ECE 321 Software Requirements Engineering

Lecture 12: Petri Nets

Petri Nets

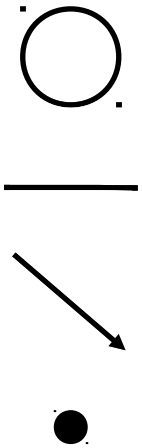
- Graphical formalism for system specification
- Describe operational model
- Allows the specification of asynchronous systems
  - Two or more actions can happen simultaneously
- Many software tools available for modeling systems using Petri Nets

Definition of a Petri Net

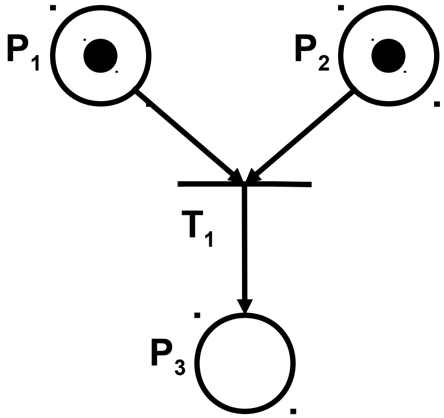
- PN = {P, T, A, M_0}
  - P is a finite set of places
  - T is a finite set of transitions
  - A is a finite set of directed arcs (arrows) connecting places to transitions and vice versa
  - M_0 is the *initial marking* of PN

The elements of Petri Nets

- Places
- Transitions
- Arrows/Arcs
- Token



- M_0 is the initial marking (state) of PN
  - $M_0 = \{1, 1, 0\}$



Petri Nets: Places

- Places hold tokens
  - Presence of a token represents the existence of some condition
- A marking is a particular arrangement of tokens
- The initial marking is a initial state of a system
- State of a system modeled by PN is represented by a marking

Petri Nets: Transitions

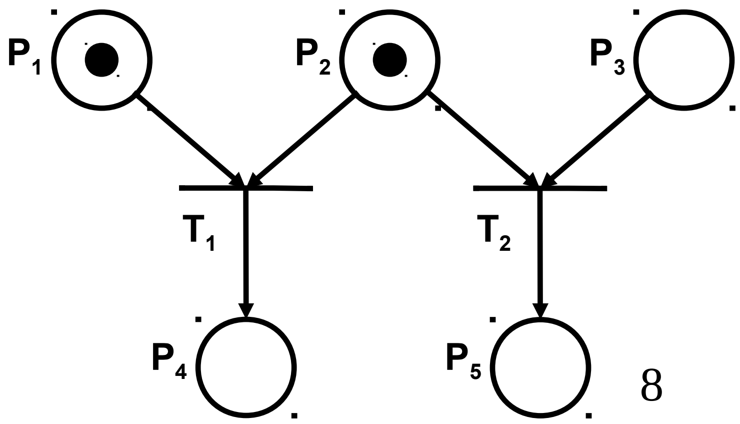
- Represent activity (some computational progress)
- May have multiple inputs and outputs

Petri Nets: Arcs

- Connect places to transitions and transitions to place
- Why not places to places or transitions to transitions?
  - By definition: the transitions model the activity that is necessary to go from one place to another

Enabled transitions

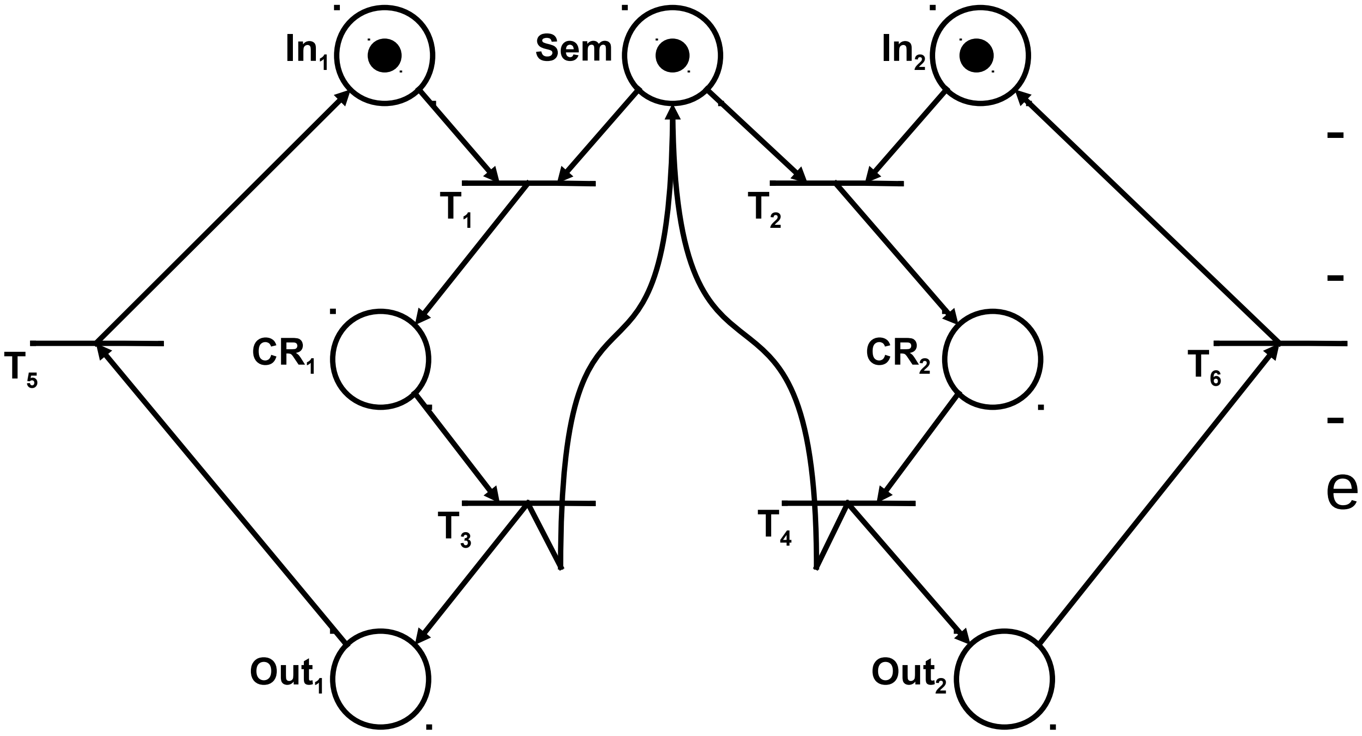
- Input place P of a transition T has an arc from P to T
  - P_1 and P_2 are input places for T_1
  - P_2 and P_3 for T_2
- Output place P of T has an arc from T to P
  - P_4 output for T_1, P_5 for T_2
- T is **enabled** if there is at least one token in each of its input places
  - Enabled: only T_1



Petri Net example: Semaphore 1/2

- Two processes can access a critical resource through a semaphore
  - e.g., a printer
- Input places: $\mathbf{In_1}$, $\mathbf{In_2}$
- Output places: $\mathbf{Out_1}$, $\mathbf{Out_2}$
- Semaphore: **Sem**
- Critical Resource: **CR**

Petri Net example: Semaphore 2/2

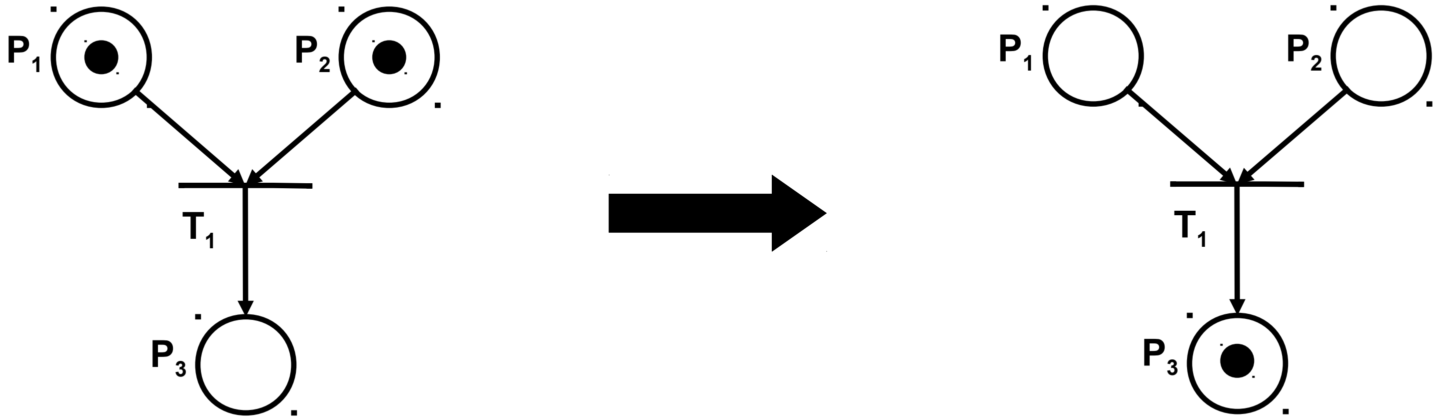


- How many places?
    - 7
- How many transitions?
    - 6
- Which transitions are enabled?
    - T_1, T_2

Firing transitions

- An enabled transition **may** fire
  - Nondeterministically selected
    - Any enabled transition can be selected
    - Different evolutions of PN are possible
  - Results in removing one token from each of its input places, and inserting one into each of its output places
  - Tokens are consumed and generated; they do not ‘flow’ through PN
  - May also **not** fire!

