Concurrent & Distributed Systems

Distributed Systems 1

Introduction to distributed systems

resources for private study

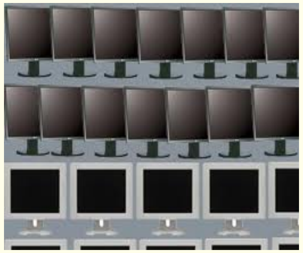
- **Books**

- *Concurrent systems: an integrated approach to operating systems, distributed systems and databases* by J. Bacon, R. Laney, J. Van der Linden, Addison-Wesley 3rd edition 2003.

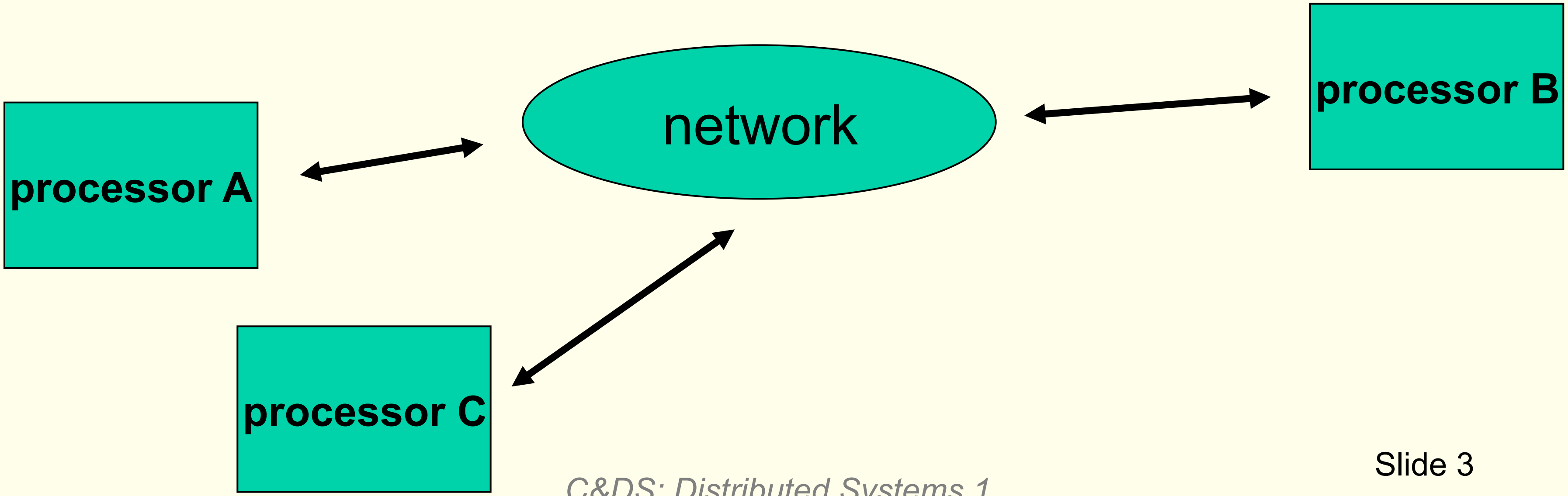
- *Operating system concepts* by A. Silberschatz by P. Galvin, G. Gagne, Wiley 8th edition 2010.

- **Emails are welcome**
  - **abb@cs.stir.ac.uk**

what is a distributed system ?

