EEE8068 Real Time Computer Systems
Discrete-event models

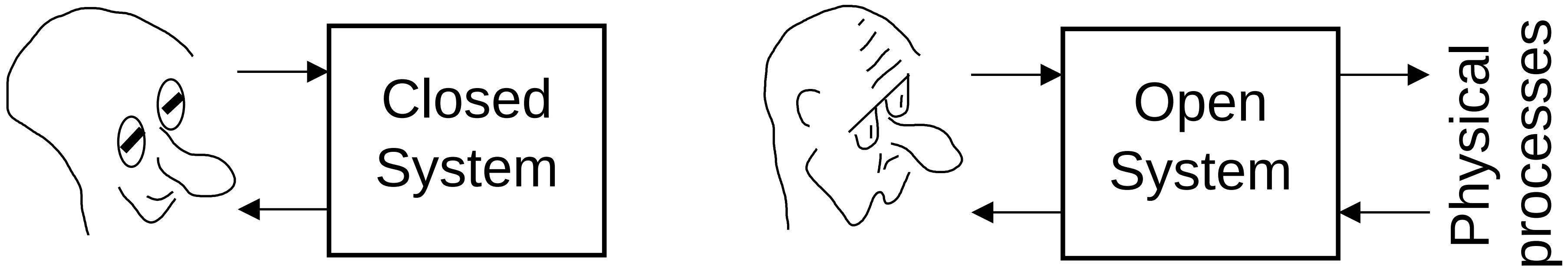
Dr. Bystrov

School of Electrical Electronic and Computer Engineering

University of Newcastle upon Tyne

Reactivity

- Open systems
  - non-determinism from multiple sources, not controllable or observable by the programmer
  - causal relations in the environment are hidden.
- Closed systems
  - execution non-determinism confined to one source
  - causal relations are easily established.



Before we go further – FSM model

- You know what it is – please revisit this topic and find the appropriate reference material (**Home Task**)
- What systems can be implemented with FSMs?
  - Real-time
  - Non-real-time
  - Control parts (mode selection)
  - Security
- What about concurrency? What is concurrency? The discussion of concurrent models (Petri nets and Reachability Graphs) follows the review of FSM

Did I mention FSM? – Revision!

Read e.g. *Williams, R.: “Real-Time systems Development”, Esevier, 2006* and use *EECE CBL* system

- Discrete-event systems
- FSM
- Moore and Mealy types
- States, transitions, trigger events and actions
- Auxiliary variables
- Hierarchical State Charts
- SW implementation options – Chapter 6 from *R.Williams_2006*
- Logic synthesis of FSM (CBL available in the EECE clusters)

Discrete-event systems

- Dynamic system
- Abrupt occurrence of (physical) events
- Possibly irregular time intervals
- Time – included or not included
- Rates and probability distributions – present or not
- States – explicitly or implicitly present
- Concurrency – present or not
- Properties: valid/invalid states, deadlock, liveness, implementability, etc.
- Use – refinement of high-level systems
- Use – high-level ad-hoc design (controlling agents, plants, valid and forbidden states and behaviours)

Finite State Diagram

Bare-bone model (logic I/O not included)

- FSD is a tuple $\Sigma = \langle S, T, s_0 \rangle$
- FSD is a directed graph
- FSD nodes
  - states $S = \{s_0, ..., s_k\}$
  - s_0 is the initital state
- Transitions $T \subseteq (S \times S)$.

Additional components may be introduced:

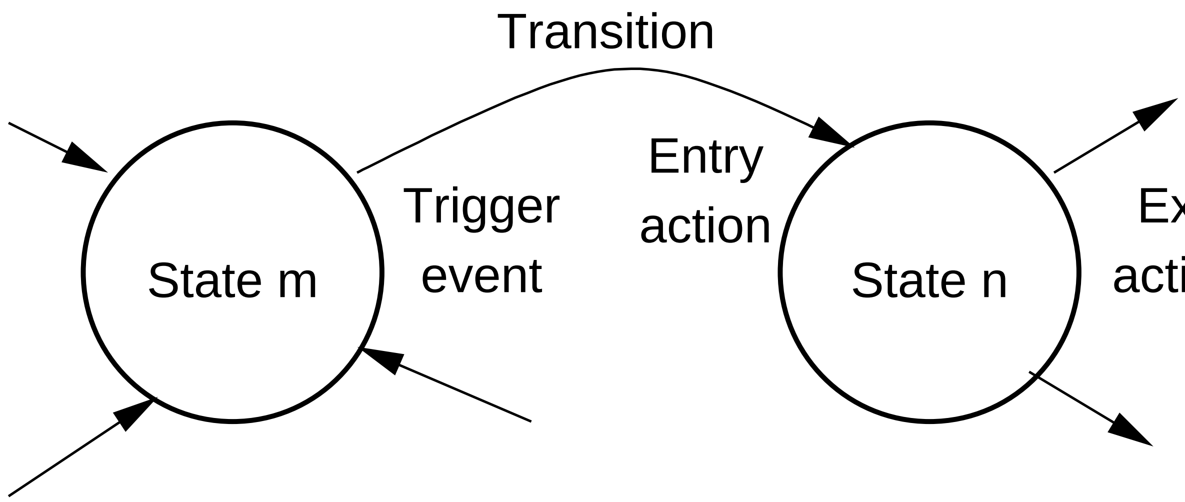
- State labels or encoding
- Transitions guards and Auxiliaty variables
- Trigger events and Actions

Moore and Mealy types

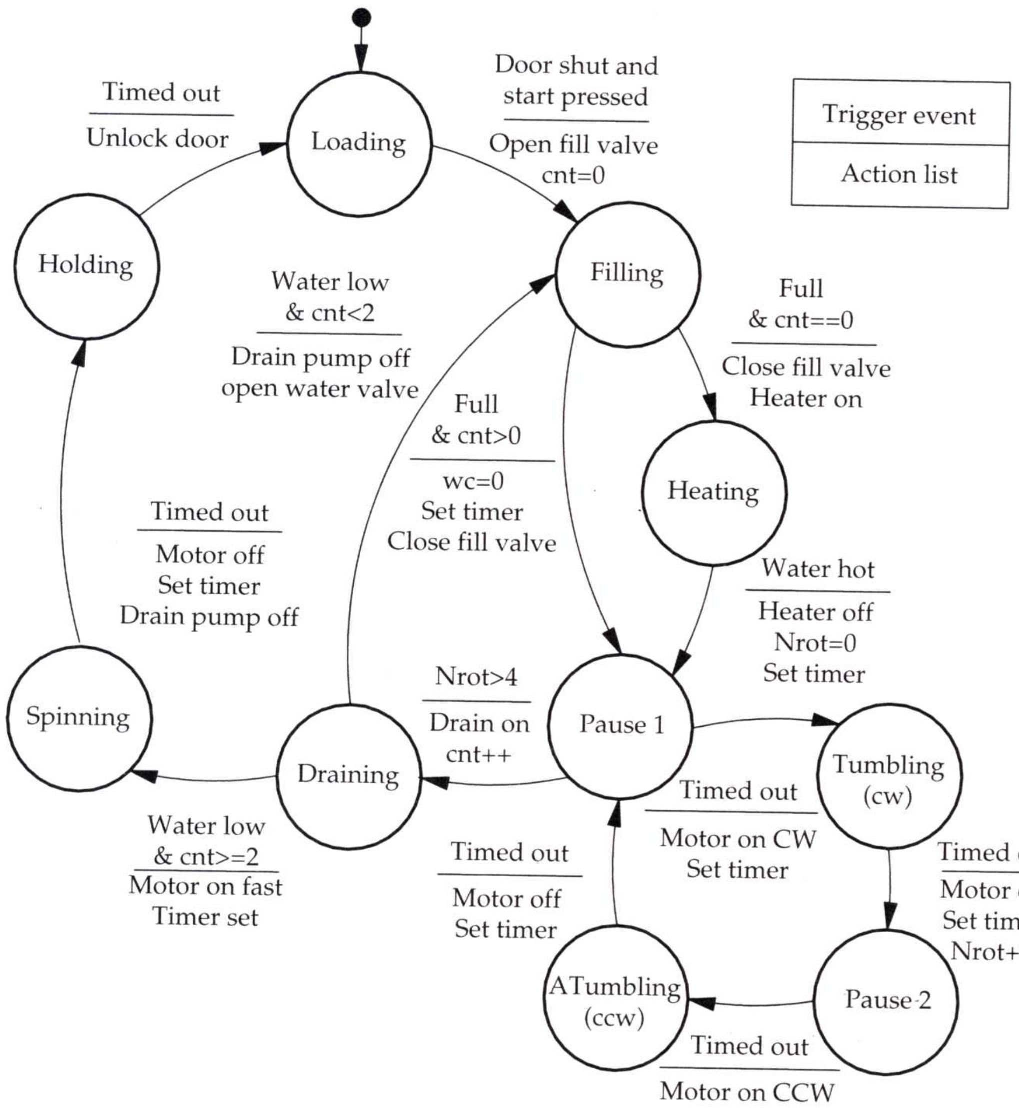
This interpretation is different from the logic synthesis (your examples).

Moore: activities take place either inside the states or on leaving a state

Mealy: activities are evoked on entry to a state



Example of a washing machine

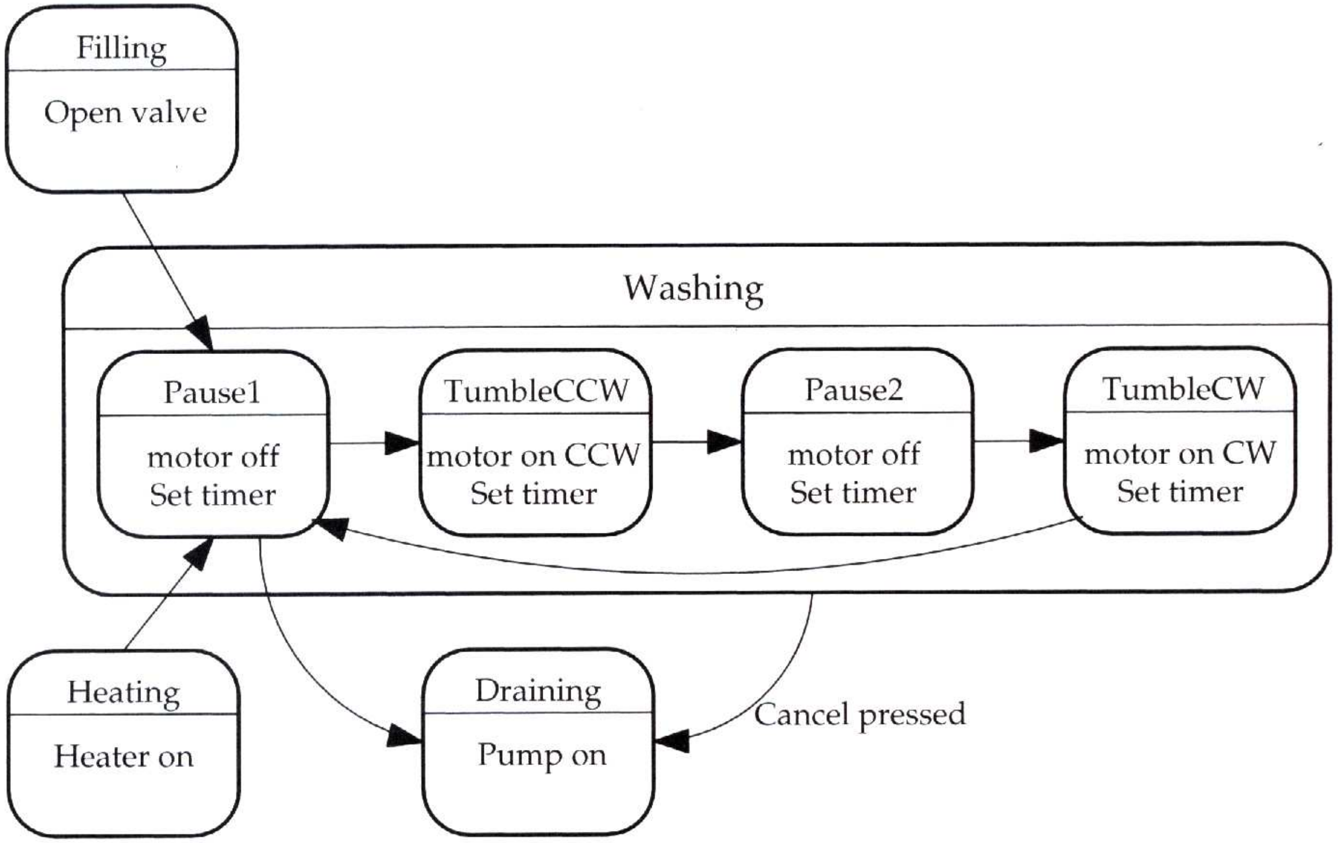

Trigger event
Action list
Loading
Door shut and start pressed / Open fill valve cnt=0
Filling
Full & cnt==0 / Close fill valve Heater on
Heating
Water hot / Heater off Nrot=0 Set timer
Full & cnt>0 / wc=0 Set timer Close fill valve
Pause 1
Timed out / Motor on CW Set timer
Tumbling (cw)
Timed out / Motor off Set timer Nrot++
Pause 2
Timed out / Motor on CCW
ATumbling (ccw)
Timed out / Motor off Set timer
Nrot>4 / Drain on cnt++
Draining
Water low & cnt<2 / Drain pump off open water valve
Water low & cnt>=2 / Motor on fast Timer set
Spinning
Timed out / Motor off Set timer Drain pump off
Holding
Timed out / Unlock door

Auxiliary variables

You have seen them in the previous illustration...

- In a bare-bone FSM all memories of the past are explicit.
  - Traces
  - Context
  - Example
- Compression of a diagram by introducing global Aux. variables
  - Traces
  - Context
  - Equivalence
  - Think of examples!

Hierarchical finite state diagrams



It is all about *encapsualtion* and *equivalence*.
Weak bisimulation may be violated as in the above example!

New Concept – Concurrency

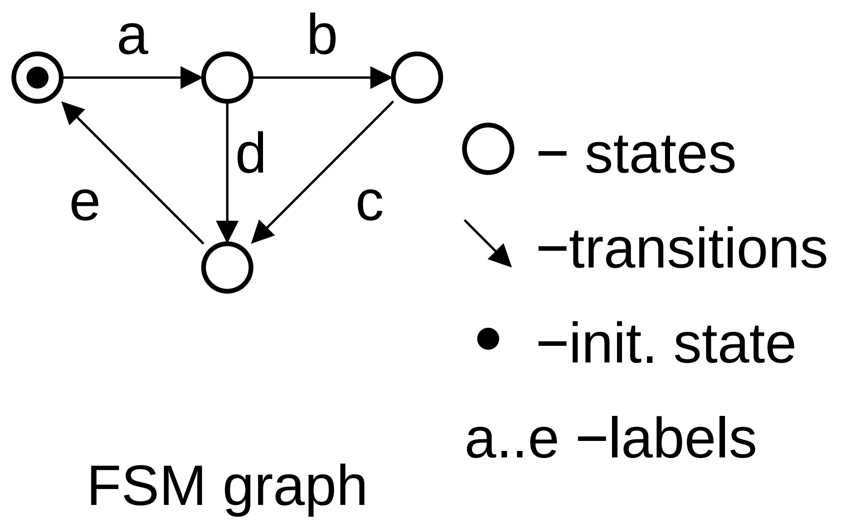
Independent processes are concurrent.

