Discrete-event models part 2: Petri Nets

Dr. Bystrov

School of Engineering

Newcastle University

New Concept – Concurrency

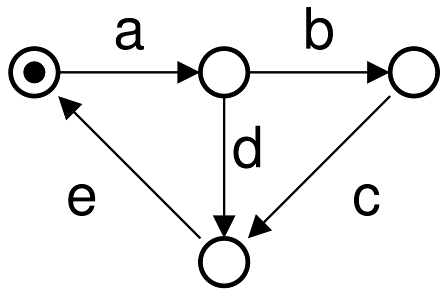
Independent processes are concurrent.

- 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 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.



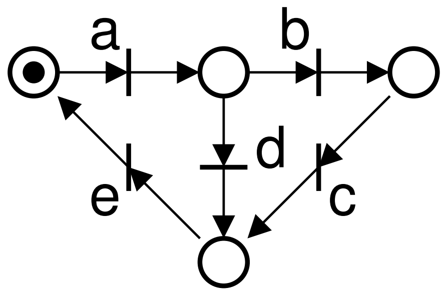
○ – states

↘ –transitions

● –init. state

a..e –labels

FSM graph



○ –places

| –transitions

● init. marking

a..e –labels

Petri net

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 $M_0 : P \rightarrow N \,|N = \{0, 1, 2, ...\}$; ⊙
- **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

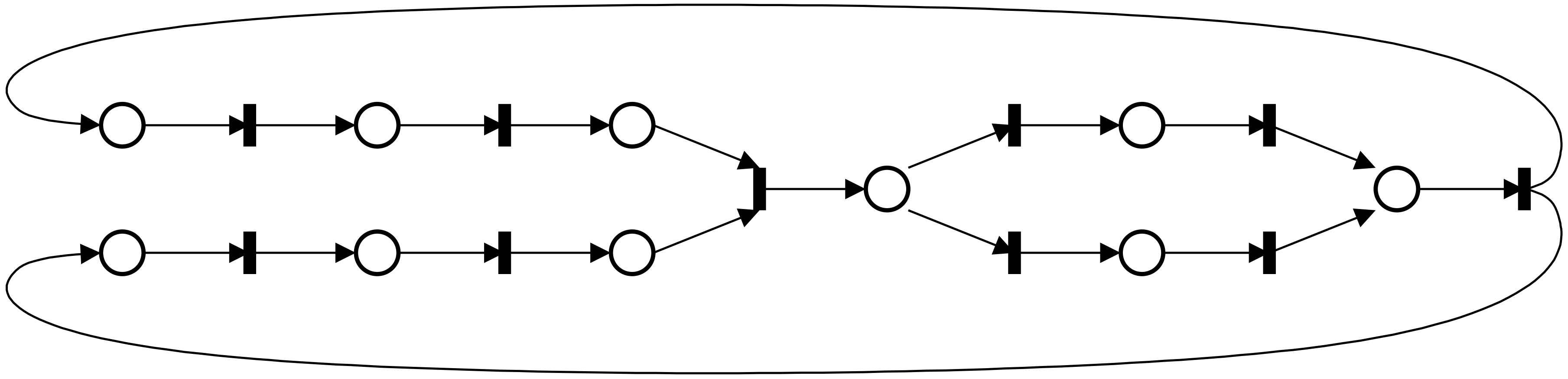
Tokens move by "firing" transitions, leading to new markings – **token game**.

We use 1-safe PNs only:

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

Playing the token game

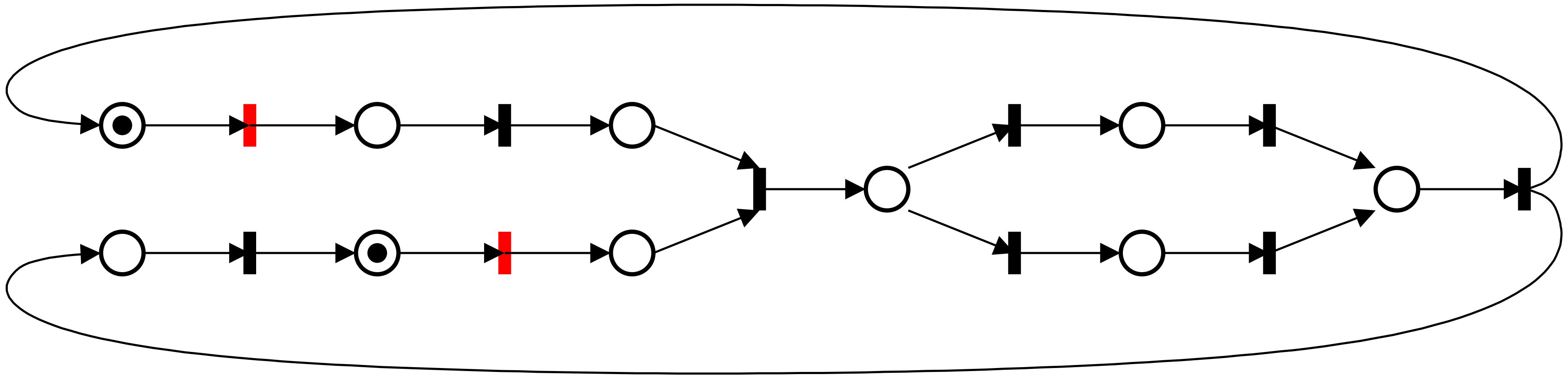
Do you play board games?



