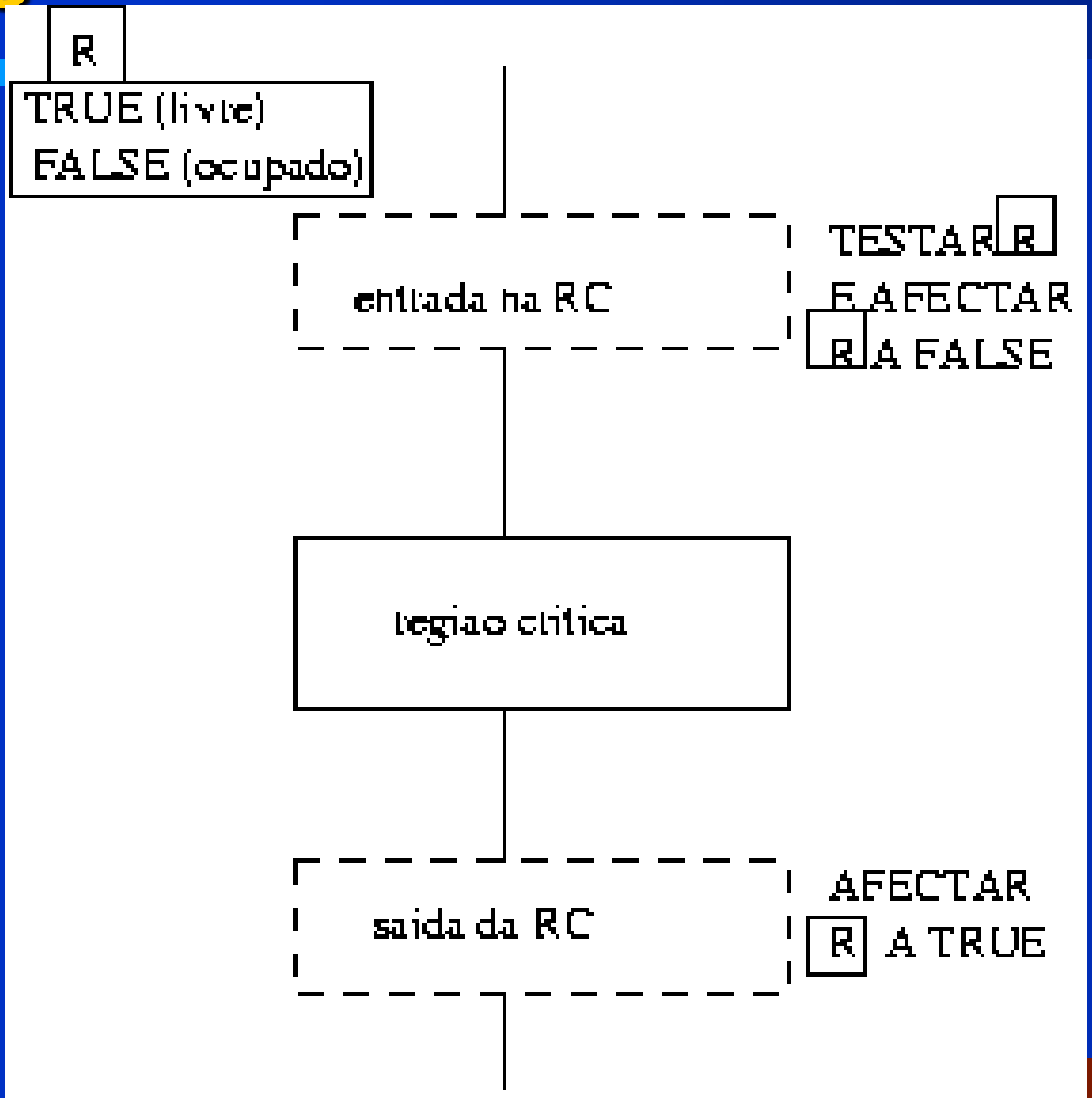
Programação Concorrente

José C. Cunha, DI-FCT/UNL

- Regiões críticas
- Soluções baseadas em Espera Activa
- Soluções baseadas em Bloqueio de Processos
- Semáforos de sincronização (E.W.Dijkstra)

Regiões Críticas – Exclusão Mútua



Acções indivisíveis ou atómicas

As acções de Entrada e Saída na Região Crítica têm de ser indivisíveis:

-- se um processo P estiver a executar Entrada, não deve poder ser interrompido e o Sistema de Operação passar o controlo da execução para outro processo Q, que vá executar Entrada também!

Porquê?

-- Isto pode suceder, mesmo que o computador só tenha 1 CPU e o SO faça multiprogramação

Porquê?

Acções atómicas hardware

a) **Em cada CPU:**

-- as instruções máquina hardware não são interrompíveis (o CPU só atende interrupções no fim da execução da instrução corrente)

-- Saída: afectar R a TRUE → está resolvido!

-- Entrada: testar R e, se estiver TRUE, afectar a FALSE → não está resolvido!

Porque exige mais do que 1 instrução máquina!

```
Ciclo: load reg, Mem[R]  ;   ler o valor de R
       jump-if-zero Ciclo ;  esperar, se for 0
       store Mem[R], 0  ;    afectar 0 a R
```

Uma solução radical?

- **Desligar o mecanismo de atendimento de interrupções pelo CPU durante a execução da Região Crítica!**

Disable interrupts;

Região Crítica;

Enable interrupts;

Disable / Enable Interrupts

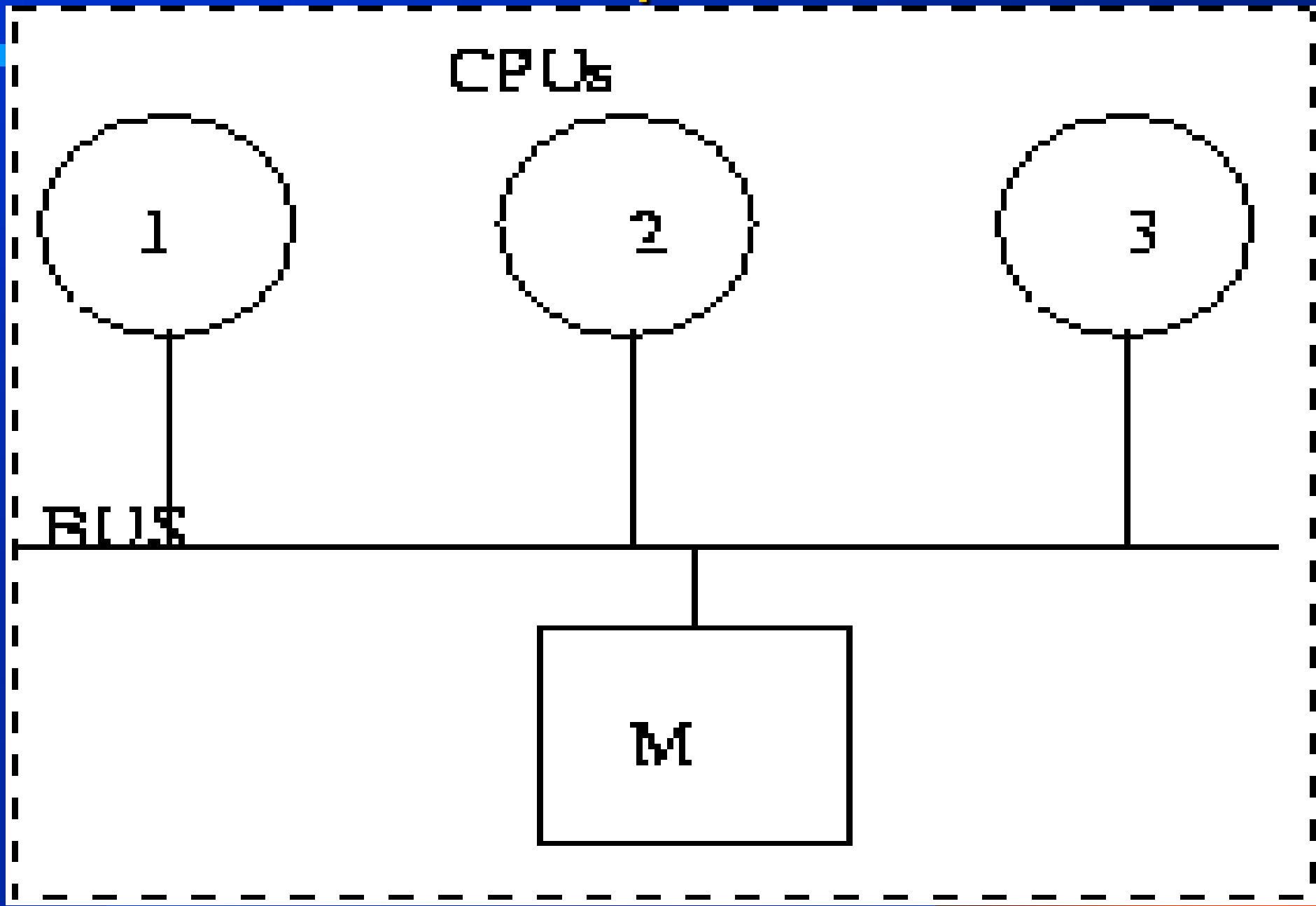
- pode demorar muito tempo;
- não é possível: para aplicações em MODO UTILIZADOR

