CSE291 - Introduction To Software Engineering (Fall 2018)

Lecture 15

Software Aarchitecture

Architectural Design

Software Architecture

“The architecture of a system is a comprehensive framework that describes its form and structure – its components and how they fit together.”

Jerrold Grochow

Architectural Design

- Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system.
- It is the first stage in the software design process.
- Architectural design is a creative process where you design a system organization that will satisfy the functional and non-functional requirements of a system.

Architectural Design Decision

These requirements include performance, security, safety, availability, and maintainability. …

Performance: Performance can be enhanced by localising operations to minimise sub-system communication. That is, try to have self-contained modules as much as possible so that inter-module communication is minimized.

Security: Security can be improved by using a layered architecture with critical assets put in inner layers.

Architectural Attributes

Availability: building redundancy in the system and having redundant components in the architecture can ensure Availability.

Maintainability: Maintainability is directly related with simplicity. Therefore, using fine-grain, self-contained components can increase maintainability.

Architectural Design Process

This involves performing a number of activities, not necessarily in any particular order or sequence.

These include

- System structuring
- Control modeling
- Modular decomposition

Architectural Design Process

System structuring: System structuring is concerned with decomposing the system into interacting sub-systems.

Control modeling*:* Control modelling establishes a model of the control relationships between the different parts of the system.

Modular decomposition*:* During this activity, the identified sub-systems are decomposed into modules.

Architectural Styles

Among many styles, the most commonly practiced ones are the following:

- Data-Centered Architecture
- Layered Architecture
- Client Server Architecture

Layered Architecture

- A layered architecture organizes a system into a set of layers each of which provide a set of services to the layer “above”and serving as client to the layer below.
- Inner layers are closer to the machine hardware than the outer layers and each layer isolates the outer layer from inner complexities
- Supports incremental development

Layered Architecture

- The outer layer only needs to know the interface provided by the inner layer
- If there are any changes in the inner layer, the outer layer is not affected as long as the interface does not change
- The layered architecture pattern is another way of achieving separation and independence.

Interaction Between Layers

- Interactions among layers are defined by suitable communication protocols.

- Normally layers are ***constrained*** so elements only see:
  - other elements in the same layer, or
  - elements of the layer below

Interaction Between Layers

Flow

- **requests** from higher layer to lower layer
- **answers** from lower layer to higher layer (**callbacks)**
- incoming data or event notification from low to high

Advantages

Independence

- Different components of the application can be **independently** deployed, maintained, and updated, on different time schedules
- Makes possible for **team members to work in parallel** on different parts of the application with minimal **dependencies.**
- Testing the components **independently** of each other.

Advantages

More secure

- Each layer may hide private information from other layers

Reusability

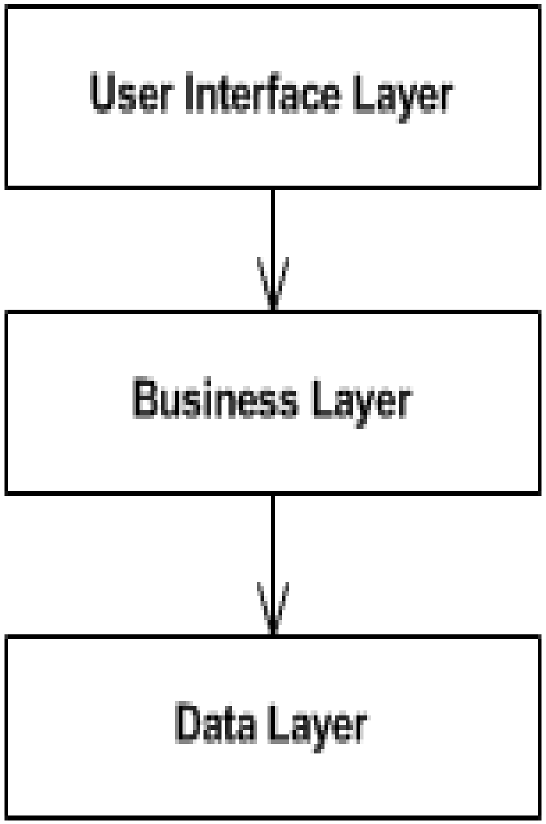
- Each layer, being cohesive and is coupled only to lower layers, makes it easier for reuse by others and easier to be replaced or interchanged