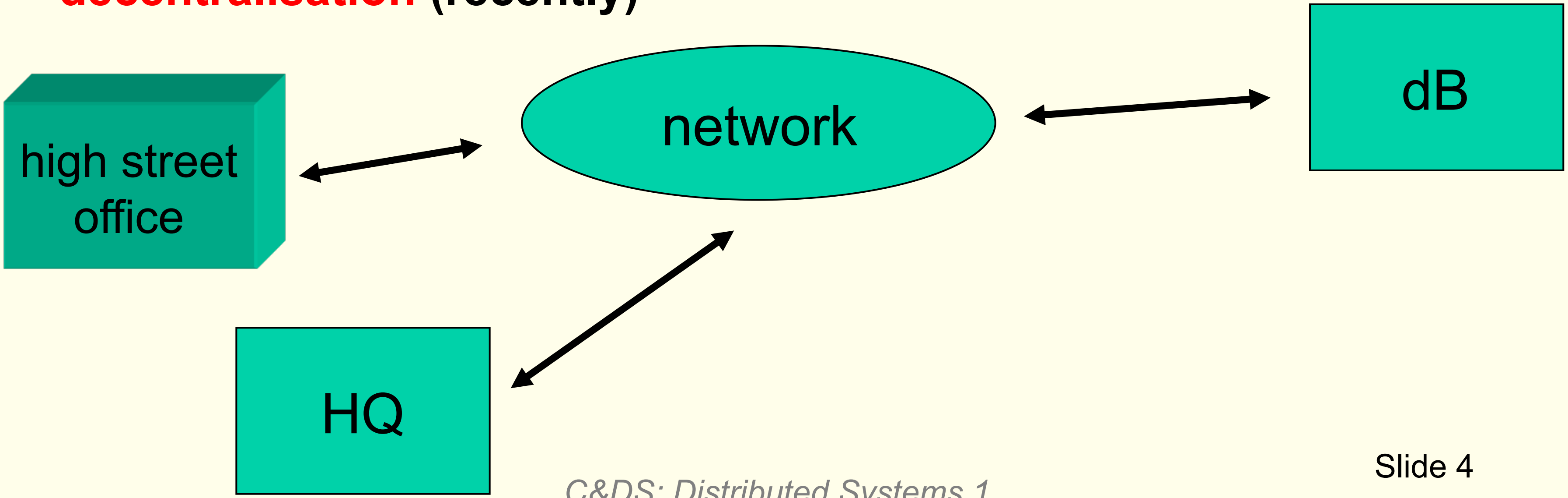


- "A *distributed system* is a collection of processors that do not share memory or a clock"
  - – Silberschatz
- loosely coupled processors
- using communications links or network to communicate
- processor + resources at a *local* site
- other processors (& their resources) are *remote*



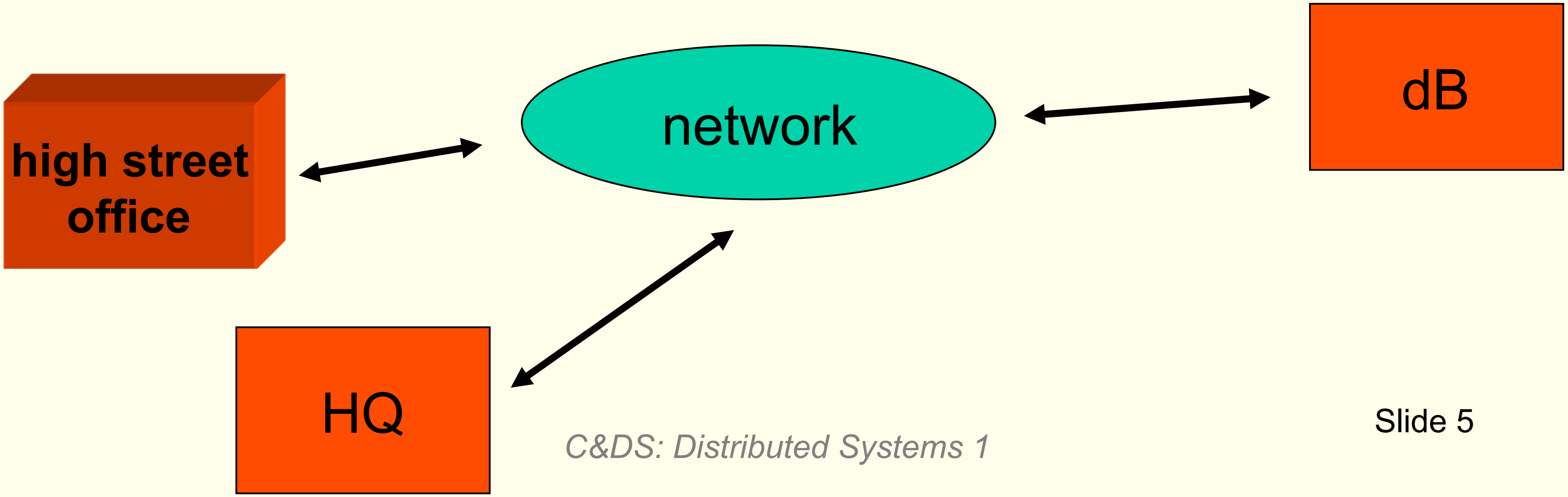
why have a distributed system ?

- share resources, e.g. data, devices
- computational speedup
- reliability
- communications, e.g. email, intra-net
- costs
- cloud computing (recently)
- decentralisation (recently)



