FROM RESEARCH TO INDUSTRY

cea tech

FORMAL VERIFICATION OF EMBEDDED SOFTWARE

THROUGH FRAMA-C

list

Julien Signoles
Software Security Labs
CEA LIST

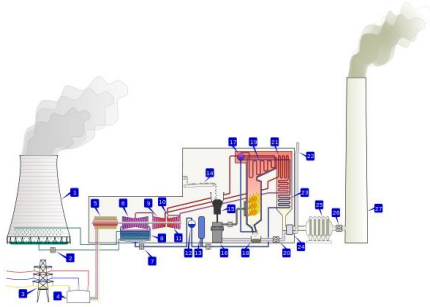
OUTLINE

- **Formal verification of embedded software through Frama-C**
  - Why do we trust (our) software?
  - What is Frama-C?
  - Some industrial usage
- **Frama-C/Value**
- **Frama-C/WP**

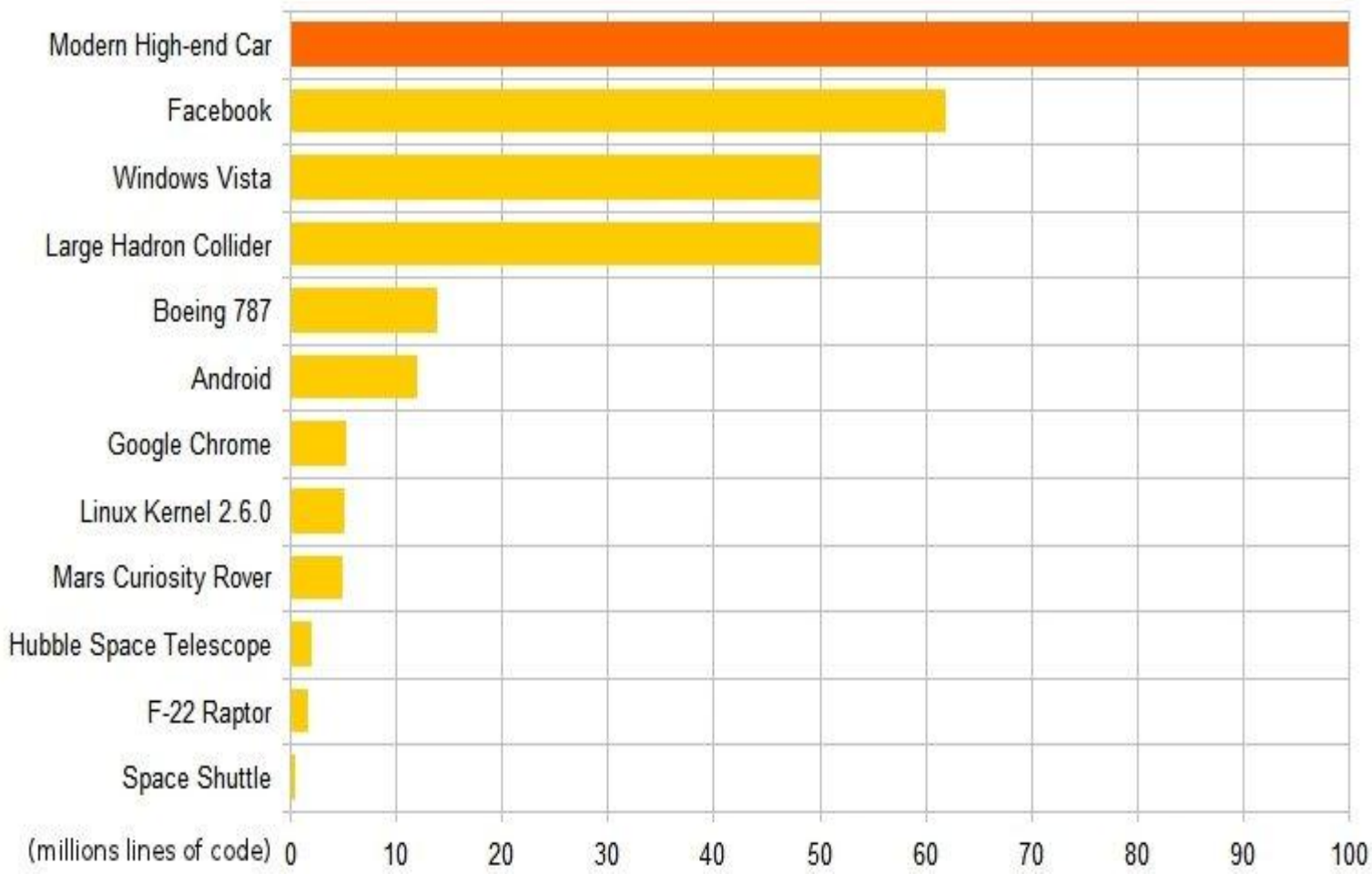
TRUST

WHY DO WE (OUR) SOFTWARE?

