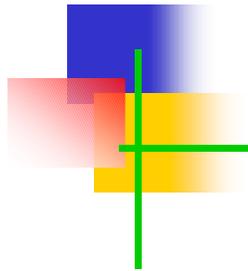


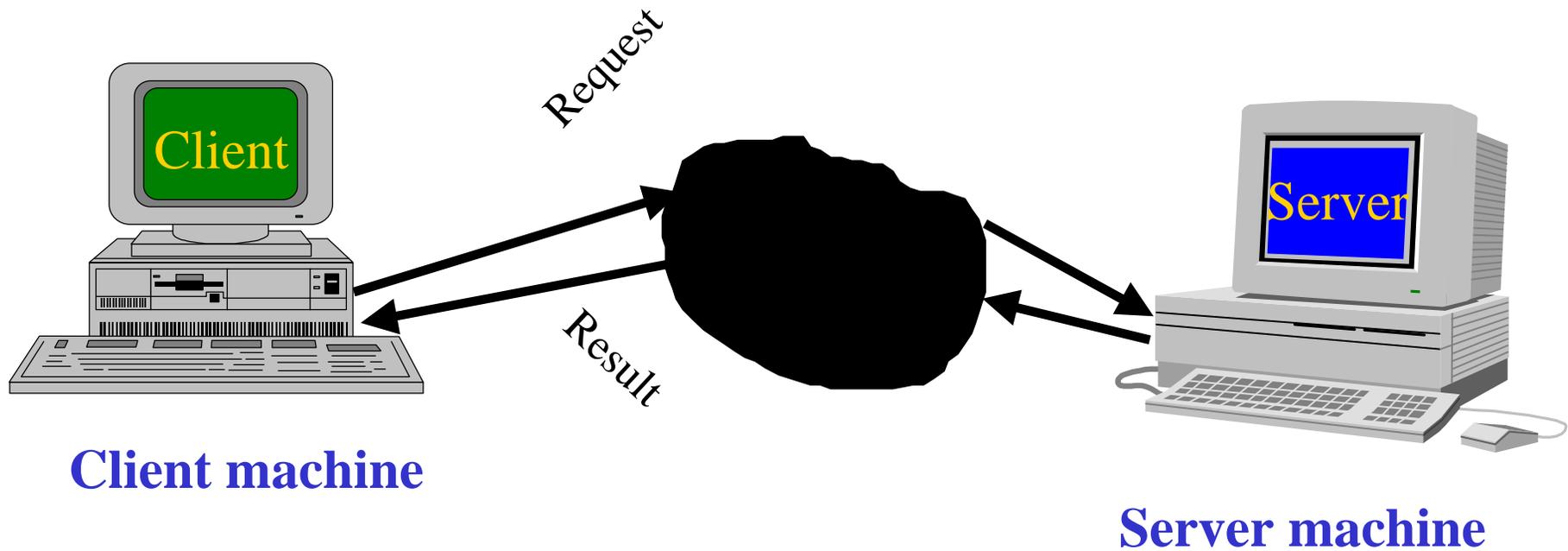
Network/Socket Programming in Java



Rajkumar Buyya

Elements of C-S Computing

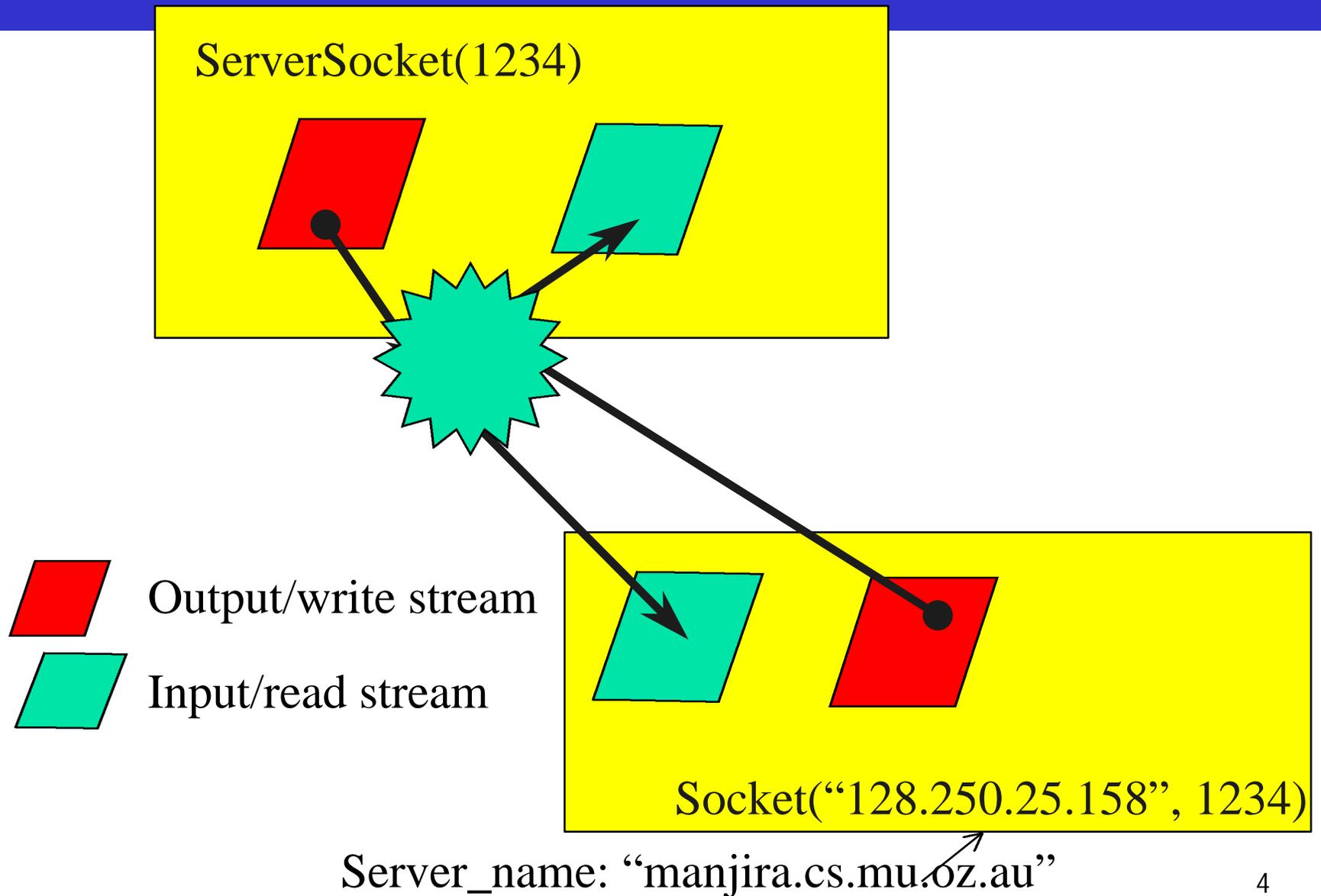
a client, a server, and network



java.net

- Used to manage:
 - ↳ **URL streams**
 - ↳ **Client/server sockets**
 - ↳ **Datagrams**

Part III - Networking



Server side Socket Operations

1. Open Server Socket:

```
ServerSocket server;  
DataOutputStream os;  
DataInputStream is;  
server = new ServerSocket( PORT );
```

2. Wait for Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to clients

```
is = new DataInputStream( client.getInputStream() );  
os = new DataOutputStream( client.getOutputStream() );
```

