Part 2

Software Engineering (2)

Final Revision

2022

تنبيهات هاامة جداااا

١. يحظر ويمنع منعا باتاا الظهور بهذه المذكرة داخل الجامعة.

٢. يحظر ويمنع منعا باتاا نسخ او نقل او تصوير أي جزء من هذه المذكرة حتى لا تعرض نفسك للمسائلة القانونية .

Chapter 3
Distributed Software Engineering

Define the following Software Reuse Concepts:

1) **Distributed System** is a group of computer systems working together as to appear as a single computer to the end-user. System components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system.
2) **Resource Sharing:** refers to sharing of hardware and software resources—such as disks, printers, files, and compilers—that are associated with computers on a network.
3) **Open Systems**: refer to systems designed around standard Internet protocols so that equipment and software from different vendors can be combined.
4) **Concurrency**: is a distributed system property refers to operating several processes at the same time on separate computers on the network.
5) **Scalability:** is a distributed systems property that refers increasing the capabilities of the system by adding new resources to cope with new demands on the system.
6) **Fault tolerance :** is a distributed system property that refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail.
7) **Transparency**: is a distributed system property that refers to what extent should the distributed system appear to the user as a single system?
8) **Quality of service (QoS):** is a property offered by a distributed system reflects the system's ability to deliver its services dependably and with a response time and throughput that are acceptable to its users.
9) **Procedural interaction.** is a models interaction technique where one computer calls on a known service offered by another computer and waits for a response.
10) **Message based Interaction:** is a models interaction technique where one computer send information about what is required to another computer. There is no necessary to wait for response.
11) **Distributed system Middleware**: is a software system that can manage the diverse components of distributed systems (e.g. different programing languages, different types of processors, different networks, etc.) and ensure that they can communicate and exchange data easily.
12) **Thin client:** is a simple computer that has been optimized for establishing a remote connection with a server-based computing environment. The presentation layer is implemented on the client and all other layers (data management, application processing, and database) are implemented on a server.
13) **Fat client:** is a complex computer that possesses considerable resources (e.g., memory, hard drive storage, and processing power) and functionality independent of a server. In fat client, some or all o the application processing is carried out on the client. Data management and database functions are implemented on the server. Automated Teller Machine (ATM) system is a good example of Fa client architecture.
14) **Architectural Pattern:** is a distributed systems architectural models include Master-Slav Architecture, Two-Tier Client Server Architecture, Multi-Tier Client Server Architecture, Distribute Components Architecture, and peer-to-peer architecture.
15) **Client-Server Computing:** is a distributed system architectural model, where an application modeled as a set of services that are provided by servers. Clients may access these services a present results to end-users.
16) **Master-Slave Architecture**: are commonly used in real time systems where there may be separa processors associated with data acquisition from the system's environment, data processing, a computation and actuator management. The 'master' process is usually responsible for computati coordination and communications and it controls the 'slave' processes which dedicated to the acq ion of data from an array of sensors.

17) **Two-Tier Client Server Architecture**: In two-tier client-server architecture, the system is implemented as a single logical server plus an indefinite number of clients that use that server.

