OpenMP

dado
- **shared**: compartilhado pelas threads (uma única instância)
  - ↳ ler e escrever simultaneamente ↳ modificações gerais
- **private**: cada thread tem uma cópia do dado
  - ↳ modificações locais

(*) variáveis são shared por padrão
iteradores de laços são private

atributos

(I) **private**
↳ local de cada thread
Obs: NÃO SÃO INICIALIZADAS!

(II) **shared**
↳ global.

(III) **firstprivate**
↳ local de cada thread
Obs: Inicializa com o valor antes do pragma

(IV) **reduction**
↳ local para cada thread
↳ ao final uma operação é aplicada nas privadas e na compartilhada
O Resultado é escrito na compartilhada

---

reduction

operadores: +, *, -, &, |, ^, &&, ||

- as variáveis locais são inicializadas com 0.
- as variáveis locais são inicializadas com 1.

ex:

```
for (i=0; i<20; i++) {
    sum += a[i];
}
```

↓

```
int sum = 0;
#pragma omp for reduction (+:sum)
for (i=0; i<20; i++){
    sum += a[i];
}
```

FOR: sempre depois de `#pragma omp parallel`

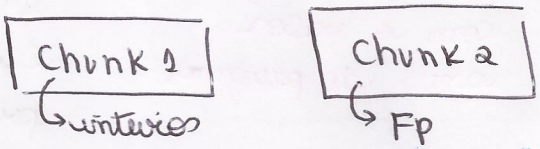
→ SCHEDULE
- **tipo:** static, dynamic e guided
- **chunk:** tamanho dos blocos dos loops

→ **static:** cada thread calcula o seu chunk sob a própria responsabilidade

padrão: $\frac{num\_iterações}{num\_threads}$

Bom: sem comunicação entre threads → não tem disputa por atividade ↓ sem bloqueios do controlador

Ruim: load unbalance



tamanho → Chunk 1 = Chunk 2

↳ vai demorar +!

↳ Não garante uma igualdade no tempo de atividade de t_1 e t_2.

→ **dynamic:**
- fila de chunks de tamanho fixo
- padrão: 1.

→ 1 Produtor vs n consumidores

Bom: É difícil ter uma thread parada

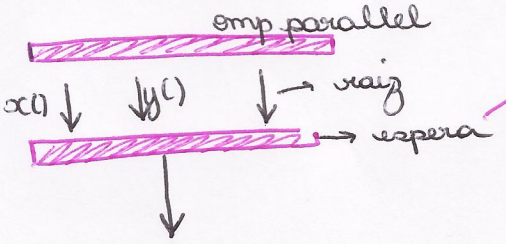
Ruim: Muita comunicação se o chunck for pequeno

↳ Por quê? As threads acabarão os chunks rapidamente e ficarão disputando na fila → perde tempo.

→ **guided:**
- fila de chunks → inicialmente grande e tenta chegar no definido ou o padrão (1).

→ **Section:** cada section é executada por uma única thread.

seções: x(), y() :



se tiver nowait, essa barreira não existe

Single: só uma thread executa

→ Critical: essa região será executada por somente uma thread por vez!

```
a = 0;
#pragma omp parallel
{
#pragma omp critical
 a += 4;
}
```

→ com 4 threads: 16.

→ Barrier: Sincroniza todas as threads em uma barreira.