Finite state machines

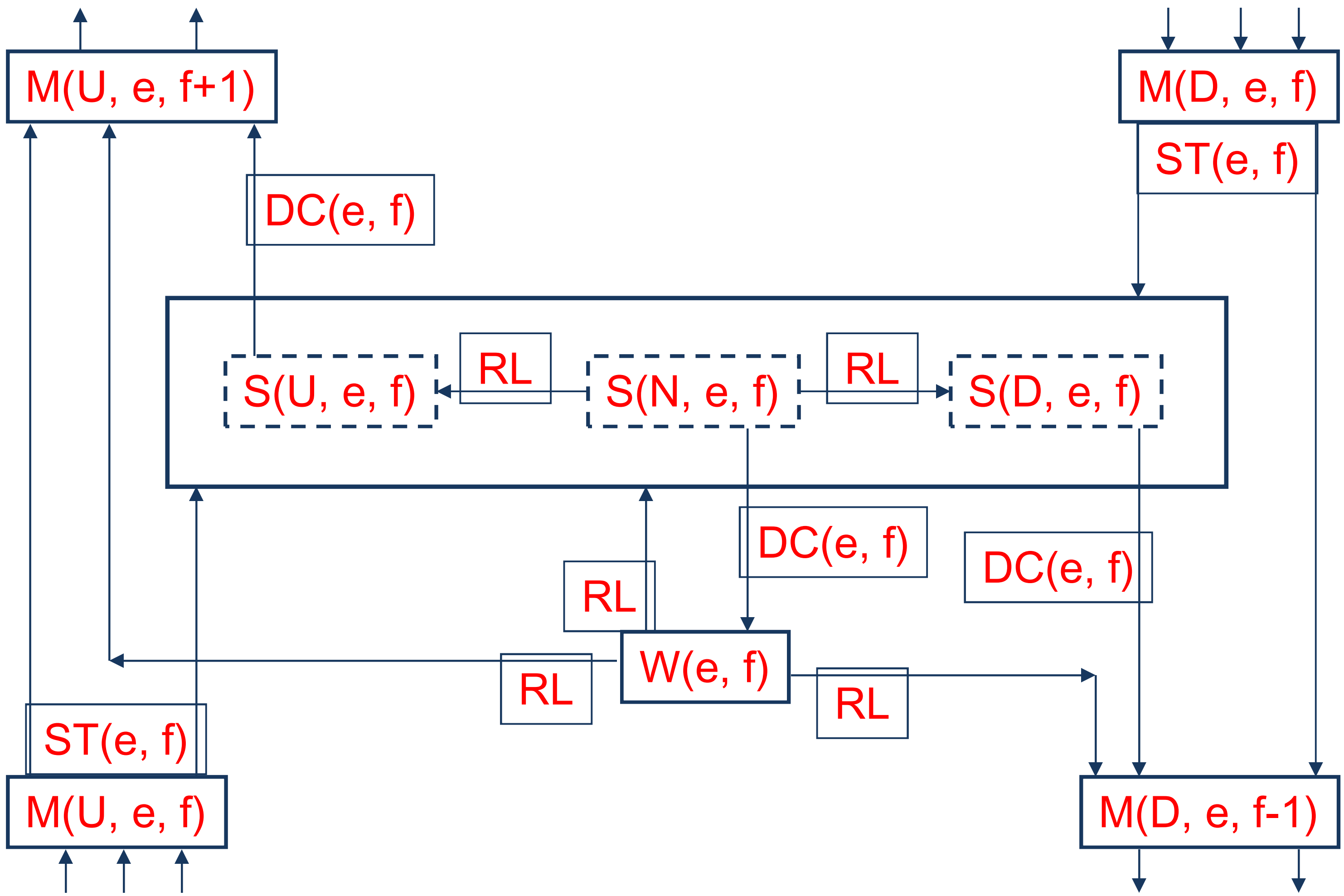
- Three elevator states can be defined:
  - M(d, e, f): Elevator e is Moving in direction d (floor f is next)
  - S(d, e, f): Elevator e is Stopped (d-bound) at floor f
  - W(e, f): Elevator e is Waiting at floor f (door closed)
- Other interesting events include:
  - DC(e, f): Door Closed for elevator e at floor f
  - ST(e, f): Sensor Triggered as elevator e nears floor f
  - RL: Request Logged (button pressed)
- Rules governing movement are:

