Variáveis de Condição

Sistemas Distribuídos

Variáveis de condição

- permitem que threads suspendam/retomem a sua execução dentro de secções críticas, de acordo com uma dada condição
- métodos: wait(), notify(), notifyAll()
- estão associadas a um monitor/lock; cada objecto tem uma variável de condição intrínseca (wait-set)

- Exemplo:

T1

```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```

obj

x = 0
Monitor: ***em espera p/ monitor:***   **Wait-Set:**

Exemplo:

T1

```
metodo1(){
   synchronized(obj) {
      while(obj.x == 0){
        obj.wait();
      }
       obj.x+=1;
    }
}
```

T2

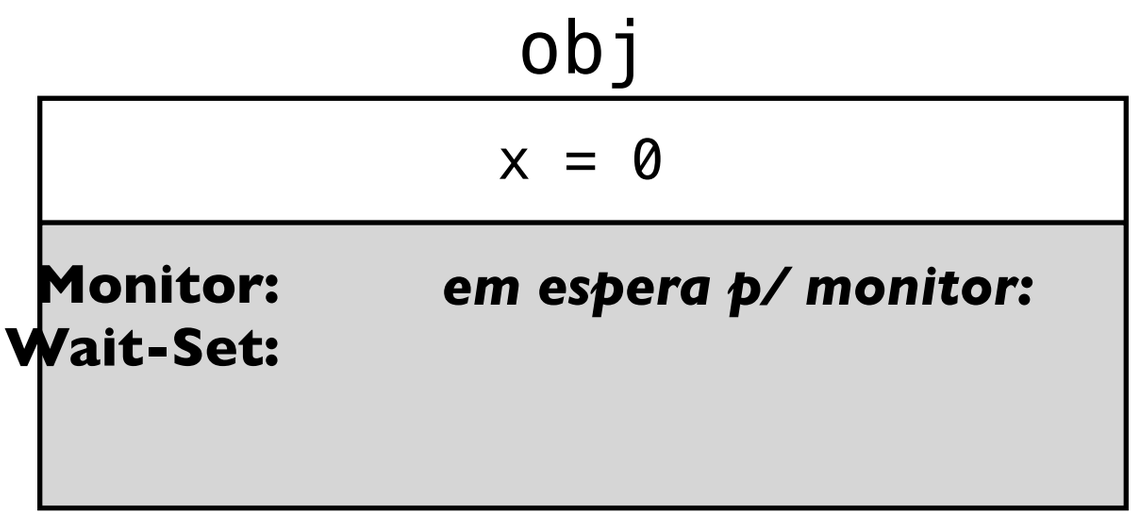
Obtém lock intrínseco do objeto **obj**

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```

Usa variável de condição associada ao lock intrínseco do objeto **obj**



- Exemplo:

T1

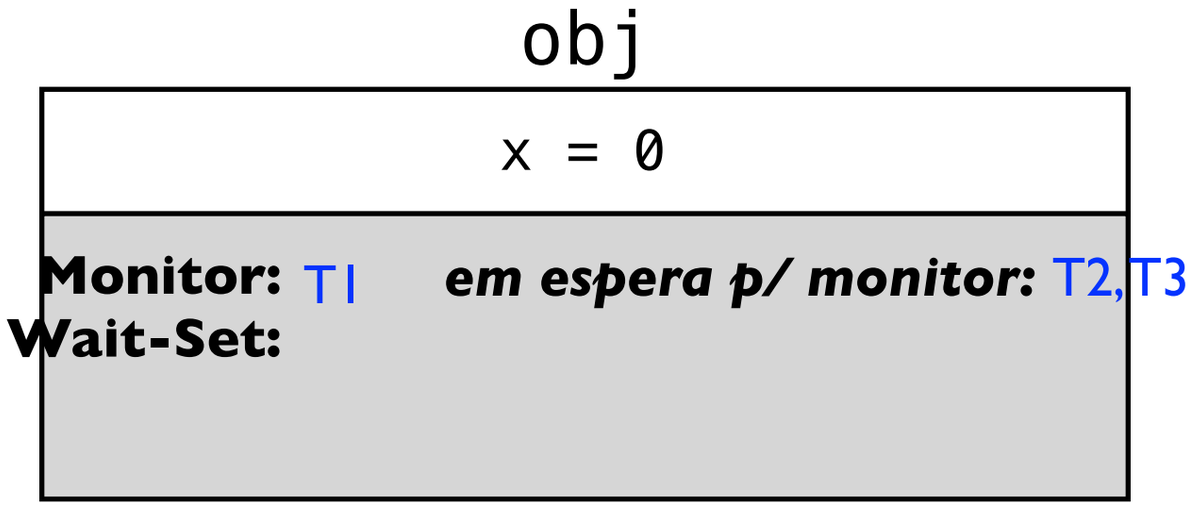
```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```



