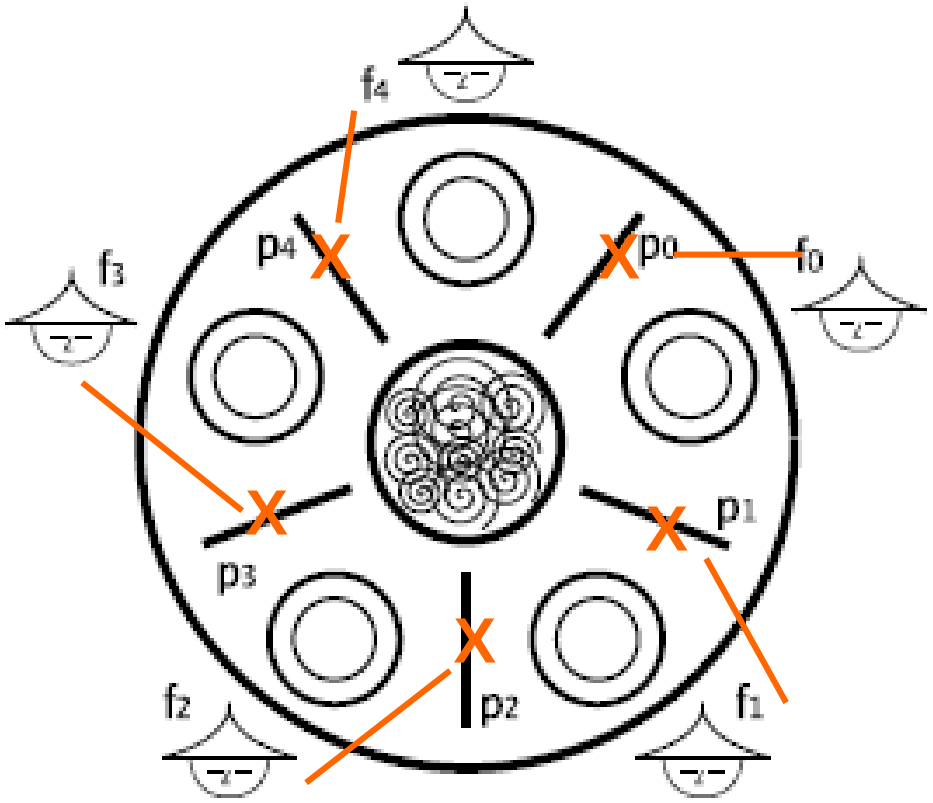
Impasses

Índice

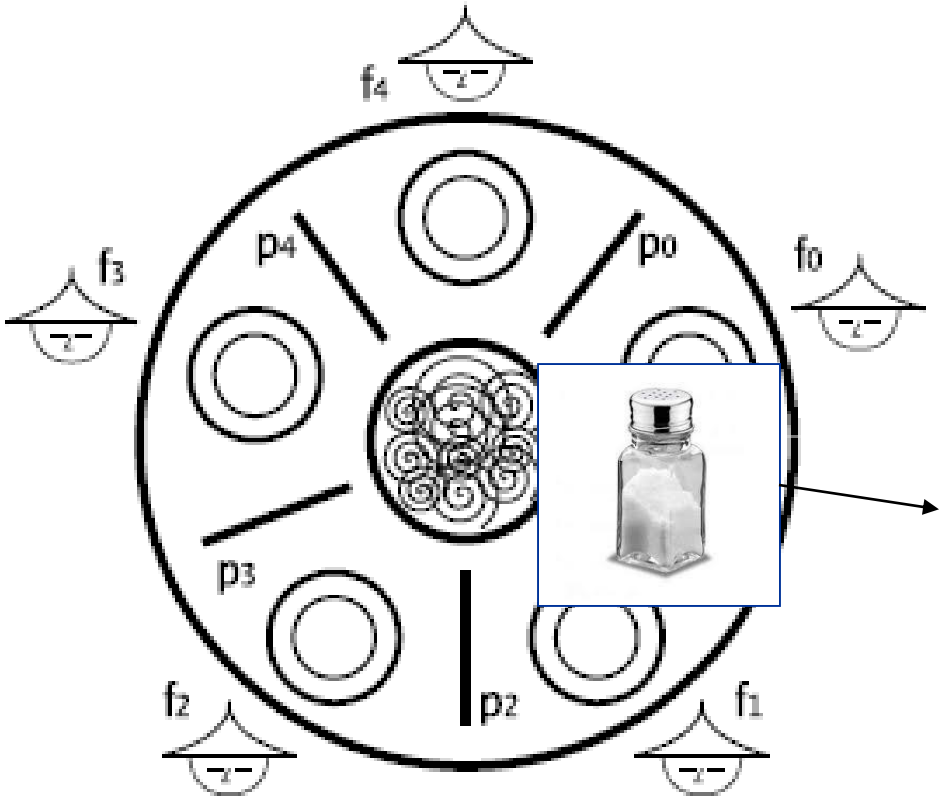
- Definição de impasse
- Exemplo com operação bancária
- Caracterização
- Condições necessárias
- Técnicas
  - Prevenção
  - Impedimento
  - Detecção e Resolução

Problema



⇒ **IMPASSE (Deadlock)**

Possível solução



Possível solução

```
#define NUMFILO 5
semaphore hashi [NUMFILO] ; // um semáforo para cada palito (iniciam em 1)
semaphore saleiro ;         // um semáforo para o saleiro

task filosofo (int i)               // filósofo i (entre 0 e 4)
{
   int dir = i ;
   int esq = (i+1) % NUMFILO] ;

   while (1)
   {
      meditar () ;
      down (saleiro) ;              // pega saleiro
      down (hashi [dir]) ;          // pega palito direito
      down (hashi [esq]) ;          // pega palito esquerdo
      up (saleiro) ;                // devolve saleiro
      comer () ;
      up (hashi [dir]) ;            // devolve palito direito
      up (hashi [esq]) ;            // devolve palito esquerdo
   }
}
```

Impasses

O controle de concorrência entre tarefas acessando recursos compartilhados:

→ **Suspensão** de algumas tarefas enquanto outras acessam os recursos de forma a garantir consistência.

Em alguns casos o uso de **semáforos** pode levar a situações de **impasse**, nas quais todas as tarefas envolvidas ficam suspensas aguardando a liberação dos recursos.

Ex.: Transferência

```
typedef struct conta_t
{
        int saldo ;             // saldo atual da conta
        sem_t lock ;            // semáforo associado à conta
...                             // outras informações da conta
} conta_t ;
```

Ex.: Transferência

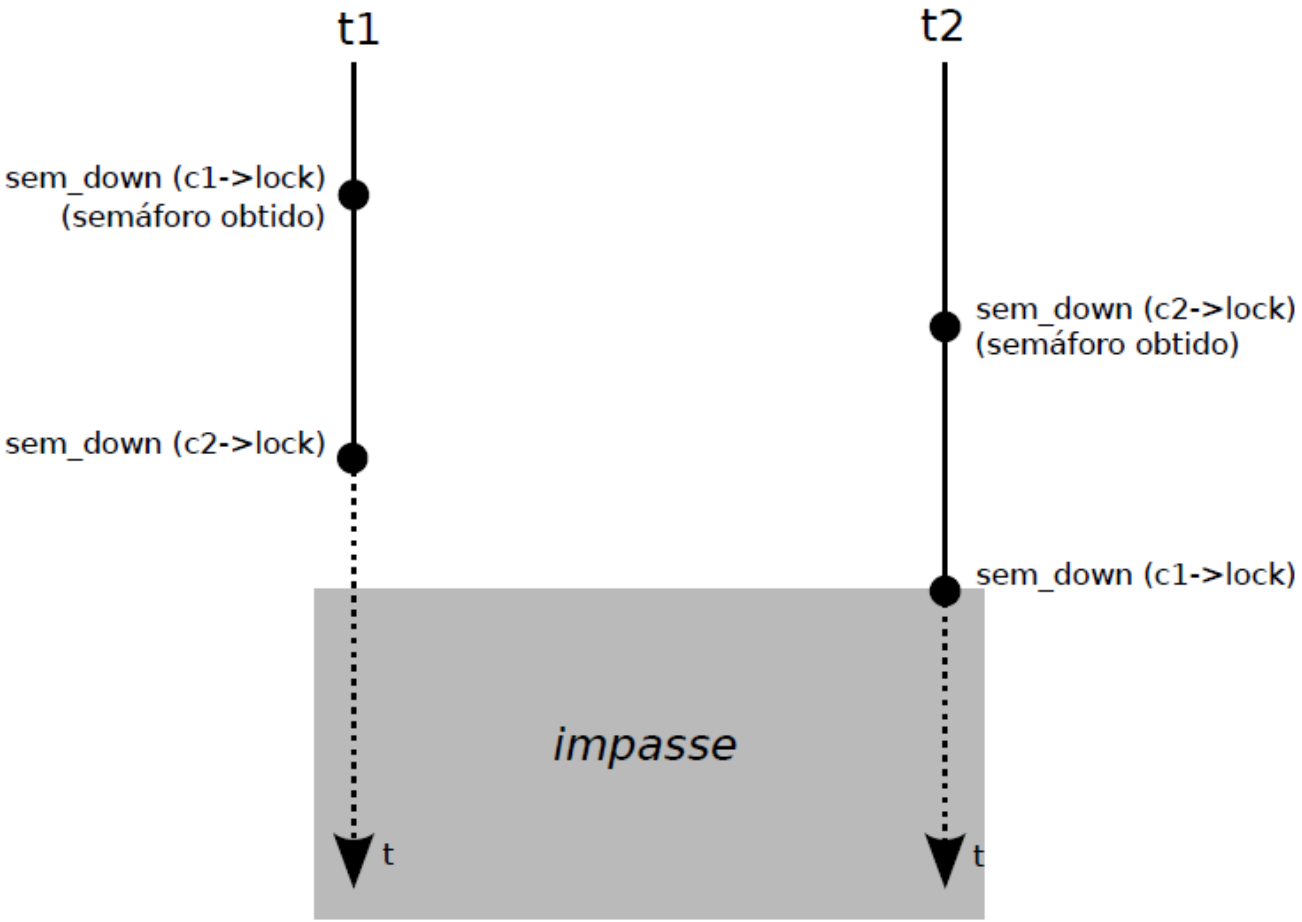
```
void transferir (conta_t* contaDeb, conta_t* contaCred, int valor)
{
        sem_down (contaDeb->lock) ;    // obtém acesso a contaDeb
        sem_down (contaCred->lock) ;   // obtém acesso a contCred

        if (contaDeb->saldo >= valor)
          {
            contaDeb->saldo -= valor ;   // debita valor de contaDeb
            contaCred->saldo += valor ; // credita valor em contaCred
          }
sem_up (contaDeb->lock) ;                // libera acesso a contaDeb
sem_up (contaCred->lock) ;               // libera acesso a contaCred
}
```

