

Specifying Software Using Petri Nets

Neelam Gupta
The University of Arizona

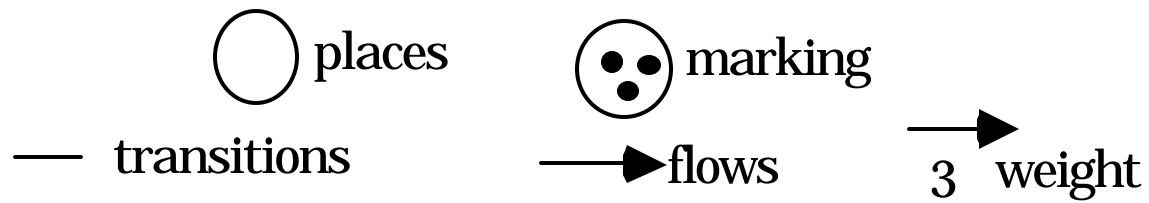
Petri Nets

- Petri nets can specify asynchronous processes in a system
- A Petri net is a **quadruple** (P, T, F, W)
 - Finite set of *places* P (denoted by circles)
 - Finite set of *transition* T (denoted by bars)
 - Finite set of *arrows* F connecting either places to transitions or transitions to places. Arrows are described by the *flow relation*

$$F \subseteq (P \times T) \cup (T \times P)$$

- A *weight function* $W: F \rightarrow N - \{0\}$ which associates a non-zero natural number to each element of F . If no weight is associated with an arrow, default weight of 1 is assumed.

Graphical Representation



Petri Nets

- Properties of Petri nets: (1) $P \cap T = \emptyset$ (2) $P \cup T \neq \emptyset$
- Given a transition t ,
Preset(t) = $\{p: (p,t) \in F\}$, also called *input places* of t
Postset(t) = $\{p: (t,p) \in F\}$, also called *output places* of t
Similarly preset and postsets can be defined for places
- *Tokens* may be associated with places
- Each assignment of tokens to places defines a *marking* of the Petri net.
- A marking of Petri net defines a *state of the system*
- Initial state defined by an *initial marking*

Firing of Transitions

- A transition is **enabled** if each of its input places contains a number of tokens that is *greater than or equal to the weight of the arrow* connecting the input place to the transition
- An enabled transition may **fire**
- Firing of a transition T removes from each input place P_i the number of tokens equal to the weight of arrow from P_i to T and then inserts into each output place P_j the number of tokens equal to the weight of arrow from T to P_j