- Exemplo:

T1

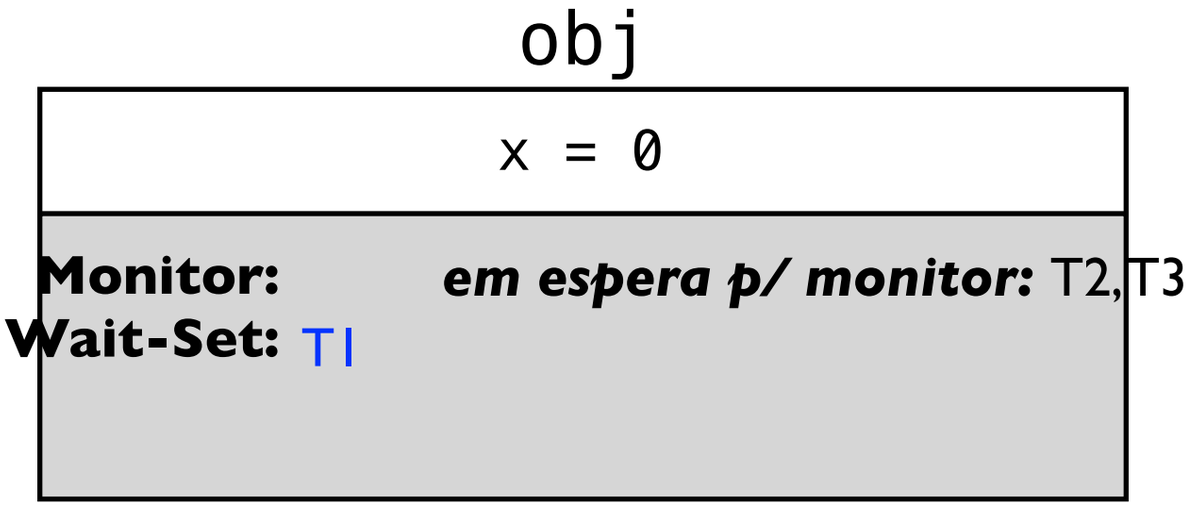
```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```



- Exemplo:

T1

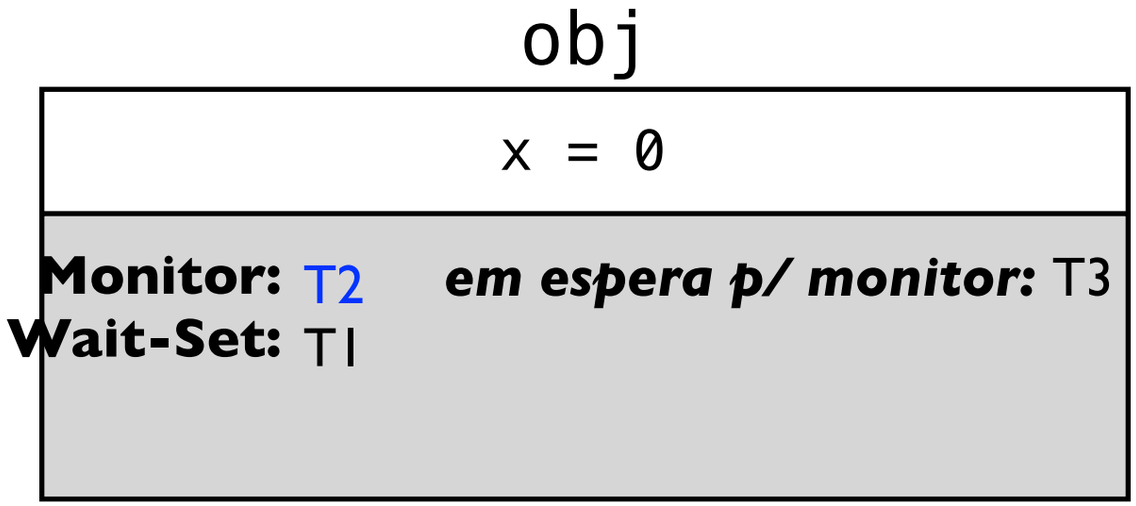
```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```