Bare bones of a PN.
Should we inject some life in it?

Playing the token game

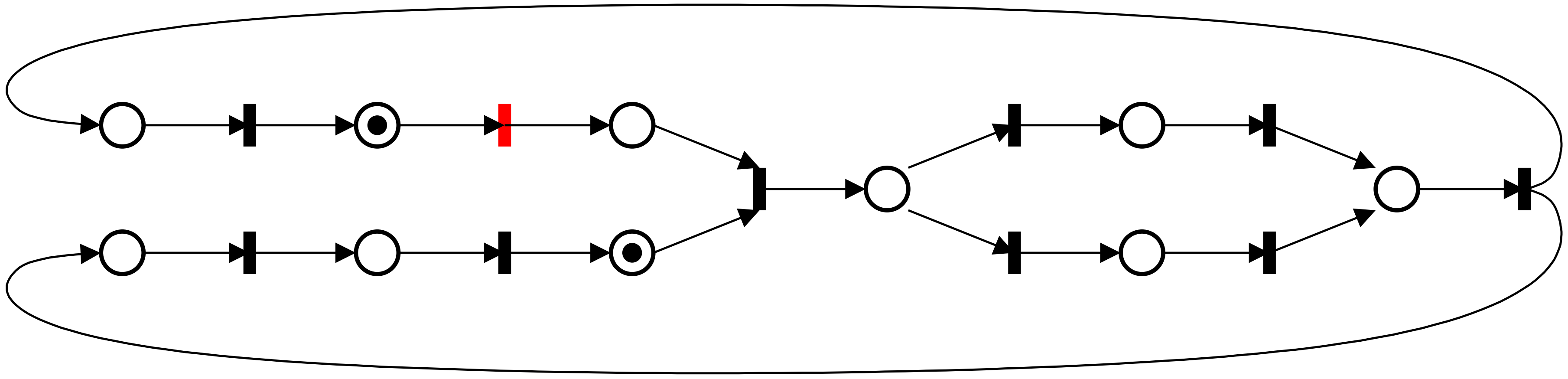
Do you play board games?



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

Playing the token game

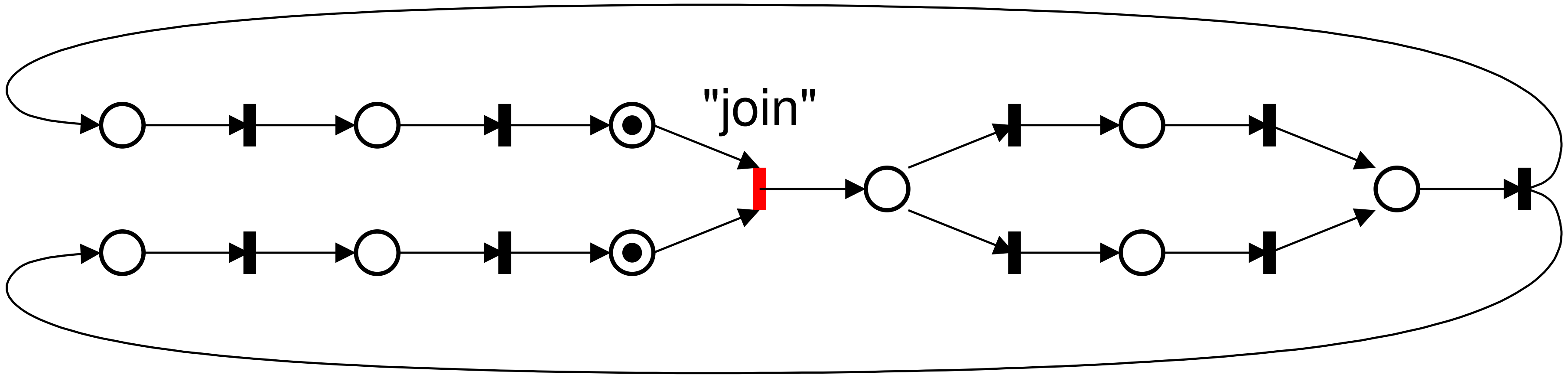
Do you play board games?