4. Perform communication with client

```
Receive from client: String line = is.readLine();  
Send to client: os.writeBytes("Hello\n");
```

5. Close sockets: client.close();

For multithreaded server:

```
while(true) {  
    i. wait for client requests (step 2 above)  
    ii. create a thread with "client" socket as parameter (the thread creates streams (as in step  
        (3) and does communication as stated in (4). Remove thread once service is provided.  
}
```

Client side Socket Operations

1. Get connection to server:

```
client = new Socket( server, port_id );
```

2. Create I/O streams for communicating to clients

```
is = new DataInputStream( client.getInputStream() );
```

```
os = new DataOutputStream( client.getOutputStream() );
```

3. Perform communication with client

```
Receiive from client: String line = is.readLine();
```

```
Send to client: os.writeBytes("Hello\n");
```

4. Close sockets: `client.close();`

A simple server (simplified code)

```
import java.net.*;
import java.io.*;
public class ASimpleServer {
    public static void main(String args[]) {
        // Register service on port 1234
        ServerSocket s = new ServerSocket(1234);
        Socket s1=s.accept(); // Wait and accept a connection
        // Get a communication stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (s1out);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
        s1out.close();
        s1.close();
    }
}
```

A simple client (simplified code)

```
import java.net.*;
import java.io.*;
public class SimpleClient {
    public static void main(String args[]) throws IOException {
        // Open your connection to a server, at port 1234
        Socket s1 = new Socket("130.63.122.1",1234);
        // Get an input file handle from the socket and read the input
        InputStream s1In = s1.getInputStream();
        DataInputStream dis = new DataInputStream(s1In);
        String st = new String (dis.readUTF());
        System.out.println(st);
        // When done, just close the connection and exit
        dis.close();
        s1In.close();
        s1.close();
    }
}
```

Echo Server Client..

```
//client.java: client interface to server
import java.io.*;
import java.net.*;
public class client
{
    int port_id;
    String server; Socket slink;
    DataOutputStream os;
    DataInputStream is;
    DataInputStream kbd;
    public client( String args[] )
    {
        server = args[0];
        port_id = Integer.valueOf(args[1]).intValue();
        try
        {
            slink = new Socket( server, port_id );
            os = new DataOutputStream( slink.getOutputStream() );
            is = new DataInputStream( slink.getInputStream() );
            kbd = new DataInputStream( System.in );
        }
    }
}
```

Echo Server Client..

```
catch( UnknownHostException e )
{
    System.err.println( "Don't know about host: " );
    System.exit(1);
}
catch( IOException e )
{
    System.err.println( "Could not get I/O for the
connection to "+server);
    System.exit(1);
}
}
}

void communicate()
{
    while(true)
    {
        try {
```

Echo Server Client..

```
if( line.equals("end") )
{
    os.close(); is.close(); slink.close();
    break;
}
String line2 = is.readLine();
System.out.println("Output: "+line2);
}
catch( IOException e )
{
    System.out.println(e); }
}
public static void main( String [] args )
{
    if( args.length < 2 )
    {
        System.out.println("Usage: java client
server_name port_id" );
        System.exit(1);
    }
}
```

Echo Server ...

```
// server.java: echo server
import java.io.*;
import java.net.*;
public class server
{
    // public final static int PORT = 4779;
    public static void main( String [] args )
    {
        ServerSocket server = null;
        DataOutputStream os = null;
        DataInputStream is = null;
        boolean shutdown = false;
        if( args.length < 1 )
        {
            System.out.println( "Usage: java server
port_num" );
            System.exit( 1 );
        }
        . . . . .
```

Echo Server ...

```
catch( IOException e )
{
    System.err.println( "Could not get I/O for the
connection to: ");
}
while(!shutdown)
{
    if( server != null )
    {
        try
        {
            Socket client = server.accept();
            System.out.println("Connected");
            InetAddress cip = client.getInetAddress();
            System.out.println( "Client IP Addr:
"+cip.toString());
            is = new DataInputStream(
client.getInputStream() );
            os = new DataOutputStream(
```

Echo Server ...

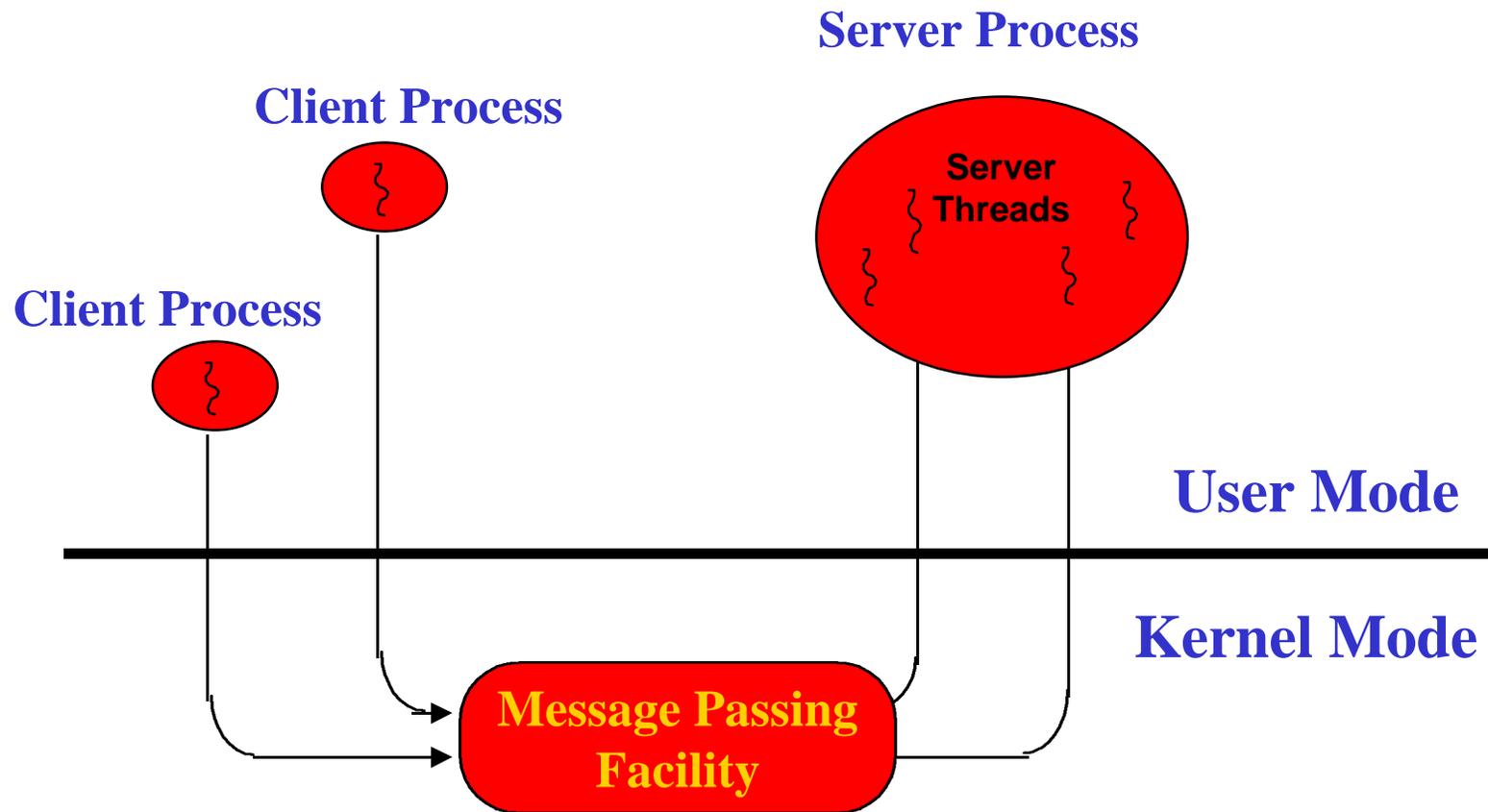
```
        if( line.startsWith("end" ) )
        {
            shutdown = true;
            break;
        }
        os.writeBytes(line.toUpperCase());
        os.writeBytes("\n");
        System.out.println(line);
    }
    is.close(); client.close();
}
catch( UnknownHostException e )
{
    System.err.println( "Server Open fails" );
}
catch( IOException e )
{
    System.err.println( "Could not get I/O for the connection
```