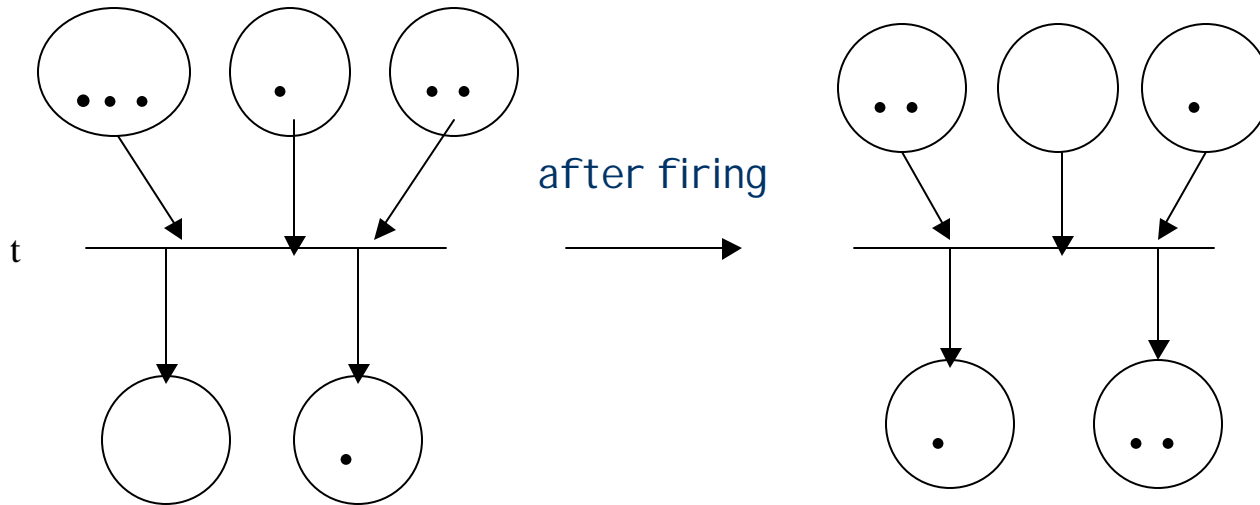
Semantics of Petri nets

Let the marking $M(p)$ of place p denote the number of tokens in place p .

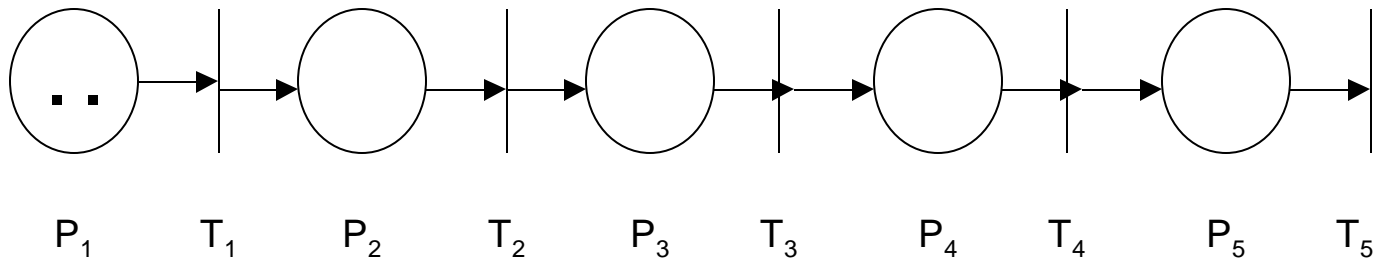
Transition t is enabled iff

- " $p \in t$'s input places, $M(p) \geq W(\langle p, t \rangle)$ t fires: produces a new marking M' in places that are either t 's input or output places or both
- if p is an input place: $M'(p) = M(p) - W(\langle p, t \rangle)$
- if p is an output place: $M'(p) = M(p) + W(\langle t, p \rangle)$
- if p is both an input and an output place:
$$M'(p) = M(p) - W(\langle p, t \rangle) + W(\langle t, p \rangle)$$

