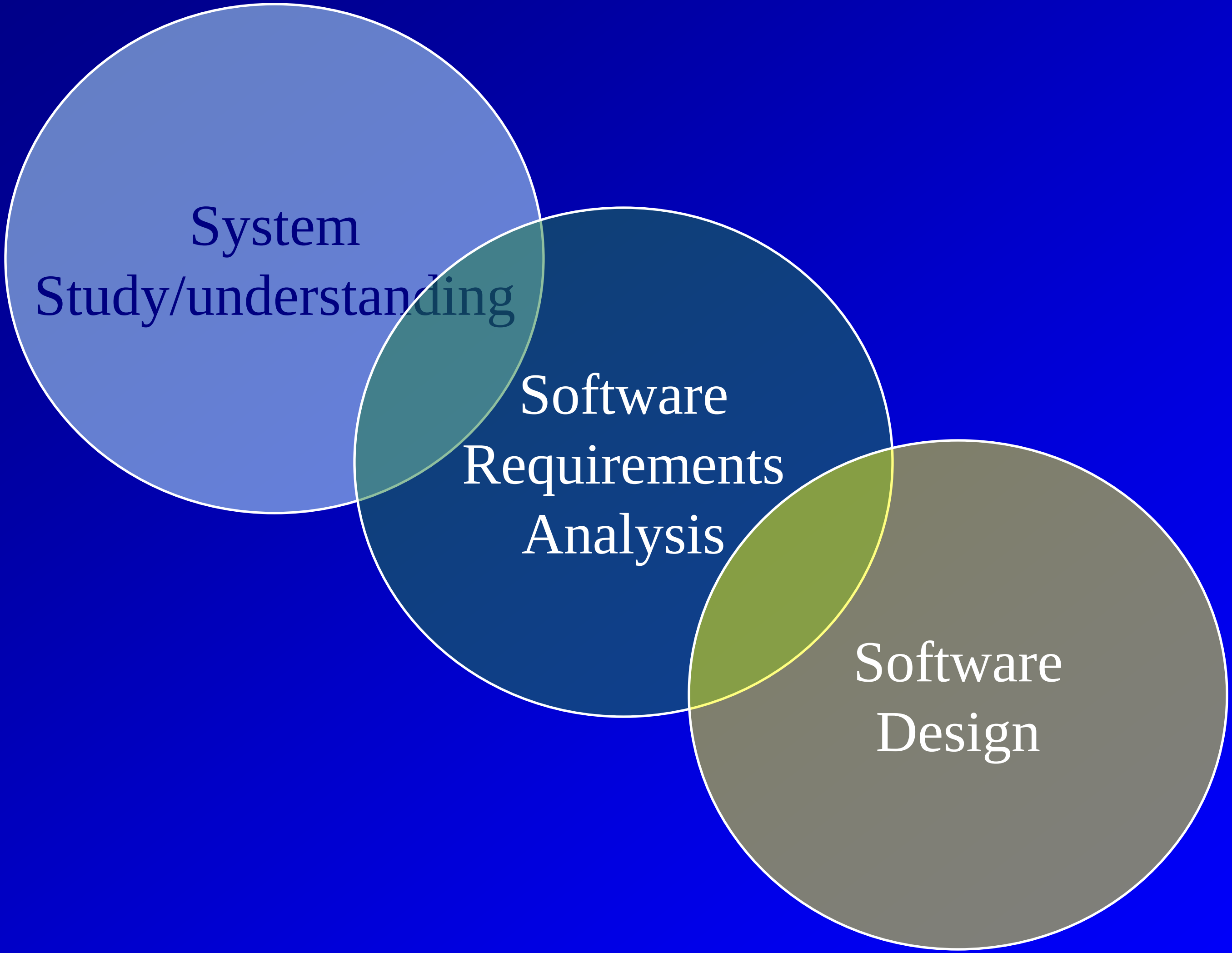
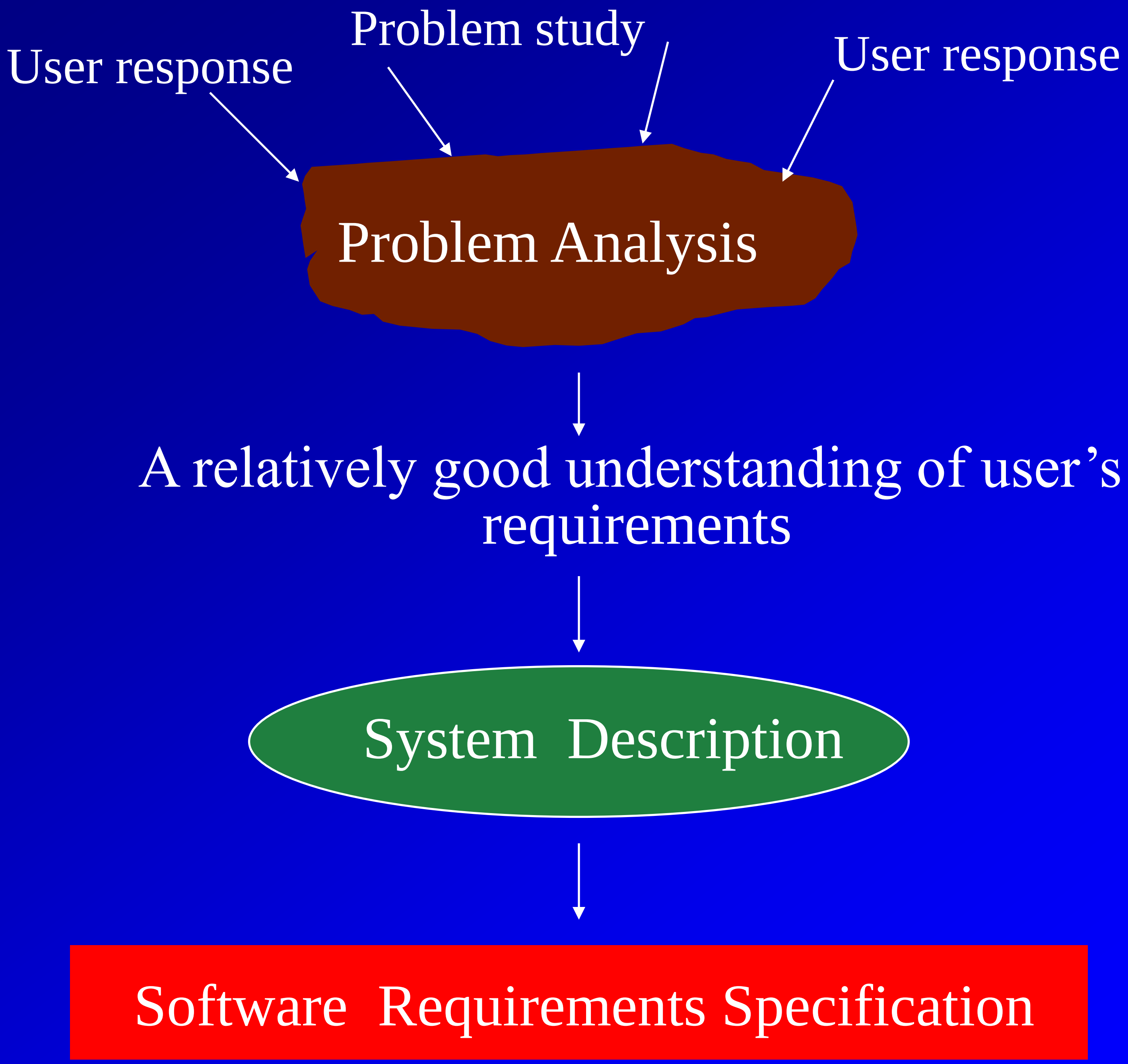
Requirements Analysis Concepts & Principles

