# Petri Nets (2)

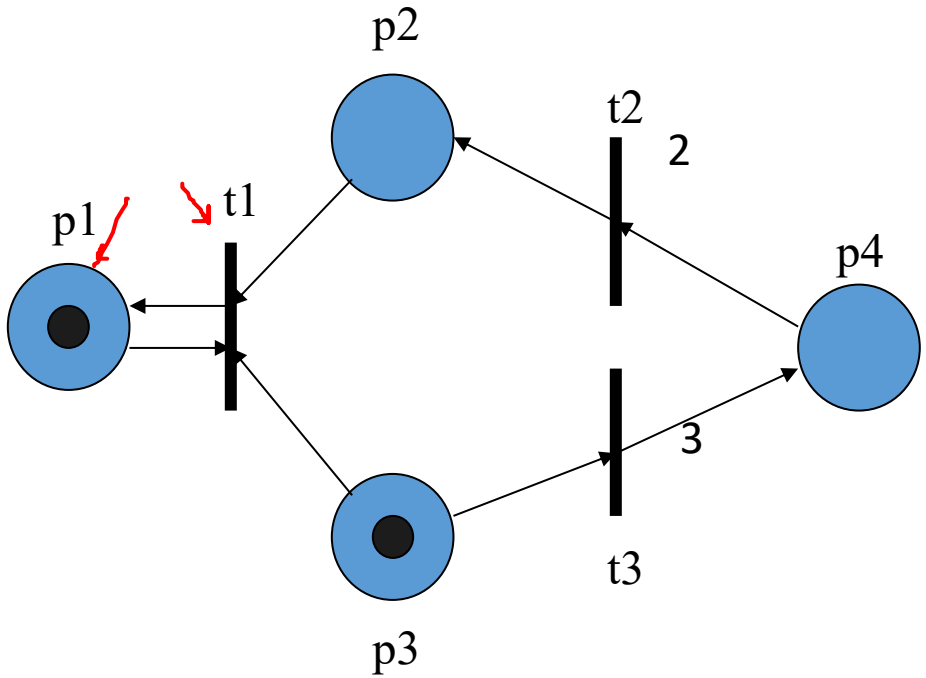
Refreshment , Properties & Examples

# Agenda

- Refreshment
- Annotations
- Properties
- Examples
- Behavioral Analysis

# Refreshment

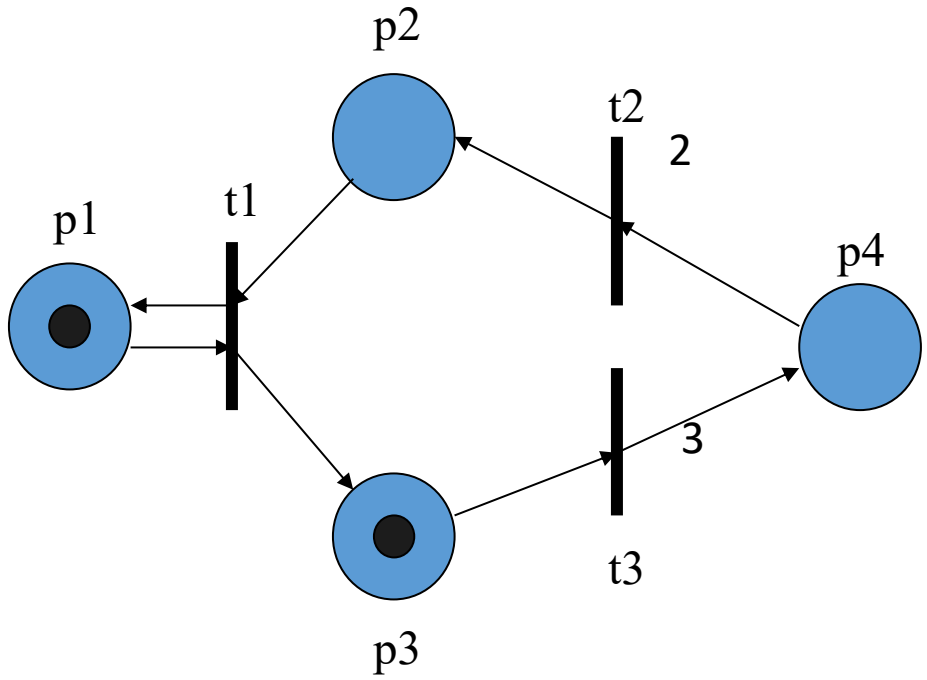
- Computational Model Introduced by Carl Adam Petri in 1962
- Widely applied in distributed computing , embedded control and networks (data and transportation).
- Describes explicitly and graphically the sequencing , causality , conflicts , concurrency , mutual exclusion ..for a control flow scenario.
- Asynchronous Model that has No Hierarchy.
- Bipartite weighted directed graph consisting of places and transitions connected with weighted arcs. Tokens are black dots
- Places (states/conditions) : Circles , Transitions (actions/events) : Bold Bars/Boxes, Arcs : flow relation weighted arrows and Tokens : Black dots or digits.



# Petri Net Marking/Annotation/Language

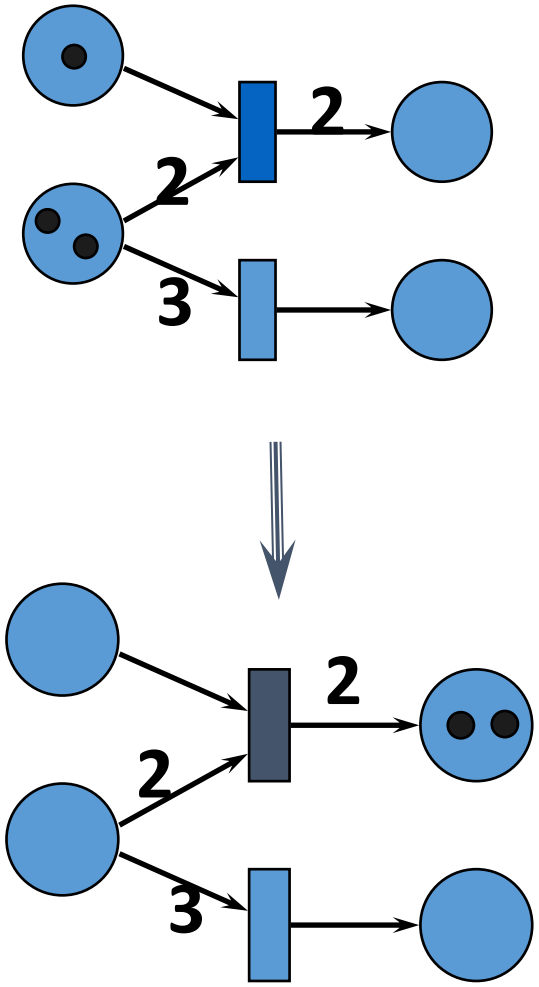
- PN ($N, M_0$)
  - N : Network composed of places P , transitions T , flow relations F and W weighted values of relations.
  - M is an n-vector (m1,m2,m3,…mi) where i is the total number of places , m is the number of tokens in place pi
  - M0 is the initial marking of all i places.
  - Example M0 :{1,0,1,0}
  - P {p1,p2,p3,p4}
  - T{t1,t2,t3}
  - F{p1-t1,t1-p1,p2-t1,t1-p3,t2-p2,p3-t3,t3-p4,p4-t2}
  - W{1,1,1,1,1,1,3,2}
  - M0 :{1,0,1,0},M1 :{1,0,0,3} ,M2{1,1,0,1},M3{1,0,1,1},

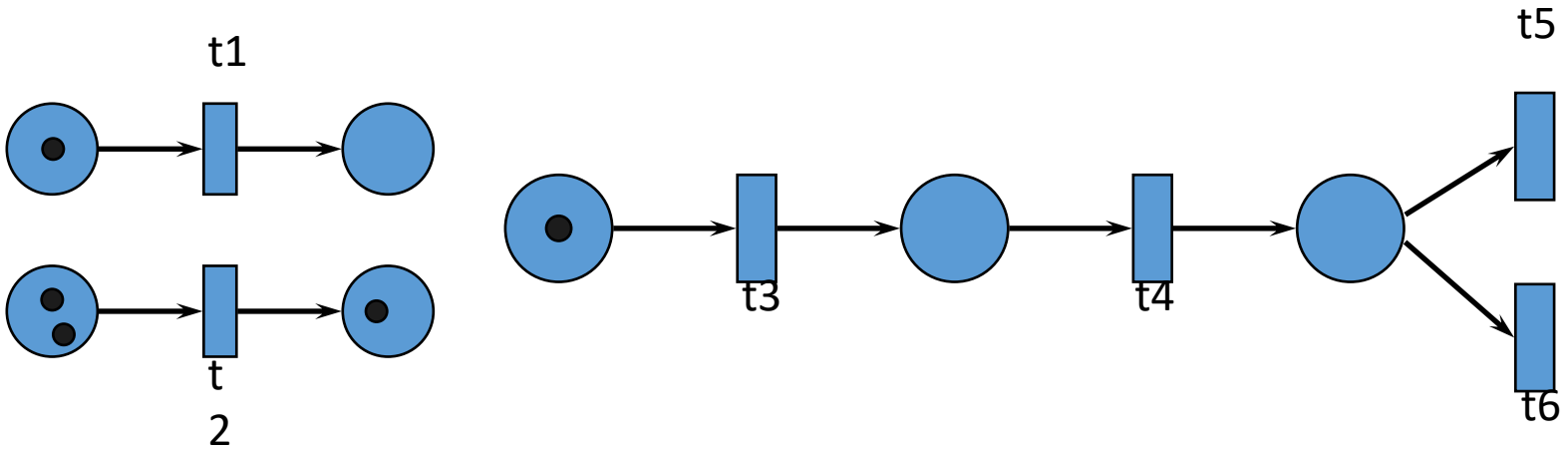
M4{1,0,0,4},M5{1,1,0,2},M6{1,1,1,0},M7{1,0,1,3}….



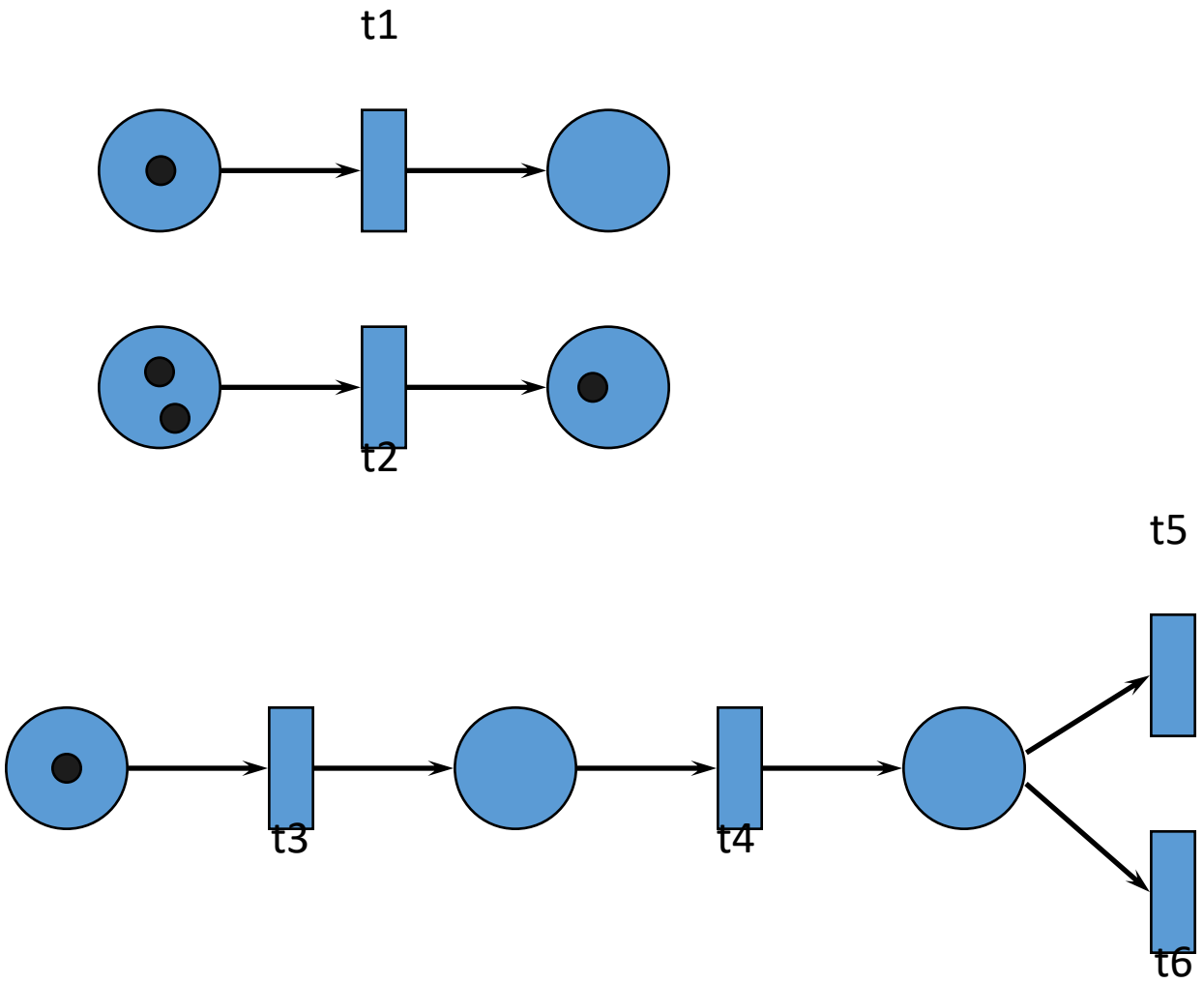
# Petri Net Firing

- Enabled Transitions are those with all input conditions satisfied ie : has enough tokens for the weighted arcs to allow relation flow
- Enabled Transitions may or may not fire ( according to stochastic probability models or deterministic assumptions)
- Concurrency , Causality , Choice properties of Petri Nets.

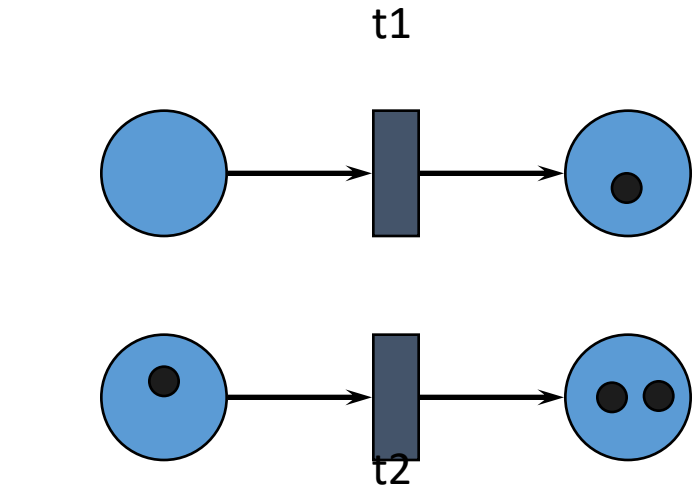


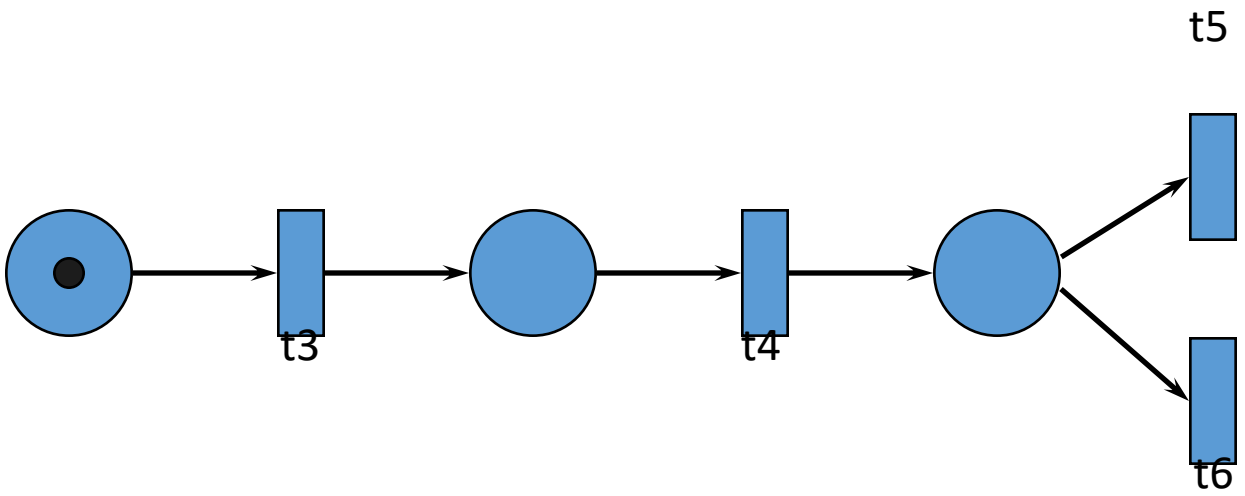
# Petri Nets Concurrency , Causality and Conflict



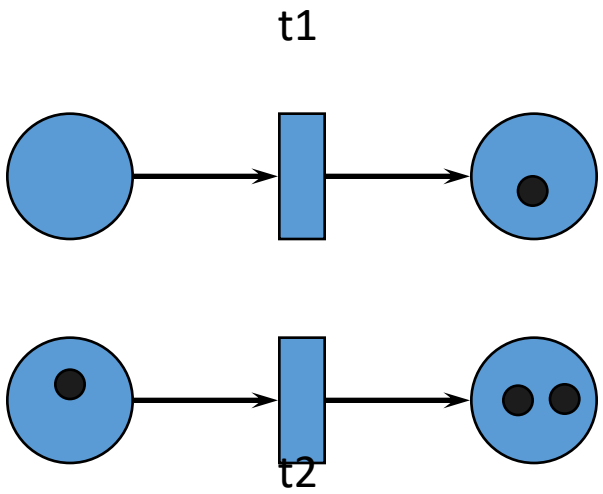
# Petri Nets Concurrency , Causality and Conflict

