Heaven's Light Is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

CSE 3105

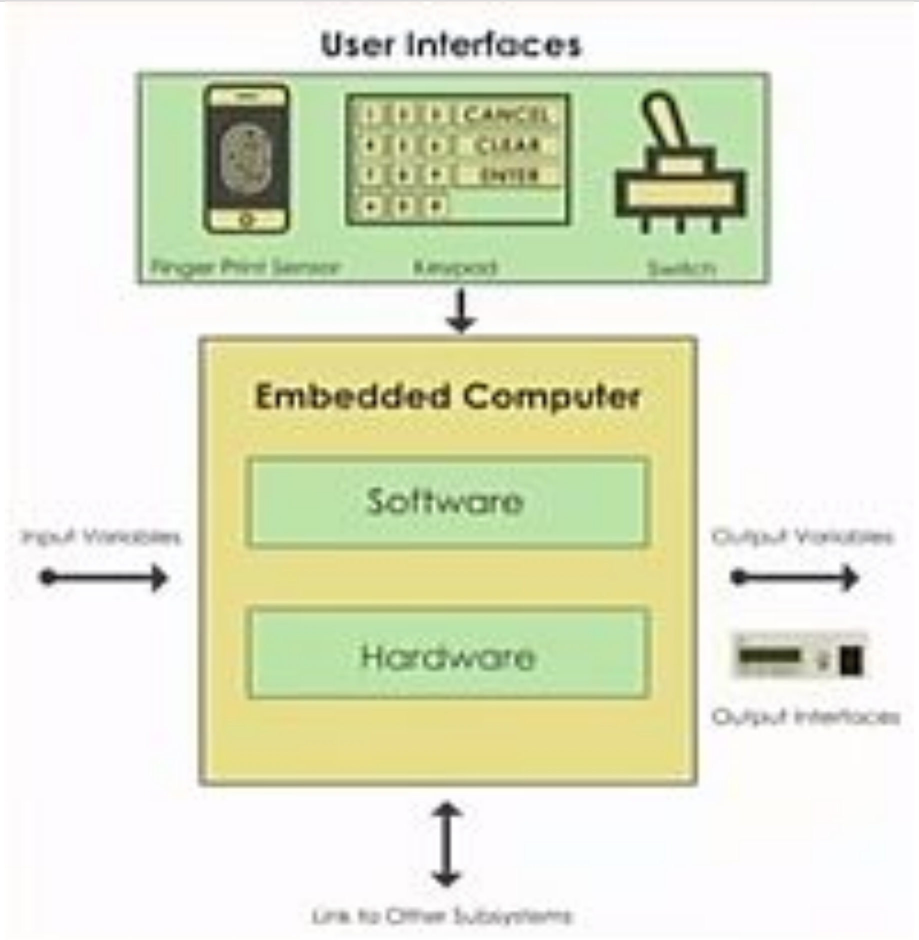
Computer Interfacing & Embedded System

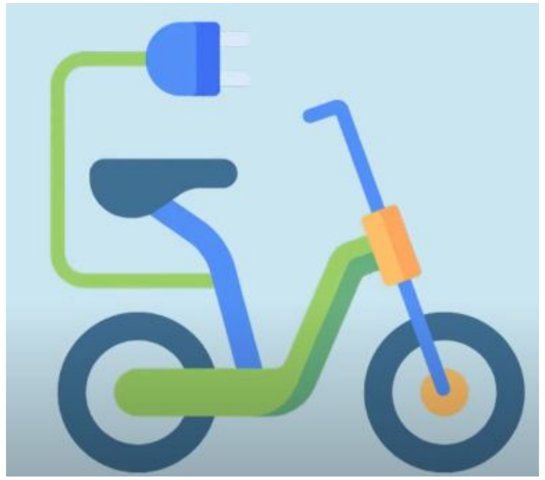
Embedded Computing

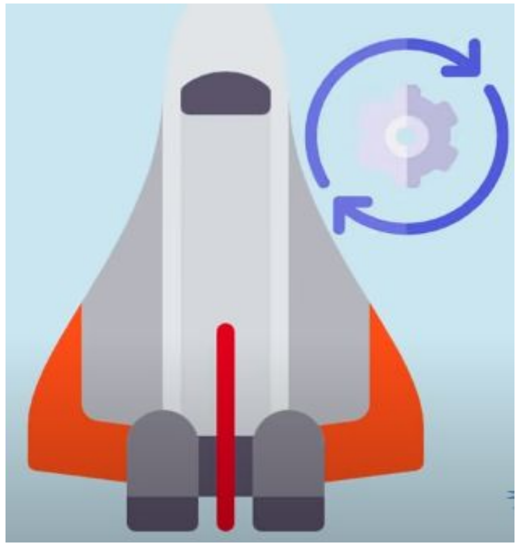
Md. Nasif Osman Khansur
Lecturer
Dept. of CSE, RUET

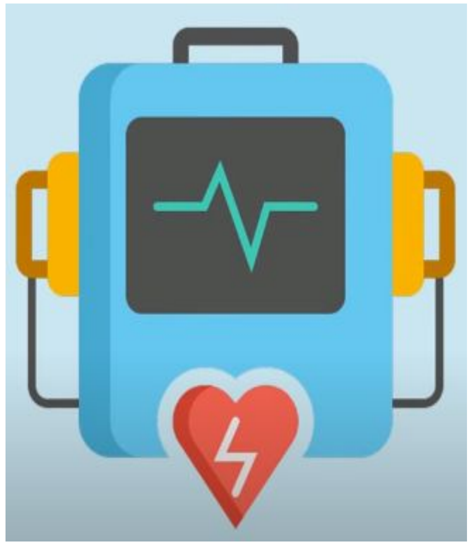
What Embedded System Engineers do?

Gadget Engineers

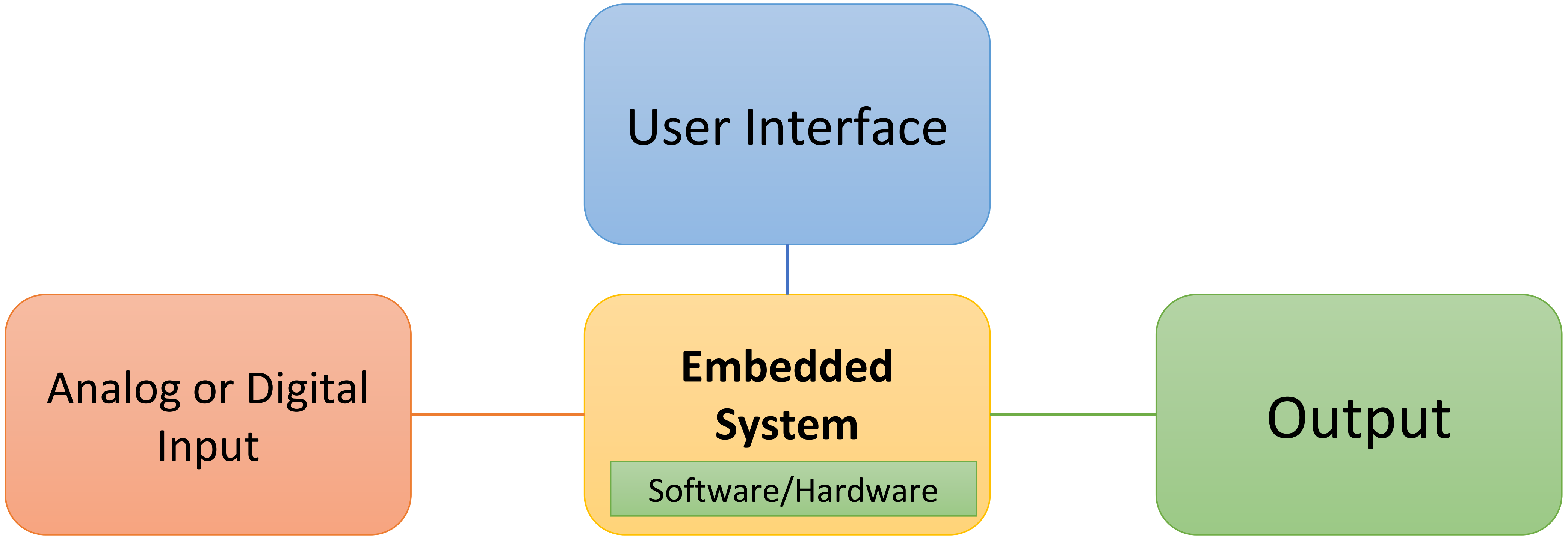




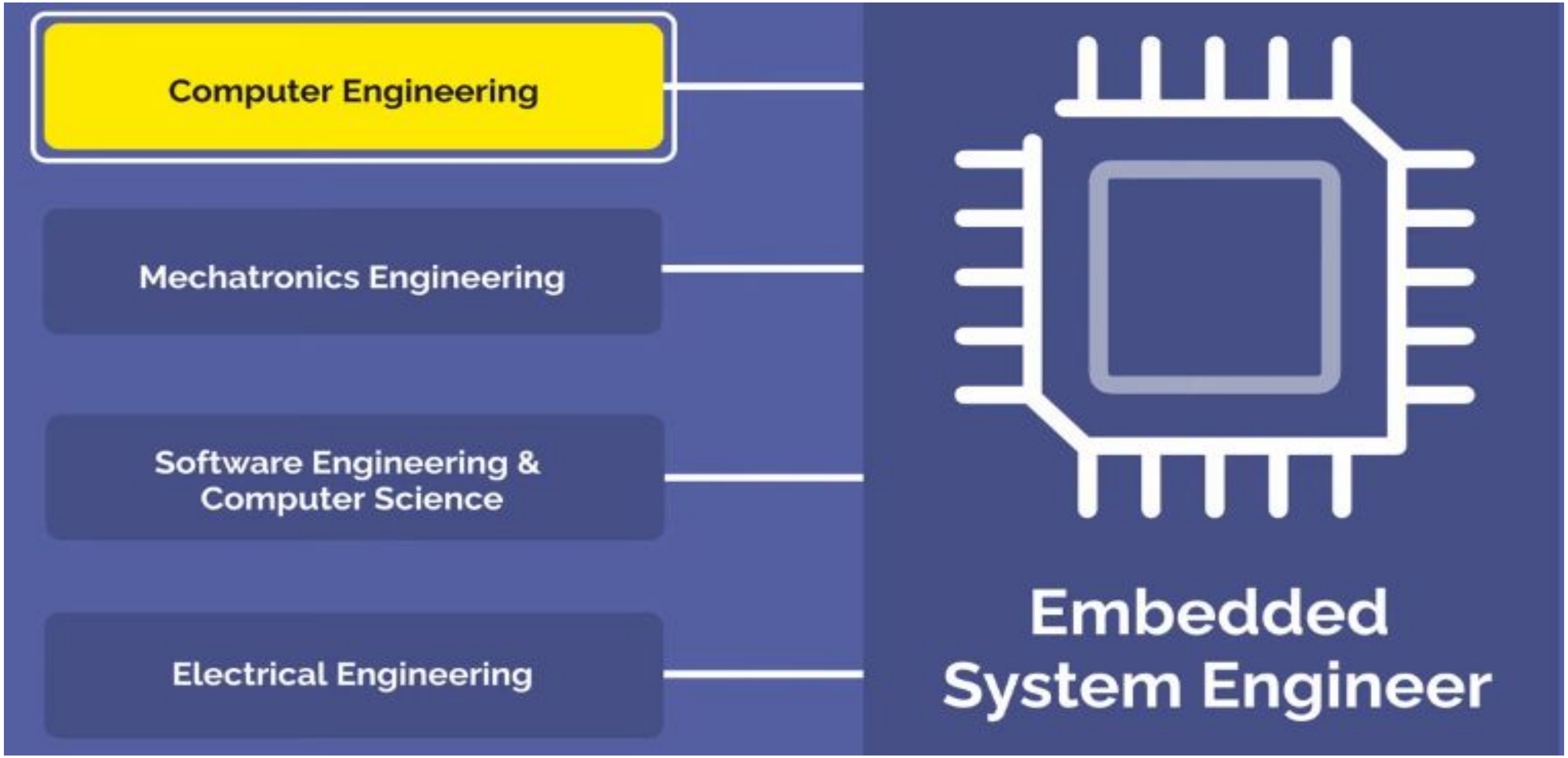




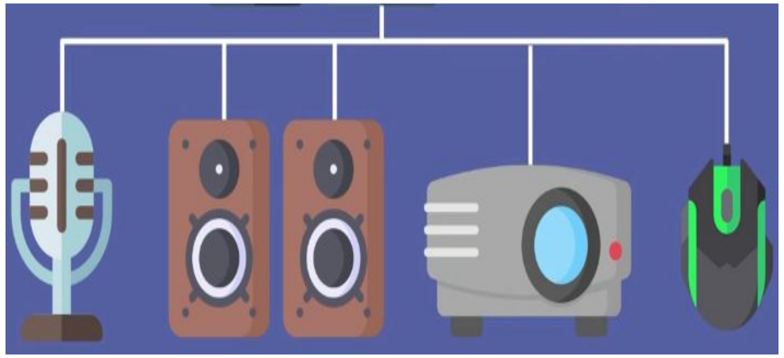
What Embedded System Engineers do?



