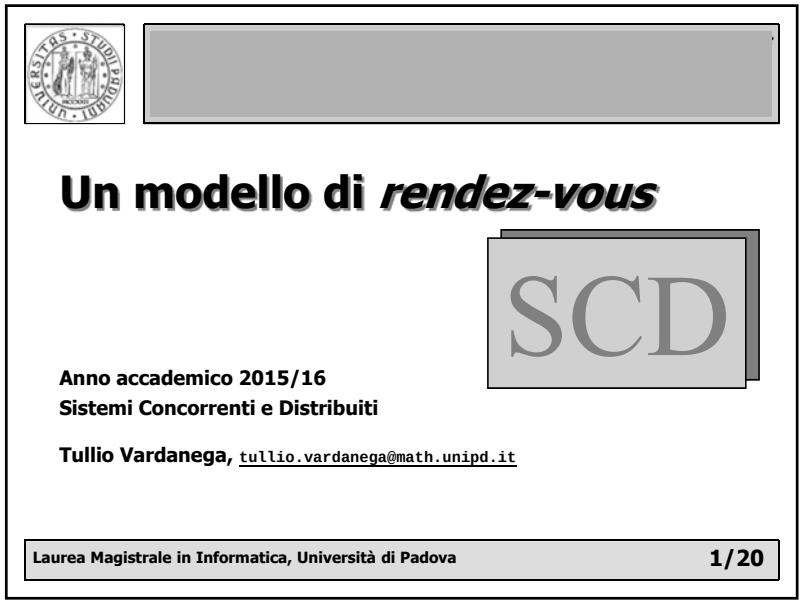
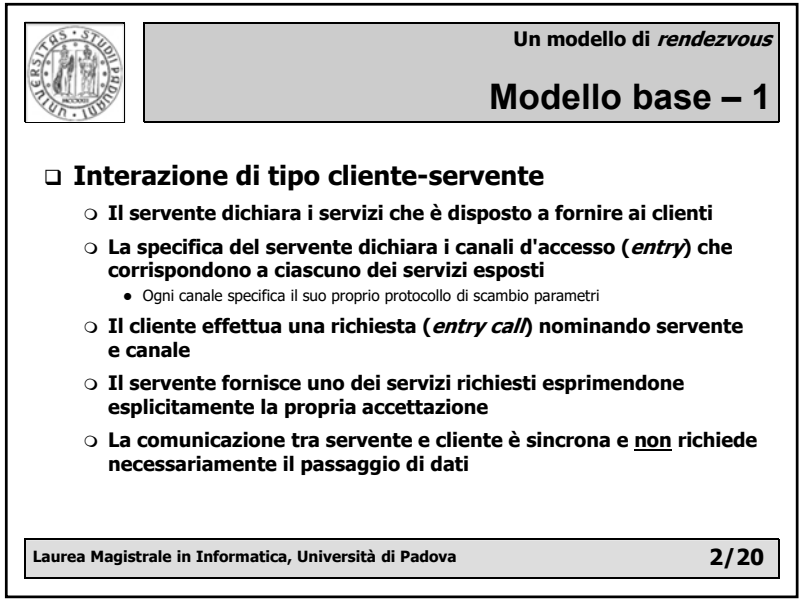


Un modello di *rendez-vous*

SCD

Anno accademico 2015/16
Sistemi Concorrenti e Distribuiti

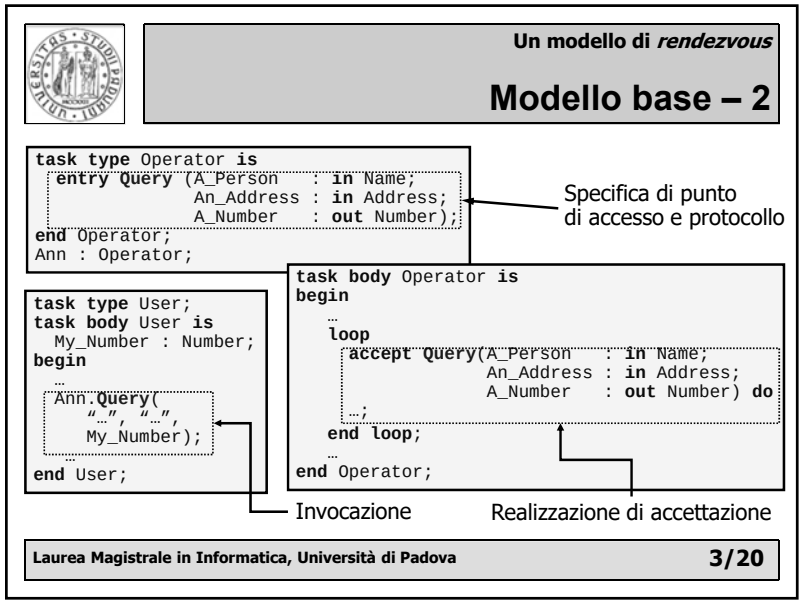
Tullio Vardanega, tullio.vardanega@math.unipd.it



