Building Ada Development Tools with ASIS-for-GNAT

Sergey Rybin

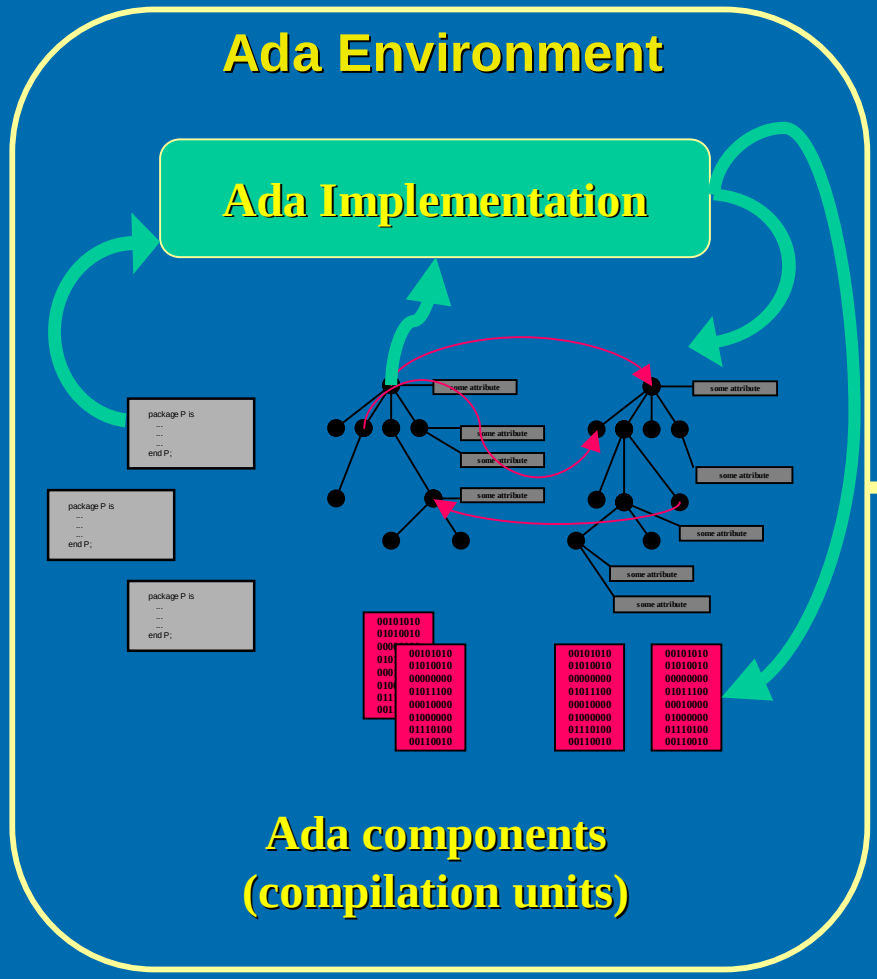
Vasily Fofanov

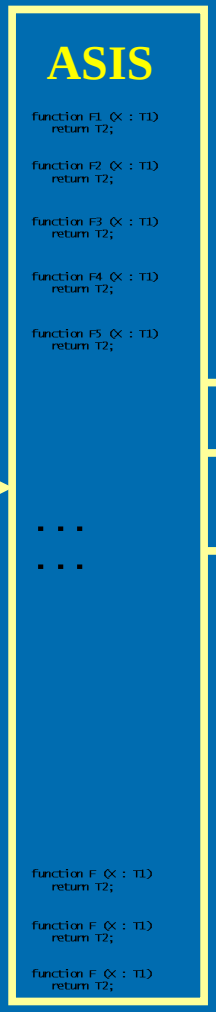
Summary

- ASIS - Ada Semantic Interface Specification
- Building ASIS-based tools
- GNAT
- ASIS-for-GNAT

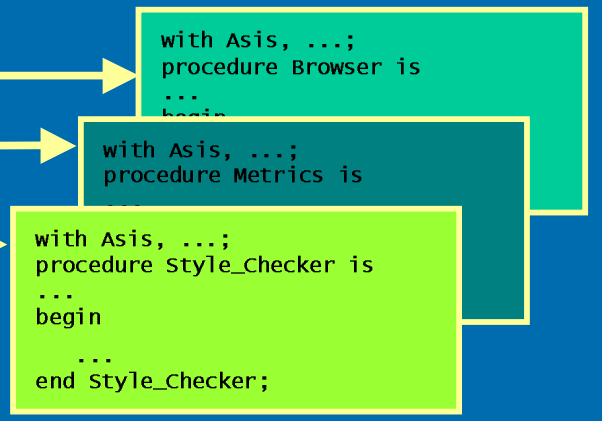
What is ASIS?

✔ **The general idea:**





Code analysis tools



What is ASIS?

- **The official definition (**ISO/IEC 15291:1998(E)**)**
  - **The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment (as defined by ISO/IEC 8652:1995) and any tool requiring information from it. An Ada environment includes valuable semantic and syntactic information. ASIS is an open and published callable interface which gives CASE tool and application developers access to this information. ASIS has been designed to be independent of underlying Ada environment implementations, thus supporting portability of software engineering tools while relieving tool developers from needing to understand the complexities of an Ada environment's proprietary internal representation**

What is ASIS?

- **ASIS API:**
  - **ASIS is written in Ada 95 as a hierarchy of Ada package specifications;**
  - **ASIS defines a set of abstract data types;**
  - **For an ASIS application, ASIS is an Ada library: an application "withes" ASIS packages, uses ASIS-defined types to define data objects, calls ASIS-defined subprograms to operate with these objects;**

What is ASIS?

✔ ASIS API:

ASIS:

```
package Asis is
  type Element is private;
 ...
end ASIS;
package Asis.Declarations is
  function Type_Declaration_View
    (Declaration : in Element) return Element;
 ...
end Asis.Declarations;
```

ASIS application:

```
with Asis, Asis.Declarations;...
procedure ASIS_Tool is
  E1, E2 : Asis.Element;
begin
   ...
   E1 := Type_Declaration_View (E2);
   ...
end ASIS_Tool;
```

How it works?

- ✔ Consider a complete example of developing and using a simple ASIS tool - a Style Checker;
- ✔ Technical requirements for a style checking tool:
  - A set of rules to check includes the following rules:
    - multi-identifier declarations are not allowed;
    - every subprogram body (or body stub) should have a separate spec;
  - Some more rules may be added later;
  - All “user-defined” Ada components in a given environment should be checked;

Style Checker - Tool Developer's needs

- to get a handle to a set of components (or to an environment) to process;
- to make a difference between predefined, implementation-specific and user-defined units;
- to go from a unit into its structure;
- to traverse an Ada code, and to check some specific rules for some specific constructs;
- to provide maintainable and extendable code;

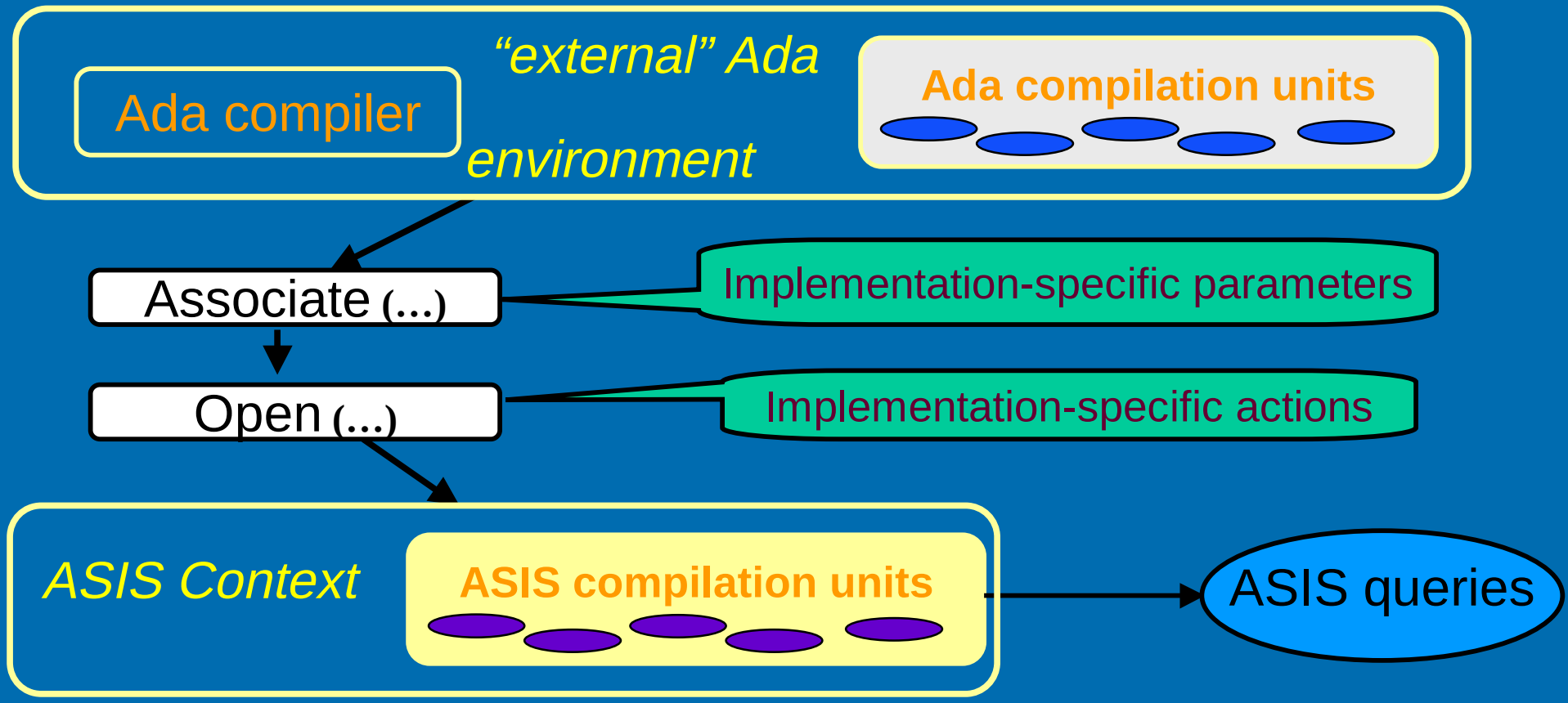
Main ASIS abstractions

- ✔ **Context**
  - **a logical handle for an Ada environment;**
- ✔ **Compilation_Unit**
  - **a logical handle for an Ada compilation unit as well as for a physical object treated as this unit by an underlying Ada implementation;**
- ✔ **Element**
  - **a logical handle for Ada syntax constructs;**

Tool developer's needs in ASIS terms:

- ✔ to define and to process an appropriate ASIS Context;
- ✔ to get ASIS Compilation Units from the Context;
- ✔ to analyze ASIS Compilation Units properties;
- ✔ to use gateways from ASIS Compilation Units into ASIS Elements;
- ✔ to analyze ASIS Elements hierarchy representing in ASIS the given Compilation Unit;

Main ASIS abstractions: Context

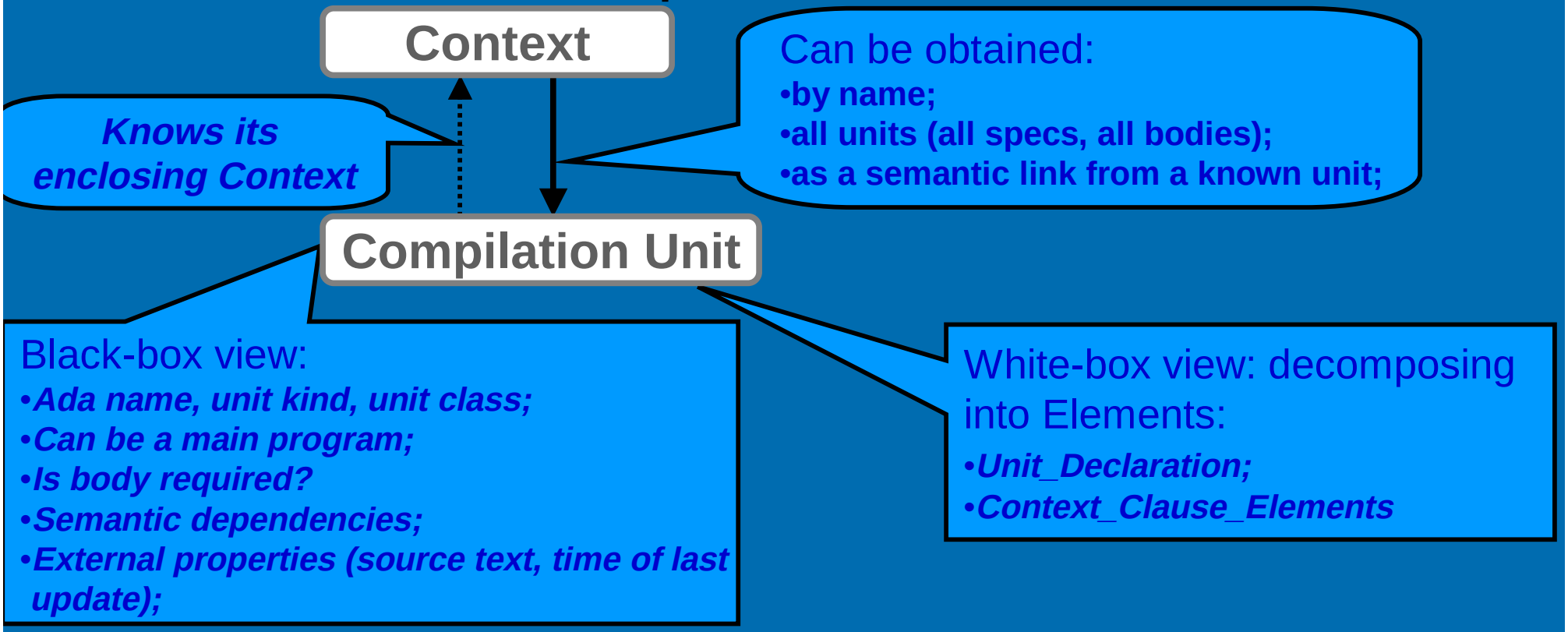


An ASIS Context is a logical handle for an Ada environment. It specifies a set of ASIS Compilation Units accessible through ASIS queries.

