U O W

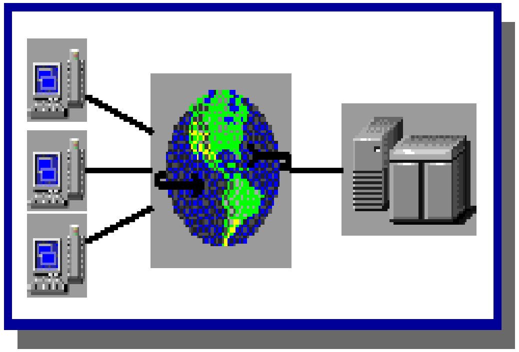
Software Requirements, Specifications and Formal Methods

Dr. Shixun Huang

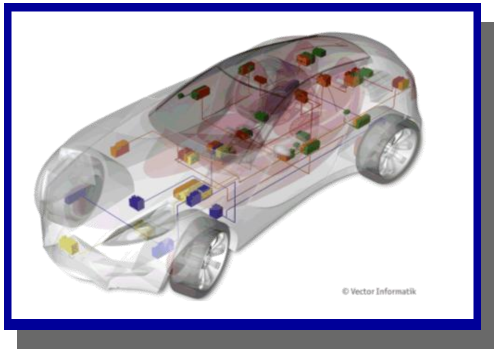
UNIVERSITY OF WOLLONGONG AUSTRALIA

Concurrent systems

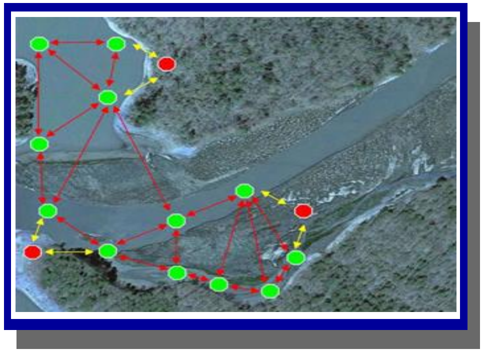
- Most modern IT systems are distributed and concurrent:



Internet and WWW



Modern car



Sensor network

Concurrent systems are difficult to design

- They possess concurrency and non-determinism.

- The execution may proceed in many different ways, e.g. depending on:
  - Whether messages are lost during transmission.
  - The scheduling of processes.
  - The time at which input is received from the environment.

- Concurrent systems have an astronomical number of possible executions.
  - It is easy for the designer to miss important interaction patterns.
  - This may lead to gaps or malfunctions in the system design.

Concurrent systems are often critical

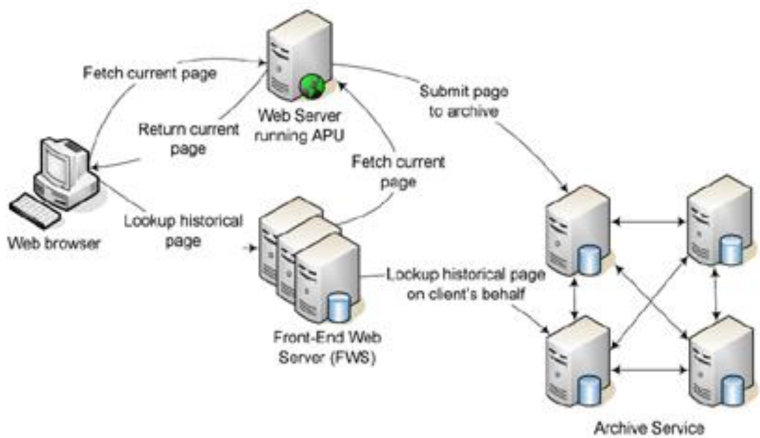
- For many concurrent systems it is essential that they work correctly from the very beginning:
  - Nuclear power-plants.
  - Aircraft control systems.
  - Hospital life support equipment.
  - Computer networks.
  - Bank system.

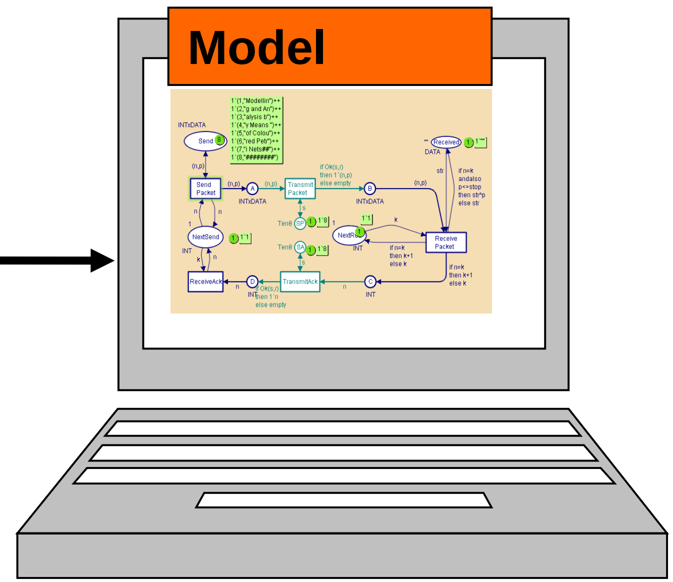
- To cope with the complexity of modern concurrent systems, it is crucial to provide methods that enable debugging and testing of central parts of the system designs prior to implementation and deployment.

