Sincronizzazione nel modello ad ambiente locale

Corso di Laurea in Ingegneria Informatica

Università degli Studi di Napoli Federico II

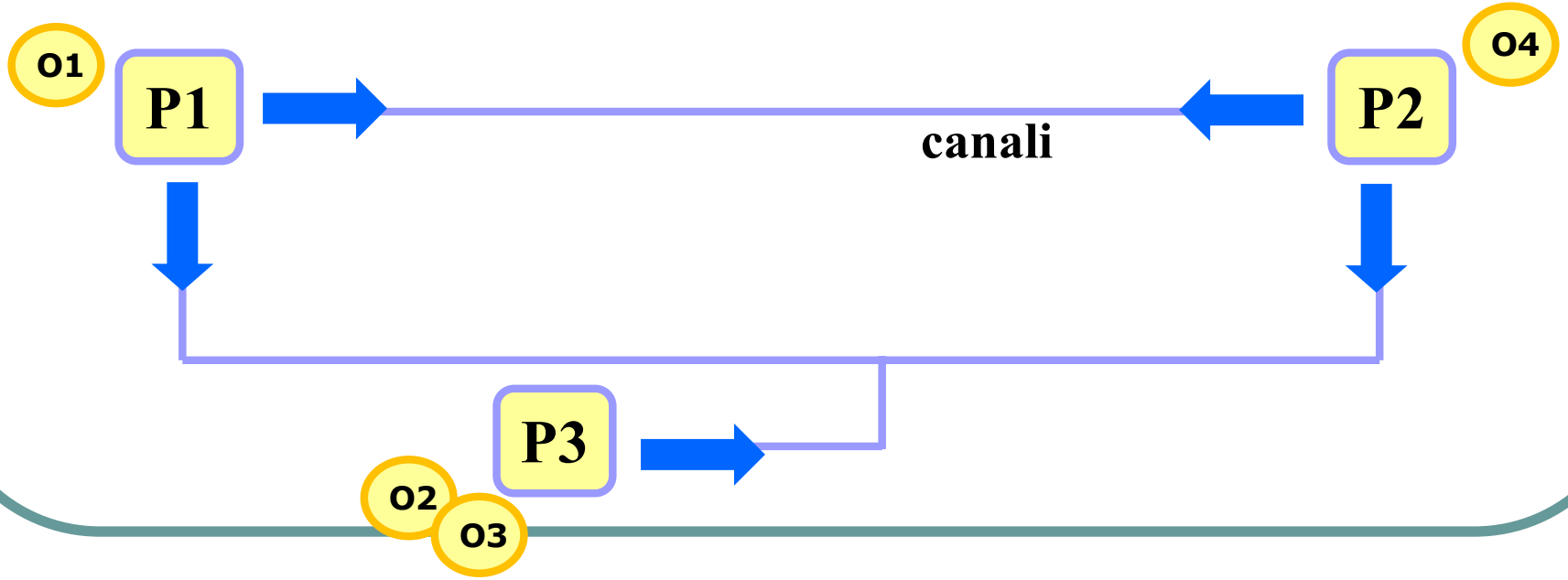
Anno Accademico 2024/2025, Canale San Giovanni

Sincronizzazione nel modello ad ambiente locale

- Sommario
  - Modello ad ambiente locale
  - Primitive di scambio messaggi
  - Sincronizzazione e indirizzamento
- Riferimenti
  - P. Ancilotti, M.Boari, A. Ciampolini, G. Lipari, "Sistemi Operativi", Mc-Graw-Hill (Cap.3; par. 3.5, 3.6)

Il modello ad ambiente locale

- Ogni processo evolve in un proprio ambiente
  - Non esiste memoria condivisa
  - Le risorse sono tutte private
  - Non possono essere modificate direttamente da altri
- Il naturale supporto fisico a questo modello sono i sistemi con architettura distribuita



Il modello ad ambiente locale

- La cooperazione si realizza mediante lo scambio diretto di messaggi per mezzo di primitive fornite dal S.O.
- Garantisce la mutua esclusione, poiché tutte le risorse sono private

