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: 7th Feb, 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

2. Control styles

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.

i) Centralised control

- One sub-system has overall responsibility for control and starts and stops other sub-systems.

ii) Event-based control

- Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

i)Centralized control

- Here, one subsystem is designated as the system controller & has responsibility for managing the execution of other sub systems.
- Centralized control models fall into two classes, depending on whether the controlled sub-system execute sequentially or in parallel

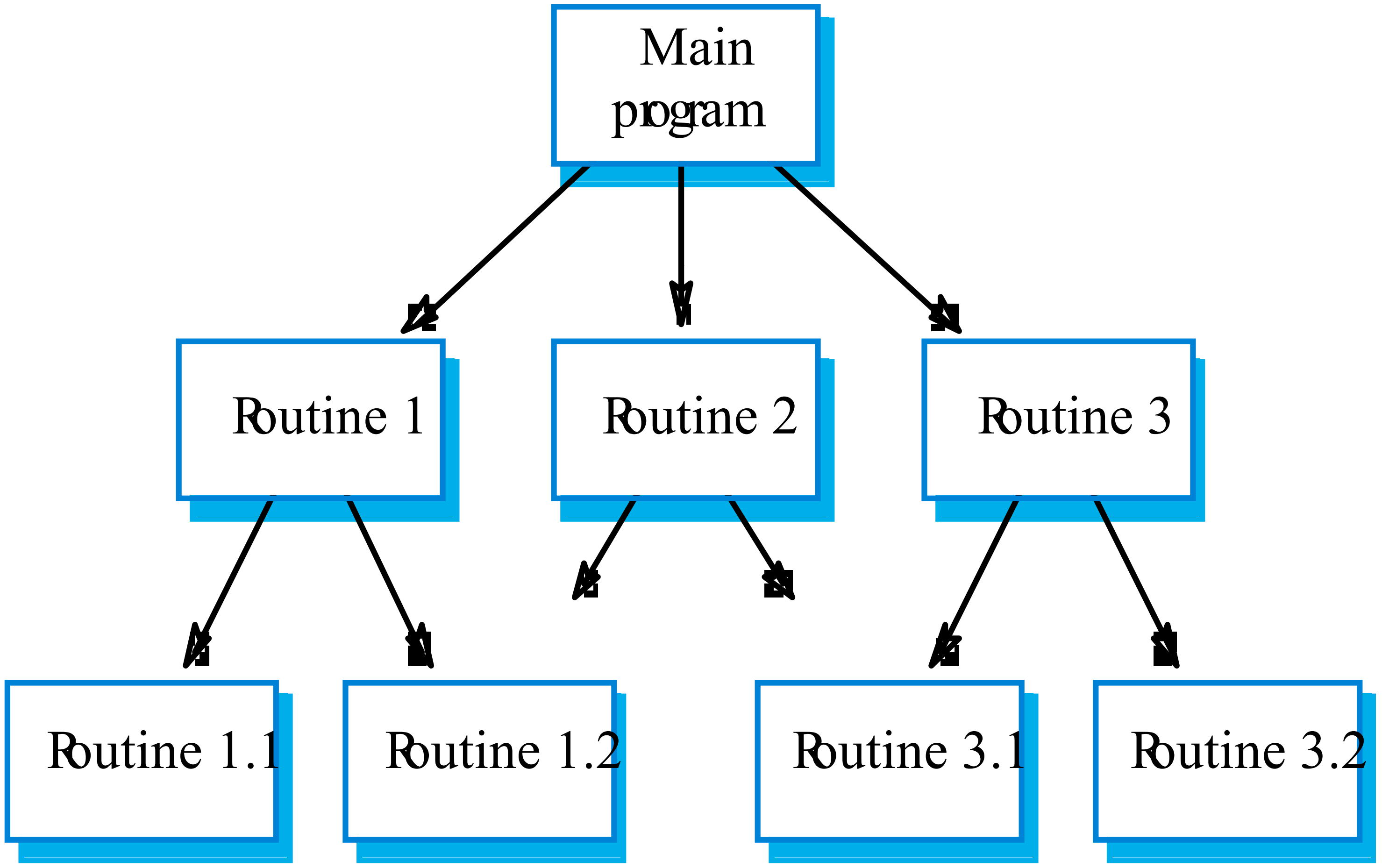
A. Call-return model

   – Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems.

B. Manager model

   – Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes.

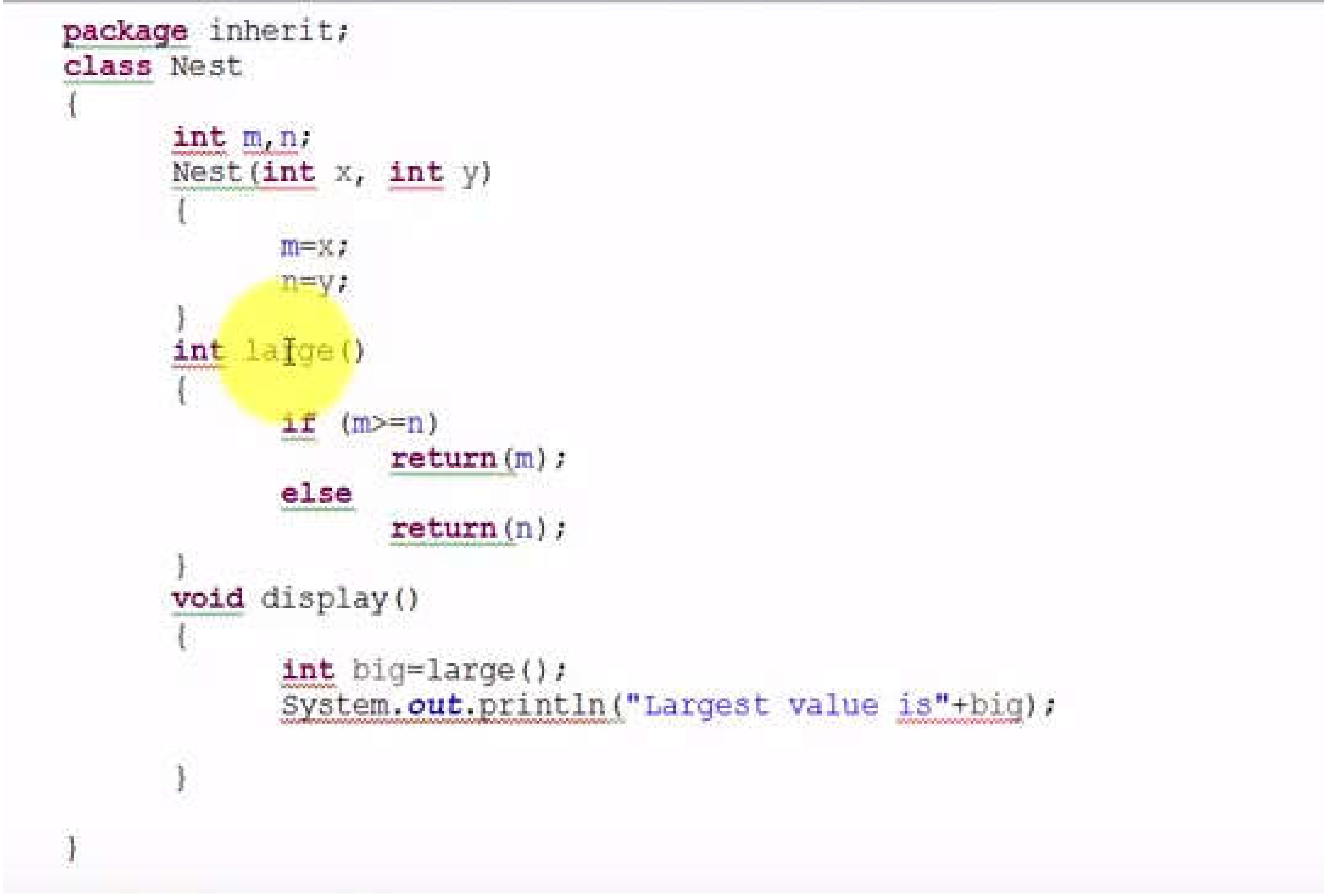
A. Call –Return Model



A. Call –Return Method

- In fig. the main program call routines 1,2 &3; Routine 1 can call Routine 1.1 & 1.2;Routine 3 can call Routines 3.1 & 3.2; & so on.
- Here, control passes from a higher-level routine in the hierarchy to the lower level
- Then it returns to a point where the routine was called.
- The currently executing subroutine has responsibility to control & can either call other routines or return control to its parents.

