FROM RESEARCH TO INDUSTRY

cea tech

Compilation and Runtime Code Generation for Performance and Security in Embedded Systems

DCE 2016 workshop during CGO / Barcelona

Damien Couroussé,
Henri-Pierre Charles

January 2016

www.cea.fr

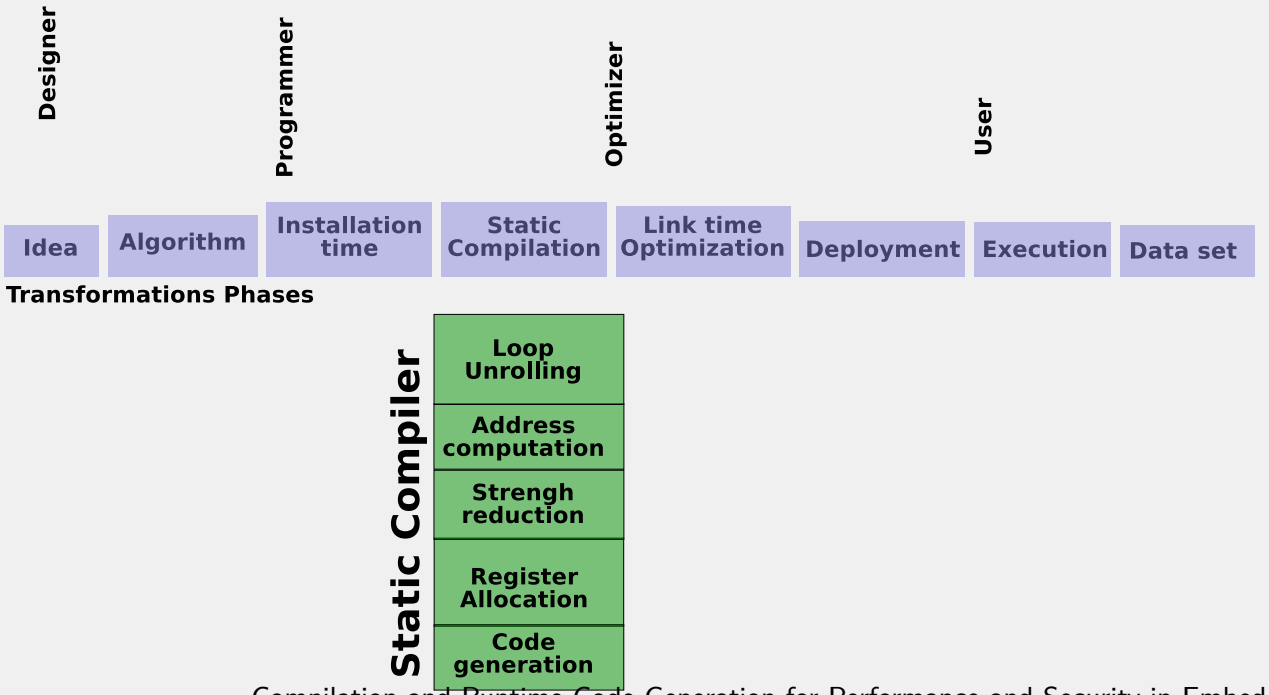
leti & list

Static Compiler Chain

Classical Compiler architecture : GCC, LLVM

- Driven by performance only
- Mono architecture
- Not energy aware
- Not data dependent

