Towards Efficient Verification of Systems with Dynamic Process Creation

√ Hanna Klaudel, IBISC, Evry university, France
Maciej Koutny, SCS, Newcastle university, UK
Elisabeth Pelz, LACL, Paris Est university, France
Franck Pommereau, LACL, Paris Est university, France

Context and motivation

- Multi-threaded programming paradigm
  - sequential code which can be executed repeatedly in concurrent threads
  - may interact through shared data and/or rendez-vous communications
- Petri net framework
  - composition operations (process algebra structure)
  - programs: (colored) Petri nets
  - the active threads identified by differently colored tokens (thread identifiers)
    - dynamic creation
    - manipulation of data

Thread identifiers: pids

- Thread identifiers in Petri net markings:
  - The potential of accelerating the state explosion problem
  - An additional threat for the efficiency of verification

- However:
  - Thread identifiers are arbitrary (anonymous) symbols whose role is to ensure a consistent execution of each thread
  - The exact identity is irrelevant, but the relationship between identifiers may be important

Observations

- Thread identifiers may be swaped with other thread identifiers without changing the resulting execution
- Some symmetric executions may be identified

- As a result, it may be expected:
  - state explosion reduced
  - infinite state systems can sometimes be reduced to finite representations
    - allowing in turn to use model checking techniques

Contribution

- A method for an efficient verification of multi-threaded systems modelled as colored Petri nets (t-nets)
  - Its core: a marking equivalence
    - that identifies global states which have essentially isomorphic future behaviour up to renaming of thread identifiers
    - may be computed efficiently and used to aggregate nodes in the marking graph
  - Supports:
    - distributed and concurrent thread identifier generation
    - testing the relationship between tread identifiers (eg. If a thread is a descendant / a sibling of another thread, …)

Summary

- Thread identifiers generation scheme
- Petri nets and colored markings
- Graph representation of markings
- Marking equivalence
- Example

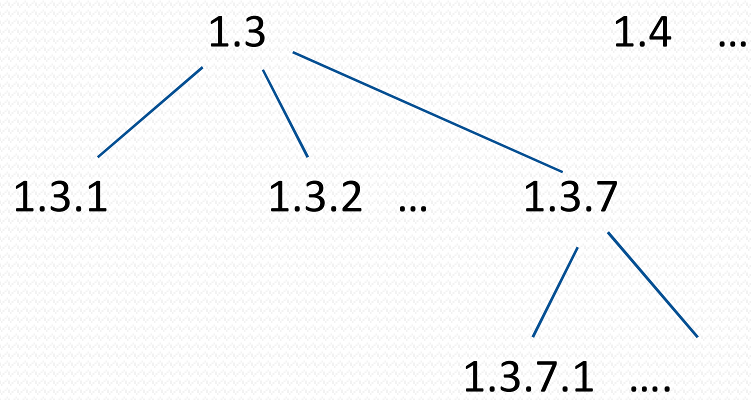
Summary

- Thread identifiers generation scheme
- Petri nets and colored markings
- Graph representation of markings
- Marking equivalence
- Example

Pid generation scheme

- Pids π, π' represented as dot-separated sequences of positive integers
  - There is a set of initial Pids
  - Each thread maintains a count g of the threads it has already spawned:

    Ex. if π=1.3 and $g(\pi)$=6 then $\nu(\pi)$=1.3.7 generates the next child of π



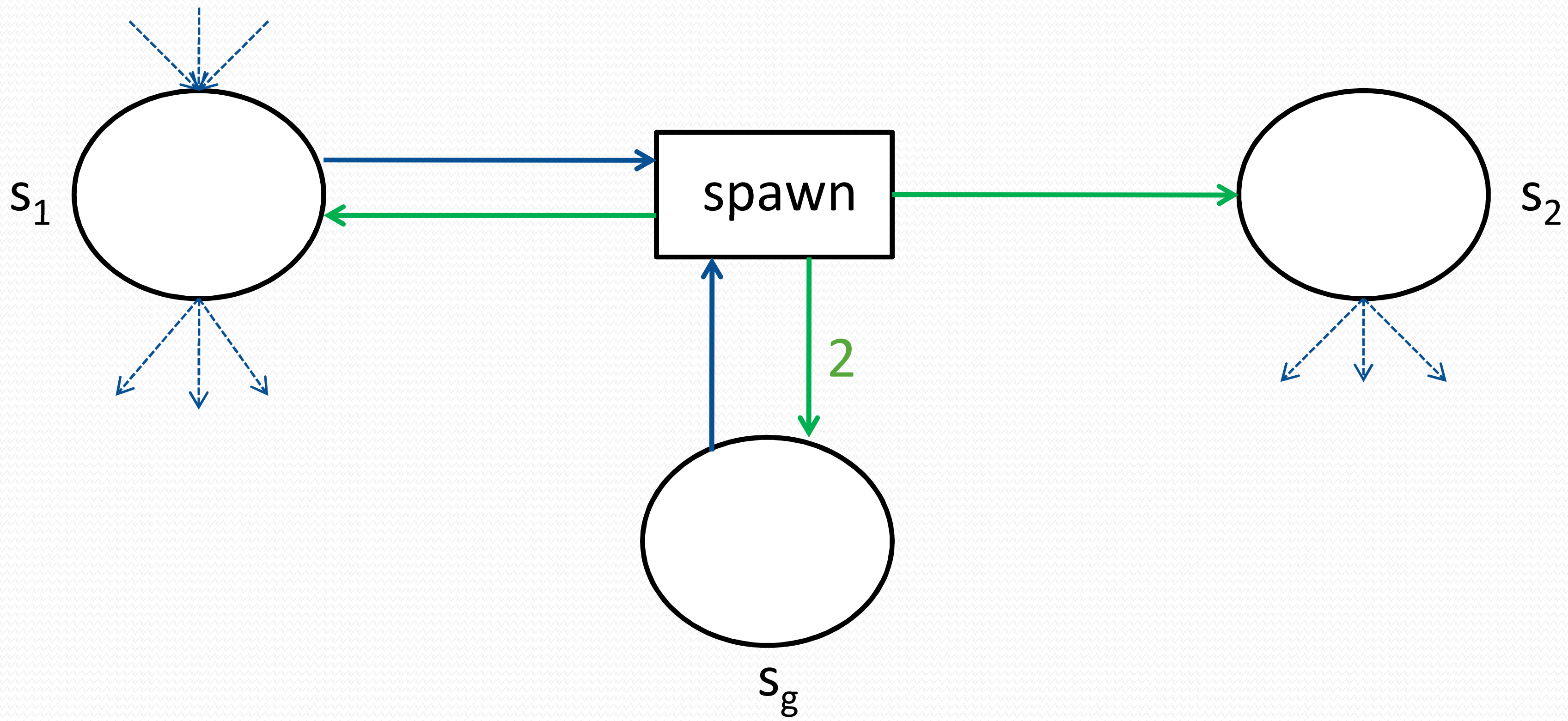
Pid generation scheme

- Operations on pids checking whether
  - $\pi <_c \pi'$: π is a parent of π' $\quad 1.3 <_c 1.3.7$
  - $\pi < \pi'$: π is an ancestor of π' $\quad 1.3 < 1.3.7.1$
  - π is a sibling of π' $\quad 1.3.7 <_b 1.3.8$
  - …

