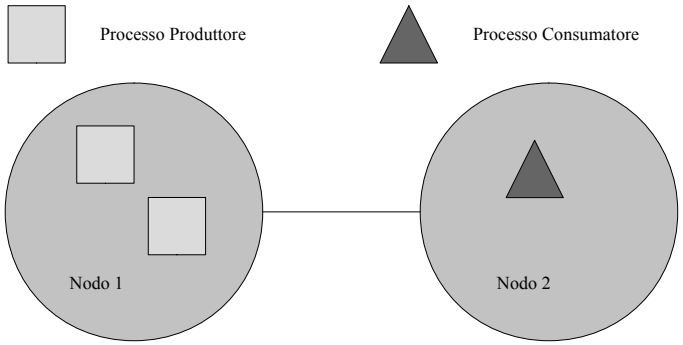
Trasparenza di distribuzione

- Il processo Produttore produce periodicamente un prodotto e lo invia in *push* al processo Consumatore
- Immaginiamo un sistema composto da
  - 2 Produttori
  - 1 Consumatore
- I processi Produttore risiedono su un nodo fisico distinto da quello del processo Consumatore
- Vogliamo trasparenza di distribuzione a livello dei processi

L'applicazione di esempio



La classe Produttore

```
type Producer is new Controlled and IProducer with record
  IC      : IConsumers.IConsumer_Ref;
  ID      : Datas.Producer_ID;
  Product : Datas.Data;
end record;

procedure Produce (This : in out Producer) is
begin
  …;
  This.IC.Consume (This.ID, This.Product);
end Produce;

procedure Set_IC
 (This : in out Producers.Producer;
  V    : IConsumers.IConsumer_Ref) is
begin
  if This.IC = null then
    This.IC := V;
  end if;
end Set_ic;
```

```
type IProducer is interface;
type IProducer_Ref is access all IProducer'Class;
procedure Produce (This : in out IProducer) is abstract;
```

Ogni istanza della classe Producer contiene un riferimento all'istanza del processo Consumer.

Per notificare al consumatore che deve risvegliarsi e consumare il prodotto basta invocare il metodo Consume sul riferimento della classe Consumer

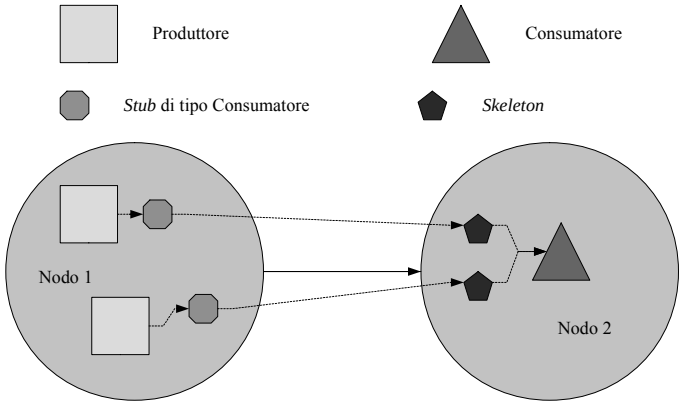
La classe Consumatore

```
procedure Consume
 (This    : in out Consumer;
  ID      : in Datas.Producer_ID;
  Product : in Datas.Data) is
begin
  ...
end Consume;
```

Deployment del sistema – 1

- Il codice funzionale delle due classi applicative deve godere di trasparenza di distribuzione
  - Non deve subire modifiche al variare della topologia del sistema
- La soluzione prevede l'associazione di uno *stub* e di uno *skeleton* dedicati alla connessione tra ciascun produttore e il consumatore
  - Lo *stub* viene posto sul nodo di residenza del corrispondente produttore
  - Lo *skeleton* su quello del consumatore *target*

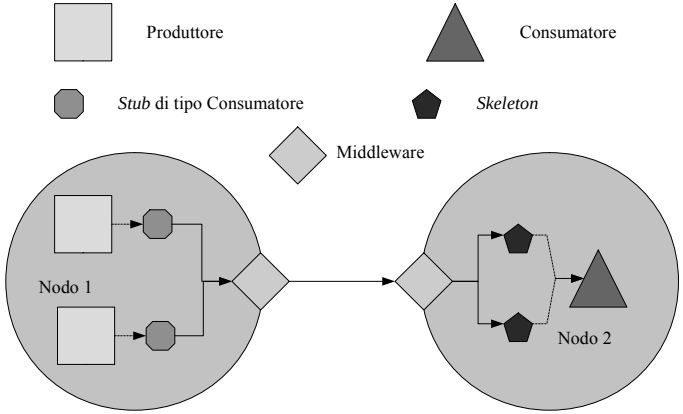
L'applicazione arricchita – 1



Il ruolo del *Middleware* – 1

- *Stub* e *skeleton* da soli non bastano per assicurare trasparenza di distribuzione
- Occorre che uno strato *Middleware* (un *broker*) risieda in ogni nodo fisico partecipante
  - Per permettere indicizzazione e localizzazione delle entità applicative del sistema
  - Per la serializzazione delle comunicazioni di livello applicazione
  - Per il trasporto delle comunicazioni

L'applicazione arricchita – 2



Deployment del sistema – 2

- Lo *stub* deve esporre lo stesso interfaccia del processo Consumatore ma realizzarlo utilizzando metodi di trasporto esposti dal Middleware
- Il processo Produttore deve incosapevolmente invocare il metodo dello *stub* invece che quello del processo Consumatore
  - Vogliamo ottenere questo effetto con il massimo di trasparenza (senza *lookup*)

