# Communicating Sequential Processes

Gabriele Sartor
gabriele.sartor@edu.unito.it

Università degli Studi di Torino

11 dicembre 2015

# Cos'è il CSP (Communicating Sequential Processes)

- Linguaggio formale per la descrizione di modelli di interazione di sistemi concorrenti
- Calcolo processi basati su scambio di messaggi
- Descritto da Tony Hoare
- Usato nell'industria come strumento di specifica e verifica di aspetti concorrenti nei sistemi

# Definizioni del CSP

| | |
|---|---|
| $0$ | Nil Process |
| $a.P$ | Prefixing $(a \in Act)$ |
| $P_1 + P_2$ | External Choice |
| $P_1 \oplus P_2$ | Internal Choice |
| $P_1 \mid P_2$ | Parallel Composition |

# Alcune regole del CSP (1)

Prefixing $\dfrac{}{a.P \xrightarrow{a} P}$

# Alcune regole del CSP (2)

External Choice

$$\frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$$

# Alcune regole del CSP (3)

Internal Choice

$$P \oplus Q \xrightarrow{\tau} P$$

$$P \oplus Q \xrightarrow{\tau} Q$$

# Alcune regole del CSP (4)

Parallel Composition

$$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \qquad \frac{Q \xrightarrow{a} Q'}{P \mid Q \xrightarrow{a} P \mid Q'}$$

$$\frac{P \xrightarrow{a} P', Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

# Note sull'implementazione

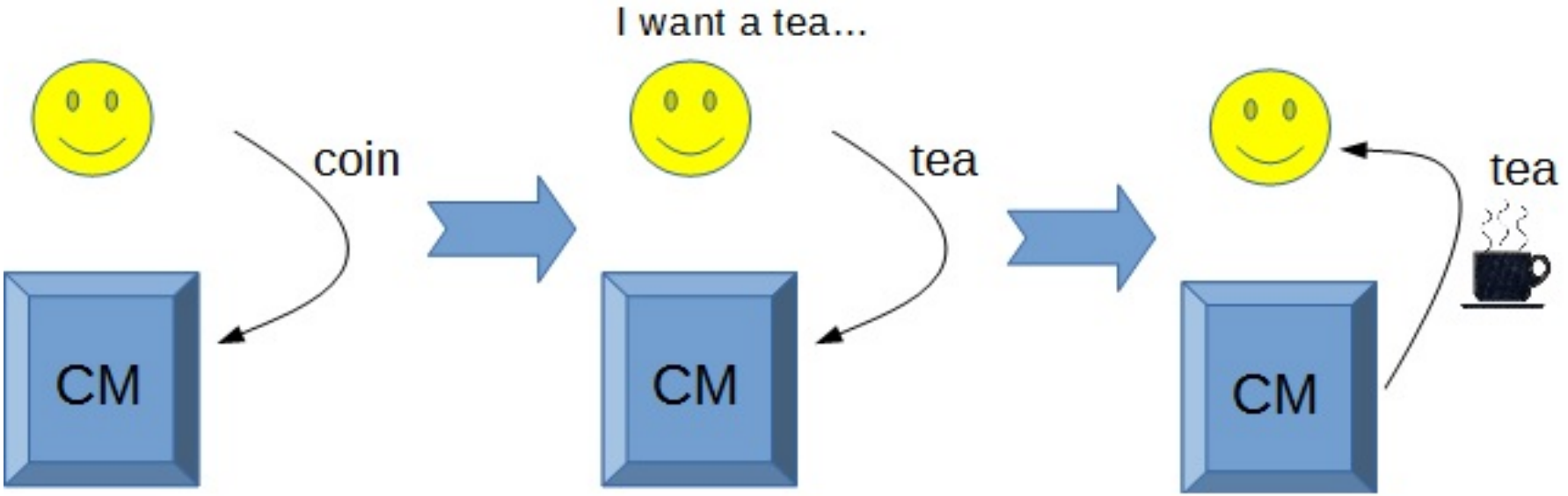
- Le azioni sono semplici stringhe
  - $azione$ = "azione"
  - $\overline{azione}$ = "- azione"
- Vengono eseguite solo comunicazioni interne
- I processi dati in input all'interprete, sono dei processi in parallelo
  - [ P1,P2,.. ] = P1 | P2 | ..