Primitive di scambio messaggi

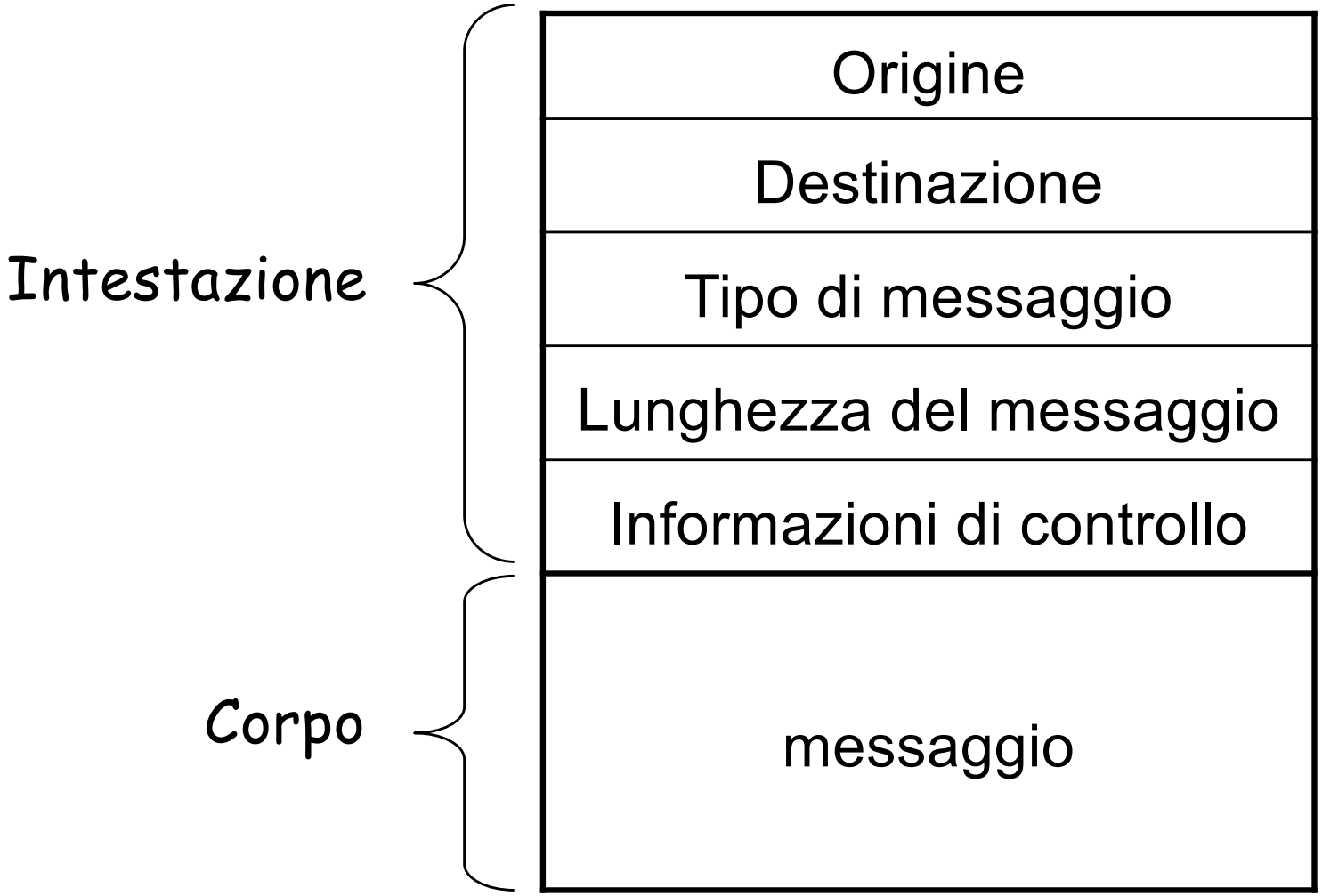
- Due tipologie di primitive

```
send (destination, message)
receive (source, message)
```

Variano fra i sistemi in base a:

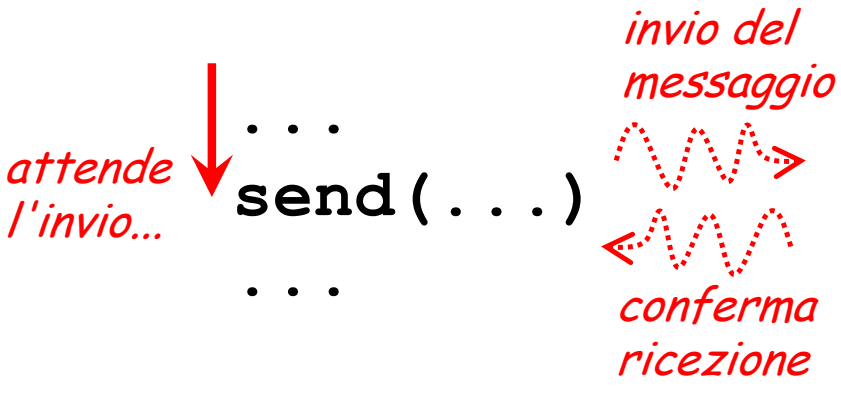
- tipo di sincronizzazione dei processi comunicanti
- indirizzamento: la modalità con cui si designano la provenienza e la destinazione