Disadvantages

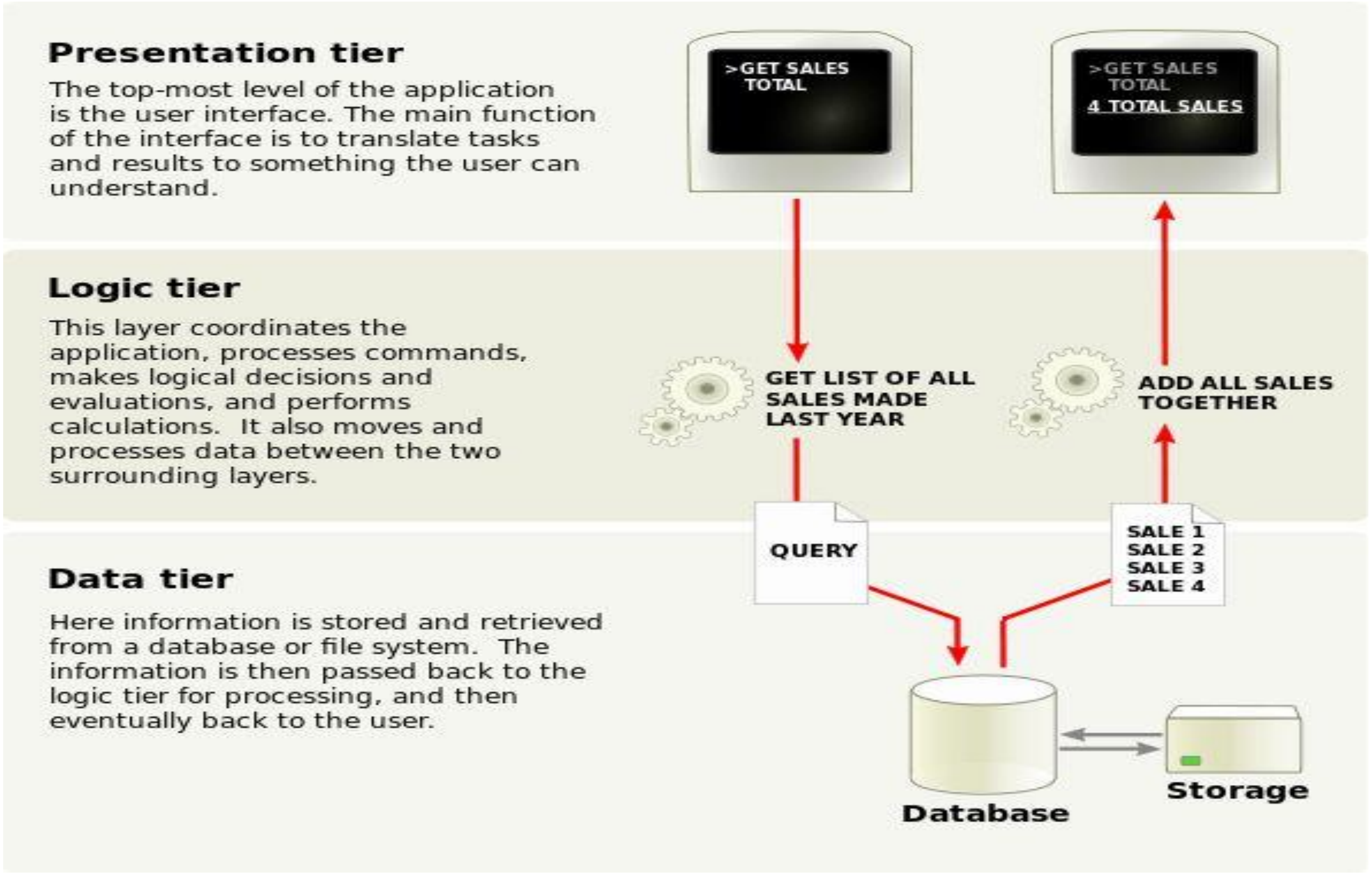
- Performance degrades if we have too many layers (extra overhead of passing through layers and also changes will pass slowly to higher layers)
- Sometimes difficult to cleanly assign functionality to the “right” layer
- Can’t be used for simple applications because it adds complexity.

