Parallel and Concurrent Programming CS3S666

Petri Nets

Introduction

- Petri nets are a graphical and mathematical modelling tool applicable to many systems.

- They are a tool for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic.

Introduction

- As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems.

Introduction

- As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behaviour of systems.

- Petri nets can be used by both practitioners and theoreticians.

Introduction

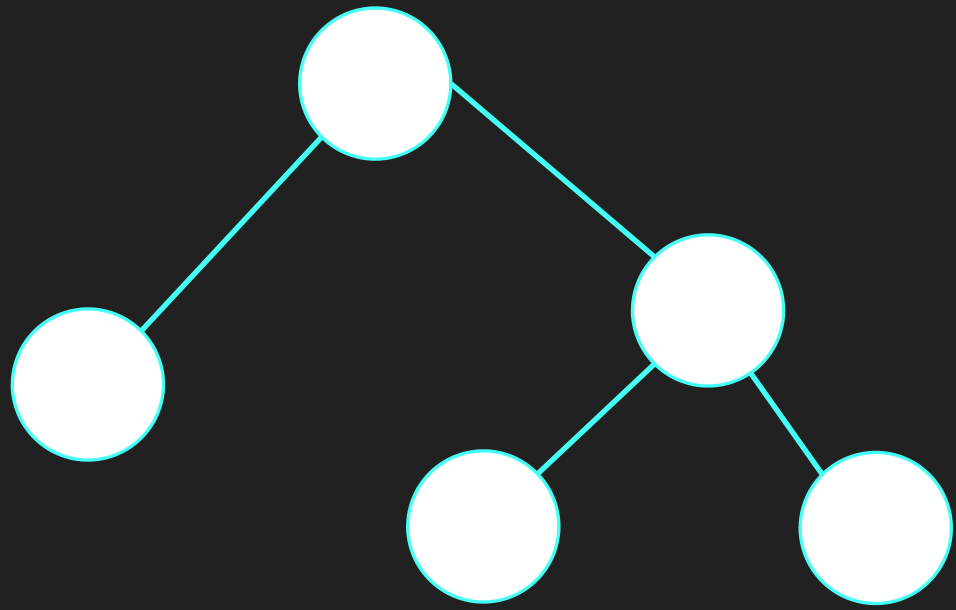
- A major weakness of Petri nets is the complexity problem, i.e., Petri-net-based models tend to become too large for analysis even for a modest-size system.

Petri Nets

- A Petri Net is a graph with the following properties:
  - Directed
  - Weighted
  - Bipartite

Graph Reminder

- G = (V, E)
  - V = (v_1,..,v_n) Set of Vertices (or Nodes)
  - E = (e_1,..,e_m) Set of Edges (or Arcs)
    - where $e_k = e(v_i,v_j)$



Circle = Node.
Line = Edge.

This graph is un-directed:
$e(v_i,v_j) = e(v_j,v_i)$

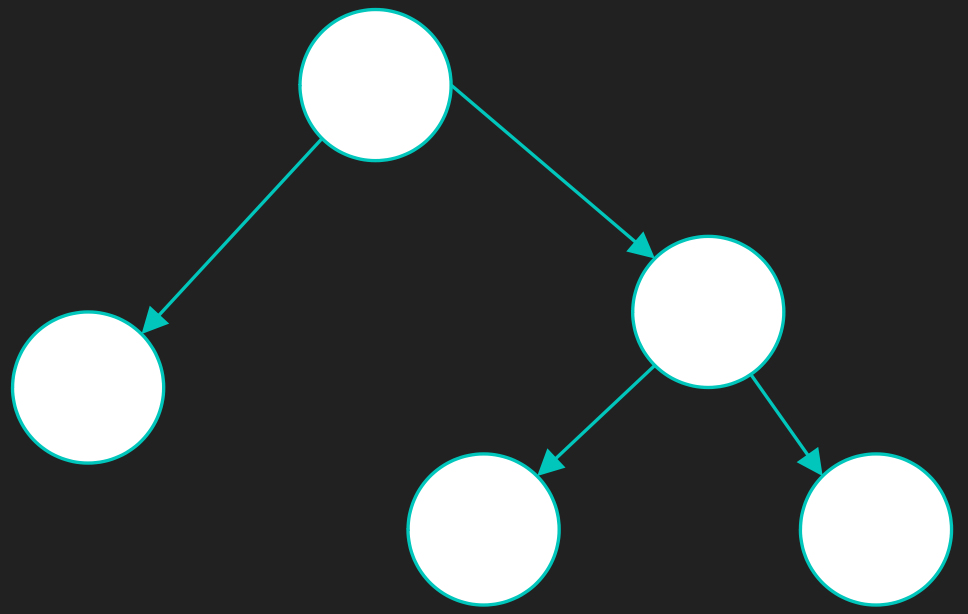
Graph Reminder

- G = (V, E)

  V = (v_1,..,v_n) Set of Vertices (or Nodes)

  E = (e_1,..,e_m) Set of Edges (or Arcs)

  where $e_k = e(v_i,v_j)$



Circle = Node.
Arrow = Edge.

This graph is **directed**:
$e(v_i,v_j) \neq e(v_i,v_j)$

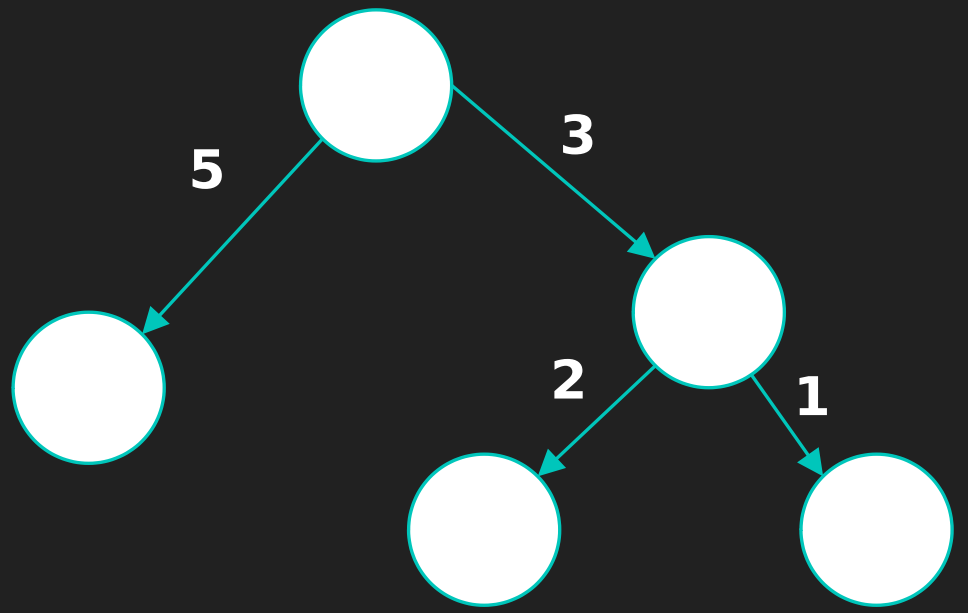
Graph Reminder

- G = (V, E)

  V = (v_1,..,v_n) Set of Vertices (or Nodes)

  E = (e_1,..,e_m) Set of Edges (or Arcs)

  where $e_k = e(v_i,v_j)$

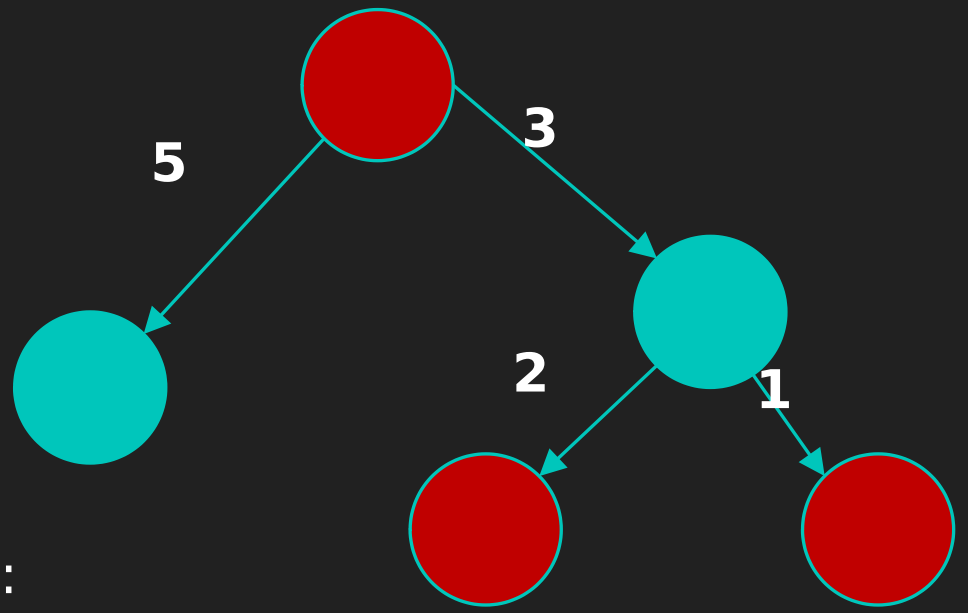


Circle = Node.
Arrow = Edge.

This graph is **directed**: $e(v_i,v_j) \neq e(v_i,v_j)$ and **weighted**.

Graph Reminder

- G = (V, E)
  - V = $(v_1,..,v_n)$ Set of Vertices (or Nodes)
  - E = $(e_1,..,e_m)$ Set of Edges (or Arcs)
    - where $e_k = e(v_i,v_j)$



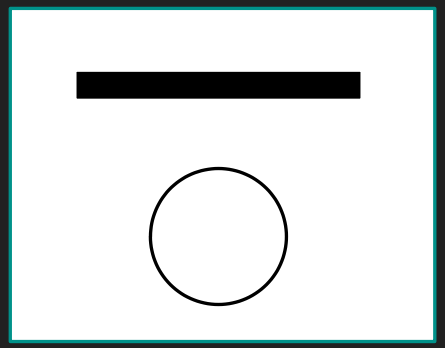
Circle = Node.
Arrow = Edge.

This graph is **directed**:
$e(v_i,v_j) \neq e(v_i,v_j)$, **weighted** and **bipartite**.

Bipartite:
Node split into two distinct sets (blue/red).
Nodes from same set not directly connected.

Petri Net

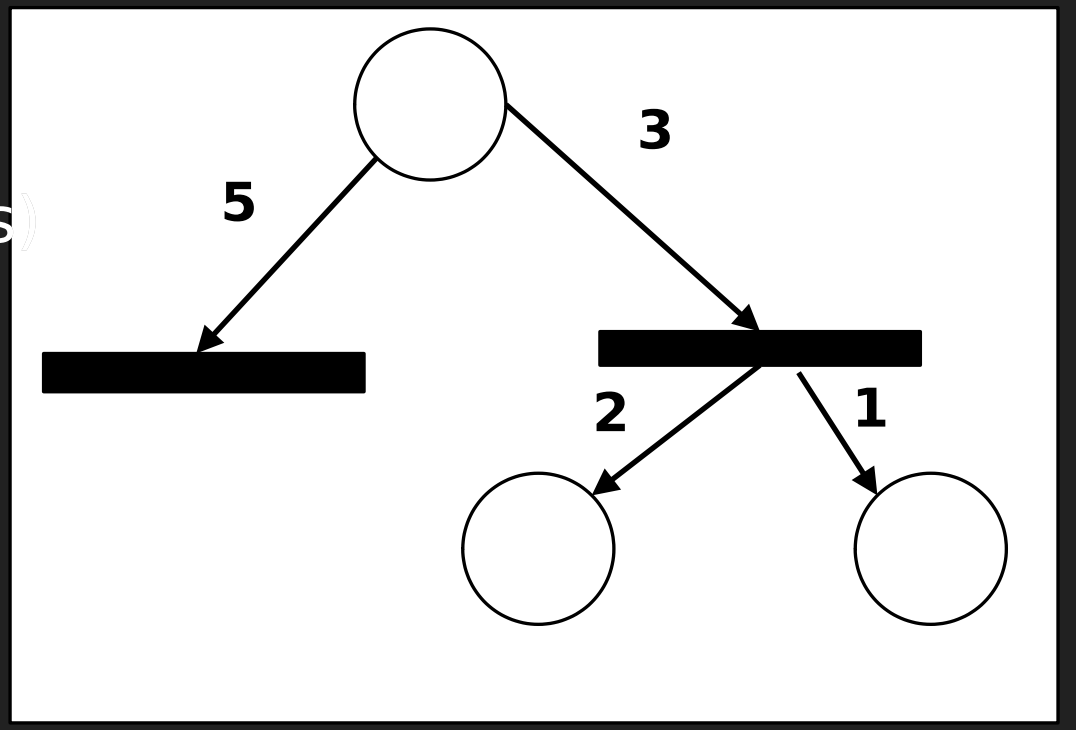
- Petri Net consists of two types of nodes:
  - Transitions
    - Represented by bar.
  - Place
    - Represented by circle.