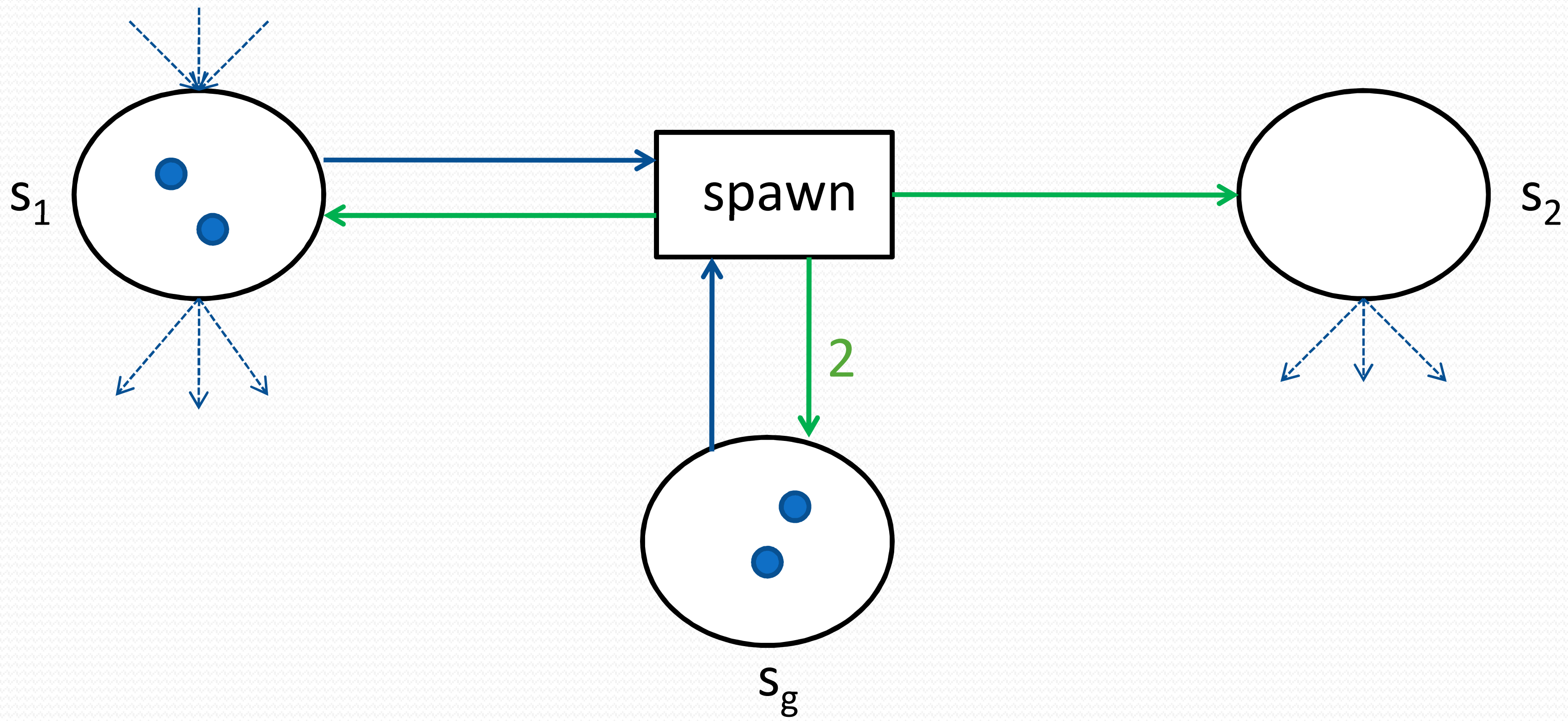
Summary

- Thread identifiers generation scheme
- Petri nets and colored markings
- Graph representation of markings
- Marking equivalence
- Example

