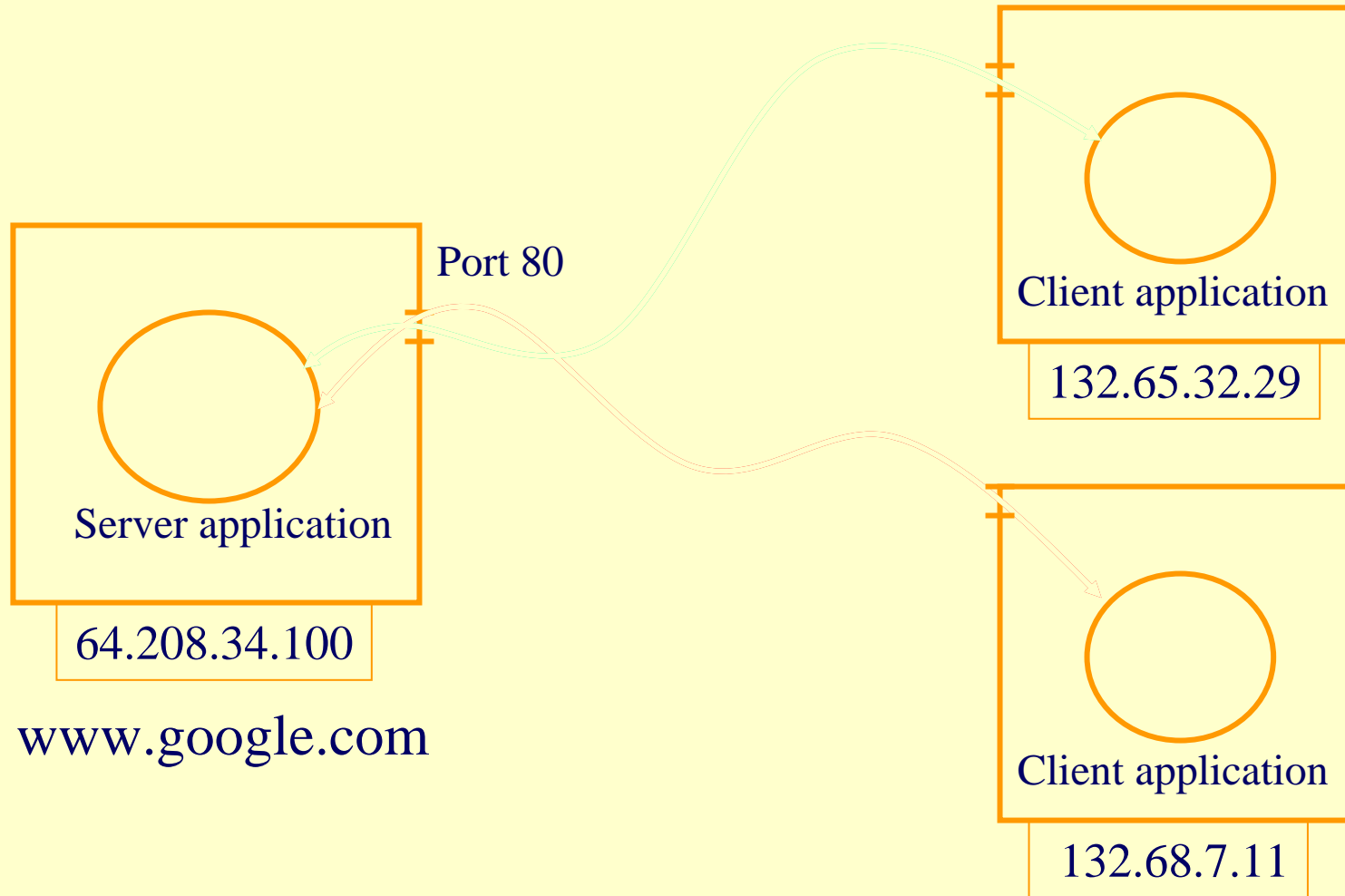


Networking in Java

Client-Server Model



Clients

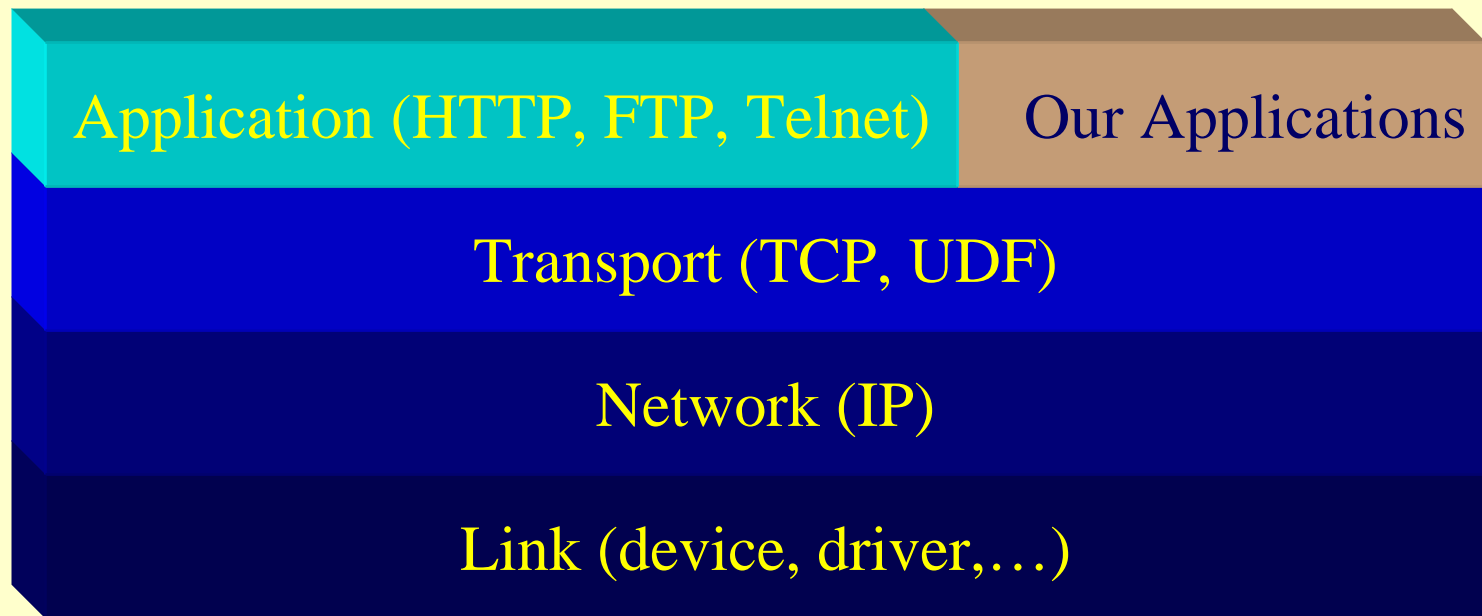
- Client - initiates connection
 - retrieves data
 - displays data
 - responds to user input
 - requests more data
- Examples:
 - Web Browser
 - Chat Program
 - PC accessing files

Servers

- **Server - responds to connection**
 - receives request for data
 - looks it up
 - delivers it
- **Examples:**
 - Web Server
 - Database Server
 - Domain Name Server

Networking Basics

- Communication layers:

