# Verification of distributed systems and use of knowledge for implementing them[1]

**Susanne Graf** and Sophie Quinton

VERIMAG and INRIA, Grenoble

FSFMA in Singapore — May 13, 2014

[1] based on a paper published in iFM 2013

# Motivation: Proving Distributed Systems Correct

Verification of Distributed Systems is a difficult task due to the induced non-determinism

- Error prone: race conditions, synchronisation errors, ...
- State explosion

# Motivation: Proving Distributed Systems Correct

Verification of Distributed Systems is a difficult task due to the induced non-determinism

- Error prone: race conditions, synchronisation errors, ...
- State explosion

To make verification scalable, a systematic approach is needed (avoiding state explosion)

- Use a (more deterministic) centralized specification / program as a basis of verification
- Use a systematic approach to distributed implementation and prove the distribution algorithm correct

# Motivation: Proving Distributed Systems Correct

Verification of Distributed Systems is a difficult task due to the induced non-determinism

- Error prone: race conditions, synchronisation errors, ...
- State explosion

To make verification scalable, a systematic approach is needed (avoiding state explosion)

- Use a (more deterministic) centralized specification / program as a basis of verification
- Use a systematic approach to distributed implementation and prove the distribution algorithm correct

Today I will talk about the second part

# Distributed Control and Implementation

Problem to be solved:

*"Given a centralized specification PN and a global constraint $\Psi$,*
*Derive a distributed implementation I for PN controlled by $\Psi$"*

# Distributed Control and Implementation

Problem to be solved:

> *"Given a centralized specification PN and a global constraint $\Psi$,*
> *Derive a distributed implementation I for PN controlled by $\Psi$"*

Our hypotheses:

- Centralized specification *PN*: w.l.g. Petri Nets

# Distributed Control and Implementation

Problem to be solved:

> *"Given a centralized specification PN and a global constraint $\Psi$,*
> *Derive a distributed implementation I for PN controlled by $\Psi$"*

Our hypotheses:

- Centralized specification *PN*: w.l.g. Petri Nets
- Distributed setting: one process per location — can learn about each other only via communication mechanisms provided by the platform

# Distributed Control and Implementation

Problem to be solved:

> *"Given a centralized specification PN and a global constraint $\Psi$,*
> *Derive a distributed implementation I for PN controlled by $\Psi$"*

Our hypotheses:

- Centralized specification *PN*: w.l.g. Petri Nets
- Distributed setting: one process per location — can learn about each other only via communication mechanisms provided by the platform
- Constraint $\Psi$: a safety constraint (here: priorities)

# Distributed Control and Implementation

Problem to be solved:

> *"Given a centralized specification PN and a global constraint $\Psi$,*
> *Derive a distributed implementation I for PN controlled by $\Psi$"*

Our hypotheses:

- Centralized specification *PN*: w.l.g. Petri Nets
- Distributed setting: one process per location — can learn about each other only via communication mechanisms provided by the platform
- Constraint $\Psi$: a safety constraint (here: priorities)

Not considered in this talk:

- uncontrollable actions, data, *timing*, progress constraints, ...

# Our approach to distributed implementation

Knowledge-based presentation for combining control **and** distribution:

1. Use knowledge to realize and optimize a transformation [RR07,BBPS09,GPQ10]:

$$PN \ + \ \Psi \ \longrightarrow \ PN' \text{ guaranteeing } \Psi$$

# Our approach to distributed implementation

Knowledge-based presentation for combining control **and** distribution:

1. Use knowledge to realize and optimize a transformation [RR07,BBPS09,GPQ10]:

$$PN \ + \ \Psi \ \longrightarrow \ PN' \text{ guaranteeing } \Psi$$

2. Derive a distributed implementation $I$ for a $PN$ by means of a protocol $Pr$ [PCT04,BGQ11]:

$$PN' \ \oplus \ Pr \ \longrightarrow \ I$$

Exist: protocols / proofs for specific settings (platform, language, implementation relation ...)

# Our approach to distributed implementation

Knowledge-based presentation for combining control **and** distribution:

1. Use knowledge to realize and optimize a transformation [RR07,BBPS09,GPQ10]:

$$PN \ + \ \Psi \ \longrightarrow \ PN' \text{ guaranteeing } \Psi$$

2. Derive a distributed implementation $I$ for a $PN$ by means of a protocol $Pr$ [PCT04,BGQ11]:

$$PN' \ \oplus \ Pr \ \longrightarrow \ I$$

   Exist: protocols / proofs for specific settings (platform, language, implementation relation ...)

Claim: A knowledge-based approach is also interesting for problem (2)

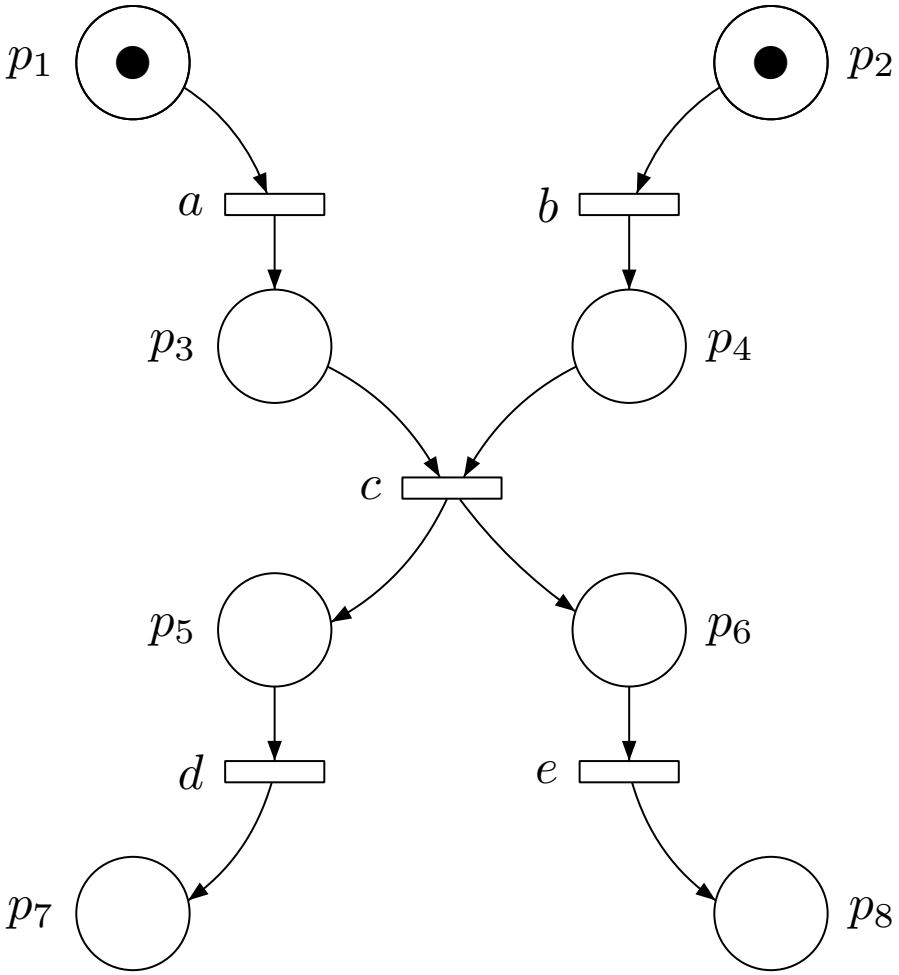
- define more efficient protocols (think in terms of knowledge)
- optimize existing protocols (exploit knowledge on framework +

# Outline

# One-safe Petri Nets

Place/Transition Nets:

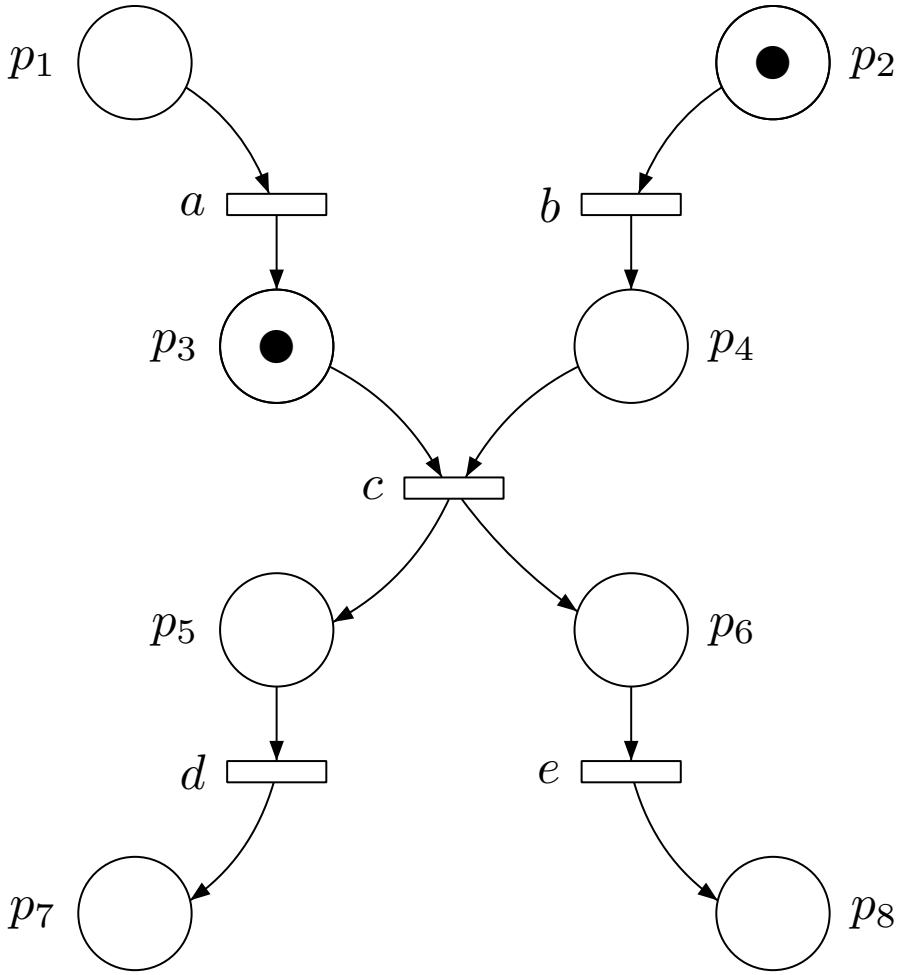
- state $s$: a set of places, e.g. $\{p_1, p_2\}$



# One-safe Petri Nets

Place/Transition Nets:

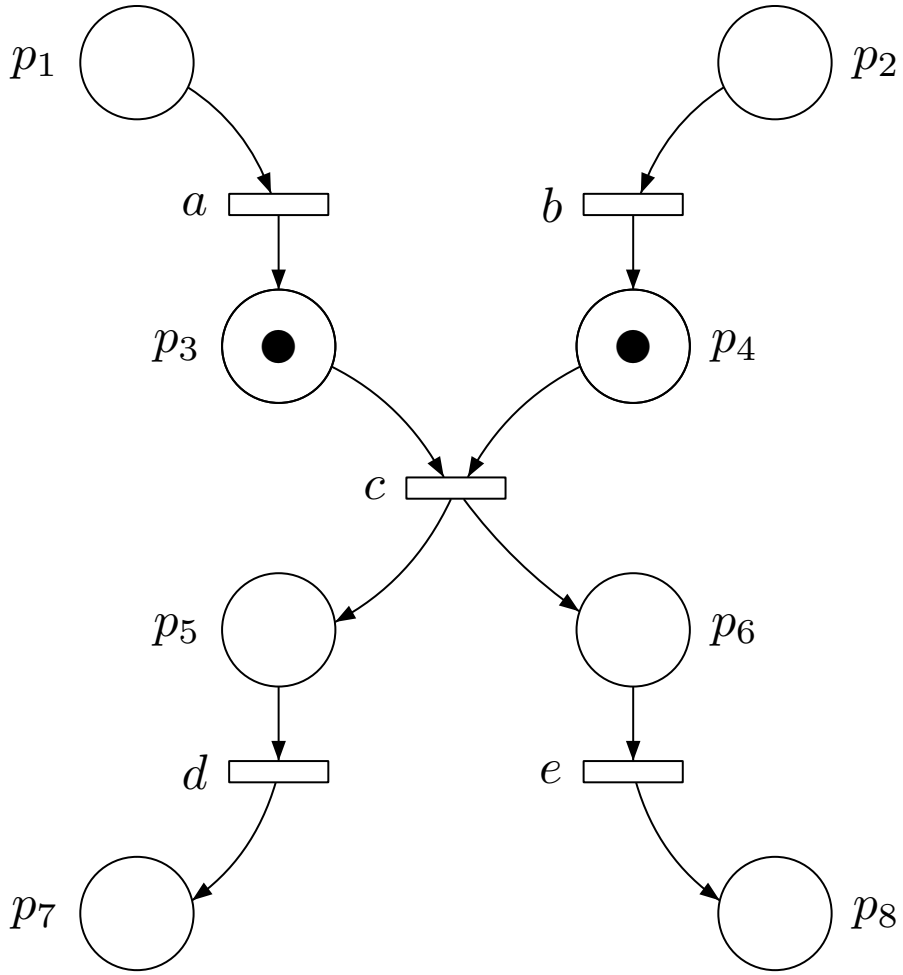
- state $s$: a set of places, e.g. $\{p_1, p_2\}$



# One-safe Petri Nets

Place/Transition Nets:

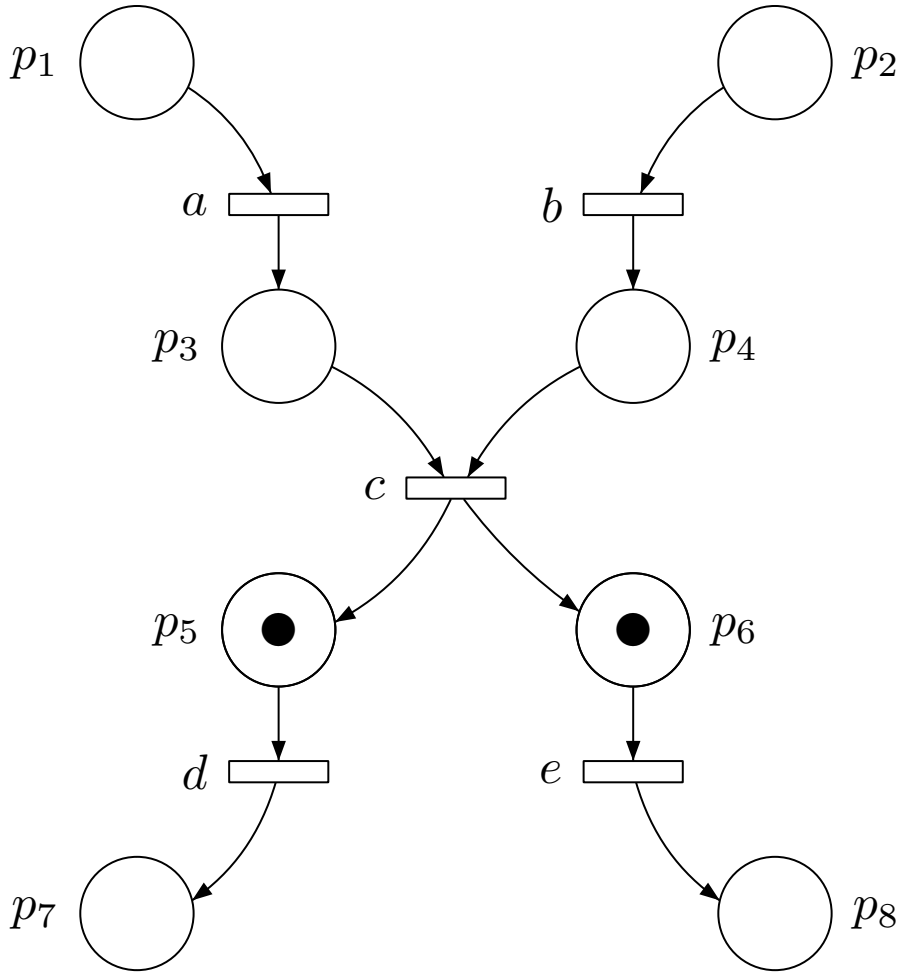
- state $s$: a set of places, e.g. $\{p_1, p_2\}$
- transition $c$ is enabled ($en_c$) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.



# One-safe Petri Nets

Place/Transition Nets:

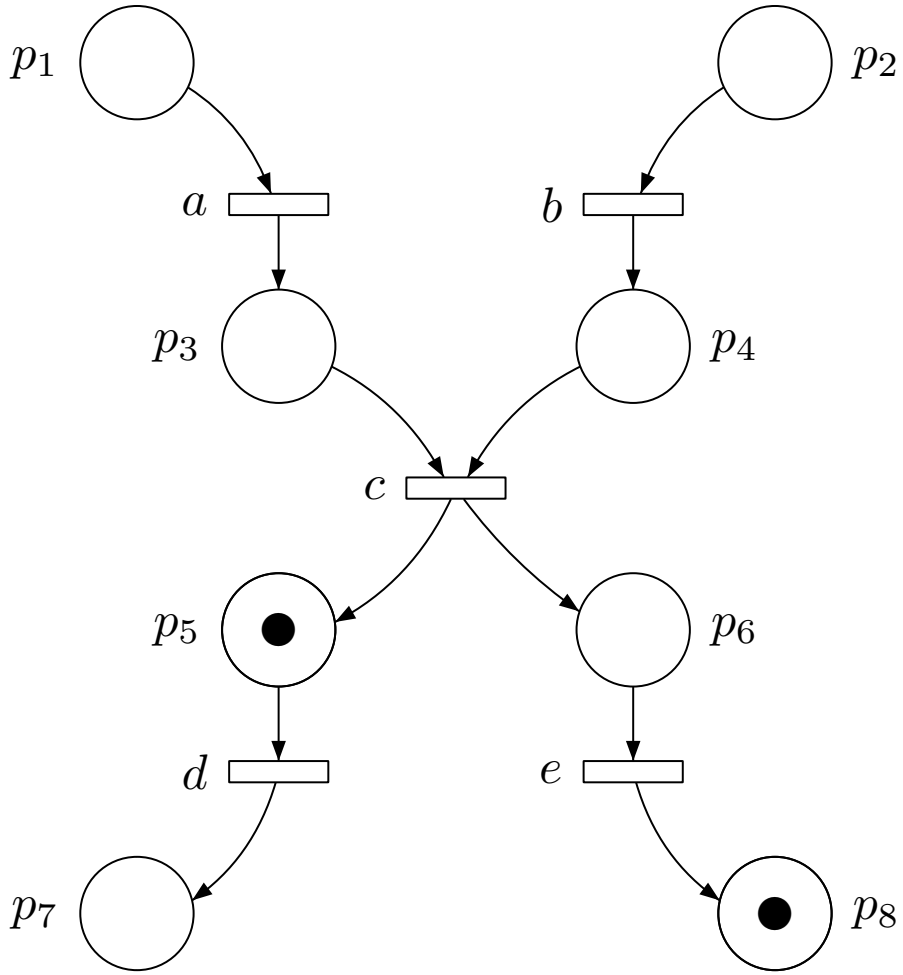
- state $s$: a set of places, e.g. $\{p_1, p_2\}$
- transition $c$ is enabled ($en_c$) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.