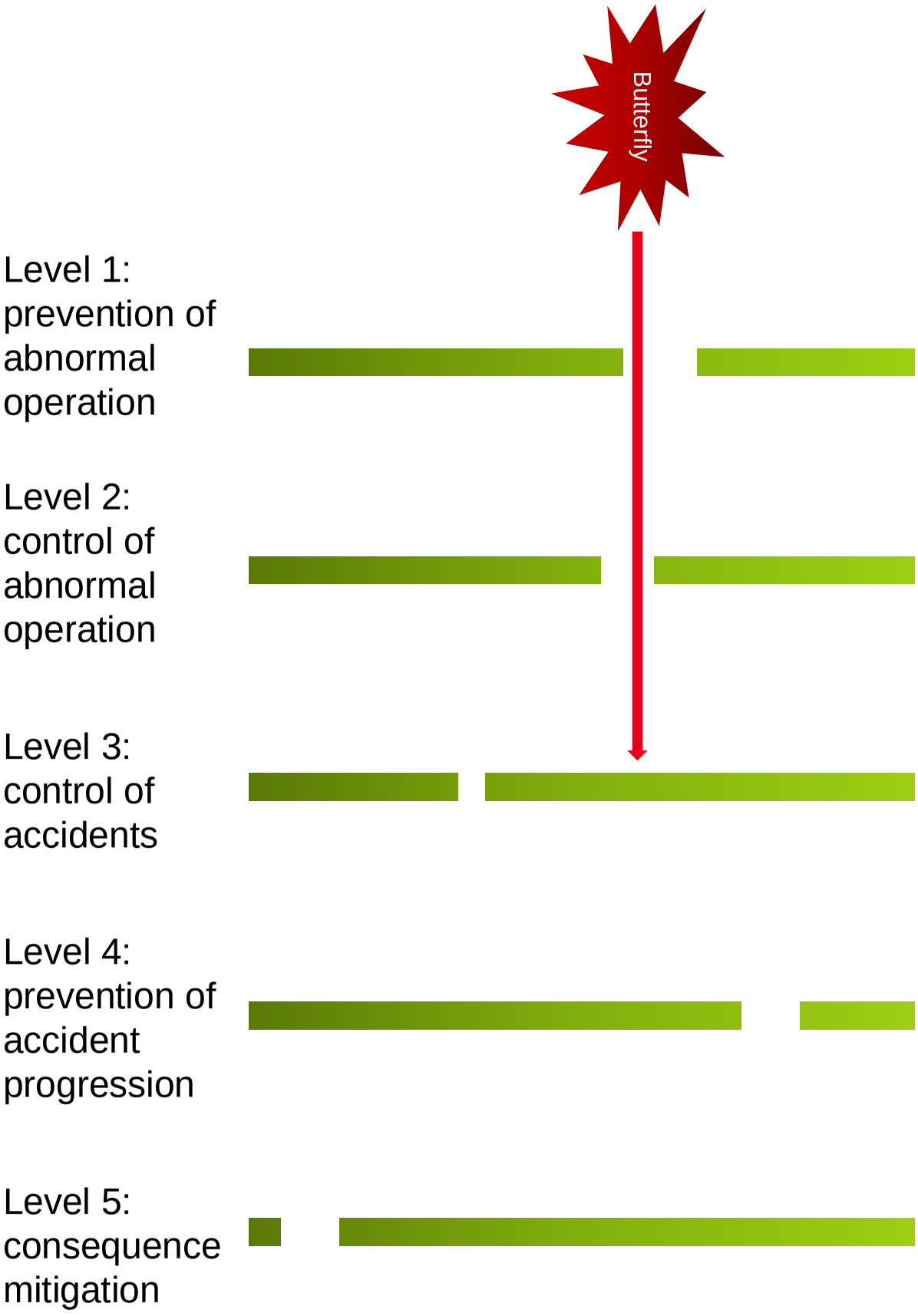
- ✓ Software systems are becoming larger and more complex
- ✓ Life-critical software cannot be permitted to exhibit unexpected behaviors or security vulnerabilities
- ✓ Stringent certification standards require strong guarantees
  - ✓ DO-178B/C for aerospace safety
  - ✓ EN 50128 for rail safety
  - ✓ ISO 26262 for road vehicles (since 2011)
  - ✓ IEC 62304 for medical equipment
  - ✓ CENELEC 61508 reference
  - ✓ Common Criteria EAL7 for security