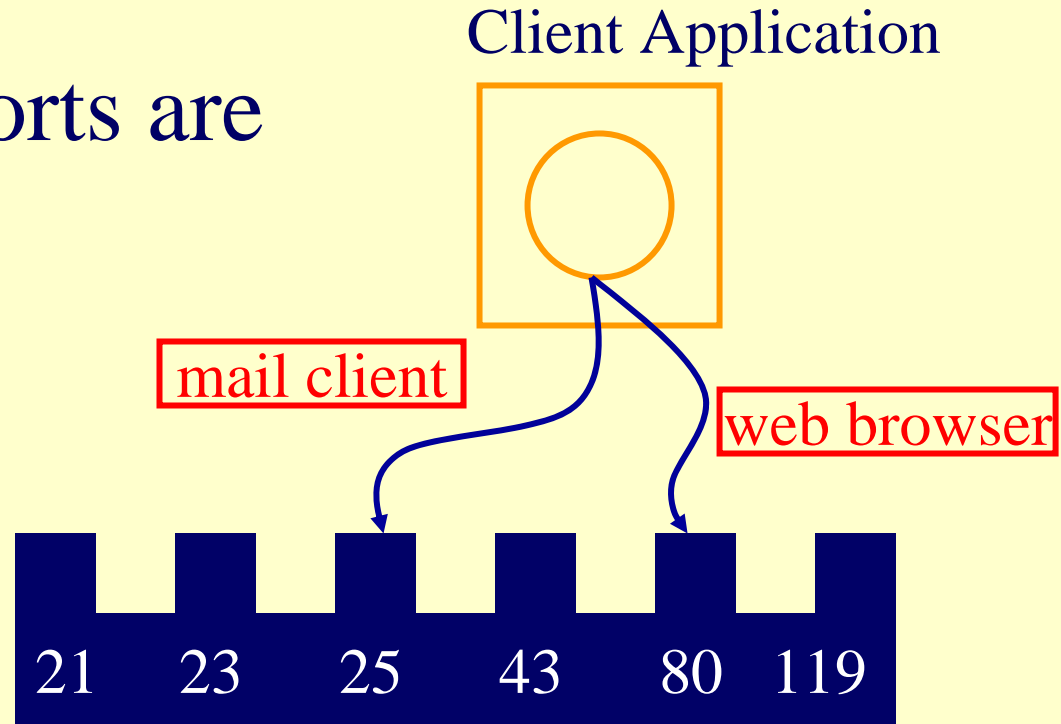


Host and Port

- Destination in the Internet is identified by
host + port
 - a 32 bits IP-address
 - a 16 bits port
- Q: Why don't we specify the port in a Web browser?
- Ports 0-1023 are restricted
 - Do not use them

Known Ports

- Some known ports are
 - 20, 21: FTP
 - 23: telnet
 - 25: SMTP
 - 43: whois
 - 80: HTTP
 - 119: NNTP



Internet Addresses

- InetAddress – a final class that represents Internet Protocol (IP) addresses and the names of hosts
- Getting the InetAddress
 - `getLocalHost`
 - Returns the local host
 - `getByName(String host)`
 - For the given host name one of its IP addresses is returned
 - `getAllByName(String host)`
 - For a given host name all its IP addresses are returned
- `getHostAddress`
 - Returns the IP address of the host
 - The address is in the form “%d.%d.%d.%d”
- `getHostName`
 - Returns the name of the host

Working with URLs

- URL (*Uniform Resource Locator*) - a reference (an address) to a resource on the Internet

http://www.cs.huji.ac.il:80/~dbi/main.html#notes

Protocol

Host
Name

Port
Number

File
Name

Reference

Creating URLs

- The class URL is defined in the package

java.net

- Basic constructor:

```
URL w3c = new URL("http://www.w3.org/");
```

- Relative links:

- Are created from baseURL + relativeURL

```
URL amaya = new URL(w3c, "Amaya/Amaya.html");
```

```
URL jigsaw = new URL(w3c, "Jigsaw/#Getting");
```

- The following two are equivalent:

```
URL dbiNotes = new URL("http://www.cs.huji.ac.il:80/" +  
    "~dbi/main.html#notes");
```

```
URL dbiNotes = new URL("http", "www.cs.huji.ac.il", 80,  
    "~dbi/main.html#notes");
```