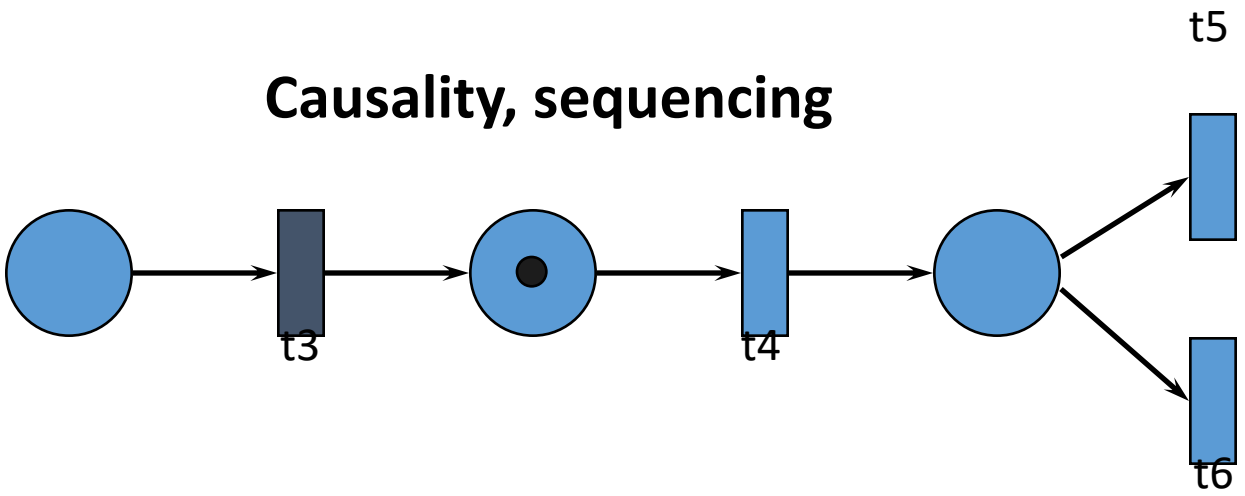



**Concurrency**

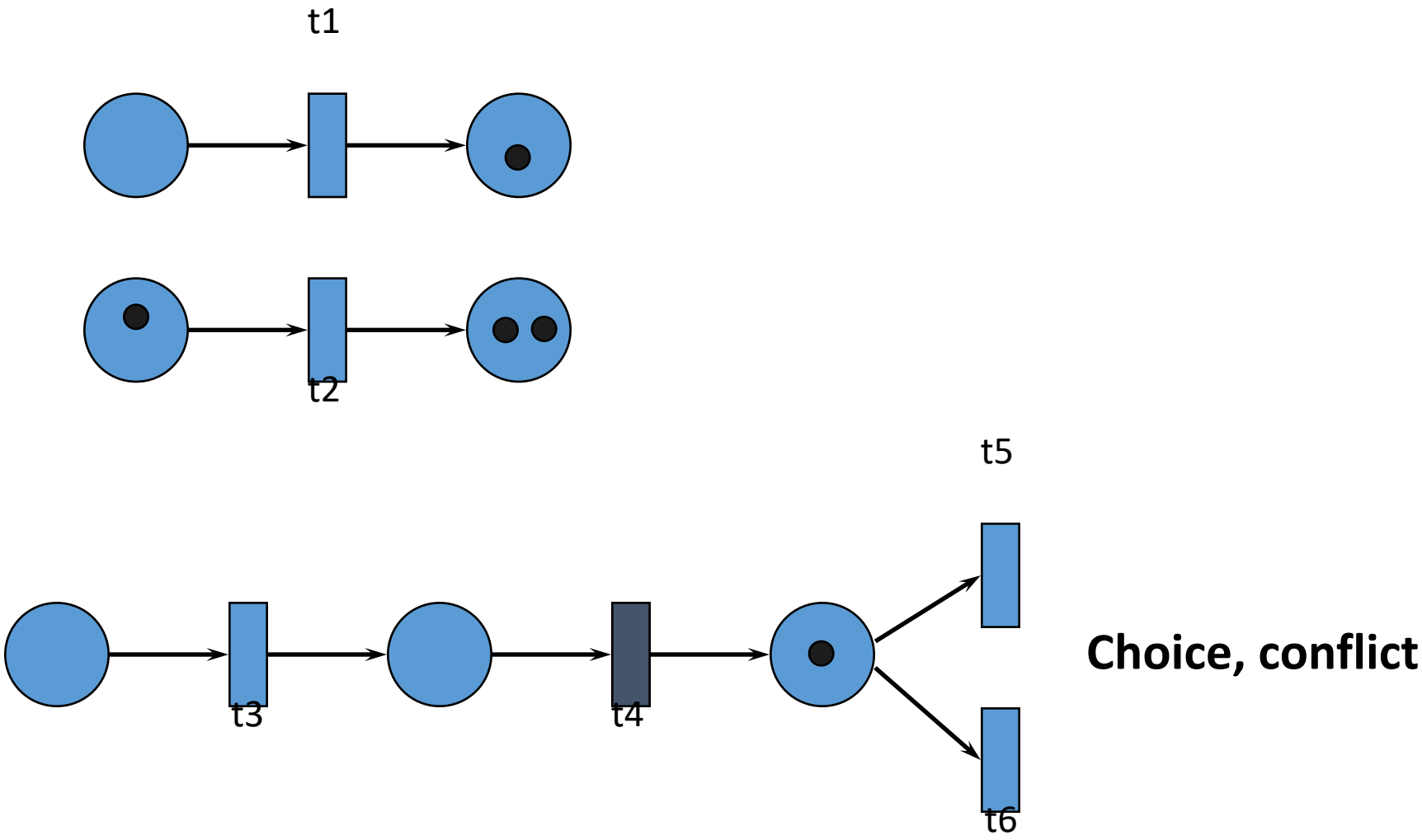


# Petri Nets Concurrency , Causality and Conflict

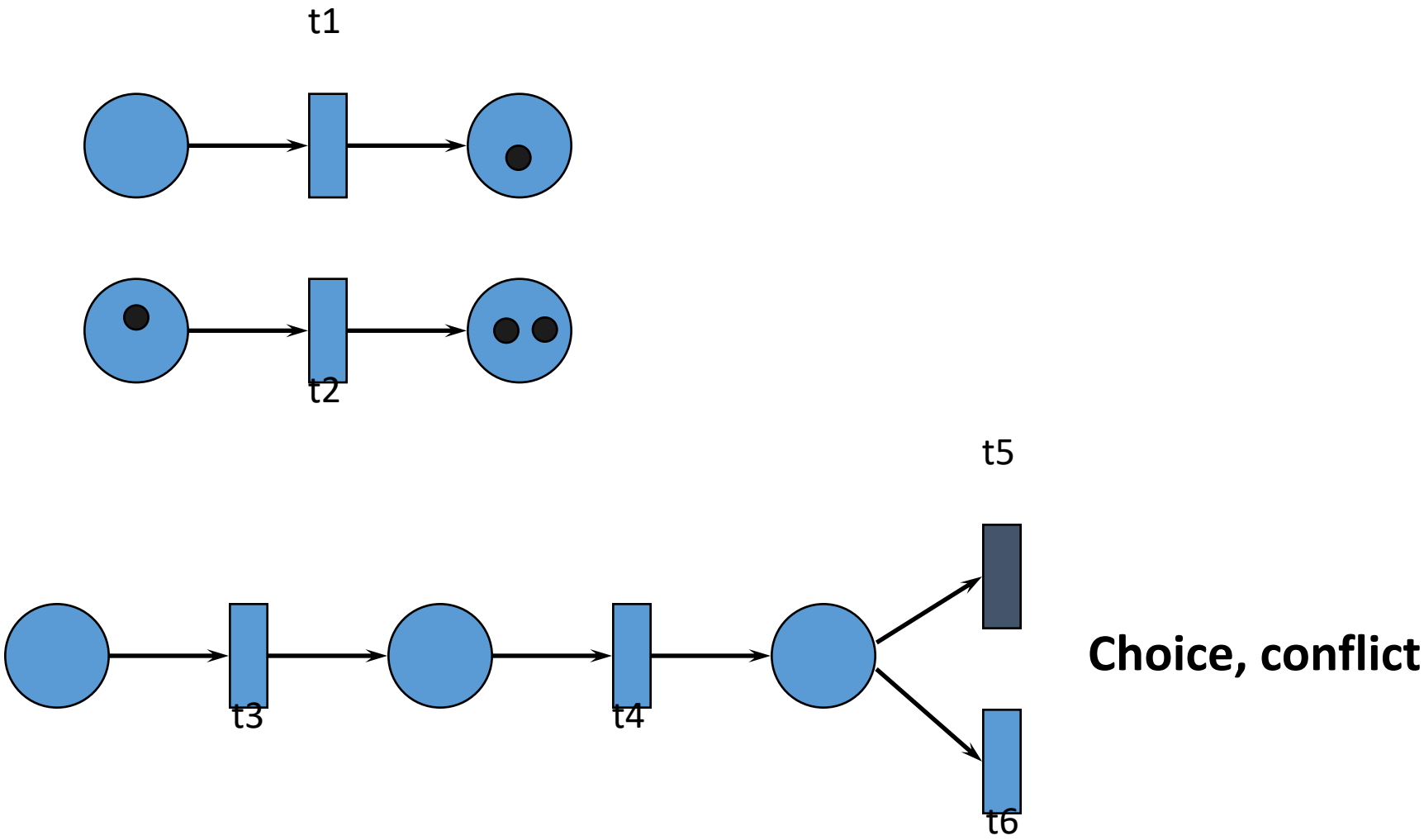


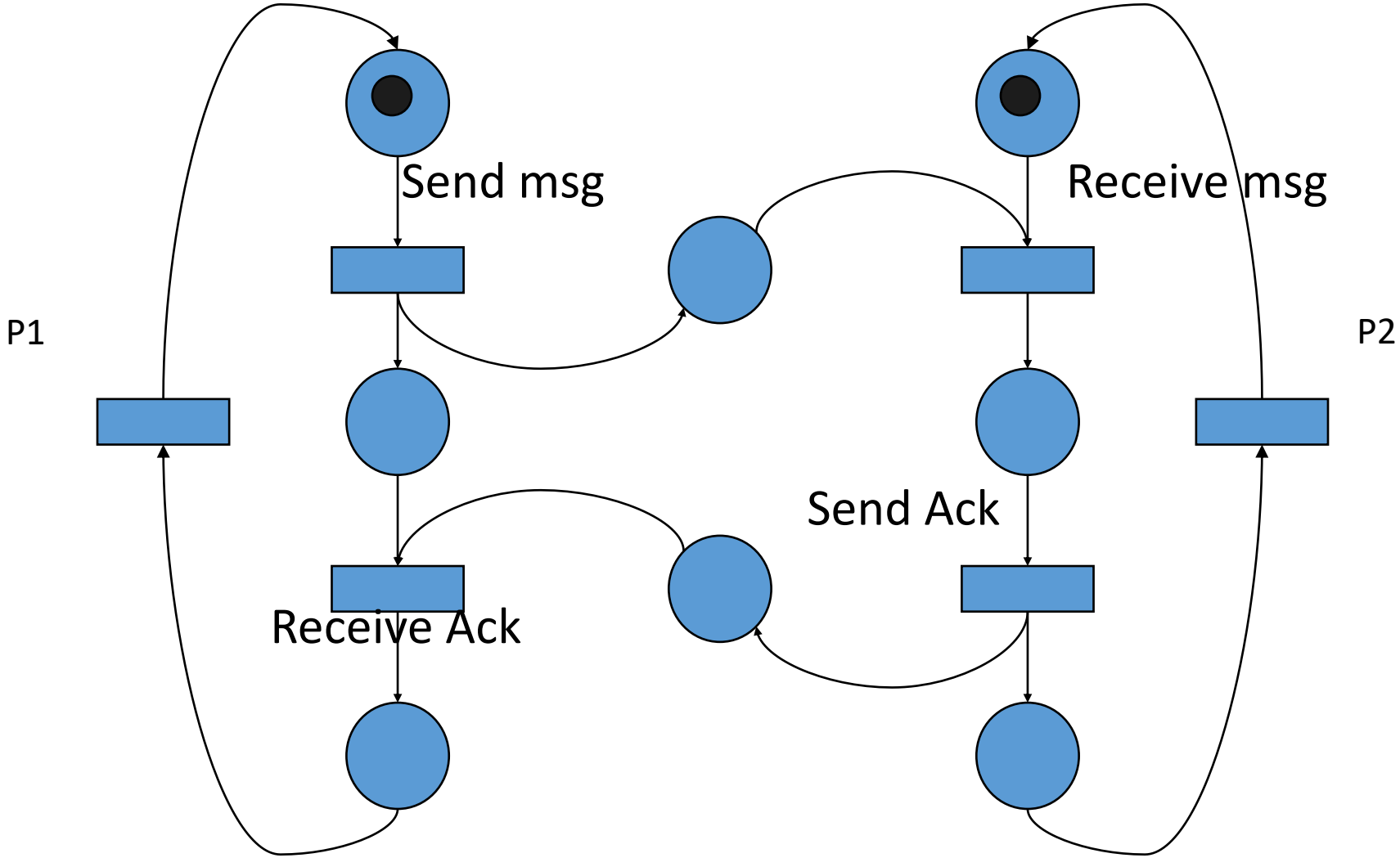
# Petri Nets Concurrency , Causality and Conflict



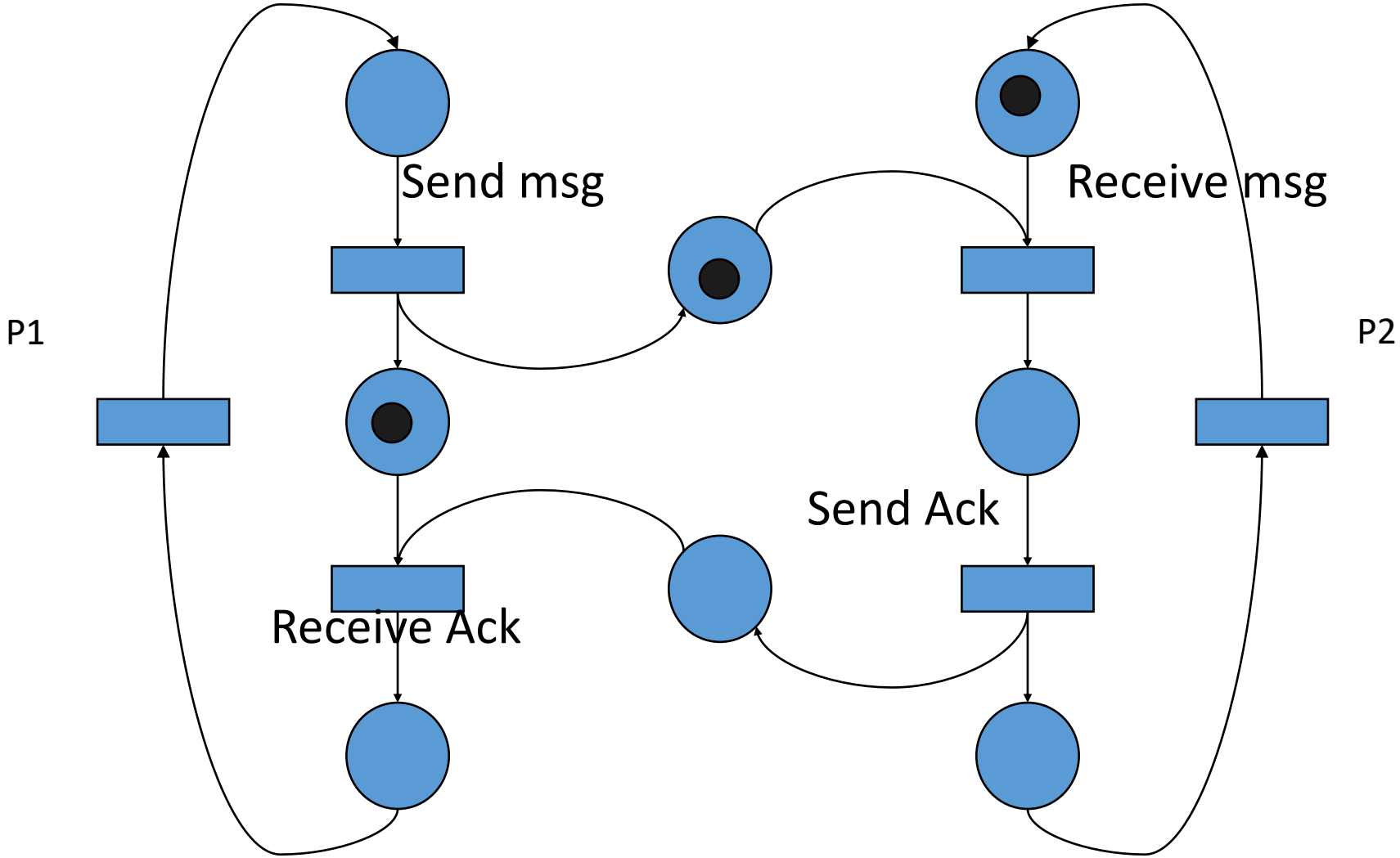
# Petri Nets Concurrency , Causality and Conflict



