



Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita
Remote Method Invocation (RMI)

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it



Architetture distribuite

- Solitamente, nell'ambito delle applicazioni informatiche si distingue tra:
 - ▶ **architetture locali**: tutti i componenti sono sulla stessa macchina;
 - ▶ **architetture distribuite**: le applicazioni e i componenti possono risiedere su nodi diversi messi in comunicazione da una rete.
 - Vantaggi:
 - Accesso a dati non disponibili localmente,
 - Uso di programmi concorrenti (parallelismo, distribuzione del carico elaborativo, ...)
 - Svantaggio:
 - maggiore complessità, specialmente riguardo alla comunicazione fra i vari componenti.



Classificazione delle applicazioni distribuite

- Le applicazioni distribuite si possono classificare in base al grado di distribuzione:
 - ▶ Client-server
 - ▶ Multi-livello
 - ▶ Applicazioni completamente distribuite

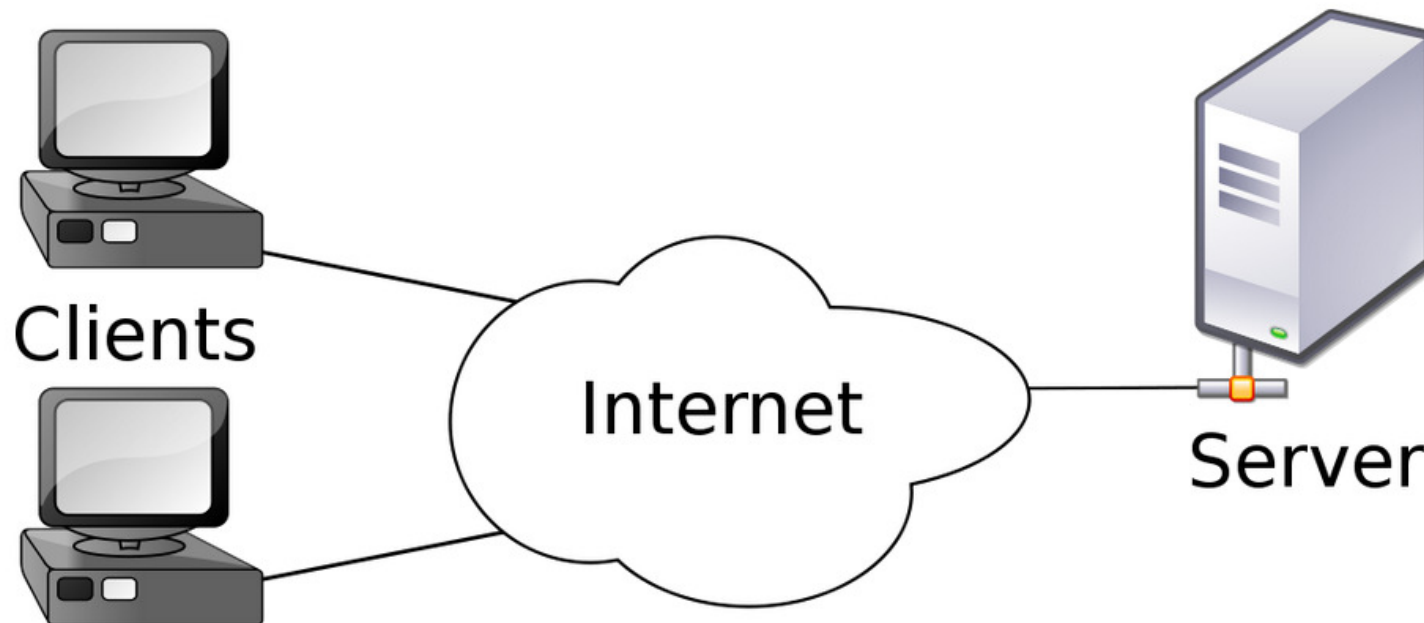


Applicazioni completamente distribuite

- Architettura ad oggetti, con oggetti client che invocano servizi offerti da oggetti server
- Il carico elaborativo è completamente distribuito
- Realizzabile con diverse tecnologie: CORBA, RMI, DCOM, ...

Client-server

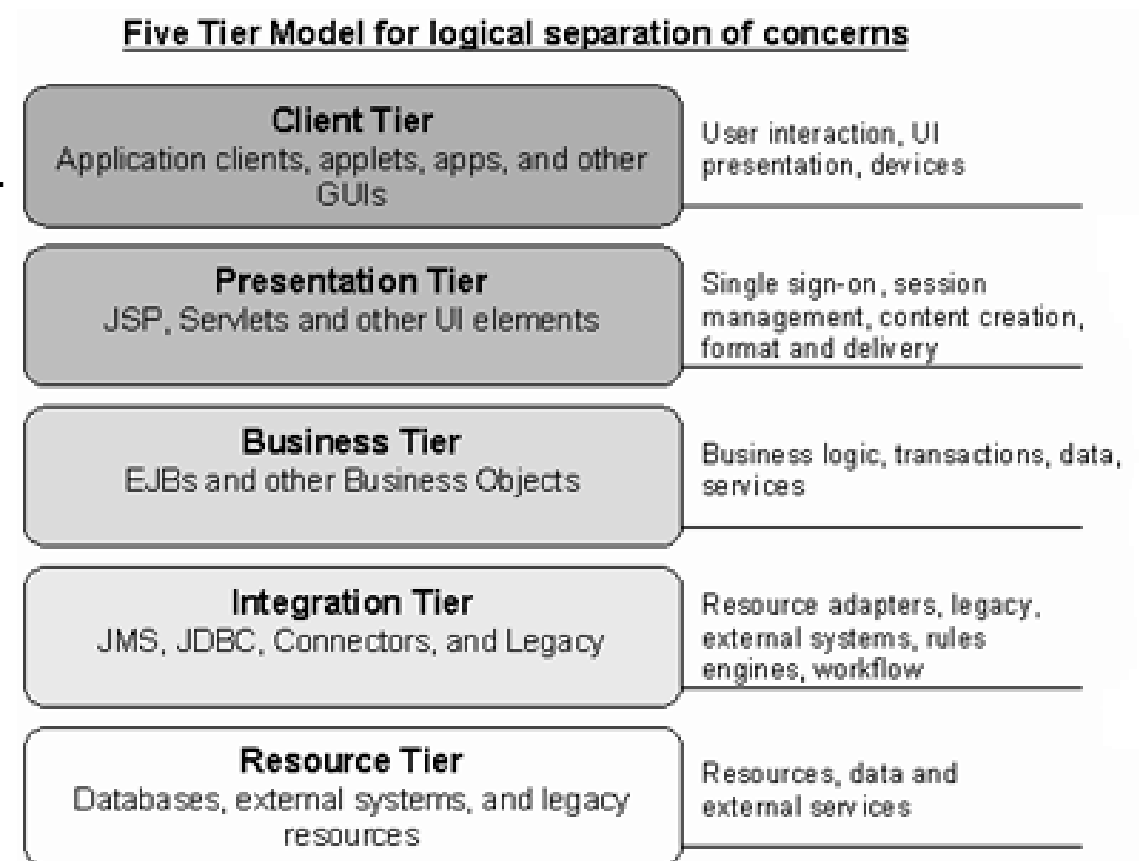
- Sono presenti solo due livelli. Solitamente il server è specializzato.
- Es. Web browser – Web server





