RATP INFRASTRUCTURES

© RATP / Bruno Marguerite

HLL Tutorial by RATP

Benjamin Blanc & Julien Ordioni — 4th of June 2019

About this tutorial

Agenda

- Duration: 1h50
- Short overview of what we'll talk about

RATP overview → Formal methods and RATP → Our tools and achievements → HLL, theoretical aspects → HLL, examples → HLL community

- Short presentation of each of us and your expectation about this tutorial
- RATP context
- HLL presentation, example, discussion and debate

Ground rules

- Participation, ask questions!
- Respect others, let them talk
- Agree to disagree

RATP

RATP, a national public service company

1949 1959 1976 1992

ÉPIC RATP (Paris)
Historic state-owned part

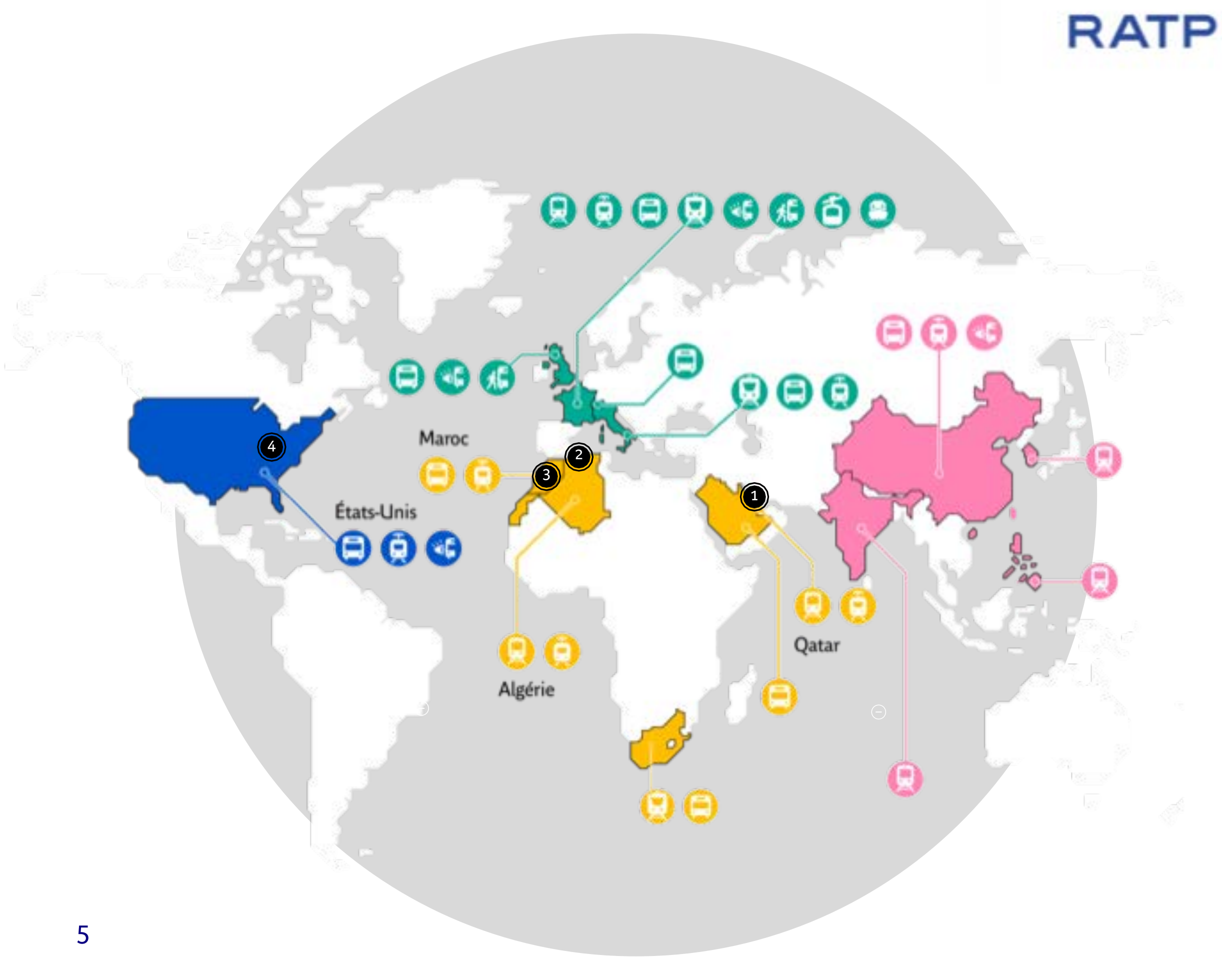
RATP Group
in 14 countries

16 million journeys
every day worldwide

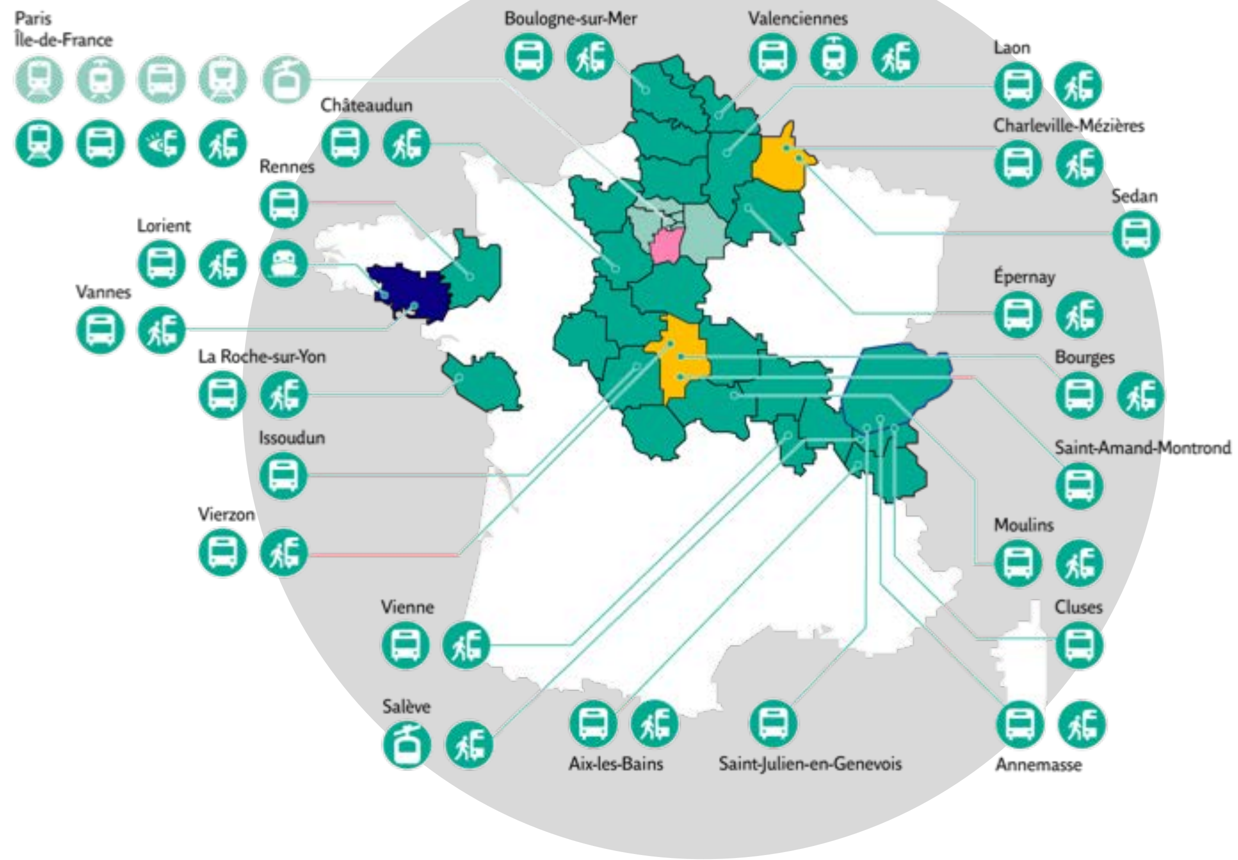
63.000 collaborators
(RATP Group)