Example: **A. Call –Return Model**

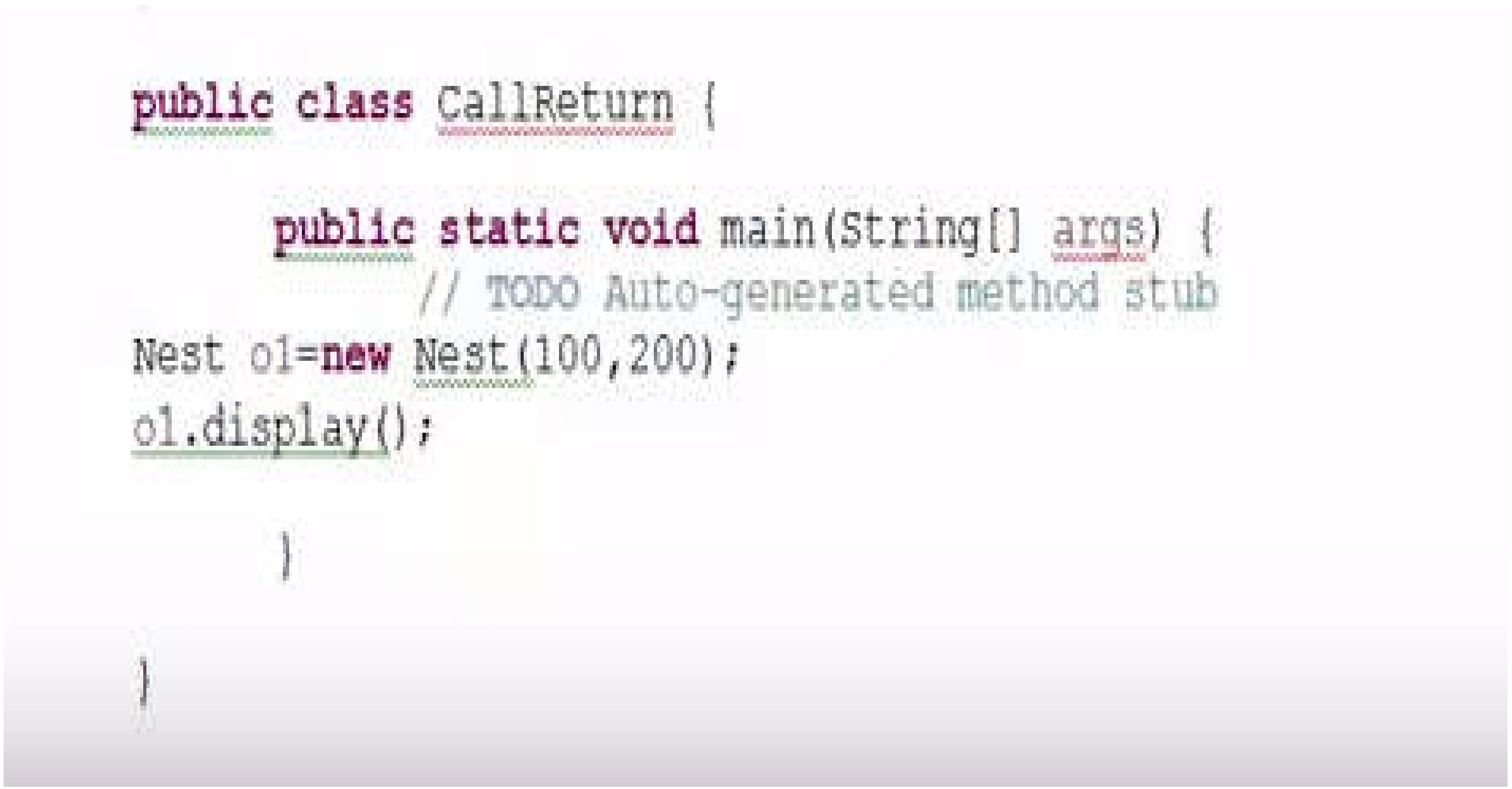


```
package inherit;
class Nest
{
        int m,n;
        Nest(int x, int y)
        {
                m=x;
                n=y;
        }
        int large()
        {
                if (m>=n)
                        return(m);
                else
                        return(n);
        }
        void display()
        {
                int big=large();
                System.out.println("Largest value is"+big);

        }

}
```

A. Call –Return Method



```
public class CallReturn {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
Nest o1=new Nest(100,200);
o1.display();

    }

}
```

A. Call –Return Method

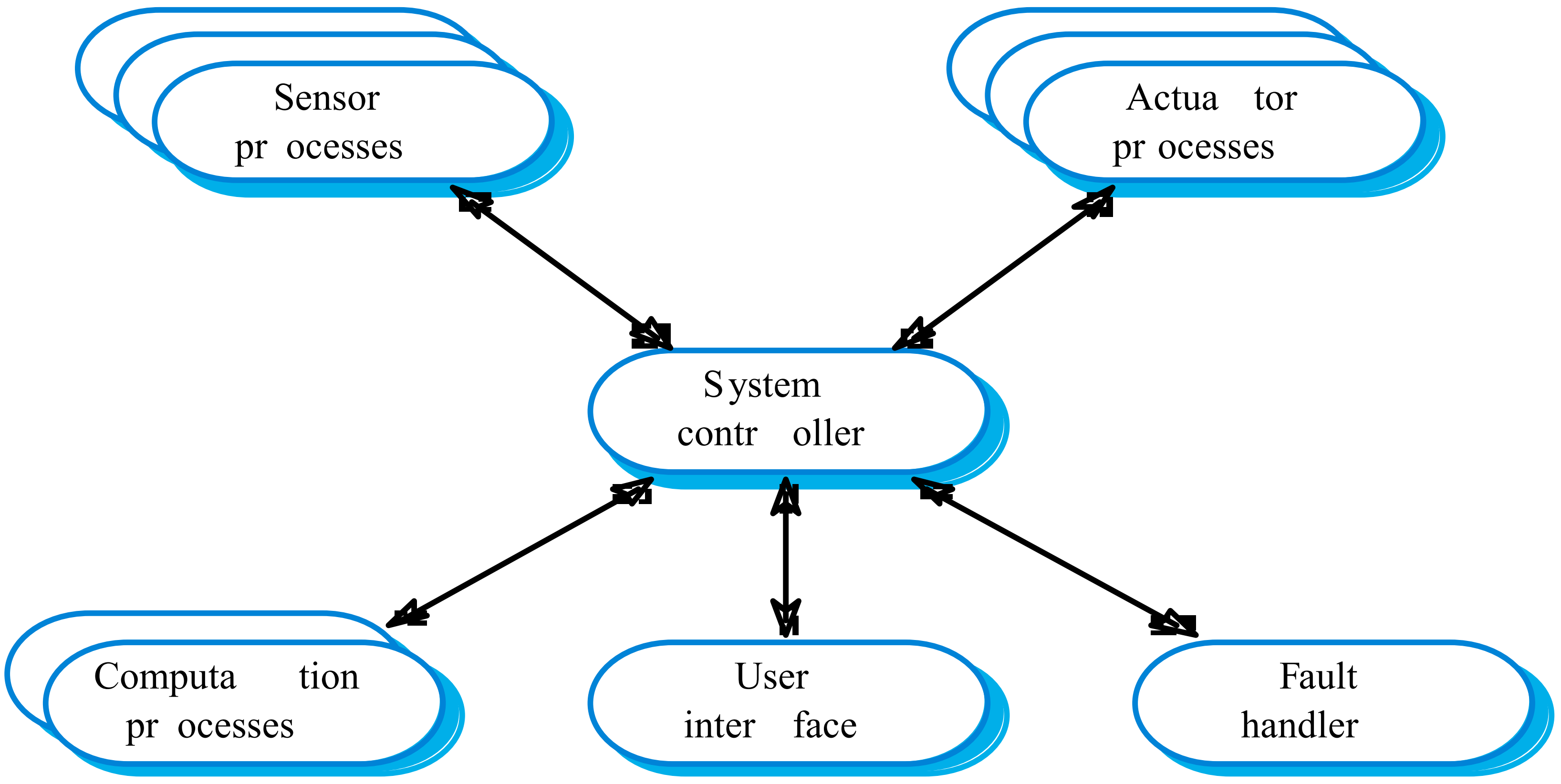
First, main()

Routine1 -> display()

Routine1.1-> large().

