Distributed Systems Labs

Contact Information:

Groups A-D:
C3, CII, SoC
D3, IT4

Traian Pop
Email: trapo@ida.liu.se
Telephone: 28 19 70
Office: B building, 3D:437

Groups E-J:
D3, IT4
others

Alexandru Andrei
Email: alean@ida.liu.se
Telephone: 28 26 98
Office: B building, 3D:439

Labs Organization

- 8 lab groups
- 14 hours / student (supervised)
- 7 labs
- 5 lab assignments
- You get 1 point

- Home page with the lab material:
  `http://www.ida.liu.se/~TDDB37/labs`

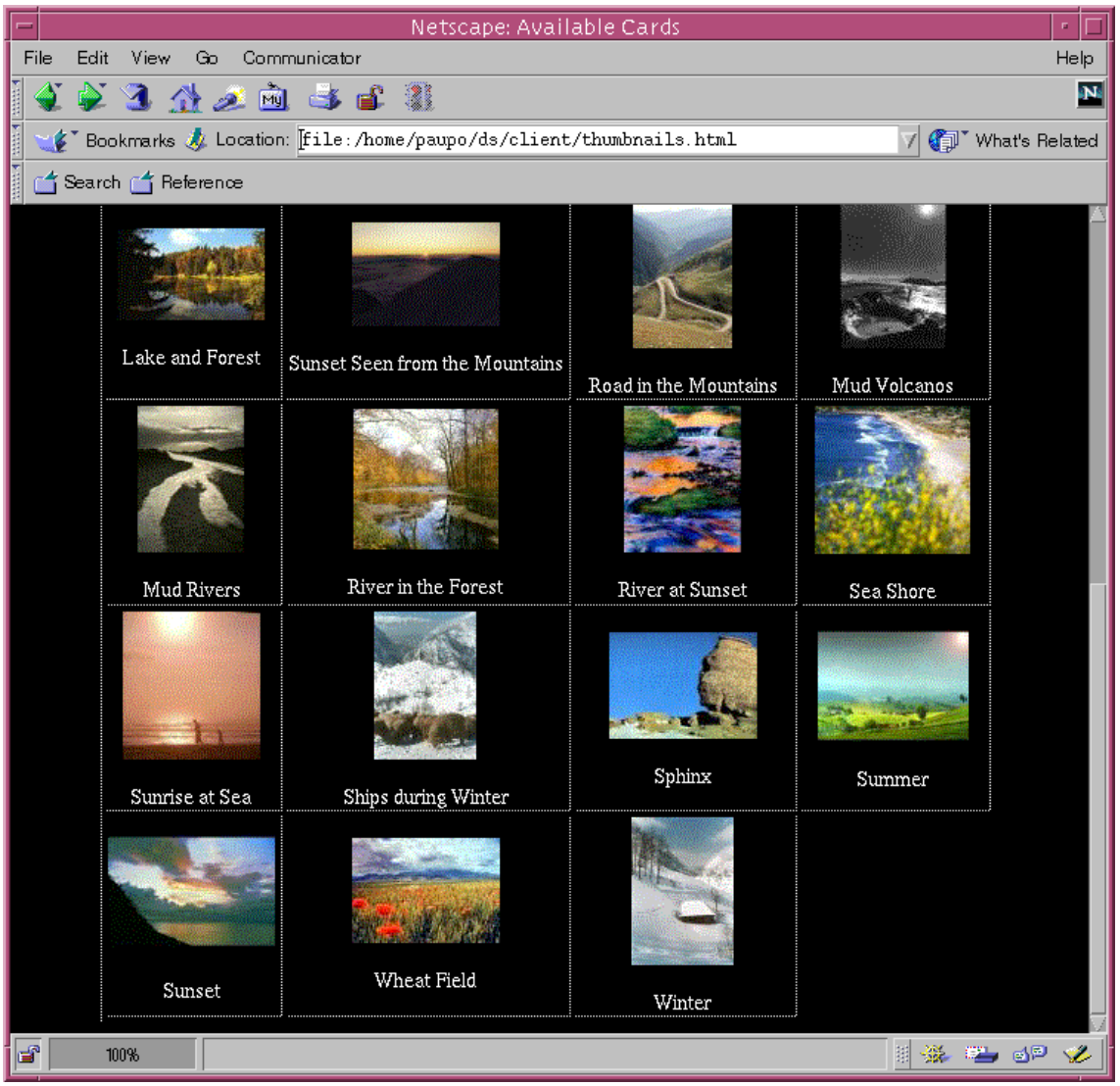
- Lab registration:
  `http://www.ida.liu.se/webreg`

- **Deadlines**
  - Signing up for the lab groups: February 10th
  - Handing in the lab assignments: **two weeks after the exam**

