Bounded Verification of Petri Nets and EOSs using Telingo: an Experience Report

CILC 2024 - Rome, 28/6/2024

Francesco Di Cosmo

Free University of Bozen-Bolzano

Tephilla Prince

IIT Dharwad, India

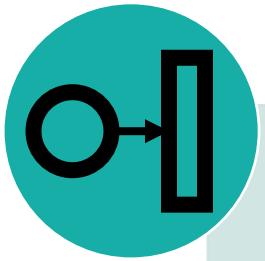
unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

॥ सा विद्या या विमुक्तये ॥
ಭಾ.ತಂ.ಸಂ. ಧಾರವಾಡ
भा. प्रौ. सं. धारवाड
IIT DHARWAD

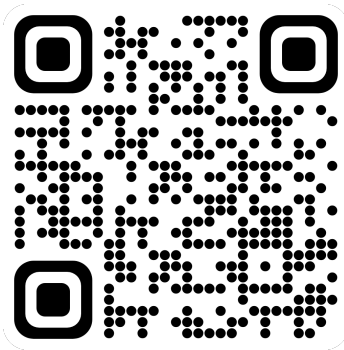
Contribution

Created a Verification Prototype

For safety, reach, cover, deadlock

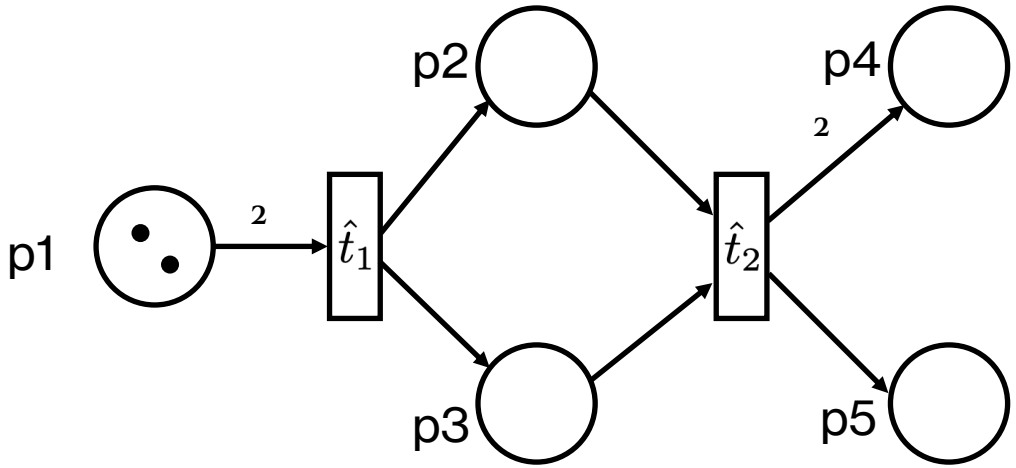


Of lossy PNs and EOSs

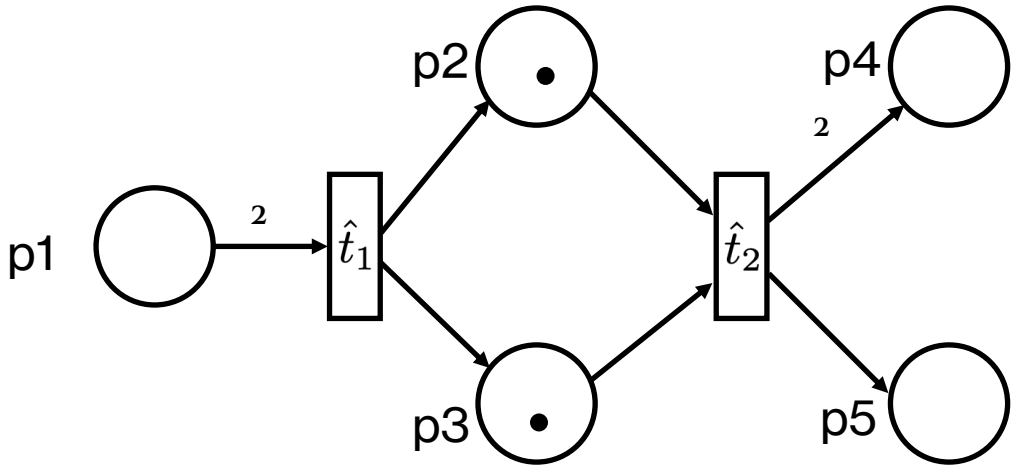


Available on Zenodo

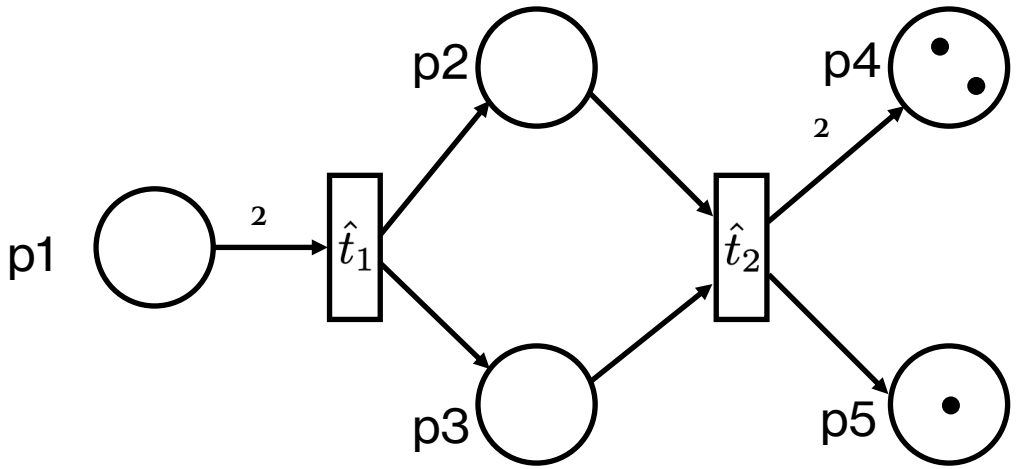
Petri Nets



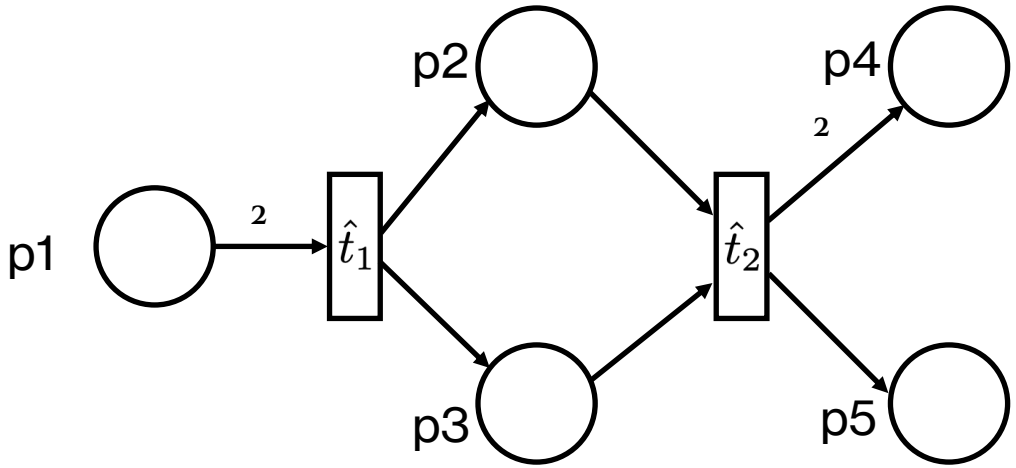
Petri Nets



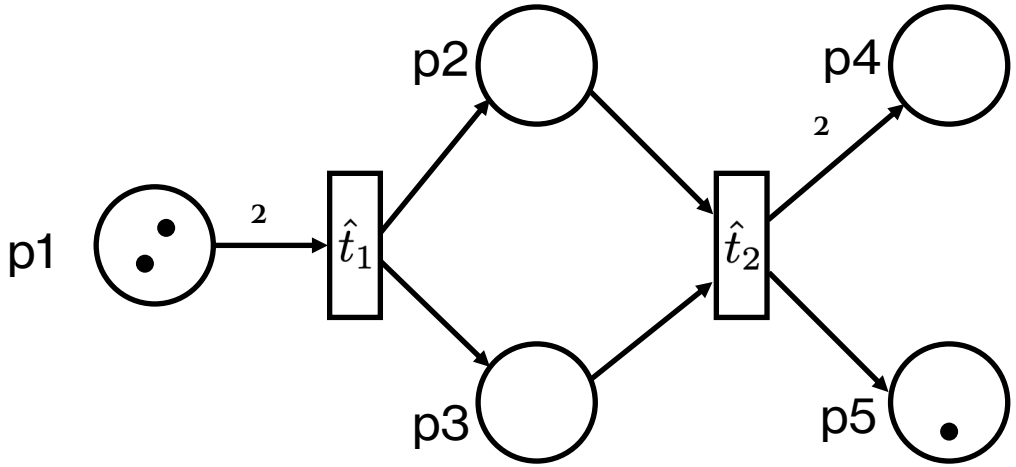
Petri Nets



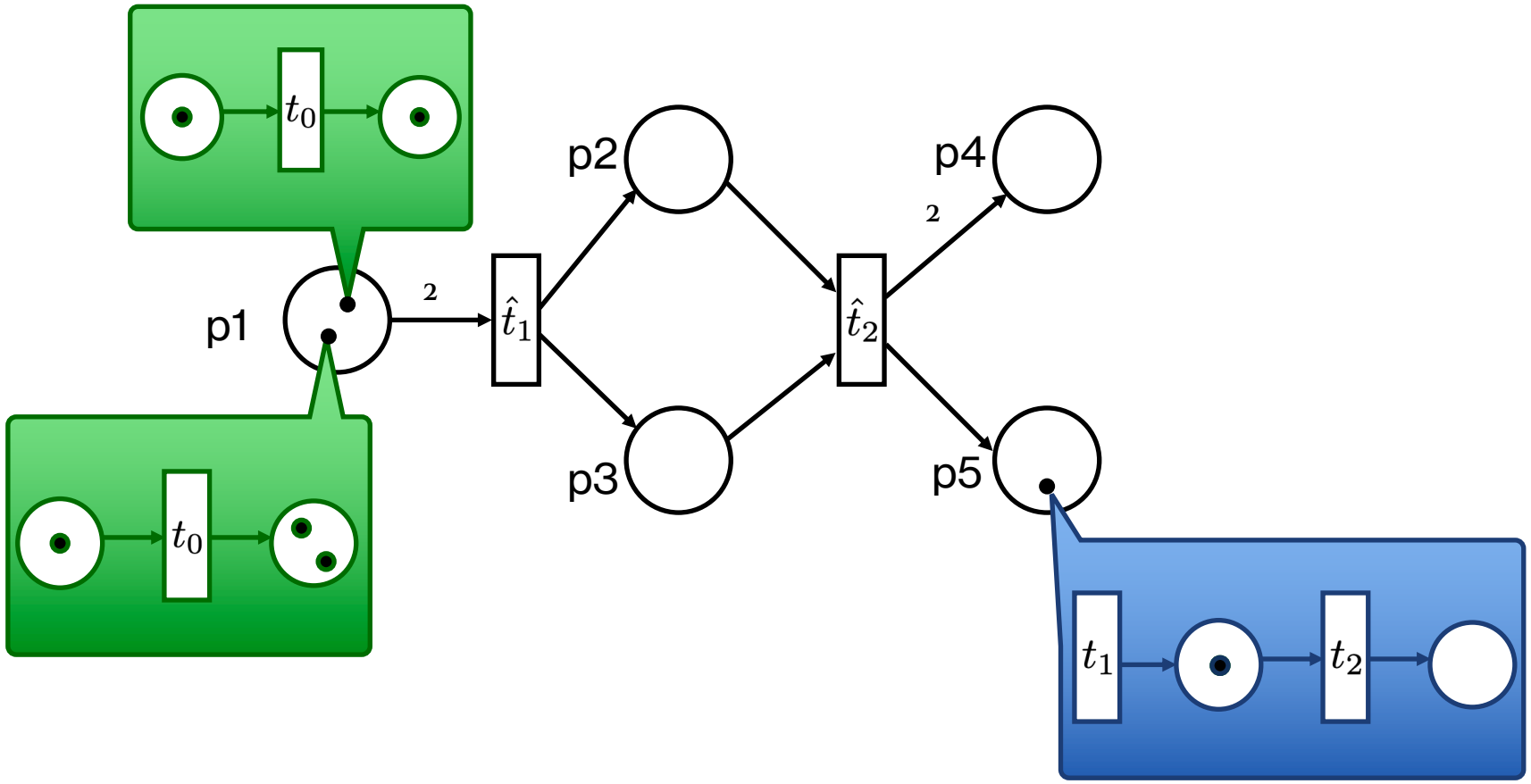
EOS – System Net



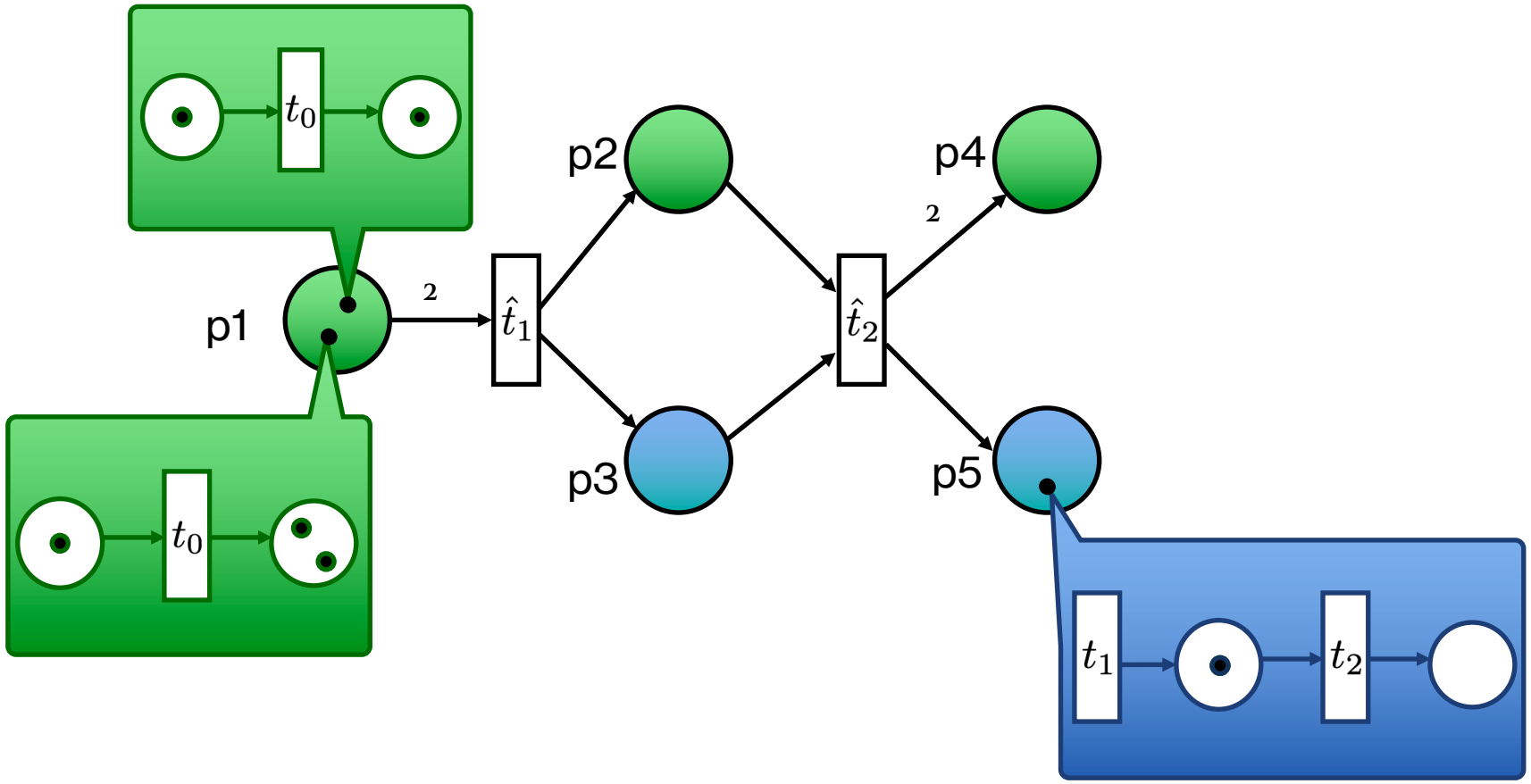
EOS – nested markings



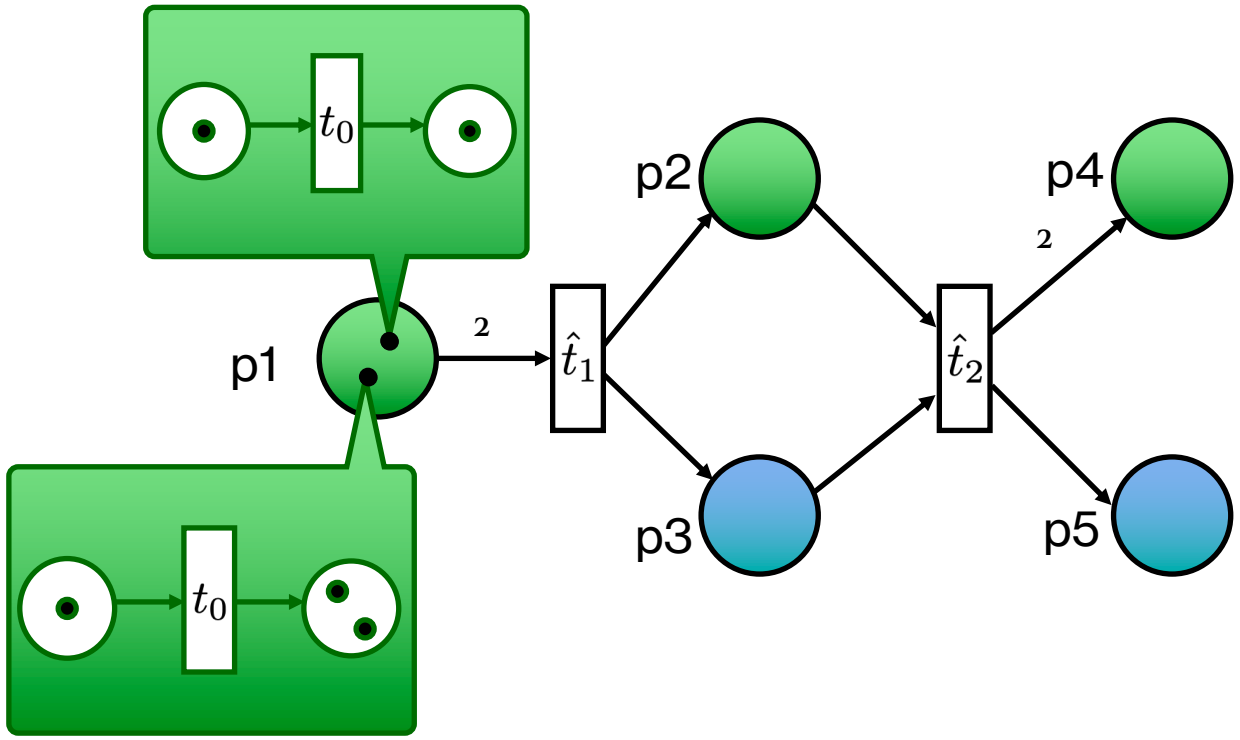
