UNIT-3
Part 2: Analysis and Design

PROCESS OVERVIEW

- Development Stages - System Conception, Analysis, System Design, Class Design, Implementation, Testing, Training, Deployment, Maintenance

- **System Design** - Overview of System Design, Estimating Performance, Making a Reuse Plan - Library, Framework Pattern`Breaking a System into Sub-systems - Layers, Partitions, Combining Layers and Partitions, Identifying Concurrency - Identifying, inherent Concurrency, Defining Concurrent Tasks, Allocation of Sub-Systems - Estimating hardware Resource Requirement, Making Hardware and Software Trade-offs, Allocating Tasks to processors, Determining Physical Connectivity, Management of Data Storage, Handling Global Resources, Choosing a Software Controlled Strategy - Procedure Driven Control, Event Driven Control, Concurrent Driven Control, Internal Control, Other Paradigms, Handling Boundary Conditions, Setting Trade-off Priorities, Common Architectural Styles - Batch Transformation, Continuous Transformation, Interactive Interface, Dynamic Simulation, Real-time System, Transaction Manage

- A software development process provides a basis for organized production of software, using a collection of predefined techniques and notations

Development stages

- System conception → Formulate tentative requirement

- Analysis → Understand the requirements by constructing models

- System design → Devise a high-level strategy for solving application problem

- Class design → Augment and adjust the real-world models from analysis so that they are amenable to the implementation

- Implementation → Translate the design into programming code and database structures

- Testing → Ensure that the application is suitable for actual use and that it truly satisfies the requirements
- Training → Help users master the new application
- Deployment → Place the application in the field
- Maintenance → Preserve the long-term viability of the application

System Conception

- System conception deals with the genesis of an application
- Initially somebody thinks of an idea for an application , prepares a business case and sells the idea to the organization
- The innovator must understand both business needs and technological capabilities

Analysis

- Analysis focuses on creation of models
- Analysts capture and scrutinize requirements by constructing models
- They specify what must be done
- During analysis, developers consider the available sources of information and resolve ambiguities
- Modeling quickens the convergence between developers and business experts, because it is much faster to work with multiple iterations of models than with multiple implementations of code

- There are two sub stages of analysis
  - Domain Analysis
  - Application Analysis
- Domain Analysis focuses on real-world things whose semantics the application captures
  - Example: Airplane Flight
- Application Analysis addresses the computer aspects of the application that are visible to users
  - Example: Flight reservation screen is part of a flight reservation system

System Design

- During system design, the developer makes strategic decisions with broad consequences
- An architecture will be formulated by choosing global strategies and policies to guide the portion of design
- The architecture is the high level plan or strategy for solving the application problem
- The choice of architecture is based on the requirements as well as past experience

Class design

- During class design, the developer expands and optimizes analysis models
- There is a shift in emphasis from application concepts toward computer concepts
- Developers choose algorithms to implement major system functions that must be suitable for particular programming languages

Implementation

- Implementation is the stage for writing the actual code
- Developers map design elements to programming language and database code

Testing

- In this stage, the tester revisit the original business requirements and verify that the system delivers the proper functionality
- Testing can also uncover accidental errors that have been introduced
- If an application runs on multiple hardware and operating system platforms, it should be tested on all of them
- Developers should check a program at several levels
  - Unit testing- methods or entire classes
  - System testing- major subsystem or entire application

Training

- An organization must train users so that they can fully benefit from an application
- Training accelerates users on the software learning curve

Deployment

- The system must work in the field , on various platforms and in various configuration
- Unexpected interaction can occur when a system is deployed in a customer environment

Maintenance

- Once development is complete and a system has been deployed, it must be maintained for continued success
- Bugs that remain in the original system will gradually appear during use and must be fixed
- Model ease maintenance and transitions across staff changes

System Design

Overview of system design

- System design is the first stage for devising the basic approach to solving the problem
- The system architecture determines the organization of the system into subsystems
- There are following stages
  - Estimate system performance
  - Make a reuse plan
  - Organize the system into subsystems
  - Identify concurrency inherent in the problem
  - Allocate subsystems to hardware
  - Manage data stores
  - Handle global resources

- Choose a software control strategy
- Handle boundary condition
- Set trade-off priorities
- Select an architectural style

Estimating performance

- Early in the planning for a new system we should prepare a rough performance estimate
- Purpose - To determine if the system is feasible
- To make simplifying assumptions
- Need not worry about details- approximate, estimate, and guess

