Building Better and Safer Programs Using SPARK and Ada

William Wong
23 March 2019

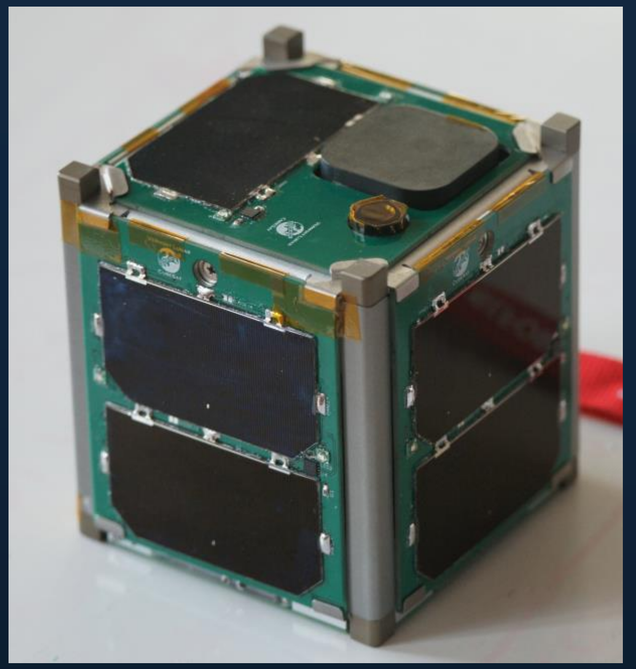
TCF19

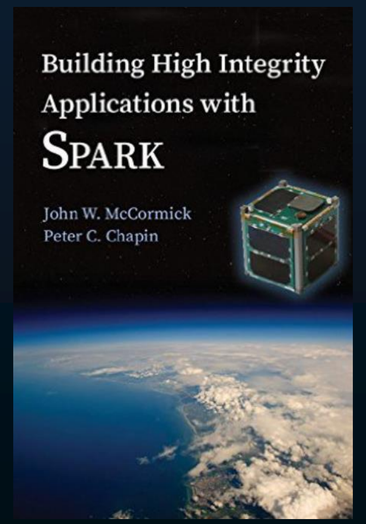
Intent of This Presentation

- **Highlight Ada and SPARK programming features**
  - **Provide a general overview of major features**
- **Get you to consider using Ada or SPARK for new projects**
  - **Make the compiler do the work**
- **Present Ada and SPARK resources**

When It Just Has to Work

- **Vermont Lunar CubeSat**
  - **Launched in NASA's ELaNa IV on an Air Force ORS-3 Minotaur 1 flight November 19, 2013 to a 500 km altitude and remained in orbit until reentry over the central Pacific Ocean, November 21, 2016. Eight other CubeSats were never heard from, two had partial contact for a week, and another worked for 4 months.**
  - **Operated until reentry after 11,071 orbits**
  - **Code written in SPARK/Ada 2014**
    - **5991 lines of code**
    - **2843 lines of SPARK annotations**
    - **98% proven automatically**