Working with the Context

```
with Asis;
with Asis.Ada_Environments;
...
procedure Style_Checker is
   My_Context : Asis.Context;
   My_Context_Parameters : Wide_String := ...;
   ...
begin
   ...
   Asis.Ada_Environments.Associate  (My_Context,
                                     "My_Context_Name",
                                      My_Context_Parameters);
   Asis.Ada_Environments.Open       (My_Context);
   ... -- Working inside opened Context
   Asis.Ada_Environments.Close      (My_Context);
   Asis.Ada_Environments.Dissociate (My_Context);
   ...
end Style_Checker;
```

Main ASIS abstractions: Compilation Unit



An ASIS Compilation Unit provides the black-box view of an Ada compilation unit. It may be decomposed and analyzed as a white-box by means of ASIS Elements.

Working with Compilation Units


```
with Asis.Compilation_Units; use  Asis.Compilation_Units;
...
   Units : Asis.Compilation_Unit_List :=
      Asis.Compilation_Units.Compilation_Units (My_Context);
begin
   ...
   for I in Units'Range loop
      Put ("Processing Unit: ");
      Put (Unit_Full_Name (Units (I)));
      case Unit_Class (Units (I)) is
         when A_Public_Declaration | A_Private_Declaration =>
            Put_Line (" (spec)");
         when A_Separate_Body =>
            Put_Line (" (subunit)");
         when others =>
            Put_Line (" (body)");
      end case;
      ...
   end loop;
   ...
```

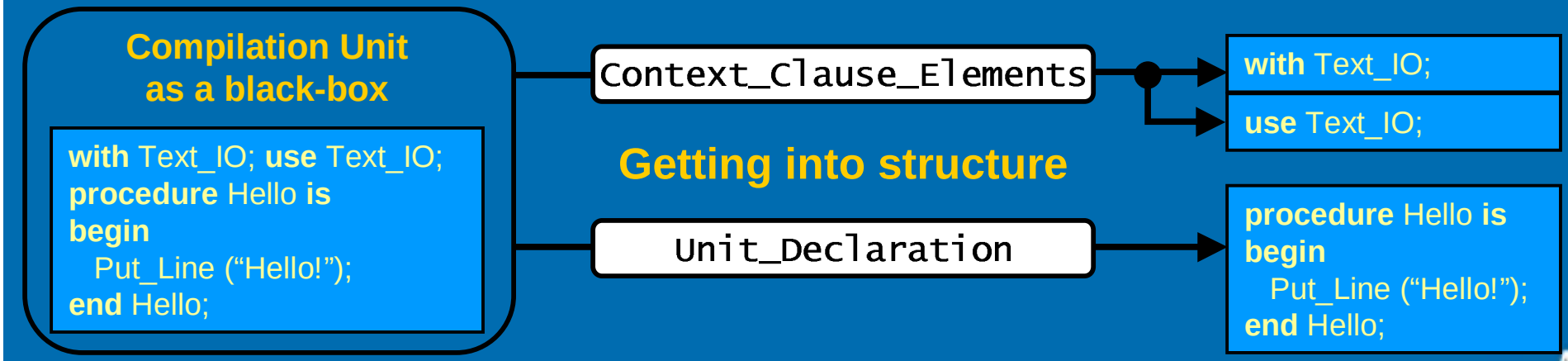
Working with Compilation Units (cont.)

```
...
for I in Units'Range loop
   ...
   case Unit_Origin (Units (I)) is
      when An_Application_Unit =>
         ...
         Process_Unit (Units (I));
         ...
      when A_Predefined_Unit |
           An_Implementation_Unit =>
         null;
   end case;
end loop;
...
```

From Compilation Units into Elements

✔RM 95, 10.1.1, 10.1.2:

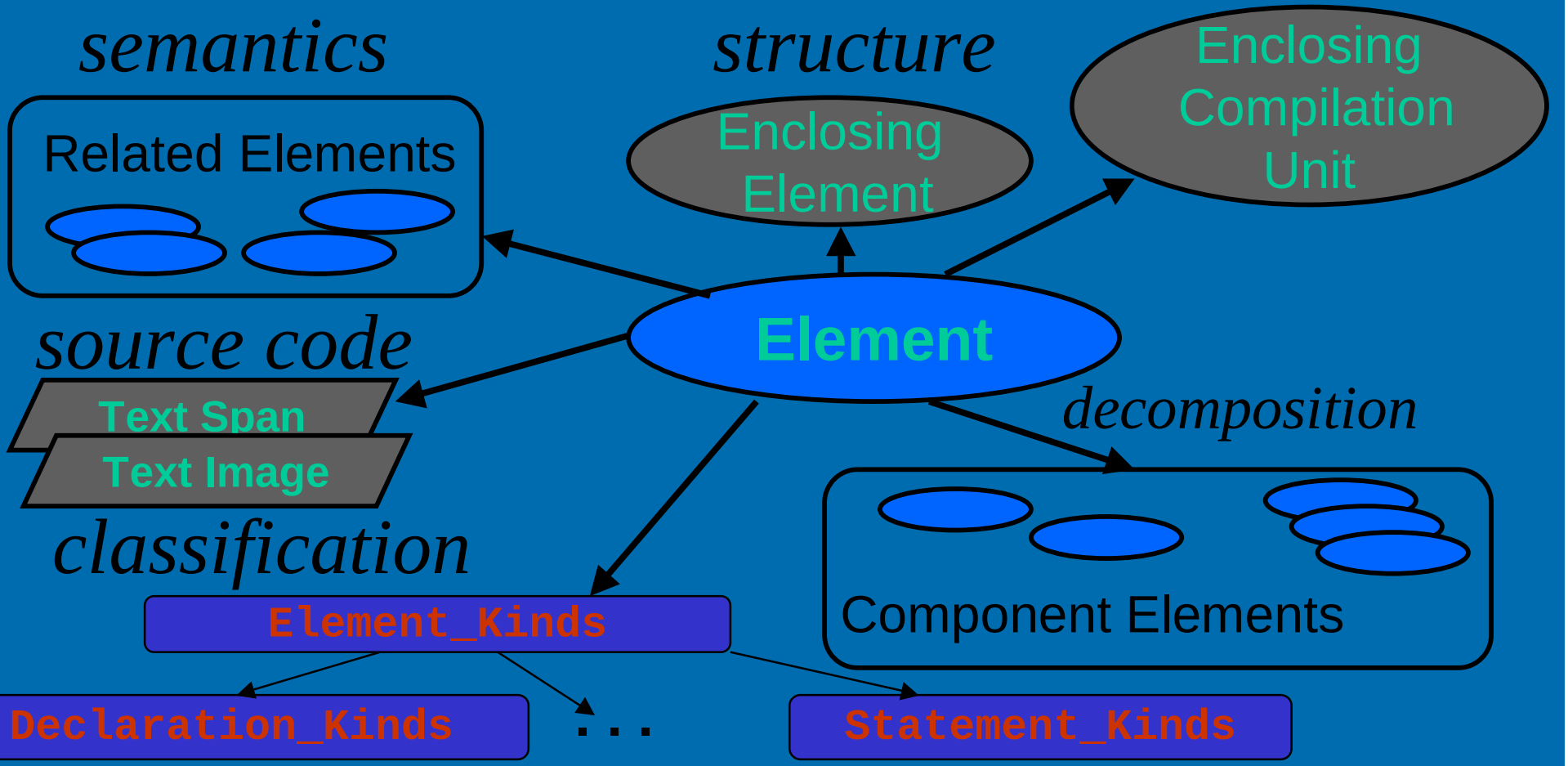
```
compilation_unit ::= context_clause library_item |
                     context_clause subunit
context_clause ::= { context_item }
library_item ::=[private] library_unit_declaration |
                  library_unit_body                 |
                 [private] library_unit_renaming_declaration
```



Using Element Gateways


```
with Asis.Elements; use Asis.Elements;
   ...
procedure Process_Unit (U : Compilation_Unit) is
  Cont_Clause_Elements : constant Element_List :=
    Context_Clause_Elements (Compilation_Unit => U,
                             Include_Pragmas  => True);

  Unit_Decl : Element := Unit_Declaration (U);

begin

  for I in Cont_Clause_Elements'Range loop
    Process_Construct (Cont_Clause_Elements (I));
  end loop;

  Process_Construct (Unit_Decl);

end Process_Unit;
```

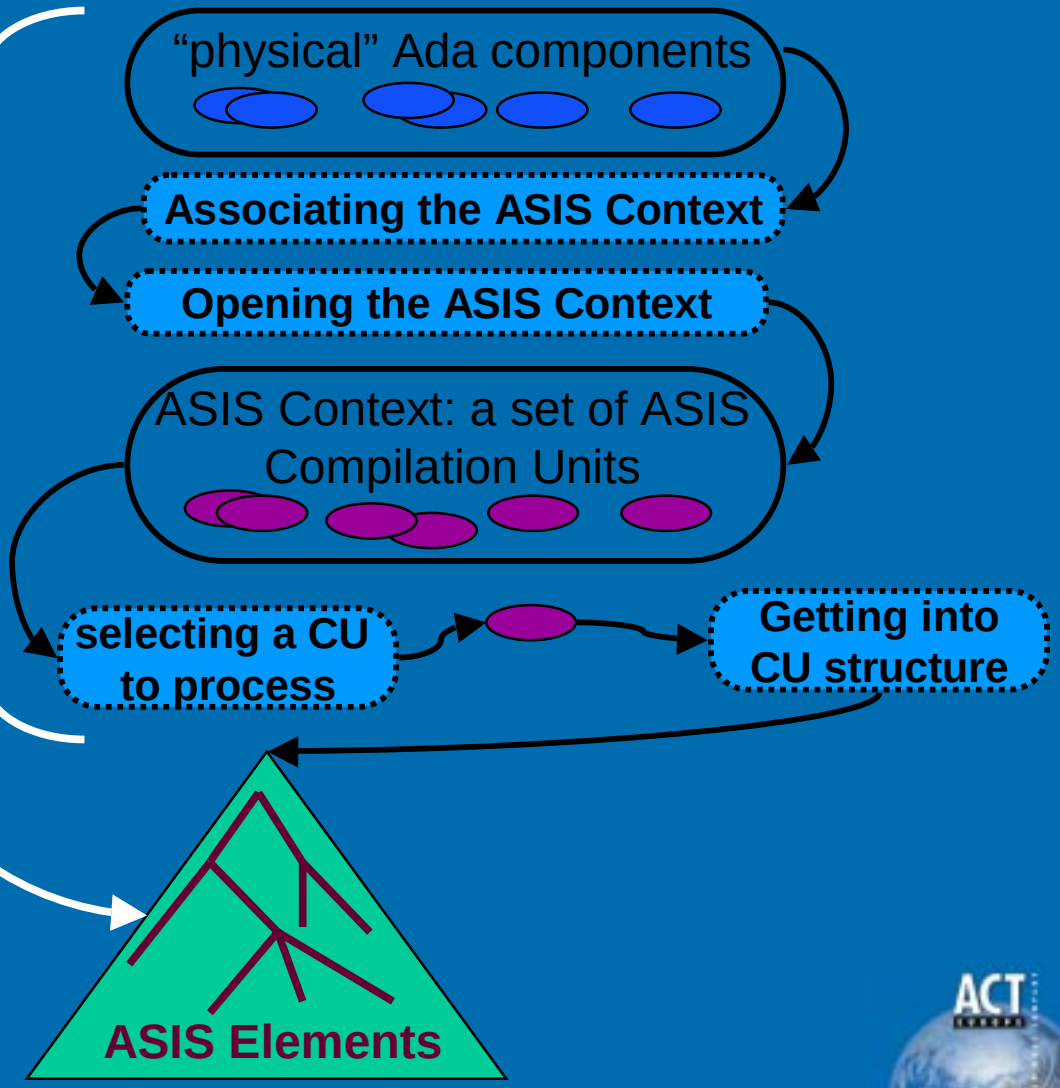
Main ASIS abstractions: Element



An ASIS Element provides a handle for (explicit and implicit) syntax components of ASIS Compilation Units.