Ex.: Transferência bancária

t1 – Transferir da conta C1 para conta C2

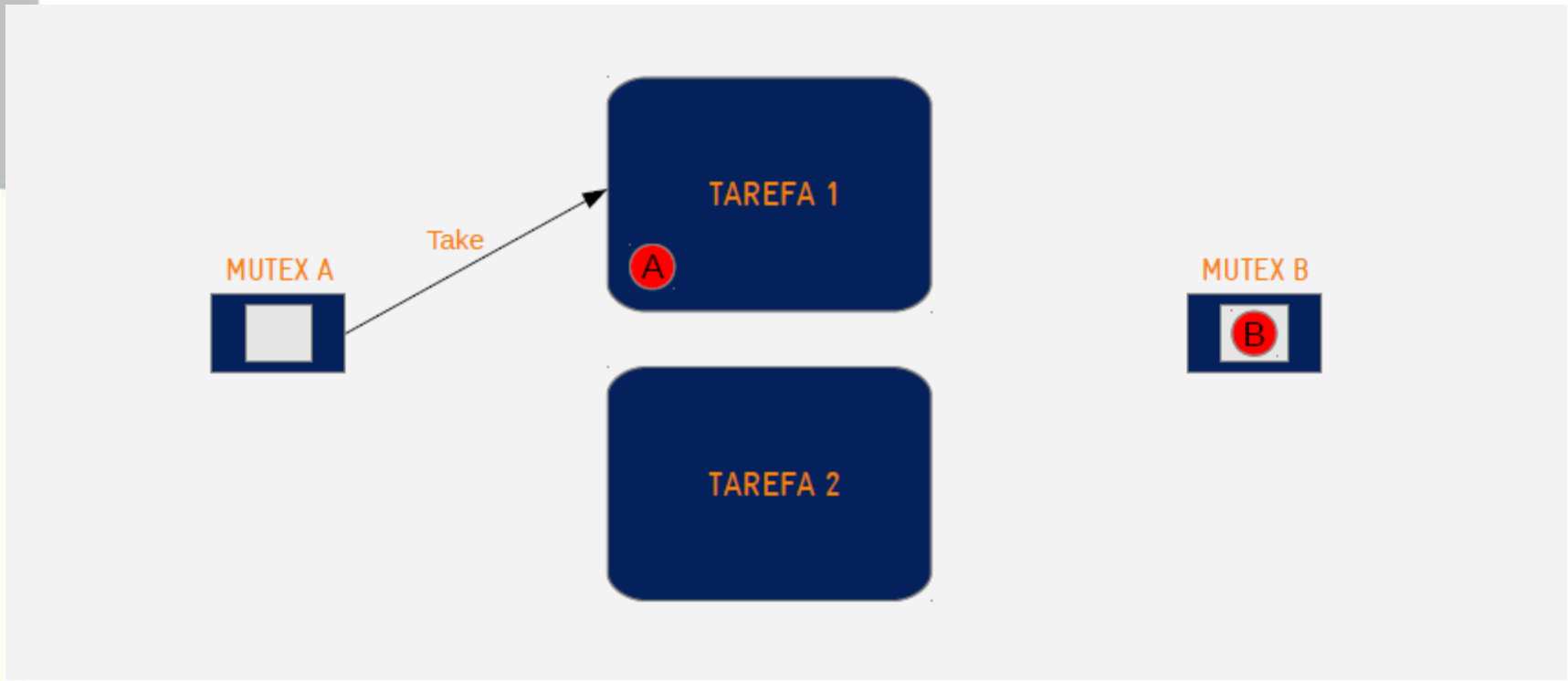
t2 – Transferir da conta C2 para conta C1

