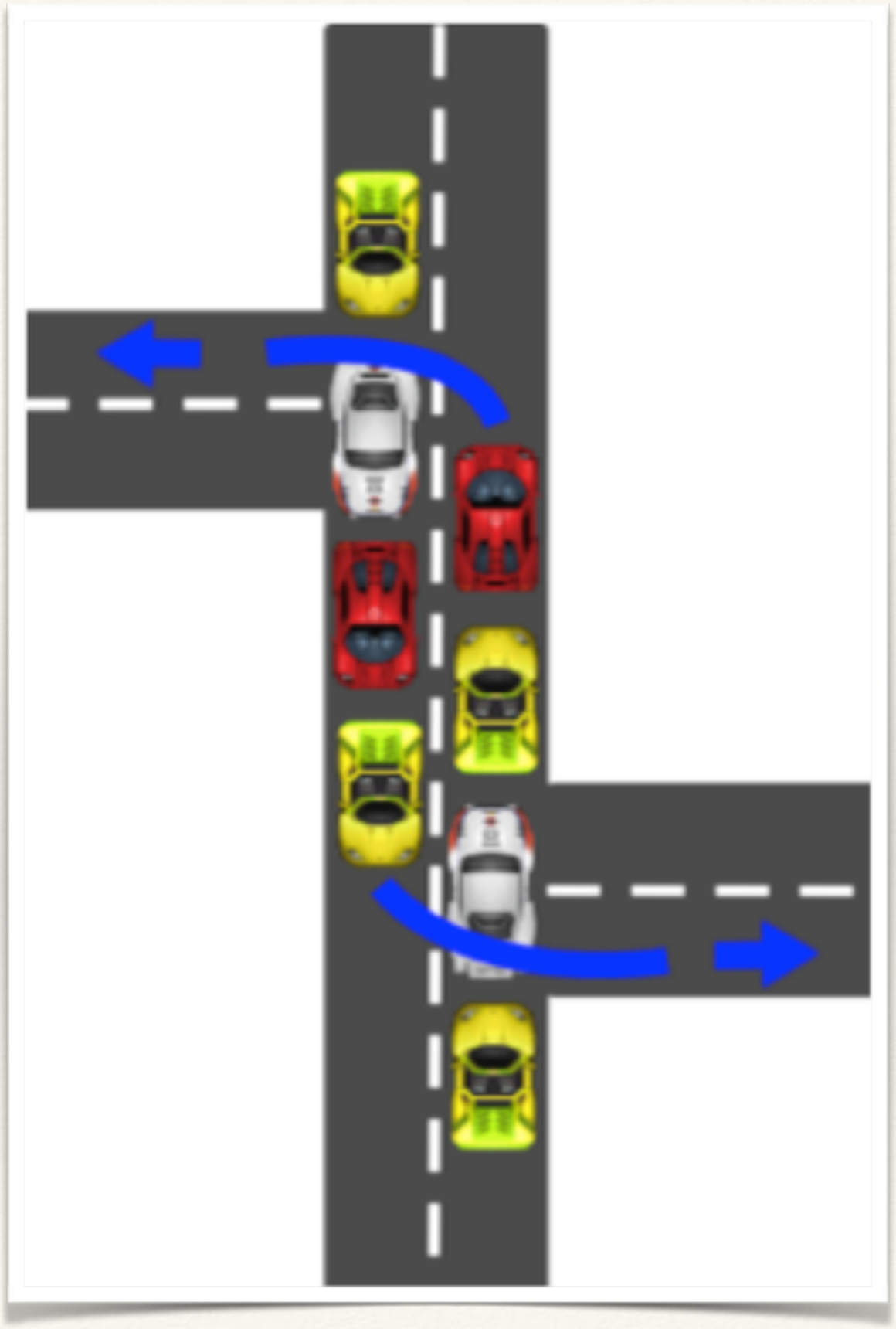
Prof. Ricardo Inácio Álvares e Silva

Deadlocks

Sistemas Operacionais



Rápida Revisão

- **Processos**: instância de programas carregados na memória, compartilhando tempos de execução das CPUs do sistema
- **Estados de processos**: novo, executando, pronto, bloqueado (esperando), terminado
- **Estado bloqueado**: quando um processo requer algum recurso do sistema para poder continuar executando, ele deve ficar em espera até obter este recurso
- **Condição de corrida**: quando o resultado de execução de dois ou mais processos depende da ordem de execução
- **Exclusão mútua**: conjunto de instruções em que apenas um processo pode executar por vez.
- **Sincronização**: técnicas para comunicação entre processos evitando condição de corrida entre eles.