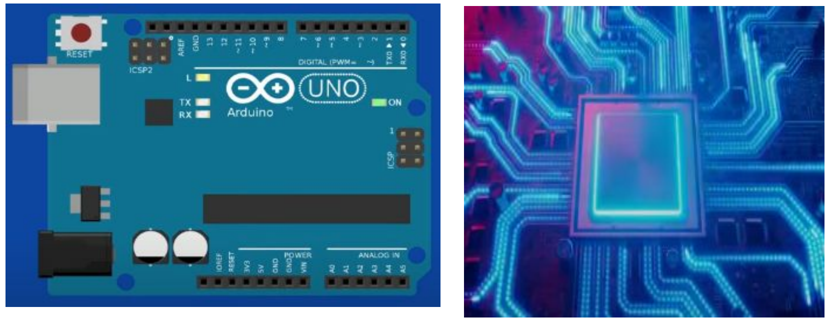
Who would choose Embedded Engineering?



University Course-works needed to become an Embedded System Engineer

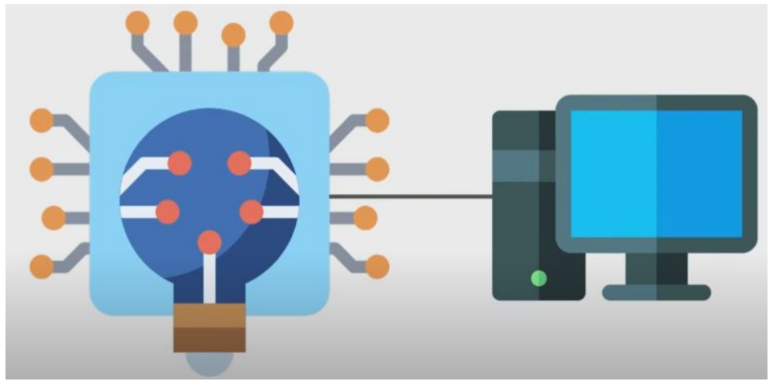


Embedded System Design

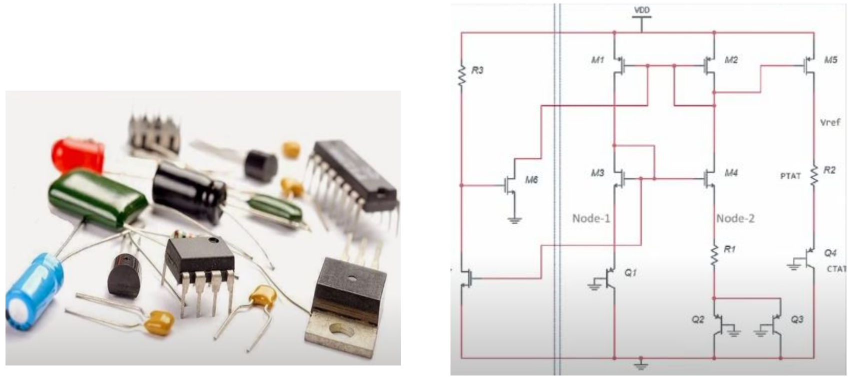


Microcontroller & Microprocessor

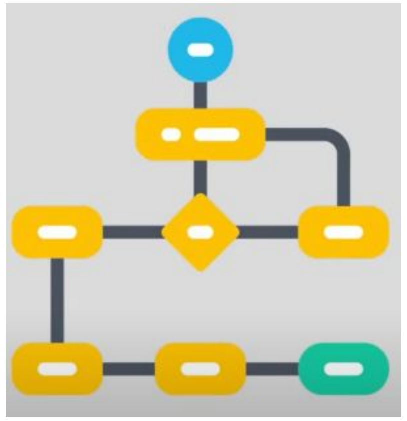
Data Structures



Computer Architecture



Analog & Digital Circuits



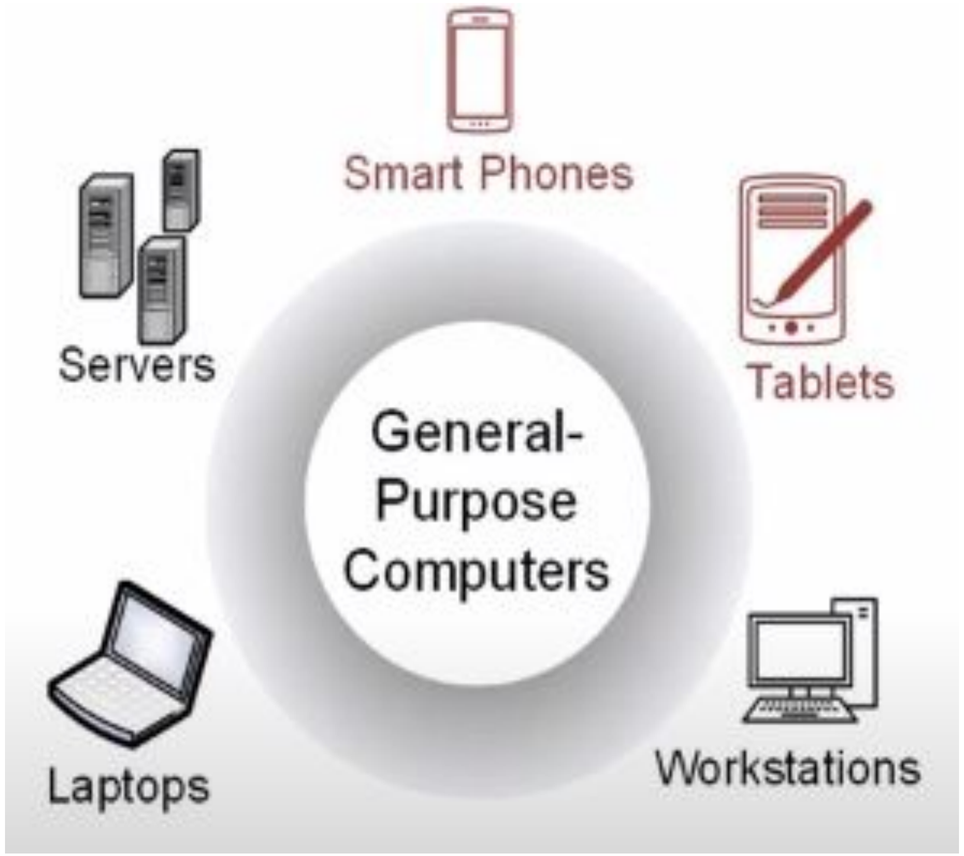
Algorithms

1.1 Computing System

- ✔ We have been brought up in the age of computing
- ✔ Computers are everywhere – Some we see, some we don't.

❑ **What General Purpose Computer can support?**

- ✔ Able to run a variety of software.
- ✔ Contain relatively high performance hardware components (fast processor, data & program storage)
- ✔ Require an OS.



1.1 Computing System

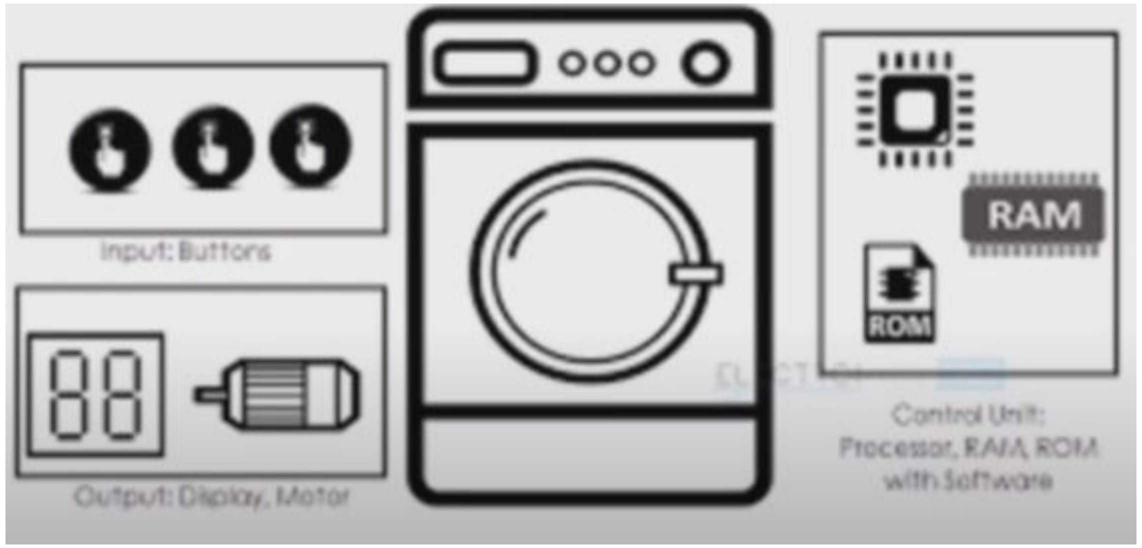
But there's another type of computing system that is often hidden in the environment – **more common and pervasive**

↓

Embedded System

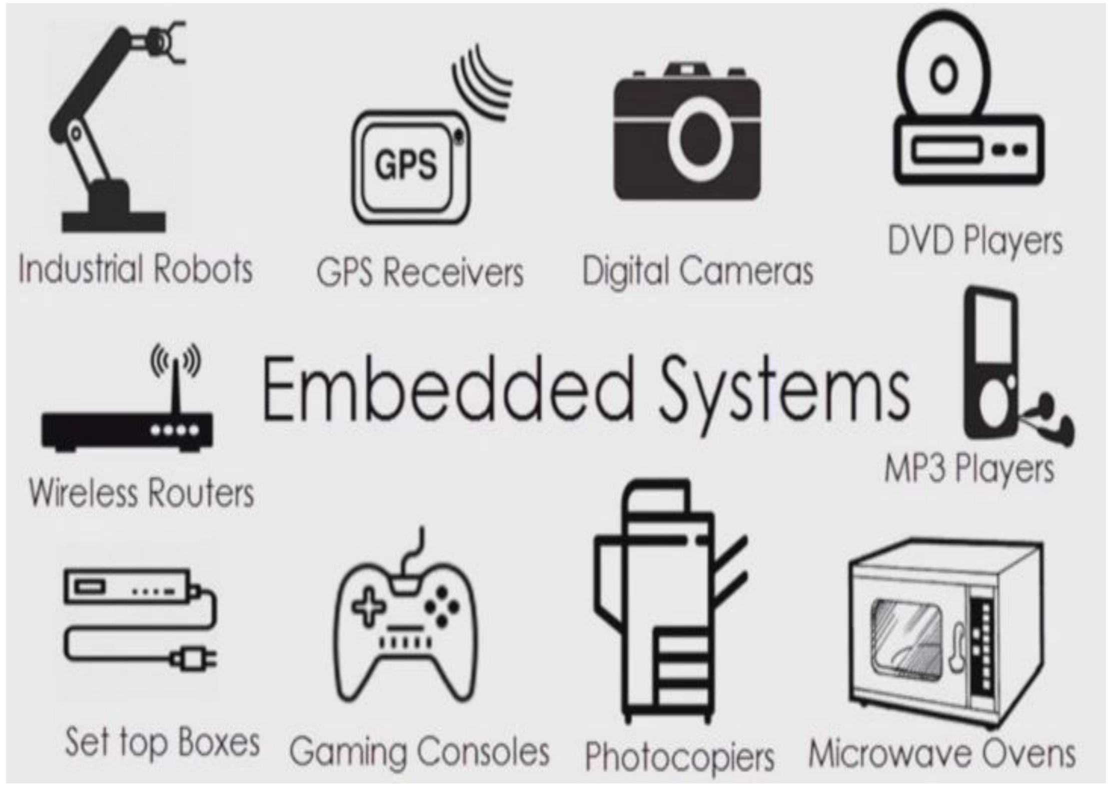
1.1 Introduction to System

- ❑ A way of working, organizing or performing one or many tasks according to a fixed set of rules, program or plan.
- ❑ Also an arrangement in which all units assemble and work together according to a program or plan.
- ❑ Some examples of Systems -
  - Time display system - A watch
  - Automatic cloth washing system – Washing Machine