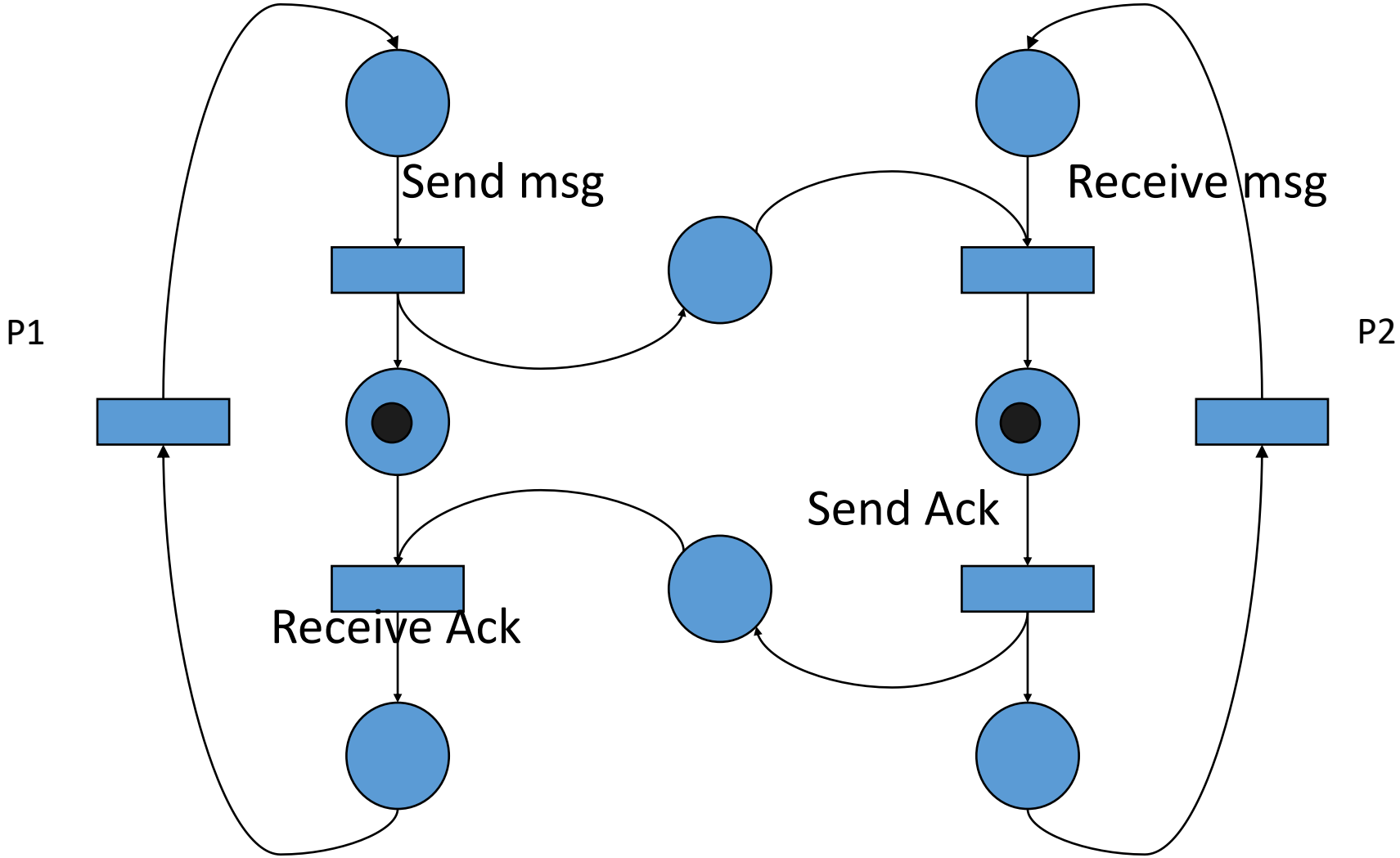
# Petri Nets Example Sender Receiver Communication



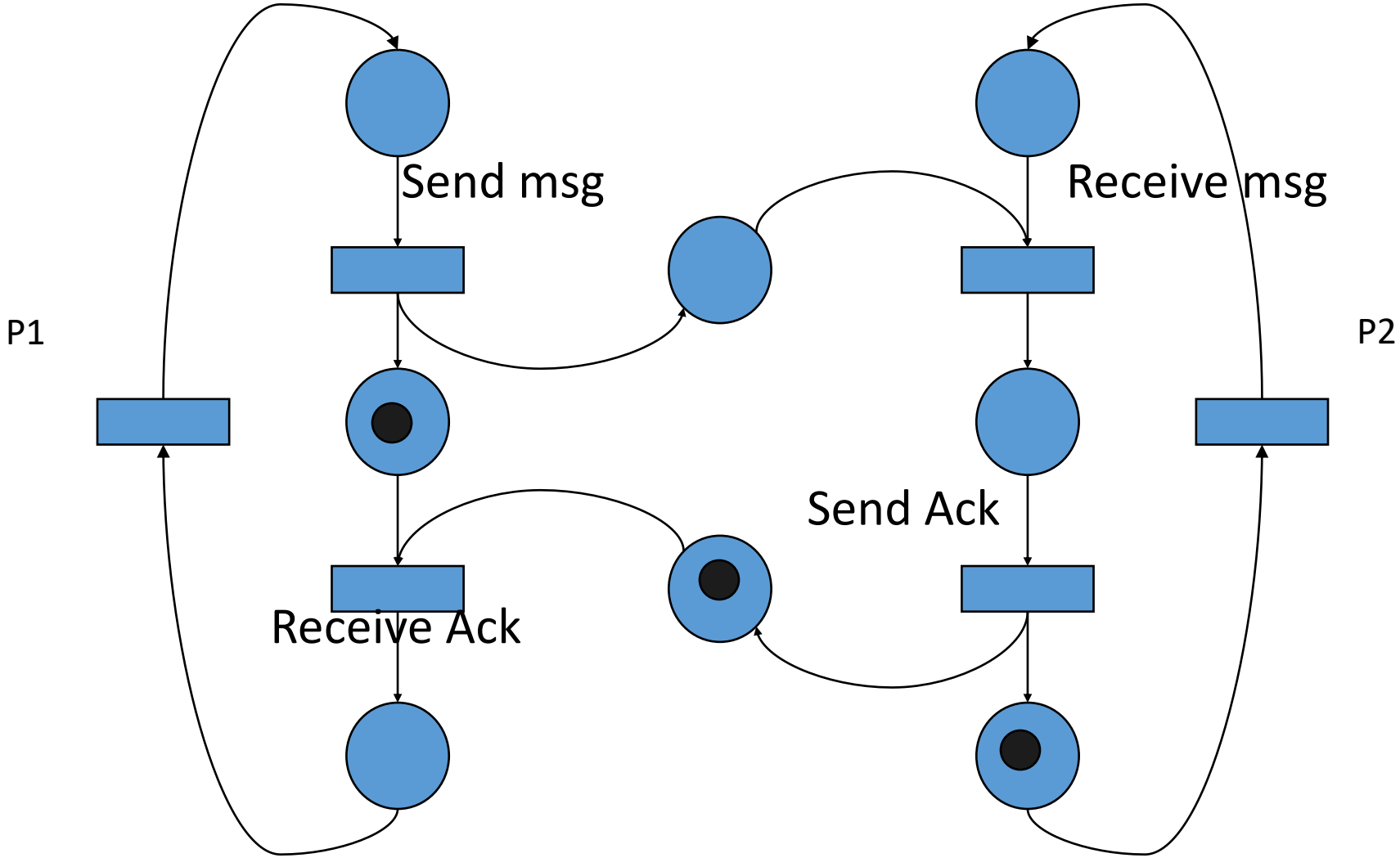
# Petri Nets Example Sender Receiver Communication



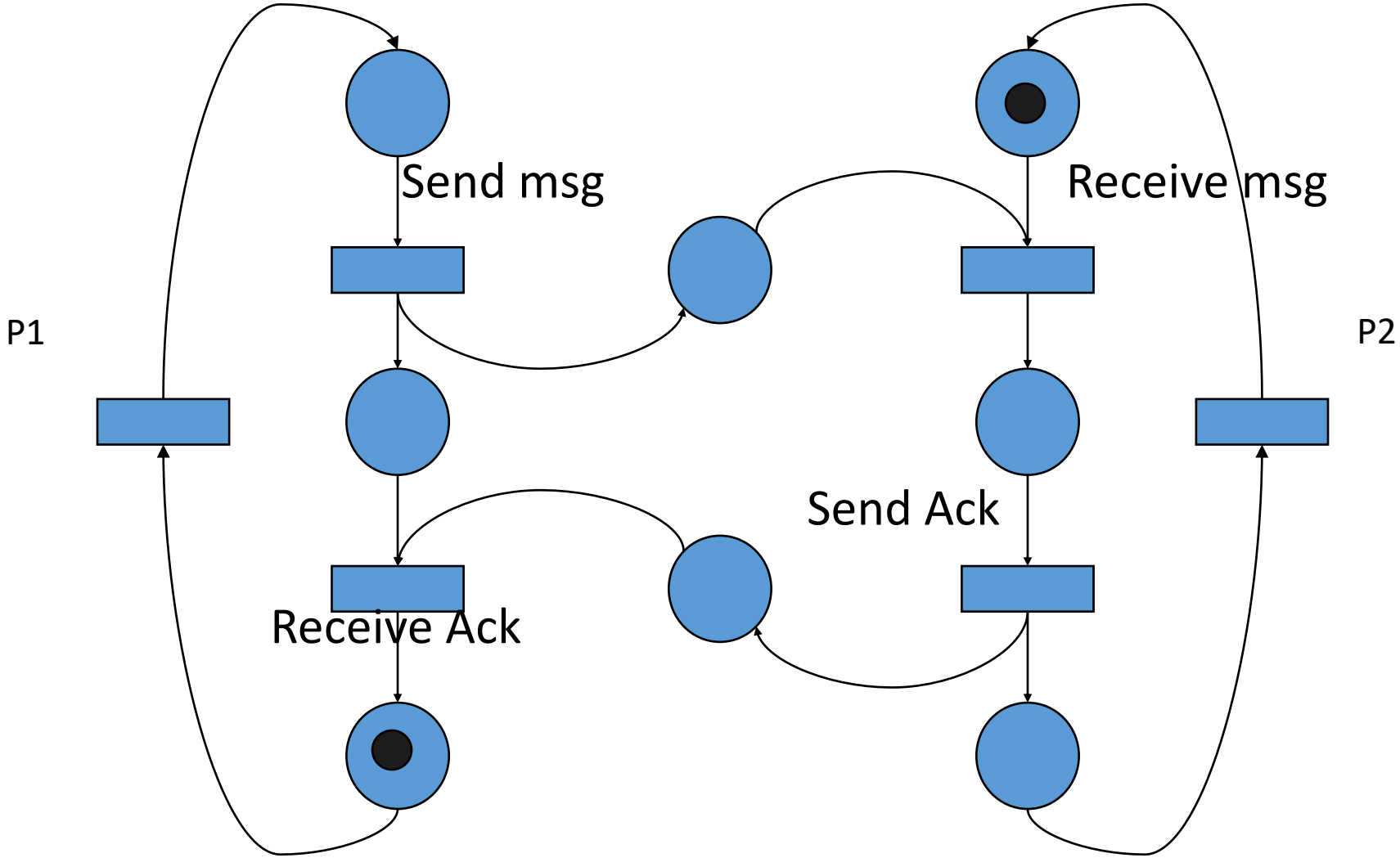
# Petri Nets Example Sender Receiver Communication



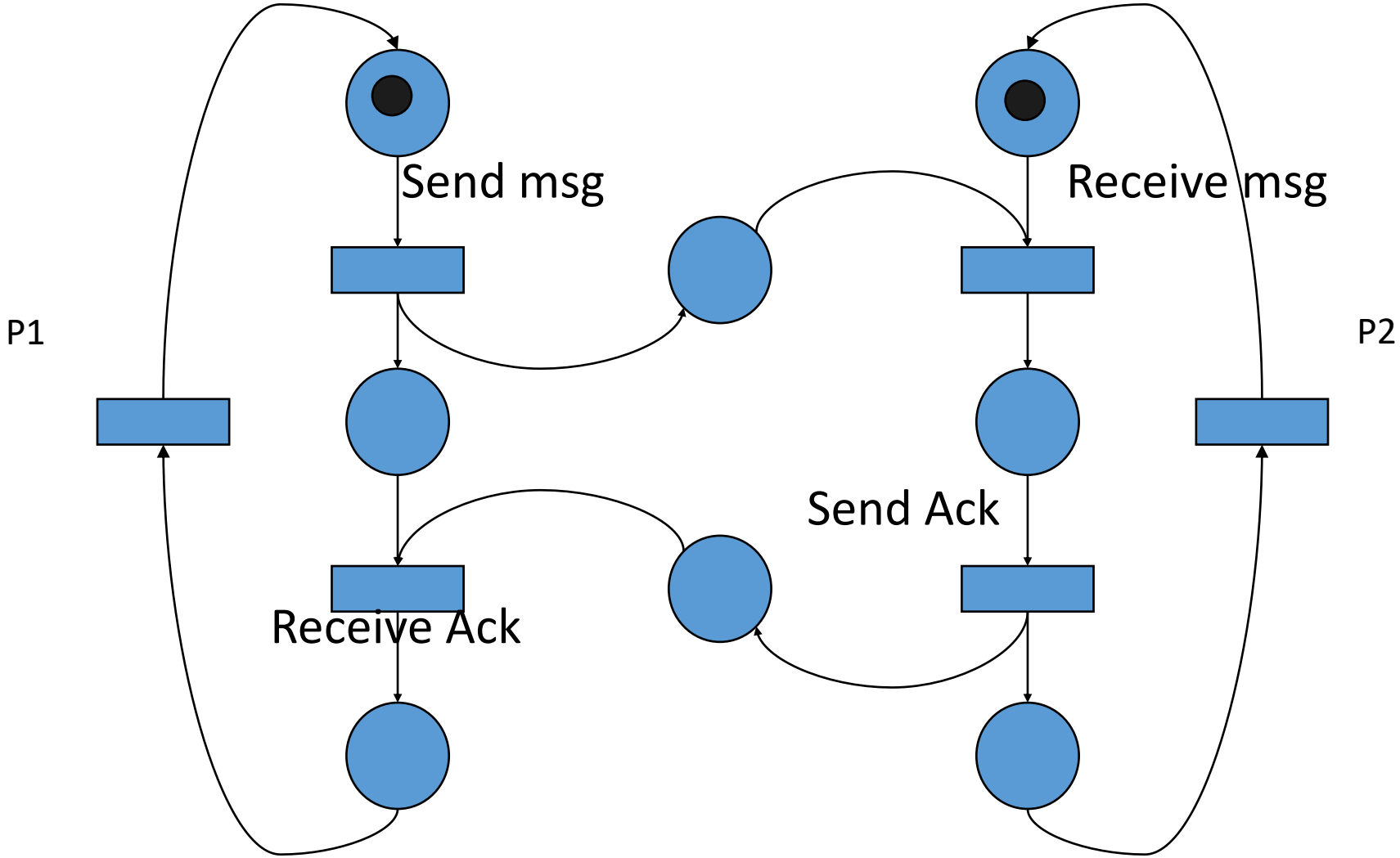
# Petri Nets Example Sender Receiver Communication



