An Account of the LNT Project (1998-2024)

Hubert Garavel

*joint work with **F. Lang, W. Serwe** and many others*

INRIA Grenoble – LIG – Université Grenoble Alpes

http://convecs.inria.fr

What is LNT?

- LNT: acronym for "LOTOS New Technology"
- A formal method designed to replace LOTOS
- Developed at INRIA Grenoble since 1998
- On-line resources about LNT:

  https://cadp.inria.fr/tutorial (see LNT section)

1. Design principles of LNT

Goals

- LNT is intended to describe critical systems
  - strong, nominal typing (no type inference)
  - static analysis (control-flow and data-flow analyses)
  - strictness (many compiler checks and warnings)

  $\Rightarrow$ catch many errors early, before exploring state spaces
- LNT is designed to be used by industry engineers
  - stay aligned with mainstream languages
  - ease of reading > ease of writing
  - simplicity: avoid esoteric symbols (CSP), omnipresent brackets (LOTOS), overloaded parentheses (µCRL), etc.

Synchretism and unification

- LNT combines ingredients from diverse sources:
  - functional programming languages
  - imperative programming languages
  - process calculi

  $\Rightarrow$ engineers and students already know 80% of LNT
- LNT provides sequential and parallel constructs
  - one can use the sequential part alone
  - the sequential part is a subset of the parallel part (contrary to LOTOS, SDL, FDR, μCRL, etc., which have two different languages for data and behaviour)

About minimality

- LNT is not "minimal" in the sense of the λ-calculus:
  - it provides **if-then-else**, **case**, and **alt** conditionals
  - it provides **while**-loops, **for**-loops, loops with **break**
  - it provides functions as a restricted form of processes

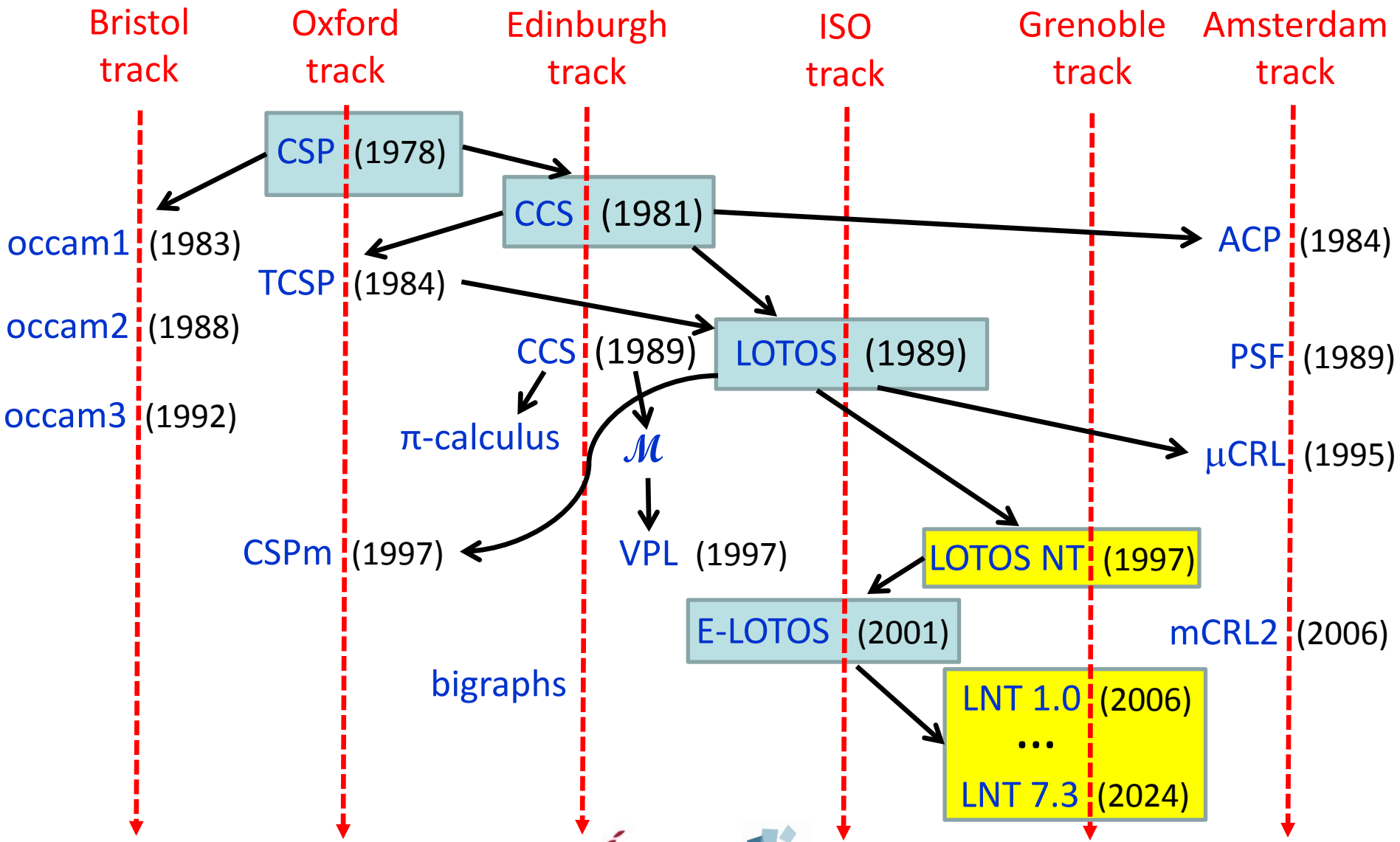
  ⇒ minimizing the number of LNT constructs is not a goal
- Alternative goals to be minimized:
  - differences between LNT and mainstream languages
  - time needed by "ordinary" engineers to learn LNT
  - time needed to write and read LNT models
  - size (number of lines) of LNT models

