Tribhuvan University
Institute of Engineering
Pulchowk Campus
Department of Electronics and Computer Engineering

Software Engineering

Chapter *Three*

Architectural Design

by

Er. Bishwas Pokharel

Lecturer, Kathford Int’l College of Engg. and Mgmt.

Date: 24th jan, 2018

3. Architectural design (6 hours)

3.1. Architectural design decisions

3.2. System organization

3.3. Modular decomposition styles

3.4. Control styles

3.5. Reference architectures

3.6. Multiprocessor architecture

3.7. Client –server architectures

3.8. Distributed object architectures

3.9. Inter-organizational distributed computing

Definition:

Design is a problem-solving process whose objective is to find and describe a way:

- To implement the system’s *functional requirements*...
- While respecting the constraints imposed by the *non-functional requirements*...
  - –including the budget and deadlines

Design as a series of decisions:

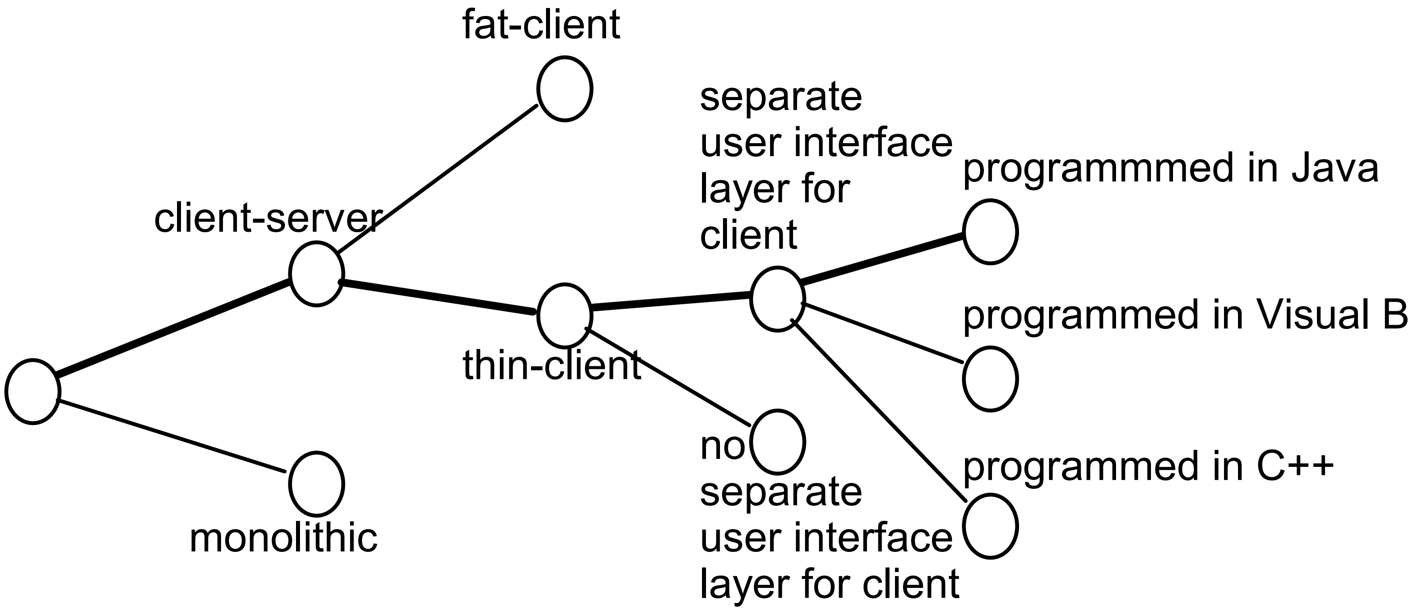
- A designer is faced with a series of *design issues*
  - These are sub-problems of the overall design problem.
  - Each issue normally has several alternative solutions:
    - design *options*.
  - The designer makes a *design decision* to resolve each issue.
    - This process involves choosing the best option from among the alternatives.

Making decisions:

- To make each design decision, the software engineer uses:
  - Knowledge of
    - the requirements
    - the design as created so far
    - the technology available
    - software design principles and ‘best practices’
    - what has worked well in the past

Design space:

- The space of possible designs that could be achieved by choosing different sets of alternatives is often called the *design space*
  - For example:



Component:

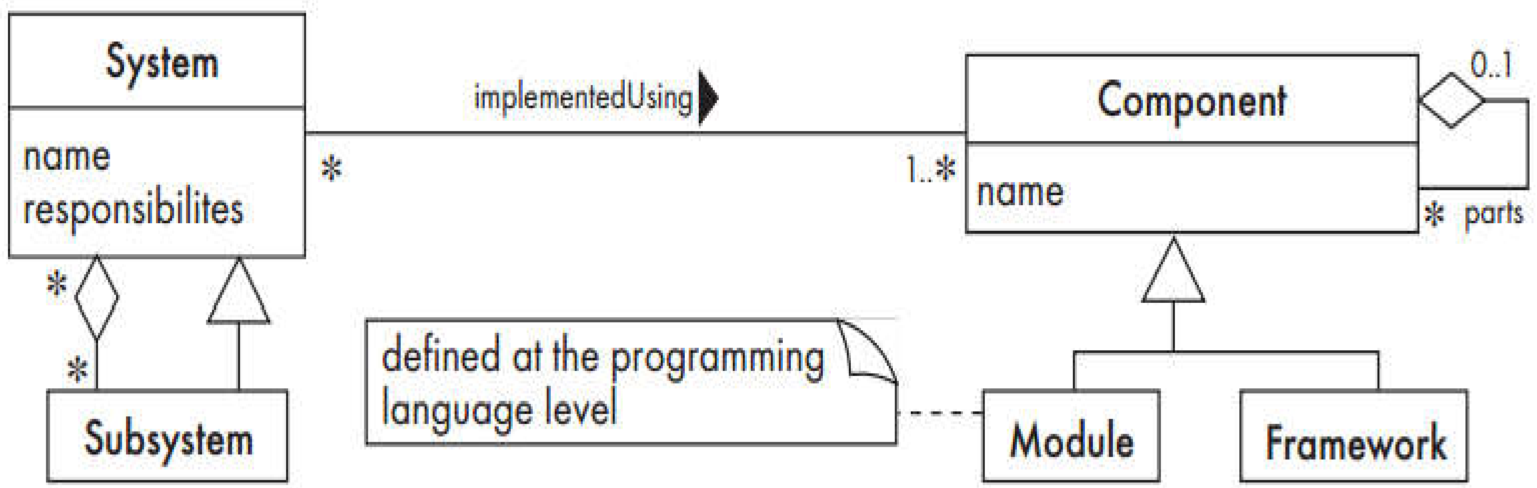
- Any piece of software or hardware that has a clear role.
  - A component can be isolated, allowing you to replace it with a different component that has equivalent functionality.
  - Many components are designed to be reusable.
  - Conversely, others perform special-purpose functions
  - Frameworks, components include source code files, executable files, dynamic link libraries (DLLs) and databases.

