Software Design

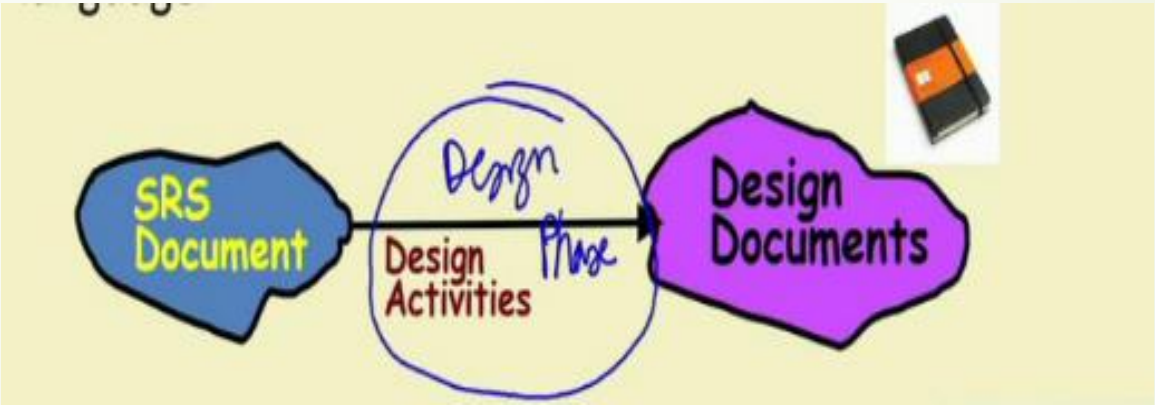
Courtesy:

Roger Pressman, Ian Sommerville &

Prof Rajib Mall

Introduction:

- The activities carried out during the design phase transform the SRS document into the design document.
  - A form easily implementable in some programming languages.



Outcome of the Design Process:

- The following items are designed and documented during the design phase.
  - Different Modules required
  - Control relationships among modules
  - Interfaces among different modules
  - Data structures of the individual modules
  - Algorithms required to implement the individual modules

Classification of Design Activities:

- A good software design is seldom realized by using a single step procedure, rather **it requires iterating over a series of steps** called the design activities.
- Two important stages
  - **Preliminary (or high-level) design:** Through high level design, a problem is decomposed into a set of modules. The control relationships among the modules are identified and also the interfaces among various modules are identified.
  - **Detailed design:** During detailed design each module is examined carefully to design its data structures and the algorithms.

Analysis vs Design:

- The **analysis results are generic and does not consider implementation** or the issues associated with specific platforms.
- The analysis model is usually documented using some graphical formalism.
- Function oriented approach:
  - Analysis model documented using DFDs.
  - Design would be documented using structure chart
- Object oriented approach
  - Both design model and analysis model documented using UML.
- **The design model reflects several decisions taken regarding the exact way system is to be implemented. The design model should be detailed enough to be easily implementable using a programming language.**

Classification of Design Methodologies:

- The design activities vary considerably based on the specific design methodology being used.
- Roughly classified as
  - **Procedural** and **Object Oriented approach**
- **Do design techniques result in unique solutions?**
- **How to distinguish superior design solution from an inferior one?**

How to characterize a good software design?

- The definition of a "good" software design can vary depending on the exact application being designed.
- **Correctness:** A good design should first of all be correct. It should correctly implement all the functionalities of the system.
- **Understandability:** A good design should be easily understandable. Unless a design solution is easily understandable, it would be difficult to implement and maintain it.
- **Efficiency:** A good design solution should adequately address resource, time, and cost optimization issues.
- **Maintainability:** A good design should be easy to change.
- **Otherwise, it would require tremendous effort to implement, test, debug, and maintain it.**

How to characterize a good software design?

- A design solution should have the following characteristics to be easily understandable:
  - It should assign consistent and meaningful names to various design components.
  - It should make use of the principles of decomposition and abstraction in good measures to simplify the design.
- A design solution is understandable, if it is modular and the modules are arranged in distinct layers.

Modularity:

- Modular design implies that the problem has been decomposed into a set of modules that have only limited interactions with each other.
- Decomposition of a problem into modules facilitates taking advantage of the ***divide and conquer*** principle.
- If different modules have either no interaction or little interactions with each other, then each module can be understood separately.
- A layered design can make the design solution easily understandable.
- A design solution is said to be highly modular, if the different modules in the solution have high cohesion and their intermodule coupling are low.

Cohesion and Coupling:

- When the function of the module co-operate with each other for performing a single objective, then module has **good cohesion.**
- Two modules are said to be highly coupled,
  - If the function calls between two modules involve passing large chunks of shared data, the modules are **tightly coupled.**
  - If the interactions occur through some shared data, then also we say that they are **highly coupled.**
- If two modules either do not interact with each other at all or at best interact by passing no data or only a few primitive data items, they are said to have **low coupling.**

Cohesion and Coupling:

- **Cohesion** is a measure of the functional strength of a module, whereas the **coupling** between two modules is a measure of the degree of interaction (or interdependence) between the two modules.
- A module that is highly cohesive and also has low coupling with other modules is said to be **functionally independent** of the other modules.
- Functional independence is a key to any good design because of…
  - Error Isolation
  - Scope of reuse
  - Understandability

Cohesion and Coupling:

- Cohesion is a measure of :
  - Functional strength of a module.
  - A cohesive module performs a single task or function.
- Coupling between two modules:
  - A measure of the degree of interdependence or interaction between the two modules.
- A module having high cohesion and low coupling:
  - Called functionally independent of other modules.
  - A functionally independent module needs very little help from other modules and therefore has minimal interaction with other modules.

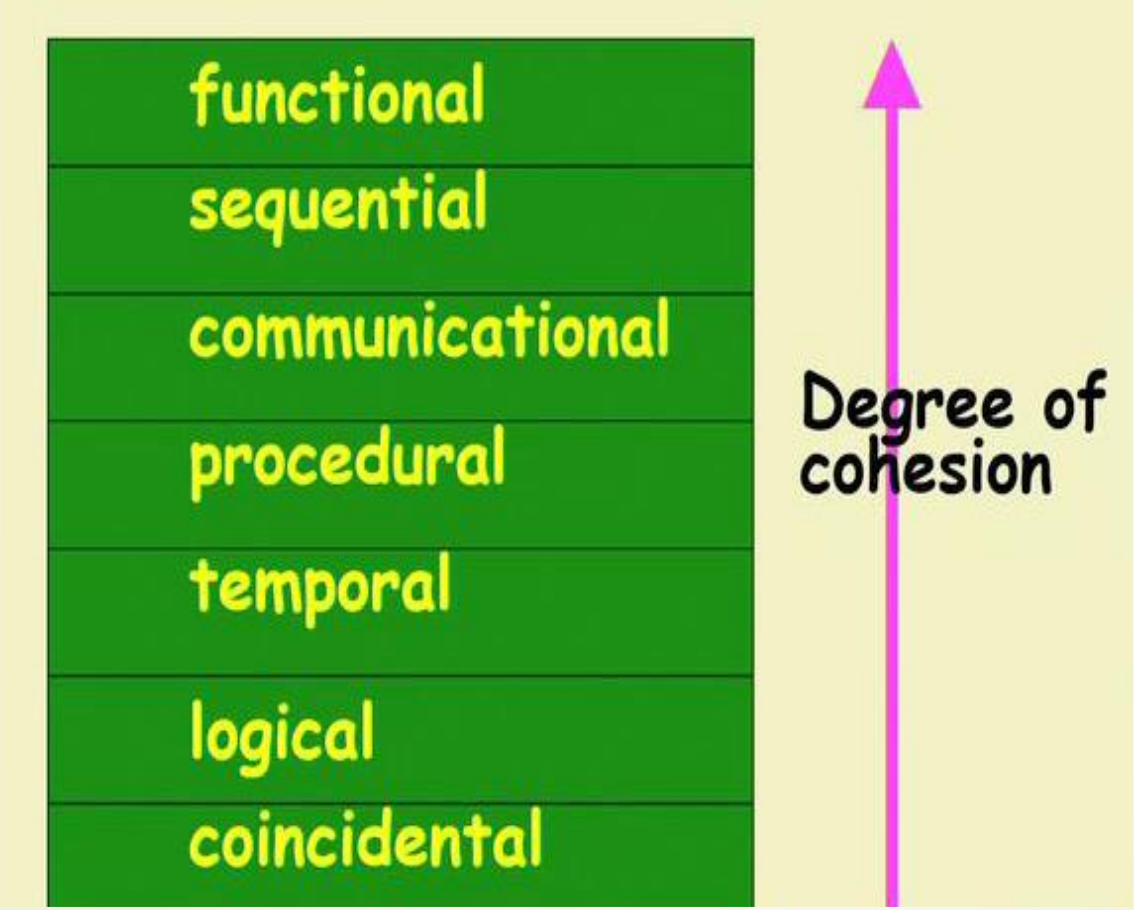
Advantages of Functional Independence

- Better understandability
- Complexity of the design is reduced,
- Different modules easily understood in isolation:
  - Modules are independent.
- Functional independence **reduces error propagation.**
  - Degree of interaction between modules is low.
  - An error in one module does not directly affect other modules.
- Also **Reuse of modules is possible.**
  - Each module does some well defined and precise function.
  - The interfaces of a module with other modules is simple and minimal.

