the connection endpoint on a computer at which a server is listening for client requests)
 - A server handle hides the server's IP address and port
- Lookup
- Data exchange

Remote Procedure Calls

- Lookup
 - The server address and port are obtained from a well-known RPC registry server
 - The client asks the registry for the RPC service by name – this is a naming service
- Data exchange – “marshalling” and “unmarshalling” arguments:
 - The process of *serialization* of arguments:
 - Argument data has to be transformed into a platform-independent format before it can be sent from a client to a server (or server to client) – this is called marshalling / un-marshalling (see serialization in Java)
 - Network communication via the Remote Procedure Call protocol

Remote Method Invocation (RMI)

- In the mid to late 1990s, with the rise of the object-based paradigm in software development, RPCs are now replaced by RMIs:



- Remote Method Invocation – the principles are the same as with RPCs:
 - A client has to obtain a reference to a remote object from a known registry
 - Calling a method on the remote object reference hides the marshalling of data and communication over the RMI protocol
 - Goal: hide aspects of distribution from a programmer, the programmer should not care whether a local or remote method is called

Advantages of RMI over RPC

- Apart from simply being a more modern approach, with the usual benefits of object-orientation, such as: encapsulation, inheritance, polymorphism, etc., RMI offers:
- **No servers, only objects**
 - Local objects, from where methods on other (remote) objects are called, don't care where these remote objects reside
- **Greater transparency**
 - Objects can be moved to other locations of a network, replicated, etc.
 - Local and remote calls can be almost indistinguishable

