

Distributed Systems: Types of Architecture Model

Aidan Murphy
E-mail: aidan.murphy@ucd.ie

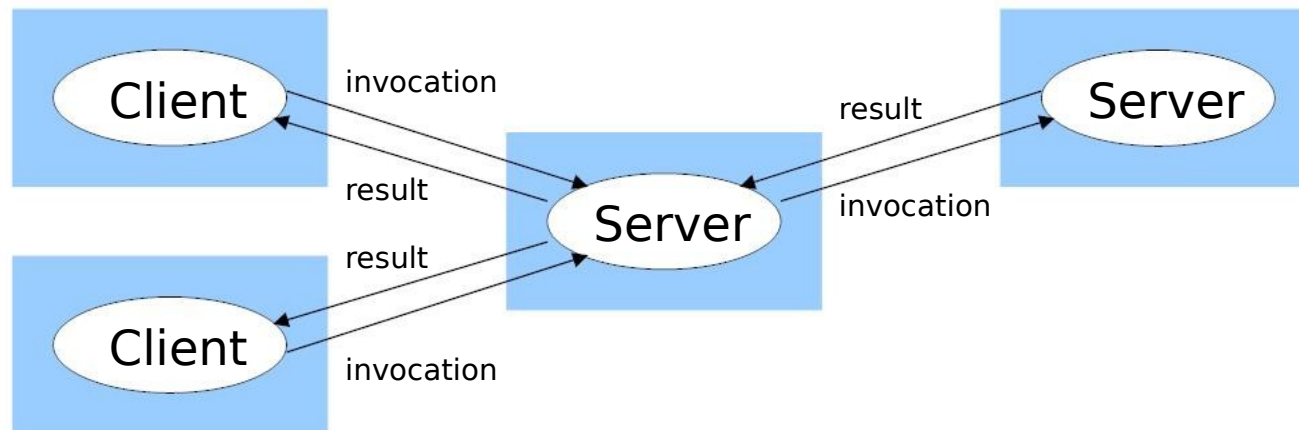
School of Computer Science
University College Dublin
Ireland



Core System Architectures

Client-Server:

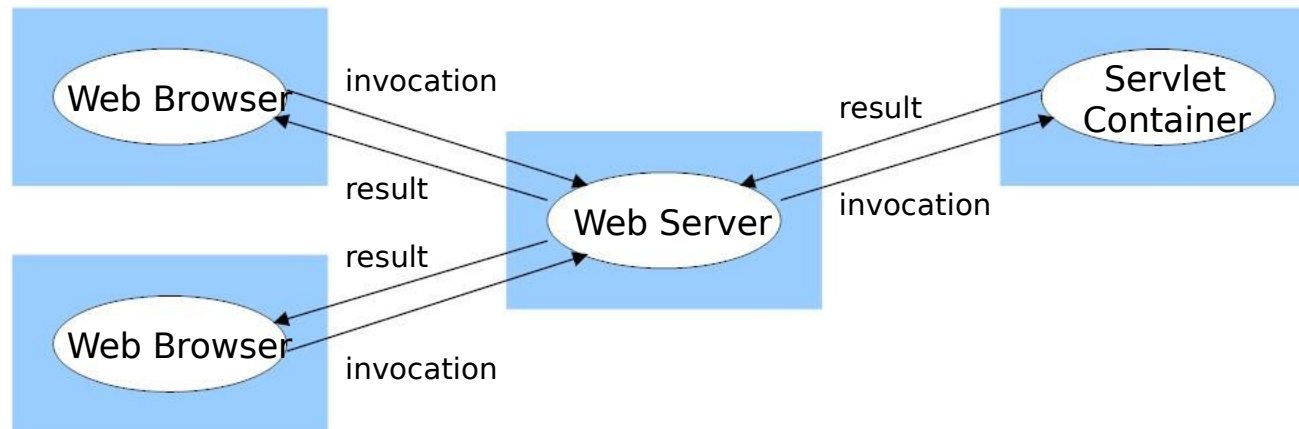
- Client processes interact with individual Server processes on separate computers in order to access the shared resources that they manage.
- Servers may themselves be clients of other servers.



Core System Architectures

Client-Server:

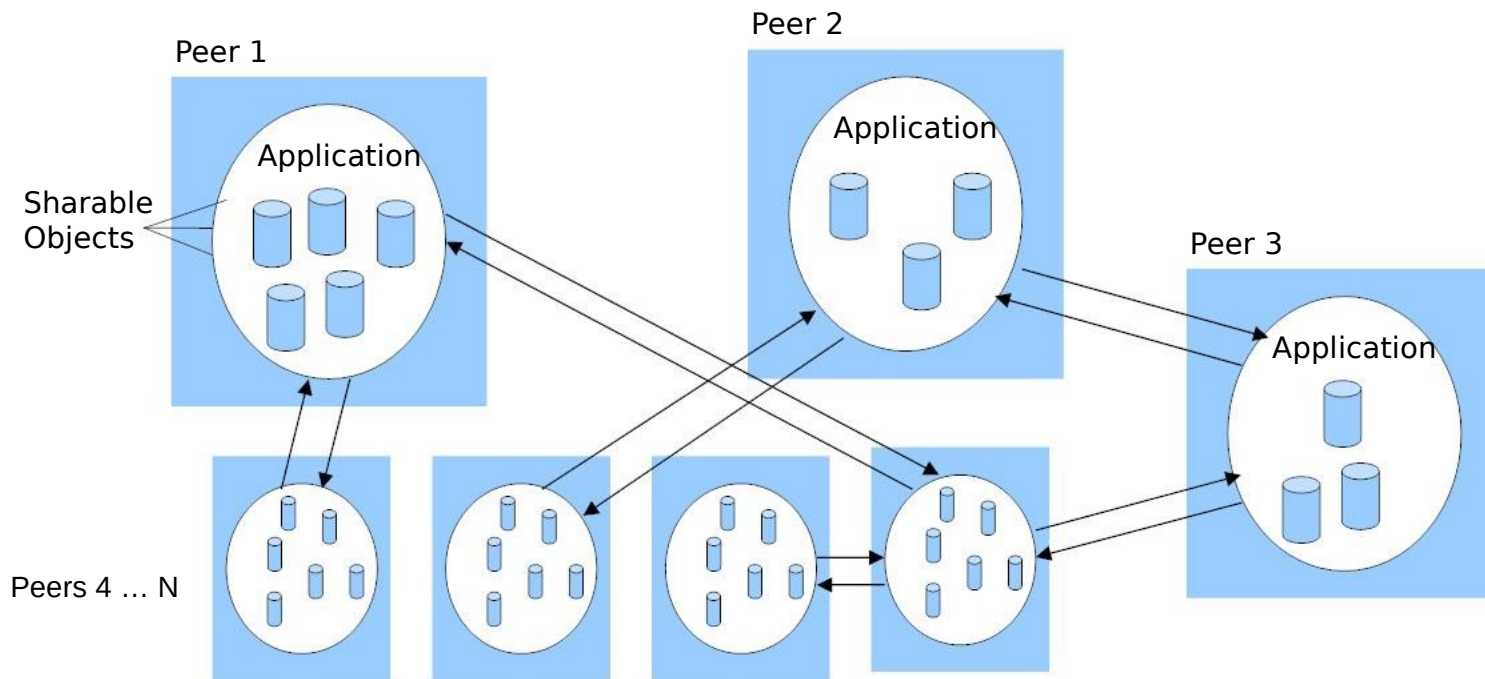
- Client processes interact with individual Server processes on separate computers in order to access the shared resources that they manage.
- Servers may themselves be clients of other servers.
- Example: The World Wide Web



