Structured Analysis and Design

Acknowledgement: Fundamentals of Software Engineering
(**Prof. Rajib Mall**)

INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR
भारतीय प्रौद्योगिकी संस्थान भुवनेश्वर
2008

Introduction

- Two distinct style of design:
  - **Function-oriented or Procedural**
    - Top-down approach
    - Carried out using ***Structured analysis*** and ***structured design***
    - Coded using languages such as C
  - **Object-oriented**
    - Bottom-up approach
    - Carried out using **UML**
    - Coded using languages such as Java, C++, C#

Structured Analysis and Structured Design

- During Structured analysis:
  - High-level functions are successively **decomposed**:
    - Into more detailed functions.

- During Structured design
  - The detailed functions are mapped to a **module structure**.

Structured Analysis

- Successive decomposition of high-level functions:
  - Into more detailed functions.
  - Technically known as top-down decomposition.

Structured Analysis

- Textual problem description converted into a graphic model.
  - Done using data flow diagrams (DFDs).
  - **DFD** graphically represents the results of structured analysis.

Structured Analysis

- The results of structured analysis can be easily understood even by ordinary customers:
  - Does not require computer knowledge.
  - Directly represents customer's perception of the problem.
  - Uses customer's terminology for naming different functions and data.

- The results of structured analysis can be reviewed by customers:
  - To check whether it captures all their requirements.

Structured Design

- The functions represented in the DFD:
  - Mapped to a **module structure**.

- **Module structure**:
  - Also called **software architecture**

Structured Analysis vs. Structured Design

- **Purpose of structured analysis**:
  - Capture the detailed structure of the system as the user views it.

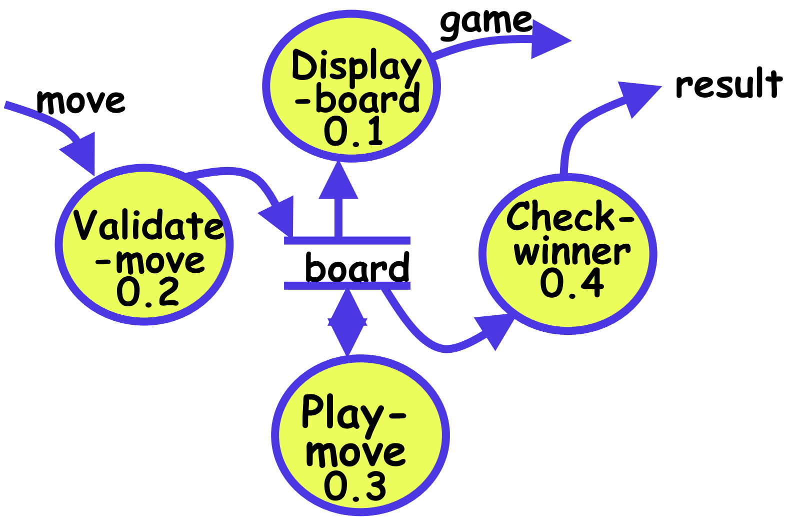
- **Purpose of structured design**:
  - Arrive at a form that is suitable for implementation in some programming language.

Structured Analysis

- Based on principles of:
  - Top-down decomposition approach.
  - Divide and conquer principle:
    - Each function is considered individually (i.e. isolated from other functions).
    - Decompose functions totally disregarding what happens in other functions.
  - Graphical representation of results using
    - Data flow diagrams (or bubble charts).

Data Flow Diagram

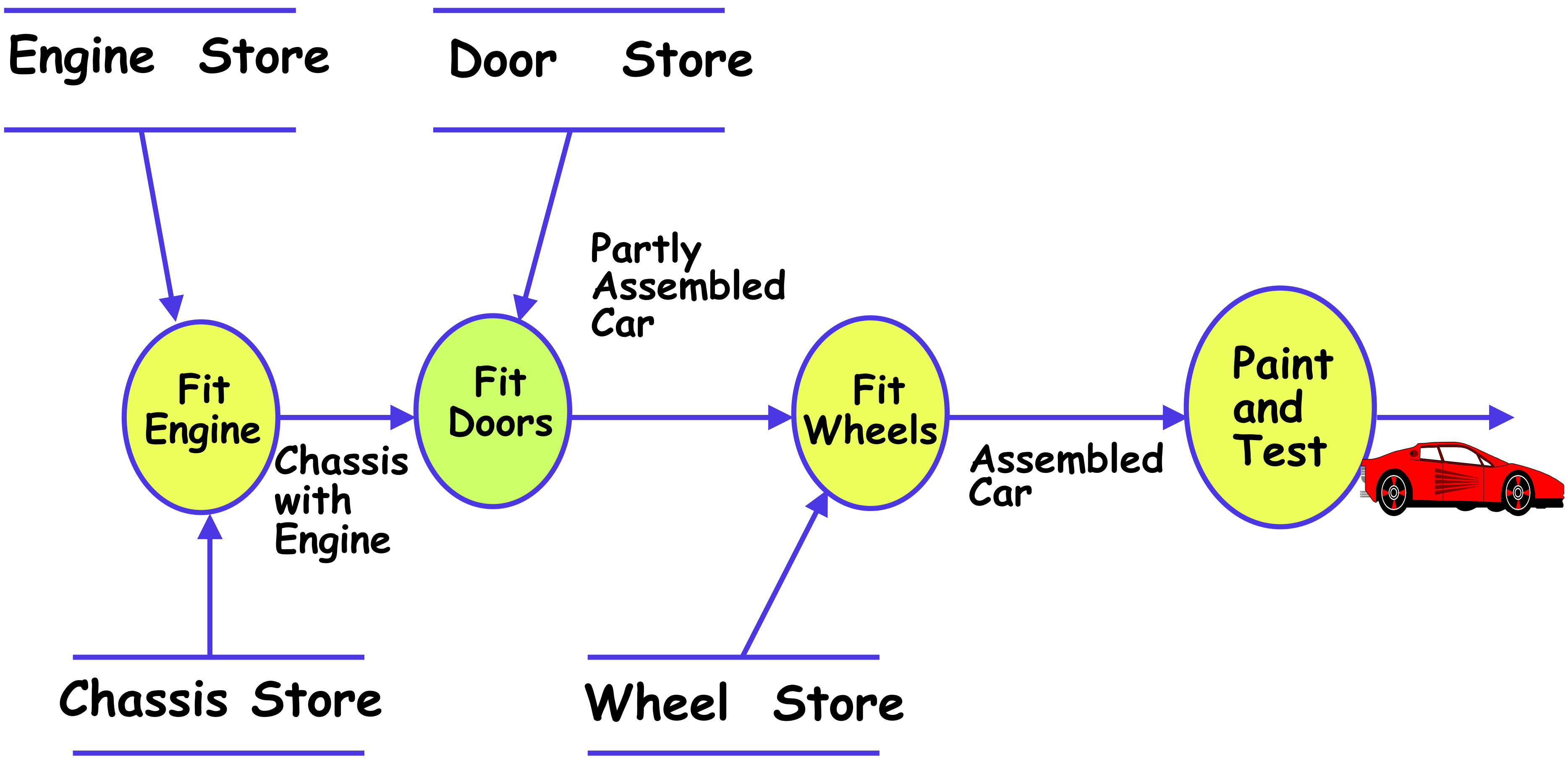
- DFD is a hierarchical graphical model:
  - Shows the different functions (or processes) of the system and
  - Data interchange among the processes.



DFD Concepts

- 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



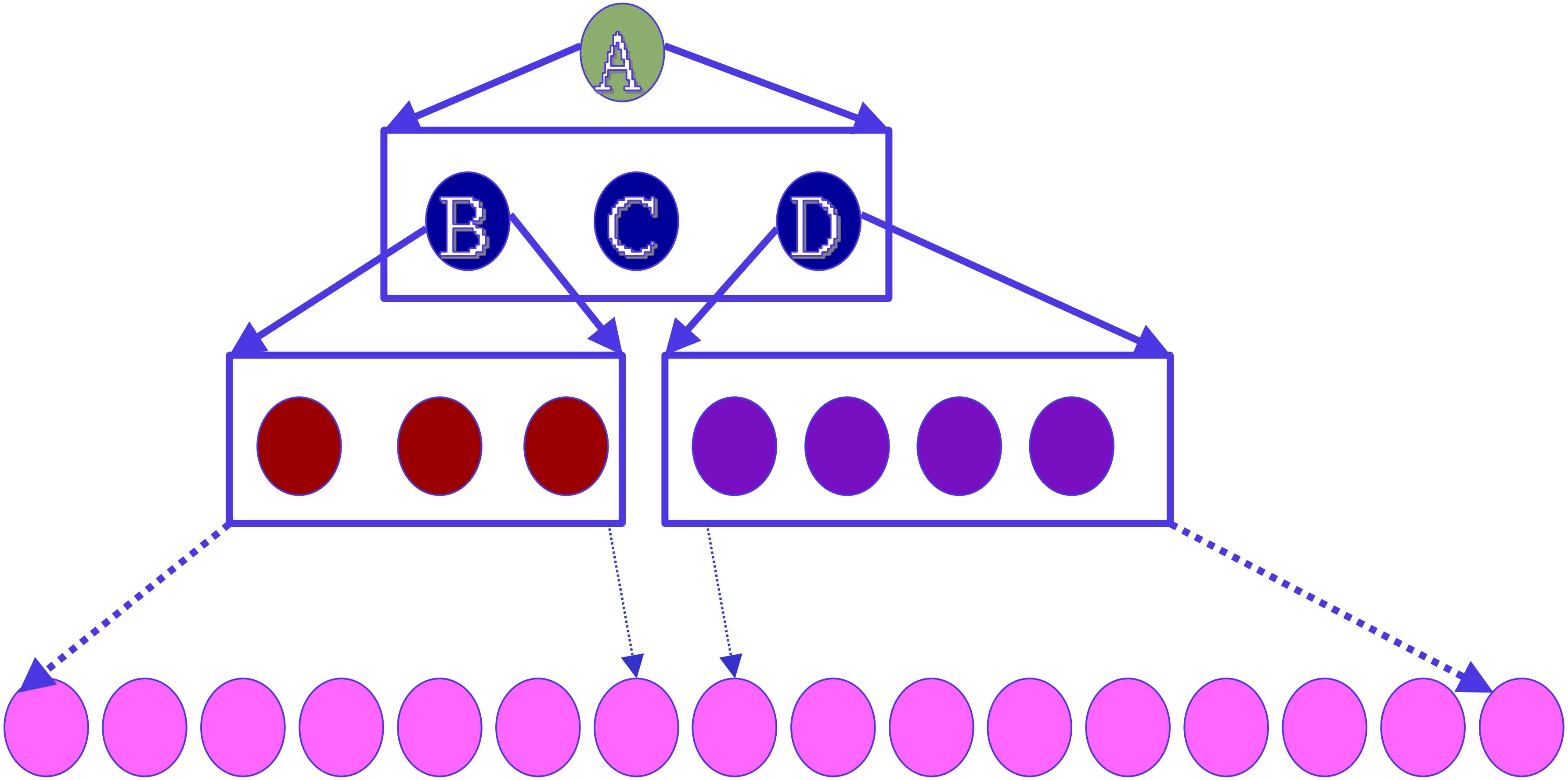
Data Flow Diagrams (DFDs)

- **A DFD model**:
  - Uses limited types of symbols.
  - Simple set of rules
  - Easy to understand --- a ***hierarchical*** model.

Hierarchical Model

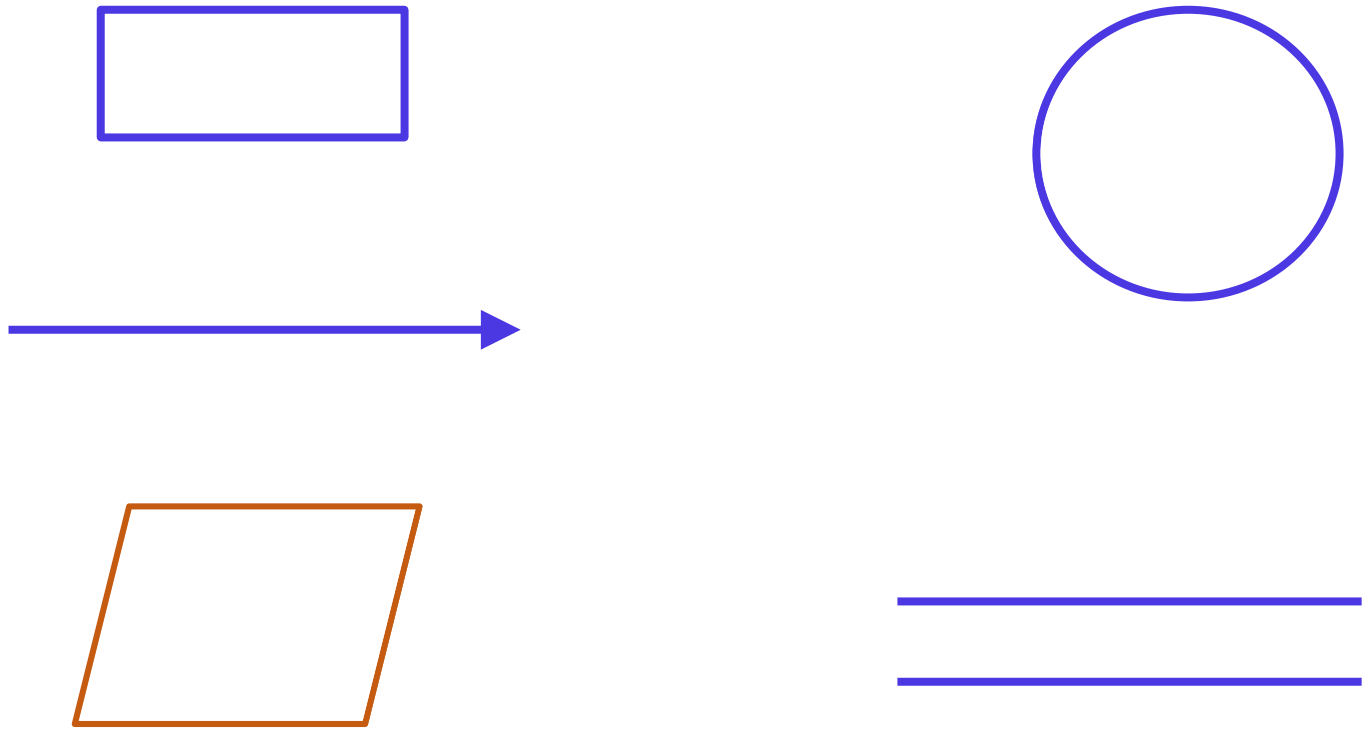
- In a hierarchical model:
  - We start with a very simple and abstract model of a system,
  - Details are slowly introduced through the hierarchies.

A Hierarchical Model



Data Flow Diagrams (DFDs)

- Primitive Symbols Used for Constructing DFDs:

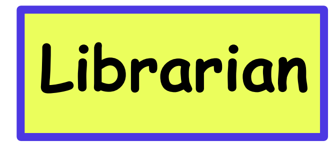


External Entity Symbol

- 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 terminator, source, or sink.

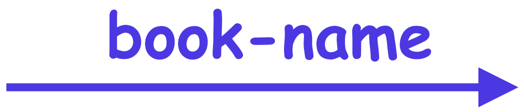
Function Symbol

- A function such as “**search-book**” is represented using a circle:

  - This symbol is called a process or bubble.
  - **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.

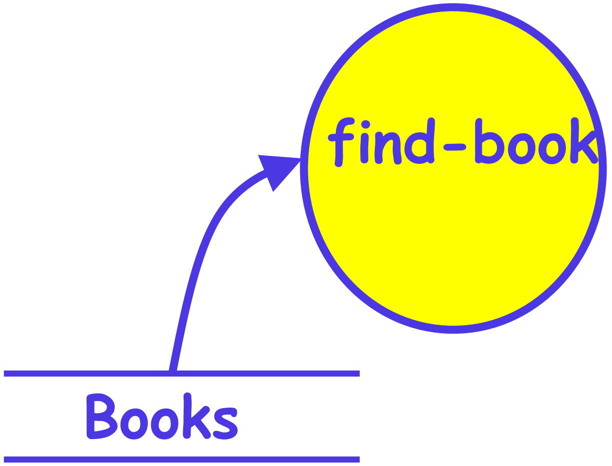


  - Represents data flow in the direction of the arrow.
  - Data flow symbols are annotated with names of data they carry.

Data Store Symbol

- Represents 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.

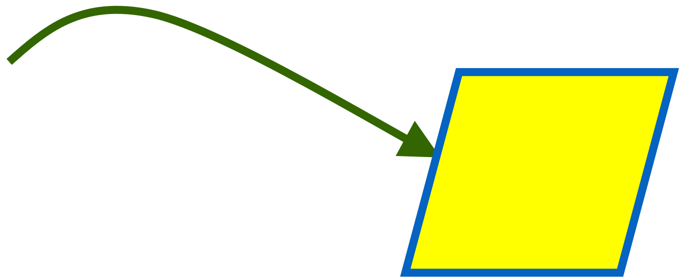
Data Store Symbol



- Direction of data flow arrow:
  - Shows whether data is being ***read*** from or ***written*** into it.
- An arrow into or out of a 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.

