SPARK: Retours d'expérience et intérêt en transport

Présenté par Yannick Moy

AdaCore

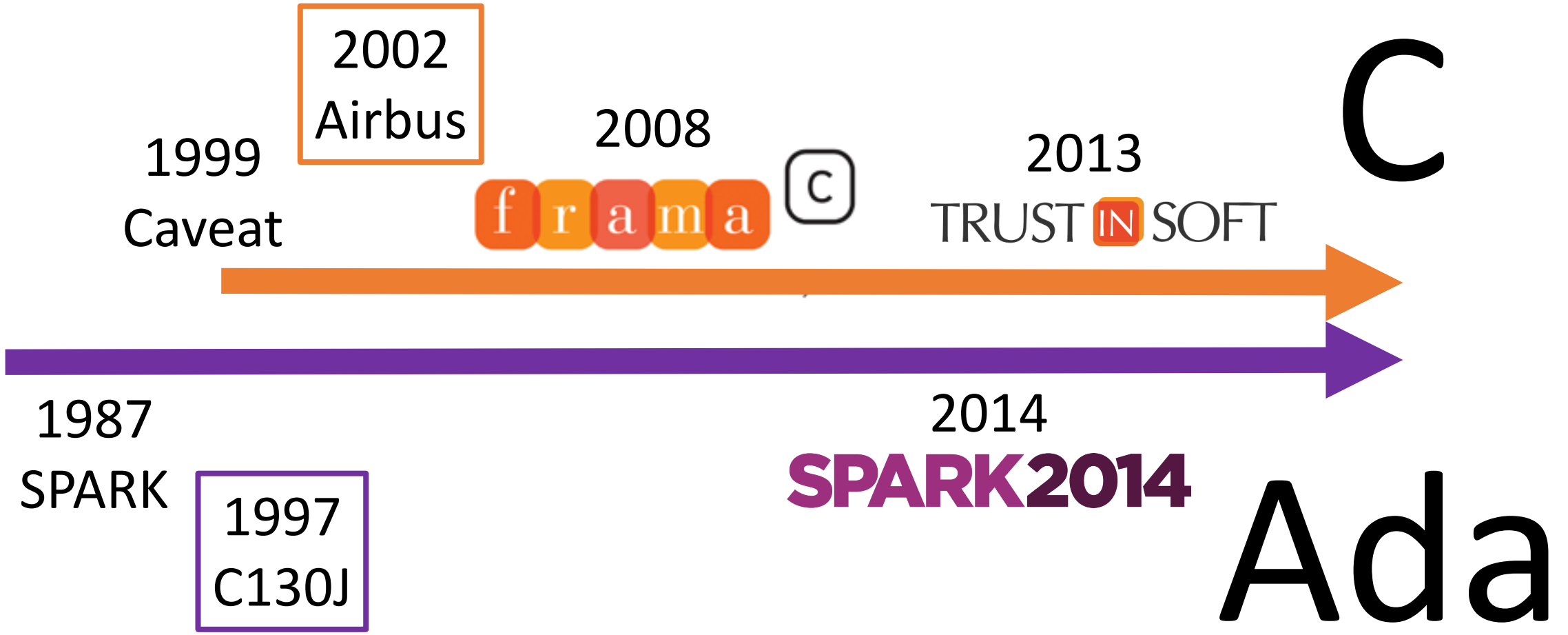
SPARK – Flow Analysis

```
procedure Stabilize (Mode    :  in Mode_T;
                     Success : out Boolean)
  with Global => (Input  => (Accel, Giro),
                  In_Out => Rotors);
```

SPARK – Proof

```
procedure Stabilize (Mode    :  in Mode_T;
                     Success : out Boolean)
   with Pre  => Mode /= Off,
        Post => (if Success then
                    Delta_Change (Rotors'Old, Rotors));
```