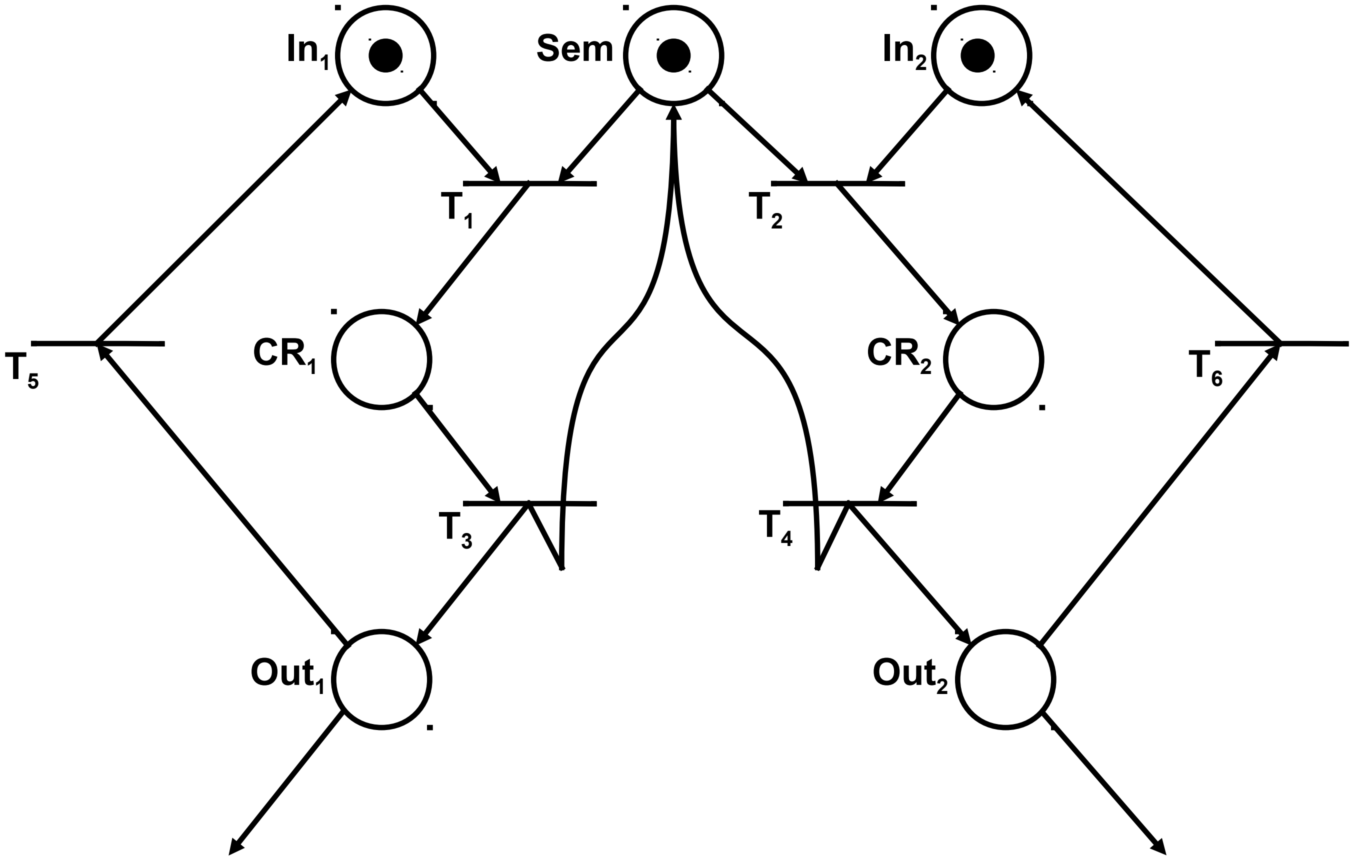
Before and after firing



Firing sequences

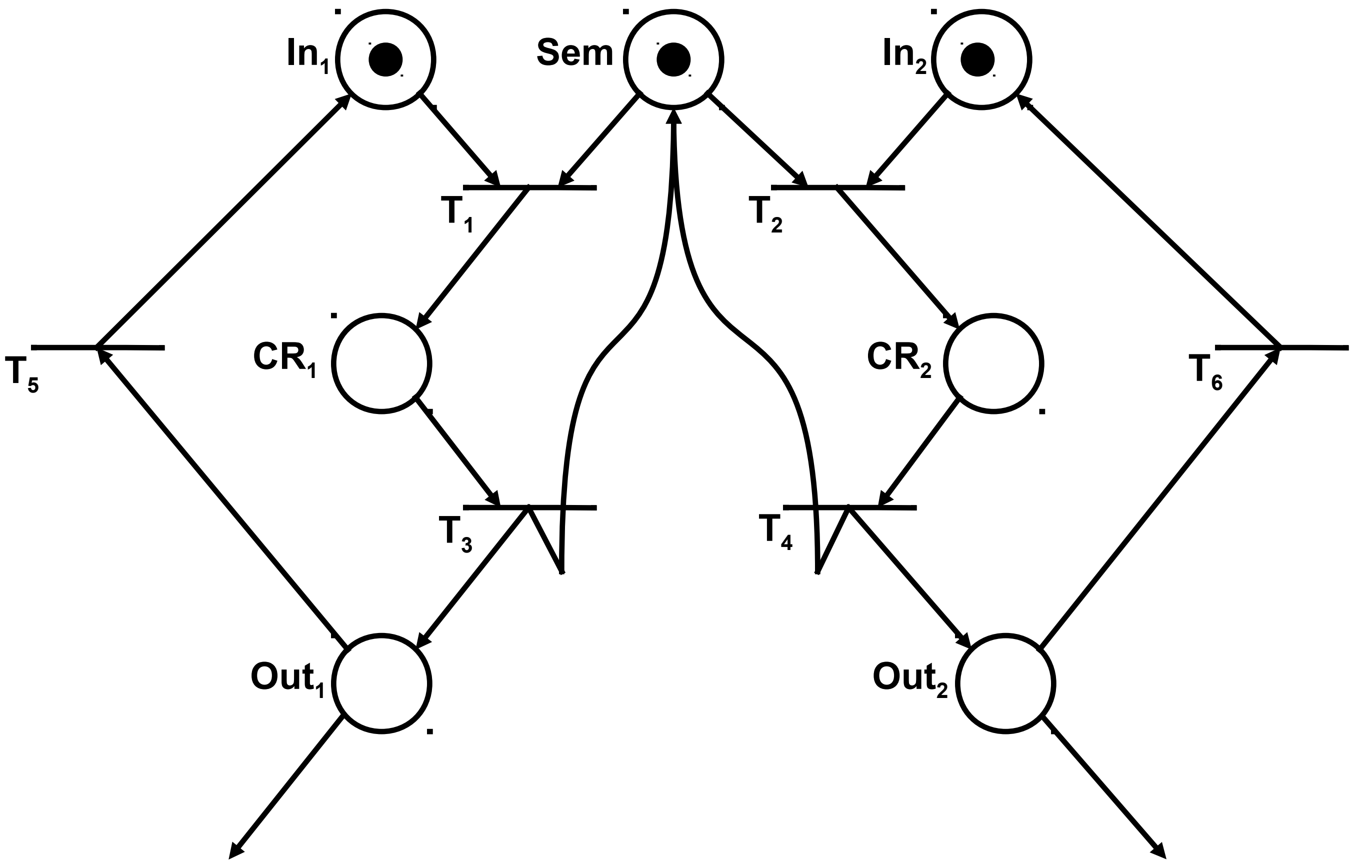
- A sequence <T_1, T_2, T_3, ..., T_n> such that
  - T_1 is enabled and fired in M_0
  - T_2 is enabled and fired in M_1
  - Etc.
- Describes the behaviour of the modeled system
- While firing transitions, we possible enable other transitions to fire

Demonstrating the semaphore's functionality



What are the possible firing sequences?

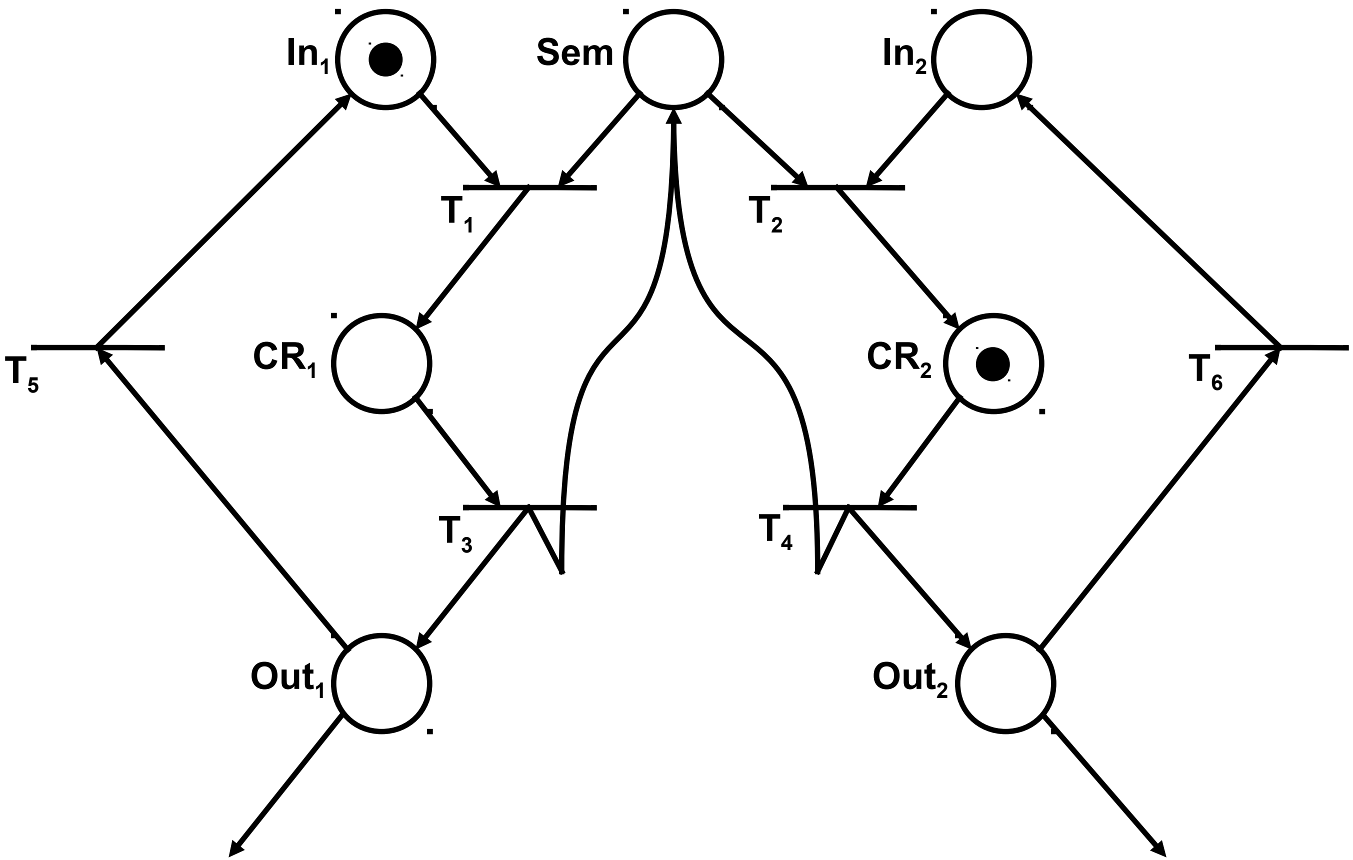
Demonstrating the semaphore's functionality



Initial PN:

Token in In_1, Sem, In_2

Demonstrating the semaphore's functionality



Initial PN:

Token in In_1, Sem, In_2

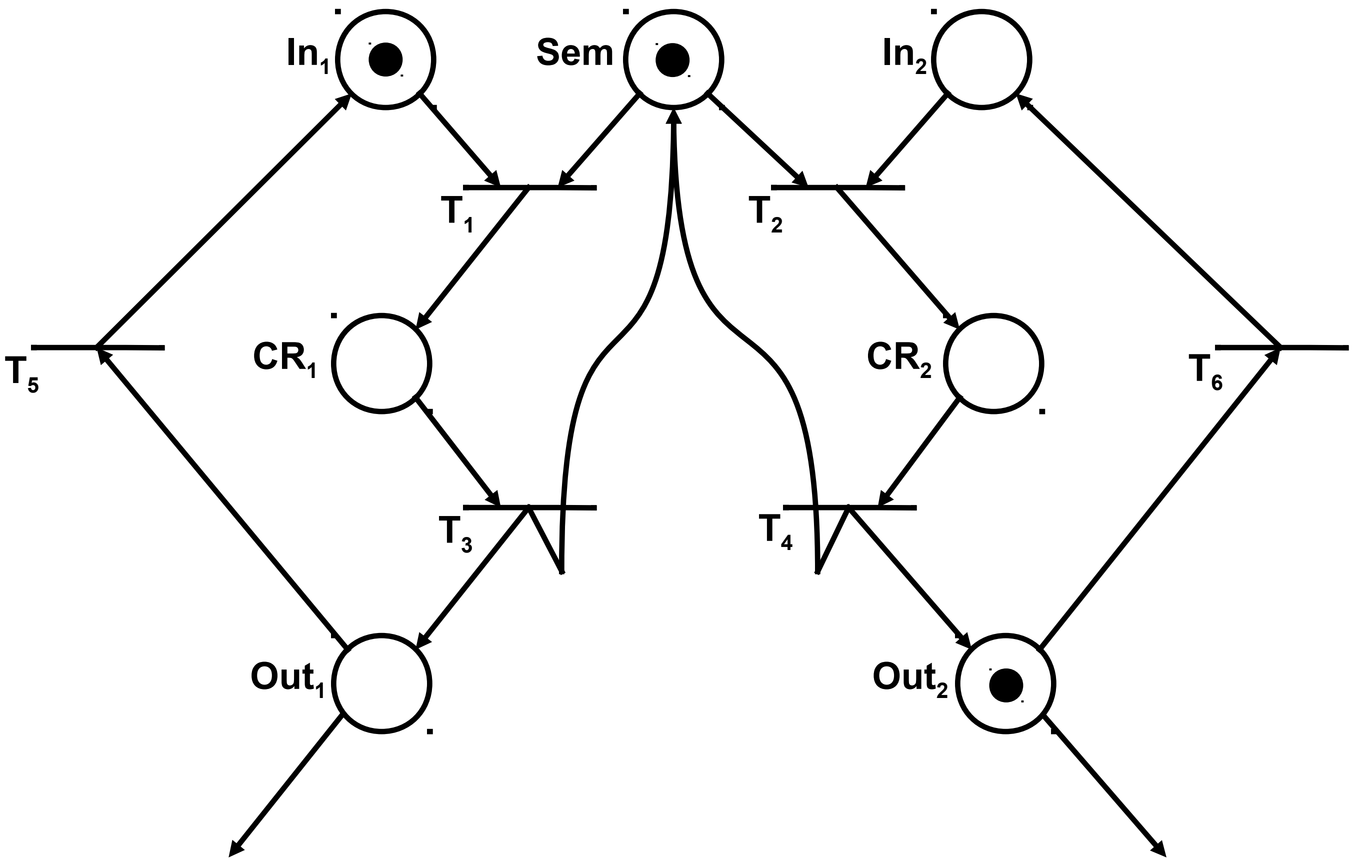
T_2 fires:

Token in CR_2

Note: T_1 is not enabled!

So cannot access CR_1

Demonstrating the semaphore's functionality



Initial PN:

Token in In_1, Sem, In_2

T_2 fires:

Token in CR_2

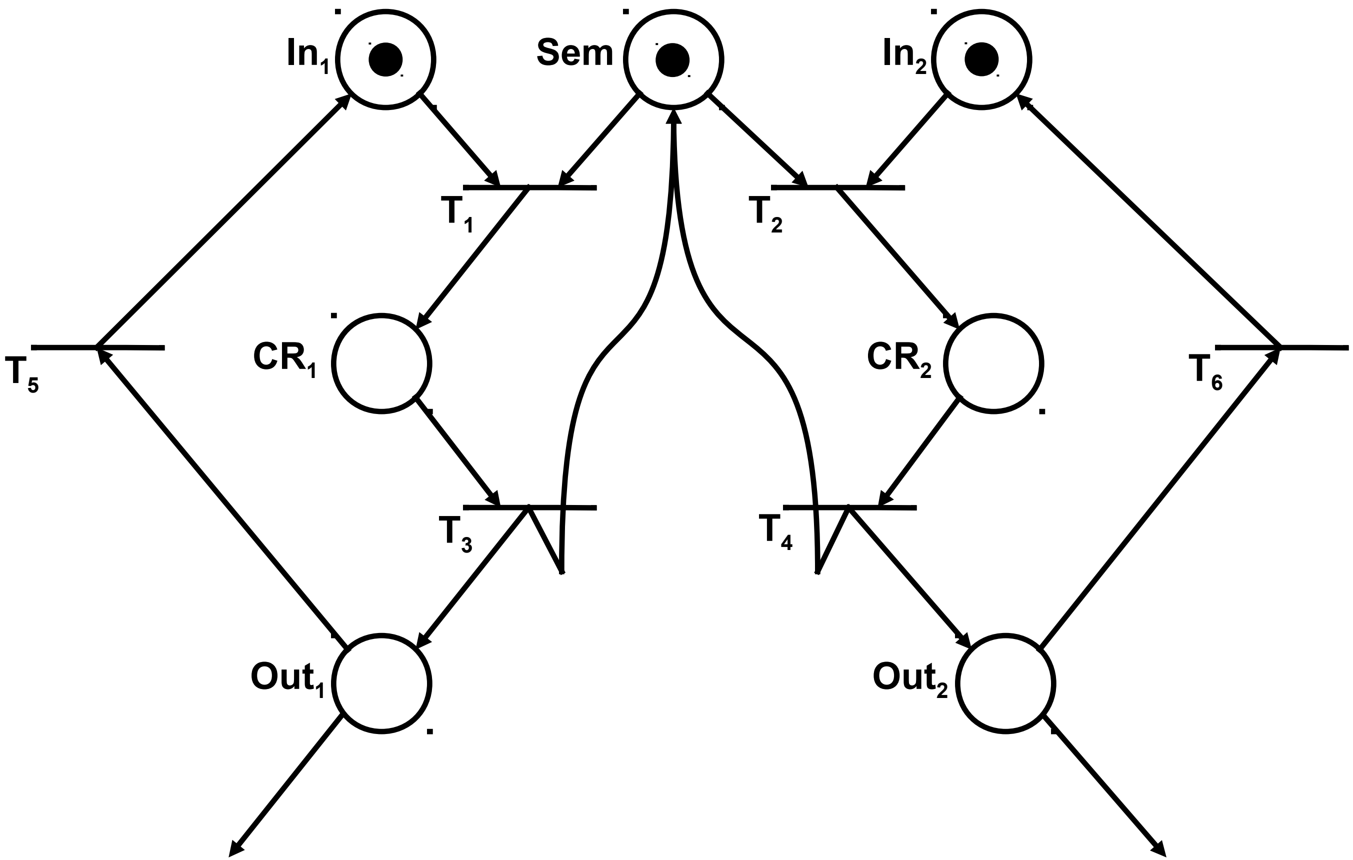
Note: T_1 is not enabled!

So cannot access CR_1

T_4 fires:

Token in In_1, Out_2, Sem

Demonstrating the semaphore's functionality



Initial PN:

Token in In_1, Sem, In_2

T_2 fires:

Token in In_1, CR_2

Note: T_1 is not enabled!

So cannot access CR_1

T_4 fires:

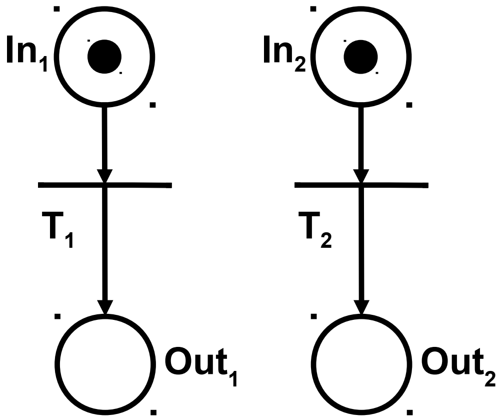
Token in In_1, Out_2, Sem

T_6 fires:

Token In_1, Sem, In_2

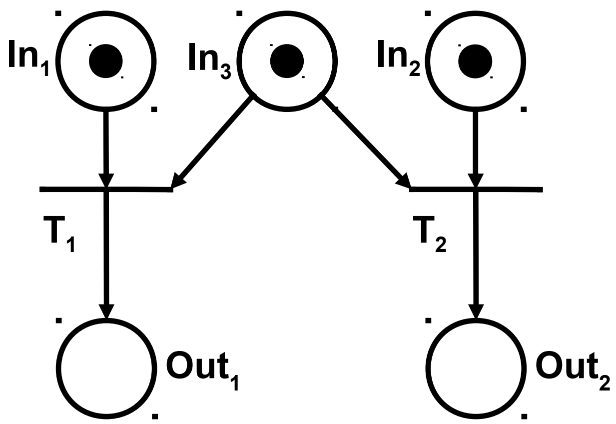
Concurrency in Petri Nets 1/3

- T_1 and T_2 are concurrent
  - They are independent
- We choose non-deterministically which one fires first
  - Order does not matter: both can be fired
  - Both <T_1, T_2, ... > and <T_2, T_1, ... > can be executed

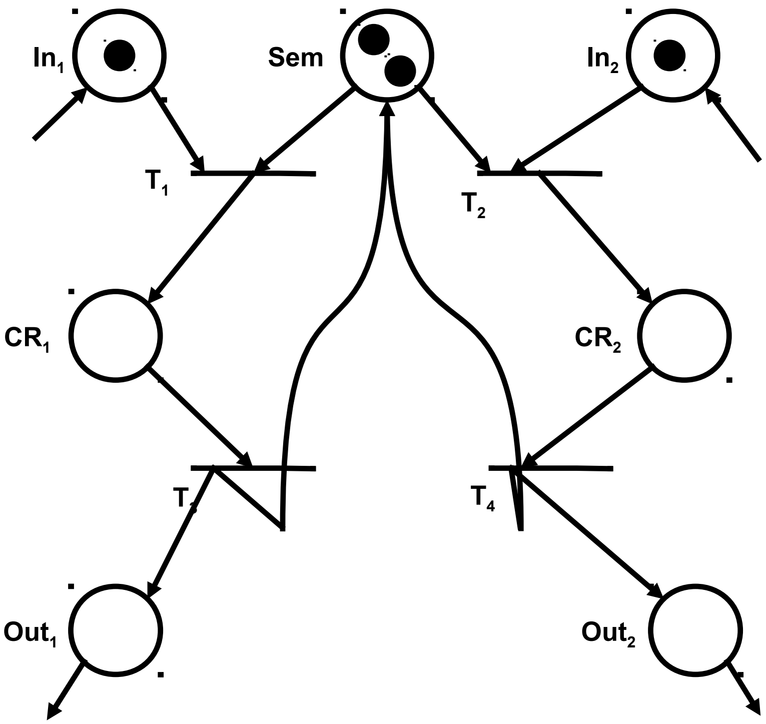


Concurrency in Petri Nets 2/3

- T_1 and T_2 are not concurrent
  - Firing one disables the other one
- Non-deterministically we choose which one to fire first
  - $< T_1, \ldots >$ **or** $< T_2, \ldots >$ can execute

