ECS656U/ECS796P
Distributed Systems

What this course is about

The Internet interconnects billions of machines, ranging from high end servers to limited capacity embedded sensing devices. Distributed systems are built to take advantage of multiple interconnected machines and achieve common goals with them.

What this course is about

The Internet interconnects billions of machines, ranging from high end servers to limited capacity embedded sensing devices. Distributed systems are built to take advantage of multiple interconnected machines and achieve common goals with them.

This module will cover the fundamental concepts and technical challenges of building distributed systems.

Teaching Patterns

- 2-hours lectures on Wednesdays
  - from 11am to 1pm on Blackboard Collaborate (QMplus)
  - Gianni (https://www.eecs.qmul.ac.uk/~gianni/)
  - Joseph (https://www.eecs.qmul.ac.uk/~joseph/)

- 2-hours lab session on Thursdays
  - From 11am to 1pm, in ITL or Eng.B10
  - Labs start in week 2

Agenda

01. Introduction (Gianni)
02. Synchronization (Joseph)
03. RPC RMI SOAP Threads (Joseph)
04. REST (Joseph)
05. Consensus Protocols and Paxos (Gianni)
06. Raft and Cloud Computing (Gianni)
07. Midterm
08. Multiplayer Game Synchronization (Joseph)
09. Peer-to-Peer and Distributed Hash Tables (Gianni)
10. Key-Value Stores (Gianni)
11. Bitcoin (Joseph)
12. Recap

Assessment

- Exam 40%
- Coursework 40% (more information will be provided by Joseph)
- Labs 20%
  - We will have four Labs each of them counting 5%. New labs will be released on week 2, 3, 5, 6
  - Once released, you have two weeks to submit the lab in QMplus
  - You can use the remaining lab sessions to work towards the completion of your Coursework (deadline week 11)
  - Labs and Coursework are submitted to QMplus

Introduction

Outline

Today, the lecture will focus on three main points:

- Definition of a Distributed System

- Goals of a Distributed System

- Types of Distributed Systems

Outline

Today, the lecture will focus on three main points:

- Definition of a Distributed System

- Goals of a Distributed System

- Types of Distributed Systems

Can you name some examples?

Go to www.menti.com and use code 6824 1010

Can you name some examples of Distributed Systems?

Mentimeter

2020/2021 Class

47

Can you name some examples?

- The Internet
- BiTorrent
- The Web (servers and clients)
- Hadoop
- Datacenters

What are **NOT** distributed systems?

- Humans interacting with each other (yeah, it might also be, but we are not interested in this!)

- A standalone machine not connected to the network and **with only one** process running on it

So, what are Distributed Systems?

Simple definition: Any system too large to fit on one computer! ☺

A first definition

- A collection of independent computers that appears to its users as a single coherent system

What you shall expect from us

- In this course we are <u>interested in the insides</u> of a distributed system

- We will look at:
  - What are the algorithms in place?
  - How you design or implement one?
  - How you maintain one?
  - What're their characteristics?

A definition

- So far we defined as : *"A collection of independent computers that appears to its users as a single coherent system"*.

- Not a good definition, if we want to study the internals of a distributed system…

Our definition

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

Our definition

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Each entity is a process running on some device

Our definition

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Each entity is a process running on some device
- Autonomous: it is standalone. If left “alone”, it will run just fine!

Our definition

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Each entity is a process running on some device
- Autonomous: it is standalone. If left "alone", it will run just fine!
- Programmable: you have written code that is running inside those processes

Our definition

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Each entity is a process running on some device
- Autonomous: it is standalone. If left “alone”, it will run just fine!
- Programmable: you have written code that is running inside those processes
- Asynchronous: each process runs according to its own clock

Our definition

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Each entity is a process running on some device
- Autonomous: it is standalone. If left "alone", it will run just fine!
- Programmable: you have written code that is running inside those processes
- Asynchronous: each process runs according to its own clock
- Failure-prone: those entities can fail!

Our definition

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Those entities will exchange some messages. Those messages can be dropped or delayed. We assume an unreliable communication channel!

in depth..

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Entity: a process on a device (PC, laptop, tablet)

in depth..

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Entity: a process on a device (PC, laptop, tablet)
- Autonomous: no shared memory. Each runs its own local OS and configuration parameters

in depth..

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Entity: a process on a device (PC, laptop, tablet)
- Autonomous: no shared memory. Each runs its own local OS and configuration parameters
- Programmable: now you understand why we excluded human interaction! ☺

in depth..

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Entity: a process on a device (PC, laptop, tablet)
- Autonomous: no shared memory. Each runs its own local OS and configuration parameters
- Programmable: now you understand why we excluded human interaction! ☺
- Asynchronous: distinguishes distributes systems from parallel systems (e.g., multiprocessor systems)

in depth..