Multi-livello (multi-tier)

- Organizzazione simile al C/S, ma con un numero maggiore di livelli.
- Es. modello three-tier:
 1. Interfaccia utente ed elaborazione locale
 2. logica applicativa
 3. gestione dati persistenti.
- Es. modello five-tier

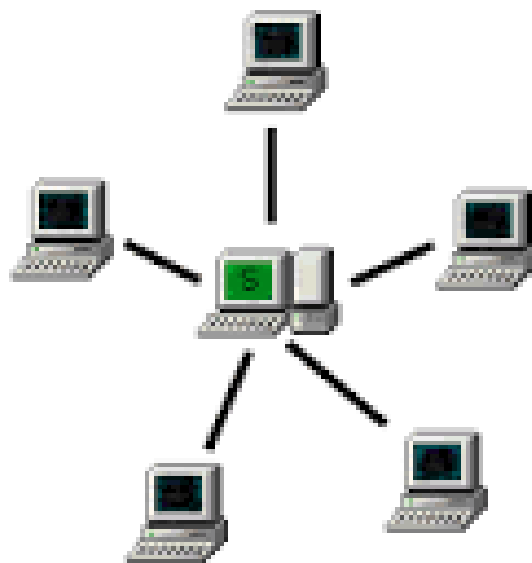




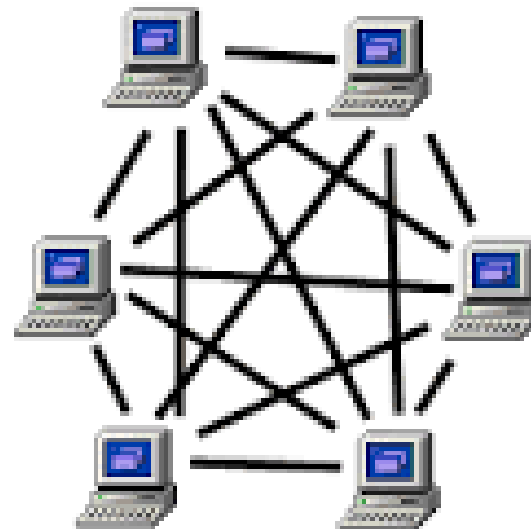
Applicazioni completamente distribuite: Peer-to-peer

- Peer-to-peer: insieme di nodi di pari livello, che possono fungere sia da client che da server verso gli altri nodi della rete
 - ▶ Es. i classici File sharing come Gnutella, Bit Torrent, eDonkey, etc.

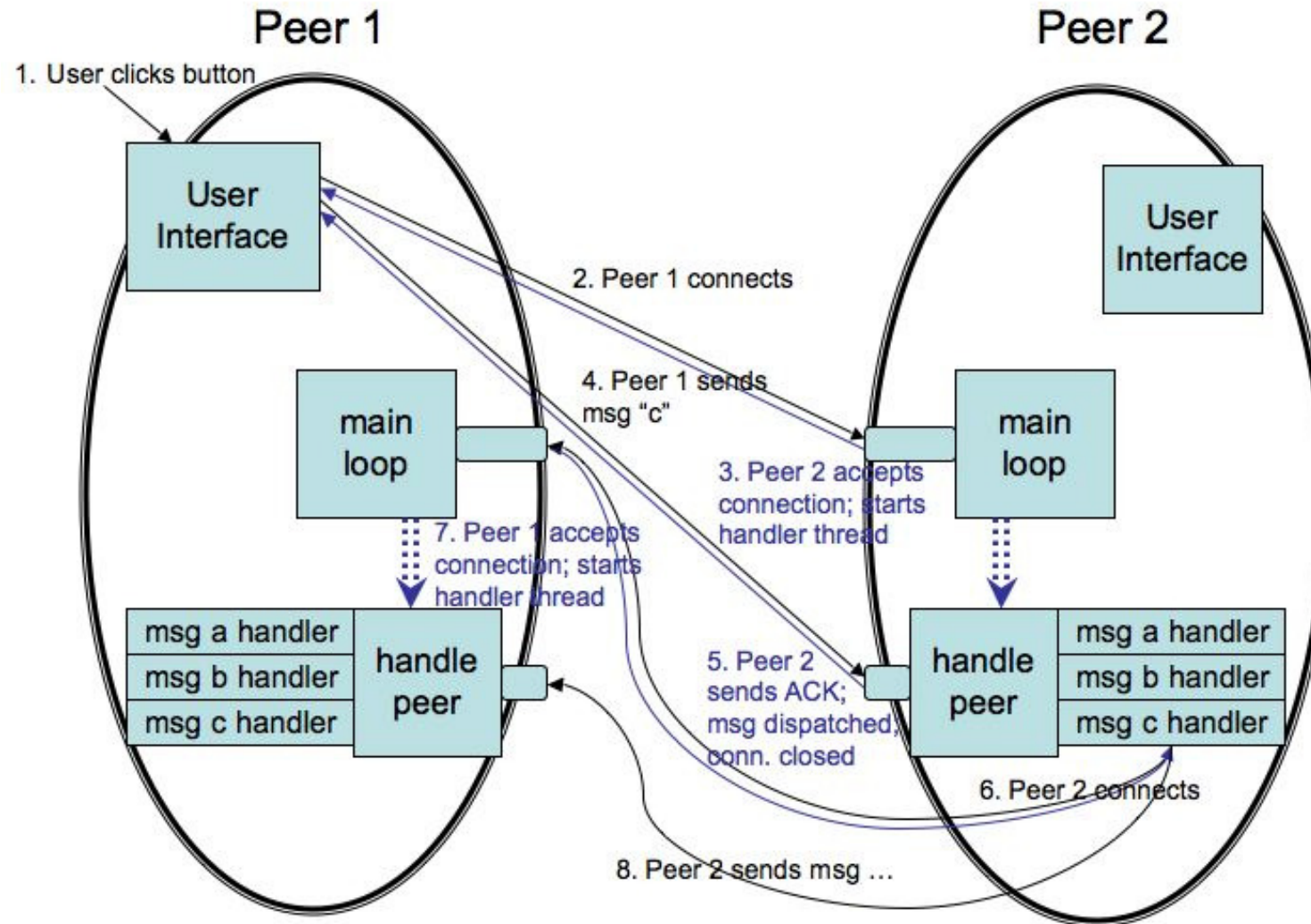
Server Based Network



Peer to Peer Network



Peer-to-peer: funzionamento tra due nodi





Middleware

- Middleware: il software che rende accessibile su Internet risorse hardware o software che prima erano disponibili solo localmente

« (...) software di connessione che consiste in un insieme di servizi e/o di ambienti di sviluppo di applicazioni distribuite che permettono a più entità (processi, oggetti, ecc.), residenti su uno o più elaboratori, di interagire attraverso una rete di interconnessione a dispetto di differenze nei protocolli di comunicazione, architetture dei sistemi locali, sistemi operativi, ecc. »; ovvero trattasi di Comunicazione tra processi (IPC).