$S(U, e, f) \wedge DC(e, f) \Rightarrow M(U, e, f+1)$

$S(D, e, f) \wedge DC(e, f) \Rightarrow M(D, e, f-1)$

$S(N, e, f) \wedge DC(e, f) \Rightarrow W(e, f)$

Finite state machines



Finite state machines

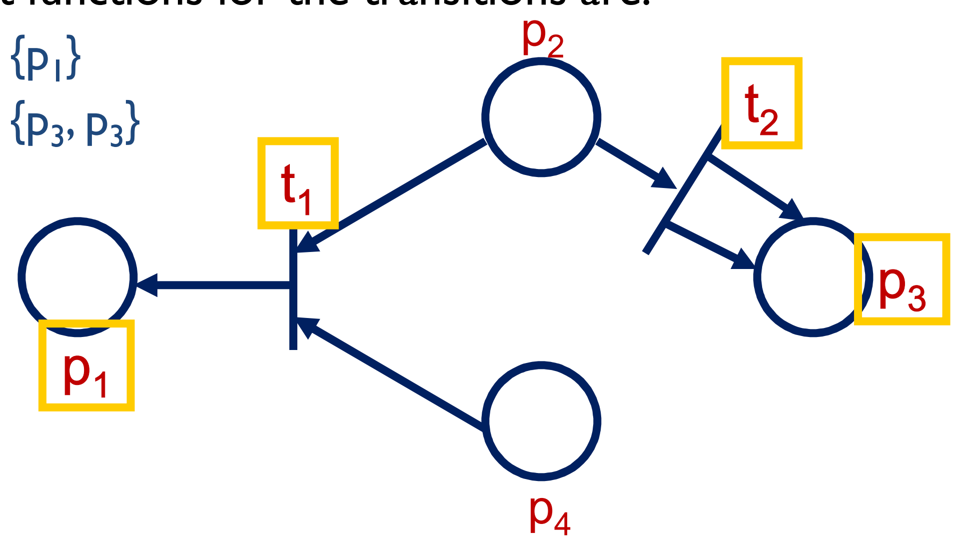
- The finite state machine is more precise than a natural language or graphical representation.
- It is almost as easy to understand
- It doesn't scale well to problems with many states.
- Finite state machines do not handles the concept of time. For this one has to use an extension of finite state machines called a statechart

Petri nets

- A major problem with specifying concurrent systems is dealing with timing.
- The problem manifests itself in many ways:
  - Synchronization
  - Race conditions
  - Deadlock
- Timing problems can be a result of poor design or implementation, which are in turn a result of poor specification.
- Petri nets are a powerful technique for sepcifying and designing systems with potential timing problems.
- Petri nets were invented by Carl Adam Petri in 1962.

Petri nets - introduction

- A Petri net consists of 4 parts: a set of places P; a set of transitions T; an input function I; and output function O.
- The set of places $P = \{p_1, p_2, p_3, p_4\}$
- The set of transitions $T = \{t_1, t_2\}$
- The input functions for the transitions are:
  - $I(t_1) = \{p_2, p_4\}$
  - $I(t_2) = \{p_2\}$
- The output functions for the transitions are:
  - $O(t_1) = \{p_1\}$
  - $O(t_2) = \{p_3, p_3\}$

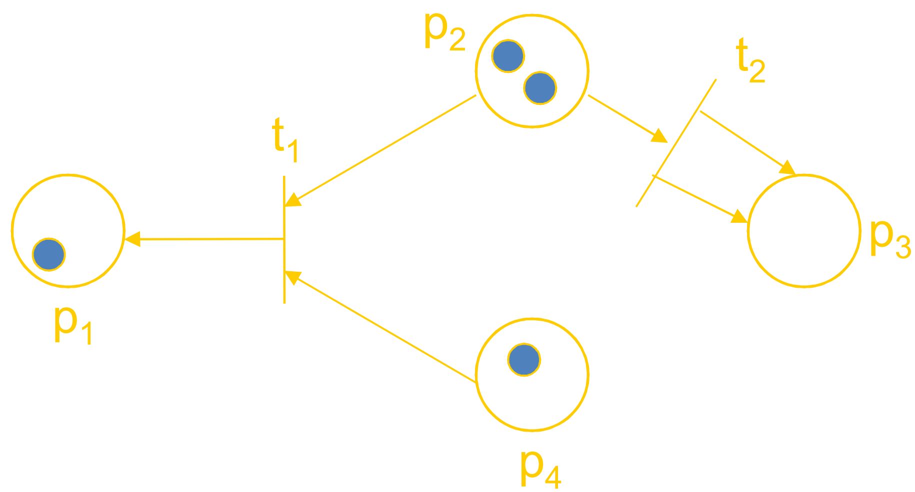


Petri nets - introduction

- More formally, a Petri net is a 4-tuple, $C = (P, T, I, O)$
- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places, $n \geq 0$,
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions, $m \geq 0$ P & T are disjoint.
- $I: T \rightarrow P^\infty$ is the input function, a mapping from transitions onto bags of places.
- $O: T \rightarrow P^\infty$ is the output function, a mapping from transitions onto bags of places.
- The marking of a Petri net is the assignment of tokens to that Petri net.

Standard finite state machine contain only a single current state. Whereas in Petri nets multiple locations, more or less comparable with states in a finite state machine, can contain one or more tokens.
A finite state machine is single threaded while a Petri net is concurrent.
In a finite state machine the active state changes in response to an event. In a Petri net transitions are executed as soon as all input locations contain at least one token.
A finite state machine can be considered as a special case of a Petri net.