# One-safe Petri Nets

Place/Transition Nets:

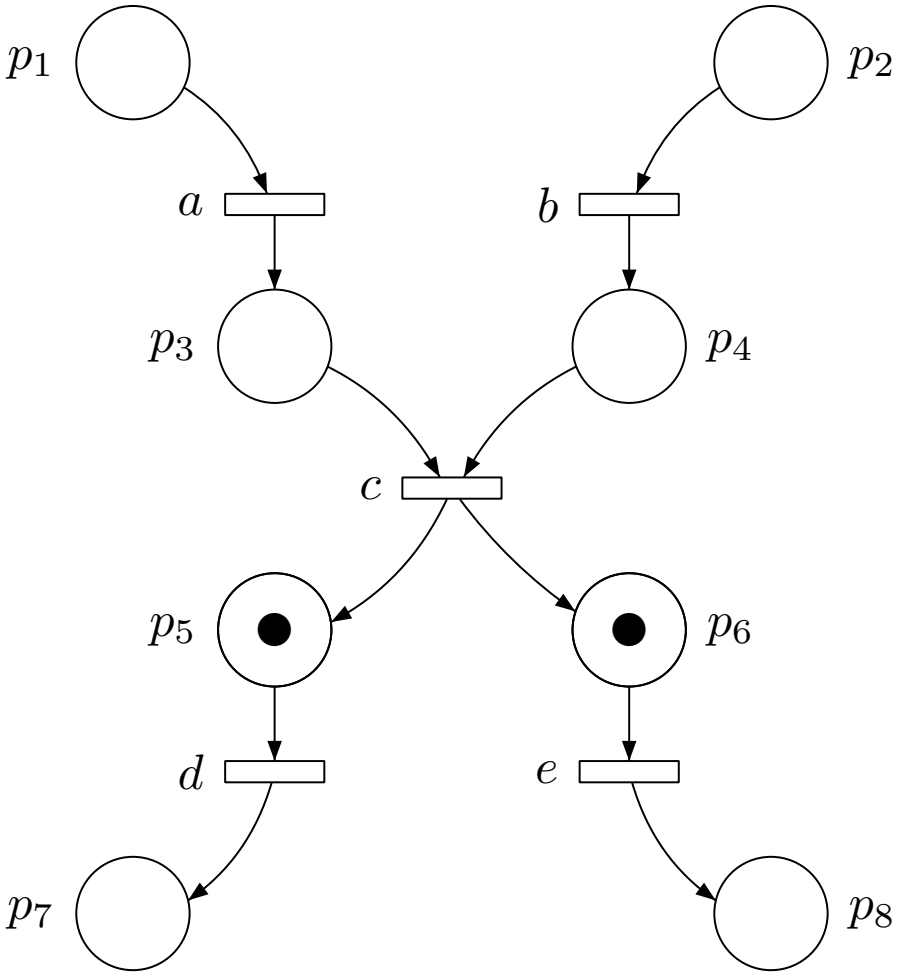
- state $s$: a set of places, e.g. $\{p_1, p_2\}$
- transition $c$ is enabled ($en_c$) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution. e.g. $\{p_5, p_8\}$



# One-safe Petri Nets

Place/Transition Nets:

- state $s$: a set of places, e.g. $\{p_1, p_2\}$
- transition $c$ is enabled ($en_c$) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution. e.g. $\{p_5, p_8\}$
- jointly enabled transitions are independent if they don't share places (e.g. $d, e$ in $\{p_5, p_6\}$)



# (Global) Control constraints

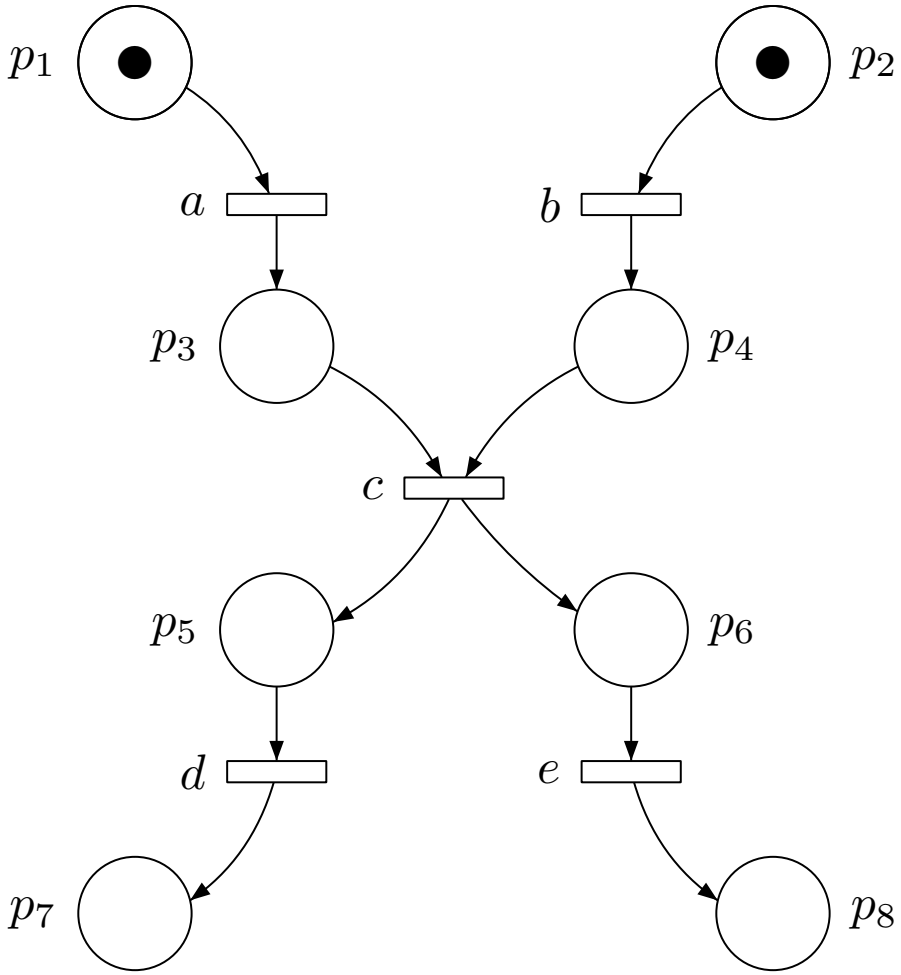
A control constraint $\Psi$ is a set of pairs (state, transition) expressing which transitions are authorized in each state, i.e. we assume the centralized control problem to be solved

Running example for this talk: (static) priority policies

- A priority policy $\ll$ is a strict partial order on the transitions
- Transition $t$ has maximal priority in state $s$ if:
  - no transition $t'$ such that $t \ll t'$ is enabled in $s$

# Priority Constraints

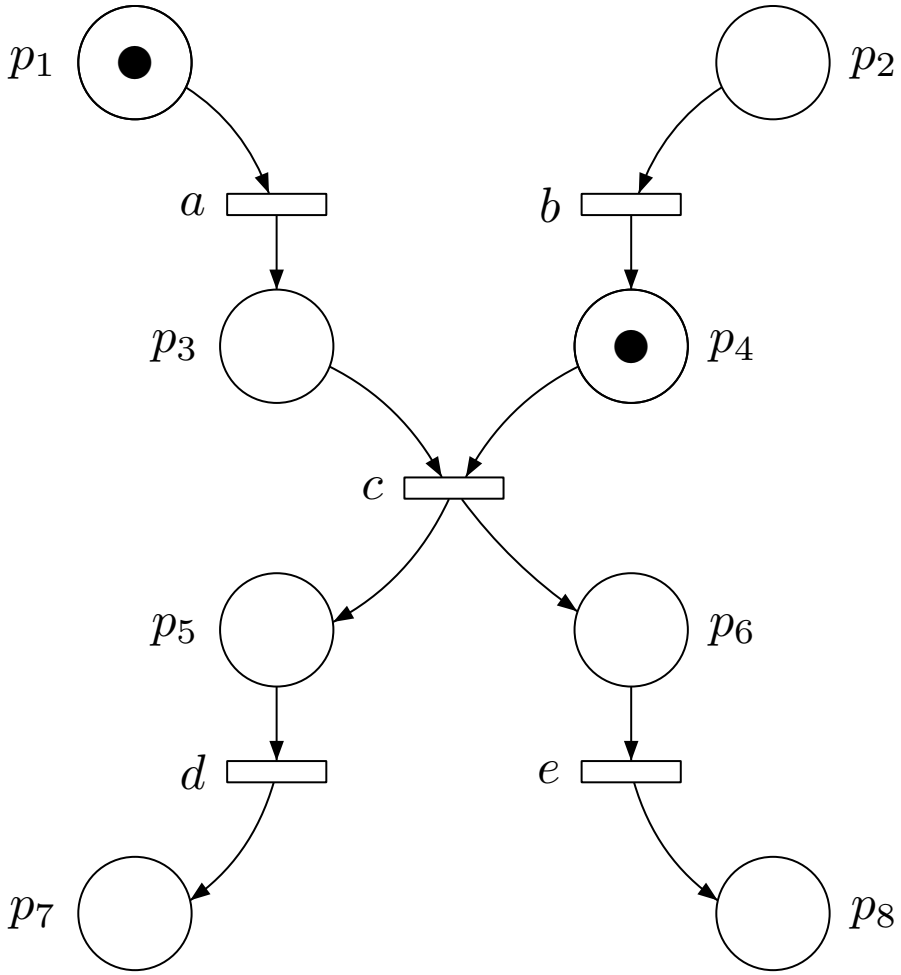
- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$




$a \ll b$ and $d \ll e$

# Priority Constraints

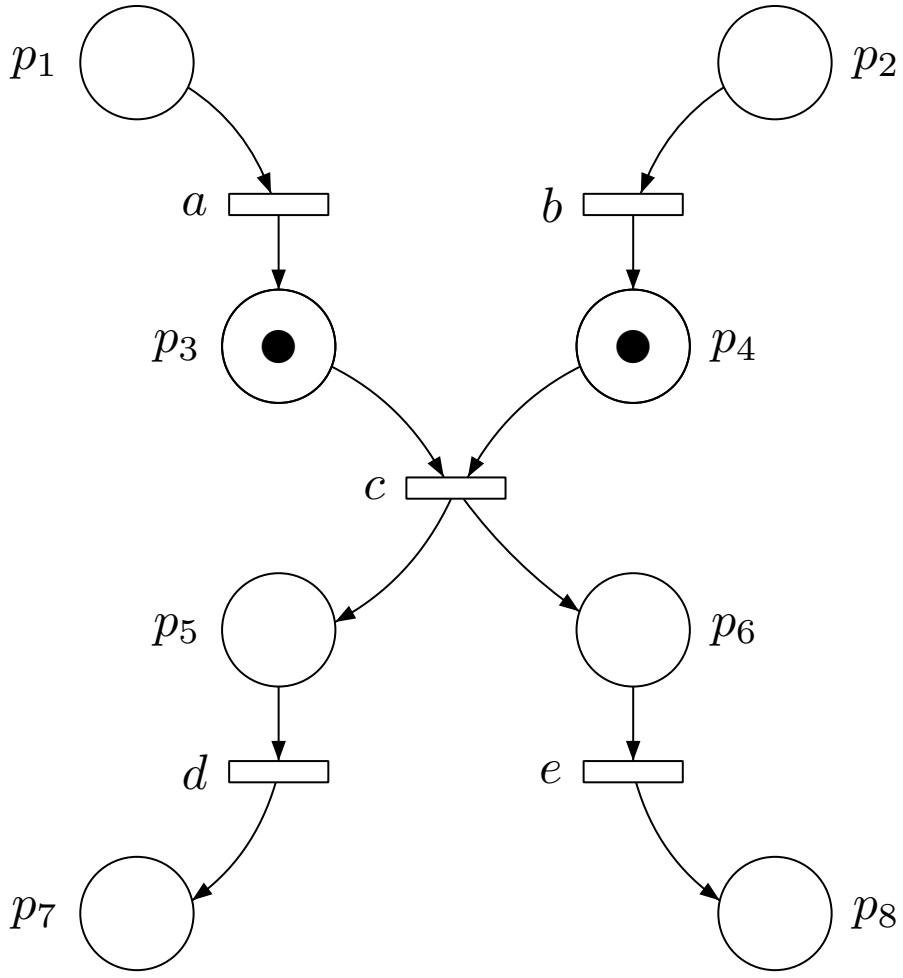
- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$




$a \ll b$ and $d \ll e$

# Priority Constraints

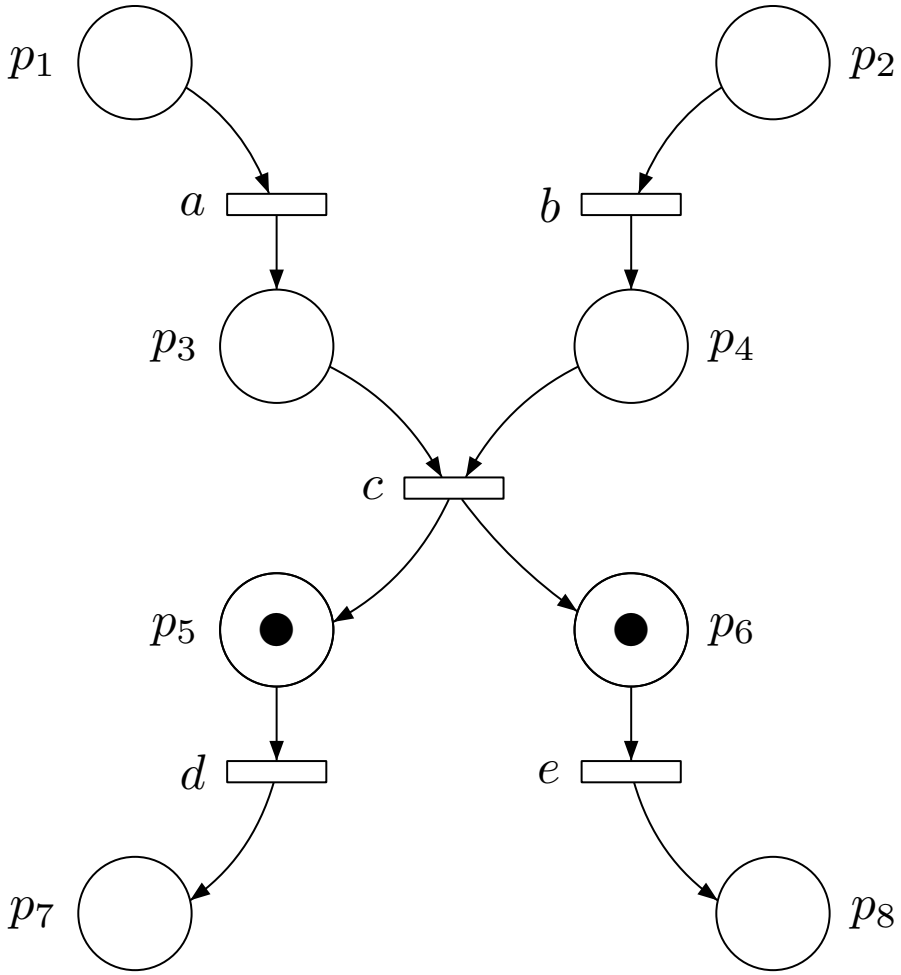
- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$




$a \ll b$ and $d \ll e$

# Priority Constraints

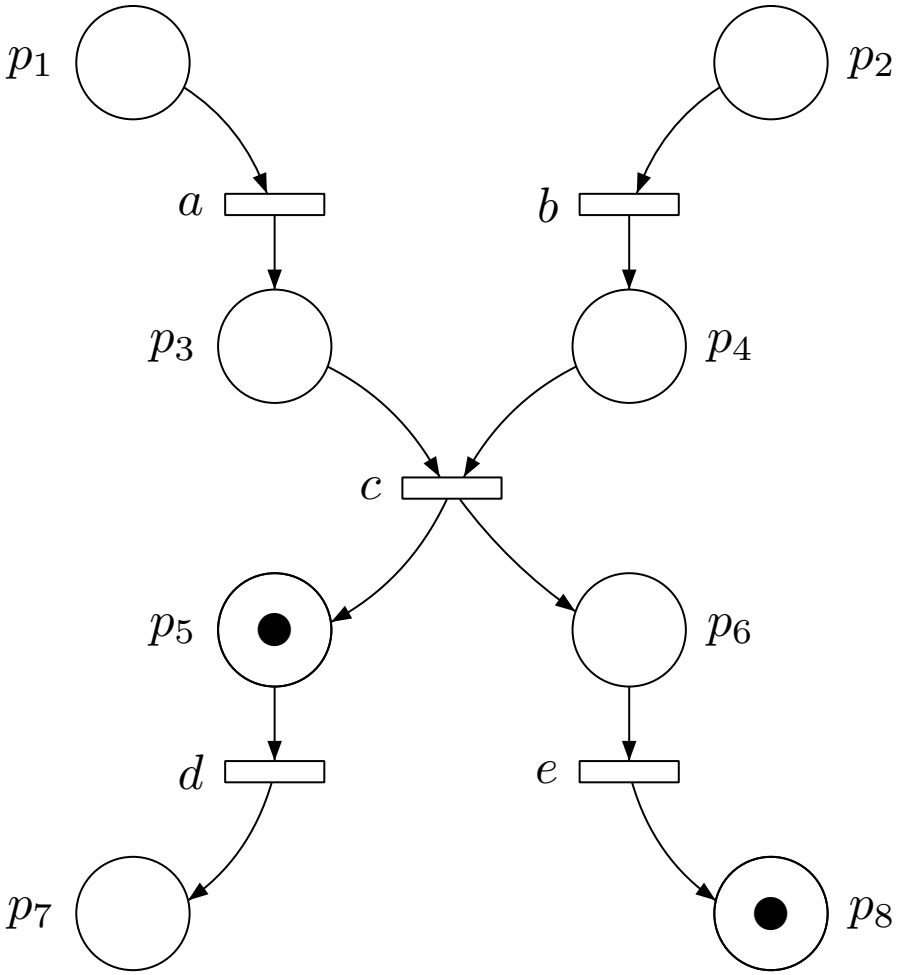
- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$