# Petri Nets Example Sender Receiver Communication

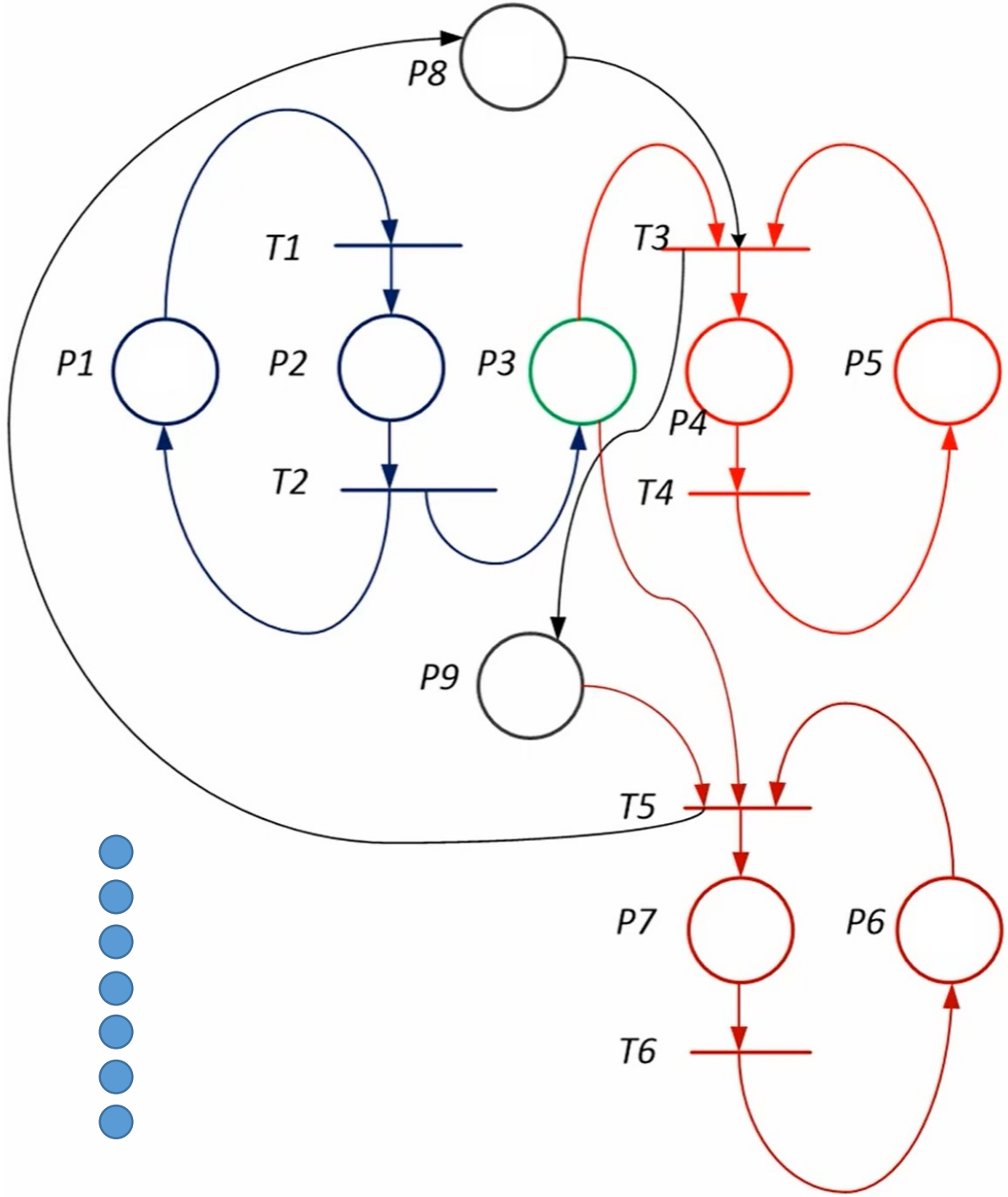


# Petri Nets Example Sender Receiver Communication



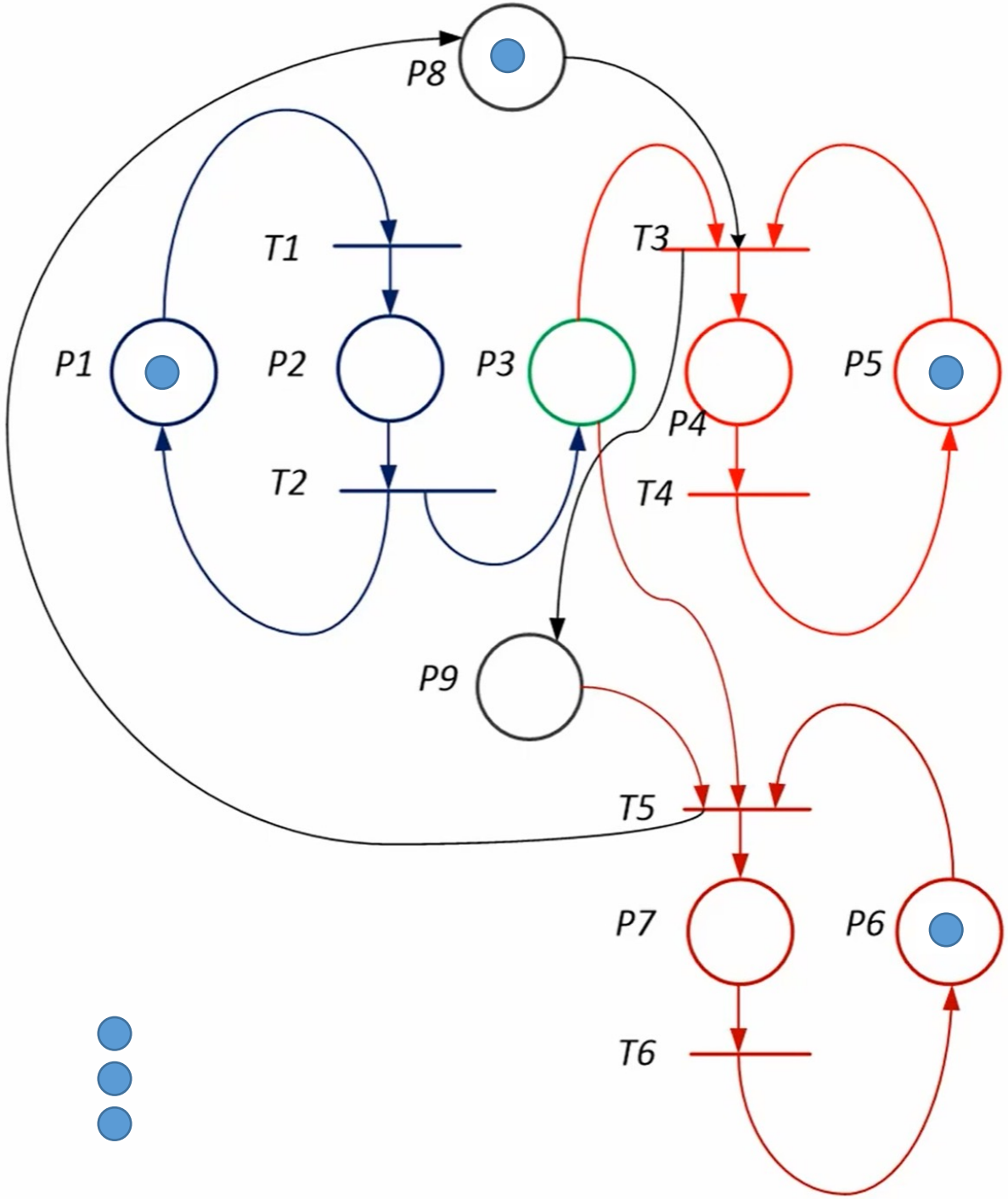
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



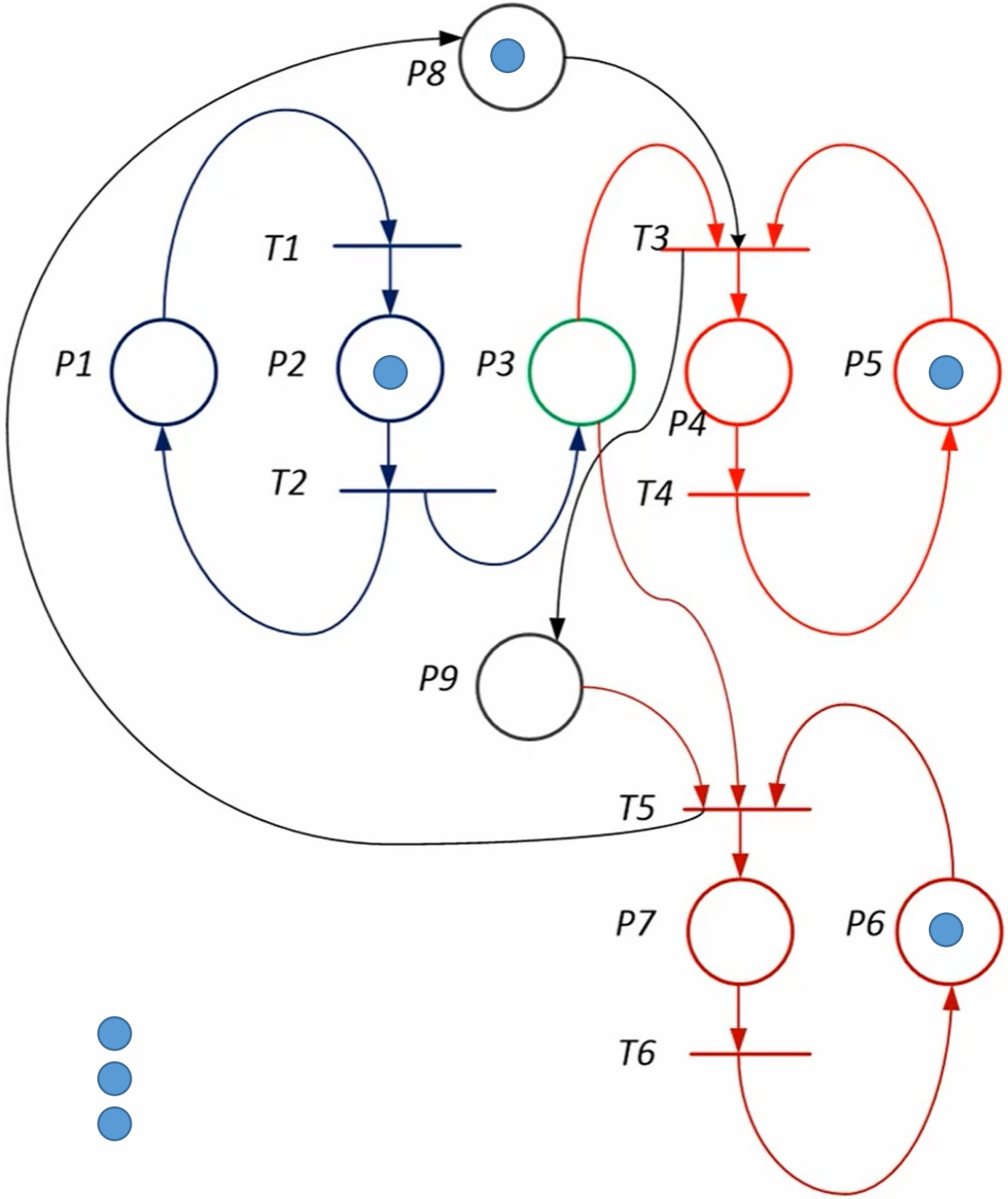
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



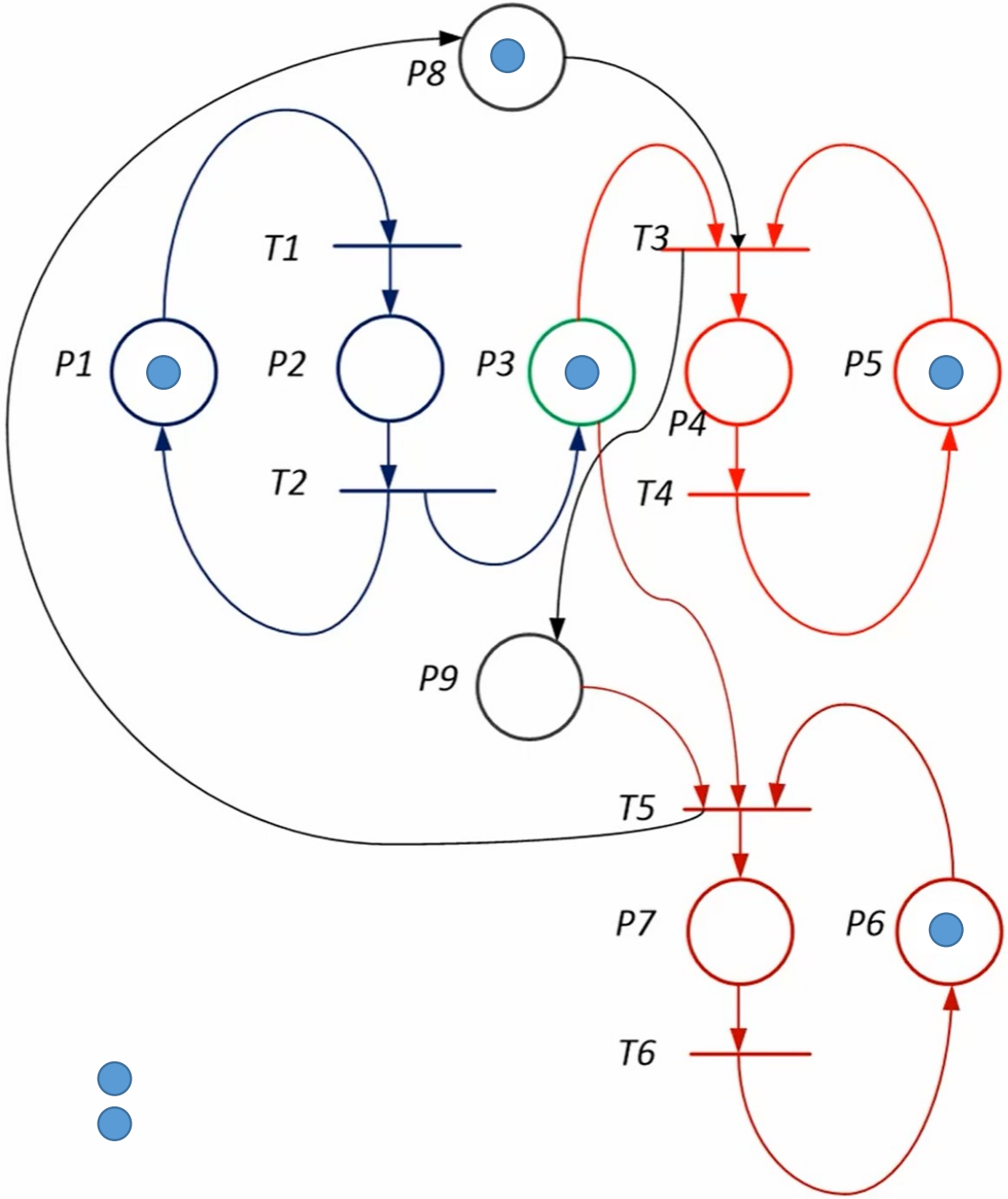
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



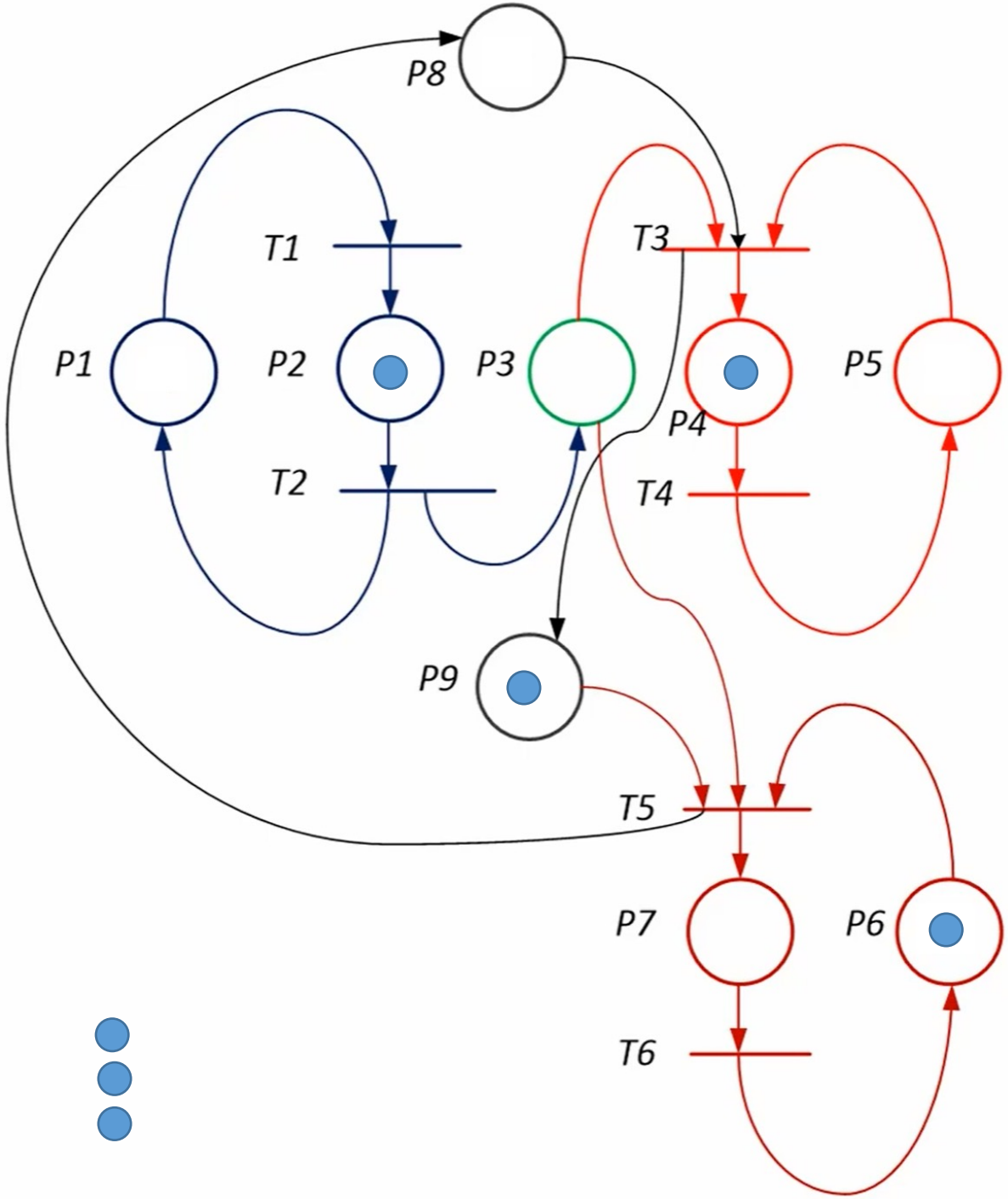
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



