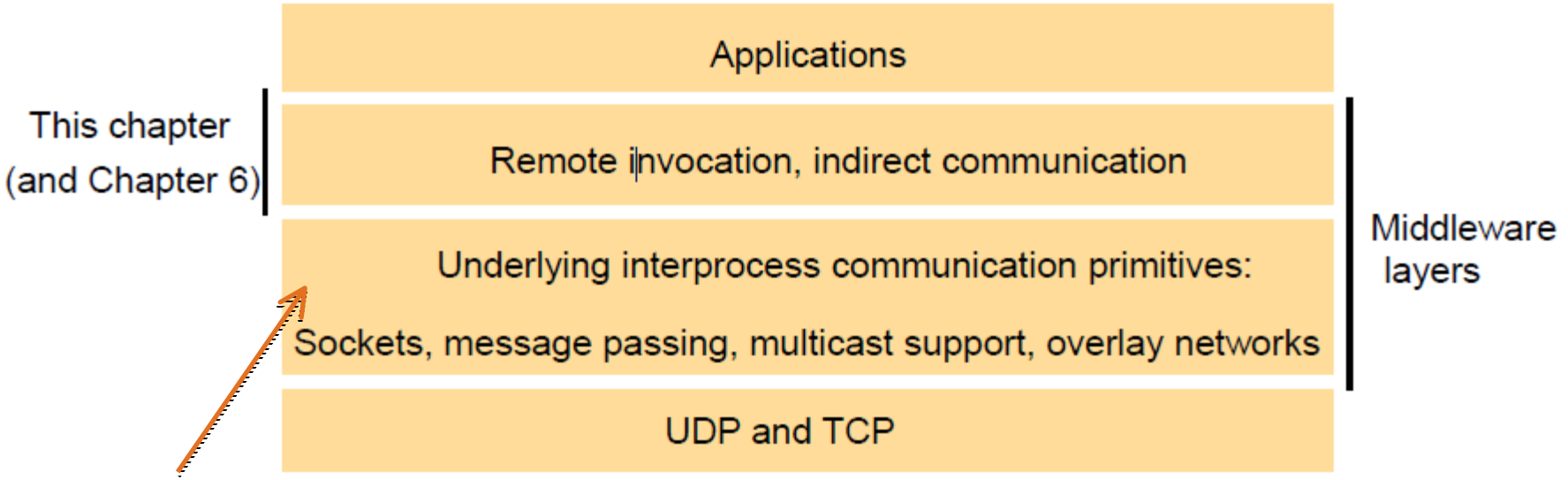
Fakultas Informatika
School of Computing
Telkom University

