Distributed Systems:
Course Information

Dr Soumyabrata DEV
https://soumyabrata.dev/

School of Computer Science
University College Dublin

Beijing-Dublin International College

Where have I been, and where am I now?

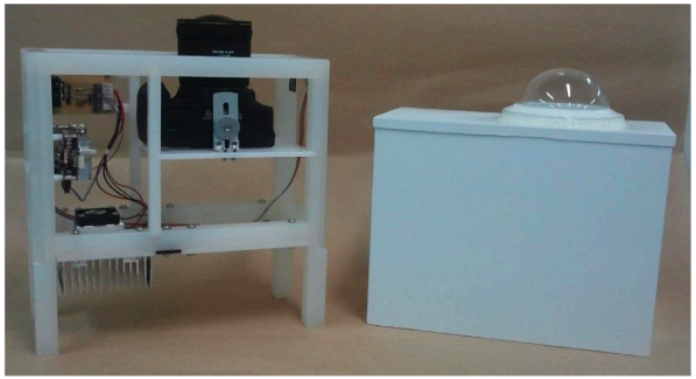
- Assistant Professor in Computer Science, University College Dublin, 2019 – *till date*
- Postdoc at ADAPT SFI Research Centre, Trinity College Dublin, Ireland (2017-2019)
- Ph.D. in Electrical Engineering, NTU Singapore (2012-2017).
- Visiting Doctoral Assistant, EPFL Lausanne (Aug-Dec 2015).
- Engineer at Ericsson (2010-2012).
- Undergrad in electronics engineering, NIT Silchar, India (2006-2010).

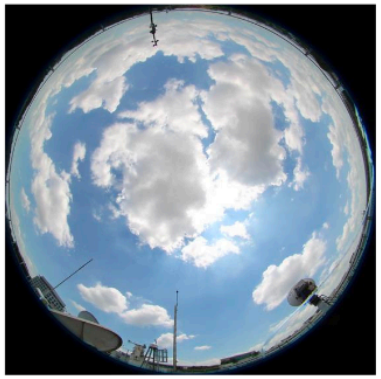
My high-level research interests

My research interests are primarily in the field of image processing, computer vision, machine learning and deep learning.

Snippet of my research

Understanding events in the atmosphere using ground-based cameras.





More details here: https://soumyabrata.dev/research/

Some Information

Marking Scheme:

- Examination 60%
- Case study 20%
- Tutorials 20%

Course Duration:

13 weeks of lecturing + 1 week of class revision

- Every Tuesday, 08:00 to 09:35
- Teaching building 4, Room 102
- We will have 13 weeks of tutorials.
  - Every Friday, 08:00 to 09:35
  - Teaching building 3, Room 301

Reading:

- "Distributed Systems: Concepts and Designs", by G. Coulouris, J. Dollimore, T. Kindberg and G. Blair, 4th Edition, Addison-Wesley, ISBN 0-321-26354-5
- Academic Papers

Slides & Notes

- Slides will be on the Moodle:
- New MOODLE for Computer Science Students

  https://csmoodle.ucd.ie/
- Login using your UCD connect account
- Click on COMP3008J Distributed Systems 2019 – 2020
- And enrolment key is **COMP3008J-2019-2020**

- Head Teaching Assistant:
  Xingyu Pan (Star)
  email: Xingyu.Pan@ucdconnect.ie

Plagiarism

When submitting tutorials and assignments, you should be aware that plagiarism is very serious.

Any work you submit should be you own work.

- Copying directly from the internet is not acceptable.
- Copying work from a classmate is not acceptable.

I want to see what *you* know and how *you* would describe or progam things.

Course Content

In this module we will mostly be talking about *Distributed Systems*.

This is when we wish to run a software system that uses many computers.

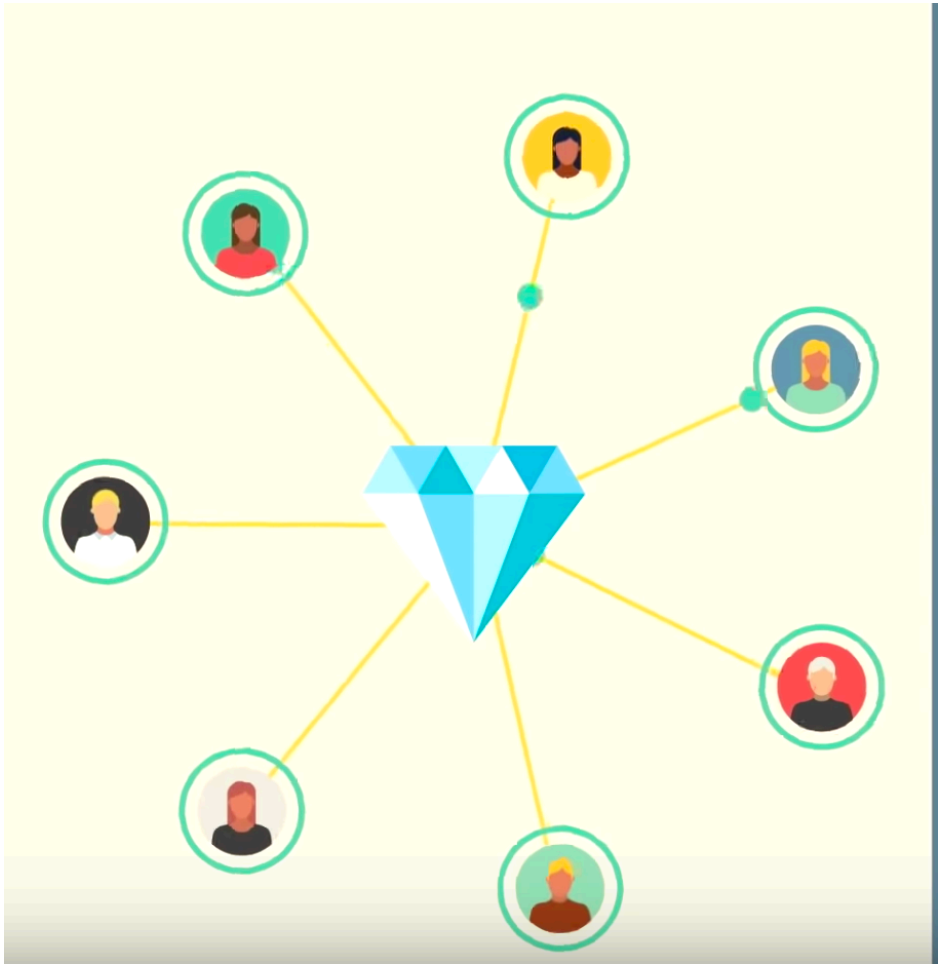
We will look at some of the important challenges in this area.

