Requirements Engineering process

- Used to discover, analyze, validate and manage requirements
- Dependent on the application domain, the people involved and the organization developing the requirements
- However, there are a number of generic activities common to all processes:
  - *feasibility study*
  - *requirements elicitation and analysis*
  - *requirements specification*
  - *requirements validation*
  - *requirements management*

Feasibility study

- A short focused study that checks
  - If the product contributes to organizational objectives
  - If the product can be engineered using current technology and within budget
  - If the product can be integrated with other products that are used
- Based on a *concise description* of the product and the user needs
- Produces a ***report*** that decides whether or not the proposed product is worthwhile

Feasibility study (2)

- The required info are collected by interviewing:
  - client manager
  - software engineers with skills in the specific *application domain*
  - experts of technology to be used
  - end users
- Example *typical questions*:
  - What if the product wasn't implemented?
  - What are current process problems?
  - How will the proposed product help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed product?

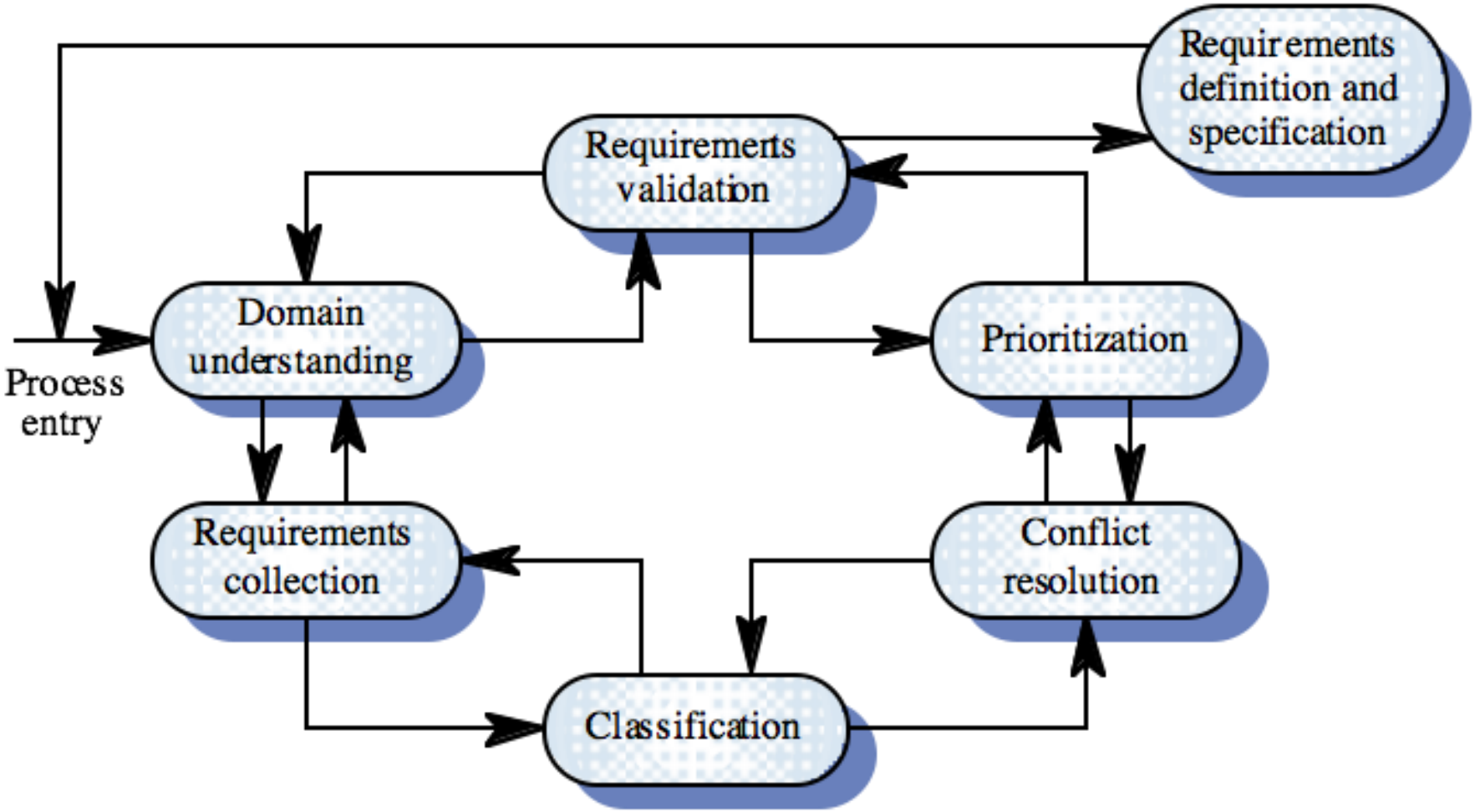
Requirements elicitation and analysis

- Sometimes called *requirements discovery*
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.
- These are called ***stakeholders*** (i.e., people who may have a direct or indirect interest in the product requirements)

Problems of requirements analysis

- Stakeholders *don't know what they really want*
- Stakeholders express requirements in their own terms
- Different stakeholders may have *conflicting requirements*
- Organizational and political factors may influence the product requirements
- The *requirements change* during the analysis process
- New stakeholders may emerge and the *business environment change*

Requirements elicitation and analysis process



Req. elicitation and analysis (2)

- Requirements elicitation techniques
  - Ethnography
  - Use cases (*scenarios*-based)
  - Prototyping
- Requirements analysis (and specification) techniques
  - **semi-formal**, based on *system models* and used by *structured analysis* methods or *object-oriented analysis* methods
  - **formal** (based on *formal languages*, such Petri Net, FSM, Z, etc.)

Requirements validation

- Concerned with demonstrating that the requirements define the product that the customer really wants
- Very important to avoid costly requirements reworks in later lifecycle phases
- *Checks* include:
  - validity: does the system provide the functions which best support the customer's needs?
  - consistency: are there any requirements conflicts?
  - completeness: are all functions required by the customer included?
  - realism: can the requirements be implemented given available budget and technology
  - verifiability: can the requirements be checked?

