Structured Design

What is Software Design?

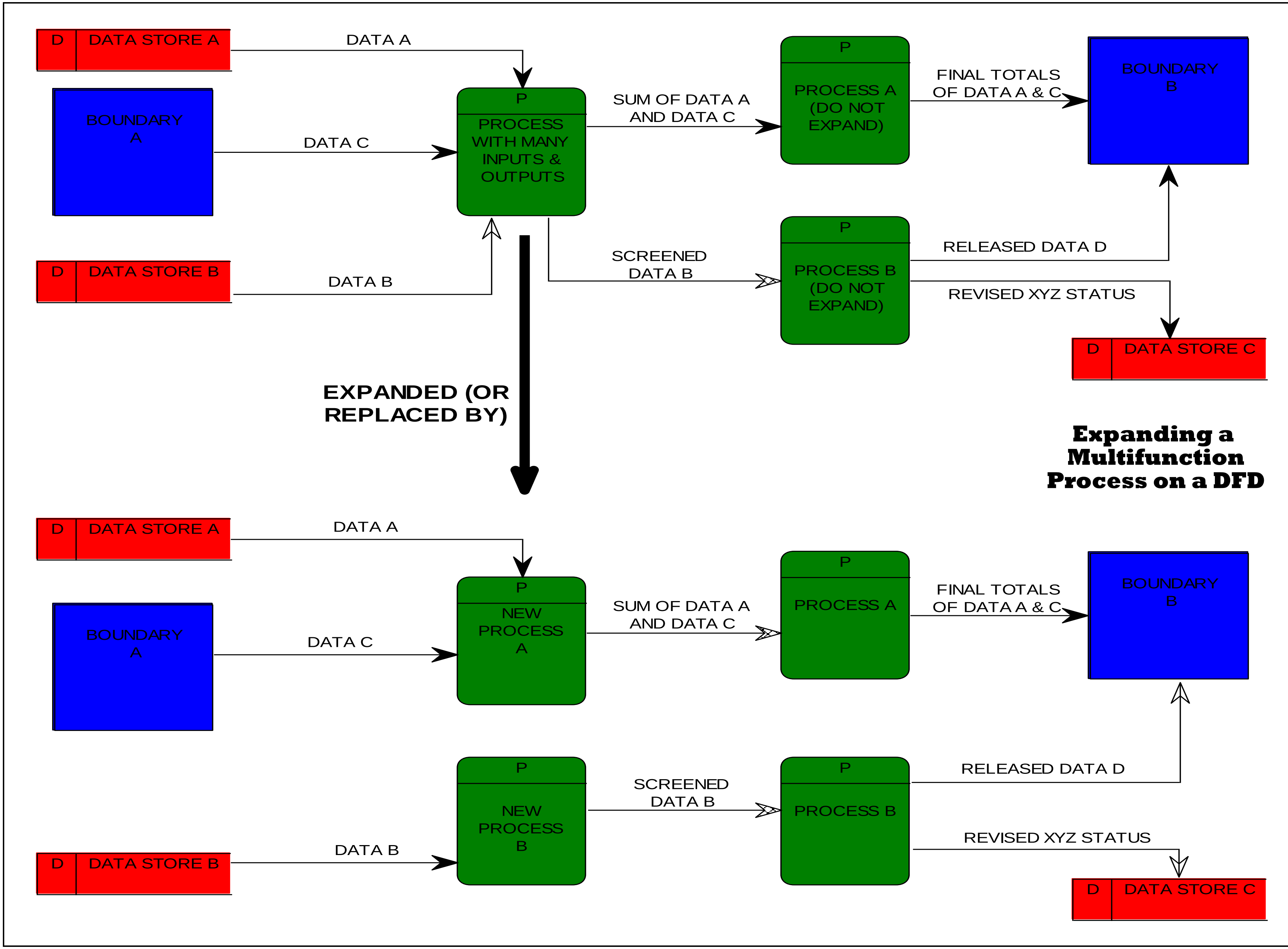
- **Introduction**
  - Software design consists of two components, modular design and packaging.
    - **Modular design** is the decomposition of a program into modules.
    - A **module** is a group of executable instructions with a single point of entry and a single point of exit.
    - **Packaging** is the assembly of data, process, interface, and geography design specifications for each module.

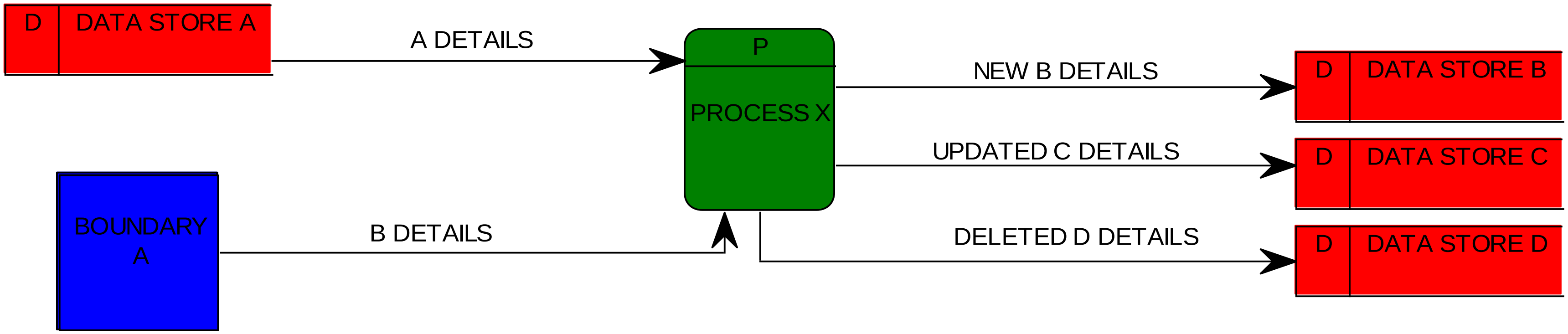
Structured Design

- **Introduction**
  - Structured design was developed by Ed Yourdon and Larry Constantine.
    - This technique deals with the size and complexity of a program by breaking up a the program into a hierarchy of modules that result in a computer program that is easier to implement and maintain.

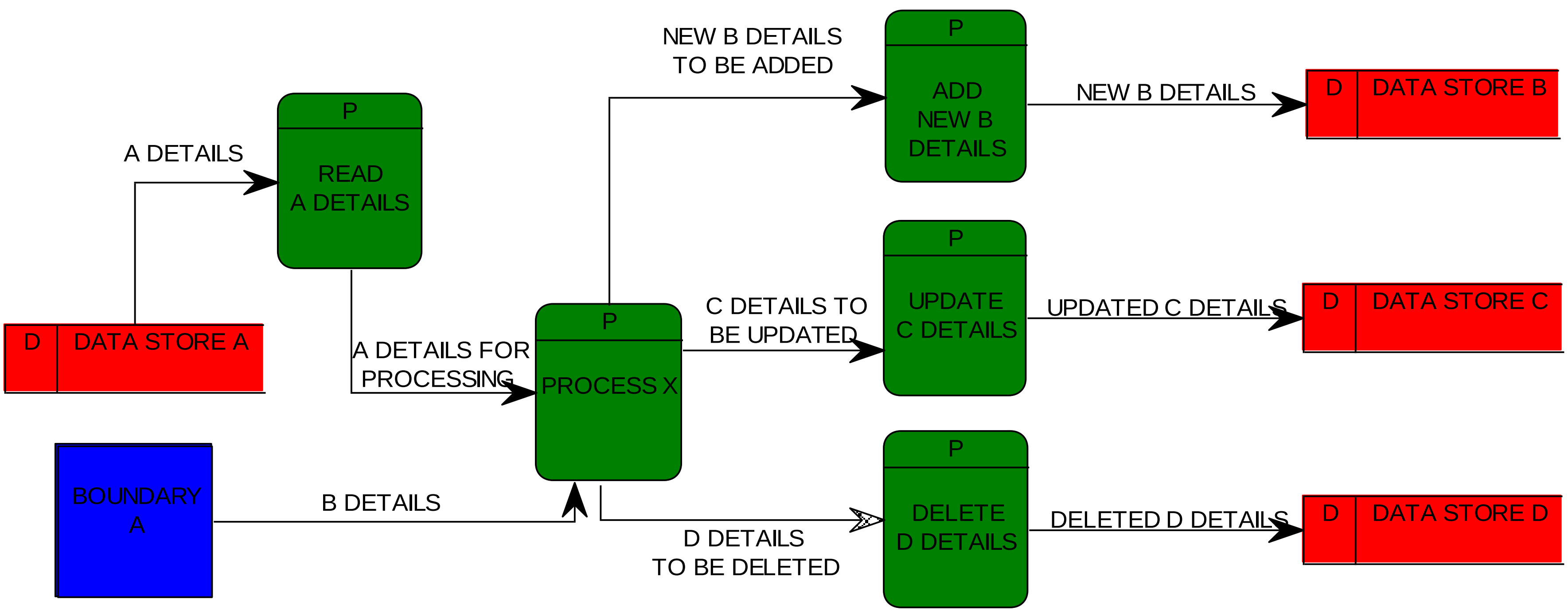
Structured Design

- **Data Flow Diagrams of Programs**
  - Structured design requires that data flow diagrams (DFDs) first be drawn for the program.
  - Processes appearing on the logical, elementary DFDs may represent modules on a structure chart.
  - Logical DFDs need to be revised to show more detail in order to be used by programmers
  - The following revisions may be necessary:
    - Processes appearing on the DFD should do one function.
    - Processes need to be added to handle data access and maintenance.
    - DFDs must be revised to include editing and error handling processes, and processes to implement internal controls.

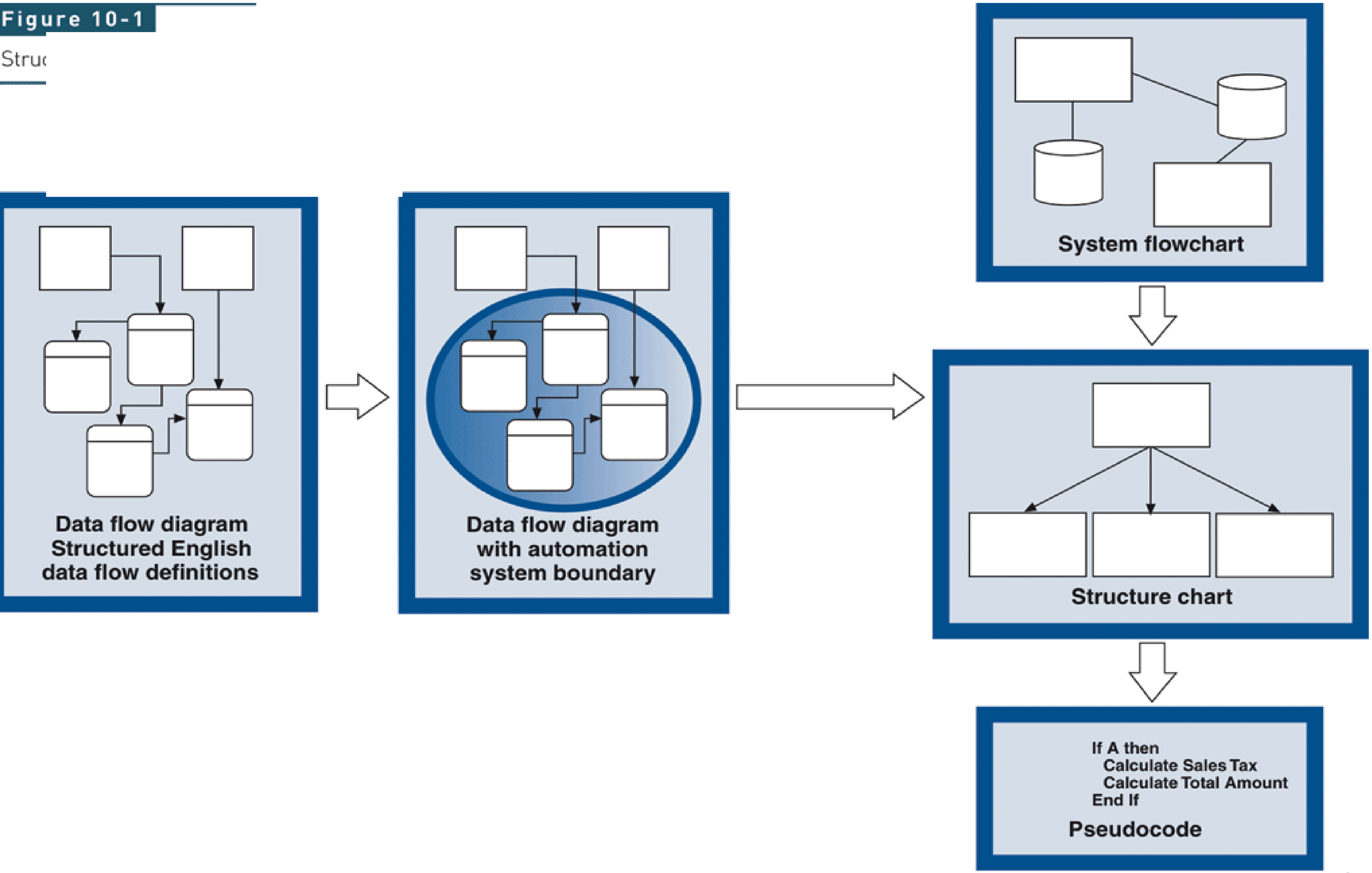

D DATA STORE A
DATA A
BOUNDARY A
DATA C
P PROCESS WITH MANY INPUTS & OUTPUTS
SUM OF DATA A AND DATA C
P PROCESS A (DO NOT EXPAND)
FINAL TOTALS OF DATA A & C
BOUNDARY B
D DATA STORE B
DATA B
SCREENED DATA B
P PROCESS B (DO NOT EXPAND)
RELEASED DATA D
REVISED XYZ STATUS
D DATA STORE C
EXPANDED (OR REPLACED BY)
Expanding a Multifunction Process on a DFD
D DATA STORE A
DATA A
BOUNDARY A
DATA C
P NEW PROCESS A
SUM OF DATA A AND DATA C
P PROCESS A
FINAL TOTALS OF DATA A & C
BOUNDARY B
P NEW PROCESS B
SCREENED DATA B
P PROCESS B
RELEASED DATA D
REVISED XYZ STATUS
D DATA STORE B
DATA B
D DATA STORE C