# CSG3L3
# SISTEM TERDISTRIBUSI

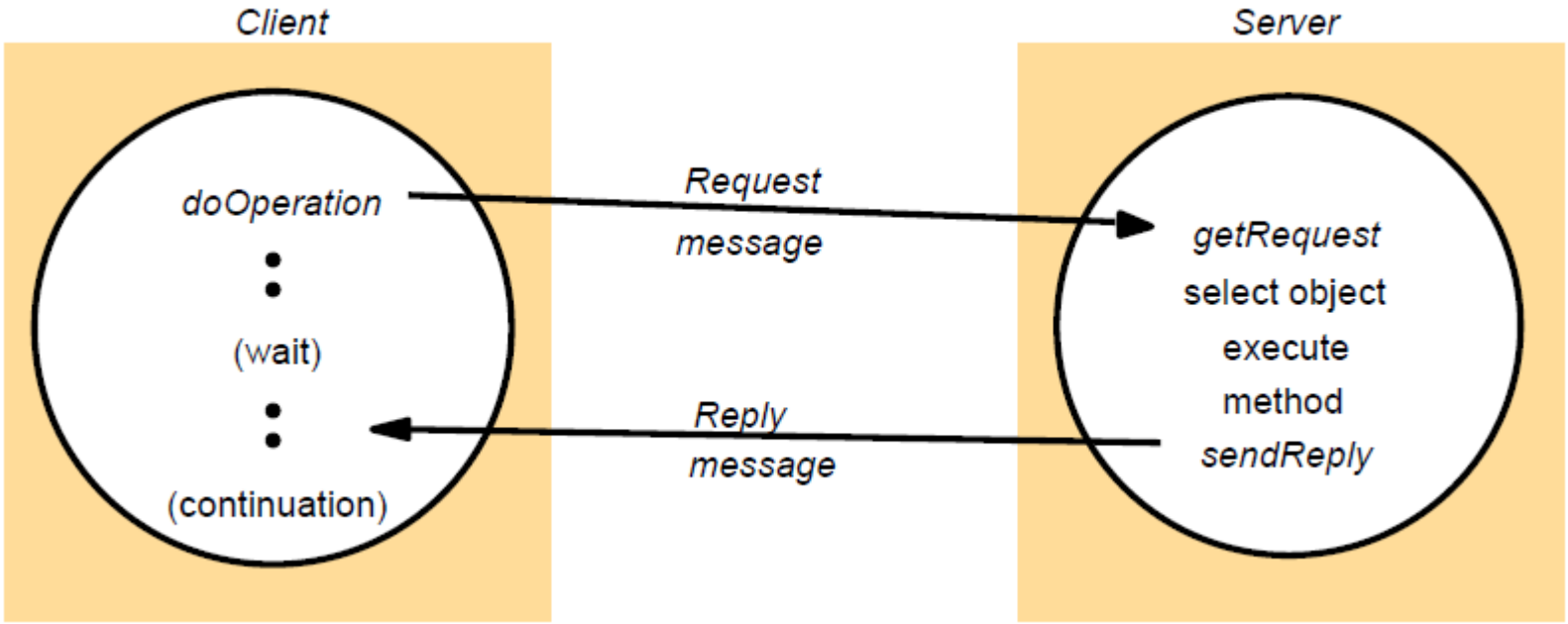
Topik 5 : *Remote Invocation*

# Pembahasan masih berkaitan dengan *Middleware Layer*




*Chapter* sebelumnya

# Disebut apakah mekanisme pada gambar di bawah ini?




**Request-Reply !**

# *Method* / Operasi yang ada pada Protokol Komunikasi *Request-Reply*

```
public byte[] doOperation(RemoteRef s, int operationId, byte[]
args) {

    /*

        * Method ini mengirimkan sebuah request /
        * pesan kepada remote server dan mengembalikan
        * (return) sebuah nilai (reply). Parameter berupa server
        * tujuan, operasi yang akan dieksekusi (di-invoke),
        * dan parameter yang dibutuhkan dalam operasi
        * tersebut.

    */

}
```

# *Method* / Operasi yang ada pada Protokol Komunikasi request-Reply (cont'd)

```
public byte[] getRequest() {

    /* Mengakuisisi request dari klien via port tertentu */

}

Public void sendReply(byte[] reply, InetAddress
clientHost, int clientPort) {

    /* Mengirimkan message balasan kepada klien tujuan
       yang didefinisikan dari alamat jaringan dan port */

}
```

# Struktur *message* pada *Request-Reply*

| | |
|---|---|
| messageType | *int (0=Request, 1= Reply)* |
| requestId | *int* |
| remoteReference | *RemoteRef* |
| operationId | *int or Operation* |
| arguments | *array of bytes* |

# Protokol RPC

| Nama | Pesan dikirim oleh | | |
|---|---|---|---|
| | **Klien** | **Server** | **Klien** |
| **R** | Request | - | - |
| **RR** | Request | Reply | - |
| **RRA** | Request | Reply | ACK |

## Contoh format *message Request-Reply* pada HTTP

| *method* | *URL or pathname* | *HTTP version* | *headers* | *message body* |
|---|---|---|---|---|
| GET | //www.dcs.qmw.ac.uk/index.html | HTTP/ 1.1 | | |