Core System Architectures

Peer-to-Peer:

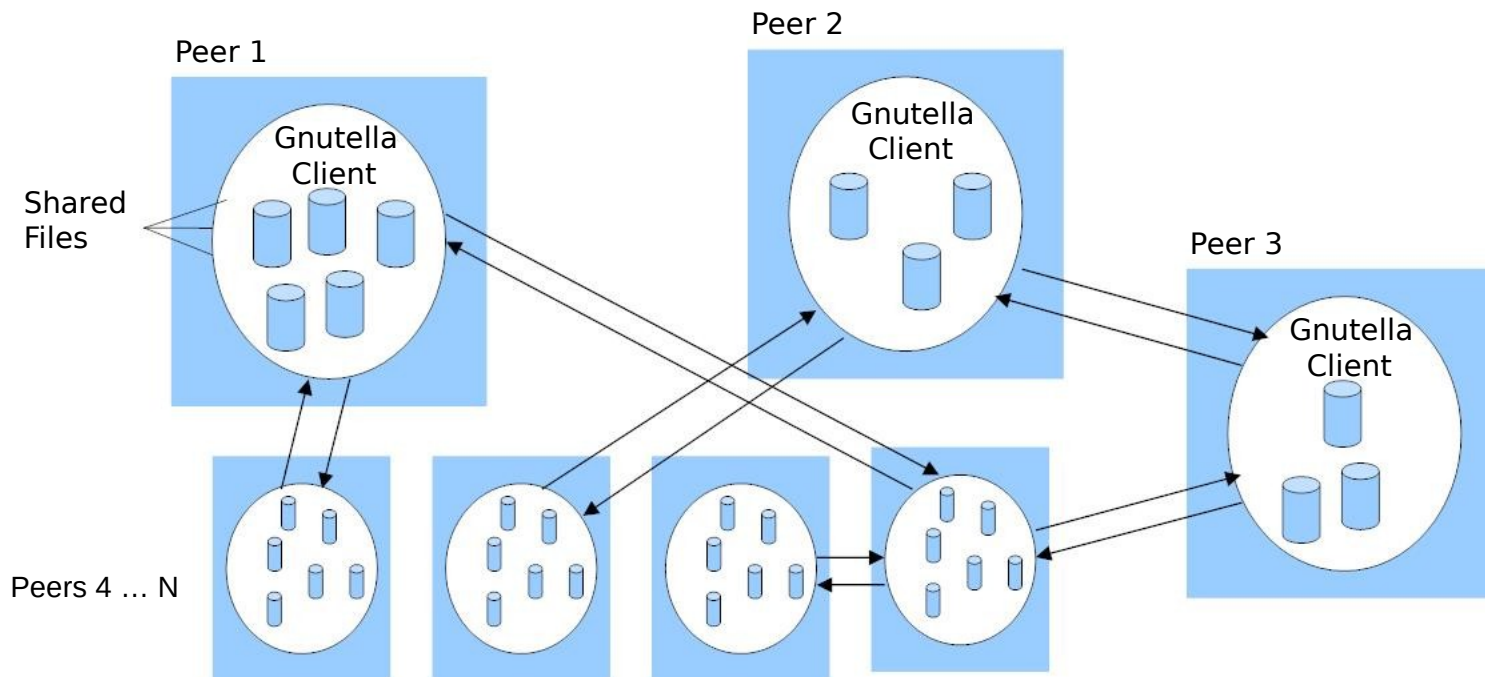
- All the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computer that the function on.