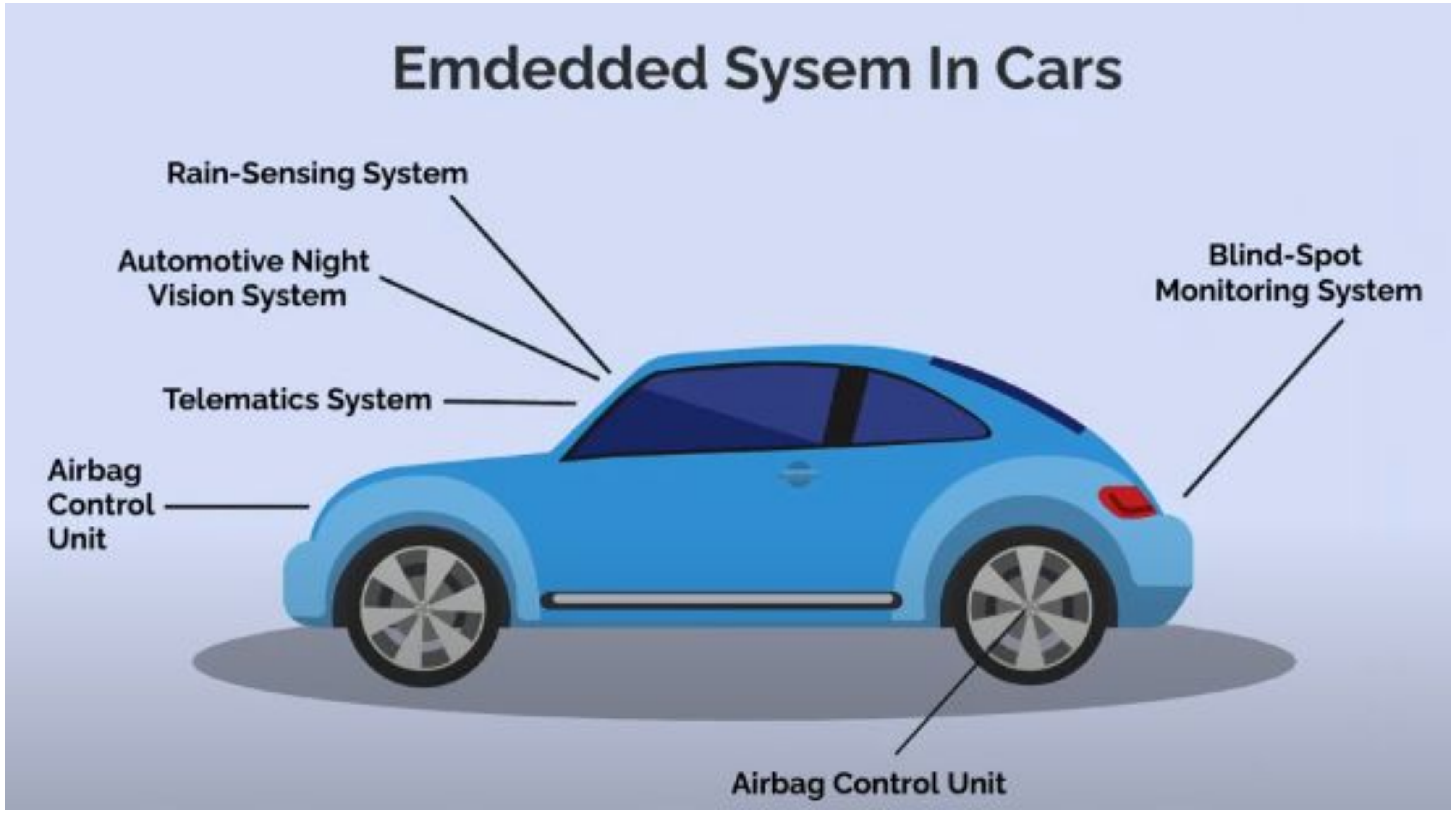


Introduction to Embedded System

- ❑ Computers are embedded within *other systems*.
  - What is *other systems* here? – Hard to define (Any other computing system other than desktop / laptop / servers)
- ❑ Designed to serve for dedicated purpose.
- ❑ In short, An Embedded System is a microcontroller based system with a dedicated function within a large system.
- ❑ Contains Firmware (only the **needed software** which is not intended to be changed frequently.)
- ❑ May contain RTOS (works as a real time task scheduler)



1.2 Introduction to Embedded System



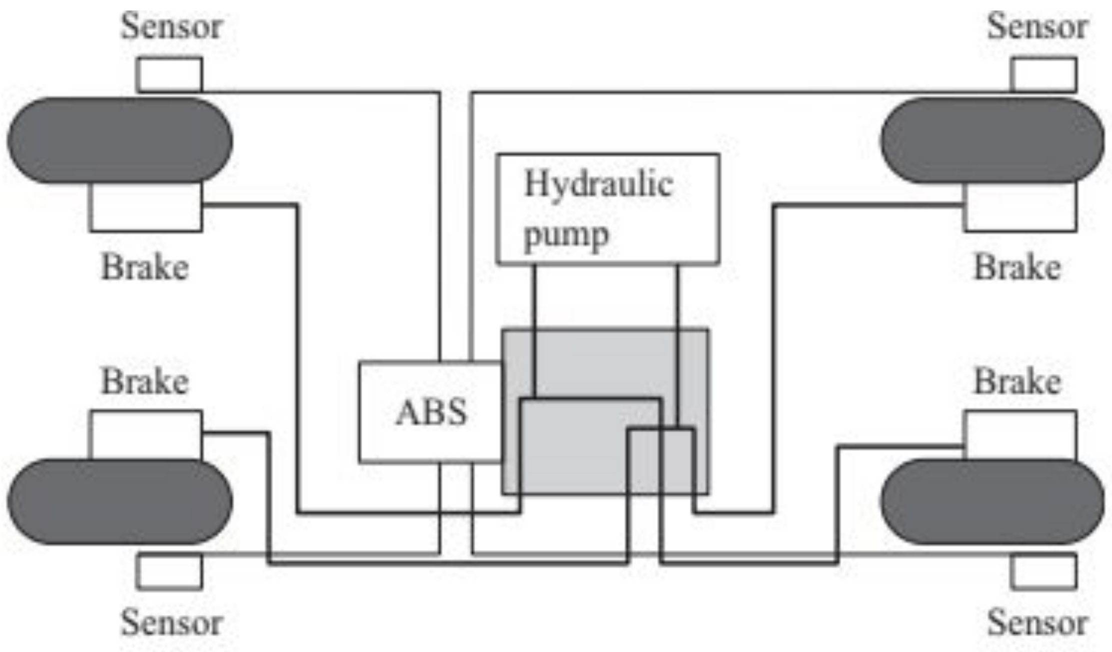
1.2.1 An Example: BMW 850i Brake and Stability Control System

- ❖ An antilock brake system (ABS) reduces skidding by pumping the brakes.

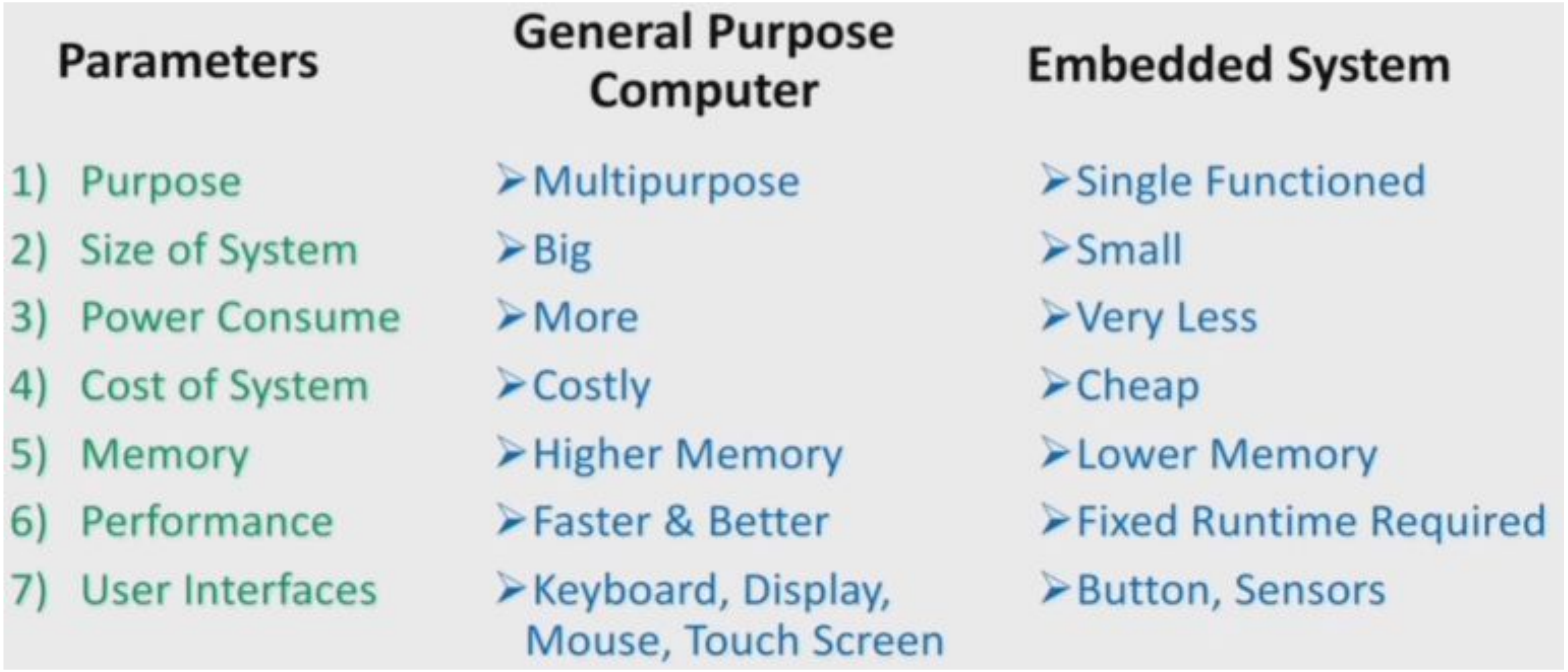
- ❖ An automatic stability control & traction (ASC+T) system intervenes with the engine during maneuvering to improve the car's stability.

- ❖ The purpose of an ABS is to temporarily release the brake on a wheel so that the wheel does not lose traction and skid. It sits between the hydraulic pump, which provides power to the brakes, and the brakes themselves.

- ❖ The ABS system uses sensors on each wheel to measure its speed of rotation.