Measuring Functional Independence:

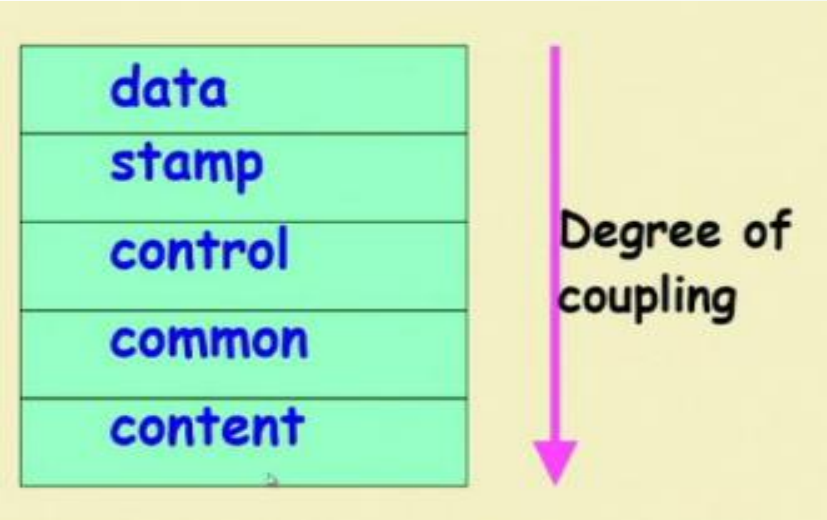
- Unfortunately, there are no ways:
  - To quantitatively measure the degree of cohesion and coupling.
  - At least classification of different kinds of cohesion and coupling will give us some idea regarding the degree of cohesiveness of a module.
- Classification can have scope for ambiguity:
  - Yet gives us some idea about cohesiveness of a module.
- By examining the type of cohesion exhibited by a module:
  - We can roughly tell whether it displays high cohesion or low cohesion.

Classification of Cohesiveness:



Classification of Coupling:

- There are no ways to precisely measure coupling between two modules:
  - Classification of different types of coupling will help us to approximately estimate the degree of coupling between two modules.



System modeling:

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

System perspectives:

- An **external perspective**, where you model the context or environment of the system.
- An **interaction perspective**, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural perspective**, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral perspective**, where you model the dynamic behavior of the system and how it responds to events.

Unified Modeling Language (UML):

- The main aim of UML is to define a standard way to visualize the way a system has been designed
  - Acts as a blueprints
  - A visual language
- UML helps software engineers, businessmen and system architects with modelling, design and analysis.
- A clear and concise way to communicate among teams/collaborators.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

UML Diagrams:

- UML is linked with object oriented design and analysis.
- Diagrams in UML can be broadly classified as:
  - **Structural Diagrams** – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
  - **Behavior Diagrams –** Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

Structural UML Diagrams:

- **Class Diagram**
  - It is the building block of all object oriented software systems.
  - We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes.
  - Class diagrams also help us identify relationship between different classes or objects.
- **Object Diagram**
  - To study the behavior of the system at a particular instant
- **Component Diagram**
  - Component diagrams are used to represent the how the physical components in a system have been organized.
- **Deployment Diagram**

Behavior Diagrams:

- **State Machine Diagrams-** represent the condition of the system or part of the system at finite instances of time.
- **Activity Diagrams –** illustrate the flow of control in a system.
- **Use Case Diagrams –** Use Case Diagrams are used to depict the functionality of a system or a part of a system.
- **Sequence Diagram –** A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.
- **Communication Diagram –** A Communication Diagram(known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects.

Data Flow Diagram (DFD):

- A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.
- It shows how data enters and leaves the system, what changes the information, and where data is stored.
- The objective of a DFD is to show the scope and boundaries of a system as a whole.
- A neat and clear DFD can depict the right amount of the system requirement graphically.
- DFD represents data flow and not control flow.

Design Approaches:

- Two fundamentally different software design approaches:
  - **Function-Oriented Design**
  - **Object-Oriented Design**
- These two design approaches are radically different.
  - However, are complementary rather than competing techniques.
- Each technique is applicable at different stages of the design process.

Function-Oriented Design:

- A system is looked upon as something that performs a set of functions.
- Starting at this high-level view of the system:
  - Each function is successively refined into more detailed functions (top down decomposition).
  - Functions are mapped to a module structure.
- Identifying the set of functions to be performed is called as the **structured analysis.**
- Mapping the identified functions to module structure **(Structured design)**.

Function Oriented Software Design

- Function oriented design techniques are currently being used in many software development projects.
- These techniques,
  - **view a system as a black-box** that provides a set of services to the users of the software.
  - These services provided by a software to its user are also known as the **high-level functions supported by the software.**
- The top down decomposition is carried out and different identified functions are mapped to modules and a module structure is created.
- **The design techniques are structured analysis/structured design (SA/SD) methodology.**

Function Oriented Design Example:

- **The function create new library member:**
  - Creates the record for a new member,
  - Assigns the unique membership number,
  - Prints a bill towards the membership
- **The system state is centralized:**
  - **Accessible to different functions**
  - Member Records: available for reference and updation to several functions:
    - Create new member
    - Delete member
    - Update member record

Function Oriented Software Design

- The structured analysis activity transforms the SRS document into a graphic model called the DFD model.
- During structure design, all the functions identified during structured analysis are mapped to a module structure (high level design). This is represented using structure chart.
- The **purpose of structure analysis** is to capture the detailed structure of the system as perceived by the user.
- The **purpose of the structured design** is to define the structure of the solution that is suitable for implementation in some programming language.

Structured Analysis and Structured Design

- **During Structured Analysis:**
  - High level functions are successively decomposed into more detailed functions.
  - The purpose of structure analysis is to capture the detailed structure of the system as the user views it.
- **During Structured Design:**
  - The detailed functions are mapped to a module structure.
  - The purpose of structured design is to arrive at a form that is suitable for implementation in some programming language.

Structured Analysis:

- Textual problem description converted into a graphic model.
  - Done using data flow diagrams (DFDs).
  - DFD graphically represents the results of structured analysis.
- The results of structured analysis can be easily understood even by ordinary customers:
  - Does not require computer knowledge.
  - Directly represents customers perception of the problem.
  - Uses customers terminology for naming different functions and data.
- Results of structured analysis can be reviewed by customers to check whether it captures all their requirements.

Structured Analysis:

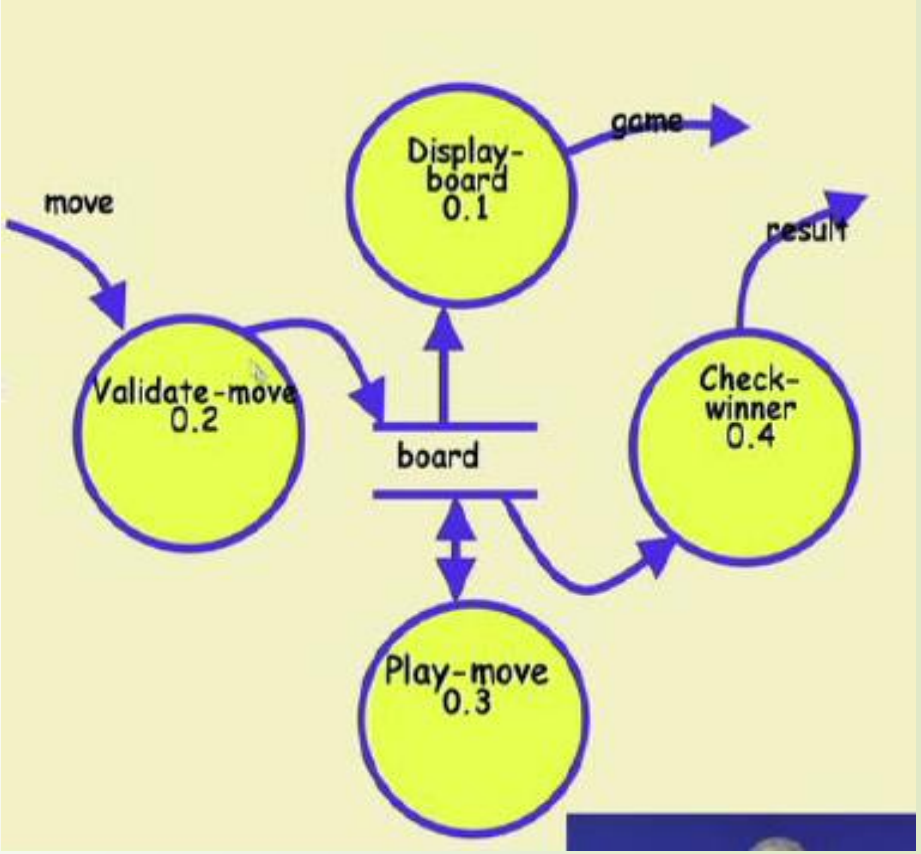
- The structure analysis technique is based on the following underlying principles:
  - **Top-down decomposition approach**
  - **Application of divide and conquer principle.**
  - **Graphical representation of the analysis results using data flow diagrams(DFDs).**
- The DFD model only represents the data flow aspects and does not show the sequence of execution of the different functions and the conditions based on which a function may or may not be executed.