Core System Architectures

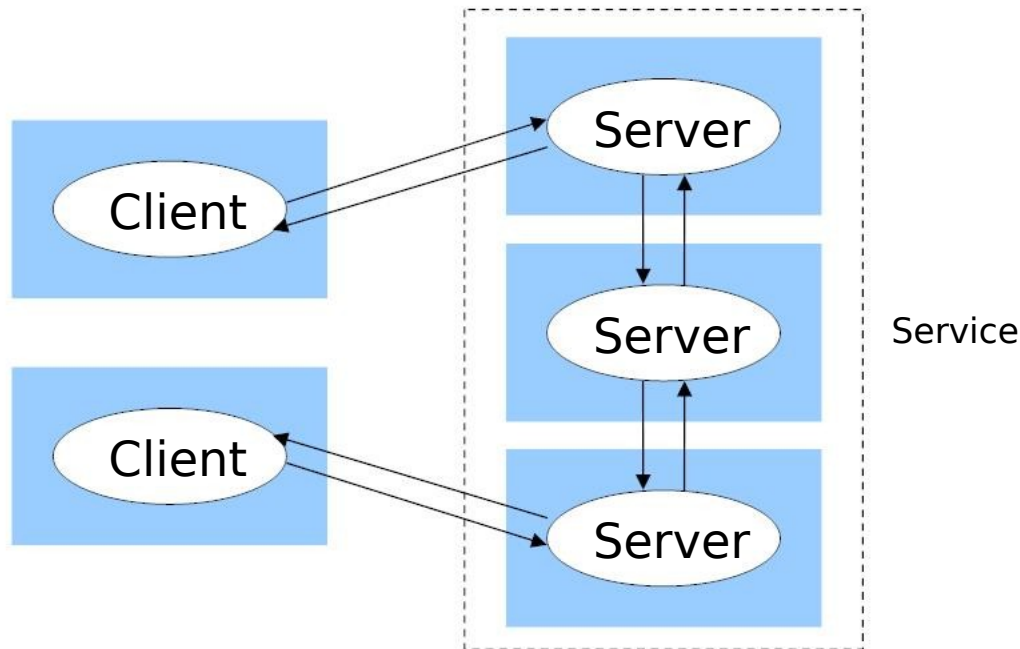
Peer-to-Peer:

- All the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computer that the function on.
- Example: Gnutella File Sharing Service.



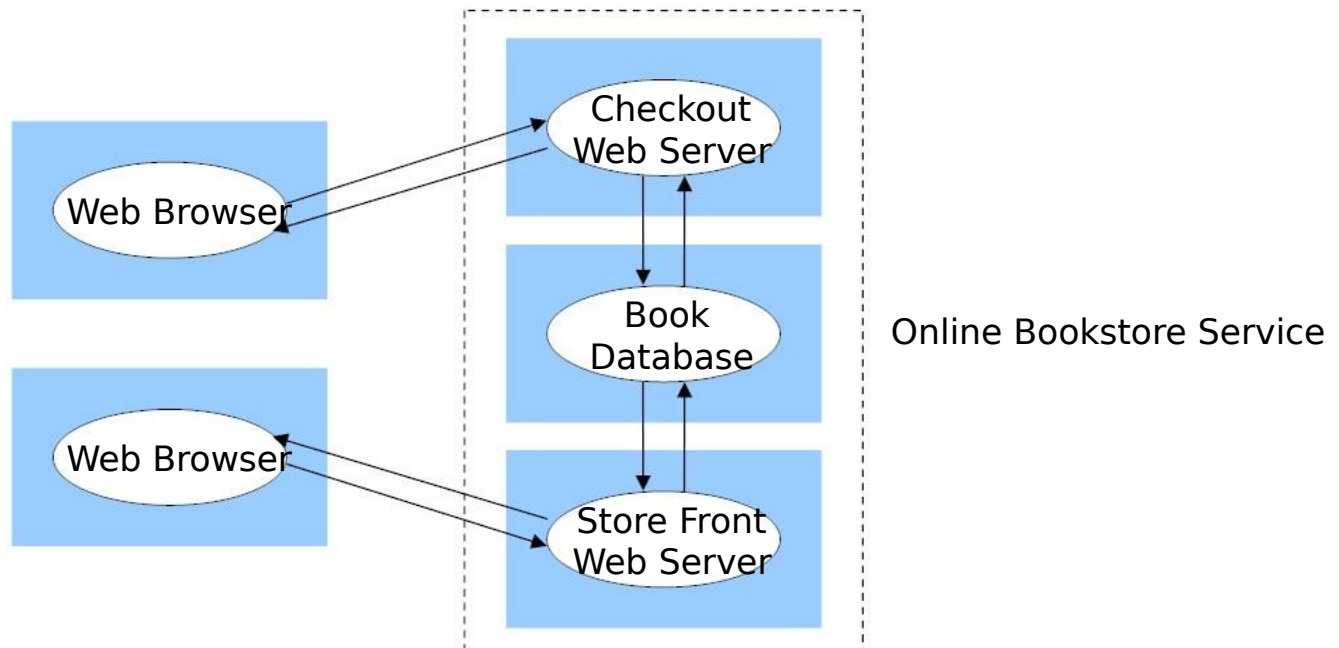
Variant System Architectures

- Services provided by **multiple servers**:
 - Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes.
 - The servers may partition the set of objects on which the service is based and distribute them between themselves, or they may maintain replicated copies of them of several hosts.



Variant System Architectures

- Services provided by multiple servers:
 - Example: **An online bookstore**
 - The store website is partitioned into two parts:
 - Store Front: the part of the website through which users browse the inventory of books for sale.
 - Checkout: the part of the website through which users purchase books



Variant System Architectures

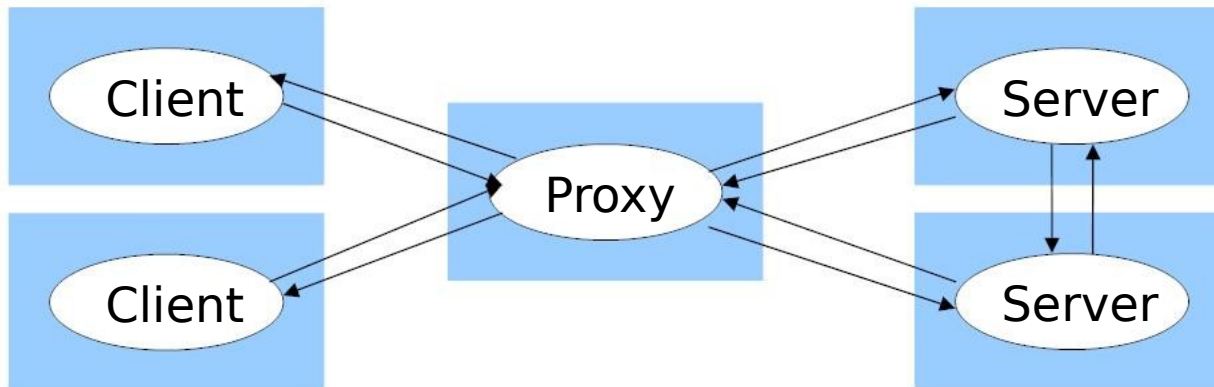
- Proxy Servers and Caches:

- A cache is a store of recently used data objects that is closer than the objects themselves.
- When a new object is received by a computer, it is added to its cache, replacing some existing objects if necessary.
- When an object is needed by a client process, the caching service first checks the cache:
 - if an up-to-date copy is available, then the caching service returns that copy
 - If not, an up-to-date copy is fetched, stored in the cache, and then returned to the client process.
- Caches may be placed **locally** to clients or in a **shared proxy server**.

