Começando a Pensar em Paralelo.

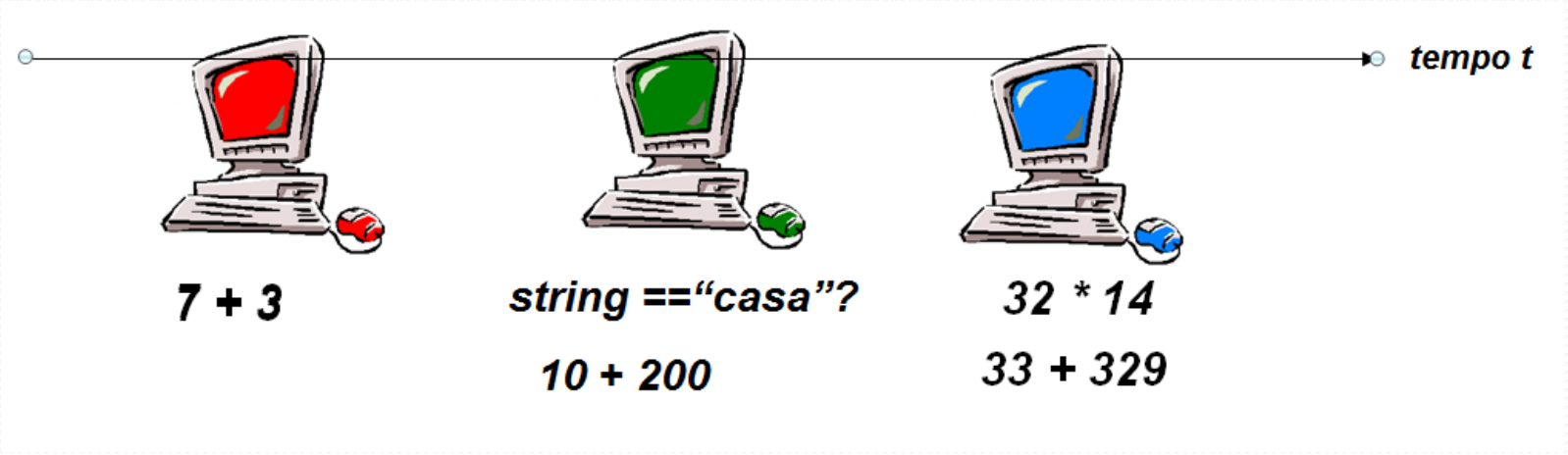
Universidade Estadual de Mato Grosso do Sul – UEMS
Curso de Ciência da Computação
Disciplina de Algoritmos Paralelos e Distribuídos

Pensando em Paralelo

- Pensar em paralelo é uma tarefa que exige disciplina e praticidade do desenvolvedor;
- As ferramentas mais adequadas irão depender de uma série de fatores como:
  - Balaceamento de carga;
  - Tamanho do problema;
  - Tempo de desenvolvimento requerido;
- Lembrar sempre que há a necessidade de se conhecer o hardware.

