

Concurrent Programming (Part II)

Lecture 12: Introduction to Distributed Systems

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

Aim of Lecture

- In the last lecture we examined fundamental abstract concepts about message passing mechanisms:
 - Synchronous (client blocking)
 - Asynchronous (client non-blocking)
 - Rendezvous
- This lecture will provide a more concrete overview of real networks and distributed systems (I'm going to cover it fairly quickly since it's mostly background material ...)
 - Examples, benefits ... and problems
 - Key networking concepts
 - Quick introduction to Java Socket class

But first a historical perspective of the Internet ...

Date	Computers	Web Servers
1969, Apr.	0	0
1979, Dec.	188	0
1989, July	130,000	0
1999, July	56,218,000	5,560,866
2009, July	?	233,000,000

Distributed computing is now everywhere ...

1. Quad-band GSM (Global System for Mobile Communications) using GPRS/EDGE (General Packet Radio Service)
2. WiFi WLAN 802.11b/g
3. Bluetooth
4. USB 2.0 ... it's still distributed networking!



And that's only one device !

Services & Protocols

Email (SMTP, POP, IMAP)

Web (HTTP, HTML, ...)

Network people like acronyms !



Unfortunately concurrent Distributed Systems are even harder to program robustly than concurrent shared memory systems ...

- We *not only* need to worry about concurrency aspects when programming distributed systems ...
- But we also need to worry about a lot of other aspects as well ...
- ***In particular, we are no longer in control of the complete behaviour of the system ... we need to consider lots of unusual scenarios ...***

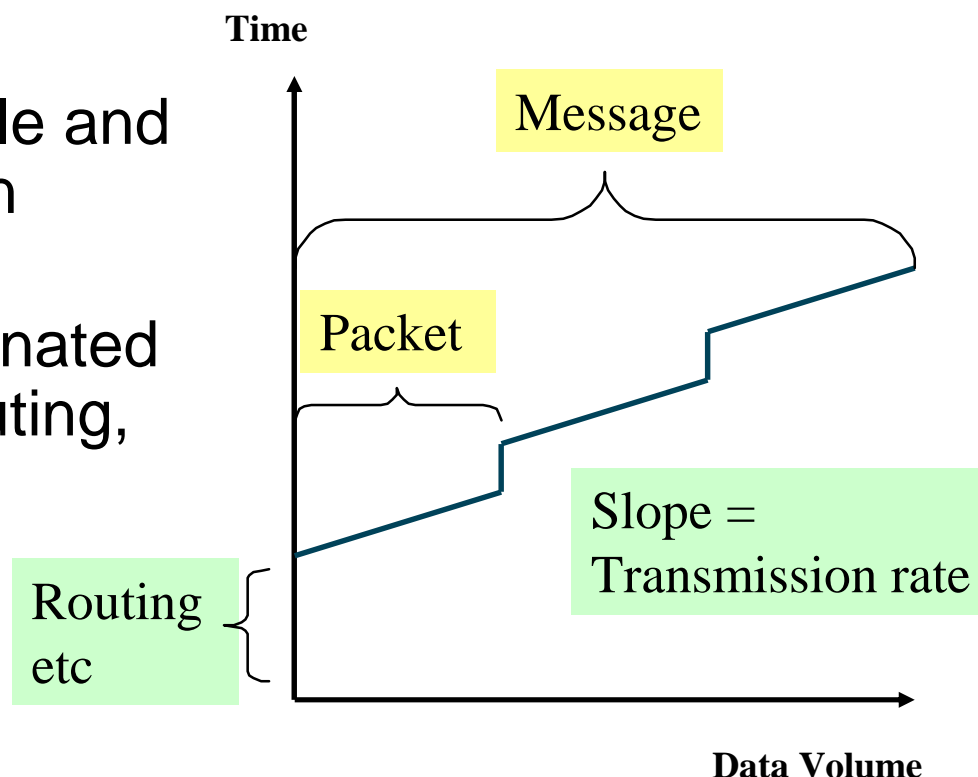
The 7 fallacies of distributed computing ...