--- só utilizável no interior do SO, para acções em MODO SUPERVISOR

e por períodos de tempo muito curtos;

- não funciona para sistemas com múltiplos CPU

Mono/Multiprocessador



Outras possíveis soluções?

Inibir o mecanismo de multiprogramação do SO, durante a Região Crítica: impede que outro processo comece a executar…

-- só pode ser aplicado no interior do núcleo do SO ou em aplicações especiais (controlo em tempo real, aplicações dedicadas ao controlo de dispositivos específicos) com acesso a funções internas do núcleo do SO!

Soluções mais gerais

- **Que sejam applicadas a Monoprocessador ou a um Multiprocessador**
- **Que sejam utilizáveis por aplicações em MODO UTILIZADOR (em vez de apenas pelo SO)**

Soluções mais gerais

Que sejam applicadas a Monoprocessador ou a um Multiprocessador

Que sejam utilizáveis por aplicações em MODO UTILIZADOR (em vez de apenas pelo SO)

- **Classes de soluções:**

a) Baseadas em Espera Activa: ciclos de teste

Soluções mais gerais

Que sejam applicadas a Monoprocessador ou a um Multiprocessador

Que sejam utilizáveis por aplicações em MODO UTILIZADOR (em vez de apenas pelo SO)

- Classes de soluções:

a) Baseadas em **Espera Activa**: ciclos de teste

b) Baseadas em **Bloqueio de processos**: suspendem a execução enquanto o processo espera

Abordagens com Espera Activa

Enquanto um recurso estiver ocupado, cada processo executa um ciclo de teste de variáveis partilhadas (Entrada):

Abordagens com Espera Activa

Enquanto um recurso estiver ocupado, cada processo executa um ciclo de teste de variáveis partilhadas (Entrada):

-- o CPU fica ocupado a executar instruções de teste, desnecessariamente

Abordagens com Espera Activa

Enquanto um recurso estiver ocupado, cada processo executa um ciclo de teste de variáveis partilhadas (Entrada):

-- o CPU fica ocupado a executar instruções de teste, desnecessariamente

--- excepto se o número de CPU for maior do que o número de Processos a executar

(é raro mas acontece…)

Abordagens com Espera Activa

Enquanto um recurso estiver ocupado, cada processo executa um ciclo de teste de variáveis partilhadas (Entrada):

-- o CPU fica ocupado a executar instruções de teste, desnecessariamente

--- excepto se o número de CPU for maior do que o número de Processos a executar

(é raro mas acontece…)

-- em alternativa: será melhor ir executando outros programas, enquanto aquele recurso não estiver livre → soluções com bloqueio

Soluções com Espera Activa

- **Duas abordagens:**

-- inteiramente baseadas em algoritmos que não exigem nenhum hardware especial

-- baseadas em algoritmos que exigem uma instrução máquina especial

Condições de uma boa solução

1. A cada instante, 1 só processo na RegCrítica;
2. Solução correcta, qualquer que seja o ritmo de progressão dos processos concorrentes;
3. O sistema não deve bloquear-se em diversos casos:

-- a Região Crítica em cada processo deve terminar em tempo finito;

-- a decisão de quem deve aceder não deve ser adiada eternamente;

-- o bloqueio / fim de um processo FORA da Região Crítica não deve afectar outros processos → deve assumir-se que os processos são independentes.

Espera Activa - Algoritmos Gerais

Exigem que o *hardware* garanta acesso atómico à Memória Central nas instruções *load* e *store*

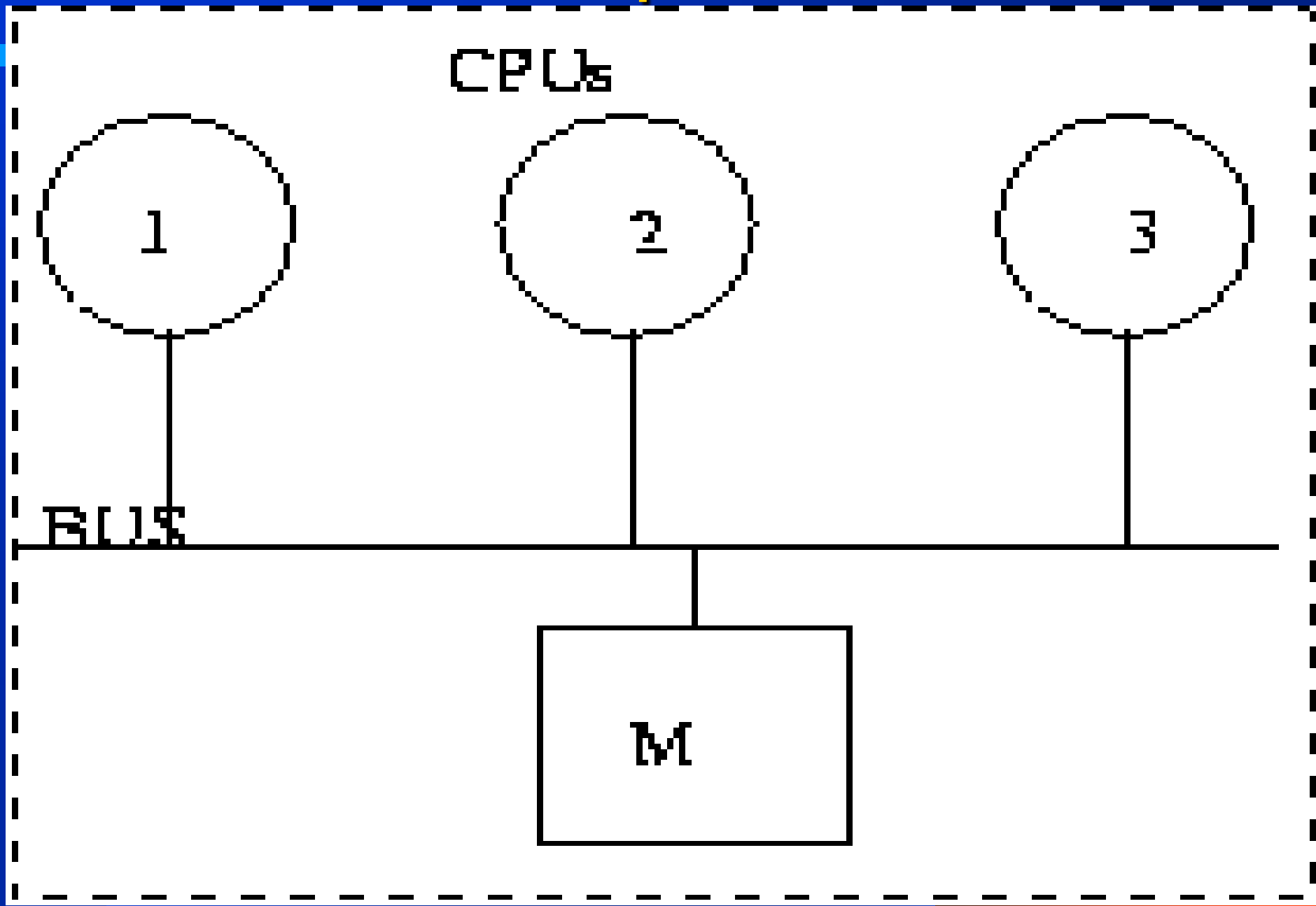
e que preservem a consistência de memória.