EOS – nested markings



EOS – typed places



EOS – object autonomous events



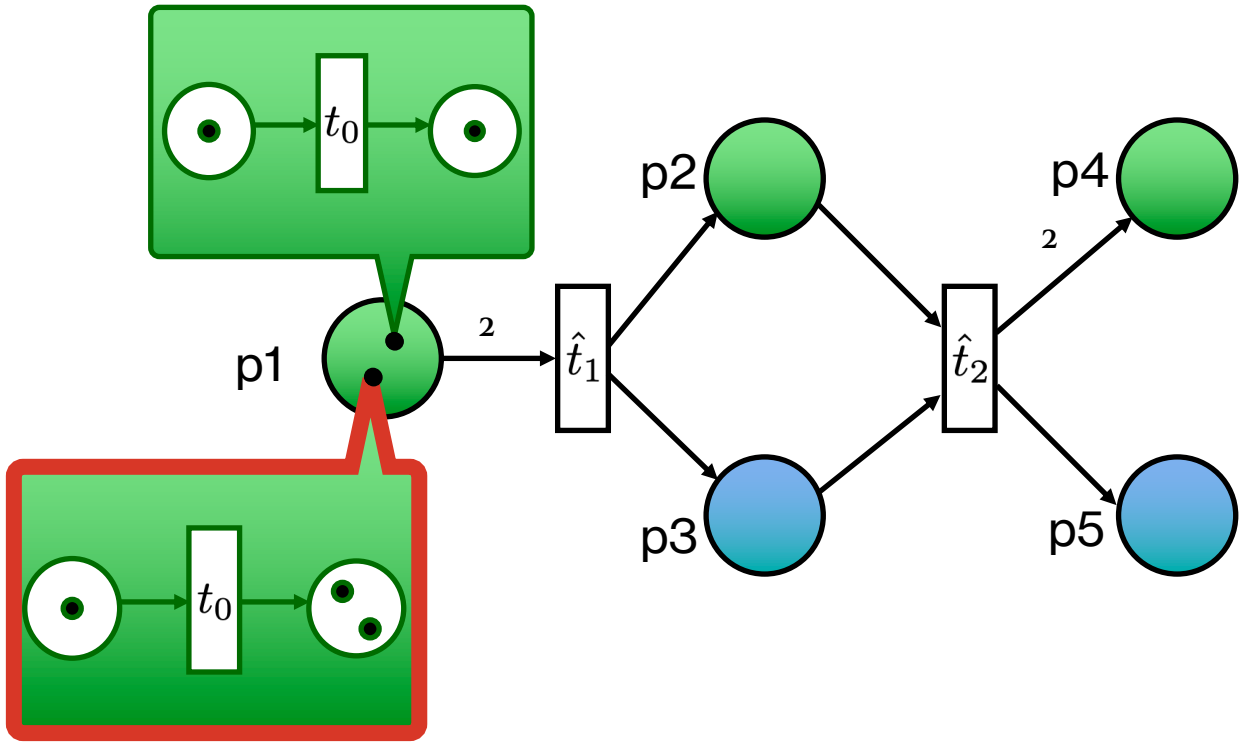
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – object autonomous events



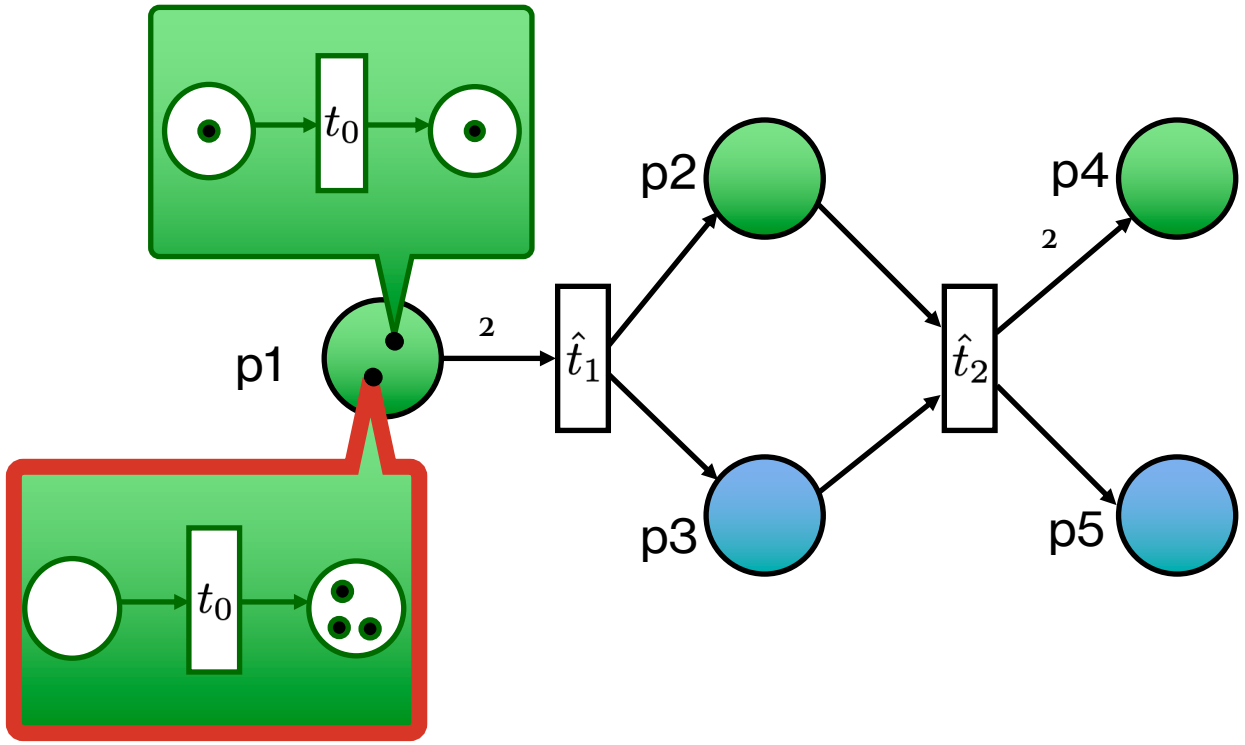
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – object autonomous events



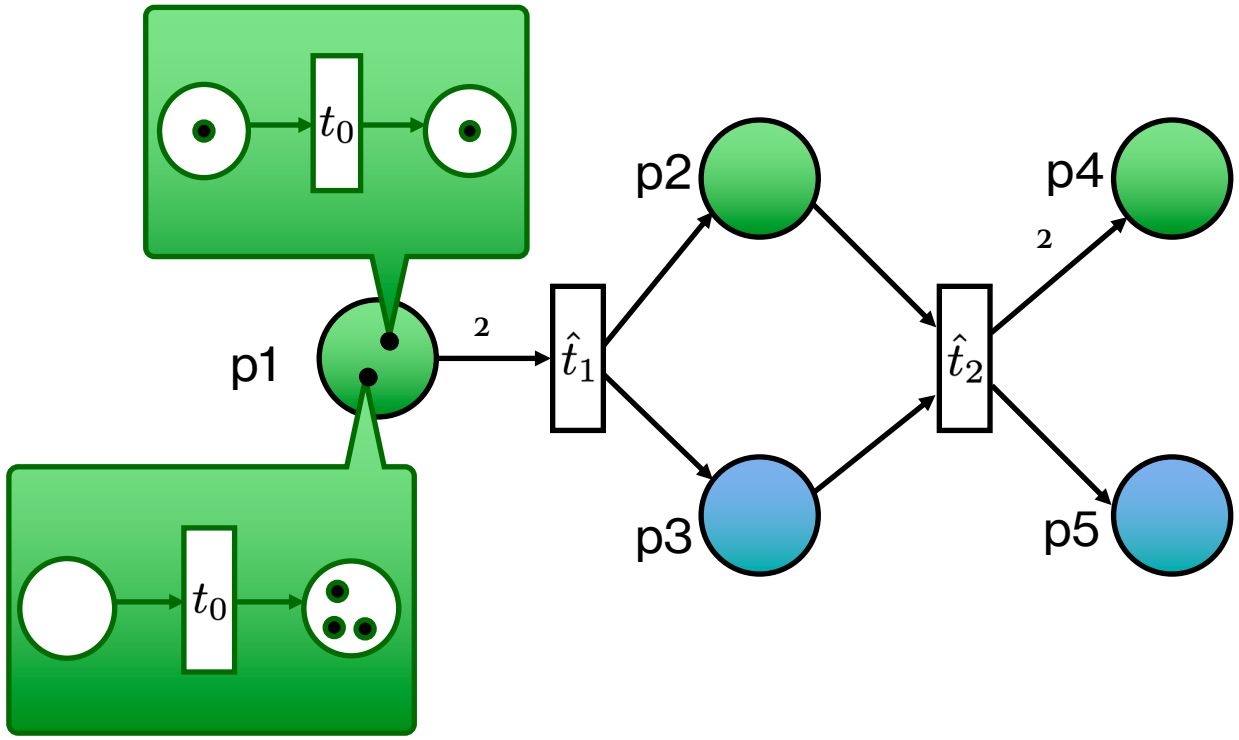
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – system autonomous events



