AdaCore

RCLAda: the Ada client library for ROS2

CUD

A.R. Mosteo, D. Tardioli (CUD)

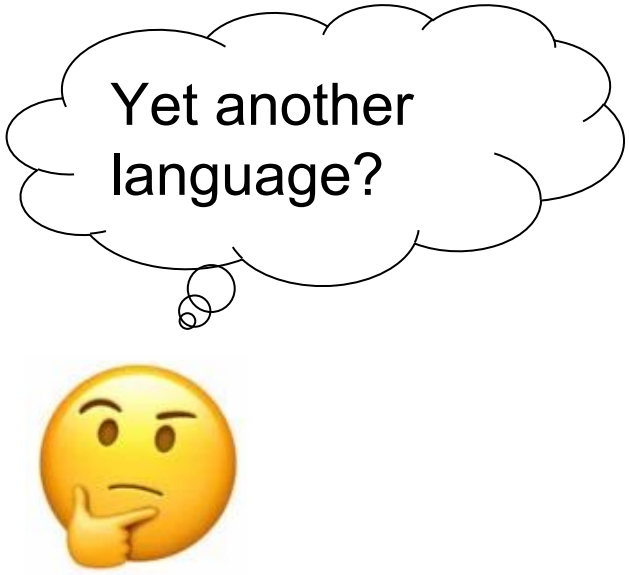
F. Chouteau, Y. Moy (Adacore)

2018-sep-30

Centro Universitario de la Defensa Zaragoza

CONTENTS

- About us
- Motivation
  - Why ROS2
  - Why Ada
- Architecture
  - Packages
  - CMake helpers
- Examples



Robotics, Perception and Real-Time group - RoPeRT

University of Zaragoza

Engineering Research Institute of Aragon

Research group

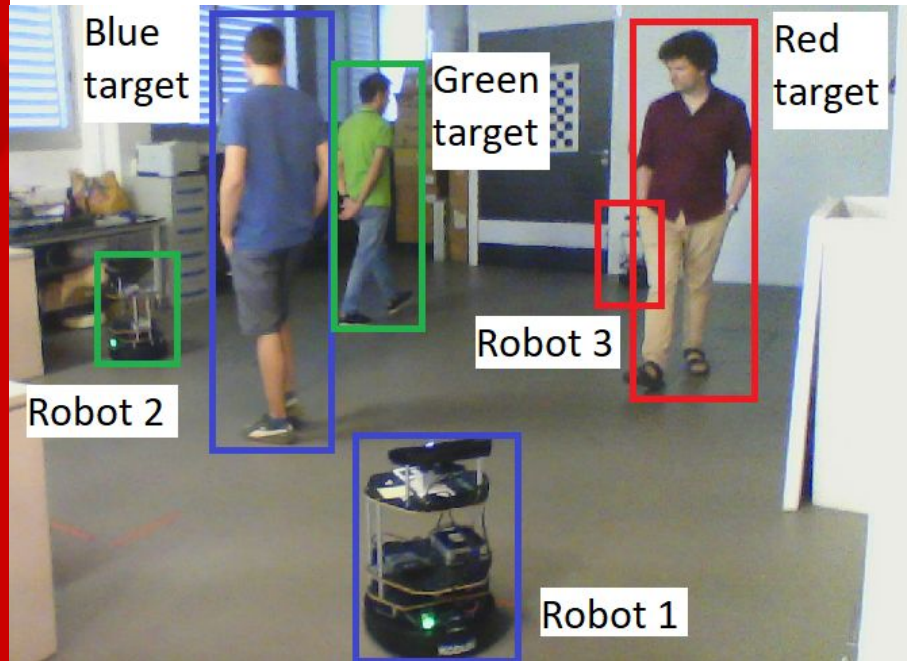
Optimal distributed coordination

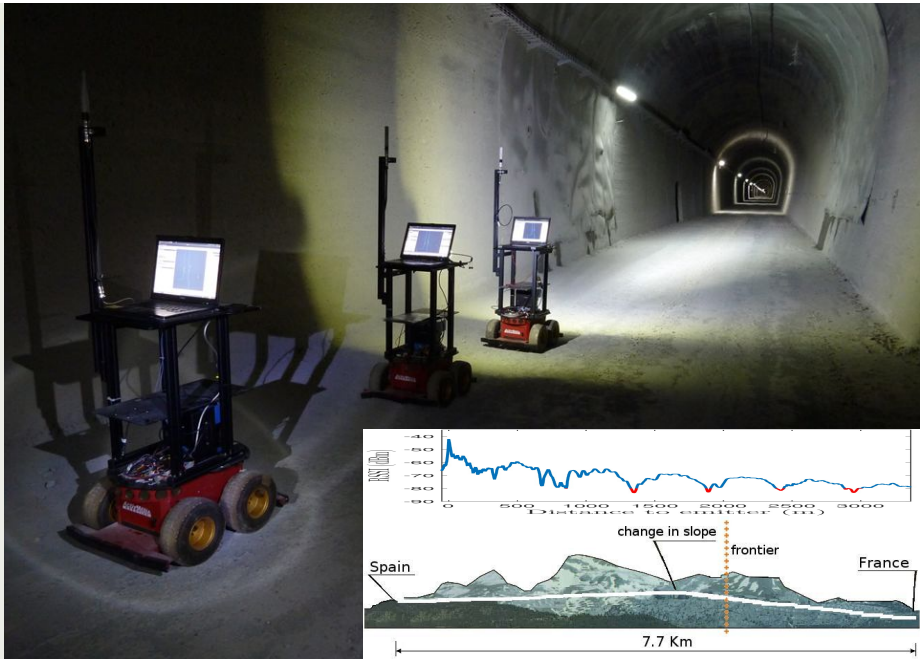
Real-time multi-hop communication

Underground drone reconnaissance

ROBOTICS
UNIVERSIDAD DE ZARAGOZA

http://robots.unizar.es/





Industrial partner

AdaCore

www.adacore.com

https://www.adacore.com/industries

GNU NYU Ada Translator / FSF GNAT-GCC / SPARK

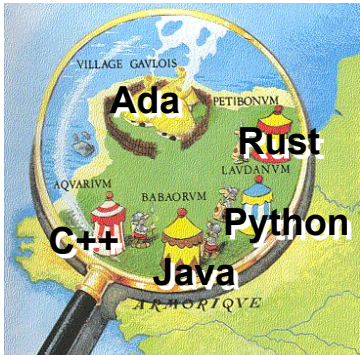
In a nutshell

We can now write ROS2 nodes in Ada 2012

Ada coin author: Steven Craeynest

Ada history

- 1975: Working group DoD / UK MoD
  - STRAWMAN first discussions
- 1978: STEELMAN requirements
  - Embedded, reliability, maintainability, efficiency requirements
  - No suitable existing candidate
- 1979: Green proposal by Jean Ichbiah of Honeywell Bull
  - Renamed to Ada
- 1983: standard ANSI/MIL-STD-1815A (Ada 83)
- 1991-1997: DoD mandate years
  - From 450 to 37 languages by 1998
- Today: niche in many critical industries



