Concurrent Programming (Part II)
Lecture 13: Client/Server Architecture using Java Sockets

Dr Kevin Bryson

K.Bryson@cs.ucl.ac.uk

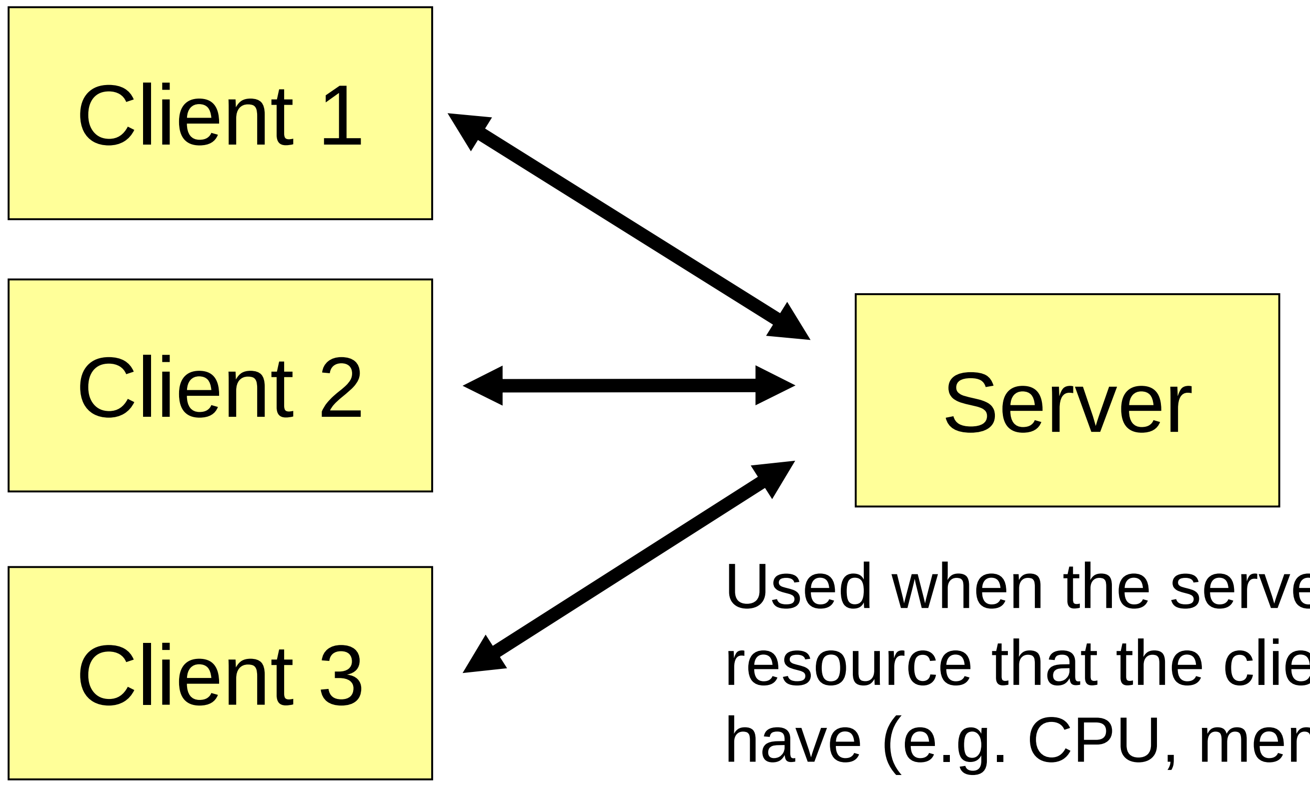
Room 8.04

Course Web Site on Moodle
http://moodle.ucl.ac.uk/course/view.php?id=753
Enrolment Key: ATOMIC

Overview

- The last lecture provided a rather brief overview of lots of aspects of distributed systems, introducing the Java socket classes towards the end, and I asked you to look at their APIs for this lecture.
- In this lecture we will review in more detail the Java socket classes and work through a client-server system written using them.
- We will then build our very own secure “private” network using 2 portables and a network hub – this will allow us to try out our Java client/server application.
- By the end of this lecture you should have a good idea how your web browser works in terms of networking.