The first summary of the design

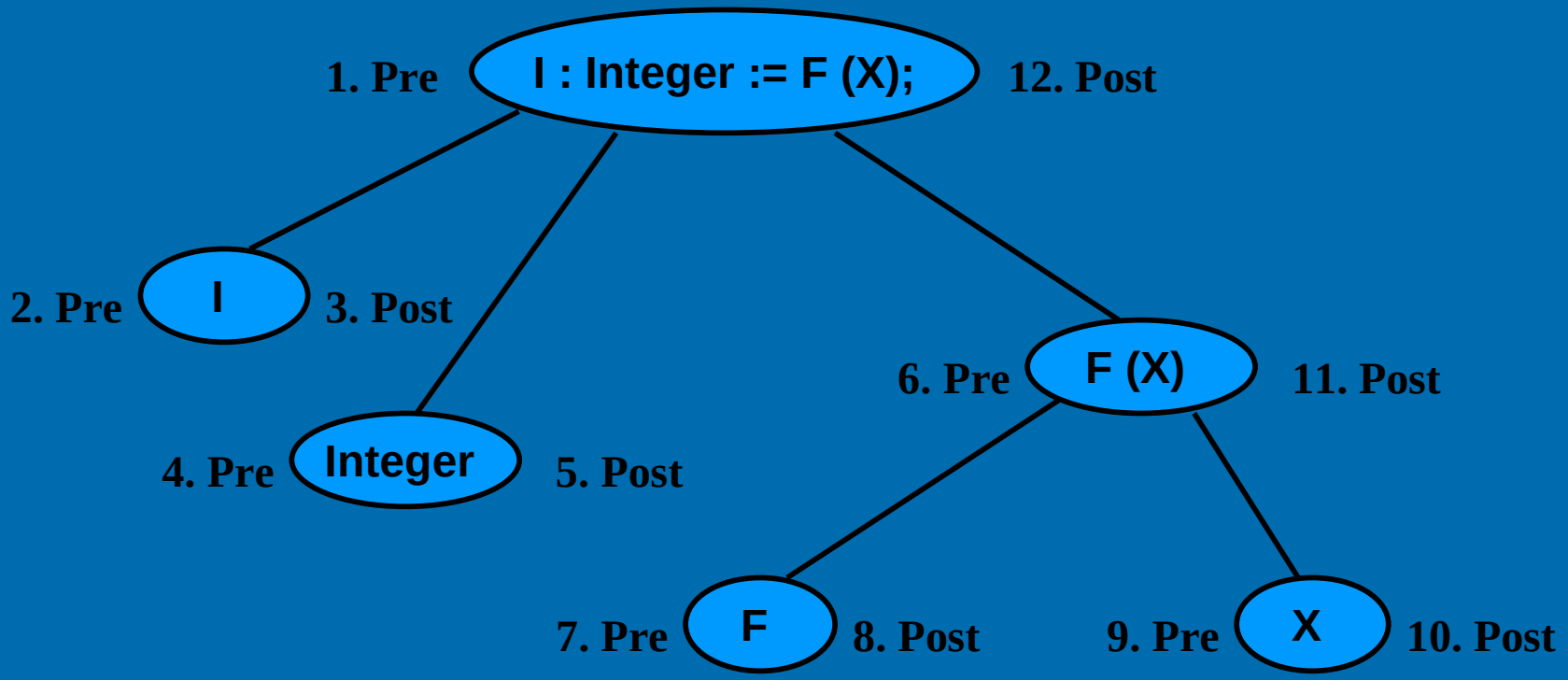
- ✔ **What do we already have:**
- ✔ **What do we need:**
  - **How to traverse the Ada syntax structure?**
  - **How to process specific Elements?**



ASIS general traversing


```
package Asis.Iterator is

   generic
      type State_Information is limited private;

      with procedure Pre_Operation
             (Element : in     Asis.Element;
              Control : in out Traverse_Control;
              State   : in out State_Information) is <>;

      with procedure Post_Operation
             (Element : in     Asis.Element;
              Control : in out Traverse_Control;
              State   : in out State_Information) is <>;

   procedure Traverse_Element
     (Element : in     Asis.Element;
      Control : in out Traverse_Control;
      State   : in out State_Information);

end Asis.Iterator;
```

Depth-first traversing



Specific Element Traversing - actual Pre_Operation for Traverse_Element


```
procedure Check_Style_Rules (Element : Asis.Element;...) is
begin
  case Element_Kind (Element) is
    when A_Declaration =>
      if Names (Element)'Length > 1 then
        Report_Style_Violation (Element, ...);
      end if;

      case Declaration_Kind (Element)is
        when A_Procedure_Body_Declaration|A_Function_Body_Declaration
            |A_Procedure_Body_Stub       |A_Function_Body_Stub =>

          if Is_Nil (Corresponding_Declaration (Element)) then
            Report_Style_Violation (Element, ...);
          end if;
        when others => null;
      end case;

    when others => null;
  end case;
end Check_Style_Rules;
```

Instantiating Traverse_Element


```
type Style_Check_State is (Not_Used);

procedure Check_Style_Rules is … end Check_Style_Rules;

procedure No_Operation (...) is begin null; end No_Operation;

procedure Recursive_Style_Check is new
   Asis.Iterator.Traverse_Element
     (State_Information => Style_Check_State,
      Pre_Operation     => Check_Style_Rules,
      Post_Operation    => No_Operation);

procedure Process_Construct (Construct : Asis.Element) is
 Process_Control : Traverse_Control := Continue;
 Process_State   : Style_Check_State;
begin
  Recursive_Style_Check
   (Construct, Process_Control, Process_State);
end Process_Construct;

-- and here we have the complete solution!!!
```

Some Important details:

- **Exceptions and Error Handling:**

```
with Asis.Exceptions; use Asis.Exceptions;
with Asis.Errors;     use Asis.Errors;
  ...
  exceptions
    when Asis_Inappropriate_Context           |
         Asis_Inappropriate_Compilation_Unit  |
         Asis_Inappropriate_Element           |
         ... =>

       Put_Line (Diagnosis);

       Put_Line (Error_Kinds'Wide_Image (Status));
       ...
```

Some Important details:

- **Reporting Elements:**

```
with Asis.Text; use Asis.Text;
   ...
   Put (First_Line_Number (Construct));
   ...
   Put (Last_Line_Number  (Construct));
   ...
   Put (Element_Image     (Construct));
```

Adding the Check for a New Rule


```
--  all the parameter associations in subprogram calls
--  and all the generic associations in generic instantiations
--  should be given in named form only

procedure Check_Style_Rules (...) is
begin
  case Element_Kind (Element) is
    when A_Declaration => ... -- as before
    when An_Association =>
        case Association_Kind (Element) is
            when A_Parameter_Association|A_Generic_Association =>
               if Is_Nil(Formal_Parameter (Element)) then
                  Report_Style_Violation (Element, ...);
               end if;
     when others =>
         null;
   end case;

end Check_Style_Rules;
```

Style_Checker - summary of the use of ASIS

- ✔ Main ASIS abstractions:
  - Context, Compilation Unit, Element;
- ✔ ASIS package hierarchy:
  - Asis, Asis.Elements, Asis.Declarations, etc.;
- ✔ Compilation Units and Elements classification;
- ✔ ASIS queries:
  - getting Compilation Units from a Context;
  - gateways from Compilation Units into Elements;
  - structural, semantic and classification queries;
- ✔ General traversing algorithm;
- ✔ Accessing source code fragments;
- ✔ ASIS exceptions and error handling policy;

How to use all this?

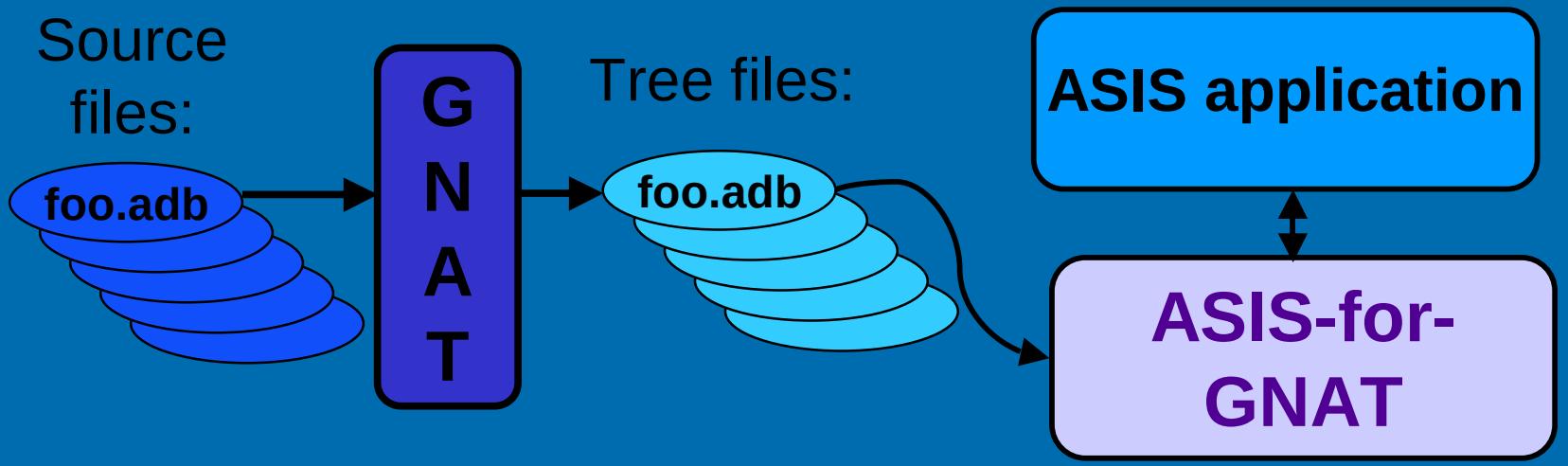
- ✔ **Building an executable with ASIS-for-GNAT (ASIS as an Ada library):**
  - **Installing ASIS-for-GNAT**
    - **compile ASIS-for-GNAT components as ordinary Ada code;**
    - **create a library file;**
  - **Creating the executable for the ASIS style-checker**

```
gnatmake style_checker.adb -largs -lasis
```

- ✔ **Preparing data for running our style checker:**
  - **interacting with the underlying compiler;**

Preparing data for running an ASIS application built with ASIS-for-GNAT:

- **Interacting between GNAT and ASIS-for-GNAT:**



Creating tree output files for ASIS:

>gcc -c -gnatc -gnatt foo.adb

Running style checker

Test compilation unit:

```
procedure Test is                -- 1
   A    : Integer := 13;         -- 2
   B, C : Integer := 13;         -- 3
   procedure P;                  -- 4

   procedure Q is                -- 5
   begin                         -- 6
      null;                      -- 7
   end Q;                        -- 8

   procedure P is                -- 9
   begin                         -- 10
      null;                      -- 11
   end P;                        -- 12
begin                            -- 13
   null;                         -- 14
end;                             -- 15
```

Style_Checker output:

```
Processing Unit: Test (body)
Style violation (multi-identifier
   declaration)
   detected in lines 3 - 3
      B, C : Integer := 13;

Style violation (a subprogram body
   without a separate spec)
   detected in lines 5 - 8

      procedure Q is             --  5
      begin                      --  6
         null;                   --  7
       end Q;

Done...
Processing Unit: Test (spec)
Done...
```

Style_Checker - Conclusions:

- ✔ ASIS
  - seems to be an effective technology for building Ada tools;
  - seems to be easy-to-start and easy-to-use;
- ✔ ASIS implementations - ASIS-for-GNAT
  - ASIS already is an available Ada 95 technology;
  - ASIS is available for GNAT (so it is possible to try it out with the public version of the GNAT technology);

Style_Checker - Questions:

- ✔ ASIS
  - How mature the ASIS technology really is?
  - What about the ASIS learning curve?
- ✔ Building and running ASIS applications
  - Is ASIS the right approach for my tool?
  - Is building the ASIS-based tools really so simple?
  - How portable ASIS tools really are?
- ✔ ASIS-for-GNAT
  - Current state, support, perspectives?
  - How to get, how to install and how to use?
  - What is inside?

The state of the ASIS technology

- ✔ ASIS for Ada 83 originated from practical needs to build development tools;
- ✔ ASIS 83 exists as de-facto standard, it has been stable for many years;
- ✔ ASIS 83 specification has been maintained and evolved into ASIS 95 by the ACM SIGAda ASIS Working Group;
- ✔ ASIS 95 exists as International Information Technology Standard ISO/IEC 15291, this standard has been developed and is now maintained by the ASIS Rapporteur Group
- ✔ ASIS is a secondary standard, based on the Ada 95 Reference Manual;

ASIS information sources

- ✔ **ASIS WG/ASIS RG Home Pages:**
  - **http://www.acm.org/sigada/wg/asiswg**
- ✔ **ASIS Mailing Lists:**
  - **for general discussions:**
    - **SIGADA-ASIS @ ACM.ORG**
  - **for technical discussions:**
    - **SIGADA-ASIS-TECH @ ACM.ORG**
  - **how to join? Just send a message to:**
    - **SIGADA-ASIS-Request @ ACM.ORG**

    or
    - **SIGADA-ASIS-TECH-Request @ ACM.ORG**

Guided Tour through the ASIS specification (how complex ASIS really is?)

- ✔ASIS package hierarchy
- ✔Where is what?
- ✔Element classification hierarchy
- ✔Structural and semantic queries
- ✔Dynamic typing of the ASIS queries
- ✔ASIS error handling policy
- ✔Accessing the source code
- ✔Data Decomposition Annex
- ✔Navigation tips

ASIS package hierarchy

- Asis
  - Asis.Errors
  - Asis.Exceptions
  - Asis.Implementation
    - Asis.Implementation.Permissions
  - Asis.Ada_Environments
    - Asis.Ada_Environments.Containers
  - Asis.Compilation_Units
    - Asis.Compilation_Units.Times
    - Asis.Compilation_Units.Relations
  - Asis.Elements
  - Asis.Iterator
  - Asis.Declarations
  - Asis.Definitions
  - Asis.Expressions
  - Asis.Statements
  - Asis.Clauses
  - Asis.Text
  - Asis.Ids
  - Asis.Data_Decomposition (optional annex)

The top Asis package

- Defines the main ASIS abstractions:
  - type Context
  - type Compilation_Unit
  - type Compilation_Unit_List
  - Compilation Unit classification
  - type Element
  - type Element_List
  - Element subtypes (mnemonic synonymies for the Element type)
  - Element classification
- no queries on its own

Controlling and querying an ASIS implementation

Asis.Implementation:

- controlling your ASIS implementation:
  - Is_Initialized
  - Initialize (...)
  - Is_Finalized
  - Finalize (...)
  - Status
  - Diagnosis
  - Set_Status (...)