An Example



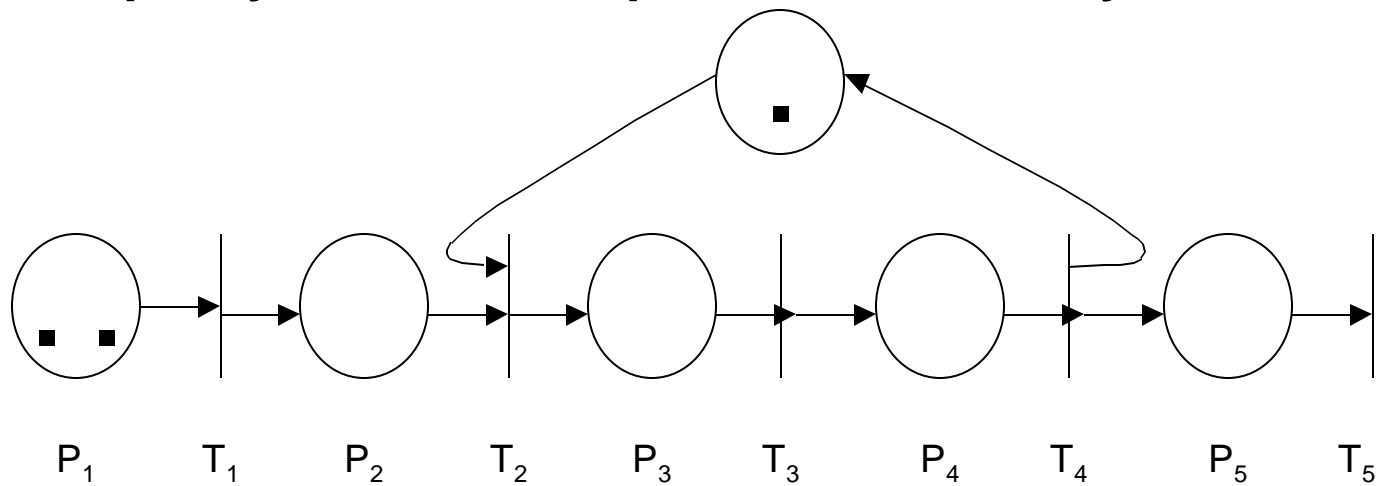
Warehouse Automation Example



Specifies requirements of a system

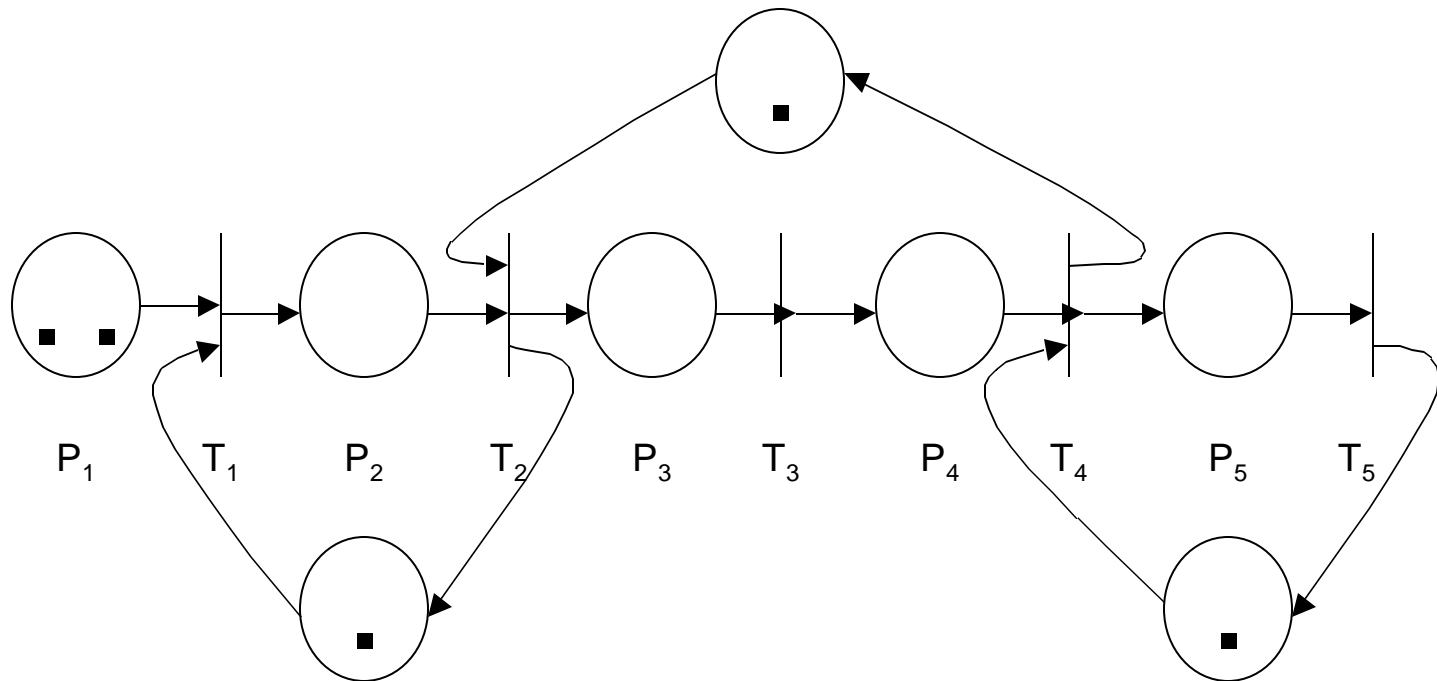
- **$P_1 \Rightarrow$ a truck arrives at the loading dock**
- **$P_2 \Rightarrow$ paperwork is processed and inventory checked**
- **$P_3 \Rightarrow$ conveyor belts move from the loading dock to the warehouse (transporting people, robots, forklifts, etc)**