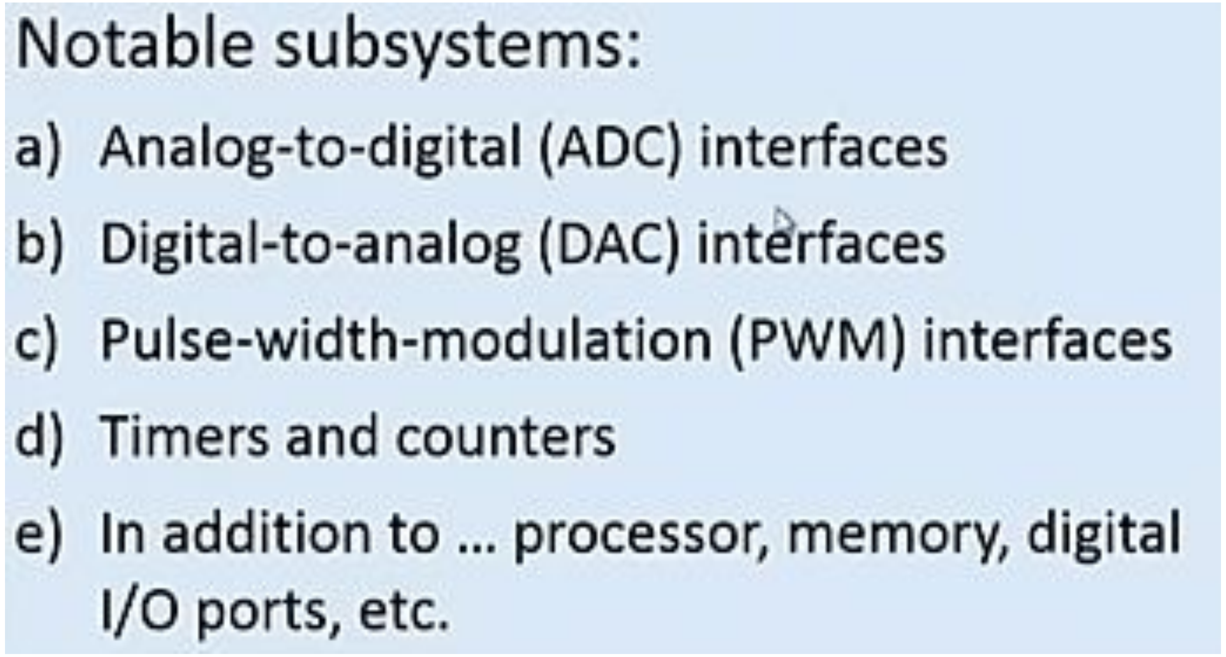


1.2.2 General-purpose Computer vs Embedded System



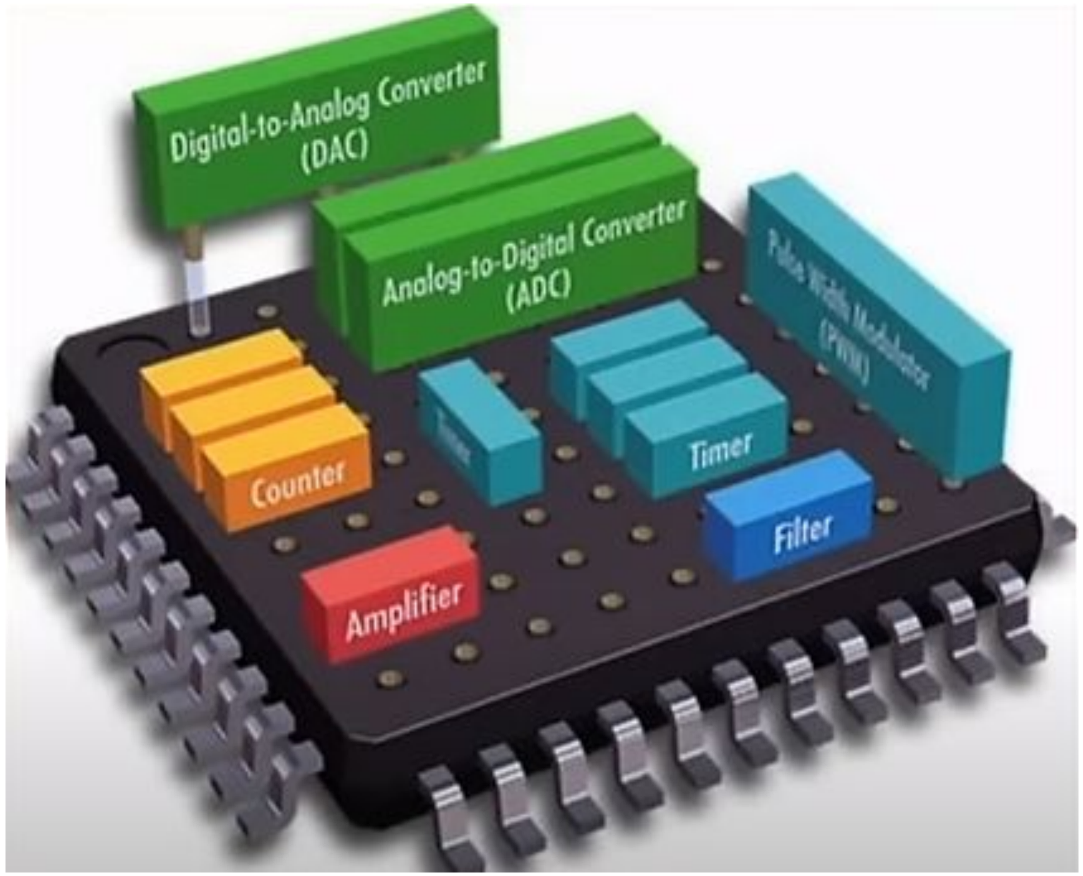
Parameters	General Purpose Computer	Embedded System
1) Purpose	➢Multipurpose	➢Single Functioned
2) Size of System	➢Big	➢Small
3) Power Consume	➢More	➢Very Less
4) Cost of System	➢Costly	➢Cheap
5) Memory	➢Higher Memory	➢Lower Memory
6) Performance	➢Faster & Better	➢Fixed Runtime Required
7) User Interfaces	➢Keyboard, Display, Mouse, Touch Screen	➢Button, Sensors

Notable Subsystems



Notable subsystems:

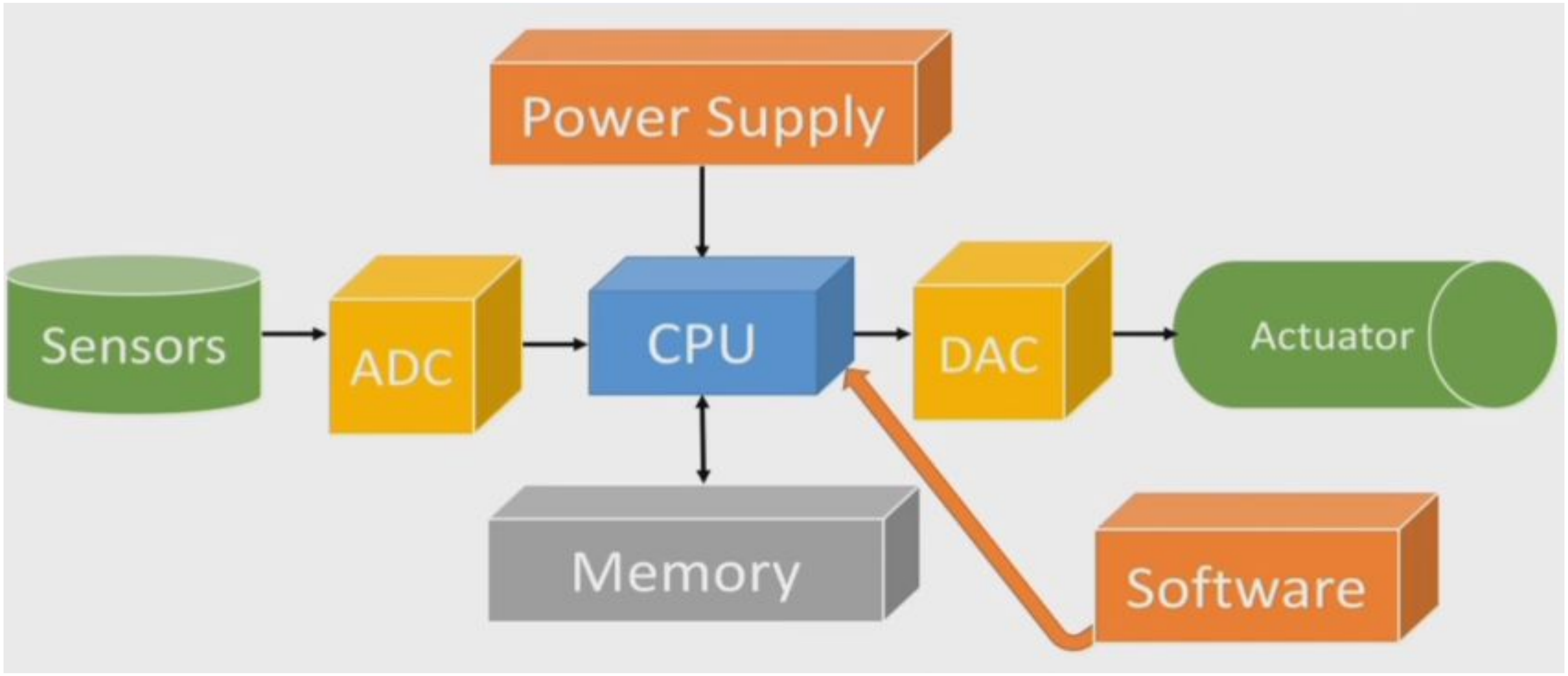
a) Analog-to-digital (ADC) interfaces
b) Digital-to-analog (DAC) interfaces
c) Pulse-width-modulation (PWM) interfaces
d) Timers and counters
e) In addition to … processor, memory, digital I/O ports, etc.


Digital-to-Analog Converter (DAC)
Analog-to-Digital Converter (ADC)
Pulse Width Modulator (PWM)
Timer
Timer
Counter
Filter
Amplifier

1.2.2 Characteristics of Embedded Computing Applications

- ❑ Complex algorithms
- ❑ User interface
- ❑ Real time
- ❑ Multi-rate
- ❑ Manufacturing cost
- ❑ Power and energy
- ❑ Size

Detailed Block Diagram of Embedded System



1.2.3 Why use Microprocessors?

There are many ways to design a digital system: custom logic, field-programmable gate arrays (FPGAs), and so on.

Why use microprocessors?

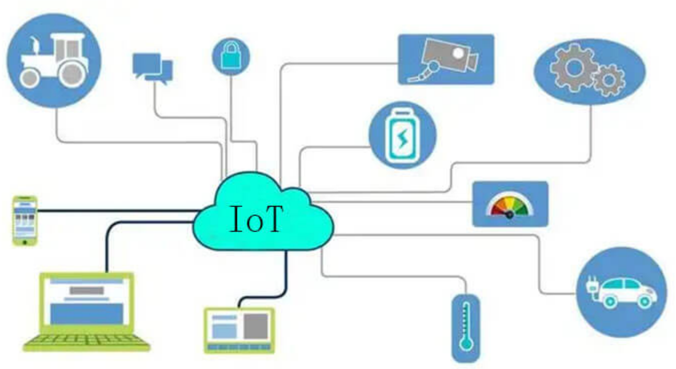
There are two answers:

- ❖ Microprocessors are a very efficient way to implement digital systems.
- ❖ Microprocessors make it easier to design families of products that can be built to provide various feature sets at different price points and be extended to provide new features to keep up with rapidly changing markets.

CPUs are flexible, efficient, and highly optimized.

1.2.4 IoT systems, and Cyber-Physical Systems

An ***IoT system*** is a set of sensors, actuators, and computation units connected by a network. The network often has wireless links but may also include wired links. The IoT system monitors, analyzes, evaluates, and acts.



Examples of IoT systems: Connected Cars, Connected homes, Smart Cities, and Smart Buildings etc.

❑ A typical IoT system works through the real-time collection and exchange of data. *An IoT system has three components:*

✔ **Smart devices :** This is a device, like a television, security camera, or exercise equipment that has been given computing capabilities. It collects data from its environment, user inputs, or usage patterns and communicates data over the internet to and from its IoT application.

✔ **IoT application :** An IoT application is a collection of services and software that integrates data received from various IoT devices. It uses machine learning or artificial intelligence (AI) technology to analyze this data and make informed decisions. These decisions are communicated back to the IoT device and the IoT device then responds intelligently to inputs.