Ex.: Transferência bancária

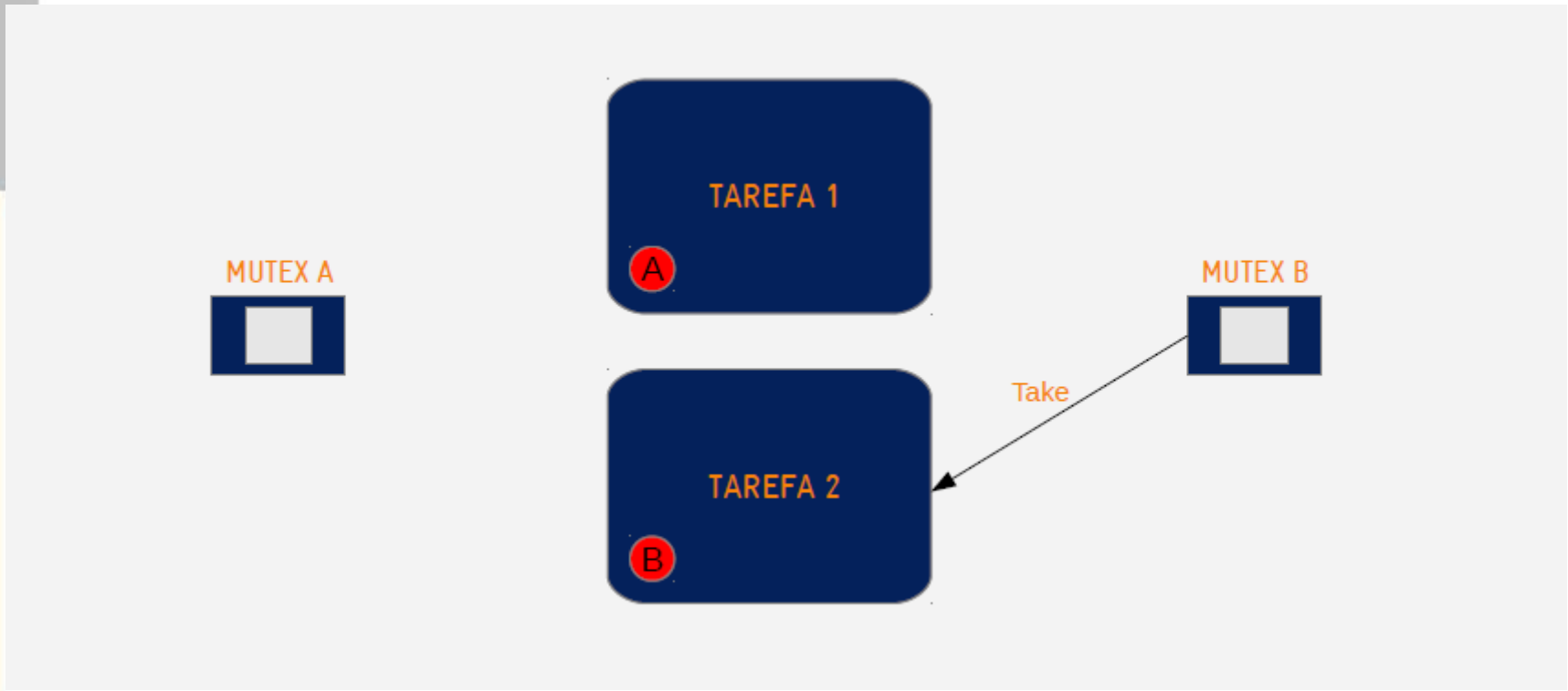


Impasses

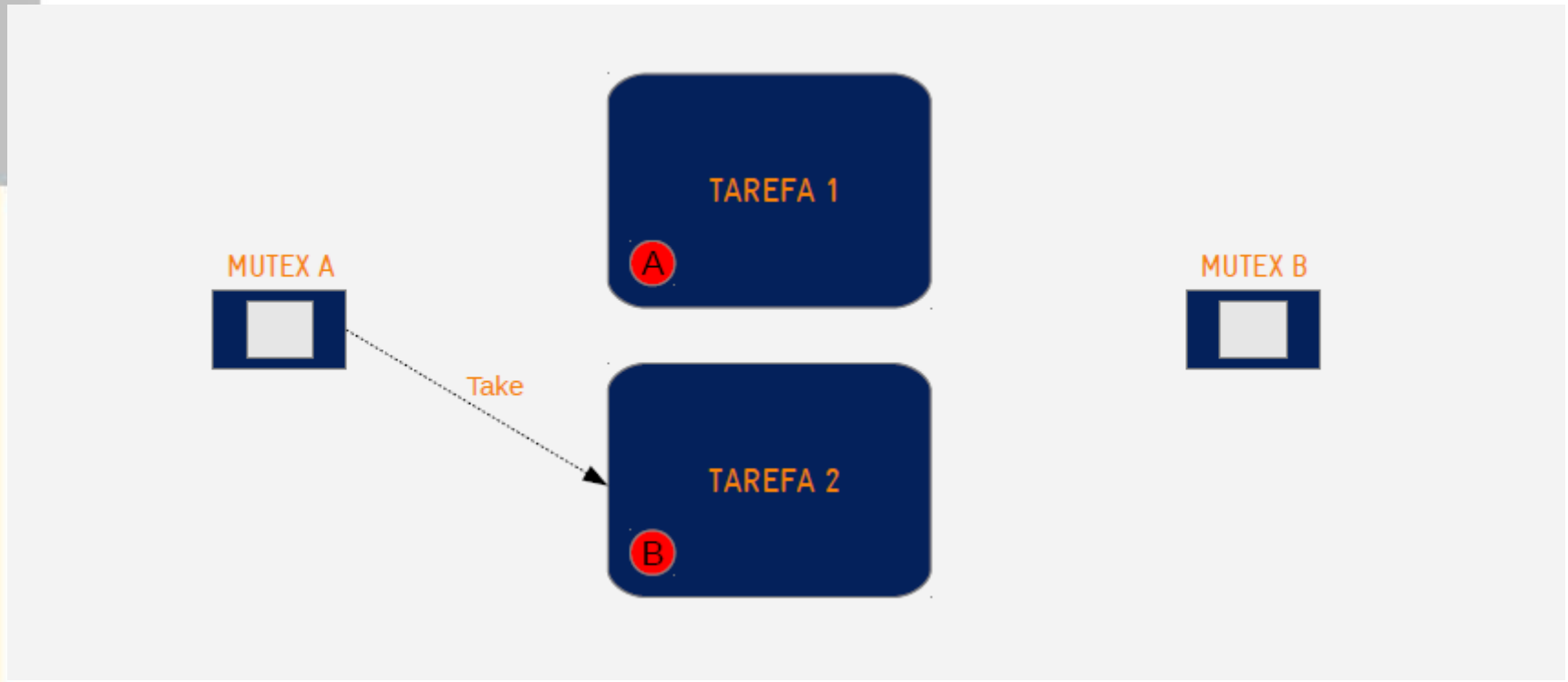
Impasses



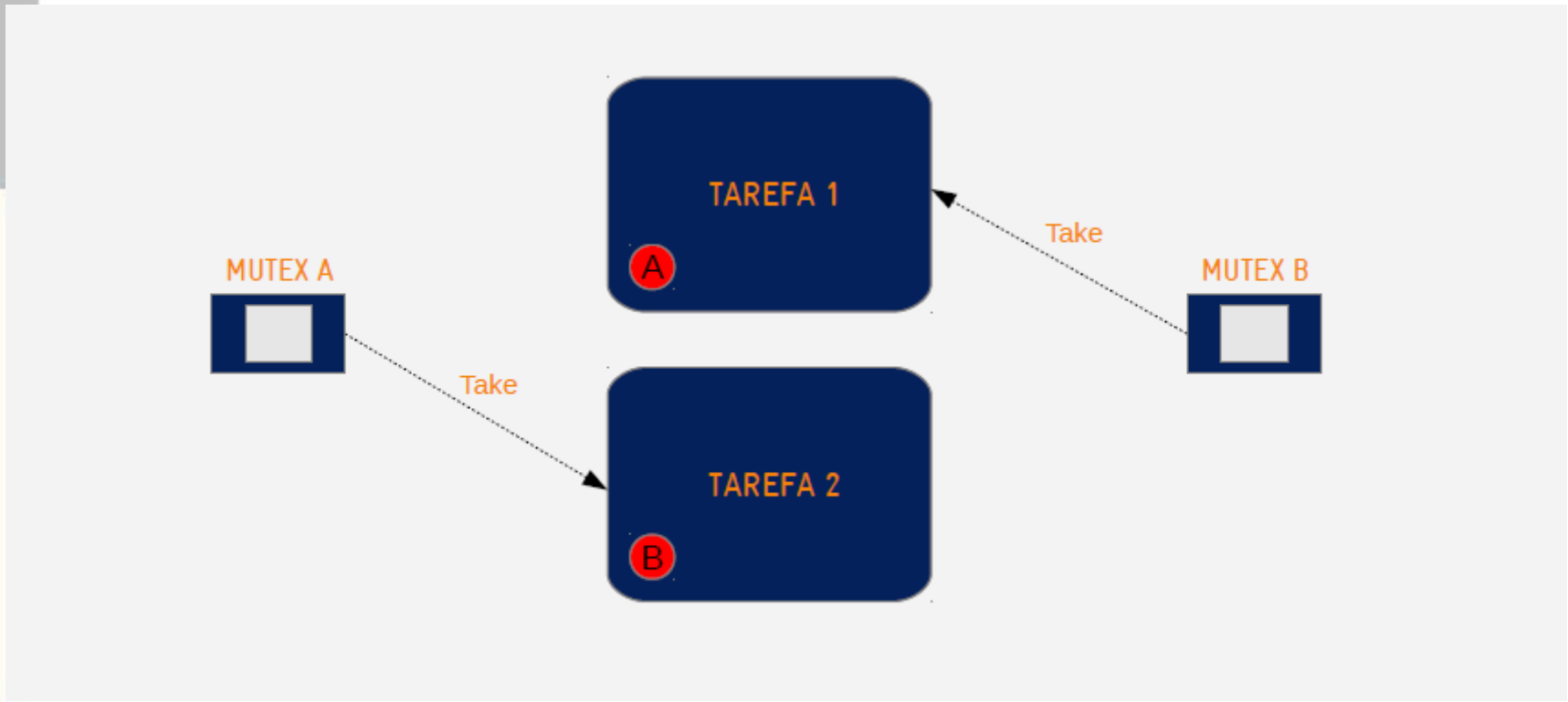
Impasses



Impasses



Impasses



Dead-lock

Impasses

É importante compreender suas principais causas e saber caracterizá-los adequadamente.

Caracterização de impasses

- Em um **impasse**, duas ou mais tarefas se encontram suspensas, aguardando eventos que dependem **somente delas**.

Caracterização de impasses

- Em um **impasse**, duas ou mais tarefas se encontram suspensas, aguardando eventos que dependem **somente delas**.

- Não existe influência de entidades externas em uma situação de impasse.

Caracterização de impasses

- Em um **impasse**, duas ou mais tarefas se encontram suspensas, aguardando eventos que dependem **somente delas**.

- Não existe influência de entidades externas em uma situação de impasse.

- Como as tarefas envolvidas detêm alguns recursos compartilhados, outras tarefas que vierem a requisitá-los também ficarão suspensas, aumentando gradativamente o **impasse**

Consequência...

Caracterização de impasses

Definição

Um conjunto de ***N* tarefas** se encontra em um impasse se cada uma das tarefas aguarda (suspensa) um evento que somente outra tarefa do conjunto poderá produzir.

Condições necessárias

C1 – Exclusão mútua : o acesso aos recursos deve ser feito de forma mutuamente exclusiva. No exemplo da conta corrente, apenas uma tarefa por vez pode acessar cada conta.

Condições necessárias

C1 – Exclusão mútua : o acesso aos recursos deve ser feito de forma mutuamente exclusiva. No exemplo da conta corrente, apenas uma tarefa por vez pode acessar cada conta.

C2 – Posse e espera : uma tarefa pode solicitar o acesso a outros recursos sem ter de liberar os recursos que já detém. No exemplo da conta corrente, cada tarefa detém o semáforo de uma conta e solicita o semáforo da outra conta para poder prosseguir.

Condições necessárias

C3 – Não-preempção : uma tarefa somente libera os recursos que detém quando assim o decidir, e não pode perdê-los contra a sua vontade. No exemplo da conta corrente, cada tarefa detém indefinidamente os semáforos que já obteve.

Condições necessárias

C3 – Não-preempção : uma tarefa somente libera os recursos que detém quando assim o decidir, e não pode perdê-los contra a sua vontade. No exemplo da conta corrente, cada tarefa detém indefinidamente os semáforos que já obteve.

C4 – Espera circular : existe um ciclo de espera pelos recursos entre as tarefas envolvidas: a tarefa *t*1 aguarda um recurso retido pela tarefa *t*2 (formalmente, $t1 \rightarrow t2$), que aguarda um recurso retido pela tarefa *t*3... : $t1 \rightarrow t2 \rightarrow t3 \rightarrow \cdots \rightarrow tn \rightarrow t1$. No exemplo da conta corrente, pode-se observar que $t1 \rightarrow t2 \rightarrow t1$.

Condições necessárias

- Estas quatro condições são **necessárias** para a formação de impasses;

Condições necessárias

- Estas quatro condições são **necessárias** para a formação de impasses;
- Se uma delas **não** for verificada, **não** existirão impasses no sistema;

Condições necessárias

- Estas quatro condições são **necessárias** para a formação de impasses;
- Se uma delas **não** for verificada, **não** existirão impasses no sistema;
- **Não** são condições **suficientes** para a existência de impasses, ou seja, a verificação destas quatro condições não garante a presença de um impasse no sistema;

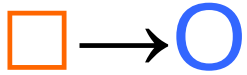
Condições necessárias

- Estas quatro condições são **necessárias** para a formação de impasses;
- Se uma delas **não** for verificada, **não** existirão impasses no sistema;
- **Não** são condições **suficientes** para a existência de impasses, ou seja, a verificação destas quatro condições não garante a presença de um impasse no sistema;
- Estas condições somente são **suficientes** se existir **apenas uma instância** de cada tipo de recurso.

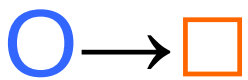
Grafos de alocação de recursos

- Tarefas são representadas por círculos O
- Recursos por retângulos □

- Posse de um recurso por uma tarefa:

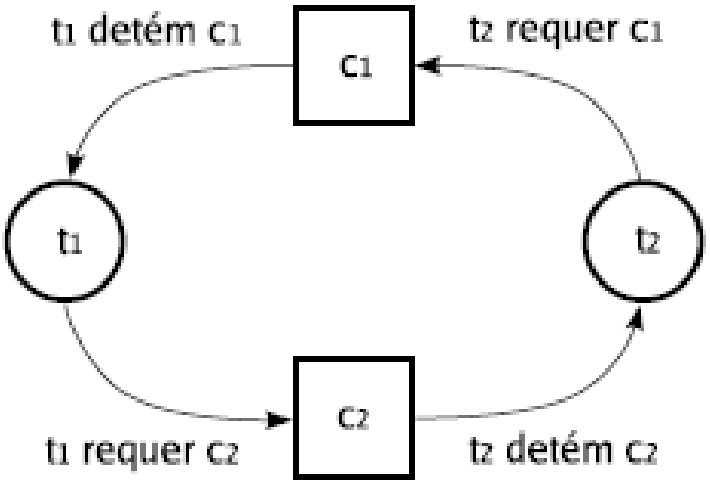


- Requisição de um recurso por uma tarefa:



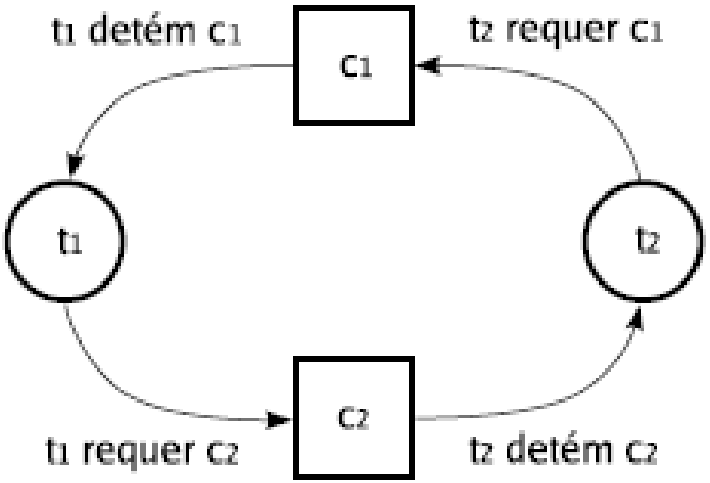
Grafos de alocação de recursos

Permitem detectar visualmente a presença de esperas circulares



Grafos de alocação de recursos

Permitem detectar visualmente a presença de esperas circulares

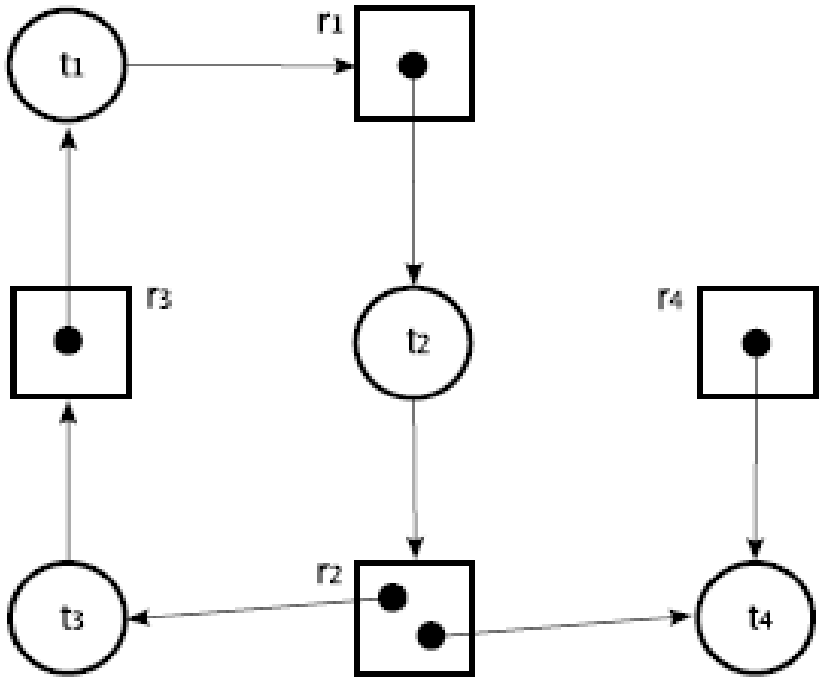


Observe que: $t1 \rightarrow c2 \rightarrow t2 \rightarrow c1 \rightarrow t1$

$\Rightarrow$ **IMPASSE**

Grafos de alocação de recursos

A existência de **múltiplas instâncias** de um recurso é representada por meio de "fichas".