Petri nets - introduction

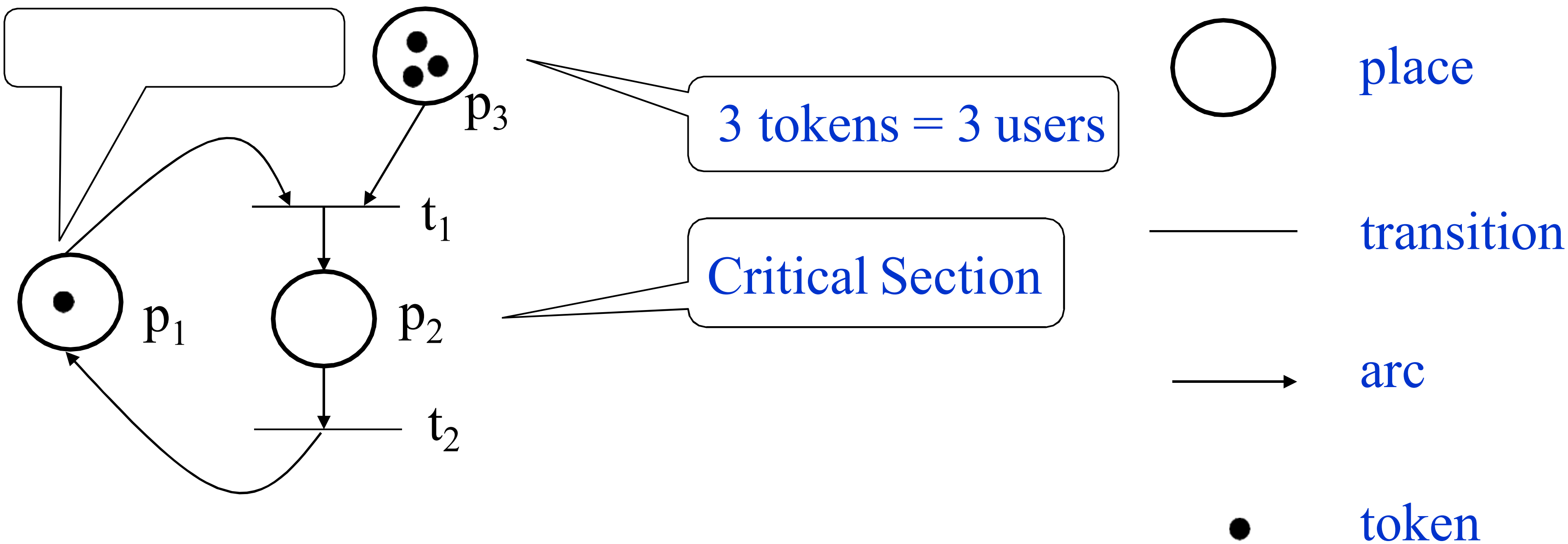


- The Petri net above contains 4 tokens.
- The marking can be represented by the vector (1, 2, 0, 1).
- Transition t_1 is enables (ready to fire) because there are tokens in p_2 and p_4.
- If the transition fires, 1 token is removed from p_2 and p_4, and 1 token is added to p_1.
- Note that the number of tokens if not conserved.

Example: Critical Section

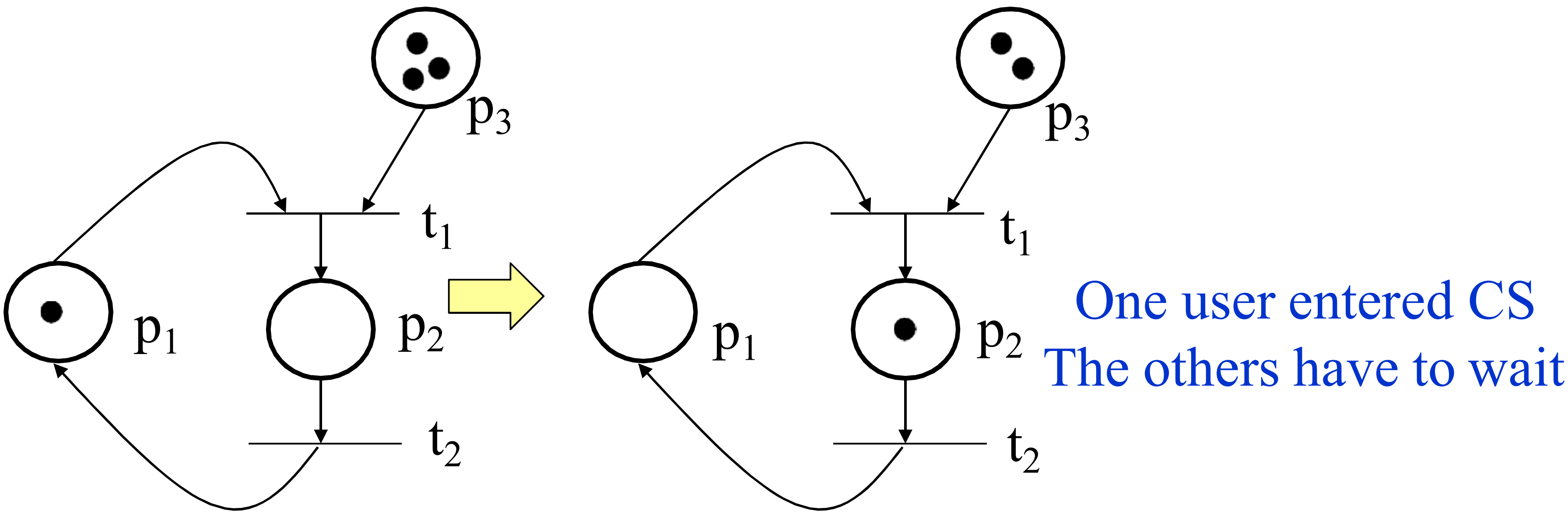
- 3 users try to access the same CS
- Only one user can access CS each time

