SOFTWARE DEVELOPMENT

Franco Gasperoni
gasperon@act-europe.fr

Laurent Pautet
pautet@enst.fr

© ACT Europe under the GNU Free Documentation License

Copyright Notice

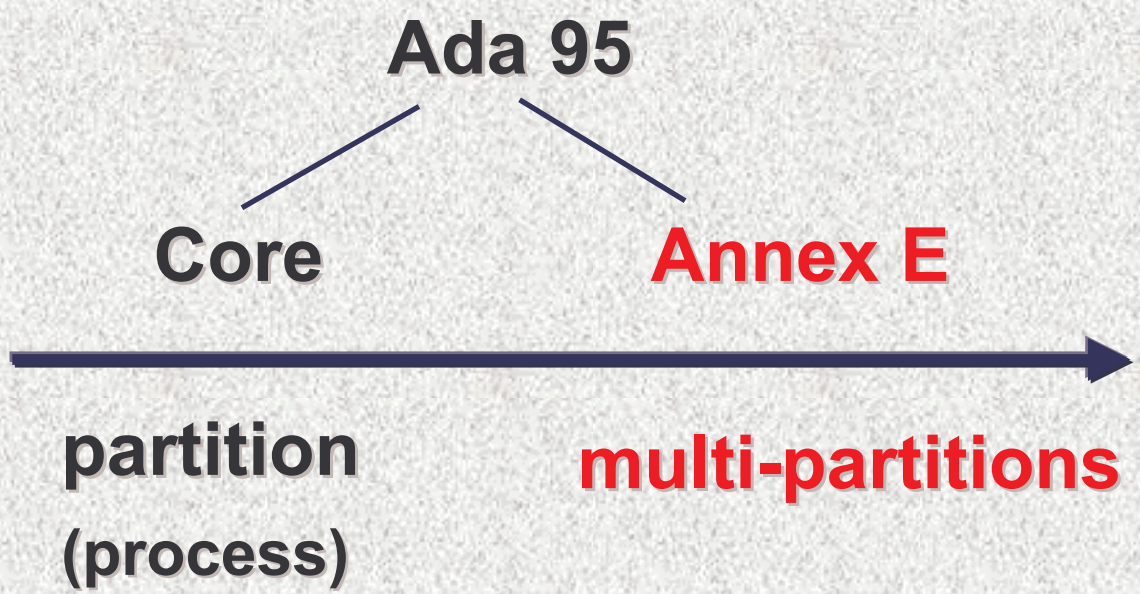
- © ACT Europe under the GNU Free Documentation License

- Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; provided its original author is mentioned and the link to http://libre.act-europe.fr/ is kept at the bottom of every non-title slide. A copy of the license is available at:
- http://www.fsf.org/licenses/fdl.html

© ACT Europe under the GNU Free Documentation License

Ada 95
Distributed Systems Annex

Ada 95 Distributed Programming



A partition comprises one or more Ada packages

© ACT Europe under the GNU Free Documentation License

Supported Paradigms

- Client/Server Paradigm (RPC)
  - Synchronous / Asynchronous
  - Static / Dynamic
- Distributed Objects
- Shared Memory

© ACT Europe under the GNU Free Documentation License

Ada Distributed Application

- No need for a separate interfacing language as in CORBA (IDL)
  - Ada is the IDL
- Some packages categorized using pragmas
  - Remote_Call_Interface (RCI)
  - Remote_Types
  - Shared_Passive (SP)
- All packages except RCI & SP duplicated on partitions using them

© ACT Europe under the GNU Free Documentation License

Remote_Call_Interface (RCI)

- Allows subprograms to be called remotely
  - Statically bound RPCs
  - Dynamically bound RPCs
    (remote access to subprogram)

© ACT Europe under the GNU Free Documentation License

Remote_Types

- Allows the definition of a remote access types
  - Remote access to subprogram
  - Remote reference to objects
    (ability to do dynamically dispatching calls across the network)

© ACT Europe under the GNU Free Documentation License

Shared_Passive

- A Shared_Passive package contains variables that can be accessed from distinct partitions
- Allows support of shared distributed memory
- Allows persistence on some implementations

© ACT Europe under the GNU Free Documentation License

Building a Distributed App in Ada 95

1. Write app as if non distributed.
2. Identify remote procedures, shared variables, and distributed objects & categorize packages.
3. Build & test non-distributed application.
4. Write a configuration file for partitionning your app.
5. Build partitions & test distributed app.

© ACT Europe under the GNU Free Documentation License

Remote_Call_Interface

An Example

© ACT Europe under the GNU Free Documentation License

Write App

```
package Types is
   type Device is (Furnace, Boiler,...);
   type Pressure is ...;
   type Temperature is ...;
end Types;
```

```
with Types; use Types;
package Sensors is
   function Get_P (D: Device) return Pressure;
   function Get_T (D: Device) return Temperature;
end Sensors;
```

```
with Types; use Types;
with Sensors;
procedure Client_1 is
   P := Sensors.Get_P (Boiler);
```

```
with Types; use Types;
with Sensors;
procedure Client_2 is
   T := Sensors.Get_T (Furnace);
```


© ACT Europe under the GNU Free Documentation License

Categorize

```
package Types is
   pragma Pure;
   type Device is (Furnace, Boiler,...);
   type Pressure is ...;
   type Temperature is ...;
end Types;
```

```
with Types; use Types;
package Sensors is
   pragma Remote_Call_Interface;
   function Get_P (D:Device) return Pressure;
   function Get_T (D:Device) return Temperature;
end Sensors;
```

```
with Types; use Types;
with Sensors;
procedure Client_1 is
   P := Sensors.Get_P (Boiler);
```

```
with Types; use Types;
with Sensors;
procedure Client_2 is
   T := Sensors.Get_T (Furnace);
```


© ACT Europe under the GNU Free Documentation License

Build & Test

```
package Types is
    pragma Pure;
    type Device is (Furnace, Boiler,...);
    type Pressure is ...;
    type Temperature is ...;
end Types;
```

```
with Types; use Types;
package Sensors is
    pragma Remote_Call_Interface;
    function Get_P (D:Device) return Pressure;
    function Get_T (D:Device) return Temperature;
end Sensors;
```

```
with Types; use Types;
with Sensors;
procedure Client_1 is
   P := Sensors.Get_P (Boiler);
```