Escopo da Aula

- Introdução a Deadlocks
  - Conceito
  - Recursos preemptáveis e não-preemptáveis
  - Condições necessárias
- Verificação de Deadlocks
  - Modelagem em grafos
- Reação a Deadlocks
  - Algoritmo de Ostrich
  - Recuperação de Deadlocks
  - Evitando Deadlocks
  - Prevenindo contra Deadlocks

Cenário de uso de computadores

- Considere um sistema com dois processos em execução, ambos possuem o objetivo de *scannear* uma imagem e copiá-la para o CD através da gravadora
- Porém, cada processo é um programa diferente:
  - O primeiro requisita uma imagem por vez do scanner, para em seguida copiá-las para o CD através do gravador
  - O segundo verifica se há um disco adequado no gravador, para somente então *scannear* as imagens e gravá-las no CD
- Ordem dos fatos:
  1. Primeiro programa requisita Scanner
  2. Segundo programa requisita gravador
  3. Primeiro programa requisita gravador, mas fica bloqueado
  4. Segundo programa requisita Scanner, mas fica bloqueado

VIDA DE PROGRAMADOR .COM.BR

#1609

- Se não houver intervenção, a situação não se resolverá nunca!
- Essa situação é denominada ***deadlock***
  - Em português, ***impasse***
- Deadlocks não estão limitados a processos e sistemas operacionais
  - pode ocorrer entre dois computadores compartilhando uma impressora
  - dois usuários acessando registros em um banco de dados

Recursos

- Deadlocks envolvem **recursos**, e a possibilidade de um único processo obter acesso exclusivo a ele, mesmo que temporariamente
- Um computador possui vários tipos de recursos, mas é possível distinguí-los em duas classes:
  - **Preemptáveis**: recursos que podem ser tomados de um processo sem efeitos colaterais
  - **Não-preemptáveis**: recursos que não podem ser tomados de um processo
- Quais as classes dos seguintes recursos?
  - Memória principal
  - CPU
  - Gravador de DVD
  - Impressora

- *Deadlocks* acometem recursos não-preemptáveis, pois a sua forma de utilização envolve os seguintes passos:
  1. Requisitar o recurso
  2. Utilizar o recurso
  3. Liberar o recurso
- Enquanto o recurso estiver sendo utilizado por algum processo, outros processos requisitantes
  - terão acesso negado (normalmente entram em loop)
  - ou, serão bloqueados

- A requisição de um determinado recurso ocorre de maneira similar, senão idêntica, à utilização de **semáforos**

```
typedef int semaphore;
semaphore resource_1;

void process_A(void) {
    down(&resource_1);
    use_resource_1();
    up(&resource_1);
}
```

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    usar_ambos_recursos();
    up(&resource_2);
    up(&resource_1);
}
```

- A ordem das requisições importa
- Compare os dois casos seguintes

```
void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    usar_ambos_recursos();
    up(&resource_2);
    up(&resource_1);
}

void process_B(void) {
    down(&resource_1);
    down(&resource_2);
    usar_ambos_recursos();
    up(&resource_2);
    up(&resource_1);
}
```

```
void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    usar_ambos_recursos();
    up(&resource_2);
    up(&resource_1);
}