Asis.Implementation.Permissions:

- querying the actual implementation of the ASIS implementation-specific features in your ASIS implementation:
  - Is_Prefix_Call_Supported
  - Default_In_Mode_Supported
  - Generic_Actual_Part_Normalized
  - Is_Line_Number_Supported
  - Implicit_Components_Supported

Controlling an ASIS Context

Asis.Ada_Environments:

- **defining, opening and querying the state of a Context:**
  - **Associate (...)**
  - **Open (...)**
  - **Close (...)**
  - **Dissociate (...)**
  - **Is_Open (...)**
  - **Has_Associations (..)**

Asis. Ada_Environments.Containers:

- **grouping Compilation Units inside a Context:**
  - **New feature, introduced in ASIS 95;**
  - **no usage experience so far;**

Black-Box Compilation Unit processing

Asis.Compilation_Units:

- **obtaining units from a Context:**
  - **Library_Unit_Declaration (...)**
  - **Compilation_Unit_Body (...)**
  - **Compilation_Units(...)**
  - **...**
- **semantic dependencies among units:**
  - **Corresponding_Children (...)**
  - **Corresponding_Body (...)**
  - **...**
- **Compilation Unit properties:**
  - **Unit_Kind (...)**
  - **Unit_Full_Name (...)**
  - **...**

Black-Box Compilation Unit processing (cont.)

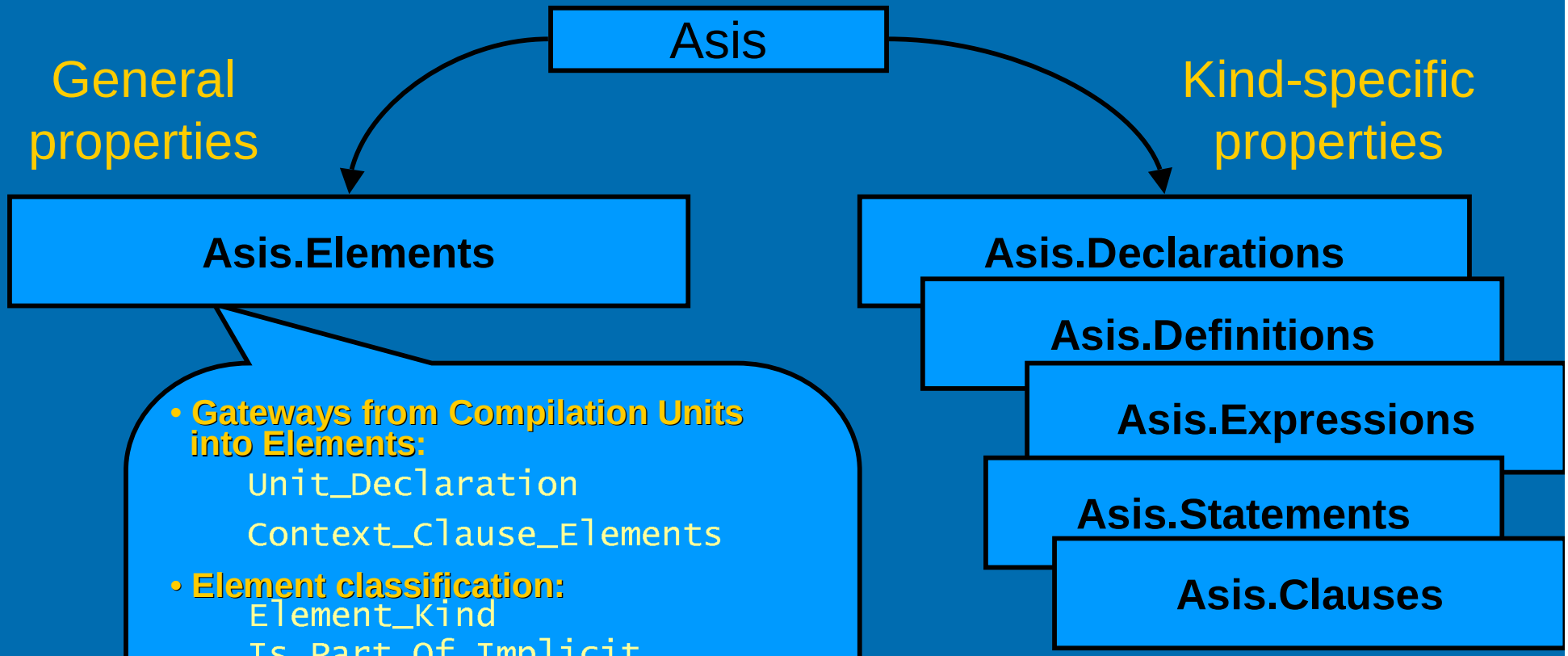
Asis.Compilation_Units.Times :

- **querying time characteristics of Compilation Units:**
  - **Time_Of_Last_Update (...)**
  - **...**

Asis.Compilation_Units.Relations:

- **obtaining integrated semantic dependencies:**
  - **Semantic_Dependence_Order(...)**
  - **Elaboration_Order (...)**

Working with Elements

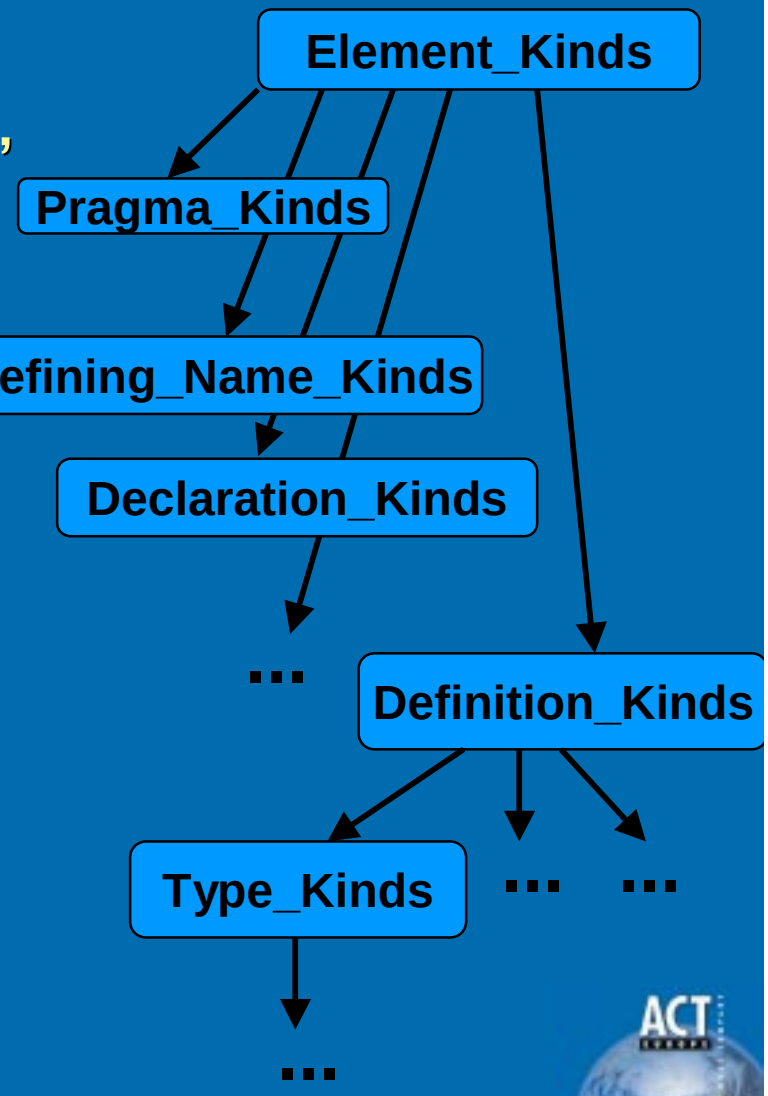


- **Gateways from Compilation Units into Elements:**
  - `Unit_Declaration`
  - `Context_Clause_Elements`
- **Element classification:**
  - `Element_Kind`
  - `Is_Part_Of_Implicit`
  - `...`
- `Is_Equal`
  `Is_Identical`
- `Enclosing_Compilation_Unit`
- `Enclosing_Element`
- **Queries working on pragmas**

Element classification hierarchy (1)

Package Asis:

```
type Element_Kinds is (Not_An_Element,
   A_Pragma, A_Defining_Name,
   A_Declaration, A_Definition,...);

type Definition_Kinds is
   Not_A_Definition,
   A_Type_Definition,
   A_Subtype_Indication, ...);

type Type_Kinds is
   (Not_A_Type_Definition,
   A_Derived_Type_Definition, …,
   An_Access_Type_Definition );

type Access_Type_Kinds is
   (Not_An_Access_Type_Definition,
   A_Pool_Specific_Access_To_Variable,
   ...);
```



Element classification hierarchy (2)

```
function Element_Kind
   (Element : in Asis.Element)
    return Asis.Element_Kinds;

function Pragma_Kind
   (Pragma_Element : in
    Asis.Pragma_Element)
    return Asis.Pragma_Kinds;

function Defining_Name_Kind
   (Defining_Name : in Asis.Defining_Name)
    return Asis.Defining_Name_Kinds;

function Declaration_Kind
   (Declaration : in Asis.Declaration)
    return Asis.Declaration_Kinds;

...
```

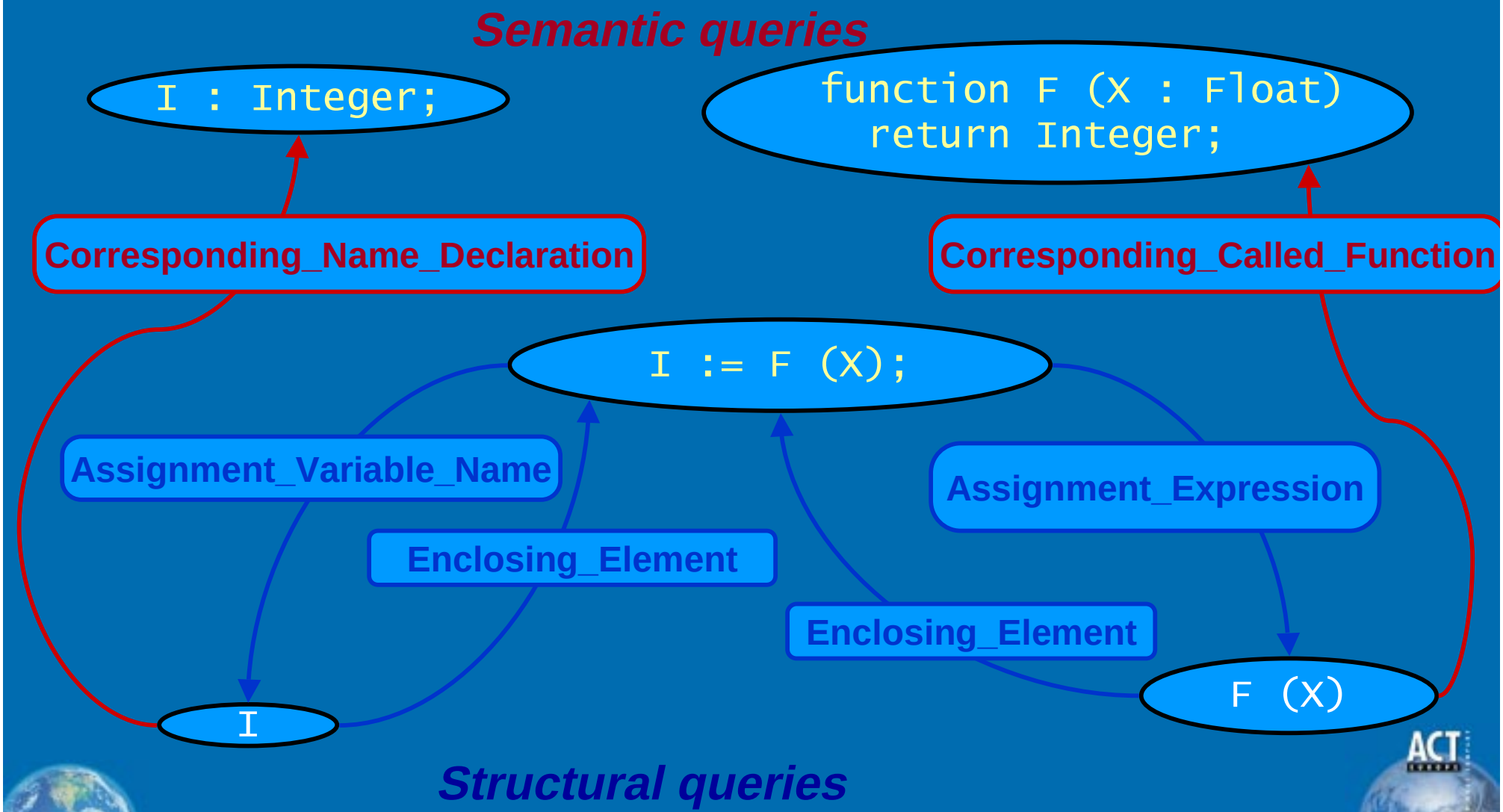
package Asis:

```
type Mumbo_Kinds is
   (Not_A_Mumbo,
    Mumbo_A, Mumbo_B,
    ...);
```

package Asis.Elements:

```
function Mumbo_Kind is
   (M : Asis.Element)
    return Mumbo_Kinds;
```

Structural and semantic queries



Generic instantiations in ASIS

```
generic
  type T is private;
package Gen_Pack is
  procedure Proc (X:T);
end Gen_Pack;

package body Gen_Pack is
  procedure Proc (X:T) is
  begin
     ...
  end Proc;
end Gen_Pack;
...
package New_Pack is new
  Pack (Integer);
...
with New_Pack;
use New_Pack;
procedure Test is
  I : Integer;
begin
  P (I);
end Test;
```

Is_Part_Of_Instance

```
package New_Pack is
  procedure Proc
    (X : Integer);
end New_Pack;
```

