Natural Specifications Yield Decidability for Distributed Synthesis of Asynchronous Systems

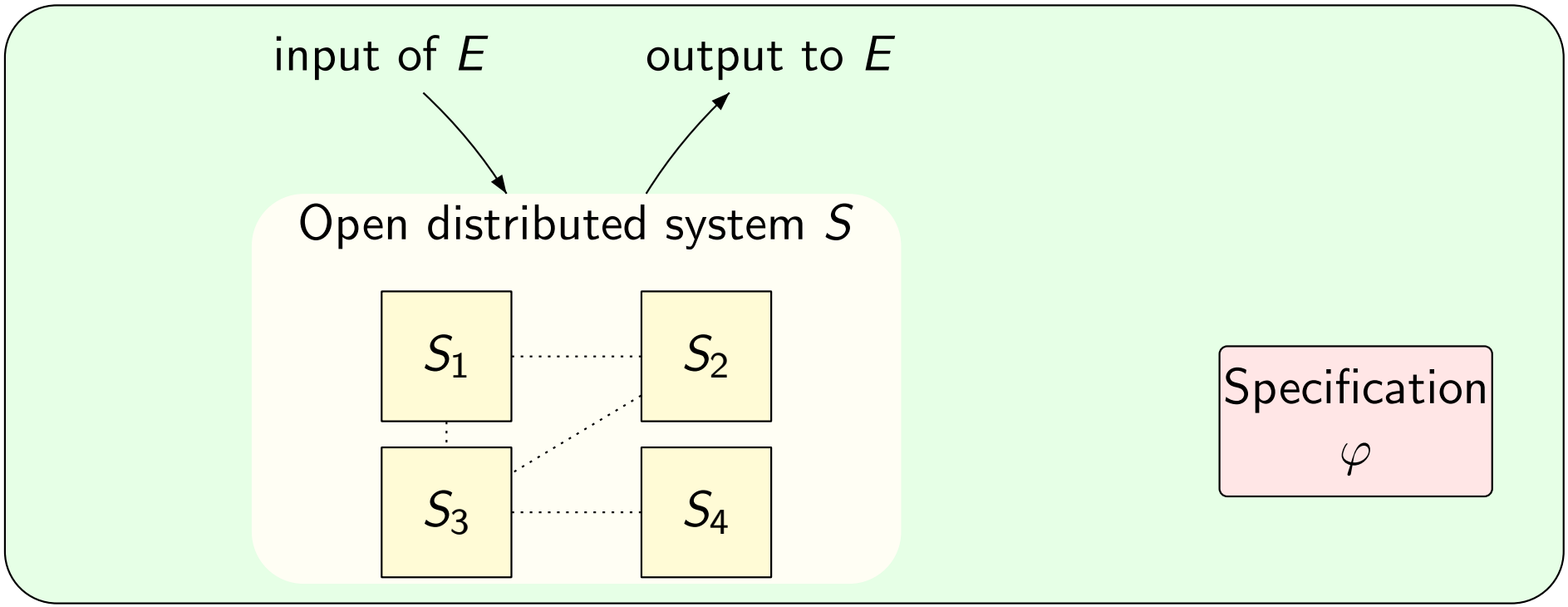
Nathalie Sznajder

LSV, ENS Cachan, France

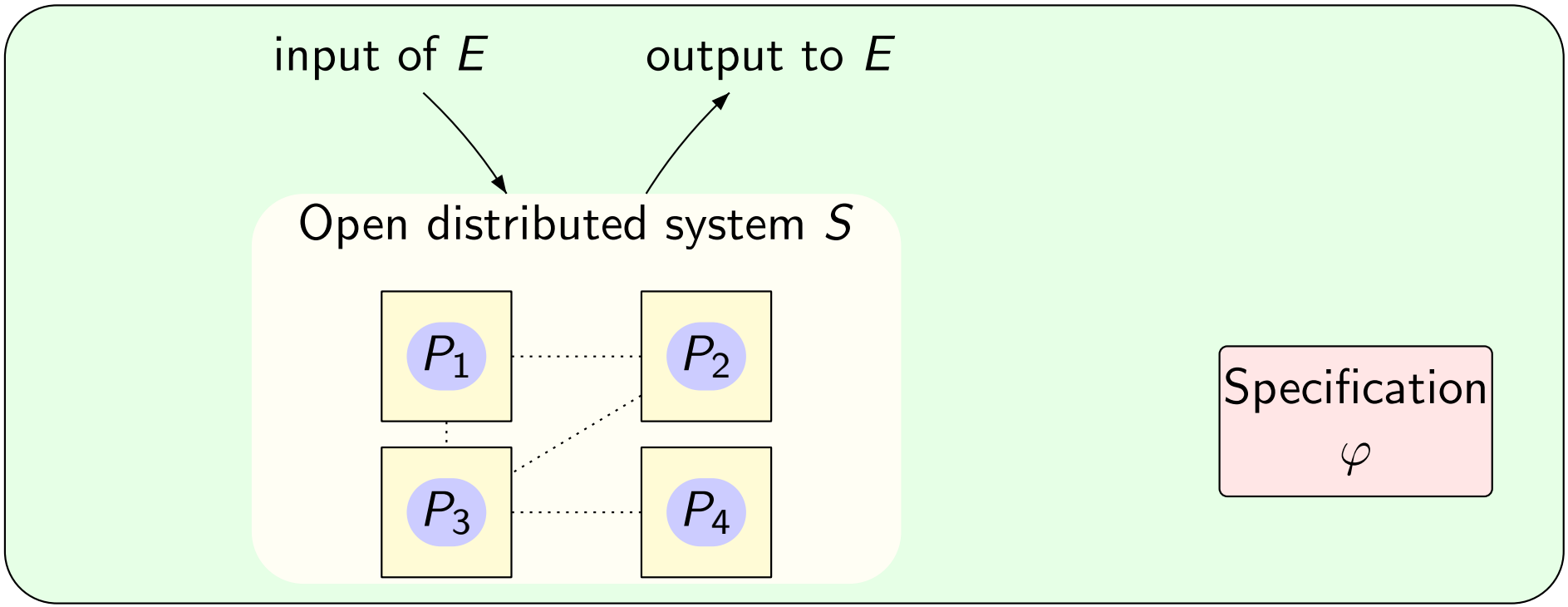
SOFSEM'09
Czech Republic

Joint work with Thomas Chatain and Paul Gastin

Synthesis of a distributed reactive system



Synthesis of a distributed reactive system



Two problems

- Decide the existence of a distributed program such that their joint behavior $P_1||P_2||P_3||P_4||E$ satisfies φ, for all E.
- Synthesis : If it exists, compute such a distributed program.

Distributed synthesis: Synchronous or asynchronous semantics?

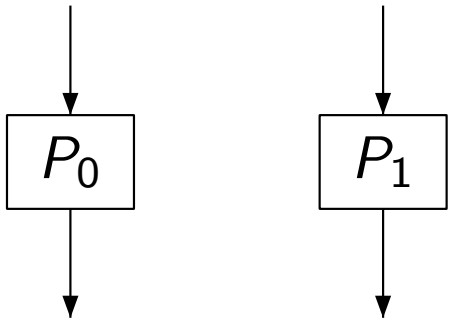
Synchronous semantics

- At each tick of a global clock, all the processes and the environment output their new value
- Introduced in [PnueliRosner90].
- In general undecidable.

Distributed synthesis: Synchronous or asynchronous semantics?

Synchronous semantics

- At each tick of a global clock, all the processes and the environment output their new value
- Introduced in [PnueliRosner90].
- In general undecidable.



Distributed synthesis: Synchronous or asynchronous semantics?

Synchronous semantics

- At each tick of a global clock, all the processes and the environment output their new value
- Introduced in [PnueliRosner90].
- In general undecidable.

Asynchronous semantics

Our model

- Processes evolve asynchronously for local actions (i.e., communications with the environment)

Asynchronous semantics

Our model

- Processes evolve asynchronously for local actions (i.e., communications with the environment)
- They can synchronize by signals = common actions initiated by only one process. A process cannot refuse reception of a signal.

Asynchronous semantics

Our model

- Processes evolve asynchronously for local actions (i.e., communications with the environment)
- They can synchronize by signals = common actions initiated by only one process. A process cannot refuse reception of a signal.
- Specifications :
  - over partial orders

Asynchronous semantics

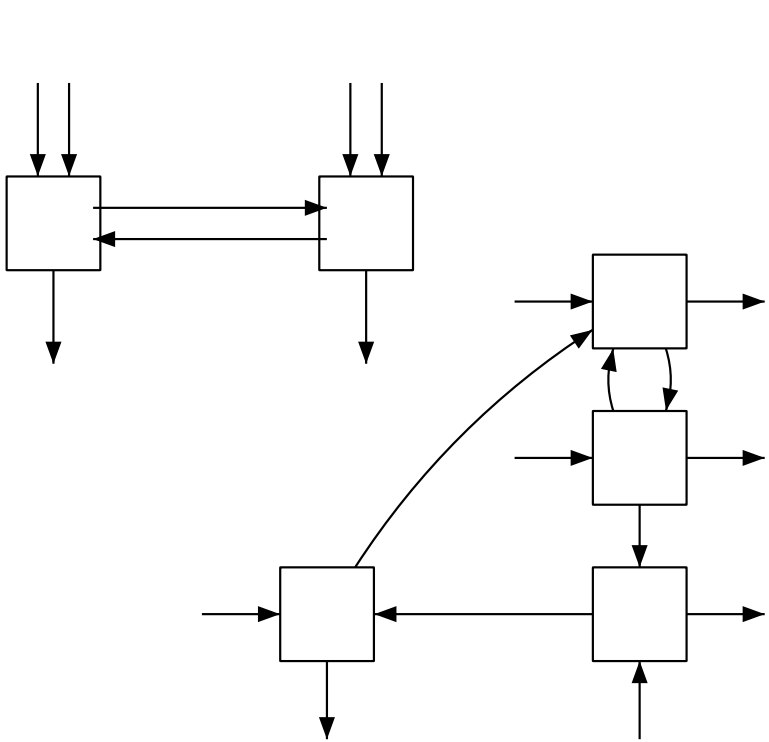
Our model

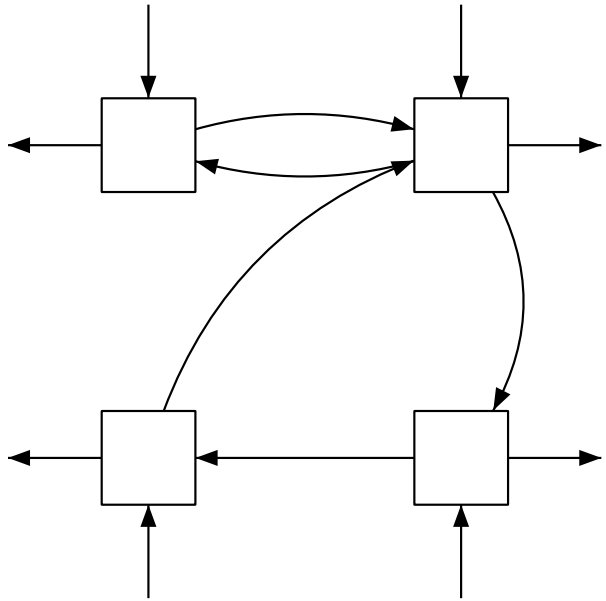
- Processes evolve asynchronously for local actions (i.e., communications with the environment)
- They can synchronize by signals = common actions initiated by only one process. A process cannot refuse reception of a signal.
- Specifications :
  - over partial orders
  - will not restrain communication abilities

Decidability Results

Theorem

Synthesis problem is decidable for strongly-connected architectures





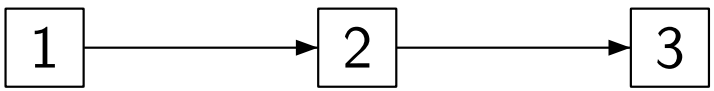
Outline

Outline

The model

Architectures

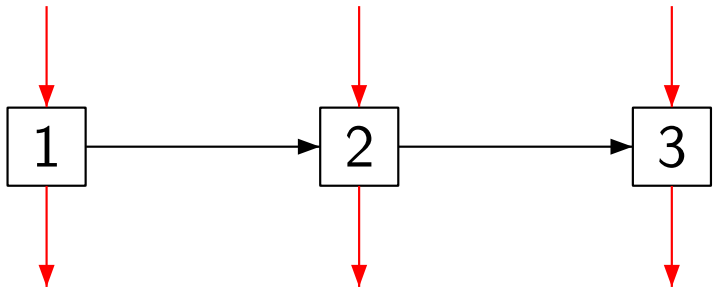
- Communication graph $(Proc, E)$



The model

Architectures

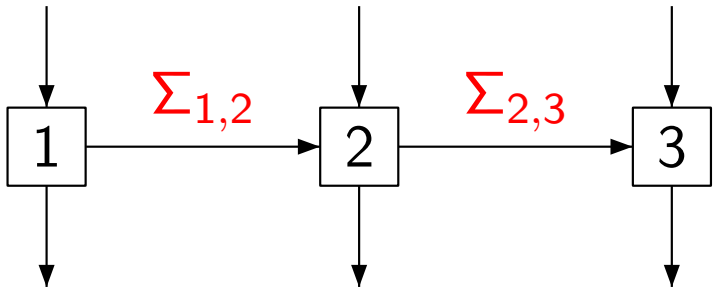
- Communication graph $(Proc, E)$
- Sets of input and output signals for each process :
  $\bigcup_{i \in Proc} \mathrm{In}_i \cup \bigcup_{i \in Proc} \mathrm{Out}_i = \Gamma$



The model

Architectures

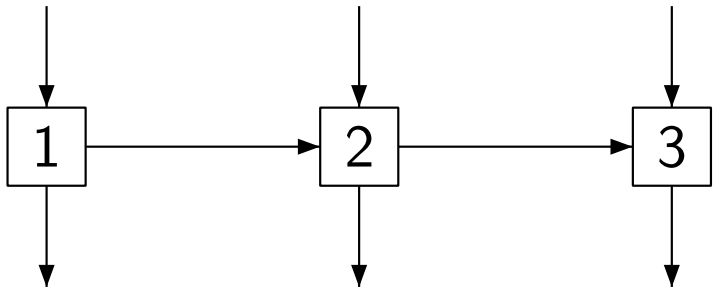
- Communication graph $(Proc, E)$
- Sets of input and output signals for each process : $\bigcup_{i \in Proc} \mathrm{In}_i \cup \bigcup_{i \in Proc} \mathrm{Out}_i = \Gamma$
- Processes choose sets $\Sigma_{i,j}$ for $i, j \in \mathrm{Proc}$



The model

Architectures

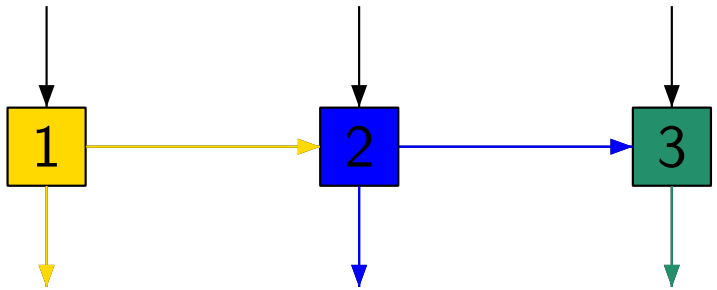
- Communication graph $(Proc, E)$
- Sets of input and output signals for each process : $\bigcup_{i \in Proc} \mathrm{In}_i \cup \bigcup_{i \in Proc} \mathrm{Out}_i = \Gamma$
- Processes choose sets $\Sigma_{i,j}$ for $i, j \in \mathrm{Proc}$
- $\Sigma = \Gamma \cup \bigcup_{(i,j) \in E} \Sigma_{i,j}$



The model

Architectures

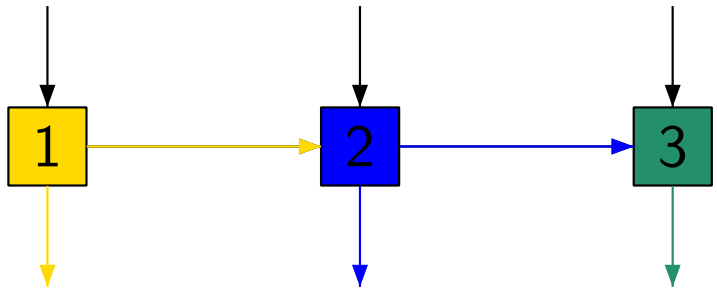
- Communication graph $(Proc, E)$
- Sets of input and output signals for each process : $\bigcup_{i \in Proc} \mathrm{In}_i \cup \bigcup_{i \in Proc} \mathrm{Out}_i = \Gamma$
- Processes choose sets $\Sigma_{i,j}$ for $i, j \in \mathrm{Proc}$
- $\Sigma = \Gamma \cup \bigcup_{(i,j) \in E} \Sigma_{i,j}$
- For each process i, Σ_i is the set of signals it can send or receive, and $\Sigma_i^c = \mathrm{Out}_i \cup \bigcup_{j,(i,j) \in E} \Sigma_{i,j}$