Model based system development

- One way to approach the challenges posed by concurrent systems is to build a model.
- A model is an abstract representation which can be manipulated by means of a computer tool.

Concurrent system





- Using a model it becomes possible to investigate how the system will behave and the properties it will possess.

UNIVERSITY OF WOLLONGONG AUSTRALIA

Why do we make models?

- We make models to:
- Gain insight in the system which is being designed.
- Get ideas to improve the design.

- Models also help us:
- To ensure completeness in the design.
- Improve the correctness of the design.

Gain insight

- Modelling and simulation usually leads to significant new insights into the design and operation of the system.
  - The modeller gains an elaborate and more complete understanding of the system (e.g., compared to reading design documents).
  - The same applies to people for who witness a presentation of a model.

- The new insight often results in a simpler and more streamlined design.
  - By investigating a model, similarities can be identified that can be exploited to unify and generalise the design and make it more logical.
  - We may also get ideas to improve the usability of the system.

Completeness

- The construction of an executable model usually leads to a more complete specification of the design.
- Gaps in the specification of the system become explicit:
  - They will prohibit the model from being executed because certain parts are missing.
  - During simulation the designers and users will discover that certain expected events are impossible in the current state.
- Modelling leads to a more complete identification and understanding of the requirements to the system.
- Models can be used to mediate discussions among designers and users of the system.

UNIVERSITY OF WOLLONGONG AUSTRALIA

Correctness

- Modelling often reveals a number of design errors and flaws.
- It is possible to control the execution of a model (unlike the real system). This means that:
  - Problematic scenarios can be reproduced.
  - It is possible to check whether a proposed modification of the design works as intended.

- Simulating a number of different scenarios does <u>not</u> necessarily lead to correct designs:
  - There may be too many scenarios to investigate.
  - The modeller may fail to identify some important scenarios.
- However, a systematic investigation of scenarios often significantly decreases the number of design errors.

Introduction to Petri Net

- Invented by Dr. Carl Adam Petri
- First introduced in Petri's dissertation: "Kommunikation mit Automaten" (1962)
- Petri Net is a tool for the study of system, and a mathematical representation of the system.

Why Petri Net (PN)?

- Petri Nets give a graph-theoretic representation of the structure and the dynamics of a discrete event system.
- They show:
  - Communications patterns
  - Control patterns
  - Information flows
- They provide a mathematical framework for:
  - Analysis
  - Validation
  - Performance evaluation
- They focus on issues of
  - Concurrency
  - Asynchronous operations