$a \ll b$ and $d \ll e$

# Priority Constraints

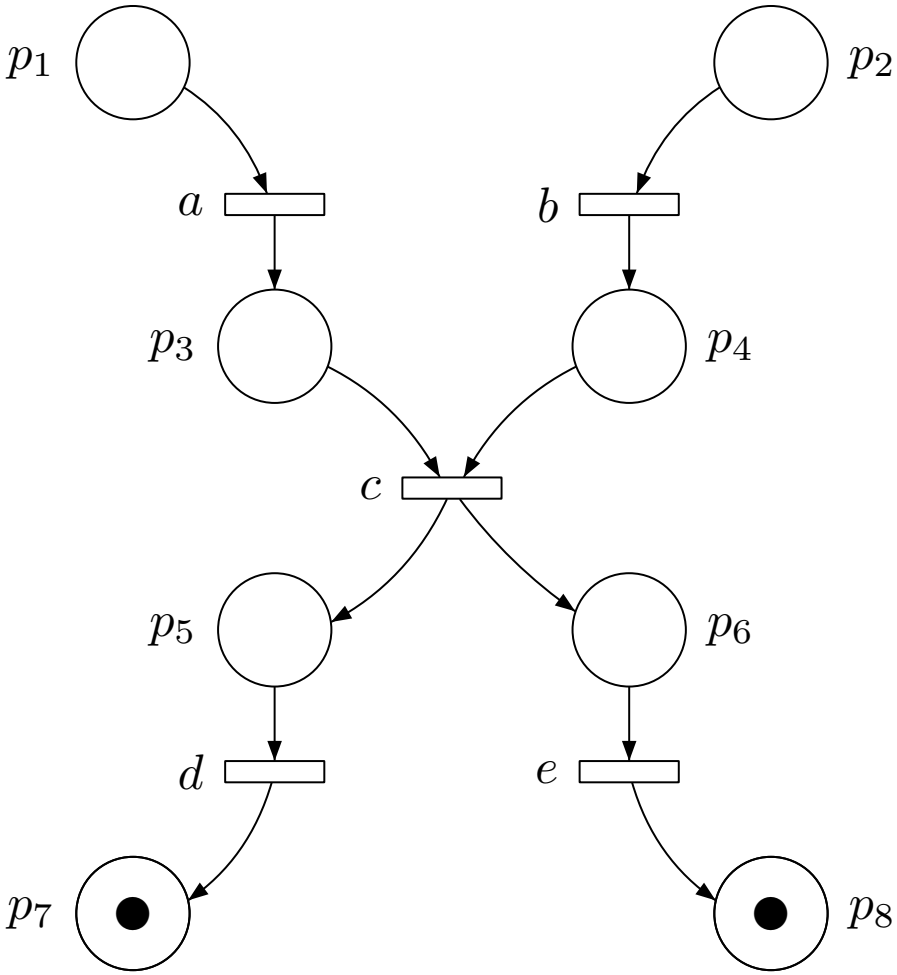
- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$




$a \ll b$ and $d \ll e$

# Priority Constraints

- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$

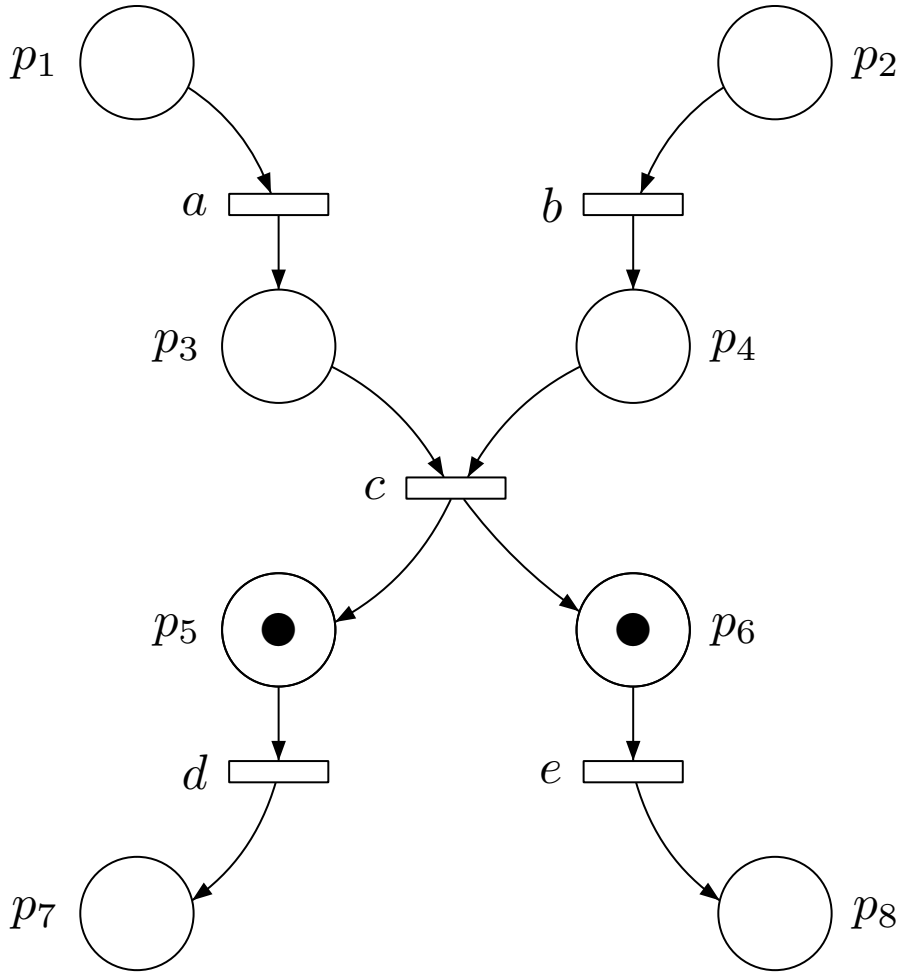



$a \ll b$ and $d \ll e$

# Priority Constraints

- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$

- Note: *independent* transitions may not be independent any more (e.g. $d, e$ in $\{p_5, p_6\}$)



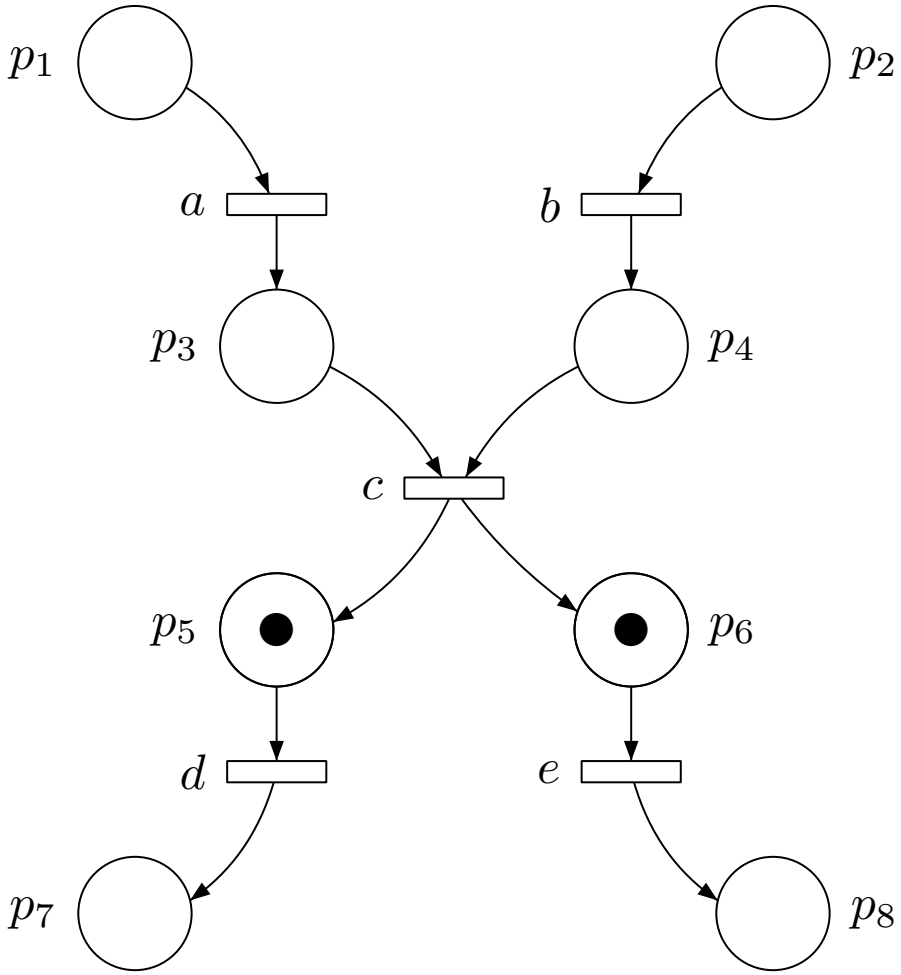

$a \ll b$ and $d \ll e$

# Priority Constraints

- A prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, $t_i$ has maximal priority in $s_i$

- Note: *independent* transitions may not be independent any more (e.g. $d, e$ in $\{p_5, p_6\}$)

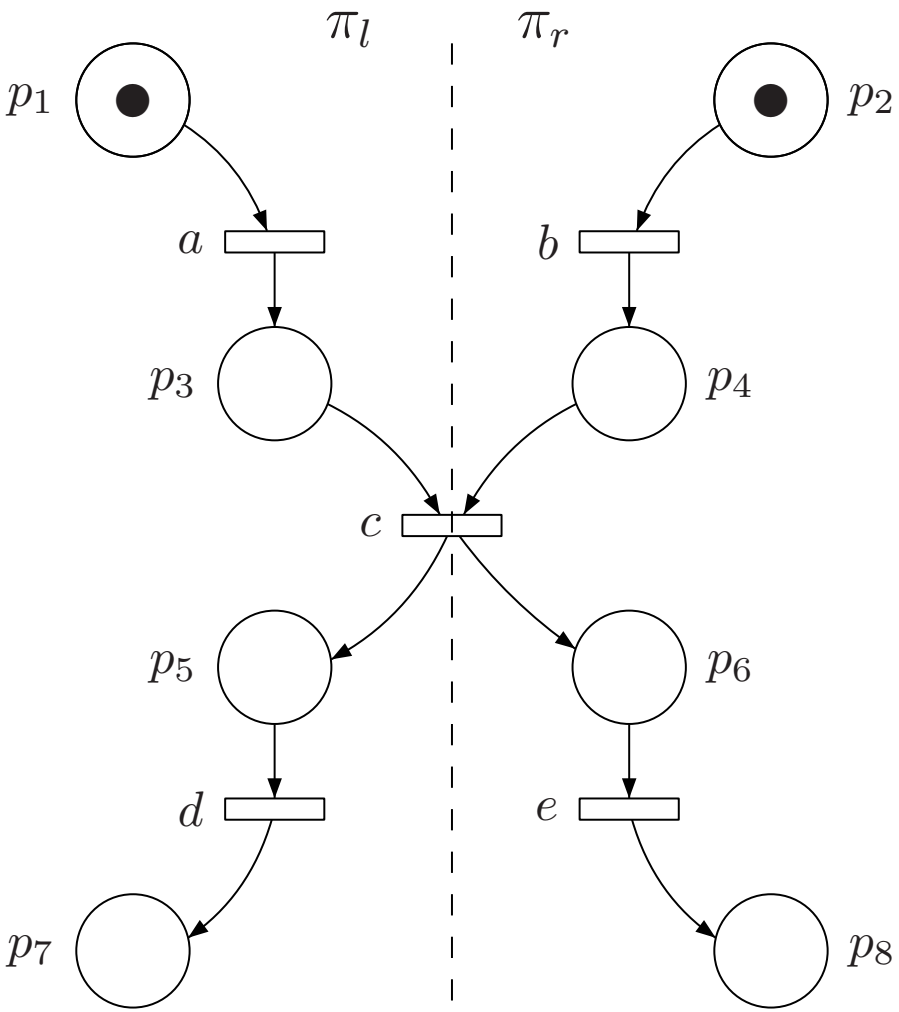
  [GPQ10]: use knowledge to optimize the transformation of the controlled system $(PN, \ll)$ into a Petri Net? which then can be analyzed & implemented "as usually"




$a \ll b$ and $d \ll e$

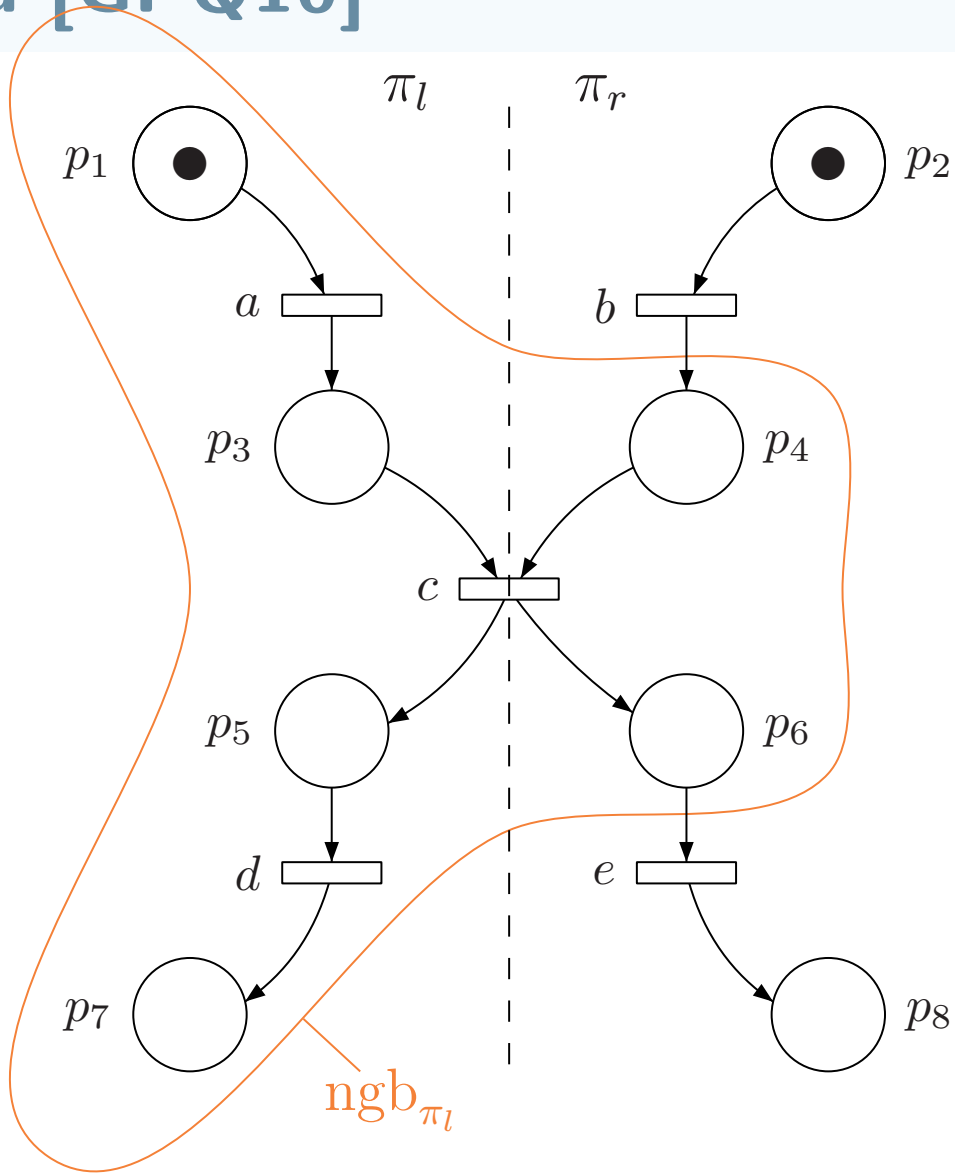
# Compositional Setting à la [GPQ10]

- A thread $\pi$ is a set of places $P_\pi \subseteq P$ (exactly 1 token) and the corresponding transitions $T_\pi \subseteq T$



# Compositional Setting à la [GPQ10]

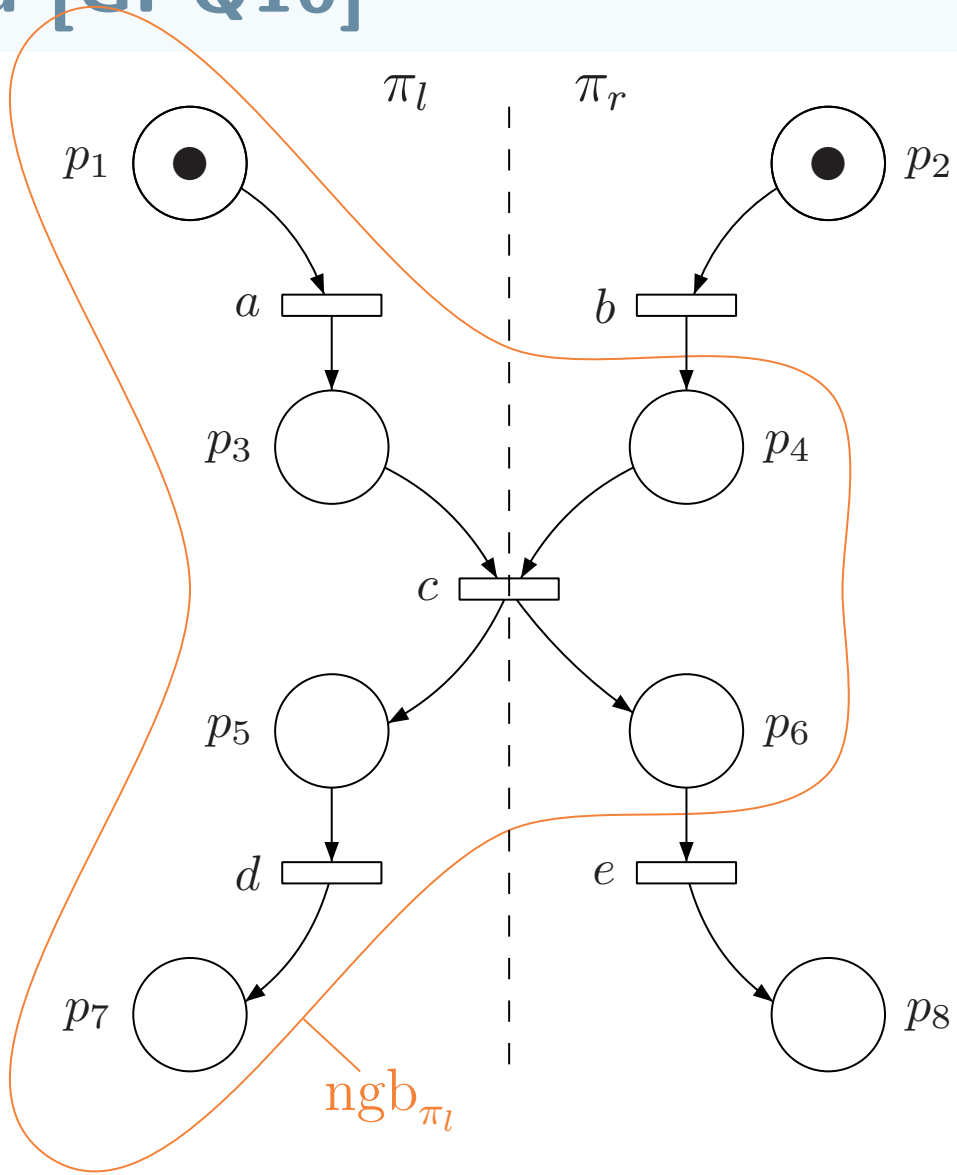
- A thread $\pi$ is a set of places $P_\pi \subseteq P$ (exactly 1 token) and the corresponding transitions $T_\pi \subseteq T$
- The neighborhood $\mathrm{ngb}_\pi$ of $\pi$ is $\bigcup_{t \in T_\pi} ({}^\bullet t \cup t^\bullet)$



# Compositional Setting à la [GPQ10]

- A thread $\pi$ is a set of places $P_\pi \subseteq P$ (exactly 1 token) and the corresponding transitions $T_\pi \subseteq T$
- The neighborhood $\mathrm{ngb}_\pi$ of $\pi$ is $\bigcup_{t \in T_\pi} ({}^\bullet t \cup t^\bullet)$
- The set of local states of $\pi$ is $\{s \cap \mathrm{ngb}_\pi \mid s \in S\}$
  The local state of $s$ in $\pi$ is denoted $s|_\pi$