- 1 token = lock



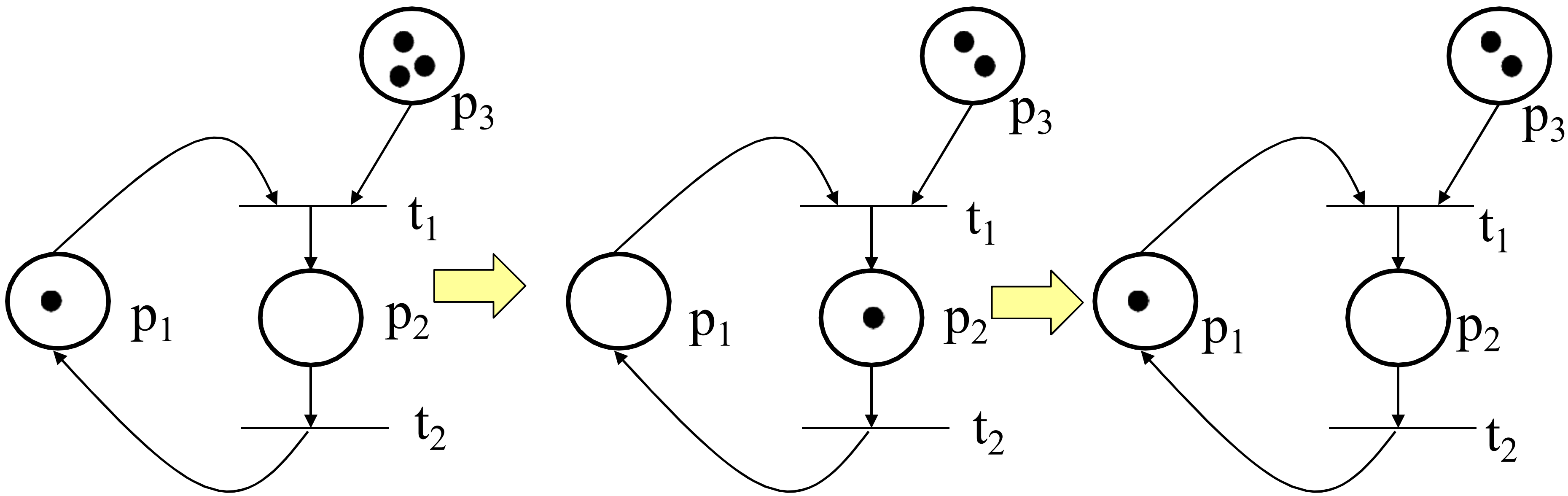
Example: Critical Section

- Multiple users try to access the same CS
- Only one user can access CS each time

