Remote_Types

An Example

Write App

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

```
package Alerts.Pool is
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

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



```
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;
```



```
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;
```

Build & Test

```
package Alerts is
   pragma Remote_Types;
   type Alert is abstract tagged private;
   type Alert_Ref is access all Alert'Class;
   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;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

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

Categorize

```
package Alerts is
   pragma Remote_Types;
   type Alert is abstract tagged private;
   type Alert_Ref is access all Alert'Class;
   procedure Handle (A : access Alert);
   procedure Log    (A : access Alert) is abstract;
private
```

```
package Alerts.Pool is
   pragma Remote_Call_Interface;
   procedure Register (A : Alert_Ref);
   function  Get_Alert return Alert_Ref;
end Medium;
```

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


Partition

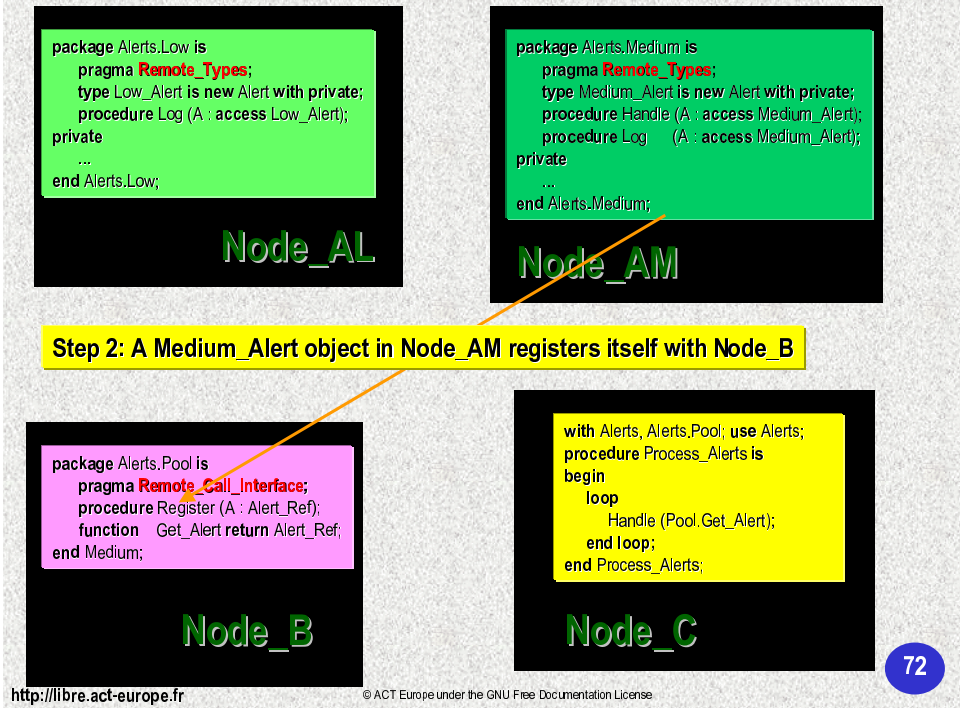
```
configuration Config_2 is
   Node_AL  : Partition := (Alerts.Low);
   Node_AM : Partition := (Alerts.Medium);
   Node_B   : Partition := (Alerts.Pool);
   Node_C   : Partition := (Process_Alerts);
end Config_2;
```