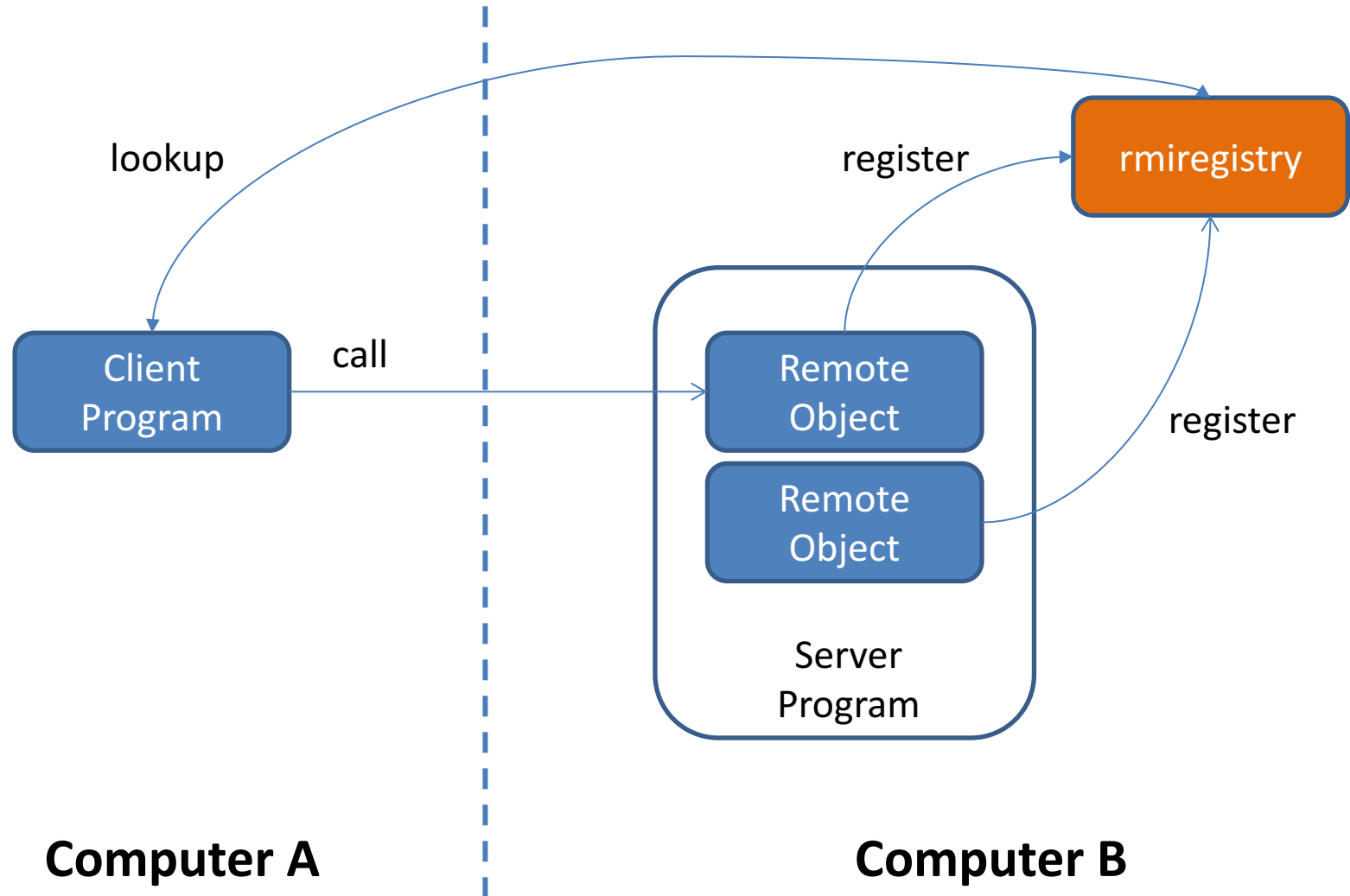
Advantages of RMI over RPC

- **Finer service granularity**
 - “Fat” monolithic server are broken up into more maintainable, “slim-line” service-providing objects
- **Highly dynamic, open environment**
 - Remote object methods can return references to other remote objects as data values
 - object copies can be shipped as data.
 - Specific objects can terminate when not required, and, due to serialization, reinitiate when next required.
- Note: Specific services of pre-OO legacy systems can be “wrapped” as objects and made available on the network

Java RMI

How to program with RMI in Java

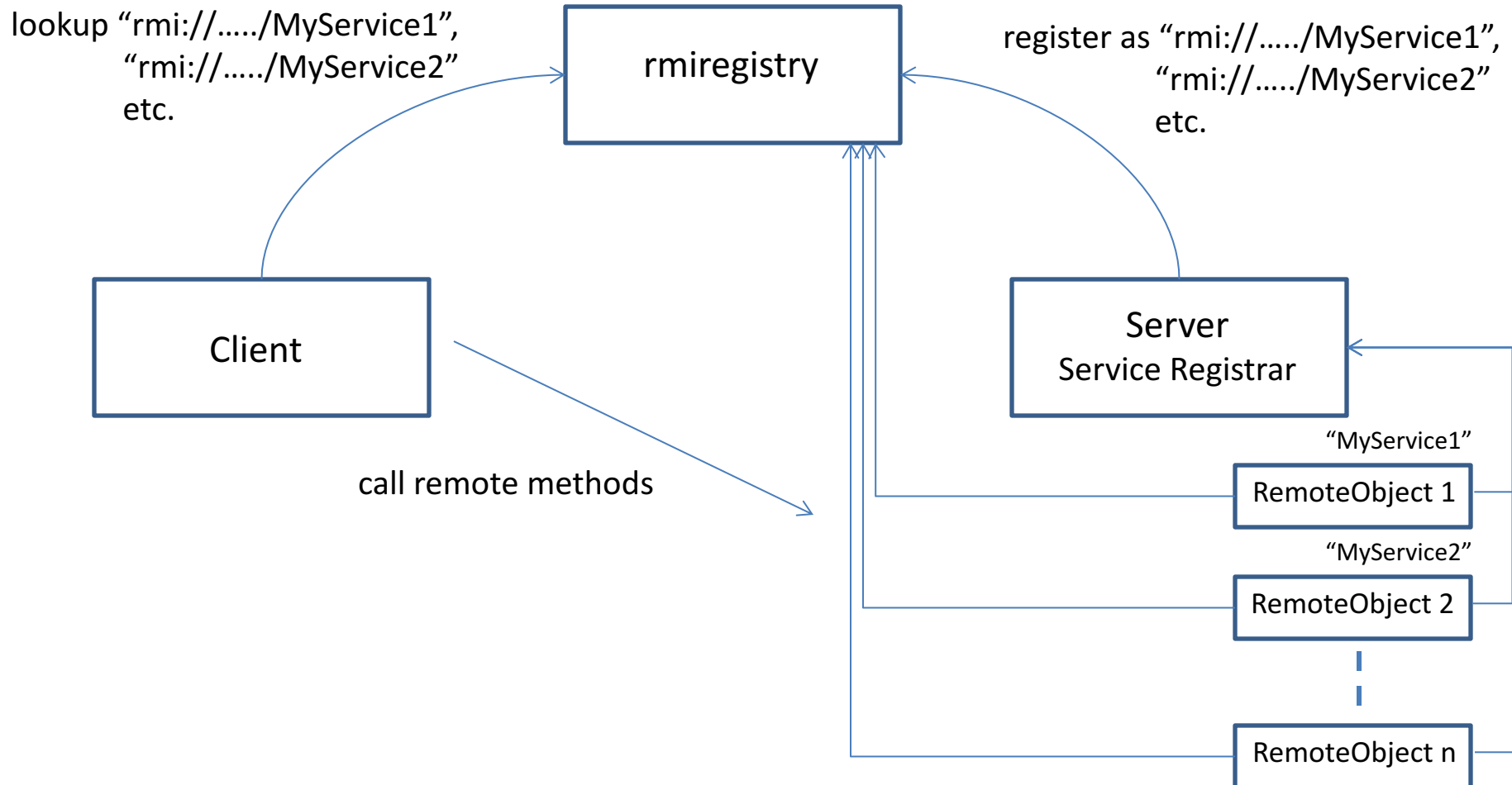
Java Remote Method Invocation (RMI)