- Example : ATM transaction
  - Suppose–The bank has 40 branches,
  - also 40 terminals
  - On a busy day half the terminals are busy at once.
  - Each customer takes one minute to perform a session
  - A peak requirement of about 40 transactions a minute.
  - Storage
  - Count the number of customers
  - Estimate the amount of data for each customer

Making a Reuse Plan

- There are two very different aspects of reuse
  - Using existing things
  - Creating reusable new things
- Reusable things include
  - Models
  - Libraries
  - Frameworks
  - Patterns

Libraries

- A library is a collection of classes that are useful in many contexts
- The collection of classes must be carefully organized, so that users can find them.
- Qualities of “Good” class libraries
  - Coherence : Well focused themes
  - Completeness : provide complete behaviour
  - Consistency : polymorphic operations should have consistent names and signatures across classes
  - Efficiency–provide alternative implementations of algorithms
  - Extensibility–define subclasses for library classes
  - Genericity–parameterized class definitions

Problems limit the reuse ability

- Argument validation : Validate arguments by collection or by individual
- Error Handling: Error codes or errors
- Control paradigms : Event-driven or procedure-driven control
- Group operations : Often inefficient and incomplete
- Garbage collection : class libraries use different strategies to manage memory allocation and avoid memory leaks
- Name collision : class names, public attributes , and public methods lie within a global name space

Frameworks

- A Framework is a skeletal structure of a program that must be elaborated to build a complete application
- A class library may accompany a framework so that the user can perform much of the specialization by choosing the appropriate subclasses rather than programming subclass behavior from scratch
- Frameworks class libraries are typically application specific and not suitable for general use.

Patterns

- A pattern is a proven solution to a general problem
- Various patterns target different phases of the software development lifecycle
  - Analysis, architecture, design and implementation
- A pattern is more likely to be correct and robust than an untested, custom solution.
- There are many benefits of patterns
  - Pattern has been carefully considered by others and has already been applied to past problems

- Patterns are prototypical model fragments that distill some of the knowledge of experts.
- Pattern vs. Framework
  - A pattern is typically a small number of classes and relationships.
  - A framework is much broader in scope and covers an entire subsystem or application.
  - ATM example
    - Transaction
    - Communication line

Breaking a System into Subsystem

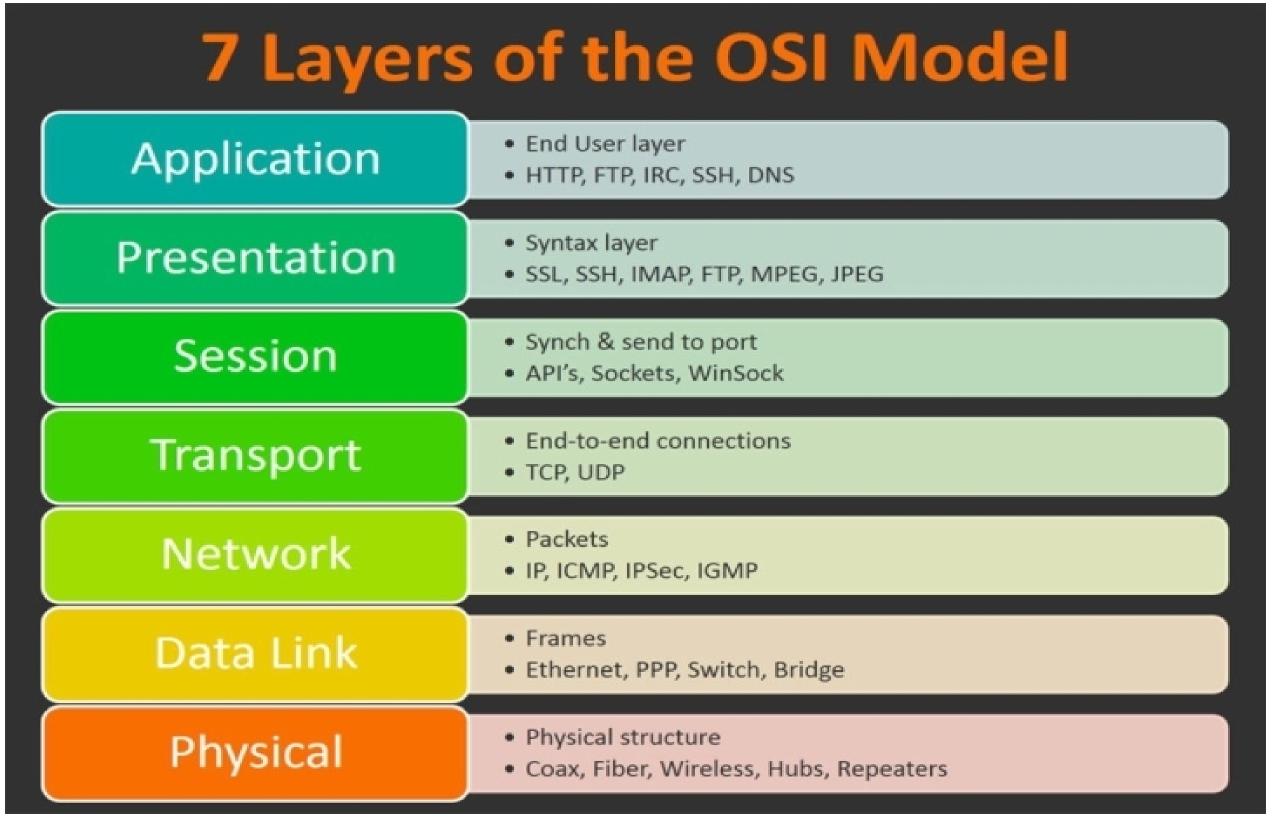
- Each subsystem is based on some common theme, such as
  - – Similar functionality
  - – The same physical location, or
  - – Execution on the same kind of hardware.