Goals

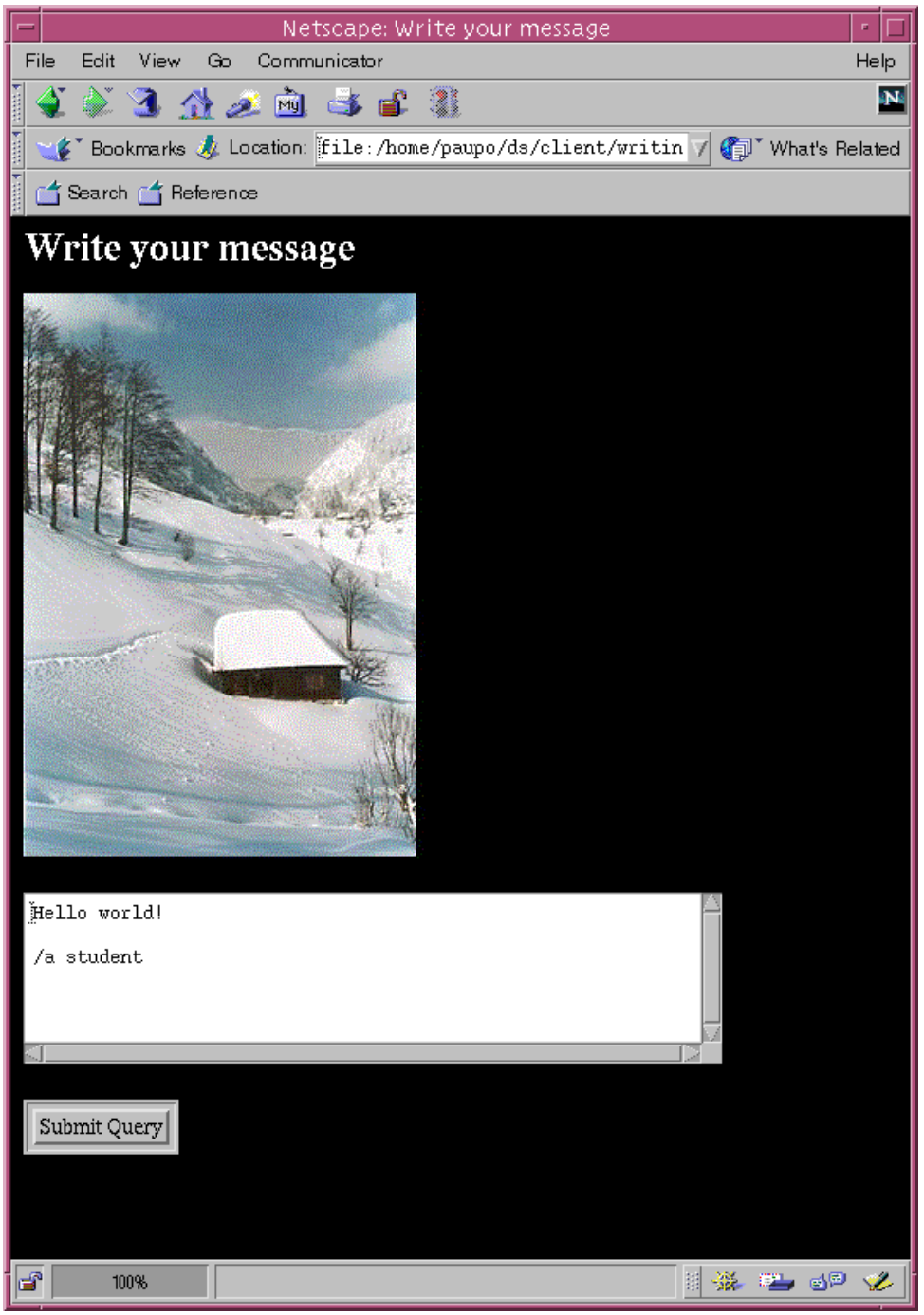
- To get 1 point :)
- Obtain knowledge related to the implementation of distributed systems using various methodologies and tools.
- Should be able to decide what methods and tools to use for a particular project involving the implementation of a distributed system.

Distributed Application: Electronic Postcards



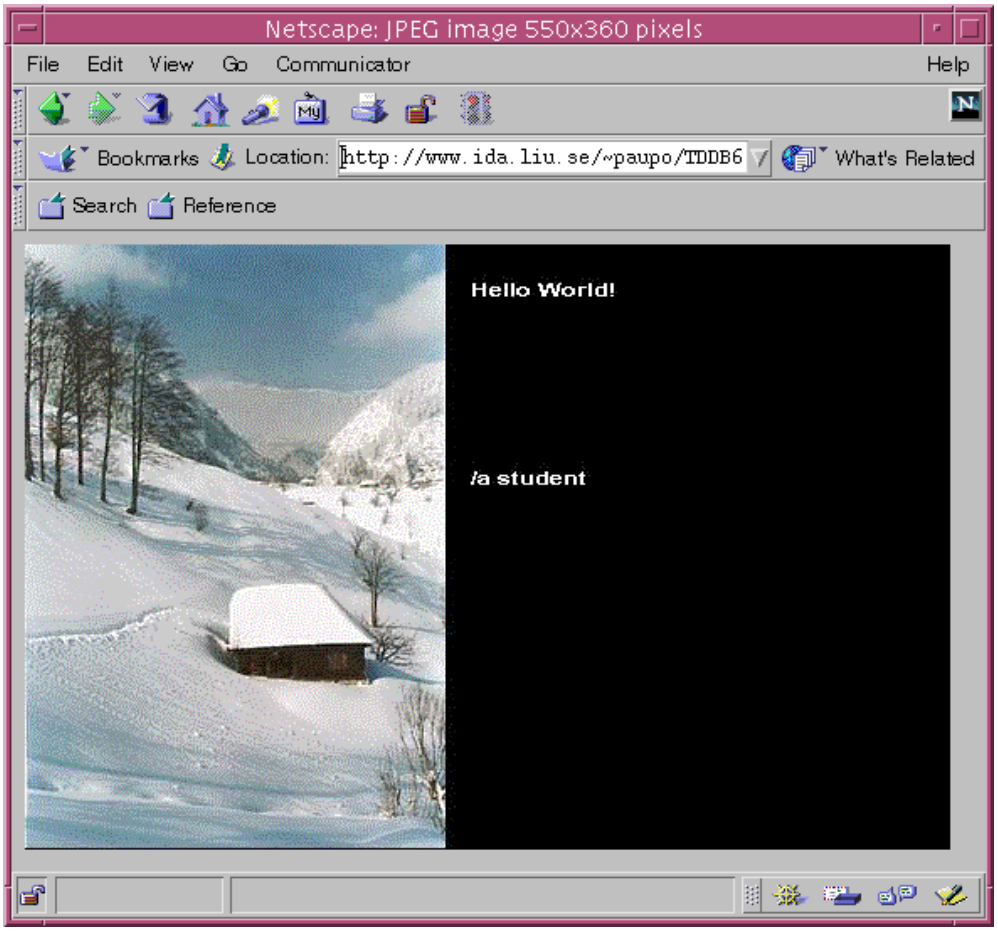
- Choosing a card to send.

Distributed Application: Electronic Postcards (Cont'd)