Routine 1.2 -> prinln(“ ”)

B. Manager Model



B. Manager Model

- In fig., shows the centralized management model of control for a concurrent system.
- This model is often used in “soft” real-time systems which do not have very tight time constraints.
- The central controller manages the execution of set of processes associated with sensors and actuators.
- The system controller decides when processes should be started or stop depending on system state variables.

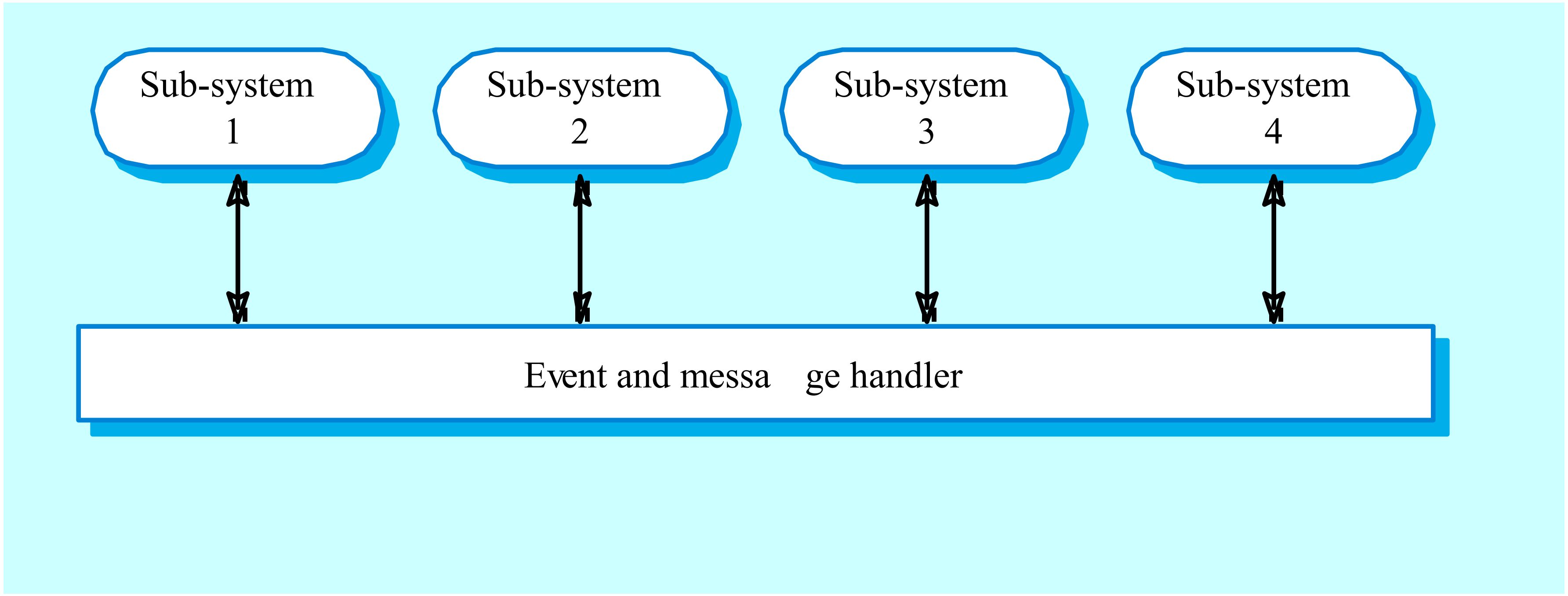
ii. Event Based Control

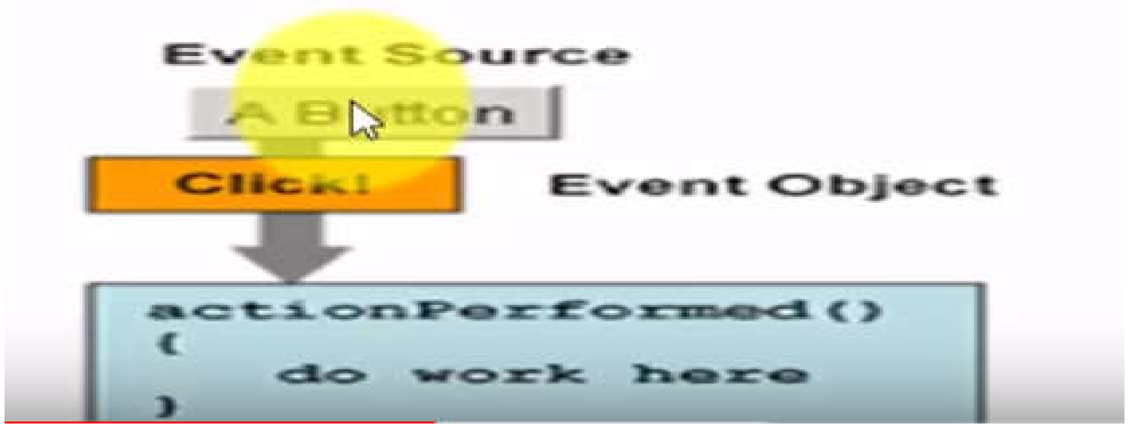
- Event driven control models are driven by externally generated events where the timing of the event is outside the control of the process that handles that event.
- Two principal event-driven models
  - A. Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so;
  - B. Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing.

A. Broadcast Model

- Here, sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event.
- Difference of this with centralized model is that the control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them.
- Subsystem decide which event they require, & the event of interest to them.
- Dof this model is that sub-systems don’t know if or when an event will be handled.

A. Broadcast Model

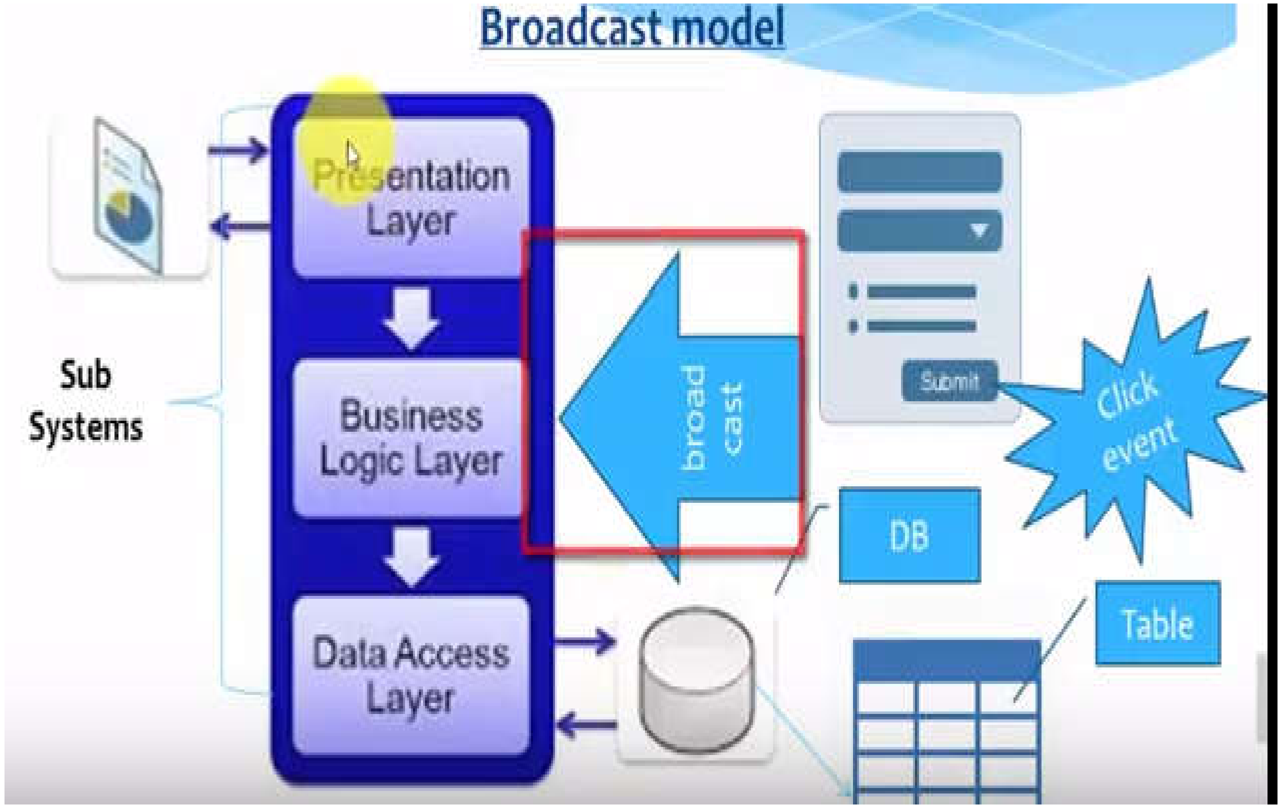




```
actionPerformed()
{
     do  work  here
}
```

