Development and Formal Verification of a Flight Stack for a High-Altitude Micro Glider

Emanuel Regnath

Toulouse, 10.10.2017

Motivation

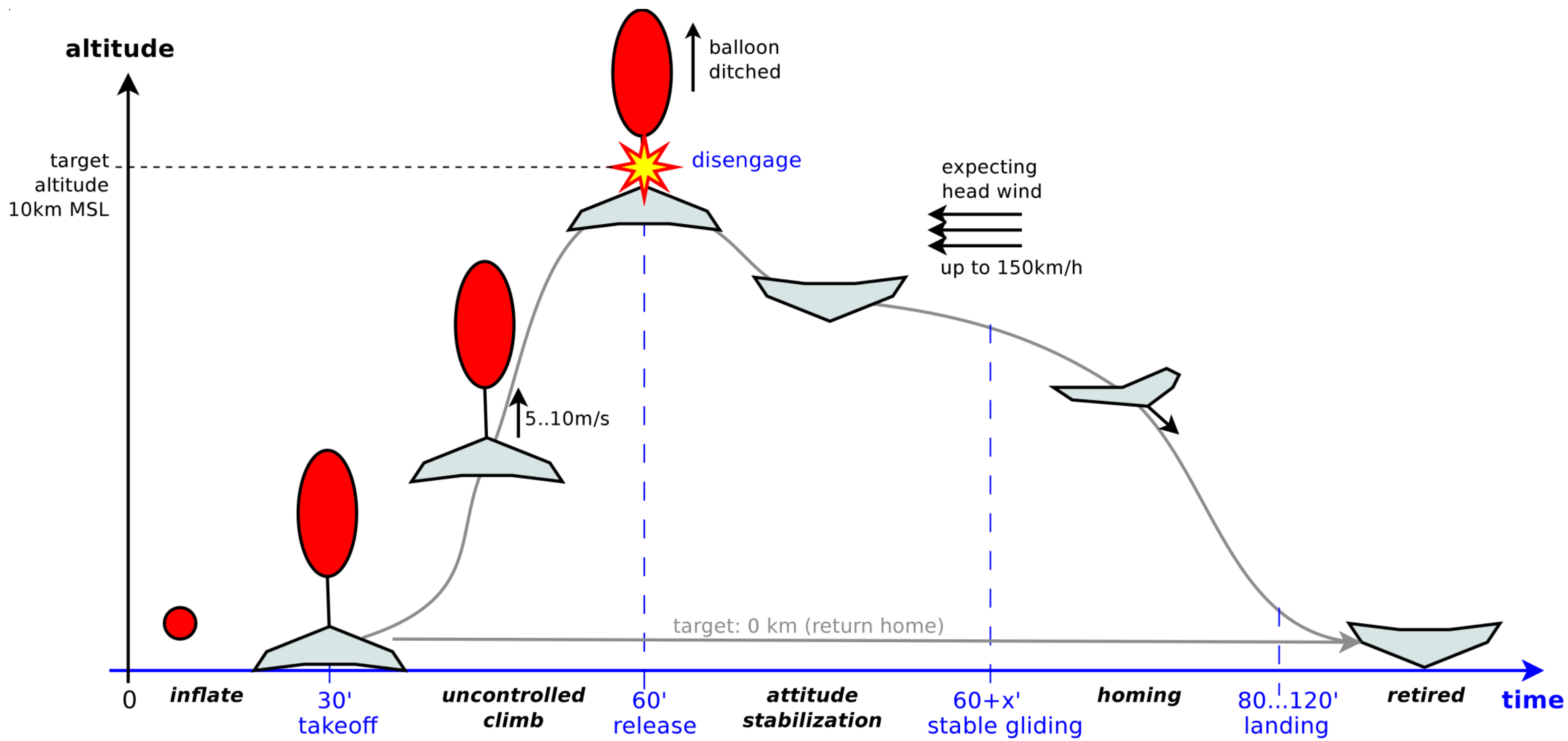
https://www.brightwork.com/blog/project-failures-boeings-787-dreamliner

Formal Verification

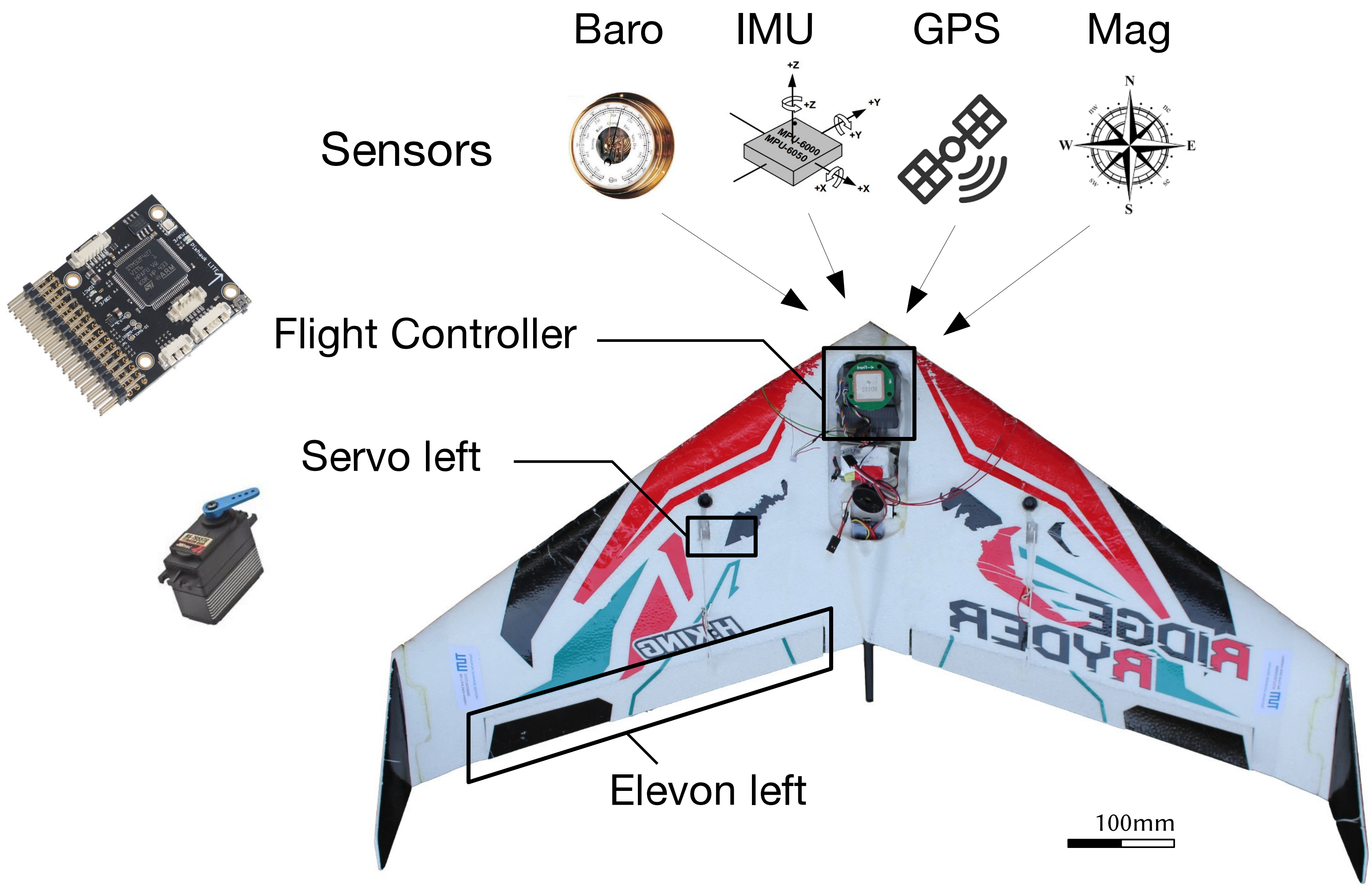
Rejected because considered to …

- require a lot of additional specification **?**
- require user interaction; little automation **?**
- require experts; results are difficult to understand **?**

