Concepts of Object-Oriented Programming

Prof. Dr. Peter Müller

Chair of Programming Methodology

Exercises 11: Concurrency and Distribution

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://localhost/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 );
  }
```

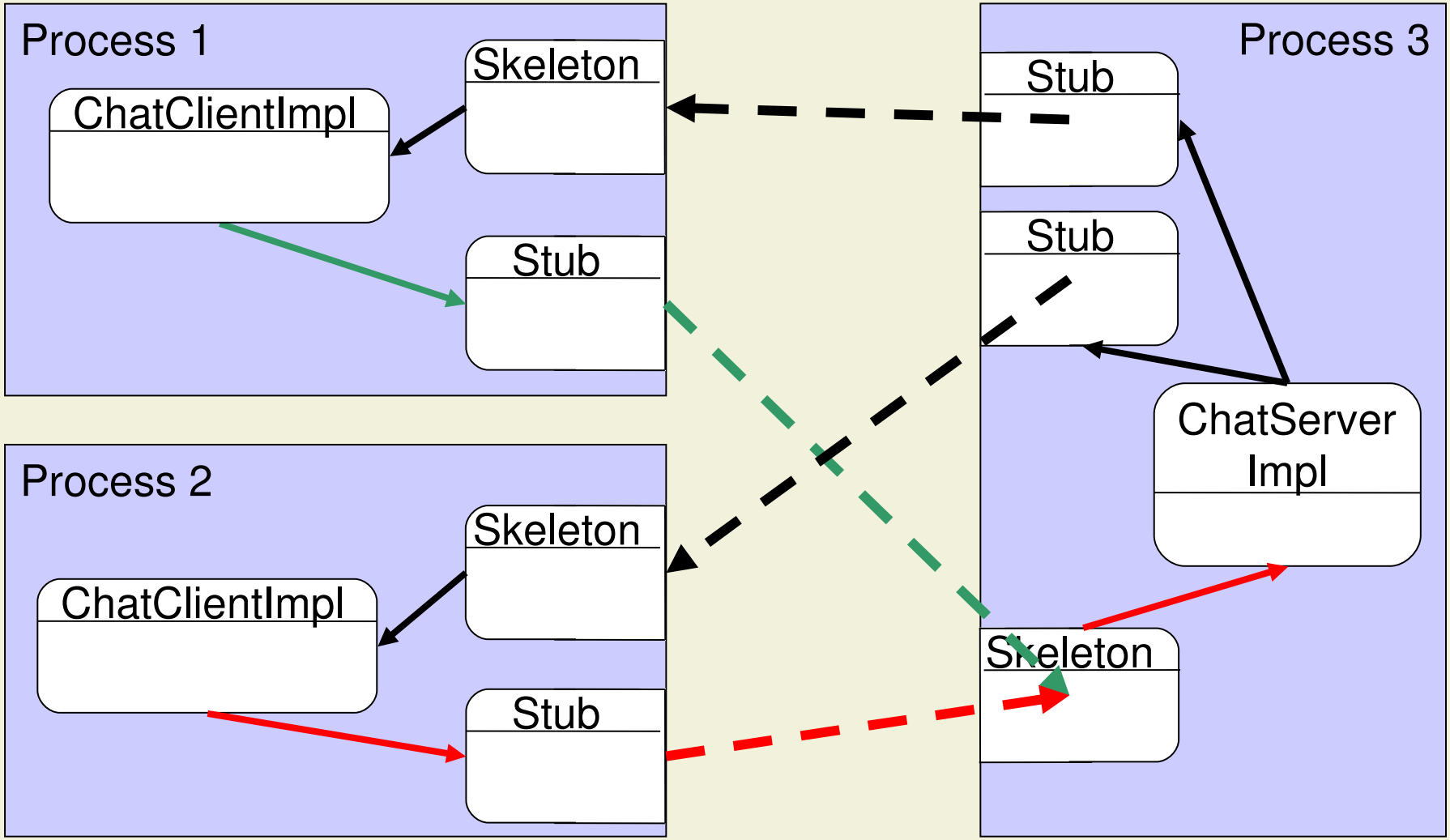
ChatServer Broadcast

```
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



Reentrant locks

- Java intrinsic locks are reentrant.
- A thread can acquire a lock it already owns.
- Non-reentrant locks require special handling to avoid deadlock.