Resolvem o problema: Monoprocessadores ou em Multiprocessadores.

Foram inventadas nos anos 1960...

Mostram a viabilidade da solução sem *hardware especial*

Mono/Multiprocessador



```
VAR LIVRE : BOOLEAN;

PROCESSO P1;
BEGIN
 WHILE NOT LIVRE DO;
 LIVRE := FALSE;
 REGIAO CRITICA 1;
 LIVRE := TRUE;
END;

PROCESSO P2;
BEGIN
WHILE NOT LIVRE DO;
LIVRE := FALSE;
REGIAO CRITICA 2;
LIVRE := TRUE;
END;

 BEGIN
 LIVRE := TRUE;
 COBEGIN
  P1; P2
 COEND
 END.
```

```
VAR C1, C2 : INTEGER;

PROCESSO P1;
BEGIN
  C1 := 0;
  WHILE C2 = 0 DO;
  REGIAO CRITICA 1;
  C1 := 1;
END;

PROCESSO P2;
BEGIN
  C2 := 0;
  WHILE C1 = 0 DO;
  REGIAO CRITICA 2;
  C2 := 1;
END;

  BEGIN
  C1 := 1; C2 := 1;
  COBEGIN
    P1; P2
  COEND
  END.
```

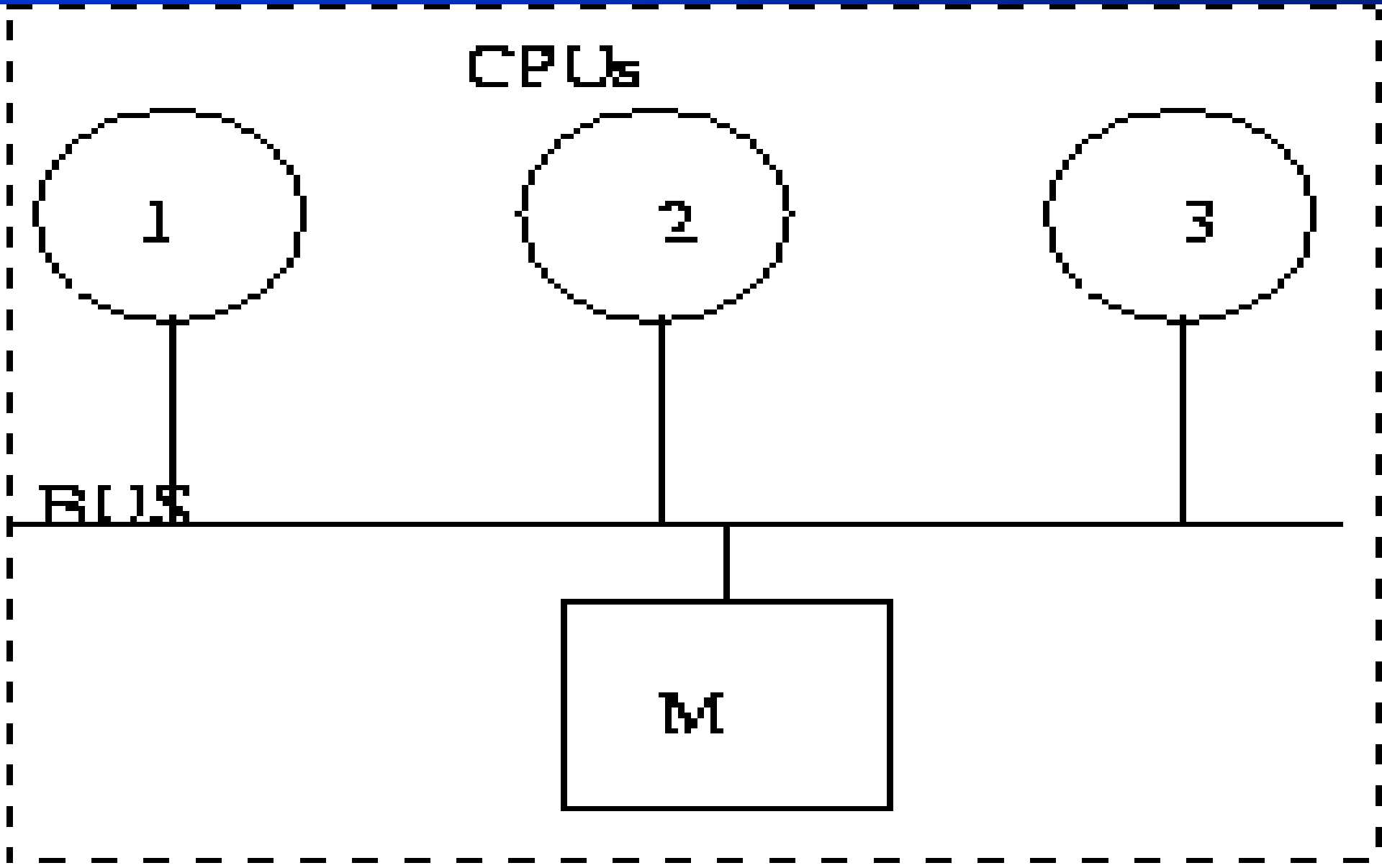
```
VAR vez: INTEGER;

PROCESSO P1;
BEGIN
  WHILE vez = 2 DO;

  REGIAO CRITICA 1;

    vez := 2;
END;

PROCESSO P2;
BEGIN
  WHILE vez = 1 DO;

  REGIAO CRITICA 2;

    vez := 1;
END;

  BEGIN
  vez := 1;
  COBEGIN
   P1; P2
  COEND
  END.
```

```
VAR C1, C2 : INTEGER;

PROCESSO P1;
BEGIN
  C1 := 0;
  WHILE C2 = 0 DO
     BEGIN C1 := 1; (* *) C1 :=0 END;
  REGIAO CRITICA 1;
    C1 := 1;
END;
PROCESSO P2;
BEGIN
  C2 := 0;
  WHILE C1 = 0 DO
     BEGIN C2 := 1; (* *) C2 :=0 END;
  REGIAO CRITICA 2;
  C2 := 1;
END;

  BEGIN
  C1 := 1; C2 := 1;
  COBEGIN
   P1; P2
  COEND
  END.
```

