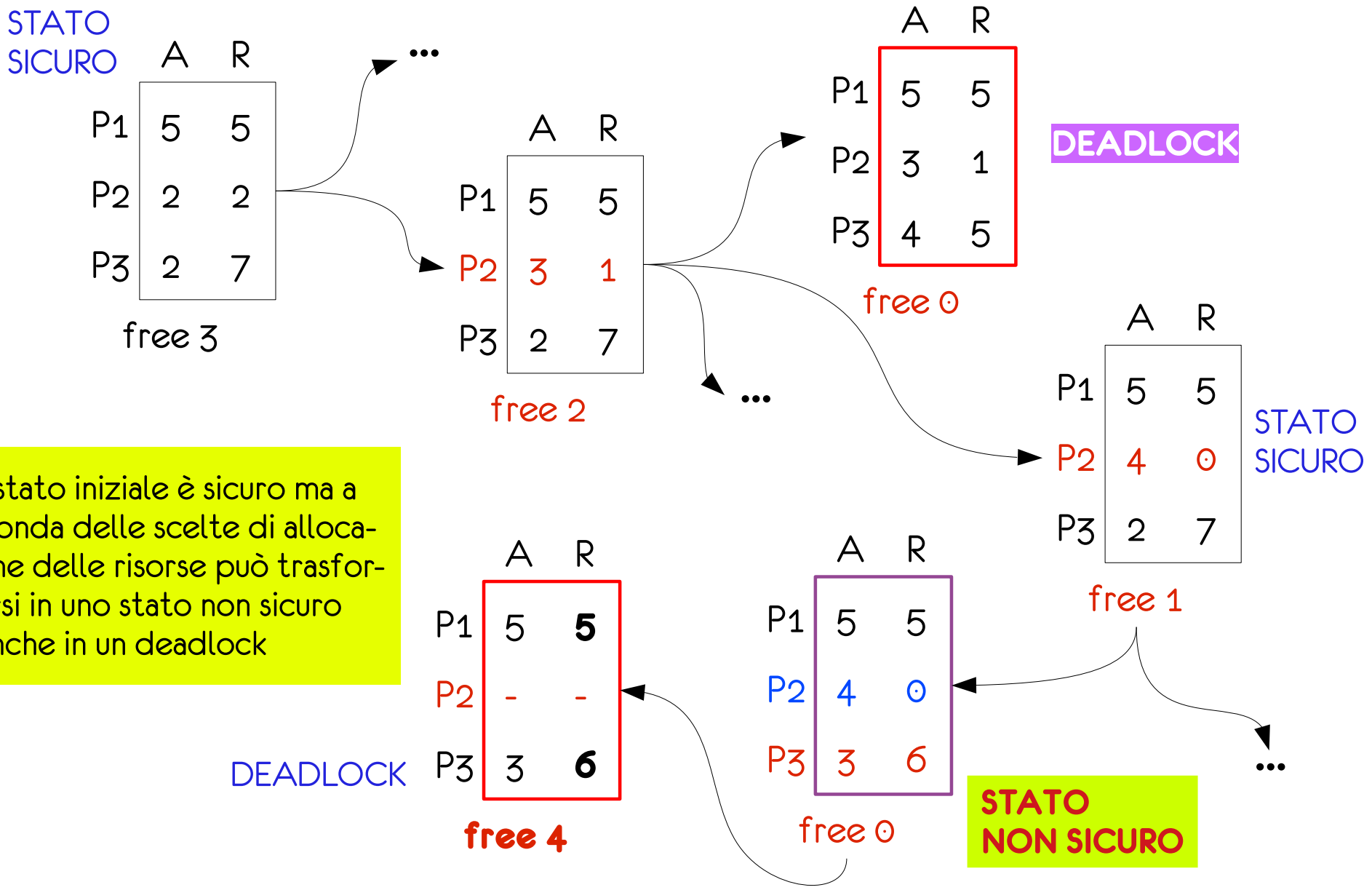
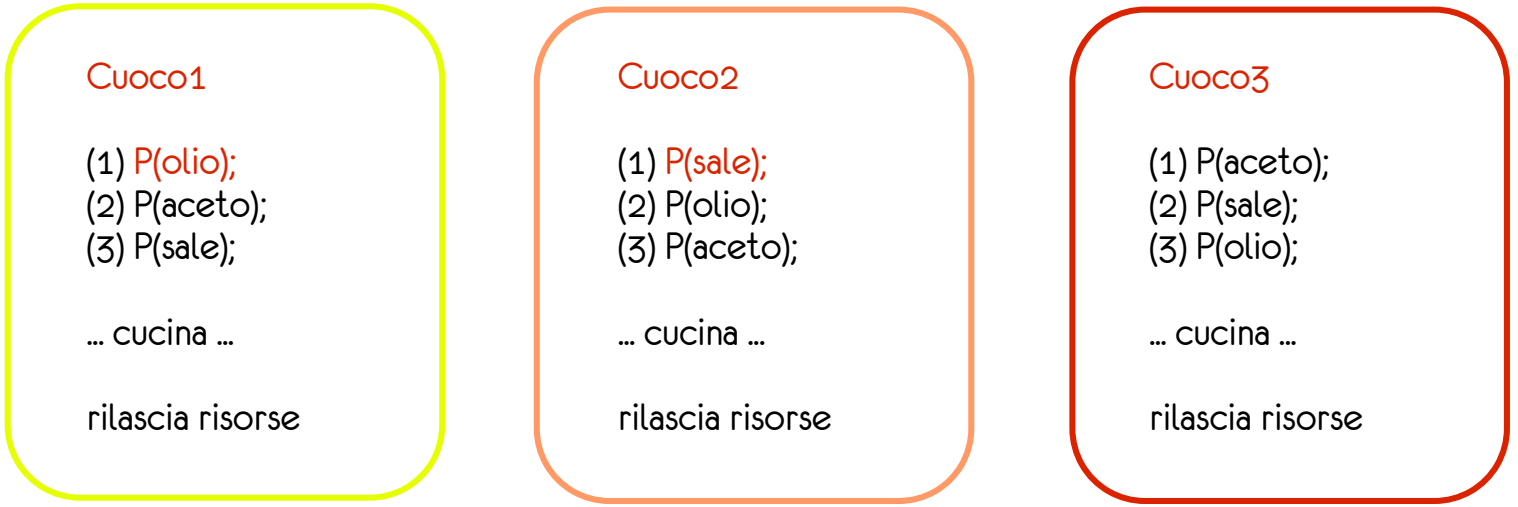
Evoluzione degli stati