One of the
5 leading players

RATP Group

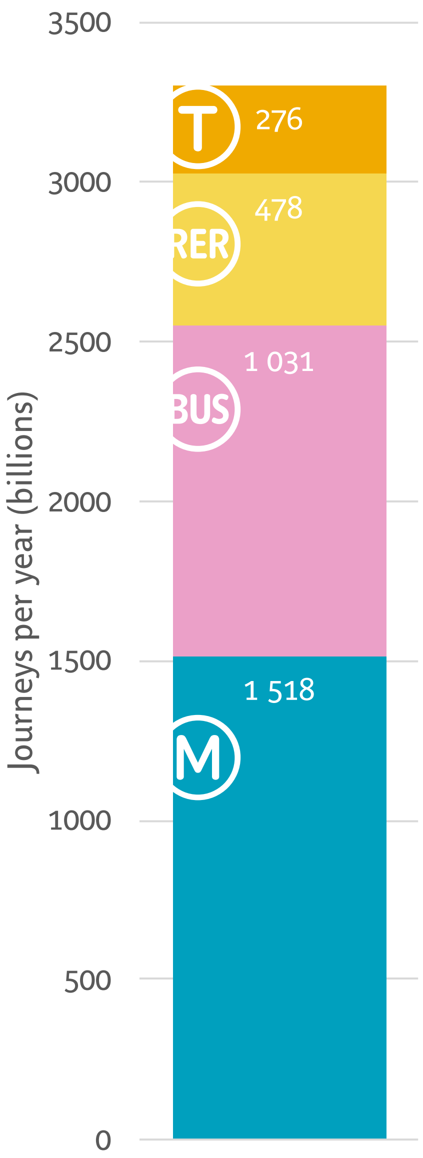


RATP GROUP
Maroc
États-Unis
Algérie
Qatar



Paris
Île-de-France
Boulogne-sur-Mer
Valenciennes
Laon
Châteaudun
Charleville-Mézières
Rennes
Sedan
Lorient
Vannes
Épernay
La Roche-sur-Yon
Bourges
Issoudun
Saint-Amand-Montrond
Vierzon
Moulins
Vienne
Cluses
Salève
Aix-les-Bains
Saint-Julien-en-Genevois
Annemasse

RATP in Paris



More than 3.3 billions journeys per year

- About 98 % of punctuality (metro)
- Up to 01:25 interval (L14)

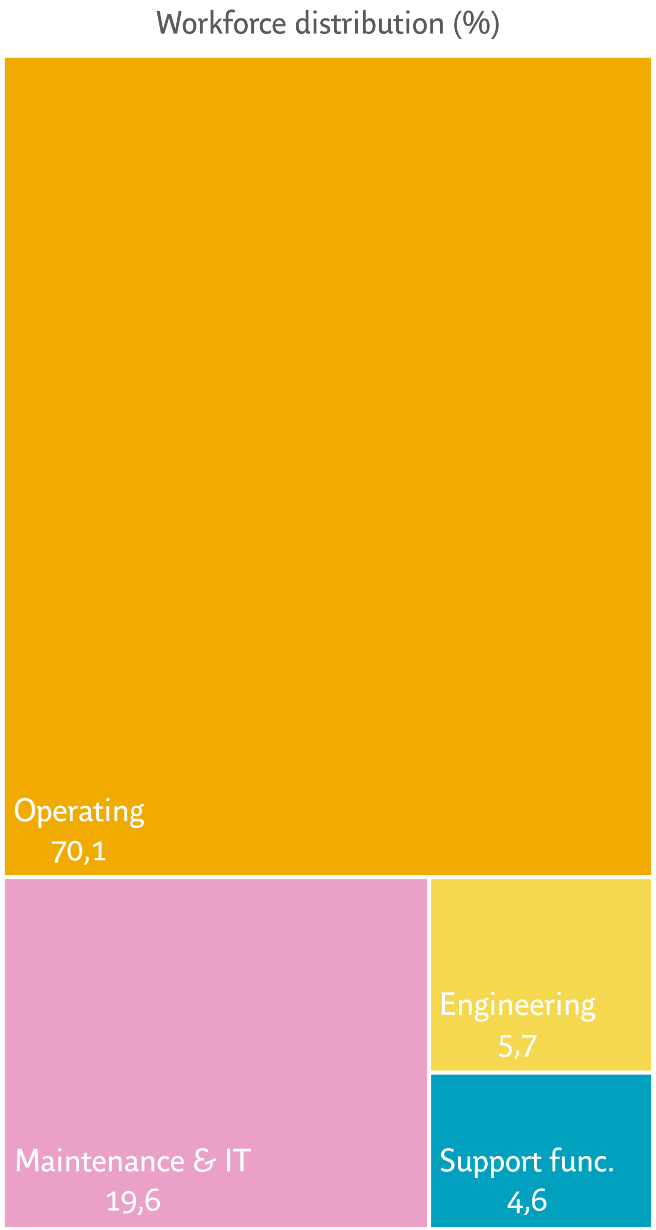
More than 1 000 trains

RER A: 40,000km per day

	Lines	km	Stations / Stops	Trains / buses
T	8	96	136	222
RER	2	115	67	357
BUS	338	3 727	7 302	4 532
M	16	206	203	694

Data: 2014

RATP and human resources



Divided into 21 departments

3.700 new collaborators in 2017

About 25.000 conductors (buses, tram, metro and RER)

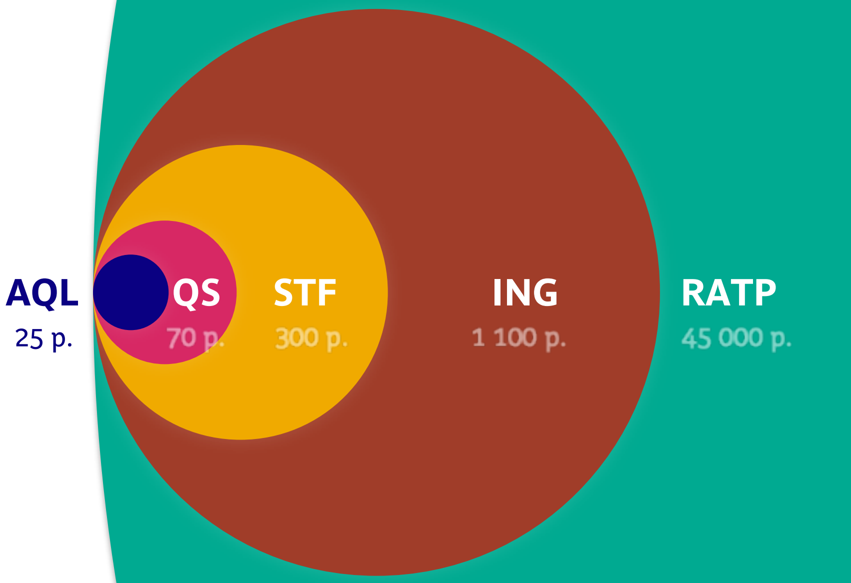
About 2.500 collaborators working on engineering

- Transportation systems
- Information systems
- Civil engineering
- Architects
- Etc.

Data:2017

OUR MISSIONS

Who are we?



Engineering Department

Dealing with **transportation systems**

Involving **passenger safety**

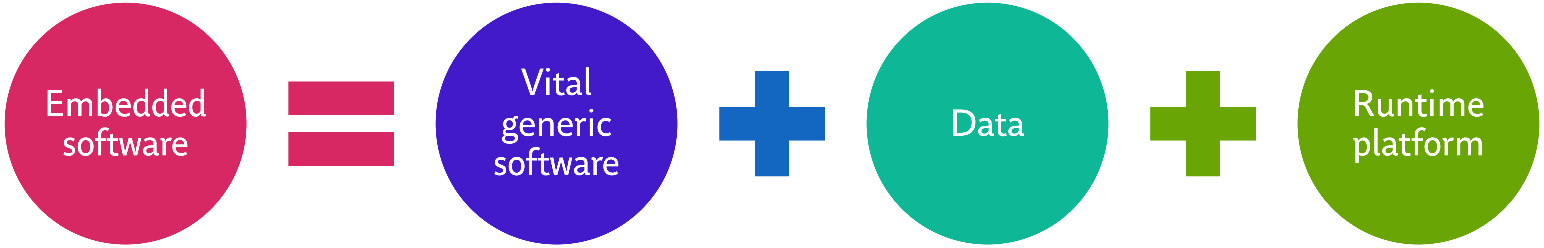
About **control/command software**

AQL, *Safety critical assessment software lab*, since end of 80s (SACEM, RER A)