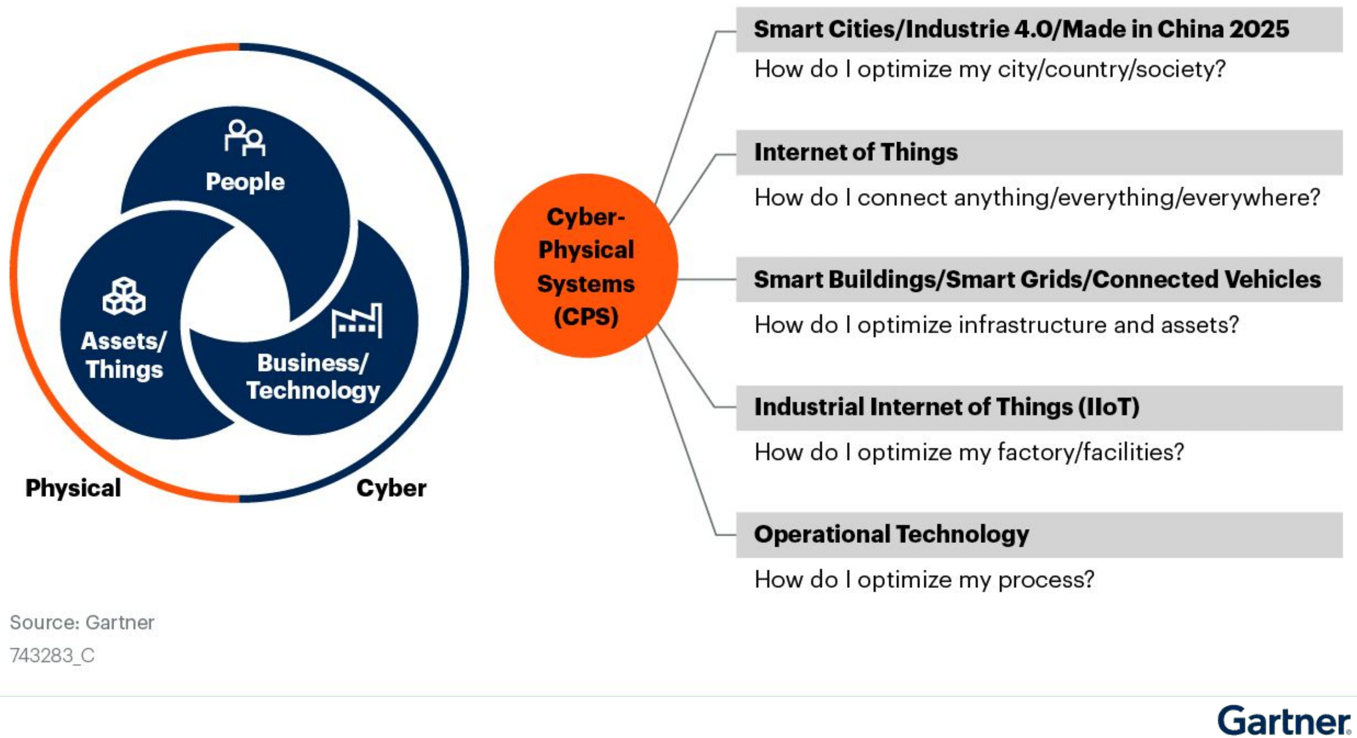
✔ **A graphical user interface :** The IoT device or fleet of devices can be managed through a graphical user interface. Common examples include a mobile application or website that can be used to register and control smart devices.

1.2.4 IoT systems, and Cyber-Physical Systems

A ***cyber-physical system*** uses computers to build controllers, such as feedback control systems. A networked control system, in which a set of interfaces of a physical machine communicates over a bus with a CPU, is an important example of a cyber-physical system.

Cyber-Physical Systems (CPS) are systems composed of *physical systems* (hardware), *software systems* and potentially other types of systems (e.g., human systems). These are closely integrated and networked to deliver some global behavior.

CPS Security Underpins Families of Interconnected Efforts


People
Assets/ Things
Business/ Technology
Physical
Cyber
Cyber-Physical Systems (CPS)
Smart Cities/Industrie 4.0/Made in China 2025
How do I optimize my city/country/society?
Internet of Things
How do I connect anything/everything/everywhere?
Smart Buildings/Smart Grids/Connected Vehicles
How do I optimize infrastructure and assets?
Industrial Internet of Things (IIoT)
How do I optimize my factory/facilities?
Operational Technology
How do I optimize my process?
Source: Gartner
743283_C
Gartner

1.2.4 IoT systems, and Cyber-Physical Systems

IoT	**CPS**
IoT systems tend to operate at lower sample rates	Cyber-Physical systems run at higher sample rate
IoT systems are often more physically distributed	CPSs are more tightly coupled
Example: A manufacturing plant	Example: An airplane or automobile.

IoT and CPSs are both examples of ***edge computing***. When our computer must respond quickly to events in the physical world, we often don’t have time to send queries to a remote data center and wait for a response. Computing at the edge must be responsive and energy efficient. Edge devices must also communicate with other devices at the edge as well as cloud computing resources.

1.2.6 Challenges in Embedded Computing system design

1. **Limited Resources**: Embedded systems often operate with limited resources such as power, memory, processing capabilities, and storage. Designing efficient algorithms and optimizing resource usage are critical.

2. **Real-Time Constraints:** Many embedded systems are required to meet strict real-time constraints. This means that tasks must be completed within specific time bounds, necessitating careful consideration of timing and latency throughout the system.

3. **Power Consumption**: Power efficiency is a significant concern in embedded systems, especially for battery-powered devices or those deployed in remote locations. Minimizing power consumption without sacrificing performance is essential.

4. **Safety and Reliability**: Embedded systems are often used in safety-critical applications such as automotive, medical devices, and industrial control systems. Ensuring the safety and reliability of these systems requires rigorous testing, fault tolerance, and compliance with industry standards.

5. **Security**: Embedded systems are vulnerable to various threats such as unauthorized access, data breaches, and malware. Implementing robust security measures is crucial to protect sensitive information and prevent cyberattacks.

1.3 The embedded system design process

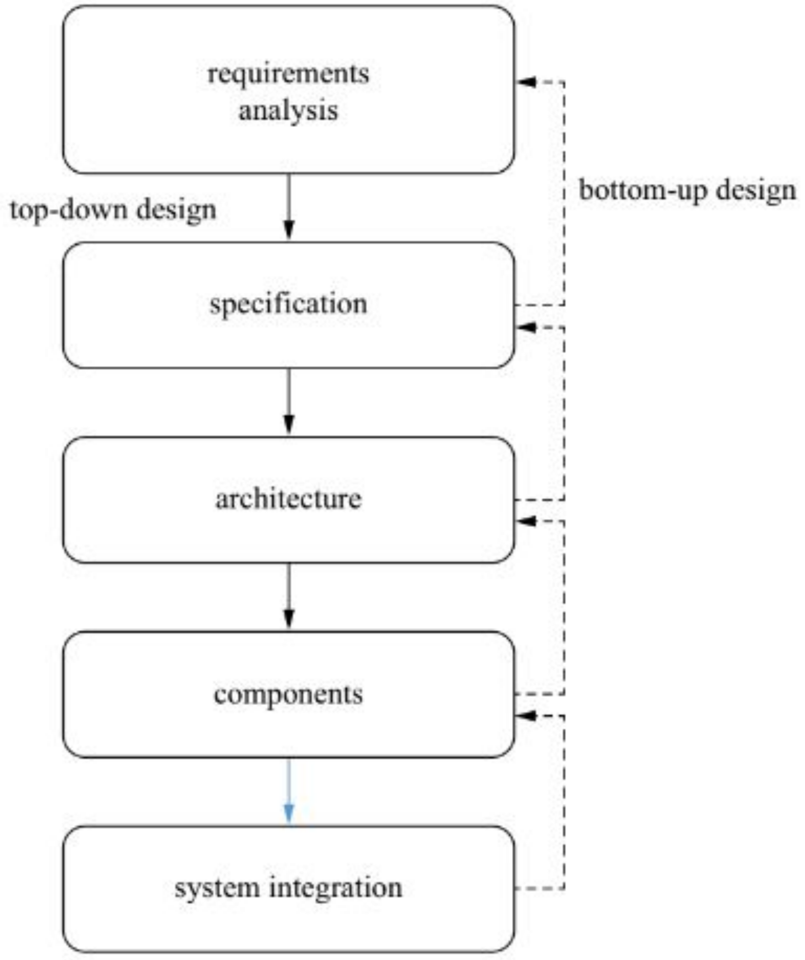


Fig. Major levels of abstraction in the design process.

- **System requirements analysis**: to capture some basic elements of the system.
- **Specification**: to create a more detailed, complete description of what we want. However, the specification states only how the system behaves, not how it is built.
- **Architecture** : The details of the system's internals begin to take shape, it gives the system structure in terms of large components.
- **Components:** to design both software modules and any specialized hardware.
- **System integration:** to build a complete system through.

Top-down: Begin with the most abstract description of the system and conclude with concrete details.
Bottom-up: Start with components to build a system. Bottom-up design is needed because we do not have perfect insight into how later stages of the design process will turn out.

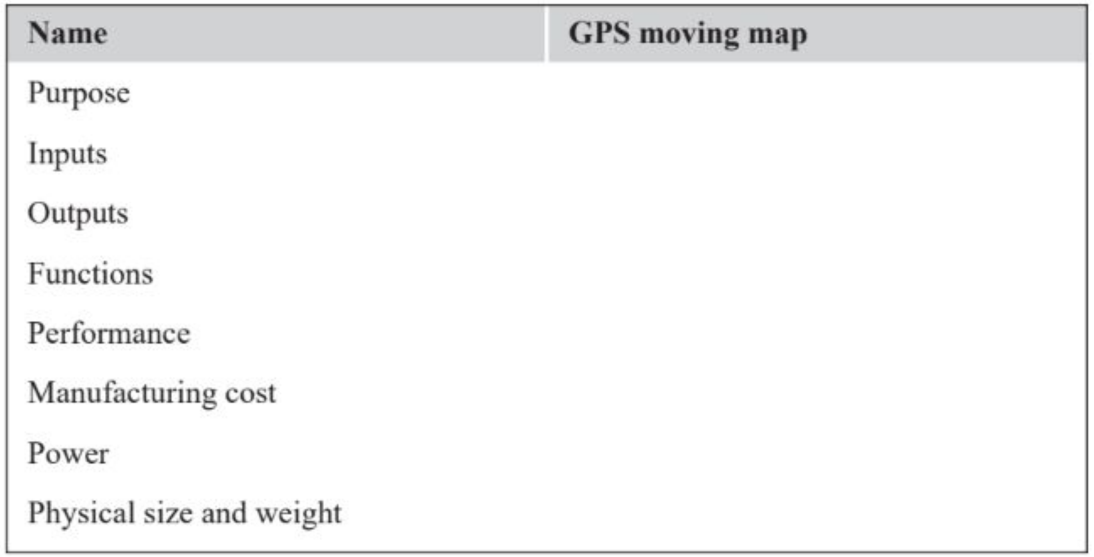
1.3.1 Requirements Analysis of a GPS Moving Map

- ✔ Before we design a system, we must know what we are designing.
- ✔ A requirement is a need for the system; a specification is a complete set of requirements for the system.

Requirements may be functional or nonfunctional. Functional requirements define how the system must work and non functional requirements detail how it should perform.

Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks. It defines what the system or product should do, how it should behave, and what constraints it should follow.

❑ Typical nonfunctional requirements include:
- Performance
- Cost
- Physical Size and Weight
- Power Consumption



Name	GPS moving map
Purpose	
Inputs	
Outputs	
Functions	
Performance	
Manufacturing cost	
Power	
Physical size and weight	

Fig. Sample requirements form that can be filled out at the start of the project.

1.3.1 Requirements Analysis of a GPS Moving Map

What requirements might we have for our GPS moving map? Here is an initial list:

• **Functionality**: This system is designed for highway driving and similar uses, not nautical or aviation uses that require more specialized databases and functions. The system should show major roads and other landmarks available in standard topographic databases.