- As the graph is bipartite, edges only exist between Transitions and Place.
  - Not Place–Place or Transition-Transition

Petri Net

- G = (P, T, E)
  - P = (p_1,..,p_n) Set of Places
  - T = (t_1,..,t_m) Set of Transitions
  - E = (e_1,..,e_q) Set of Edges (or Arcs)
    - where e_k = e(p_i,t_j) or e(t_i,p_j)

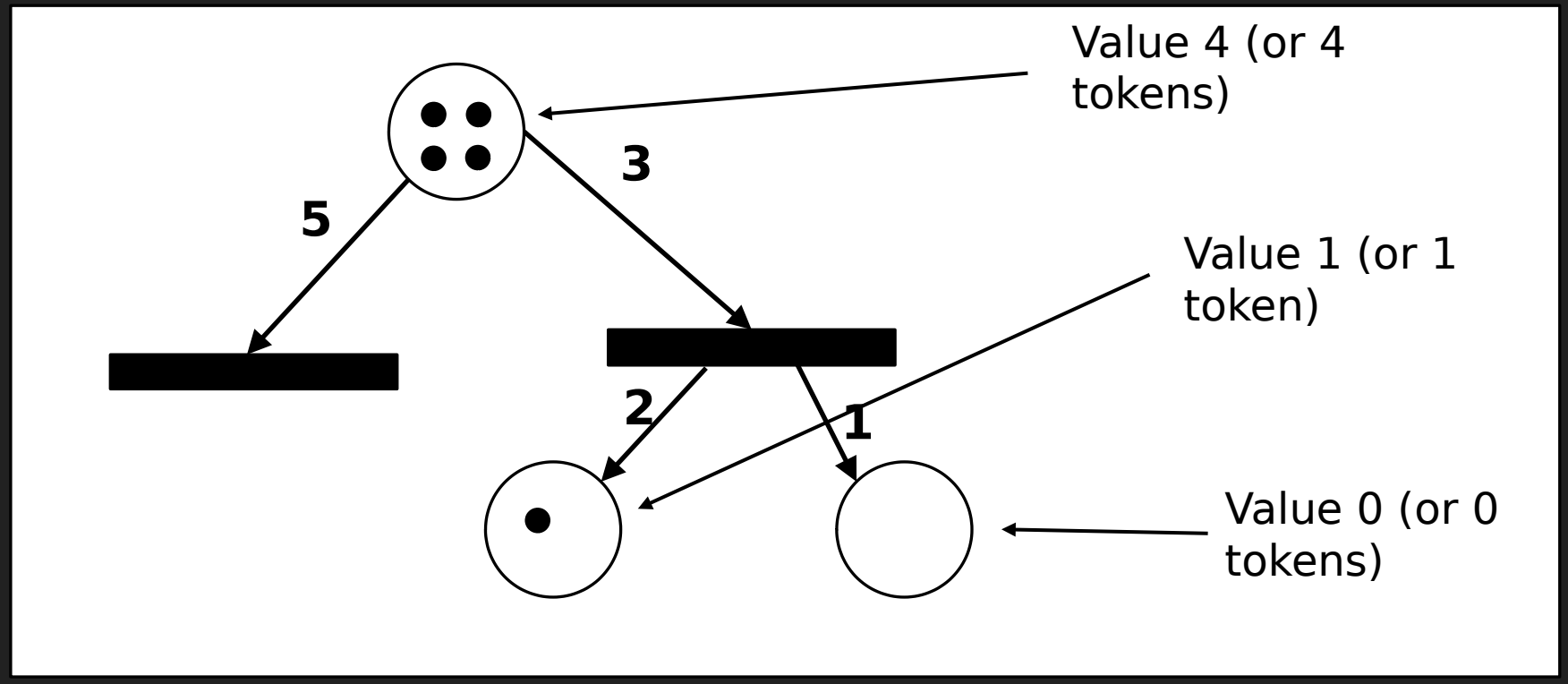


Petri Net

- A marking (state) assigns to each place a nonnegative integer.
- If a marking assigns to place *p* a nonnegative integer *k* , we say that *p* is marked with *k* tokens.
- Pictorially, we place *k* black dots (tokens) in place *p* .

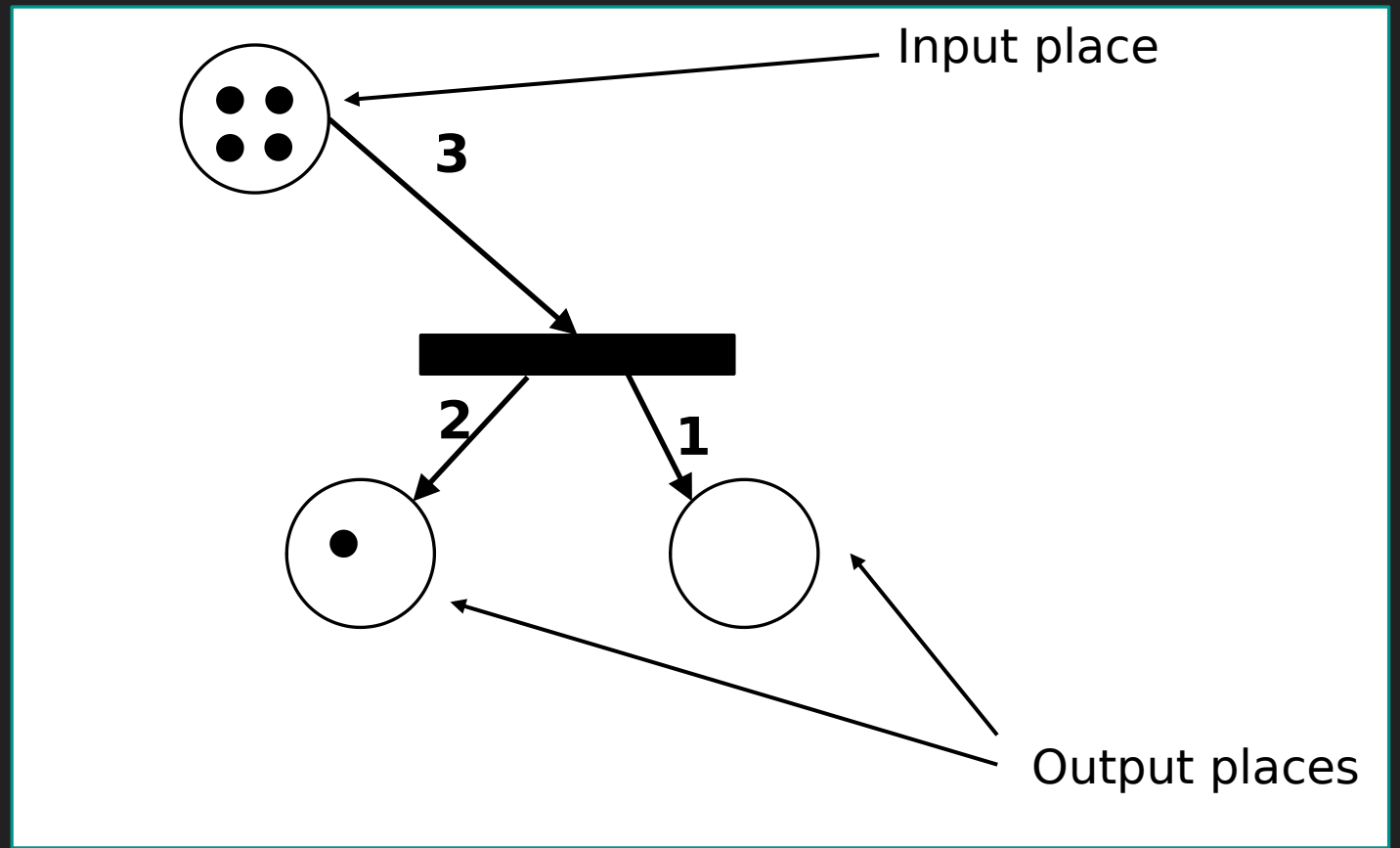
Petri Net

- Has an initial state or marking $\mathbf{M_o}$



Petri Net

- A transition has input places and output places.



Petri Nets

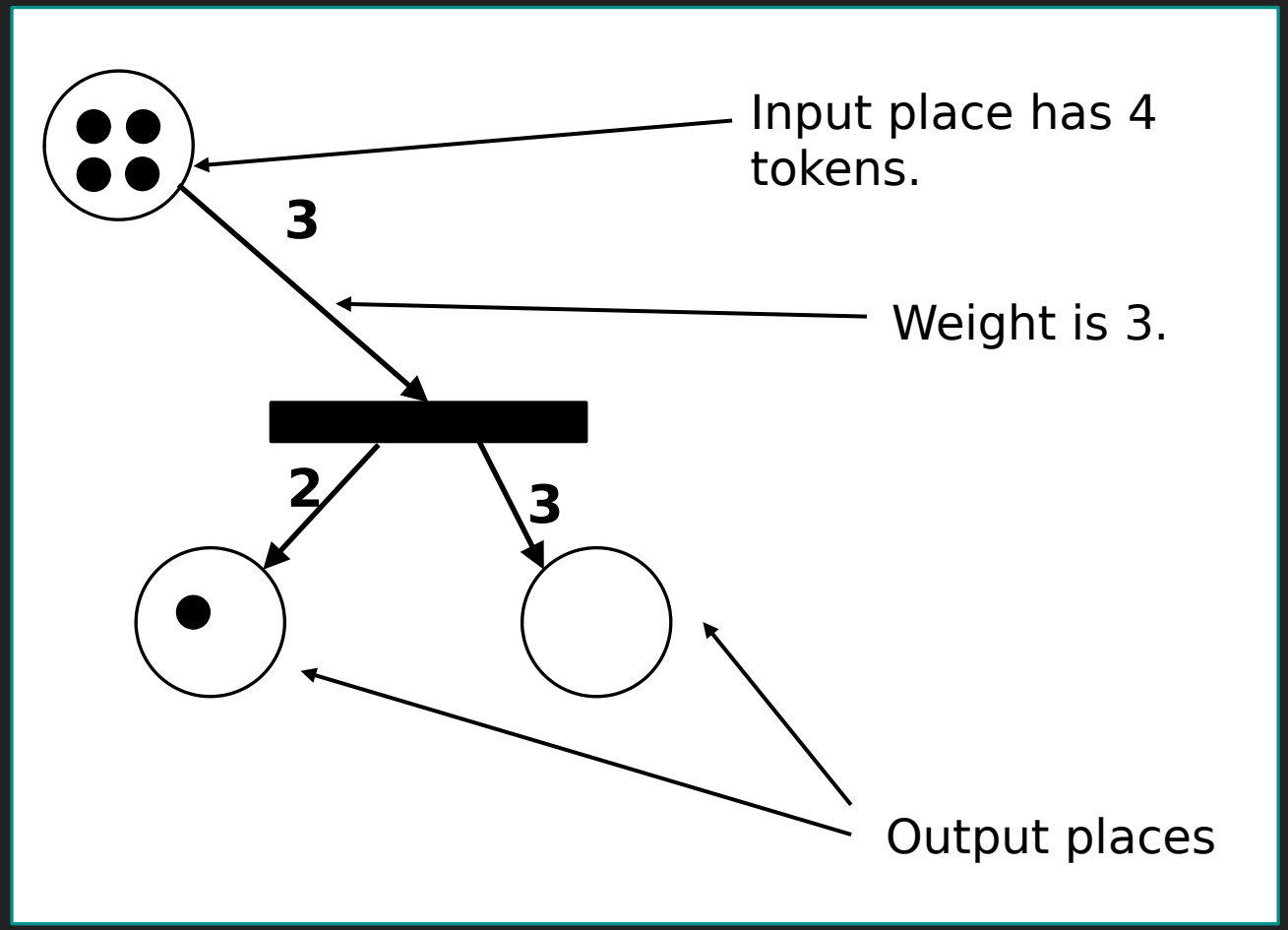
- A transition can have multiple input places.
- A transition can have multiple output places.

Transition Enabling and Firing

- There is only one rule to learn:
  - A transition is enabled if there are at least as many tokens in each place precursing a transition as the weights on the edges.
  - Firing removes the number of tokens from the input places specified by incoming weight and adds the number of tokens to the output places specified by weight.