AQL

Safety critical software assessment lab

Assess that the running embedded software, including data, has a safe behavior regarding the traveler safety



For:

- Train control/command systems(CBTC)
- Computerized-based or hybrid interlocking systems(PMI, PHPI)
- Other safety critical software(PSD, DIL, DOF)

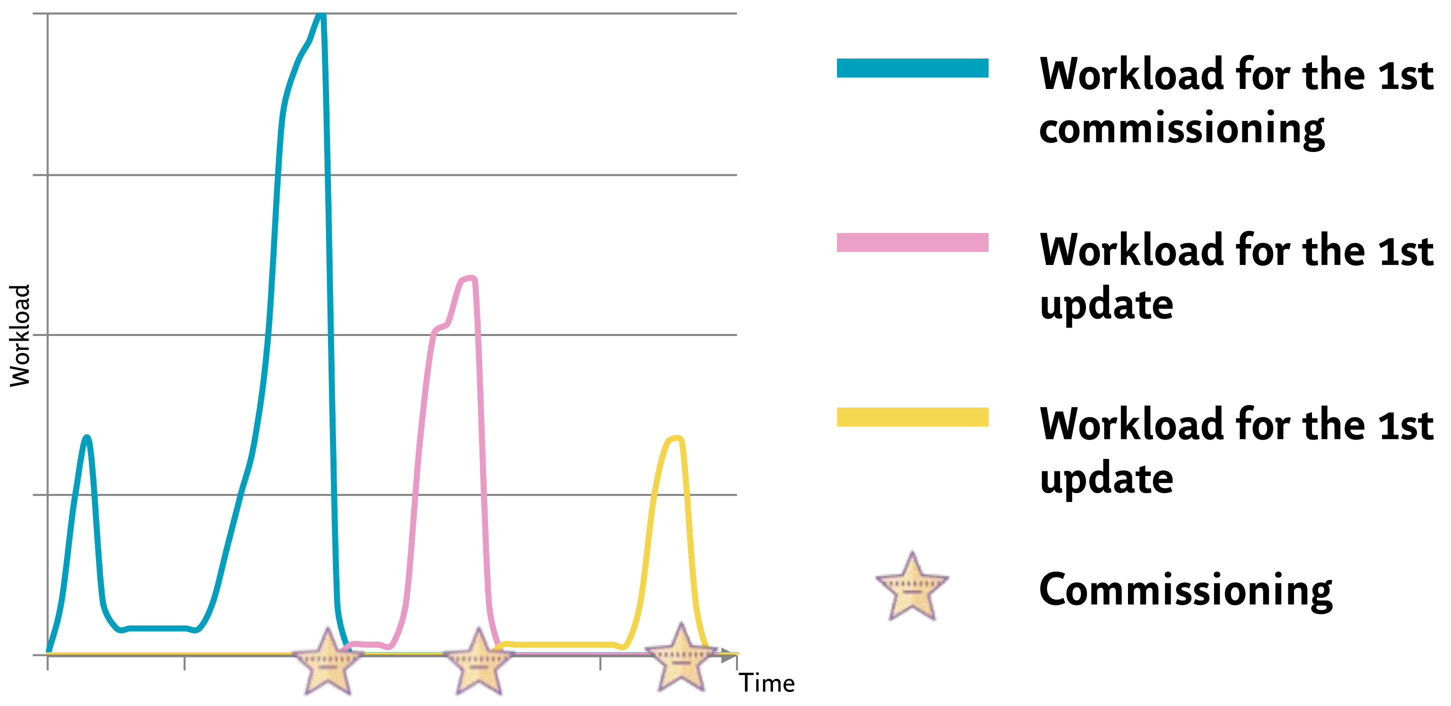
With:

- State of the art tools and methods
- Dedicated tools: HIL testing environment , proof servers (1 TB RAM)

Internal assessment

When?

- Work until the commissioning
- Work late (after supplier validation)
- Quick update assessments thanks to our automatic methods
- Maximum use of certificates

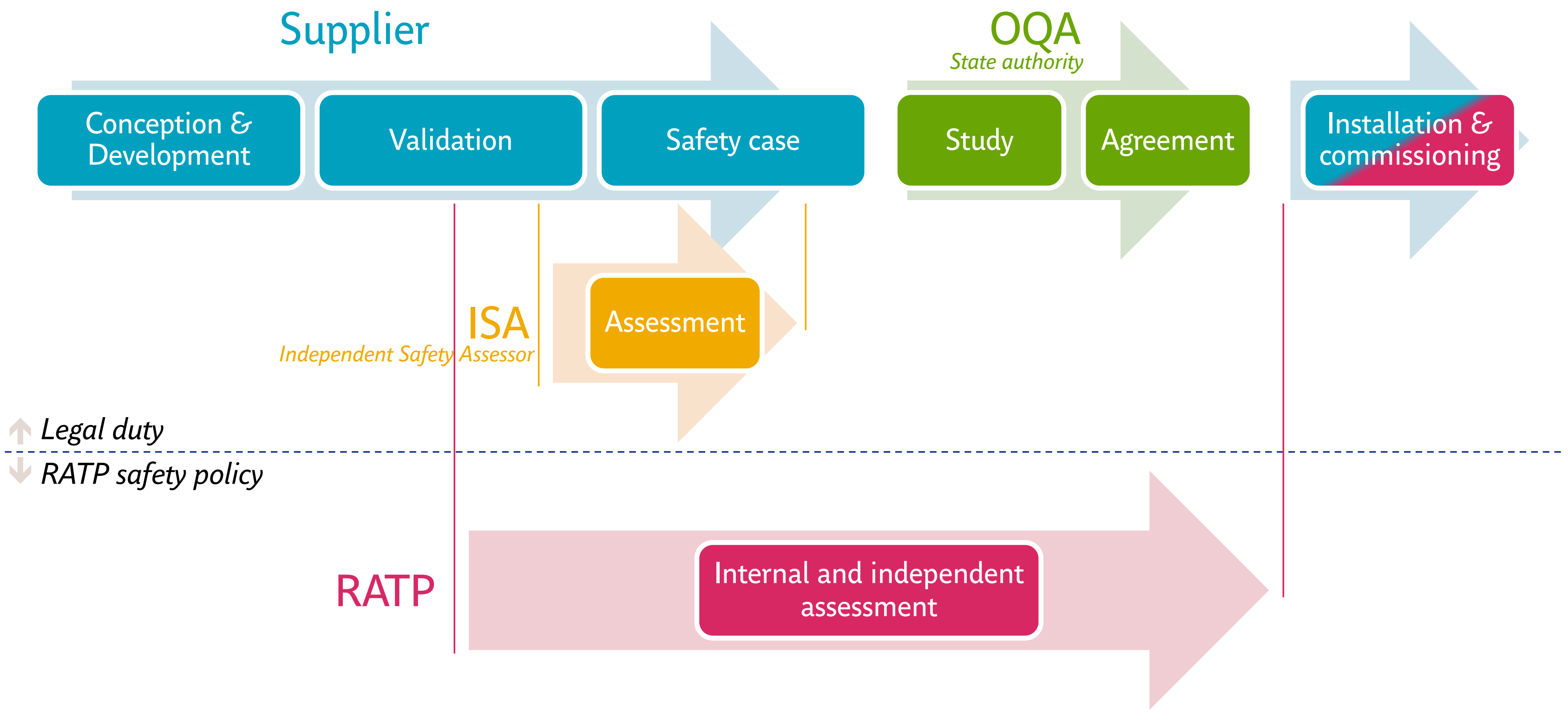


Internal assessment

Why?