Data Flow Diagrams (DFDs)

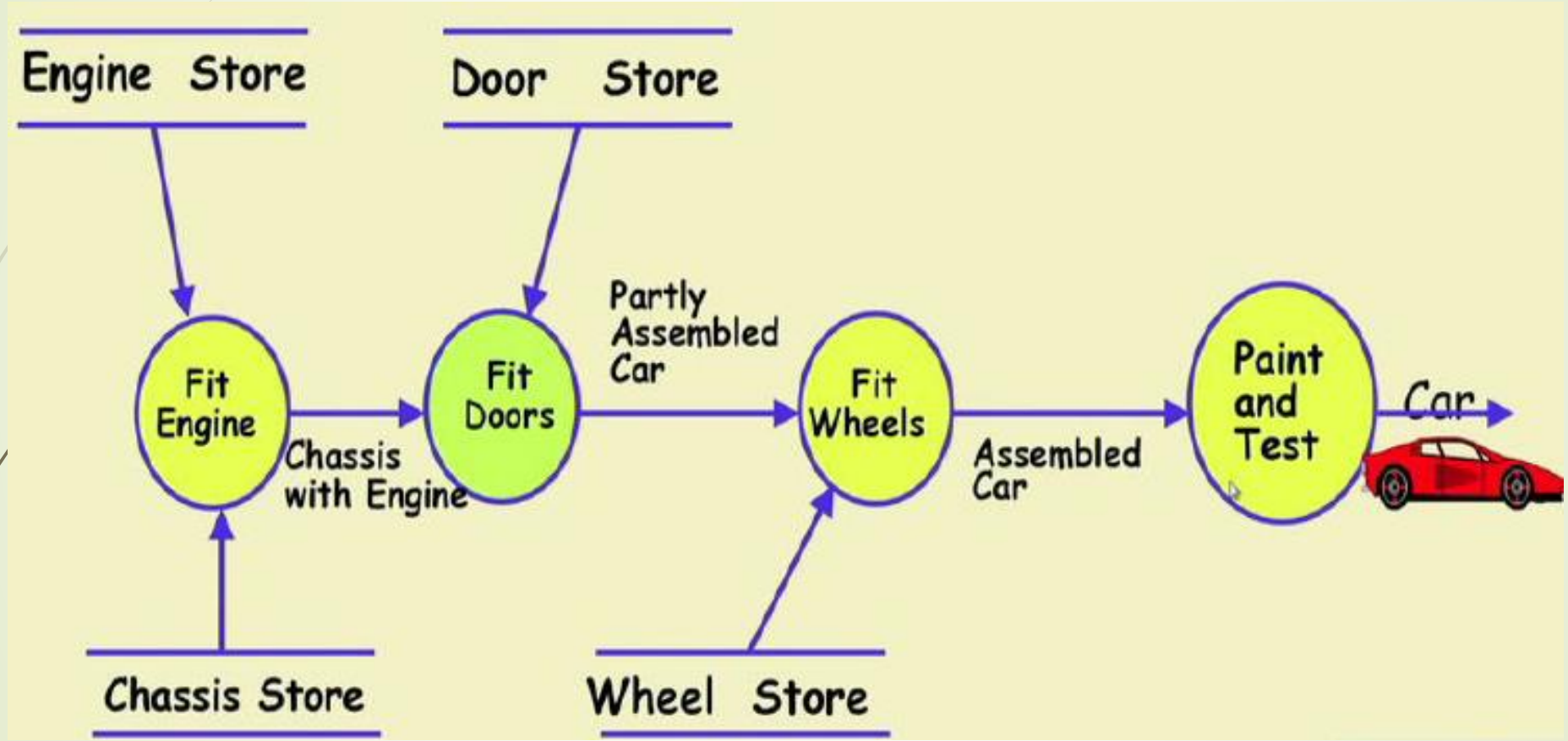
- The DFD (also known as bubble chart) is a simple graphical formalism that can used to represent
  - **a system in terms of the input data to the system,**
  - **various processing carried out on those data, and**
  - **the output data generated by the system.**
- A DFD is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among those functions.
- A DFD model uses a very limited number of primitive symbols to represent the functions performed by a system and the data flow among these functions.

Data Flow Diagrams (DFDs)

- DFD is a hierarchical graphical model:
  - Shows the different functions (or processes)of the system
  - Data interchange among the processes
- A DFD model:
  - Uses limited types of symbols.
  - Simple set of rules.
  - Easy to understand--- a hierarchical model
- It is useful to consider each function as a processing station:
  - Each function consumes some input data.
  - Produces some output data.



Data Flow Model of a Car Assembly Unit:

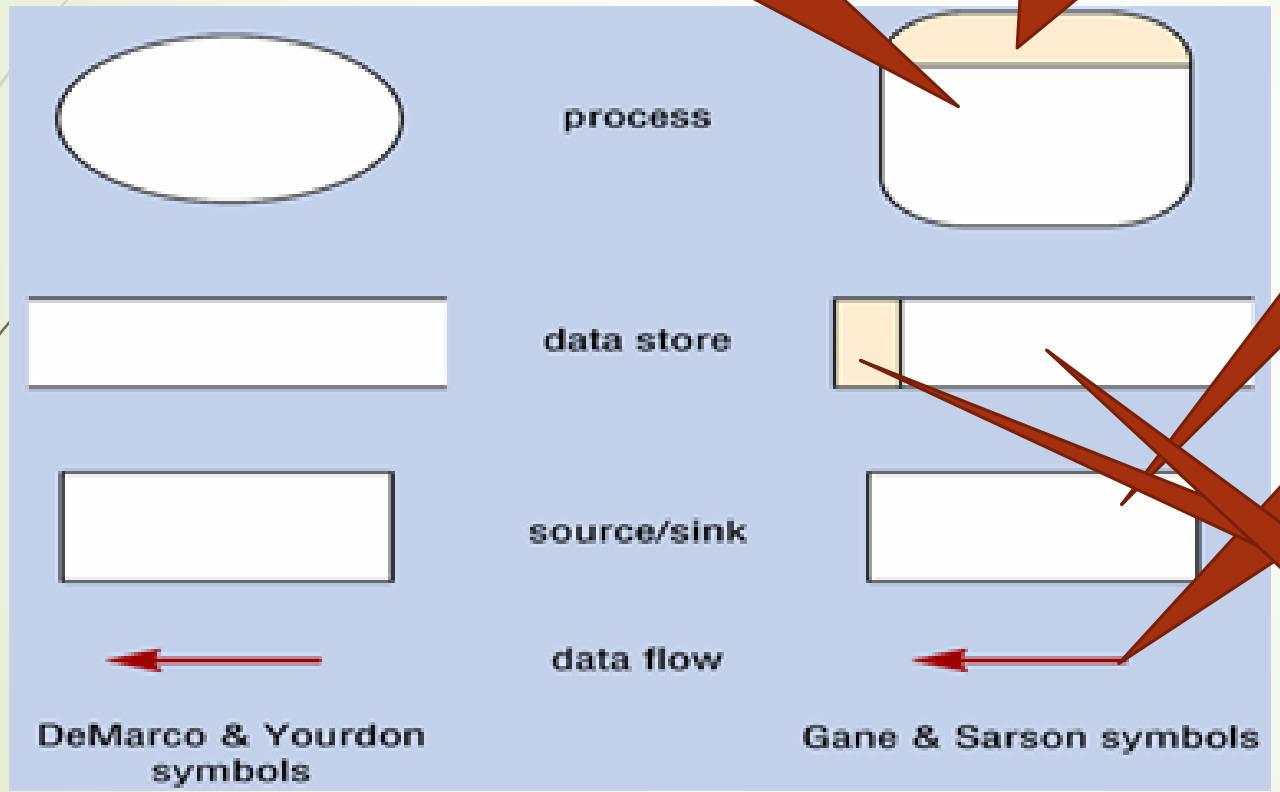


Dat ramming M s

Meaningful label for process
-Generate PayCheck
-Calculate overtime pay
-Compute GPA

Used to number process

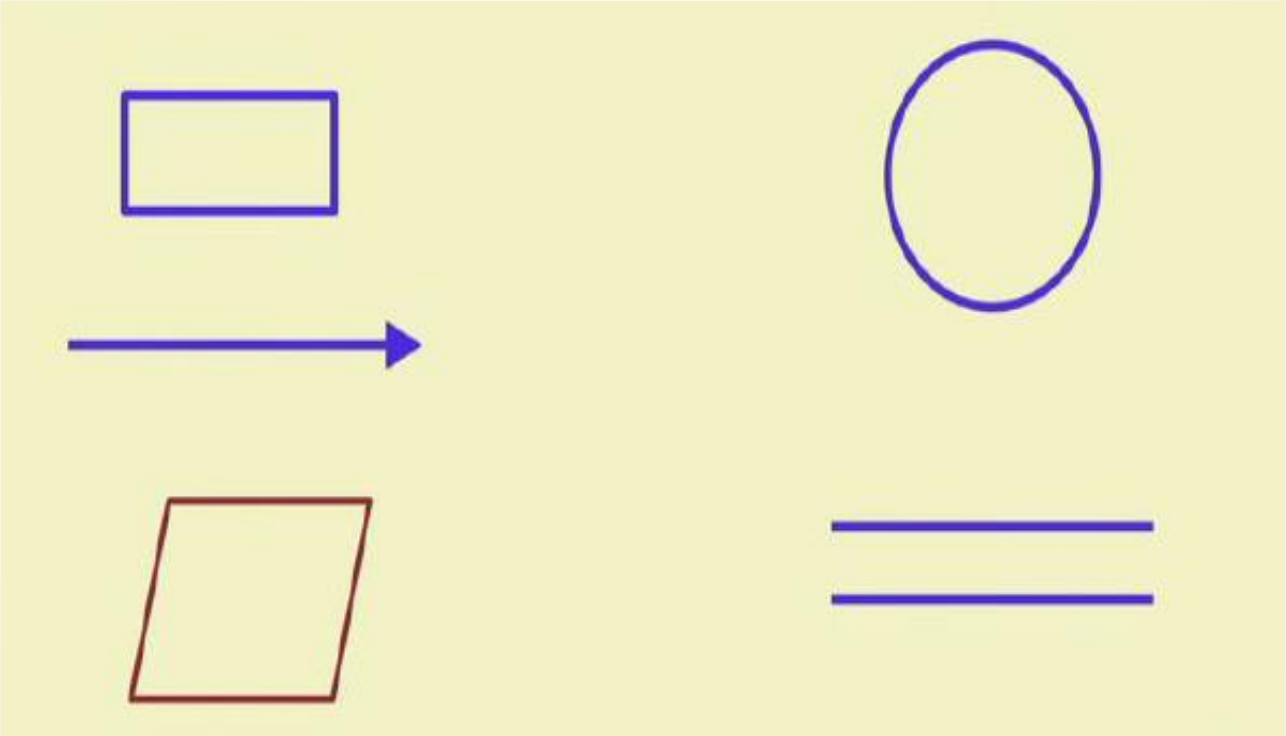
Has a name which state what the external agent is
-Customer
-Teller
-Account office
-Inventory Control System



Sliced with meaningful name for the data in motion
-Customer Order
-Sales Receipt
-PayCheck

Meaningful label for data store
-Student File
- Transcripts

Basic Symbols used for Constructing DFDs



Symbols and Notations Used in DFDs:

- **External entity:** An outside system that sends or receives data, communicating with the system being diagrammed. They are also known as terminators, sources and sinks or actors.
- **Process:** Any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.
- **Data store:** Files or repositories that hold information for later use, such as a database table or a membership form.
- **Data flow:** The route that data takes between the external entities, processes and data stores.

