Pemrograman Jaringan 11

RMI

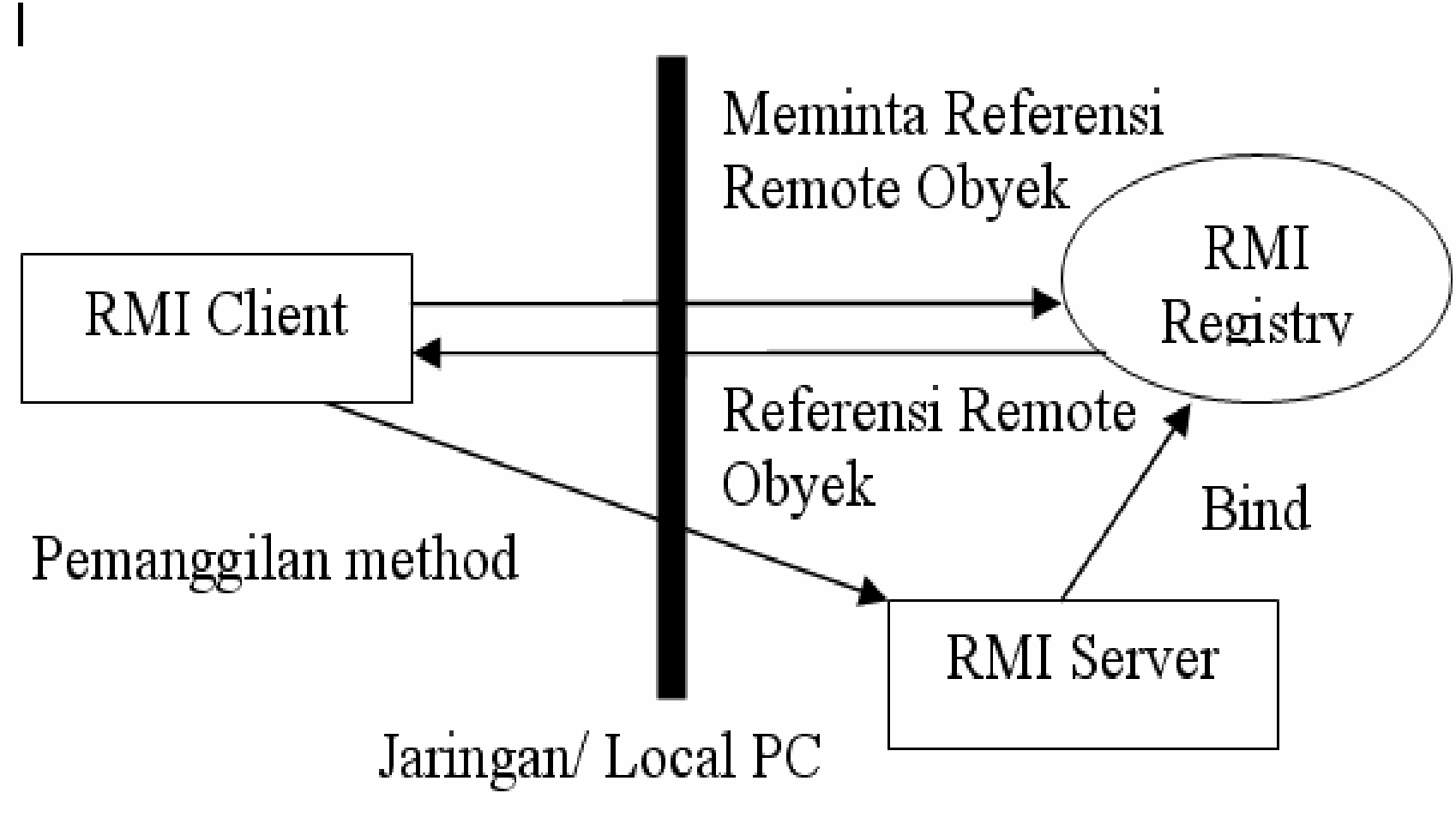
Pengantar RMI

- RMI adalah salah satu bagian dari J2SE yang digunakan untuk membangun aplikasi terdistribusi menggunakan bahasa Java.
- RMI adalah kumpulan kelas dalam Java yang digunakan untuk menangani pemanggilan (invocation) method secara jarak jauh (remote) dalam suatu jaringan atau Internet.
- Idenya memisahkan obyek-obyek secara terdistribusi dalam mesin-mesin yang berbeda.
- RMI menggunakan prinsip pemrograman berorientasi obyek dimana obyek satu dapat saling berkomunikasi dengan obyek lainnya.
- Untuk membangun aplikasi RMI dibutuhkan Interface.
- RMI terdiri dari RMI client dan server.