- Structured
  - Separate specifications
- Strongly, statically typed
  - Named types (even pointers)
- Imperative (Pascal-like)
  - Object oriented, optionally
- High-level concurrency
  - Tasks, Rendezvous, Monitors
- Design-by-contract
  - Pre-, post-conditions
  - Type, loop invariants
- Comparable in purpose to C++
  - Emphasis in
    - Maintainability
    - Correctness
    - Early error detection

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Hello is
begin
  Put_Line ("Hello, ROSCon!");
end Hello;
```

```
type Speeds  is new Float;
type Lengths is new Float;

spd : Speeds  := 0.0;
len : Lengths := spd; -- Bzzzt
```

```
procedure Inc (X : in out Integer)
  with Pre => X < Integer'Last;

type Prime is new Positive
   with Predicate =>
     (for all D in 2 .. Prime / 2 =>
         Prime mod D /= 0);
```

Ada numeric types

```
type Robot_ID is new Natural;                -- Type compatibility is by name
type Task_ID  is new Natural;                --
type Distance is range 0 .. 1_000_000_000; -- Explicit bounds

type Coordinate is range -180.0 .. 180.0;   -- Floating point with range

type Probability is digits 5                 -- Floating point with
                    range 0.0 .. 1.0;        -- minimum guaranteed precision

type Laser_Readings is delta 10.0 / 2**8     -- Binary fixed point
                       range 0.0 .. 10.0;