- Dekker

```
VAR C1, C2, vez : INTEGER;

PROCESSO P1;
BEGIN
  C1 := 0;
  WHILE C2 = 0 DO
      IF vez = 2 THEN
          BEGIN  C1 := 1;
              WHILE vez = 2 DO;
                  C1 := 0
          END;
  REGIAO CRITICA 1;
  C1 := 1;  vez := 2;
END;

PROCESSO P1;
BEGIN
  C2 := 0;
  WHILE C1 = 0 DO
      IF vez = 1 THEN
          BEGIN  C2 := 1;
              WHILE vez = 1 DO;
                  C2 := 0
          END;
  REGIAO CRITICA 1;
  C2 := 1; vez := 1;
  END;

  BEGIN
  C1 := 1; C2 := 1; vez := 1;
  COBEGIN
    P1; P2
  COEND
  END.
```

MultiprocessadorMemóriaPartilhada



Exemplo

Processo P1:

Entrada(x); RegiãoCrítica-P1; Saida(x)

Processo P2:

Entrada(x); RegiãoCrítica-P2; Saida(x)

Processo P3:

Entrada(x); RegiãoCrítica-P3; Saida(x)

Soluções baseadas em instruções hardware especiais

- **Seja x a variável partilhada de controlo do acesso:**

x = 1 → Região Crítica LIVRE (no início)

x = 0 → Região Crítica OCUPADA

Entrada: `repeat`
`res = Teste-e-Afecta-a-Zero(x)`
`until res == 1;`

Saida: `x = 1;` **(*liberta a Região Crítica*)**

Instrução Test-and-Set

- **Test-and-Set(x, reg)**

Sendo:

x: um endereço de Memória Central

reg: um Registador do CPU

- **Test-and-Set(x, reg)**

-- num passo atómico ou indivisível, faz as duas acções seguintes:

```
reg = Mem[x];   -- obtém valor corrente
Mem[x] = 0;     -- afecta-lhe 0
```

- **No processo P1:**

Entrada(x):

```
repeat
  reg1 = Test-and-Set(x, reg1)
until reg1 == 1;
```

- **No processo P2:**

Entrada(x):

```
repeat
  reg2 = Test-and-Set(x, reg2)
until reg2 == 1;
```

- **No processo P3:**

```
Saida(x):  x = 1;
```

Realização de Test-and-Set

- Num **Monoprocessador:**

é uma instrução máquina hardware:

é atómica ou indivisível (não interrompível).

- Num **Multiprocessador:**

Test-and-Set(x,reg):

- pede e aguarda acesso ao BUS (CPU em HOLD)
- obtém acesso exclusivo ao BUS
- executa as acções: (reg = Mem[x]; Mem[x] = 0)
- liberta o BUS

→ Um Árbitro Hardware controla o BUS: só um dos CPU consegue acesso ao BUS de cada vez

Nos processadores Intel 80x86

Exchange Register with Memory

XCHG reg, mem

reg recebe conteúdo da célula de memória

célula de memória recebe conteúdo de reg

Conclusões sobre soluções baseadas em Espera Activa

Se o número de CPU for maior do que o de processos

a) Algoritmo de Dekker:

→ não é modular e é dificilmente extensível para um número dinamicamente variável de processos

b) Baseado em Test-and-Set:

→ a espera activa implica um desperdício de tempo do CPU →em ambos casos a) e b)

Soluções baseadas em Bloqueio de processos

Um processo que deve aguardar, passa ao estado BLOQUEADO e liberta o CPU para executar outro programa.

Quando a Região Crítica fica LIVRE, há que reactivar um dos processos BLOQUEADOS, se os houver.

O mecanismo mais genérico, desta classe de soluções:

-- SEMÁFORO de SINCRONIZAÇÃO.

Semáforos binários (E.W.Dijkstra)

Um tipo de dados abstracto caracterizado por:

-- um valor inicial: 1

-- duas operações indivisíveis ou atómicas:

P(S): o processo invocador bloqueia-se até que S ==1 e, então, põe S a 0 ***(Lock)***

P = ´´*proberen*´´: ´´testar´´, em Holandês.

V(S): o processo invocador põe S a 1

e prossegue sem se bloquear ***(Unlock)***

V = ´´verhogen´´: ´´incrementar´´

Soluções com Bloqueio de Processos

para exclusão mútua entre Processos distintos:

Semáforos Binários

Semáforos (E.W.Dijkstra)

Tipo de dados abstracto caracterizado por:

-- um valor inteiro não negativo (S)

-- três operações indivisíveis:

Semáforos (E.W.Dijkstra)

Tipo de dados abstracto caracterizado por:

-- um valor inteiro não negativo (S)

-- três operações indivisíveis:

Inicializar(S): atribui um valor não negativo

Semáforos (E.W.Dijkstra)

Tipo de dados abstracto caracterizado por:

-- um valor inteiro não negativo (S)

-- três operações indivisíveis:

Inicializar(S): atribui um valor não negativo

P(S): o processo invocador bloqueia-se até que S>0 e então decrementa S em 1 unidade

P = ´´*proberen*´´: ´´testar´´, em Holandês.

Semáforos (E.W.Dijkstra)

Tipo de dados abstracto caracterizado por:

-- um valor inteiro não negativo (S)

-- três operações indivisíveis:

Inicializar(S): atribui um valor não negativo

P(S): o processo invocador bloqueia-se até que S>0 e então decrementa S em 1 unidade

P = ´´*proberen*´´: ´´testar´´, em Holandês.

V(S): o processo invocador incrementa S de uma unidade e prossegue sem se bloquear

V = ´´verhogen´´: ´´incrementar´´

Realização de Semáforos

- **Cada Semáforo é representado por:**

-- uma variável inteira (valor S)

-- uma fila de espera de processos bloqueados f(S)

Realização de Semáforos

- **Cada Semáforo é representado por:**

-- uma variável inteira (valor S)

-- uma fila de espera de processos bloqueados f(S)

- **Quando há processos concorrentes e 1 só CPU:**

-- um só processo no estado EXECUTANDO;

Realização de Semáforos

- **Cada Semáforo é representado por:**

-- uma variável inteira (valor S)

-- uma fila de espera de processos bloqueados f(S)

- **Quando há processos concorrentes e 1 CPU:**

-- um só processo no estado EXECUTANDO;

-- outros: PRONTOS, mas aguardam a sua vez:

estão numa fila de processos prontos (READY);

Realização de Semáforos

- **Cada Semáforo é representado por:**

-- uma variável inteira (valor S)

-- uma fila de espera de processos bloqueados f(S)

- **Quando há processos concorrentes e 1 CPU:**

-- um só processo no estado EXECUTANDO;

-- outros: PRONTOS, mas aguardam a sua vez:

estão numa fila de processos prontos (READY);

-- se a operação P(S) bloqueia: o processo passa a

BLOQUEADO na FILA f(S)

Realização de Semáforos

- **Cada Semáforo é representado por:**

-- uma variável inteira (valor S)

-- uma fila de espera de processos bloqueados f(S)