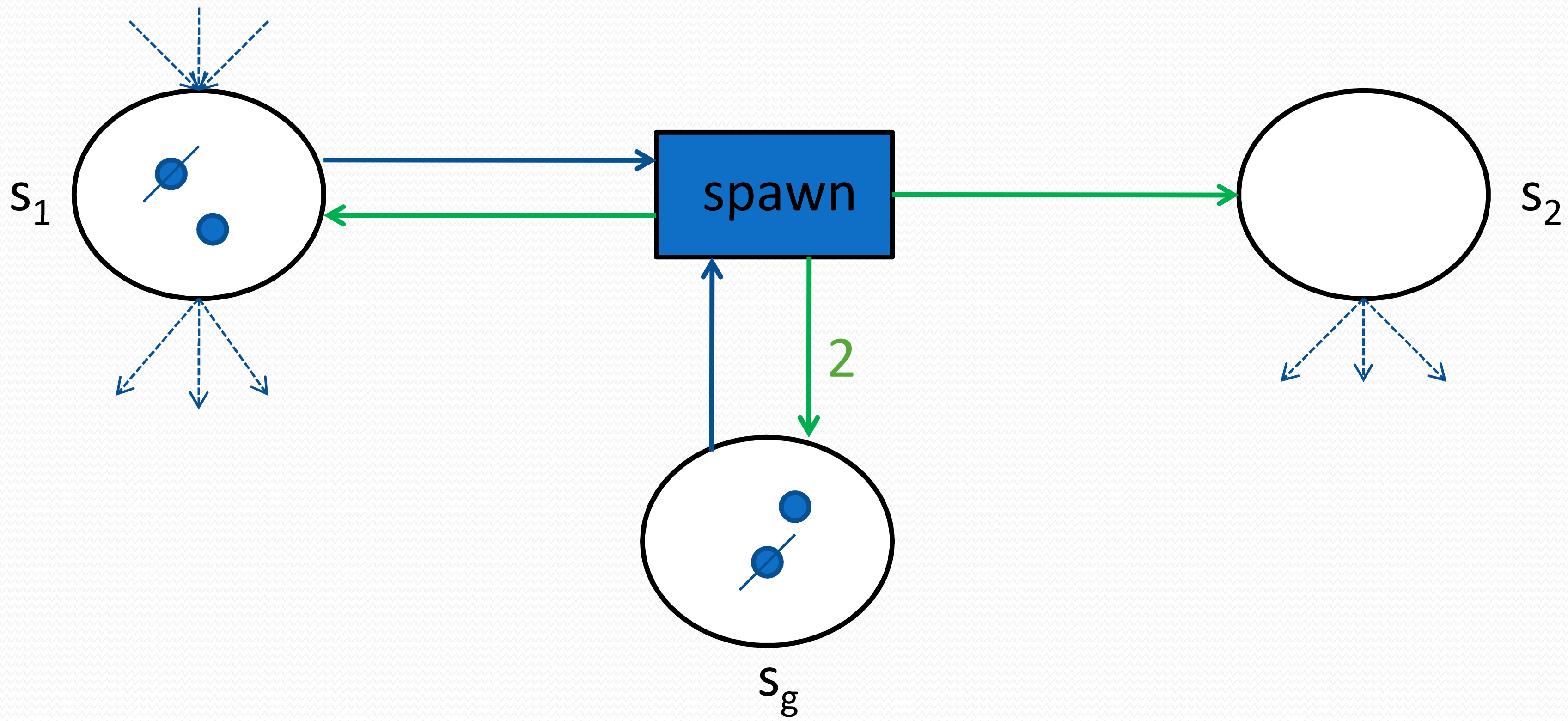
Petri nets



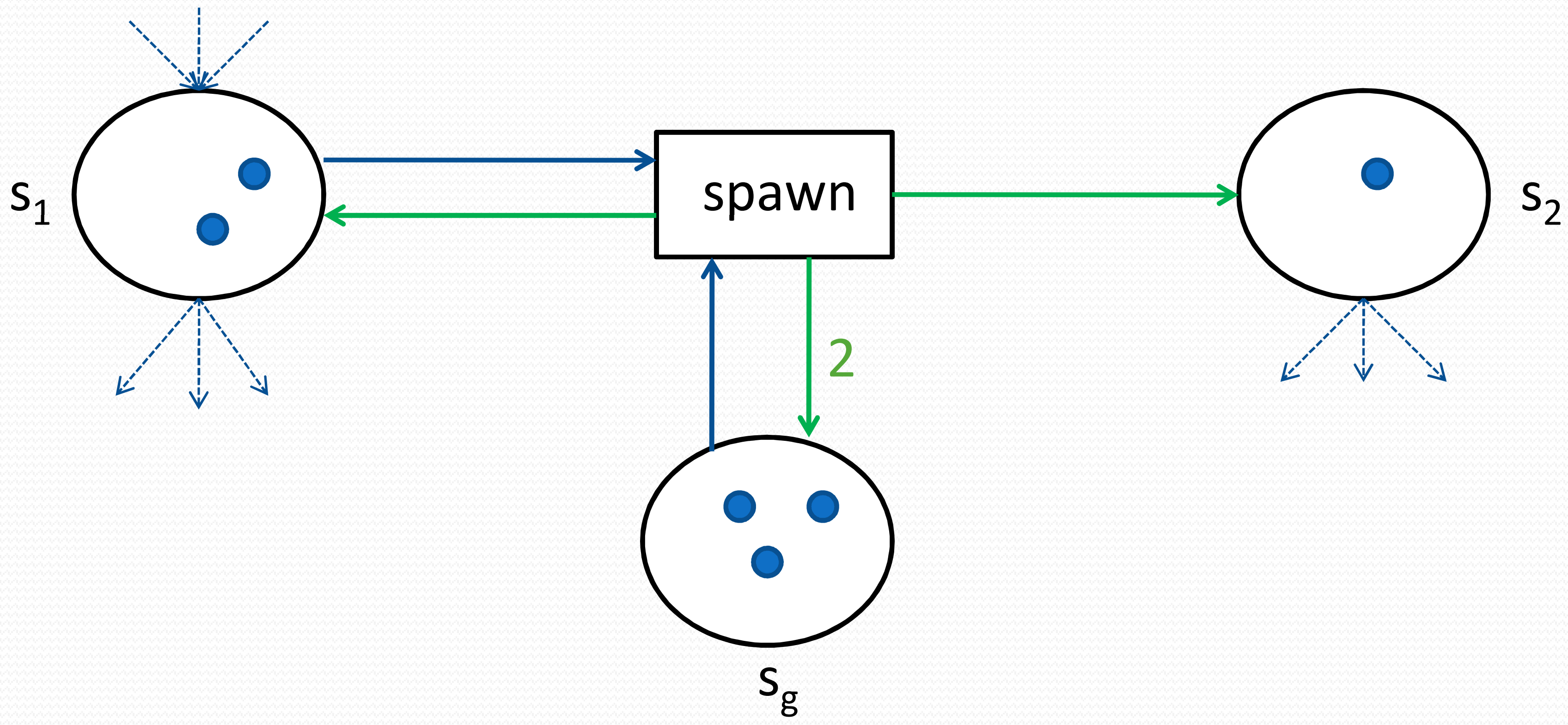
Petri nets



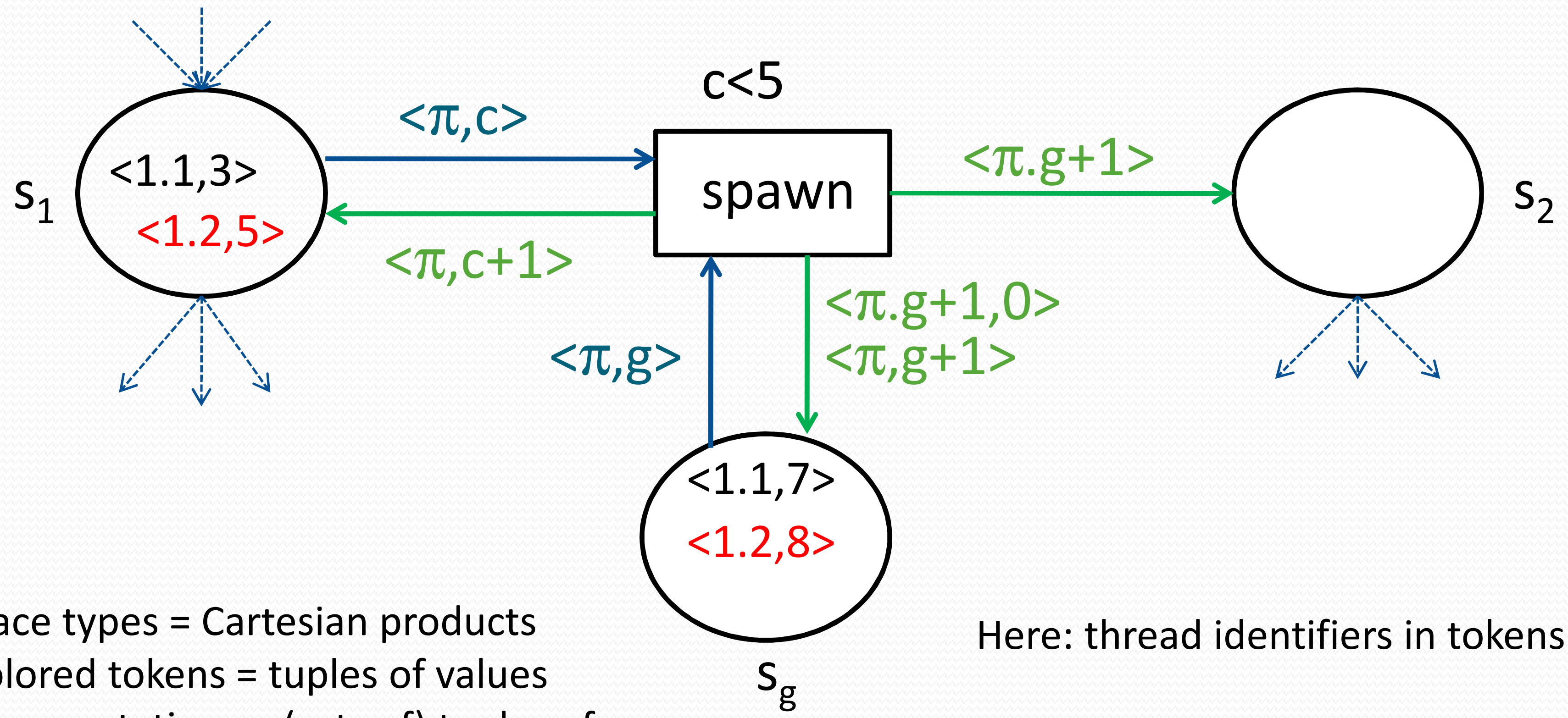
Petri nets



Petri nets



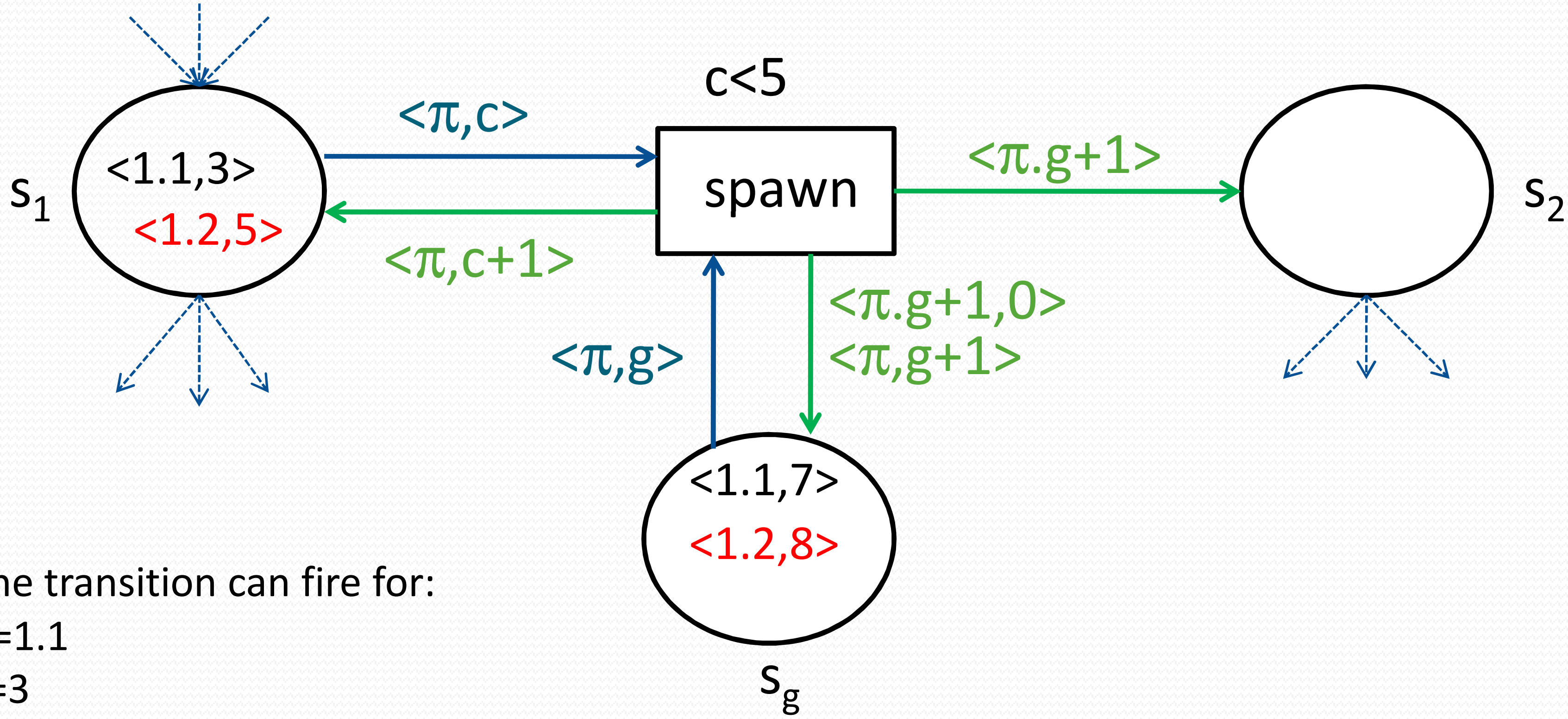
Colored Petri nets (t-nets)



Place types = Cartesian products
Colored tokens = tuples of values
Arc annotations = (sets of) tuples of variables
Transition guards = Boolean expressions

Here: thread identifiers in tokens

Colored Petri nets (t-nets)