Module:

- A component that is defined at the programming language level
  - For example, methods, classes and packages are modules in Java.
  - The modules in the C programming language are files and functions

System

- A logical entity, having a set of definable responsibilities or objectives, and consisting of hardware, software or both.
  - A system can have a specification which is then implemented by a collection of components.
  - A system continues to exist, even if its components are changed or replaced.
  - The goal of requirements analysis is to determine the responsibilities of a system
  - **Subsystem**: A system that is part of a larger system, and which has a definite interface



Domain model explaining the concepts of system, subsystem, component and module as used in this book

Top-down and bottom-up design:

- **Top-down design**
  - First design the very high level structure of the system.
  - Then gradually work down to detailed decisions about low-level constructs.
  - Finally arrive at detailed decisions such as:
    - the format of particular data items;
    - the individual algorithms that will be used.

- **Bottom-up design**
  - Make decisions about reusable low-level utilities.
  - Then decide how these will be put together to create high-level constructs.
- **A mix of top-down and bottom-up approaches are normally used:**
  - Top-down design is almost always needed to give the system a good structure.
  - Bottom-up design is normally useful so that reusable components can be created.

Different aspects of design

- *Architecture design*:
  - The division into subsystems and components,
    - How these will be connected.
    - How they will interact.
    - Their interfaces.
- *Class design*:
  - The various features of classes.
- *User interface design*
- *Algorithm design*:
  - The design of computational mechanisms.
- *Protocol design*:
  - The design of communications protocol.

- **Overall goals of good design:**
  - Increasing profit by reducing cost and increasing revenue
  - Ensuring that we actually conform with the requirements
  - Accelerating development
  - Increasing qualities such as
    - Usability
    - Efficiency
    - Reliability
    - Maintainability
    - Reusability

Design Principle 1:Divide and conquer

Trying to deal with something big all at once is normally much harder than dealing with a series of smaller things.

-Separate people can work on each part.
-An individual software engineer can specialize.
-Each individual component is smaller, and therefore easier to understand.

-Parts can be replaced or changed without having to replace or extensively change other parts

Ways of dividing a software system

A distributed system is divided up into clients and servers.

A system is divided up into subsystems

A subsystem can be divided up into one or more packages.

A package is divided up into classes.

A class is divided up into methods

DesignPrinciple2: Increase cohesion where possible

- A subsystem or module has high cohesion if it keeps together things that are related to each other, and keeps out other things.
- This makes the system as a whole easier to understand and change.

Type of cohesion:

—Functional, Layer, Communicational, Sequential, Procedural,Temporal,Utility

Design Principle 3: Reduce coupling where possible

- Coupling occurs when there are interdependencies between one module and another.

- When interdependencies exist, changes in one place will require changes somewhere else.

Type of coupling:

—Content, Common, Control, Stamp, Data, Routine Call, Type use, Inclusion/Import, External

Design Principle 4: Keep the level of abstraction as high as possible

- Ensure that your designs allow you to hide or defer consideration of details, thus reducing complexity.

- A good abstraction is said to provide information hiding Abstractions allow you to understand the essence of a subsystem without having to know unnecessary details

Abstraction and classes

- Classes are data abstractions that contain procedural abstractions.
- Abstraction is increased by defining all variables as private.
- The fewer public methods in a class, the better the abstraction Super classes and interfaces increase the level of abstraction
- Methods are procedural abstractions
  —Better abstractions are achieved by giving methods fewer parameters

Design Principle 5: Increase reusability where Possible

- Design the various aspects of your system so that they can be used again in other contexts.

- Generalize your design as much as possible
- Follow the preceding three design principles.

- Design your system to contain hooks Simplify your design as much as possible

Design Principle 6: Design for flexibility

- Actively anticipate changes that a design may have to undergo in the future, and prepare for them.
- Reduce coupling and increase cohesion
- Create abstractions
- Do not hard-code anything
- Leave all options open
  —Do not restrict the options of people who have to modify the system later.
- Use reusable code and make code reusable

Design Principle 7: Design for Portability

- Have the software run on as many platforms as possible.
- Avoid the use of facilities that are specific to one particular environment
  E.g. a library only available in Microsoft Windows

Design Principle 8: Design for Testability

- Take steps to make testing easier
- Design a program to automatically test the software

Techniques for making good design decisions

Step 1: List and describe the alternatives for the design decision.

Step 2: List the advantages and disadvantages of each alternative with respect to your objectives and priorities.

Step 3: Determine whether any of the alternatives prevents you from meeting one or more of the objectives.

Step 4: Choose the alternative that helps you to best meet your objectives.

Step 5: Adjust priorities for decision making.

Example: Imagine a system has the following objectives, starting with top priority