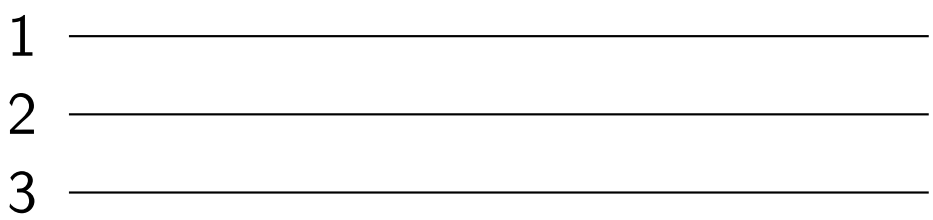


The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b

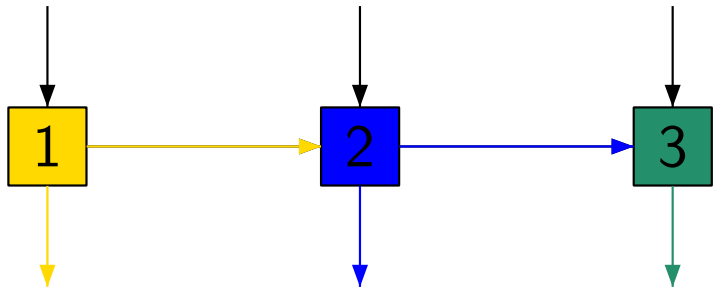


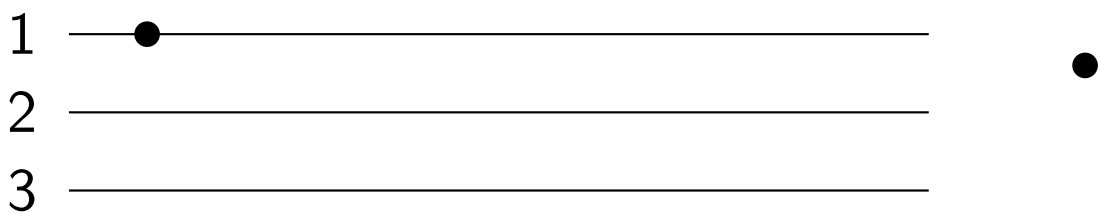


The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b

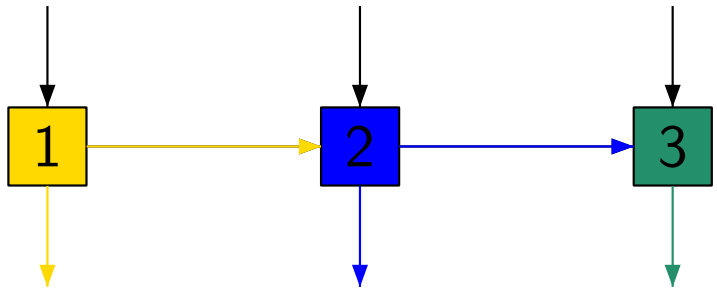


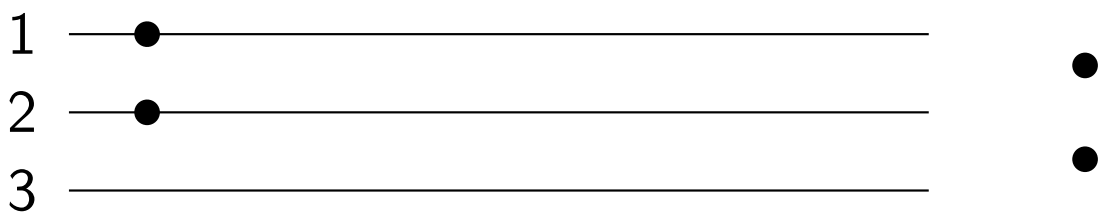


The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b

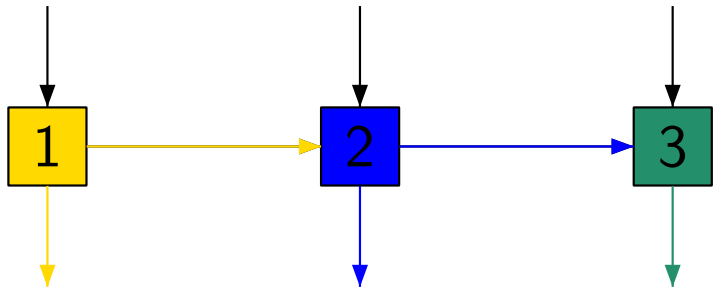


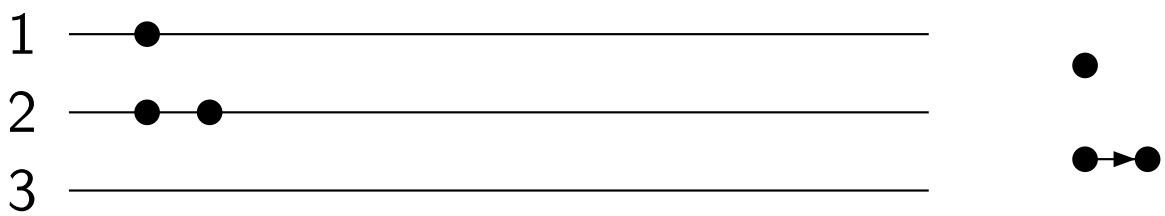


The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b

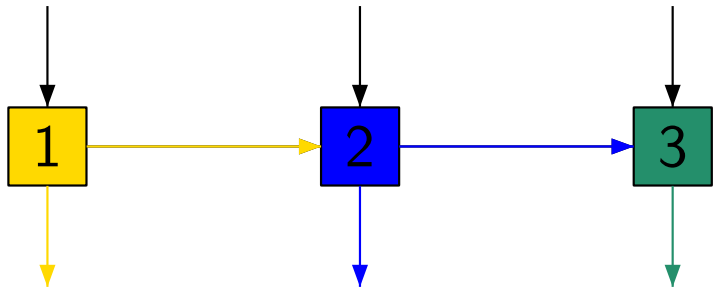


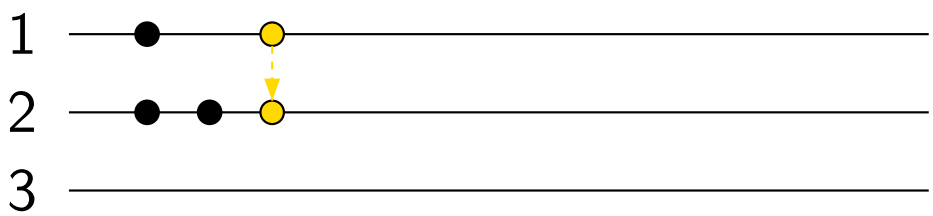


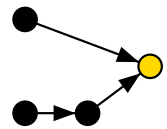
The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b



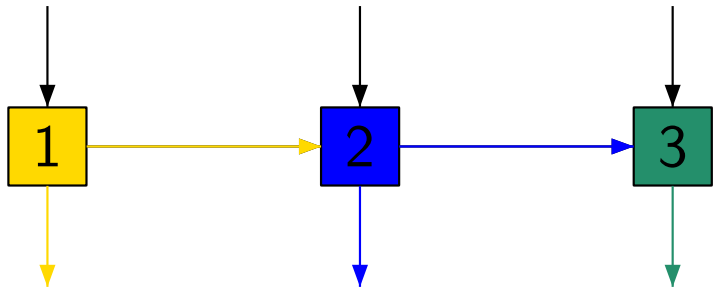


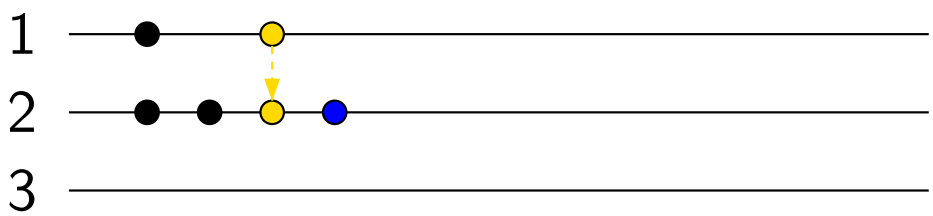


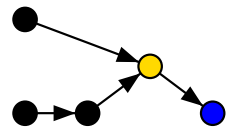
The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b



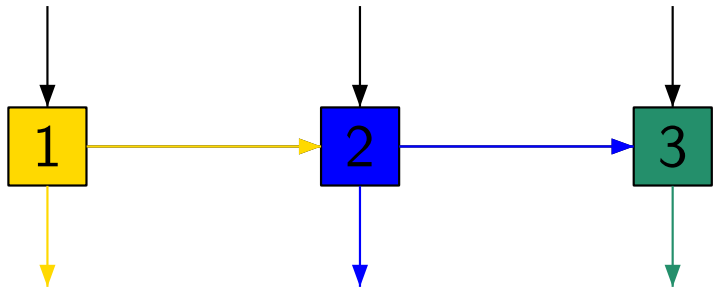


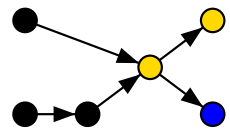


The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b

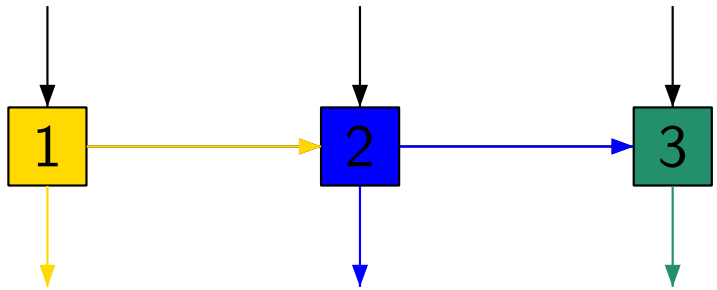


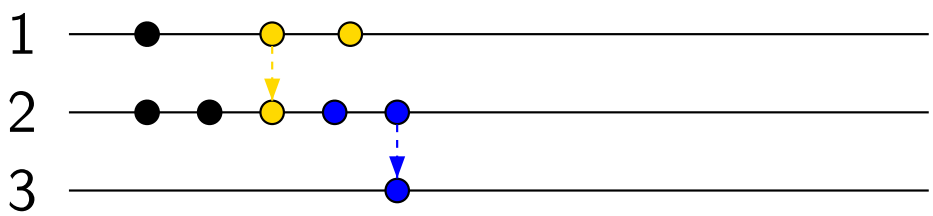


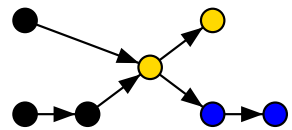
The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b



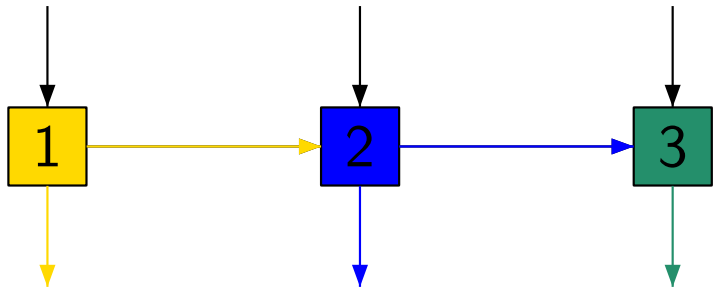


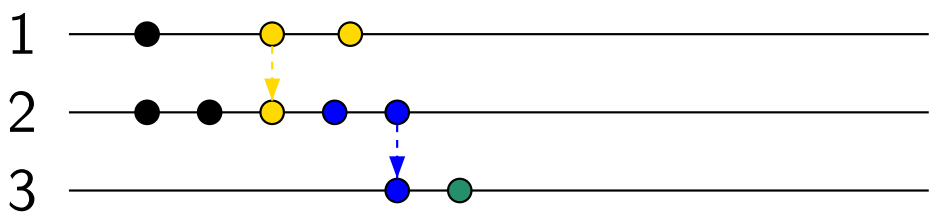


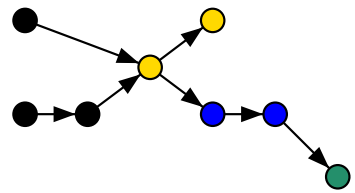
The model: runs

Runs

A run is a Mazurkiewicz trace $t = (V, \lambda, \leq)$ over (Σ, D) where aDb if there is a process that takes part both in a and b







The model

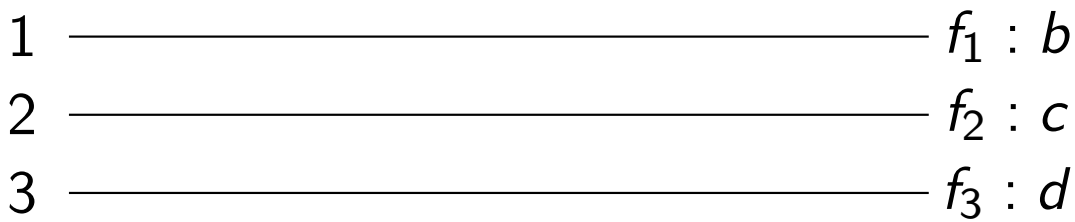
Strategies

- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory

The model

Strategies

- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory



The model

Strategies

- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory

The model

Strategies

- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory

The model

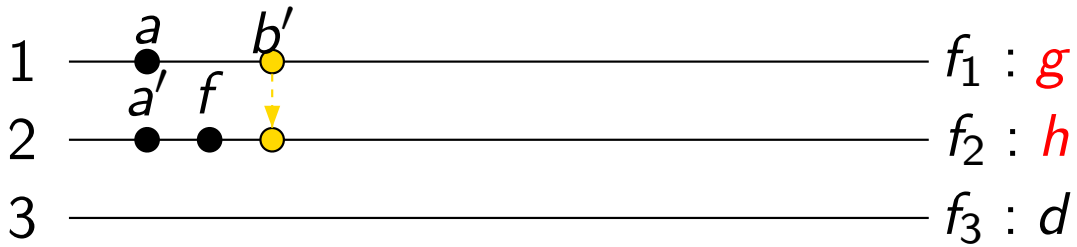
Strategies

- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory

The model

Strategies

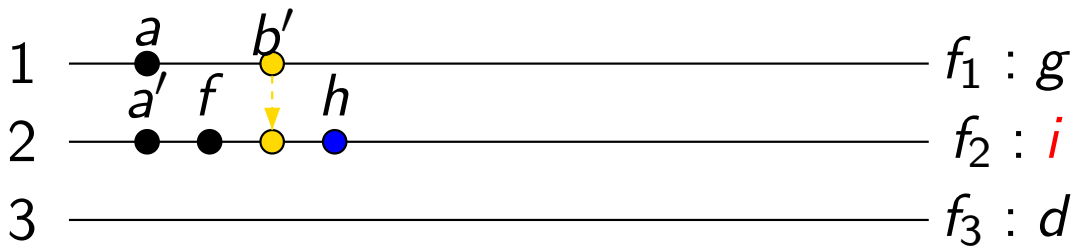
- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory



The model

Strategies

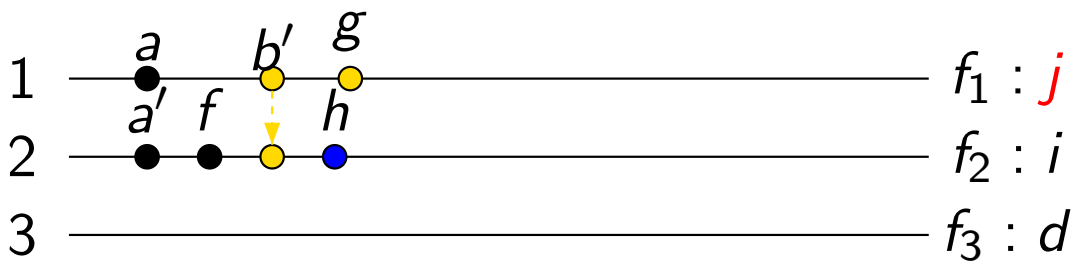
- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory



The model

Strategies

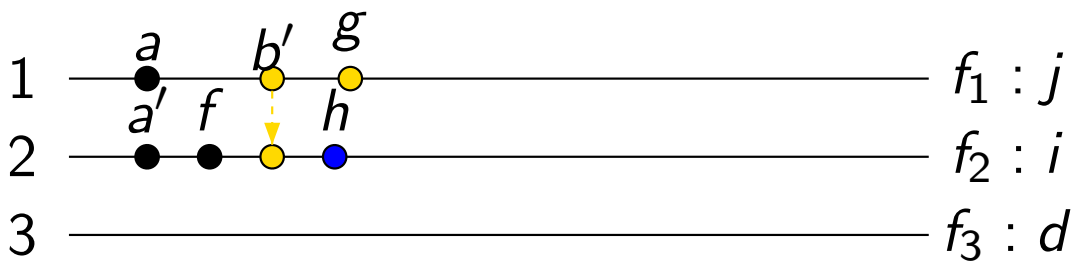
- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory



The model

Strategies

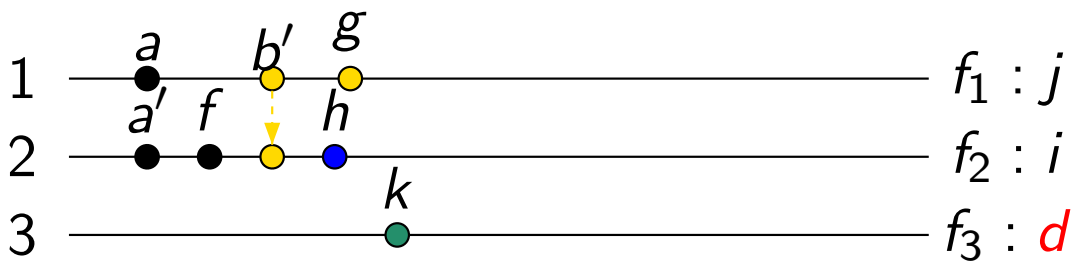
- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory
- A run respects a strategy $(f_i)_{i \in \mathrm{Proc}}$ (is a $(f_i)_{i \in \mathrm{Proc}}$-run) if each controllable action of process i is labelled according to the strategy f_i



The model

Strategies

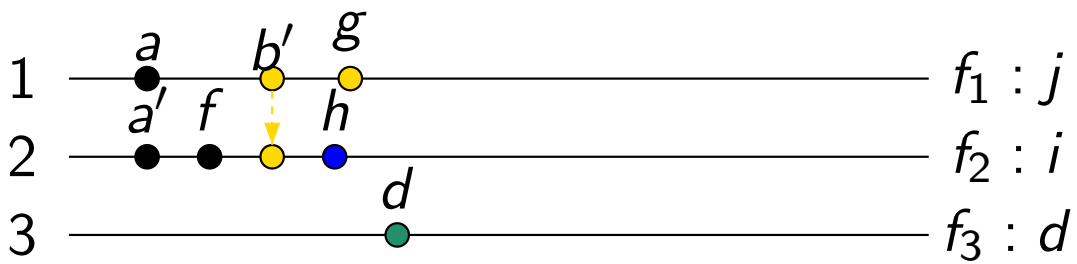
- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory
- A run respects a strategy $(f_i)_{i \in \mathrm{Proc}}$ (is a $(f_i)_{i \in \mathrm{Proc}}$-run) if each controllable action of process i is labelled according to the strategy f_i



The model

Strategies

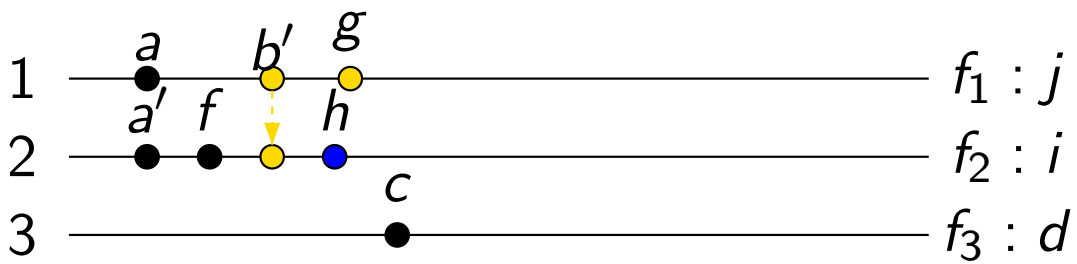
- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory
- A run respects a strategy $(f_i)_{i \in \mathrm{Proc}}$ (is a $(f_i)_{i \in \mathrm{Proc}}$-run) if each controllable action of process i is labelled according to the strategy f_i