Before / After

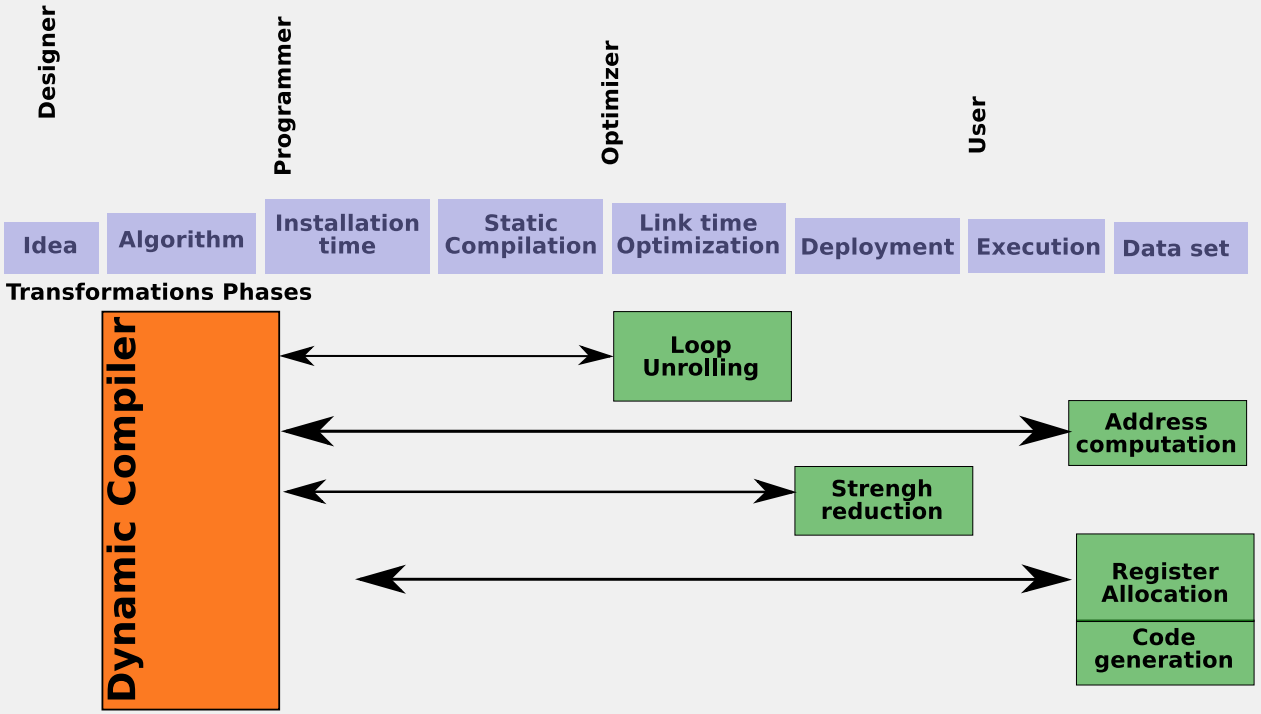


Dynamic Compiler Chain

Future Compilers Architecture

- Multi objective (execution time, power, thermal constraints)
- Multi-target (Heterogeneous multi SoC)
- Data driven (dynamically)

Before / After



Definitions

Definitions

Static compilation "classical" binary code generation (gcc, icc, clang, ...)

Dynamic Compilation binary code generated at run-time (DBT)

JIT run-time dynamic compilation based on complex Intermediate representation (Java, LLVM)