- Exemplo:

T1

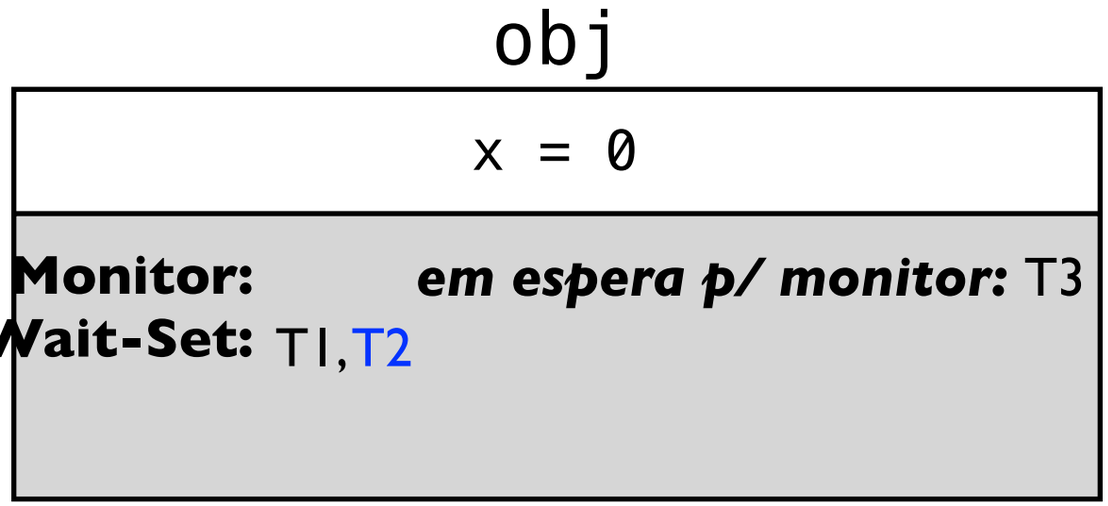
```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```



- Exemplo:

T1

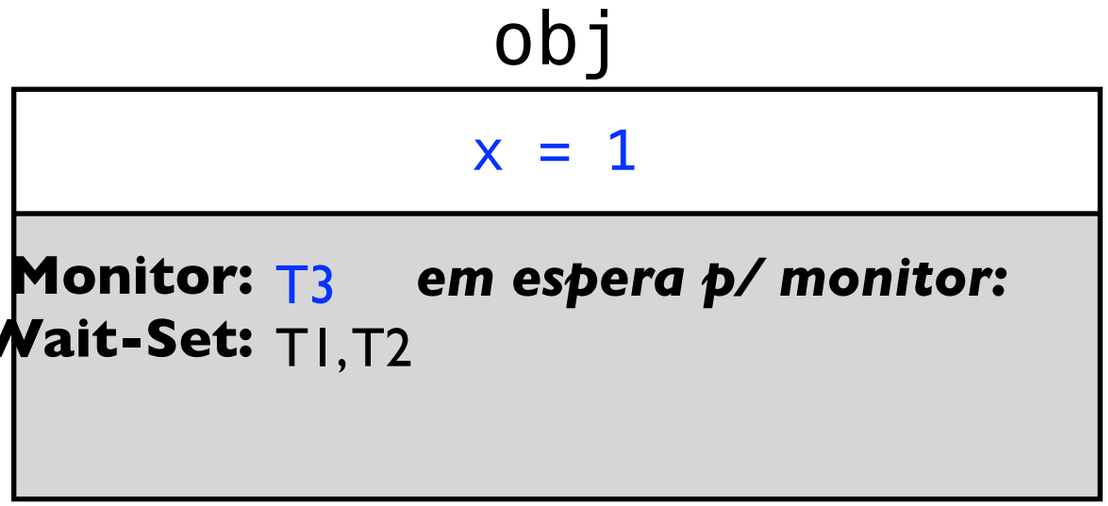
```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
     obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
     obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```