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

Playing the token game

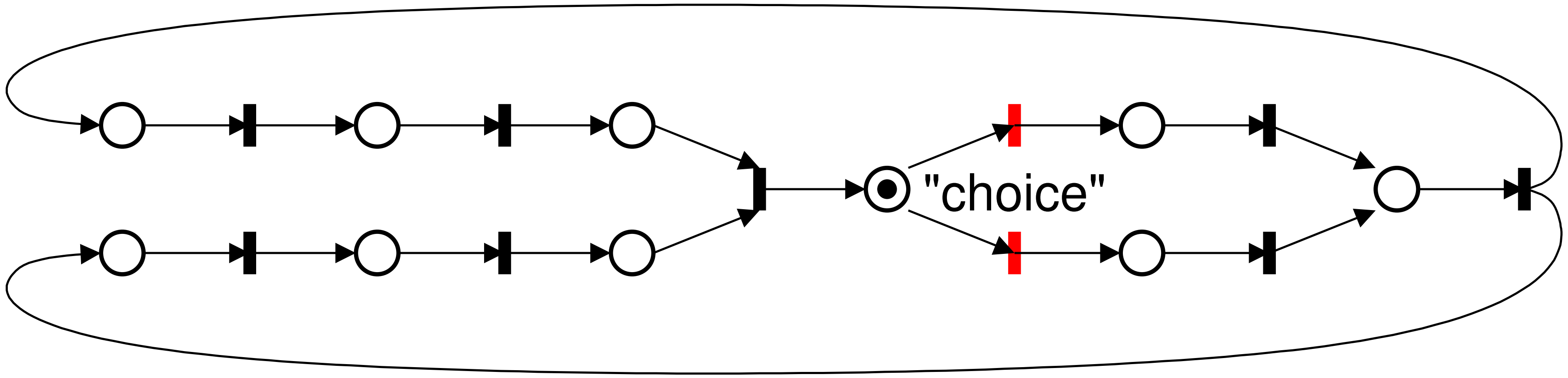
Do you play board games?



“Join” transition performs 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

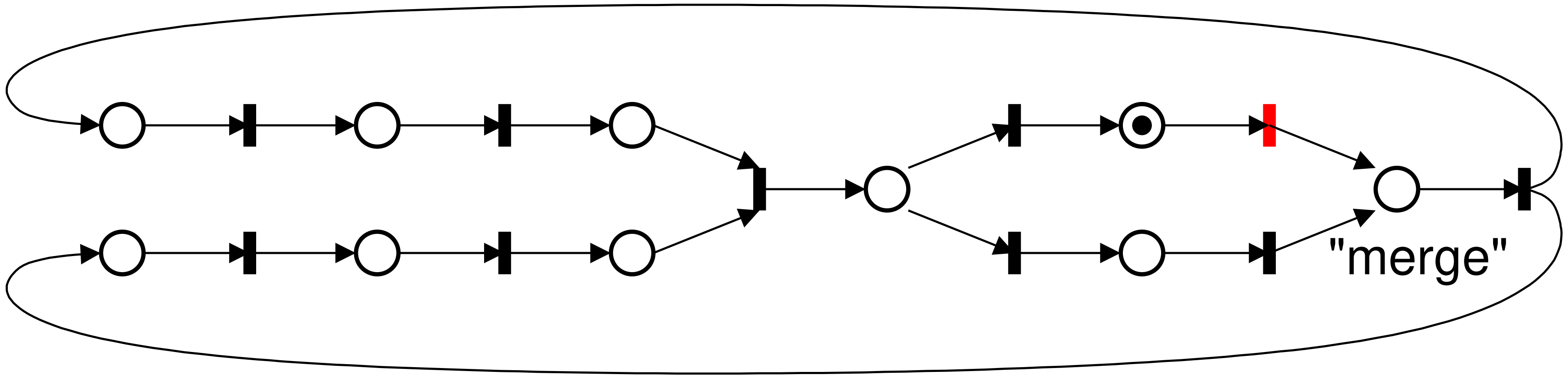
Do you play board games?



Now the atomic (unsplittable) token must decide 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