Un modello di *rendezvous*

Modello base – 1

- **Interazione di tipo cliente-servente**
  - **Il servente dichiara i servizi che è disposto a fornire ai clienti**
  - **La specifica del servente dichiara i canali d'accesso (*entry*) che corrispondono a ciascuno dei servizi esposti**
    - Ogni canale specifica il suo proprio protocollo di scambio parametri
  - **Il cliente effettua una richiesta (*entry call*) nominando servente e canale**
  - **Il servente fornisce uno dei servizi richiesti esprimendone esplicitamente la propria accettazione**
  - **La comunicazione tra servente e cliente è sincrona e non richiede necessariamente il passaggio di dati**



Un modello di *rendezvous*

Modello base – 2

```
task type Operator is
   entry Query (A_Person   : in Name;
                An_Address : in Address;
                A_Number   : out Number);
end Operator;
Ann : Operator;
```

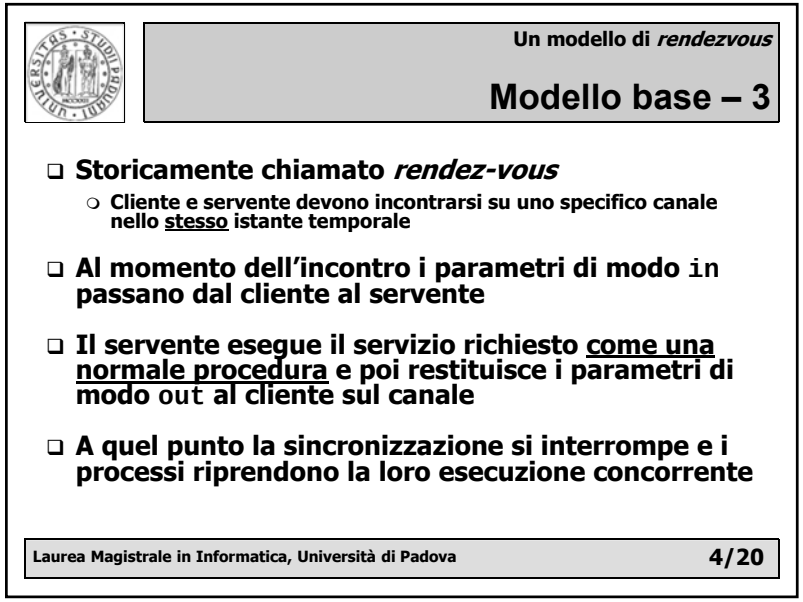
Specifica di punto di accesso e protocollo

```
task type User;
task body User is
  My_Number : Number;
begin
  …
  Ann.Query(
     "…", "…",
     My_Number);
  …
end User;
```

```
task body Operator is
begin
   …
   loop
      accept Query(A_Person   : in Name;
                   An_Address : in Address;
                   A_Number   : out Number) do
      …;
   end loop;
   …
end Operator;
```

Invocazione

Realizzazione di accettazione



Un modello di *rendezvous*

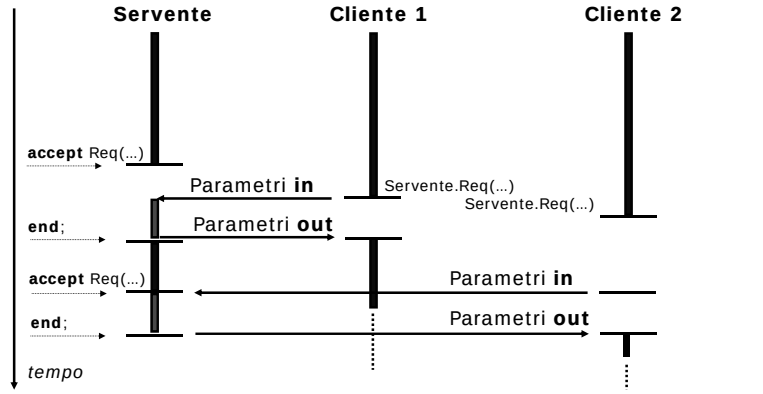
Modello base – 3

- **Storicamente chiamato *rendez-vous***
  - **Cliente e servente devono incontrarsi su uno specifico canale nello stesso istante temporale**
- **Al momento dell'incontro i parametri di modo `in` passano dal cliente al servente**
- **Il servente esegue il servizio richiesto come una normale procedura e poi restituisce i parametri di modo `out` al cliente sul canale**
- **A quel punto la sincronizzazione si interrompe e i processi riprendono la loro esecuzione concorrente**