A. Broadcast Model

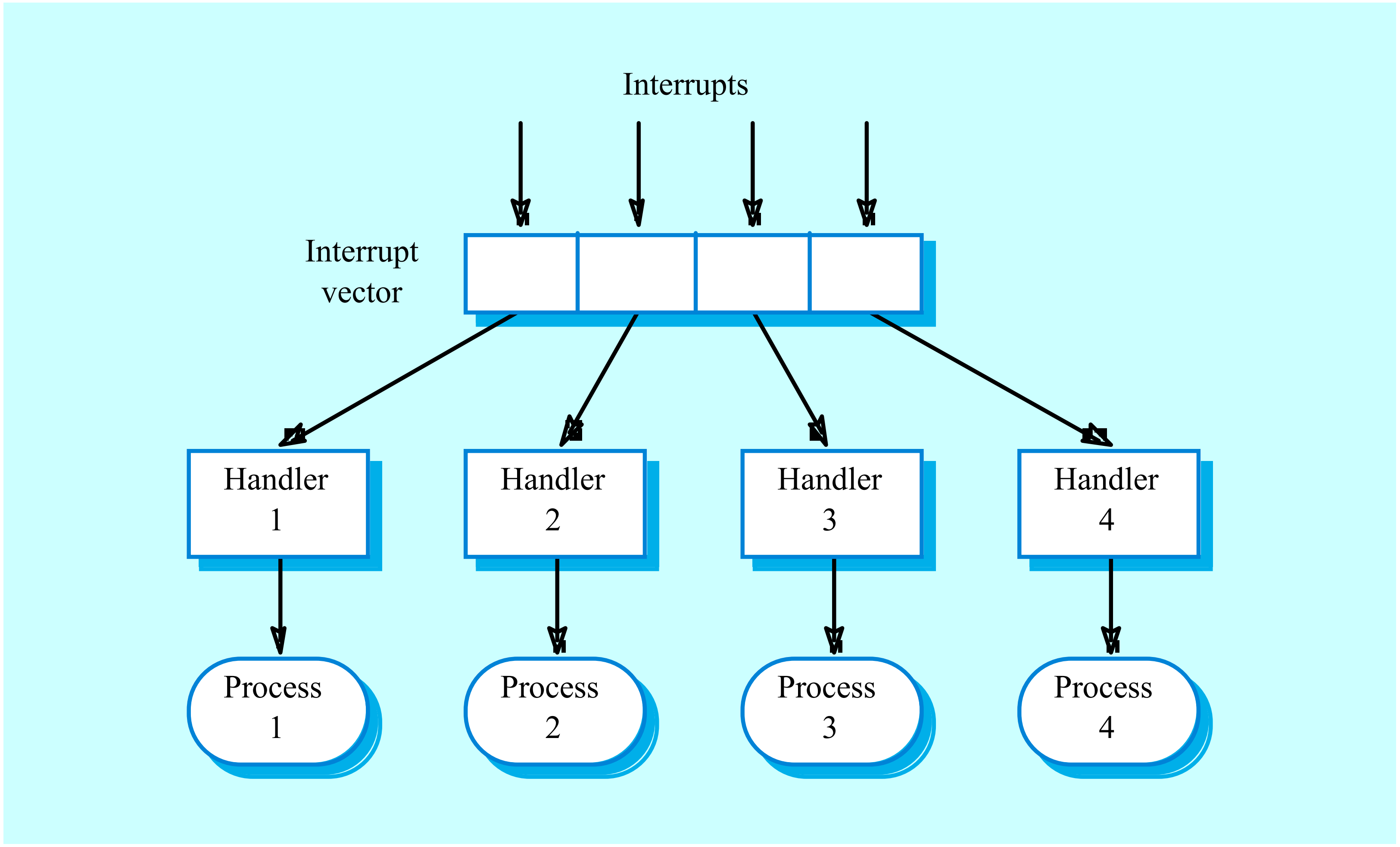
After form is filled and submit, event occurs and it is broadcast on 3 layers simultaneously.



B. Interrupt-driven systems

- Used in real-time systems where fast response to an event is essential.
- There are known interrupt types with a handler defined for each type.
- Each type is associated with a memory location and a hardware switch causes transfer to its handler.
- Allows fast response but complex to program and difficult to validate.