Requirements validation techniques

- *Informal* reviews
- *Formal* reviews
  - *walkthroughs*
  - *inspections*
- Prototyping
- Test-case generation
- Automated consistency analysis (for requirements specified in formal languages)

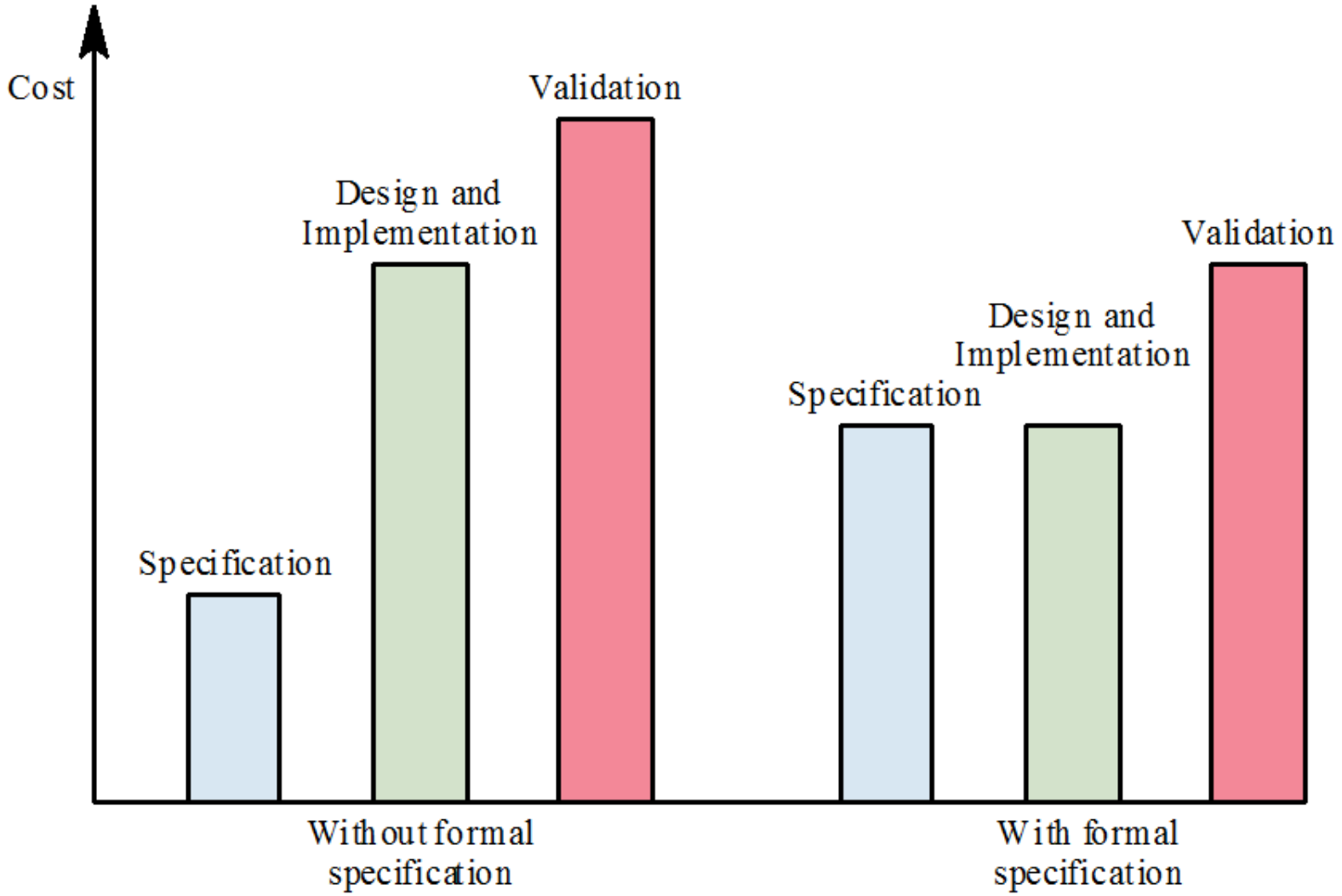
Requirements management

- The process of managing changing requirements during the requirements engineering process and product development
- Classification of requirements in terms of evolution:
  - **enduring** requirements (low probability of change)
  - **volatile** requirements (high probability of change):
    - *mutable requirements (requirements that change due to the system's environment)*
    - *emergent requirements (requirements that emerge as understanding of the product improves)*
    - *consequential requirements (requirements that result from the introduction of the computer system)*
    - *compatibility requirements (requirements that depend on other systems or organizational processes)*