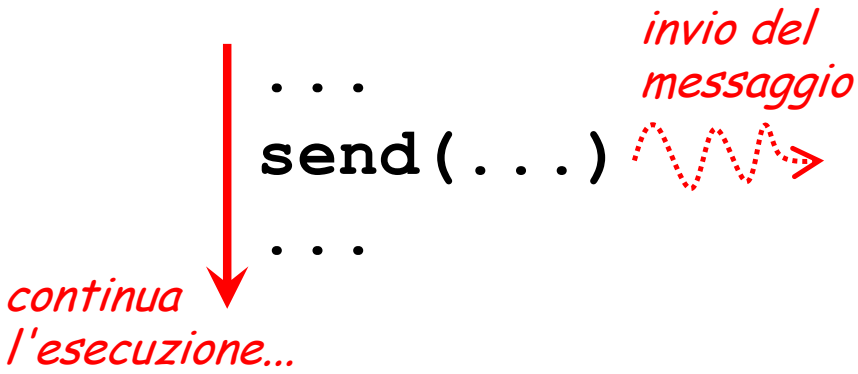
Formato di un Messaggio



Sincronizzazione: Send

All'invio di un messaggio (send), due possibilità





Send sincrona: Il processo che esegue la send rimane in attesa, fino a che il messaggio è stato ricevuto

Send asincrona: Il processo che esegue la send continua la sua esecuzione, senza attendere l'avvenuta consegna del messaggio

Sincronizzazione: Receive

- Analogamente per la primitiva receive

Receive Bloccante: Se il messaggio non è stato ancora inviato, rimane in attesa fino alla ricezione

Receive Non Bloccante: Continua l'esecuzione senza attendere l'avvenuta consegna del messaggio

Se il sender ha già inviato il messaggio, il processo lo riceve e continua l'esecuzione

Sincronizzazione

- Il sender e il receiver possono porre in attesa il processo, a seconda del tipo di primitive utilizzate
- Sono tre le combinazioni generalmente utilizzate

Sincronizzazione

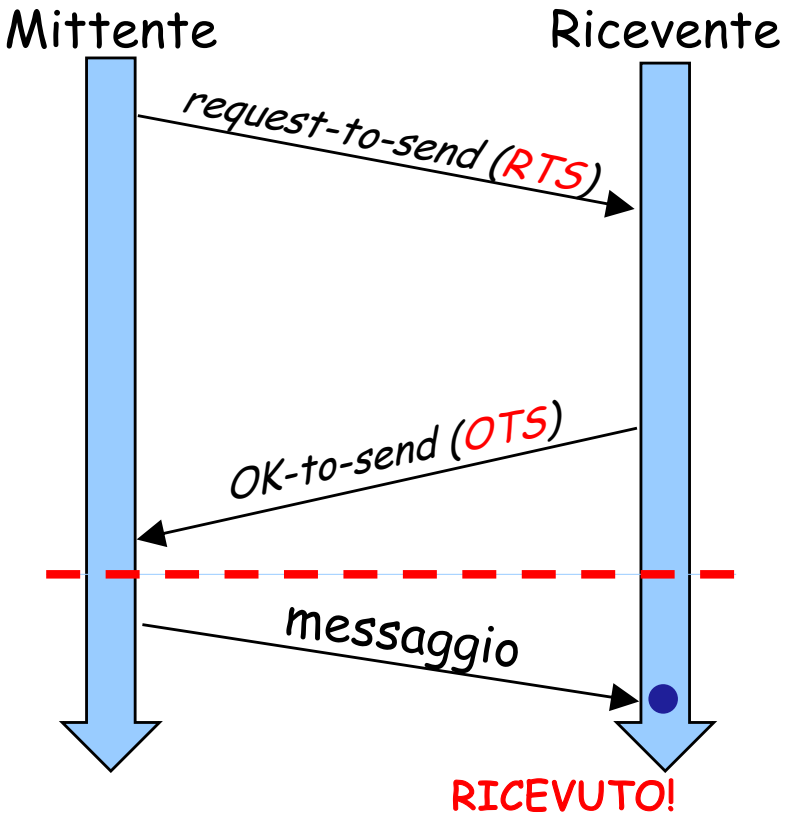
- Send sincrona, receive bloccante
  - Stretta sincronizzazione tra sender e receiver
  - Entrambi in attesa della consegna del messaggio
  - Denominato anche "**rendezvous**"
- Send asincrona, receive bloccante
  - Il sender continua a eseguire dopo l'invio
  - Il receiver rimane in attesa fino all'arrivo
- Send asincrona, receive non bloccante
  - Nessuna delle due parti rimane in attesa