Build & Test

```
package Types is
   pragma Pure;
   type Device is (Furnace, Boiler,...);
   type Pressure is ...;
   type Temperature is ...;
end Types;
```

```
with Types; use Types;
package Sensors is
   pragma Remote_Call_Interface;
   function Get_P (D:Device) return Pressure;
   function Get_T (D:Device) return Temperature;
end Sensors;
```

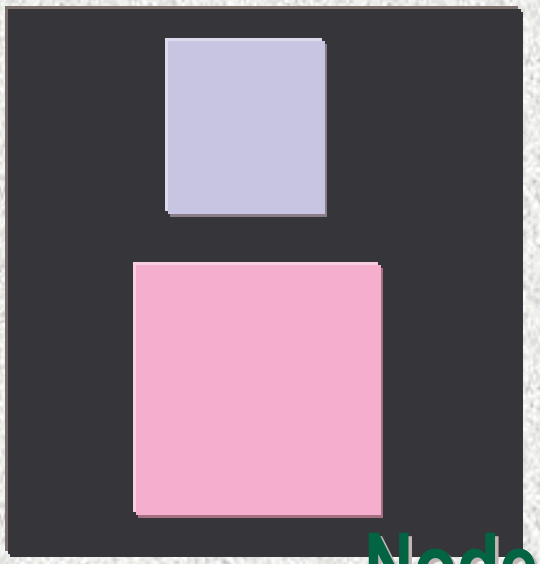
```
with Types; use Types;
with Sensors;
procedure Client_2 is
   T := Sensors.Get_T (Furnace);
```

Partition

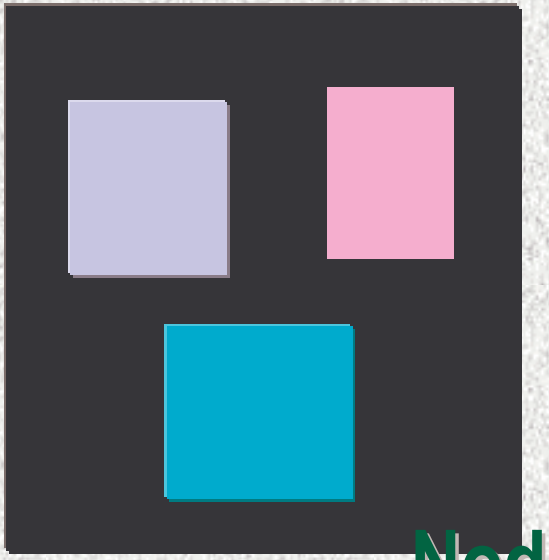
```
configuration Config_1 is
    Node_A : Partition := (Sensors);
    Node_B : Partition := (Client_1);
    Node_C : Partition := (Client_2);
end Config_1;
```