- **P4 => conveyor belt move back from warehouse to loading dock with merchandise.**
- **P5 => goods loaded into the truck**
- **Concurrent handling of multiple trucks can be modelled by multiple tokens**
- **But a token is present in P3, then another token cannot be present in P4 (the conveyor belts cannot move in two opposite direction at the same time)**
- **Therefore, the above model can be modified as follows to specify the above requirements of the system.**



- **Now a token can enter P3 only if there is no token in P4**

- Similarly to specify the requirements that only paperwork of one truck can be processed at a time and merchandise of only one truck can be loading into the truck at a time, the above model can be modified as follows



In modeling a system using a Petri net,
usually,

- A *transition* represents an *event*.
- *Firing* of a transition represents *occurrence* of an event or an activity
- **Presence of a token in a place** represents **existence of a condition**

For example, a place may model a resource and existence of tokens in the place indicate the availability of the resource

Non-determinism

- Any of the enabled transitions may fire
- Model does not specify which fires, nor when it fires

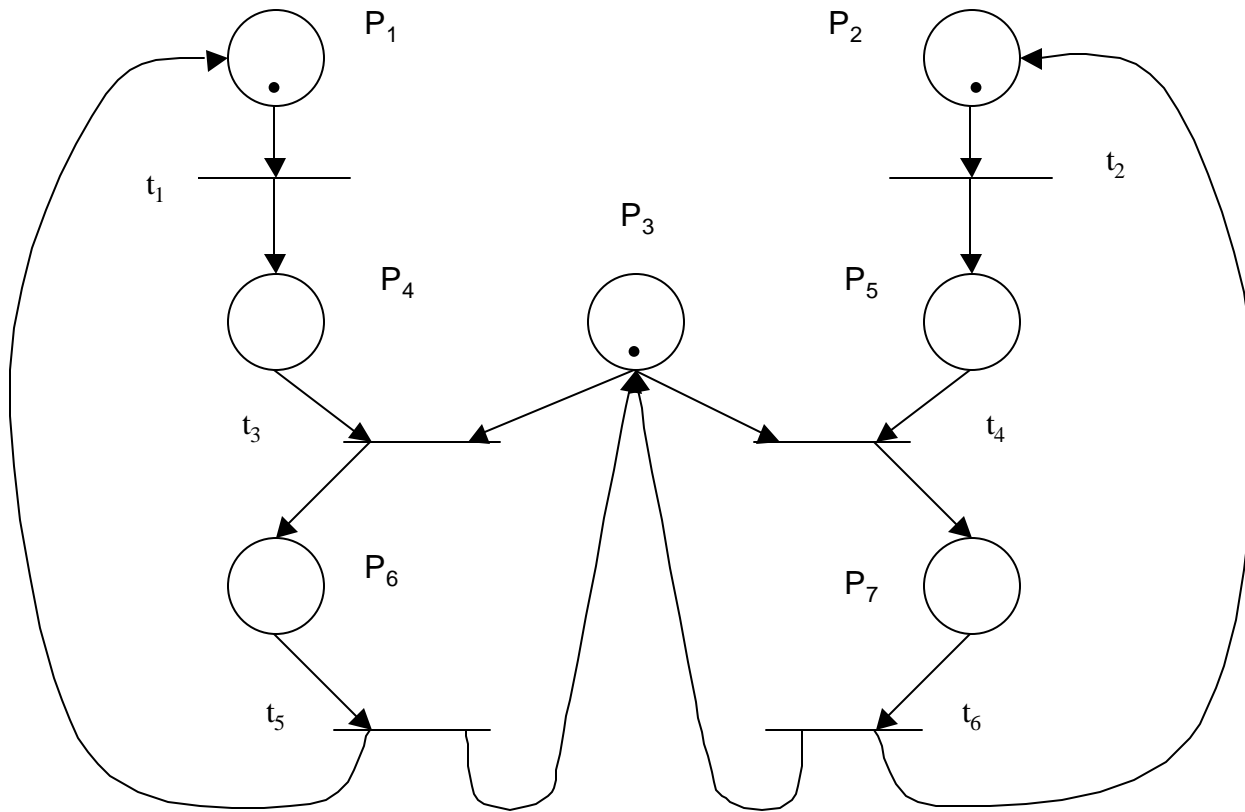


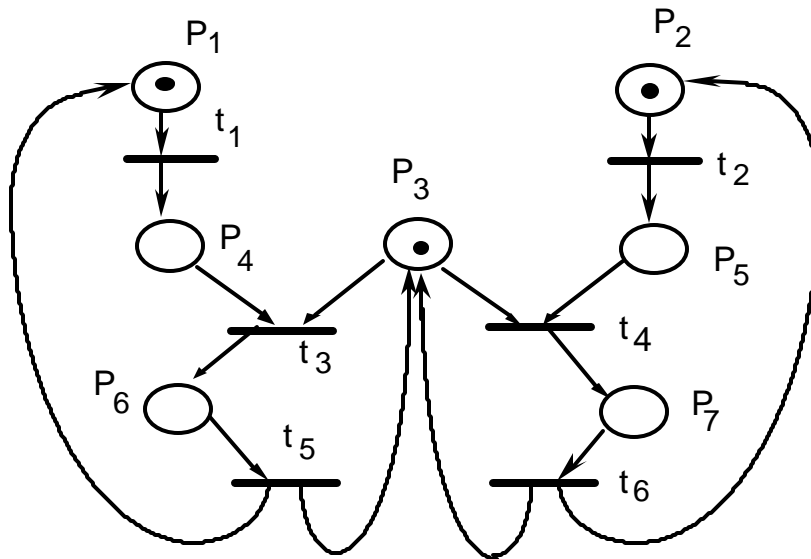
Figure 1

Some possible Firing sequences of transitions in this system:

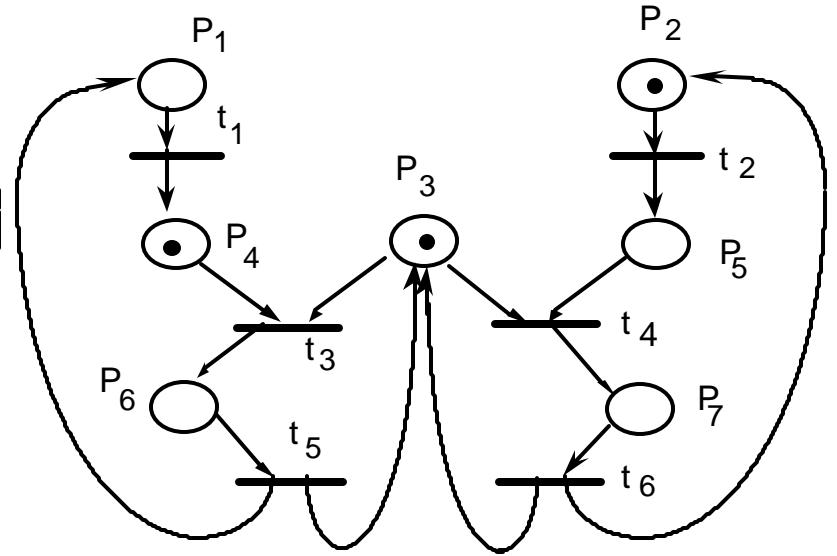
$t_1, t_2, t_3, t_5, t_1, t_4, t_6, t_2, \dots$