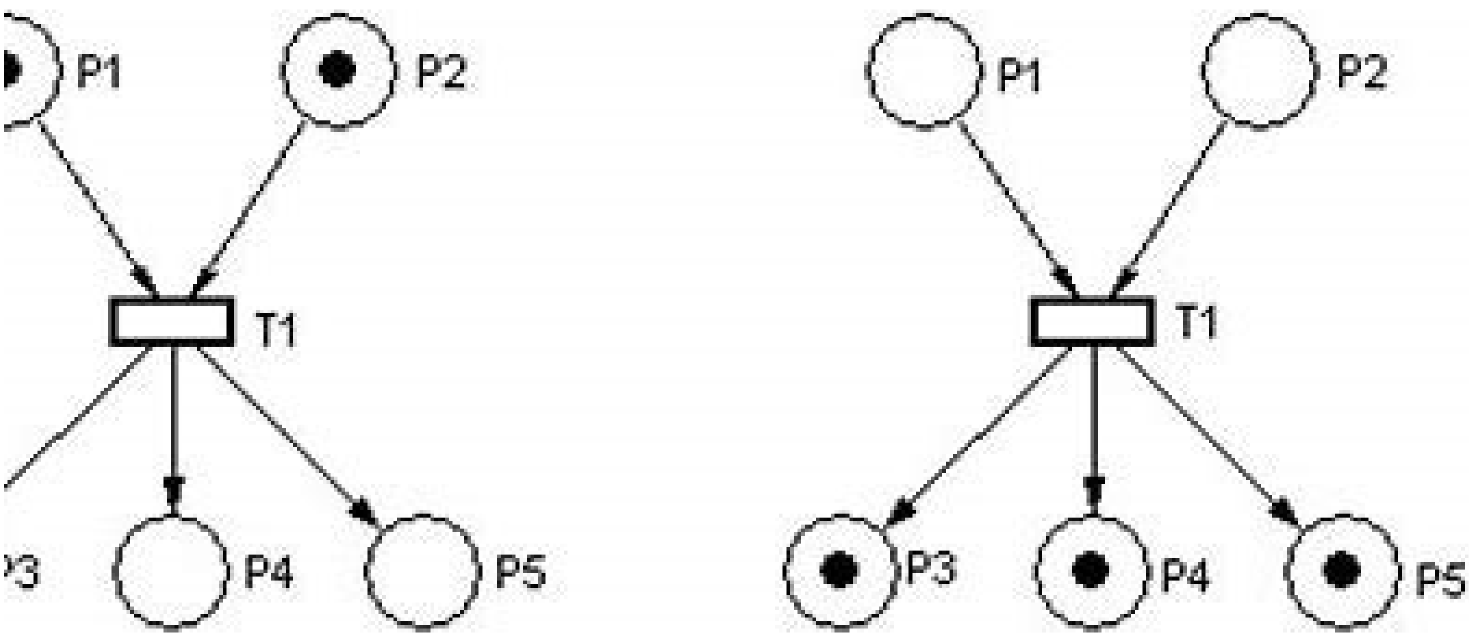


Example: Critical Section

- Multiple users try to access the same CS
- Only one user can access CS each time



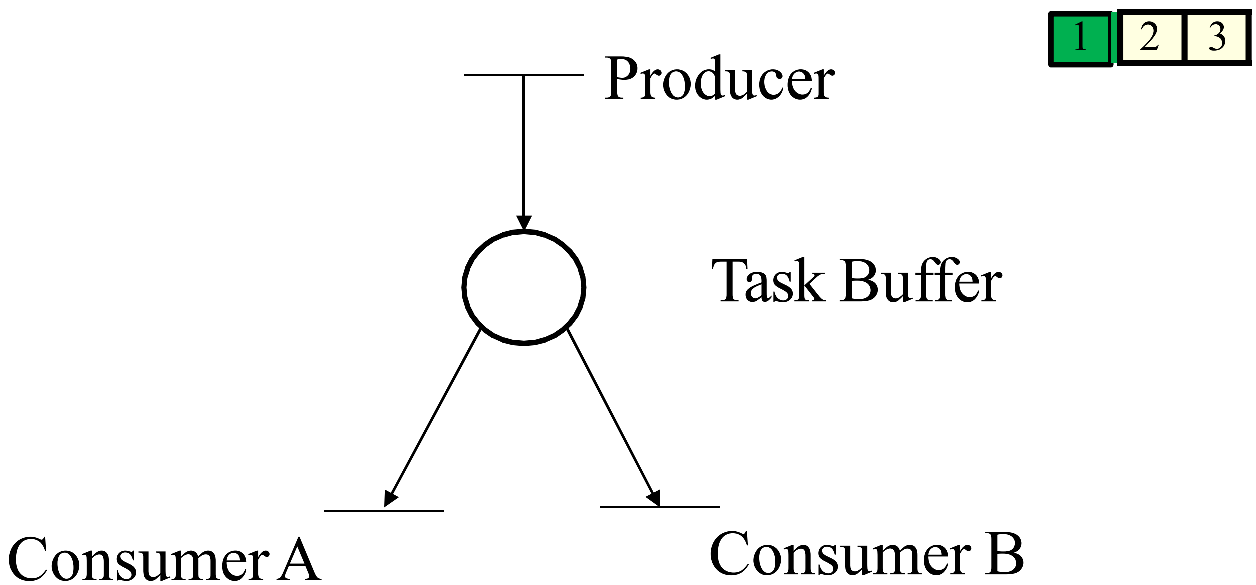
One user completes access.
Another one can enter now.