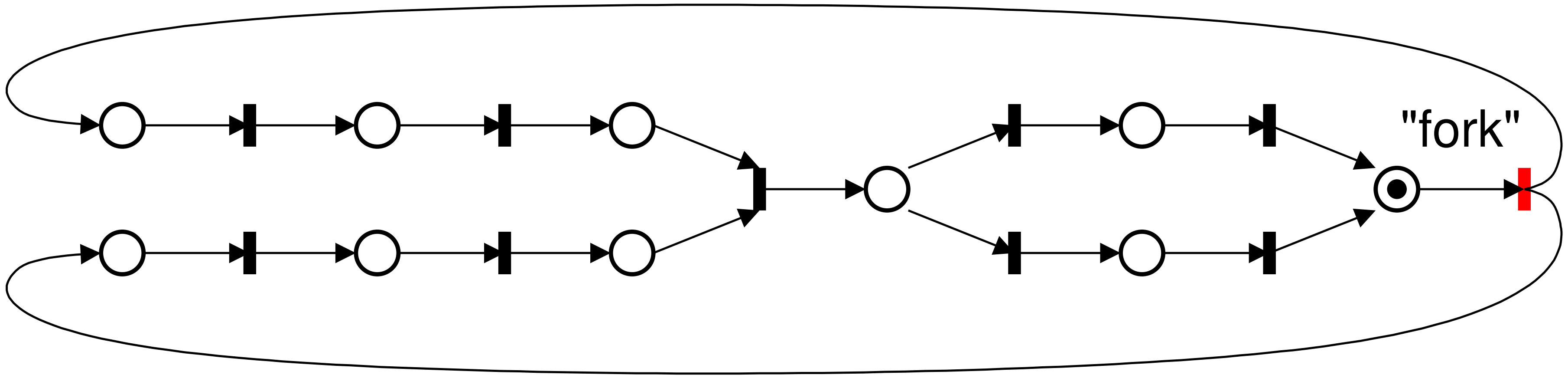
Do you play board games?



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

Playing the token game

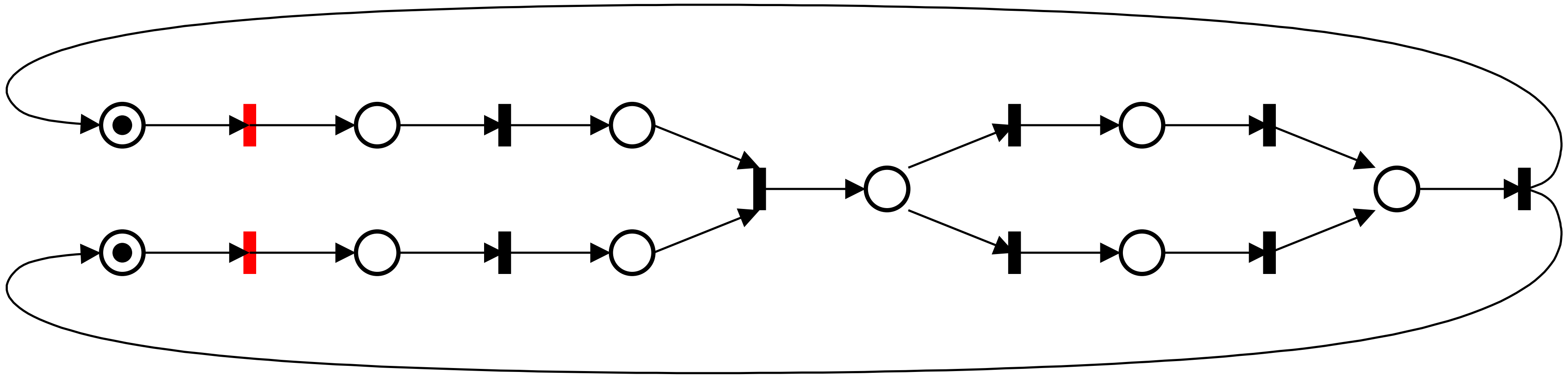
Do you play board games?



This is how we multiply tokens. . .

Playing the token game

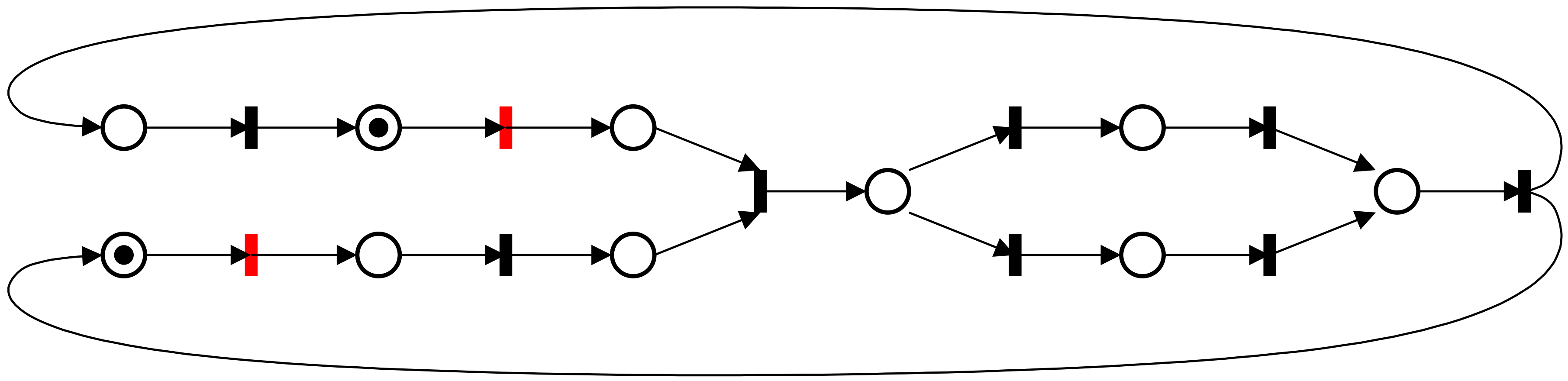
Do you play board games?