Corresponding_Declaration

Corresponding_Called_Entity

Corresponding_Body

Enclosing_Element

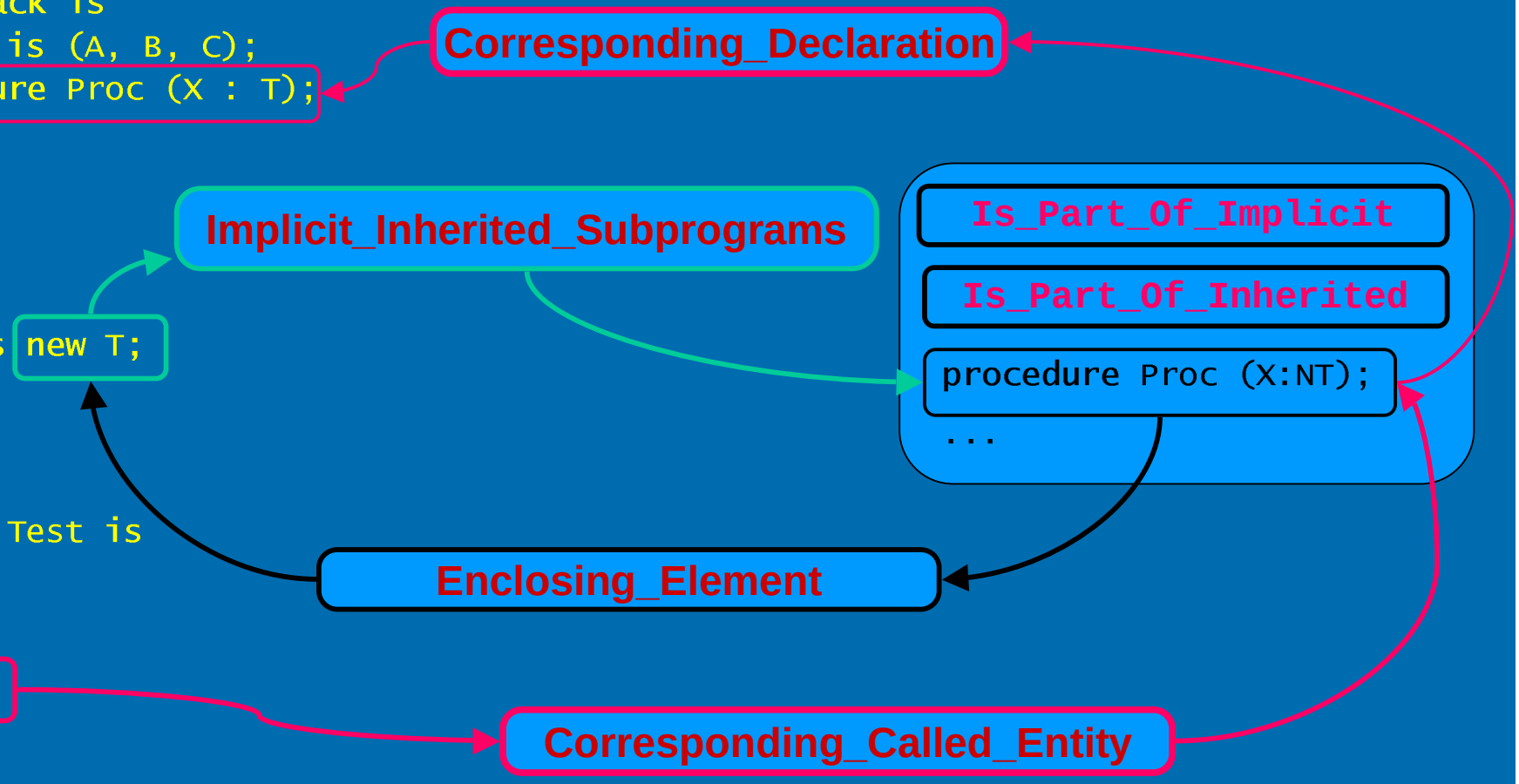
Is_Part_Of_Instance

```
package body New_Pack is
  procedure Proc
    (X : Integer)
  is
  begin
    …
  end Proc;
end New_Pack;
```

Corresponding_Body

Implicit declarations in ASIS

```
package Pack is
   type T is (A, B, C);
   procedure Proc (X : T);
   ...
end Pack;

...

type NT is new T;

...

procedure Test is
  I : NT;
begin

   P (I);

end Test;
```



Comment sentinels

--|ER	Element Reference
--|CR	Child Reference
--|AN	Application Note
--|IP	Implementation Permissions
--|IR	Implementation Requirements

```
--|ER--------------------------------------
--|ER An_Ordinary_Type_Declaration - 3.2.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Discriminant_Part
--|CR    function Type_Declaration_View
--
-----------------------------------------
-- Subclause 15.7 function Discriminant_Part
-----------------------------------------
  function Discriminant_Part
    (Declaration : in Asis.Declaration)
     return           Asis.Definition;
-----------------------------------------
...
```

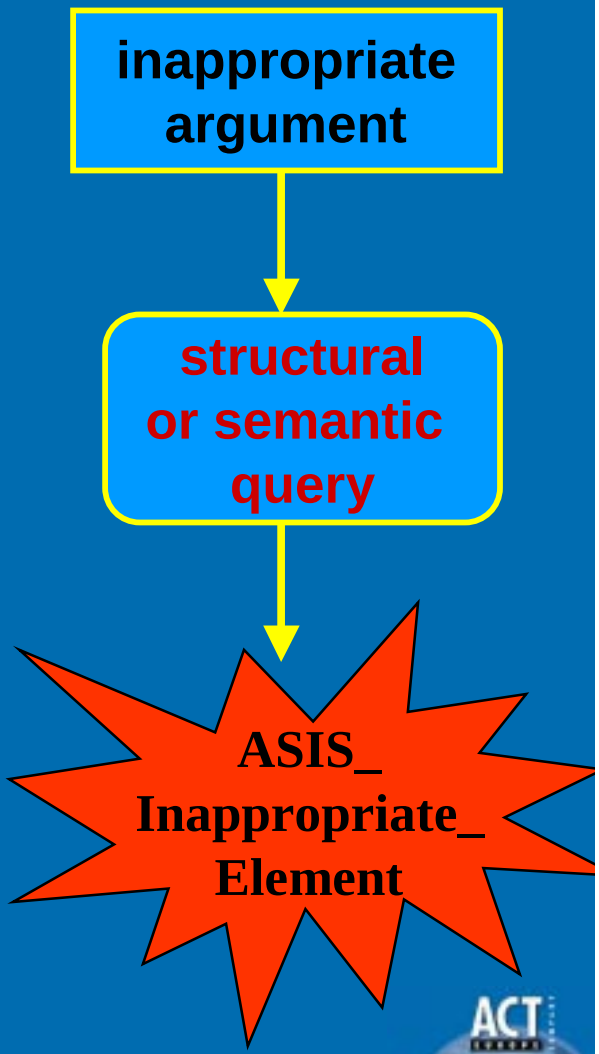
Data Decomposition

Asis.Data_Decomposition

- **provides means for**
  - **decomposing data values using the ASIS type information and an abstract data stream representing a data value of that type;**
  - **getting representation information for composite types and values;**
- **main abstractions:**
  - **type** Record_Component **is private;**
  - **type** Array_Component **is private;**
  - **type** Portable_Data **is array (**Portable_Positive **range <>) of** Portable_Value**;**

“Dynamic type control” in ASIS

```
   ------------------------------------------
   -- 18.3  function Assignment_Expression
   ------------------------------------------
   function Assignment_Expression
     (Statement : in Asis.Statement)
      return Asis.Expression;
   ------------------------------------------
   -- Statement   - Specifies the assignment
   -- statement to query
   --
   -- Returns the expression from the right hand
   -- side of the assignment.
   --
   -- Appropriate Statement_Kinds:
   --       An_Assignment_Statement
   --
   -- Returns Element_Kinds:
   --       An_Expression
```

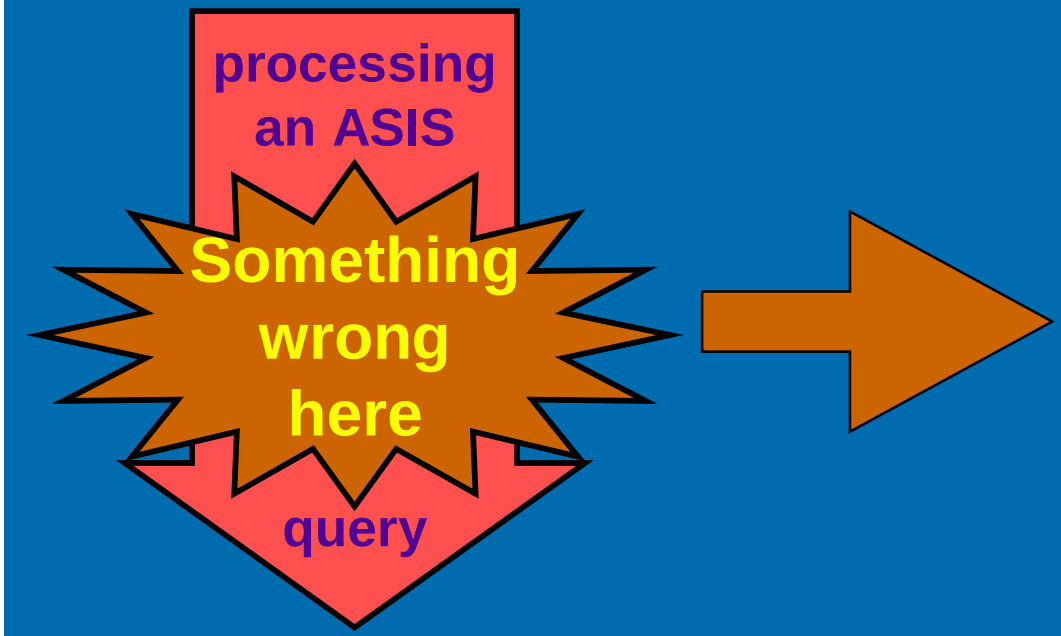


Accessing the source code

package Asis.Text:

- **main abstractions:**
  - **type Line is private;**
  - **type Span is record...**
- **obtaining Spans and Lines (Line lists):**
  - **Span of an Element, a Compilation Unit or a full compilation;**
  - **Lines covering an Element or a Span;**
  - **Range of Lines;**
- **obtaining text images:**
  - **of an Element;**
  - **of a Line (full image, comment image or non-comment image);**

ASIS error handling policy



- An ASIS-defined exception should be raised;
- An ASIS Error Status should be set;
- The Diagnosis string should be formed;

- Asis.Exceptions: defines ASIS exceptions;
- Asis.Errors : defines ASIS error kinds;
- Asis.Implementation:
  - **`Status`** - queries for a current Error Status value
  - **`Diagnosis`** - queries for a current Diagnosis string
  - **`Set_Status`** - resets Error Status Value and Diagnosis string

Navigation tips

- ASIS queries are grouped into child packages according to the abstractions they are working on and a piece of the ASIS functionality they provide (Asis.Implementation, Asis.Text; Asis.Compilation_Units etc.);
- Queries working with Elements are additionally grouped according to the general kind of their argument (Asis.Declarations, Asis.Statements etc.)
- Inside each package working on Elements of a particular Element kind queries are ordered according to RM 95 section/subsection ordering;
- Names of all the semantic queries start from Corresponding_ or Implicit_;
- Use comment sentinels when browsing through the ASIS packages;
- asistant interactive help (ASIS-for-GNAT)

Basic ASIS application cycle

```
Asis.Implementation.Initialize (...);
   Asis.Ada_Environments.Associate (...);
      Asis.Ada_Environments.Open (...);

         --  Use the various ASIS queries:
         --  (fetch a unit, its attributes, get its
         --  Unit_Declaration element,
         --  traverse its elements, etc.

      Asis.Ada_Environments.Close (...);
   Asis.Ada_Environments.Dissociate (...);
Asis.Implementation.Finalize (...);
```

- Portability of ASIS tools;
- Validity and obsoleteness of the data retrieved though ASIS;
- Read-only nature of ASIS and developing non-read-only ASIS tools;

Portability of ASIS tools

```
Asis.Implementation.Initialize (...);
   Asis.Ada_Environments.Associate (...);
      Asis.Ada_Environments.Open (...);

         --  Use the various ASIS queries:
         --  (fetch a unit, its attributes, get its
         --  Unit_Declaration element,
         --  traverse its elements, etc.

      Asis.Ada_Environments.Close (...);
   Asis.Ada_Environments.Dissociate (...);
Asis.Implementation.Finalize (...);
```

Depends on implementation-specific features

Dependency on implementation-specific features may be considerably reduced or even eliminated

Isolating the implementation-specific aspects in your tool

Top-level driver:
- initializes an ASIS implementation;
- processes implementation-specific parameters;
- defines and opens Contexts to process;

Should not use ASIS queries dealing with Compilation Units and Elements;

Iterating trough a Context:
- Obtains from a Context Compilation Units to process;
- Investigates black-box unit properties if needed;
- Selects Unit for white-box processing;

Should not go into Elements

Processing a single unit:
- decomposes a unit into Elements;
- selects Element to analyze or to traverse;
- calls to routines working on Elements;

Element processing

concentrates on Elements only;

- implementation-specific

- may contain some implementation-specific features

- implementation-independent

Validity and obsoleteness of the data retrieved through ASIS


```
procedure Wrong_Use_Of_ASIS is
   Cont  : Context;
   Unit  : Compilation_Unit;
   Elem  : Element;
begin
   Initialize;
   Associate (Cont, "Some Name", Some_Params):
   Open (Cont);
   Unit := Library_Unit_Declaration ("Some_Unit", Cont);
   Elem := Unit_Declaration (Unit);
   Close (Cont); --  !!! Closing the context!!!
   --  !!! Unit and Elem are not valid any more
   Open (Cont);
   declare
      List1 : Element_List := Context_Clause_Elements (Unit);
      --  Wrong use of Unit! ASIS_Failed will be raised!
      List2 : Element_List:= Names (Elem);
      --  Wrong use of Elem! ASIS_Failed will be raised!
      ...
   begin
   ...
```

Read-only nature of ASIS

- ✔ When working, ASIS should not make any change in the underlying Ada environment;
- ✔ When an ASIS Context remains opened, ASIS supposes that data structures making up a physical environment this Context is associated with can not be changed by any ASIS or non-ASIS program.
- ✔ If this is not the case, the execution of an ASIS application is erroneous;
- ✔ When an ASIS Context remains closed, there is no limitations on changes in a physical environment (that is why an Element should not be used after closing and reopening its enclosing Context);

Developing non-read-only tools with ASIS

Top-level driver: defines the ASIS context to process

```
Ctx : Asis.Context;
```

```
Associate (Ctx, ...);
...
Open (Ctx);
...                    --  (1)
Close (Ctx);
```

Non-ASIS component:

Making changes in the physical environment

```
Open (Ctx);
...                    --  (2)
Close (Ctx);
```

...