Concurrency

- Concurrent processes as first-class citizen
- Primitive concepts borrowed from process calculi
  - no shared memory between parallel processes
  - nondeterministic choice (on control branches and data)
  - multiway synchronous communication (rendezvous)
- Non-primitive concepts:
  - state machines (do not scale up to complex systems)
  - shared variables (too many possible semantics)
  - FIFO queues of messages

  $\Rightarrow$ all these concepts can be derived from primitive ones

Process calculi: a complicated story



Main sources of inspiration for LNT (1)

- GCL (*Guarded Command Language*) – E. Dijkstra (1975)
  - nondeterministic choice
- CSP (*Communicating Sequential Processes*) – T. Hoare (1978)
  - concurrent processes without shared memory
  - atomic synchronous communication (rendezvous)
- CCS (*Calculus of Communicating Systems*) – R. Milner (1980)
  - semantics: LTS, τ-transitions, SOS rules, bisimulations…
- SML (*Standard Meta Language*) – R. Milner (1983)
  - constructor types, pattern-matching "**case**"

Main sources of inspiration for LNT (2)

- occam – D. May @ INMOS (1983)
  - proof that CSP can evolve into an industrial language
- Ada – J. Ichbiah et al. @ Honeywell Bull (1983)
  - clever syntax for structured programming constructs
- NIL / Hermes – R. Strom et al. @ IBM (1984)
  - static detection of uninitialized variables ("typestate")
- LOTOS – ISO standard 8807 (1989)
  - processes parameterized by gates, disable operator
- E-LOTOS – ISO standard 15437 (2001)
  - functional data types instead of ADTs, imperative style

Functional or imperative style?

- Situation:
  - abstract data types in LOTOS / SDL / μCRL are rejected
  - functional programming is not widely adopted
  - E-LOTOS' functional/imperative mix is unsatisfactory

  ⇒ LNT adopts a "truly imperative" style

- But "mutable" variables may raise semantic issues:
  - side effects in expressions, especially Boolean guards
  - write-write or read-write conflicts on shared variables
  - variables used but not assigned before

Static analysis

- To avoid semantic issues with the imperative style: static analysis (aka control and data-flow analyses)
- Two main roles:
  - preserve semantics (e.g., forbid uninitialized variables)
  - emit pertinent warnings about dubious parts of code
- Practical issues:
  - static analysis algorithms are involved and error-prone
  - they address undecidable questions (~halting problem)
  - they are pessimistic (may reject correct LNT programs)

Example 1

```
var X, Y: nat in
    INPUT (?X);
    if X < 100 then
       Y := 1
     end if;
     if sqrt (X) < 10 then
        Y := Y + 1    -- is Y properly initialized here?
      end if
end var
```

The exact frontier between correct and incorrect LNT models depends on compiler's cleverness

Example 2

```
par
    X := 0
||
    while false loop
        X := 1
    end loop
end par
```

-- should the compiler report a write-write conflict
-- on variable X in the parallel composition?