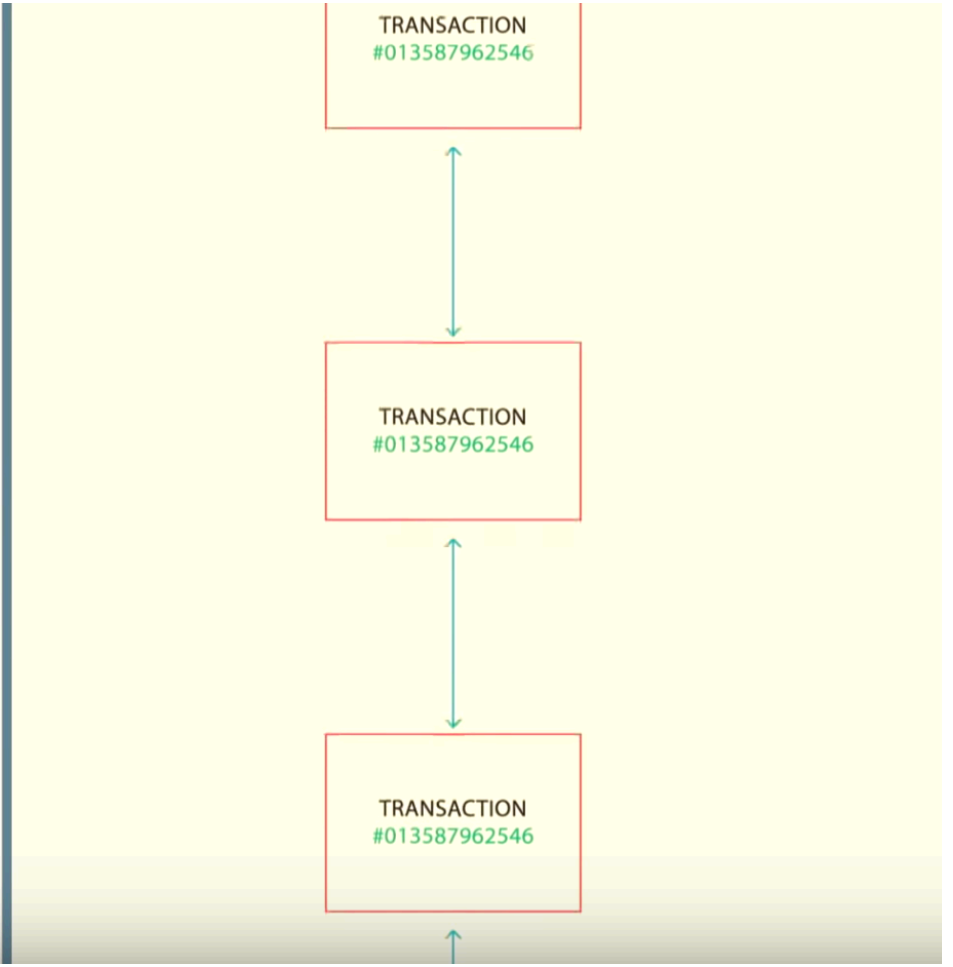
Then we will look at different types of distributed system in more detail.

Distributed Systems:
Introduction

These course slides are adapted from the original course slides prepared by Dr Anca Jurcut, University College Dublin.

Blockchain: A distributed system





Today's Objectives

To understand what is a Distributed System?

To explore some examples of Distributed Systems

To understand the benefits / issues / challenges associated with Distributed Systems

To begin investigating various Distributed System Models.

What is a Distributed System?

The term *Distributed System* has been used to describe a wide range of computer systems:

- from weakly-coupled systems such as the Internet,
- to very strongly-coupled systems such as multi-processor systems.

This course adopts the view of a *Distributed System* as:

- a collection of independent computers that appear to the users of the system as a single computer.

Or, more specifically:

- any system that consists of hardware or software components located at networked computers that communicate and coordinate their actions via *message passing*.

Why Use Distributed Systems?

The goal behind the design of this type of *Distributed System* is often to connect users and resources in a *transparent*, *open*, and *scalable* way.

Why Use Distributed Systems?

The goal behind the design of this type of *Distributed System* is often to connect **users** and resources in a *transparent*, *open*, and *scalable* way.

- The entity using the distributed system.

  E.g. human users, external systems

Why Use Distributed Systems?

The goal behind the design of this type of *Distributed System* is often to connect users and **resources** in a *transparent*, *open*, and *scalable* way.

- The entity using the distributed system.
- Abstract concept covering hardware and/or software entities.

  E.g. disks, printers, files, databases, …

Why Use Distributed Systems?

The goal behind the design of this type of *Distributed System* is often to connect users and resources in a ***transparent***, *open*, and *scalable* way.

- The entity using the distributed system.
- Abstract concept covering hardware and/or software entities.
- Hiding the actual structure of the system from the user.
  - We don't always know where the resources are really located

Why Use Distributed Systems?

The goal behind the design of this type of *Distributed System* is often to connect users and resources in a *transparent*, ***open***, and *scalable* way.

- The entity using the distributed system.
- Abstract concept covering hardware and/or software entities.
- Hiding the actual structure of the system from the user.
- Supporting the addition / removal of resources at run-time
  - Can we add a new type of printer to our system?

Why Use Distributed Systems?

The goal behind the design of this type of *Distributed System* is often to connect users and resources in a *transparent*, *open*, and ***scalable*** way.

- The entity using the distributed system.
- Abstract concept covering hardware and/or software entities.
- Hiding the actual structure of the system from the user.
- Supporting the addition / removal of resources at run-time
- The system can expand to meet increased demand

  Does the system continue to work effectively as the number of users increases?

Why Use Distributed Systems?

The goal behind the design of this type of *Distributed System* is often to connect users and resources in a *transparent*, *open*, and ***scalable*** way.

- The entity using the distributed system.
- Abstract concept covering hardware and/or software entities.
- Hiding the actual structure of the system from the user.
- Supporting the addition / removal of resources at run-time
- The system can expand to meet increased demand

  Does the system continue to work effectively as the number of users increases?

When developing a Distributed System, the designer must consider what level of transparency, openness and scalability that system will require.