▸ RATP safety policy

Provide an internal and independent assessment of safety critical systems before commissioning



IN FORMAL METHODS WE TRUST

To share basics…

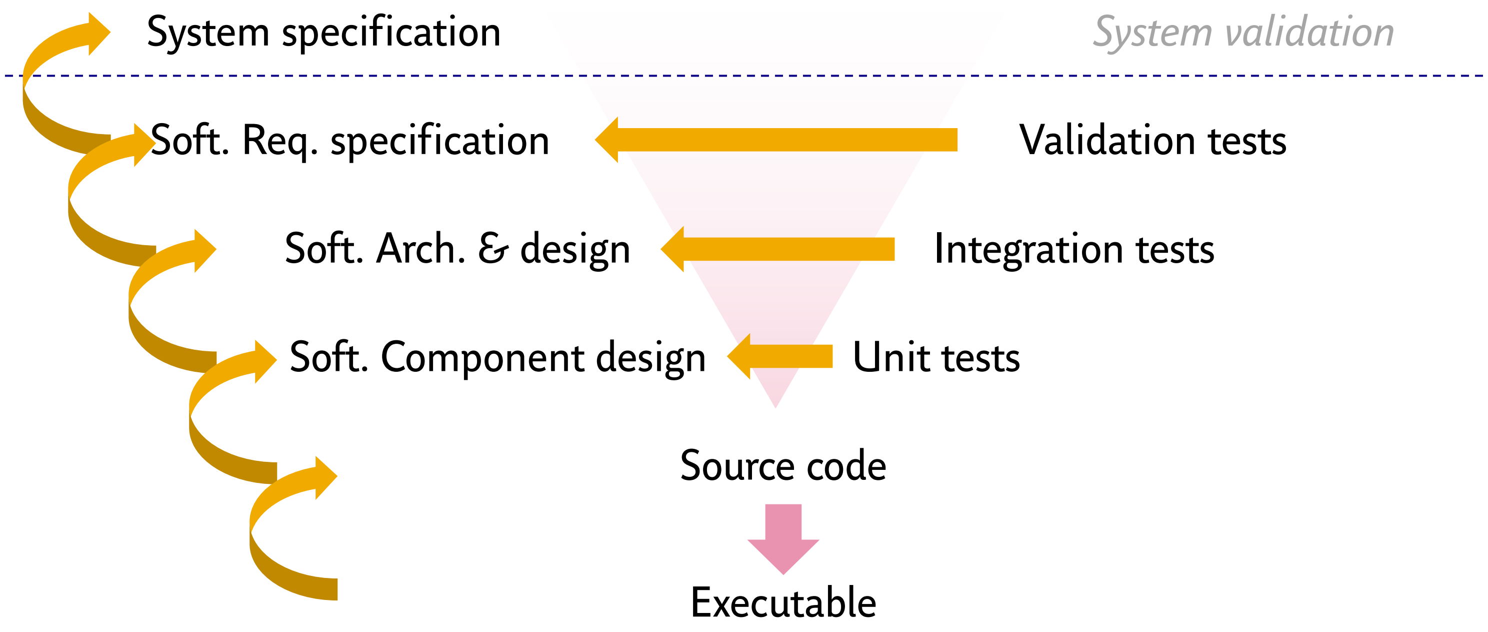
"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

Edsger W. Dijkstra

The Humble Programmer, 1972

Typical assessment activities

- Validation of each steps of the refinements
  - System ⇨ software specification
  - Software specification ⇨ source code
  - Code source ⇨ executable
- Manual code review
- Tests validation and test coverage validation



Our assessment activities

- Classical method

Validation of each V-cycle step

  - A lot of manual activities
  - Relative efficiency
  - … but available in all cases

- With **formal methods**

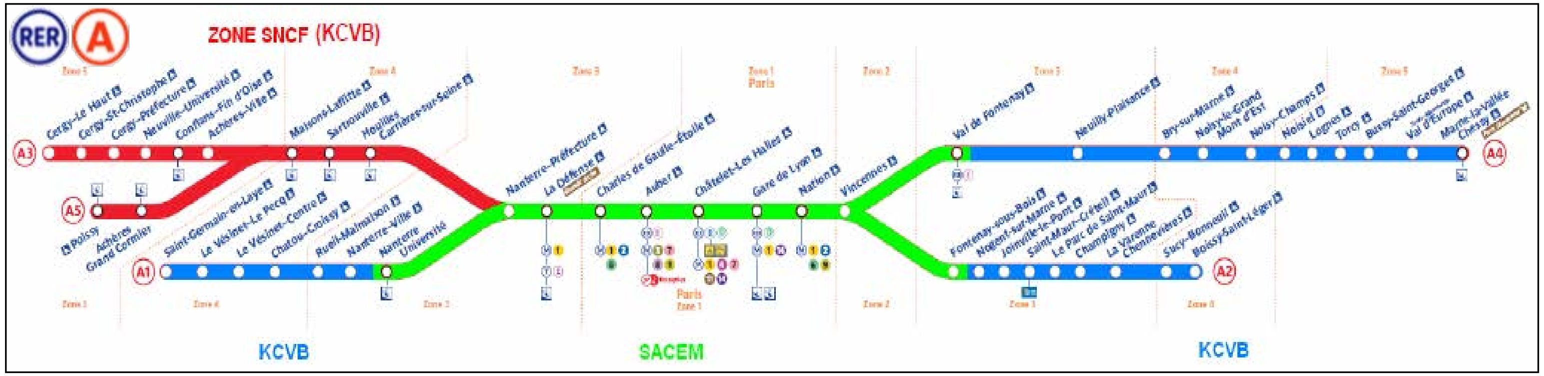
Exhaustive, accurate and non-ambiguous

  - 100 % sure to discover problems with formal methods *(feedback)*
  - Requires specific tools
  - May be long, complex, indeterminate…
  - Requires sharp proof engineering skills
  - Efficient system update management

SPOILER ALERT

Retro-modelling?

1989, SACEM, the first computerized ATP system



- Started in 1977, the development experienced new methods for safety related to computer-based application using:
  - Rigorous development model, coded mono-processor, application software written in MODULA-2 (about 60 000 lines of code)
- Concern for safety of the ATP software
- Decision taken for "**retro-modelling**" the application code using the "Z notation" (*Hoare logic*) with **Jean-Raymond Abrial** and **Stéphane Natkin**

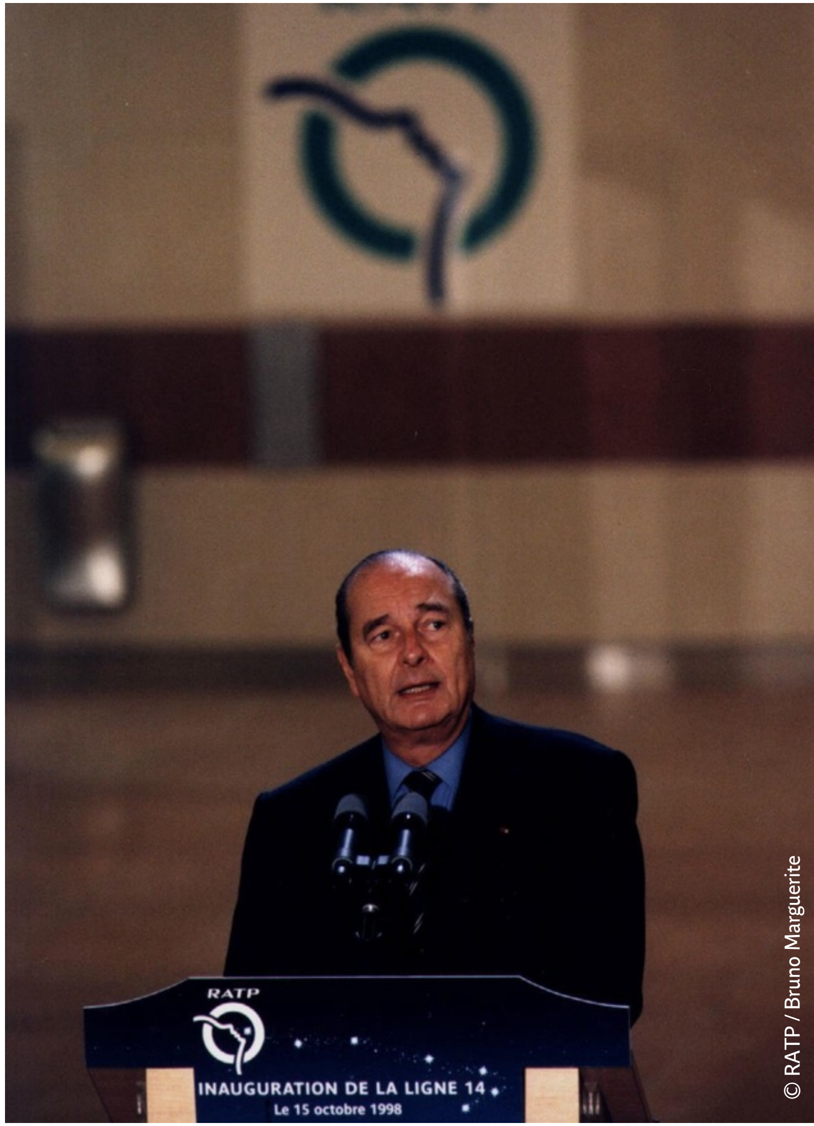
More than 10 unsafe scenarios discovered and corrected before revenue service in 1989