Pengantar RMI (2)

- RMI server biasanya akan membuat beberapa remote obyek dan referensi-nya yang dapat diakses oleh RMI client menggunakan suatu URL dan menunggu RMI client meminta ke server.
- Sedangkan RMI client akan membuat koneksi ke server dan meminta pemanggilan ke beberapa remote obyek berdasarkan referensi yang diterimanya.
- RMI client akan menggunakan remote obyek sebagai lokal obyek.
- Setiap remote obyek yang dibuat oleh RMI server didaftarkan terlebih dahulu ke dalam RMI registri, agar ketika client membutuhkannya dapat meminta dengan mudah ke RMI registry.

Arsitektur RMI



Pengantar RMI (3)

- RMI Server akan mendaftarkan remote obyeknya ke RMI Registry melalui *bind* dengan nama unik. RMI Client yang akan melakukan suatu pemanggilan method dari remote obyek, harus meminta referensi obyek ke RMI Registry berdasarkan nama kelas obyek tersebut.
- Dalam RMI harus ada pendefinisian interface (behaviour) dan implementasi interface (berupa kelas)
- RMI hanya dimiliki oleh bahasa Java saja.

Teknologi Terkait dengan RMI

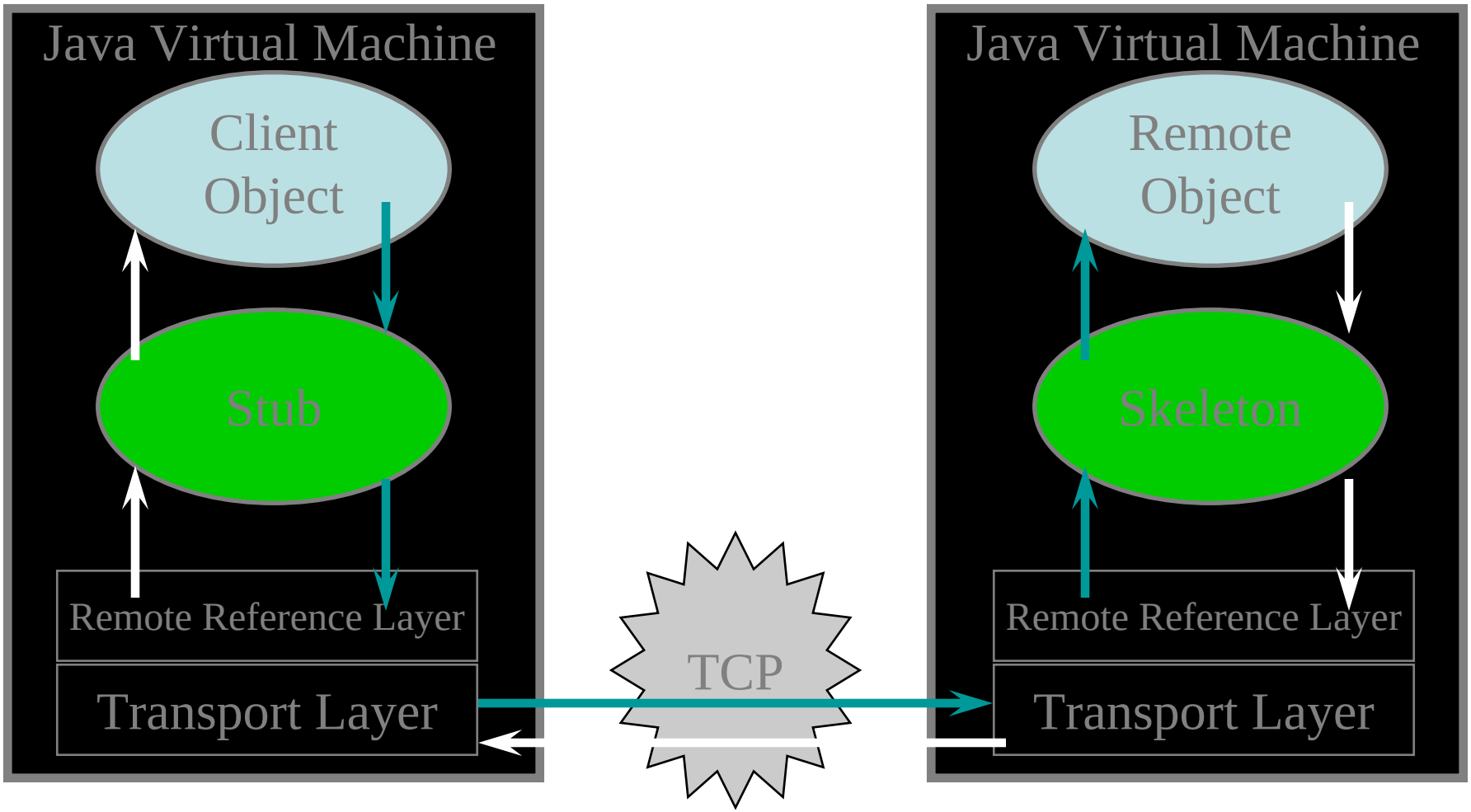
- **RPC** ("Remote Procedure Calls")
  - Developed by Sun
  - Platform-specific
- **CORBA** ("Common Object Request Broker Architecture")
  - Developed by OMG
  - Access to non-Java objects (as well as Java)
- **DCOM** ("Distributed Common Object Model")
  - Developed by Microsoft
  - Access to Win32 objects
- **LDAP** ("Lightweight Directory Access Protocol")
  - Finding resources on a network