Java RMI

- Registration (“binding” of remote objects)
 - A set of remote objects has to be registered with a *naming service* – in case of Java RMI, this is the **rmiregistry**
- Obtaining a remote object reference
 - A client can use the naming service to lookup a URL to obtain a remote object reference
 - The application can pass and return remote object references as part of its operation
- Communication with remote objects
 - RMI enables method invocation between objects that are not situated in the same **Java Virtual Machine**
 - To the programmer, this looks like **standard Java method invocation**
 - The client actually has a **local proxy** to the server, which is generated during the compilation process

Java RMI



Using RMI

- Design and implement the remote services of the distributed application:
 - Define the remote services as `rmi.Remote` interfaces
 - Implement the remote services as Java objects
 - Write a server starter program to deploy the application
 - This will construct and register the remote objects
 - Write a client for the application
 - The client will lookup and invoke the services of the remote objects.

Defining the Remote Service

- To illustrate the basics of RMI, we will use the simple “shout” example (download the rmishout example from the practicals web page):
 - To define the remote service, we write a remote interface
 - The interface must extend `java.rmi.Remote`
 - All methods on the interface may throw a `java.rmi.RemoteException`
 - Such exceptions are thrown during a remote method call if there is a communication failure or protocol error

An Example Remote Interface

```
package cs3517.examples.rmishout;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ShoutServerInterface
    extends Remote
{
    public String shout( String s )
        throws RemoteException;
}
```

- This is an interface definition for a remote object (service), called “ShoutServer”
- It provides the method “shout” to clients, which has to be implemented by the actual remote object

Remote Object as an Implementation of a Remote Interface

- To do:
 - Create the remote object that implements one or more remote interface(s)
 - Define a constructor for the remote object
 - Provide an implementation for each method declared by the remote interface(s)
 - We introduce the following convention:
 - The name of a class implementing an interface consists of a part of the name of the interface + an extra “**Impl**” at the end


An Example Implementation

```
package cs3517.examples.rmishout;

import java.rmi.server.UnicastRemoteObject;

public class ShoutServerImpl
    extends UnicastRemoteObject;
    implements ShoutServerInterface
{
    public ShoutServerImpl()    throws RemoteException
    { }

    public String shout( String s )
        throws RemoteException
    {
        return s.toUpperCase();
    }
}
```



The Server

- Implement a “registrar” for your remote objects:
 - We call it the “Server Mainline”
 - Creates and installs a security manager
 - Instantiates remote objects and export them
 - Registers (“exports”) these remote objects using the “Naming” service (RMI registry) – this is called “binding”
- Remote object registration with the RMI registry
 - Each remote object is given a unique URL, e.g.:
 - `rmi://hawk.csd.abdn.ac.uk:12345/MyService`
 - If the port is omitted, it defaults to 1099 (port where rmiregistry is listening)
 - Note that, for security reasons, an application can bind, unbind or rebind remote object references only with a registry running on the same host