Layered Architecture

- user interface layer responsible for displaying information and handling interaction is using services from the business layer who is responsible with performing actual system functions.
- The business layer is using services on its own from the data layer who is responsible for storing and retrieving data.



Layered Architecture



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

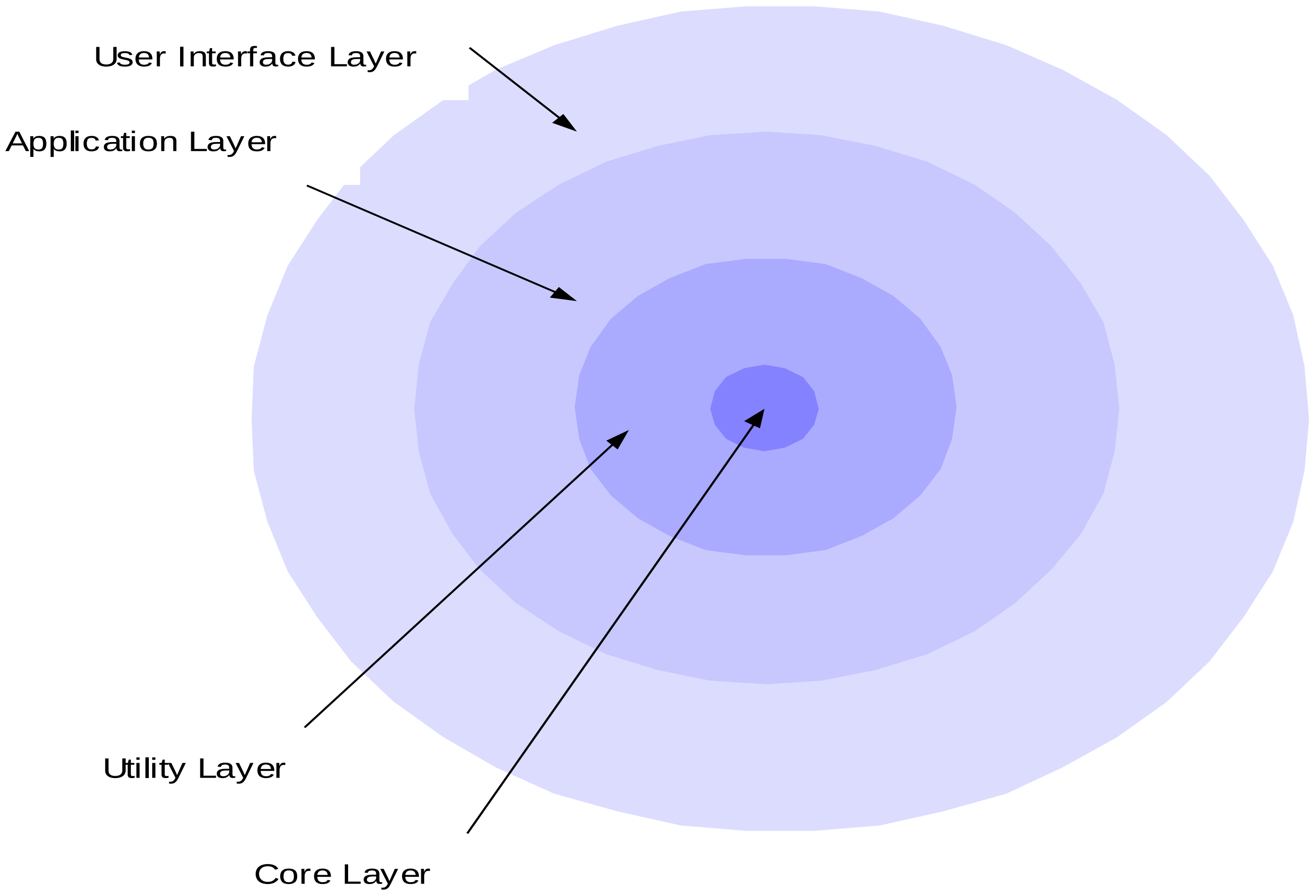
Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Layered Architecture