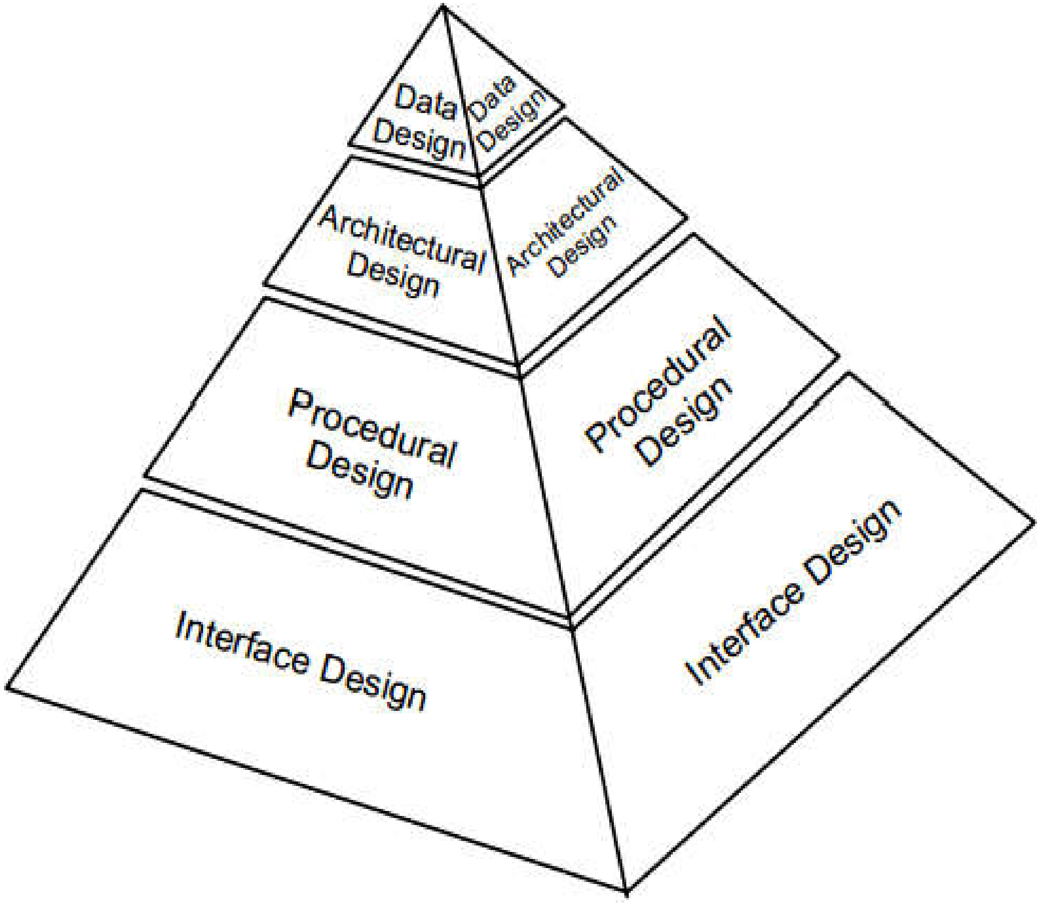
- **Security**: Encryption must not be breakable within 100 hours of computing time on a 400Mhz Intel processor, using known cryptanalysis techniques.
- **Maintainability**. No specific objective.
- **CPU efficiency**. Must respond to the user within one second when running on a 400MHz Intel processor.

- **Network bandwidth efficiency**: Must not require transmission of more than 8KB of data per transaction.

- **Memory efficiency**. Must not consume over 20MB of RAM.

- **Portability.** Must be able to run on Windows , as well as Linux

Design Model



Data Design

- Data design is developed by transforming entity relationship diagram (identified during the requirements phase) into data structures that are required to implement the software.
- The data design process includes identifying the data, defining specific data types and storage mechanisms.
- The structure of data can be viewed at three levels, namely, program component level, application level, and business level.

- At the **program component level**, the design of data structures and the algorithms required to manipulate them is necessary if a high-quality software is desired.
- At the **application level**, the translation of a data model into a database is essential to achieve the specified business objectives of a system.
- At the **business level**, the collection of information stored in different databases should be reorganized into data warehouse, which enables data mining that has influential impact on the business

Software Architecture(Architecture Design)

- *Software architecture* is process of designing the global organization of a software system, including:

  Dividing software into subsystems.
  Deciding how these will interact.
  Determining their interfaces.

- The architecture is the core of the design, so all software engineers need to understand it.
- The architecture will often constrain the overall efficiency, reusability and maintainability of the system.

The importance of software architecture

Why you need to develop an architectural model?

- To enable everyone to better understand the system.
- To allow people to work on individual pieces of the system in isolation.
- To prepare for extension of the system.
- To facilitate reuse and reusability

Architectural design decisions

- During architectural design process, system architects have to make many fundamental decisions

- Based on their knowledge & experience, have to answer the following fundamental questions:

  - Is there a generic application architecture that can be used?
  - How will the system be distributed?
  - What architectural styles are appropriate?

- What approach will be used to structure system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

Later, we will discuss about the architecture styles and patterns.

Procedural Design

- Procedural Design is also called **Component-Level-Design**
- A large program, like a pizza, needs to be cut into smaller pieces in order to be easily grasped. The pieces of a computer program are called modules and the act of cutting it up is called modularization.
- To achieve effective modularity, design concepts like functional independence are considered to be very important.
- Where Cohesion and Coupling is managed.

Interface Design

- The interface design elements for software depict information flows into and out of the system and how it is communicated among the components defined as part of the architecture.
- There are three elements of interface design:

(1) the user interface (UI);

(2) external interfaces to other systems, devices, networks, or other producers or consumers of information; and

(3) internal interfaces between various design components.

Interface Design

- The interface design elements for software depict information flows into and out of the system and how it is communicated among the components defined as part of the architecture.
- There are three elements of interface design:

(1) the user interface (UI);

(2) external interfaces to other systems, devices, networks, or other producers or consumers of information; and

(3) internal interfaces between various design components.

Architectural Design Process

Involves 3 activities:

