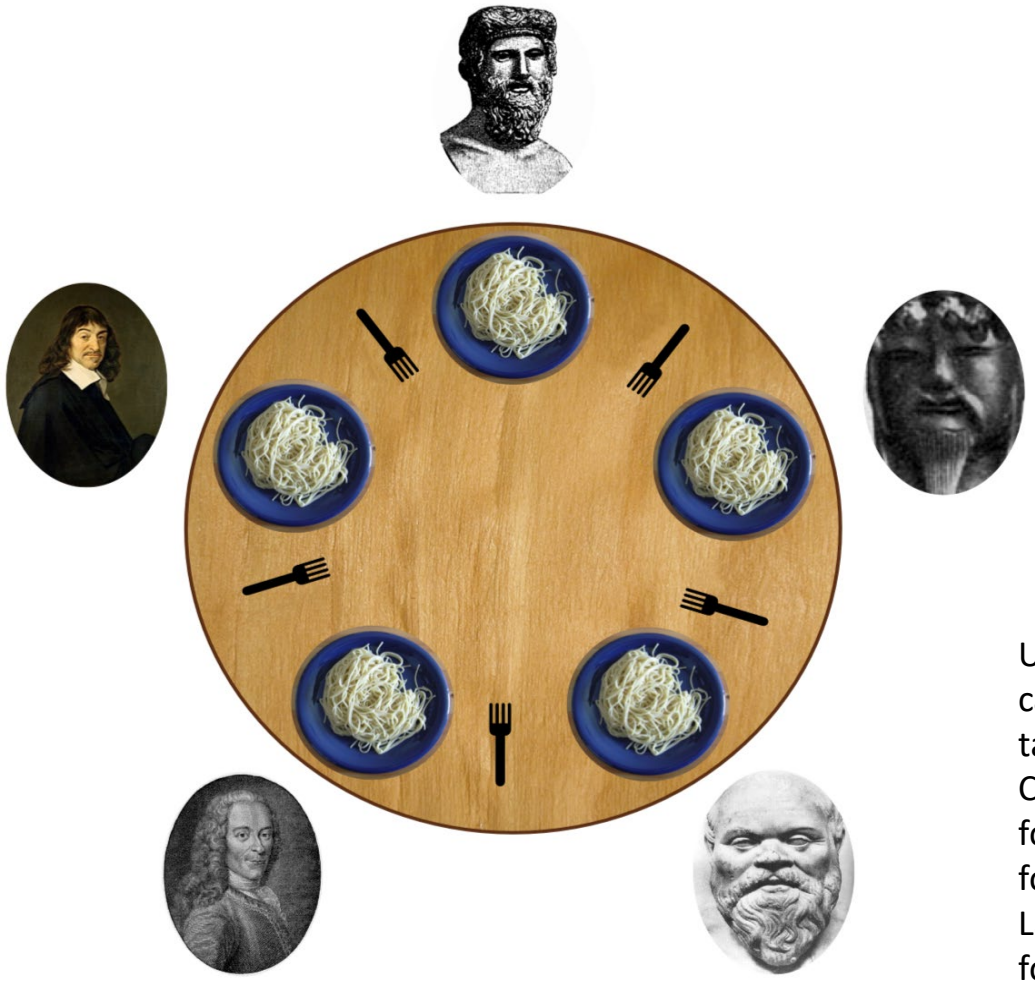
