Konzepte objektorientierter Programmierung

Prof. Dr. Peter Müller

Werner Dietl

Software Component Technology

Exercises 11: Distribution

Wintersemester 03/04

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Chat Application

- One server that:
  - accepts registrations from clients
  - broadcasts messages received from one client to all registered clients

- Multiple clients that:
  - register with a server
  - send a message to the server to broadcast it
  - receive messages from the server and output them

ChatServer Interface

```
package common;

import java.rmi.*;

public interface ChatServer extends Remote {

    void register( ChatClient c )
         throws RemoteException;

    void broadcast( String s )
         throws RemoteException;
}
```

ChatClient Interface

```
package common;

import java.rmi.*;

public interface ChatClient extends Remote {

    void receive( String s )
        throws RemoteException;

}
```

ChatClient Implementation

```
public class ChatClientImpl
  extends UnicastRemoteObject
  implements ChatClient, Runnable {

    private ChatServer mycs;

    public ChatClientImpl( ChatServer cs )
      throws java.rmi.RemoteException {
        mycs = cs;
        mycs.register( this );
    }

    public synchronized void receive(String s)
      throws java.rmi.RemoteException {
        System.out.println("Message: " + s );
    }
```

ChatClient Run Method

```
public void run() {
    BufferedReader ir = new BufferedReader(
            new InputStreamReader(System.in));
    String msg;

    while( true ) {
      try {
          msg = ir.readLine();
          mycs.broadcast( msg );
      } catch( Exception e ) {
          System.err.println("Problem...");
      }
    }
}
```

ChatClient Main Method

```
public static void main( String[] args ) {
  String url = "rmi://gem/ChatServer";

  try {
      ChatServer cs =
            (ChatServer) Naming.lookup(url);
      new Thread(
            new ChatClientImpl(cs)).start();
  } catch( Exception e ) {
      System.err.println("Problem...");
      e.printStackTrace();
  }

}
```

ChatServer Implementation

```
public class ChatServerImpl extends
  UnicastRemoteObject implements ChatServer {

  private LinkedList myclients;

  public ChatServerImpl()
    throws java.rmi.RemoteException {
      myclients = new LinkedList();
  }

  public synchronized void register(ChatClient c)
    throws java.rmi.RemoteException {
      myclients.add( c );
  }
```

ChatServer Broadcast

```
public synchronized void broadcast( String s )
   throws java.rmi.RemoteException {

   ListIterator it = myclients.listIterator(0);
   ChatClient c;

   while( it.hasNext() ) {
      c = (ChatClient) it.next();

      c.receive( s );
   }
}
```

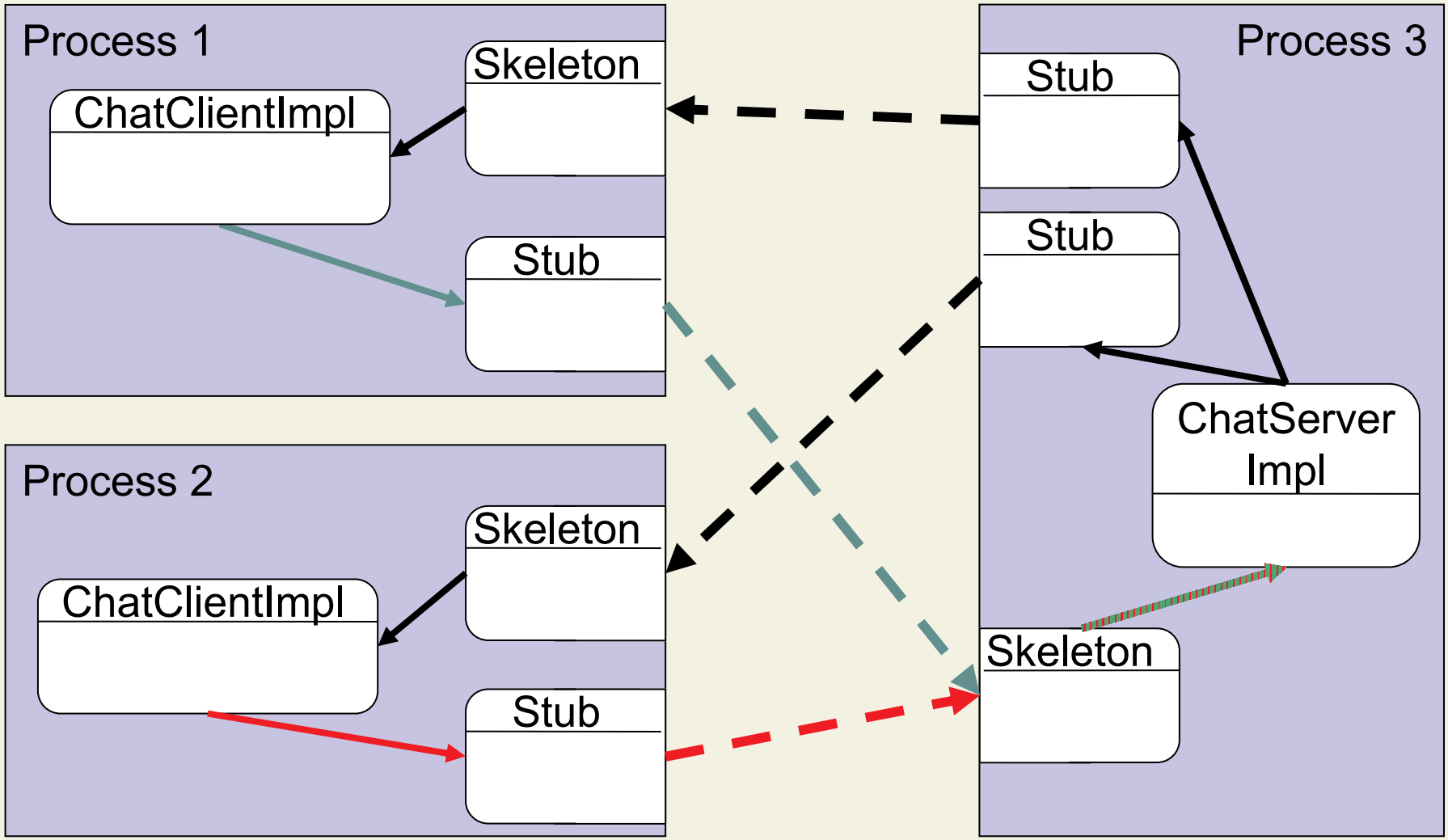
ChatServer Main Method

```
public static void main( String[] args ) {
   try {

      Naming.rebind( "ChatServer",
           new ChatServerImpl( ) );

   } catch( Exception ex ) {
       System.err.println("Problem…");
       ex.printStackTrace();
   }
}
```