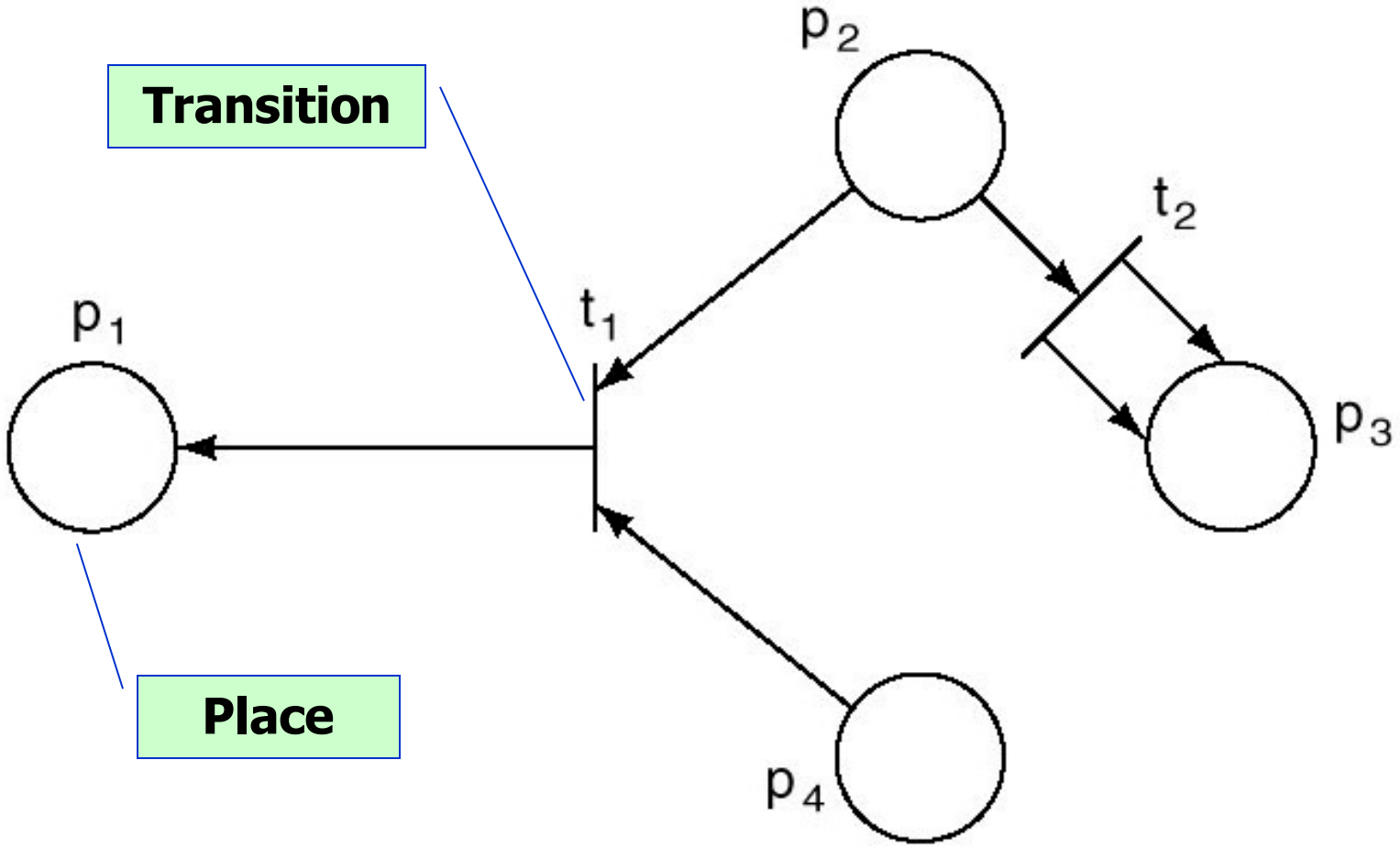
Requirements management planning

- During the requirements engineering process, you have to plan:
  - requirements (unique) identification
  - change impact analysis
    - in terms of costs and feasibility
  - traceability policies
    - relationships between requirements, their sources and the system design
  - CASE tool support
    - the tool support required to help manage requirements change