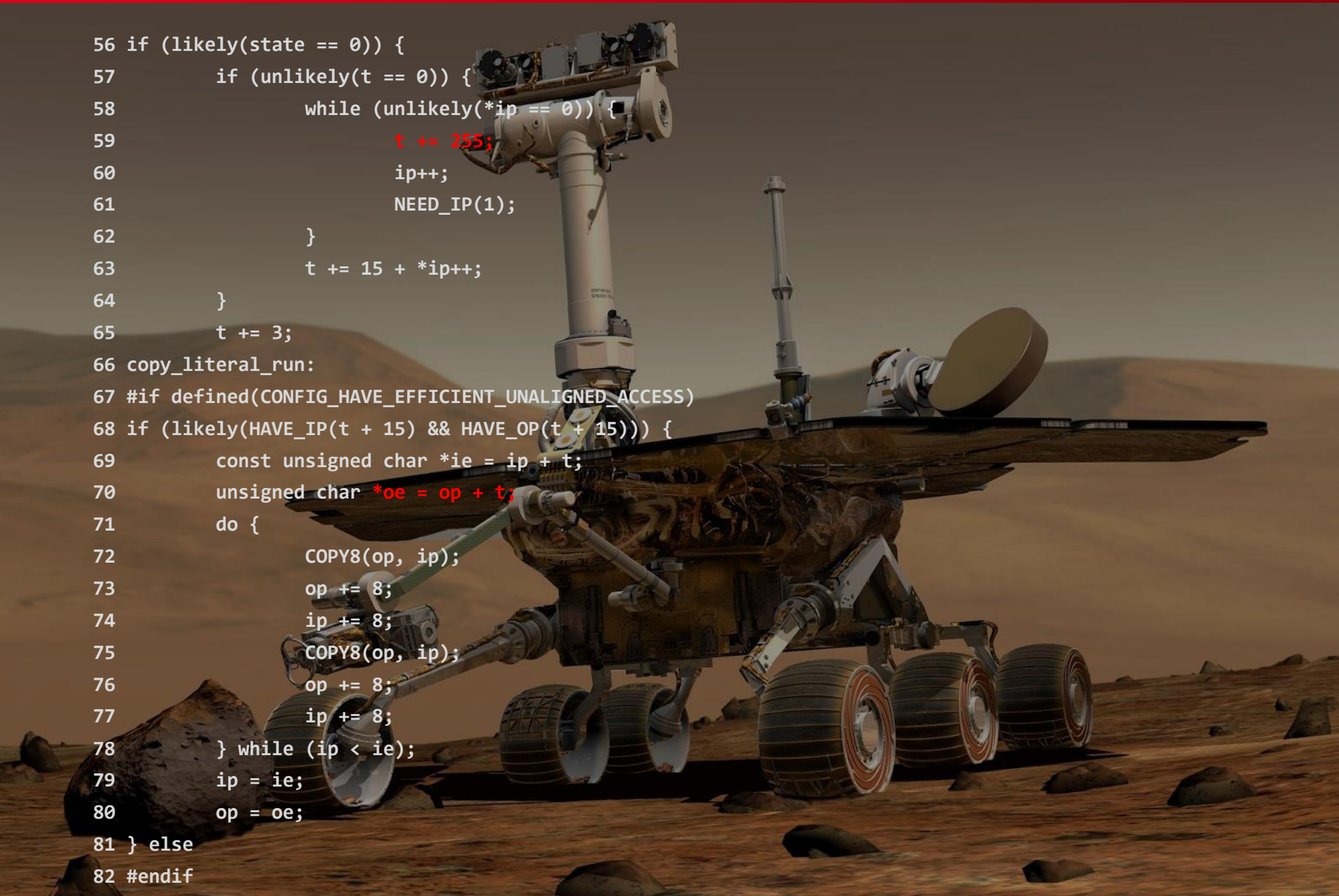


26262
ISO
Common Criteria

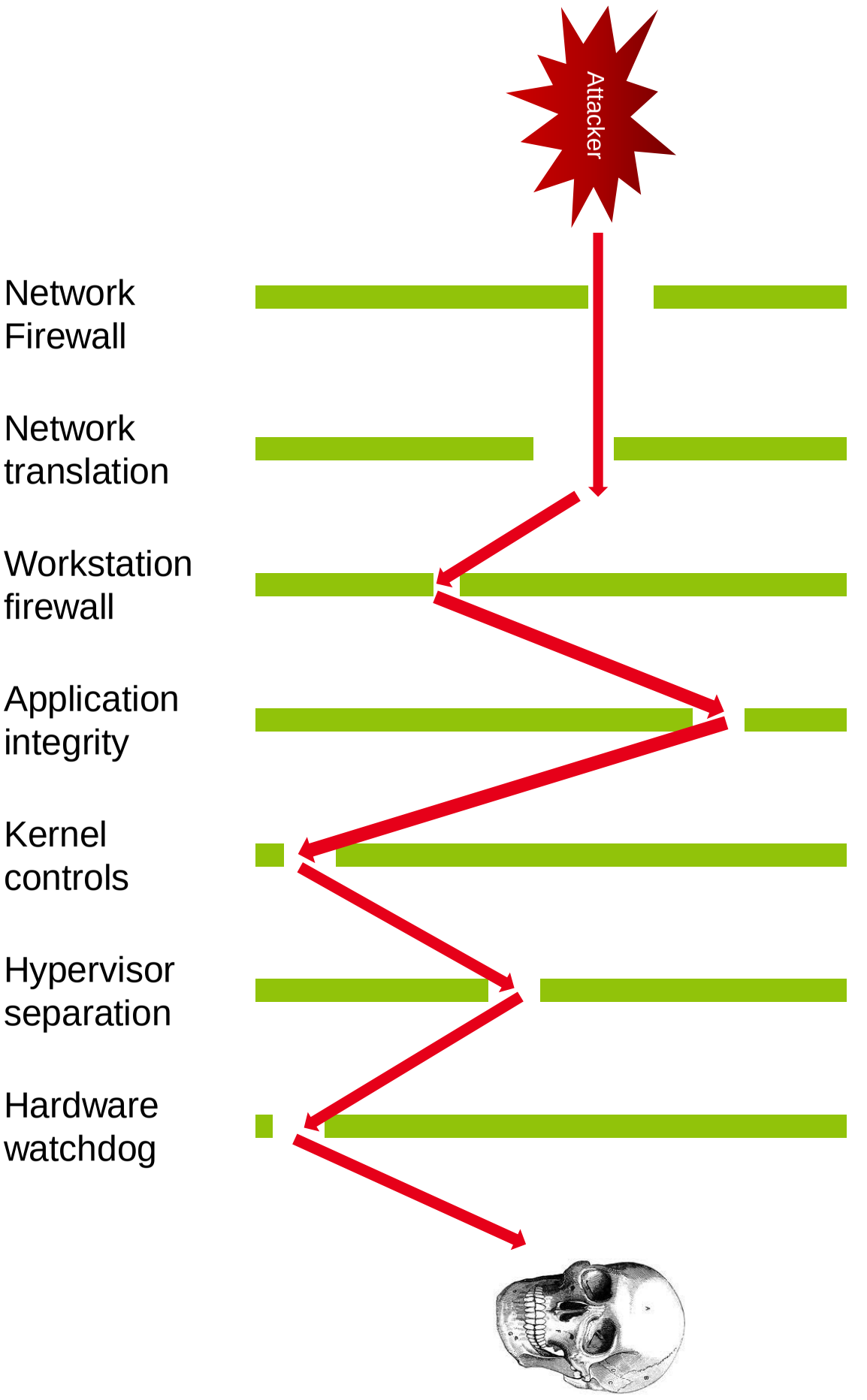


the number of errors partly depends on the software size…


Butterfly
Level 1: prevention of abnormal operation
Level 2: control of abnormal operation
Level 3: control of accidents
Level 4: prevention of accident progression
Level 5: consequence mitigation



```
56 if (likely(state == 0)) {
57         if (unlikely(t == 0)) {
58                 while (unlikely(*ip == 0)) {
59                         t += 255;
60                         ip++;
61                         NEED_IP(1);
62                 }
63                 t += 15 + *ip++;
64         }
65         t += 3;
66 copy_literal_run:
67 #if defined(CONFIG_HAVE_EFFICIENT_UNALIGNED_ACCESS)
68 if (likely(HAVE_IP(t + 15) && HAVE_OP(t + 15))) {
69         const unsigned char *ie = ip + t;
70         unsigned char *oe = op + t;
71         do {
72                 COPY8(op, ip);
73                 op += 8;
74                 ip += 8;
75                 COPY8(op, ip);
76                 op += 8;
77                 ip += 8;
78         } while (ip < ie);
79         ip = ie;
80         op = oe;
81 } else
82 #endif
```


Attacker
Network Firewall
Network translation
Workstation firewall
Application integrity
Kernel controls
Hypervisor separation
Hardware watchdog