Requirements Analysis and Specification

- Many projects fail:
  - because they start implementing the system:
  - without determining whether they are building what the customer really wants?


System Study/understanding
Software Requirements Analysis
Software Design


User response
Problem study
User response
Problem Analysis
A relatively good understanding of user's requirements
System Description
Software Requirements Specification

Requirements Analysis Activities

- Study System Specification, SW Project plan & feasibility report
- Problem Recognition
- Evaluation & understanding
- Modeling
- Specification
- Review

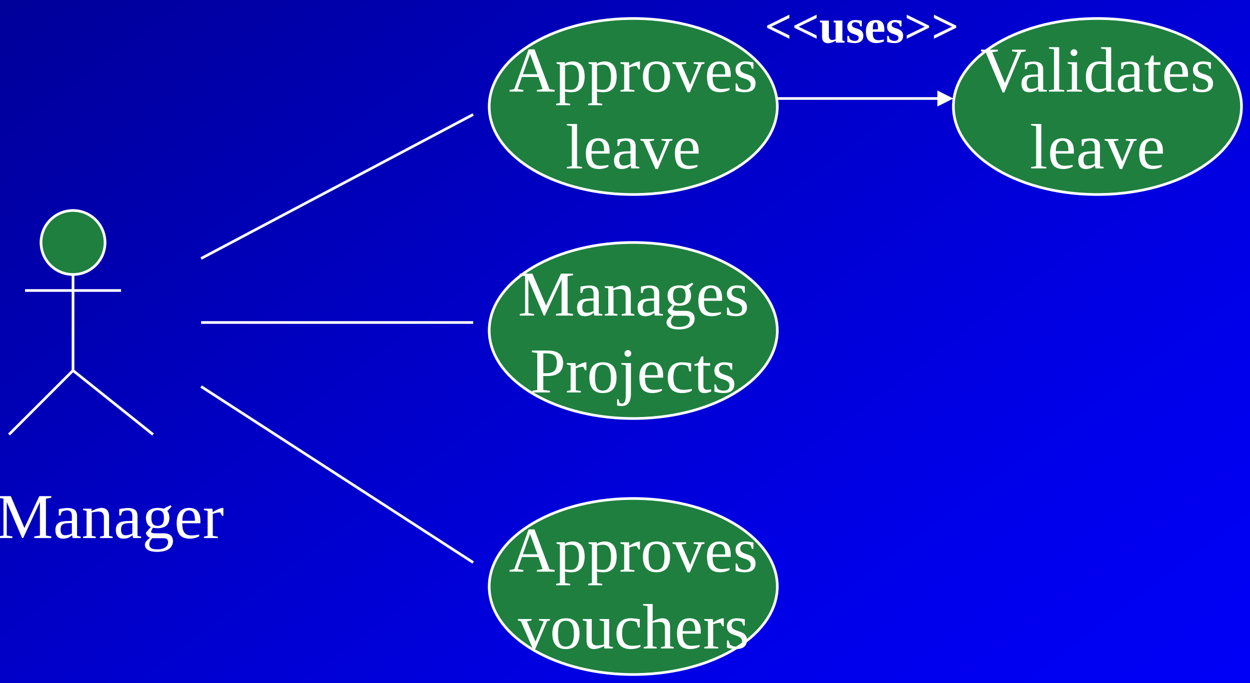
Requirement Elicitation

Fact Gathering Techniques

- Interviews
- Questionnaires
- Analyzing documents
- Observation
- study…

Requirement Elicitation contd.

Use Case diagram (usage scenario of HR department)



Requirement Elicitation contd.

- Scenarios viewed differently by different actors
- Use U_C Modeling : Priority is assigned for each use case
- System functionality with priorities

Analysis Principles

- Information (data) domain must be represented thoroughly
- Functions to perform, must be defined and modeled properly
- Behavior, as consequence of external events must be represented correctly
- Models depicting information/data, function & behavior must be partitioned to uncover detail in layered fashion (i.e. step by step)
- Analysis process should move from essential information (i.e. most abstract level) to implementation (most detailed) level.

Analysis Principles…

- Understand the problem clearly before creating analysis and design **models,**
- Develop prototypes,
- Record the origin & the reason for every requirement (trace),
- Rank (prioritize) all requirements,
- Work to eliminate ambiguity in requirements, if any.

Analysis Principles (contd.)

Information Domain

- Information contents & relationships (Data Model)
- Information Flow (data flow)
- Information structure (data dictionary) and internal organization of data (data structure)

Analysis Principles (contd.)