© ACT Europe under the GNU Free Documentation License

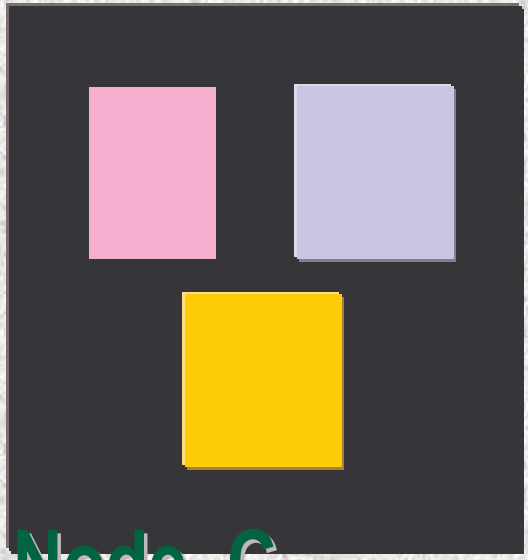
Partition



Node_A



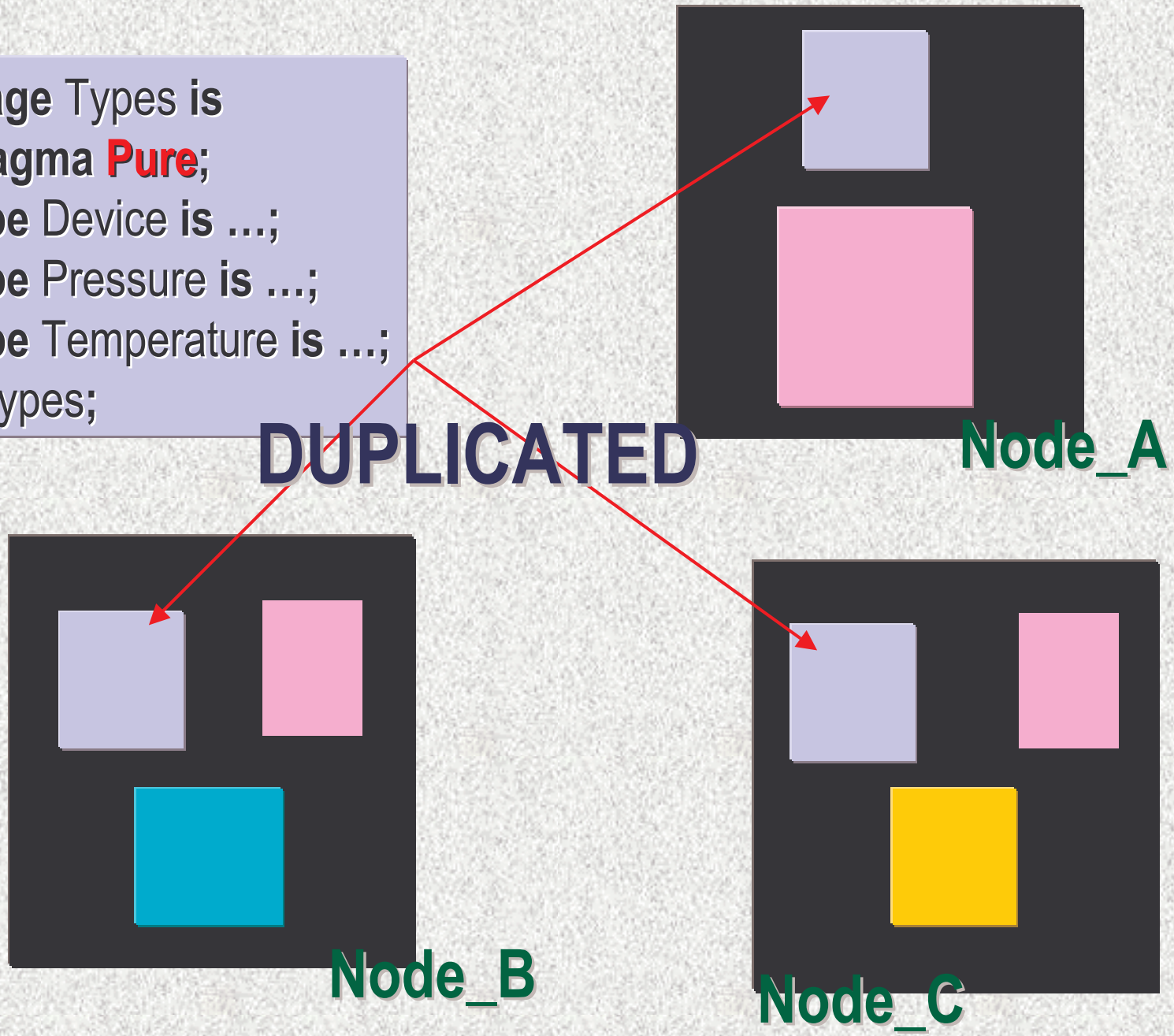
Node_B



Node_C

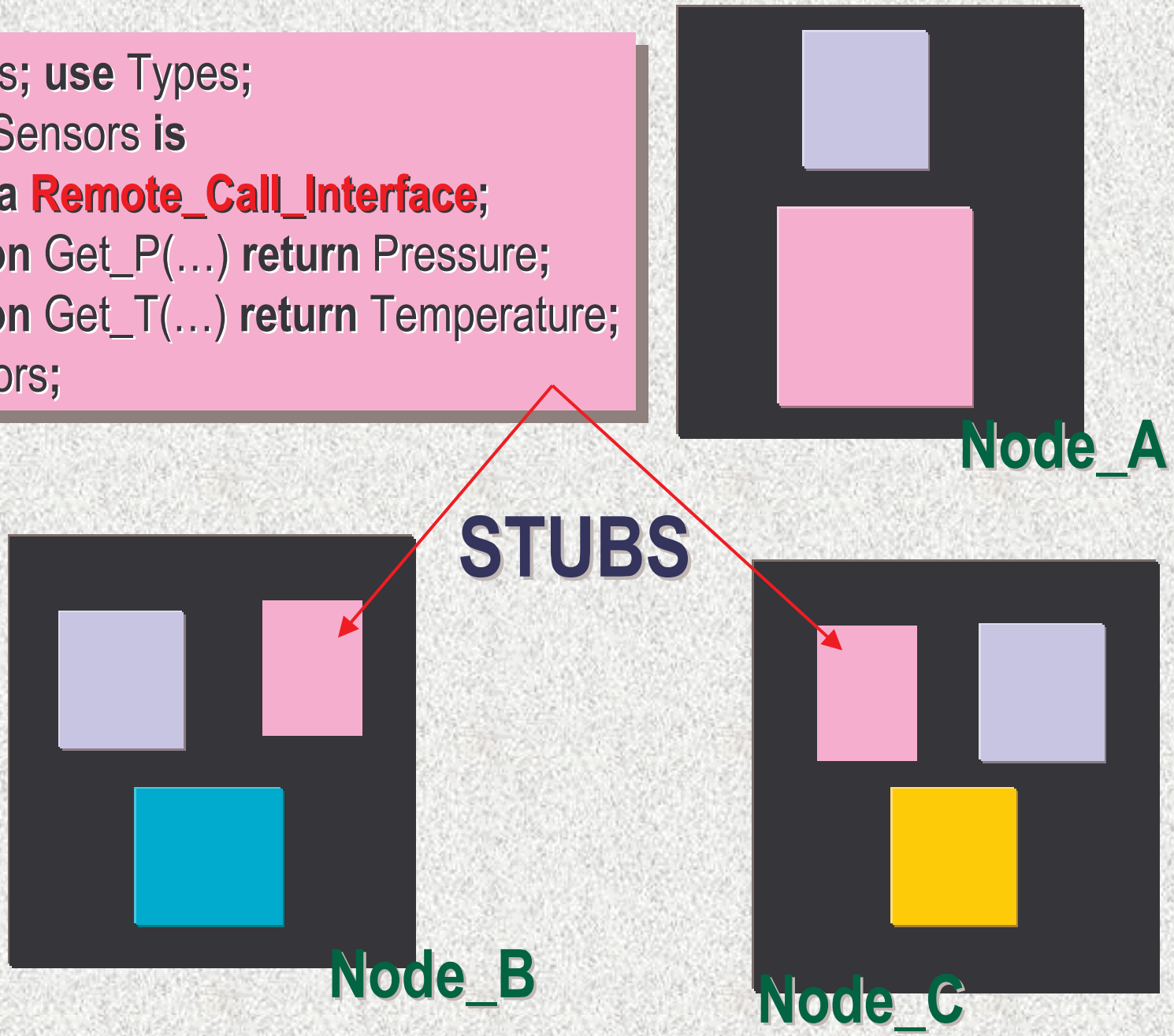
© ACT Europe under the GNU Free Documentation License

```
package Types is
   pragma Pure;
   type Device is ...;
   type Pressure is ...;
   type Temperature is ...;
end Types;
```



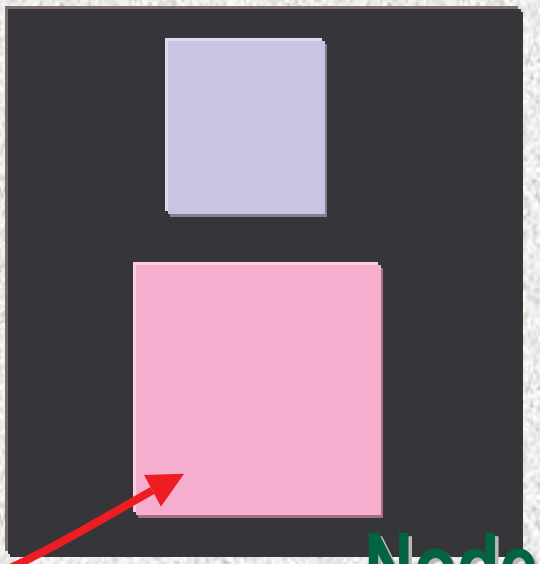
© ACT Europe under the GNU Free Documentation License

```
with Types; use Types;
package Sensors is
   pragma Remote_Call_Interface;
   function Get_P(…) return Pressure;
   function Get_T(…) return Temperature;
end Sensors;
```