Consequences

- ***B-Book*** by Jean-Raymond Abrial
- Industrialization of the **Atelier B** together with INRETS, SNCF, GEC-Alsthom & Steria (now Clearsy)

1998 introducing the first computerized ATP/ATO

- 100% vital software build using B
- Paving the way for modern CBTC systems



© RATP / Bruno Marguerite

From B to retro-modelling

After METEOR L14

- Only 2 suppliers were using B method
- European regulation required competition balance for public procurement
  - This clause used in tender documents had to be removed: *"… **the proof for obtention of the adequate safety level shall be brought either using B method**, either another method should it present an equivalent proving capacity (to be demonstrated by the tenderer)…"*

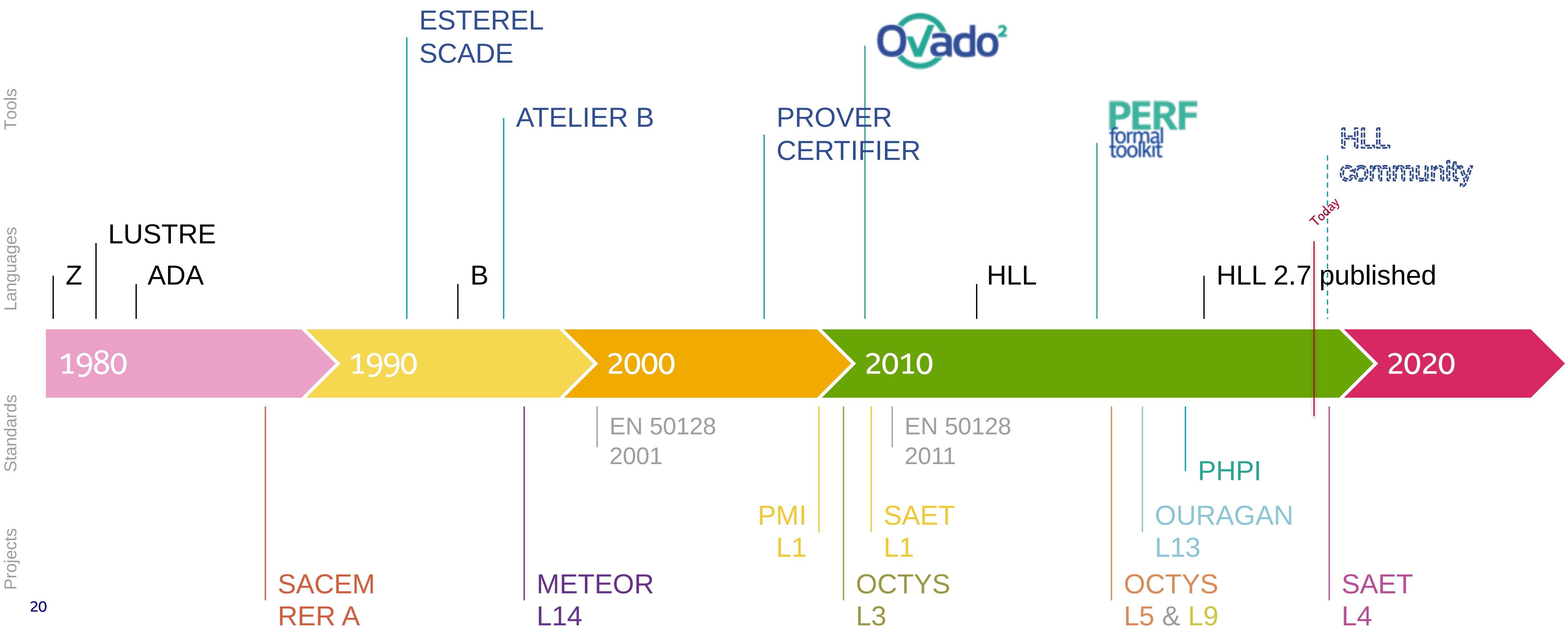
RATP still convinced in using formal methods

- We specified a formal proof tool-chain called "*Prover Certifier*" to **perform formal proof over a software developed** with a semi-formal approach and **without supplier software modification**

PROVER®

- Provided to Ansaldo (CBTC) & Thales (CBI)
- 2010, (re)birth of the retro-modelling approach with PERF method

RATP & formal methods



In formal methods we trust

Also for our usage

2008-2013: First formal verification on CBI (Petri nets + SAT solver)

- Birth of the Proof toolkit (*Prover Certifier*)
- Birth of HLL
- Simple structure & boolean equations

Since 2010, application to CBTC safety properties

- Birth of PERF Method and PERF formal toolkit
- Based on HLL & SAT solver
- Different translators