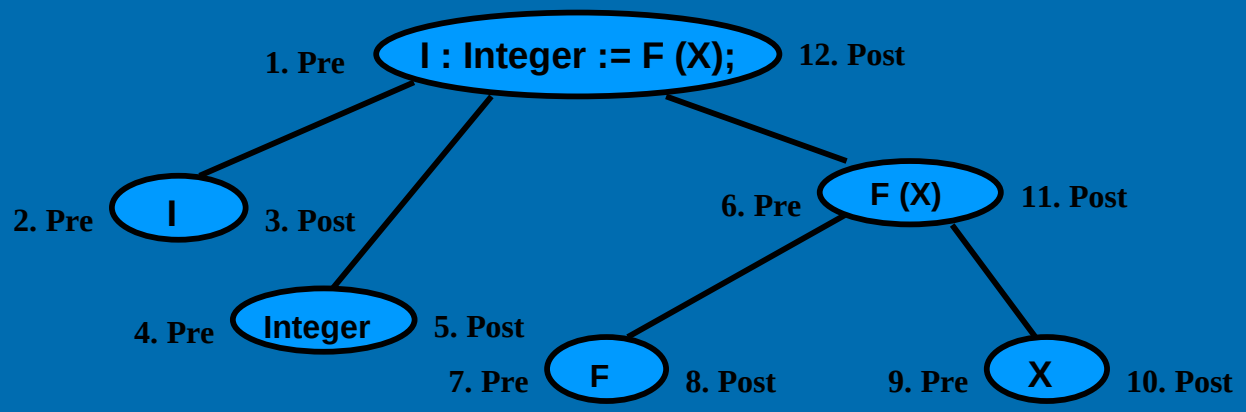
ASIS components

The content of Ctx in (1) and (2) may be different!

More about the ASIS general traversal algorithm

Reminder: what is depth-first traversal

```
generic
   type State_Information is limited private;
   with procedure Pre_Operation (...)
   with procedure Post_Operation (...)
procedure Traverse_Element
    (Element : in Asis.Element;
     Control : in out Traverse_Control;
     State   : in out State_Information);
```



Controlling the traversal

Package ASIS:

```
type Traverse_Control is (
   Continue,                  --  Continues the normal
                              --  depth-first traversal.

   Abandon_Children,          --  Prevents traversal of
                              --  the current element's
                              --  children.

   Abandon_Siblings,          --  Prevents traversal of
                              --  the current element's
                              --  children and remaining
                              --  siblings.

   Terminate_Immediately);-- Does exactly that.
```

Using the traversal State

A natural way way to control indentations when pretty-printing:

```
type Ident_State is record
       Spaces : Natural := 0;
end record;
```

```
procedure Pre_Op
 (E :         Element;
  C : in out Control:
  I : in out Ident_State) is
begin
  case Element_Kind (E) is
    when ... =>
      I.Spaces := I.Spaces + Tab;
     ...
  end case;
   ...
end Pre_Op;
```

```
procedure Post_Op
 (E :         Element;
  C : in out Control:
  I : in out Ident_State) is
begin
  case Element_Kind (E) is
    when ... =>
      I.Spaces := I.Spaces - Tab;
     ...
  end case;
   ...
end Post_Op;
```

Make your own ASIS secondary layer

- ✔ Asis may be of relatively low level when comparing with what is needed for your tool;
- ✔ Make useful combinations of ASIS queries reusable – build your own ASIS secondary layer;
- ✔ Follow the ASIS documentation style when describing secondary queries (lists of appropriate and returned Element/Compilation Unit kinds);
- ✔ Follow the ASIS error handling policy in your tools and in your secondary queries

ASIS secondary queries - some examples


```
function Acts_As_Spec
   (Declaration : Asis.Element)
    return        Boolean;
--  Checks if its argument is a subprogram body declaration for
--  which no separate subprogram declaration exists. Returns
--  False for any unexpected argument

function Is_Renaming_As_Body
   (Declaration : Asis.Element)
    return        Boolean;
--  Checks if its argument is a renaming-as-body
--  declaration. Returns False for any unexpected argument.

function Components
   (Element : Asis.Element)
    return    Asis.Element_List;
--  Returns the list of all the first-level components of its
--  argument. Nil_Element is returned for a terminal component.
```

ASIS secondary queries - some ideas

- ✔ **Object-Oriented ASIS queries:**
  - **getting the derivation class;**
  - **getting the list of the primitive operations for a type;**
- ✔ **getting the enclosing declarative region;**
- ✔ **getting the list of overloaded subprograms visible at a given place;**
- ✔ **getting the full expanded name for a short use-visible name;**

Working with ASIS lists

- ✔ **Asis Compilation Unit and Element Lists are one-dimensional unconstrained arrays. Therefore:**
  - **Initialize list variables when declaring them;**
  - **Keep declarations of list variables together to avoid extra block statements;**
  - **Use subprograms to process lists (that is, declare list objects as formal parameters rather then as variables);**
  - **Use Traverse_Element instances to process Elements containing lists in their structure;**
  - **Be aware of pragmas in Element Lists;**
  - **Consider providing your own List abstraction;**

Working with pragmas

Body_Decl:

```
procedure Hello is
begin
   pragma Page;
   Put ("Hello");
   pragma List (Off);
   New_Line;
end Hello;
```

```
Body_Statements
  (Declaration => Body_Decl,
   Include_Pragmas => True)
```

```
pragma Page;
Put ("Hello");
pragma List (Off);
New_Line;
```

```
Body_Statements
  (Declaration => Body_Decl,
   Include_Pragmas => False)
```

```
Put ("Hello");
New_Line;
```

Why and when to use ASIS?

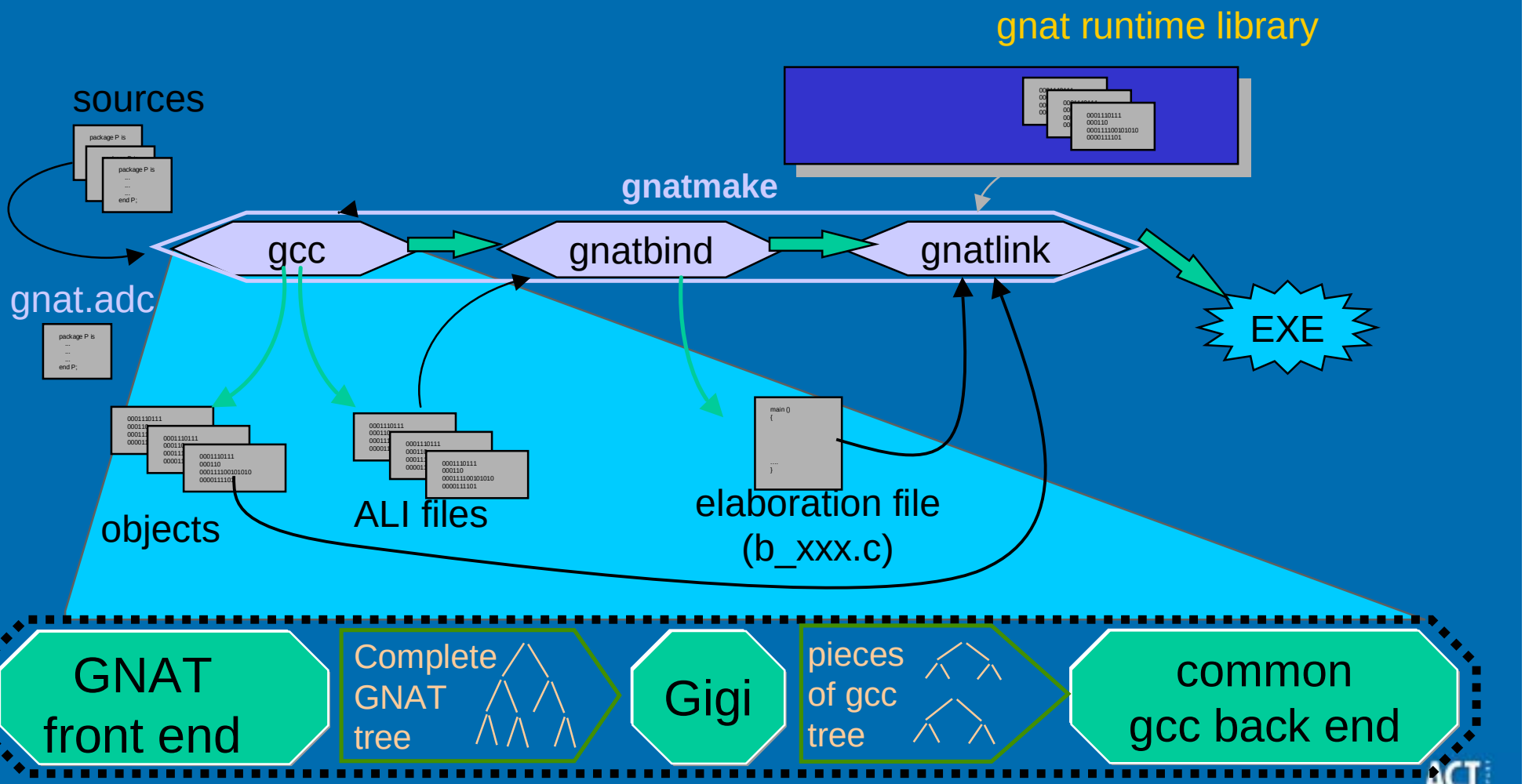
- ✔What can and what can not ASIS do?
  - ASIS provides FULL STATICALLY-DETERMINABLE SYNTAX and SEMANTIC information about compilation units in the Ada environment;
  - ASIS can NOT provide information about DYNAMIC properties of Ada programs;
  - ASIS is a READ-ONLY interface (but this does not block building not-read-only tools with ASIS);

Why and when to use ASIS?

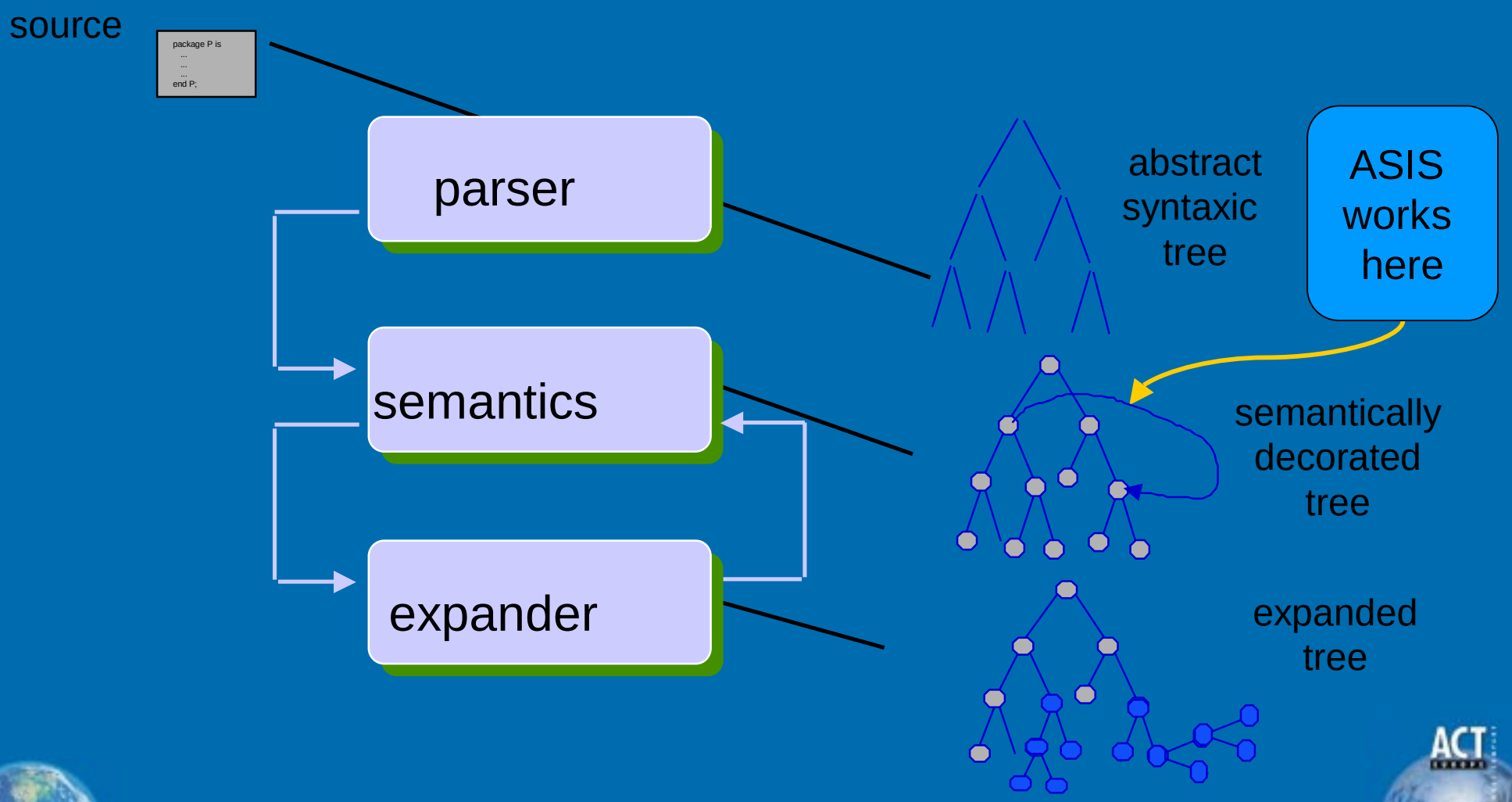
✔Examples of tools that can benefit from ASIS:

- browsers
- call tree tools
- code reformators
- coding standards compliance tools
- correctness verifiers
- debuggers
- design tools
- document generators
- metrics tools
- quality assessment tools
- reverse and re-engineering tools
- test tools
- feature use detectors
- translators
- style checkers
- symbolic computators

The GNAT compilation System



Front End Internals



Source Based Library

P.adb

```
with Q;
procedure P is
   ...
   delay 3.0
   ...
end P;
```

Q.ads

```
package Q is
   ...
end Q;
```

front

end

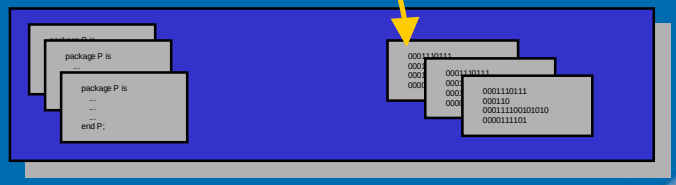
front

end

front

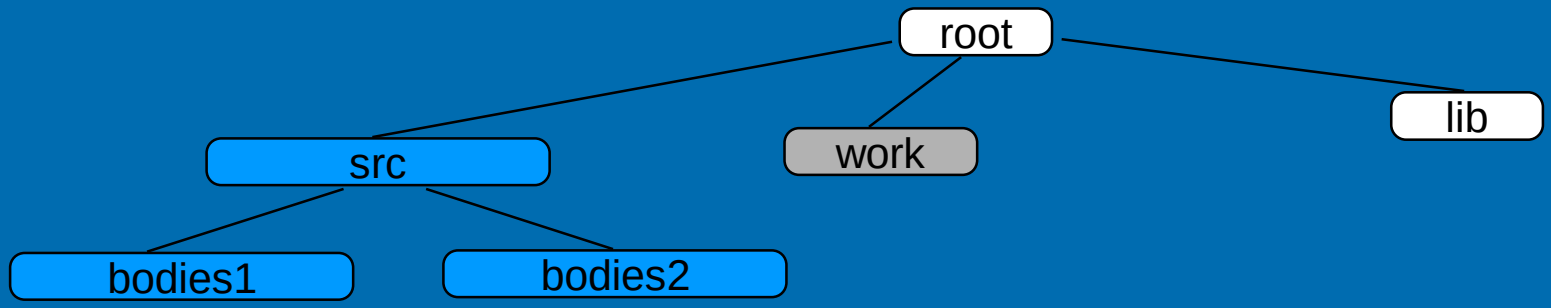
end

rtsfind mechanism



GNAT runtime sources

Source and object search paths

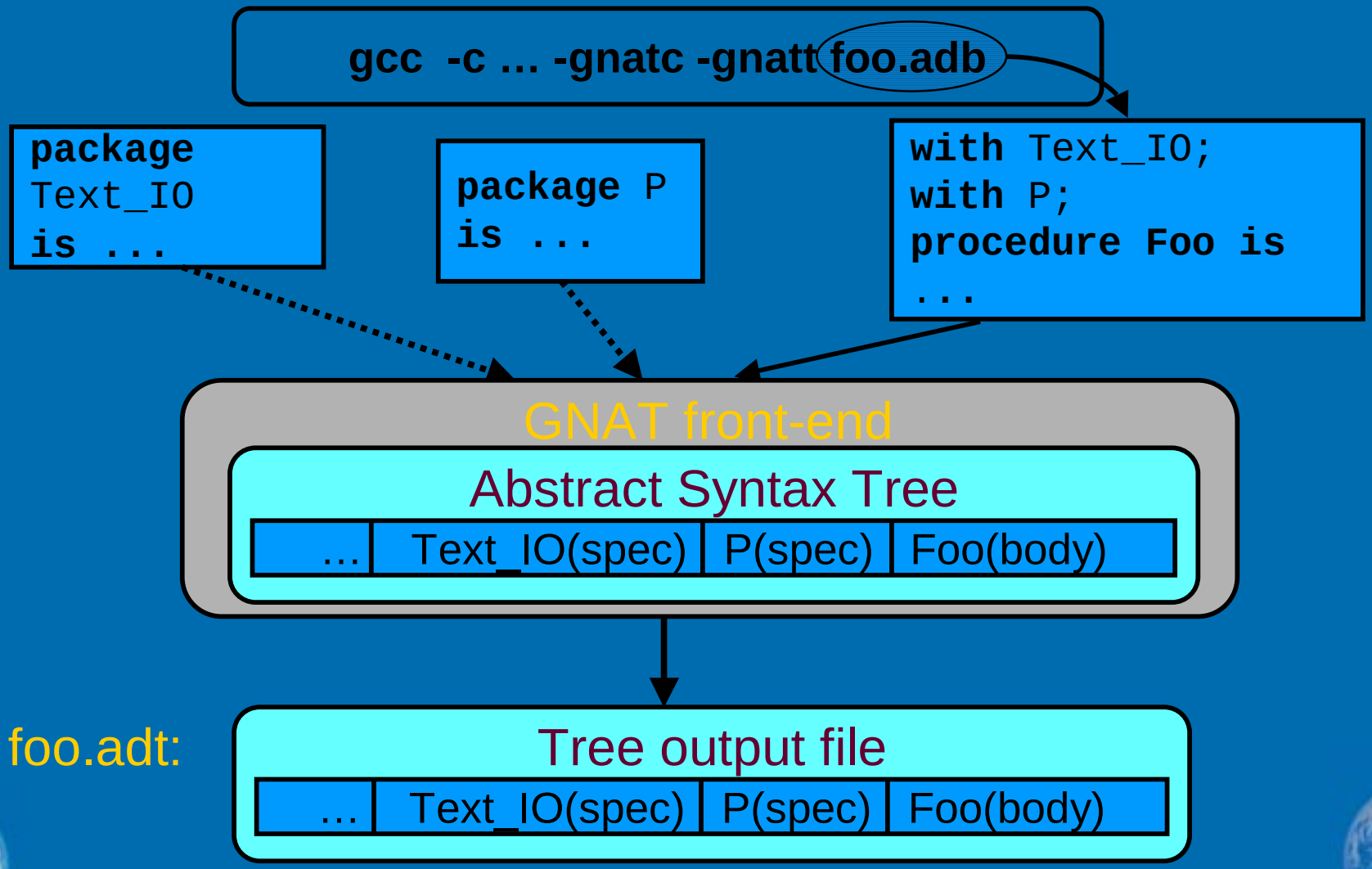


- ✔ user sources in src with alternate bodies in bodies1, bodies2
- ✔ shared sources and objects in lib
- ✔ expect exec and objects in work

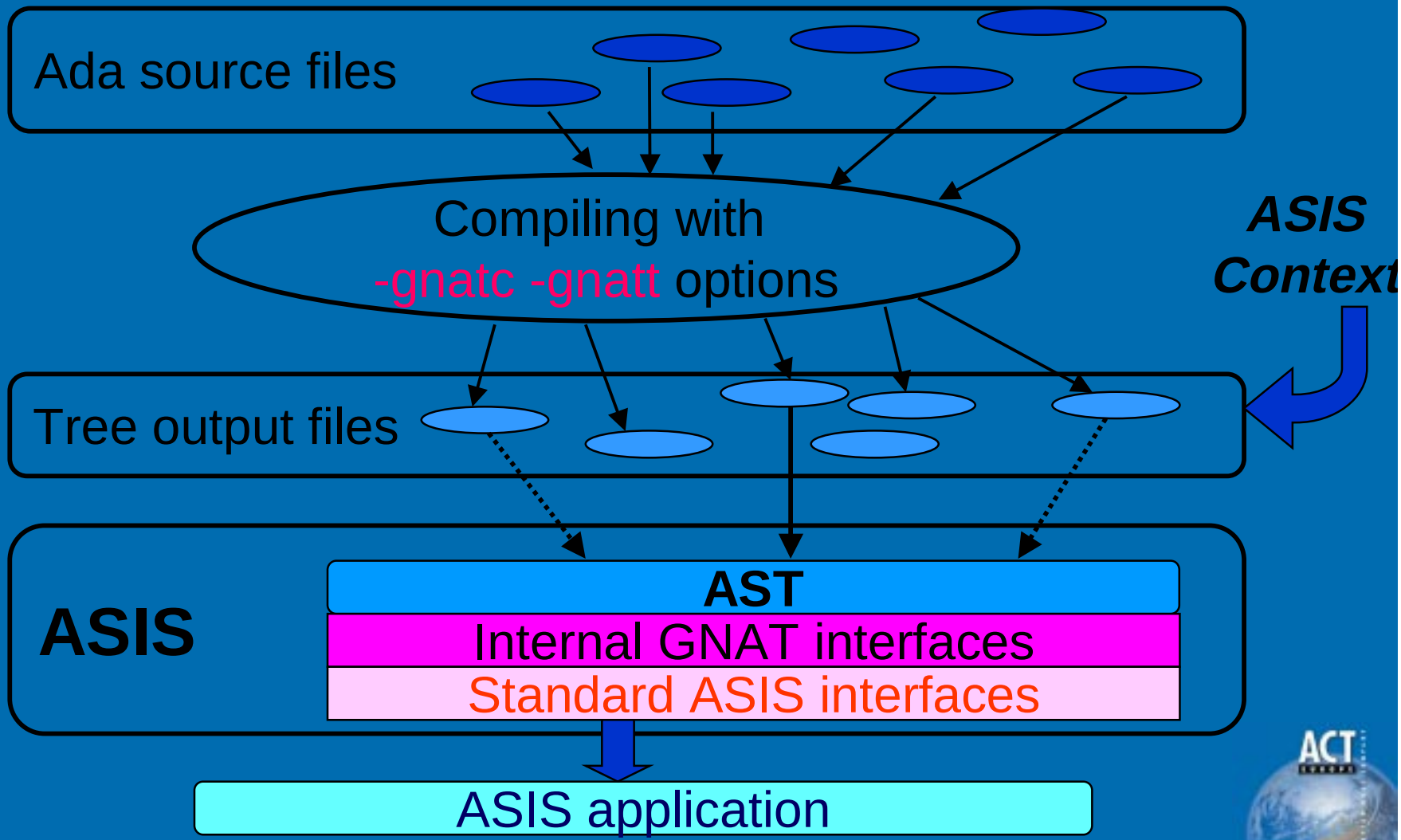
```
cd work;
gnatmake -aI../src -aI../src/bodies1 \
         -I../lib ../src/main.adb
```

- ✔ file naming rules;
- ✔ search path command line options;
- ✔ GNAT-specific environment variables
  - ADA_SOURCE_PATH
  - ADA_OBJECTS_PATH

ASIS-for-GNAT: How it works (tree files)



ASIS-for-GNAT: How it works (ASIS Context)



ASIS-for-GNAT: internal data structures

- A trivial Ada program:

```
procedure Demo is
  begin
     null;
  end Demo;
```

- And the beginning of its tree (continued on the next slide):

```
Node #1 N_Compilation_Unit (Node_Id=1356) (source,analyzed)
 Parent = <empty>
 Sloc = 10  demo.adb:1:11
 Library_Unit = Node #1 N_Compilation_Unit (Node_Id=1356)
 Context_Items = List #2 (List_Id=-99999991)
 Unit = Node #5 N_Subprogram_Body (Node_Id=1363)
 Aux_Decls_Node = Node #11 N_Compilation_Unit_Aux (Node_Id=1357)
 Has_No_Elab_Code = True
 Acts_As_Spec = True

  List #2 (List_Id=-99999991)
  |Parent = Node #1 N_Compilation_Unit (Node_Id=1356)
  |
  Node #3 N_With_Clause (Node_Id=2023s)
   Sloc = Standard_Location
   Name = Node #4 N_Identifier "system" (Node_Id=2022s)
   Library_Unit = N_Compilation_Unit (Node_Id=1367)
   Corresponding_Spec = N_Defining_Identifier "system" (Entity_Id=1370)
   First_Name = True
   Last_Name = True
   Implicit_With = True
```

* (previous slide)

```
ode #4 N_Identifier "system" (Node_Id=2022s)
  Parent = Node #3 N_With_Clause (Node_Id=2023s)
  Sloc = Standard_Location
  Chars = "system" (Name_Id=300000323)
  Entity = N_Defining_Identifier "system" (Entity_Id=1370)

Node #5 N_Subprogram_Body (Node_Id=1363) (source,analyzed)
 Parent = Node #1 N_Compilation_Unit (Node_Id=1356)
 Sloc = 0  demo.adb:1:1
 Specification = Node #6 N_Procedure_Specification (Node_Id=1362)
 Declarations = <empty list> (List_Id=-99999990)
 Handled_Statement_Sequence = Node #8 N_Handled_Sequence_Of_Statements (Node_Id=1364)
 Acts_As_Spec = True

  Node #6 N_Procedure_Specification (Node_Id=1362) (source)
   Parent = Node #5 N_Subprogram_Body (Node_Id=1363)
   Sloc = 0  demo.adb:1:1
   Defining_Unit_Name = Node #7 N_Defining_Identifier "demo" (Entity_Id=1358)

    Node #7 N_Defining_Identifier "demo" (Entity_Id=1358) (source)
     Parent = Node #6 N_Procedure_Specification (Node_Id=1362)
     Sloc = 10  demo.adb:1:11
     Chars = "demo" (Name_Id=300000912)
     Next_Entity = N_Defining_Identifier "system" (Entity_Id=1370)
     Scope = N_Defining_Identifier "standard" (Entity_Id=2s)
     Ekind = E_Procedure
     Etype = N_Defining_Identifier "_void_type" (Entity_Id=864s)
     Scope_Depth = 1 (Uint = 500032769)
     Has_Completion = True
     Is_Compilation_Unit = True
     Is_Frozen = True
     Is_Immediately_Visible = True
     Is_Public = True
     Needs_Debug_Info = True

    Node #8 N_Handled_Sequence_Of_Statements (Node_Id=1364) (source,analyzed)
     Parent = Node #5 N_Subprogram_Body (Node_Id=1363)
     Sloc = 29  demo.adb:3:4
     Statements = List #9 (List_Id=-99999989)

      List #9 (List_Id=-99999989)
      |Parent = Node #8 N_Handled_Sequence_Of_Statements (Node_Id=1364)
      |
      Node #10 N_Null_Statement (Node_Id=1365) (source,analyzed)
       Sloc = 29  demo.adb:3:4

  Node #11 N_Compilation_Unit_Aux (Node_Id=1357) (source)
   Parent = Node #1 N_Compilation_Unit (Node_Id=1356)
   Sloc = 10  demo.adb:1:11
```