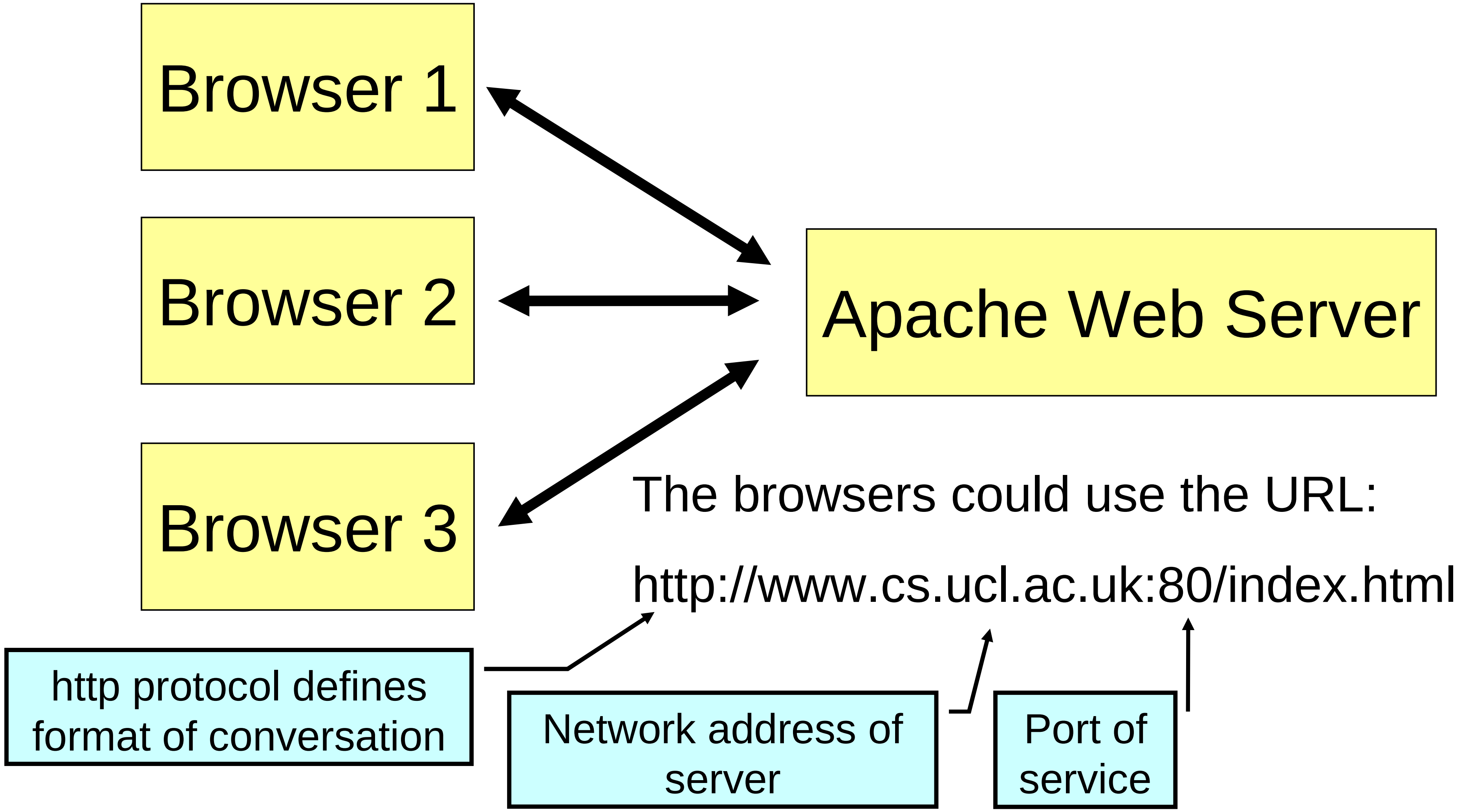
Client/Server Architecture



Used when the server has some resource that the clients do not have (e.g. CPU, memory, disk, executables, web page files, etc.)

The server usually services multiple clients.

Client/Server Architecture – for example a web server and multiple web browser clients.


Browser 1
Browser 2
Browser 3
Apache Web Server
http protocol defines format of conversation
Network address of server
Port of service

The browsers could use the URL:

http://www.cs.ucl.ac.uk:80/index.html

Recall Network addresses

- Network is a collection of nodes connected by a communication medium (electric wires, optical cables, radio transmitters for wireless networks, …)
- The Internet uses the Internet Protocol (IP) which specifies how machines should address and communicate with each other at a low level.
- Each machine on the network is given a unique IP address consisting of 4 digits from 0 to 255 (i.e. **128.16.6.8**).
- It is more convenient to specify machines in terms of their textual name, such as **www.cs.ucl.ac.uk**, this is resolved into the appropriate IP address using a Domain Name Server (DNS).
- Our demo machines will have manually configured IP addresses of **169.254.76.207** and **169.254.76.200**.

Recall Port Numbers

- Server processes listen to different **port numbers** on the server machine. This is essential to allow multiple servers to run on one machine.
- Client processes communicate to server processes by specifying the IP address of the server machine and also the port number.
- Common server processes, such as web servers, listen to well known port numbers so that particular types of clients, in this case web browsers, know which port to connect to by default (in this case port 80).
- Our remote file server example will use port 9999 which is not commonly used by any other server (only one server process can listen to a particular port at any given time).