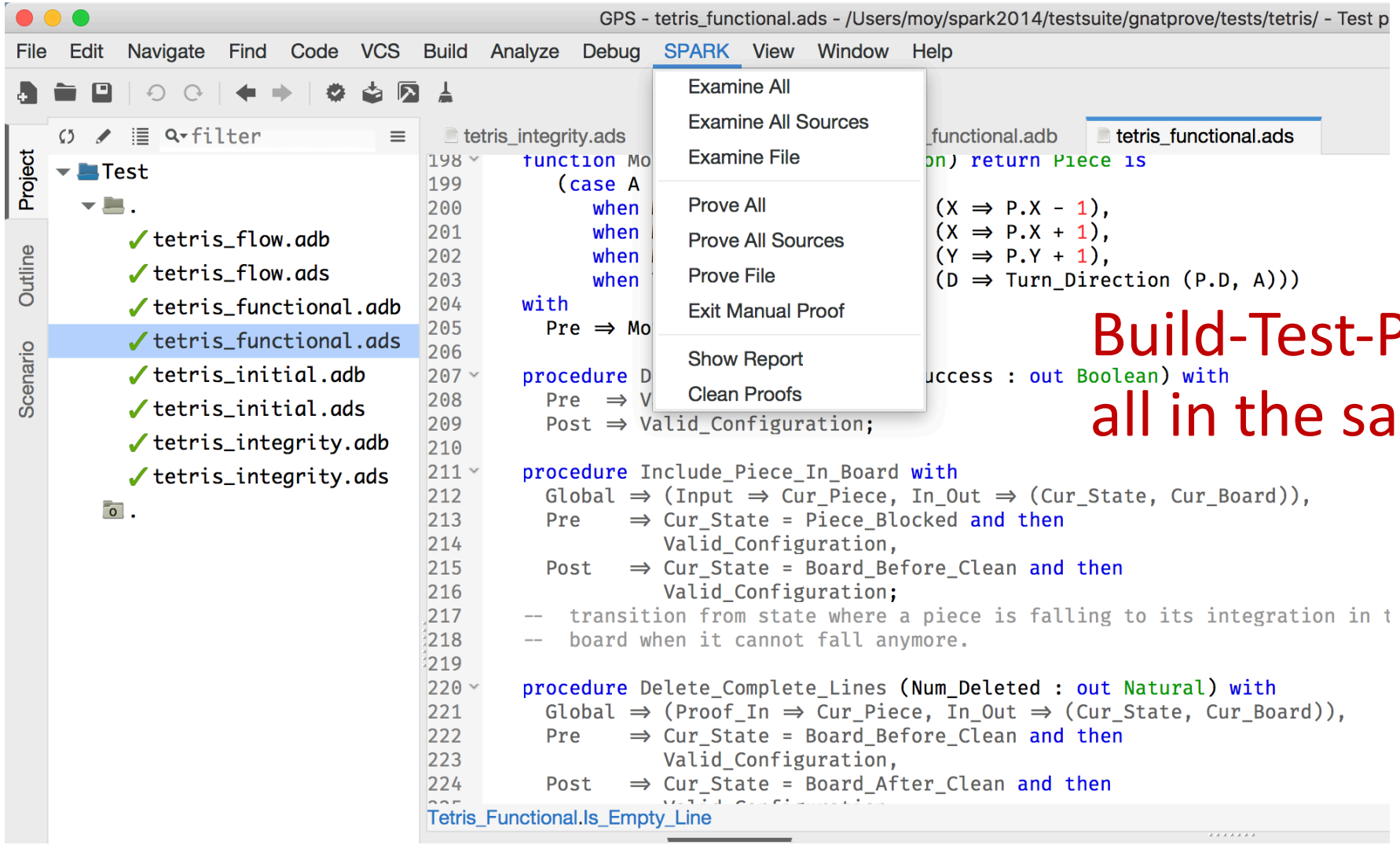
Formal Program Verification



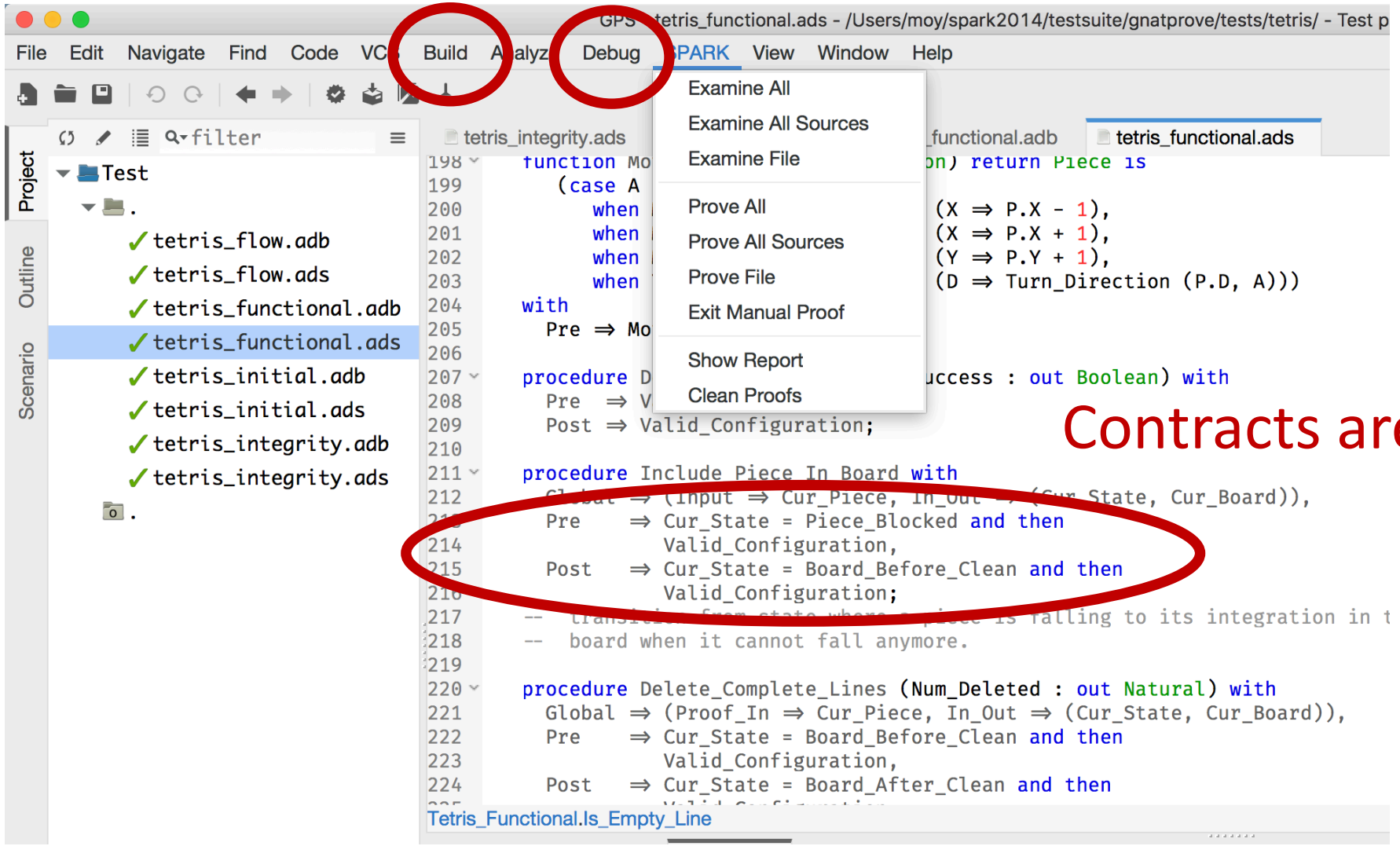
Formal Verification of Avionics Software Products, Souyris et al., Airbus, FM 2009

Are We There Yet? 20 Years of Industrial Theorem Proving with SPARK, Chapman and Schanda, Altran, ITP 2014