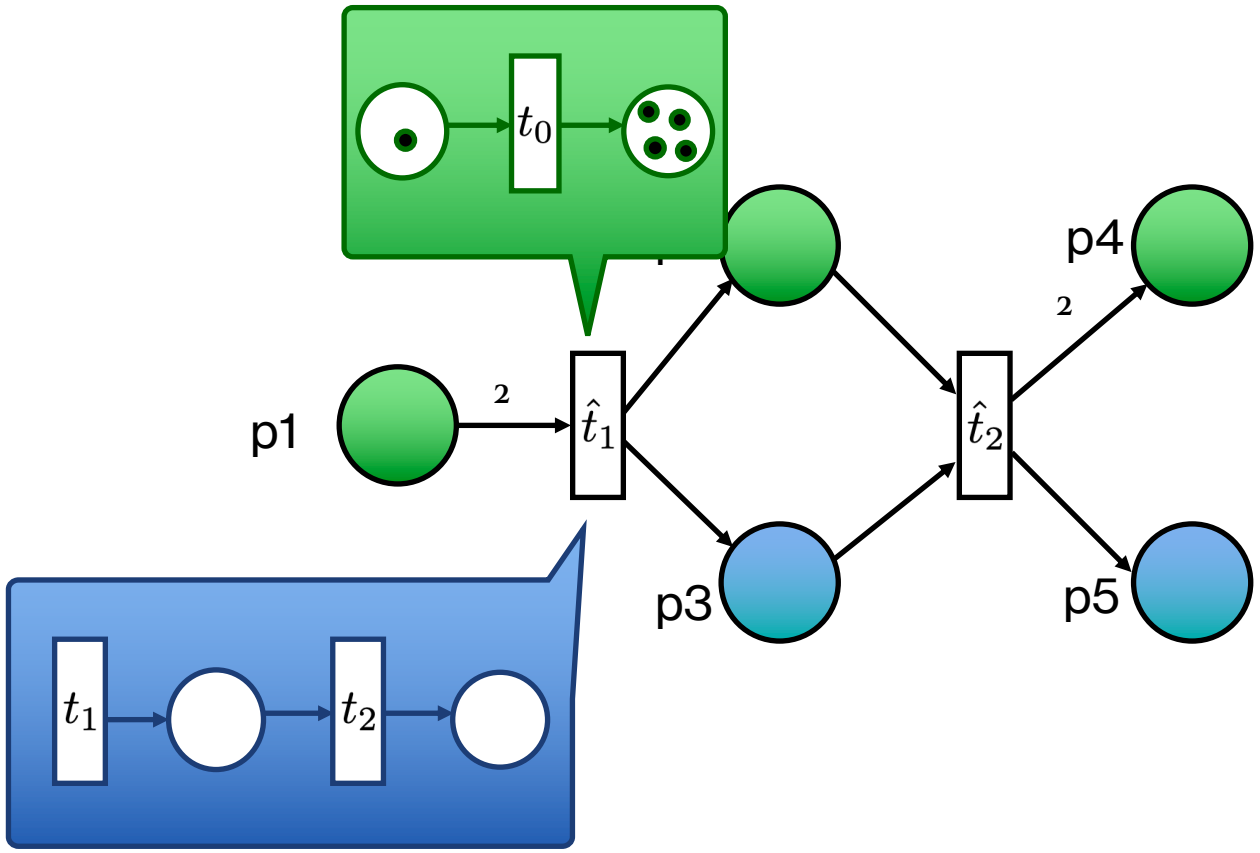
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – system autonomous events



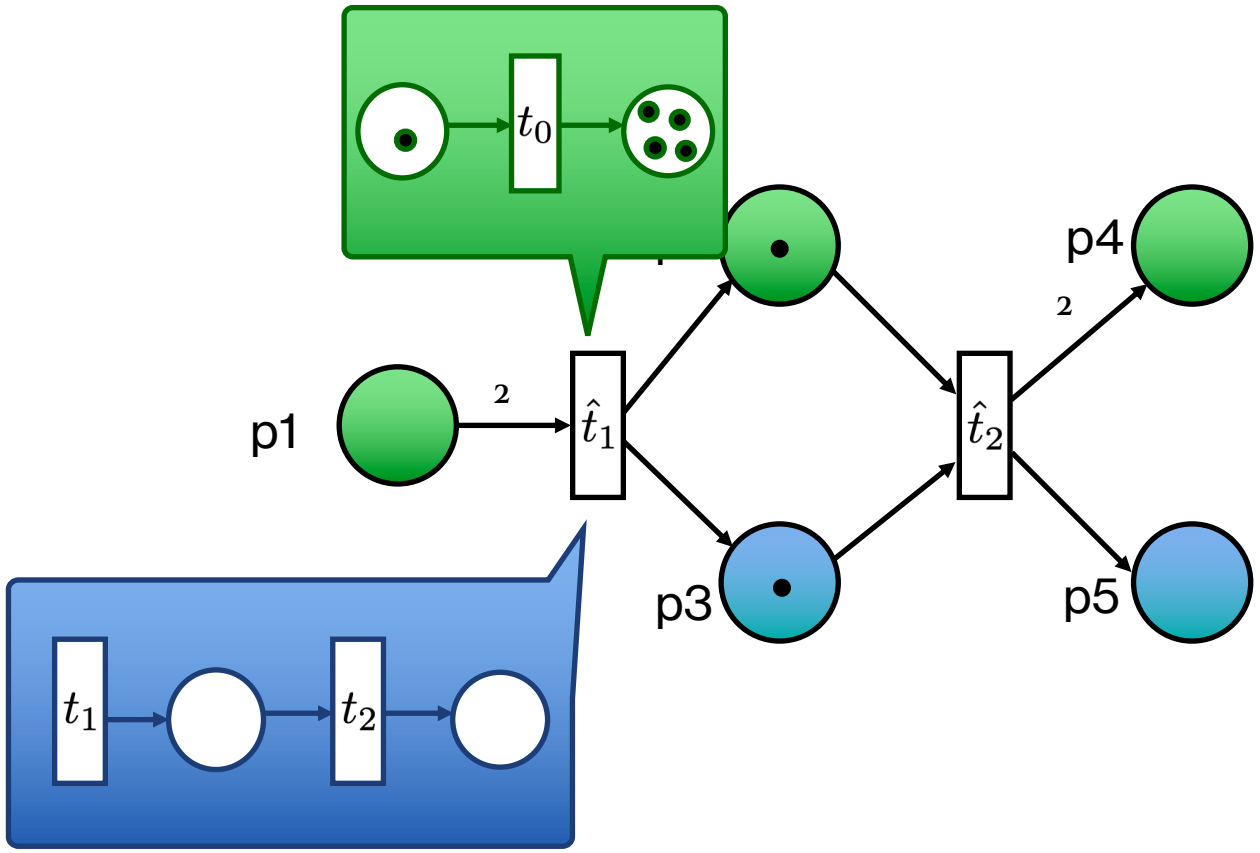
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – system autonomous events



