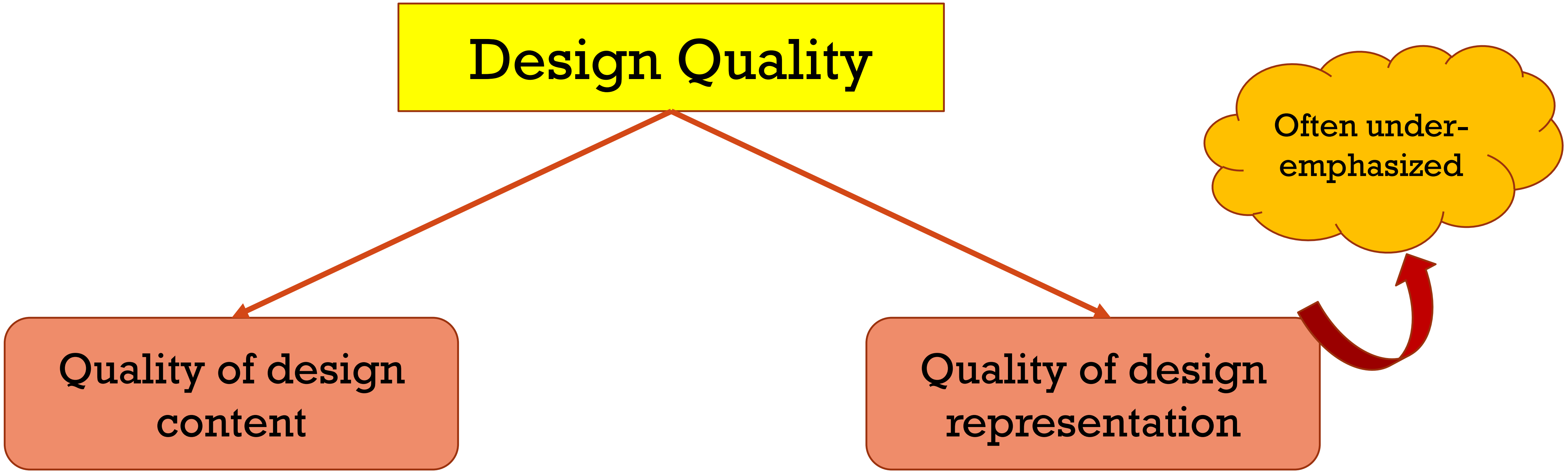
SOFTWARE PROJECT DESIGN

Yesoda Bhargava

INTRODUCTION

- Quality of software design is important because high quality implementation from poor design quality is impossible.



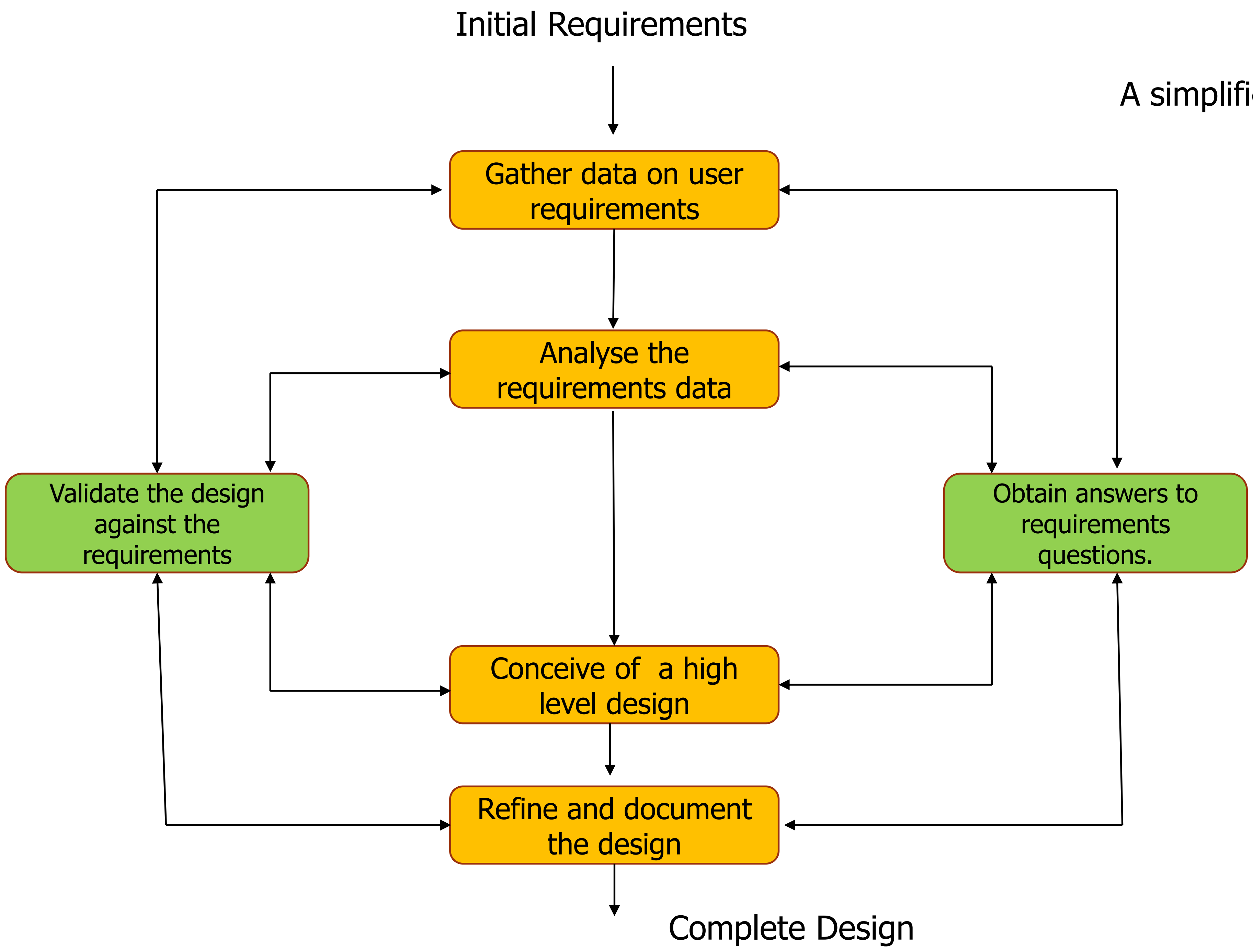
DRAWBACKS OF POOR DESIGN REPRESENTATION

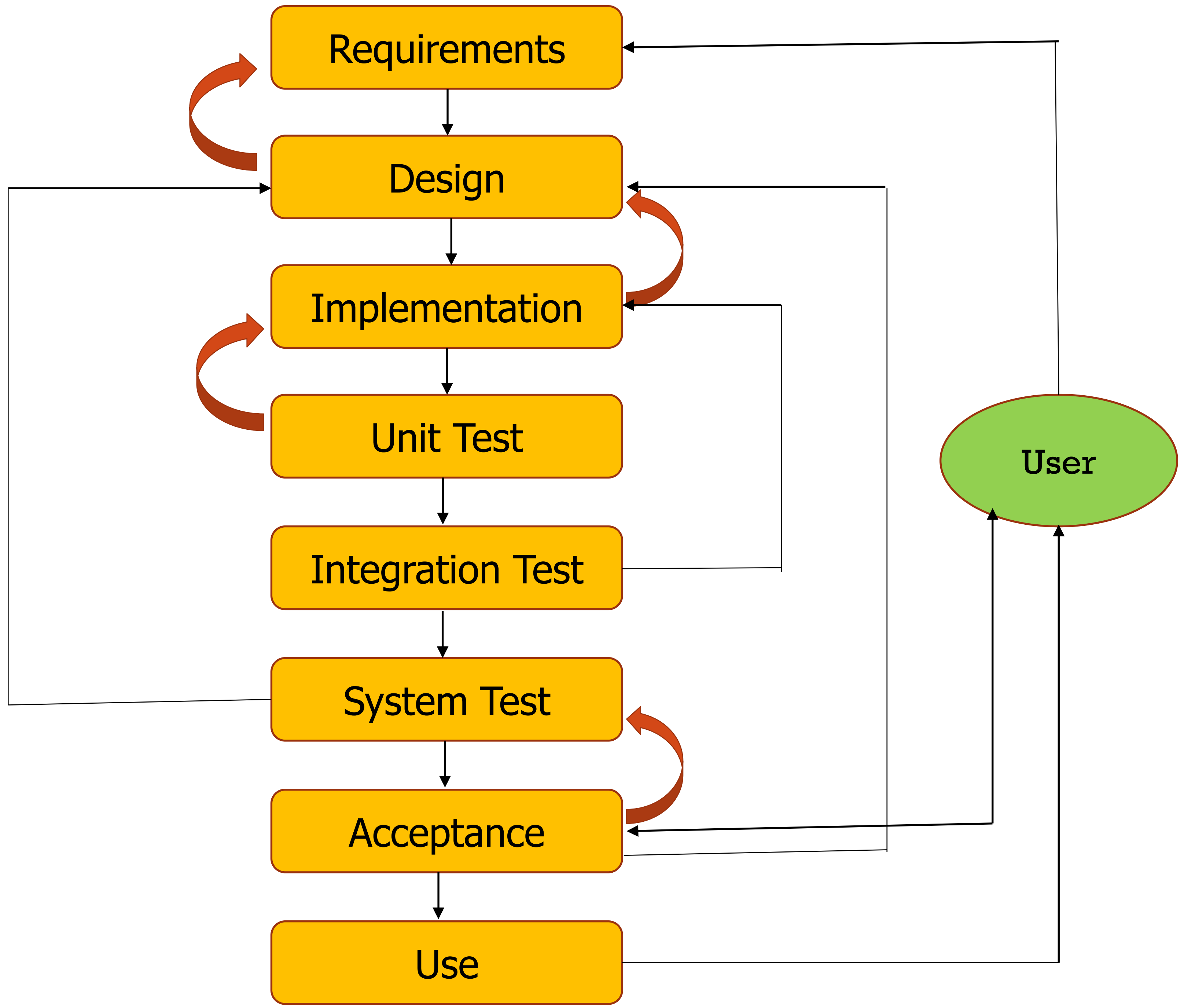
- Design with high quality content likely will be poorly implemented if they are badly represented.
- Poor representation makes design hard to understand, and problems will not be recognised until implementation or even later.

THE DESIGN PROCESS

- Software design is a creative process that cannot be reduced to routine procedure.
- It is a highly structured process.
- General trajectory : defining product's purpose, gathering relevant data, producing an overview design, and then filling in the details.

A simplified design process.


Initial Requirements
Gather data on user requirements
Analyse the requirements data
Validate the design against the requirements
Obtain answers to requirements questions.
Conceive of a high level design
Refine and document the design
Complete Design


Requirements
Design
Implementation
Unit Test
Integration Test
System Test
Acceptance
Use
User

THE CONCEPTUAL DESIGN

- At the outset, the initial requirements of software project are incomplete and imprecise.
- Get enough information before you start designing.
- Follow this approach:
  - Focus on the high-level issues until you know enough to produce the overall conceptual design.
  - Complete and document the conceptual design.
  - Once you have satisfied yourself that the conceptual design is solid, you can begin to focus on details.
  - The conceptual design now provides the framework for cataloguing the knowledge you gain and ensuring you do not lose important details.

DESIGN QUALITY

- Should be sufficiently complete, accurate, and precise solution to a problem in order to ensure its quality implementation.
- **DESIGN COMPLETENESS**
  - One can waste time over-specifying the design; however under-specified design will be expensive and error prone to implement.
  - Complete specification of a software design requires a great deal of information.
  - Includes defining classes and objects and their relationships, identifying the interactions among them, defining the required data and state transformations, and specifying the system inputs and outputs.
  - Complete and unambiguous definition of all this generally requires significantly more documentation than the source code.
  - Source code includes all design decisions but in a very terse form.
  - But do not spend more time over-specifying the design.