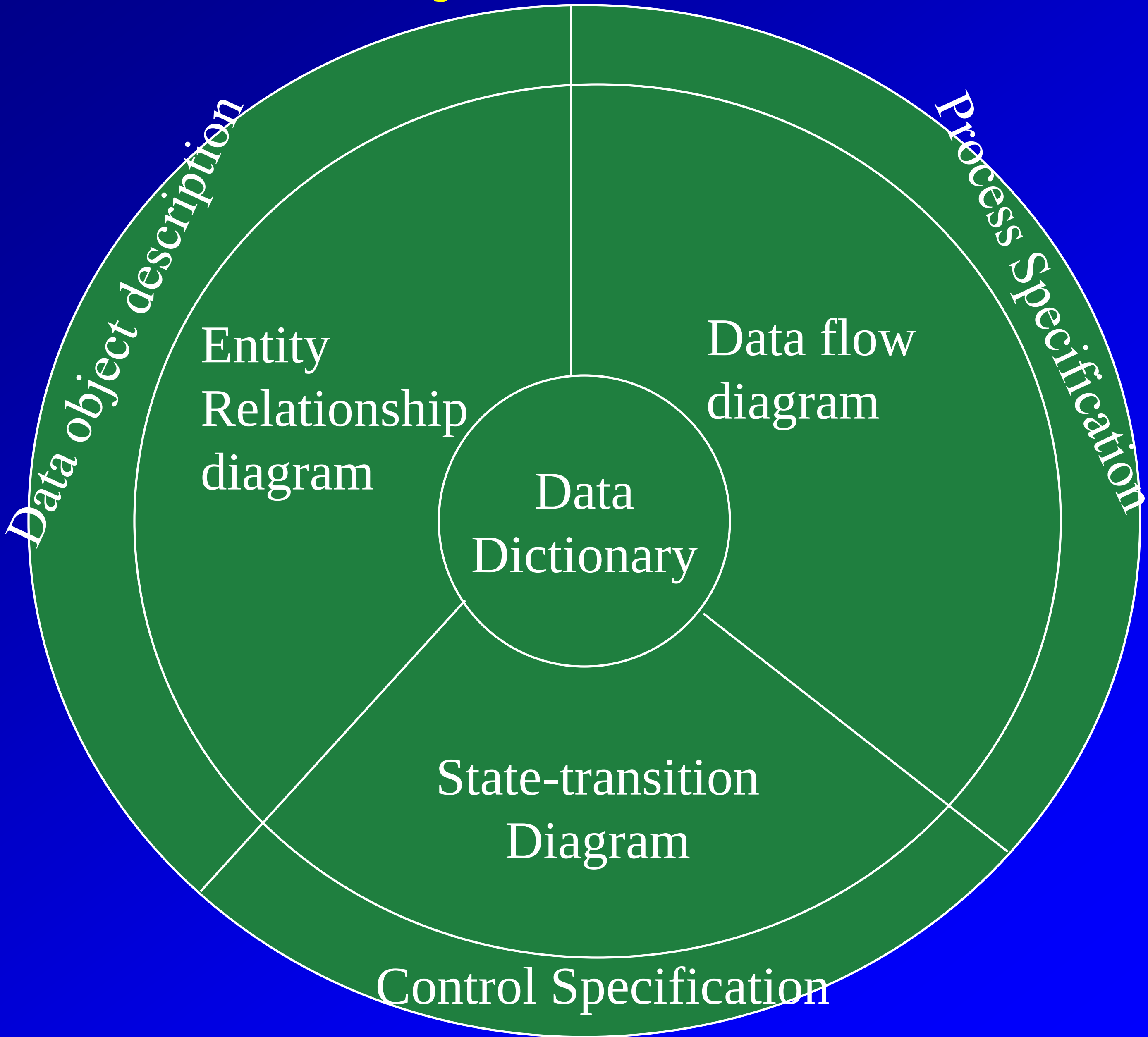
Various models of **Structured Analysis-**

- Functional Model (DFD)
- Structural Model (ERD)
- Behavioral Model (time dependant behavior) (STD)

Modeling

- A model is an abstract representation of a view of the system,
- Build different models of the system to view it from various ways/angles
- Benefits of modeling
  - focuses on important features of the system while avoiding less important features
  - discuss changes and corrections to the user's requirements with low cost and minimal risk
  - verify that Analyst correctly understands the user's requirements

