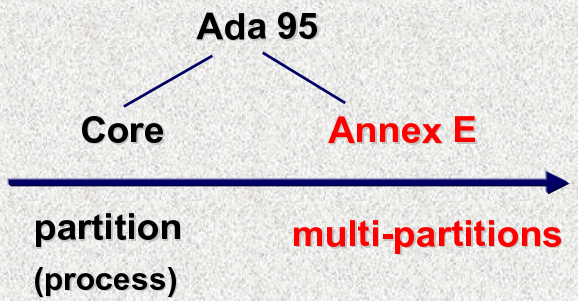
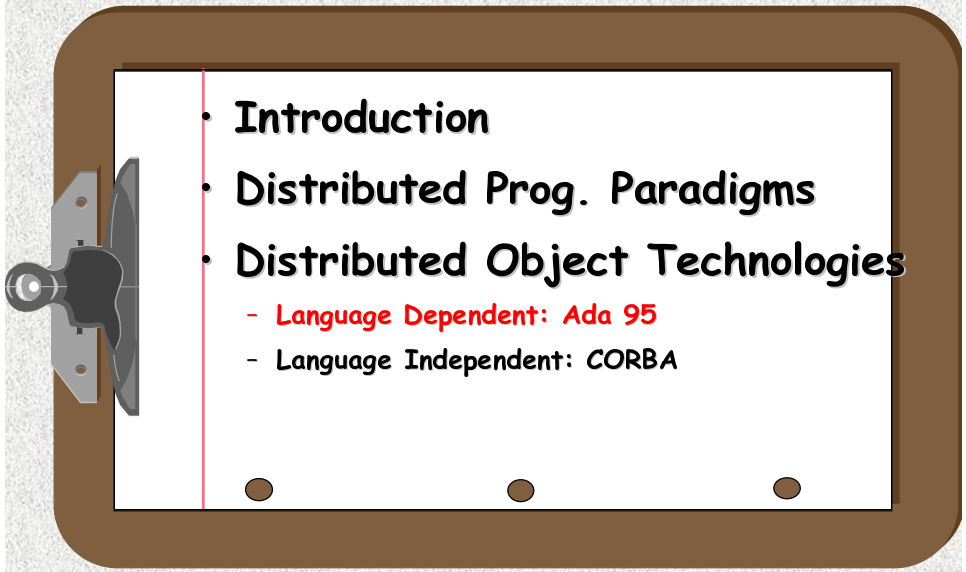
Ada 95 Distributed Programming



A partition comprises one or more Ada packages



Supported Paradigms

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

Ada 95 Distributed Systems Annex

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)

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

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

Remote_Call_Interface (RCI)

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

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);
```


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.

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);
```