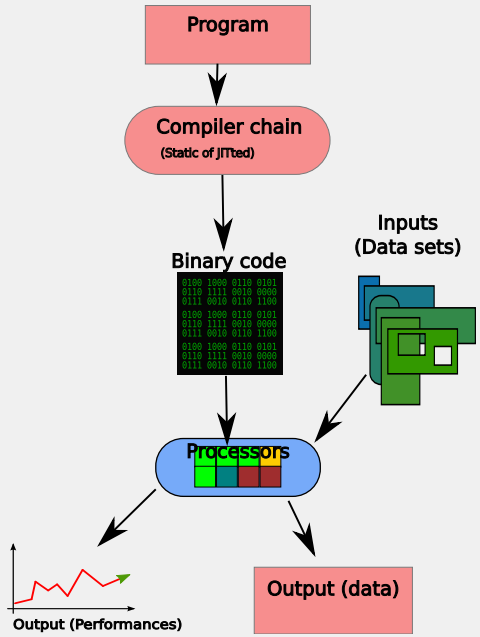
Innovations

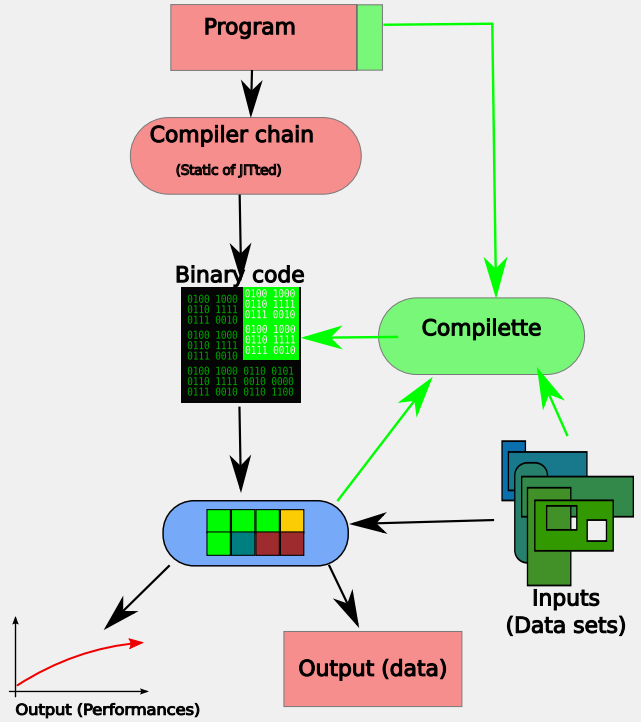
Compilette : small binary code generator embedded into application able to optimize code depending on data sets

deGoal : a tool which help to generate *Compilettes*

Static Compiler Versus Compilette

Static or dynamic code generation for performance





Static Compiler

- Run once
- Does not know data set characteristics
- Slow compilation (even with JIT)

Compilette

- Adapt on the fly
- Knowledge of the architecture
- Knowledge of the application

Features

deGoal features

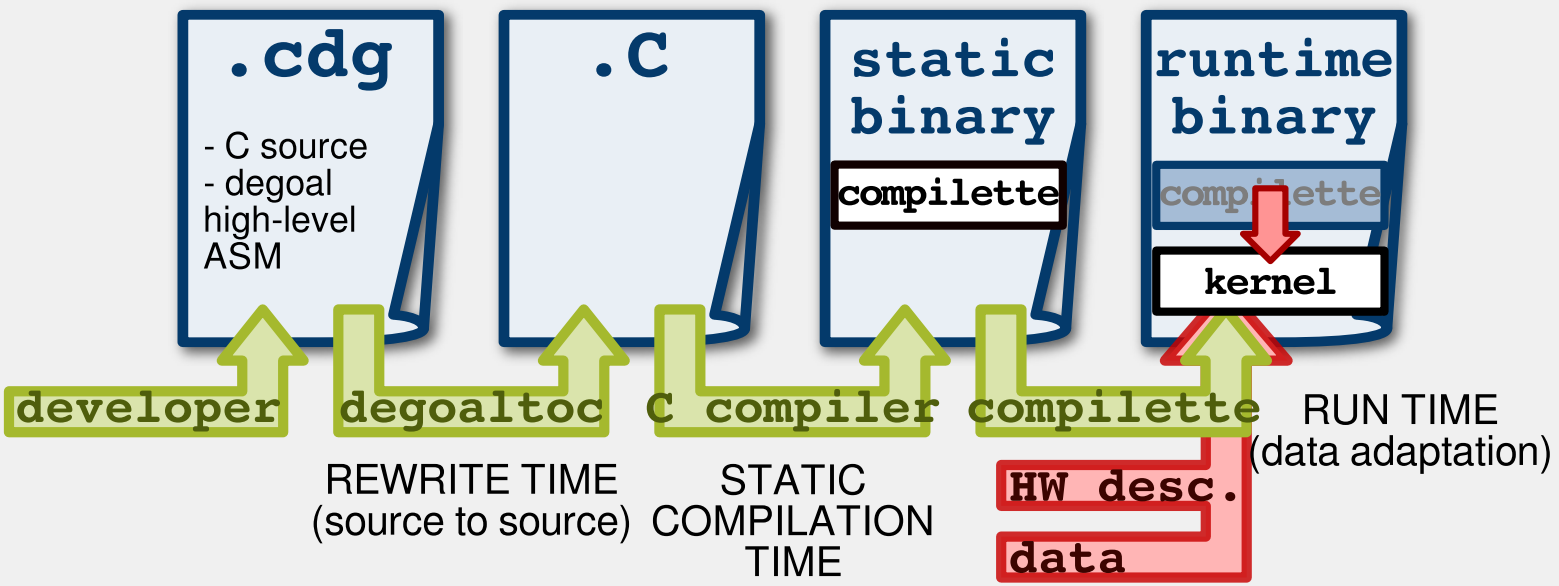
- Portable "assembly language"
- Source to source compiler
- Registers
  - Typed : `int`, `float`, `complex`, ...
  - Vector support : dynamic size
- Mix runtime data & binary code
- Correct use of any multimedia instruction