Adding Data Access and Maintenance Processes to a DF



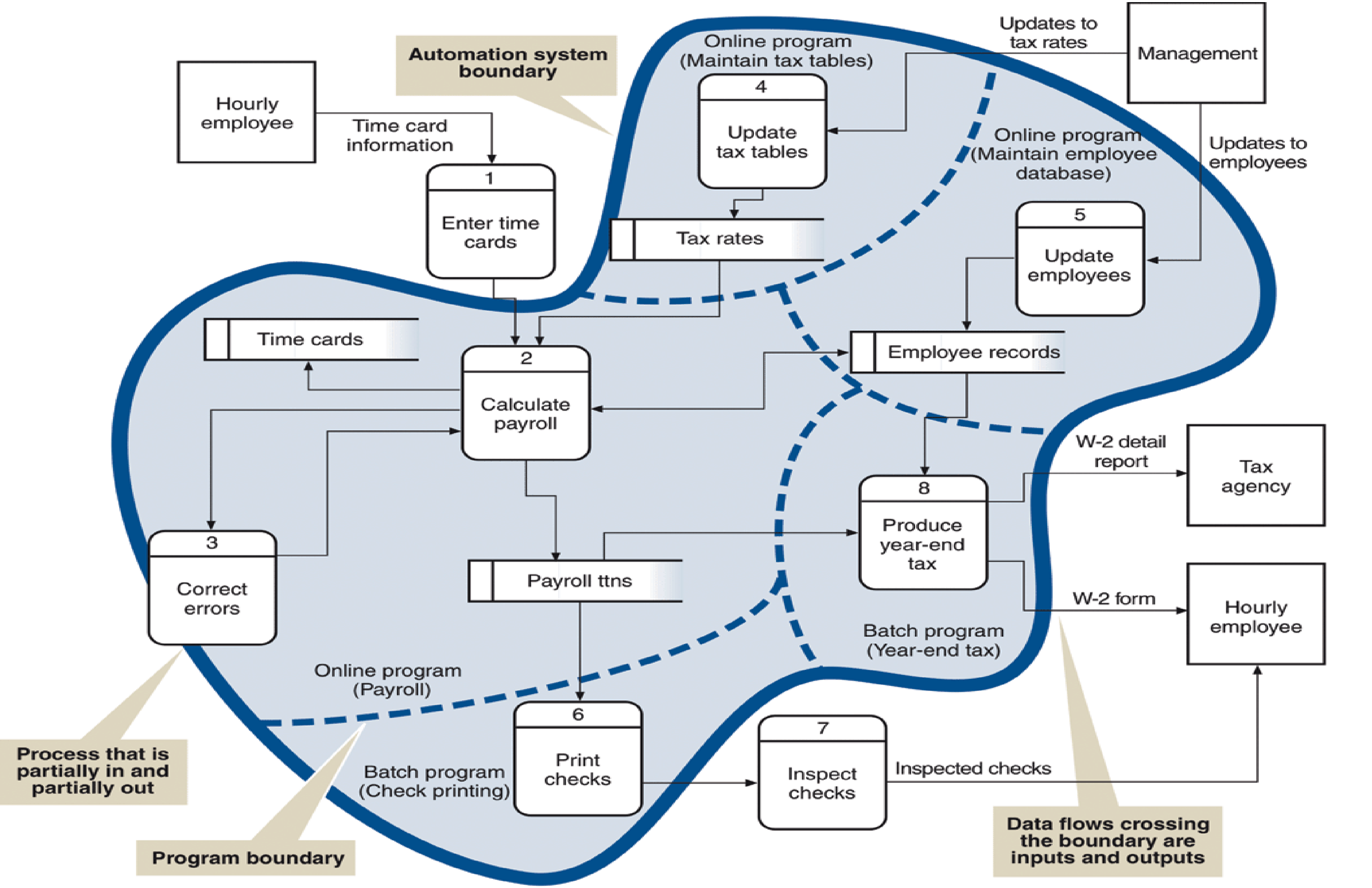
Structured Design Models

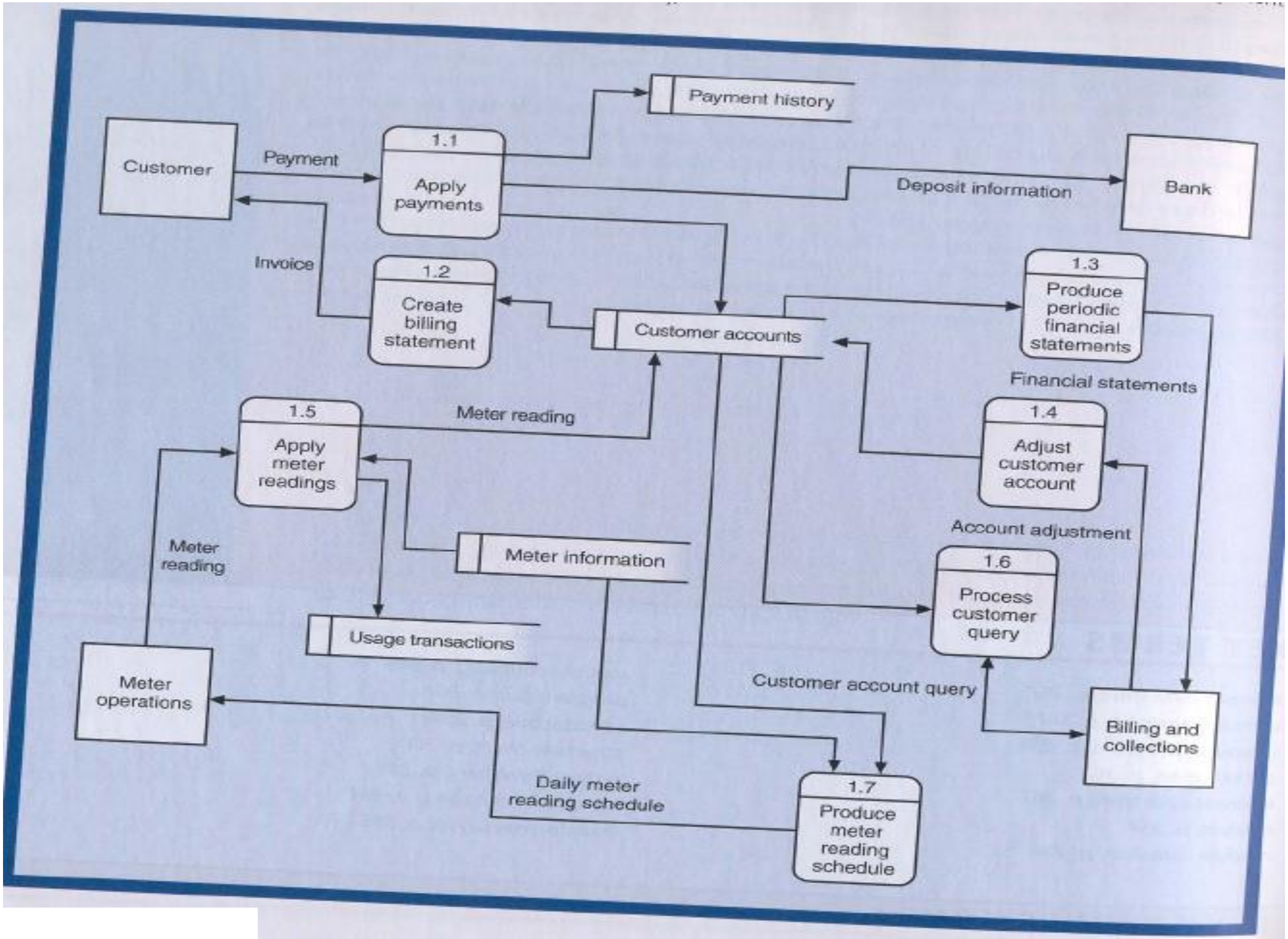


The Automation System Boundary

- Partitions data flow diagram processes into manual processes and automated systems
- Processes can be inside or outside boundary
- Data flows can be inside and outside of boundary
  - Data flows that cross system boundary represent inputs and outputs of system
  - Data flows that cross boundaries between programs represent program-to-program communication

DFD with Automation System Boundary




Payment history
Customer
Payment
1.1
Apply payments
Deposit information
Bank
Invoice
1.2
Create billing statement
Customer accounts
1.3
Produce periodic financial statements
Financial statements
1.5
Apply meter readings
Meter reading
1.4
Adjust customer account
Account adjustment
Meter reading
Meter information
1.6
Process customer query
Usage transactions
Meter operations
Customer account query
Billing and collections
Daily meter reading schedule
1.7
Produce meter reading schedule

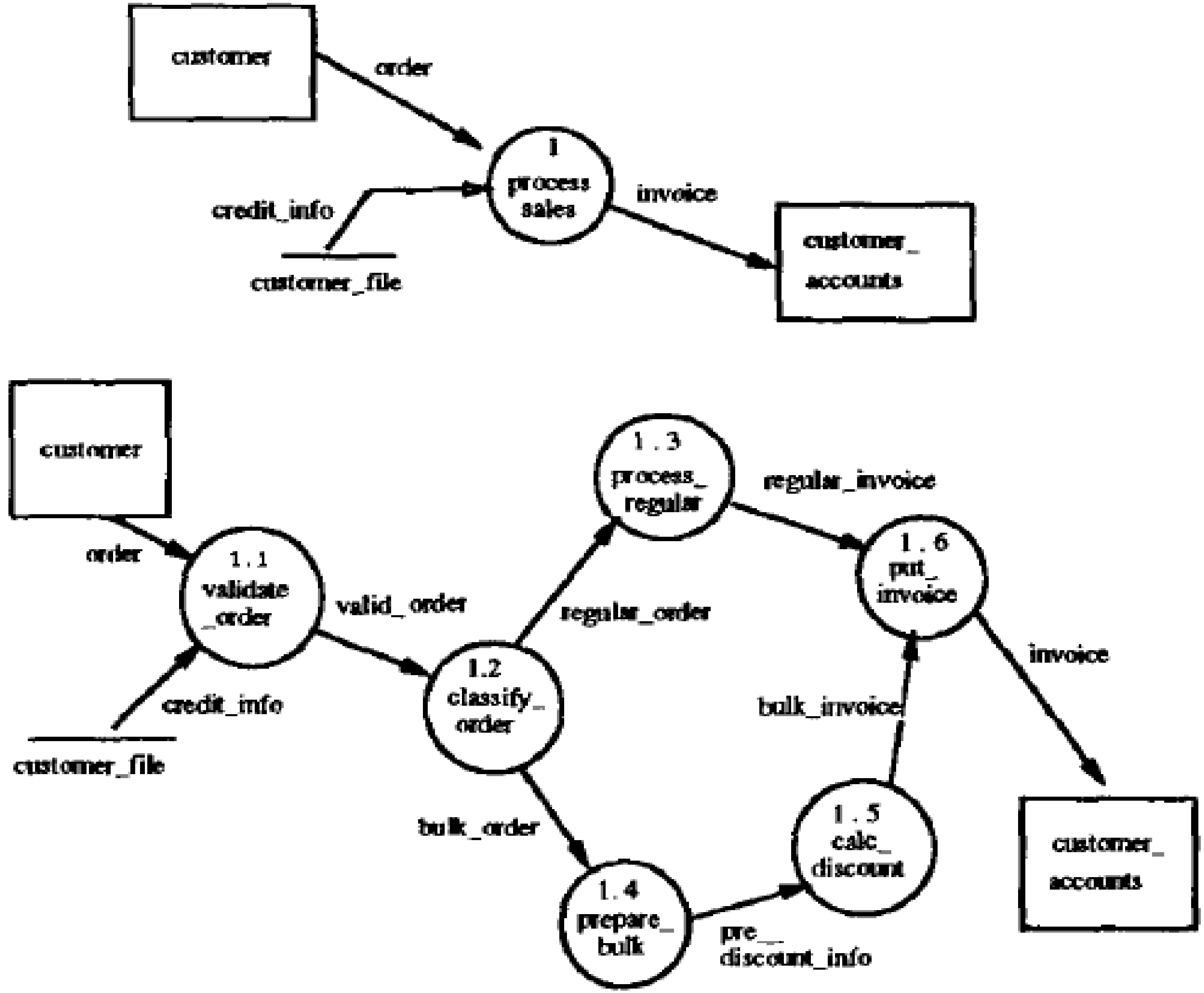


Figure 1: sample data flow diagram

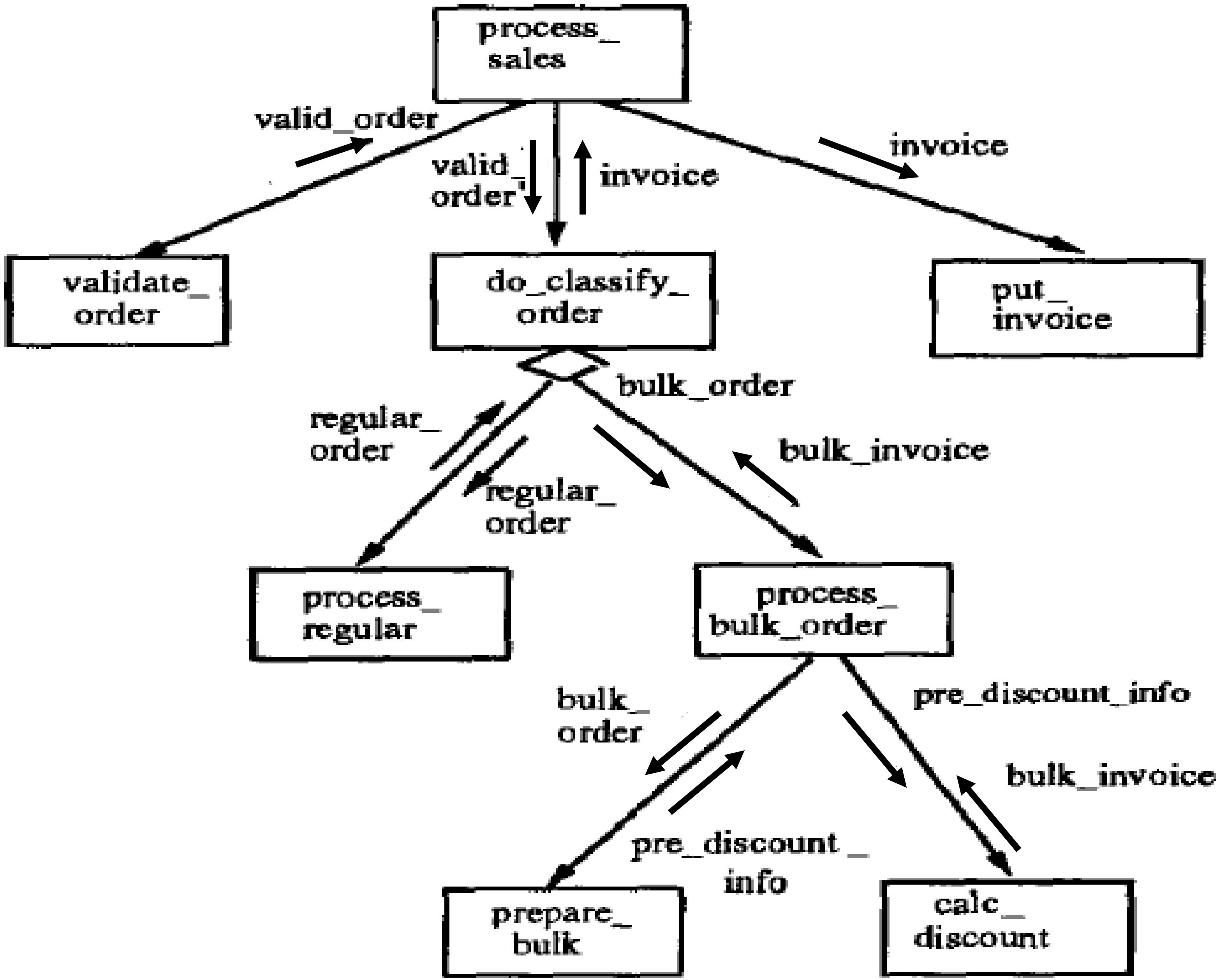


Figure 2: sample structure chart

DFD vs SC

- A data flow diagram is a natural tool for the analysis phase of the software life cycle, whereas a structure chart is useful in the design and implementation phases.
- There is a great difference in graphical outlook between data flow diagrams and structure charts, and the bridging of this gap is vital in ensuring a smooth transition between software development phases.
- To this end, structured methodology provides us with two important strategies called transform analysis and transaction analysis.
- Then the structured chart can be evaluated according to guidelines such as coupling and cohesion.

