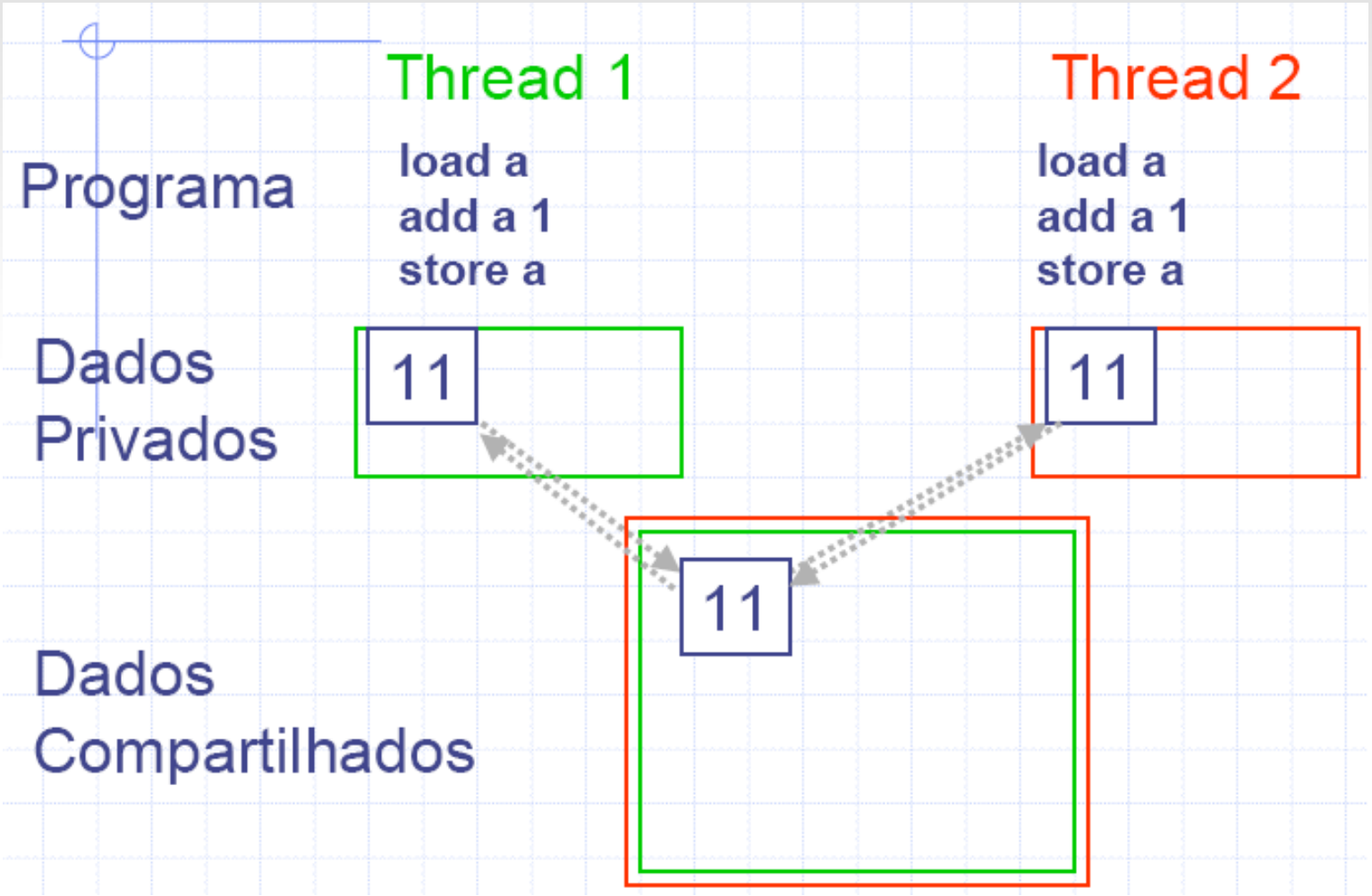
OpenMP avançado

Sincronização

- Pode ser necessário assegurar que as ações em variáveis compartilhadas ocorram da maneira correta
  - Exemplos práticos:
    - a *thread* 1 deve escrever na variável *A* antes da *thread* 2 faça a sua leitura
    - a *thread* 1 deve ler a variável *A* antes que a *thread* 2 faça sua escrita.
  - Note que as atualizações para variáveis compartilhadas (p.ex. $a = a + 1$) não são atômicas!
    - Se duas *threads* tentarem fazer isto ao mesmo tempo, uma das atualizações pode ser perdida.