Técnicas de tratamento de impasses

- A existência de impasses pode levar à **paralisação** de todo o sistema.
- Diversas **técnicas de tratamento de impasses** foram propostas.

Técnicas de tratamento de impasses

Quatro abordagens possíveis:

1. Algoritmo do avestruz;

Técnicas de tratamento de impasses

Quatro abordagens possíveis:

1. Algoritmo do avestruz;
   - ⇒ **Windows, Linux, Solaris, etc.**

Técnicas de tratamento de impasses

Quatro abordagens possíveis:

1. Algoritmo do avestruz;
   - ⇒ **Windows, Linux, Solaris, etc.**

2. Regras estruturais que previnam impasse;

Técnicas de tratamento de impasses

Quatro abordagens possíveis:

1. Algoritmo do avestruz;
   - ⇒ **Windows, Linux, Solaris, etc.**

2. Regras estruturais que previnam impasse;

3. Atuar de forma pró-ativa, se antecipando aos impasses e impedindo sua ocorrência;

Técnicas de tratamento de impasses

Quatro abordagens possíveis:

1. Algoritmo do avestruz;
   - ⇒ **Windows, Linux, Solaris, etc.**

2. Regras estruturais que previnam impasse;

3. Atuar de forma pró-ativa, se antecipando aos impasses e impedindo sua ocorrência;

4. Agir de forma reativa, detectando os impasses que se formam no sistema e tomando medidas para resolvê-los.

Prevenção de impasses

Técnicas de prevenção de impasses buscam garantir que impasses nunca possam ocorrer no sistema.

Como?

Prevenção de impasses

Técnicas de prevenção de impasses buscam garantir que impasses nunca possam ocorrer no sistema.

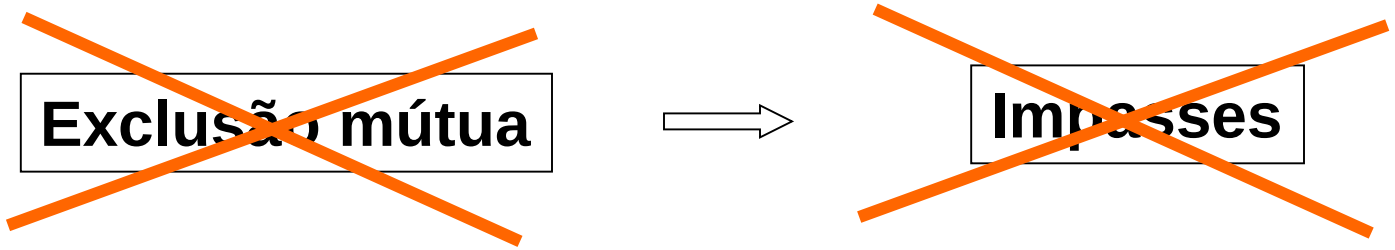
Como?

Evitando que as **4 condições necessárias** sejam satisfeitas.

“Se ao menos uma das quatro condições for quebrada por regras estruturais, os impasses não poderão ocorrer.”

Prevenção de impasses

1 - Exclusão mútua



Mas, como garantir a integridade de recursos compartilhados sem usar mecanismos de exclusão mútua?

Prevenção de impasses

1 - Exclusão mútua

Exemplo:

Gerência de impressoras: um processo *servidor de impressão* (*printer spooler*) gerencia a impressora e atende as solicitações dos demais processos.

Não é facilmente aplicável a outros tipos de recurso.

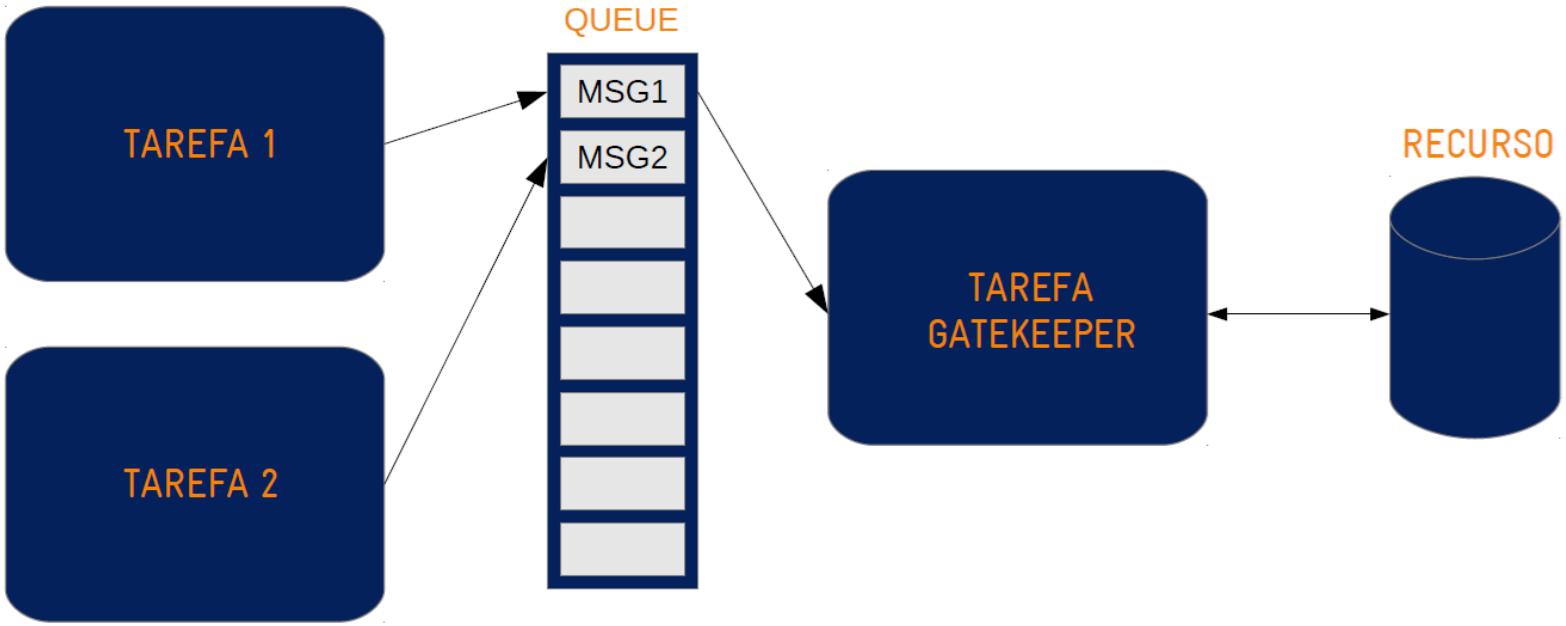
GATEKEEPER

- *Gatekeeper* é uma técnica para implementar um mecanismo de **exclusão mútua** sem os riscos de impasses.

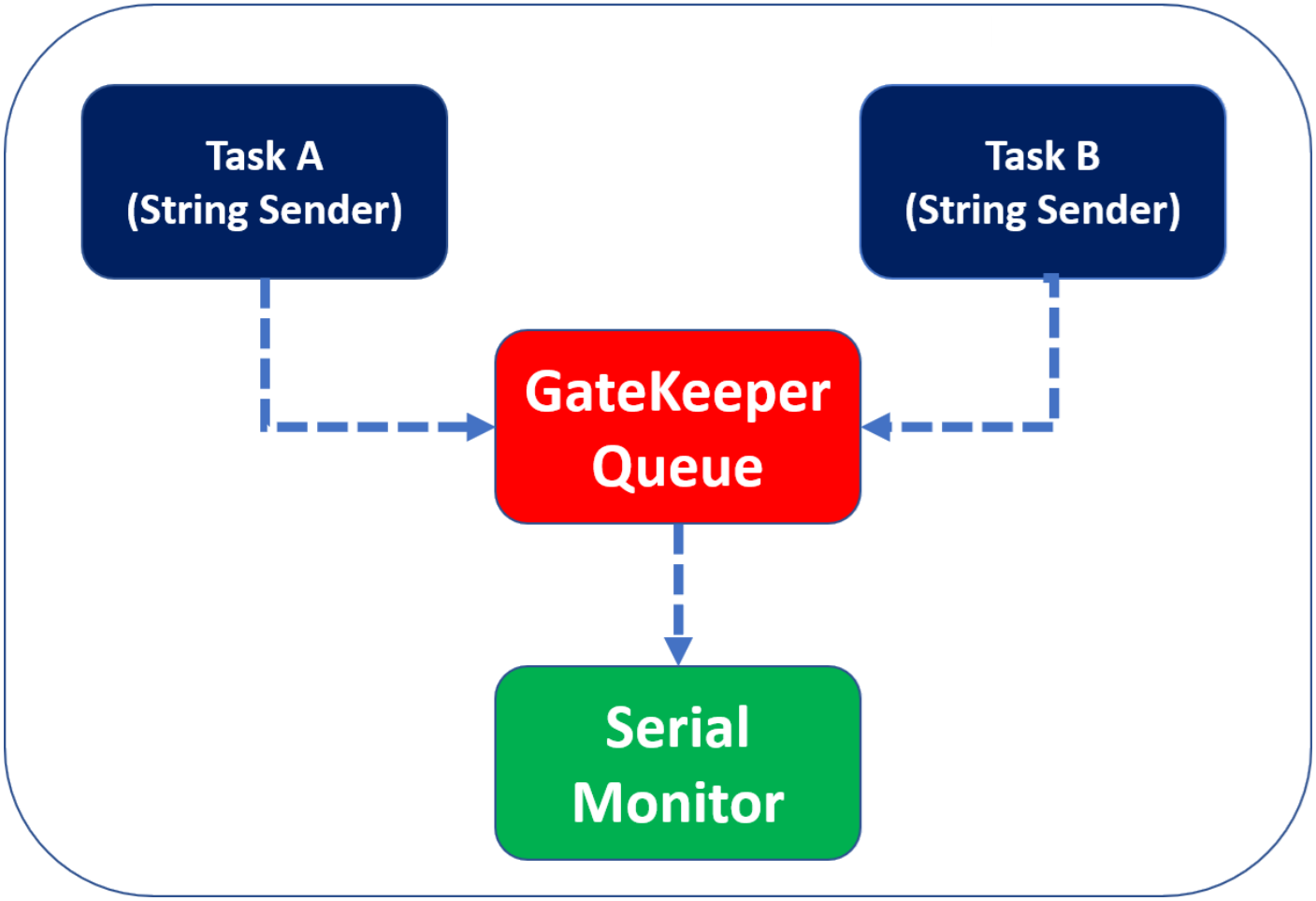
- O *gatekeeper* possui acesso exclusivo a determinado recurso, fornecendo serviços para outras tarefas acessarem este recurso.

- Todas as tarefas que querem acessar o recurso protegido, devem utilizar os serviços fornecidos pela tarefa *gatekeeper*.