The frontier between correct and incorrect models is also a matter of personal taste

2. Development tools for LNT

Executability

- Specifications vs programs:
  - specifications are declarative, programs are imperative
  - such a difference is advocated by Z, TLA+, etc.
  - but engineers dislike doing the work twice
- LNT (as CSP, LOTOS, etc.) makes no such difference:
  - Traditional concept of *executable formal method*
  - LNT is detailed enough to express algorithms
  - LNT models are meant to be executable
    (at least with simulation or rapid prototyping)
  - Yet, LNT has nondetermism, pre-/post- conditions…

Implementing LNT

- For a new language such as LNT, one needs compilers/translators
- INRIA Grenoble has been developing tools for LNT since 1998
- Four successive (yet overlapping) phases

1998-2018: TRAIAN 1 & 2

- PhD thesis of Mihaela Sighireanu (1999) contributions to E-LOTOS ("LOTOS NT" dialect)
- TRAIAN: a compiler (or "transpiler") for LOTOS NT
  - only handles LOTOS NT types and functions
  - generates C code (no need for LNT-specific byte code)
  - written using attribute grammars (SYNTAX + FNC2)
  - 11 releases of TRAIAN: v1.0 (1998) → v2.9 (2019)
- TRAIAN is heavily used for compiler construction
  - 13 compilers written using SYNTAX + TRAIAN
  - most of their code (63-91%) is written in LNT itself

Compilers/translators built using TRAIAN

compiler	LNT lines	C lines	SX lines	LNT ratio
PIC2LNT	3712	430	1711	63.4%
NTIF	7046	1273	1387	72.6%
AAL	7849	934	1591	73.1%
SVL	9089	476	3025	72.2%
CTRL2BLK	9871	466	580	90.4%
CHP2LOTOS	10,323	1871	1570	75.0%
EXP.OPEN	11,569	3458	1536	69.8%
ATLANTIF	13,738	393	1433	88.3%
FSP2LOTOS	20,449	2639	4163	75.0%
TRAIAN 3.8	33,076	5564	3700	78.1%
GRL2LNT	37,738	1851	1759	91.3%
LNT2LOTOS	38,610	2836	4390	84.2%
MCLEXPAND	43,337	6641	4364	79.7%

2006-2020: LNT2LOTOS

- LNT2LOTOS: a translator from LNT to LOTOS
  - developed at Bull's request (to get rid of LOTOS ADTs)
  - enables reuse for LNT of the existing CADP tools
  - started with LNT types and functions
  - progressively expanded to handle LNT processes
  - "lightweight" translation: no type checking, etc.
    most checks deferred to the target LOTOS compiler
- Since 2010: LOTOS abandoned at INRIA Grenoble
  - replacement of LOTOS by LNT
  - LNT successfully used in 30+ cases studies