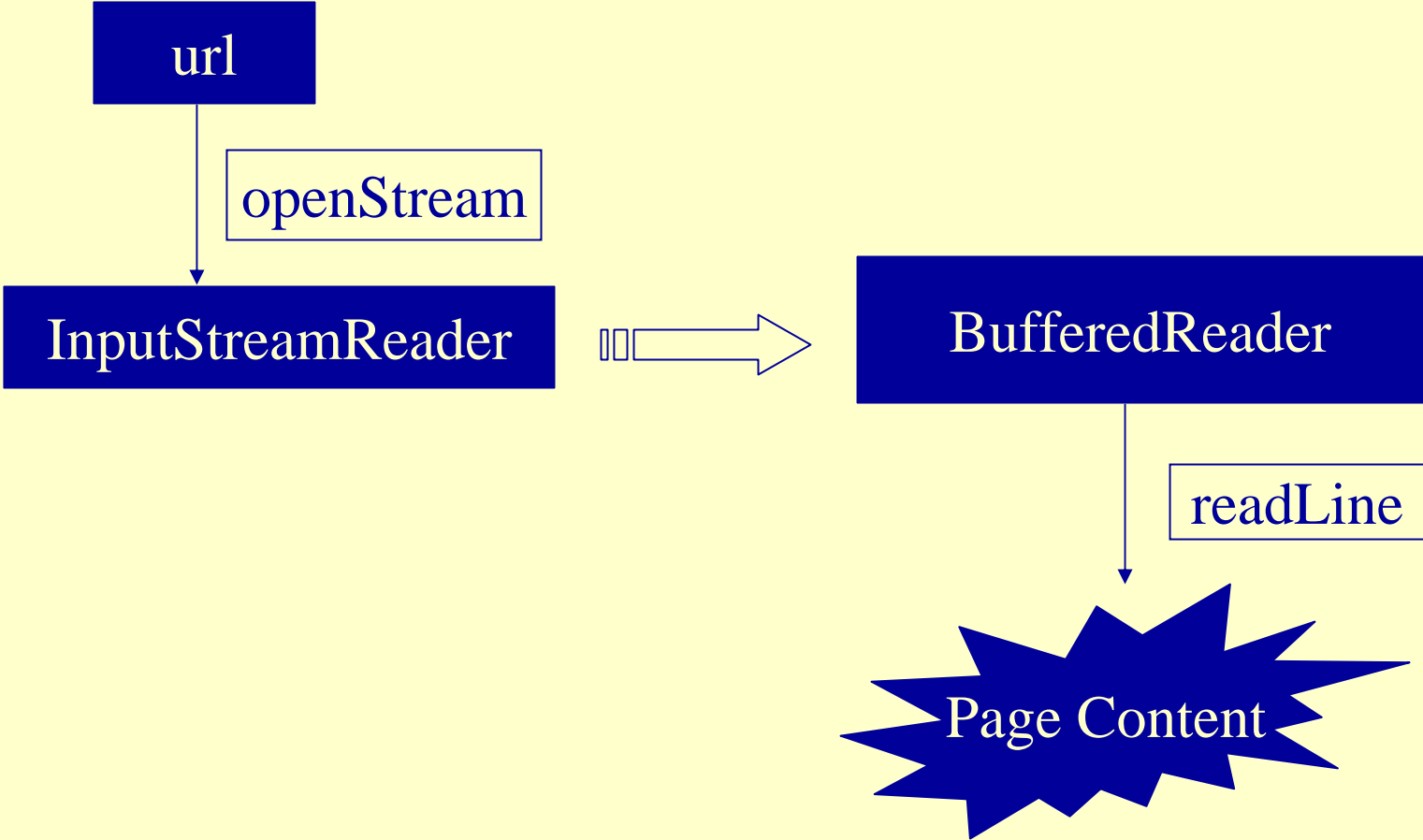
- **The main usage of URL is for parsing *URLs***

- getting the protocol
- getting the host
- getting the port
- getting the file name
- getting the reference