Variant System Architectures

- **Proxy Servers and Caches:**

- A proxy is a server process that acts as an intermediary between a set of clients and a set of servers.
- The clients send a message to the proxy passing a second message that should be forwarded on a server.
- The proxy keeps track of what messages were sent by which client, acts as a client and sends the message to the relevant server, and finally, upon receipt of a response from the server, forwards that response on to the client.



Variant System Architectures

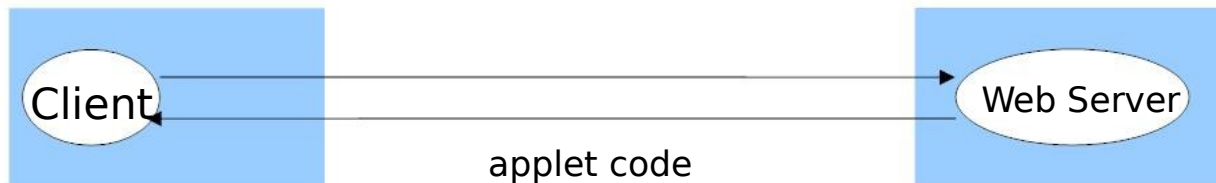
• Mobile Code:

- This is code that can be sent from one computer to another and run at the destination.
 - This is not easy to do because many languages are compiled to machine code, which is platform specific.
 - One option would be to download the source code, but this is limited because: (1) there is no guarantee that any libraries used would exist on the client machine, and (2) the client machine would need a compiler to be stored and would need to know how to invoke that compiler.
- Some languages, such as Java, can be used as they employ the concept of a virtual machine to make themselves platform independent.
- The advantage of mobile code is that it is executed locally resulting in good interactive response times.
- It is useful in cases where we want to augment a clients basic capabilities (e.g. a 3D viewer for a web browser).
- The main issue with mobile code is security – how can we control what the code does on the local machine in order to ensure that no malicious activity occurs.

Variant System Architectures

- Mobile Code:

- Example: Java Applets
- Java code that can be downloaded from a web server and executed via the web browser.
- Strict security policies are enforced which require authority validated certificates in order gain access to many aspects of the client machine (known as a Signed Applet).



- Step 1: Client Request Results in the Downloading of Applet Code



- Step 2: Client Interacts with the Applet

Variant System Architectures

• Network Computers

- An alternative to the traditional desktop computers that employ local installations of both the Operating System (OS) and most required applications.
- Network Computers (NC) contain minimal (if any) permanent data storage (only processor + memory) and download both the OS and any required applications from a remote server.
- Where permanent data storage is provided, only a minimal amount of software is maintained, with most of the space used as a cache to improve download speeds.
- Applications are run locally, but any user files are managed by remote file server.
- The goals behind NC are:
 - to reduce the cost of a computer,
 - to minimise the technical knowledge required to maintain it, and
 - to support migratory users.

The main drawback of NC is the reliance on the network as no network means no productivity!!!

Variant System Architectures

Thin Clients

- A software layer that supports a window-based user interface on a computer that is local to the user while executing application programs on a remote computer.
- Has similarly low management and hardware costs as the network computer architecture, but instead of downloading the code, it runs them on a remote compute server (typically a multiprocessor machine or cluster).
- The main drawback of Thin Clients is its reliance on the network:
 - Network failure during execution of the program can result in a loss of data
 - Updating of the interface can be slow due to network latency
 - Performance degrades rapidly as graphical usage increases.



Variant System Architectures

• Thin Clients

- Example: X-11 Window System
- This is a service, developed for UNIX, which manages the display and interactive input devices (keyboard, mouse) of the computer on which it runs.
- It consists of an extensive library of procedures, known as the X-11 protocol, for displaying and modifying graphical objects in windows as well as the creation and manipulation of the windows themselves.
- The X-11 system is known as the window server process, and the clients are the application programs that the user is currently interacting with.
- The client programs interact with the server by invoking operations in the X-11 protocol using a Remote Procedure Call (RPC) mechanism.



Distributed Systems:

Design Requirements for Distributed Architectures

Performance Issues

• Responsiveness

- Users of interactive applications require a **fast and consistent response** to interaction.
 - The key factor is the response time
- Because client systems often need to **access shared resources**, response times for remote services can be affected by:
 - Load and performance of the server
 - Latency and performance of the Network
 - Time costs associated with the use of middleware services and OS communication services
 - The processing time for the actual code that implements the service
- **Maximum responsiveness** is often achieved by:
 - Minimising the number of software layers used by the application
 - Minimising the quantities of data that is transferred between the client and the server
- Example: Web Browsing
 - Locally cached web pages and images load more quickly than similarly sized remote ones and simple text-based pages load more quickly than graphically intensive pages.

Performance Issues

• Throughput

- Throughput is the rate at which computational work is done.
- It is similar to responsiveness, with the exception that it focuses on task completion times rather than interface update times.
- However, in contrast with responsiveness, it is affected by the processing speeds of both the client and the server.
- When analysing throughput, it is also vital to consider the throughput of each of the software layers that the architecture employs.

• Balancing Computational Loads

- Distributed systems aim to allow tasks to be performed concurrently without competing for the same resources and at the same time exploiting all available computational resources (processor, memory, and network capacities).
- For example, Javascript programs that run inside browsers can reduce the load on the web server, enabling it to provide a better service.
- Alternatively, the website can be replicated on multiple web servers and a load balancing technique, such as multiple addresses for a single domain name, can be used to spread the load over the replica servers.