Analysis Models



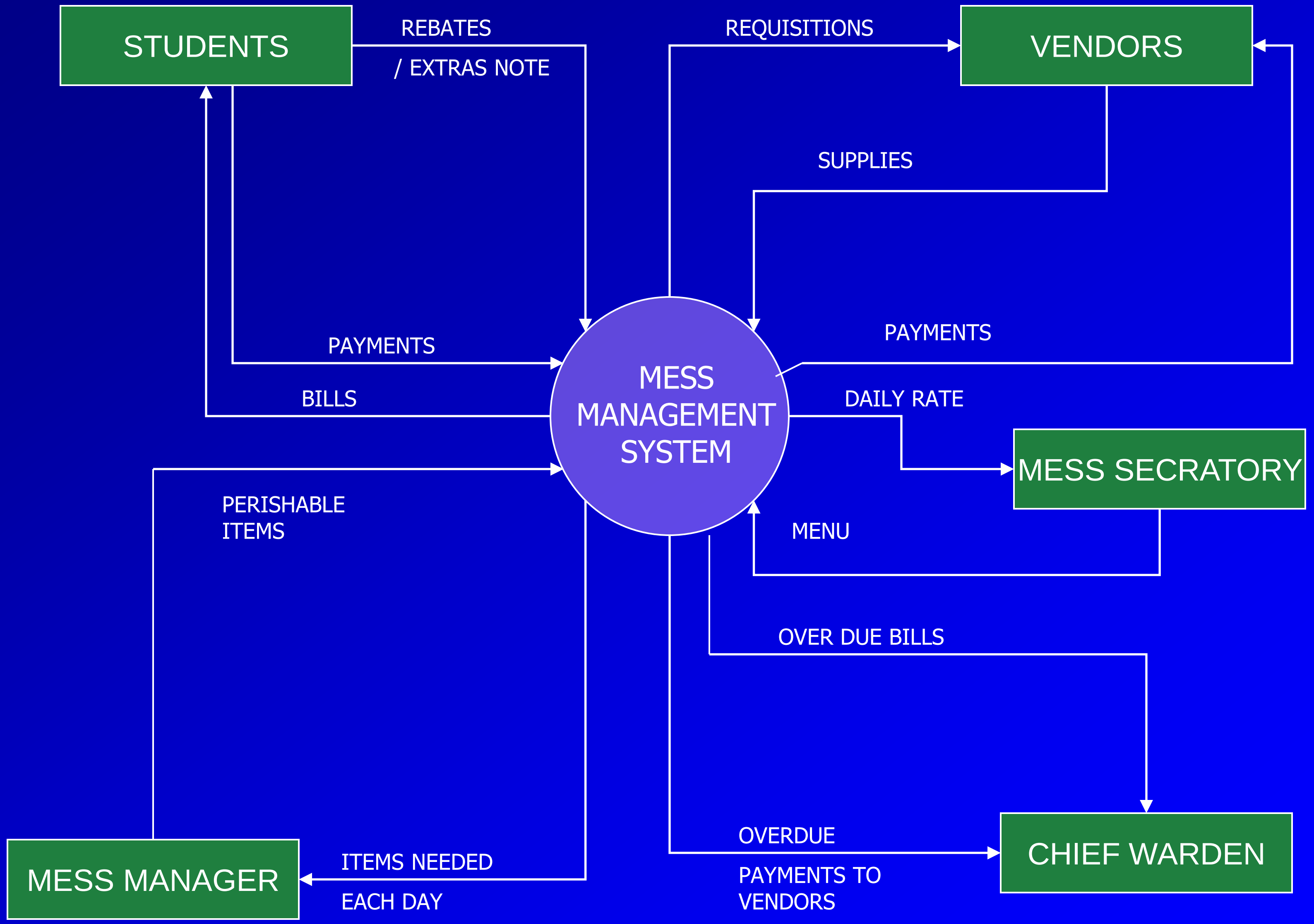
Analysis Models

Data Flow Diagram	functional view
Entity Relationship Diagram	Stored Data view
State Transition Diagram	Time-dependent behavioral view

Data Flow Diagram (DFD)

- A DFD depicts the flow of information within a system and between the system and the outside world.
- It does not show sequence and control information.
- A Data Flow Diagram (DFD) contains:
  - External entities, Processes, Data flows, Data stores.

LEVEL 0 DFD for Mess Management System



DFD Primitives

External Entity represents a source or sink for data. These are usually outside the scope of the area under study.

Process transforms or manipulates data

Data flow carries data between an entity and a process, a process and a store, or between two processes

Data Store represents a repository of data

Connector symbols are used if the diagram spans multiple pages.

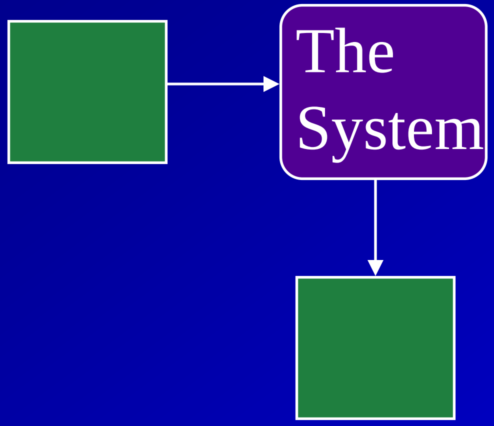
DFD Hierarchy

- DFD can be used at different levels of abstraction.
- The highest level DFD is called a Context Diagram
- A process may be exploded into a lower level DFD to show more details.
- In such a case, the net input and output flows must be balanced across the DFD levels.
- When decomposing a DFD, you must protect inputs to and outputs from a **process** at the next level of decomposition. This is called **balancing**.

- The lower level DFD may have data stores that are local to it, and not shown in the higher level DFD.

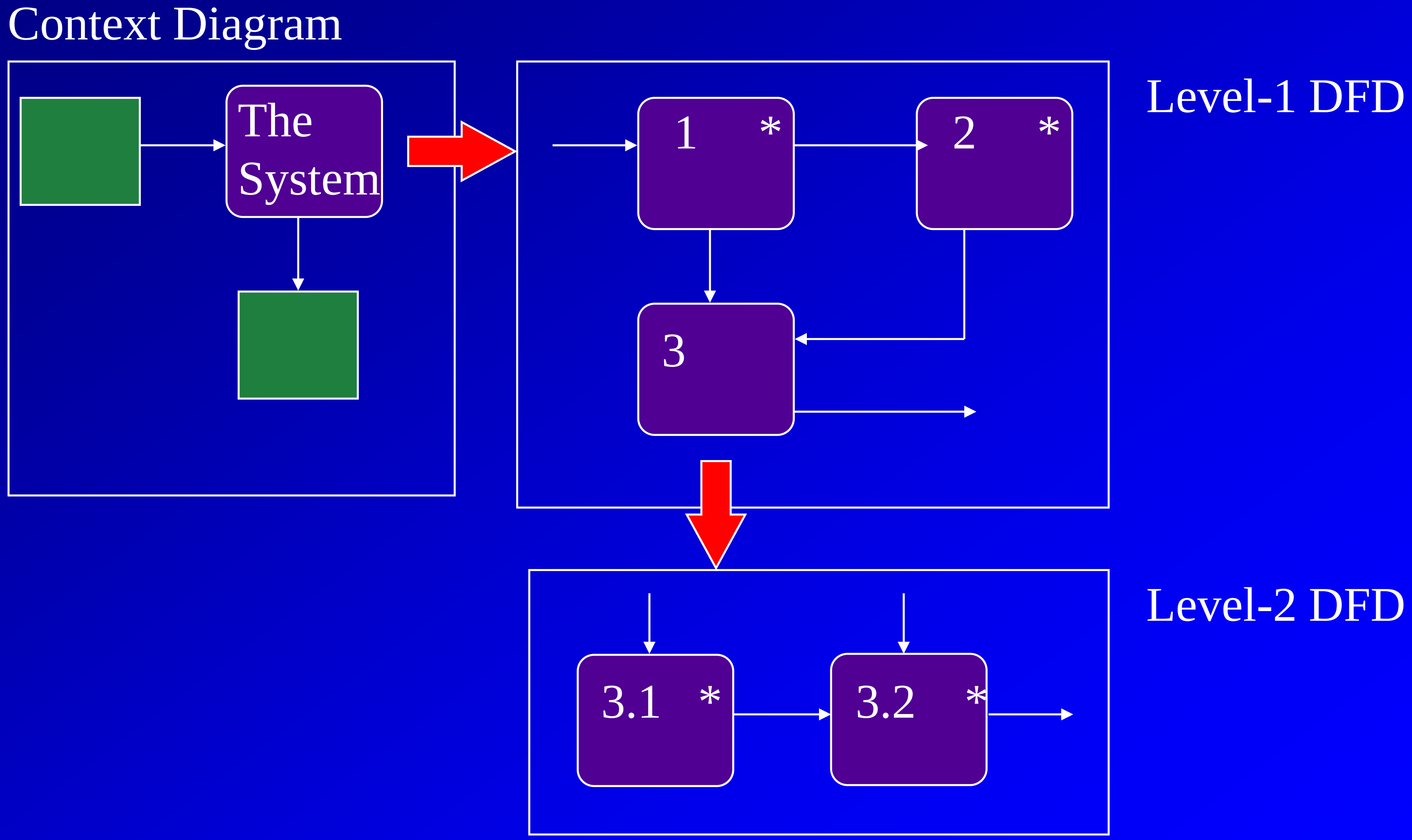
Context Diagram (DFD Level 0)