**Request**

| *HTTP version* | *status code* | *reason* | *headers* | *message body* |
|---|---|---|---|---|
| HTTP/1.1 | 200 | OK | | resource data |

**Reply**

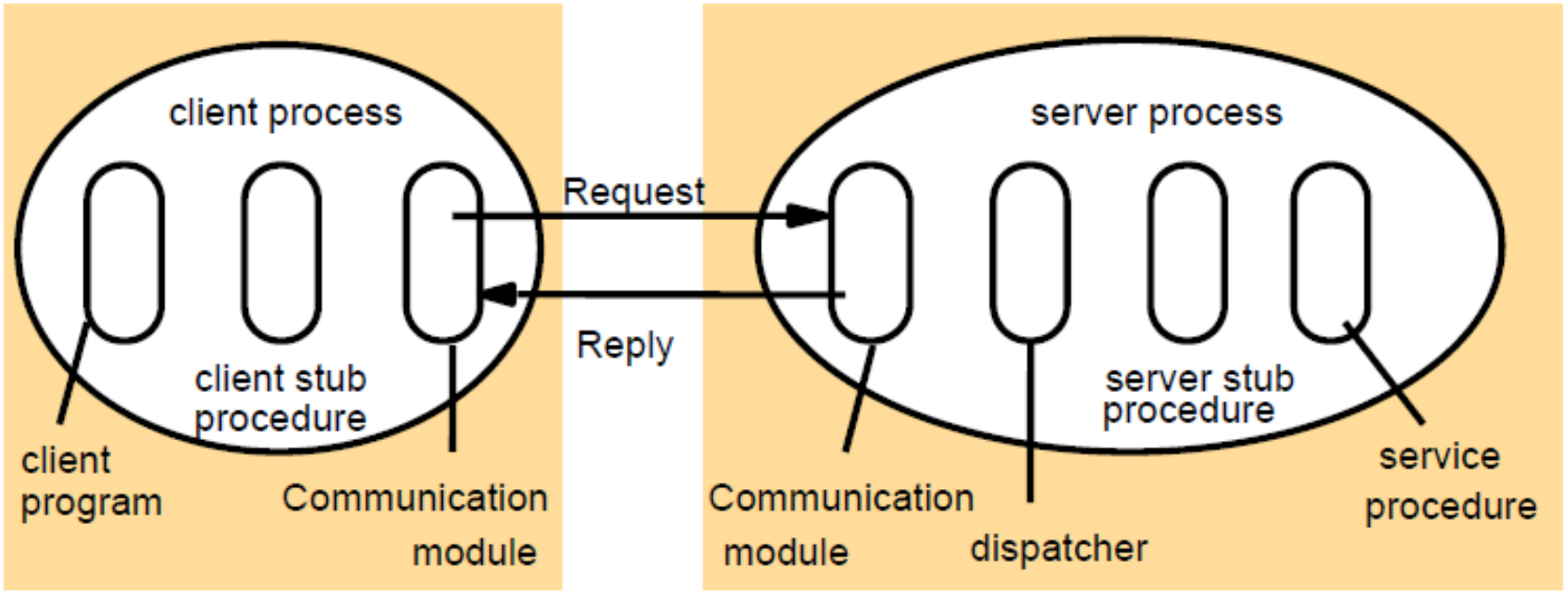
# Apa itu IDL? Gunanya? Perhatikan IDL CORBA berikut ini.

```
// In file Person.idl
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

# *"Call Semantic"*

| Pengukuran toleransi terhadap kesalahan | | | ***Call Semantic*** |
|---|---|---|---|
| **Ulang pengiriman *request*?** | **Filter terhadap duplikasi** | **Eksekusi prosedur atau pengiriman pesan *reply* ulang** | |
| Tidak | Tidak perlu | Tidak perlu | Mungkin |
| Ya | Tidak | Eksekusi ulang prosedur | Minimal satu kali |
| Ya | Ya | Pengiriman ulang pesan *reply* | Maksimal satu kali |

# Bisakah Anda menjelaskan bagaimana urutan eksekusi pada RPC? Berikut ini adalah ilustrasinya

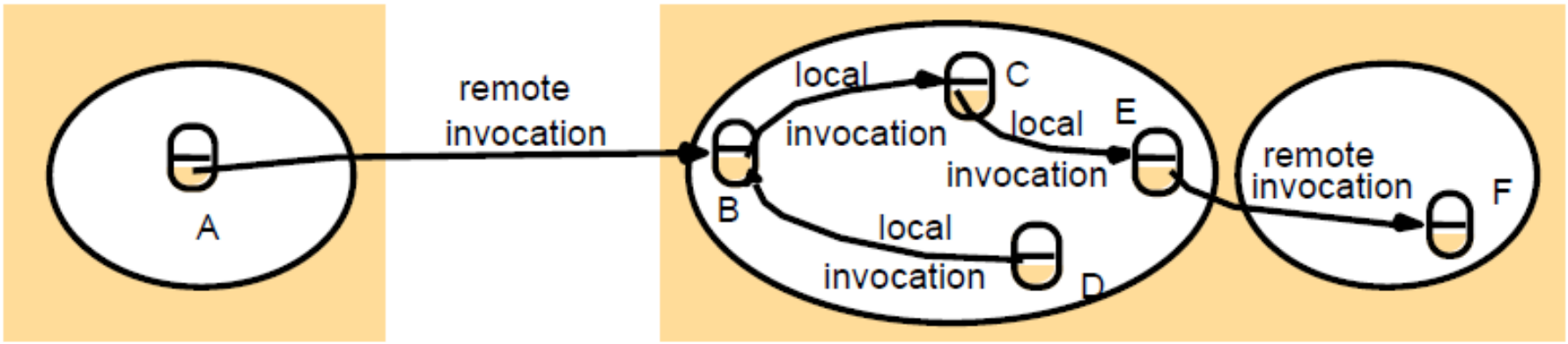


# Contoh *File Interface* di Sun XDR

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
```