- **THE USERS' NEEDS**
- Consider the needs of those who will use the design.
- The principal users are the implementers, the design and code reviewers, the documenters, the test developers, the testers, the maintainers and enhancers.
- **<u>YOU ARE NOT THE ONLY DESIGN USER.</u>**

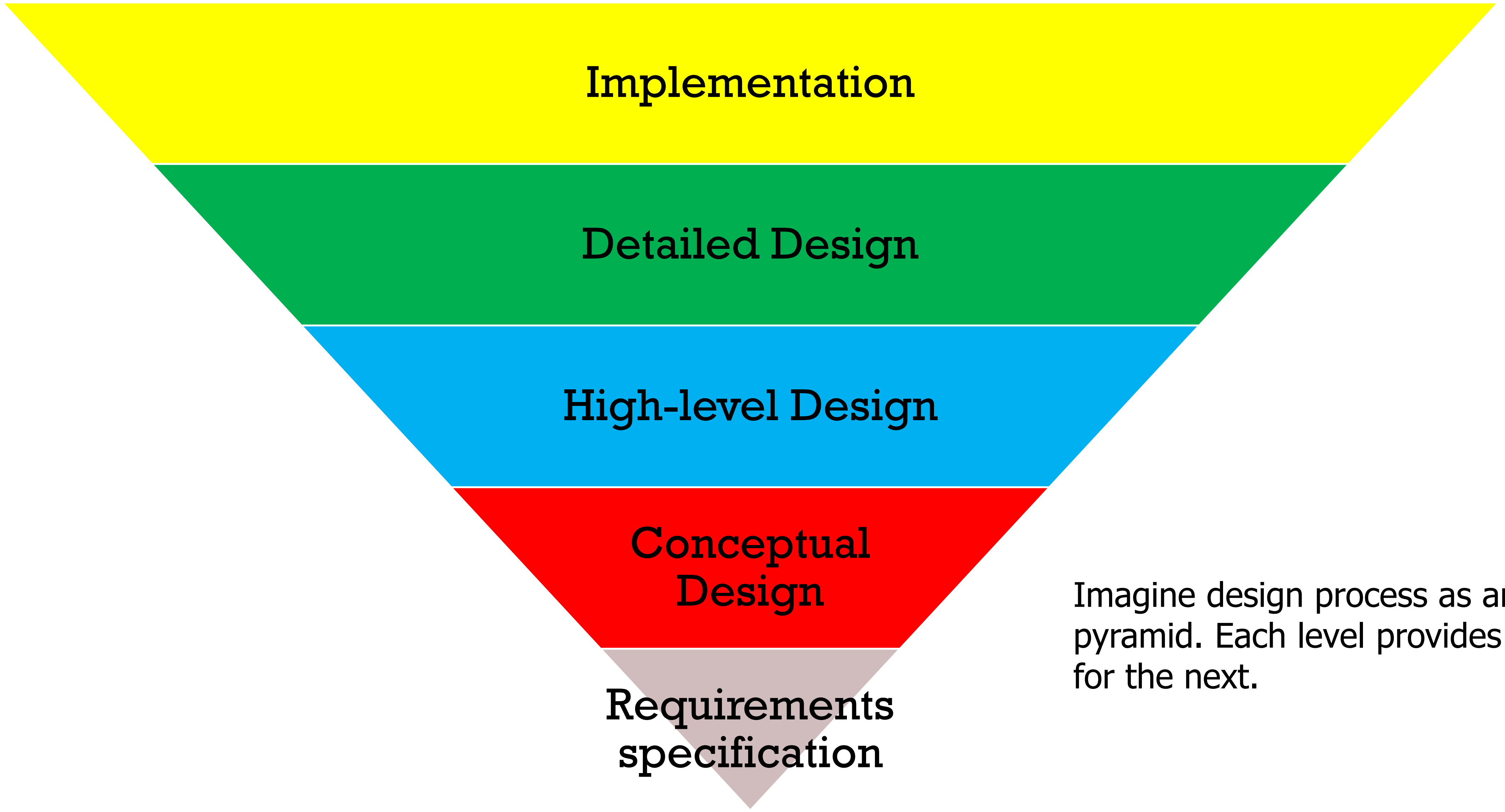
RESPONSIBILITY OF SOFTWARE DESIGNERS

- A picture of where the program fits into the system.
- A logical picture of the program itself.
- A list of all related objects: scope, class structure, functions provided, where initialized, where destructed.
- A list of all external variables: including scope, limits, constraints, where declared, where initialized.
- A precise description of all external calls and references.
- A clear statement of the program’s logic (pseudocode).

DESIGN PRECISION

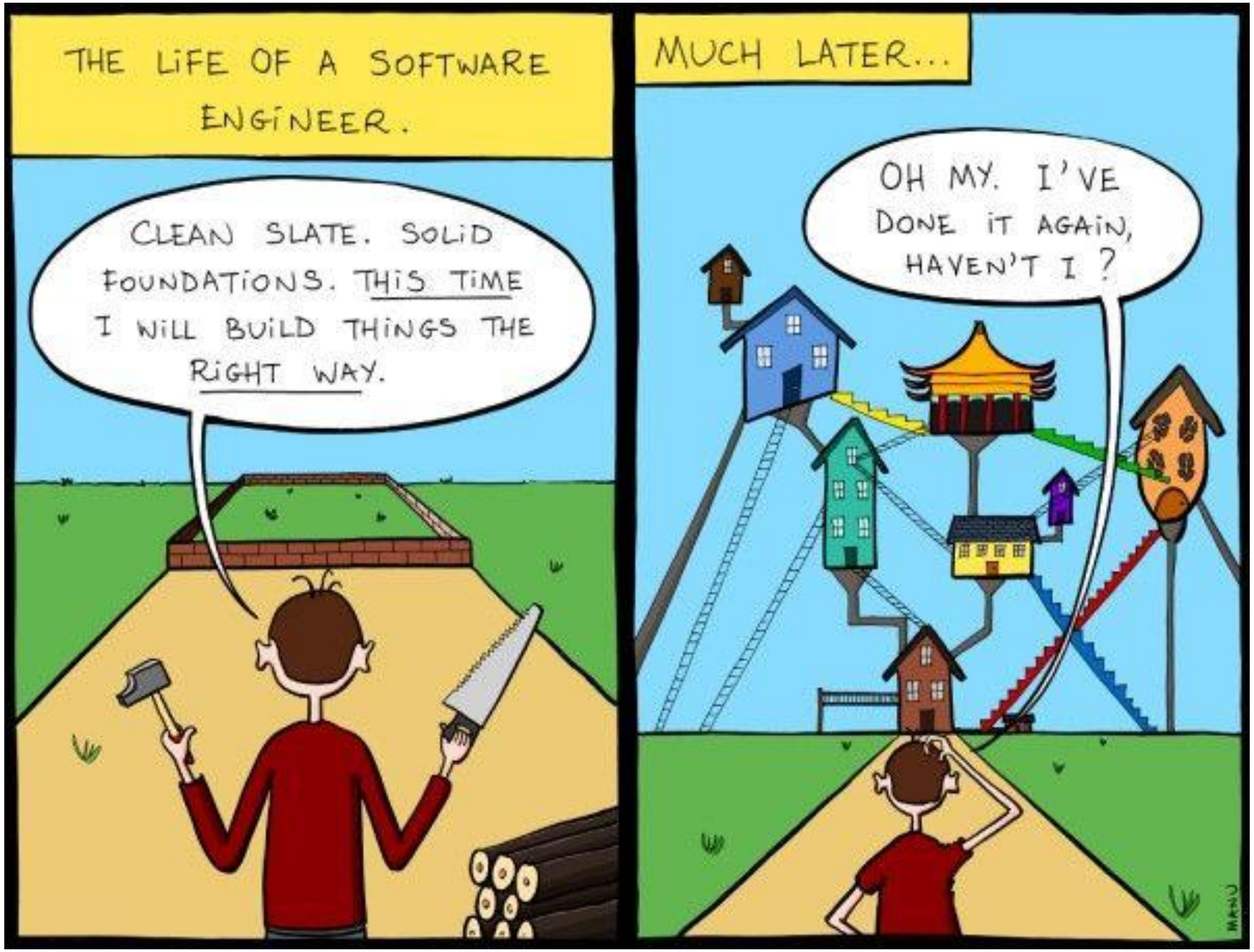
- Lack of a precise design is the source of many implementation errors.
- If you do not have a precise specification for what the design must contain, your decisions on where to stop will generally be inconsistent.
- “Anyone would know how to do this” and “let’s not bother to document it”.
- Imprecise design leads to finishing that design during implementation and this can be highly error prone.

DESIGN LEVEL



Imagine design process as an inverted pyramid. Each level provides a foundation for the next.

If you do not make and document all the necessary high-level design decisions in high level design you will have to reconstruct them again during detailed design, you will have to reconstruct them in implementation.

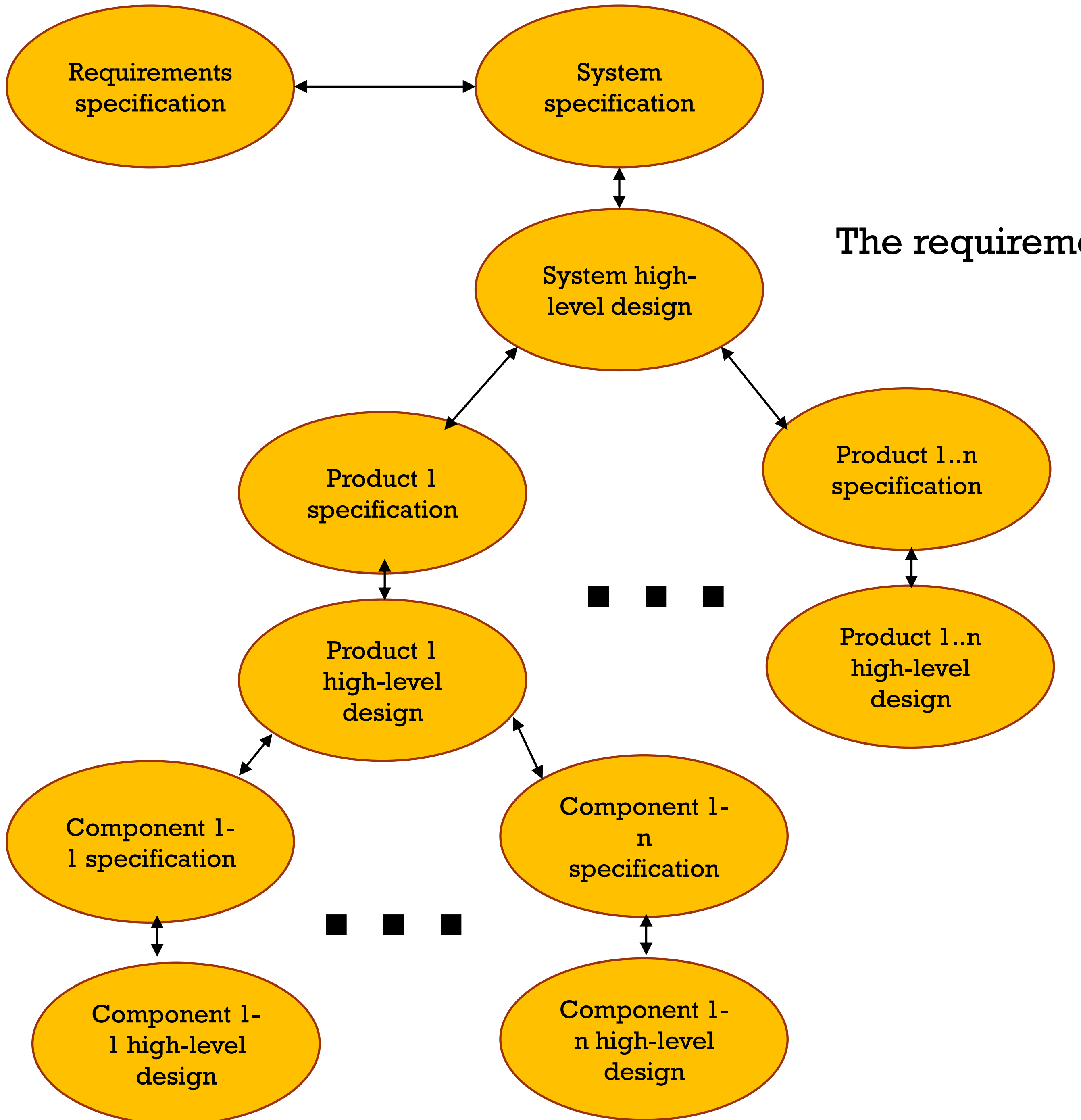

THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?
MANU

- Even when you are the implementer, the process of reconstructing all your mentally completed but not documented design can be both time consuming and error prone.
- At the implementation stage, you are thinking at an implementation level.
- Without considerable effort and complete change in mindset, you will have trouble reconstructing all the relevant design details.
- This is when design errors are most likely.
- This appears at every stage.
- **Poll : Design process is a static/dynamic process.**

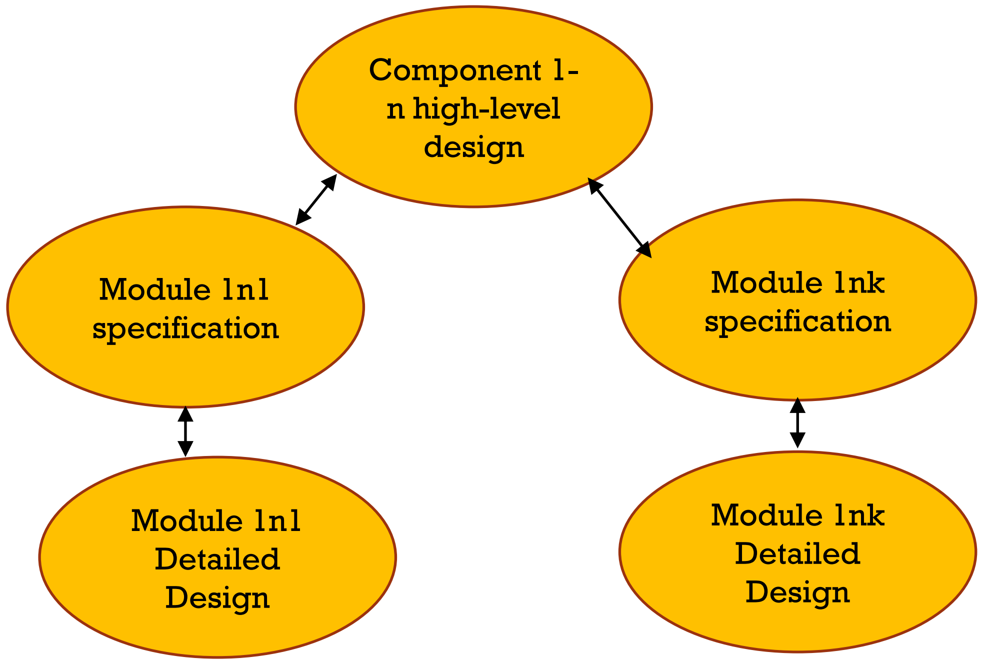
STRUCTURING THE DESIGN PROCESS

- Even though the design process is dynamic, it is helpful to think of it in process terms.
- But do not constrain the design behaviour, structure helps in bringing precision to the what and how of design process.
- You need a framework in order to maintain control of what you have done and to track your design status.