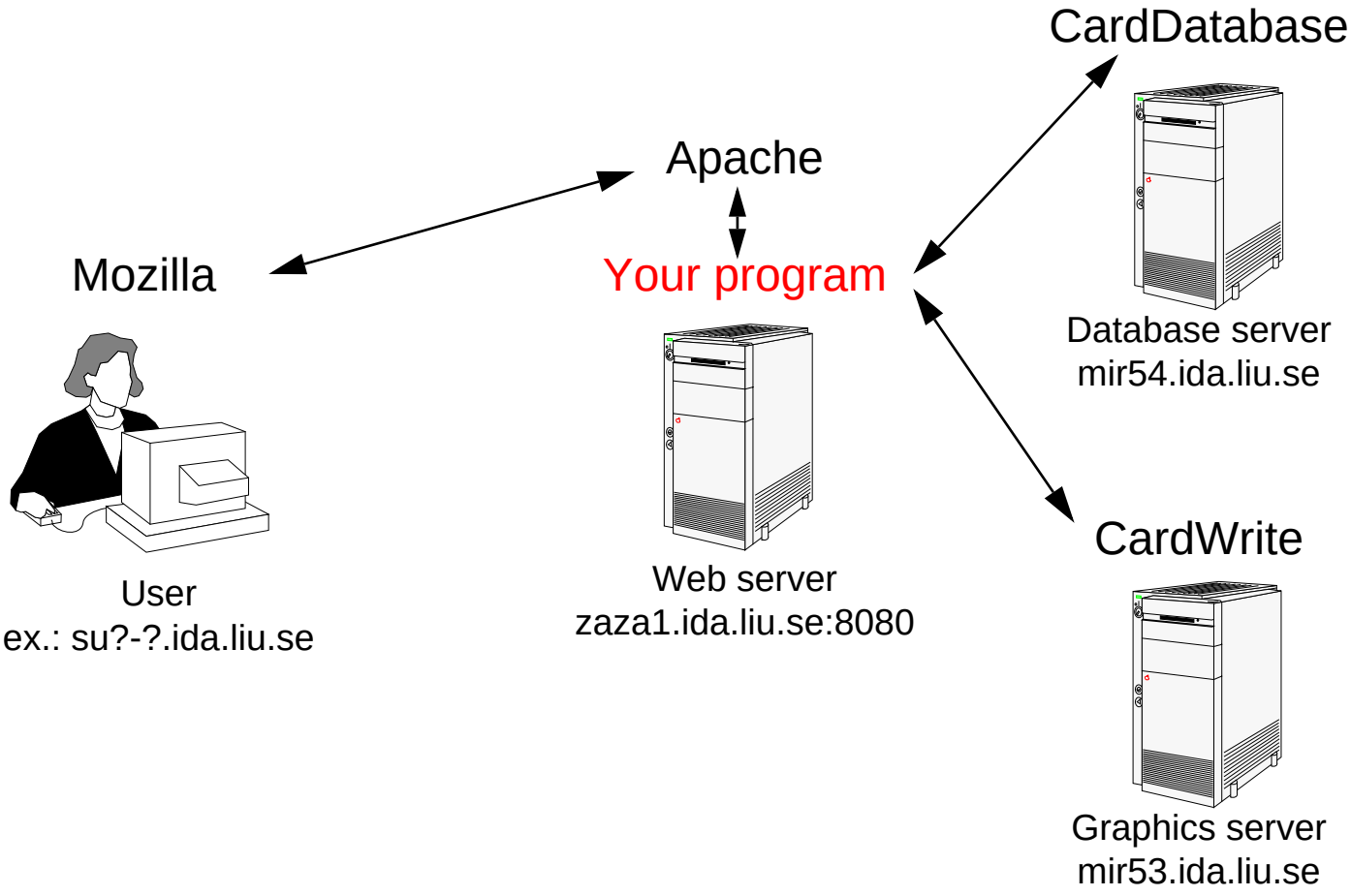
- Writing your message.

Distributed Application: Electronic Postcards (Cont'd)



- An e-mail notification is sent to the recipient.
- The card stays for a while on the web server.
- The message is embedded in the image so that it can be downloaded and stored.

The Architecture of the Application



About the Existing Components

- Web browser: Mozilla, Netscape, Internet Explorer, etc.
  - runs on the user's machine (your machine)

- Web server: Apache
  - number one web server on the Internet: 67.38% (January 2004)
    (source **`http://www.netcraft.com/Survey/`**)
  - free
  - runs on **`zaza1.ida.liu.se:8080`**
  - configured to take the documents from **`/home/<user>/TDDB37/`**

About the Existing Components (Cont'd)

- Database server: CardDatabase
  - stores information related to the cards
  - legacy application
  - written in C++
  - runs on **`mir54.ida.liu.se`**

```
class Card {
    ...
public:
    Card(char* name, char* fileName, char* thumbFileName,
         int x, int y, int width, int height);

    char* getName(void);
    char* getFileName(void);
    char* getCardURL(void);
    char* getThumbURL(void);

    void getTextArea(int* x, int* y,
         int* width, int* height);

    int getX(void);
    int getY(void);
    int getWidth(void);
    int getHeight(void);

    void dump(void);
};
```

```
class CardDatabase {
...
public:
    CardDatabase(char* fileName);
    int getCardsNumber(void);
    Card* getCard(int cardNumber);

    void dump(void);
};
```

About the Existing Components (Cont'd)

- Graphics server: CardWrite
  - writes the message on the card
  - written in Java
  - runs on **`mir53.ida.liu.se`**

```
public class CardWrite {
...
     public CardWrite() {
          ...
     }
     public void setCard(String inputFileName, int x, int y, int w, int h) {
          ...
     }
     public void setText(String message) {
          ...
     }
     public void setFont(Font font) {
          ...
     }
     public void setColor(Color color) {
          ...
     }
     Exception writeSignedCard(String outputFileName) {
          ...
     }
}
```

Your Task

- To implement a program that offers an Electronic Postcard Service using the existing components.

- Your program will be both:
  - a client for CardDatabase and CardWrite
  - a server for the web browser

- Implementation will be done using several methods:
  Lab assignment 2
  - C or C++ program executed through CGI, communicating with sockets
    or
  - Java program running as Servlet, communicating with sockets (2nd option)

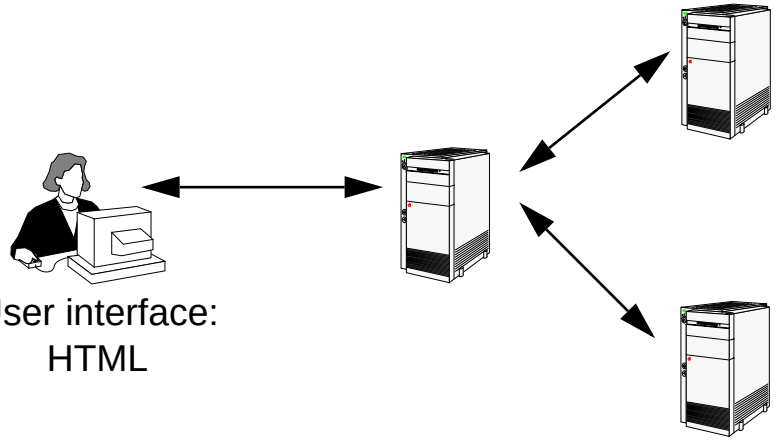
  Lab assignment 3
  - C++ or Java program using CORBA

  Lab assignment 4
  - CardDatabase server with CORBA (written in either C++ or Java)

Lab Assignment 1

- Write the user interface in HTML.
  The user interface will be the same for the assignments 2 to 4, regardless of the particular implementation method.