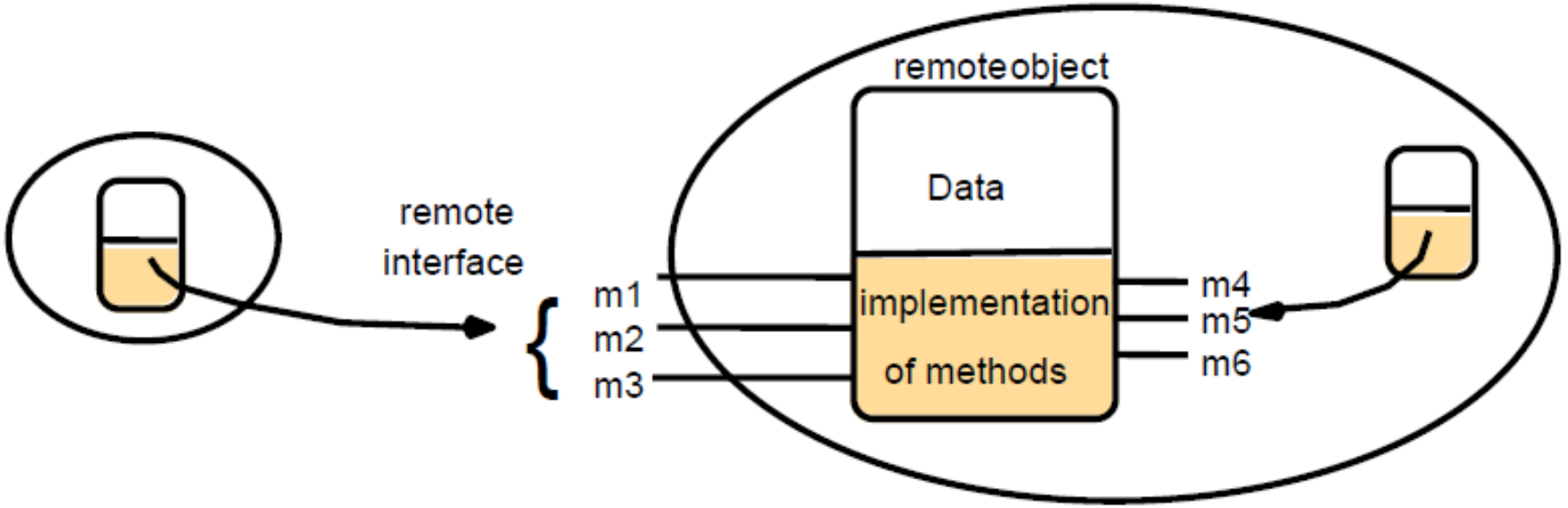
```
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
  version VERSION {
    void WRITE(writeargs)=1;    1
    Data READ(readargs)=2;      2
  }=2;
} = 9999;
```