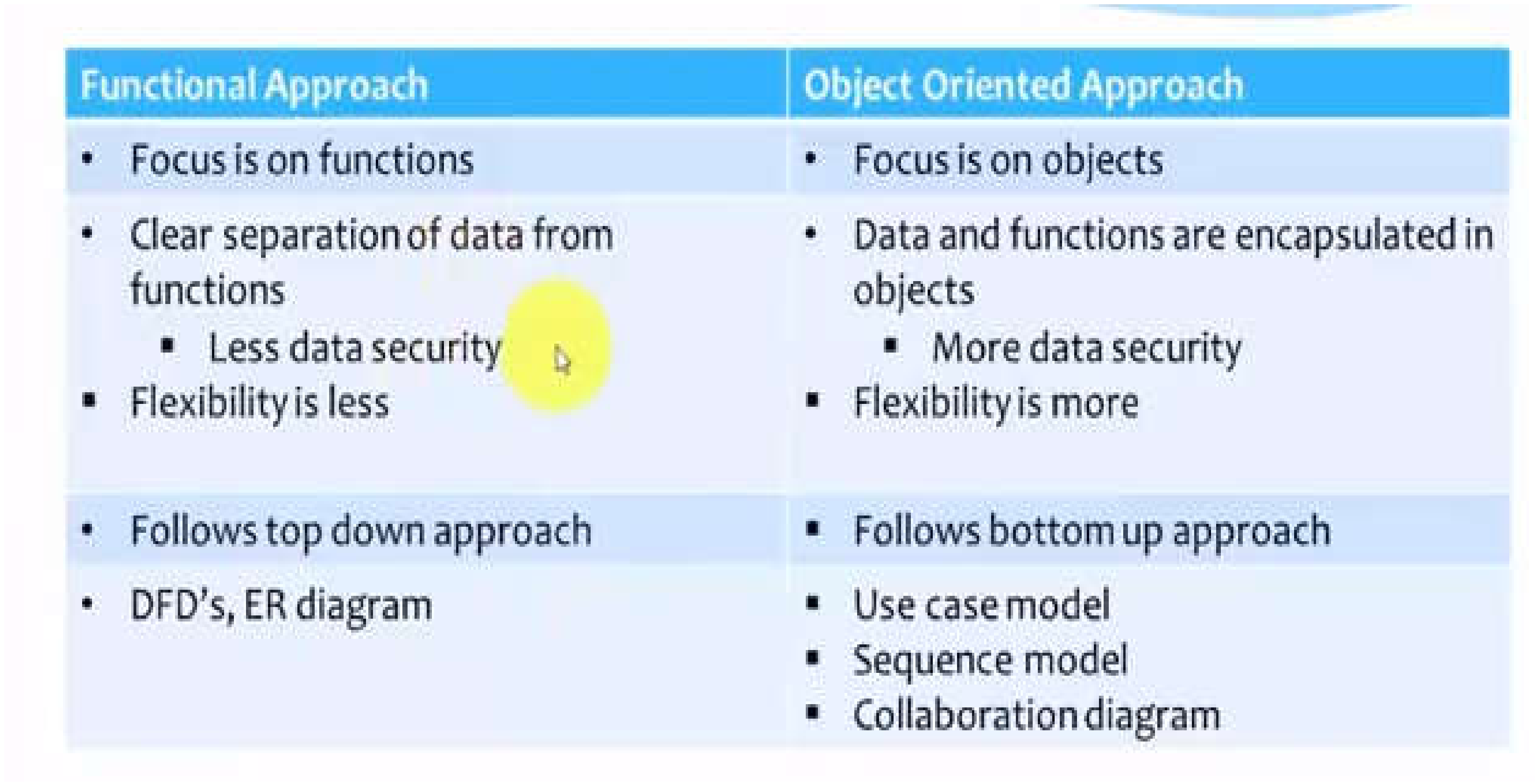
B. Interrupt-driven systems



B. Interrupt-driven systems

- In fig., there are known number of interrupt types with the handler defined for each type.
- Each interrupt is associated with a memory location where its handler’s address is stored.
- When an interrupt of a particular type is received, h/w switch causes control to be transferred immediately to its handler.
- This interrupt handler may then start or stop other processes in response to the event signalled by the interrupt.

Design strategy comparison



Functional Approach	Object Oriented Approach
• Focus is on functions	• Focus is on objects
• Clear separation of data from functions ▪ Less data security ▪ Flexibility is less	• Data and functions are encapsulated in objects ▪ More data security ▪ Flexibility is more
• Follows top down approach	▪ Follows bottom up approach
• DFD's, ER diagram	▪ Use case model ▪ Sequence model ▪ Collaboration diagram

Distributed systems

- Virtually all large computer-based systems are now distributed systems.
- Here, Information processing is distributed over several computers rather than confined to a single machine.
- Distributed software engineering is therefore very important for enterprise computing systems.

System types

- Personal systems that are not distributed and that are designed to run on a personal computer or workstation.
- Embedded systems that run on a single processor or on an integrated group of processors.
- Distributed systems where the system software runs on a loosely integrated group of cooperating processors linked by a network.

Distributed systems architectures

- Client-server architectures
  - Distributed services which are called on by clients.
  - Servers that provide services are treated differently from clients that use services.
- Distributed object architectures
  - No distinction between clients and servers.
  - The server may be thought of as a set of interactive objects whose location is irrelevant.
  - Any object on the system may provide and use services from other objects.

Client-server architectures

- The application is modelled as a set of services that are provided by servers and a set of clients that use these services.
- Clients know of servers but servers need not know of clients.
- Clients and servers are separate logical processes as shown in fig below (Fig.1)
- Several server processes can run on a single server processor so there is not necessarily 1:1 mapping between processors & processes.