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

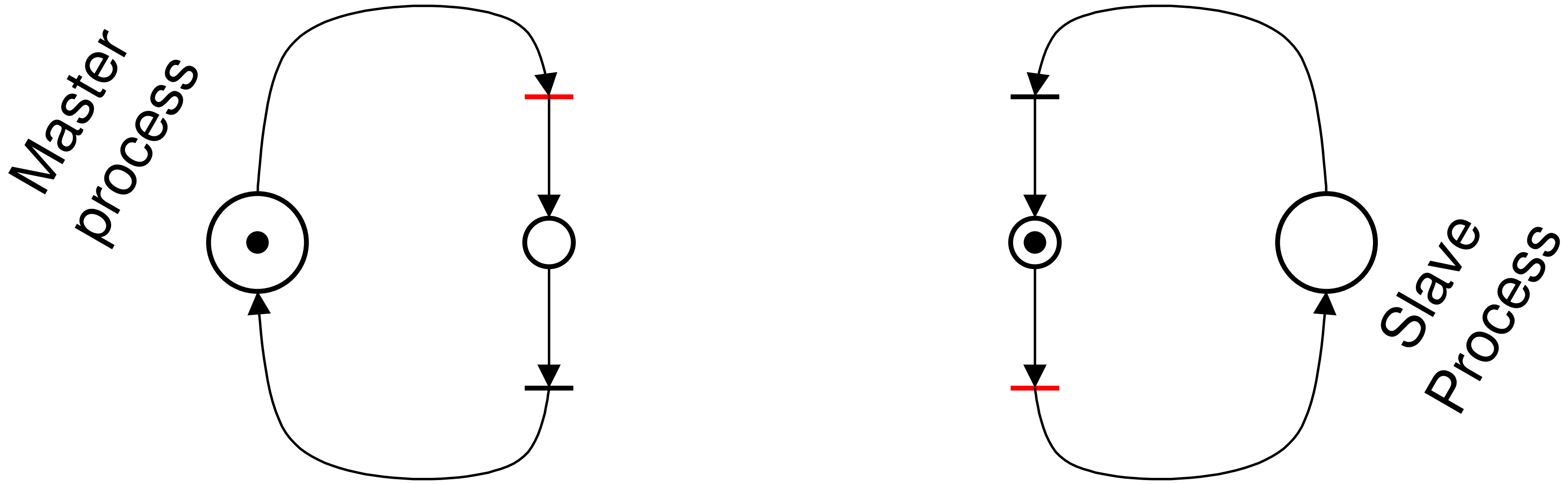
Playing the token game

Do you play board games?



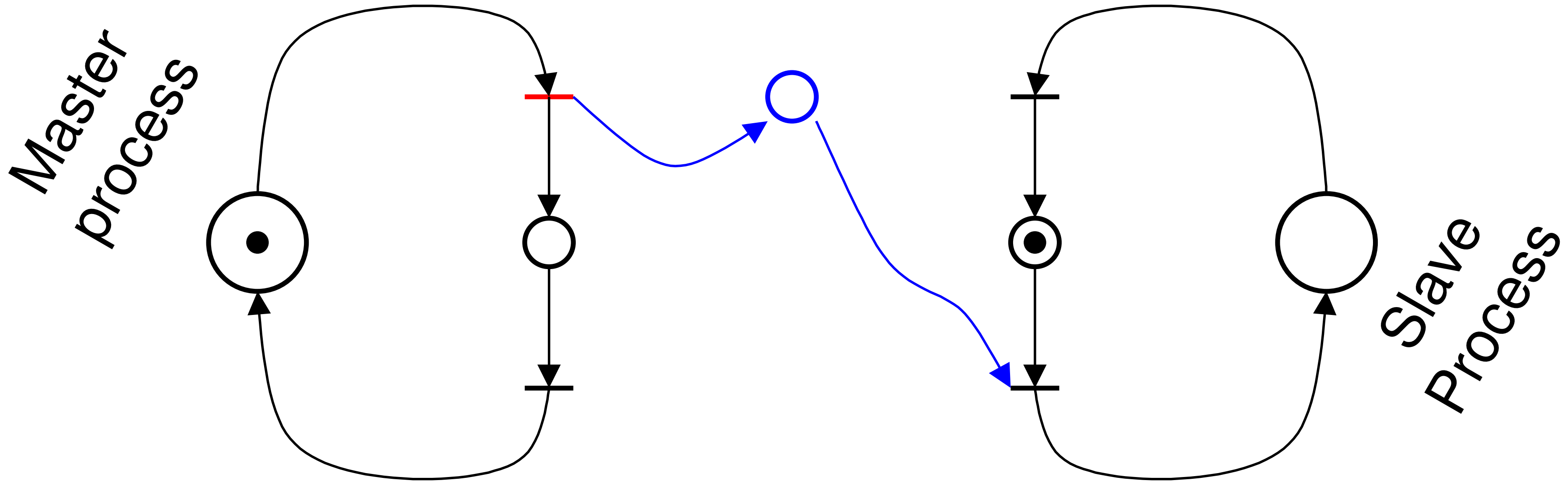
Dough!!! We've missed the initial marking!
We may arrive to it after the next cycle or after a couple of cycles or . . . never.
However, it is clearly reachable.
That is why we call it a "**reachable marking**".