The Structure Chart

- Describes functions and sub-functions of each part of system
- Shows relationships between modules of a computer program
- Simple and direct organization
  - Each module performs a specific function
  - Each layer in a program performs specific activities
- Chart is tree-like with root module and branches

Structured Design

- **Structure Charts**
  - The primary tool used in structured design is the structure chart.
    - **Structure charts** are used to graphically depict a modular design of a program.
      - Specifically, they show how the program has been partitioned into smaller more manageable modules, the hierarchy and organization of those modules, and the communication interfaces between modules.
      - Structure charts, however, do not show the internal procedures performed by the module or the internal data used by the module.

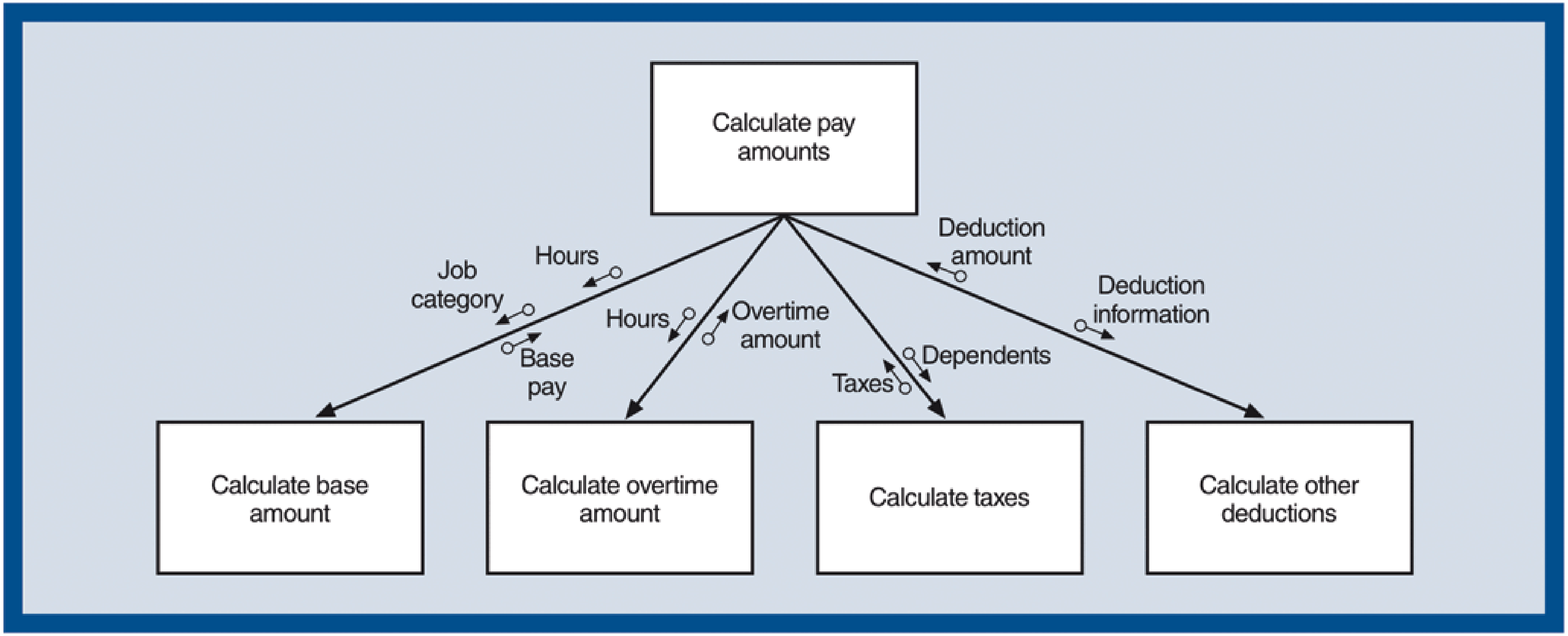
Structured Design

- **Structure Charts**
  - Structure chart modules are depicted by named rectangles.
  - Structure chart modules are presumed to execute in a top-to-bottom, left-to-right sequence.
  - An arc shaped arrow located across a line (representing a module call) means that the module makes iterative calls.
  - A diamond symbol located at the bottom of a module means that the module calls one and only one of the other lower modules that are connected to the diamond.
  - Program modules communicate with each other through passing of data.

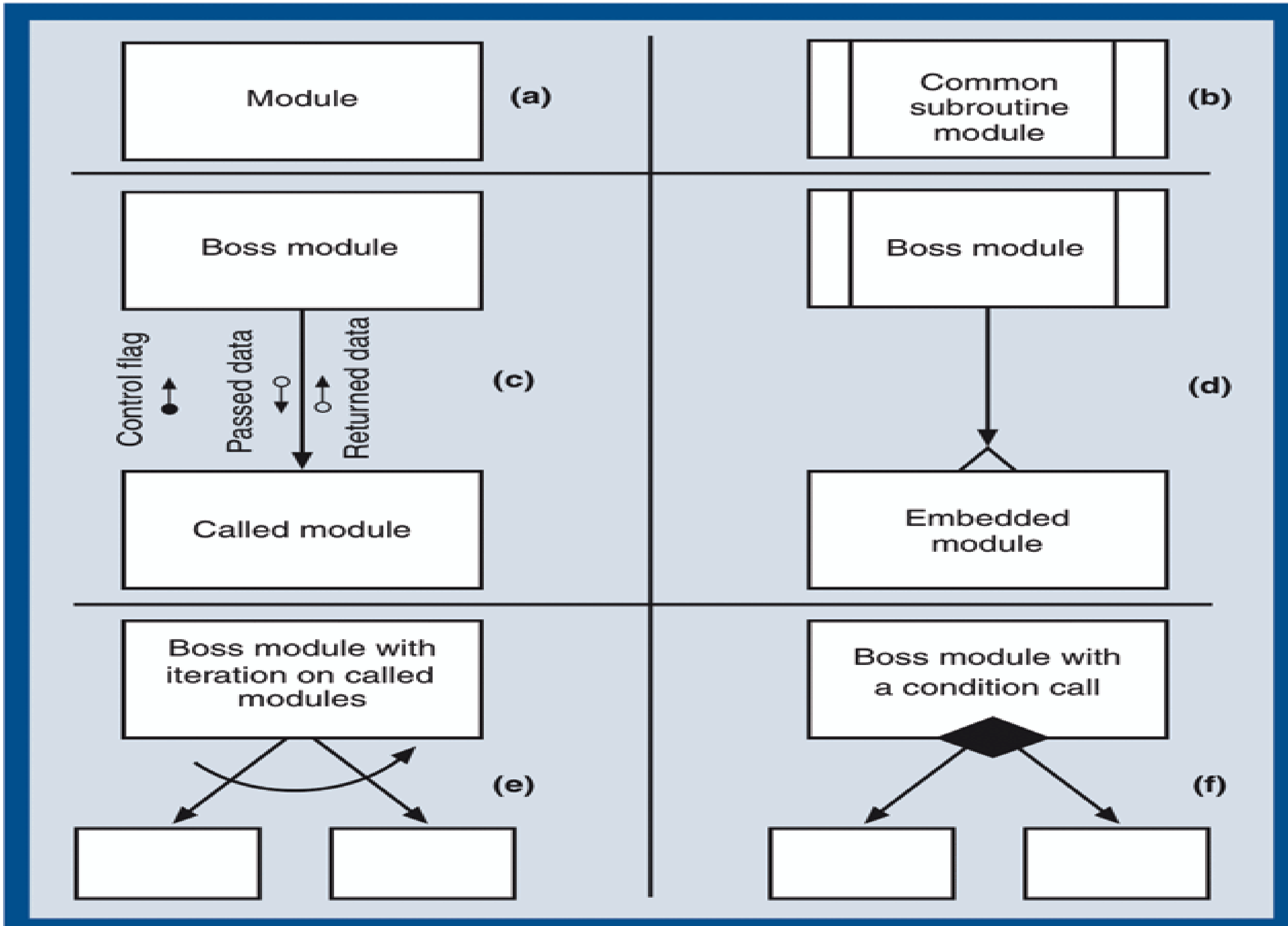
Structured Design

- **Structure Charts**
  - Programs may also communicate with each other through passing of messages or control parameters, called **flags**.
  - Library modules are depicted on a structure chart as a rectangle containing a vertical line on each side.

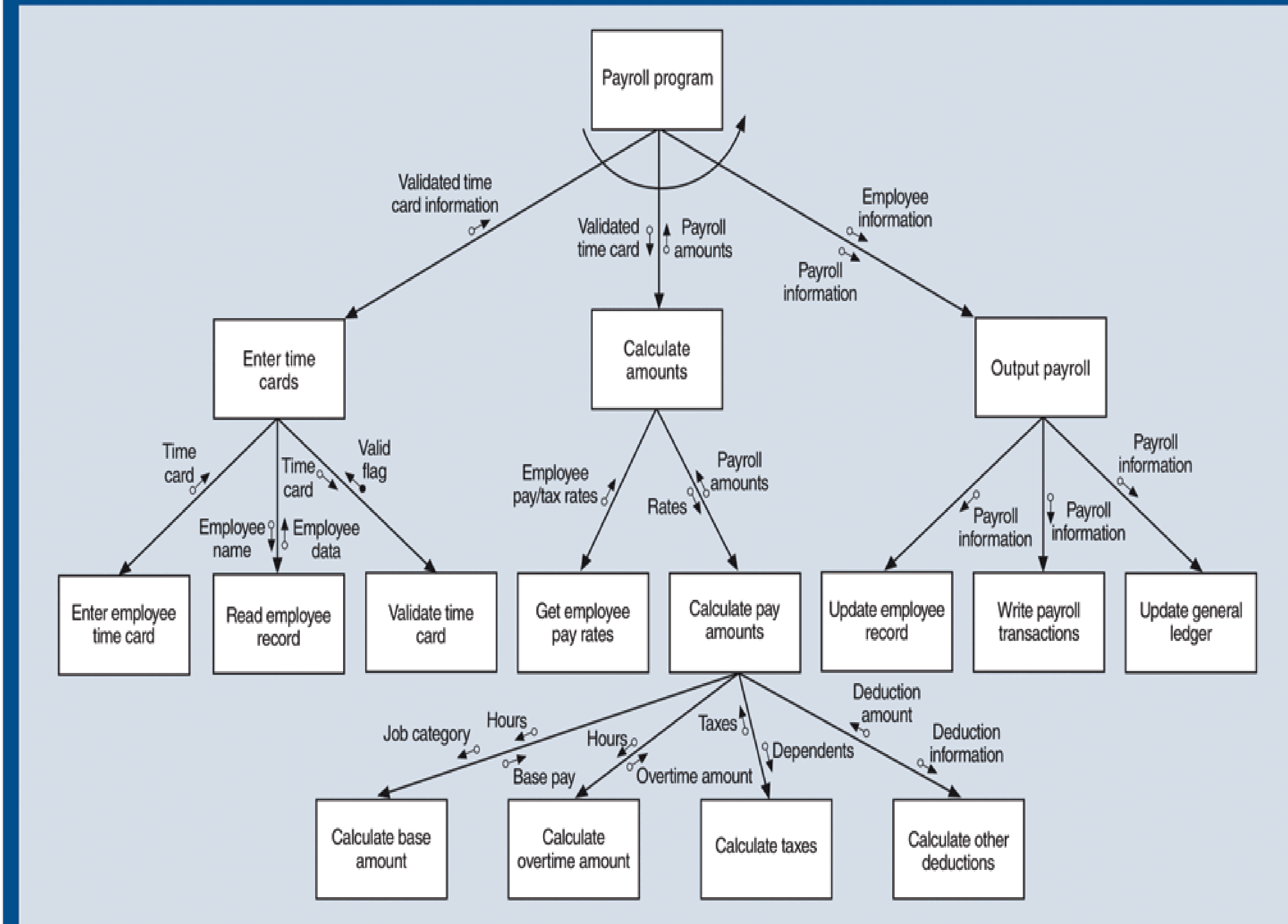
A Simple Structure Chart for the Calculate Pay Amounts Module



Structure Chart Symbols



Structure Chart for Entire Payroll Program



Developing a Structure Chart

- Transaction Analysis
  - The development of a structure chart based on a DFD that describes the processing for several types of transactions.
- Transform Analysis
  - The development of a structure chart based on a DFD that describes the input-process-output

Structured Design

- **Transform Analysis**
  - One approach used to derive a program structure chart from program DFD is transform analysis.
    - **Transform analysis** is an examination of the DFD to divide the processes into those that perform input and editing, those that do processing or data transformation (e.g., calculations), and those that do output.
      - The portion consisting of processes that perform input and editing is called the **afferent**.
      - The portion consisting of processes that do actual processing or transformations of data is called the **central transform**.
      - The portion consisting of processes that do output is called the **efferent**.

- Transform and transaction analyses are two supplementary procedures for producing structure charts.
- In transform analysis, we follow the input and output data streams of a data flow diagram to determine the central portion of the system responsible for the main transform of data.
- In this way, a balanced structured chart can be derived accordingly.
- In transaction analysis, we try to isolate a transaction center which captures an input transaction, determines its type, and then processes it in the appropriate branch of the center. Figure 3 and Figure 5 are respectively a transform and a transaction data flow diagram.