Computação Concorrente

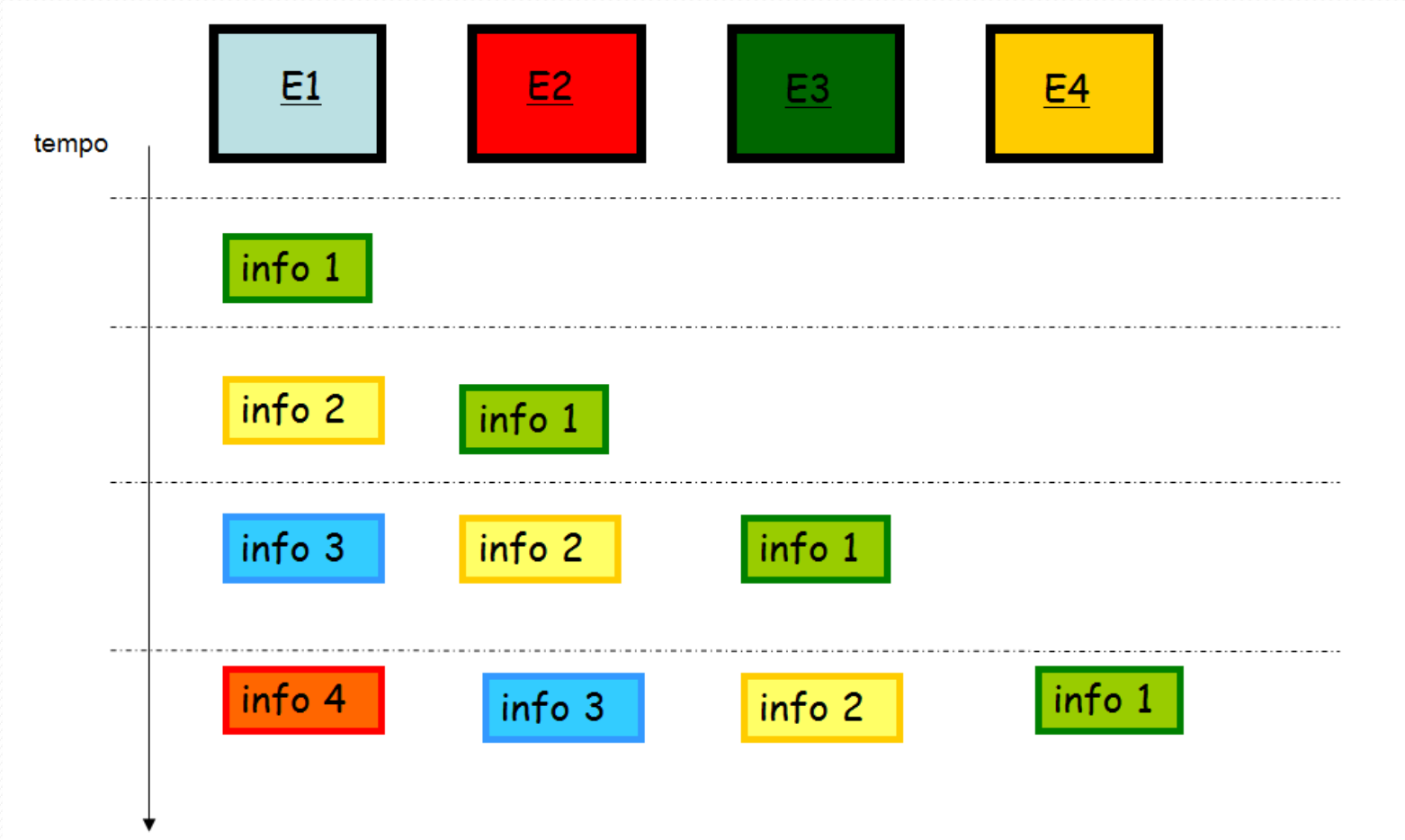
- Há duas formas de se explorar a concorrência em computação:
  - Paralelismo de controle e paralelismo de dados;



Paralelismo de Controle

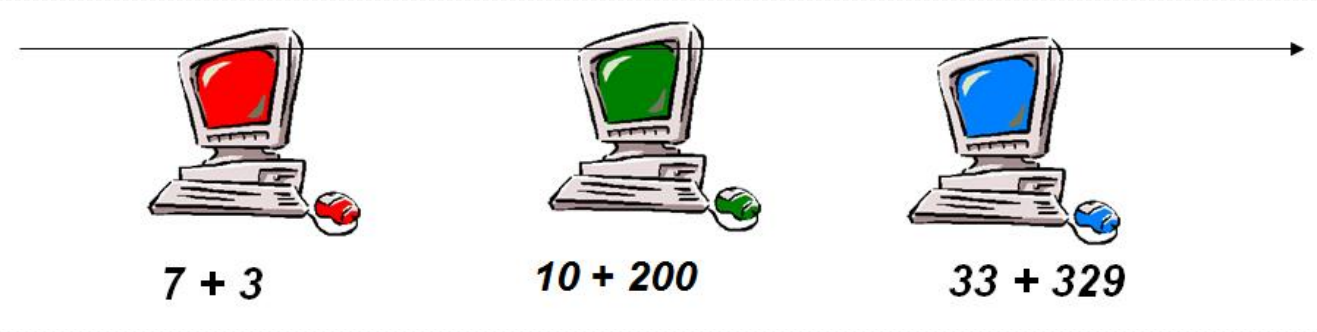
- Aplicando operações diferentes a diferentes dados simultaneamente.
  - Fluxo de dados entre processos pode ser arbitrariamente complexo.
- Exemplo:
  - *Pipelining:* Cada estágio trabalha em velocidade plena sobre uma parte particular da computação. A saída de um estágio é a entrada do estágio seguinte.