Deployment del sistema – 3

```
ConsumerAPInst_Stub_For_Producer1 : aliased ConsumerAP_Stub.c1_T :=
    ConsumerAPInst_Stub.Get_c1_As_Consumer
      (Deployment.ProducerAPInst_producer1_produce_Cyclic);
ConsumerAPInst_Stub_For_Producer1_Access : access ConsumerAP_Stub.c1_T :=
    ConsumerAPInst_Stub_For_Producer1'Access;

ConsumerAPInst_Stub_For_Producer2 : aliased ConsumerAP_Stub.c1_T :=
    ConsumerAPInst_Stub.Get_c1_As_Consumer
      (Deployment.ProducerAPInst_producer2_produce_Cyclic);
ConsumerAPInst_Stub_For_Producer2_Access : access ConsumerAP_Stub.c1_T :=
    ConsumerAPInst_Stub_For_Producer1'Access;


Producer1_Instance : aliased Producer1_T;
Producer2_Instance : aliased Producer2_T;

Producer1_Instance.Set_IC (ConsumerAPInst_Stub_For_Producer1'Access);
Producer2_Instance.Set_IC (ConsumerAPInst_Stub_For_Producer2'Access);
```


L'attività dello *stub*

```
procedure Consume
    (From : Deployment.Entity_Type;
     ID : Datas.Producer_ID;
     Product : Datas.Data) is
  Request : ConsumerAP_Stub_Request (Consume_2);
  Msg : Message_Types.Message_Type;
  Success : Boolean;
begin
  Request.Consume_2_ID := ID;
  Request.Consume_2_Product := Product;
  ConsumerAP_Stub_Marshaller.Marshal (Request, Msg);
  VM_Middleware_Send
      (Deployment.Map.Get_Entity_Type
         (Deployment.Get_Corresponding_Skeleton
           (Deployment.Map.ConsumerAP_Instances_Table (VM_Stub_Of), From)),
       Deployment.Map.ConsumerAPTo_VM_Instances
         (VM_Stub_Of,
          Deployment.Map.C1_Consume),
       Msg,
       Success);
end Consume;
```

Look-up del mittente

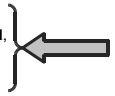
Look-up del destinatario

Middleware di lato destinatario

- Un processo Receiver per ogni connessione in ingresso proveniente da uno specifico nodo
  - Rispetto al problema dato verrà dunque creato un solo processo Receiver sul Nodo 2
- Per ogni messaggio in arrivo sulla connessione il processo Receiver determina lo *skeleton* di destinazione e gli gira il messaggio
  - Lo *skeleton* è realizzato come un processo

L'attività dello *skeleton*

```
  Request : ConsumerAP_Stub.Internals.ConsumerAP_Stub_Request;
begin
 ConsumerAP_Stub.Internals.
   ConsumerAP_Stub_Marshaller.Unmarshal(Request, Message);
 case Request.Operation is
  when ConsumerAP_Stub.Internals.Consume_2 =>
    Consumers.Consume
      (P2.ConsumerAPInst.Get_c1_as_Consumer.all,
       Request.Consume_2_ID,
       Request.Consume_2_Product);
  when others => <error handling>;
 end case;
....
```



Trasporto tramite *socket*

- Per ogni nodo del sistema viene creata una istanza di *Middleware* che svolge
  - Le funzioni di *Name Server*
  - Le funzione di aggancio con il livello trasporto
- Due processi specializzati si occupano di inizializzazione e ripristino del livello trasporto
  - In aggiunta al processo Receiver in servizio su ogni connessione in ingresso al nodo

Name Server

Tabella di *Naming* per il Nodo 2

```
DNS : constant Naming_Table_Type :=
  (Node1 =>
        (NodeAddress      => No_Sock_Addr, – no outgoing link to Node 1
         Sequence_Count => 0,
         QoS                  => NOT_GUARANTEED_DELIVERY),
   Node2 =>
        (NodeAddress      =>
          (GNAT.Sockets.Family_Inet,
           GNAT.Sockets.Inet_Addr ("127.0.0.1"),
           GNAT.Sockets.Port_Type (12001)),
         Sequence_Count => 0,
         QoS                  => NOT_GUARANTEED_DELIVERY));
```


Il ruolo del *Middleware* – 2

- Una tabella con una riga per ogni nodo
  - Da inizializzare all'avvio del sistema

```
type Connection_Entry is record
  Address          : Sock_Addr_Type; -- indirizzo nodo
  Socket_Send      : Socket_Type;    -- connessione uscente
  Socket_Receive : Socket_Type;      -- connessione entrante
  Send_Set         : Boolean := False; -- Socket_Send inizializzata
  Receive_Set      : Boolean := False; -- Receive_Send inizializzata
end record;
```

  - Per ogni connessione uscente si inizializza il campo Socket_Send (usato da VM_MW_Send)
  - Per ogni connessione entrante si inizializza il campo Socket_Receive (su cui attende Receiver)
  - Invio e ricezione abilitati solo dopo l'inizializzazione

Inizializzazione del trasporto

- Il *Middleware* di nodo crea un *socket* "*in*" per accettare tutte le connessioni in entrata
- Poi crea un processo per ogni connessione *out*
  - Il processo crea un *socket* e lo usa per comunicare con la sua destinazione
    - Riprovando fino a che non ha successo
  - Poi inserisce il *socket* "*out*" nella tabella di *Middleware* e lo marca come inizializzato
- Poi crea un altro processo che si mette in attesa *socket* "*in*"
  - All'arrivo di un messaggio crea un *socket* legato al nodo mittente e aggiorna la tabella