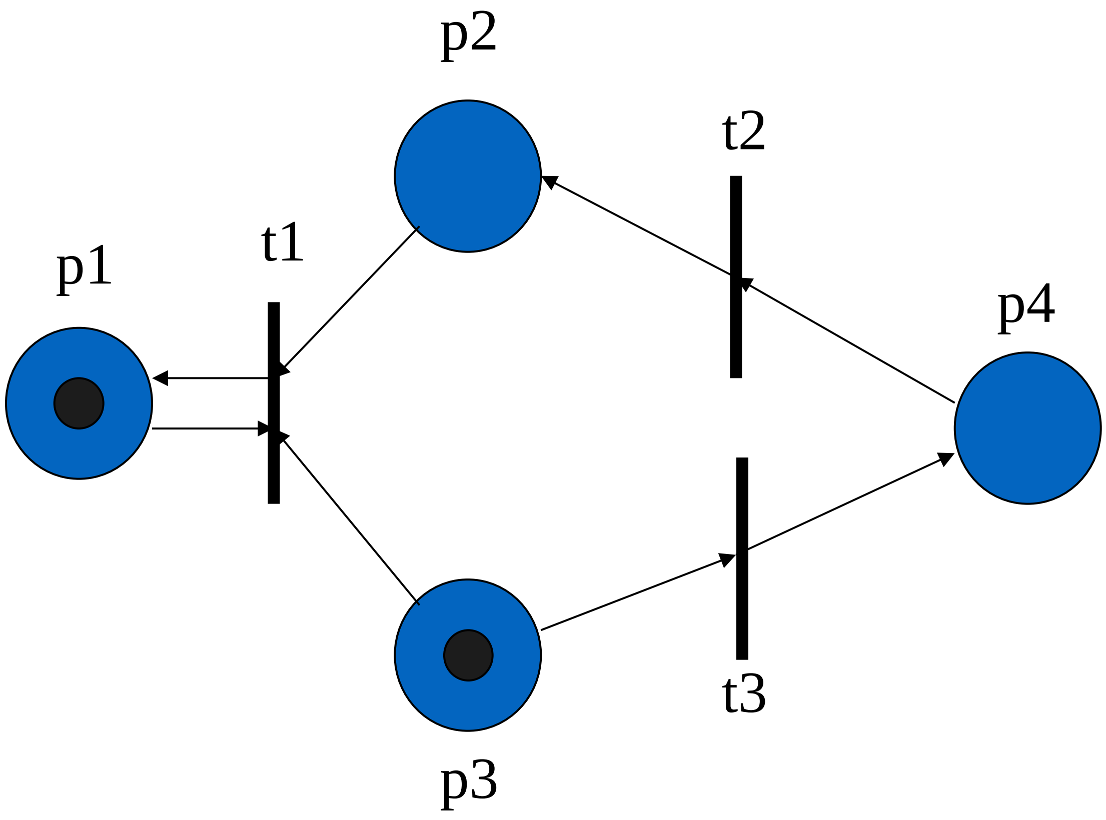
UNIVERSITY OF WOLLONGONG AUSTRALIA

PN definition

A Petri Net is a bipartite directed multi-graph, $G=(V,A)$, where $V=\{V_1, V_2, \ldots, V_s\}$ is a set of vertices and $A = \{a1, a2, \ldots, ar\}$ is a bag of directed arcs, $ai = \{vj, vk\}$ with $vj, vk \in V$.

The set V are partitioned into two disjoint sets P *(circle nodes, i.e. places)* and T *(bar nodes, i.e. transition)* such that $V = P \cup T$, $P \cap T = \emptyset$, and for each directed arc, $ai \in A$, if $ai = (vj,vk)$, the either $vj \in P$ and $vk \in T$ or $vj \in T$ and $V_k \in P$.

An example of PN



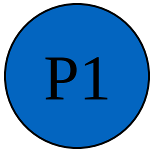
PN composition

A Petri Net is a bipartite directed multi-graph

– Bipartite: two types of nodes

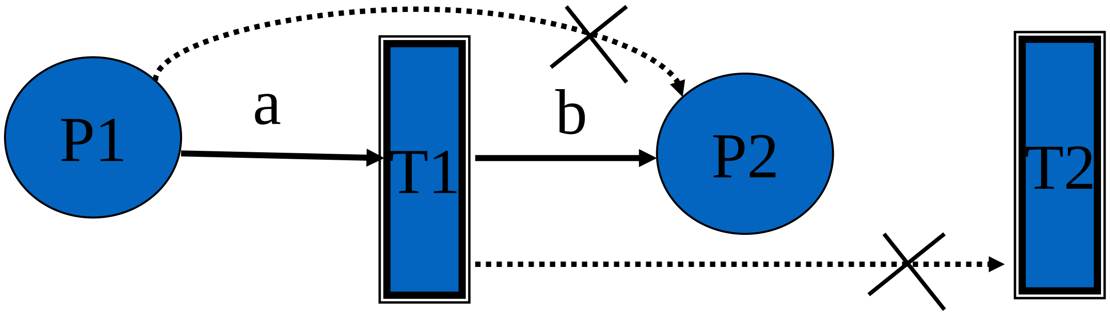
» Circle nodes denotes places

» Bar nodes denotes transitions



– Directed: the arcs that join two nodes are directed.

» Arcs can connect places to transitions and transitions to places only



Definition

A *Petri net structure*, C, is a four-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 set of transitions, $m \geq 0$.

The set of places and the set of transitions are disjoint, $P \cap T = \emptyset$.

$I: T \rightarrow P$ is the input function, a mapping from a transition to bags of places, and indicates the input places of a transition.

$O: T \rightarrow P$ is the output function, a mapping from a transition to bags of places, and indicates the output places of a transition.

Example

A Petri net structure:

C = (P,T,I,O)   P={p1,p2,p3,p4,p5} T={t1,t2,t3,t4}

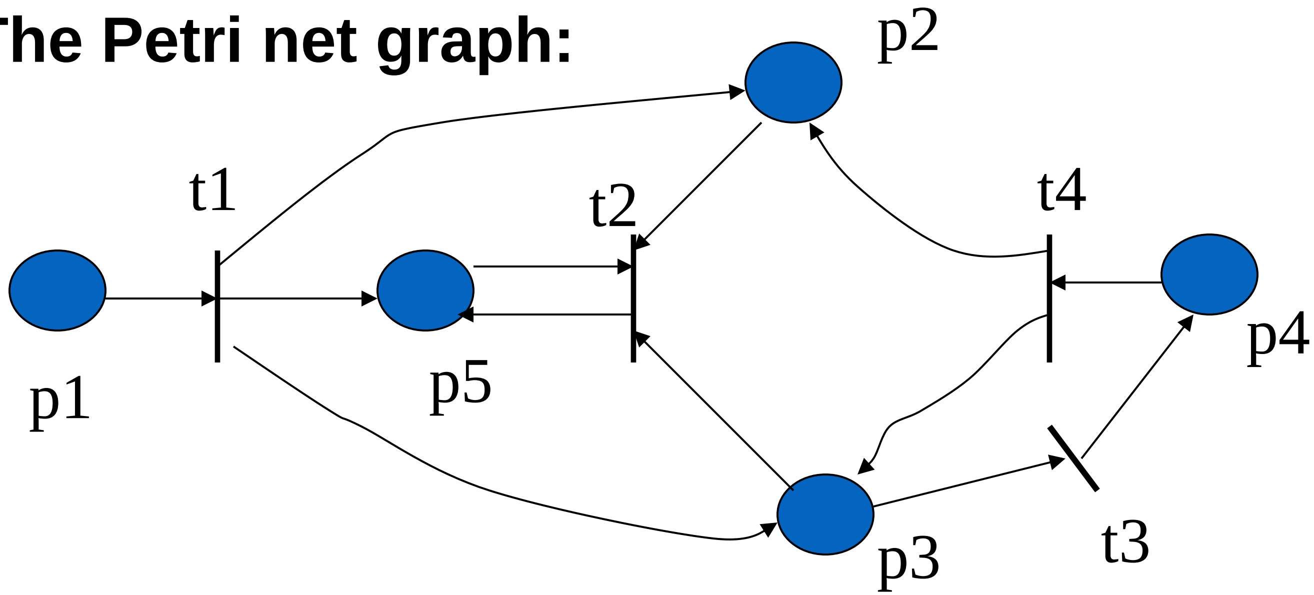
I(t1) = {p1}   O(t1) = {p2,p3,p5}

I(t2) = {p2,p3,p5}   O(t2) = {p5)

I(t3) = {p3}   O(t3) = {p4}

I(t4) = {p4}   O(t4) = {p2,p3}

The Petri net graph:



Marked PN

- A marked PN contains tokens
- Tokens are depicted graphically by dots (•) and reside in places
- A marking of a PN is a mapping that assigns a non-negative integer (the number of tokens) to each place of the Petri Net
- The marking characterizes the state of the Petri Net
- The initial marking is referred to as μ or m.

Definition for marking μ

Definition: *A marking* μ of a Petri net $C = (P,T,I,O)$ is a function from the set of places P to the nonnegative integers N.

$$\mu: P \to N$$

The marking μ can also be defined as an n-vector,

$\mu = (\mu_1, \mu_2, \ldots, \mu_n)$. The vector μ gives for each place p_i a μ_i number of tokens.

The definitions of a marking as a function and as a vector are obviously related by $\mu(p_i) = \mu_i$.

Example of Petri Net Markings

A Petri net structure:

C = (P,T,I,O) *P={p1,p2,p3,p4,p5}* *T={t1,t2,t3,t4}*

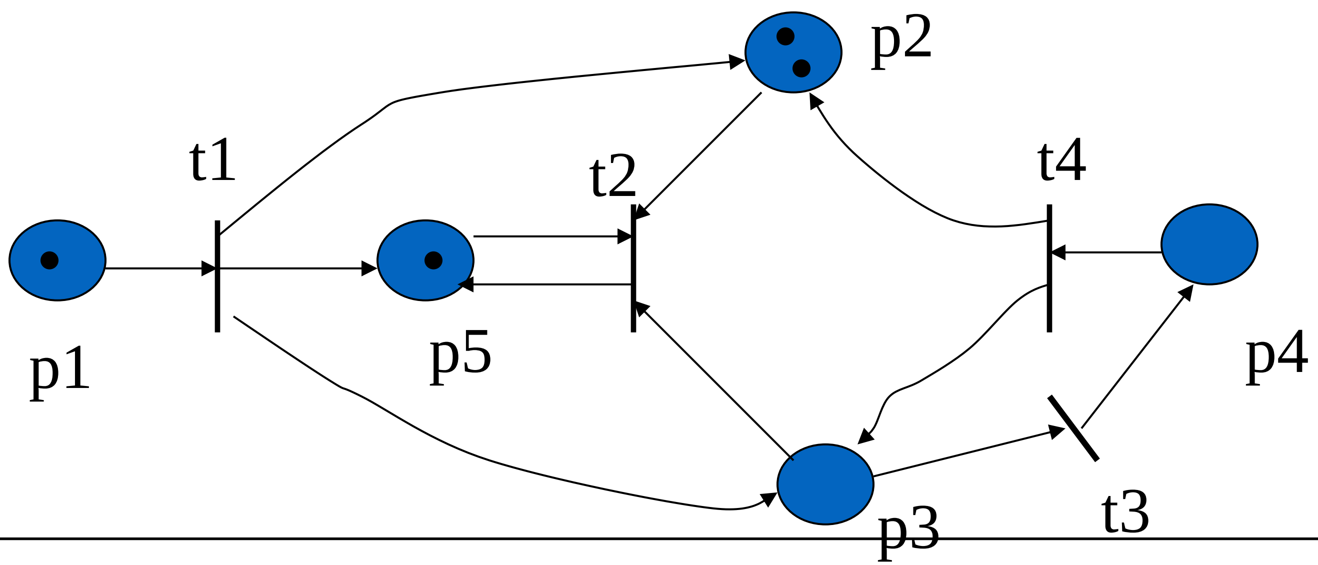
I(t1) = {p1} *O(t1) = {p2,p3,p5}*

I(t2) = {p2,p3,p5} *O(t2) = {p5)*

I(t3) = {p3} *O(t3) = {p4}*

I(t4) = {p4} *O(t4) = {p2,p3}*

The marking is *μ*=(1,2,0,0,1**)**

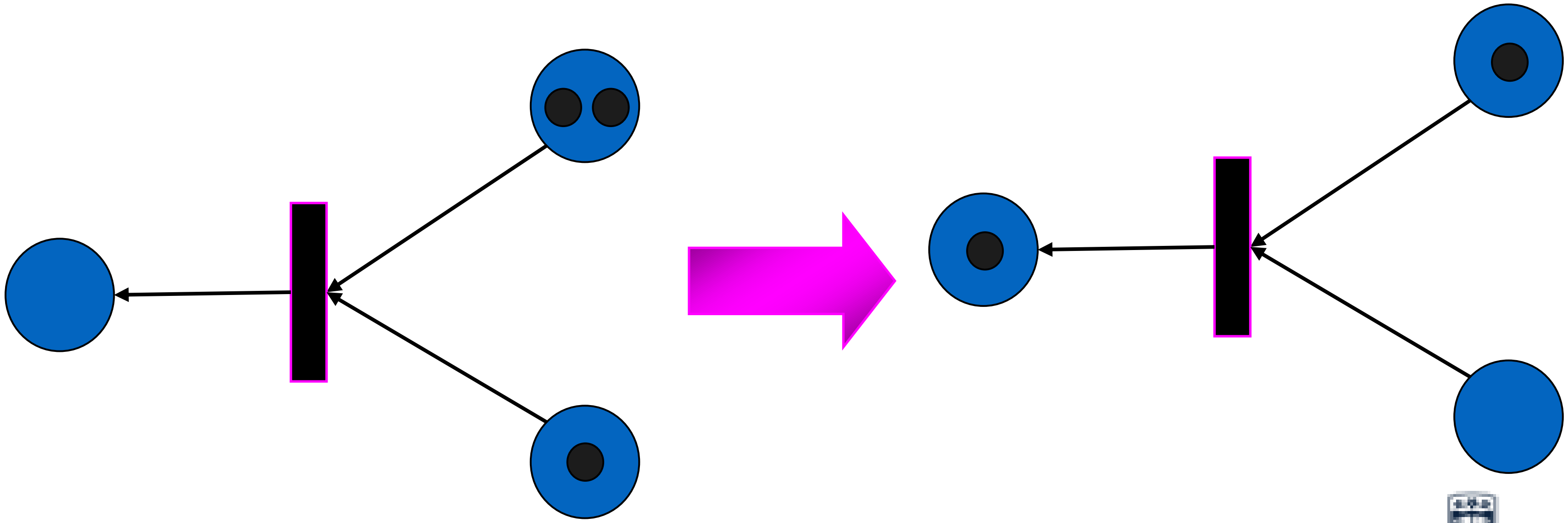


Execution Rules for Petri Nets

- A transition t is called *enabled* in a certain marking, if:
  - For every arc from an input place p to transition t, there exists a distinct token in the marking
- An enabled transition can be *fired* and result in a new marking
- Firing of a transition *t* in a marking is an atomic operation
- The state of a Petri net is defined by its marking, i.e., number of tokens in each places
- The firing of a transition represents a change in the state of the Petri net.

Execution Rules for Petri Nets (cont.)

- Firing a transition results in two things:
  1. Subtracting one token from the marking of any input place p for every arc connecting place p to transition t, i.e. decrement token for all I(t)
  2. Adding one token to the marking of any output place p for every arc connecting transition t to place p, i.e. increment token for all O(t)



Definitions

A transition $tj \in T$ in a marked Petri net $C = (P,T,I,O)$ with marking μ is enabled if for all $pi \in P$,

$$\mu(pi) \geq \#(pi, tj)$$

number of token within place pi — *number of arcs from input place pi to transition tj*

A transition *tj* in a marked Petri net with marking μ may fire whenever it is enabled.

Firing an enabled transition *tj* results in a new marking μ' defined by, for all $pi \in P$,

$$\mu'(pi) = \mu(pi) - \#(pi, tj) + \#(pi, tj)$$

new tokens in pi = original tokens in pi - tokens from pi + tokens to pi

Non-determinism

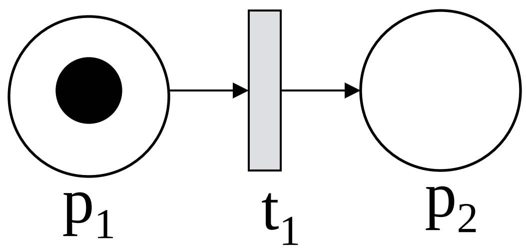
- Even a transition is enable, the execution of transition is non-deterministic.