# Implementazione : tipi

```
type Program = [Process]
data Process = Pfix Action Process
             | EC Process Process
             | IC Process Process
             | NP
           deriving Show
type Action = String
```

# Alla macchinetta del caffè

Abbiamo 2 entità che hanno vita indipendente, ma che possono comunicare!



# Esempio di riduzione

Programma?

Consumer | CoffeeMachine

$$\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0) \oplus 0 \mid \text{coin}.(\overline{coffee}.0 + \overline{tea}.0)$$

# Esempio di riduzione (1)

Scelta interna del primo processo : 0

$$\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0) \oplus 0 \mid \text{coin}.(\overline{coffee}.0 + \overline{tea}.0)$$

$$0 \mid \text{coin}.(\overline{coffee}.0 + \overline{tea}.0)$$

# Esempio di riduzione (2)

Scelta interna del primo processo : $\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0)$

$$\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0) \oplus 0 \mid \text{coin}.(\overline{coffee}.0 + \overline{tea}.0)$$

$$\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0) \mid \text{coin}.(\overline{coffee}.0 + \overline{tea}.0)$$

$$\text{coffee}.0 \oplus \text{tea}.0 \mid \overline{coffee}.0 + \overline{tea}.0$$

$$\text{coffee}.0 \mid \overline{coffee}.0 + \overline{tea}.0$$

$$0 \mid 0$$

# Esempio di riduzione (3)

Scelta interna del primo processo : $\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0)$

$$\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0) \oplus 0 \mid \text{coin}.(\overline{coffee}.0 + \overline{tea}.0)$$

$$\overline{coin}.(\text{coffee}.0 \oplus \text{tea}.0) \mid \text{coin}.(\overline{coffee}.0 + \overline{tea}.0)$$

$$\text{coffee}.0 \oplus \text{tea}.0 \mid \overline{coffee}.0 + \overline{tea}.0$$

$$\text{tea}.0 \mid \overline{coffee}.0 + \overline{tea}.0$$

$$0 \mid 0$$

# Esempio di riduzione : implementazione

Processi in linguaggio CSP

Parsificazione dei due processi

**C** : IC (Pfix "-coin" (IC (Pfix "coffee" NP) (Pfix "tea" NP))) NP
**CM** : Pfix "coin" (EC (Pfix "-coffee" NP) (Pfix "-tea" NP))

**Riduzione del programma** :
[ C, CM ]

# Esempio di riduzione : esecuzione (1)

```
(-coin.(coffee.0&tea.0)&0)|coin.(-coffee.0+-tea.0)

Riduco pos :0.

0|coin.(-coffee.0+-tea.0)
Interpretazione finita.
```

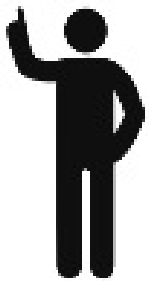
# Esempio di riduzione : esecuzione (2)

```
(-coin.(coffee.0&tea.0)&0)|coin.(-coffee.0+-tea.0)

Riduco pos :0.

-coin.(coffee.0&tea.0)|coin.(-coffee.0+-tea.0)

Riduco pos :0 e 1.
(coffee.0&tea.0)|(-coffee.0+-tea.0)

Riduco pos :0.

coffee.0|(-coffee.0+-tea.0)

Riduco pos :0 e 1.
0|0
Interpretazione finita.
```