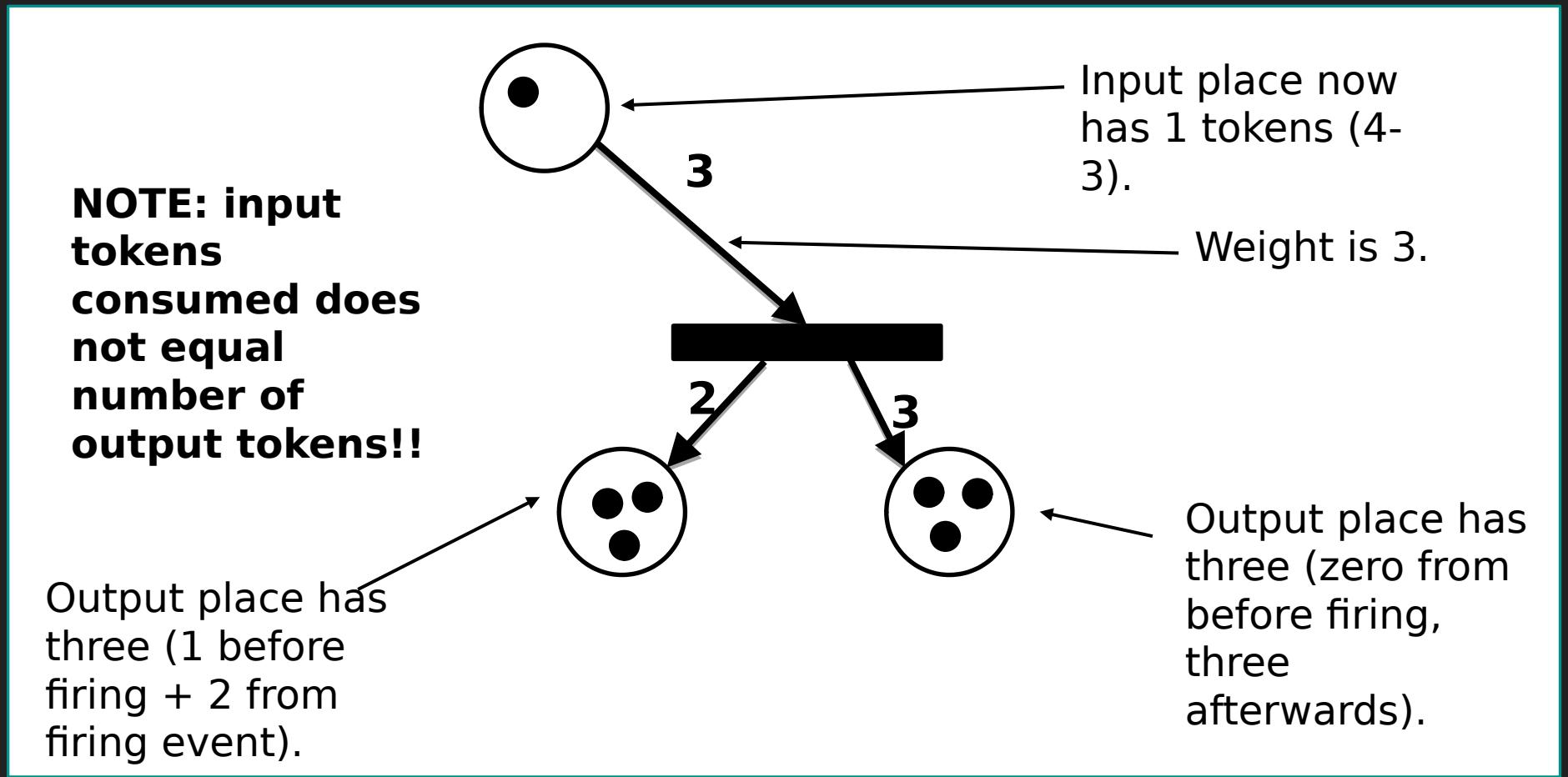
Firing Example

- Before Firing…



Firing Example

- After Firing…

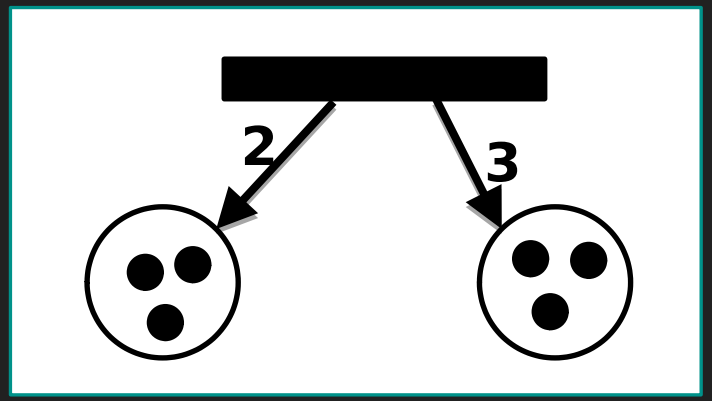


Firing

- Transitions represent events.
- Places represent preconditions and postconditions on events.
- Tokens could represent a number of things e.g. a lock or the number of available resources.

Special Transitions

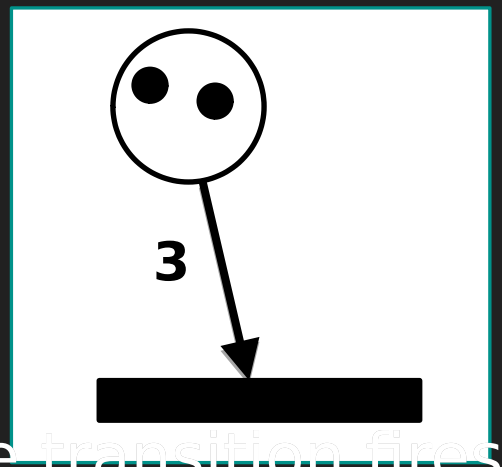
- A transition without any input places is called a source.



- Every time the above transition fires it will produce five tokens but consumes none. It is always enabled.

Special Transitions

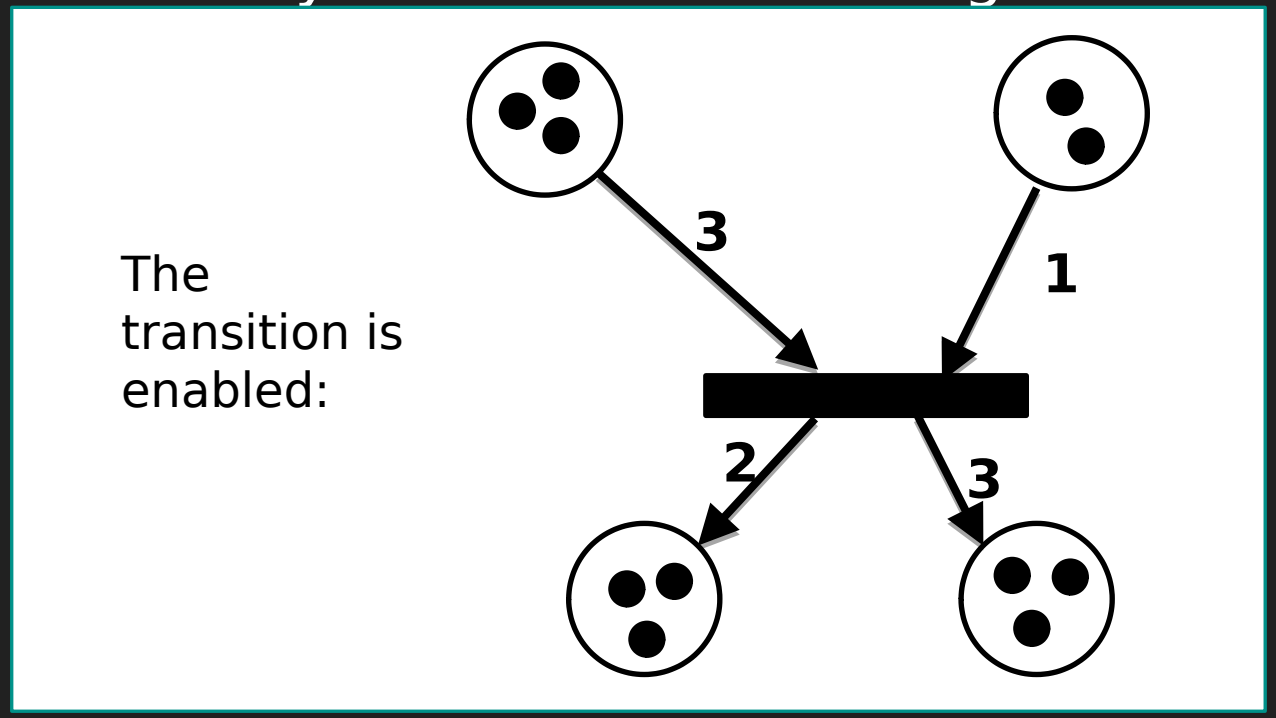
- A transition without any output places is called a sink.



- Every time the above transition fires it will consume three tokens.

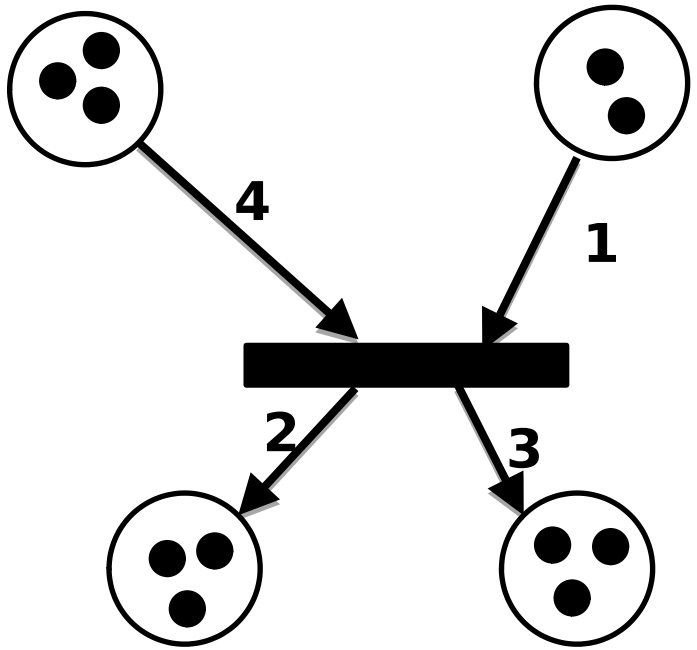
Multiple Inputs

- If a transition has multiple inputs it is only enabled if all inputs have at least as many tokens as each weight.



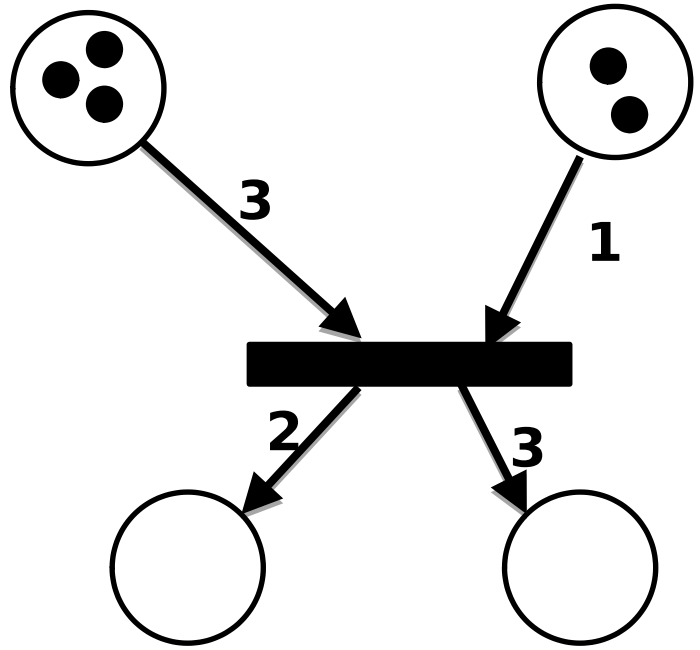
Multiple Inputs

The transition is not enabled:



Multiple Inputs

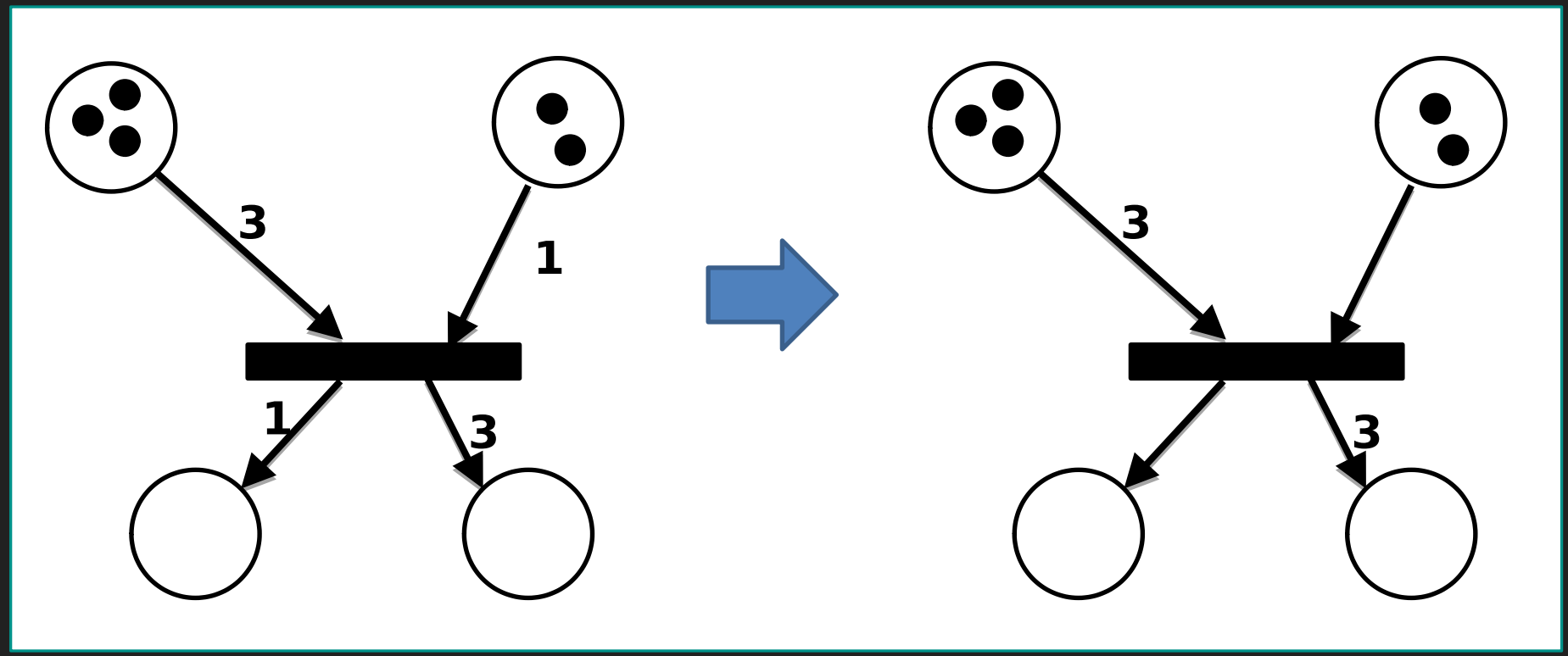
The transition is ?



Note: The state of the output places do not affect the ability of the transition to fire.

Petri Nets

- If an arc has a weight of 1 we simply omit the weight:

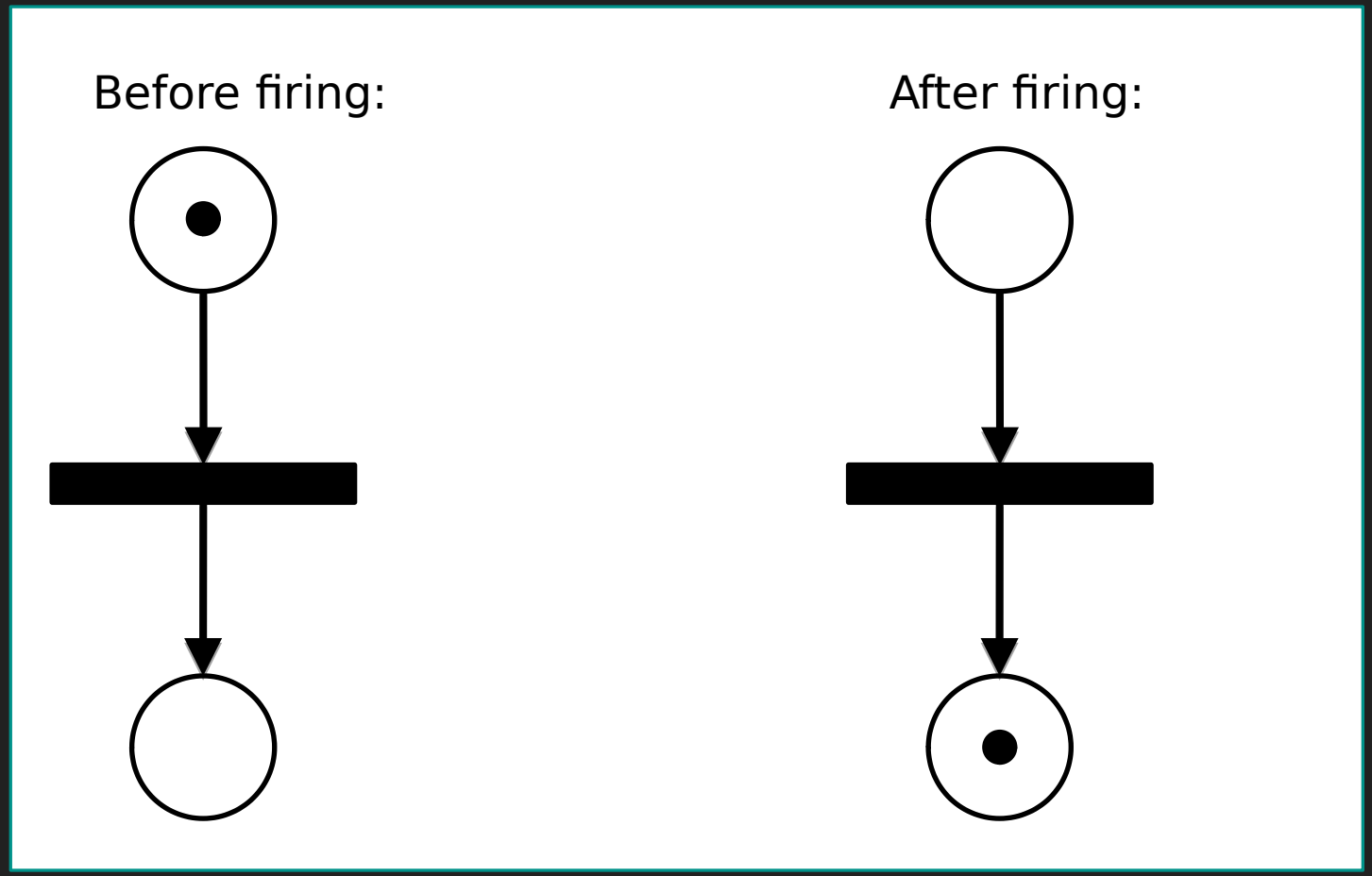


When do transitions fire?

- If a transition is enabled it can fire at any time.
- Petri Nets that we've covered have no concept of time (more on this later).
- However, we can update the model as a set of sequences.