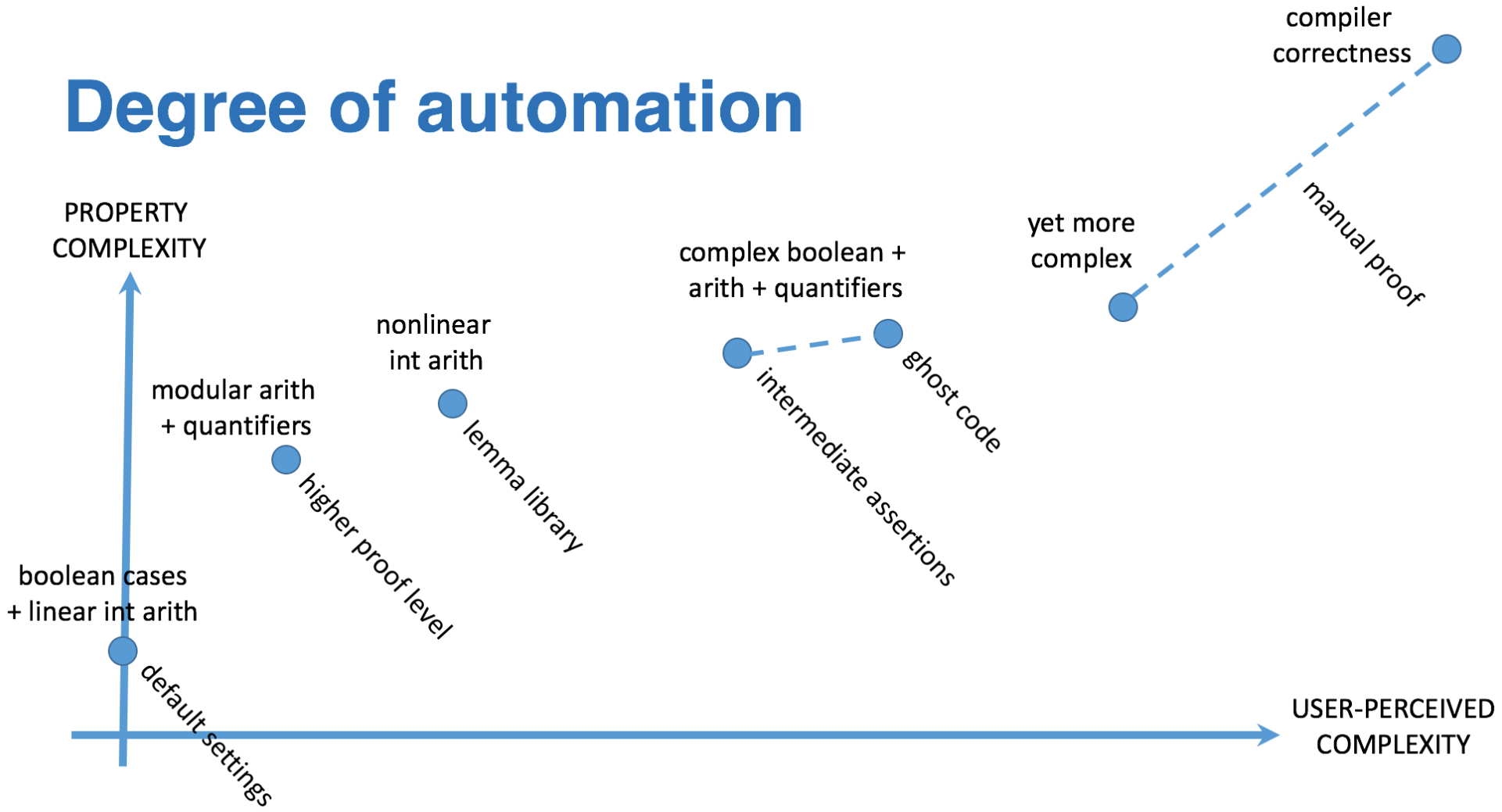
Key Features - Integration



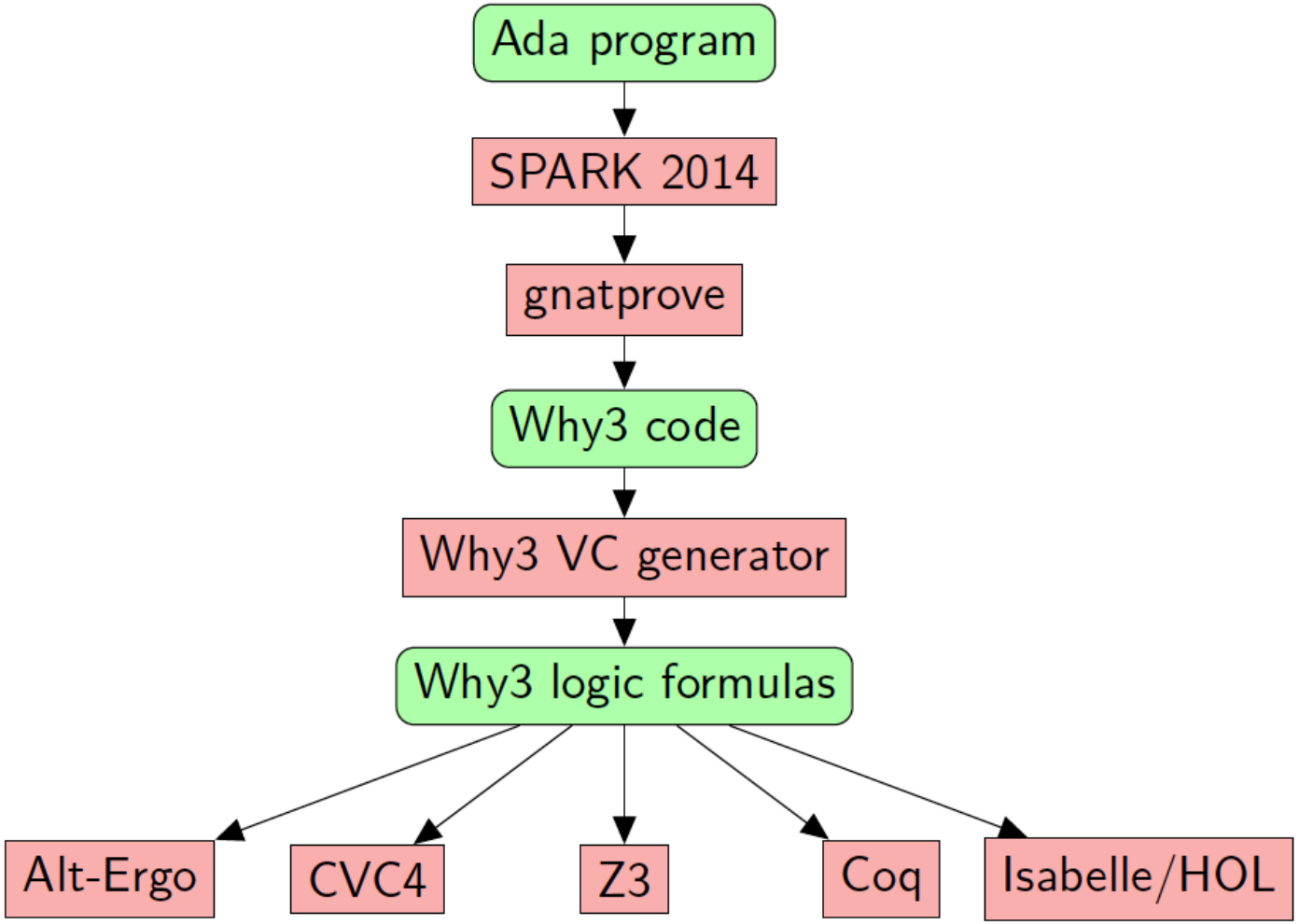
Key Features - Integration



Key Features - Automation



Key Features - Automation



Key Features - Interaction

Path

saturation.adb

```
5    type Saturable_Value is record
6       Value : Unsigned_16;
7       Upper_Bound : Unsigned_16;
8    end record;
9
10   function Saturate (Val : Saturable_Value) return Saturable_Value
     --  Val = (Value => 16383, Upper_Bound => 49152)
11     with SPARK_Mode,
12     Post =>
13       (if Val.Value <= Val.Upper_Bound then
           Saturate'Result = (Value => 49152, Upper_Bound => ?) and
         --  Val = (Value => 16383, Upper_Bound => 49152)
14         Saturate'Result.Value = Val.Value) and
15       (if Val.Value > Val.Upper_Bound then
16          Saturate'Result.Value = Val.Upper_Bound)
17   is
18   begin
19     return Val'Update
       --  Saturate'Result = (Value => 49152, Upper_Bound => 49152)
20       (Value => Unsigned_16'Max (Val.Value, Val.Upper_Bound));
21   end Saturate;
```

Counterexample

Levels of Software Assurance

Stone Level

Strong semantic coding standard

Program respects all the SPARK language legality rules

Enforces safer use of language features:

- Restricted concurrency (Ravenscar profile)
- Expressions and functions without side-effects

Forbids language features that make analysis difficult:

- Unrestricted pointers
- Exception handlers

Bronze Level

Initialization and correct data flow

Program passes SPARK flow analysis without violations

Detects programming errors:

- Read of uninitialized data
- Problematic aliasing between parameters
- Data race between concurrent tasks

Checks user specifications:

- Data read or written
- Flow of information from inputs to outputs

Silver Level

Absence of run-time errors

Program passes SPARK proof without violations

Detects programming errors:

- Divide by zero
- Array index out of bounds
- Integer, fixed-point and floating-point overflow
- Integer, fixed-point and floating-point range violation
- Explicit exception raised
- Violation of Ceiling Priority Protocol

Gold Level

Proof of key integrity properties

Program passes SPARK proof without violations

Checks user specifications:

- Type invariants (weak and strong)
- Preconditions
- Postconditions

Checks correct use of OO wrt Liskov Substitution Principle

Platinum Level

Proof of full functional correctness

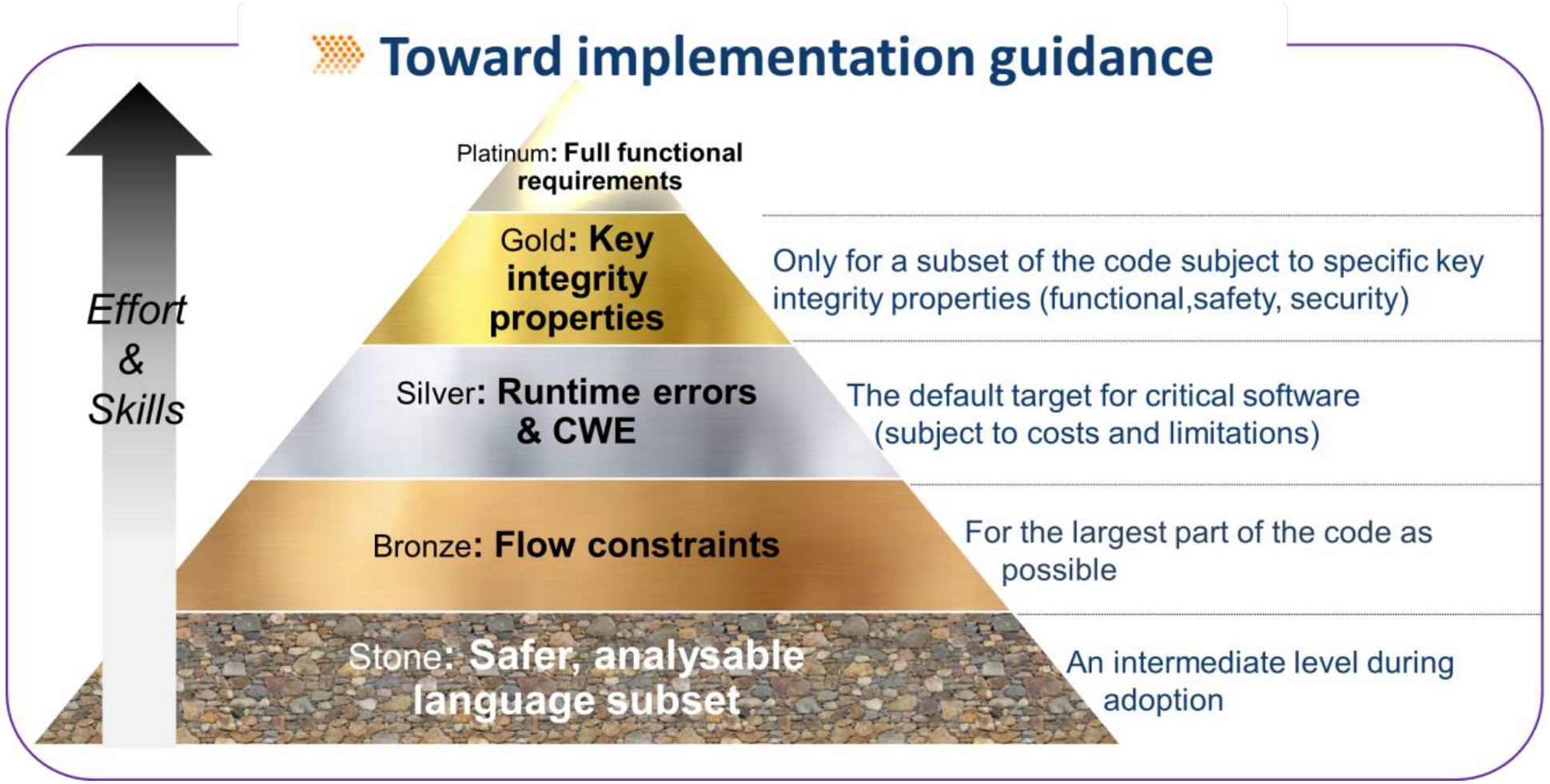
Program passes SPARK proof without violations