Paralelismo de Controle



Paralelismo de Dados

- Uso de vários processadores para executar a mesma operação ao mesmo tempo sobre elementos distintos de dados;
- Um aumento de *k* vezes no número de unidades funcionais levaria a um aumento de *k* vezes a vazão do sistema;



Aplicações

- Aplicações ou programas podem ser executados mais rapidamente, considerando duas formas de paralelismo:
  - Tratar o programa sequencial como uma série de tarefas;
  - Cada tarefa = uma instrução ou bloco de instruções;

Ou

  - Especificar um programa paralelo, que resolve o problema por meio da especificação de tarefas ou processos concorrentes;

Explorando o Paralelismo

- Particionamento:
  - Identificar em um programa as tarefas que possam ser executadas em paralelo (simultaneamente em mais de um processador).
    - Caso extremo: Cada linha do programa correspondendo a uma tarefa.

Obs.: Um bom desempenho só é alcançado quando um número máximo de comandos são executados simultaneamente.

- É preciso considerar possíveis dependências de dados;
  - Problema: pode ocorrer de quase todos os comandos serem dependentes;

Exemplo: prog. Sequencial -> paralelismo de instruções.

```
program nothing(){
   input (A,B,C);
   if  A>B then {
     C=A-B;
            output (C);
   } else {
     C = B-A;
            output (A,B,C)
   }
   A=0; B=1;
   for i=1 to 3 {
            input(C);
            A=A+C;
            B=B*C;
   }
   output (A,B,C);
}
```

Exemplo: prog. Sequencial -> paralelismo de instruções.