Obtained results

- Auto adaptive dynamic libraries
- Runtime Portable Optimization
- Multiple metrics :
  - Faster generated code
  - Smaller generated code
  - 3 order of magnitude faster than JIT/LLVM
  - 4 order of magnitude smaller than JIT/LLVM

Objective : Binary Code Generation (Data dependent & Architecture dependent)

Compilation flow



deGoal support

Architecture	Port status	SIMD support	Instruction bundling
ARM Thumb-2 (+NEON/VFP)	✓	✓	OoO/InO
STxP70 (STHORM / P2012)	✓	N/A	✓
K1 (Kalray MPPA)	✓	✓	✓
PTX (GPU NVIDIA)	✓	✓	N/A
ARM32	✓	✗	N/A
MSP430	✓	N/A	N/A
MIPS	✓	N/A	N/A
ARM64	C	✗	N/A
X86	C	✗	N/A

Code Polymorphism
–
Runtime Code Generation as a Software Protection for Embedded Systems

Polymorphic runtime code generation

Definition

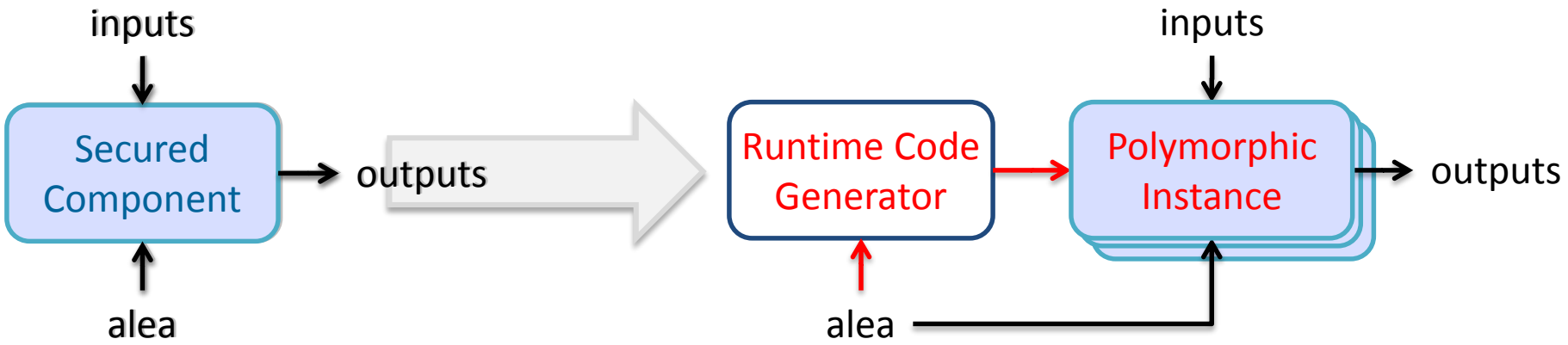
- Regularly **changing the behavior** of a (secured) component, **at runtime**, while maintaining **unchanged** its **functional properties**, with runtime code generation

© CEA. All rights reserved DACLE Division | January 2014 | 11

Polymorphic runtime code generation

Definition

- Regularly **changing the behavior** of a (secured) component, **at runtime**, while maintaining **unchanged** its **functional properties**, with runtime code generation



Lightweight

Portability: add security mechanisms everywhere it is needed.
no need for dedicated hardware,
but can cooperate with security-oriented hardware

Flexibility: customizable level of security, upgradable, patch-able.

Code polymorphism

Definition

- Regularly **changing the behavior** of a (secured) component, **at runtime**, while maintaining **unchanged** its **functional properties**, with runtime code generation

What for?

- Protection against reverse engineering of SW
  - **the secured code is not available before runtime**
  - **the secured code regularly changes its form (code generation interval ω)**
- Protection against physical attacks
  - polymorphism changes the spatial and temporal properties of the secured code: side channel & fault attacks
  - combine with usual SW protections against focused attacks
- Compatible with State-of-the-Art HW & SW Countermeasures

How?