  1. Multiple transitions can be enabled at the same time, any one of them can be fired

  2. None are required to fire - they fire at will (randomly), between time 0 and infinity, or not at all

Examples

- Below is an example Petri net with two places and one transaction.
- Transition node is ready to ***fire*** if and only if there is at least one token at place p1

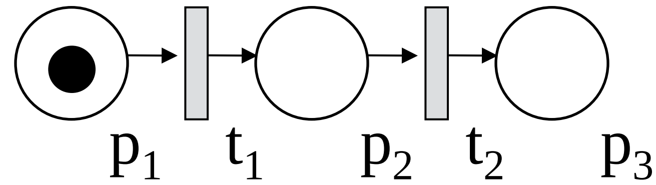


state transition of form **μ**=(1,0) → **μ1=(**0,1**)**

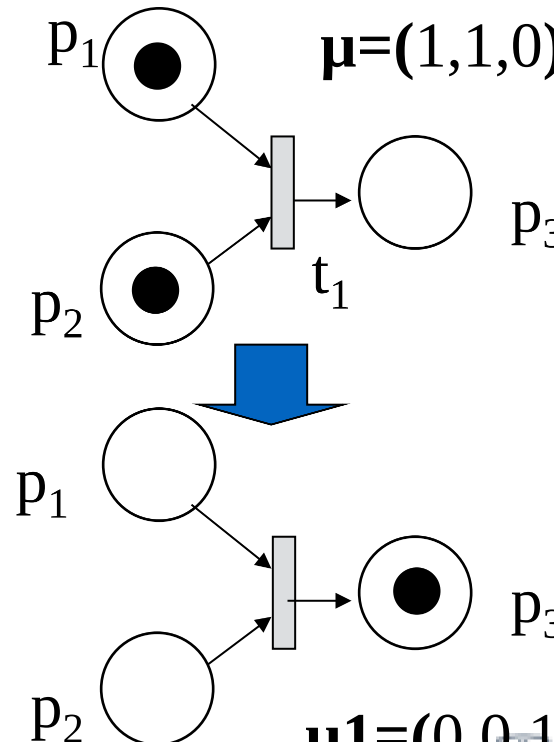
Examples (cont.)

- Sequential Execution
  Transition t_2 can fire only after the firing of t_1. This impose the precedence of constraints "t_2 after t_1."
- Synchronization
  Transition t_1 will be enabled only when there are at least one token at each of its input places.
- Merging
  Happens when tokens from several places arrive for service at the same transition.

$\mu=(1,0,0) \rightarrow \mu1=(0,1,0) \rightarrow \mu2=(0,0,1)$



$\mu=(1,1,0)$



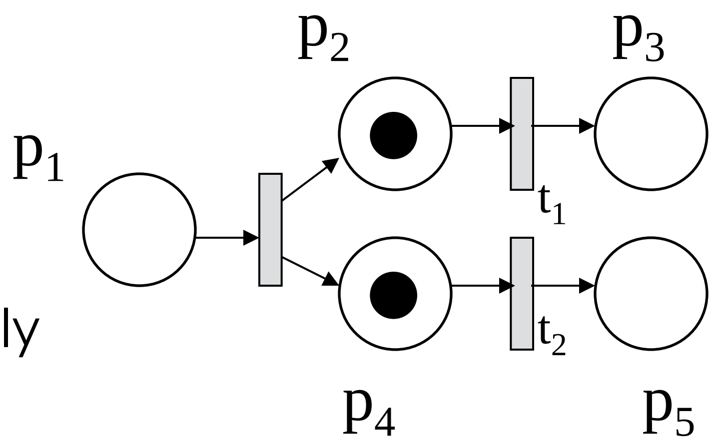
$\mu1=(0,0,1)$

Examples (cont.)

- Concurrency

  t_1 and t_2 are concurrent.

  - with this property, Petri net is able to model systems of distributed control with multiple processes executing concurrently in time.



$\boldsymbol{\mu}$**=(**0,1,0,1,0**)**$\rightarrow$ $\boldsymbol{\mu}$**1=(**0,0,1,0,1**)**