Enabled.ready to file

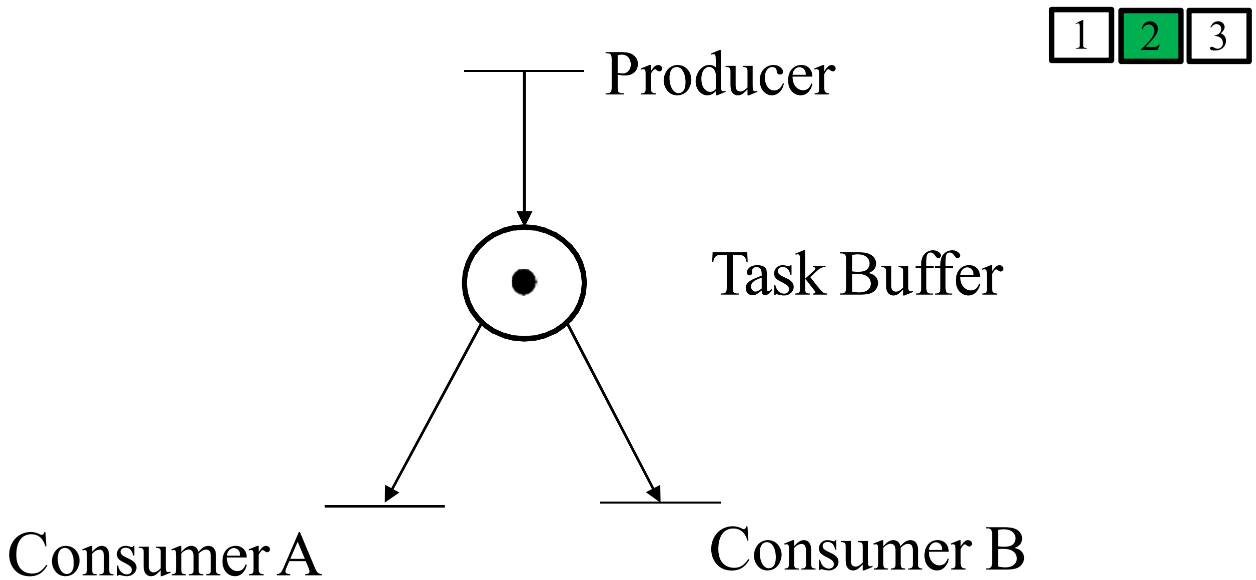
Firing complete

Example: Producer-Consumer



Producer produces tasks and put the tasks in the task buffer.
Consumers take tasks from the task buffer and execute them.
One task can only be executed by one consumer – either A or B.

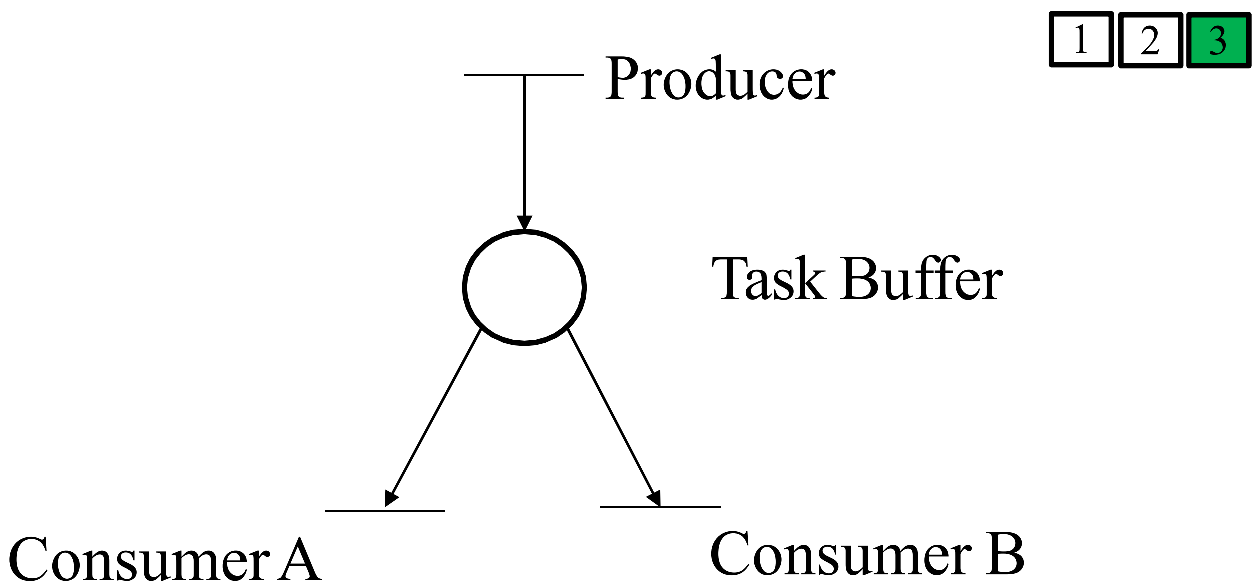
Example: Producer-Consumer



Producer produced one task.

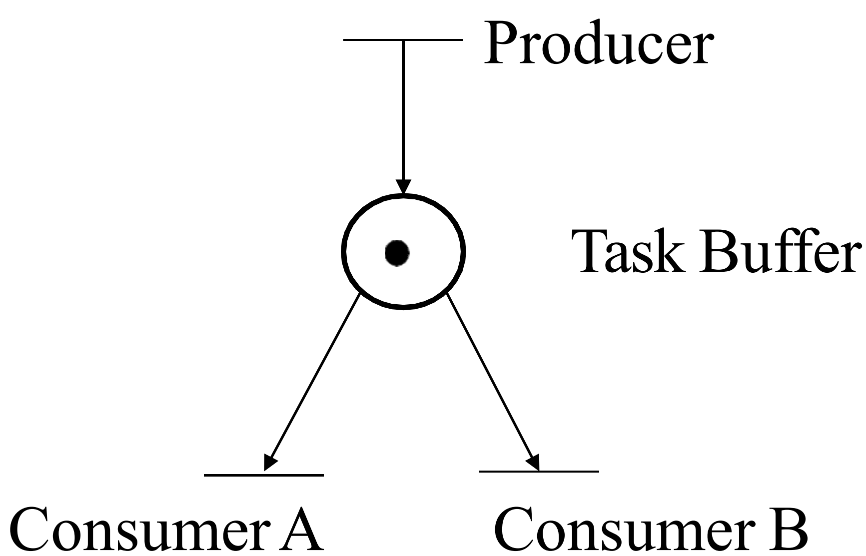
Either A or B can be fired. But only one will be fired!