- La «colla» che tiene insieme le applicazioni distribuite.



Middleware

- Software che permette ad un'applicazione di interoperare con altro software, senza che l'utente debba capire e codificare le operazioni di basso livello che rendono possibile l'interazione.
- Interazioni sincrone: il sistema richiedente attende la risposta
- Interazioni asincrone: il sistema richiedente non attende la risposta: quest'ultima sarà accettata quando arriverà.



RMI: Remote Method Invocation

- Consente a processi Java distribuiti di comunicare attraverso una rete
- È una tecnologia specifica del mondo Java
- Un processo (client) può fare una chiamata di metodo ad un oggetto remoto (server) **come se** l'oggetto fosse sulla stessa macchina.
`String response = server.sayHello();`



Architettura client-server usando RMI

- Caso tipico
 - ▶ Server crea oggetti remoti, li rende visibili e aspetta che i client invochino metodi su di essi
 - ▶ Client ottiene riferimenti a oggetti remoti e invoca metodi su di essi
- RMI fornisce il meccanismo di comunicazione con cui server e clients comunicano per costituire l'applicazione distribuita
 - ▶ tipicamente un client ottiene servizi dal server invocando metodi di oggetti remoti con una sintassi identica a quella per gli oggetti locali (residenti sulla stessa JVM), ma con una semantica un po' diversa

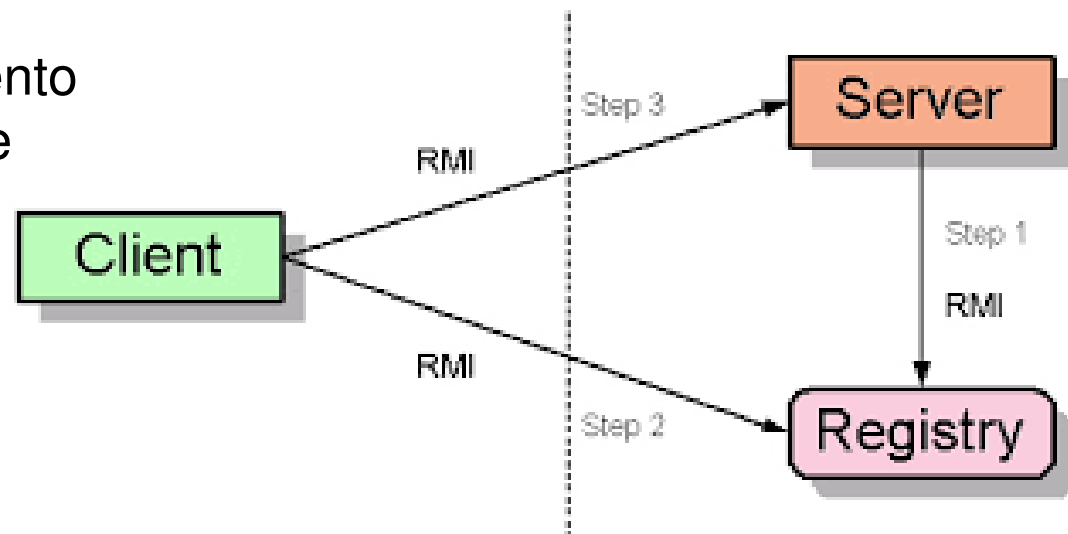


Location transparency

- Con RMI, il client può anche ignorare dove sta l'oggetto remoto (server).
- Il client possiede un riferimento a un oggetto, senza sapere se questo oggetto è locale o remoto.
 - ▶ Se è remoto, ci pensa RMI a gestire le comunicazioni col server.

Registry

- Se il client NON possiede un riferimento a un oggetto in grado di fornirgli il servizio desiderato, lo può cercare usando il servizio di Registry.
 - Il server pubblica il servizio sul Registry.
 - Il client non conosce il server, ma sa che sul Registry sono disponibili informazioni sui server.
Quindi cerca sul Registry e gli viene restituito un riferimento al server
 - Il client usa il riferimento al server per ottenere il servizio (ma può anche passare in giro il riferimento)





Message passing

- Una chiamata a metodo può avere argomenti e l'esecuzione del metodo può restituire un risultato.
- Quando l'oggetto proprietario del metodo è remoto, sia gli argomenti che il risultato sono inviati come messaggi.
- RMI rende la comunicazione dei messaggi trasparente al programmatore.
 - ▶ Cioè non dobbiamo preoccuparci di costruire il messaggio, di come viene spedito e di come viene ricevuto e ricostruito a destinazione.



Passaggio dei parametri

- **server** è un oggetto remoto.
String response = server.sayHello(obj) ;
 - ▶ Se **obj** è un oggetto remoto, viene passato il riferimento all'oggetto remoto.
 - ▶ Se **obj** è un oggetto locale, al metodo remoto viene inviata una copia serializzata (passaggio per valore con deep copy).
 - ▶ I tipi primitivi vengono passati comunque per valore.
- C'è il solito problema che l'oggetto remoto potrebbe non conoscere la classe di **obj**. In questi casi si può passare il codice delle classi necessarie per l'elaborazione remota.



Funzionalità di RMI

- Localizzazione di oggetti remoti
 - ▶ Gli oggetti remoti sono registrati presso un registro di nomi (rmiregistry) che fornisce una "naming facility", oppure
 - ▶ Si passano riferimenti a oggetti remoti
- Comunicazione con oggetti remoti
 - ▶ È gestita da RMI, per i programmi accedere a oggetti remoti non fa differenza rispetto agli oggetti locali.
- Dynamic class loading
 - ▶ essendo possibile passare oggetti ai metodi di oggetti remoti, e poiché gli oggetti (non remoti) vengono passati per valore, RMI oltre a trasmettere i valori dei parametri, consente di trasferire il codice degli oggetti a run-time.