Mission – Novel Weather Balloon



Introduction to the Scenario

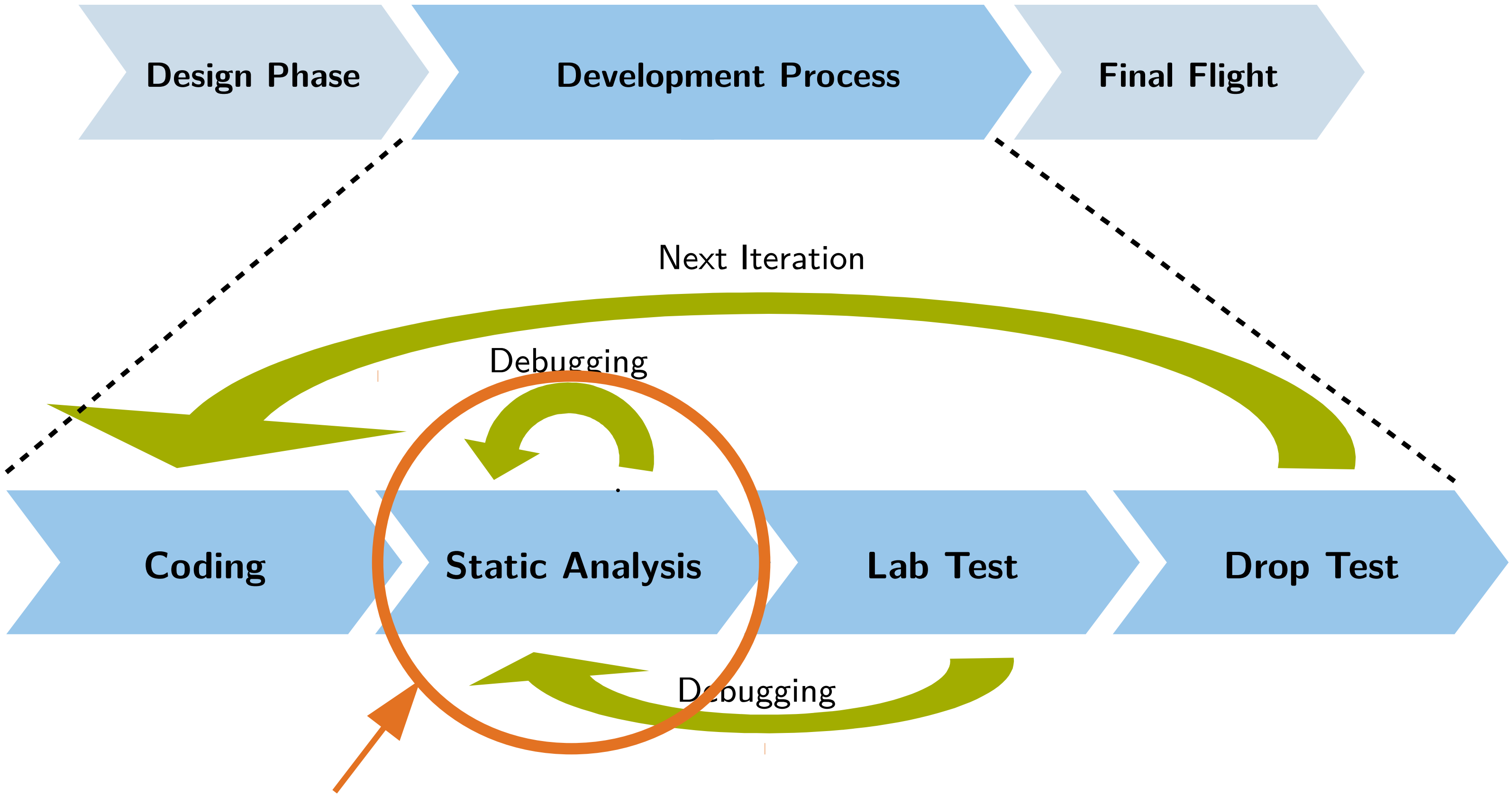


The Roll of System Testing

- Full system tests, including external effects (wind, etc..)
- Risky and high effort (Time&Money) ⇒ as little as possible
- Germany: Must not fly above 100m AGL ⇒ limited




Development Process



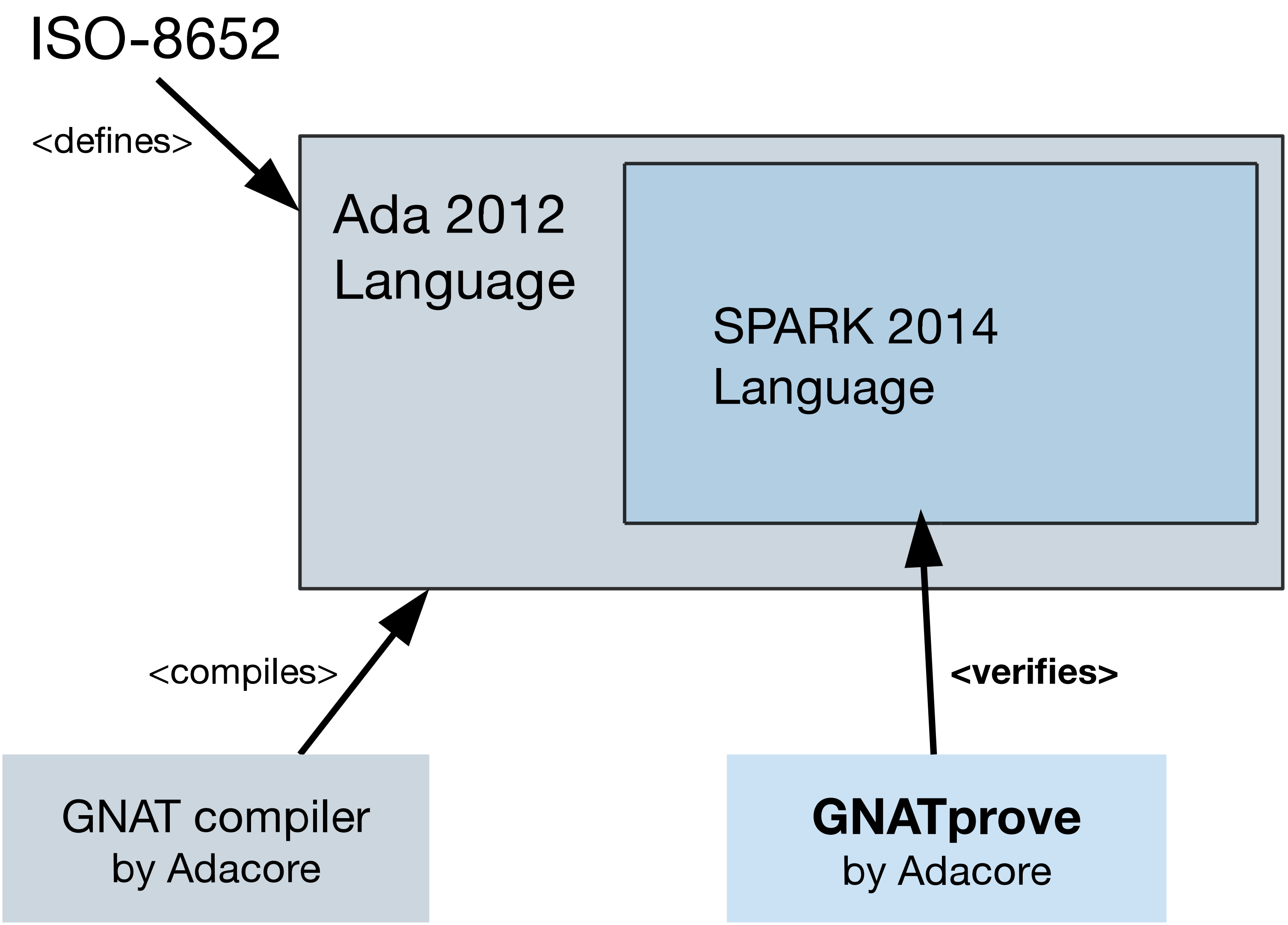
1. Fast: before compiling
2. Normal: Continuous Integration with git
3. Nightly "deep" verification runs with long timeouts