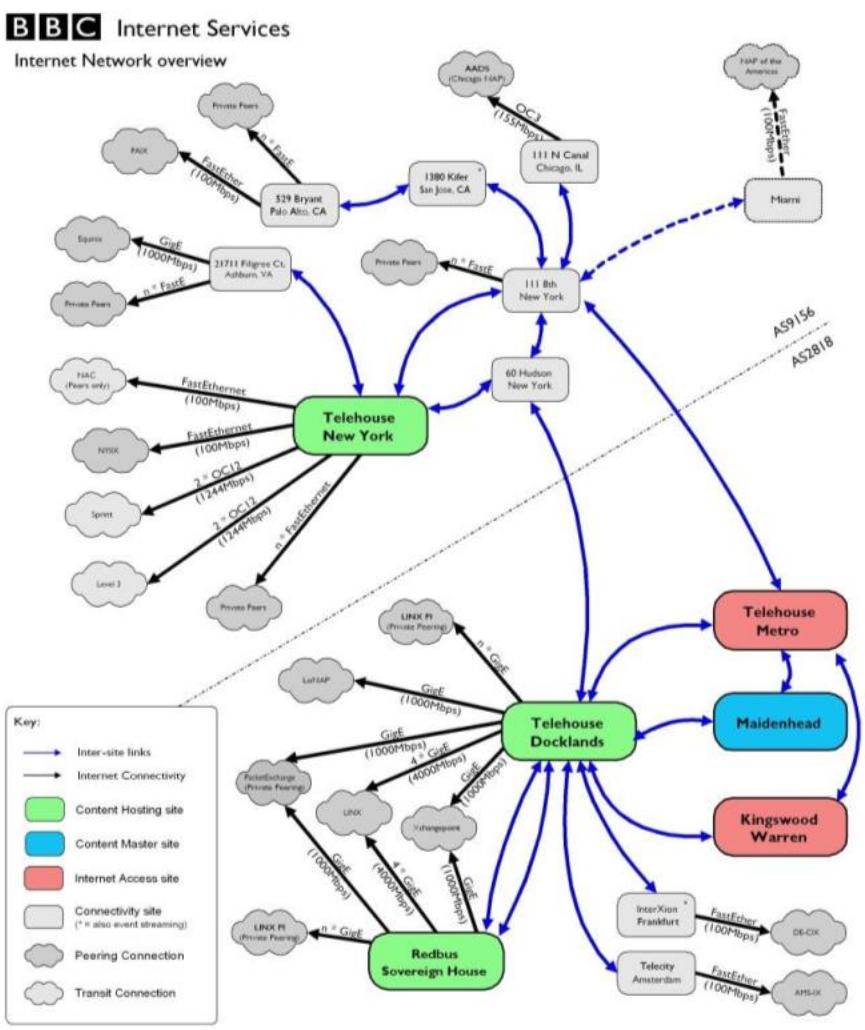
Distributed Systems:
Examples

Example: The Internet

A vast interconnected collection of computer networks of many different types.

Programs running on computers connected to it interact by passing messages (packets) using a common means of *communication*.

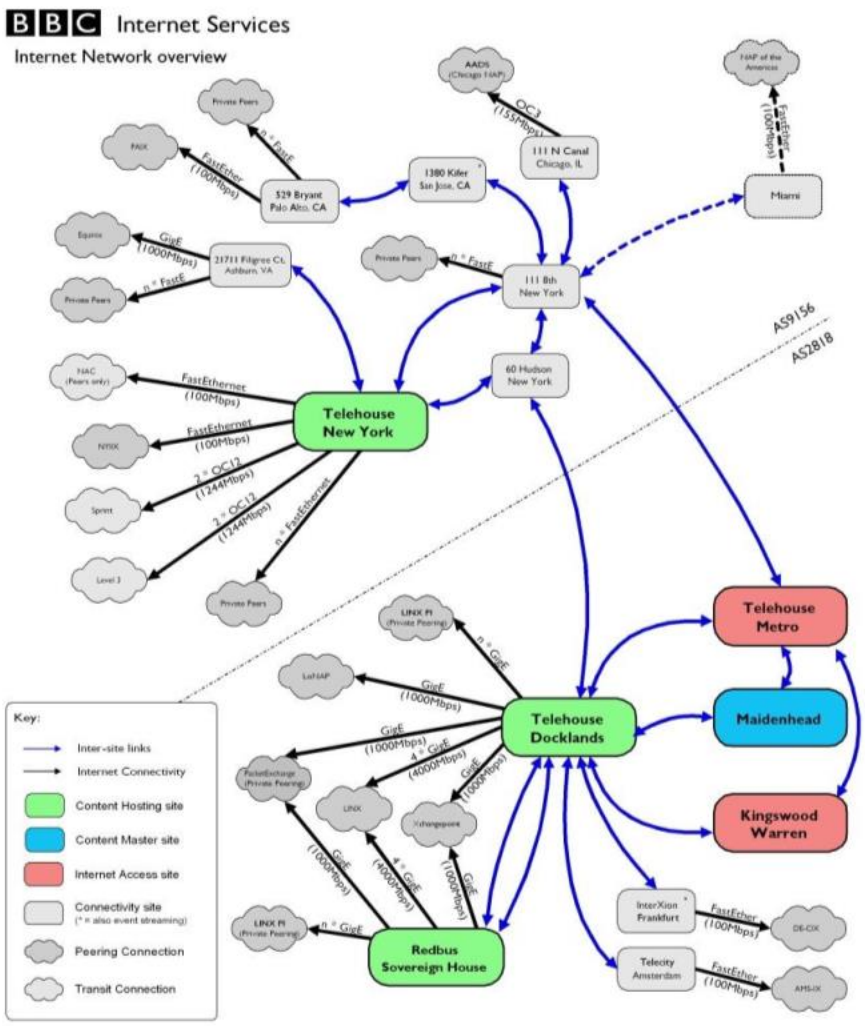
It allows users to access services such as the World Wide Web (*resources*).



Example: The Internet

A vast interconnected collection of computer networks of many different types.

- *Transparency* is achieved through the use of logical network addresses defined via the Internet Protocol (IP).
- It is *open* in that new resources can be easily added / removed.
- It can *scale* up to around 4 billion devices (IPv4).



Class Question

Can you think of any other examples of Distributed Systems?

- How does its components communicate?
- What resources does it use?
- Is it transparent?
- Is it open?
- Is it scalable?