Checks complete user specifications:

- Type invariants (weak and strong)
- Preconditions
- Postconditions

Checks loop termination (loop variant)

Software Assurance Levels



Industrial Practice

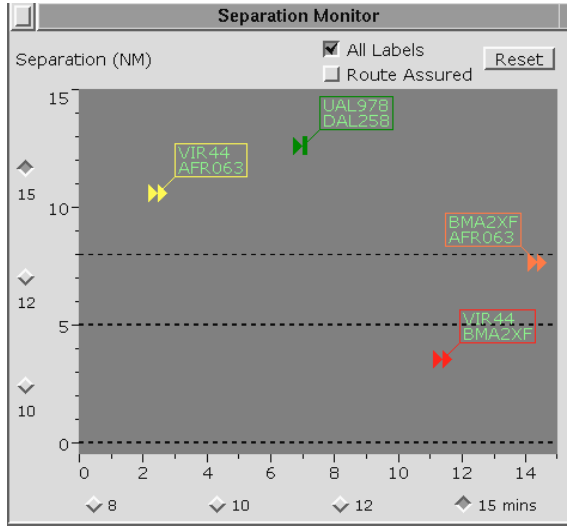
Established Practice at Altran UK

Software Integrity Level		SPARK Software Assurance Level			
DAL	**SIL**	**Bronze**	**Silver**	**Gold**	**Platinum**
A	4				
B	3				
C	2				
D	1				
E	0				

Past Projects at Altran UK

SHOLIS: 1995
DEFSTAN 00-55 SIL4
First Gold

C130J: 1996 - now
Bronze (Lockheed Martin) and Gold (UK RAF and BAE Systems)

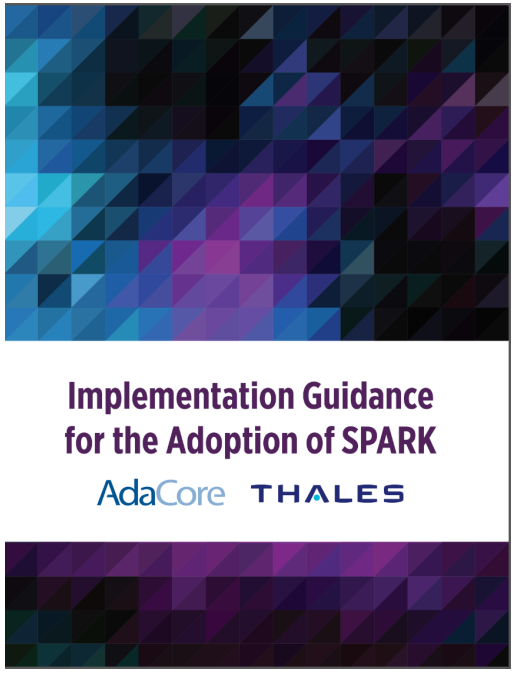


iFACTS: 2006 - now
Silver (NATS)

Adoption Experiments at Thales

Use case 1: porting to new platform context: 300 klocs radar software target: Stone level significant manual refactoring (several days) on the way to completion on 300 klocs	Use case 2: demonstrate compliance to LLR context: small numerical function target: Gold level difficulties in expressing suitable context property was not proved automatically
Use case 3: identify and fix weakness context: 100s slocs code generator target: Gold level half a day to reach Silver property related to inner memory bounds two days to reach Gold	Use case 4: guarantee safety properties context: 7 klocs command & control target: Gold level one day to reach Silver property expressed as automaton four days to reach Gold

Adoption Guidelines with Thales



For every level, we present:

- Benefits, Impact on process, Costs and limitations
- Setup and tool usage
- Violation messages issued by the tool
- Remediation solutions

Guidance was put to test:

- During adoption experiments at Thales
- On example (SPARK tool) presented in last section