- Three web pages (these will be later created dynamically):
  1. The first page presents all the cards in the database as thumbnails.
  2. The second page has the image of the chosen card and a text field to input the greeting message.
  3. The third page consists of the card with the message embedded into it.



Lab Assignment 1 (Cont'd): URL, HTML

- URL - Uniform Resource Locator
  URLs are the addresses of documents on the web.

  `protocol://machine.name/path/document`
  example: **`http://www.ida.liu.se/~trapo/index.html`**

- HTML - Hypertext Markup Language
  The set of "markup" symbols (codes or tags) inserted in a file intended for display on a World Wide Web browser. The markup tells the Web browser how to display a Web page's words and images for the user.

  example:

```
<HTML>
<BODY>Hello World!</BODY>
</HTML>
```

Lab Assignment 1 (Cont'd)

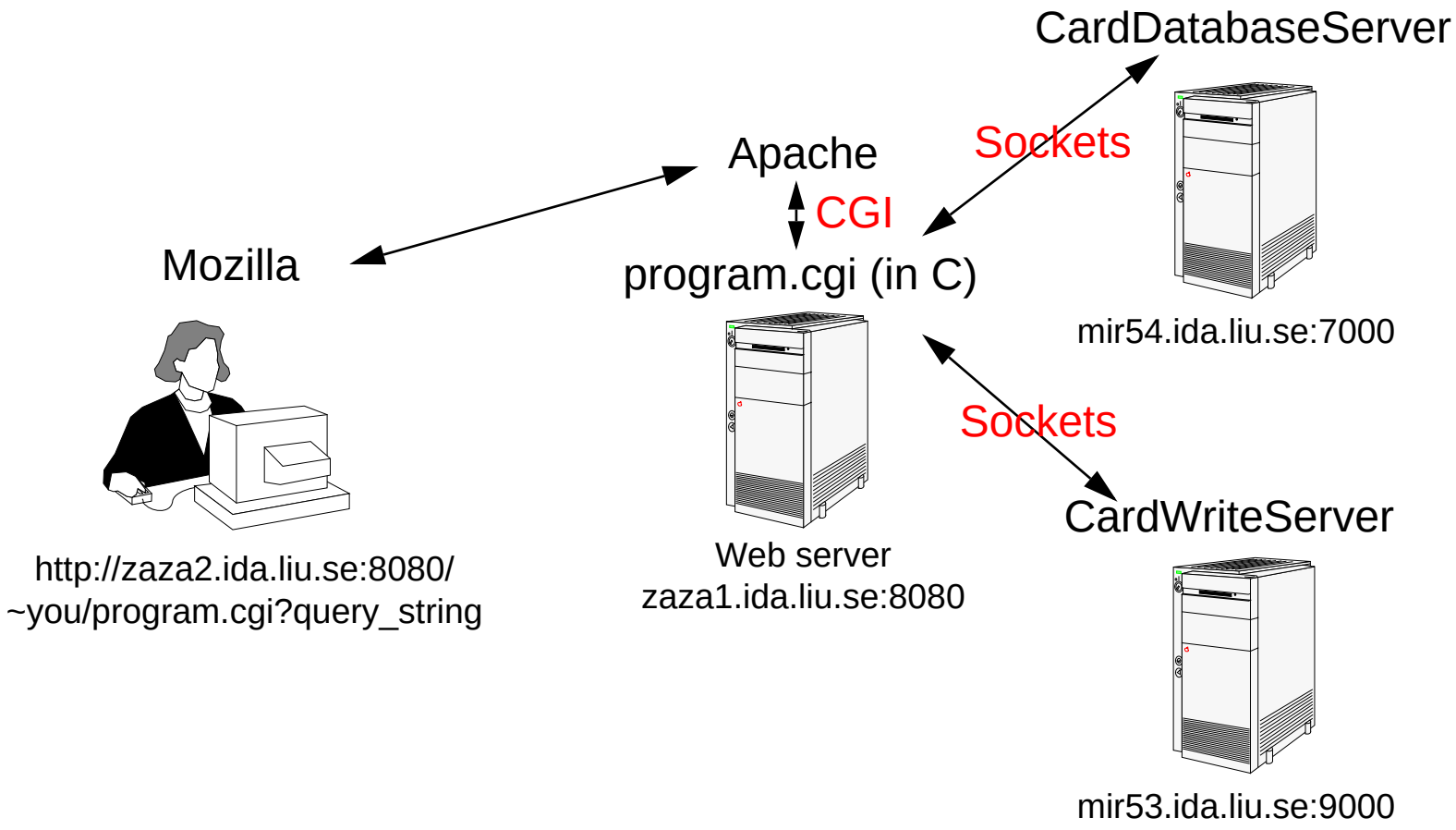
- Your home page (if any) is in **~/www-pub**.
  The URL **http://www-und.ida.liu.se/~trapo/index.html**
  corresponds to the file named **/home/trapo/www-pub/index.html**

- However, we will use a different web server: **su1-1.ida.liu.se:8080.**
  The URL **http://su1-1.ida.liu.se:8080/~trapo/index.html**
  now corresponds to the file named **/home/trapo/TDDB37/index.html**

- Details about the first assignment accessible through the course home page:
  It also contains:
  - an HTML tutorial
  - an example that uses all the tags you need for the HTML interface

- The cards' images are in **http://www.ida.liu.se/~TDDB37/labs/cards**
  (mirrored at **http://www.ida.liu.se/~trapo/TDDB37/cards/)**
- URL for thumbnails is
  http://www.ida.liu.se/~TDDB37/labs/cards/thumbs
  (mirrored at **http://www.ida.liu.se/~trapo/TDDB37/cards/thumbsbs/)**

Lab Assignment 2



Lab Assignment 2 (Cont'd): CGI

- CGI - Common Gateway Interface
  CGI is a standard for interfacing external applications with web servers.
  - static vs. dynamic