Sincronizzazione

- La send asincrona e la receive bloccante sono quelle che si trovano più spesso nei sistemi operativi e linguaggi di programmazione

Implementazione di send sincrona a mezzo di send asincrona

- È possibile realizzare la **send sincrona** usando **send asincrona + receive bloccante**
- Il programma deve determinare se il messaggio è stato ricevuto, inviando dei messaggi aggiuntivi



Implementazione di send sincrona a mezzo di send asincrona (sender)

```
procedure sendSincrona(dest, mess)
{
    sendAsincrona (dest, messRTS)
       /* messRTS è un messaggio di "pronto
          ad inviare" (Request To Send) */
    receiveBloccante (dest, messOTS)
       /* messOTS è un messaggio di "pronto
          a ricevere" (OK To Send) */
    sendAsincrona (dest, mess)
}
```

Implementazione di send sincrona a mezzo di send asincrona (receiver)

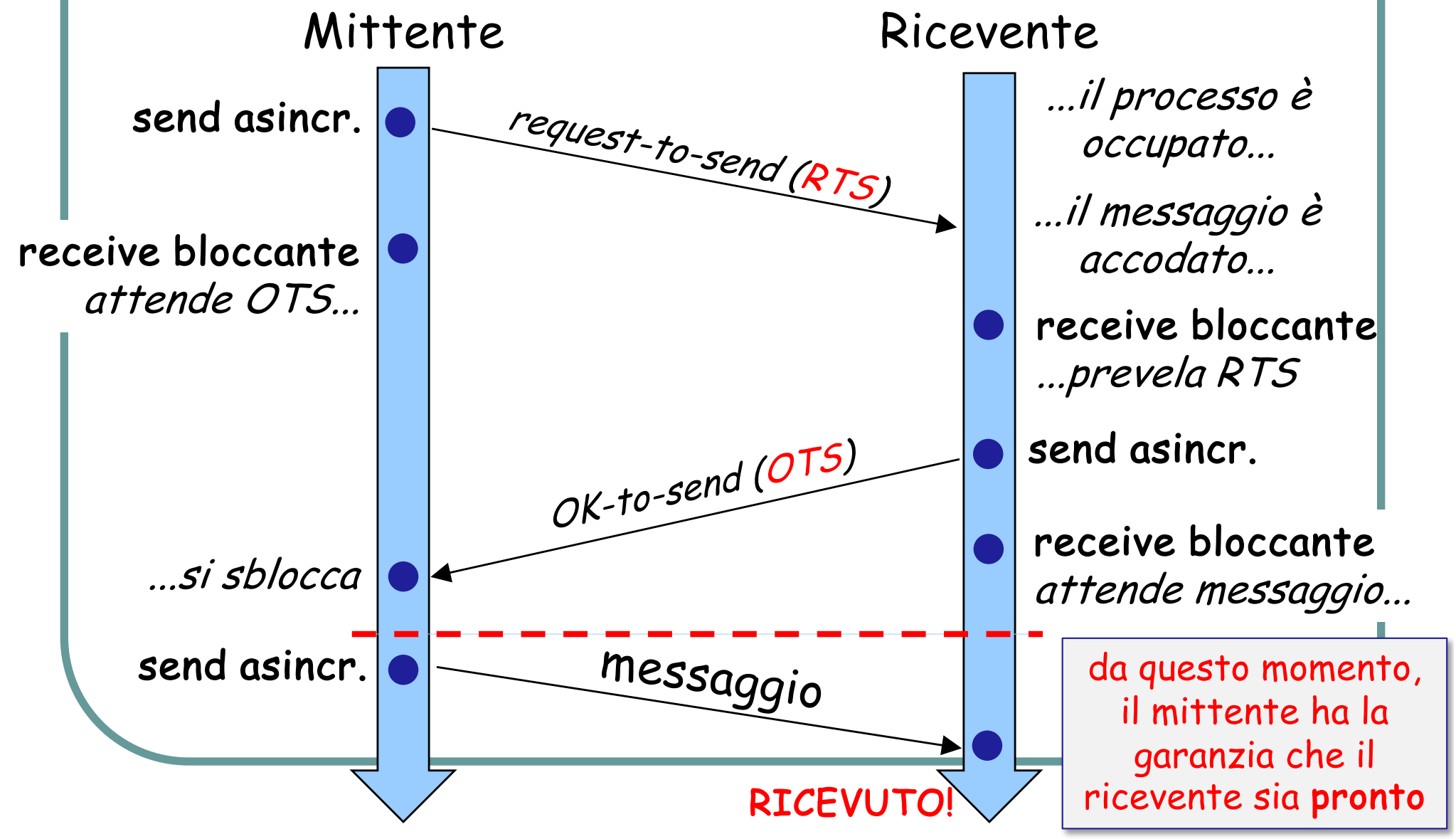
```
procedure receive(source, mess)
{
    receiveBloccante (source, messRTS)

    sendAsincrona (source, messOTS)

    receiveBloccante (source, mess)
}
```

