

**Specification and Validation of a  
Concurrent System: An Educational Project**

Gérard BERTHELOT

CEDRIC-IIE

Institut d'Informatique d'Entreprise

France

Laure PETRUCCI

LSV, CNRS UMR 8643

ENS de Cachan

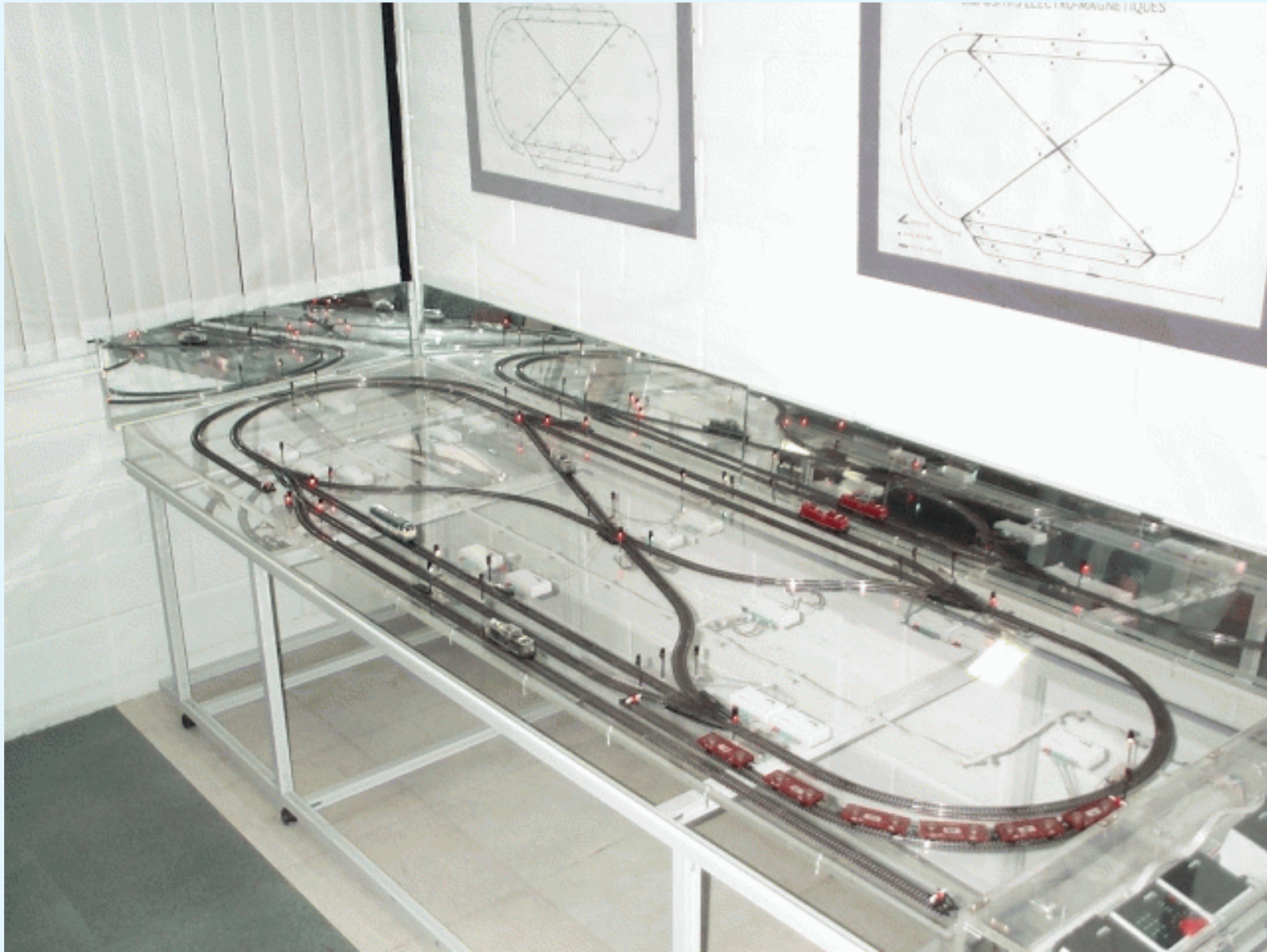
France

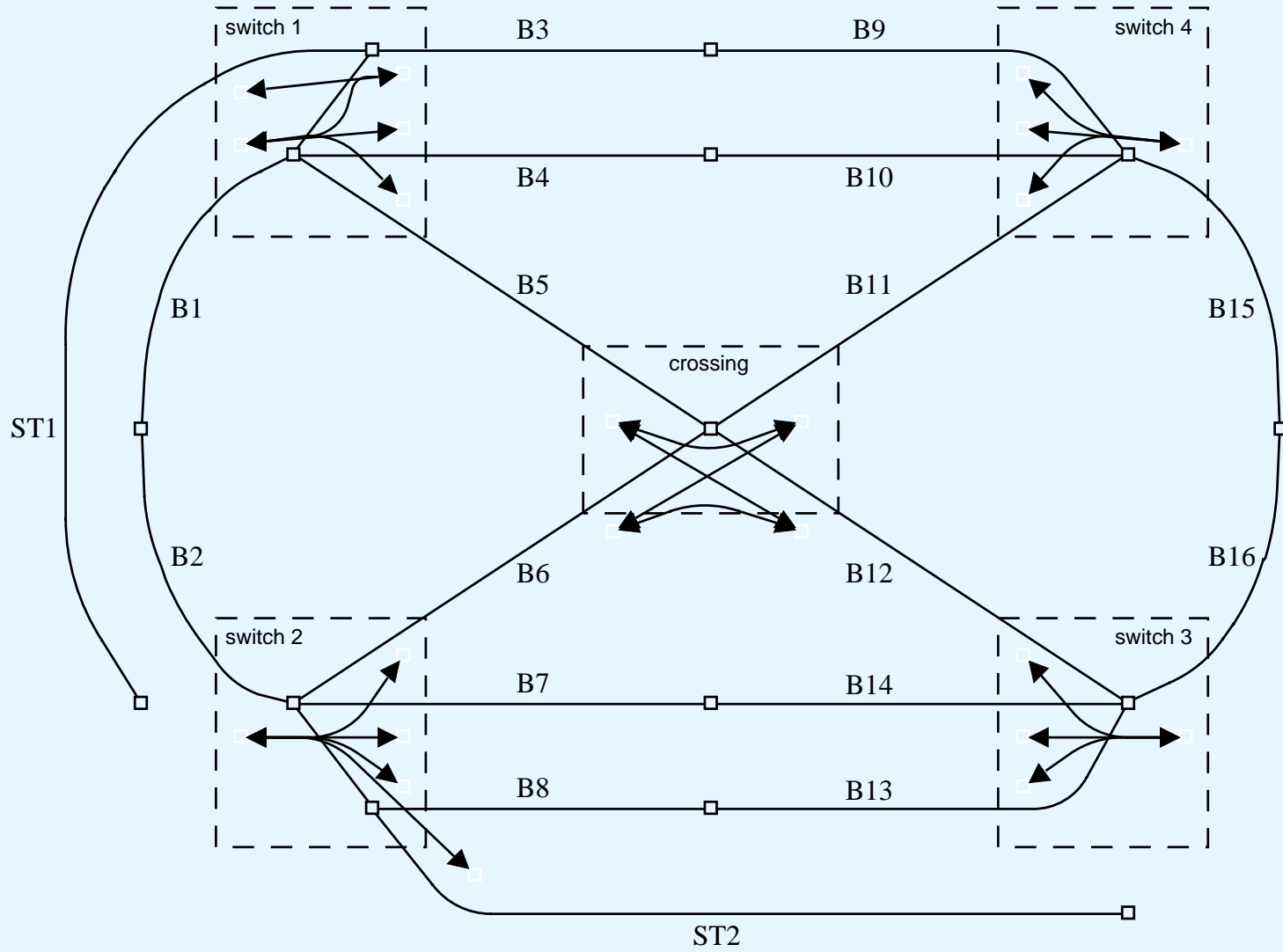
## Goals of the project

- master a project **from** the specification **to** the implementation **via** a validation phase.
- tackle the problems inherent to the specification of a concurrent program.
- identify and formalize the important properties expected from the system.
- use the model for programming.