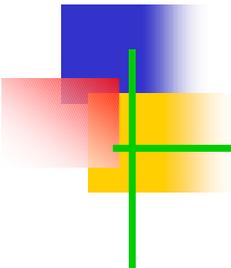
Echo Server

```
System.out.println( "Server Down" );  
    try {  
        server.close();  
    } catch(IOException e) {}  
}  
}
```

Threads in Action...

Multithreaded Server





Client/Server Computing

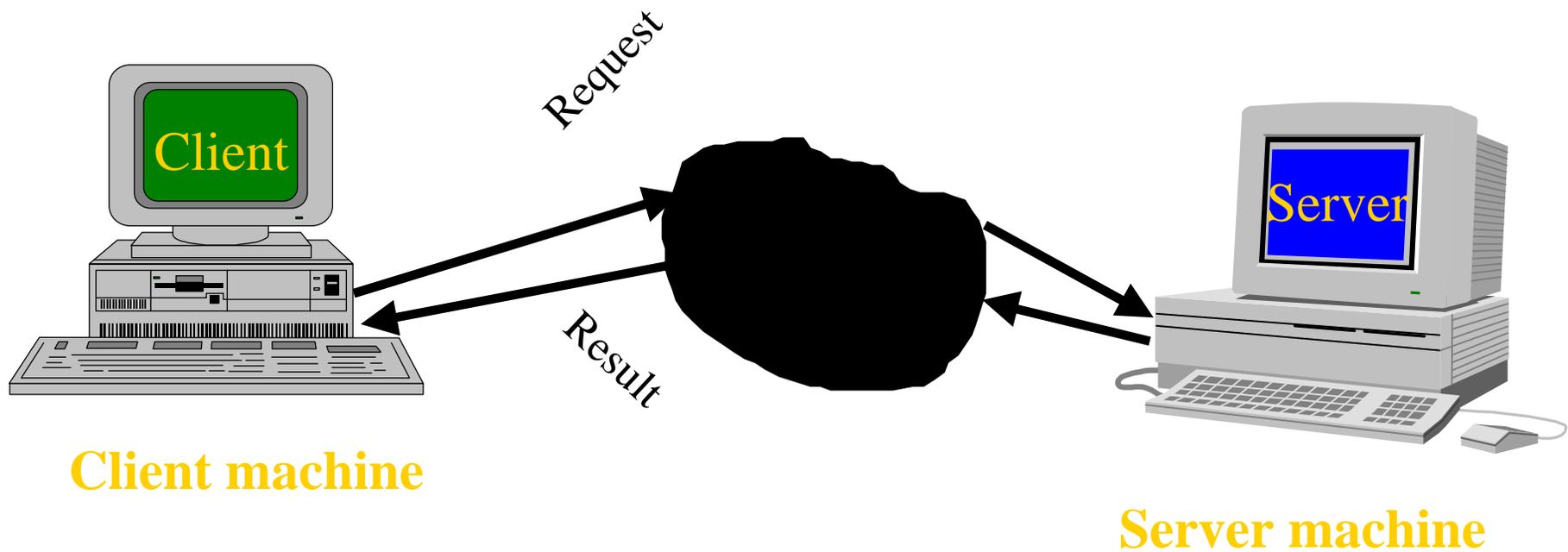
Rajkumar Buyya

Client Server Definition

- “server software accepts requests for data from client software and returns the results to the client”

Elements of C-S Computing

a client, a server, and network



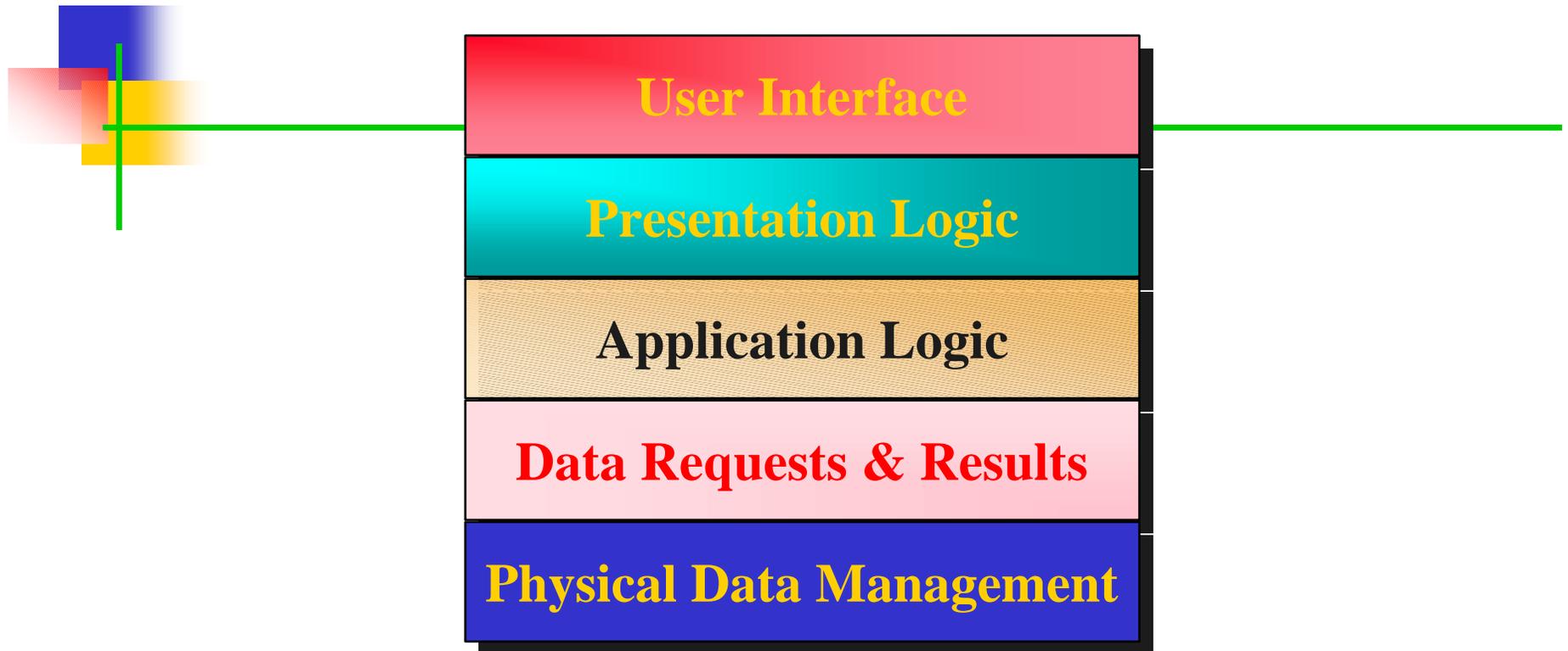
Where Operations are Done

In CS Relationship “most of the application processing is done on a computer (**client side**), which obtains application services (such as database services) from another computer (**server side**) in a master slave configuration.