- Construction of URLs can throw

MalformedURLException

Reading From A URL



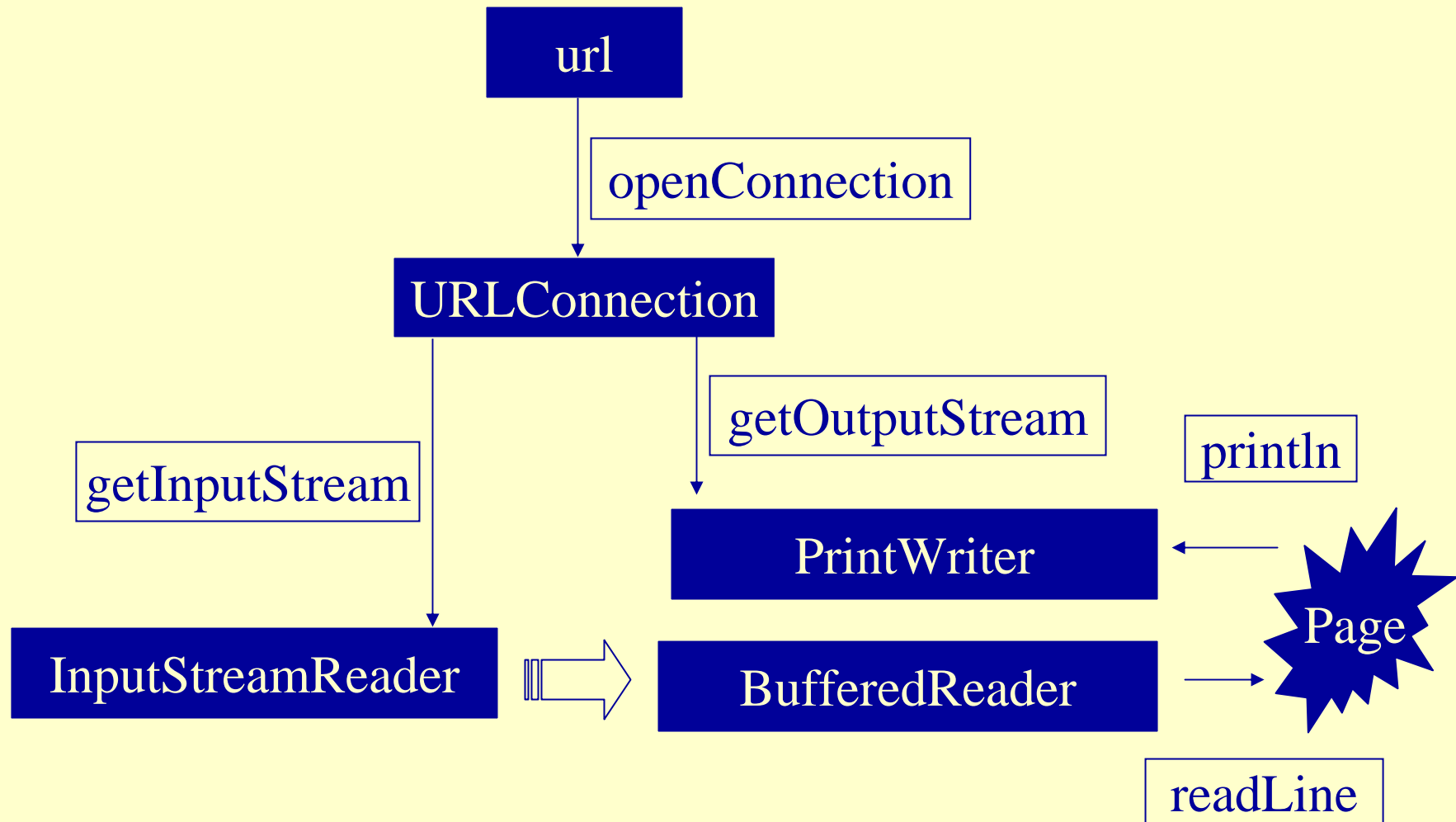
```
import java.io.*;
import java.net.URL;

// Types the content of a given web page
// Usage: java TypeWebPage <url>
class TypeWebPage {

    public static void main(String[] args) {
        try {
            URL url = new URL(args[0]);
            BufferedReader reader =
                new BufferedReader(new InputStreamReader(url.openStream()));

            String line;
            while ((line = reader.readLine())!=null) {
                System.out.println(line);
            }
        } catch (IOException ioe) {
            System.err.println("Reading failed");
        } catch (ArrayIndexOutOfBoundsException abe) {
            System.err.println("Wrong usage.");
        }
    }
}
```

Connecting to A URL



Interacting with a CGI script

1. Create a URL
2. Open a connection to the URL
3. Set output capability on the URLConnection
4. Get an output stream from the connection
 - This output stream is connected to the standard input stream of the `cgi-bin` script on the server
5. Write to the output stream
6. Close the output stream

HTTP connections

- You can create a connection to an HTTP server with an object

HttpURLConnection

- This object extends the **URLConnection** object

- **getResponseCode**
- **getResponseMessage**
- **setRequestMethod**

- Look in the Java API



HTTP/1.1 200 OK

<HTML>

...