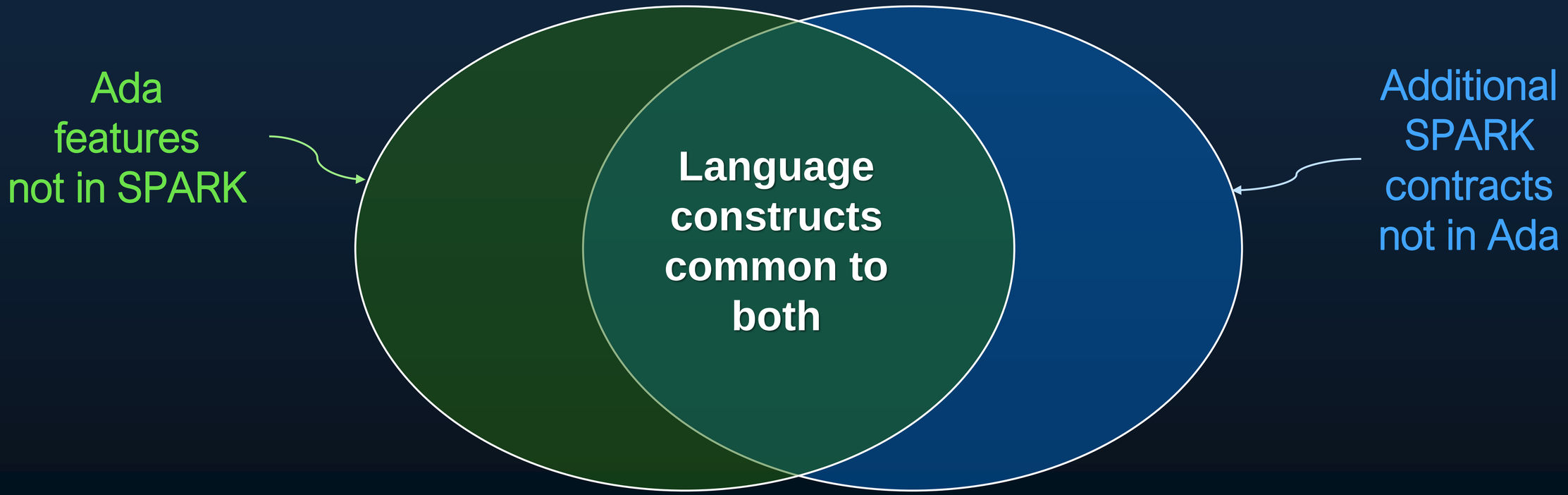


TCF19

What is Ada and SPARK

- **Ada**
  - **A programming language develop in 1980 for the U.S. Department of Defense (DoD) for high integrity, embedded and real-time systems**
  - **Major revisions: Ada 95, Ada 2005, Ada 2012**
  - **Supports strong typing, modularity mechanisms (packages), run-time checking, parallel processing (tasks, synchronous message passing, protected objects, and nondeterministic select statements), exception handling, and generics. object-oriented programming, including dynamic dispatch**
- **SPARK**
  - **Subset of Ada, SPARK 2014 is based on Ada 2012**
  - **Provable programs using contracts**

SPARK and Ada



SPARK also serves as a testbed for Ada 202x

Why Consider Ada and SPARK

- **Improve code quality by reducing errors**
- **Rising security requirements**
- **Rising safety/liability issues**
- **Reduce programming costs (more on this later)**
- **Portability**
- **Provable programs when using SPARK**

Reducing Development/Support Costs

- **MITRE**
  - **Annual Average Costs for Software Maintenance, 1994**
- **Steve Zeigler**
  - **Comparing Development Costs of C and Ada, March 30, 1995**
- **Capers Jones**
  - **Letter to CrossTalk, Oct. 1998**
- **VDC Research**
  - **Controlling Costs with Software Language Costs, 2018**
  - **https://www.adacore.com/papers/controlling-costs-with-ada**

