Sistemas embarcados

Técnicas de projeto de software de sistemas embarcados

Projeto com laço infinito

O projeto de software com laço infinito é baseado no conceito *background / foreground.*

Um laço infinito executa determinadas tarefas através da chamada de módulos (implementados no formato de **funções**) para realizar as operações desejadas.

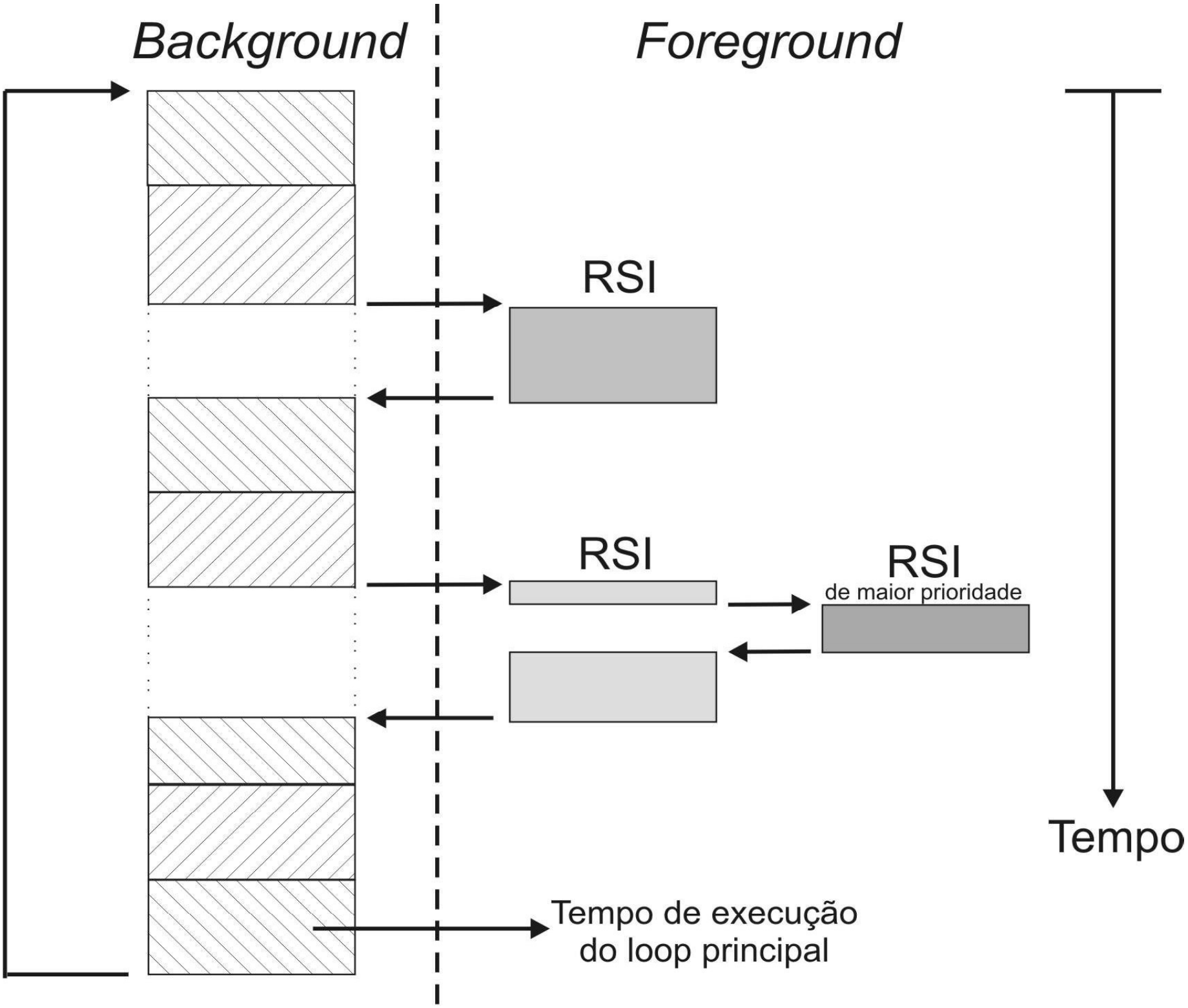
Este laço infinito é conhecido por *background*.