External Entity Symbol:

- Represented by a rectangle
- External entities are either users or external systems:
  - Input data to the system or
  - Consume data produced by the system
  - Sometimes external entities are called as terminator or source.

Function Symbol:

- A function such as “search-book” is represented using a circle:
- This symbol is called a **process or bubble or transform.**
- Bubbles are annotated with corresponding function names.
- A function represents some activity, **Function names should be verbs.**

Data Flow Symbol:

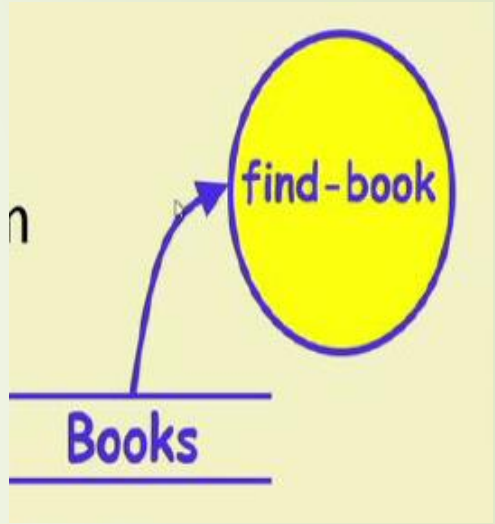
- A directed arc or line.
- Represent data flow in the direction of the arrow.
- Data flow symbols are annotated with names of data they carry.

Data Store Symbol:

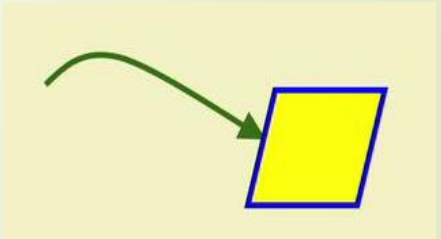
- Represent a logical file
- A logical file can be
  - A data structure
  - A physical file on disk.
- Each data store is connected to a process:
  - By means of a data flow symbol.

Data Store Symbol:

- Direction of data flow arrow:
  - Shows whether data is being read from or written into it.
- An arrow into or out of data store:
  - Implicitly represents the entire data of the data store
  - Arrows connecting to a data store need not be annotated with any data name.

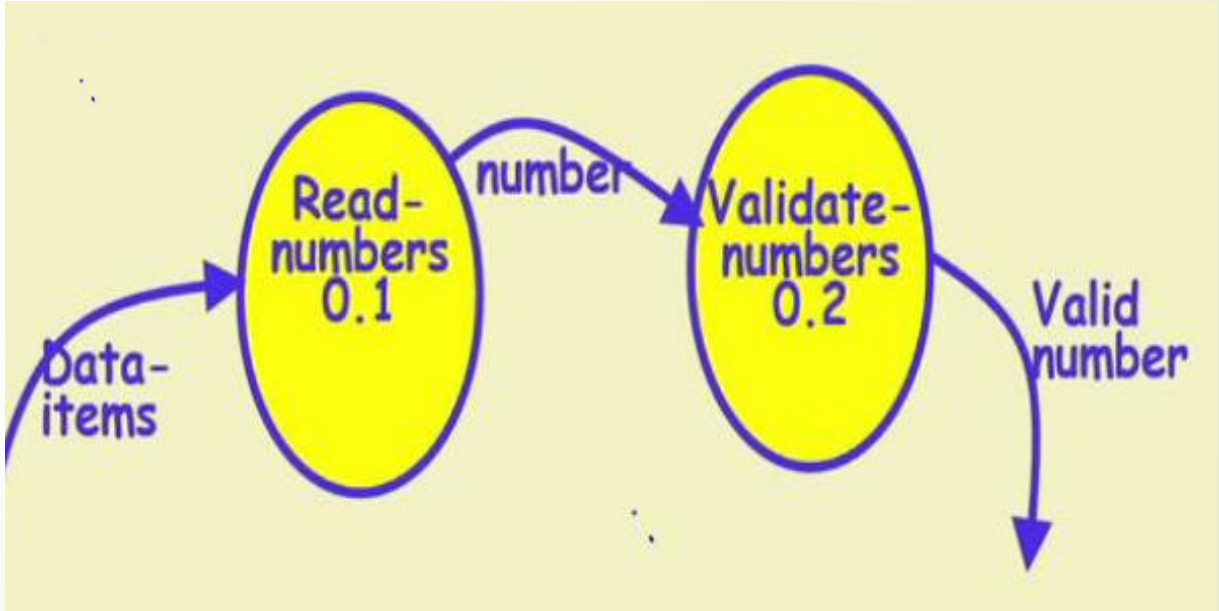


Output Produced by the system is represented by parallelogram.



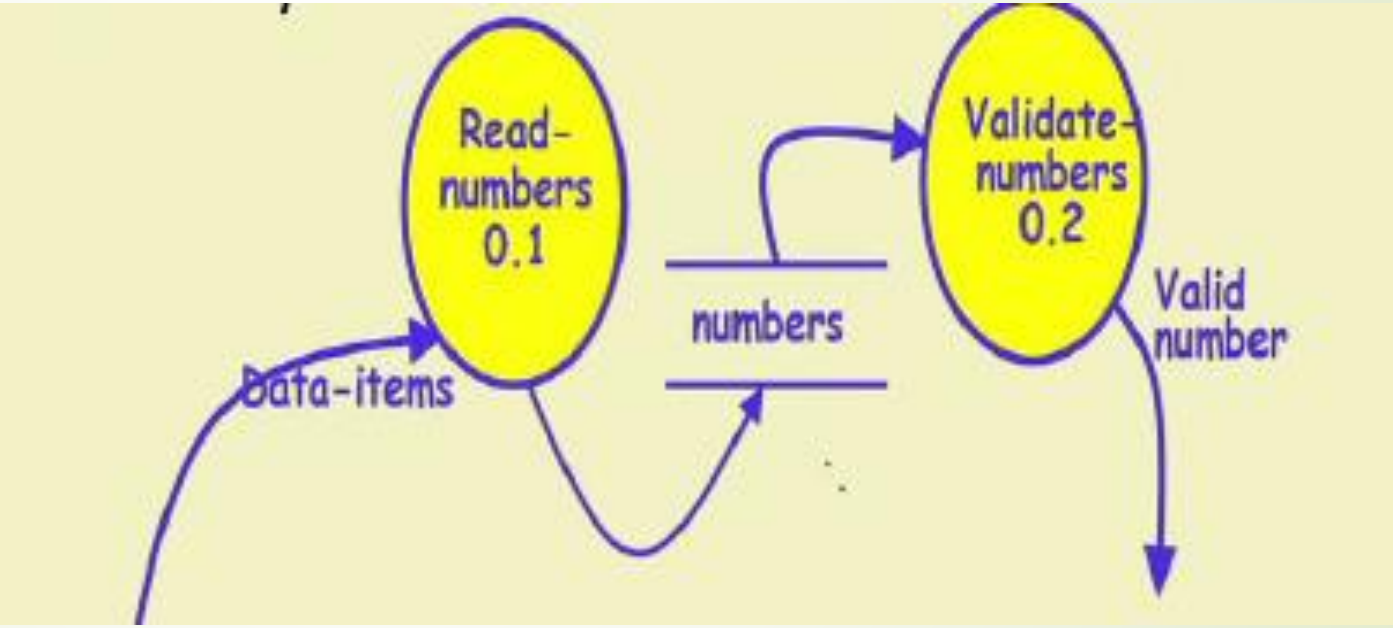
Synchronous Operation

- If two bubbles are directly connected by a data flow arrow, they are synchronous.