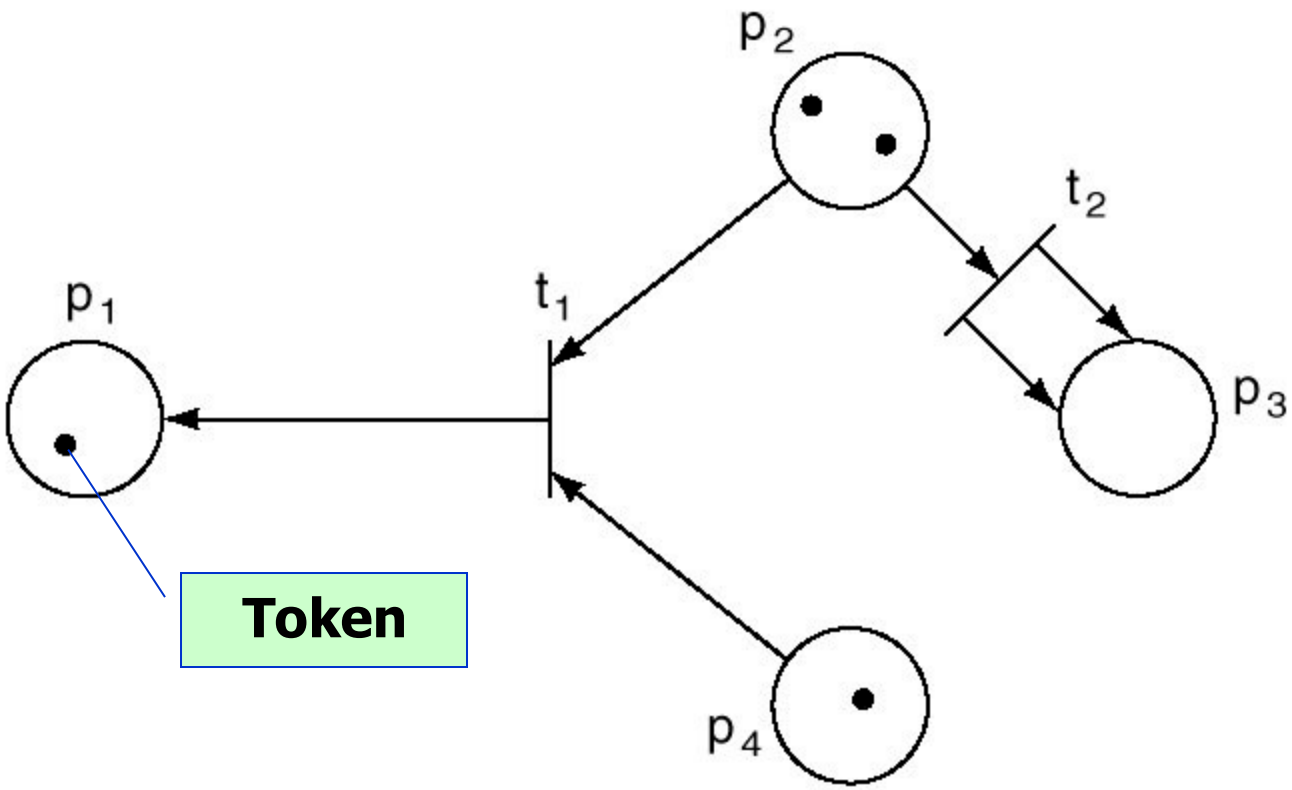
Formal vs. *informal* specification



Formal specification with *Petri Net*

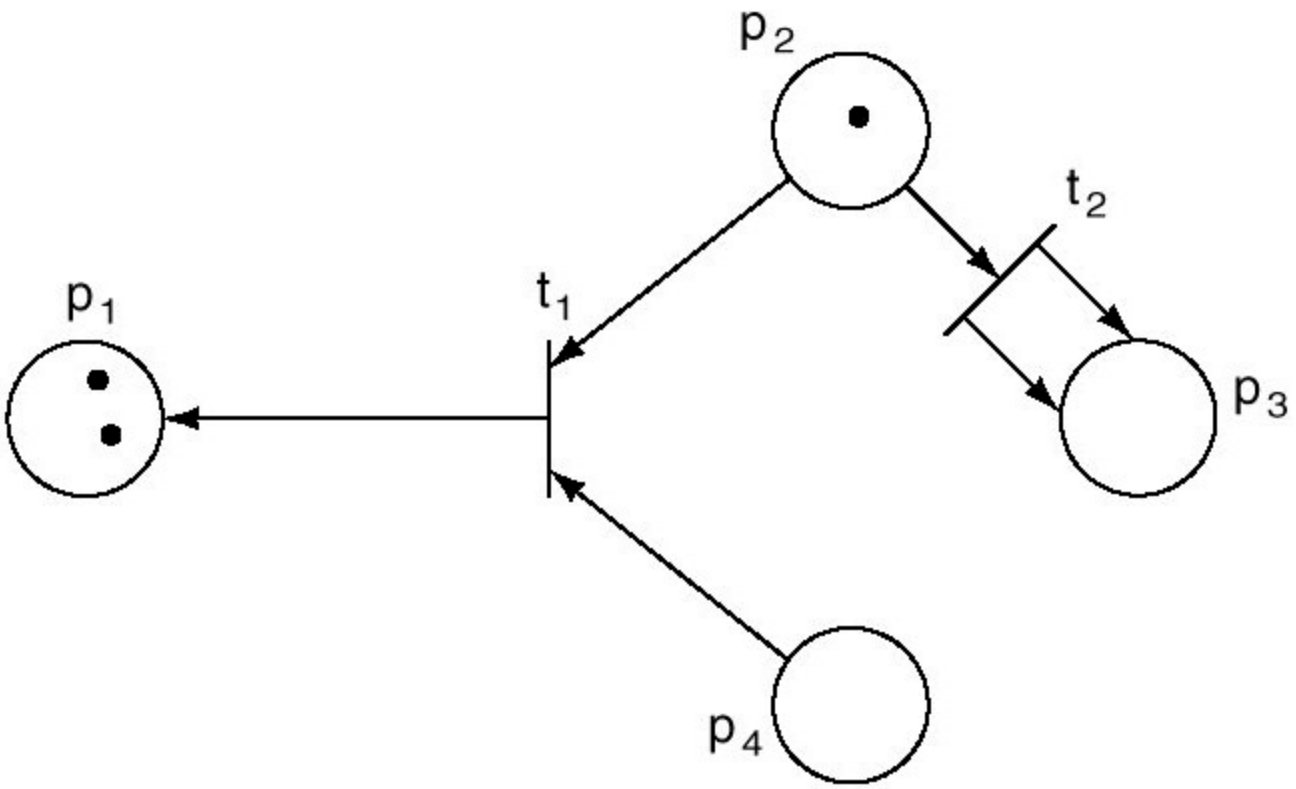


Marked Petri Net (1)



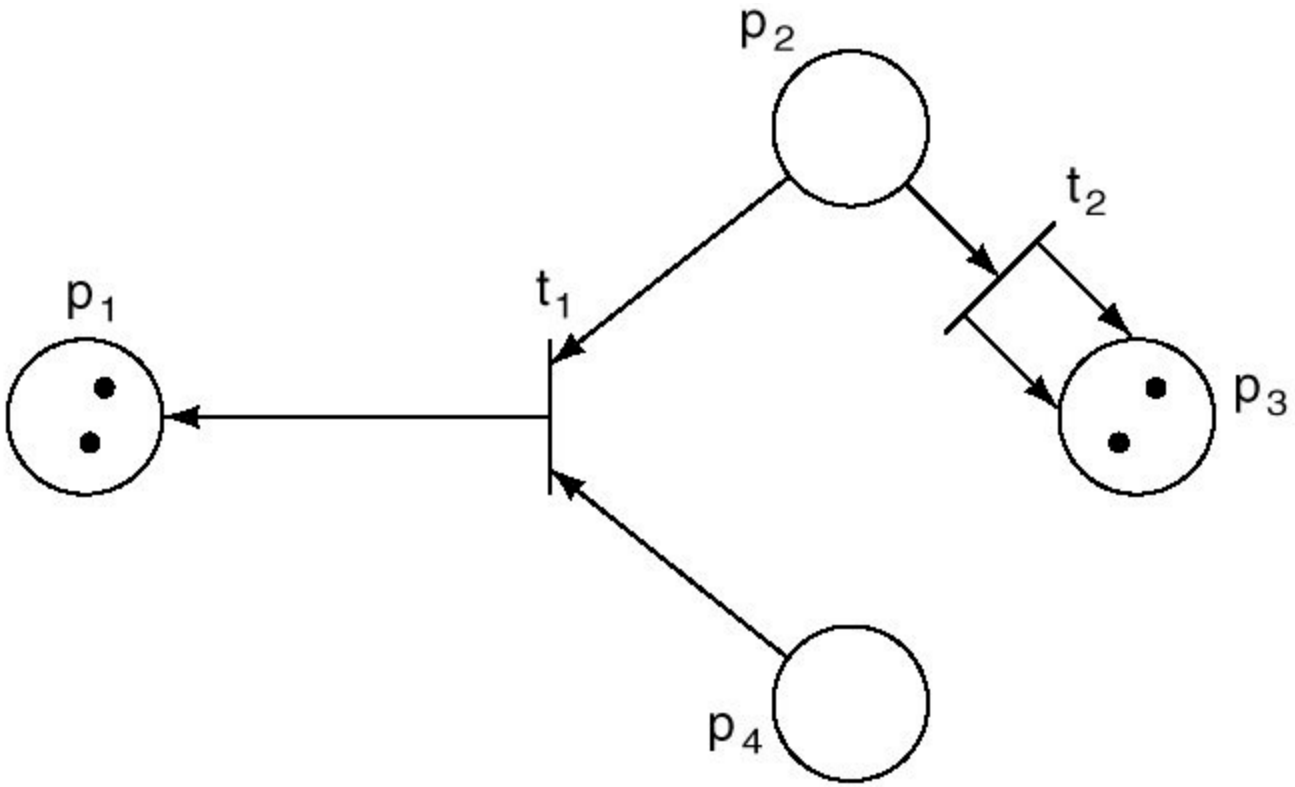
Marked Petri Net (2)