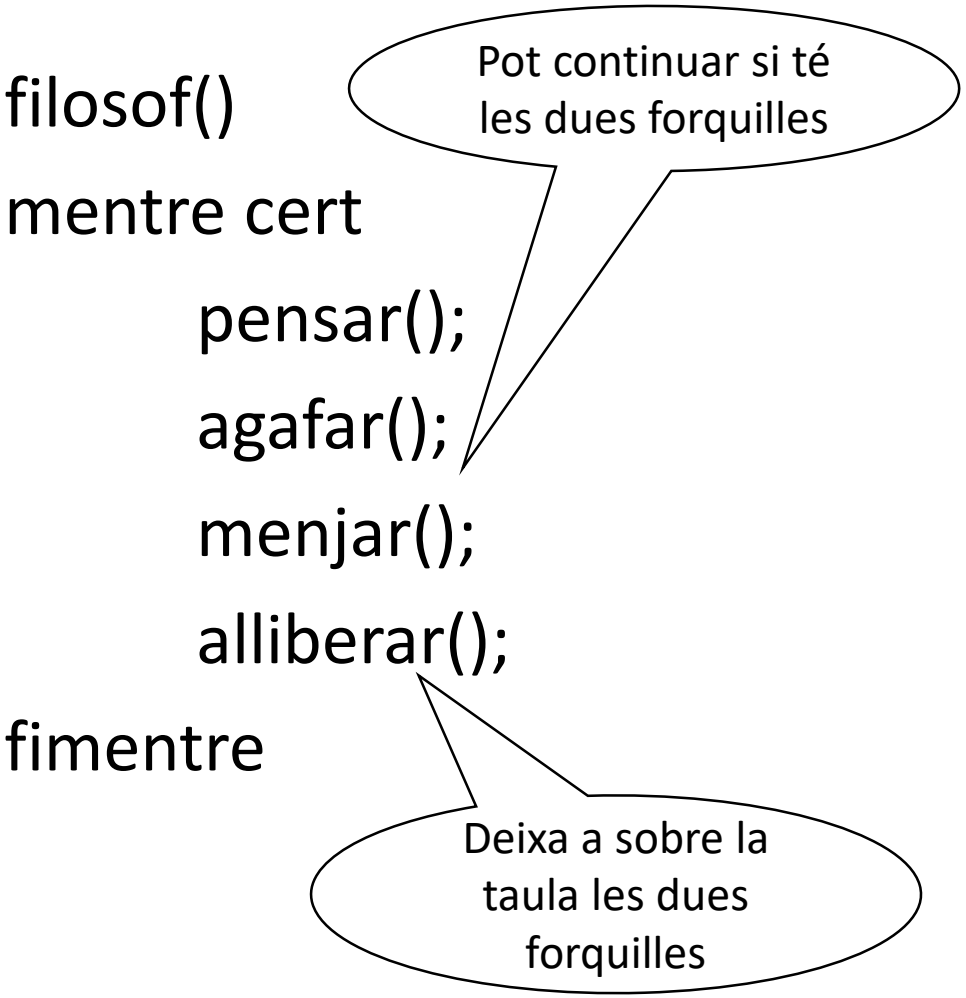
Problema del sopar dels filòsofs