The requirements-specification-design cycle


Requirements specification
System specification
System high-level design
Product 1 specification
Product 1..n specification
Product 1 high-level design
Product 1..n high-level design
Component 1-1 specification
Component 1-n specification
Component 1-1 high-level design
Component 1-n high-level design

Contd…


Component 1-n high-level design
Module 1n1 specification
Module 1nk specification
Module 1n1 Detailed Design
Module 1nk Detailed Design

The requirements-specification-design cycle

DESIGN PHASES

- Define the need (requirements definition)
- Define the solution (system specification)
- Conceptualize the solution (the system high-level design)
- Divide up the work (the product specifications)
- Define the product design (product high-level design)
- Break the product into components (the components specification)
- Break the components into modules (module specifications)
- Detail the solution (module detailed design)
- Implement the solution (module implementation and test)

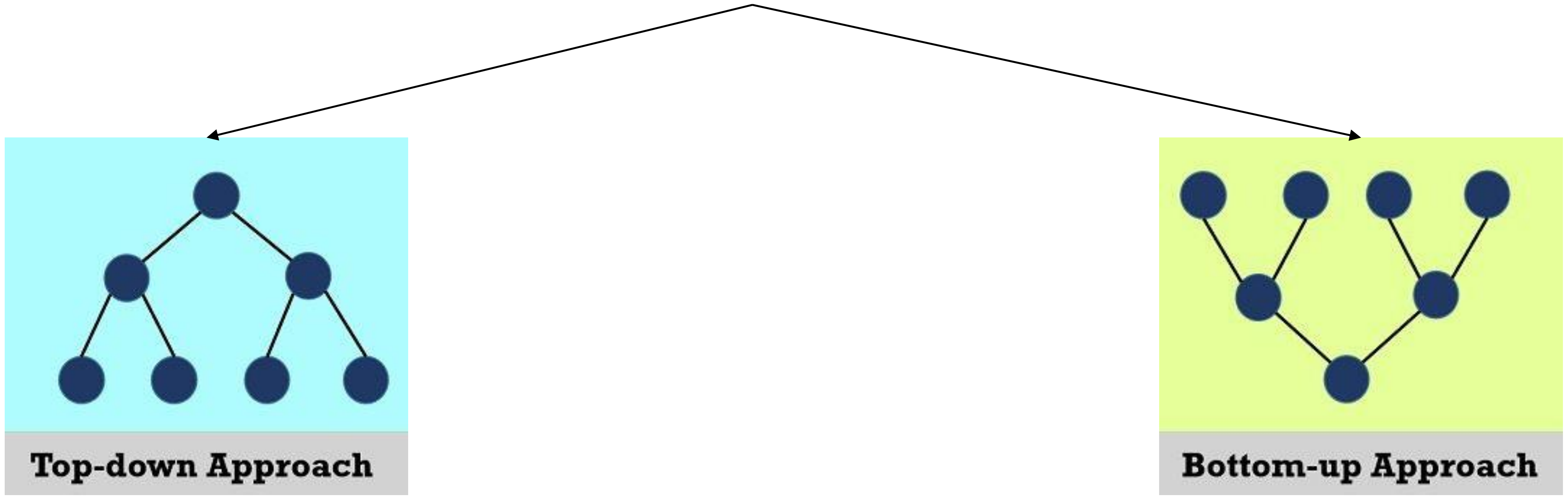
- The specifications and high level designs are progressively refined until the module level is reached.

DESIGN SPECIFICATION

- Is a complete and precise statement of what the program must do.
- Specifications tell about existence of the solution and its constraints and invariants.
- This is treated as a separate phase because it often involves multiple product layers.
- From the Requirements-Specifications-Design cycle it is clear that:
- As development proceeds the system is divided into major products, each of which must be specified and designed.
- Products may compose several components that must be specified and designed.
- Specification is refined for each product level.

A good design will resolve many issues.

SOFTWARE DESIGN APPROACH



Most-widely used.
Problem broken down into the major tasks.
Each task broken down into sub-tasks.
Until each task is simplified into a self-contained module.

When the problem is too large and complex it may be very difficult to decompose.
Much easier to attack parts of the problem individually.
Taking easier aspects first and gaining insights on how to solve the difficult tasks.
Piecing together of systems to give rise to more complex systems.

TOP-DOWN DESIGN

- Breaking down of a system to gain insight into its composition sub-systems.
- Overview of the system is formulated, specifying but not detailing, any first-level systems.
- Each subsystem is refined to greater detail, until the entire specification is reduced to base element.
- It starts with a big picture, breaks down from there into smaller segments.
- Emphasize planning and complete understanding of the system.
- No coding can begin until a sufficient level of detail has been reached in the design of at least some part of the system.

Top-down view

Suppose you wish to make pancakes.

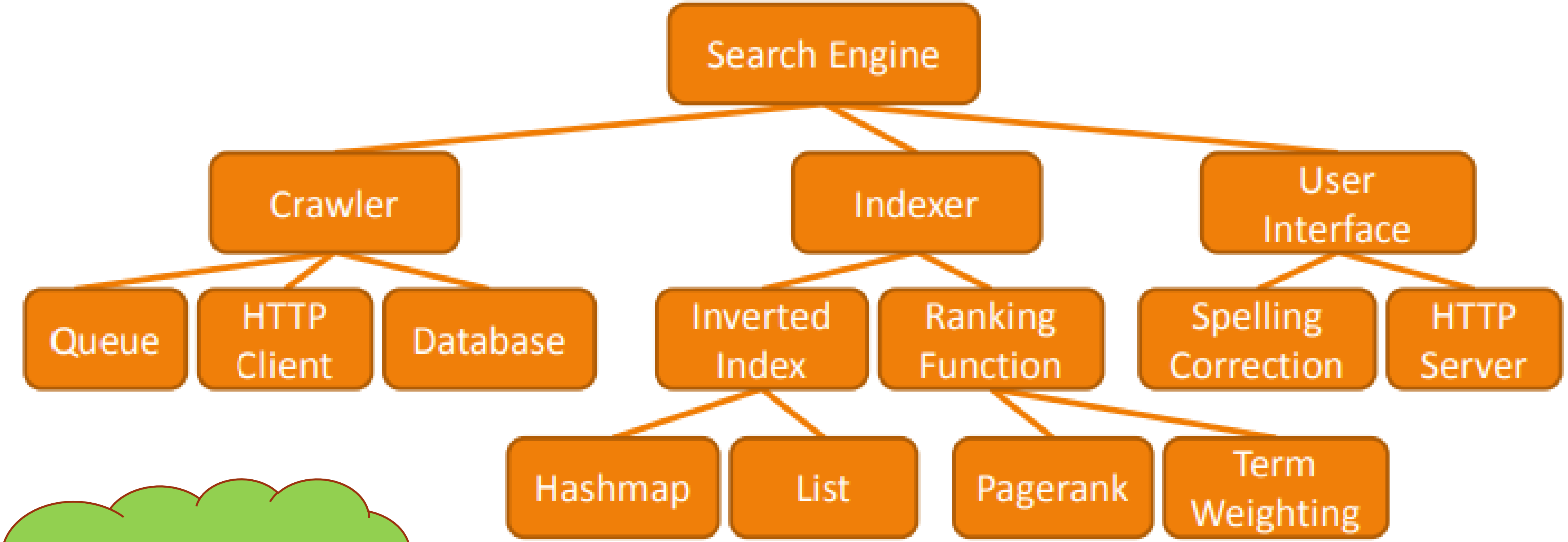
Make some pancakes

- Organize kitchen
  - Clean surfaces
  - Get out mixing bowl, whisk, spoon, sieve
  - Get out plain flour, salt, eggs, milk, butter
  - Put on apron
- Make pancakes
  - Sift salt and flour into bowl.
  - Breaks eggs into bowl.
  - Whisk
  - Add water ad milk
  - Add butter
  - Whisk
  - Cook
    - Get pan to room temperature.
    - Pour batter.
    - Spread batter to edges.
    - Check bottom of pancake using spatula.
    - When brown, flip.
    - Check bottom of pancake again.
    - When brown finish.
- Serve

ADVANTAGES OF TOP-DOWN DESIGN

- **Broader perspective**: If not viewed from the top, the information system may be implemented without first understanding the business from the general management viewpoint.
- **Improved integration:** helps in planning and designing such that the existing system can be evolved rather than just being implemented.
- **Improved management support**: If not viewed from the top, planners may lack sufficient management acceptance of the role of information system in helping them to achieve business objectives.
- **Better understanding:** Viewing from top helps planners and designers to implement the information system across the entire business rather than simply to individual operating units.

Top-Down Design for building a Search Engine



Refine the design at each step.

Decomposition/ Divide and Conquer"

Each separate sub-task is expanded and defined until the problem is solved. Each sub-task is tested and verified before it is expanded further.

- The top down approach starts from the general and moves to the specific.
- Basically, you start with a general idea of what is needed for the system and then ask the end-users what they need specifically.

BOTTOM-UP DESIGN

- Piecing together of systems to give rise to more complex systems.
- The individual base elements of the system are first specified in great detail.
- Later linked together to form larger subsystems.
- The beginnings are small but grow in complexity and completeness over time.
- Emphasizes coding and early testing.
- Modules may be coded w/o having a clear idea of how they link to other parts of the system.
- More suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Building blocks are an example of bottom-up design because the parts are first created and then assembled without regard to how the parts will work in the assembly.

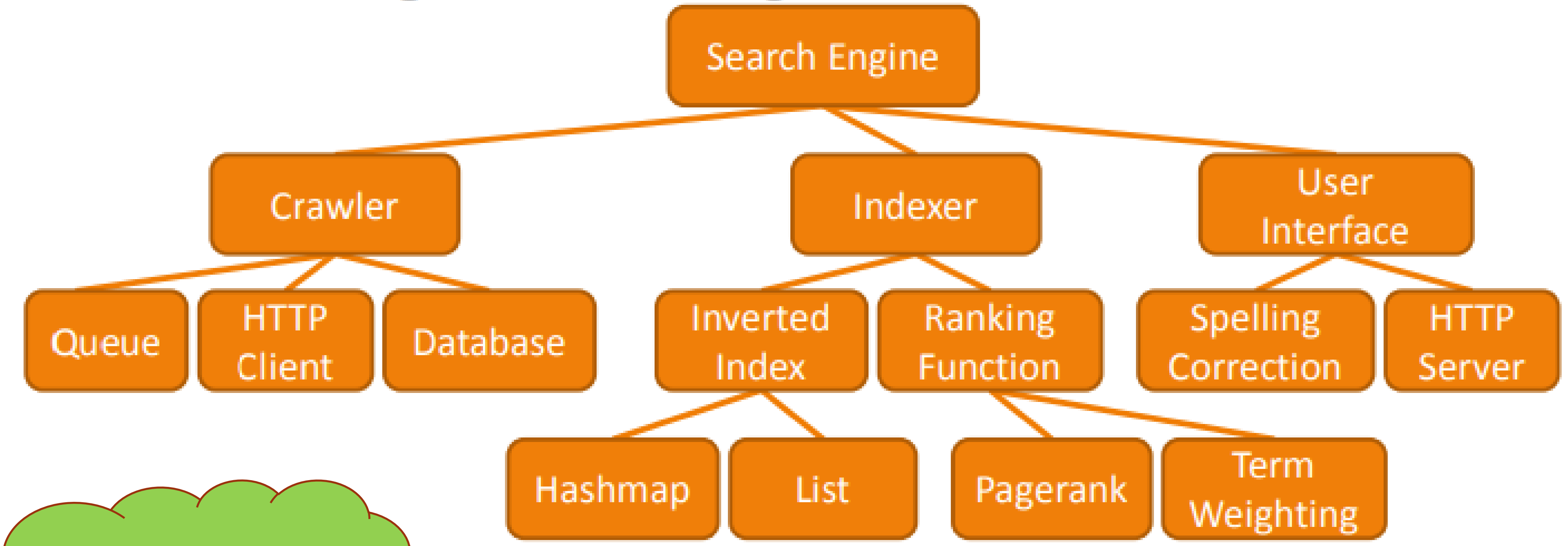
ADVANTAGES & DISADVANTAGES

- Bottom-up design can help to attack parts of the problem individually, taking the easier aspects first thereby gaining the insight and experience to tackle the more difficult tasks, finally integrating them together to form the complete solution.
- Can be faster and less costly to develop than using a top-down approach.
- Lends itself to more experimentalism.
- The disadvantage of resulting in creation of disparate information system and databases that are redundant or not easily integrated without substantial network.
- Suffers from the disadvantage that the parts of the program may not fit together very easily.
- Lack of consistency between modules and considerable reprogramming may be needed.