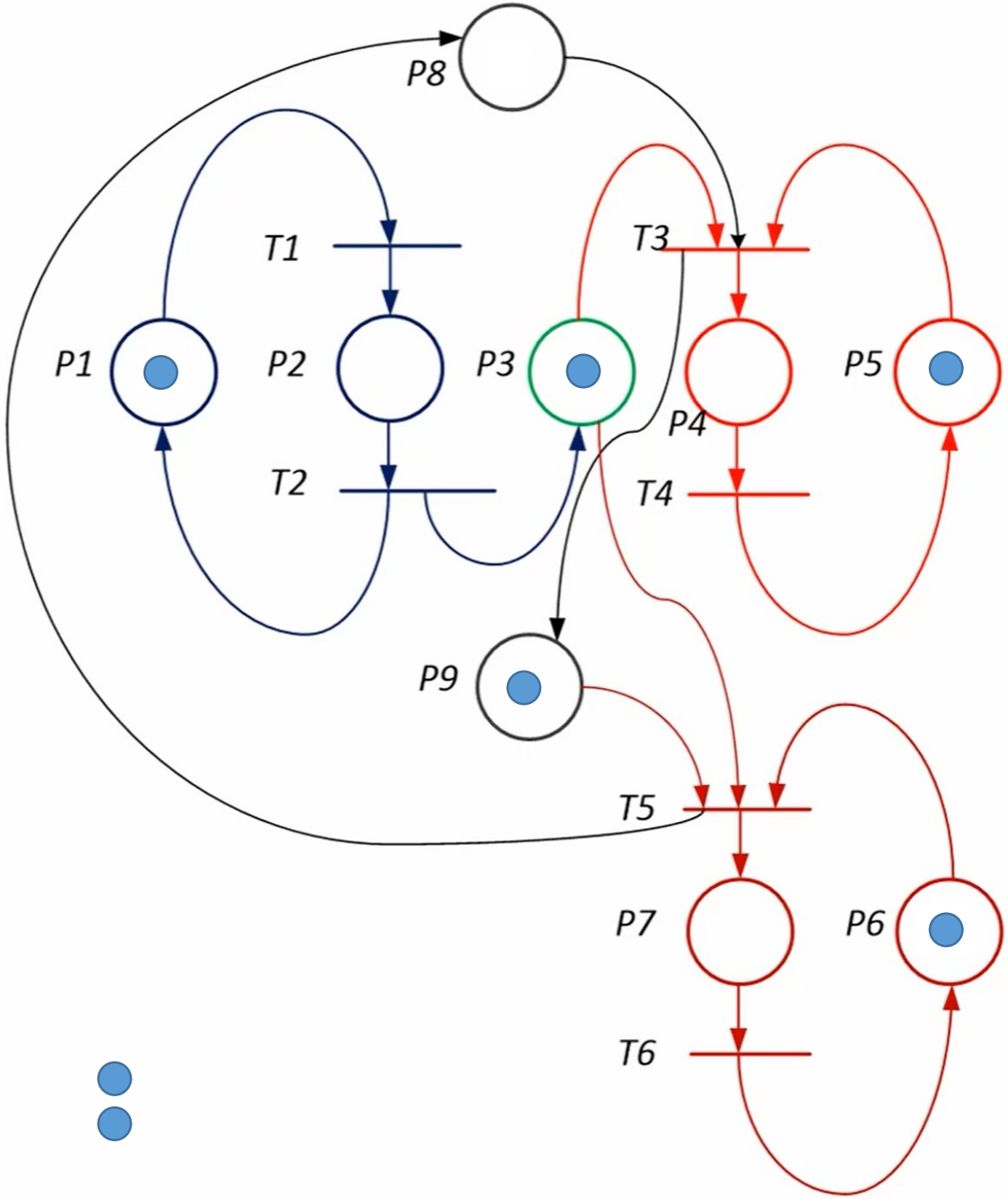
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



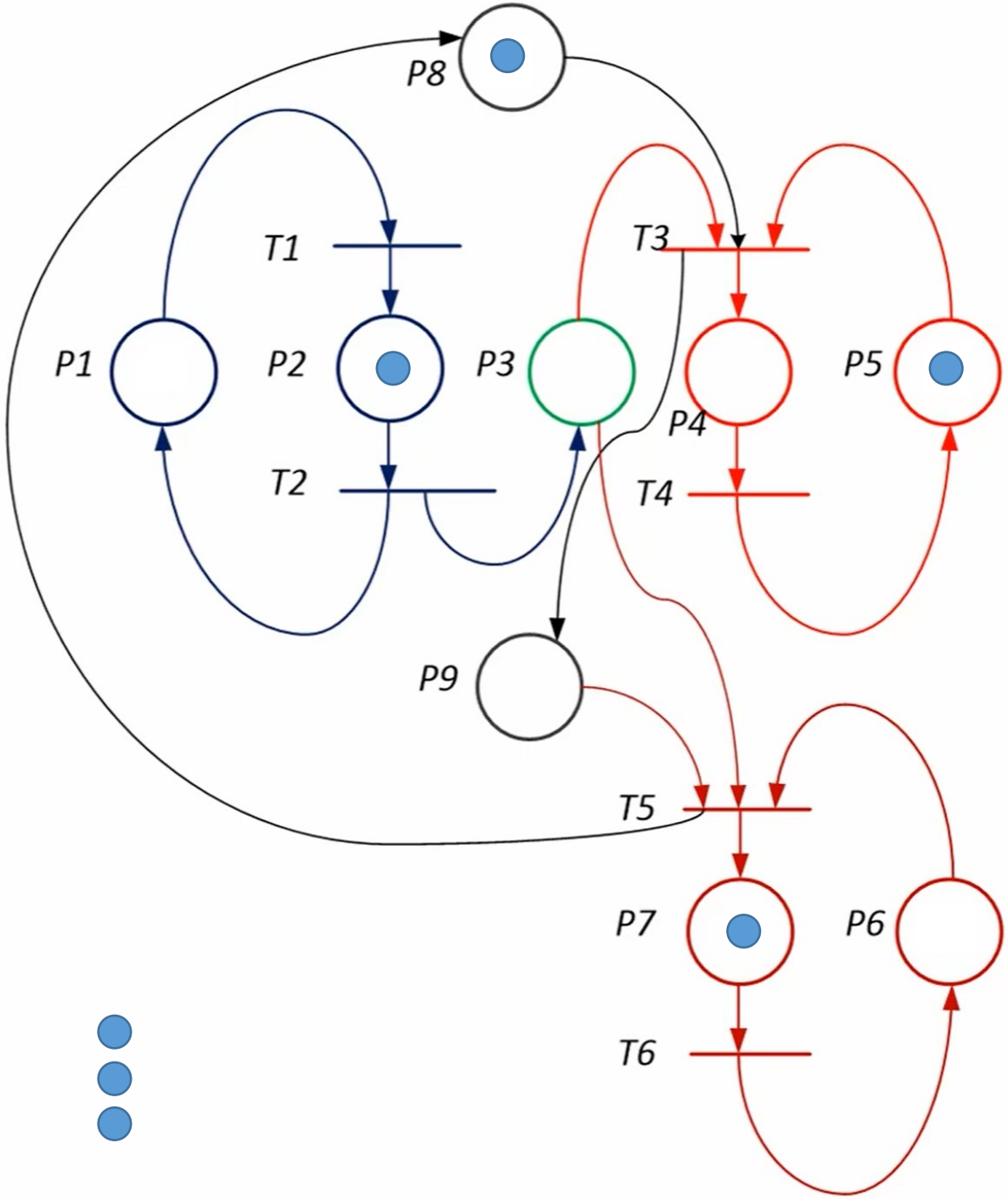
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



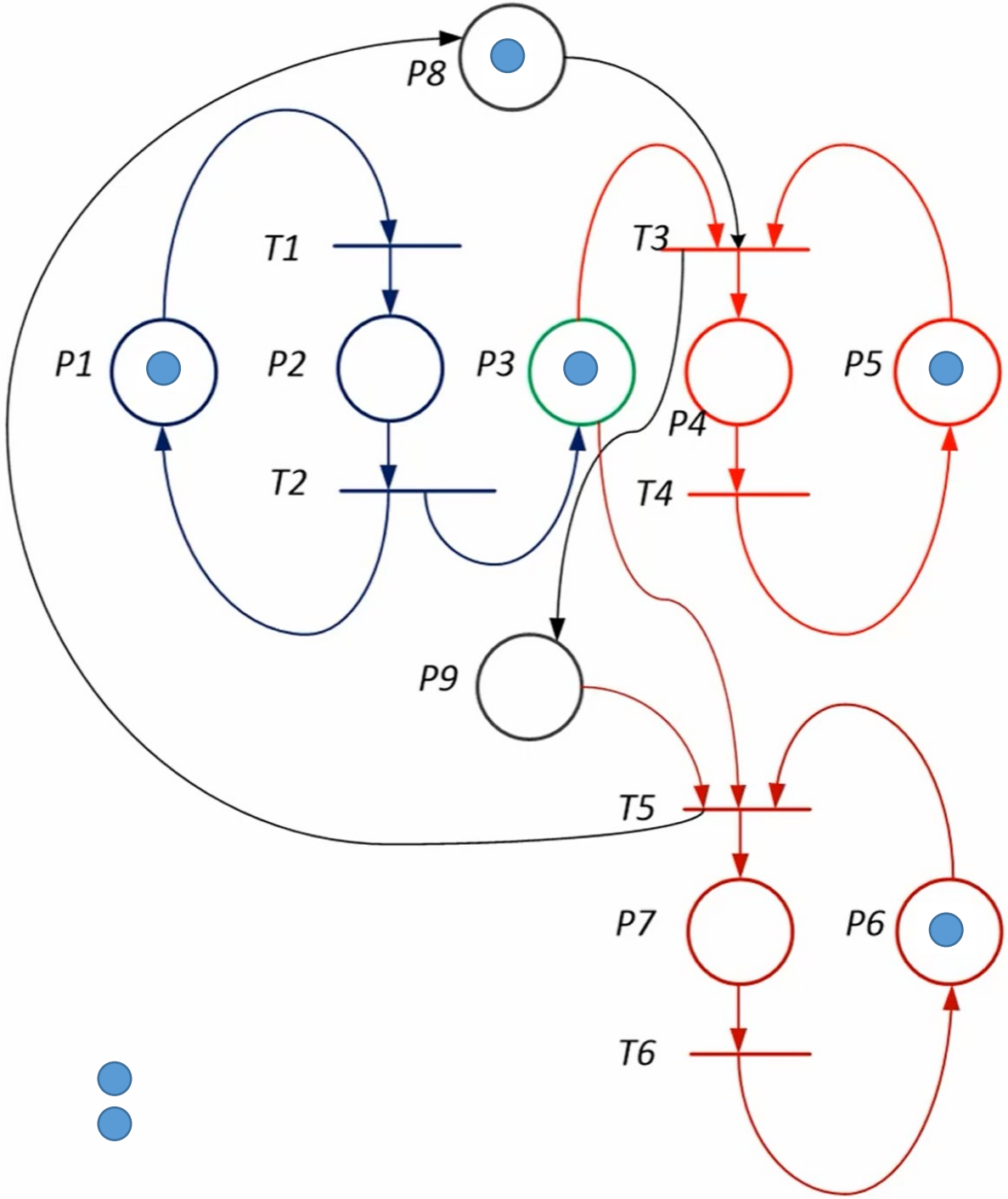
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



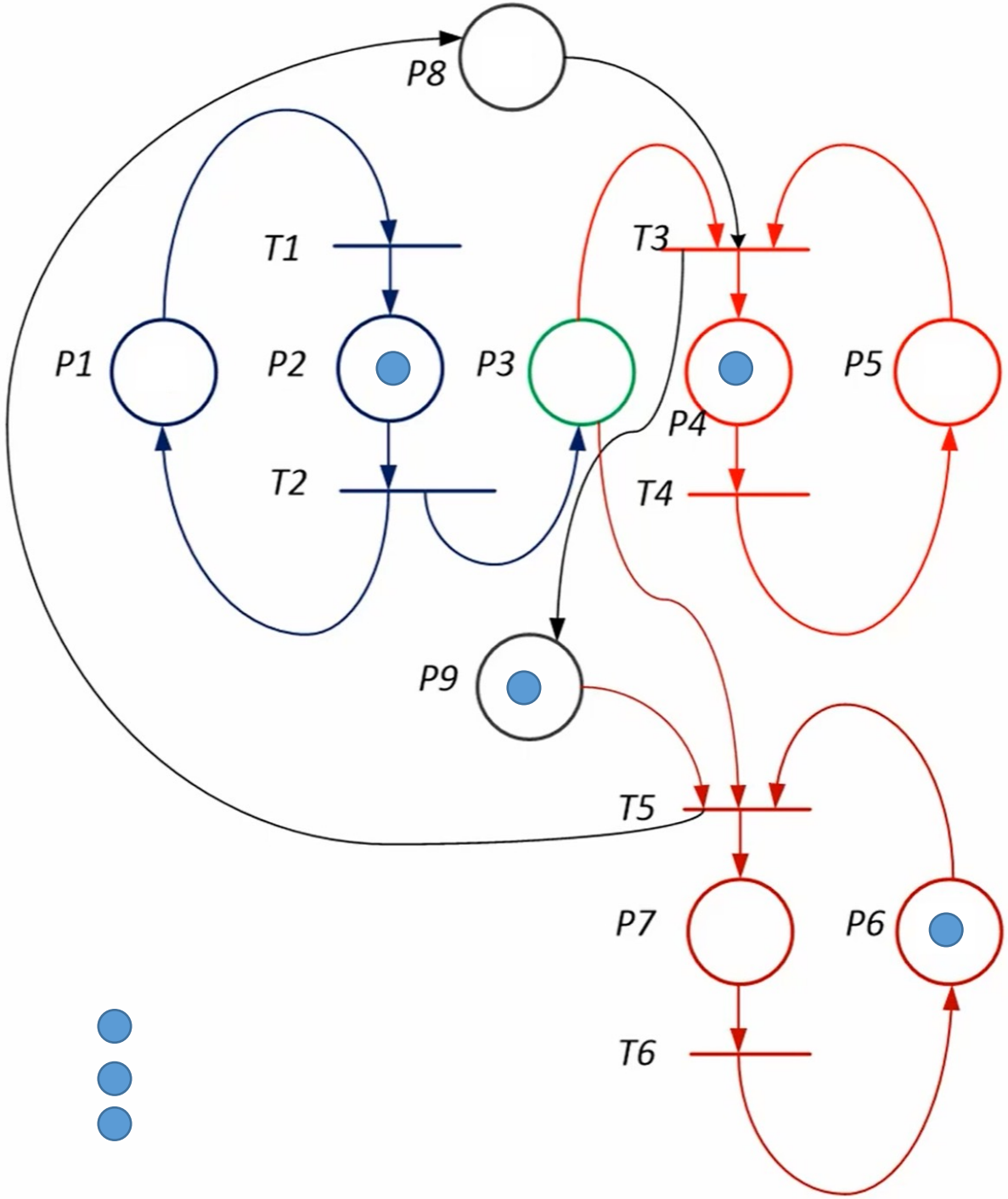
# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5

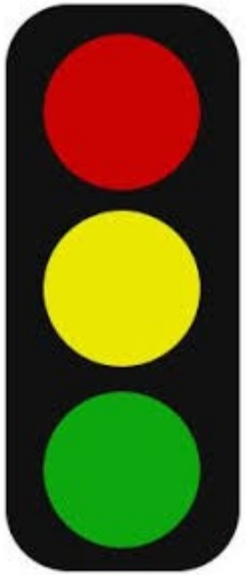


# Petri Nets Example Producer Consumer Problem

- P1 is a producer enabler for T1
- P3 is a consumer enabler for both T3 and T5
- P5 is consumer enabler for T3
- P6 is consumer enabler for T5



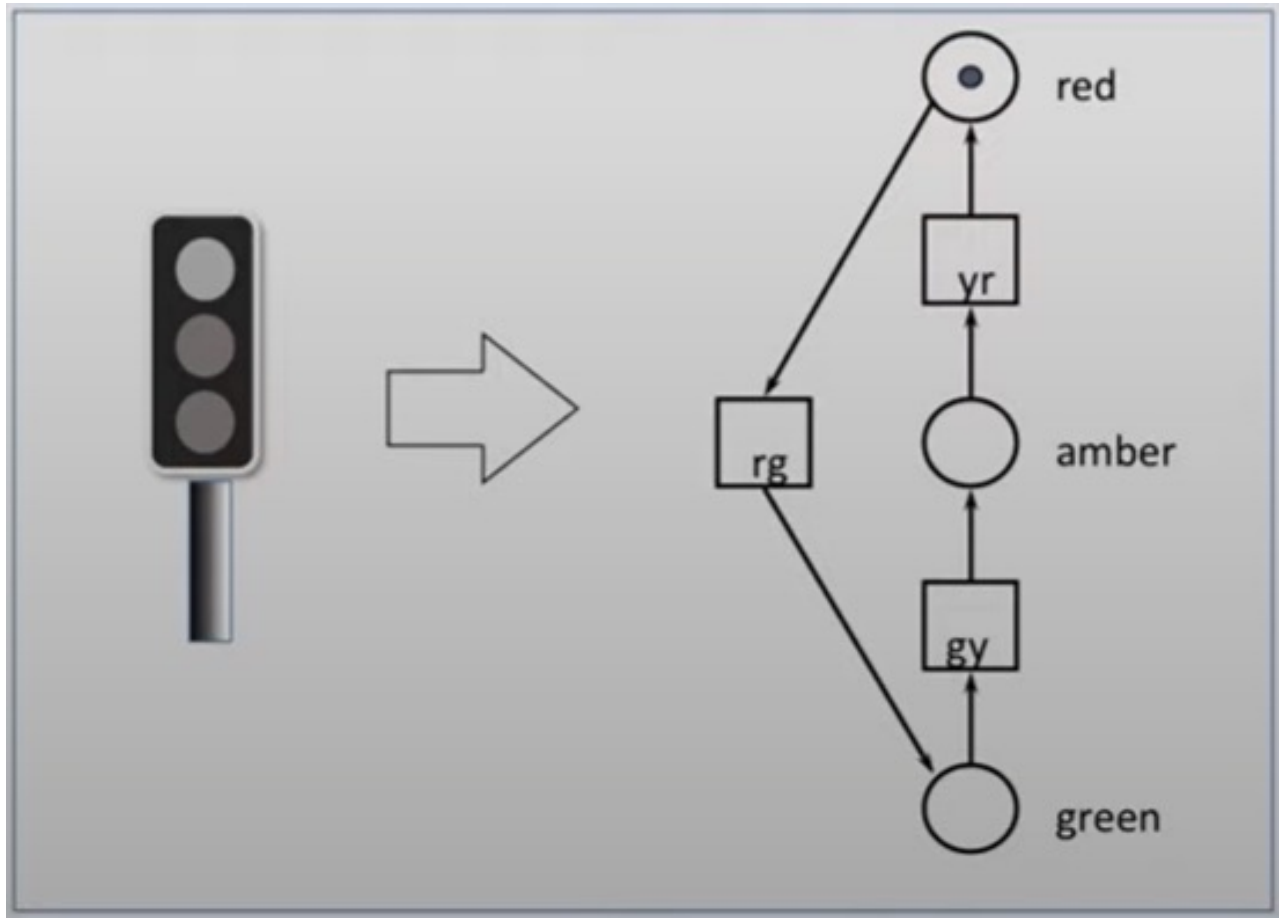
# Traffic Light Modeling



- Designing traffic light system for one way crossroad junction.
- The Sign should move from Green to Amber to Red then goes back to Green.
- Each road will have a light sign. Both signs must be controlled in a way that prevents accidents and allow only one road to flow while the other waits .
- Synchronization , Mutual Exclusion , Control Flow -> Petri Net.