Projeto com laço infinito

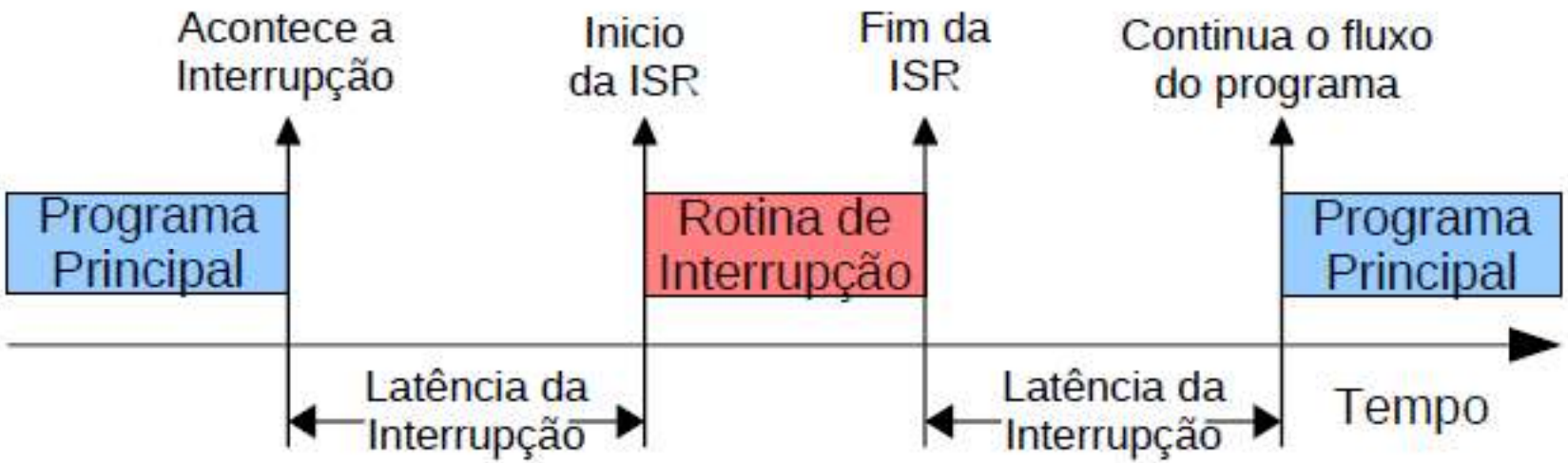
Rotinas de serviço de interrupção (RSI) são utilizadas para o tratamento de eventos assíncronos.

Quando o sistema encontra-se nestas rotinas, define-se que está em *foreground*.

As posições de *foreground* e *background* são também conhecidas por “Nível de Interrupções” e “Nível de Tarefas”, respectivamente.


Background
Foreground
RSI
RSI
RSI
de maior prioridade
Tempo
Tempo de execução
do loop principal

Projeto com laço infinito



Projeto com laço infinito

```
for ( ;  ; )
{
    if(display == 1)
    {
      display = 0;
      atualiza_display();
    }

    if(teclado == 1)
    {
      teclado = 0;
      le_teclado();
    }

    if(serial == 1)
    {
      serial = 0;
      le_serial();
    }
}
```

Nesta técnica, geralmente, são utilizadas variáveis *flags* para indicar quais tarefas devem ser executadas.

Estas *flags* são alteradas pelas rotinas de interrupção.

Projeto com laço infinito

Prós

- ✓ Simplicidade.
- ✓ Atendimento rápido de interrupções.
- ✓ Facilidade para projeto de baixo consumo. Processador permanece em estado “dormente” e “acorda” para executar as interrupções.
- ✓ Consumo de memória RAM é facilmente gerenciável.