# Apa yang bisa Anda simpulkan dari gambar di bawah ini?

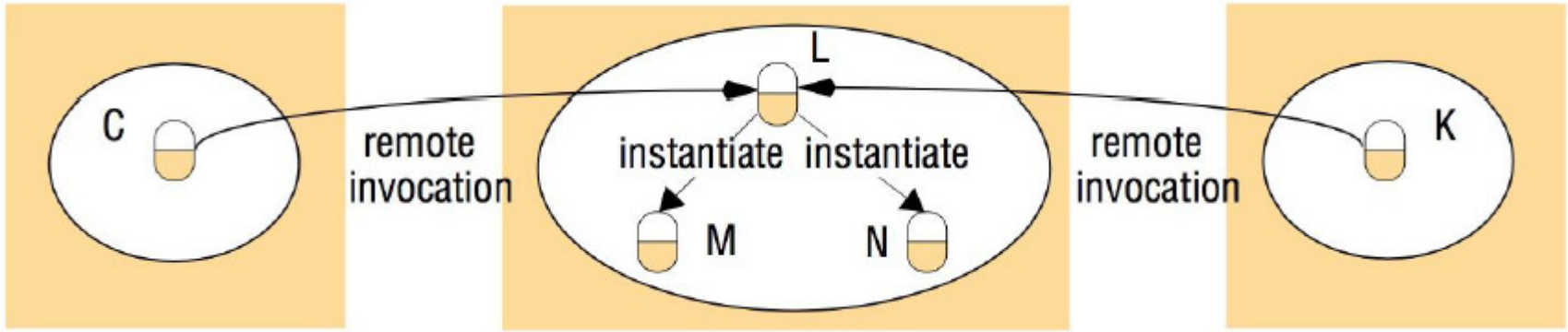



**Server men-*treat* (memperlakukan) *remote invocation* sebagai *local invocation***

# Pemanggilan *remote procedure* membutuhkan sebuah *remote object* yang masing-masing memiliki *remote interface* tersendiri