type Euros is delta 0.001 digits 12;         -- Decimal fixed point

type Weekdays is (Monday, Tuesday, Wednesday, Thursday, Friday);
type Escaped_Robot_Counter is array (Weekdays) of Natural
      with Default_Component_Value => 0;   -- Arbitrarily indexed arrays
```

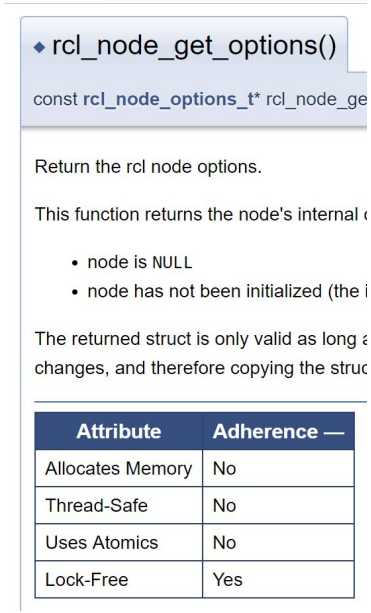
- Ada Rapporteur Group
  - Receives suggestions, requests, comments
  - Prioritizes “not” doable in current Ada
- Ada Reference Manual (ARM)
  - AARM: Annotated ARM for experts, compiler writers
  - All are ISO standards
- Ada Conformity Assessment Test Suite (ACATS)

Feature	Ada 83	Ada 95	Ada 2005	Ada 2012	Ada 202X
Packages	✓	✓	✓	✓	✓
Generics	✓	✓	✓	✓	✓
Derived ADTs	✓	✓	✓	✓	✓
Object orientation (tagged types)		✓	✓	✓	✓
Multiple inheritance (abstract interfaces)			✓	✓	✓
Design by Contract				✓	✓
Numeric types (fixed, floating, decimal, custom)	✓	✓	✓	✓	✓