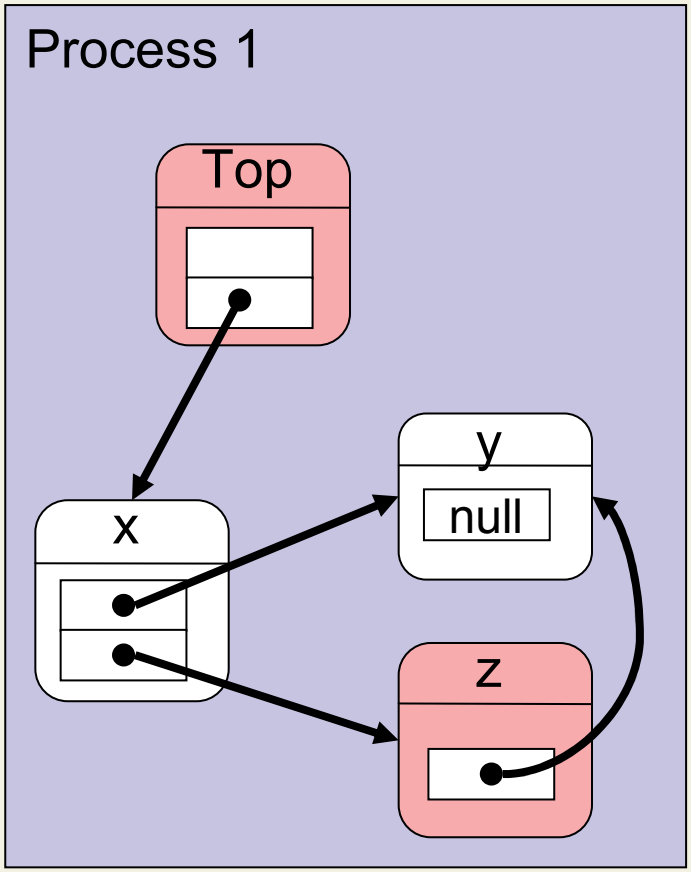
Process Interaction



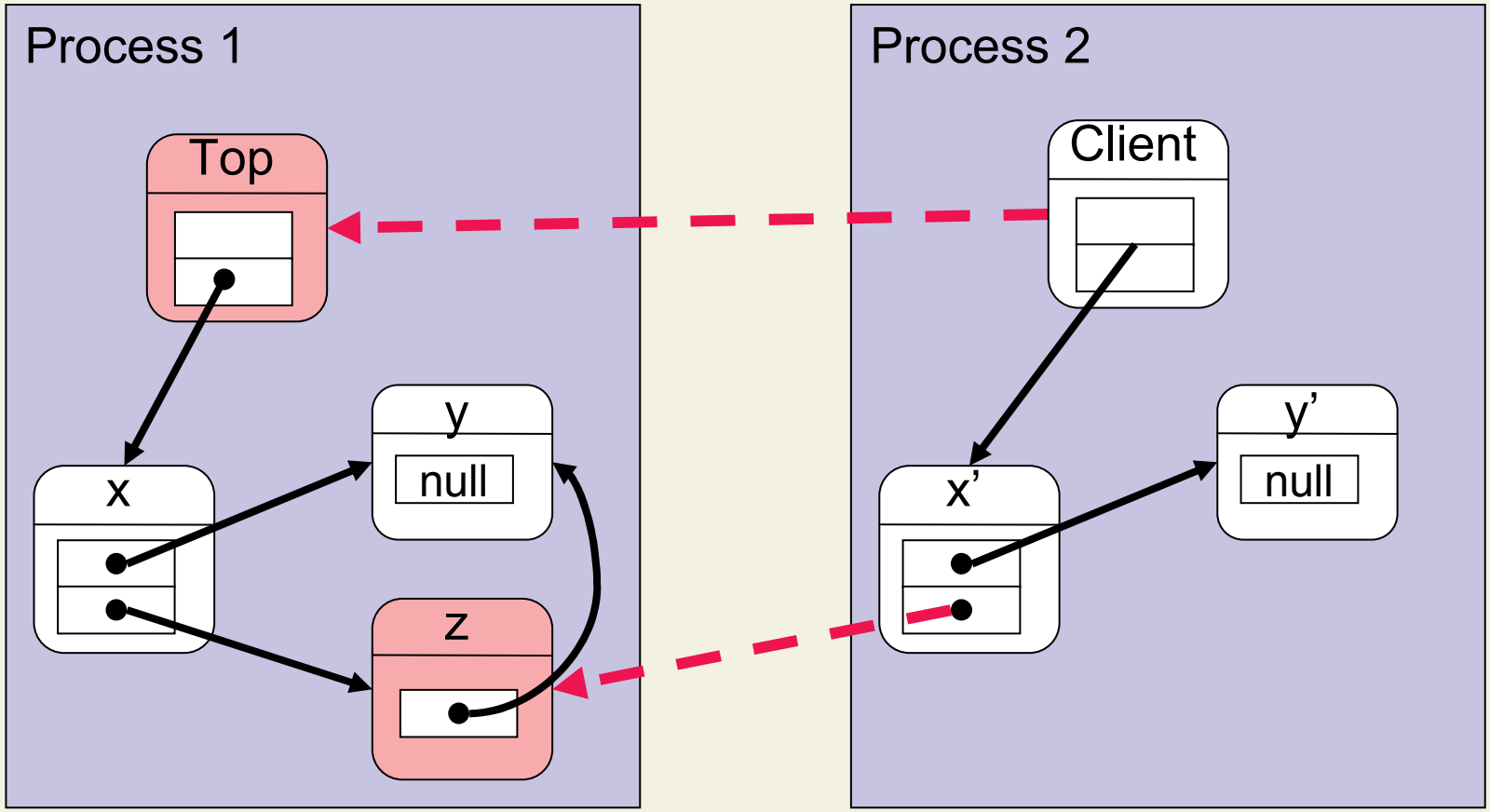
Consistency of Object Structures

- Client and Server that share an object structure
- The client modifies an object through one path
- And then reads the modified value through a different path
- The two values don’t match…
- Why?

Object Layout on the Server



Interaction with the Client



Top Interface and Server Main Method

```
public interface Top extends Remote {

    X getX() throws RemoteException;

}

public class Server {

    public static void main( String[] args ) {
        Top mytop = new TopImpl();
        Naming.rebind( "Top", mytop );
    }

}
```

Class X

```
public class X
  implements java.io.Serializable {
    private Y myy;
    private Z myz;

    public X( Y y, Z z ) { myy = y; myz = z; }

    public String getValue() {
       return myy.value; }

    public void setValue( String s ) {
       myy.value = s; }

    public Z getZ() { return myz; }
}
```

Class Y and Interface Z

```
public class Y
   implements java.io.Serializable {
      public String value;

      public Y( String v ) {
           value = v;
      }
}

public interface Z extends Remote {
    String getValue() throws RemoteException;
}
```

Implementation of Top

```
public class TopImpl extends