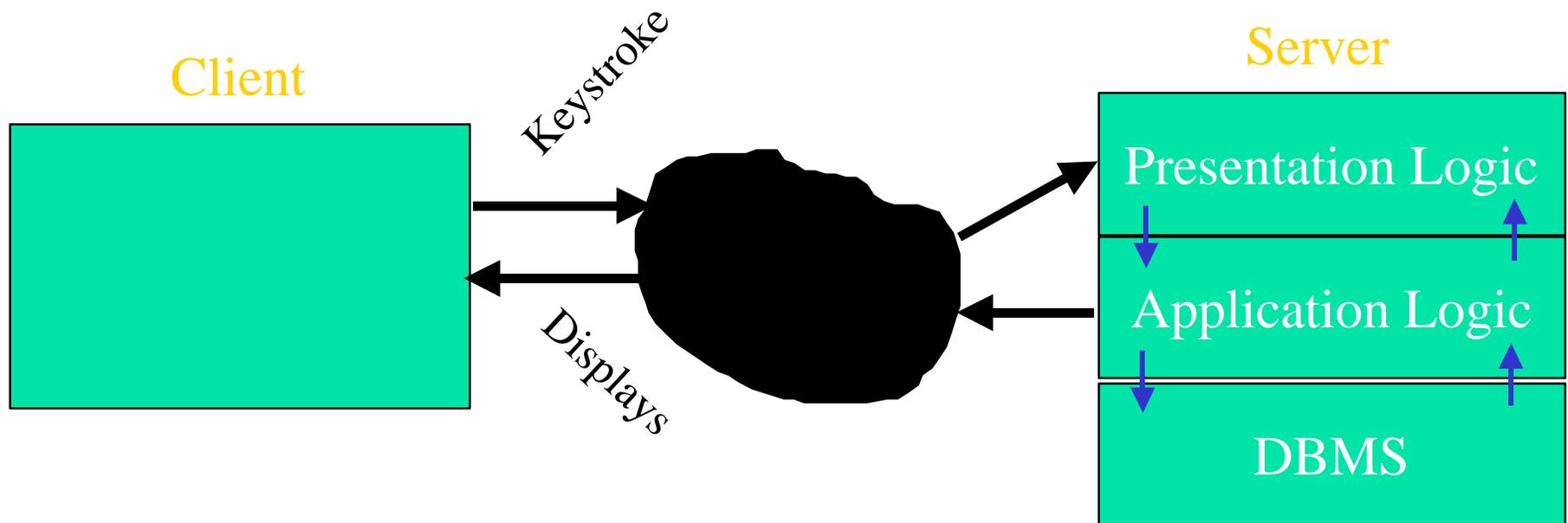
CS-Focus is on

- In client-server computing major focus is on SOFTWARE

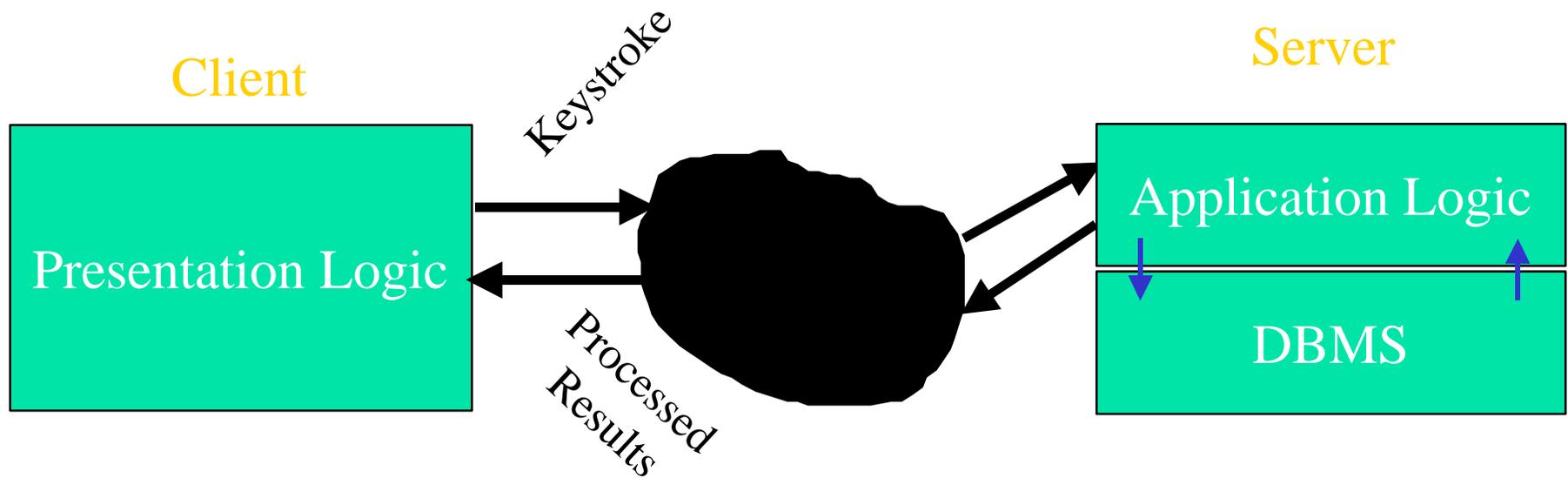
Application Tasks



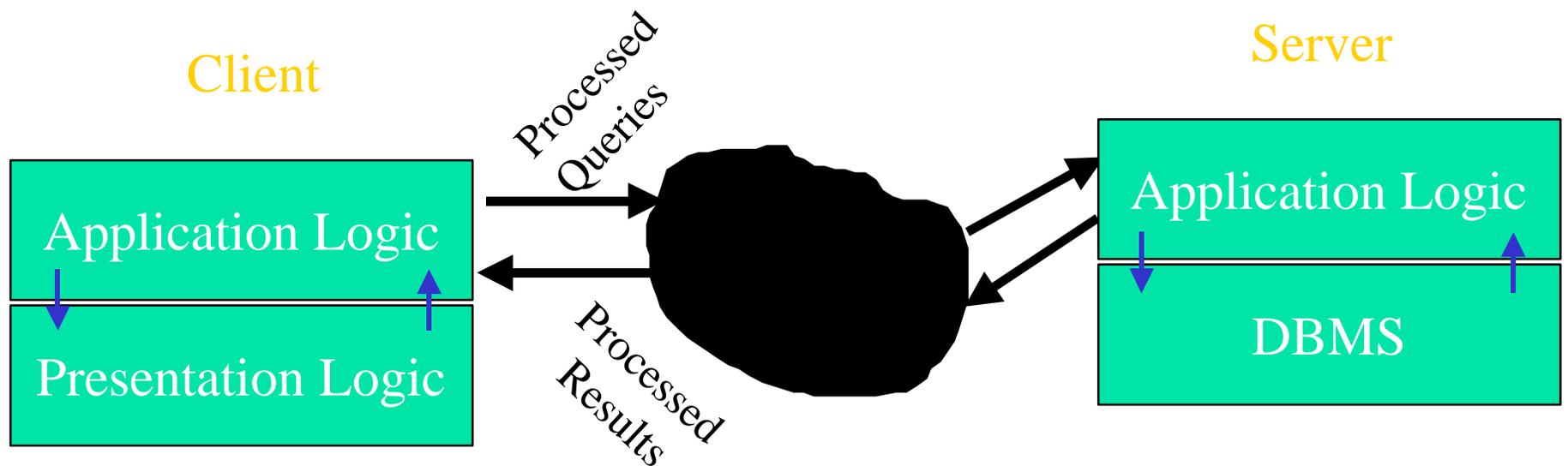
Client (dumb) - Server Model