Simple communication example



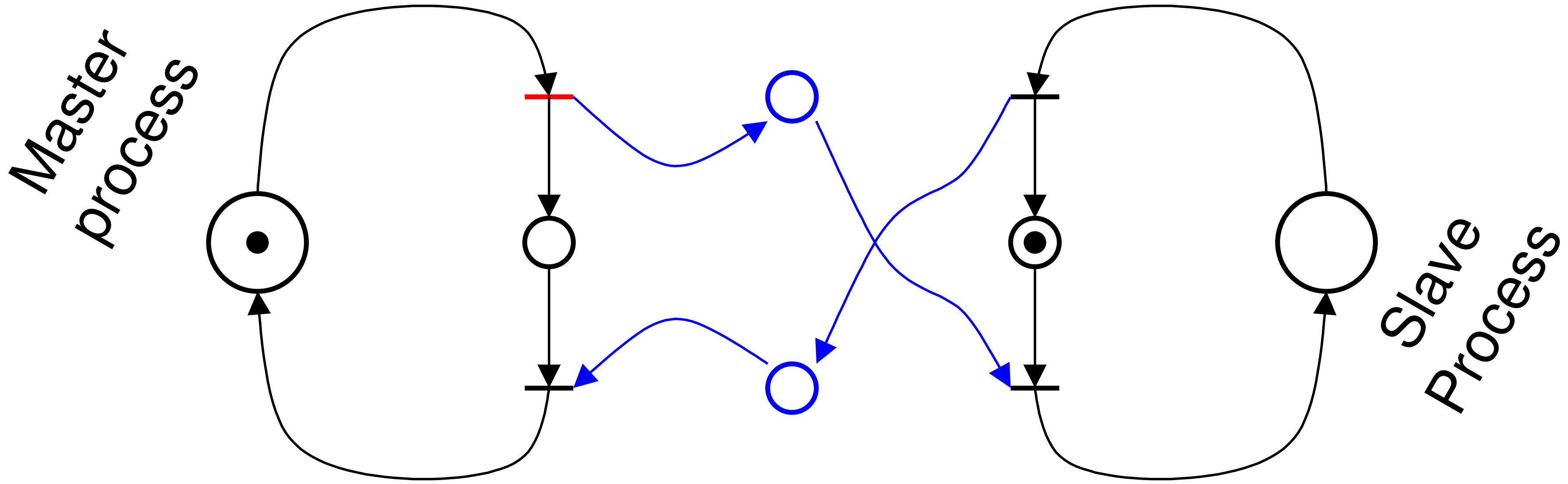
- Two processes are given.
- They are very similar and independent for now.
- How to make them play the roles of a Master and a Slave?