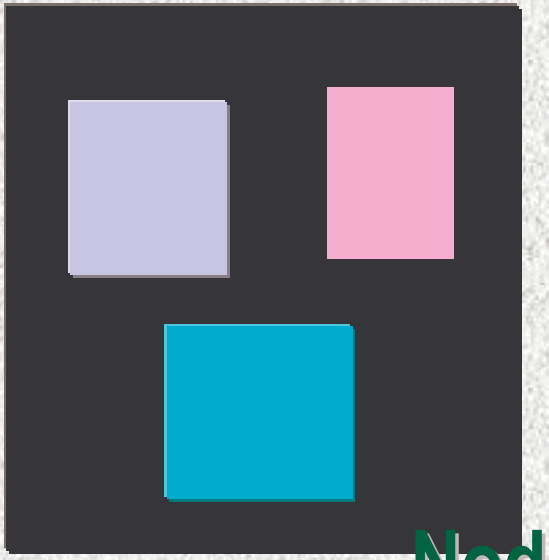
© ACT Europe under the GNU Free Documentation License

```
with Types; use Types;
package Sensors is
   pragma Remote_Call_Interface;
   function Get_P(…) return Pressure;
   function Get_T(…) return Temperature;
end Sensors;
```

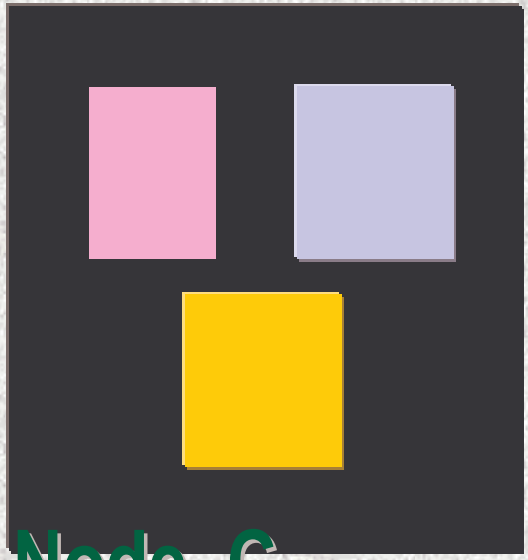


Node_A

SKELETON
+ BODY

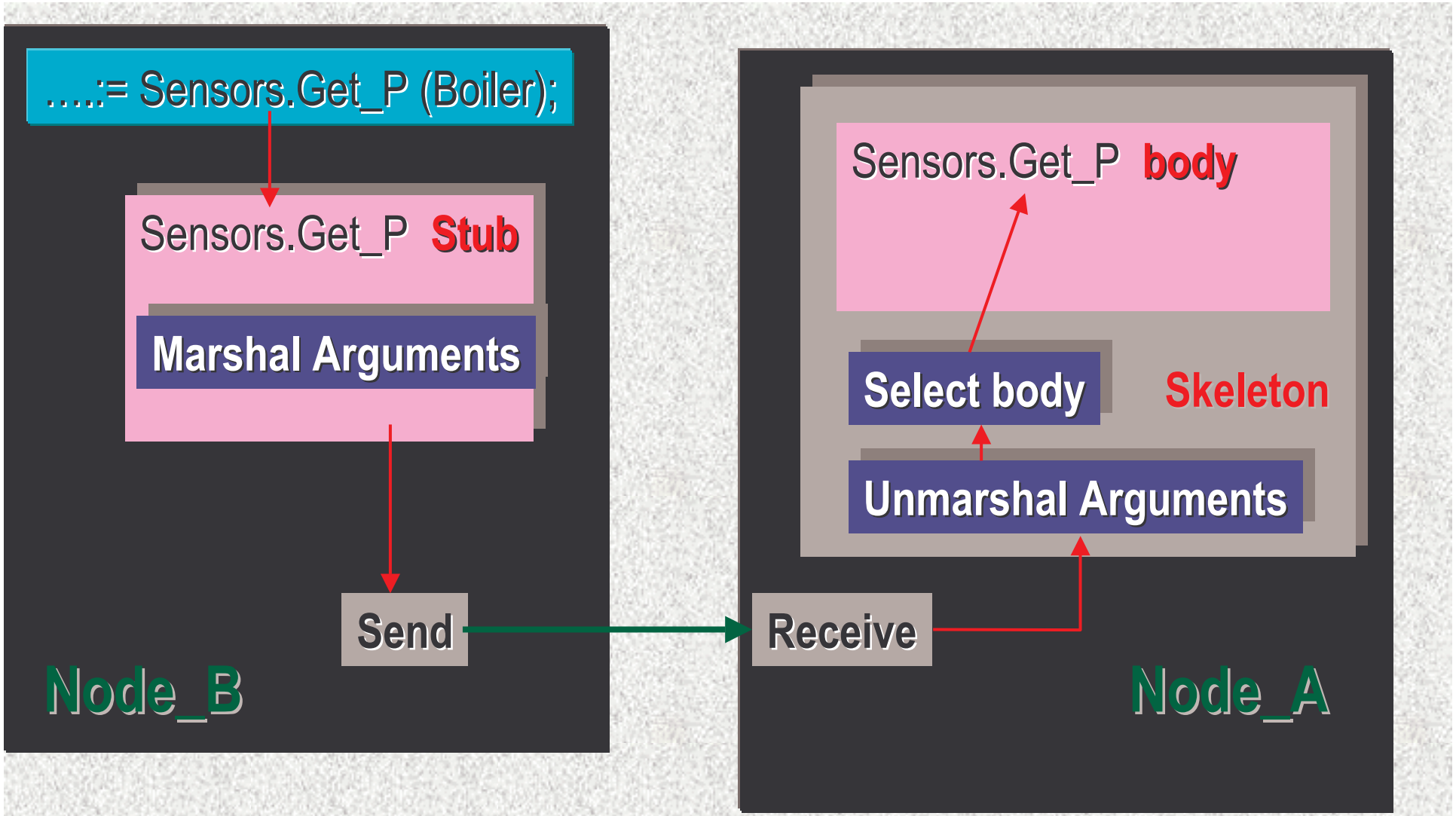


Node_B



Node_C

© ACT Europe under the GNU Free Documentation License

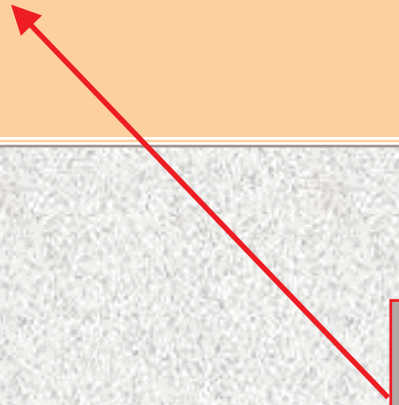


© ACT Europe under the GNU Free Documentation License

Asynchronous Procedure Calls

```
with Types; use Types;
package Sensors is
   pragma Remote_Call_Interface;
   ...
   procedure Log (D : Device; P : Pressure);
   pragma Asynchronous (Log);

end Sensors ;
```



+ returns immediately
+ exceptions are lost
+ parameters must be in

© ACT Europe under the GNU Free Documentation License

Issues with asynchronous calls

```
with RCI1, RCI2;
procedure Client is
   RCI1.P (1); RCI2.P (2)
```

```
package RCI1 is
   pragma Remote_Call_Interface;
   procedure P (X : Integer);