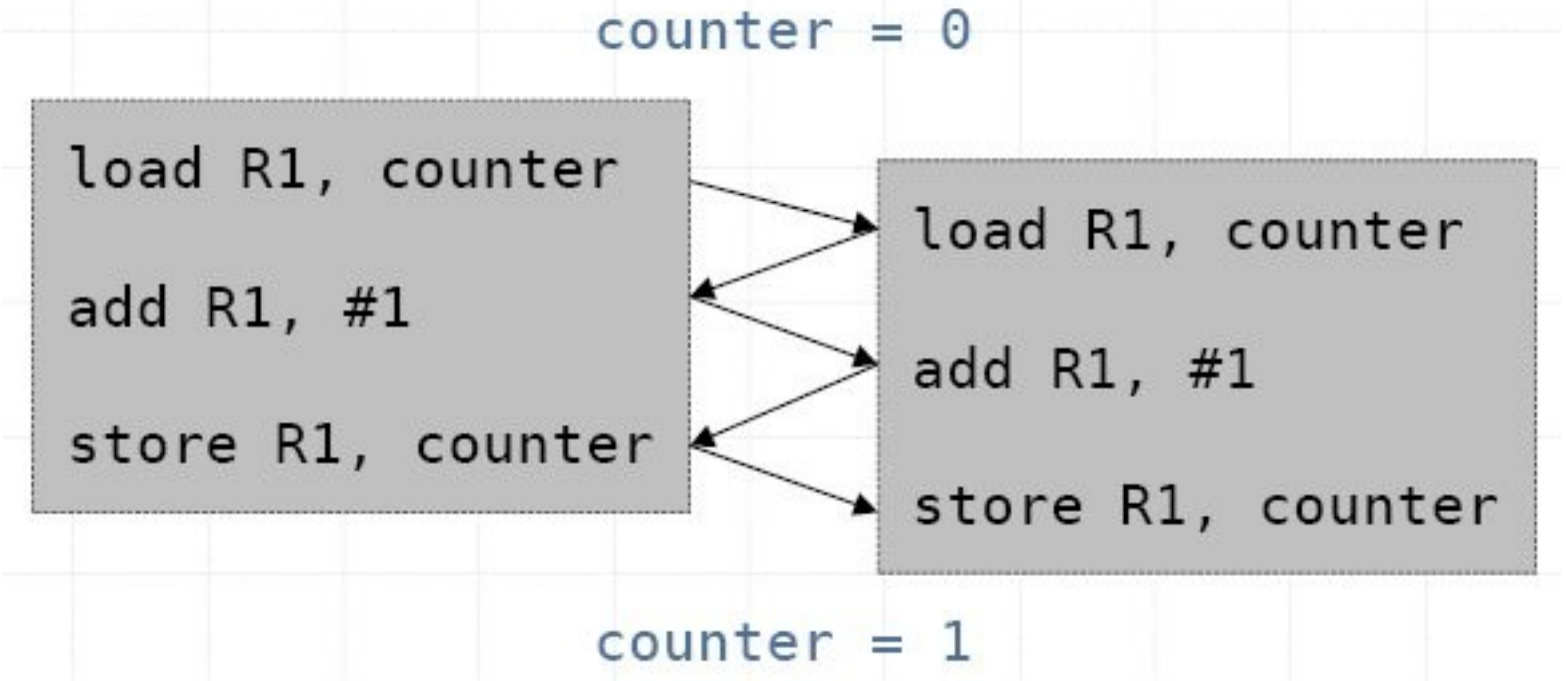
Exemplo de Sincronização



Problemas em operações não atômicas

- Supondo uma dada variável compartilhada `counter` e uma operação do tipo:
  `counter++`

- Supondo-se 2 threads pode-se ter a seguinte situação:



Solução do problema

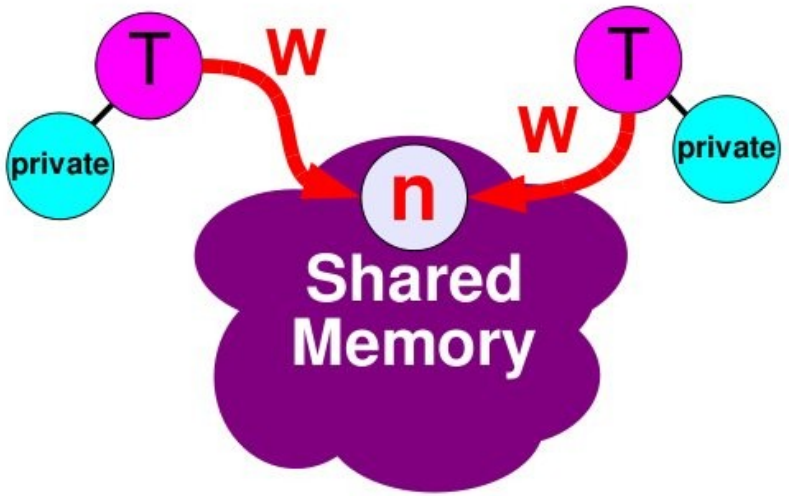
- Resolver cada ação envolvendo uma variavel compartilhada (como, por exemplo, counter++) **atomicamente**
  - Isto é, sem sem interrompida para intercalação e/ou sem sofrer ação de outro processo em paralelo
- Criação de uma **Seção Crítica**
  - Parte do programa que deve ser executada atomicamente
  - Normalmente implementada como um procedimento conhecido por **Exclusão Mútua (Mutual Exclusion)**: garante que apenas uma thread faz uma dada operação por vez.
    - Recursos da linguagem de prog.: Semáforos, monitores,

Condição de corrida (*Data racing*)

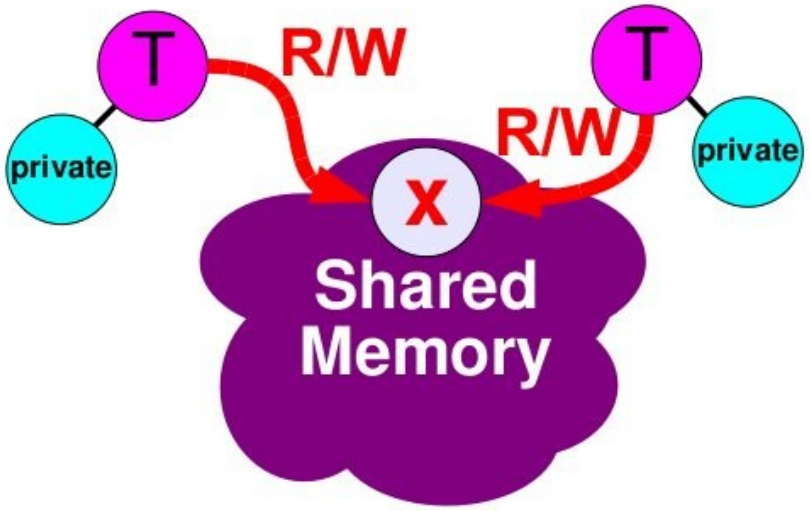
- Duas ou mais diferentes threads tentam acessar a mesma área comum de memória:
  - De forma assíncrona
  - Sem a implementação de nenhuma trava ou barreira
  - Implementam operações do tipo *Load/Store*
- Significa que uma atualização em uma área de memória comum não está bem protegida

Exemplos de condições de corrida

```
#pragma omp parallel shared(n)
  {n = omp_get_thread_num();}
```




```
#pragma omp parallel shared(x)
  {x = x + 1;}
```