Example: Mobile Computing

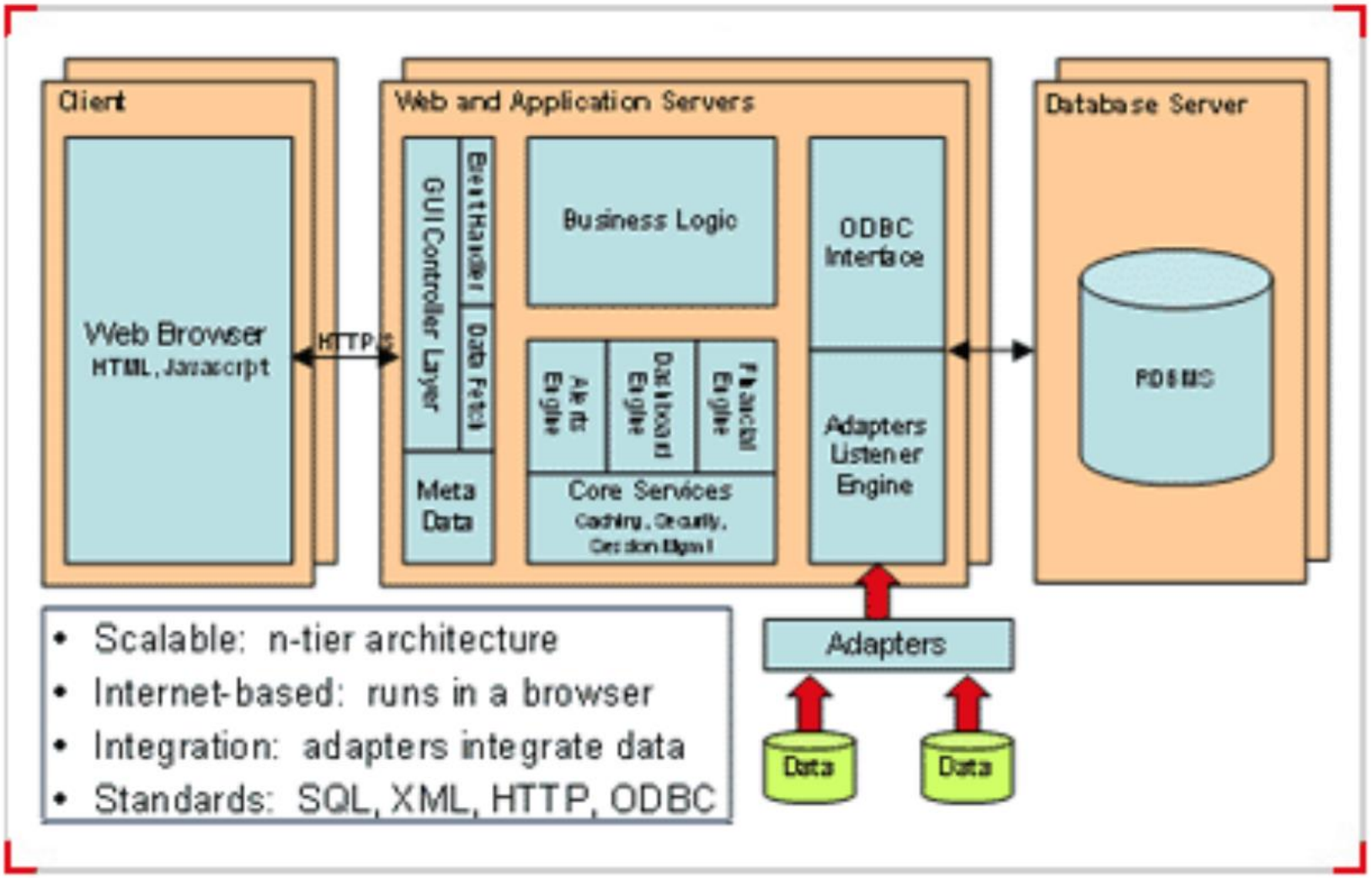
Example: “Bump” is an app for iPhone and Android.

Transfers files between phones.

Each phone detects a “bump” and communicates with a cloud servers to give location and timing information.

The phones also require data from GPS satellites.