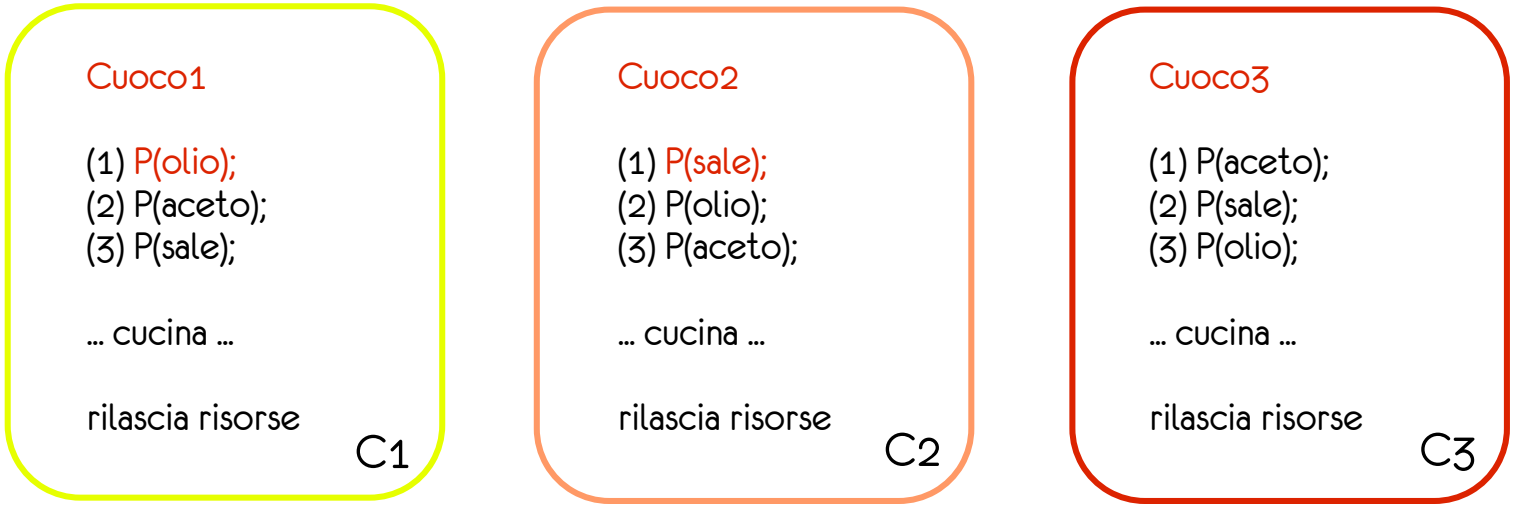
Lo stato iniziale è sicuro ma a seconda delle scelte di allocazione delle risorse può trasformarsi in uno stato non sicuro e anche in un deadlock