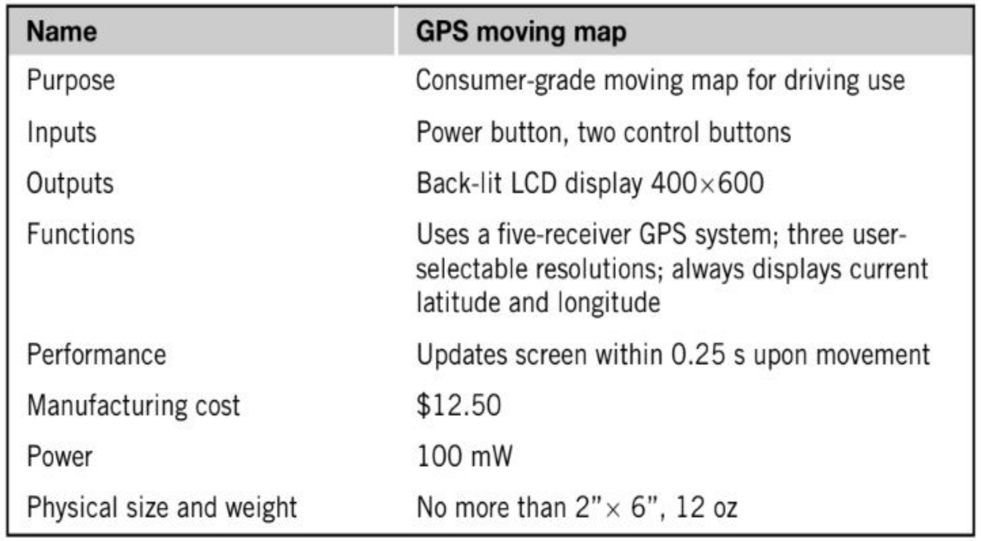
• **User interface**: The screen should have at least 400 600 pixel resolution. The device should be controlled by no more than three buttons. A menu system should pop up on the screen when buttons are pressed, to allow the user to make selections to control the system.

• **Performance:** The map should scroll smoothly with screen updates at least 10 frames per second. Upon power-up, a display should take no more than 1 s to appear, and the system should be able to verify its position and display the current map within 15 s.

• **Cost:** The selling cost (street price) of the unit should be no more than $50. Selling price in turn helps to determine the allowable values for manufacturing cost. For example, if we use a rule-of-thumb in which the street cost is 4 that of the manufacturing cost, the cost to build this device should be no more than $12.50. Selling price also influences the amount of money that should be spent on nonrecurring engineering costs. High NRE costs will not be recoverable if the profit on each device is low.

• **Physical size and weight:** The device should fit comfortably in the palm of the hand.

• **Power consumption:** The device should run for at least 8 h on four AA batteries with at least 30 min of those 8 h comprising operation with the screen on.



Name	GPS moving map
Purpose	Consumer-grade moving map for driving use
Inputs	Power button, two control buttons
Outputs	Back-lit LCD display 400×600
Functions	Uses a five-receiver GPS system; three user-selectable resolutions; always displays current latitude and longitude
Performance	Updates screen within 0.25 s upon movement
Manufacturing cost	$12.50
Power	100 mW
Physical size and weight	No more than 2"× 6", 12 oz

Fig. The Requirement Chart

1.3.2 Specification of a GPS Moving Map

The specification is more precise and complete; it serves as the contract between the customer and the architects.

The specification should be understandable enough so that someone can verify that it meets the system requirements and overall expectations of the customer.

- **A specification of the GPS system would include several components**:
  - ✓ data received from the GPS satellites;
  - ✓ map data;
  - ✓ user interface;
  - ✓ operations that must be performed to satisfy customer requests;
  - ✓ background actions required to keep the system running, such as operating the GPS receiver.

1.3.3 Architecture Design of a GPS Moving Map

- ✔ The specification does not say how the system does things, only what the system does.
- ✔ Describing how the system implements those functions is the purpose of the ***architecture***.
- ✔ **The architecture is a plan for the overall structure of the system that will be used later to design the components that make up the architecture.**

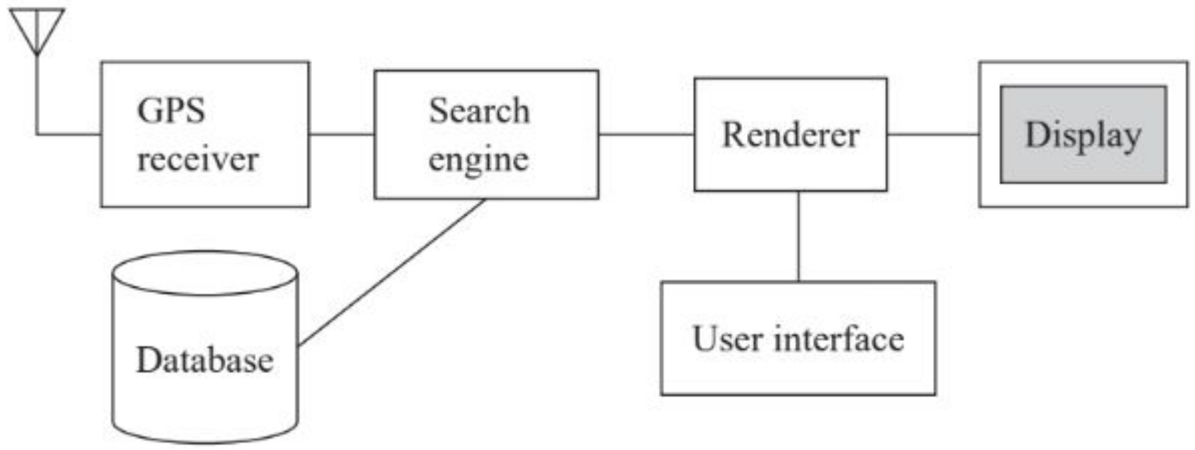


Fig. Block diagram for the moving map.

1.3.4 Designing hardware and software components of a GPS Moving Map

❖ Architecture design is still quite abstract; we haven’t yet specified which operations will be performed by software running on a CPU, what will be done by special purpose hardware.

❖ We need to refine the system block diagram into two block diagrams: one for hardware and another for software.

- The hardware block diagram clearly shows that we have one CPU surrounded by memory and I/O devices. In particular, we have chosen to use two memories: a frame buffer for the pixels to be displayed and a separate program/data memory for general use by the CPU.
- The software block diagram follows the system block diagram, but we have added a timer to control when we read the buttons on the user interface and render data onto the screen.

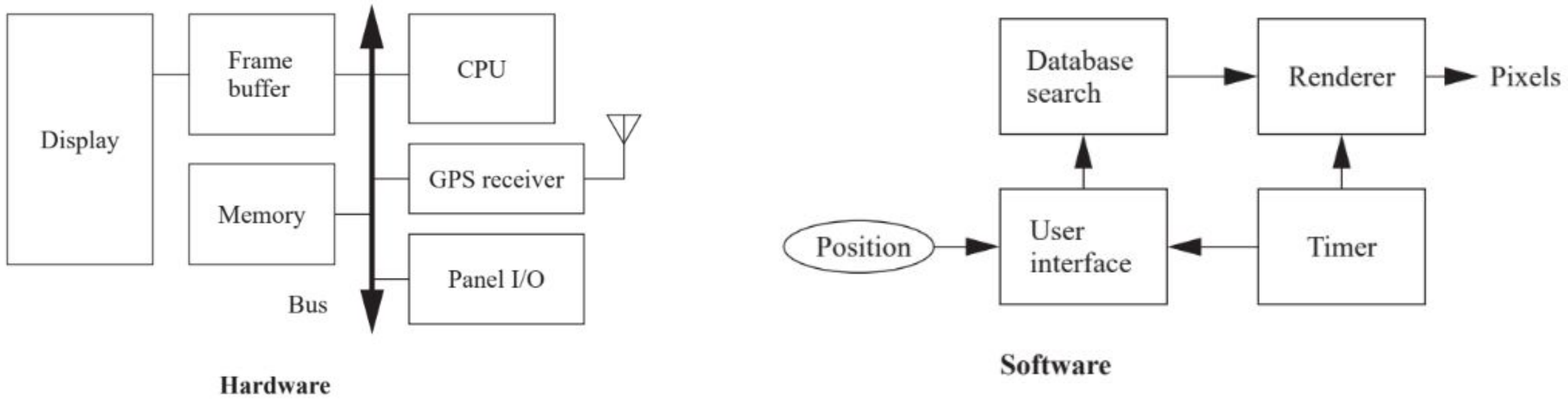


Fig. Hardware and software architectures for the moving map .

UML – Structural & Behavioral Descriptions

UML is an object-oriented modeling language.

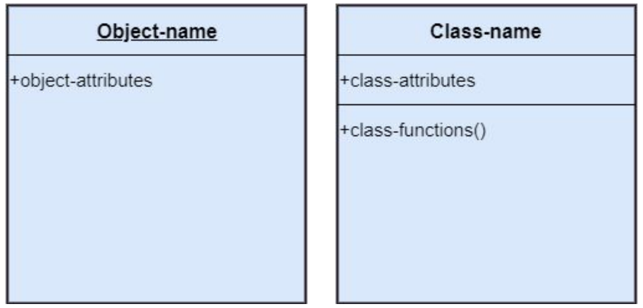
It is often helpful to conceptualize the tasks like creating requirements and specifications, architecting the system, designing the code, and designing the tests as diagrams. UML was designed to be useful at many levels of abstraction in the design process.

Structural diagrams or descriptions in UML (Unified Modeling Language) represent the static structure of a system. They depict the components of the system and their relationships. Structural diagrams are used to visualize the organization, composition, and relationships between the different elements in a system.

Behavioral diagrams or descriptions in UML represent the dynamic behavior of a system. They describe how the components of a system interact with each other and how the system responds to external stimuli. Behavioral diagrams are used to visualize the behavior of a system over time.

1.3.6 - 1.3.7 UML – Structural Descriptions

- ❖ The principal component of an object-oriented design is, naturally enough, the ***object***. *It describes the behavior and the functions of a system.*
- ❖ An object includes a set of ***attributes*** that define its internal state. When implemented in a programming language, these attributes usually become variables or constants held in a data structure. In some cases, we will add the *type* of the attribute after the attribute name for clarity.
- ❖ The object is identified in two ways: It has a unique name, and it is a member of a ***class*****. The name is underlined to show that this is a description of an object and not of a class.** *A Class is a set of identical things that outlines the functionality and properties of an object.*
- ❖ The text in the folded corner page icon is a ***note***; it does not correspond to an object in the system and only serves as a comment.



+ = public
- = private
# = protected
~ = package

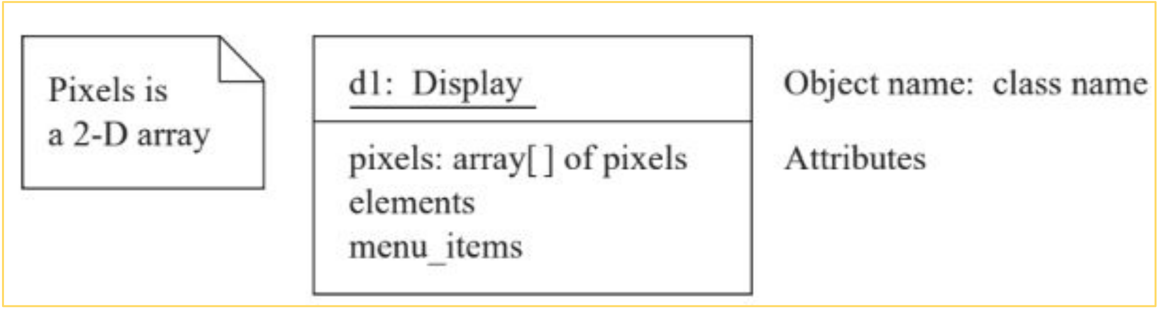


Fig. An object in UML notation.