Implementazione di una RMI

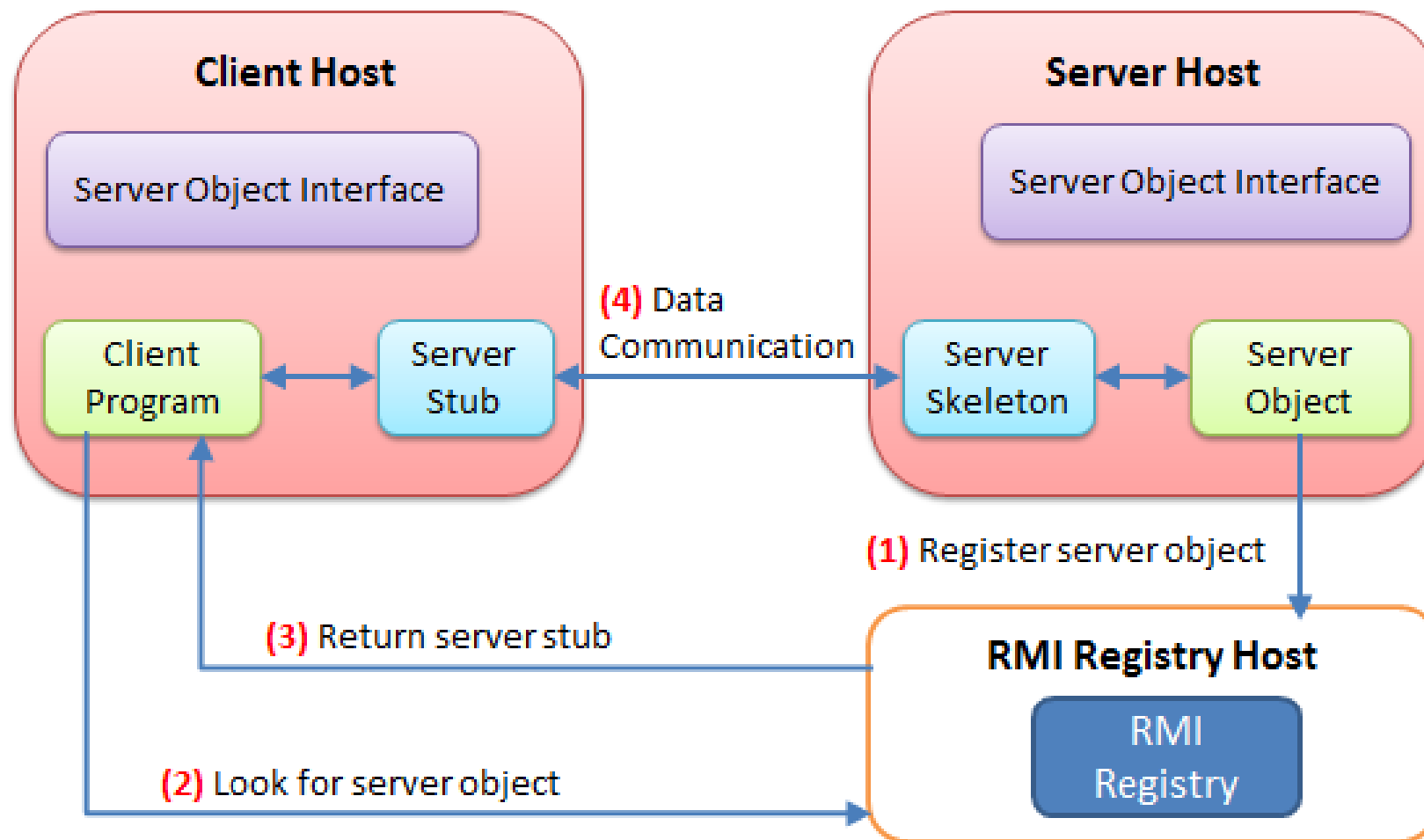
- Per ogni oggetto remoto ci sono:
 - ▶ Un oggetto lato client (chiamato stub)
 - ▶ Un oggetto lato server (chiamato skeleton)
- Un'invocazione di un metodo di un oggetto remoto è gestita localmente usando lo stub, che funge da surrogato locale dell'oggetto remoto.
- L'invocazione risulta nella creazione di un messaggio (contenente il nome del metodo e gli argomenti) ed il suo invio al server.
 - ▶ *parameter marshalling*
- Il messaggio è ricevuto dallo skeleton.
- Lo skeleton ricostruisce i parametri (*unmarshalling*) e chiama il metodo.



