Ações atômicas e o comando await

duas formas de obtermos "ordens" corretas

- exclusão mutua (combinar ações em blocos atômicos
- condição de sincronização - atrasar a execução até que uma condição seja válida.

Ações atômicas de grão fino são implementadas pelo hardware

Exemplo:

```
int y=0, z=0;
co {
    <x = y + 2>;
    //
    <y = 1>, <z = 2>;
}
```

possíveis valor de x:

0, 1, 3, 2 → otimização

Referência crítica: referência para uma variável modificada por outro processo concorrente.

De entre dois processos concorrentes não há referências críticas a execução parece atómica

- nenhum dos valores dos quais um depende muda
- o outro processo não vê valores intermediários

Ex:

```
int x=0, y=0;
co {
    x = x + 1;
    y = y + 1;
}
```

Mas, em geral isto não acontece.

Logo usa-se um requisito mais geral:

- Propriedade no máximo uma vez

uma atribuição $x = e$ satisfaz a propriedade se:

1- e contém no máximo 1 referência crítica, e x não é lido por outro processo, ou

2- e não contém referências críticas, nesse caso x pode ser lido.

Idéia:
Ter execuções que pareçam atômicas com apenas uma referência não é possível saber quando ocorre a atualização

Ex.:

```
int x=0, y=0;
co {
    x = y + 1;
    // ; referência crítica
    y = y + 1;
}
```

vale no máximo 1 vez (0 ou 1) e será consistente no contexto.

```
int x = 0, y = 0
co {
    x = y + 1;        não satisfaz
    //
    y = x + 1;
}
```

```
int y = 1, x, z;
co {
    y = y + 1;
    //
    x = y;
    //
    z = y;
    //
    if (x != z) print ("Bingo")
}
```

não vale

Primitiva para sincronização: await

conceito com o objetivo de criar uma sequência de comandos atômicos

< await (B) S; >

B: condição de atraso

S: comando

Exemplos:

< await (s > 0) s = s - 1; >

x = 0, y = 0;

< x = x + 1; y = y + 1; > → excl. mútua

< await (count > 2); >

a implementação desta primitiva é difícil.

importante: se a condição B respeita a propriedade no máximo uma vez, < await (B); > é equivalente a while (!B).

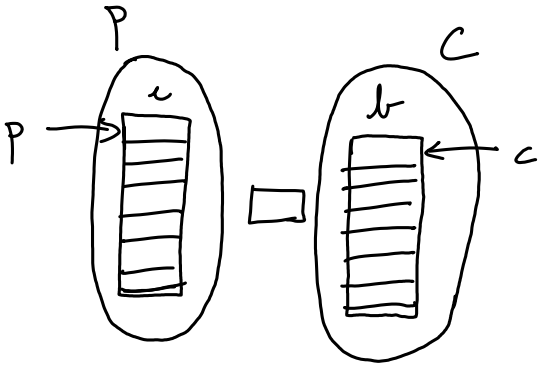
Exemplo:

Produtor / Consumidor com comunicação através de uma variável buffer

Produtor contém um vetor a [n]

Consumidor deve copiar a[n] em b[n]

variáveis p e c que controlam os itens produzidos e consumidos



condição de sincronização

$$0 <= p <= c + 1$$

```
int buf, p = 0, c = 0;
proces Producer {
    int a[n];
    while (p < n) {
        < await (p == c); > while (p > c);
        buf = a[p];
        p = p + 1;
    }
}
```

```
process Consumer {
    int b[n];
    while(c<n){
     < await (p>c); >   while(p==c);
     b[c]=buf;
     c=c+1;
    }
}
```