When you first build a distributed system you tend to make the following assumptions:

1. Latency is zero
2. Bandwidth is infinite
3. The network is reliable
4. The network is secure
5. Network topology doesn't change
6. There is one administrator
7. Transport cost is zero

Fallacy 1 and 2 – *Latency is 0 & Bandwidth infinite*

- Networks have very variable and non-negligible transmission times
- Short message times dominated by OS latency, network routing, and congestion
- Long messages influenced by volume of data (network bandwidth)



WHY DOES IT MATTER?

It is often impossible to distinguish a failed component from just a slow one.

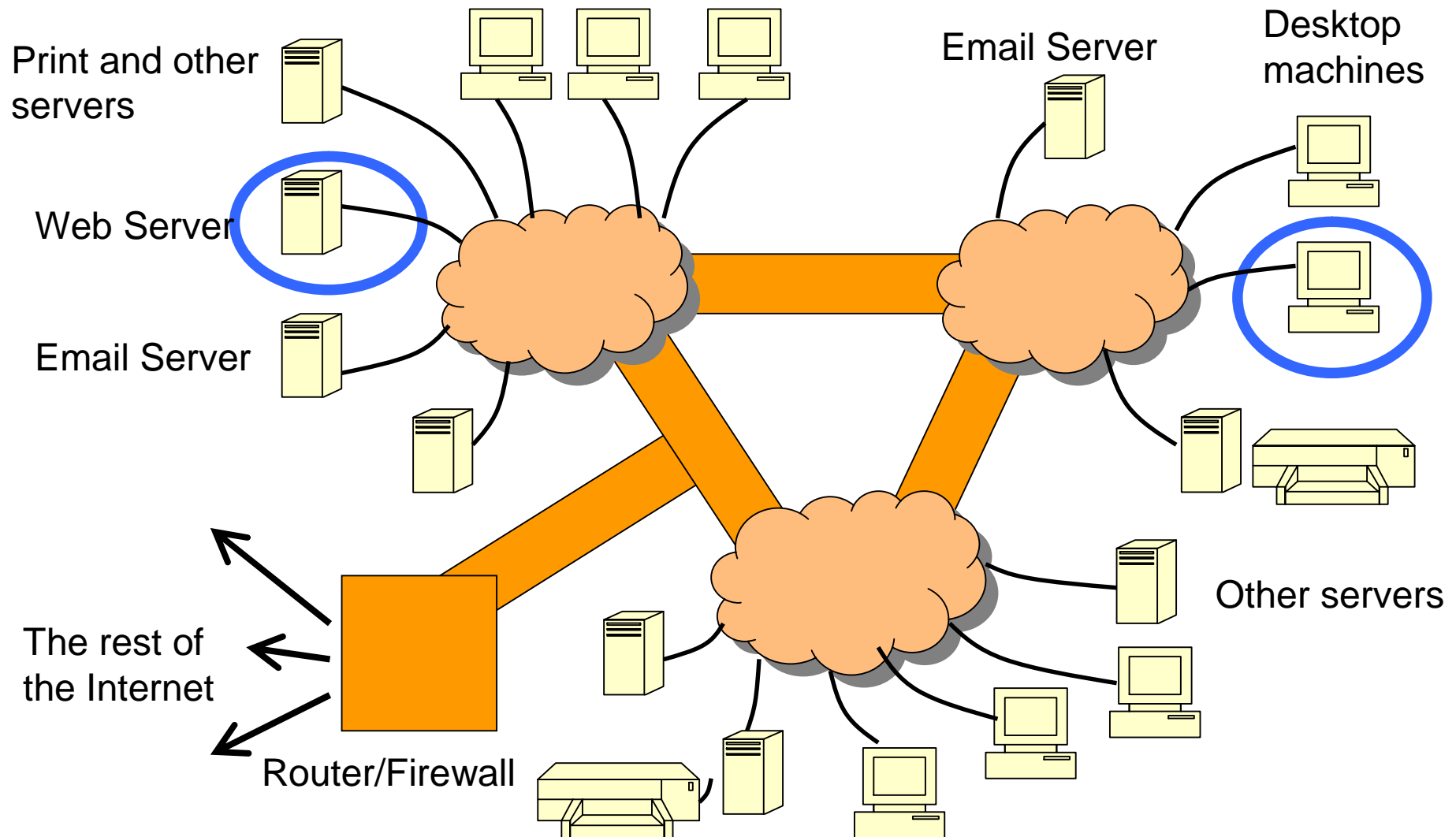
Fallacy 3 – *The network is reliable*


- “NFS server not responding” ... “No route to host” ... etc.
- Individual processes on machines may fail due to bugs, etc.
- Individual machines may fail due to: crashes, power outages, *air conditioning problems*, ...
- Similarly, network components (e.g. routers) may fail (for instance when people steal the network switches ... !)
- Generally it is impossible to tell if the failure occurred before or after carrying out a particular operation ... or if the network is just being slow.
- **Partial failures** are the *most problematic* ... how do you maintain consistency when only one part of a system has failed? Particularly if other parts have not detected the failure ...

