SISTEMAS OPERACIONAIS
AULA 10 – DEADLOCK E STARVATION

Prof.ª Sandra Cossul, Ma.

DEADLOCKS

DEADLOCK - INTRODUÇÃO

- **Deadlocks –** um dos principais problemas dos programas concorrentes

- Um conjunto de N processos está em **deadlock** quando cada um dos N processos está **bloqueado** à espera de um evento que somente pode ser causado por um dos N processos do conjunto.
  - Essa situação só pode ser alterada por uma iniciativa que parta de um processo fora do conjunto dos N processos!
  - Não existe uma solução eficiente!

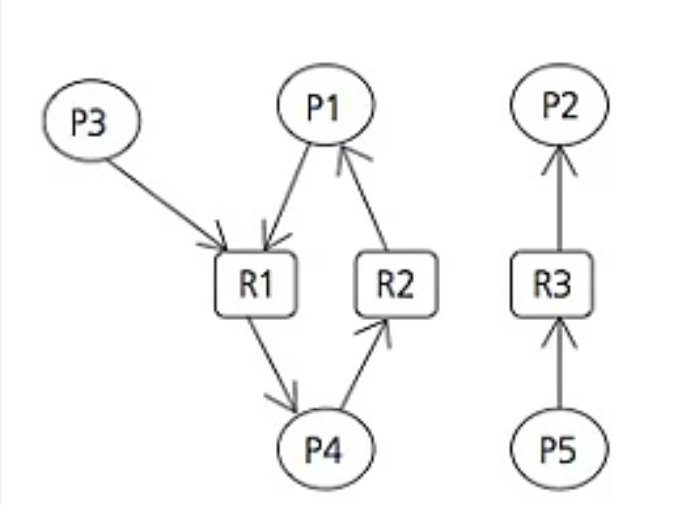
DEADLOCK - INTRODUÇÃO

- **Para cada recurso compartilhado** → semáforo ou mecanismo equivalente (suspender um processo enquanto outros acessam o recurso)
- Processos solicitam e aguardam a liberação de cada semáforo para poder acessar o recurso correspondente

- **Situações de impasse (deadlock)** → processos envolvidos ficam bloqueados aguardando a liberação de semáforos e nada mais acontece....