Finding Defects – Expectation

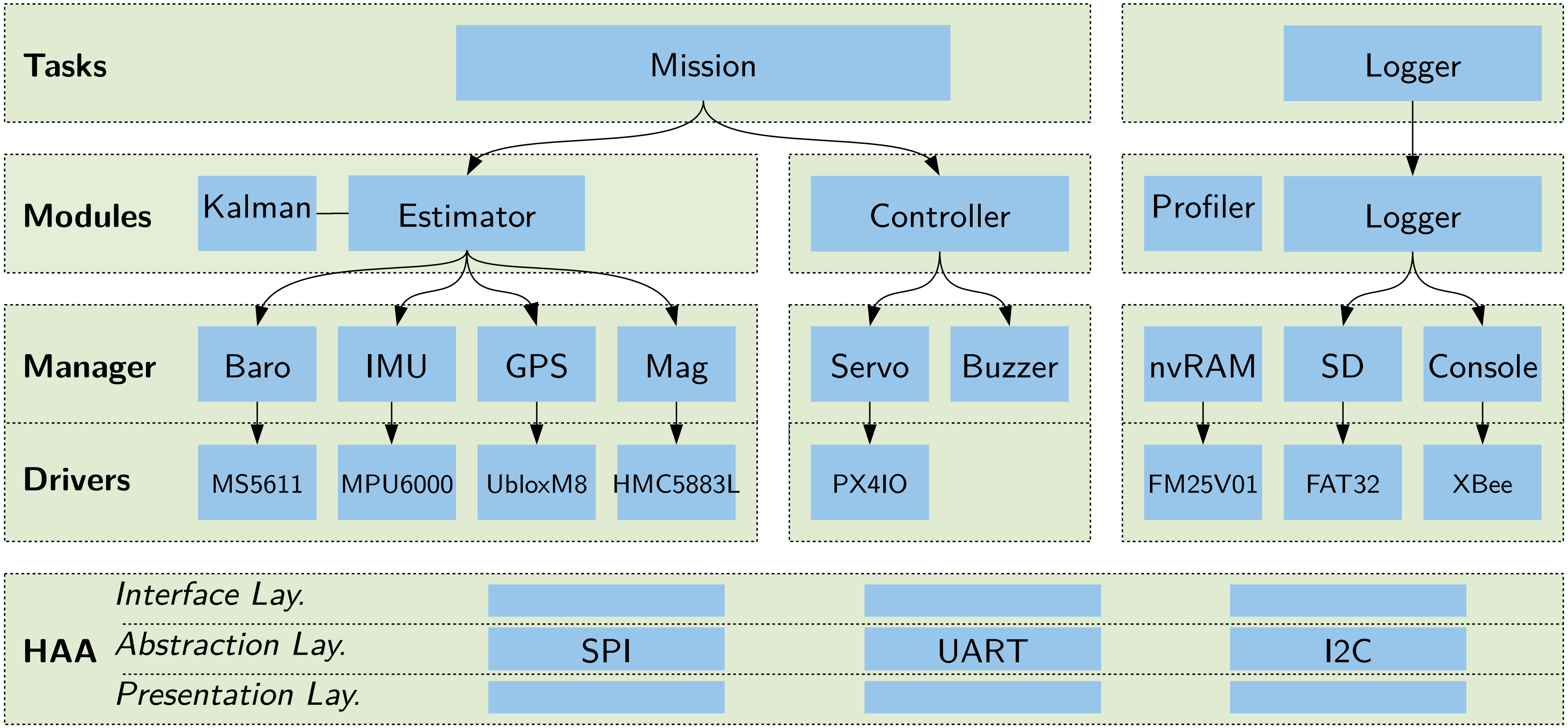
Static Analysis → System Testing → Operation

- most by static analysis (each developer & nightly runs)
  - replace unit testing
  - identify under-specification
- few by system testing
  - defects which were missed by static analysis
  - defects which require context beyond source code
  - logging of exceptions: no reproduction issues
- none during operation
  - nevertheless: logging of exceptions & in-air reset

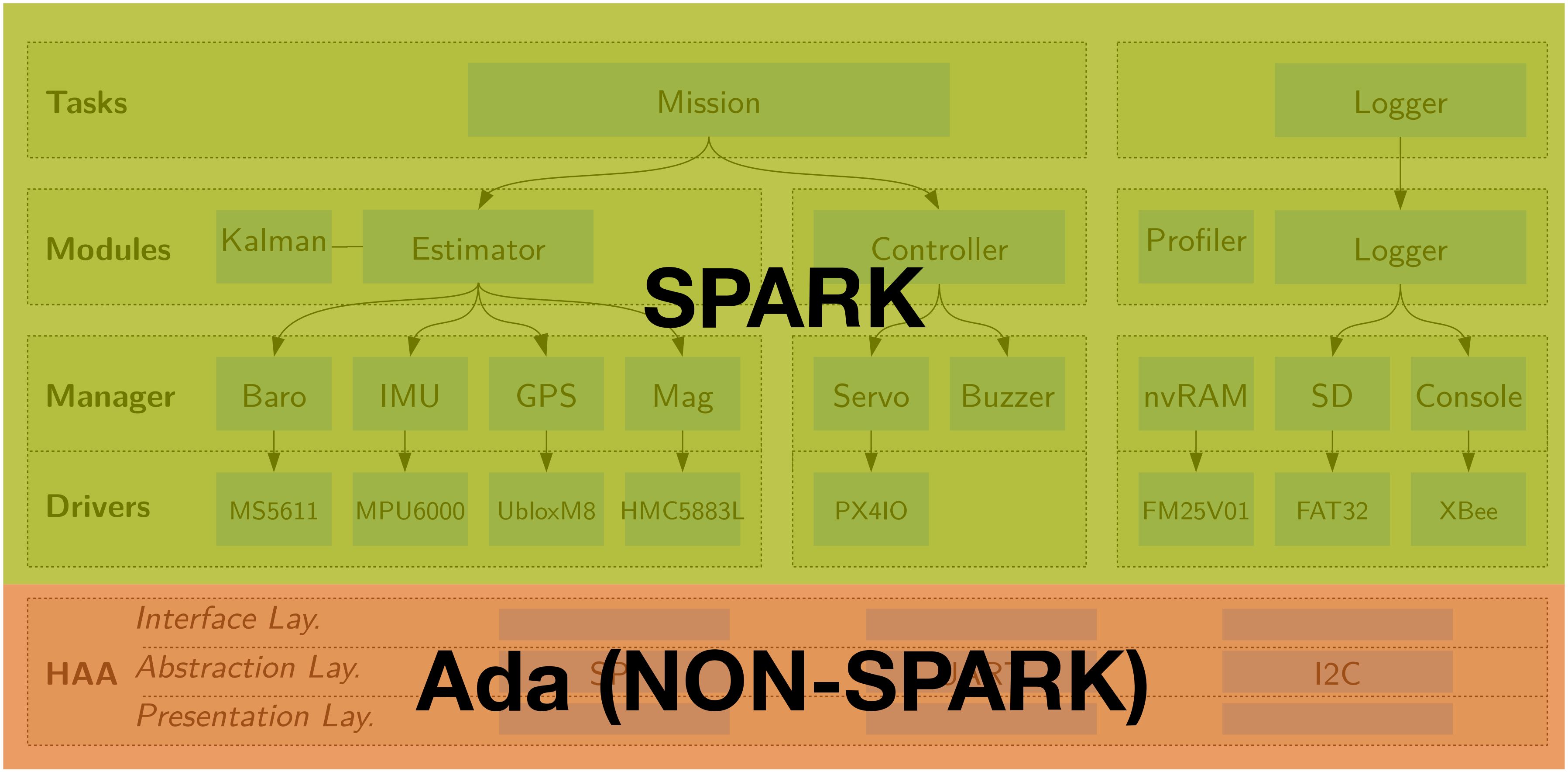
Ada & SPARK


ISO-8652
<defines>
Ada 2012 Language
SPARK 2014 Language
<compiles>
<verifies>
GNAT compiler by Adacore
GNATprove by Adacore