</HTML>

URLEncoder

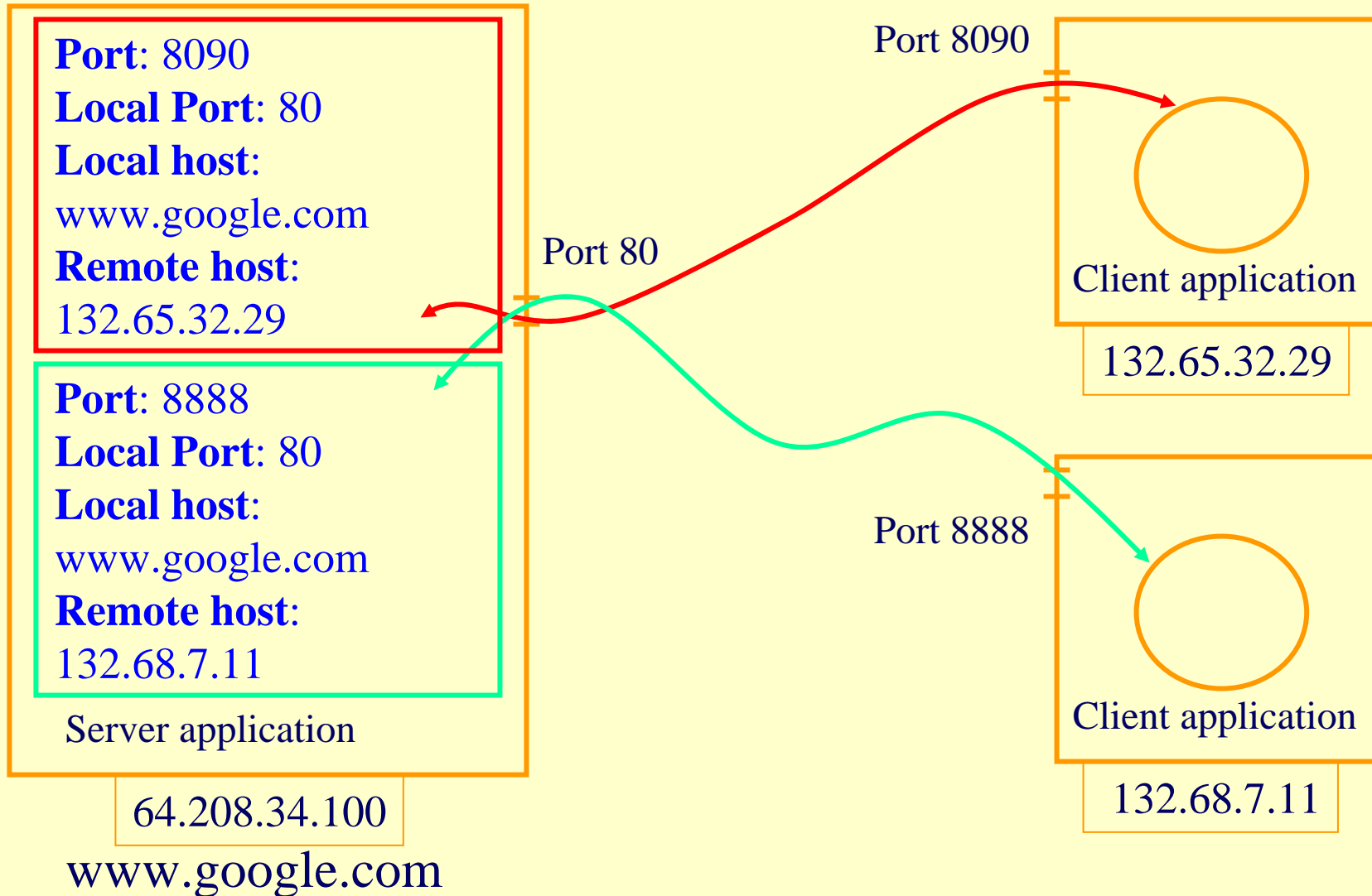
- contains a utility method **encode** for converting a string into an encoded format
- To convert a string, each character is examined in turn:
 - The ASCII characters 'a' – 'z', 'A' – 'Z', '0' – '9', ".", "-", "_", "*" remain the same
 - Space is converted into a plus sign '+'
 - All other characters are converted into the 3-character string "%xy", where xy is the two-digit hexadecimal representation of the lower 8-bits of the character

URL Connection Example

- The next example connects to a CGI script on *www.walla.co.il* – a search tool is given a word to search