Ex: Client-server architectures

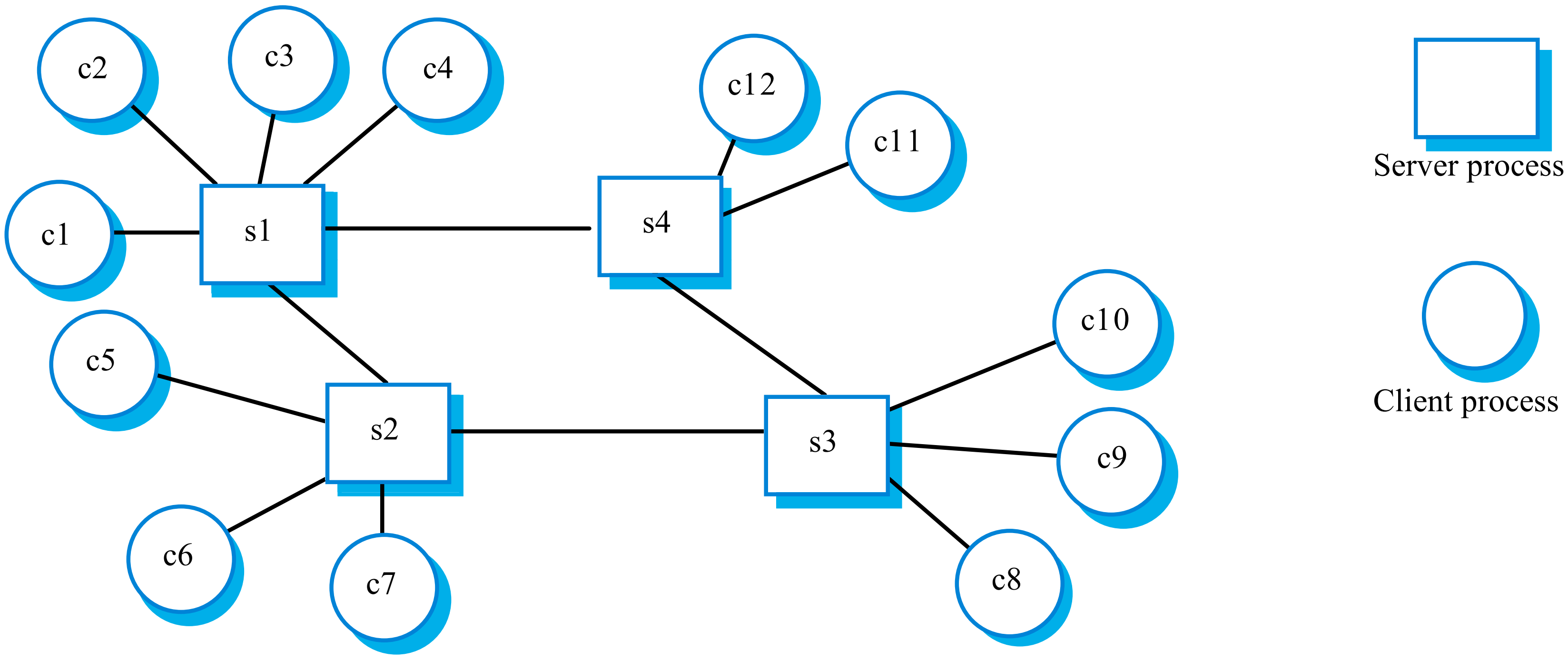


Fig. 1

Ex: Client-server architectures

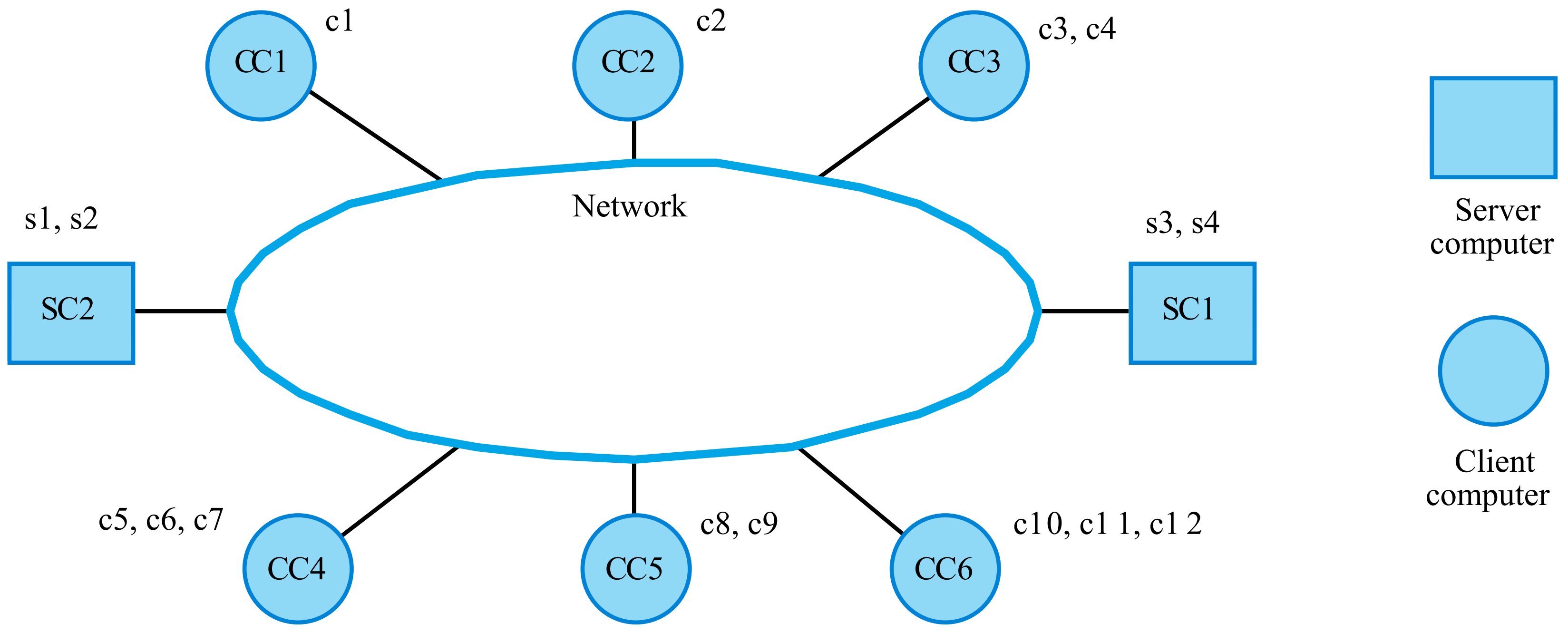


Fig.2

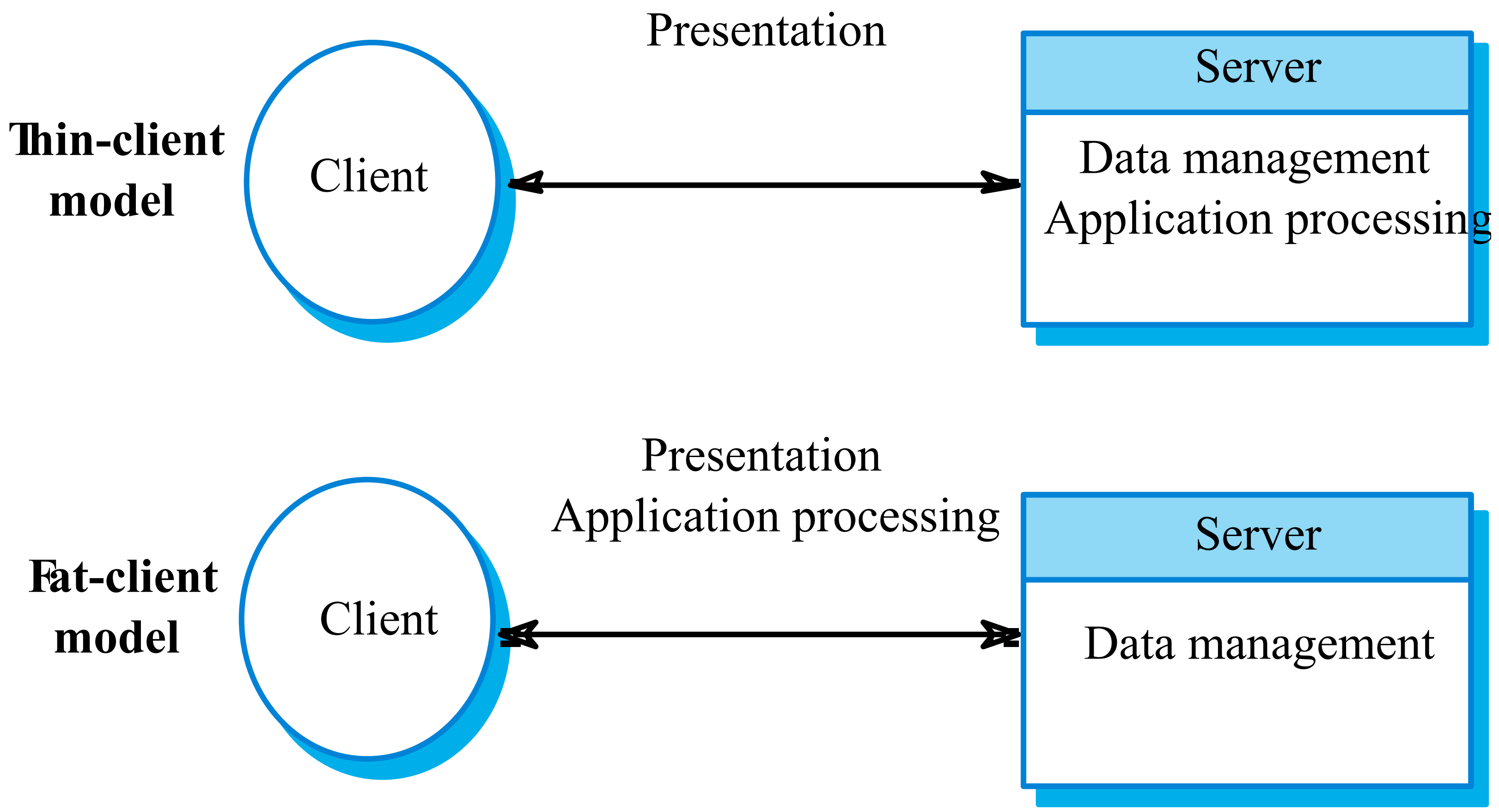
Example - Computers in a C/S network

- Fig. 2 shows the physical architecture of the system with six client & two server computers.
- These can run the client & server processes as shown in Fig. 1

Thin and fat clients

- Thin-client model
  - In a thin-client model, all of the application processing and data management is carried out on the server.
  - The client is simply responsible for running the presentation software.
- Fat-client model
  - In this model, the server is only responsible for data management.
  - The software on the client implements the application logic and the interactions with the system user.