El **sopar dels filòsofs** és un exemple de paradoxa que conté una abraçada mortal (o *deadlock*), proposat i resolt per Edsger Dijkstra el 1965. Cal dir que aquest problema és una adaptació d'un altre en què pensadors xinesos menjaven amb dos bastonets, on és més lògic l'ús del bastonet del filòsof veí per poder menjar.

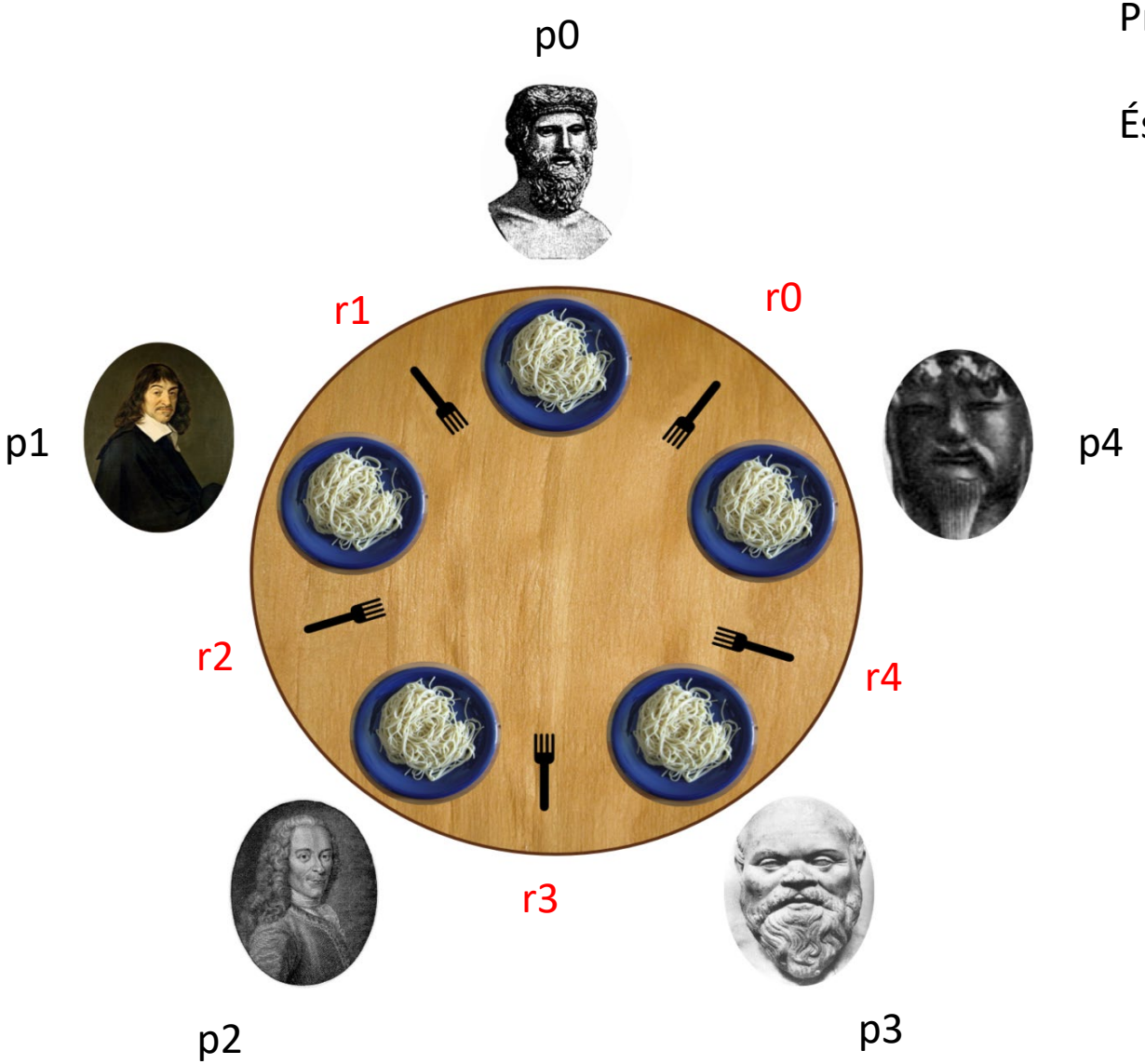
Un cert nombre N de filòsofs estan asseguts al voltant d'una taula rodona. Davant de cada filòsof hi ha un plat. A l'esquerra i a la dreta de cada filòsof hi ha una forquilla, de tal manera que hi ha N forquilles.
Cada filòsof només pot pensar o menjar. Per menjar un filòsof necessita les dues forquilles que té a la dreta i a l'esquerra. Quan acaba de menjar deixa lliure les dues forquilles.
La paradoxa s'esdevé quan tots els filòsofs volen menjar alhora, i tots agafen primer la forquilla de la mateixa banda. Aleshores tenim tots els filòsofs amb una forquilla a la mà i esperant que el filòsof del costat deixi la forquilla. Però perquè el filòsof del costat deixi la forquilla és necessari que el filòsof del seu costat deixi la forquilla, cosa que no passarà fins que el filòsof del seu costat deixi la forquilla...

Procés filòsof



S'ha de complir:

- Un filòsof només pot menjar si té els les dues forquilles disponibles.
- Exclusió mútua. Una forquilla només la pot utilitzar un filòsof
- S'ha d'assegurar que hi hagi un progrés.
- S'ha d'assegurar una espera limitada.
- Ha de ser eficient. Si no hi ha competència per un a forquilla, s'ha de poder utilitzar per algun dels filòsof veïns.



Primera solució.

