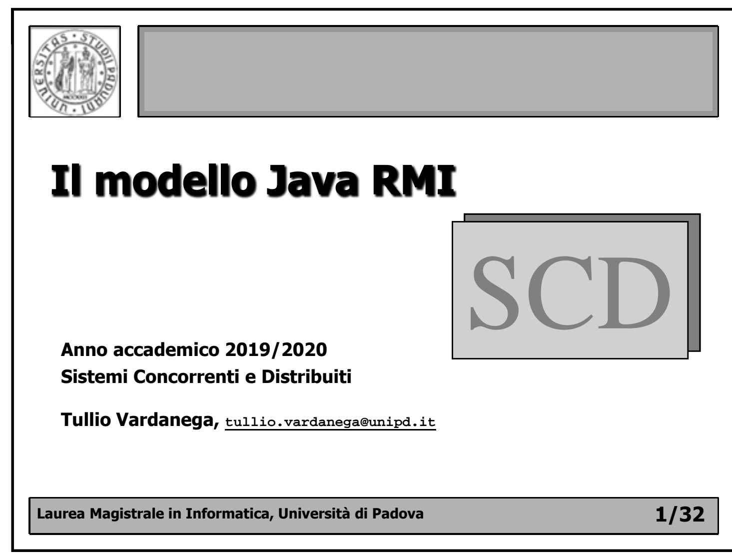
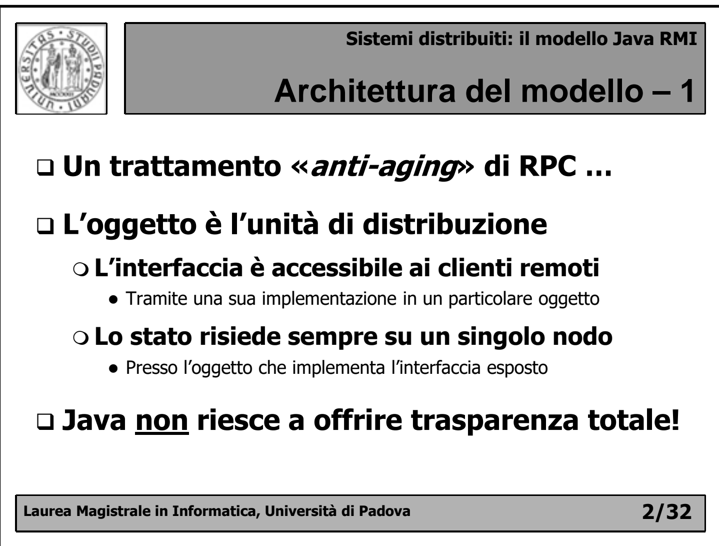
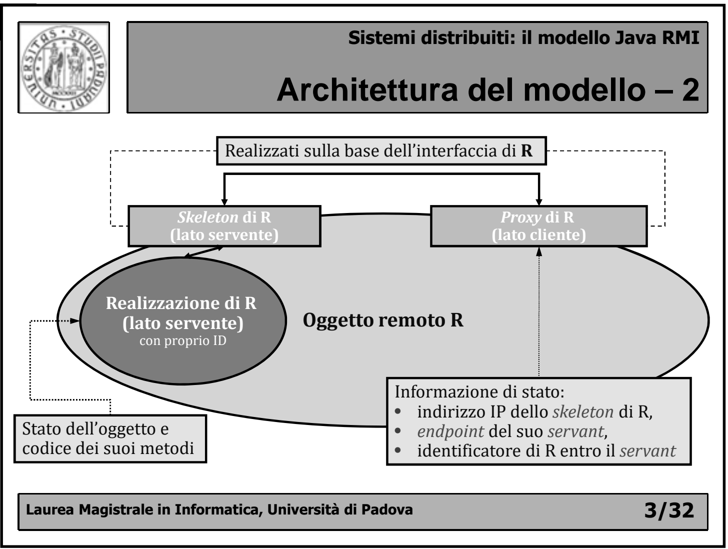
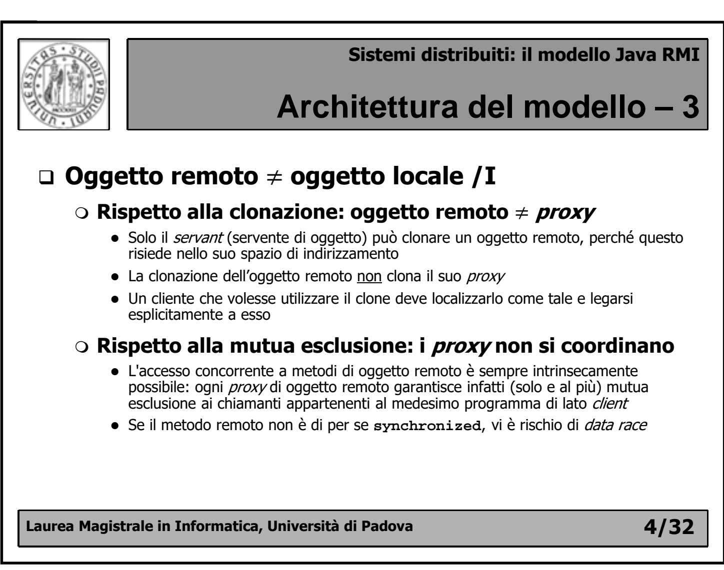
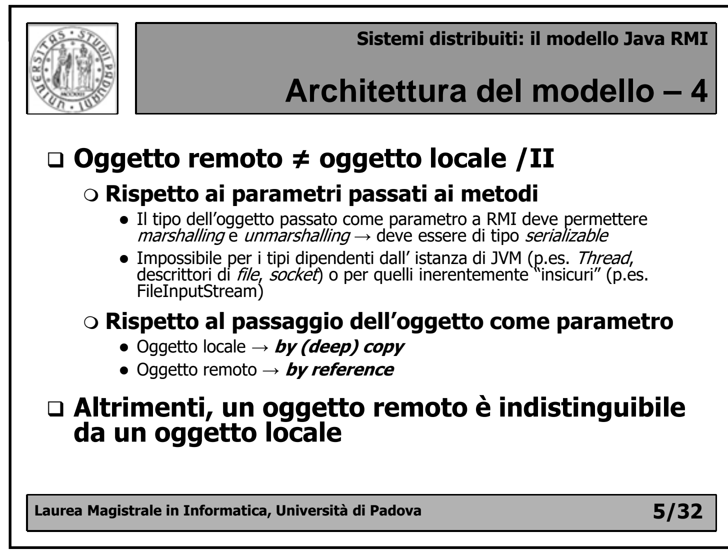
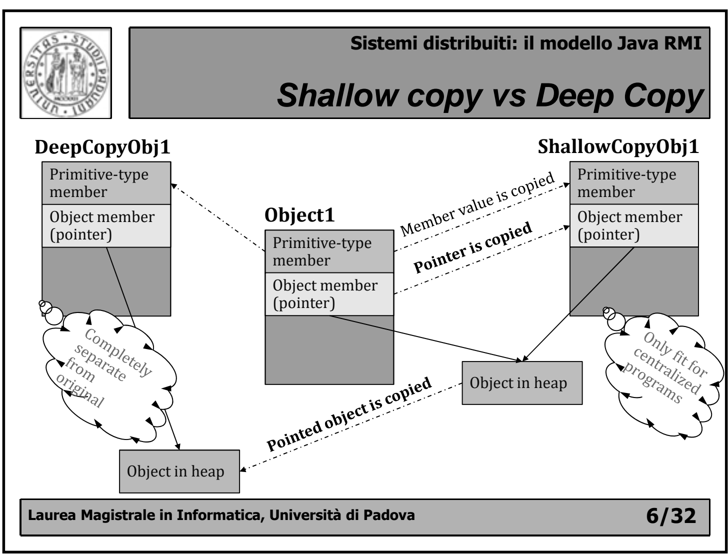