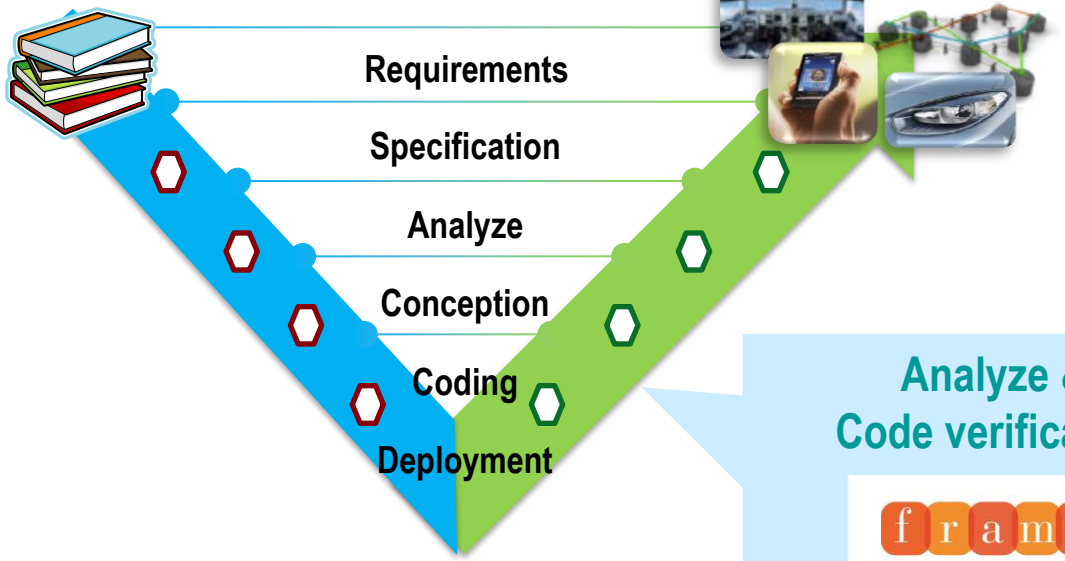
```
2552 #ifndef OPENSSL_NO_HEARTBEATS
2553 int
2554 tls1_process_heartbeat(SSL *s)
2555         {
2556         unsigned char *p = &s->s3->rrec.data[0], *pl;
[…]
2561         /* Read type and payload length first */
2562         hbtype = *p++;
2563         n2s(p, payload);
2564         pl = p;
[…]
2571         if (hbtype == TLS1_HB_REQUEST)
2572                 {
[…]
2583                 /* Enter response type, length and copy payload */
2584                 *bp++ = TLS1_HB_RESPONSE;
2585                 s2n(payload, bp);
2586                 memcpy(bp, pl, payload);
2587                 bp += payload;
2588                 /* Random padding */
2589                 RAND_pseudo_bytes(bp, padding);
2590
2591                 r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer,
3 + payload + padding);
2592
2593                 if (r >= 0 && s->msg_callback)
2594                         s->msg_callback(1, s->version,
TLS1_RT_HEARTBEAT,
```

TOOL

FOR HIGH CONFIDENCE

SOFTWARE & SYSTEMS

V&V on the second phase of the cycle



Transfert industriel TRL 4-5-6-7

Maturation TRL 3-4-5

Ressourcement TRL 1-2-3

Analyze & Code verification

frama C Software Analyzers

open source

http://frama-c.com

HOW DO WE TRUST A SOFTWARE?

- **Provide guarantees that it has « no error »**
  - It never crashes
  - It behaves as expected
- **What is the expected behavior?**
  - Specification
- **How to ensure that the runtime behavior is the expected one?**
  - Full compliance of the software wrt its specification
- **Is it possible? At what cost?**

WHAT WOULD BE A PERFERT TOOL?

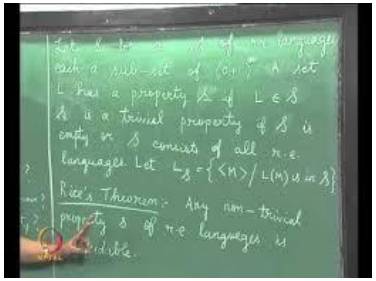
What perfect tool T in order to guarantee that a program P satisfies a specification S?

1. **provide P and S as input to T**
2. **press « enter »: automatic analysis**
3. **T looks at S and T, without executing them: static analysis**
4. **T instantaneously provides an answer**
   - ✓ yes, P satisfies S
   - ✓ no, it doesn't : there is an issue at this program point.
5. **be sure that the tool answer is correct**

Unfortunately, perfection does not exist…

RICE THEOREM (1953)

An automatic exact static analysis is not possible



Any property which is

- extensional (which depends on the program semantics and not on the syntax)
- not trivial (neither always true nor false)

is

- undecidable (any algorithm which decides whether such a property is valid either loops forever or is wrong for an infinity of input programs)

PRAGMATIC SOLUTIONS TO A THEORETICAL ISSUE

Relax (at least) one constraint

- **Test: execute the program in order to find bugs**
  - Most standard industrial practice
  - Not exhaustive: cannot prove the absence of errors
- **Model checking (Clarke & Emerson 81, Queille & Sifakis 82)**
  - Verify a model of the program
  - Compliance of the program wrt its model?

- **Deductive methods (Dijkstra 76)**
  - Mathematical reasoning about the program
  - Not fully automatic

- **Abstract interpretation (Cousot & Cousot 77)**
  - Correct approximation of the program behavior
  - May answer « I don’t know »

FRAMA-C

- developed by **CEA LIST** since 2005 (in collaboration with Inria)
- **platform for the analysis of source code written in C** (ISO C 99)

- Open source (LGPL v2.1)
- Close source extensions for industrial usage

- **several analyzers** inside
- **extensible and collaborative** setting

- both for academic and industrial purposes

Several tools inside a single platform

http://frama-c.com

FRAMA-C: EXTENSIBLE AND COLLABORATIVE PLATFORM

- **plug-in based architecture** *à la* Eclipse
- each analyzer is a plug-in
- each plug-in is connected to the Frama-C **kernel**
  - the kernel provides general services
  - the kernel synthesizes information
- **analyzers may collaborate each other**
  - use results of others analysis
  - exchange formal ACSL properties to be verified
  - **ACSL** : *langua franca* between analyzers