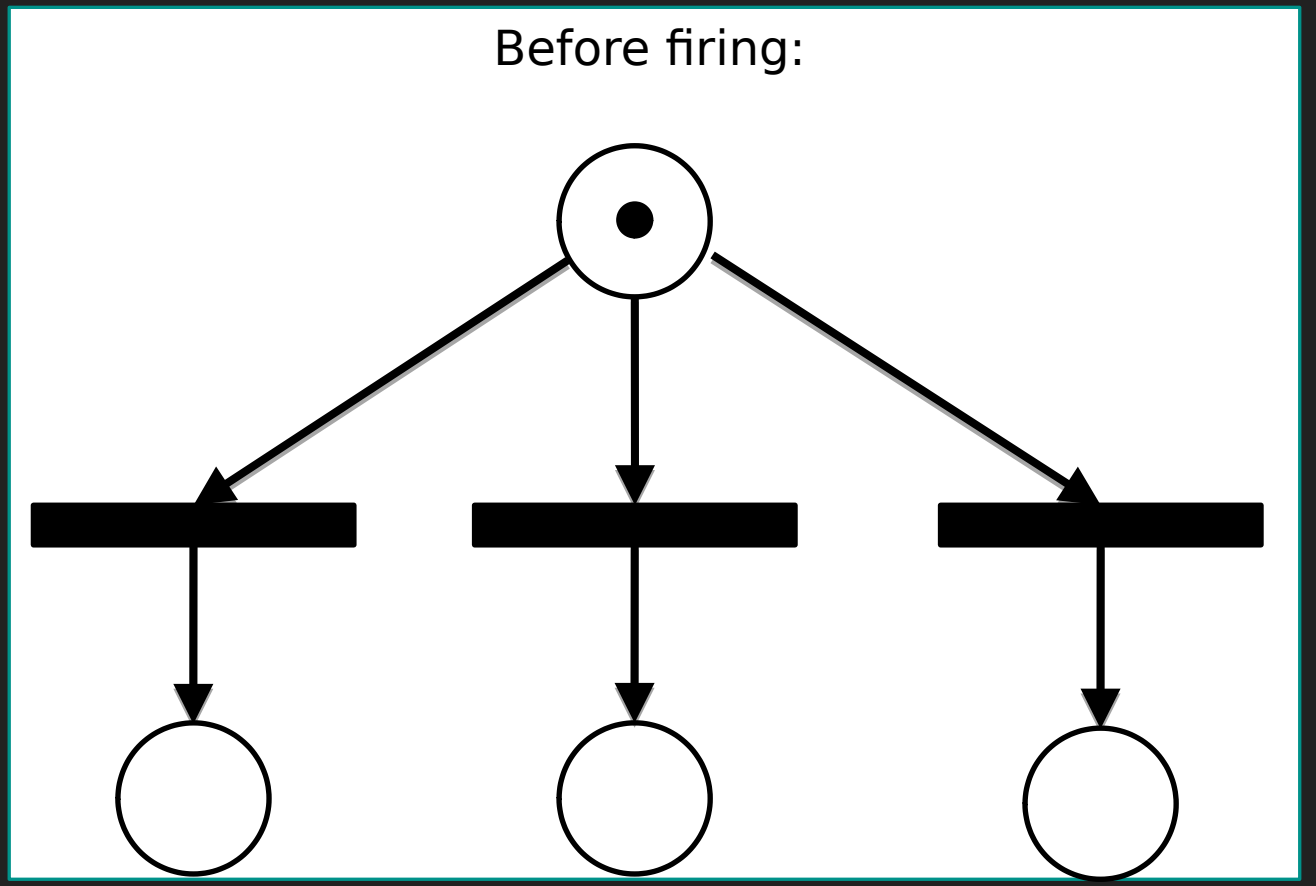
Primitive Petri Net Structures

- **Sequence**



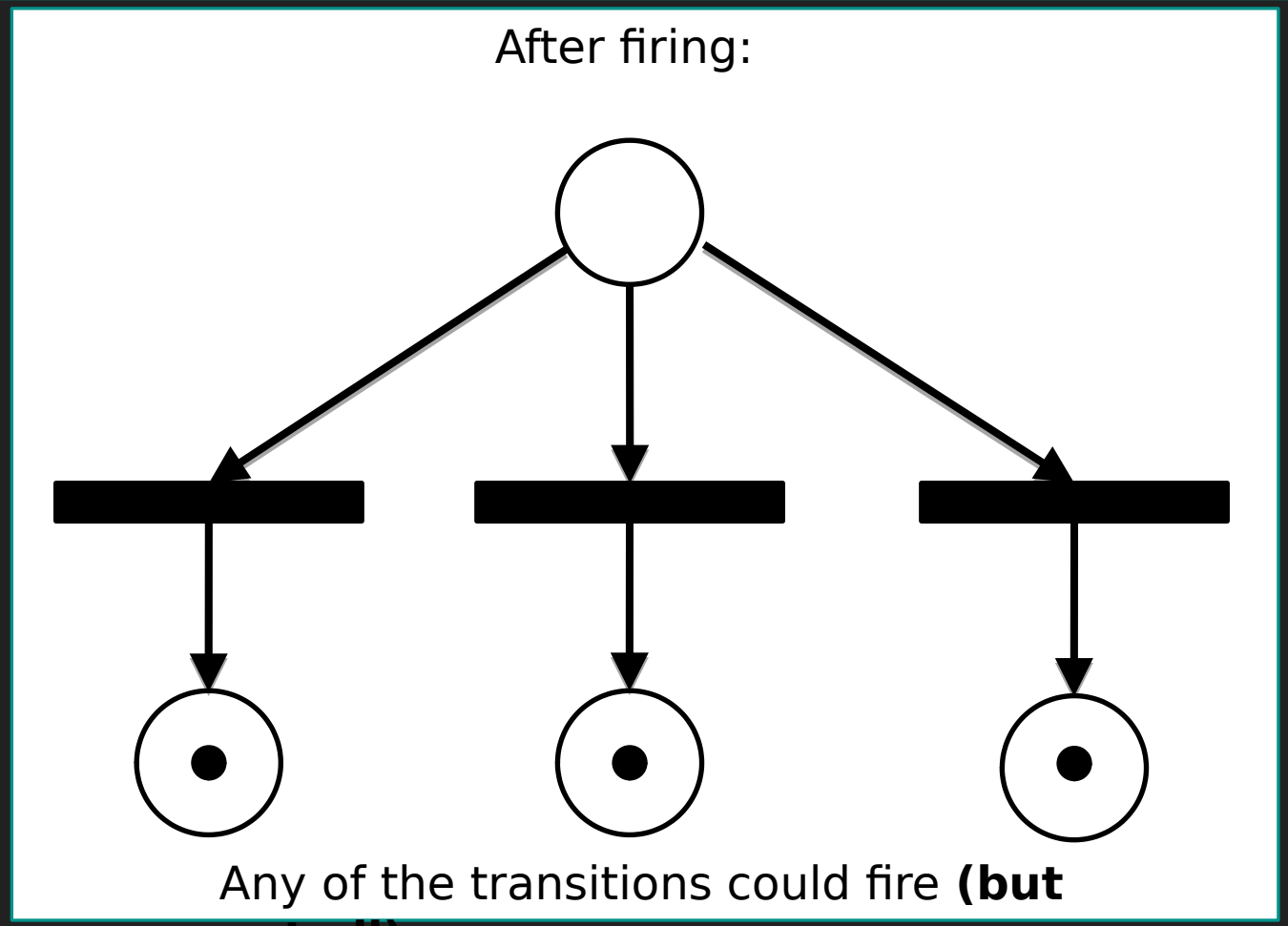
Primitive Petri Net Structures

- Conflict



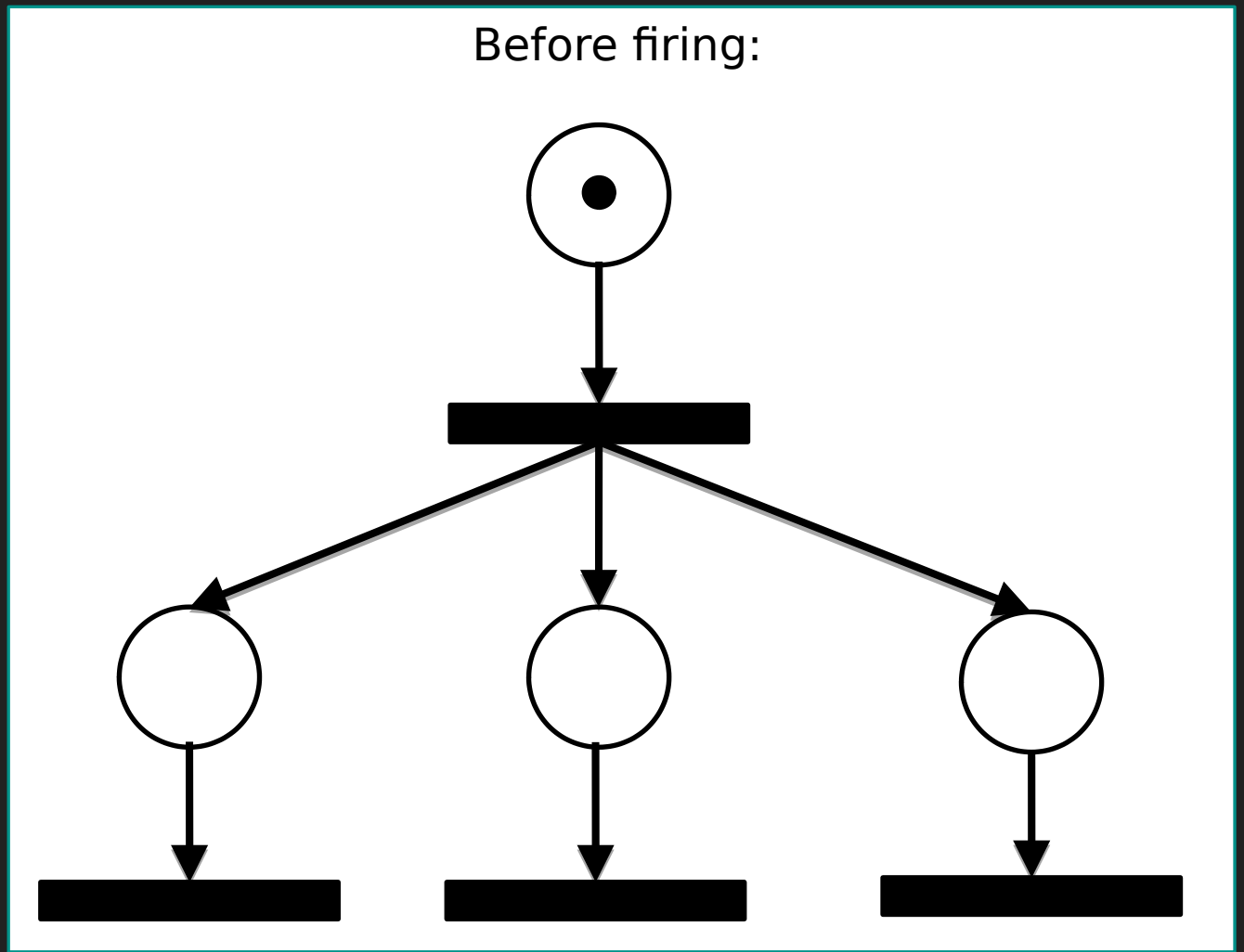
Primitive Petri Net Structures

- **Conflict**



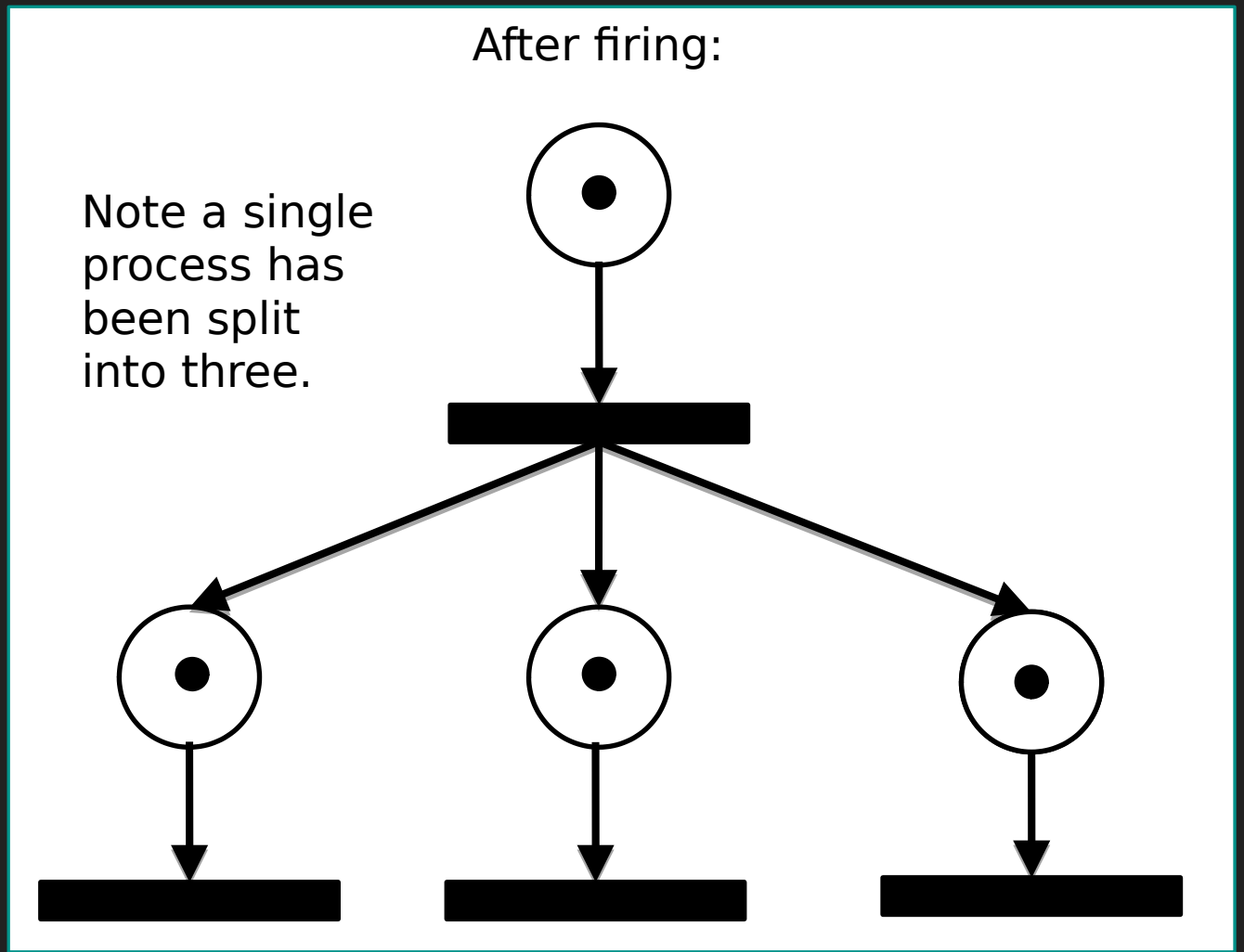
Primitive Petri Net Structures

- Concurrency



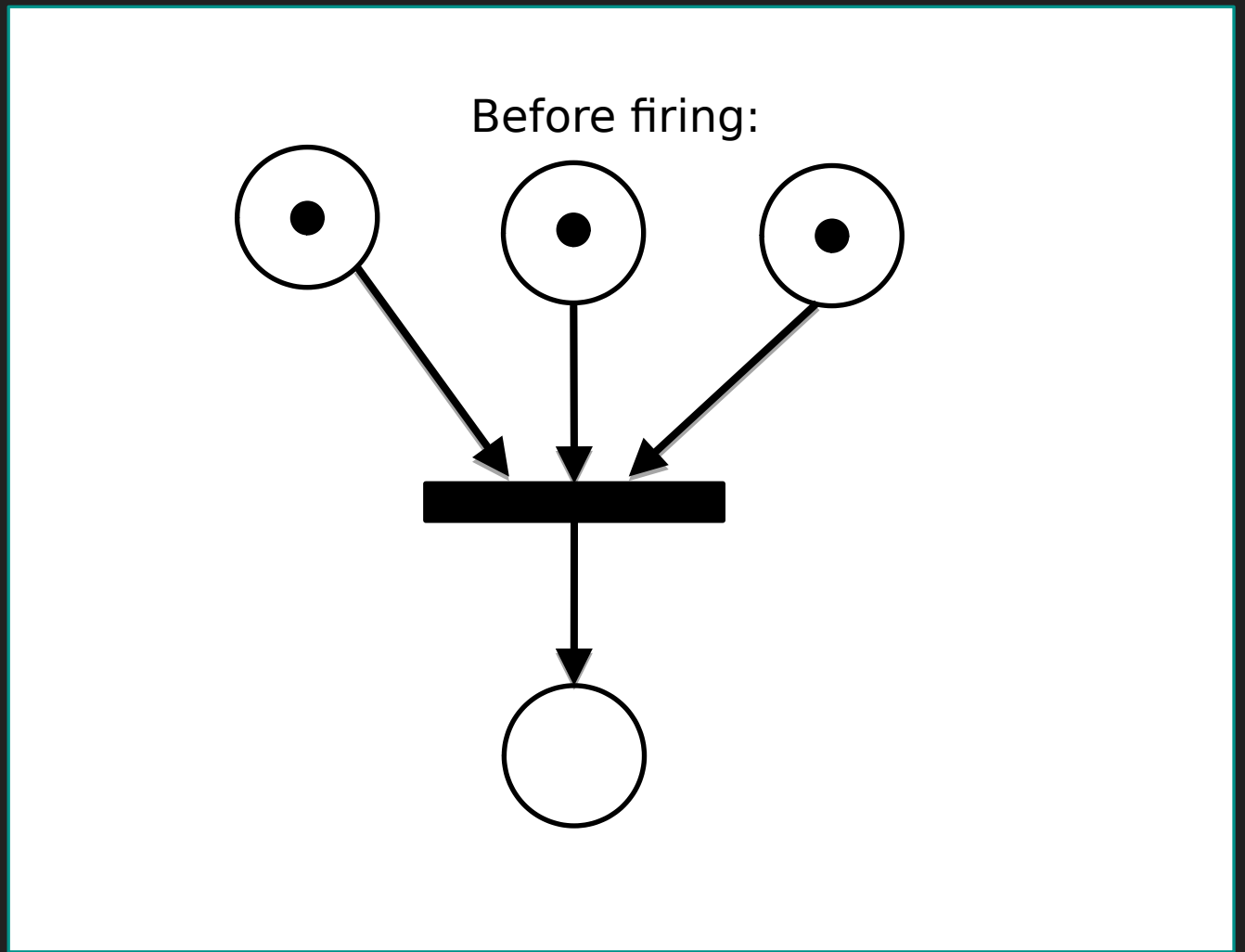
Primitive Petri Net Structures

- **Concurrency**