- Exemplo:

T1

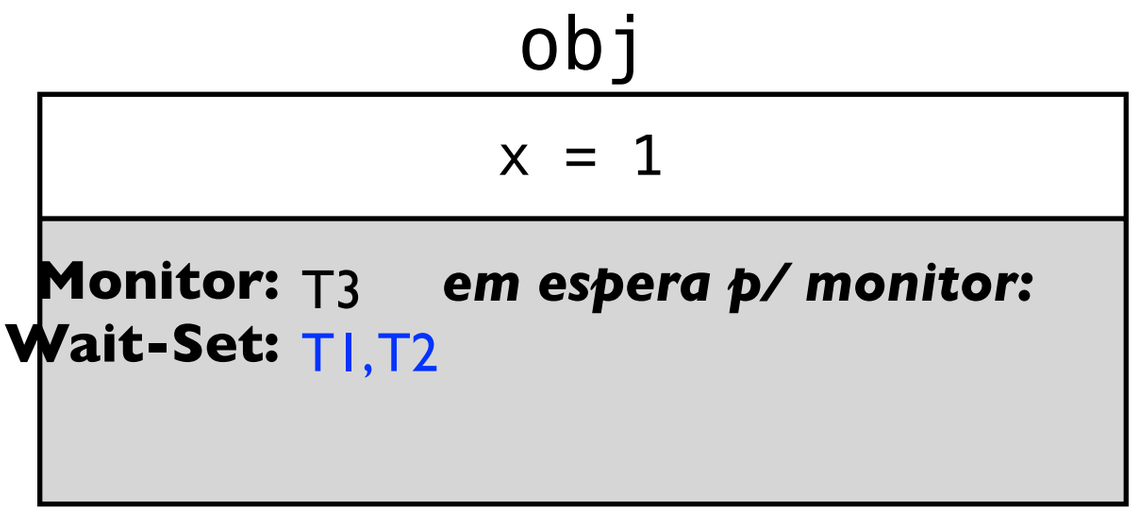
```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```



- Exemplo:

T1

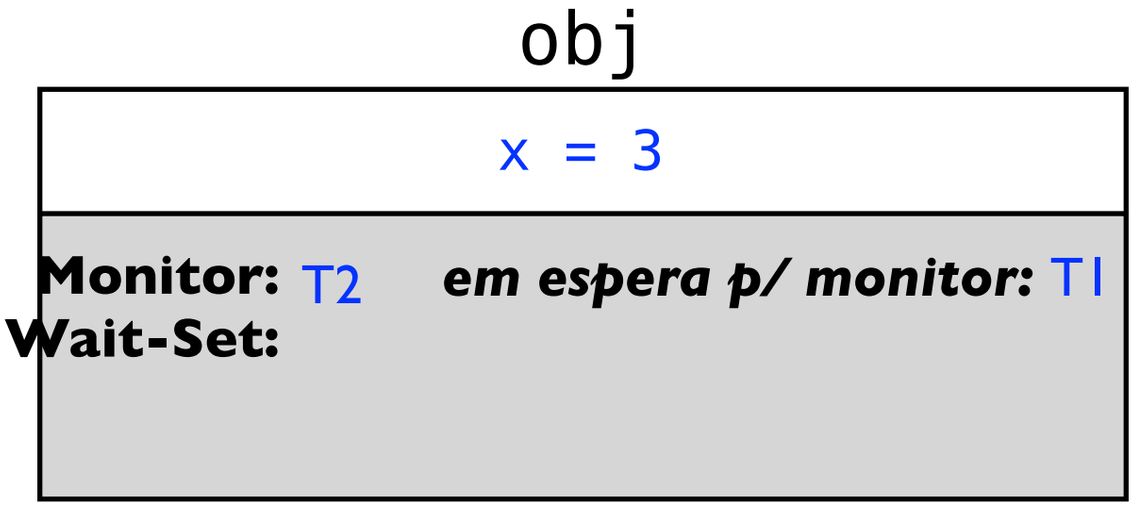
```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```