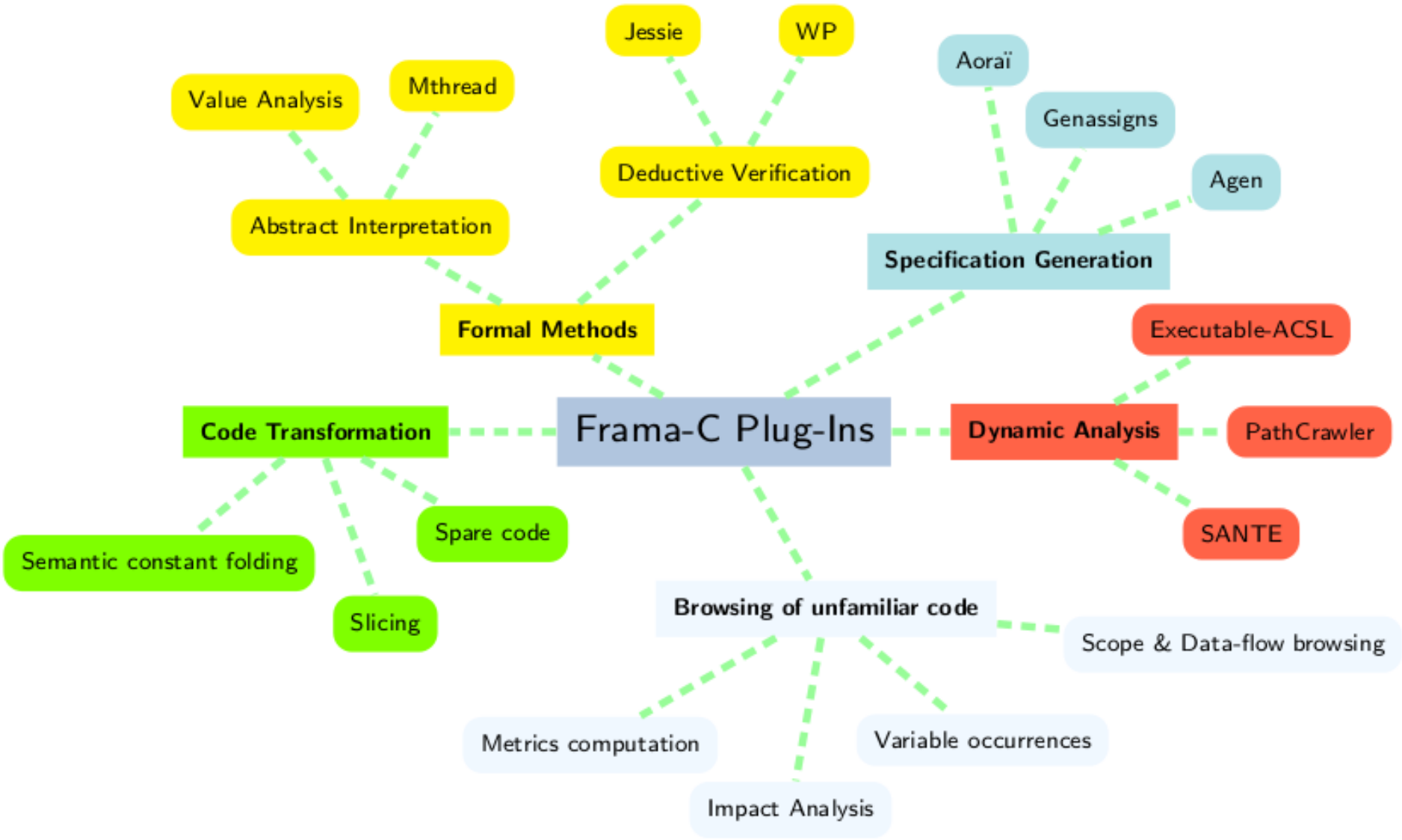
FRAMA-C: DEVELOPMENT PLATFORM

- developed in **OCaml** (>> 100 kloc)
- based on **Cil** (C Intermediate Langage, Berkeley)

- **library dedicated to code analysis of C code**

Development of plug-ins by third party

- **powerful low-cost analyzer**
- **extension** of existing analyzers
- experimentation platform
- dedicated plug-ins for **specific tasks** (e.g. verifying coding rules)


Jessie
WP
Aoraï
Value Analysis
Mthread
Genassigns
Deductive Verification
Agen
Abstract Interpretation
Specification Generation
Formal Methods
Executable-ACSL
Code Transformation
Frama-C Plug-Ins
Dynamic Analysis
PathCrawler
Spare code
Semantic constant folding
SANTE
Slicing
Browsing of unfamiliar code
Scope & Data-flow browsing
Metrics computation
Variable occurrences
Impact Analysis

- properties are formalized using unequivocal specifications
- they can be hand-written
- they can be automatically generated
- they can be automatically verified


```
/*@ requires -1000 <= x <= 1000;
    ensures \result >= 0;
    */
```

```
int abs(int x)
{
  int r;
  if (x >= 0)
    r = x;
  else
    r = - x;
  return r;
}
```

```
int abs(int x)
{
  int r;
  if (x >= 0)
    r = x;
  else
    r = - x;
  return r;
}
```

INDUSTRIAL EXAMPLES OF USAGES

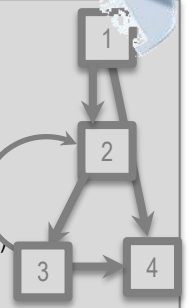
HIGH-QUALITY SOFTWARE AND SYSTEMS

le Analysis :

while (x>=1000) {
x= x-a;
}

: init(x)
: (1000 ≤ x) ∧ (P1 ∨ P3)
P3 : P2[x+a]
P4 : (x<1000) ∧ (P1 ∨ P3)

Research of pre-conditions for infinite loops
(Threat example) ➔ x ≥ 1000 ∧ a ≤ 0



SAFETY
IEC60880

SAFETY
DO-178C

ANALYZING INTRINSIC FAULTS | SCADASYSTEMS

Can we guarantee the absence of defaults in large system-level code?

- **Pinpoint all runtime errors and help investigate their cause**
- **Structural properties on memory separation and cyclic behaviors.**
- **Using collaborative, automated analyses.**
- **Justification and certification to IEC60880 Class 1**

```
> 100+ kloc
> C source code
> Highest
      certification
      requirements

> 80% code coverage
> 200 alarms
```

VERIFYING PROPERTIES | #AVIONICS

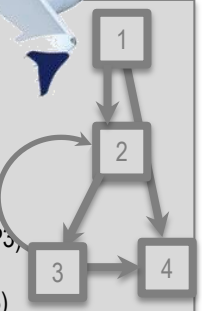
How do we prove that embedded software properties are satisfied?