EXAMPLE OF BOTTOM-TOP APPROACH

- In statistical analysis, analysts take a sample from a small population and then infer the results to the overall population.
- Doctors examine specific symptoms and then infer the general disease that causes those symptoms.
- IKEA furniture.
- Bottom-up approach begins with the specific details and moves up to the general.

Bottom-Up for building a Search Engine



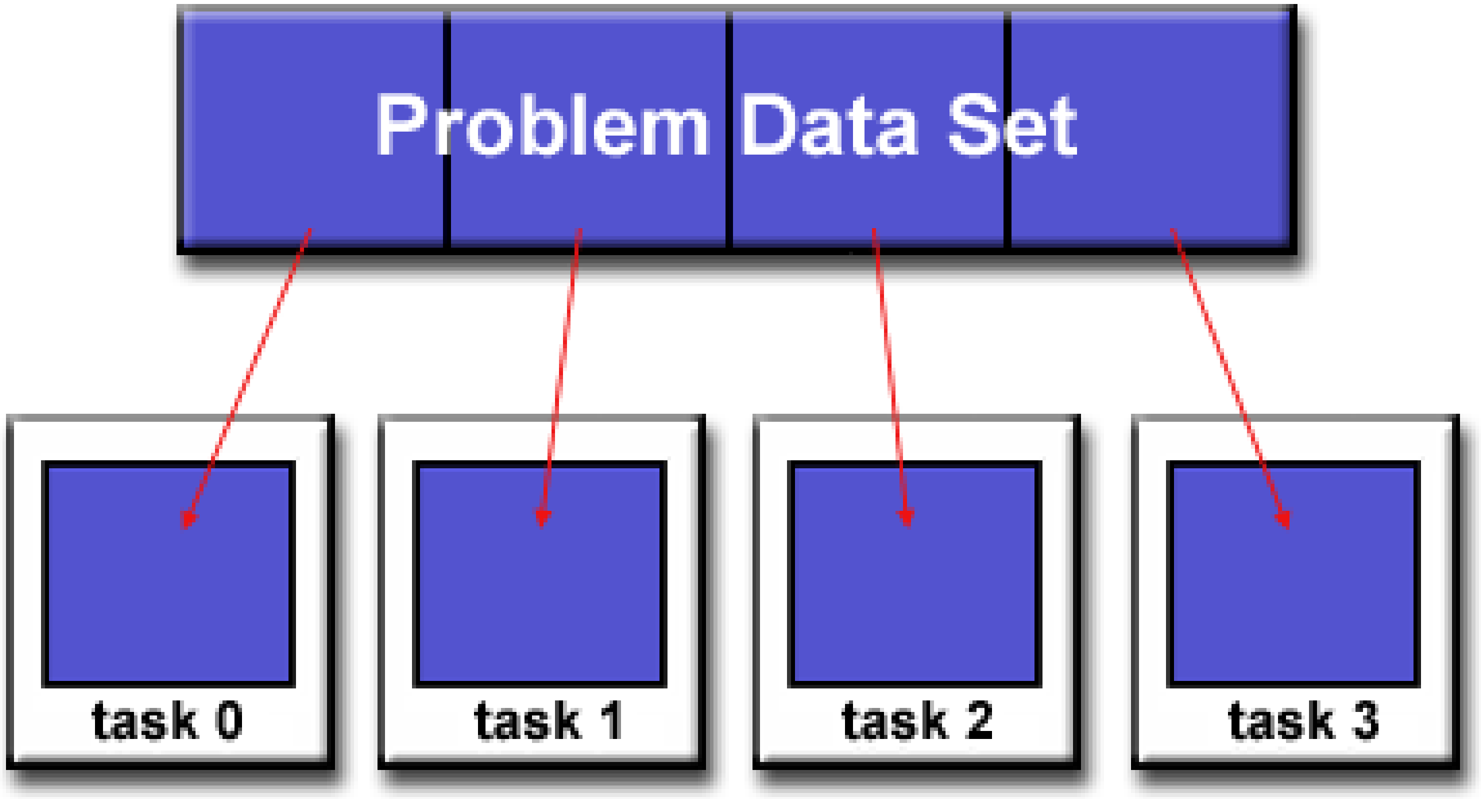
Start with parts.

Composition

Match the content to its proper place in the below table

Key	Top-Down Model	Bottom-Up Model
Focus	Focus is on breaking the bigger problem into smaller one and then repeat the process with each problem.	Identifying and resolving smallest problems and then integrating them together to solve the bigger problem.
Redundancy	High ratio of redundancy as the size of the project increases.	Better suited as it ensures minimum data redundancy and focus is on reusability.
Approach	Based on decomposition approach.	Based on composition approach.
Issues	May not be possible to break the problem into set of smaller problems.	Difficult to identify overall functionality of system in initial stages.

WHAT IF THE PROBLEM IS COMPLEX?



STRUCTURED ANALYSIS/STRUCTURED DESIGN (SASD)

1. Diagrammatic notation designed to help people understand the system.
2. Based on data flow diagram.
3. Focuses on well-defined system boundary.
4. SASD allows the analyst to understand the system and its activities in a logical way.

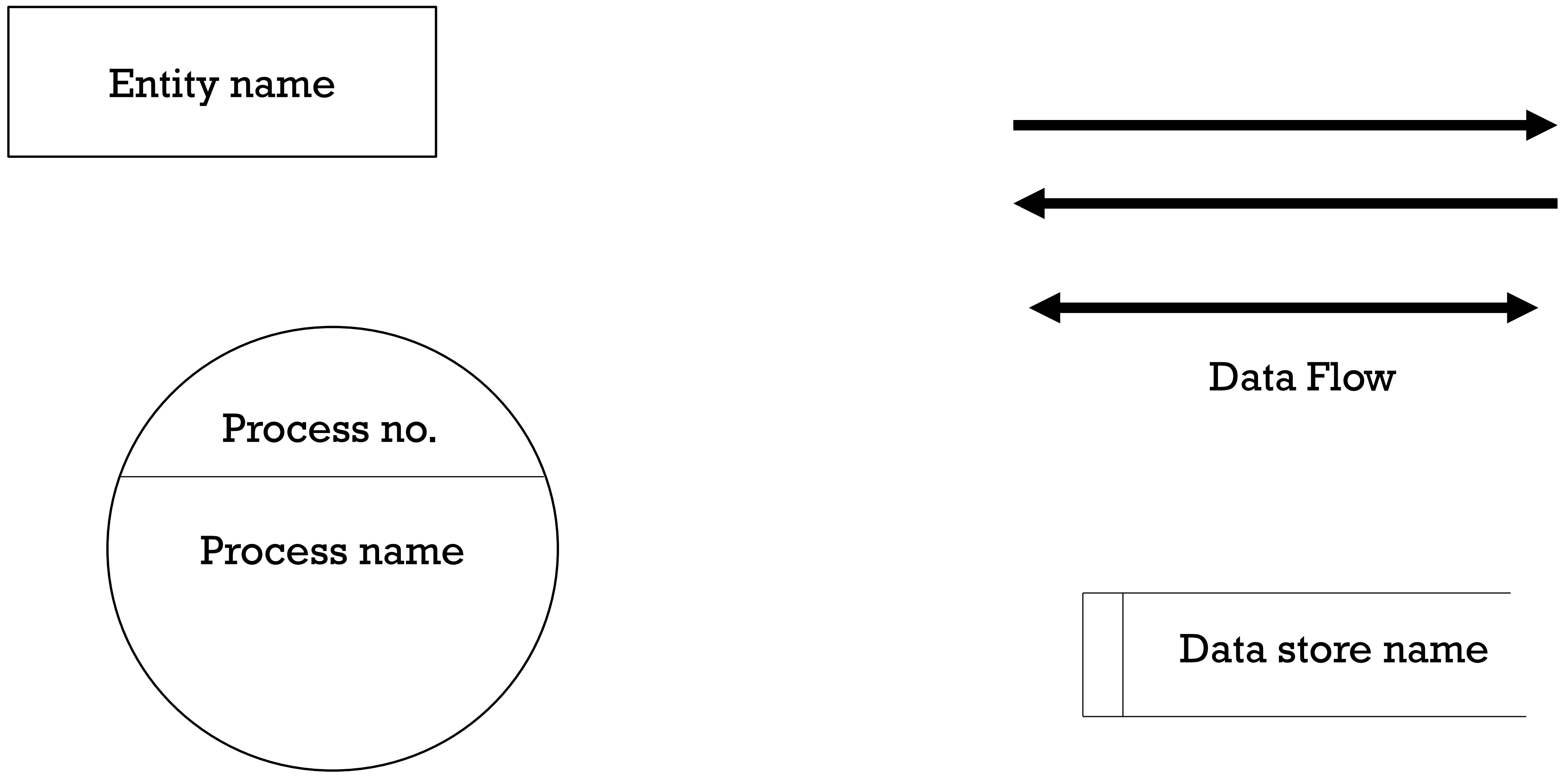
ATTRIBUTES OF SA/SD

1. Graphical description of information flow in the application/software.
2. Division of processes providing a clear picture of system flow.
3. Logical flow.
4. Works from high-level overview to lower level details.
5. Captures the detailed structure of the system as the user views it.
6. Purpose: arrive at a form suitable for implementation in a programming language.
7. Basic principle : top-down approach and divide and conquer.
8. One of the examples of SA/SD is **Data Flow Diagrams (DFD)**.

DATA FLOW DIAGRAM (DFD)

1. Also called as the "Bubble Chart".
2. It is a hierarchical graphical model.
3. Describes how the data flow through the system.
4. Graphical nature of DFD makes it a good communication tool between user and analyst or system designer and analyst.
5. Provides an overview of data processed by the system, transformations performed, data type storage, results produced and the flow of information.
6. Shows data interchange among processes.
7. Useful to consider every DFD unit as a processing station, consuming some input data and producing some output data.

STANDARD DFD SYMBOLS



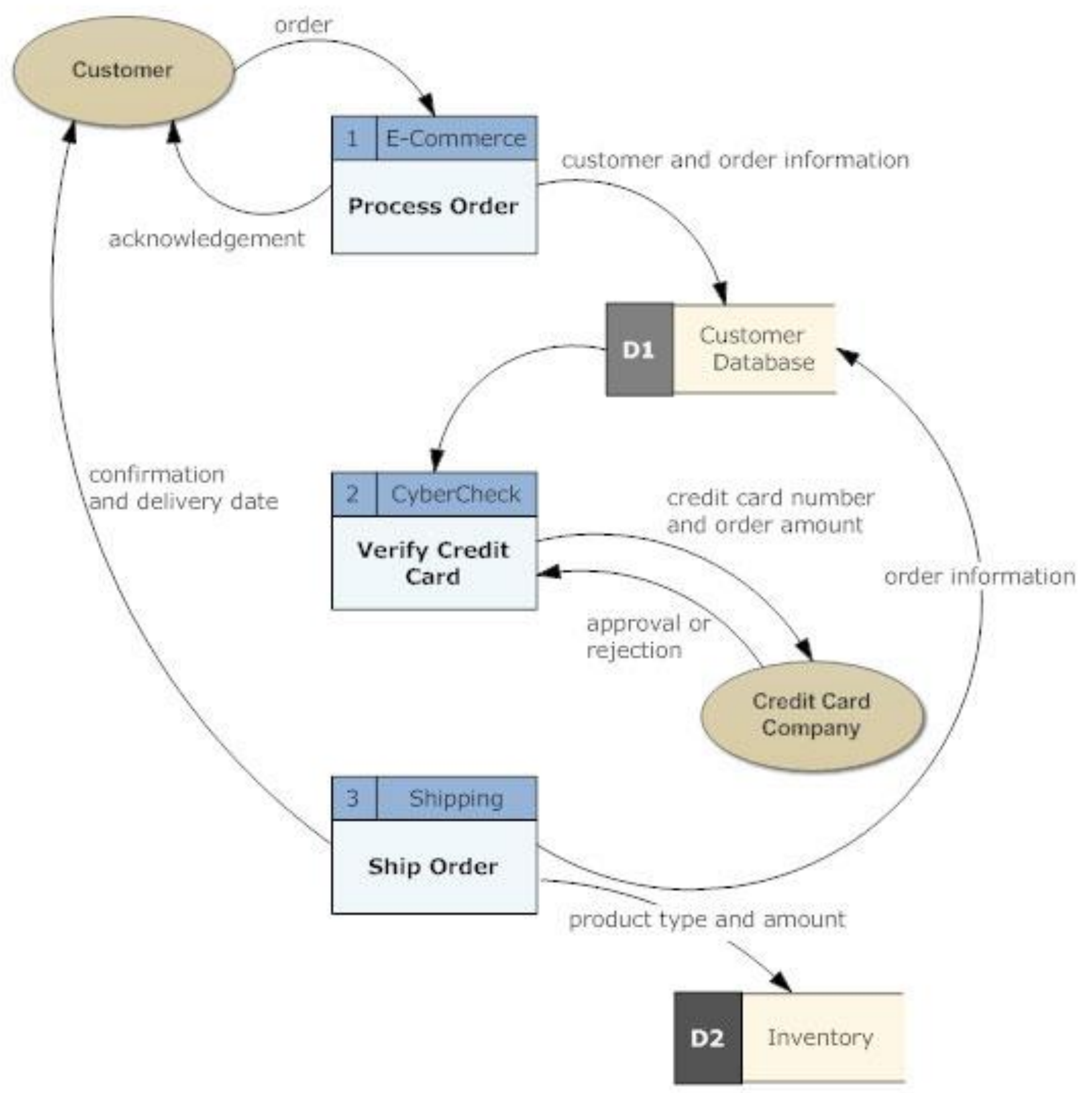
SYMBOLS USED IN DFDS

- DFDs consist of 4 symbols which are joined with lines.
- There may be a single DFD for the system or it may be exploded into several levels named Level 1, Level 2, and Level 3,etc.
- The top level diagram is often called a *context diagram* consisting of a single process that shows the overall view of the system.
- **Entities**: Sometimes called as source or sink (destinations) of the diagram. An entity may be people, place, customer, program or organization. Other entities can interact with the system.
- **Data Flow**: This shows the movement of data from one point to another in the diagram. The data flow is given a simple name such as *library book deposit* or *booking a ticket.* The data flows from entity to process or process to process.
- **Processes**: show the transformation of input data flows to output data flows. The name of the process describes what happens to data as it flows into the system. Name could consist of single strong verb or singular object like *calculate_net_pay, compute_balance,* etc.
- **Data Store:** is basically a database or repository. May be a database file or transaction file. Processes may store or retrieve data from a data store. Arrow pointing to data store indicates writing to the data store and from a data store indicates reading the data. A double headed arrow indicates both the operations of reading and writing in the data store.

