Sistemas Concorrentes e Distribídos

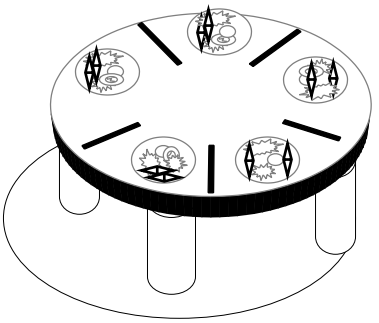
Problemas e Verificação

Verificação de Programas Concorrentes

- **Provas formais de corretude**
- **lógica temporal**
- **linguagens que permitem verificações formais de propriedades**
  - **ADA (Rendevouz)**
  - **CCS**
  - **CSP**
  - **Estelle**
  - **Lotus**
  - **Redes de Petri**
  - **Pi Calculus**

Problema dos Filósofos

- problema clássico de exclusão mútua
- filósofos pensam e comem em um tempo arbitrário
- para comer: 2 pauzinhos
- evitar
  - starvation (esfomeação)
  - deadlock
- soluções
  - semáforo
  - monitor



SOLUÇÃO POR SEMÁFORO

```
#define N 5 /* Número de filósofos */
#define RIGHT(i) (((i)+1) %N)
#define LEFT(i) (((i)==0) ? (N-1) : (i)-1)
#define THINKING 0
#define HUNGRY 1
#define EATING 2
semaphore mutex =1;
semaphore s[N]; /* inicializados com zero */
```

```
void take_forks(int i) {
    wait(&mutex);
    state[i]= HUNGRY;
    test(i);
    signal(&mutex);
    wait(&s[i]);
}
```

```
void test(int i) {
    if ( (state[i] == HUNGRY) && (state[LEFT(i)] != EATING)
    && (state[RIGHT(i)] != EATING) ) {
        state[i] = EATING;
        signal(&s[i]);
    }
}
```

```
void put_forks(int i) {
    wait(&mutex);
    state[i]= THINKING;
    test(LEFT(i));
    test(RIGHT(i));
    signal(&mutex);
}
```

```
void philosopher(int i) {
    while(.....) {
        think();
        take_forks(i); /*obtém 2 pausinhos ou bloqueia*/
        eat();
        put_forks(i);
    }
}
```

SOLUÇÃO POR MONITOR

Pascal

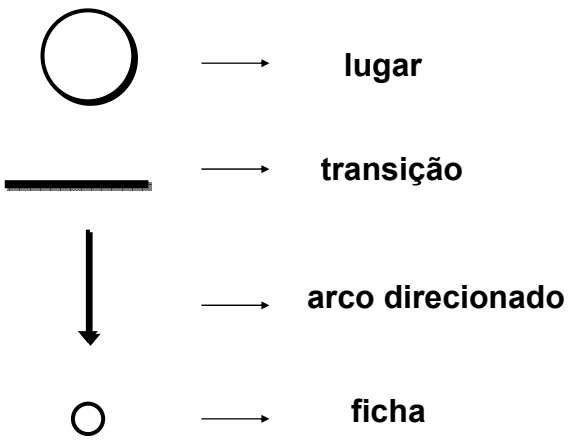
```
type dining-philosophers = monitor
  var state : array [0..4] of :(thinking, hungry, eating);
  var self : array [0..4] of condition;
  procedure entry pickup (i: 0..4);
   begin
              state[i] := hungry,
              test (i);
              if state[i] != eating then self[i], wait,
   end;
  procedure entry putdown (i: 0..4);
   begin
              state[i] := thinking;
              test (i+4 mod 5);
              test (i+1 mod 5);
   end;
  procedure test(k: 0..4);
   begin
              if state[k+4 mod 5] != eating
              and state[k] = hungry
              and state[k+1 mod 5] ] != eating
              then begin
                  state[k] := eating;
                  self[k].signal;
              end;
   end;
begin
              for i := 0 to 4
              do state[i] := thinking;
end.
```

JAVA

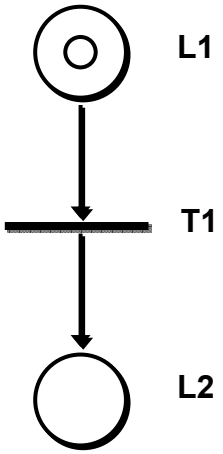
```
monitor DiningPhilosophers {
  int state = new int [5];
  static final int thinking = 0;
  static final int hungry= 1;
  static final int eating= 2;
  condition[] self = condition[5];
  public dinningPhilosophers {
              for(int i=0;i<5;i++)
                  state[i] = thinking;

  public entry pickup (int i) {
              state[i] := hungry;
              test (i);
              if (state[i] !=eating)
                  self[i].wait;
   }
   public entry putdown (int i) {
              state[i] := thinking;
              test (i+4 mod 5);
              test (i+1 mod 5);
   }
   private test(int i) {
              if (state[k+4 mod 5] != eating)
              && (state[k] == hungry)
              && (state[k+1 mod 5] != eating) {
                  state[k] := eating;
                  self[k].signal;
              }
    }
}
```