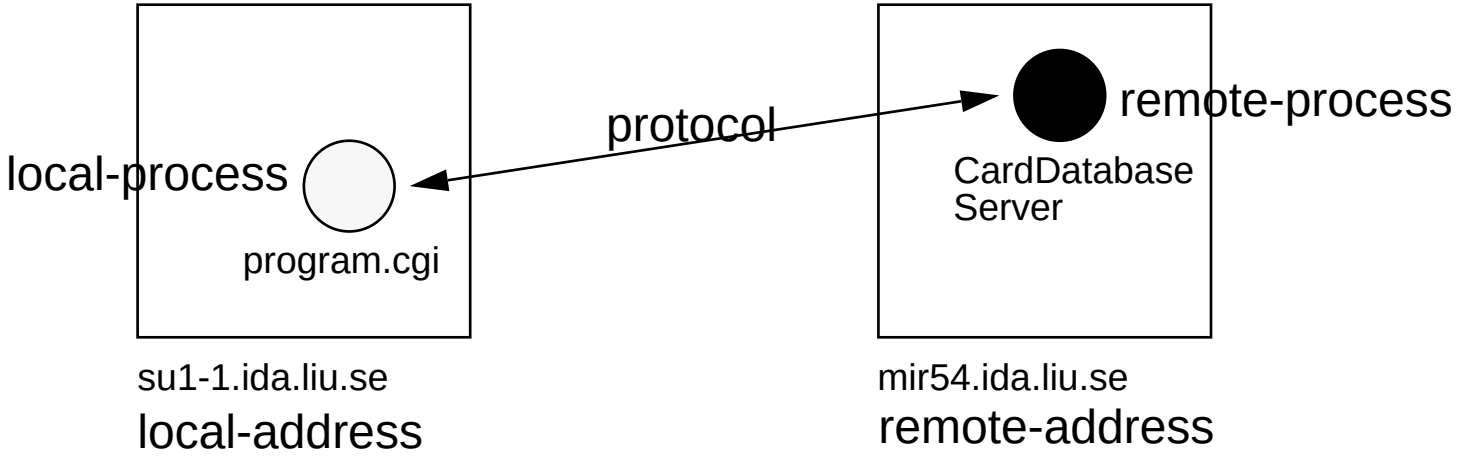
  - example:
    http://machine.name/path/program.cgi?name1=value1&name2=value2

```
string = getenv("QUERY_STRING");
printf("Content-type: text/html\n"); /* for the HTTP */
printf("\n");
printf("<HTML>\n"); /* the HTML document */
printf("<BODY>Hello World!");
printf(" Your query string is: %s", string);
printf("</BODY></HTML>");
```

    On the web browser:
    Hello World! You query string is: name1=value1&name2=value2

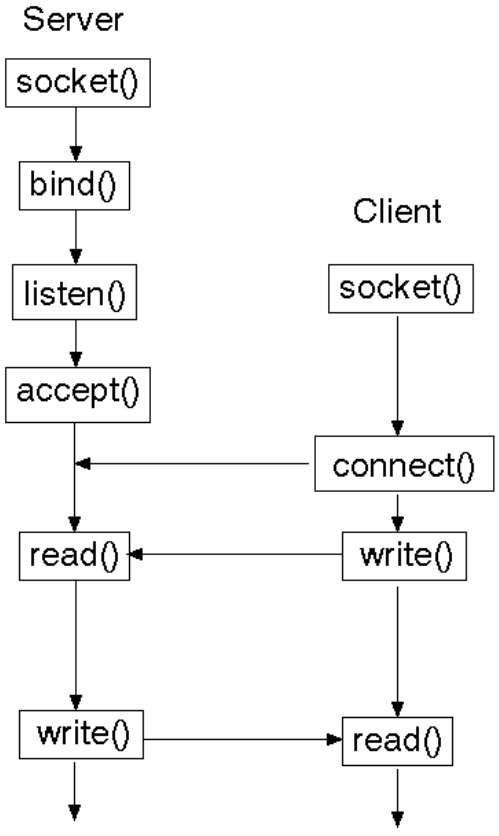
Lab Assignment 2 (Cont'd): Sockets

- Inter Process Communication
  - A communication between two processes running on two computer systems can be completely specified by the association: {protocol, local-address, local-process, remote-address, remote-process}



- **Socket** = half association:
  {protocol, local-address, local-process} or
  {protocol, remote-address, remote-process}.

Lab Assignment 2 (Cont'd): Sockets




```
/* the server */

int sockfd, newsockfd;

if((sockfd = socket(...)) < 0)
     error("socket error");
if(bind(sockfd, ...) < 0)
     error("bind error");
if(listen(sockfd, 5) < 0)
     error("listen error");

for(;;) {
     /* blocks */
     newsockfd = accept(sockfd, ...);
     if(newsockfd < 0)
          error("accept error");

     if(fork() == 0) {
          /* we are in the child */
          close(sockfd);
          /* process the request */
          do_something(newsockfd);
          exit(0);
     }

     close(newsockfd); /* parent */
}
```

```
/* the client */

int sockfd, newsockfd;

if((sockfd = socket(...)) < 0)
     error("socket error");

if(connect(sockfd, ...) < 0)
     error("bind error");

/* request something (e.g., card info) */
request_something(newsockfd);

close(sockfd);
```

Lab Assignment 2 (Cont'd): CardDatabaseServer

- CardDatabaseServer
  - written in C++
  - runs on **`mir54.ida.liu.se`** at port 7000

- Has a simple protocol for requesting information about cards.
  Example using **`telnet`** as a client:

```
~> telnet mir54.ida.liu.se 7000
Trying 130.236.176.85...
Connected to mir54.ida.liu.se.
Escape character is '^]'.
getCardsNumber
27
getCardInfo 4
Field at Sunset
field_sunset.jpg
http://www.ida.liu.se/~trapo/TDDB37/cards/field_sunset.jpg
http://www.ida.liu.se/~trapo/TDDB37/cards/thumbs/
field_sunset_t.jpg
20
350
460
140
getCardInfo 45
ERROR: bad card number 45
get Cards
ERROR: bad request string
exit
Connection closed by foreign host.
~>
```