Modello base – 4

- **Nella forma base**
  - **Il servente si sospende in attesa di una richiesta**
    - Come previsto per l'entità *Server* nel modello di concorrenza di lezione C01
  - **Il cliente si sospende fino alla disponibilità del servizio**
  - **La chiamata del cliente viene posta in una coda associata al canale (*entry queue*)**
  - **L'ordine di accodamento è normalmente FIFO ma può essere configurato diversamente**
    - Per esempio su base prioritaria
    - Ma con le conseguenze che questo può comportare in termini di *starvation*

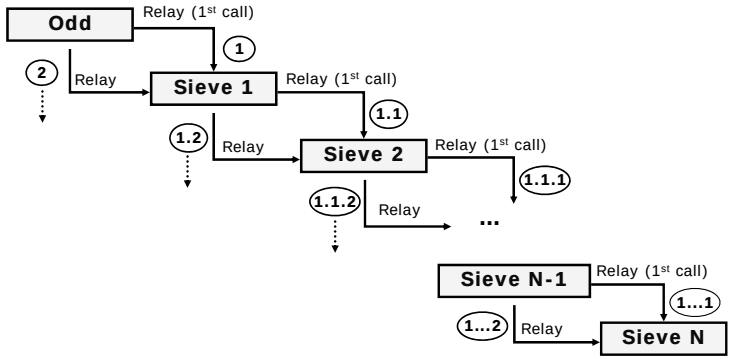
Modello base – 5



Esempio – 1

- **Il crivello di Eratostene rivisitato**
  - **Una diversa realizzazione dell' algoritmo visto nella lezione precedente**
  - **Ogni coppia di processi nella sequenza comunica tramite *rendez-vous***
  - **L'effetto di sincronizzazione rende superflua la mutua esclusione sui dati del servizio**
    - L'accodamento FIFO sulla coda d'accesso preserva l'ordine dei valori da esaminare (proprietà di serializzazione)

Esempio – 2



Esercizio

- **Studiare le modalità di terminazione del programma «Erathostenes-II» associato alla lezione corrente**
- **Spiegare come e perché esse differiscono da quelle della versione precedente dello stesso programma («Erathostenes-I»)**

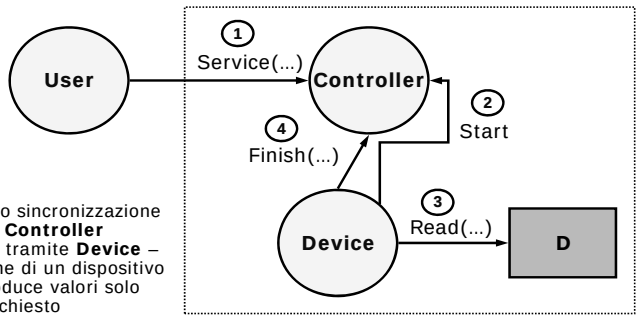
Sincronizzazione tripartita – 1

- **Il modello *rendez-vous* è**
  - **Sincrono rispetto alla comunicazione**
  - **Asimmetrico rispetto all'interfaccia e alla denominazione**
  - **Bidirezionale rispetto al flusso dei dati**
- **Le azioni svolte dal servente durante la sincronizzazione possono coinvolgere processi terzi**
  - **Ciò permette di realizzare forme complesse di sincronizzazione sempre preservando separazione funzionale tra i processi coinvolti**

Sincronizzazione tripartita – 2

- **Il coinvolgimento di processi terzi nella sincronizzazione di lato *server* ammette due forme distinte e duali**
  - **Annidando accettazioni**
    - Modellando una **macchina a stati** in cui alcuni stati sono raggiungibili solo a partire da un dato stato iniziale
  - **Invocando accettazioni nella realizzazione di una accettazione**
    - Realizzando la fornitura di un **servizio composito** che racchiude il possibile contributo di più serventi che si siano ripartiti tra loro il lavoro