Bagian Arsitektur RMI

- *Transport Layer*
- *Remote Reference Layer*
- *Stub* dan *Skeleton Layer*

- Remote Objects
  - Ada pada server
  - Diakses seperti layaknya obyek lokal

RMI Layers


Java Virtual Machine
Client Object
Stub
Remote Reference Layer
Transport Layer
TCP
Java Virtual Machine
Remote Object
Skeleton
Remote Reference Layer
Transport Layer

Stub & Skeleton

- Merupakan interface antara aplikasi dan RMI system.
- Stub bertindak sebagai client side proxy
- Skeleton bertindak sebagai server side proxy
- Selama remote invocation stub bertanggung jawab untuk:
  - Meminta lokasi remote server obyek pada remote reference layer
  - Marshalling : merangkaian argumen pada output stream
  - Memberitahu remote reference layer bahwa semua data parameter telah terkirim, sehingga pemanggilan method sesungguhnya dapat dilakukan oleh server
  - Unmarshalling: rangkaian nilai yang diterima dari remote obyek
  - Memberitahu remote reference layer bahwa pemanggilan telah lengkap
- Skeleton bertanggung jawab untuk:
  - Marshalling: nilai kembalian atau exception kepada stub client
  - Mengirimkan panggilan method pada server object sesungguhnya

Remote Reference

- Menemukan lokasi remote obyek
- Membuat panggilan point to point dan rekoneksi secara otomatis
- Mengaktifkan proses server baru jika belum pernah diaktifkan sebelumnya
- Memelihara replikasi (panggandaan) jika diperlukan