Il modello Java RMI
SCD
Anno accademico 2019/2020
Sistemi Concorrenti e Distribuiti
Tullio Vardanega, tullio.vardanega@unipd.it
Laurea Magistrale in Informatica, Università di Padova
1/32

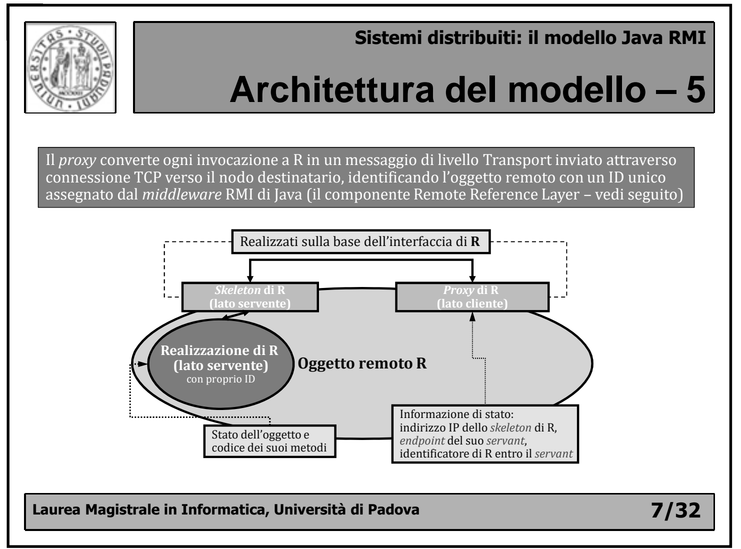

Sistemi distribuiti: il modello Java RMI
Architettura del modello – 1
Un trattamento «anti-aging» di RPC …
L'oggetto è l'unità di distribuzione
L'interfaccia è accessibile ai clienti remoti
Tramite una sua implementazione in un particolare oggetto
Lo stato risiede sempre su un singolo nodo
Presso l'oggetto che implementa l'interfaccia esposto
Java non riesce a offrire trasparenza totale!
Laurea Magistrale in Informatica, Università di Padova
2/32

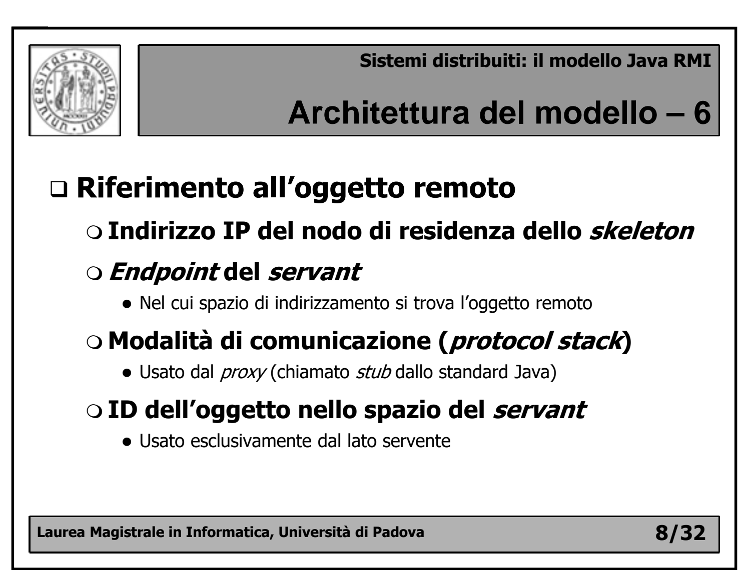

Sistemi distribuiti: il modello Java RMI
Architettura del modello – 2
Realizzati sulla base dell'interfaccia di R
Skeleton di R (lato servente)
Proxy di R (lato cliente)
Realizzazione di R (lato servente)
con proprio ID
Oggetto remoto R
Informazione di stato:
• indirizzo IP dello skeleton di R,
• endpoint del suo servant,
• identificatore di R entro il servant
Stato dell'oggetto e codice dei suoi metodi
Laurea Magistrale in Informatica, Università di Padova
3/32