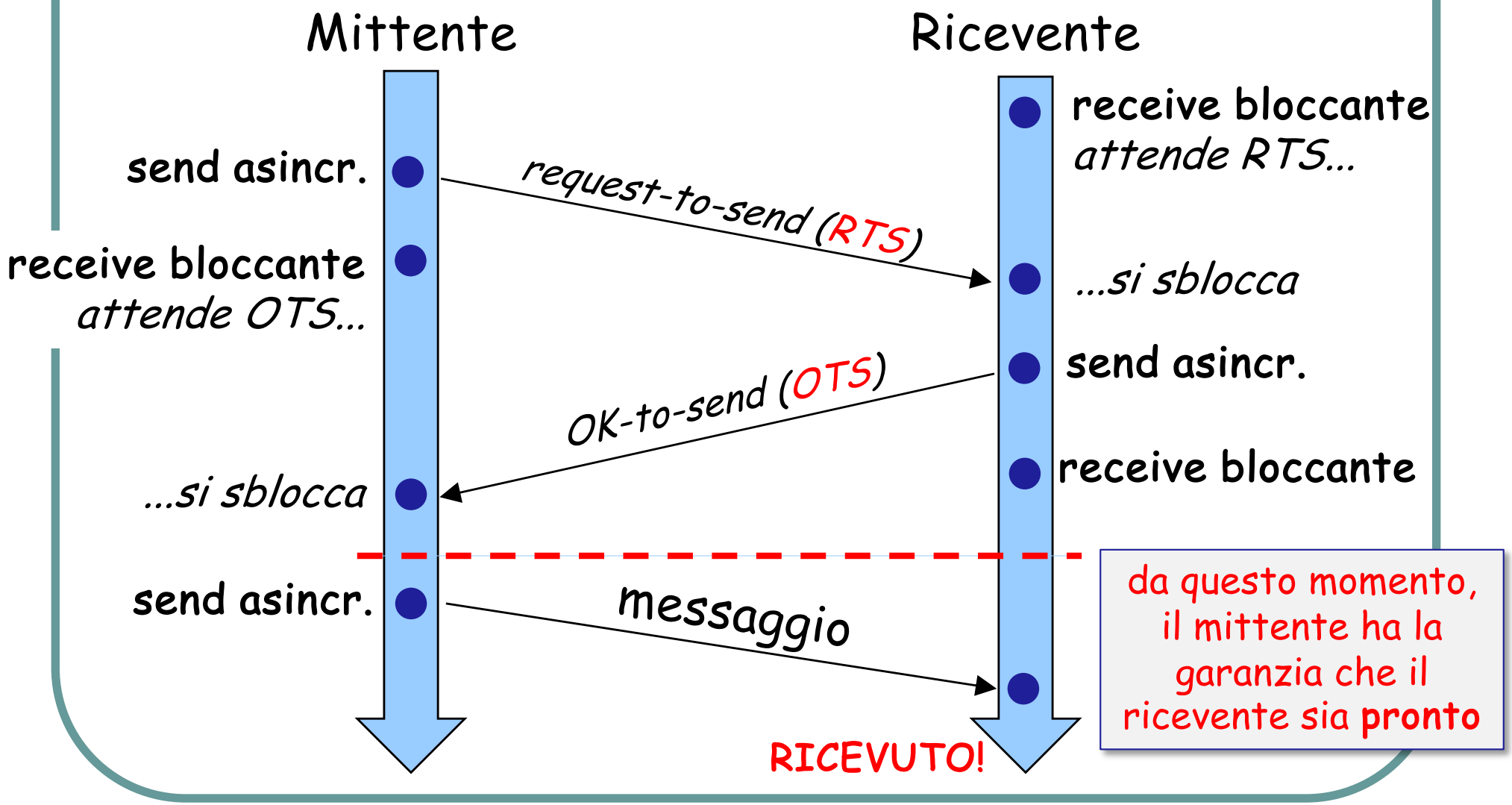
Rendezvous

Caso 1: Il ricevente non è ancora pronto a ricevere



Rendezvous

Caso 2: Il ricevente è già pronto a ricevere

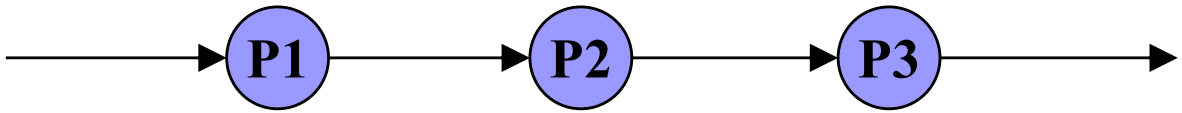


Indirizzamento

- La destinazione e la provenienza possono essere gestite in più modi
  - **comunicazione diretta simmetrica**
    - Il mittente esplicita il PID del destinatario nella send()
    - Il destinatario esplicita il PID del mittente nella receive()
  - **comunicazione diretta asimmetrica**
    - Il mittente esplicita il PID del destinatario nella send()