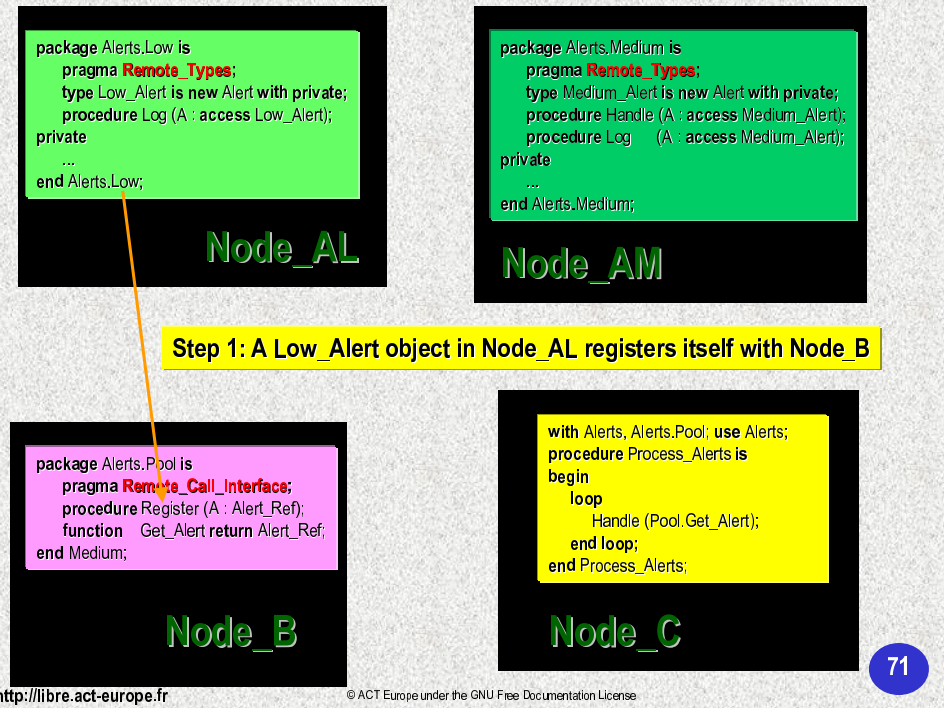


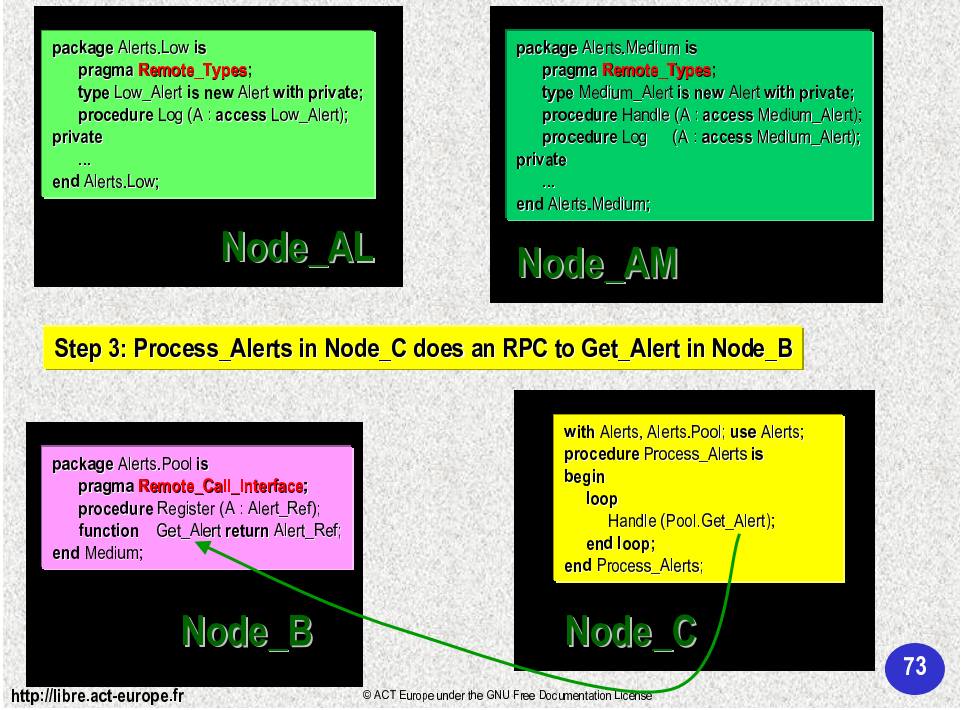
```
package Alerts.Low is
   pragma Remote_Types;
   type Low_Alert is new Alert with private;
   procedure Log (A : access Low_Alert);
private
   ...
end Alerts.Low;
```

```
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;
```