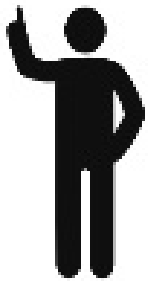
# Esempio di riduzione : esecuzione (3)

```
(-coin.(coffee.0&tea.0)&0)|coin.(-coffee.0+-tea.0)

Riduco pos :0.

-coin.(coffee.0&tea.0)|coin.(-coffee.0+-tea.0)

Riduco pos :0 e 1.
(coffee.0&tea.0)|(-coffee.0+-tea.0)

Riduco pos :0.

tea.0|(-coffee.0+-tea.0)

Riduco pos :0 e 1.
0|0
Interpretazione finita.
```

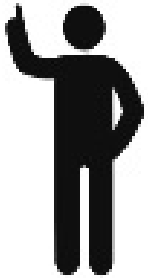
# Un altro esempio : acquisto libro



Cliente

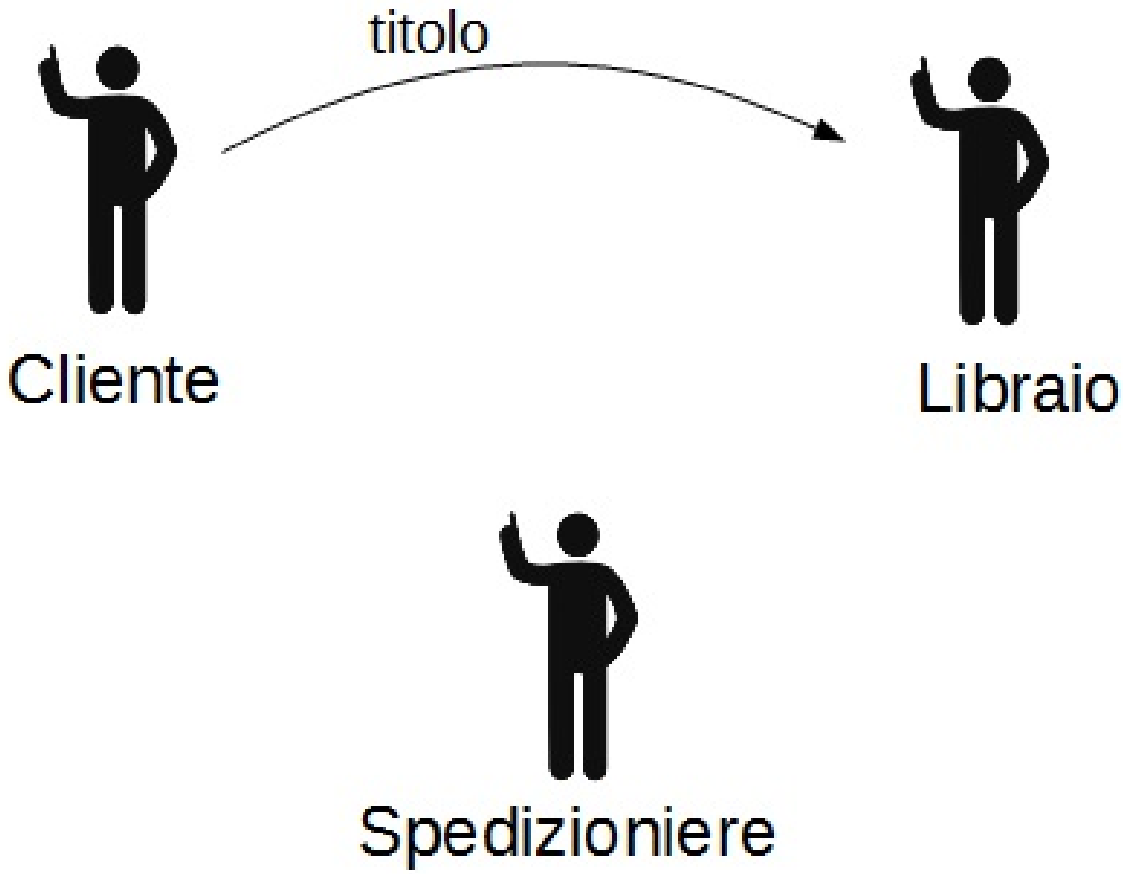


Libraio

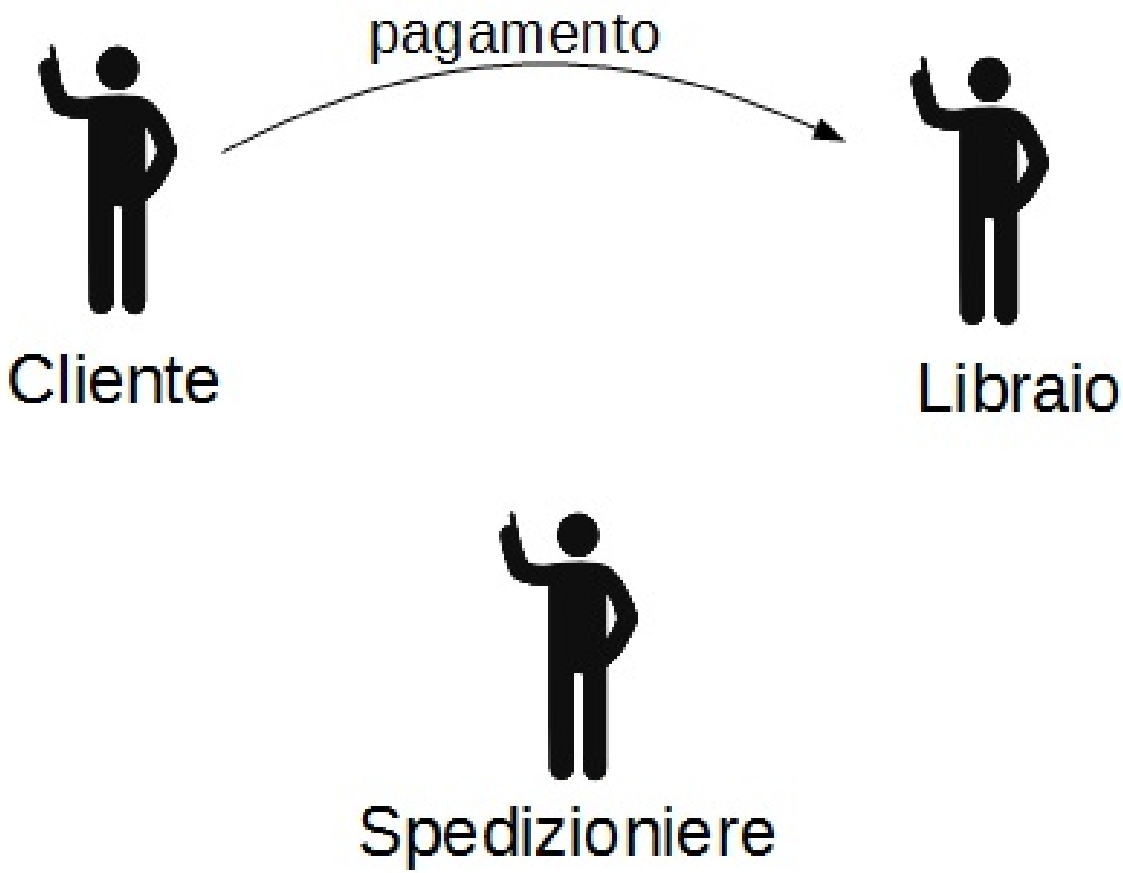


Spedizioniere

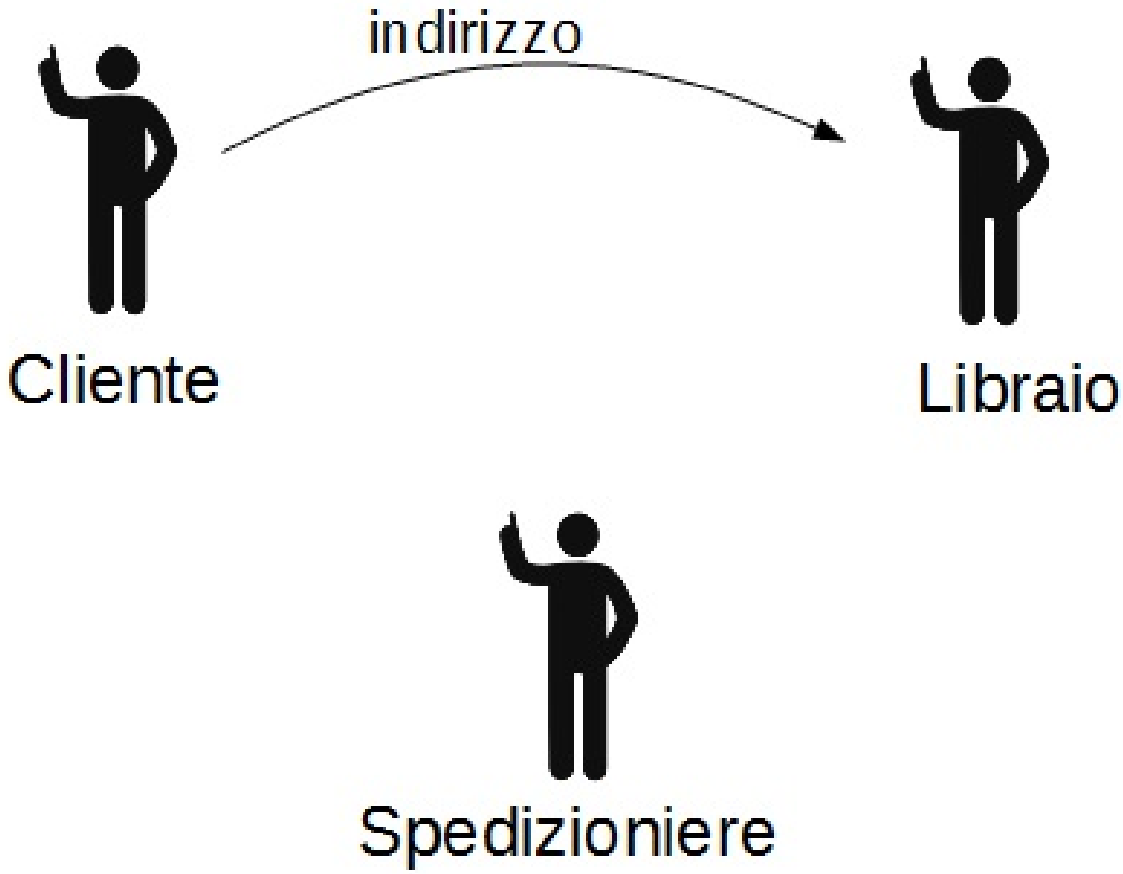
# Un altro esempio : acquisto libro (1)