Key Classes & Key Methods

<u>Class Socket</u>

`Socket socket = new Socket(host, port)`

i.e. `new Socket(“www.cs.ucl.ac.uk”, 80)`

To communicate between a pair of sockets:

`socket.getInputStream()` … to get the input stream.
`socket.getOutputStream()` … to get output stream.

<u>Class ServerSocket</u>

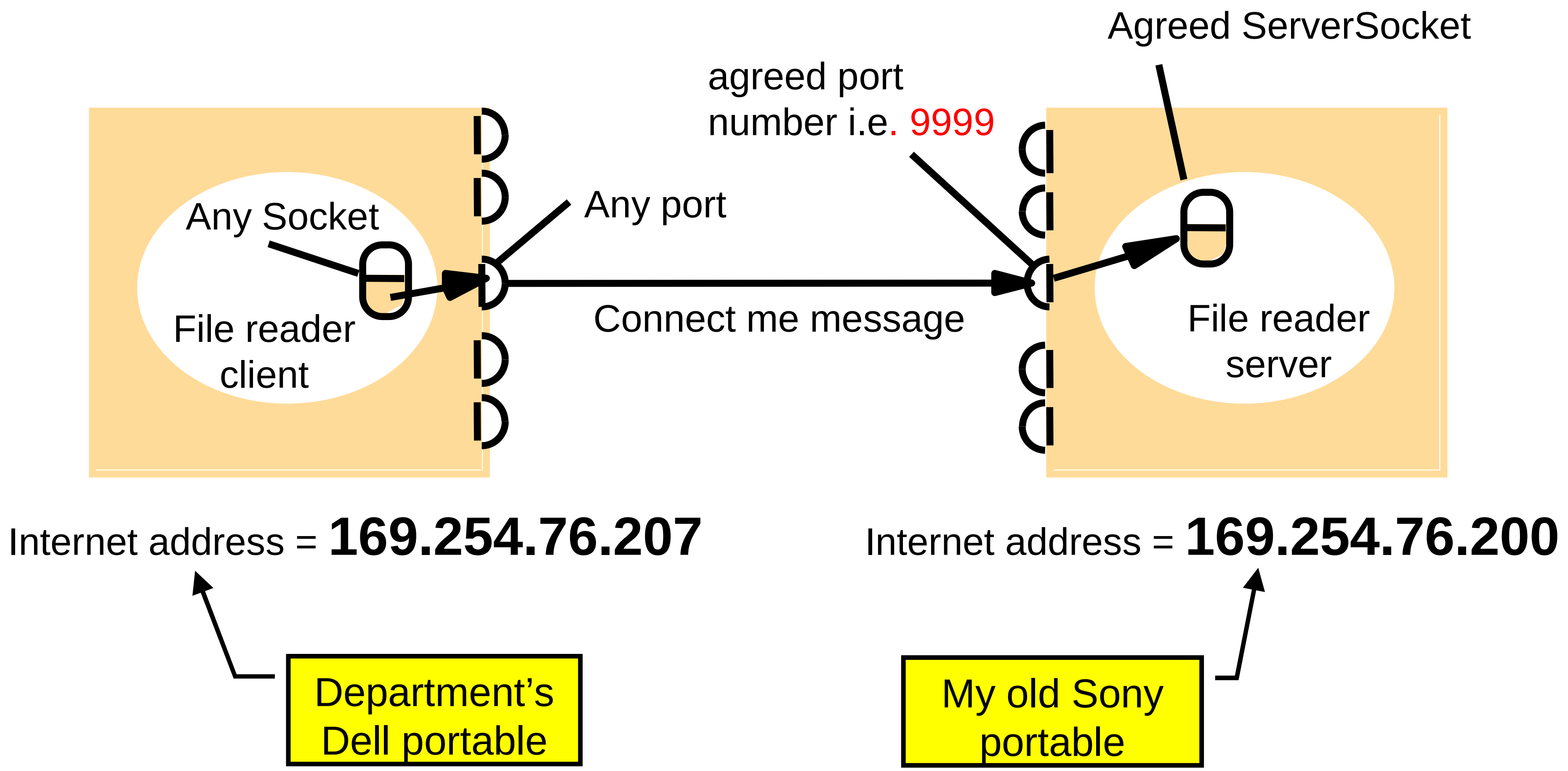
`SocketServer listen = new SocketServer(port)`