  E.g. the local state of $\{p_1, p_2\}$ in $\pi_l$ is $\{p_1\}$
  the local state of $\{p_1, p_4\}$ in $\pi_l$ is $\{p_1, p_4\}$

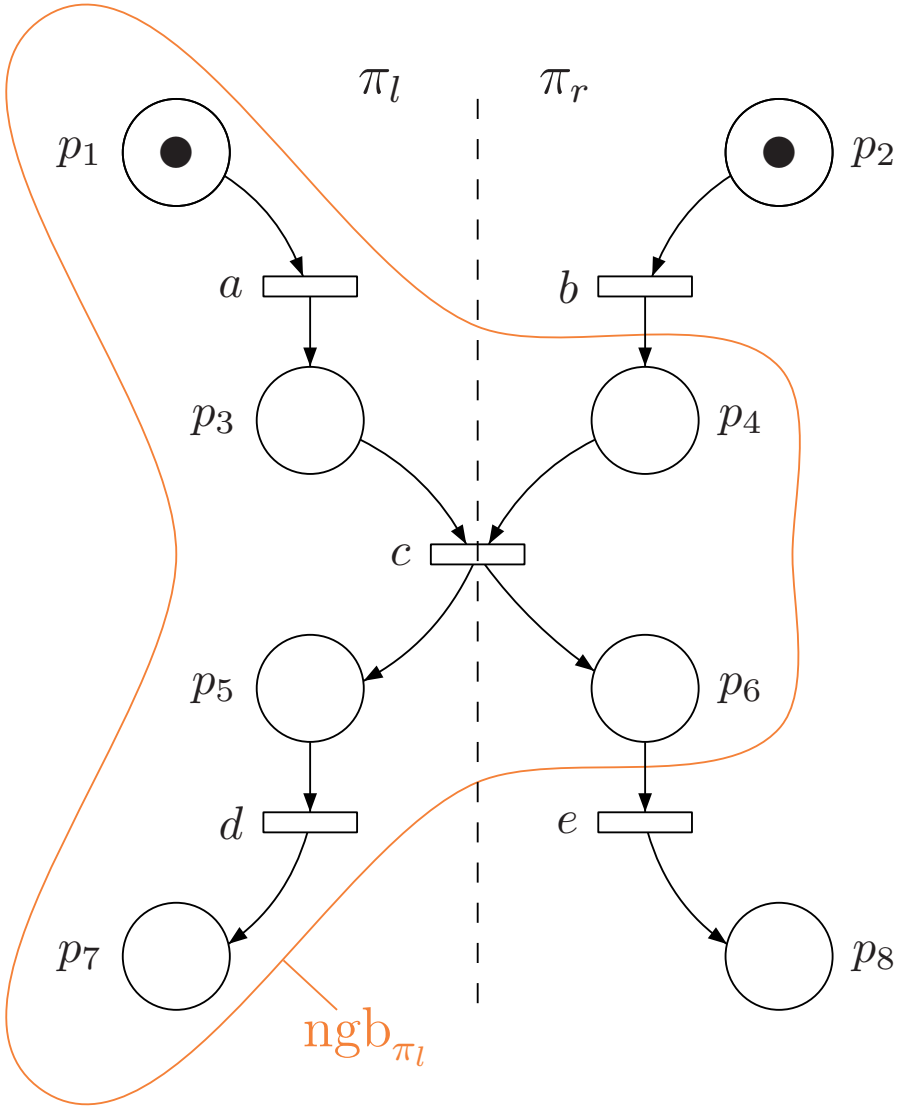


# Definition of Knowledge

- Thread $\pi$ knows a property $\varphi$ in a local $s|_\pi$ if $\varphi$ holds in all reachable $s'$ such that $s'|_\pi = s|_\pi$

$$s|_\pi \models K_\pi \varphi$$

e.g. $\{p_1\} \models K_{\pi_l} p_2$
as $\{p_1, p_8\}$ is unreachable

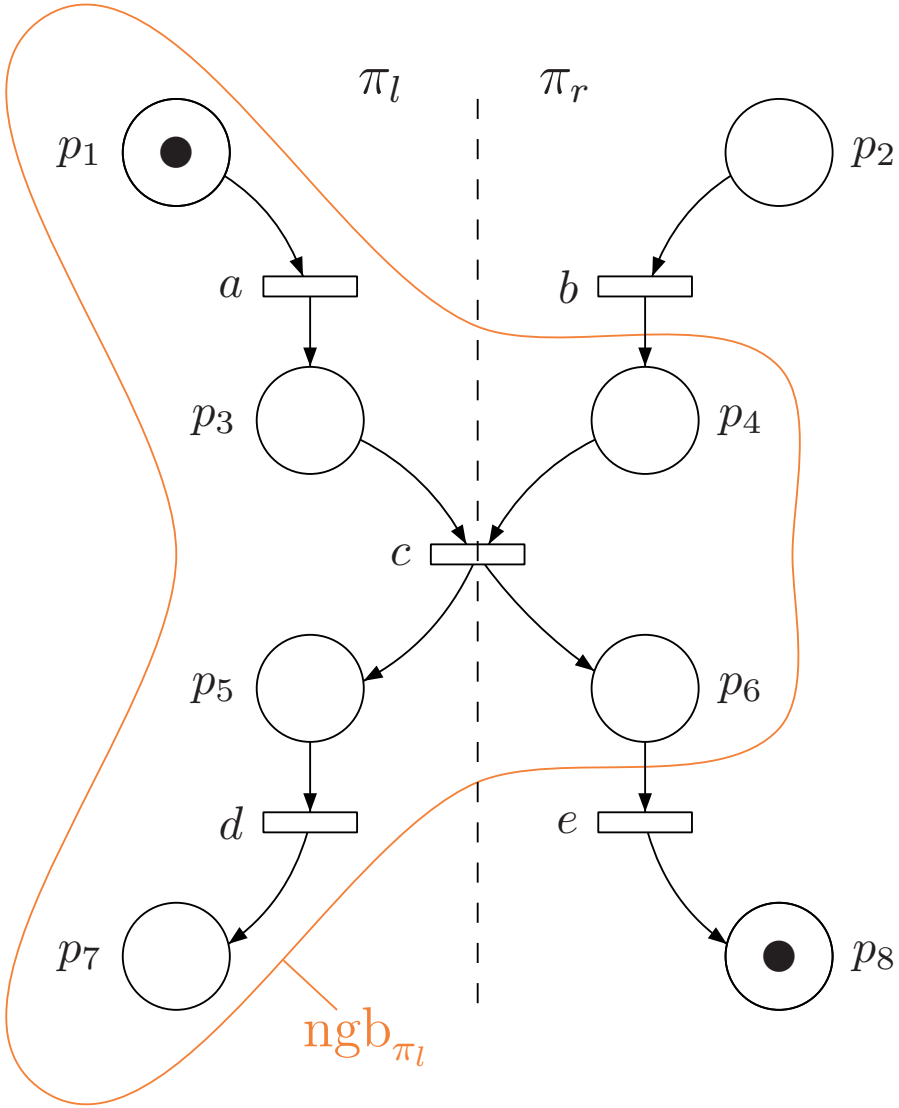


# Definition of Knowledge

- Thread $\pi$ knows a property $\varphi$ in a local $s|_\pi$ if $\varphi$ holds in all reachable $s'$ such that $s'|_\pi = s|_\pi$

$$s|_\pi \models K_\pi \varphi$$

e.g. $\{p_1\} \models K_{\pi_l} p_2$
as $\{p_1, p_8\}$ is unreachable

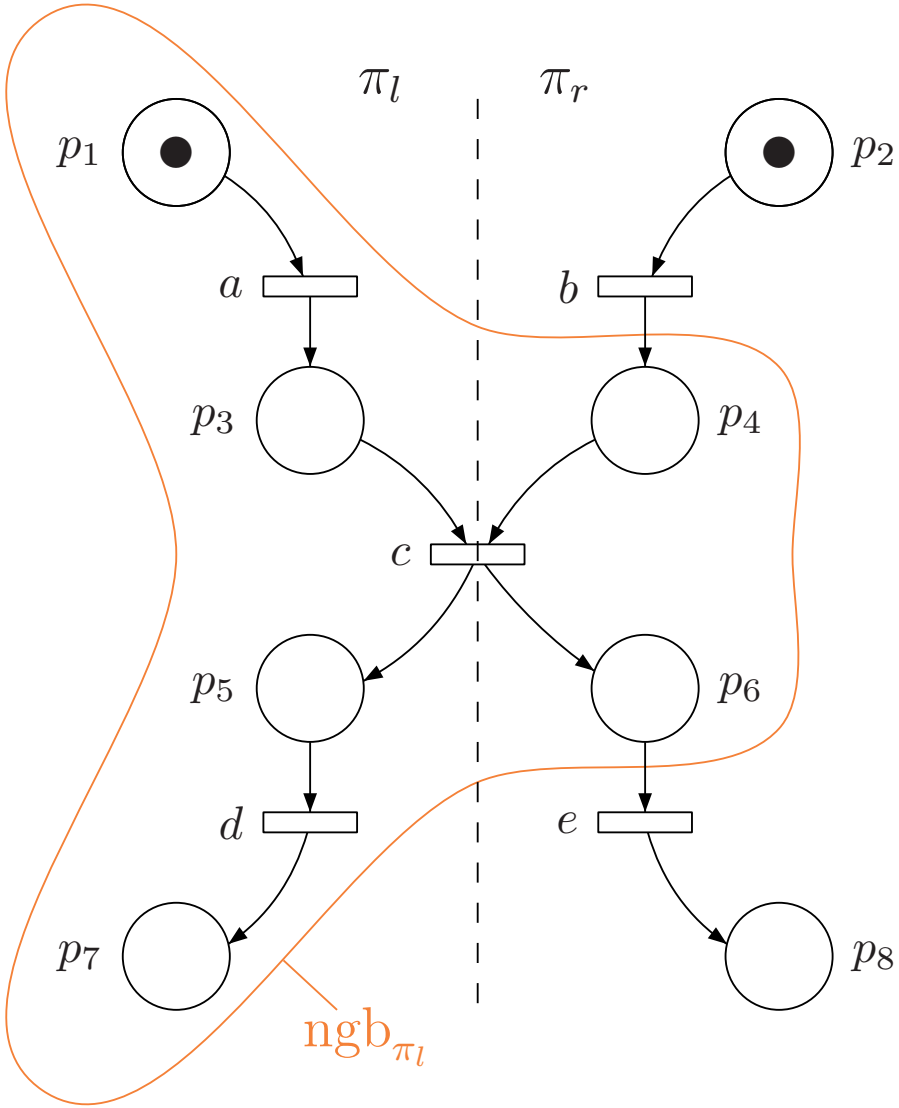


# Definition of Knowledge

- Thread $\pi$ knows a property $\varphi$ in a local $s|_\pi$ if $\varphi$ holds in all reachable $s'$ such that $s'|_\pi = s|_\pi$

$$s|_\pi \models K_\pi \varphi$$

e.g. $\{p_1\} \models K_{\pi_l} p_2$
as $\{p_1, p_8\}$ is unreachable



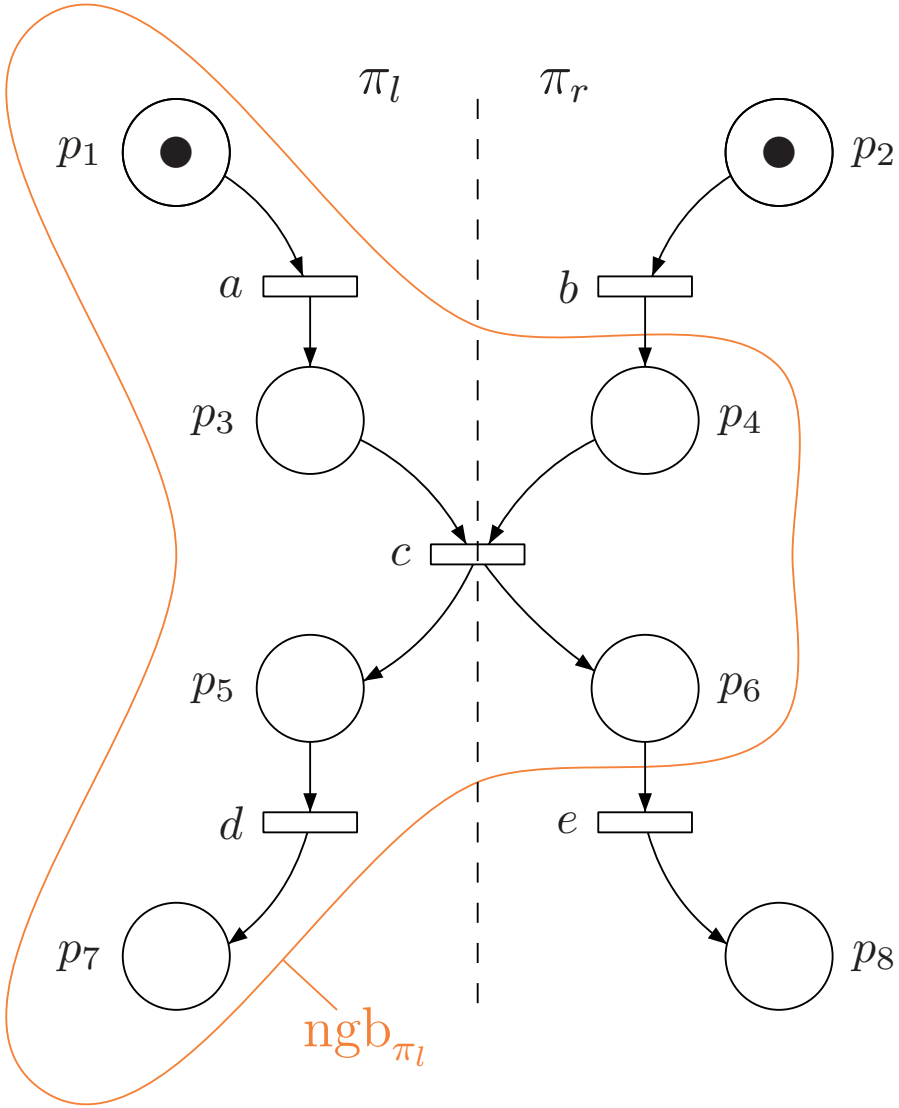
# Definition of Knowledge

- Thread $\pi$ knows a property $\varphi$ in a local $s|_\pi$ if $\varphi$ holds in all reachable $s'$ such that $s'|_\pi = s|_\pi$

$$s|_\pi \models K_\pi \varphi$$

e.g. $\{p_1\} \models K_{\pi_l} p_2$
as $\{p_1, p_8\}$ is unreachable

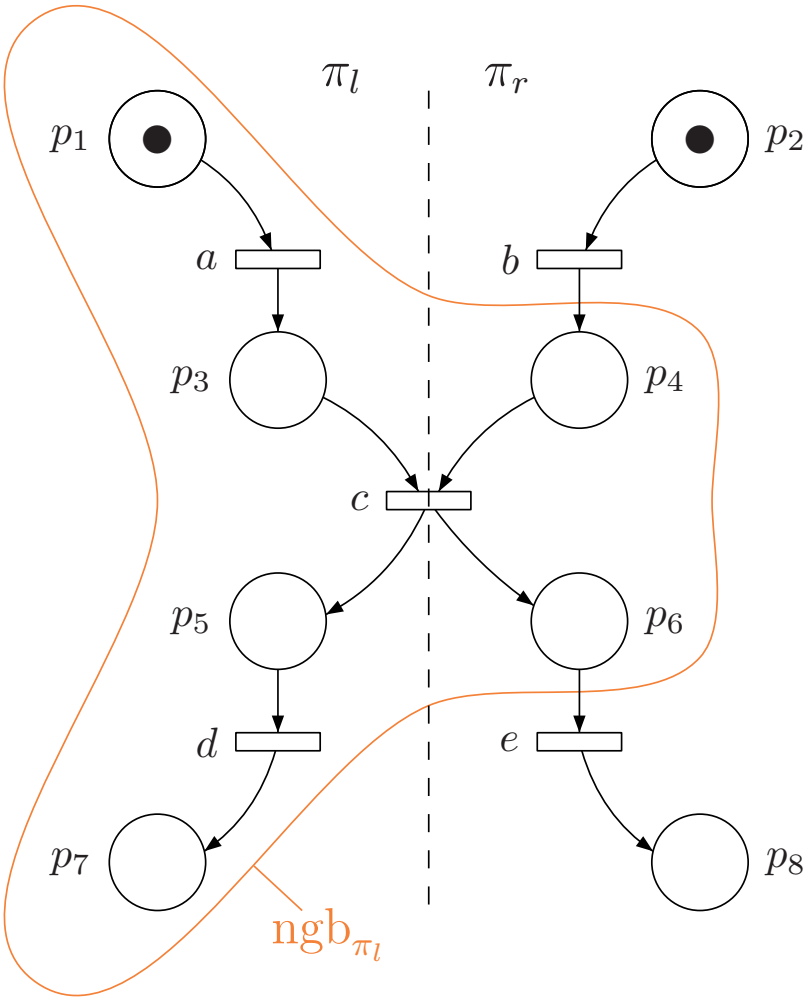
Knowledge is stable:
if $s|_\pi \models K_\pi \varphi$, then
$\varphi$ continues to hold (globally) unless a local move occurs



# Knowledge for enforcing constraints

What are useful knowledge properties for enforcing priorities?

- $\pi_l$ can fire transition $a$ if $max_a$: $en_a \wedge \neg en_b$
- $en_a$ is a local condition, always *known* in $\pi_l$: $s|_{\pi_l} \models K_{\pi_l} en_a$ or $s|_{\pi_l} \models K_{\pi_l} \neg en_a$



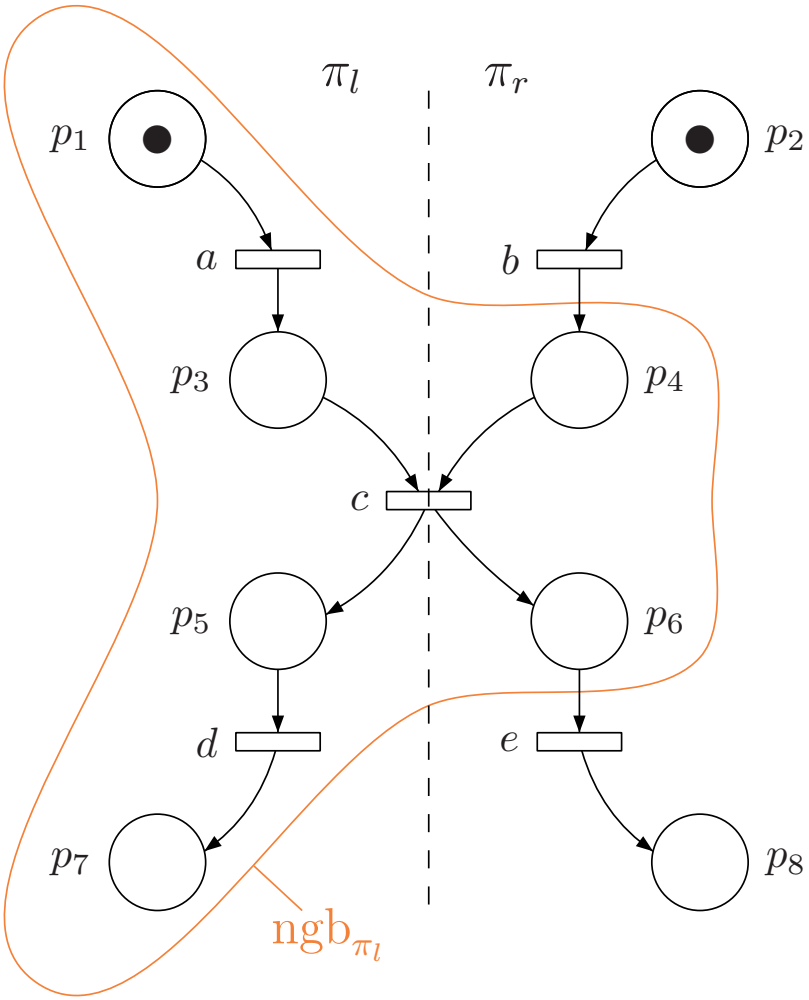

$a \ll b$ and $d \ll e$

# Knowledge for enforcing constraints

What are useful knowledge properties for enforcing priorities?