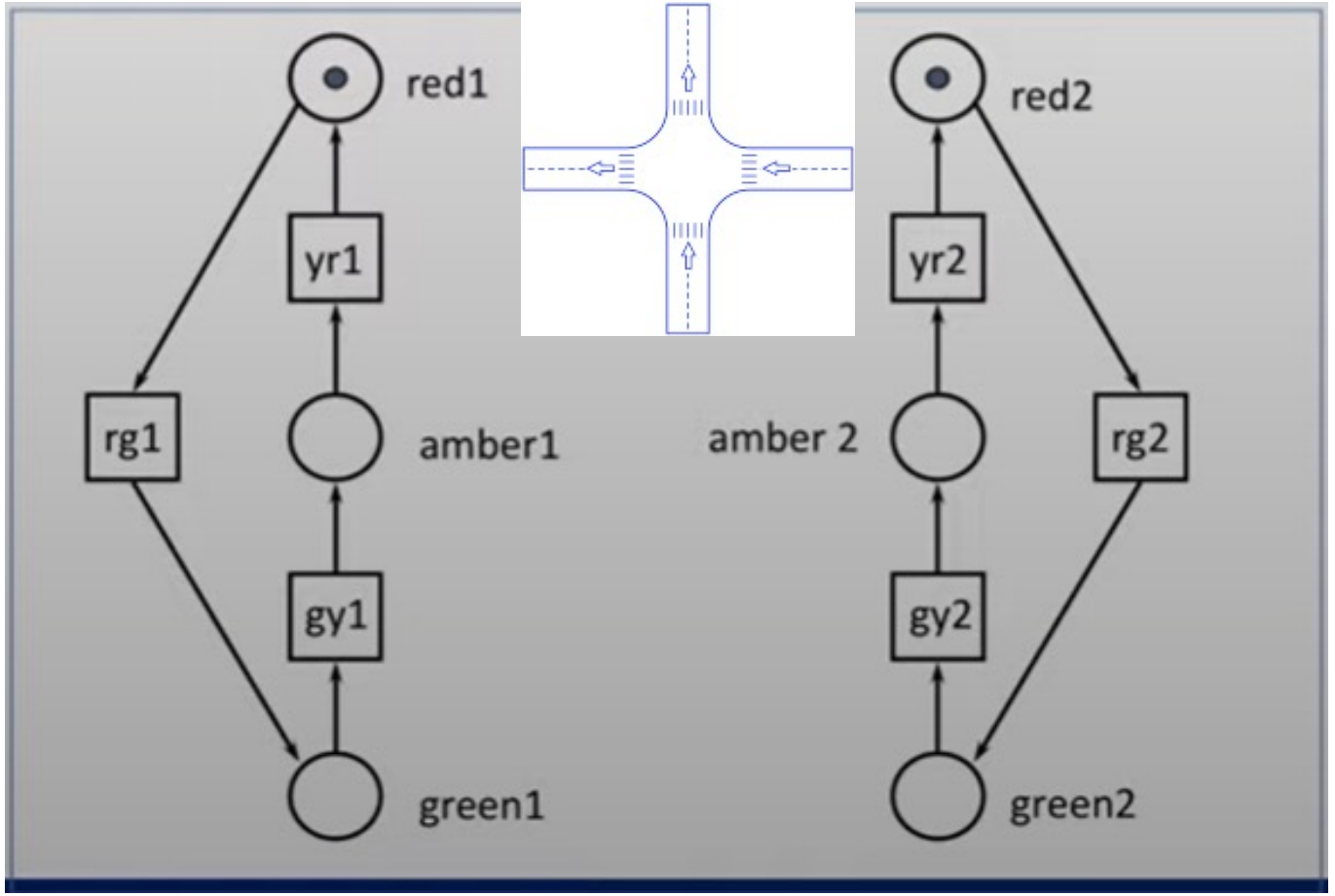
# Traffic Light Modeling

- Timing is not modeled
- Extension
  - Timed Petri Nets



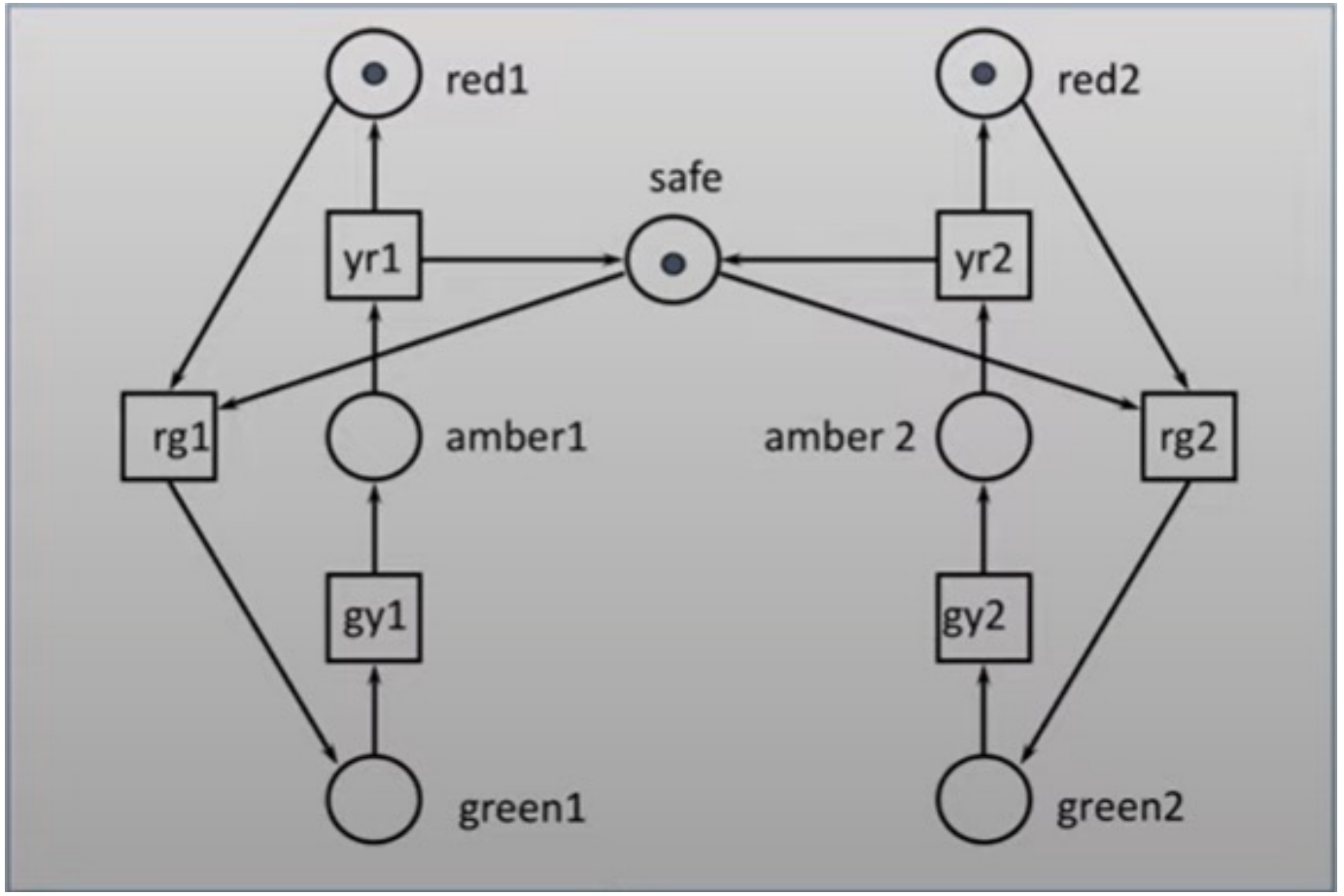
# Traffic Light Modeling

- Simplified Intersection
- Independent Control
  Can lead to disaster.
- Synchronization
  Condition



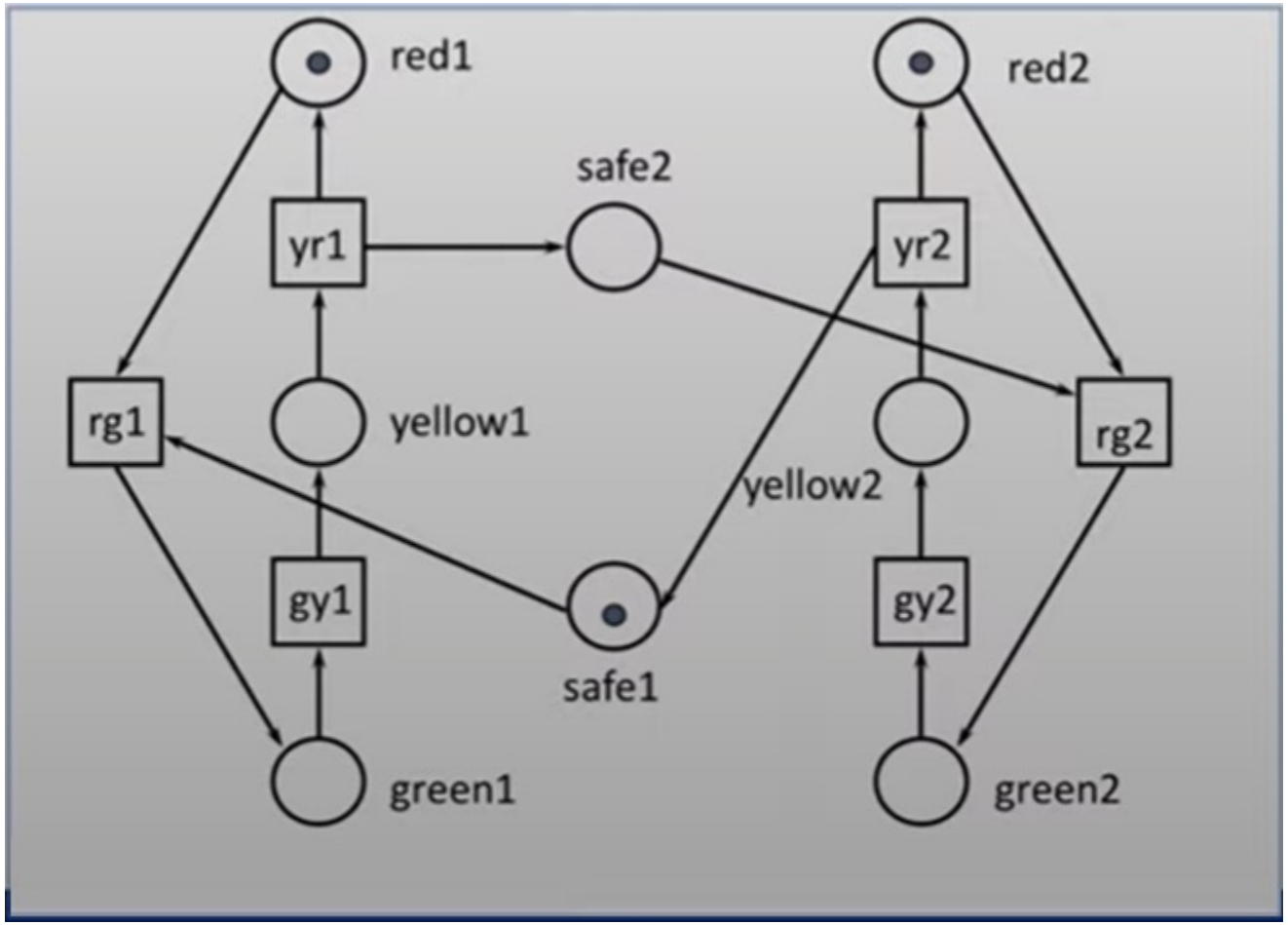
# Traffic Light Modeling

- Safe Place
- Conflict ?
- Control Flow



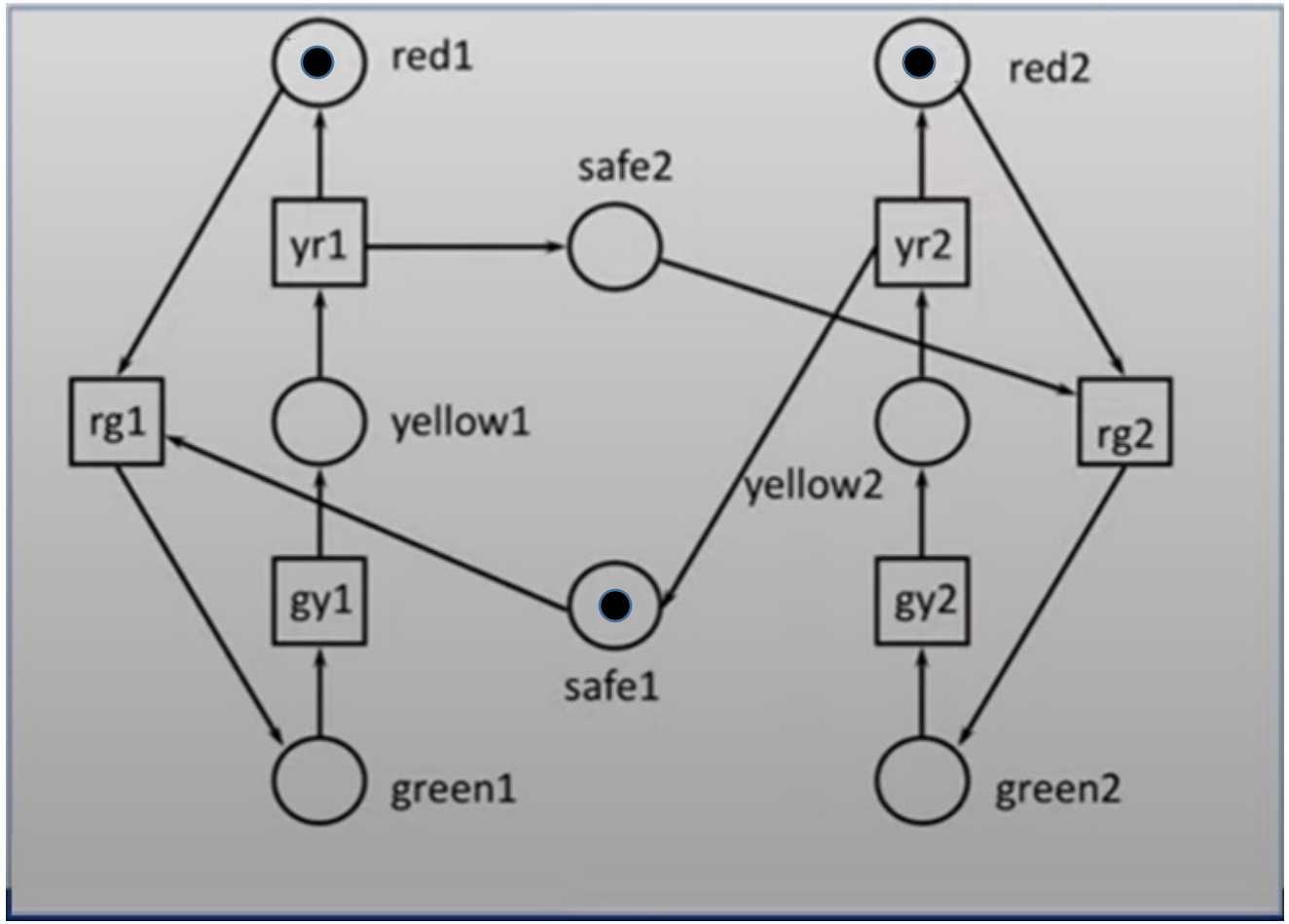
# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