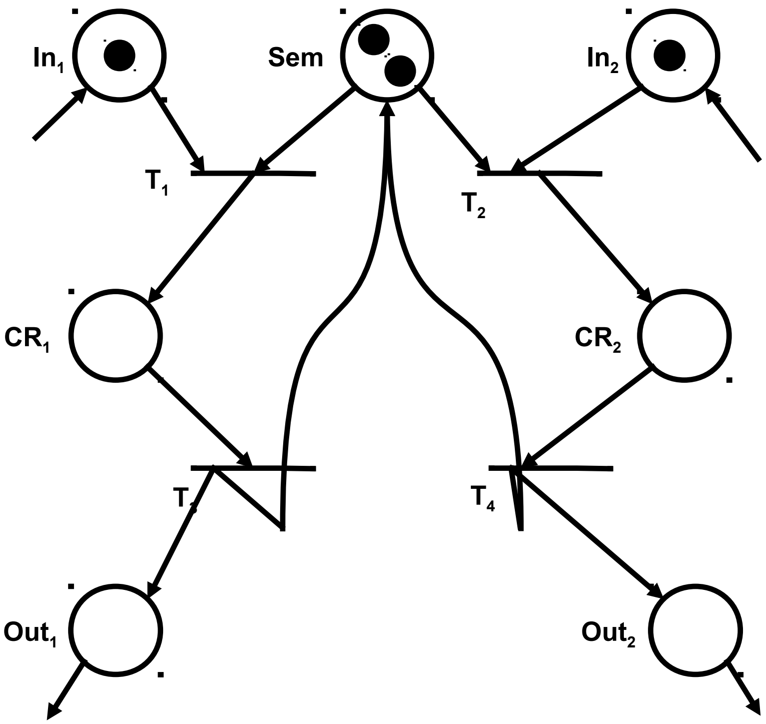


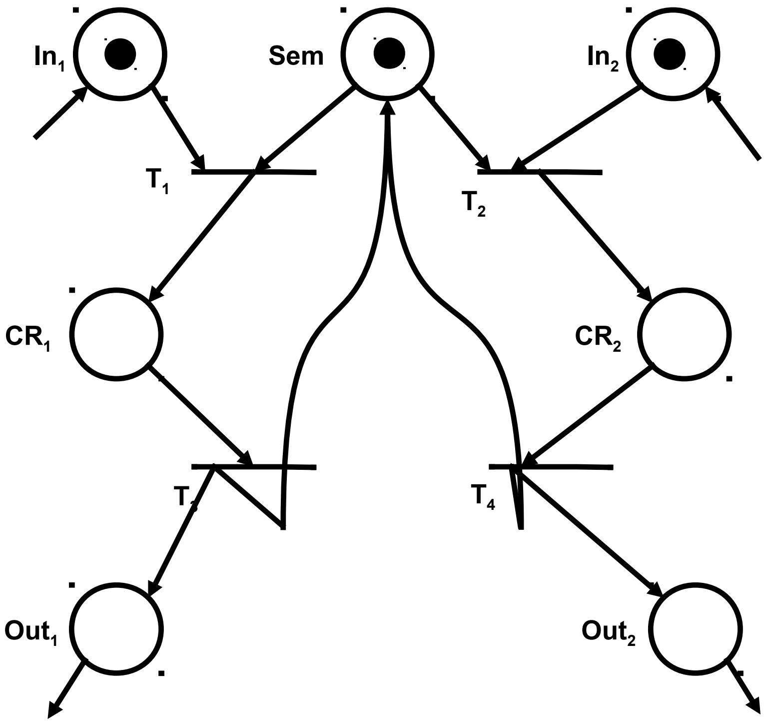
Concurrency in Petri Nets 3/3



Are T_1 and T_2 concurrent?

Concurrency in Petri Nets 3/3



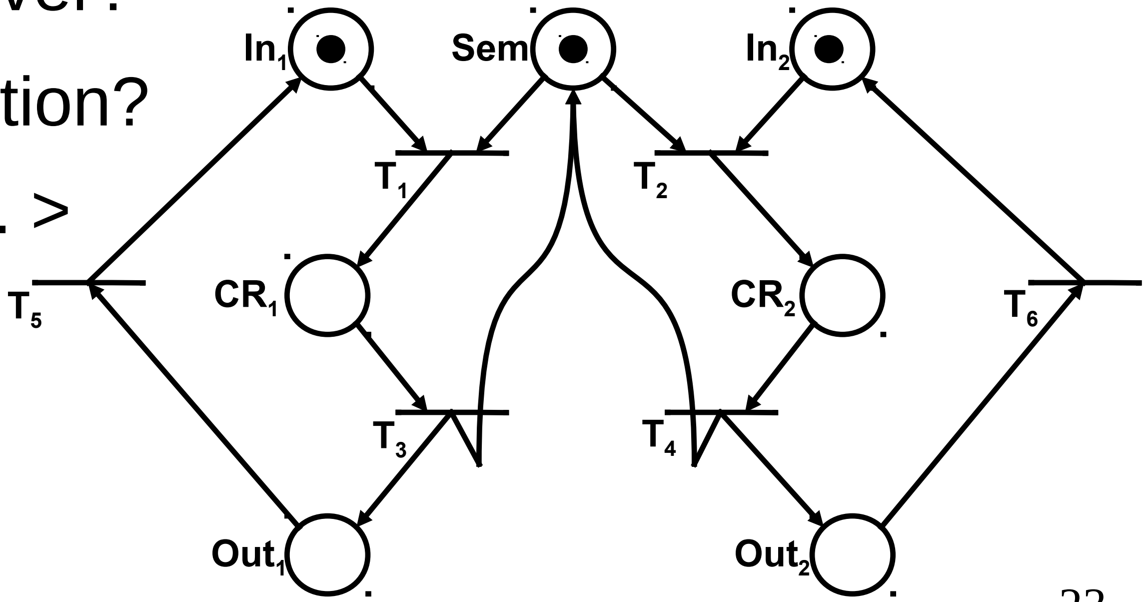


Are T_1 and T_2 concurrent? **YES**

And now? **NO**

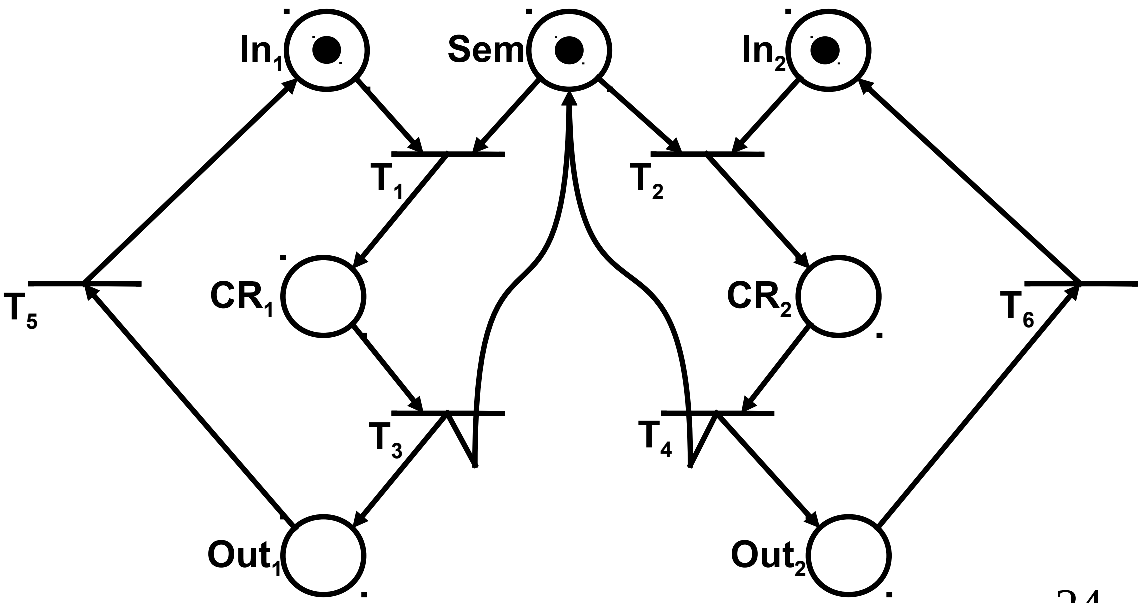
Concurrency: Starvation

- Occurs when enabled transition will never fire
- Will this model run forever?
- Can you identify starvation?
  - < T_1, T_3, T_5, T_1, T_3, ... >
  - Process 2 starves



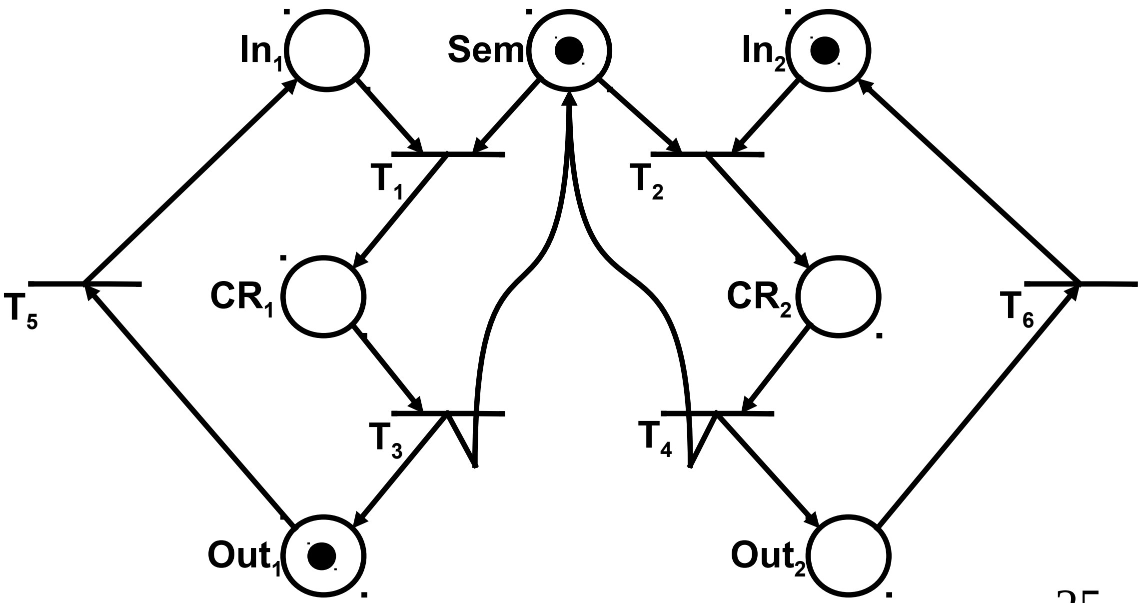
Concurrency: Starvation

- We can fix starvation by assigning priorities to transitions
- To which transitions?
  - T_1 and T_2