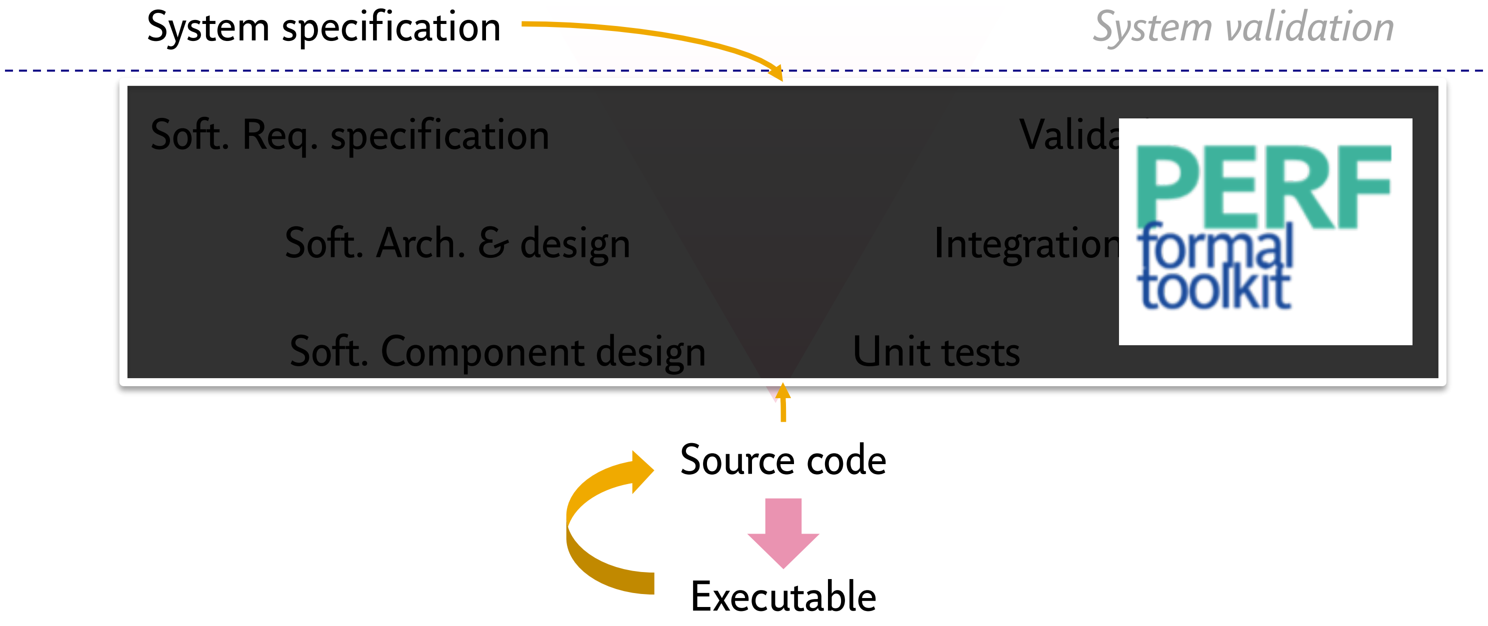
PERF formal toolkit

Retro-modelling

Because it's efficient

PERF Method

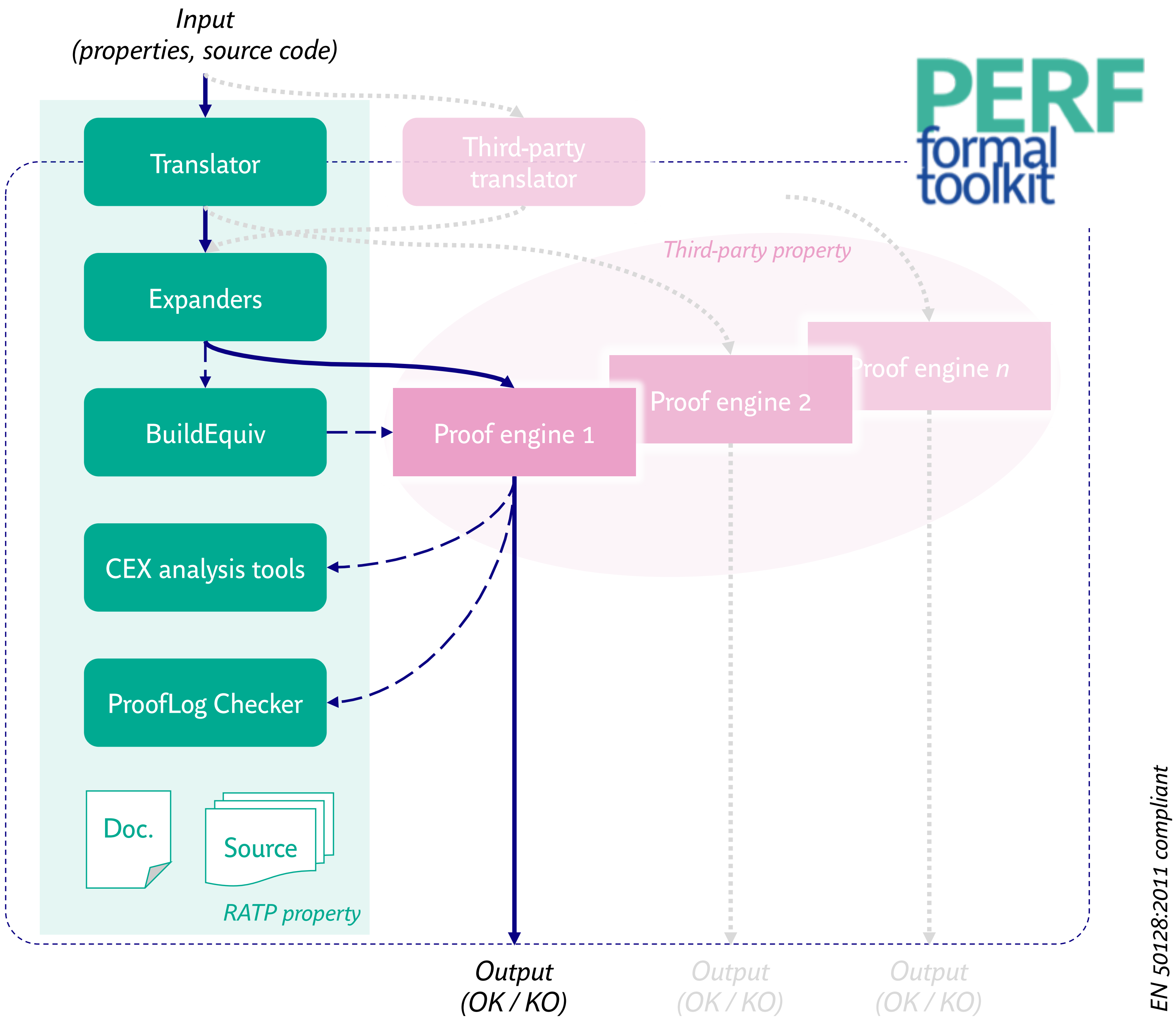
- *Proof Executed over a Retro-engineered Formal model*
- A RATP *PERF formal tookit* combined with third-party SAT proof engine
- Suitable for different projects and suppliers
- Independent of the software development cycle
- HLL Property level (component, soft. or system level) depends on the needs



Performed on projects: 1 3 4 5 6 8 9 11 12 13

PERF

PERF formal tookit overview



PERF main results in the RATP context

2014
- **PERF Toolkit first results**
- **Application to CBTC**
- Software-level safety properties

2015
- **PERF on Manual ADA (Main Line)**
- Intermediate-level safety properties

2016
- **System-Level safety properties**
- **Tracking CBTC function**
- Beginning of CIFRE PhD: from B Model to HLL (and PERF toolkit)

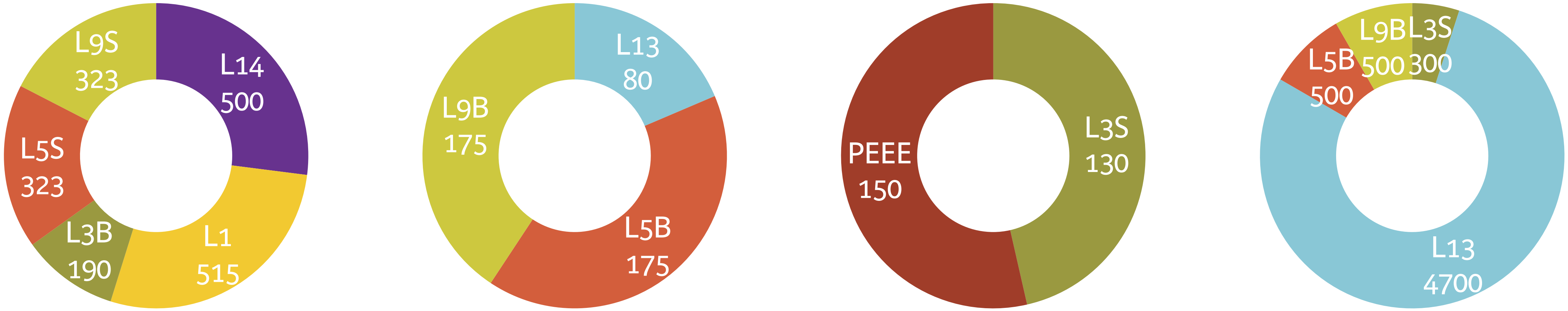
2017
- **B2HLL: first theoretical results**
- CBTC proof replays

2018
- **PHPI first revenue in service** (Functionnal HLL model)

2019
- **RBS2HLL: safety-based HLL model**
- GRAAL approach still in progress…

Facts and figures about PERF

B (k loc) | **Manual C (k sloc)** | **Manual ADA (k loc)** | **SCADE (*.scade)**



RATP Safety assessment using **PERF Method** vs "classic" and manual analysis

B | Manual C | Manual ADA | SCADE

HLL translators

B2HLL

- CIFRE PhD still in progress
- Soon: industrialization
- Details on TASE 2019, Guilin China

SCADE 5 & 6

- SCADE 5 translator used for OCTYS
- SCADE 6 translator used for OURAGAN
- ANSYS is now included in the HLL brainstorming

C & ADA

- Customized for generated code
- C or ADA subset
- Not easy to use on manual C & ADA code

The proof engineering

Modelling…

- Software
  - Automatic translators but a minimal understanding of the generated HLL is needed for CEX analysis or grey box modelling
  - Ensure that translator application conditions are granted
- Properties
  - Be careful of implicit hypothesis
  - Define the appropriate level of property modelling
  - And contain consequences on other activities
  - Not too fast (*else true* issue)!

Application to projects

- Scalability of methods and tools
- Team training and globalization
- Balance between costs, confidence and efficiency

To be continued…

Proof engines

- SMT or symbolic solvers to complete SAT solvers weaknesses
- *MooN* certified proof

Translators

- Develop new translators through the community
- Industrialize PhD work with B2HLL translator
- Improve the RATP RBS2HLL translator

B2HLL
RBS2HLL

HLL Community

INTERESTED?

- Build a (legal & technic) frame around HLL with interested designers, users, academics
- Publish sHLL specification?