Thin and fat clients



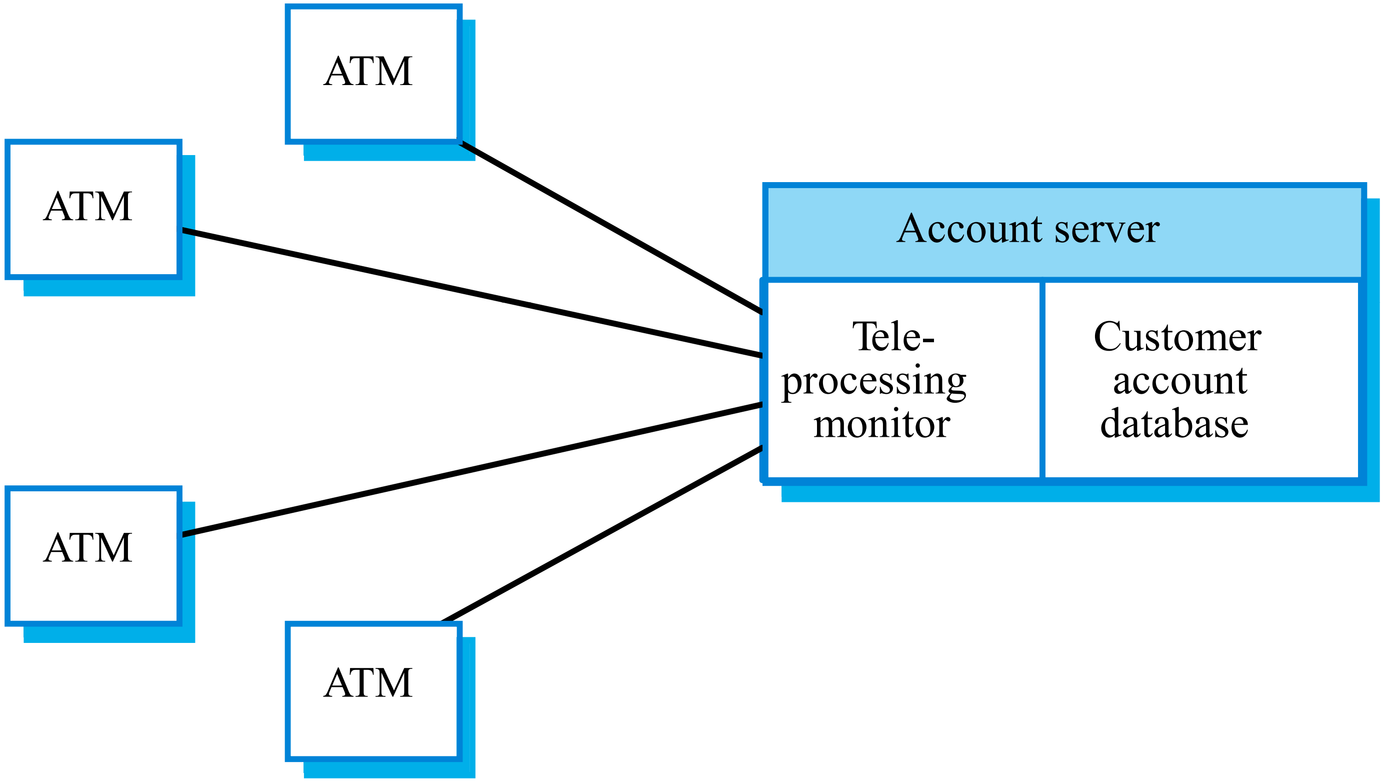
Thin clients

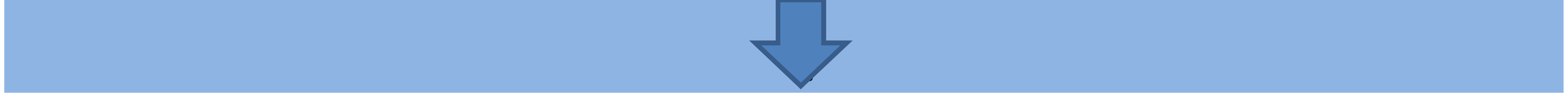
- Used when legacy systems are migrated to client server architectures.
  - The legacy system acts as a server in its own right with a graphical interface implemented on a client.
  - The application itself act as a server-handles the application processing & data management
- A major disadvantage is that it places a heavy processing load on both the server and the network.

Fat clients

- More processing is delegated to the client as the application processing is locally executed.
- The server is essentially a transaction server that manages all database transactions.
- Example- Banking ATM system, where ATM is the client & the server is a mainframe running the customer account database.
- The hardware in the teller machine carries out a lot of customer related processing associated with a transaction.
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients

Client Server ATM System





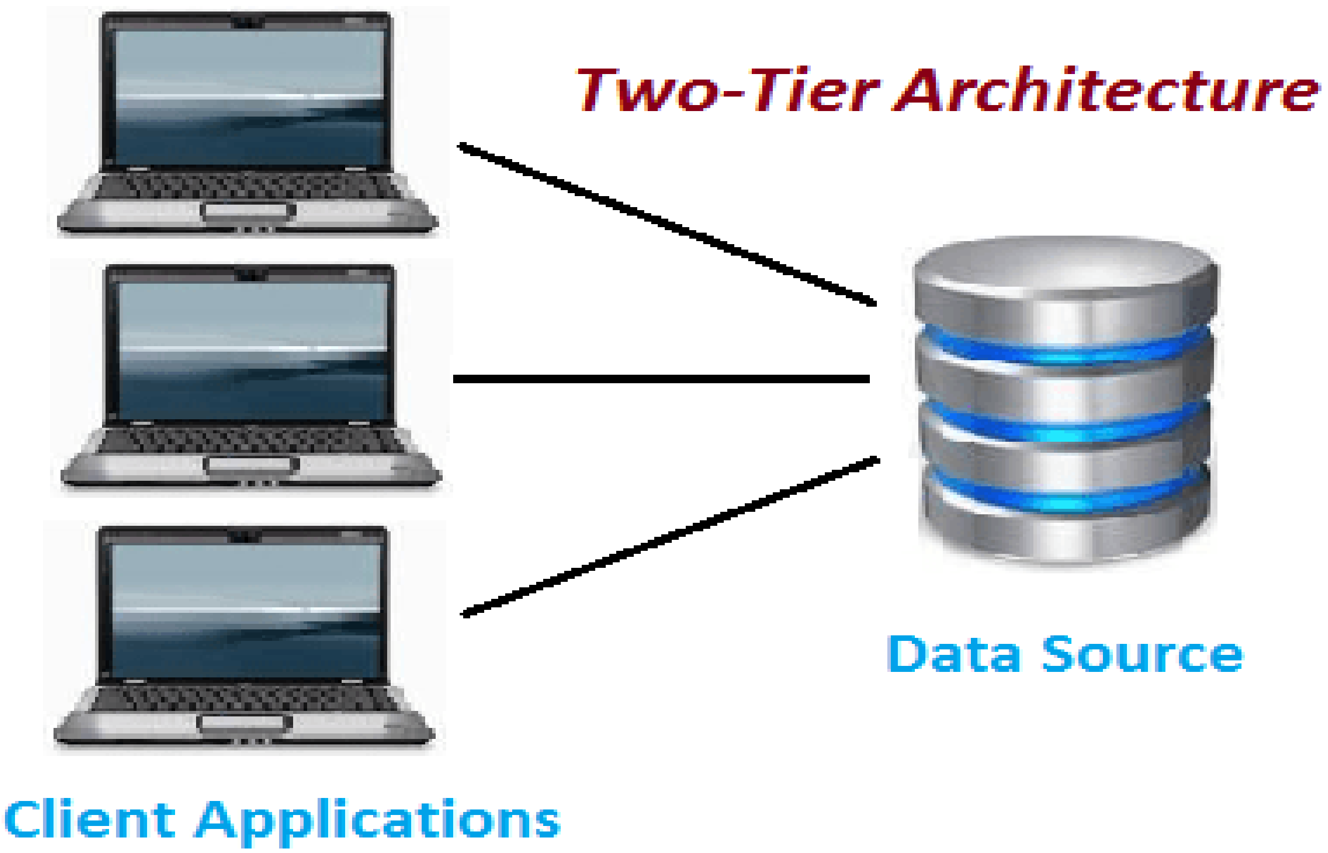
Client Server ATM System

- Fig. above is the ATM distributed system
- The ATMs are not connected directly to the customer database but to a teleprocessing monitor.
- It is a middleware system that organizes communication with remote clients & serializes the client transaction processing by the database.
- Using serial transaction means that the system can recover from faults without corrupting system data.

Disadvantages- Fat clients

- The fat-client model distributes processing more effectively than thin client model but the system management is more complex
- Application functionality is spread over many computers
- When the application software is to be changed, reinstallation is needed on every computer
- This can be a major cost if there are hundreds of clients in the system.

Two-Tier Architecture:



Two-Tier Architecture:

- The two-tier is based on Client Server architecture.
- The two-tier architecture is like client server application.
- The direct communication takes place between client and server.
- There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster.