- **Quando há processos concorrentes e 1 CPU:**

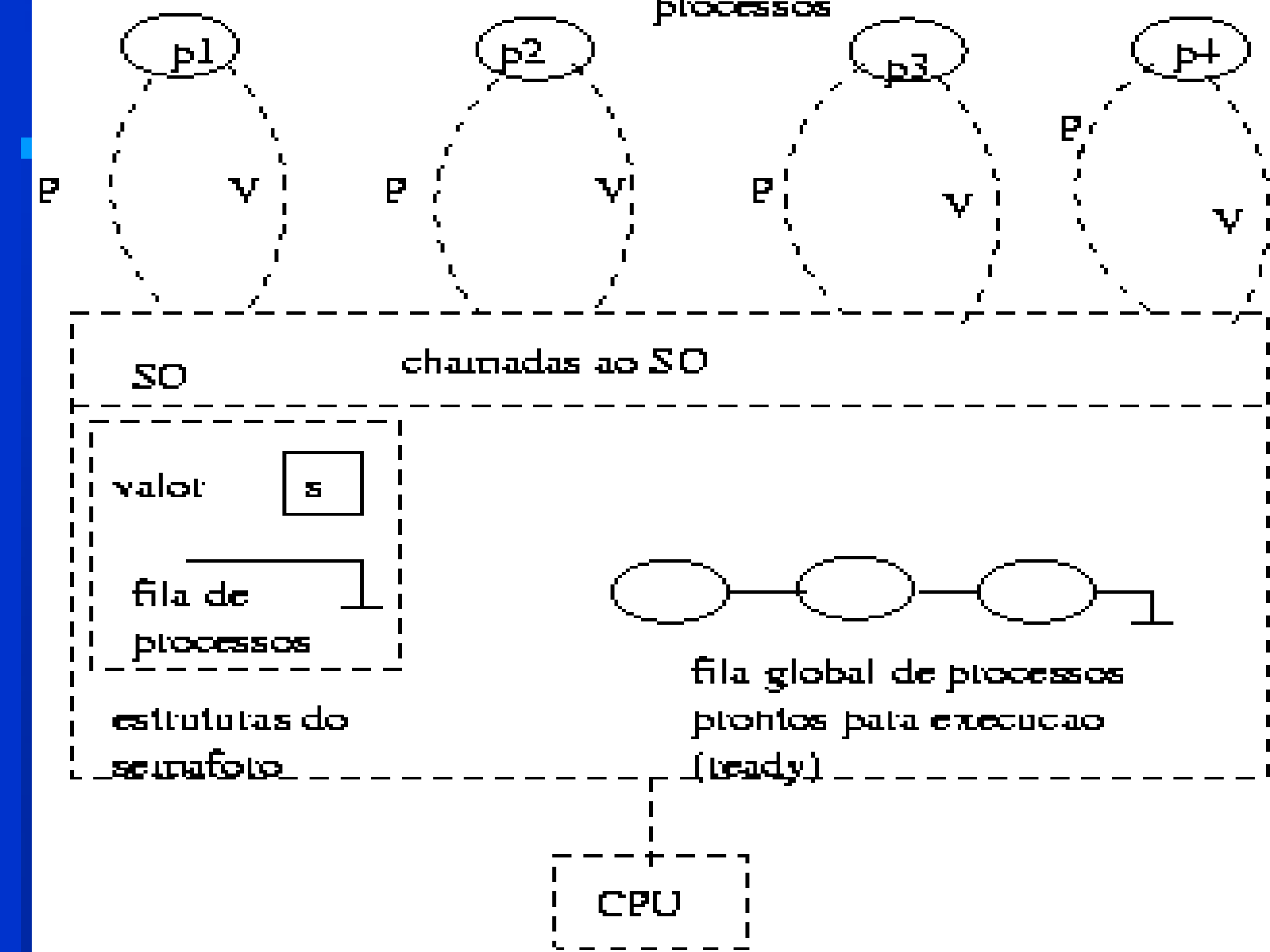
-- um só processo no estado EXECUTANDO;

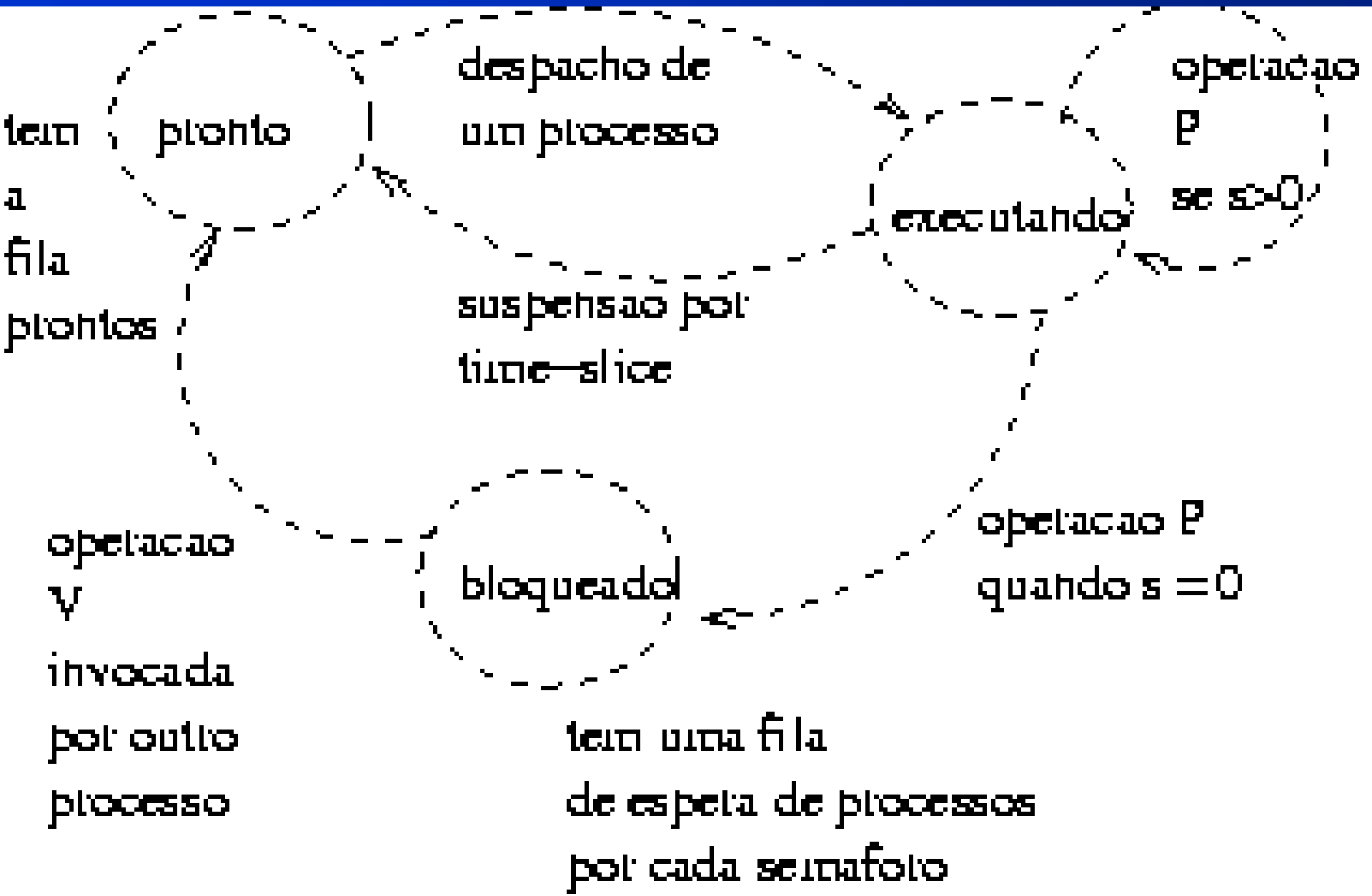
-- outros: PRONTOS, mas aguardam a sua vez:

estão numa fila de processos prontos (READY);

-- se a operação P(S) bloqueia: o processo passa a BLOQUEADO na FILA f(S)

-- se um processo faz V(S): pode permitir que um outro processo volte a ficar PRONTO e saia da fila f(S) voltando à fila READY.


processos
p1
p2
p3
p4
P
V
P
V
P
V
P
V
SO
chamadas ao SO
valor
s
fila de
processos
estrututas do
semafoto
fila global de processos
prontos para execucao
(ready)
CPU


tem
a
fila
prontos
pronto
despacho de
um processo
executando
operacao
P
se s>0
suspensao por
time-slice
operacao
V
invocada
por outro
processo
bloqueado
operacao P
quando s =0
tem uma fila
de espera de processos
por cada semaforo

Uma implementação de Semáforos:

Há uma fila única de processos aguardando a vez de execução: fila_PRONTOS

Há uma fila associada a cada semáforo S:

fila_semáforo_f(S)

onde aguardam os processos bloqueados no semáforo S

```
P(S): (*operação indivisível*)
  if S>0 then S = S-1
  else
     begin (*bloqueia este processo*)
       salvaguarda estado/regs da máquina;
       estado(processo) = BLOQUEADO;
       inserir(processo, fila_semáforo_f(S));

       rotina-despacho-SO escolhe outro processo
       da fila_PRONTOS, para execução
     end
```

Endereço de retorno após P()

.....