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

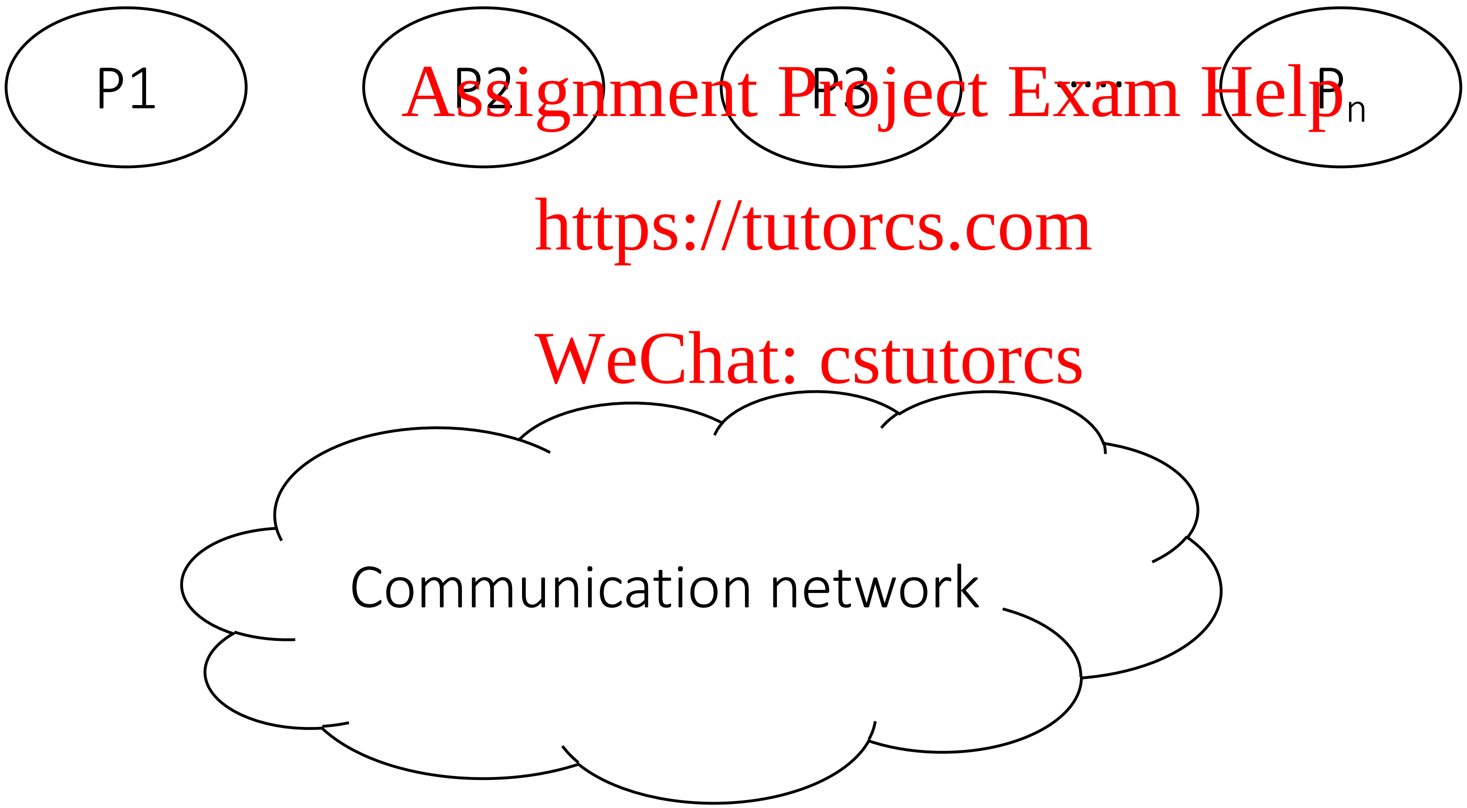
- Entity: a process on a device (PC, laptop, tablet)
- Autonomous: no shared memory. Each runs its own local OS and configuration parameters
- Programmable: now you understand why we excluded human interaction! ☺
- Asynchronous: distinguishes distributes systems from parallel systems (e.g., multiprocessor systems)
- Failure-prone: a PC, laptop, tablet can easily crash!

in depth..

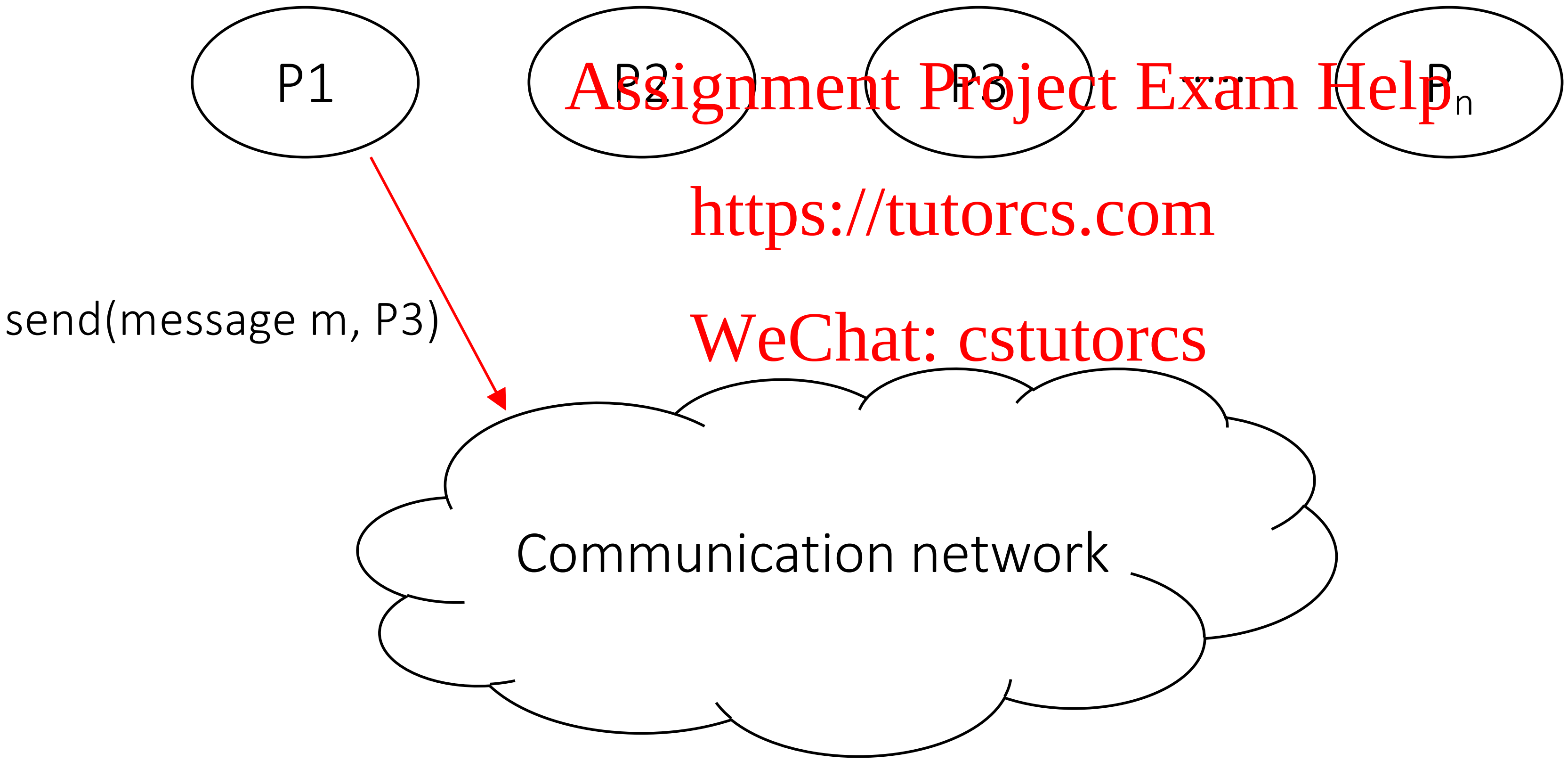
A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- Communication medium: Wireless/ Wired