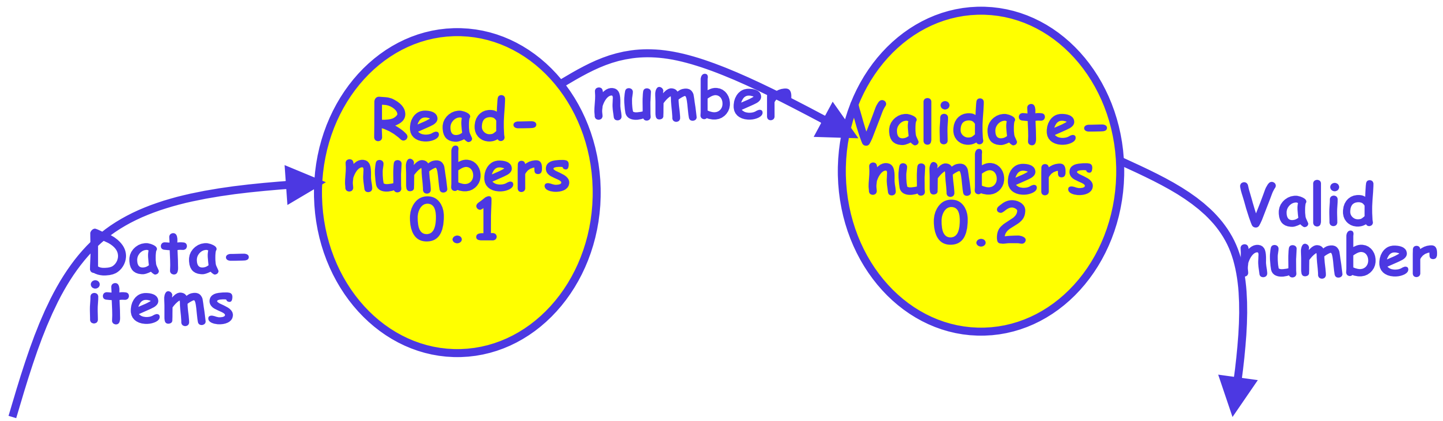
Output Symbol

- Output produced by the system



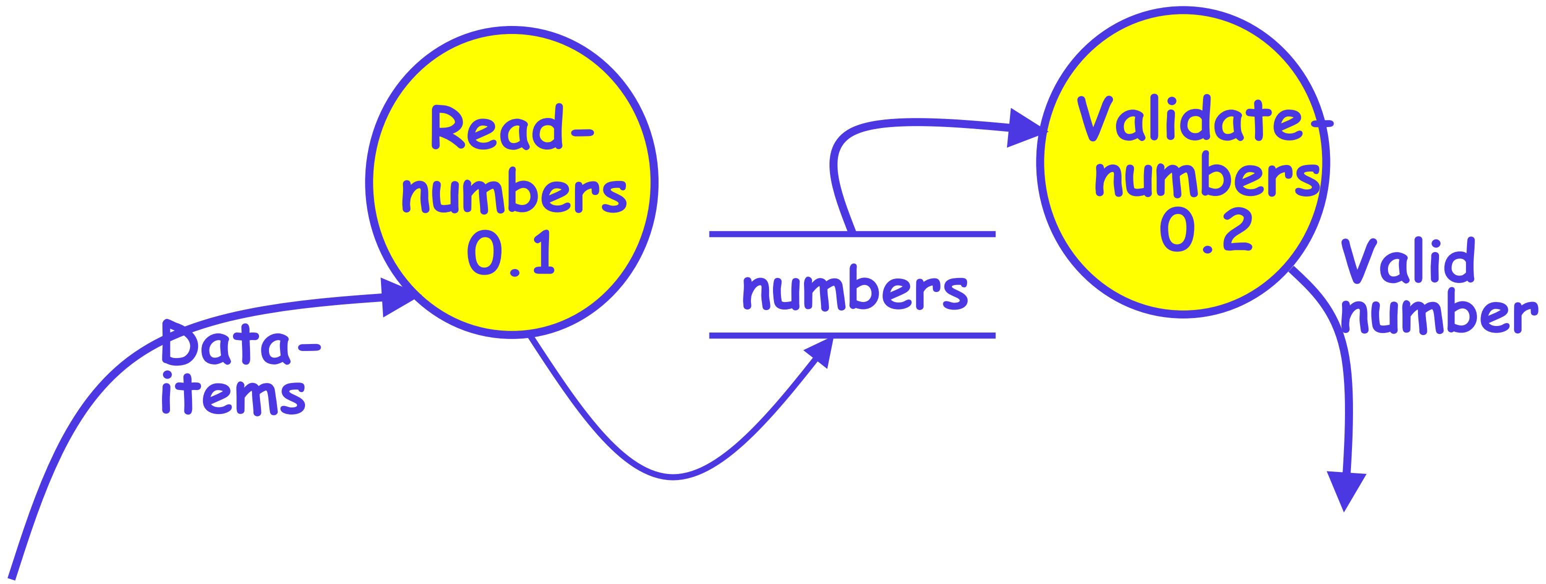
Synchronous Operation

- If two bubbles are directly connected by a data flow arrow:
  - They are synchronous



Asynchronous Operation

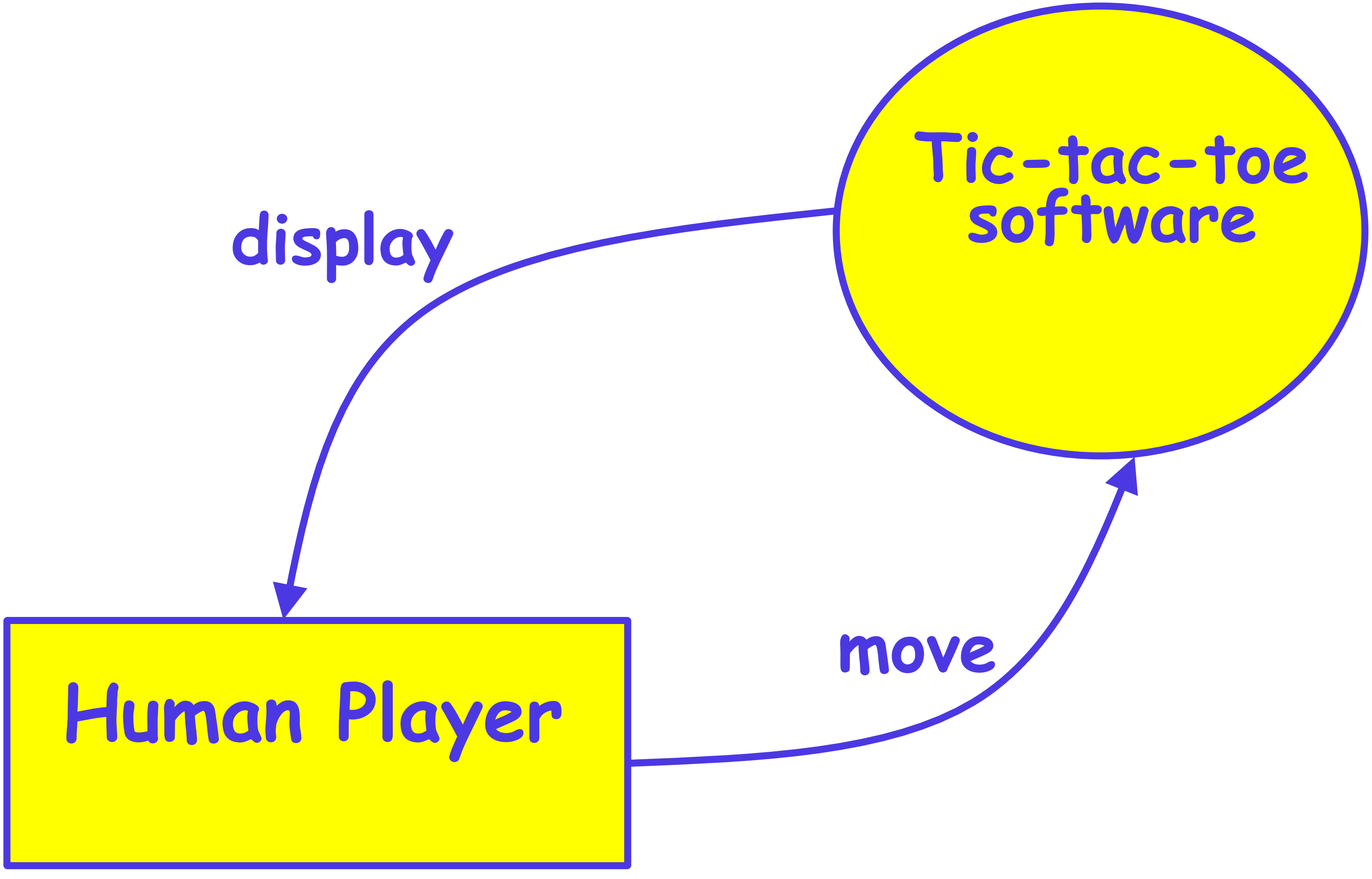
- If two bubbles are connected via a data store:
  - They are not synchronous.



How is Structured Analysis Performed?

- Initially represent the software at the most abstract level:
  - Called the **context diagram.**
  - The entire system is represented as a single bubble,
  - This bubble is labelled according to the main function of the system.

Tic-tac-toe: Context Diagram



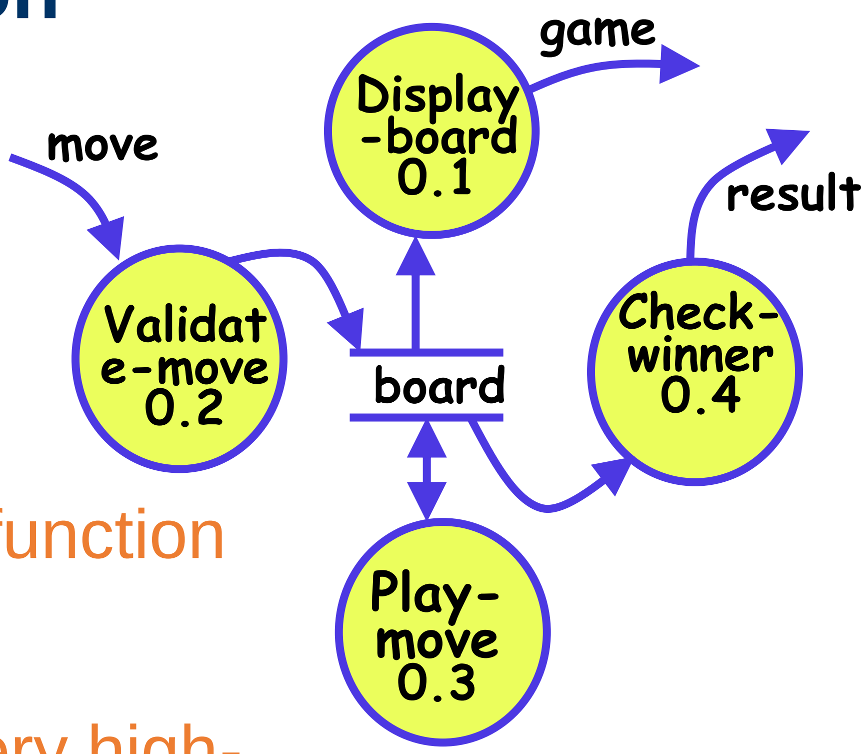
Context Diagram

- A context diagram shows:
  - External entities.
  - Data input to the system by them,
  - Output data generated by the system.
- The context diagram is also called the **level 0 DFD.**

Context Diagram

- Establishes the context of the system, i.e.
  - Represents
    - **Data sources**
    - **Data sinks.**

Level 1 DFD Construction



- Examine the SRS document:
  - Represent each high-level function as a bubble.
  - Represent data input to every high-level function.
  - Represent data output from every high-level function.

Higher Level DFDs

- Each high-level function is separately decomposed into subfunctions:
  - Identify the subfunctions of the function
  - Identify the data input to each subfunction
  - Identify the data output from each subfunction

- These are represented as DFDs.

Decomposition

- Decomposition of a bubble:
  - Also called factoring or exploding.

- Each bubble is decomposed into
  - Between 3 to 7 bubbles.

Decomposition

- Too few bubbles make decomposition superfluous:
  - If a bubble is decomposed to just one or two bubbles:
    - Then this decomposition is redundant.

Decomposition

- Too many bubbles:
  - **More than 7 bubbles at any level of a DFD.**
  - **Make the DFD model hard to understand.**

Decompose How Long?

- Decomposition of a bubble should be carried on until:
  - A level at which the function of the bubble can be described using a simple algorithm.

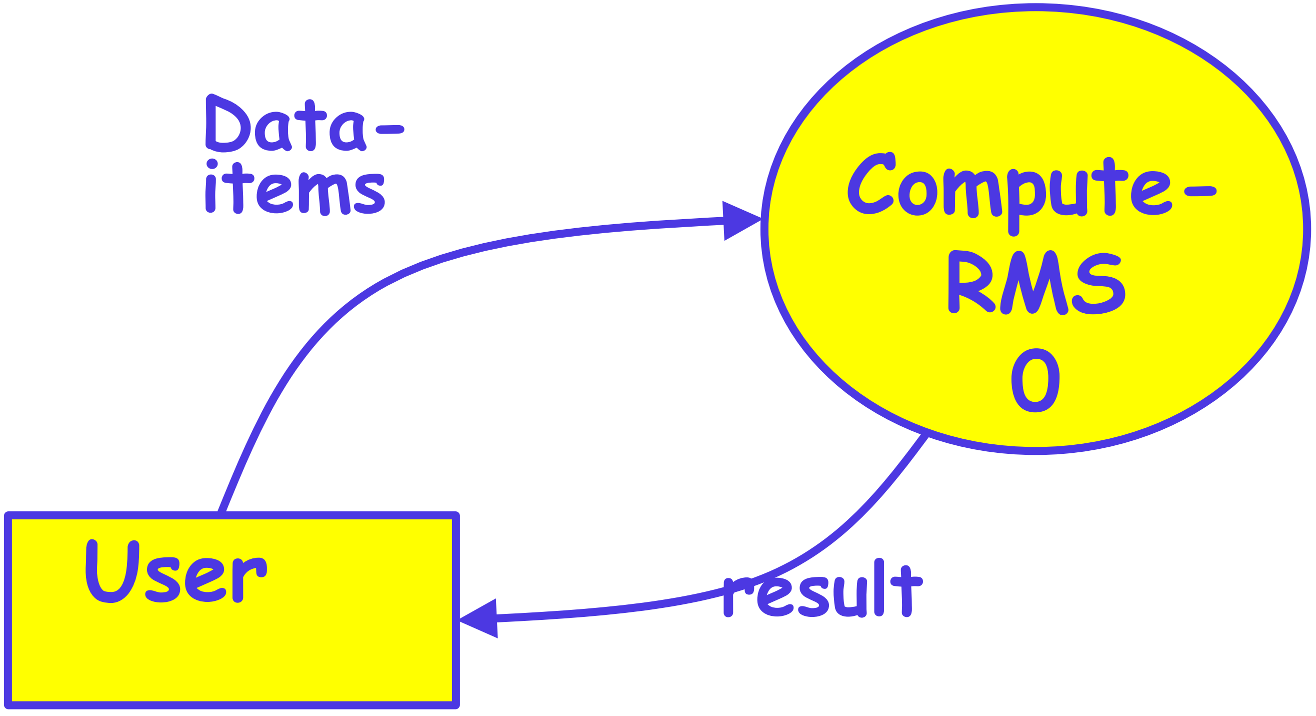
Example 1: RMS Calculating Software

- Consider a software called RMS calculating software:
  - Reads three integers in the range of -1000 and +1000
  - Finds out the root mean square (rms) of the three input numbers
  - Displays the result.

Example 1: RMS Calculating Software

- The **context diagram** is simple to develop:
  - The system **accepts 3 integers** from the user
  - **Returns the result** to the user.

Example 1: RMS Calculating Software



Context Diagram

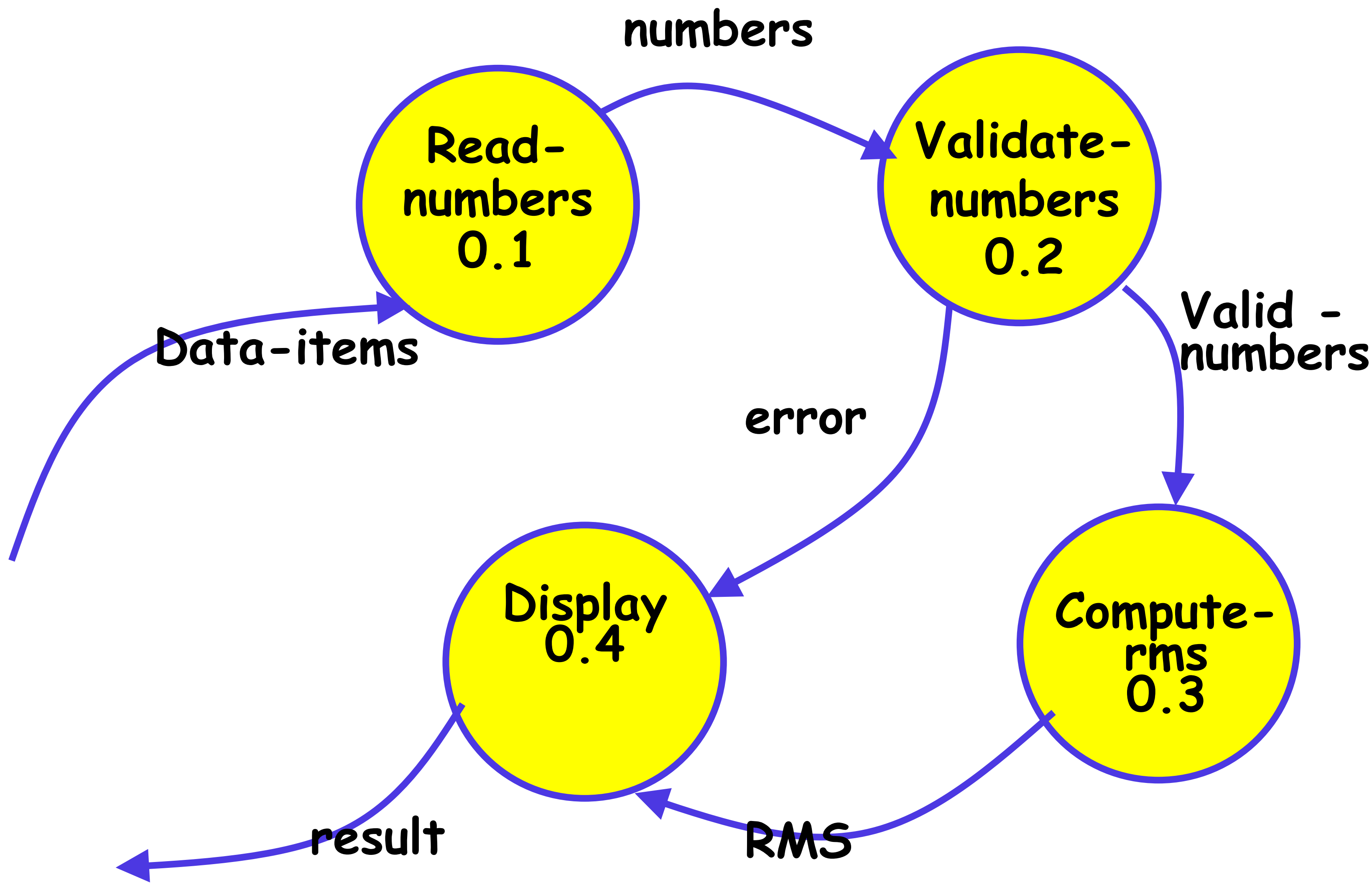
Example 1: RMS Calculating Software

- From a cursory analysis of the problem description:
  - We can see that the system needs to perform several things.