The model

Strategies

- Strategies are functions $f_i : \Sigma_i^* \rightarrow \Sigma_i^c$ with local memory
- A run respects a strategy $(f_i)_{i \in \mathrm{Proc}}$ (is a $(f_i)_{i \in \mathrm{Proc}}$-run) if each controllable action of process i is labelled according to the strategy f_i



The model

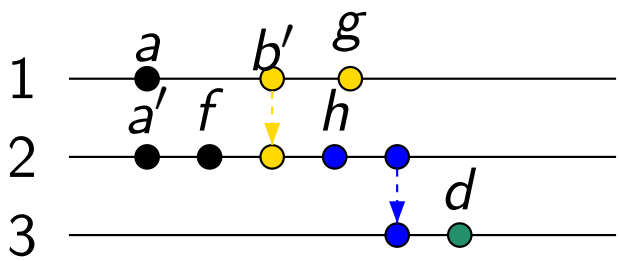
Observable runs

Given a run $t = (V, \lambda, \leq)$, we define the observable run by $\pi_\Gamma(t) = (\Gamma, \lambda_{|\Gamma}, \leq \cap(\Gamma \times \Gamma))$

The model

Observable runs

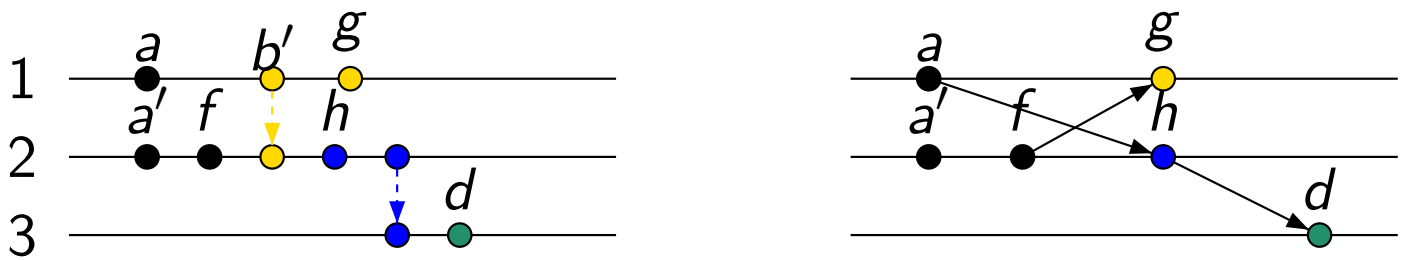
Given a run $t = (V, \lambda, \leq)$, we define the observable run by $\pi_\Gamma(t) = (\Gamma, \lambda_{|\Gamma}, \leq \cap(\Gamma \times \Gamma))$



The model

Observable runs

Given a run $t = (V, \lambda, \leq)$, we define the observable run by $\pi_\Gamma(t) = (\Gamma, \lambda_{|\Gamma}, \leq \cap(\Gamma \times \Gamma))$



The synthesis problem

Given

- $\mathcal{A} = (\mathrm{Proc}, E, \Gamma)$
- φ a specification over Γ-labelled partial orders (observable runs)

Do there exist

- sets $\Sigma_{i,j}$ for each $(i,j) \in E$
- and strategies $f_i : \Sigma_i^* \to \Sigma_i^c$ for each $i \in \mathrm{Proc}$

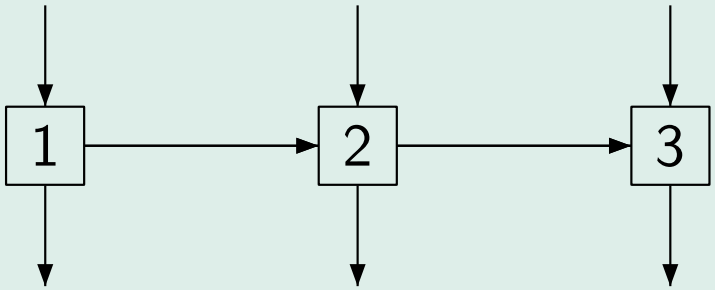
such that every $(f_i)_{i \in \mathrm{Proc}}$-run t is such that $\pi_\Gamma(t) \models \varphi$? If so, compute them