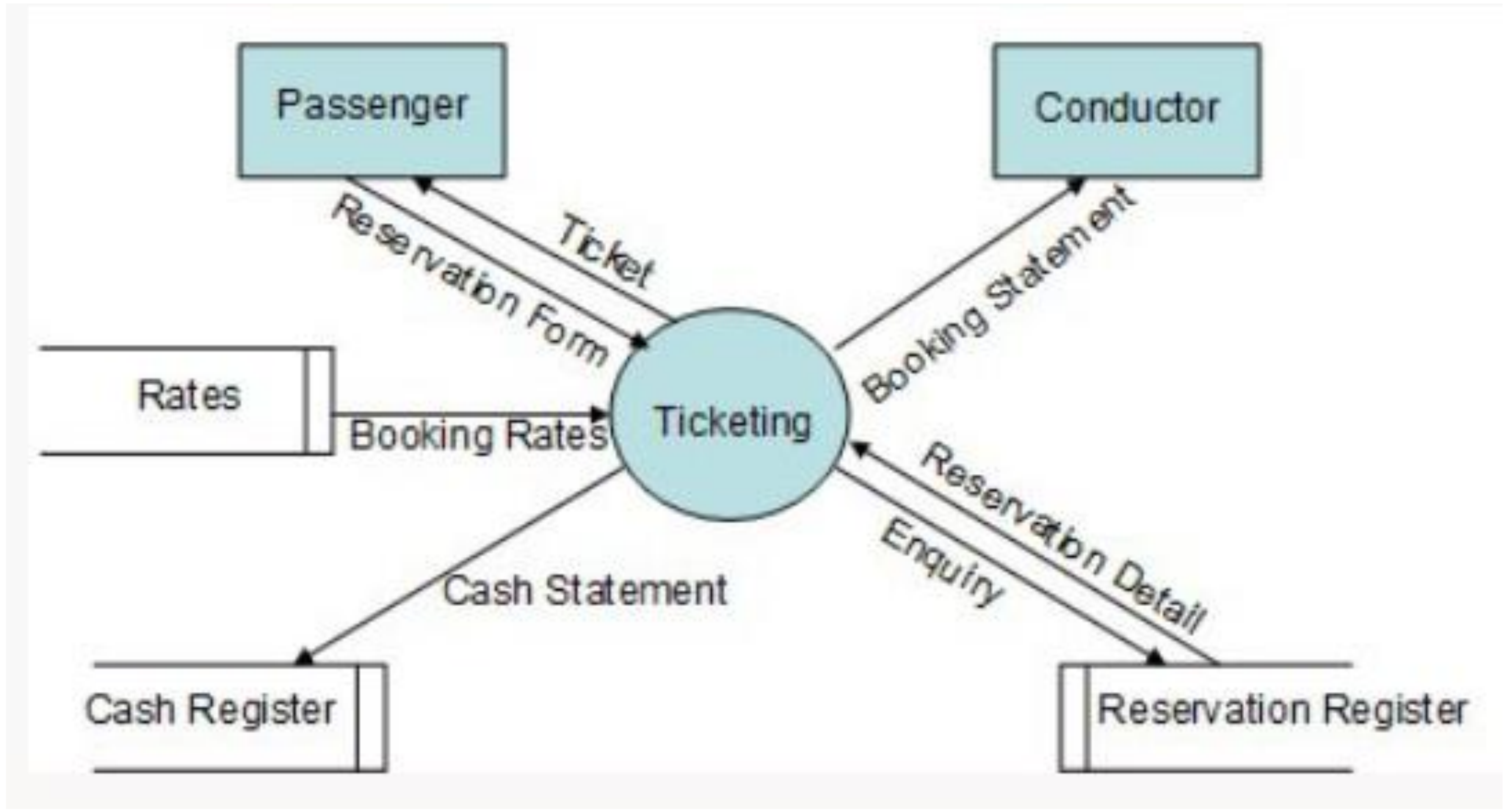
CONSTRUCTION OF DFD

- Following steps may be followed to construct a DFD:
  - Processes should be named and numbered for easy identification.
  - The direction of flow is from top to bottom and left to right.
  - Data flow from the upper left corner source to the lower right corner destination.
  - The names of data stores, sources, and destinations are written with the first letter capital.
- To start with, a context diagram is made.
- A context diagram should have single process.
- It is to understand the current system with boundaries.