Concurrency: Starvation

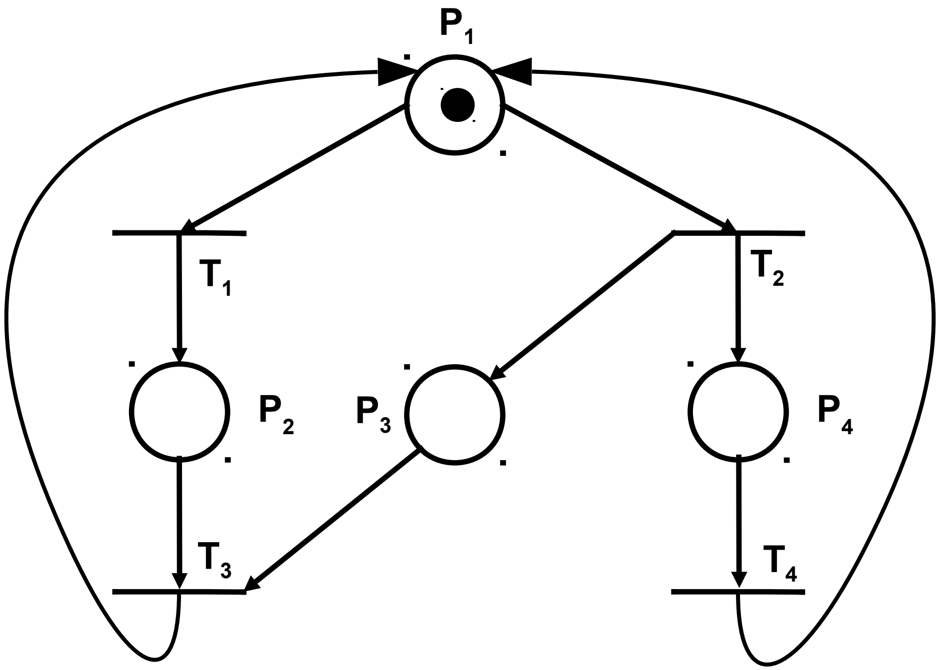
- We can fix starvation by assigning priorities to transitions
- To which transitions?
  - T_1 and T_2



Concurrency: Deadlock

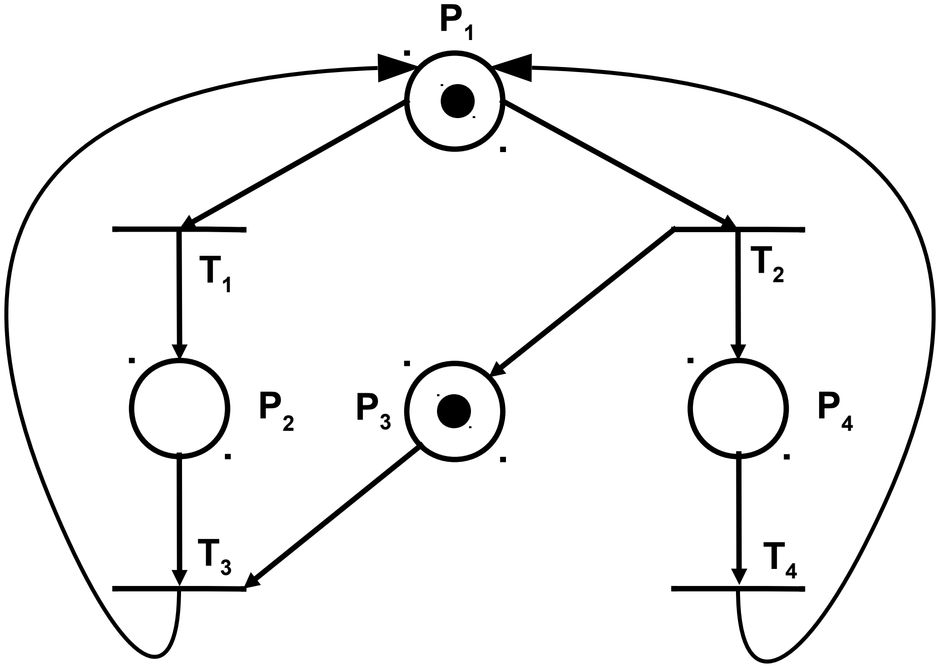
- None of the transitions are enabled
- Model stops in an ‘unreasonable’ place

Will the execution of this model result in deadlock?



Yes:
$< T_1 >$

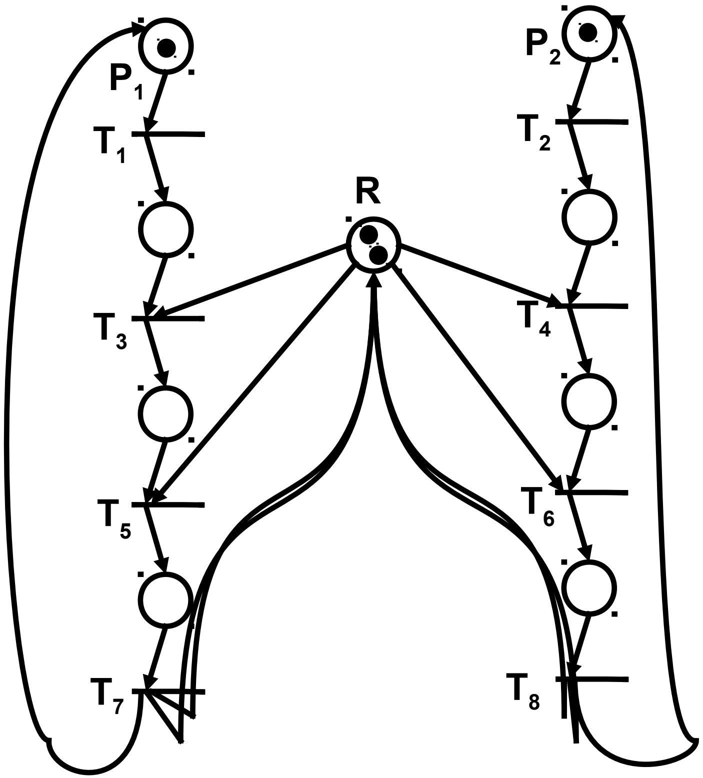
Will adding a token help to avoid deadlock?



No:
< T_1, T_3, T_1 >

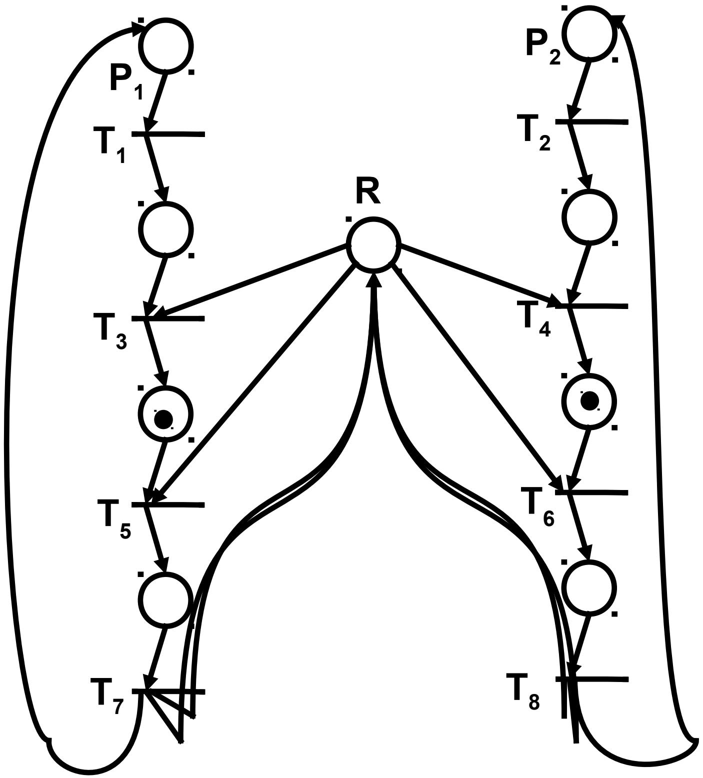
A modified semaphore system

- Two processes P_1 and P_2
- Two resources (tokens in R)
- Each process can get both resources
- Is this PN deadlock free?
- Is this PN starvation free?



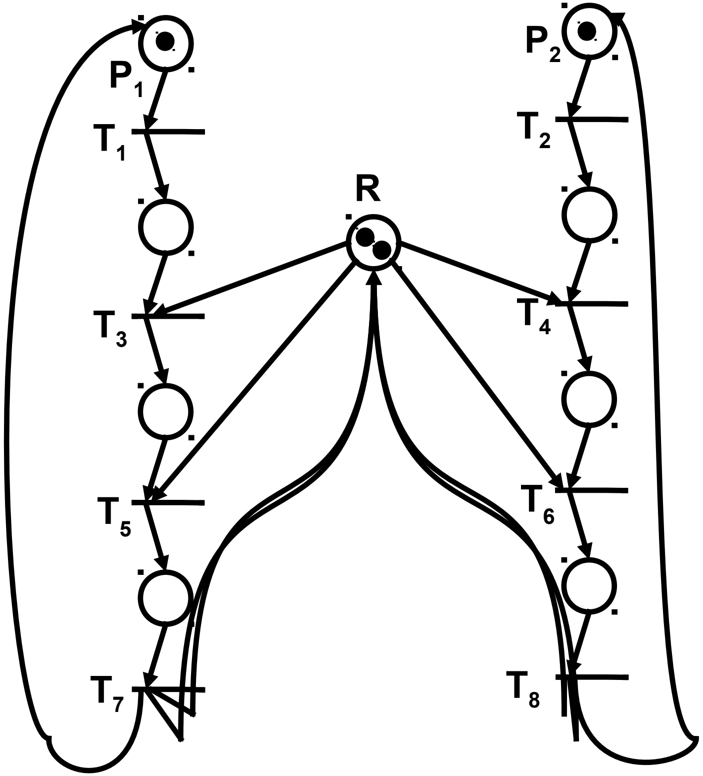
A modified semaphore system: Deadlock

- < T1, T3, T2, T4 >
- No transitions are enabled
- Both processes hold one resource and wait for the other process to release a resource



A modified semaphore system: Live systems

- A system where no deadlock can occur is said to be **live**
- How can we avoid deadlock in this system?
  - Assign priorities to transitions
- Assign higher priority to T_5 and T_6