   pragma Asynchronous;
end RCI1;
```

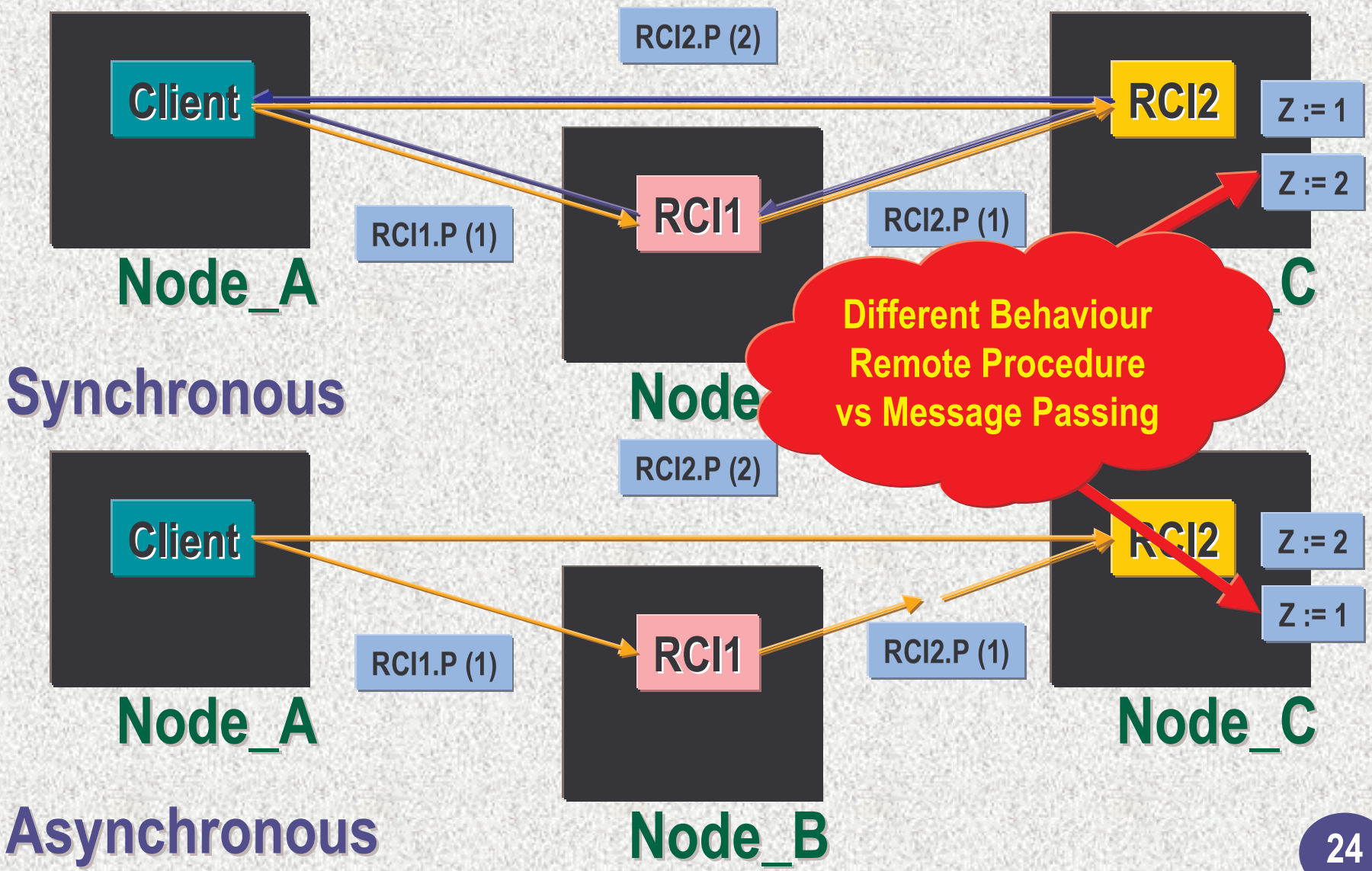
```
package RCI2 is
   pragma Remote_Call_Interface;
   procedure P (X : Integer);
   pragma Asynchronous;
end RCI2;
```

```
with RCI2;
package body RCI1 is
   procedure P (X : Integer) is
   begin
      RCI2.P (X);
   end P;
end RCI1;
```

```
package body RCI2 is
   Z : Integer := 0;
   procedure P (X : Integer) is
   begin
      Z := X;
   end P;
end RCI2;
```


© ACT Europe under the GNU Free Documentation License

Synchronous vs Asynchronous



© ACT Europe under the GNU Free Documentation License

Remote_Types

An Example

© ACT Europe under the GNU Free Documentation License

Write App

```
package Alerts is
      type Alert is abstract tagged limited private;
      procedure Handle (A : access Alert);
      procedure Log      (A : access Alert) is abstract;
 private
      ...
end Alerts;
```

```
package Alerts.Pool is
      type Alert_Ref is access all Alert'Class;
      procedure Register (A : Alert_Ref);
      function    Get_Alert return Alert_Ref;
end Alerts.Pool ;
```