- deGoal: runtime code generation for embedded systems
  - fast code generation
  - tiny memory footprint: proof of concept on TI's MSP430 (512 B RAM)
  - Easy targeting of application-specific instrutions or HW features

leti & list
© CEA. All rights reserved DACLE Division | January 2014 | 13

Countermeasures & polymorphism

State of the Art

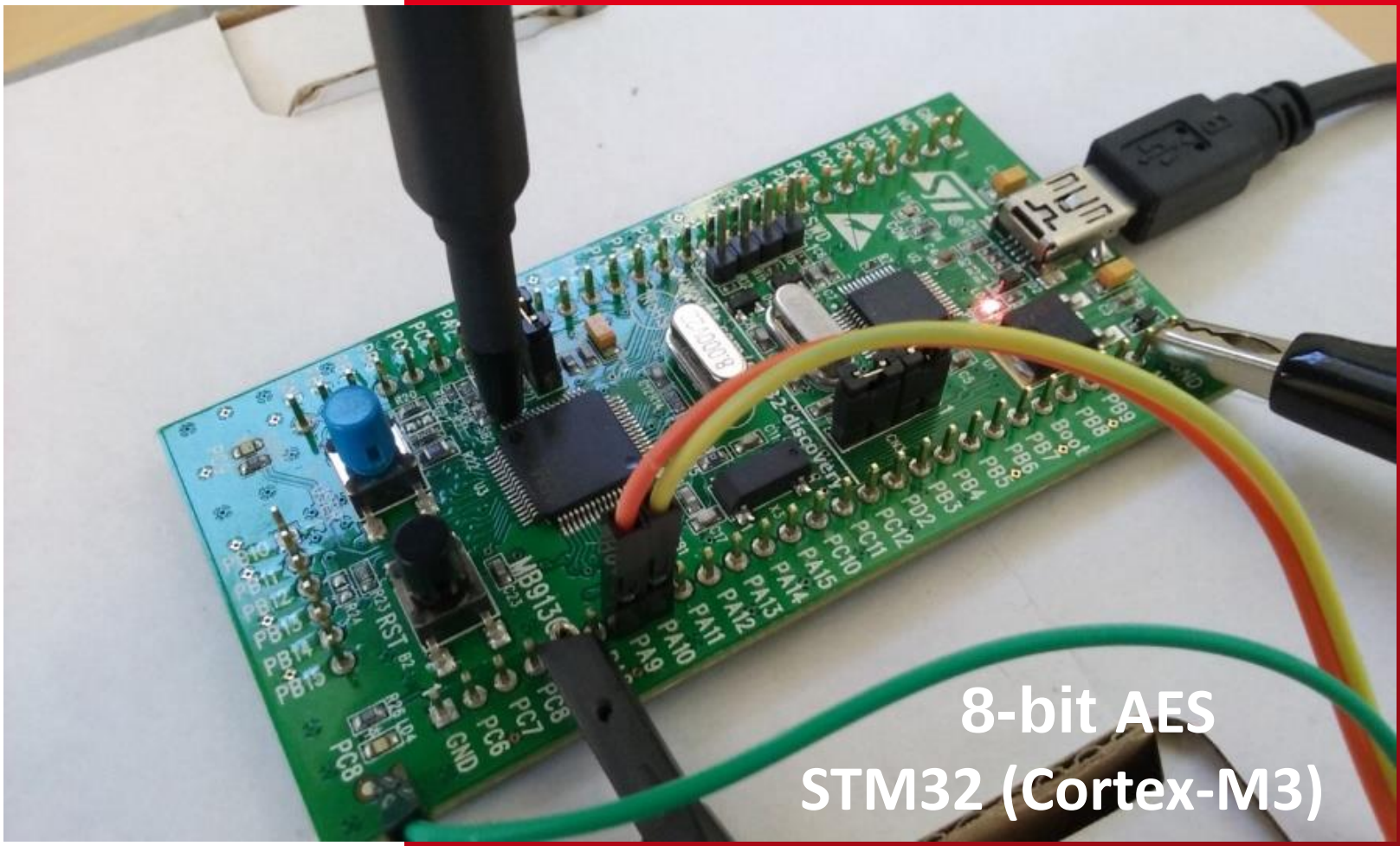
- Random register renaming [May 2011a, Agosta 2012]
- Out-of-Order execution :
  - At the instruction level [May 2011b, Bayrak 2012]
  - At the control flow level [Agosta 2014, Crane 2015]
- Execution of dummy instructions [Ambrose 2007, Coron 2009, Coron 2010]
- A few proof-of-concept implementations, not suitable for embedded devices [Amarilli 2011, Amarilli 2011, Agosta 2012]