Example: Context Diagram



- Represents an entire system as a single process
- Shows the scope of the system
- Highlights the interfaces between the system and the outside terminators (external entities)

Leveled DFDs



DFD Guidelines

- Ideally, a DFD at any level, should have no more than half of a dozen processes and related stores.
- Choose meaningful names for processes, flows, stores, and terminators
  - use active unambiguous **verbs** for process names such as Issue, Purchase, Create, Calculate, Compute, Produce, etc.
- **Number** each and every process.

DFD Guidelines (continue..)

- Make sure that DFD is internally consistent
  - avoid infinite sinks
  - avoid spontaneous generation of processes
  - beware of unlabeled flows and unlabeled processes
  - beware of read-only or write-only stores

DFD Guidelines (continue..)

- Avoid overly complex DFDs
- Balance DFDs across the levels
- Data flows and stores must be described in the **Data dictionary**.
- Processes must be supported either by a lower level DFD or a **mini-specification** (algorithm).

Lowest Level Processes in a DFD

- A process that cannot be exploded further has an associated mini specification.
- Mini spec can be in:
  - Structured English
  - Decision Table/Decision Tree

Example: Process Specification in **Structured English** (with proper indentation)

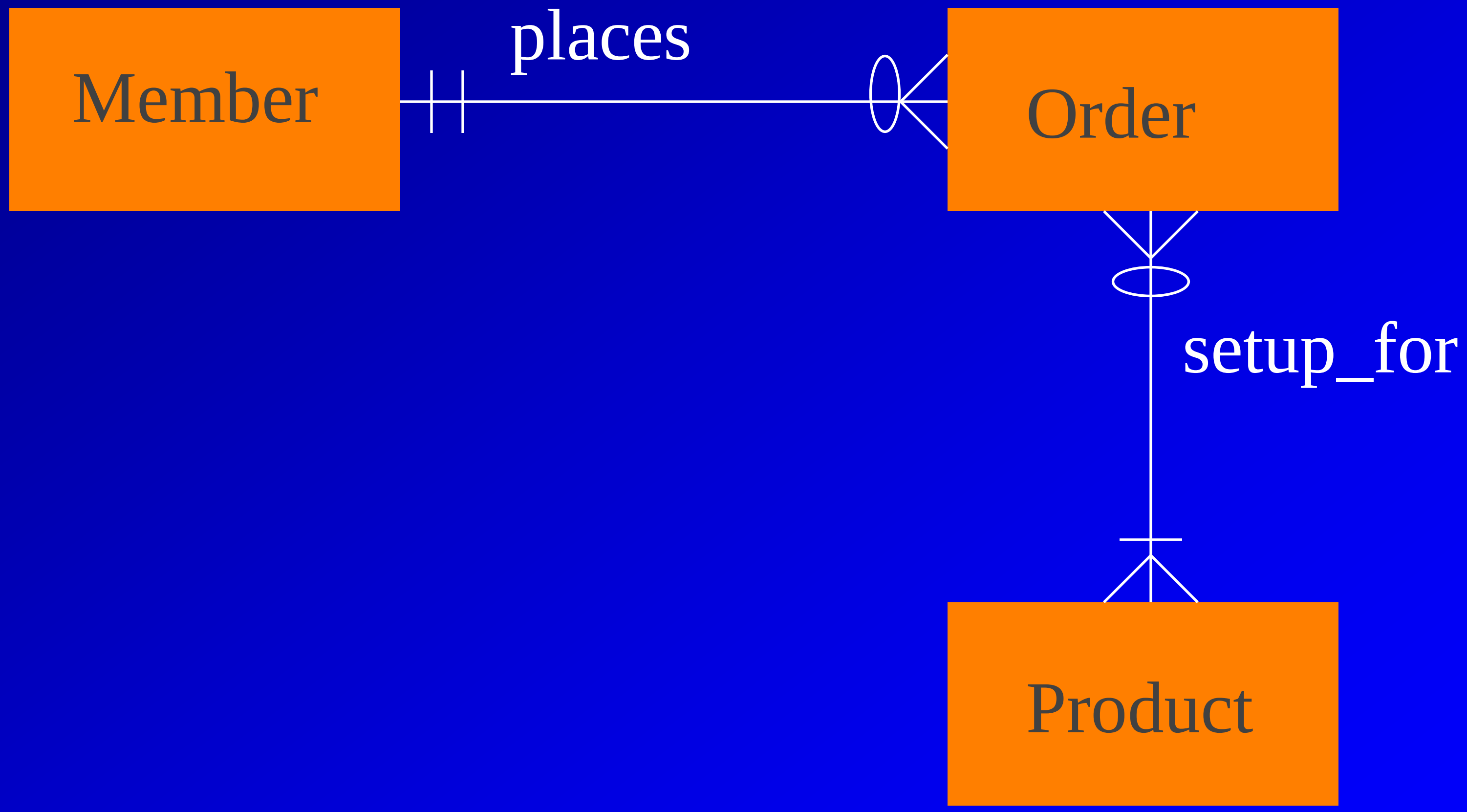
```
FOR EACH  item in the Indent DO
    IF  Item is on Rate-Contract
          prepare PO to Rate-Contract vendor
    ELSE IF  Item is proprietary
          prepare PO to manufacturer
    ELSE
          prepare regular PO to registered vendor
END FOR
```