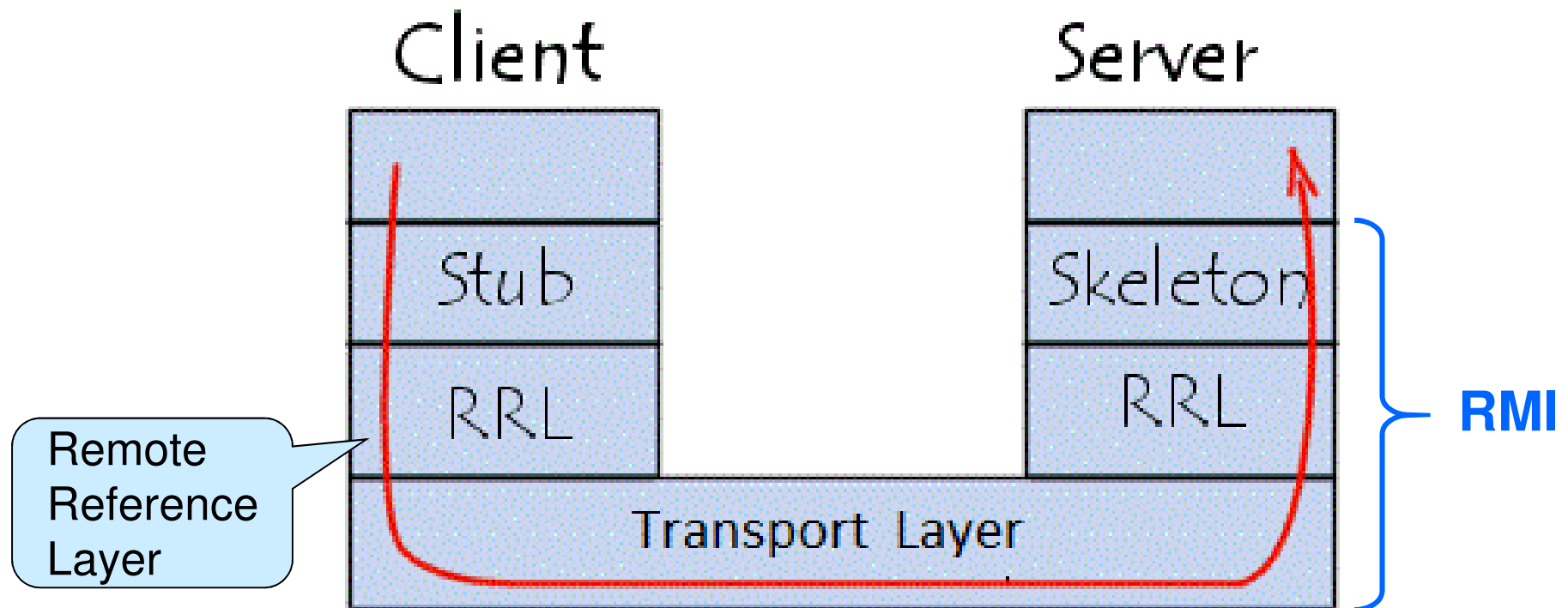
Implementazione di una RMI

- Stub e skeleton sono creati e gestiti dentro agli oggetti remoti direttamente da RMI
 - ▶ Trasparenti ai programmatori
 - ▶ Nelle versioni vecchie di RMI erano prodotti dai compilatori RMI e visibili ai programmatori

RMI: sequenza di operazioni



Layer RMI





Oggetti remoti

- Un server RMI è un oggetto remoto.
- Un oggetto remoto è descritto da un'interfaccia remote, cioè un'interfaccia che estende `java.rmi.Remote`.
- Un oggetto remoto deve implementare tale interfaccia.
- I metodi definiti in tali interfacce devono dichiarare l'eccezione `java.rmi.RemoteException`
- Esempio di interfaccia remote:

```
import java.math.BigInteger;
import java.rmi.RemoteException;
public interface PowerService extends Remote {
    public BigInteger square(int n) throws RemoteException;
    public BigInteger power(int n1, int n2)
                                throws RemoteException;
}
```



Oggetti remoti

- Un oggetto remoto deve fare due cose:
 - ▶ implementare un'interfaccia remota
 - ▶ estendere **UnicastRemoteObject**
- Invocare un metodo di un tale oggetto risulta in una RMI (invocazione di metodo remoto).



Passaggio di parametri

- I casi sono due:
 - ▶ Si passa un oggetto locale o un oggetto di tipo primitivo
 - RMI fa automaticamente marshaling e unmarshaling di parametri, ma – se il tipo dell'oggetto non è primitivo– bisogna che il programmatore lo metta in grado di farlo, fornendogli oggetti [serializzabili](#).
 - Oggetti passati a (o ritornati da) un oggetto remoto devono implementare l'interfaccia **Serializable**.
 - ▶ Si passa un oggetto remoto
 - Che implementa un'interfaccia remota (che estende **Remote**)
 - Che estende **UnicastRemoteObject**



Passaggio di parametri e serializzazione

```
interface MyInterface extends Remote {  
    MyClass f(MyClass x) throws RemoteException;  
}
```

- L'interfaccia remota **MyInterface** dichiara il metodo **f**, che ha un argomento di tipo **MyClass**
- Se l'oggetto di classe **MyClass** da passare è locale, deve essere serializzabile:

```
class MyClass implements Serializable {  
    private static final long serialVersionUID = 1;  
    private int value;  
    public MyClass(int value) {  
        this.value = value;  
    }  
    public int getValue() {  
        return value;  
    }  
}
```



Sviluppo di una semplice applicazione

1. Definizione dell'interfaccia remota
 - ▶ Deve estendere `java.rmi.Remote`
2. Definizione del codice dell'oggetto remoto
 - ▶ deve implementare l'interfaccia remota
 - ▶ deve estendere la classe `java.rmi.server.UnicastRemoteObject` e chiamare uno dei suoi costruttori
 - oppure usare uno dei metodi statici
`UnicastRemoteObject.exportObject()`
3. Definizione del codice del client
 - ▶ deve richiedere al **registry** un riferimento all'oggetto remoto
 - ▶ deve assegnare il riferimento ad una variabile che ha l'interfaccia remota come tipo



Interfaccia `Hello.java`

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface Hello extends Remote {  
    String sayHello() throws RemoteException;  
}
```

- Ci serve per creare un'applicazione client-server dove il client invoca il metodo `sayHello` del server.
 - ▶ NB: per cominciare non passiamo parametri



Server HelloImpl.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject
    implements Hello {
```

La classe `HelloImpl` estende `UnicastRemoteObject` e implementa un'interfaccia remota

```
    public HelloImpl() throws RemoteException {
        super();
    }
    public String sayHello() throws RemoteException {
        return "Hello, world!";
    }
}
```



Server HelloImpl.java

```
public static void main(String args[]) {  
    try {  
        HelloImpl obj = new HelloImpl();  
        Registry registro = LocateRegistry.getRegistry();  
        registro.rebind("Hello", obj);  
        System.err.println("Server ready");  
    } catch (Exception e) {  
        System.err.println("Server exception: " + e.toString());  
        e.printStackTrace();  
    }  
}
```

obj è un riferimento a oggetto remoto

L'esecuzione del main arriva in fondo, ma il processo non termina, perché è uno `UnicastRemoteObject`



Registrazione di un oggetto remoto presso un Java RMI registry

- Per poter chiamare un metodo di un oggetto remoto, il client deve prima procurarsi uno stub dell'oggetto remoto.
- A questo scopo, il client cerca il nome del servizio su un Registry.
- Ovviamente bisogna che il servizio sia stato registrato sul Registry, perché il client lo possa trovare: Java RMI fornisce un API per associare un nome di servizio allo stub di un oggetto remoto su un Registry.
- Una volta che un oggetto remoto è stato registrato sul server, i client possono cercare l'oggetto in base al nome, in modo da ottenere un riferimento all'oggetto remoto, e poterne poi chiamarne i metodi.