Two-Tier Architecture:

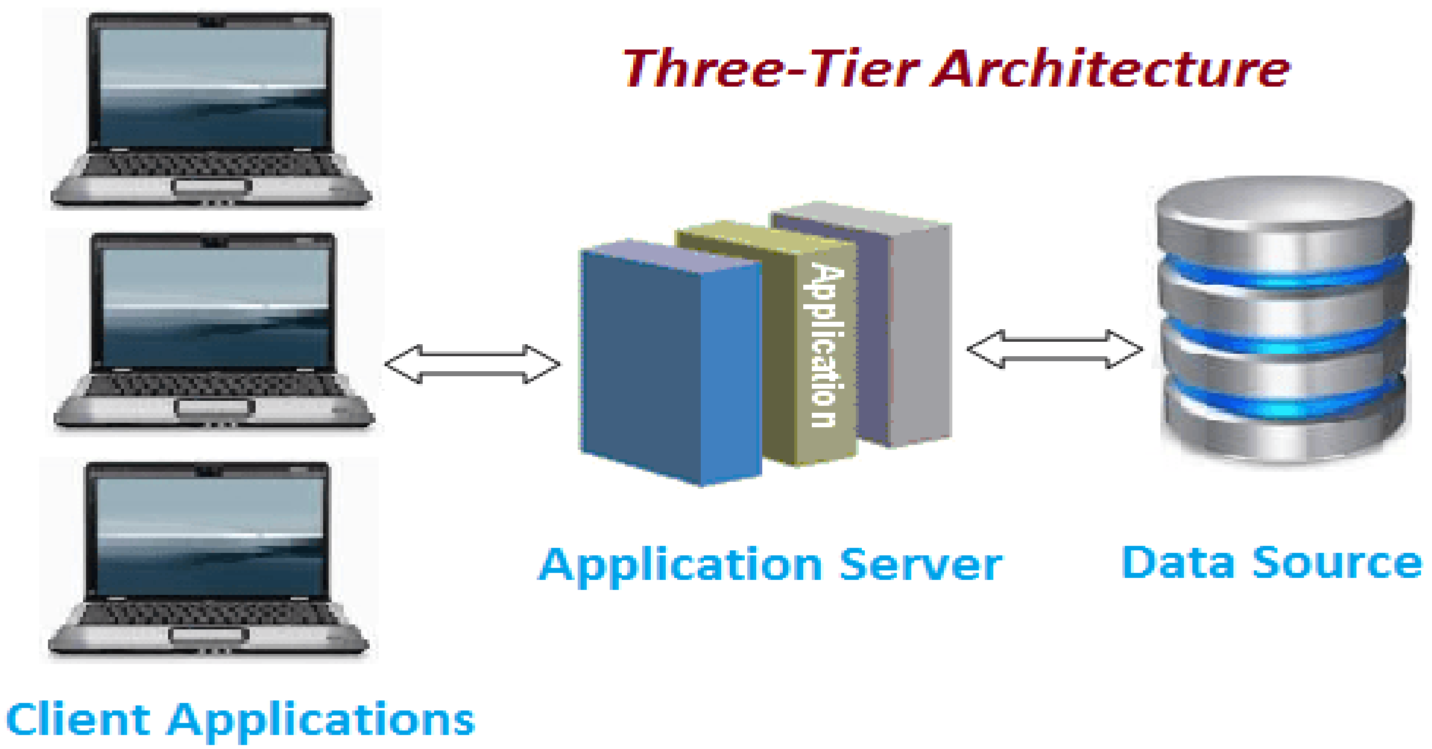
Advantages:

- Easy to maintain and modification is bit easy
- Communication is faster

Disadvantages:

- In two tier architecture application performance will be degrade upon increasing the users.
- Cost-ineffective

Three-Tier Architecture:



Three-Tier Architecture:

1) Client layer:

It is also called as *Presentation layer* which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

Three-Tier Architecture:

2) Business layer:

In this layer all business logic written like validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

Three-Tier Architecture:

3) Data layer:

In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.

Three-Tier Architecture:

Advantages

- High performance, lightweight persistent objects
- Scalability – Each tier can scale horizontally
- Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
- High degree of flexibility in deployment platform and configuration

Three-Tier Architecture:

- Better Re-use
- Improve Data Integrity
- Improved Security – Client is not direct access to database.
- Easy to maintain and modification is bit easy, won't affect other modules
- In three tier architecture application performance is good.

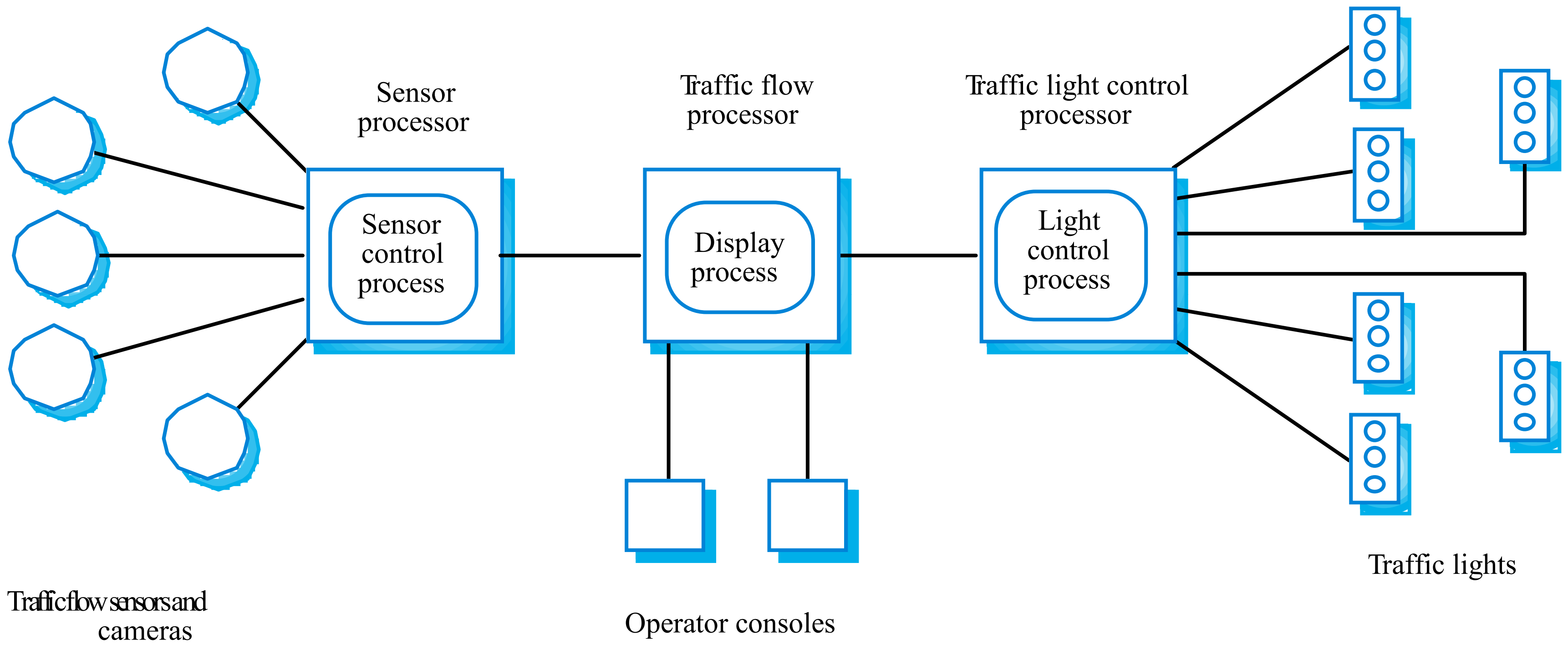
Disadvantages

- Increase Complexity/Effort

Multiprocessor architectures

- Simplest distributed system model where
- System composed of multiple processes which may (but need not) execute on different processors.
- This process is common in large real-time systems.
- These systems
  - Collect information
  - Make decision using information &
  - Send signals to actuator to modify the system's environment
- Distribution of process to processor may be pre-determined or may be under the control of a dispatcher.

Example - A multiprocessor traffic control system



Example - A multiprocessor traffic control system

- In fig. – a simplified model of the traffic control system is shown.
- A set of distributed sensors collects information on the traffic flow & processes locally
- Operators make decisions using this information & give instruction to a separate traffic light control process
- Here, there are separate logical processes for managing sensors, control room & traffic light which run on separate processors.

Chapter 3 finished !!

10 marks

!! Attendance please !!

Thank You!!!