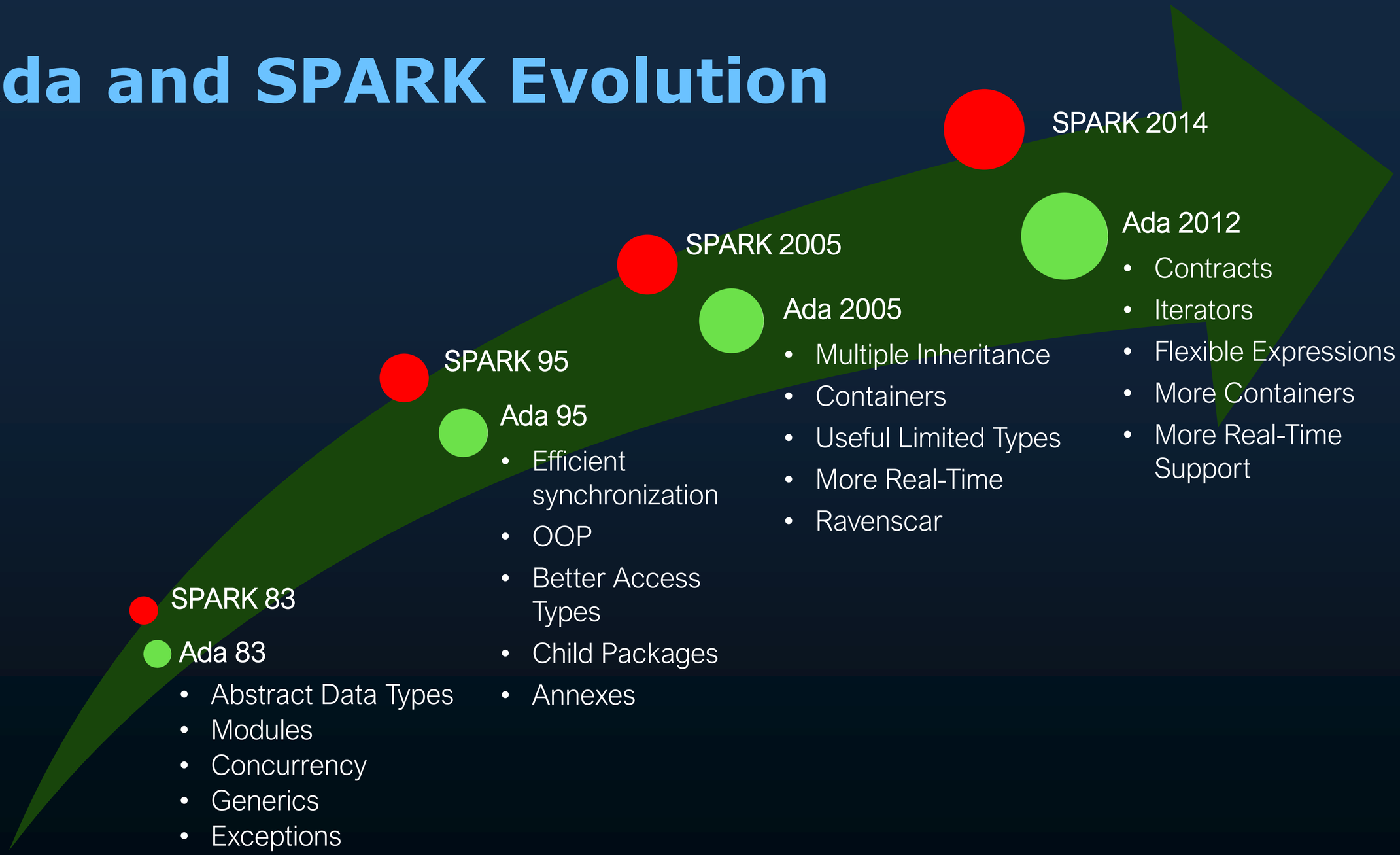
SPARK Reliability Example

- MULTOS is an “operating system” for smartcards
- 100,000 lines of SPARK code
- Industry standard is 5 defects per 1,000 lines
  - Thus approximately 500 defects expected
- Only 4 defects reported 1 year after delivery
  - 0.04 per KSLOC
  - Corrected under warranty (!)
- Ultra-high reliability achieved

SPARK Productivity Example

- **Industry standard is 10 lines of code per day**
  - **Fully documented, tested, everything**
  - **In any language**
- **MULTOS project achieved 28 lines of code per day**
- **Very high levels of productivity achieved**

Ada and SPARK Evolution



11 Myths About Ada *(on electronicdesign.com)*

1. Ada is dead
2. Ada code is slow and bulky
3. Ada doesn't let you get "down and dirty" with the hardware
4. Ada is stuck in the 1980s
5. Ada requires a whole new way of thinking about programming
6. Ada requires special compiler technology and does not fit in well with other languages
7. Ada is not used in academia
8. Ada doesn't have many tools beyond the compiler
9. Ada is too complex
10. Ada was designed by committee
11. Most software problems are due to poorly specified requirements rather than coding errors, so the programming language doesn't matter that much

Ada Features

- Intelligent syntax
- Precise data type definition
- Built-in concurrency and real-time support
- Nested procedures and functions
- Object-oriented programming
- Generic templates
- Programming in the large
- Contracts

Ada Feature: Intelligent Syntax

Lets the compiler do the checking

```
-- Assignment using := not =
My_variable := 100;

-- Matching begin/end names
-- Edit templates handle extra verbosity
procedure Swap ( X, Y: in out integer ) is
  T : integer := X ;
begin
  X := Y ;
  Y := T ;
end Swap;
```