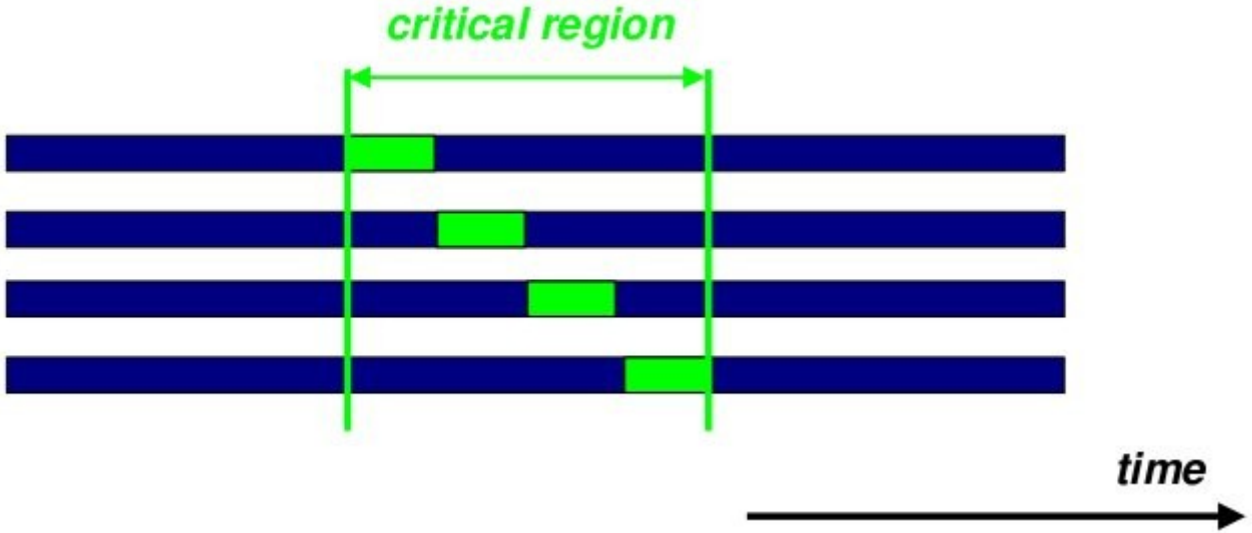
Sincronização explicita

- Pode ser necessário sincronizar ações em variáveis compartilhadas. Exemplos:
  - Assegurar a ordenação correta de leituras e escritas
  - Proteger a atualização de variáveis compartilhadas (não atômicas por padrão)

- Importante lembrar que existe uma barreira implícita no final das diretivas *DO/FOR*, *SECTIONS* e *SINGLE*

Seções Críticas em OpenMP

- Bloco de código que só pode ser executado por uma *thread* por vez
  - Pode ser utilizado para proteger a atualização de variáveis compartilhadas e evitar a condição de corrida
  - A diretiva *CRITICAL* permite que as seções críticas recebam nomes
    - Se uma *thread* está em uma seção crítica com um dado nome, nenhuma outra *thread* pode estar em uma seção crítica com o mesmo nome (embora elas possam estar em seções críticas com outros nomes)
    - Se o nome é omitido, um nome nulo é assumido



Exemplo básico para seção crítica em OpenMP

```
#include <omp.h>
main()
{
int x;
x = 0;
#pragma omp parallel shared(x)
  {
  #pragma omp critical
  x = x + 1;
  }  /* end of parallel section */
}
```

Seção Crítica – Exemplo 2

- Colocando e retirando dado de uma pilha

```
#pragma omp parallel shared(stack), private(inext,inew)
...
#pragma omp critical (stackprot)
{
inext = getnext(stack};
}
work(inext,inew);
#pragma omp critical (stackprot)
if (inew > 0) putnew(inew,stack);
}
...
```

Diretiva *ATOMIC*

- Usada para proteger uma atualização única para uma variável compartilhada
- Aplica-se apenas a uma única sentença (forma ”light” de se definir uma seção crítica)
- Sintaxe em C/C++:

```
#pragma omp atomic
        statement
```

  - Onde *statement* deve ter uma das seguintes formas:
  - x binop = expr, x++, ++x, x--, or –x
    - and binop is one of +, *, -, /, &, ^, <<, or >>