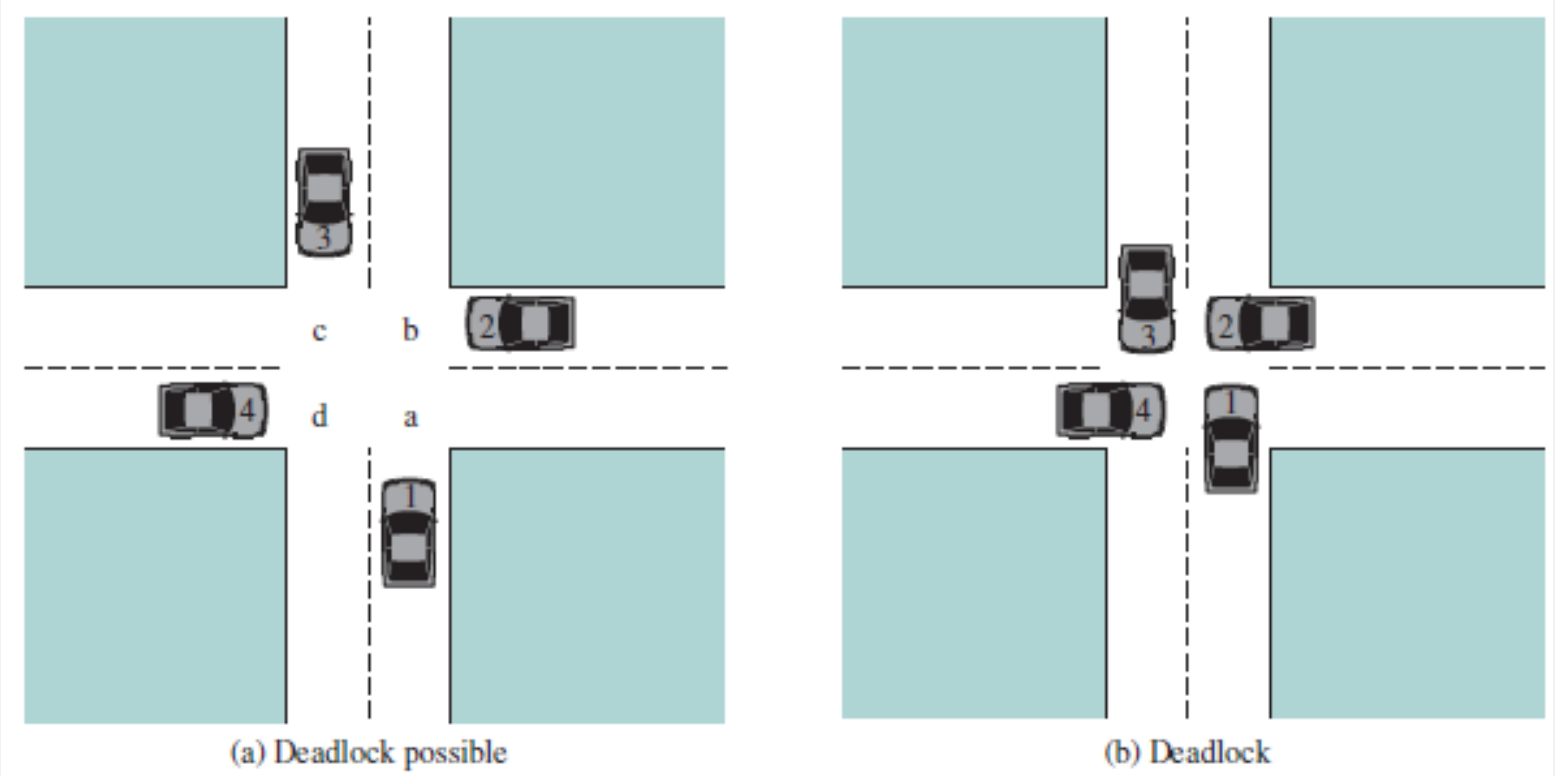
DEADLOCK – DEMONSTRAÇÃO 1

- **P5** bloqueado à espera do recurso R3
  - Não está em deadlock, pois o processo **P2** não esta bloqueado
  - P2 vai executar até o fim e liberar o recurso R3, e então P5 pode executar



- **P4** bloqueado à espera do recurso R2, ocupado pelo processo P1
- **P1 e P3** bloqueados à espera do recurso **R1**, ocupado pelo processo P4
- **P1, P3 e P4** estão em deadlock

DEADLOCK – DEMONSTRAÇÃO 2



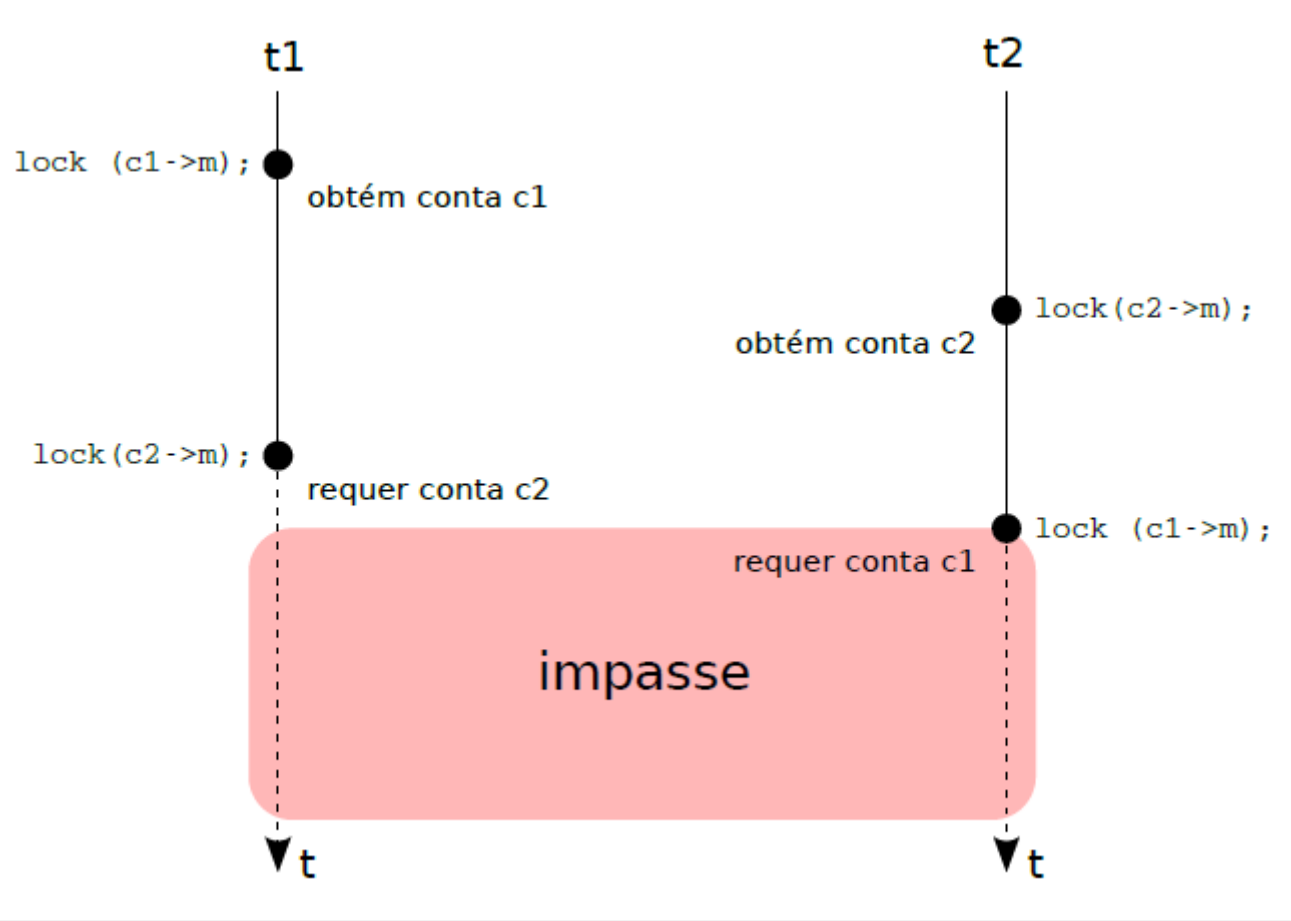
DEADLOCK – DEMONSTRAÇÃO 3

- Caso dois clientes do banco (representados por duas tarefas t1 e t2) resolvam fazer simultaneamente operações de transferência entre suas contas:

- (t1 transfere um valor v1 de c1 para c2)
- (t2 transfere um valor v2 de c2 para c1)

- Pode acontecer impasse (deadlock!)

DEADLOCK – DEMONSTRAÇÃO 3



DEADLOCK – TIPOS DE RECURSOS

- **Recursos reutilizáveis**
  - Pode ser usado com segurança por apenas um processo por vez e não é esgotado;
  - **Ex.:** processadores, canais de E/S, memória principal e secundária, dispositivos e estuturas de dados (como arquivos, bases de dados e semáforos)

- **Recursos consumíveis**
  - Pode ser criado (produzido) e destruído (consumido)
  - Quando o recurso é adquirido por um processo, ele é esgotado (não existe mais)
  - **Ex.:** interrupções, sinais, mensagens e informações em buffers de E/S.

DEADLOCK – CONDIÇÕES DE OCORRÊNCIA

1. **Existência de recursos que** precisam ser acessados **de forma exclusiva** (*exclusão mútua)*

2. Possibilidade de processos **manterem recursos alocados enquanto esperam** por **recursos adicionais** (*posse e espera)*

3. Necessidade de os **recursos serem liberados** pelos **próprios processos** que os estão utilizando *(não-preempção)*

4. **Possibilidade da formação de uma espera circular** do tipo: o processo P1 espera pelo recurso R1 que está com o processo P2, que espera pelo recurso R2 que está com o processo P3, etc., e PN espera pelo recurso RN que está com o processo P1 (*espera circular)*.

DEADLOCK – CONDIÇÕES DE OCORRÊNCIA

- As quatro condições são **necessárias** para a formação de deadlocks
- Se uma delas não for verificada, não existem deadlocks no sistema.

- Por outro lado, não são condições **suficientes** para a existência de deadlock
- Ou seja, a verificação dessas 4 condições não garante a presença de um deadlock
- Essas condições somente são suficiente se existir **apenas uma instância de cada tipo de recurso**

GRAFOS DE ALOCAÇÃO DE RECURSOS

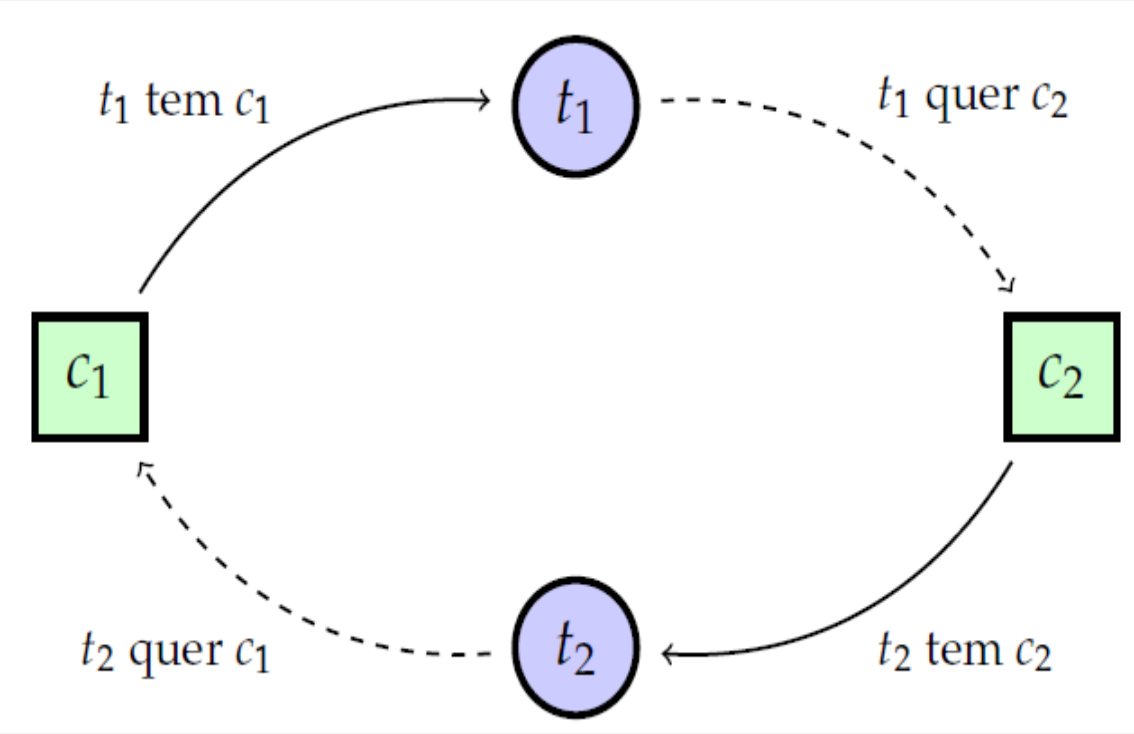
- É possível **representar graficamente** a alocação de recursos entre as tarefas de um sistema concorrente.
- Visão mais clara da distribuição dos recursos
- Detecção visual da presença de esperas circulares que podem caracterizar deadlocks