2016-2020: TRAIAN 3.0

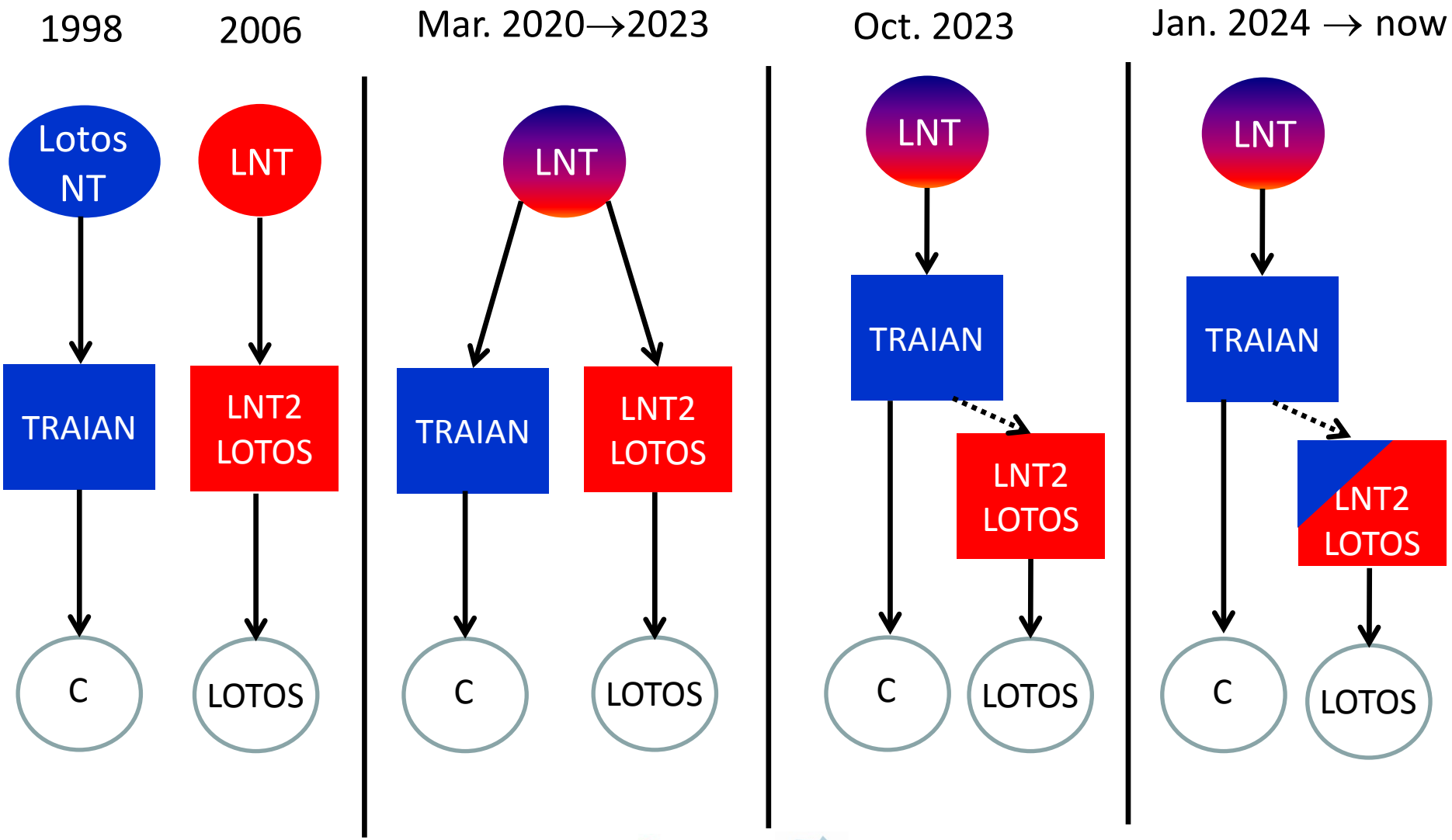
- Practical issues with TRAIAN 2:
  - FNC2 attribute grammars were verbose and tedious
  - FNC2 was no longer maintained (and no source code)
  - FNC2 executables were 32-bit, hitting 3-4 GB limit

  $\Rightarrow$ maintenance and evolution of TRAIAN 2 was difficult
- 2016-2020: complete rewrite of TRAIAN
  - SYNTAX+FNC2 replaced by SYNTAX+LNT technology
  - TRAIAN 3.0: entirely different from TRAIAN 2.9, yet producing exactly the same C code (modulo renaming)
  - TRAIAN 3.0 bootstrapped using TRAIAN 2.9