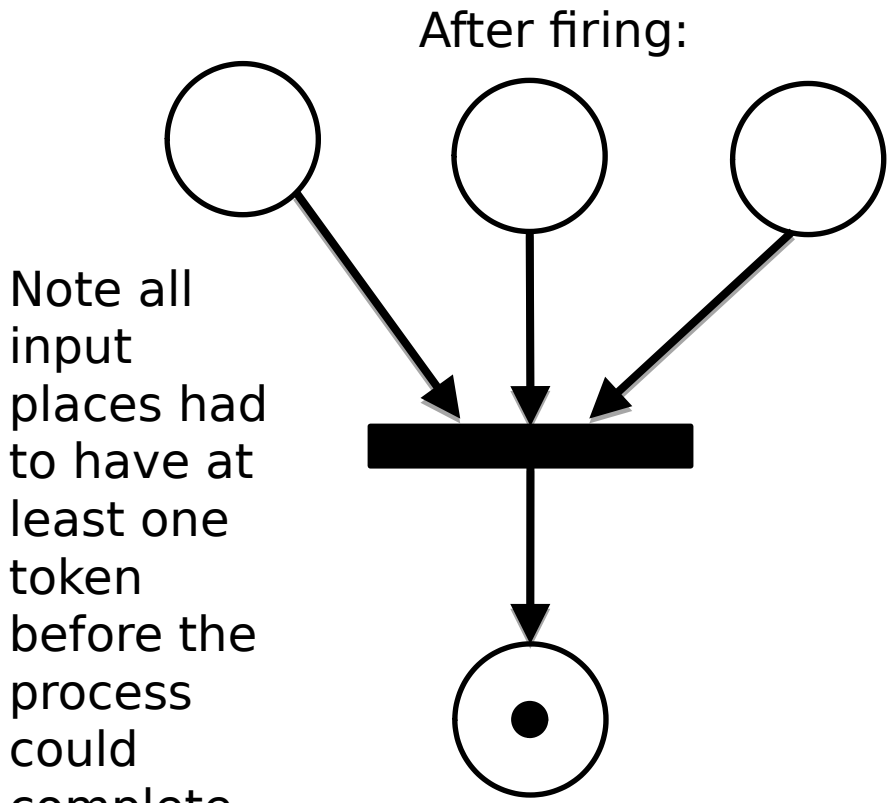
Primitive Petri Net Structures

- **Synchronization**



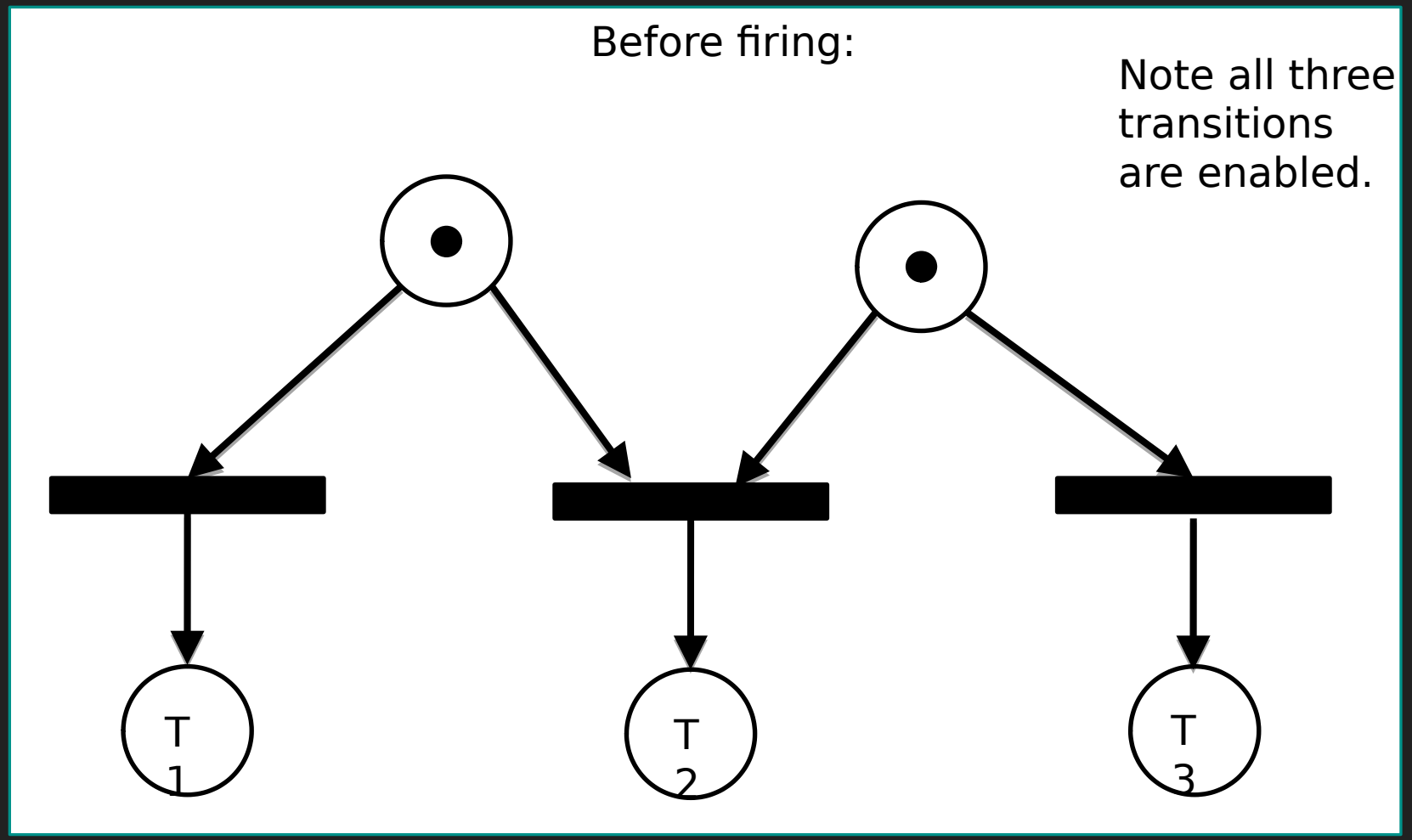
Primitive Petri Net Structures

- **Synchronization**



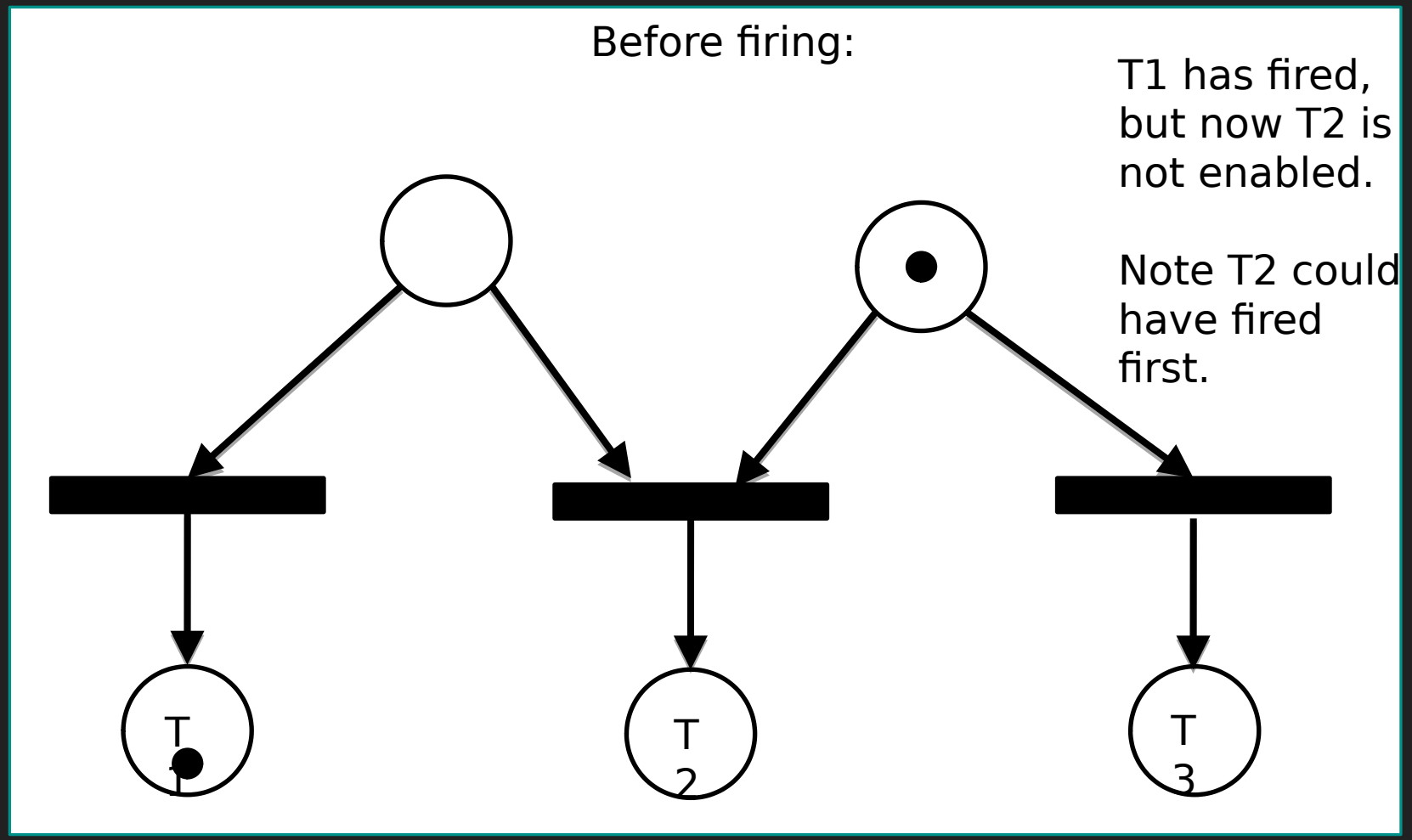
Primitive Petri Net Structures

- **Confusion**



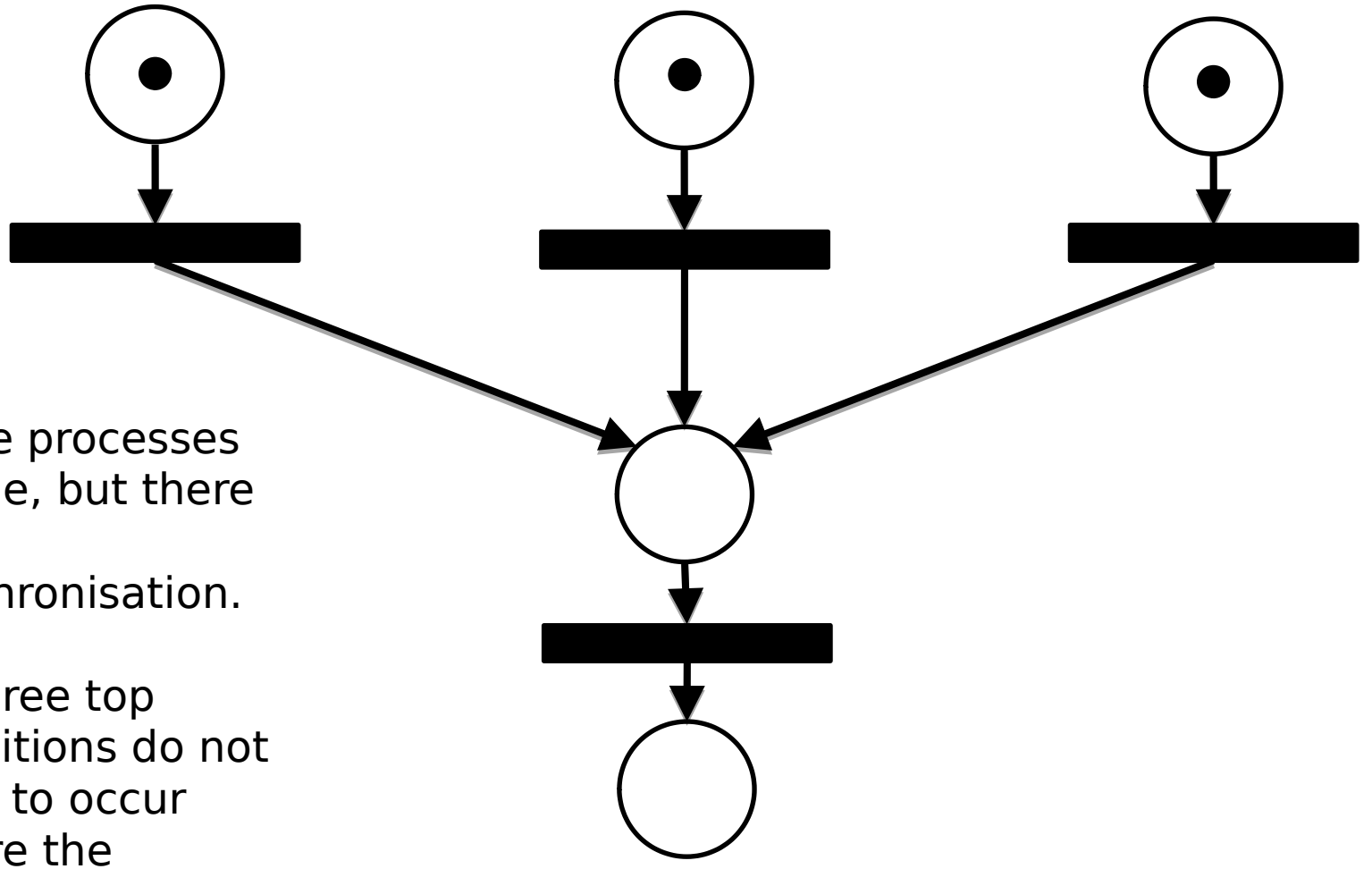
Primitive Petri Net Structures

- Confusion



Primitive Petri Net Structures

- Merging

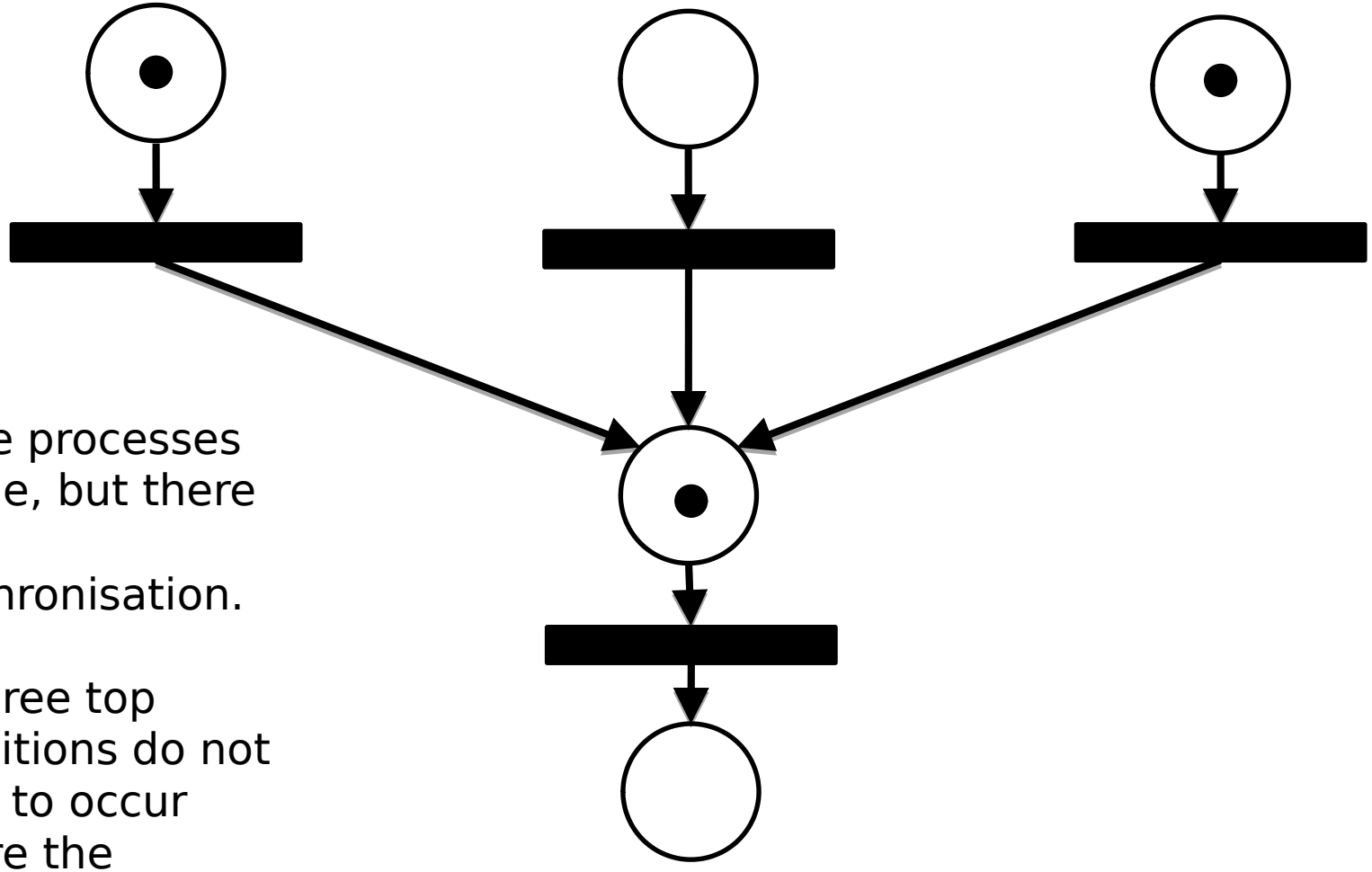