i.e. `new SocketServer(80)`

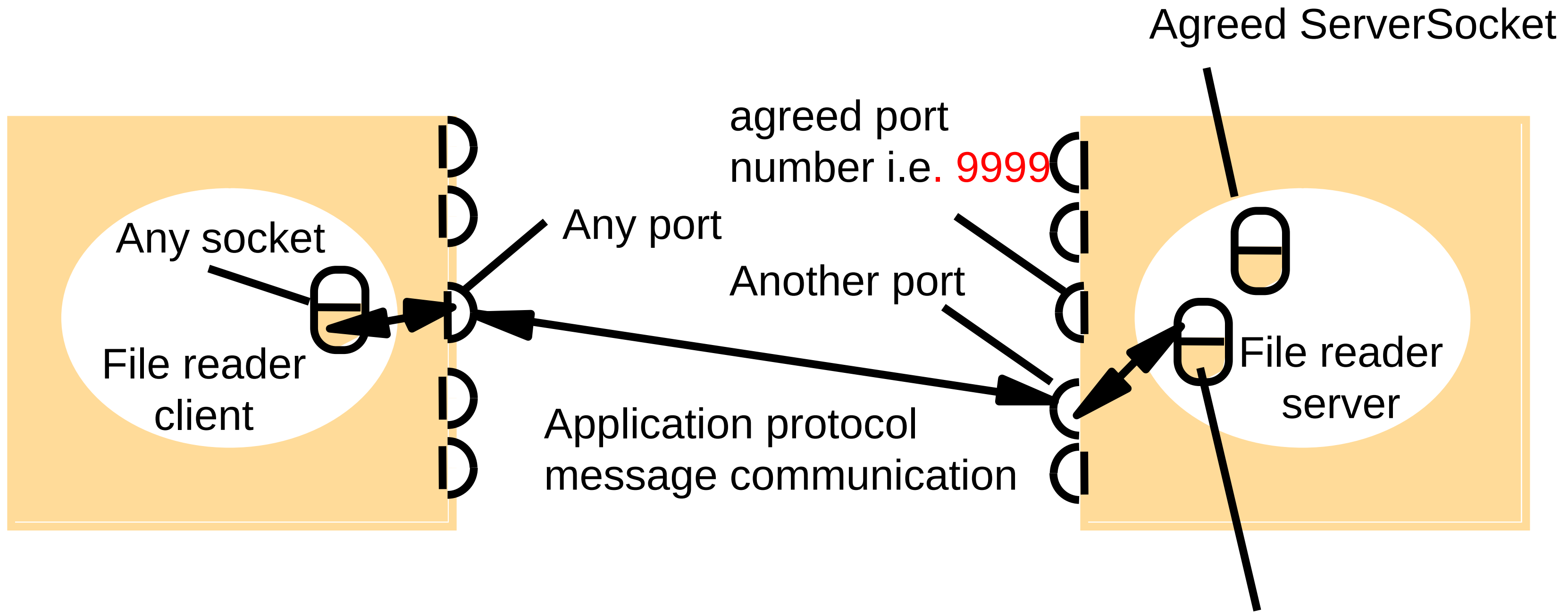
To accept a client request for communication:

`listen.accept()` … returns the server-side socket, thus allowing communication with the client side socket (i.e. between a pair of sockets).

Java socket communication - accepting a request …


Agreed ServerSocket
agreed port number i.e. 9999
Any port
Any Socket
File reader client
Connect me message
File reader server
Internet address = 169.254.76.207
Internet address = 169.254.76.200
Department's Dell portable
My old Sony portable

Java socket communication - processing a request



Socket created by the ServerSocket `accept` method. This is used process the request.

Remote File Reader Demo

- A Java program can read and write files on its local disk using the java.io package.
- This demo allows a client machine to access files on a server machine’s disk using a client/server framework.
- When the client process wants to read a file it needs to communicate over the network with the server.
- The client process sends the name of the file to be read.
- The server ‘accepts’ the request and reads the filename of the file required.
- The server then reads the lines of the required file, sending them back to the client process across the socket connection.
- This can be used to read secret files on a server machine like “secret_rendezvous.txt” (using a ‘Trojan Horse’ …)
- Let us first read the code, then build our private communication network and try the demo …

Questions

- What happens if the client starts before the server?
- What happens if two clients try to connect to the server at about the same time? Limitations?
- Why does the server appear multi-threaded in this case when serving two clients?
- Could we use individual threads to serve each client connection? A multithreaded server …

Summary

- We have analysed in some detail the example code for a socket-based client/server system.

- We have examined its performance when satisfying multiple clients.

- In the next lecture we will look at creating a multi-threaded server and the use of Thread Pools for optimization.
- Read through the multi-threaded version of the server that I’ve handed out for the next lecture noting any differences.