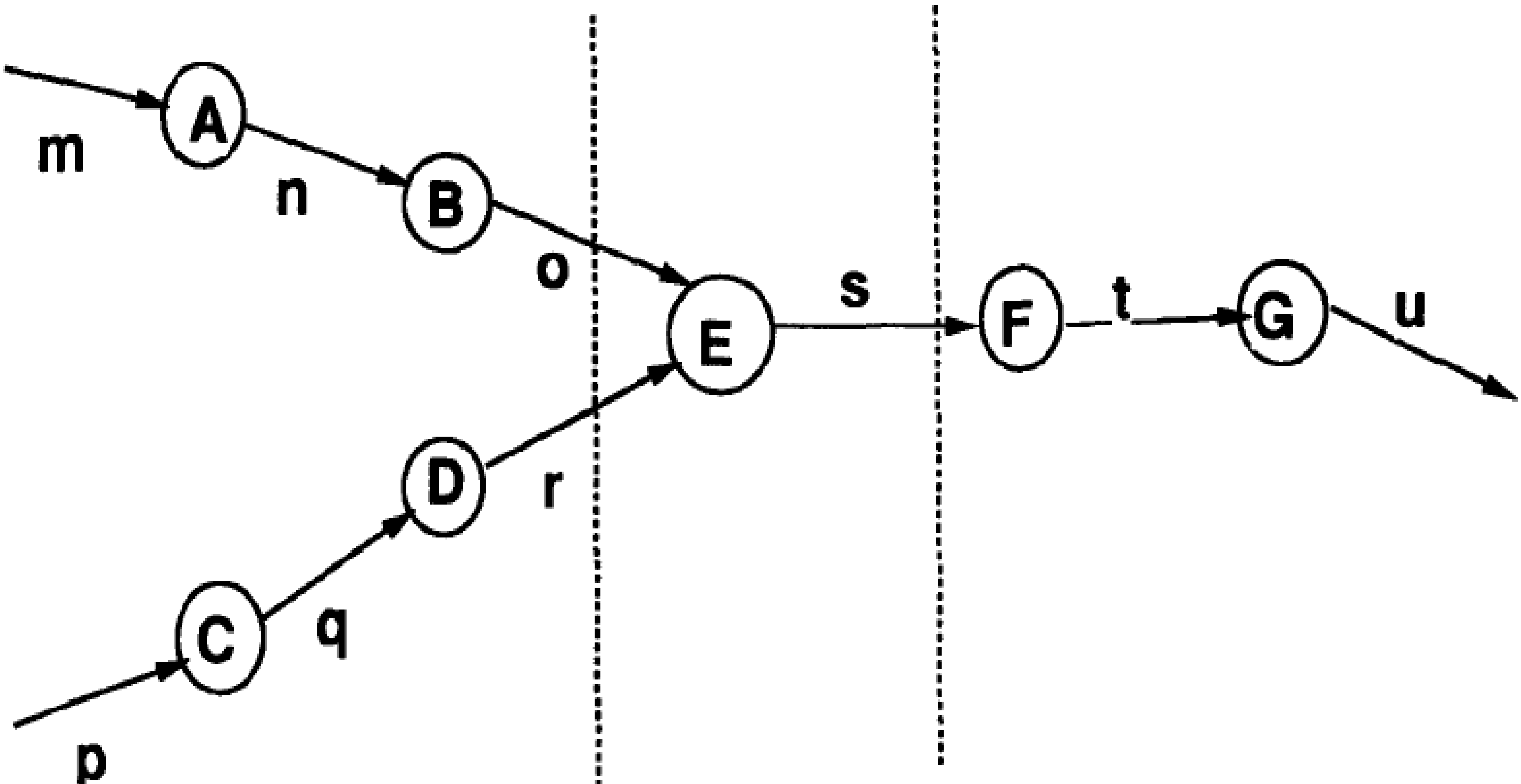


Figure 3 transform data flow diagram

Transform analysis

- In transform analysis, a data flow diagram contains a *transform center*, an *afferent streams* and an *efferent streams* (Figure 3).
- The transform center is the collection of processes which make up the major function of the system.
- An afferent stream is a string of processes which start off by reading data from a physical source, and then convert it into a more abstract form suitable for the transform center.
- An efferent stream, on the other hand, is a string of processes which convert output data horn the transform center into a more physical form suitable for output to the real world.

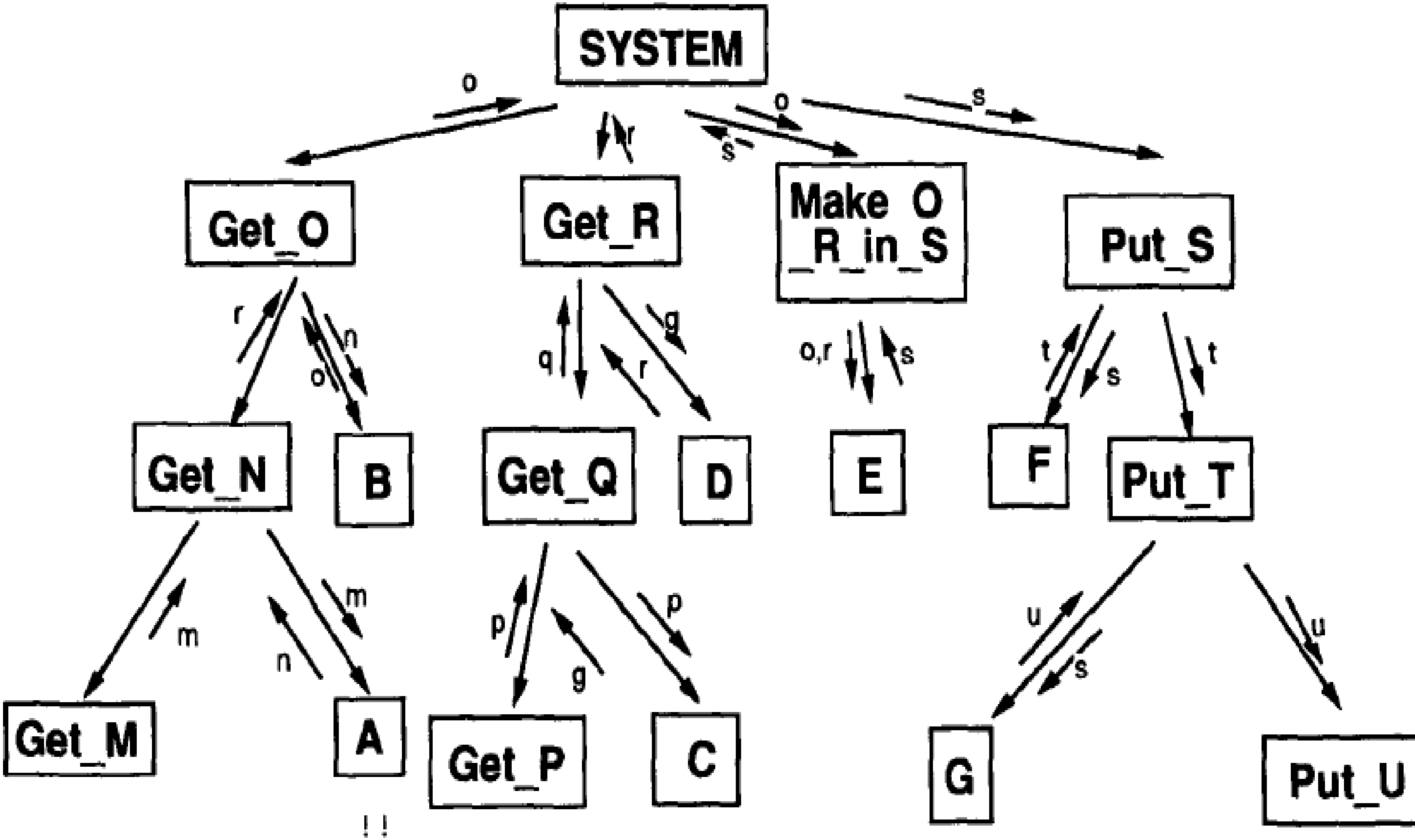
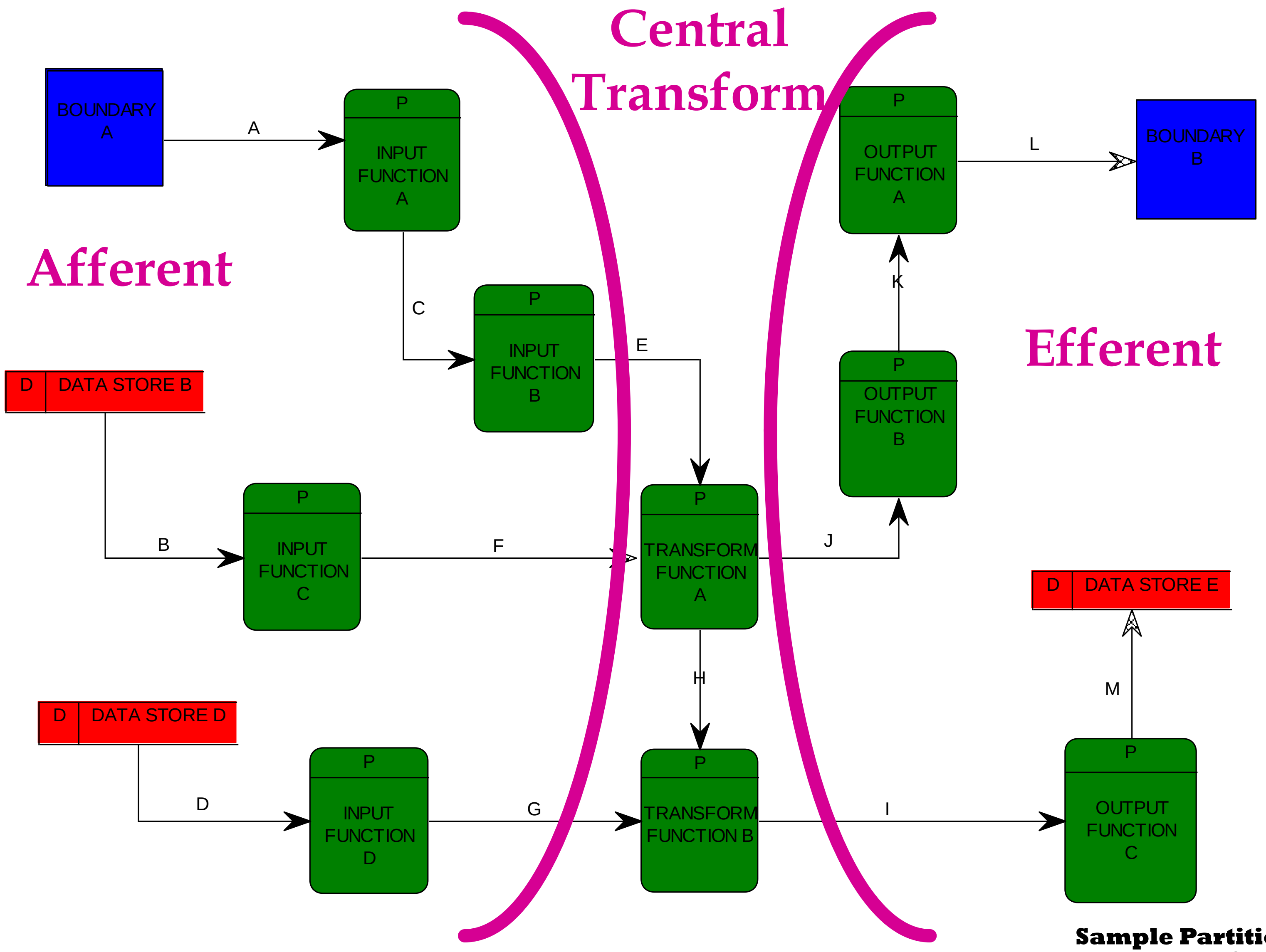


Figure 4: derived structure chart from Figure 3



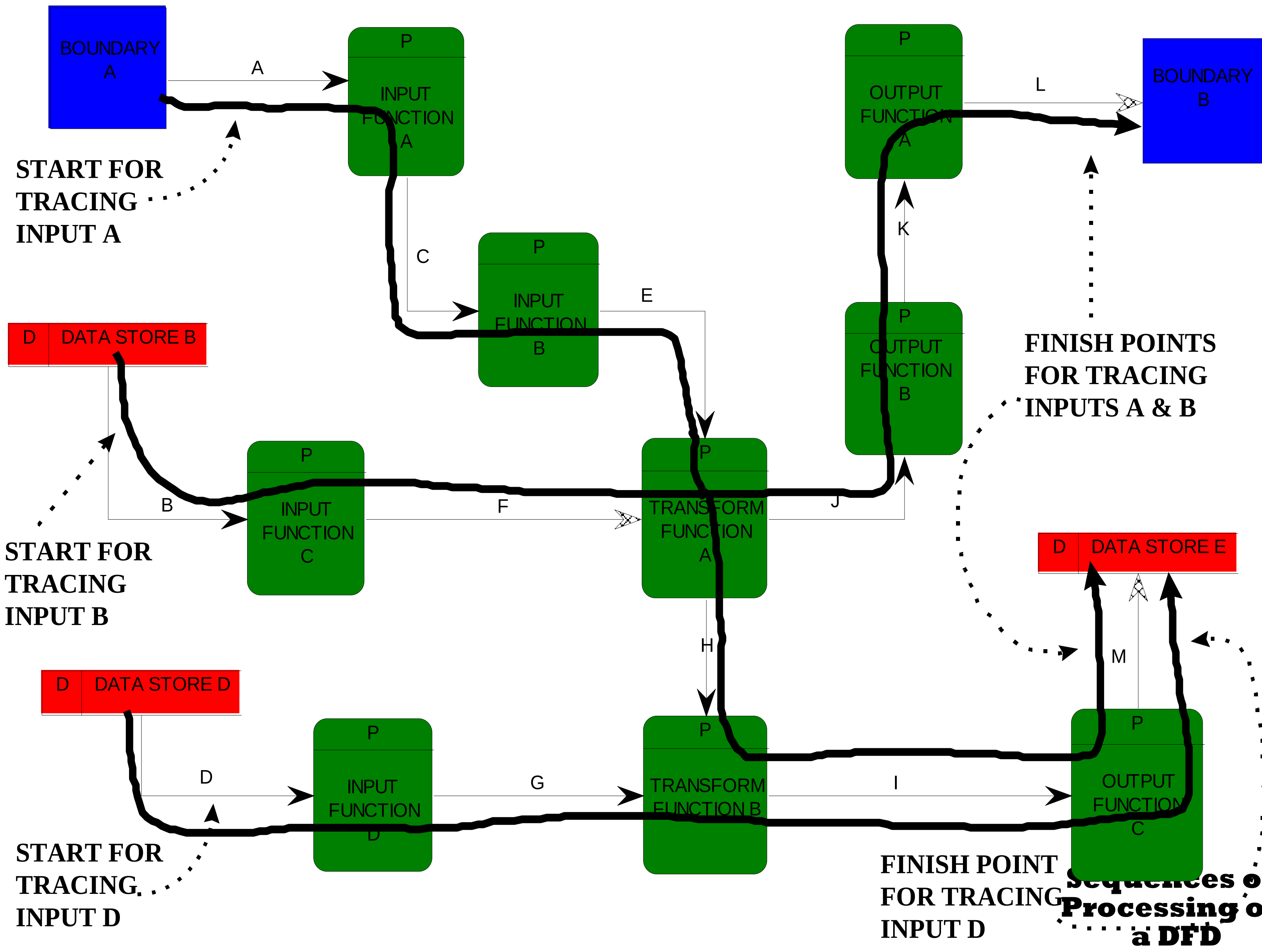
Sample Partitioned DFD during Transform Analysis

Structured Design

- **Transform Analysis**
  - The strategy for identifying the afferent, central transform, and efferent portions of a begins by first tracing the sequence of processing for each input.
    - There may be several sequences of processing.
    - A sequence of processing for a given input may actually split to follow different paths.