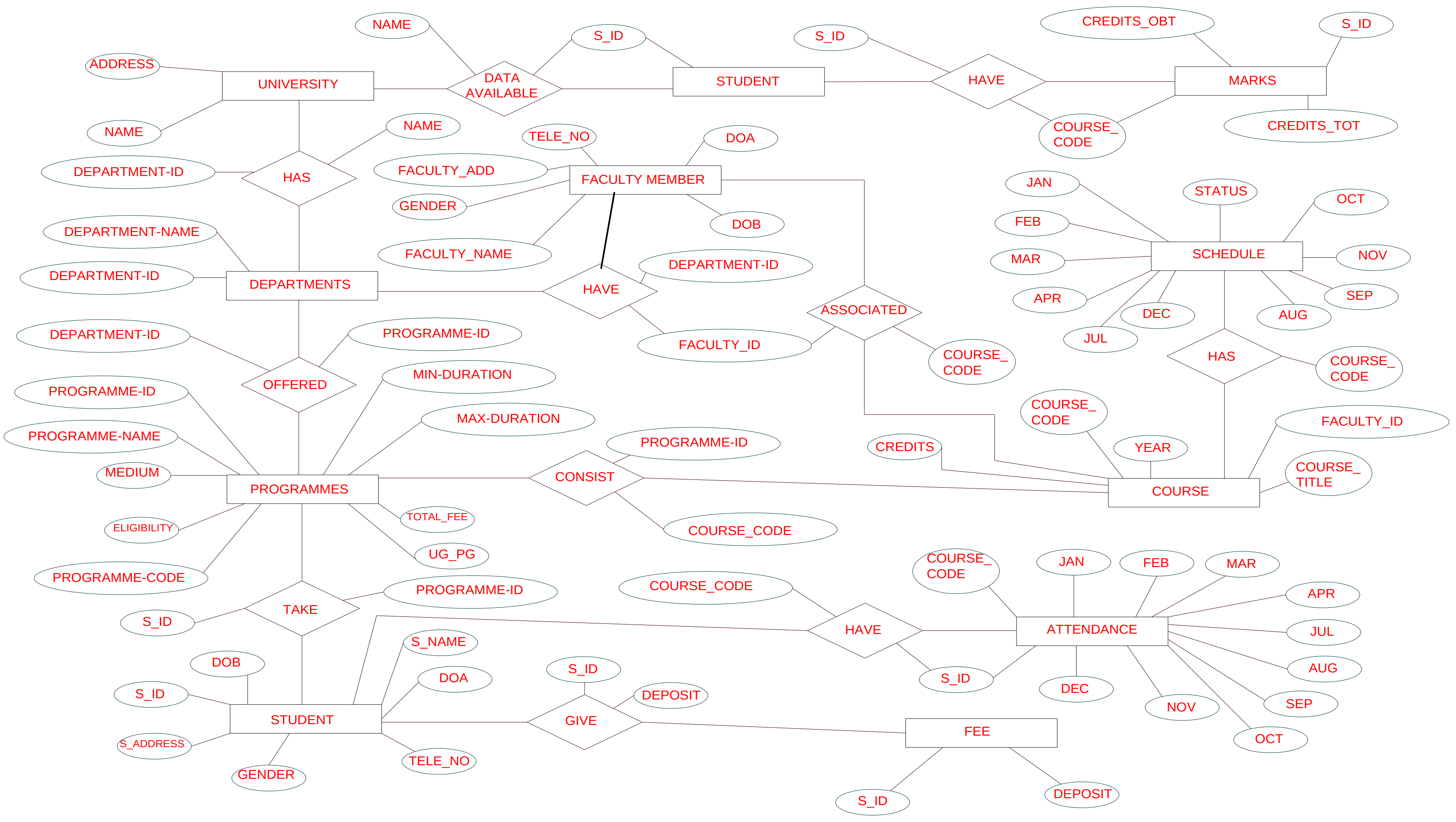
Data Modeling
Entity Relationship Diagram (ERD)

- It gives a logical view of the data and the relationships between them in a system.
- Major components:
  - Entities
  - Relationships between entities
  - Cardinality

one-to-one	recursive relationship
one-to-many	super type -subtype
many-to-many	

Example: ERD





NAME
S_ID
S_ID
CREDITS_OBT
S_ID
ADDRESS
UNIVERSITY
DATA AVAILABLE
STUDENT
HAVE
MARKS
NAME
COURSE_ CODE
CREDITS_TOT
NAME
TELE_NO
DOA
DEPARTMENT-ID
HAS
FACULTY_ADD
FACULTY MEMBER
GENDER
JAN
STATUS
OCT
FEB
DOB
DEPARTMENT-NAME
FACULTY_NAME
MAR
SCHEDULE
NOV
DEPARTMENT-ID
DEPARTMENT-ID
DEPARTMENTS
HAVE
APR
SEP
ASSOCIATED
DEC
AUG
JUL
DEPARTMENT-ID
PROGRAMME-ID
FACULTY_ID
COURSE_ CODE
HAS
COURSE_ CODE
OFFERED
MIN-DURATION
PROGRAMME-ID
COURSE_ CODE
MAX-DURATION
FACULTY_ID
PROGRAMME-NAME
PROGRAMME-ID
CREDITS
YEAR
MEDIUM
CONSIST
COURSE_ TITLE
PROGRAMMES
COURSE
TOTAL_FEE
ELIGIBILITY
COURSE_CODE
UG_PG
PROGRAMME-CODE
COURSE_ CODE
JAN
FEB
MAR
COURSE_CODE
PROGRAMME-ID
APR
S_ID
TAKE
HAVE
ATTENDANCE
JUL
S_NAME
DOB
S_ID
DOA
AUG
S_ID
S_ID
DEPOSIT
DEC
S_ID
NOV
SEP
STUDENT
GIVE
FEE
S_ADDRESS
OCT
TELE_NO
GENDER
S_ID
DEPOSIT

The Data Dictionary

- The data dictionary is an organized listing of all data elements pertinent to the system with precise rigorous definitions so that both user and SA will have common understanding of all data elements.
- It describes:
  - meaning and composition of data flows, data stores
  - relevant values and units of elementary data elements
  - details of relationships between stores that are highlighted in a ERD

Data Dictionary Notation

Example:

```
customer-name =  title + first-name +
                 (middle-name) + last-name
title = [Mr. | Miss | Ms. | Mrs.| Dr. | Prof. ]
first-name = {legal-character}
order = customer-name  + shipping-addr
                       + {item}
```