Reduces typographical errors

Ada Feature: Precise Data Types

- **Precise data type definition**
  - **Fixed point and complex values**
  - **Constraint checking for variable values and array ranges**
  - **Bit and byte layout record definitions**
- **Storage pools are part of the language, not an add-on library**

Ada Feature: Precise Data Types

Fine grain specifications

```
-- Integers
type Buffer_Size is range 1 .. 200;

-- Fixed point
type My_Fixed is delta 0.01 range 0.0 .. 10.0;

-- Decimal
type Percentage is delta 0.1 digits 4 range 0.0 … 100.0 ;

-- modular
type Byte is mod 256;

-- Subtypes
subtype Positive is Integer range 1..Integer'Last;
```

Checked at Run-Time

Ada Feature: Constraint Checking

Prevent buffer overflows

```
procedure sample is
  -- Flexible Array Bounds
  my_array : array (-10 .. 10) of integer;

  -- Enumerated Array Bounds
  type days is (Sunday, Monday, Tuesday, Wednesday,
       Thursday, Friday, Saturday);

  hours: array(Sunday .. Saturday) of integer;

  -- Range check of variables
  type Buffer_Size is range 1 .. 200;
  My_buffer_size : Buffer_Size := 200;

 -- Generates Constraint Error
 My_buffer_size := My_buffer_size + 1 ;
```

Checked at run-time if range cannot be proven at compile time

Ada Feature: Bit Field Definitions

No masks needed

```
for T use record
  I at 0 range  0 .. 15;  -- two bytes
  A at 2 range  0 .. 12;
  B at 2 range 13 .. 17;
  C at 2 range 18 .. 23;
  D at 5 range  0 ..  7; -- one byte
end record;
```

Coexistance of big and little endian support

```
Big endian
Byte 0       1       2       3       4       5
     01234567012345670123456701234567012345670123456701234567
     IIIIIIIIIIIIIIIIAAAAAAAAAAAAABBBBBCCCCCCDDDDDDDD

Little endian
Byte         5       4       3       2       1       0
     DDDDDDDDCCCCCCBBBBBAAAAAAAAAAAAAIIIIIIIIIIIIIIII
     76543210765432107654321076543210765432107654321076543210
```

TCF19

Ada Feature: Concurrency Support

- **Runtime provided or support can be mapped to RTOS**
- **Standard features:**
  - **Tasks**
  - **Rendevous**
  - **Selective Wait**
  - **Guards**
  - **Protected Types:** Allows safe usage of variable data without the need for any explicit mutual exclusion and signaling mechanisms
- **Concurrency profiles** *(next slide)*

Ada Feature: Concurrency Profiles

- **Full**
  - **All features are available**
- **Ravenscar**
  - **Subset of Ada concurrency support, no dynamic task priorities**
  - **Designed for safety-critical applications**
- **Jorvik** *(Extended Ravenscar)*
  - **Relaxes some restrictions such as those related to protected entries**

Ada Feature: Nested Procedures

Nested procedures keep local functionality local

```
procedure Nested_function_example is
   -- local variables accessible by nested procedure check_limit
   Count: Integer := 0;
   Limit: Integer := 2;
   procedure check_limit ( C : Integer ) is
   begin
      if C <= Limit then
         Count := Count + 1;
      end if;
   end check_limit;

   type Vector is array (Integer range <>) of Integer;
   X : Vector := (1,2,3,4,5);
   Index : Integer;
begin
   -- map check_limit over vector X
   for Index in X'Range loop
      check_limit( X( Index ));
   end loop;
end Nested_function_example;
```

Ada uses nested functions instead of lambdas

TCF19

Ada Feature: Object Oriented Programming

- Polymorphism
- Interfaces
- Multiple inheritance
- Inheritance related pre and post condition contracts

Ada Feature: Generics

- Generic subprograms (procedures)
  - Comparable to Java and C++ template function definitions
- Generic packages (classes)
  - Comparable to Java and C++ template class definitions
- Generic parameters
  - Example: Allows the definition of a sort algorithm for any kind of array