- **Processos** representados por **círculos** e **Recursos** por **retângulos**
- **Seta normal** – recurso alocado **Seta pontilhada** – requisição de recurso

GRAFOS DE ALOCAÇÃO DE RECURSOS

- **Grafo de alocação de recursos da situação de deadlock ocorrida no exemplo da transferência de valores entre contas bancárias.**

- Como há um só recurso de cada tipo (apenas uma conta c1 e uma conta c2), as quatro condições necessárias se mostram também suficientes para caracterizar um impasse.



GRAFOS DE ALOCAÇÃO DE RECURSOS

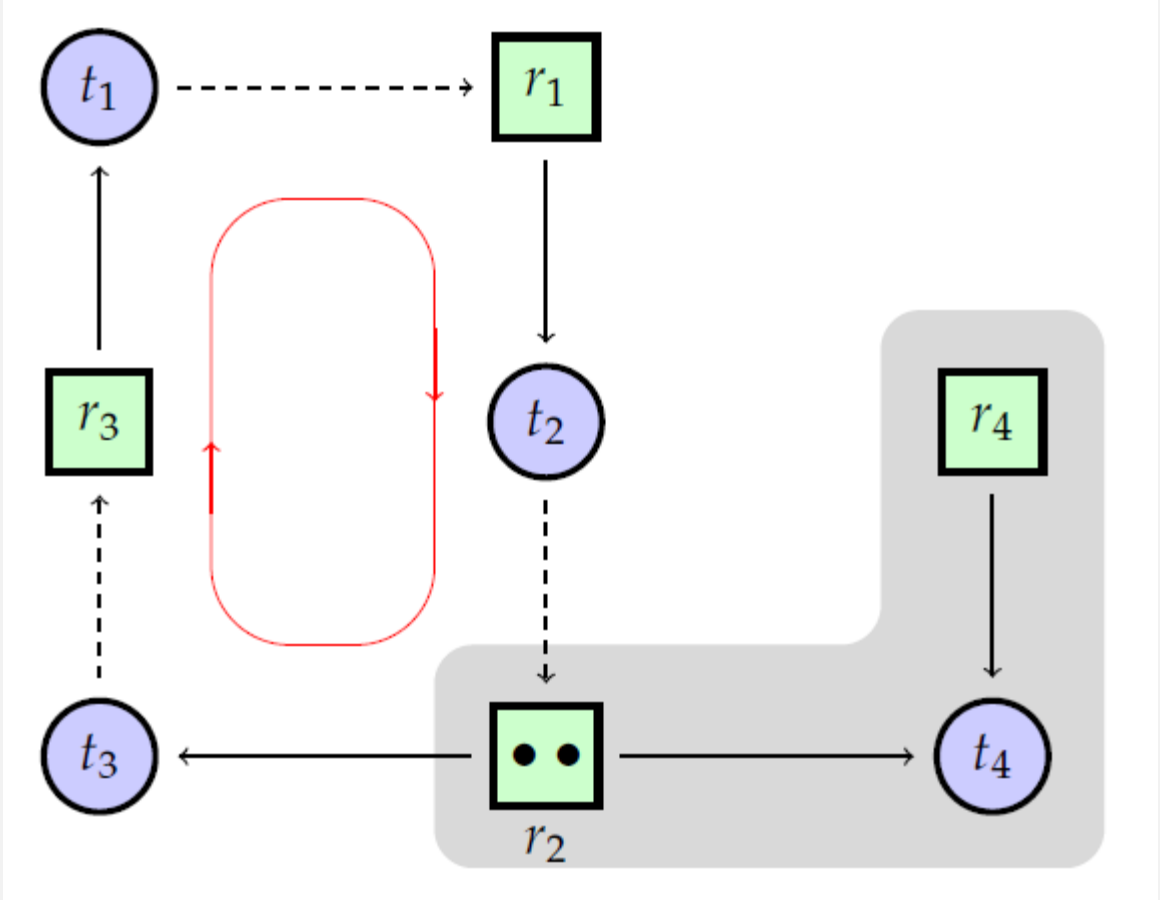
- Caso o sistema tiver **múltiplas instâncias do mesmo recurso**, cada um pode ser **alocado** de forma **independente**
  - 2 impressoras instaladas

- No grafo de alocação de recursos, a existência de múltiplas instâncias de um recurso é representada através de "pontos" dentro dos retângulos.