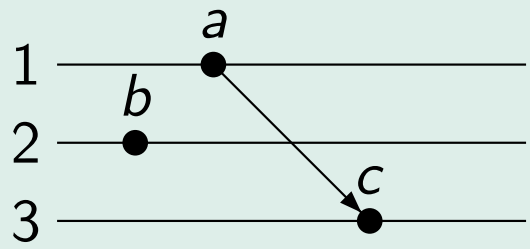
Specifications

Communication induces order relation

Specifications

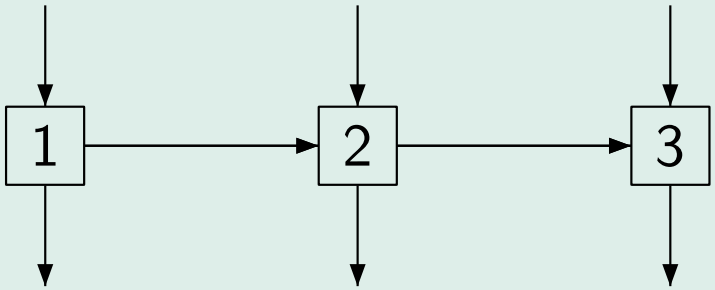
Communication induces order relation

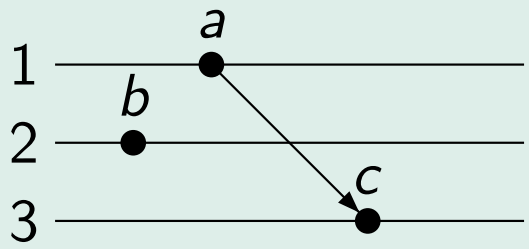


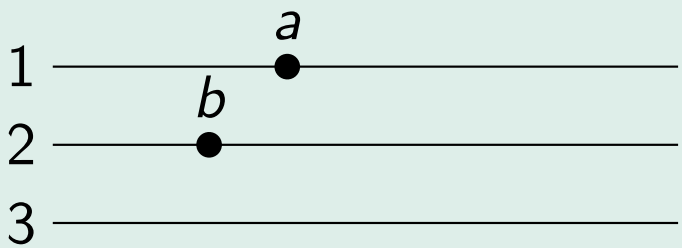


Specifications

Communication induces order relation

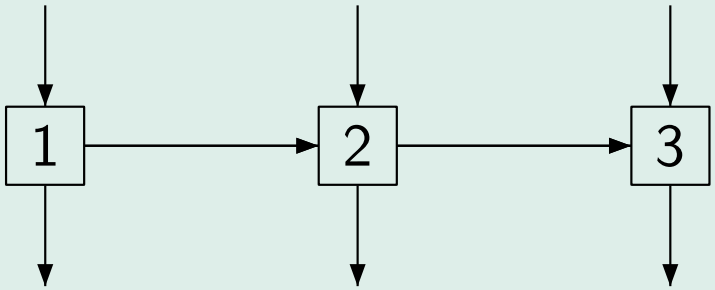


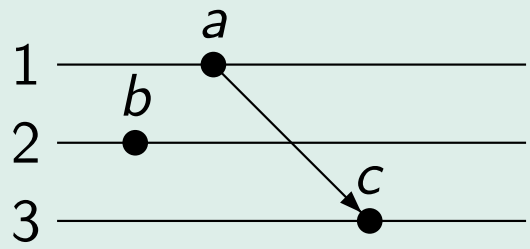


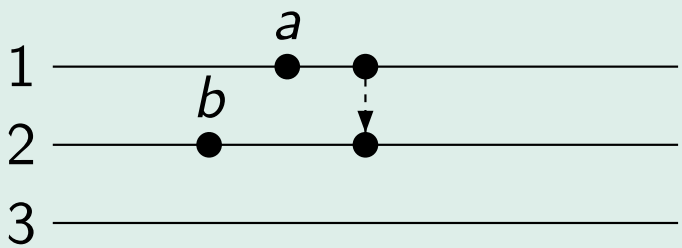


Specifications

Communication induces order relation

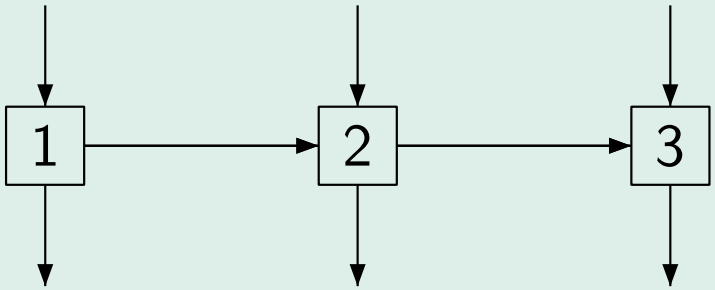


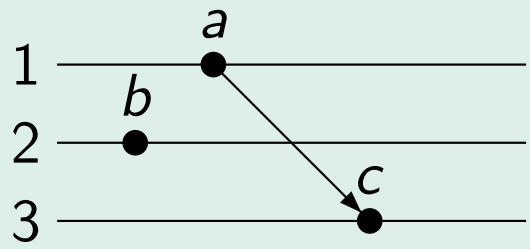


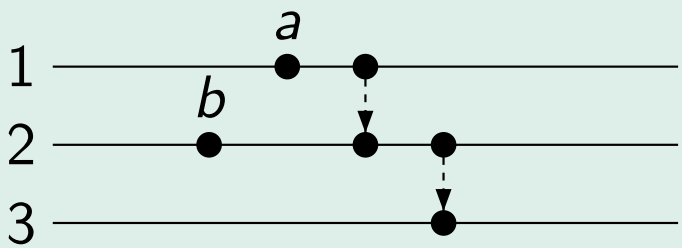


Specifications

Communication induces order relation

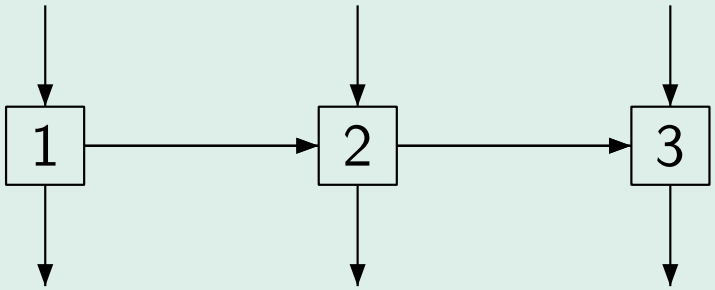


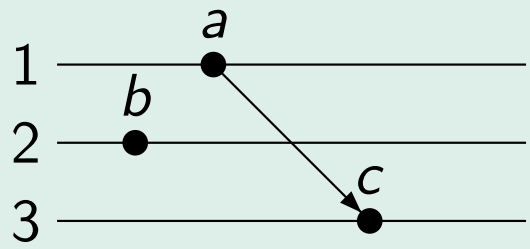


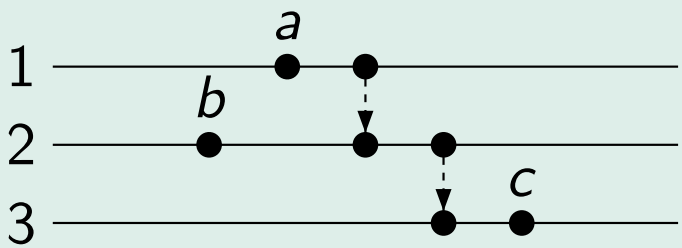


Specifications

Communication induces order relation

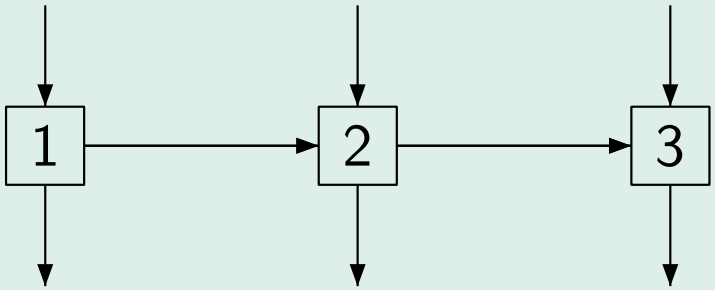


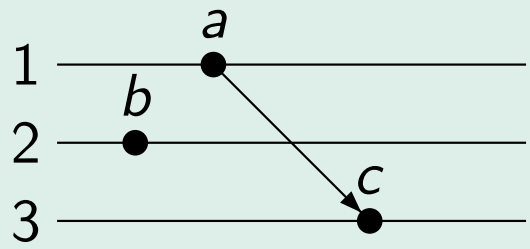


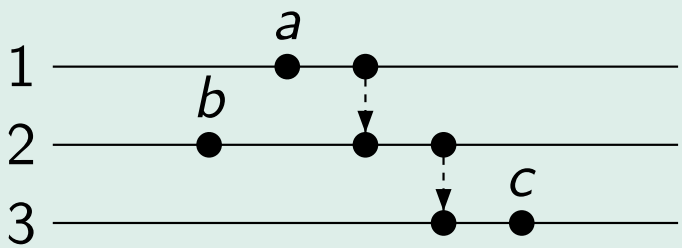


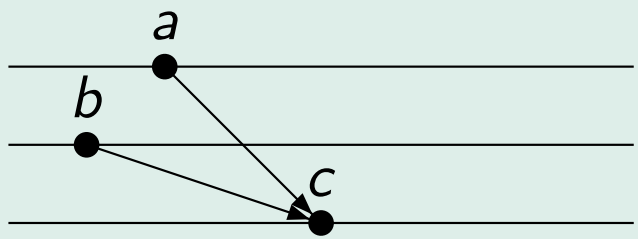
Specifications

Communication induces order relation



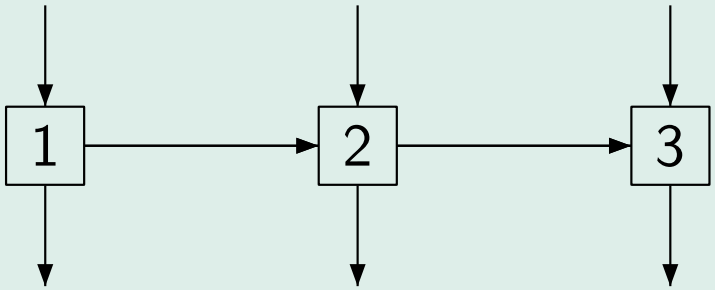


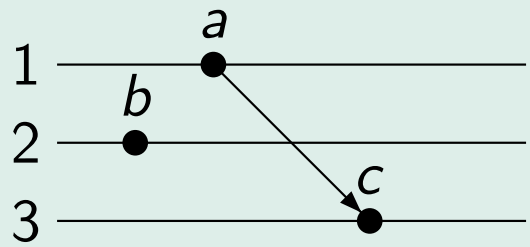


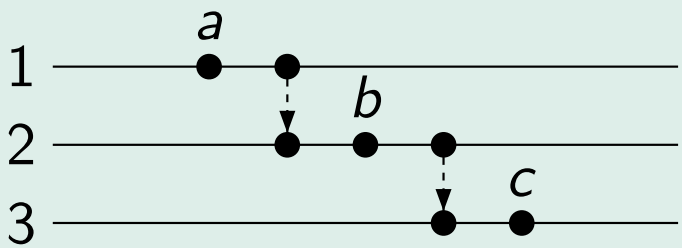


Specifications

Communication induces order relation

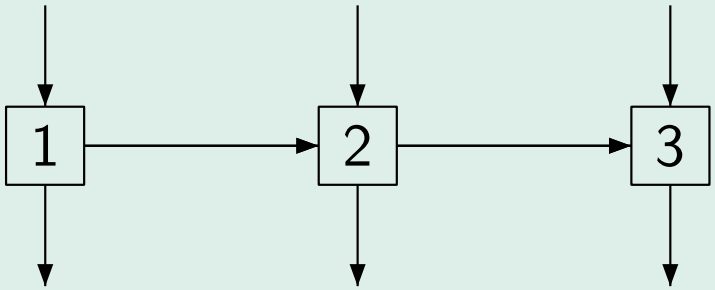


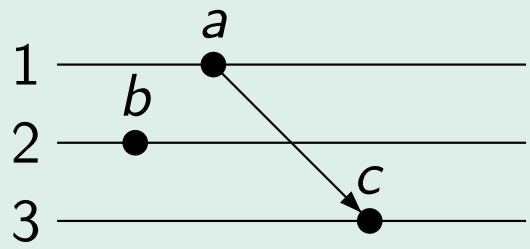


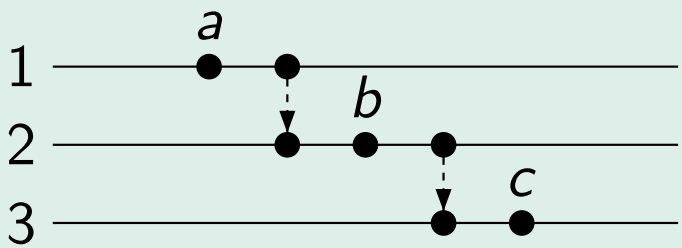


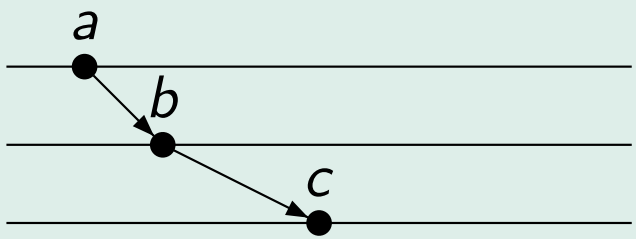
Specifications

Communication induces order relation



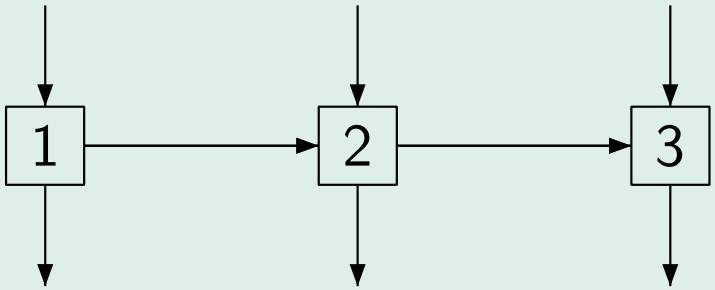


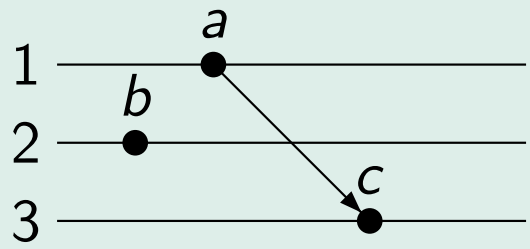


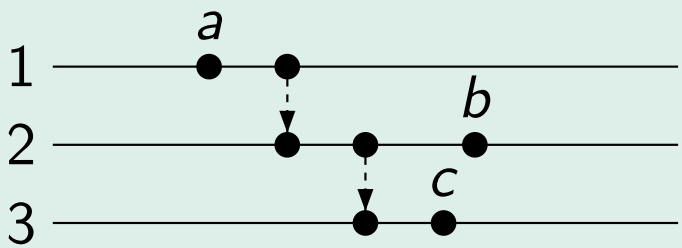


Specifications

Communication induces order relation

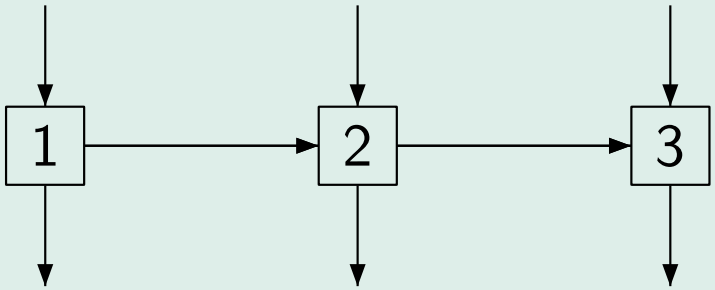


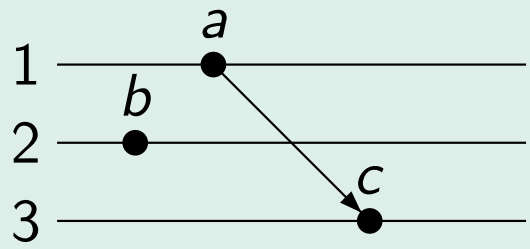


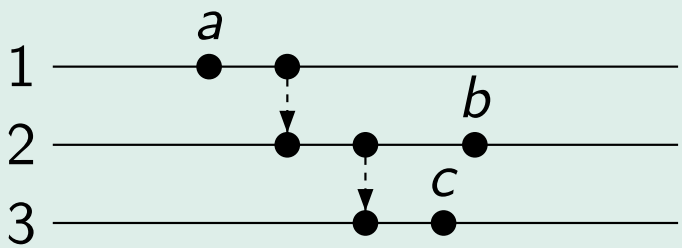


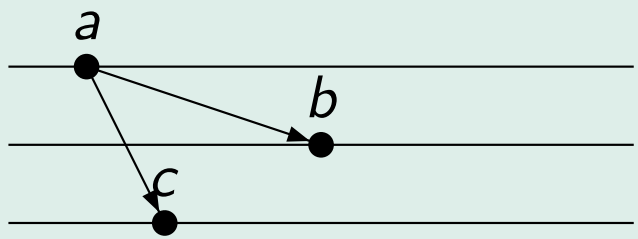
Specifications

Communication induces order relation









Specifications

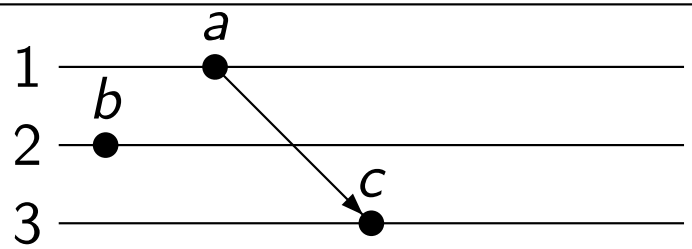
Restrictions on specifications

- Specifications should not discriminate between a partial order and its order extensions

Specifications

Restrictions on specifications

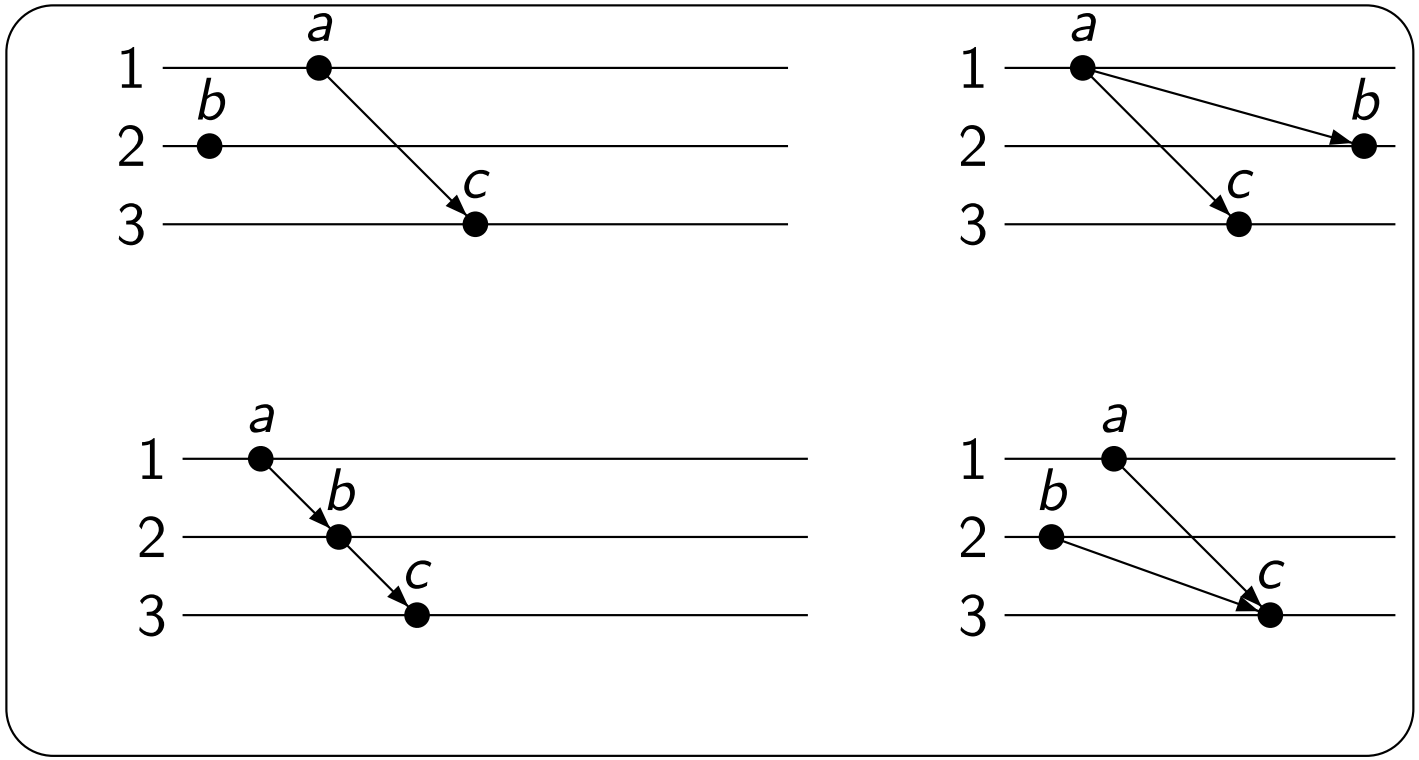
- Specifications should not discriminate between a partial order and its order extensions



Specifications

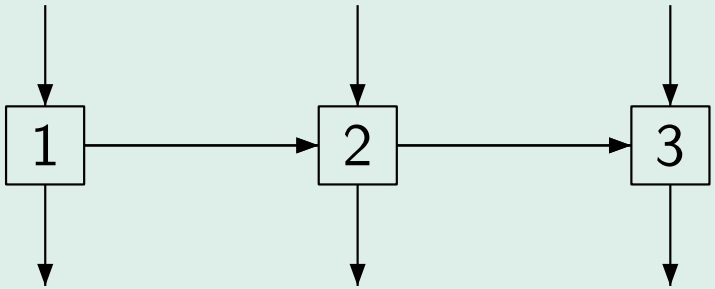
Restrictions on specifications

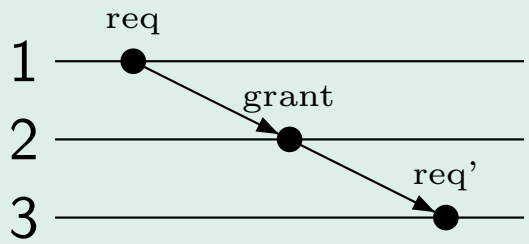
- Specifications should not discriminate between a partial order and its order extensions



Specifications

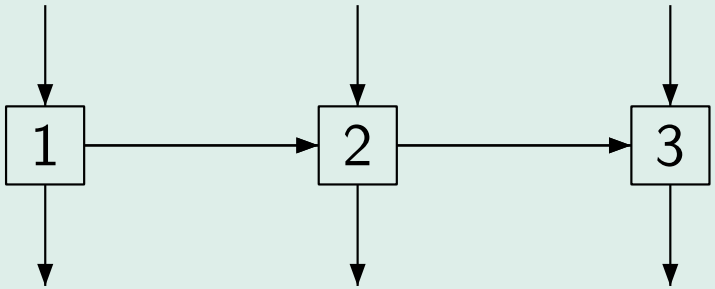
Input events are not controllable by processes