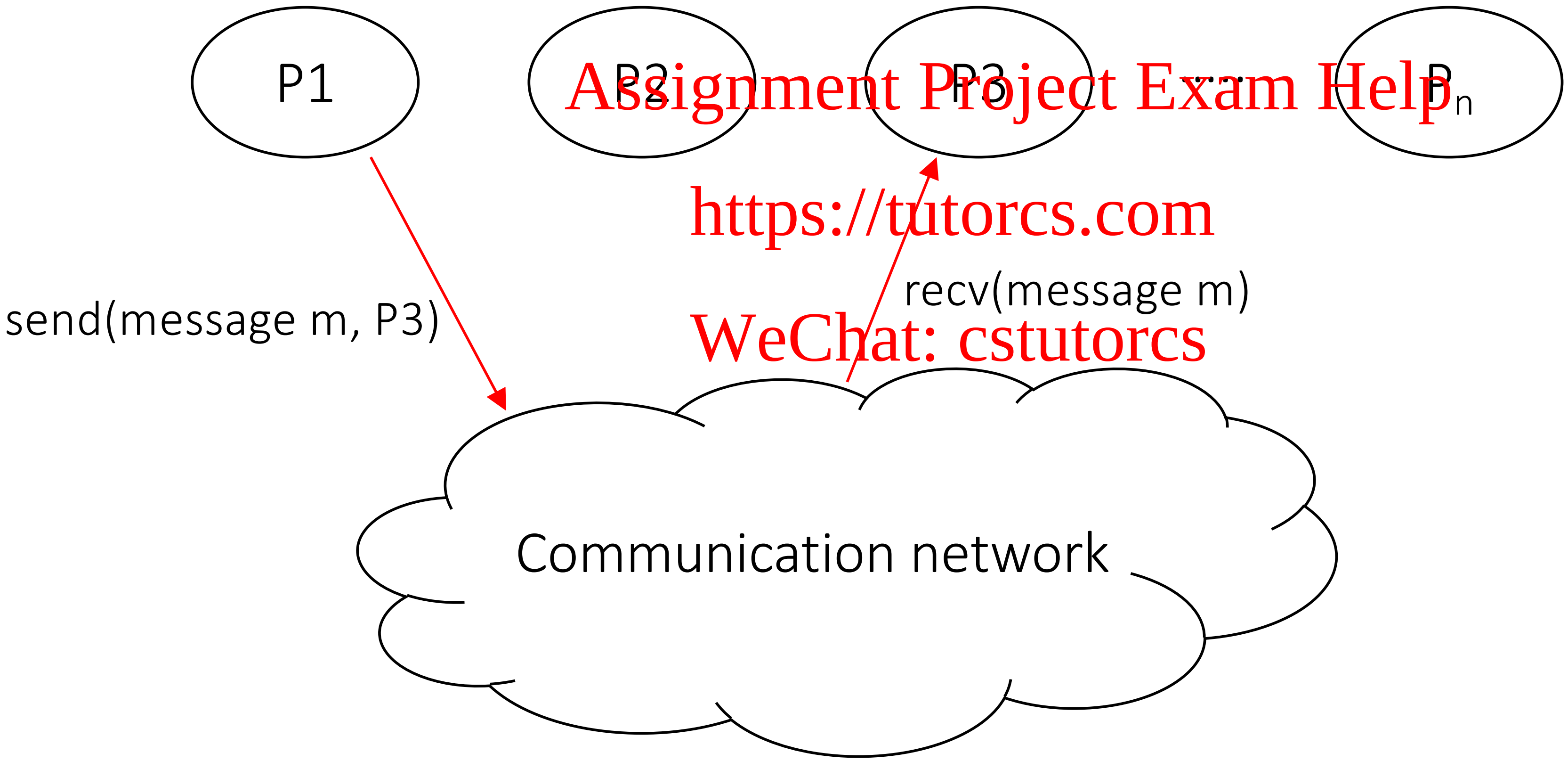
Distributed Systems in a figure



Distributed Systems in a figure



Distributed Systems in a figure



Food for researchers!

- **Peer to peer systems:** computers connected to each other via the Internet (Gnutella, Kazaa, BitTorrent)

- **Cloud infrastructures:** HW and SW components needed to support the computing requirements of a cloud model (AWS, Azure, Google Cloud)

- **Cloud storage:** a service model in which data is maintained, managed and backed up remotely and made available over a network (Key-value stores, NoSQL, Cassandra)

- **Cloud programming:** how to take advantage of a distributed resources for processing (MapReduce, Storm)
- **Coordination:** how to coordinate the resources (Paxos, Raft)
- Managing many clients and servers concurrently

Many challenges around..