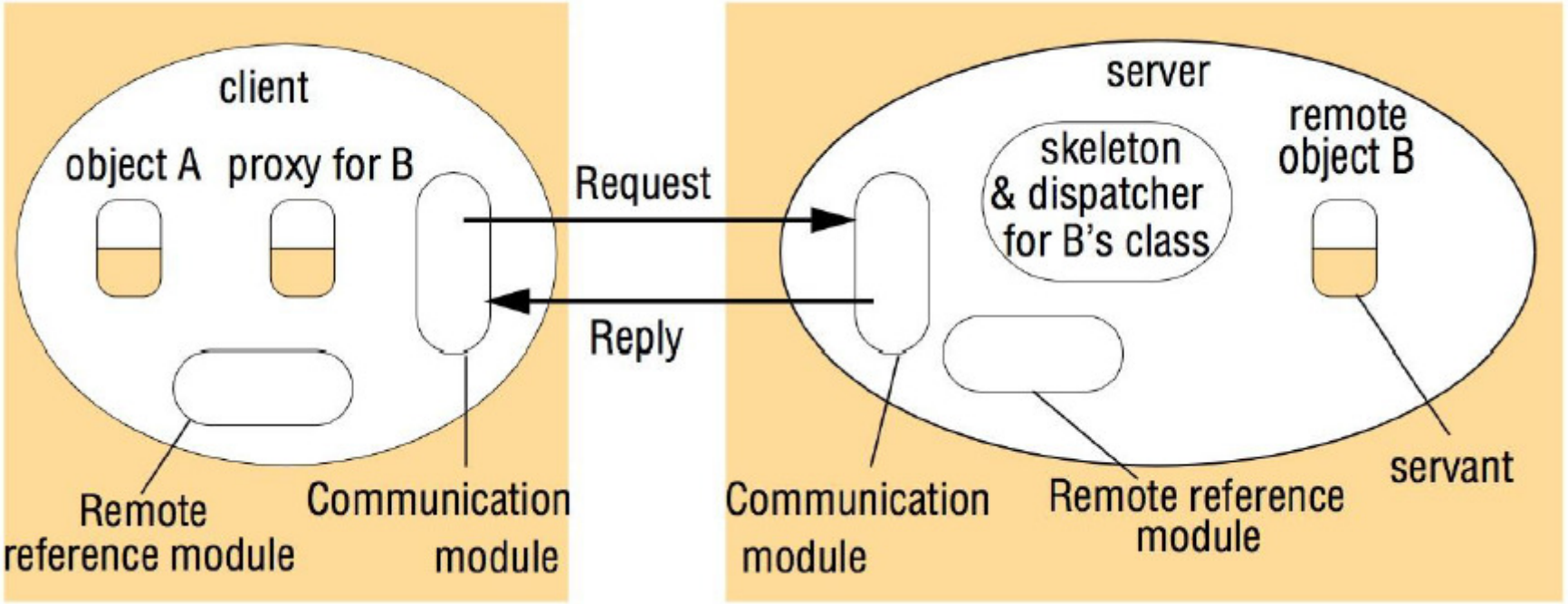


# Pembentukan *remote object* tergantung pada prosedur apa yang akan dieksekusi




Proses C menghasilkan *remote object* M dan proses K menghasilkan *remote object* N yang dihasilkan oleh proses L

# Peran *proxy* dan skeleton pada RMI




Cari referensi lain berkaitan dengan ilustrasi gambar di atas.

# *Remote Interface* pada Bahasa Pemrograman Java

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject  getAllState() throws RemoteException;   1
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException;
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```

# Penamaan *class* di *registry* Java MRI

```
void rebind(String name, Remote obj) {

    /* Method ini digunakan oleh server untuk
    mendaftarkan ID dari sebuah remote object
    berdasarkan nama */

}

void bind(String name, Remote obj) {

    /* Method ini merupakan alternatif yang digunakan server
    untuk mendaftarkan remote object berdasarkan nama,
    tetapi jika nama tersebut telah terdaftar maka terjadi
    Exception */

}
```

# Penamaan *class* di *registry* Java MRI (cont'd)

**void** unbind(**String** name, Remote obj) {

    /* Menghapus daftar *binding* berdasarkan nama objek */

}

Remote lookup(**String** name) {

    /* Sebuah fungsi yang digunakan klien untuk mencari
       sebuah *remote object* tertentu berdasarkan namanya */

}

**String**[] list();

Fungsi yang mengembalikan kumpulan nama *remote object* yang terdaftar pada *registry*.