Tasks	✓	✓	✓	✓	✓
Monitors		✓	✓	✓	✓
Real-time systems annex		✓	✓	✓	✓
Ravenscar profile			✓	✓	✓
Multiprocessor affinities, Multiprocessor Ravenscar				✓	✓
Parallel constructs (blocks, loops)					✓

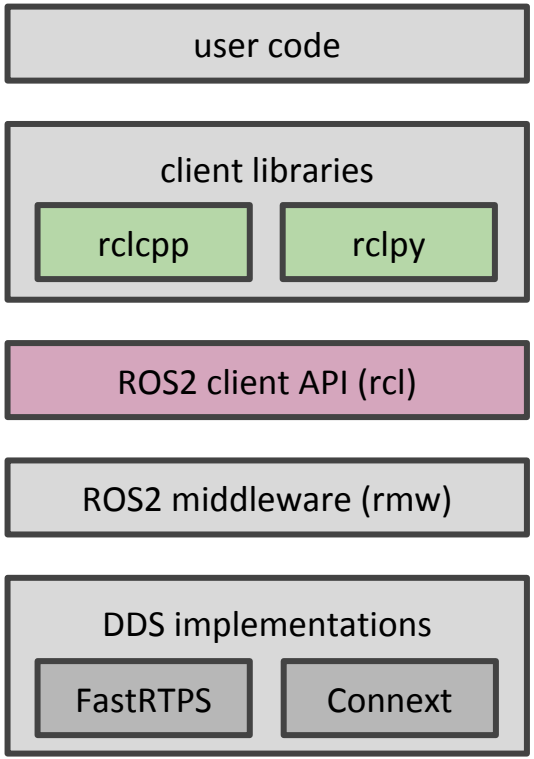
Source: https://www.adacore.com/about-ada

ROS2

- Emphasis on
  - Embedded
  - Real-time
- Traditional strong points of Ada
  - Annex C: systems programming
    - Interrupts, atomics, volatiles, task identification
  - Annex D: (hard) real time
    - Priorities, schedulers, monotonic clock, RAVENSCAR
  - Ada Reference Manual (ISO/IEC 8652:2012)
  - Industries requiring certification (aero but… autonomous robots?)
- Related: SPARK Ada subset for formal proofs on code

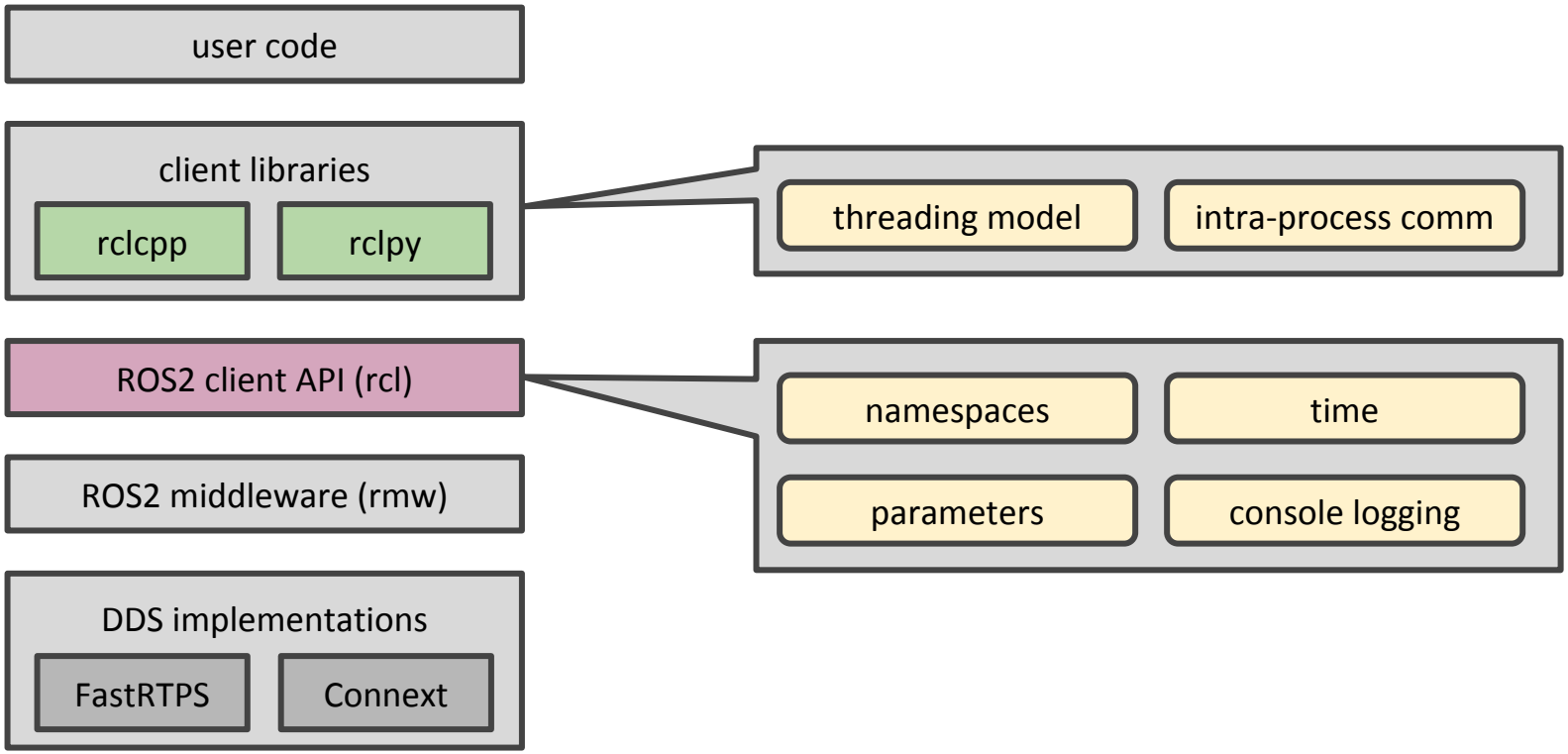


ROS2 client support

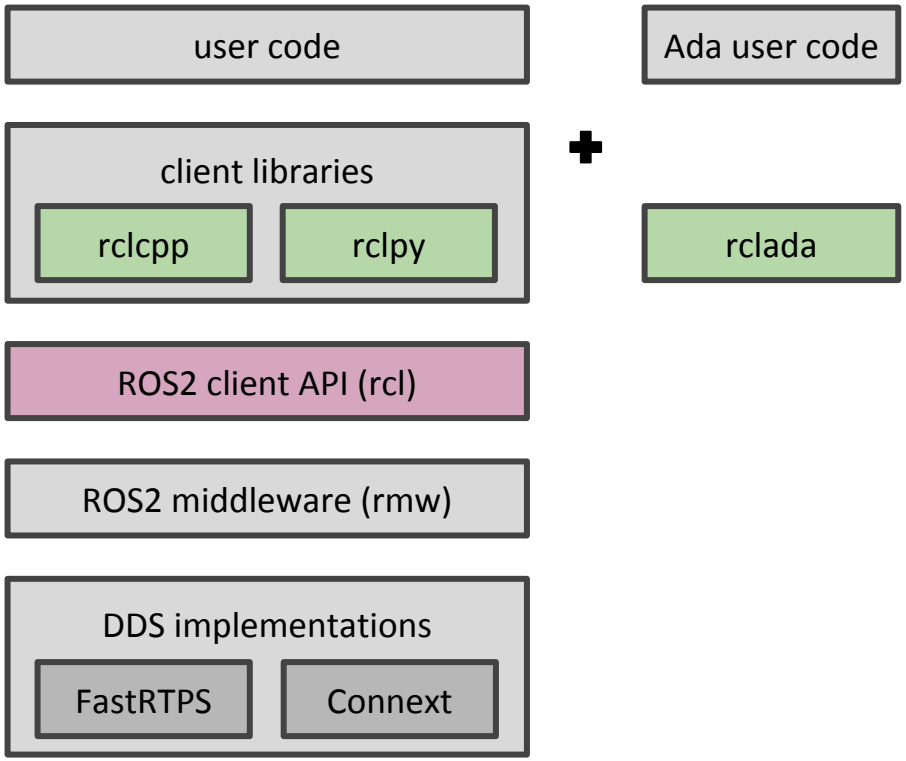


Original diagram by Deanna Hood, William Woodall: https://goo.gl/oCHR7H