Altro esempio: i 3 cuochi



- Supponiamo che Cuoco1 abbia l'olio e Cuoco2 il sale, il sistema è in uno stato sicuro?
- In questo momento non c'è deadlock ma ...
- ... esiste un ordinamento che è una sequenza sicura?

i 3 cuochi



- Tutti i possibili ordinamenti:
  - C1, C2, C3: no C1 dipende da C2 che lo segue
  - C1, C3, C2: idem
  - C2, C1, C3: no C2 dipende da C1 che lo segue
  - C2, C3, C1: idem
  - C3, C1, C2: no C1 dipende da C2
  - C3, C2, C1: no C2 dipende da C1

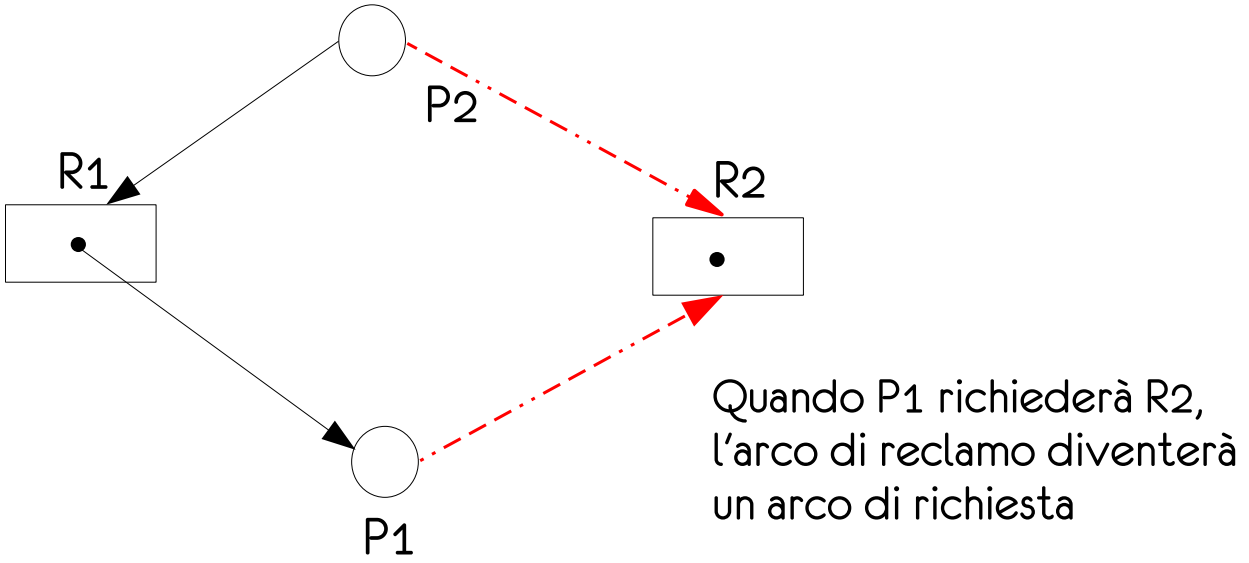
Lo stato non è sicuro

Deadlock avoidance

- Per evitare il deadlock il SO cerca di mantenere l'esecuzione in uno stato sicuro
- **E se una scelta sbagliata portasse a uno stato non sicuro?**
- in questo caso non è più possibile riportare il sistema in uno stato sicuro e molto facilmente si genererà un deadlock