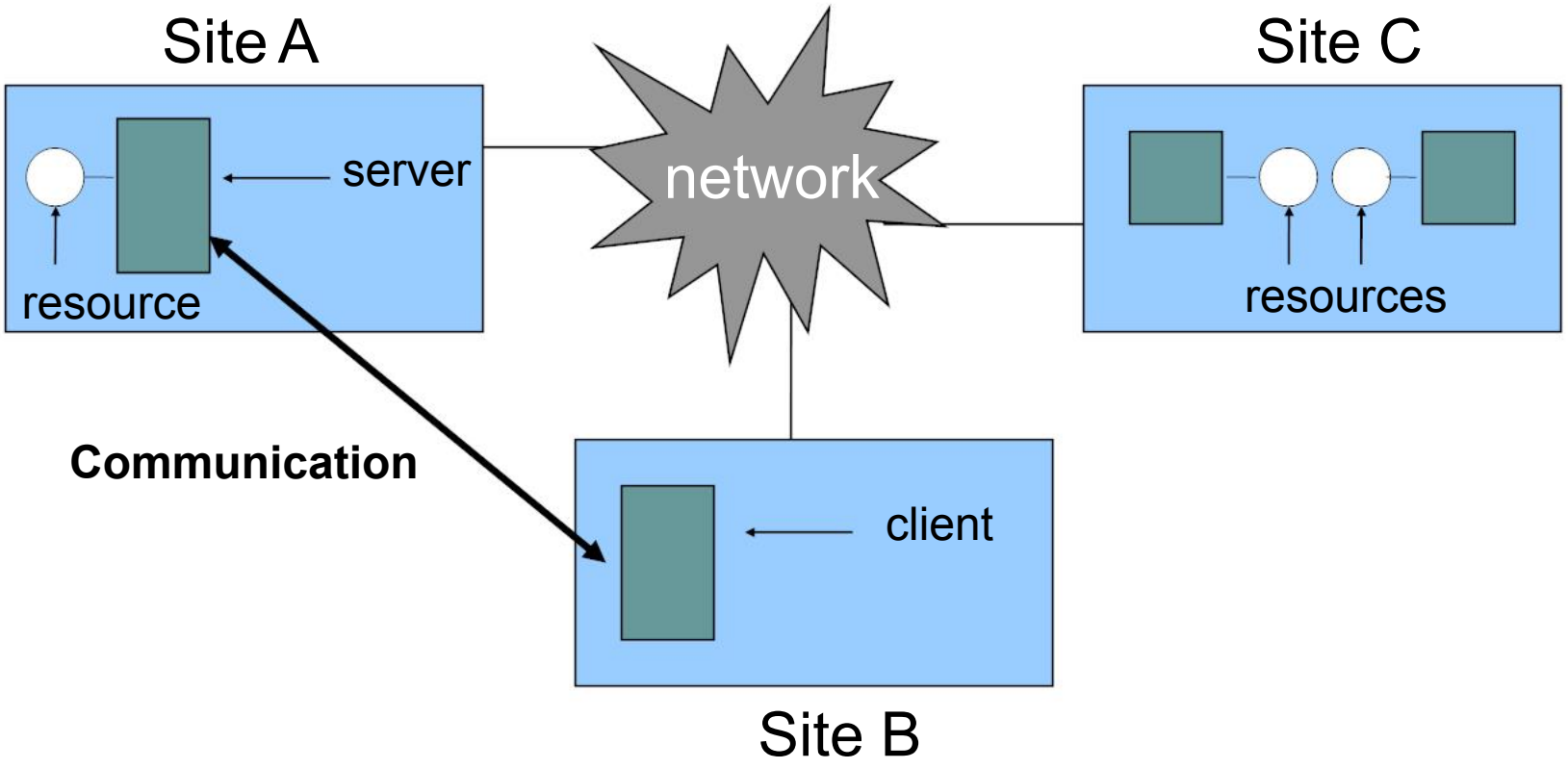
Example: Web Application



Distributed Systems:
Terminology

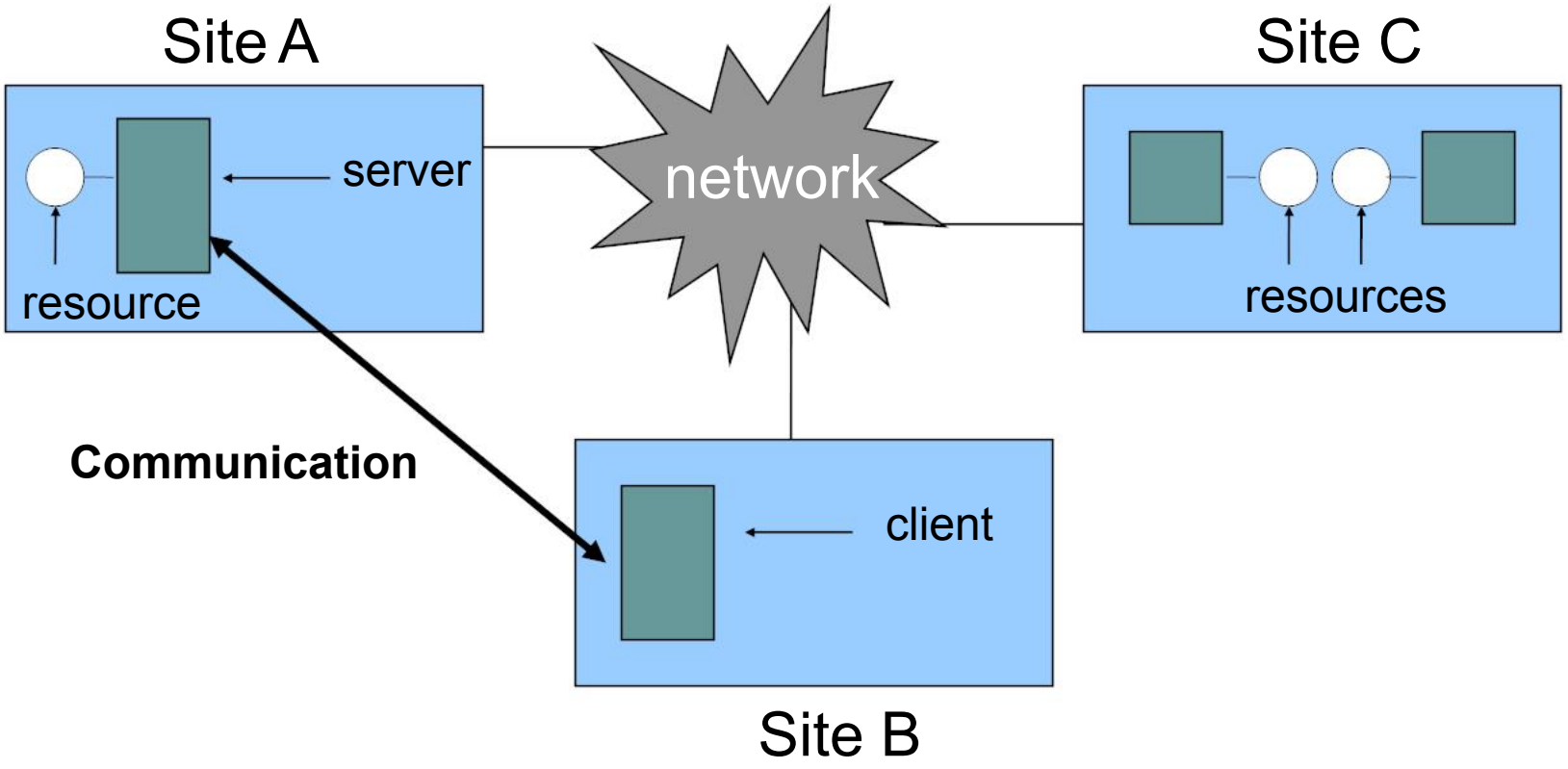
Terminology

A *site* is a geographical location that contains one or more hosts in the DS (Distributed System).



Terminology

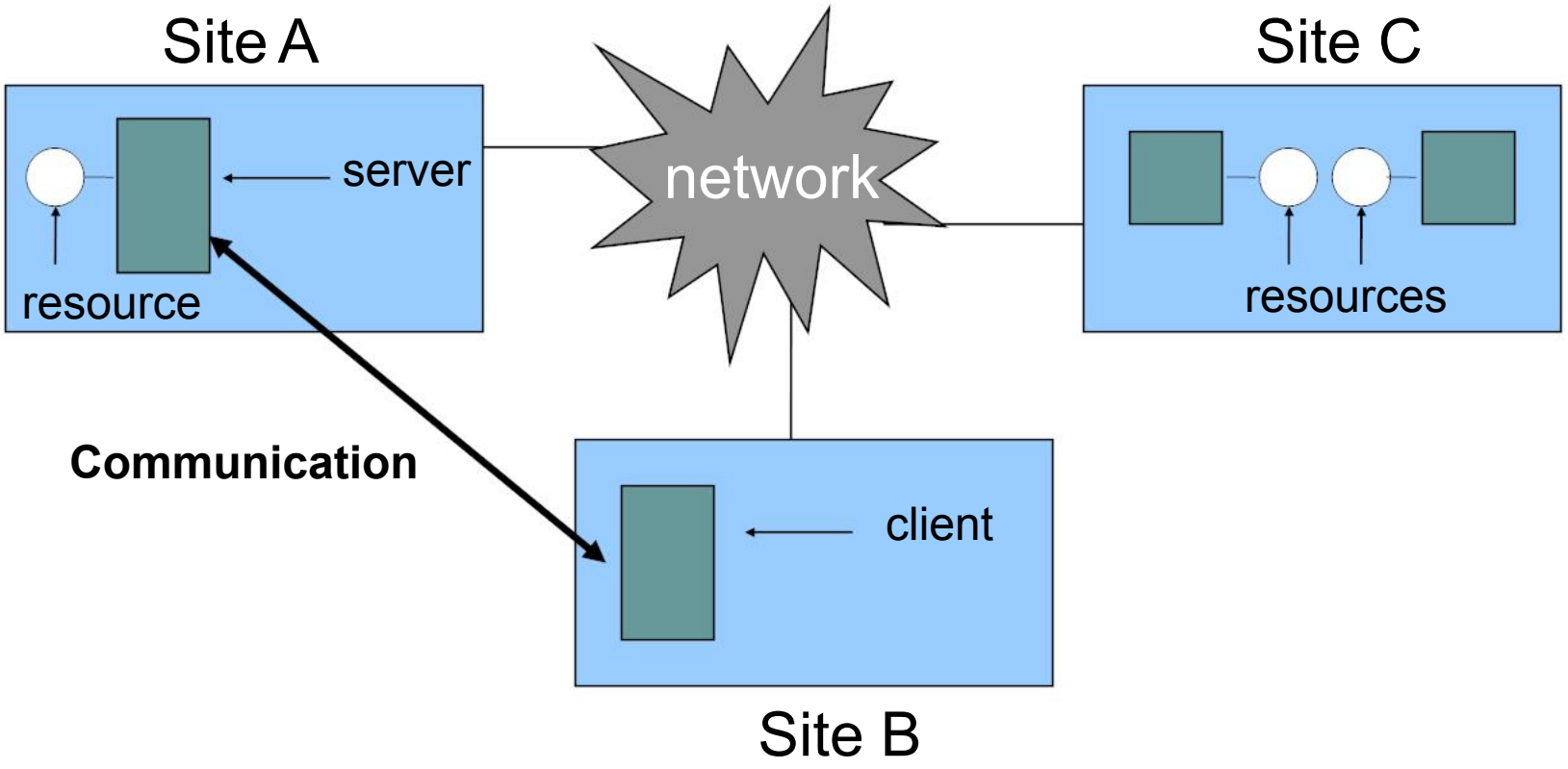
A *host* (also known as a *node* or a *machine*) is a specific machine that is connected into the distributed system.