Contras

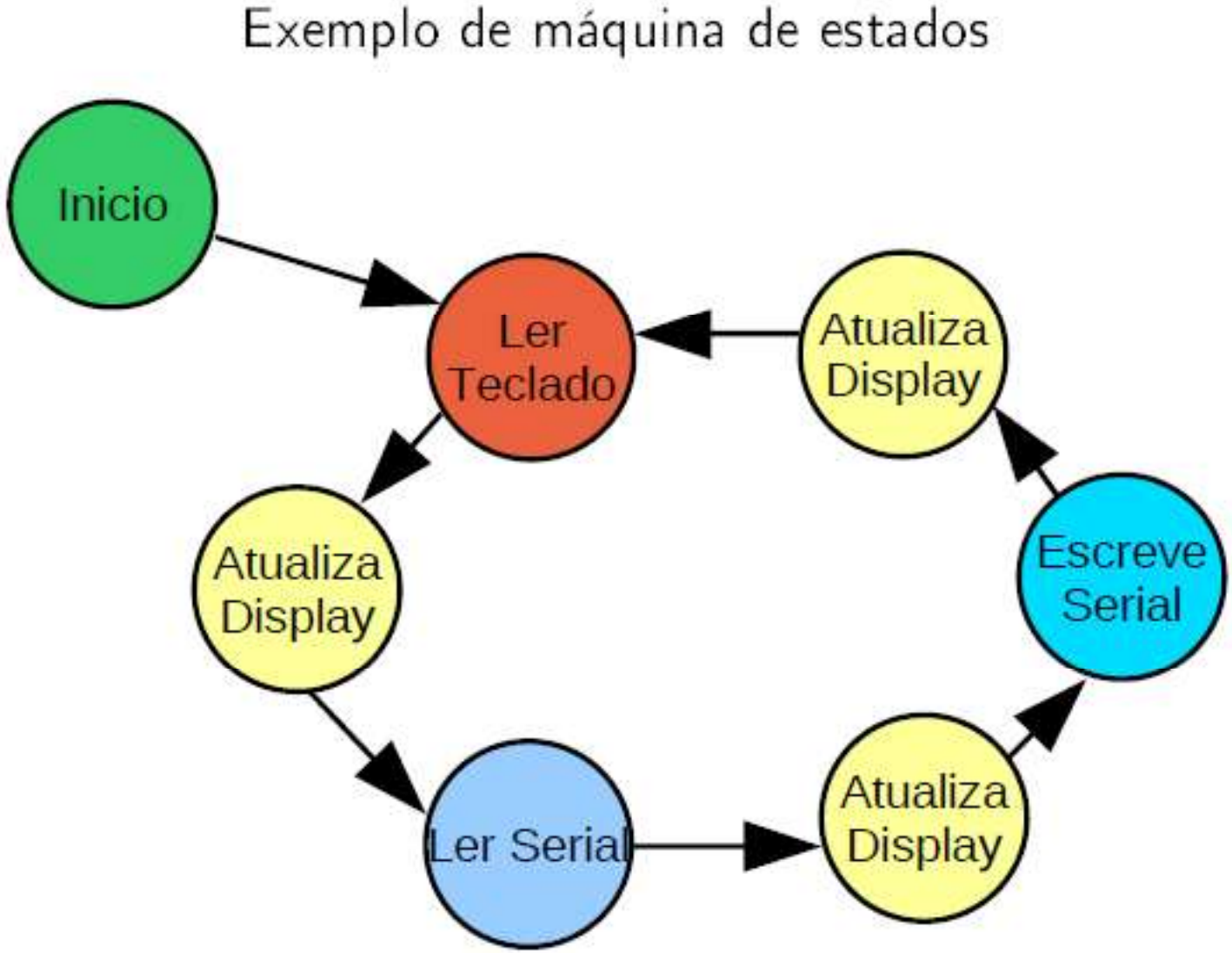
- ✗ Tempo de execução das rotinas não-determinístico
- ✗ Difícil manutenção conforme aumenta a complexidade do software.
- ✗ Não é facilmente escalável.
- ✗ Não recomendado para sistemas de tempo real, por ser difícil de garantir prazos para execução das tarefas.

Projeto com máquinas de estados finitos (FSM)

Uma técnica mais organizada de se projetar o software de um sistema embarcado é a utilização do modelo de uma máquina de estados finitos (FSM).

Nesta técnica, o sistema é separado em estados em que ele pode estar e cada estado executa uma das tarefas apenas.

Projeto com máquinas de estados finitos (FSM)



```
void main(void){
  char slot;
  //Inicializa...
  for(;;){  //inicio do loop infinito
  //********************* top-slot ************************
  //************ início da maquina de estado *************
    switch(slot){
      case 0:
        LeTeclado();         slot = 1; break;
      case 1:
        AtualizaDisplay(); slot = 2; break;
      case 2:
        RecebeSerial();     slot = 3; break;
      case 3:
        AtualizaDisplay(); slot = 4; break;
      case 4:
        EnviaSerial();      slot = 5; break;
      case 5:
        AtualizaDisplay(); slot = 0; break;
      default:
        slot = 0; break;
    }
  //************* fim da maquina de estado ***************
  //******************** bottom-slot **********************
  } //fim loop infinito (!?)
}
```

Implementação em C
com switch-case

Projeto com FSM

Prós

- ✓ Simplicidade.
- ✓ Atendimento rápido de interrupções.
- ✓ Facilidade para projeto de baixo consumo. Processador permanece em estado “dormente” e “acorda” para executar as interrupções.
- ✓ Consumo de memória RAM é facilmente gerenciável.
- ✓ Mais escalável que o laço infinito, pois é mais fácil adicionar tarefas

Contras

- ✗ Tempo de execução das rotinas ainda é não-determinístico.
- ✗ Não recomendado para sistemas de tempo real, por ser difícil de garantir prazos para execução das tarefas.