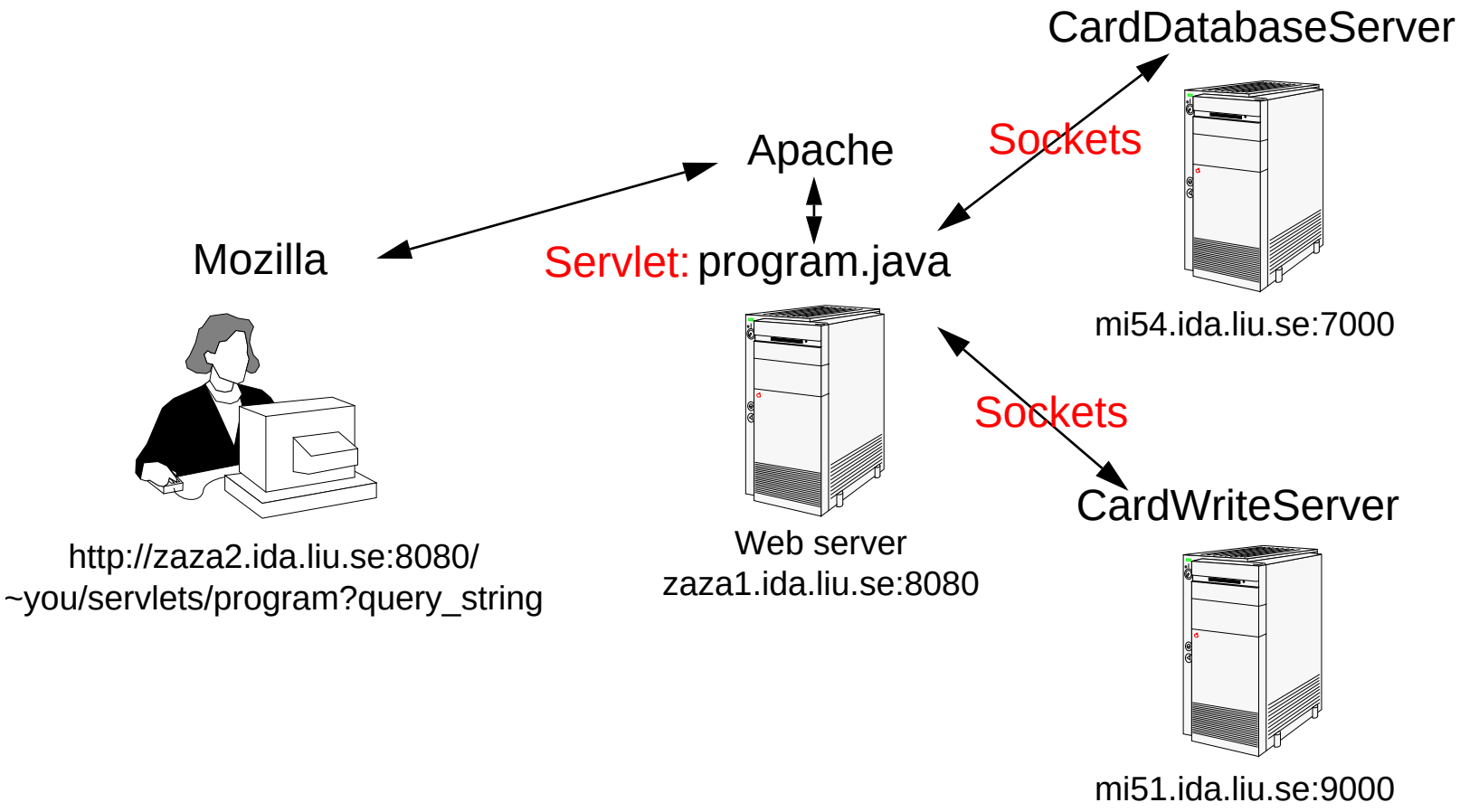
Lab Assignment 2 (Cont'd): CardWriteServer

- CardWriteServer
  - written in Java
  - runs on **`mir53.ida.liu.se`** at port 9000

- Has a simple protocol for requesting a card's image that has the message on it. Example using **`telnet`** as a client:

```
~> telnet mir53.ida.liu.se 9000
Trying 130.236.176.84...
Connected to mir53.ida.liu.se.
Escape character is '^]'.
field_sunset.jpg
002-4189664-9234420.jpg
20 350 460 140
Hello World!

/a student
end-of-message
http://www.ida.liu.se/~trapo/TDDB37/cards/temp/
002-4189664-9234420.jpg
exit
Connection closed by foreign host.
~>
```

Lab Assignment 2: Option #2



Lab Assignment 2(Cont'd): Servlets

- Drawbacks of CGI
  - primitive, low-level interaction between the web server and the application
  - slow: has to spawn a new program for every access;
    this involves overheads related to the operating system
  - security problems

- Applet: a piece of Java code running on the client (the web browser).

- **Servlet**: a piece of Java code running on the web server.
  - SUN Mircosystem's "invention"
  - tightly integrated with the web server that controls their execution
  - offers the power of Java language
  - the Java Servlet Development Kit (JSDK) speeds up the implementation
  - the output from a servlet can be **inserted** in a page using the <SERVLET> tag.

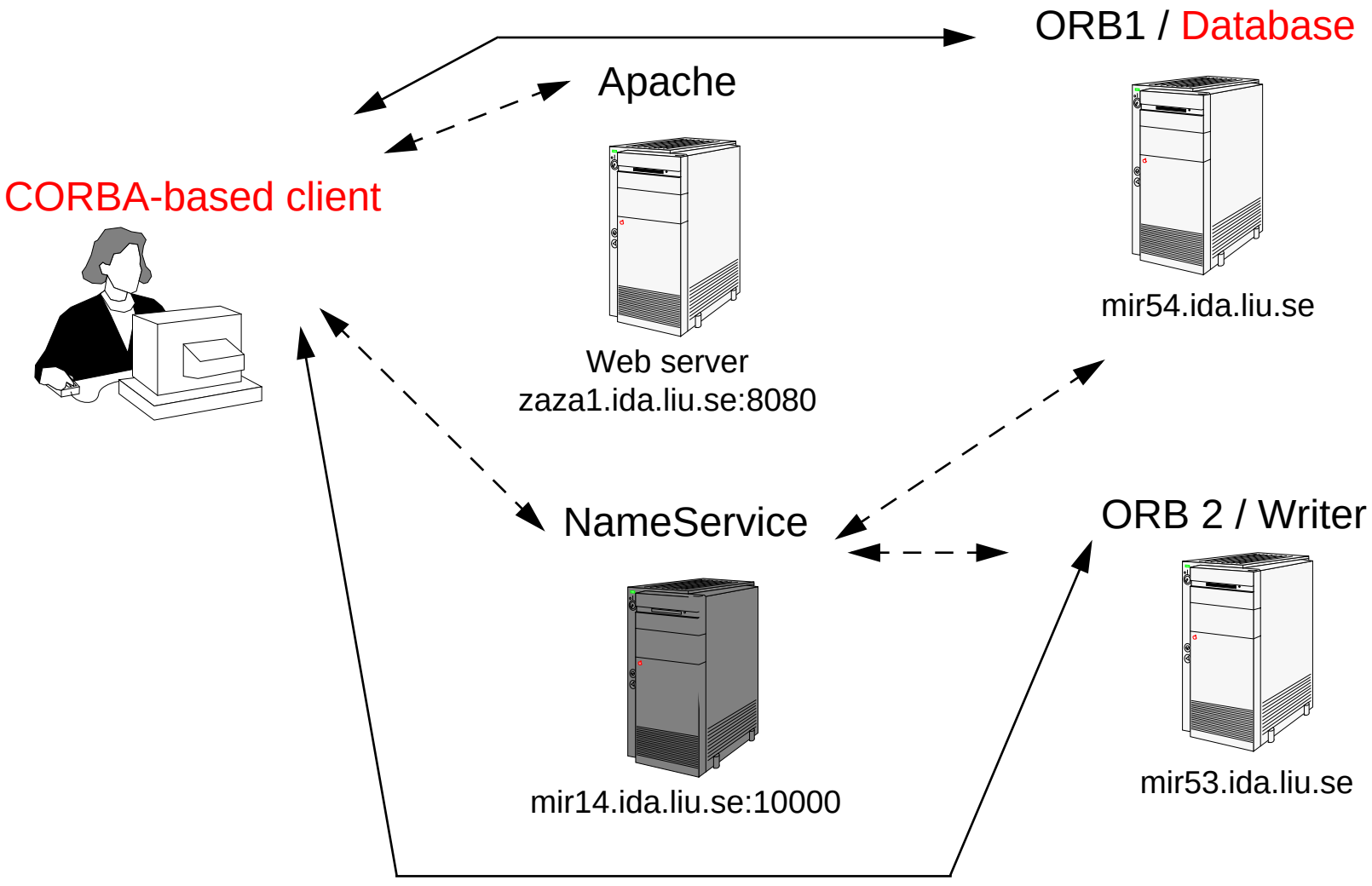
    example: **`<servlet code=DateServlet.class></servlet>`**