Només menja un filòsof

És un problema d'exclusió mútua, es necessita un semàfor

Semafor taula = 1

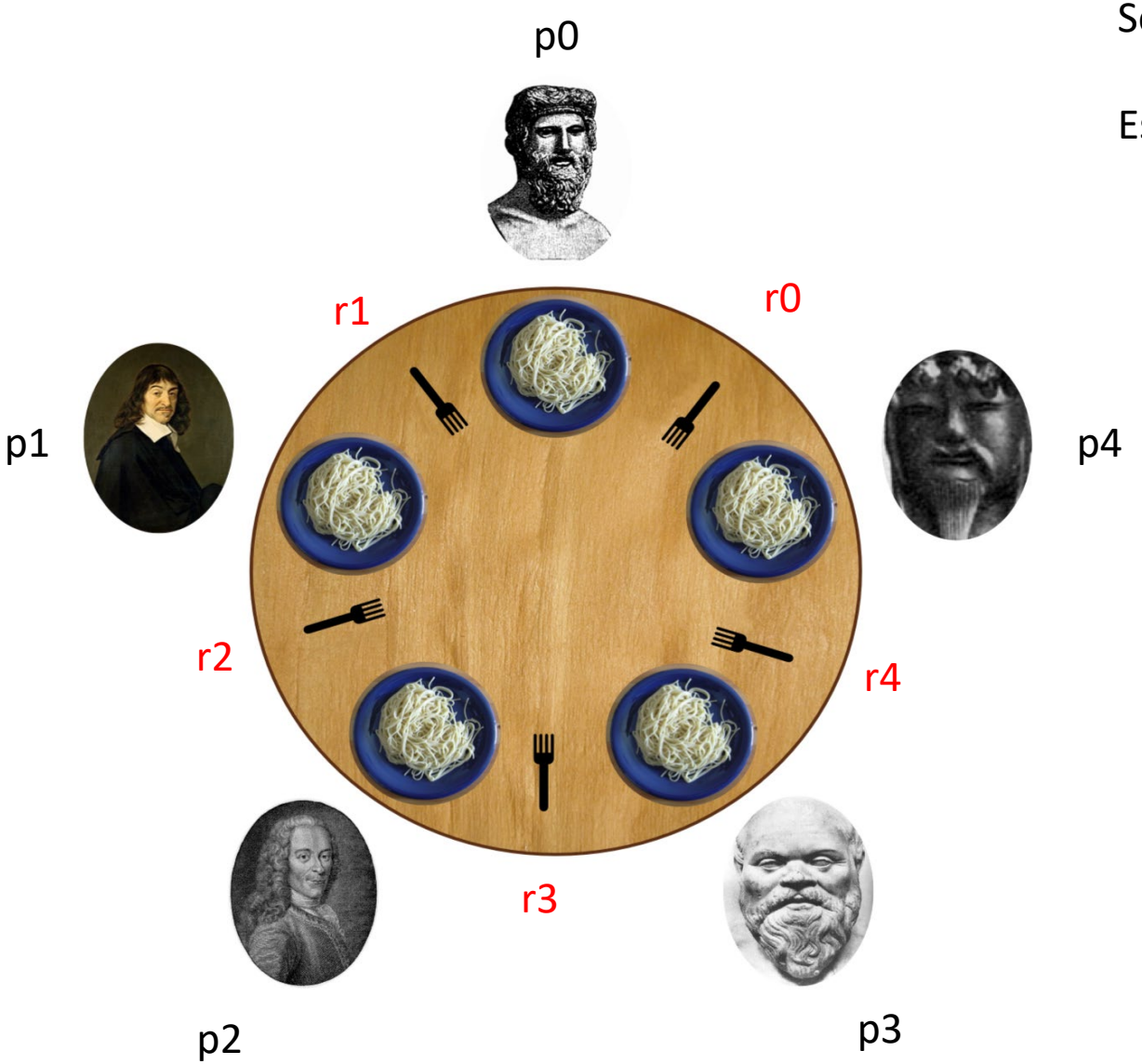
```
Filòsof()
mentre cert
        pensar();
        wait(taula);
        menjar();
        signal(taula);
fmentre
```