- **Automate unit proofs on critical code**
- **Using C code interactive verification**
- **A380, A350 & A400M programs**
- **Qualification and certification to DO178B/C level A**

Code Analysis :

```
1 : while (x>=1000) {
2 :   x= x-a;
3 :   }
4 :
```

P1 : init(x)
P2 : (1000 ≤ x) ∧ (P1 ∨ P3)
P3 : P2[x+a]
P4 : (x<1000) ∧ (P1 ∨ P3)



Research of pre-conditions for infinite loops (Threat example) ➔ $x \geq 1000 \wedge a \leq 0$

SAFETY

VERIFYING PROPERTIES | #AVIONICS

- **« Source code matches the data flow and control flow defined in the software architecture »**
  DO178B §6.3.4.b

- **Traditionally performed by code review**

- **Automation objectives**
  - correction (pointers, data alignment)
  - ease of use (push-button)

VERIFYING PROPERTIES | #AVIONICS

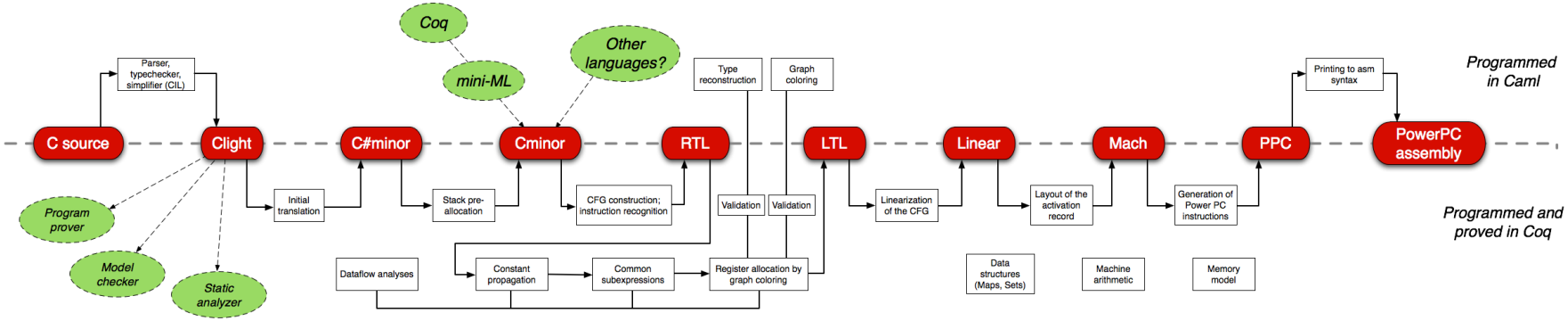
Integration Plugin	Generated ACSL, launch analysis, get results, consolidate and export in reports	Developed by industrials: Airbus SA (and Atos)
Inout, from analysis	Analysis of read/written locations	Standard framework
Semantic analysis	Pointers resolution	
Frama-C kernel	C compilation, Uniform and consistent framework	

FORMAL COMPILER VERIFICATION | #DRAGONS

AIRBUS
AN EADS COMPANY

Can we formally design and implement an optimizing C language compiler?

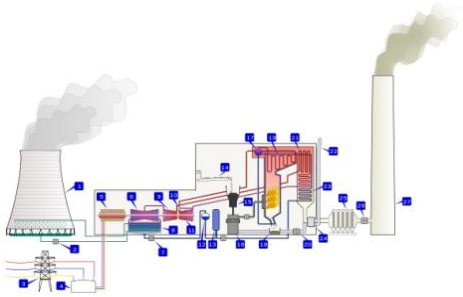
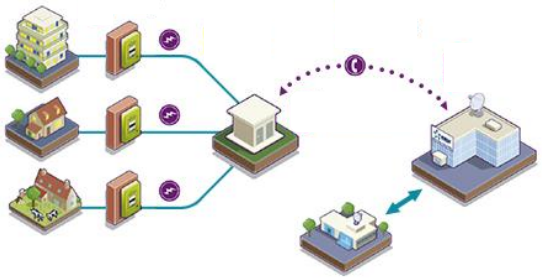
- **Compcert compiler designed and implemented in formal higher-order logics with Coq**
- **MISRA-C 2004, recursive functions, dynamic memory allocation, PPC/ARM/IA32 targets**
- **Justification and certification to DO-178C Level A**



FROM SAFETY

TO SECURITY

Software-intensive systems are spreading widely


SOCIETAL ACCEPTANCE DEMANDS 100% SECURITY GUARANTEES

Formal modeling & validation of software assets

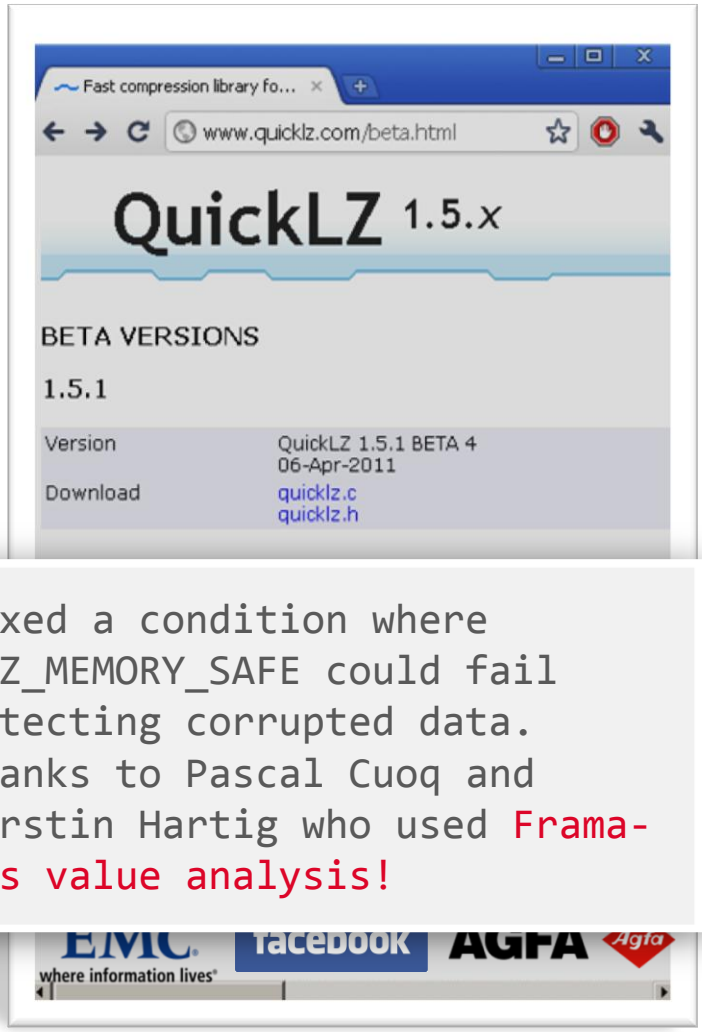
- Frama-C: verify *functional security*, check *data-flow boundaries*, guarantee the absence of *runtime vulnerabilities,* generate *test cases*…

```
2552 #ifndef OPENSSL_NO_HEARTBEATS
2553 int
2554 tls1_process_heartbeat(SSL *s)
2555         {
[…]
2561         /* Read type and payload length first */
2562         hbtype = *p++;
2563         n2s(p, payload);
2564         pl = p;
[…]
2571         if (hbtype == TLS1_HB_REQUEST)
2572                 {
[…]
2583                 /* Enter response type, length and copy payload */
2584                 *bp++ = TLS1_HB_RESPONSE;
2585                 s2n(payload, bp);
2586                 memcpy(bp, pl, payload);
2587                 bp += payload;
2588                 /* Random padding */
2589                 RAND_pseudo_bytes(bp, padding);
2590
2591                 r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer,
3 + payload + padding);
2592
2593                 if (r >= 0 && s->msg_callback)
2594                         s->msg_callback(1, s->version,
TLS1_RT_HEARTBEAT,
2595                                 buffer, 3 + payload + padding,
```