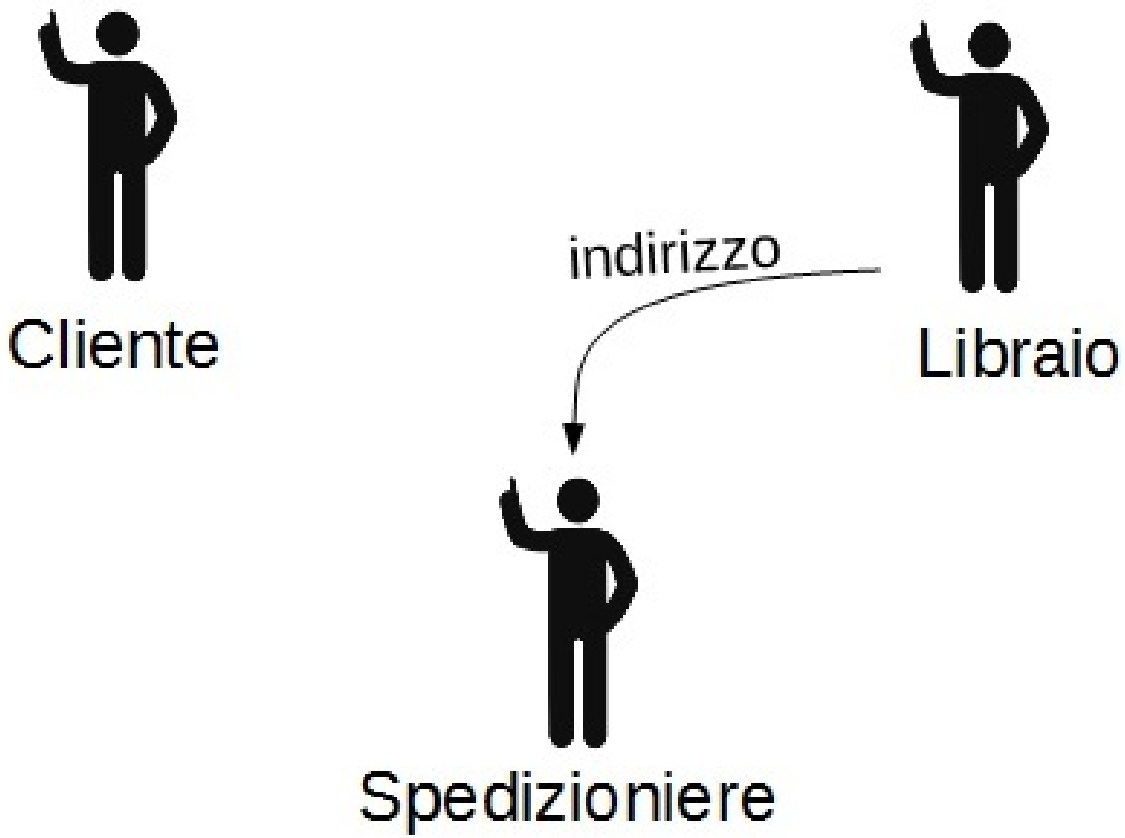
# Un altro esempio : acquisto libro (2)