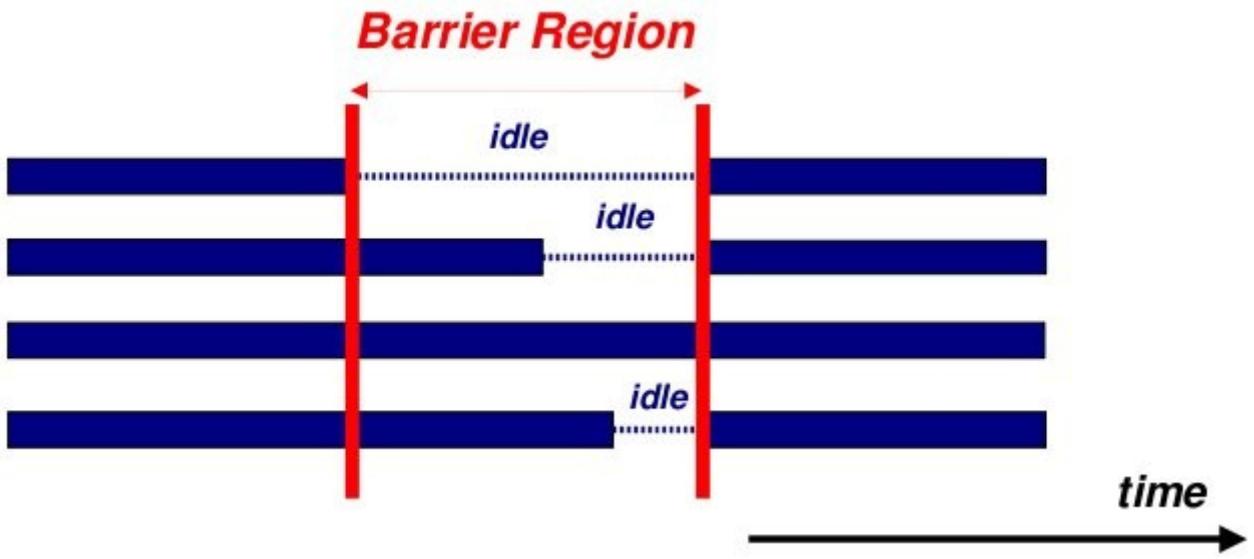
Diretiva ATOMIC - Exemplo

- Computar o grau de cada vértice em um grafo:

```
#pragma omp parallel for
for (j=0; j<nedges; j++) {
#pragma omp atomic
    degree[edge[j].vertex1]++;
#pragma omp atomic
    degree[edge[j].vertex2]++; }
```

Sincronização por Barreira
Diretiva *BARRIER*

- Nenhuma *thread* pode prosseguir além de uma barreira até que todas as outras *threads* cheguem até ela
  - Se alguma *thread* não encontrar a barreira: *DEAD LOCK!!!*
  - Usar com moderação pois pode causar atrasos na computação!



Código Exemplo:

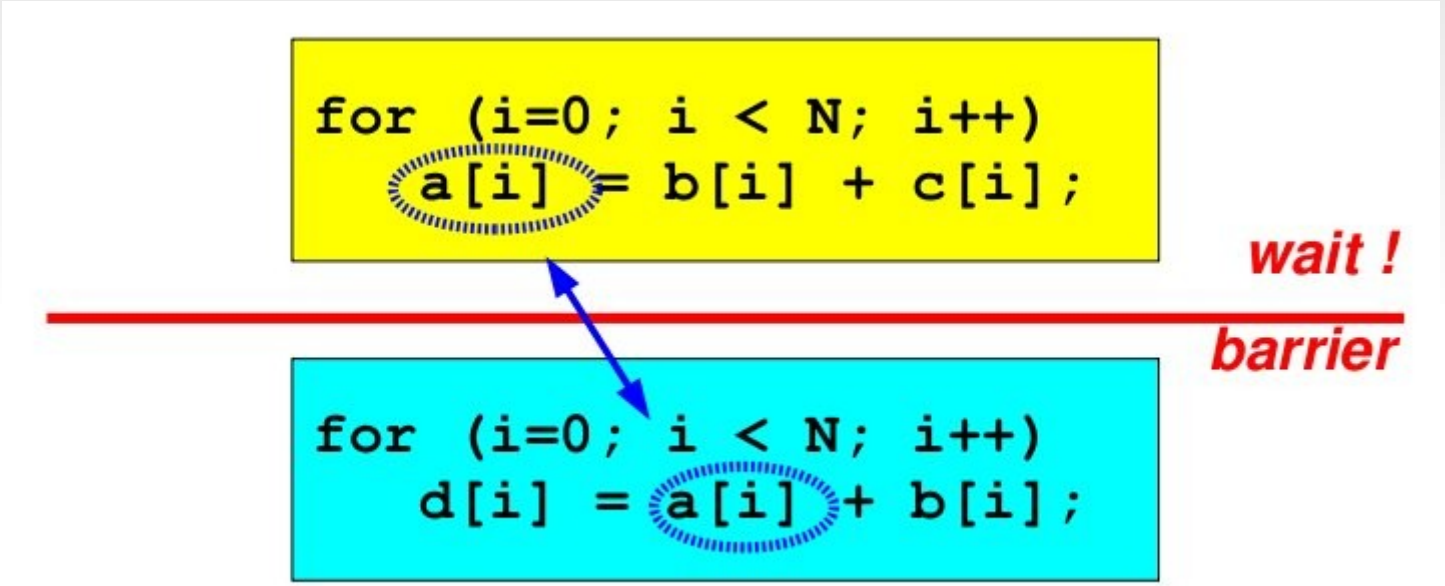
```
#pragma omp parallel private(i,myid,neighb)
{
myid = omp_get_thread_num();
neighb = myid – 1;

if (myid == 0) neighb = omp_get_num_threads()-
  1;
...

a[myid] = a[myid]*3.5;

#pragma omp barrier
b[myid] = a[neighb] + c
...
}
```