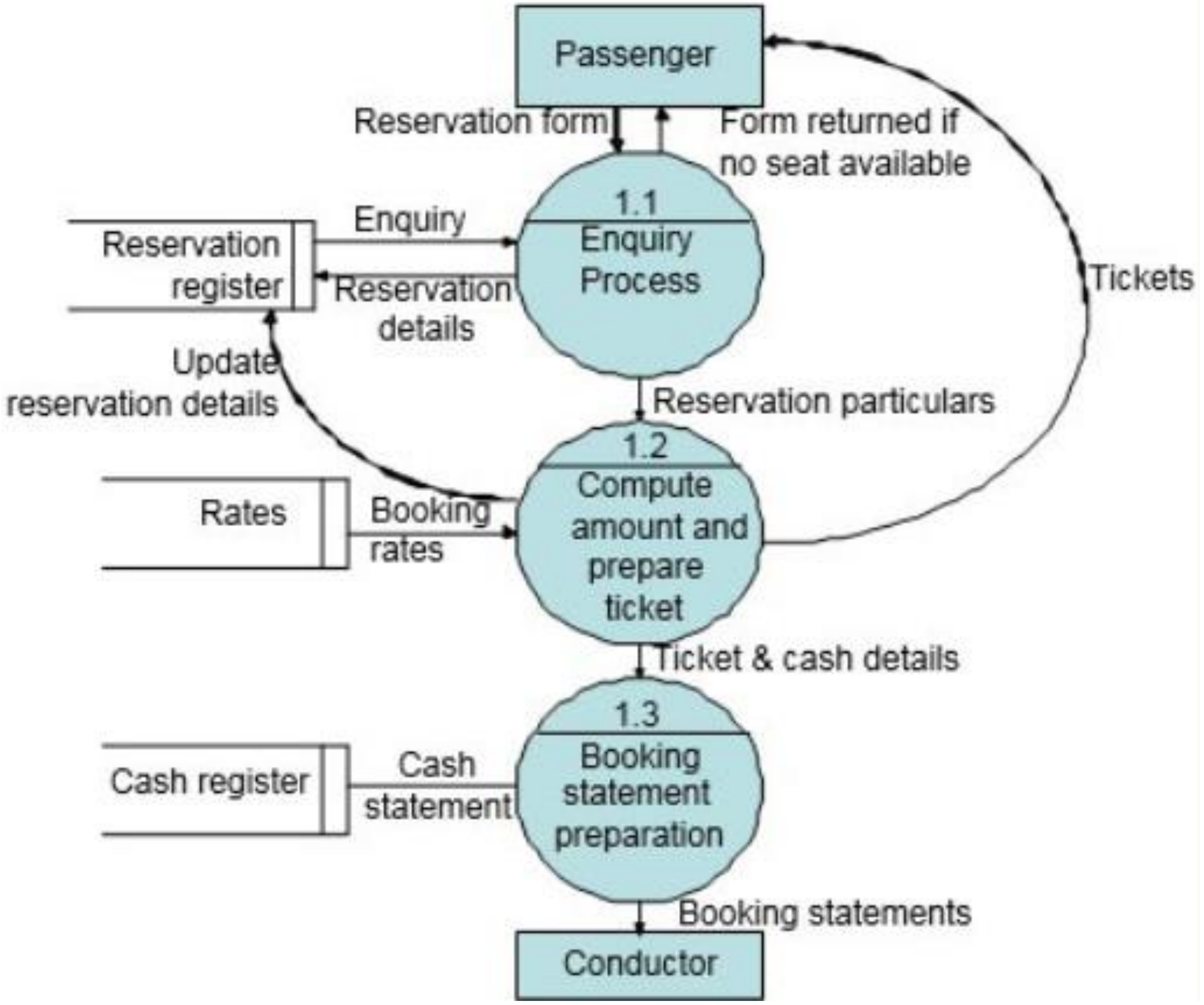
Data Flow Diagram - Online Order System



EXAMPLE: RAILWAY TICKET RESERVATION

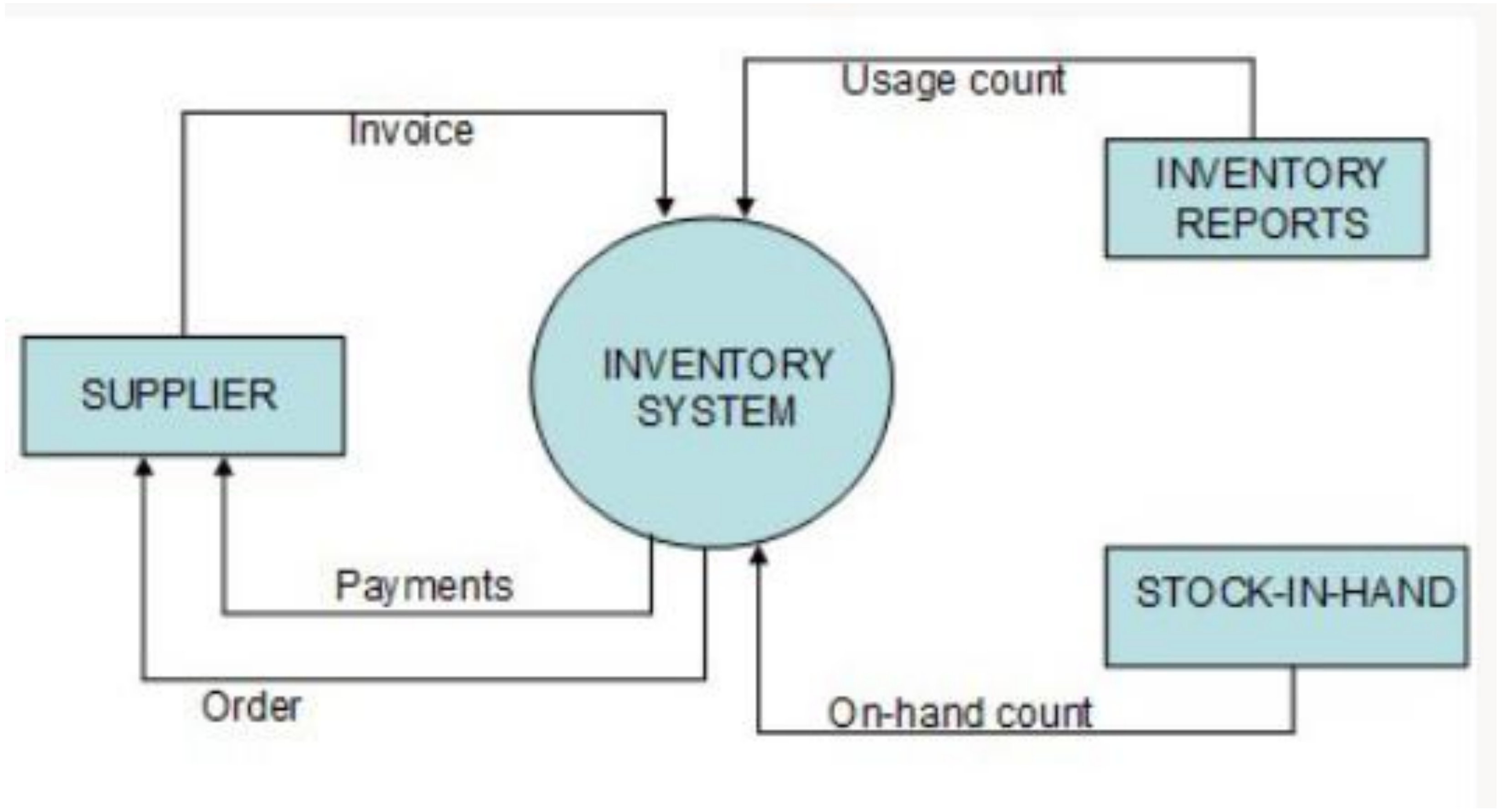


Context Diagram/Level 0 DFD

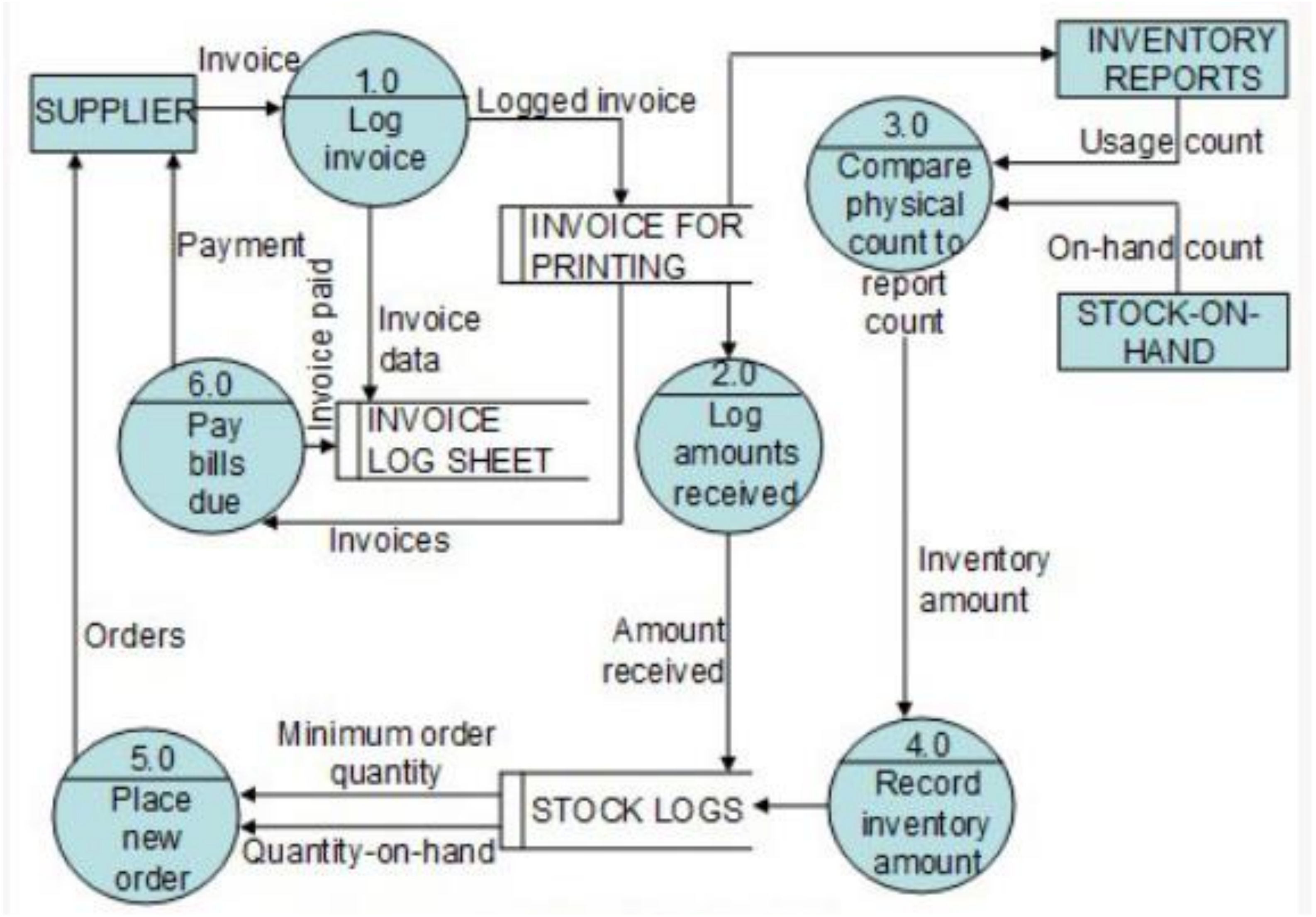


First level DFD, an overview of the system.

EXAMPLE: INVENTORY CONTROL

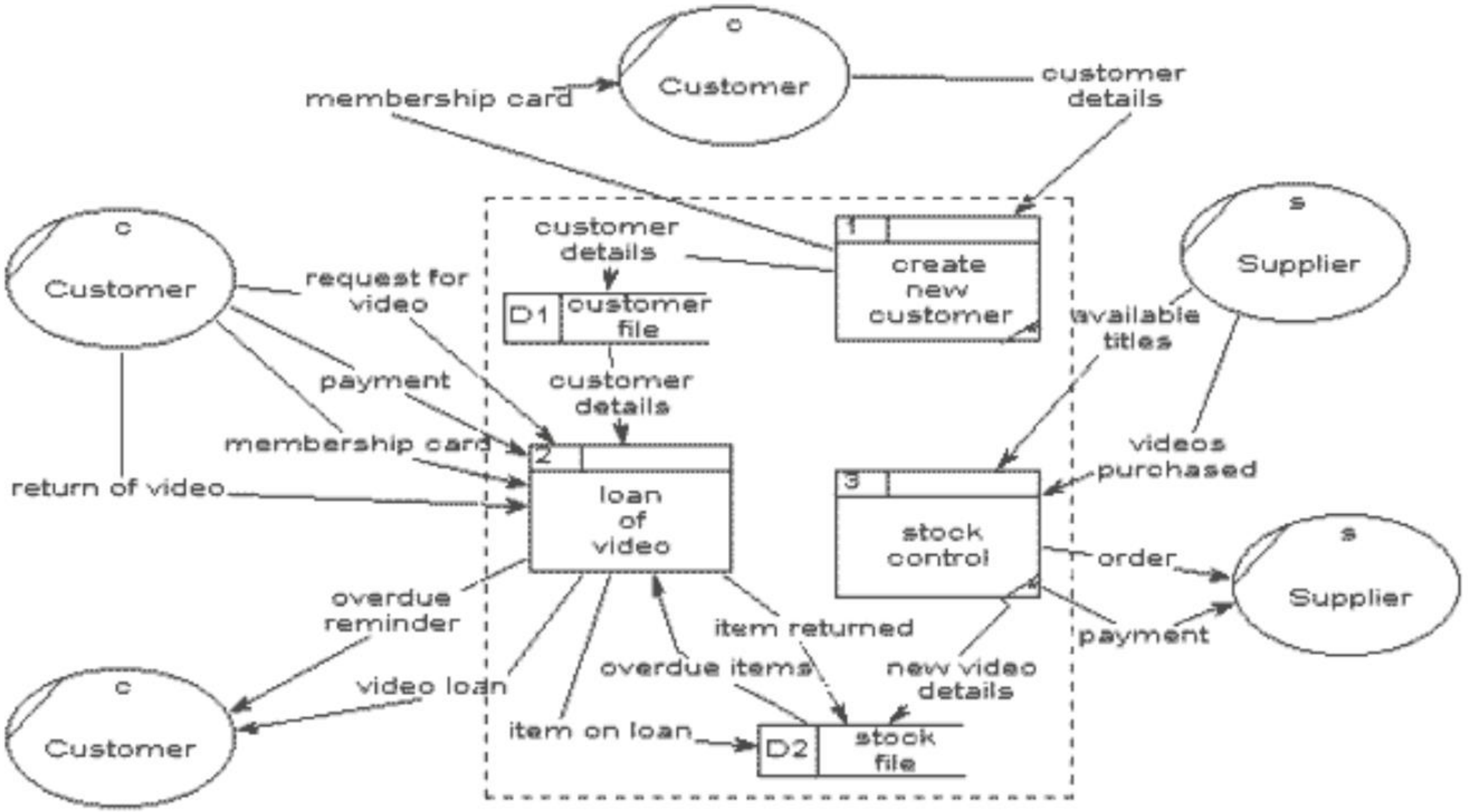


Context Diagram

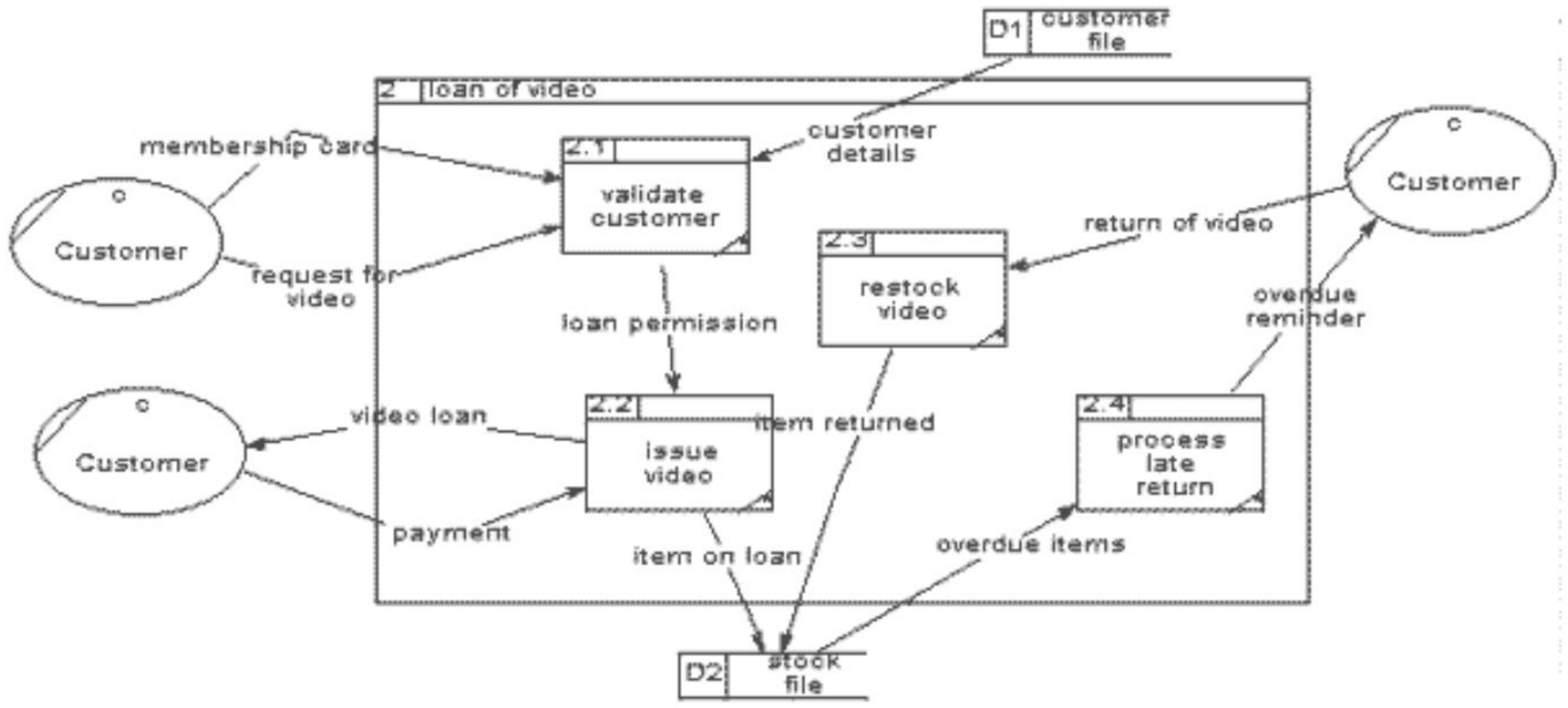

SUPPLIER
Invoice
1.0
Log invoice
Logged invoice
INVOICE FOR PRINTING
INVENTORY REPORTS
3.0
Compare physical count to report count
Usage count
On-hand count
STOCK-ON-HAND
Payment
Invoice paid
Invoice data
6.0
Pay bills due
INVOICE LOG SHEET
2.0
Log amounts received
Invoices
Inventory amount
Orders
Amount received
Minimum order quantity
Quantity-on-hand
5.0
Place new order
STOCK LOGS
4.0
Record inventory amount