Terminology

Hosts can be *clients* and/or *servers*:

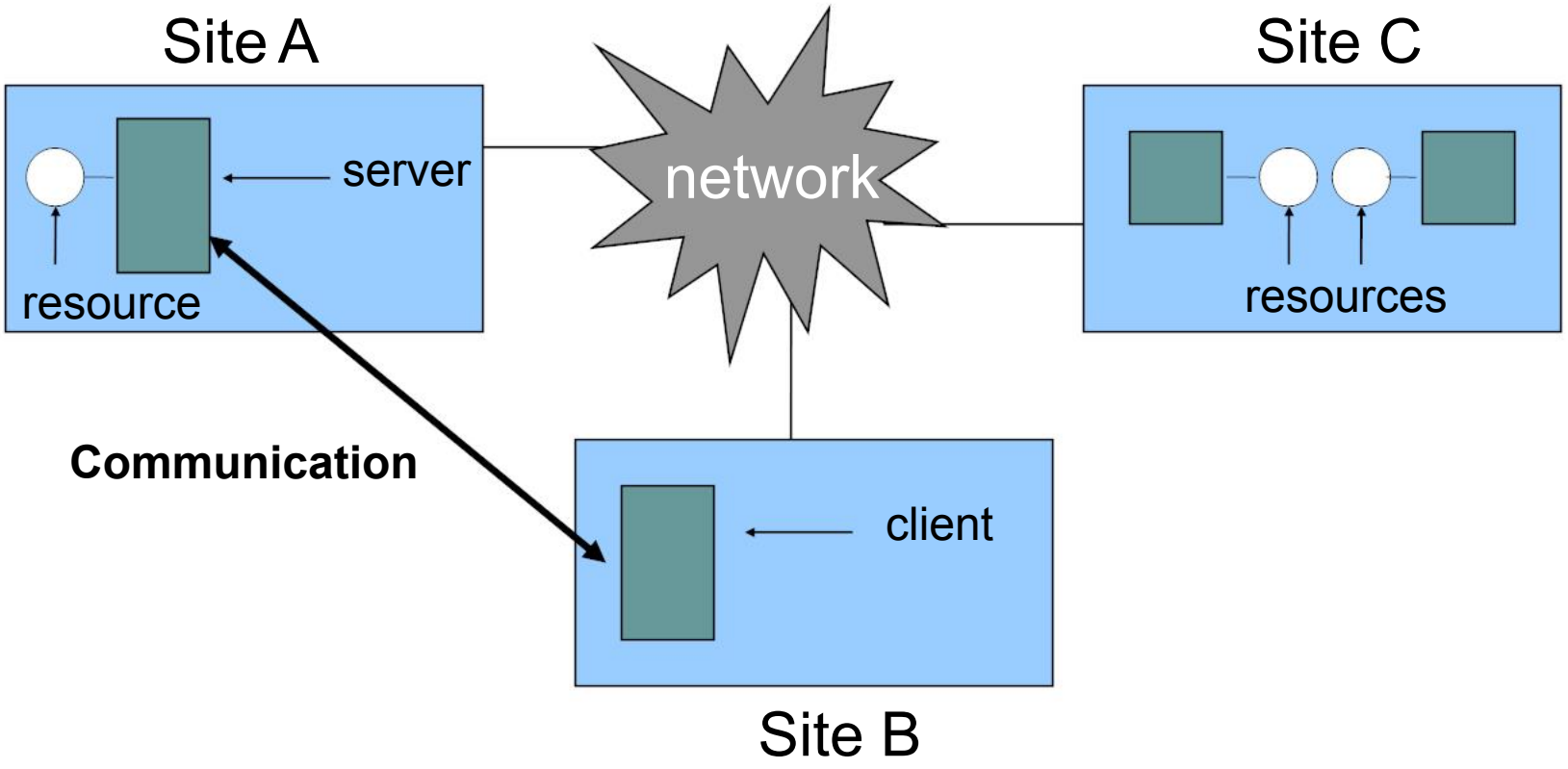
- A *server* makes resources available to the DS.
- A *client* accesses the servers and utilizes any available resources.



Terminology

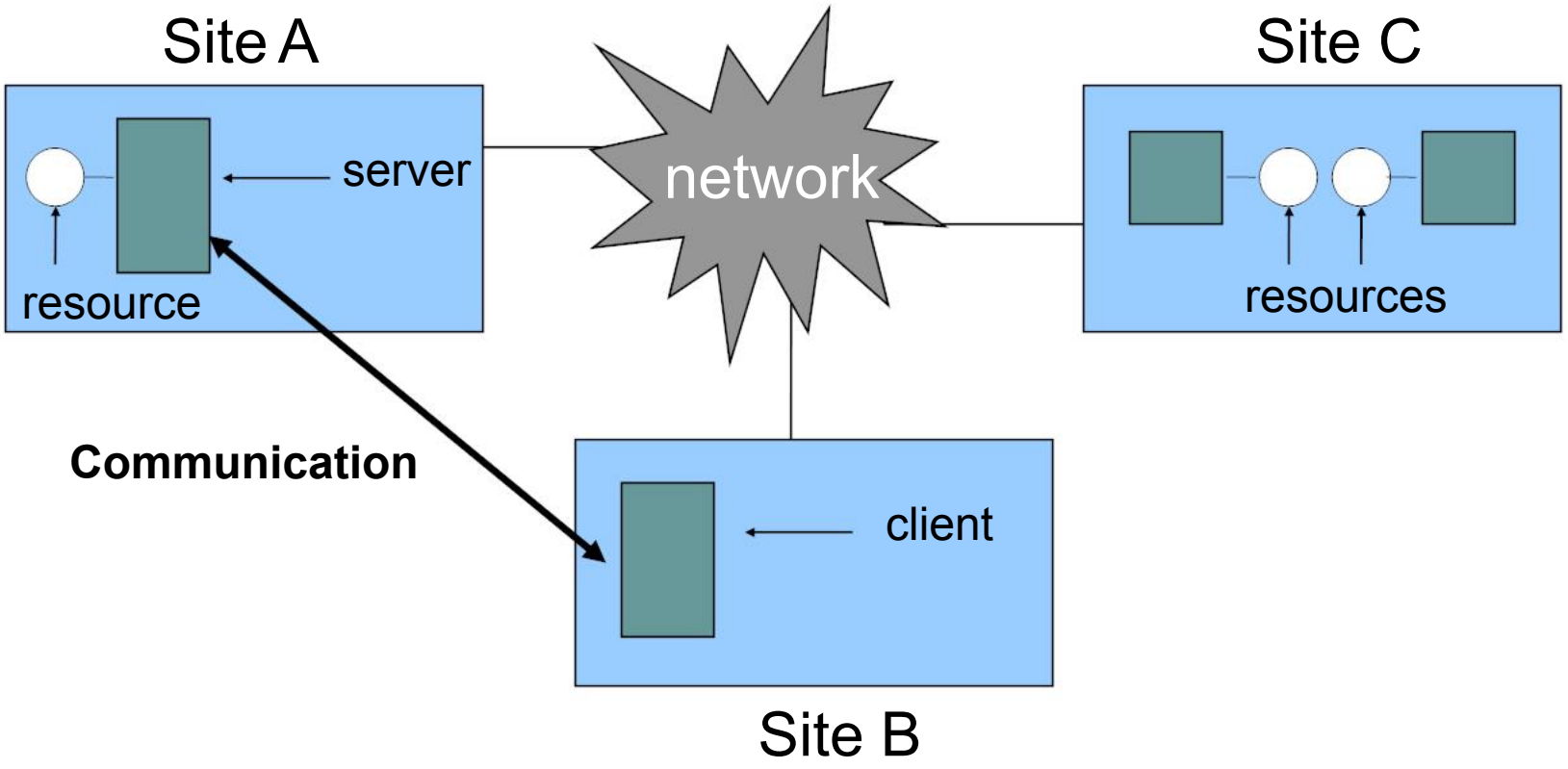
A *resource* is a program or data that resides on a host.