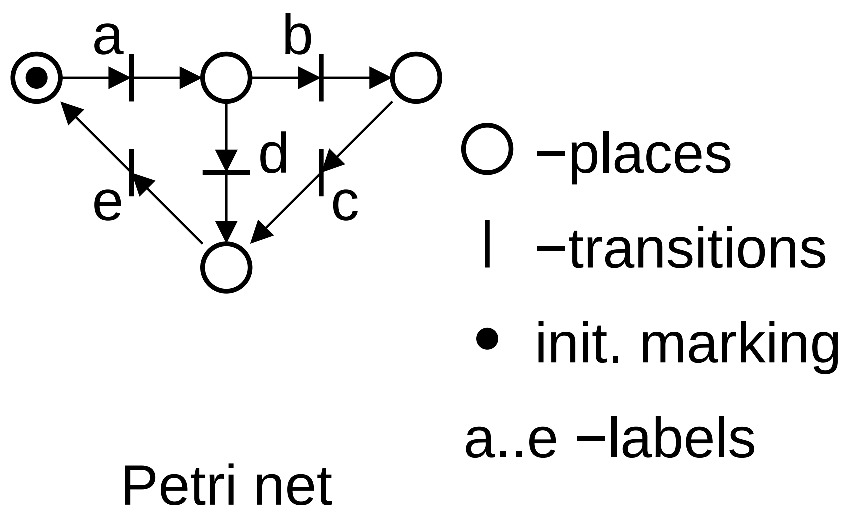
- Everything in nature is related to some extent. . .
- Is there true concurrency somewhere?
    - Have you seen our EM shielded room?
- On the other hand. . . does a complete order exist in nature?
- Concurrency is an abstraction opposed to the abstraction of the order.
- The execution of concurrent processes may overlap in time.

Please give me a better/your definition!

New Model – Petri nets

Intuition: Petri nets are similar to FSM graphs.





Difference:

- Transitions labels in FSM graphs can be associated with logic expressions “guards” controlling the transition.
- Petri nets can have several “tokens” present at a time.
- Petri nets are designed to capture concurrency. . .

Petri net definition

- **PN is a tuple** $\Sigma = \langle P, T, F, M_0 \rangle$
- PN is a directed graph
- PN has 2 types of nodes
  - **places** $P = \{p_0, ..., p_k\}$ ○
  - **transitions** $T = \{t_0, ..., t_n\}$ |
- **Flow relation** $F \subseteq (P \times T) \cup (T \times P)$; ↘
- **Initial marking** is a mapping $M_0 : P \to N \,|N = \{0, 1, 2, ...\}$; ⊙
- **1-safe** PNs: place capacity=1,
  - for 1-safe PNs $N = \{0, 1\}$, “1” for a **token** in the place

<u>We will be using 1-safe PNs only!!!</u>

Rules of the token game

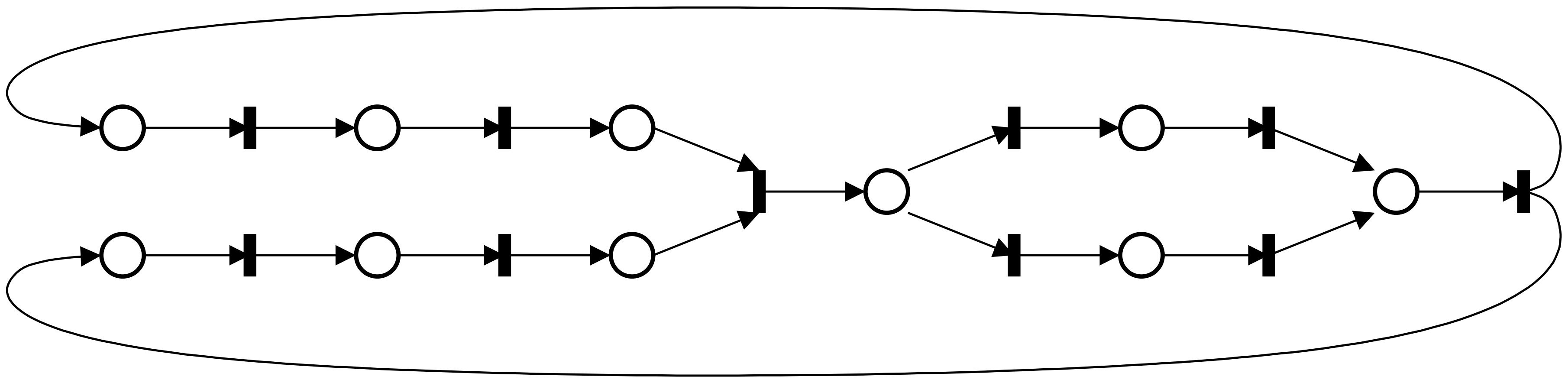
Tokens can move from place to place by “firing” transitions, leading to new markings – **token game**.

A few more definitions (1-safe PNs):

- a **marking** is a mapping $M : P \rightarrow N$;
- **preset** of node x: $\bullet x = \{y \,|\, (y,x) \in F\}$;
- **postset** of node x: $x\bullet = \{y \,|(x,y) \in F\}$;
- transition $t \in T$ is **enabled** if $\bullet t \overset{M}{\rightarrow} 1$; the enabled transition is denoted as t^*;
- enabled transitions may **fire** (or may not);
- **firing** enforces $\bullet t \overset{M}{\rightarrow} 0$ and $t\bullet \overset{M}{\rightarrow} 1$, thus leading to the new marking M'.

Playing the token game

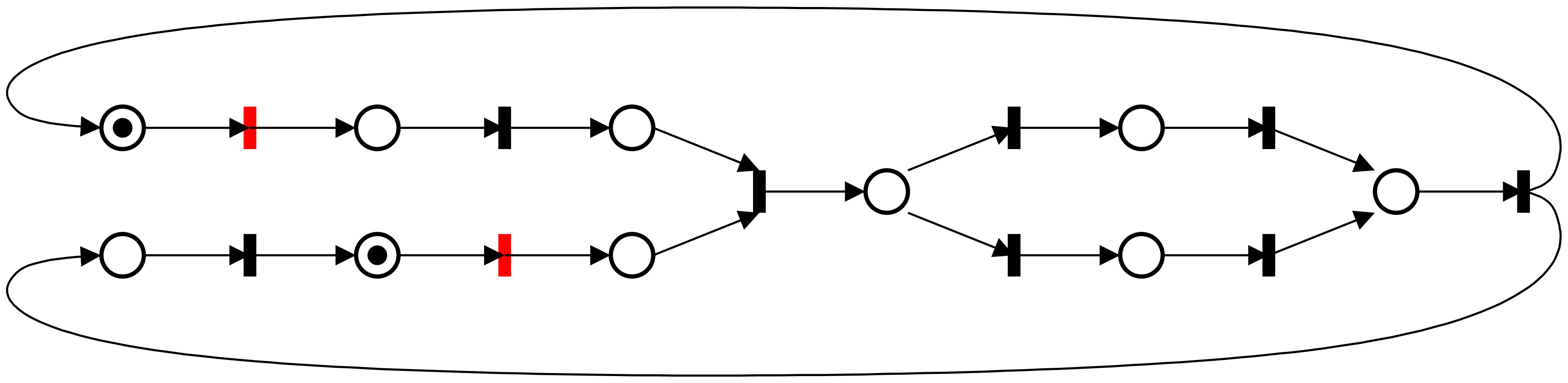
Do you play board games?



Barebones of the PN.
Should we inject some life in it?

Playing the token game

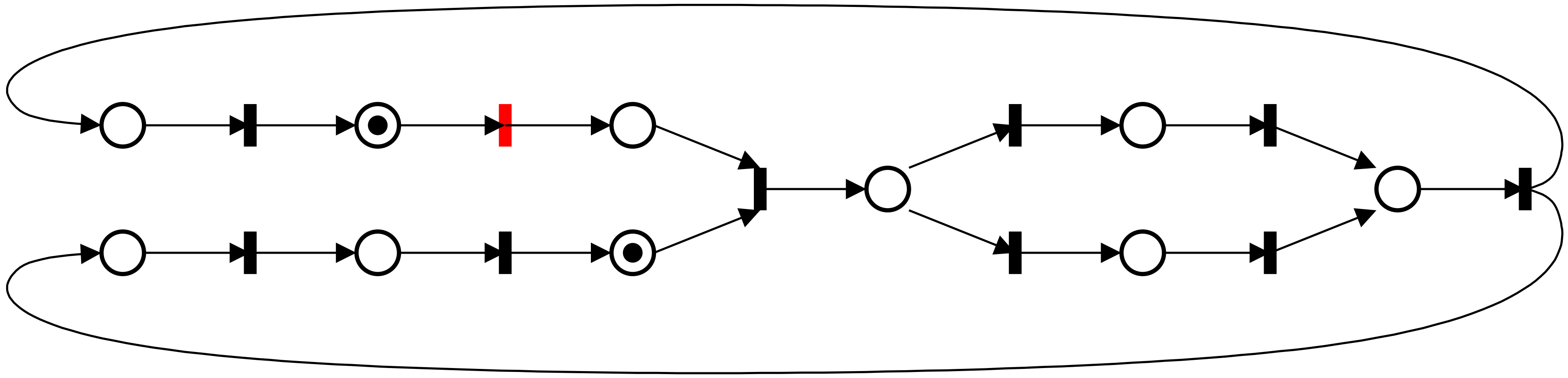
Do you play board games?



Two tokens are injected, thus forming the initial marking.
Red transitions are enabled.
They can fire any time (immidiatelly or a year after the next year. . .)
The continuously enabled transition will eventually fire.

Playing the token game

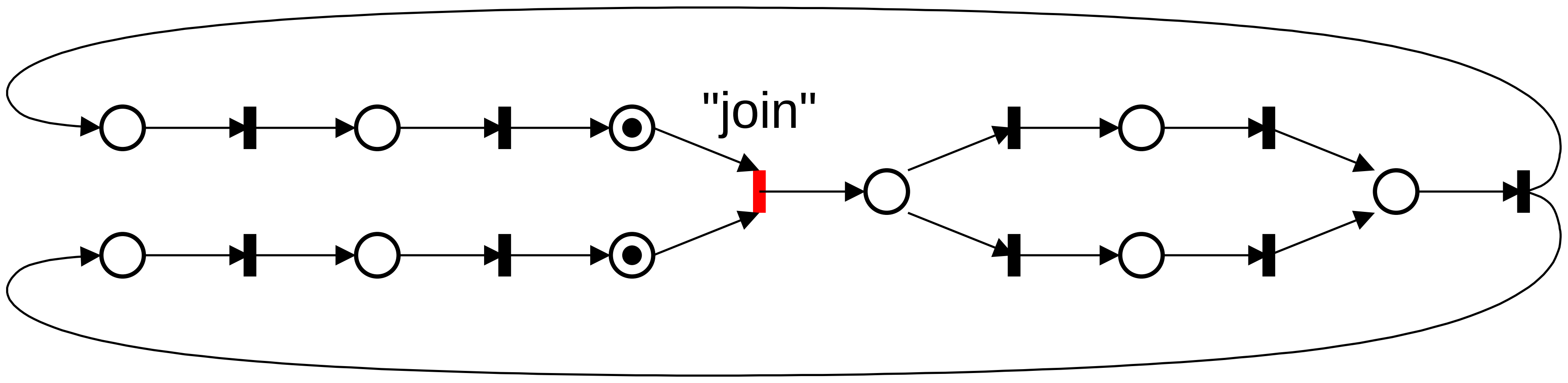
Do you play board games?



Both fired in the same time.
. . . can happen, but very unlikely in real life.

Playing the token game

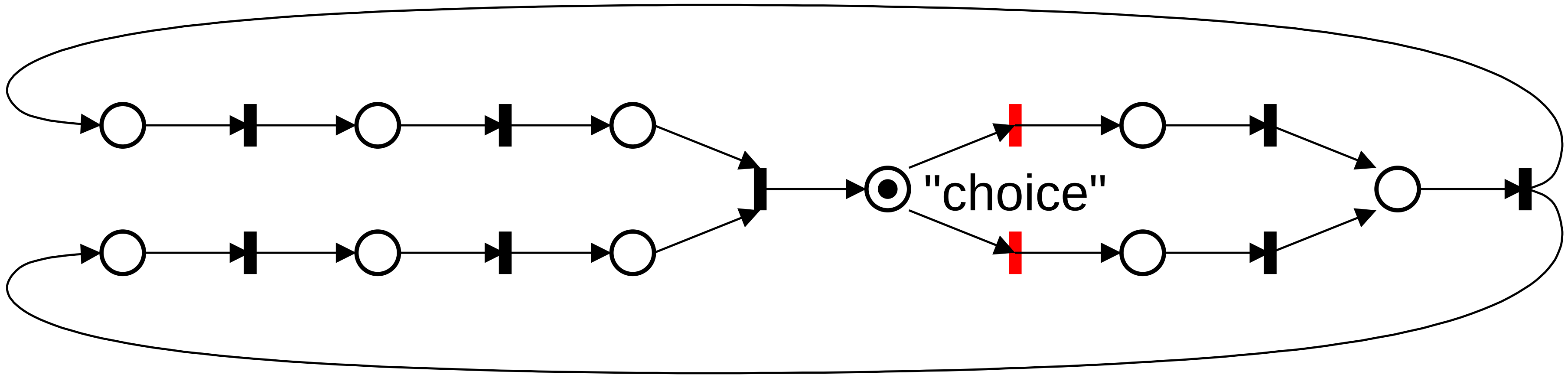
Do you play board games?



“Join” transition performes synchronisation between threads.
It becomes enabled only after all input tokens have gathered together in its predecessors.
It is similar to “AND” operation.

Playing the token game

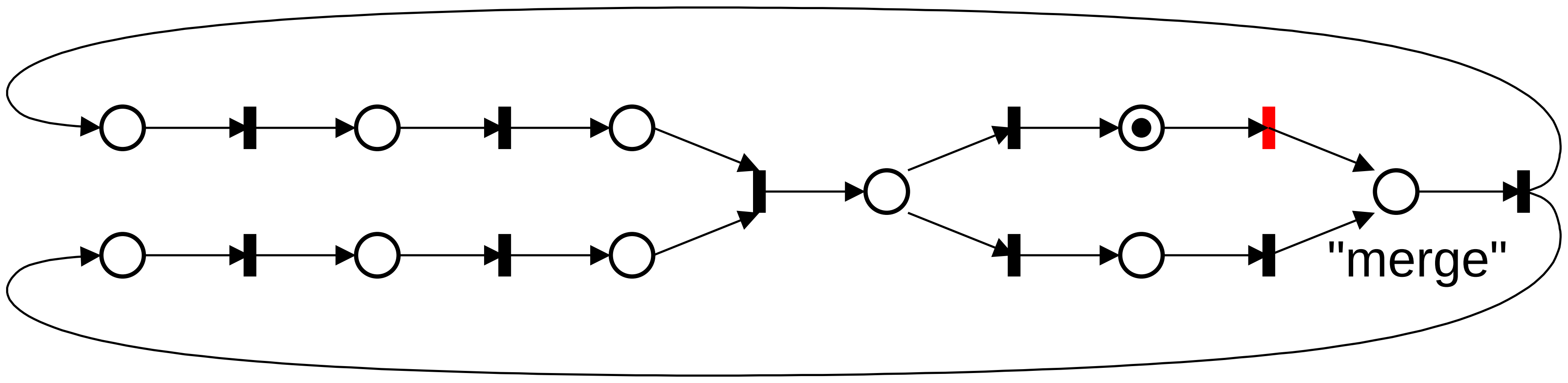
Do you play board games?



Now the atomic (unsplitable) token must deside which way to take.
It is making its “choice”.
Once it is gone, both transitions become **disabled**.
One of them will be disabled without firing – potential **hazard**!!!