Amb Erlang

```
dinar() ->
        register(mutex, spawn(?MODULE, initSem, [])),
        N = 5,
        Cicles = 4,
        Pid = self(),
        spawn(fun()-> filosof(0, 'Aristotle', Cicles, Pid) end),
        spawn(fun()-> filosof(1, 'Kant', Cicles, Pid) end),
        spawn(fun()-> filosof(2, 'Spinoza', Cicles, Pid) end),
        spawn(fun()-> filosof(3, 'Marx', Cicles, Pid) end),
        spawn(fun()-> filosof(4, 'Russel', Cicles, Pid) end),
        esperaFinal(N),
        io:format("Menjador tancat.~n"),
        stop().
```

```
filosof(_Num, Nom, 0, Pid) ->
                io:format("~s marxa.~n", [Nom]),
                Pid!final,
                ok;

filosof(Num, Nom, Cycle, Pid) ->
                io:format("~s està pensant.~n", [Nom]),
                timer:sleep(rand:uniform(1000)),

                io:format("~s te gana.~n", [Nom]),
                wait(),
                io:format("~s està menjant.~n", [Nom]),
                timer:sleep(rand:uniform(1000)),
                signal(),
                io:format("~s ha deixat de menjar.~n", [Nom]),
                filosof(Num, Nom, Cycle-1, Pid).
```

```
esperaFinal(1) ->
        receive final -> io:format("No queda cap filosof ~n") end;
esperaFinal(N) ->
        receive
                final ->
                        io:format("Queden ~p filosofs~n", [N-1]),
                        esperaFinal(N-1)
        end.
```



Segona solució.

Menja més d'un filòsof

Es necessita un semàfor per cada forquilla

```
Semafor forquilla[5] = [1,1,1,1,1];

Filòsof(i)
mentre cert
        pensar();
        wait(forquilla[i]);
        wait(forquilla[(i+1)%5]);
        menjar();
        signal(forquilla[i]);
        signal(forquilla[(i+1)%5]);
fmentre
```

Amb Erlang -> filosof2.erl

Condicions necessàries per l'interbloqueig:

- Hi ha d'haver exclusió mútua: Els recursos només es poden assignar a un sol procés.

  No es pot eliminar: Un recurs només es pot assignar a un procés

- Retenció i espera: Un recurs manté assignat a un procés mentre espera l'assignació d'un altre recurs.

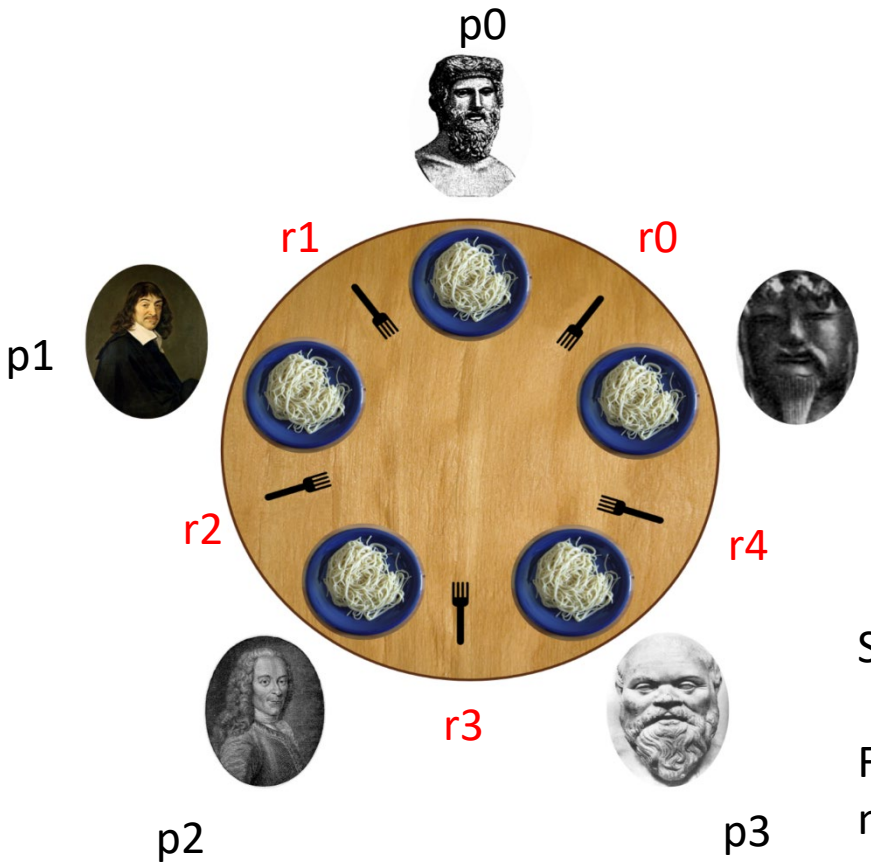
  Es pot evitar, però necessita algoritmes de sincronització més complexes

- No hi ha apropiació: No podem prendre un recurs que ja està assignat a un procés, s'ha d'esperar a que sigui alliberat.