Remote_Call_Interface

An Example

Partition

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

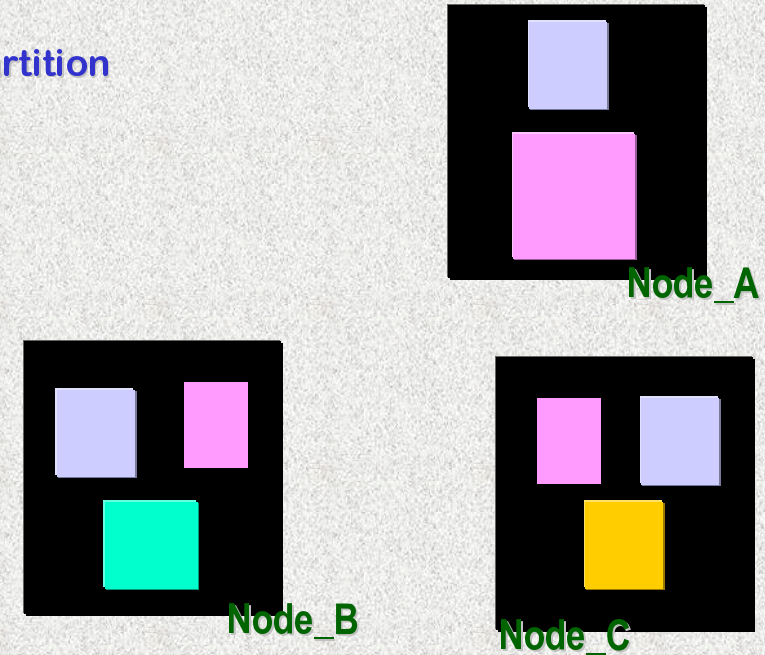

Build & Test

```
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);
```


Partition



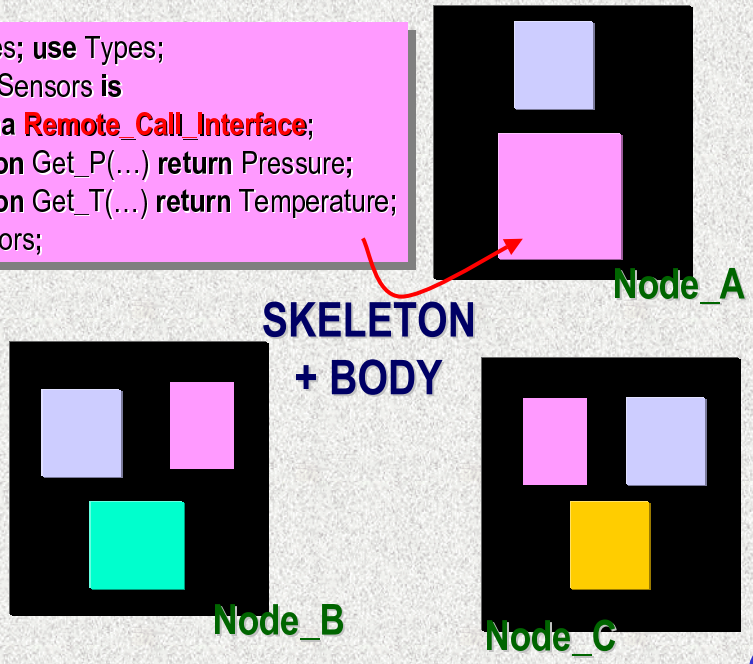
Build & Test

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

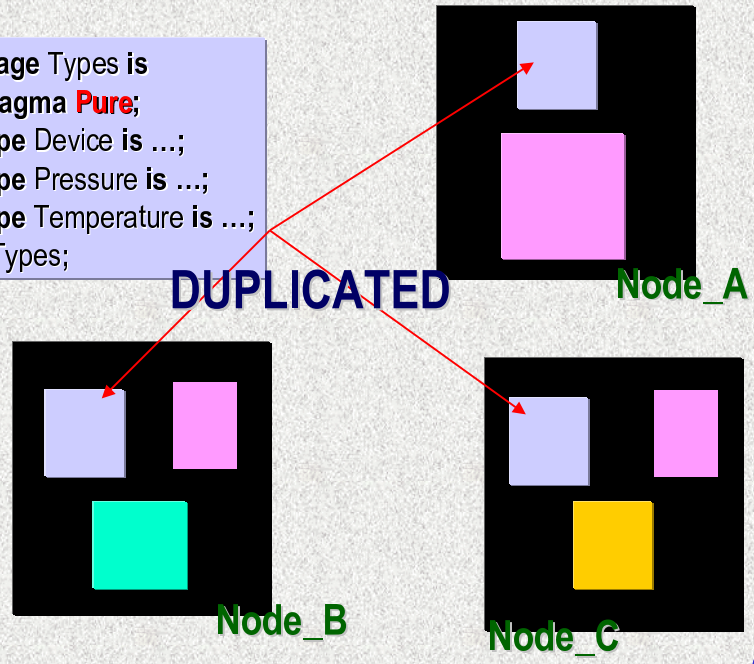
```
with Types; use Types;
package Sensors is
   pragma Remote_Call_Interface;
   function Get_P (D:Device) return Pressure;
   function Get_T (D:Device) return Temperature;
end Sensors;
```