SW Architecture



SW Architecture



Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Absence of Run-Time Errors

```
package MyPack with SPARK_Mode is

  subtype Percentage is Natural;

  Global_Ratio : Percentage;

  procedure set_ratio( alt, maxalt : Integer )
  is begin
    Global_Ratio := alt * 100 / maxalt;
  end set_ratio;

end MyPack;
```

Absence of Run-Time Errors

```
1  package MyPack with SPARK_Mode is
2
3     subtype Percentage is Natural;
4
5     Global_Ratio : Percentage;
6
7     procedure set_ratio( alt, maxalt : Integer )
8     is begin
9       Global_Ratio := alt * 100 / maxalt;
10    end set_ratio;
11
12 end MyPack;
```

```
Mypack:9:35 medium: overflow check might fail
Mypack:9:41 medium: divide by zero might fail
Mypack:9:41 medium: range check might fail
```

Absence of Run-Time Errors

```
package MyPack with SPARK_Mode is

  subtype Percentage is Natural;

  Global_Ratio : Percentage;

  procedure set_ratio( alt, maxalt : Integer ) with
    Pre => alt,maxalt > 0 and alt < Integer'Last/100
  is begin
    Global_Ratio := alt * 100 / maxalt;
  end set_ratio;

  set_ratio( 42, 62 );
end MyPack;
```

Absence of Run-Time Errors

```
1  package MyPack with SPARK_Mode is
2
3    subtype Percentage is Natural;
4
5    Global_Ratio : Percentage;
6
7    procedure set_ratio( alt, maxalt : Integer ) with
8      Pre => alt,maxalt > 0 and alt < Integer'Last/100
9    is begin
10     Global_Ratio := alt * 100 / maxalt;
11   end set_ratio;
12
13   set_ratio( 42, 62 );
14 end MyPack;
```

```
Mypack:10:35 info: overflow check proved
Mypack:10:41 info: division check proved
Mypack:10:41 info: range check proved
Mypack:13:3 info: Precondition proved
```

Absence of Run-Time Errors

```
package MyPack with SPARK_Mode is

  subtype Tar_Alt is Integer range 10 .. 10_000;
  subtype Alt is Integer range 0 .. 100_000;
  subtype Percentage is Natural;

  Global_Ratio : Percentage;

  procedure set_ratio( val : Alt; max : Tar_Alt )
  is begin
    Global_Ratio := val * 100 / max;
  end set_ratio;

end MyPack;
```

Absence of Run-Time Errors

```
1  package MyPack with SPARK_Mode is
2
3     subtype Tar_Alt is Integer range 10 .. 10_000;
4     subtype Alt is Integer range 0 .. 100_000;
5     subtype Percentage is Natural;
6
7     Global_Ratio : Percentage;
8
9     procedure set_ratio( val : Alt; max : Tar_Alt )
10    is begin
11       Global_Ratio := val * 100 / max;
12    end set_ratio;
13
14 end MyPack;
```

```
Mypack:11:35 info: overflow check proved
Mypack:11:41 info: division check proved
Mypack:11:41 info: range check proved
```

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Flow Analysis

```
 1 package MyPack with SPARK_Mode is
 2
 3   subtype Percentage is Natural;
 4
 5   Global_Ratio : Percentage;
 6
 7   procedure set_ratio( alt, maxalt : Integer )
 8   is begin
 9     Global_Ratio := alt * 100 / maxalt;
10   end set_ratio;
11
12 end MyPack;
```

```
Mypack:9:19 info: initialization of "Global_Ratio" proved
```

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Dimension Checking

Scientific: angular rate = 20 deg / 100 ms = 200 deg/s

C Program:

```c
Float angle = 20;
Float dt    = 0.1;

Float rate = dt / angle;
```

SPARK Program:

```ada
angle  : Angle_Type := 20.0 * Degree;              -- Value: 0.524
dt     : Time_Type  := 100.0 * Milli * Second;  -- Value: 0.100
rate   : Angular_Velocity_Type := dt / angle;

yaw    : Angle_Type := 20.0;
```

Dimension Checking

Scientific: angular rate = 20 deg / 100 ms = 200 deg/s

C Program:

```
Float angle = 20;
Float dt    = 0.1;

Float rate = dt / angle;
```

SPARK Program:

```
1 angle : Angle_Type := 20.0 * Degree;                -- Value: 0.524
2 dt    : Time_Type  := 100.0 * Milli * Second; -- Value: 0.100
3 rate  : Angular_Velocity_Type := dt / angle;
4
5 yaw   : Angle_Type := 20.0;
```

```
Mypack:3:17 dimensions mismatch in assignment
Mypack:3:17 expected dimension [A.T**(-1)], found [T.A**(-1)]
Mypack:5:17 warning: assumed to be "20.0 Rad"
```

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

Physical Dimensions
Compliance with
physical laws

Functional Requirements

```
-- Functional Requirement
function FR_poshold_iff_no_course() return Boolean is (
    (Have_Course and G_state.mode /= MODE_POSHOLD) or
    (not Have_Course and G_state.mode = MODE_POSHOLD)
) with Ghost;

-- Functional Requirement
function FR_arrive_iff_near_target() return Boolean is (
    if (Have_Home_Position and Have_My_Position) then
        (dist2home < TARGET_R and G_state.mode = MODE_ARRIVED) or
        (dist2home >= TARGET_R and dist2home <= 2.0*TARGET_R)  or
        (dist2home > 2.0*TARGET_R and G_state.mode /= MODE_ARRIVED)
    else G_state.mode /= MODE_ARRIVED
) with Ghost;

-- Update the controller mode, depending on state
procedure Update_Homing() with
  Post => FR_poshold_iff_no_course and FR_arrive_iff_near_target;
```