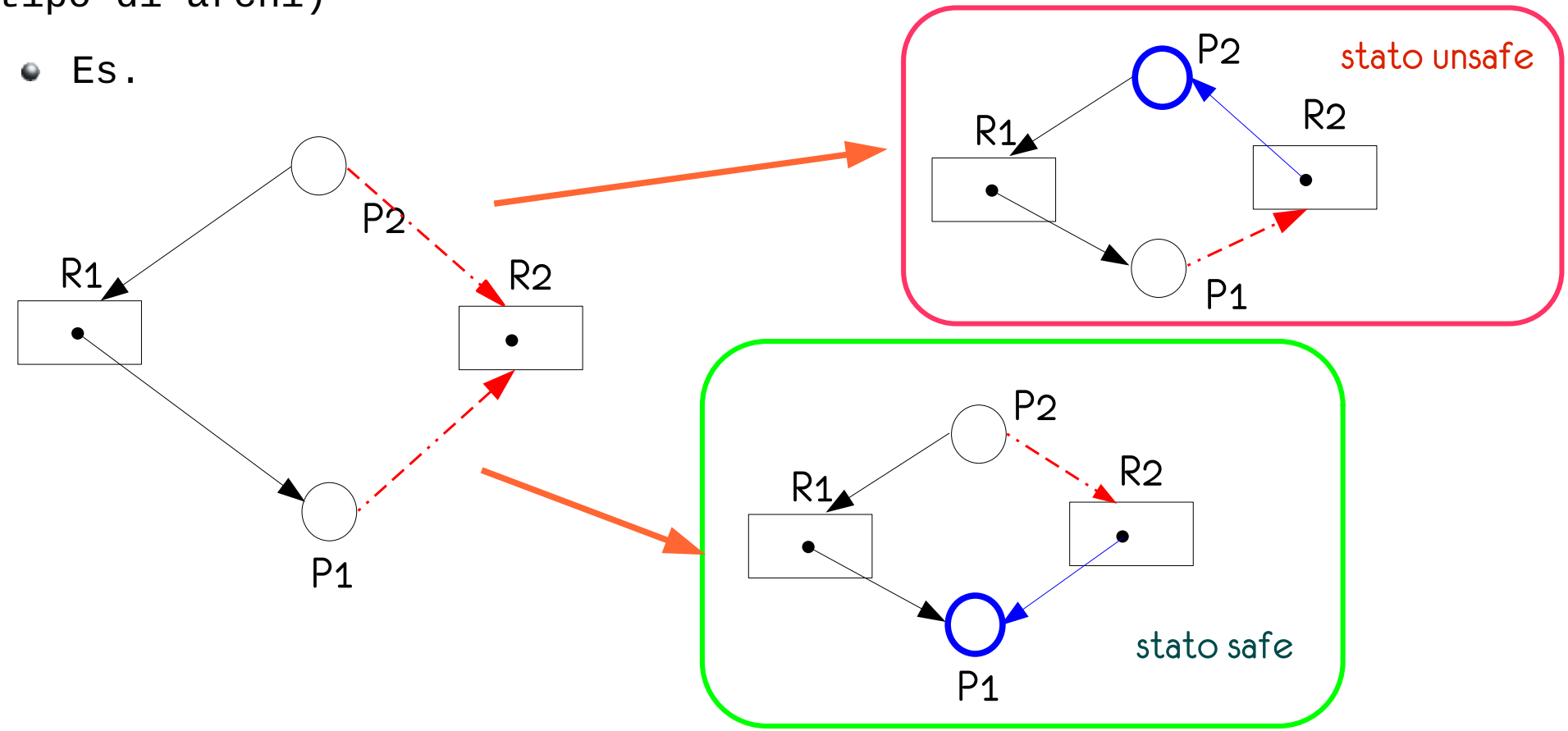
Algoritmo di deadlock avoidance

- Questo metodo funziona solo se ogni classe di risorsa ha **una istanza**
- È possibile prevenire il deadlock utilizzando una variante del grafo di assegnazione delle risorse ottenuto introducendo un terzo tipo di arco:
  - arco di reclamo (claim edge): Pi → Rj indica che Pi richiederà Rj *in futuro*; è rappresentato da una linea tratteggiata