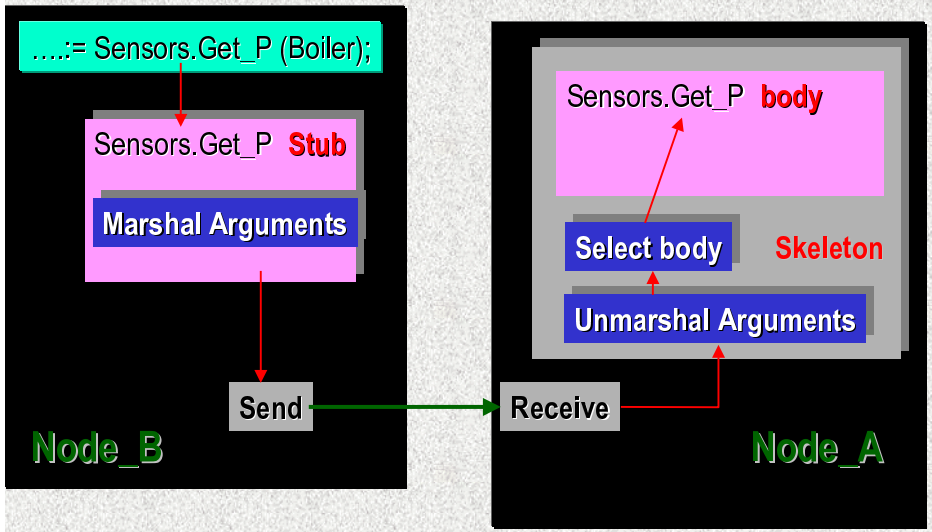
```
with Types; use Types;
with Sensors;
procedure Client_2 is
   T := Sensors.Get_T (Furnace);
```

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

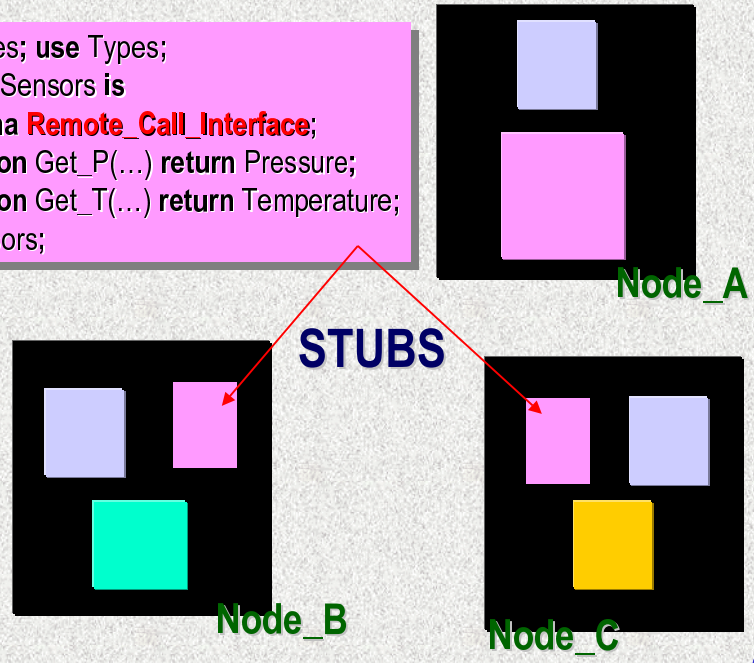



```
package Types is
   pragma Pure;
   type Device is ...;
   type Pressure is ...;
   type Temperature is ...;
end Types;
```





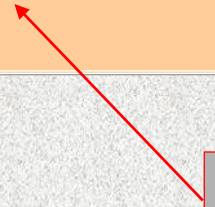

```
with Types; use Types;
package Sensors is
   pragma Remote_Call_Interface;
   function Get_P(...) return Pressure;
   function Get_T(...) return Temperature;
end Sensors;
```



Asynchronous Calls

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

end Bank;
```



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