Sincronizzazione tripartita – 3



- Utilizzando sincronizzazione tripartita, **Controller** realizza – tramite **Device** – l'astrazione di un dispositivo **D** che produce valori solo quando richiesto
- L'entità **Device** è una macchina a stati che usa l'accettazione delle sue *entry call* come evento di transizione

Sincronizzazione tripartita – 4

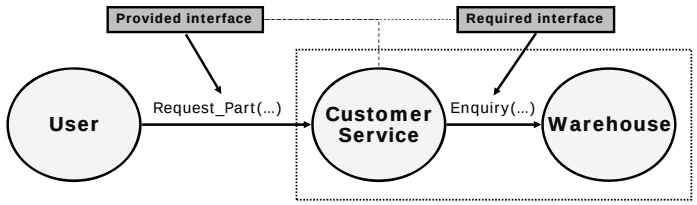
```
task User;
task Device;
task Controller is
 entry Service (I : out Integer);
 entry Start;
 entry Finish (K : out Integer);
end Controller;
```

```
task body User is
…
 Controller.Service (Val);
 …
end User;
```

```
task body Controller is
begin
 loop
  accept Service (I : out Integer) do
   accept Start;
   accept Finish (K : out Integer) do
    I := K; -- azione sincronizzata
   end Finish;
  end Service;
 end loop;
end Controller;
```

```
task body Device is
 Val : Integer;
 procedure Read
   (I : out Integer);
begin
 loop
  Controller.Start;
  Read(Val);
  Controller.Finish(Val);
 end loop;
end Device;
```


Sincronizzazione tripartita – 5



- Strutturazione gerarchica con *encapsulation*
- Il servizio Request_Part(…) nasconde al cliente la necessità di eventuale approvvigionamento presso componenti incapsulati all'interno del servente

Sincronizzazione tripartita – 6

```
task Warehouse is
 entry Enquiry
   (Item : Part_Number;
    In_Stock : out Boolean);
end Warehouse;

task Customer_Service is
 entry Request_Part
   (Order : Part_Number;
    Part  : Spare_Part;
    Order : Order_Number);
end Customer_Service;
```

Difetto: i servizi di Warehouse sono in questo modo visibili a tutti e non solo a Customer_Service

```
task body Customer_Service is
 In_Stock : Boolean;
 …
begin
 loop
  …
  accept Request_Part
     (Order : Part_Number;
      Part  : Spare_Part;
      Order : Order_Number) do
   …
   if In_Stock then
    Part  := The_Part; Order := None;
   else
    Warehouse.Enquiry(Order, In_Stock);
    if In_Stock then
     … -- go get part from Warehouse
     Part  := The_Part; Order := Next_Order_Nr;
    end if;
   end if;
  end Request_Part;
 end loop;
end Customer_Service;
```


Punti d'accesso privati – 1

- **Un servente non deve necessariamente esporre al pubblico tutti i suoi canali**
- **Alcuni possono essere ristretti per motivi di incapsulazione e/o di astrazione**
- **La dichiarazione dei canali deve in tal caso distinguere tra pubblici e privati**

Punti d'accesso privati – 2

```
task User;
task Controller is
 entry Service (I : out Integer);
private
 entry Start;
 entry Finish (K : out Integer);
end Controller;
```

In questo modo la visibilità ai canali privati è ristretta al solo ambito (*scope*) del processo **Controller**

```
task body User is
…
 Controller.Service(Val);
 …
end User;
```

```
task body Controller is
 task Device;
 task body Device is
  Val : Integer;
  procedure Read (I : out Integer) is … ;
 begin
  loop
   Controller.Start;
   Read(Val);
   Controller.Finish(Val);
  end loop;
 end Device;
-- continues in sidebox
```

```
begin -- Controller
 loop
  accept Service (I : out Integer) do
   accept Start;
   accept Finish (K : out Integer) do
    I := K;
   end Completed;
  end Service;
 end loop;
end Controller;
```


Esercizio

- **Studiare le soluzioni al problema dei filosofi a cena proposte nei programmi associati alla lezione 3 (C02)**
- **Verificare quali di tali soluzioni usino il *rendez-vous* per assicurare mutua esclusione nelle rispettive sezioni critiche**

Casi d'errore

- **Una eccezione sollevata durante la sincronizzazione ne causa l'abbandono e si propaga a entrambi i partecipanti**
- **Emettere una richiesta d'accesso verso un processo terminato è un errore a tempo di esecuzione e solleva una eccezione nel chiamante**

Stati d'esecuzione di processo