Features that Matter

Stone Level – Large Language Subset

`SPARK_Mode => On`

- Ada types, expressions, statements, subprograms

  `SPARK_Mode => Off`

  - Ada pointers
  - Ada exception handlers

- Ada generics
- Ada object orientation
- Ada concurrency

Stone Level – Large Language Subset

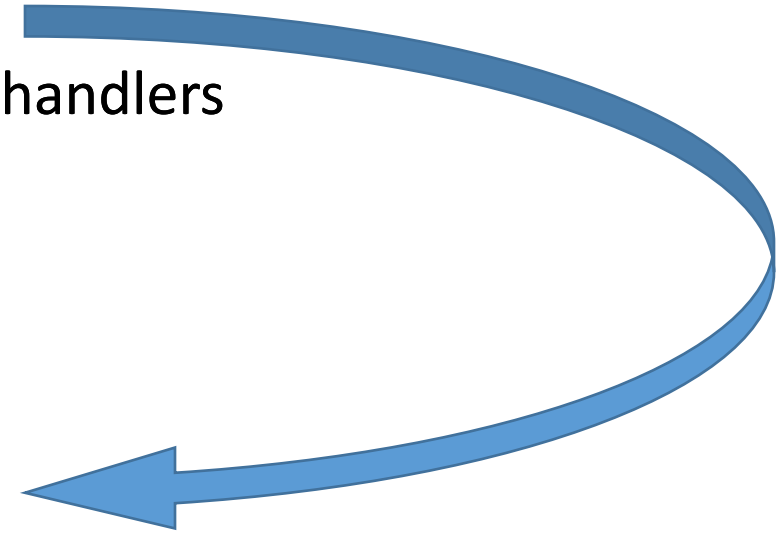
`SPARK_Mode => On`

- Ada types, expressions, statements, subprograms

`SPARK_Mode => Off`

- Ada pointers
- Ada exception handlers

- Ada generics
- Ada object orientation
- Ada concurrency
- Ada pointers



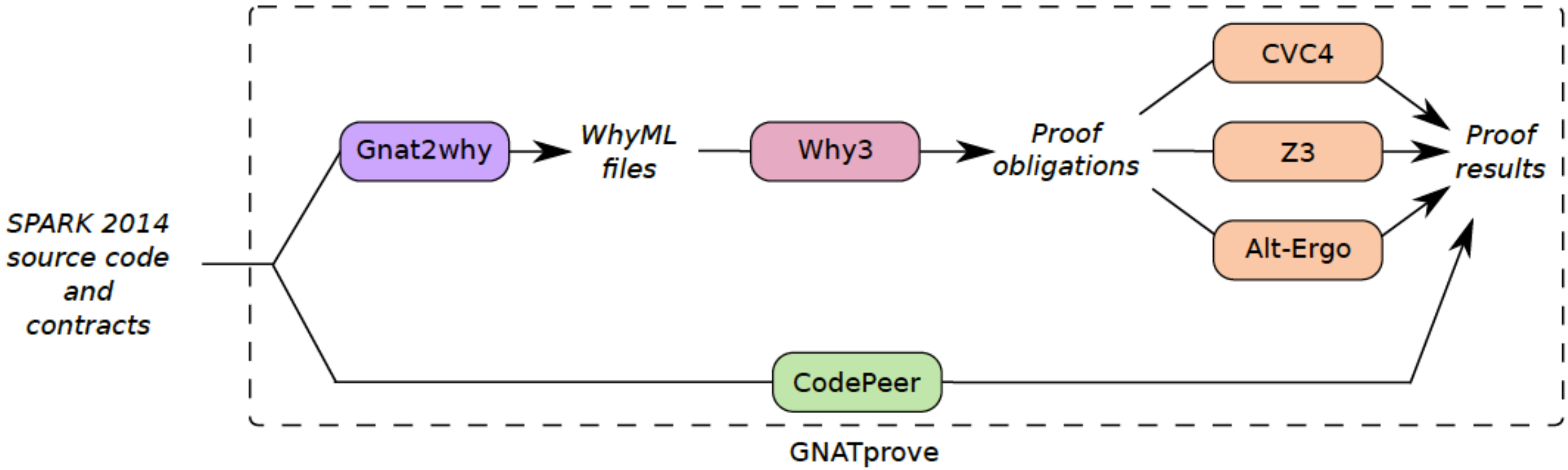
work in progress to include safe Rust-like pointers in SPARK

Bronze/Silver Level – Generation of Contracts

Example: SPARKSkein Skein cryptographic hash algorithm (Chapman, 2011)
target: Silver level

initial version (SPARK 2005)	current version (SPARK 2014)
41 non-trivial contracts for effects and dependencies	1 – effects and dependencies are generated
31 conditions in preconditions and postconditions on internal subprograms	0 – internal subprograms are inlined
43 conditions in loop invariants	1 – loop frame conditions are generated
23 annotations to prevent combinatorial explosion	0 – no combinatorial explosion

Silver/Gold Level – Combination of Provers



Silver/Gold Level – Combination of Provers

Example: Safe bounds on trajectory computation (submitted to VSTTE 2017)
target: Gold level

```
procedure Compute_Speed (N         : Frame;
                         Factor    : Ratio_T;
                         Old_Speed : Float64;
                         New_Speed : out Float64)
with Global => null,
     Pre    => N < Frame'Last and then
               Invariant (N, Old_Speed),
     Post   => Invariant (N + 1, New_Speed);
```

```
Delta_Speed := Drag + Factor * G * Frame_Length;
New_Speed   := Old_Speed + Delta_Speed;
```

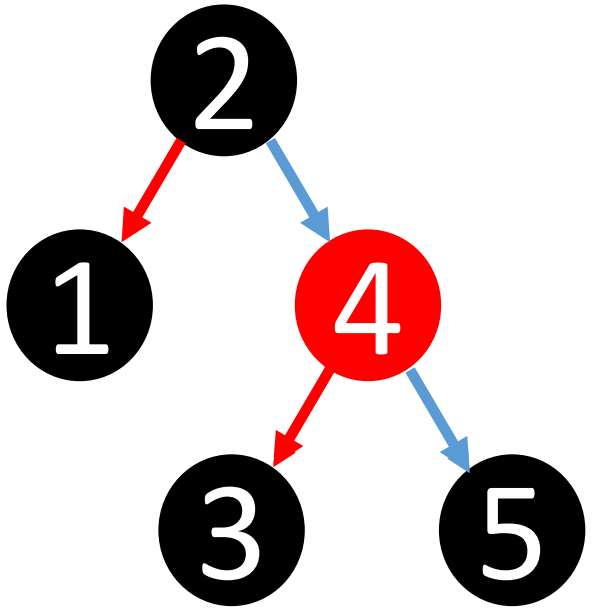
VC	CVC4	Alt-Ergo	Z3	CodePeer	AE_fpa	Colibri
Delta_Speed in -Bound .. Bound				1	3	0
In_Bounds (High_Bound(N))				1	1	
In_Bounds (Low_Bound(N))			0	1	2	
Float64(N_Bv) * Bound + Bound = (Float64(N_Bv) + 1.0) * Bound			42			0
Float64(N) * Bound + Bound = (Float64(N) + 1.0) * Bound		44		1	25	0
Float64(N) * (-Bound) Bound = (Float64(N) + 1.0) * (-Bound)				1		0
T(1) = 1.0	0	0		1	0	0
Float64(N) + 1.0 = Float64(N + 1)	0	1			1	0
New_Speed >= Float64 (N) * (-Bound) Bound	27					0
New_Speed >= Float64 (N + 1) * (-Bound)			1			0
New_Speed <= Float64 (N) * Bound + Bound	26					0
New_Speed <= Float64 (N + 1) * Bound			1			0
Post-condition	20	0			1	