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
     loop
          Handle (Pool.Get_Alert);
     end loop;
end Process_Alerts;
```


©

```
package Alerts.Low is
      type Low_Alert is new Alert with private;
      procedure Log (A : access Low_Alert);
private
      ...
end Alerts.Low;
```

```
with Alerts.Pool; use Alerts.Pool;
package body Alerts.Low is
      ...
begin
      Register (new Low_Alert);
end Alerts.Low;
```


© ACT Europe under the GNU Free Documentation License

```
package Alerts.Medium is
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```

```
with Alerts.Pool; use Alerts.Pool;
package body Alerts.Medium is
   ...
begin
   Register (new Medium_Alert);
end Alerts.Medium;
```


© ACT Europe under the GNU Free Documentation License

Categorize

```
package Alerts is
   pragma Pure;
   type Alert is abstract tagged limited private;
   procedure Handle (A : access Alert);
   procedure Log    (A : access Alert) is abstract;
private
```

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   type Alert_Ref is access all Alert'Class;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Alerts.Pool ;
```

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
   loop
      Handle (Pool.Get_Alert);
   end loop;
end Process_Alerts;
```

```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

```
package Alerts.Medium is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```


© ACT Europe under the GNU Free Documentation License

Build & Test

```
package Alerts is
   pragma Pure;
   type Alert is abstract tagged limited private;
   procedure Handle (A : access Alert);
   procedure Log    (A : access Alert) is abstract;
private
   ...
end Alerts;
```

```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

```
package Alerts.Medium is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   type Alert_Ref is access all Alert'Class;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Alerts.Pool ;
```

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
   loop
      Handle (Pool.Get_Alert);
   end loop;
end Process_Alerts;
```

Partition

```
configuration Config_2 is
    Node_AL  : Partition := (Alerts.Low);
    Node_AM : Partition := (Alerts.Medium);
    Node_B    : Partition := (Alerts.Pool);
    Node_C    : Partition := (Process_Alerts);
end Config_2;
```


© ACT Europe under the GNU Free Documentation License

What Happens When Executing the Distributed Program ?

© ACT Europe under the GNU Free Documentation License

```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

Node_AL

```
package Alerts.Medium is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```

Node_AM

Step 1: A Low_Alert object in Node_AL registers itself with Node_B

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   type Alert_Ref is access all Alert'Class;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

Node_B

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
   loop
      Handle (Pool.Get_Alert);
   end loop;
end Process_Alerts;
```

Node_C

© ACT Europe under the GNU Free Documentation License

```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

Node_AL

```
package Alerts.Medium is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```

Node_AM

Step 2: A Medium_Alert object in Node_AM registers itself with Node_B

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   type Alert_Ref is access all Alert'Class;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

Node_B

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
   loop
      Handle (Pool.Get_Alert);
   end loop;
end Process_Alerts;
```

Node_C

© ACT Europe under the GNU Free Documentation License

```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

Node_AL

```
package Alerts.Medium is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```

Node_AM

Step 3: Process_Alerts in Node_C does an RPC to Get_Alert in Node_B

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   type Alert_Ref is access all Alert'Class;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

Node_B

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
   loop
      Handle (Pool.Get_Alert);
   end loop;
end Process_Alerts;
```

Node_C

© ACT Europe under the GNU Free Documentation License

```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

Node_AL

```
package Alerts.Medium is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```

Node_AM

Step 4: Get_Alert returns a pointer to an Alert object (Low_Alert or Medium_Alert)

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   type Alert_Ref is access all Alert'Class;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

Node_B

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
   loop
      Handle (Pool.Get_Alert);
   end loop;
end Process_Alerts;
```

Node_C

© ACT Europe under the GNU Free Documentation License

```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

Node_AL

?

```
package Alerts.Medium is
   pragma Remote_Types;
   type Medium_Alert is new Alert with private;
   procedure Handle (A : access Medium_Alert);
   procedure Log    (A : access Medium_Alert);
private
   ...
end Alerts.Medium;
```

Node_AM

Step 5: Node_C performs a dispatching RPC. It calls Handle in Node_AL or Node_AM

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   type Alert_Ref is access all Alert'Class;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

Node_B