Metodi dell'interfaccia Registry

Modifier and Type	Method	Description
void	<code>bind(String name, Remote obj)</code>	Binds a remote reference to the specified name in this registry
<code>String[]</code>	<code>list()</code>	Returns an array of the names bound in this registry
Remote	<code>lookup(String name)</code>	Returns the remote reference bound to the specified name in this registry
void	<code>rebind(String name, Remote obj)</code>	Replaces the binding for the specified name in this registry with the supplied remote reference
void	<code>unbind(String name)</code>	Removes the binding for the specified name in this registry



Classe LocateRegistry

Tutti i metodi sono **public static** e restituiscono un **Registry**

Method	Description
<code>getRegistry();</code>	permette di ottenere un riferimento al registro sull'host corrente che è in attesa sulla porta di default (1099)
<code>getRegistry(int port);</code>	permette di ottenere un riferimento al registro sull'host corrente che è in attesa sulla porta port
<code>getRegistry(String host);</code>	permette di ottenere un riferimento al registro sull'host host che è in attesa sulla porta di default (1099)
<code>getRegistry(String host, int port);</code>	permette di ottenere un riferimento al registro sull'host host che è in attesa sulla porta port
<code>createRegistry(int port);</code>	permette di creare un registro sull'host corrente e sulla porta port



Client: HelloClient.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;

public class HelloClient {
    public static void main(String[] args) {
        try {
            Registry registro = LocateRegistry.getRegistry(1099);
            Hello stub = (Hello) registro.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```



Attivazione del registry

- Su sistemi Unix, Linux, Solaris, etc.:
`rmiregistry &`
- Su Windows:
`start rmiregistry`
- Per default, il registry usa TCP, porta 1099.
 - ▶ Per usare una porta diversa, ad es. la 2001: **`rmiregistry 2001`**



Lanciamo `rmiregistry`

```
gigi@hp-850g2-lavazza: ~  
File Edit View Search Terminal Help  
gigi@hp-850g2-lavazza:~$ rmiregistry &  
[1] 3813  
gigi@hp-850g2-lavazza:~$ ps aux | grep rmi  
gigi      3270  0.0  0.2 728624 32948 ?        Sl   19:58   0:01 gnome-terminal  
gigi      3813  0.0  0.2 5485556 45772 pts/5    Sl   20:08   0:02 rmiregistry  
gigi      5458  0.0  0.0 15964  2204 pts/5    S+   21:22   0:00 grep --color=auto rmi  
gigi@hp-850g2-lavazza:~$
```



Controlliamo cosa fa `rmiregistry`

```
gigi@hp-850g2-lavazza: ~  
File Edit View Search Terminal Help  
gigi@hp-850g2-lavazza:~$  
gigi@hp-850g2-lavazza:~$ sudo netstat -tulpn  
[sudo] password for gigi:  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name  
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN      978/smbd  
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN      978/smbd  
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN      2417/dnsmasq  
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      524/cupsd  
tcp6       0      0 :::445                  :::*                   LISTEN      978/smbd  
tcp6       0      0 :::1099                  :::*                   LISTEN      3813/rmiregistry  
tcp6       0      0 :::139                   :::*                   LISTEN      978/smbd  
tcp6       0      0 :::1:631                 :::*                   LISTEN      524/cupsd  
udp        0      0 0.0.0.0:5353            0.0.0.0:*               516/avahi-daemon: r  
udp        0      0 0.0.0.0:40228           0.0.0.0:*               516/avahi-daemon: r  
udp        0      0 127.0.1.1:53            0.0.0.0:*               2417/dnsmasq  
udp        0      0 0.0.0.0:68              0.0.0.0:*               1992/dhclient
```



Sorgenti

The image displays two overlapping file explorer windows. The top window shows a directory structure with tabs: Didattica, Prog_CD, Code, 2021, Lez12, Hello, and Server. It contains a table of files:

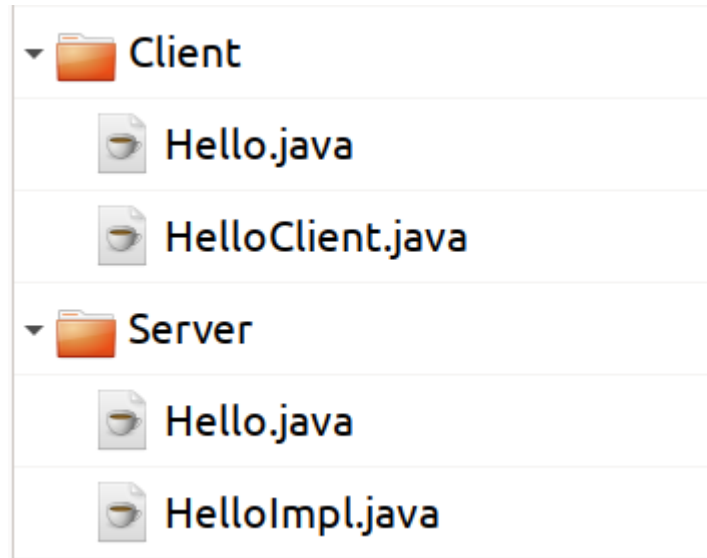
Name	Size	Modified
HelloImpl.java	1,1 kB	19:42
Hello.java	148 bytes	11 mag 2017

The bottom window shows a directory structure with tabs: Documents, Didattica, Prog_CD, Code, 2021, Lez12, Hello, and Client. It contains a table of files:

Name	Size	Modified
Hello.java	148 bytes	11 mag 2017
HelloClient.java	661 bytes	19:41