- Exemplo:

T1

```
metodo1(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=1;
   }
}
```

T2

```
metodo2(){
   synchronized(obj) {
     while(obj.x == 0){
       obj.wait();
     }
      obj.x+=2;
   }
}
```

T3

```
metodo3(){
   synchronized(obj) {
     obj.x = 1;
     obj.notifyAll();
   }
}
```

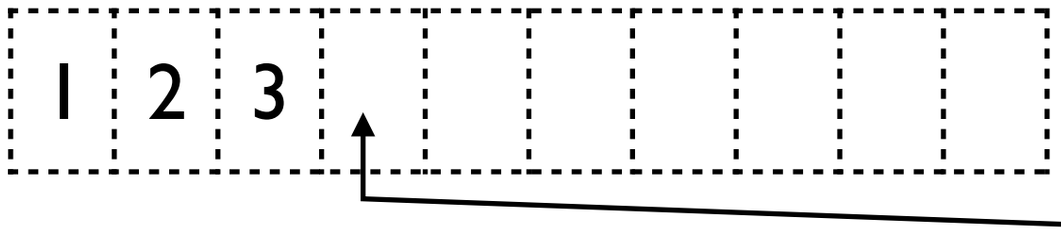
obj

x = 4
Monitor: T1 ***em espera p/ monitor:***   **Wait-Set:**

Exercícios

1) Implemente uma classe BoundedBuffer que ofereça as operações void put(int v) e int get() sobre um array cujo tamanho é definido no momento da construção de uma instância. O método put deverá bloquear enquanto o array estiver cheio e o método get deverá bloquear enquanto o array estiver vazio. Os métodos oferecidos podem estar sujeitas a invocações de threads concorrentes sobre uma instância partilhada. A classe BoundedBuffer deverá garantir a correcta execução em cenário multi-thread.

Exercícios



BoundedBuffer
(LIFO - Last In First Out)

```
private int[] values
int poswrite;
```

```
void put(int v);
int get();
```

Produtor
implements Runnable

Consumidor
implements Runnable

Condições a respeitar:

- **put(v):** Bloquear enquanto o array estiver cheio;
- **get():** Bloquear enquanto o array estiver vazio.

Usando *wait* e *notify/notifyAll* e exclusão mútua;

Teste:

- BoundedBuffer com 10 posições
- Consumidor invoca get 20x
- Produtor invoca put 20x

Exercícios

2) Considere um cenário produtor/consumidor sobre o BoundedBuffer do exercício anterior, com P produtores e C consumidores, com um número total de threads C + P = N e tempos de produção e consumo Tp e Tc. Obtenha experimentalmente o número óptimo de threads de cada tipo a utilizar para maximizar o débito.

Exercício 2

Cenário de Teste:

Tc = 0.5seg ← *tempo de espera entre cada get (tempo de consumo)*

Tp = 1seg ← *tempo de espera entre cada put (tempo de produção)*

totalOps = 100 ← *no total, produzir 100 items e consumir 100 items*

N = C + P = 10 ← *total de threads*

C = ?