- A subsystem is a group of classes, associations, operations, events, and constrains .
- A subsystem is usually identified by the services it provides.
- Each subsystem has a well-defined interface to the rest of the system .
- The relation between two subsystems can be
  - – Client-server relationship
  - – Peer-to-peer relationship

- The decomposition of systems
  - Subsystems is organized as a sequence of
  - Horizontal layers,
  - Vertical partitions, or
  - Combination of layers and partitions.

Breaking a System into Subsystem - Layers

- Each built in terms of the ones below it and providing the implementation basis for the one above it.
- The objects in each layer can be independent.
  - E.g. – A client-server relationship
- Problem statement specifies only the top and bottom layers:
  - The top is the desired system.
  - The bottom is the available resources.
- The intermediate layers is than introduced.



7 Layers of the OSI Model

Application	• End User layer • HTTP, FTP, IRC, SSH, DNS
Presentation	• Syntax layer • SSL, SSH, IMAP, FTP, MPEG, JPEG
Session	• Synch & send to port • API's, Sockets, WinSock
Transport	• End-to-end connections • TCP, UDP
Network	• Packets • IP, ICMP, IPSec, IGMP
Data Link	• Frames • Ethernet, PPP, Switch, Bridge
Physical	• Physical structure • Coax, Fiber, Wireless, Hubs, Repeaters

- Two forms of layered architectures: –
  - Closed architecture
    - Each layer is built only in terms of the immediate lower layer.
  - Open architecture
    - A layer can use features on any lower layer to any depth.
    - Do not observe the principle of information hiding

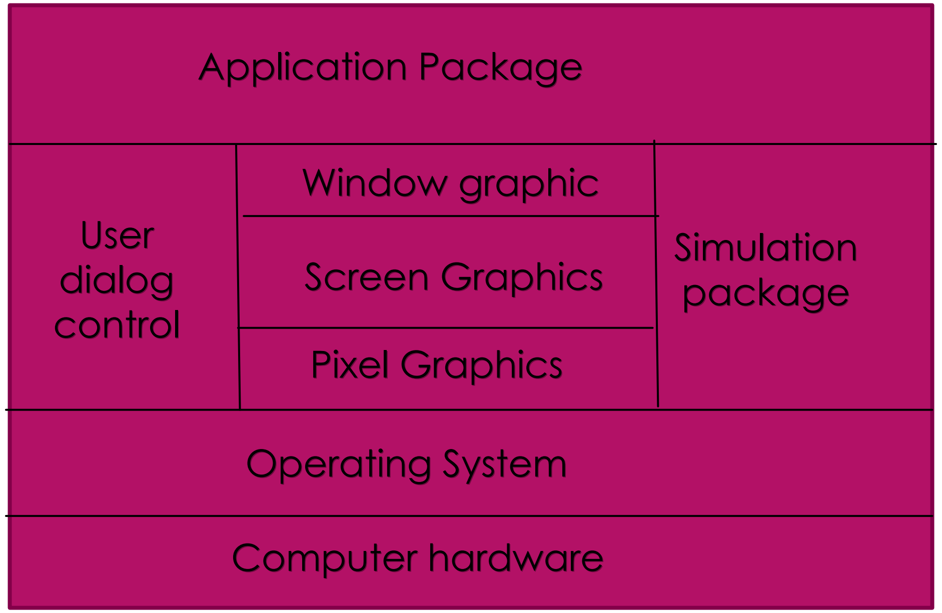
Breaking a System into Subsystem - Partitions