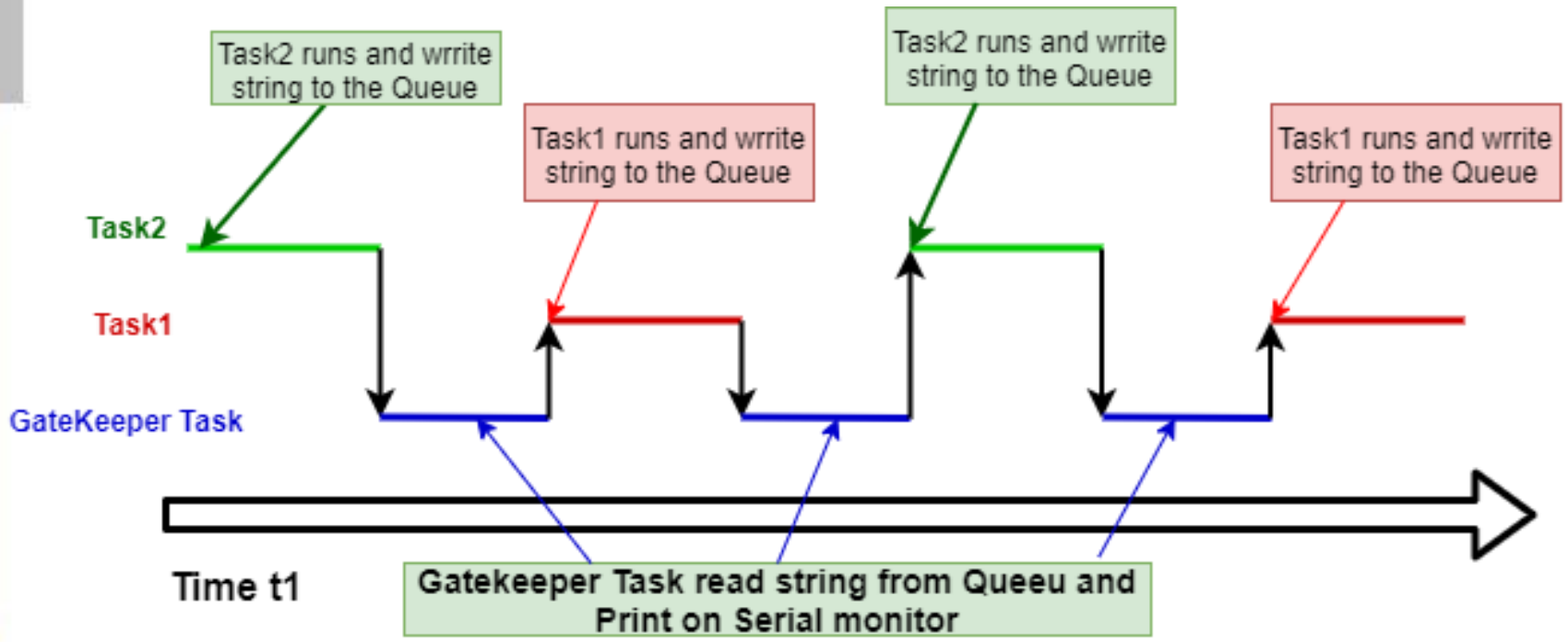
GATEKEEPER


QUEUE
MSG1
MSG2
TAREFA 1
TAREFA 2
TAREFA GATEKEEPER
RECURSO

GATEKEEPER

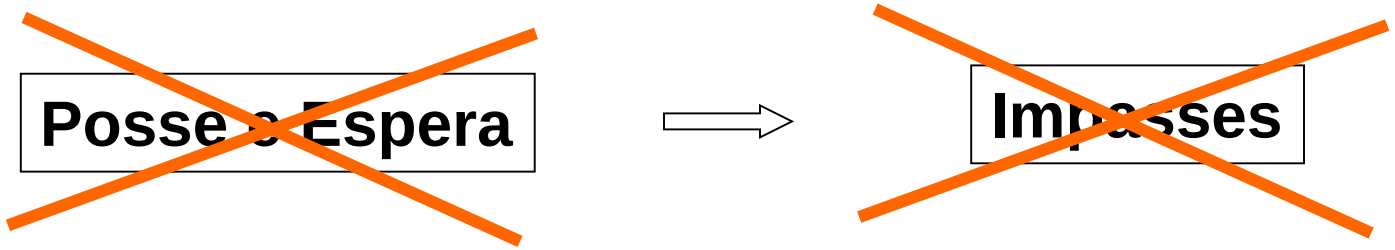

Task A
(String Sender)
Task B
(String Sender)
GateKeeper
Queue
Serial
Monitor

GATEKEEPER


Task2 runs and wrrite string to the Queue
Task1 runs and wrrite string to the Queue
Task2 runs and wrrite string to the Queue
Task1 runs and wrrite string to the Queue
Task2
Task1
GateKeeper Task
Time t1
Gatekeeper Task read string from Queeu and Print on Serial monitor

Prevenção de impasses

2 - Posse e Espera



Caso as tarefas usem apenas um recurso de cada vez, solicitando-o e liberando-o logo após o uso, impasses não poderão ocorrer.

Prevenção de impasses

Exemplo:

Seria possível separar a operação de transferência em duas operações isoladas: débito em *c*1 e crédito em *c*2 (ou vice-versa).

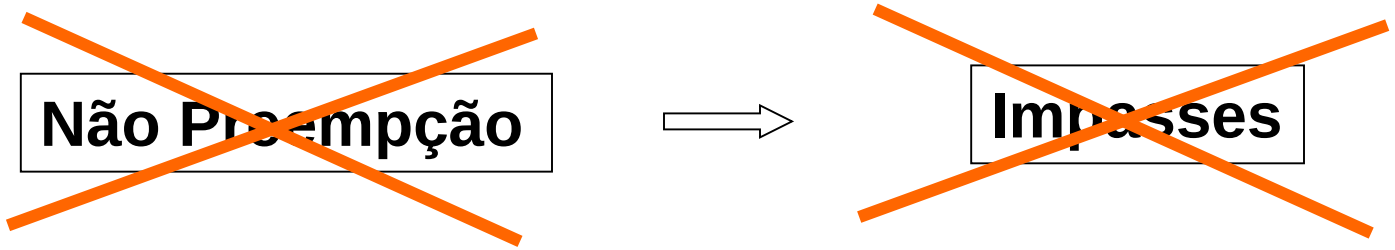
Outras possibilidades:

- Somente permitir a execução de tarefas que detenham todos os recursos necessários antes de **iniciar**.

- Ou associar um prazo (*time-out*) às solicitações de recursos.

Prevenção de impasses

3 – Não Preempção



Prevenção de impasses

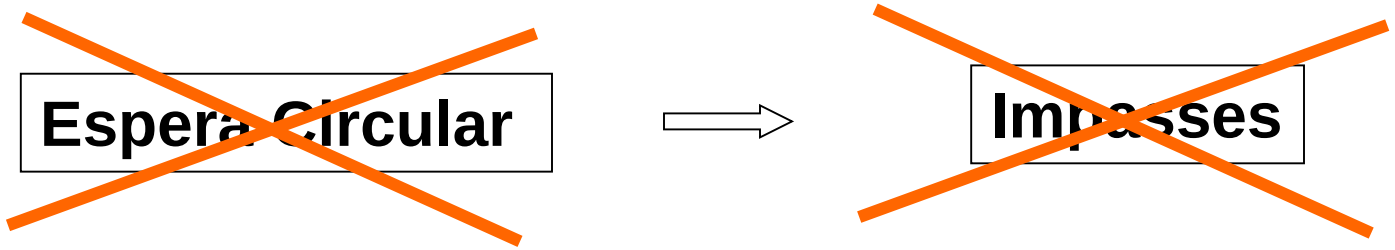
3 – Não Preempção

Técnica frequentemente usada em **recursos** cujo estado interno pode ser salvo e restaurado de forma transparente para a tarefa $\Rightarrow$ páginas de **memória** e o contexto do **processador**.

De difícil aplicação sobre recursos como *arquivos* ou áreas de *memória compartilhada* $\Rightarrow$ a preempção viola a exclusão mútua e pode deixar inconsistências no estado interno do recurso.

Prevenção de impasses

4 – Espera Circular



Prevenção de impasses

4 – Espera Circular

Um impasse é uma cadeia de dependências entre tarefas e recursos que forma um ciclo.

Prevenção de impasses

4 – Espera Circular

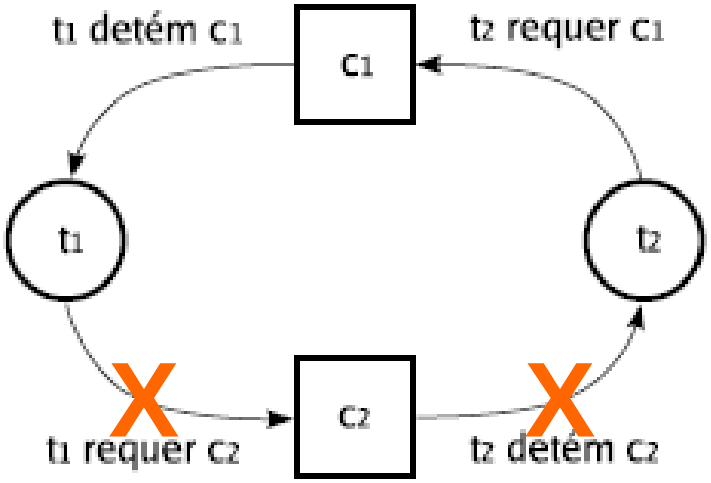
A estratégia mais simples para **prevenir a formação de ciclos** é ordenar todos os recursos do sistema de acordo com uma ordem global única e forçar as tarefas a solicitar os recursos obedecendo a essa ordem.

No exemplo da transferência: cada tarefa deveria acessar primeiramente a conta mais antiga (ou a mais nova).

Solução mais promissora!

Prevenção de impasses

4 – Espera Circular



Prevenção de impasses

Resumo

Condição a ser quebrada	Abordagem
Exclusão mútua	*Spooler; Gatekeeper*
Posse e espera	Um recurso de cada vez
Sem preempção	“Tomar de volta” os recursos
Espera circular	Ordenar numericamente recursos