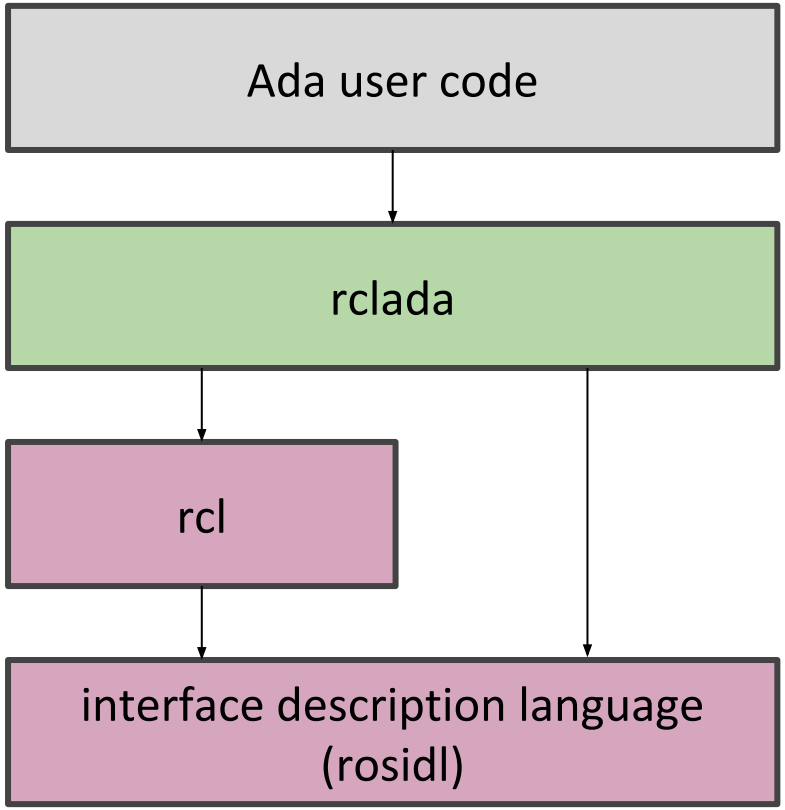
ROS2 client support



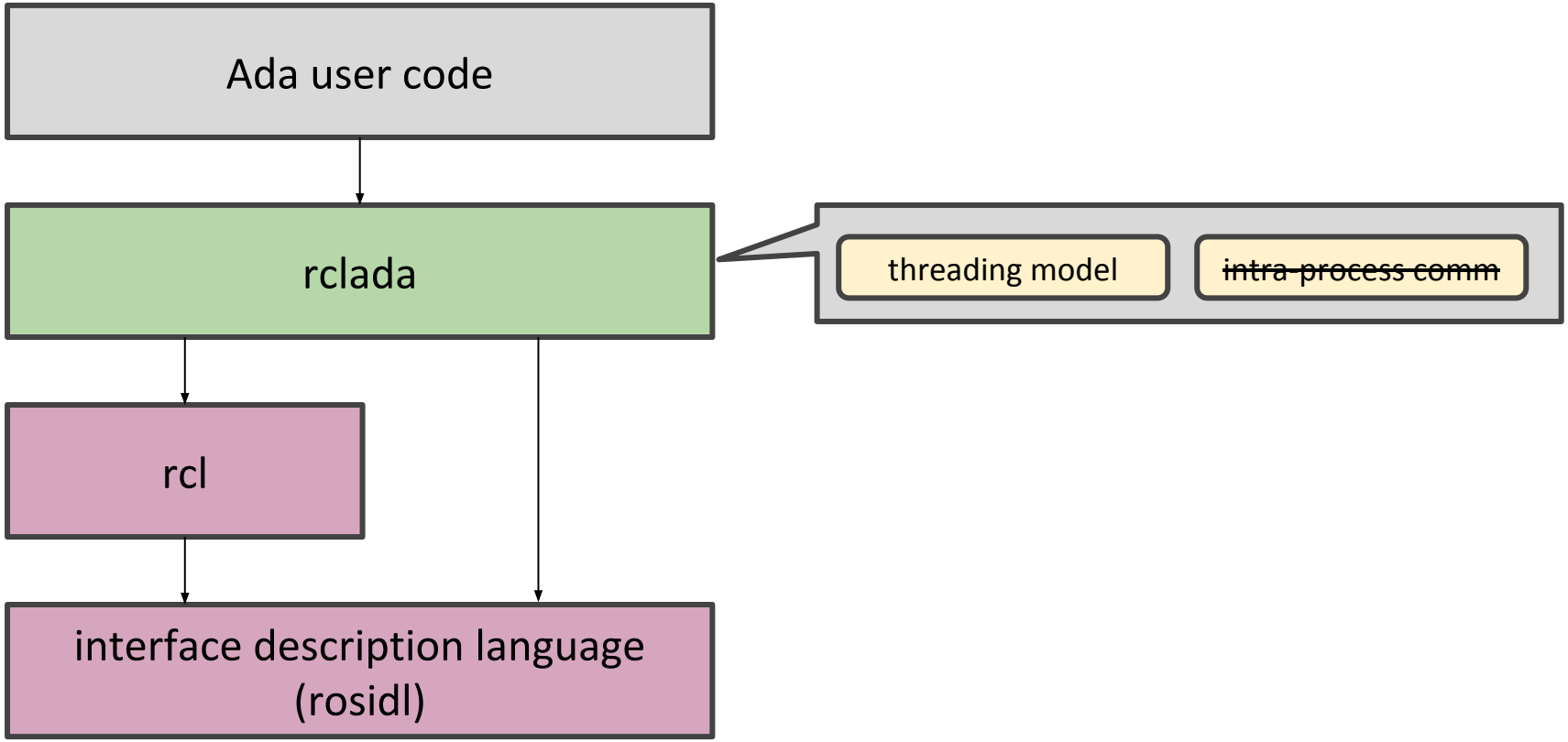
ROS2 client support



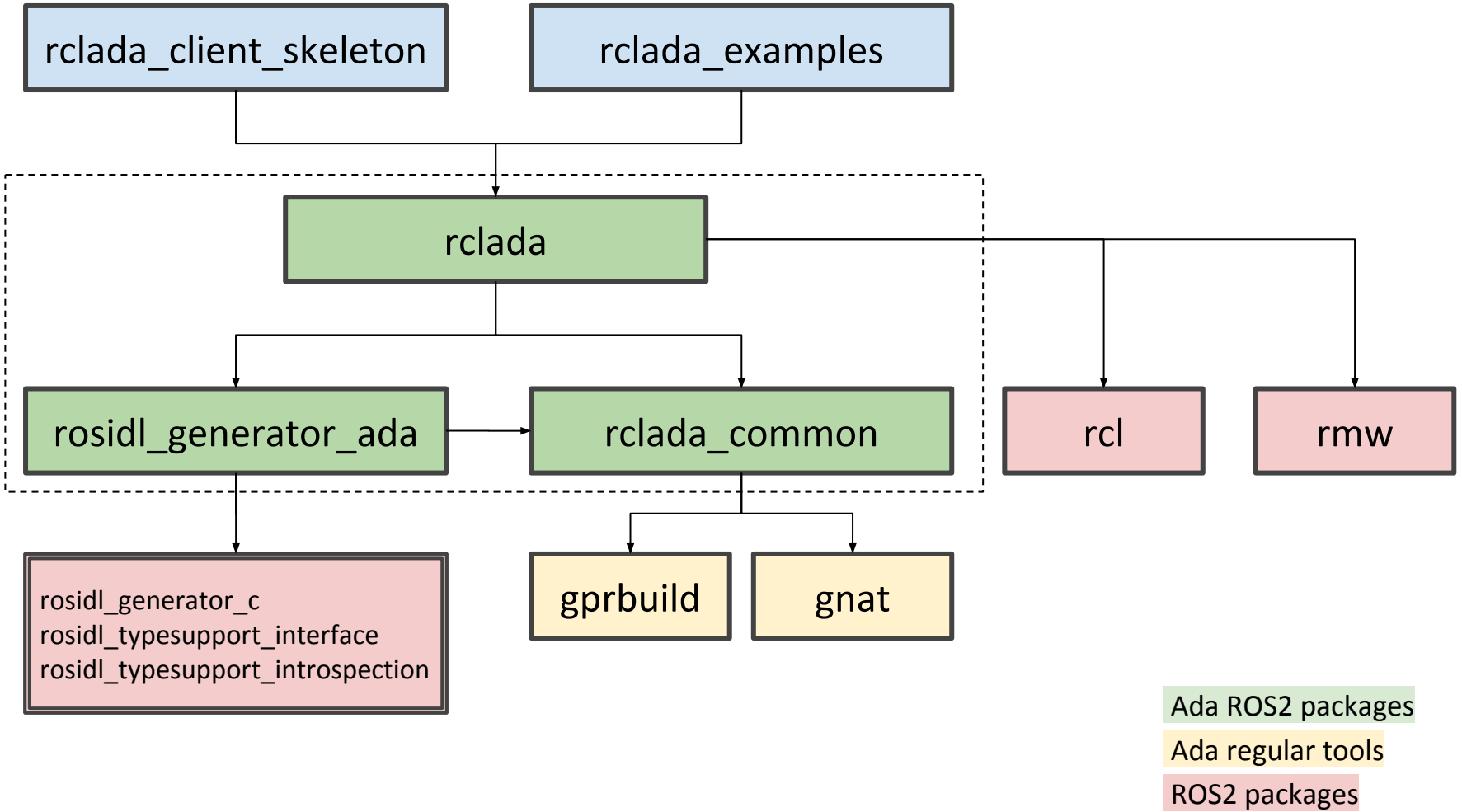
ROS2 architecture II



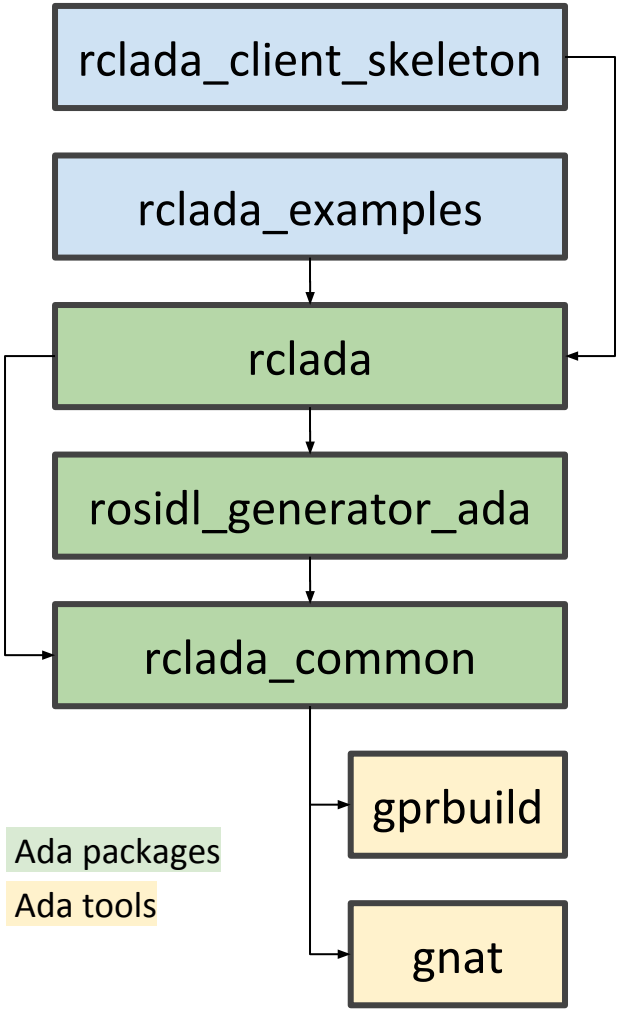
ROS2 architecture II



Actual packages



Ada ROS2 packages



Empty quickstart package

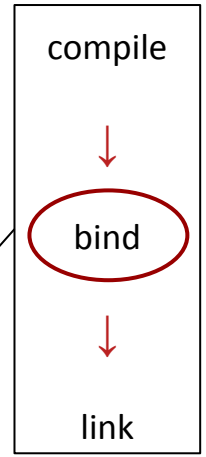
Usual suspects (talker, listener, add_two_ints, …)

Client facilities (nodes, topics, services, …) & self-tests

Message support

CMake functions

Ada build system
(ARM ch.10)



Ada compiler (gnat-gcc) and tools

Ada binder ≈ CMake

- **Check unit consistency**
  - *No outdated *.o*
- **Find safe elaboration order**
  - *No “init order fiasco”*
- **Generate `main` procedure**
  - *Perform initialization*
  - *Call user’s main*
  - *Perform cleanup*

Ada packages

Specification (*.ads)