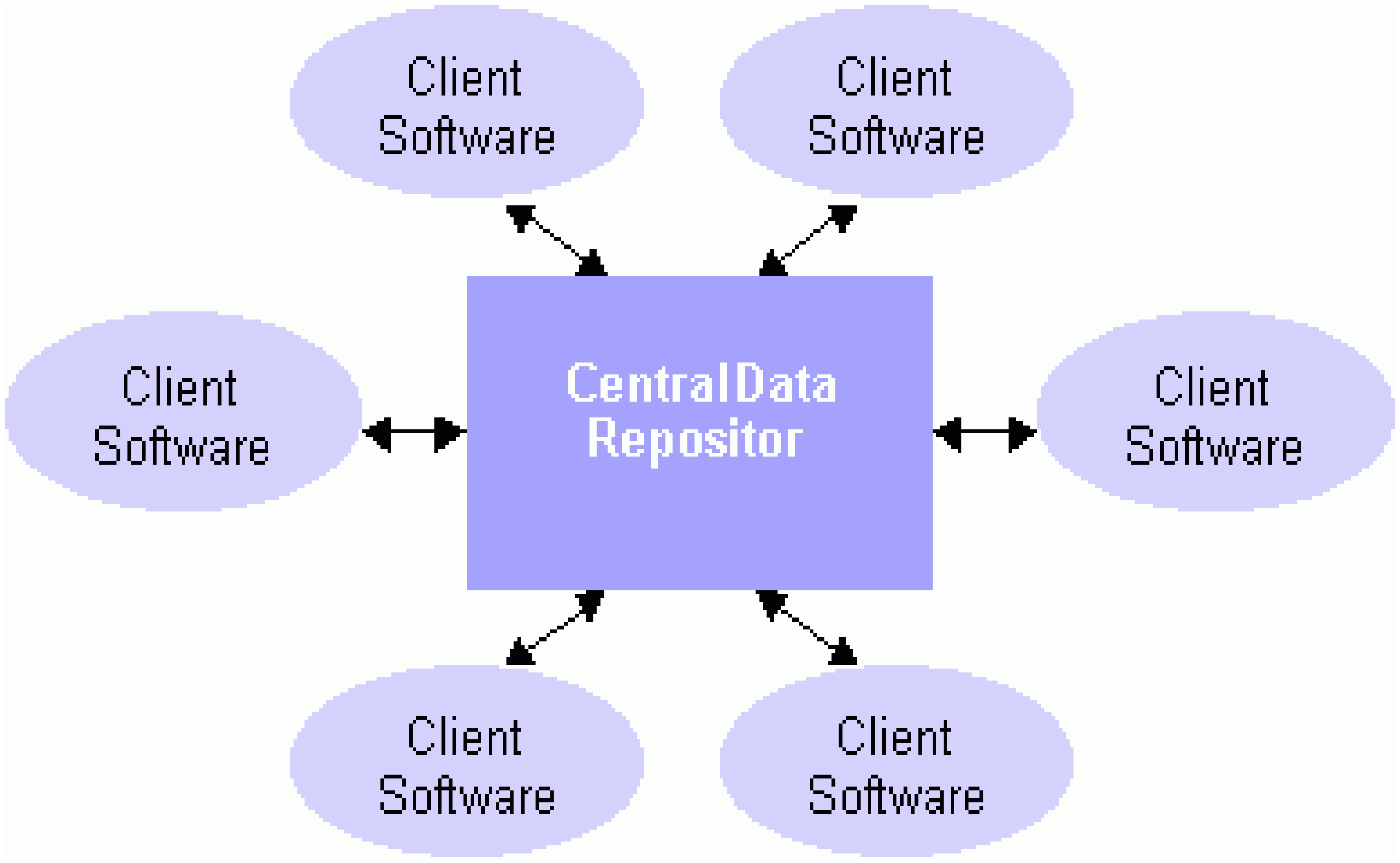
Data-Centered or Repository Architecture

In any system, sub-systems need to exchange information and data. This may be done in two ways:

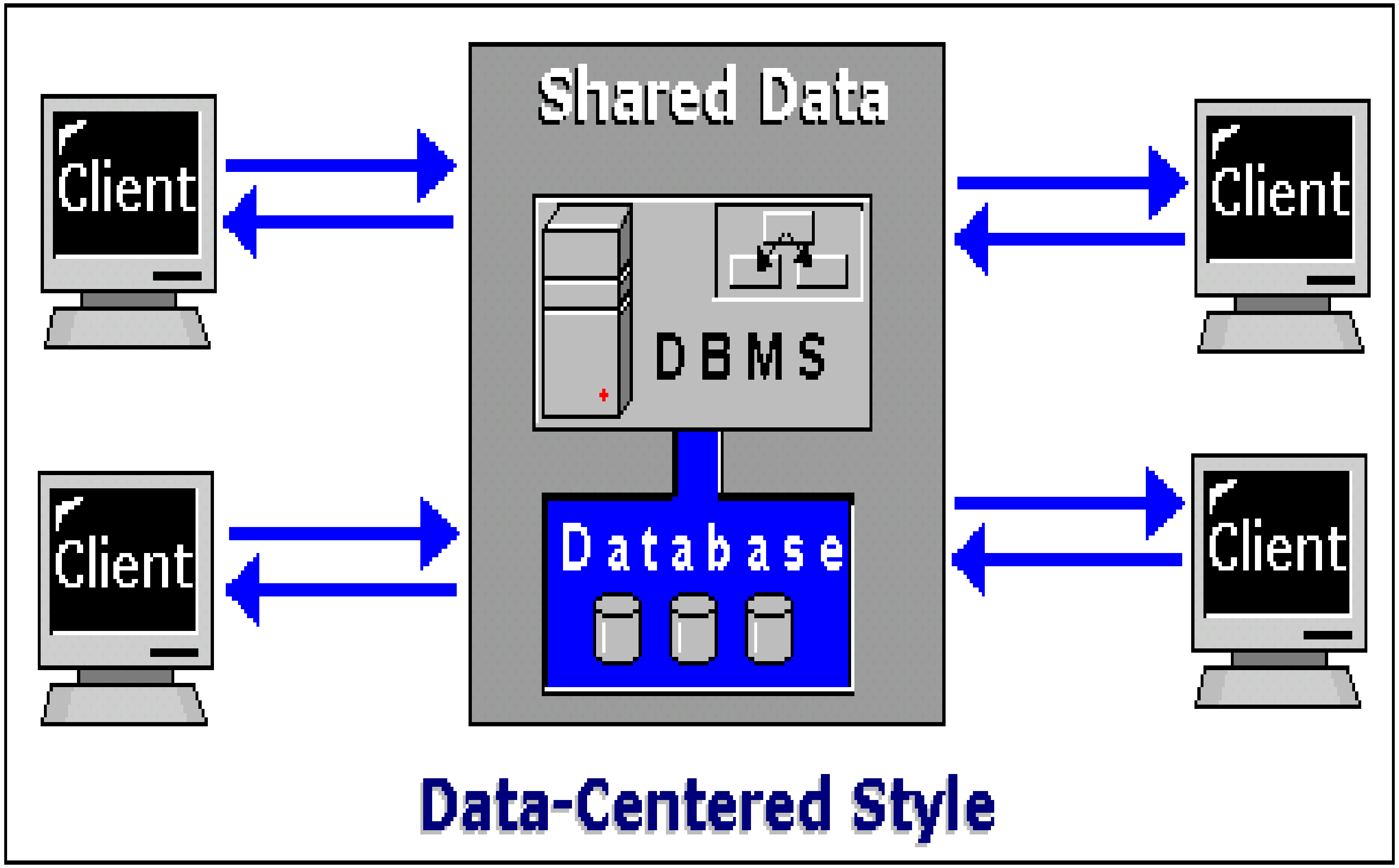
- Shared data is held in a central database or repository and may be accessed by all sub-systems
- Each sub-system maintains its own database and passes data to other sub-systems

- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

Data-Centered or Repository Architecture



Data-Centered or Repository Architecture



Data-Centered or Repository Architecture

- **Advantages**
  - An efficient way to share large amounts of data.
  - The sub-systems need not be concerned with how data is produced and the centralised management e.g. backup, security, etc.

- **Disadvantages**
  - Sub-systems must agree on a repository data model. This inevitably leads to a compromise.
  - The repository is a single point of failure so problems in the repository affect the whole system.

Client/Server Model

- The client-server model is a distributed system model that shows how data and processing is distributed across a range of components.
- The functionality of the system is organized into services, with each service delivered from a separate server.
- Clients are users of these services and access servers to make use of them.