Sorgenti



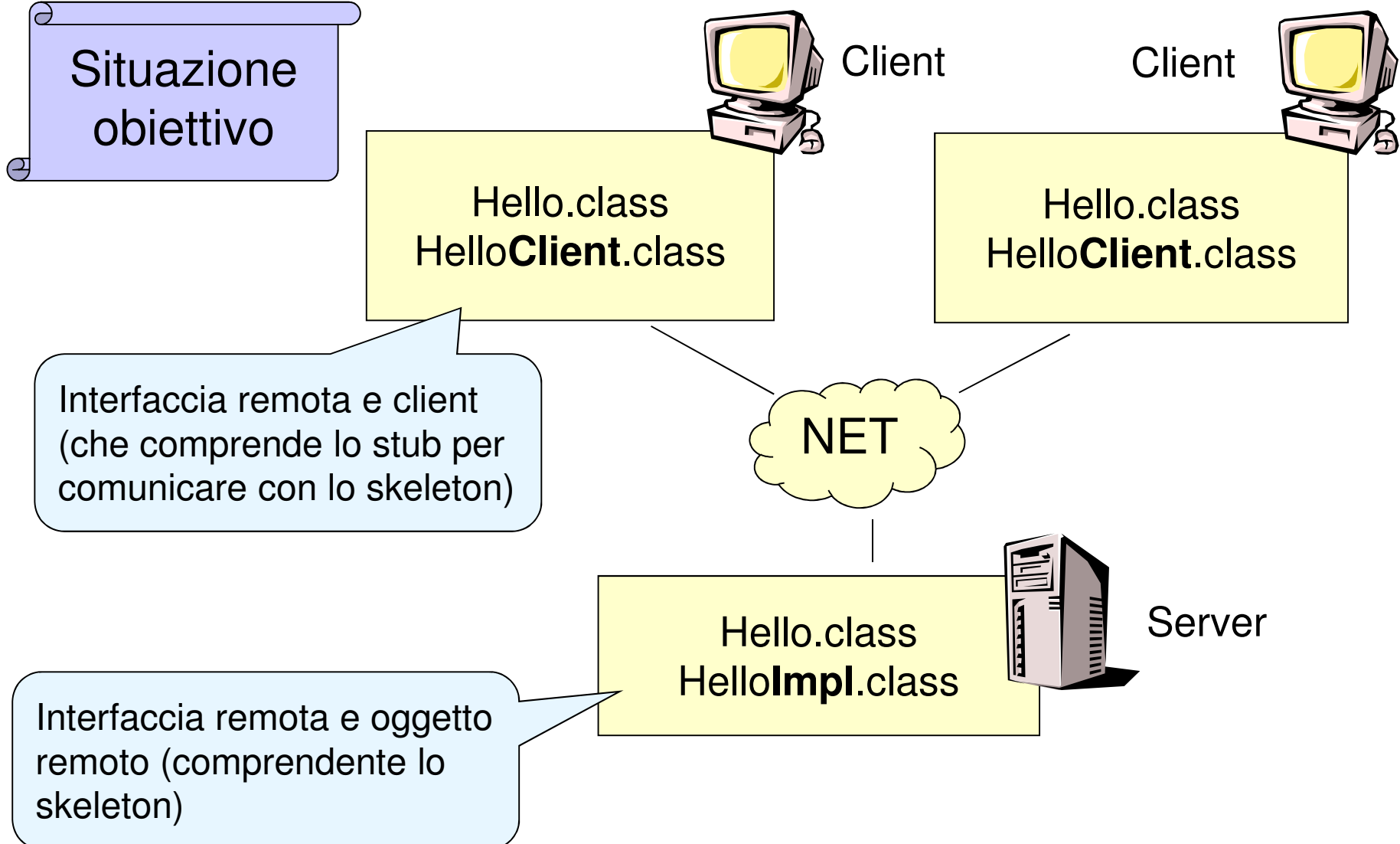


Compilazione

```
cd <path>/Hello/Server  
javac *.java  
cd ../Client  
javac *.java
```

Deploy statico

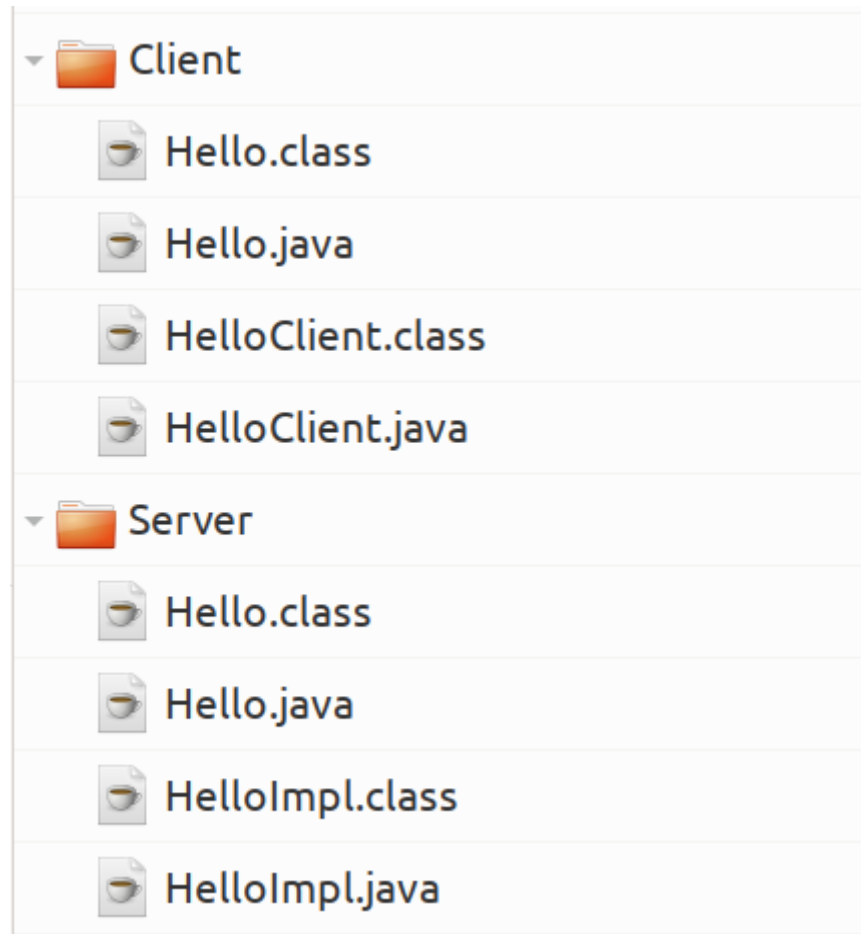
Situazione
obiettivo





Deploy delle classi compilate

- Client e server stanno in directory diverse
- Potrebbero stare su macchine diverse, ma noi testiamo su una macchina sola...





Esecuzione

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/Hello/Server
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/Hello/Server$ rmiregistry &
[1] 13605
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/Hello/Server$ java HelloImpl
Server ready
█

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/Hello/Client
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/Hello/Client$ java HelloClient
response: Hello, world!
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/Hello/Client$ java HelloClient
response: Hello, world!
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/Hello/Client$ █
```



Un esempio un pochino più complesso

- Il metodo remoto ha un argomento di tipo non primitivo
- Che quindi deve essere serializzabile.



Classe Person

```
public class Person implements java.io.Serializable {  
    private String name;  
    public Person(String n) {  
        this.name = n;  
    }  
    public String getName() {  
        return this.name;  
    }  
}
```



Interfaccia `HelloPerson.java`

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface HelloPerson extends Remote {  
    public String sayHello() throws RemoteException;  
    public String sayHello(Person p) throws RemoteException;  
}
```



Server HelloPersonImpl.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloPersonImpl extends UnicastRemoteObject
implements HelloPerson {
    private static final long serialVersionUID = 1L;
    public HelloPersonImpl() throws RemoteException {
        super();
    }
    public String sayHello() throws RemoteException {
        return "Hello, world!";
    }
    public String sayHello(Person p) {
        return "Hello, " + p.getName();
    }
}
```

Il server riceve un oggetto di classe **Person** e accede (localmente) alle sue proprietà.



Server HelloPersonImpl.java