Example: Producer-Consumer



After firing A or B, the task is executed.

Example: Producer-Consumer



Petri Nets

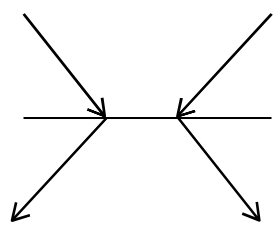
What will happen
after token pushing?

Dataflow (similar structure but different semantics)

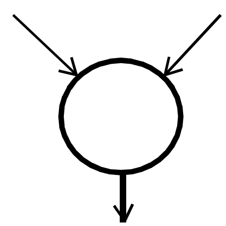
What's the difference between Petri Nets and Dataflow?

Definition of a Petri Net (Self-reading)

- A Petri Net is a bipartite graph (P, T, A) that comprises of
  - A set of transitions: T
  - A set of places: P
  - A set of directed arcs: A



a transition



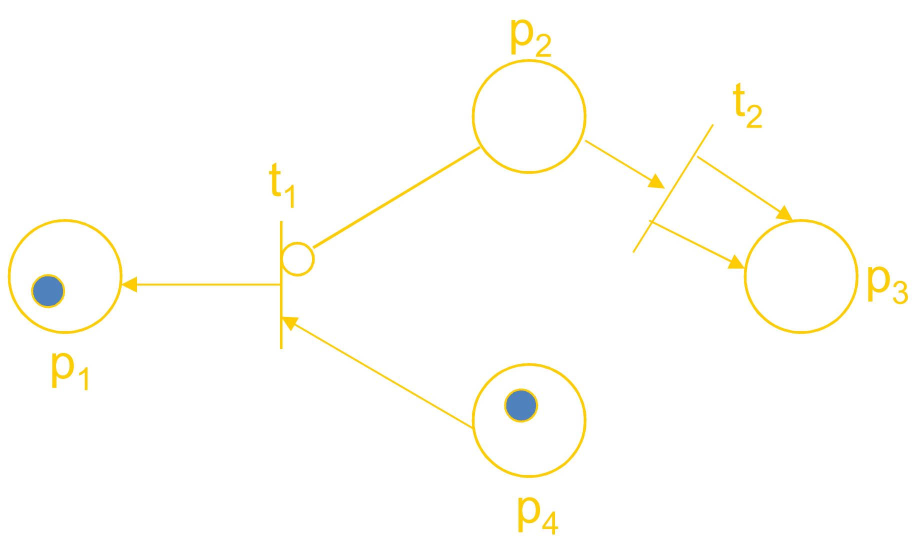
a place

Petri nets - introduction

- Petri nets are nondeterministic. If more than 1 transition could fire, one is selected arbitrarily.
- In the previous slide, both t_1 and t_2 are enabled. If t_1 fires, the resultant marking becomes (2, 1, 0, 0).
- Formally a marking M of a Petri net C = (P,T, I, O), is a function from the set of places P to te set of non-negative integers:
  - $M: P \rightarrow (0, 1, 2, \ldots)$
- A marked Petri net is then a 5-tuple (P,T, I, O, M).

Petri net - introduction

- An extension to a Petri net is an inhibitor arc.
- A transition is enabled if there is a token on each of its (normal) input arcs, and no token on any inhibitor input arcs.
- Transition t_1 is enabled below.