Example of Execution Rules for Petri Nets

Condition: for all $p_i \in P$, $\mu(p_i) \geq \#(p_i, t_j)$

μ=(1,0,0,2,1**),**

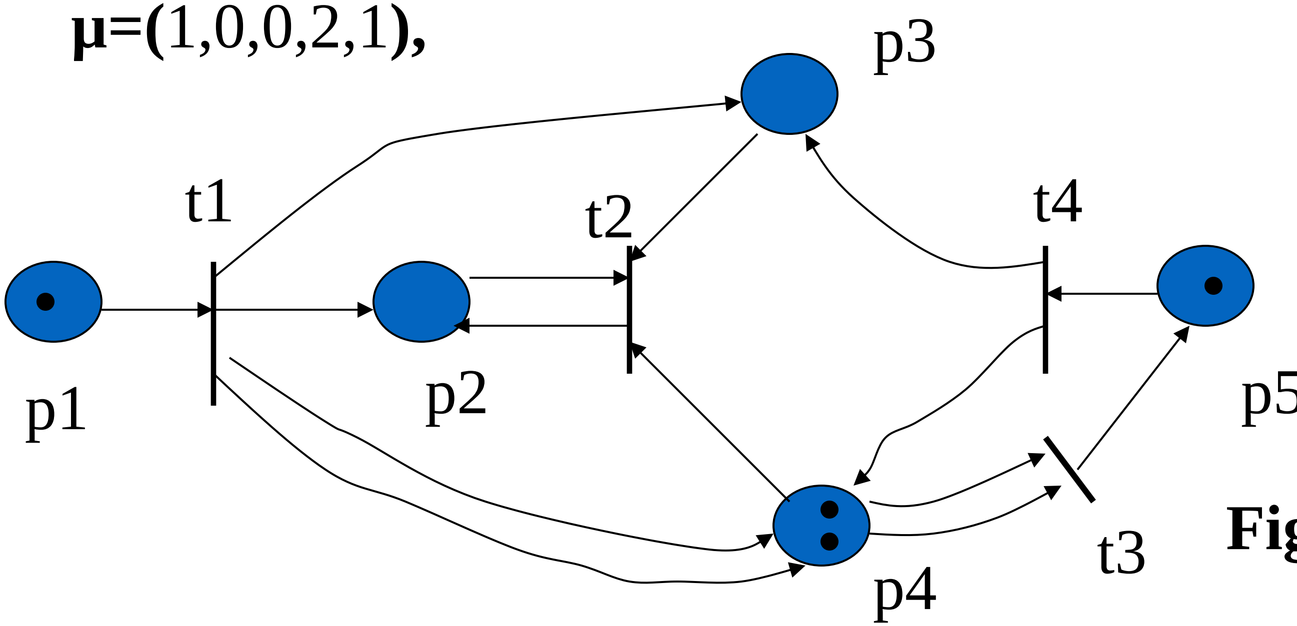


Figure1

Transitions t1, t3, and t4 are enabled

Because the execution of PN is non-determinism, we assume that the fire order is t4, t1, t3

Example of Execution Rules for Petri Nets

The marking result from firing transition t4 in Figure 1

μ'(pi) = μ(pi) - #(pi, tj) + #(pi, tj)

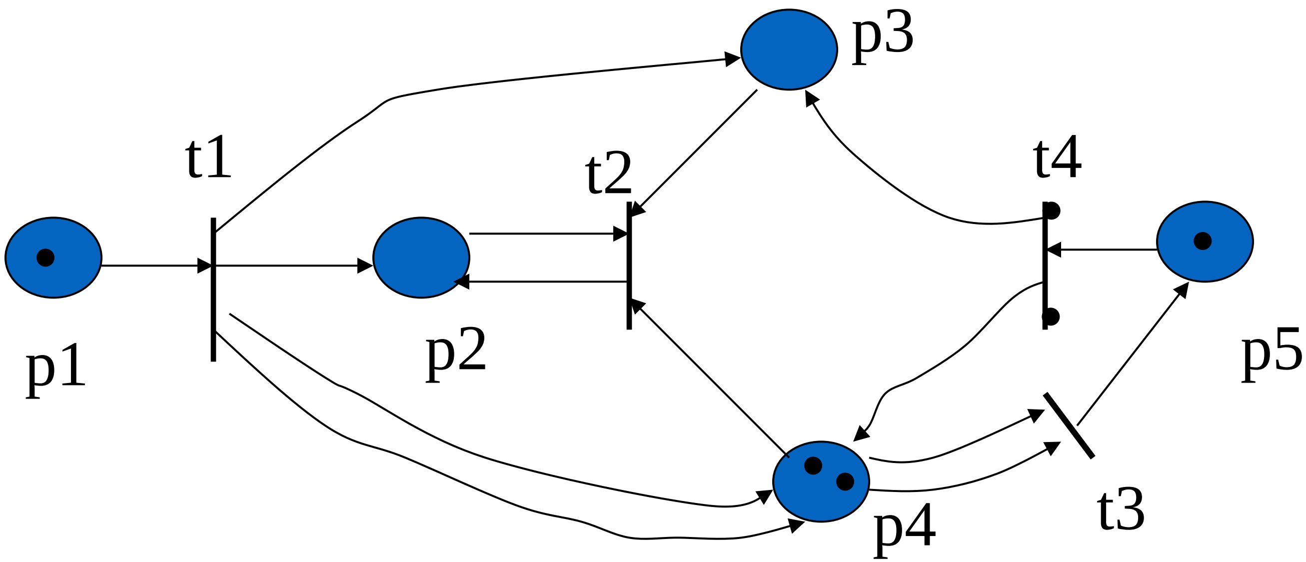


Figure2

μ=(1,0,0,2,1**)** → **μ1=(**1,0,1,3,0**)**

Example of Execution Rules for Petri Nets (continuing)