In this model, the application is modeled as

1) A set of services that are provided by servers
2) A set of clients that use these services.

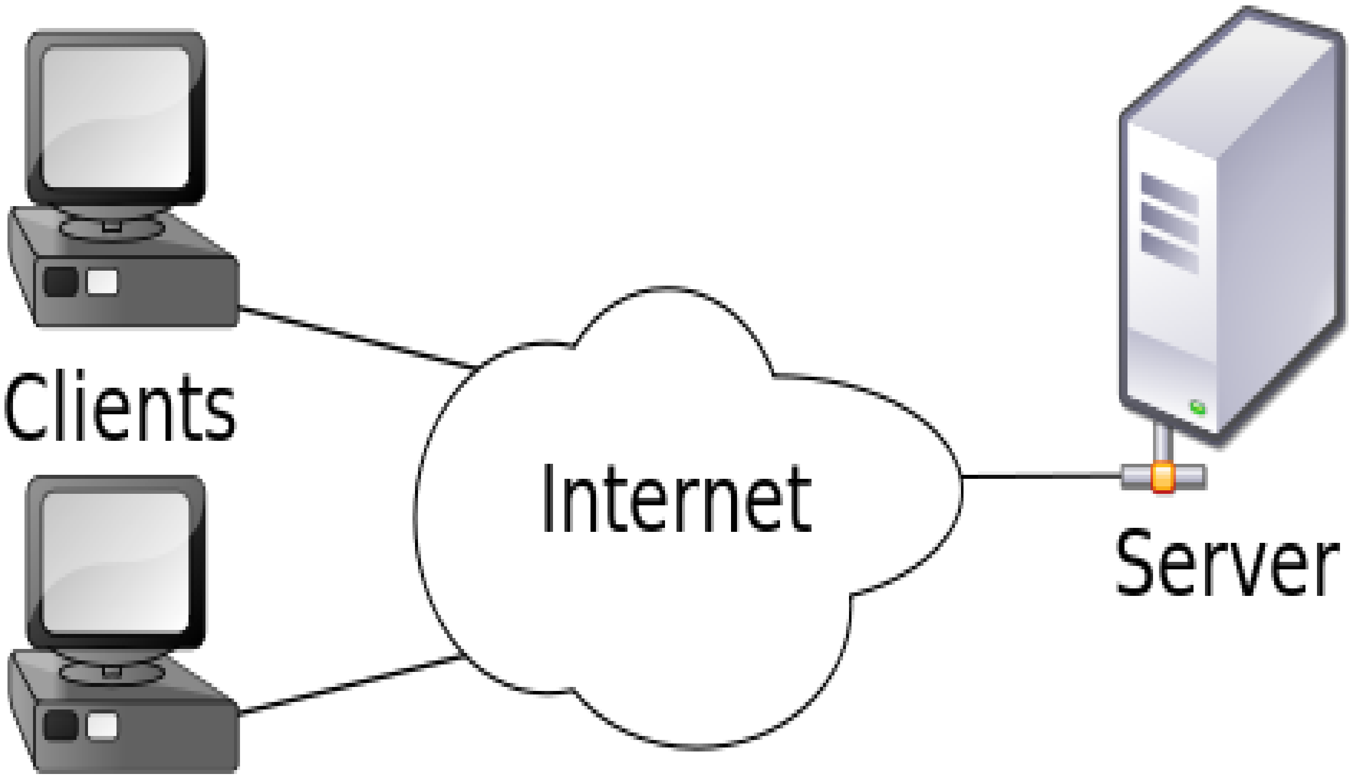
Client/Server Model

- The system is organized as a set of servers, which provide specific services such as printing, data management, etc. and a set of clients, which call on these services.

- These clients and servers are connected through a network which allows clients to access servers.