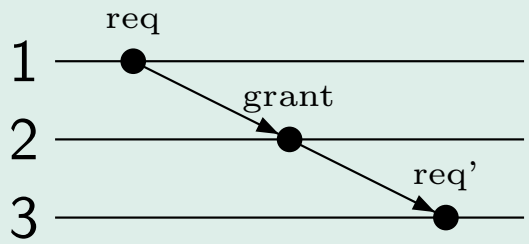


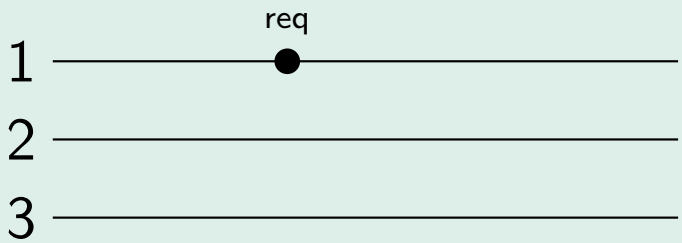


Specifications

Input events are not controllable by processes

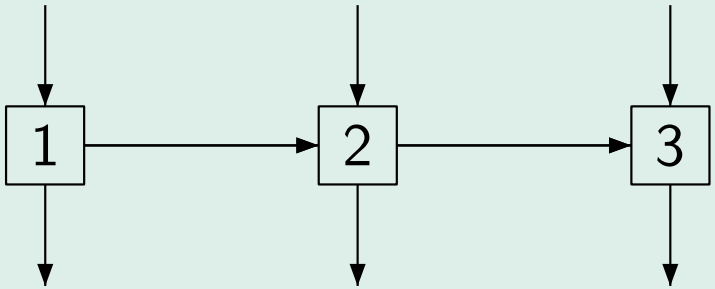


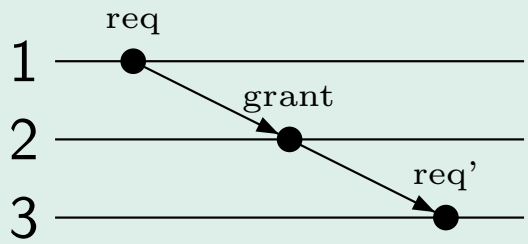


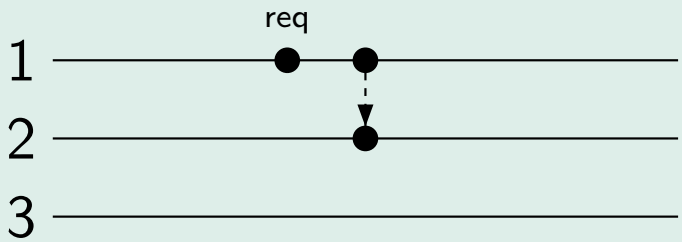


Specifications

Input events are not controllable by processes

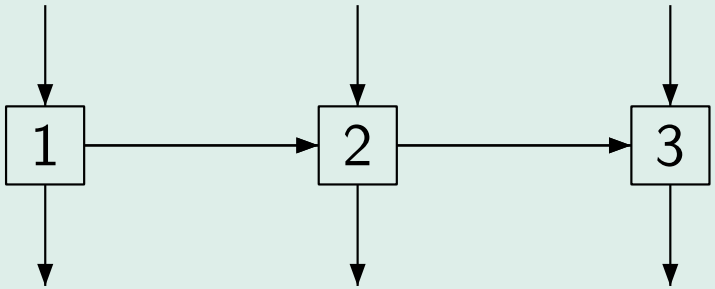


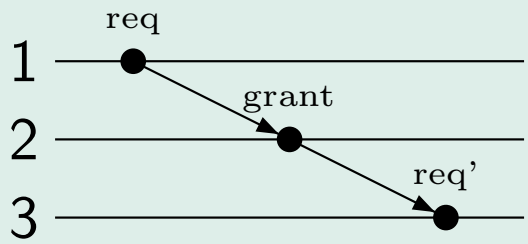


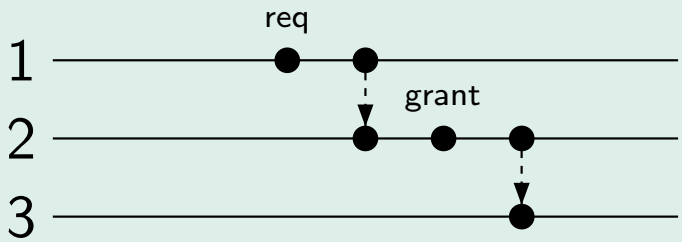


Specifications

Input events are not controllable by processes

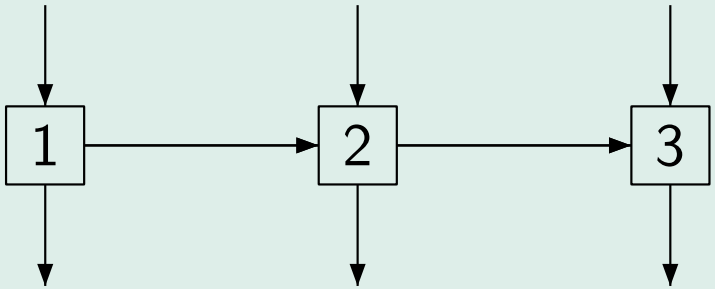


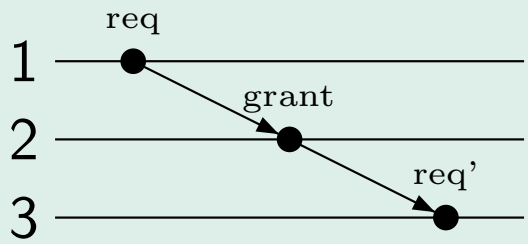


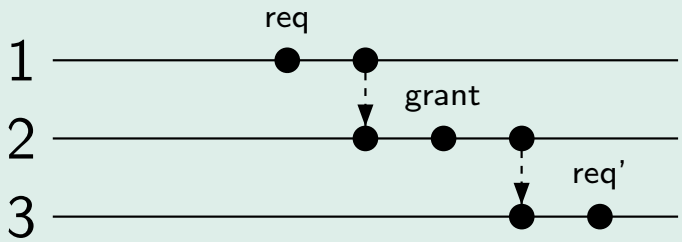


Specifications

Input events are not controllable by processes

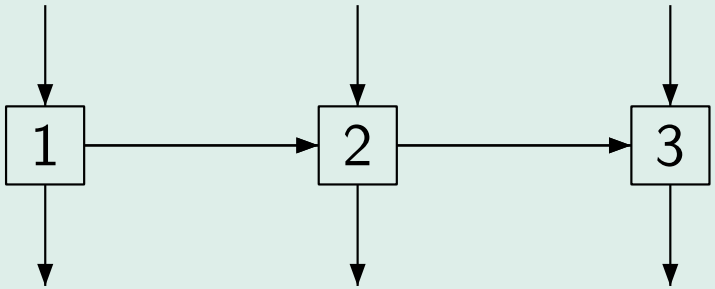


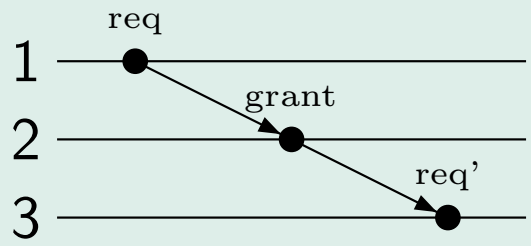


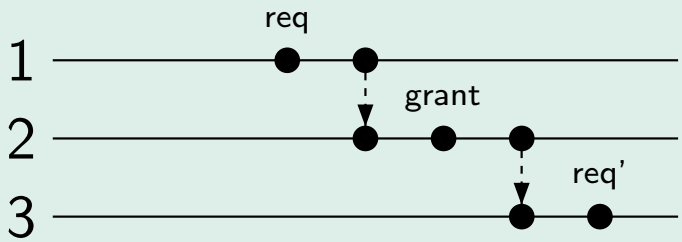


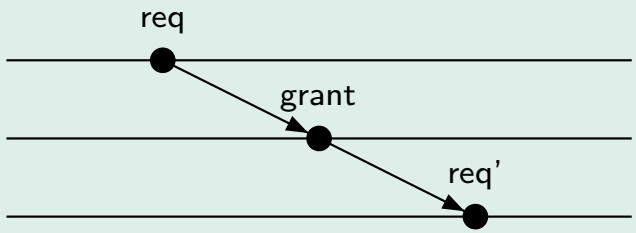
Specifications

Input events are not controllable by processes



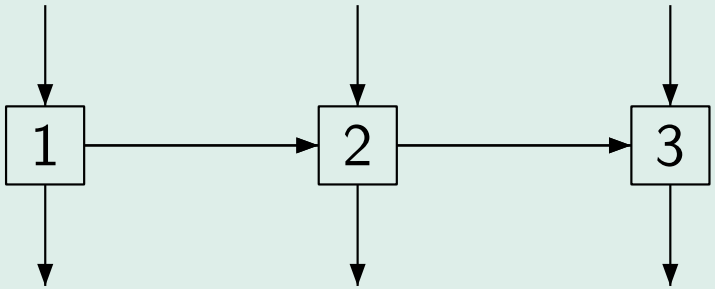


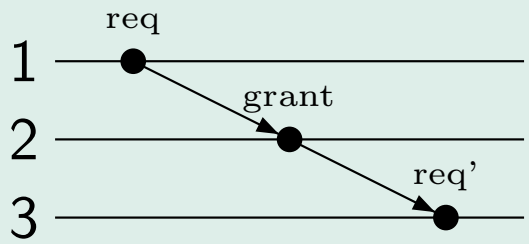


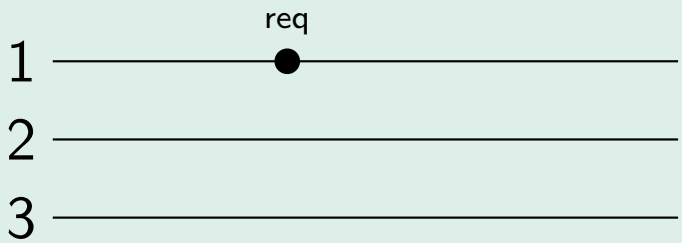


Specifications

Input events are not controllable by processes

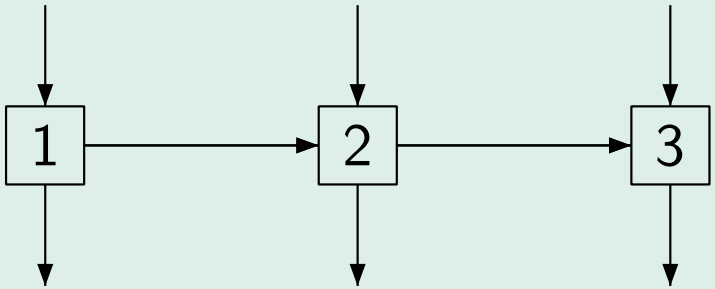


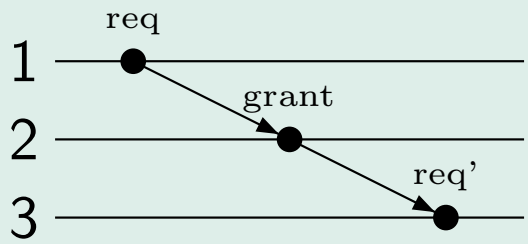


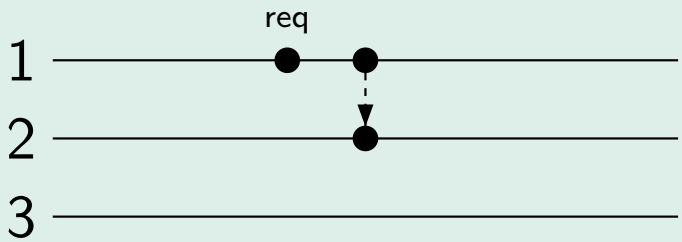


Specifications

Input events are not controllable by processes

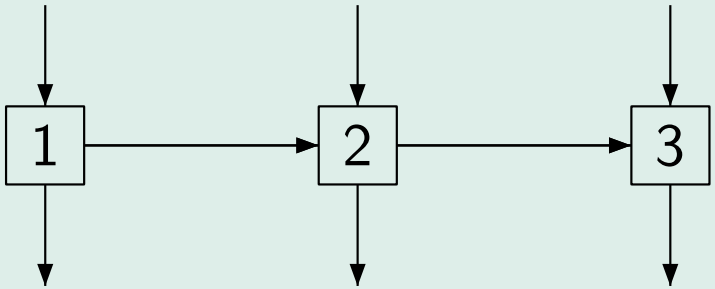


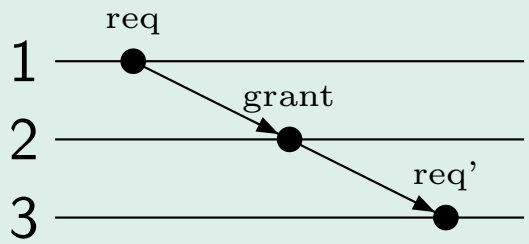


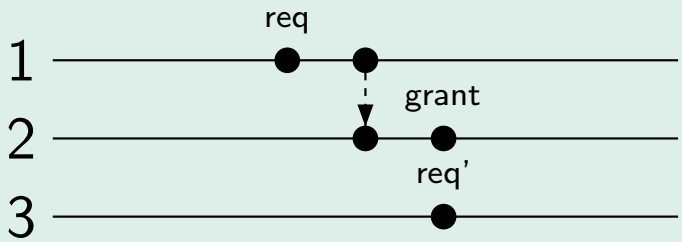


Specifications

Input events are not controllable by processes

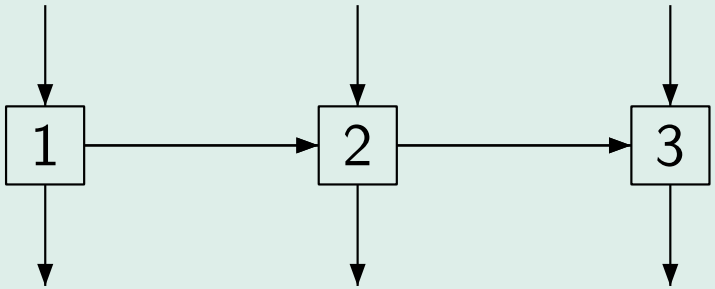


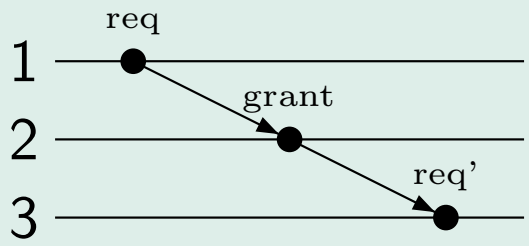


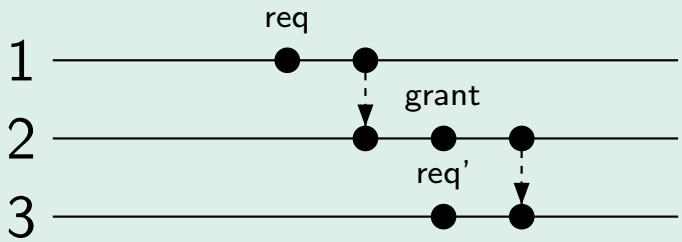


Specifications

Input events are not controllable by processes

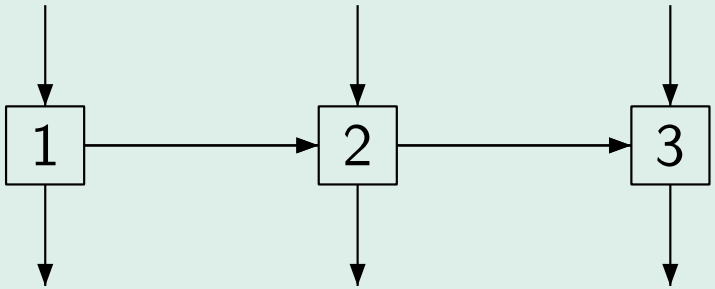


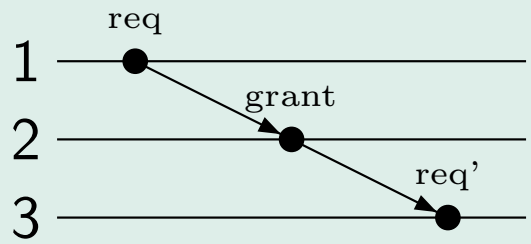


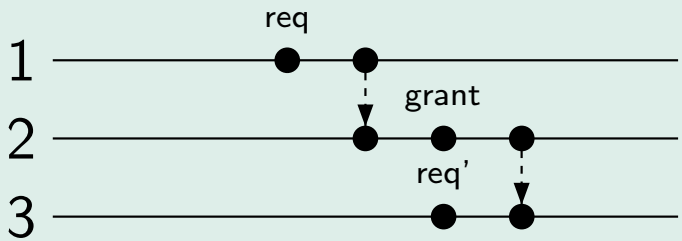


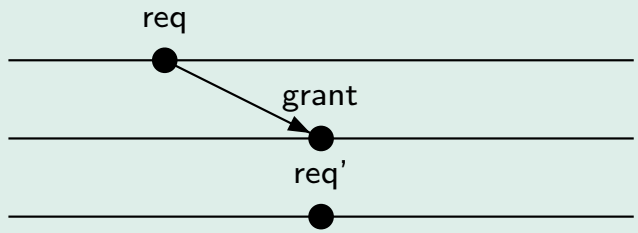
Specifications

Input events are not controllable by processes









Specifications

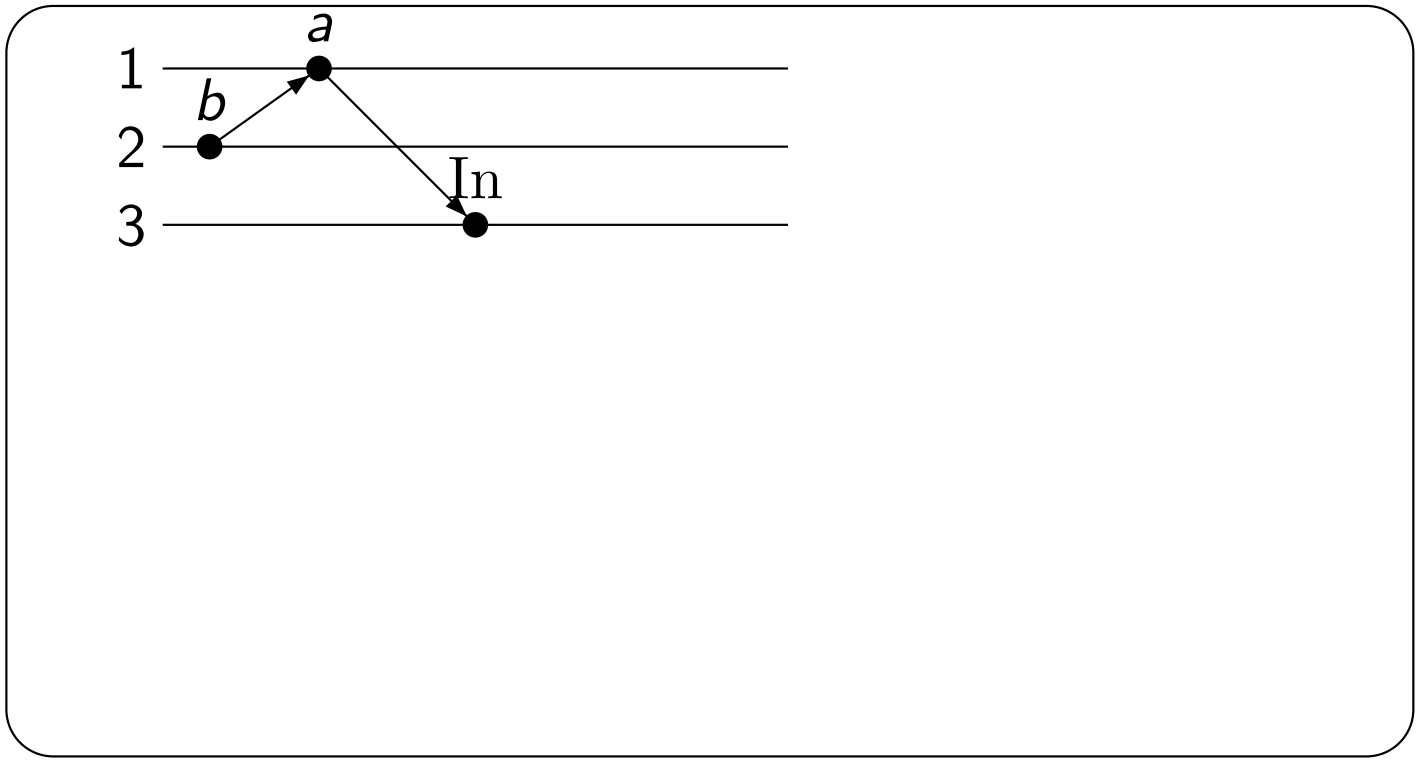
Restrictions on specifications

- Specifications should not discriminate between a partial order and its order extensions
- Specifications should not discriminate between a partial order and its "weakenings"

Specifications

Restrictions on specifications

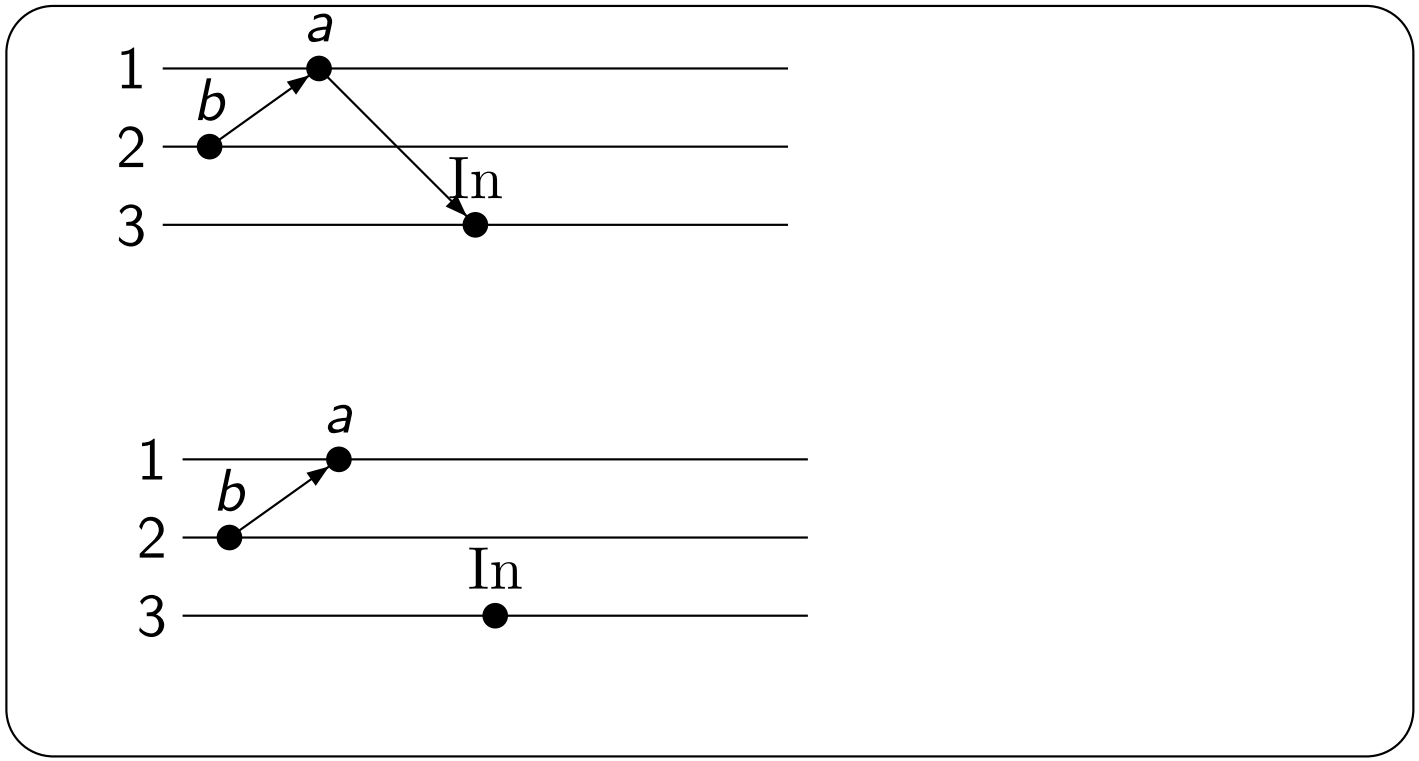
- Specifications should not discriminate between a partial order and its order extensions
- Specifications should not discriminate between a partial order and its "weakenings"



Specifications

Restrictions on specifications

- Specifications should not discriminate between a partial order and its order extensions
- Specifications should not discriminate between a partial order and its "weakenings"



Outline

Decidability Results

Theorem

The synthesis problem over singleton architectures is decidable for regular specifications.

Decidability Results

Theorem

The synthesis problem over singleton architectures is decidable for regular specifications.

Theorem

The distributed synthesis problem over strongly connected architectures is decidable for acceptable specifications.

Decidability Results

Theorem

The synthesis problem over singleton architectures is decidable for regular specifications.

Theorem

The distributed synthesis problem over strongly connected architectures is decidable for acceptable specifications.

Proof

By reduction to the singleton case.

Strongly connected architectures

Proposition

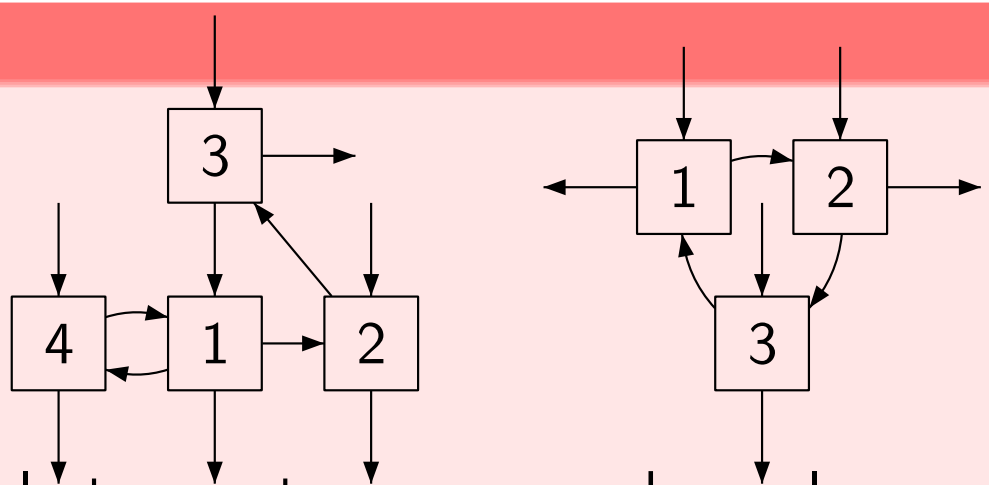
If there is a winning strategy over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected one.

Strongly connected architectures

Proposition

If there is a winning strategy over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected one.

Proof



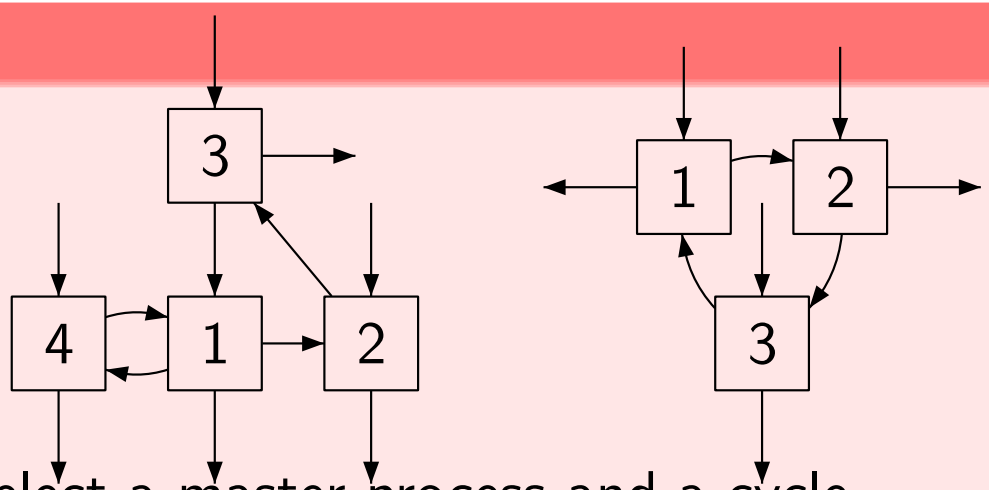
- We select a master process and a cycle.