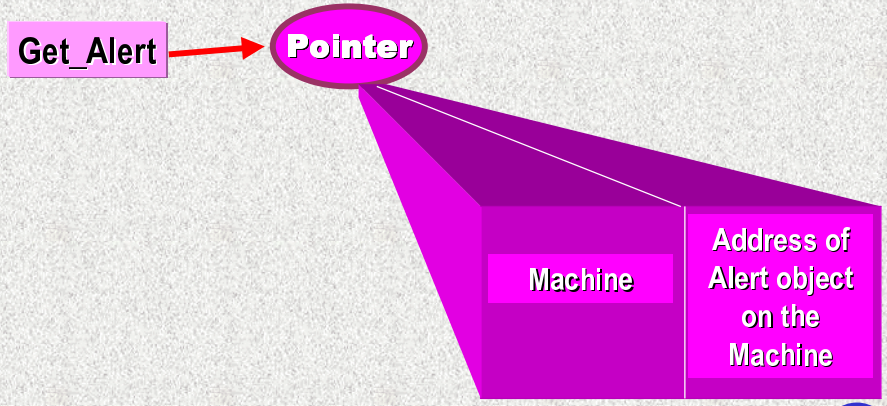
What Happens When Executing the Distributed Program ?

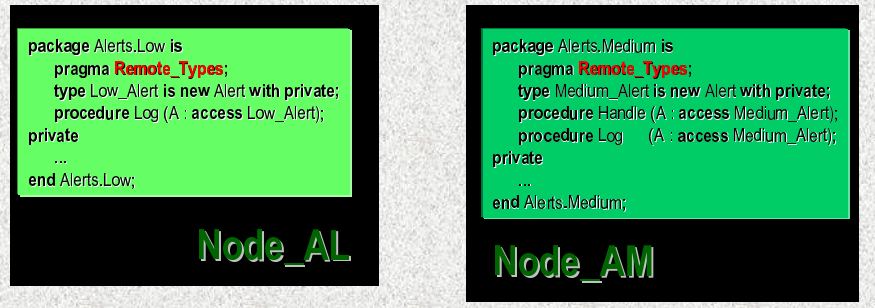




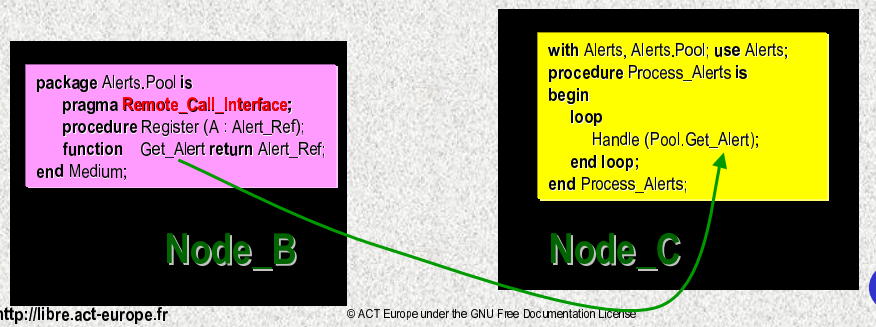


What Does Get_Alert Return ?



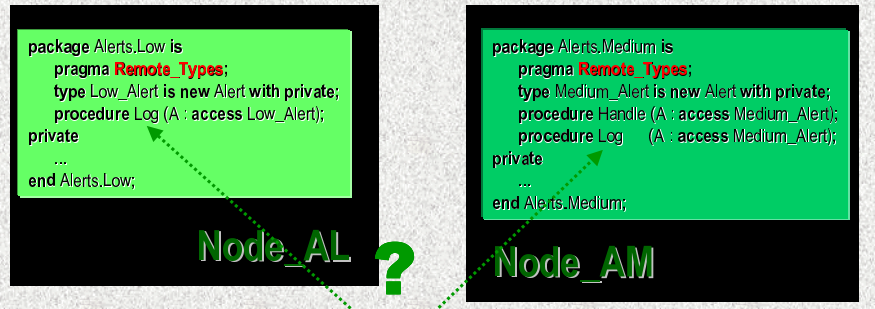


Step 4: Get_Alert returns a pointer to an Alert object (Low_Alert or Medium_Alert)



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



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