D D…

=	is composed of
+	and
[|]	either / or
{ }	repetition of (string)
()	optional data
---	for comments

Modeling Time-Dependent System Behavior

- State Transition Diagram
- Entity Life Histories

State Transition Diagram (STD)

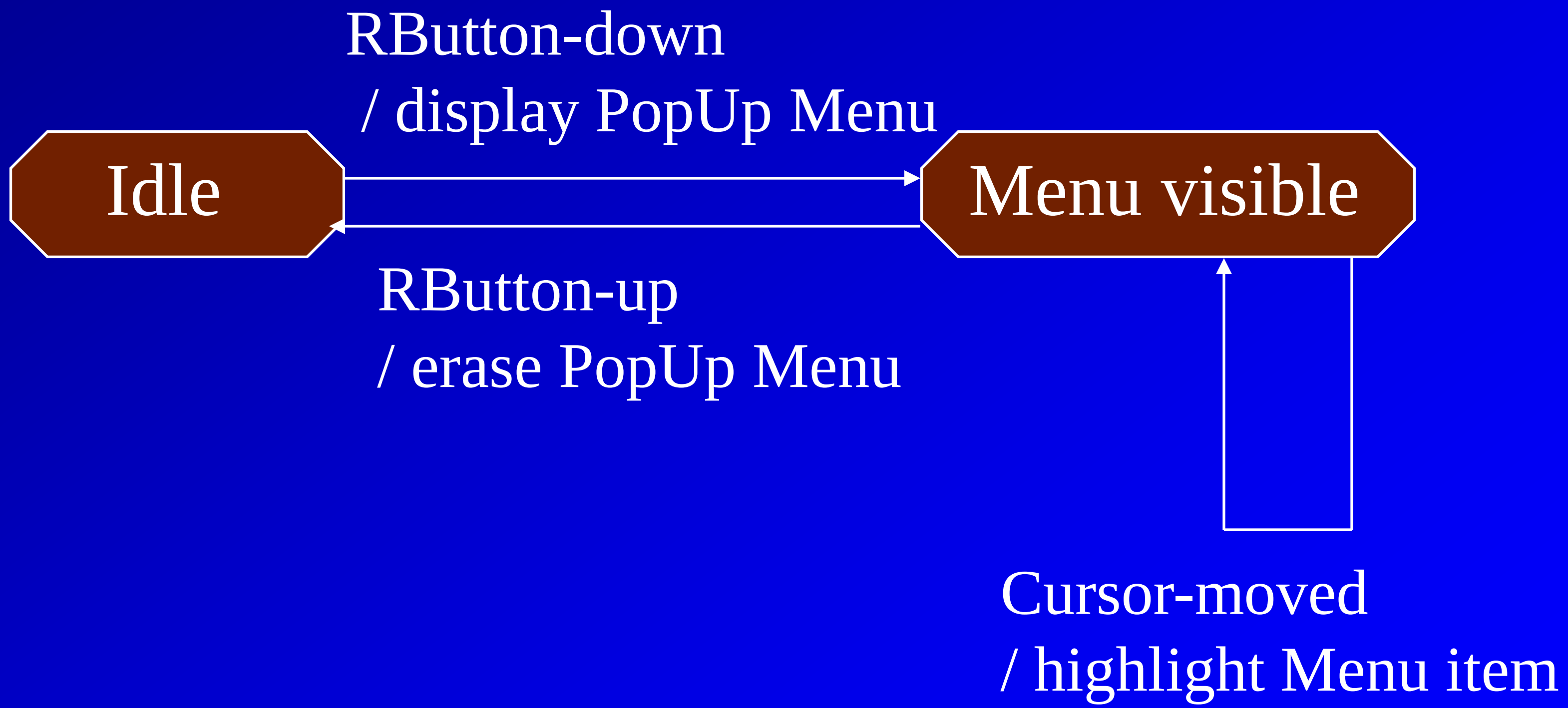
- It highlights the time-dependent behavior of a system.
- It shows ‘what’ happens ‘when’
- Major components
  - States
  - Transitions between states
- Useful for describing behavior of
  - real-time systems, interactive systems

State Transition Diagram…

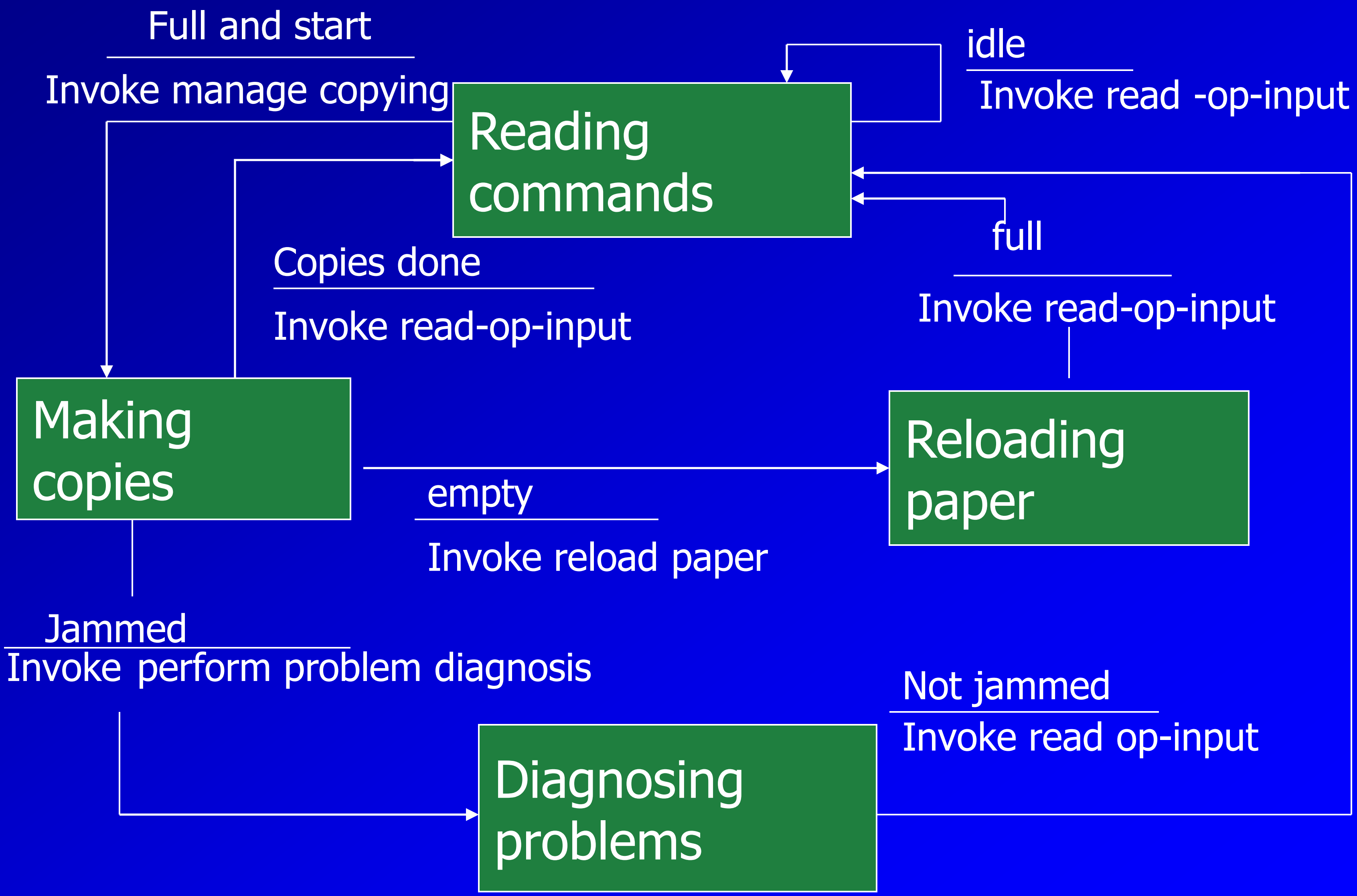
- The State Transition Diagram (STD) represents the behavior of a system by displaying it’s states and the events which cause system to change its state.