$t_1, t_3, t_5, t_1, t_3, t_5, t_1, t_3, t_5 \dots$

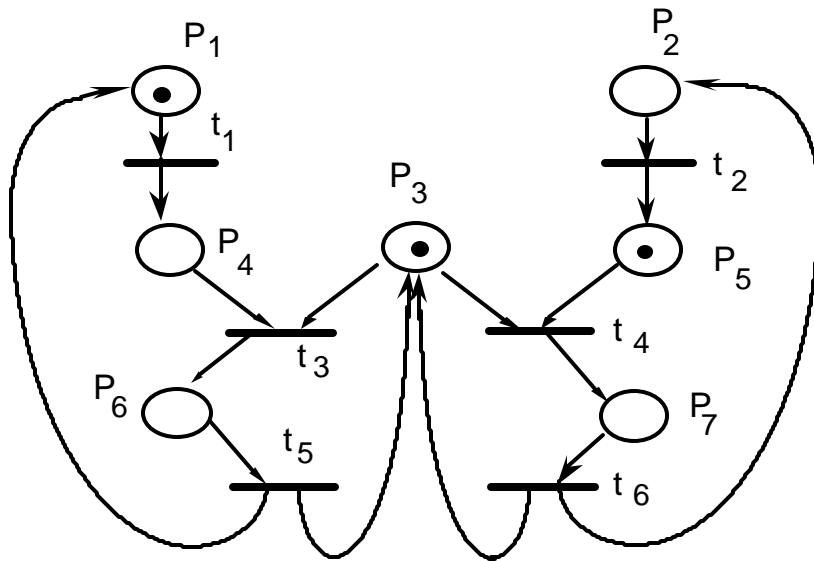
after (a) either (b) or (c) may occur, and then (d)



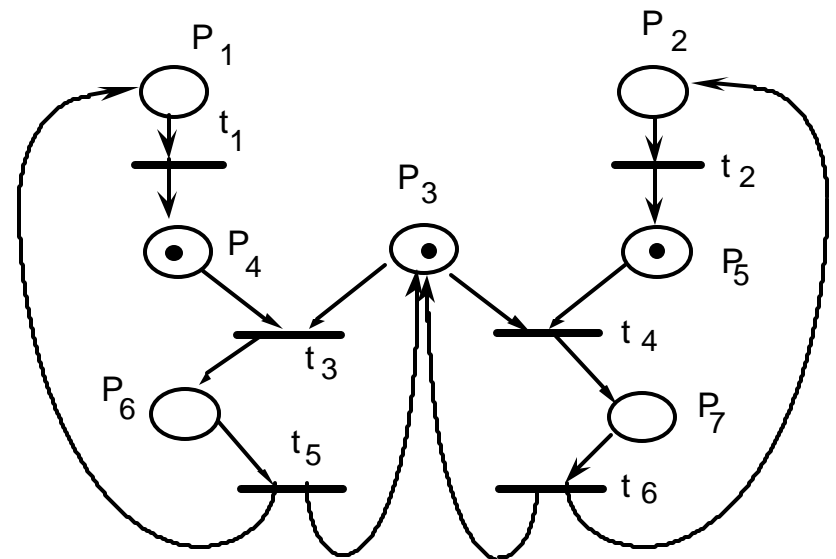
(a)



(b)



(c)