Redes de Petri
Componentes Básicos

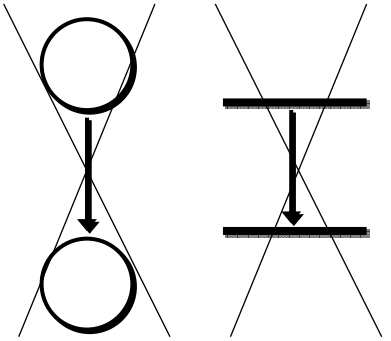


Pode

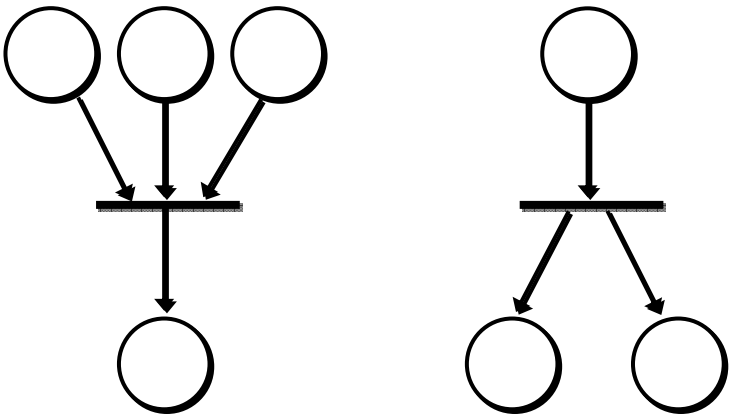


Não Pode

- **Lugares não podem ser ligados a lugares**
- **Transições não podem ser ligadas a transições**

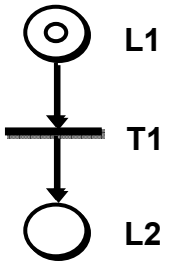


Exemplos corretos

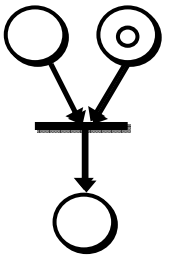


Transição Habilitada

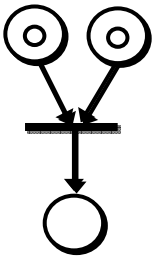
- **Há pelo menos UMA FICHA em cada um dos LUGARES de onde provêm os ARCOS QUE CHEGAM em uma transição. Diz-se, então, que a transição está habilitada para o *DISPARO.***



HABILITADA



NÃO HABILITADA

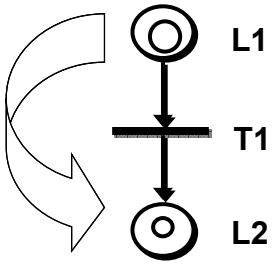


HABILITADA

Disparo de Transição

- **Retira-se uma ficha de cada um dos lugares de onde provêm os ARCOS QUE CHEGAM a transição ("lugares de entrada") e deposita-se uma ficha em cada um dos lugares para onde vão os ARCOS QUE SAEM da transição ("lugares de saída").**

DISPARO



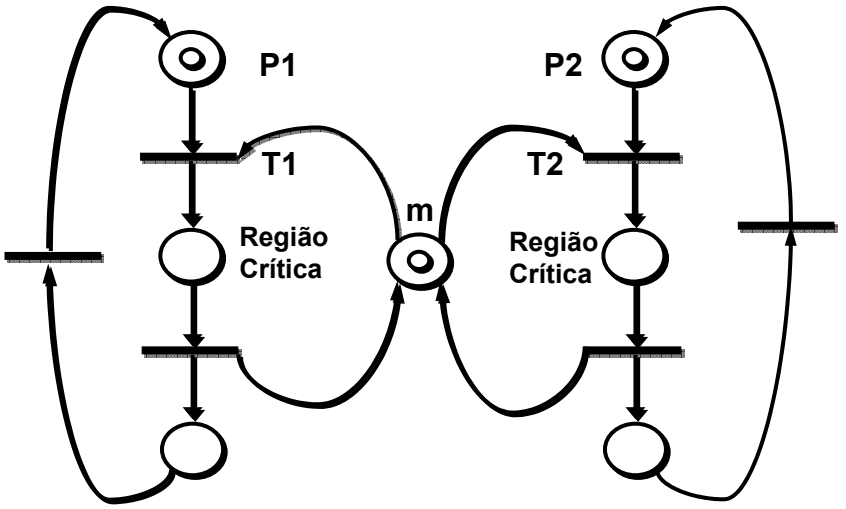
MODELAGEM COM REDES DE PETRI

- **Lugares**
  - **Estados estáveis de um processo ou thread**
    - » **Pronto para executar**
    - » **Bloqueado em semáforo**
    - » **Dentro da Região Critica**
      - **Pode ser refinado (explodir um lugar em uma RP arbitrariamente complexa)**
- **Transições**
  - **Momentos "instávies" de um processo ou thread**
    - » **Execução de instrução ou bloco de instruções**
    - » **Evolução, avaliação de condições, desvio**
- **Lugares como semáforos (sem disciplina)**
- **Fichas**
  - **"program counter" de um processo ou thread**
  - **Marca o estado corrente**
  - **Contagem (número de fichas em um lugar)**
  - **Informação sendo transportada**