Ada Feature: Programming in the Large

- **Ada is designed to handle large, complex projects**
- **Hierarchical package system**
- **Exception handling**
- **Encapsulation**
- **Access types**
- **OOP contract support** *(already mentioned)*
- **Generics** *(already mentioned)*

Programming in the Large: Exceptions

Exception handling organized around statement blocks

```
begin

   -- some code

exception
   when Constraint Error | Storage Error =>
      -- error handling code
   when others =>
      -- other error handling code
end;
```

Actually easier to read syntax than some other programming languages

Programming in the Large: Encapsulation

```
package Stacks is
   type Stack_Type (Size : Integer) is private;

private
   type Stack_Type (Size : Integer) is record
      V : Integer;
   end record;

end Stacks;
```

Public discriminants on private types

Only expose what is necessary

Programming in the Large: Access Types

- **Mitigating factors**
  - **Arrays are cannot be used as pointers**
  - **Parameters are implicitly passed by reference**
- **Access types** (pointers) **are distinct types that can be limited to referencing specific areas such as data pools**
- **Accessibility checks allow only valid references**

Ada Feature: Contracts

- Added in Ada 2012
- Contract types
  - Procedure pre and post conditions
  - Subtype predicates
  - Type invariants
  - Loop invariants
- Required by SPARK to support provability

Contracts Example: Pre/Post

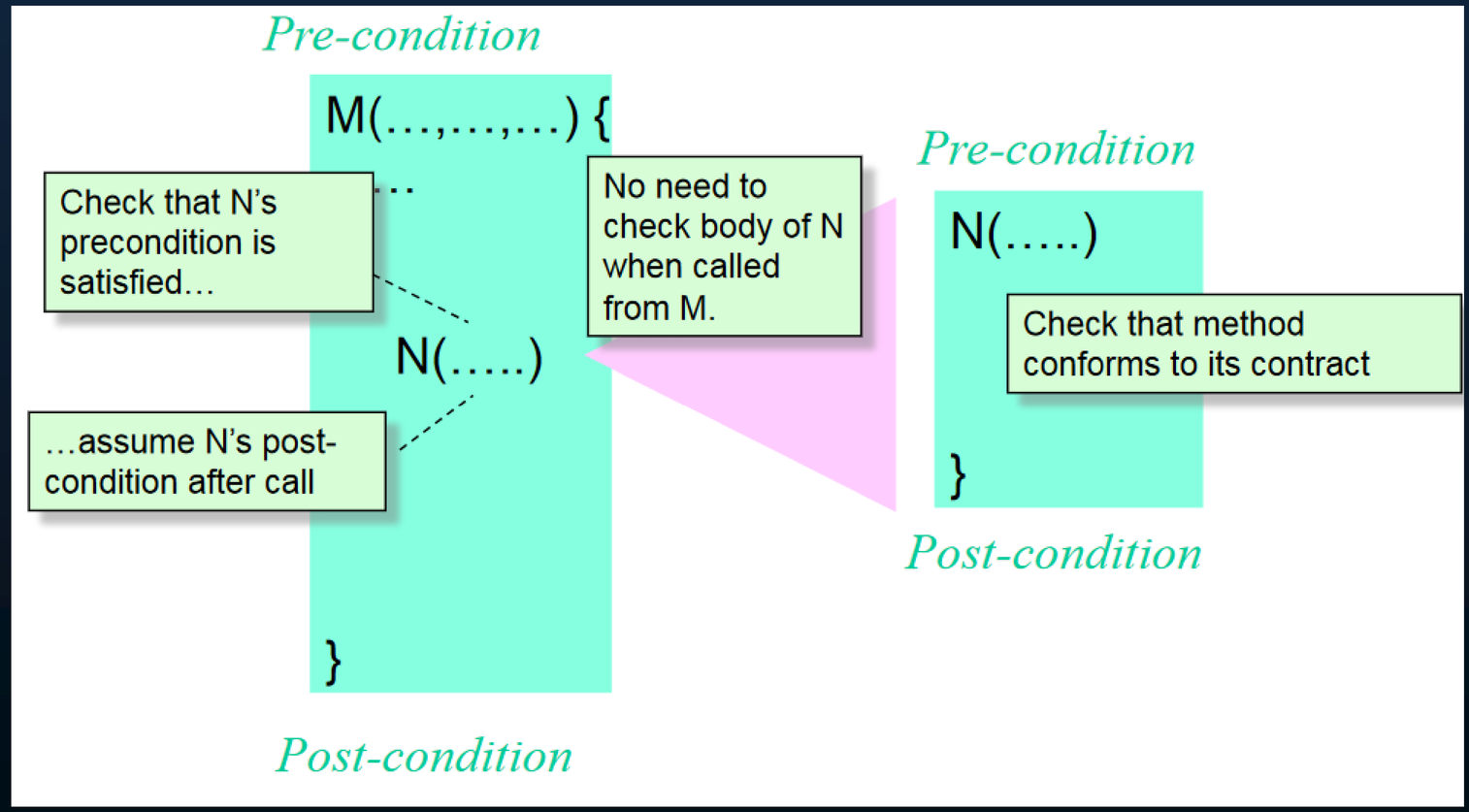
Low-Level Requirements

```
procedure Push (This : in out Stack;  Value : in Content) with
   Pre    =>  not Full (This),
   Post   =>  not Empty (This)
              and Top (This) = Value
              and Extent (This) = Extent (This'Old) + 1
              and Unchanged (This'Old, Within => This);

...

function Full (This : Stack) return Boolean;

function Empty (This : Stack) return Boolean;

function Top (This : Stack) return Content with
   Pre => not Empty (This);

function Extent (This : Stack) return Content_Count;

function Unchanged (Invariant_Part, Within : Stack) return Boolean;
```