Playing the token game

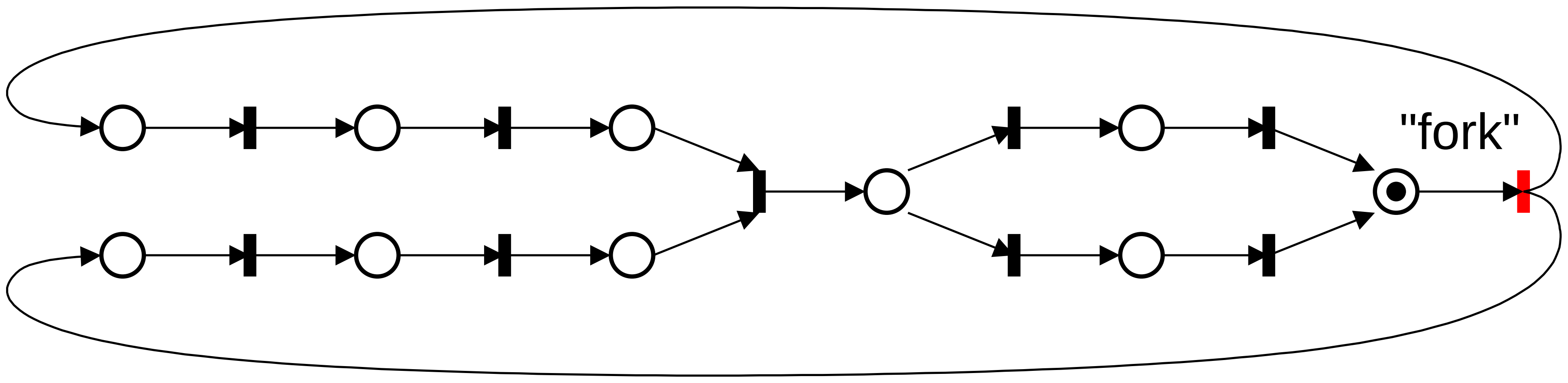
Do you play board games?



“Merge” place is avaiting for at least one predecessor to fire.
It is similar to “OR” operation.
It may exceed its capacity if both predecessosrs fire (“**unsafe** net”)!!!
This should not happen in a correct 1-safe PN.

Playing the token game

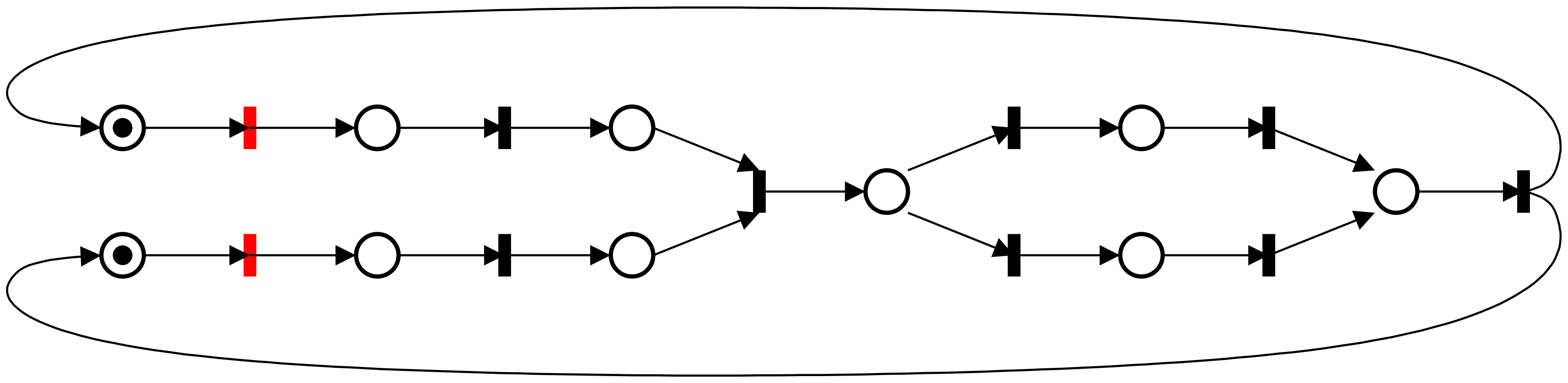
Do you play board games?



This is how we multiply tokens. . .

Playing the token game

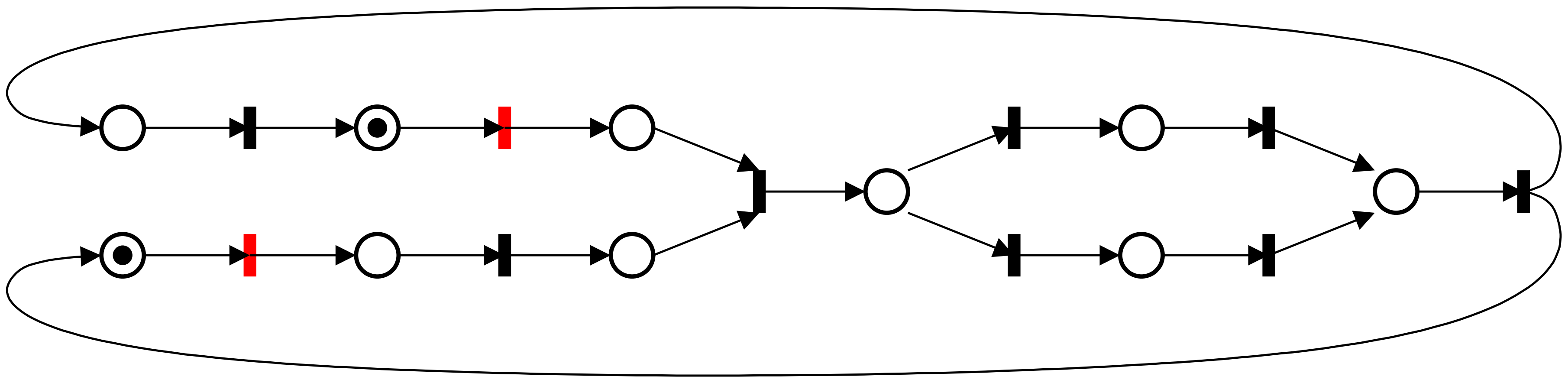
Do you play board games?



Keeping firing in no particular order.
Marked places are concurrent and hence independent.

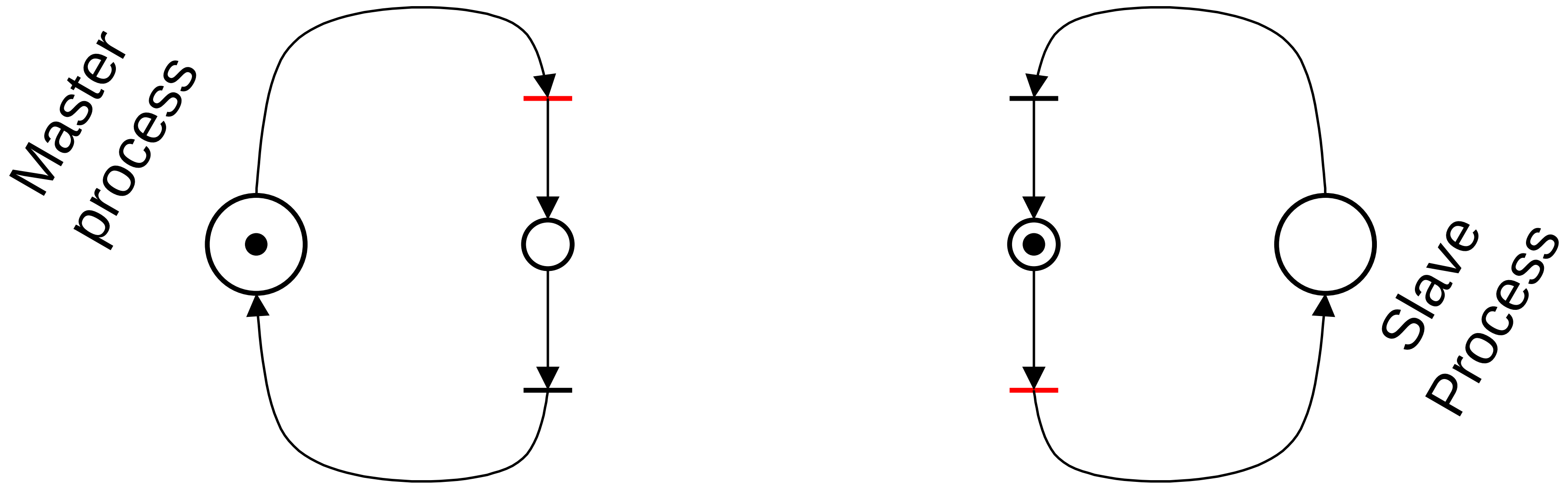
Playing the token game

Do you play board games?



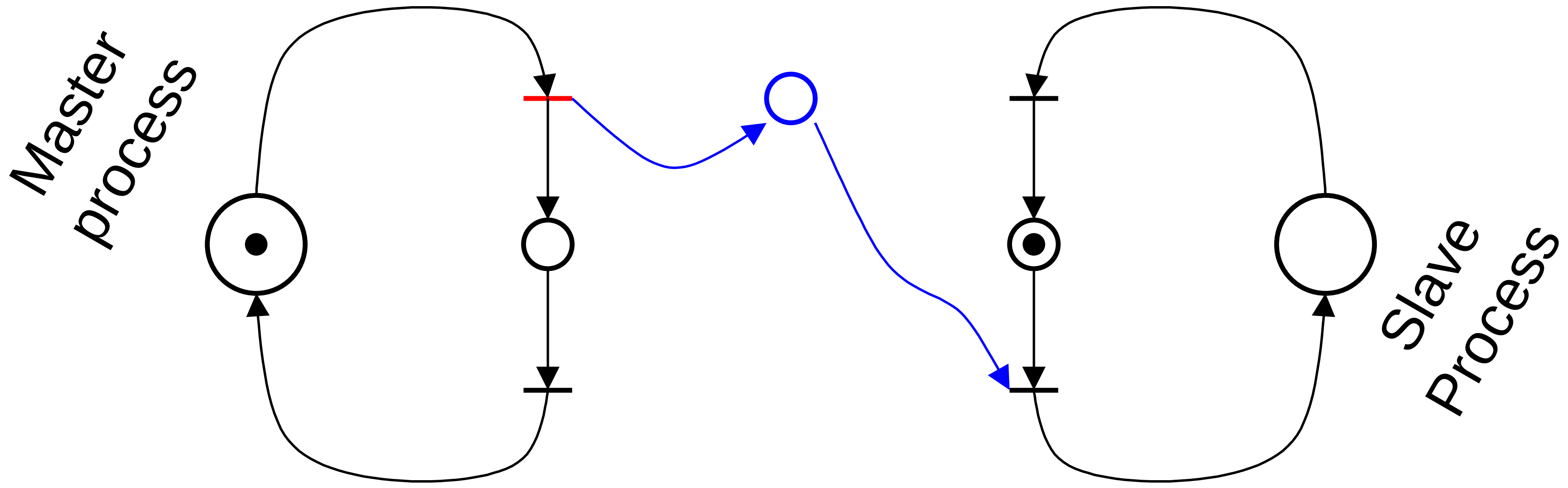
Dough!!! We’ve missed the initial marking!
We may arrive to it after the next cycle or after a couple of cycles or . . . never.
However, it is clearly reachable.
That is why we call it a “**reachable marking**”.

Simple communication example



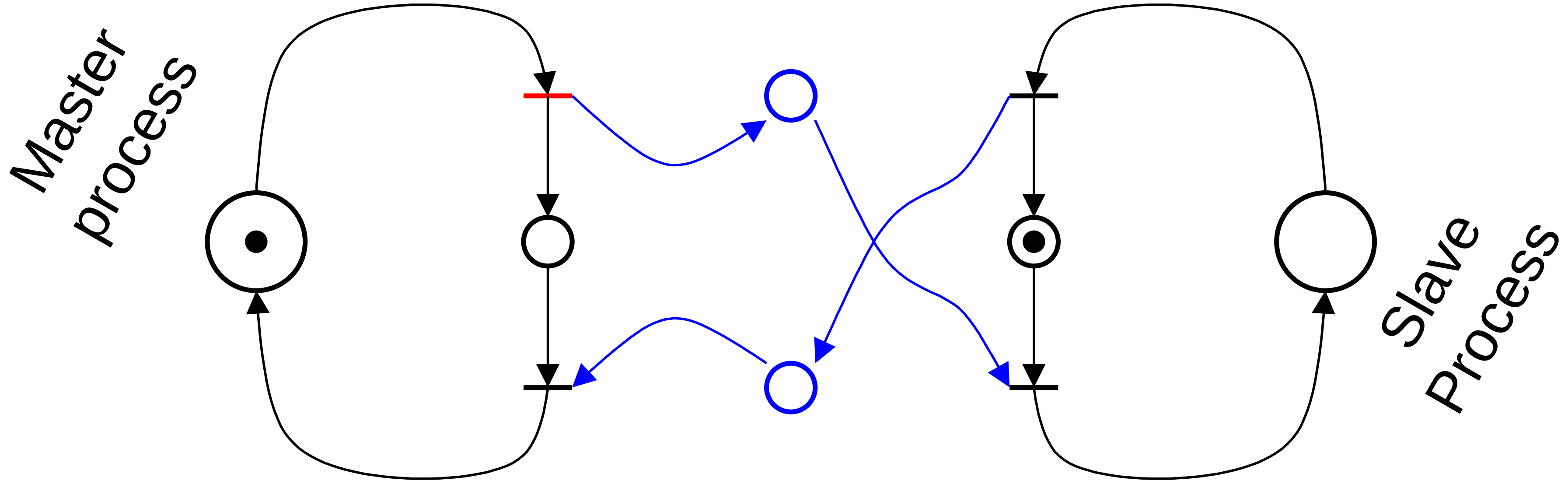
- Two processes are given.
- They are very similar and independent for now.
- How to make them play the roles of a Master and a Slave?

Simple communication example



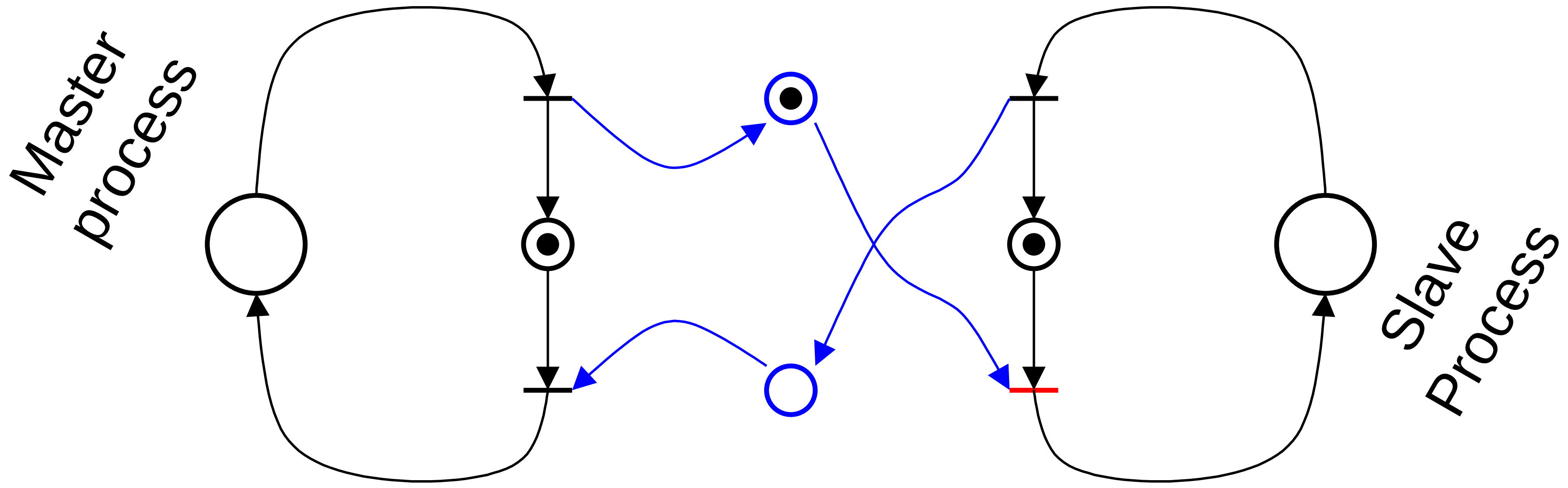
- After Master finishes (produces data), Slave starts (consumes data).
- This is the **producer-consumer** relationship.
- What is the trouble here?