Verification Goals

We want to formally verify

Absence of run-time errors
Division by zero, overflows

Integration Correctness
Valid inputs and outputs

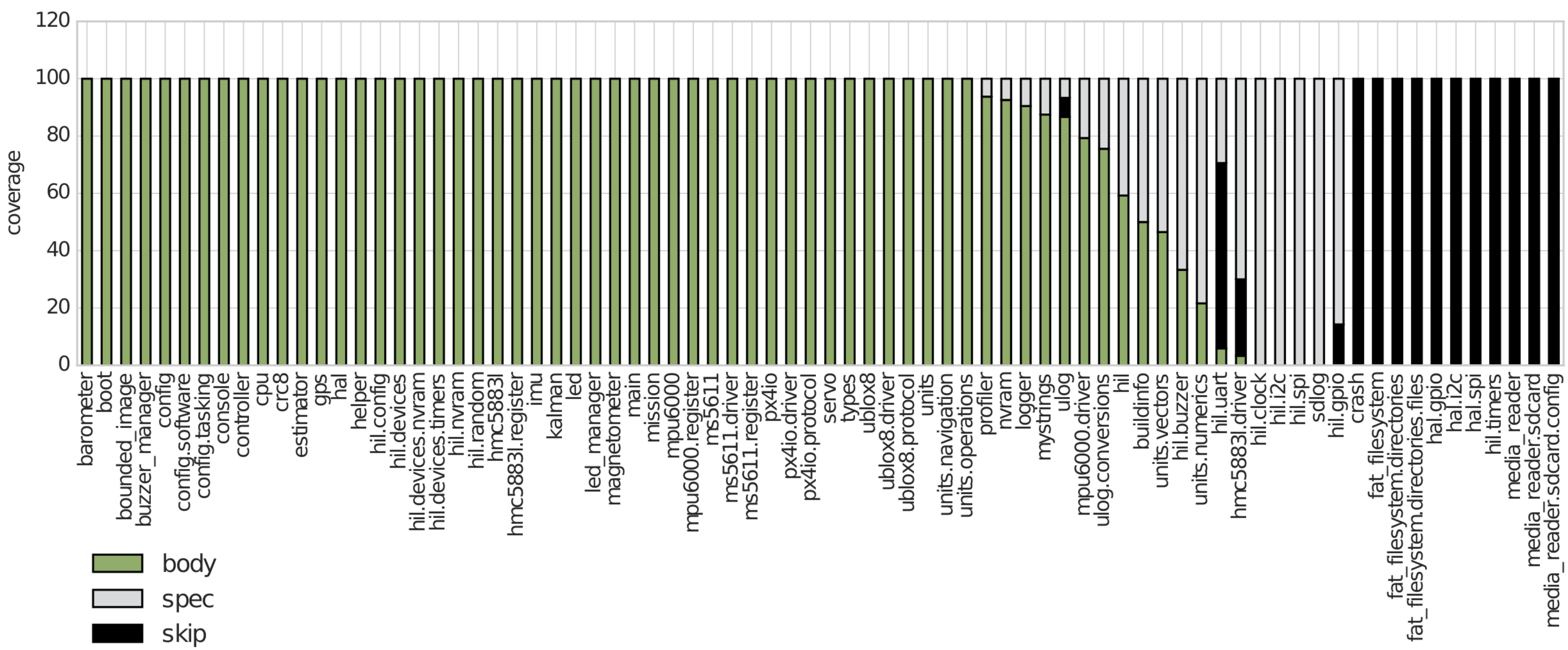
Functional Behavior
Input to output relation

Information Flow
Global variables,
Input to output dependencies

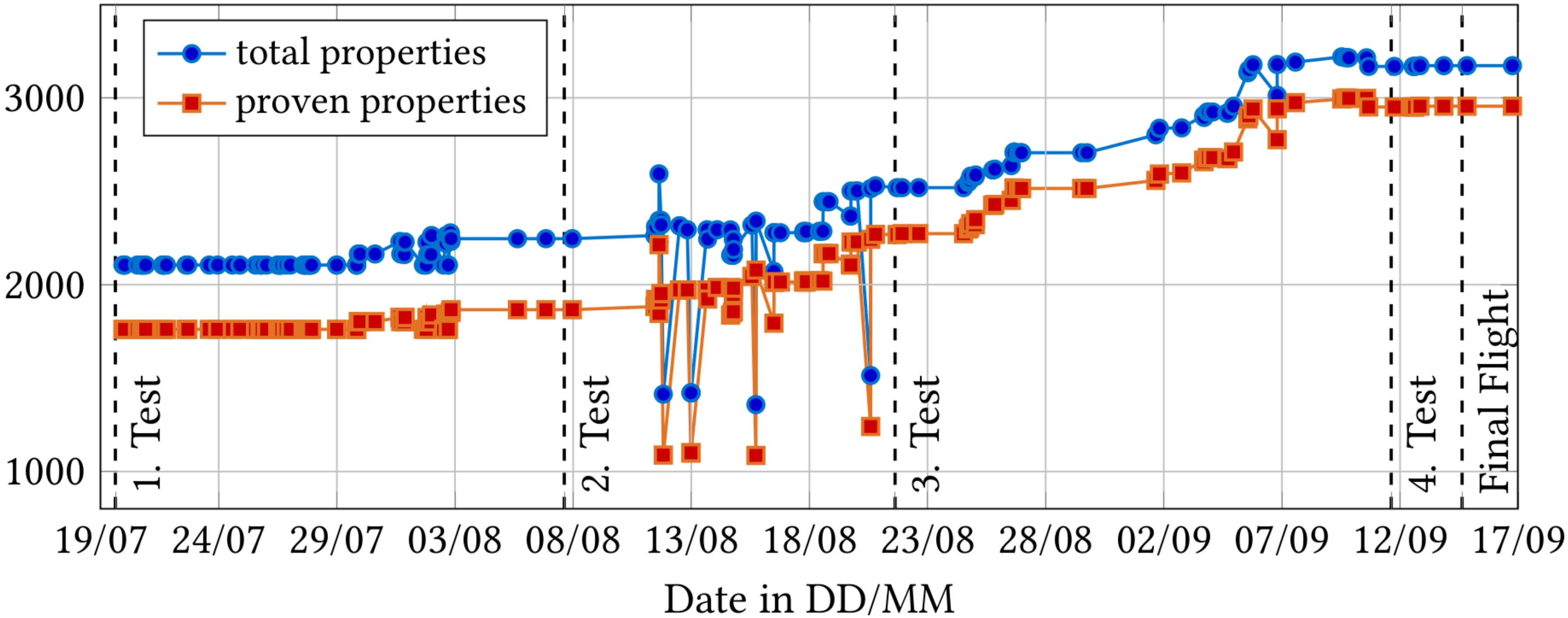
Physical Dimensions
Compliance with
physical laws