Key Server Mainline Code

```
package cs3517.examples.rmishout;

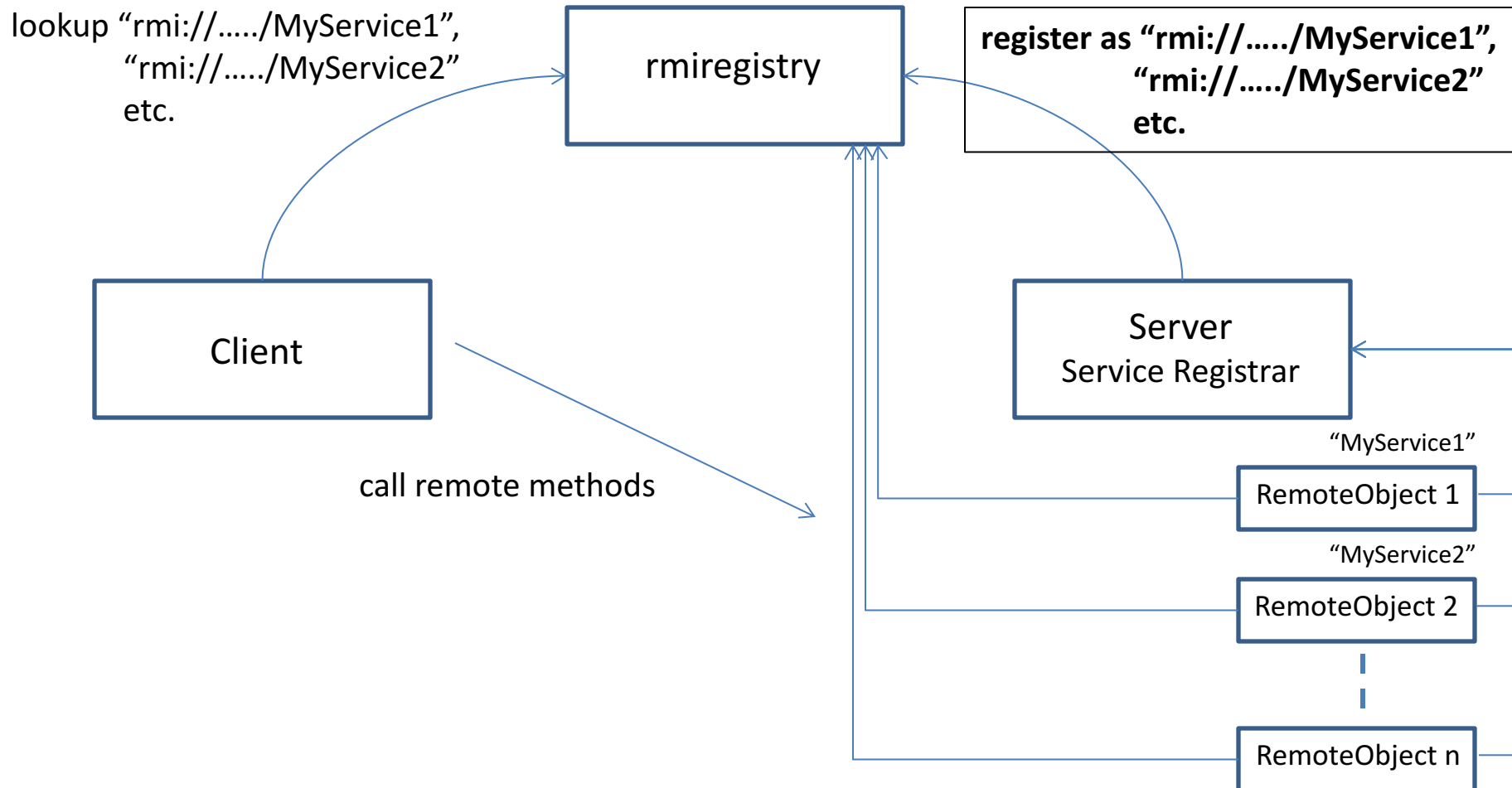
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
public class ShoutServerMainline
{
    public static void main(String args[])
    {
        try {
            if (System.getSecurityManager() == null)
                System.setSecurityManager(new SecurityManager());

            ShoutServerImpl service = new ShoutServerImpl();
            String url = "rmi://" + host + ":" + port + "/Shout";
            Naming.rebind(url, service);
        } // catch & handle exceptions.
    } // NB: The server mainline will not shut down; why?
}
```

Implementing a Client

- In common with the server, the client must create and install a security manager
- Look up the remote object with the Naming service by its URL
 - This gives the programmer a reference to the remote object (a stub is generated within the client VM)
- Finally invoke methods on that remote object

Binding Remote Objects



Key Client Code

```
package cs3515.examples.rmishout;

import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class ShoutClient
{
    public static void main(String args[])
    {
        try {
            if (System.getSecurityManager() == null)
                System.setSecurityManager(new SecurityManager());
            String url = "rmi://" + host + ":" + port + "/Shout";
            ShoutServerInterface service =
                (ShoutServerInterface) Naming.lookup(url);
            System.out.println(service.shout( "hello" ));
        } // catch and handle exceptions.
    }
}
```