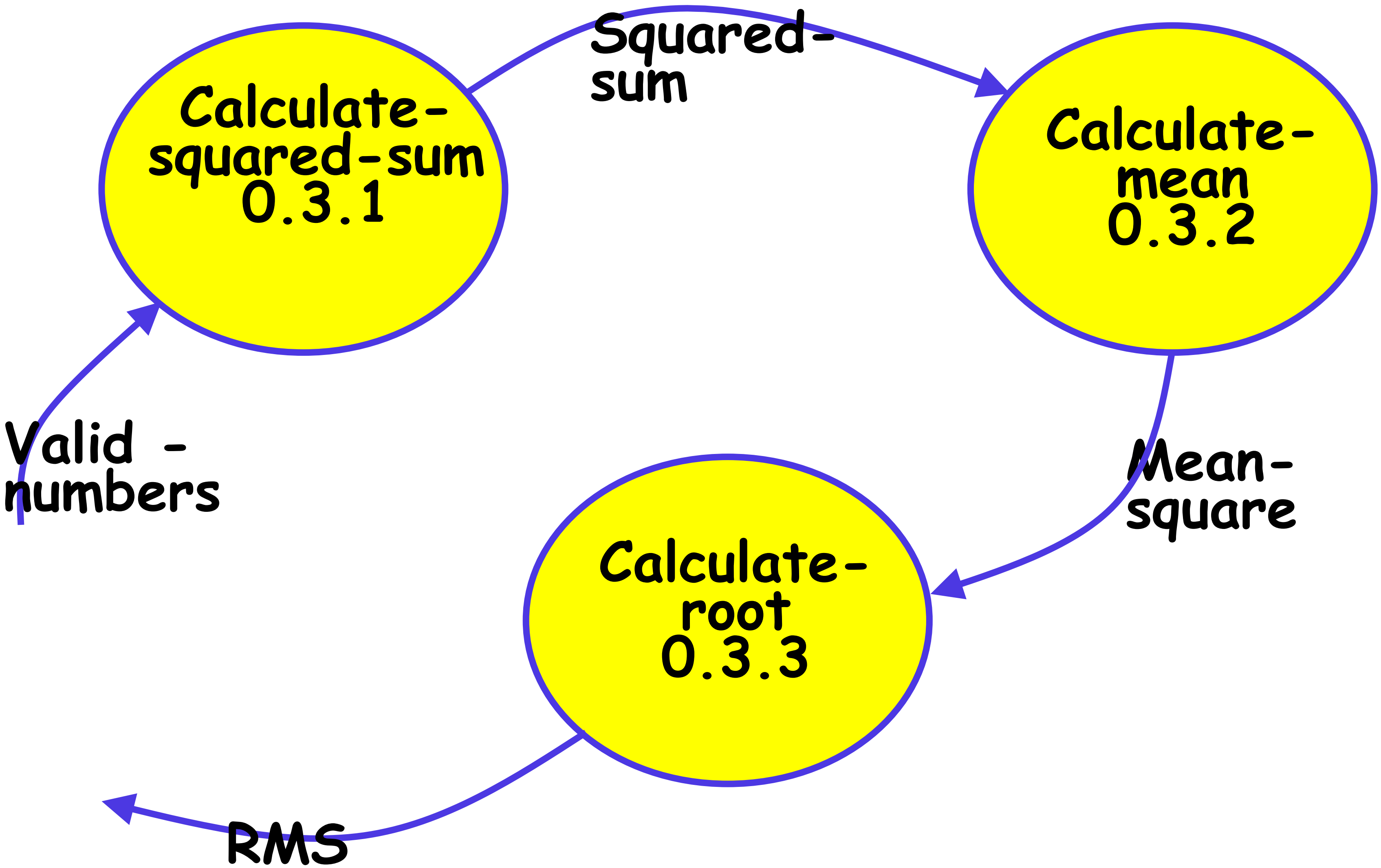
Example 1: RMS Calculating Software

- Accept input numbers from the user:
  - **Validate** the numbers,
  - **Calculate** the root mean square of the input numbers
  - **Display** the result.

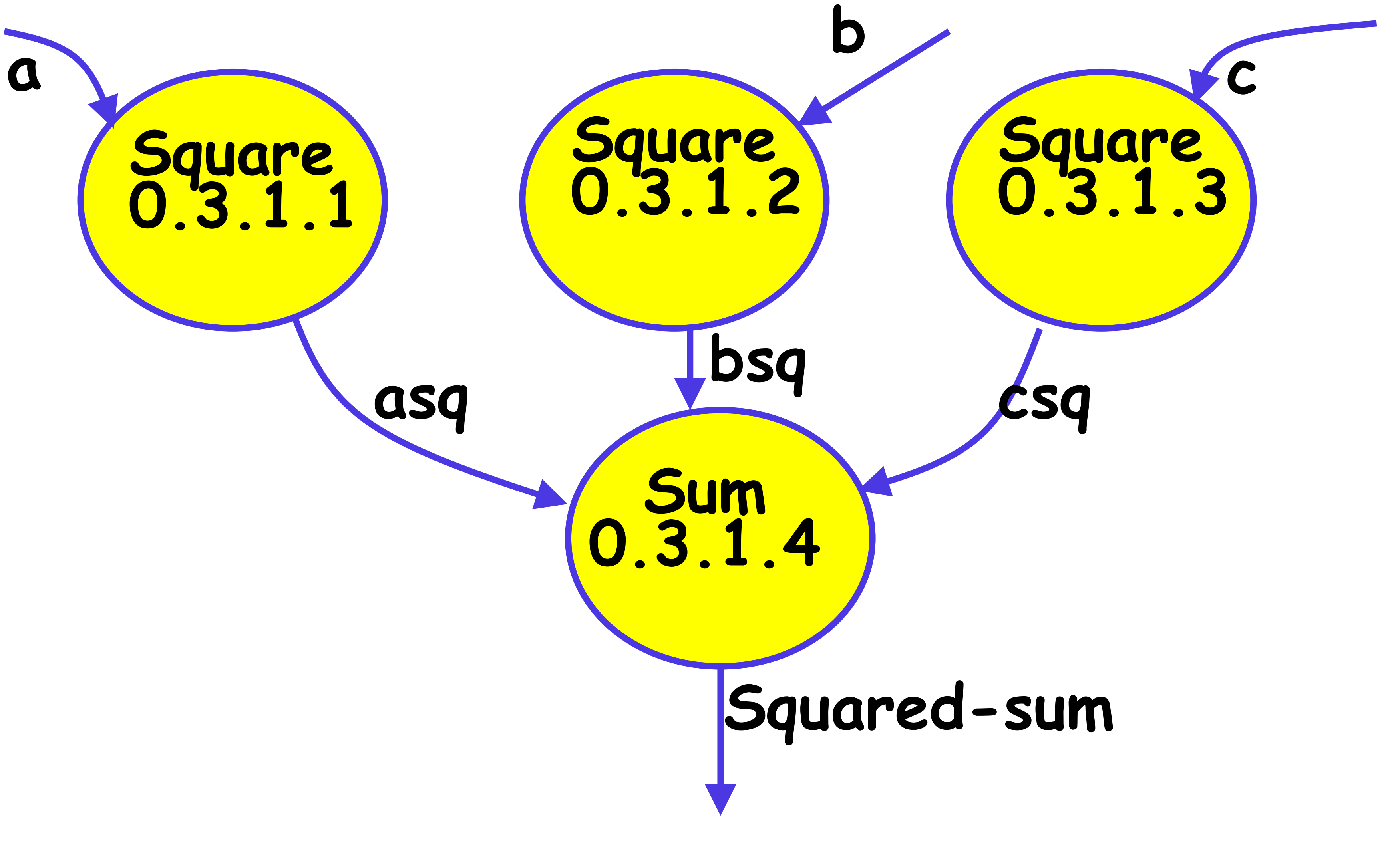
Example 1: RMS Calculating Software



Example 1: RMS Calculating Software



Example: RMS Calculating Software



Example: RMS Calculating Software

- Decomposition is never carried on up to basic instruction level:
  - A bubble is not decomposed any further:
    - If it can be represented by a simple set of instructions.

Data Dictionary

- A DFD is always accompanied by a data dictionary.

- A data dictionary lists all data items appearing in a DFD:
  - Definition of all composite data items in terms of their component data items.
  - All data names along with the purpose of the data items.

- For example, a data dictionary entry may be:
  - **grossPay = regularPay+overtimePay**

Importance of Data Dictionary

- Provides all engineers in a project with **standard terminology** for all data:
  - A consistent vocabulary for data is very important
  - Different engineers tend to use different terms to refer to the same data,
    - Causes unnecessary confusion.

Importance of Data Dictionary

- Data dictionary provides the definition of different data:
  - In terms of their component elements.
- For large systems,
  - The data dictionary grows rapidly in size and complexity.
  - Typical projects can have thousands of data dictionary entries.
  - It is extremely difficult to maintain such a dictionary manually.

Data Dictionary

- CASE (Computer Aided Software Engineering) tools come handy:
  - CASE tools capture the data items appearing in a DFD automatically to generate the data dictionary.

Data Dictionary

- CASE tools support queries:
  - About definition and usage of data items.

- For example, queries may be made to find:
  - Which data item affects which processes,
  - A process affects which data items,
  - The definition and usage of specific data items, etc.

Data Definition

- Composite data are defined in terms of primitive data items using following operators:
- **+:** denotes composition of data items, e.g
  - **a+b** represents data ***a*** and ***b***.
- **[,,,]:** represents selection,
  - i.e. any one of the data items listed inside the square bracket can occur.
  - For example, **[a,b]** represents either **a** occurs or **b** occurs.

Data Definition

- **( ):** contents inside the bracket represent optional data
    - which may or may not appear.
    - a+(b) represents either a or a+b occurs.
- { }: represents iterative data definition,
  - e.g. {name}5 represents five name data.

Data Definition

- {name}* represents
  - zero or more instances of name data.
- = represents equivalence,
  - e.g. **a=b+c** means that **a** represents **b** and **c**.
- * *: Anything appearing within * * is considered as comment.

Data Dictionary for RMS Software

- **numbers**=valid-numbers=a+b+c
- **a**:integer * input number *
- **b**:integer * input number *
- **c**:integer * input number *
- **asq**:integer
- **bsq**:integer
- **csq**:integer
- **squared-sum**: integer
- **Result**=[RMS,error]
- **RMS**: integer * root mean square value*
- **error**:string * error message*

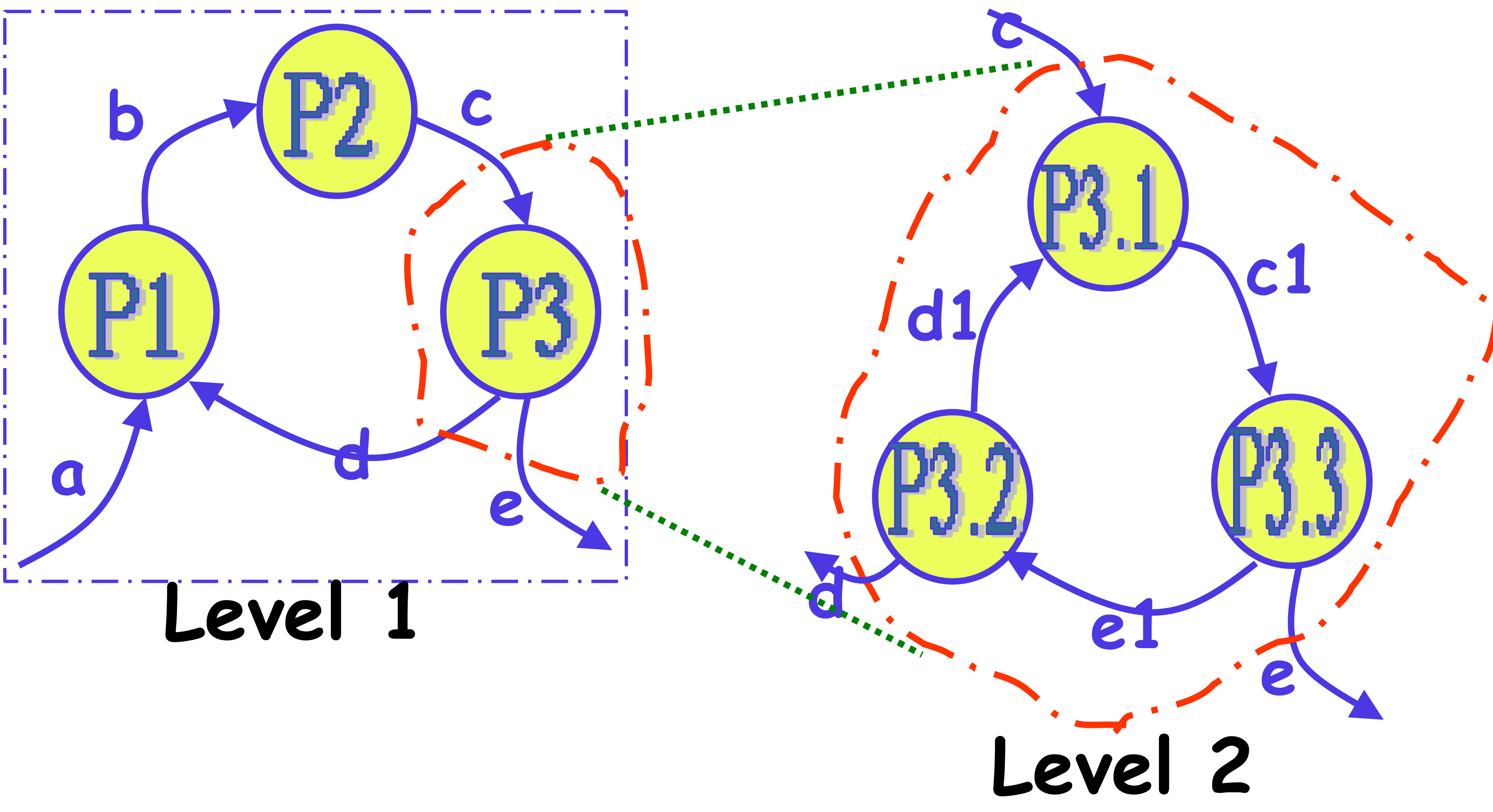
Balancing a DFD

- **Data flowing into or out of a bubble:**
  - Must **match the data flows at the next level** of DFD.

- In the level 1 of the DFD,
  - Data item **c** flows into the bubble **P3** and the data item **d** and **e** flow out.

- In the next level, bubble P3 is decomposed.
  - The decomposition is balanced as data item c flows into the level 2 diagram and d and e flow out.

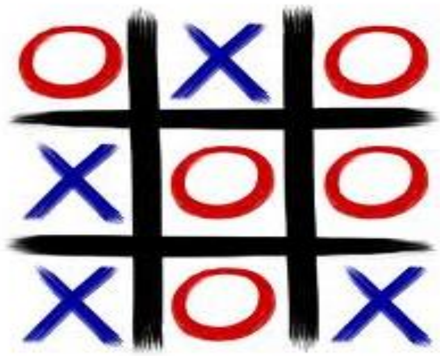
Balancing a DFD



Numbering of Bubbles

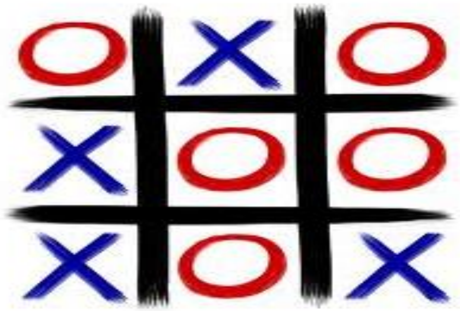
- Number the bubbles in a DFD:
  - Numbers help in uniquely identifying any bubble from its bubble number.
- The bubble at context level:
  - Assigned number 0.
- Bubbles at level 1:
  - Numbered 0.1, 0.2, 0.3, etc
- When a bubble numbered x is decomposed,
  - Its children bubble are numbered x.1, x.2, x.3, etc.