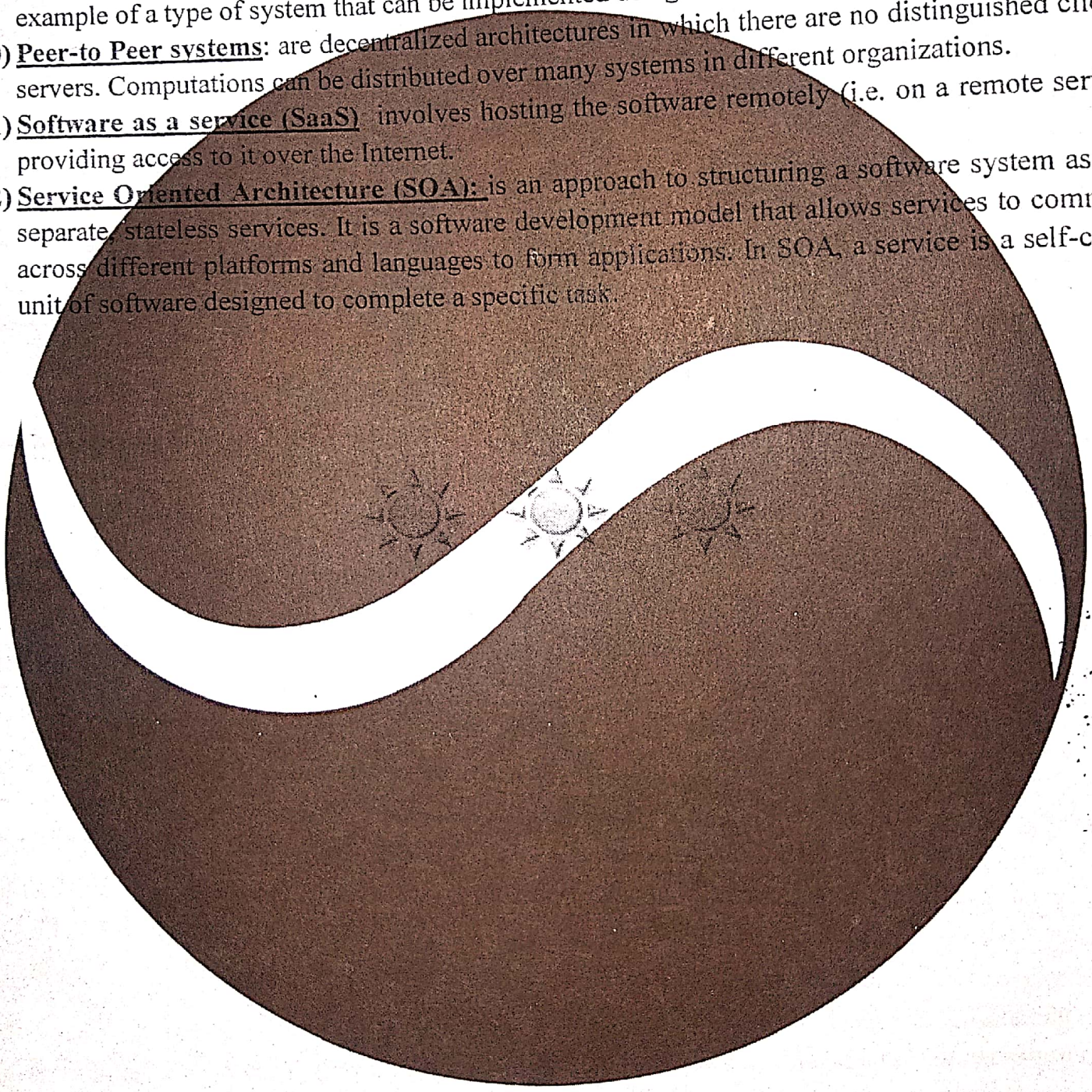
18) **Multi-Tier Client Server Architecture**: In a 'multi-tier client–server' architecture, the different layers of the system, namely presentation, data management, application processing, and database, are separate processes that may execute on different processors.

19) **A Distributed Component Architecture**: is a general approach of distributed systems that refers to design systems as a set of services, without attempting to allocate these services to layers in the system. The services are called and managed through a middleware. Data-mining systems are a good example of a type of system that can be implemented using a distributed component architecture

20) **Peer-to Peer systems**: are decentralized architectures in which there are no distinguished clients and servers. Computations can be distributed over many systems in different organizations.

21) **Software as a service (SaaS)** involves hosting the software remotely (i.e. on a remote server) and providing access to it over the Internet.

22) **Service Oriented Architecture (SOA):** is an approach to structuring a software system as a set of separate, stateless services. It is a software development model that allows services to communicate across different platforms and languages to form applications. In SOA, a service is a self-contained unit of software designed to complete a specific task.