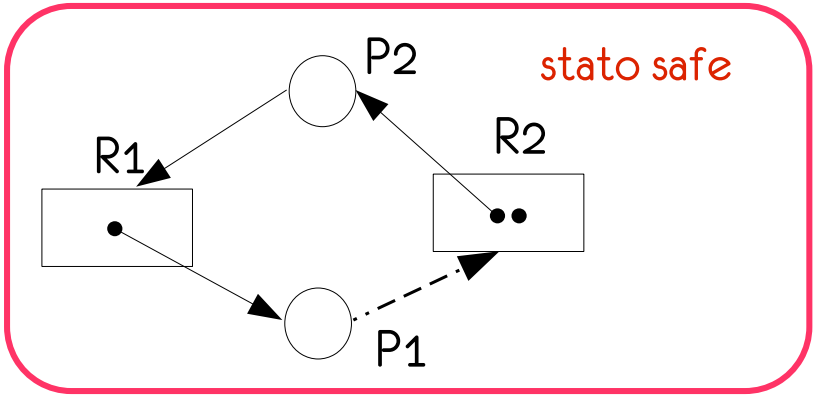
Algoritmo di deadlock avoidance

- **All'inizio tutti i processi inseriscono nel grafo di assegnazione un claim edge per ciascuna risorsa di cui avranno bisogno**
- È possibile trasformare un arco di reclamo in un arco di richiesta **SSE** non si genera un ciclo (costituito da qualsiasi tipo di archi)
  - Es.

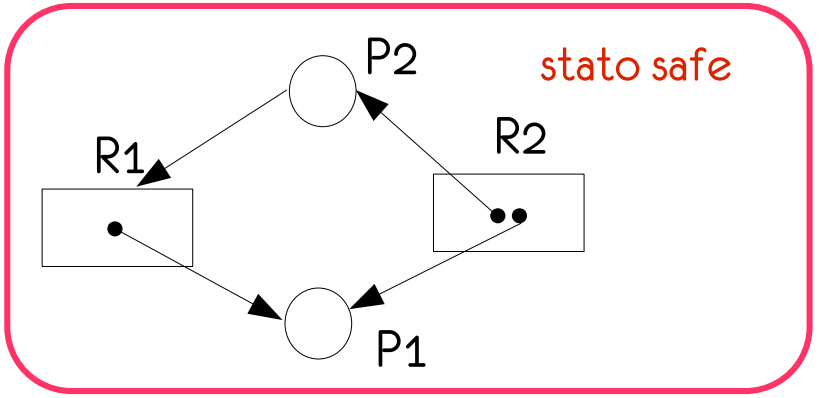


Algoritmo di deadlock avoidance

NB: se io avessi due istanze di
R2 lo stato sarebbe safe!!
La condizione non è più sufficiente



NOTA: quando un processo rilascia
una risorsa l'arco di assegnazione ritorna
ad essere un arco di reclamo



Avoidance: algoritmo del banchiere

- Algoritmo più generale, si applica anche quando i processi richiedono n>1 risorse di un certo tipo
- Metafora: i processi sono visti come clienti di una banca a cui possono richiedere un prestito fino a un certo massimo
- Informazione richiesta: ogni nuovo processo deve dichiarare all'inizio il numero massimo di risorse (dei vari tipi) di cui avrà bisogno
- M = numero delle classi di risorsa gestite
- N = numero dei processi
- Complessità: $O(N^2M)$

Algoritmo del banchiere: variabili

- **disponibili[M]**: indica la disponibilità per ogni classe di risorsa
- **massimo[N][M]**: per ciascun processo indica il numero massimo di risorse di ciascun tipo che saranno richieste
- **assegnate[N][M]**: indica quante risorse di ciascuna classe sono assegnate a ogni processo
- **necessarie[N][M]**: indica quante risorse di ciascun tipo ancora mancano ai vari processi (necessarie = massimo - assegnate)

Algoritmo del banchiere

- L'algoritmo soddisfa una richiesta di un processo **SSE** l'assegnazione delle risorse richieste porta ad uno stato sicuro
- È diviso in due algoritmi:

  1) un algoritmo per verificare che uno stato è sicuro

  2) un algoritmo di gestione di una richiesta (che utilizza il precedente)