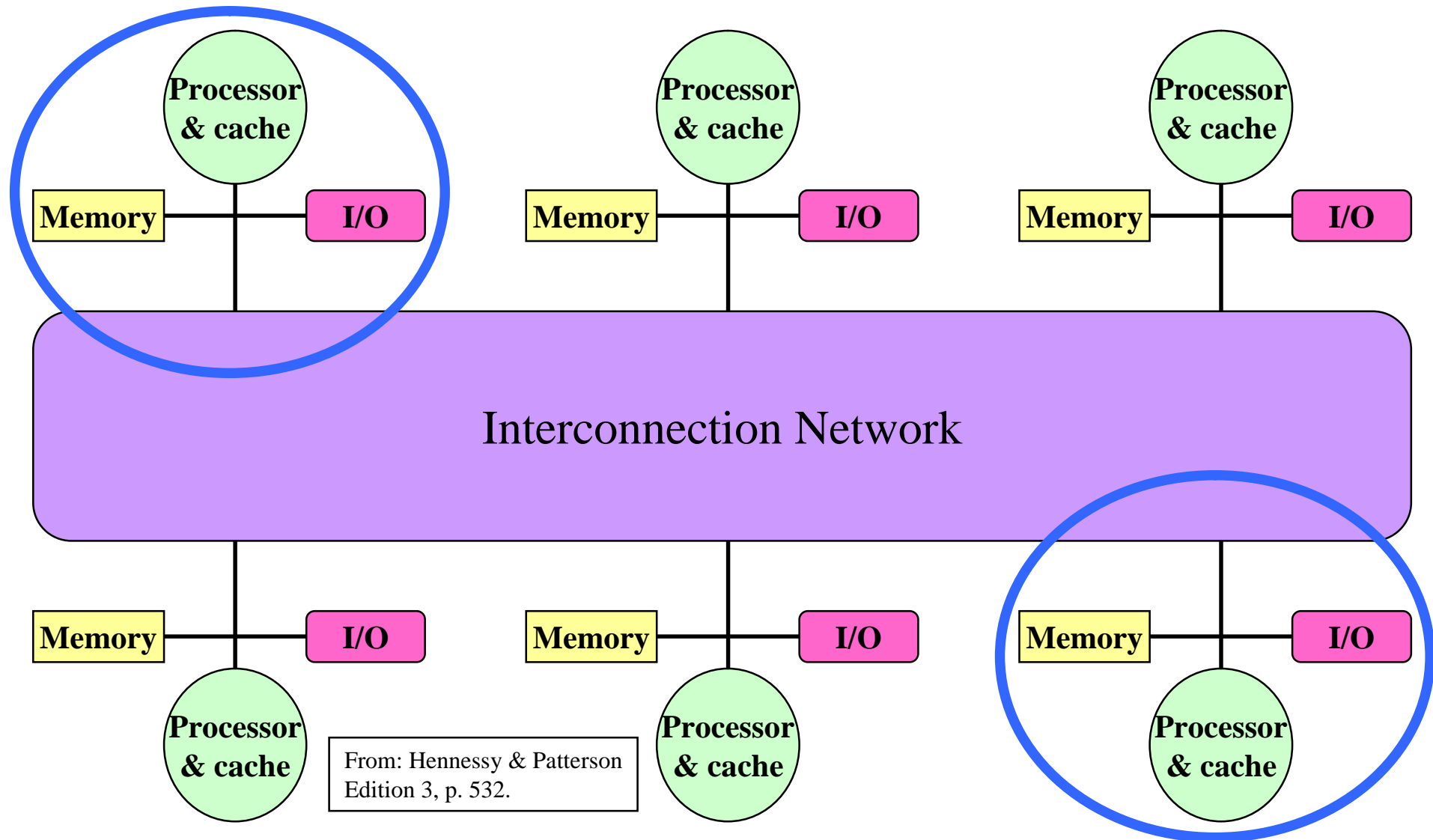
Partial Failures

- Machines fail *independently*.
- This can also be seen as a virtue in well designed systems, leading to greater availability and robustness.
- Need for decentralised algorithms avoiding *single point of failure*. Need for careful system and algorithm design – *a total system design*.
- *But partial failure is also one of the hardest things to deal with in distributed systems.*