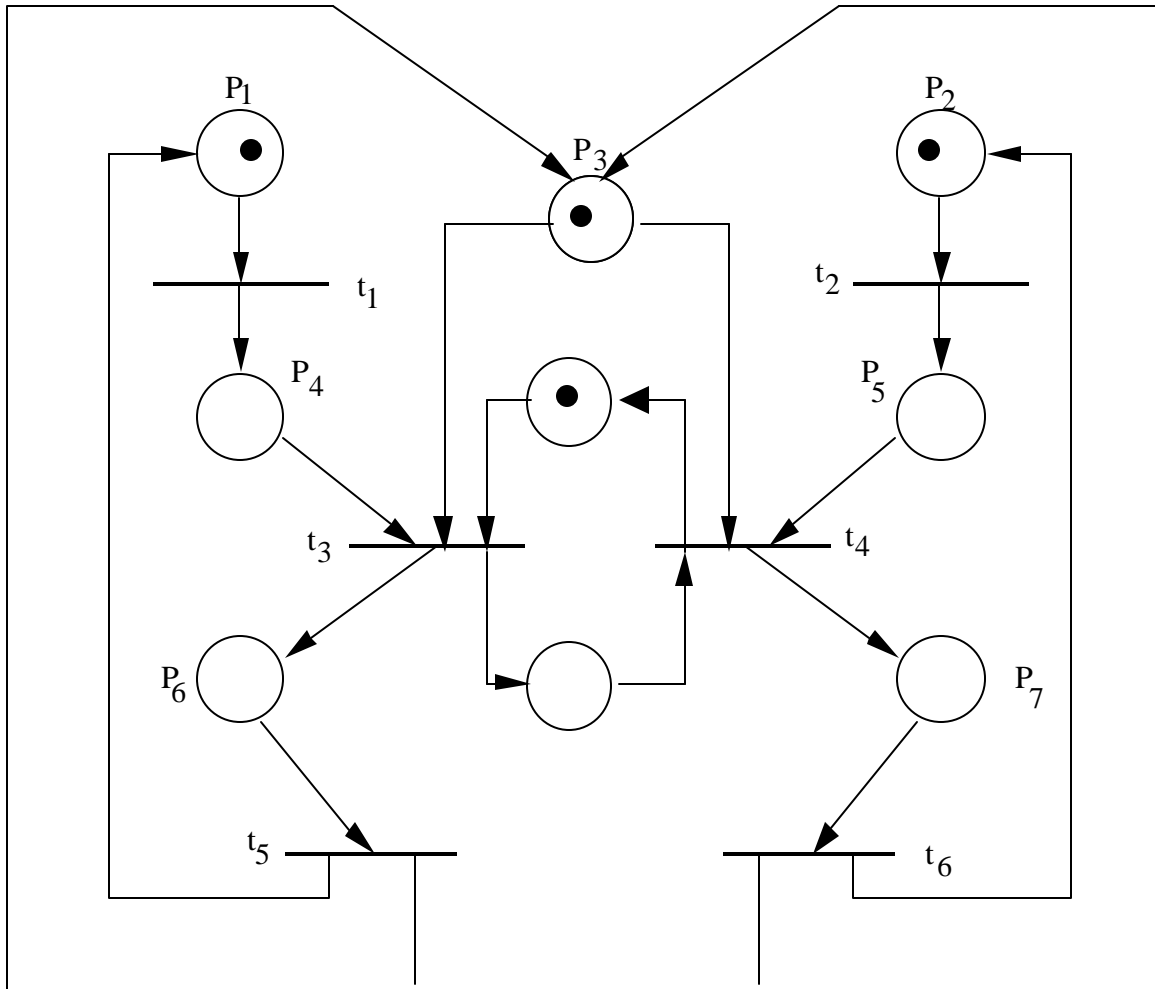


(d)

In Fig 1

- The activities corresponding to t_1 and t_2 can proceed asynchronously (e.g two students editing two different files)
- Firing of t_3 prevents t_4 from firing and vice versa. Thus t_3 and t_4 are in conflict.
- Model imposes no policy to resolve the conflict
- **Starvation (i.e., a process never gets a resource)** is possible in this example for the following firing sequences:
 $t_1, t_3, t_5, t_1, t_3, t_5, t_1, t_3, t_5, t_1, t_3, t_5, \dots$
 $t_2, t_4, t_6, t_2, t_4, t_6, t_2, t_4, t_6, t_2, t_4, t_6, \dots$
- If initial marking of P_3 contains 2 tokens, then both activities can proceed asynchronously. And no more conflict between t_3 and t_4 .