- **Failures:** no longer the exception, but rather a norm (Microsoft in "Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis" in ACM SIGCOMM 2015)

- **Scalability:** 1000s of machines and Terabytes of data

- **Asynchrony:** clock skew and clock drift (you cannot fully rely on message timestamps between machines)

- **Concurrency:** 1000s of machines interacting with each other accessing the same data

The idea behind all of this

Present a single-system image so the distributed system “looks like” a single computer rather than a collection of separate computers

- Hide internal organization, i.e., communication details
- Provide a uniform interface

The idea behind all of this

Present a single-system image so the distributed system “looks like” a single computer rather than a collection of separate computers

- Hide internal organization, i.e., communication details
- Provide a uniform interface

Why this is good?

The idea behind all of this

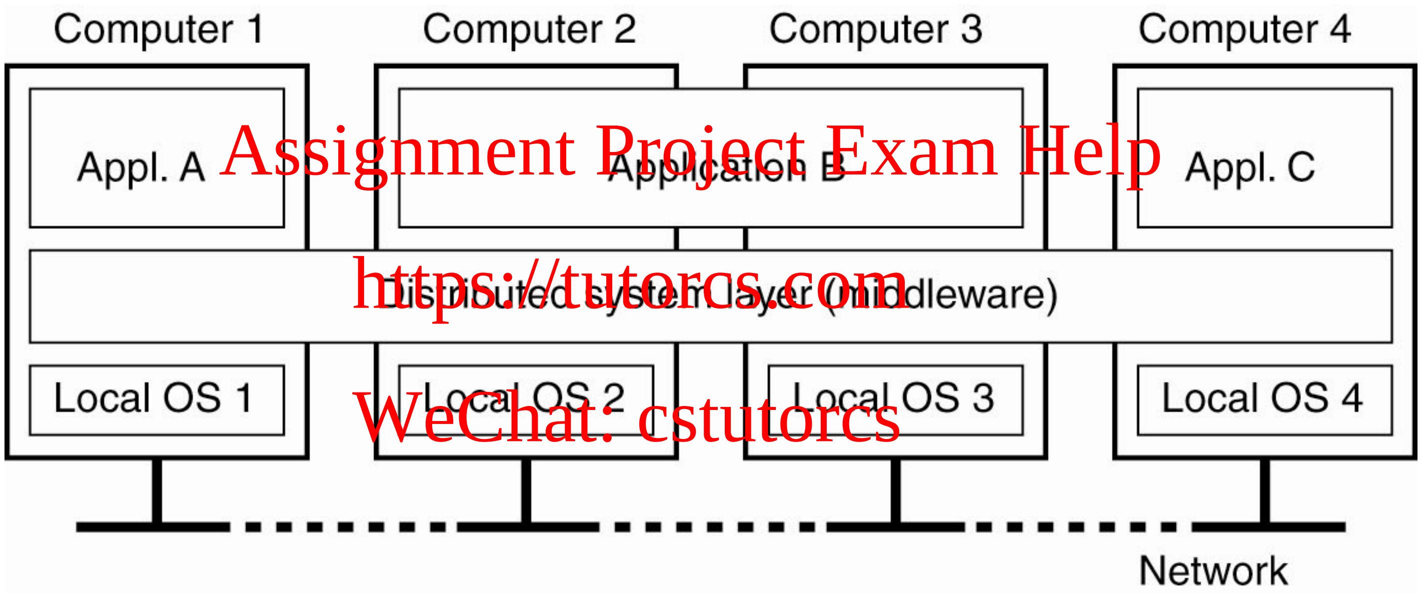
Present a single-system image so the distributed system “looks like” a single computer rather than a collection of separate computers

- Hide internal organization, i.e., communication details
- Provide a uniform interface

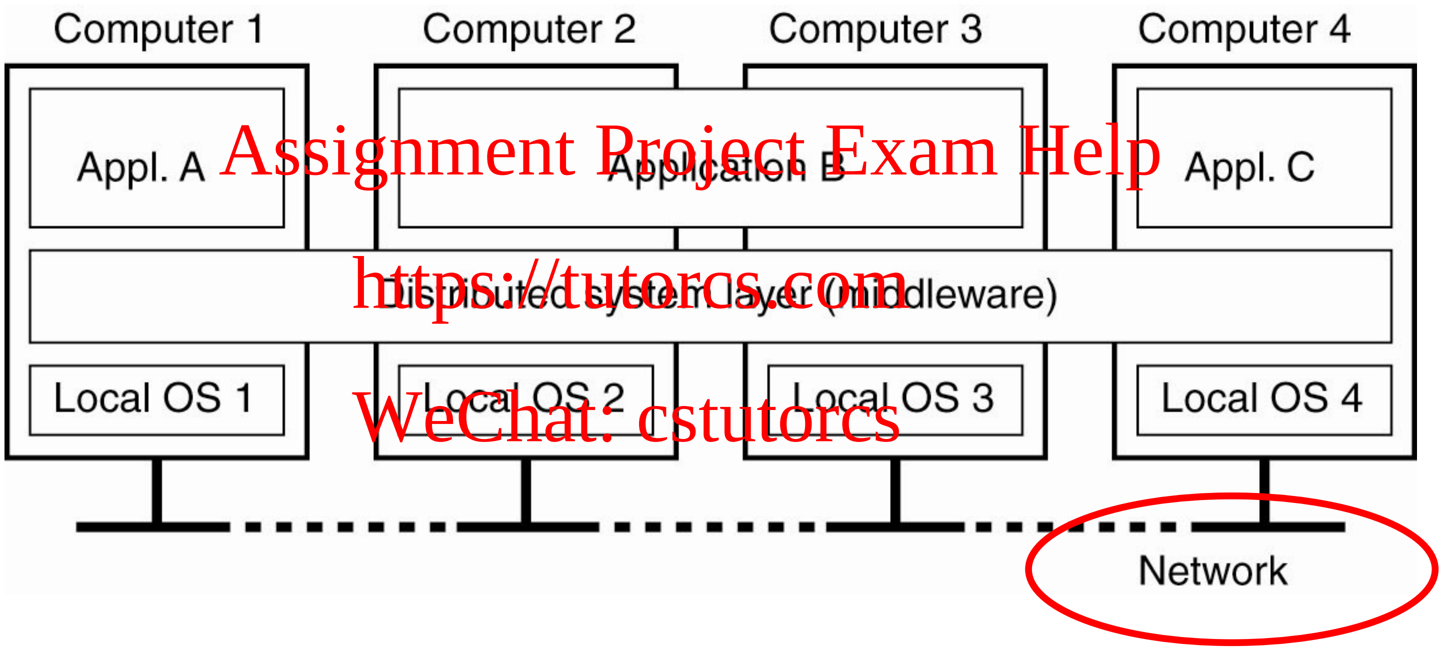
Why this is good?