Answer The Following Questions:

1) What are the FIVE benefits/characteristics of developing systems as distributed systems?

Answer:

- ***Resource Sharing*** A distributed system allows the sharing of hardware and software resources—such as disks, printers, files, and compilers.
- ***Openness systems*** designed around standard Internet protocols so that equipment and software from different vendors can be combined.
- ***Concurrency*** In a distributed system, several processes may operate at the same time on separate computers on the network.
- ***Scalability*** In principle at least, distributed systems are scalable in that the capabilities of the system can be increased by adding new resources to cope with new demands on the system.
- ***Fault Tolerance*** The availability of several computers and the potential for replicating information means that distributed systems can be tolerant of some hardware and software failures

2) What are the most important design issues that have to be considered in distributed systems engineering?

Answer:

1. ***Transparency*** : mean that distributed system should appear to the user as a single system.
2. ***Openness*** mean that distributed system should be designed using standard protocols that support interoperability.
3. ***Scalability*** refers to how the distributed system can be constructed so that it is saleable.
4. ***Security*** refers to how can usable security policies be defined and implemented in distributed systems.
5. **Quality of service (QoS)** refers to how the system should be implemented to deliver an acceptable quality of service to all users.
6. **Failure management** refers to how can system failures be detected, contained, and repaired.

3) What are the most types of attacks that target distributed systems?

Answer:

The types of attack that a distributed system must defend itself against are:

- **Interception**, where communications between parts of the system are intercepted by an attacker so that there is a loss of confidentiality.
- **Interruption**, where system services are attacked and cannot be delivered as expected.
  - *Denial of service attacks* involve bombarding a node with illegitimate service requests so that it cannot deal with valid requests.
- **Modification**, where data or services in the system are changed by an attacker
- **Fabrication**, where an attacker generates information that should not exist and then uses this to gain some privileges.

4) What are the two types of interactions between components in distributed systems?

Answer:

- **Procedural interaction,** where one computer calls on a known service offered by another computer and waits for a response. This type of interaction is carried out using *Remote Procedure Call (RPC)*
- **Message-based interaction,** involves the sending computer send information about what is required to another computer. There is no necessity to wait for a response. This type of interaction is carried out using *Message Passing* technique

Remote Procedure Calls (RPC)	Message Passing
▪ Procedural communication in a distributed system is implemented using remote procedure calls (RPC).	▪ Message-based interaction normally involves one component creating a message that details the services required from another component.
▪ In a remote procedure call, one component calls another component as if it was a local procedure or method. *The middleware* in the system intercepts this call and passes it to a remote component.	▪ Through the system middleware, this is sent to the receiving component.
▪ This carries out the required computation and, via *the middleware*, returns the result to the calling component.	▪ The receiver parses the message, carries out the computations and creates a message for the sending component with the required results.
▪ A Problem with RPCs is that the caller and the callee need to be available at the time of the communication, and they must know how to refer to each other	▪ In a message-based approach, it is not necessary for the sender and receiver of the message to be aware of each other. They simple communicate with the middleware

5) Explain and sketch a model to clarify the role of Middleware in Distributed system?

Answer:

- The components in a distributed system may be implemented in different programming languages and may execute on completely different types of processor. Models of data, information representation, and protocols for communication may all be different.

Middleware is software that can manage these diverse parts, and ensure that they can communicate and exchange data. This concept is illustrated in **Figure 1**, which shows that middleware is a layer between the operating system and application programs. Middleware can provide two types of supports:

- *Interaction support*, where the middleware coordinates interactions between different components in the system.
- *The provision of common services*, where the middleware provides reusable implementations of services that may be required by several components in the distributed system.