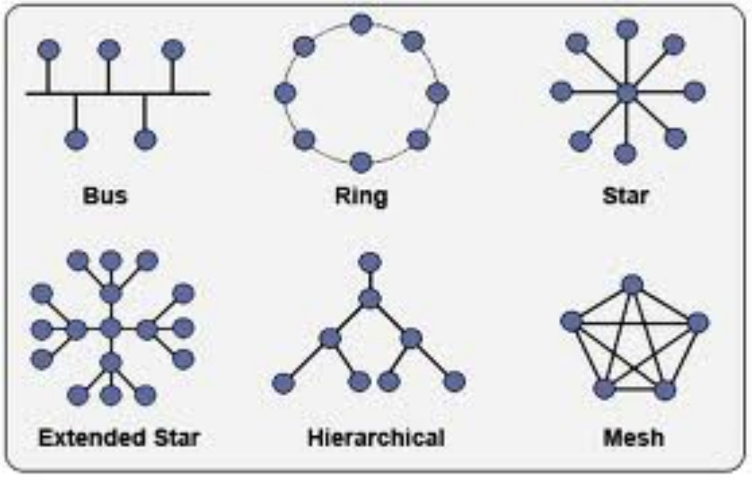
terminology

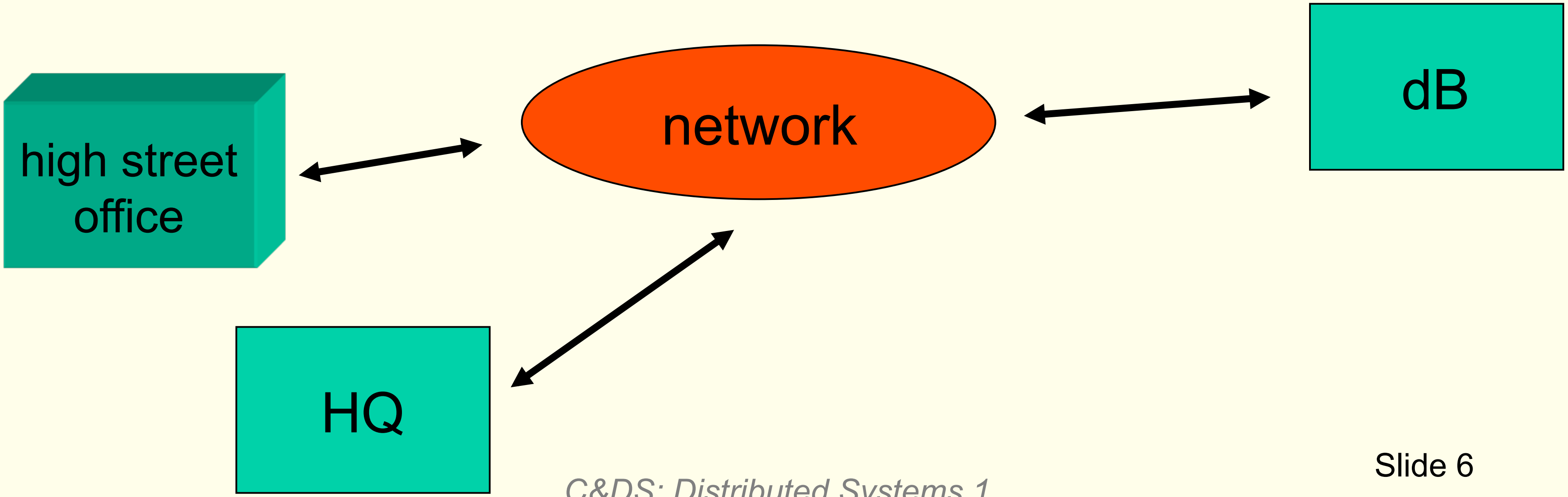
- often a site is also called**: node**, **computer**, **machine**, **host**
- more accurately, a **site** is the ***location*** of processors and other resources
  - resources at a site include **: processors, printers, disks, specialised hardware, etc.**
  - processors are also called **: node, computer, machine, host**
  - a ***host*** is a particular **computer** at a ***site***
- a **client** on a **host** may access a **server** on another **host** at another **site** to access a particular **resource**



what is the network ?

- **LAN**
- **MAN**
- **WAN**
- **dedicated links**
- **can be shared access, packet switched or circuit switched**





characteristics of a *distributed system*

- **if a distributed system is just a special kind of concurrent system, why study them?**
- **distributed systems have characteristics that must be taken into account in the design and implementation of software.**
- ***independent failure modes***
  - **the components of a distributed computation and the network connecting them may fail independently of each other.**
  - **e.g. network failures may partition the system**
  - **e.g. node failures may make certain services or data unavailable.**
- ***no global time***
  - **Each component of the system has a local clock but the clocks might not record the same time.**
  - **The clock hardware is not guaranteed to run at the same rate on each node.**
- ***no shared memory***

characteristics of a distributed system

- ***communications delay***
  - there is a certain latency associated with communications between nodes.
  - thus, it takes time for the effects of an event at one point in a distributed system to propagate throughout the system.
  - The latency is not fixed, it varies - the delay is different for each communication (jitter)

- ***inconsistent state***
  - concurrency, failures, and communication delays imply that the view of any state maintained by the distributed computation will not be consistent throughout the system.
  - that is, nodes may differ in their opinion about the current state of the system.

characteristics of a distributed system

- ***independent failure modes (revisited)   ….. From Google University ….***
- Failure is the defining difference between distributed and local programming, so you have to design distributed systems with the expectation of failure. Imagine asking people, "If the probability of something happening is one in 10^{13}, how often would it happen?" Common sense would be to answer, "Never." ….. But if you ask a physicist, she would say, "All the time. In a cubic foot of air, those things happen all the time."
- When you design distributed systems, you have to say, "Failure happens all the time." So when you design, you design for failure. It is your number one concern. What does designing for failure mean? **One classic problem is partial failure**. If I send a message to you and then a network failure occurs, there are two possible outcomes. One is that the message got to you, and then the network broke, and I just didn't get the response. The other is the message never got to you because the network broke before it arrived.
- So if I never receive a response, how do I know which of those two results happened? I cannot determine that without eventually finding you. The network has to be repaired or you have to come up, because maybe what happened was not a network failure but you died. How does this change how I design things? For one thing, it puts a multiplier on the value of simplicity. The more things I can do with you, the more things I have to think about recovering from.