Structured Design

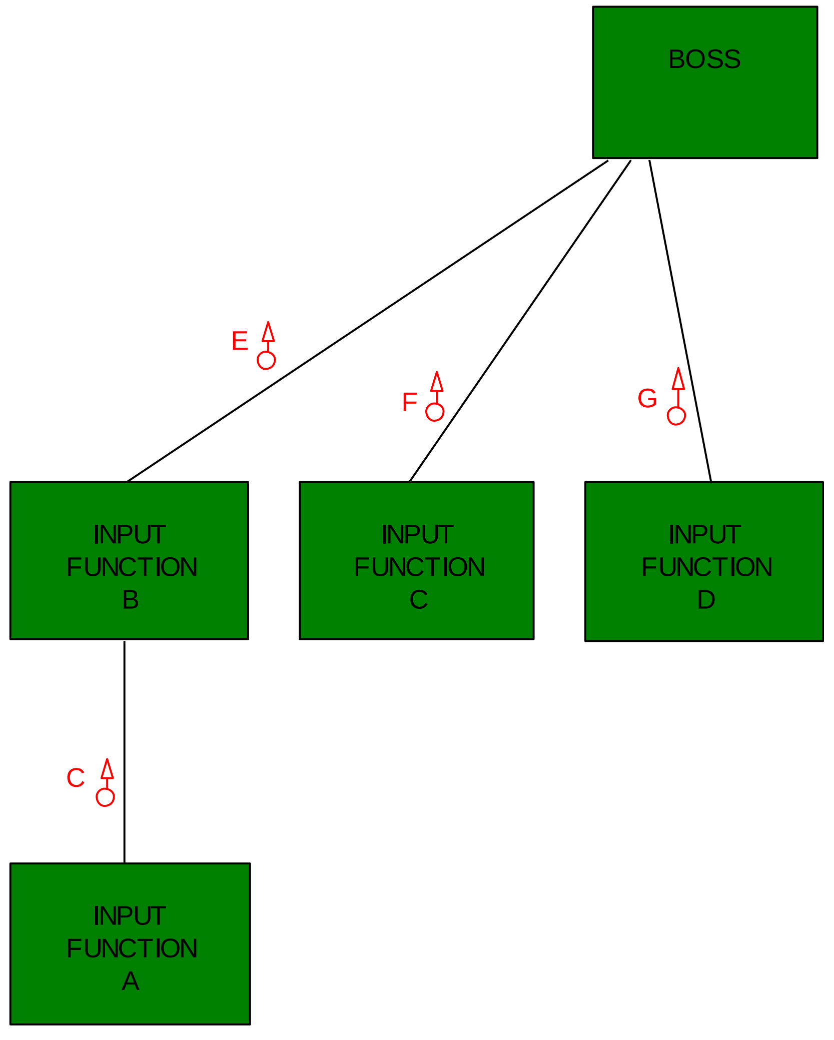
- **Transform Analysis**
  - Once sequence paths have been identified, each sequence path is examined to identify process along that path that are afferent processes.
    - The steps are as follows:
      - **Step 1** - Beginning with the input data flow, the data flow is traced through the sequence until it reaches a process that does processing (transformation of data) or an output function.
      - **Step 2 -** Beginning with an output data flow from a path, the data flow is traced backwards through connected processes until a transformation processes is reached (or a data flow is encountered that first represents output).
      - **Step 3 -** All other processes are then considered to be part of the central transform!



Sequences of Processing on a DFD

Structured Design

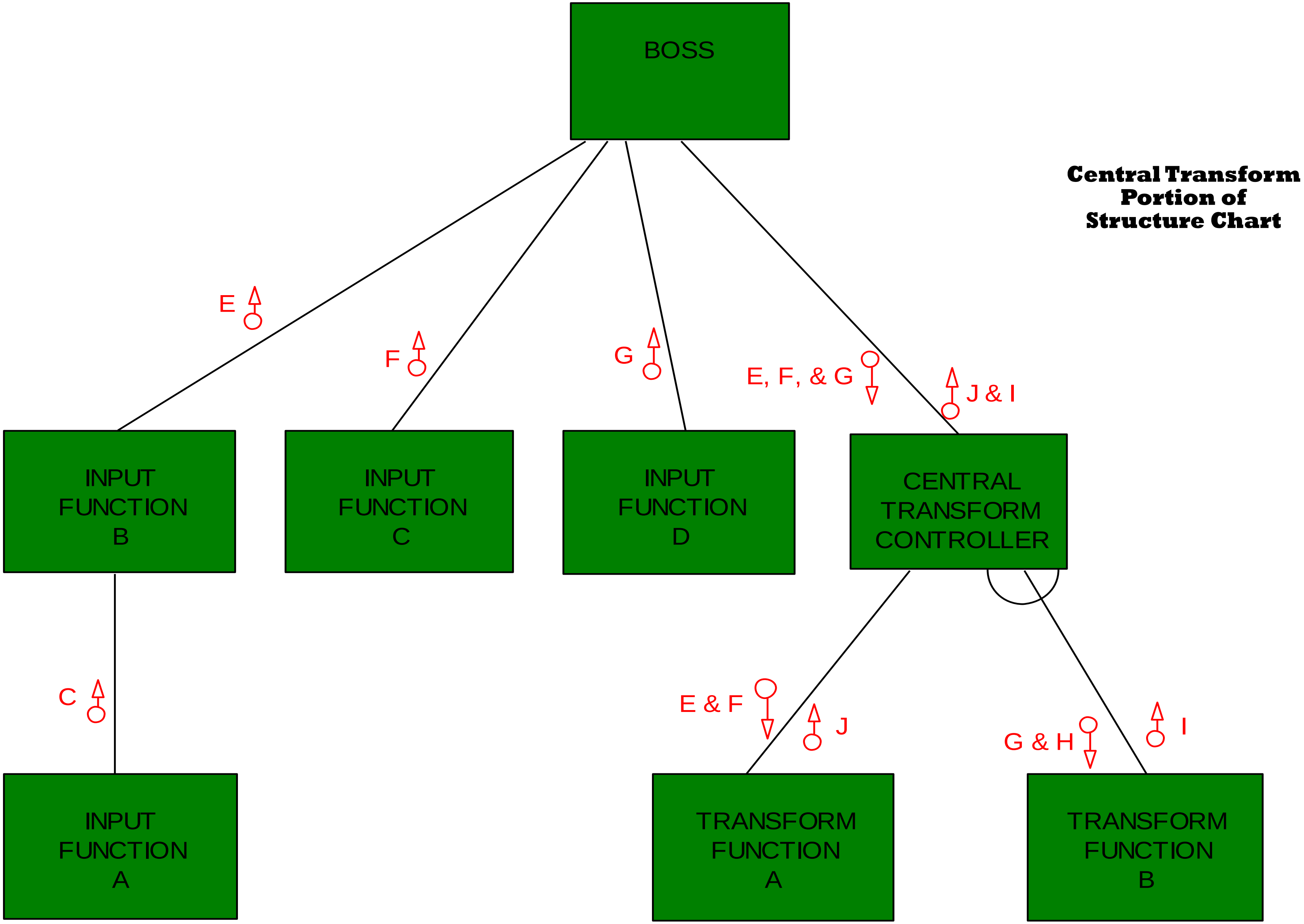
- **Transform Analysis**
  - Once the DFD has been partitioned, a structure chart can be created that communicates the modular design of the program.
    - **Step 1** - Create a process that will serve as a “commander and chief” of all other modules.
      - This module manages or coordinates the execution of the other program modules.
    - **Step 2** - The last process encountered in a path that identifies afferent processes becomes a second-level module on the structure charts.
    - **Step 3** - Beneath that module should be a module that corresponds to its preceding process on the DFD.
    - This would continue until all afferent processes in the sequence path are included on the structure chart.



Afferent Portion of Structure Chart

Structured Design

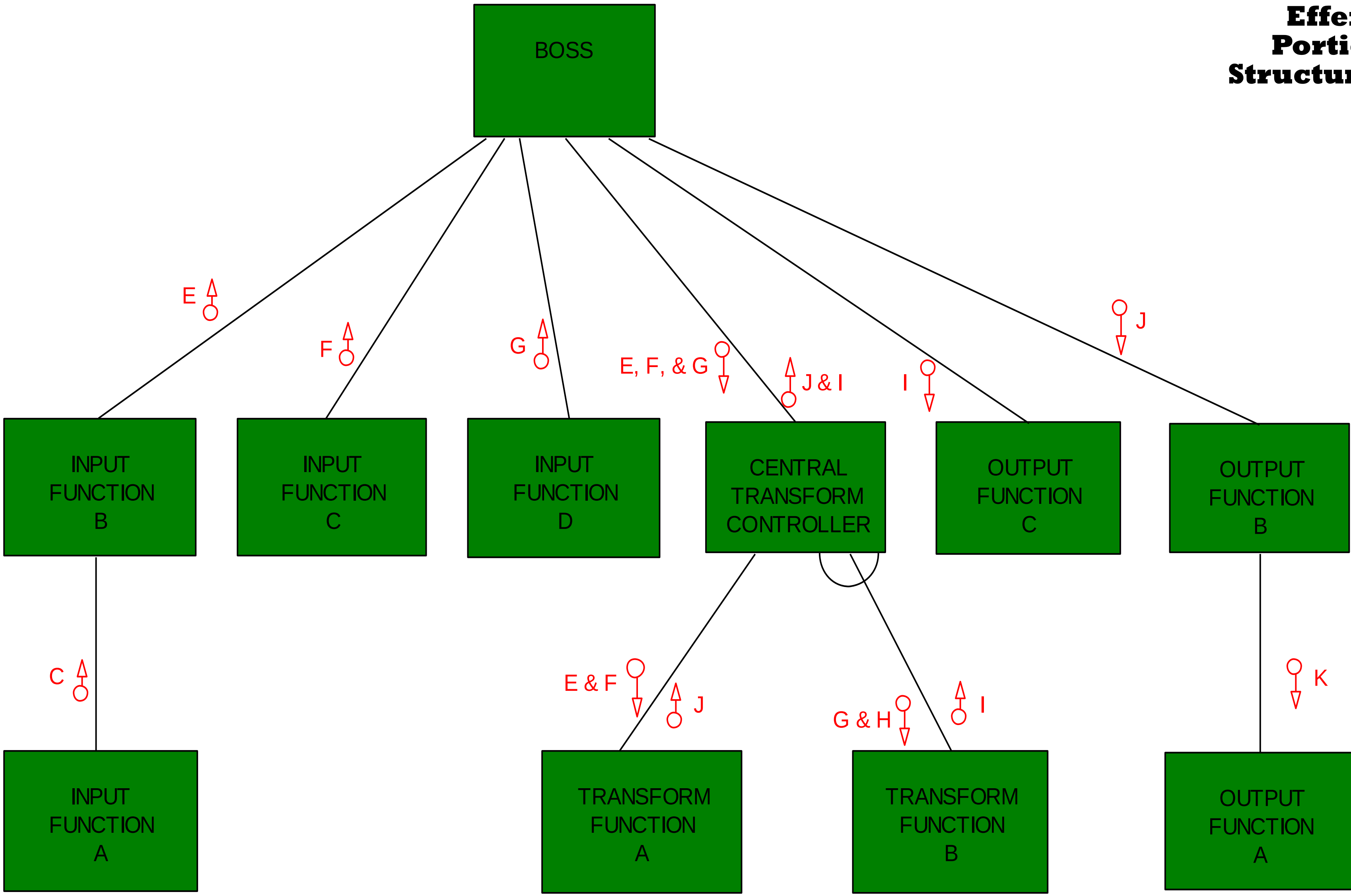
- **Transform Analysis**
  - **Step 4 -** If there is only one transformation process, it should appear as a single module directly beneath the boss module.
    - Otherwise, a coordinating module for the transformation processes should be created and located directly above the transformation process..
  - **Step 5 -** A module per transformation process on the DFD should be located directly beneath the controller module.