Strongly connected architectures

Proposition

If there is a winning strategy over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected one.

Proof



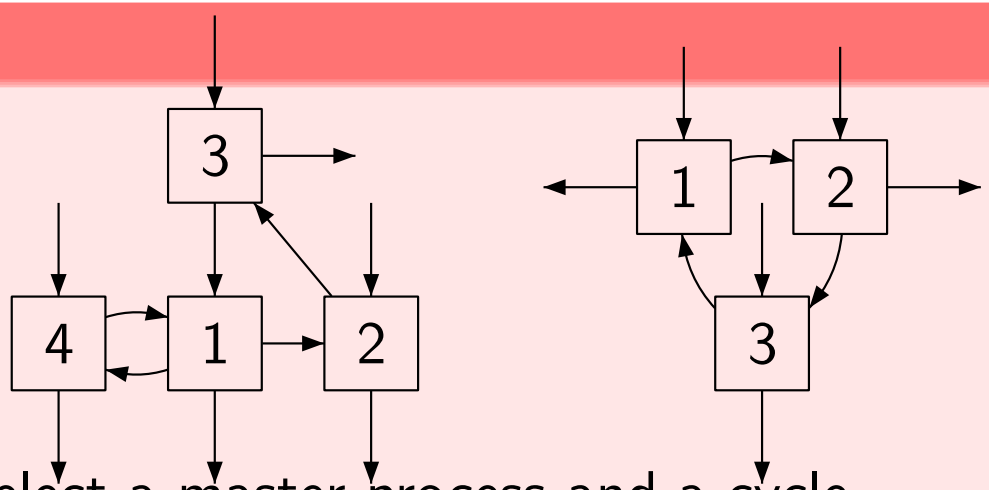
- We select a master process and a cycle.
- Master process will centralize information and tell other processes which value to output

Strongly connected architectures

Proposition

If there is a winning strategy over the singleton architecture then one can define internal signals sets and a distributed winning strategy for the strongly connected one.

Proof

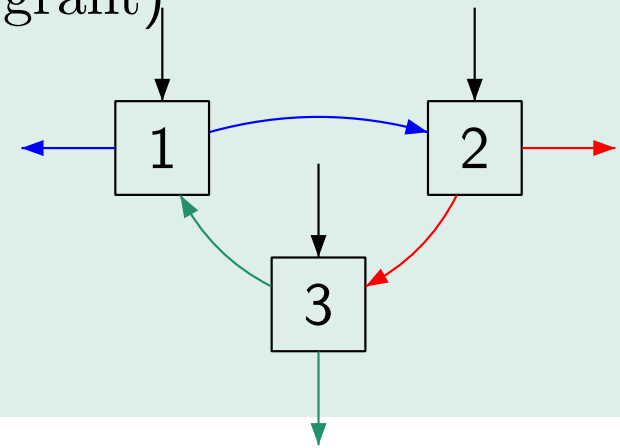


- We select a master process and a cycle.
- Master process will centralize information and tell other processes which value to output
- Objective: create a run that will be a weakening of a f-run over the singleton

Centralize information

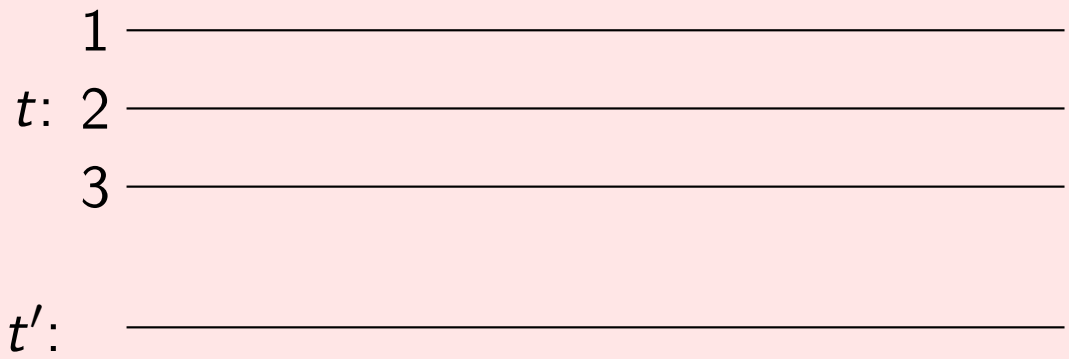
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_3 and no alert

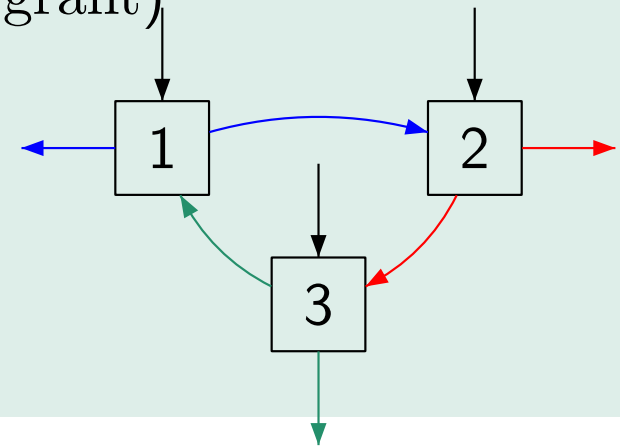
Master collect information by sending a signal Msg through the cycle



Centralize information

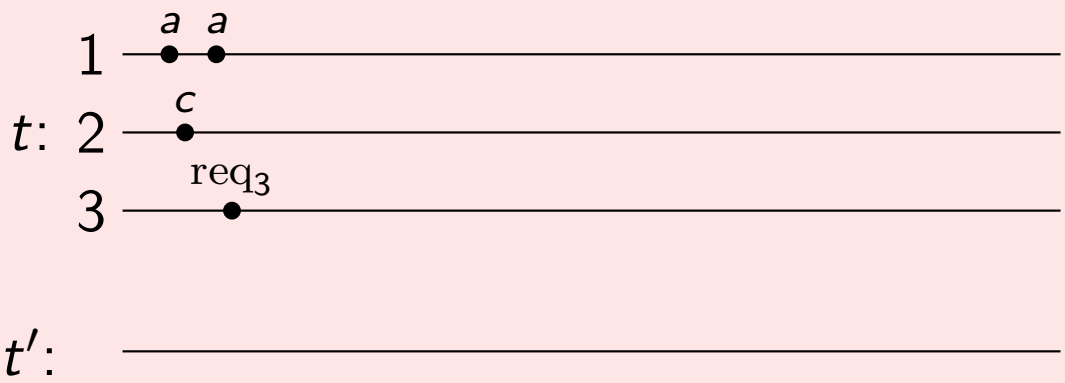
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_3 and no alert

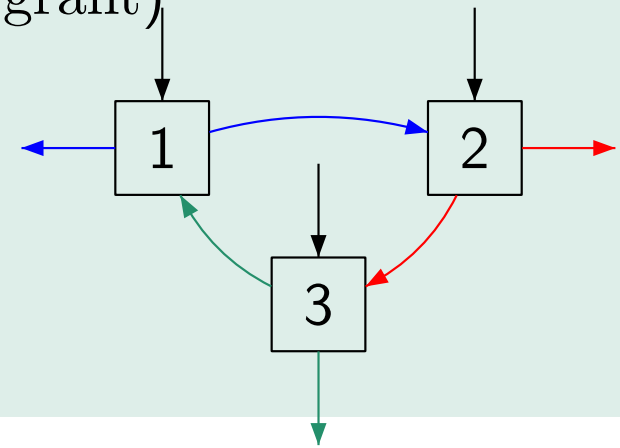
Master collect information by sending a signal Msg through the cycle



Centralize information

Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



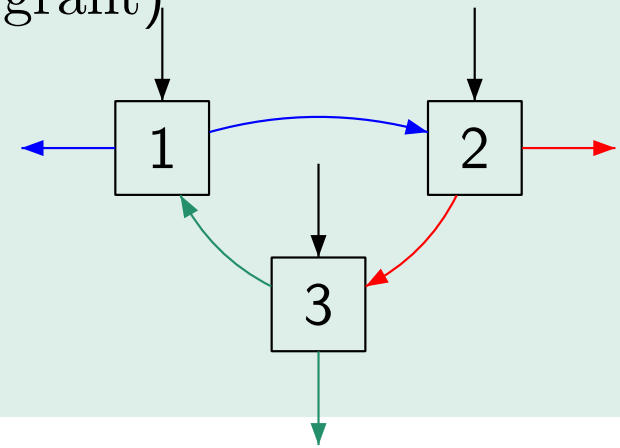
$f(\sigma) = \mathrm{grant}$ iff σ contains req_3 and no alert

Master collect information by sending a signal Msg through the cycle

Centralize information

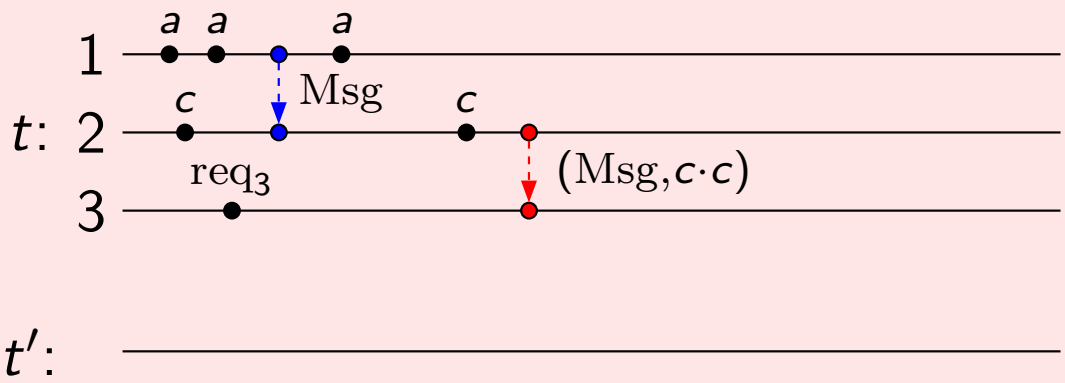
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_3 and no alert

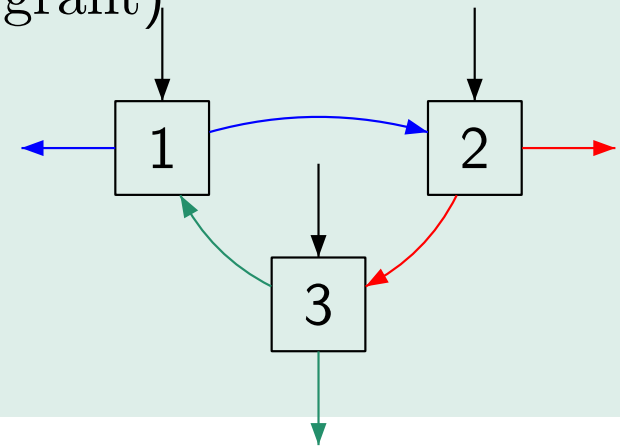
Master collect information by sending a signal Msg through the cycle



Centralize information

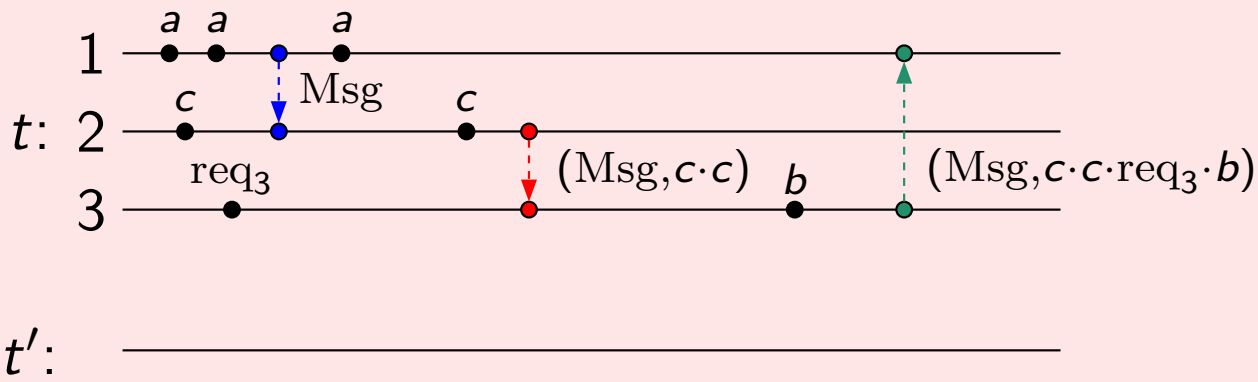
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_3 and no alert

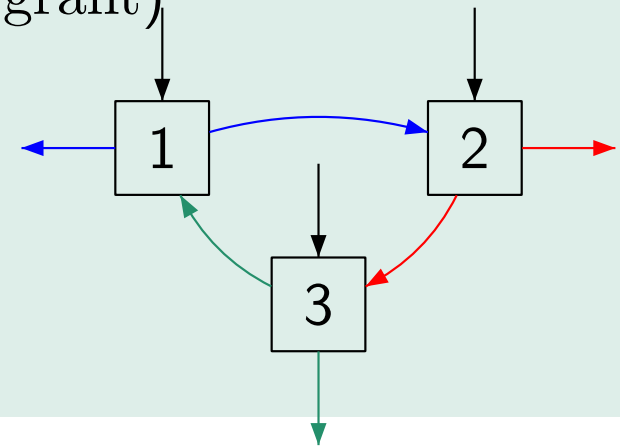
Master collect information by sending a signal Msg through the cycle



Centralize information

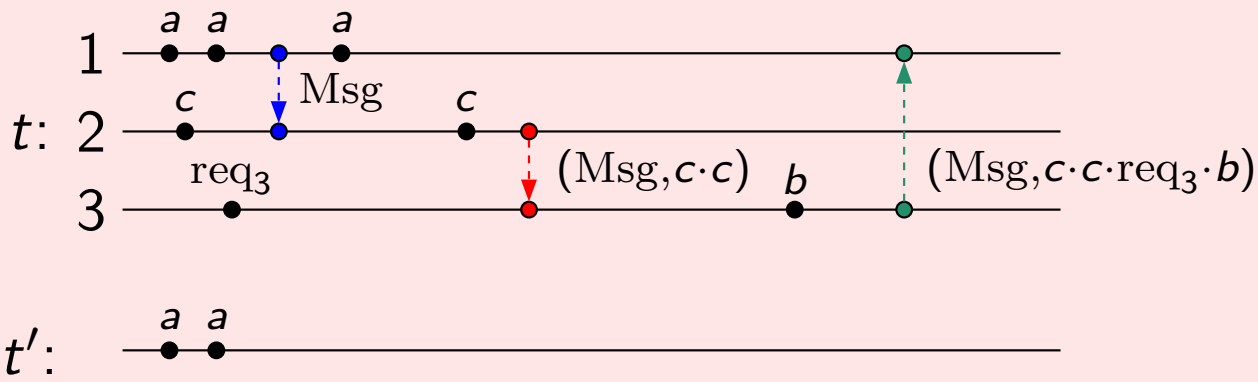
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_3 and no alert

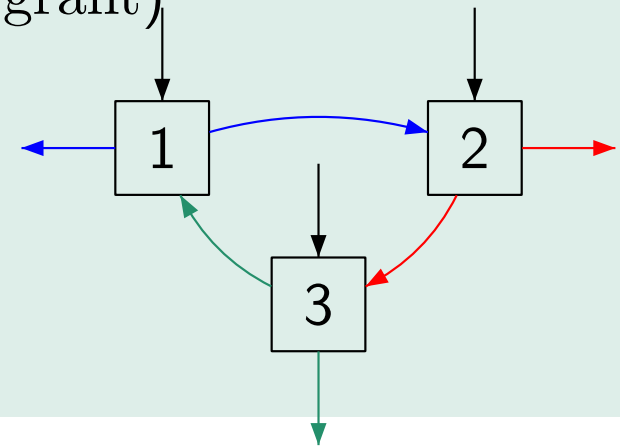
Master collect information by sending a signal Msg through the cycle



Centralize information

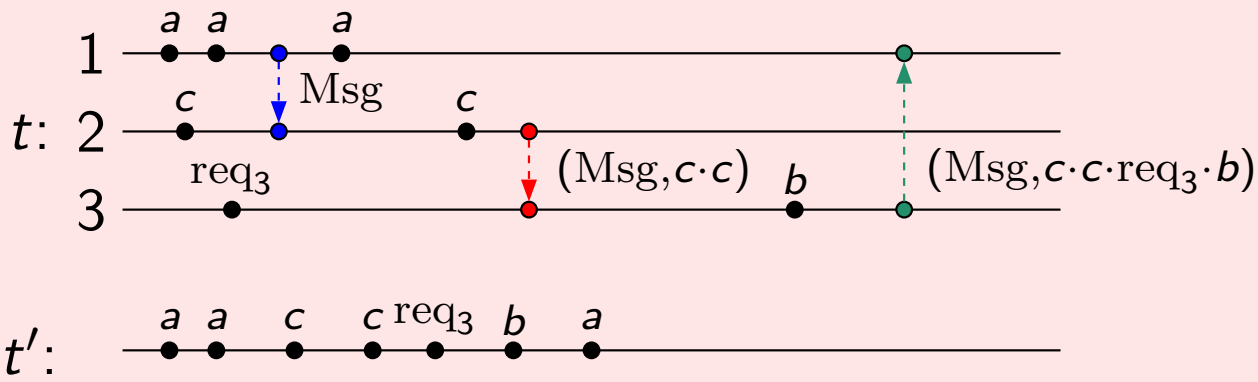
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_3 and no alert

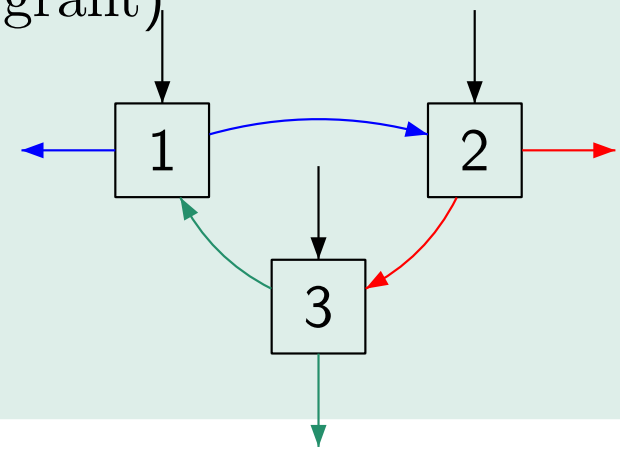
Master collect information by sending a signal Msg through the cycle