- after firing transition $\mathbf{t_1}$

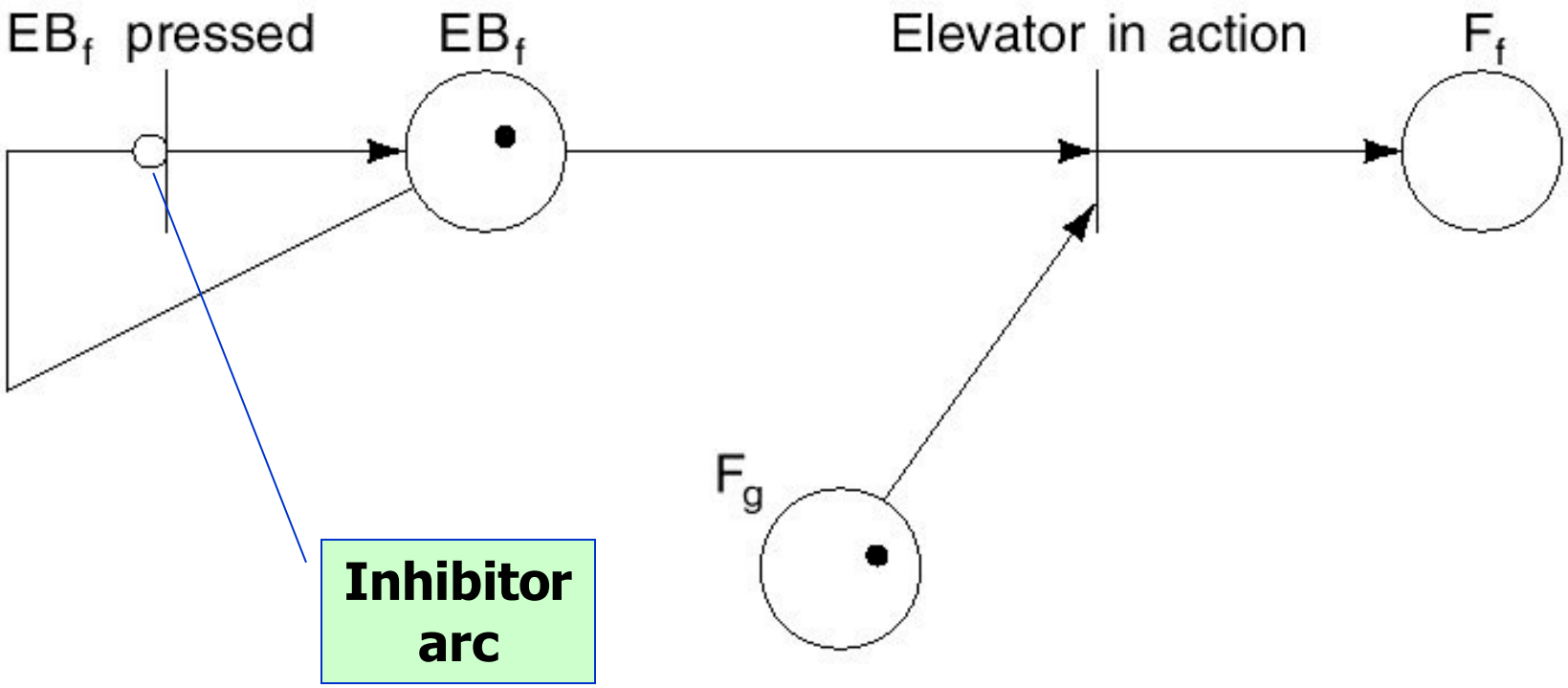


Marked Petri Net (3)