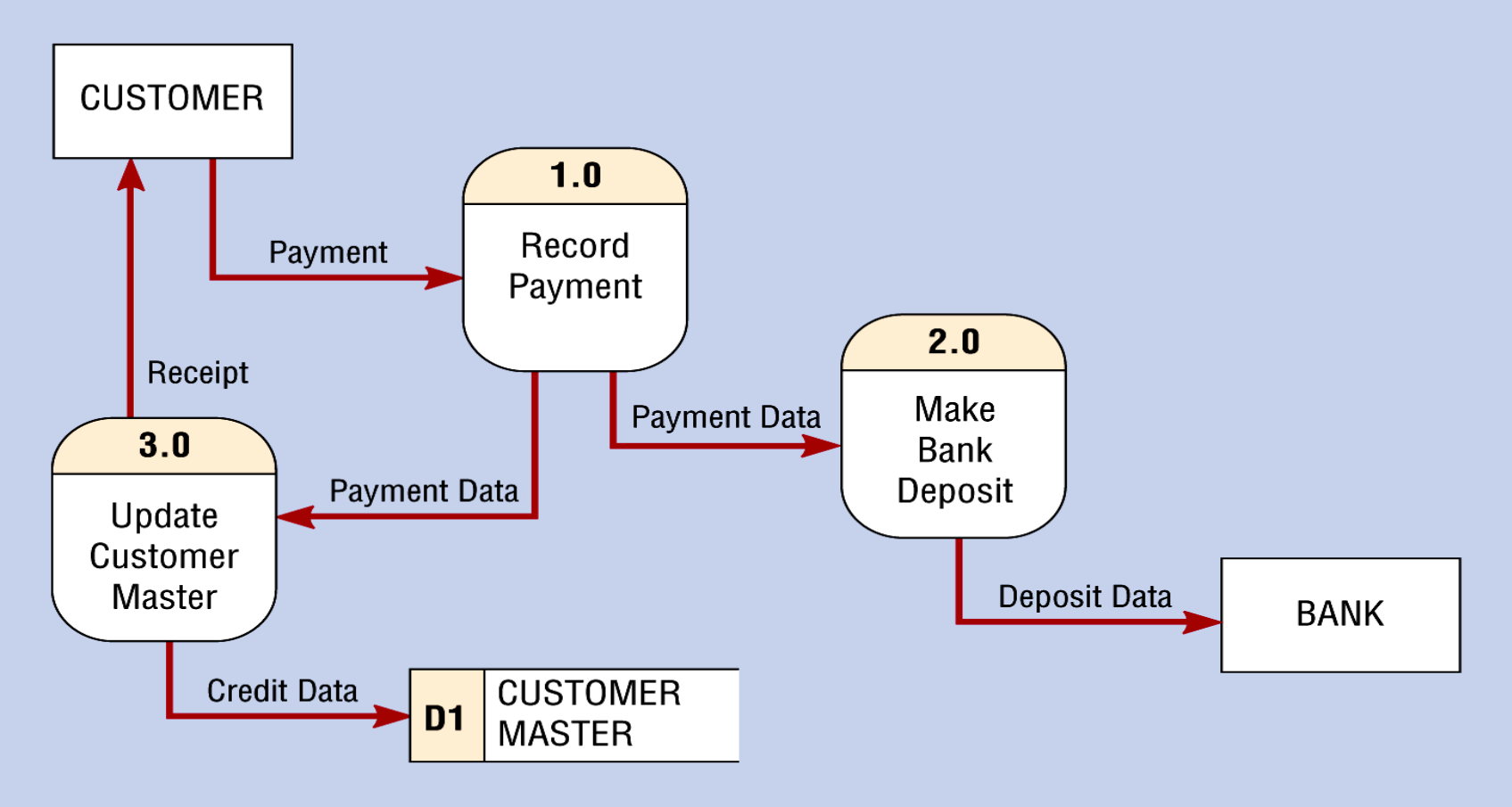


Asynchronous Operation

- If two bubbles are connected via a data store, they are not synchronous.

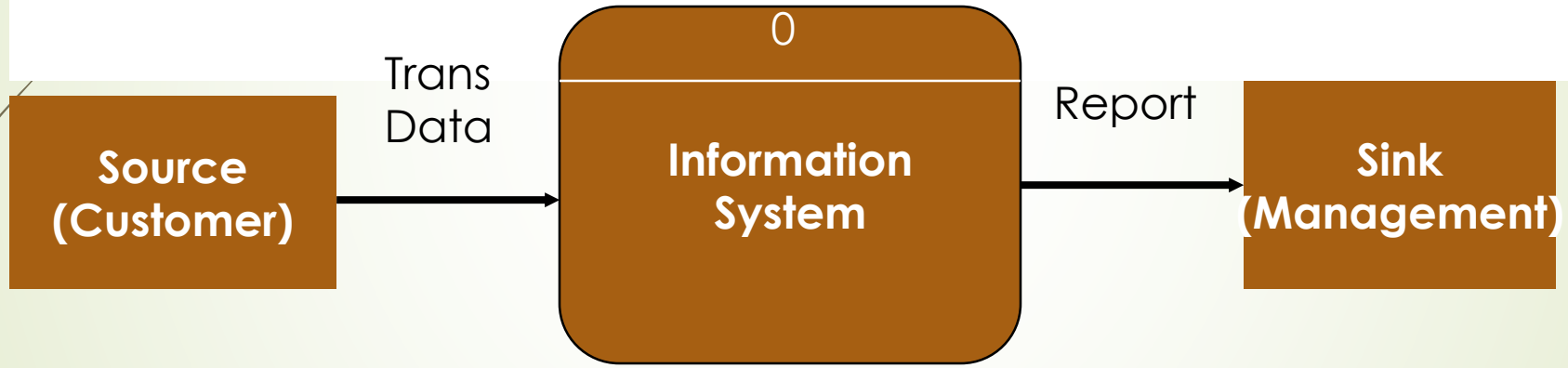


An Example - A Data Flow Diagram for a Banking System



An Information System : A Generic View

In general, a system could be viewed as a single Process



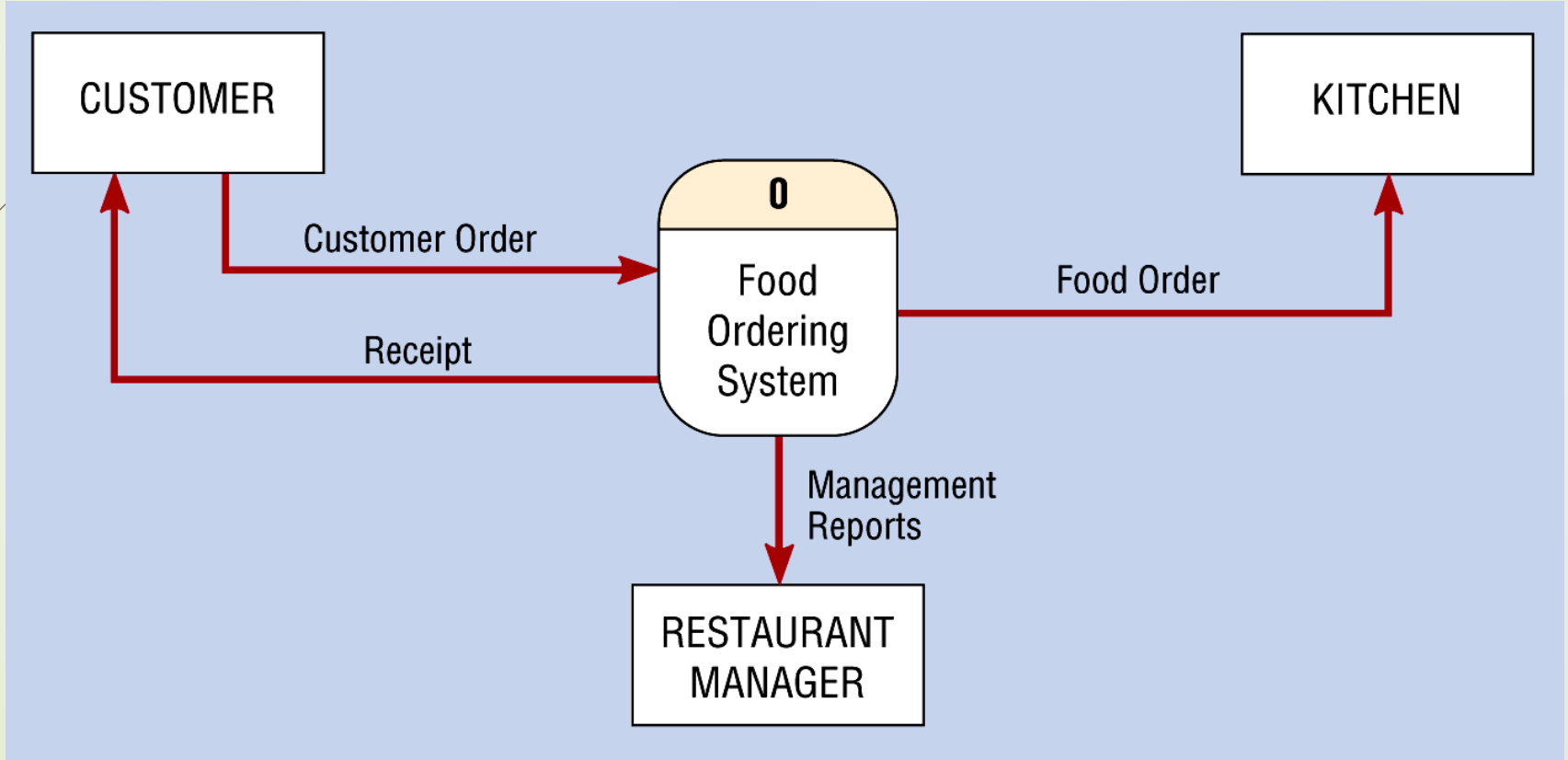
There can be multiple sources and sinks!

This generic diagram is called "**Context Diagram**"

A Context Diagram

- An overview of an organizational system that shows the system boundary, sources / sinks that interact with the system, and the major information flows between the entities and the system
- The context diagram establishes the context in which the system operates; that is, who are the users, what data do they input to the system, and what data they receive from the system.
- A Context Diagram addresses only one process.
- An example ...