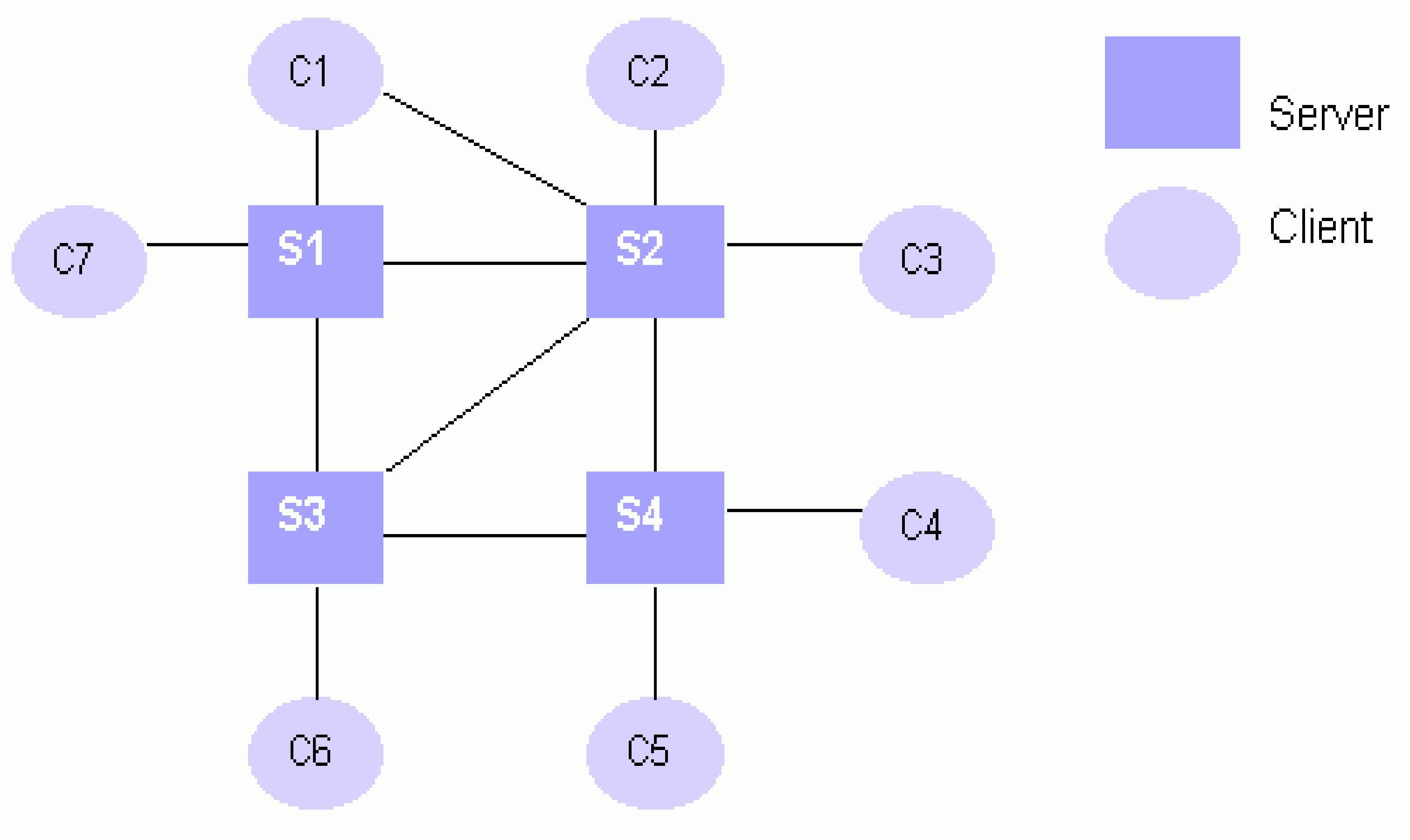
- Clients know the servers but the servers do not need to know all the clients.

Client/Server Model



Client/Server Model

General client-server organization



Client/Server Model

Following are some of the representative server types in client-server systems:

File servers: In this case, client requests selected records from a file and the server transmits records to the client over the network.

Database servers: In this case, the client sends requests, such as SQL queries, to the database server; the server processes the request and returns the results to the client over the network.

Client/Server Model

Transaction servers: In this configuration, client sends requests that invoke remote procedures on the server side; server executes procedures invoked and returns the results to the client.

Groupware servers: Groupware servers provide set of applications that enable communication among clients using text, images, bulletin boards, video, etc.

Client/Server Model

Advantages:

- Adding new servers or upgrading existing servers becomes easier.
- Distribution of data is straightforward in this case.

Disadvantages:

- Each service is a single point of failure so susceptible to denial of service attacks or server failure.
- Performance may be unpredictable because it depends on the network.