1. System Structuring

   - Decomposition of main system into many sub system

2. Control modeling

   -To identify control relationships between parts of the system

3.Modular Decomposition

   -Decomposing the systems into modules

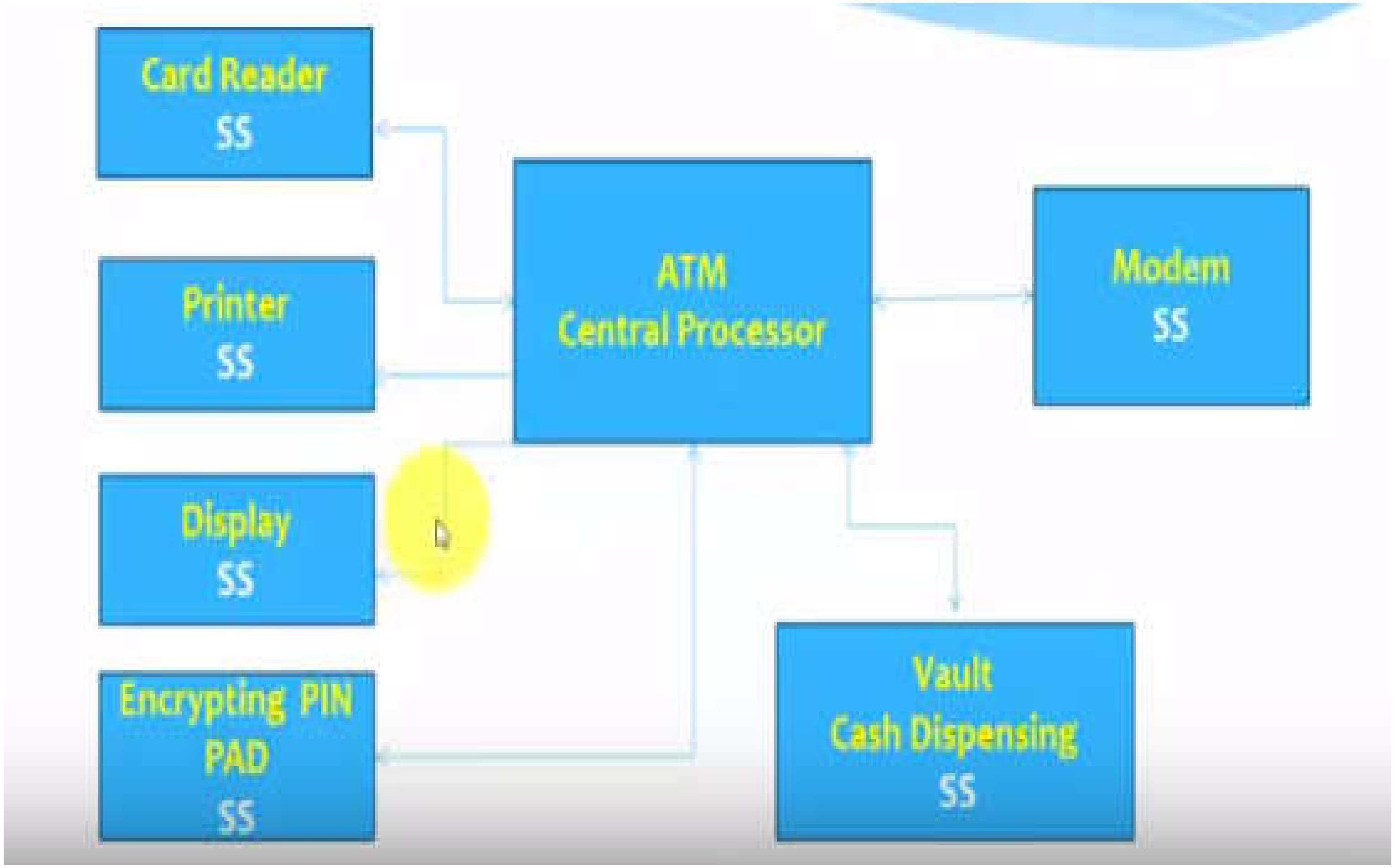
1. System Structuring

- Concerned with decomposing the system into interacting sub-systems.
- The architectural design is normally expressed as a block diagram presenting an overview of the system structure.
- More specific models showing how sub-systems

  share data,

  - are distributed and,
  - interface with each other.

1. System Structuring



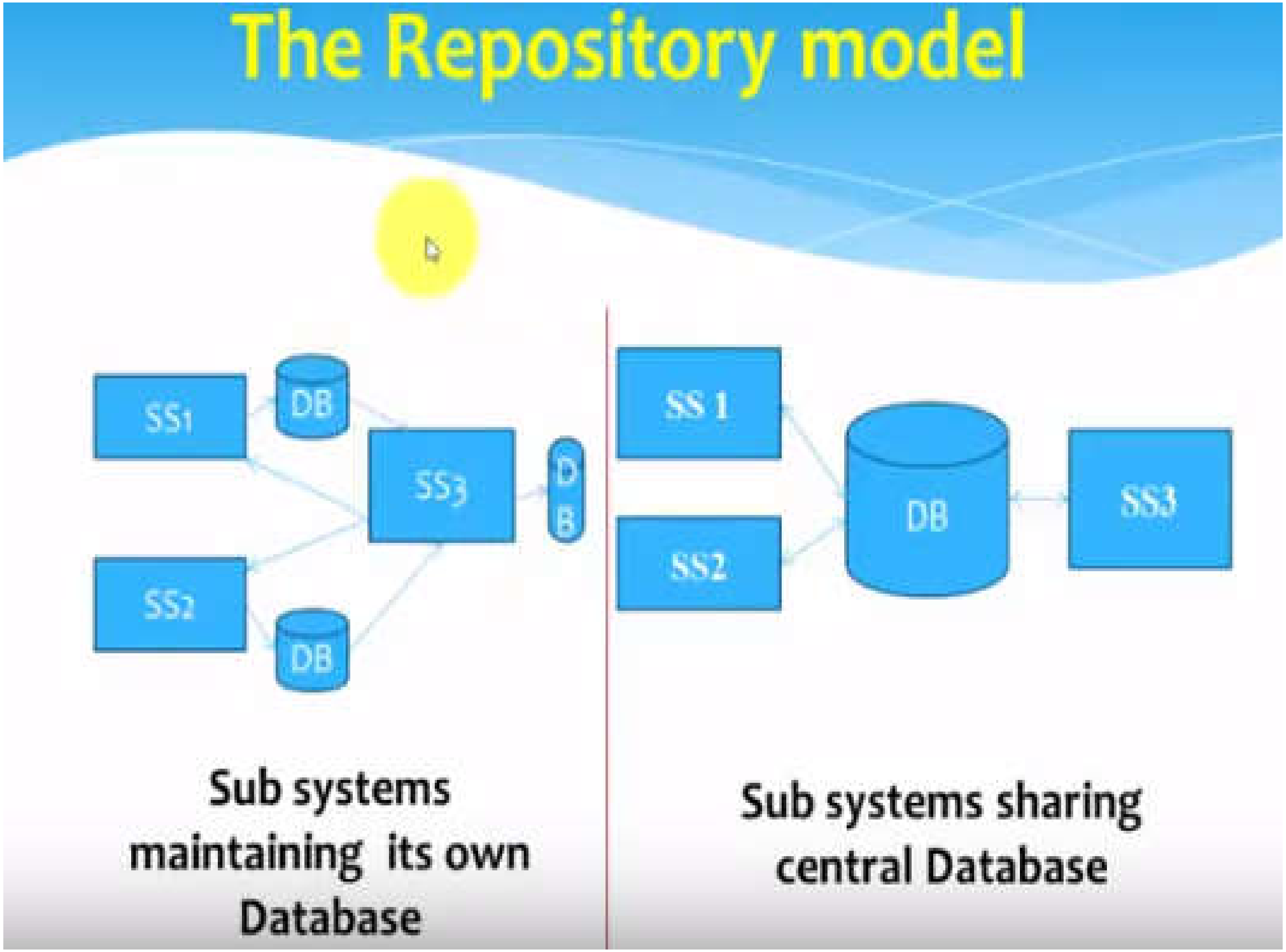
1. System Structuring

System Structuring involves many types,widely used 3 models are:

1. The repository model
2. Client-Server model
3. The layered model

1. Repository Model

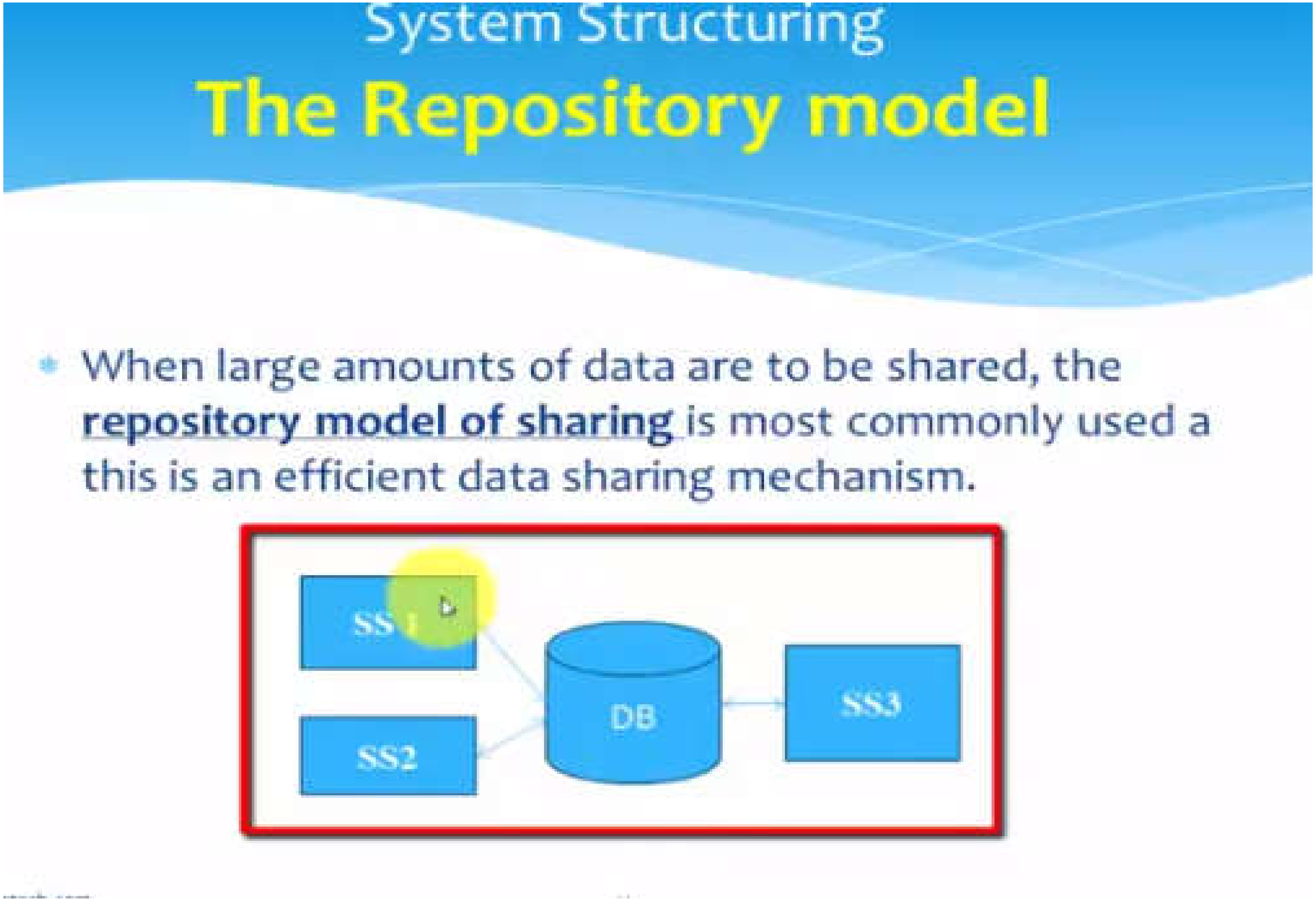
- Sub-systems making up a system must exchange information so that they work together effectively.
- This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.
- Suited for applications where data is generated by one sub-system and used by another.