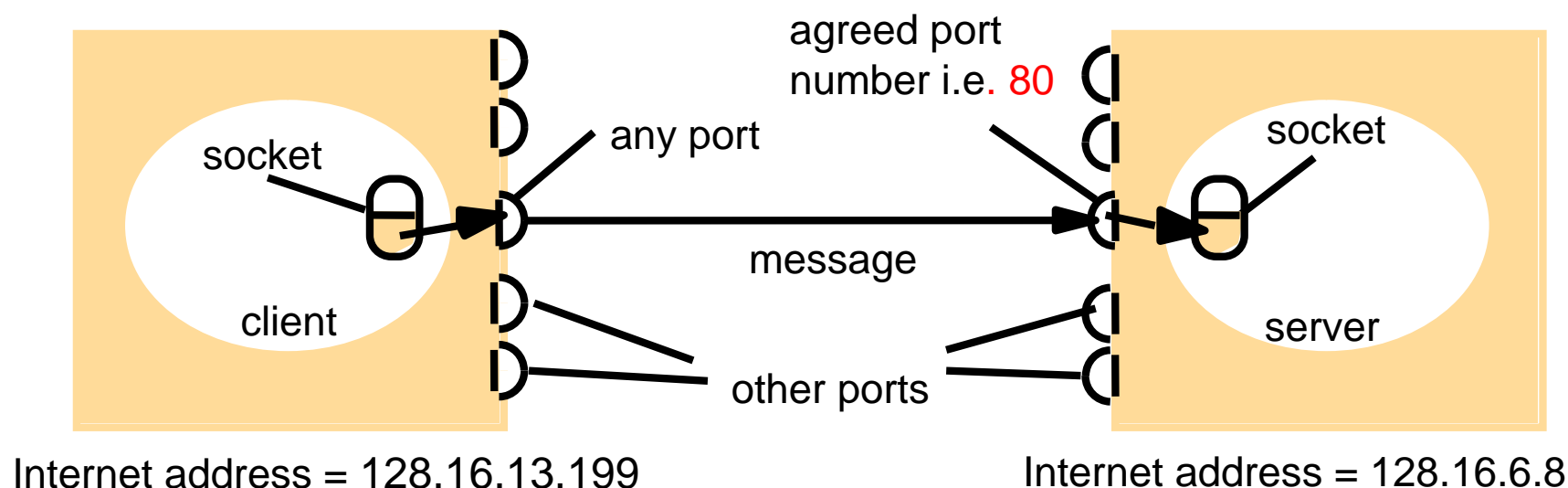
A typical intranet ... somewhere in this intranet a web browser is talking to a web server ...



But we do not need to worry about this  complexity ... we can abstract it away by working with high level network APIs