Simple communication example



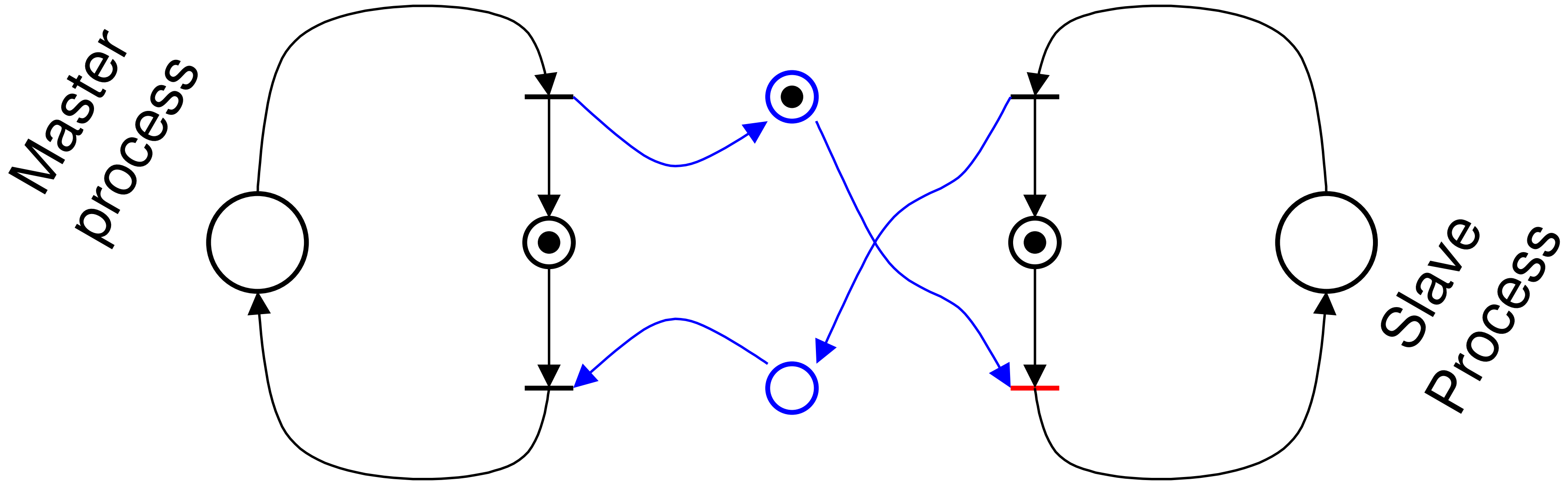
- After Master finishes (produces data), Slave starts (consumes data).
- This is the **producer-consumer** relationship.
- What is the trouble here?

Simple communication example



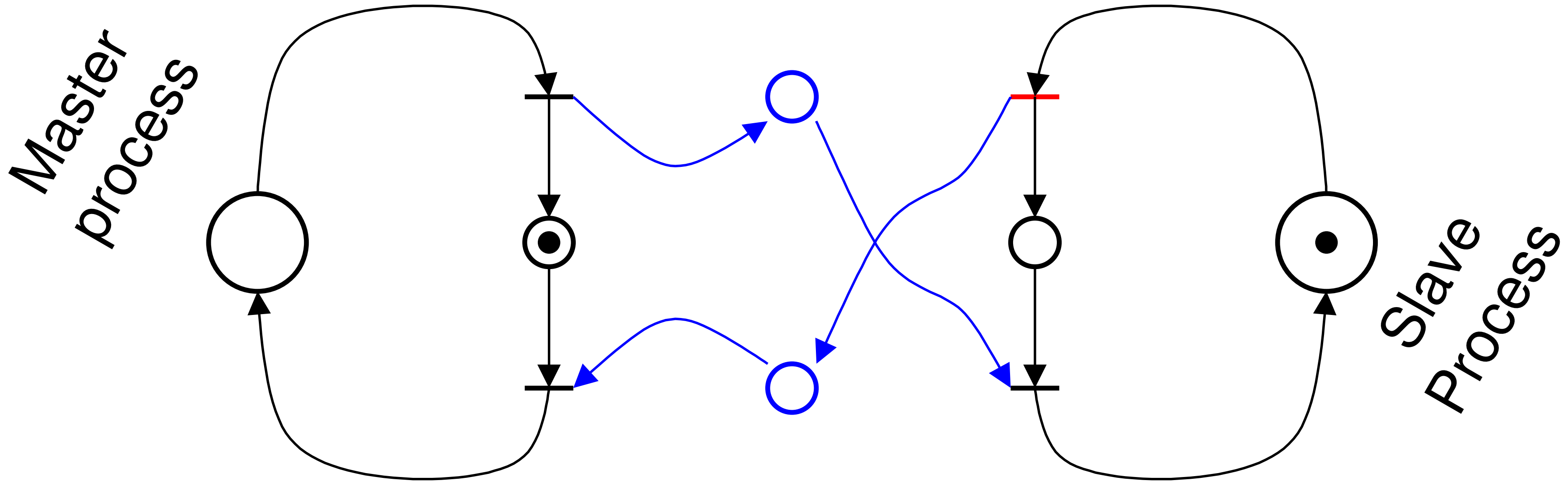
- Preventing Master from flooding Slave with tokens.
- Feedback.
- Request-acknowledgement handshake.
- Delay-insensitive interface.
- Token game begins!