Example 2: Tic-Tac-Toe Computer Game



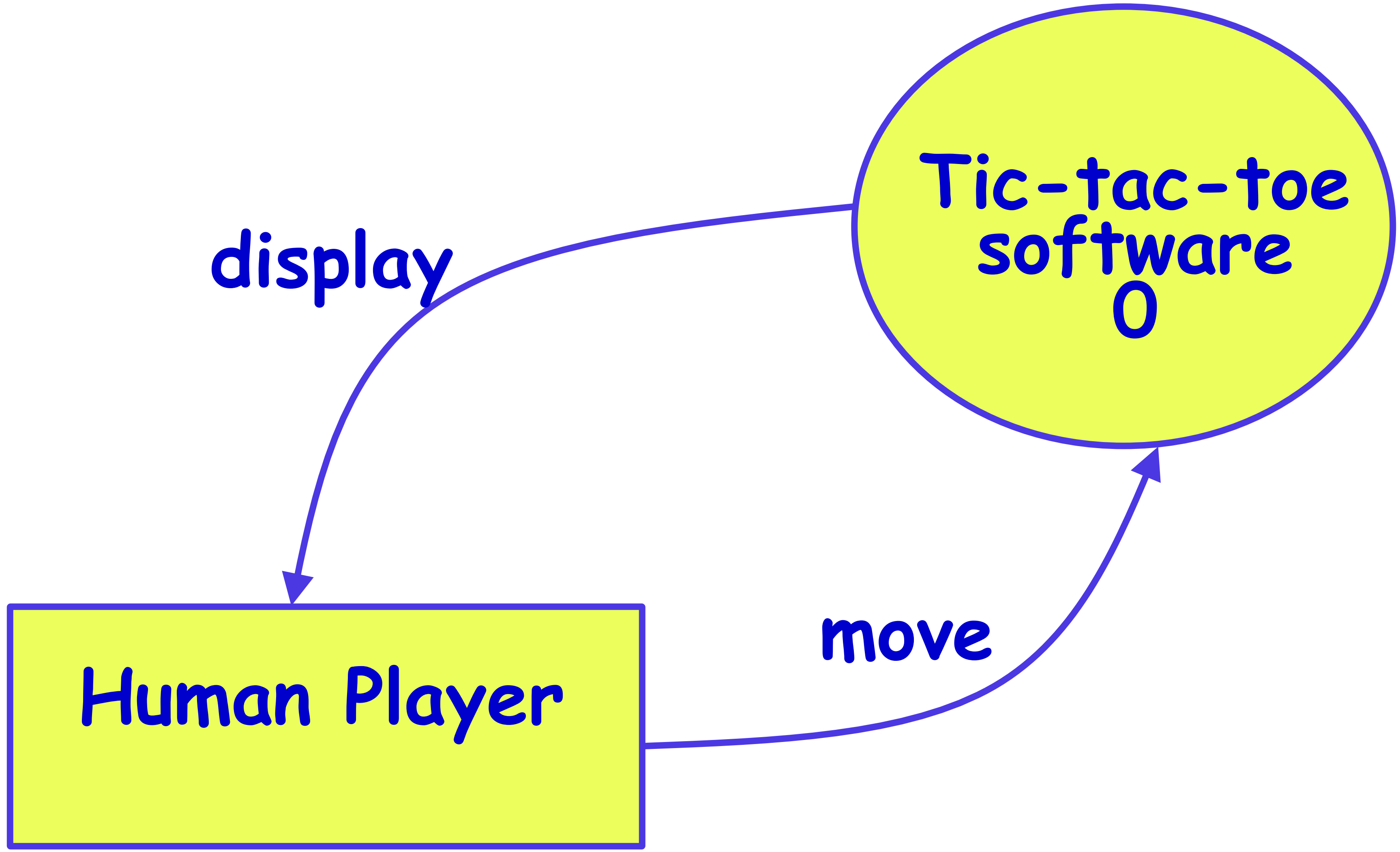
- A human player and the computer make alternate moves on a 3 X 3 square.
- A move consists of marking a previously unmarked square.
- The user inputs a number between 1 and 9 to mark a square
- Whoever is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins.

Example: Tic-Tac-Toe Computer Game

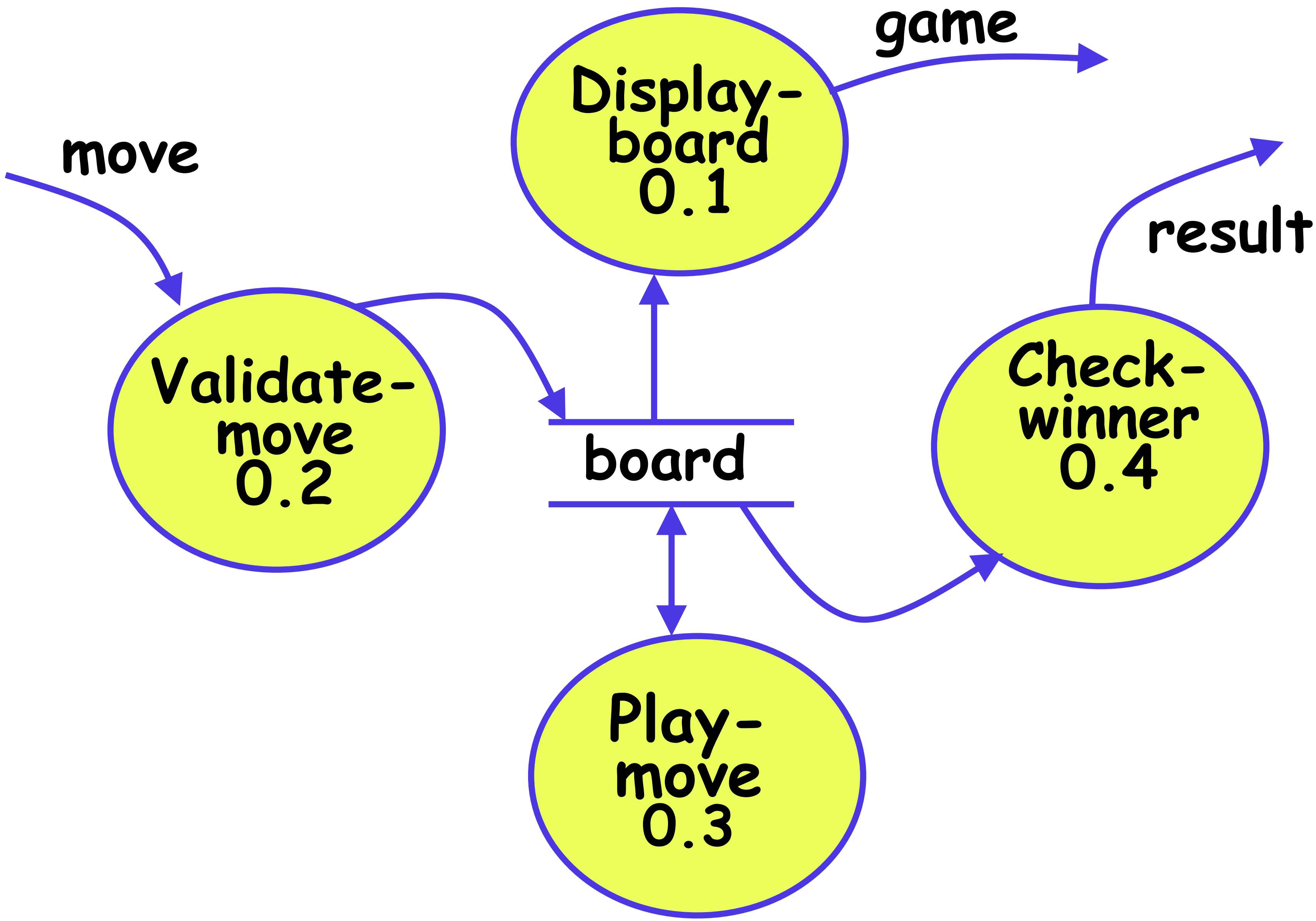


- As soon as either of the human player or the computer wins,
  - A message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
  - And all the squares on the board are filled up,
  - Then the game is drawn.
- The computer always tries to win a game.

Context Diagram for Example



Level 1 DFD


game
Display-
board
0.1
move
result
Validate-
move
0.2
board
Check-
winner
0.4
Play-
move
0.3

Data Dictionary

- Display=game + result
- move = integer
- board = {integer}9
- game = {integer}9
- result=string

Example 3: Trading-House Automation System (TAS)

- A large trading house wants us to develop a software:
  - To automate book keeping activities associated with its business.
- It has many regular customers:
  - Who place orders for various kinds of commodities.

Example 3: Trading-House Automation System (TAS)

- The trading house maintains names and addresses of its regular customers.

- Each customer is assigned a unique customer identification number (CIN).

- As per current practice when a customer places order:
  - The accounts department first checks the **credit-worthiness** of the customer.

Example: Trading-House Automation System (TAS)

- The credit worthiness of a customer is determined:
  - By analyzing the history of his payments to the bills sent to him in the past.

- If a customer is not credit-worthy:
  - His orders are not processed any further
  - An appropriate order rejection message is generated for the customer.

Example: Trading-House Automation System (TAS)

- If a customer is credit-worthy:
  - Items he/she has ordered are checked against the list of items the trading house deals with.

- The items that the trading house does not deal with:
  - Are not processed any further
  - An appropriate message for the customer for these items is generated.

Example: Trading-House Automation System (TAS)

- The items in a customer's order that the trading house deals with:
  - Are checked for availability in inventory.

- If the items are available in the inventory in desired quantities:
  - A bill with the forwarding address of the customer is printed.
  - A material issue slip is printed.

Example: Trading-House Automation System (TAS)

- The customer can produce the material issue slip at the store house:
  - Take delivery of the items.
  - Inventory data adjusted to reflect the sale to the customer.

Example: Trading-House Automation System (TAS)

- If an ordered item is not available in the inventory in sufficient quantity:
  - To be able to fullfill pending orders, store details in a "pending-order" file :
    - out-of-stock items along with quantity ordered.
    - customer identification number

Example: Trading-House Automation System (TAS)

- The purchase department:
  - would periodically issue commands to generate indents.

- When generate indents command is issued:
  - The system should examine the "pending-order" file
  - Determine the orders that are pending
  - Total quantity required for each of the items.

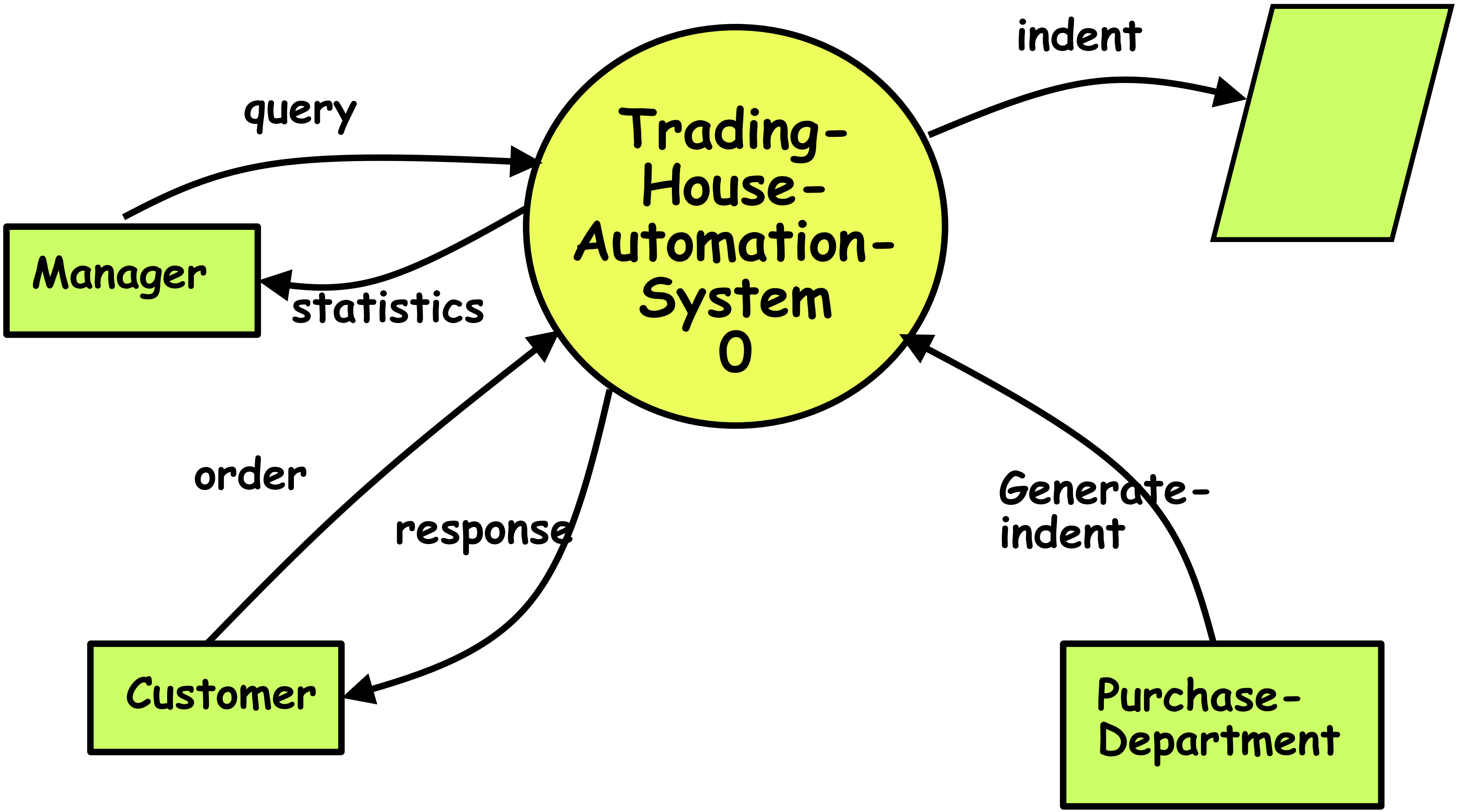
Example: Trading-House Automation System (TAS)

- TAS should find out the addresses of the vendors who supply the required items:
  - Examine the file containing vendor details (their address, items they supply etc.)
  - Print out indents to those vendors.

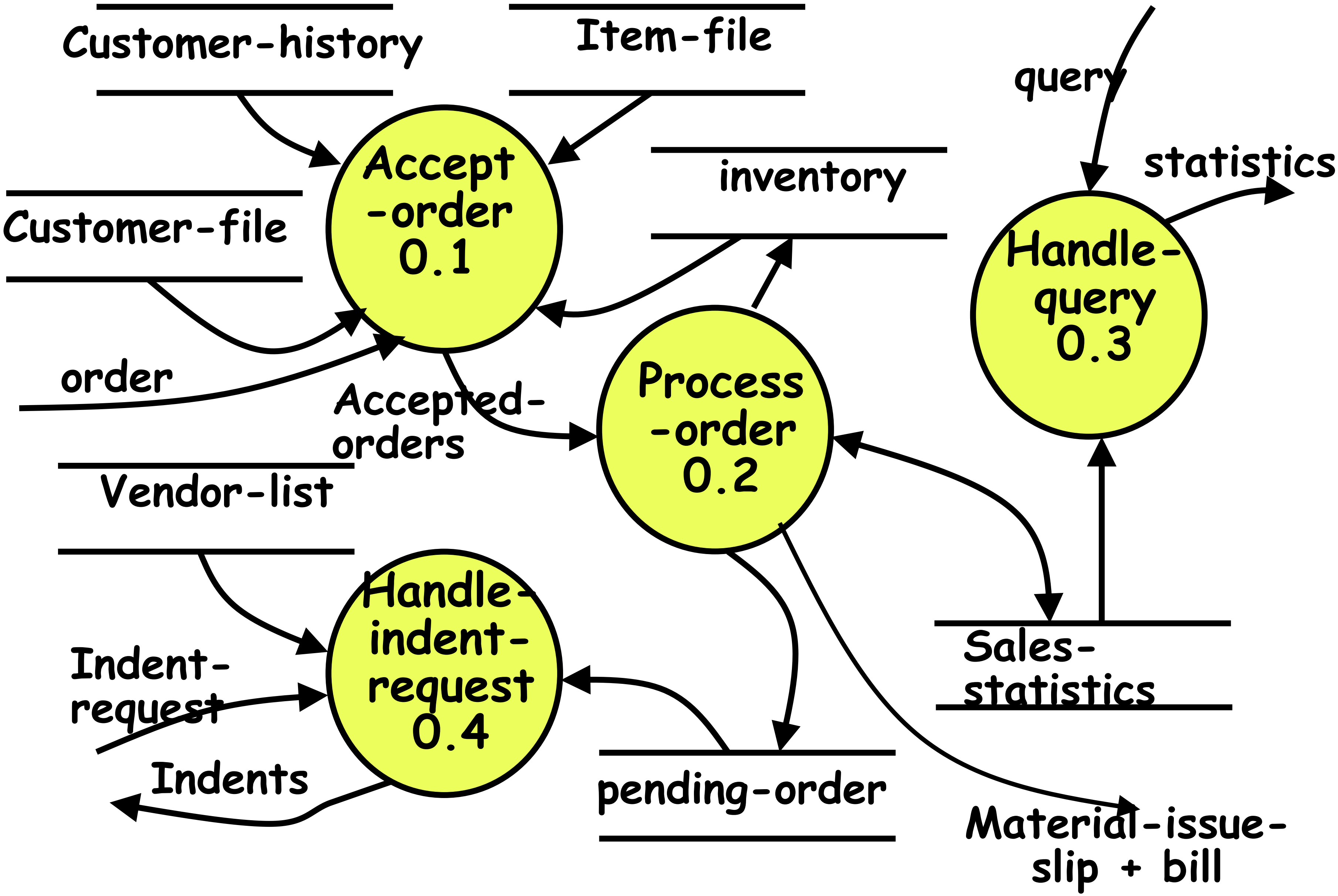
Example: Trading-House Automation System (TAS)

- TAS should also answers managerial queries:
  - Statistics of different items sold over any given period of time
  - Corresponding quantity sold and the price realized.

Context Diagram



Level 1 DFD


Customer-history
Item-file
query
statistics
inventory
Customer-file
Accept -order 0.1
Handle-query 0.3
Process -order 0.2
order
Accepted-orders
Vendor-list
Handle-indent-request 0.4
Indent-request
Indents
Sales-statistics
pending-order
Material-issue-slip + bill

Example: Data Dictionary

- response: [bill + material-issue-slip, reject-message]
- query: period /* query from manager regarding sales statistics*/
- period: [date+date,month,year,day]
- date: year + month + day
- year: integer
- month: integer
- day: integer
- order: customer-id + {items + quantity}*
- accepted-order: order /* ordered items available in inventory */
- reject-message: order + message /* rejection message */
- pending-orders: customer-id + {items+quantity}*
- customer-address: name+house#+street#+city+pin

Example: Data Dictionary

- item-name: string
- house#: string
- street#: string
- city: string
- pin: integer
- customer-id: integer
- bill: {item + quantity + price}* + total-amount + customer-address
- material-issue-slip: message + item + quantity + customer-address
- message: string
- statistics: {item + quantity + price }*
- sales-statistics: {statistics}*
- quantity: integer

Guidelines For Constructing DFDs

- Context diagram should represent the system as a single bubble:
  - Many beginners commit the mistake of drawing more than one bubble in the context diagram.