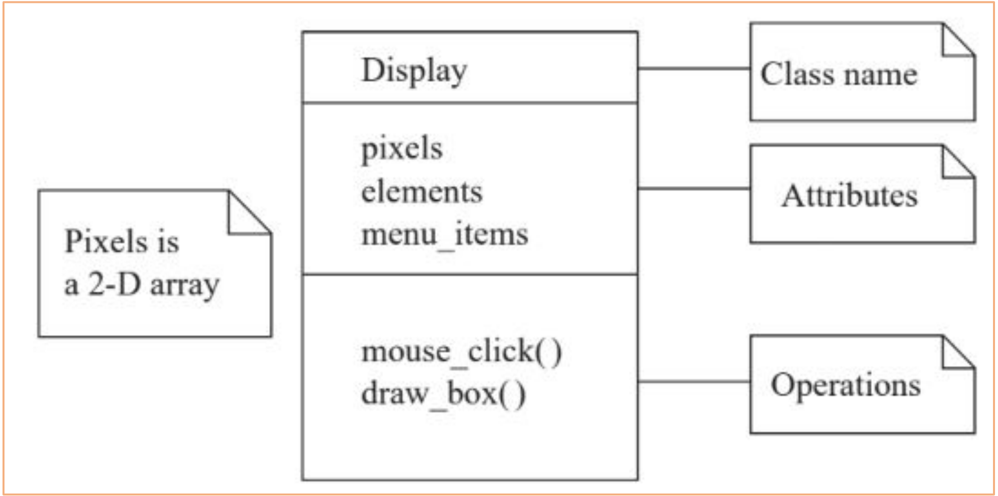
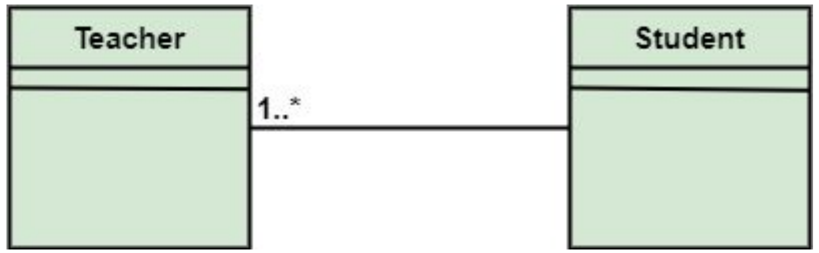


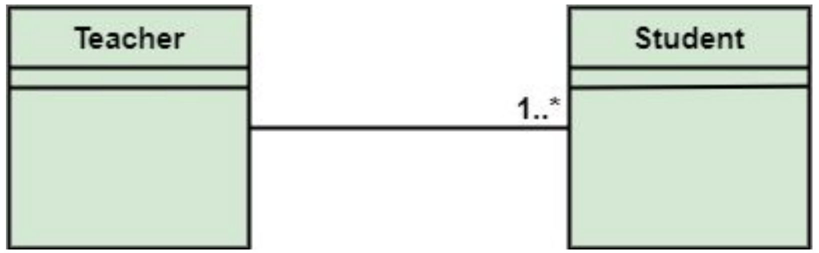
Fig. A class in UML notation.

1.3.6 - 1.3.7 UML – Structural Descriptions

- ❑ There are several types of relationships that can exist between objects and classes:
- ▯ **Association** occurs between objects that communicate with each other but have no ownership relationship between them.
- ▯ **Aggregation** describes a complex object made of smaller objects.
- ▯ **Composition** is a type of aggregation in which the owner does not allow access to the component objects.
- ▯ **Generalization** allows us to define one class in terms of another.



A single student can associate with many teachers.

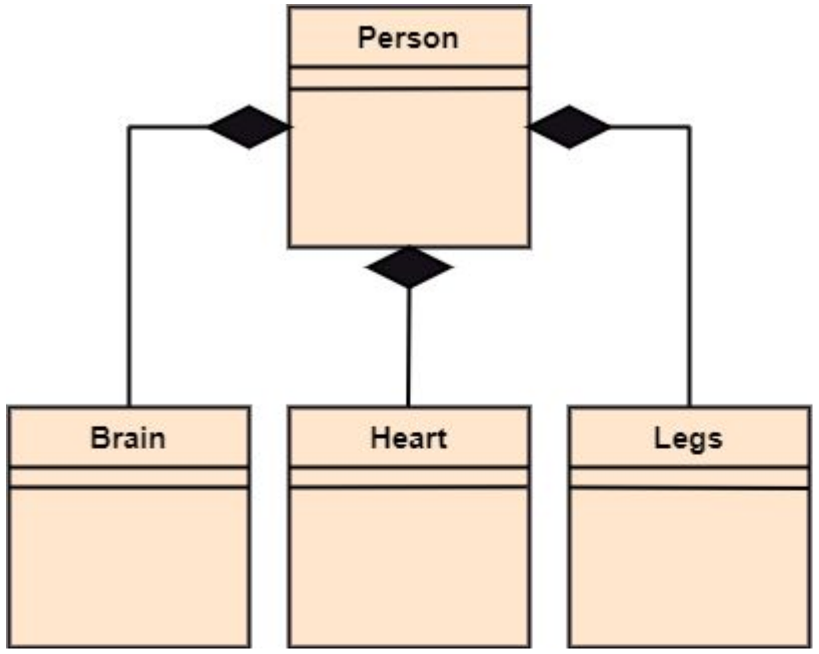


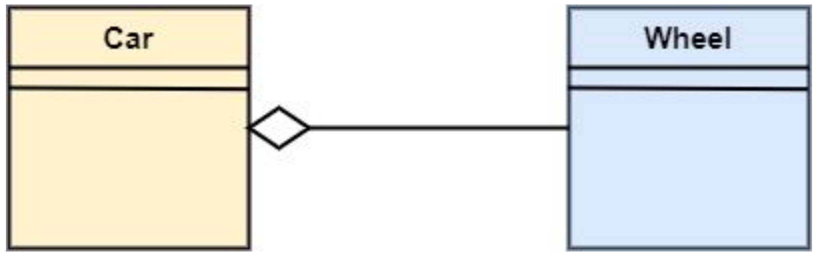
A single teacher has multiple students.

Association relationship is a structural relationship in which different objects are linked within the system. It exhibits a binary relationship between the objects representing an activity.

1.3.6 - 1.3.7 UML – Structural Descriptions

Aggregation is a subset of association, is a collection of different things. It represents has a relationship. Here we are considering a car and a wheel example. A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation relationship.

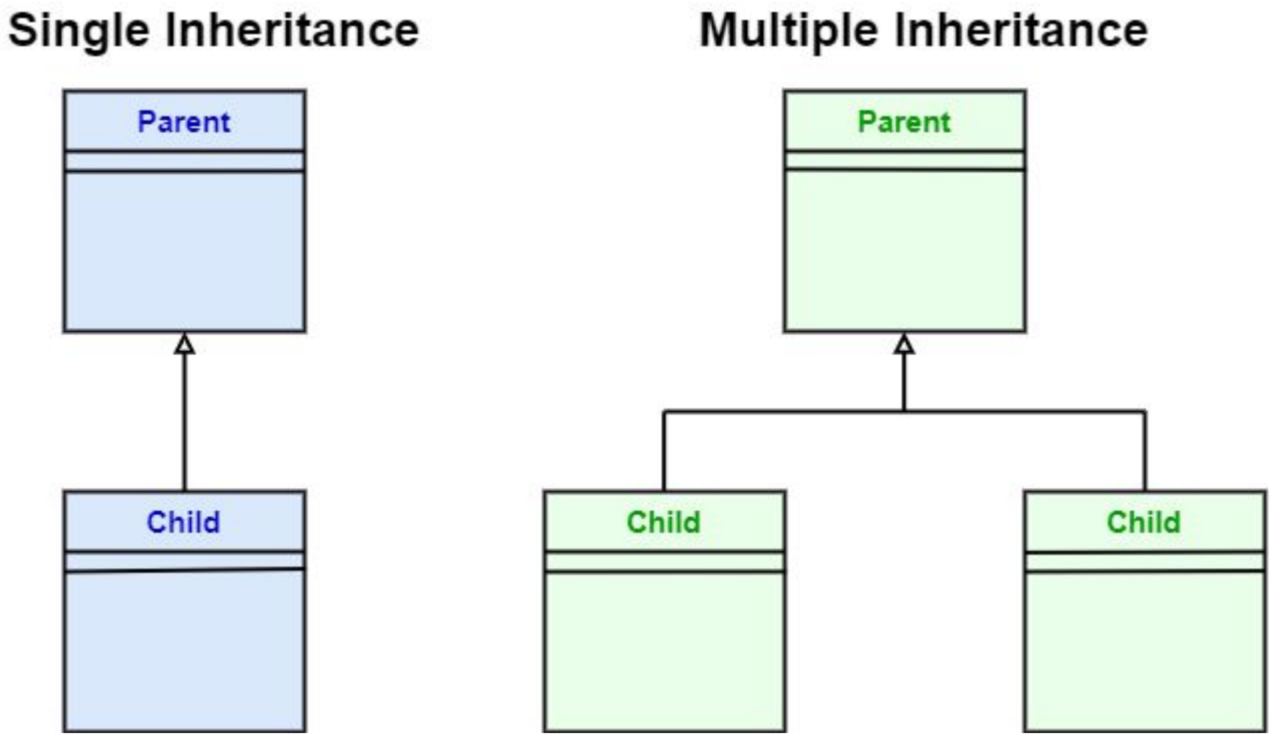




The composition is a part of aggregation, and it portrays the whole-part relationship. It depicts dependency between a composite (parent) and its parts (children), which means that if the composite is discarded, so will its parts get deleted. In the example above, the composition association relationship connects the Person class with Brain class, Heart class, and Legs class. If the person is destroyed, the brain, heart, and legs will also get discarded.

1.3.6 - 1.3.7 UML – Structural Descriptions

In UML modeling, a generalization relationship is a relationship that implements the concept of object orientation called inheritance. The generalization relationship occurs between two entities or objects, such that one entity is the parent, and the other one is the child. The child inherits the functionality of its parent and can access as well as update it.



1.3.6 - 1.3.7 UML – Structural Descriptions

A derived class inherits all the attributes and operations from its base class.

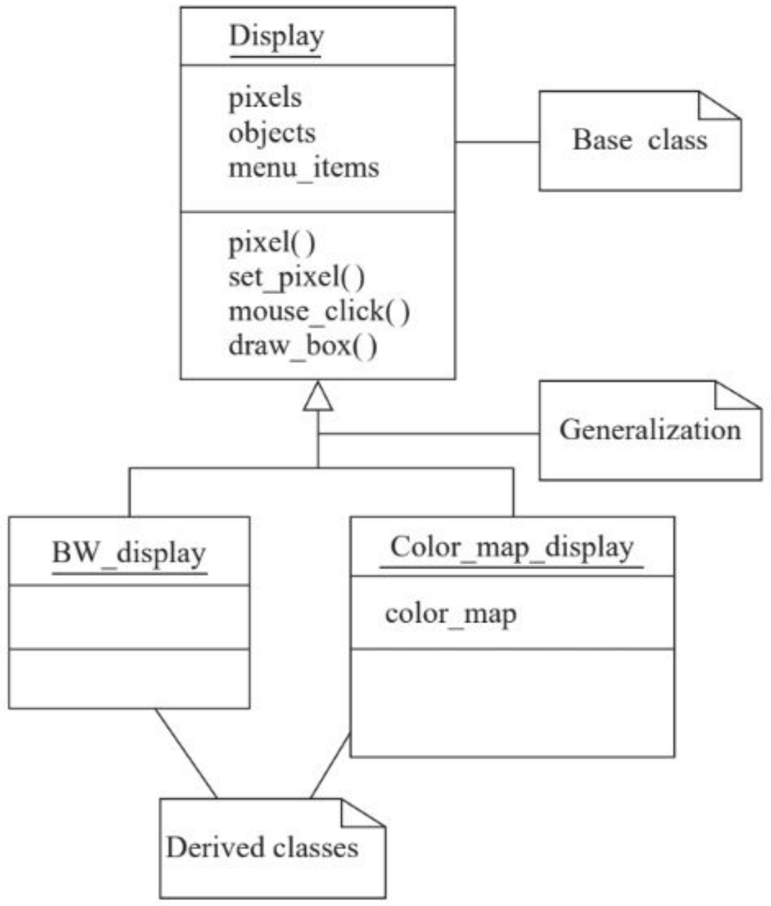


Fig. Derived classes as a form of generalization in UML notation.