Three processes merge, but there is no synchronisation.

All three top transitions do not need to occur before the bottom transition

Primitive Petri Net Structures

- Merging

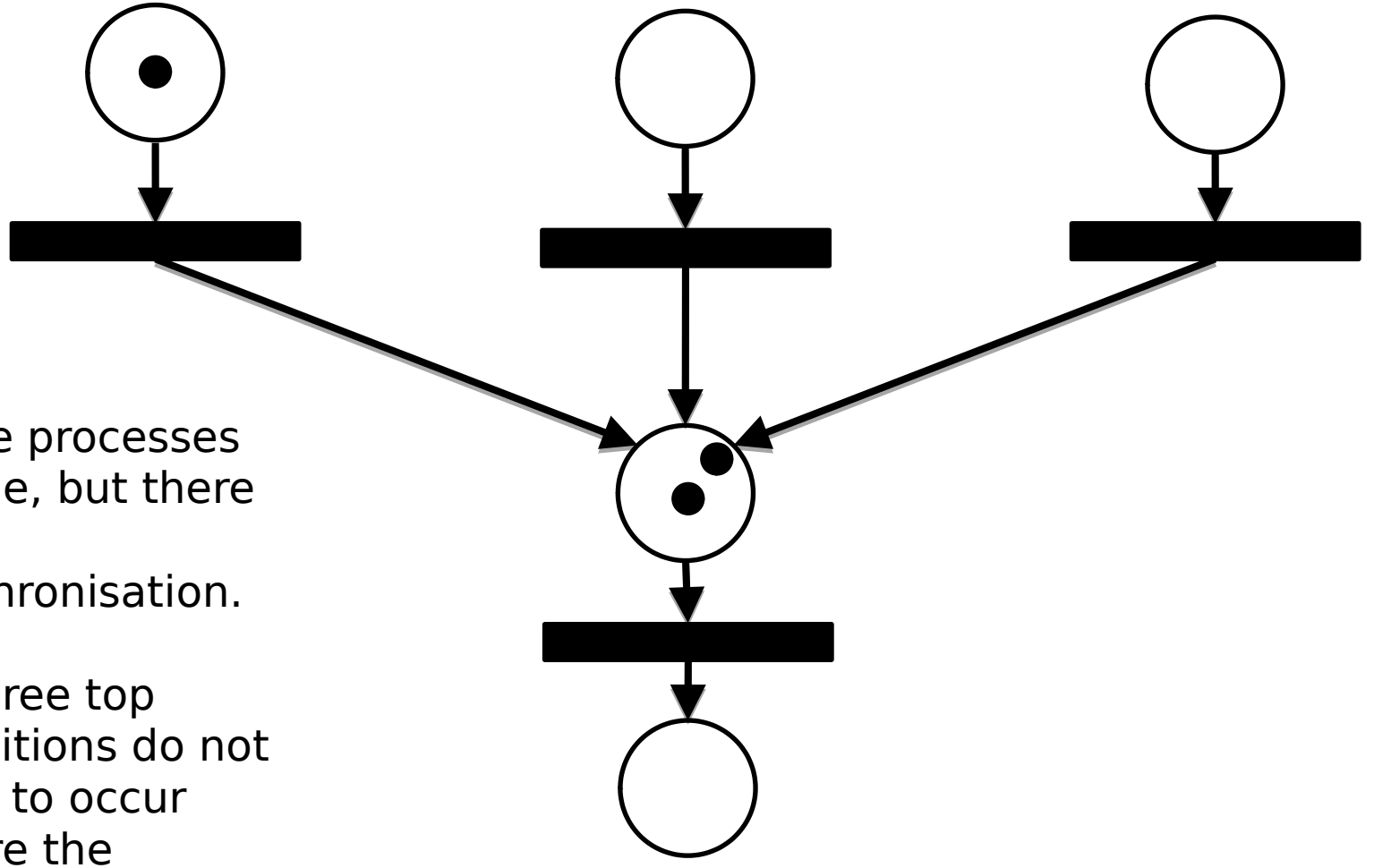


Three processes merge, but there is no synchronisation.

All three top transitions do not need to occur before the bottom transition

Primitive Petri Net Structures

- Merging

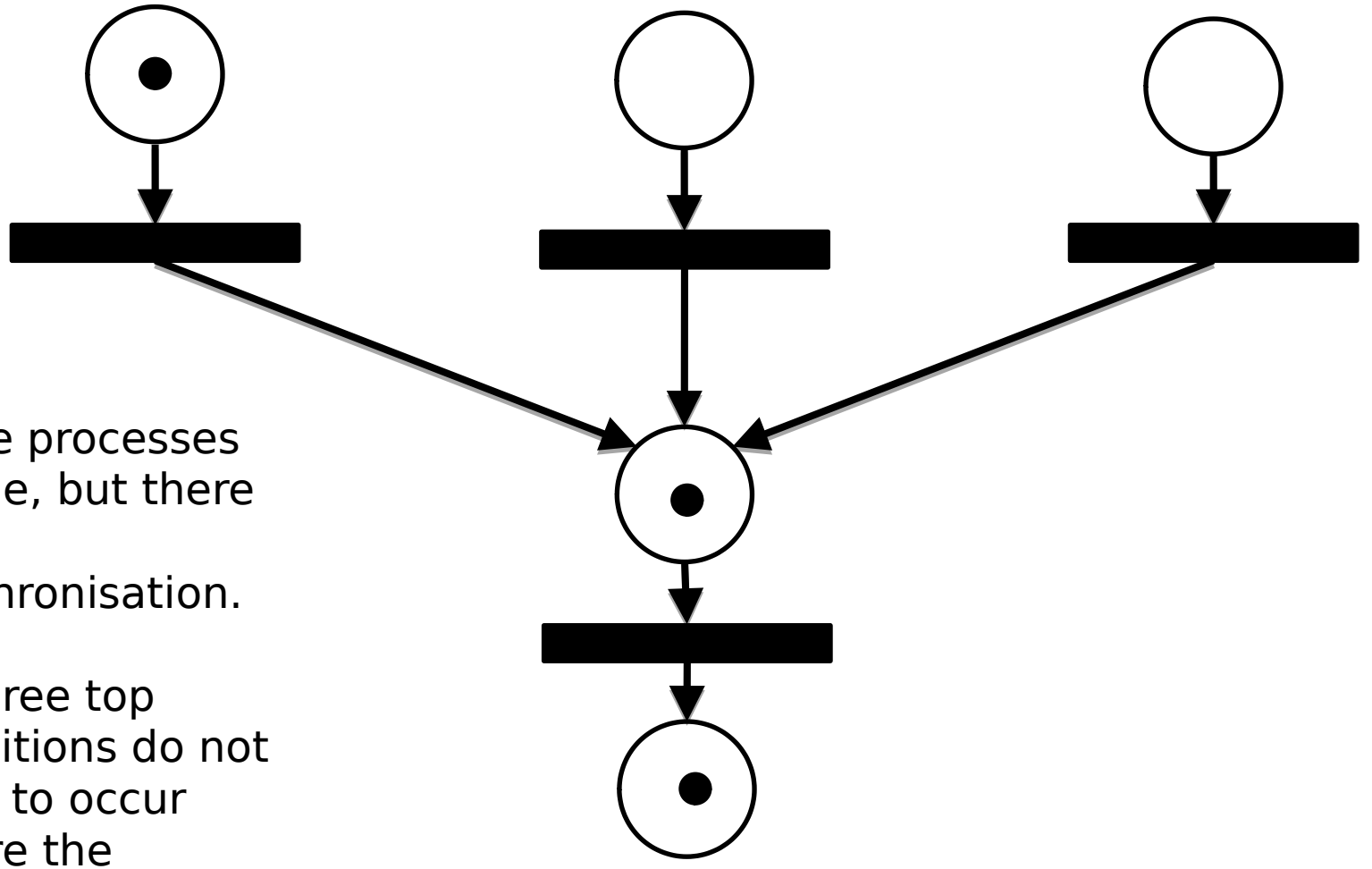


Three processes merge, but there is no synchronisation.