        UnicastRemoteObject implements Top {
  private X x;

  public TopImpl() throws RemoteException {
     Y y = new Y( "Y server initial" );
     Z z = new ZImpl( y );
     x = new X( y, z );
  }

  public X getX() throws RemoteException {
     return x;
  }
}
```

Z Implementation

```
public class ZImpl extends
        UnicastRemoteObject implements Z {
  private Y myy;

  public ZImpl( Y y )
        throws java.rmi.RemoteException {
     myy = y;
  }

  public String getValue()
        throws java.rmi.RemoteException {
     return myy.value;
  }
}
```

Client Setup

```
public class Client {
  public static void main( String[] args ) {
     String url = "rmi://gem/Top";

     Top t = null;
     X thex = null;

     try {
        t = (Top) Naming.lookup( url );
        thex = t.getX();
     } catch( Exception e ) {
        e.printStackTrace();
     }

...
```

Client Main Program

```
println("x.getY(): " + thex.getValue());
println("x.getZ().getY(): " +
  thex.getZ().getValue());

println("Calling x.setValue(...)... ");
thex.setValue("New Client Value!");

println("x.getY(): " + thex.getValue());
println("x.getZ().getY(): " +
  thex.getZ().getValue());
```

Interaction with the Client

```
x.getY(): Y server initial
x.getZ().getY(): Y server
   initial
```

```
x.getY(): New Client Value!
x.getZ().getY(): Y server
   initial
```