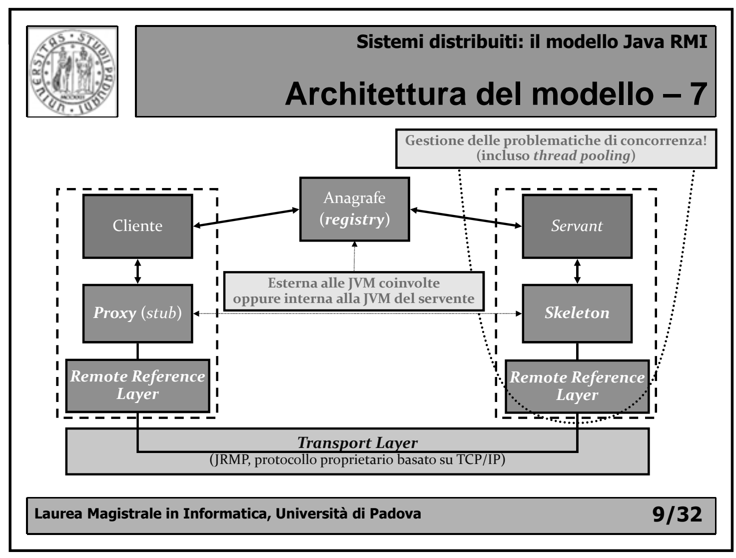

Sistemi distribuiti: il modello Java RMI
Architettura del modello – 3
Oggetto remoto ≠ oggetto locale /I
Rispetto alla clonazione: oggetto remoto ≠ proxy
Solo il servant (servente di oggetto) può clonare un oggetto remoto, perché questo risiede nello suo spazio di indirizzamento
La clonazione dell'oggetto remoto non clona il suo proxy
Un cliente che volesse utilizzare il clone deve localizzarlo come tale e legarsi esplicitamente a esso
Rispetto alla mutua esclusione: i proxy non si coordinano
L'accesso concorrente a metodi di oggetto remoto è sempre intrinsecamente possibile: ogni proxy di oggetto remoto garantisce infatti (solo e al più) mutua esclusione ai chiamanti appartenenti al medesimo programma di lato client
Se il metodo remoto non è di per se synchronized, vi è rischio di data race
Laurea Magistrale in Informatica, Università di Padova
4/32


Sistemi distribuiti: il modello Java RMI
Architettura del modello – 4
Oggetto remoto ≠ oggetto locale /II
Rispetto ai parametri passati ai metodi
Il tipo dell'oggetto passato come parametro a RMI deve permettere marshalling e unmarshalling → deve essere di tipo serializable
Impossibile per i tipi dipendenti dall' istanza di JVM (p.es. Thread, descrittori di file, socket) o per quelli inerentemente "insicuri" (p.es. FileInputStream)
Rispetto al passaggio dell'oggetto come parametro
Oggetto locale → by (deep) copy
Oggetto remoto → by reference
Altrimenti, un oggetto remoto è indistinguibile da un oggetto locale
Laurea Magistrale in Informatica, Università di Padova
5/32


Sistemi distribuiti: il modello Java RMI
Shallow copy vs Deep Copy
DeepCopyObj1
Primitive-type member
Object member (pointer)
Object1
Primitive-type member
Object member (pointer)
ShallowCopyObj1
Primitive-type member
Object member (pointer)
Member value is copied
Pointer is copied
Completely separate from original
Only fit for centralized programs
Object in heap
Pointed object is copied
Object in heap
Laurea Magistrale in Informatica, Università di Padova
6/32


Sistemi distribuiti: il modello Java RMI
Architettura del modello – 5
Il proxy converte ogni invocazione a R in un messaggio di livello Transport inviato attraverso connessione TCP verso il nodo destinatario, identificando l'oggetto remoto con un ID unico assegnato dal middleware RMI di Java (il componente Remote Reference Layer – vedi seguito)
Realizzati sulla base dell'interfaccia di R
Skeleton di R (lato servente)
Proxy di R (lato cliente)
Realizzazione di R (lato servente) con proprio ID
Oggetto remoto R
Stato dell'oggetto e codice dei suoi metodi
Informazione di stato: indirizzo IP dello skeleton di R, endpoint del suo servant, identificatore di R entro il servant
Laurea Magistrale in Informatica, Università di Padova
7/32


Sistemi distribuiti: il modello Java RMI
Architettura del modello – 6
Riferimento all'oggetto remoto
Indirizzo IP del nodo di residenza dello skeleton
Endpoint del servant
Nel cui spazio di indirizzamento si trova l'oggetto remoto
Modalità di comunicazione (protocol stack)
Usato dal proxy (chiamato stub dallo standard Java)
ID dell'oggetto nello spazio del servant
Usato esclusivamente dal lato servente
Laurea Magistrale in Informatica, Università di Padova
8/32