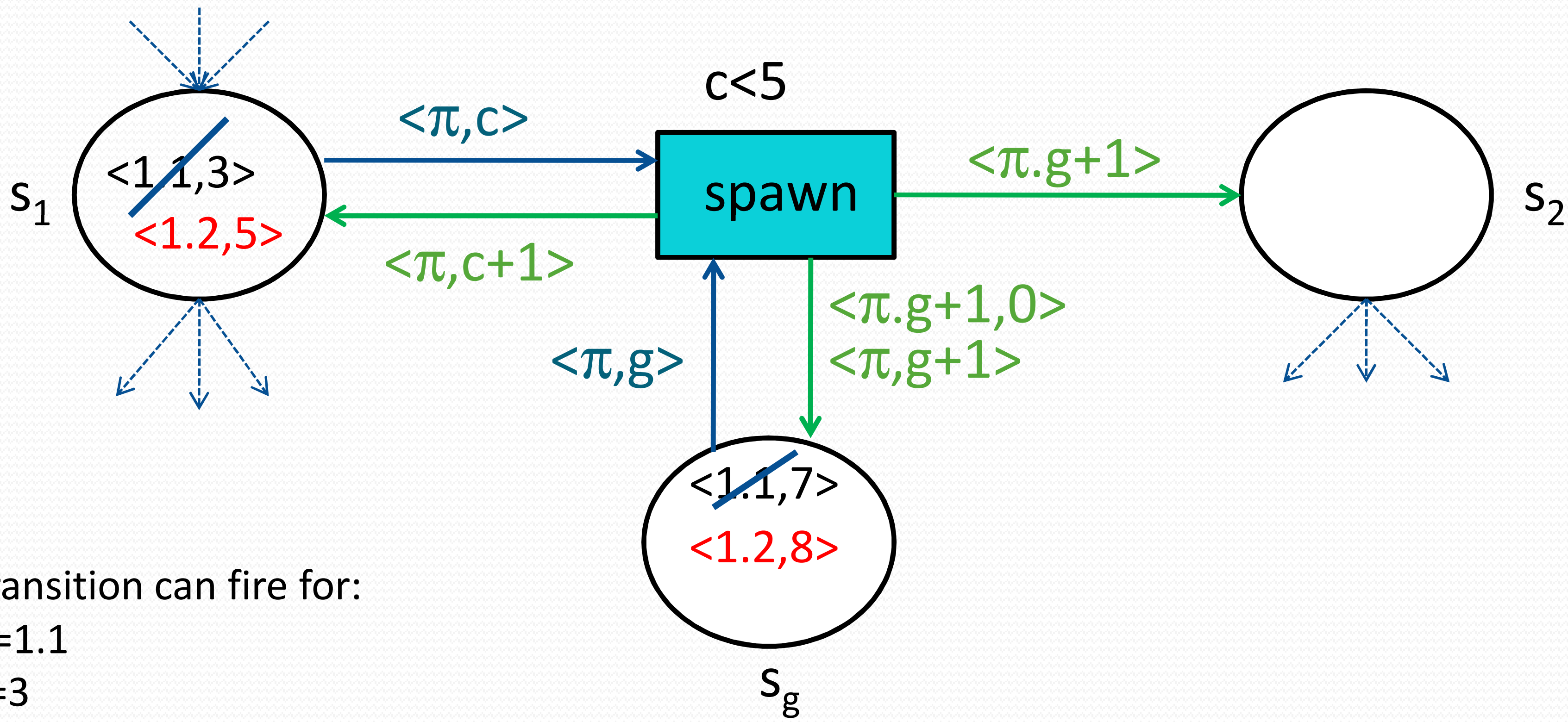
The transition can fire for:

π=1.1

c=3

g=7

Colored Petri nets (t-nets)



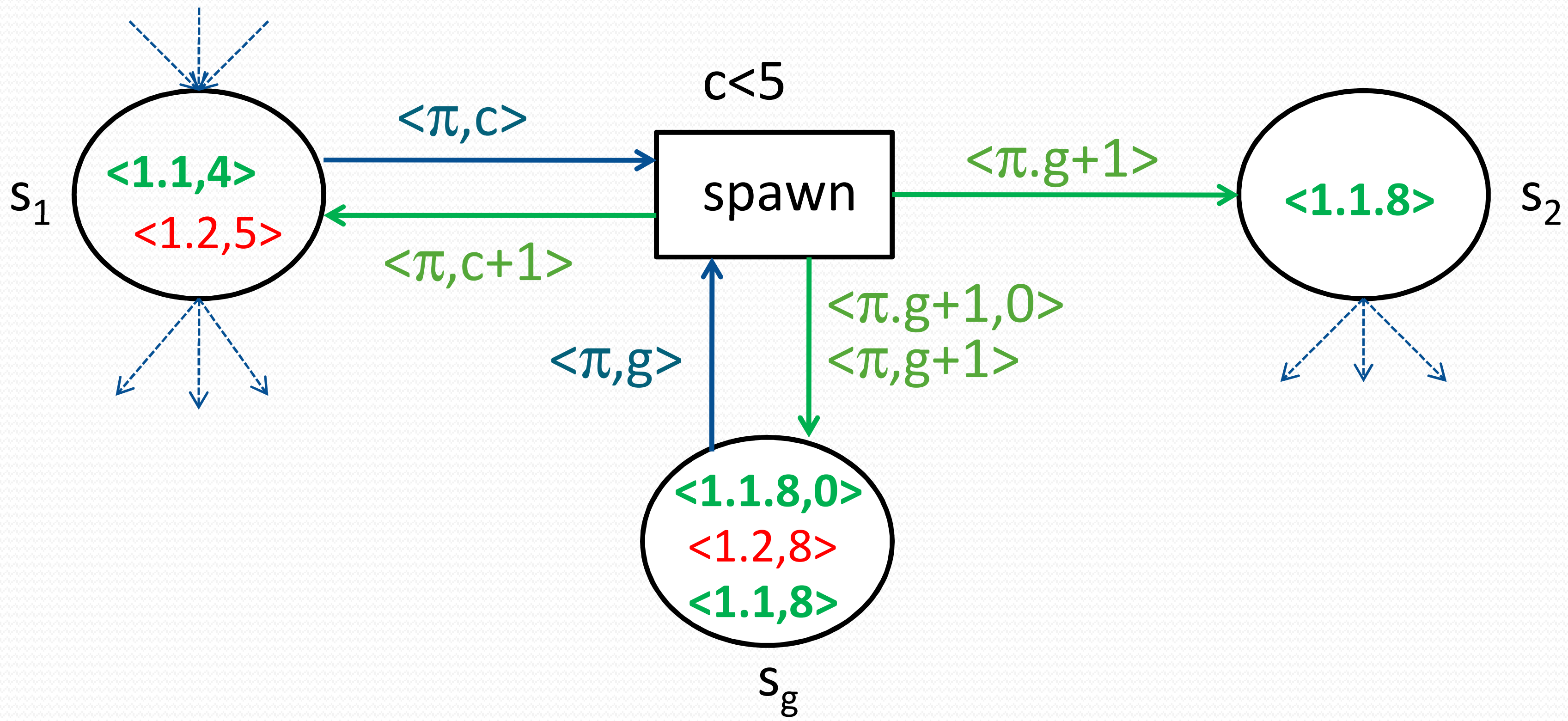
Transition can fire for:

π=1.1

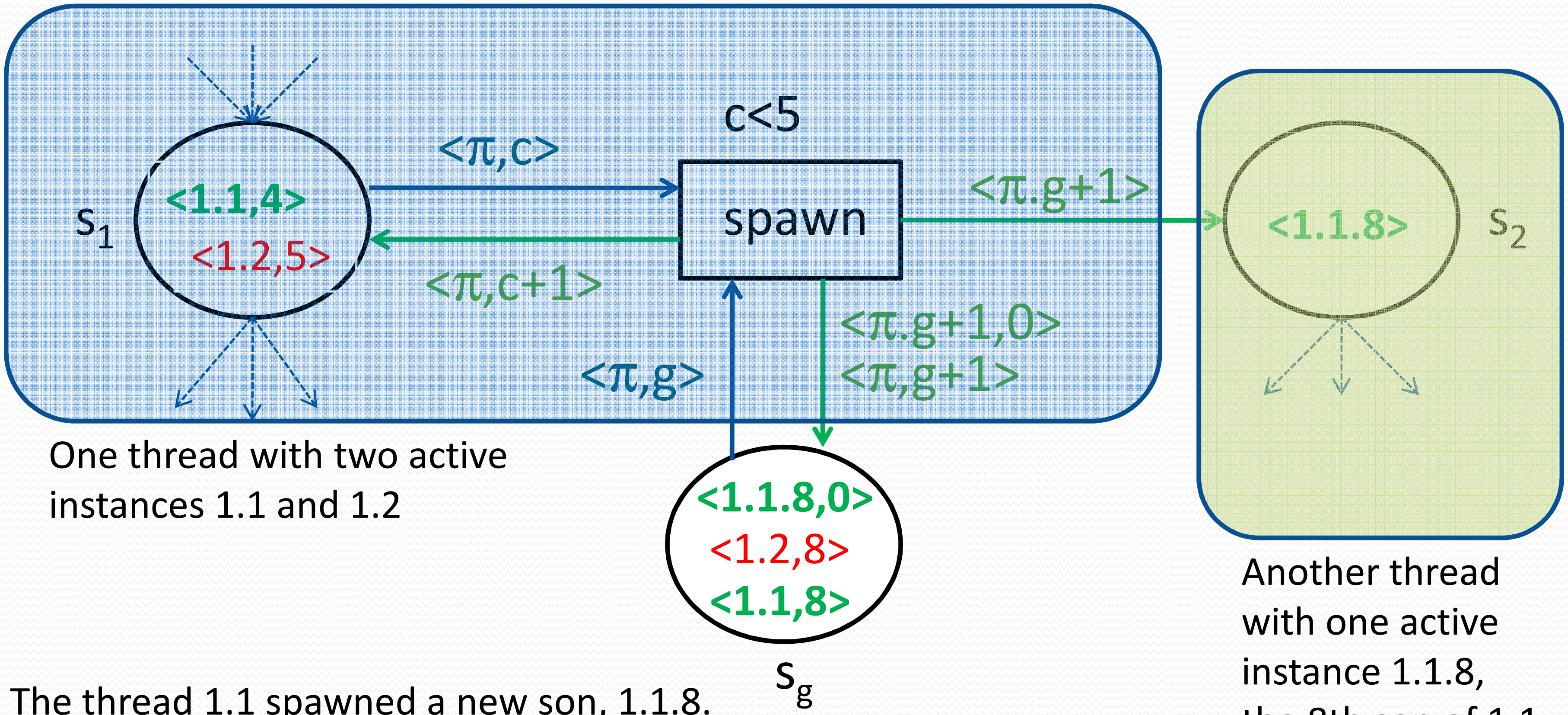
c=3

g=7

Colored Petri nets (t-nets)