Checked at Run-Time

Contracts: Pre/Post Advantage

- **Allows code analysis of caller without access to called code because pre and post conditions are defined**



Contracts Example: Predicates

Predicates apply to types

```
procedure Show_Dynamic_Predicate_Courses is

  pragma Assertion_Policy (Dynamic_Predicate => Check);

package Courses is
  type Course_Container is private;

  type Course is record
    Name       : Unbounded_String;
    Start_Date : Time;
    End_Date   : Time;
  end record
    with Dynamic_Predicate =>
      Course.Start_Date <= Course.End_Date;

  -- additional definitions
begin
  -- additional code
end Show_Dynamic_Predicate_Courses;
```

Static and dynamic predicates are supported

Contracts Example: Type invariant

Predicates apply to types

```
procedure Show_Type_Invariant is
  pragma Assertion_Policy (Type_Invariant => Check);
  package Courses is
     type Course is private
      with Type_Invariant => Check (Course);
     type Course_Container is private;

     function Check (C : Course) return Boolean;
   private
     type Course is record
        Name       : Unbounded_String;
        Start_Date : Time;
        End_Date   : Time;
     end record;
     function Check (C : Course) return Boolean is
       (C.Start_Date <= C.End_Date);
  end Courses;
  begin
  end Show_Type_Invariant;
```

Used exclusively to check private types

Contracts Example: Loop invariant

Loop invariant condition must be true for each iteration

```
function Get_Prime (Low, High : Positive) return Natural is
   J : Positive := Low;
 begin
   while J <= High loop
     if Is_Prime (J) then
       return J;
     end if;

     pragma Loop_Invariant
       (J in Low .. High
         and
       (for all K in Low .. J => not Is_Prime (K)));

    J := J + 1;
   end loop;
   return 0;
 end Get_Prime;
```

Used for formal verification

What Can We Prove, Statically?

- **Freedom from run-time errors (!!)**
  - **No buffer overflow, numeric overflow, divide by zero, invalid array indexes, etc.**
  - **Safe to turn off exception checks!**
- **Data and Information flow**
  - **No uninitialized variables, no unused assignments, etc.**
  - **Data only go where you mean them to go**
- **Functional correctness at unit level**
- **Arbitrary properties, e.g., security and safety**

Proving Functional Correctness Example

```
procedure Push (This : in out Stack;  Value : in Content) with
    Pre    =>  not Full (This),
    Post   =>  not Empty (This)
               and Top (This) = Value
               and Extent (This) = Extent (This'Old) + 1
               and Unchanged (This'Old, Within => This) ,
    Global => null,
    Depends => (This =>+ Value);

...

function Full (This : Stack) return Boolean;
function Empty (This : Stack) return Boolean;
function Top (This : Stack) return Content with
    Pre => not Empty (This);
function Extent (This : Stack) return Content_Count;
function Unchanged (Invariant_Part, Within : Stack) return Boolean;
```

Verified Statically

Pass the tests the first time!