P(S); ← chamada ao sistema

E: ..... ← ponto onde retomará a execução

.....

Se P(S) bloqueia o processo, o valor do Program Counter (PC) que é guardado pelo SO, é o valor do endereço de retorno E, como se fosse um CALL de subrotina….

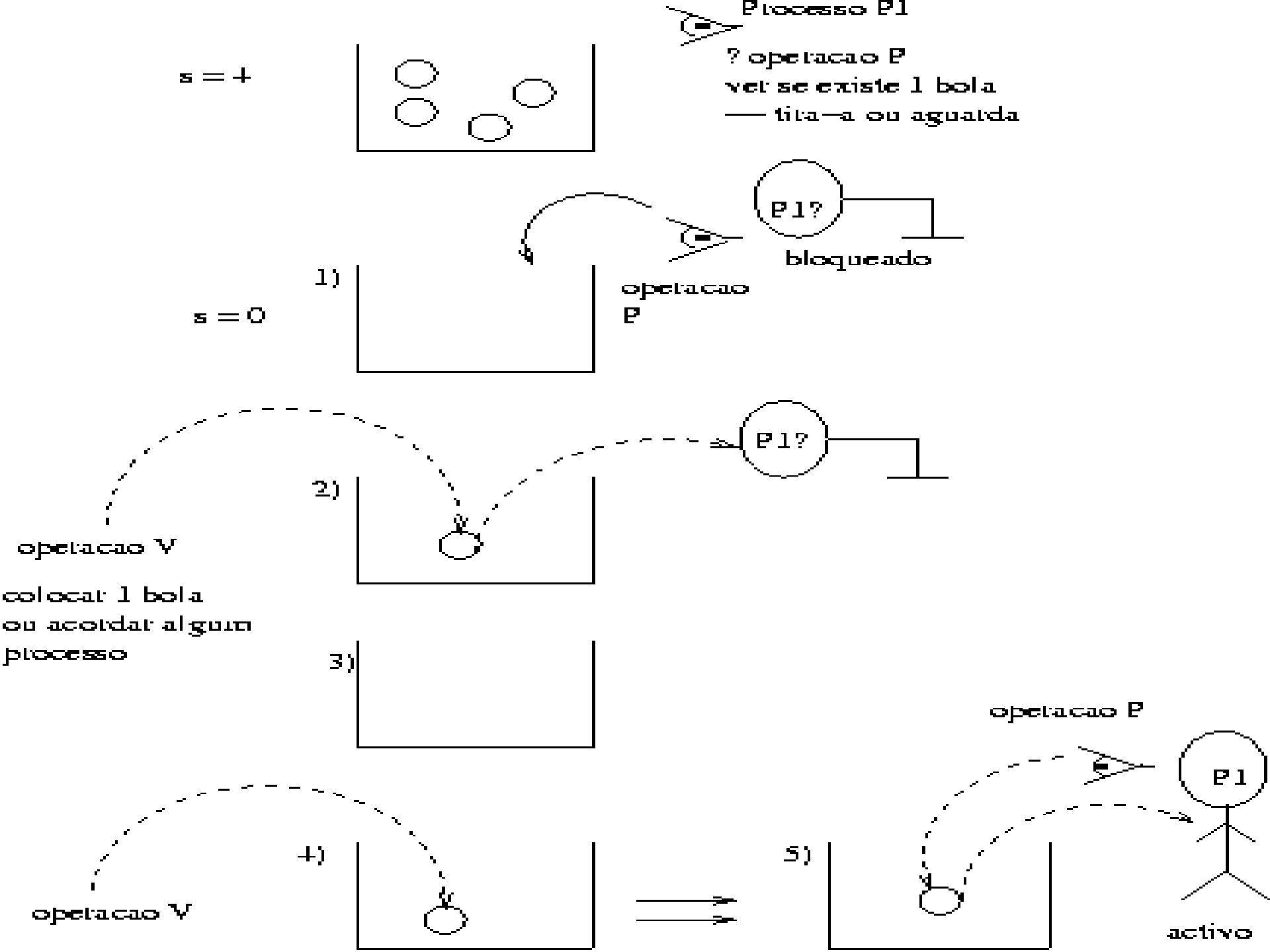
```
V(S): (*operação indivisível*)
  if (fila_semáforo_f(S) VAZIA) then S = S+1
  else
     begin (*desbloqueia um processo*)
        remover(fila_semáforo _f(S), Proc);
        inserir(Proc, fila_PRONTOS);
        estado(Proc) = PRONTO
     end
```

P(s):

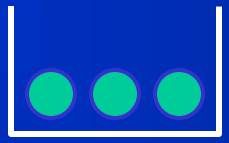
```
IF s > 0 THEN s := s - 1 ;
ELSE
 BEGIN
  guarda estado do processo corrente
  em zonas de memoria do SO;

  estado(processo_corrente) := bloqueado';

  insere(processo_corrente, fila do semaforo);

  rotina despacho_do_SO escolhe novo processo;
 END
```

V(s):

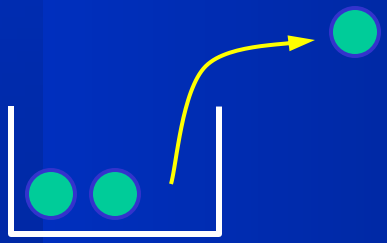
```
IF fila do semaforo vazia THEN s := s + 1;
ELSE
 BEGIN
  remove um processo (proc) da fila do semaforo;

  insere(proc, fila de processos prontos);

  estado(proc):= pronto;
 END
```


Processo P1
? operacao P
ver se existe 1 bola
— tira-a ou aguarda
s = +
P1?
bloqueado
1)
s = 0
operacao
P
P1?
2)
operacao V
colocar 1 bola
ou acordar algum
processo
3)
operacao P
P1
4)
5)
operacao V
activo

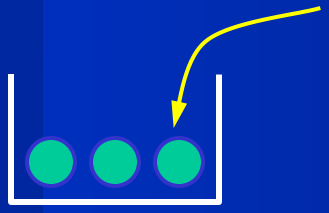
semáforo com o valor 3
caixa com 3 bolas