- **Convenzione notazionale**:

  Dati due vettori di uguale lunghezza X e Y si indica con $X < Y$ il fatto che per ogni indice i $X[i] < Y[i]$, si indica con $X \leq Y$ il fatto che per ogni indice i $X[i] \leq Y[i]$ e si indica con $Z = X + Y$ il fatto che per ogni indice i $Z[i] = X[i] + Y[i]$

Algoritmo di verifica della sicurezza

1. Siano **Lavoro** e **Fine** due array di lunghezza **M** ed **N**
2. **Lavoro = Disponibili**
3. **Fine[i] = falso**, per ogni i∈[1,N]
4. Cerca i∈[1,N] | **Fine[i]==false** ∧ **Necessarie[i]≤Lavoro**
5. se l'hai trovato:
   1. **Lavoro = Lavoro + Assegnate[i]**
   2. **Fine[i] = true**
   3. **goto 4**
6. altrimenti **goto 7**
7. se **∀i∈[1,N]**, **Fine[i]==true** lo **stato è sicuro**

Esempio con M == 1

STATO SICURO?

	A	R
P1	5	5
P2	2	2
P3	2	7

free 3

```
disponibili = 3
necessarie = {5, 2, 7}
assegnate  = {5, 2, 2}

fine = {false, false, false}
lavoro = disponibili, cioè è uguale a 3
```

```
c'è i∈[1,N] |
    fine[i]==false ∧
    necessarie[i]≤lavoro?
Sì  i == 2!!
```

```
lavoro = lavoro+assegnate[2]
fine[2] = true
```

è come se fossi passata virtualmente nello stato

	A	R
P1	5	5
P2	-	-
P3	2	7

free 5

```
disponibili = 3
necessarie = {5, 2, 7}
assegnate  = {5, 2, 2}

fine = {false, true, false}
lavoro = 5
```

Esempio con M == 1

	A	R
P1	5	5
P2	-	-
P3	2	7

free 5

```
disponibili = 3
necessarie = {5, 2, 7}
assegnate  = {5, 2, 2}

fine = {false, true, false}
lavoro = 5
```

```
c'è i∈[1,N] |
    fine[i]==false ∧
    necessarie[i]≤lavoro?
Sì  i == 1!!
```

```
lavoro = lavoro+assegnate[1]
fine[1] = true
```

è come se fossi passata virtualmente nello stato

	A	R
P1	-	-
P2	-	-
P3	2	7

free 10

```
disponibili = 3
necessarie = {5, 2, 7}
assegnate  = {5, 2, 2}

fine = {true, true, false}
lavoro = 10
```

Esempio con M == 1

	A	R
P1	-	-
P2	-	-
P3	2	7

free 10

```
disponibili = 3
necessarie = {5, 2, 7}
assegnate  = {5, 2, 2}

fine = {true, true, false}
lavoro = 10
```

LO STATO VALUTATO È SICURO

```
c'è i∈[1,N] |
    fine[i]==false ∧
    necessarie[i]≤lavoro?
Sì  i == 3!!
```

```
lavoro = lavoro+assegnate[3]
fine[3] = true
```

è come se fossi passata virtualmente nello stato

	A	R
P1	-	-
P2	-	-
P3	-	-

free 12

```
disponibili = 3
necessarie = {5, 2, 7}
assegnate  = {5, 2, 2}

fine = {true, true, true}
lavoro = 12
```

Algoritmo di gestione delle richieste

1. Consideriamo un processo **j**, sia **Richieste** un vettore di M elementi, **Richieste[i]** è il num. di risorse di classe **i** richieste da **j** in un certo istante
2. Se **Richieste > Necessarie[j] ERRORE!** Il processo viola le sue stesse dichiarazioni iniziali di necessità
3. Se invece **Richieste > Disponibili** aspetta
4. Altrimenti **simula l'esecuzione** della richiesta:
    1. **Disponibili = Disponibili - Richieste**
    2. **Assegnate[j] = Assegnate[j] + Richieste**
    3. **Necessarie[j] = Necessarie[j] - Richieste**
5. **Verifica se lo stato raggiunto è sicuro**:
    1. **Se sicuro**: si effettua l'assegnazione
    2. **Se non è sicuro**: si ripristinano i valori precedenti e si sospende il processo