- Vertically divided into several subsystems
- Independent or weakly coupled
- Each providing one kind of service.
- E.g. A computer operating system includes
  - – File system
  - – Process control
  - – Virtual memory management
  - – Device control

- Layers vary in their level of abstraction.
- • Layers depend on each other.
- • Partitions divide a system into pieces.
- • Partitions are peers that are independent or mutually dependent. (peer-to-peer relationship)

Combining layers and Partitions

- We can decompose a system into subsystems by combining layers and partitions
- Layers can be partitioned and partitions can be layered



Application Package

User dialog control

Window graphic

Screen Graphics

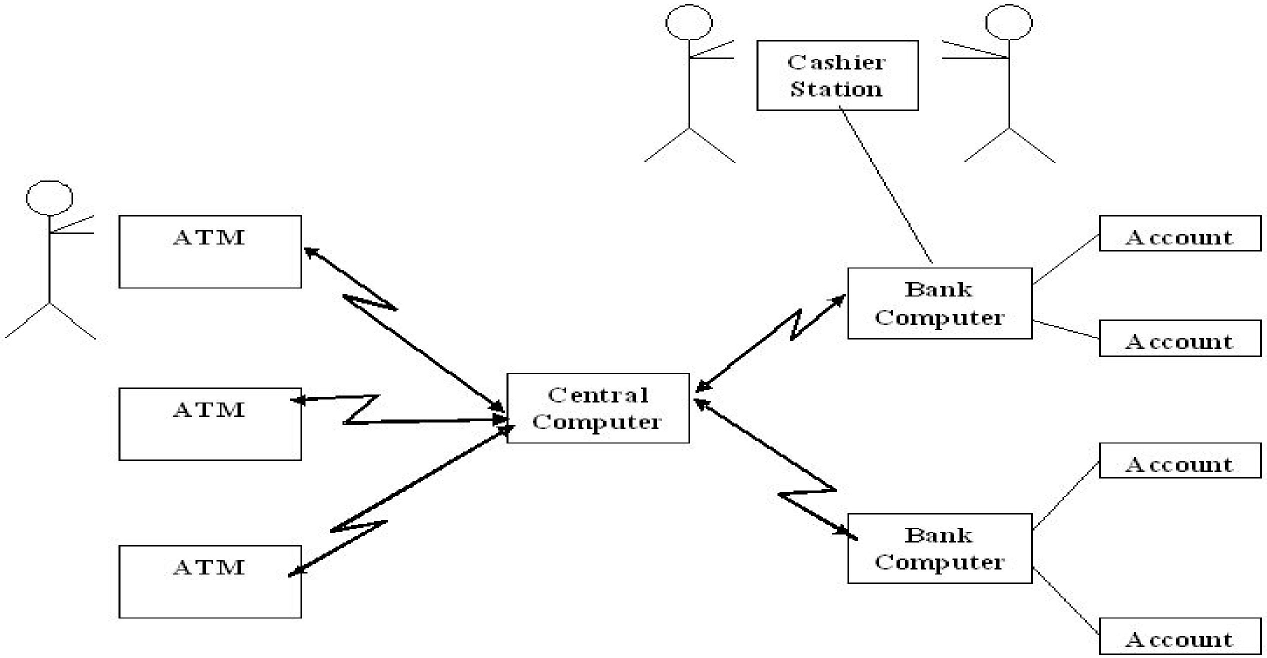
Pixel Graphics

Simulation package

Operating System

Computer hardware

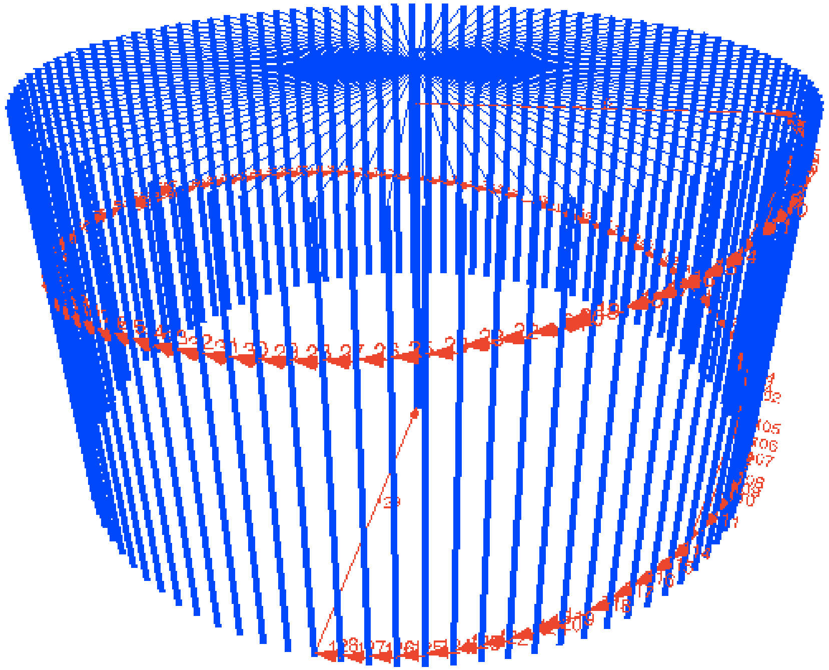
Case Study: ATM System

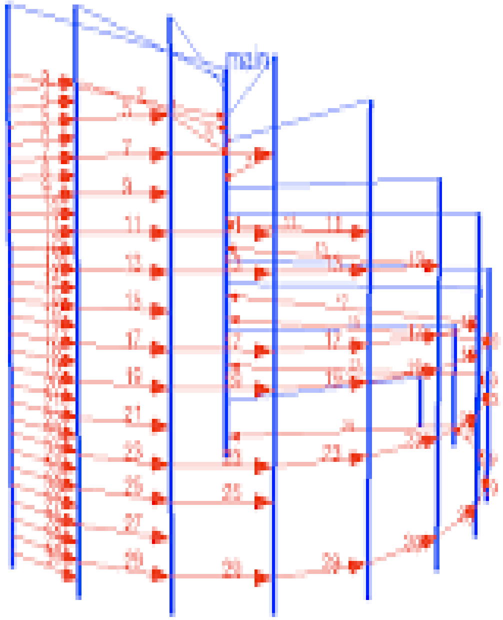


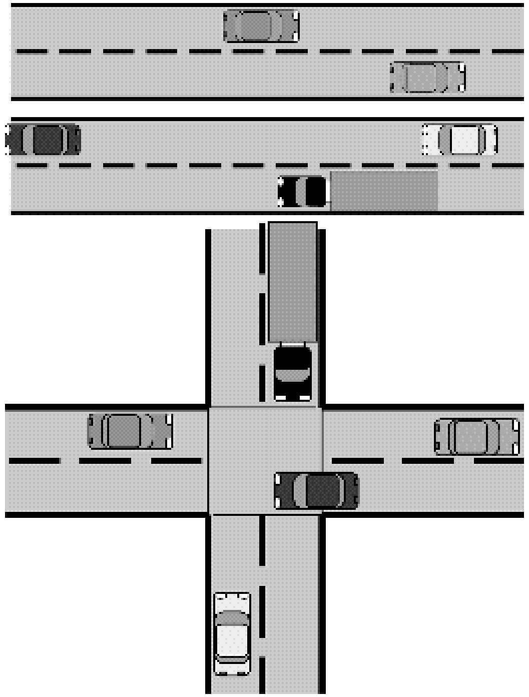
Identifying Concurrency

- One processor may support many objects
- We can implement many objects on a single processor if the objects cannot be active together
- One important goal of system design is to identify the objects that must be active concurrently and objects that have mutually exclusive activity
- Identify the objects that must be active concurrently and objects that have mutually exclusive activity.
- You can fold the latter objects into a single thread of control or task.

- Two steps:
  - Identifying inherent concurrency
  - Defining Concurrent Tasks.







Identifying Inherent Concurrency

- State model guide to identifying concurrency.
- If two objects receive events at the same time without interacting then its called **inherently concurrent.**
- If the events are not synchronized the two objects can not be on a single thread of control.
- Only a single object at a time may be active on a thread.
- Often problem statement specifies that distinct hardware units can implement the objects

- For Example ATM:

Each machine should continue to operate locally in event of central system failure, then would have no choice but to include a CPU in each ATM machine with full control program.

Defining concurrent Tasks

- By studying state diagram of individual objects and exchange of events among them.
- we can fold many objects onto a single thread of control.
- A *thread of control* is path through a set of state diagram on which only a single object at a time is active.

- A thread remains within a state diagram until an object sends an event and passes to the receiver of event.
- A thread splits if the object sends an event and continues executing.
- Only single object at a time is active.
- ATM Ex. :

  If a central computer directly controls the ATM, we can combine ATM object with Bank transaction object as a single task

Allocation of Subsystem

- Must allocate each concurrent subsystem to hardware unit.
- Either general purpose or specialized functional unit.
- Criteria would be
  - Estimate hardware resource requirement
  - Making hardware-software trade-off
  - Allocating Tasks to processors
  - Determining Physical connectivity.