Final GNATprove Results

- SPARK subprogram coverage: 82%



Final GNATprove Results



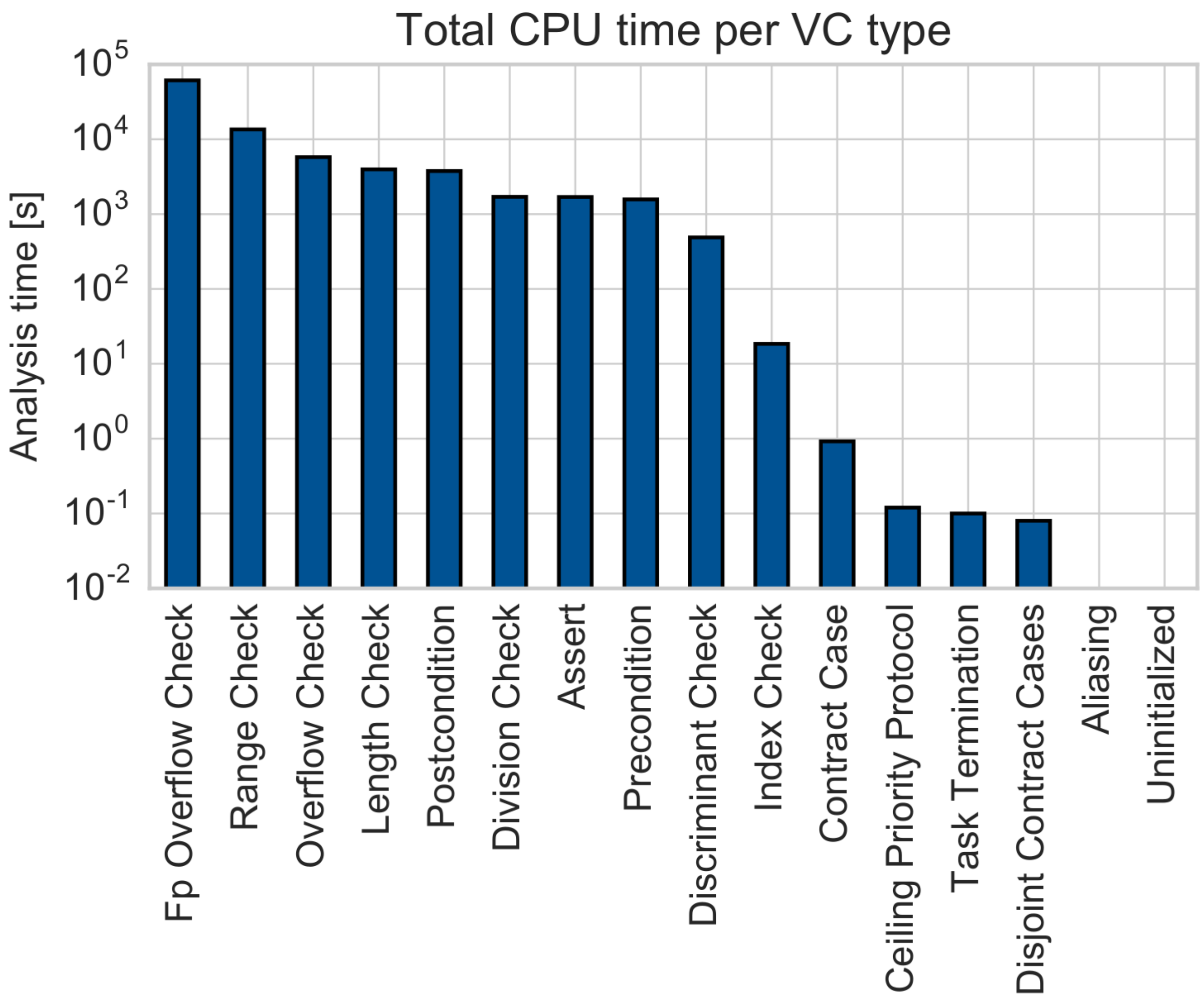
Final GNATprove Results

Totals of verified properties

Absence of run-time errors	Integration Correctness	Functional Requirements
1487 / 1711 (86.9%)	**277 / 282 (98.2%)**	**2 / 2 (100%)**

Information Flow	Physical Dimensions
1539 / 1540 (99.9%)	**?/? (100%)**

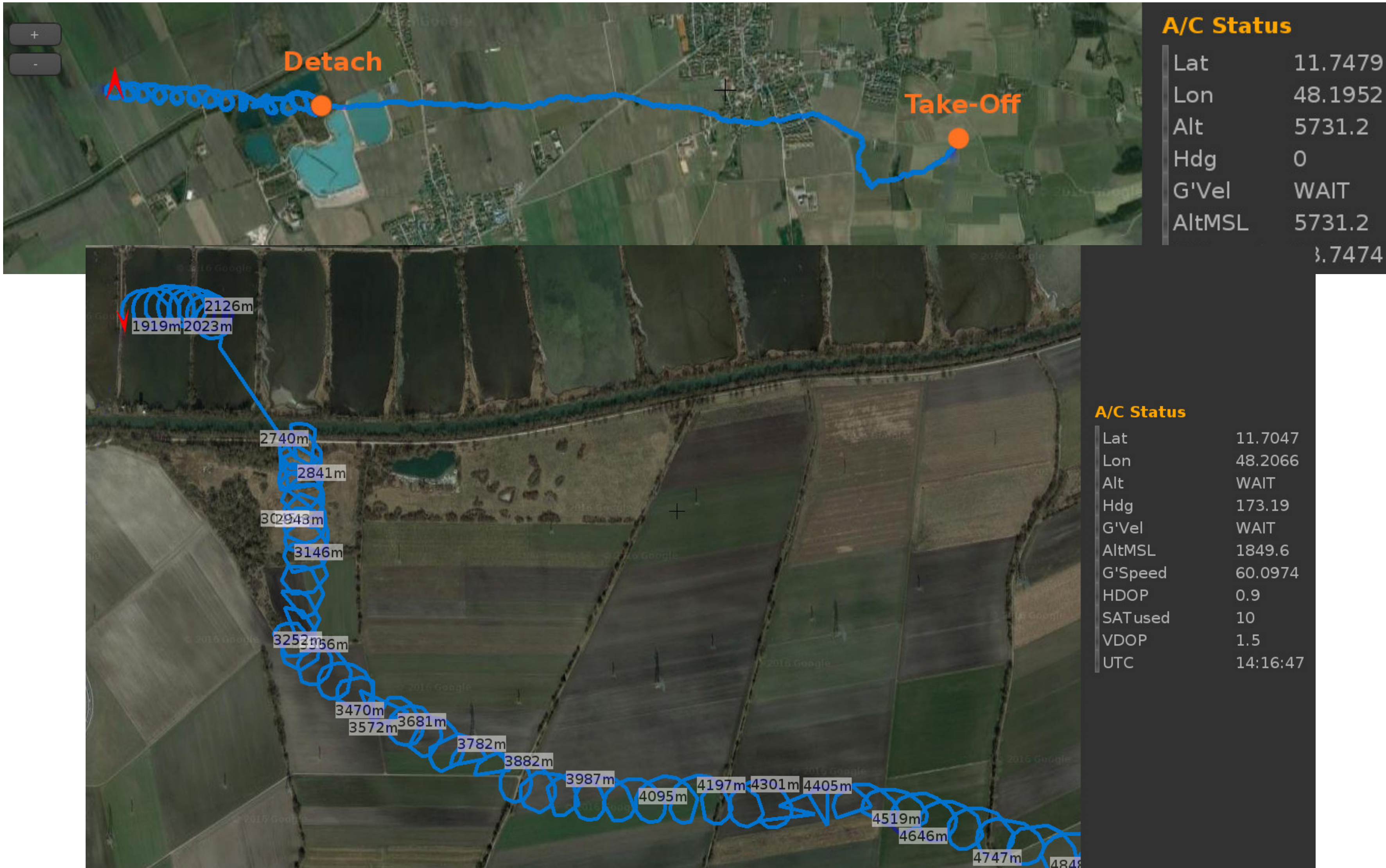
Final GNATprove Results



Final Flight on 2016-09-14

Target Altitude: 6100 m AGL

Final Flight on 2016-09-14



Finding Defects – Reality

Static Analysis → System Testing → Operation

- most by static analysis (each developer & nightly runs)
  - removed all stupid bugs
  - identified under-specification
- few by system testing
  - masking defects during analysis
  - ignoring failed proofs
  - incomplete specification
- **one during operation**
  - faulty but non-crashing behavior
  - missed during system testing
  - unverified assumptions about sensor data (beyond code)