    in the web browser: **`Wed Jan 20 22:53:57 MET 2003`**

Lab Assignment 2(Cont'd): Servlet Example

```
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;


/**
 * This is a simple example of an HTTP Servlet.
 */
public class SimpleServlet extends HttpServlet {

    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        ServletOutputStream out = res.getOutputStream();

        // set content type and other response header fields first
        res.setContentType("text/html");

        // then write the data of the response
        out.println("<HEAD><TITLE> SimpleServlet Output </TITLE></HEAD><BODY>");
        out.println("<h1> SimpleServlet Output </h1>");
        out.println("<P>This is output from SimpleServlet.");
        out.println("</BODY>");
        out.close();
    }

    public String getServletInfo() {
        return "A simple servlet";
    }
}
```

Lab Assignment 3

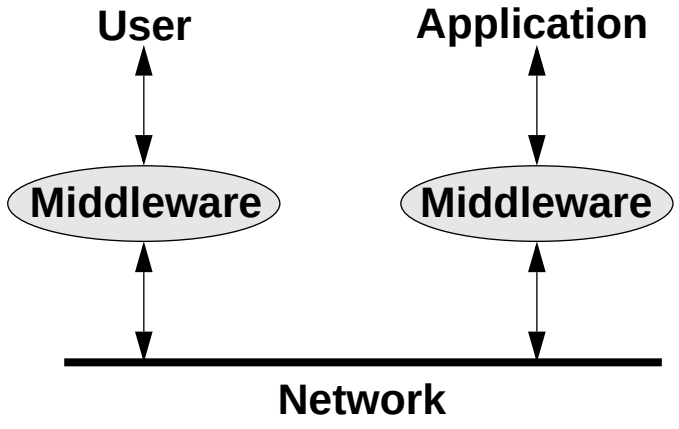


Lab Assignment 3(Cont'd): CORBA

- Lectures during the course about CORBA.

- Difficulties of distributed programming:
  - several inter-networked machines: different hardware
  - different operating systems, programming languages: different software
  - integration of legacy systems (e.g., CardsDatabase)

- Middleware
  - set of services that bridge the gap between the users and the applications
  - makes the network transparent (behave as locally)
  - hides the details of hardware, OS, software components



Objects and Distributed Systems

- A distributed application can be viewed as a set of **objects**.

- Objects:
  - consist of data + code
  - objects can be clients, servers or both.
  - modelling with objects does not imply the use of object oriented programming

- Middleware:
  - **Object brokers**: allow objects to find each other in a distributed system, and interact with each other
  - **Object services**: allow to create, name, move, copy, store, delete, restore and manage objects.

CORBA

- Object Management Group (OMG)
  - industry consortium formed in 1989 with the goal to develop standards for the development of distributed heterogeneous applications.

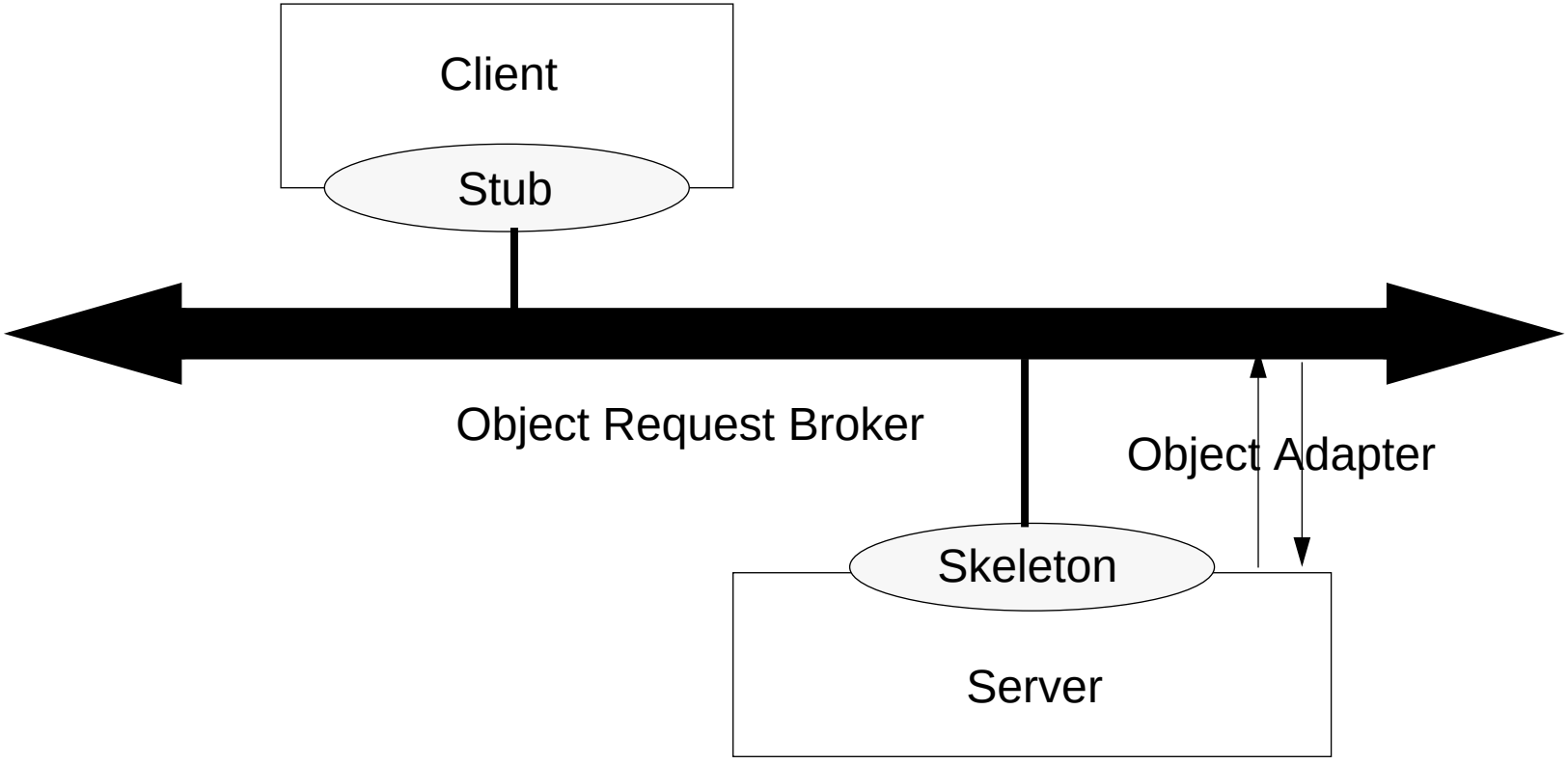
- CORBA - Common Object Request Broker Architecture
  - a **standard** (specification) to support the development and integration of distributed systems consisting of objects.