- $\pi_l$ can fire transition $a$ if $max_a$: $en_a \wedge \neg en_b$
- $en_a$ is a local condition, always *known* in $\pi_l$: $s|_{\pi_l} \models K_{\pi_l} en_a$ or $s|_{\pi_l} \models K_{\pi_l} \neg en_a$

Question: are there local states $s|_{\pi_l}$ in which also $\neg en_b$ holds ?

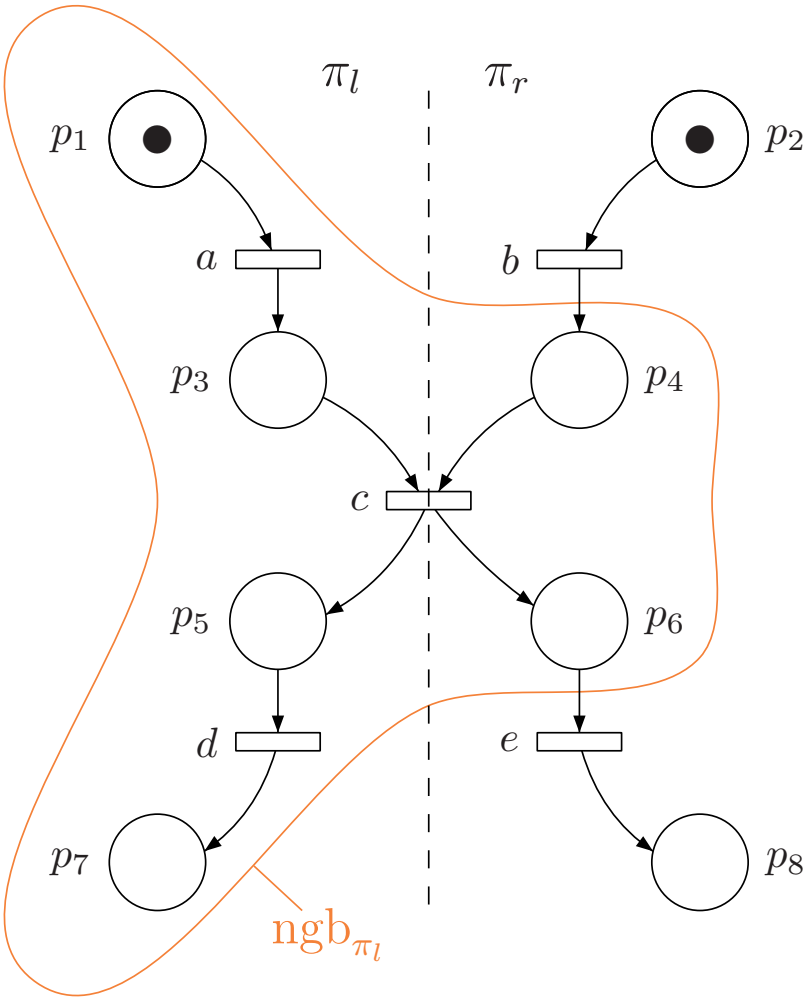



$a \ll b$ and $d \ll e$

# Knowledge for enforcing constraints

What are useful knowledge properties for enforcing priorities?

- $\{p_1\} \models K_{\pi_l} en_a$ but $\{p_1\} \not\models K_{\pi_l} \neg en_b$

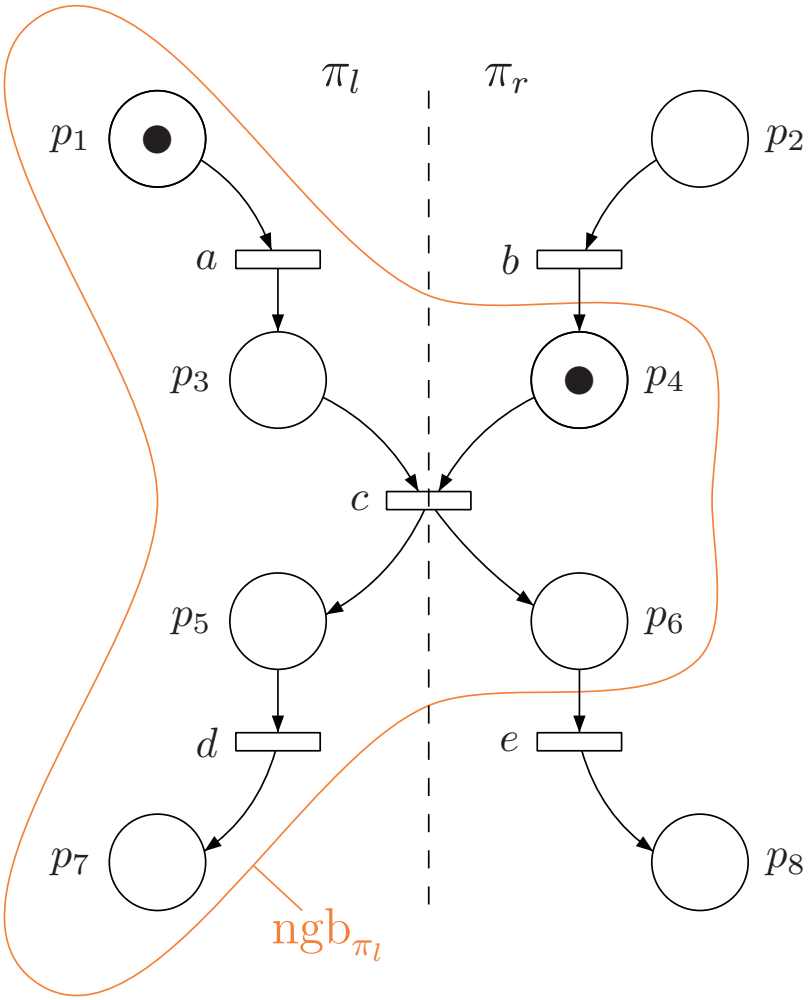



$a \ll b$ and $d \ll e$

# Knowledge for enforcing constraints

What are useful knowledge properties for enforcing priorities?

- $\{p_1\} \models K_{\pi_l} en_a$ but $\{p_1\} \not\models K_{\pi_l} \neg en_b$
- $\{p_1, p_4\} \models K_{\pi_l} en_a$ and $\{p_1, p_4\} \models K_{\pi_l} \neg en_b$

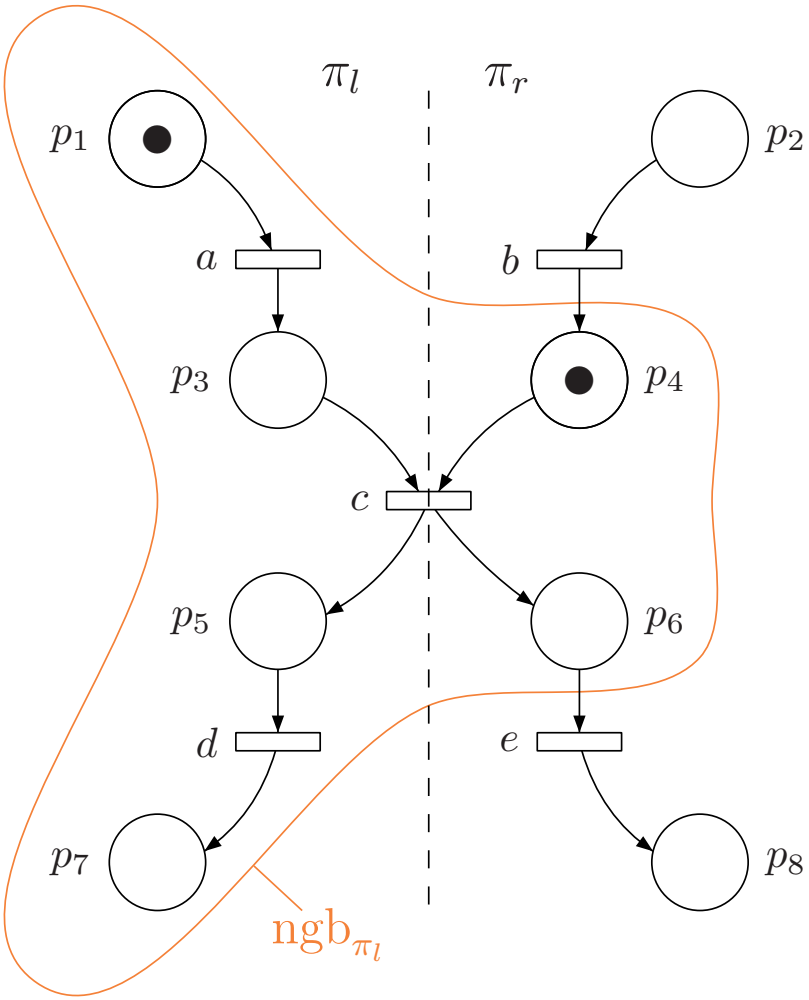



$a \ll b$ and $d \ll e$

# Knowledge for enforcing constraints

What are useful knowledge properties for enforcing priorities?

- $\{p_1\} \models K_{\pi_l} en_a$ but $\{p_1\} \not\models K_{\pi_l} \neg en_b$
- $\{p_1, p_4\} \models K_{\pi_l} en_a$ and $\{p_1, p_4\} \models K_{\pi_l} \neg en_b$
- $\{p_5, p_6\} \models K_{\pi_l} en_d$ but $\{p_5, p_6\} \models K_{\pi_l} en_e$
- $\{p_5\} \models K_{\pi_l} en_d$ and $\{p_5\} \models K_{\pi_l} \neg en_e$




$a \ll b$ and $d \ll e$

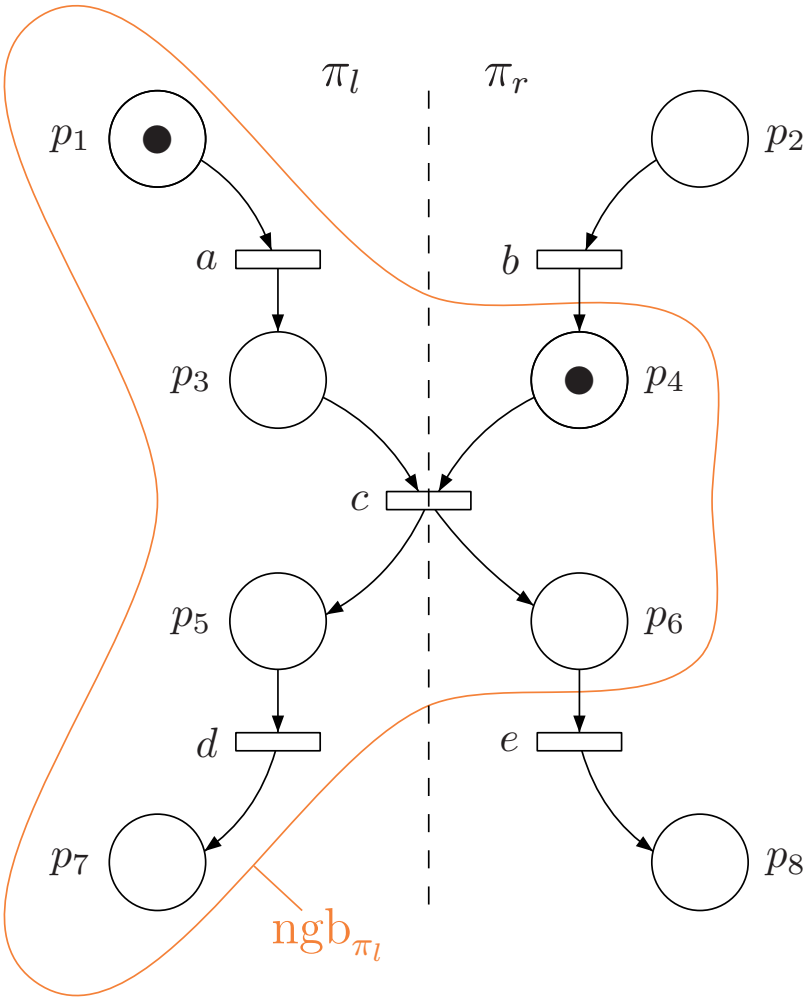
# Knowledge for enforcing constraints

Here, we can enforce the global constraint by just adding local conditions:

- allows $a$ only in the local state $\{p_1, p_4\}$ of $\pi_l$
- allows $d$ only in the local state $\{p_5\}$ of $\pi_l$

Achieve a distributed solution: use a standard solution for PN

In the general case: one need to add new transitions (synchronizations) but exploiting knowledge helps to minimize the number of synchronizations

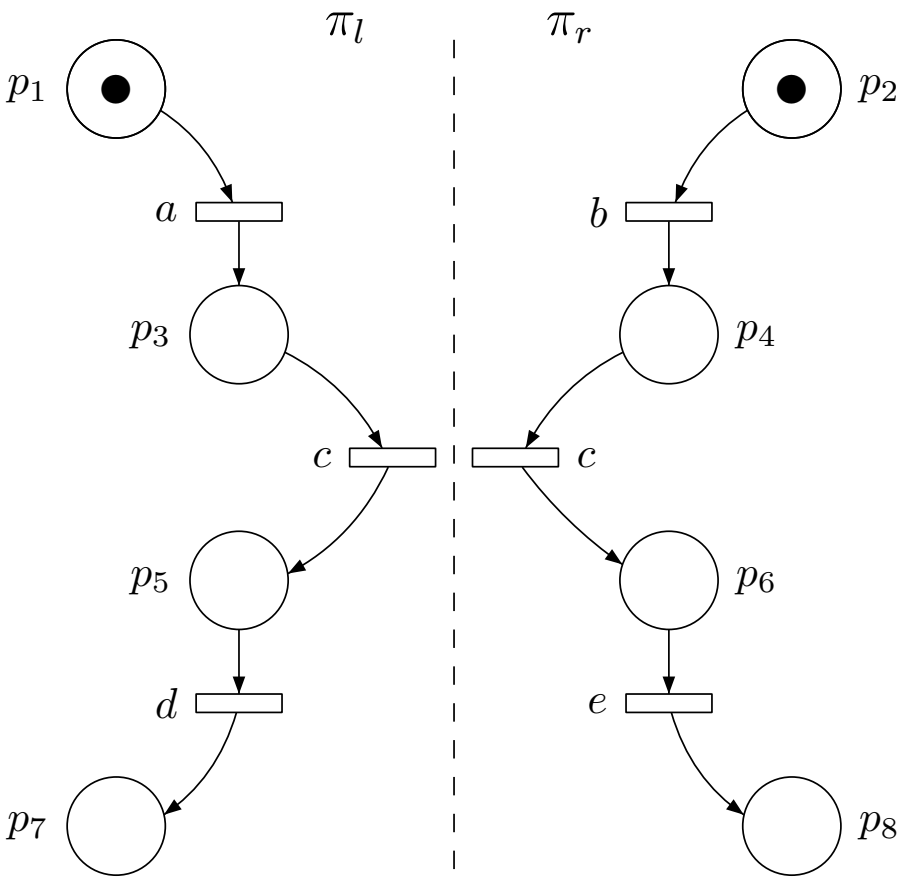



$a \ll b$ and $d \ll e$

# Outline

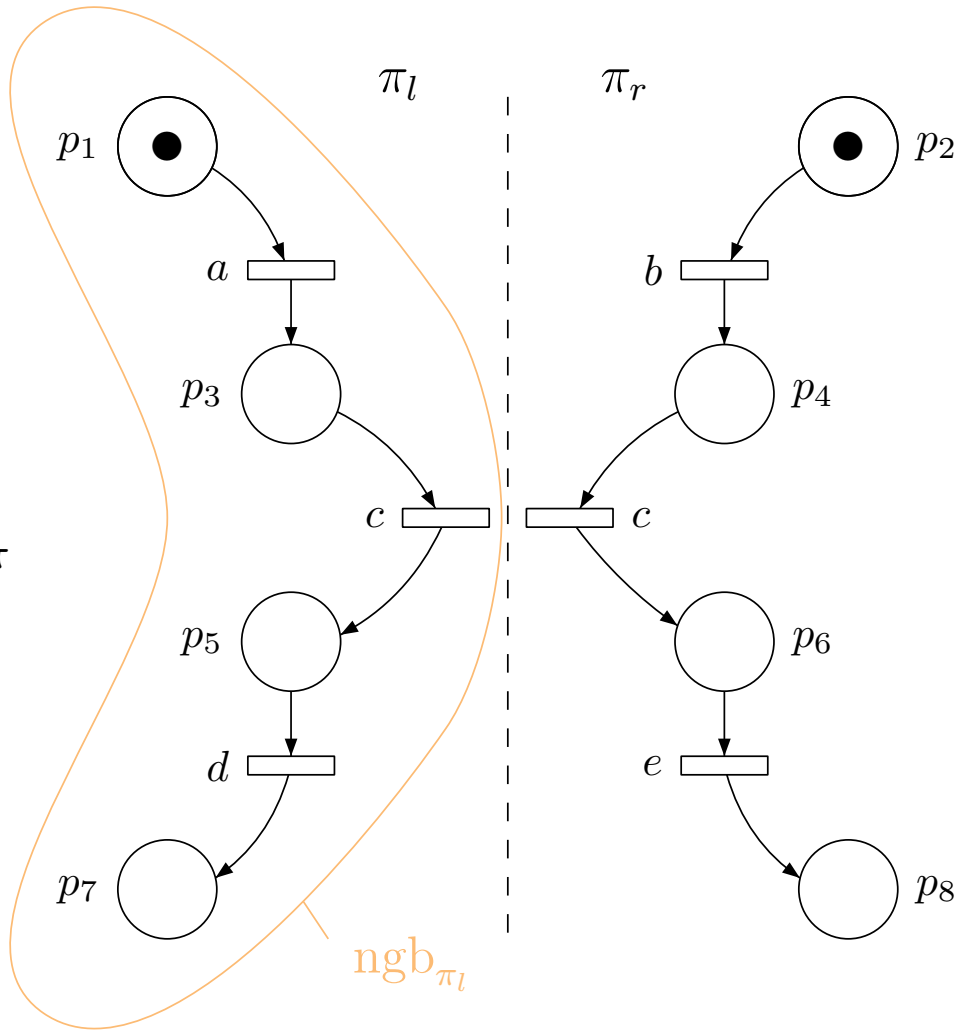
# Distributed setting

- A process $\pi$ is a set of places $P_\pi \subseteq P$ (exactly 1 token) and $T_\pi$ contains for each transition in which $\pi$ is involved, a corresponding local transition