Simple communication example



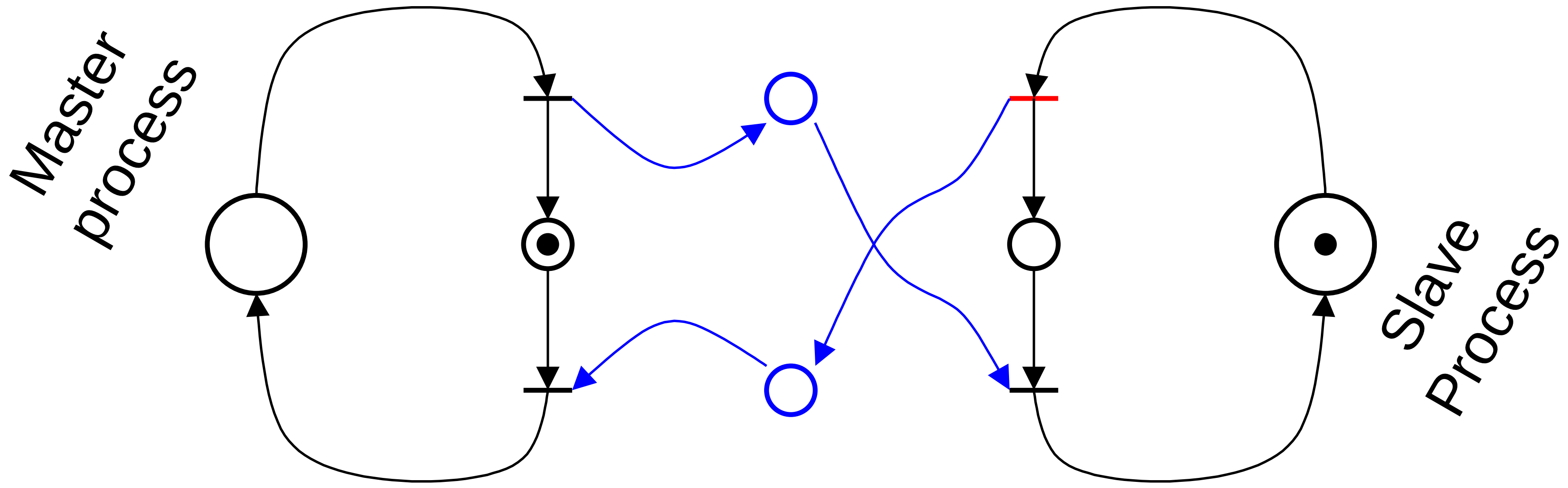
- Preventing Master from flooding Slave with tokens.
- Feedback.
- Request-acknowledgement handshake.
- Delay-insensitive interface.
- Token game begins!

Simple communication example



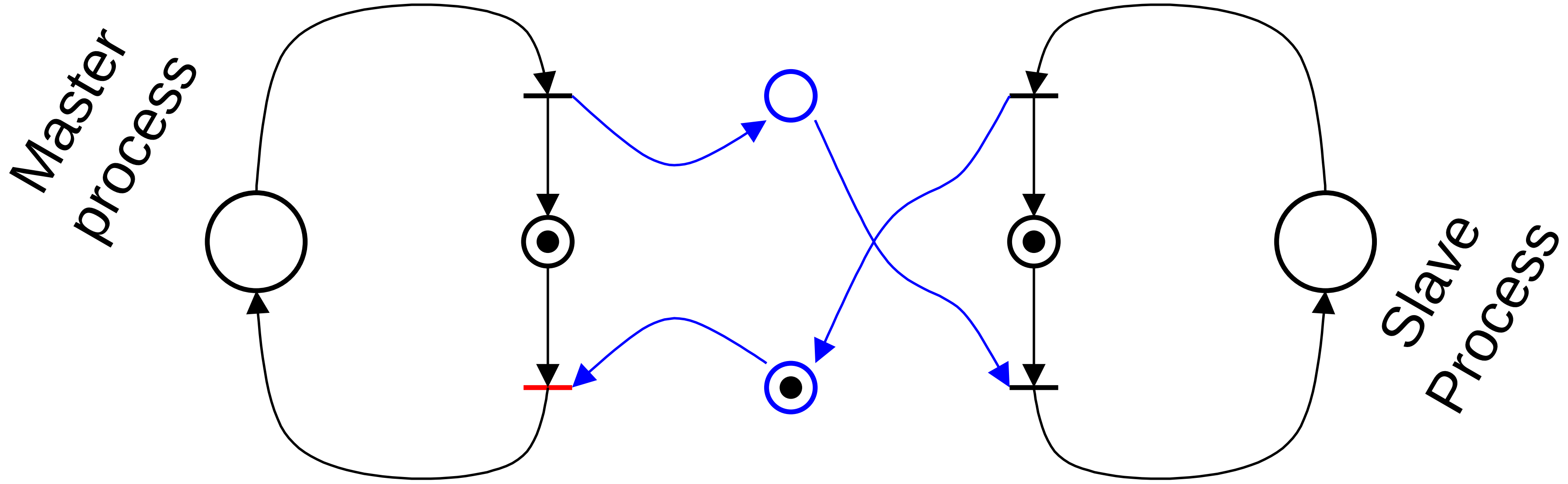
- Master process is finished.
- Request is issued by Master.
- Request input of Slave is enabled and about to fire.

Simple communication example



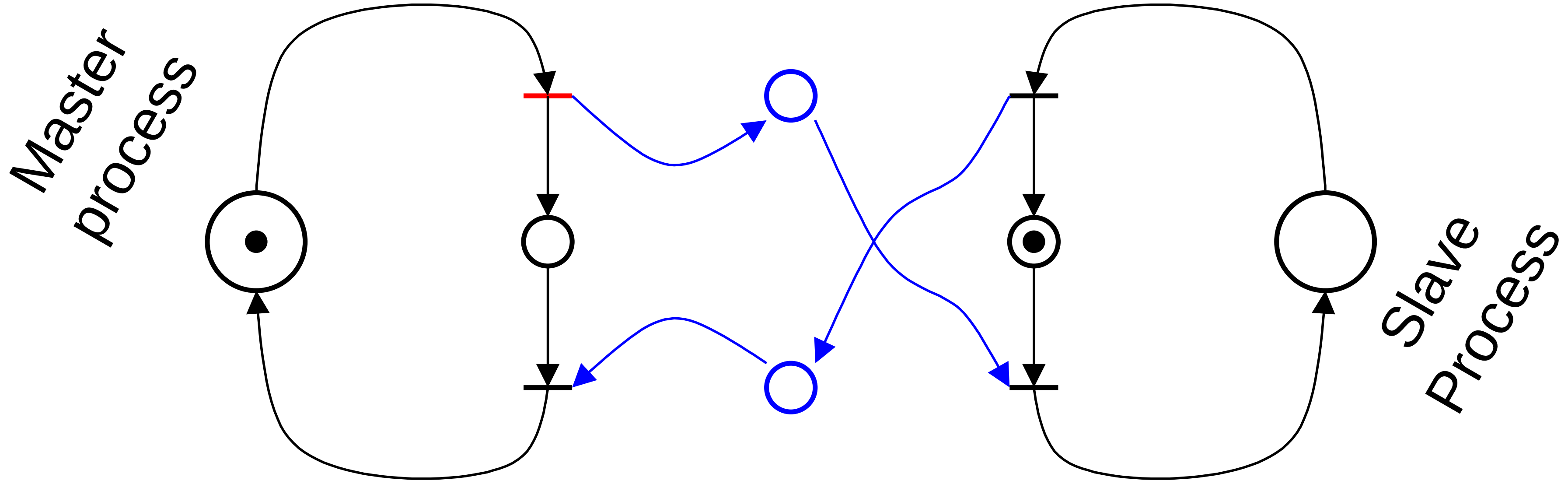
- Slave request has fired.
- Slave process has started.
- Slave acknowledgement is enabled, it will fire after Slave process finishes.

Simple communication example



- Slave acknowledgement has fired.
- Master acknowledgement is enabled and is about to fire.

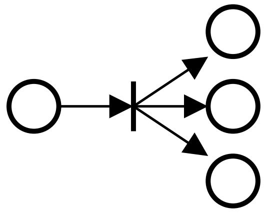
Simple communication example



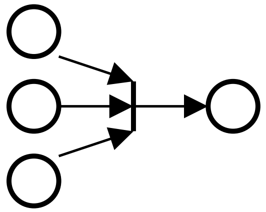
- Master acknowledgement has fired.
- Master starts the new process.
- We have arrived back to the initial marking.

PN building blocks: Fork-join

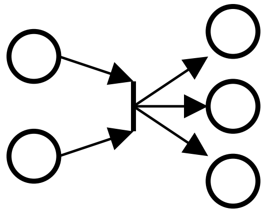
- "Fork" starts concurrent processes.
  - Increases concurrency.



- "Join" synchronises concurrent processes.
  - Reduces concurrency



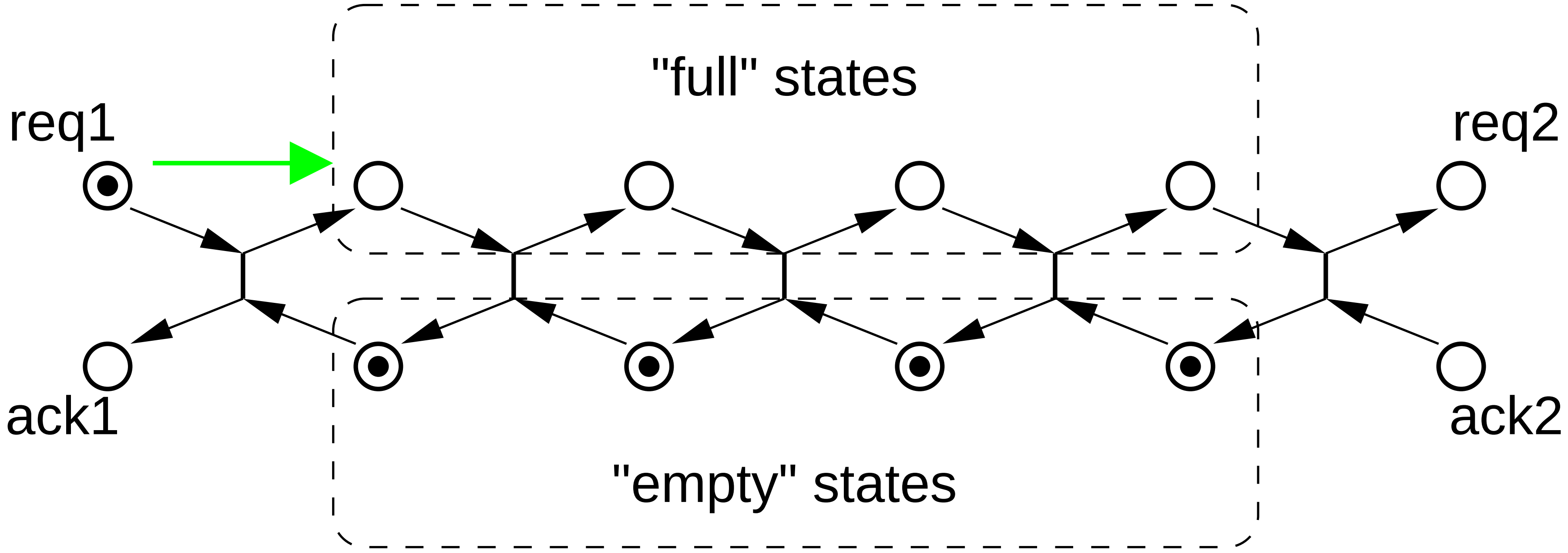
- Combined fork-join.
  - Synchronises concurrent processes.
  - Generates new concurrency.



- Less synchronisation $\Rightarrow$ more freedom $\Rightarrow$ faster system!

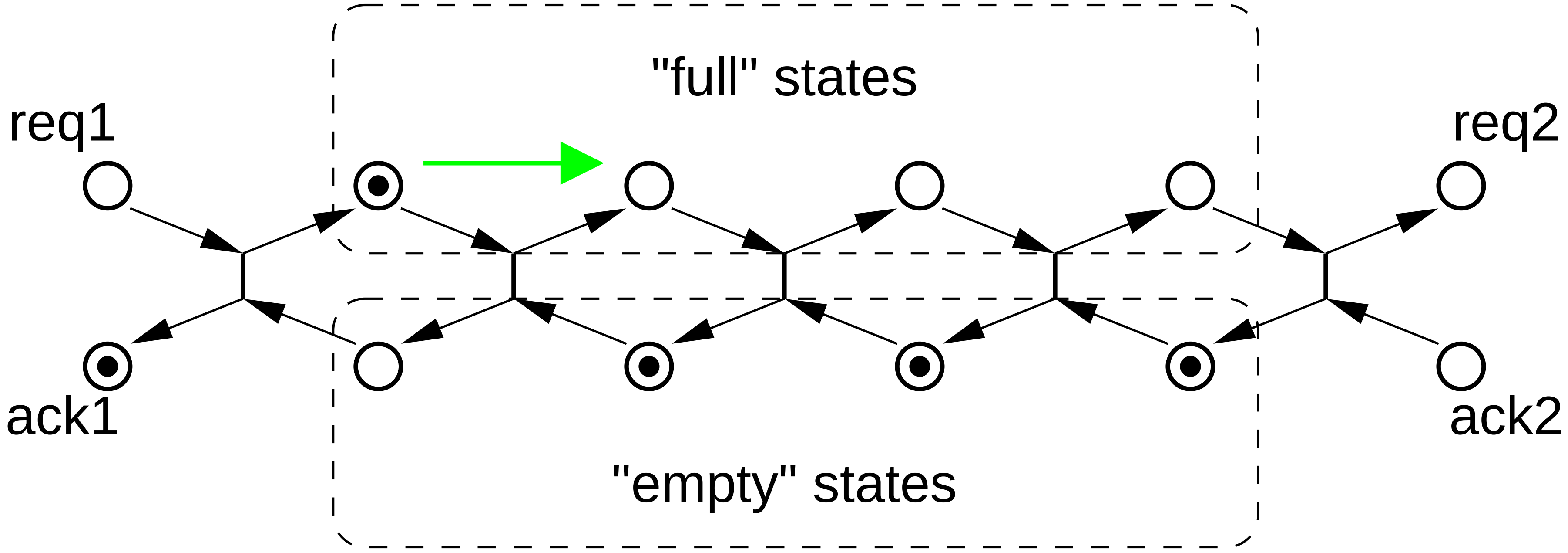
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



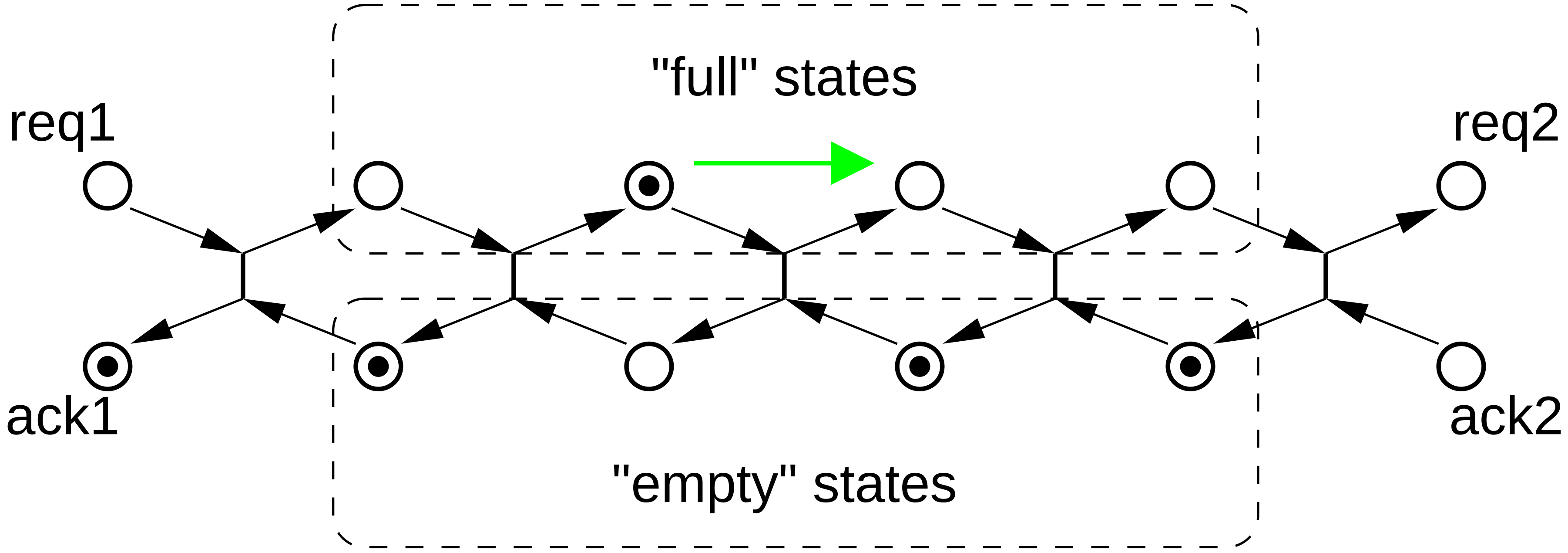
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