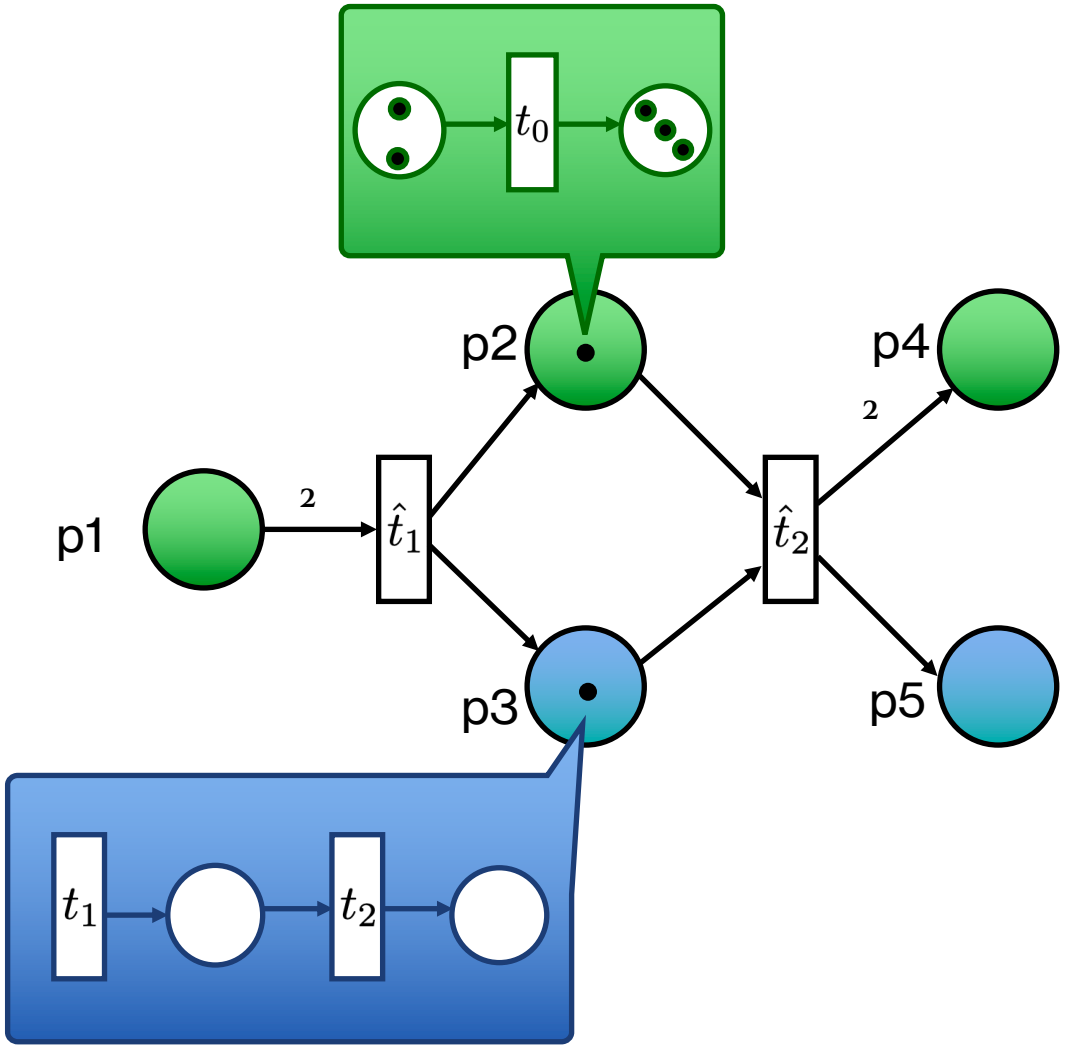
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – system autonomous events



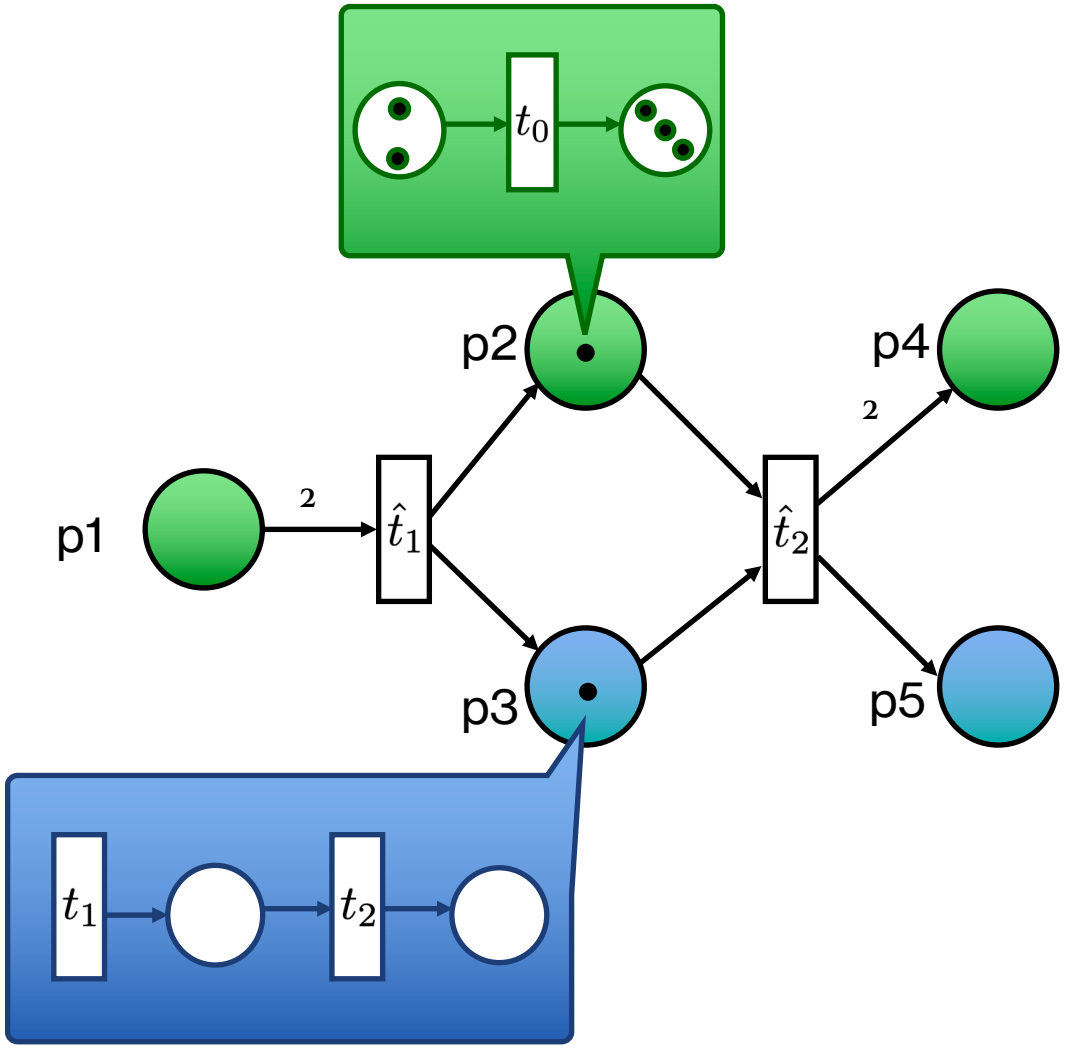
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – synchronization events



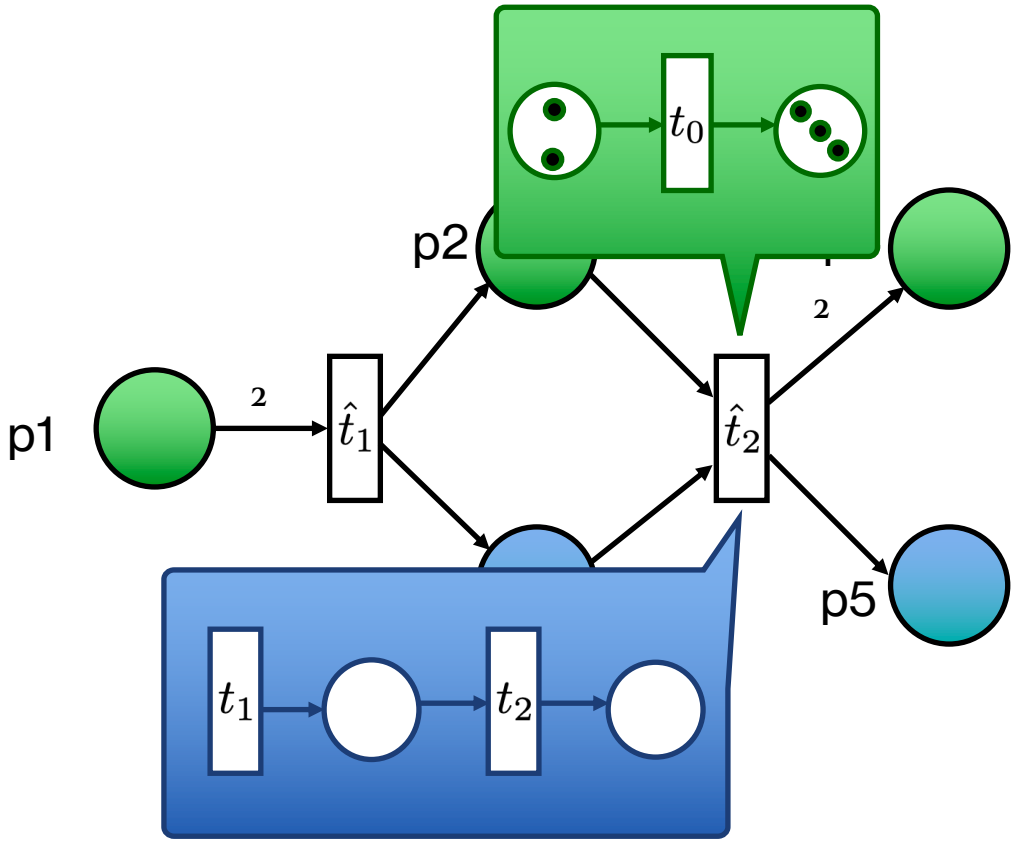
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – synchronization events