    - Il destinatario non indica un PID. Viene a conoscenza del PID del mittente alla ricezione del messaggio, tramite parametro di uscita
  - **comunicazione indiretta**
    - Il mittente fa riferimento ad una mailbox nella send()
    - Il destinatario fa riferimento alla stessa mailbox, da cui preleva il messaggio tramite receive()

Comunicazione Diretta

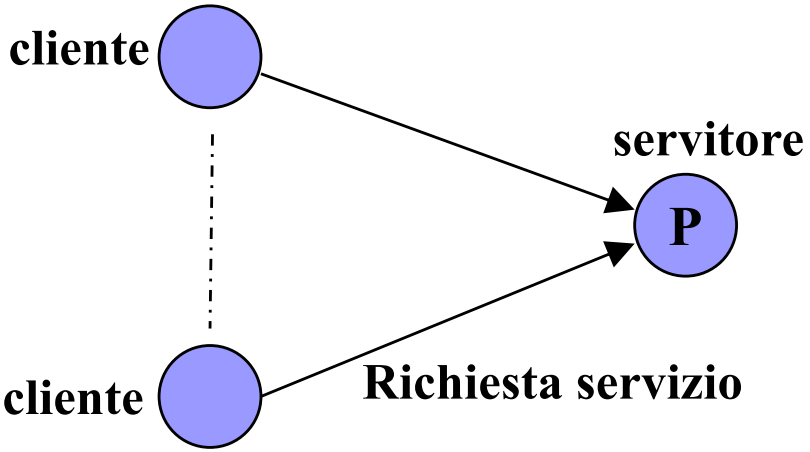
- Diretta e Simmetrica: **schema a pipeline**



Send (msg, P2)

Receive (msg, P1)

- Diretta e Asimmetrica: **cliente-servitore**

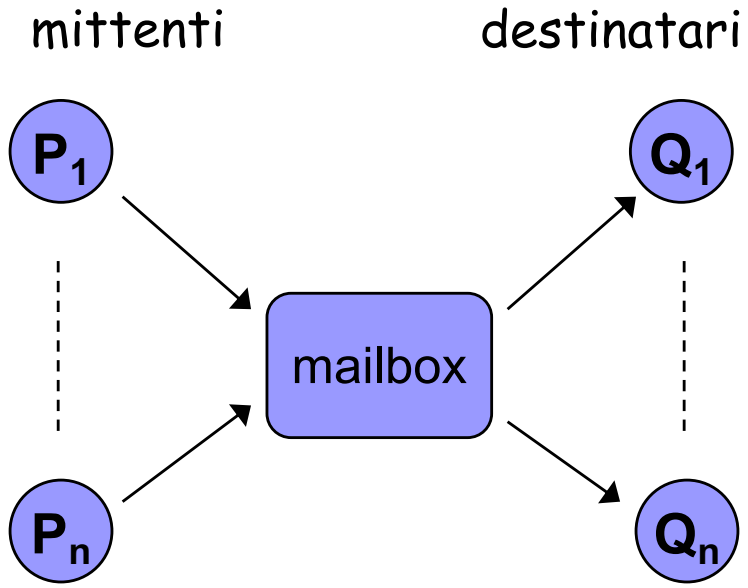


Send (msg, P)