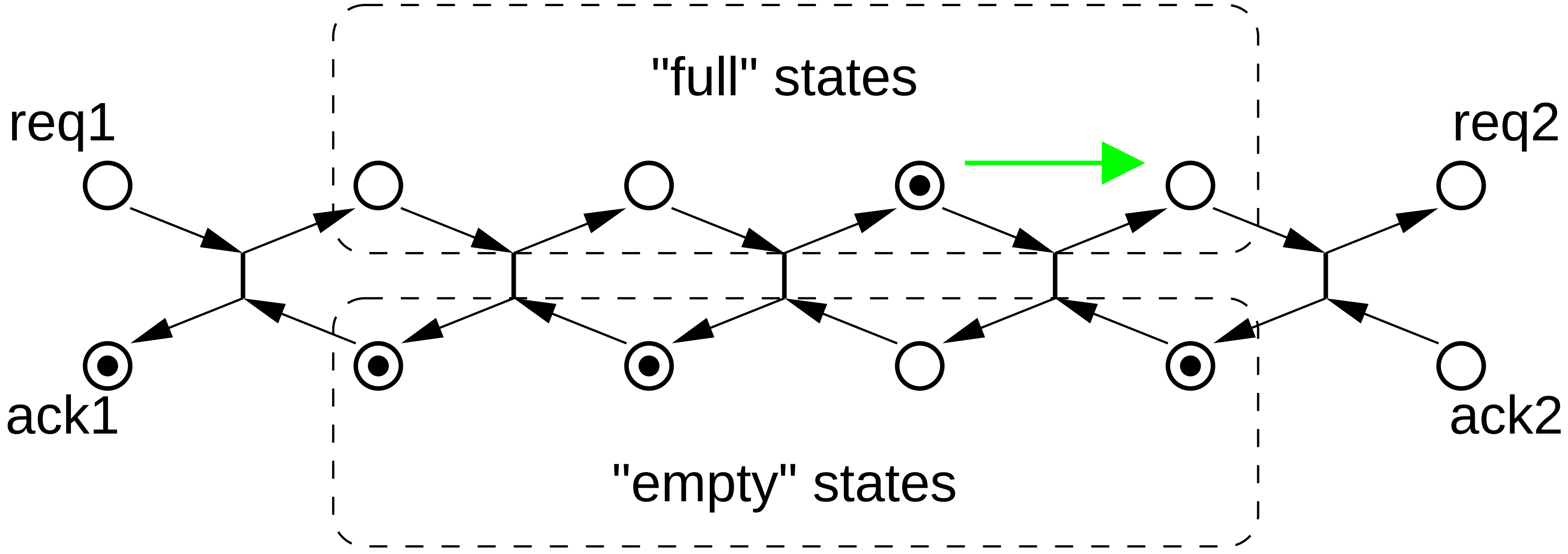
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



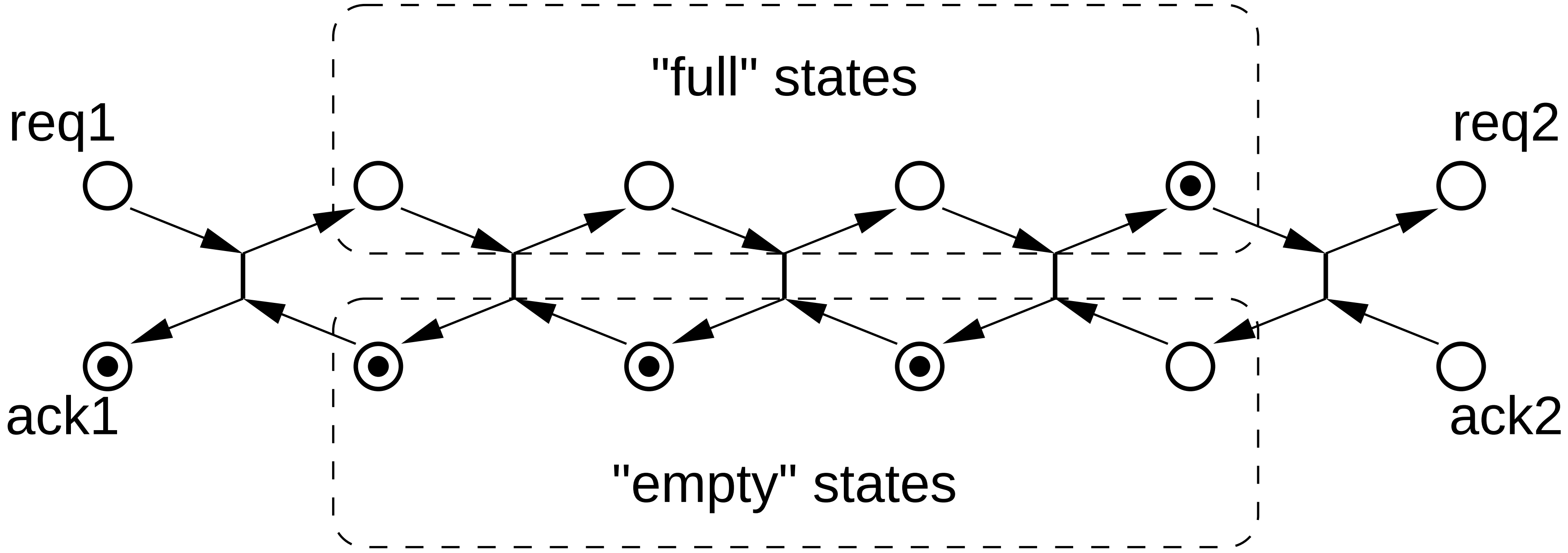
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



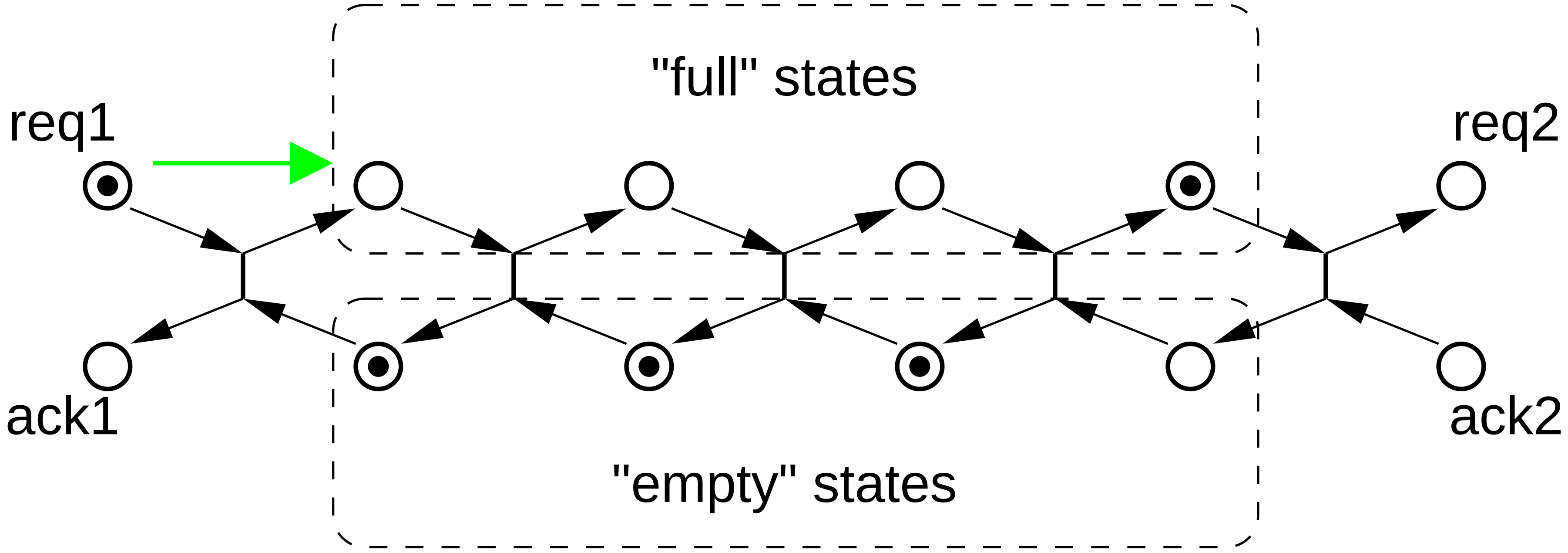
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



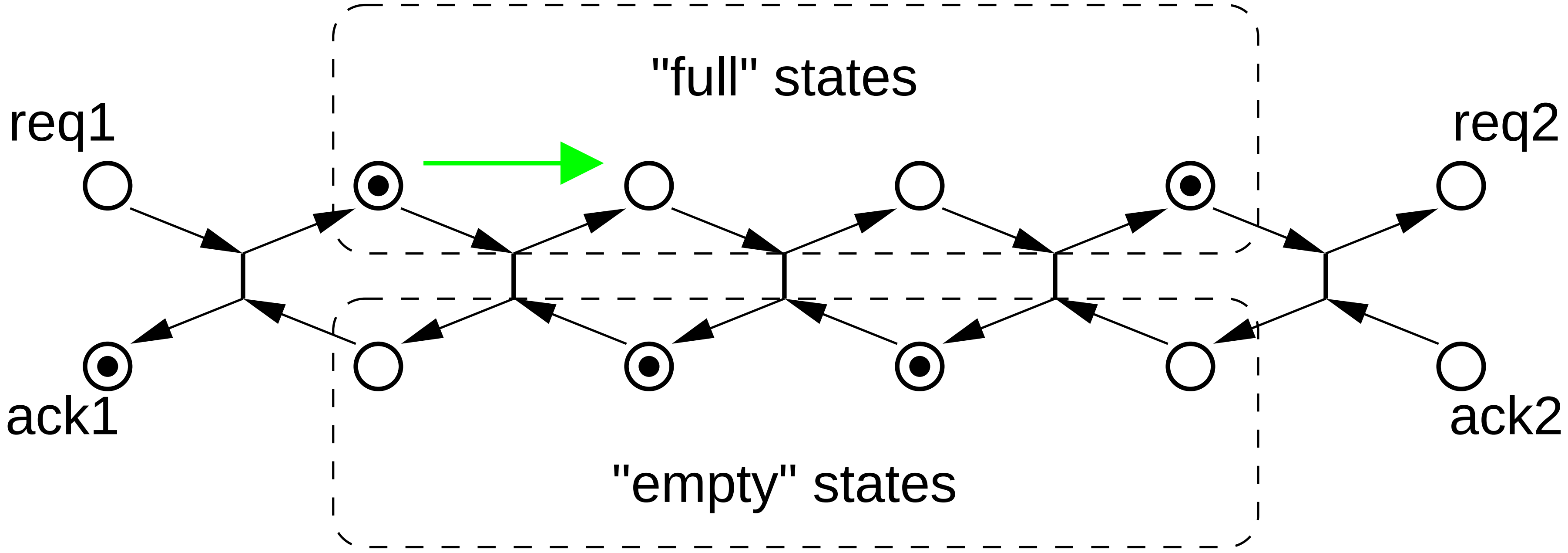
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



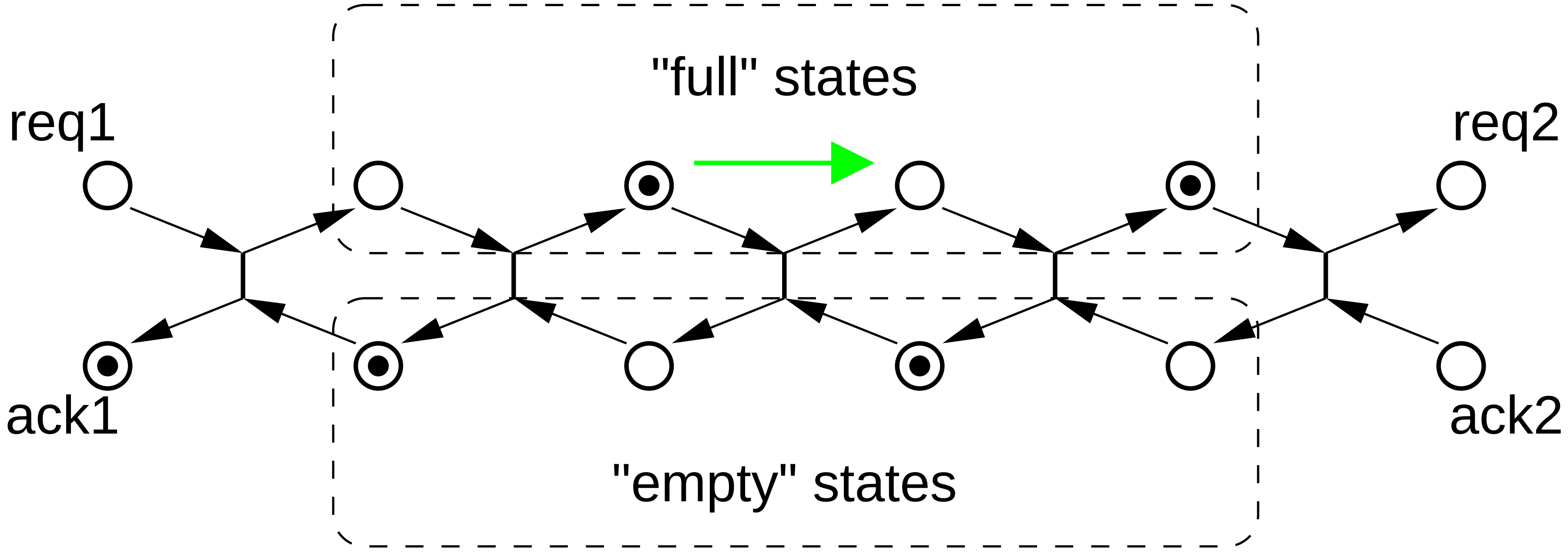
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