Estimating hardware resource requirement

- System designer must estimate the required CPU processing power by computing steady-state load.
  - I.e. Product of number of transaction per second and time required to process a transaction.
- You should increase estimate to allow for transient effects(i.e. failure cases)
- Steady-state and peak load are important.

- For. Ex.

  ATM machine just provide user interface and some local processing. So single CPU would work for each ATM.

- Consortium computer is essentially just a routing machine – as it receives ATM request and sending them to appropriate bank computer. So it involves multiple CPUs.
- While Bank computers perform data processing and database application.
- Appropriate choice depends upon needed throughput and reliability.

Making Hardware-Software Trade-offs

- You must decide which subsystem will be implemented in hardware and which in software.
- Two main reason for implementing subsystems in hardware:
  - Cost – it is easier buy floating point chip than to implement floating point in software.
  - Performance – more efficient hardware is available.

Allocating Tasks to processors

- Several reasons for assigning task to processors
- *Logistics* : Some tasks are required at specific physical locations or to permit independent operations.
- For ex. At ATM, when communication network is down on that case ATM must have its own CPU and programming logic.

- Communication Limits
  - Sometime available communication bandwidth between task & piece of Hardware is not sufficient then response is not getting on time or its taking more time to complete.
- Computation Limits
  - Assign separates processors for computation process.

Determining Physical Connectivity

- Connection topology:
  - Choose the topology for connecting physical units.
  - Association in the class model indicating physical connection.
  - For ATM communicate with Consortium.
  - Also client-server relationship also correspond to physical connections.

- Repeated units
  - Topology has a regular pattern such as
    - Linear Sequence , A matrix, A tree, A Star
- If you have boosted performance for particular kinds of units or group of units, then you must specify their topology.
- Communications
  - Choose the form of connection channel & communication protocol.

- System designer may specify the exact interfaces among units for interaction mechanisms.
  - For Ex. Interactions may be Asynchronous, Synchronous or blocking.
- Based on estimate bandwidth & latency of the communication channels and choose the correct kinds of communication channels.

Management of data storage

- There are several alternatives for data storage.
- You can use separately or in combinations.
  - Data structure
  - Files
  - Database
- Based on cost, access time and reliability different kinds of data storage.
  - For Ex. PC application may use memory data structure or files.
  - An accounting system may use a database,

- Data storage – Files
  - Files are cheap, simple and permanent.
  - Files implementation vary for different computer systems.
  - For ex. Implementation for sequential files are mostly standard, but commands and storage formats for random-access files and indexed files vary

- Kinds of data that belongs in files
  - Data with high volume and low information density (historical records)
  - Quantities of data with simple structure.
  - Data that are accessed sequentially.
  - Data that can be fully read into memory

- **<u>Data storage – Database</u>**
  - Various type of DBMS available
    - Relational DB
    - OO DB.
- DBMS cache frequently accessed data in memory.
- DB makes application easier to port different hardware and OS.
- Disadvantage – complex interface

- Kinds of data that belongs in DB
  - Data that requires updates from multiple users.
  - Data that must accessed by multiple application programs.
  - Data that require updates via transaction.
  - Data that are long-lived and highly valuable.
  - Data that must be secure.

Note:

- Most application need a database, you should use relational DBMS.
- RDBMS features are sufficient for most application.

Handling Global Resources

- System designer must identify
  - Global resources
  - Mechanism for controlling the access.
- There are several kinds of global resources.
  - Physical units : Processors, tape drives and communication satellites.
  - Space: Disk space, workstation screen and buttons on a mouse

  - Logical names: Object IDs, Filenames and class names.
  - Access to shared data: Database
- Resource can be
  - Physical object : It can control itself by specifying protocol
  - Logical object: Conflicting access in a shared environment.
  - To solve conflicting access, you should introduce “Guardian object”.

- Guardian object - own each global logical resource and have control access to it.
  - All access must be pass through Guardian object only
- In ATM example, Bank code and Account numbers are global resources.
- Bank codes must be unique within consortium.
- Account number must be unique within the Bank.

Choose the Software control strategy

- Two kinds of control system in Software system:
  - External control
  - Internal control
- External control concerns the flow of *externally visible events* among objects in the system.
- Three kinds of control *external event*:
  - *Procedure-driven sequential*
  - *Event-driven sequential*
  - *Concurrent*

- Control style depend on the available resource (language, OS) and on the kind of interaction.