void process_B(void) {
    down(&resource_2);
    down(&resource_1);
    usar_ambos_recursos();
    up(&resource_1);
    up(&resource_2);
}
```

Definição de Deadlock

"um conjunto de processos está em deadlock se cada um destes processos está em espera por um evento que apenas um outro deste mesmo conjunto pode realizar."

- Na maioria dos casos, o evento a que a definição se refere é a liberação de algum recurso

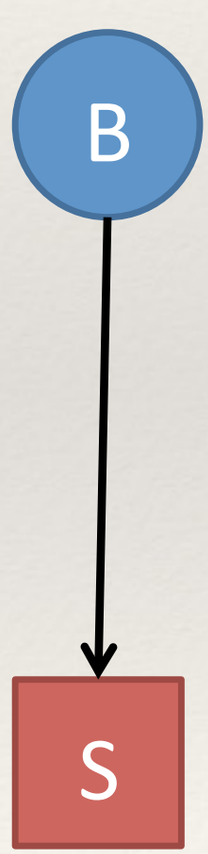
- Condições mínimas necessárias à existência de deadlocks:

  1. Exclusão mútua para acesso aos recursos
  2. Acesso e espera, um processo pode requisitar acesso a mais recursos, mesmo em posse de outros
  3. Recursos não-preemptáveis, não podem ser retirados a força de um processo
  4. Condição circular de espera