Situação a se usar barreira



A cláusula *NOWAIT*

- De forma a minimizar os tempos gastos com sincronização em OpenMP as diretivas/pragmas permitem o uso de uma cláusula opcional: *nowait*
  - Se esta cláusula estiver presente as threads não sincronizam (como por padrão) ao final de uma construção paralela como, por exemplo, ”for”

```
#pragma omp for nowait
{
        :
}
```

```
!$omp do
        :
        :
!$omp end do nowait
```

A more elaborate example

```
#pragma omp parallel if (n>limit) default(none) \
        shared(n,a,b,c,x,y,z) private(f,i,scale)
{
    f = 1.0;
#pragma omp for nowait

    for (i=0; i<n; i++)
       z[i] = x[i] + y[i];

#pragma omp for nowait

    for (i=0; i<n; i++)
       a[i] = b[i] + c[i];

          ....
#pragma omp barrier
    scale = sum(a,0,n) + sum(z,0,n) + f;
          ....
} /*-- End of parallel region --*/
```

Statement is executed by all threads

parallel loop
(work is distributed)

parallel loop
(work is distributed)

parallel region

synchronization

Statement is executed by all threads

Cláusula IF

- Somente ativa uma seção paralela se o valor de uma dada expressão lógica for verdadeira
  - Se for falsa executa serial

```
#pragma omp parallel if (n > some_threshold) \
        shared(n,x,y) private(i)
  {
    #pragma omp for
     for (i=0; i<n; i++)
        x[i] += y[i];
  } /*-- End of parallel region --*/
```

Balanceamento de carga em laços paralelo

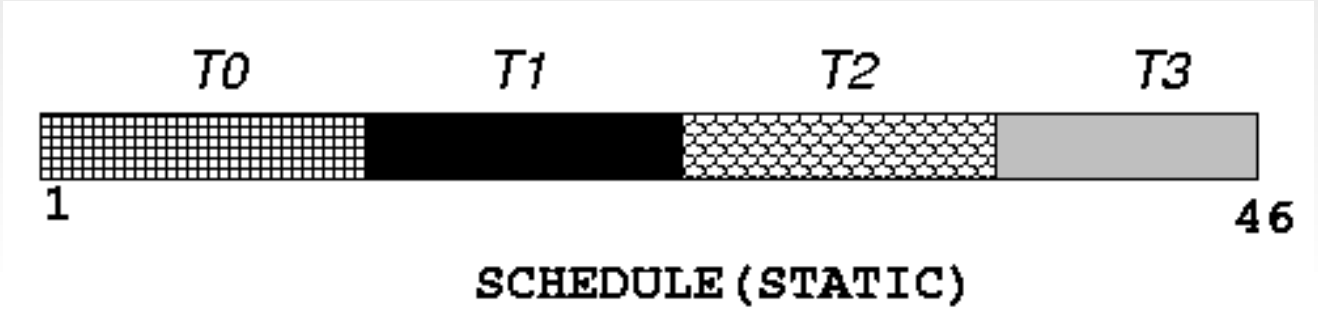
- Em algumas operações sobre laços podem existir desbalanceamento de carga entre os processadores
  - Reduz ganhos de desempenho pois os processos mais rápidos deverão esperar (ociosos) pelo mais lento no ponto de sincronização
  - Exemplo:
    - Laço que chama uma função com argumento dado pelo índice que controla o laço
      - A função pode gastar mais ou menos CPU a cada iteração