```
with Alerts, Alerts.Pool; use Alerts;
procedure Process_Alerts is
begin
   loop
      Handle (Pool.Get_Alert);
   end loop;
end Process_Alerts;
```

Node_C

© ACT Europe under the GNU Free Documentation License

Asynchronous Method Invocation

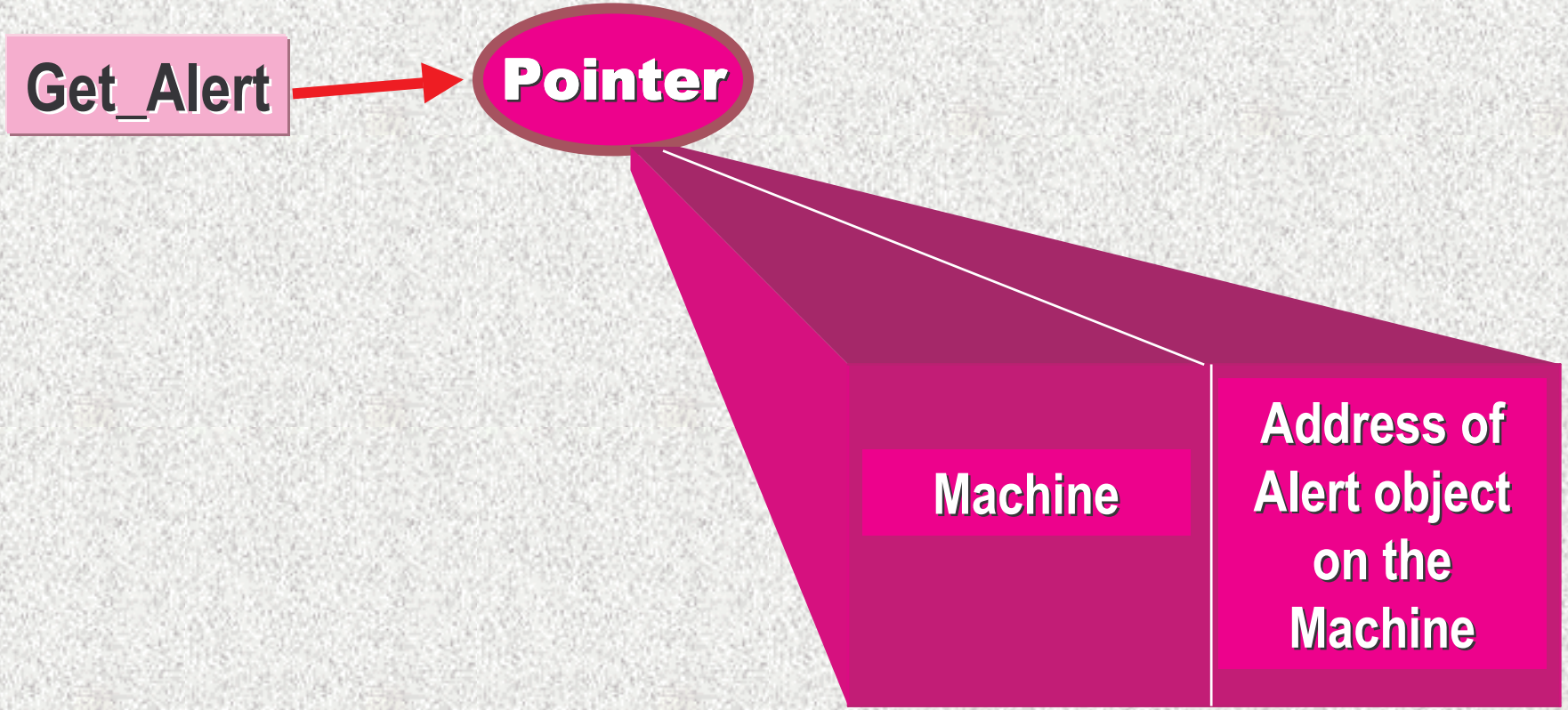
```
package Alerts.Pool is
      pragma Remote_Call_Interface;
      type Alert_Ref is access all Alert'Class;
      pragma Asynchronous (Alert_Ref);
      procedure Register (A : Alert_Ref);
      function    Get_Alert return Alert_Ref;
end Alerts.Pool ;
```

```
package Alerts is
      pragma Pure;
      type Alert is abstract tagged limited private;
      procedure Handle (A : access Alert);
      procedure Log      (A : access Alert) is abstract;
 private
      ...
end Alerts;
```

+ returns immediately
+ exceptions are lost
+ parameters must be in

© ACT Europe under the GNU Free Documentation License

What Does Get_Alert Return ?



© ACT Europe under the GNU Free Documentation License

Remote Access to Class Wide Type

- **At compile time:**
  - **You do not know what operation you'll dispatch to**
  - **On what node that operations will be executed on**

© ACT Europe under the GNU Free Documentation License

Shared_Passive

An Example

© ACT Europe under the GNU Free Documentation License

Shared Data

```
with Types; use Types;
package Sensors is
   pragma Shared_Passive;
   type State is record
      Open  : Boolean := False;
      P     : Pressure := 0;
      T     : Temperature := 0;
   end record;
   States : array (1 .. 16) of State;
   protected Controller is
      procedure Open (I : out Natural);
      procedure Close (I : in Natural);
   end Controller;
end Sensors;
```

```
with Sensors; use Sensors;
procedure Update_State is
   S: Natural;
begin
   Controller.Open (S);
   while Active loop
      States (S) := Get_State;
   end loop;
   Controller.Close (S);
end Update_State;
```

+ shared objects

+ shared objects
+ atomic operations

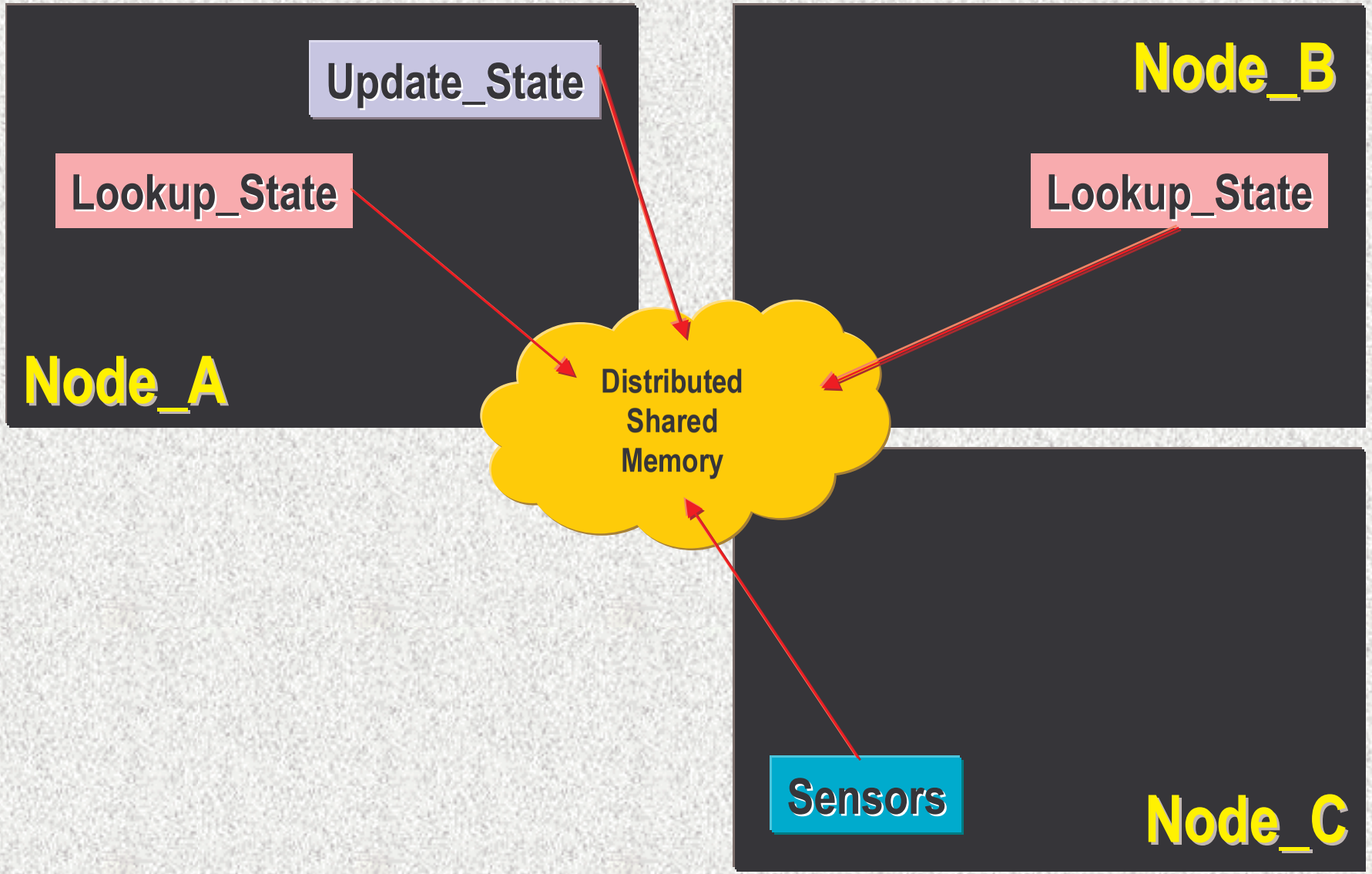
© ACT Europe under the GNU Free Documentation License