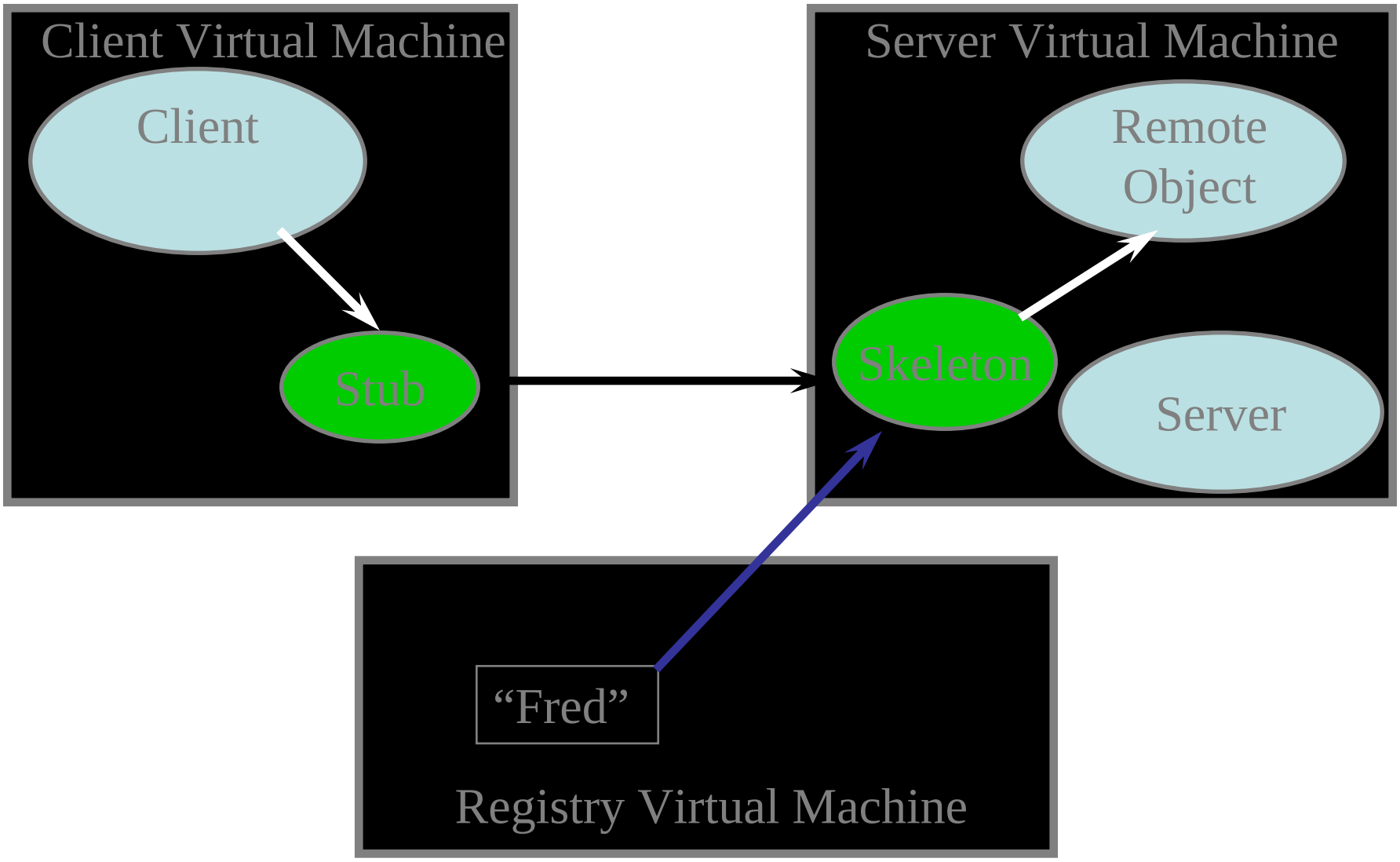
Transport Layer

- Memelihara tabel yang berisi obyek-obyek pada JVM
- Membuat dan memelihara dua koneksi antara 2 JVM menggunakan TCP/IP
- Transport Layer hanya tersedia di level virtual machine (pada java.net)
- Menerima dan merespon setiap pemanggilan dari atau ke server dan client