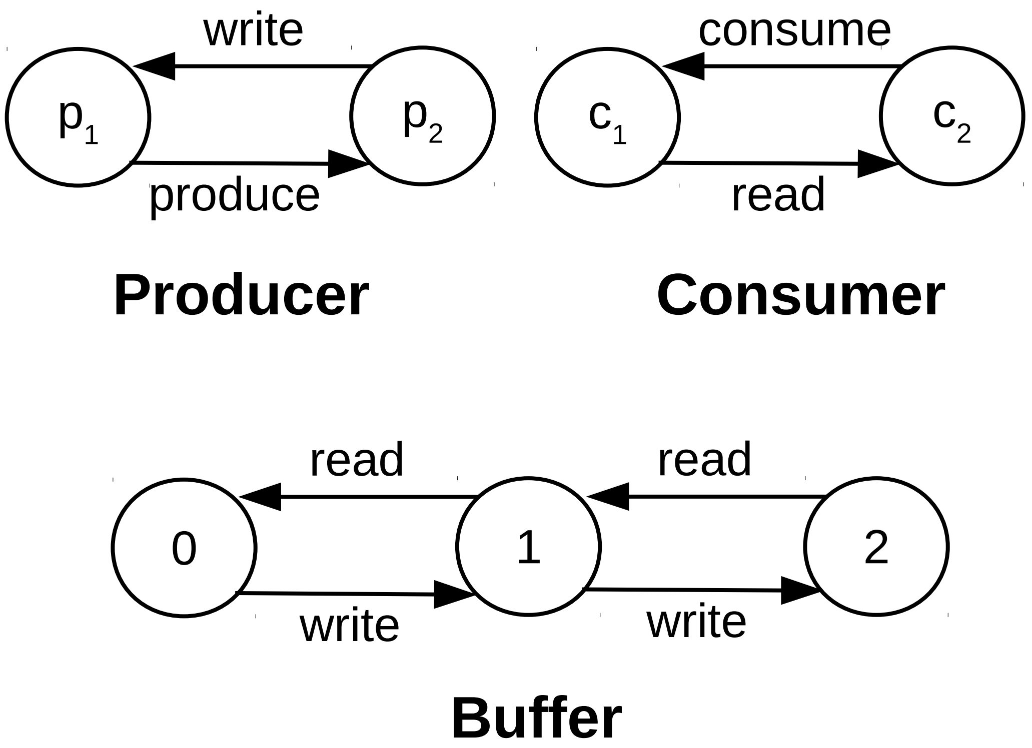


Detecting starvation and deadlock

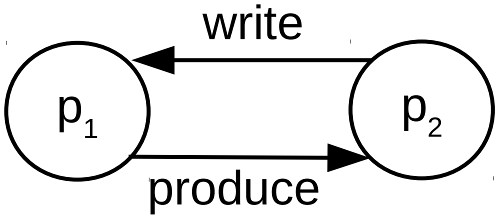
- Can be done using static and dynamic analysis tools
  - e.g., PIPE (Platform Independent PetriNet Editor)
  - Will be used in the lab and in the last assignment
  - Also very useful to validate your own Petri Nets while studying them!

Remember our consumer-producer system?

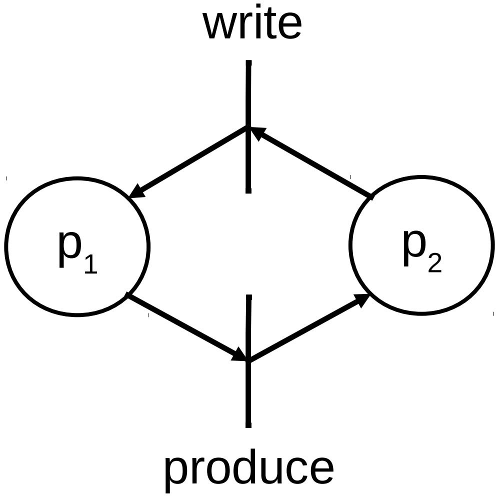
- Problems:
  - State space explosion
  - Single thread



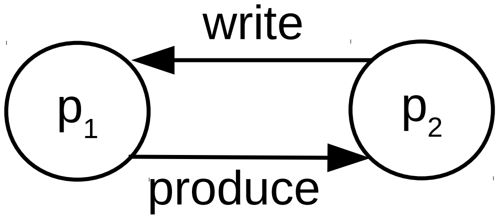
The producer component in a Petri Net



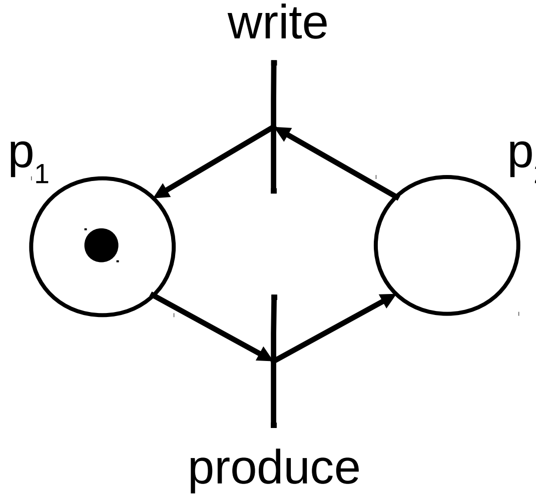
Producer



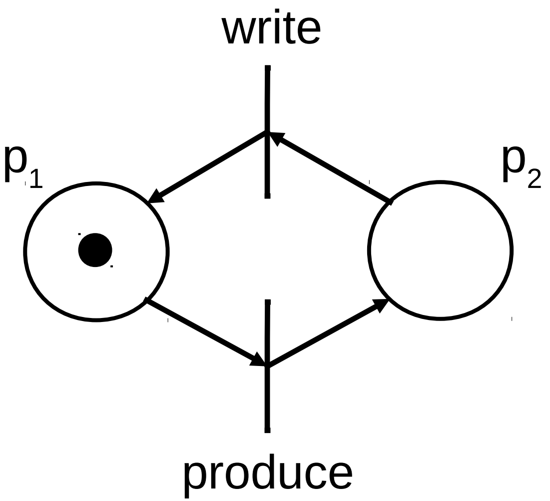
The producer component in a Petri Net

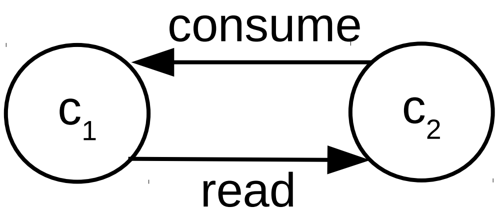


Producer

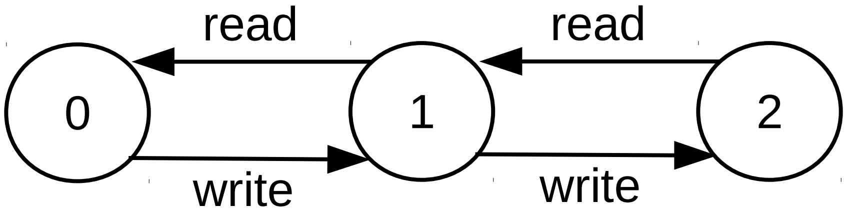


Can you draw the other Petri Nets?