Tell processes what to output

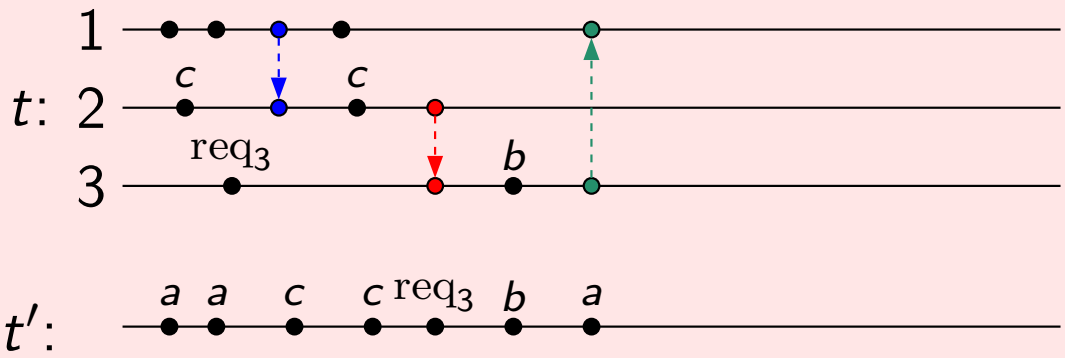
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

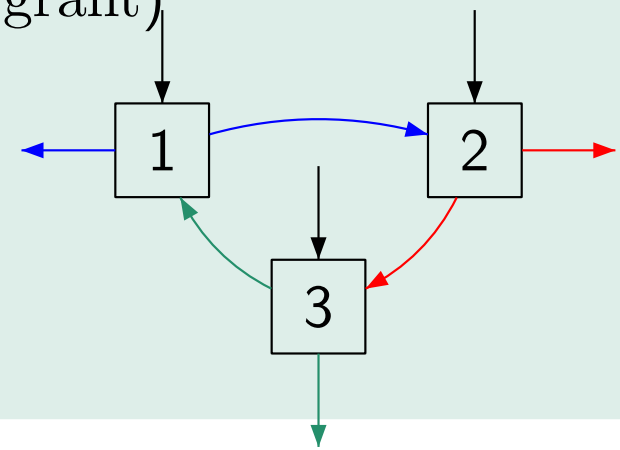
Master sends orders to other processes to simulate the strategy f



Tell processes what to output

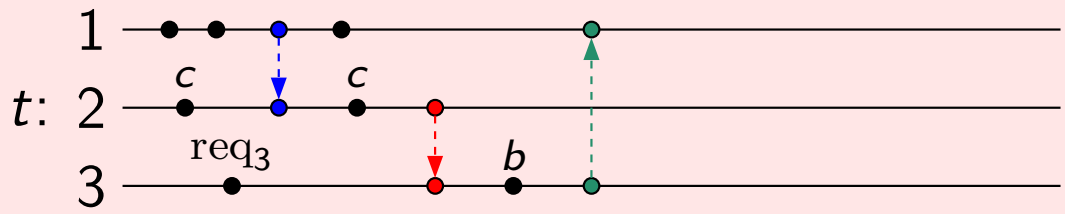
Example

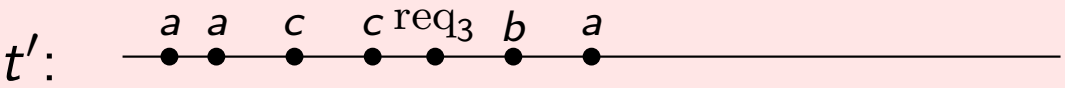
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



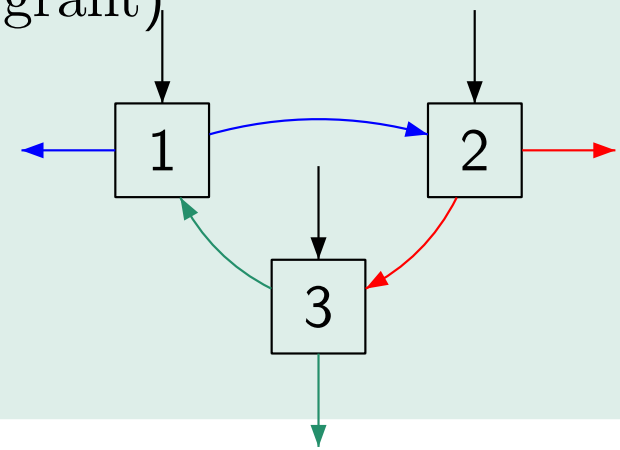


f : grant

Tell processes what to output

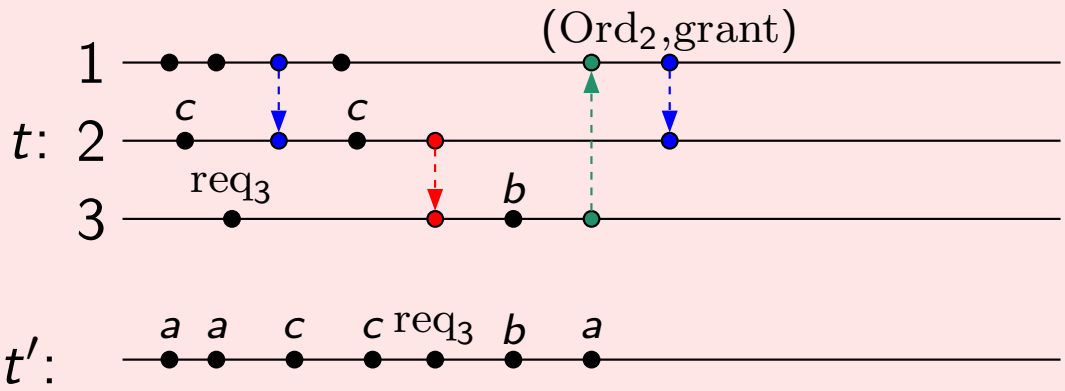
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f

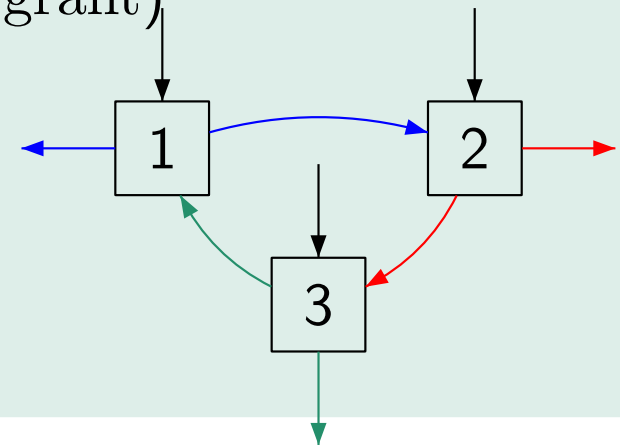


f : grant

Tell processes what to output

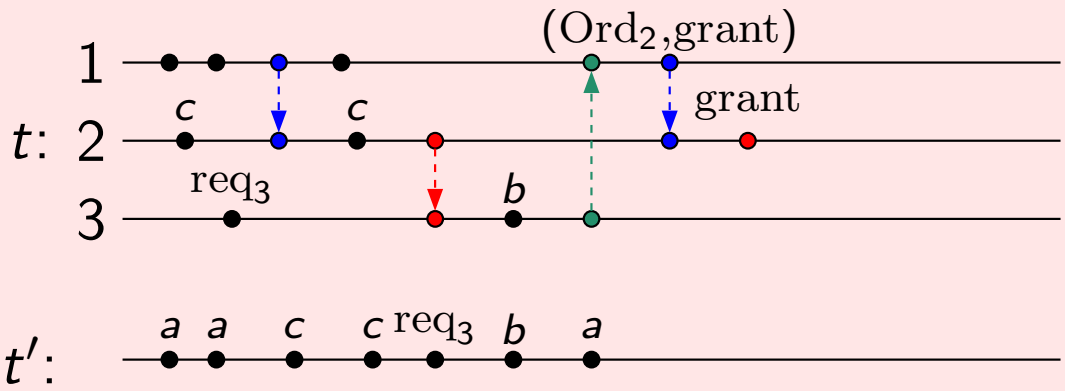
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f

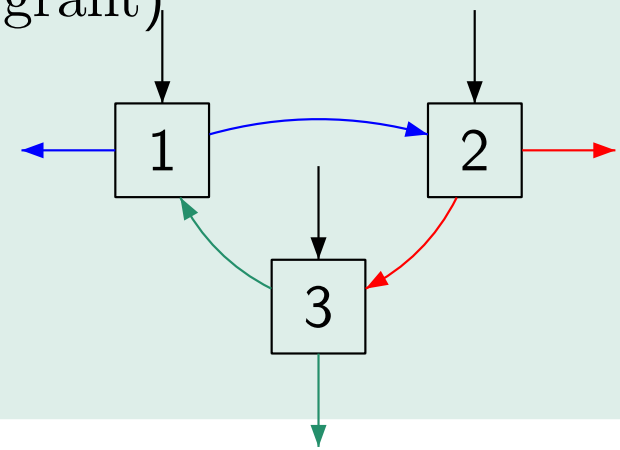


f : grant

Tell processes what to output

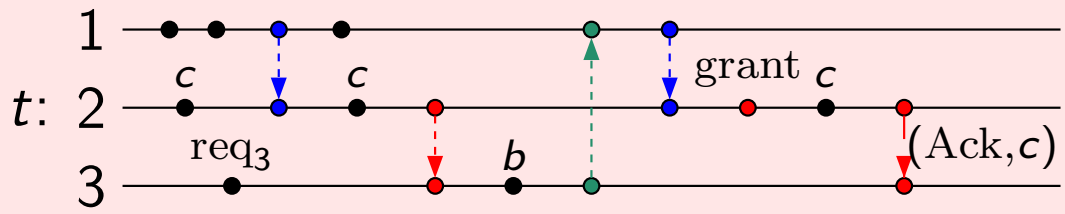
Example

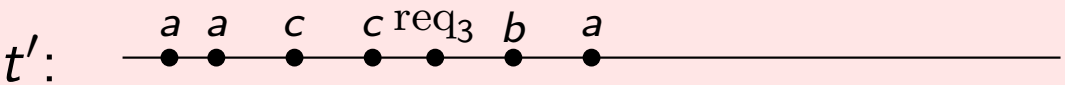
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



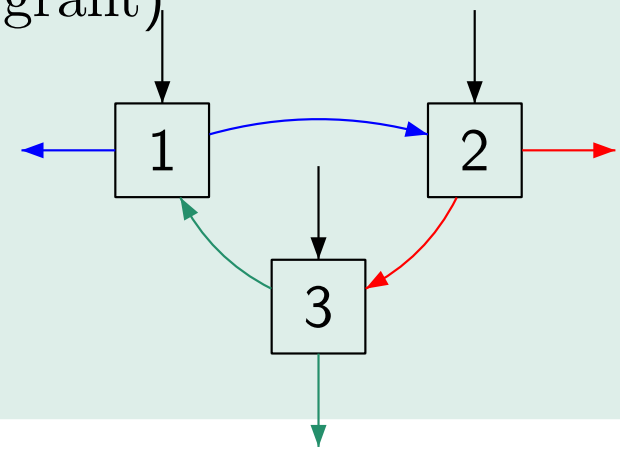


f : grant

Tell processes what to output

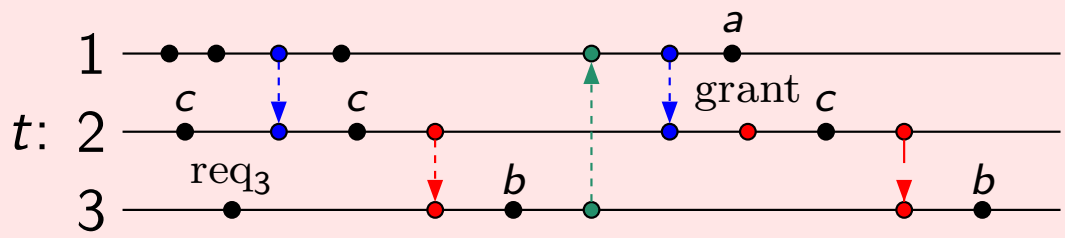
Example

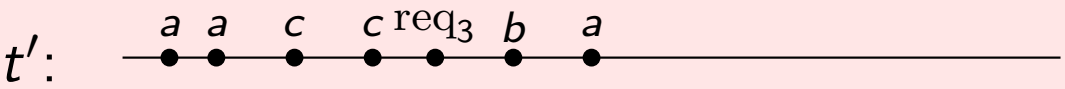
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



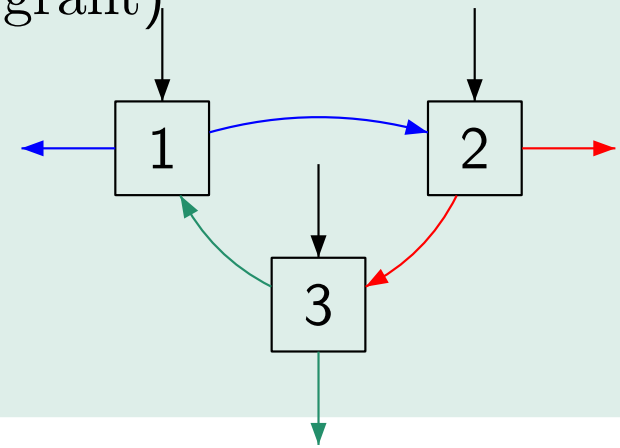


f : grant

Tell processes what to output

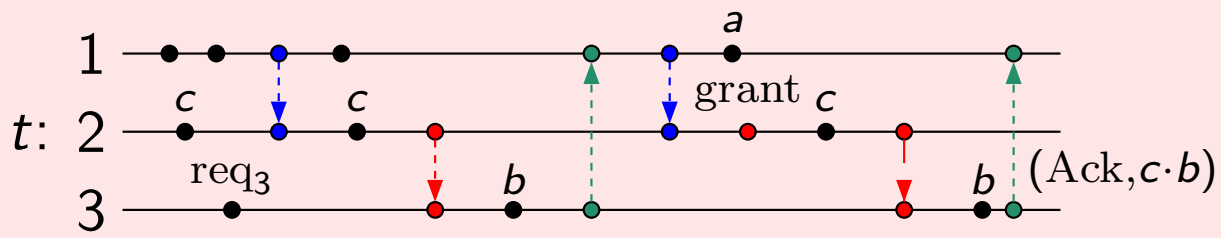
Example

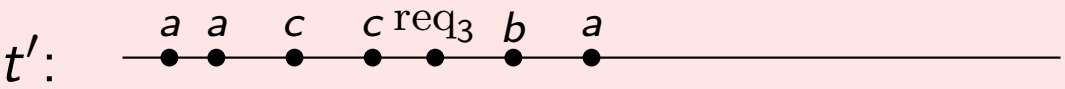
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



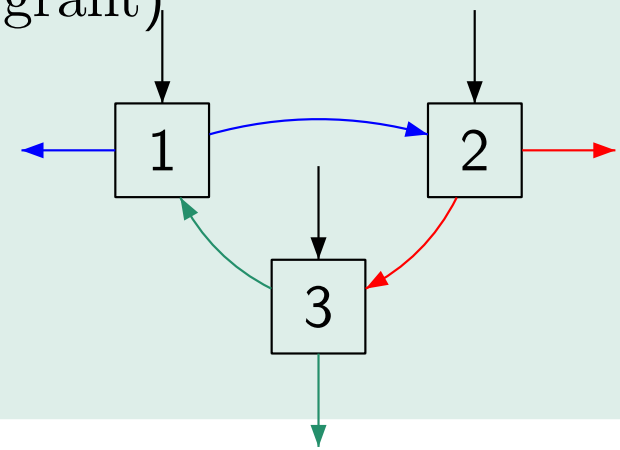


f : grant

Tell processes what to output

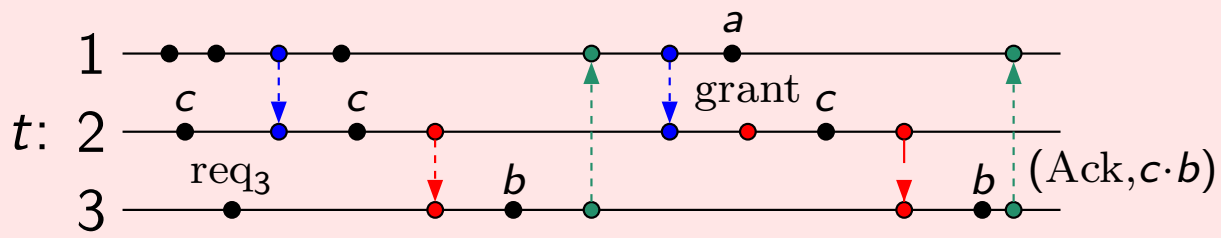
Example

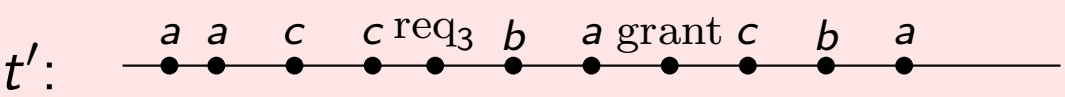
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



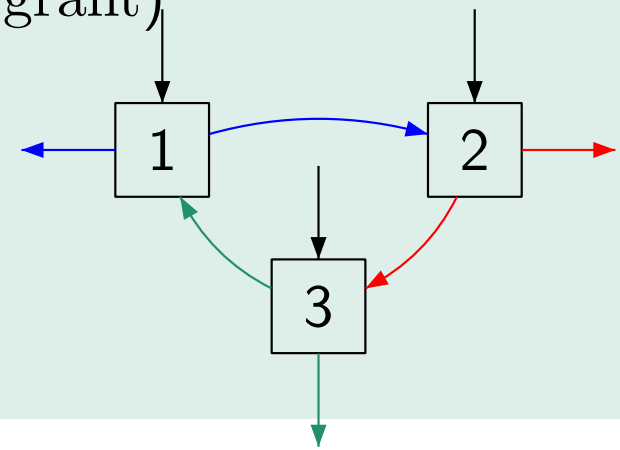


f : grant

Tell processes what to ouptut (2)