Gold/Platinum Level – Auto-Active Verification

Example: Functional correctness of red-black trees (NFM 2017)
target: Platinum level

Auto-Active = portmanteau of **Auto**matic and inter**Active**

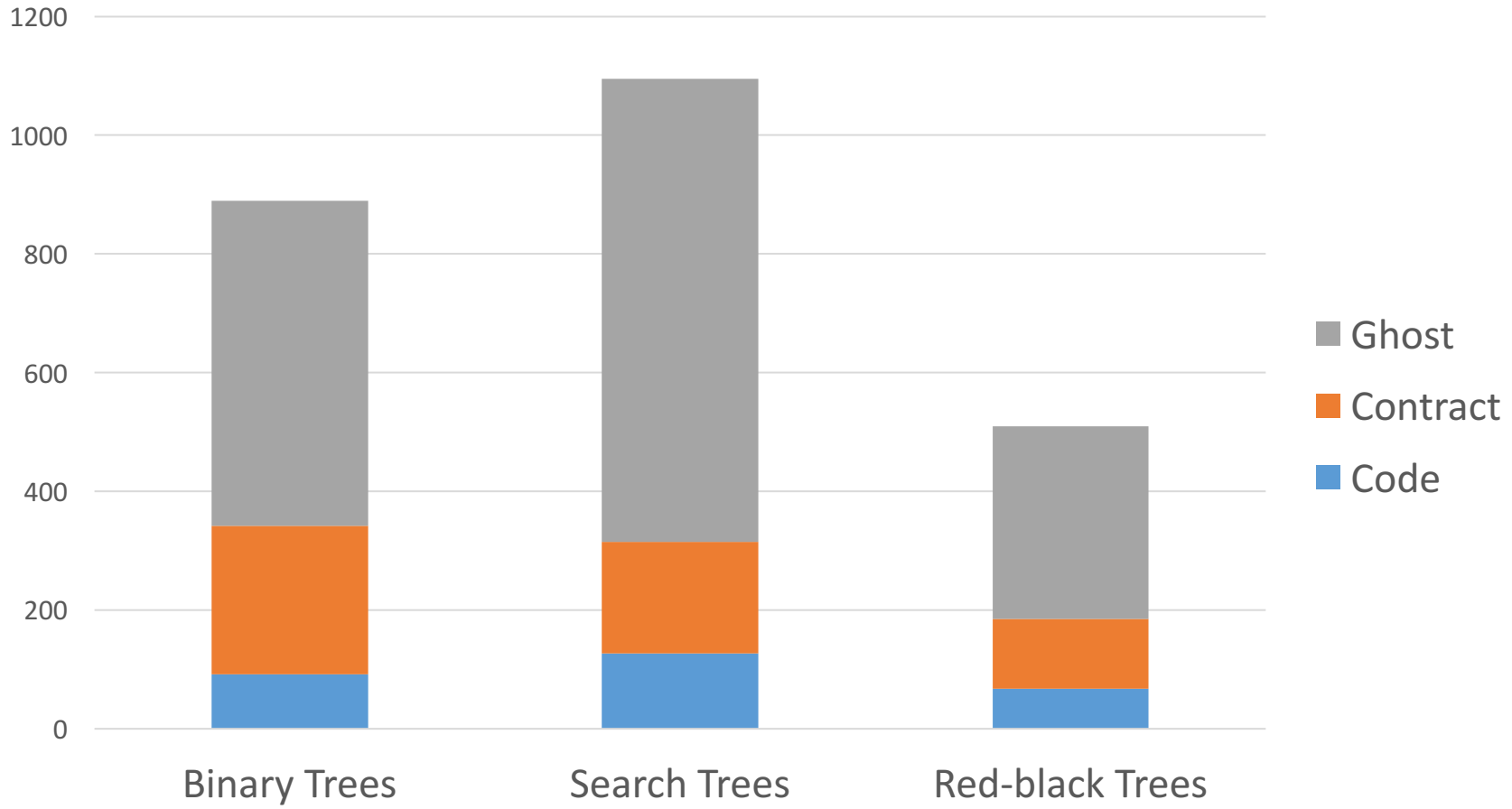
supported by **ghost** code: contracts, loop invariants, intermediate assertions, lemma procedures



ghost code used to:

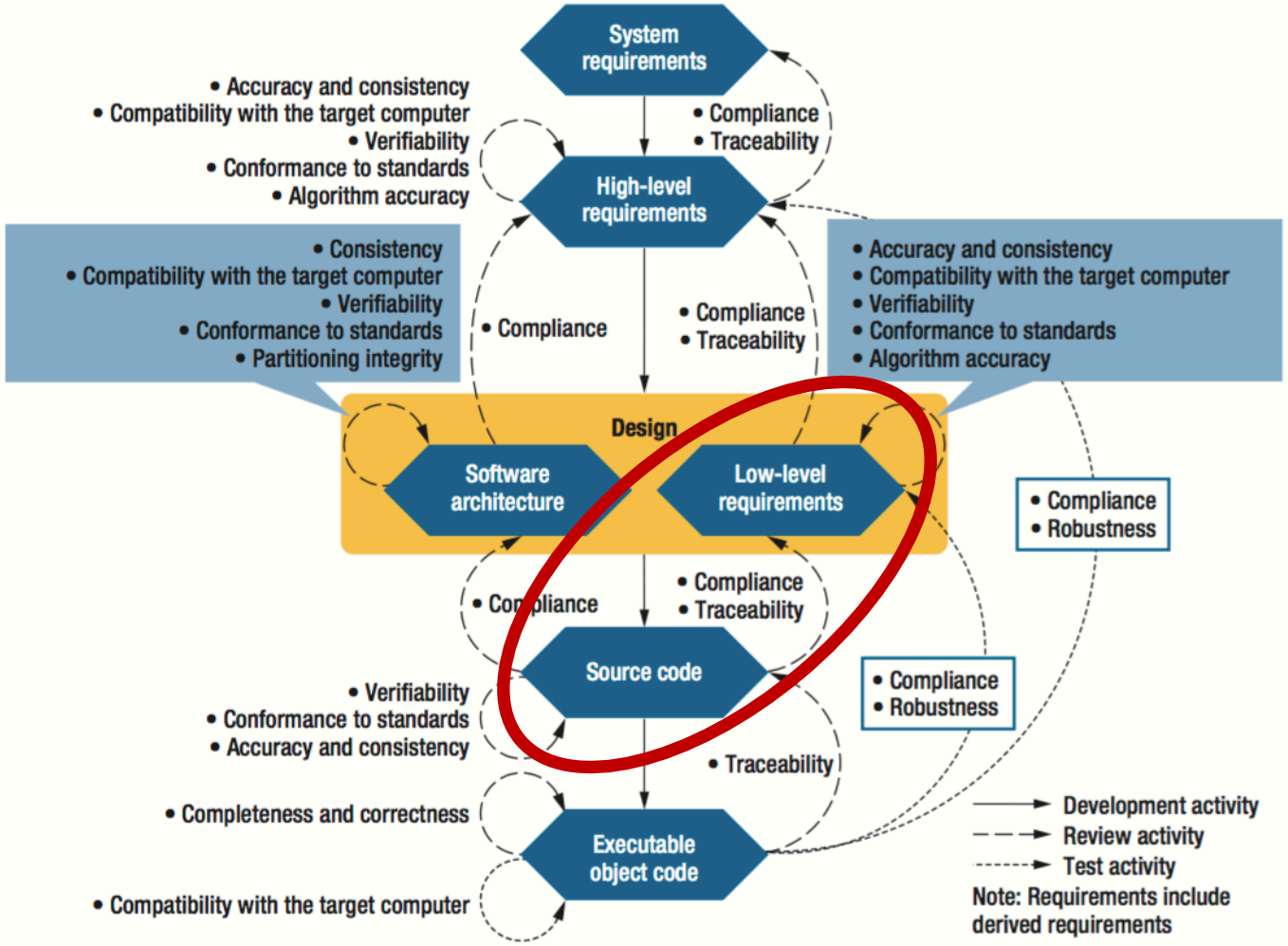
- define model of data used in specifications
- prove intermediate lemmas (e.g. for inductive proofs)
- provide witness for property (e.g. for transitivity relation)

Gold/Platinum Level – Auto-Active Verification



Use in Certification

Avionics – DO-178C and DO-333

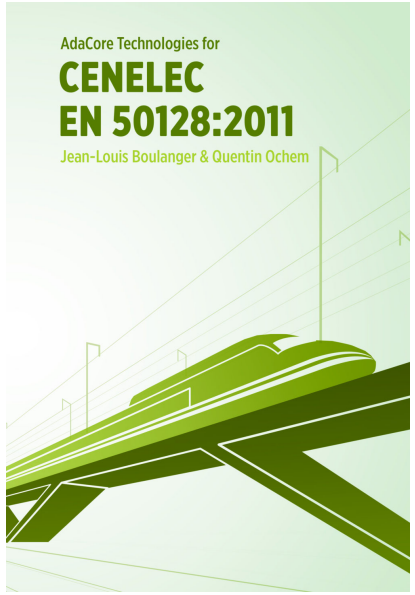


AdaCore Technologies for DO-178C / ED-12C

https://www.adacore.com/books/do-178c-tech

Railway – EN 50128

- D.2 Analyzable Programs
- D.4 Boundary Value Analysis
- D.10 Data Flow Analysis
- D.14 Defensive Programming
- D.18 Input Partition Testing
- D.24 Failure Assertion Programming
- D.28 Formal Methods
- D.34 Interface Testing
- D.35 Language Subset
- D.38 Modular Approach
- D.49 Strong Typing
- D.53 Structured Programming
- D.54 Suitable Programing Languages



AdaCore Technologies
for CENELEC EN 50128:2011

https://www.adacore.com/books/cenelec-en-50128-2011

Automotive – ISO 26262

- Table 1 — Topics to be covered by modelling and coding guidelines
- Table 2 — Notations for software architectural design
- Table 6 — Methods for the verification of the software architectural design
- Table 9 — Methods for the verification of software unit design and implementation
- Table 11 — Methods for deriving test cases for software unit testing
- Table 13 — Methods for software integration testing
- Table 14 — Methods for deriving test cases for software integration testing

working on a booklet
AdaCore Technologies
for ISO 26262

Conclusion

Levels of Software Assurance

From strong semantic coding standard to full functional correctness

Every level implicitly builds on the lower levels

Lower levels require lower costs/efforts

Good match from DAL/SIL to Bronze-Silver-Gold-Platinum

Good match to certification objectives in transport

Adoption greatly facilitated by detailed level-specific guidance

SPARK Resources

SPARK toolset

http://www.adacore.com/sparkpro http:// www.adacore.com/community

SPARK adoption guidance

www.adacore.com/knowledge/technical-papers/implementation-guidance-spark

SPARK blog and resources (User's Guide)

http://www.spark-2014.org

SPARK online training

http://u.adacore.com