Receive (msg, idCliente)

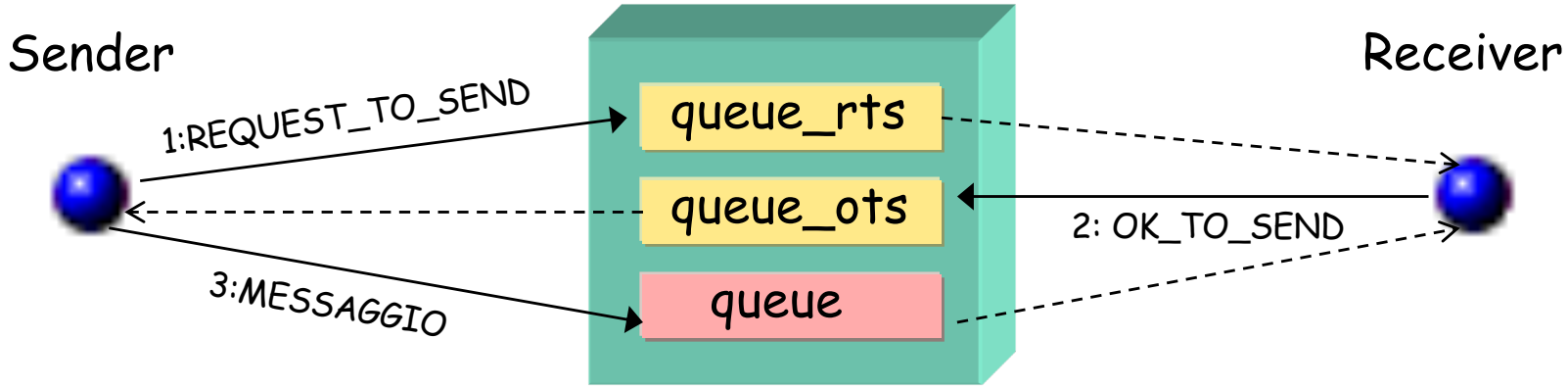
Comunicazione Indiretta

- I messaggi vengono inviati ad una struttura dati condivisa (detta "coda" o "mailbox")
- Vantaggio: sender e receiver sono maggiormente indipendenti, per la presenza della mailbox
- Possibili schemi di comunicazione:
  - **one-to-one**
  - **one-to-many**
  - **many-to-one**
  - **many-to-many**



Send (msg, mailbox)
Receive (msg, mailbox)