## Outline

- ① Description of the hardware model railway
- ② Conception of a model
- ③ Properties proved
- ④ From the specification to the implementation
- ⑤ Conclusion





## Hardware and software monitoring

- PC running QNX operating system
- connection to the railway via serial port (RS232)
  - read information about trains passages from sensors
  - send orders to a specific train (speed and direction)
  - send orders to move switches and crossing
  - send orders to traffic lights

## Modeling of a real train system

### Operational characteristics:

- a train has a given travel plan,
- trains have different travel plans,
- a train cannot go backwards unless it is scheduled in its plan.

### Security requirements:

- at most one train on a section, switch or crossing,
- no deadlock.

## Difficulties of this approach

- complex modeling because:
  - a transition for each train for each move between sections,
  - transitions are not connected only to neighboring sections: need to book sections in advance,
  - difficult to use hierarchies.
- analysis hard to perform:
  - the number of reachable states depends a lot on the routes chosen,
  - the trains must be distinguished and named.

In the sequel, we focus on an adaptative point of view.

## Adaptative routing

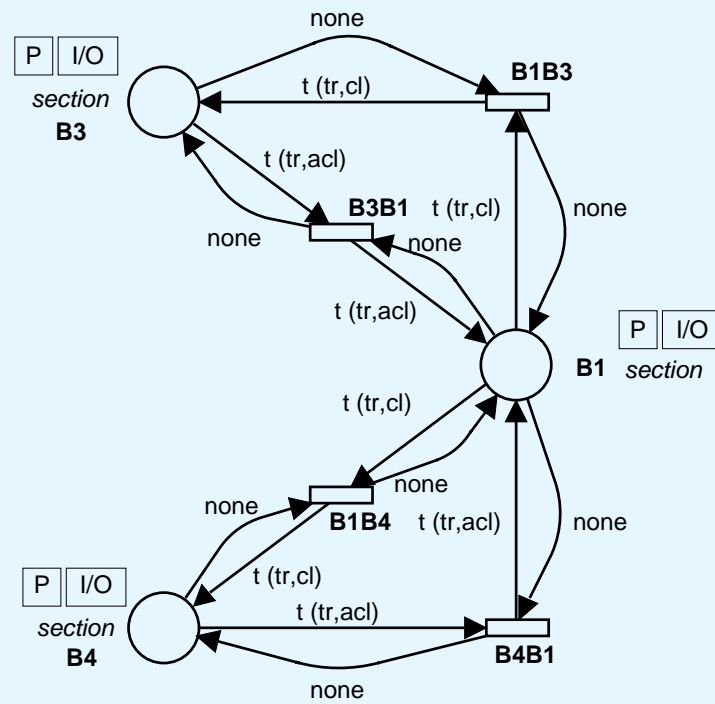
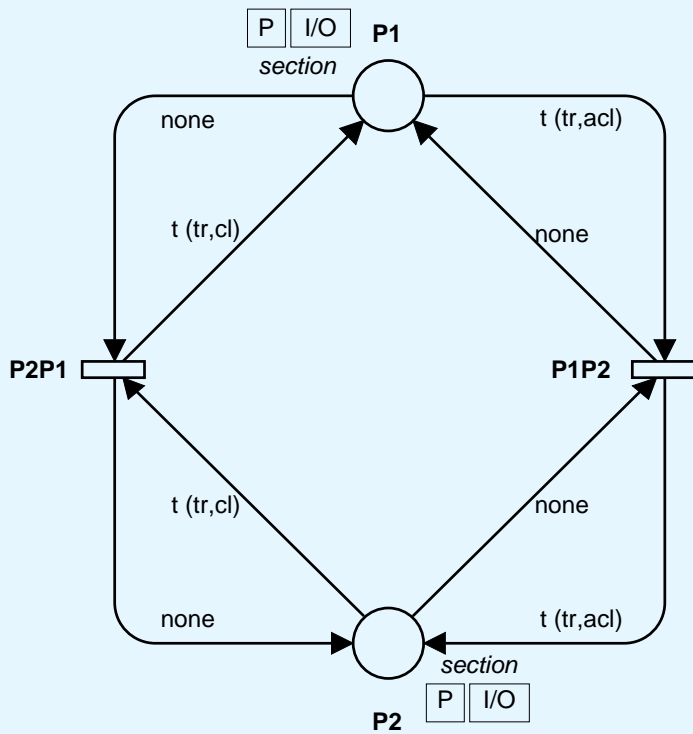
Behaviour of trains depends on local conditions.