- Programs may be executed remotely on the host.
- Data may be loaded from / saved to a given host.



Terminology

A *task* (also known as a *computation* or *job*) is a specific set of instructions that a users asks the distributed system to execute



Distributed Systems:
Benefits, Issues and Challenges

Benefits, Issues and Challenges

Creating a Distributed System is not trivial:

- It requires the decomposition of the system in to a number of components.
- At the very least, we must consider:
  - what these components will do
  - where they will be located
  - who each component will interact with
  - how they will interact with one another to realise system goals
- This can often result in a system that is more complex that a single centralised system.

So, why do it?

- What are the benefits?
- What are the issues?
- What are the challenges?

Resource Sharing

Each node can access and utilize the available resources of the other nodes in the system.

Examples:

- At a given point in time, a user at node A may access and utilize a laser printer resource (via a print daemon) located at node B.
- At the same time, a user at node B may access a file (via a network file server) that resides on node A.

Benefits:

- This means that we can directly access resources that we would otherwise not be able to.
- We can share our own resources with our friends / colleagues.

Issues:

- How do we control access to the resource?
- What happens if a resource becomes unavailable?

Concurrency

Multiple machines allows multiple users to work in parallel.

- This means the system must be able to handle multiple tasks concurrently.
- The tasks themselves can be decomposed into subtasks that can be processed at the same time on different machines.

Examples:

- Five different users attempt to print a 100 page document at the same time.
- 10,000 users attempt to access the same website at the same time.
- Consider an online auction: If two concurrent bidders (Smith and Jones) place bids concurrently, how can we ensure that their bids are recorded correctly.

Concurrency

Benefits:

- Computational Speedup – the system can exploit parallelism to reduce the time taken to complete a task.

Issues:

- Too many users trying to do the same thing can lead to resources becoming overloaded, thus increasing the time taken to complete the task.
- Concurrency has costs in terms of task management, network communication overheads, …
- How can we ensure that (possibly conflicting) operations are performed correctly?

Global Time

Some tasks carried out by a distributed system require a shared concept of time.

Examples:

- In a banking system, all transactions must be recorded using a single global clock to ensure that those transactions are applied to customer accounts in the correct order.
- Two spray paint robots in a car manufacturing plan that try to coordinate their activities in-order to speed painting of the car.

Issues:

- A Distributed System has, by default, no global clock, therefore any solution requiring a global clock must include an appropriate implementation.
- Unfortunately, all current approaches to implementing a global clock (as we will see in our future classes) provide only limited accuracy.

Reliability

Reliability refers to the ability of the system to continue functioning after the failure of a component.

- *System perspective*: If tasks are distributed over a number of nodes, then the failure of one node will not affect the operation of the other nodes.
- *User perspective*: If a particular resource becomes available, I can search the distributed system for another suitable resource.

Reliability is often related to the level of redundancy in the system.

- i.e. how many copies there are or the key components.
- *In theory*: A Distributed System is more reliable than a single-processor system due to replication of resources, and distribution of work load.
- *In practice*: This is not always the case!

Scalability

Scalability is concerned with how well can a system handle increasing numbers of users and resources.

- In theory, a DS would scale well, because all we have to do is add more machines and make the rest of the system aware of them.
- Unfortunately, this is NOT EASY to do in practice!

Benefits:

- The system can adapt to changing requirements both in terms of number of users and number of resources.

Issues:

- Does the new machine have the right resources?
- What impact does adding the machine have on network usage?
- How does the addition of a machine affect the reliability of the system?

Distributed System Challenges

Heterogeneity

- The heterogeneity of a Distributed System refers to its ability to work with different:

  Networks, Computer Hardware, Operating Systems, Programming Languages, Implementations by Different Developers

Examples:

- the Internet supports heterogeneity through the Internet Protocol Suite.
- Communication between programs written in different languages must cater for different type-encoding schemes (e.g. int, String)
- Programs developed for standardised frameworks must operate on all implementations of that framework (e.g. EJB, Servlets).

Distributed System Challenges

Openness

Openness refers to the degree to which a system is extensible.

- In DS, this often refers to the potential for adding and integrating new resource-sharing services.
- Openness requires that key software interfaces be made available to developers (published) and that a uniform communication mechanism be defined.
- Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors.

In practice, openness can often only be achieved on a limited scale.

Distributed System Challenges

Security

Security is concerned with the protection of (information) resources that may be of value to their owner, but which need to be shared.

- E.g. credit card information on the internet, police access to regional files.

Security of this information often takes two forms:

- Security of information transmitted within the distributed system (encryption).
- Security of access to a resource (proof of identity).

In publicly accessible systems, the security challenge must also deal with issues such as Denial of Service Attacks.

Distributed System Challenges

Failure Handling

Computer systems sometimes fail.

- Hardware faults may cause errors in processes, or even stop the process from finishing.
- In a distributed system, failure is partial – the failure of one component does not cause the whole system to fail.

In designing a distributed system we must consider:

- Failure Detection
- Failure Masking (hiding the failure)
- Failure Tolerance
- Recovery from Failure
- Redundancy
- Level of Availability
  - 99.9% uptime (3 nines) = 8.76 hours downtime per year
  - 99.999% uptime (5 nines) = 5.26 minutes downtime per year

Distributed System Challenges

Transparency

The concealment from the user of the specific set of components that make up a distributed system.