- Easily expandable: adding new computers is hidden from users
- Availability: failure in one component can be covered by other components

So, how does it look like?

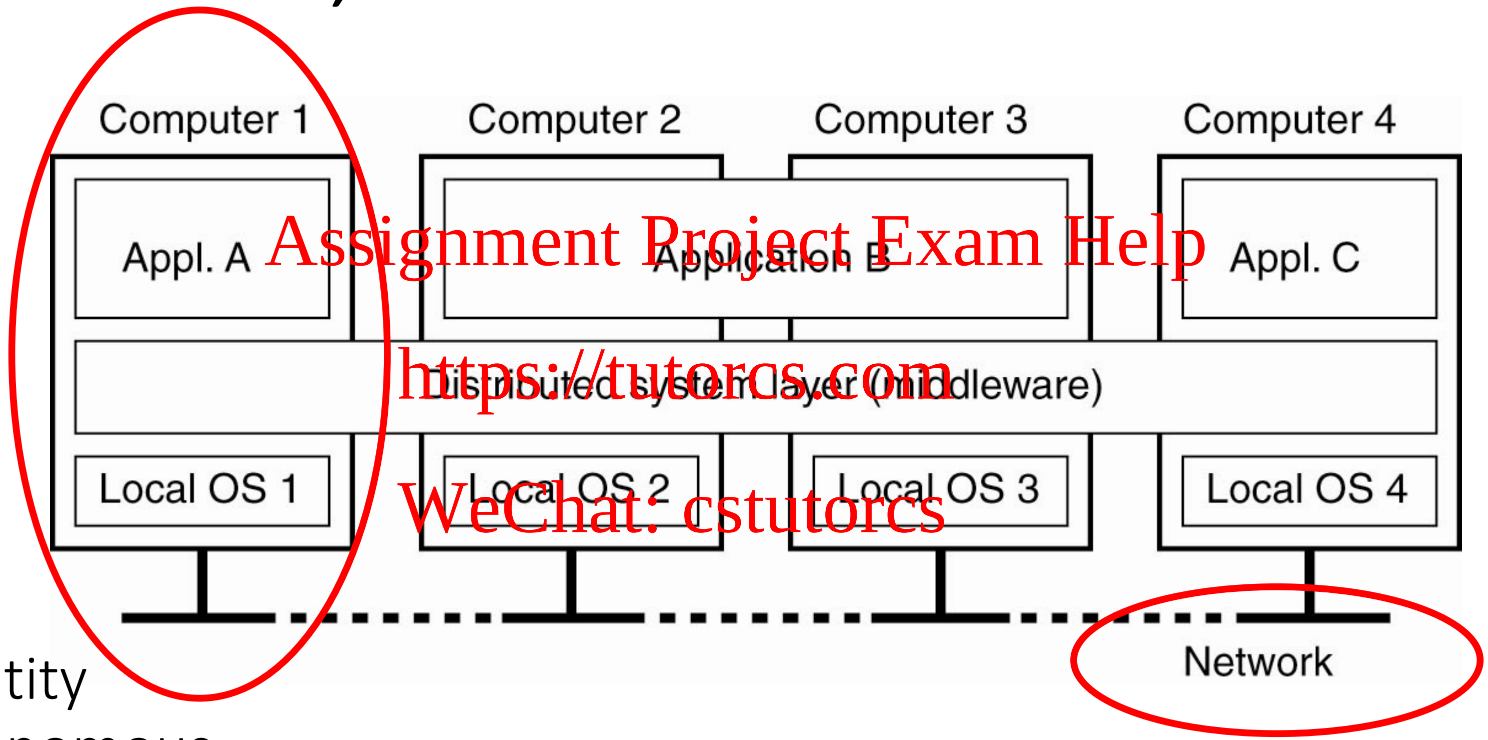


So, how does it look like?



This is the communication channel

So, how does it look like?