How to avoid starvation



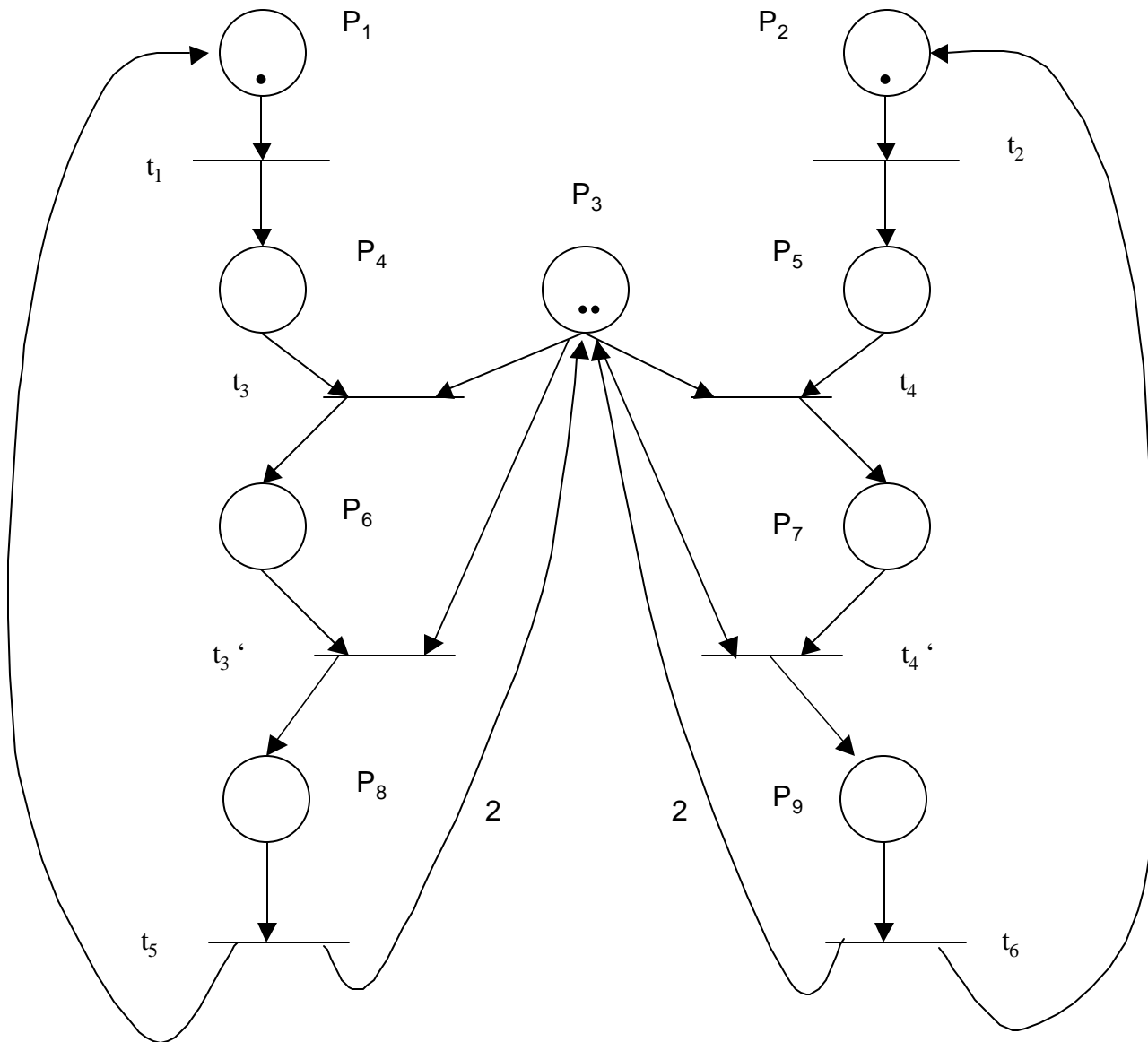
imposes alternation

Usually in a Petri Net

- **Firing of a transition represents occurrence of an event or an activity**
- **Presence of a token in a place represents existence of a condition**
- E.g. A place may model a resource and existence of tokens in the place indicated availability of the resource

In Fig 1

- The activities corresponding to t_1 and t_2 can proceed asynchronously (e.g two students editing two different files)
- Firing of t_3 prevents t_4 from firing and vice versa. Thus t_3 and t_4 are in conflict.
- Model imposes no policy to resolve the conflict
- Starvation possible in this example.
 $t_1, t_3, t_5, t_5, t_1, t_3, t_5, t_1, \dots$
 $t_2, t_4, t_6, t_2, t_4, t_6, t_2, t_4, \dots$
- If initial marking of P_3 contains 2 tokens, then both activities can proceed asynchronously. And no more conflict between t_3 and t_4 .



- In the firing sequence t_1, t_2, t_3, t_4 , Petri net reaches a state in which no transition is enabled i.e., a **Deadlock State**
- Deadlock state modeled well by Petri Nets

A deadlock-free net