characteristics of a distributed system

- *independent failure modes (revisited)* ***….. Which is best? ….***

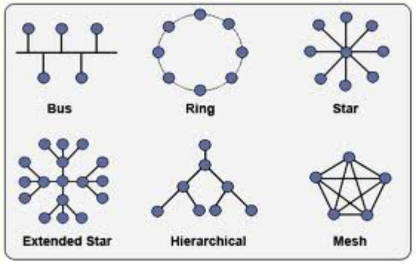
- *Invoke remote method* **`increment_my_bank_account_by(x)`**

- *Invoke remote method* **`new_total_for_my_bank_account_is(y)`**

?

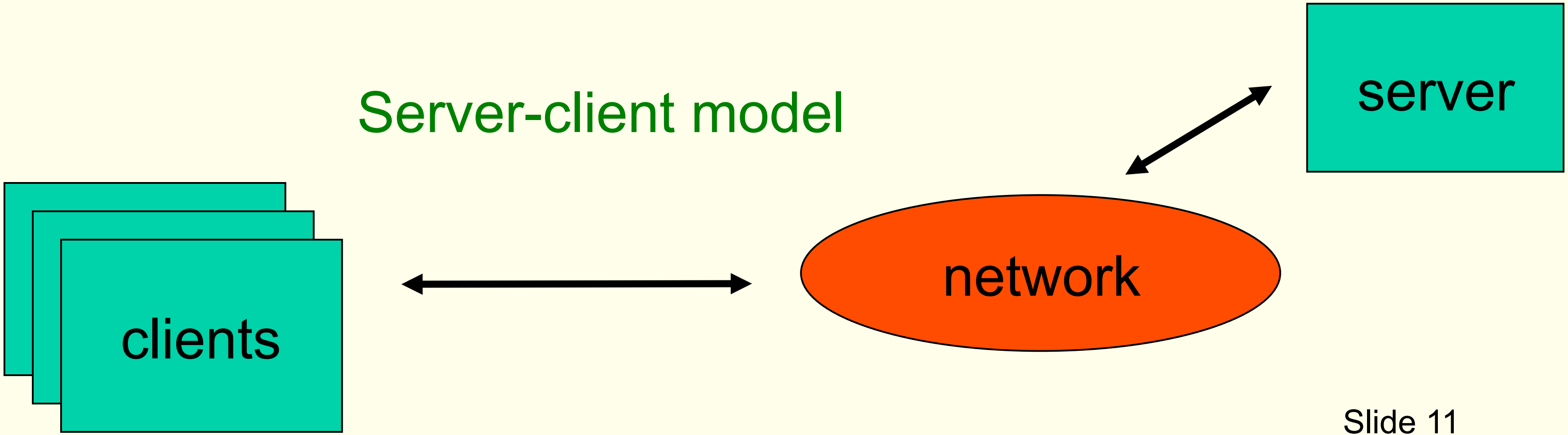
architectures

- **physical**
  - **bus**
  - **ring**
  - **mesh**
  - **point to point**
  - **again, familiar from CSC9W6**

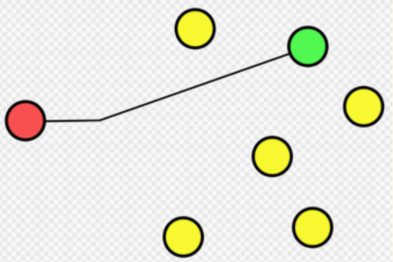


- **logical**
  - **one to one**
  - **any to one (server-client)**
  - **one to many (broadcast)**
  - **one to one-of-many**
  - **many to many**