Internal control refers to the *flow of control within the process.*

- It is only used in implementation therefore neither *concurrent nor sequential*.

Procedure-driven Control

- Control resided inside the *program code*.

Basic working:

- Procedure request for input and wait for it.
- When input arrives, control resumes within the procedure that made the call.

Advantage:

- Easy to implement with conventional programming.

Disadvantages:

- It requires the concurrency inherent in objects

- Procedure-driven paradigm is suitable only if state model shows a regular alternation of input and output events.

Note:

- C++ and Java are procedural languages. That is why they fail to support concurrency inherent in objects.

Event-driven Control

- Control resides within a *dispatcher*

Basic Working

- Developers attach application procedures to events
- Dispatcher call the procedure when events occurs.
- Procedure calls to the dispatcher send output or enable input but do not wait for it in-line.

- Once event is over, procedure return control to the dispatcher instead on retaining.

<u>Advantage:</u>

- More flexible control than procedure-driven systems.
- Event-driven system are more modular and can handle error conditions better than procedure-driven system.

Concurrent Control

- Control resides concurrently in several independent objects, each a separate task.

Basic Working:

- A task can wait for input but other tasks continue execution.
- OS resolves scheduling conflicts among tasks and usually supplies a queuing mechanism so that events are not lost.

Internal control

- Internal object interaction are similar to external object interaction because you can use the same implementation mechanisms.
- Importance difference is
  - External interaction inherently involve events, because objects are independent.
  - Internal interaction involve operation as procedure calls.

- Other paradigm are possible such as
  - Rule based systems
  - Logic programming system
  - Other forms Nonprocedural programs
- Developer used such languages in limited areas only such as
  - Artificial Intelligence
  - Knowledge based programming

Handling Boundary Conditions

- Consider boundary conditions as well and address kinds of issues.
- Initialization
  - System must initialize constant data, parameters, global variables, guardian objects, and possibly the class hierarchy.
  - Initialize concurrent task is most difficult because object are independent.

- Termination
  - Termination is simpler than initialization because internal object can be abandoned.
  - In concurrent system, one task must notify other task of its termination.
- Failure:
  - Failure is unplanned termination of a system.
  - Basically it arise from user errors or from external breakdown.
  - It can also arise from bugs in the system

Common Architectural Styles

Several kinds of system listed :

- ❑ Batch transformation
- ❑ Continuous transformation
- ❑ Interactive interface
- ❑ Dynamic simulation
- ❑ Real-time system
- ❑ Transaction Manager

Batch transformation

- A batch transformation performs sequential computations.

Working:

- Application receives the input
- Compute the answer
- No ongoing interaction with outside world

Example:

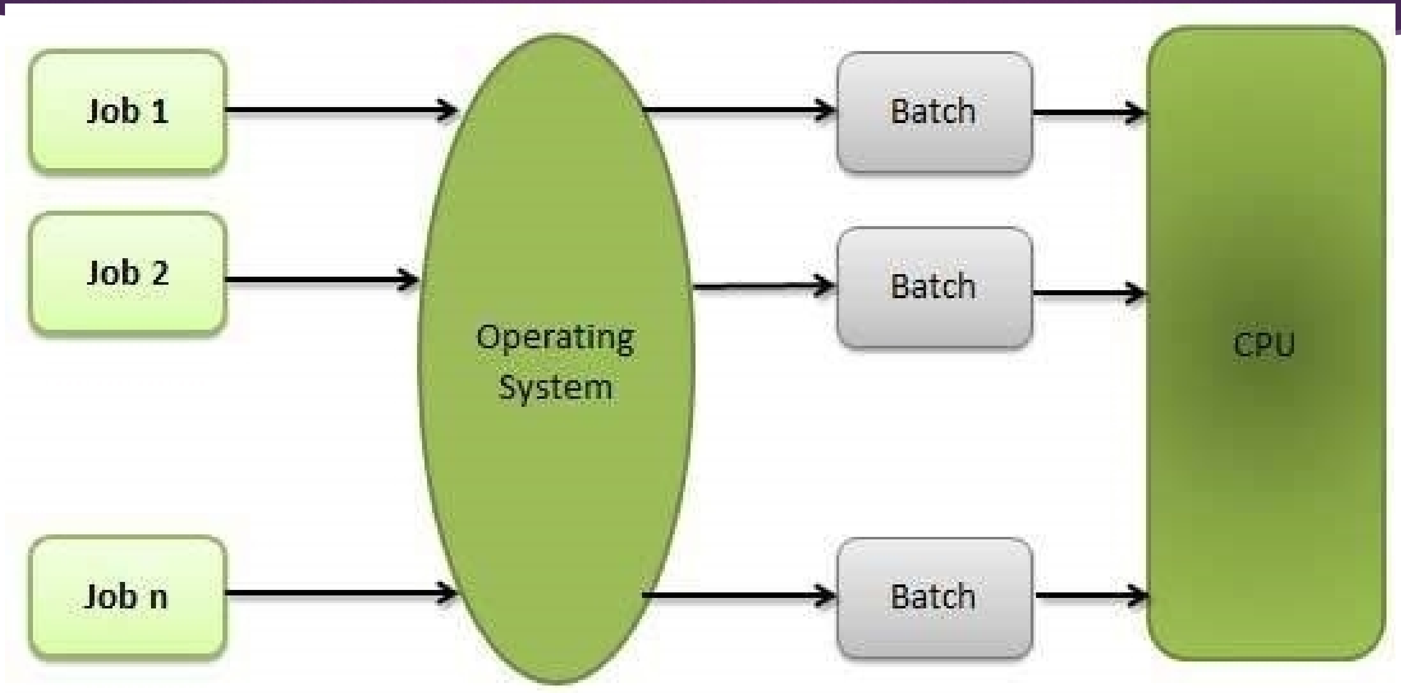
- Computational problems such as compilers, payroll processing etc.