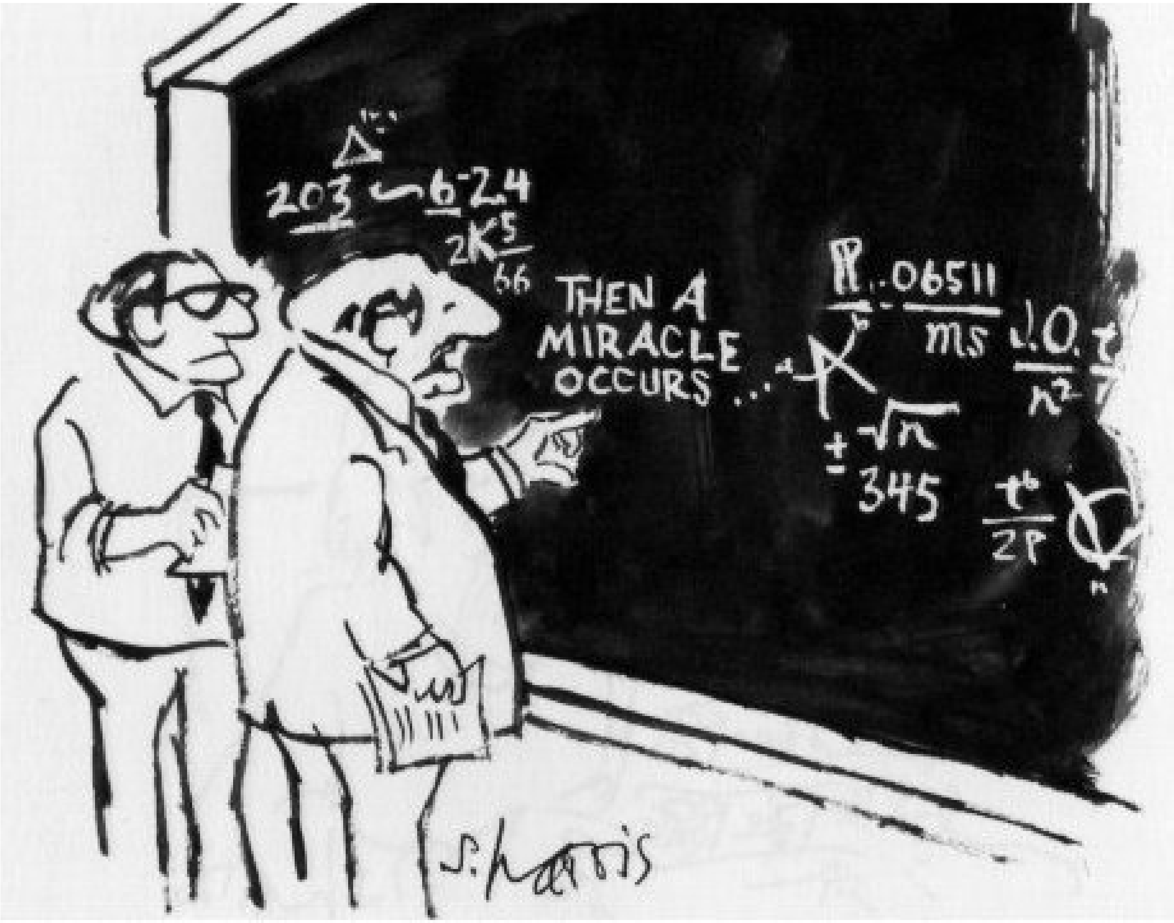
Conclusion

- Very little debugging work
  - Practically no exceptions during system testing
  - No issues with reproduction and isolation of failures

- SPARK tools work very well
  - Defect detection with almost no additional effort
  - Results are precise: `Mypack:9:35: overflow check might fail`
  - Effective multi-threading: separation of critical tasks
  - Verification automation as continuous integration with git
  - Verification of physical dimensions
  - Floats are difficult but possible
  - Verification of high-level behavior is difficult but possible

code released to open source: https://github.com/tum-ei-rcs/StratoX

Questions



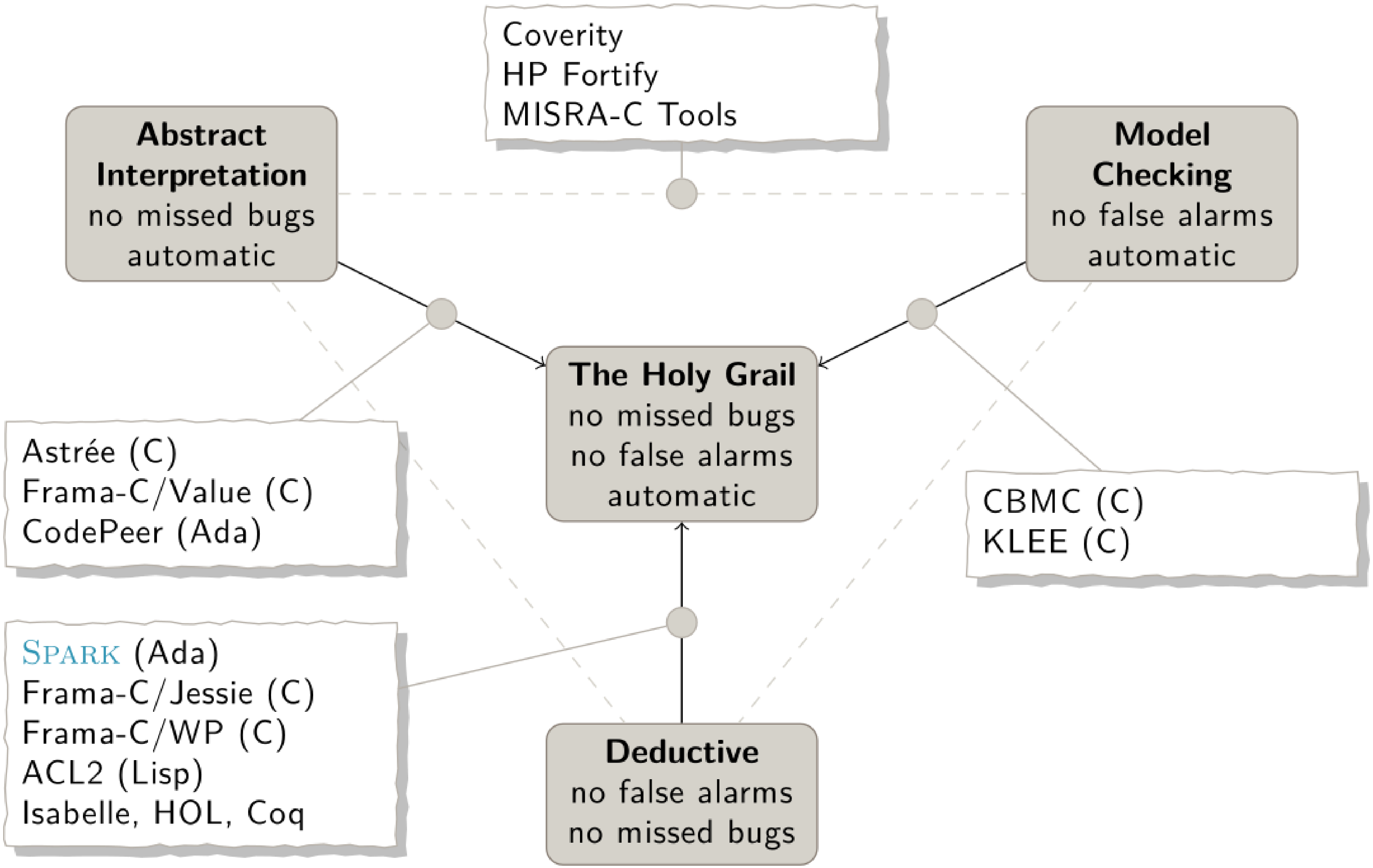
"I think you should be more explicit here in step two."

from *What's so Funny about Science?* by Sidney Harris (1977)