A train can:

- choose among several tracks,
- go backwards when it is impossible to continue.

⇒ complex routing policies to be designed.

# Specification of basic moves between sections

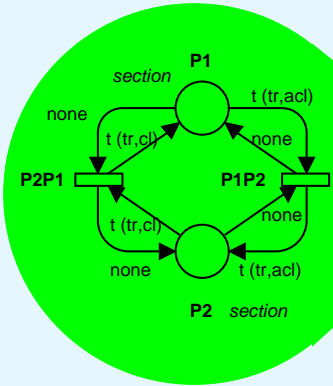


## Modeling the main loop

Two ways to obtain a global model:

- ① composition of basic components (**flat** model):  
large and intricate net.
- ② **hierarchical** model:
  - net representation close to the physical railway,
  - modification of a policy requires only to modify a single component.

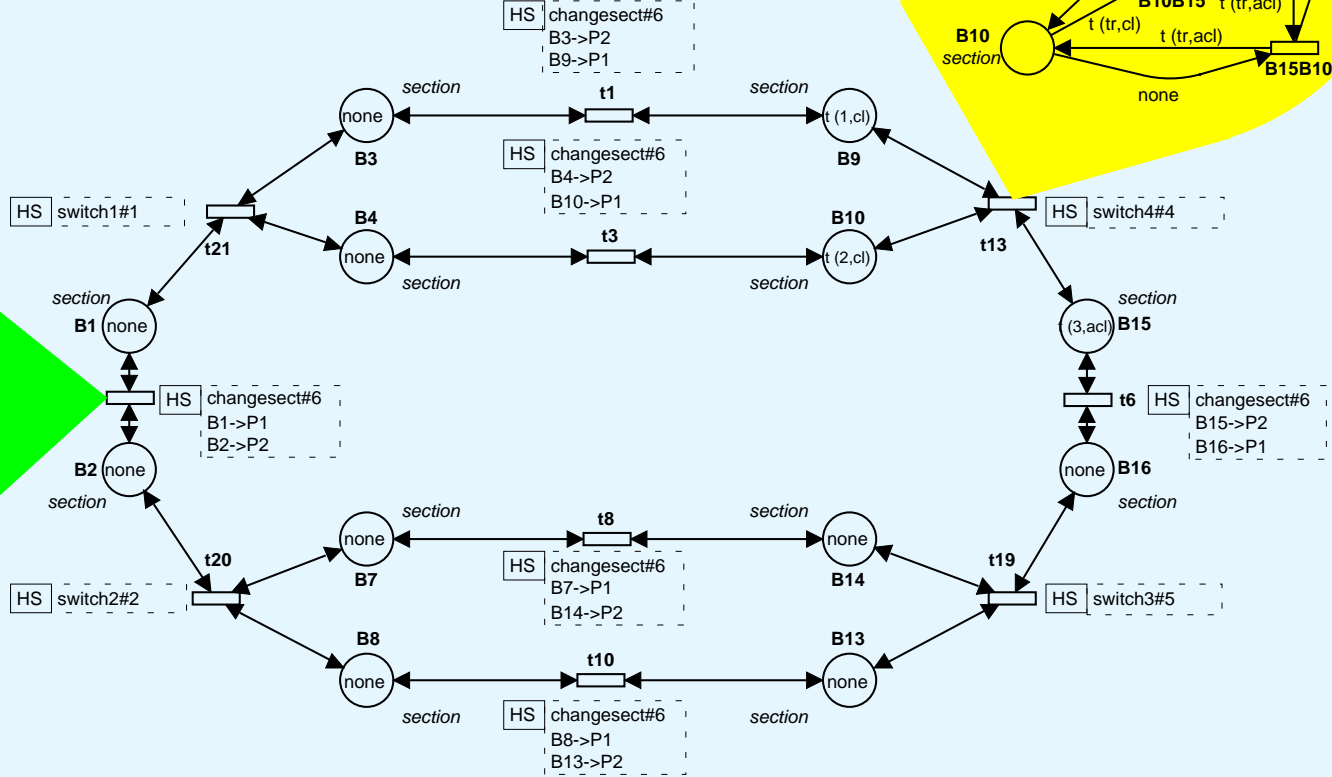
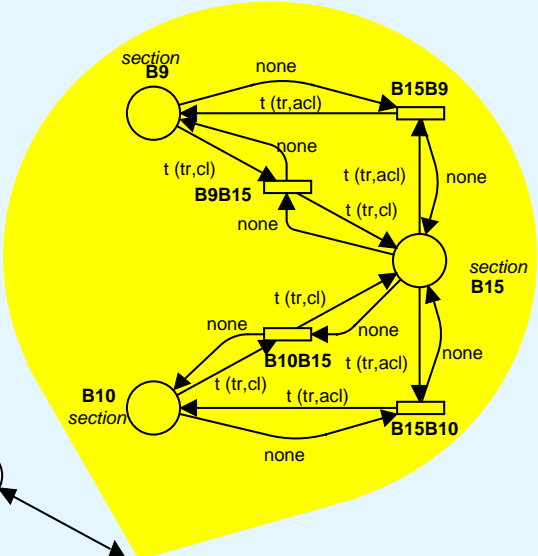
Students usually start as in ① but quickly switch to approach ②, having found out that it is easier to learn how to build hierarchies than to deal with a large net.



```

val n=3;
color direction = with cl | acl;
color name = int with 1..n;
color train = product name * direction;
color section = union t:train + none;
var tr:name;