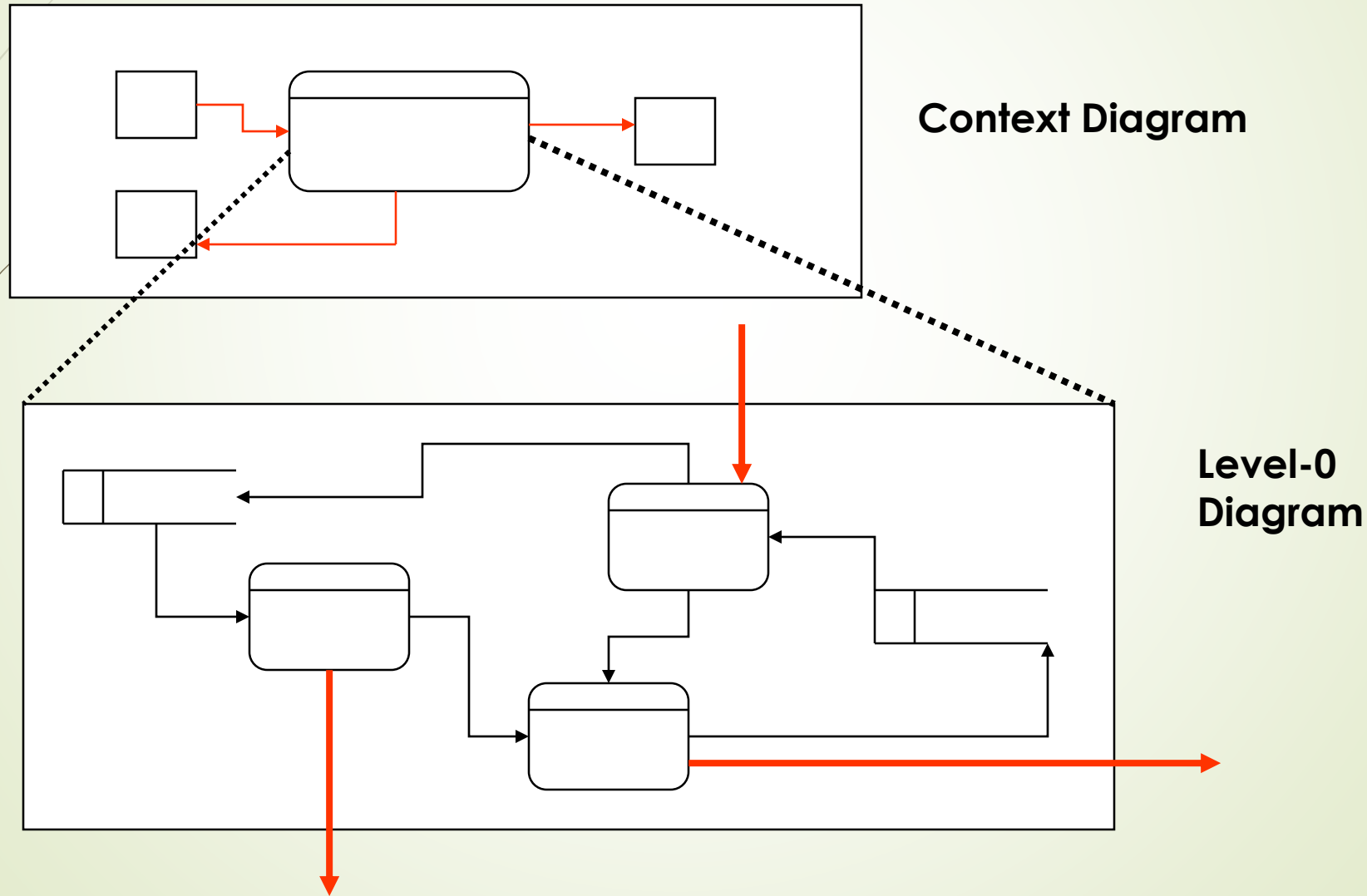
An Example - A Context Diagram for a Fast-Food IS



Process Decomposition

- In general, a system could be too complex to understand when viewed as a single Process
- We need a Process Decomposition scheme
  - i.e., to separate a system into its subsystems (sub-processes), which in turn could be further divided into smaller subsystems until the final subsystems become manageable units (i.e., primitive processes!)
- **A divide and conquer strategy!!**

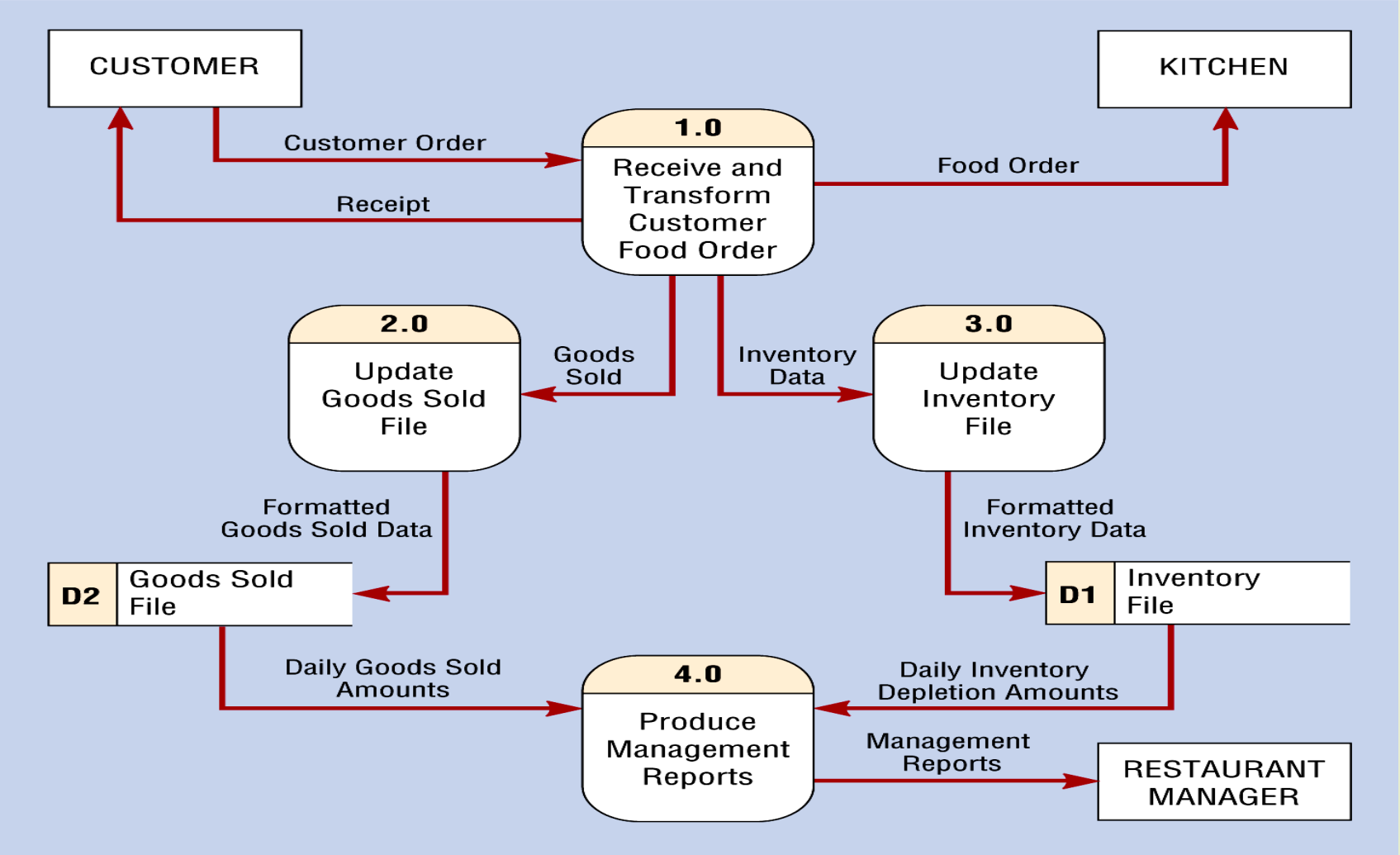
Decomposition Overview



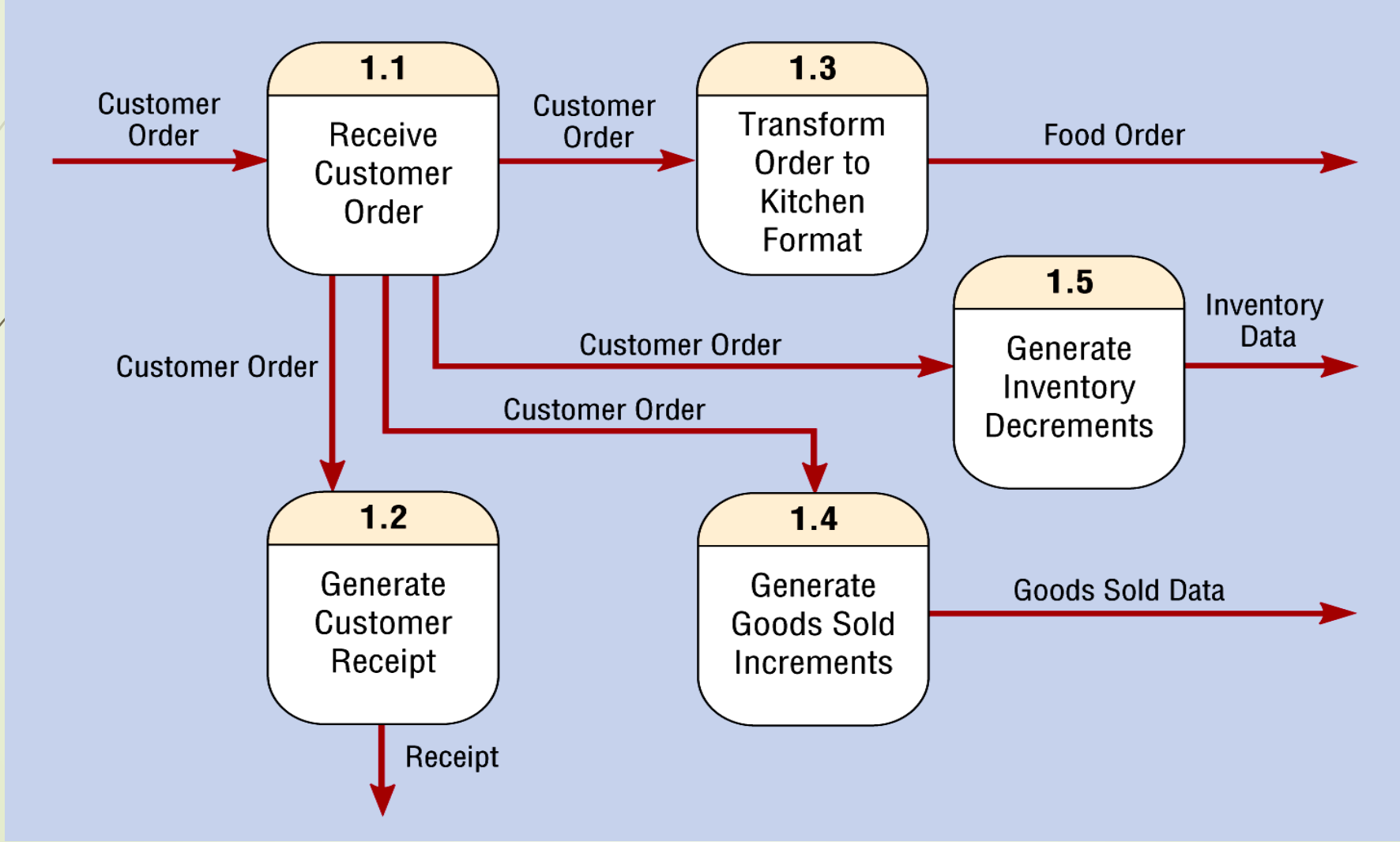
Level-0 Diagram

- A DFD that represents the primary functional processes in the system at the highest possible level
- An Example ...