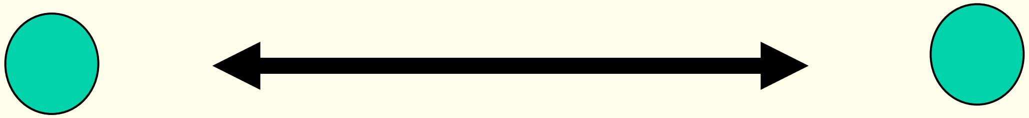
Server-client model



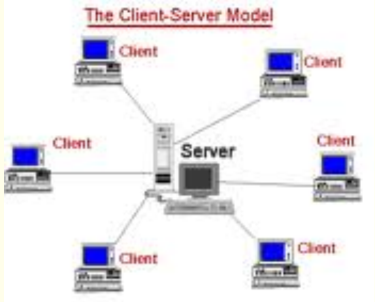
one to one



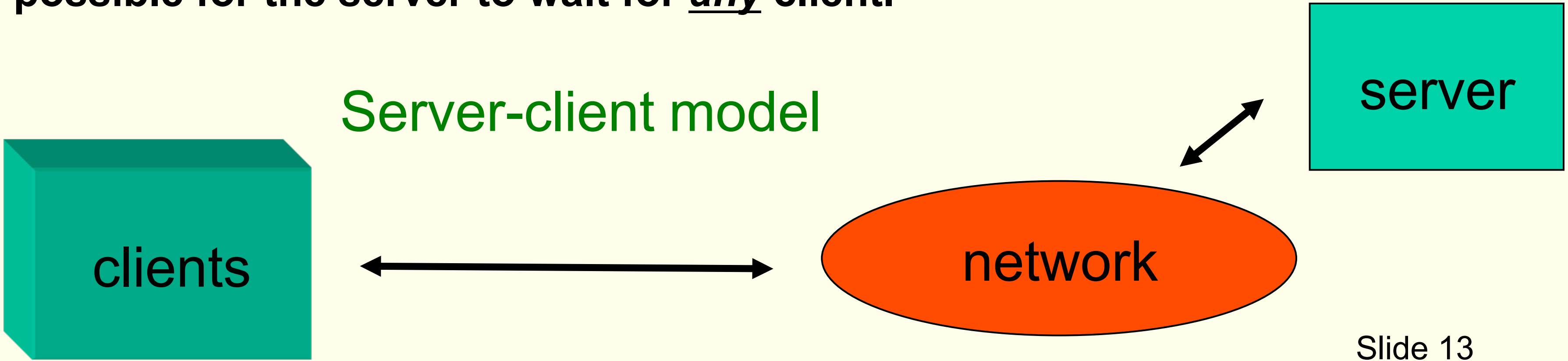
- **this arrangement is appropriate in systems that have a *static configuration* of interactions between individual processes.**
- for example, consider the following Unix command: **cat /etc/passwd | less**
- **this is known as a *pipeline* command.**
  - **the *cat* process sends the contents of the file */etc/passwd* to another process running the *less* command.**
- **in a one-to-one interaction, the names of the processes involved are often known in advance.**
- **in distributed systems this can be made dynamic using (separate) Registries – we will see this throughout in the course.**



any to one



- a common arrangement is multiple *clients* interacting with a single *server*, e.g. :
  - mail server and mail clients (Eudora etc.)
  - web browsers and web servers
- here, the server will accept a request from any potential client process.
- clients will typically invoke a “well-known” server. The name is public.
- the server cannot know which client will interact with it next, so it must be possible for the server to wait for *any* client.



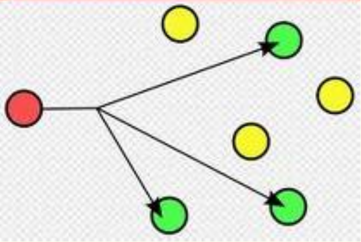
Server-client model

clients

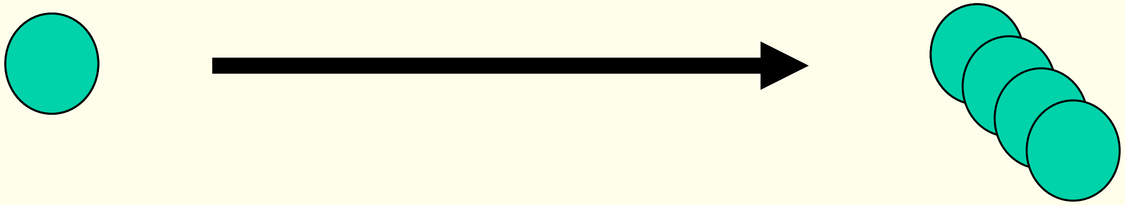
network

server

one to many (1 of 2)

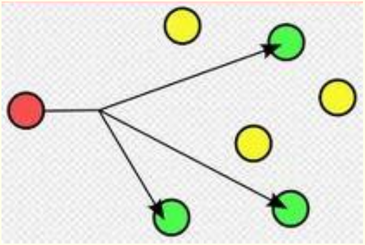


- can be used to notify a set of interested parties that an event has occurred.
- there are two forms:
  - *broadcast:* The communication is sent out to *everyone*.
  - *multicast:* The communication is sent to a *specific set* of recipients.
- with broadcast, there is usually no record of who has and who hasn’t received the communication.
  - It is simply whoever was “listening” at the time.
- multicast may be implemented as a communication service by the OS, otherwise it can be viewed as a series of one-to-one interactions.
- common in wireless systems, e.g. wireless sensor networks

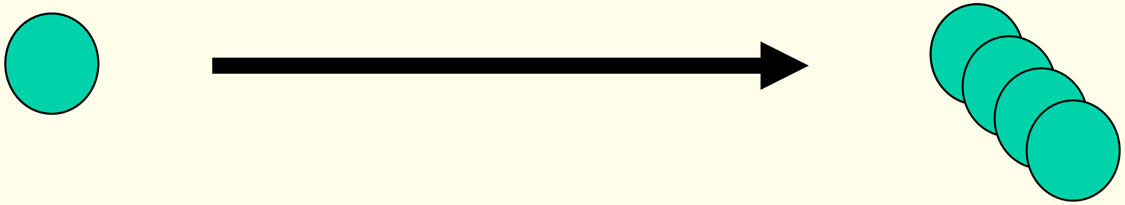


Slide 14

one to many (2 of 2)