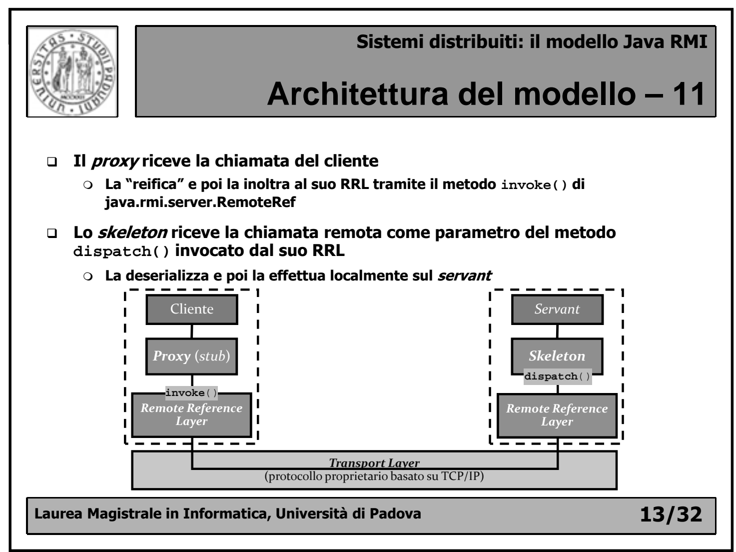


Sistemi distribuiti: il modello Java RMI

Architettura del modello – 10

- **Il *proxy* stesso è serializzabile**
  - **Passato come parametro (*by-copy*), può essere usato come riferimento all'oggetto remoto**
    - Non occorre copiare il codice del *proxy* presso il cliente: basta specificare le classi necessarie per rigenerarlo in loco
- **L'uso di RMI comporta la riproduzione del *proxy* presso il chiamante**
  - **Copia di quanto depositato nel *registry***

Laurea Magistrale in Informatica, Università di Padova 12/32



Sistemi distribuiti: il modello Java RMI

A look under the hood – 1

1. ***The servant creates an instance of the remote object, which must extend UnicastRemoteObject***
   - ***The constructor for UnicastRemoteObject enables the remote object to service incoming RMI calls***
   - ***A TCP socket bound to an arbitrary port is created***
   - ***A thread is also created to listen for connections on that socket***
2. ***The servant registers the remote object with the RMI registry and hands it the corresponding proxy***
   - ***The RMIRegistry holds remote object proxies and hands them off to clients on request***
   - ***The proxy contains the information needed to "call back" to the servant when the client invokes it***

Laurea Magistrale in Informatica, Università di Padova **14/32**

Sistemi distribuiti: il modello Java RMI

A look under the hood – 2

3. ***The client obtains the proxy by calling the RMI registry***
   - ***If the server specified a codebase for clients to obtain the proxy's classfile, that information will be passed to the client via the registry***
   - ***The client can then use the codebase to resolve the proxy class to load the stub classfile***

Laurea Magistrale in Informatica, Università di Padova **15/32**

Sistemi distribuiti: il modello Java RMI

A look under the hood – 3

4. ***When the client issues an RMI to the servant the proxy class creates a "RemoteCall"***
   - ***This opens a socket to the servant on the port specified in the proxy and sends the RMI header information to it***
5. ***The proxy class calls RemoteCall.executeCall to cause the RMI to happen***
   - ***Using RemoteCall, the proxy class serializes the call arguments into a Java object and marshals them over the connection***

Laurea Magistrale in Informatica, Università di Padova **16/32**

Sistemi distribuiti: il modello Java RMI

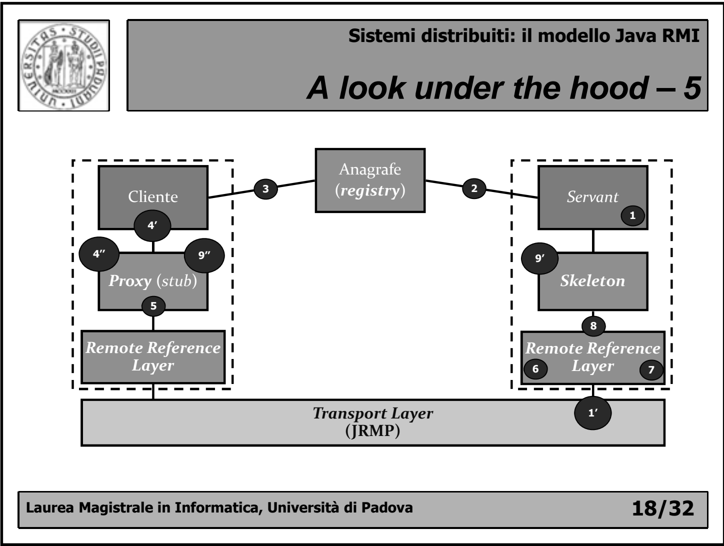
A look under the hood – 4

6. ***When a client connects to the servant's socket, a new thread is forked on the servant's side to service the incoming call***
   - ***The original thread can continue listening to the original socket so that new calls from other clients can be made***
7. ***The servant reads the RMI header information and creates a RemoteCall to unmarshal the incoming RMI arguments***
8. ***The servant calls the "dispatch" method of the skeleton class which calls the target method on the object and pushes the result back to the socket***
9. ***The return value of the RMI is unmarshaled on the client side, and returned from the proxy back to the client code***

Laurea Magistrale in Informatica, Università di Padova 17/32

Sistemi distribuiti: il modello Java RMI

A look under the hood – 5



Laurea Magistrale in Informatica, Università di Padova 18/32

Sistemi distribuiti: il modello Java RMI

RMI e concorrenza – 1

- ***RMI Spec @ 3.2 Thread Usage in RMI***
  - ***A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread***
  - ***The RMI runtime makes no guarantees with respect to mapping invocations to threads***
  - ***Since remote method invocation on the same remote object may execute concurrently, a remote object implementation needs to make sure its implementation is thread-safe***
    - *"it's your problem, baby"*

Reentrancy?