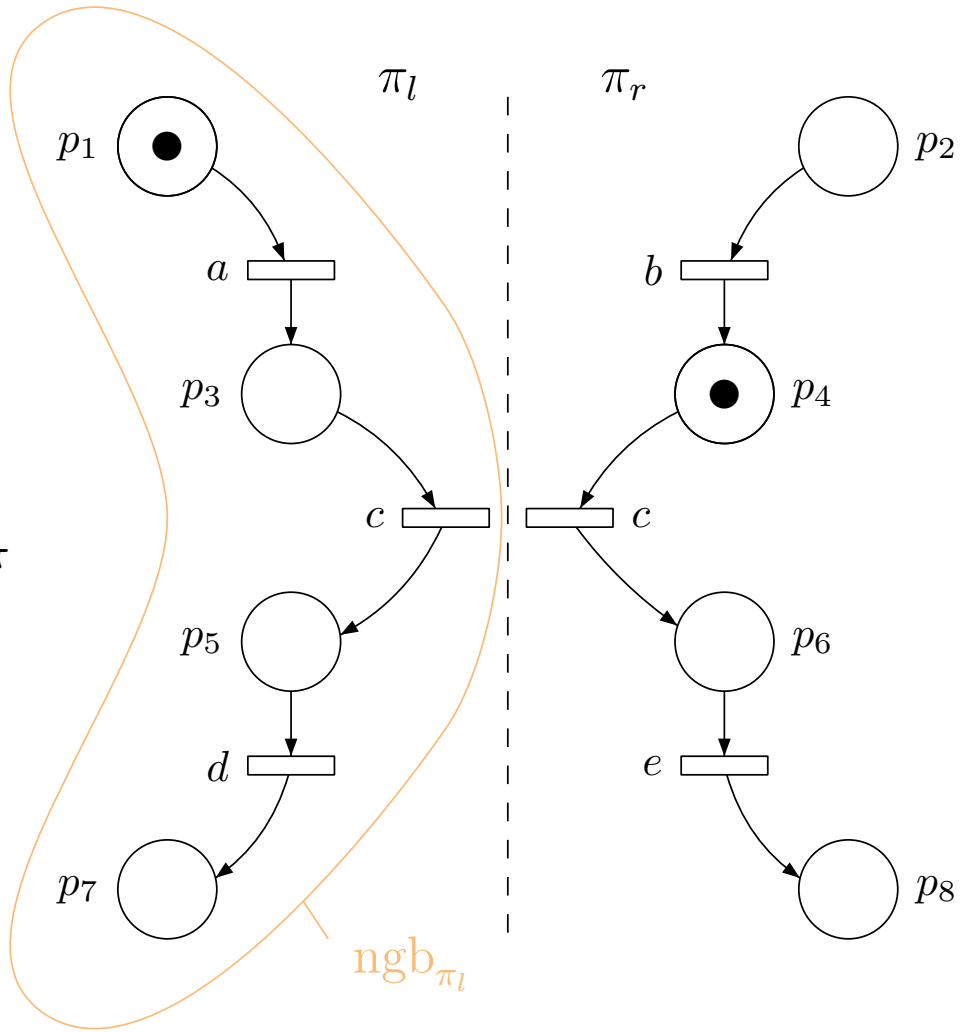
# Distributed setting

- A process $\pi$ is a set of places $P_\pi \subseteq P$ (exactly 1 token) and $T_\pi$ contains for each transition in which $\pi$ is involved, a corresponding local transition
- The neighborhood $\mathrm{ngb}_\pi$ of $\pi$ is exactly the set of local places $P_\pi$



# Distributed setting

- A process $\pi$ is a set of places $P_\pi \subseteq P$ (exactly 1 token) and $T_\pi$ contains for each transition in which $\pi$ is involved, a corresponding local transition
- The neighborhood $\mathrm{ngb}_\pi$ of $\pi$ is exactly the set of local places $P_\pi$
- Everything else is unchanged

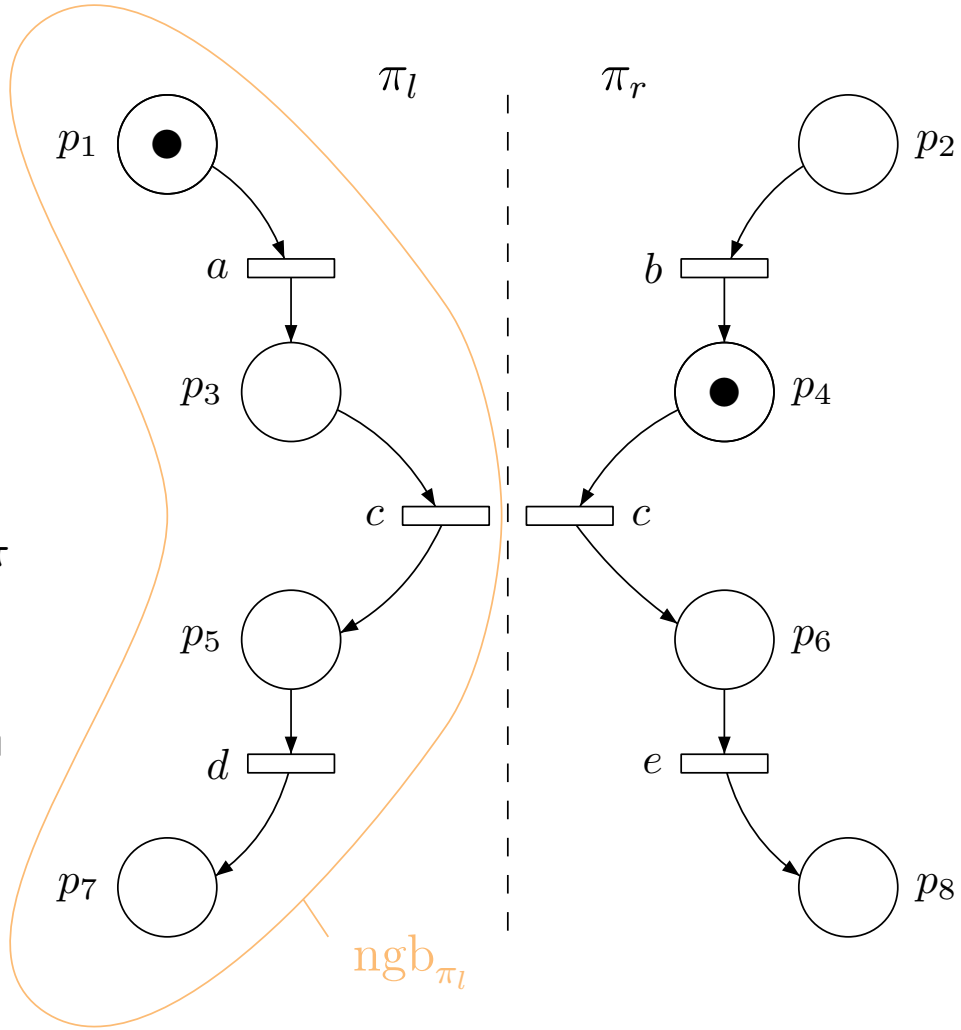


# Distributed setting

- A process $\pi$ is a set of places $P_\pi \subseteq P$ (exactly 1 token) and $T_\pi$ contains for each transition in which $\pi$ is involved, a corresponding local transition
- The neighborhood $\mathrm{ngb}_\pi$ of $\pi$ is exactly the set of local places $P_\pi$
- Everything else is unchanged

We have now a new Petri Net with a **different transition set**.

Question: what does it mean that a distributed execution implements a centralized one ?



# Implementation relations $\preceq$

$\preceq$ must support the methodology:

1. verify $PN \models \varphi$ for some global property $\varphi$
2. preserve $\varphi$ on a distributed implementation $I$ ($PN \models \varphi$ and $I \preceq PN$ guarantees $I \models \varphi$)

# Implementation relations $\preceq$

$\preceq$ must support the methodology:

1. verify $PN \models \varphi$ for some global property $\varphi$
2. preserve $\varphi$ on a distributed implementation $I$ ($PN \models \varphi$ and $I \preceq PN$ guarantees $I \models \varphi$)

(Almost) Minimal requirement on $\preceq$: guarantee *sequential consistency*

(1) Transition correctness (local traces are projections of a trace of PN)
(2) Atomicity (all local traces are projections of the same trace)

# Implementation relations $\preceq$

$\preceq$ must support the methodology:

1. verify $PN \models \varphi$ for some global property $\varphi$
2. preserve $\varphi$ on a distributed implementation $I$ ($PN \models \varphi$ and $I \preceq PN$ guarantees $I \models \varphi$)

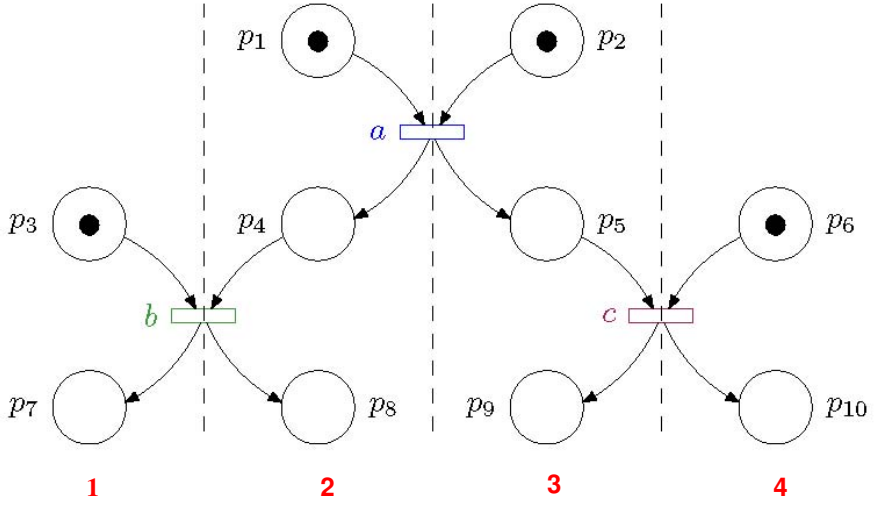
(Almost) Minimal requirement on $\preceq$: guarantee *sequential consistency*

(1) Transition correctness (local traces are projections of a trace of PN)
(2) Atomicity (all local traces are projections of the same trace)

Typical implementation relations add:

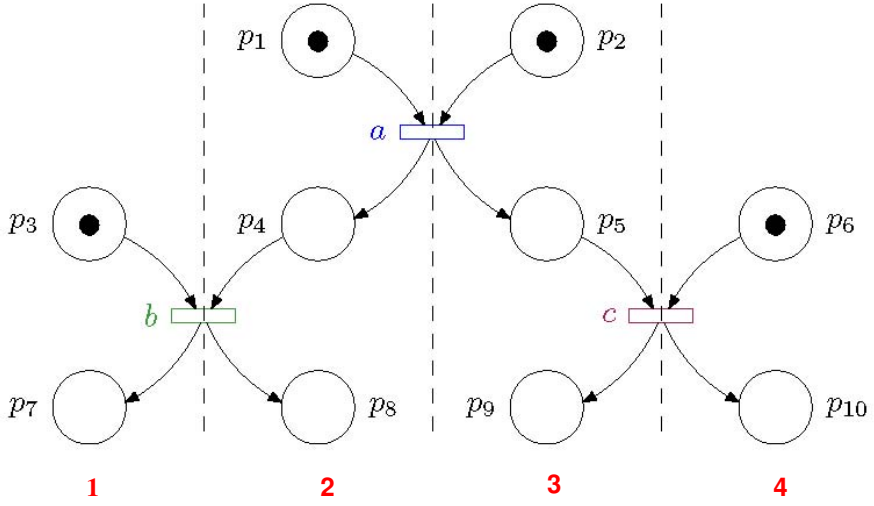
(3) Inter process order constraints (synchronize before/after joint transitions
(4) Progress (or coverage) constraints

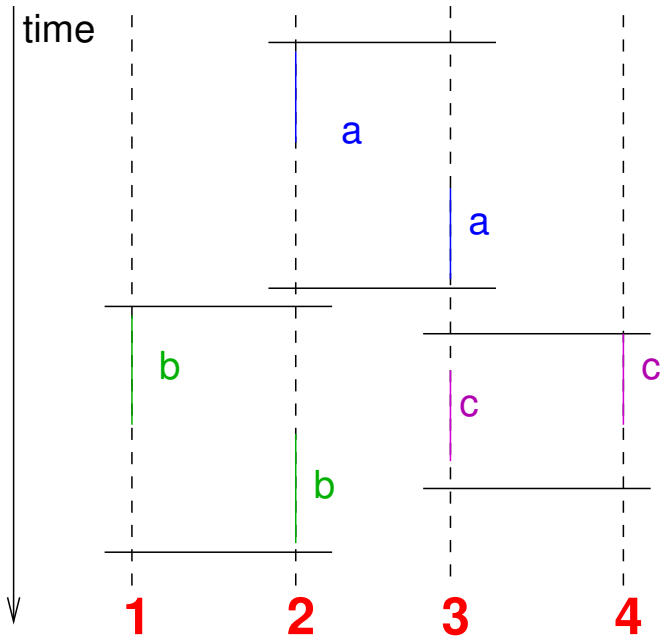
# Illustrating some Implementation relations



# Illustrating some Implementation relations

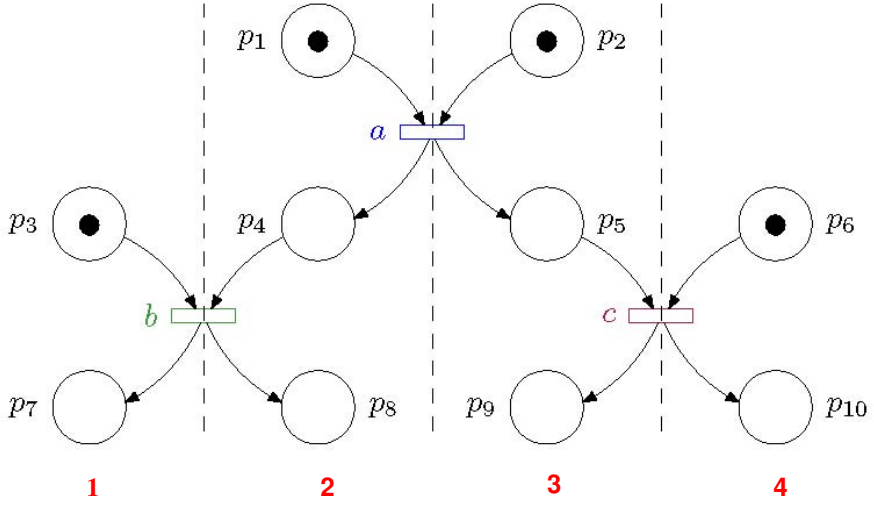
$\preceq_{ss}$: requires synchronization before and after transitions

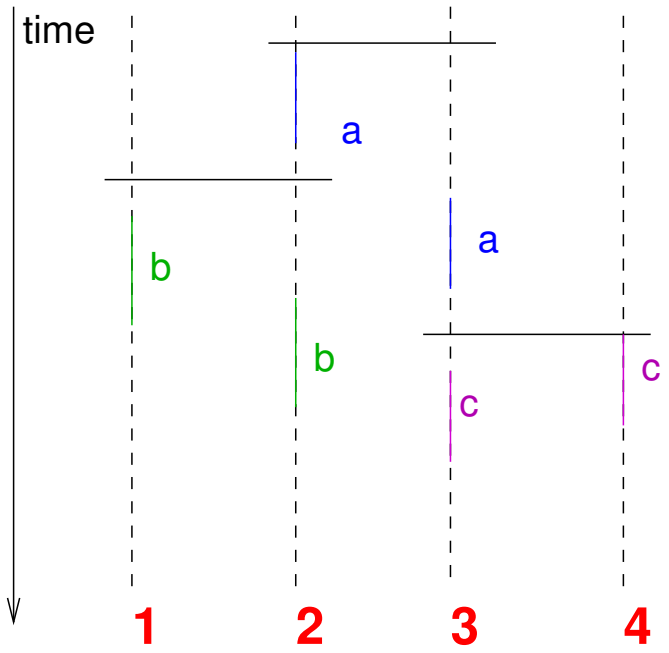


# Illustrating some Implementation relations

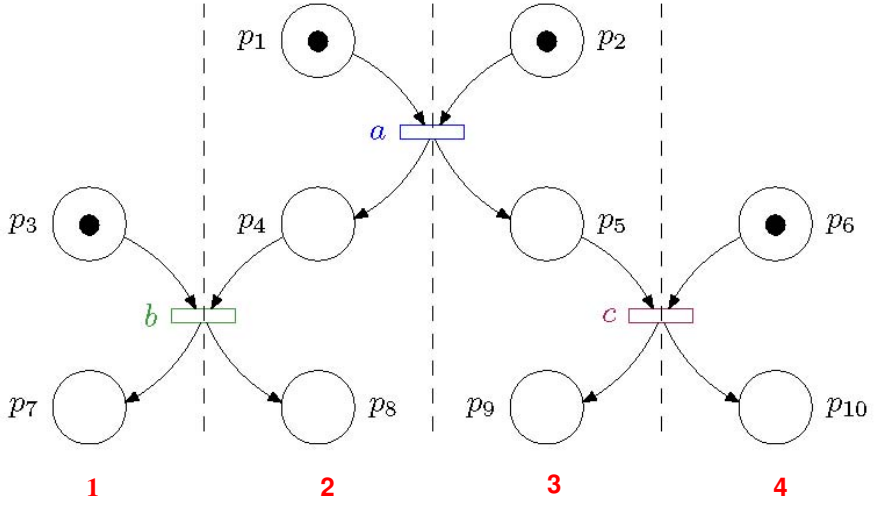
$\preceq$: requires synchronization only before transitions