Esempio: Rendezvous con mailbox

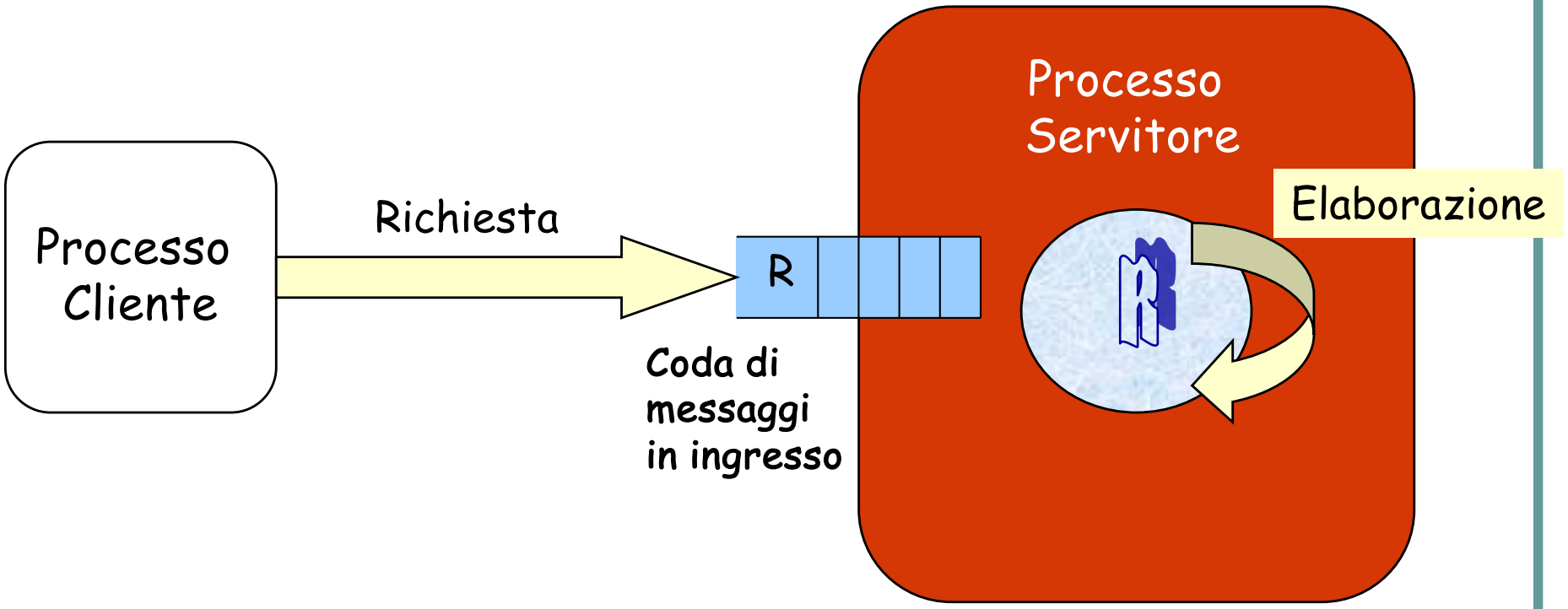


Processo Servitore

- Nel caso di comunicazione diretta asimmetrica, o indiretta di tipo *many-to-one* si parla di comunicazione **client-server** e **processo servitore**

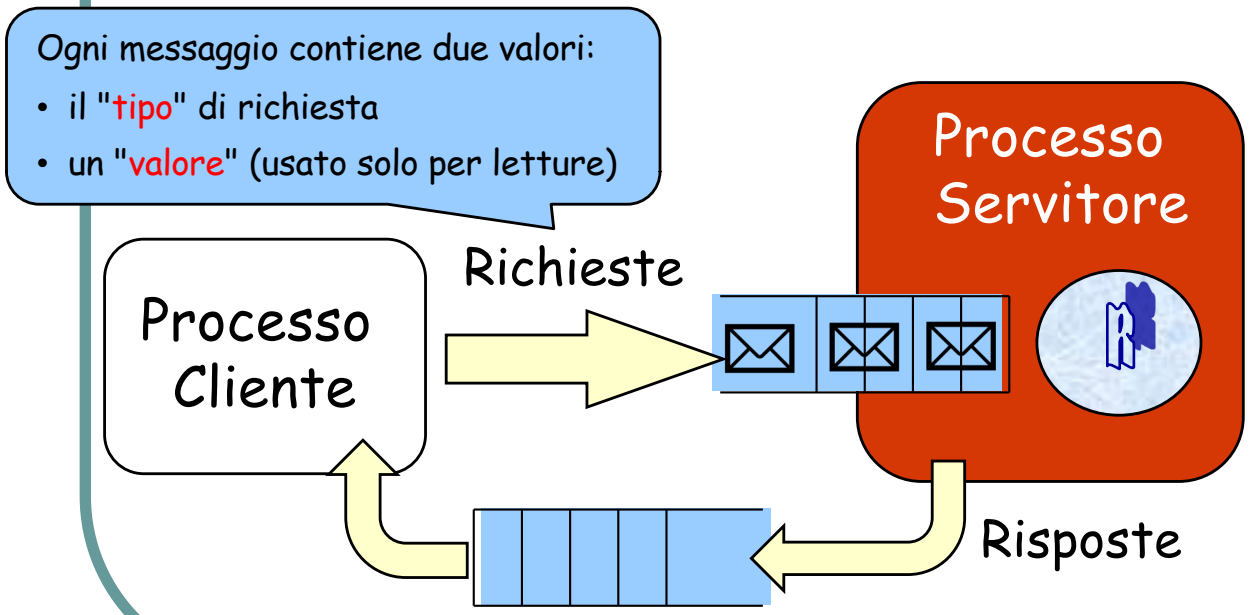
- **Processo servitore**: incapsula la risorsa comune, offrendo a processi esterni le funzionalità di accesso alla risorsa
  - riceve messaggi di richiesta
  - opera sulla risorsa
  - fornisce eventuali risposte

Modello di interazione (con mailbox)



Esempio: Processo Servitore

- Esempio di processo servitore (risorsa "caldaia")
- 3 tipi di richieste
  - Accensione della caldaia
  - Spegnimento della caldaia
  - Lettura della temperatura corrente




```
if(msg.tipo == ACCENDI) {
  R.stato = ACCESO
}
else if(msg.tipo == SPEGNI) {
  R.stato == SPENTO
}
else if(msg.tipo == LEGGI) {
  risposta = R.temperatura
  send(risposta)
}
```