Impedimento de impasses

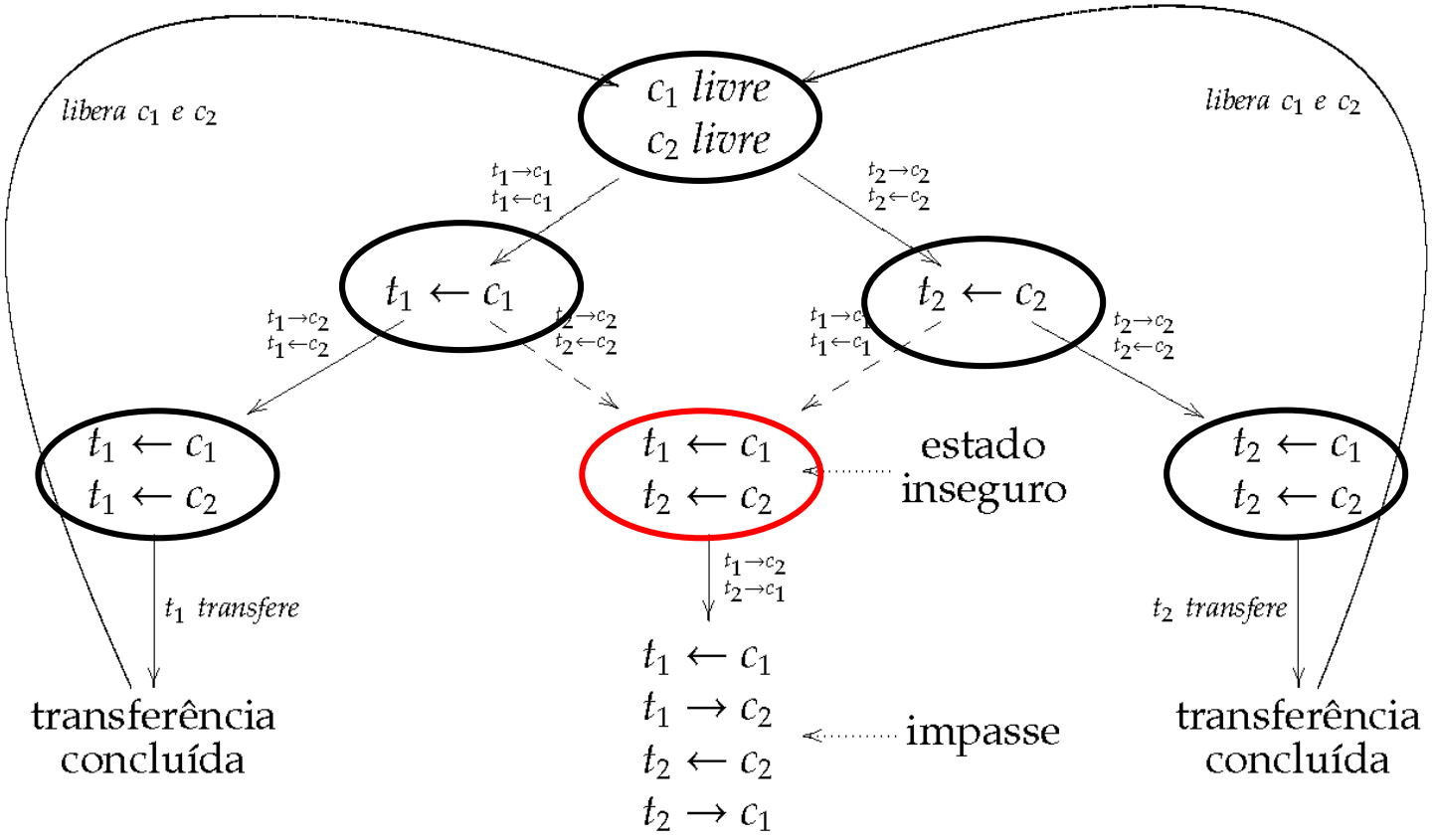
Consiste em **negar acesso a recursos** que possam levar a impasses.

Necessita de conhecimento prévio sobre o comportamento das tarefas $\Rightarrow$ pouco utilizada na prática.

Uma noção essencial nas técnicas de impedimento de impasses é o conceito de estados (in)seguros.

Impedimento de impasses

Ex.: Evitando o *estado inseguro*



Impedimento de impasses

Ex.: Evitando o *estado inseguro*

Um algoritmo de impedimento de impasses deveria **negar** as alocações em tracejado, pois elas levam o sistema a esse estado inseguro.

Exemplo → Algoritmo do banqueiro: Djikstra em 1965.

Detecção e resolução de impasses

- Nesta abordagem, nenhuma medida preventiva é adotada para prevenir ou evitar impasses.
- As tarefas executam normalmente suas atividades, alocando e liberando recursos conforme suas necessidades.
- Quando ocorrer um **impasse**, o sistema o detecta, determina quais as tarefas e os recursos envolvidos e toma medidas para desfazê-lo.

Detecção e resolução de impasses

Detecção:

- Inspeção do grafo de alocação de recursos;
- O grafo é atualizado a cada alocação ou liberação de recurso;

$\Rightarrow$ Algoritmos de busca de ciclos em grafos têm custo computacional elevado.

Detecção e resolução de impasses

Resolução:

Eliminar tarefas : tarefas envolvidas no impasse são eliminadas, liberando seus recursos para que as demais tarefas possam prosseguir.

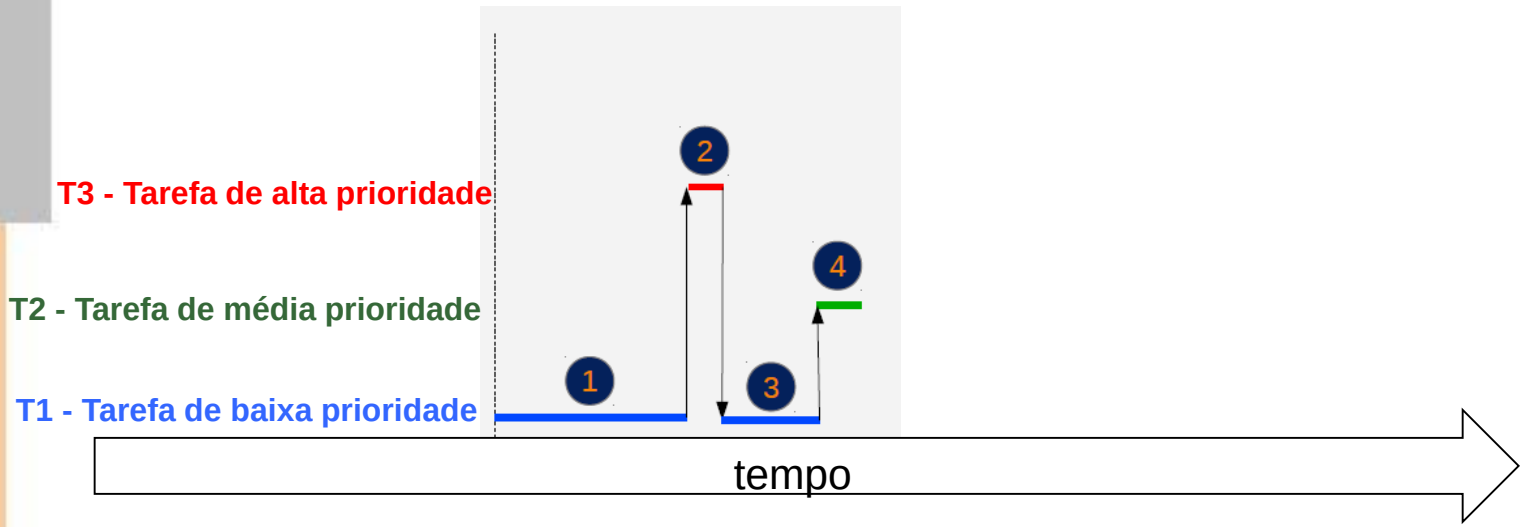
Retroceder tarefas : uma ou mais tarefas envolvidas no impasse têm sua execução parcialmente desfeita, de forma a fazer o sistema retornar a um estado seguro anterior ao impasse.

Detecção e resolução de impasses

Esta solução é geralmente implementada em grandes computadores *mainframe*, especialmente **sistemas em lote** onde eliminar e reiniciar um processo é aceitável.

Inversão de prioridade

Inversão de prioridade



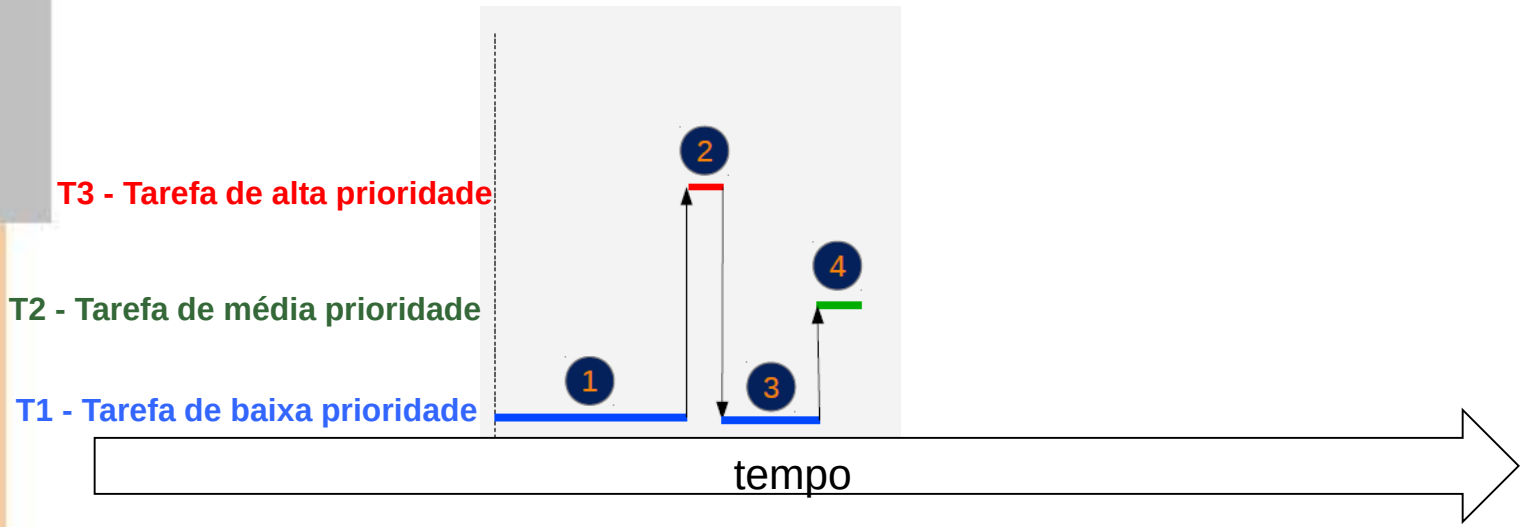
1) T1 detém o mutex e é interrompida para a execução da tarefa T3.

2) T3 tenta usar o mutex, mas dorme porque o mutex esta com T1.

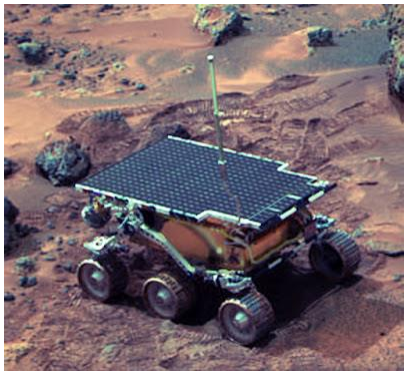
3) T1 retorna à execução, e antes de liberar o mutex é interrompida por T2, que tem maior prioridade que T1.

4) T2 é executada, e enquanto isso, T3, que tem maior prioridade, continua esperando!