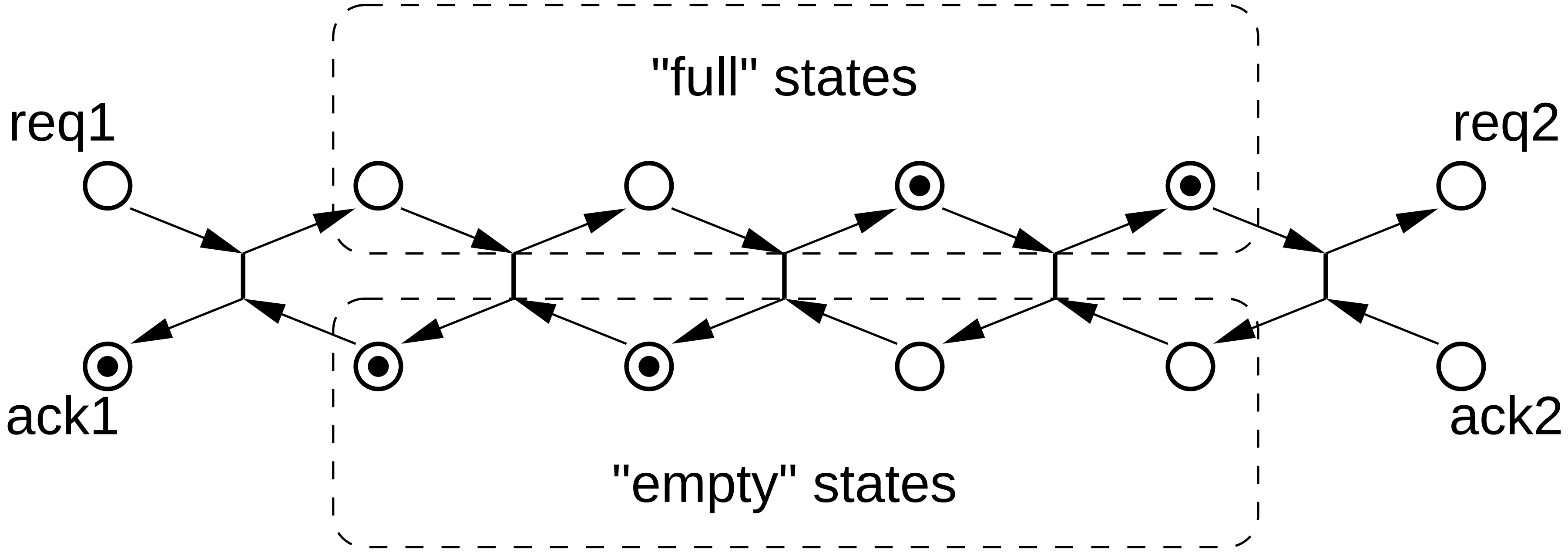
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



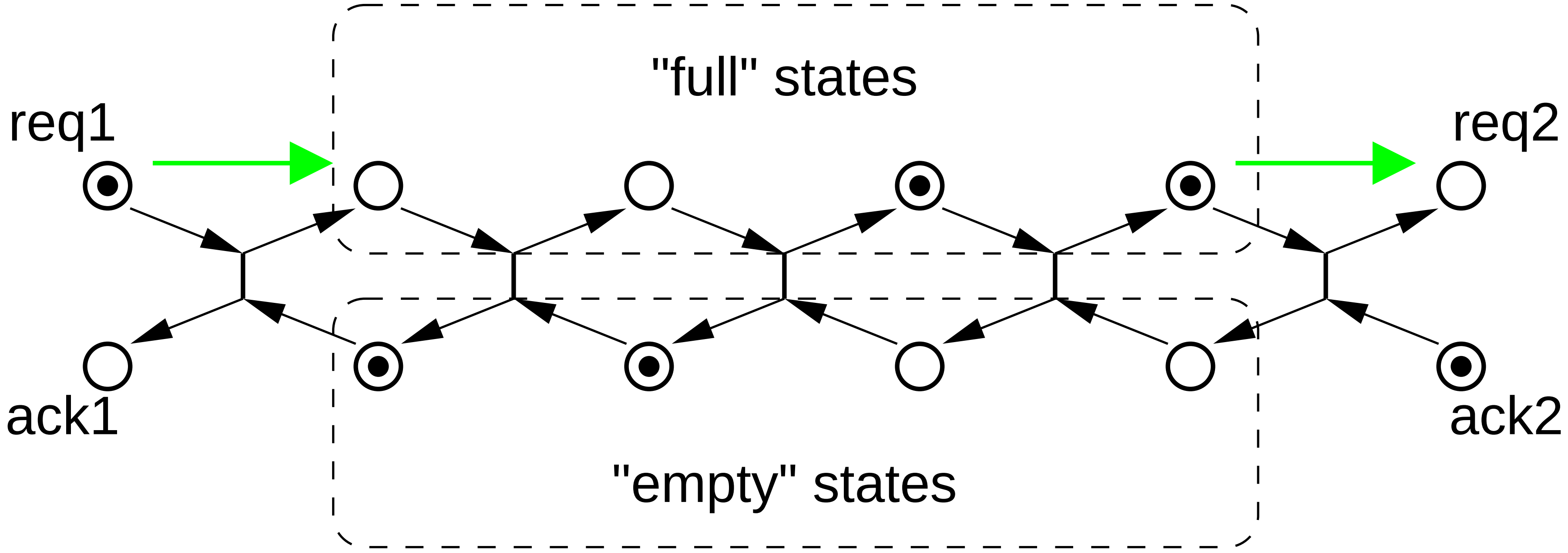
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



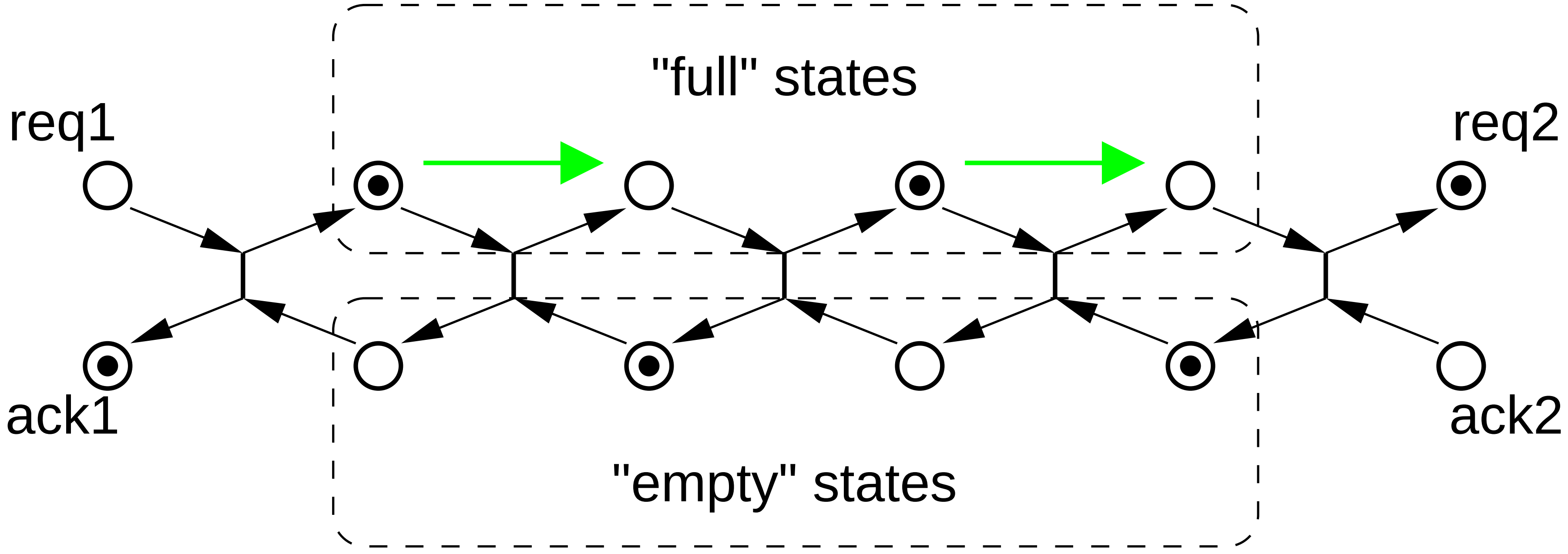
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)



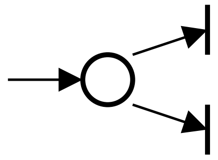
Pipeline model

- What is a pipeline (First In – First Out, FIFO)
- Difference between a pipeline and a shift register
- Behavioural model of a linear pipeline (Petri Net)

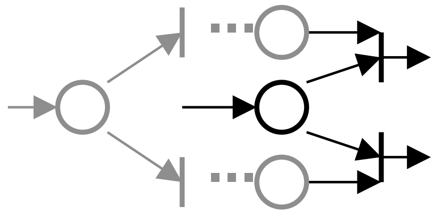


PN building blocks: choice

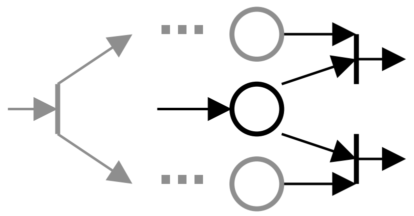
- Free choice
  - usually represents the lack of knowledge about its mechanism;
  - often takes place in the environment.



- Controlled choice
  - the control mechanism is known and represented in the model;
  - can be implemented as a device.

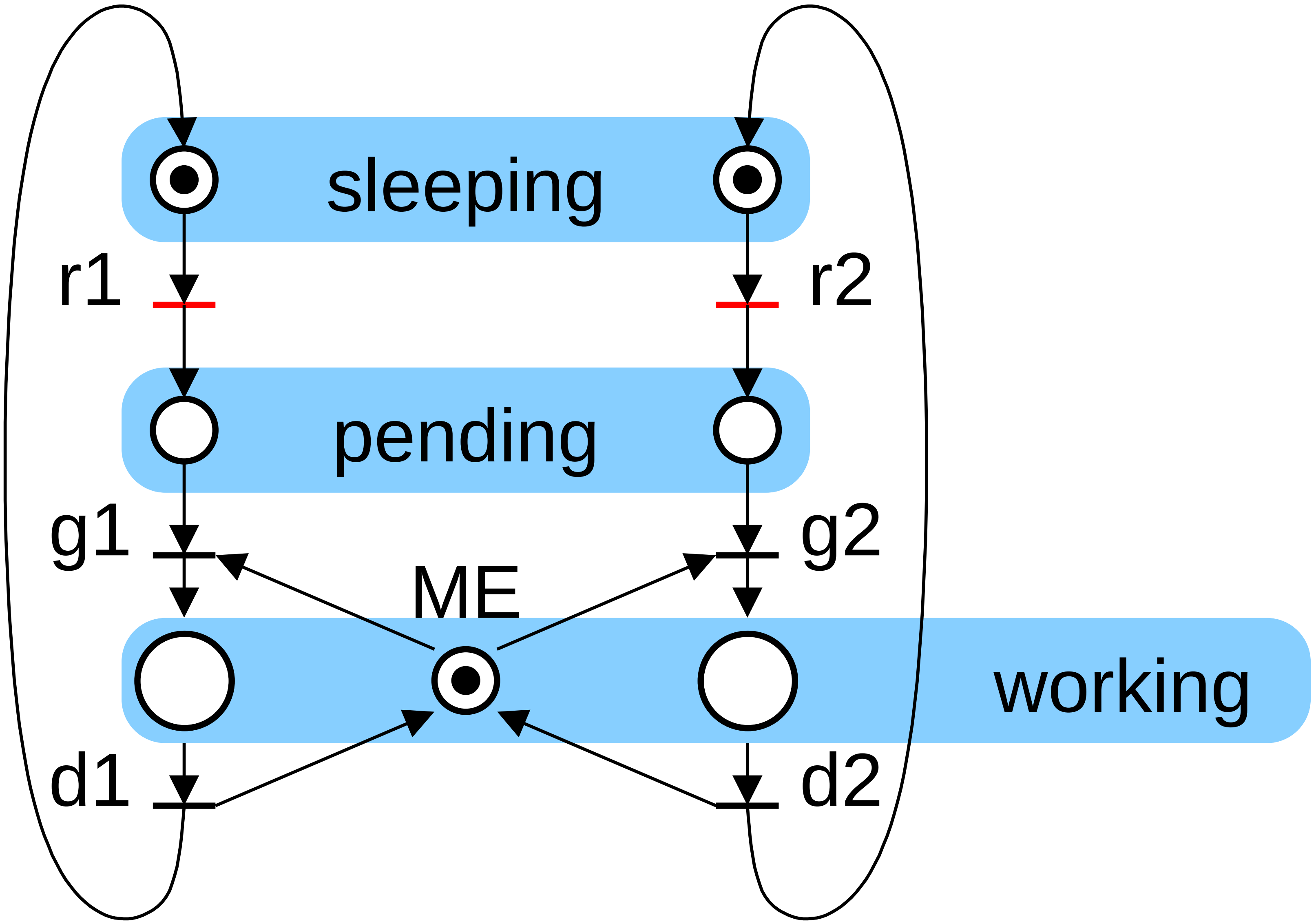


- Arbitration
  - Buridan's Ass problem [Aristotle];
  - can be implemented as a device.

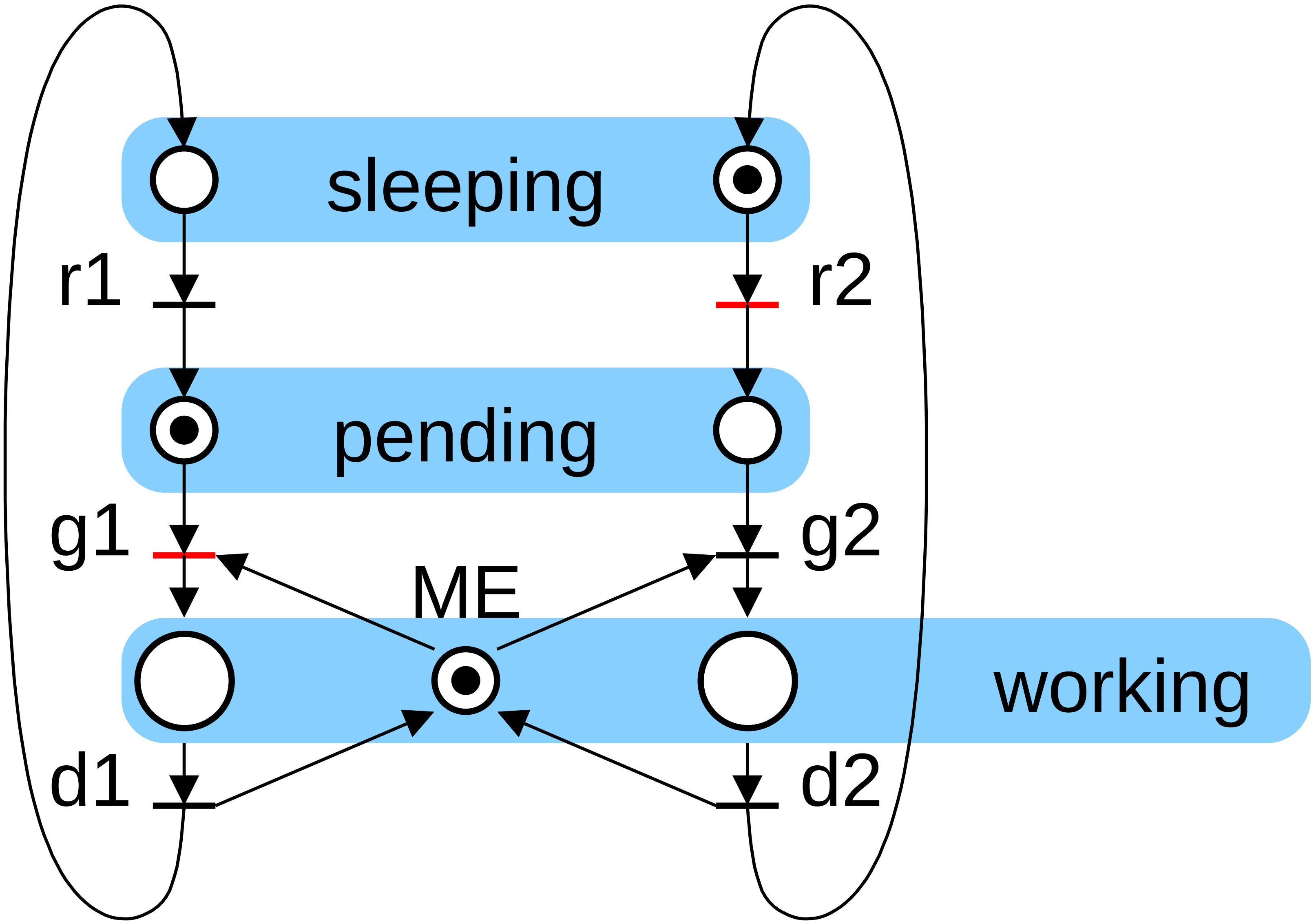


- Confusion (we are not going to use it. . .)
  - should be refined for implementation.