[ISO 1992] Reference Model for Open Distributed Processing:

- *Access*: enables local and remote resources to be accessed using identical operations (e.g. folder containing remote & local files).
- *Location*: enables resources to be accessed without knowledge of their physical or network location (e.g. domain based URLs).
- *Concurrency*: enables several processes to operate concurrently using shared resources without interference between them.
- *Replication*: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas.

Distributed System Challenges

Transparency

The concealment from the user of the specific set of components that make up a distributed system.

[ISO 1992] Reference Model for Open Distributed Processing:

- *Failure*: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components (e.g. emails delivered even when a mail server is down).
- *Mobility*: allows the movement of resources and clients within a system without affecting the operation of users or programs.
- *Performance*: allows the system to be reconfigured to improve performance as loads vary.
- *Scaling*: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Distributed Systems: System Models

System Models

Real world Distributed Systems (DS) should be designed to function correctly in the widest range of circumstances and in the face of many possible difficulties and threats, such as:

- *Widely varying modes of use*: The components of a DS are subject to wide variations in workload.
  - the main page of the **amazon.com** website is accessed millions of times a day
  - multimedia applications require high bandwidth and low latency
- *Wide range of system environments*: A distributed system must accommodate heterogeneous hardware, operating systems, and networks.
  - wireless networks operate at a fraction of the speed of local networks
  - a system deployment may range from tens of computers to millions of computers
- *Internal Problems*: Non-synchronized clocks, conflicting data updates, hardware failure, …
- *External Threats*: Attacks on data integrity and security, denial of service, …

Architectural Models

Defines the set of components that make up a system and the interplay between those components.

It combines:

- A simplified functional model of the individual components
- The placement of components across a network of computers
- The interrelationships between the components

At the core of this model is an initial classification of components into

- *Server processes*: the provider of a service
- *Client processes*: the invoker (user) or a service
- *Peer processes*: processes that cooperate and communicate in a symmetrical manner to perform a task.

Based on this classification, the responsibilities of each process can be determined and used to asses workloads and the impact of failures for each of them.

Software Architecture for DS

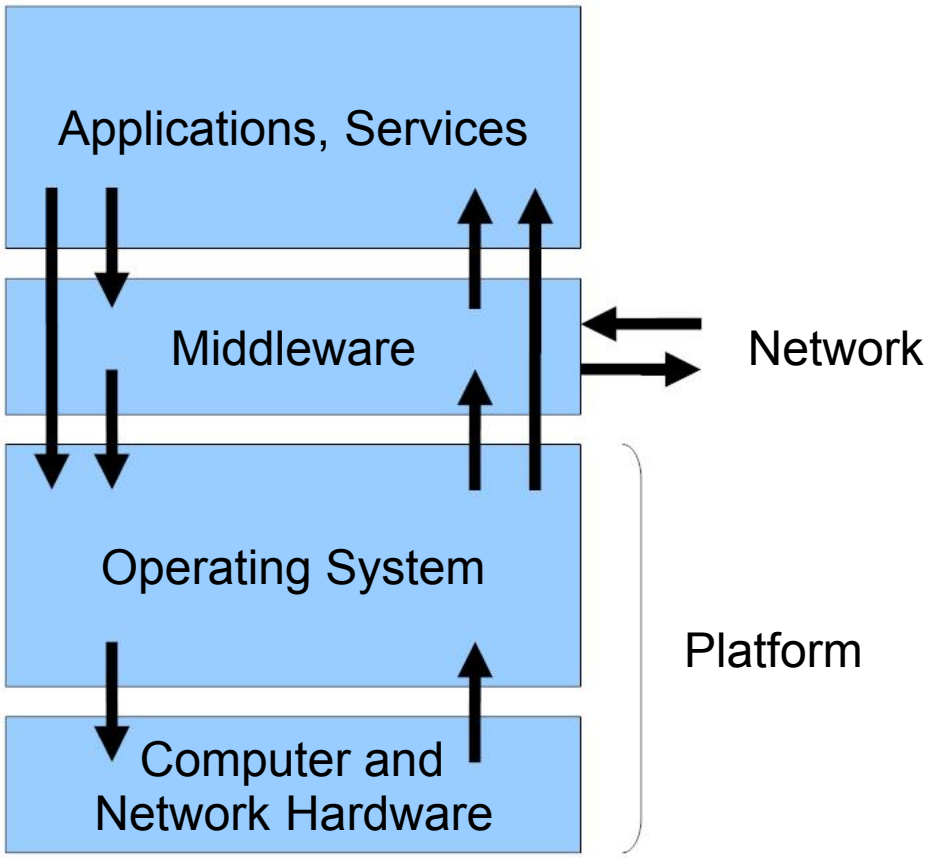
This layered model presents an abstraction of an individual computer that acts as a node within some Distributed System.

It consists of:

Platform: A combination of hardware and software upon that provides the basis for all applications.

Middleware: A layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application programmers

Applications, Services: The actual applications / services that are implemented on the node.



What is Middleware?

Middleware is the set of processes or objects in a group of computers that interact with each other to implement communication and resource-sharing support for Distributed Systems.

- It delivers an abstract layer of support that acts as a building block for the construction of the target application(s).

It provides high-level features such as:

- Remote Method Invocation
- Support for communication between groups of processes
- Event Notification
- Support for partitioning, placement, and retrieval of shared data objects amongst cooperating computers
- Support for the replication of shared data objects
- Real-time Multimedia Transmission

Examples of Middleware:

- Sun RPC, CORBA, Java RMI, Web Services, Microsoft's Distributed Component Object Model (DCOM).

Limitations of Middleware

Middleware is an abstracted communication infrastructure component that aims to simplify the process of implementing a Distributed System.

The services offered by middleware are based on a number of assumptions about the nature of interaction between components of a Distributed System.

For example, a common assumption made is that all communication is *atomic*.

While for most systems this is sufficient, it is not for others:

- A reliable FTP service employs a non-standard form of interaction in which the client downloading the file keeps track of the progress.
- If the download fails at any point, then the client requests that the download be restarted from the point at which the failure occurred.
- This kind of service cannot normally be implemented through middleware.

Limitations of Middleware

More generally, this is known as the “end-to-end argument” (Saltzer et al., 1984):

- “Some communication-related functions can be completely and reliably implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that function as a feature of the communication system itself is not always sensible. (An incomplete version of the function provided by the communication system may sometimes be useful as a performance enhancement)”

The point here is:

- The correct behaviour of a distributed system depends on checks, error-correction mechanisms and security measures at many levels.
- Some require access to data that is specific to the application meaning that communication system checks alone are insufficient.
- So, the remaining checks must be carried out in the application layer, leading to a potential duplication of work.

Thank you

For general enquries, contact:

Please contact the Head Teaching Assistant: Xingyu Pan (Star), Xingyu.Pan@ucdconnect.ie