All three top transitions do not need to occur before the bottom transition

Primitive Petri Net Structures

- Merging

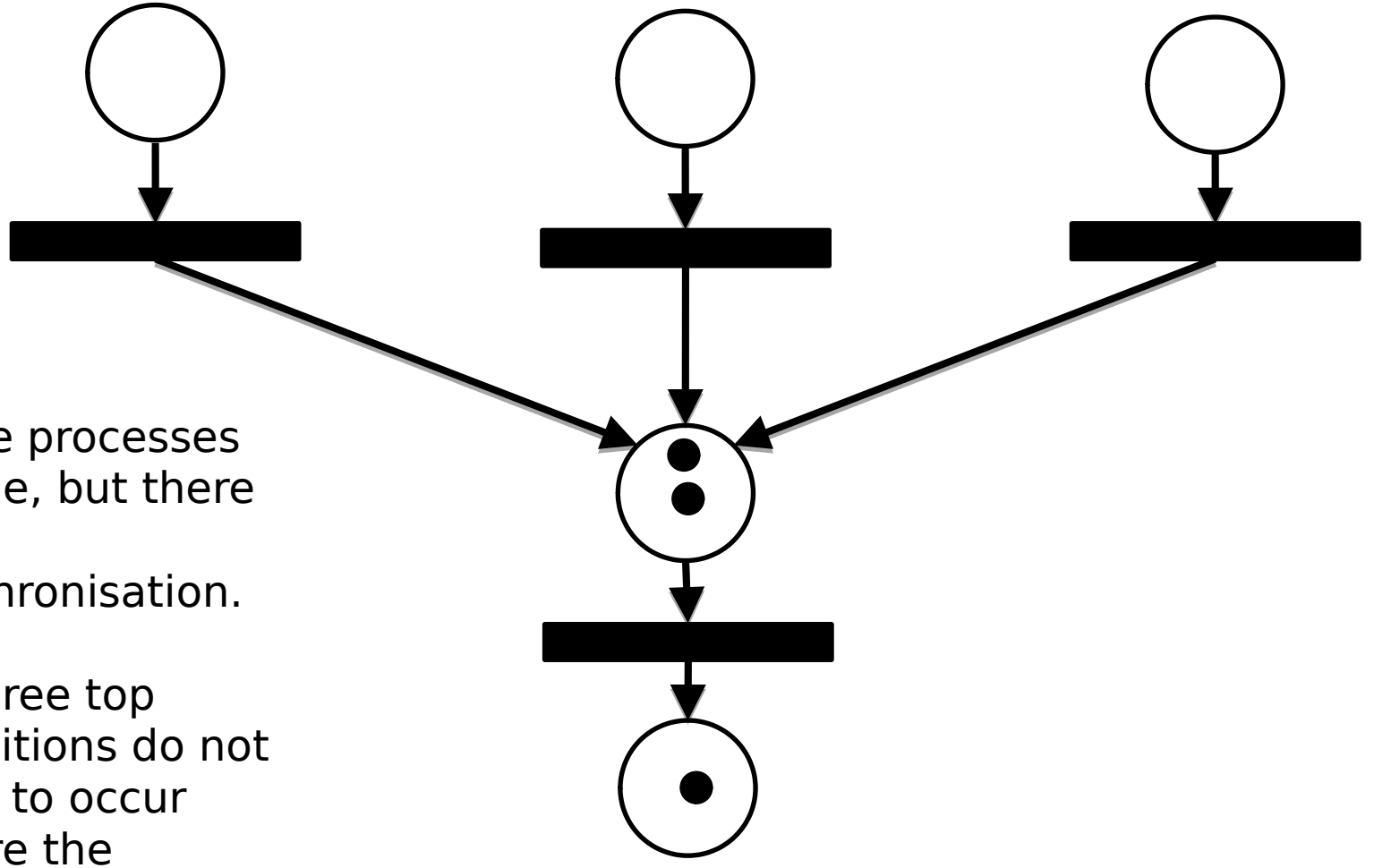


Three processes merge, but there is no synchronisation.

All three top transitions do not need to occur before the bottom transition

Primitive Petri Net Structures

- Merging

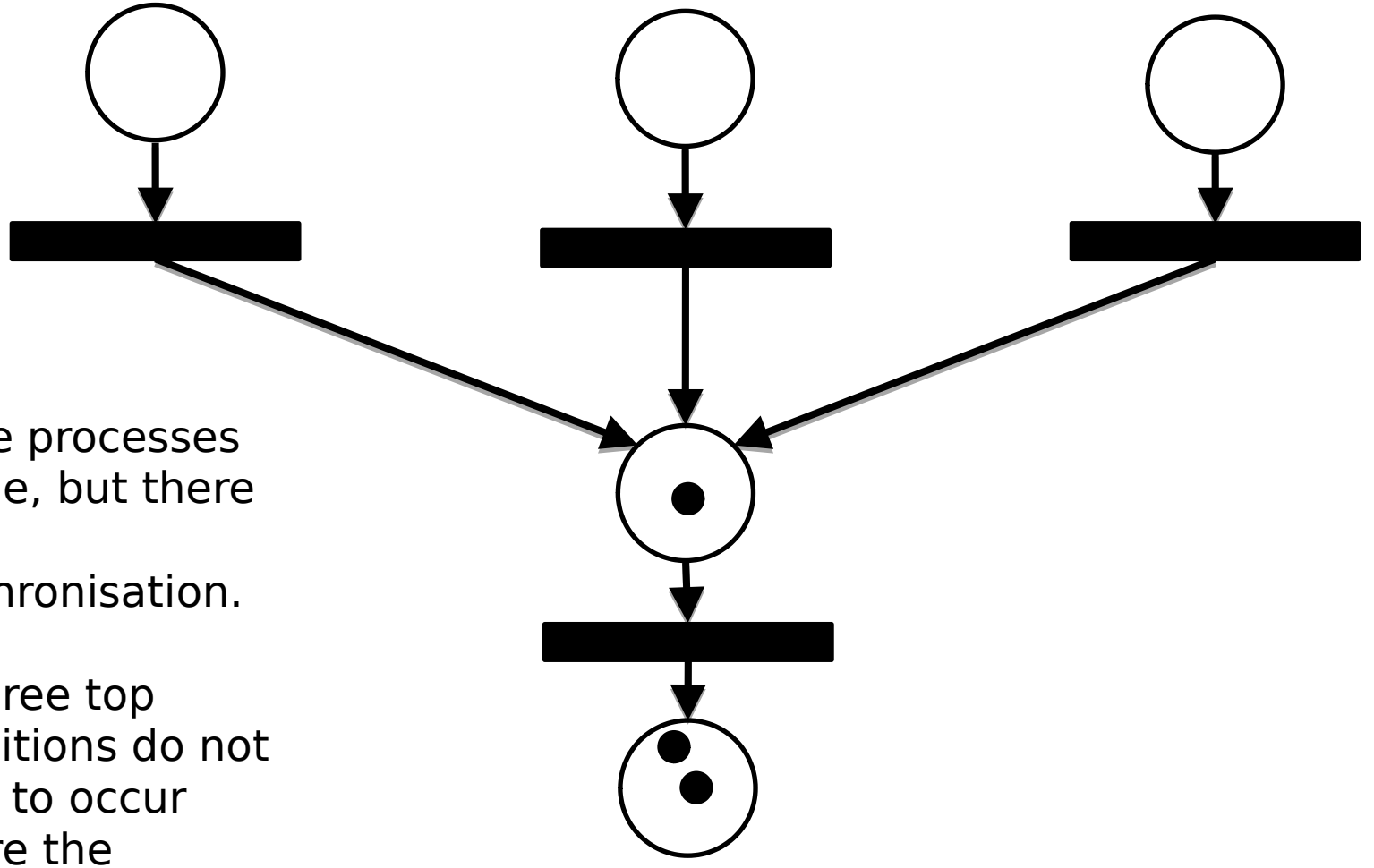


Three processes merge, but there is no synchronisation.

All three top transitions do not need to occur before the bottom transition

Primitive Petri Net Structures

- Merging

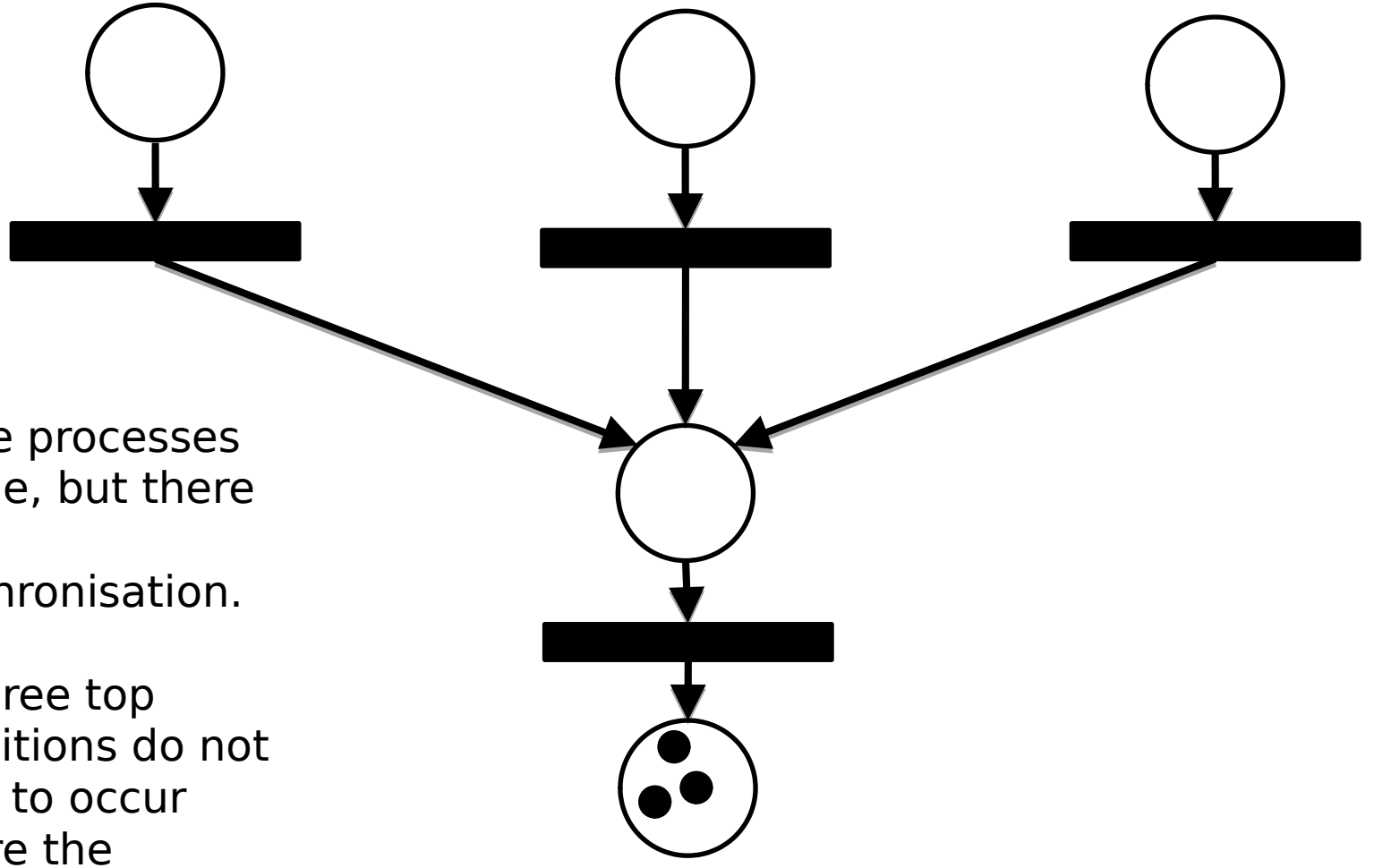


Three processes merge, but there is no synchronisation.

All three top transitions do not need to occur before the bottom transition

Primitive Petri Net Structures

- Merging



Three processes merge, but there is no synchronisation.

All three top transitions do not need to occur before the bottom transition

Summary

- We have learnt the basics of Petri Nets.
- Next week we will look at using them to model concurrent systems.