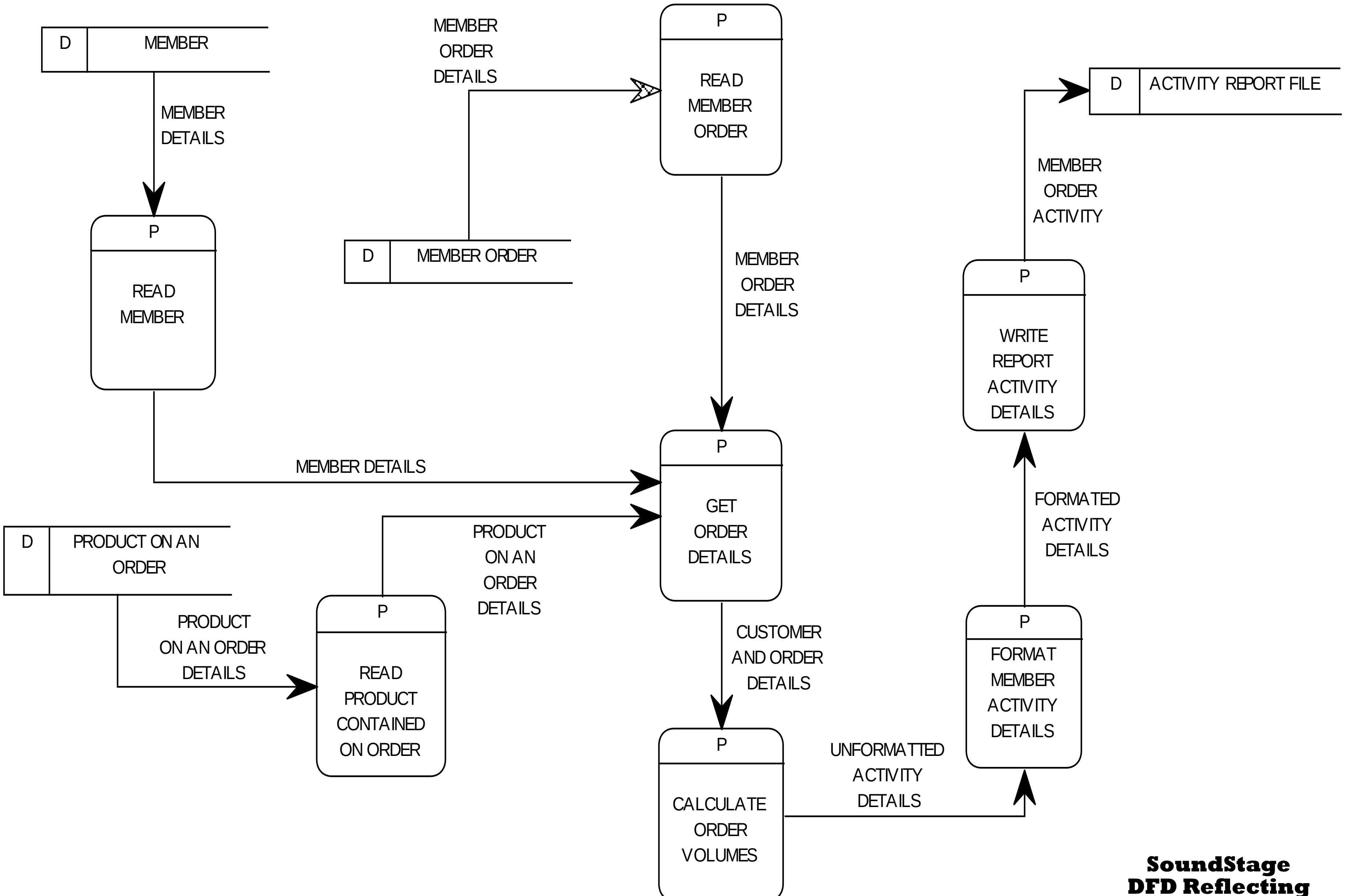

BOSS
Central Transform
Portion of
Structure Chart
E
F
G
E, F, & G
J & I
INPUT FUNCTION B
INPUT FUNCTION C
INPUT FUNCTION D
CENTRAL TRANSFORM CONTROLLER
C
E & F
J
G & H
I
INPUT FUNCTION A
TRANSFORM FUNCTION A
TRANSFORM FUNCTION B

Structured Design

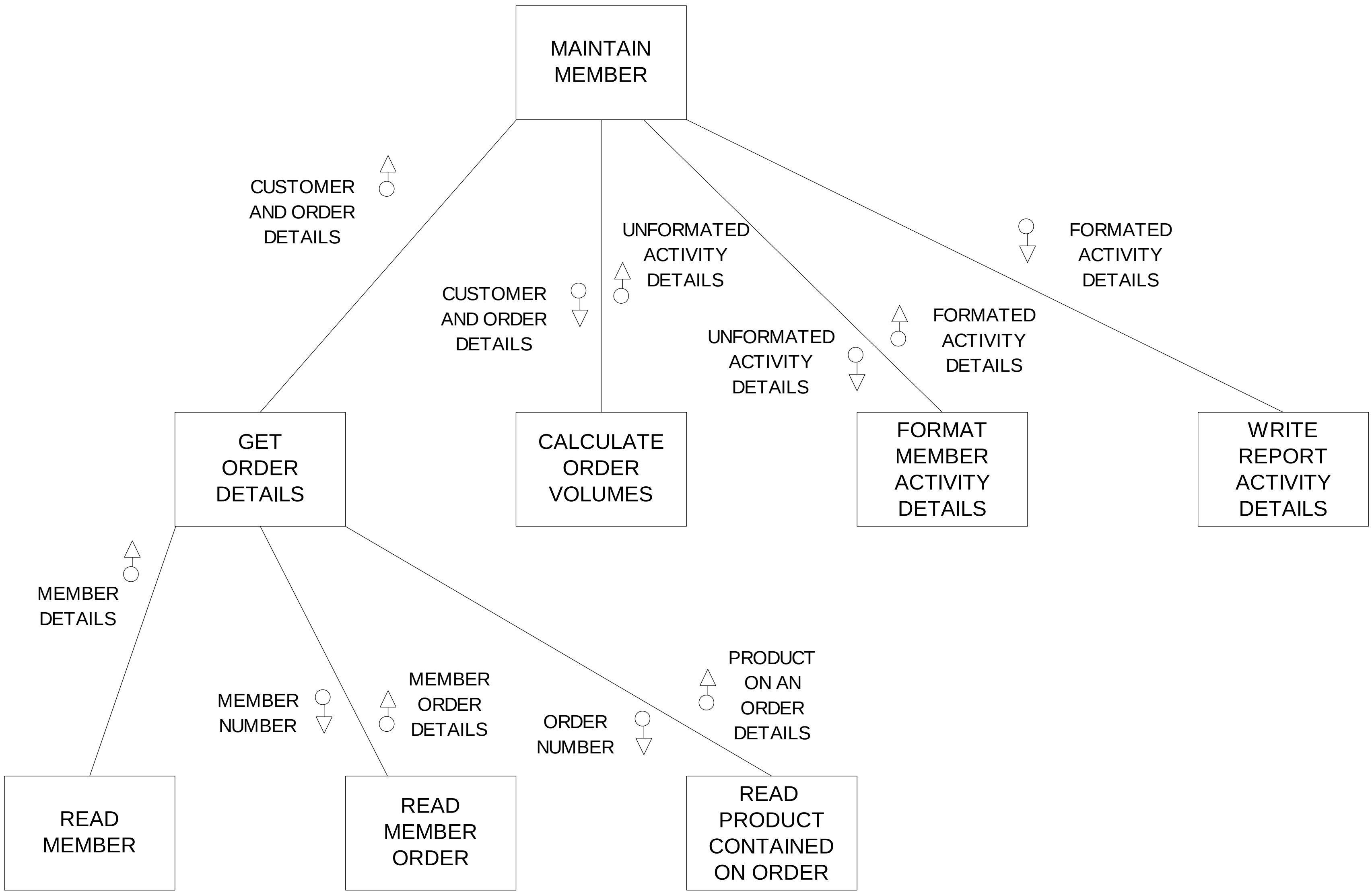
- **Transform Analysis**
  - **Step 6 -** The last process encountered in a path that identifies efferent processes becomes a second-level module on the structure chart.
  - **Step 7 -** Beneath the module (in step 6) should be a module that corresponds to the succeeding process appearing on the sequence path.
    - Likewise any process immediately following that process would appear as a module beneath it on the structure chart.

Efferent Portion of Structure Char





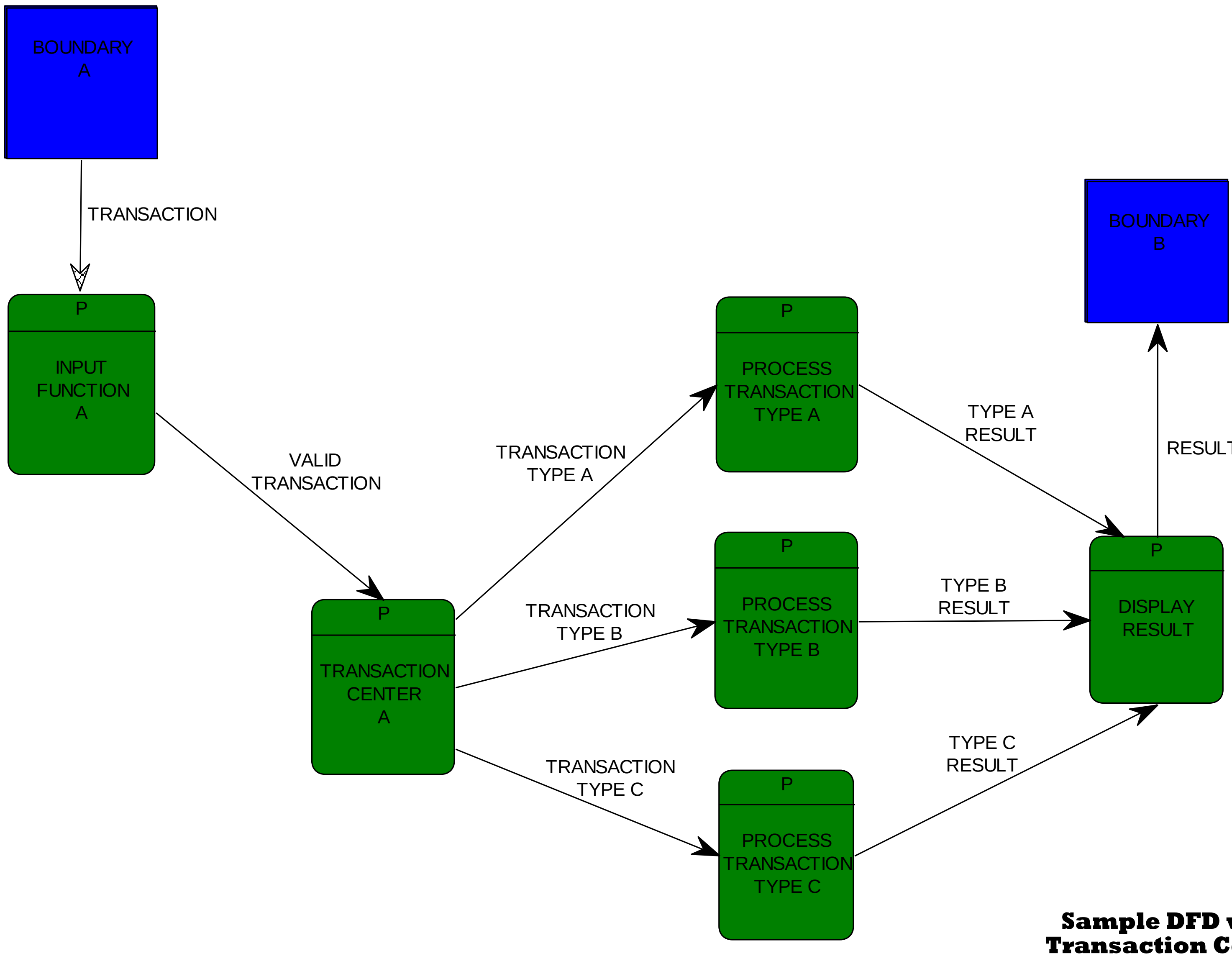
SoundStage DFD Reflecting Central Transform



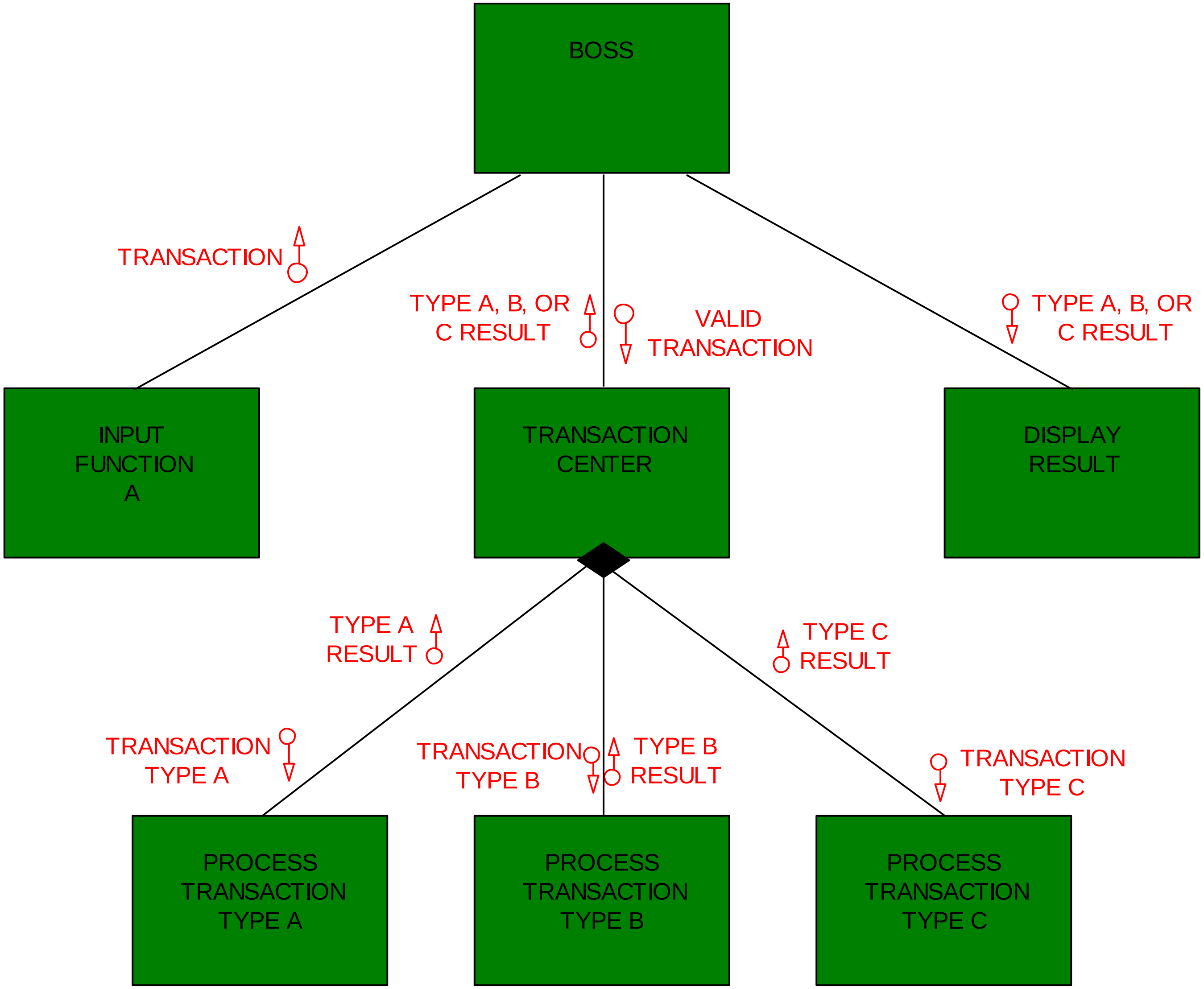
SoundStage Structure Chart from Transform Analysis

Structured Design

- **Transaction Analysis**
  - An alternative structured design strategy for developing structure charts is called transaction analysis.
    - **Transaction analysis** is the examination of the DFD to identify processes that represent transaction centers.
      - A **transaction center** is a process that does not do actual transformation upon the incoming data (data flow); rather, it serves to route the data to two or more processes.
        - » You can think of a transaction center as a traffic cop that directs traffic flow.
        - » Such processes are usually easy to recognize on a DFD, because they usually appear as a process containing a single incoming data flow to two or more other processes.