Quality of Service

- **Quality of Service** is measured in terms of reliability, security, and performance.
- More recently, Quality of Service measures have begun to include adaptability:
 - The ability of the system to meet changing system configurations and resource availability.
- In the previous slides performance was measured in terms of responsiveness and throughput.
 - Another factor that is sometimes associated with performance is **time-criticality**:
 - The handling of data that must be processed or transferred from one process to another at a fixed rate.
 - For example, a video streaming service must be designed to ensure that the successive frames of video must be delivered in sufficient time for the client to be able to display the video to the service users.

Use of Caching and Replication

- These represent **two key techniques** for overcoming the Performance and Quality of Service issues outlined on the previous slides.
- **Caching** can significantly reduce client response times.
 - Maintaining a local copy of resources reduces the need to access remote servers.
- **Replication** can reduce network costs and balance computational loads.
 - Making copies of resources and distributed those resources throughout the distributed system allows clients to access copies of resources that have least network costs.
 - Also, making copies of a resource reduces the competition for that resource, and careful placing of that resource and increase the computational resources available to complete a given task.

Additional Reading

📖 Distributed Systems, Concepts and Design
(4th ed.),

📖 By Coulouris, Dollimore and Kindberg

📖 Read Section 2.3 “Architectural Models” and Section 2.4
“Fundamental Models”, pp 40-76

Distributed Systems:

Distributed Concurrency

Introduction

🍂 **Centralized systems** have a single common memory and clock:

🍃 We can always determine when an event occurred.

🍂 **Distributed systems** often have no common memory and no common clock:

🍃 Determining when an event occurred can be a little more difficult.

🍂 Unfortunately, common clocks are needed by distributed systems:

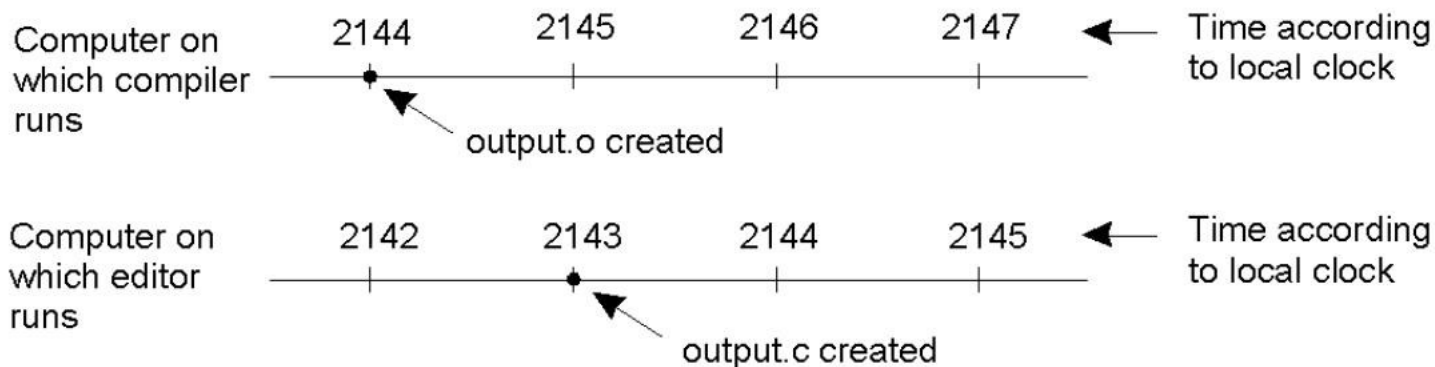
🍃 Without a common clock we cannot identify a global ordering of the events that occur in the components of our system.

🍃 Nor can we synchronize the activities of the components in the system.

Examples

🍀 Example 1: Two sharpshooters in a multiplayer online game kill the same target. Which one gets the points?

🍀 Example 2: The make command:



🍀 For distributed systems, we must create the impression of having a global common clock.

🍀 Can we set all clocks in a distributed system to have the same time?

Distributed Systems: Physical Clocks

Physical Clocks

🍃 “Exact” time was computed by astronomers

🍃 The difference between two transits of the sun is termed a **solar day**.

🍃 Dividing a solar day by $24 \times 60 \times 60$ gives you a **solar second**.

🍃 However, the Earth is slowing:

🍃 The length of a year has reduced by 35 days in 300 million years!