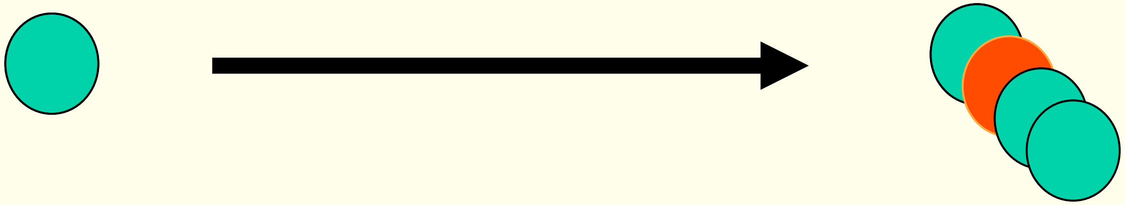


- one issue relating to multicast is how the intended recipients of a multicast are identified.
- in some systems, a name is assigned to a ***multicast group*** that processes can join.
  - A simple example is a mailing list.
- one-to-many communication can be used in fault-tolerant systems where a given task is carried out by more than one process.
- for example, three processes might compute a result and vote on its value.
  - If one process fails, the other two can still agree on the result.



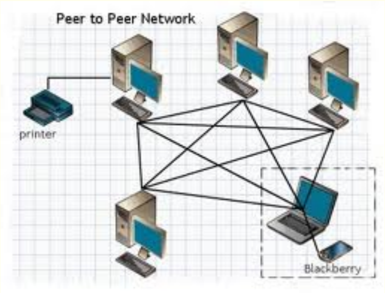
any to one-of-many

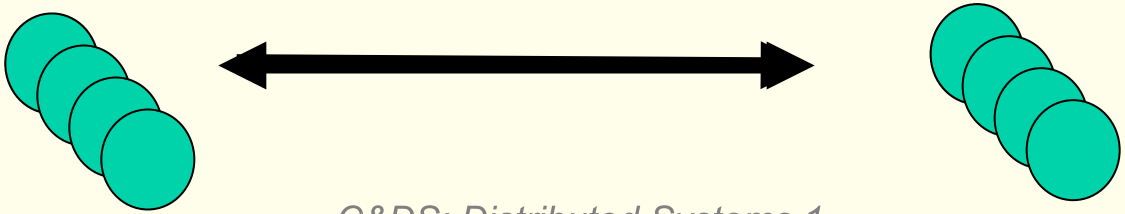
- **a service might be provided by a set of anonymous server processes.**
- **a client needs to request a service from any free server process rather than a particular named server.**
- **the servers may be implemented as a single multi-threaded process, or may be separate processes on different machines connected to the network.**
- **really only included for completeness - usually reduces to the other classifications**
  - **For example, a client might send a multicast message to the set of servers.**
  - **The free servers respond and the client chooses the first.**
- **as an example, consider a file server where any process in a pool may service a request.**



many to many

- **here for completeness, as normally an issue with shared data**
- **processes & threads may interact by reading and writing shared data.**
- **using this technique, any number of processes (threads) can interact.**
- **we know from earlier in the course - such interaction requires the use of mutual exclusion to prevent chaos from happening.**
- **we can use mechanisms such as *locks*, *semaphores*, and *condition_variables* to help us.**
- **less common in distributed systems:**
  - **not even P2P networks.**
  - **structured** Vs **unstructured**