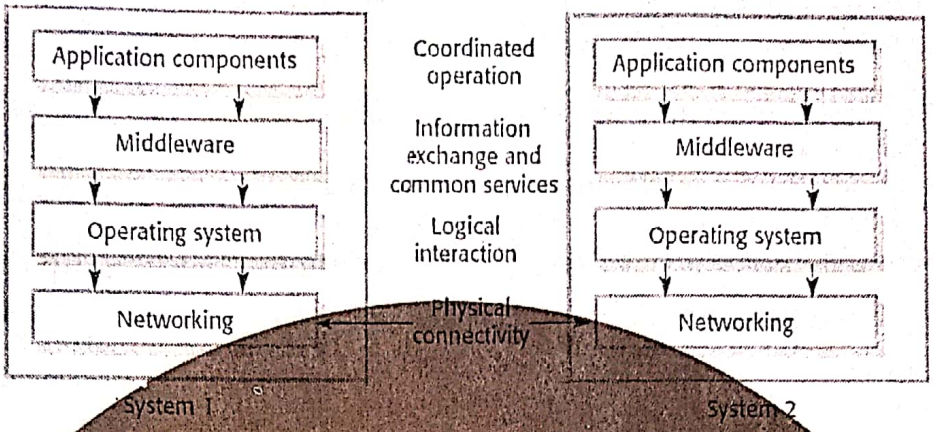


Figure 1 Middleware in a distributed system

6) Explain and sketch a model to clarify how *client-server system* is designed in a layer model?

Answer:

Client–server systems depend on there being a clear separation between the presentation of information and the computations that create and process that information. Consequently, you should design the architecture of distributed client–server systems so that they are structured into several logical layers, with clear interfaces between these layers. This allows each layer to be distributed to a different computer. **Figure 2** illustrates this model, showing an application structured into four layers:

1. ***A Presentation layer*** that is concerned with presenting information to the user and managing all user interaction.
2. ***A Data-handling layer*** that manages the data that is passed to and from the client. This layer may implement checks on the data, generate web pages, and so on.
3. ***An Application-processing layer*** that is concerned with implementing the logic of the application and so providing the required functionality to end-users.
4. ***A Database layer*** that stores the data and provides transaction management and query services.

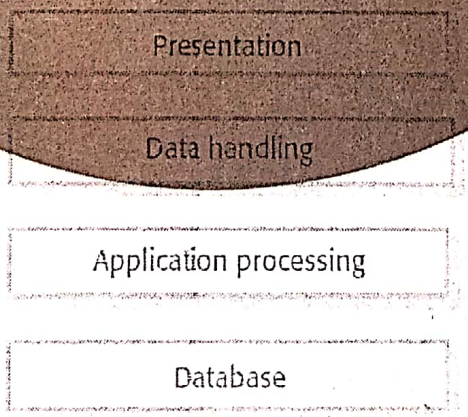


Figure 2 Layered architectural model for client–server application

7) What are the five architectural styles/patterns of distributed systems?

Answer:

1. ***Master-slave architecture***, which is used in real-time systems in which guaranteed interaction response times are required.
2. ***Two-tier client–server architecture***, which is used for simple client–server systems and in situations where it is important to centralize the system for security reasons.
3. ***Multi-tier client–server architecture***, which is used when the server has to process a high volume of transactions.
4. ***Distributed component architecture***, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client–server systems.
5. ***Peer-to-peer architecture***, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other. It may also be used when a large number of independent computations may have to be made.

8) Explain how a traffic control system in a city can be designed as *Master-Slave Architecture*?

Answer:

Master-slave architectures are commonly used in real-time systems where,

- the ***'Master' process*** is usually responsible for computation, coordination and communications and it controls the 'slave' processes.
- ***Slave' processes*** are dedicated to specific actions, such as the acquisition of data from an array of sensors.

✓ ***A traffic control system*** in a city is a good example of Master-Slave Architecture, which has three logical processes that run on separate processors. The master process is the control room process, which communicates with separate slave processes that are responsible for collecting traffic data and managing the operation of traffic lights as depicted in Figure 3.