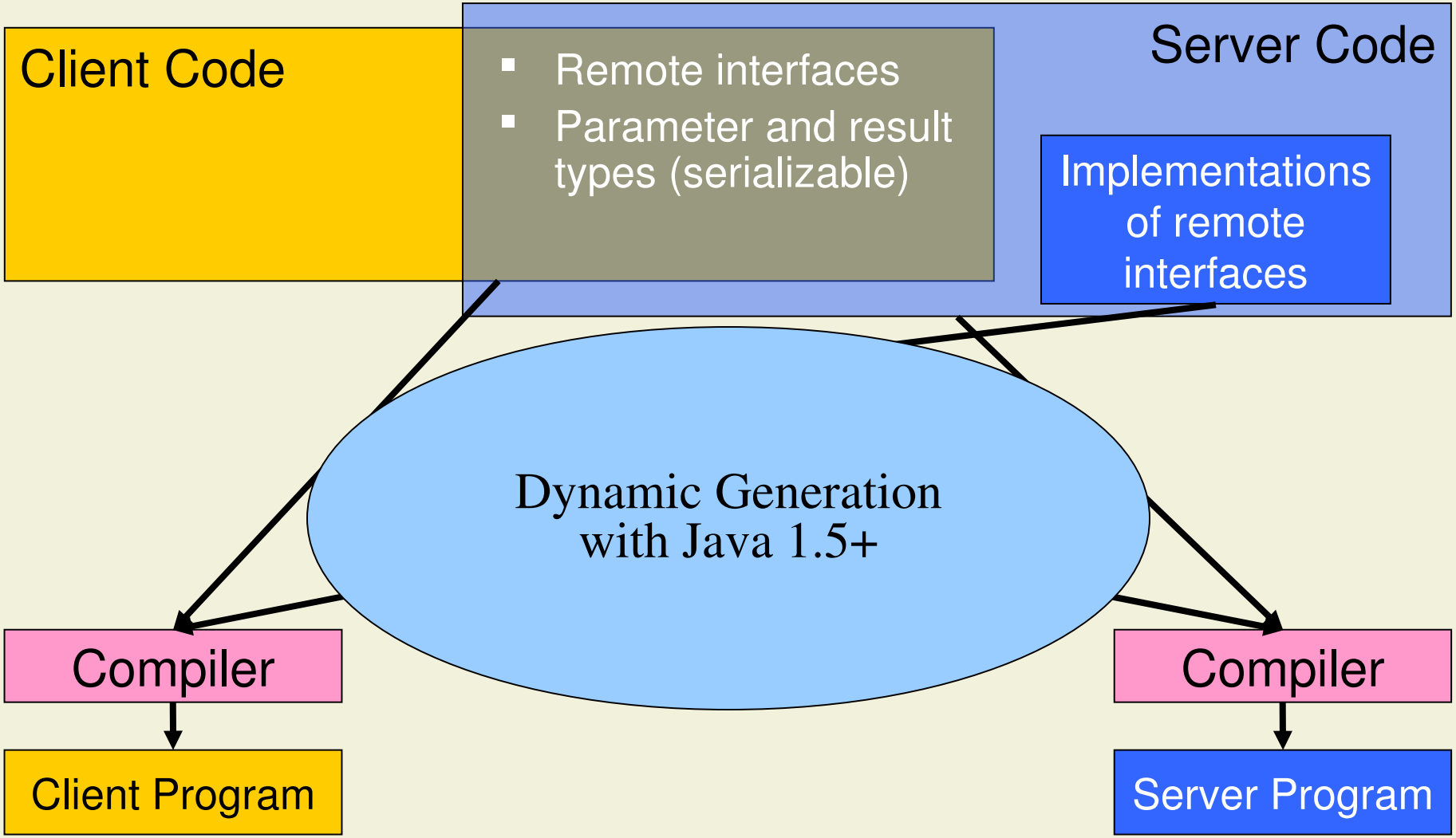
```
public synchronized void m() {
   // this is locked
   n (); }

public synchronized void n() {
   // this is already locked
   // Deadlock!
}
```