True Client-Server Model

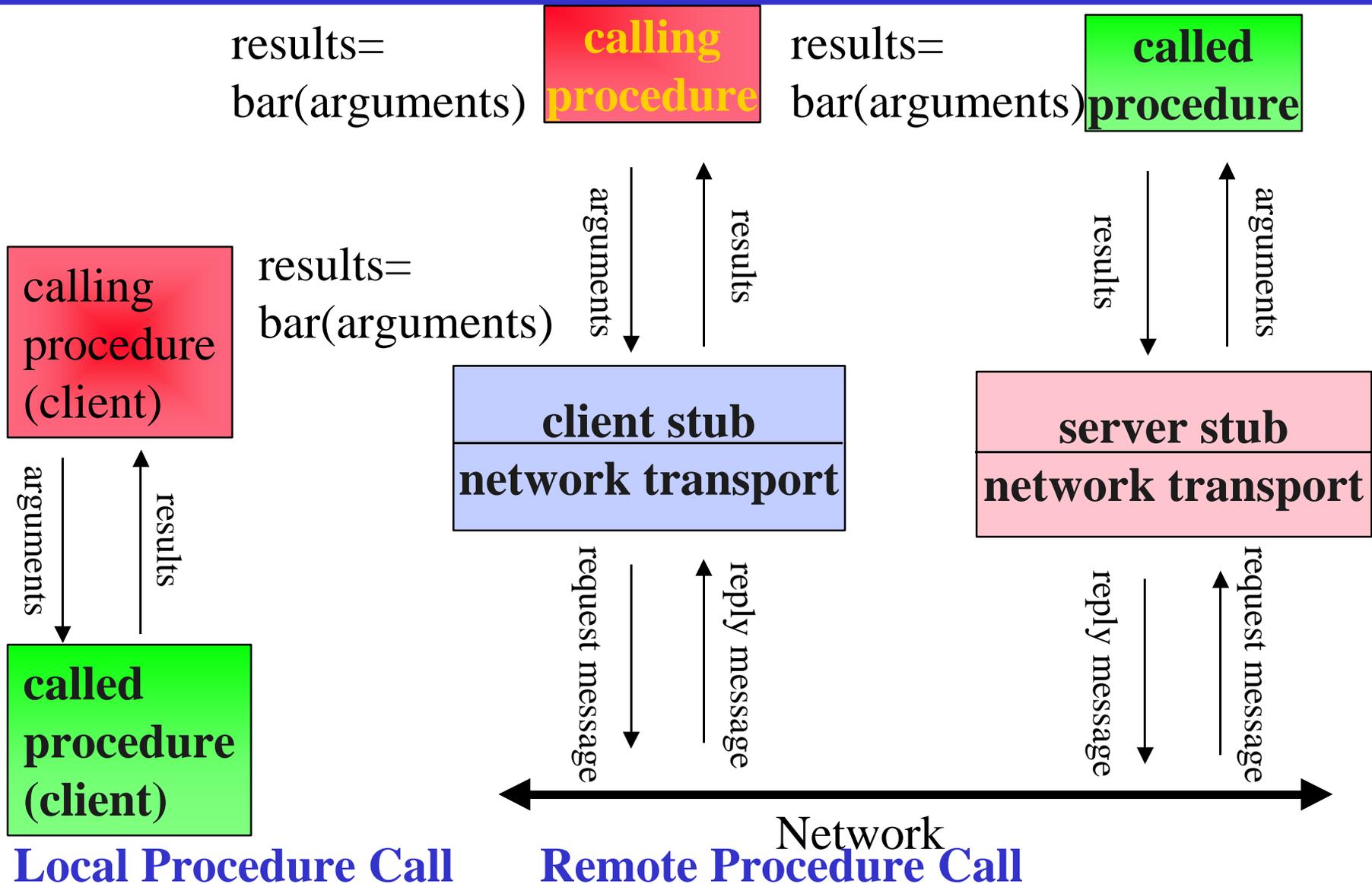


Distributed Client-Server Model

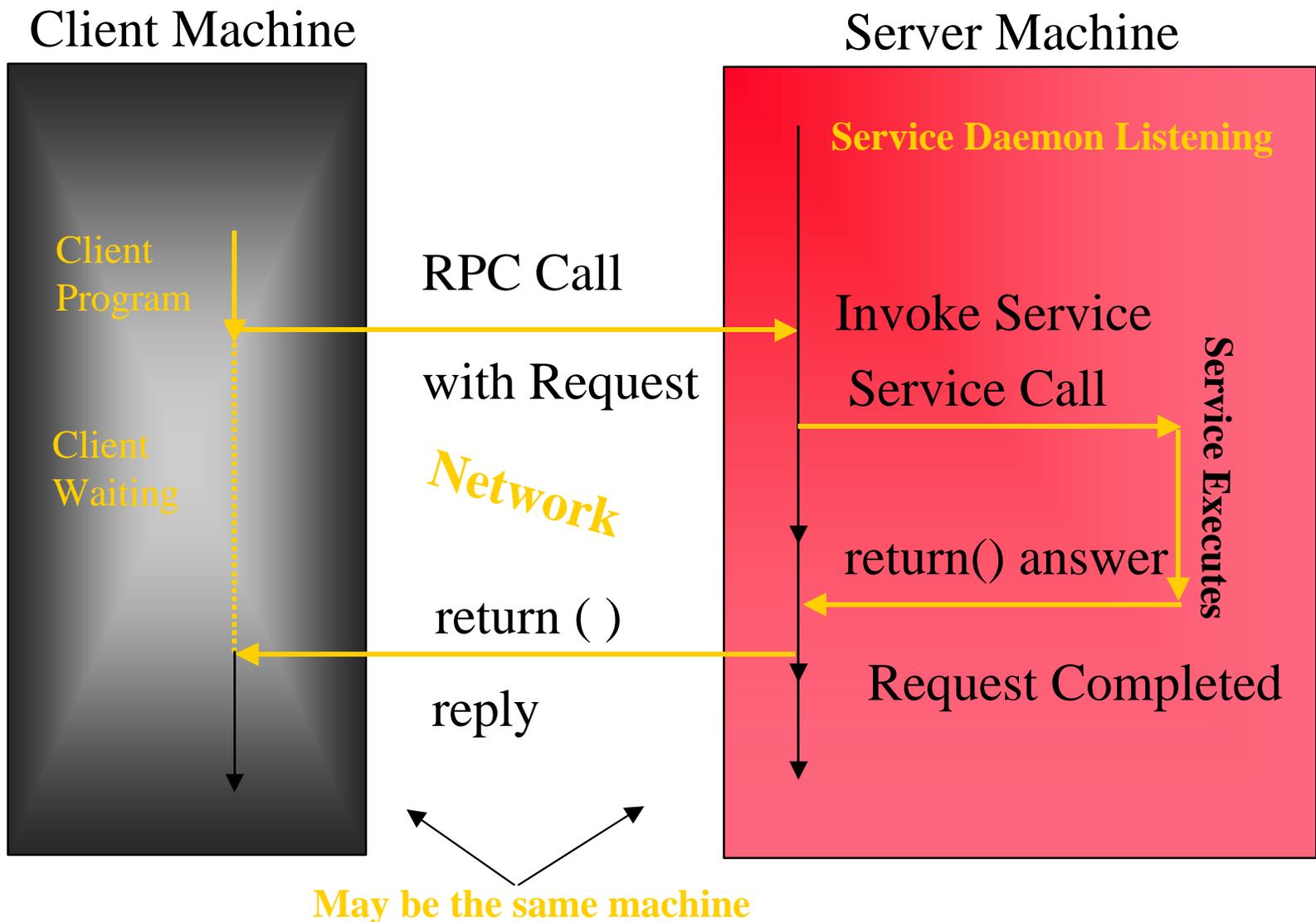


- 
- Client-server computing is distributed access, not a distributed computing.