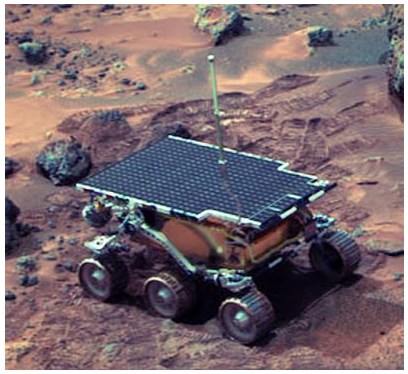
Inversão de prioridade

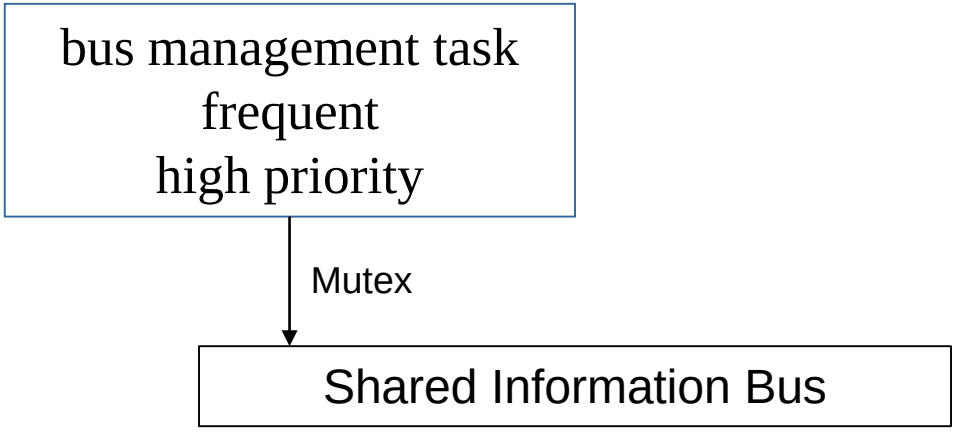


http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/mars_pathfinder.html

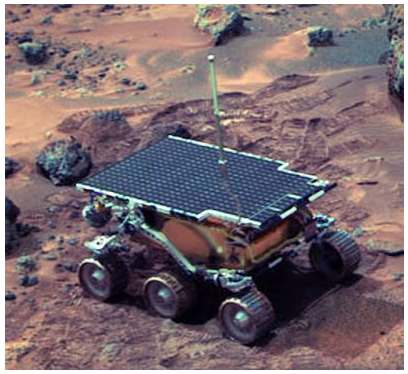


Inversão de prioridade





Inversão de prioridade

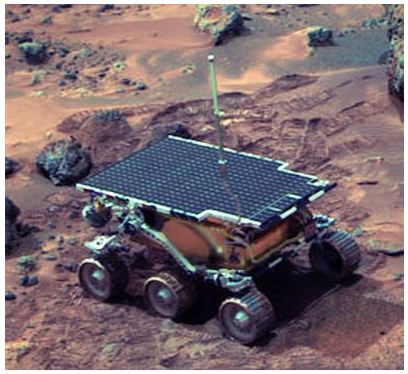


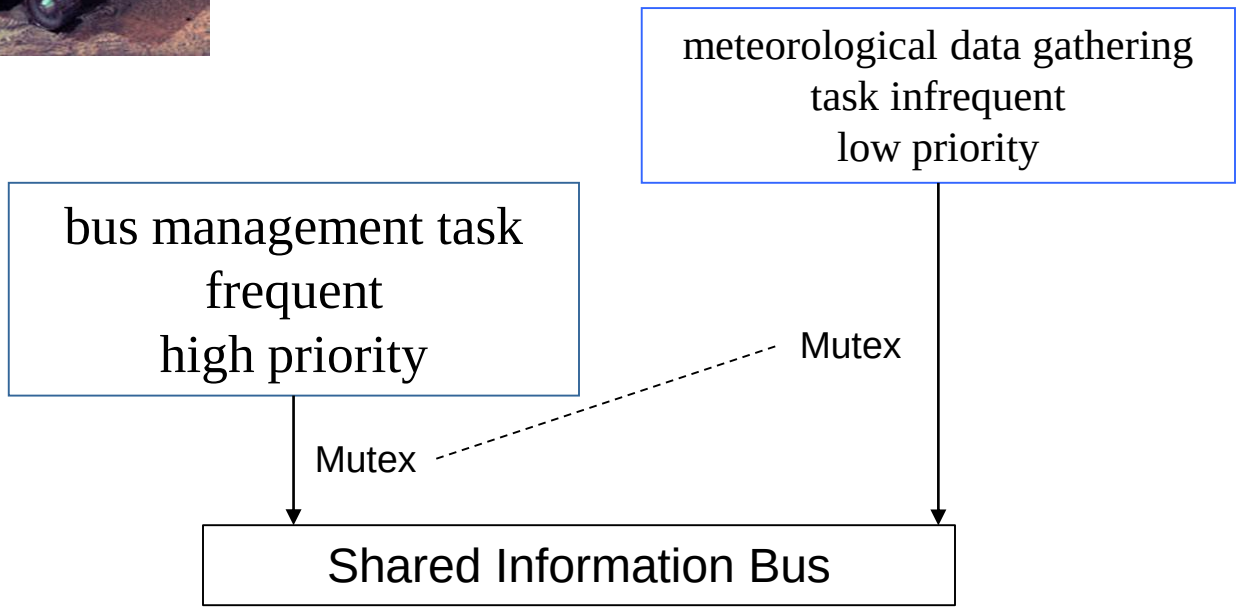
meteorological data gathering
task infrequent
low priority