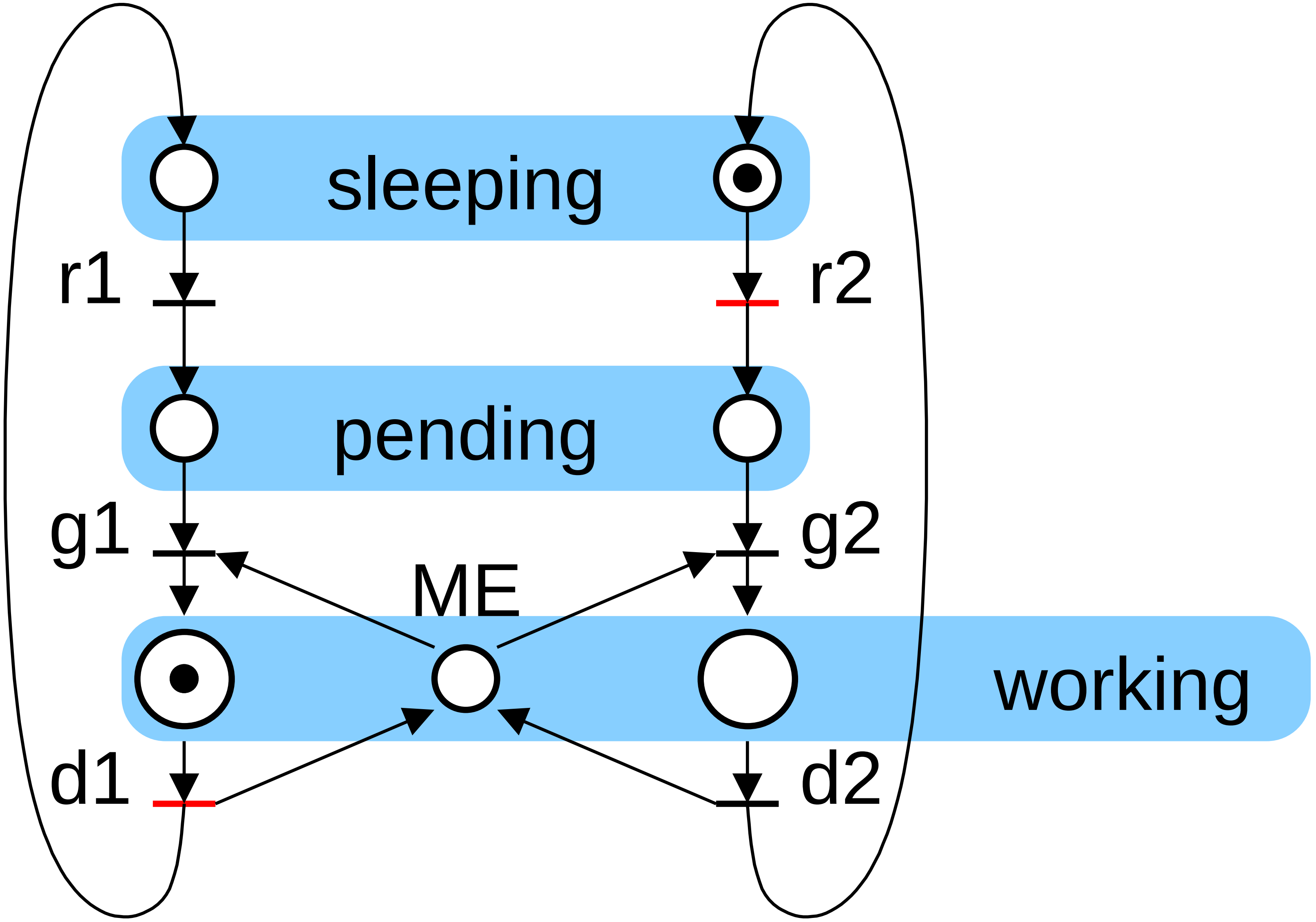
Asynchronous arbiter model



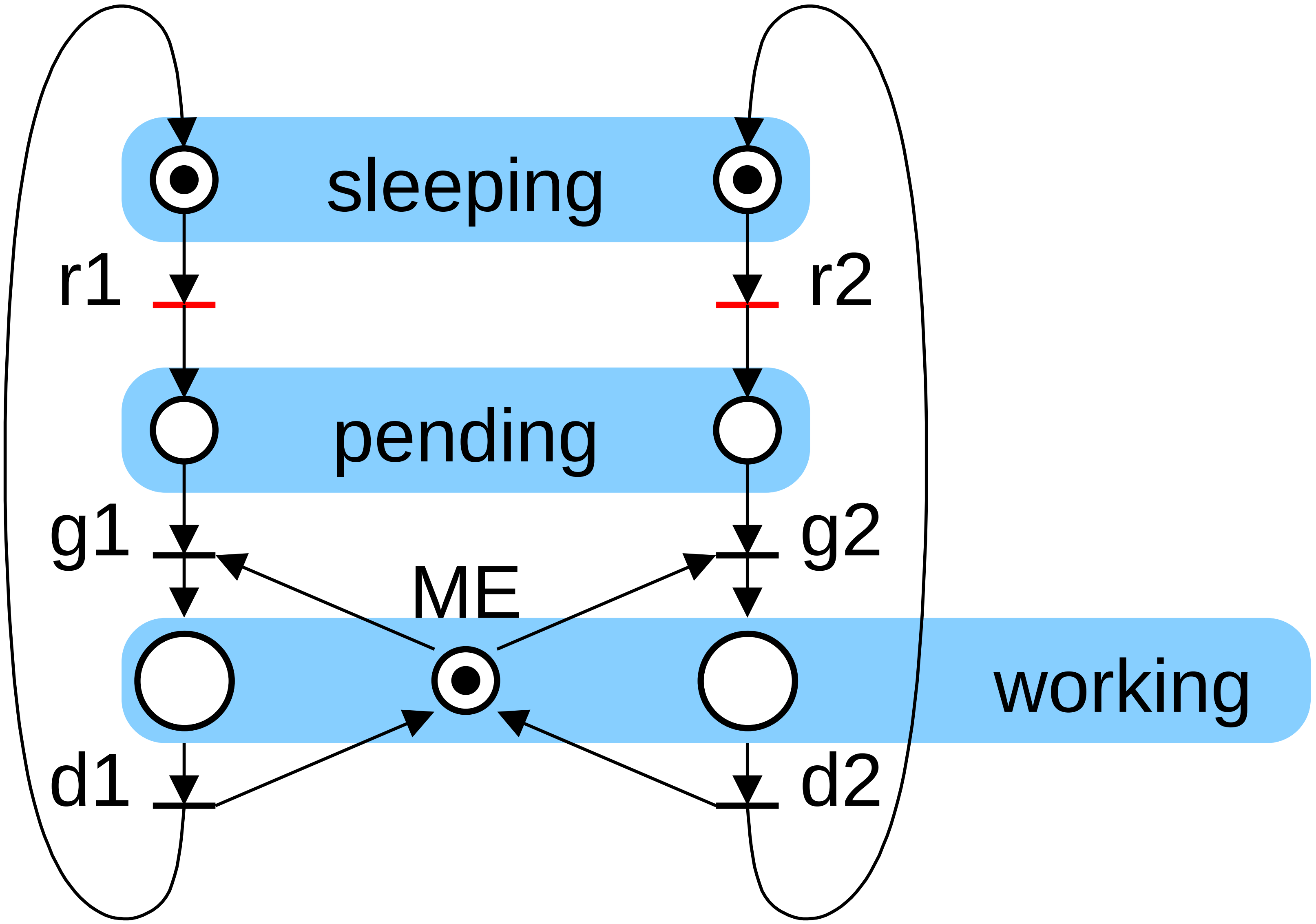
Asynchronous arbiter model



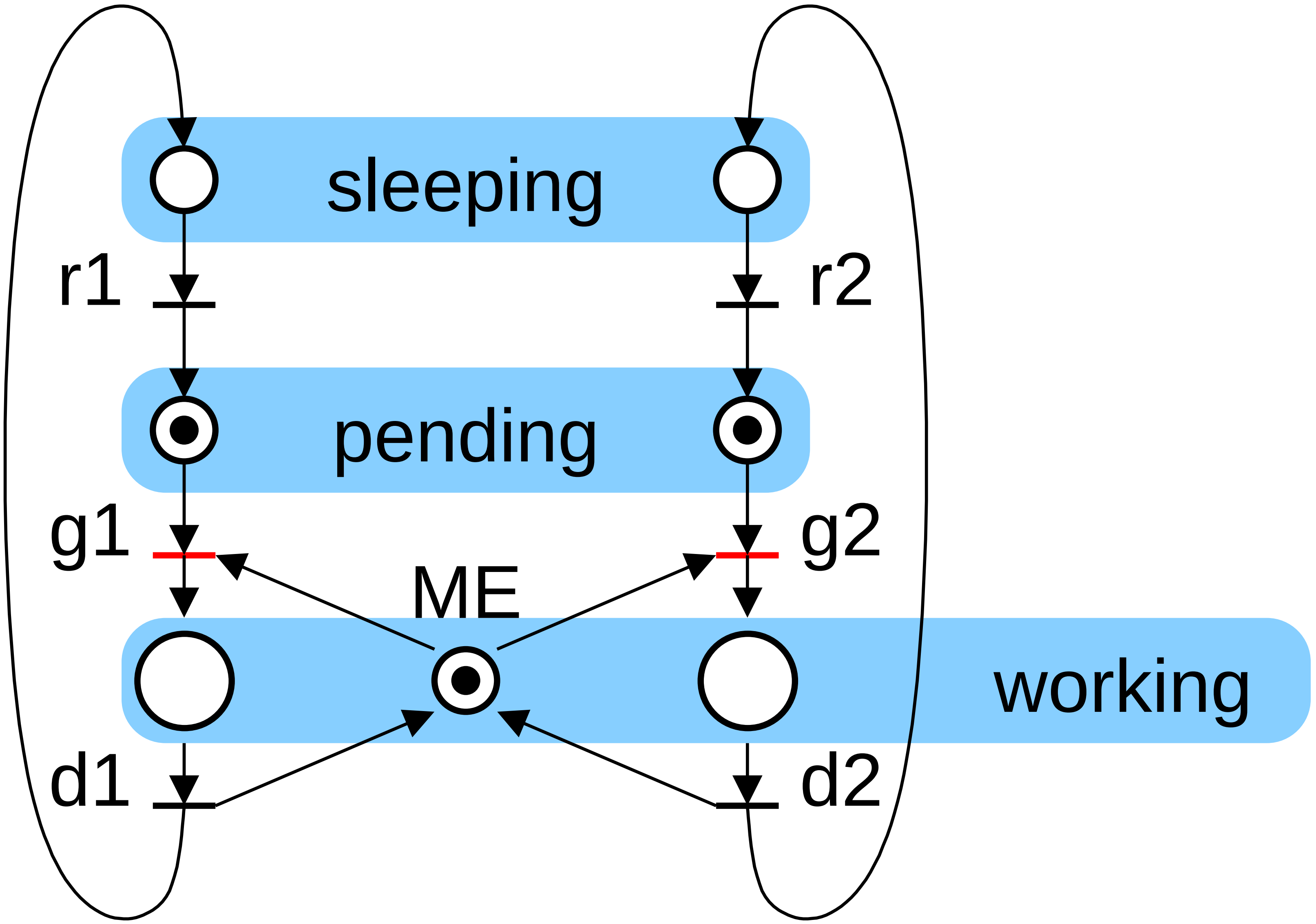
Asynchronous arbiter model



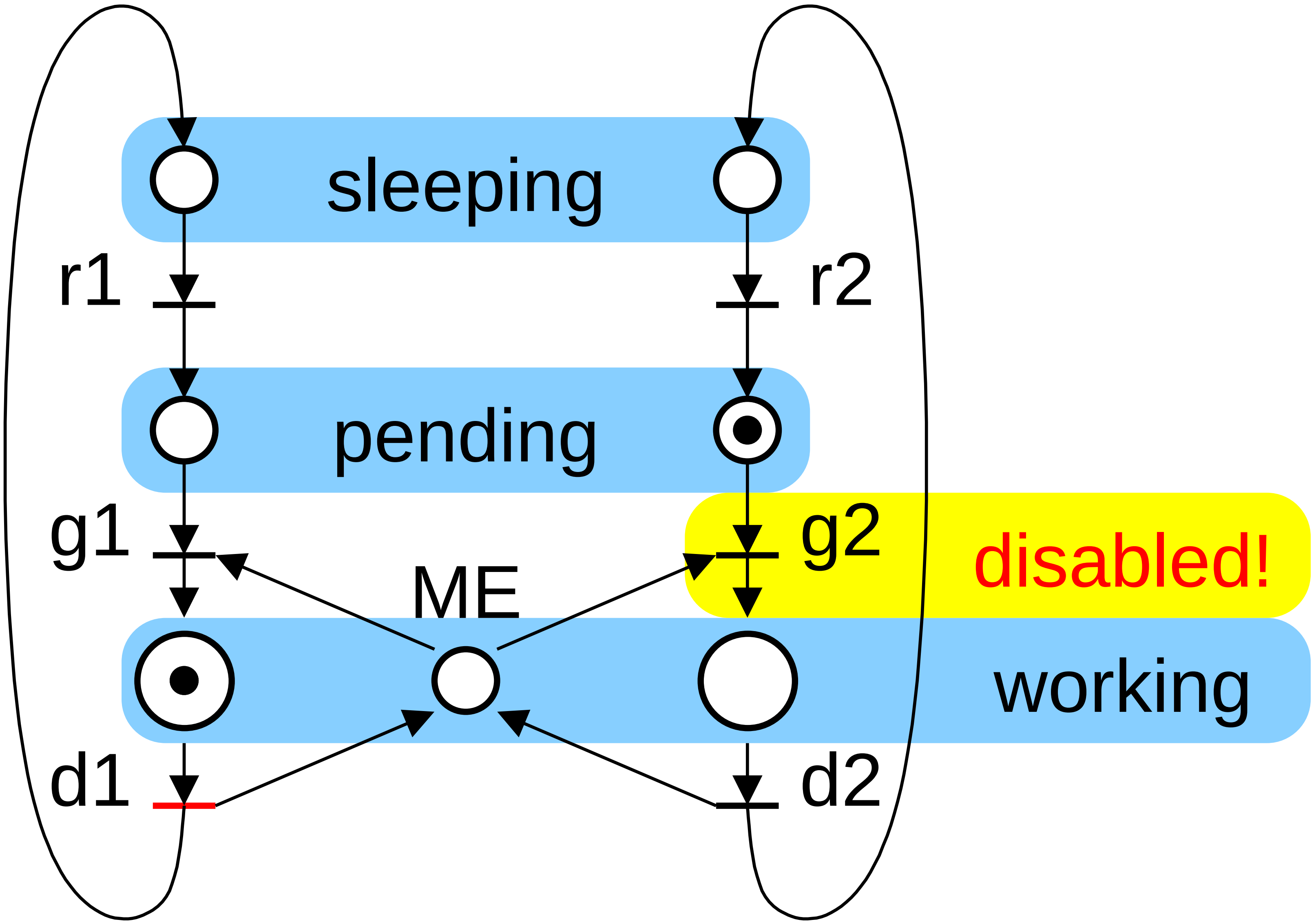
Asynchronous arbiter model



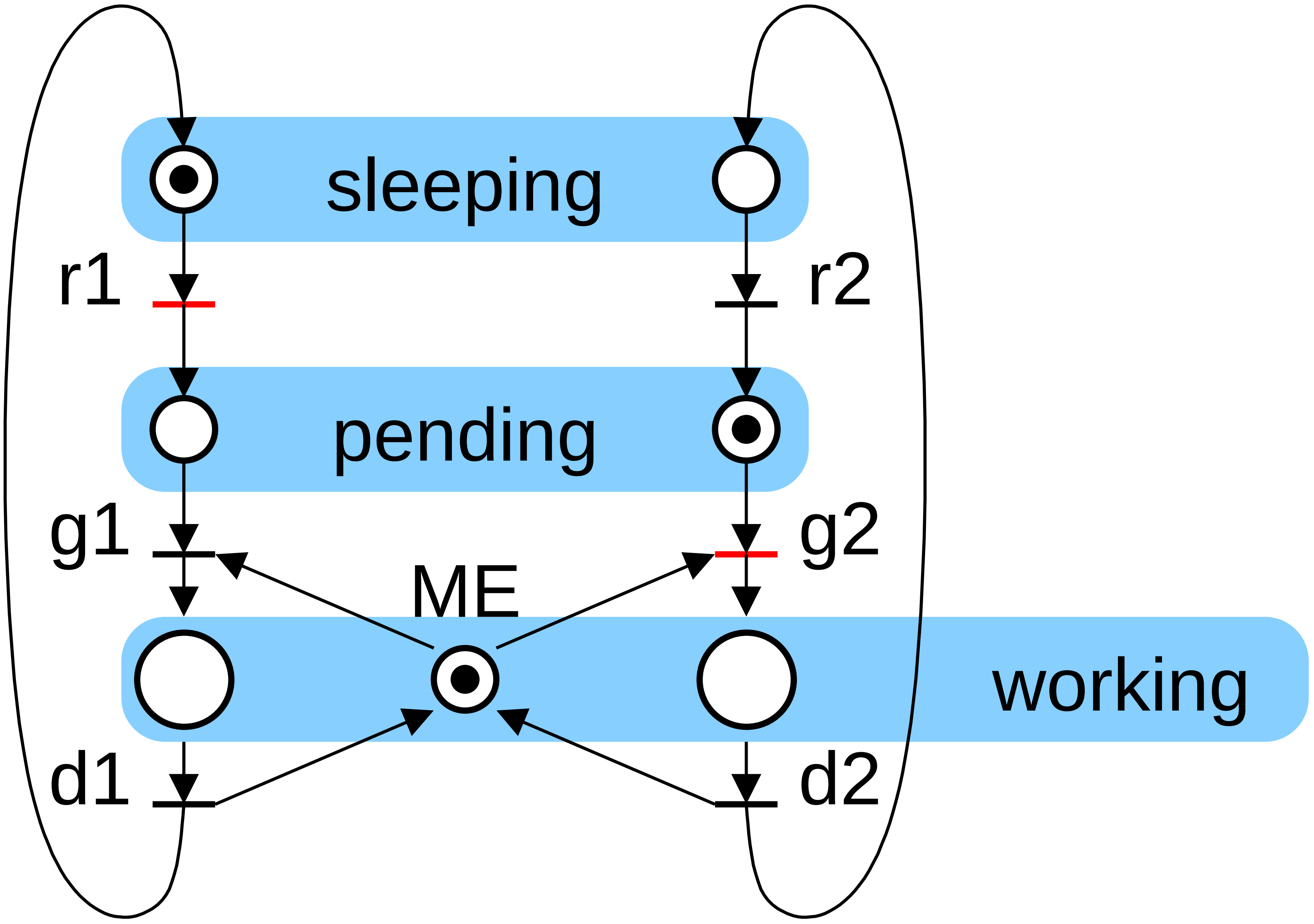
Asynchronous arbiter model



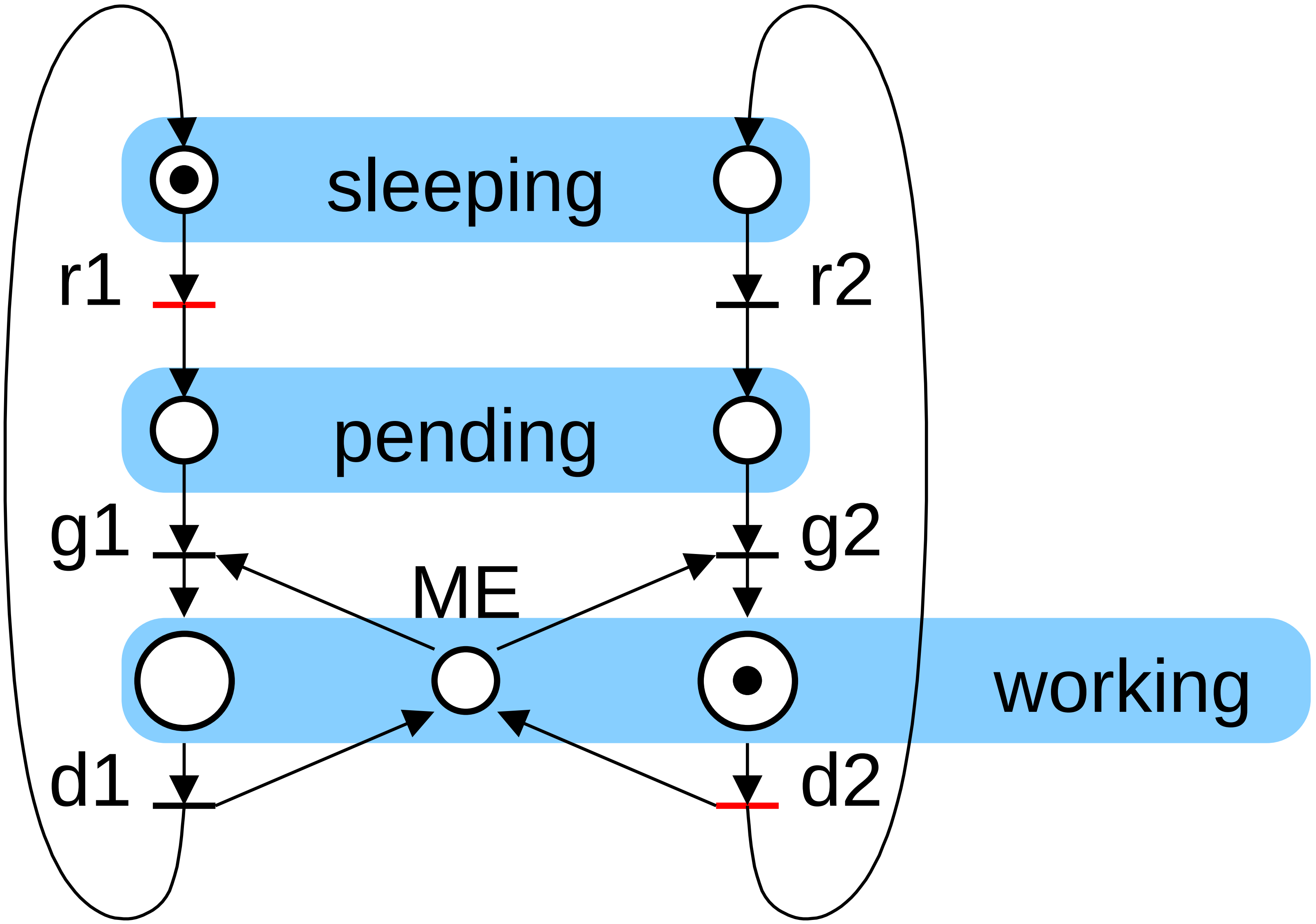
Asynchronous arbiter model



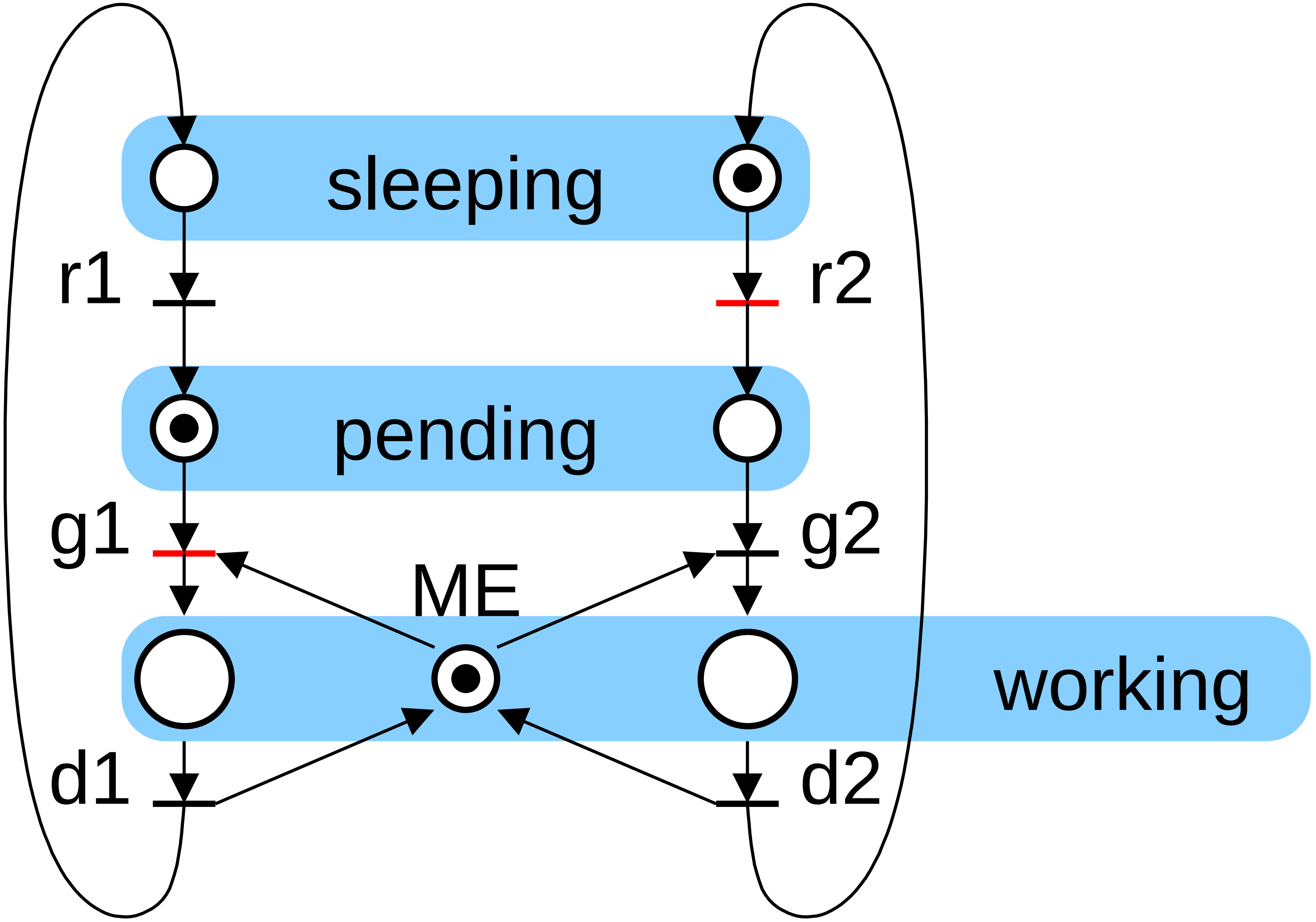
Asynchronous arbiter model



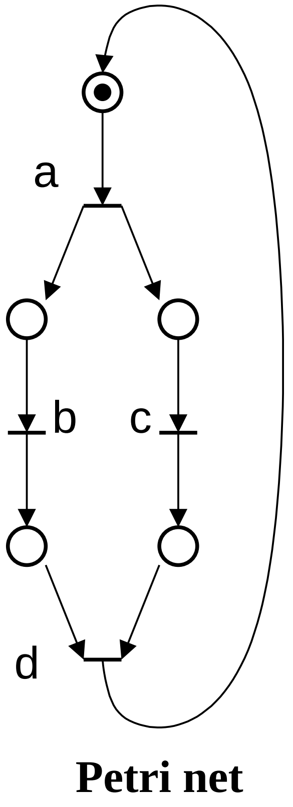
Asynchronous arbiter model

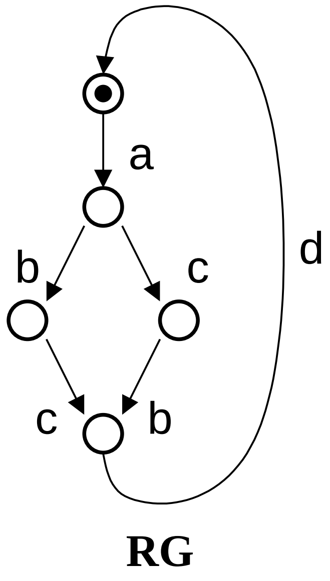


Asynchronous arbiter model



Reachability Graphs (fork-join)

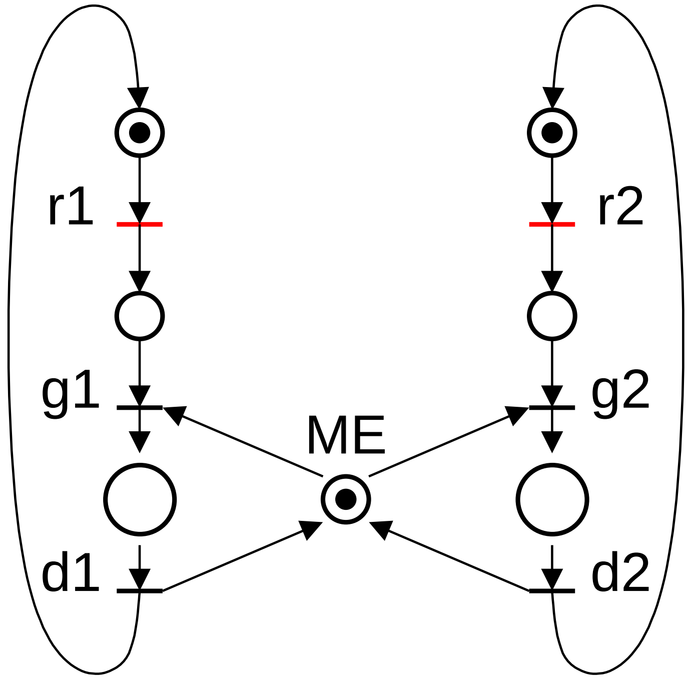




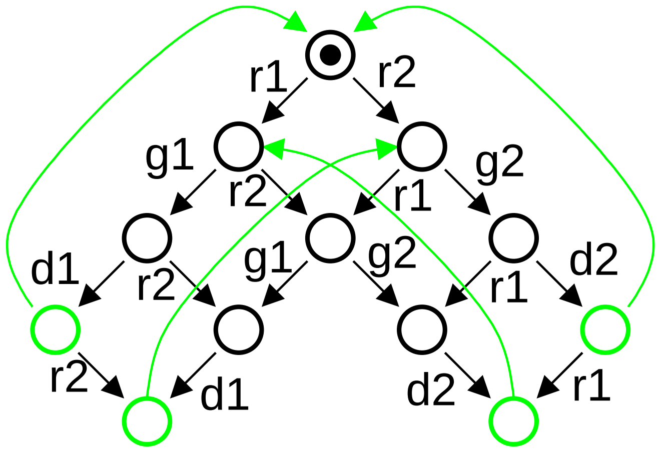
- Concurrency creates “diamonds” in RGs
- The diamonds are semimodular, distributive lattices

read Foundations of Maths→Set Theory→Lattice Theory

Reachability Graphs and hazards



Arbiter Petri Net



Reachability Graph

"Green" states are the instances of previously encountered states

- Remember the dead donkey!
- The diamond net is “split”
  - Non-semimodular, distributive lattice

The properties checked

- Safety (at most one token in a place)
- Deadlock freedom
- Liveness (any marking can be reached from the given reachable marking)
- Persistence (choice, arbitration)
  - (non-)determinism
- Early evaluation (in non-distributive RGs)
  - the output is formed before all inputs have arrived
- Reachability of some particular marking/state
- ...

Summary

- Reactivity in models (open/closed models)
- FSM models (read the book by Rob Williams)
- Concurrency modelling
  - Petri nets
  - Reachability graphs
  - verification