Non-Reentrant locks

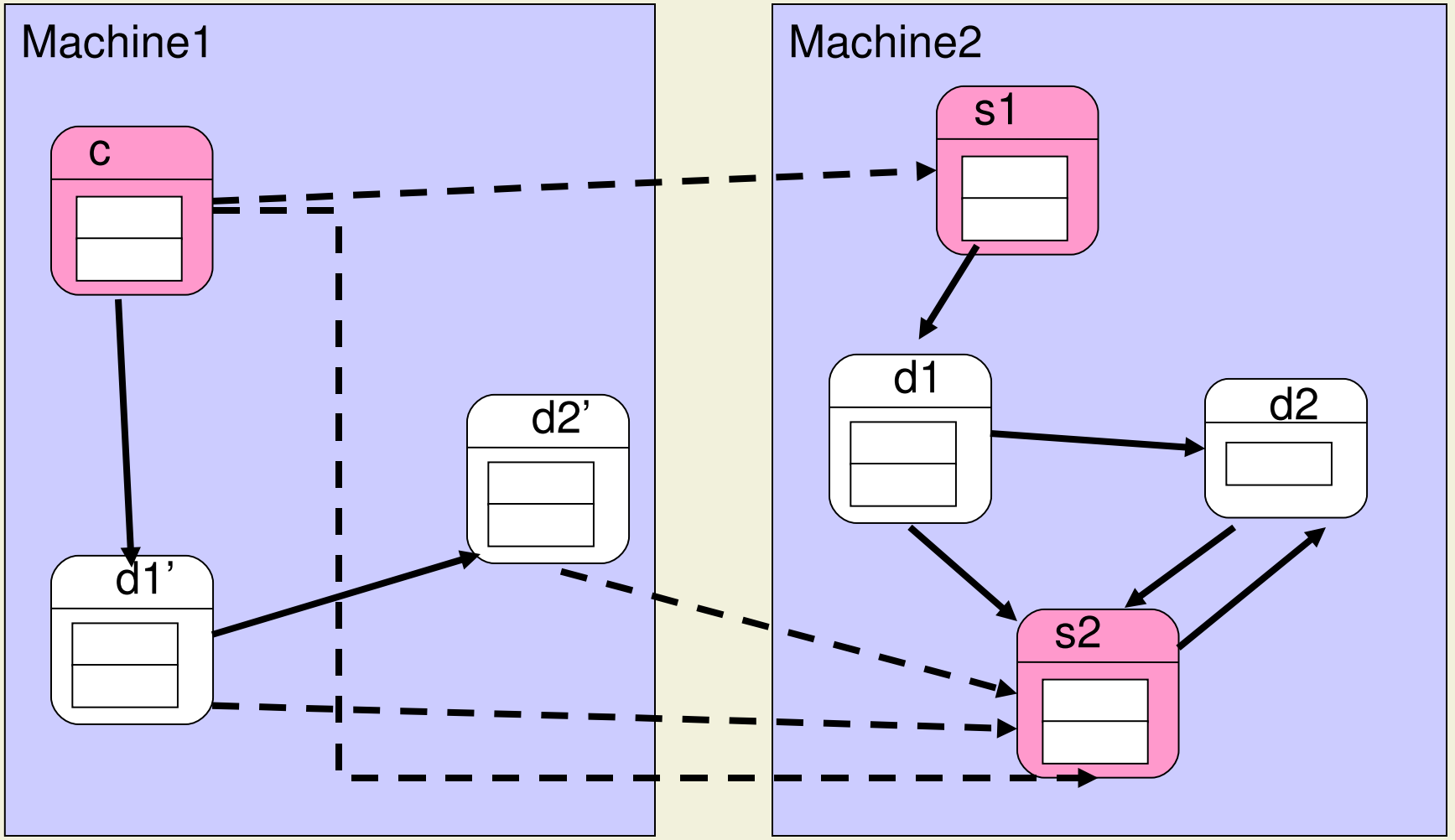
- Programmer must take care

```
public synchronized void m() {
   // this is locked
   _m (); }

private void _m() {
   // this is already locked
   _n (); }

private void _n() {
   ...
   _m (); }
```