Guidelines For Constructing DFDs

- All external entities should be represented in the context diagram:
    - External entities should not appear at any other level of DFD.
- Only 3 to 7 bubbles should be allowed:
    - Each bubble should be decomposed to between 3 and 7 bubbles.

Guidelines For Constructing DFDs

- A common mistake committed by many beginners:
  - Attempting to represent **control information** in a DFD.
  - e.g. trying to represent the order in which different functions are executed.

Guidelines For Constructing DFDs

- A DFD model does not represent control information:
  - When or in what order different functions (processes) are invoked
  - The conditions under which different functions are invoked are not represented.
  - For example, a function might invoke one function or another depending on some condition.
  - Many beginners try to represent this aspect by drawing an arrow between the corresponding bubbles.

Guidelines For Constructing DFDs

- All functions of the system must be captured in the DFD model:
  - **No function specified in the SRS document should be overlooked.**
- Only those functions specified in the SRS document should be represented:
  - **Do not assume extra functionality of the system not specified by the SRS document.**

Commonly Made Errors

- Unbalanced DFDs
- Forgetting to name the data flows
- Unrepresented functions or data
- External entities appearing at higher level DFDs
- Trying to represent control aspects
- Context diagram having more than one bubble
- A bubble decomposed into too many bubbles in the next level
- Terminating decomposition too early

Shortcomings of the DFD Model

- DFD models suffer from several shortcomings:

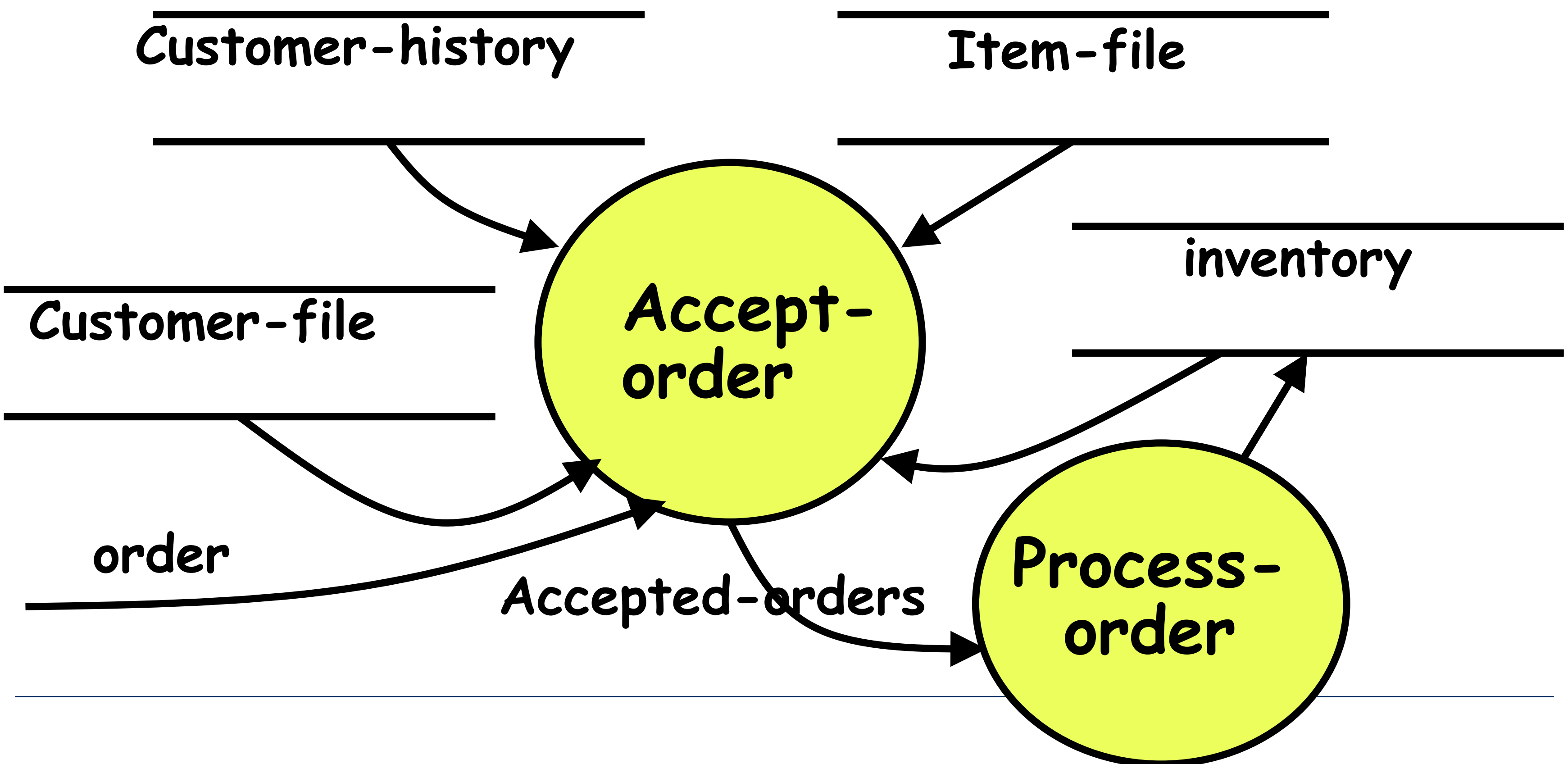
- DFDs leave ample scope to be imprecise.
  - In a DFD model, we infer about the function performed by a bubble from its label.
  - A label may not capture all the functionality of a bubble.

Shortcomings of the DFD Model

- For example, a bubble named find-book-position has only intuitive meaning:
  - Does not specify several things:
    - What happens when some input information is missing or is incorrect.
    - Does not convey anything regarding what happens when book is not found
    - or what happens if there are books by different authors with the same book title.

Shortcomings of the DFD Model

- Control information is not represented:
  - For instance, order in which inputs are consumed and outputs are produced is not specified.



Shortcomings of the DFD Technique

- A DFD does not specify **synchronization** aspects:
  - For instance, the DFD in TAS example does not specify:
    - Whether process-order may wait until the accept-order produces data
    - Whether accept-order and handle-order may proceed simultaneously with some buffering mechanism between them.

DFD Tools

- Several commercial and free tools available.

- Commercial:
  - Visio
  - Smartdraw
  - Edraw
  - Visual analyst

- Free:
  - Dia (GNU open source)

Structured Design

- The aim of structured design
  - **Transform the results of structured analysis (DFD representation) into a structure chart.**

- A **structure chart** represents the **software architecture**:
  - **Various modules** making up the system,
  - **Module dependency** (i.e. which module calls which other modules),
  - **Parameters** passed among different modules.

Structure Chart

- Structure chart representation
  - Easily implementable using programming languages.

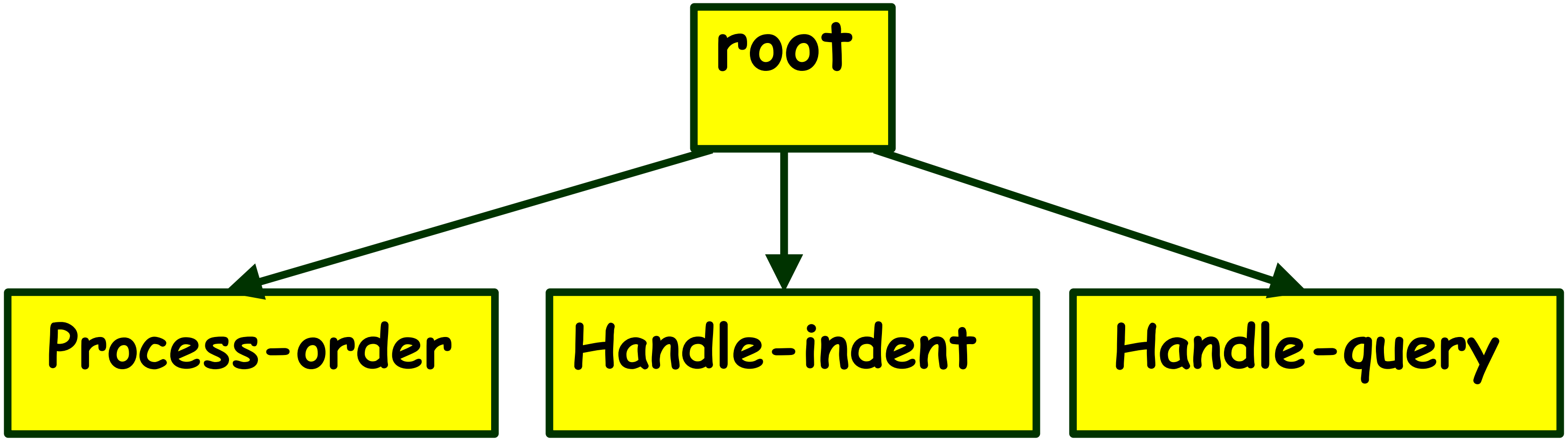
- Main focus of a **structure chart**:
  - Define the **module structure** of a software,
  - **Interaction** among different modules,

Basic Building Blocks of Structure Chart

- Rectangular box:
  - A rectangular box represents a module.
  - Annotated with the name of the module it represents.

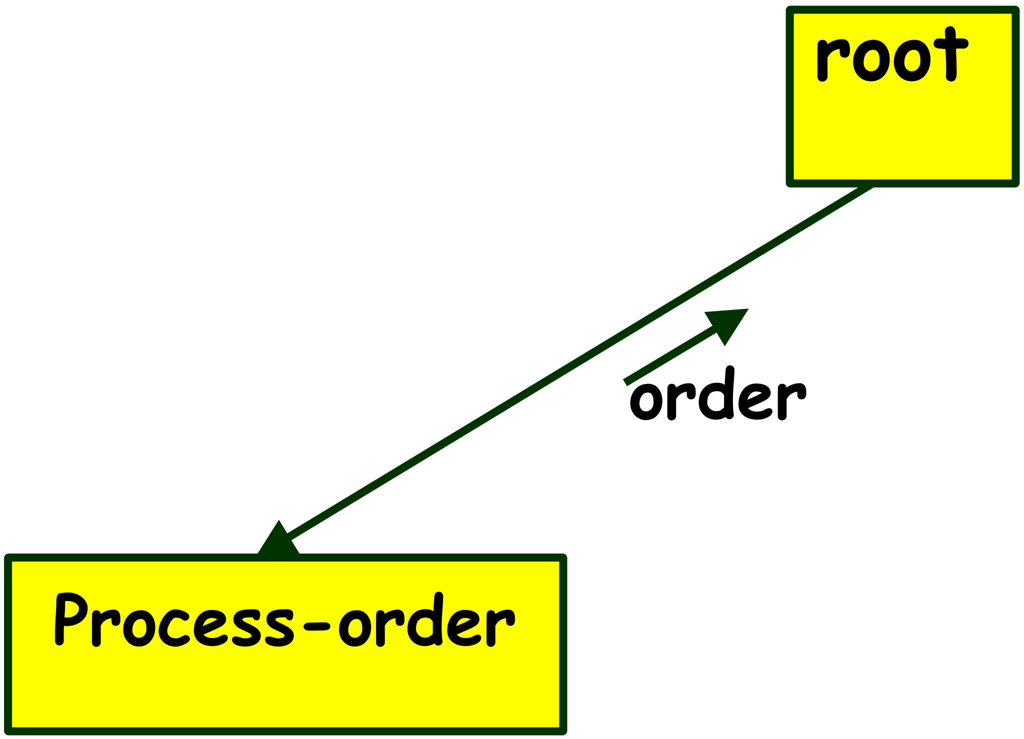
Arrows

- An arrow between two modules implies:
  - **During execution control is passed from one module to the other in the direction of the arrow.**



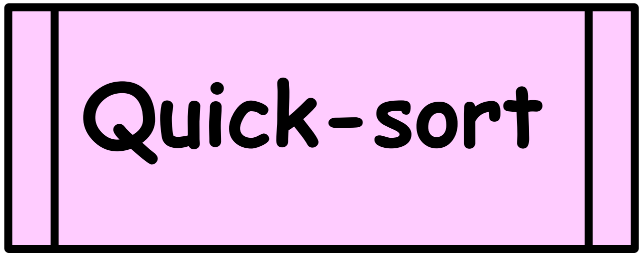
Data Flow Arrows

- Data flow arrows represent:
  - Data passing from one module to another in the direction of the arrow.



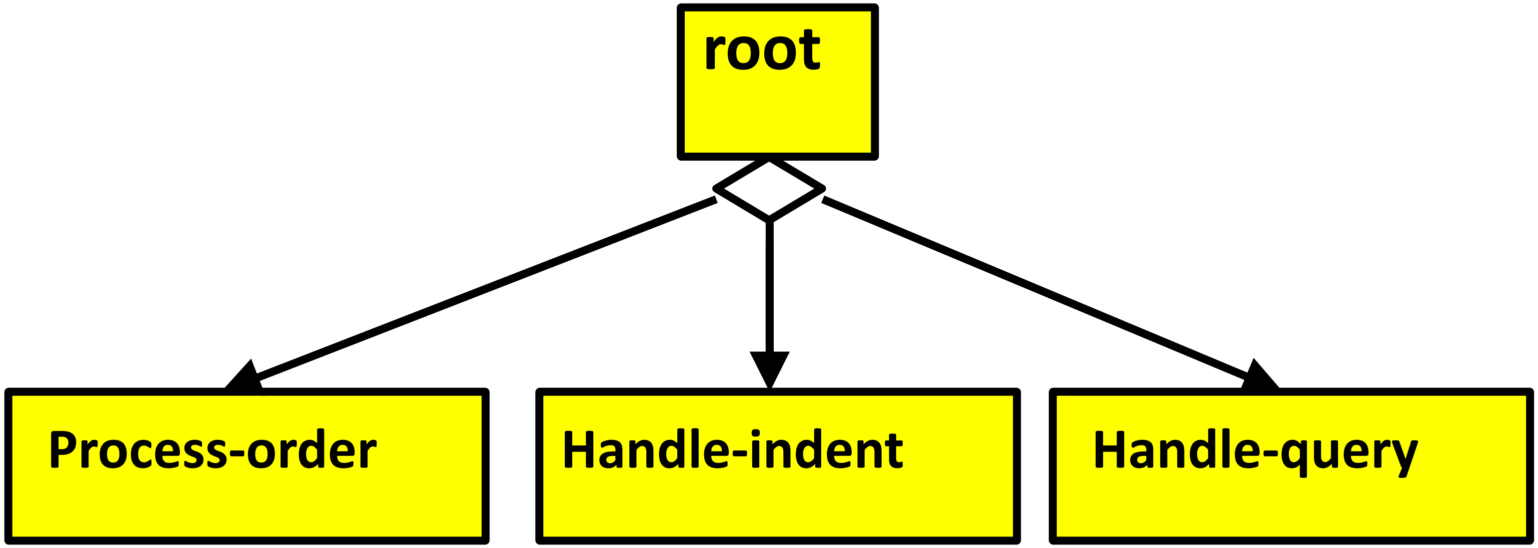
Library Modules

- Library modules represent frequently called modules:
  - A rectangle with double side edges.
  - Simplifies drawing when a module is called by several modules.



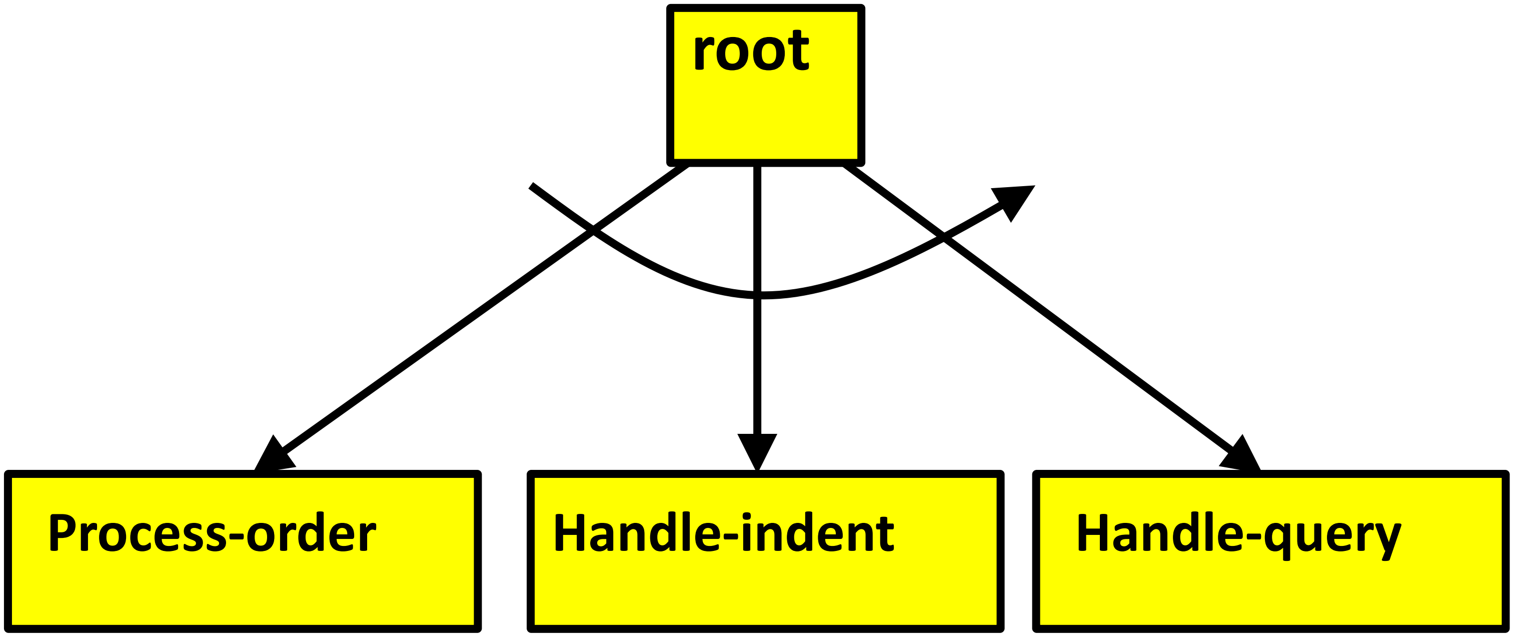
Selection

- The diamond symbol represents:
  - **One module of several modules connected to the diamond symbol is invoked depending on some condition.**



Repetition

- A loop around control flow arrows denotes that the concerned modules are invoked repeatedly.