Simple communication example



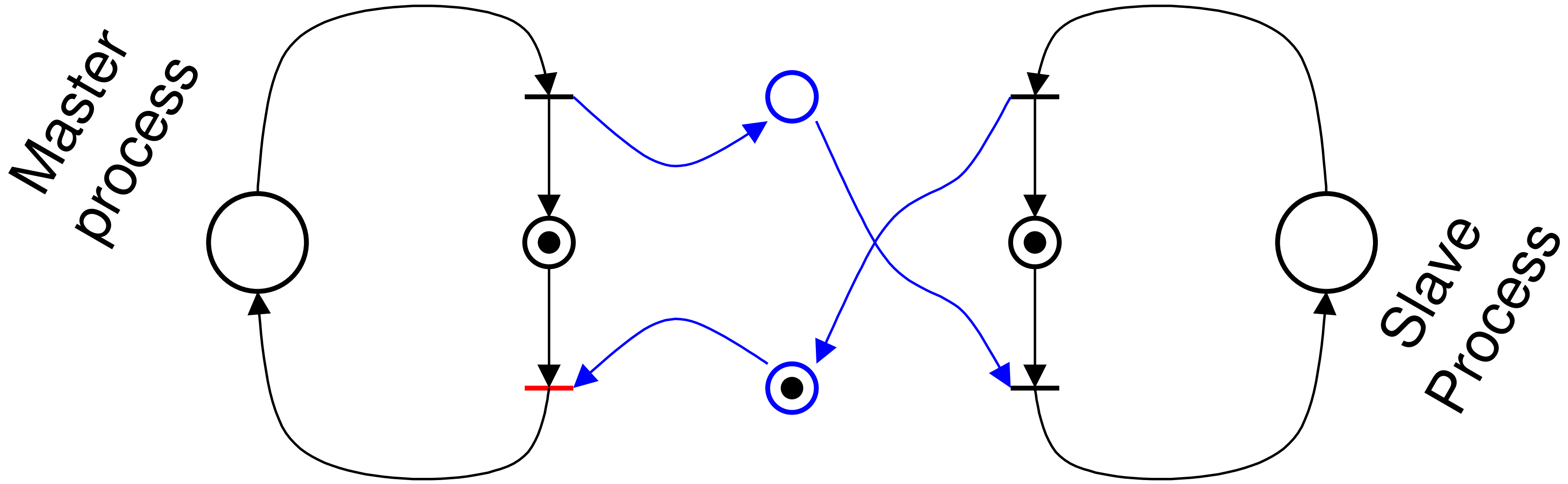
- Master process is finished.
- Request is issued by Master.
- Request input of Slave is enabled and about to fire.

Simple communication example



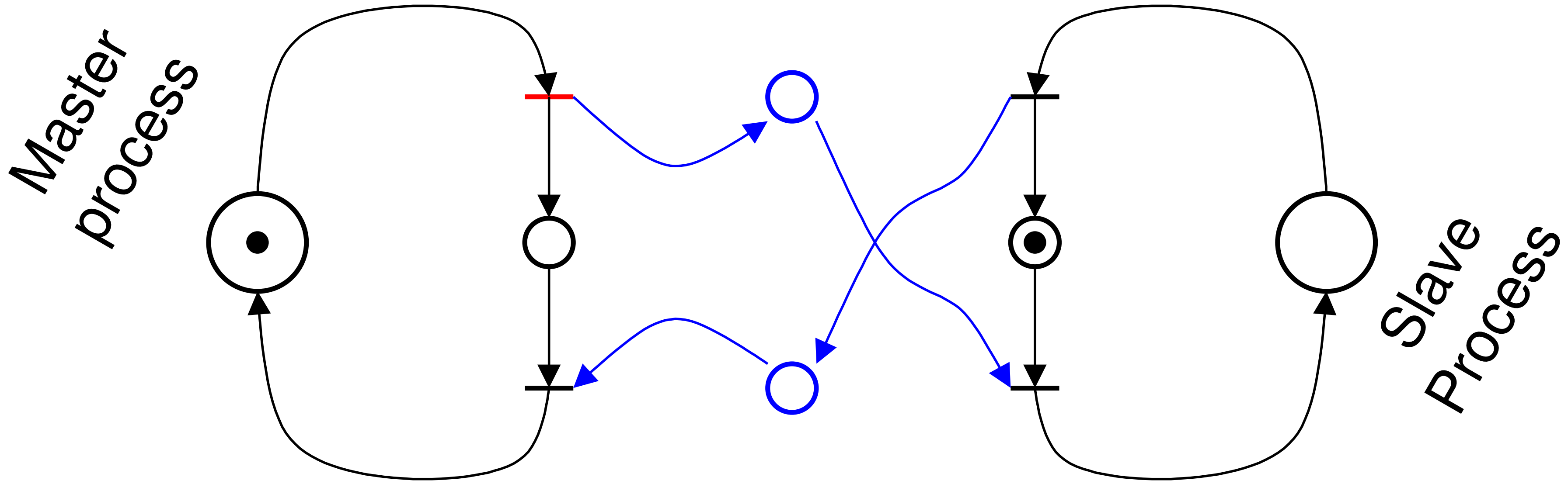
- Slave request has fired.
- Slave process has started.
- Slave acknowledgement is enabled, it will fire after Slave process finishes.

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.

Summary

- Concurrency concept
- Petri net definition, 1-safe PN model
- Examples
- Handshake

Next: Building blocks for Petri net modelling