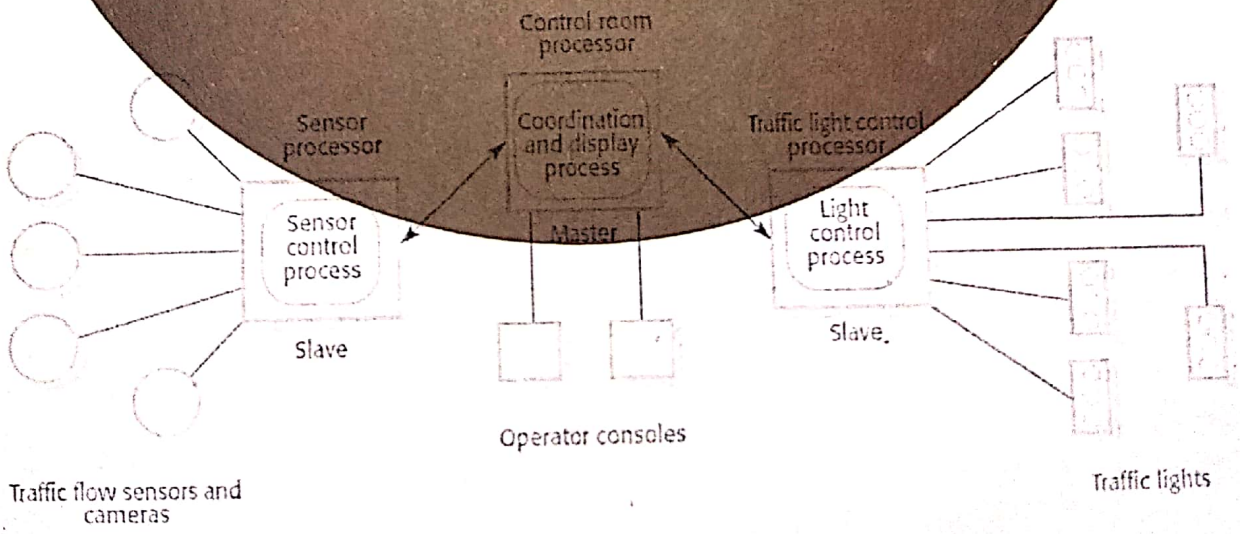


Figure 3 A traffic management systems with master–slave architecture

9) Sketch a model to clarify Thin- and fat-client architectural models?

Answer:

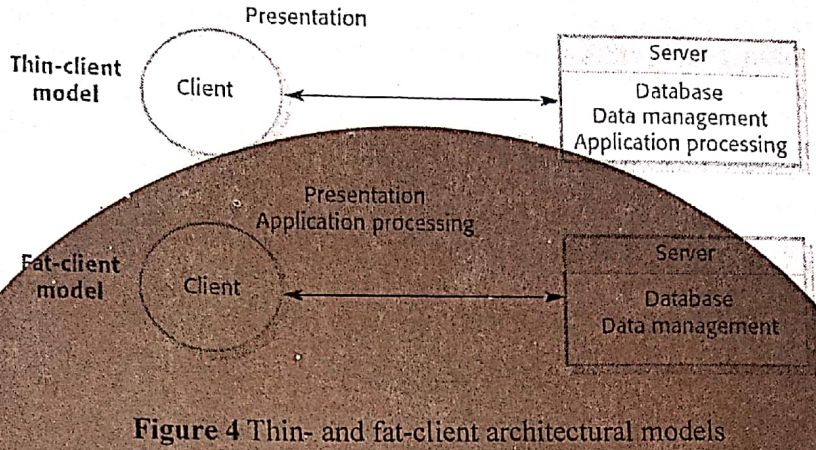


Figure 4 Thin- and fat-client architectural models

10) Explain using a model how *internet-banking system* can be designed as Three-tier processing architecture?