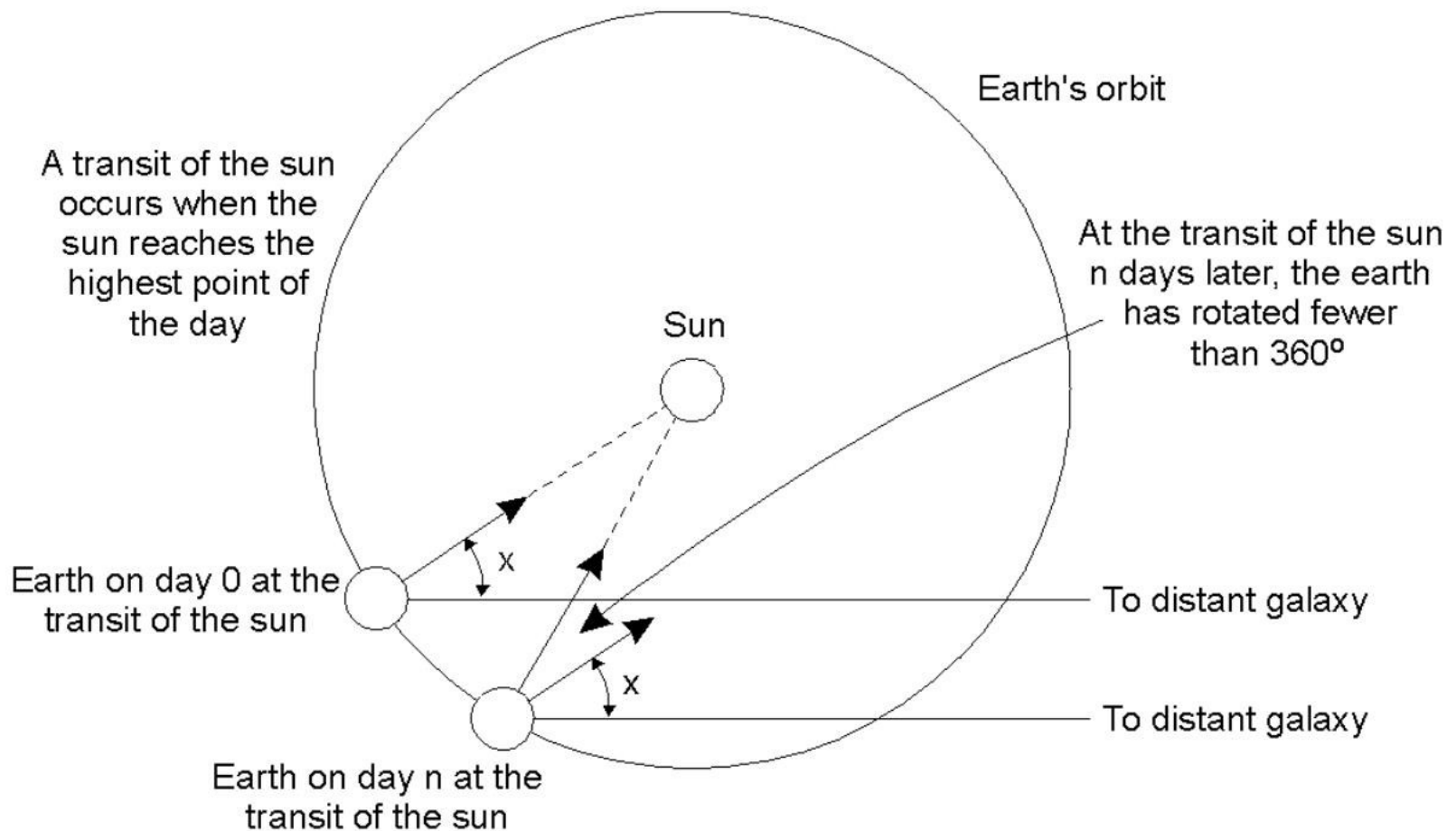
🍃 Also, there are short term fluctuations caused by turbulence deep in the Earth's core

🍃 We can improve on our initial results by calculating the length of a *mean solar day*.

🍃 From this we can calculate a *mean solar second*.

Physical Clocks

☛ Computation of the mean solar day.



Physical Clocks

👉 Physicists take over (Jan 1, 1958) and count transitions of cesium 133 atom:

👉 9,192,631,770 cesium transitions == 1 solar second

👉 Accuracy:

👉 within 2 nanoseconds per day

👉 or one second in 1,400,000 years

👉 50 International labs have cesium 133 clocks.

👉 **International Atomic Time (TAI)**: mean of the reported clock ticks from these 50 cesium 133 clocks.

👉 TAI is mean number of ticks of cesium 133 clocks since midnight on January 1, 1958 divided by 9,192,631,770 .

👉 **Universal Coordinated Time (UTC)**: It is based on International Atomic Time (TAI) with leap seconds added at irregular intervals to compensate for the slowing of the Earth's rotation.

Physical Clocks

🍃 There are two ways to receive UTC:

🍃 By Radio:

🍃 UTC is broadcast by National Institute of Standards and Technology (NIST) from Fort Collins, Colorado over shortwave radio station WWV.

🍃 WWV broadcasts a short pulses at the start of each UTC second.

🍃 This has an accuracy of 10 milliseconds.

🍃 By Satellite:

🍃 GEOS (Geostationary Environment Operational Satellite) also offer UTC service.

🍃 This has an accuracy of 0.5 milliseconds.

🍃 For more information on UTC, see:

🍃 <http://en.wikipedia.org/wiki/Utc>

Physical Clocks

👉 Computers maintain an internal clock, known as a **Real-Time Clock (RTC)**.

👉 This clock is powered by a **separate battery** so that it continues to function even when the computer is turned off.

👉 On start up, Operating Systems read the current time from the RTC uses this to initialize a virtual clock that is used by all processes.

👉 Unfortunately, it is impossible to guarantee that the real-time clocks of two computers will run at the same speed.

👉 These clocks use piezoelectric crystals which vibrate causing an electric signal to be generated at a very precise frequency.

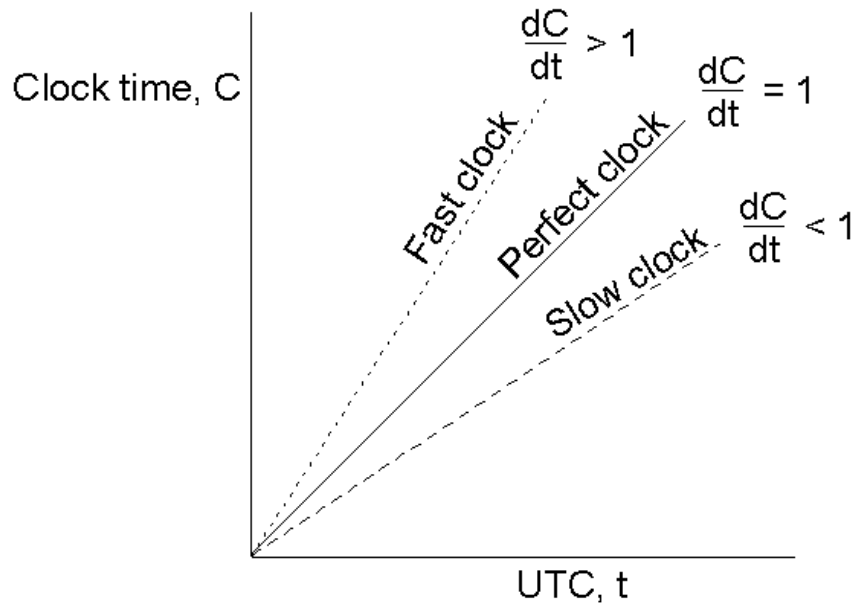
👉 Unfortunately, this frequency can vary, leading to different clock speeds.