  - specifies the middleware services used by objects

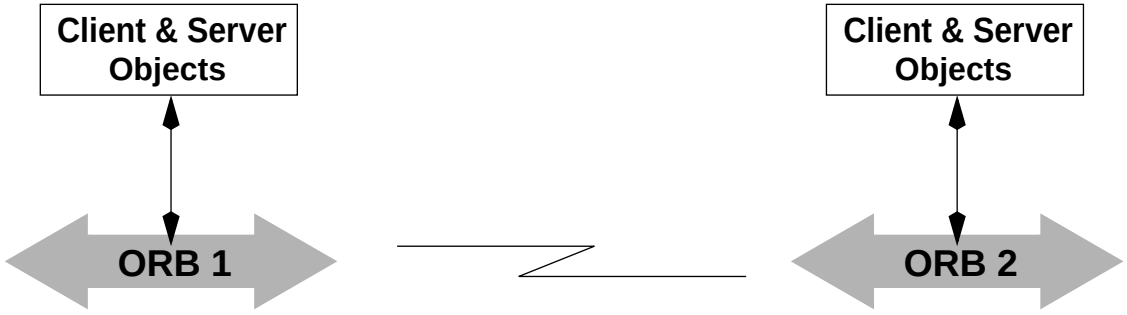
  - Object
    - has an unique ID, the “object reference”
    - has an interface describing its services
    - can be written in any programming language supported by CORBA
    - can be distributed everywhere there is an ORB

ORB

- ORB - Object Request Broker
  - a particular implementation of the CORBA standard
  - for the labs we will use ORBacus, a free CORBA implementation



Inter-ORB Architecture



- GIIOP - General Inter-ORB Protocol
  Specifies a set of message formats and common data representations for interactions between ORBs and is intended to operate over any connection oriented transport protocol.

- IIOP - Internet Inter-ORB Protocol
  Is a particularization of GIOP; it specifies how GIOP messages have to be exchanged over a TCP/IP network.

IDL

- Object interfaces are specified in IDL - Interface Definition Language
  Example:

```
interface Database
{
        short getCardsNumber();

        string getCardName(in short cardNumber);
        string getCardFile(in short cardNumber);
        string getCardURL(in short cardNumber);
        string getCardTumb(in short cardNumber);
        boolean getCardArea(in short cardNumber, out short x, out short y,
                 out short width, out short height);
};
```

- The IDL interface is translated by an IDL compiler into **stubs** and **skeletons** for a particular programming language and ORB.

Lab Assignment 3(Cont'd): Client Example

```
// IDL
interface Hello
{
    void say_hello();
};

// Client in C++
#include <OB/CORBA.h>
#include <Hello.h>

#include <fstream.h>

int main(int argc, char* argv[], char*[])
{
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv, "Hello-Client");

    CORBA::Object_var obj = orb -> string_to_object("relfile:/Hello.ref");
    Hello_var hello = Hello::_narrow(obj);

    hello -> say_hello();
}
```

Lab Assignment 4

- Implement the CardDatabase server using CORBA.
- You will get:
  - the IDL file

```
interface Database
{
    short getCardsNumber();

    string getCardName(in short cardNumber);
    string getCardFile(in short cardNumber);
    string getCardURL(in short cardNumber);
    string getCardTumb(in short cardNumber);
    boolean getCardArea(in short cardNumber, out short x, out short y,
                out short width, out short height);
};
```

  - the database (a text file) with the information about the cards

```
...
Sunrise at Sea,sea_sunrise.jpg,sea_sunrise_t.jpg,73,400,430,490
...
```

Lab Assignment 4: (Cont'd): Server Example

```
public class Server {
    public static void main(String args[]) {
        try {
            // Create ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
            // Resolve Root POA
            org.omg.PortableServer.POA rootPOA =
                org.omg.PortableServer.POAHelper.narrow(
                orb.resolve_initial_references("RootPOA"));
            // Get a reference to the POA manager
            org.omg.PortableServer.POAManager manager = rootPOA.the_POAManager();
            // Create implementation object
            Hello_impl helloImpl = new Hello_impl(rootPOA);
            Hello hello = helloImpl._this(orb);
            // Save reference
            try {
                String ref = orb.object_to_string(hello);
                String refFile = "Hello.ref";
                FileOutputStream file =new FileOutputStream(refFile);
                PrintWriter out = new PrintWriter(file);
                out.println(ref); out.flush(); file.close();
            } catch(java.io.IOException ex){
                System.err.println("hello.Server: can't write to '" + ex.getMessage() + "'");
                System.exit(1);
            }
            // Run implementation
            manager.activate();
            orb.run();
        }
    }
}
```

Lab Assignment 5: Mutual Exclusion in Distributed Systems

- Writer server has a problem: it does not enforce mutual exclusion

```
interface Writer
{
    void setCard(in string inputFileName,
                 in short x, in short y, in short width, in short height);
    void setMessage(in string message);
    string writeCard(in string outputFileName);
};
```

- Assignment: enforce mutual exclusion using semaphores (C/C++) or synchronization (Java)