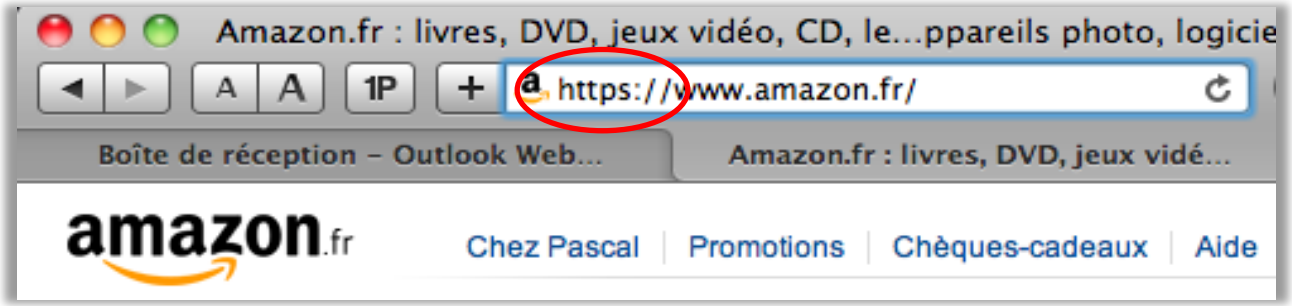
DETECTING SECURITY FLAWS | COTSCOMPRESSIONLIBRARY

How do we reach the sophisticated "last vulnerabilities"?

- **Detect all occurrences of a given category of vulnerabilities.**
- **Using automated code analyses**
- **Handling general-purpose code constructs**
- **Ongoing analysis of the normative context**



VERIFICATION OF POLAR SSL | COTSCRYPTOLIBRARY



POLAR SSL

TRUST IN SOFT

TOOL EXPOSITION & EVALUATION

NIST National Institute of Standards and Technology U.S. Department of Commerce

TRUST IN SOFT

ADIT MINISTÈRE DES AFFAIRES ÉTRANGÈRES Liberté • Égalité • Fraternité RÉPUBLIQUE FRANÇAISE

Are software analysis tools as exhaustive as they claim?

- **NIST's Ockham Criteria provides "known good" and "known bad" code**
- **Evaluated on the Juliet Test Suite – 27000 testcases**
- **Only tool able to satisfy the criteria in 2014**
- **Several bugs found in the Juliet benchmark**

```
<weakness id="2958">
<name>invalid memory access</name>
<location
path="synthetic/testcases/CWE126_Buffer_Over
read/s02/CWE126_Buffer_Overread__malloc_wcha
r_t_loop_03.c" line="70">
</location>
<grade severity="4"/>
<output><textoutput><![CDATA[../ppc/share/li
bc/wchar.c:32:[kernel] warning: out
of bounds write. assert \valid(tmp);
                  stack: wmemset ::
testcases/CWE126_Buffer_Overread/s02/CWE126_
Buffer_Overread__malloc_wchar_t_loop_03.c:70
<-
                           goodG2B1 ::
testcases/CWE126_Buffer_Overread/s02/CWE126_
Buffer_Overread__malloc_wchar_t_loop_03.c:12
3 <-

CWE126_Buffer_Overread__malloc_wchar_t_loop_
03_good
]]></textoutput></output></weakness>
```

TRUST IN SOFT

Frama-C for software safety and security

www.trust-in-soft.com

SOFTWARE DEVELOPERS

- Industrial support
- Commercial licenses
- Preinstalled workstations

SOFTWARE INTEGRATORS

Off-the-shelf validation kits for common open-source packages

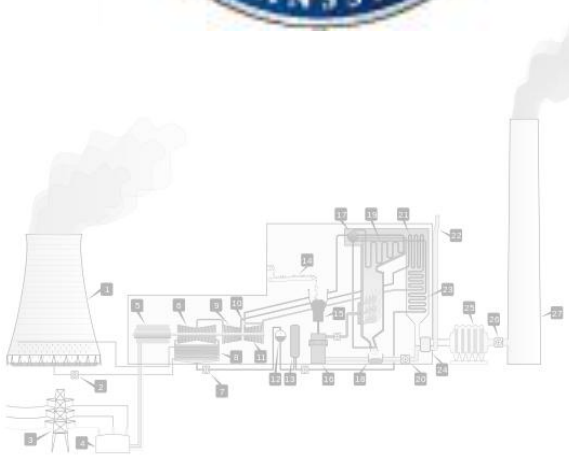
SERVICE PROVIDERS

Dedicated affiliate programs

CYBERSECURITY STANDARDISATION

What is the state of the art in software analysis?

- **Guarantee the absence of runtime errors such as buffer overflows or invalid pointers**
- **Verify that an implementation adheres to a formal security model**
- **Impact analysis, monitor insertion, etc.**



NEXT

DECADE

TECHNOLOGIES

RUNTIME MONITORING AND VERIFICATION

Is it possible to toughen runtime checks in dynamic and compositional systems?