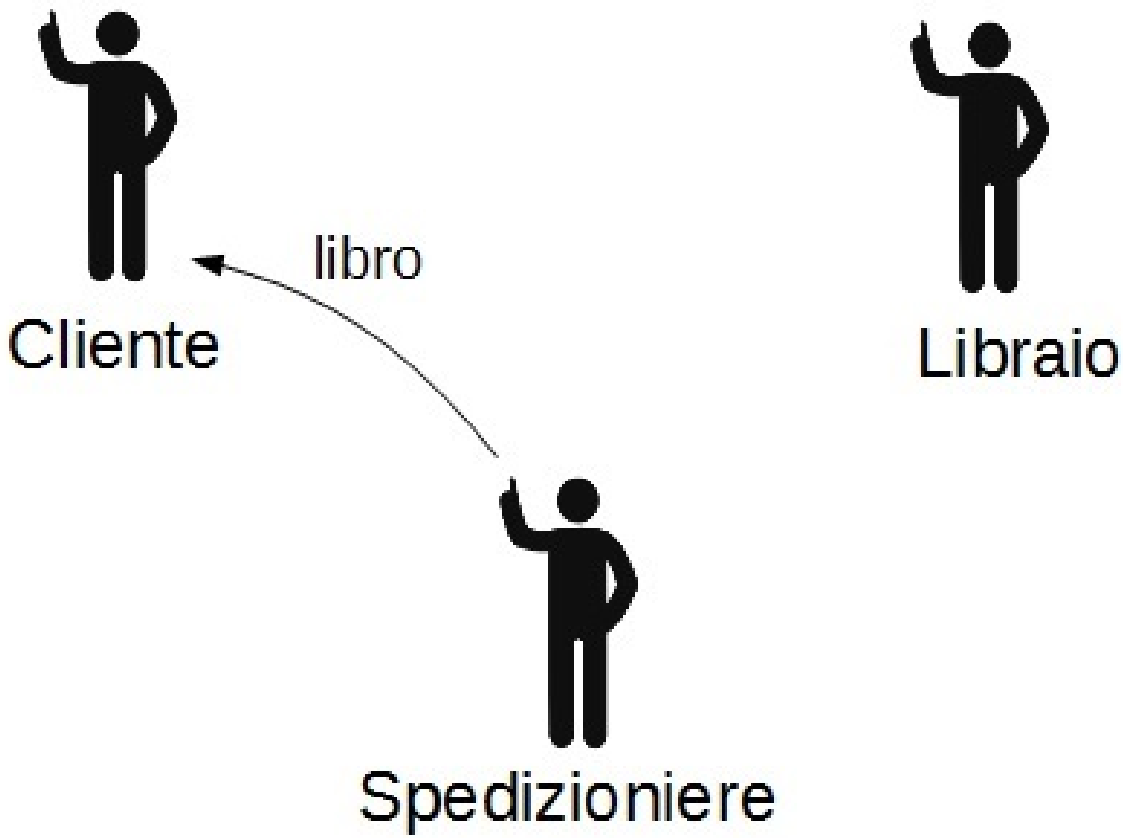
# Un altro esempio : acquisto libro (3)



# Un altro esempio : acquisto libro (4)



# Un altro esempio : acquisto libro (5)



# Esempio di riduzione : esecuzione (1)

```
(-titolo1.-paga.-ind.libro.0&-titolo2.-paga.-ind.libro.0)|(titolo1.paga.ind.-ind2.0+titolo2.paga.ind.-ind2.0)|ind2.-libro.0

Riduco pos :0.

-titolo1.-paga.-ind.libro.0|(titolo1.paga.ind.-ind2.0+titolo2.paga.ind.-ind2.0)|ind2.-libro.0

Riduco pos :0 e 1.
-paga.-ind.libro.0|paga.ind.-ind2.0|ind2.-libro.0

Riduco pos :0 e 1.
-ind.libro.0|ind.-ind2.0|ind2.-libro.0

Riduco pos :0 e 1.
libro.0|-ind2.0|ind2.-libro.0

Riduco pos :1 e 2.
libro.0|0|-libro.0

Riduco pos :0 e 2.
0|0|0
Interpretazione finita.
```

# Esempio di riduzione : esecuzione (2)

```
(-titolo1.-paga.-ind.libro.0&-titolo2.-paga.-ind.libro.0)|(titolo1.paga.ind.-ind2.0+titolo2.paga.ind.-ind2.0)|ind2.-libro.0

Riduco pos :0.

-titolo2.-paga.-ind.libro.0|(titolo1.paga.ind.-ind2.0+titolo2.paga.ind.-ind2.0)|ind2.-libro.0

Riduco pos :0 e 1.
-paga.-ind.libro.0|paga.ind.-ind2.0|ind2.-libro.0

Riduco pos :0 e 1.
-ind.libro.0|ind.-ind2.0|ind2.-libro.0

Riduco pos :0 e 1.
libro.0|-ind2.0|ind2.-libro.0

Riduco pos :1 e 2.
libro.0|0|-libro.0

Riduco pos :0 e 2.
0|0|0
Interpretazione finita.
```

# Perché Haskell?


Aspetti positivi :

- L'utilizzo del pattern matching è particolarmente adatto ad implementare un interprete
- Haskell fornisce di un'ampia libreria di funzioni
- Più semplice il debug

Aspetti negativi :

- Complicato implementare funzioni random