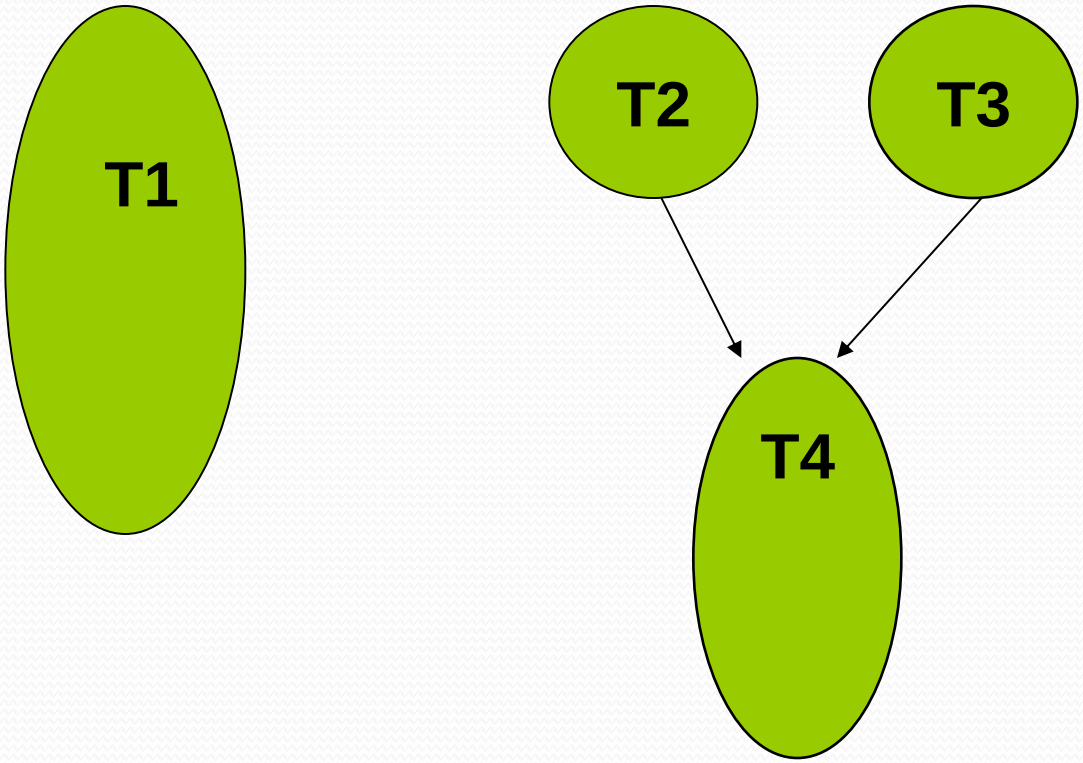
```
Tarefa T1
input (A,B,C);
if  A>B then{
   C=A-B;
     output (C);
}else{
   C = B-A;
        output (A,B,C)
}
```

```
Tarefa T2
A = 0;
```

```
Tarefa T3
B=1;
```

```
Tarefa T4
for i=1 to 3 {
     input(C);
     A=A+C;
     B=B*C;
}
output (A,B,C)
```

Dependência



Exemplo: Soma de n números.