```
public static void main(String args[]) {  
    try {  
        HelloPersonImpl obj = new HelloPersonImpl();  
        Registry registro = LocateRegistry.getRegistry();  
        registro.rebind("HelloPerson", obj);  
        System.err.println("Server ready");  
    } catch (Exception e) {  
        System.err.println("Server exception: " + e.toString());  
        e.printStackTrace();  
    }  
}
```















Client HelloPersonClient.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;

public class HelloClient {
    public static void main(String[] args) {
        try {
            Registry registro = LocateRegistry.getRegistry(1099);
            HelloPerson stub = (HelloPerson)
                registro.lookup("HelloPerson");
            String response = stub.sayHello();
            System.out.println("response: " + response);
            Person someone = new Person("Emerenziano Paronzini");
            response = stub.sayHello(someone);
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```




Deploy delle classi compilate

Name	
▼	Client
	HelloClient.class
	HelloClient.java
	HelloPerson.class
	HelloPerson.java
	Person.class
	Person.java
▼	Server
	HelloPerson.class
	HelloPerson.java
	HelloPersonImpl.class
	HelloPersonImpl.java
	Person.class
	Person.java



Esecuzione

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Server$ pwd
/home/gigi/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Server
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Server$ rmi
registry &
[1] 25839
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Server$ java
a HelloPersonImpl
Server ready
█

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client$ pwd
/home/gigi/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client$ java HelloClient
response: Hello, world!
response: Hello, Emerenziano Paronzi
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client$ █
```



Gestione del registry

- Solitamente il registry sta già girando, e dobbiamo solo localizzarlo e usarlo.
- Per questo va bene il codice visto
- Per fare delle prove, è comodo lanciare il registry insieme al server e terminarlo quando il server termina.
- Si può inserire l'attivazione del `rmiregistry` nel codice del server



Altra piccolo modifica

- Il registry può essere lanciato direttamente dal server.
 - ▶ Non bisogna più lanciare il processo “a mano”

```
HelloPersonImpl obj = new HelloPersonImpl();  
Registry registro = LocateRegistry.createRegistry(1099);  
registro.rebind("HelloPerson", obj);  
System.err.println("Server ready");
```

Lancia un **rmiregistry**
sulla porta 1099



Risultato del lancio del server

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Server$ javac *.java
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Server$ java HelloPersonImpl
Server ready
[]

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client$ pwd
/home/gigi/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client$ java HelloClient
response: Hello, world!
response: Hello, Emerenziano Paronzi
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez12/HelloPerson/Client$ netstat -tulpn
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:5939          0.0.0.0:*                 LISTEN      -
tcp        0      0 127.0.0.53:53          0.0.0.0:*                 LISTEN      -
tcp        0      0 127.0.0.1:631          0.0.0.0:*                 LISTEN      -
tcp        0      0 127.0.0.1:5432         0.0.0.0:*                 LISTEN      -
tcp6       0      0 :::1099                 :::*                    LISTEN      26143/java
tcp6       0      0 :::1/16                 :::*                    LISTEN      2840/kdeconnectd
tcp6       0      0 :::45013                :::*                    LISTEN      26143/java
tcp6       0      0 :::1:631                :::*                    LISTEN      -
```

È il processo server!



RMI & LA SICUREZZA



Premessa

- La trattazione accurata delle questioni relative alla sicurezza non è tra gli obiettivi di questo corso.
- Vedremo solo alcuni meccanismi base forniti da Java.



Permessi

- Le applicazioni viste finora giravano tutte sulla medesima macchina.
- In tale caso Java non fa alcun controllo, a meno che non glielo chiediamo esplicitamente.



Il Java **SecurityManager**

- Attraverso il **SecurityManager** di Java è possibile specificare i permessi di accesso a qualunque risorsa.
- Per creare un security manager:

```
if (System.getSecurityManager() == null) {  
    System.setSecurityManager(new SecurityManager());  
}
```



Permessi di default

- Se lanciamo un server dotato di security manager senza specificare altro, vengono usati i permessi di default.
- In questo modo, al client non è neanche consentito l'accesso al **`rmiregistry`**.



Permessi di default

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD...
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Server$ java HelloImpl
Server ready
[]

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Client
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Client$ java HelloClient localhost
Client exception: java.security.AccessControlException: access denied ("java.net.SocketPermission" "127.0.0.1:1099" "connect,resolve")
java.security.AccessControlException: access denied ("java.net.SocketPermission" "127.0.0.1:1099" "connect,resolve")
    at java.base/java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
    at java.base/java.security.AccessController.checkPermission(AccessController.java:897)
    at java.base/java.lang.SecurityManager.checkPermission(SecurityManager.java:322)
    at java.base/java.lang.SecurityManager.checkConnect(SecurityManager.java:824)
    at java.base/java.net.Socket.connect(Socket.java:604)
    at java.base/java.net.Socket.connect(Socket.java:558)
    at java.base/java.net.Socket.<init>(Socket.java:454)
    at java.base/java.net.Socket.<init>(Socket.java:231)
    at java.rmi/sun.rmi.transport.tcp.TCPDirectSocketFactory.createSocket(TCPDirectSocketFactory.java:40)
    at java.rmi/sun.rmi.transport.tcp.TCPEndpoint.newSocket(TCPEndpoint.java:411)
    at java.rmi/sun.rmi.transport.tcp.TCPChannel.createConnection(TCPChannel.java:97)
    at java.rmi/sun.rmi.transport.tcp.TCPChannel.newConnection(TCPChannel.java:82)
    at java.rmi/sun.rmi.server.UnicastRef.newCall(UnicastRef.java:33)
    at java.rmi/sun.rmi.registry.RegistryImpl_Stub.lookup(RegistryImpl_Stub.java:111)
    at HelloClient.main(HelloClient.java:22)
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Client$
```

È la riga che contiene
LocateRegistry.getRegistry(1099)



Specifichiamo i permessi

- Un modo semplice per specificare i permessi consiste nel
 - 1) Preparare un file contenente la specifica dei permessi
 - 2) Lanciare il programma dando alla macchina Java a direttiva di usare quei permessi.

- Un esempio di file:

```
grant {  
    permission java.net.SocketPermission "127.0.0.1:1099",  
        "connect,resolve";  
    permission java.security.AllPermission;  
};
```

In questo modo si permette praticamente tutto.



Esecuzione con politiche permissive

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Server$ java HelloImpl
Server ready
^Cgigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Server$ java -Djava.security.policy=./policy HelloImpl
Server ready
█

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/RMI/Hell...
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Client$ java -Djava.security.policy=./policy HelloClient localhost
response: Hello, world!
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/RMI/Hello_test_3/Client$ █
```



Come ci regoliamo

- Negli esempi che vedremo NON useremo un security manager, approfittando del fatto che mettiamo sempre tutti i programmi dell'applicazione sulla medesima macchina.



Bibliografia

- Molte delle informazioni presenti su questa presentazione sono state estratte da:
 - ▶ Capitolo 6 di Concurrent and Distributed Computing in Java, di Vijay K. Garg
 - ▶ Capitolo 13 di Creating Components: Object Oriented, Concurrent, and Distributed Computing in Java, di Charles W. Kann