The marking result from firing transition t1 in Figure 2

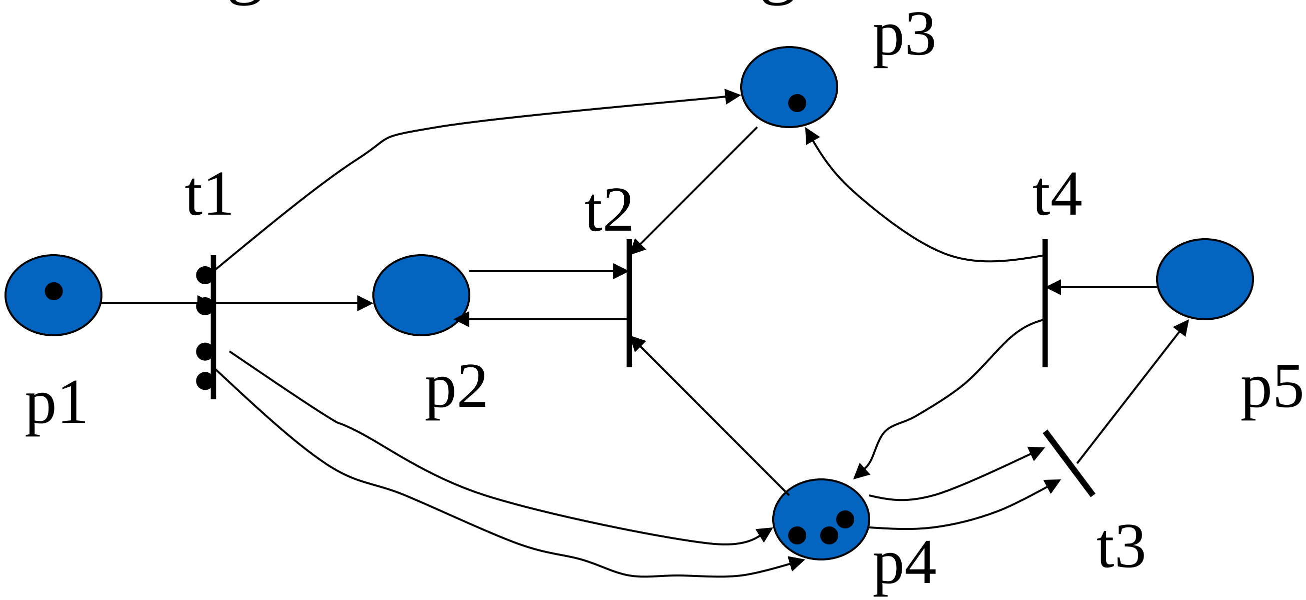


Figure3

μ1=(1,0,1,3,0**)** → **μ2=(**0,1,2,5,0**)**

Example of Execution Rules for Petri Nets (continuing)

The marking result from firing transition t3 in Figure 3

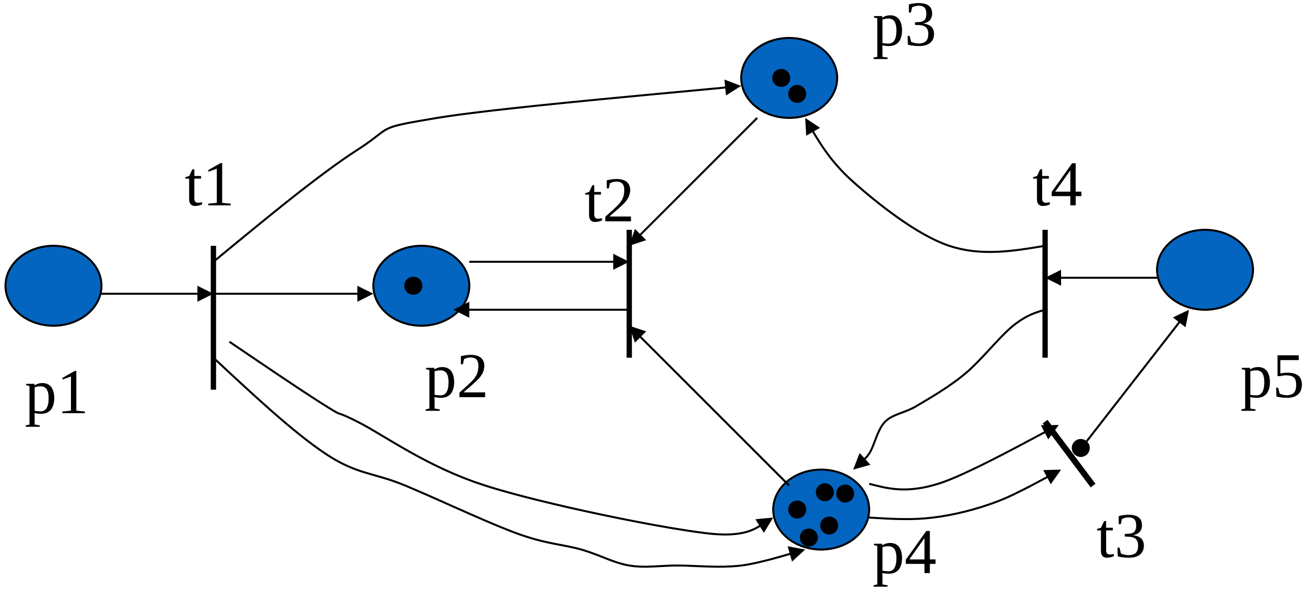


Figure4

μ2=(0,1,2,5,0) **→ μ3=(**0,1,2,3,1**)**

Petri Net Applications

- **performance evaluation**
- **communication protocols**
- distributed-software systems
- distributed-database systems
- concurrent and parallel programs
- industrial control systems
- discrete-events systems
- multiprocessor memory systems
- dataflow-computing systems
- fault-tolerant systems
- etc, etc, etc

Conclusion

- Petri Nets
  - executable
  - concurrent, asynchronous, distributed, parallel, nondeterministic and/or stochastic systems
  - graphical tool
    - visual communication aid
  - mathematical tool
    - state equations, algebraic equations, etc
  - communication between theoreticians and practitioners