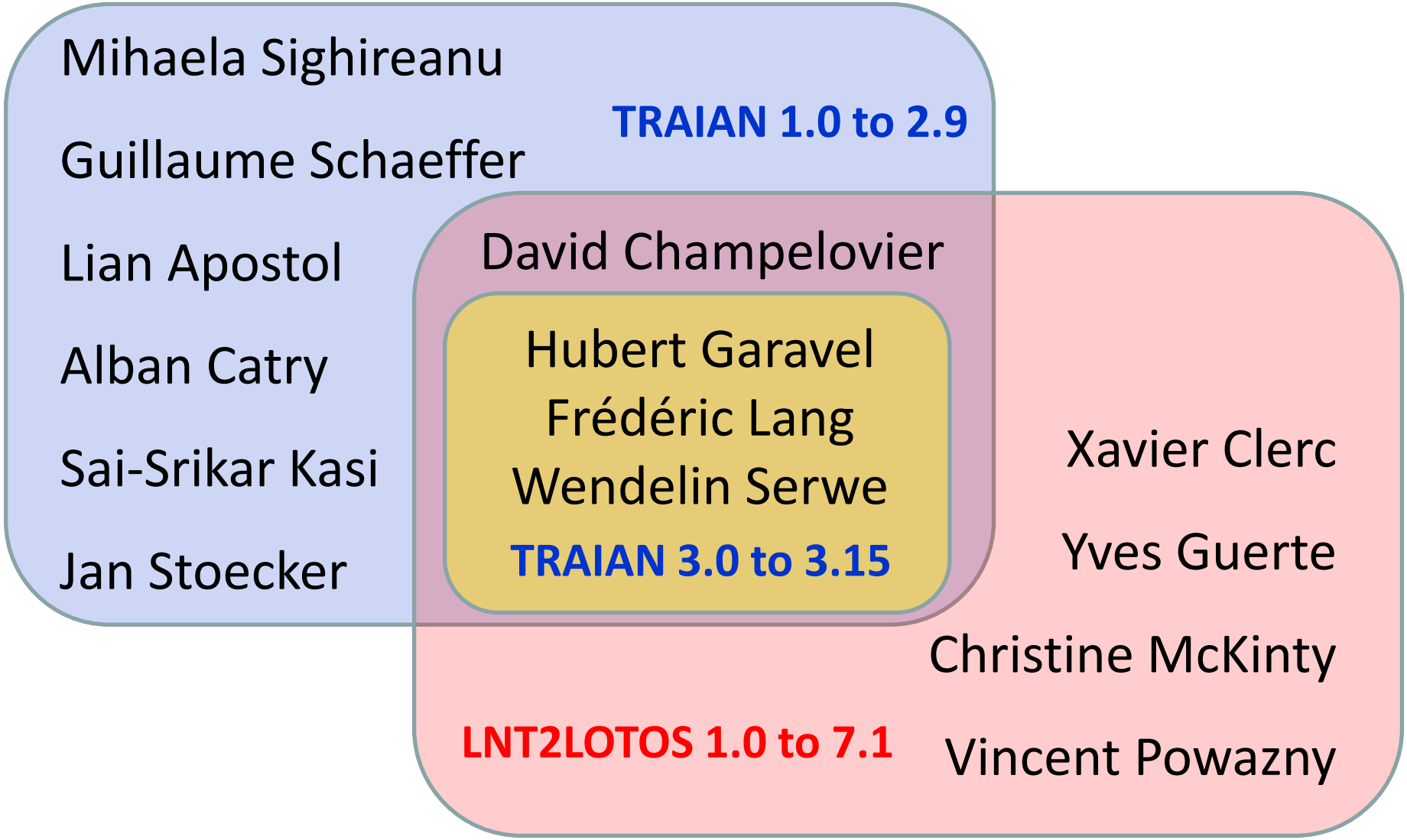
2020-now: The Great Convergence

- 2020: Two different LNT languages and compilers
  - TRAIAN 3.0: produces C code for LNT types/functions
  - LNT2LOTOS: produces LOTOS code (handles processes)
- Practical issues:
  - both compilers were incompatible in many details
  - we could not maintain two different LNT dialects
- We progressively evolved both compilers:
  - discussion and selection of the "best" features for LNT
  - unification of syntax, semantics, libraries, tests, docs
  - TRAIAN is now the front-end called before LNT2LOTOS

Great Convergence steps



The LNT team(s)



3. Conclusion

Summary

LNT: a computer language combining two different models of computation:

- Sequential computation (types and functions)
  - application domain: compiler construction
  - so far: 13 compilers/translators written in LNT
- Parallel computation (processes and events)
  - application domain: hardware/software/telco systems
  - so far: 30+ case studies done with LNT
  - 15 translators "X → LNT" developed

Current status

- LNT exists and is operational:
  - since 2010, LNT fully replaces LOTOS in Grenoble
  - using LNT does not increase the size of state spaces
  - LNT used by several companies
  - LNT used to teach concurrency in universities

- Robust compilers for LNT are available:
  - TRAIAN (58,000 lines of code): 4 releases / year
  - LNT2LOTOS (45,000 lines of code): 12 releases / year
  - LNT test suites totalling 15+ million lines of code

Next steps

- The LNT language is (slightly) evolving:
  - based on case studies and "X → LNT" translators
  - feedback/suggestions welcome
- The LNT tools are evolving fast:
  - better error messages for novice users
  - more precise static analyses
  - separation of roles between TRAIAN and LNT2LOTOS (LNT2LOTOS → LOTOS code generator)

Possible collaborations

- Upgrade old formal models to LNT:
  - can LNT replace prior formal methods?
  - feedback welcome to enhance LNT
  - papers for MARS@ETAPS workshops
- Create back-ends for LNT:
  - TRAIAN could export a decorated abstract tree (XML or JSON)
  - new translators "LNT → X" could be developed (in addition to LNT2LOTOS)