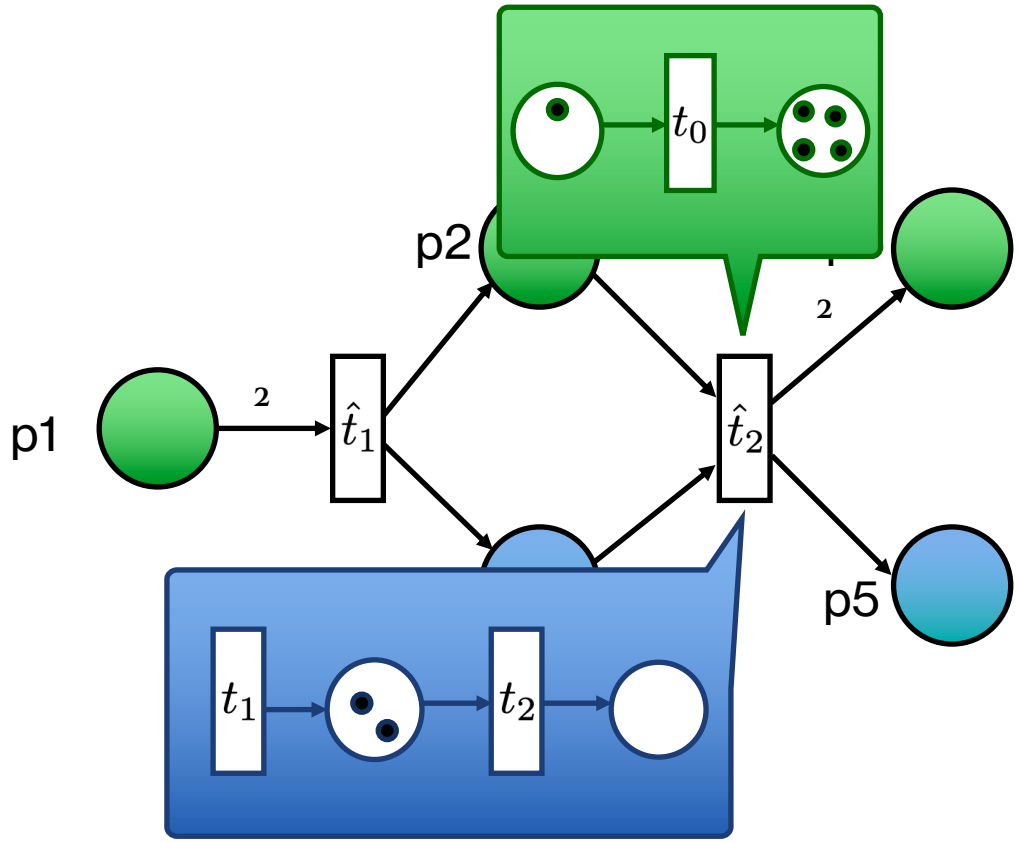
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – synchronization events



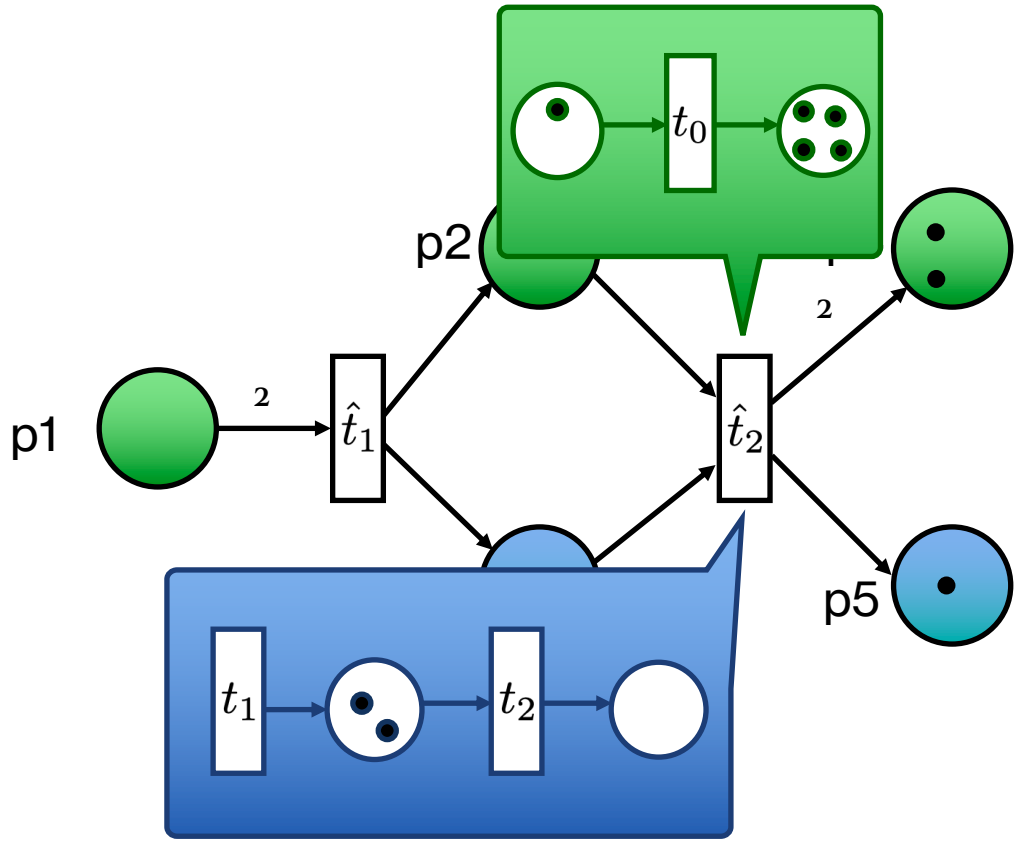
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – synchronization events