- after firing transition $\mathbf{t_2}$

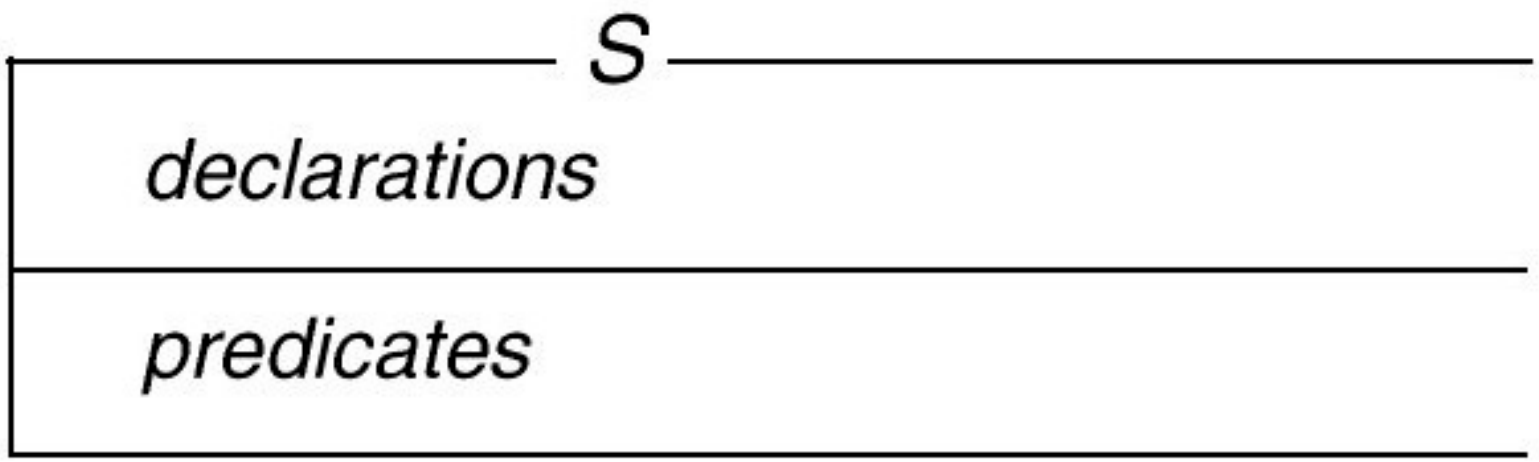


Petri Net: example

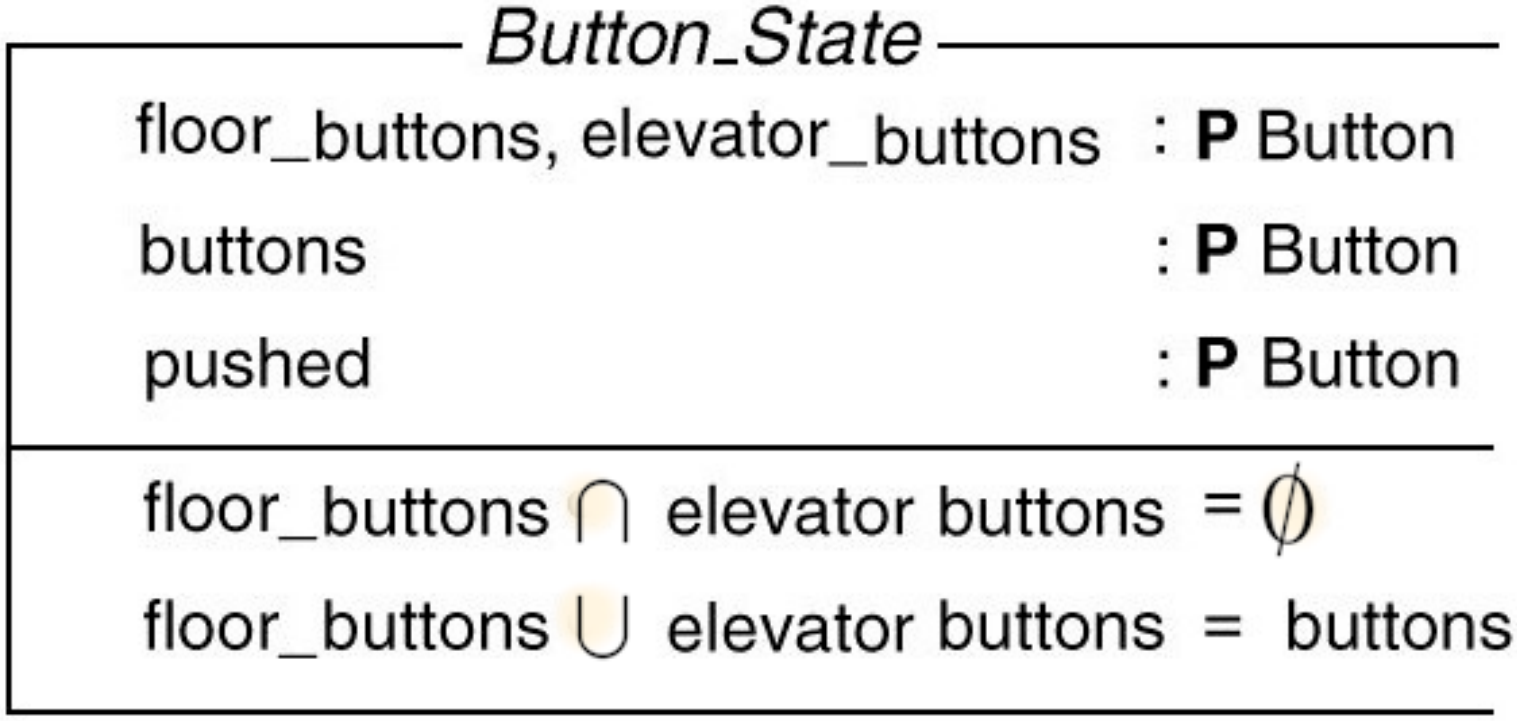


Formal specification with *Z*

- Consists of a set of *schemas*
- *Schema template*:



The Z language
state specification example



Abstract Initial State

```
Button_init := [Button_State' | pushed' = ∅]
```

The Z language
operation specification example

Push_Button

Δ *Button_State*

button?: Button

$$(button? \in buttons) \land$$
$$(((button? \notin pushed) \land (pushed' = pushed \cup \{button?\})) \lor$$
$$((button? \in pushed) \land (pushed' = pushed)))$$

Formal verification

- Formal verification is a technique to check several quality characteristics of a system
- The aim is to explore all the possible states of the system to check for properties of interest such as
  - *safety* (nothing bad will happen)
  - *liveness* (something good will happen)
  - *fairness* (independent subsystems make progress)
  - common design flaws such as:
    - *deadlock* (the system should not reach a state in which no further action is possible)
    - *livelock/starvation* (when a subsystem is prevented from taking any action because of resource contention)

Deductive Reasoning

- *Deductive reasoning* and *model checking* are two alternative approaches to formal verification
- **Deductive reasoning** requires the specification of some desired properties as formulas, by means of which a proof system (i.e., a set of axioms and rules) allows the designer to prove that the system meets the expected behavior formally
- This approach may be applied to verify infinite state systems. Generally, the procedure can be partially automated. However, no limit can be assumed on the amount of memory or time that may be needed to verify the system (i.e., find a proof)
- Such a technique, indeed, is very time consuming and can only be performed by experts of logical reasoning