Sample DFD with Transaction Center



Sample Structure Chart with Transaction Center

Structured Design

- **Transaction Analysis**
  - The primary difference between transaction analysis and transform analysis is that transaction analysis recognizes that modules can be organized around the transaction center rather than a transform center.

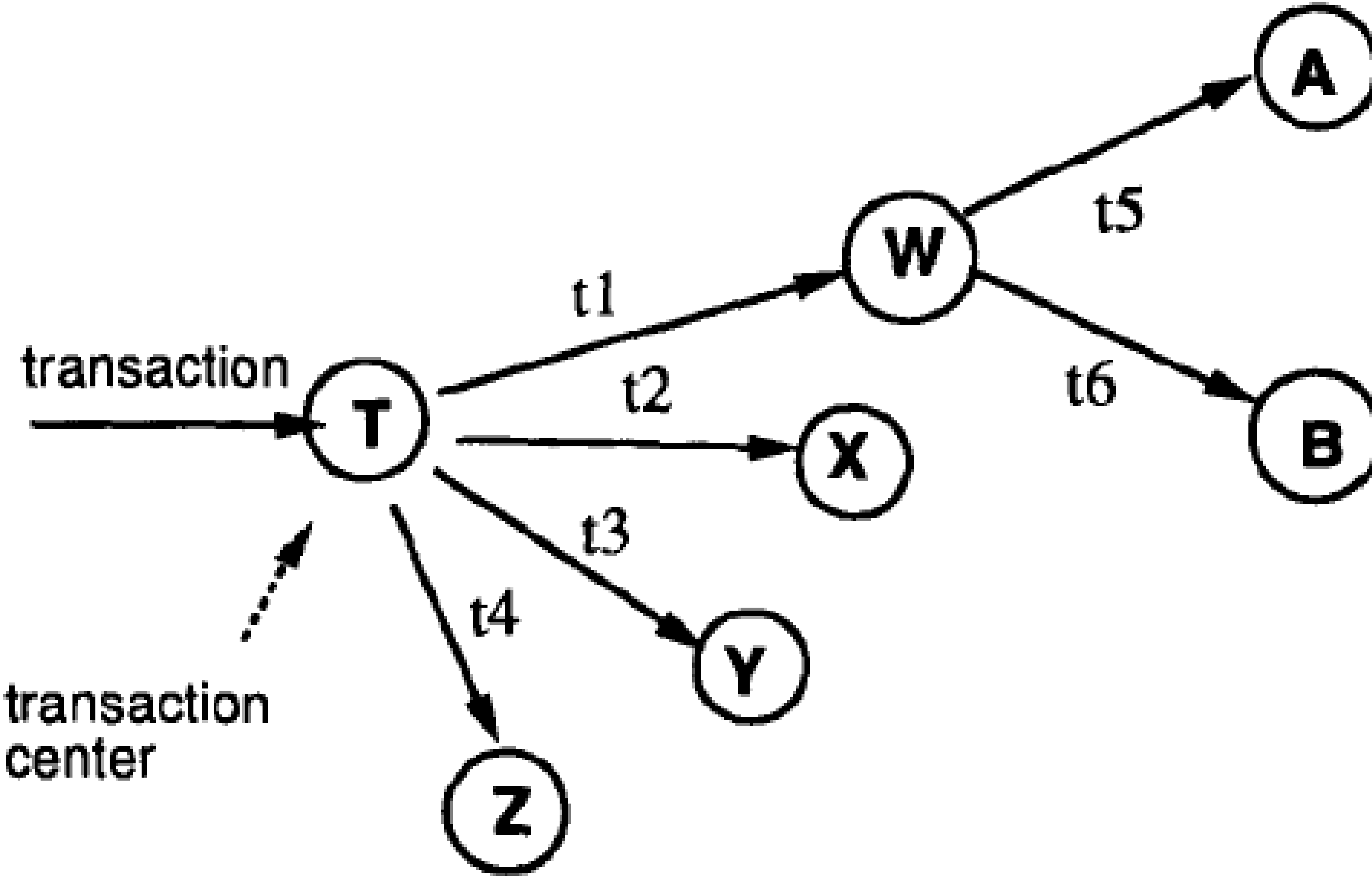


Figure 5 transaction data flow diagram

Transaction Analysis

- In transaction analysis, there is a transaction center which captures an input transaction, determines its type, and then processes it in the appropriate branch of the center (Figure 5). Transaction analysis consists of three steps:
  - 1. Find each transaction center by locating the respective first node and all its subsequent branches;
  - 2. Reduce each of them into a single node for the ease of transform analysis;
  - 3. Re-expand the nodes afterwards.

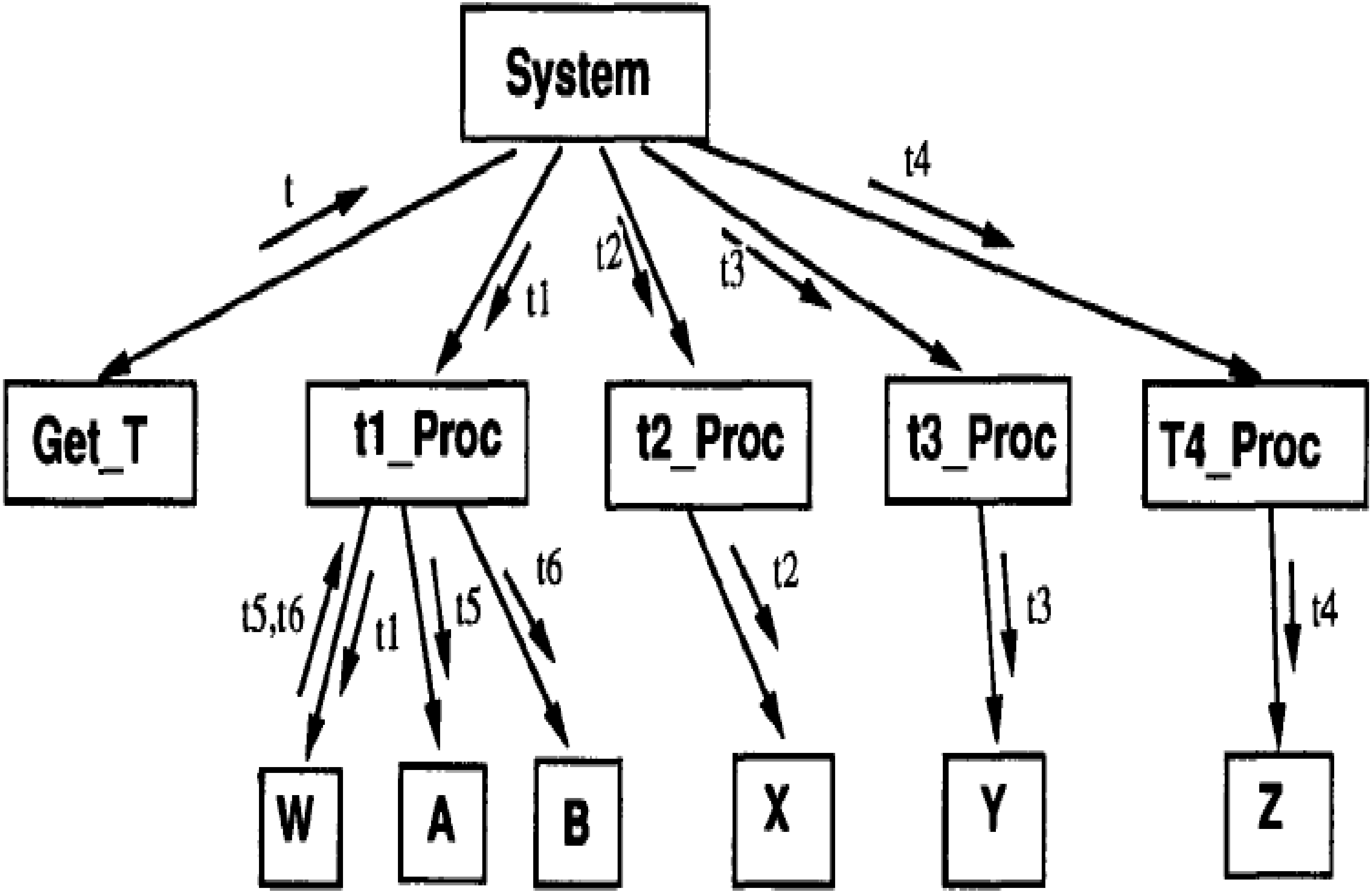


Figure 6 derived structure chart from Figure 5

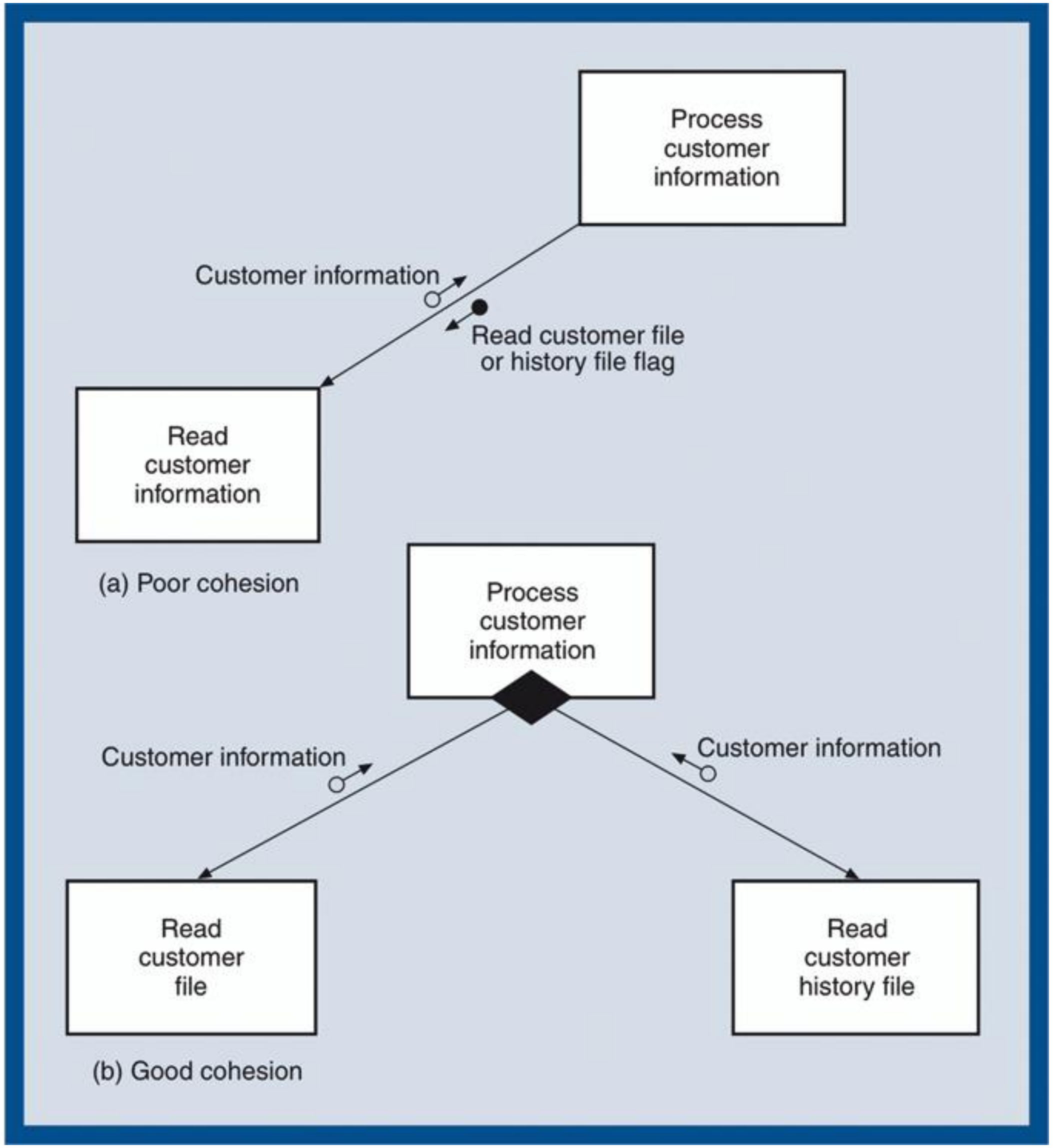
- Actually, it is possible that a system is constructed by both of transform and transaction analyses.
- By using decomposition and level techniques of DFD, the transaction centers are hidden as if they are single processes, and applying transform analysis first, then using transaction analysis to deal with the hidden parts.

Evaluating the Quality of a Structure Chart

- Module coupling
  - Measure of how module is connected to other modules in program
  - Goal is to be loosely coupled

- Module cohesion
  - Measure of internal strength of module
  - Module performs one defined task
  - Goal is to be highly cohesive

Examples of Module Cohesion



Module Algorithm Design—Pseudocode

- Describes internal logic of software modules
- Variation of structured English that is closer to programming code
- Syntax should mirror development language
- Three types of control statements used in structured programming
  - Sequence – sequence of executable statements
  - Decision – if-then-else logic
  - Iteration – do-until or do-while

```
Payroll program
DoUntil No more time cards
     Call Enter time cards
     Call Calculate amounts
     Call Output payroll
End Until

Calculate amounts
     Call Get employee pay rates
     Call Calculate pay amounts

Calculate pay amounts

Call Calculate base amount
If (HoursWorked > 40) Then
     Call Calculate overtime amount
End If
Call Calculate taxes
If (SavingsDeduction=yes) or (MedicalDeduction=yes) or (UnitedWay=yes) Then
     Call Calculate other deductions
End if

Calculate taxes

Get Tax Rates based on Number Dependents, Payrate
Calculate Income Tax = PeriodPayAmount * IncomeTaxRate
If YTD Pay < FICA MaximumAmount Then
     Calculate EmpFICA = PeriodBasePay * FICAEmployeeRate
     Calculate CorpFICA = PeriodBasePay * FICACorpRate
End If
If StateTaxRequired Then
     Get StateTaxRate based on State, NumberDependents, Payrate
     Calculate StateTax = PeriodPayAmount * StateTaxRate
End If
If OvertimePay > 0 Then
     Calculate OvertimeIncomeTax = PeriodOvertimePay * IncomeTaxRate
     Add OvertimeIncomeTax to Incometax
     If YTDPay < FICAMaximum Amount Then
            Calculate EmpOvertimeFICA = PeriodOvertimePay * FICAEmployeeRate
            Calculate CorpOvertimeFICA = PeriodOvertimePay * FICACorpRate
     End If
     If StateTaxRequired Then
            Calculate StateOvertimeTax = PeriodOvertimePay * StateTaxRate
     End If
End If
```