Projeto com FSM temporizada

- Pode-se temporizar a FSM de forma a tornar o funcionamento do sistema mais previsível.

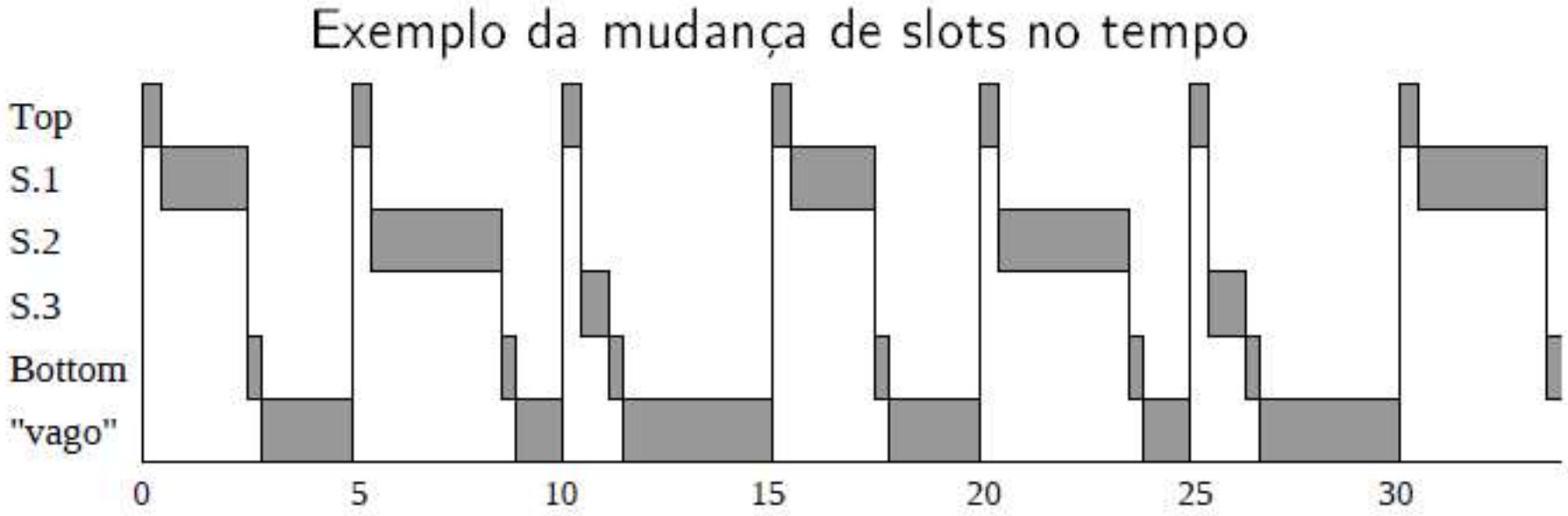
- Deste modo cada estado tem um tempo máximo constante de execução.

- Porém, ainda é difícil garantir que uma determinada tarefa será executa dentro de um determinado prazo.

```
void main(void){
  char slot;
  //Inicializa...
  for(;;){  //inicio do loop infinito
  //********************* top-slot ************************
    ResetaTimer(5000);  //5 ms para cada slot
    AtualizaDisplay();
  //************ início da maquina de estado *************
    switch(slot){
      case 0:
        LeTeclado();        slot = 1; break;
      case 1:
        RecebeSerial();     slot = 2; break;
      case 2:
        EnviaSerial();      slot = 0; break;
      default:
        slot = 0; break;
    }
  //************ fim da maquina de estado **************
  //******************* bottom-slot ***********************
    AguardaTimer();
  } //fim loop infinito (!?)
}
```

Implementação de FSM em C com switch-case e temporização

Projeto com FSM temporizada



Projeto com FSM temporizada