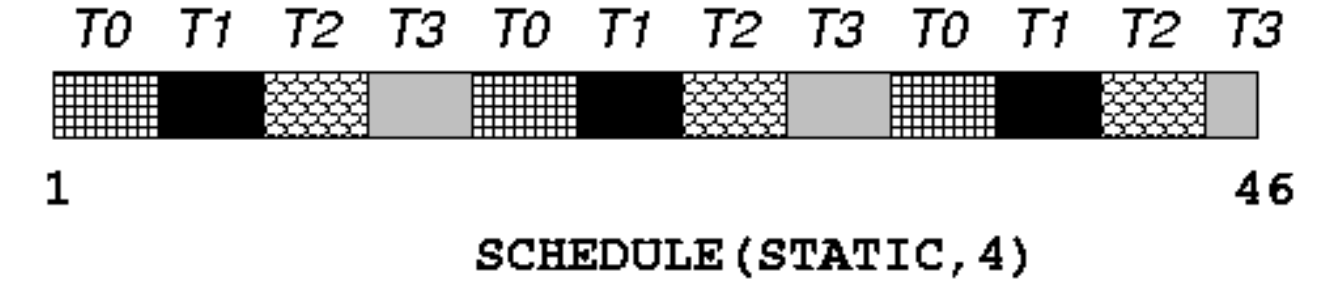
```
#pragma omp parallel for
for (int i=0; i<N; i++) {
    a[i] = checar_se_e_primo(i); }
```

Escalonamento
Cláusula *SCHEDULE*

- A cláusula *SCHEDULE* permite uma variedade de opções por especificar quais iterações dos laços são executadas por quais *threads*.
  - Sintaxe em C/C++:
    - `schedule (kind[, chunksize])`
  - onde *kind* pode ser *STATIC, DYNAMIC, GUIDED* ou *RUNTIME*
  - e *chunksize* é uma expressão inteira com valor positivo, a qual divide o espaço de iterações em pedaços, cada um com *chunksize* iterações
    - os pedaços são atribuídos ciclicamente a cada *thread* (escalonamento *block cyclic*)
  - Ex.: `#pragma for schedule(DYNAMIC,4)`

Escalonamento *SCHEDULE STATIC*





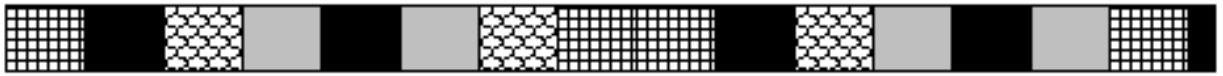
Escalonamento DYNAMIC

- Divisão do espaço de iteração em pedaços de tamanho *chunksize*
  - Atribuição para as *threads* com uma política *firstcome-first-served*
  - i.e. se uma *thread* terminou um pedaço, ela recebe o próximo pedaço na lista
- Quando nenhum valor de *chunksize* é especificado, o valor padrão é 1

Escalonamento *GUIDED*

- Similar ao *DYNAMIC*, mas os pedaços iniciam grandes e se tornam pequenos exponencialmente
  - O tamanho do próximo pedaço é (a grosso modo) o número de iterações restantes dividido pelo número de *threads*
  - O valor *chunksize* especifica o tamanho mínimo dos pedaços
- Quando nenhum valor de *chunksize* é especificado, o padrão é 1

Escalonamentos *DYNAMIC* e *GUIDED*



1 SCHEDULE(DYNAMIC,3) 46

1 SCHEDULE(GUIDED,3) 46