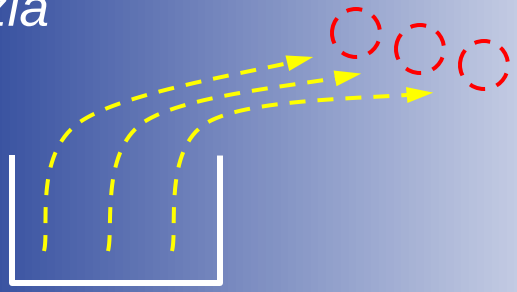
operação P
tirar uma bola



operação V
pôr uma bola



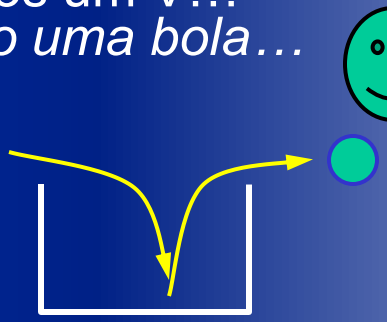
após 3 P o semáforo fica a 0
caixa vazia



os P seguintes bloqueiam
aguarda-se que haja bolas na caixa

à espera de bolas

após um V…
pondo uma bola…



à espera de bolas

Regiões Críticas – Exclusão mútua

Basta inicializar um Semáforo exmut
com valor igual a 1
no início a fila f(exmut) está VAZIA.
e cada processo deve fazer:

P(exmut);

Região Crítica;

V(exmut)

Exemplo

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut)

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut)

* **(valor do PC guardado)**

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

* **(valor do PC guardado)**

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

* **(valor do PC guardado)**

Os 3 processos estão PRONTOS. O Sistema de Operação escolhe um deles para EXECUTANDO

- **Processo P1 faz P(exmut)**

Como o valor S>0, o processo não se bloqueia

Fica S=0 e P1 prossegue EXECUTANDO

- **Processo P1 faz P(exmut)**

Como o valor S>0, o processo não se bloqueia

Fica S=0 e P1 prossegue EXECUTANDO

- **Se P1 perde o controlo durante a Região Crítica por expirar o TIME-SLICE:**

- **Processo P1 faz P(exmut)**

Como o valor S>0, o processo não se bloqueia

Fica S=0 e P1 prossegue EXECUTANDO

- **Se P1 perde o controlo durante a Região Crítica por expirar o TIME-SLICE:**

-- P1 passa para o fim da FILA PRONTOS

-- outro processo é escolhido para EXECUTANDO: seja o processo P2

- **Processo P1 faz P(exmut)**

Como o valor S>0, o processo não se bloqueia

Fica S=0 e P1 prossegue EXECUTANDO

- **Se P1 perde o controlo durante a Região Crítica por expirar o TIME-SLICE:**

-- P1 passa para o fim da FILA PRONTOS

-- outro processo é escolhido para EXECUTANDO: seja o processo P2

- **Processo P2 faz P(exmut)**

Como S=0

- **Processo P1 faz P(exmut)**

Como o valor S>0, o processo não se bloqueia

Fica S=0 e P1 prossegue EXECUTANDO

- **Se P1 perde o controlo durante a Região Crítica por expirar o TIME-SLICE:**

-- P1 passa para o fim da FILA PRONTOS

-- outro processo é escolhido para EXECUTANDO: seja o processo P2

- **Processo P2 faz P(exmut)**

Como S=0: o processo P2 bloqueia-se na FILA f(exmut) e passa a BLOQUEADO

-- outro processo é escolhido para EXECUTANDO: seja o processo P3

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut)

*** (valor do PC guardado)**

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

*** (valor do PC guardado)**

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

*** (valor do PC restaurado)**

- **Processo P3 faz P(exmut)**

Como o valor S=0, o processo P2 bloqueia-se na FILA f(exmut) e passa a BLOQUEADO

-- outro processo é escolhido para EXECUTANDO: seja o processo P1

- **Processo P3 faz P(exmut)**

Como o valor S=0, o processo P2 bloqueia-se na FILA f(exmut) e passa a BLOQUEADO

-- outro processo é escolhido para EXECUTANDO: seja o processo P1

Na FILA f(exmut) estão bloqueados: P2 e P3

Valor do semáforo S=0

No estado EXECUTANDO: P1

retoma a execução a partir do ponto onde tinha ficado

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut)

***** **(valor do PC restaurado)**

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

***** **(valor do PC guardado)**

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

***** **(valor do PC guardado)**

- **Processo P1 completa a Região Crítica:**

V(exmut): como a fila f(exmut) NÃO está VAZIA:

- **Processo P1 completa a Região Crítica:**

V(exmut): como a fila f(exmut) NÃO está VAZIA:

-- um dos processos é escolhido e passa a PRONTO:

seja o processo P2;

na FILA f(exmut) mantém-se bloqueado: P3

Valor do semáforo S=0

-- o processo P1 não se bloqueia e prossegue EXECUTANDO

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut);

*****

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

***** **(valor do PC guardado)**

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

***** **(valor do PC guardado)**

- **Processo P1 termina.**
- **O SO escolhe um dos processos PRONTOS:**

o processo P2 é o único

-- o processo P2 retoma a execução a partir do ponto onde tinha ficado

na FILA f(exmut) mantém-se bloqueado: P3

Valor do semáforo S=0

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut);

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

* (valor do PC restaurado)

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

* (valor do PC guardado)

- **Processo P2 completa a Região Crítica:**

V(exmut): como a fila f(exmut) NÃO está VAZIA:

-- um dos processos é escolhido e passa a PRONTO:

é o processo P3;

a FILA f(exmut) fica VAZIA

Valor do semáforo S=0

-- o processo P2 não se bloqueia e prossegue EXECUTANDO

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut);

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

*

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

* (valor do PC guardado)

- **Processo P2 termina.**
- **O SO escolhe um dos processos PRONTOS:**

o processo P3 é o único

-- o processo P3 retoma a execução a partir do ponto onde tinha ficado

a FILA f(exmut) mantém-se VAZIA

Valor do semáforo S=0

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut);

*

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

*

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

* (valor do PC restaurado)

- **Processo P3 completa a Região Crítica:**

V(exmut): como a fila f(exmut) ESTÁ VAZIA:

-- incrementa o valor do semáforo

a FILA f(exmut) mantém-se VAZIA

Valor do semáforo S=1

-- o processo P3 não se bloqueia e prossegue EXECUTANDO

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut);

*

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

*

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut);

*

- **Processo P3 termina.**
- **O SO escolhe um dos processos PRONTOS:**

  .....

a FILA f(exmut) mantém-se VAZIA

Valor do semáforo S=1

(tal como no início)

Processo P1:

P(exmut); RegiãoCrítica-P1; V(exmut);

*

Processo P2:

P(exmut); RegiãoCrítica-P2; V(exmut)

*

Processo P3:

P(exmut); RegiãoCrítica-P3; V(exmut)

*

Outros Problemas de sincronização

Semáforo: mecanismo genérico para resolver problemas de interacção de processos concorrentes.