SEMÁFORO

- **O lugar m representa a permissão para entrar na região crítica. Para isso um processo precisa obter a ficha em P1 ou P2, sinalizando que vai entrar na região critica e obter a ficha de m. Somente uma transição pode disparar, o disparo de t1 desabilita t2 e estabelece uma espera para o processo P2, até o processo 1 sair da região crítica e repor a ficha em m.**

MODELAGEM: SEMÁFORO



PRODUTORES E CONSUMIDORES

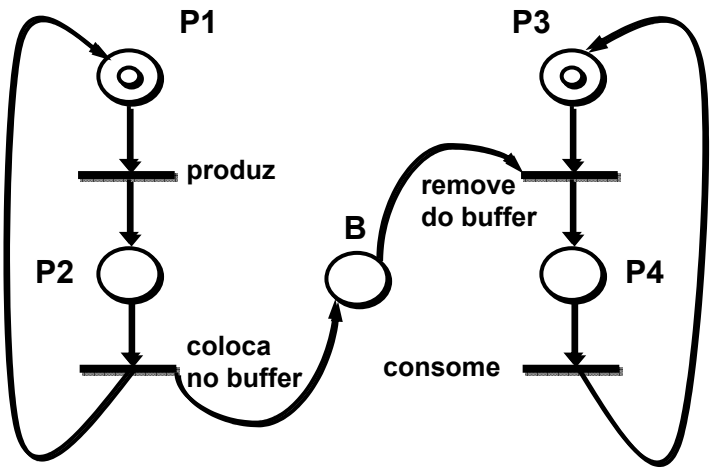
- **Este problema envolve o compartimento de um BUFFER por dois processos.**

  Processo Produtor → Cria um objeto que é colocado no BUFFER.

  Processo consumidor → Espera o produto ser colocado no BUFFER, e consome-o

- **O lugar B referente ao BUFFER B. Cada ficha representa um item que deve ser produzido mais ainda não consumido.**

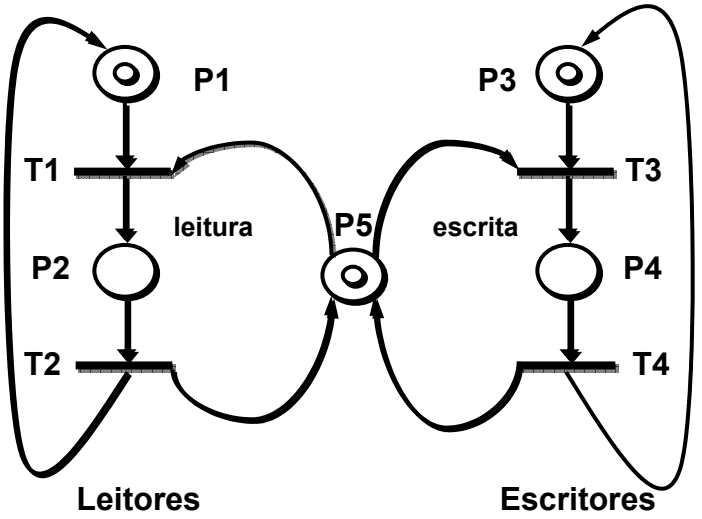
MODELAGEM: PRODUTORES/CONSUMIDORES



LEITORES E ESCRITORES

- **Neste problema todos os processos compartilham uma fila, uma variável e um objeto.**
- **Os processos Leitores nunca modificam o objeto enquanto os processos Escritores o fazem deste modo, os processos escritores devem obrigatoriamente excluir todos os processos escritores, mas múltiplos processos leitores podem acessar o objeto compartilhando simultaneamente.**

MODELAGEM: LEITORES/ESCRITORES



FILÓSOFOS

- **Este problema refere-se a cinco filósofos que comem e pensam alternativamente. Os filósofos estão sentados ao redor de uma mesa com comida chinesa, entre cada filósofo há um pauzinho, para comer a comida chinesa é necessária dois pauzinhos. O problema é que se todos os filósofos pegam o pauzinho a sua esquerda ficam esperando pelo que fica do lado direito.**
- **Os lugares C1...C5, representam os pauzinhos. Cada filósofo é representado por dois lugares Mj e Ej. Para um filósofo passar do estado de meditação para refeição precisa dos dois pauzinhos livres.**

MODELAGEM: FILÓSOFOS