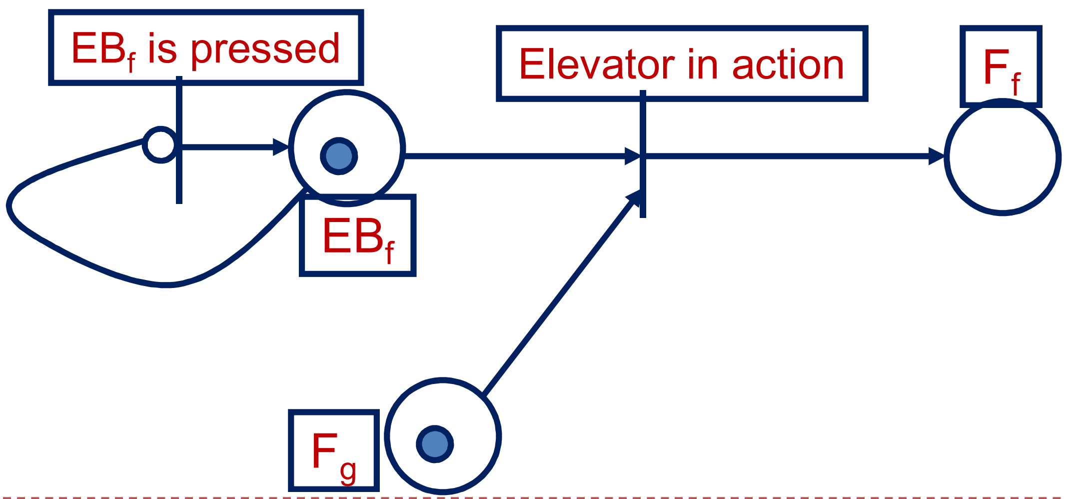


Petri nets – elevator problem

- There are n elevators installed in a building with m floors.
- Each floor is represented by a place, F_f, $1 \leq f \leq m$.
- An elevator is represented by a token.
- A token in F_f denotes that an elevator is at floor f.
- Constraint
  - Each elevator has a set of m buttons – one for each floor.
  - These are illuminated when pressed, cause the elevator to travel to that floor, and turned off when the elevator arrives at that floor.
  - Additional places are needed to model this.
- The elevator button for floor f in elevator e is denoted $EB_{f,e}$ with $1 \leq f \leq m$, $1 \leq e \leq n$
- For the sake of simplicity we suppress the subscript e denoting the elevator.

Petri nets – elevator problem

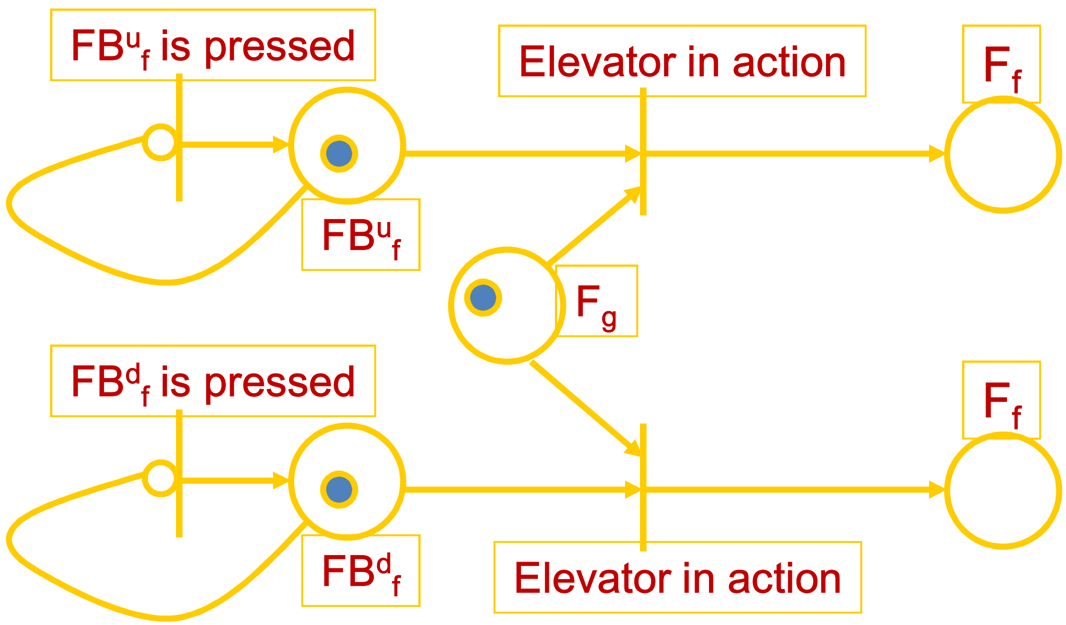
- A token in EB_f denotes that the elevator button for floor f is illuminated.
- Assume that EB_f is not initially illuminated.
- Note that the button is illuminated the first time it is pressed and subsequent presses do nothing (until elevator arrives that the correct floor).
- Assume that the elevator is currently at floor g.



Petri nets – elevator problem

- Each floor has two buttons (except the bottom and top floors) to represent an up-elevator and a down-elevator. These buttons illuminate when pressed, and are cancelled when an elevator traveling in the right direction arrives at the floor.
- The floor buttons are represented by places FB^d_f and Fb^u_f
- The situation when an elevator reaches floor f from floor g with one or both buttons illuminated is given by:

Petri nets – elevator problem



Petri nets – elevator problem

- When an elevator has no requests in needs to remain on the same floor.
- This is easily achieved,
- If there are no requests, no “Elevator in action” is enabled.

Petri nets - properties

- Petri nets have some additional useful properties.
- It is possible to prove that the system has no deadlocks or livelocks.
- They can be used to perform a proof of correctness.
- The system is simple and can be implemented as part of a tool.

Z

- Z is a popular formal specification language.
- Use of Z requires knowledge of set theory, functions and discrete mathematics, including first-order logic.
- Like the earlier formal systems examined, there are several variants of Z.