An Example - A decomposed Context Diagram - Level 0 Diagram



An Example - A further decomposition
A Level-1 Data Flow Diagram



Process Decomposition Rules

- Generic Decomposition Rules:
  - A process in a DFD could be either a parent process or a child process, or both.
  - A parent process must have two or more child processes.
  - A child process may further be decomposed into a set of child processes.

Three Major Types of Process

- **Function Process**
  - A function is a set of related activities of the business (e.g., Marketing, Production, etc.)
- **Event Process**
  - An event process is a logical unit of work that must be completed as a whole. (e.g., Process customer credit verification)
- **Primitive Process**
  - a primitive process is a discrete, lowest-level activity/task required to complete an event. (e.g., Check the credit card balance)

Naming Rules for Processes

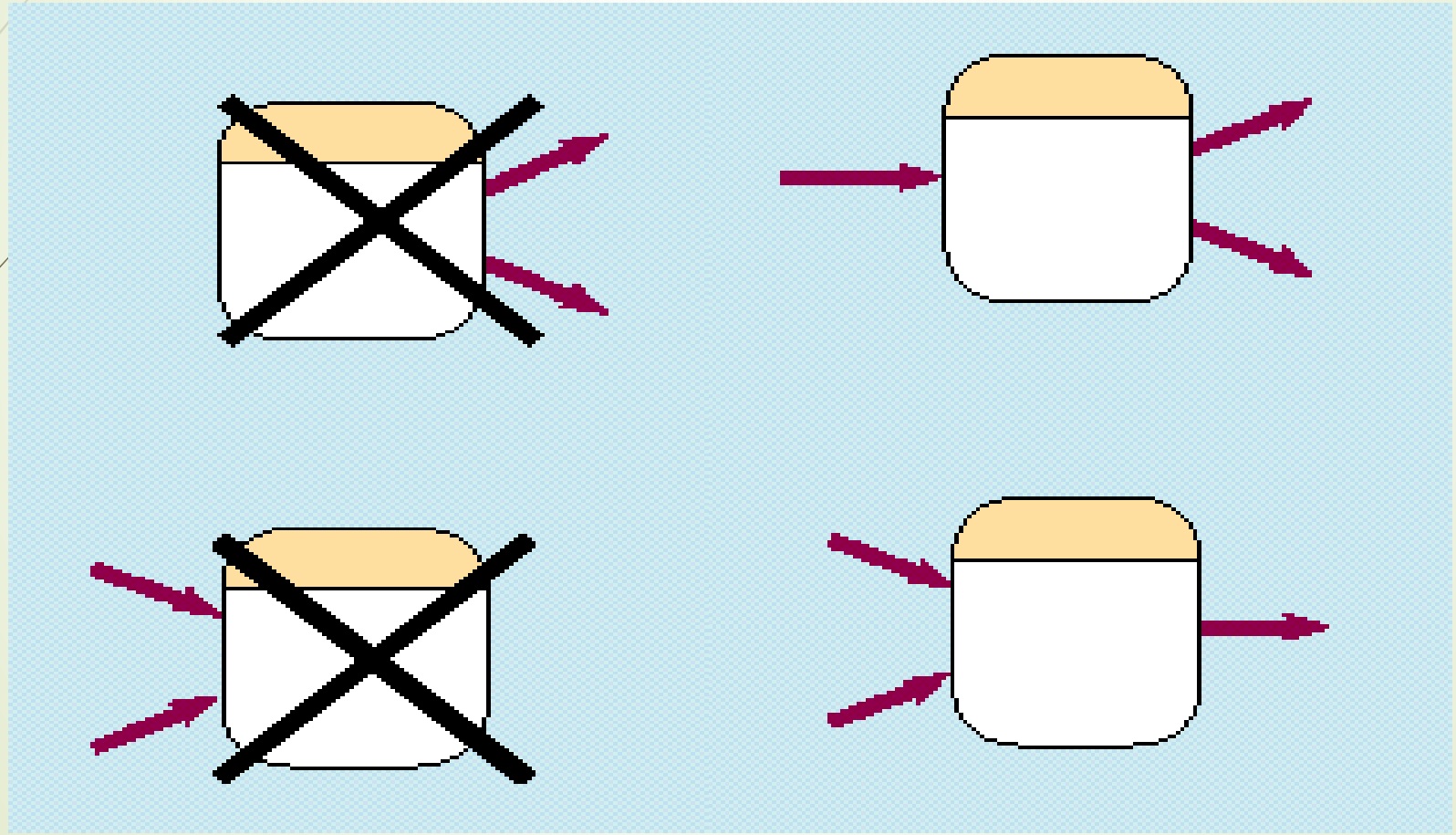
- Function Process - use a Noun
- Event Process - Use a general action verb
  - Process Student registration.
  - Respond to ...
  - Generate ...
- Primitive Process - use a strong action verb
  - Validate Student ID
  - Check ...
  - Calculate ...

Rules for Processes

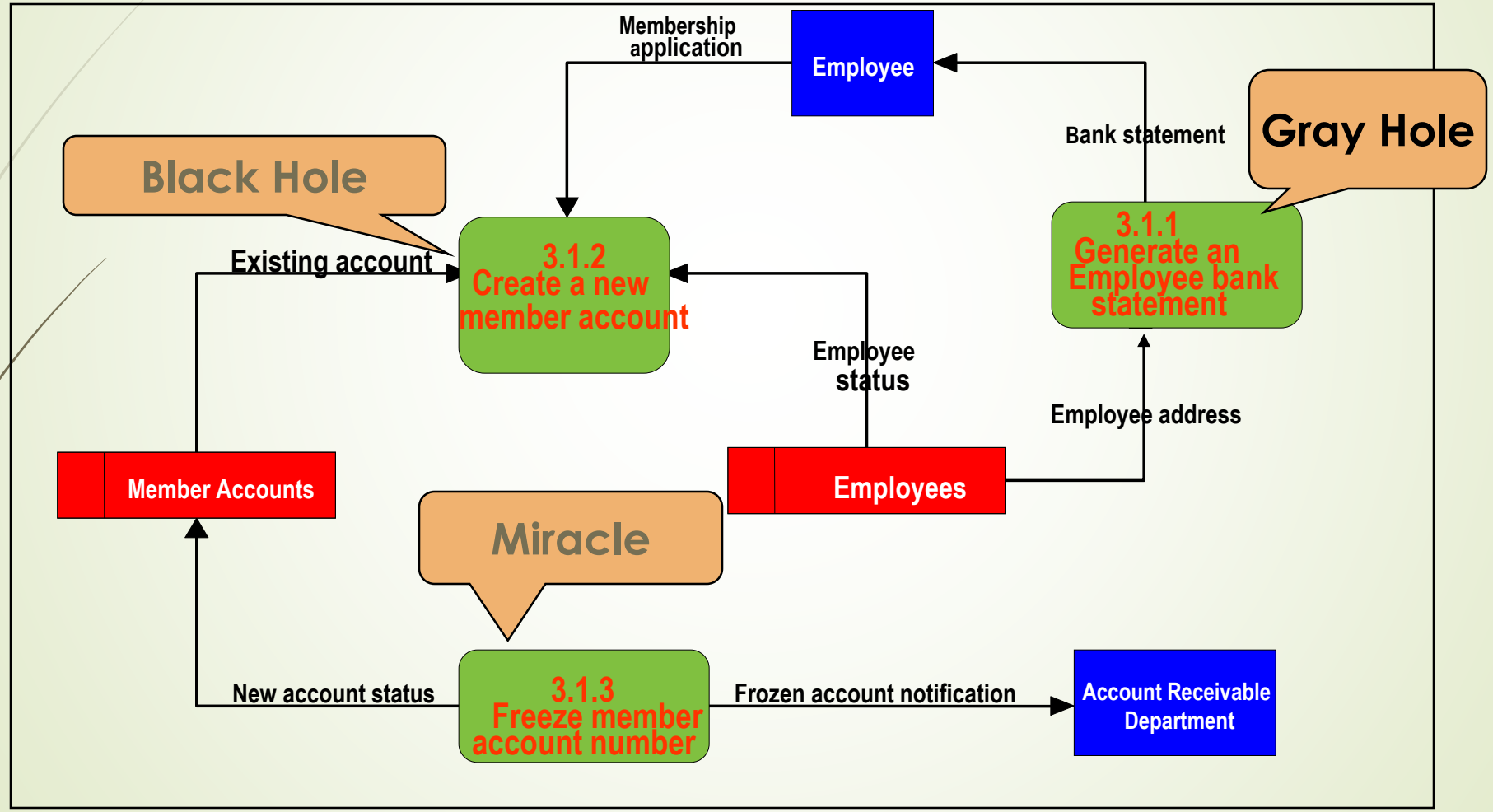
- No process can have only outputs (a miracle!)
- No process can have only inputs (a black hole!)
- No process can produce outputs with insufficient inputs (a gray hole!)