Intuitive meaning of the marking



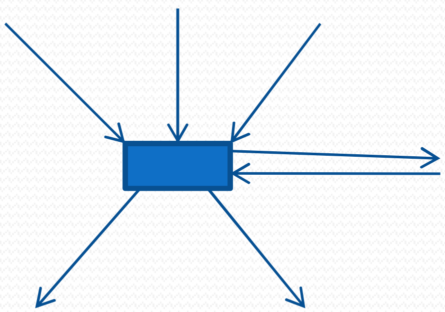
One thread with two active instances 1.1 and 1.2

Another thread with one active instance 1.1.8, the 8th son of 1.1

The thread 1.1 spawned a new son, 1.1.8.
g(1.1) becomes 8
g(1.1.8) is 0

Main characteristics of t-nets

- Place types:
  - Generator place P×N
    - Tokens like <1.1,3>
  - Data or control-flow place P× P×…×P×D×…×D
    - Tokens like <1.1, 1.2.5, 6, 8> or simply <1.1>
- Transition guards and arc inscriptions: general
- Syntactic restrictions on transition input/output



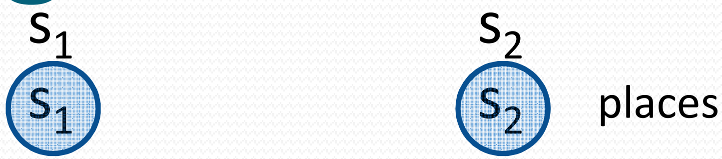
Main characteristics of t-nets

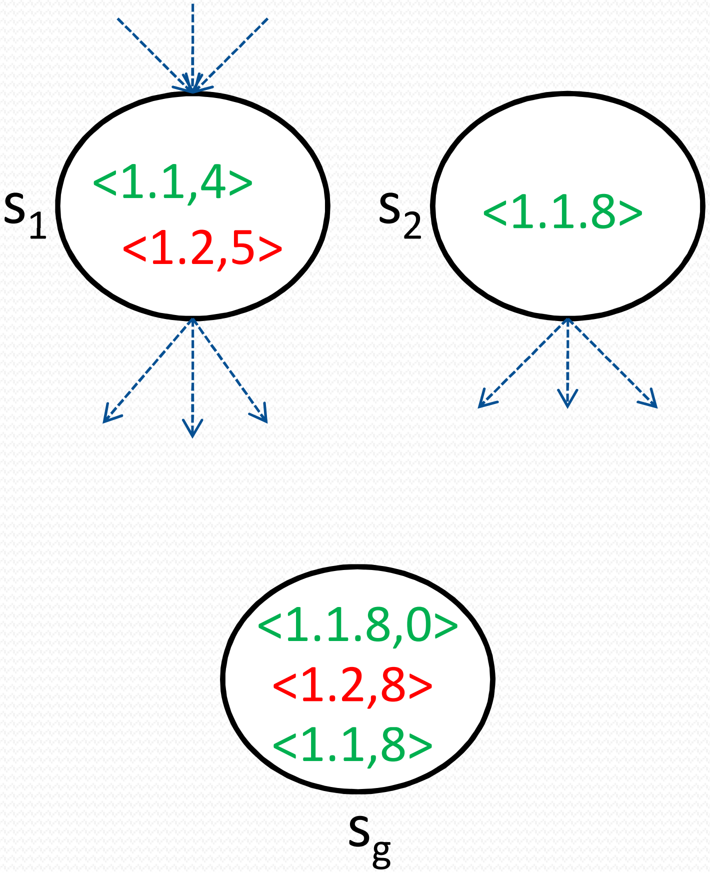
- Assumptions on initial marking:
    - All data places empty,
    - The generator place contains exactly <1, 0>
    - There is exactly one control-flow place marked, it contains <1>
- Property of t-nets:
    - The markings are control-safe (sequential threads and no duplication of control-flow tokens):
        - Exactly one token owned by π in the generator place and exactly one token owned by π in a control-flow place
        - Or tokens owned by π appear only in data places

Summary

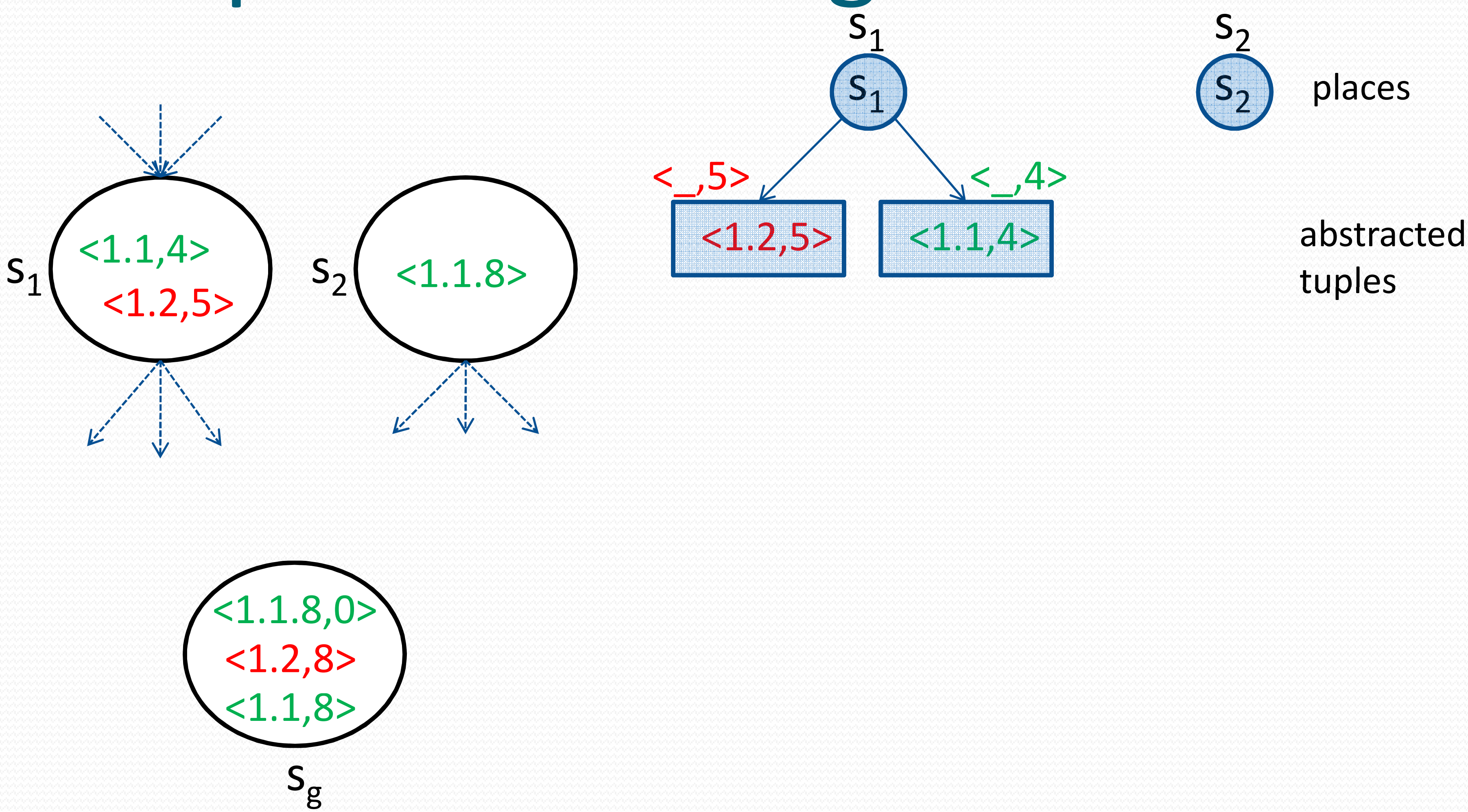
- Thread identifiers generation scheme
- Petri nets and colored markings
- Graph representation of markings
- Marking equivalence
- Example