GRAFOS DE ALOCAÇÃO DE RECURSOS

- **Grafo de alocação de recursos com múltiplas instâncias**

- Não representa um impasse, porque a qualquer momento a tarefa t4 (que não está esperando recursos) pode liberar uma instância do recurso r2, solicitado por t2, permitindo atender a demanda de t2 e desfazendo o ciclo.



TÉCNICAS DE TRATAMENTO DE DEADLOCKS

TÉCNICAS DE TRATAMENTO DE DEADLOCKS

- Deadlocks causam o bloqueio de processos (threads) que detém recursos → **paralisação gradativa de todos processos (ou threads) que dependem dos recursos envolvidos** → paralisação de todo o sistema

- Embora o risco de deadlocks seja uma questão importante, os sistemas operacionais de mercado (Windows, Linux, MacOS, etc.) adotam a solução mais simples: **ignorar o risco**, na maioria das situações.
  - Devido ao **alto custo computacional e a dependência lógica das aplicações envolvidas**
- **Gestão de impasses** fica por conta dos desenvolvedores de aplicações

DEADLOCK – ABORDAGENS

1. **Prevenção de Deadlock**
   - não permitir que aconteça uma das condições de ocorrência de deadlock

2. **Impedimento de Deadlock**
   - não atender um pedido de alocação de recurso se essa alocação pode levar a deadlock

3. **Detecção e resolução de Deadlock**
   - atender os pedidos de alocação de recurso sempre que possível, mas periodicamente verificar a presença de deadlock e tomar uma ação para recuperação

1. PREVENÇÃO DE DEADLOCK

- **Negação de uma das quatro condições necessárias:**

  - **Exclusão mútua**
  - não pode ser desabilitada, se for preciso, deve ser concedida pelo SO
  - Como garantir a integridade de recursos compartilhados sem usar mecanismos de exclusão mútua?

1. PREVENÇÃO DE DEADLOCK

- **Negação de uma das quatro condições necessárias:**

  - **Posse e espera**
  - **1)** Os processos usam apenas um recurso por vez, solicitando-o e liberando-o logo após o uso
  - **2)** Um processo somente executa quando todos os seus recursos solicitados estão disponíveis (pode levar os processos a reter os recursos por mais tempo que o necessário, diminuindo o desempenho do sistema)
  - **3)** Associar um prazo às solicitação de recursos (ao solicitar um recurso, o processo define um tempo máximo de espera por ele; caso o prazo termine, o processo pode tentar novamente ou desistir, liberando os demais recursos que detém.

1. PREVENÇÃO DE DEADLOCK

- **Negação de uma das quatro condições necessárias**

  - **Não preempção –** um processo deve liberar seu primeiro recurso se requerer um segundo e este não for concedido. Ou o SO pode exigir que um processo libere um recurso em favor de outro.
  - Forçar o processo a liberar o recurso (por exemplo, o processador nas trocas de contexto)
  - Pode não funcionar em recursos como arquivos, porque a preempção viola a exclusão mútua e pode provocar inconsistências no estado do recurso.

1. PREVENÇÃO DE DEADLOCK

- **Negação de uma das quatro condições necessárias**

  - **Espera circular –** pode ser prevenida definindo uma ordenação linear dos tipos de recursos.
  - Forçar os processos a solicitar os recursos obedecendo essa ordenação

  - No exemplo da transferência de fundos, o número de conta bancária poderia definir uma ordem global (c1 < c2, por exemplo). Assim, todas as tarefas deveriam solicitar primeiro o acesso à conta mais antiga e depois à mais recente (ou vice-versa, mas sempre na mesma ordem para todas as tarefas).
  - Com isso, elimina-se a possibilidade de deadlocks.

2. IMPEDIMENTO DE DEADLOCK

- Tenta **evitar** que o ponto de deadlock seja atingido
- Uma decisão é tomada **dinamicamente** analisando se a solicitação de recursos atual (se concedida) vai levar a um deadlock
- Conhecimento dos pedidos de alocação de recursos **futuros.**

- **Duas abordagens:**
  - Não iniciar um processo se a demanda pode levar a deadlock
  - Não conceder um recurso adicional a um processo se a alocação pode levar a deadlock

2. IMPEDIMENTO DE DEADLOCK

- **Restrições:**
  - Os requisitos máximos de recursos para cada processo devem ser indicados com antecedência
  - Os processos considerados devem ser independentes (a ordem que eles executam não influencia nos requisitos de sincronização)
  - Deve haver um número fixo de recursos para alocar
  - Nenhum processo pode sair enquanto retém recursos

- Devido a essas restrições, as técnicas de impedimento são pouco utilizadas na prática.

3. DETECÇÃO E RESOLUÇÃO DE DEADLOCK

- **Pedidos de alocação de recursos são concedidos aos processos assim que possível!**

- Periodicamente, o SO executa um algoritmo que permite detectar a condição de espera circular

- Quando ocorrer um deadlock, o sistema deve detectá-lo, determinar quais os processos e recursos envolvidos e tomar medidas para desfazê-lo.