```



## Properties to be proved

The students are asked to find out properties to be proved. They usually start with the following ones:

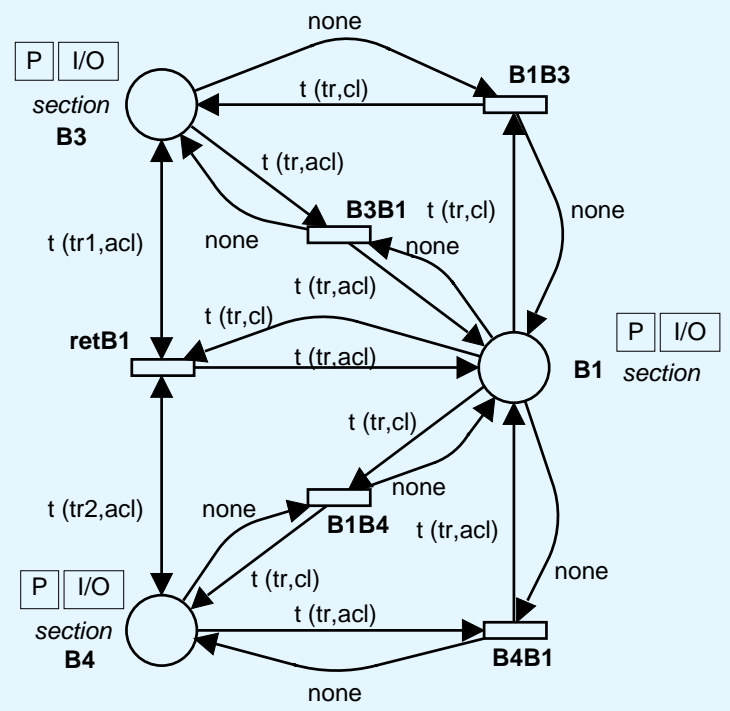
- **no collision**: an accident can never happen,
- **no global deadlock**: at least one train is running,
- **no local deadlock**: no train is stopped for ever.

➡ these properties can be checked using the **standard report** provided by DESIGN/CPN.

## First results

Before doing an exhaustive analysis, students perform a few simulations. Here, it ends immediately since the initial marking is a **deadlock**.

➡ New policy for switches:



## Further adjustments

Occurrence graph generation:

trains	nodes	arcs
3	1,321	3,781
4	21,574	72,026

⇒ certainly too large for 5 trains.