TCF19

Proving Abstract Properties Example

```
type Move_Result is (Full_Speed, Slow_Down, Keep_Going, Stop);

procedure Move
    (Train          : in   Train_Id;
     New_Position   : in   Train_Position;
     Result         : out  Move_Result)
with
      Pre   =>  Valid_Id (Train) and
                Valid_Move (Trains (Train), New_Position) and
                At_Most_One_Train_Per_Track and
                Safe_Signaling,

      Post  =>  At_Most_One_Train_Per_Track and
                Safe_Signaling;


function At_Most_One_Train_Per_Track return Boolean;

function Safe_Signaling return Boolean;

...
```

TCF19

Discussion: Why assert() is not a contract

- C/C++ support the *assert()* macro
- Each assert instance only checks a condition at the point of execution
- The macro provides only limited runtime checking
- Other issues
  - The macro is expanded by the pre-processor so some operations are not available such as *sizeof* the is computed by the compiler
  - Macro placement allows preconditions to tested but post conditions can be a challenge especially with multiple exit points

SPARK Feature: Ghost Code

- Additional code that is only used to prove a program
  - Ghost code not impact the behaviour of the program being verified
  - Ghost definitions are only used at compile time so there is never runtime overhead
    - Ghost code can be explicitly included in the program for debugging purposes
- Ghost entities include packages, subprograms, types and variables
  - Ghost entities can only be used by contracts and assertion pragmas
  - Ghost variables cannot be assigned to program variables

SPARK Feature: Ghost Code Example

Ghost code looks like regular code but it does not become part of the final executable

```
Log      : Integer_Array with Ghost;
Log_Size : Natural with Ghost;

procedure Add_To_Total (Incr : in Integer) with
  Post => Log_Size = Log_Size'Old + 1
    and Log = Log'Old'Update (Log_Size => Incr);

procedure Add_To_Total (Incr : in Integer) is
begin
   Total := Total + Incr;
   Log_Size := Log_Size + 1;
   Log (Log_Size) := Incr;
end Add_To_Total;
```

Used only for formal verification

Issues Using Ada and SPARK

- **Interfacing with other languages like C/C++**
  - **Import and export of functions and procedures is supported**
- **Compiler availability**
  - **Commercial and open source options available**
  - **Native targets include x86, ARM, RISC-V, PowerPC, SPARC**
- **Compiler support for target**
  - **GNAT Common Code Generator (CCG) compiler generates C source code**

New Ada and SPARK Supporters

NVIDIA https://www.adacore.com/press/adacore-enhances-security-critical-firmware-with-nvidia

Denso https://www.adacore.com/press/denso-spark-automotive-research

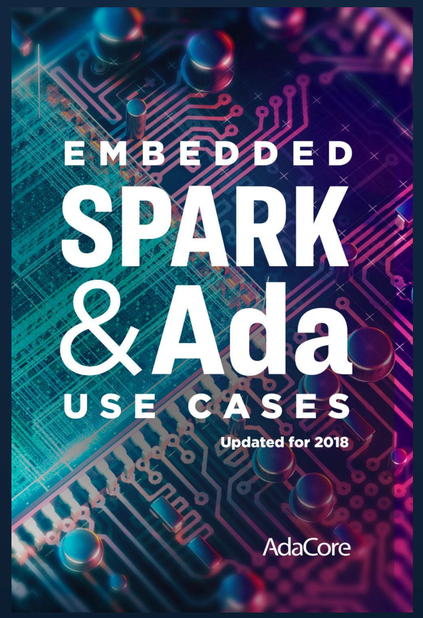
LASP https://www.adacore.com/press/lasp-selects-gnat-pro-for-clarreo