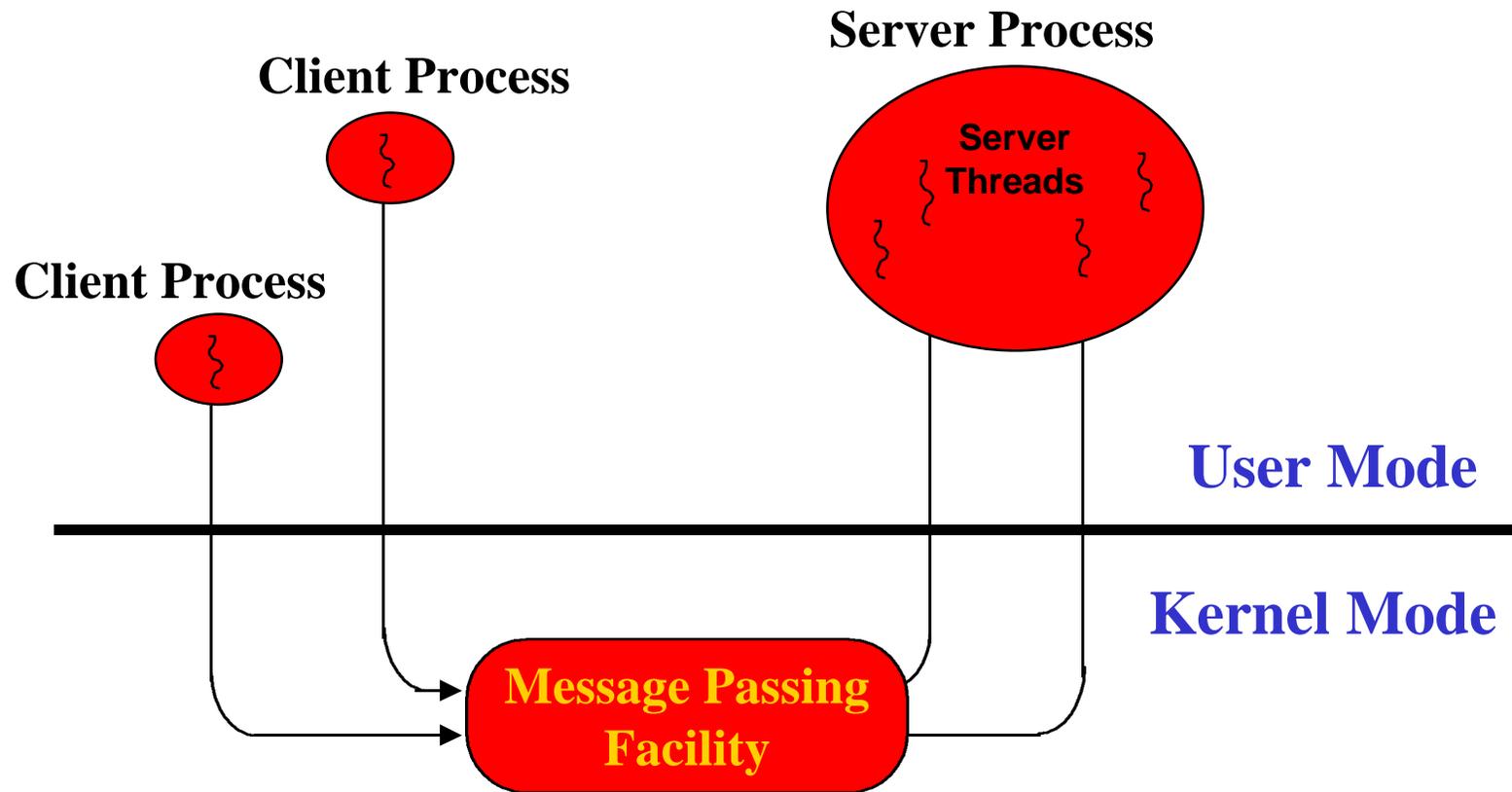
RPC Look and Feel like Local Calls



Flow Control in a Synchronous RPC



Multithreaded Server



Categories of Servers

- File Server
- Data Server
- Compute Server
- Database Server
- Communication Server
- Video Server

File Server

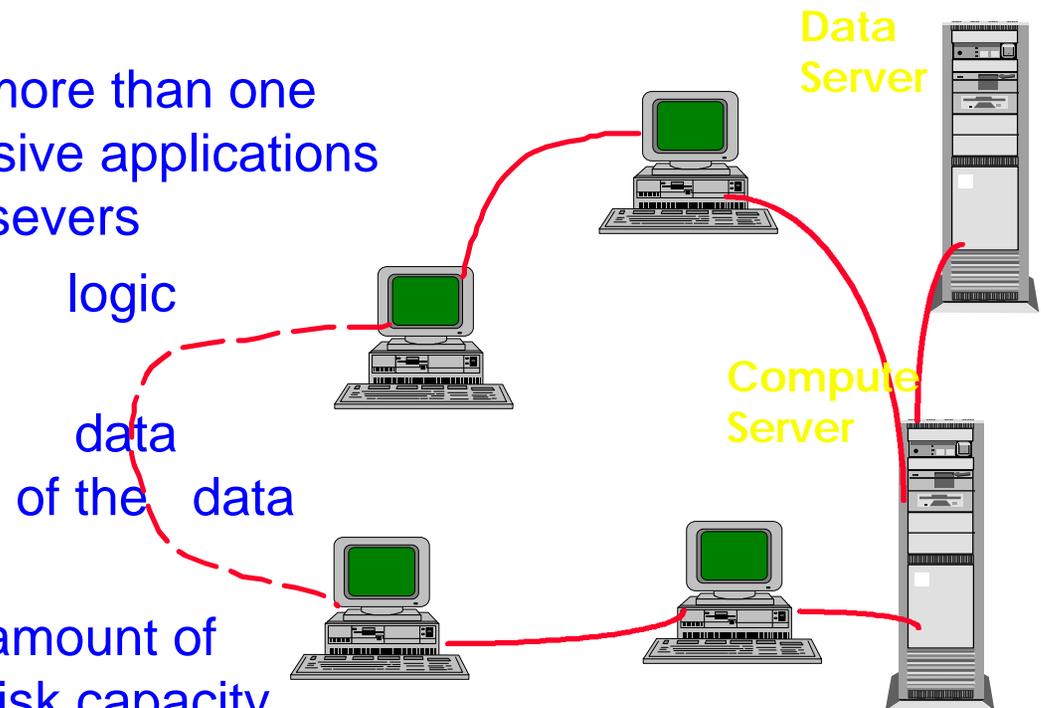
- File Servers manage a work group's application and data files, so that they may be shared by the group.
- Very I/O oriented
- Pull large amount of data off the storage subsystem and pass the data over the network
- Requires many slots for network connections and a large-capacity, fast hard disk subsystem.