Graphs of markings

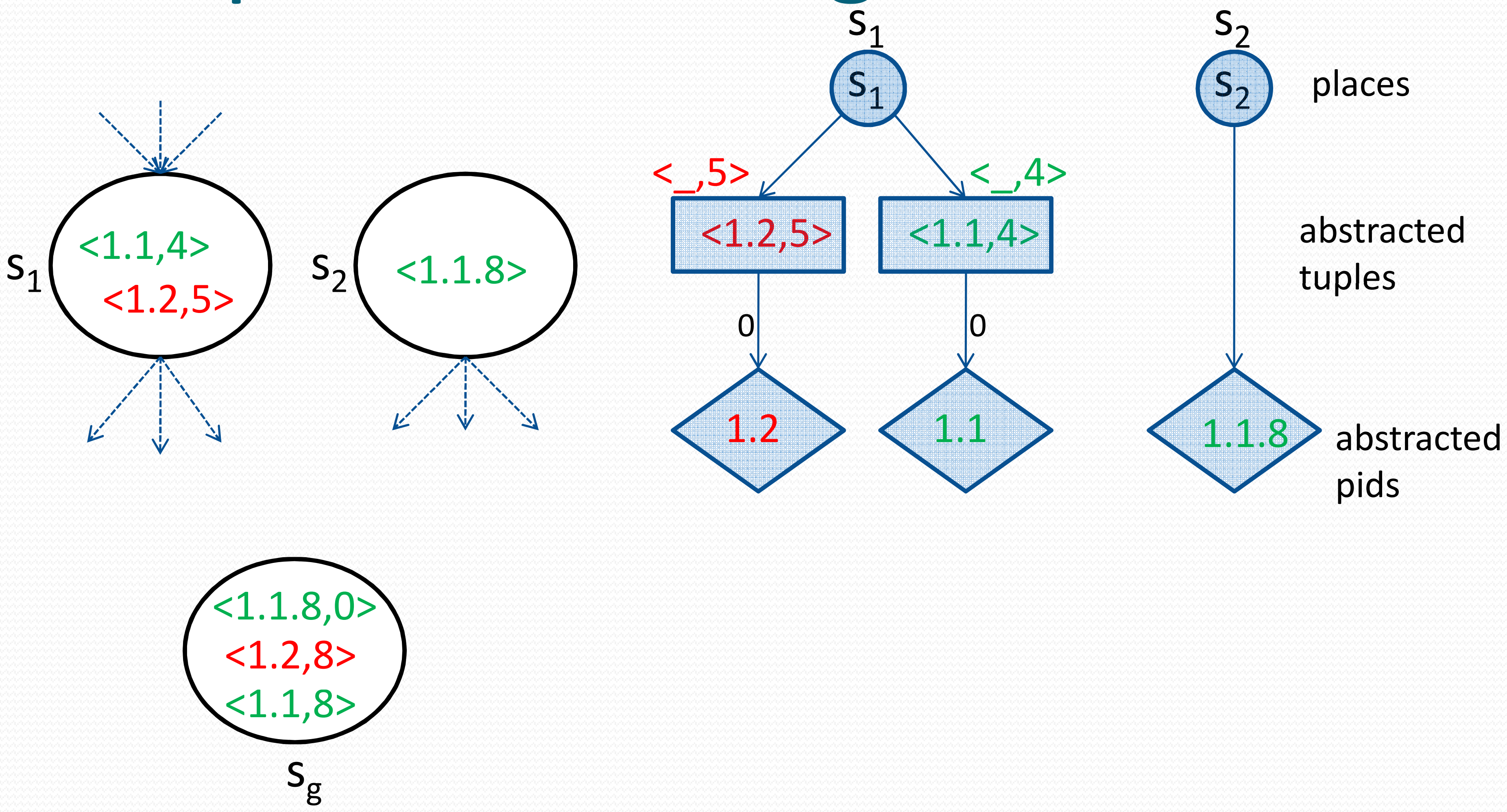




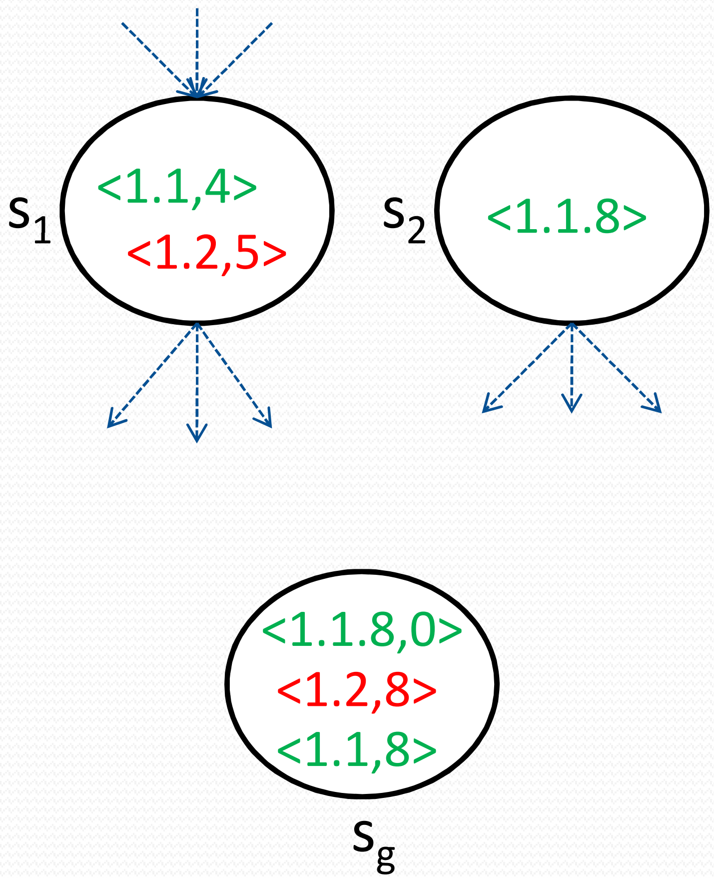
Graphs of markings

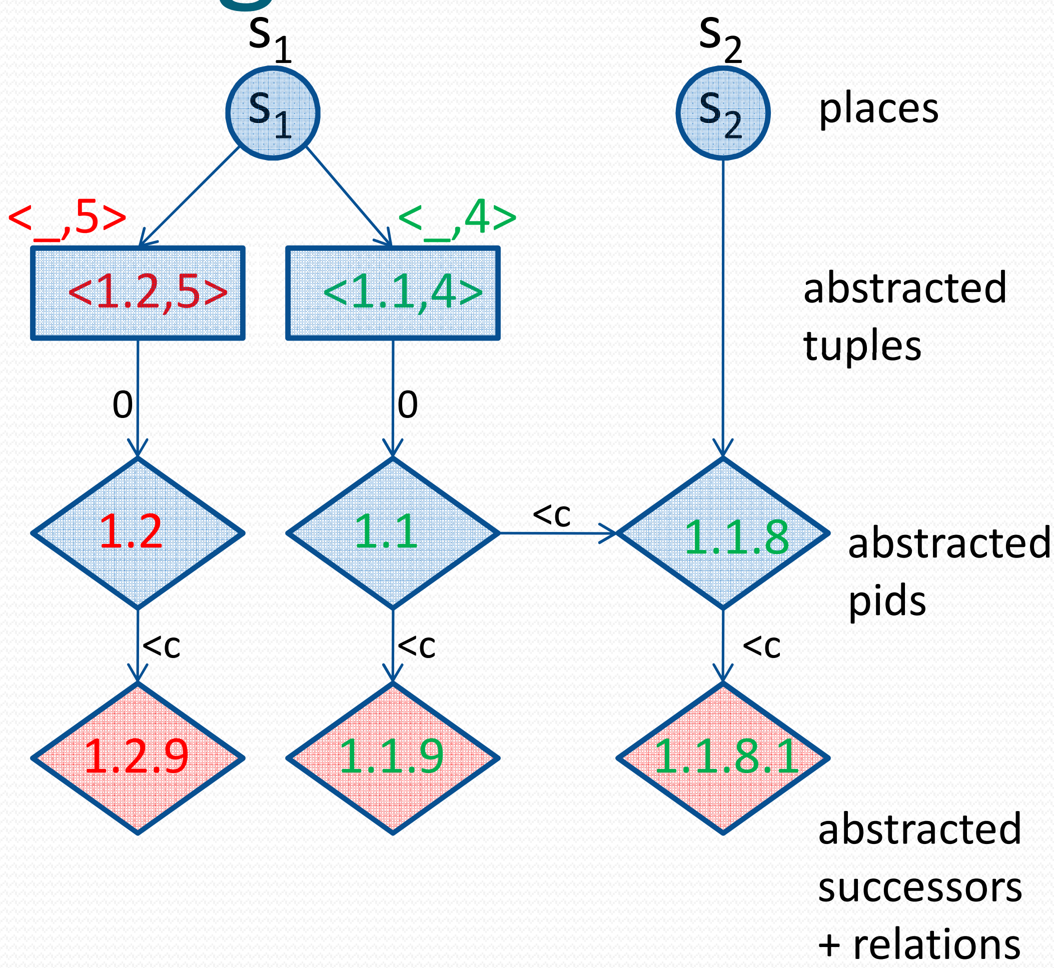


Graphs of markings

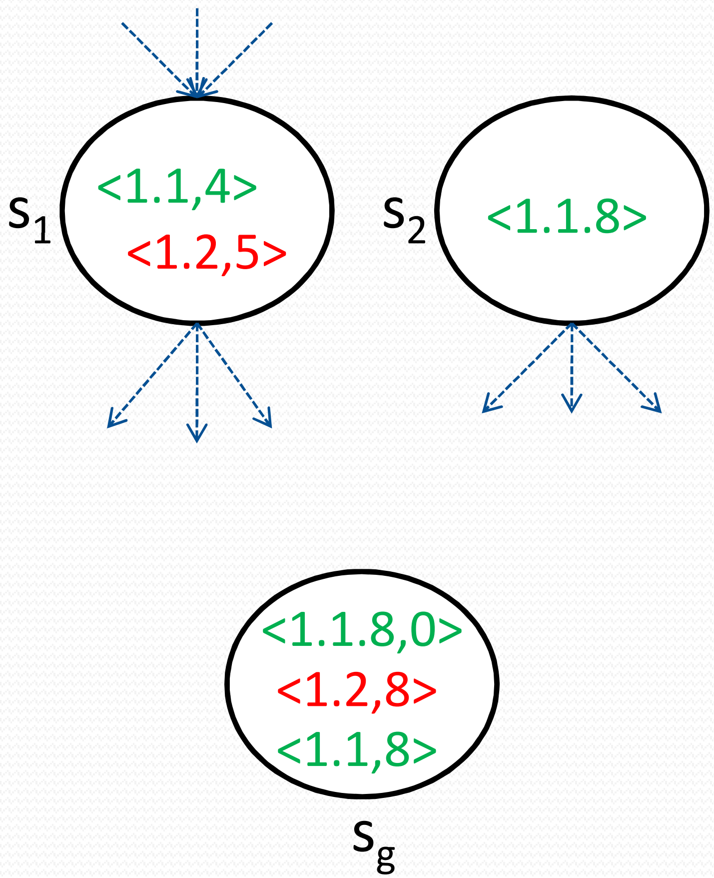


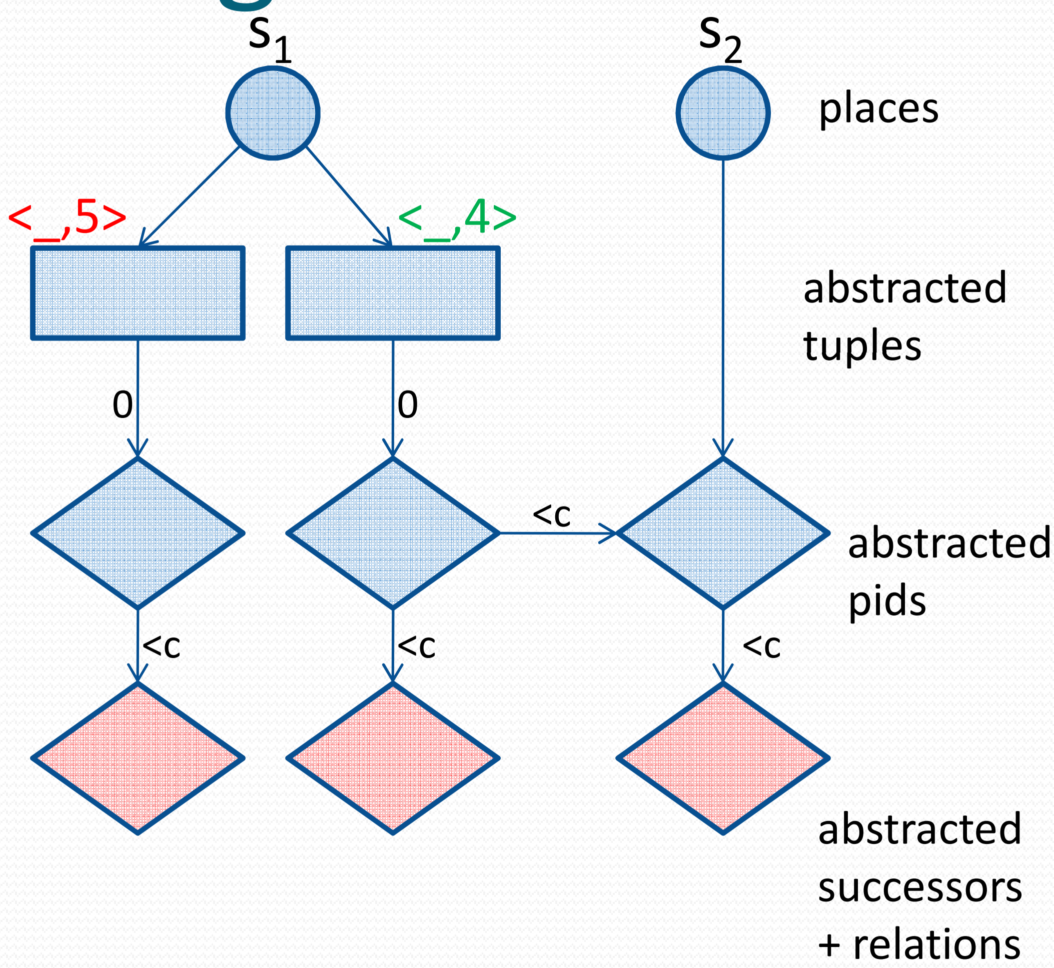
Graphs of markings





Graphs of markings





Summary

- Thread identifiers generation scheme
- Petri nets and colored markings
- Graph representation of markings
- Marking equivalence
- Example

Marking equivalence