👉 The difference in time values is called **clock skew**.

Underlying model for synchronization models

- Each machine has a timer that interrupts **H times a second**
 - Interrupt handler keeps track of the number of ticks since some agreed-upon time in the past
 - Call the value of the clock C
- Notationally, when UTC time is t , the value of the clock on machine p is $C_p(t)$
- In a perfect world, $C_p(t) = t$ for all p and all t

Towards Global Time



- This clock is C .
- Using UTC time, the value of clock C at UTC time t on machine p is $C_p(t)$.
- **For a perfect time**, $C_p(t) = t$ and $dC/dt = 1$.
- For an ideal timer, $H = 60$, should generate 216,000 ticks per hour.

Towards Global Time

- But typical errors mean that the range of ticks per second will vary from 215,998 to 216,002.
- Manufacturer specs can give you the maximum drift rate (✂).
- Every ϵt seconds, the worst case drift between two clocks will be at most $2\epsilon t$.
- To guarantee two clocks never differ by more than ϵ , the clocks must re-synchronize every $\epsilon/2\epsilon$ seconds using one of the various *clock synchronization algorithms*.

Clock Synchronization Algorithms

☛ Centralized Algorithms:

☛ A single component of the system is responsible for delivering a common global system time.

☛ Examples:

☛ Cristian's Algorithm (1989)

☛ Berkeley Algorithm (1989)

☛ Decentralized Algorithms:

☛ Multiple components of the system are responsible for delivering a common global system time.

☛ Examples:

☛ Averaging Algorithms (e.g. NTP)

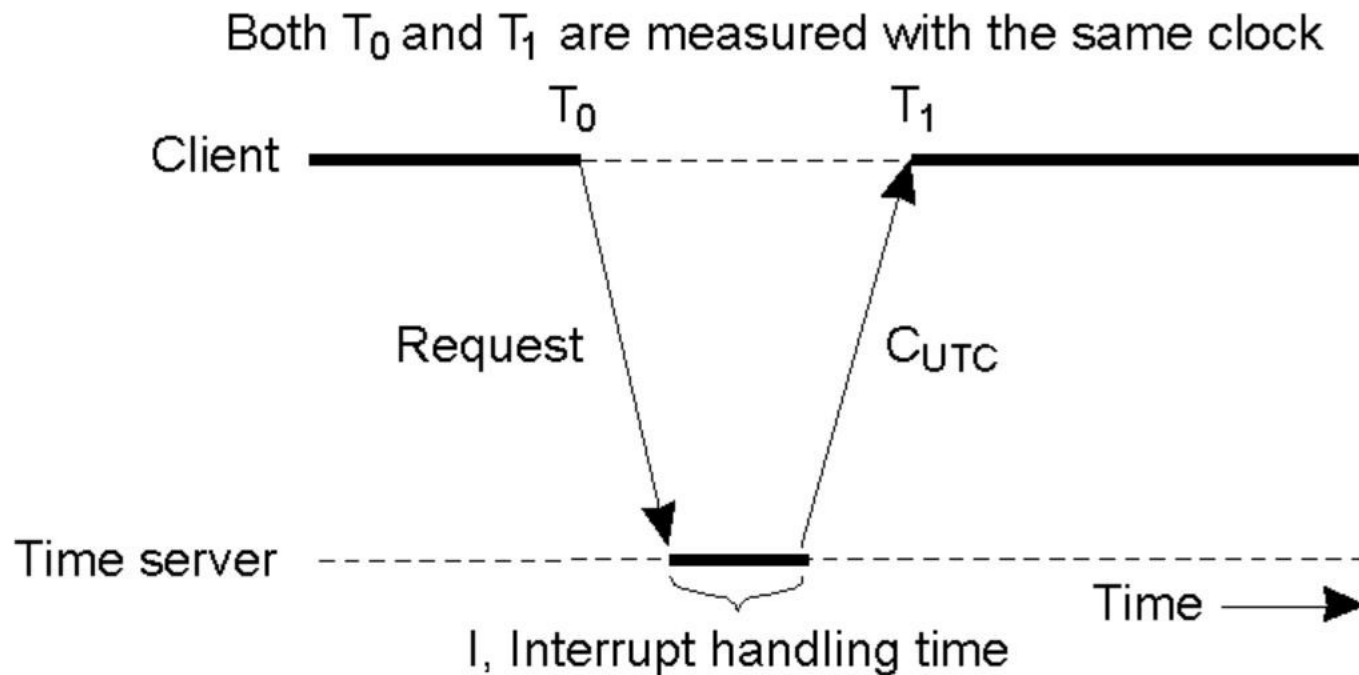
☛ Multiple External Time Sources

Cristian's Algorithm

- Assume one machine (the time server) has a WWV (or GOES) receiver and all other machines are to stay synchronized with it.
- Every $\Delta/2\alpha$ seconds, each machine sends a message to the time server asking for the current time.
- Time server responds with message containing current time, C_{UTC} .

Cristian's Algorithm


🍀 Getting the current time from a time server



Cristian's Algorithm

- Minor problem – the one-way delay from the server to client is “significant” and may vary considerably.
- What to do?
 - Measure this delay and add it to C_{UTC} .
 - The best estimate of delay is $(T1 - T0)/2$.
- In cases when $T1 - T0$ is above a threshold, then ignore the measurement.
- Can subtract off “I” (the server interrupt handling time).
- *Can use average delay measurement or relative latency (shortest recorded delay).*
- Caveat
 - Round Trip Time (RTT) estimated by the client using its own clock, thus affected by client’s clock drift
 - But this effect is small if clock drift small relative to real RTT value