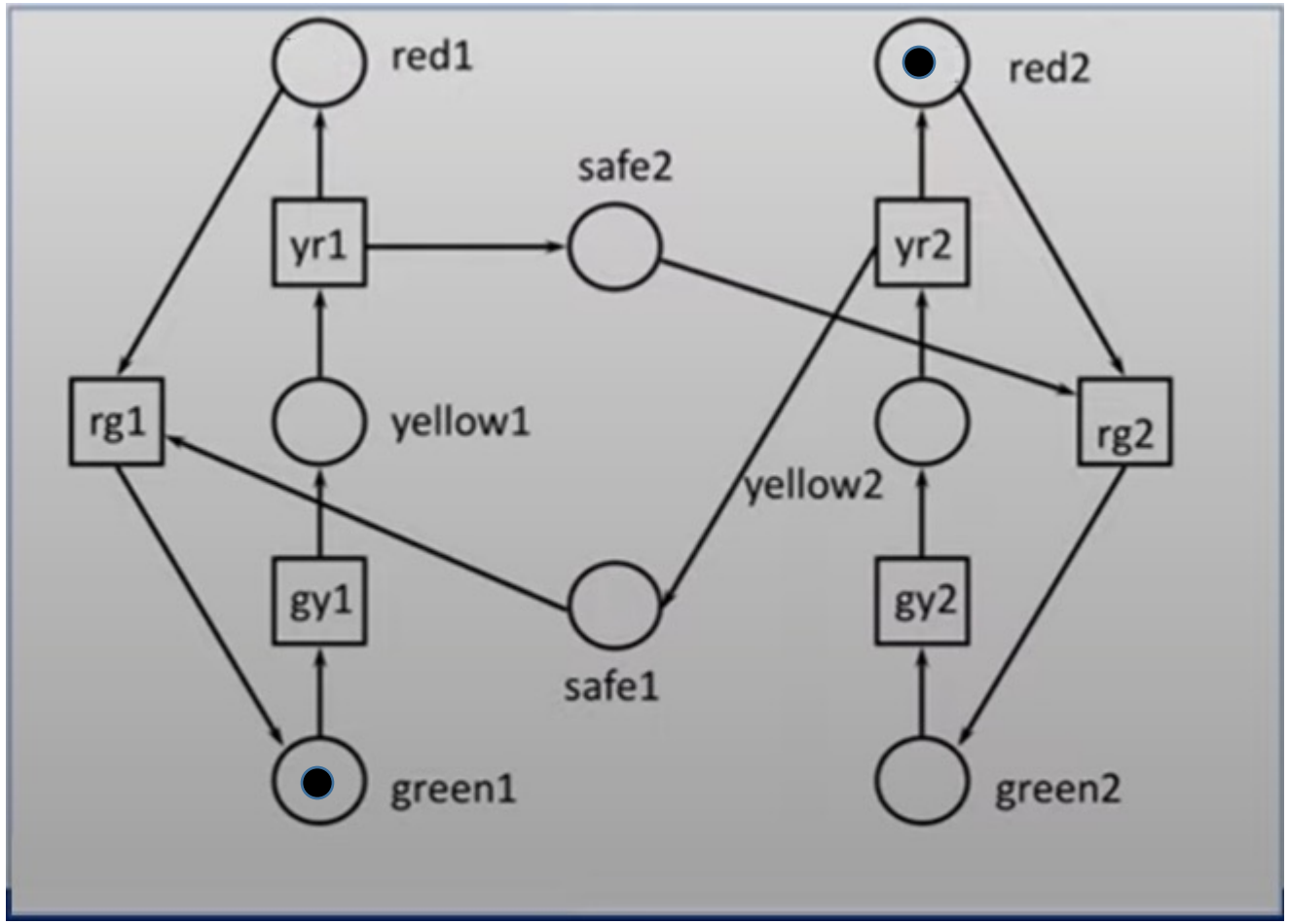
# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



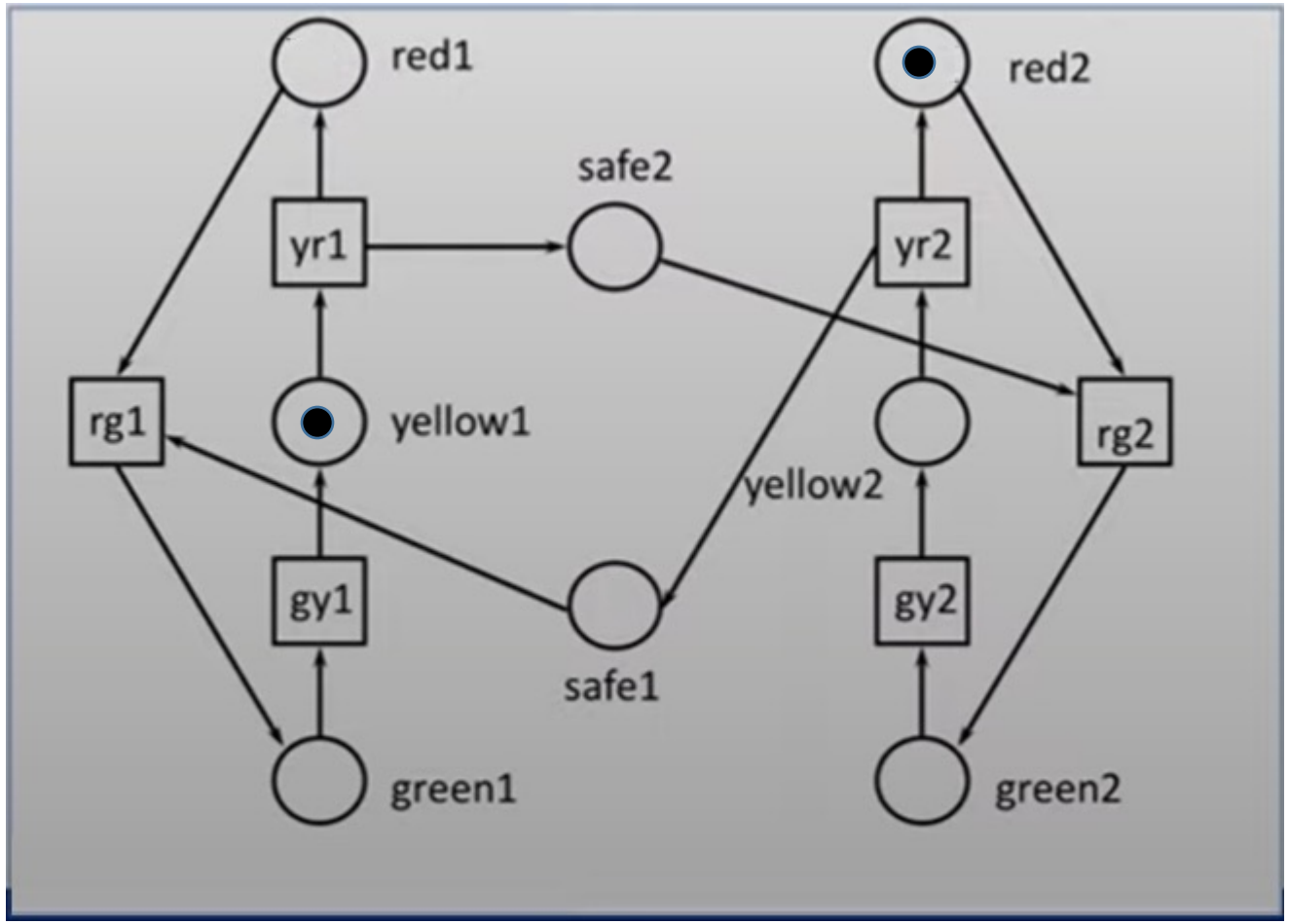
# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



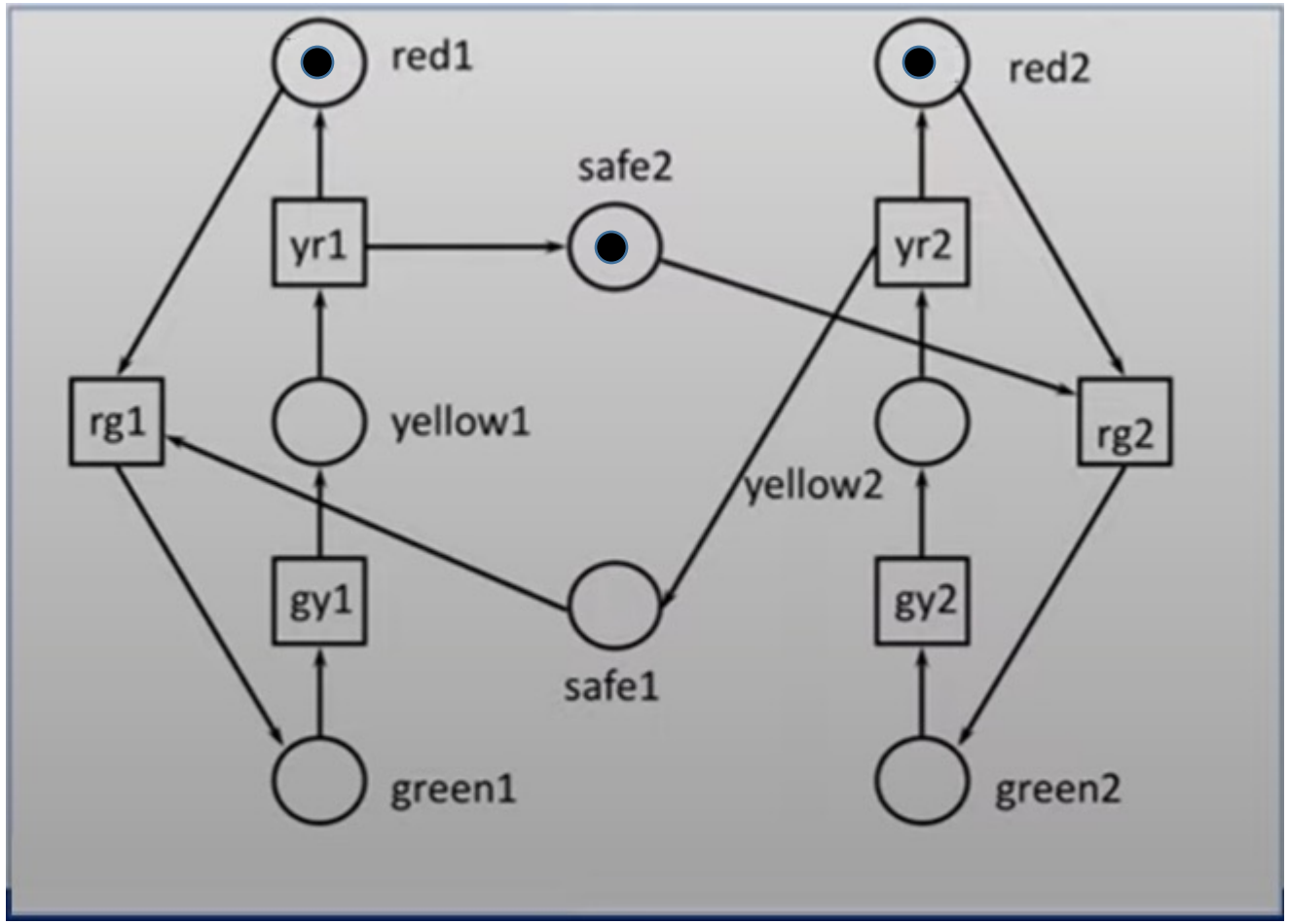
# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



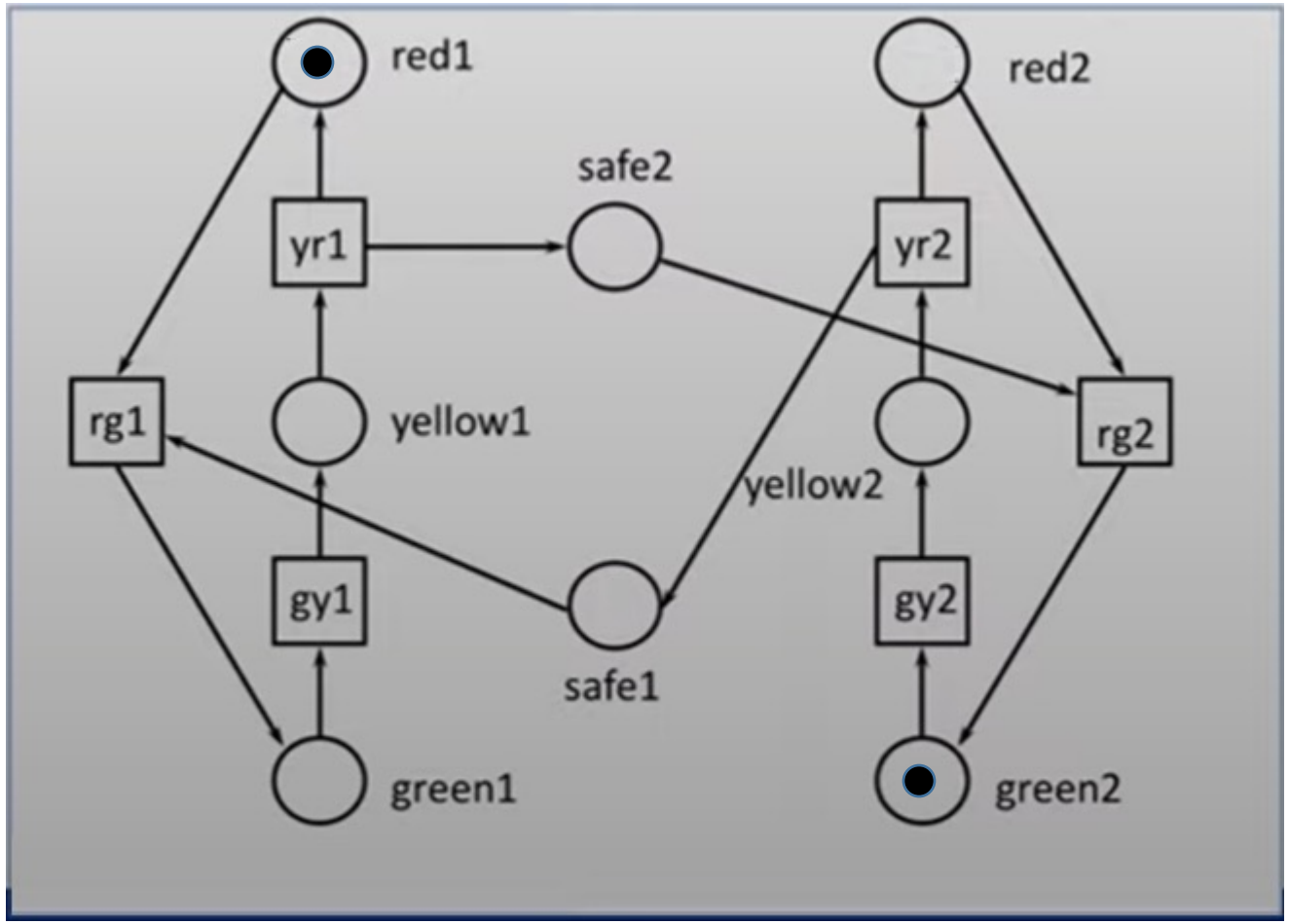
# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



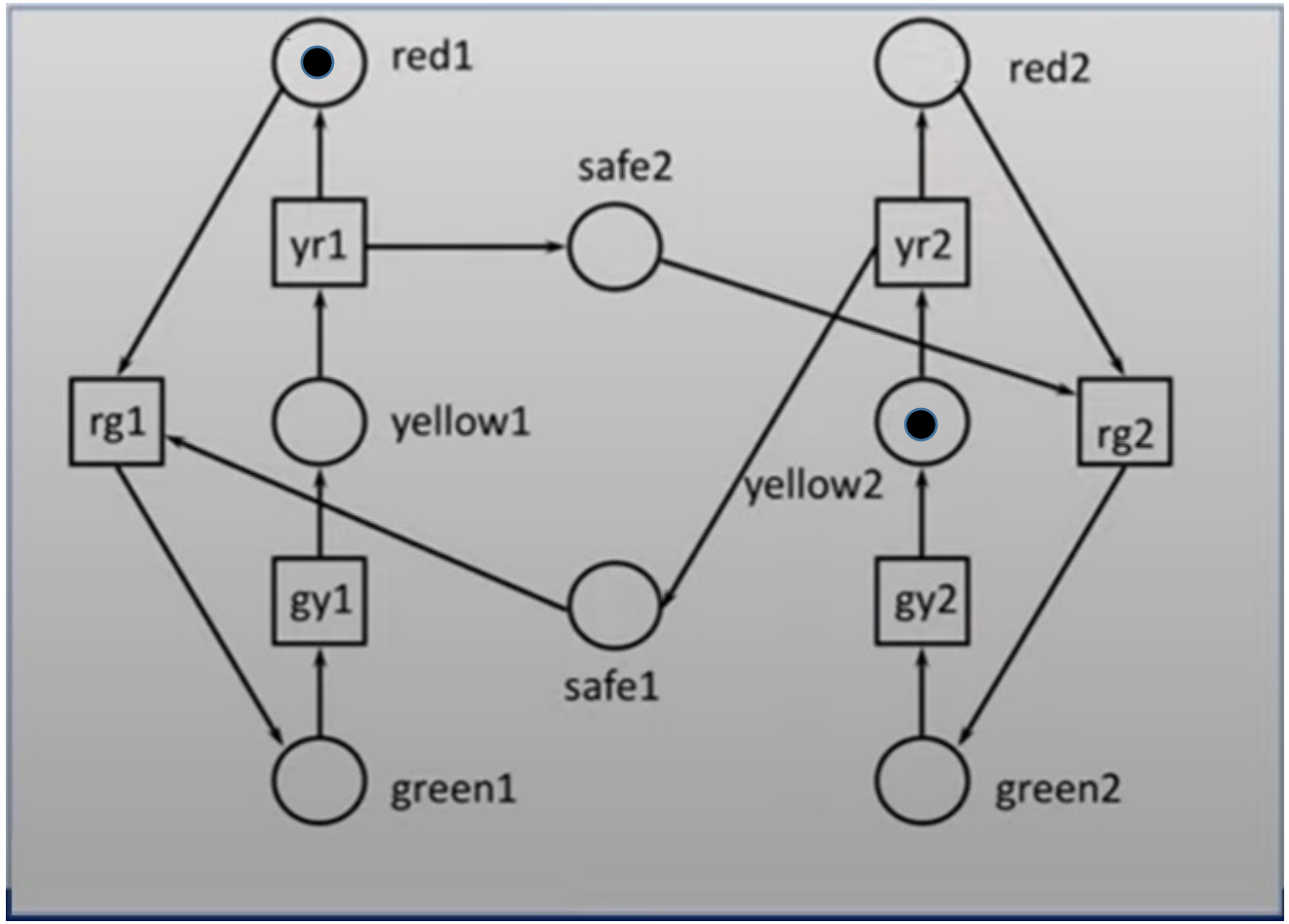
# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