Compute Server

- Performs Application logic processing
- Compute Servers requires
 - ↳ processors with high performance capabilities
 - ↳ large amounts of memory
 - ↳ relatively low disk subsystems
- By separating data from the computation processing, the compute server's processing capabilities can be optimized

Data Server

- Data-oriented; used only for data storage and management
- Since a data server can serve more than one compute server, compute-intensive applications can be spread among multiple servers
- Does not prefer any application logic processing
- Performs processes such as validation, required as part of the data management function.
- Requires fast processor, large amount of memory and substantial Hard disk capacity.



Database Server

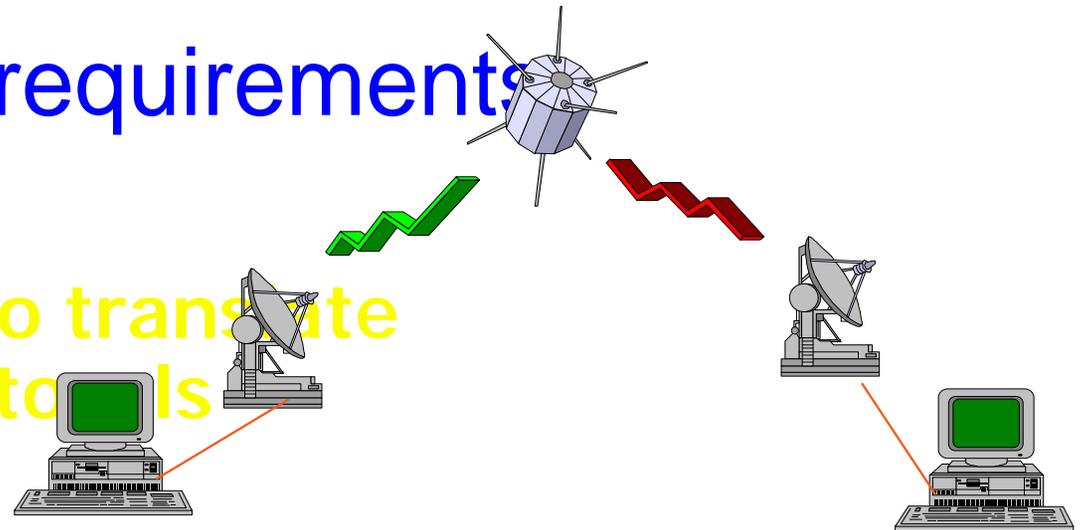
- Most typical use of technology in client-server
- Accepts requests for data, retrieves the data from its database(or requests data from another node)and passes the results back.
- Compute server with data server provides the same functionality.
- The server requirement depends on the size of database, speed with which the database must be updated, number of users and type of network used.

Communication Server

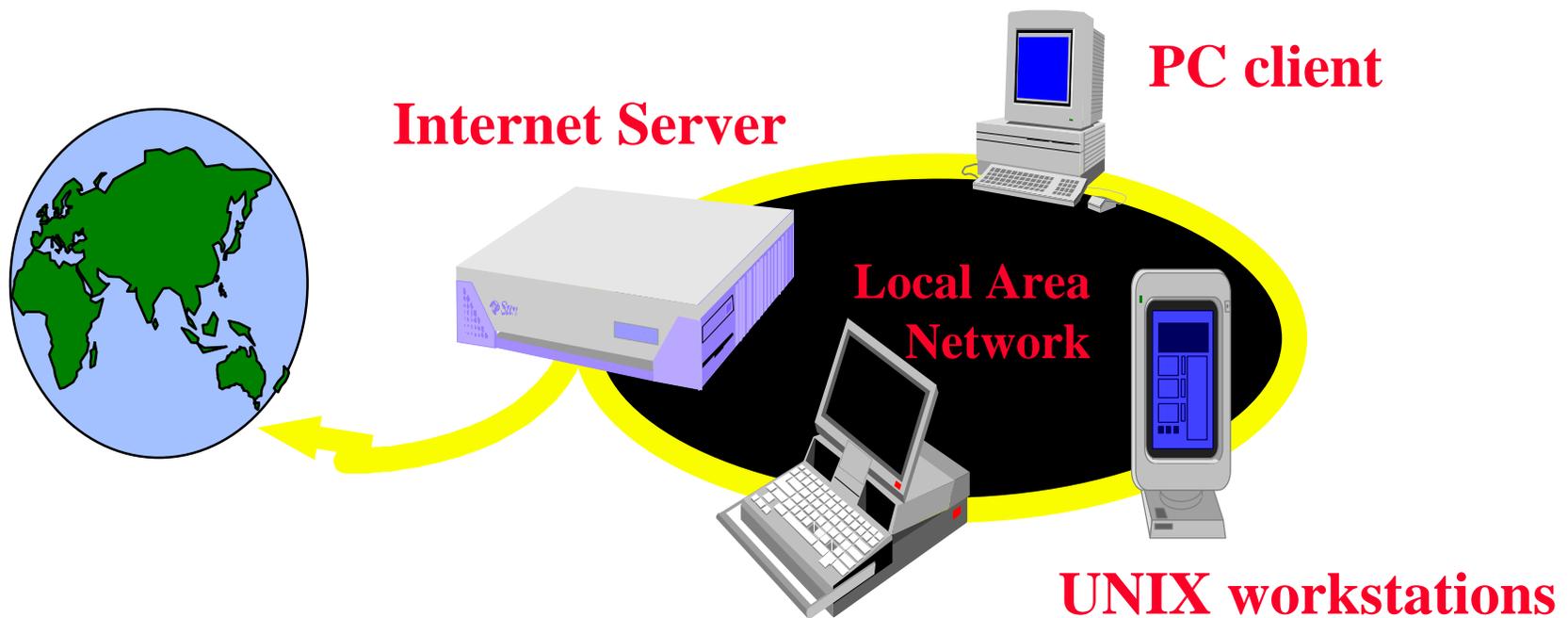
- ❖ Provides gateway to other LANs, networks & Computers
- ❖ E-mail Server & internet server
- ❖ Modest system requirements

- 👉 multiple slots

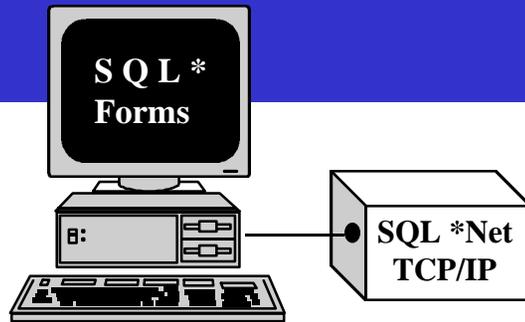
- 👉 fast processor to translate networking protocols