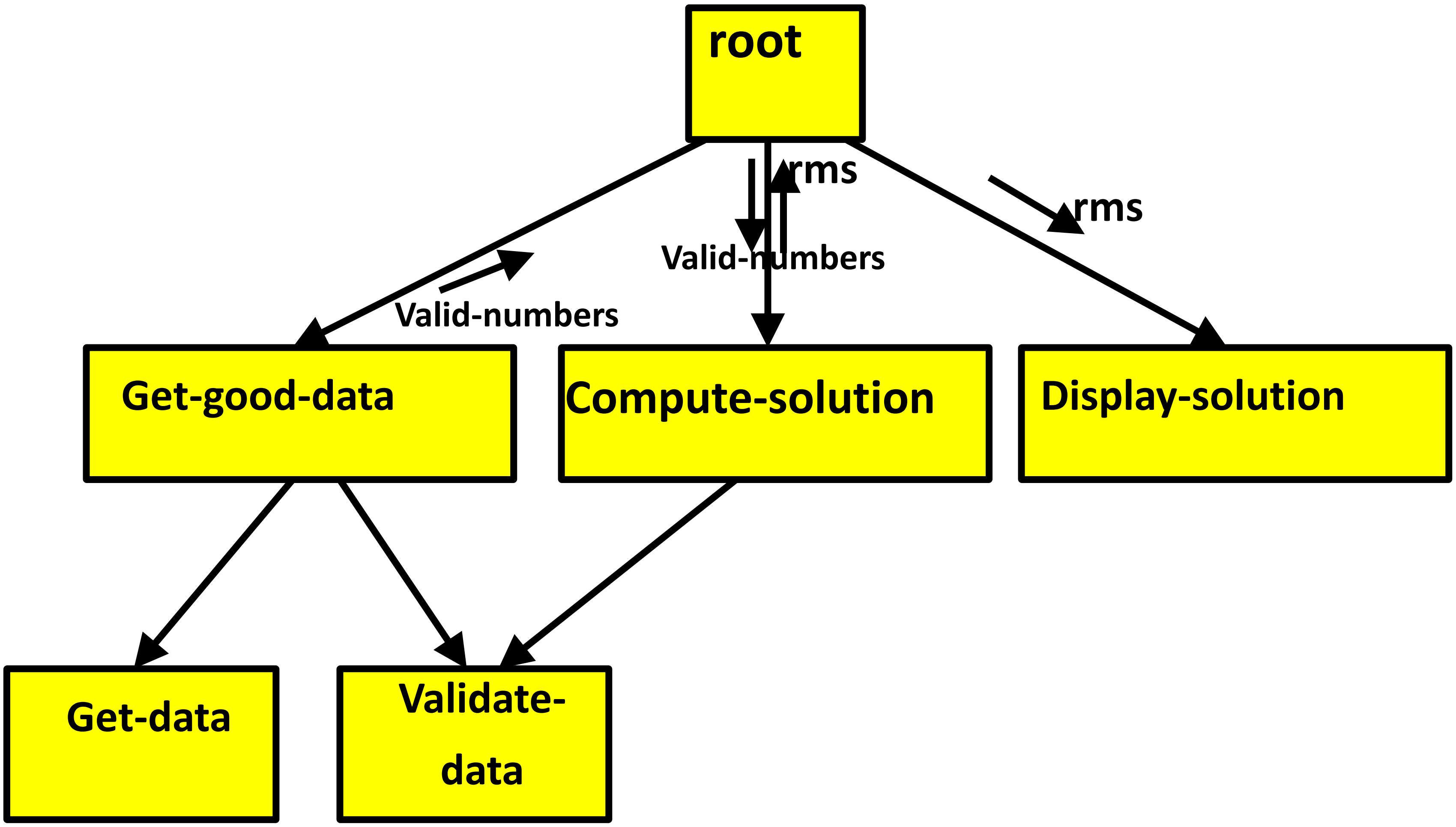
Structure Chart

- There is only one module at the top:
  - the root module.
- There is at most one control relationship between any two modules:
  - if module **A** invokes module **B**,
  - Module **B** cannot invoke module **A**.
- The main reason behind this restriction:
  - Modules in a structure chart should be arranged in **layers** or **levels.**

Structure Chart

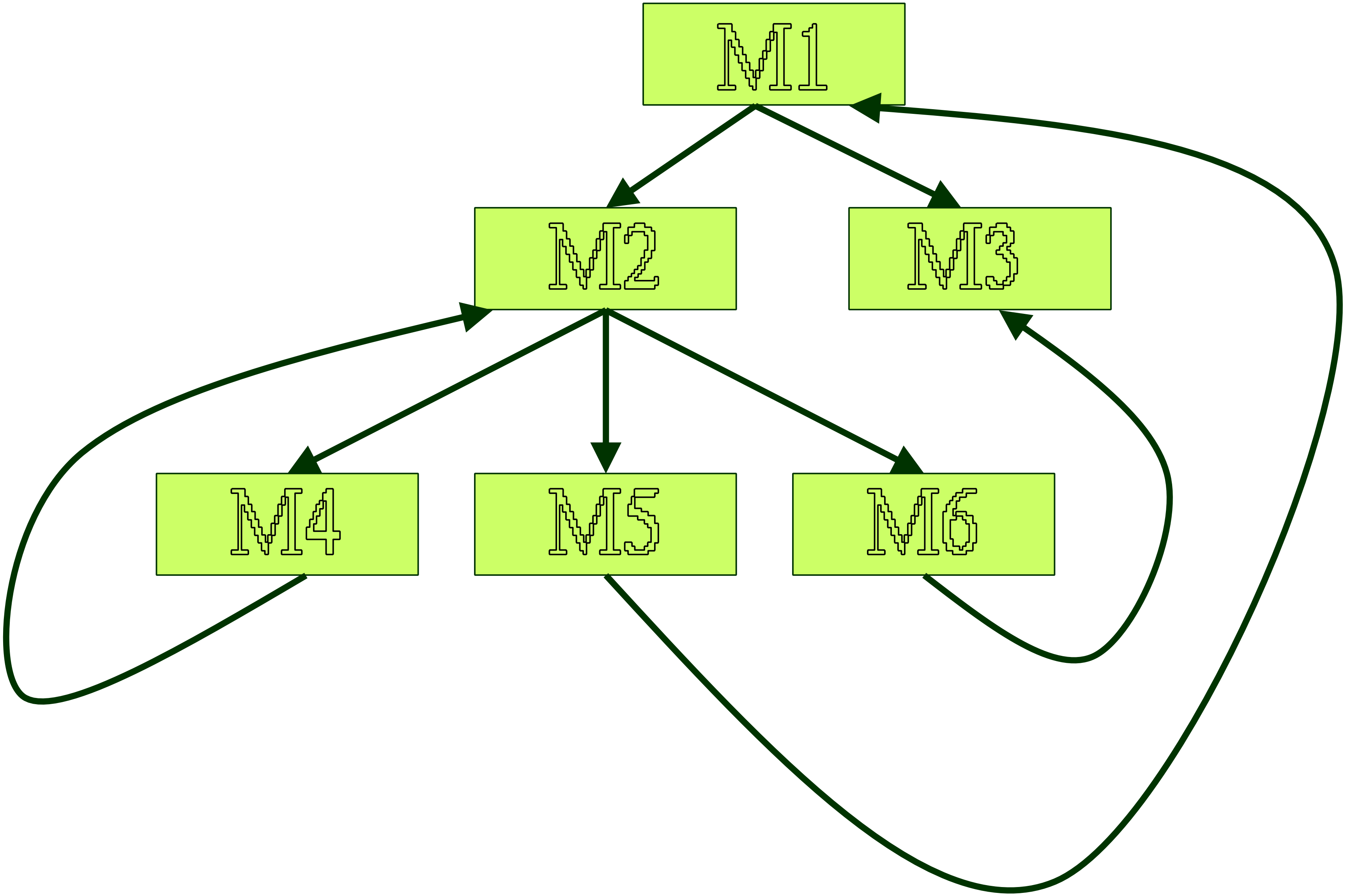
- The principle of abstraction:
  - does not allow lower-level modules to invoke higher-level modules:
  - But, two higher-level modules can invoke the same lower-level module.

Example



root
rms
rms
Valid-numbers
Valid-numbers
Get-good-data
Compute-solution
Display-solution
Get-data
Validate-data

Bad Design



Shortcomings of Structure Chart

- By looking at a structure chart:
  - we can not tell the order in which the different modules are invoked.
  - we can not say how many times a module calls another module

Transformation of a DFD Model into Structure Chart

- Two strategies exist to guide transformation of a DFD into a structure chart:

  - **Transform Analysis**
  - **Transaction Analysis**

Transform Analysis

- The first step in transform analysis:
  - Divide the DFD into 3 parts:
    - **input,**
    - **logical processing,**
    - **output.**

Transform Analysis

- Input portion in the DFD:
  - processes which convert input data from physical to logical form.
  - e.g. read characters from the terminal and store in internal tables or lists.

- Each input portion:
  - called an afferent branch.
  - Possible to have more than one afferent branch in a DFD.

Transform Analysis

- Output portion of a DFD:
  - transforms output data from logical form to physical form.
    - e.g., from list or array into output characters.
  - Each output portion:
    - called an efferent branch.

- The remaining portions of a DFD
  - called central transform

Transform Analysis

- Derive structure chart by drawing one functional component for:
  - **the central transform,**
  - **each afferent branch,**
  - **each efferent branch.**

Transform Analysis

- Identifying the highest level input and output transforms:
  - requires experience and skill.

- Some guidelines:
  - **Trace inputs until a bubble is found whose output cannot be deduced from the inputs alone.**
  - **Processes which validate input are not central transforms.**
  - **Processes which sort input or filter data from it are.**

Transform Analysis

- First level of structure chart:
  - Draw a box for each input and output units
  - A box for the central transform.

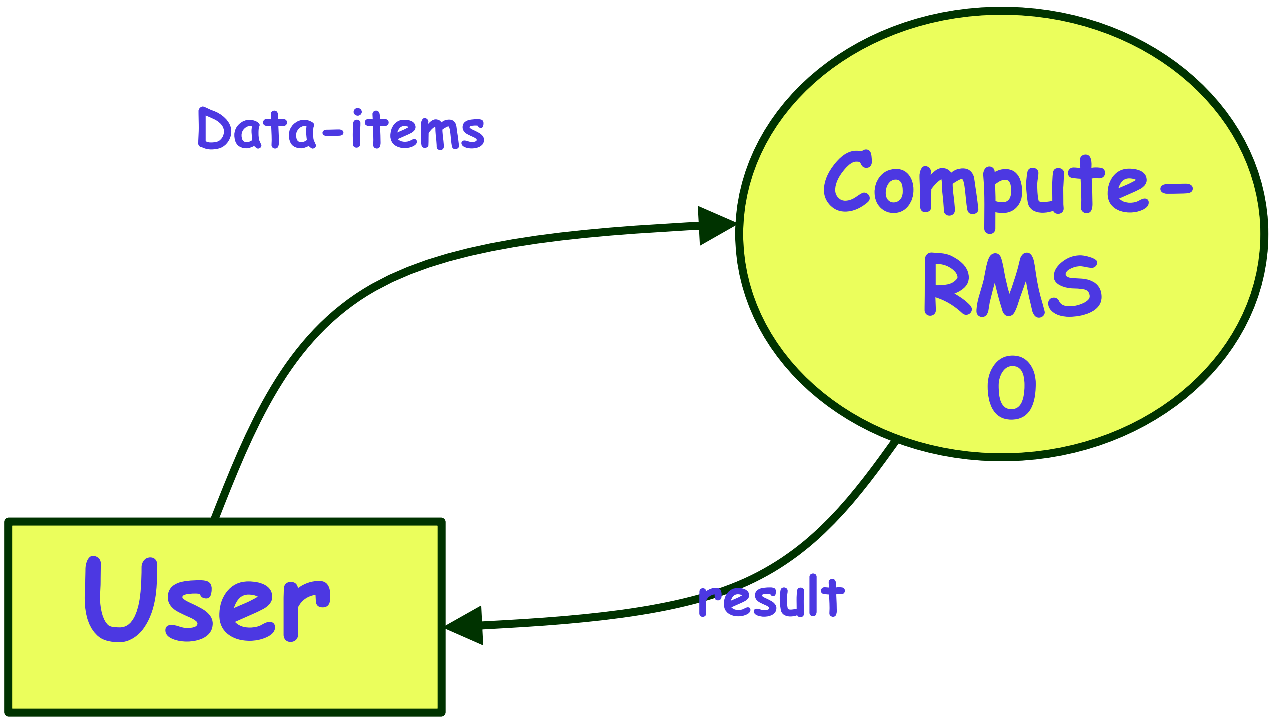
- Next, refine the structure chart:
  - Add subfunctions required by each high-level module.
  - Many levels of modules may required to be added.

Factoring

- The process of breaking functional components into subcomponents.

- Finally check:
  - Whether all bubbles in DFD have been mapped to modules.