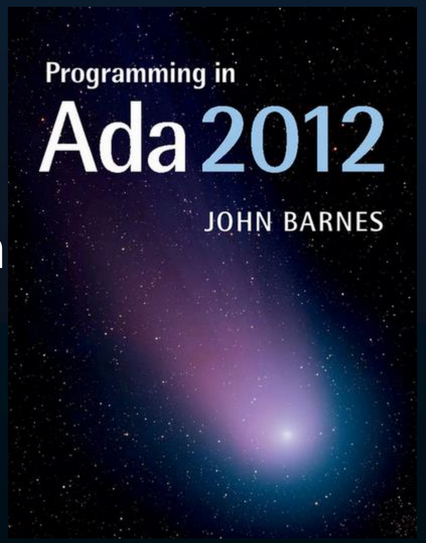
RealHeart https://www.adacore.com/press/scandinavian-real-heart-selects-adacore-embedded-software-development-platform-for-revolutionary-artificial-heart

French Security Agency https://blog.adacore.com/security-agency-uses-spark-for-secure-usb-key

New Ada and SPARK Resources

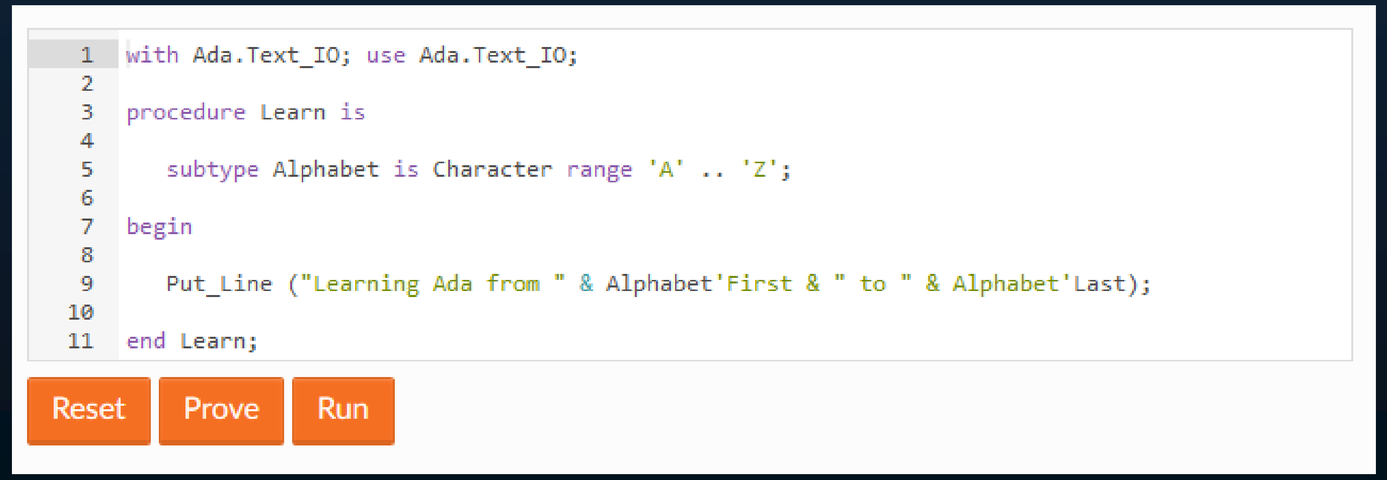
- **Books**
  - **Programming in Ada 2012 by John Barnes**
  - **https://www.adacore.com/books**
- **Learn about Ada and SPARK on the web**
  - **https://learn.adacore.com**
  - **https://www.adacore.com/resources**
- **Open source x86 and ARM Tools: GNAT Community Edition**
  - **https://www.adacore.com/download**





LEARN.adacore.com

- **Introductory courses for Ada and SPARK**
- **Interactive compiler windows**



TCF19

- Open to small teams and individuals
- Many projects based on bare-board platforms
- Results are online for 2019 at https://www.makewithada.org/

Takeaway

- Ada and SPARK are ideal for embedded programming even on microcontrollers
- Ada continues to give more power to the programmer
- SPARK takes Ada a big step further via static verification
- Significant productivity and reliability gains are possible
- The risks are worth the gains

Questions?

Thanks to Adacore's Patrick Rogers for use of slides from his Adacore Days 2018 presentation