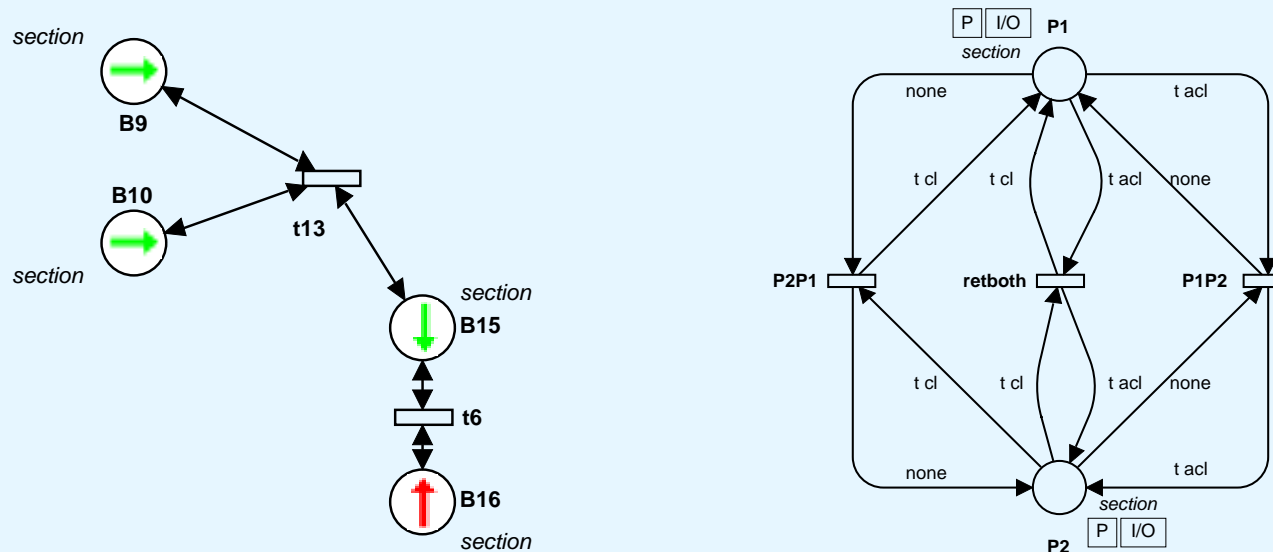
➔ need to remove trains names.

Occurrence graph for 4 anonymous trains: 2,166 nodes and 7,157 arcs.

## Analysis of properties

- All best upper and lower integer bounds = 1: ensures that collisions are impossible.
- 6 dead markings: surprising at first sight.


Visualization of a dead marking on the net  $\Rightarrow$  **New policy** to move between sections.



## Last analysis of the main loop

Occurrence graph for 5 trains: 24,556 nodes and 97,020 arcs.

Standard report:

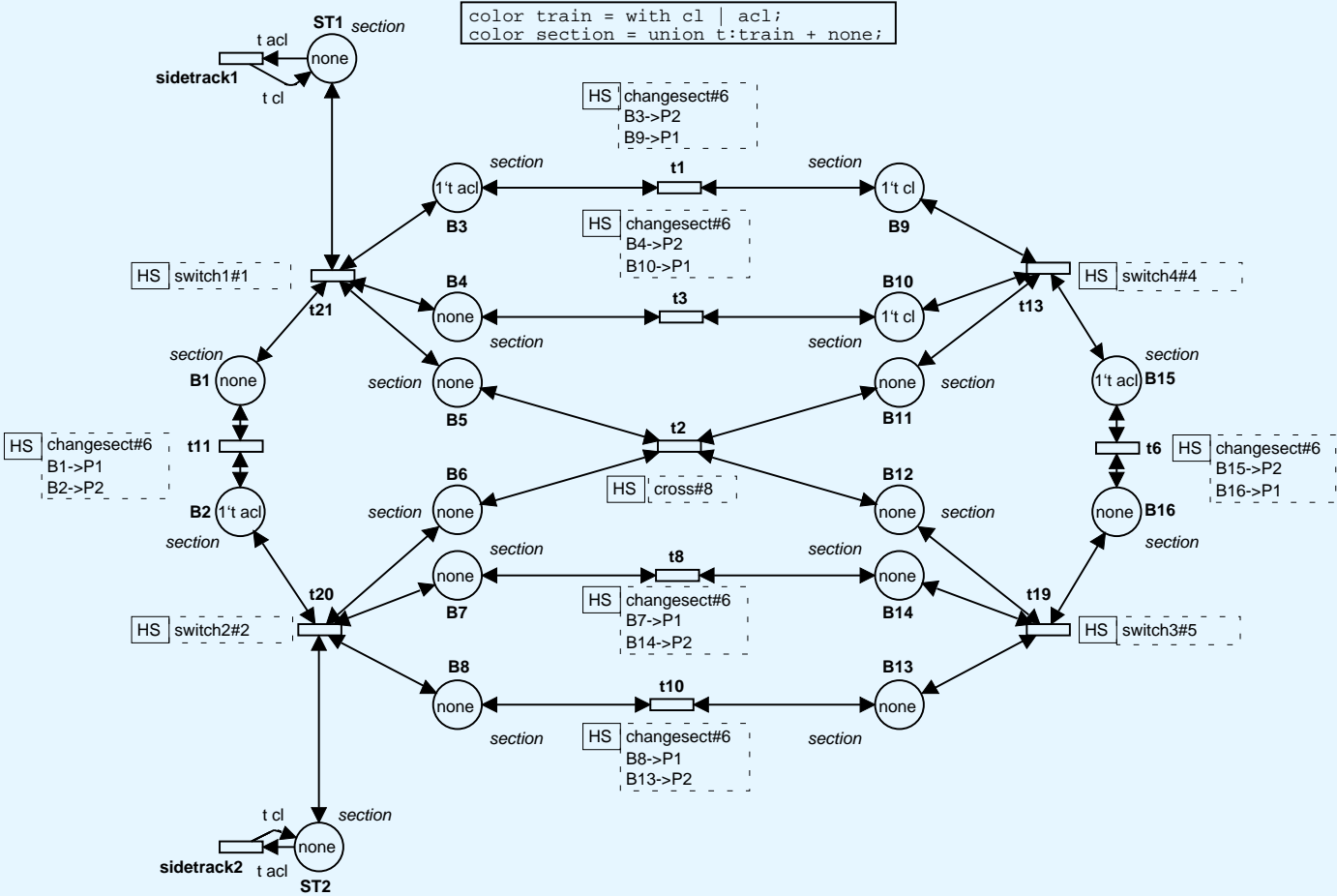
- same bounds,
- no dead marking,
- no home marking 

Further analysis: there are 2 terminal strongly connected components with 792 nodes each:

- ① all the trains go clockwise,
- ② all the trains go anti-clockwise.

# Construction of the full model

➔ Add the 2 sidetracks, the crossing and modify the switches accordingly.



## Analysis results

Occurrence graph generation:

trains	nodes	arcs
4	48,957	228,790
5	274,082	1,500,384

Properties (5 trains):

- same bounds,
- no deadlock,
- all reachable states are home markings,
- all the transitions are live  $\Rightarrow$  no train is forever stuck in a section.

## From the specification to the implementation

Mainly 2 possibilities:

- ① writing a simplified **Petri net simulator**:  
sticks to the net designed previously, in particular there is always one token per place  
→ no concurrency.
- ② splitting the net into a set of **synchronized processes**:  
one process per train, blocks, crossing and switches are shared resources;  
processes are synchronized using semaphores.

In all cases, students must formalize the transformation they make and show that the properties proved are preserved.

## Conclusion

### Benefits for the students:

- even in a limited period of time, students are able to handle the tools to design a system and prove non trivial properties,
- become aware of the interest of splitting the problem into communicating processes,
- experiences in mastering the difficulties of concurrent programming: synchronization, deadlocks, critical sections, ...

## Other aspects that could be studied

- All best upper multi-set bounds =  $1't(c1)++1't(ac1)++1'none$ : on any section it is possible to have a train going either one way or the other,
- check more complex properties using the occurrence graph inspection functions and the temporal logic library,
- manage time aspects,
- automatic code generation.