SearchWalla

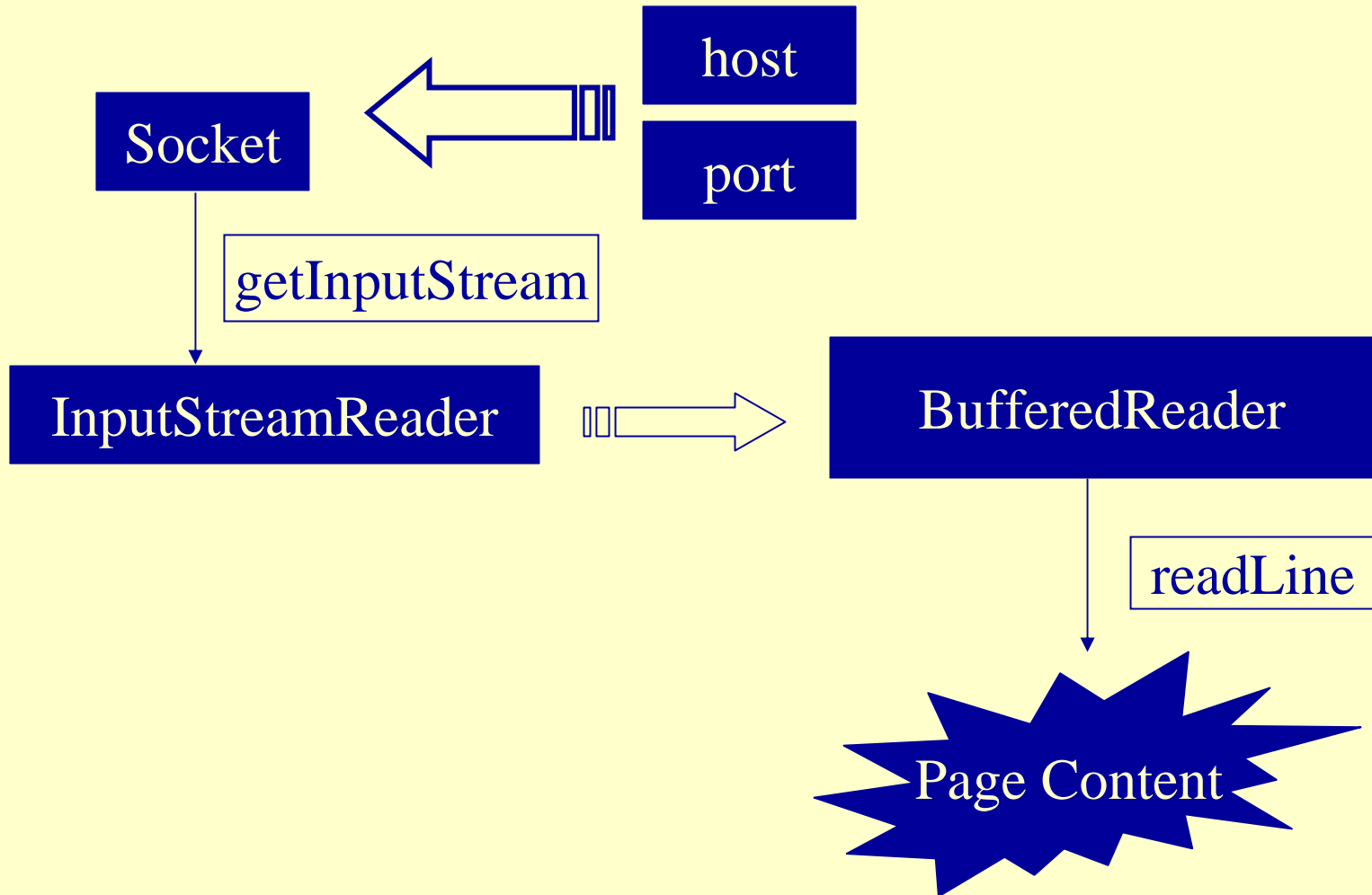
Client Requests for a Connection



Socket

- Class `Socket` – implements the client side of the connection
- Class `ServerSocket` – implements the server side of the connection

Using a Socket



Using a Socket

```
// Constructors (partial list)
public Socket()
public Socket(InetAddress address, int port);
public Socket(String host, int port);

// Methods (partial list)
public void close();


public InetAddress getInetAddress();
public int getLocalPort();

public InputStream getInputStream();
public OutputStream getOutputStream();

public int getPort();
public String toString();
```

ServerSocket

The size of requests queue with default of 50 requests



```
// Constructors (partial list)
```

```
public ServerSocket(int port);  
public ServerSocket(int port, int count);
```

```
// Methods (partial list)
```

```
public Socket accept();  
public void close();
```

```
public InetAddress getInetAddress();  
public int getLocalPort();
```

```
public String toString();
```

What happens when a running program reaches accept()?

Why don't we have getPort in addition to getLocalPort?

More on Server Socket

- A **ServerSocket** waits for requests to come in over the network
- It performs some operation based on that request, and then possibly returns a result to the requester
- The actual work of the **ServerSocket** is performed by an instance of the **SocketImpl** class
- The abstract class **SocketImpl** is a common superclass of all classes that actually implement sockets
- It is used to create both client and server sockets