```
package RCL.Logging is

   procedure Initialize;

   -- public methods

private

   -- protected methods
   -- 1-pass compiler needed info

end RCL.Logging;
```

Implementation (*.adb)

```
package body RCL.Logging is

   procedure Initialize is
   begin
      -- do whatever has to be done
   end Initialize;

   -- private methods

end RCL.Logging;
```

Main procedure (*.adb)

```
with RCL.Logging;

procedure RCL.Talker is
begin
   Logging.Initialize;
end RCL.Talker;
```

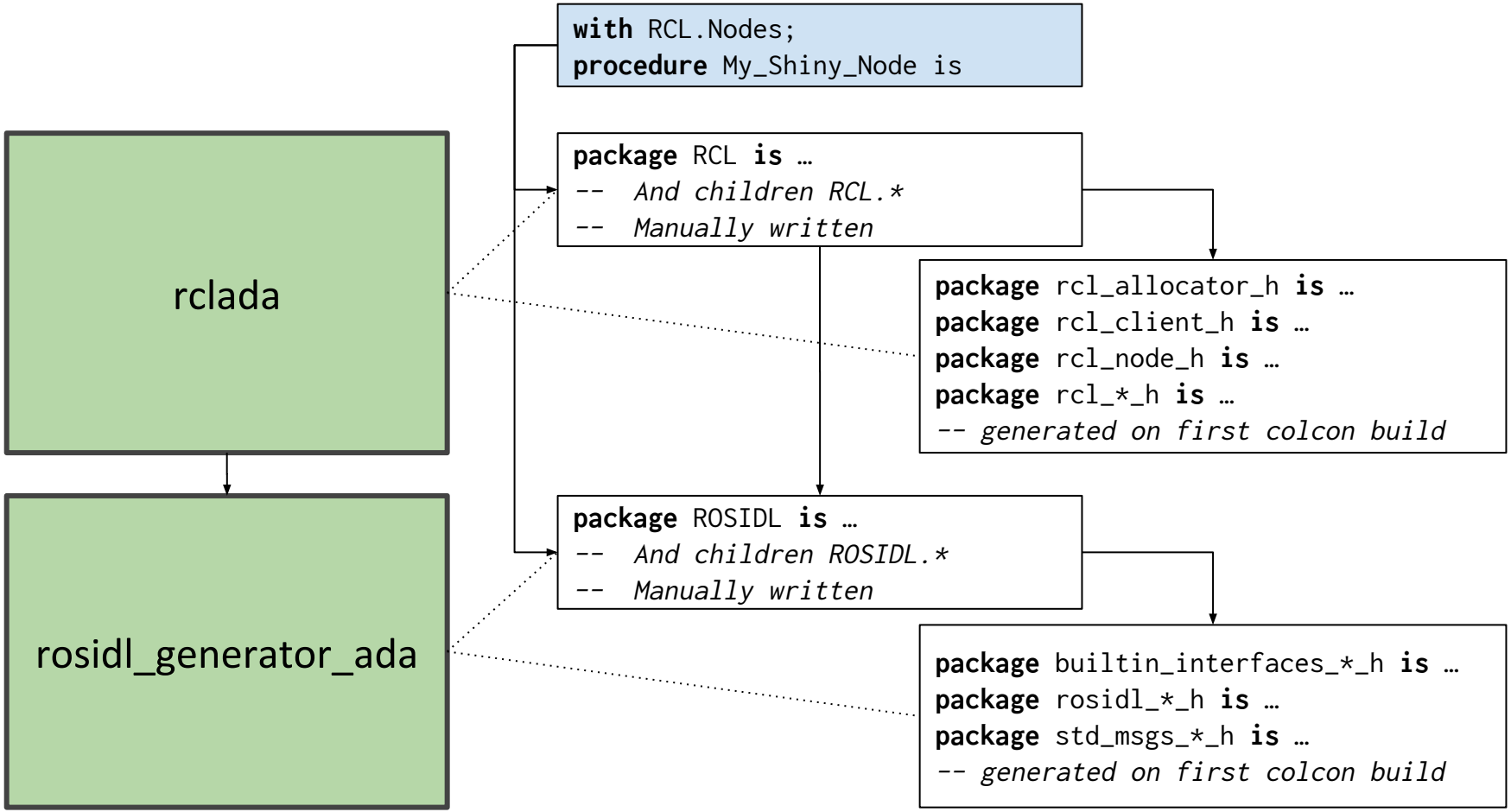
- Writing bindings:
  - Manual writing
    - ☺ No need to be exhaustive
    - ☺ High quality (thick binding)
    - ☹ More effort
    - ☹ May become de-sync'd
  - Automated generation
    - ☺ "Less" work
    - ☺ Completeness
    - ☺ Assured consistency
    - ☹ Lower quality (thin binding)
    - ☹ Might not compile
- Ada/GNAT support:
  - Annex B: interface to other languages
    - C/C++, Fortran, Cobol
  - `gcc -fdump-ada-spec file.h`

```
/* C prototype */
int initialize(options_t *opts,
               char *argv[]);
```

```
-- Ada automatic binding
function Initialize
   (opts : access Options_T;
    argv : System.Address)
    return Interfaces.C.int
    with Import, Convention => C;
```

```
-- Ada manual binding
type Arg_Array is
  array (Natural range <>) of aliased
  Interfaces.C.Strings.Chars_Ptr
  with Convention => C;

function Initialize
   (opts : in out Options_T;
    argv :        Arg_Array)
    return Interfaces.C.int
    with Import, Convention => C;
```

RCLAda: leverage *colcon* for best of both worlds



Ada packages

Completion level

rclada

- Main features:
  - `RCL.Node` : Complete ■
  - `RCL.Publisher` : Complete ■
  - `RCL.Subscription` : Complete ■
  - `RCL.Client` : Complete ■
  - `RCL.Service` : Complete ■
- Support:
  - `RCL.Allocators` : Complete ■
  - `RCL.Calendar` : Complete ■
  - `RCL.Executors` : Complete ■
  - `RCL.Graph` : Complete ■
  - `RCL.Options` : Partial ■
  - `RCL.Timer` : Complete ■
  - `RCL.Wait` : Complete ■

rosidl_generator_ada

- Messages:
  - `ROSIDL.Dynamic`: Complete ■
  - `ROSIDL.Typesupport`: Complete ■
- Dynamic access (through introspection):
  - Typesupport: Complete ■
  - Simple types: Complete ■
  - Nested types: Complete ■
  - Array types: Complete ■
  - Matrix types: Complete ■
- Static access (through generated types):
  - Typesupport: Pending ■
  - Simple types: Pending ■
  - Nested types: Pending ■
  - Array types: Pending ■
  - Matrix types: Pending ■

Message fields