```
configuration Config_3 is
   Producer   : Partition := (Update_State);
   Consumer   : Partition := (Lookup_State);
   Whiteboard : Partition := (Sensors);
end Config_3;
```


© ACT Europe under the GNU Free Documentation License

What Happens When Sharing Data ?

© ACT Europe under the GNU Free Documentation License



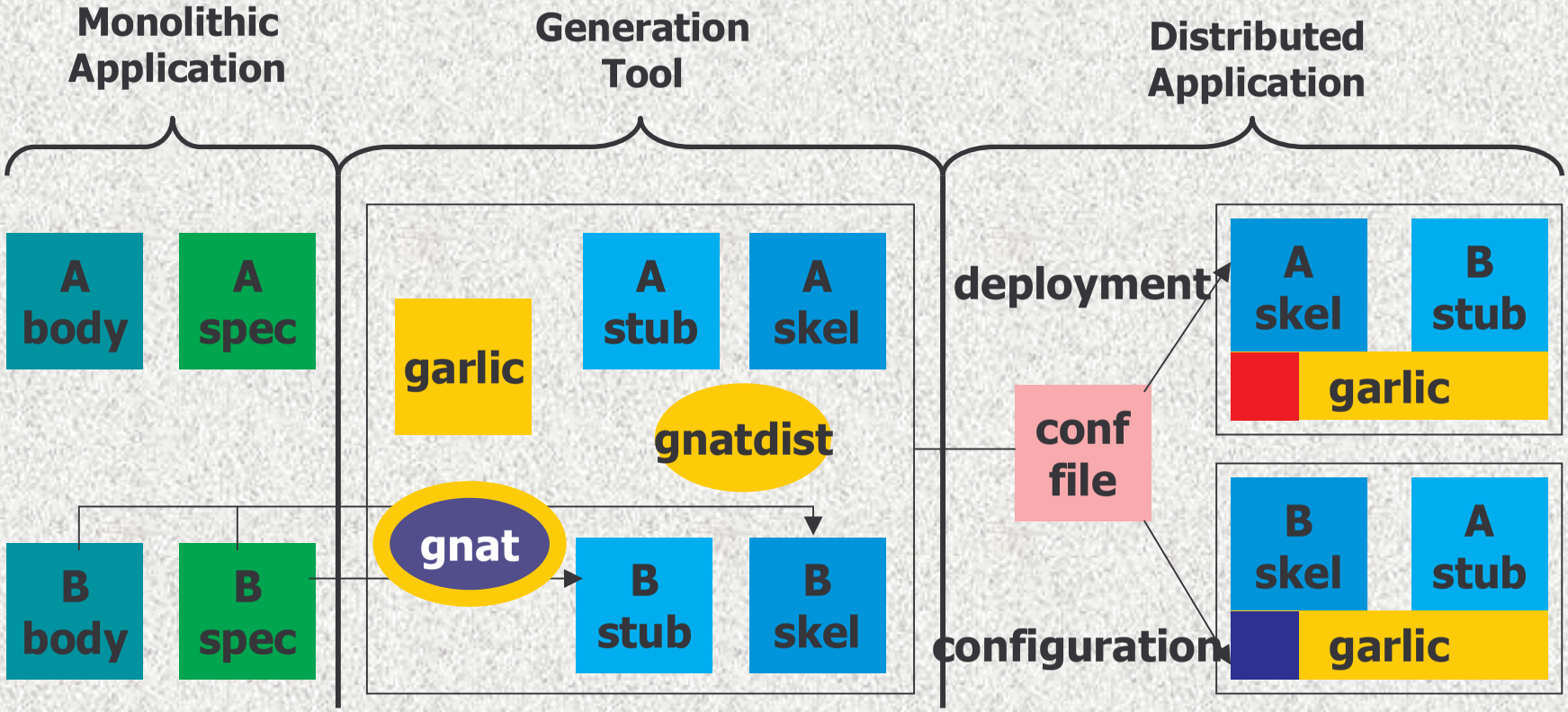
© ACT Europe under the GNU Free Documentation License

Partitioning Tool GLADE

- GNAT compiler
  - check categorized packages
  - generate stubs and skeletons
- GARLIC middleware
  - process request (invocation, abortion, exception, …)
  - provide internal services (naming, termination, …)
- GNATDIST partitioning tool
  - configure transport protocols or storage supports
  - configure run-time (concurrent requests, task pools, …)
  - configure components (filters, versioning, termination, …)

© ACT Europe under the GNU Free Documentation License

How to partition a distributed application



© ACT Europe under the GNU Free Documentation License

Conclusions

- Ada Distributed Systems Annex
  - Close to RMI (Ada talking to Ada)
  - Many distribution mechanisms
    - Message Passing (Asynchronous Remote Invocation)
    - Remote Procedures
    - Remote Objects
    - Shared Objects
- GLADE
  - Distribution code generation
  - Middleware configuration
  - Distributed application smart deployment

© ACT Europe under the GNU Free Documentation License