Our approach

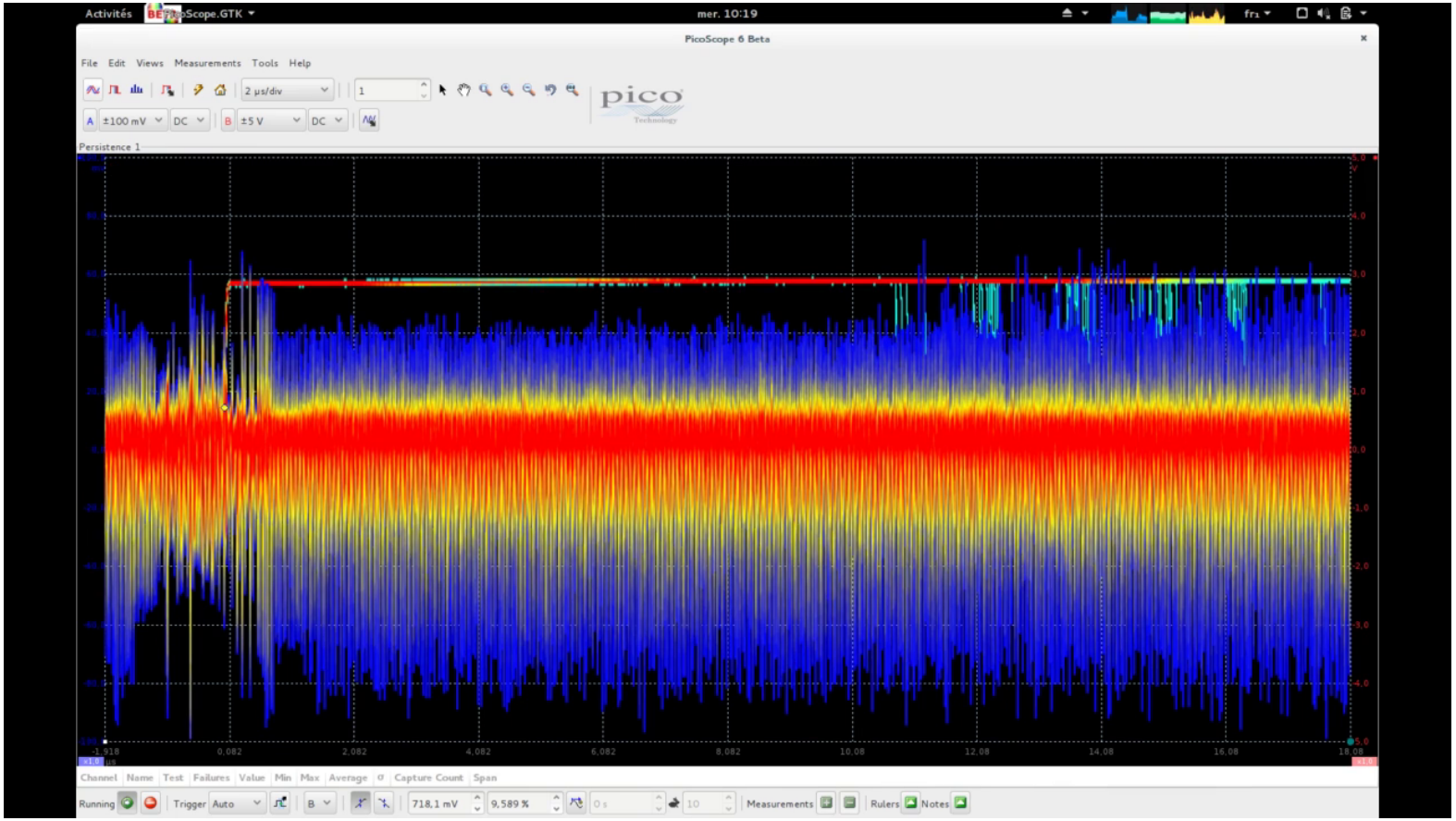
- Pure software → portability, genericity
- Combination of *all* the polymorphic levers found in the state of the art,
  - Currently at the basic block level
- Modest overhead (execution time & memory footprint)
- With runtime code generation