3. DETECÇÃO E RESOLUÇÃO DE DEADLOCK

- **Detecção de deadlocks**
  - Inspeção do grafo de alocação de recursos (que deve ser mantido pelo sistema e atualizado a cada alocação e liberação de recurso)

  - Problemas:
    - Custo de manutenção contínua do grafo de alocação de recursos.
    - Custo da análise do grafo - algoritmos de busca de ciclos em grafos tem custo computacional elevado (pode prejudicar o desempenho do sistema).
    - Se executar esporadicamente, deadlocks podem demorar muito para serem detectados, o que também é ruim para o desempenho do sistema.

3. DETECÇÃO E RESOLUÇÃO DE DEADLOCK

- Quando detectado, é necessário alguma abordagem para **resolução:**
  1. **Terminar todos os processos em deadlock** (solução mais comum nos SOs)
  2. **Fazer backup de todos os processos em deadlock para algum ponto de verificação anterior e reiniciar todos os processos** (construir mecanismos de reversão e reinicialização e problema pode acontecer novamente)
  3. **Terminar sucessivamente os processos em deadlock até que o deadlock não exista mais** (deve estabelecer uma ordem de seleção para terminar os procesos, com algum critério de custo mínimo)
  4. **Preempção sucessiva de recursos até que o deadlock não exista mais** (também utilizar algum critério de seleção e um processo com recurso impedido deve ser revertido para um ponto anterior)

3. DETECÇÃO E RESOLUÇÃO DE DEADLOCK

- No caso dos critérios de seleção, dos itens 3 e 4, estes podem ser:
  - **Menor tempo consumido de uso de processador até o momento**
  - **Menor quantidade de dados de saída produzidos até o momento**
  - **Maior estimativa de tempo restante**
  - **Menor número de recursos alocados até o momento**
  - **Prioridade mais baixa**

3. DETECÇÃO E RESOLUÇÃO DE DEADLOCK

- Abordagem interessante, mas pouco usada fora de situações muito específicas
  - **Custo de detecção elevado**
  - **Alternativas de resolução envolvem perder processsos ou parte das execuções já realizadas**

- Exemplo de aplicação: gerenciamento de transações em sistemas de bancos de dados
  - Mecanismos para criar checkpoints dos registros envolvidos antes da transação
  - Para efetuar *rollback* da mesma em caso de deadlock

STARVATION

STARVATION

- **Starvation** – inanição (morre de fome)

- Ocorre quando um processo nunca é executado, pois processos de prioridade maior sempre o impedem de ser executado

- A execução de diversos processos simultâneos deve seguir uma regra de escalonamento adequada
  - Quando o escalonamento não é feito adequadamente, pode haver starvation

STARVATION

- **Starvation** é DIFERENTE de **Deadlock**
  - no deadlock os processos permanecem bloqueados

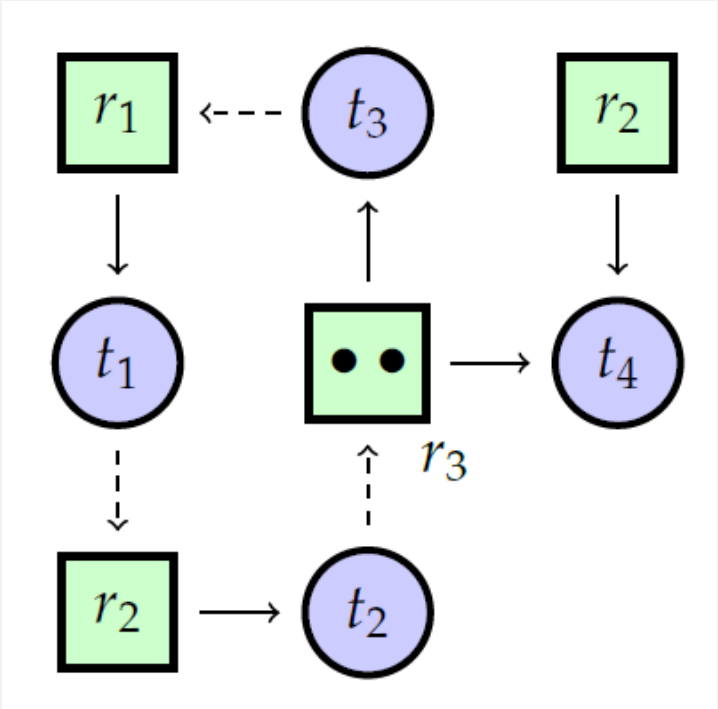
- **Solução:**
  - Delegar um tempo máximo de espera