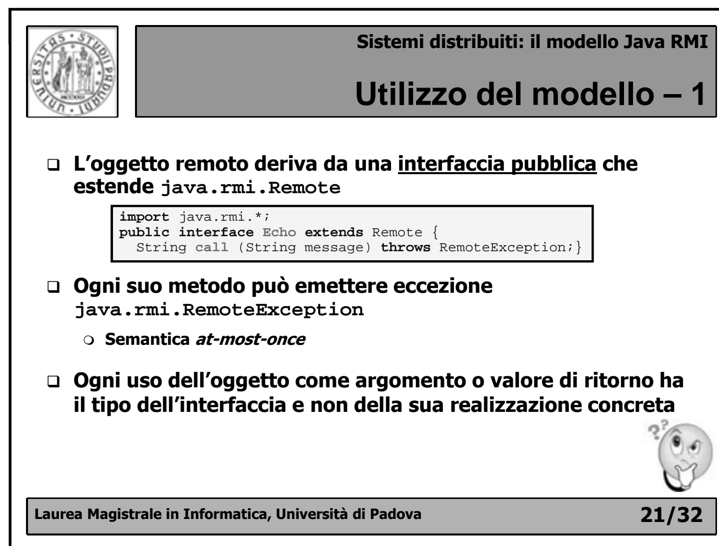
Laurea Magistrale in Informatica, Università di Padova 19/32

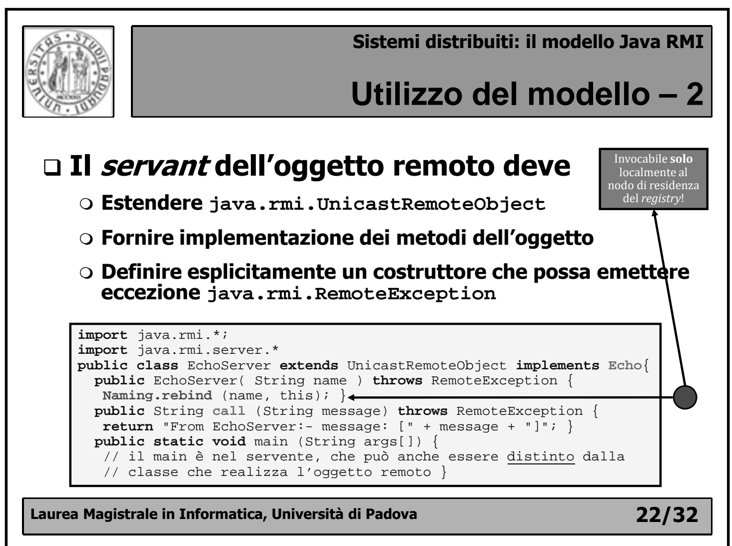
Sistemi distribuiti: il modello Java RMI

RMI e concorrenza – 2

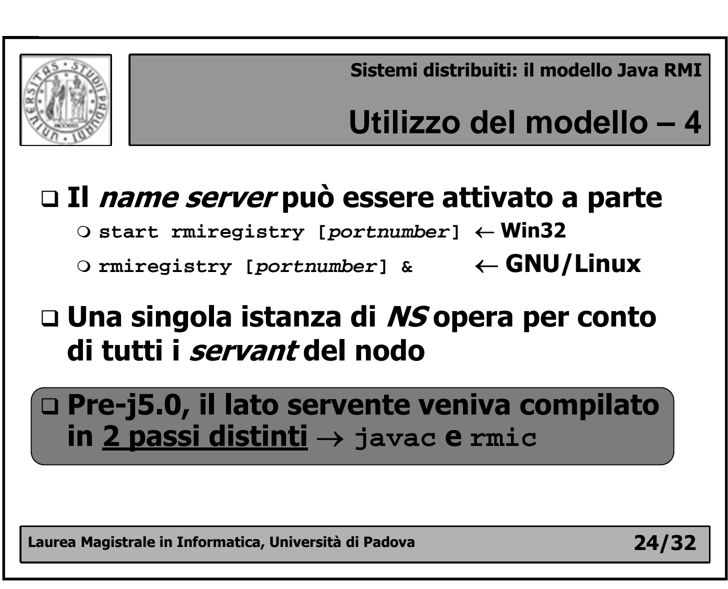
- **L'invocazione di lato cliente è bloccante**
  - **Uno stesso cliente non può inviare richieste concorrenti allo stesso oggetto remoto**
  - **Il *proxy* di quel cliente non rischia *data race*, a meno di aver condiviso il *proxy* …**
- **Dal lato servente, ogni invocazione in arrivo è potenzialmente servita in un *thread* distinto**
  - **L'implementazione *thread-safe* dell'oggetto remoto gestisce la sincronizzazione di chiamate concorrenti**