Level 1 DFD

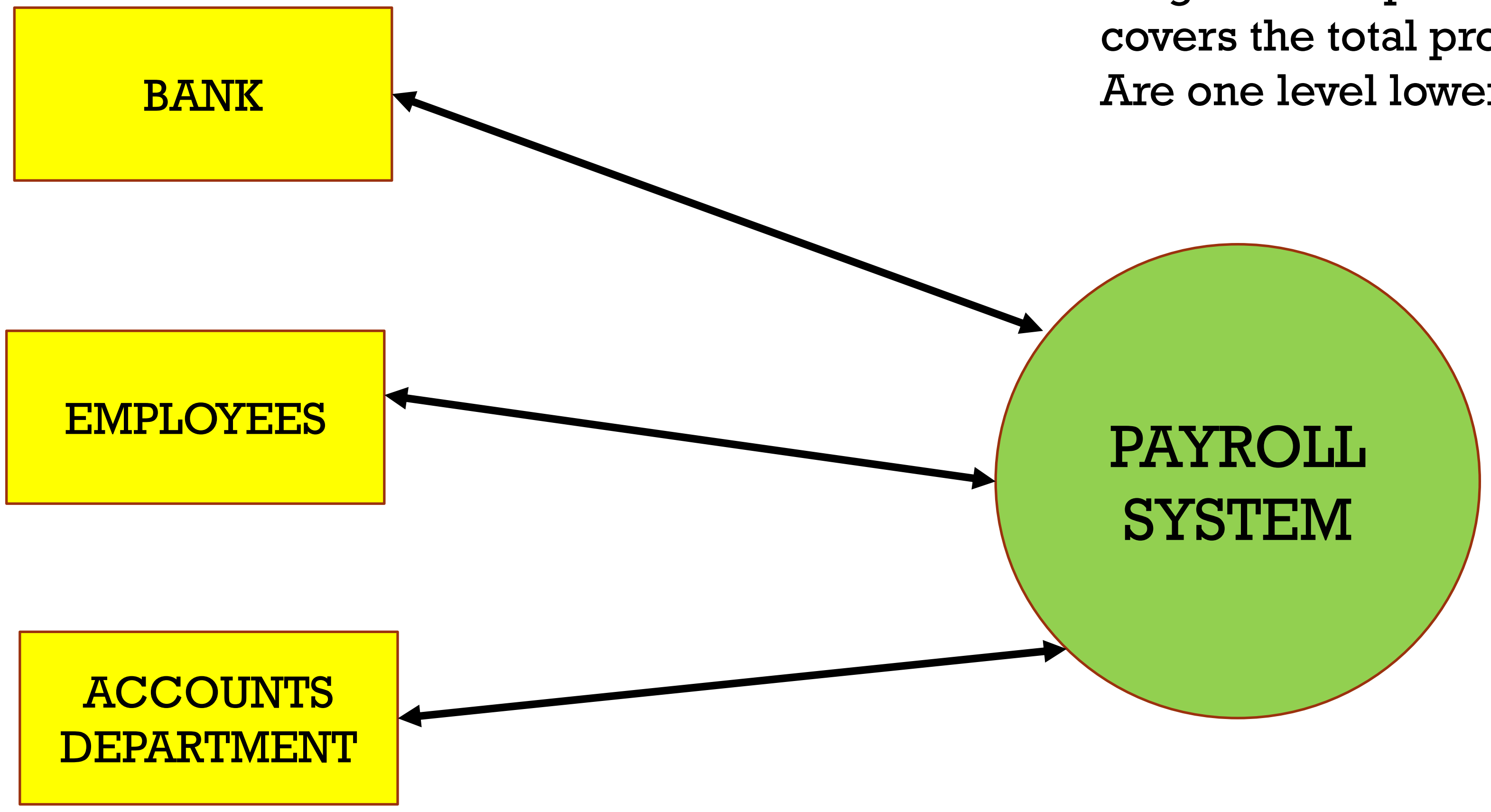
A level 1 DFD for Video-Rental



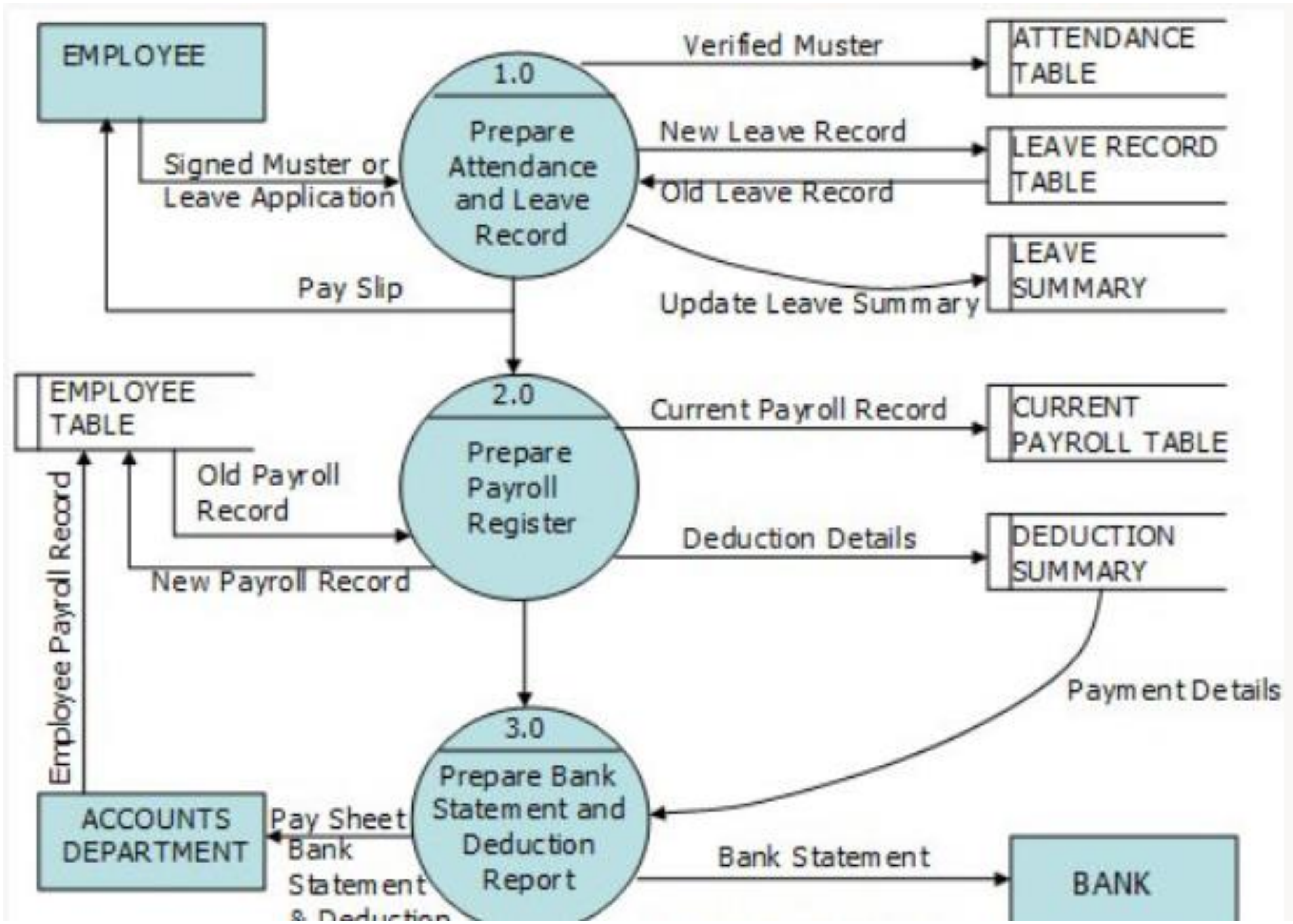
level 2 DFD for process "2: Loan of video" of the level 1 DFD



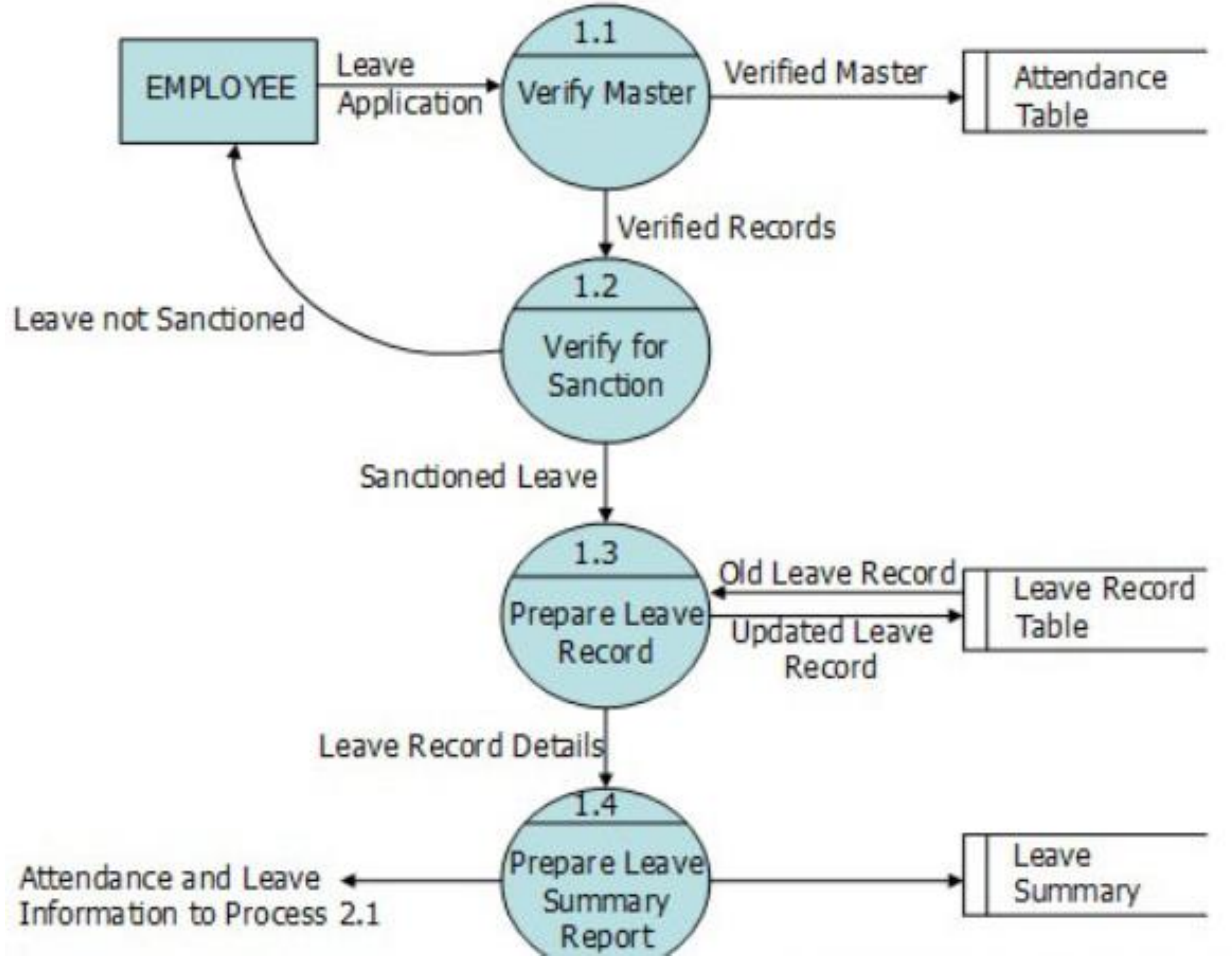
The main process given in the context diagram is exploded into sub-processes which covers the total process. The sub-processes Are one level lower than the parent process.



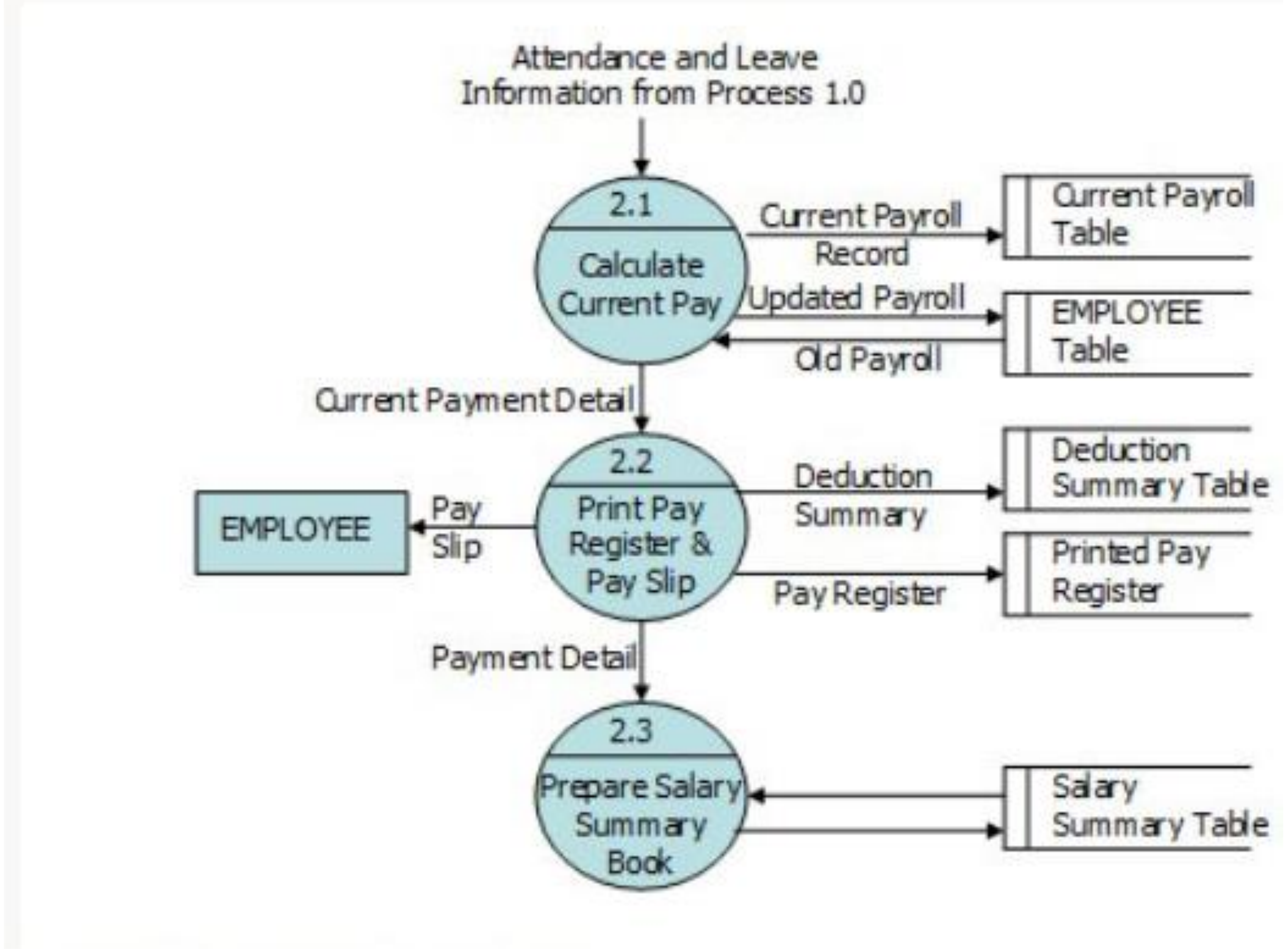
Context diagram of a payroll system.