Model Checking

- **Model Checking** was defined by Clarke and Emerson as "*an automated technique that, given a finite-state model of a system and a logical property, systematically checks whether this property holds for (a given initial state in) that model*"
- Model Checking verifies that some properties, expressed using a formal logic, are satisfied by a model of the system
- The verification requires that all possible states of the model are explored. As a consequence, only finite-state transition systems can be verified
- This approach is fully supported by automatic tools (e.g. SPIN). Every time the model violates a desired property, the tool produces a counterexample, which illustrates the behavior that caused the violation of the property

Semi-formal specs: *system models*

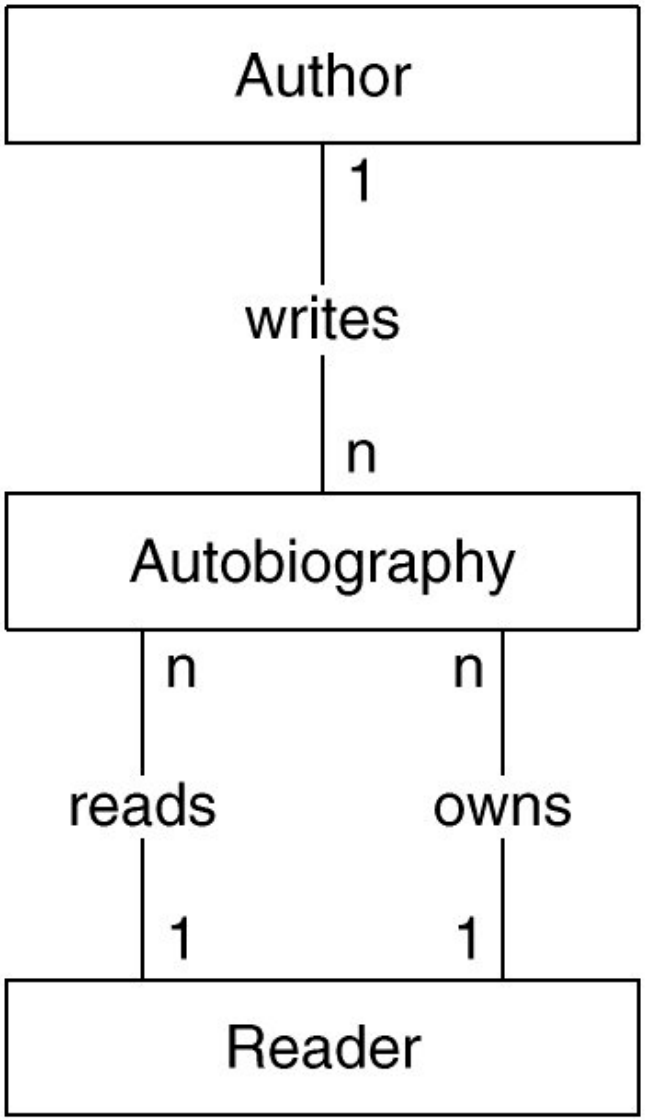
- A **system model** is an *abstract representation* of a system, produced to facilitate the understanding of the system behavior and properties at system development time
- System models are produced by requirements analysis (*specification*) methods that make use of *semi-formal techniques*
- Such requirements analysis methods can be:
  - **structured (or procedural)** analysis methods
  - **object-oriented** analysis methods
- For a complete system description it is necessary to represent various aspects (*data*, *behavior* and *control*)

Model types

- **data model:** to represent the static and structural aspects (*data requirements*)
  - *ERD (not UML)*
  - *class diagram (UML)*
- **behavioral model**: to represent the functional aspects (*functional requirements*)
  - *data flow diagram (not UML)*
  - *use case diagram (UML)*
  - *activity diagram (UML)*
  - *interaction diagram (UML)*
- **dynamic model**: to represent the control aspects, as well as how the functions of the behavioral model modify the data of the data model
  - *state diagram (UML)*

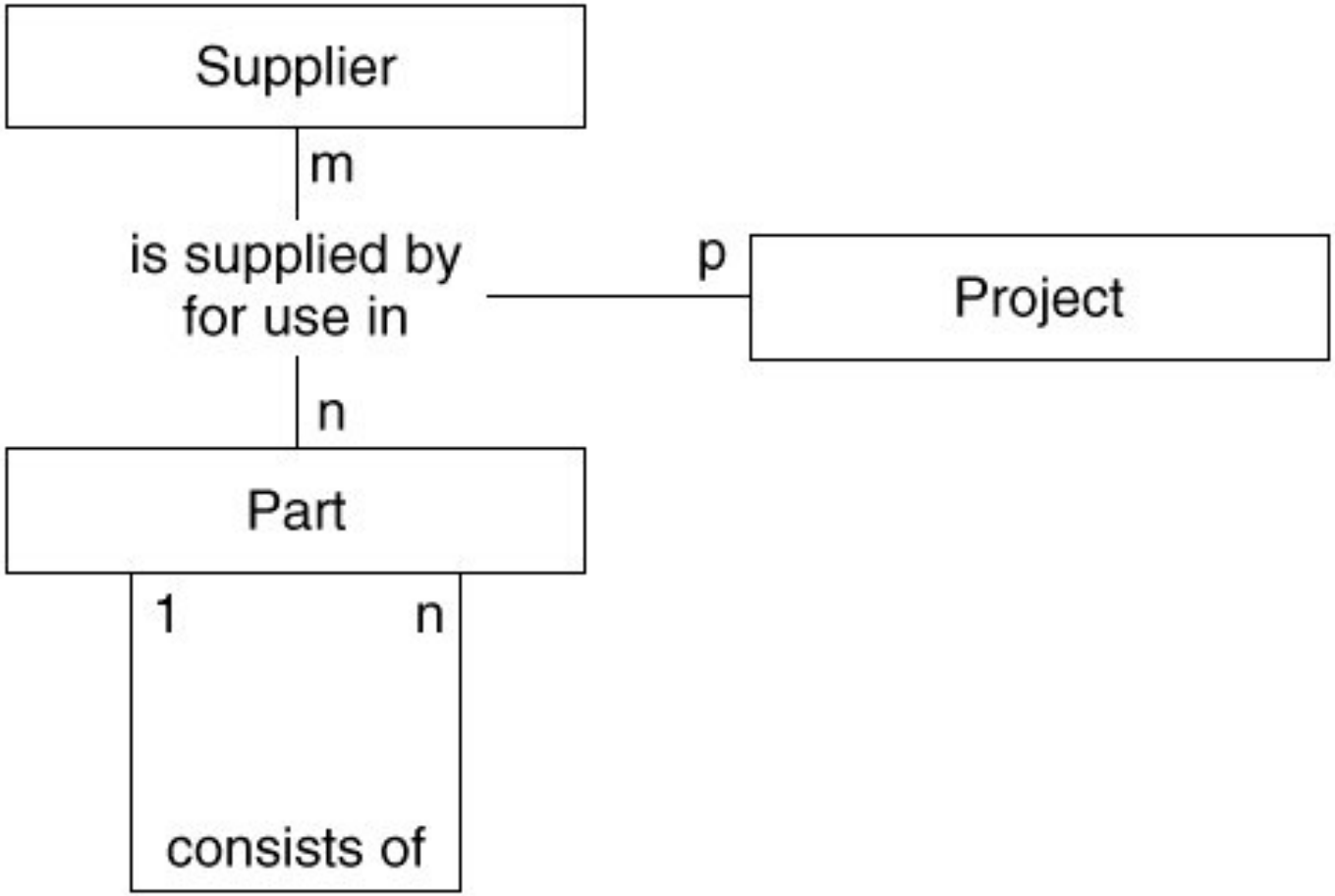
Entity Relationship Diagram (ERD)