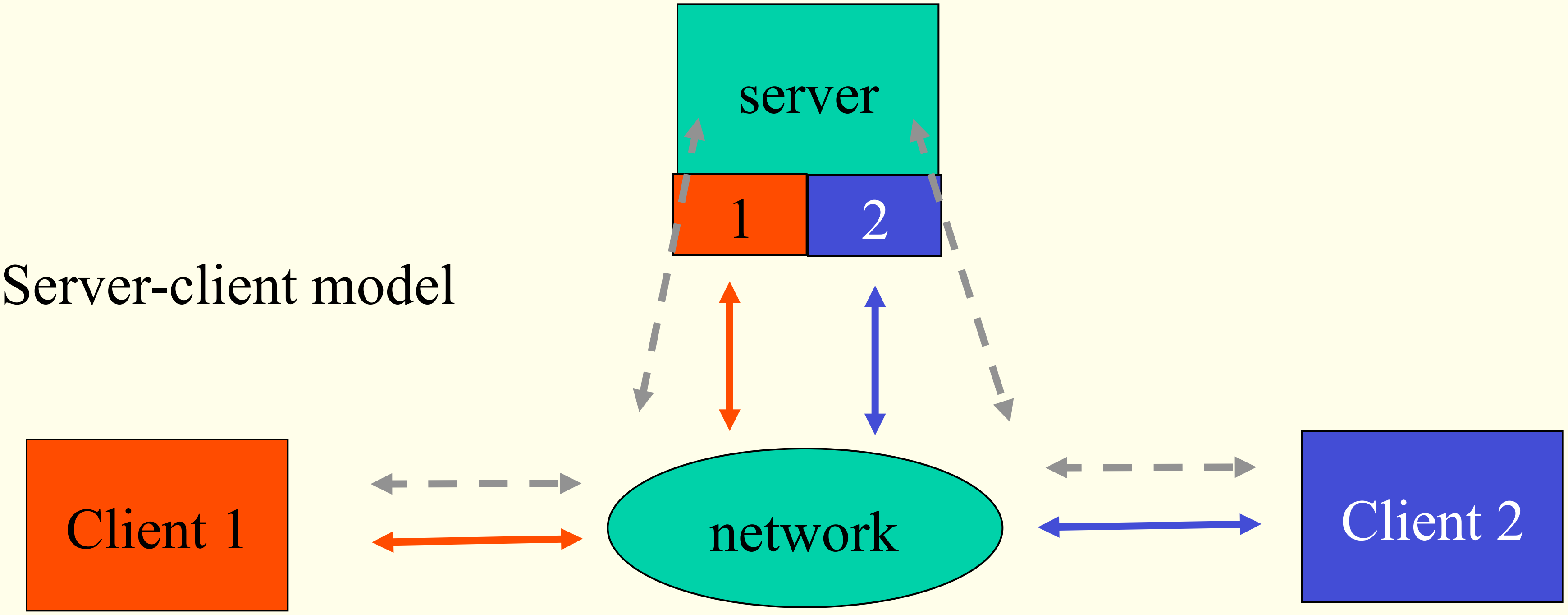




server - client

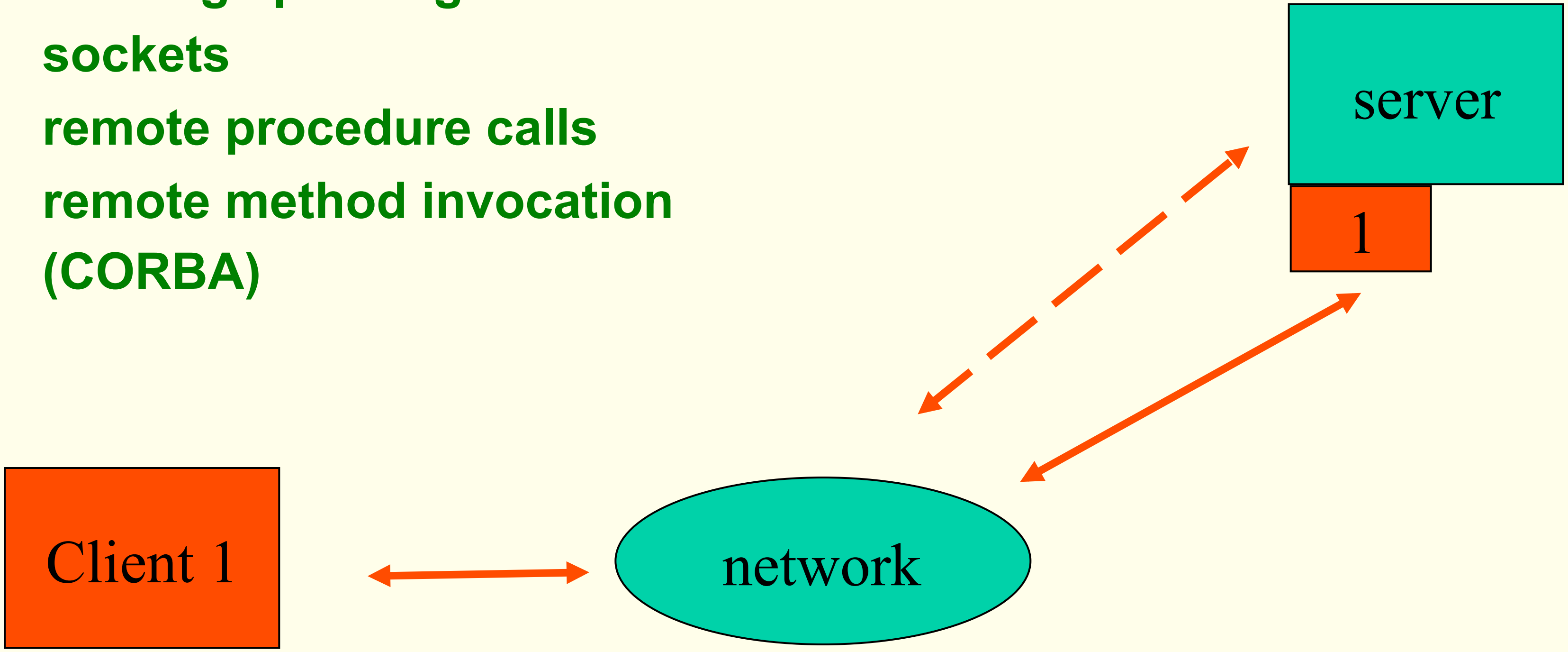
- any to one - in the large
- one to one for each client - often the server or servers have a separate process or thread to deal with a client
- however 1 shared access process or thread