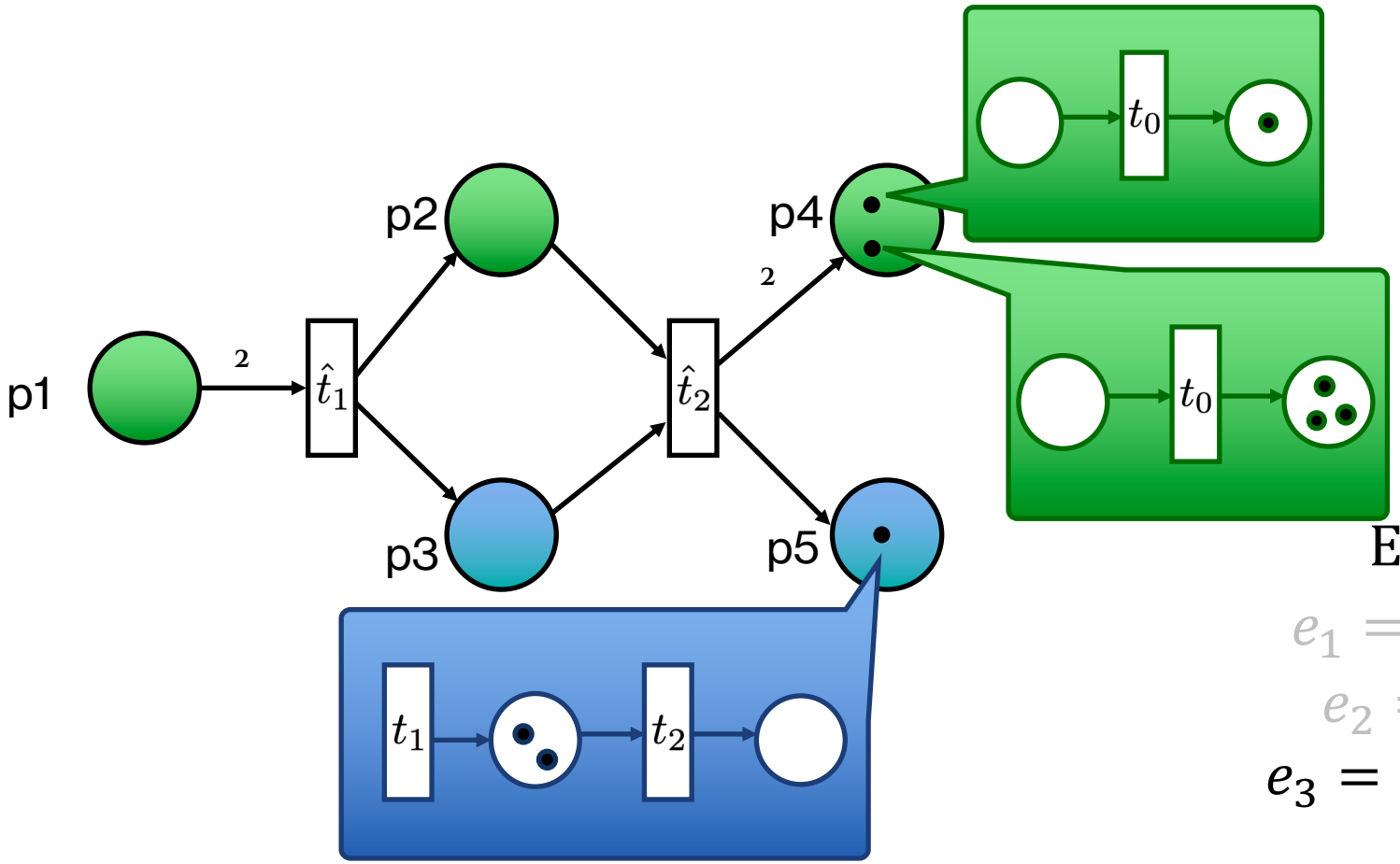
EVENTS

$e_1 = (id_{p_1}, t_0)$

$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

EOS – synchronization events



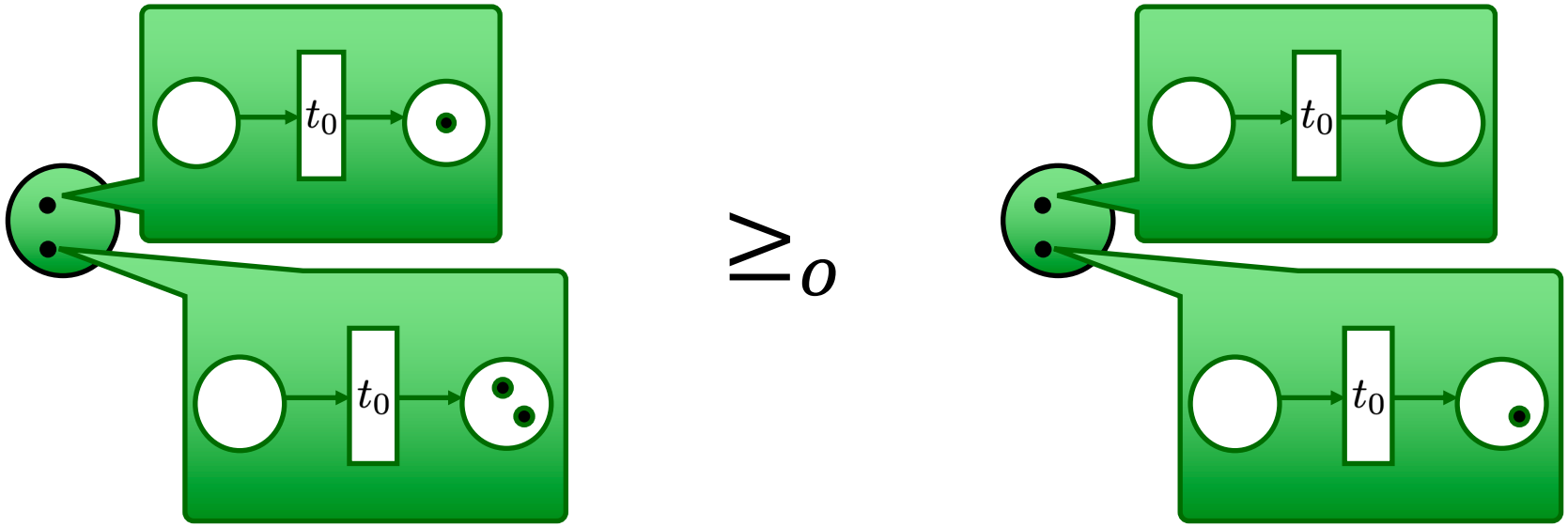
EVENTS

$e_1 = (id_{p_1}, t_0)$

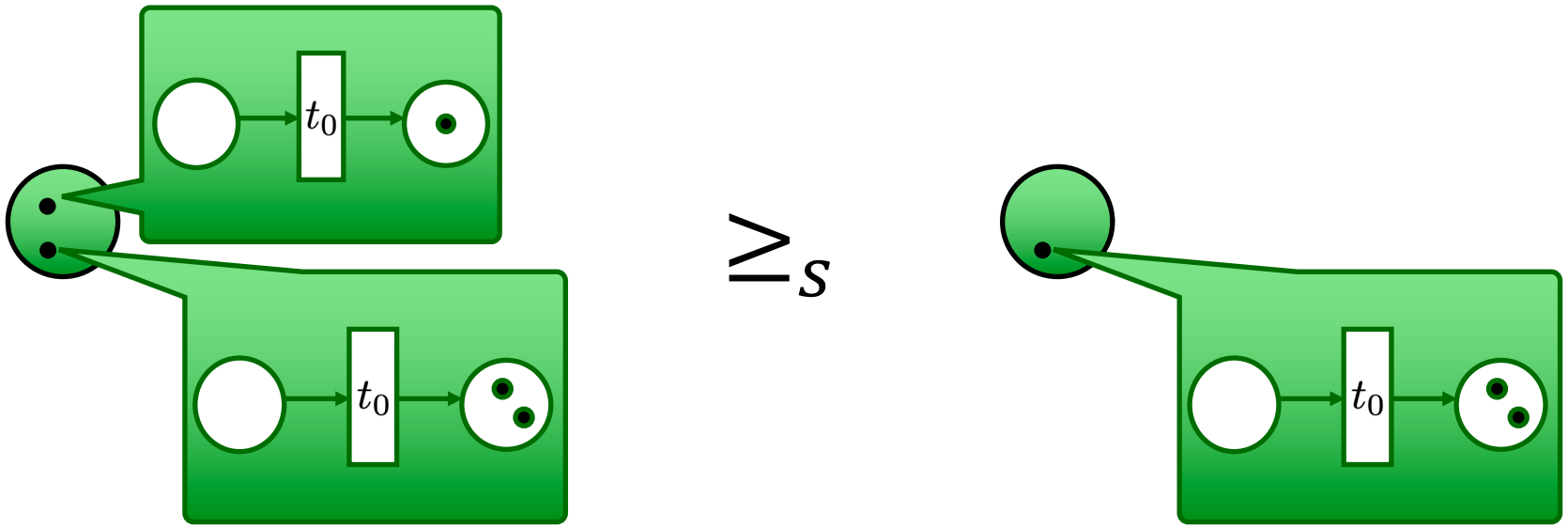
$e_2 = (\widehat{t_1}, \emptyset)$

$e_3 = (\widehat{t_2}, t_0 t_1 t_1)$

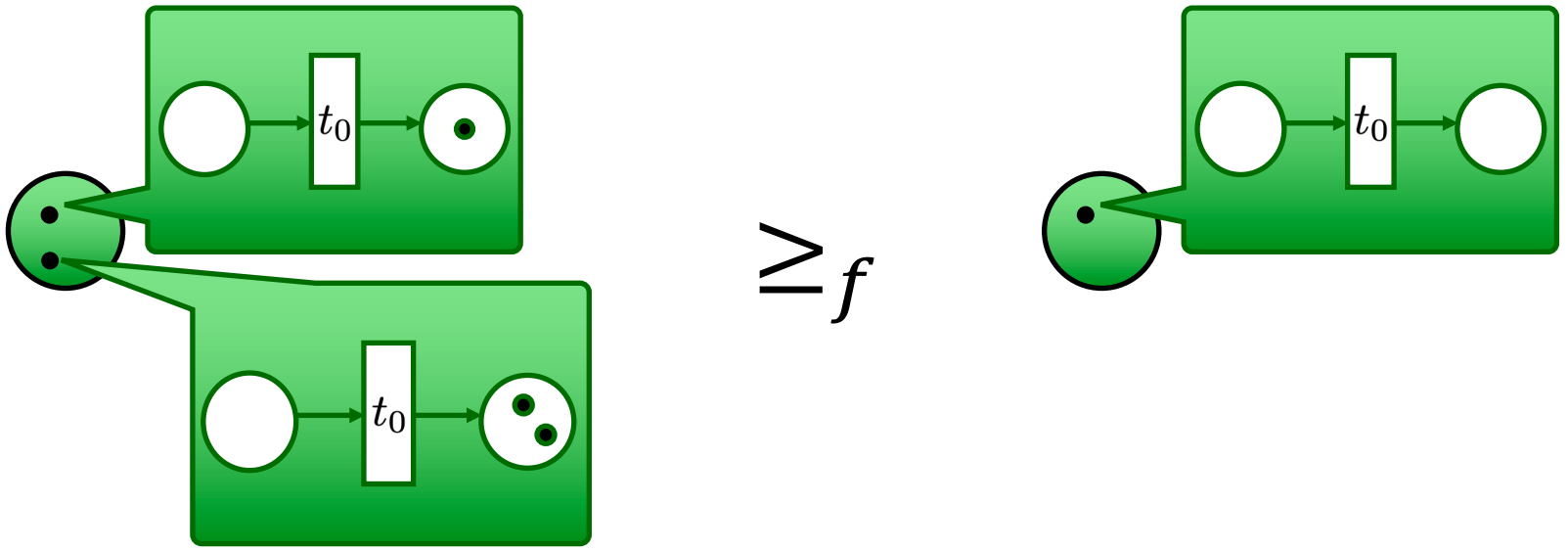
Object lossiness



System lossiness



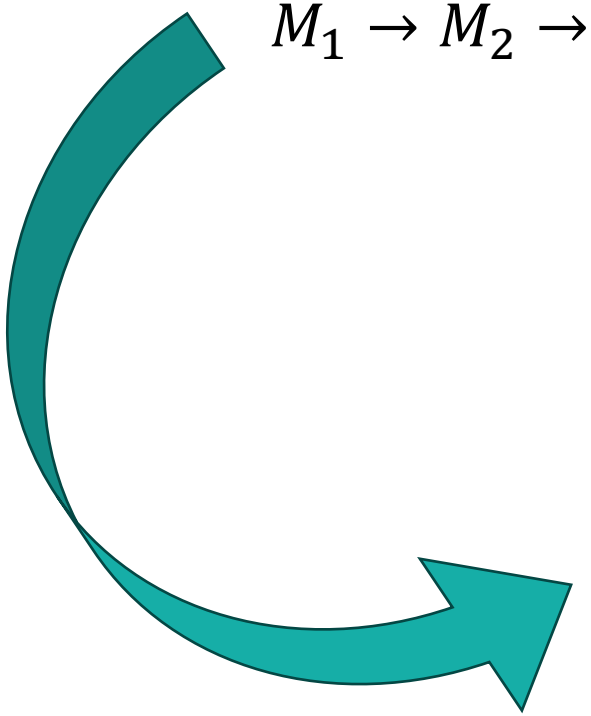
Full lossiness



$(\preccurlyeq, \ell)$-lossy runs

Perfect runs: only standard steps

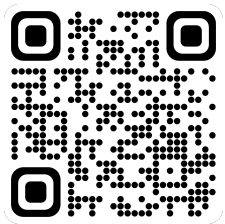
$M_1 \to M_2 \to M_3 \to \cdots$



$(\preccurlyeq, \ell)$-runs: at most $\ell \leq |\mathbb{N}|$ steps of type $\preccurlyeq$

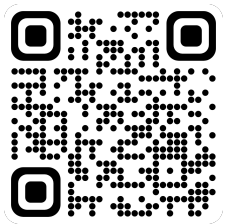
$M_1 \to M_2 \succcurlyeq M'_3 \to M'_4 \succcurlyeq M'_5 \succcurlyeq M'_6 \to M_7 \to \cdots$

$(\preccurlyeq, \ell)$-lossy problems



Is the system **robust** up to ℓ occurrences of $\preccurlyeq$?