License Considerations

- ✔ GNAT: GPL (GNU Public License):
  - is available with full sources;
  - any GNAT user is allowed to distribute a product contained original or modified GNAT component PROVIDED THAT FULL SOURCES ARE AVAILABLE FOR THE FULL PRODUCT
  - this rule is “transitive”: any program built using at least one source component covered by GPL is automatically covered by GPL and should be distributed with full sources
- ✔ ASIS-for-GNAT: Library (Modified) GPL:
  - is available with full sources;
  - if an ASIS tool only uses ASIS-for-GNAT as an Ada library, this does not impose any special license requirements on the tool;
- ✔ ASIS-for-GNAT allows to develop both free and proprietary tools

ASIS-for-GNAT: the Structure of the Distribution

- ✔ Is distributed in the source form;
- ✔ directory hierarchy containing a set of text files;
  - asis - ASIS implementation sources;
  - gnat - needed GNAT components;
  - tools - ASIS-based tools;
  - examples - ASIS application examples;
  - tutorial - hands-on ASIS tutorial;
  - templates - ASIS application templates;
  - documentation - documentation;
  - obj - needed for installation procedure;
  - README - the file describing the installation procedure;
- ✔ No platform-specific components;

ASIS-for-GNAT: installation and usage

- ✔ To install means to compile all the implementation components and to create a library file:
  - Makefiles for UNIX systems or .bat file for MS Windows 95/98/NT

  and updating GNAT environment variables;
- ✔ Binary ASIS distribution will come soon;
- ✔ Building the executables for ASIS applications:

```
gnatmake my_asis_appl -largs -lasis
```

- ✔ Preparing the data for ASIS tools

```
gnatmake -c -gnatc -gnatt main_prog.ads
```

- ✔ Typically a Context is defined as a search path for tree files;
- ✔ There are different ways to define a Context (e.g. trees may be created on the fly)

ASIS-for-GNAT: existing tools and examples

- ✔ **Tools:**
  - **asistant**
    - Interactive interpreter of ASIS queries;
    - Useful when learning ASIS and experimenting with ASIS queries;
  - **gnatelim**
    - Detects unused subprograms
  - **gnatstub**
    - Creates an empty body "stub" for a library package
- ✔ **Examples:**
  - **Display_Source**
    - Reproduces the source of a compilation unit
    - May be used as a "template" for code analysis tools
  - **ASIS Checker**
    - Checks Ada code against a set of rules, the set of rules may easily be changed
    - Some of the rules from "Ada 95 Quality and Style" are implemented as default

ASIS-for-GNAT: templates and tutorial

- ✔ **ASIS Application Templates:**
  - Implement simple solutions for the main components of the basic ASIS application cycle;
  - Allow to build simple ASIS applications by providing only the actual Pre- and Post-Operation for the instantiation of Asis.Iterator.Traverse_Element;
- ✔ **Tutorial:**
  - Contains simple ASIS tasks:
    - To find and to apply the needed sequence of the ASIS queries in the asistant environment;
    - To build a simple ASIS tool from the ASIS Application Templates;
  - Full solutions are provided for all tasks;
  - May be used as a set of ASIS examples

ASIS-for-GNAT: current state and perspectives

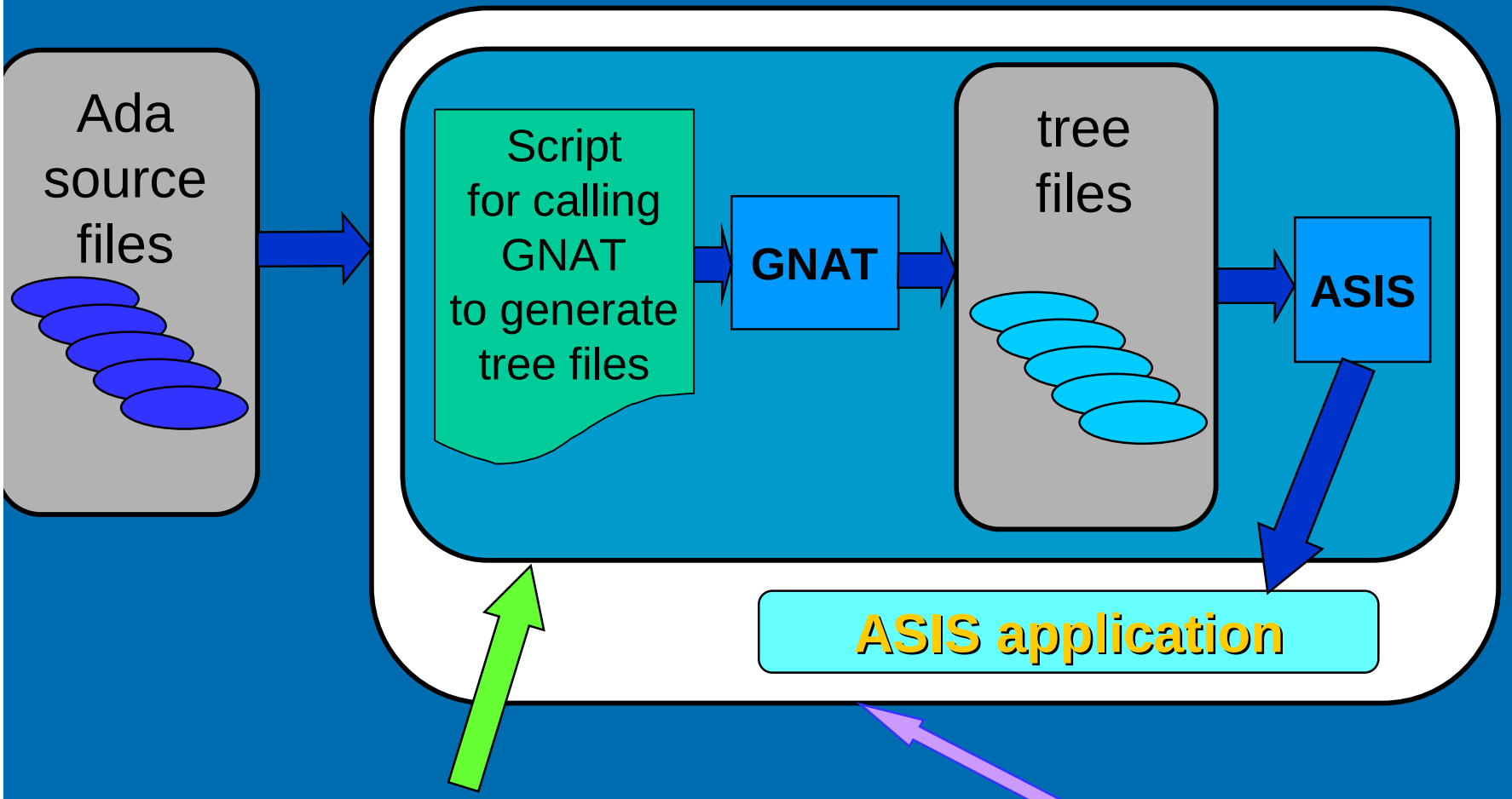
- ✔ **Completeness:**
  - Corresponds to ISO ASIS standard (some dusty corners remain, but the main parts of the ASIS 95 functionality are implemented and extensively tested)
- ✔ **Portability:**
  - Does not contain any platform-specific components;
  - Available for all supported platforms;
- ✔ **Quality control:**
  - Testing suite based on ACVC 2.1
- ✔ **Availability:**
  - Available as a fully-supported ACT product
  - Public releases are available for the corresponding GNAT public releases
- ✔ **There are plans for new tools and secondary/extension ASIS libraries**

ASIS-for-GNAT: technical support

- ✔ **Customer releases:**
  - **Latest versions of the GNAT/ASIS technology;**
  - **Implementation of special extensions by customer request;**
- ✔ **Consulting in ASIS, ASIS application development, ASIS83-to-ASIS95 tool porting;**
- ✔ **Bug fixing and providing workarounds for critical problems;**
- ✔ **ASIS training**
  - **Hands-on ASIS tutorials;**
  - **On-site training;**
- ✔ **Development of ASIS-based tools**

sales@act-europe.fr
sales@gnat.com

ASIS-for-GNAT as a Stand-Alone ASIS



Using asistant to learn ASIS and to debug your ASIS applications

- ✔`asistant` **is a part of the ASIS-for-GNAT user interface;**
- ✔`asistant` **provides the following possibilities:**
  - **interactive calls to ASIS queries;**
  - **easy recovering in case of failures;**
  - **logging interactive sections and reusing them;**
  - **interactive browsing of the ASIS tree;**
  - **online help on ASIS queries**
- ✔**Recommended use of asistant**
  - **for learning ASIS and mastering the ASIS technology;**
  - **for investigating the ASIS-related problems when working on ASIS tools;**

Some asistant details

- ✔ asistant uses a simple functional language:

```
SET (e, Unit_Declaration (cu))
IF (NOT (Is_Nil (e)), PRINT (e))
```

- ✔ Expression types: Boolean, Integer, String, and *all ASIS private types*
- ✔ All ASIS queries are directly callable
- ✔ There are a small set of service queries:
  - viewing the values of asisitant variables;
  - conditional execution;
- ✔ Batch execution, logs recording
- ✔ ASIS tree browsing

asistant on-line help


```
>help(is_identical)
Is_Identical syntax:
    Is_Identical (CUNIT, CUNIT) return BOOLEAN
    Is_Identical (ELEMENT, ELEMENT) return BOOLEAN

>help(a_loop_statement)
Appropriate ASIS structural queries for
   A_Loop_Statement:
    Label_Names
    Statement_Identifier
    Loop_Statements
```

Some `asistant` commands

- ✔ SET (<ID>[, <expr>])
  - *-- create new variable, set to expr*
- ✔ IF (<bool-expr>,<expr1>[,<expr2>])
  - *-- compute expr1 or expr2*
- ✔ HELP (<topic-name>)
  - *-- show online help for the topic*
- ✔ RUN [("<filename>")]
  - *-- launch / resume script*
- ✔ PAUSE
  - *-- pause script*
- ✔ PRINT (<expression>)
  - *-- evaluate and print expression*

Script example: startup.scr

✔ **Purpose: Prepares a unit for browsing**

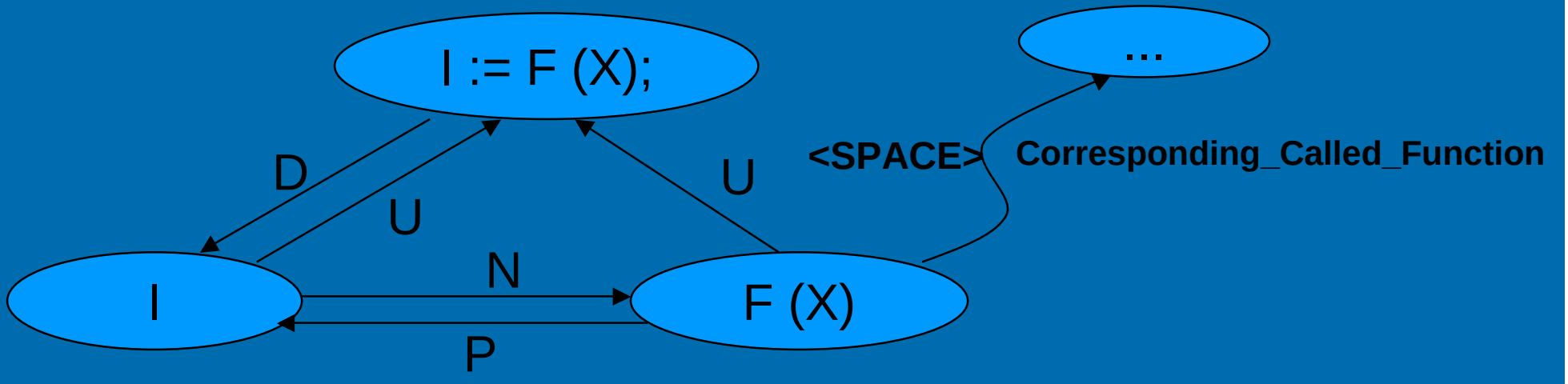
```
Initialize ("")
SET (ctx)
Associate (ctx, "", "")
Open (ctx)
IF (body,
    SET (cu,
        Compilation_Unit_Body
            (name, ctx)),
     SET (cu,
        Library_Unit_Declaration
            (name, ctx)))
 SET (u, Unit_Declaration (cu))
 SET (e, BROWSE (u))
```

✔ **Usage:**

```
SET (name, "Dummy")
SET (body, true)
RUN ("startup.scr")
```

– **start browsing from the top of the package body Dummy**

ASIS tree browser



- ✔ Implemented as an `asistant` Browse query
- ✔ U, D, P, N will move up, down, to the previous or to the next position in an ASIS tree
- ✔ SPACE allows to enter any query leading from Element to Element

ASIS-for-GNAT is ***THE solution*** for highly portable free and proprietary software analysis tools