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

Server side

```
// A server that says 'hello'
// note that it cannot compile as is - missing exceptions
class HelloServer {

    public static void main(String[] args) {
        int port = Integer.parseInt(args[0]);
        ServerSocket server = null;
        try {
            server = new ServerSocket(port);
        } catch (IOException ioe) {
            System.err.println("Couldn't run"+ "server on port "+port);
            return;
        }
        while(true) {
            try {
                Socket connection = server.accept();
                BufferedReader reader =
                    new BufferedReader(new InputStreamReader(connection.getInputStream()));
                PrintWriter writer =
                    new PrintWriter(new OutputStreamWriter(connection.getOutputStream()));

                String clientName = reader.readLine();
                writer.println("Hello "+clientName);
                writer.flush();
            } catch (IOException ioe1) {}
        }
    }
}
```


Client side

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

// A client of an HelloServer
class HelloClient {

    public static void main(String[] args) {
        String hostname = args[0];
        int port = Integer.parseInt(args[1]);

        Socket connection = null;
        try {
            connection = new Socket(hostname, port);
        } catch (IOException ioe) {
            System.err.println("Connection failed");
            return;
        }

        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                connection.getInputStream()));
            PrintWriter writer = new PrintWriter(new OutputStreamWriter(
                connection.getOutputStream()));

            writer.println(args[2]); // client name
            String reply = reader.readLine();
            System.out.println("Server reply: "+reply);
            writer.flush();
        } catch (IOException ioel) {
        }
    }
}
```

Datagrams

- Datagram packets are used to implement a connectionless, packet based, delivery service
- Each message is routed from one machine to another based solely on information contained within that packet
- Multiple packets sent from one machine to another might be routed differently, and might arrive in any order
- Packets may be lost or duplicated during transit
- The class **DatagramPacket** represents a datagram in Java

DatagramPacket Class

```
//Constructors
```

```
public DatagramPacket(byte ibuf[], int ilength);  
public DatagramPacket(  
    byte ibuf[], int ilength, InetAddress iaddr, int  
    iport);
```

```
// Methods
```

```
public synchronized InetAddress getAddress();  
public synchronized int getPort();  
public synchronized byte[] getData();  
int getLength();
```

```
void setAddress(InetAddress iaddr);  
void setPort(int iport);  
void setData(byte ibuf[]);  
void setLength(int ilength);
```

DatagramSocket

- This class represents a socket for sending and receiving datagram packets
- Addressing information for outgoing packets is contained in the packet header
- A socket that is used to read incoming packets must be bound to an address (sockets that are used for sending must be bound as well, but in most cases it is done automatically)
- There is no special datagram server socket class
- Since packets can be lost, the ability to set timeouts is important

Class DatagramSocket

// Constructors

DatagramSocket()

DatagramSocket(int port)

DatagramSocket(int port, InetAddress iaddr)

// Methods

void close()

InetAddress getLocalAddress()

int getLocalPort()

int getSoTimeout()

void receive(DatagramPacket p)

void send(DatagramPacket p)

setSoTimeout(int timeout)

Echo Servers

- A common network service is an echo server
- An echo server simply sends packets back to the sender
- A client creates a packet, sends it to the server, and waits for a response
- Echo services can be used to test network connectivity and performance

Echo Client Example

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

public class EchoClient {
    static int echoPort = 7; static int msgLen = 16; static int timeOut=1000;

    public static void main(String argv[]) {
        try {
            DatagramSocket socket = new DatagramSocket();
            DatagramPacket packet;
            byte msg[] = new byte[msgLen];

            InetAddress echoHost = InetAddress.getByName(argv[0]);
            packet = new DatagramPacket(msg,msgLen,echoHost,echoPort);

            socket.send(packet);
            socket.setSoTimeout(timeOut);
            socket.receive(packet);
        }
        catch (InterruptedException e) {System.out.println("Timeout");}
        catch (Exception e) {}}}
```

EchoServer

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

public class EchoServer {
    static int echoPort = 7000; static int msgLen = 1024;

    public static void main(String args[]) {
        try {
            DatagramSocket socket = new DatagramSocket(echoPort);
            DatagramPacket p,reply;
            byte msg[] = new byte[msgLen];
            packet = new DatagramPacket(msg,msgLen);

            for (;;) {
                sock.receive(p);
                System.out.println(p.getAddress());
                reply =
                    new DatagramPacket(p.getData(),p.getLength(),p.getAddress(),p.getPort());
                socket.send(reply);
            }
        } catch (Exception e) {} }}
}
```


Java Net Classes

Class	Description
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
InetAddress	This class represents an Internet Protocol (IP) address.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
URL	A pointer to a "resource" on the World Wide Web.
URLConnection	The superclass of all classes that represent a communications link between an application and a URL.