Integrating Structured Application Design with Other Design Tasks

- Structure chart must be modified or enhanced to integrate design of user interface and database
  - Are additional modules needed?
  - Does pseudocode in modules need modification?
  - Are additional data couples needed to pass data?
- Structure charts and system flowcharts must correspond to planned network architecture
  - Required protocols, capacity, and security

Summary

- For traditional structured approach to systems design, primary input is data flow diagram
  - DFD is enhanced by adding system boundary
  - Designer describes processes within each DFD boundary using one or more structure charts
- Structure charts developed using
  - Transaction analysis – multiple transaction types
  - Transform analysis – single transaction from input to output

Summary (continued)

- Structure charts may be based on three-layer architecture
  - Modules will be clearly identified by layer
  - Structure chart may be decomposed if layers execute on multiple systems
- Structured design may also include
  - System flowcharts to show data movement
  - Module pseudocode to describe internal logic of structure chart module

Case study

TV Survey

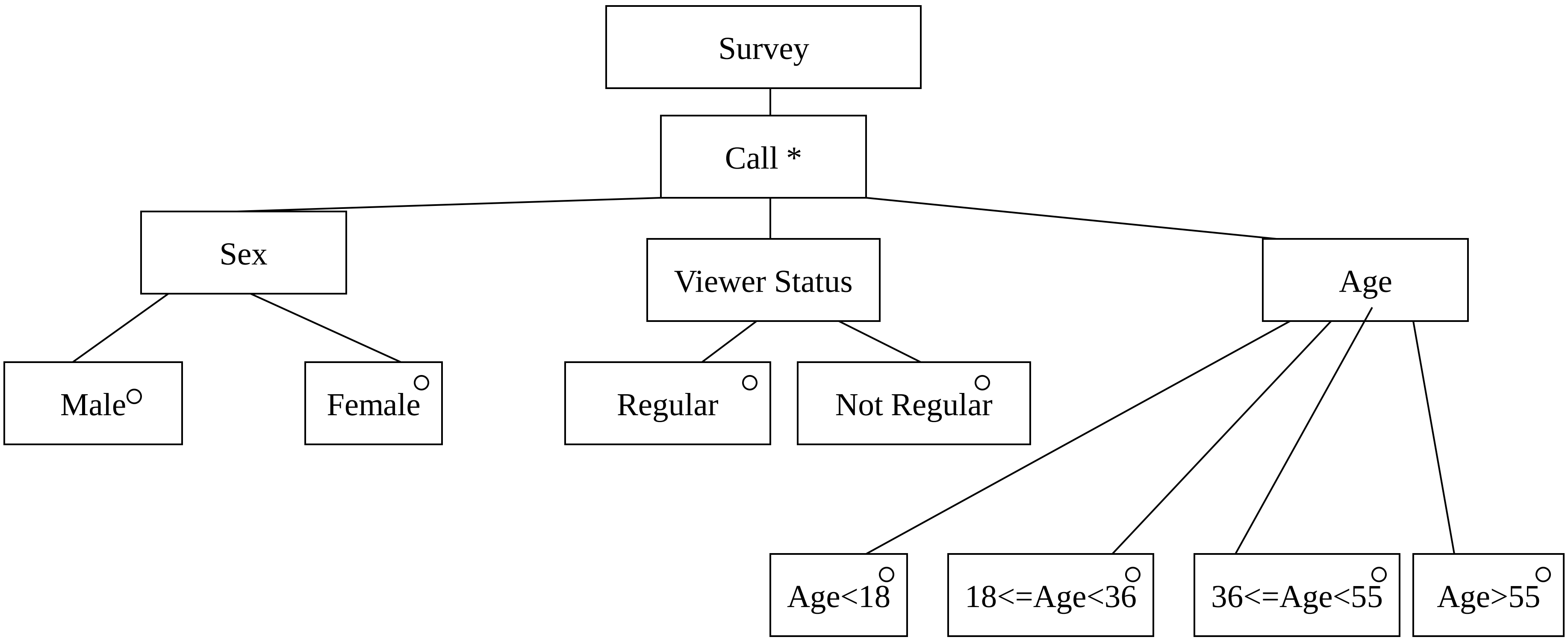
Television Rating Service

Emphasizes WHILE notation for top down design

A television rating service makes a survey of the viewing audience to sample the popularity of tv shows. When a call is made concerning a particular show, the sex and age of the person called, as well as whether or not the person watches the program regularly, are recorded. Write a program that will process the data gathered for the show. The total number of people called, the number who said they watched the show regularly, and the percentage of those who watch the show on a regular basis should be printed. The program should also print a table showing the percentages of those who watch the show by sex and age categories. The output table should look something like the sample shown here.

```
SEX      UNDER 18          18 to 35       36 to 55          OVER 55
-------------------------------------------------------------------
MALE        12.2%            47.5%          34.3%              6.0%
FEMALE      18.5%            32.4%          35.6%             13.5%
-------------------------------------------------------------------
```

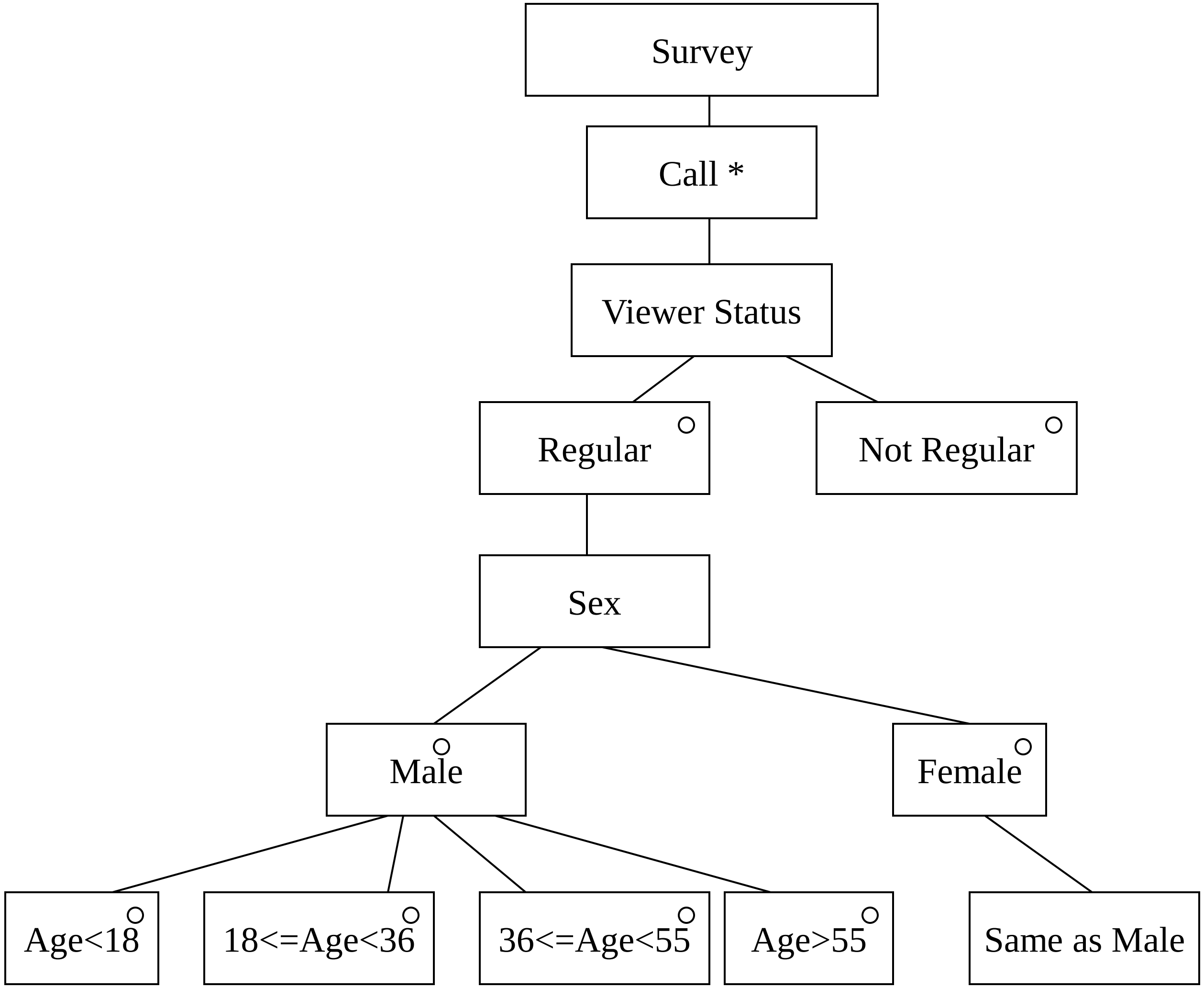
TV Survey


Survey
Call *
Sex
Viewer Status
Age
Male
Female
Regular
Not Regular
Age<18
18<=Age<36
36<=Age<55
Age>55

Requirements demand adjusting view

- Sex and age are typically related only through the person, not as subcategories.
- This problem views them as subcategories of one another.
- It doesn't make any difference which is a subcategory of the other.
- Next slide shows an adjusted view.

TV Survey

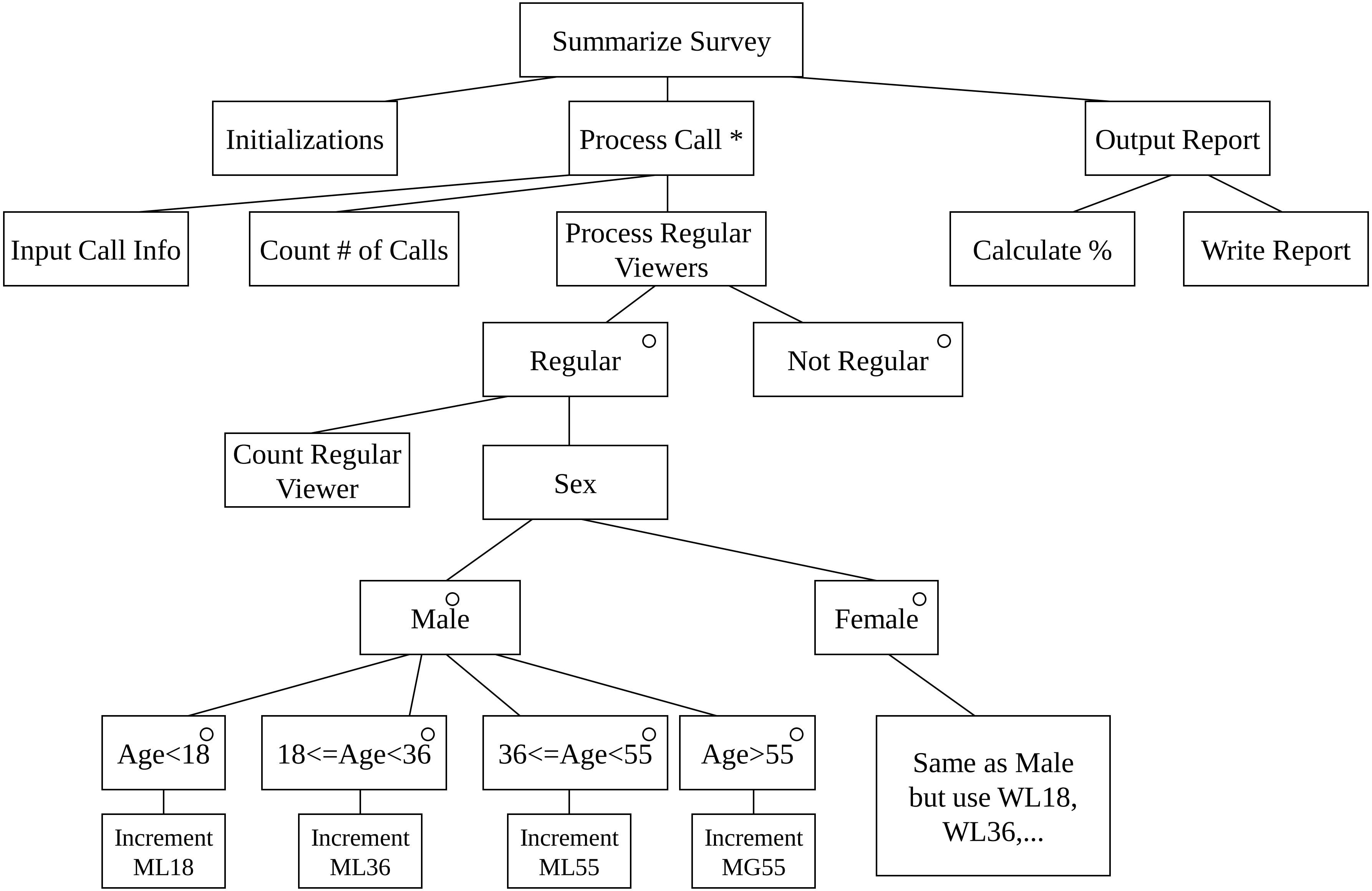

Survey
Call *
Viewer Status
Regular
Not Regular
Sex
Male
Female
Age<18
18<=Age<36
36<=Age<55
Age>55
Same as Male

Step 2

Adapt the structure chart
to accommodate the
program specifications:
DO A **TOP-DOWN DESIGN**

TV Survey

Structure Chart Conversion to Top Down Design



Step 3

Write code to implement the design

TV Survey Implementation

```
#include <iostream.h>
main ()
{  char sex, regular;
   float PercentRegular;
   int age, rv, tv, num_read, NumberToRead;
   int ml18,ml36,ml55,mg55,fl18,fl36,fl55,fg55;
   // initializations
   tv=0; rv=0;
   ml18=0;ml36=0;ml55=0;mg55=0;
   fl18=0;fl36=0;fl55=0;fg55=0;
   cin >> NumberToRead;
   num_read = 0;
```

```
while (num_read < NumberToRead)
    {
                       // PROCESS CALL
          // input call information
       cin >> sex >> age >> regular;
          // count number of calls
       tv++; num_read++;
          // process regular viewer
       if ( (regular == 'R') || (regular == 'r'))
        {     // count as a regular viewer
          rv++;
             // tabulate age/sex info
          if ( (sex == 'M') || (sex == 'm'))
            {  // process Male
               if (age < 18)
                  ml18++;
               else if (age < 36)
                  ml36++;
               else if (age < 55)
                  ml55++;
               else
                  mg55++;
             }
           else
            {  // process Female
               if (age < 18)
                  fl18++;
               else if (age < 36)
                  fl36++;
               else if (age < 55)
                  fl55++;
               else
                  fg55++;
             }
         }
    } // loop
  PercentRegular=rv / tv;
  // write reports to output device
  // ........
} // end
```

Step 4

Use **FUNCTIONS** to implement the design

TV Survey
Implementation
w/functions

It’s Your Turn!

Software System Design

- translates SRS into a

  ===> software system architecture:

  - system's static structure
  - system's possible dynamic behaviour
  - data structures
  - user interface design