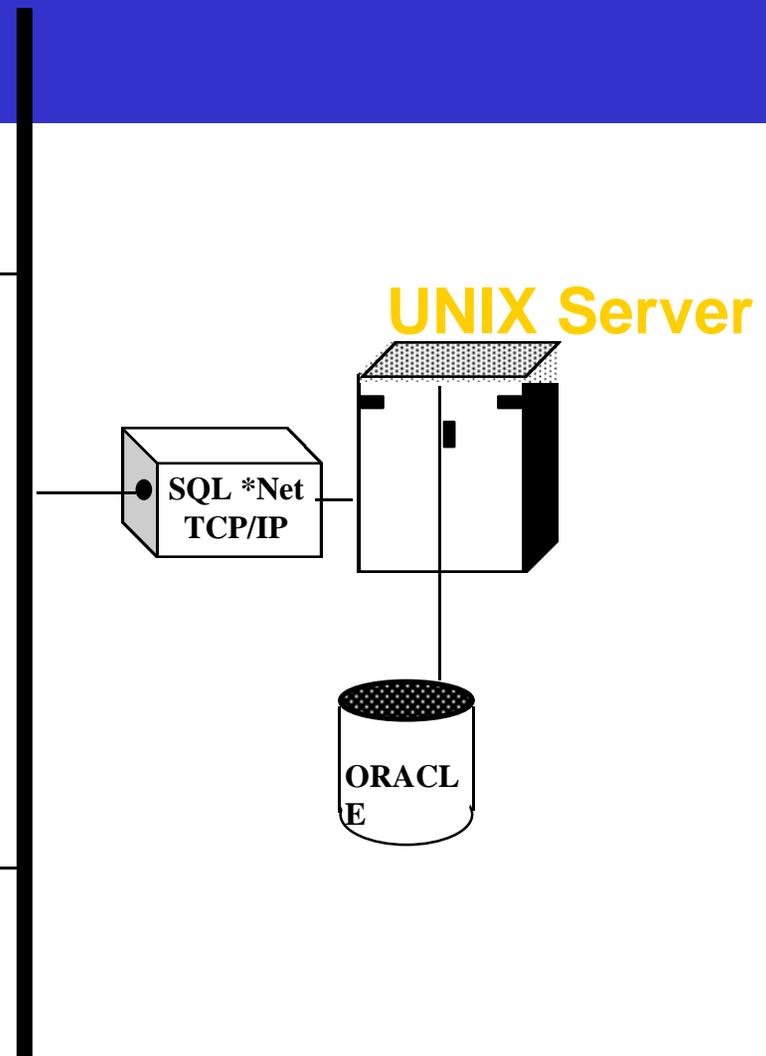
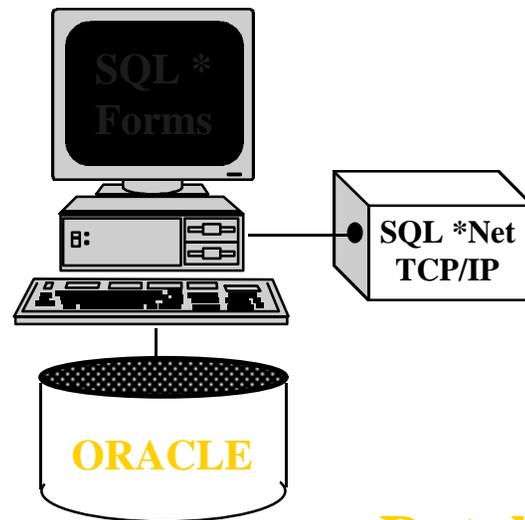
Internet Server



**Distributed processing
application connects to remote
database**

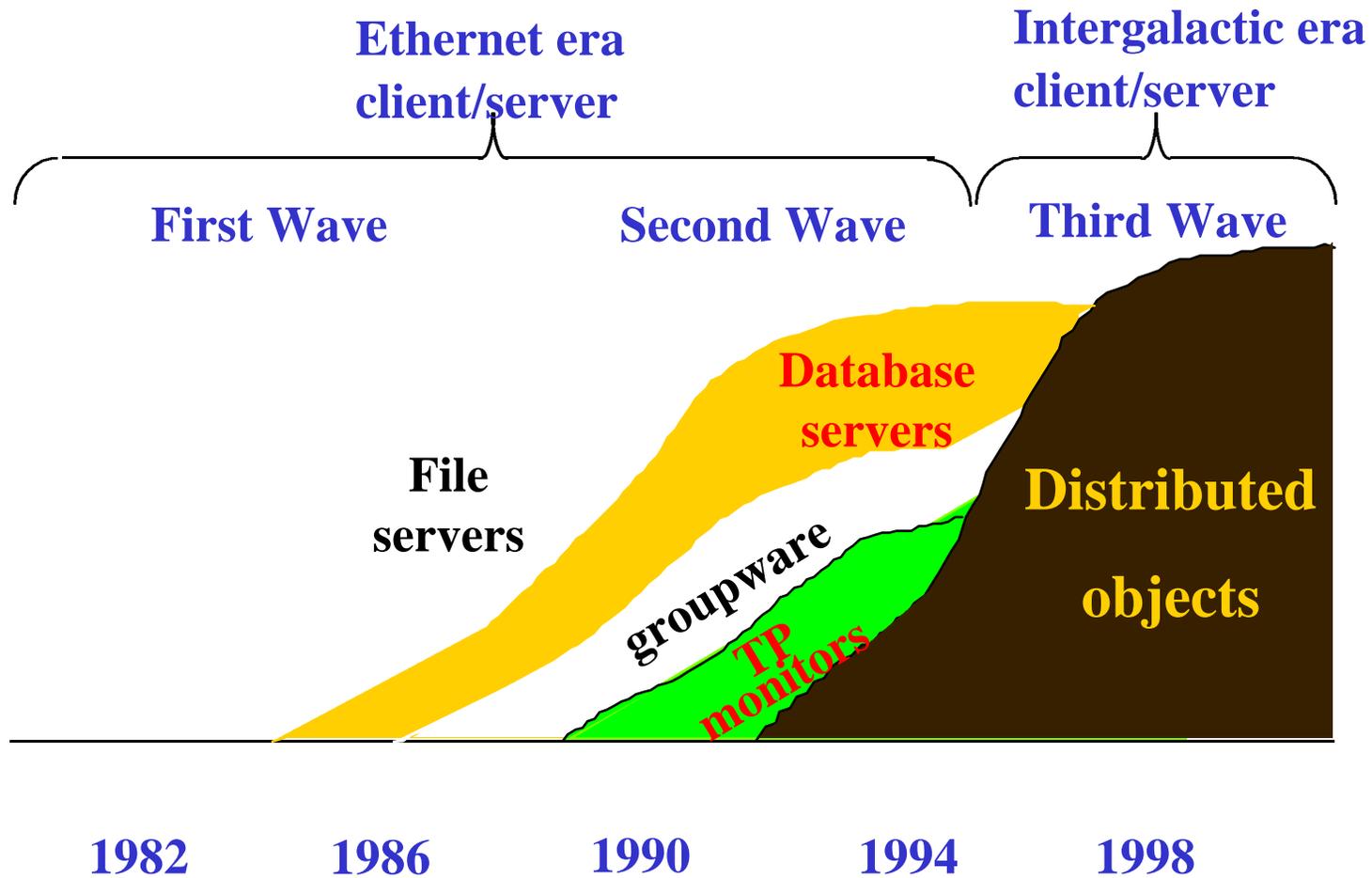


**Distributed database application
connects to local database which
connects to remote database**



Database Configurations

Client-Server Waves



The Client/Server Infrastructure