```
declare
   Support : ROSIDL.Typesupport.Message_Support :=
             ROSIDL.Typesupport.Get_Message_Support
               (Pkg_Name, Msg_Type);
   Msg : ROSIDL.Dynamic.Message := Init (Support);
begin
   Msg (“valid”).As_Bool := True;
   Msg (“X”).As_Float32  := 1.0;
   --  Individual values

   Msg (“Values”).As_Array (42).As_Int8 := 0;
   --  Array indexing

   Msg (“Image”).As_Matrix ((100, 50, 1)).As_Int8 := 0;
   --  Matrix indexing
end;
```

rosidl_generator_ada

Obtain message type

Reference to fields
- No data copy

1D vector indexing
- Bounds checked

Matrix indexing
- Tuple of indices

Support for colcon_cmake

rclada_common

- `ada_begin_package()`
- `ada_end_package()`

  Needed to propagate Ada information through ROS2 packages

- `ada_add_executables(TARGET SRCDIR DSTDIR EXECUTABLES)`

  Declares an Ada executable to be built and exported (tab completion)
- `ada_add_library(TARGET SRCDIR GPRFILE)`

  Declares an Ada library project to be built and exported to other Ada packages

- `ada_import_msgs(PKG_NAME)`

  Generates bindings to the typesupport handle functions

  Could disappear once RCLAda is integrated in build farm

- `ada_generate_binding(TARGET SRCDIR GPRFILE INCLUDE)`

  Invokes the binding generator in the context of an Ada project

Talker

```
procedure Talker is
   Support : constant ROSIDL.Typesupport.Message_Support :=
               ROSIDL.Typesupport.Get_Message_Support ("std_msgs", "String");
   Node    : Nodes.Node           := Nodes.Init   (Utils.Command_Name);
   Pub     : Publishers.Publisher := Node.Publish (Support, "/chatter");

   task Publisher;
   task body Publisher is
      Count  : Positive               := 1;
      Period : constant Duration      := 1.0;
      Next   : Calendar.Time          := Calendar.Clock;
      Msg    : ROSIDL.Dynamic.Message := ROSIDL.Dynamic.Init (Support);
   begin
      loop
         Msg ("data").Set_String ("Hello World:" & Count'Img);
         delay until Next;
         Pub.Publish (Msg);
         Counter := Count + 1;
         Next    := Next  + Period; -- Next := @ + Period; -- in Ada 202x
       end loop;
   end Publisher;

begin
   Node.Spin (Until => Forever);
end Talker;
```

Dynamic handle retrieval

Node initialization in the stack

Topic creation

An Ada task without sync entries

Duration is a built-in Ada type

Message allocation

Message fields are
- indexed by name
- type checked
- bounds checked

Delay without drift

Spin forever (named parameter)

(note: all Ada tasks have masters that await their completion)

Listener

```
procedure Listener is

   procedure Callback (Node : in out Nodes.Node'Class;
                       Msg  : in out ROSIDL.Dynamic.Message;
                       Info :        ROSIDL.Message_Info) is
   begin
      Logging.Info ("Got chatter: '" & Msg ("data").Get_String & "'");
   end Callback;

   Node : Nodes.Node := Nodes.Init ("listener");

begin
   Node.Subscribe
     (ROSIDL.Typesupport.Get_Message_Support ("std_msgs", "String"),
      "/chatter",
      Callback'Access);

   Node.Spin (Until => Forever);
end Listener;
```

Callback definition

Standard ROS2 Logging

Ada String (not null-terminated)

Register callback
- Using procedure address

LISTENER

Services

```
procedure Server is
   -- Omitted declarations

   procedure Adder
     (Node : in out Nodes.Node'Class;
      Req  :        ROSIDL.Dynamic.Message;
      Resp : in out ROSIDL.Dynamic.Message)
   is
      A : constant ROSIDL.Int64 := Req ("a").As_Int64;
      B : constant ROSIDL.Int64 := Req ("b").As_Int64;
   begin
      Resp ("sum").As_Int64 := A + B;
   end Adder;

begin
   Node.Serve
     (ROSIDL.Typesupport.Get_Service_Support
        ("example_interfaces", "AddTwoInts"),
      "add_two_ints",
      Adder'Access);
end Server;
```

SERVER

```
procedure Client is -- Synchronous version
   -- Omitted declarations

   Request : ROSIDL.Dynamic.Message := … ;

begin
   Request ("a").As_Int64 := 2;
   Request ("b").As_Int64 := 3;
   declare
      Response : constant ROSIDL.Dynamic.Message :=
                   Node.Client_Call (Support,
                                     "add_two_ints",
                                     Request);
   begin
       Logging.Info ("Got answer:" &
                     Response ("sum").As_Int64.Image);
   end;
end Client;
```

Blocking call

CLIENT

Everything on the stack: Ada indefinite types

```
declare
   type Int_Array is array (Positive range <>)
                          of Integer;

   Arr   : Int_Array (1 .. 100);
   Hello : constant String := “Hello”;

   Other_Arr : Int_Array (1 .. Get_Elsewhere);
begin
   -- Variable stack use so measure it or limit it!
end;

declare
   type Unconstrained (Length : Natural) is record
      Name : String (1 .. Length);
   end record;

   U1 : constant Unconstrained := Get_Unconstrained;
   U2 :          Unconstrained (10);
begin
```

Indefinite type *(unknown size at compile time)*
- but definite values! *(known size at runtime)*

Constrained by declaration with static size

Constrained by initialization with static size

Constrained by declaration with unknown size
- at compile time

Constrained by initialization with unknown size

Constrained by declaration with static size

Indefinite concurrent executor type