Processes in a DFD- Correct vs. Incorrect

Incorrect **Correct**



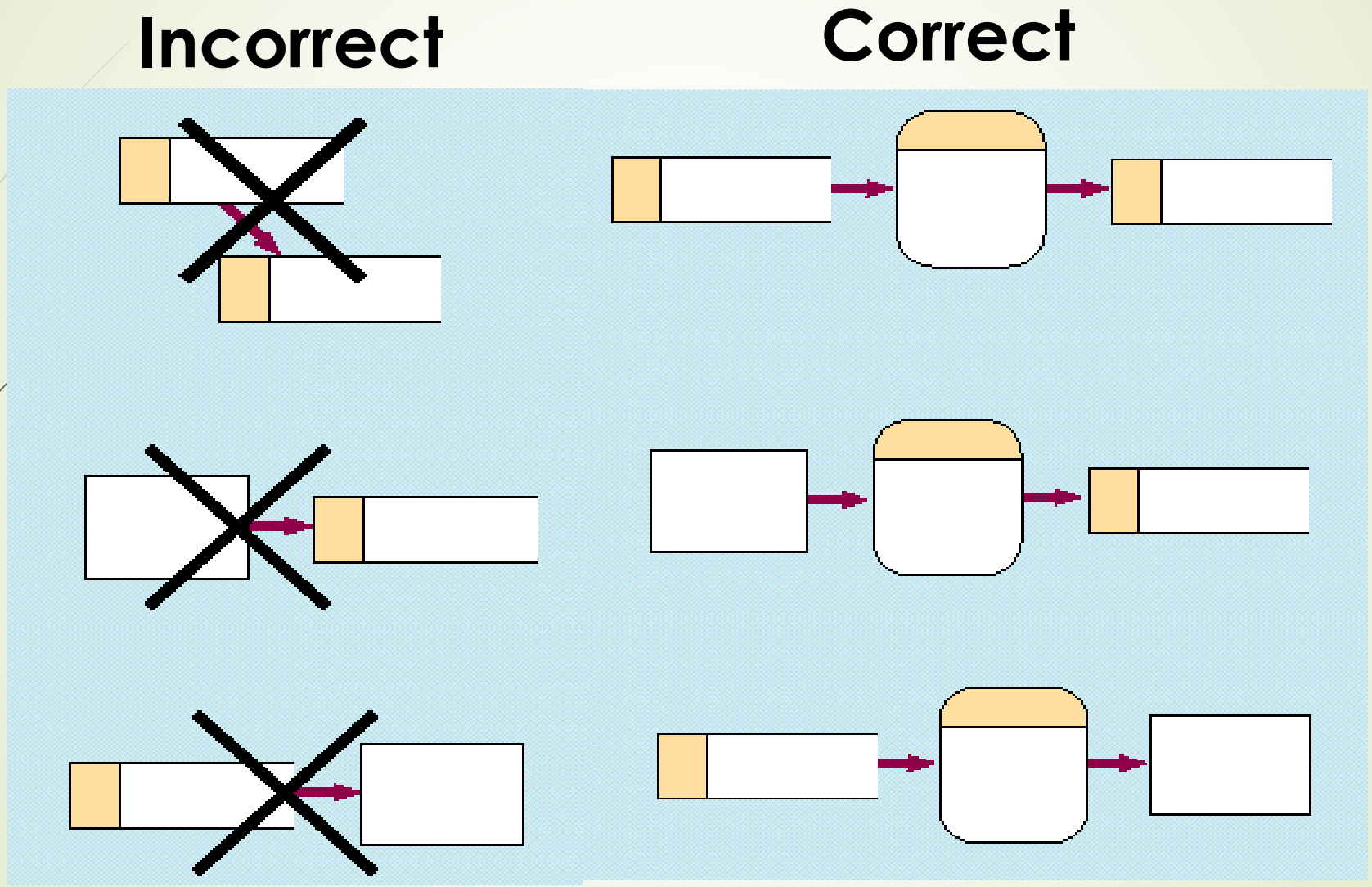
Can You Identify Errors in This Diagram?



Rules for Data Stores

- Data cannot move directly from one data store to another data store
  - it must be moved by a process.
- Data cannot move directly from an outside source to a data store
  - it must be moved by a process.
- Data cannot move directly to an outside sink from a data store
  - it must be moved by a process.
- You need to use a Noun phrase to label each data flow

Data Flows in a DFD: Incorrect vs. Correct



Naming Rules Data Flow

- Use a *singular noun phrase* for each data flow
  - Ex: customer data, shipping report, …, etc.
- Carry logical meaning only,
  - i.e., *no implication on data form* or *data structure*
- Minimum flow (no data flooding!!)
- Should *never be "Unnamed!!"*
  - otherwise, there might be a modeling error.

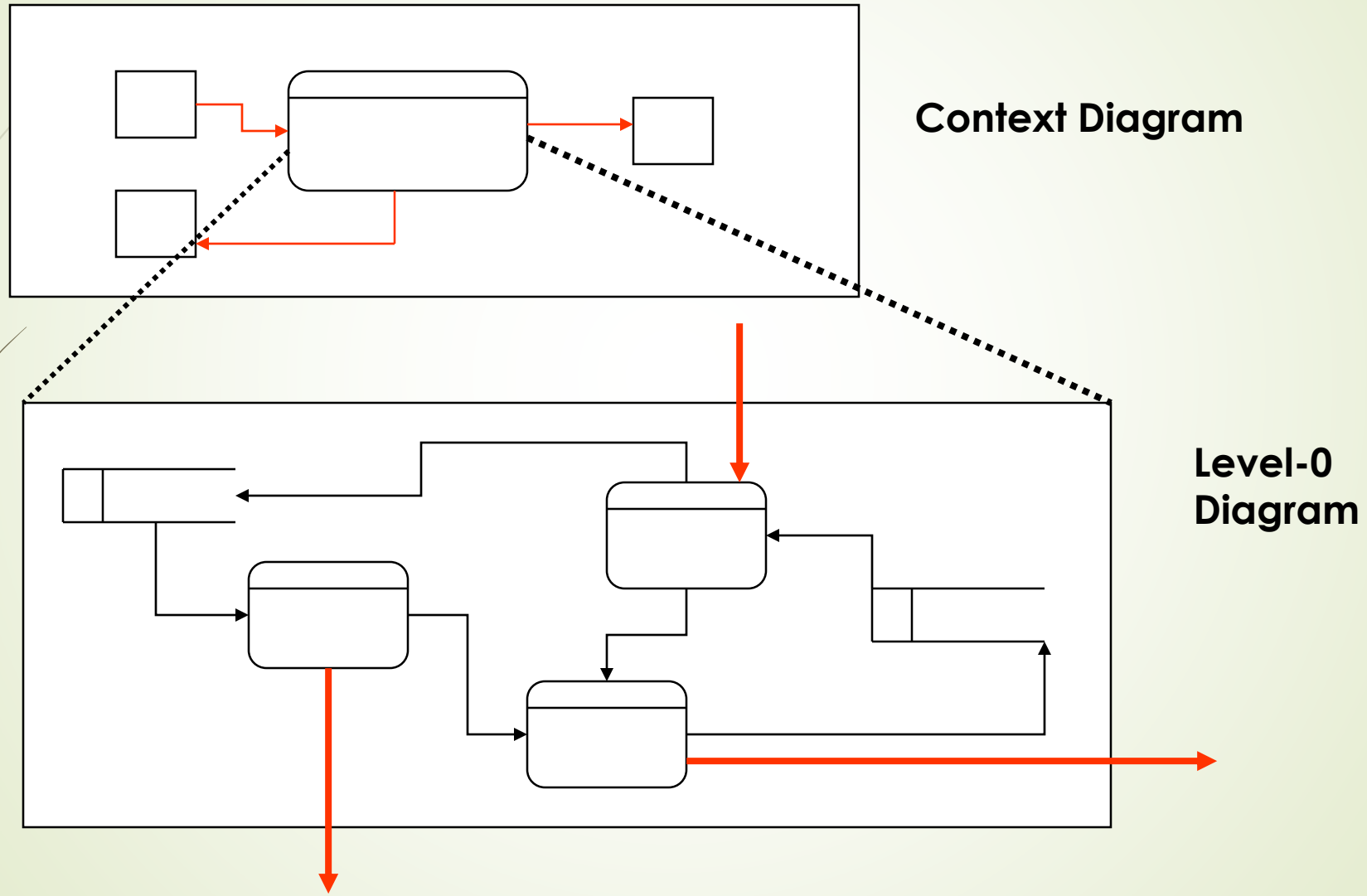
Naming Scheme for Other DFD Components

- **Process (Event)**
  - Use an Action Verb Phrase
  - Process member order, Generate bank statement, ...
- **External Agent (Sink/Source)**
  - Use a singular descriptive noun
  - Ex: Student, Customer, etc.
- **Data Store**
  - Use a plural descriptive noun (Members, Customers, etc.)
  - Or use a noun + file (Inventory file, Goods sold file)

Basic Rule in DFD Decomposition

- Balancing Principle
  - the decomposed DFD (i.e., the next lower level DFD) should retain the same number of inputs and outputs from its previous higher level DFD (i.e., No new inputs or outputs when a DFD is decomposed)

Basic Rule in DFD Decomposition…



Thank You.