The Repository model

Sub systems maintaining its own Database

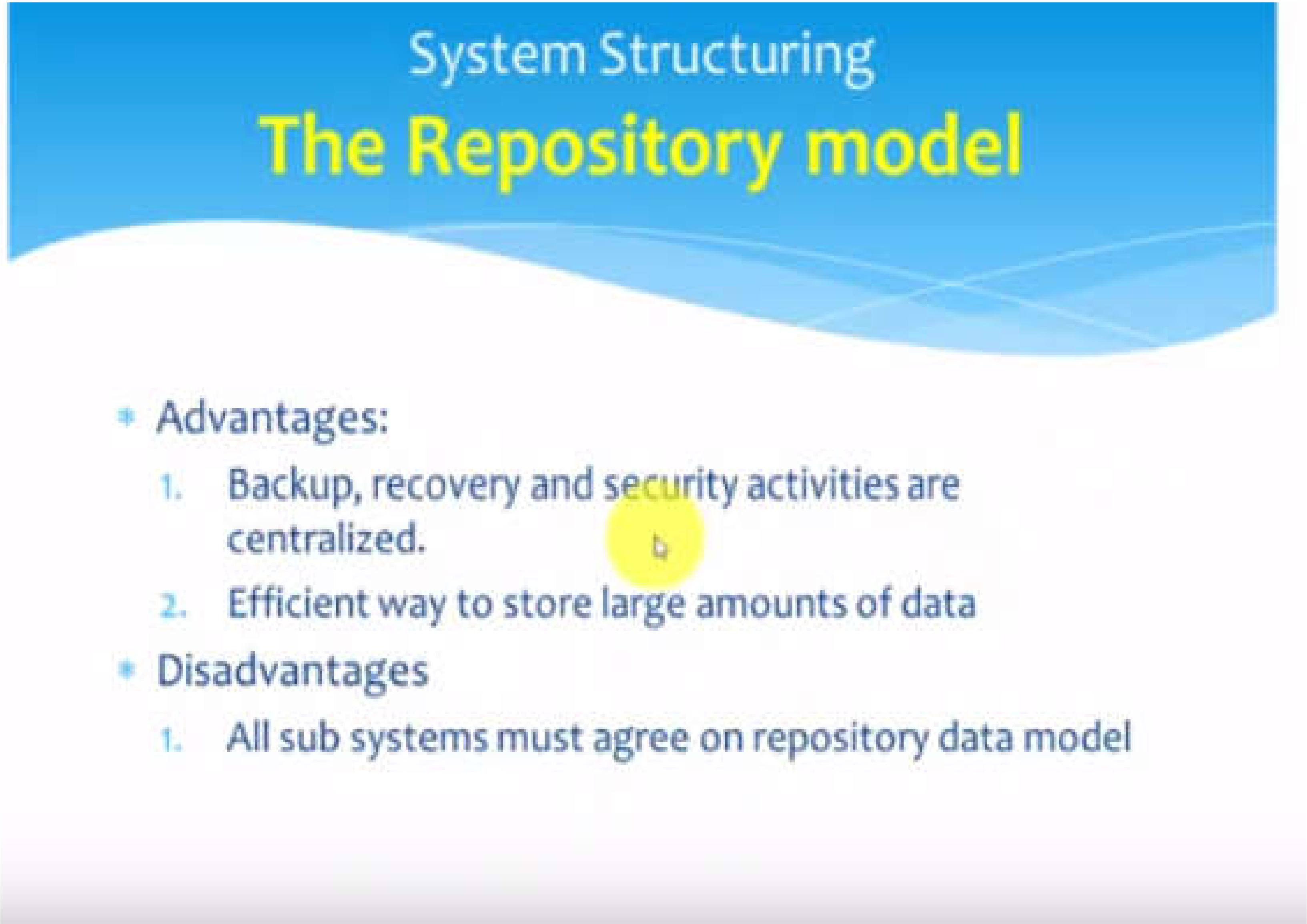
Sub systems sharing central Database



System Structuring

The Repository model

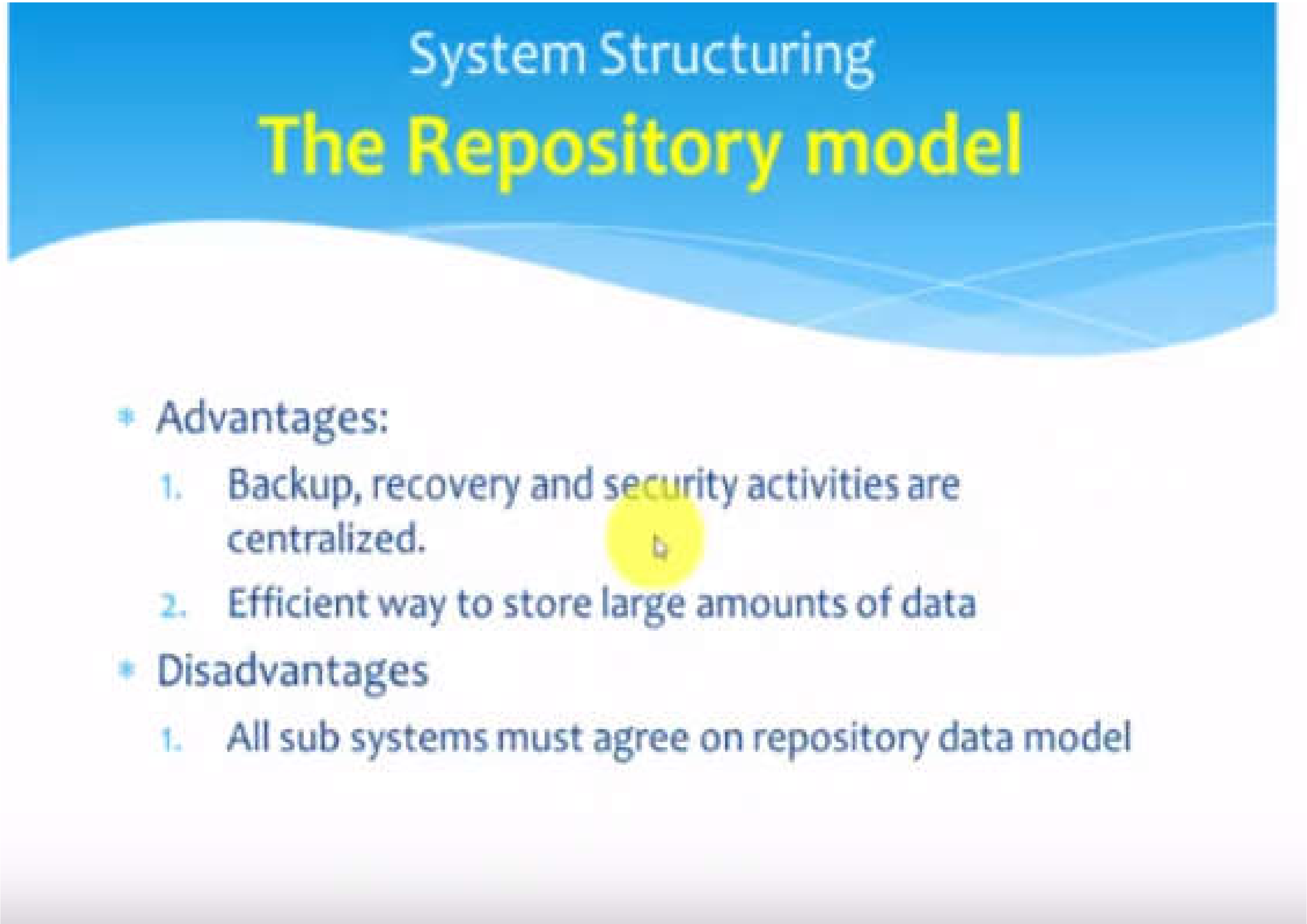
- When large amounts of data are to be shared, the **repository model of sharing** is most commonly used a this is an efficient data sharing mechanism.



System Structuring

The Repository model

- Advantages:
  1. Backup, recovery and security activities are centralized.
  2. Efficient way to store large amounts of data
- Disadvantages
  1. All sub systems must agree on repository data model



System Structuring

The Repository model

- Advantages:
  1. Backup, recovery and security activities are centralized.
  2. Efficient way to store large amounts of data
- Disadvantages
  1. All sub systems must agree on repository data model

2. Client-server model

A system model where the system is organized as a set of services and associated servers and clients that access & use the services.

The major components of this model are:

1. A set of servers that offer services to other subsystems such as printing, data management, etc.
2. A set of clients that call on services offered by the servers.
3. A network that allows clients to access these services.