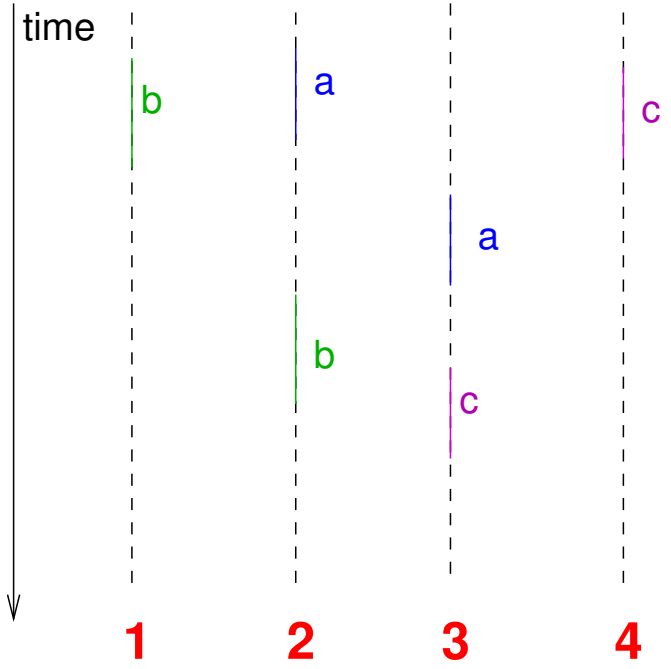


# Illustrating some Implementation relations

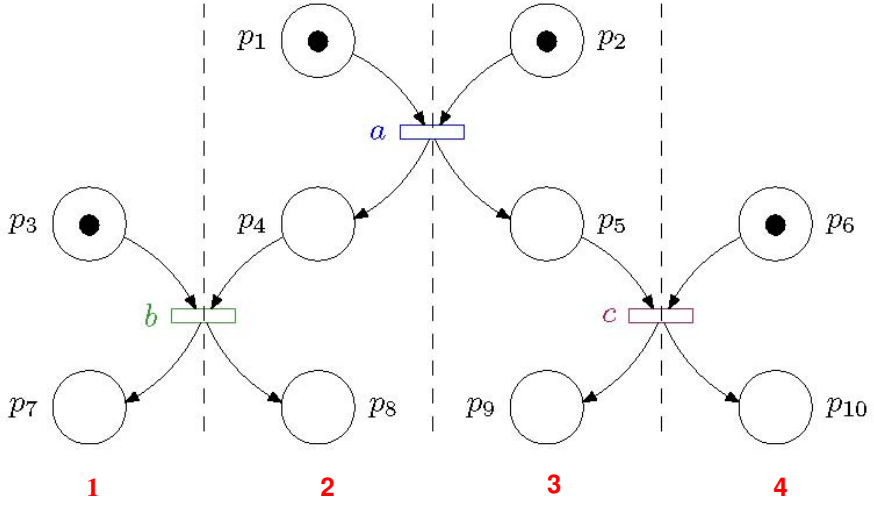
$\preceq_{ns}$: requires no synchronization

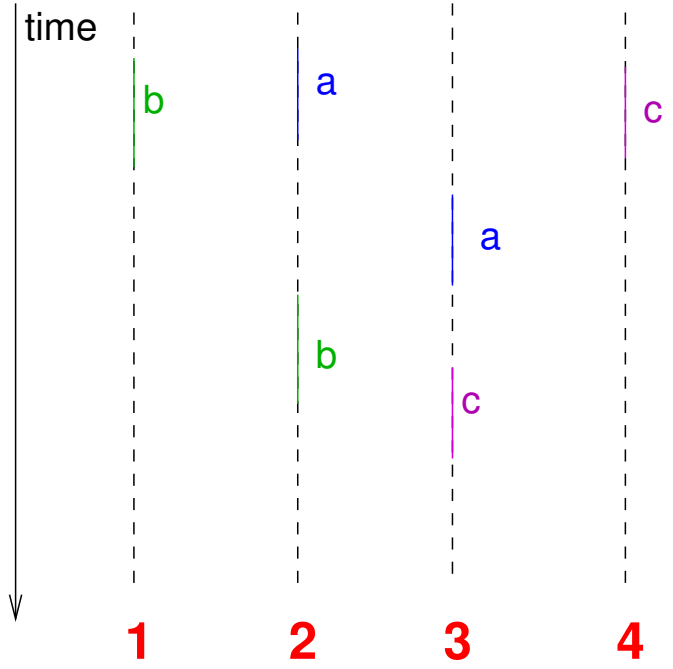


# Illustrating some Implementation relations

$\preceq_{ns}$: requires no synchronization




In case of conflict: need to *control* local processes for any relation $\preceq$ (conflict resolution)

# Knowledge characterizing enabling conditions

For $\preceq$, the enabling condition $go_t^\pi$ for a local transition $t$:

1. $t$ is globally enabled (in the Petri Net sense) or already partially executed:

$$in_t = en_t^\pi \wedge \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

# Knowledge characterizing enabling conditions

For $\preceq$, the enabling condition $go_t^\pi$ for a local transition $t$:

1. $t$ is globally enabled (in the Petri Net sense) or already partially executed:

$$in_t = en_t^\pi \wedge \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

2. $t$ has no unresolved conflict:

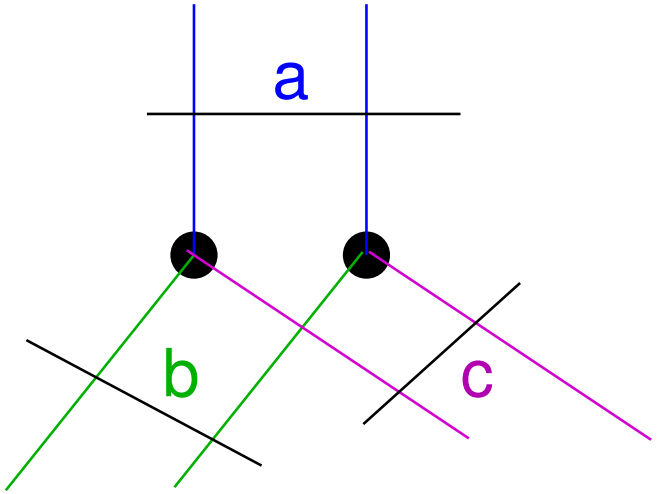
# Knowledge characterizing enabling conditions

For $\preceq$, the enabling condition $go_t^\pi$ for a local transition $t$:

1. $t$ is globally enabled (in the Petri Net sense) or already partially executed:

$$in_t = en_t^\pi \wedge \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

2. $t$ has no unresolved conflict:



# Knowledge characterizing enabling conditions

For $\preceq$, the enabling condition $go_t^\pi$ for a local transition $t$:

1. $t$ is globally enabled (in the Petri Net sense) or already partially executed:

$$in_t = en_t^\pi \wedge \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

2. $t$ has no unresolved conflict: there must exist a predicate $select_t$ such that

$$select_t \implies \Box select_t \wedge \forall t' \,.\, (t' \text{ in conflict with } t \implies \Box \neg select_{t'})$$

# Knowledge characterizing enabling conditions

For $\preceq$, the enabling condition $go_t^\pi$ for a local transition $t$:

1. $t$ is globally enabled (in the Petri Net sense) or already partially executed:

$$in_t = en_t^\pi \wedge \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

2. $t$ has no unresolved conflict: there must exist a predicate $select_t$ such that

$$select_t \implies \Box select_t \wedge \forall t' \,.\, (t' \text{ in conflict with } t \implies \Box \neg select_{t'})$$

$\pi$ must know $go_t^\pi$

# Knowledge characterizing enabling conditions

For $\preceq$, the enabling condition $go_t^\pi$ for a local transition $t$:

1. $t$ is globally enabled (in the Petri Net sense) or already partially executed:

$$in_t = en_t^\pi \wedge \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

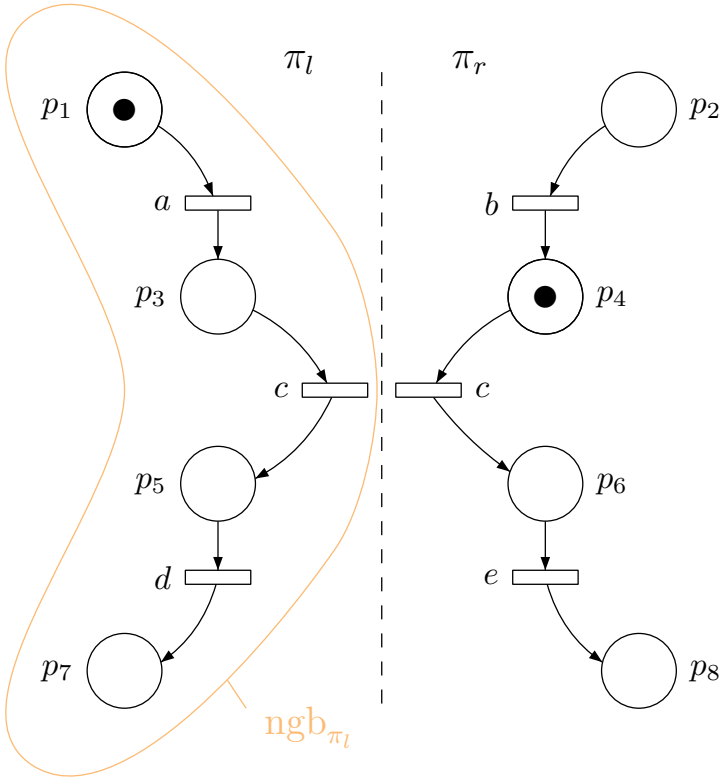
2. $t$ has no unresolved conflict: there must exist a predicate $select_t$ such that

$$select_t \implies \Box select_t \wedge \forall t' \,.\, (t' \text{ in conflict with } t \implies \Box \neg select_{t'})$$

$\pi$ must know $go_t^\pi$
and
*proc* can evaluate this knowledge property on its local state (distributed setting)

# Knowledge of the global specification preserved in a distributed setting

Is the knowledge computed on the Petri Net useful?
What may be preserved:

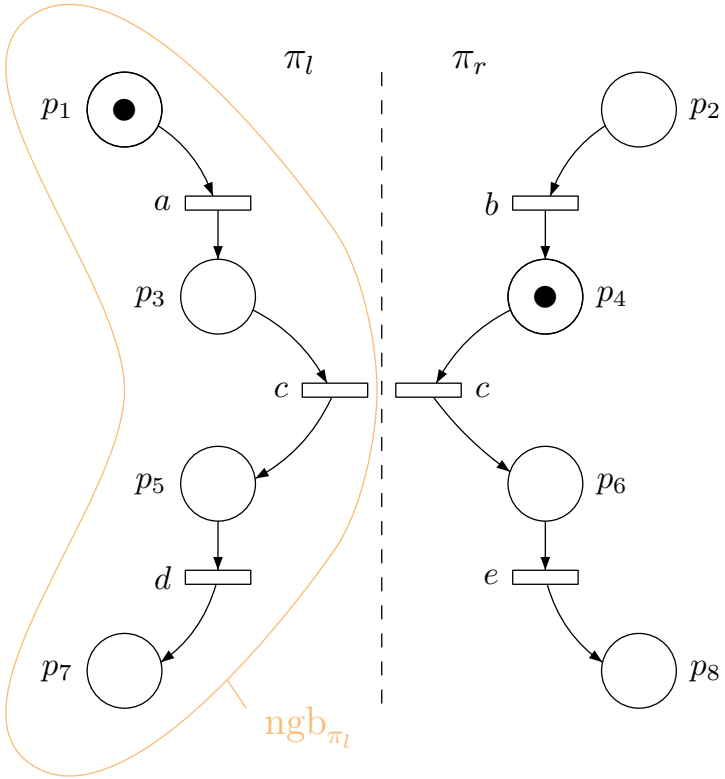



$a \ll b$ and $d \ll e$

# Knowledge of the global specification preserved in a distributed setting

Is the knowledge computed on the Petri Net useful?
What may be preserved:

- One may use the Petri Net knowledge and weaken it by taking into account the incertainty induced by $\preceq$

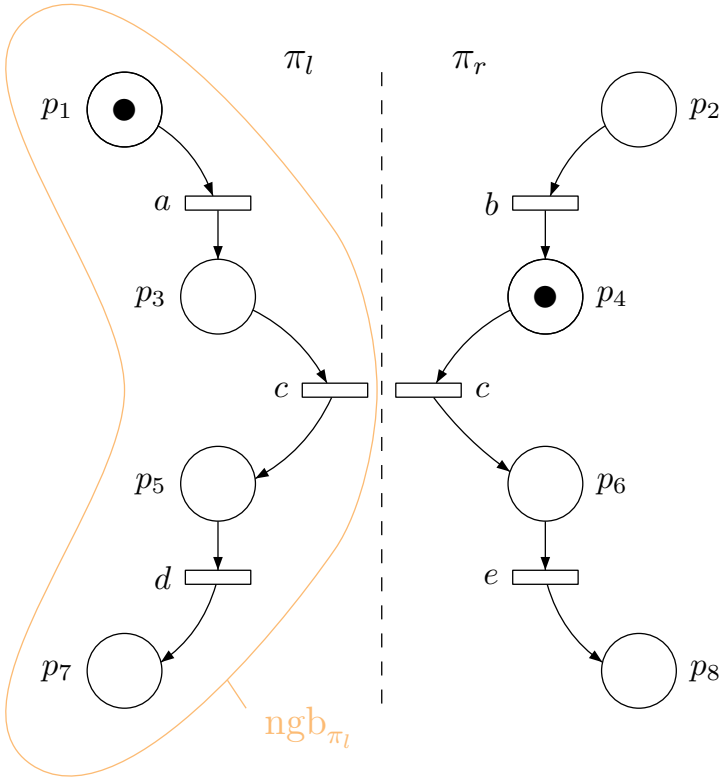



$a \ll b$ and $d \ll e$

# Knowledge of the global specification preserved in a distributed setting

Is the knowledge computed on the Petri Net useful?
What may be preserved:

- One may use the Petri Net knowledge and weaken it by taking into account the incertainty induced by $\preceq$
- (Non) enabledness of a transition in a local state

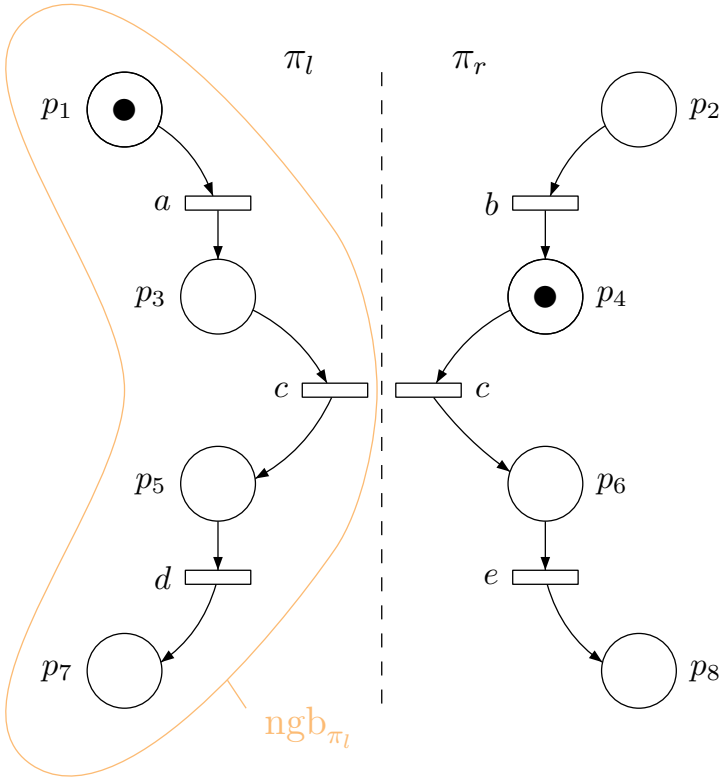



$a \ll b$ and $d \ll e$

# Knowledge of the global specification preserved in a distributed setting

Is the knowledge computed on the Petri Net useful?
What may be preserved:

- One may use the Petri Net knowledge and weaken it by taking into account the incertainty induced by $\preceq$
- (Non) enabledness of a transition in a local state
- Not preserved: knowledge for achieving synchronization

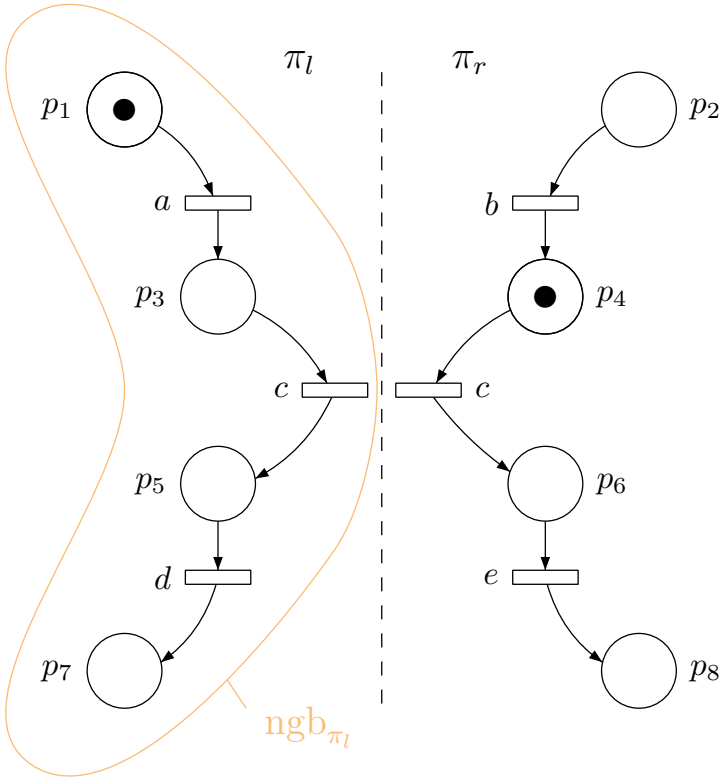



$a \ll b$ and $d \ll e$

# Knowledge of the global specification preserved in a distributed setting

Is the knowledge computed on the Petri Net useful?
What may be preserved:

- One may use the Petri Net knowledge and weaken it by taking into account the incertainty induced by $\preceq$
- (Non) enabledness of a transition in a local state
- Not preserved: knowledge for achieving synchronization