RESUMO

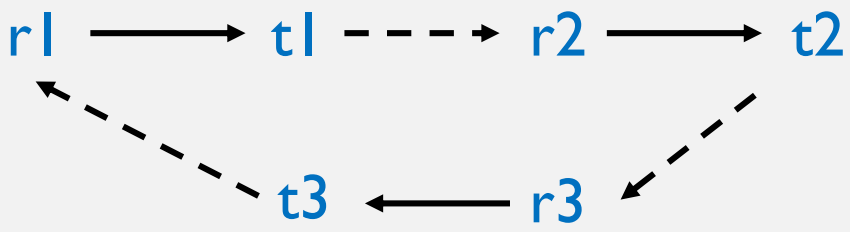
- **Deadlock** é um **bloqueio** de um conjunto de processos
- Bloqueio é **permanente**, a não ser que o SO tome alguma ação drástica como **terminar** um ou mais processos ou forçar o processo a **retroceder**
- 3 abordagens para lidar com deadlock: **prevenir**, **impedir** ou **detectar e resolver**!
- Algoritmos de detecção de deadlock podem avaliar processos e recursos em **tempo real** e determinar se um conjunto de processos está ou vai entrar em um estado de deadlock
- Ao acontecer um deadlock, o sistema pode tentar se recuperar **terminando um ou mais processos** ou **liberando recursos** alocados a um processo em deadlock.

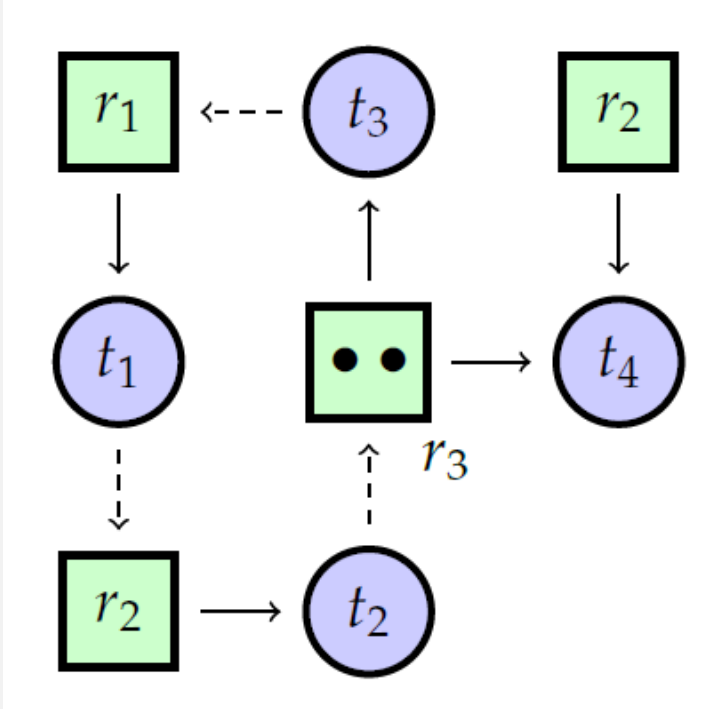
EXERCÍCIO 1

- No grafo de alocação de recursos da figura a seguir, indique o(s) ciclo(s) onde existe um deadlock:



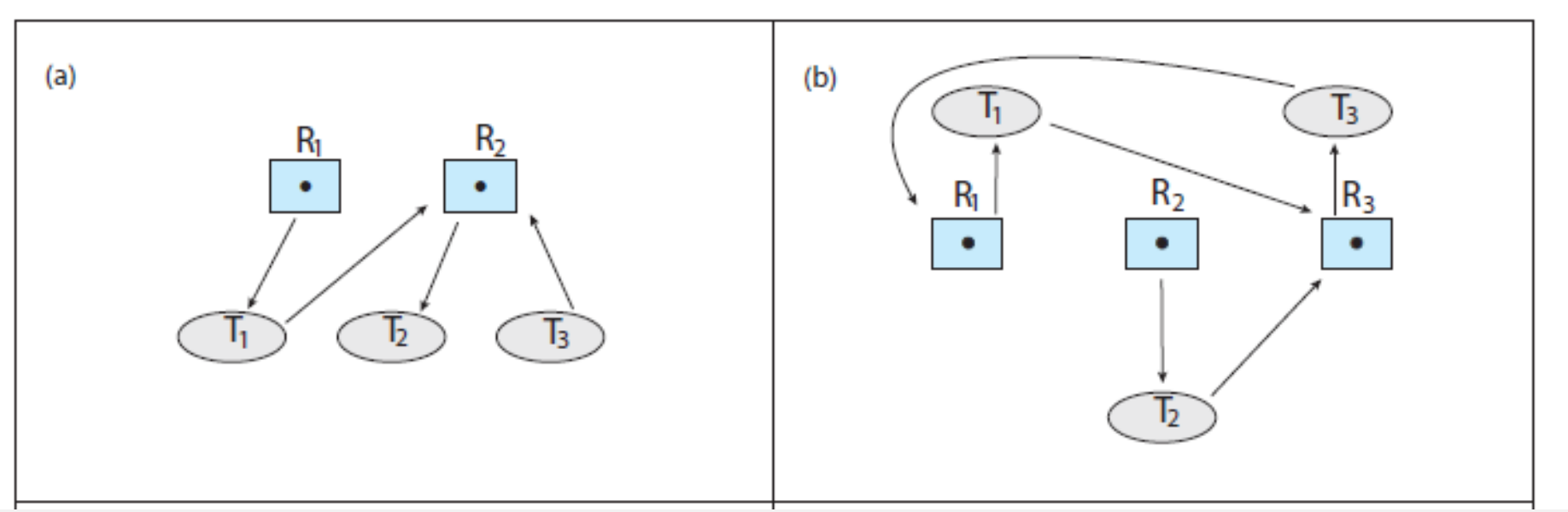
EXERCÍCIO 1 - RESOLUÇÃO





EXERCÍCIO 2

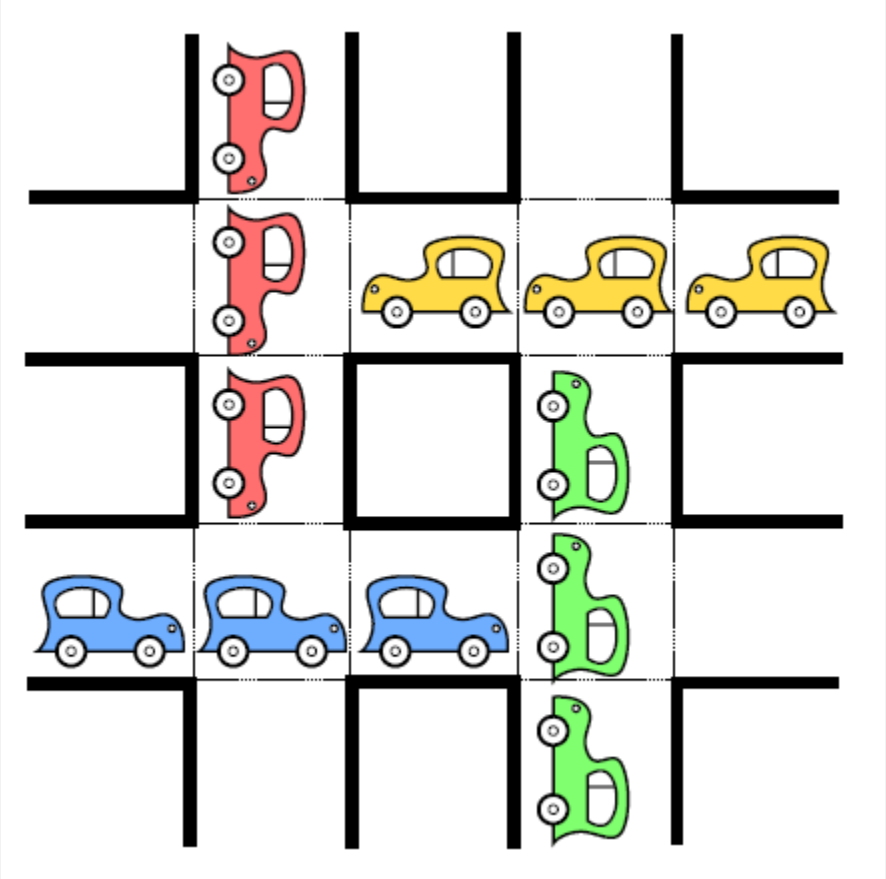
- No grafo de alocação de recursos da figura a seguir, verifique se acontece deadlock. Se não, indique a ordem de execução das tarefas e, caso ocorrer ocorrer deadlock, indique o(s) ciclo(s) onde está ocorrendo.



EXERCÍCIO 2

- A figura a seguir representa uma situação de impasse em um cruzamento de trânsito. Todas as ruas têm largura para um carro e sentido único.
- Mostre que as quatro condições necessárias para a ocorrência de impasses estão presentes nessa situação.
- Em seguida, defina uma regra simples a ser seguida por cada carro para evitar essa situação; regras envolvendo algum tipo de informação centralizada não devem ser usadas.

EXERCÍCIO 2



EXERCÍCIO 2 - RESOLUÇÃO

- **Exclusão mútua –** o acesso a rua deve ser exclusivo para cada carro (um por vez pode acessar a rua)
- **Posse e espera –** cada carro solicita o acesso a 2 ruas (sem liberar a primeira)
- **Não-preempção –** cada carro que detém o acesso a uma rua não perde o acesso de forma imprevista ou é retirado (só libera a rua quando quiser)
- **Espera circular –** cada carro aguarda um recurso retido pelo outro carro e assim sucessivamente.

PRÓXIMA AULA

- Gestão de Processos:
  - Escalonamento de processos

BIBLIOGRAFIA

- Tanenbaum, A. S. **Sistemas Operacionais Modernos.** Pearson Prentice Hall. 3rd Ed., 2009.
- Silberschatz, A; Galvin, P. B.; Gagne G.; **Fundamentos de Sistemas Operacionais.** LTC. 9th Ed., 2015.
- Stallings, W.; **Operating Systems: Internals and Design Principles**. Prentice Hall. 5th Ed., 2005.
- Oliveira, Rômulo, S. et al. **Sistemas Operacionais** - V11 - UFRGS. Disponível em: Minha Biblioteca, Grupo A, 2010.
- *baseado nos slides da Prof.ª Roberta Gomes (UFES) e do Prof. Felippe Fernandes da Silva