Answer:

✓ *Internet banking system* is a good example of three tier processing architecture, where, there are three tiers in the system as depicted in Figure 5

- *The bank's customer database* which usually hosted on a mainframe computer and provides database services.
- *A web server* provides data management services such as web page generation and some application services.
- *Application services* such as facilities to transfer cash, generate statements, pay bills, and so on are implemented in the web server and as scripts that are executed by the client. The user's own computer with an Internet browser is the client. This system is scalable because it is relatively easy to add servers (scale out) as the number of customers increase.

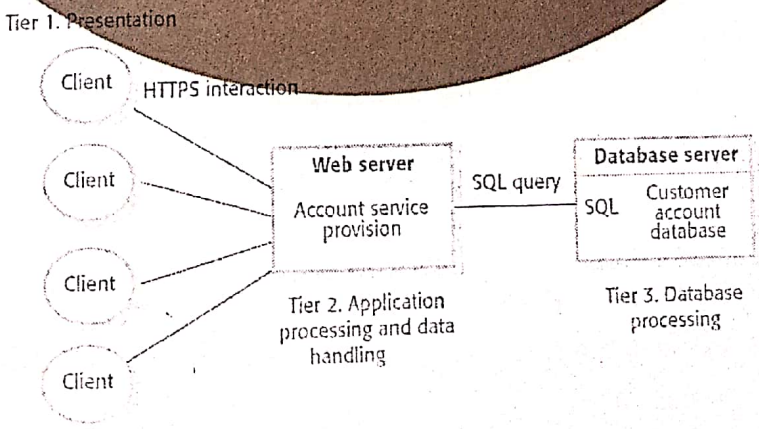


Figure 5: Three-tier architecture for an internet banking system.

11) What are benefits and challenges of Distributed component architectures?

Answer:

Benefits of distributed component architecture

- ✓ It allows the system designer to delay decisions on where and how services should be provided.
- ✓ It is a very open system architecture that allows new resources to be added as required.
- ✓ The system is flexible and scalable.
- ✓ It is possible to reconfigure the system dynamically with objects migrating across the network as required.

Challenges of distributed component architecture

- They are more complex to design than client–server systems. Distributed component architectures are difficult for people to visualize and understand.
- Standardized middleware for distributed component systems has never been accepted by the community. Different vendors, such as Microsoft and Sun, have developed different, incompatible middleware.

12) What are the common P2P Architectural models? Explain what is the meaning of semi centralized P2P architecture model

Answer:

- ✓ **The logical network architecture**
  - Decentralized architectures.
  - Semi-centralized architectures.
- ✓ **Application architecture:** The generic organization of components making up a p2p application.

- ***In a Semi-centralized architecture*** the role of the server (sometimes called a super peer) is to help establish contact between peers in the network or to coordinate the results of a computation. For example, if **Figure 6** represents an instant messaging system, then network nodes communicate with the server (indicated by dashed lines) to find out what other nodes are available. Once these nodes are discovered, direct communications can be established and the connection to the server becomes unnecessary. Therefore, nodes n2, n3, n5, and n6 are in direct communication.

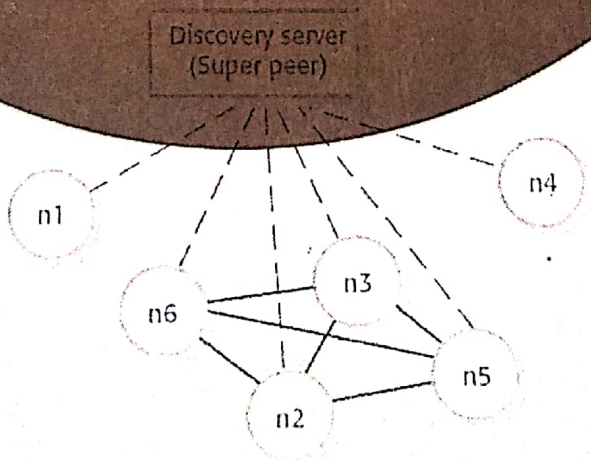


Figure 6. A semi-centralized p2p architecture.