Server-client model



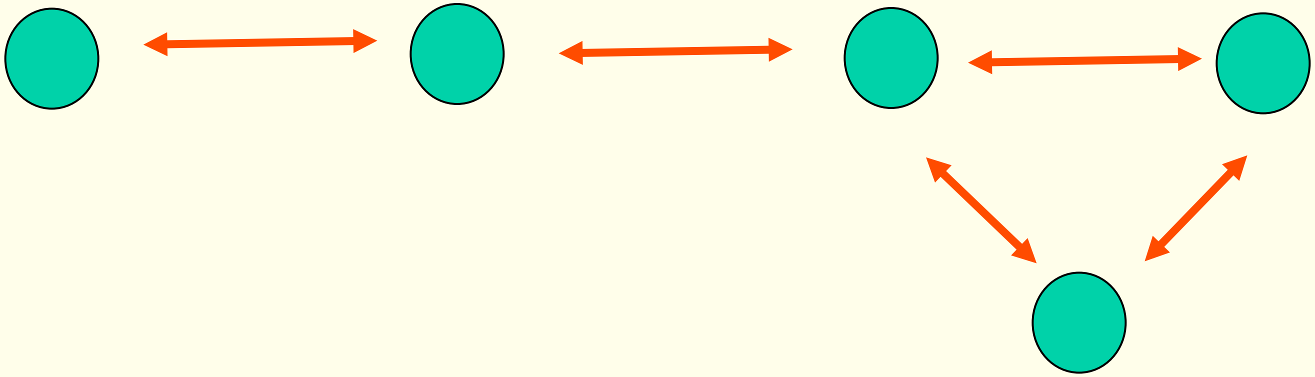
distributed communication

- much of what we do will focus on *one to one*
- no shared memory
- files
- message passing
- sockets
- remote procedure calls
- remote method invocation
- (CORBA)



distributed coordination

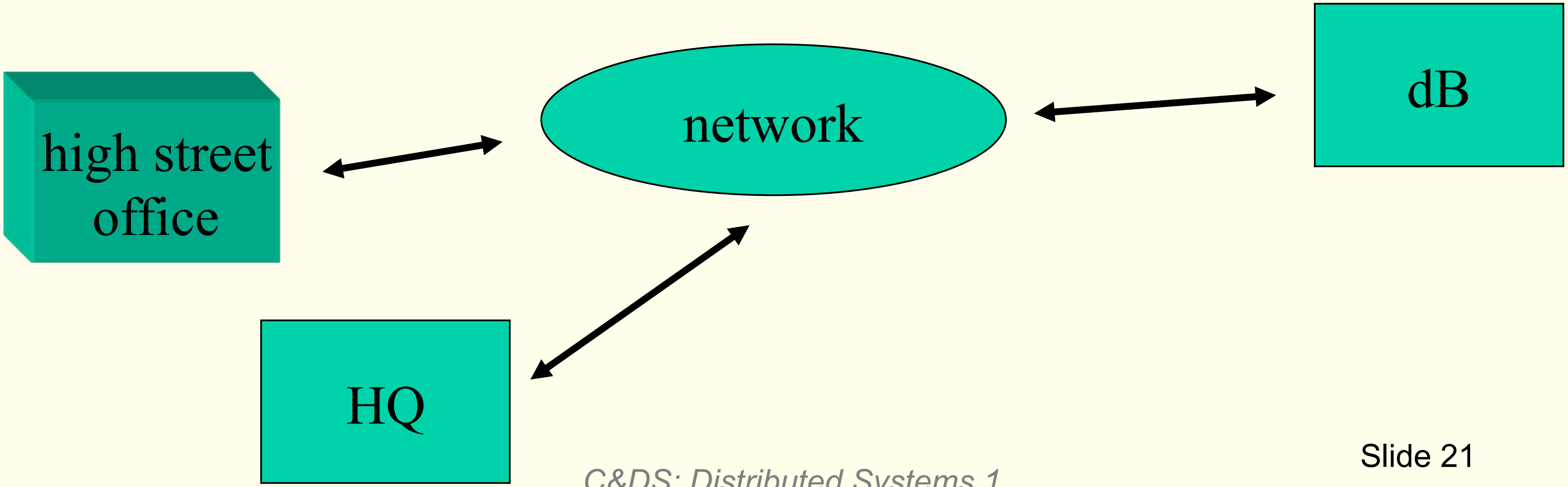
- distributed time & event ordering
- mutual exclusion
- deadlocks
- election algorithms



example systems

- Aircraft
- Company IT systems
- Communications networks
- Control of utility networks

- real-time
- security
- distributed data & control
- high reliability



example systems … cond

- **Massively multiplayer online games** - **performance**
- **Virtual worlds** - **performance**
- **Blockchains and crypto-currencies** - **decentralisation**