Mutex

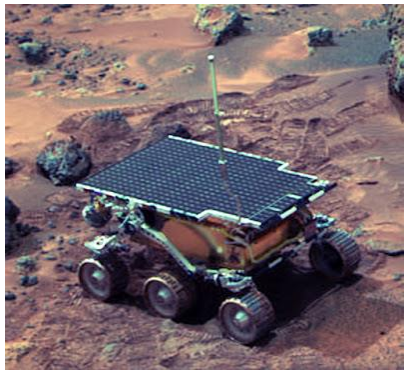
Shared Information Bus

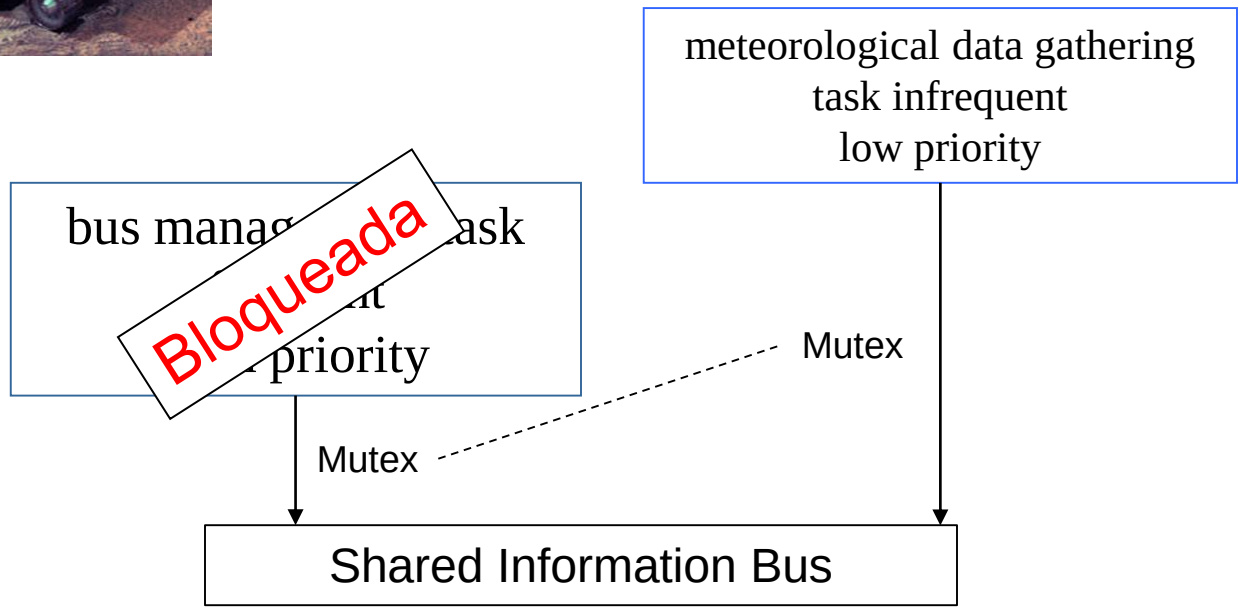
Inversão de prioridade



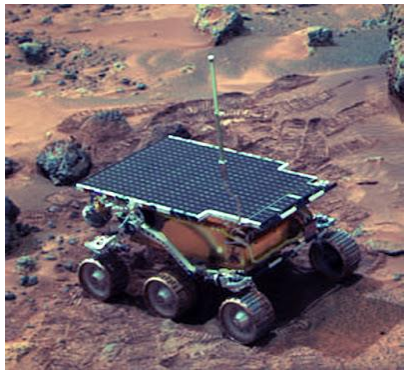


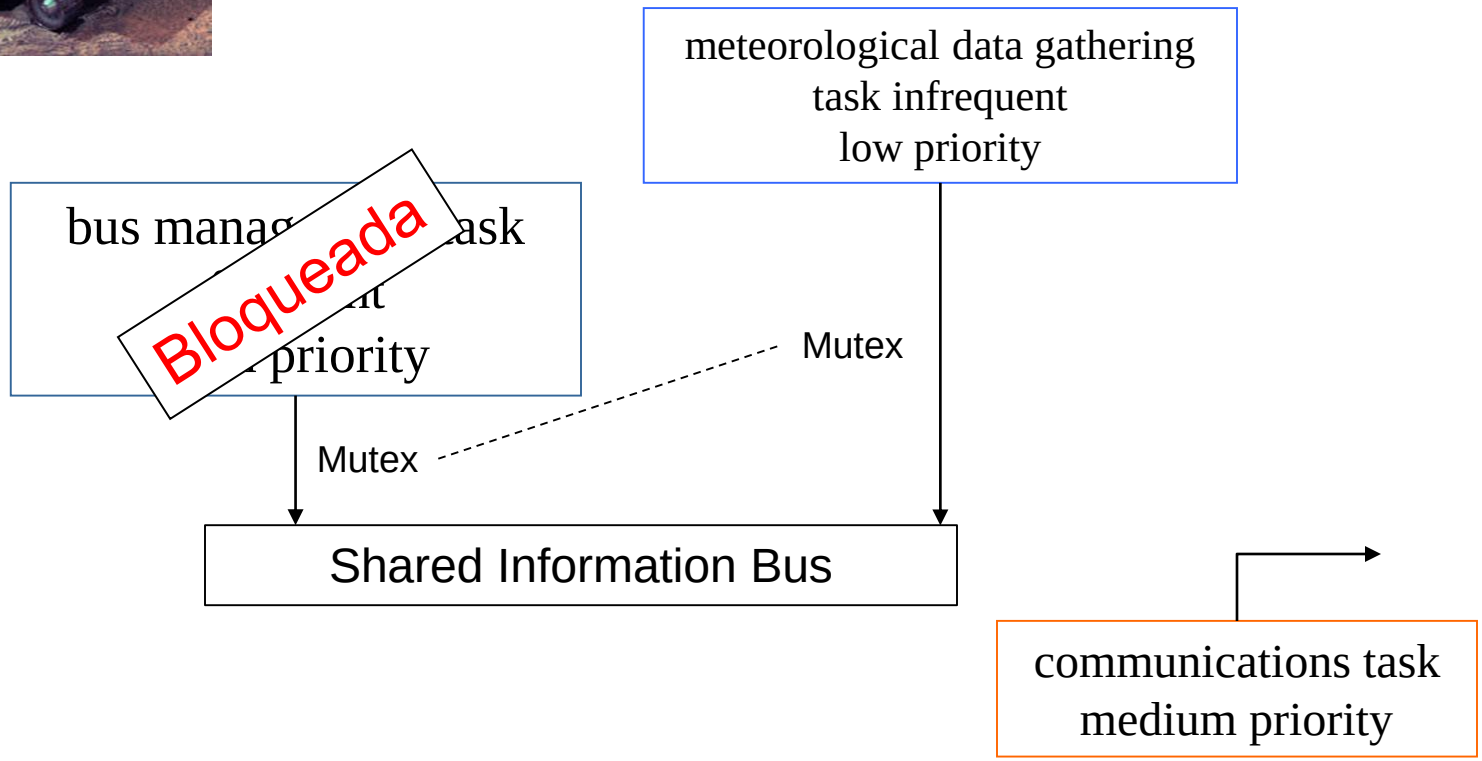
Inversão de prioridade



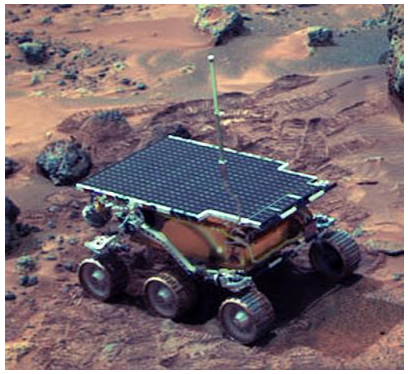


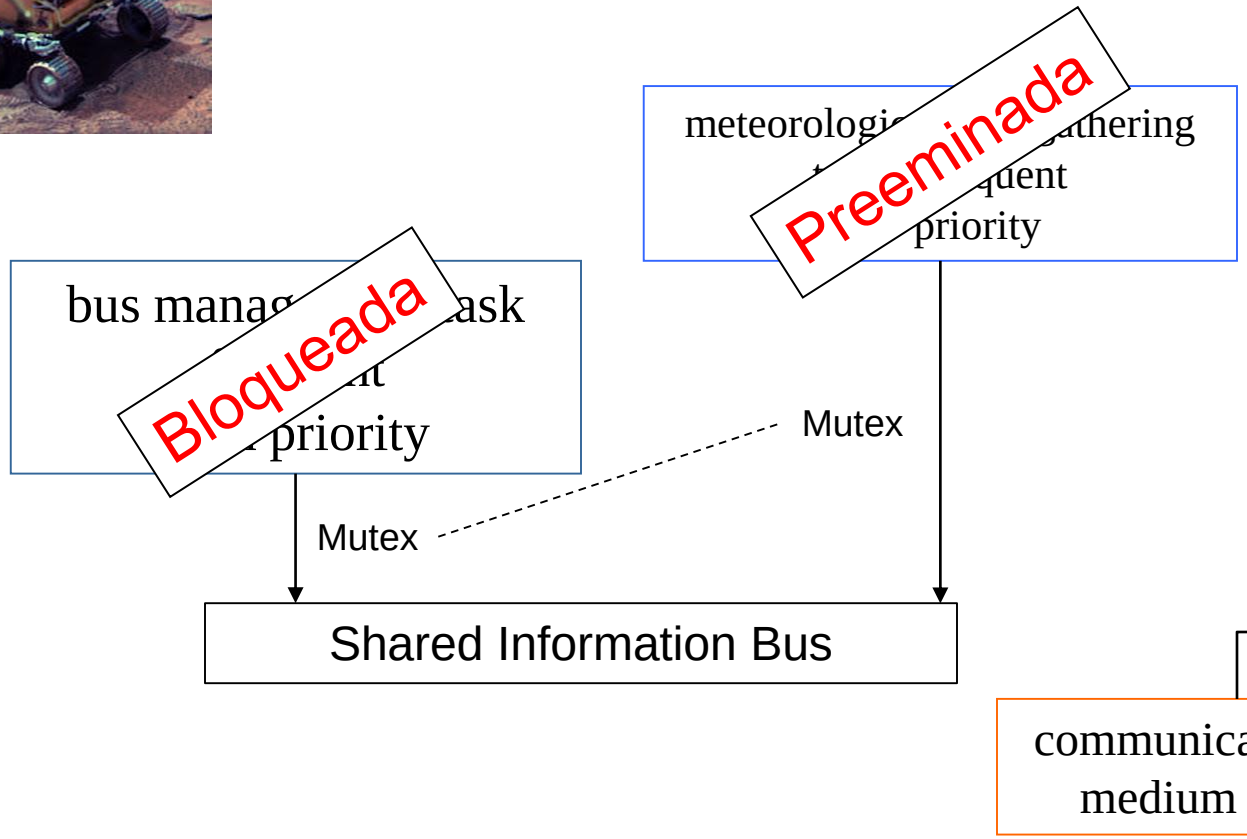
Inversão de prioridade



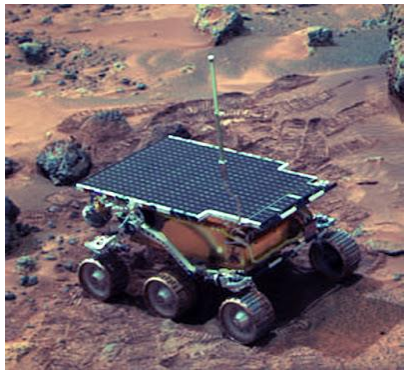


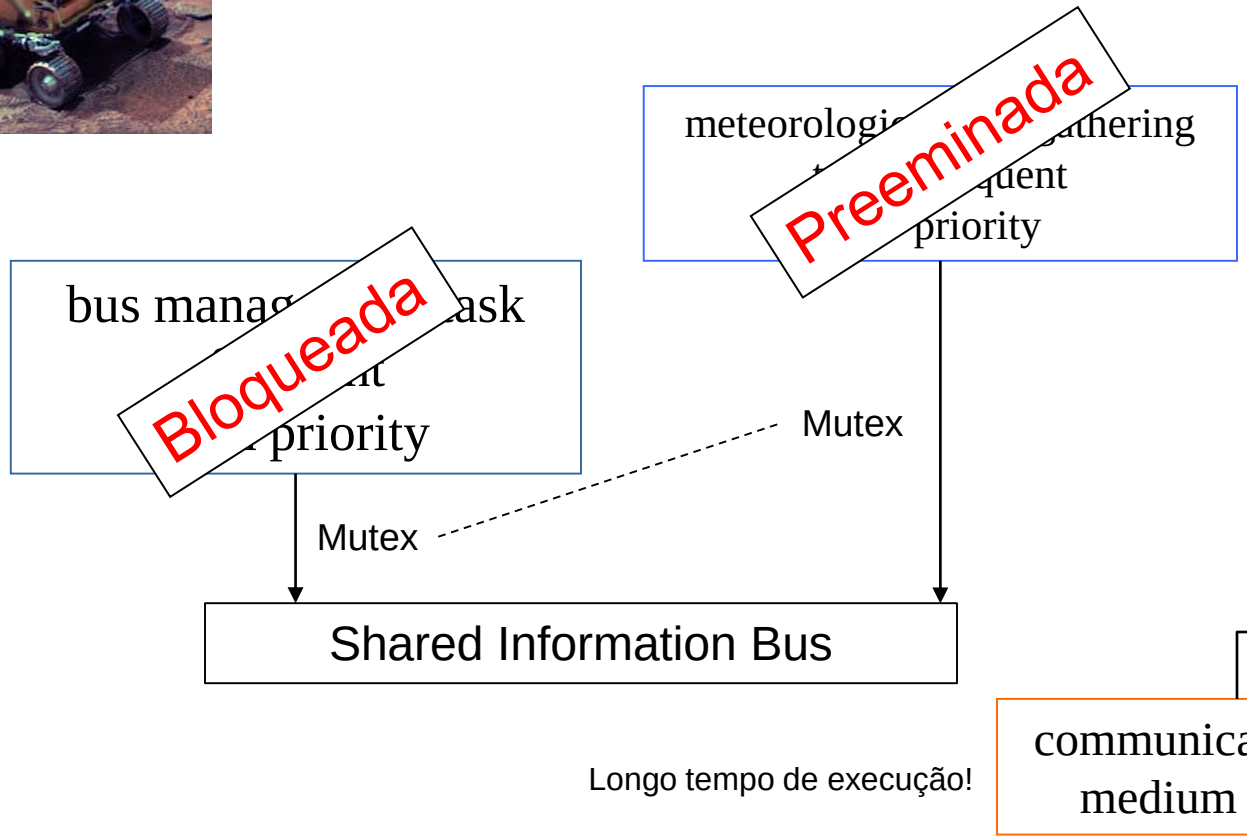
Inversão de prioridade



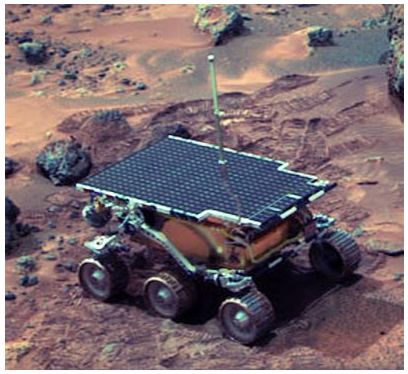


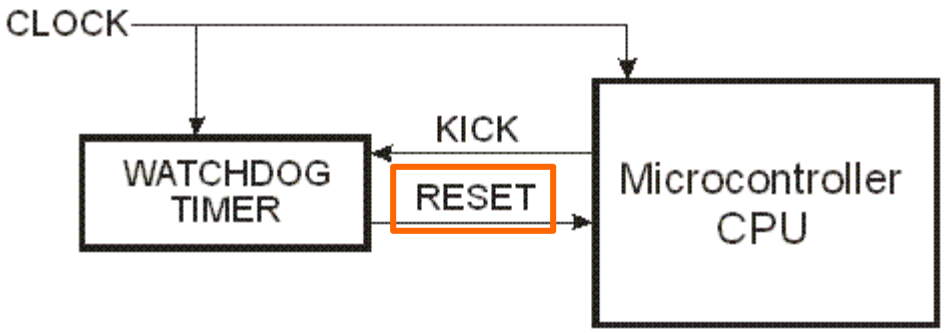
Inversão de prioridade



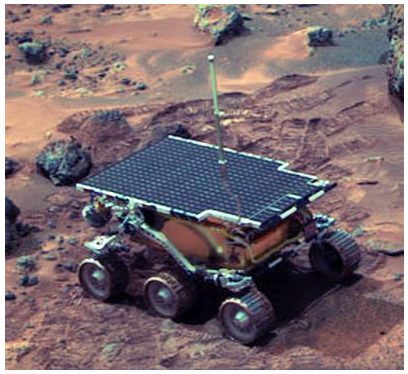


Inversão de prioridade





Inversão de prioridade



Exemplo fictício com WDT kicks:

```
while(1)
{
    //wait for a button press
    if(!digitalRead(BUTTON)){

        //turn on Flash Power
        digitalWrite(FLASHPWR, HIGH);

        //run the flash sequence
        runSequence();

        //loop in here for a while and run more sequences when requested
        //this lets the flash stay on for 2 minutes while still running sequences
        for(int i =0; i<2400; i++){
            if(!digitalRead(BUTTON)){
                runSequence();
                i=0;
            }
            _delay_ms(50);
            wdt_reset(); //reset watchdog
        }

        //if here, it's been 2 minutes sine the last button press
        //shut off flash unit
        digitalWrite(FLASHPWR, LOW);
    }

    _delay_ms(50);
    wdt_reset(); //reset watchdog
}
```