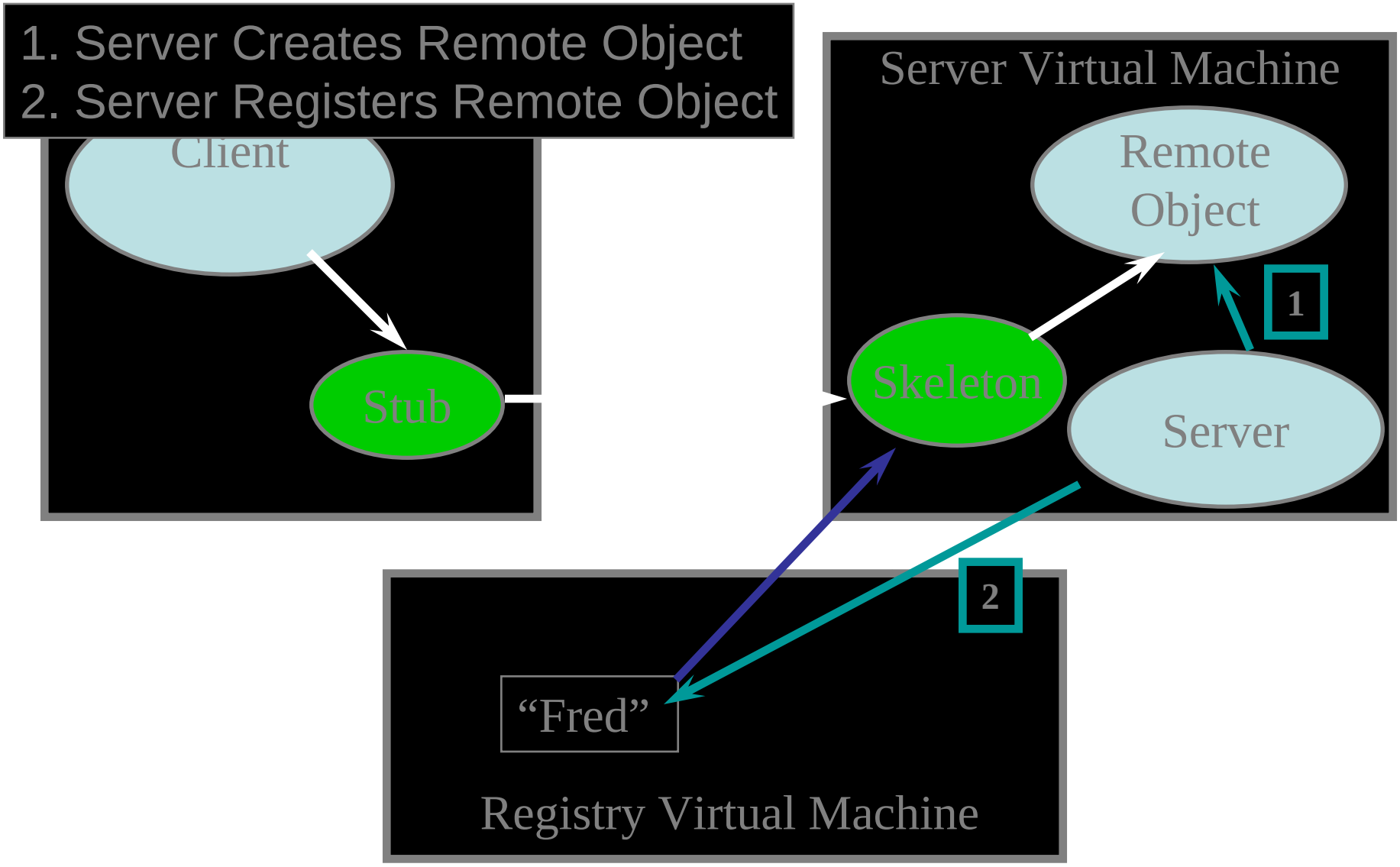
RMI Registry

- Tool RMI registry menggunakan rmiregistry dengan port default **1099**
- Ketika server membuat remote method dengan cara membuat lokal obyek yang menerapkan method dari interface tersebut, maka obyek akan diekspor ke RMI, dan diregisterkan ke RMI Registry dengan **public name**. RMI Registry akan membuat layanan listen yang menunggu permintaan dari client.
- Di sisi Client, RMI Registry diakses menggunakan static class naming. Class ini menyediakan metode lookup() untuk melakukan query ke registry. Metode lookup menerima URL yang menyatakan nama server dan nama service yang diminta dan kemudian mengembalikan remote reference obyek yang diminta. Format URL RMI:
- rmi://<hostName>[:<name_service_port>]/<service_name>
- RMI registry proses yang berjalan pada host machine