- **Automate monitor generation from formal specifications**
- **Benefit from static analyses to infer specifications and minimize overhead**
- **Use in conjunction with hardware enablers**

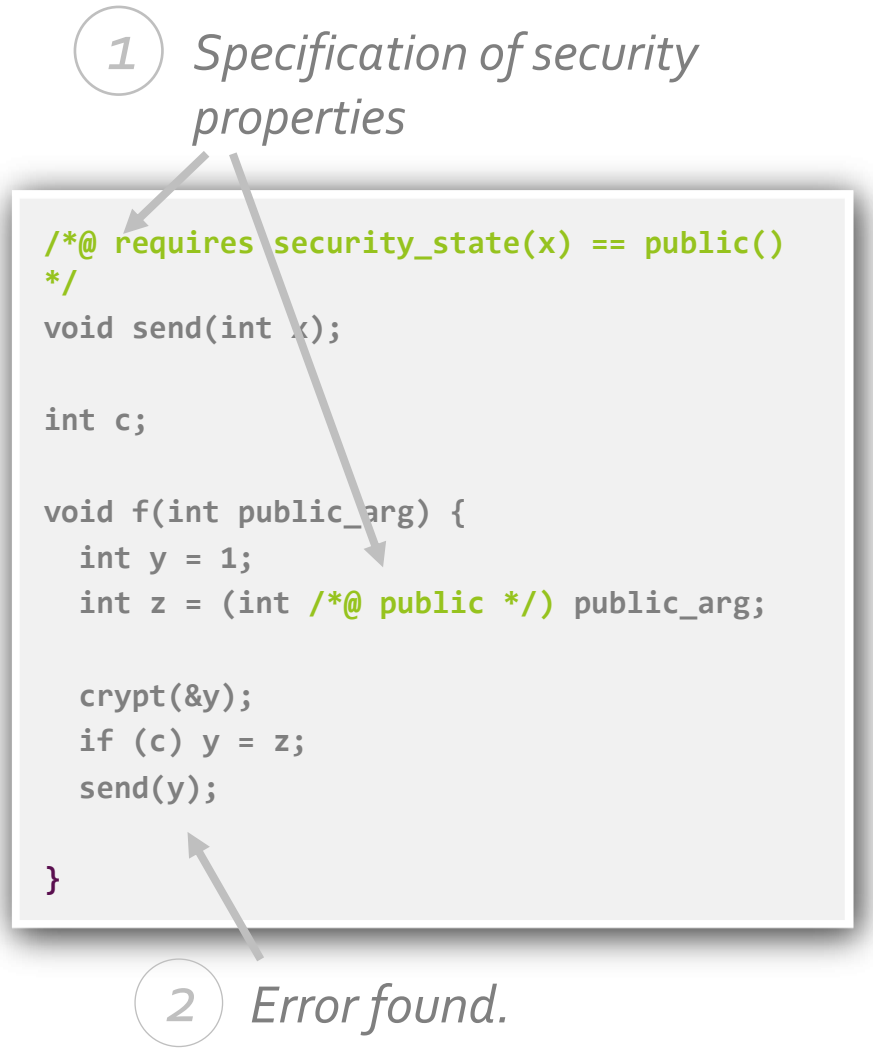
```
int div (int x, int y) {
  // make sure *p != 0;
  return x / *p;
}
```

```
int div (int x, int y) {
  if (!is_valid(p)
      || *p==0)
    e_acsl_fail();
  return x / *p;
}
```

ADVANCED VALIDATION OF A SET OF INFORMATION FLOW PROPERTIES

Can we build on existing analysis tools to specify and verify custom privacy properties?

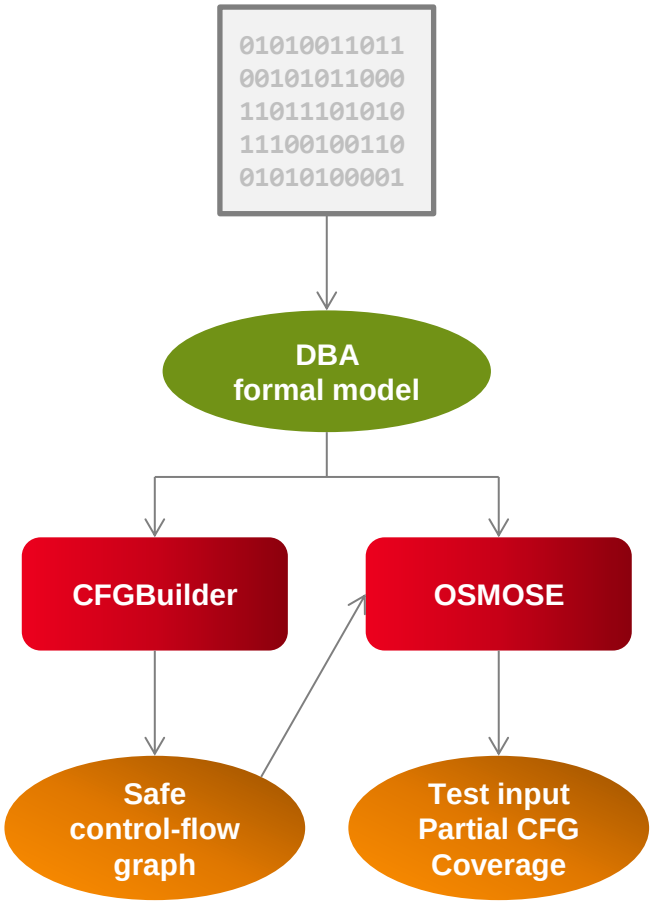
- **Specification and verification of non-interference properties**
- **Mixed static / dynamic techniques**
- **Quantitative information flow analysis at TRL2**



MACHINE CODE SECURITY ANALYSIS

How much can we check when source code isn't an option?

- Intermediate language for machine code analysis (DBA)
- Safe control-flow graph reconstruction based on advanced refinement techniques
- Test data generation through dynamic symbolic execution




SAFRAN
Sagem
SEARCH-LAB
SECURITY EVALUATION ANALYSIS AND RESEARCH LABORATORY
UNIVERSITY OF CALIFORNIA
SANTA CRUZ
NASA
galois
Fraunhofer
ALSTOM
SRI International
cnrs
IBM
DASSAULT
AVIATION
BULL
Inria
AIRBUS
AN EADS COMPANY
CS
OPPIDA
EXPERT EN SÉCURITÉ DES SYSTÈMES D'INFORMATION
THALES
AMOSSYS
eDF
EADS
AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION
ANSSI
AREVA
IRSN
Schneider Electric
TRUST IN SOFT
Rolls-Royce
ESTEREL Technologies
Atos
Trusted Labs
RENAULT
e-rsb
intel
Adelard
SIEMENS
Valeo
ONERA
SHERPA

Software Security Laboratory
Software & Systems Engineering Department
CEA LIST

Julien Signoles
julien.signoles@cea.fr