```
package RCL.Executors.Concurrent is

   type Runner_Pool is array (Positive range <>) of Runner;
   -- Runner task type declaration omitted

   type Executor (Max_Nodes  : Count_Type :=
                     Default_Nodes_Per_Executor;
                  Queue_Size : Count_Type :=
                     Count_Type (System.Multiprocessors.Number_Of_CPUs) * 32;
                  Threads    : Positive :=
                     Positive (System.Multiprocessors.Number_Of_CPUs);
                  Priority   : System.Priority :=
                     System.Max_Priority) is
   new Executors.Executor (Max_Nodes) with
      record
         Pool  : Runner_Pool  (1 .. Threads);
         Queue : Queues.Queue (Capacity => Queue_Size,
                               Ceiling  => Priority);
         Started : Boolean := False;
      end record;

end RCL.Executors.Concurrent;
```

Parent abstract type in RCL.Executors

Task pool type

Executor type with discriminants
- Nodes supported
- Queue size
- Threads
- Priority

System.* defined in ARM

OO derivation syntax

Members constrained by discriminants
- Standard Ada bounded queues
- All Ada bounded containers are stack based

See rclada_test_multicore.adb
- One producer
- Pooled consumers

ROS2 allocators ⇔ Ada storage pools

- Ada defines Storage_Pool type for different:
  - memory areas (typical in some small boards) (associated to pointer types)
  - allocation policies (including user-defined)
- ROS2 allocators mapped into Ada storage pools
  - transparent use in Ada programs
  - immediate testing of RCLAda & ROS2 use of allocators via GNAT.Debug_Pools

```
$ rclada_test_allocators 1
Total allocated bytes:              2335
Total logically deallocated bytes:  2335
Total physically deallocated bytes: 0
Current Water Mark:                 0
High Water Mark:                    415

$ rclada_test_allocators 4
Total allocated bytes:              8095
Total logically deallocated bytes:  8095
Total physically deallocated bytes: 0
Current Water Mark:                 0
High Water Mark:                    415
```

```
type Int_Ptr is access Integer -- named pointer type
  with Storage_Pool => Debug_Pool;

type Node_Access is access all RCL.Nodes.Node'Class
  with Storage_Size => 0;      -- No heap allocations

pragma No_Allocators;
pragma No_Implicit_Heap_Allocations;
pragma No_Standard_Allocators_After_Elaboration;
pragma No_Standard_Storage_Pools;
--  See Restrictions in GNAT manual for many more
```

Allocator details

```
typedef struct rcutils_allocator_t
{
  void * (*allocate)(size_t size,
                     void * state);

  void (* deallocate)(void * pointer,
                      void * state);

  void * (*reallocate)(void * pointer,
                       size_t size,
                       void * state);

  void * (*zero_allocate)(size_t number_of_elements,
                          size_t size_of_element,
                          void * state);
                          void * state;
} rcutils_allocator_t;
```

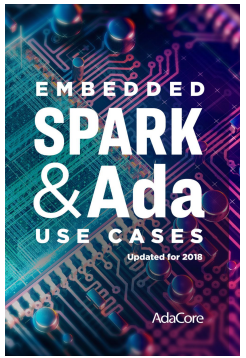
```
package System.Storage_Pools is

   type Root_Storage_Pool is tagged private;

   procedure Allocate
     (Pool                     : in out Root_Storage_Pool;
      Storage_Address          :    out Address;
      Size_In_Storage_Elements : in     Storage_Count;
      Alignment                : in     Storage_Count)
   is abstract;

   procedure Deallocate
     (Pool                     : in out Root_Storage_Pool;
      Storage_Address          : in     Address;
      Size_In_Storage_Elements : in     Storage_Count;
      Alignment                : in     Storage_Count)
   is abstract;
```

```
Pool  : aliased GNAT.Debug_Pools.Debug_Pool;                    -- Ada pool, compiler provided
Alloc : aliased RCL.Allocators.Allocator (Pool'Access);         -- ROS2 allocator, wrapping Ada pool
Node  :         RCL.Node := Node.Init
                            (Options => (Allocator => Alloc'Access)); -- Set node allocator
```

SPARK highlights

- SPARK
  - Subset of Ada
    - Same compiler
    - Extra tools for verification/proofs
  - Historically: special comments about code
  - Since Ada 2012/SPARK 2014: Ada contracts
    - Checked by the compiler
- Can prove:
  - Absence of runtime errors (exceptions)
    - Runtime checks can be safely disabled
  - Properties of the program
    - Guided by the programmer
- Similar in some respects to Frama-C
  - But Ada has fewer undefined behaviors
  - The SPARK subset grows with each version
- If interested:
  - Take a free book!
  - Drop by `comp.lang.ada` (yes, NNTP)

```
procedure Increment (X : in out Integer)
  with Global  => null,
       Depends => (X => X),
       Pre     => X < Integer'Last,
       Post    => X = X'Old + 1;
```

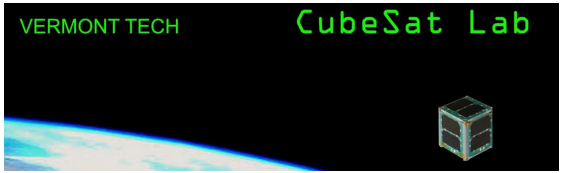



```
type Prime is new Positive
   with Dynamic_Predicate =>
     (for all D in 2 .. Prime / 2 =>
        Prime mod D /= 0);
```

SPARK success stories

CubeSat from Vermont Tech

http://www.cubesatlab.org



- Three years of flight time (2013-2016)
- Others: 1x4month, 2x<1week, 8xUnheard of

IRONSIDES DNS server

`https://ironsides.martincarlisle.com/`

Proven free of (among others):

- Buffer overflows
- Integer overflows
- Information leaks
- Race conditions

SND navigation algorithm

`https://github.com/riveras/spark-navigation`

- IROS 2014 paper on errors in robotic navigation algorithms
- SND reimplemented in SPARK
- Proven without runtime errors
- Possible target to integrate with RCLAda

Tokeneer ID station project (NSA)

`https://www.adacore.com/tokeneer`

Not only SPARK but full development methodology

- Formal language (Z) for specification
- ~10KLOC
- 4 defects since delivery

Distinguishing Features

- No heap allocations (in RCL)
  - Guaranteed by language restrictions & libraries
- Relies on automatic low-level binding
  - Early detection of mismatches on ROS2 API changes
- Language ingrained in safety/HRT culture
  - Enforced safe program initialization / task completion
  - Strong static type system (incl. numerics) (plus predicates)
  - A convenient path to formal verification with SPARK
  - SPARK is compiled with the same Ada toolset
- Strong backwards & cross-platform compatibility

THANKS FOR YOUR ATTENTION


https://github.com/ada-ros/ada4ros2/


amosteo@unizar.es


@mosteobotic

Acknowledgements:
Dirk Thomas
William Woodall
Esteve Fernandez
ROS answers

?

Centro Universitario de la Defensa Zaragoza
cud.unizar.es

Academia General Militar · Ctra. Huesca s/n · 50090 Zaragoza · 976 739 500