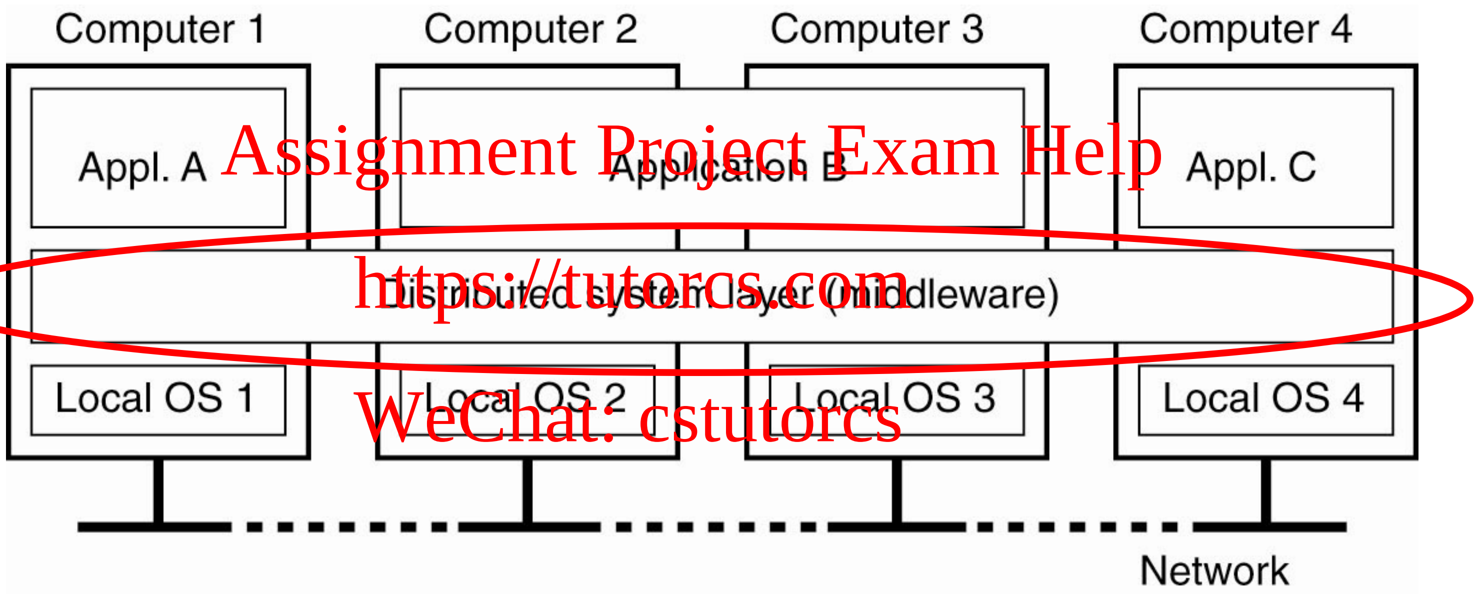


This is the entity which is autonomous, programmable and failure prone

This is the communication channel

So, how does it look like?

What about this?



The middleware

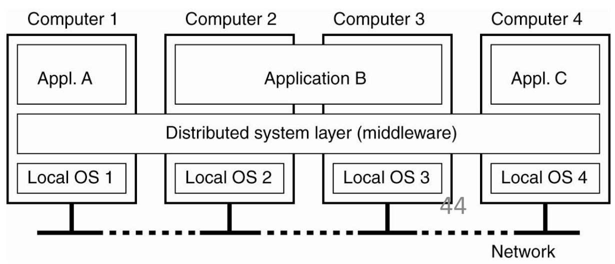
The middleware is a software layer situated between applications and operating systems. Allows independent computer to work together closely

- Hides the intricacies of distributed applications
- Hides the heterogeneity of hardware, operating systems and protocols
- Provides uniform and high-level interfaces used to make interoperable, reusable and portable applications
- Provides a set of common services that minimizes duplication of efforts and enhances collaboration between applications

The middleware (cont'd)

Middleware is similar to an operating system because it can support other application programs, provide controlled interaction, prevent interference between computations and facilitate interaction between computations on different computers via network communication services.

A typical operating system provides an application programming interface (API) for programs to utilize underlying hardware features. Middleware, however, provides an API for utilizing underlying operating system features.



The middleware: examples

- CORBA (Common Object Request Broker Architecture)

- DCOM (Distributed Component Object Management) – being replaced by .net

- Sun's ONC RPC (Remote Procedure Call)

- RMI (Remote Method Invocation)
- SOAP (Simple Object Access Protocol)

The middleware: examples

- All of the previous examples support communication across a network

- They provide protocols that allow a program running on one kind of computer, using one kind of operating system, to call a program running on another computer with a different operating system
  - The communicating programs must be running the *same* middleware

Recap

- **What:** A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate though an unreliable communication medium

- **Who:** AWS, Azure, Google cloud

- **How:** Middleware

Outline

Today, the lecture will focus on three main points:

- Definition of a Distributed System

- Goals of a Distributed System

- Types of Distributed Systems

The goals

- Resource Accessibility

- Transparency
- Openness
- Scalability

The goals

- Resource Accessibility

- Transparency
- Openness
- Scalability

Resource accessibility

- Support user access to remote resources (printers, data files, web pages, CPU cycles) and the fair sharing of the resources
- Economics of sharing expensive resources
- Performance enhancement – due to multiple processors
- Resource sharing introduces security problems.

The goals

- Resource Accessibility

- Transparency

- Openness

- Scalability

Transparency

- A distributed system that appears to its users & applications to be a single computer system is said to be *transparent*.

- Users & apps should be able to access remote resources in the same way they access local resources.

- Software hides some of the details of the distribution of system resources.

- Transparency has several dimensions.

Transparency

- A distributed system that appears to its users & applications to be a single computer system is said to be *transparent*.

- Users & apps should be able to access remote resources in the same way they access local resources.

- Software hides some of the details of the distribution of system resources.

- Transparency has several dimensions.

Dimension 1: distribution

Transparency	Description
Access	Hide differences in data representation & resource access (enables interoperability)
Location	Hide location of resource (can use resource without knowing its location)
Migration	Hide possibility that a system may change location of resource (no effect on access)
Replication	Hide the possibility that multiple copies of the resource exist (for reliability and/or availability)
Concurrency	Hide the possibility that the resource may be shared concurrently
Failure	Hide failure and recovery of the resource. How does one differentiate betw. slow and failed?
Relocation	Hide that resource may be moved <u>during use</u>

Dimension 2: degree

- Too much emphasis on transparency may prevent the user from understanding system behavior.

The goals

- Resource Accessibility

- Transparency

- Openness

- Scalability

Openness

- An <u>open</u> distributed system is one that is able to interact with other open distributed systems even if the underlying environments are different. This is accomplished:
  - Well defined interfaces
  - Should be able to support application portability
  - Systems should be able to interoperate

Why being “open” is good?

- **Interoperability**: the ability of two different systems or applications to work together
  - A process that needs a service should be able to talk to any process that provides the service.
  - Multiple implementations of the same service may be provided, as long as the interface is maintained

- **Portability**: an application designed to run on one distributed system can run on another system which implements the same interface.