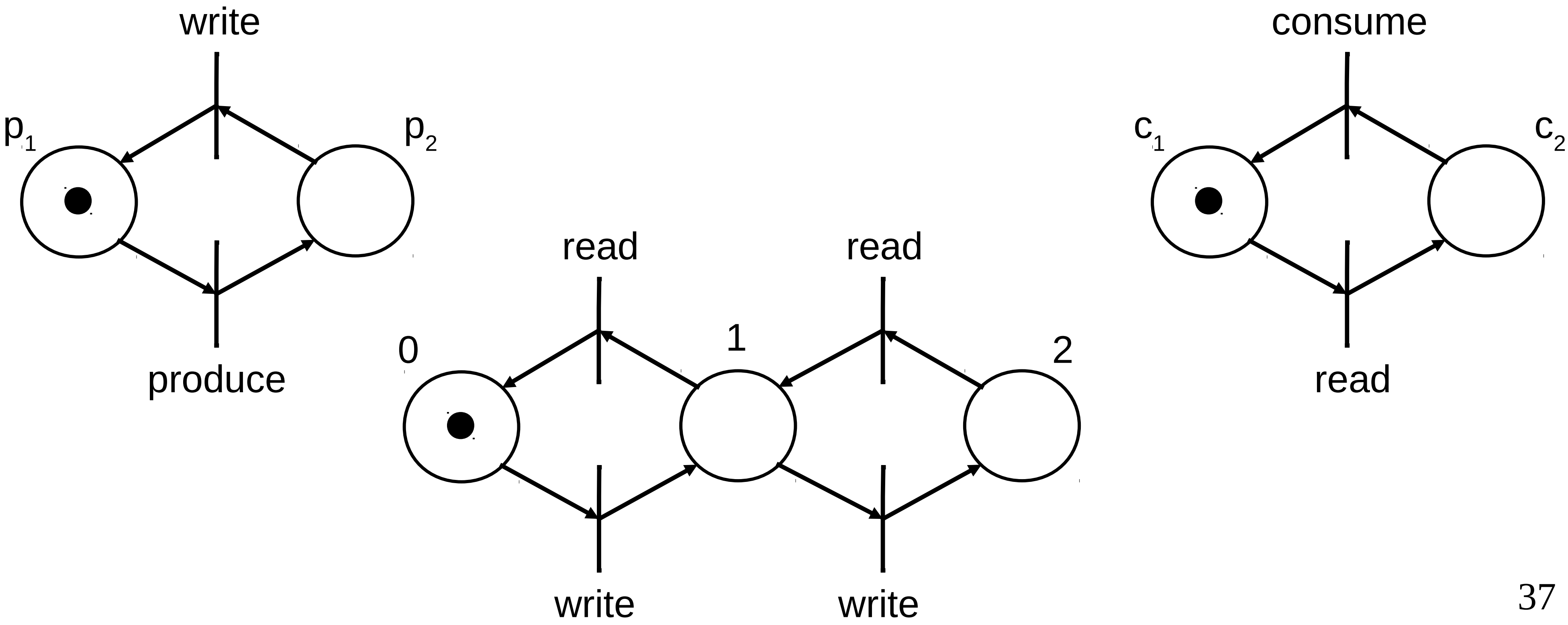


Consumer

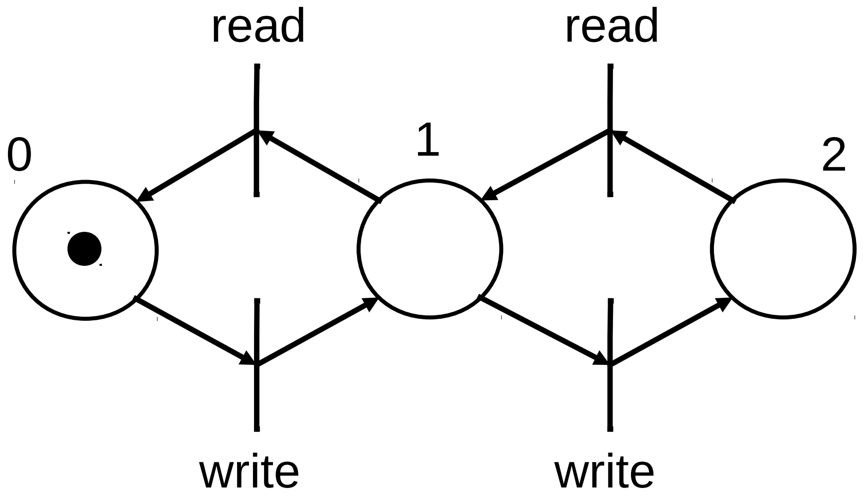


Buffer

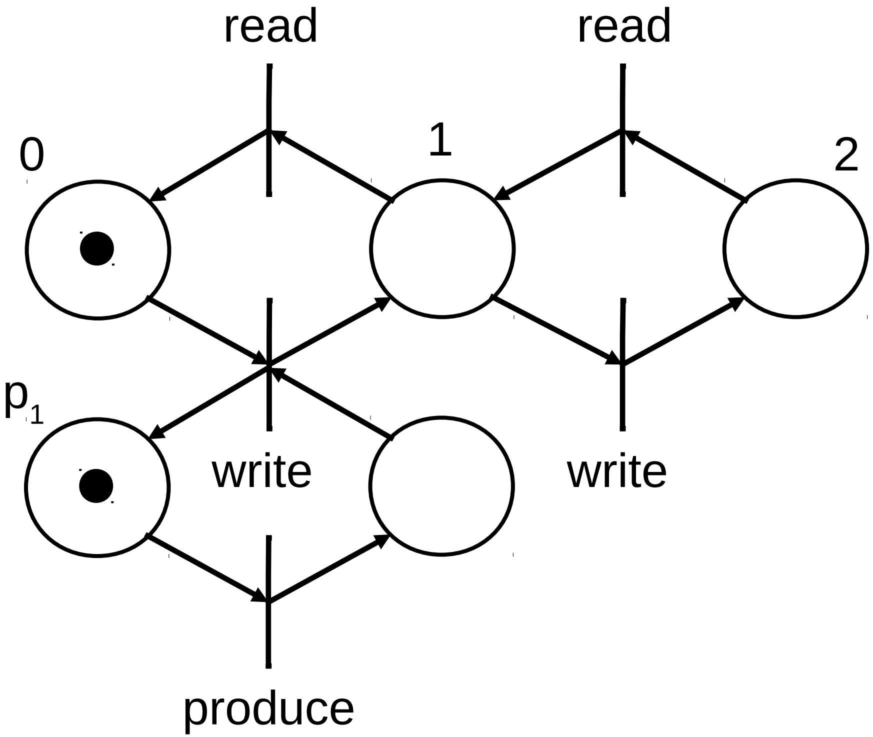
The consumer-producer components in a Petri Net



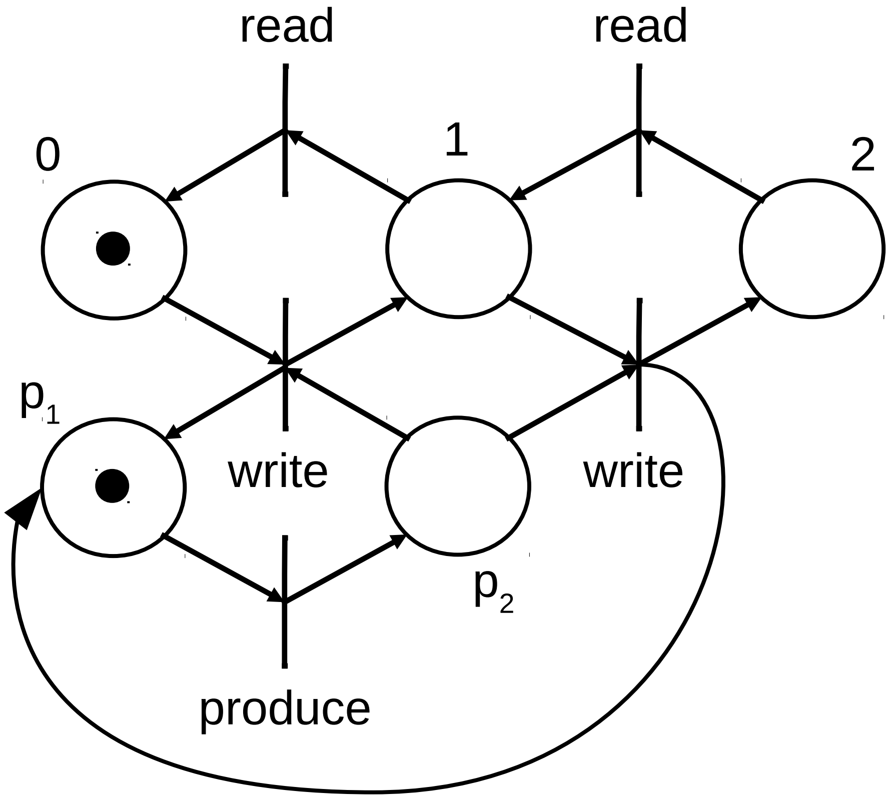
How can we combine these Petri Nets?



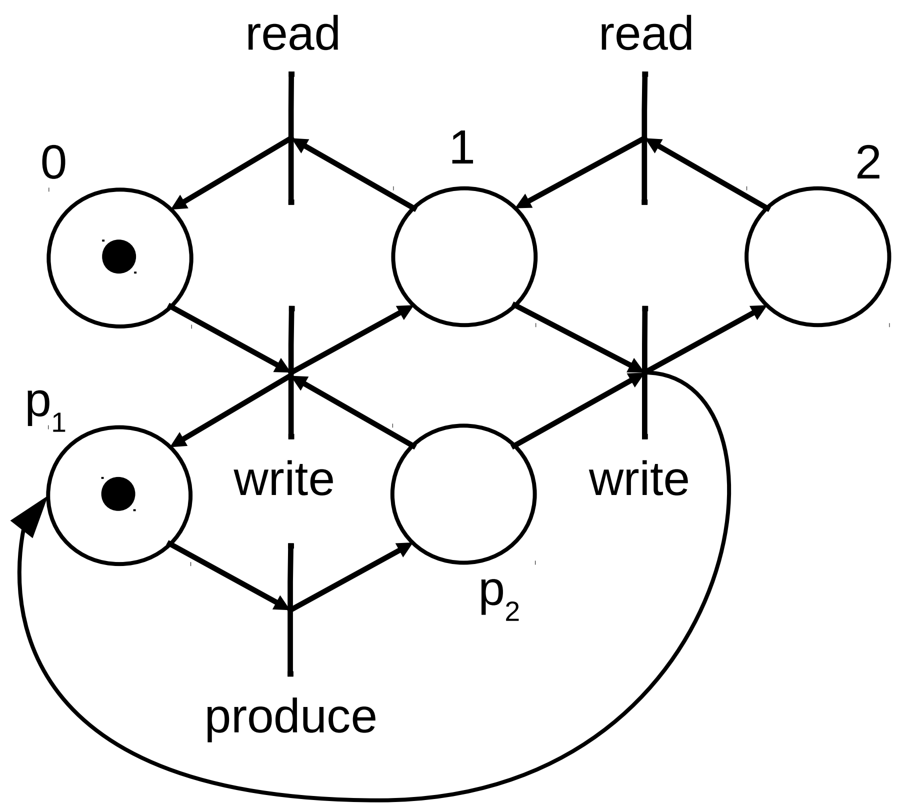
How can we combine these Petri Nets?

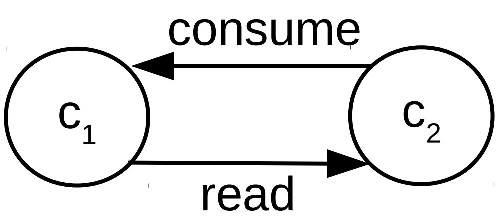


How can we combine these Petri Nets?



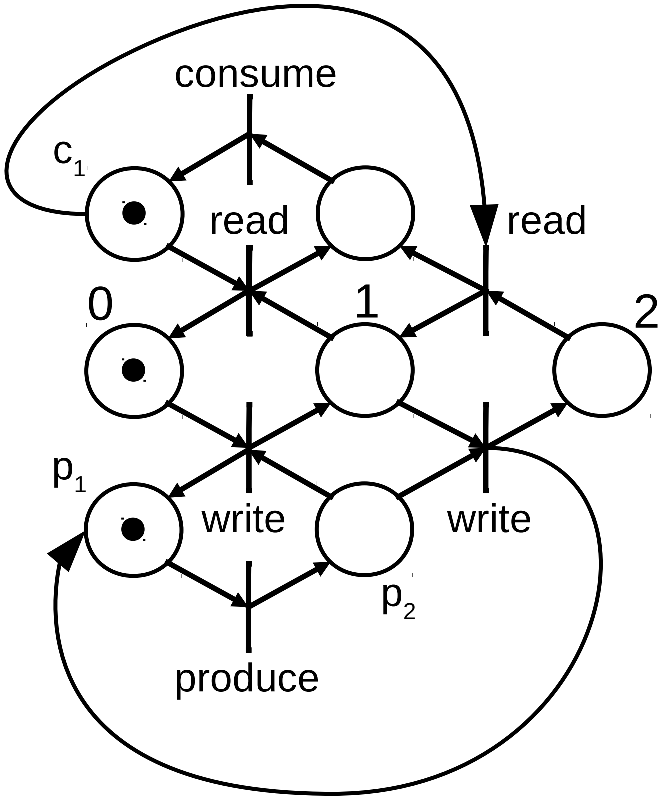
Can you add the consumer?

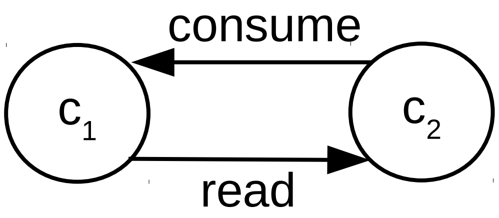




Consumer

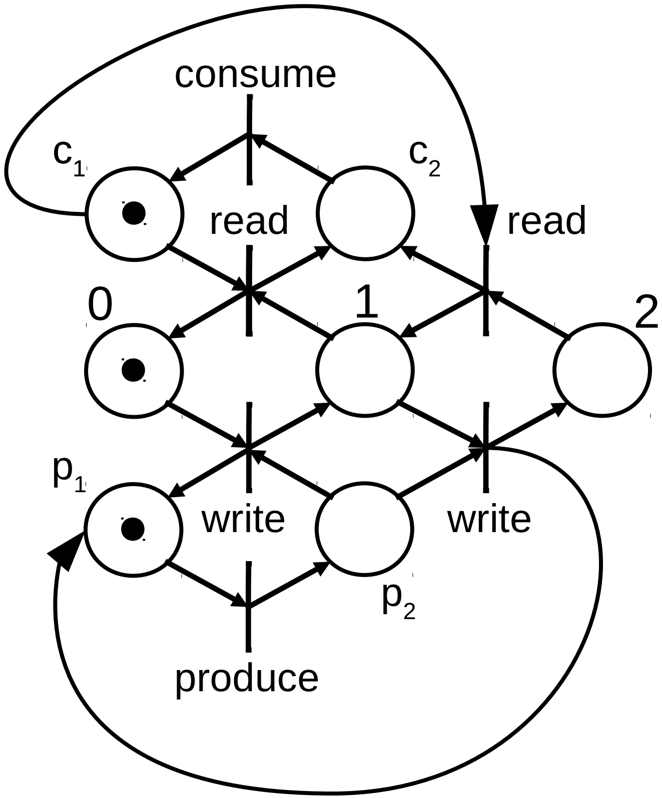
Can you add the consumer?