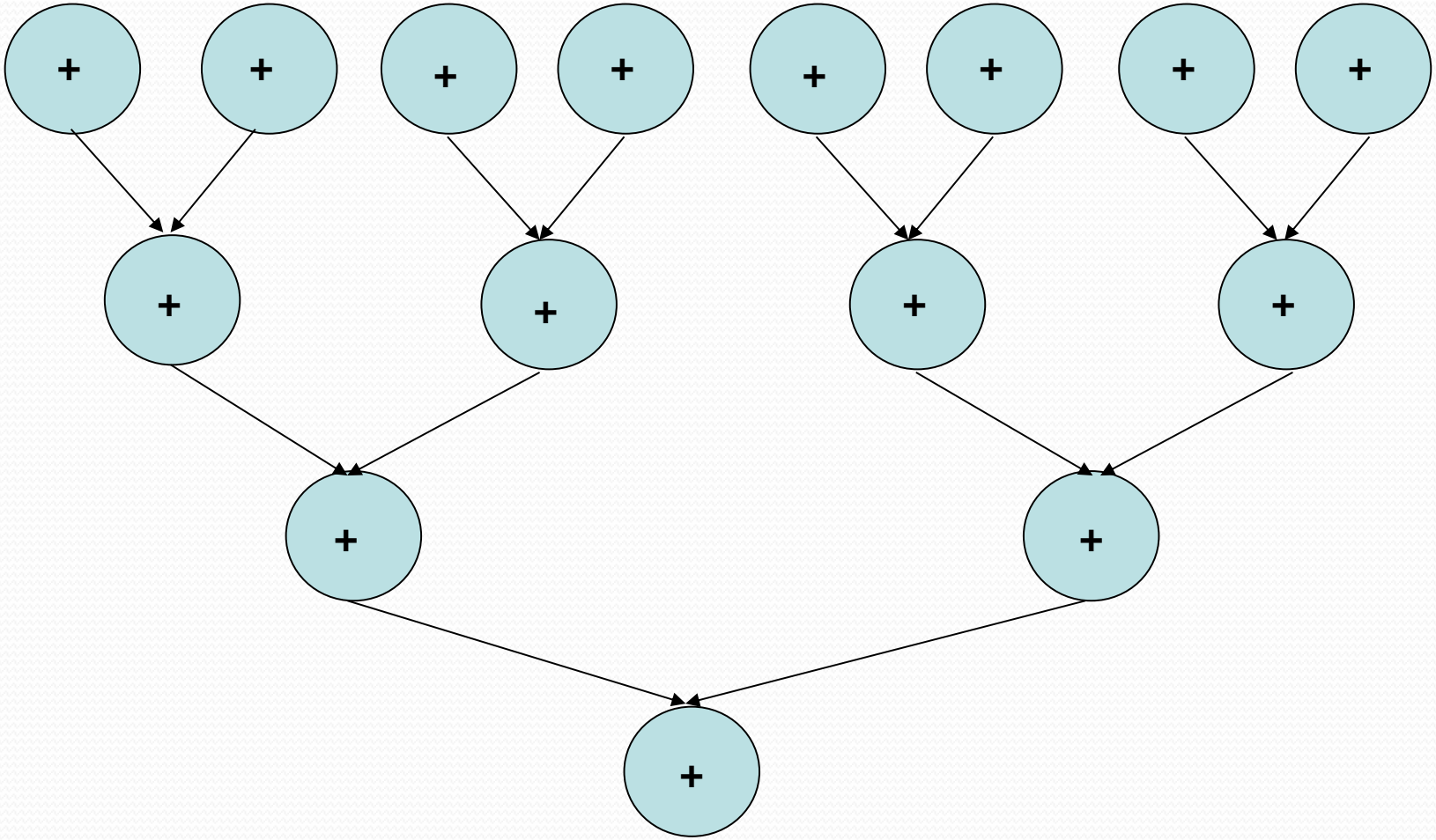
Problema: somar *n* números quaisquer

Programa seqüencial:

```
read (n, vetor);
soma = 0;
for (i = 0; i < n; i++)
     soma = soma + vetor[i];
```

Como paralelizar o problema?

Modelando a soma paralela de n números.

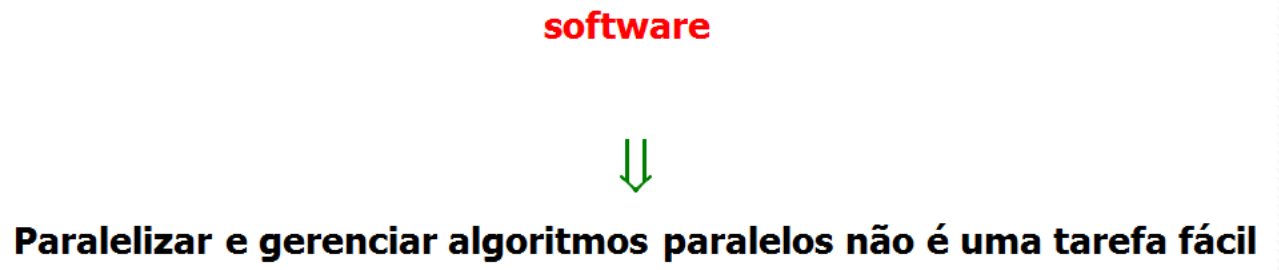


Terminologia

- Um algoritmo é escalável se o nível de paralelismo aumenta no mínimo linearmente com o tamanho do problema.
- Uma arquitetura é dita escalável se continua a alcançar o mesmo desempenho por processador, mesmo para problemas maiores, com o aumento de processadores.
  - Se aplicável, o usuário pode resolver problemas maiores no mesmo intervalo de tempo através da compra de uma máquina paralela com maiores dimensões.
- Algoritmos paralelos de dados são mais escaláveis do que algoritmos com paralelismo de controle.
  - O nível de paralelismo de controle é geralmente constante, independente do tamanho do problema, enquanto o nível de paralelismo de dados é uma função crescente do tamanho do problema.