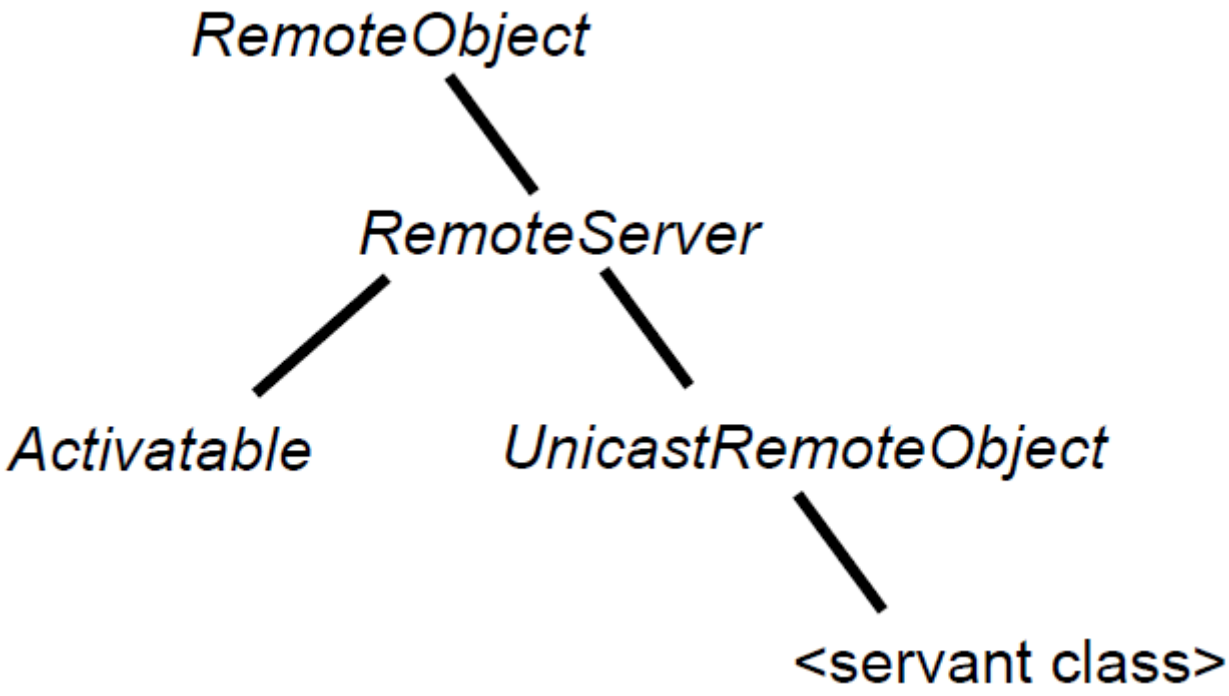
# Perhatikan dan cari tahu tujuan dari *code* berikut ini:

```
import java.rmi.*;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        try{
            ShapeList aShapeList = new ShapeListServant();
                Naming.rebind("Shape List", aShapeList );
            System.out.println("ShapeList server ready");
            }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());}
    }
}
```

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeLi
      private Vector theList;                    // contains the list of Shapes
      private int version;
     public ShapeListServant()throws RemoteException{...}
     public Shape newShape(GraphicalObject g) throws RemoteException {        1
          version++;
               Shape s = new ShapeServant( g, version);                       2
               theList.addElement(s);
               return s;
     }
     public  Vector allShapes()throws RemoteException{...}
     public int getVersion() throws RemoteException { ... }
}
```

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
    public static void main(String args[]){
     System.setSecurityManager(new RMISecurityManager());
     ShapeList aShapeList = null;
     try{
          aShapeList  = (ShapeList) Naming.lookup("//bruno.ShapeList")     ;
     1
          Vector sList = aShapeList.allShapes();                              2
     } catch(RemoteException e) {System.out.println(e.getMessage());
     }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
    }
}
```

# Hirarki *Class* Java RMI



# Ada Pertanyaan?

# Referensi

- Coulouris, G. F., Dollimore, J., & Kindberg, T. (2012). *Distributed Systems: Concepts and Design 5th Edition*. London: Pearson Education.

Fakultas Informatika
School of Computing
Telkom University

# THANK YOU