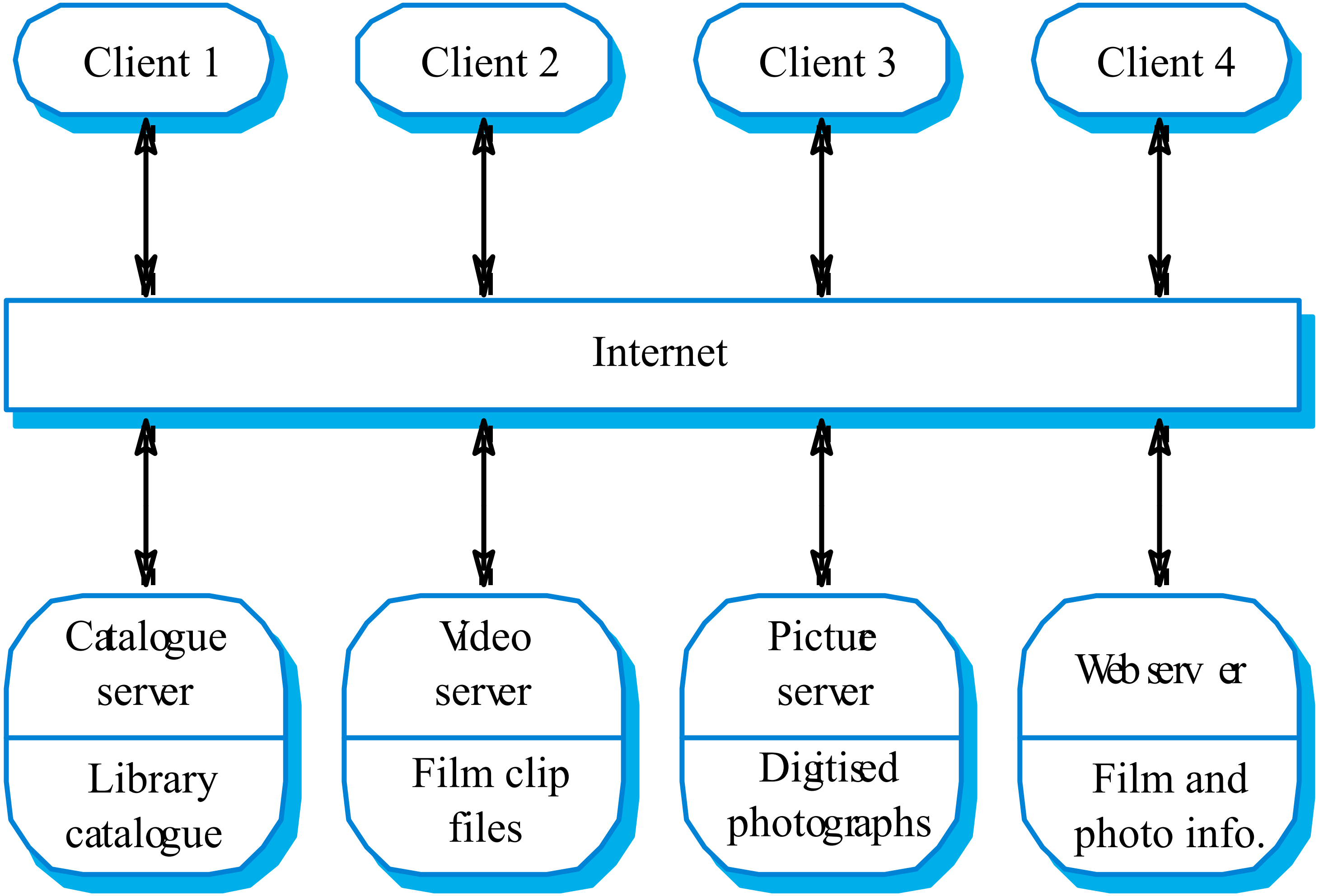


Fig:Client-server model

Client –Server Model

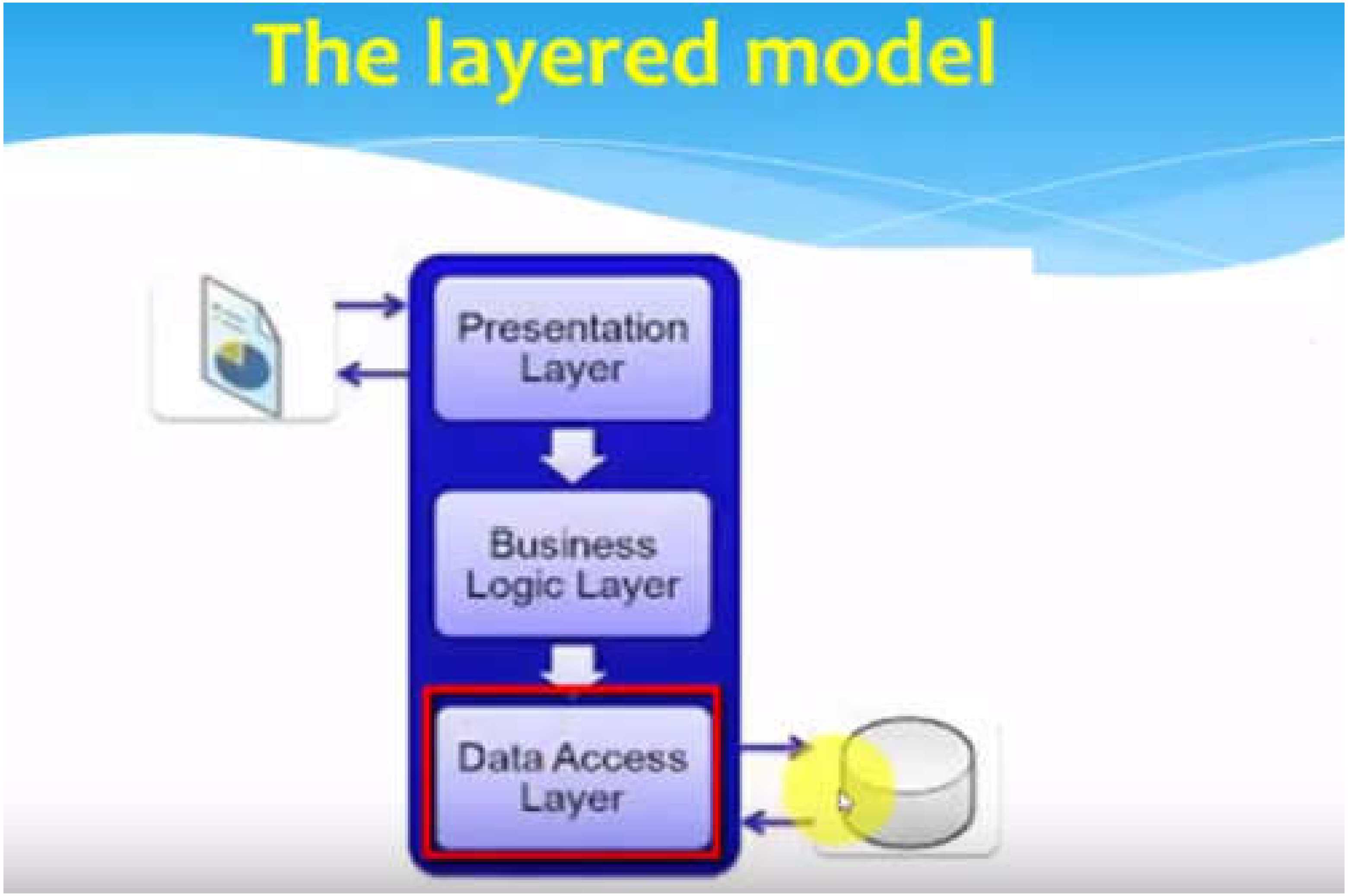
- Advantages
  - – Distribution of data is straightforward;
  - – Makes effective use of networked systems. May require cheaper hardware;
  - – Easy to add new servers or upgrade existing servers.
- Disadvantages
  - – No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;
  - – Redundant management in each server;
  - – No central register of names and services - it may be hard to find out what servers and services are available.