EMPLOYEE
1.0
Prepare Attendance and Leave Record
Verified Muster
ATTENDANCE TABLE
New Leave Record
LEAVE RECORD TABLE
Old Leave Record
Signed Muster or Leave Application
LEAVE SUMMARY
Update Leave Summary
Pay Slip
EMPLOYEE TABLE
2.0
Prepare Payroll Register
Current Payroll Record
CURRENT PAYROLL TABLE
Old Payroll Record
New Payroll Record
Deduction Details
DEDUCTION SUMMARY
Employee Payroll Record
Payment Details
3.0
Prepare Bank Statement and Deduction Report
ACCOUNTS DEPARTMENT
Pay Sheet
Bank Statement & Deduction
Bank Statement
BANK

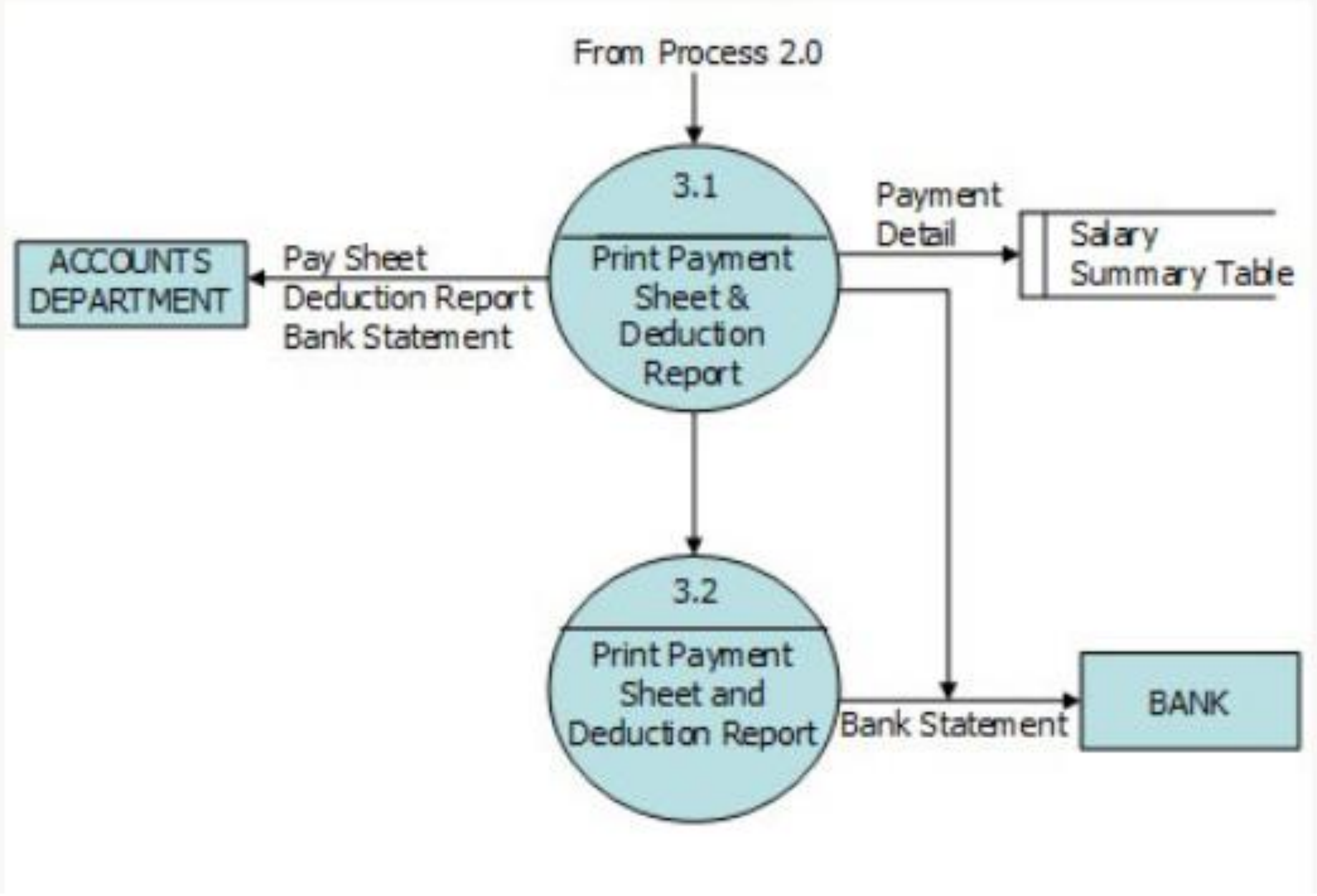
Level 1 DFD



Level 2 DFD for Process 1



Level 2 DFD for Process 2



Level 2 DFD for Process 3

DFD CREATION GUIDELINES

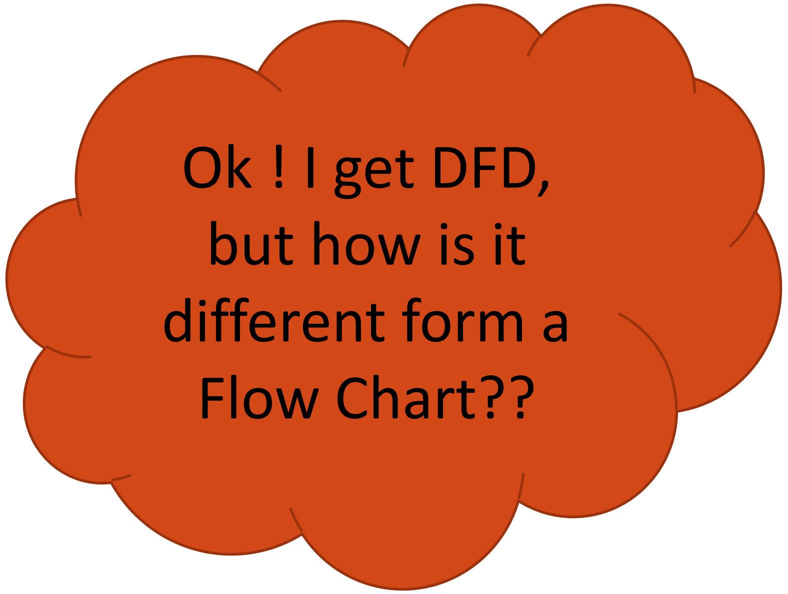
1. All functions of the system must be captured in the DFD model.
2. No function specified in the Software Requirements Specification (SRS) document should be overlooked.
3. Only those functions specified in the SRS document should be represented.
4. Please do not assume extra functionality of the system not specified by the SRS document.

SHORTCOMINGS OF THE DFD MODEL

1. DFDs leave ample scope to be imprecise.
   1. A label may not capture all the functionality of the bubble.
2. Control information may not always be clear.
3. The ultimate level to which decomposition is carried out is subjective.

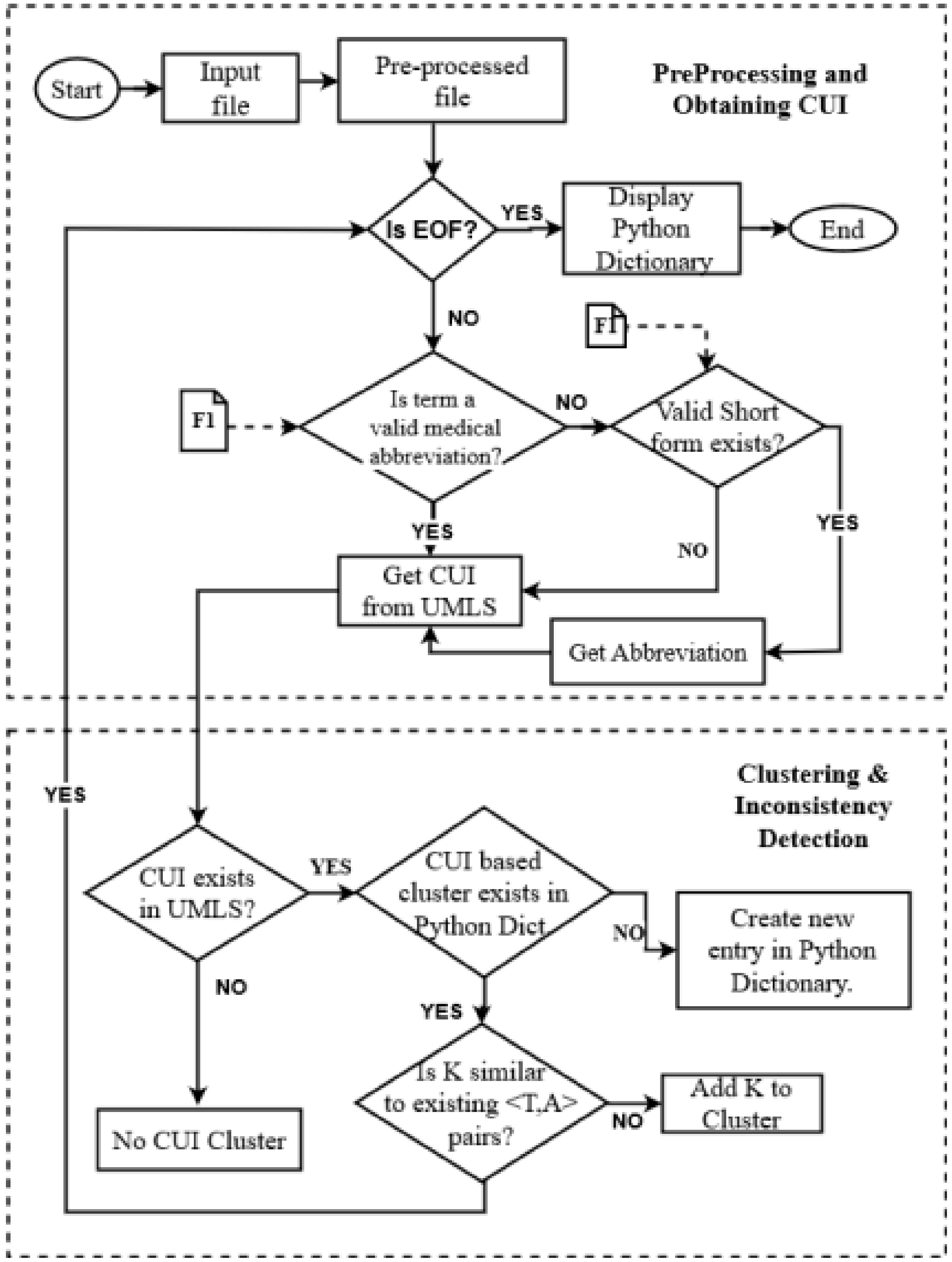
ONLINE SPACE FOR CREATING DFD

1. Commerical
   1. SmartDraw (30 day free trial)
   2. Visio
   3. Edraw
   4. Creately
2. Dia (GNU open source)


Ok ! I get DFD, but how is it different form a Flow Chart??

FLOW CHART

1. Convenient technique to represent the flow of information in a system.
2. Difficult to identify modules of a software from its flow chart representation.
3. Data interchange among the modules is not represented in a flow chart.



Start
Input file
Pre-processed file
PreProcessing and Obtaining CUI
Is EOF?
YES
Display Python Dictionary
End
NO
F1
F1
Is term a valid medical abbreviation?
NO
Valid Short form exists?
YES
NO
YES
Get CUI from UMLS
Get Abbreviation
YES
Clustering & Inconsistency Detection
CUI exists in UMLS?
YES
CUI based cluster exists in Python Dict
NO
Create new entry in Python Dictionary.
NO
YES
Is K similar to existing <T,A> pairs?
NO
Add K to Cluster
No CUI Cluster

Source: Devasish Mahato, Disha Dudhal, Dhanashree Revagade, and Yesoda Bhargava. 2019. A Method to Detect Inconsistent Annotations in a Medical Document using UMLS. In Proceedings of the 11th Forum for Information Retrieval Evaluation (FIRE '19). Association for Computing Machinery, New York, NY, USA, 47-51. DOI:https://doi.org/10.1145/3368567.3368577