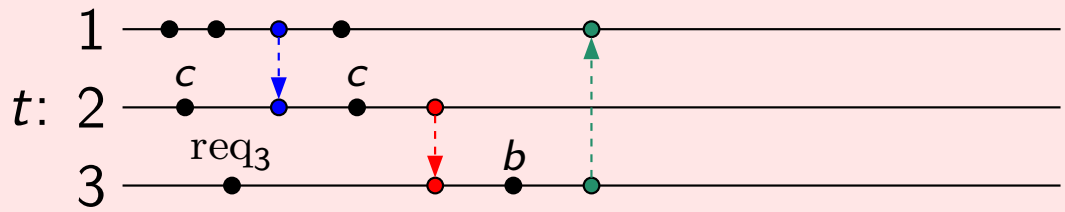
Example

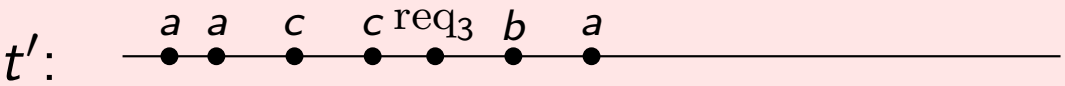
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



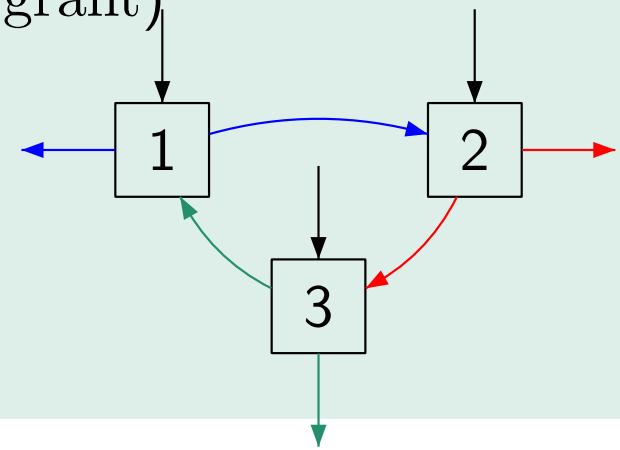


f : grant

Tell processes what to ouptut (2)

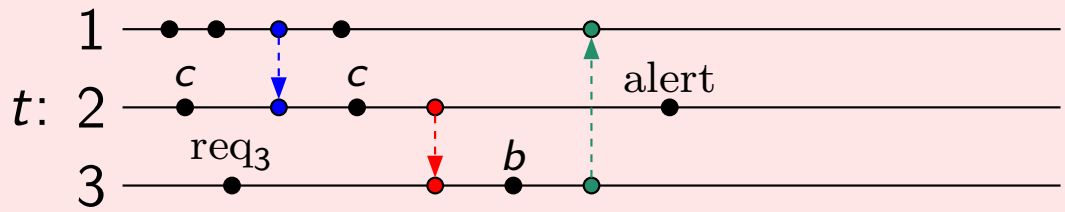
Example

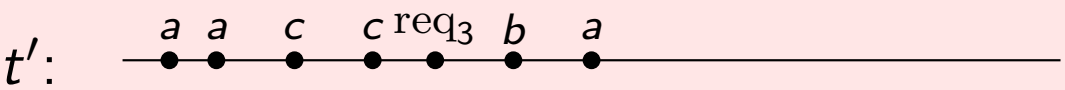
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



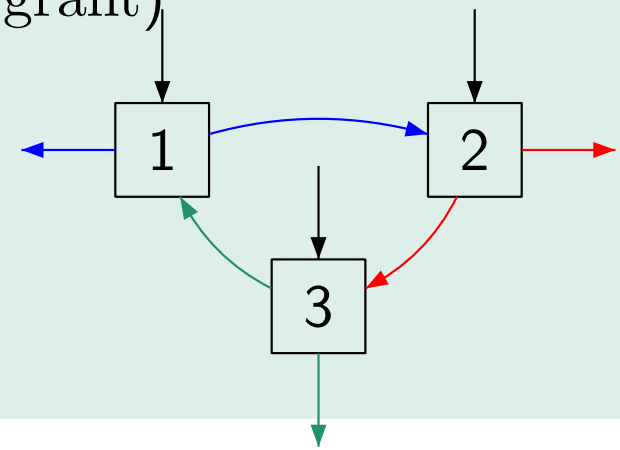


f : grant

Tell processes what to ouptut (2)

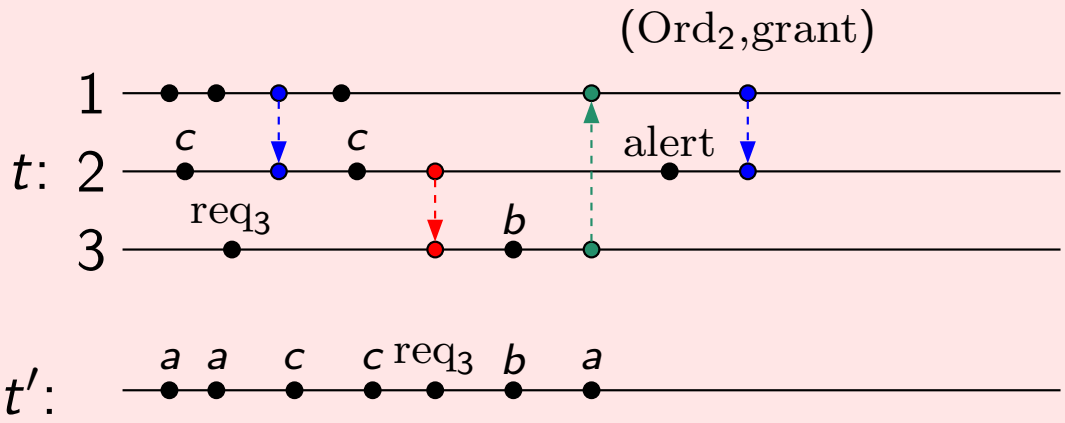
Example

$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f

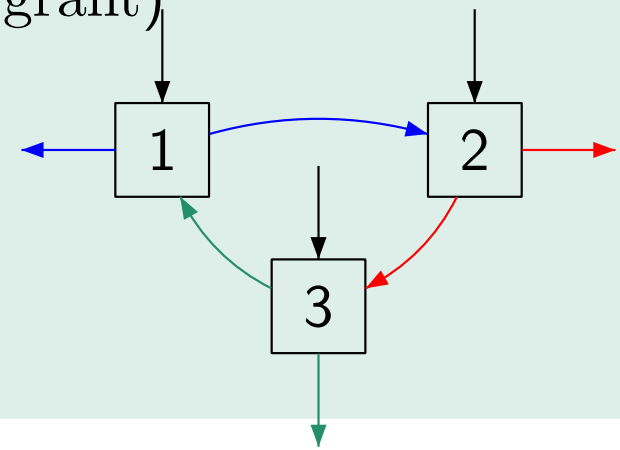


f : grant

Tell processes what to ouptut (2)

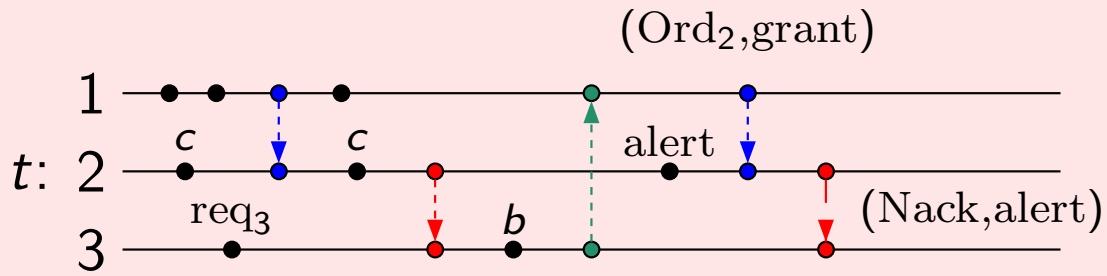
Example

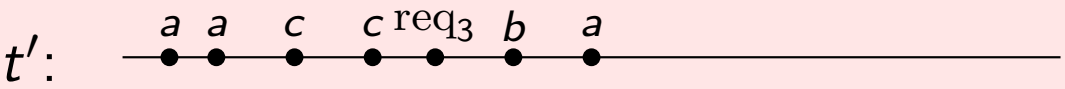
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



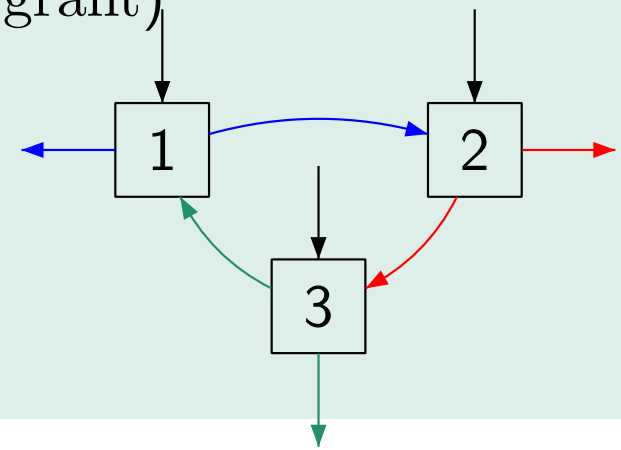


f : grant

Tell processes what to ouptut (2)

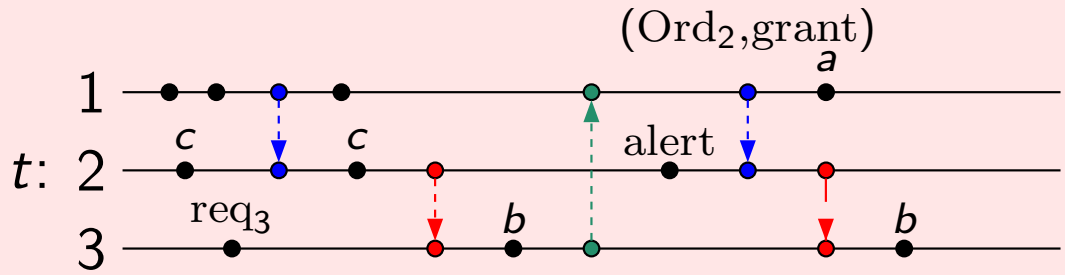
Example

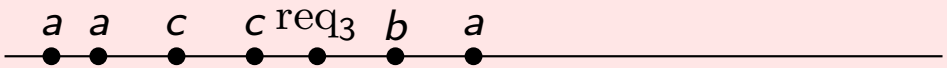
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



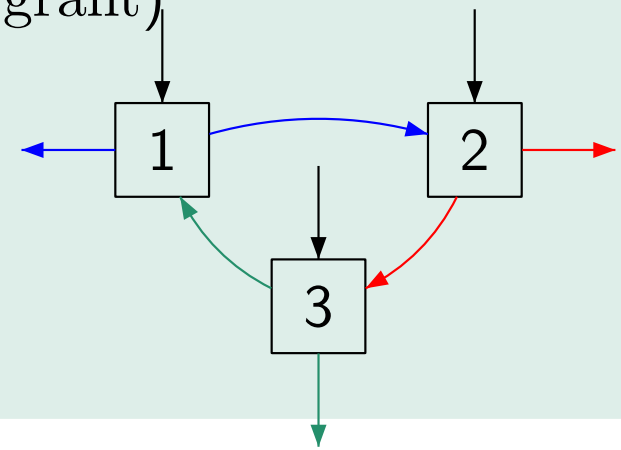


f : grant

Tell processes what to ouptut (2)

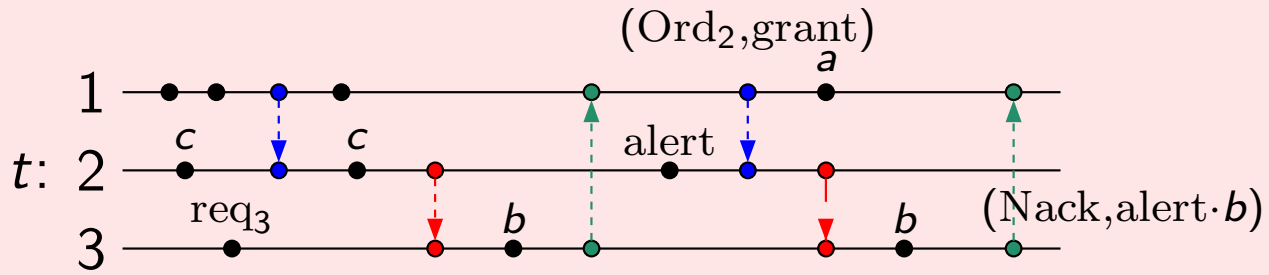
Example

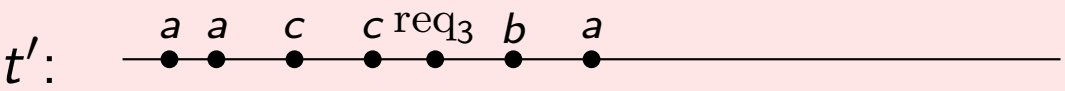
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



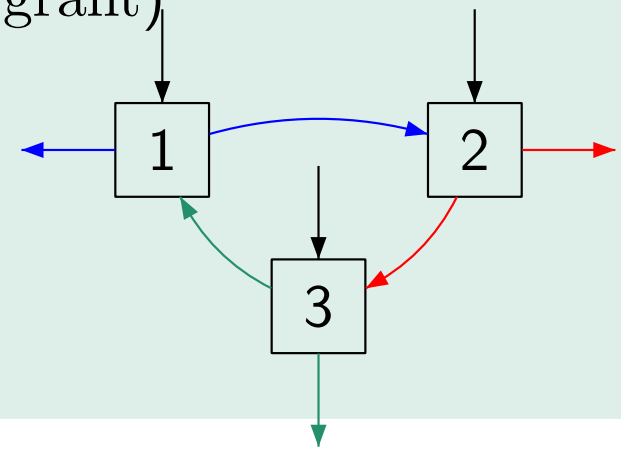


f : grant

Tell processes what to ouptut (2)

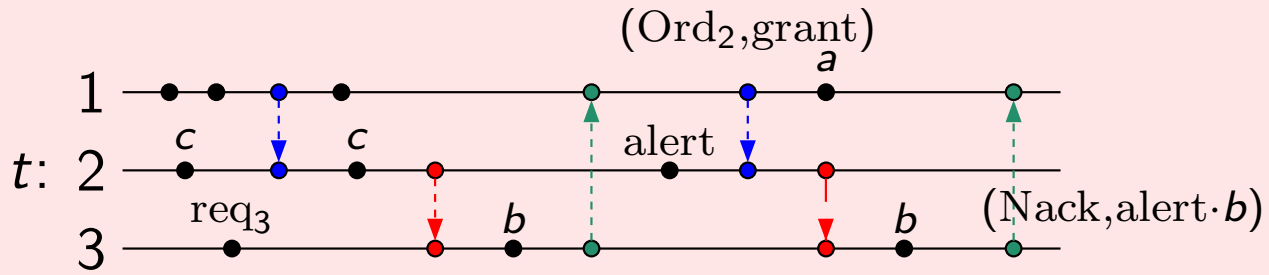
Example

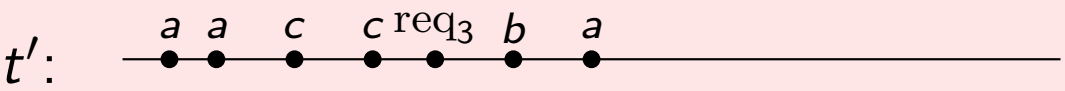
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f



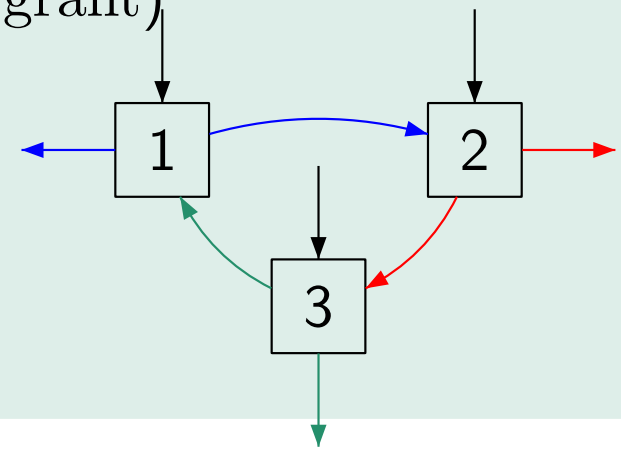


f : grant

Tell processes what to ouptut (2)

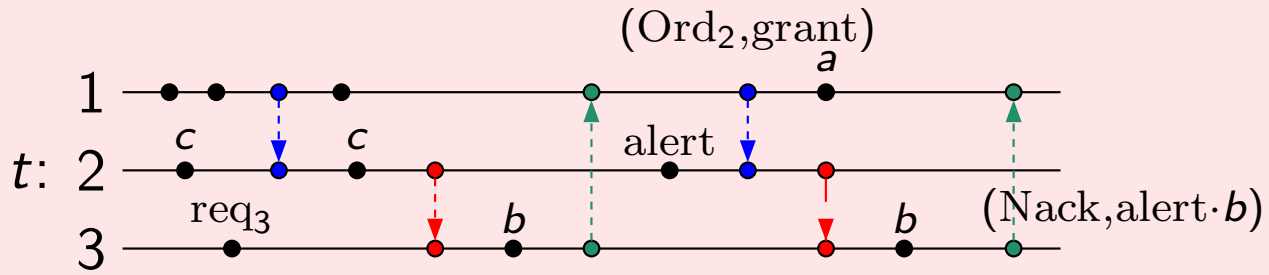
Example

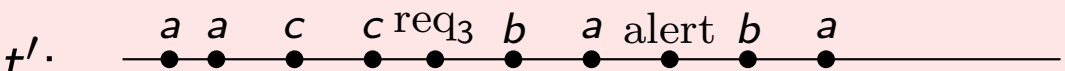
$$\mathrm{req}_3 \rightarrow \mathsf{F}_{32}(\neg\, \mathsf{Y}_2\, \mathrm{alert} \leftrightarrow \mathrm{grant})$$



$f(\sigma) = \mathrm{grant}$ iff σ contains req_2 and no alert

Master sends orders to other processes to simulate the strategy f





f : grant

Proof - end

Lemma

t' is an f-run.

Lemma

$\pi_\Gamma(t)$ is a weakening of t'.

Conclusion

Then $t' \models \varphi$ and, by closure property $\pi_\Gamma(t) \models \varphi$.

Proof - end

Lemma

t' is an f-run.

Lemma

$\pi_\Gamma(t)$ is a weakening of t'.

Conclusion

Then $t' \models \varphi$ and, by closure property $\pi_\Gamma(t) \models \varphi$.

Proposition

If there are communication sets $\Sigma_{i,j}$ for $(i,j) \in E$ and a winning distributed strategy on the strongly connected architecture, then there is a winning strategy on the singleton.

Proof

Easier.

Conclusion

- Asynchrony removes undecidability causes
- We have defined a new model of communication
- We have identified a class of decidable architectures
- Hopefully, many more to come!