Client –Server Model

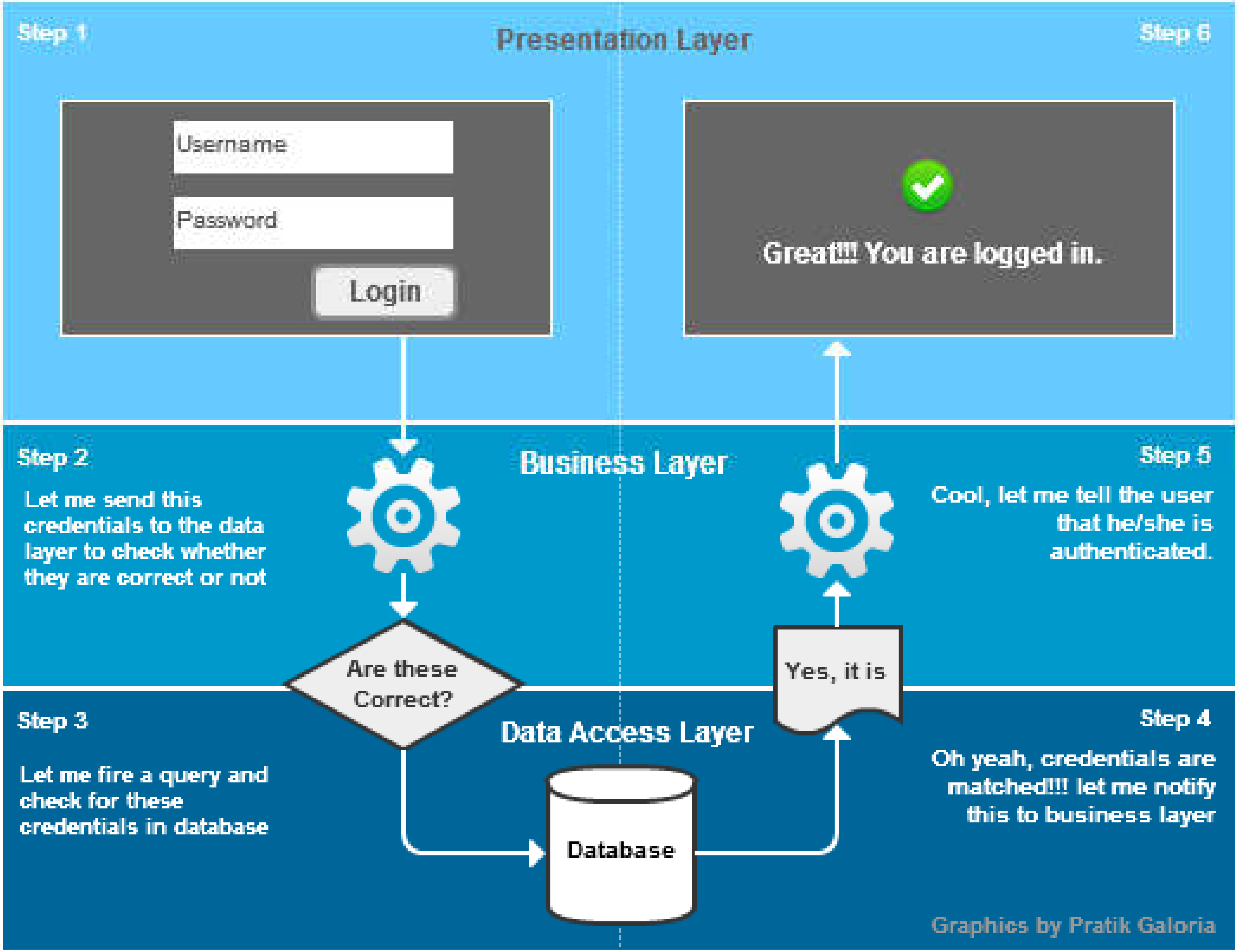
- Advantages
  - – Distribution of data is straightforward;
  - – Makes effective use of networked systems. May require cheaper hardware;
  - – Easy to add new servers or upgrade existing servers.
- Disadvantages
  - – No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;
  - – Redundant management in each server;
  - – No central register of names and services - it may be hard to find out what servers and services are available.

3. Layered Model

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Each layer can be thought of as an abstract machine whose machine language is defined by the services provided by the layer beneath it.
- This language used is to implement the next level of abstract machine.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.



The layered model



Step 1

Presentation Layer

Step 6

Step 2

Let me send this credentials to the data layer to check whether they are correct or not

Business Layer

Step 5

Cool, let me tell the user that he/she is authenticated.

Step 3

Let me fire a query and check for these credentials in database

Data Access Layer

Step 4

Oh yeah, credentials are matched!!! let me notify this to business layer

Graphics by Pratik Galoria

!! Attendance please !!

Thank You!!!