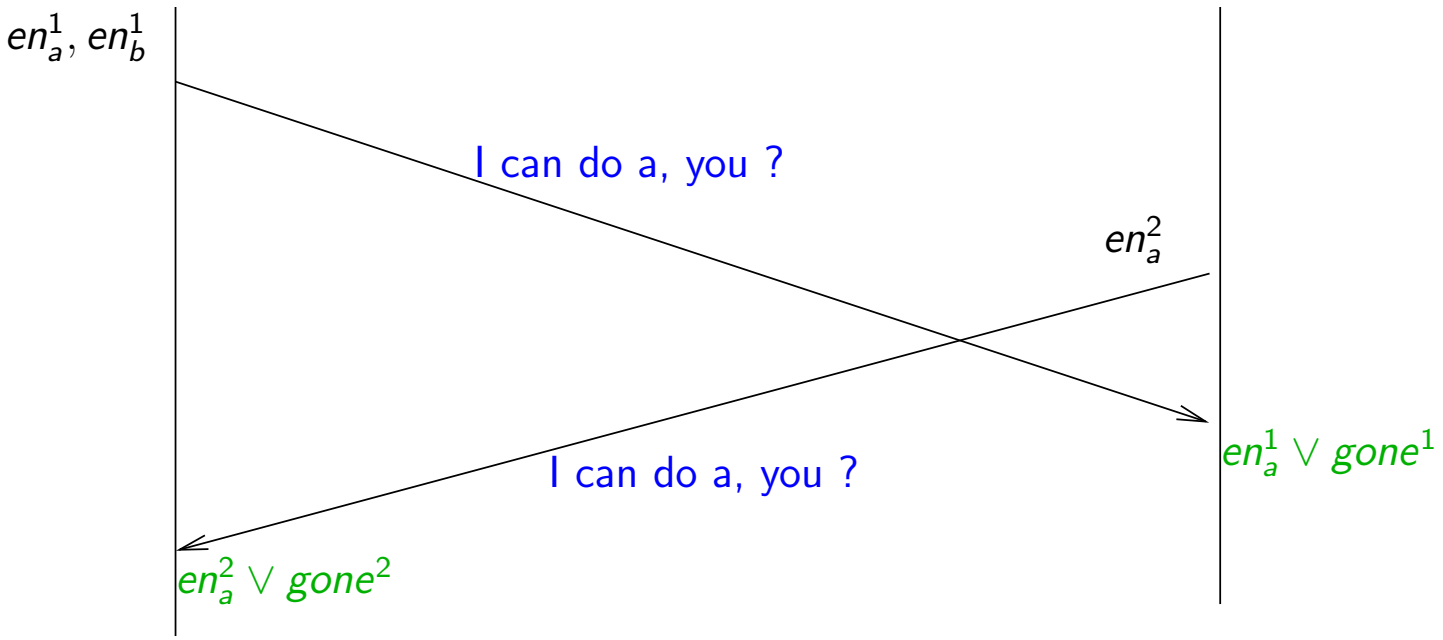



$a \ll b$ and $d \ll e$

**Fact**: to achieve synchronization, to resolve conflicts one must communicate

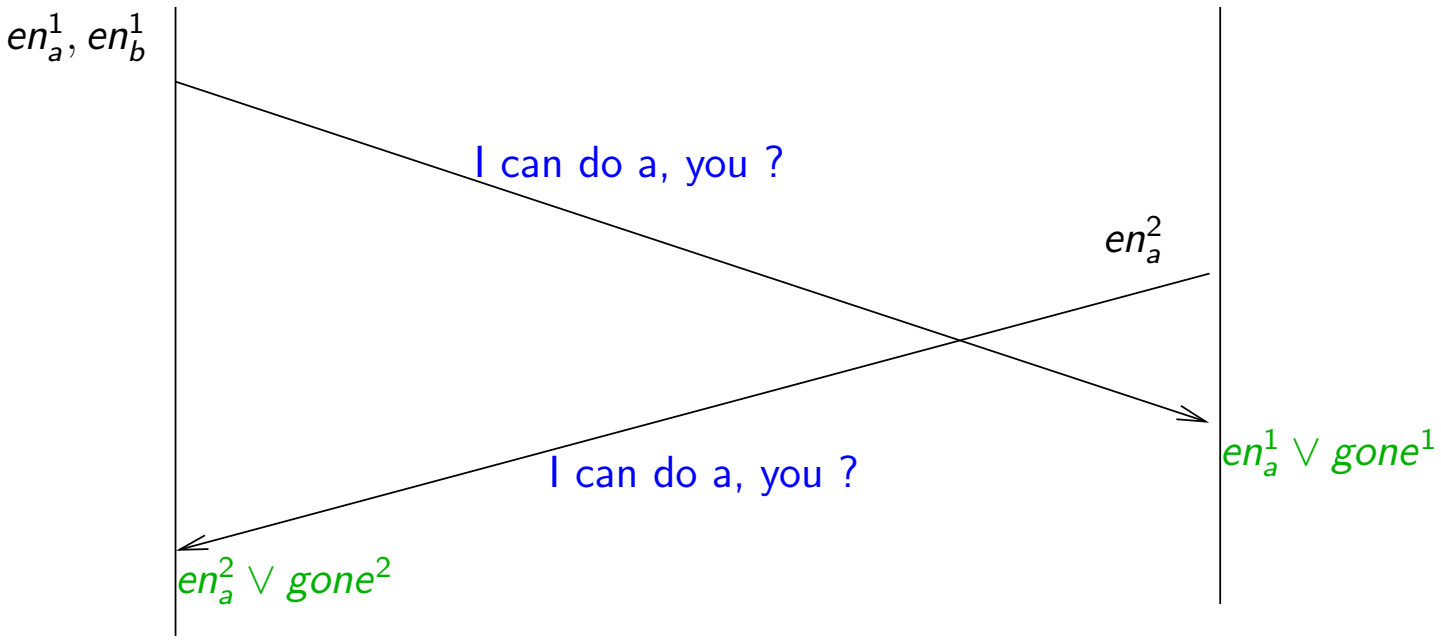
# Knowledge through communication

A typical protocol for achieving distributed implementation:



# Knowledge through communication

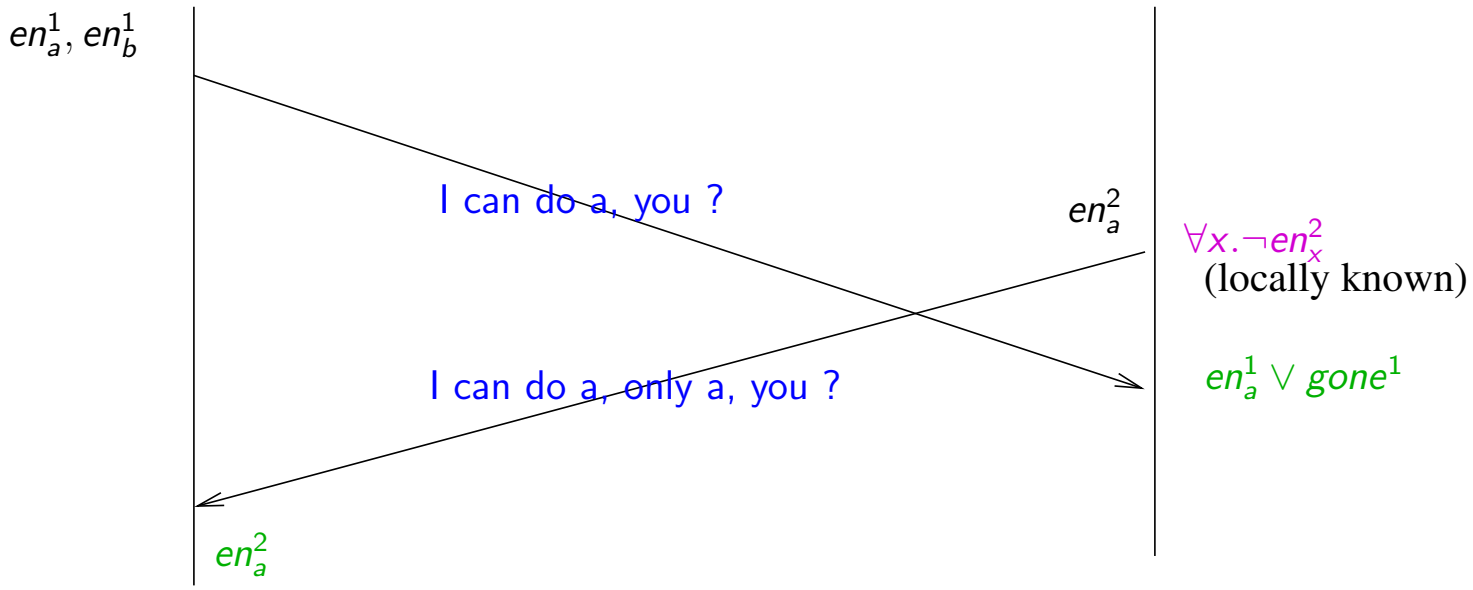
A typical protocol for achieving distributed implementation:




- No useful information gained – neither in case of a positive nor a negative response (I can do a is potentially non persistent information)

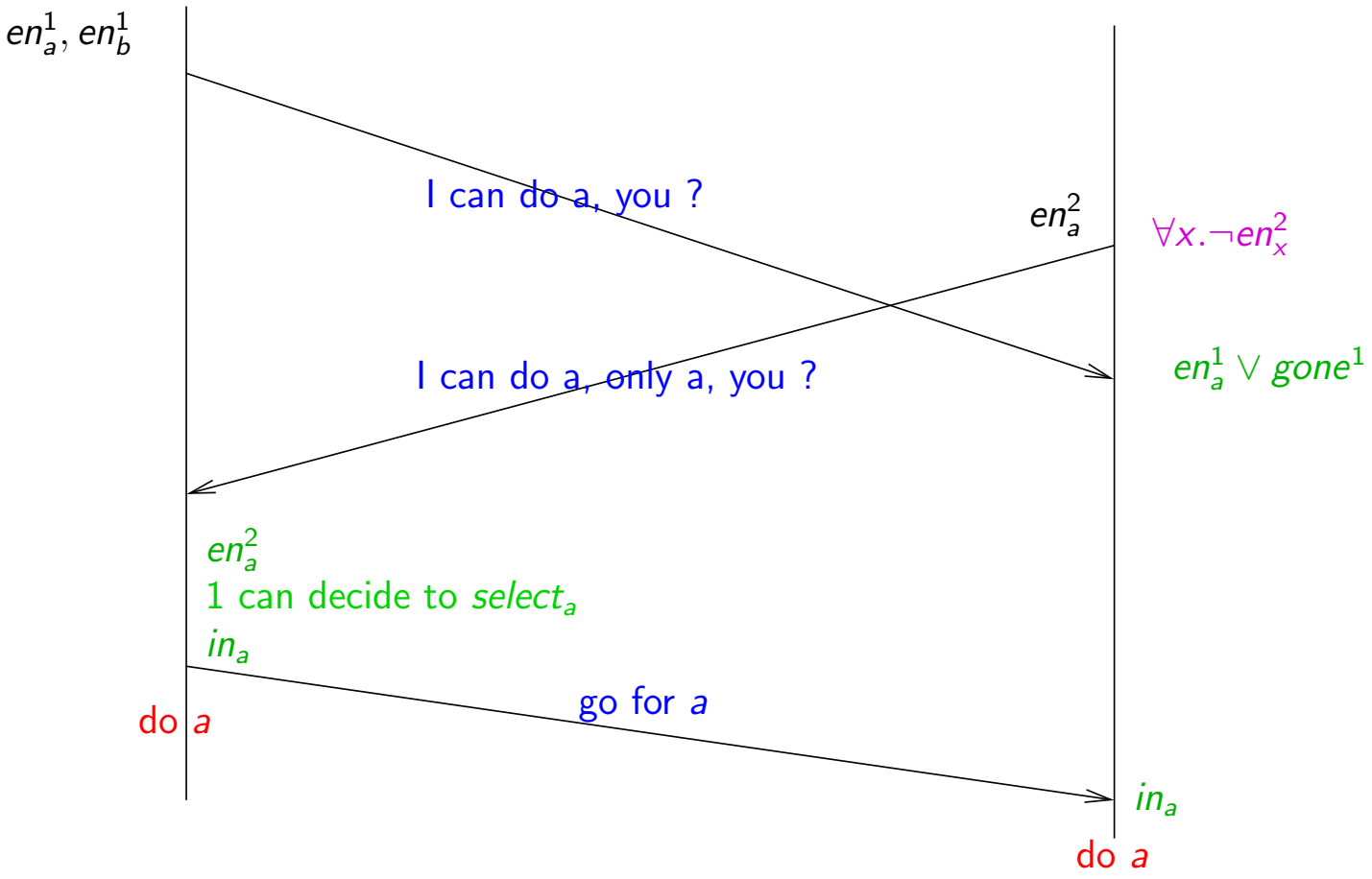
# Knowledge through communication

A typical protocol for achieving distributed implementation:



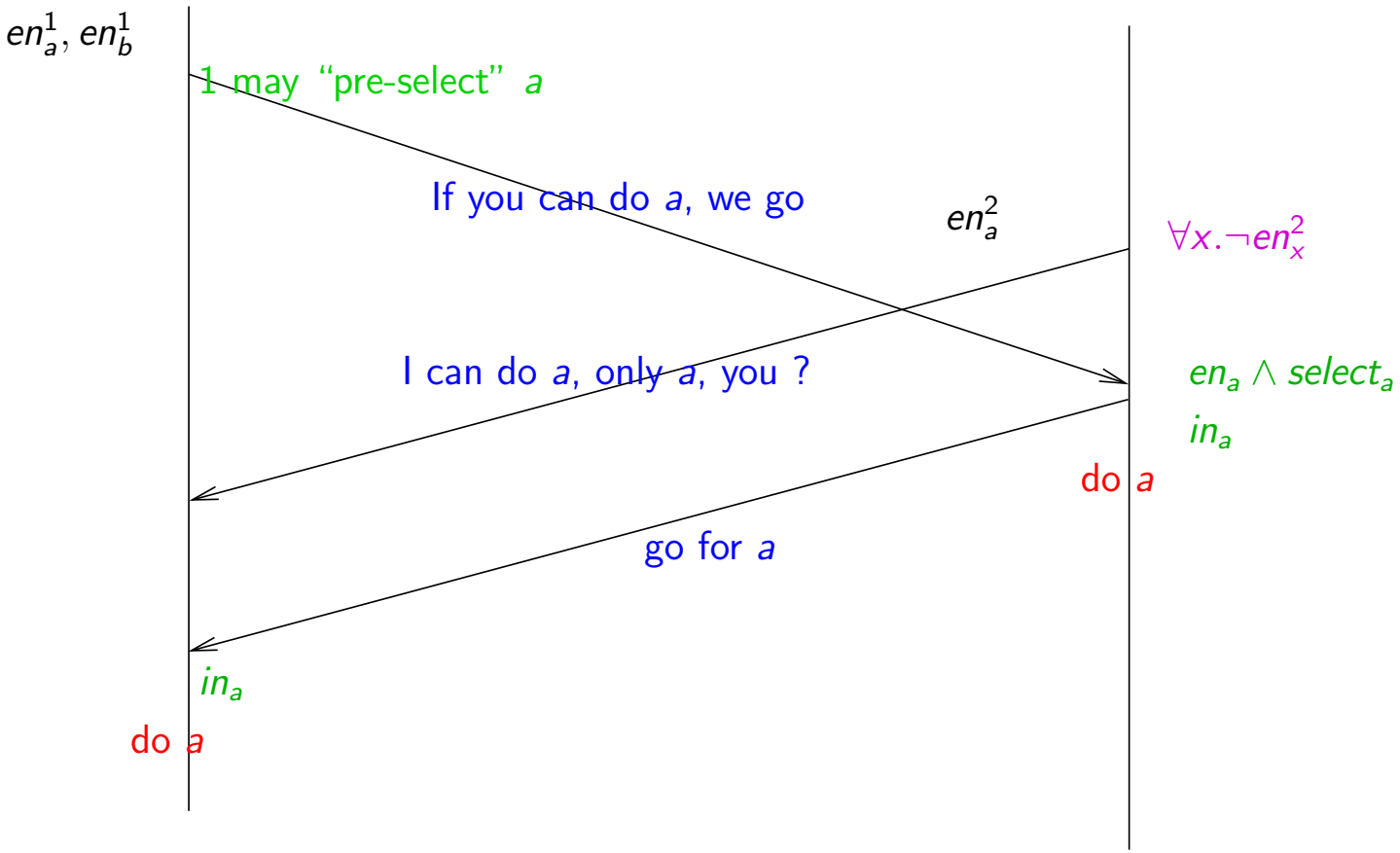

- process 1 can now decide to set $select_a$ (I can only do a is stable information for the synchronization partner)

# Knowledge through communication



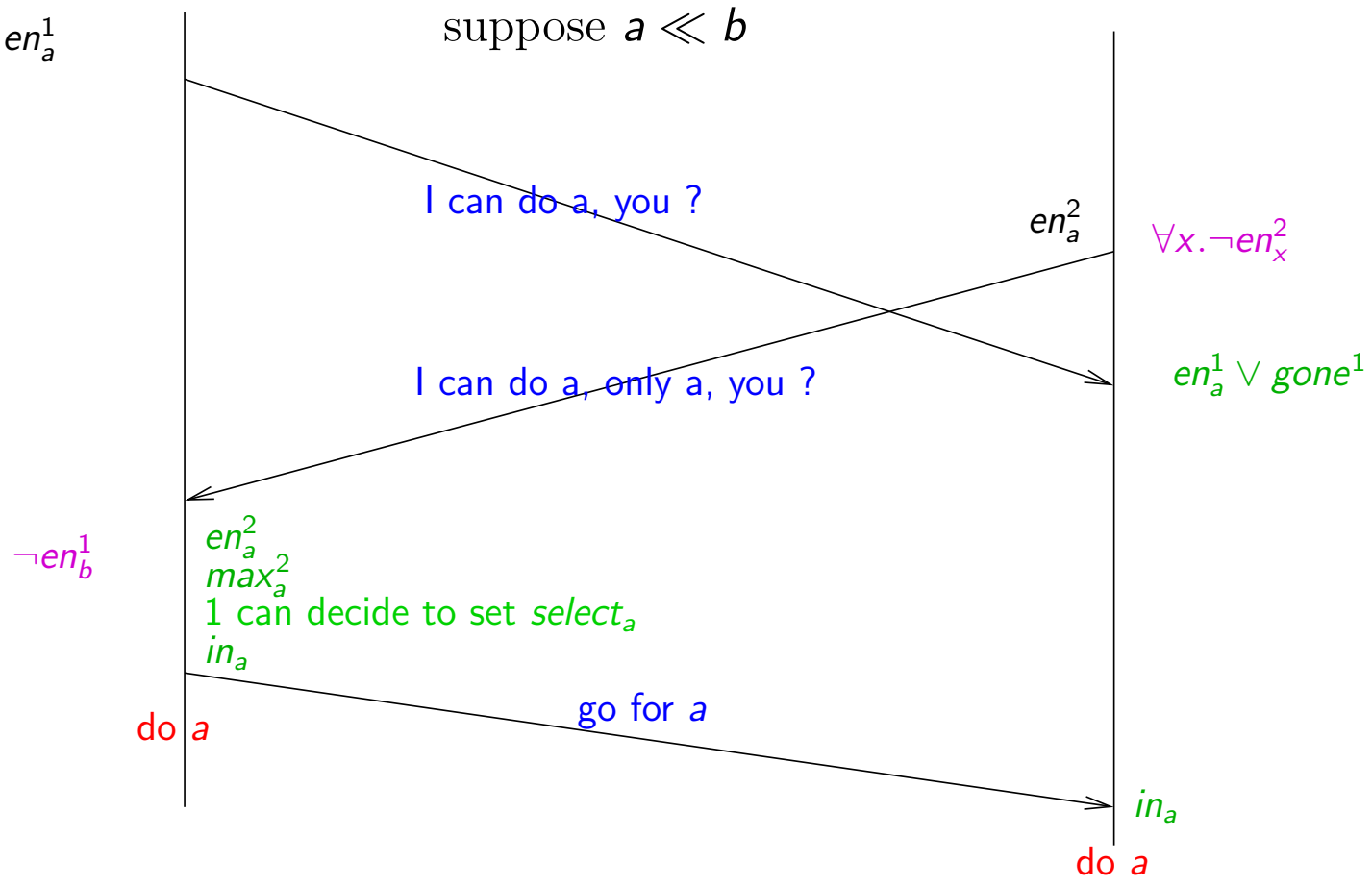

- conveye information providing stronger knowledge — when possible (e.g. information about absence of conflict)

# Knowledge through communication




- try to resolve conflicts early

# Knowledge through communication




- combine static and dynamic knowledge: avoid requesting knowledge that is statically available

# Discussion

The explicit use of knowledge is useful for reasoning about the distribution of centralized specifications. It provides a generic, framework independent way for

- optimizing existing and developing new distribution algorithms
- providing correctness proofs for them due to separation of concerns:
  - characterize the required implementation relation as a set of (knowledge) properties
  - prove that the proposed protocol guarantees them

## Perspectives

- take into account data, timing, ... (discrete and continuous)
- formulate the platform characteristics (communication primitives) in terms of knowledge
- devise modular proofs for distribution strategies