- **Extensibility**: Easy to add new components, features

The goals

- Resource Accessibility
- Distribution Transparency
- Openness
- Scalability

Scalability

- Dimensions that may scale:
  - With respect to size
  - With respect to geographical distribution

- A scalable system still performs well as it scales up along any of the two dimensions

Scalability

- Dimensions that may scale:
  - With respect to size: This is clear, no need to say more about it.
  - With respect to geographical distribution

- A scalable system still performs well as it scales up along any of the two dimensions

Scalability

- Dimensions that may scale:
  - With respect to size
  - With respect to geographical distribution

- A scalable system still performs well as it scales up along any of the two dimensions

Geographic scalability

- A system that can handle an increase in workload that results from an increase in the size of the geographical area that it serves. The aim is to serve a larger geographical area just as easy as you can serve a smaller area.

Example 1: Netflix

- Think about **Netflix**! Netflix uses a Distributed Database Management Systems so that data can be stored locally in locations with the highest demand. This improves access time.

“Caching”

- Idea: Normally creates a (temporary) replica of something closer to the user

- Replication is often more permanent

- User (client system) decides to cache, server system decides to replicate

This is hard!

- Having multiple copies leads to inconsistencies: modifying one copy makes that copy different from the rest

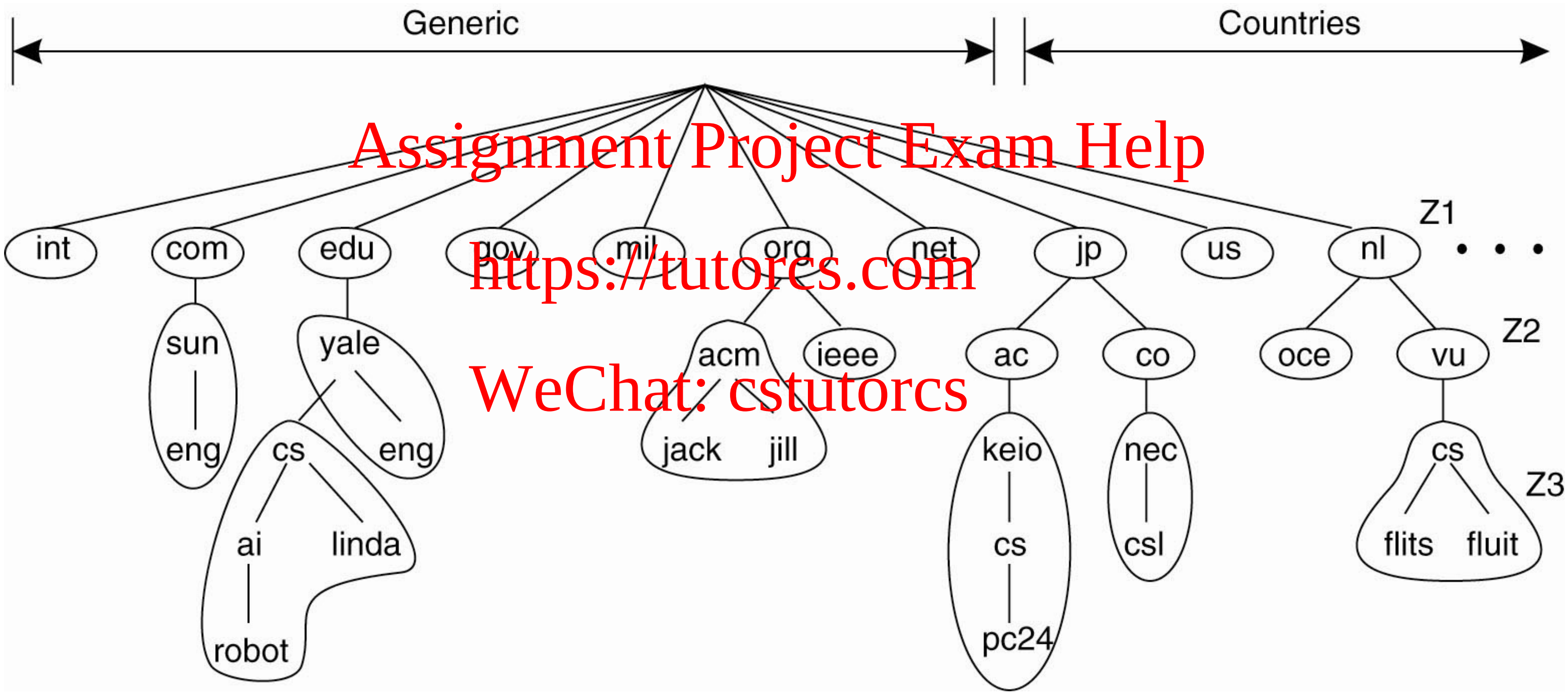
- Always keeping copies consistent and in a general way requires **global synchronization** on each modification