- For batch transformation problem, we can use class and interaction models.
- As it contains input, output and intervening stages.
- Most important thing is it should define a clean series of steps.

- Compiler has five class model
  - One for input, one for output and three for intermediate representation.

Steps for designing batch transformation:

- Break overall transformation into stages
  - Each stage performing one part of transformation.
- Prepare class models for input, output and intermediate stages.
- Expand each stage until the operation are easy to implement.
- Restructure the final pipeline for optimizations.


Job 1
Job 2
Job n
Operating System
Batch
Batch
Batch
CPU

Continuous transformation

- It is a system in which the outputs actively depends on changing inputs.
- In batch transformation, output compute one time.
- In continuous transformation, updates output frequently.
- System can’t recomputed each set of output each time an input changes.

- Implement continuous transformation with a pipeline of functions.
- Pipeline propagates the effect of each input change.
- To improve the performance of pipeline, define intermediate and redundant objects.

Steps for designing a pipeline for continuous transformation

- Break overall transformation into stages
  - Each stage performing one part of transformation.
- Define input, output and intermediate models between each pair of stages.
- Differentiate each operation – that is propagate the incremental effects of each change to an input through the pipeline as series of incremental updates.
- Add additional intermediate objects for optimization

Interactive Interface

- System that is dominated by interaction between the system and external agents, such as humans or devices.
- System can not control the agents.
- Example of interactive system include
  - forms-based query interface,
  - Workstation windowing system and
  - Control panel for a simulation.

- Major concerns of interactive interface are
  - Communication protocol between system and agents
  - Syntax of possible interaction
  - Presentation of output
  - Flow of control within the system
  - Performance
  - Error handling

Steps for designing interactive Interface

- Separate interface classes from the application classes.
- Use predefined classes to interact with external agents
- Use the state model as the structure of the program.(i.e. concurrent, event-driven or procedure-driven control)
- Isolate physical events from logical event.
- Fully specify the application functions that invoked by the interface.

Dynamic Simulation

- A dynamic simulation models tracks real-world objects.
  - Ex. Economic models, video games.
- Objects and operation com directly from the application.
- Two ways for implementing control:
  - Explicit controller to application object
  - Objects can exchange messages among themselves.

Steps in designing dynamic simulation

- Identify active real-world objects from the class model.
- Identify discrete events.
- Identify continuous dependencies. (i.e. one attributes may depend on other)
- Discrete events between objects can be exchanged as a part of timing loop.

Real-time system

- A real-time system is an interactive system with *tight time constraints* on actions.
- Two types of real-time system
  - Hard real-time system
    - Critical application that require a guaranteed response within the time constraints.
  - Soft real-time system
    - Highly reliable, but occasionally violate time constraints.

- Real-time design is complex and involves issue such as
  - Interrupt Handling
  - Prioritization of tasks
  - Coordinating multiple CPUs

Transaction manager

- Main function is to store and retrieve data.
- It deal with multiple users who read and write data at the same time.
- It also secure data from unauthorized access.
- It is built on top of a DBMS

Steps in designing

- Map the class model to database structure.
- Determine the units of concurrency.
- Determine the units of transaction.
- Design concurrency control for transaction.

Thank you