HLL, THE CORE OF PERF

HLL, (not) a modelling language

HLL is the pivotal language of the PERF methodology

Programs under proof are not directly developped in HLL: Translators allow to import Scade, C, Ada designs

HLL is a target language for these designs and a high level input language of model-checking tools

These tools are intended to satisfy safety properties on the designs, according to possible environment constraints

These tools rely on the synchronous observer approach

SAT/SMT Based Model-Checking

Given a symbolic representation of a system: (*In*, *S*, *Init*, *X*)

A property *P:*

- Safety: something bad never happens
- Liveness: something good eventually happens

Does the property hold for all computations of the system?

Induction scheme is correct for safety properties:

1. Initiation: All initial states satisfy P

   $Init(S) \Rightarrow P(S)$

2. Consecution: All successors of valid P-states are valid P-states

   $P(S) \wedge X(In,S,S') \Rightarrow P(S')$

A saturated counter

```
Constants:
int N := 10;

Declarations:
int unsigned 4 cpt;

Definitions:
I(cpt) := 0;   // init
X(cpt) := if cpt = N then 0 else cpt + 1; // transition

Proof Obligations:
(0 <= cpt) & (cpt <= N); // saturation
(cpt + 2) % (N+1) = X(X(cpt)); // circular behaviour
```

A saturated counter

The property evolves synchronously with the design it observes, it must be true for all cycle

The proof scheme will then follow the two steps:

1. $I(0 \leq cpt)$? Yes because $I(cpt) = 0$
2. $0 \leq cpt \wedge X(cpt) \Rightarrow X(0 \leq cpt)$?

   $0 \leq cpt \wedge \text{if } cpt = N \text{ then } 0 \text{ else } cpt + 1 \Rightarrow 0 \leq \text{if } cpt = N \text{ then } 0 \text{ else } cpt + 1$

   1. $(0 \leq N) \Rightarrow 0 \leq 0$
   2. $(0 \leq cpt) \Rightarrow 0 \leq cpt + 1$

In a nutshell

Data flow: a variable represents an infinite stream of data

Synchronous: all flows have the same length

Cyclic: time is abstracted as a unique discrete global clock (unmentioned)

Declarative: The focus is on the input/output relationship rather than on control structure

This is the language familly of Scade, Simulink, LabView, etc.

Datatypes

Atomic: boolean and integer (potentially bounded and signed)

```
int[0,15] y;
int unsigned 4 z;
```

Enumerated set of identifiers

```
enum {red, green, blue}  color;
```

Hierarchy of finite sets

```
sort monostable, bistable < relays;
sort NS1-4.0.4, NS1-12.0.8 < monostable;
```

Structures and tuples :

```
struct {abs: int, ord:int}         point;
```

Arrays with a statically defined set of sizes

Combinatorial functions

Dataflow Equations

Two possible formulations:

- `v := e, f;`
- `I(v) := e;`
  `X(v) := f;`

e	e_0	e_1	e_2	e_3	e_4	...
f	f_0	f_1	f_2	f_3	f_4	...
v	e_0	f_0	f_1	f_2	f_3	...

Cyclic references must be broken by a latch:

- `X(e)`
- `Pre(e)`
- `Pre(e, i)`

e	e_0	e_1	e_2	e_3	e_4	...
X(e)	e_1	e_2	e_3	e_4	e_5	...
Pre(e)	nil	e_0	e_1	e_2	e_3	...
i	i_0	i_1	i_2	i_3	i_4	...
Pre(e, i)	i_0	e_0	e_1	e_2	e_3	...

Equivalent formulation:

- `v := pre(f,e);`

Data flow operators

Pointwise application of usual operators:

e	e_0	e_1	e_2	e_3	e_4	...
f	f_0	f_1	f_2	f_3	f_4	...
e+f	e_0+f_0	e_1+f_1	e_2+f_2	e_3+f_3	e_4+f_4	...

Logical operators are lazy

Arithmetics is bounded and exact: memories and inputs must be statically bounded

Unintialized flows produce a *nil* value that must not appear in observable flows (proofs, outputs, constraints)

Those checks are performed by model-checking tools (sanity checks)

Arrays

Arrays of static size:

```
odd[i] := if i = 0 then FALSE elif i = 1 then TRUE else odd[i-2]
```

Arrays can have a memory definition:

```
SW[i] := false, if i = 0 then a else SW[i-1];
```

Arrays are accessed lazily on their definitions

Out of bounds access is considered an error

Functions

Functions are combinatory combinations of its potentially infinite inputs

```
int Fibonacci(int);
Fibonacci(i):= if i <= 2 then 1 else Fibonacci(i-1) else
Fibonacci(i-2);
```

But functions can not refer to memories values of its inputs:

```
bool bad_rising_edge(bool);
bad_rising_edge(x) := false, x & ~X(x);
bad_rising_edge_2(x) := x & ~x; // equivalent
```

Functions are limited to a single output value

Input types must be scalar

Quantifiers

Quantification over finite sets allows compact definitions:

```
// Does array A of 10 integers  contains an even value
SOME i:[0,9] (A[i] % 2 = 0);

// Does all even indices of A contains even values
ALL i :[0,9] ( i % 2 + 0 -> A[i] % 2 = 0);
```

Also has some numerical extensions:

- SUM
- PROD
- $max
- $min

HLL in practice

The purpose of an HLL file may be twofold:

1. Formalizing knowledge
2. Solving requests using a SAT based tool (as of today)

The process is as follows :

1. Translation to a lower level language (LLL) where everything is bit blasted
2. Solving sanity checks (partial definitions, array indices, arithmetics overflow)
3. Launching tool:
   1. properties are proved
   2. properties are falsifiable: analyzing counter-example
   3. properties are indeterminate: analyzing step counter-example and adding lemmas

HLL CASE STUDY

Train Mapping: Overview

Can we (RATP) manage to prove high level safety properties on a critical CBTC component?

Train mapping allows to locate the rear side of communicating trains, and the track elements occupied by non-communicating trains otherwise.

A preliminary study at system level (Octys) lead to the definition of three properties required by other components

Internal research project, based on a B based specification, joint work with ClearSy

Helps to specify the forthcoming B2HLL Translator

Train Mapping: Specification

Low level software specification in pseudo-B:

- Static constants : nb of trains, tracks, switchs, time thresholds, length
- Range of integer to characterize each element (switchs, trains, track elements)
- Some enums : switch positions (left/right/none), status of track section (free/unknown/occupied/…)
- Static description of the current line (which switch is on which track section, arrangement of track sections, etc)
- Sets of inputs (messages from the track, switchs; time stamps; messages from the trains)
- Sets of outputs (status of trains; locations of trains)
- Operations as modification of sets of internal definitions and outpts, specified as loops over system elements

Train Mapping: HLL Architecture

Translation into sHLL (HLL + while loops):

- Basic types as range of integers:

```
int [c_indet, c_nb_XX] t_XX;
```

- Global data structures by family:

```
struct {status: t_status,
        occupying: t_train,
        is_free: bool}          t_state_track;
t_state_track^(c_nb_track)      t_tab_state_track;
```

- Set of free input vars for messages:

```
input_track_status          t_status;
input_train_position        t_train_pos;
```

- Two tabs for each family:

```
tab_fam1_in           t_tab_fam1;
tab_fam2_out          t_tab_fam2;
```

- Each function takes input tabs and produce output tabs

- Global cyclic loop:

```
I(tab_fam1_in) := …;
X(tab_fam1_out) := (tab_fam1_in  with status :=
                        input_track_status);
```

Train Mapping: Topology

Line topology:

- Several line descriptions at different level
- Each is defined as an HLL function:

```
t_switch switch_on_track(t_track, t_direction);
switch_on_track(t,d):=
       if t = 2 // Track id
       then if d = 1 // Direction is Up
            then 2 // Switch Id
            else c_indet // No more switch
       else…;
```

- huge amount of data, especially since all possible itineraries are statically computed
- 150000 lines of HLL for a ZC in Line 5 (1 out of 5)

Train Mapping: Validation

- In the pseudo-B specification, 5 **coherence** properties has been stated as proved in the B model:

```
// If occupying is not indet, then status of a track
section cannot be free or unknown
ALL ts : t_track(
tab_track_out[ts].occupying != c_indet
<->
(tab_track_out[ts].status = c_occ1
 #
 tab_track_out[ts].status = c_occ2)
);
```