P = ?

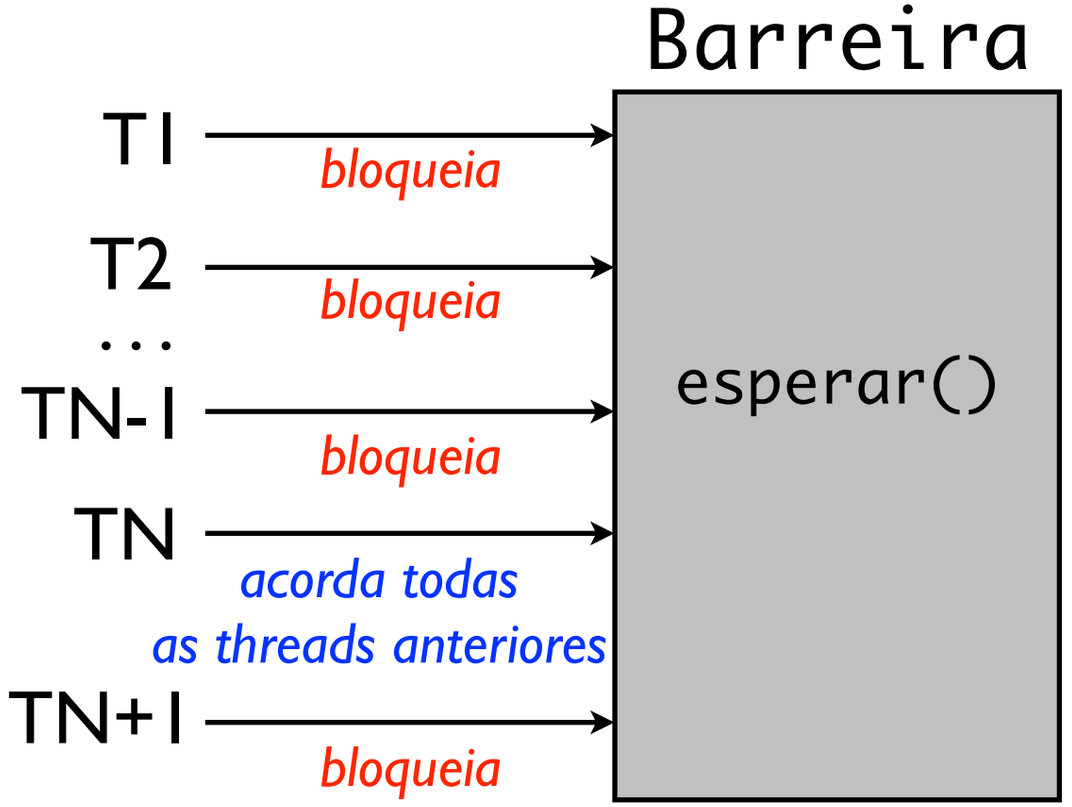
Objectivo:

Descobrir **C** e **P** de forma a maximizar o débito (ops/seg):

$$\text{débito} = \frac{\text{totalOps}}{(\text{tempoFim} - \text{tempoInicio})}$$

Exercícios

3) Implemente uma classe Barreira que ofereça um método esperar() cujo objectivo é garantir que cada thread que o invoque se bloqueie até que o número de threads nesta situação tenha atingido o valor N passado ao construtor de uma sua instância. Um objecto Barreira deverá poder ser reutilizado em sucessivas invocações de esperar().



Pontos-Chave

- permitem que threads suspendam/retomem a sua execução **dentro de secções críticas**, de acordo com uma dada condição
- **estão associadas a um monitor/lock**; é necessário obter o lock antes de utilizar a variável de condição
- a condição deve ser testada sempre com **while**, para impedir inesperados causados por interrupções
- métodos: **wait()**, notify(), **notifyAll()**