Laurea Magistrale in Informatica, Università di Padova 20/32

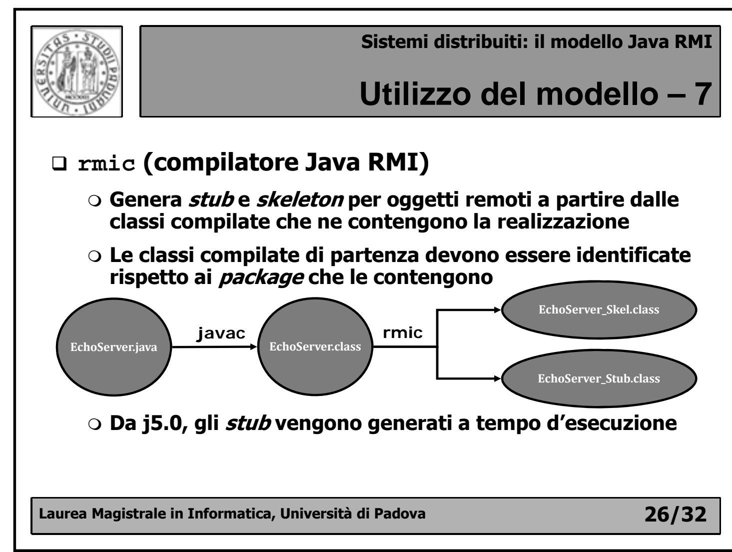

Sistemi distribuiti: il modello Java RMI
Utilizzo del modello – 1
L'oggetto remoto deriva da una interfaccia pubblica che estende java.rmi.Remote
import java.rmi.*;
public interface Echo extends Remote {
String call (String message) throws RemoteException;}
Ogni suo metodo può emettere eccezione java.rmi.RemoteException
Semantica at-most-once
Ogni uso dell'oggetto come argomento o valore di ritorno ha il tipo dell'interfaccia e non della sua realizzazione concreta
Laurea Magistrale in Informatica, Università di Padova
21/32

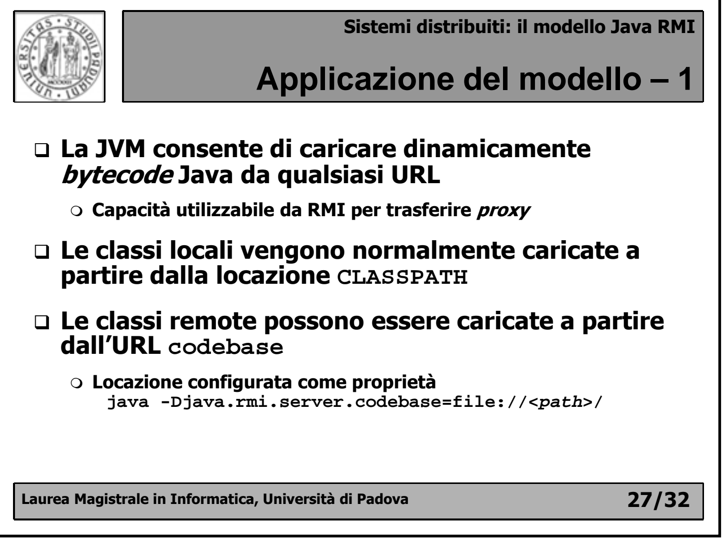

Sistemi distribuiti: il modello Java RMI
Utilizzo del modello – 2
Il servant dell'oggetto remoto deve
Estendere java.rmi.UnicastRemoteObject
Fornire implementazione dei metodi dell'oggetto
Definire esplicitamente un costruttore che possa emettere eccezione java.rmi.RemoteException
Invocabile solo localmente al nodo di residenza del registry!
import java.rmi.*;
import java.rmi.server.*
public class EchoServer extends UnicastRemoteObject implements Echo{
public EchoServer(String name) throws RemoteException {
Naming.rebind (name, this); }
public String call (String message) throws RemoteException {
return "From EchoServer:- message: [" + message + "]"; }
public static void main (String args[]) {
// il main è nel servente, che può anche essere distinto dalla
// classe che realizza l'oggetto remoto }
Laurea Magistrale in Informatica, Università di Padova
22/32

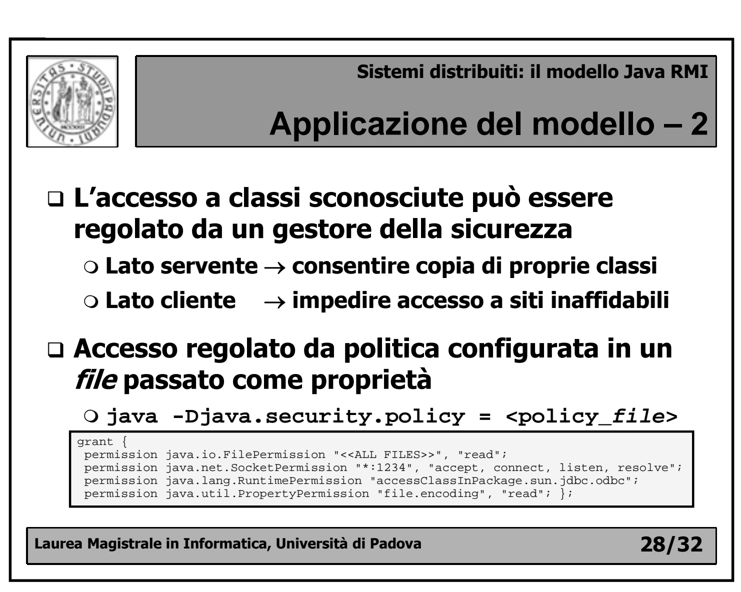
Sistemi distribuiti: il modello Java RMI
Utilizzo del modello – 3
La logica del servant è specificata nel suo main che crea istanze dell'oggetto remoto
Ogni istanza va registrata presso l'anagrafe degli oggetti remoti del nodo del servente, mantenuto da un processo dedicato (rmiregistry)
Naming.bind
lega un nome (stringa URL) all'oggetto remoto (al suo riferimento) in una associazione unica e non modificabile
Naming.rebind
crea una nuova associazione (nome, riferimento) sovrascrivendo quella precedente
Il gestore del registro (name server locale) ascolta su una porta assegnata (default: 1099)
Laurea Magistrale in Informatica, Università di Padova
23/32