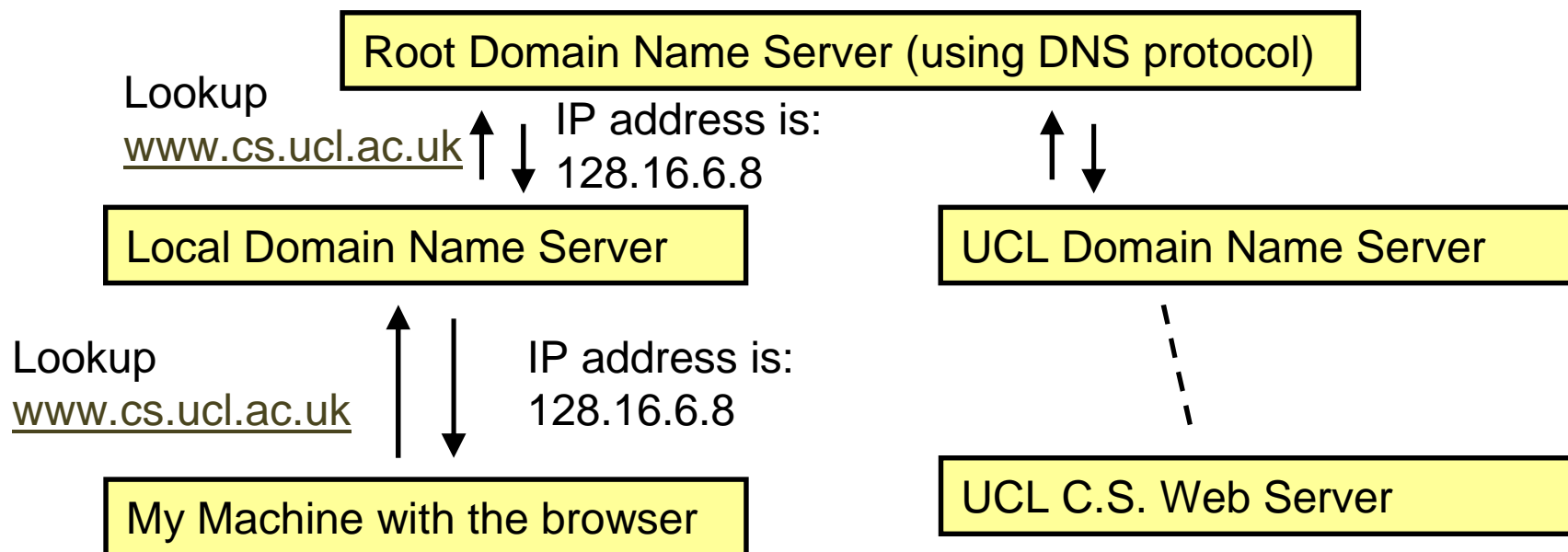


**Key concepts: Each machine is given an 'IP address',
Unique port number used for each 'network service'**



- Different machines on the Internet are located using IP addresses which consist of 4 digits
- For instance, www.cs.ucl.ac.uk has the IP address 128.16.6.8.
- Communication is via 'ports' on the machine which are given different numbers up to 65536.

But how does a machine name get converted to an IP address?



- A machine name like www.cs.ucl.ac.uk is looked up within a Domain Name Server, another distributed system using the DNS protocol.
- Domain Name Servers have a hierarchical organization so that they *scale* well.
- Quite often there are backup Domain Name Servers so that if a primary one fails, a secondary one is used.

What is a port number?

- Port numbers are used for setting up connections
- Generally a machine has 65536 port numbers.
- Port numbers below 1024 are *reserved* for well known services and generally only super-user can listen to them.
- For instance:
 - FTP (port 21) and Telnet (port 23).
 - Simple Mail Transfer Protocol (SMTP) listens to port 25.
 - Web server (HTTP) generally listens to port 80.
- Client applications (browsers) know which ports to connect to.

So what is a socket?

- A socket is defined as an *endpoint for communication*
- Concatenation of IP address and port
- The socket **128.16.6.8:80** refers to port **80** on host **128.16.6.8** (if you want to talk to the computer science web server)
- A communication link consists of a pair of sockets
- Link is two-way ... but essentially just consists of transferring text in a particular protocol (i.e. HTTP).

An introduction to Java Sockets

- Java sockets come in two flavors: server socket and client socket
- Client-Server methodology
 - Server creates `ServerSocket` and “binds” it to a “well known” port
 - Server listens on this socket (blocking)
 - A client that wants to talk to the server must create a `Socket` and connect it to the server socket, given by a `<host, port>` pair
 - When the server accepts it, a bi-direction stream is created for communication between the client & server.

Java Socket classes

Only three classes involved (much simpler than Swing !)

- java.net.InetAddress
- java.net.Socket
- java.net.ServerSocket

For the next lecture I want you to read the APIs for these classes and see if you understand how they are used

In the next lecture we (you !) will network two machines together and demonstrate how a socket-based client-server system works – in particular we will examine concurrency issues ...

Summary

- We have introduced the characteristics of distributed systems
- Introduced key networking concepts (IP address, port number, sockets)
- Mentioned the Java Socket classes
 - java.net.InetAddress
 - java.net.Socket
 - java.net.ServerSocket
- Have a look at these classes for the next lecture.