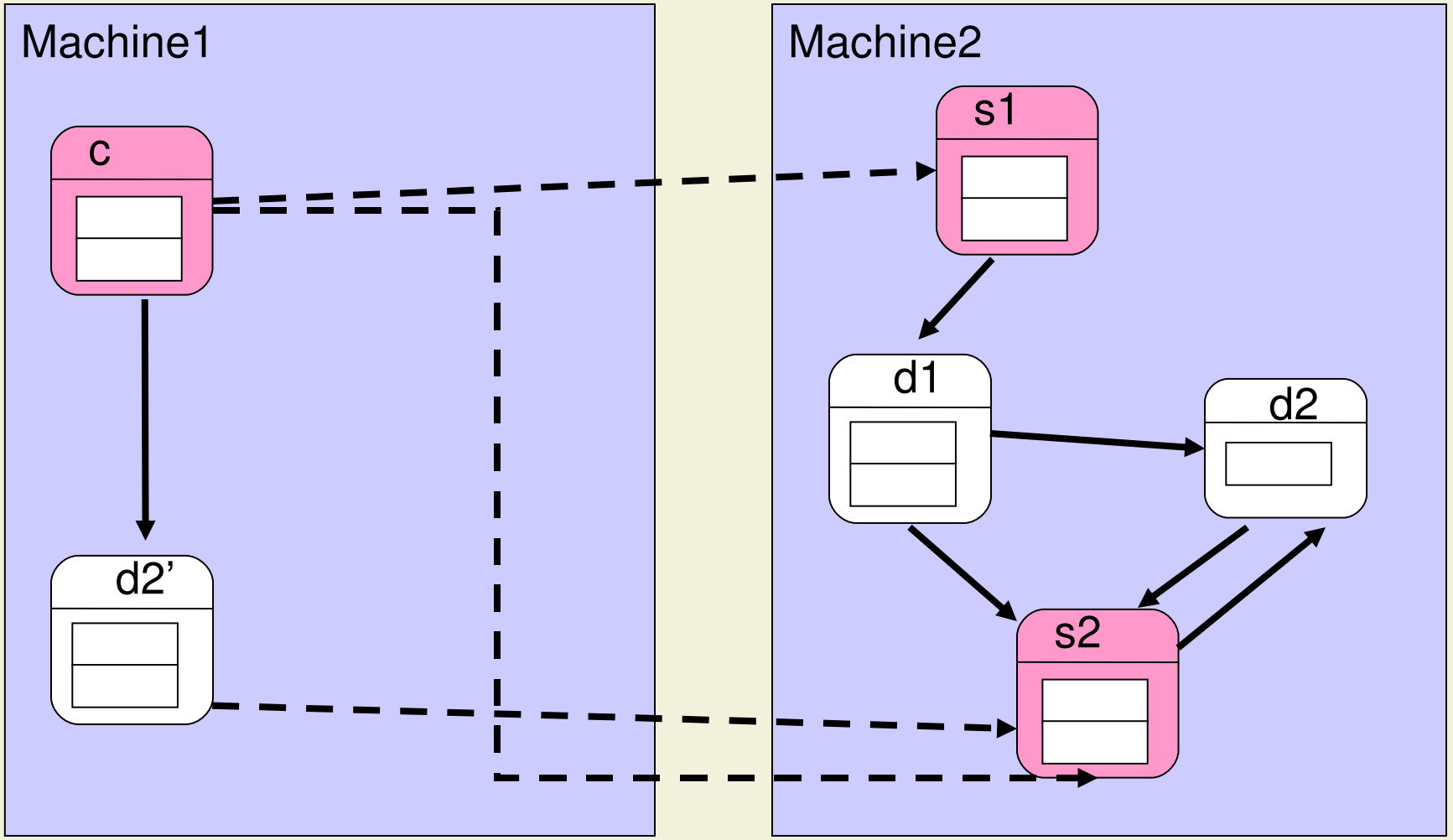
Programming with Remote Objects



After s1.getData()



After s2.getData()



Questions?