- Global synchronization **precludes** large-scale solutions

Example 2: DNS

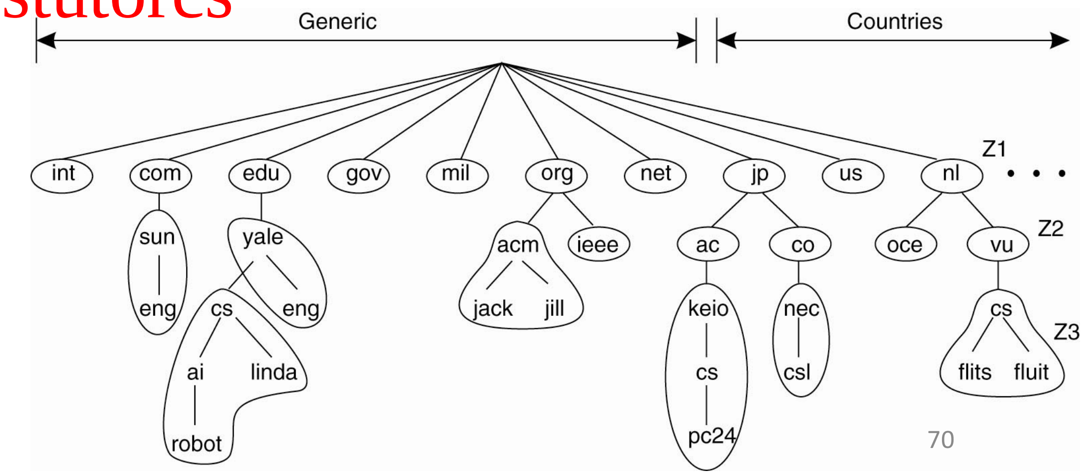
- DNS namespace is organized as a tree of domains; each domain is divided into zones; names in each zone are handled by a different name server
  - WWW consists of many (millions?) of servers

Example 2: DNS



Example 2: DNS

- Example: resolving flits.cs.vu.nl
  - first passed to the server of zone Z1 which returns the address of the server for zone Z2, to which the rest of name, flits.cs.vu, can be handed. The server for Z2 will return the address of the server for zone Z3, which is capable of handling the last part of the name and will return the address of the associated host.



What impact scalability?

- Scalability is negatively affected when the system is based on
  - Centralized server: one for all users
  - Centralized data: a single database for all users
  - Centralized algorithms: one site collects all information, processes it, distributes the results to all sites.
    - Complete knowledge: good
    - Time and network traffic: bad

Decentralization

- No machine has complete information about the system state

- Machines make decisions based only on local information

- Failure of a single machine doesn't ruin the algorithm

Decentralization is your friend

- A scalable distributed system must avoid centralising:

  - components (e.g., avoid having a single server)

  - tables (e.g., avoid having a single centralised directory of names)

  - algorithms (e.g., avoid algorithms based on complete information).

Decentralization is your friend

- When designing algorithms for distributed systems the following design rules can help avoid centralisation:

  - Do not require any machine to hold complete system state.
  - Allow nodes to make decisions based on local information.
  - Algorithms must survive failure of nodes.
  - No assumption of a global clock.

Summary

- **Resource accessibility:** sharing and enhanced performance
- **Transparency:** easier use
- **Openness:** support interoperability, portability, extensibility
- **Scalability**: with respect to size (number of users) and geographic distribution

Outline

Today, the lecture will focus on three main points:

- Definition of a Distributed System

- Goals of a Distributed System

- Types of Distributed Systems

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
  - Clouds

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
  - Clouds

Clusters

- A collection of similar processors (PCs, workstations) running the same operating system, connected by a high-speed LAN.

- Parallel computing capabilities using inexpensive PC hardware

- Example: High Performance Clusters (HPC)
  - CERN
  - run large parallel programs
  - Scientific, military, engineering apps; e.g., weather modeling

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
  - Clouds

Grids

- Grid computing is the use of widely distributed computer resources to reach a common goal.

- Similar to clusters but processors are more loosely coupled, tend to be heterogeneous (hardware, software, networks, security policies) and are not all in a central location.

- Can handle workloads similar to those on supercomputers, but grid computers connect over a network (Internet) and supercomputers' CPUs connect to a high-speed internal bus/network

Grids

Example:

- As of October 2016, over 4 million machines running the open-source Berkeley Open Infrastructure for Network Computing (BOINC) platform are members of the World Community Grid. One of the projects using BOINC is SETI@home, which was using more than 400,000 computers to achieve 0.828 TFLOPS as of October 2016. As of October 2016 Folding@home, which is not part of BOINC, achieved more than 101 x86-equivalent petaflops on over 110,000 machines

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
  - Clouds

Cloud Computing

- **Grid computing** and **cloud computing** are conceptually similar that can be easily confused. The concepts are quite similar, and both share the same vision of providing services to the users through sharing resources among a large pool of users.

- Cloud computing is a type of internet-based computing where an application doesn't access the resources directly, rather it makes a huge resource pool through shared resources. It is modern computing paradigm based on network technology that is specially designed for remotely provisioning scalable and measured IT resources.

Cloud Computing vs Grid

	Grid	Cloud
Underlying concept	Utility Computing	Utility Computing
Main benefit	Solve computationally complex problems	Provide a scalable standard environment for network-centric application development, testing and deployment
Resource distribution / allocation	Negotiate and manage resource sharing; schedulers	Simple user <-> provider model; pay-per-use
Domains	Multiple domains	Single domain
Character / history	Non-commercial, publicly funded	Commercial

Cloud Computing

- Examples:
  - Amazon Web Services (AWS)
  - Google cloud compute engine

- The course on "Cloud Computing" will extensively cover all the related aspects

Rule of thumb

Finally, some rules of thumb that are relevant to the study and design of distributed systems

- <u>Trade-offs</u>: many of the challenges faced by distributed systems lead to conflicting requirements (well this is valid for everything I would say)
  - Scalability vs performance
  - Flexibility vs reliability

Rule of thumb

Finally, some rules of thumb that are relevant to the study and design of distributed systems

- **Separation of Concerns**: When tackling a large, complex, problem, it is useful to split the problem up into separate concerns and address each concern individually (leads to highly modular or layered systems, which helps to increase a system's flexibility).
  - Communication vs replication vs consistency

Rule of thumb

Finally, some rules of thumb that are relevant to the study and design of distributed systems

- **End-to-End Argument**: aka, where to implement a given functionality? (Implementing it at the wrong level not only forces everyone to use that, but may render it less useful than if it was implemented at a higher level)
  - Application level vs lower layer in the system

Rule of thumb

Finally, some rules of thumb that are relevant to the study and design of distributed systems

- **Keep It Simple**: Overly complex systems are error prone and difficult to use. If possible, solutions to problems and resulting architectures should be simple rather than mind-numbingly complex