one-to-many relations



ERD

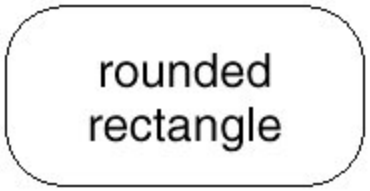
many-to-many relations



Data Flow Diagram (DFD)

Source or destination of data

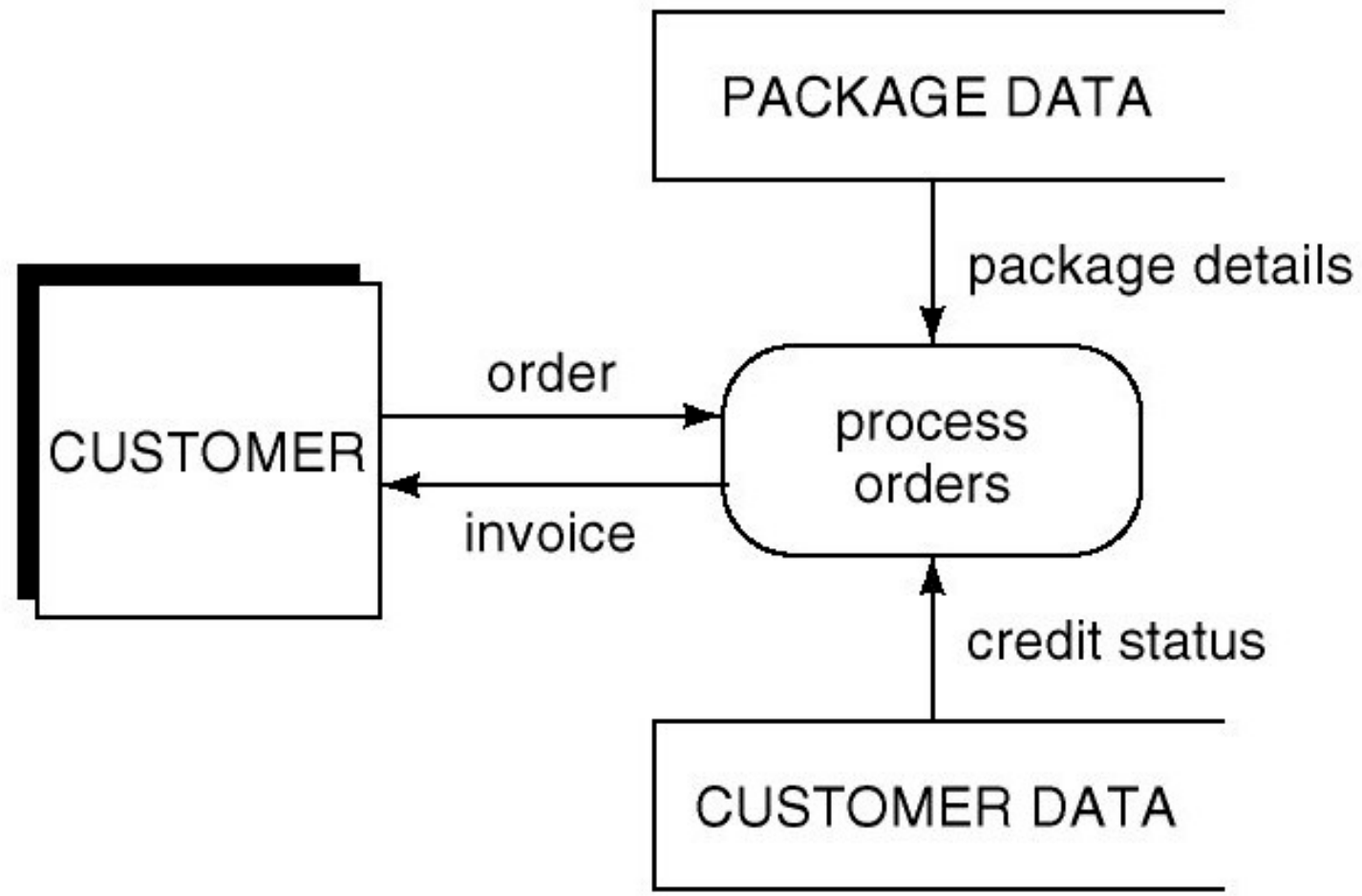
Flow of data



Process which transforms a flow of data

Store of data

DFD example
first refinement



DFD example
second refinement