- Let M and M’ be two reachable markings of a t-net
- Intuitively, M and M’ are **equivalent** if they represent global states of the system which have essentially isomorphic future behavior up to renaming of thread identifiers.
- Theorem:

M and M’ are equivalent

iff

Graph(M) and Graph(M’) are isomorphic

Summary

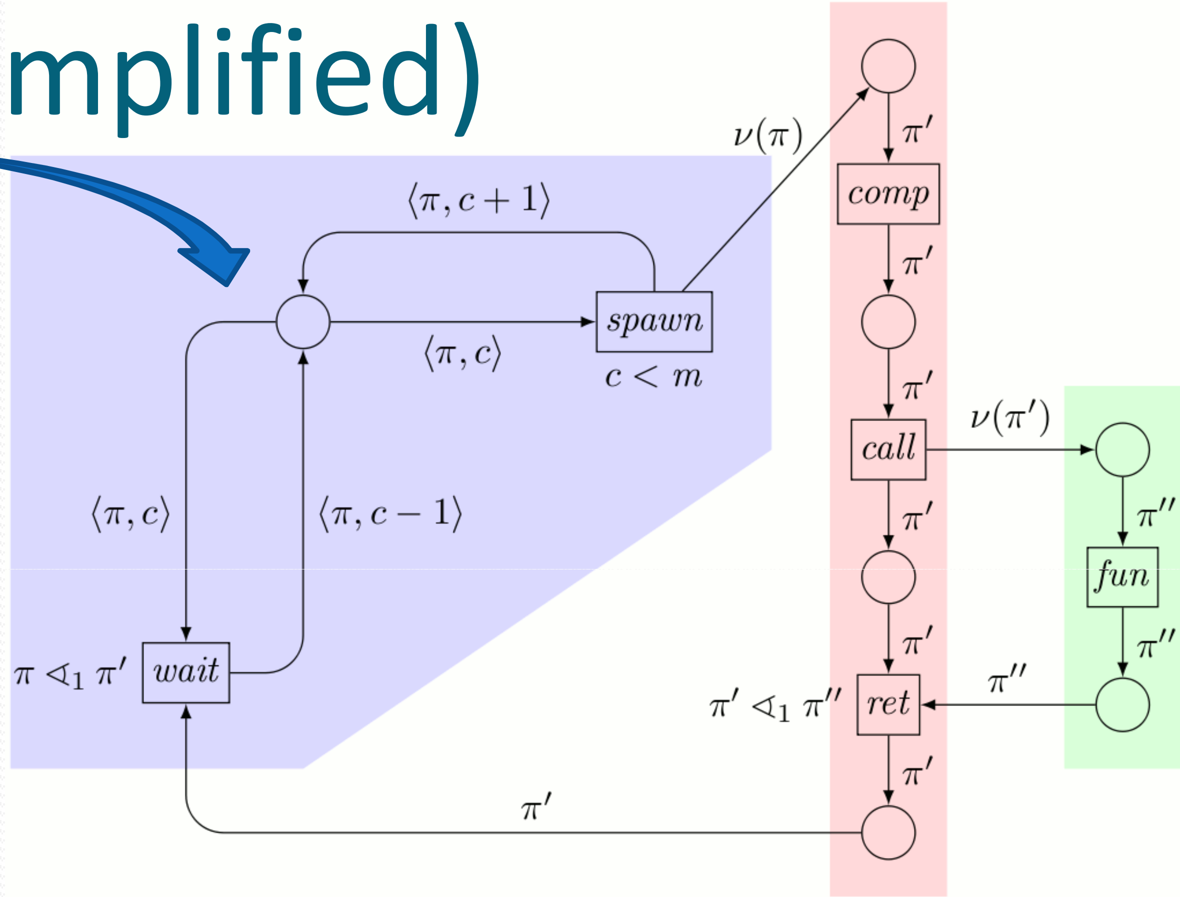
- Thread identifiers generation scheme
- Petri nets and colored markings
- Graph representation of markings
- Marking equivalence
- Example