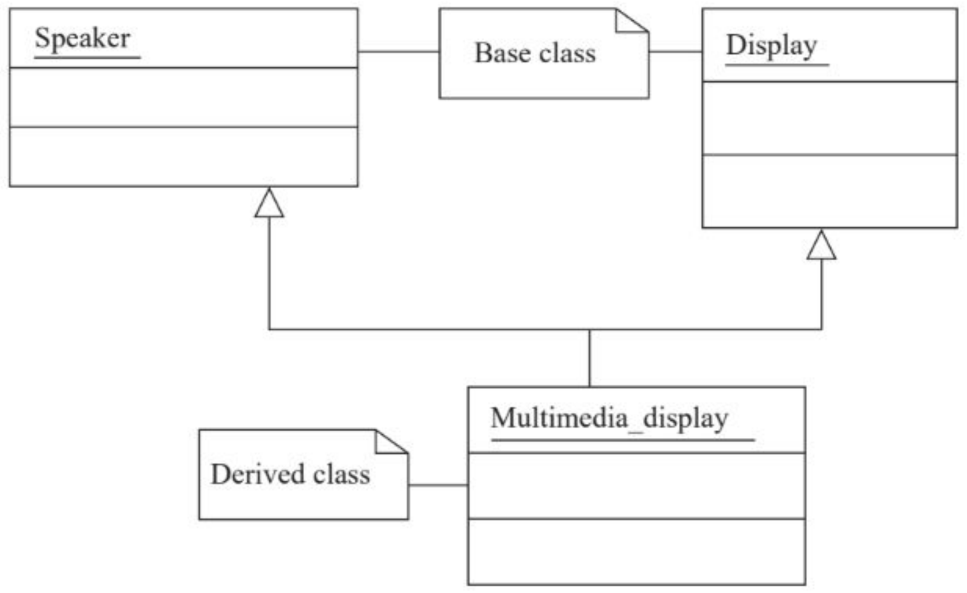


Fig. Multiple inheritance in UML notation.

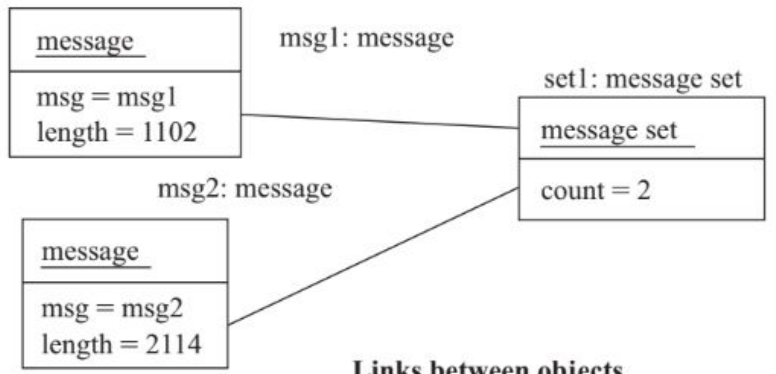
1.3.6 - 1.3.7 UML – Structural Descriptions

A link describes a relationship between objects; association is to link as class is to object. We need links because objects often do not stand-alone; associations let us capture type information about these links.

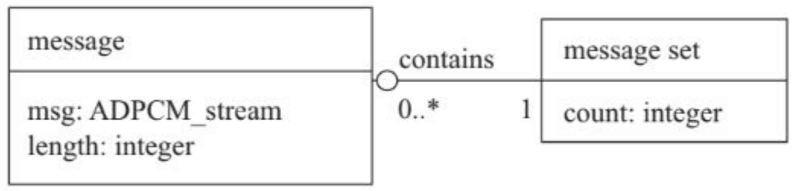
Here in the right side example,

- There is a set of messages that keeps track of the current number of active messages (two in this example) and points to the active messages.
- The link defines the *contains* relation.
- When generalized into classes, we define an association between the message set class and the message class.
- The association is drawn as a line between the two labeled with the name of the association: contains.
- The ball and the number at the message class end indicate that the message set may include zero or more message objects.
- Sometimes we may want to attach data to the links themselves; we can specify this in the association by attaching a class-like box to the association's edge, which holds the association's data.

Typically, we find that we use a certain combination of elements in an object or class many times. We can give these patterns names, which are called stereotypes in UML. A stereotype name is written in the form ***<< signal >>***.



Links between objects

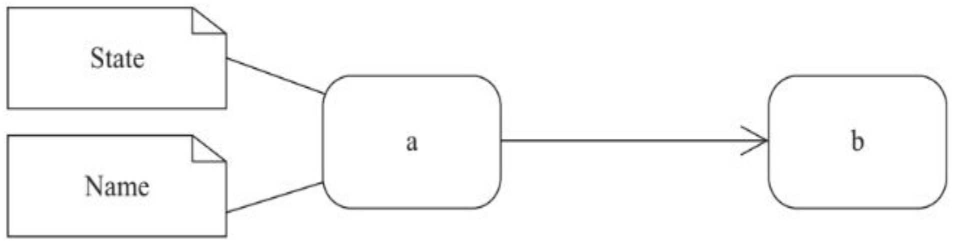


Association between classes

Fig. Links and associations.

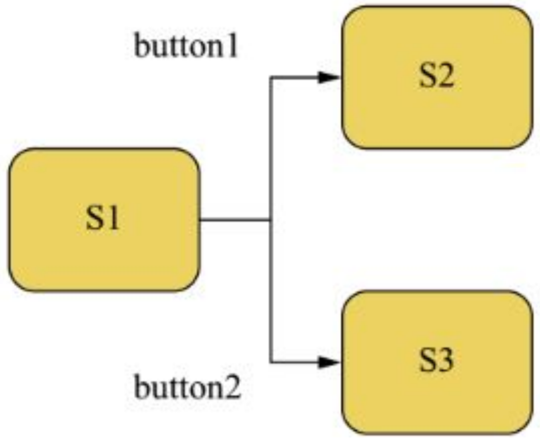
1.3.8 UML – Behavioral Descriptions

- We must specify the behavior of the system as well as its structure. One way to specify the behavior of an operation is a **state machine**.
- The transition between two states is shown by a *skeleton arrow.*
- Changes from one state to another are triggered by the occurrence of **events**. An event is some type of action.



A state and transition in UML notation.

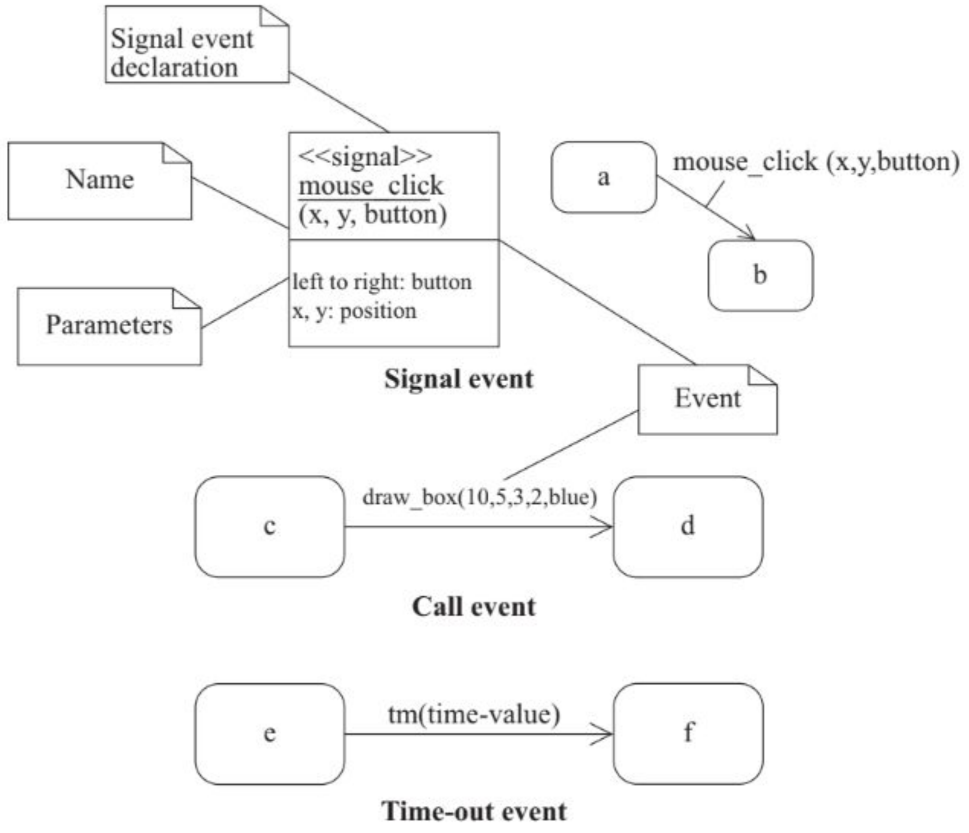
The machine transitions from S1 to S2 or S3 only when button1 or button2 is pressed.



Events in UML state machines.

1.3.8 UML – Behavioral Descriptions

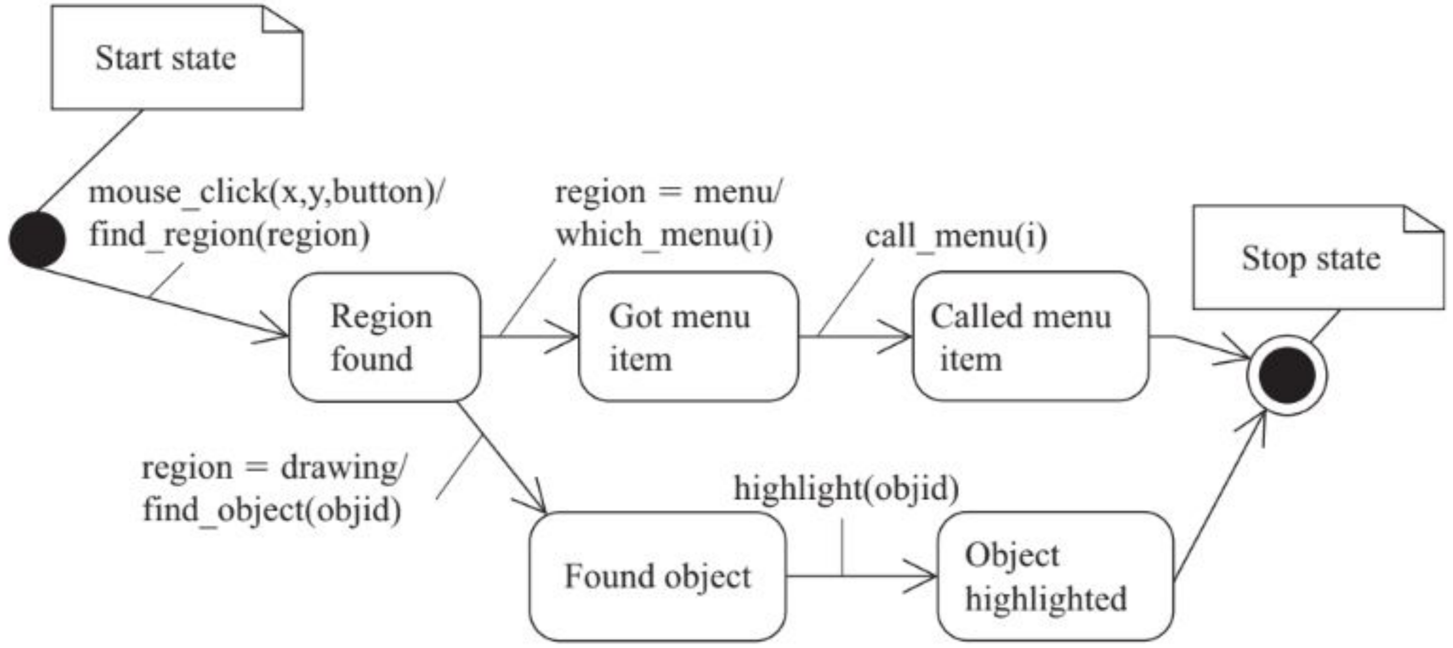
- UML defines several special types of events,
    - A **signal** is an asynchronous occurrence. It is defined in UML by an object that is labeled as a «signal». The object in the diagram serves as a declaration of the event’s existence.
    - A **call** event follows the model of a procedure call in a programming language.
    - A **time-out** event causes the machine to leave a state after a certain amount of time. The label tm(time-value) on the edge gives the amount of time after which the transition occurs. A time-out is generally implemented with an external timer.



Signal, call, and time-out events in UML notation.

1.3.8 UML – Behavioral Descriptions

- ✔ Let's consider a simple state machine specification for the operation of the *display.*
- ✔ The start and stop states are special states that help us organize the flow of the state machine.
- ✔ The states in the state machine represent different conceptual operations. In some cases, we take conditional transitions out of states based on inputs or the results of some computation done in the state. In other cases, we make an unconditional transition to the next state.
- ✔ Both the unconditional and conditional transitions make use of the call event.



A state machine specification in UML notation.

References

Chapter 1 - Computers as Components 5th Ed - Principles of Embedded Computing System Design - Marilyn Wolf - MK (2022)

thank you!