Consumer

Can you add the consumer?

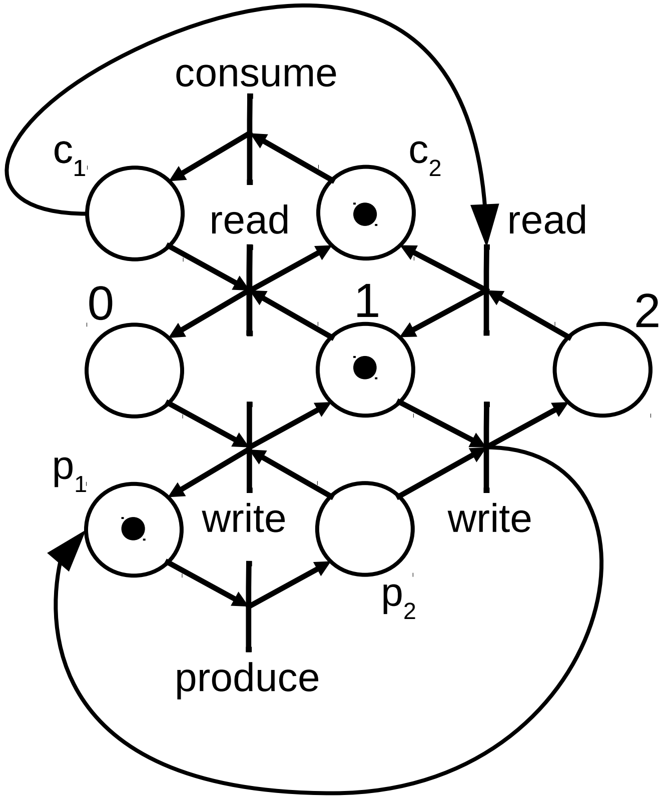


- Which transitions are enabled for this marking?
  - Only produce
    - Which makes sense, since we need to produce something before we can write/read/consume

Combined Petri Nets vs. Combined FSMs

- State space of components is additive for Petri Nets
  - No state space explosion!
- We can express concurrency

Concurrency in the consumer-producer system



- Which transitions are enabled now?
  - Produce and consume
- After firing one of these, the other is still enabled
- Consume and produce are concurrent

Limitations of Petri Nets 1/2

- Tokens are anonymous
  - Do not have any information other than their presence
  - Sufficient to express control flow
  - Insufficient to make a decision based on information in the token

Limitations of Petri Nets 2/2

- Weak selection policy
  - Not possible to specify a selection policy that forces to fire a transition is several others are enabled
    - Can be fixed by assigning priorities to transitions
- No ‘timing’ notion
  - Can’t model how long a computation or a transition takes

Petri Net Extensions

- All these limitations can be overcome using extensions to the standard model
- These extensions follow the same ‘rules’ as Petri Nets
- So you can use them in most standard tools

Petri Net extensions: Hierarchical decomposition

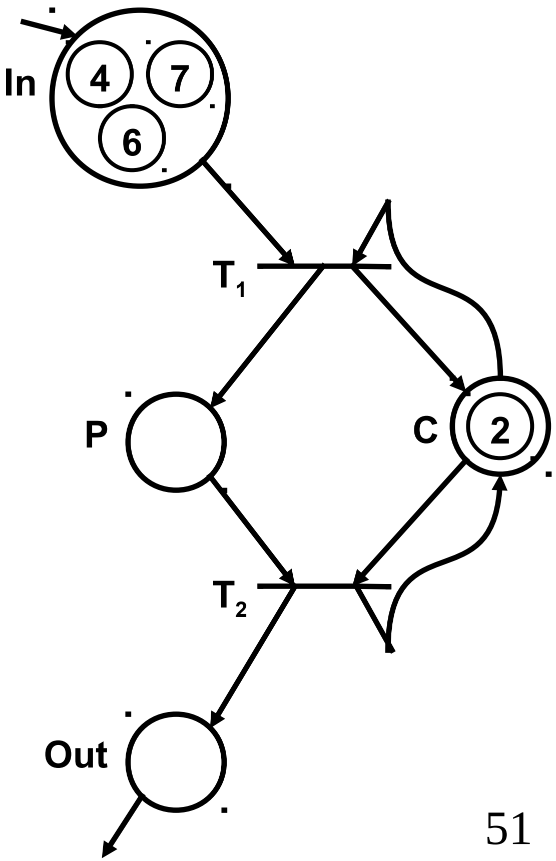
- Transition or place can represent another PN
- Useful in visualization (reduce clutter)
- Helps to think about the model at different levels of abstraction

Petri Net extensions: Assigning values to tokens 1/2

- Tokens can carry values of an appropriate type
  - Integer, string, array, etc.
  - A set of variables
- We are not modeling data structures, but use the extension to model operational behaviour
  - Information in the tokens is used from control point of view

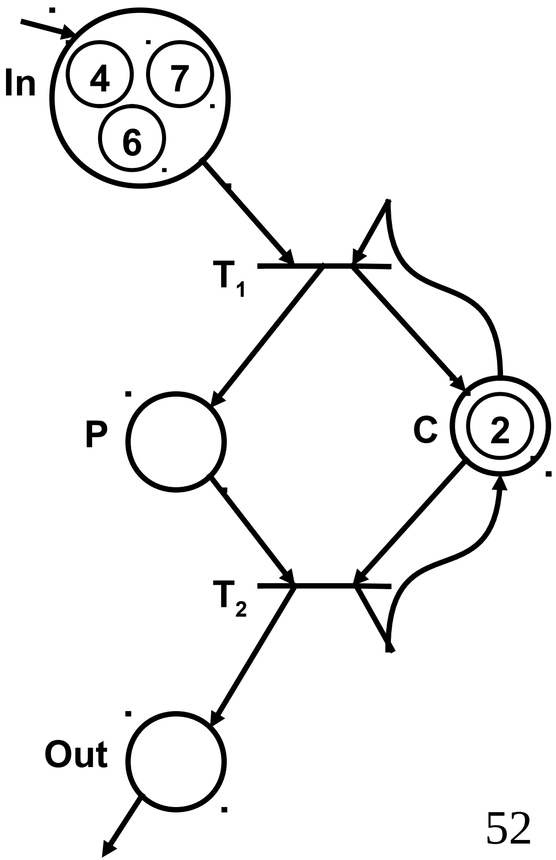
Petri Net extensions: Assigning values to tokens 2/2

- In contains 3 tokens with the values 4, 6 and 7
- C contains one token with the value 2



Petri Net extensions: Execution model 1/2

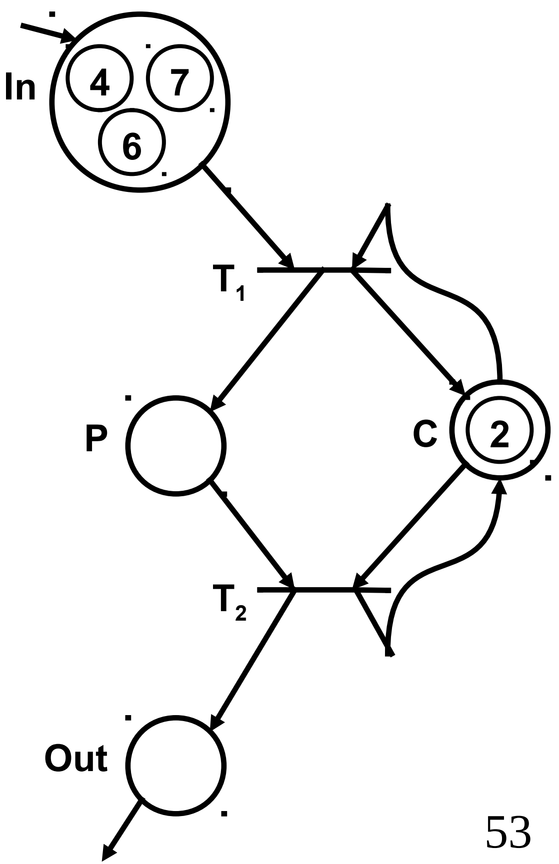
- In this model, transitions have predicates and functions
- We can use the token values together with these predicates and functions to define control flow



Petri Net extensions: Execution model 2/2

transition	predicate	function
T_1	$C > 0$ $In > 5$	$C := C - 1$
T_2	true	$C := C + 1$

- P can only hold up to two tokens
- C makes sure that this is true
  - C will always have a token (but value may be different)
- 2 ‘ready-tuples’ are available
  - Tuples of tokens that can fire a transition
  - 7 and 2; and 6 and 2



Petri Nets example: Message dispatcher system 1/5

- Dispatcher receives messages from 2 channels
- It checks the parity of each message:
  - If parity is wrong, it sends a message through reply channel
  - If parity is OK, it places the message into buffer
- Buffer can store up to 10 messages
- When buffer is full, the dispatcher sends the whole content of it to the processing unit through another channel
- No message can be placed into full buffer

Petri Nets example: Message dispatcher system 2/5

- P1 and P2 are places that model the input channels
  - Bit strings are coming through them
  - Hold regular tokens
- P5 and P6 are places that represent the reply channel
  - Represent the reply because of finding message that failed parity test
  - Hold regular token
- P3 is a place that represents the buffer
  - Stores incoming messages
  - Holds regular tokens
  - Every incoming bit string is attached to the bit string within buffer; buffer holds one token that is a concatenation of all messages
  - We use regular token since its value is irrelevant from the control point of view
- C is a place that counts how many messages are in the buffer
  - Holds **integer** tokens
- P4 is a place that represents the processing unit
  - Token in P4 represents sending the entire content of buffer to the processing unit
  - Holds regular tokens

Petri Nets example: Message dispatcher system 3/5

- T11 is a transition that checks message in P1 for parity
  - If the message has even number of 1’s then it is pushed to P3
- T21 is a transition that checks message in P2 for parity
  - If the message has even number of 1’s then it is pushed to P3
- T12 is a transition that checks message in P1 for non parity
  - If the message has odd number of 1’s then a token is generated in P5
- T22 is a transition that checks message in P2 for non parity
  - If the message has odd number of 1’s then a token is generated in P6
- T3 is a transition that checks if buffer (P3) is full
  - If buffer if full, then its content is sent to P4 and counter is reset

Petri Nets example: Message dispatcher system 4/5

- Definition of predicates and functions:

Transition	Predicate	Function
T11	P1 has even number of 1’s AND C<10	C = C+1, P3 gets a token
T21	P2 has even number of 1’s AND C<10	C = C+1, P3 gets a token
T12	P1 has odd number of 1’s	P5 gets a token
T22	P2 has odd number of 1’s	P6 gets a token
T3	C == 10	C = C – 10, P4 gets a token, P3 gets a token

Petri Nets example: Message dispatcher system 5/5