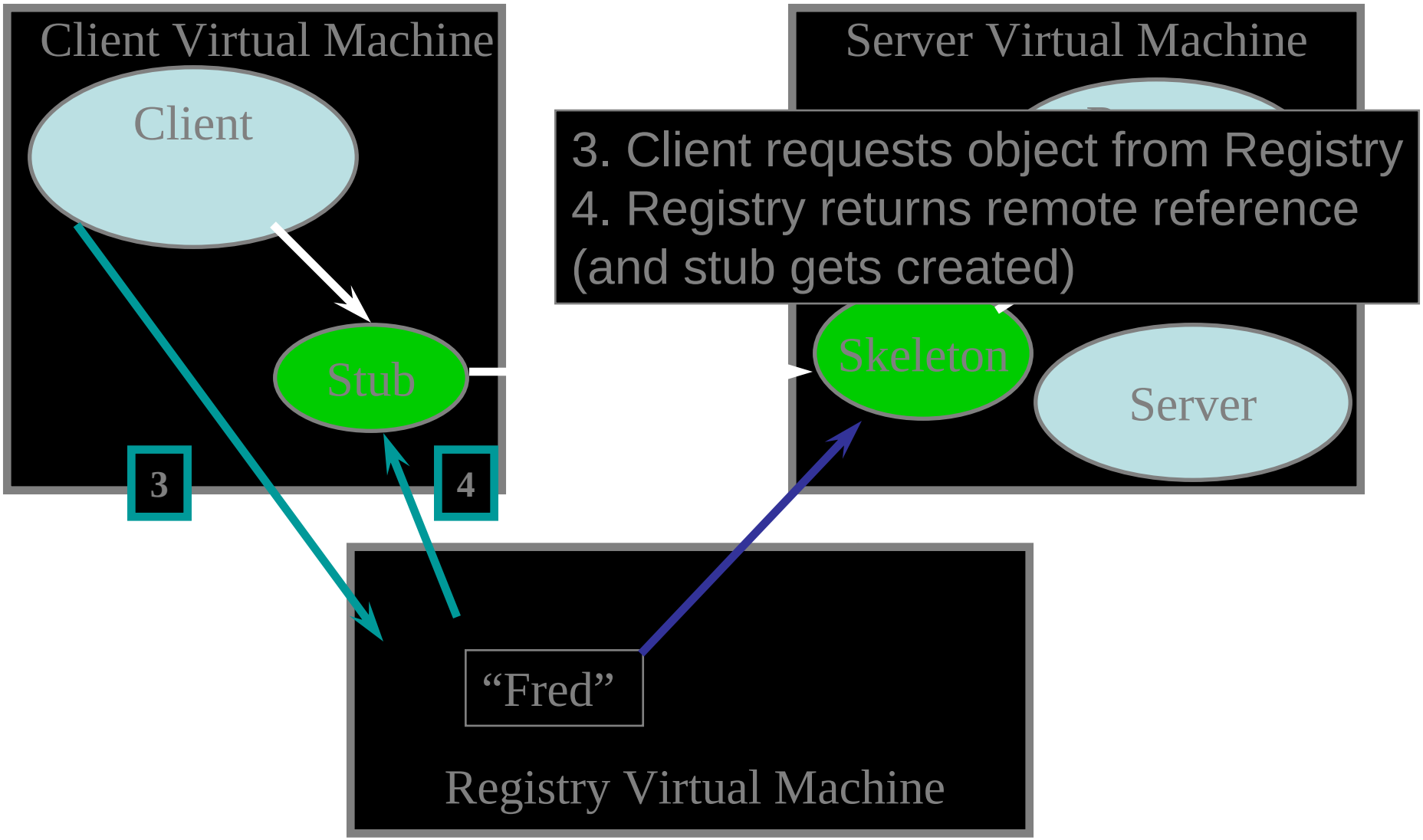
RMI System Architecture



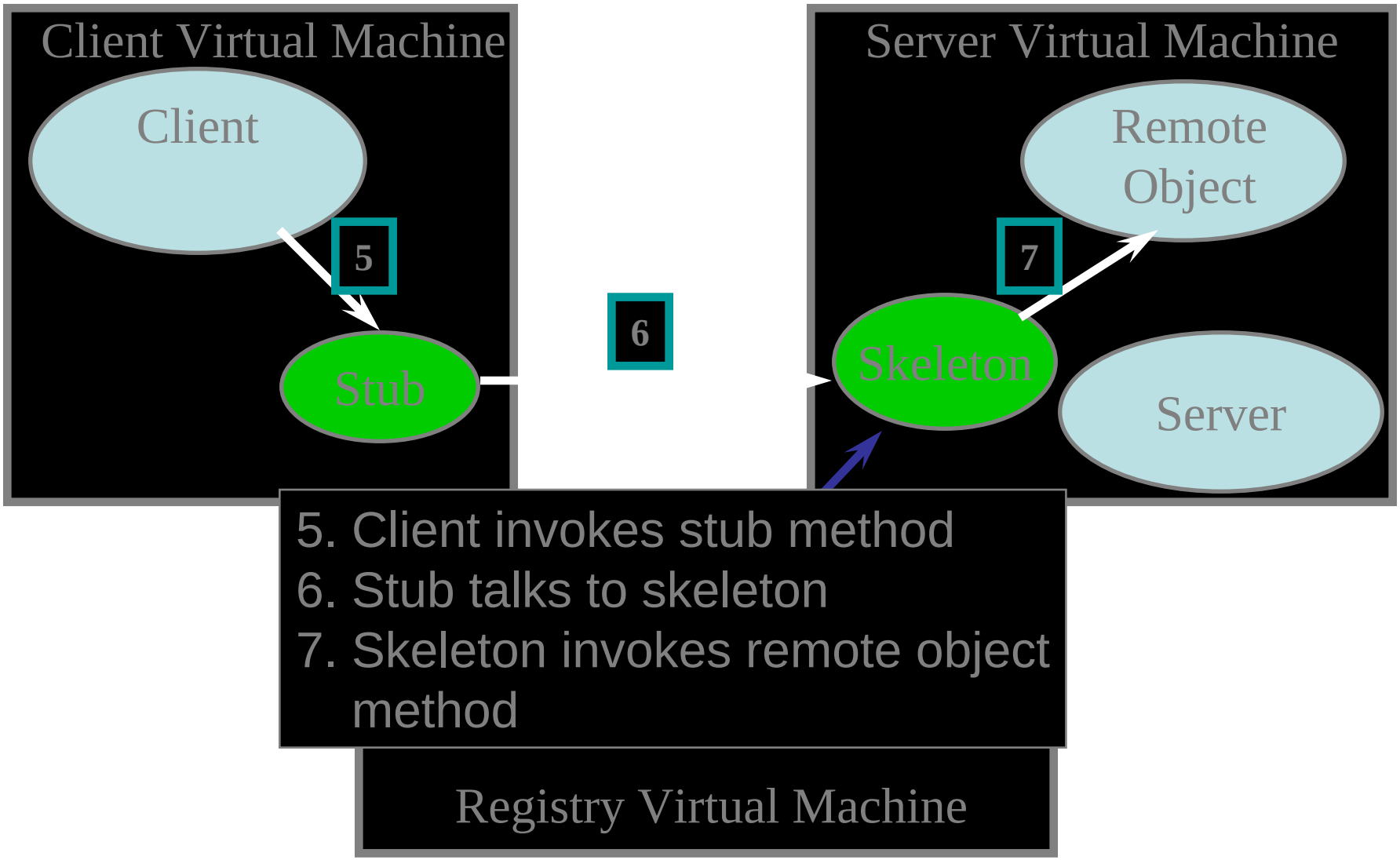
RMI Flow


1. Server Creates Remote Object
2. Server Registers Remote Object
Client
Stub
Server Virtual Machine
Remote Object
1
Skeleton
Server
2
“Fred”
Registry Virtual Machine

RMI Flow


Client Virtual Machine
Client
Stub
3
4
Server Virtual Machine
3. Client requests object from Registry
4. Registry returns remote reference
(and stub gets created)
Skeleton
Server
"Fred"
Registry Virtual Machine

RMI Flow


Client Virtual Machine
Client
5
Stub
6
Server Virtual Machine
Remote Object
7
Skeleton
Server
5. Client invokes stub method
6. Stub talks to skeleton
7. Skeleton invokes remote object method
Registry Virtual Machine

Pembuatan Aplikasi RMI

- Definisikan interface
- Interface ini akan diimplementasikan baik oleh client maupun server
- Template:

```
public interface NamaInterface extends java.rmi.Remote{
     public NamaMethod(parameter) throws
java.rmi.RemoteException;
     ...
}
```

Contoh

Contoh:

```
public interface Hai extends java.rmi.Remote {
   public void setText(String text)
      throws java.rmi.RemoteException;
   public String getText()
      throws java.rmi.RemoteException;
}
```

- Kelas NamaImplementasi berarti mengimplementasikan NamaInterface sesuai dengan yang telah didefinisikan diatas sehingga nama method-methodnya harus juga melemparkan Error ke java.rmi.RemoteException.
- Kelas ini juga mengextends dari kelas java.rmi.server.UnicastRemoteObject yang menangani remote object, membuat remote object, dan menangani panggilan dari client.