  Es pot fer que sigui apropiatiu. Que si es detecta un interbloqueig, es pugui prendre el recurs al procés. També necessita algoritmes més complexes

- Espera circular: Es produeix un bucle, un cercle tancat de processos esperant pels recursos assignats a altre processos.

  És la més simple d'evitar. S'ha d'evitar que tots els processos sol·licitin els recursos en el mateix ordre: creixent o decreixent



Tercera solució.
Algoritme LR (Right Left)
Evita l'interbloqueig

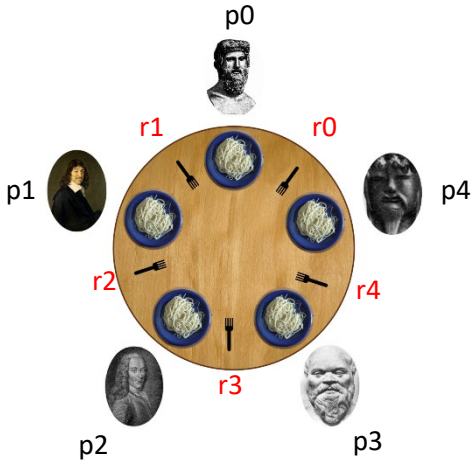
```
Semafor forquilla[5] = [1,1,1,1,1];

Filòsof(i  0:3)
mentre cert
        pensar();
        wait(forquilla[i]);
        wait(forquilla[(i+1)%5]);
        menjar();
        signal(forquilla[i]);
        signal(forquilla[(i+1)%5]);
fmentre
```

```
Filòsof(4)
mentre cert
        pensar();
        wait(forquilla[0]);
        wait(forquilla[(4]);
        menjar();
        signal(forquilla[0]);
        signal(forquilla[4]);
fmentre
```

Amb Erlang -> filosof3.erl

Solució òptima:

- En lloc d'un problema d'excussió mútua, s'ha de tractar com una sincronització d'ordre d'instruccions.
  - Quan un filòsof vol menjar, ha de verificar l'estat dels seus veïns. Podrà menjar si cap d'ells ho fa. En cas contrari, s'haurà d'esperar.
  - Es necessita
    - Llista per senyalar l'estat de cadascun dels filòsof:
      - Pensant
      - Amb gana
      - Menjant
    - Llista de semàfors per sincronitzar els filòsof (Inicialitzat a zero [bloquejat])
    - Un semàfor per assegurar l'exclusió mútua quan es modifica la llista d'estats (Variables compartida)



Semafor sincro[5] = [0,0,0,0,0];
Estat[5] = [“Pensant”, ...”Pensant”];
Semafor mutex = 1;

```
Filòsof(i)
mentre cert
        pensar();
        wait(mutex);
        Estat[i] = “Amb gana”;
        Canvi d’estat(i);
        signal(mutex);
        wait(sincro [i]);
        menjar();
        wait(mutex);
        Estat[i] = “Pensant”;
        Canvi d’estat((i-1)%5);
        Canvi d’estat((i+1)%5);
        signal(mutex);
fmentre
```

```
Canvi d’estat(i)
    si   ((Estat[i] == “Amb gana”) i
         (Estat[(i-1)%N] != “Menjant” ) i
         (Estat[(i+1)%N]!= “Menjant”))
              Estat[i] = “Menjant”;
              signal(sincro[i]);
    fsi
```

La funció: canvi d’estat, s’executa en exclusió mútua

Quan un filòsof deixa de menjar, s’ha de verificar si hi ha algun veí que estigui esperant la forquilla

Amb Erlang -> filosof4.erl