Example 1: RMS Calculating Software

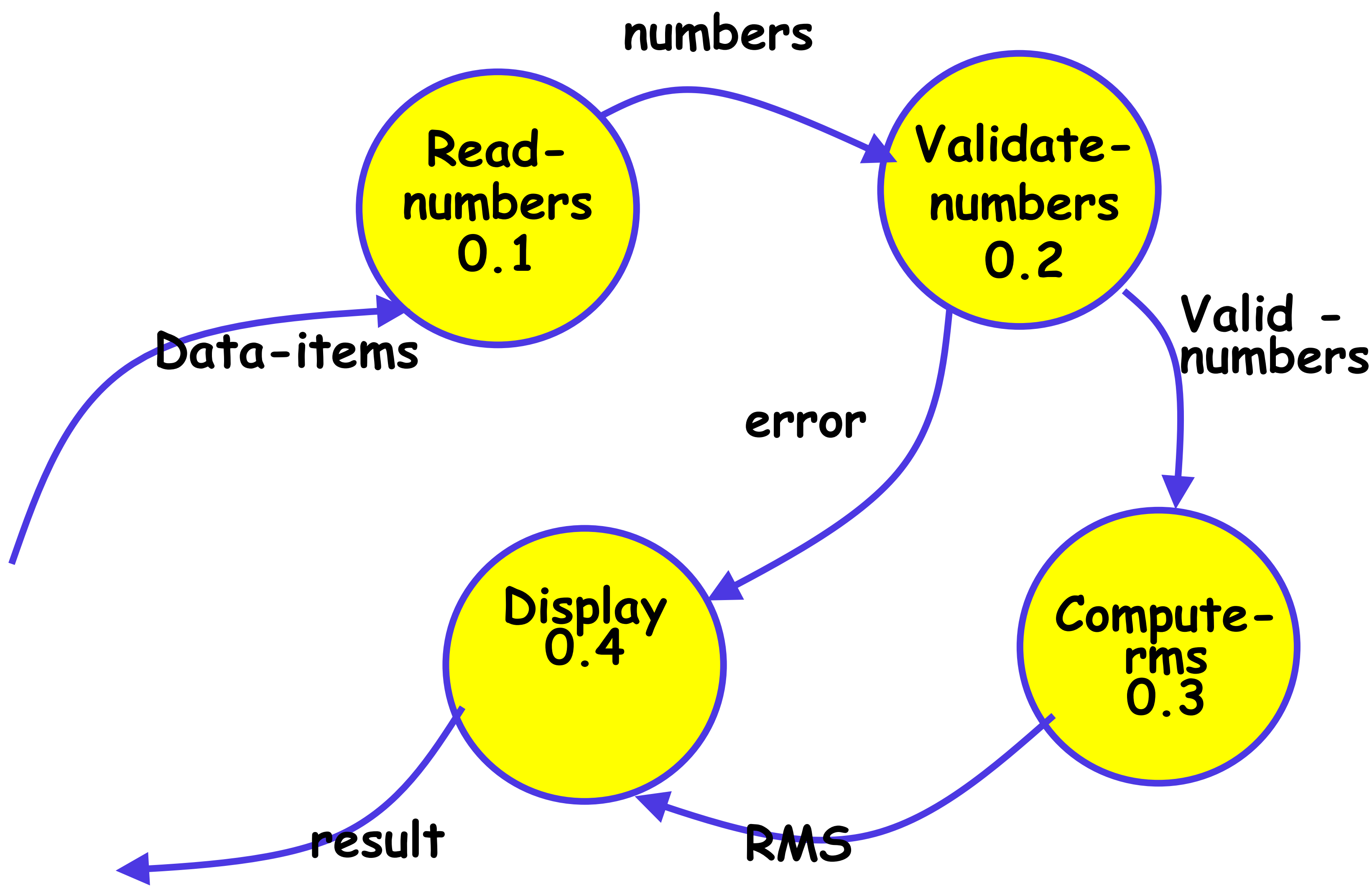


Context Diagram

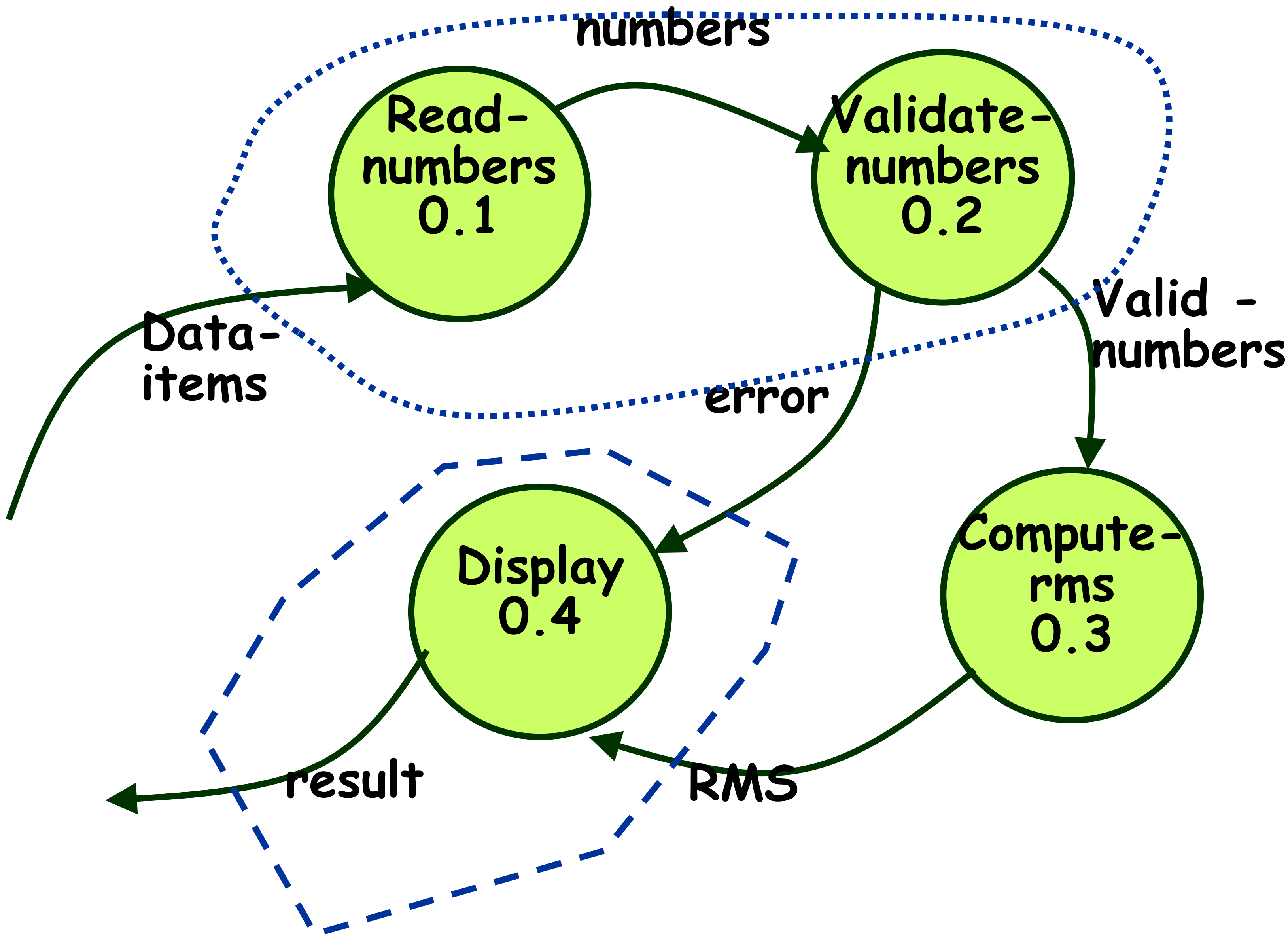
Example 1: RMS Calculating Software

- From a cursory analysis of the problem description,
  - easy to see that the system needs to perform:
    - accept the input numbers from the user,
    - validate the numbers,
    - calculate the root mean square of the input numbers,
    - display the result.

Example 1: RMS Calculating Software



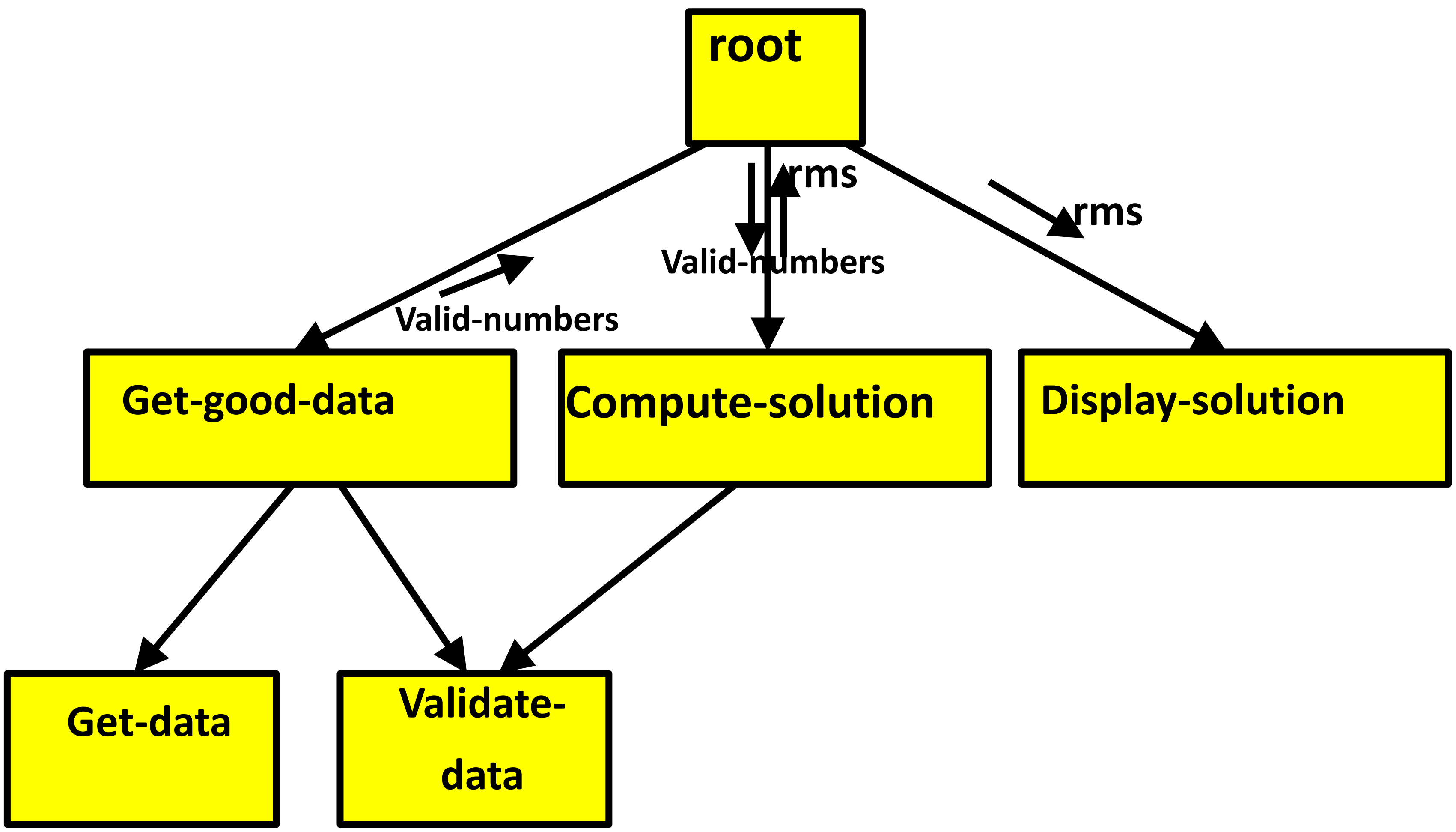
Example 1: RMS Calculating Software



Example 1: RMS Calculating Software

- By observing the **level 1** DFD:
  - Identify ***read-number*** and ***validate-number*** bubbles as the **afferent** branch
  - ***Display*** as the **efferent** branch.

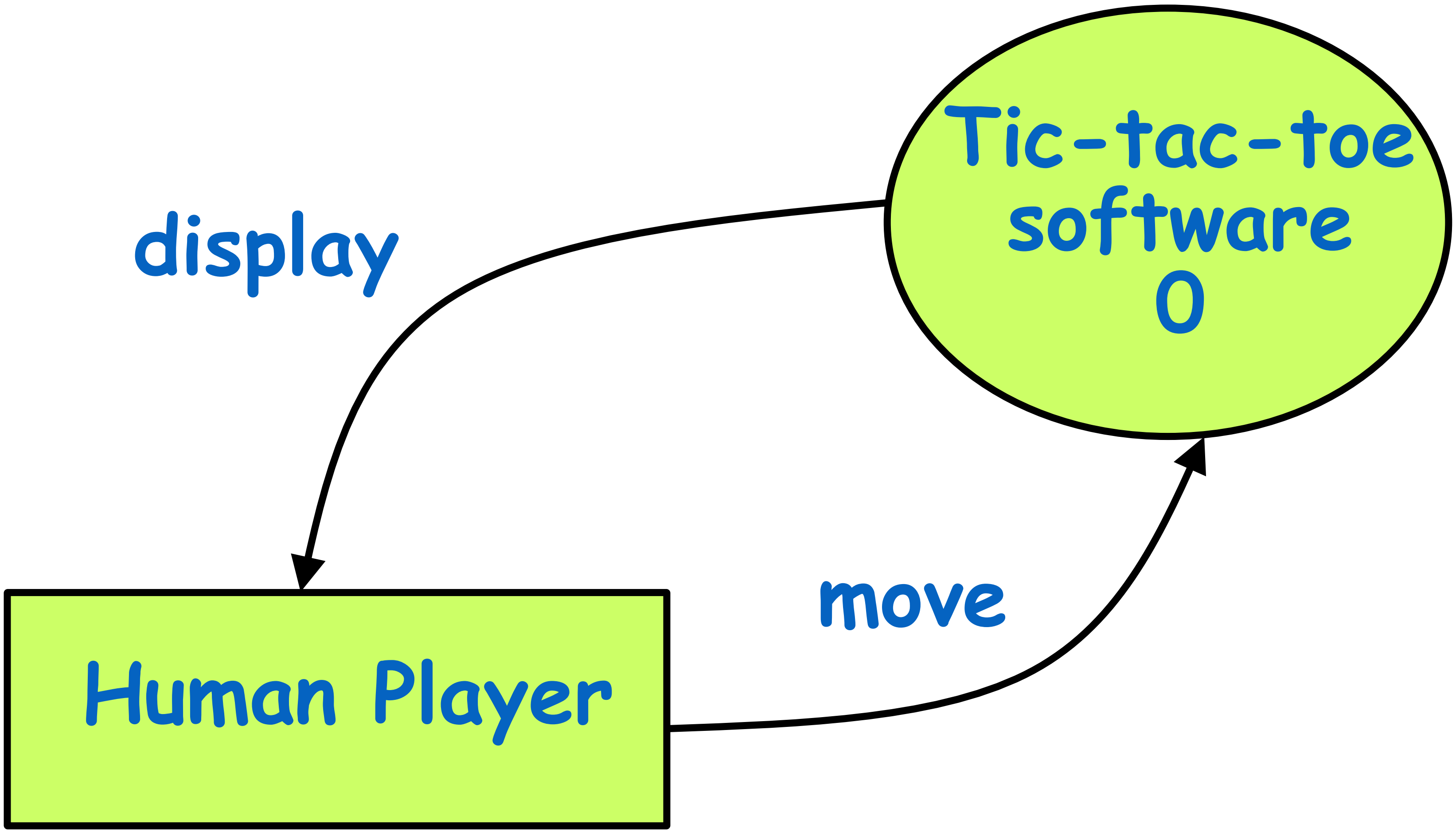
Example 1: RMS Calculating Software



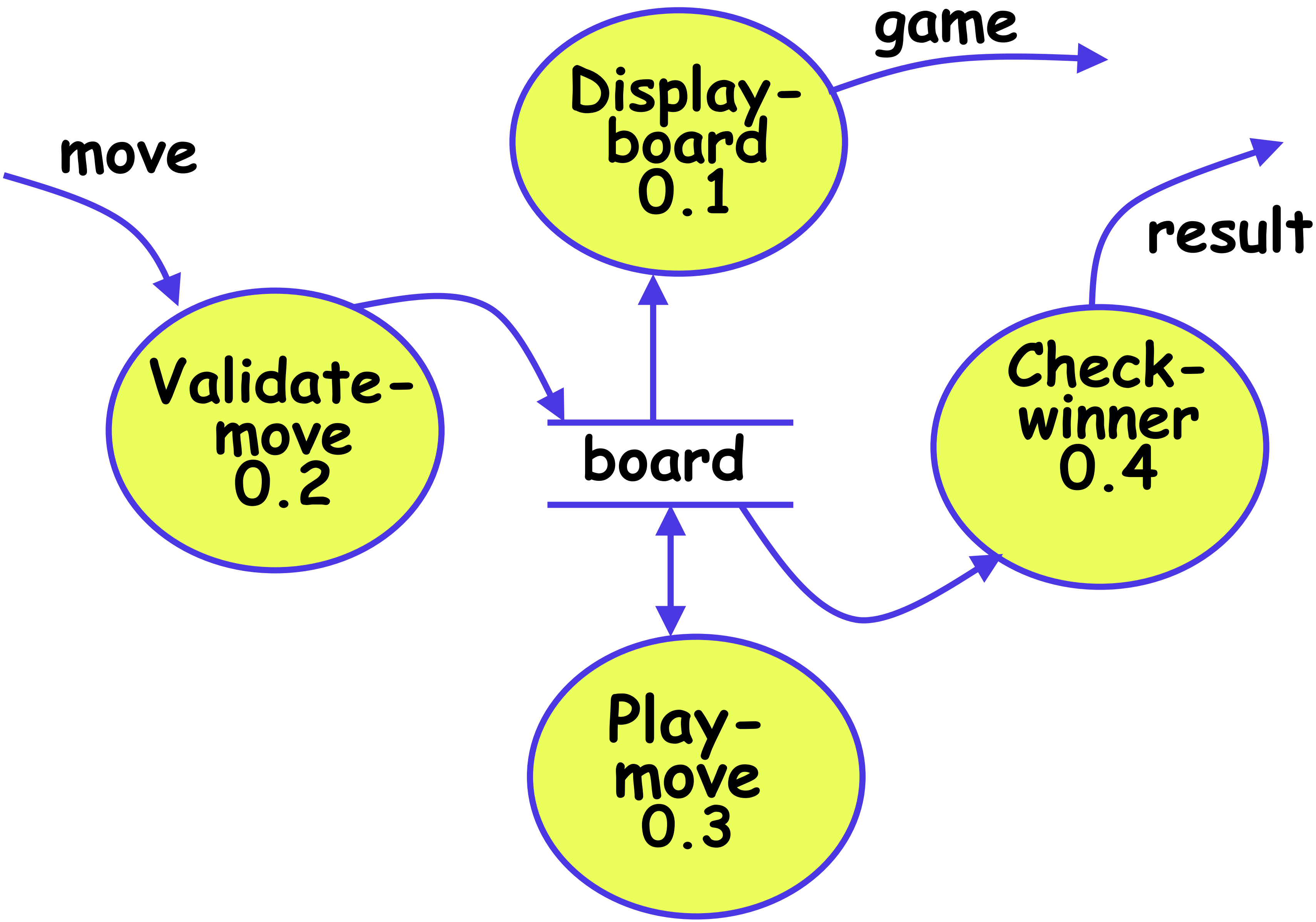
Example 2: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
  - A message congratulating the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
  - And all the squares on the board are filled up,
  - Then the game is drawn.
- The computer always tries to win a game.