- **Tipos de utilização de semáforos:**

-- semáforos de exclusão mútua (inicial: 1)

-- semáforos contadores (inicial: n)

-- semáforos privados (só um processo faz P)

- **Exemplos….**

Produtores e consumidores

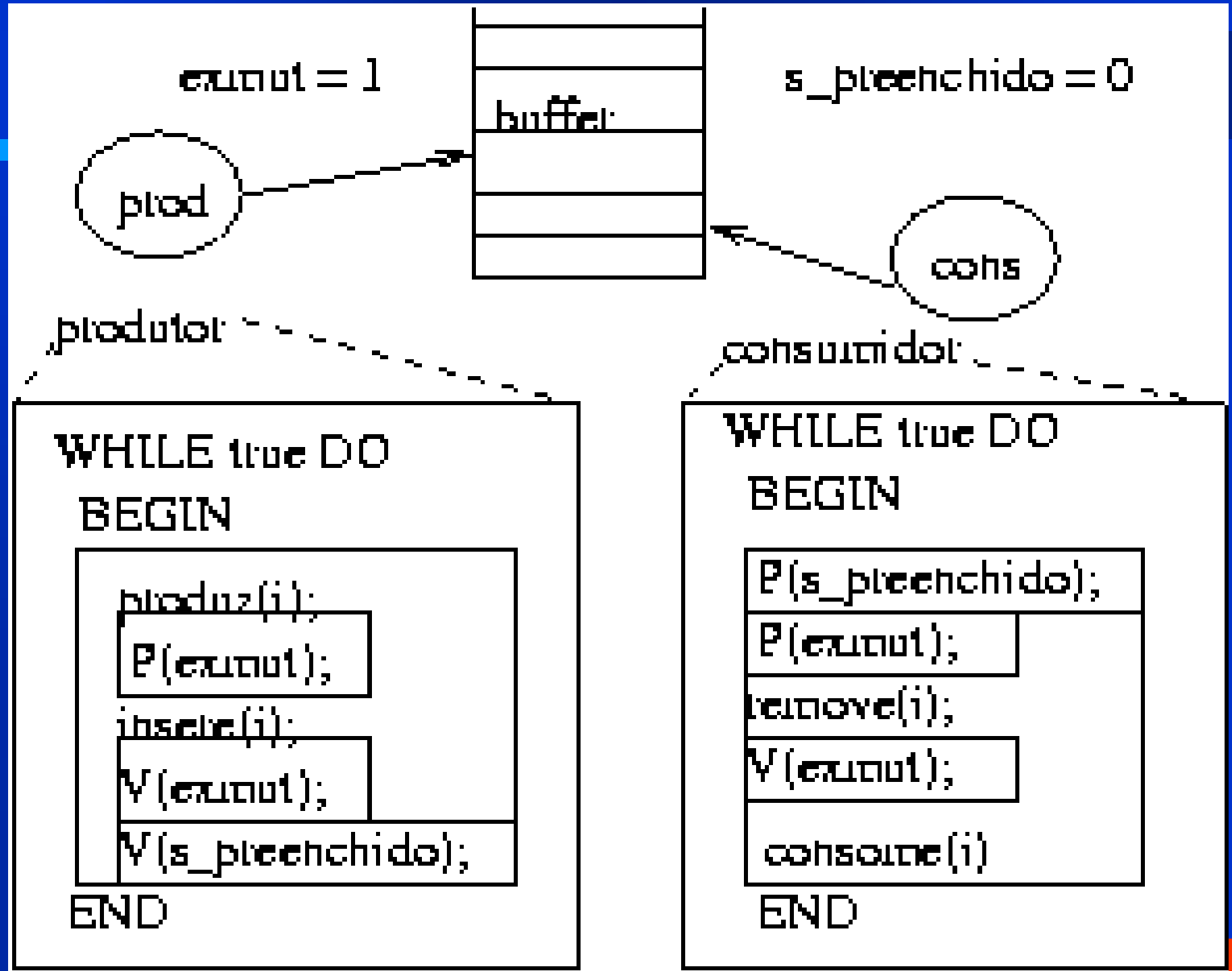
- **Processo Produtor:**
  -- produz resultados
  -- põe num Buffer
- **Processo Consumidor:**
  -- retira do Buffer
  -- trata valores obtidos
- **O Buffer está em Memória Partilhada.**

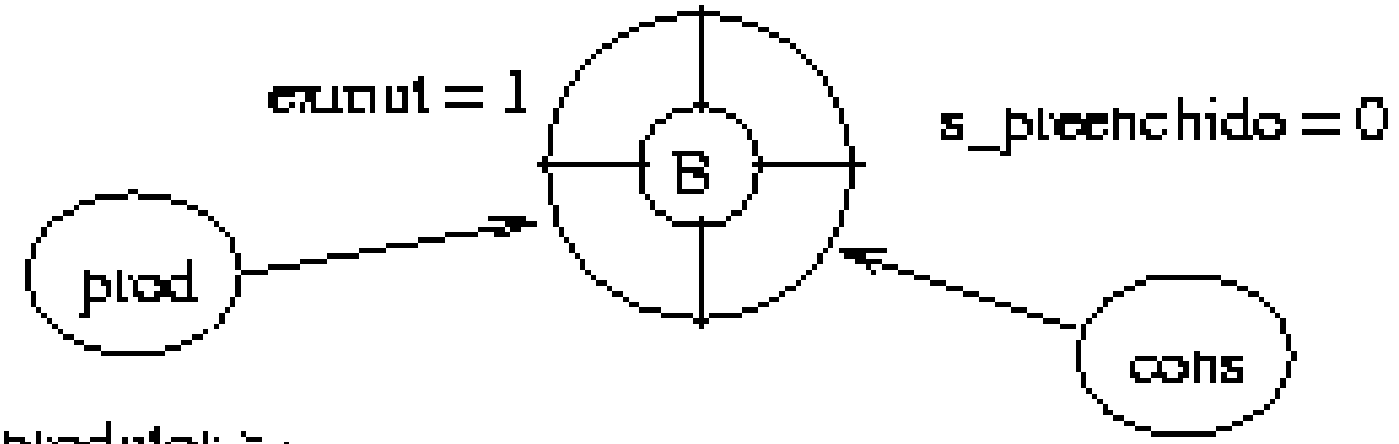
Que problemas de sincronização?

1. **Acesso concorrente ao Buffer:** deve ser uma Região Crítica
2. **Consumidor:** não pode retirar, se o Buffer estiver VAZIO
3. **Produtor:** não pode colocar, se o Buffer estiver cheio (MAX : capacidade)

- **Soluções baseadas em Semáforos:**

1. Um semáforo de exclusão mútua
2. Um semáforo contador (inicial 0)
3. Um semáforo contador (inicial MAX)


exmut = 1
buffer
s_preenchido = 0
prod
cons
produtor
consumidor
WHILE true DO
BEGIN
produz(i);
P(exmut);
insere(i);
V(exmut);
V(s_preenchido);
END
WHILE true DO
BEGIN
P(s_preenchido);
P(exmut);
remove(i);
V(exmut);
consome(i)
END



produtor

```
WHILE true DO
  BEGIN
    produz(i);
    P(s_vazias);
    P(exmut);
    insere(i);
    V(exmut);
    V(s_preenchido);
  END
```

consumidor

```
WHILE true DO
  BEGIN
    P(s_preenchido);
    P(exmut);
    remove(i);
    V(exmut);
    consome(i)
    V(s_vazias);
  END
```

STOP

Soluções com Bloqueio de Processos

para exclusão mútua entre Processos distintos:

Semáforos Binários

equivalentes ao tipo de dados mutex definido pela norma POSIX: usado para Threads de um mesmo Processo

mutual exclusion → mutex

Mutex - exclusão mútua -- POSIX

- **Mecanismo para exclusão mútua entre *Threads* de um mesmo Processo**
- **Tipo de dados: `pthread_mutex_t`**
- **Possui dois estados:**
  - **Fechado (0) ou Aberto (1)**

- **Operações:**

```
pthread_mutex_init

pthread_mutex_lock      /*reservar*/
pthread_mutex_unlock    /*libertar*/

pthread_mutex_destroy
```

Uso de um *mutex entre Threads*

- **Garantir o acesso exclusivo a um recurso partilhado:**

```
pthread_mutex_t exmut;
pthread_mutex_init( &exmut, &attr );
  -- cria mutex      (attr é opcional, pode ser NULL)
```

em cada *thread*:

```
pthread_mutex_lock( &exmut );
  -thread bloqueia até exmut estar livre
Reg. Crítica…
pthread_mutex_unlock( &exmut );
  -liberta exmut e permite reactivar um
  thread bloqueado
```