Pembuatan Aplikasi RMI

- Definisikan kelas implementasi dari interface
- Kelas yang mengimplementasikan interface.
- **Template**:

```
public class NamaImplementasi extends
java.rmi.server.UnicastRemoteObject implements NamaInterface{
     public NamaMethod(parameter) throws
java.rmi.RemoteException;
     ...
}
```

Contoh

```
public class HaiImpl
      extends java.rmi.server.UnicastRemoteObject
      implements Hai {
   private String s;
   public HaiImpl() throws java.rmi.RemoteException {
        // untuk mengaktifkan
        // program UnicastRemoteObject untuk
        // membangun sambungan RMI dan
        // inisialisasi remote object
     super();
   }
   public void setText(String s)
      throws java.rmi.RemoteException {
        this.s = s;
   }
   public String getText()
         throws java.rmi.RemoteException {
        return this.s;
   }
}
```

Pembuatan Aplikasi RMI (3)

- Buat stub dan skeleton
  - Cara jika menggunakan RMI versi 1 (stub dan skeleton):

    `rmic –v1.1 NamaImplementasi`
  - Cara jika menggunakan RMI versi 2 (stub saja):

    `rmic –v1.2 NamaImplementasi`
  - Contoh: `rmic –v1.2 HaiImpl` akan menghasilkan:

    `Hai.class`

    `HaiImpl.class`

    `HaiImpl_Stub.class`

Pembuatan Aplikasi RMI (4)

- Buat aplikasi remote server
- Aplikasi server ini akan membuat instant (object) dari kelas implementasi yang telah dibuat pada langkah-langkah sebelumnya dan juga akan mendaftarkan obyek tersebut ke RMI Registry dengan suatu URL tertentu.