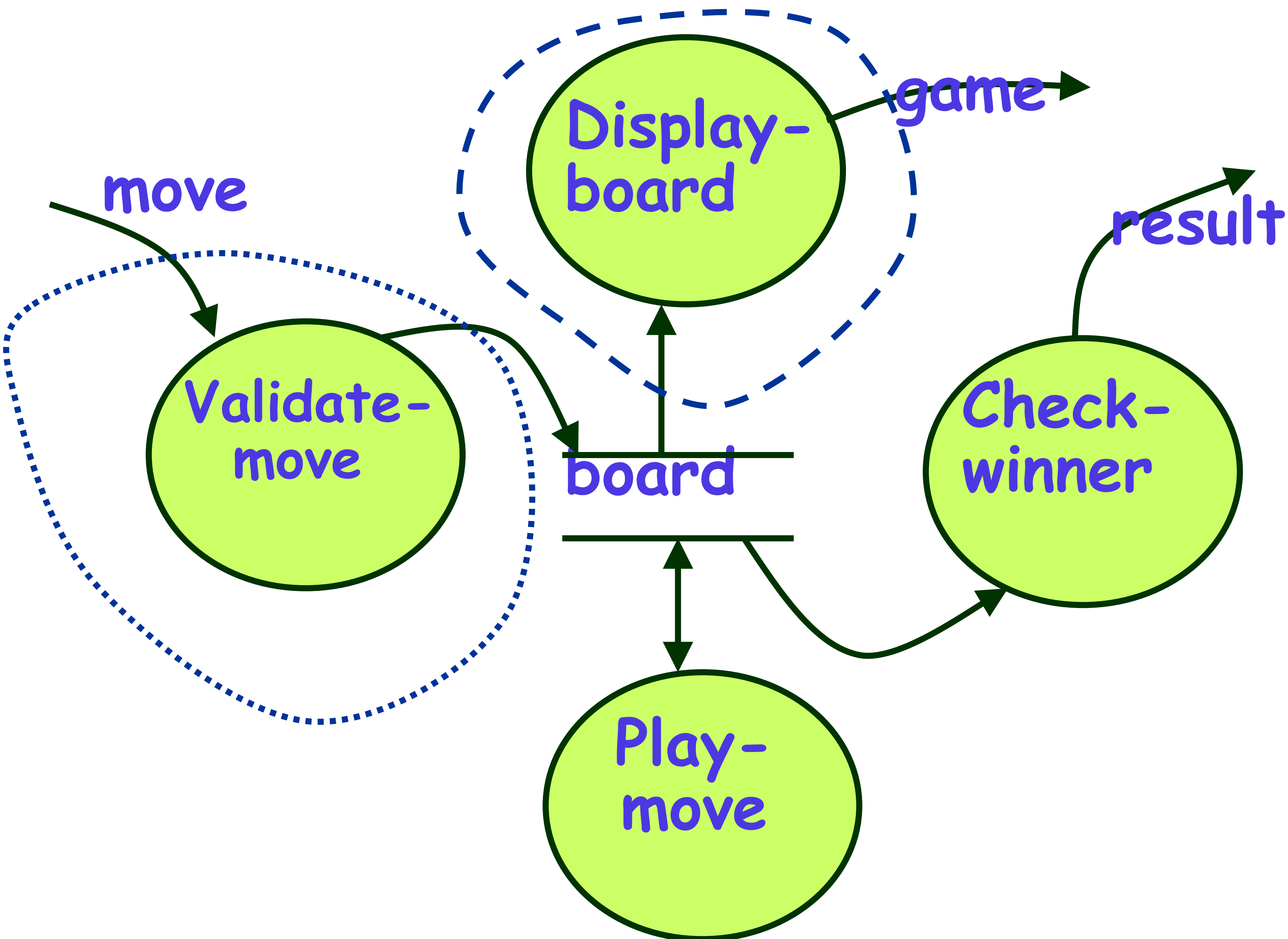
Context Diagram for Example 2



Level 1 DFD

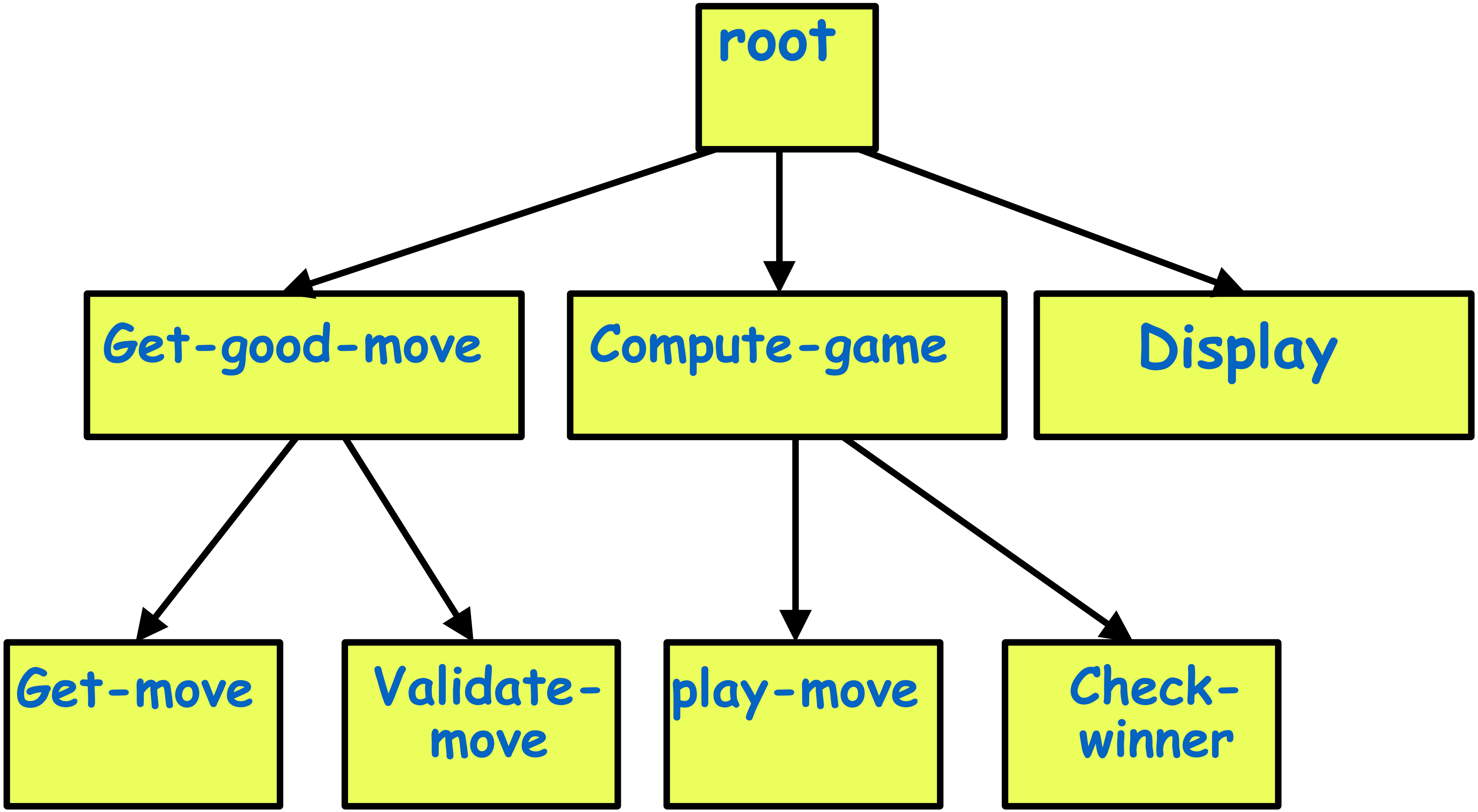

game
Display-
board
0.1
move
result
Validate-
move
0.2
board
Check-
winner
0.4
Play-
move
0.3

Level 1 DFD



game
Display-
board
move
result
Validate-
move
board
Check-
winner
Play-
move

Structure Chart



Transaction Analysis

- Useful for designing transaction processing programs.

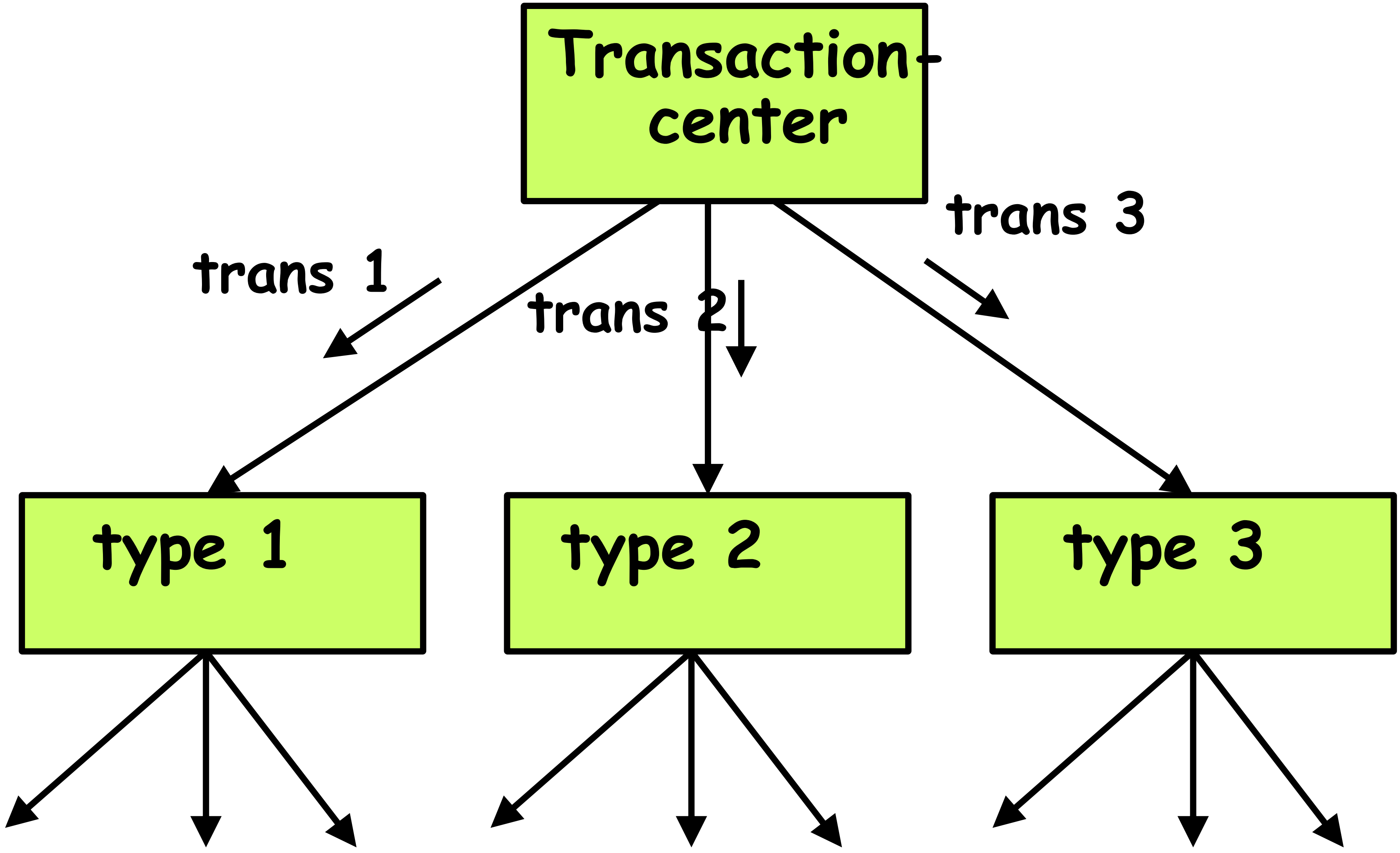
  - **Transform-centered systems:**
    - Characterized by **similar processing steps for every data item** processed by input, process, and output bubbles.

  - **Transaction-driven systems:**
    - **One of several possible paths** through the DFD is traversed depending upon the input data value.

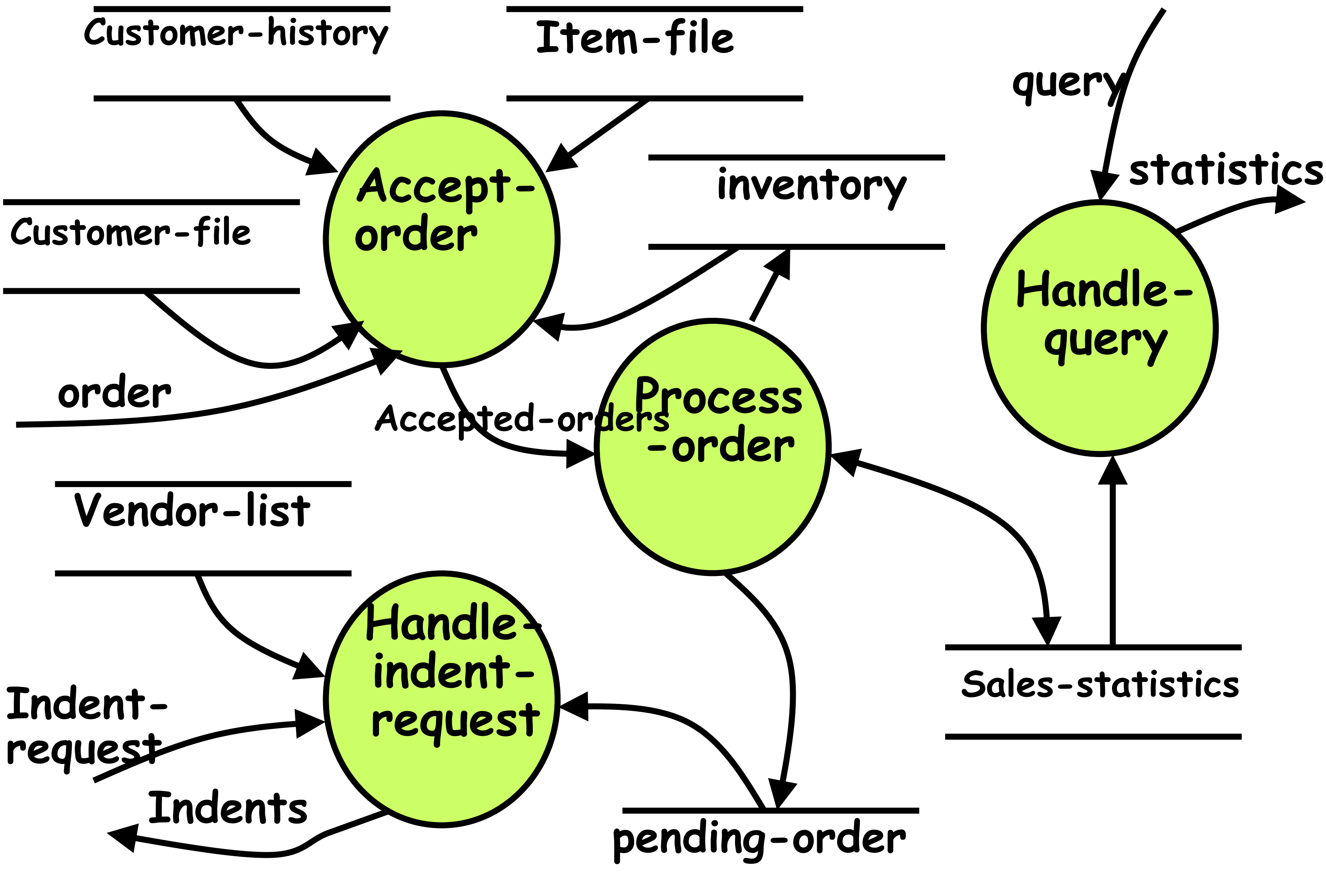
Transaction Analysis

- **Transaction:**
  - Any input data value that triggers an action:
  - For example, selected menu options might trigger different functions.
  - Represented by a tag identifying its type.
- Transaction analysis uses this tag to divide the system into:
  - **Several transaction modules**
  - **One transaction-center module.**

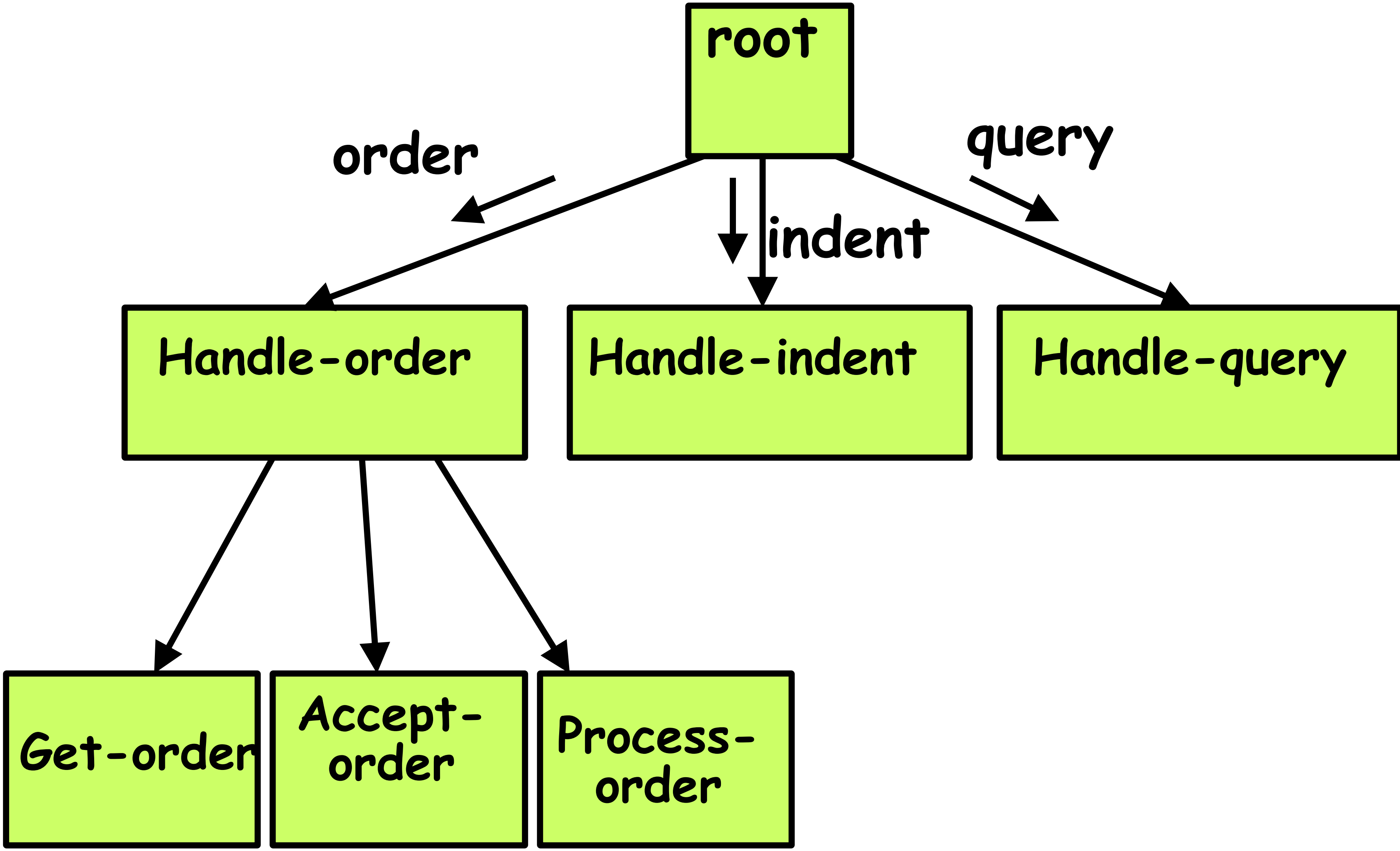
Transaction analysis



Level 1 DFD for TAS



Structure Chart



Summary

- We discussed a function-oriented software design methodology:
  - Structured Analysis/Structured Design(SA/SD)
  - Incorporates features from some important design methodologies.

- SA/SD consists of two parts:
  - Structured analysis
  - Structured design.

Summary

- The goal of structured analysis:
  - functional decomposition of the system.

- Results of structured analysis:
  - represented using Data Flow Diagrams (DFDs).

Summary

- During structured design,
  - The DFD representation is transformed to a structure chart representation.
- DFDs are very popular:
  - Because it is a very simple technique.

Summary

- A DFD model:
  - Difficult to implement using a programming language:
  - Structure chart representation -implemented using a programming language.

Summary

- Several CASE tools are available:
  - Support structured analysis and design.
  - Maintain the data dictionary,
  - Check whether DFDs are balanced or not.