Example (simplified)

k – initial threads

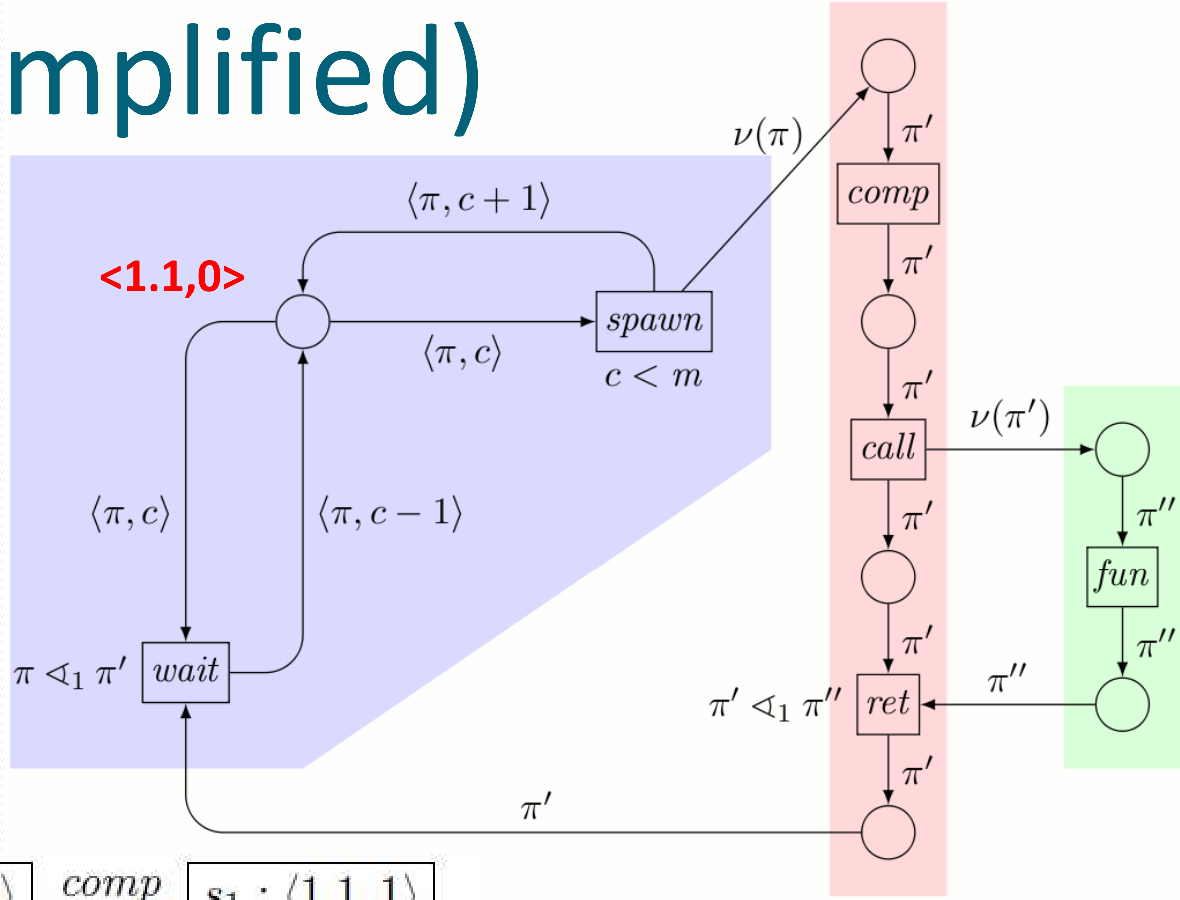
m – maximum number of active children

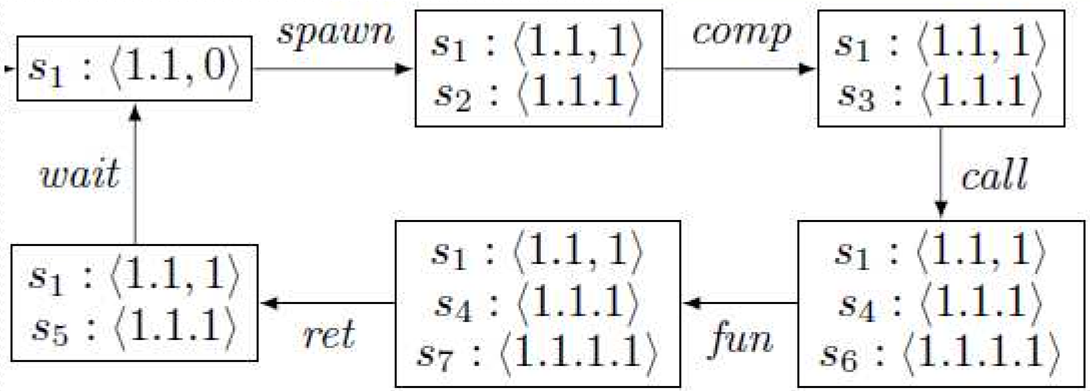
$N_{k,m}$



Example (simplified)

k – initial threads **= 1**
m – maximum number
of active children **= 1**
$N_{k,m}$ **=** $\mathbf{N_{1,1}}$

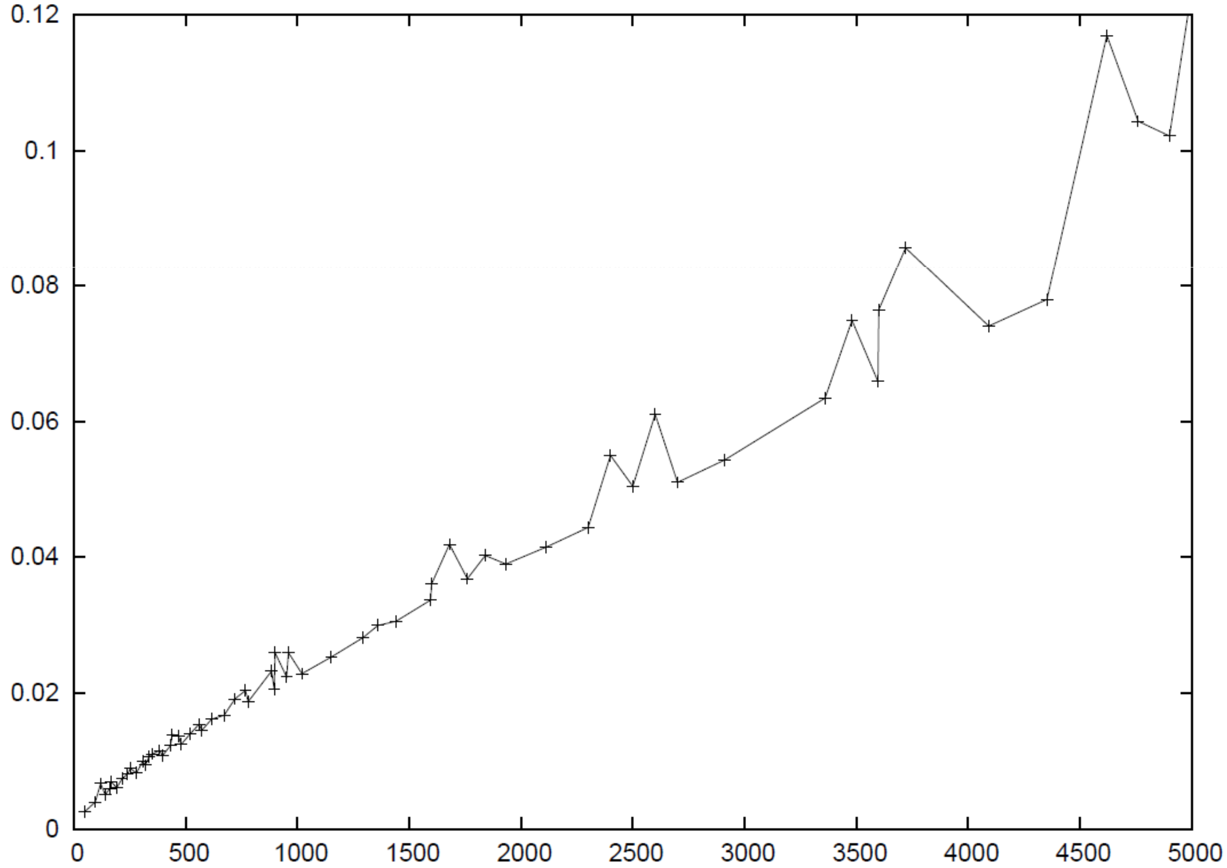




The state graph of $N_{1,1}$ - only 6 states, while infinite behavior if classical PN transition rule.

Without marking equivalence, no cycle in the net behavior.

Experimental results



Experiments using
- SNAKES (Petri nets)
- NetworkX (graph iso)

Global time irrelevant

Time spent on computing graph iso with respect to the size of the graphs (vertices*arcs)

Conclusion

- Introduced a method for modeling and analysis of multi-threaded state systems
  - It uses colored and composable Petri nets, allowing to capture systems with dynamic (and concurrent) process creation and manipulating data
  - It defines and computes a specialized marking equivalence allowing to aggregate markings in the reachability graph
  - In some situations, this aggregation may produce a finite representation of an infinite state system

Thank you !