$(\preccurlyeq, \ell)$-lossy problems



Is the system **robust** up to ℓ occurrences of $\preccurlyeq$?

$(\preccurlyeq, \ell)$-deadlock freeness

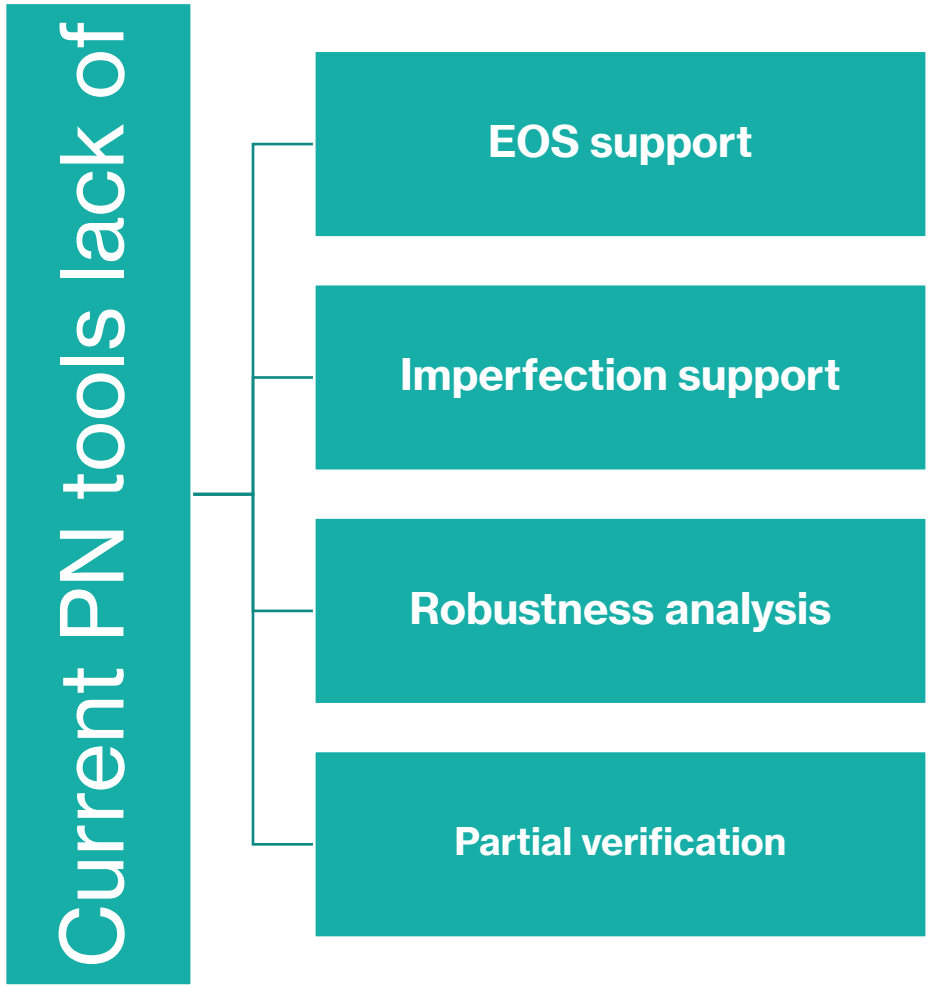
Input

An EOS E and an initial marking M.

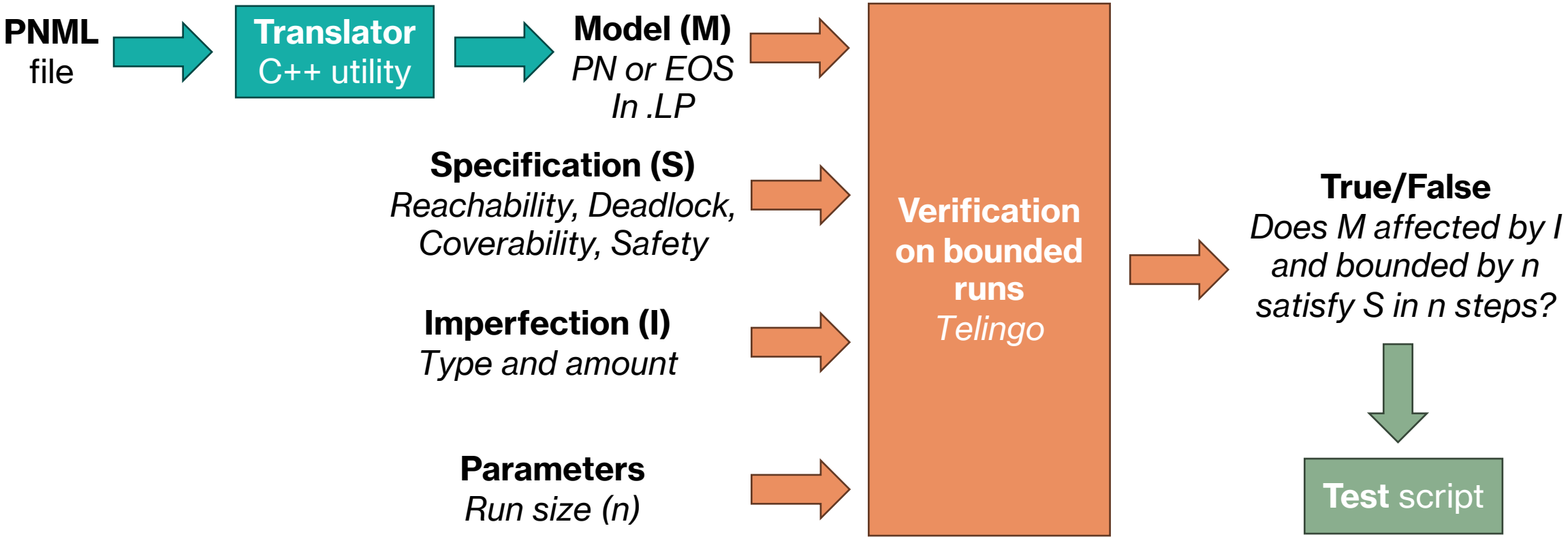
Output

Is there a $(\preccurlyeq, \ell)$-run from M to a marking where no event is enabled?

Motivation


Current PN tools lack of
EOS support
Imperfection support
Robustness analysis
Partial verification

Prototype


PNML
file
Translator
C++ utility
Model (M)
PN or EOS
In .LP
Specification (S)
Reachability, Deadlock,
Coverability, Safety
Imperfection (I)
Type and amount
Parameters
Run size (n)
Verification
on bounded
runs
Telingo
True/False
Does M affected by I
and bounded by n
satisfy S in n steps?
Test script

Why bounded verification?

Problems on general EOS	$\leq_f$	$\leq_o$	$\leq_s$
0-reach	U	U	U
0-cover	U	U	U
ℓ-reach/cover	U	U	U
ω-reach/cover	D	U	U

F. Di Cosmo, S. Mal, T. Prince, *Deciding Reachability and Coverability in Lossy EOS*, PNSE'24

Why Telingo?

Telingo is **declarative** and supports **temporal constraints**

- E.g., `:- &tel(>? (lossy >(>? lossy)` allows at most one lossy step
- The meaning of lossy is declared orthogonally to EOS specification

Telingo **returns finite runs**

- Perfectly matches bounded verification

Encoding of PNs and EOSs is **elegant in ASP**

- E.g., when compared to SMT – R. Phawade, T. Prince, S. Sheerazuddin et al., *Bounded Model Checking for Unbounded Client Server Systems,* Arxiv (2022)

Correctness and performances

Correct answers

- Checked on MCC benchmarks

Slow on PNs

- Compared with Tapaal

Prohibitively slow on EOSs

- Nesting exacerbates grounding

problem	lossiness	−imax =5 (s)	−imax =10 (s)	−imax =20 (s)	TAPAAL (s)
deadlock	none	UNSAT in 0.052	SAT in 0.622	SAT in 82.754	SAT in $5e-6$
deadlock	any	SAT in 0.009	SAT in 0.010	SAT in 0.009	NA
1-safeness	none	UNSAT in 0.010	UNSAT in 0.014	UNSAT in 0.027	UNSAT in 0
1-safeness	any	UNSAT in 0.013	UNSAT in 0.018	UNSAT in 0.032	NA

Table 1
Comparative Results with TAPAAL for the Eratosthenes-PT-010 PN from the MCC benchmarks [12].

Give it a try

NWN Telingo Analyzer on Zenodo

Translate PNs from PNML to ASP

Analyze robustness under lossiness

Replicate our tests