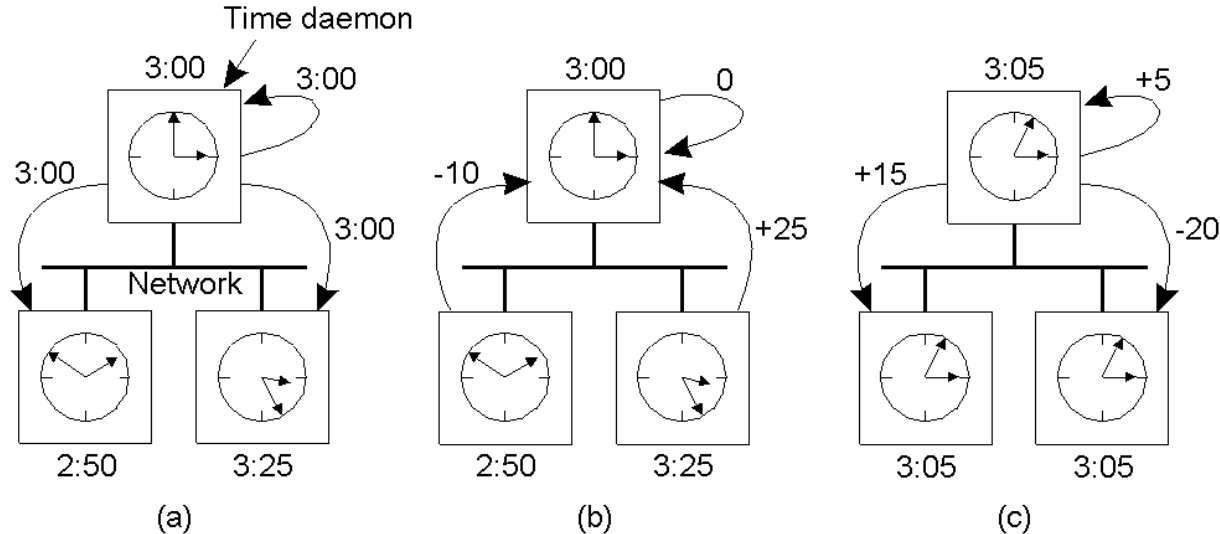
Cristian's Algorithm

- The major problem – the client clock is fast  arriving value of C_{UTC} will be smaller than client's current time, C .
- We cannot make time go backwards!!!
- What to do?
 - One needs to gradually slow down client clock by adding less time per tick.

The Berkeley Algorithm

- For internal synchronization (i.e., keeping the clocks synchronized with one another) and intended to be used on a LAN.
 - One computer is master, others are slaves
 - Requires a single master/coordinator
 - The master periodically polls all slaves for their time values using Cristian's Algorithm.
 - The master computes the average time and tells each slave how to adjust its clock
 - If the master fails a new master needs to be elected
- 🍃 The average clock of the clients tends to be more accurate than that of only a single client.

The Berkeley Algorithm



- Steps:

(a) The time daemon asks (by **polling**) all the other machines for their clock values

(b) The machines answer

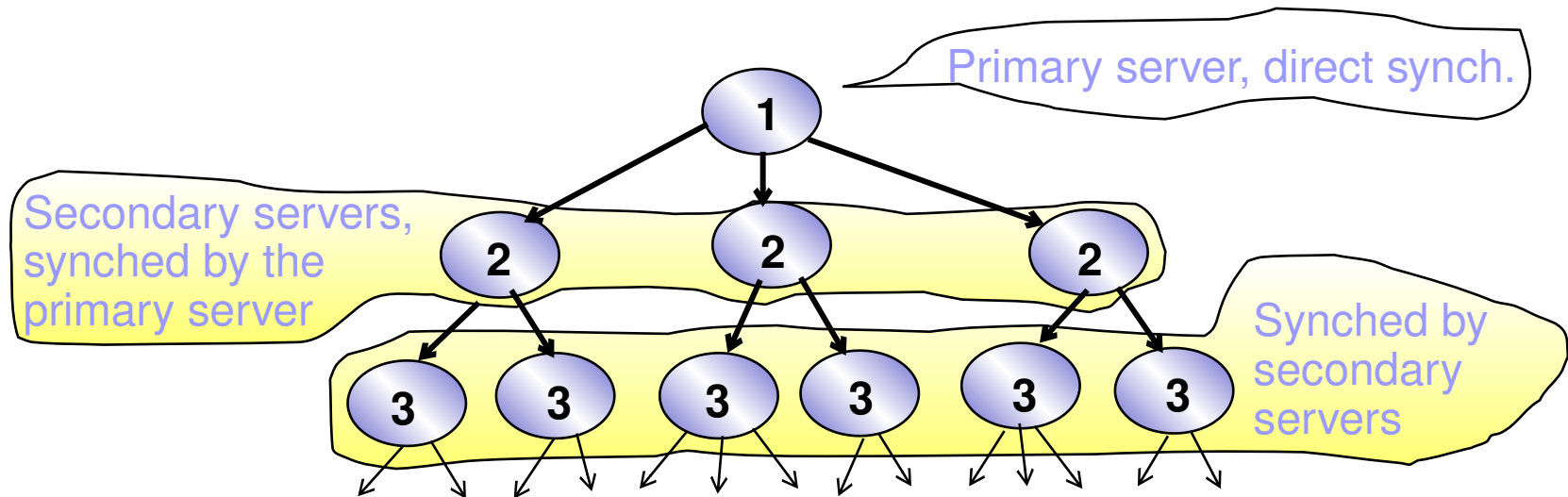
(c) The time daemon tells everyone how to adjust their clock

Averaging Algorithms

- 🍃 Every R seconds, each machine broadcasts its current time.
- 🍃 The local machine collects all other broadcast time samples during some time interval, S .
- 🍃 The simple algorithm: the new local time is set as the **average of the value** received from all other machines.
- 🍃 A slightly more sophisticated algorithm: Discard the m highest and m lowest to reduce the effect of a set of faulty clocks.
- 🍃 One of the most widely used algorithms in the Internet is the **Network Time Protocol (NTP)**.
 - 🌊 Achieves worldwide accuracy in the range of 1-50 milliseconds.

Network Time Protocol (NTP)

- Uses a network of time servers to synchronize all processors on a net.
- Time servers are connected by a synchronization subnet tree.
 - The root is adjusted directly.
 - Each node synchronizes its children nodes.



Thank you



Remember:

Next week we have a tutorial after this lecture!

For general enquiries, contact: aidan.murphy@ucd.ie