Convergindo entre Computação Paralela e Distribuída

- No início existia uma grande dificuldade de manter máquinas paralelas a frente aos projetos de chip-único.
  - Computadores paralelos se tornaram mais difíceis de se construir e se usar do que se esperava.
  - Motivo desta dificuldade:



Modelos de Programação Paralela

- São os modelos formais que nos ajudam a demonstrar que os algoritmos são ótimos ou a obter “ótimos” resultados.
- Sua importância também está direcionada com a possibilidade de relacionar a complexidade do algoritmo sequencial com o algoritmo paralelo.
- Por meio da remoção dos detalhes da codificação, como por exemplo a comunicação e a sincronização, o programador pode focar-se nas características estruturais do problema e do processamento concorrente.

Aspectos Relacionados pelo Modelo

- O modelo deve ser capaz de oferecer facilidades tais que permita:
  - Decompor o programa em tarefas paralelas;
  - Mapear as tarefas nos processadores físicos:
    - Custo de comunicação;
    - Heterogeneidade dos processadores;
  - Sincronização entre tarefas: é preciso ter conhecimento do estado global da estrutura de execução do programa (quando é necessário sincronizar?)

Modelos de computação Paralela

- Modelo de computação sequencial: Von Neumann;
- Plataforma base para usuários e projetistas:
  - Complexidade de tempo do pior caso: Tempo máximo que o algoritmo pode levar para executar qualquer entrada com *n* elementos;
  - Complexidade de tempo esperado: Complexidade média;
  - Critério de custo uniforme: Qualquer instrução PRAM leva uma unidade de tempo para ser executada.

Modelos de computação Paralela

- Computação Paralela:
  - O desempenho do programa paralelo depende de certos fatores que são dependentes da máquina:
    - O grau de concorrência;
    - Escalonamento e alocação de processadores;
    - Comunicação e sincronização;
- Medidas de complexidade:
  - Tempo de execução;
  - Número de processadores;
  - Custo -> Tempo x processadores;

Modelo PRAM Ideal

- Conjunto de *p* processadores operando sincronamente sob o controle de um único relógio, compartilhando o espaço global de memória;
- Algoritmos desenvolvidos para este modelo são do tipo SIMD:
  - Todos os processadores executam o mesmo conjunto de instruções, e ainda a cada unidade de tempo, todos os processadores estão executando a mesma instrução mas usando dados diferentes;

Passos do Algoritmo PRAM

- Fase de Leitura: os processadores acessam simultaneamente locais de memória para a leitura. Cada processador acessa no máximo uma posição de memória e armazena o dado lido em sua memória local;
- Fase de Computação: os processadores executam operações aritméticas básicas com seus dados locais;
- Fase de Gravação: os processadores acessam simultaneamente locais de memória global para a escrita. Cada processador acessa no máximo uma posição de memória e grava um certo dado que está armazenado localmente.

Padrões de Acesso no Modelo PRAM

- *Exclusive Read* (ER): vários processadores não podem ler ao mesmo tempo no mesmo local de memória.
- *Exclusive Write* (EW): vários processadores não podem escrever ao mesmo tempo no mesmo local de memória.
- *Concurrent Read* (CR): vários processadores podem ler ao mesmo tempo no mesmo local de memória.
- *Concurrent Write* (CW): vários processadores podem escrever ao mesmo tempo no mesmo local.
- Combinações: EREW, CREW, ERCW, CRCW.

Prioridade do CRCW

- Para resolver os conflitos de vários processadores tentarem escrever ao mesmo tempo no mesmo local de memória global:
  - Comum: Vários processadores concorrem na escrita no mesmo local de memória global durante o mesmo instante de relógio – todos devem escrever o mesmo valor;
  - Arbitrário: dentre os vários processadores, um é selecionado arbitrariamente, e seu valor armazenado no local de memória disputado;
  - Prioridade: dentre os vários processadores, aquele com o menor índice é escolhido para escrever o seu valor na memória;
  - Fraco: todos os processadores escrevem o mesmo valor 0 ou 1;
  - Forte: os valores escritos simultaneamente podem ser diferentes. Ficará armazenado na memória o maior valor escrito.