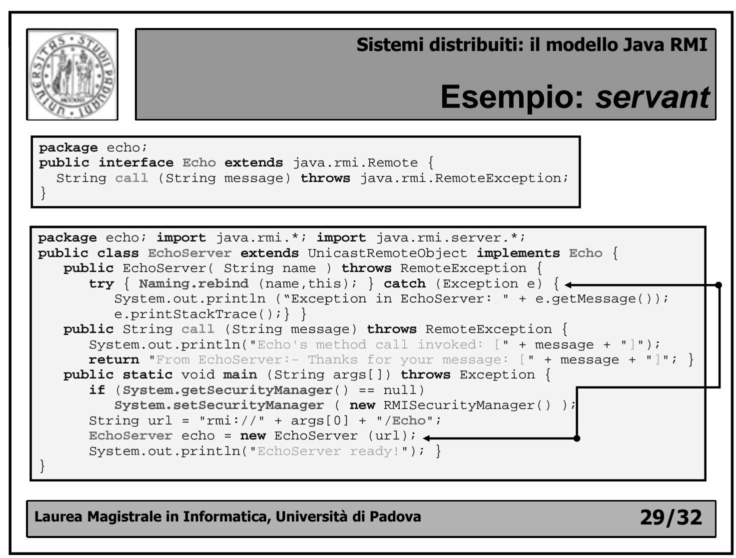

Sistemi distribuiti: il modello Java RMI
Utilizzo del modello – 4
Il name server può essere attivato a parte
start rmiregistry [portnumber] ← Win32
rmiregistry [portnumber] & ← GNU/Linux
Una singola istanza di NS opera per conto di tutti i servant del nodo
Pre-j5.0, il lato servente veniva compilato in 2 passi distinti → javac e rmic
Laurea Magistrale in Informatica, Università di Padova
24/32

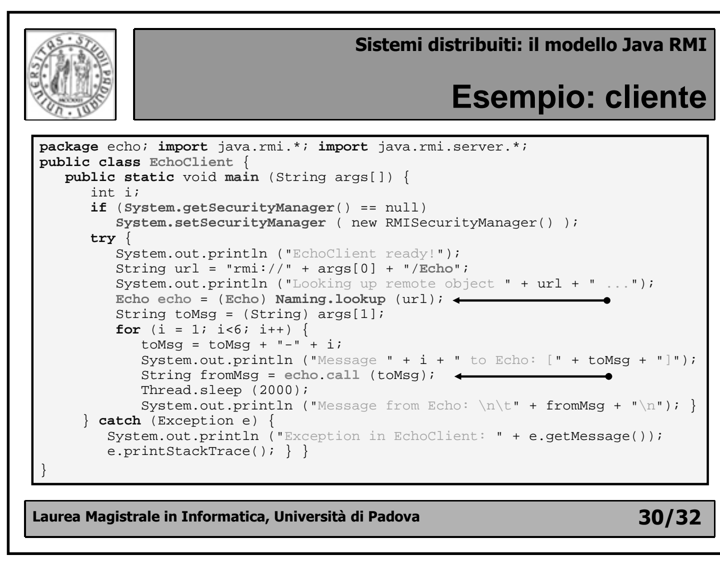
Sistemi distribuiti: il modello Java RMI
Utilizzo del modello – 5
Il cliente dell'oggetto remoto
Fa look-up presso un qualunque NS tramite
Naming.lookup (String name)
name è l'URI della specifica dell'oggetto remoto (la sua interfaccia pubblica) presso il nodo e la porta dove è in ascolto il NS
Ottenendo un riferimento con il tipo dell'interfaccia e non della classe che lo realizza!
Il NS ha un ObjID riservato: closure implicita
Da ora in avanti l'oggetto remoto è usabile come un oggetto locale
Laurea Magistrale in Informatica, Università di Padova
25/32

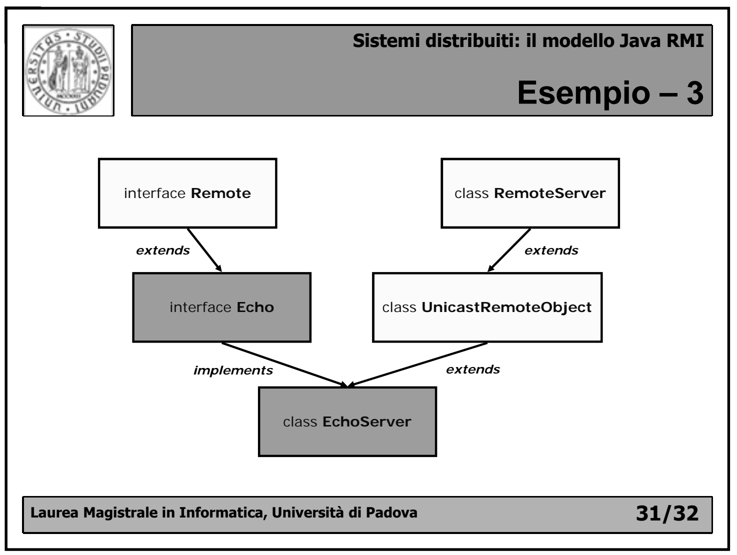

Sistemi distribuiti: il modello Java RMI
Utilizzo del modello – 7
rmic (compilatore Java RMI)
Genera stub e skeleton per oggetti remoti a partire dalle classi compilate che ne contengono la realizzazione
Le classi compilate di partenza devono essere identificate rispetto ai package che le contengono
EchoServer.java
javac
EchoServer.class
rmic
EchoServer_Skel.class
EchoServer_Stub.class
Da j5.0, gli stub vengono generati a tempo d'esecuzione
Laurea Magistrale in Informatica, Università di Padova
26/32