© CEA. All rights reserved DACLE Division | January 2014 | 14

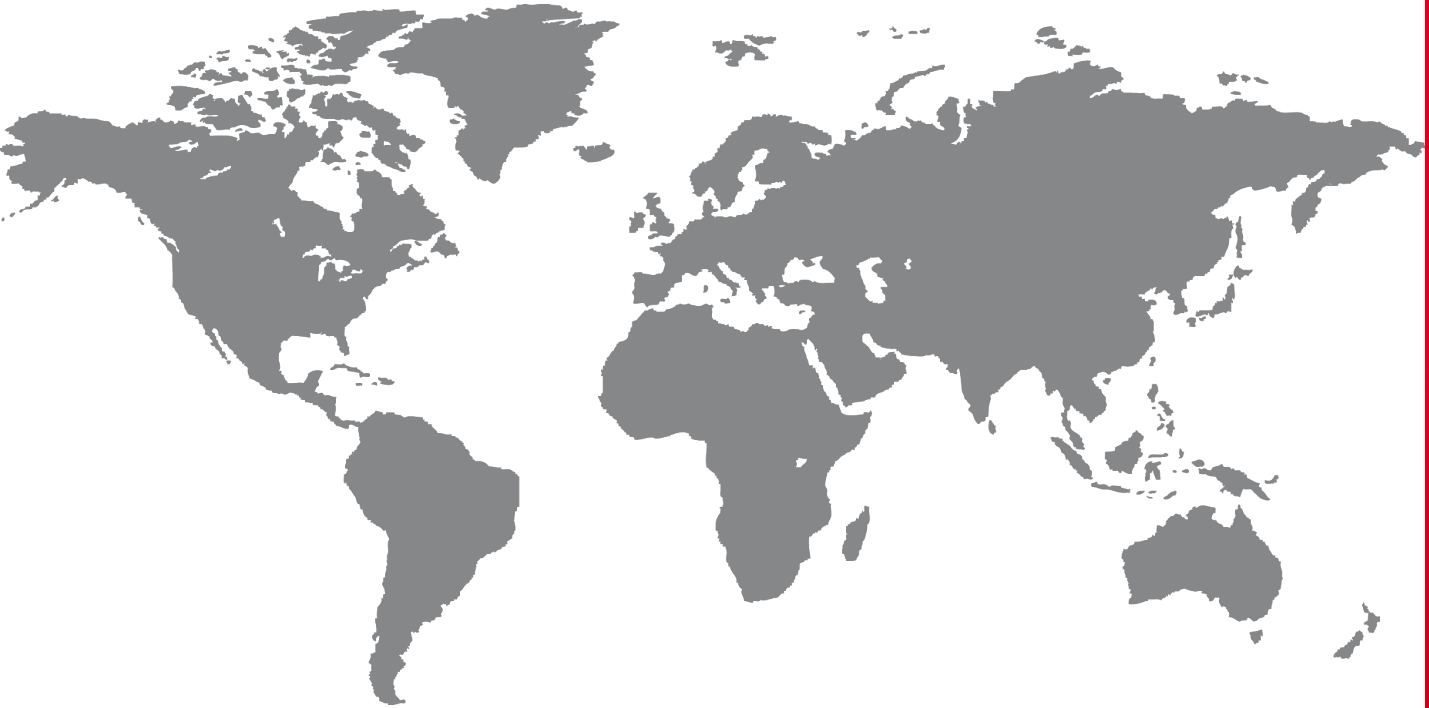
Demo



Demo



leti & list



Let’s discuss !

INSTITUT CARNOT CEA LETI

INSTITUT CARNOT CEA LIST

MINATEC®

digiteo

leti

Centre de Grenoble
17 rue des Martyrs
38054 Grenoble Cedex

list

Centre de Saclay
Nano-Innov PC 172
91191 Gif sur Yvette Cedex