- The server builds an object and register it with a particular URL
- Use `Naming.rebind` (replace any previous bindings) or `Naming.bind` (throw AlreadyBoundException if a previous binding exists)

```
import java.rmi.Naming;

public class HaiServer {
   public HaiServer() {
      try {
         Hai hallo = new HaiImpl();
         Naming.rebind("rmi://localhost:1099/Hai",
                       hallo);
      } catch (Exception e) {
         System.out.println("Trouble: " + e);
      }
   }

   public static void main(String args[]) {
      new HaiServer();
   }
}
```

Catch

```
catch(java.rmi.UnknownHostException e) {
    System.out.println("Host tidak diketahui!"
    + e);
}
catch(java.rmi.RemoteException e) {
    System.out.println("Ada kegagalan eksekusi
    remote object!" + e);
}
catch(java.net.MalformedURLException e) {
    System.out.println("Internal error " + e);
}
catch(java.rmi.NotBoundException e) {
    System.out.println("Proses bind gagal! " +
    e);
}
```

Pembuatan Aplikasi RMI (5)

- Buat aplikasi remote client
- Pada aplikasi client, client akan mencari obyek pada remote server dan melakukan casting ketipe yang sesuai dengan nama interface yang didefinisikan pada langkah pertama dan menggunakan obyek tersebut sebagai obyek lokal.

- Look up the object from the host using `Naming.lookup`, cast it to the appropriate type, then use it like a local object

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

public class HaiClient {
   public static void main(String[] args) {
      try {
         Hai helloClient = (Hai)
             Naming.lookup("rmi://localhost/Hai");
         helloClient.setText("ini dari helloClient");
         System.out.println(helloClient.getText());
      }
      catch (MalformedURLException murle) {
         System.out.println();
         System.out.println("MalformedURLException");
         System.out.println(murle);
      }
      catch (RemoteException re) {
         System.out.println();
         System.out.println("RemoteException");
         System.out.println(re);
      }
      catch (NotBoundException nbe) {
         System.out.println();
         System.out.println("NotBoundException");
         System.out.println(nbe);
      }
   }
```

Pembuatan Aplikasi RMI (6)

- Jalankan RMIRegistry
  - `start rmiregistry`
- Jalankan server
- Jalankan client

RMI Integration & Configuration

- RMI Stub & Skeleton ditempatkan pada HTTP server agar bisa didownload
- Client harus menginstall RMISecurityManager agar bisa me-load class RMI secara remote
- Client membutuhkan file policy agar dapat berkoneksi dengan Registry
  - Bisa juga di set di JAVA_HOME\jre\lib\security pada file

```
grant {
   // rmihost - RMI registry and the server
   // webhost - HTTP server for stub classes
   permission java.net.SocketPermission
      "rmihost:1024-65535", "connect";
   permission java.net.SocketPermission
      "webhost:80", "connect";
};
```

Pada HaiClient2

```
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

public class HaiClient2 {
    public static void main(String[] args) {
        System.setSecurityManager(new java.rmi.RMISecurityManager());
        try {
            Hai helloClient = (Hai)
               Naming.lookup("rmi://localhost/Hai");
        helloClient.setText("ini dari helloClient");
        System.out.println(helloClient.getText());
        }
        catch (MalformedURLException murle) {
            System.out.println();
            System.out.println("MalformedURLException");
            System.out.println(murle);
        }
        catch (RemoteException re) {
            System.out.println();
            System.out.println("RemoteException");
            System.out.println(re);
        }
        catch (NotBoundException nbe) {
            System.out.println();
            System.out.println("NotBoundException");
            System.out.println(nbe);
        }
    }
}
```

Buat hai.policy

```
grant {
   permission java.net.SocketPermission “*:1024-65535”, “connect,
   accept”;
   permission java.net.SocketPermission “*:80”, “connect”;
};
```

Pendistribusian

- Dari sisi Server
  - Remote Service Interface
  - Remote Service Implementation
  - Stub Remote Service Implementation
  - Semua server class
- Contoh:
  - Hai.class
  - HaiImpl.class
  - HaiImpl_Stub.class
  - HaiServer.class

Pendistribusian

- Dari sisi Client
  - Remote Service Interface
  - File policy
  - Semua client class
- Contoh:
  - Hai.class
  - HaiClient2.class
  - Hai.policy

Lalu…

- Kopikan ke folder rmi HTTPServer:
  - Hai.class
  - HaiImpl.class
  - HaiImpl_Stub.class
  - HaiServer.class

- Jalankan rmiregistry di server
- Jalankan Server:
  - java –Djava.rmi.server.codebase=http://localhost/rmi/ -Djava.rmi.server.hostname=localhost HaiServer
- Jalankan Client:
  - java -Djava.security.policy=hai.policy HaiClient2 localhost

Keterbatasan dari RMI

- Java-only
- Uses TCP, not UDP
- At least two sockets per connection
- Untested for huge loads

Sun vs. Microsoft

- RMI is not shipped as part of Microsoft's products
- RMI will still work in applications
  - include java.rmi.* class files in your classpath
  - download rmi.zip from ftp.microsoft.com
- RMI will work in applets
  - include java.rmi.* class files (or rmi.zip) in your codebase
  - IE4: only if they're signed
  - extra download time

Praktikum