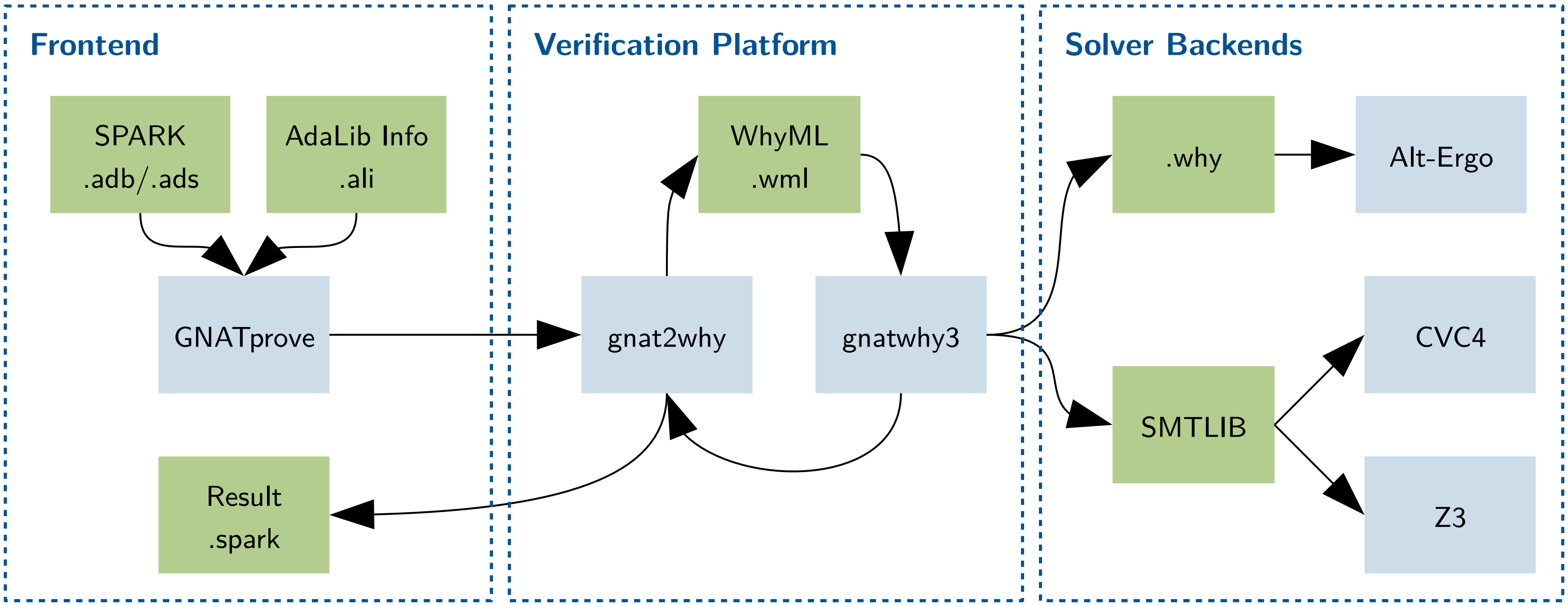
References

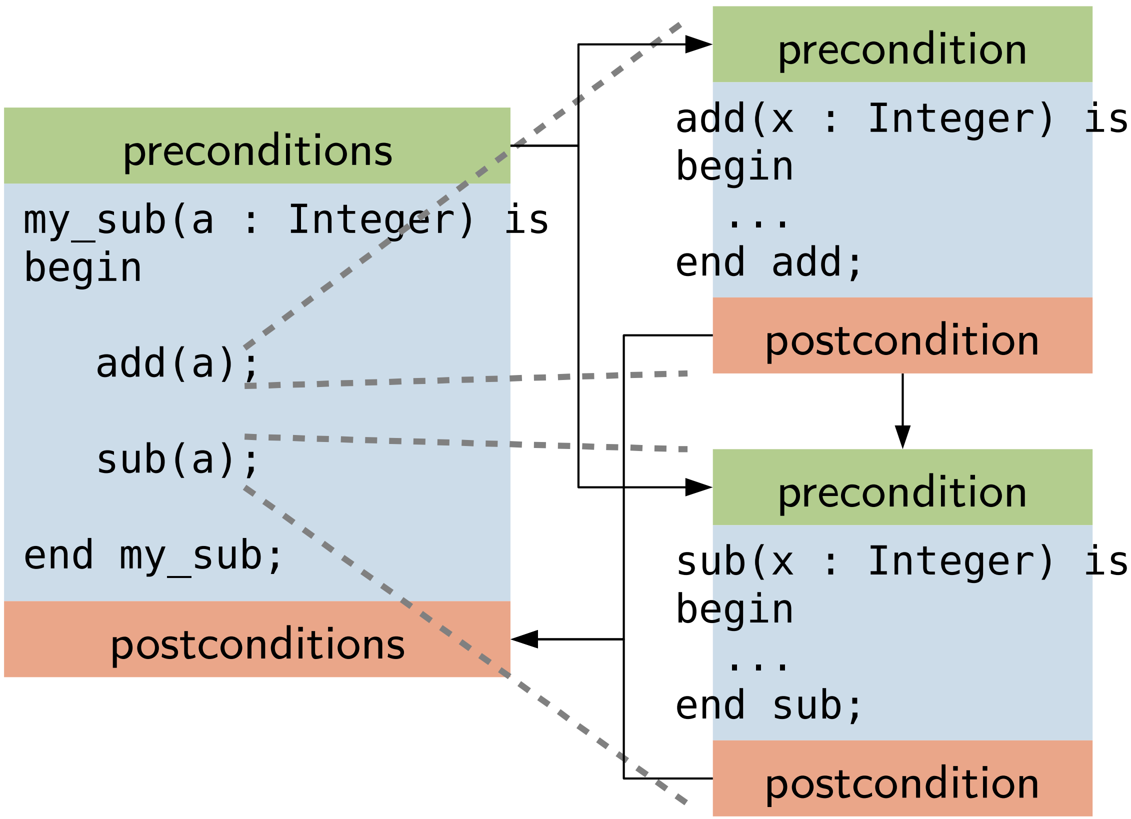
- M. Becker, E. Regnath, Samarjit Chakraborty “Development and Verification of a Flight Stack for a High-Altitude Glider in Ada/SPARK 2014”, In 36th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Trento, IT.
- “Airworthiness directives; the boeing company airplanes”, Federal Aviation Administration, Tech. Rep. 2015-10066, May 1, 2015.
- Adacore, SPARK 2014 reference manual.
- Adacore, SPARK 2014 user guide, version 18.0.
- Ada reference manual, ISO/IEC Std. 8652, 2012

Formal Verification



GNATprove internal

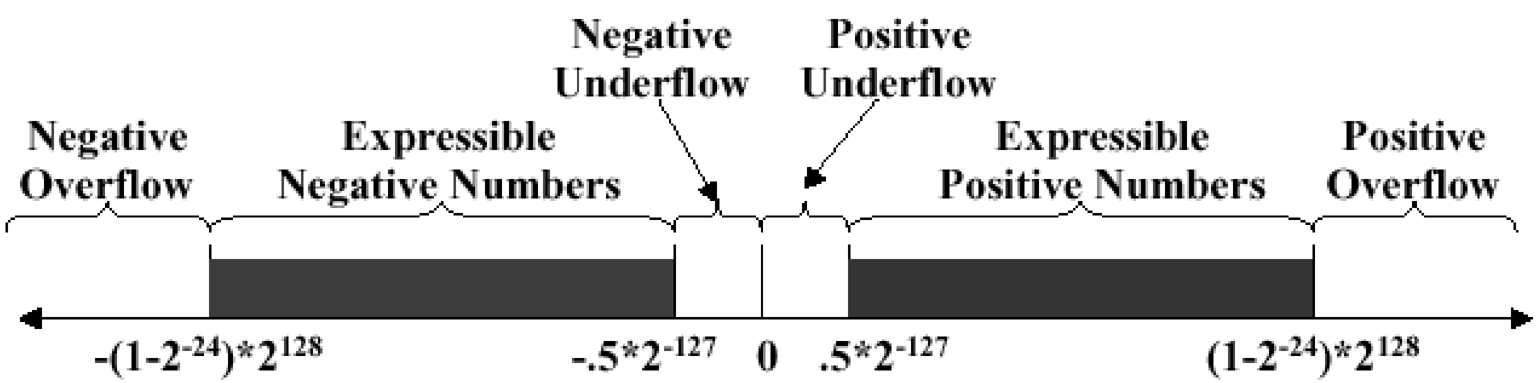



preconditions
my_sub(a : Integer) is
begin
add(a);
sub(a);
end my_sub;
postconditions
precondition
add(x : Integer) is
begin
...
end add;
postcondition
precondition
sub(x : Integer) is
begin
...
end sub;
postcondition

FP Underflow

```
-- Float Underflow
function Sin ( x : Float ) return Float with
        Post => Sin'Result in -1.0 .. 1.0;                  -- OK

pragma Assert ( ( Sin(x) )**2 in -1.0 .. 1.0 );  -- Might fail
```



Final GNATprove Results

SPARK Analysis	Total	Flow	Interval	Proved	Justified	Unproven
Data Dependencies						
Flow Dependencies						
Initialization	1540	1510			29	1
Non Aliasing	16	16				
Run-Time Checks	1711		366	1117	4	224
Assertions	15			15		
Functional Contracts	282			277		5
LSP Verification						
Total	3564	1526	366	1409	33	230

Subprogram Coverage : 538 / 1227 (43.8%) 538 / 654 (82%)

Proven Properties: 3334 / 3564 (93.5%)

Proven Run-Time Errors: 1487 / 1711 (86.9%)