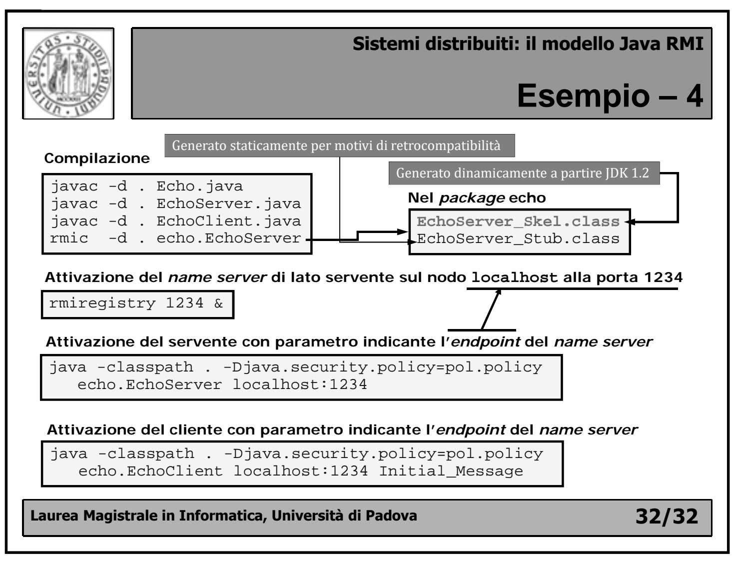

Sistemi distribuiti: il modello Java RMI
Applicazione del modello – 1
La JVM consente di caricare dinamicamente bytecode Java da qualsiasi URL
Capacità utilizzabile da RMI per trasferire proxy
Le classi locali vengono normalmente caricate a partire dalla locazione CLASSPATH
Le classi remote possono essere caricate a partire dall'URL codebase
Locazione configurata come proprietà
java -Djava.rmi.server.codebase=file://<path>/
Laurea Magistrale in Informatica, Università di Padova
27/32


Sistemi distribuiti: il modello Java RMI
Applicazione del modello – 2
L'accesso a classi sconosciute può essere regolato da un gestore della sicurezza
Lato servente → consentire copia di proprie classi
Lato cliente → impedire accesso a siti inaffidabili
Accesso regolato da politica configurata in un file passato come proprietà
java -Djava.security.policy = <policy_file>
grant {
permission java.io.FilePermission "<<ALL FILES>>", "read";
permission java.net.SocketPermission "*:1234", "accept, connect, listen, resolve";
permission java.lang.RuntimePermission "accessClassInPackage.sun.jdbc.odbc";
permission java.util.PropertyPermission "file.encoding", "read"; };
Laurea Magistrale in Informatica, Università di Padova
28/32


Sistemi distribuiti: il modello Java RMI
Esempio: servant
package echo;
public interface Echo extends java.rmi.Remote {
String call (String message) throws java.rmi.RemoteException;
}
package echo; import java.rmi.*; import java.rmi.server.*;
public class EchoServer extends UnicastRemoteObject implements Echo {
public EchoServer(String name) throws RemoteException {
try { Naming.rebind (name,this); } catch (Exception e) {
System.out.println (“Exception in EchoServer: " + e.getMessage());
e.printStackTrace();} }
public String call (String message) throws RemoteException {
System.out.println("Echo's method call invoked: [" + message + "]");
return "From EchoServer:- Thanks for your message: [" + message + "]"; }
public static void main (String args[]) throws Exception {
if (System.getSecurityManager() == null)
System.setSecurityManager (new RMISecurityManager());
String url = "rmi://" + args[0] + "/Echo";
EchoServer echo = new EchoServer (url);
System.out.println("EchoServer ready!"); }
}
Laurea Magistrale in Informatica, Università di Padova
29/32


Sistemi distribuiti: il modello Java RMI
Esempio: cliente
package echo; import java.rmi.*; import java.rmi.server.*;
public class EchoClient {
public static void main (String args[]) {
int i;
if (System.getSecurityManager() == null)
System.setSecurityManager (new RMISecurityManager());
try {
System.out.println ("EchoClient ready!");
String url = "rmi://" + args[0] + "/Echo";
System.out.println ("Looking up remote object " + url + " ...");
Echo echo = (Echo) Naming.lookup (url);
String toMsg = (String) args[1];
for (i = 1; i<6; i++) {
toMsg = toMsg + "-" + i;
System.out.println ("Message " + i + " to Echo: [" + toMsg + "]");
String fromMsg = echo.call (toMsg);
Thread.sleep (2000);
System.out.println ("Message from Echo: \n\t" + fromMsg + "\n"); }
} catch (Exception e) {
System.out.println ("Exception in EchoClient: " + e.getMessage());
e.printStackTrace(); } }
}
Laurea Magistrale in Informatica, Università di Padova
30/32


Sistemi distribuiti: il modello Java RMI
Esempio – 3
interface Remote
class RemoteServer
extends
extends
interface Echo
class UnicastRemoteObject
implements
extends
class EchoServer
Laurea Magistrale in Informatica, Università di Padova
31/32


Sistemi distribuiti: il modello Java RMI
Esempio – 4
Generato staticamente per motivi di retrocompatibilità
Compilazione
Generato dinamicamente a partire JDK 1.2
javac -d . Echo.java
javac -d . EchoServer.java
javac -d . EchoClient.java
rmic -d . echo.EchoServer
Nel package echo
EchoServer_Skel.class
EchoServer_Stub.class
Attivazione del name server di lato servente sul nodo localhost alla porta 1234
rmiregistry 1234 &
Attivazione del servente con parametro indicante l’endpoint del name server
java -classpath . -Djava.security.policy=pol.policy
echo.EchoServer localhost:1234
Attivazione del cliente con parametro indicante l’endpoint del name server
java -classpath . -Djava.security.policy=pol.policy
echo.EchoClient localhost:1234 Initial_Message
Laurea Magistrale in Informatica, Università di Padova
32/32