- STD indicates what actions are taken as a consequence of a particular event.

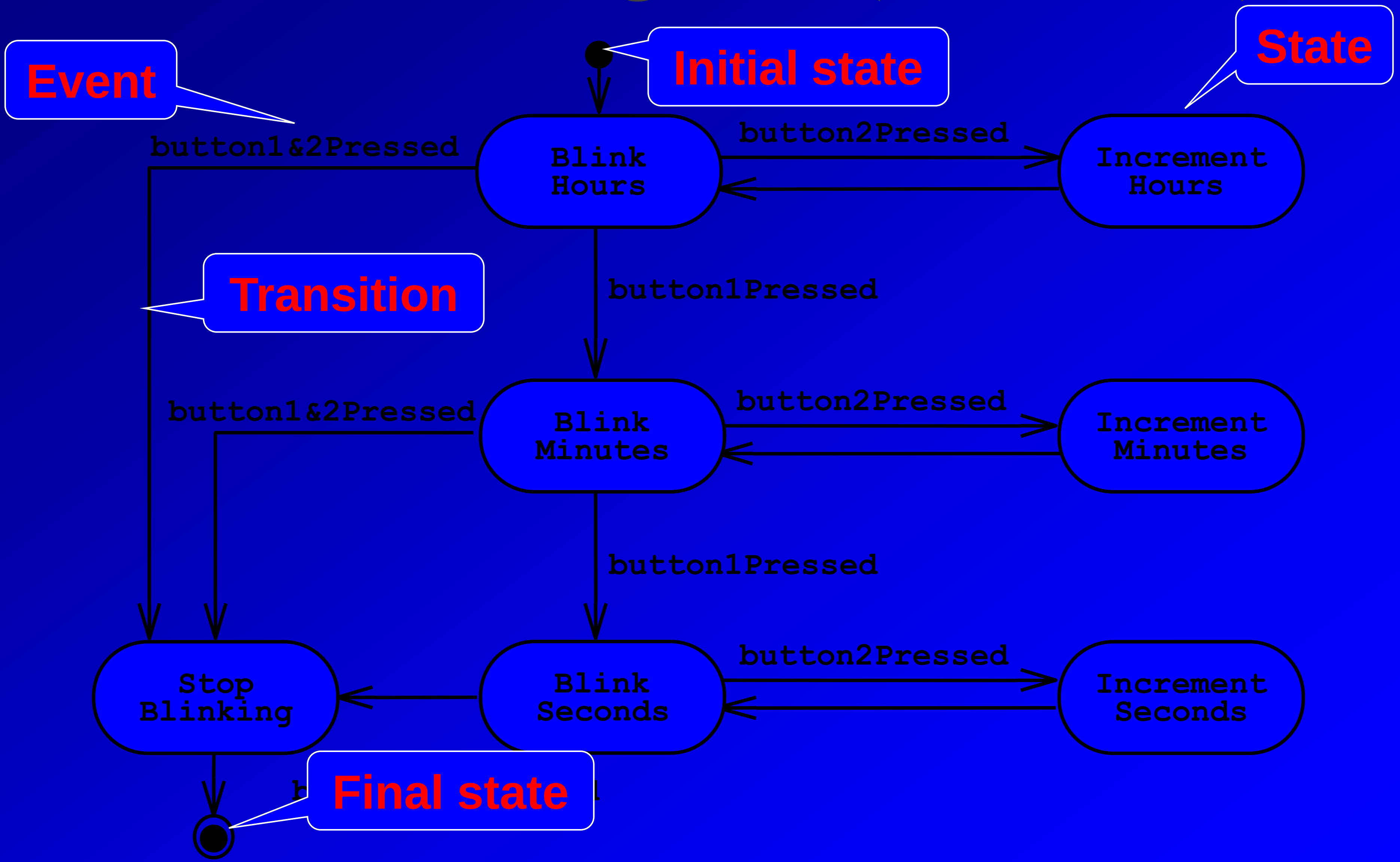
Example: State Transition Diagram



STD for photocopier software



Statechart Diagrams (ex. Digital Watch)



Cross Checking the different Models

- Balancing ERD against the DFD, Process specs.
- Balancing DFD against the STD
- Balancing ERD against Data dictionary
- Balancing DFD against Data dictionary

Balancing ERD and DFD, Process Specs.

- Every store on the DFD must correspond to an entity or a relationship or a combination of an entity and relationship on the ERD.
- Entity names on the ERD and data store names on the DFD must match.
- The data dictionary entries must apply to both the DFD and the ERD data elements.