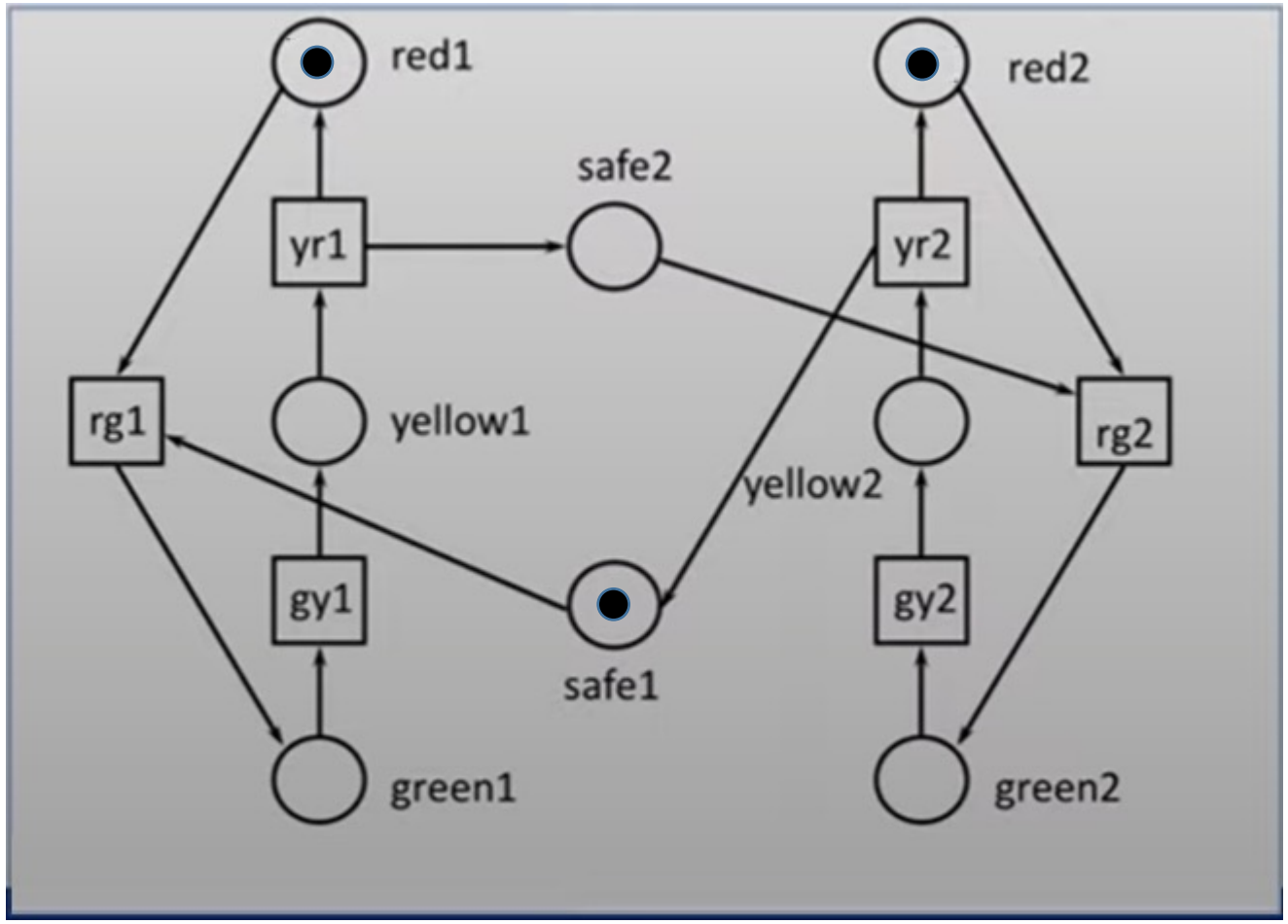
# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



# Traffic Light Modeling

- 2 Safe Places
- Fair
- Control Flow



# Simulation Tools

- http://petri.hp102.ru/pnet.html

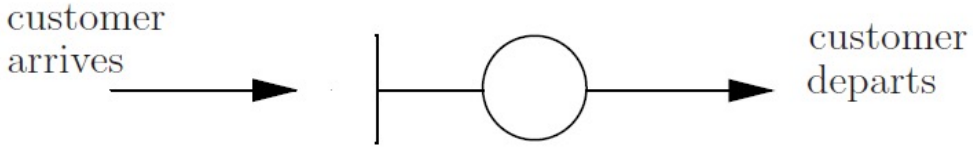
# Behavioral Analysis

- General problems of Safety and Blocking in common control designs are detailed into specific properties when dealing with Petri-Nets
  - Bounded-ness
  - Fairness
  - Conservation
  - Cover-ability
  - Persistence
  - Liveness

- Many of these properties are motivated by the fact that Petri nets are often used in resource sharing environments where we would like to ensure efficient and fair usage of the resources.

# Boundedness

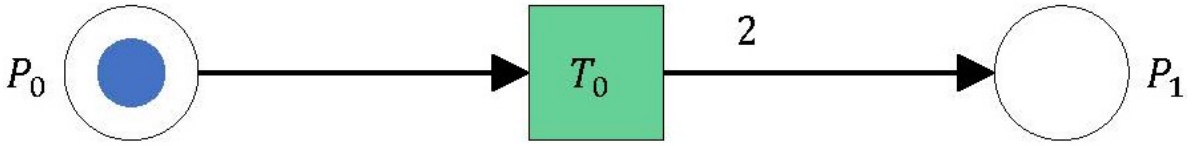
- The tokens in place *Q* of the Petri net represent customers entering a queue. Clearly, allowing queues to grow to infinity is undesirable, since it means that customers
- wait forever to access a server.
- classical system theory, a state variable that is allowed to grow to infinity is generally an indicator of instability in the system. Similarly here, unbounded growth in state components (markings) leads to some form of instability.



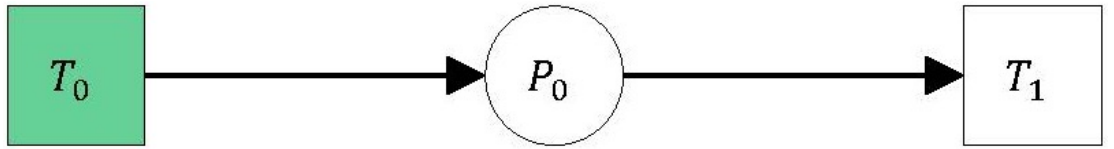

Place $p_i \in P$ in Petri net $N$ with initial state $x_0$ is said to be *k-bounded*, or *k-safe*, if $x(p_i) \leq k$ for all states $x \in R(N)$, that is, for all reachable states.

# Boundedness

- For the given initial marking, the Petri Net in this figure is bounded because m_0 ≤ 1 and m_1 ≤ 2. In general, the Petri Net is bounded: let the initial value of m_0 = q_0 and the initial value of m_1 = q_1; m_0 ≤ q_0 and m_1 ≤ 2q_0+q_1

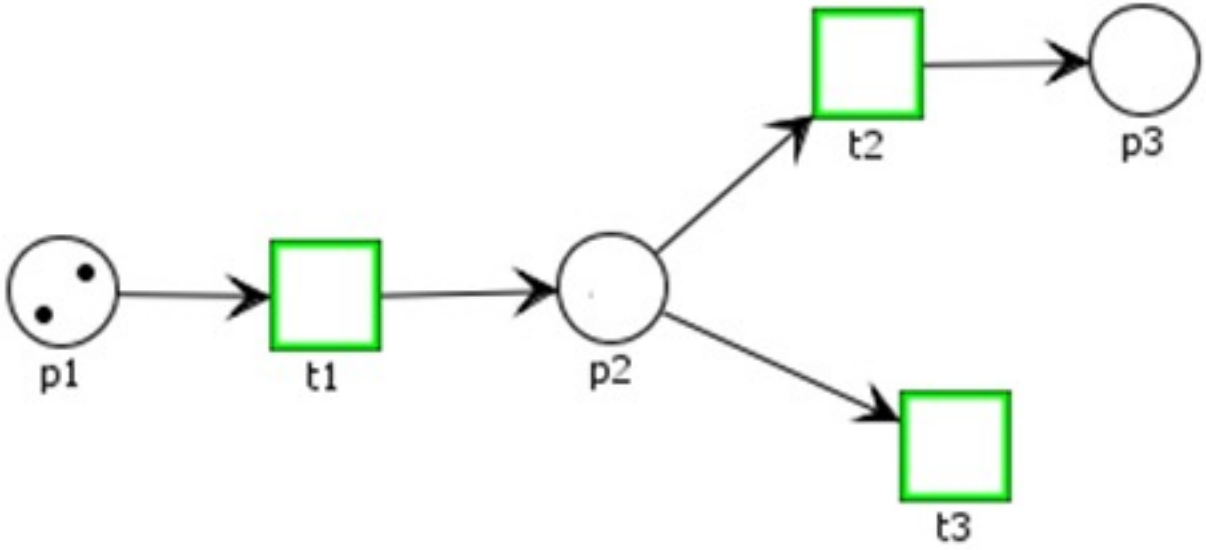



- An example of an unbounded Petri Net Regardless of the initial marking, the Petri Net in this figure is not bounded (or it is unbounded) because m_0 → ∞ when T_0 keeps firing.



# Boundedness

- For the given Petri Net, the number of tokens for P_1,P_2 and P_3 is represented by m_1,m_2 and m_3 respectively. Based on the definition of boundedness, the given Petri Net is bounded because m_1 ≤ 2,m_2 ≤ 2 and m_3 ≤ 2. In other words, there is a natural number (2 in this case) such that the number of tokens in P_1,P_2 and P_3 never exceeds the number.



# Conservation

- Conservation is a property of Petri nets to maintain a fixed number of tokens for all states reached in a sample path. However, this may be too constraining a property.

>> There exists a fixed total number of tokens at any state.

# Liveness

- A complement to the properties of deadlock and blocking is the notion of live transitions. Here, instead of being concerned with being unable to fire any transition or unable to eventually reach a marked state, we are concerned with being able to eventually fire a given transition.

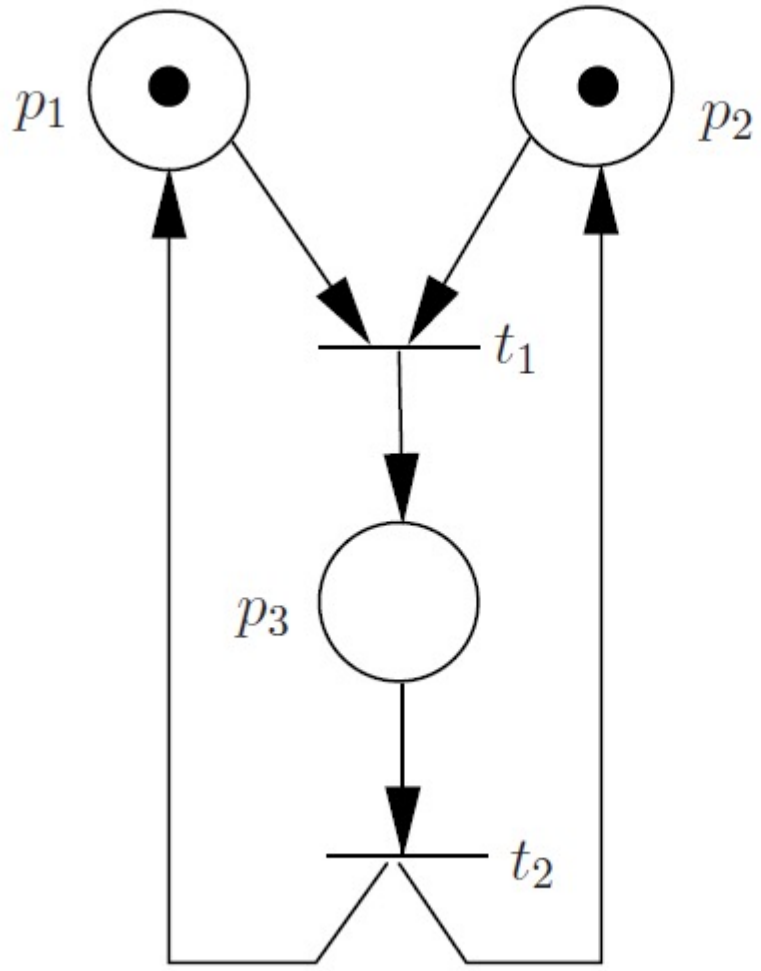
>> There always exist a sample path such that a transition can eventually fire from any state reached.

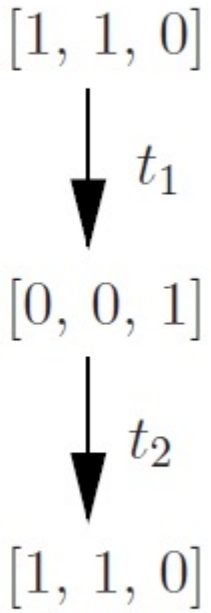
# Persistence

- Persistence is the property of a Petri net not to disable any enabled transition because of the firing of another enabled transition.
- >> A Petri net is said to be *persistent* if, for any two enabled transitions, the firing of one cannot disable the other..

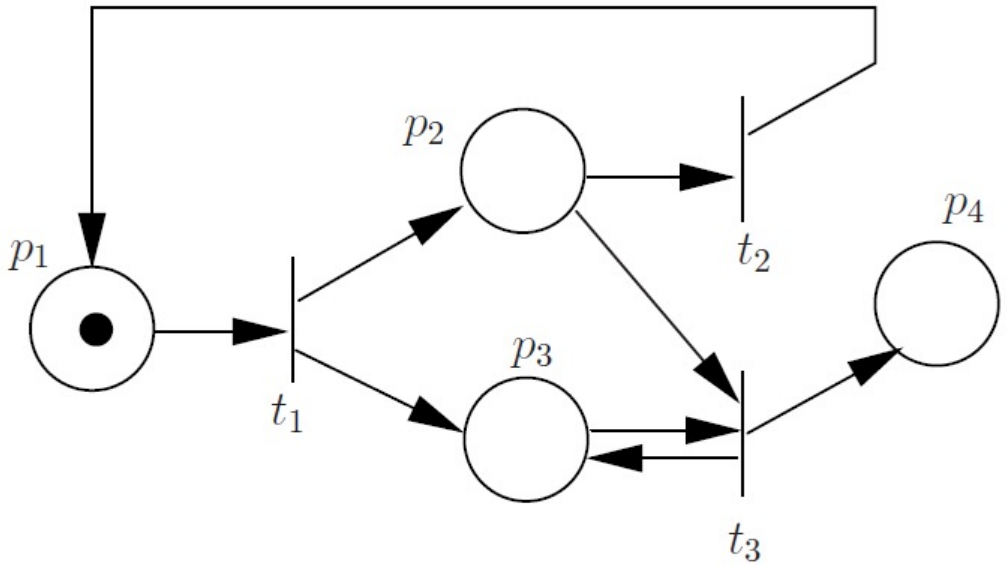
# Coverability

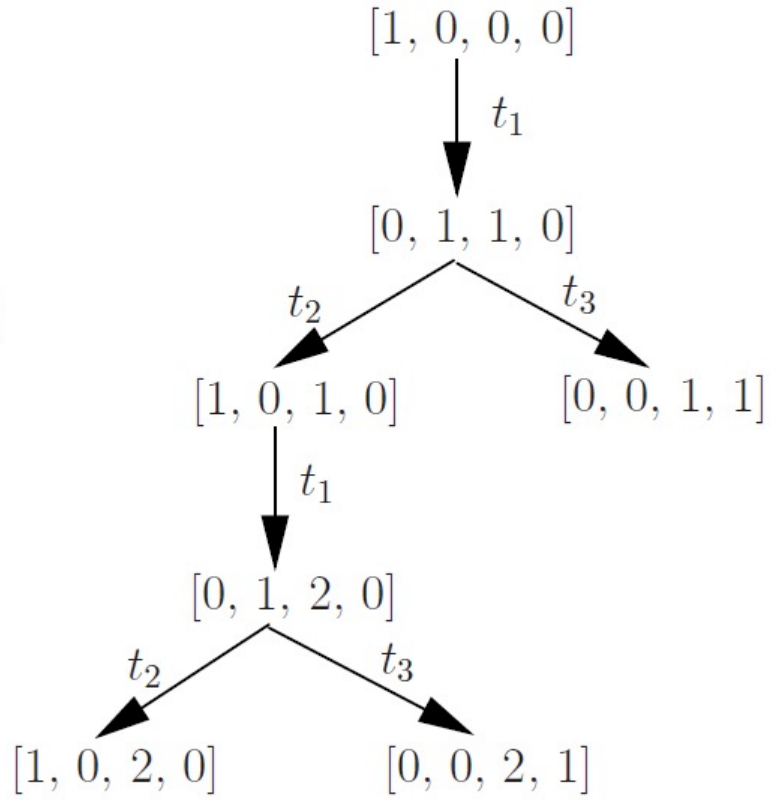
- traversing the states that can result for petri net markings due to firing.
- This technique is based on the construction of a tree where nodes are Petri net states and arcs represent transitions
- Shown a tree whose root node is [1, 1, 0]. We then examine all transitions that can fire from this state, define new nodes in the tree, and repeat until all possible reachable states are identified.
- In this simple case, the only transition that can initially fire is
- *t*1, and the next state is [0, 0, 1]. Now *t*2 can fire, and the next state is [1, 1, 0]. Since this state already exists (it is the root node), we stop there.



# Reachability




- Right branch firing t3 leads to a deadlock or terminal state
- Left branch firing t2 repeats the behavior with an extra token in P3
- This is called infinite reachability tree (unbounded)

# Reachability Example

- Initial State M0=(2,0,0)
- Can you reach Mn=(2,0,1)