- Trying to prove these properties in HLL helped find some mistranslations of pseudo-B and typos, thanks to debugger
- Some coherence constraints on inputs have been added
- Performances may stall on complex functions, hard to debug

Train Mapping: Abstract Topology

Abstract topology:

- Functions are declared but not defined
- A set of constraints describes authorized configuration:

```
// A switch is on a unique track section
ALL ts1 :t_track, ts2:t_track, sw: t_switch
(SOME d: t_dir ( switch_on_track(ts1,d) = sw)
&
(SOME d: t_dir ( switch_on_track(ts2,d) = sw)
->
ts1 = ts2);
```

- This set has been validated through actual Line 5 and 1 data configuration
- Some constraints may be proved against other constraints = lemmas

Train Mapping: Abstract Topology

Recursive definitions helps to simplify the constraints:

```
exists_path(ts1,ts2) :=
  if exists_path_dir(ts1,ts2,up,c_nb_track) then up
  elif exists_path_dir(ts1,ts2,down,c_nb_track) then down
  else c_indet;

exists_path_dir(ts1,ts2,dir,nb) :=
 (nb != 0 & ts1 != c_cv_indet & ts2 != c_cv_indet & ts1 != ts2)
  &
  ( is_neighbor_dir(ts1,ts2,dir)
    #
    SOME tsi : t_track (
        is_neighbor_dir(ts1,tsi,dir)
        &
        exists_path_dir(tsi,ts2,dir,nb-1)
     )
);
```

Train Mapping: Validation

Refinement properties have been proved according to a higher level system description:

- What should happen when the software does not receive a message from a train for a long time?
- What should happen when the software receives a message from a train not already mapped?
- What should happen when the software receives a message from a train already mapped?
- How the software can merge several track sensors to improve mapping accuracy?
- How the software should follow lost equipment from ground sensors?
- How the system should sweep over the topology in order to clean mapping operations?

Train Mapping: Validation

What should happen when the software receives a message from a train already mapped?

- Condition:
  - no other train between last position and current
  - current position is inside ZC
  - Train structure has been maintained
- Then:
  - current track section is tagged with train
  - range of track section between last and current is cleaned, according to train status (exact -> free or approximate -> unknown)
  - approximation status of train is propagated

This property can be proved on abstract topology

Train Mapping: Safety Properties

However these properties cannot ensure alone general safety properties of the software

From global Octys safety analysis:

- All trains present in the ZC are represented in the global structures
- Their position is upstream their real position, up to the worst pullback
- Order in the line is equal to order in the representation

These **safety** properties need to refer to actual trains: a specific model of trains have been developed

Train Mapping: Train Model

Hypotheses:

- No train smaller than shunt holes
- Trains don't go back

Real trains:

- Superset of communicating trains
- Are identified by their real position and direction onto track sections
- Are partially ordered

Constraints:

```
// Trains are entering on a edge of the ZC
ALL tp : t_tp_reel (
~inside_ZC(tp) & X(inside_ZC(tp))
->
SOME ts : t_track (
  exists_real_path(ts,X(tp_real_arr(tp)),tp_real_dir(tp))
  &
  nb_neigbors(ts) = 1));
```

Train Mapping: Train Model

Constraints:

```
// Trains move along feasible paths
ALL tp : t_tp_reel (
inside_ZC(tp) & X(inside_ZC(tp))
->
exists_real_path(tp_real_arr(tp),X(tp_real_arr(tp)),tp_real_dir(tp))
&
exists_real_path(tp_real_avt(tp),X(tp_real_avt(tp)),tp_real_dir(tp))
);

// Switches don’t move under a train
ALL sw: t_switch, tp : t_tp_reel, ts : t_track, dir : t_dir (
    inv_track_dir_switch_div(ts,dir) = sw
    &
    is_under_train(tp,ts)
    ->
    X(position_switch(sw)) = position_switch(sw)
 );
```

Train Mapping Results

Metrics:

- Functional model: 2805 lines of code
- Abstract Topology: 945 lines
- Train Model: 1640 lines

Still under investigation!

Different level of properties: coherence, refinement,safety

Limited topology and trains evolution

Counter-examples under analysis for some sub-functions

Huge potential: all possible implementations are taken into account

Available Tools

Prover Technology: PSL

Systerel: S3

SafeRiver: SafeProver

HLL COMMUNITY

HLL community

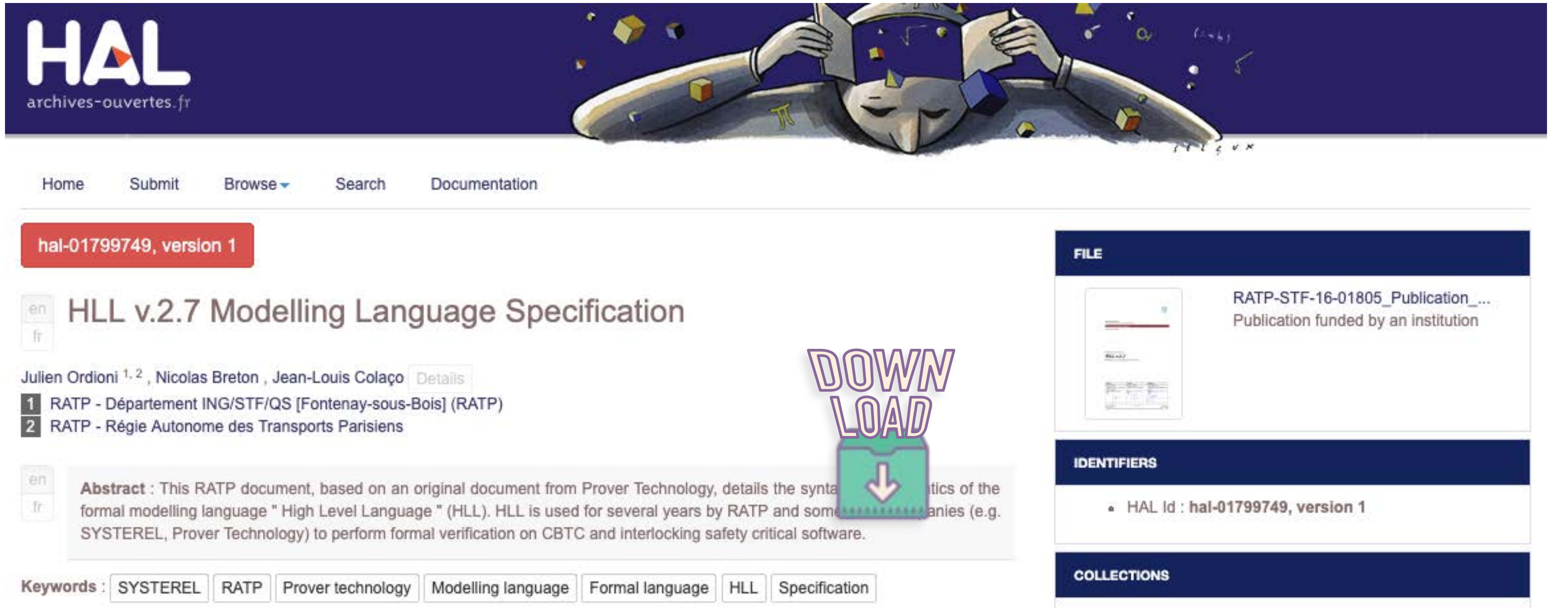
A community for HLL designers, users, editors and academics to:

- Share materials, knowledge, common basis
- **Build together** the evolution of HLL
- Guarantee **sustainability**
- Make **HLL a state of the art** of software validation

13th February 2018: HLL meeting with academics

ENS/UPMC/IRIF, INRIA, INP-ENSEEIHT/IRIT, LRI, LIP6, LORIA, CEA, ONERA

RATP position



- We are historical co-founder of HLL
- We are a (power) user of HLL
- We want to share HLL
- We want to show its efficiency, scalability and accuracy
- We need stability, sustainability and backward compatibility
- We have technical needs for our projects

Interested?

Join us!

- julien.ordioni@ratp.fr
- benjamin.blanc@ratp.fr

THANK YOU

RATP
INFRASTRUCTURES