Modelagem de Deadlocks

- A utilização de recursos por processos pode ser demonstrada graficamente (e formalmente), através da seguinte notação:

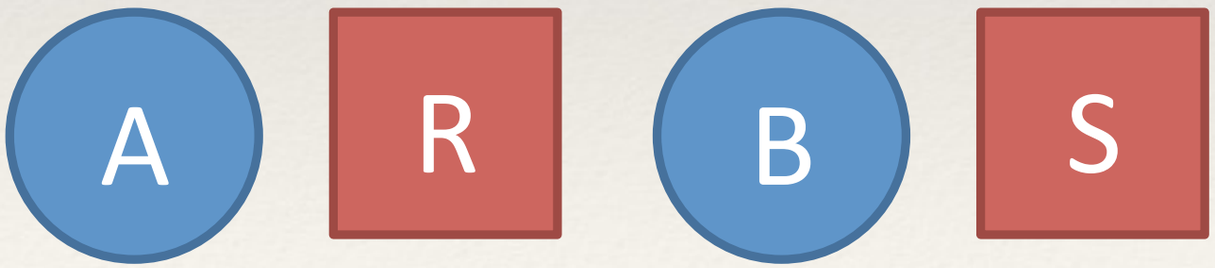
Processo A segurando recurso R



Processo B requisitando recurso S

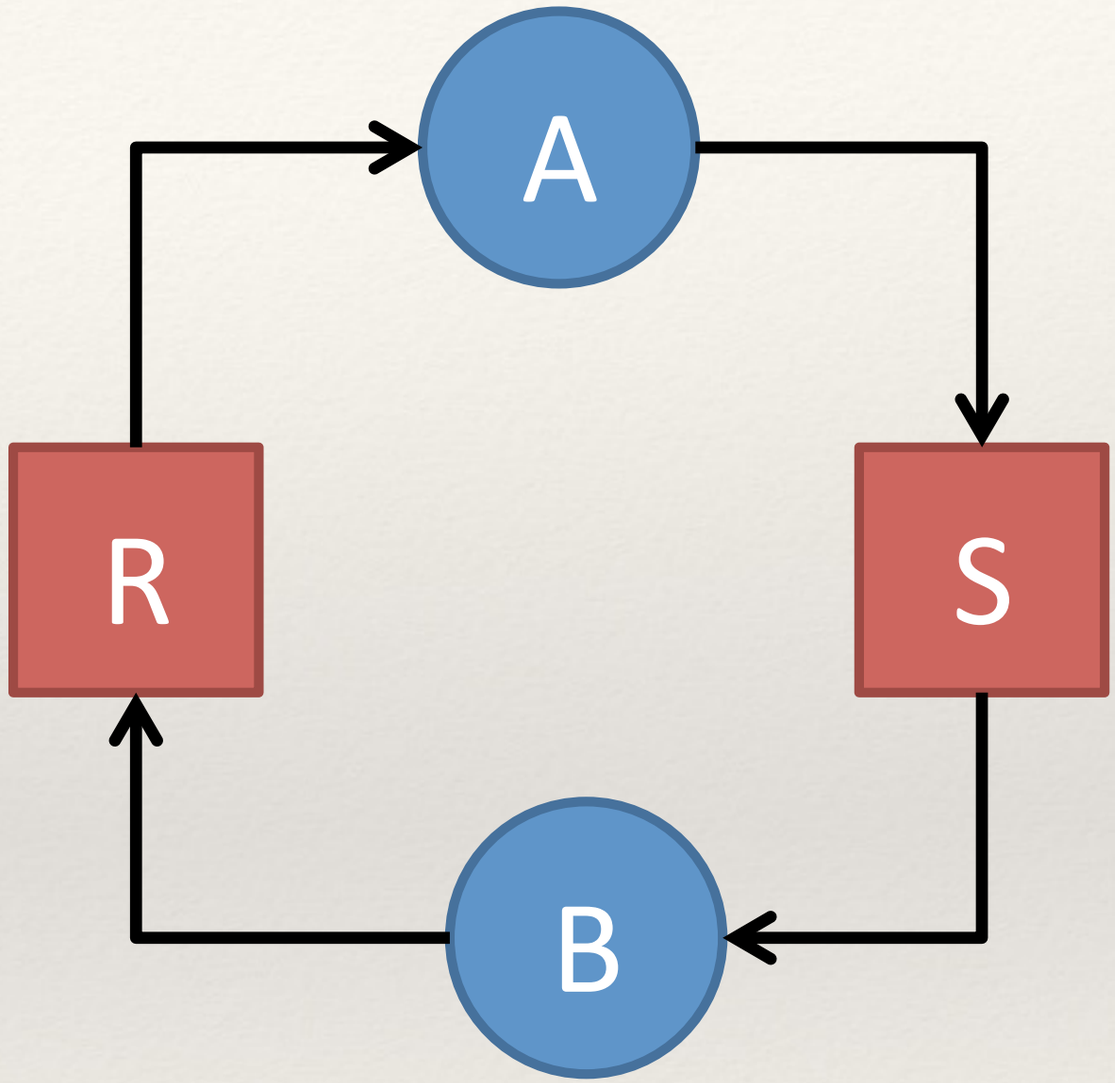
- Dessa forma, um *deadlock* pode ser demonstrado através de um **grafo**
- Considere a seguinte sequência de eventos:
  - Processo A requisita recurso T
  - Processo B requisita recurso S
  - Processo A requisita recurso S
  - Processo B requisita recurso T

- Represente a sequência de eventos utilizando os elementos:

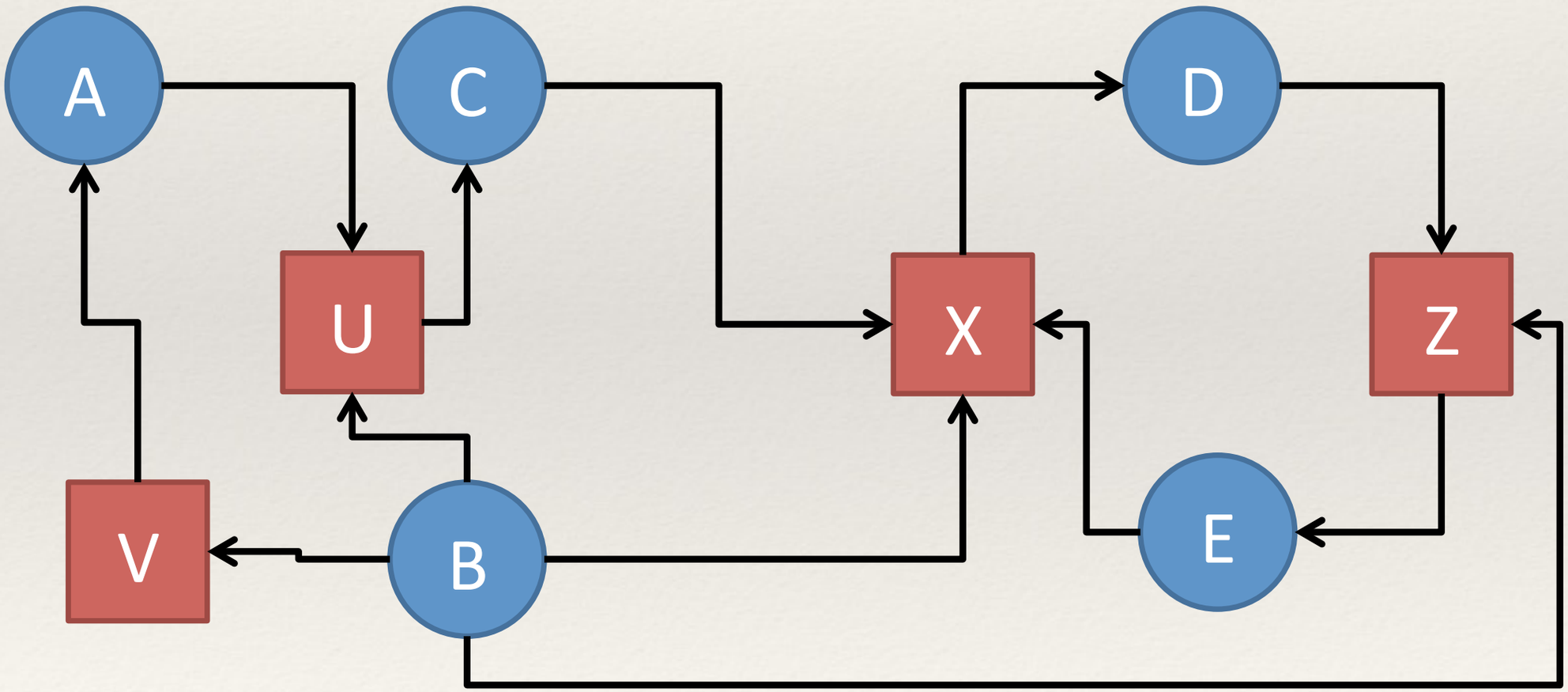


- Ao lado, a modelagem correta da interação entre recursos e processos no slide anterior

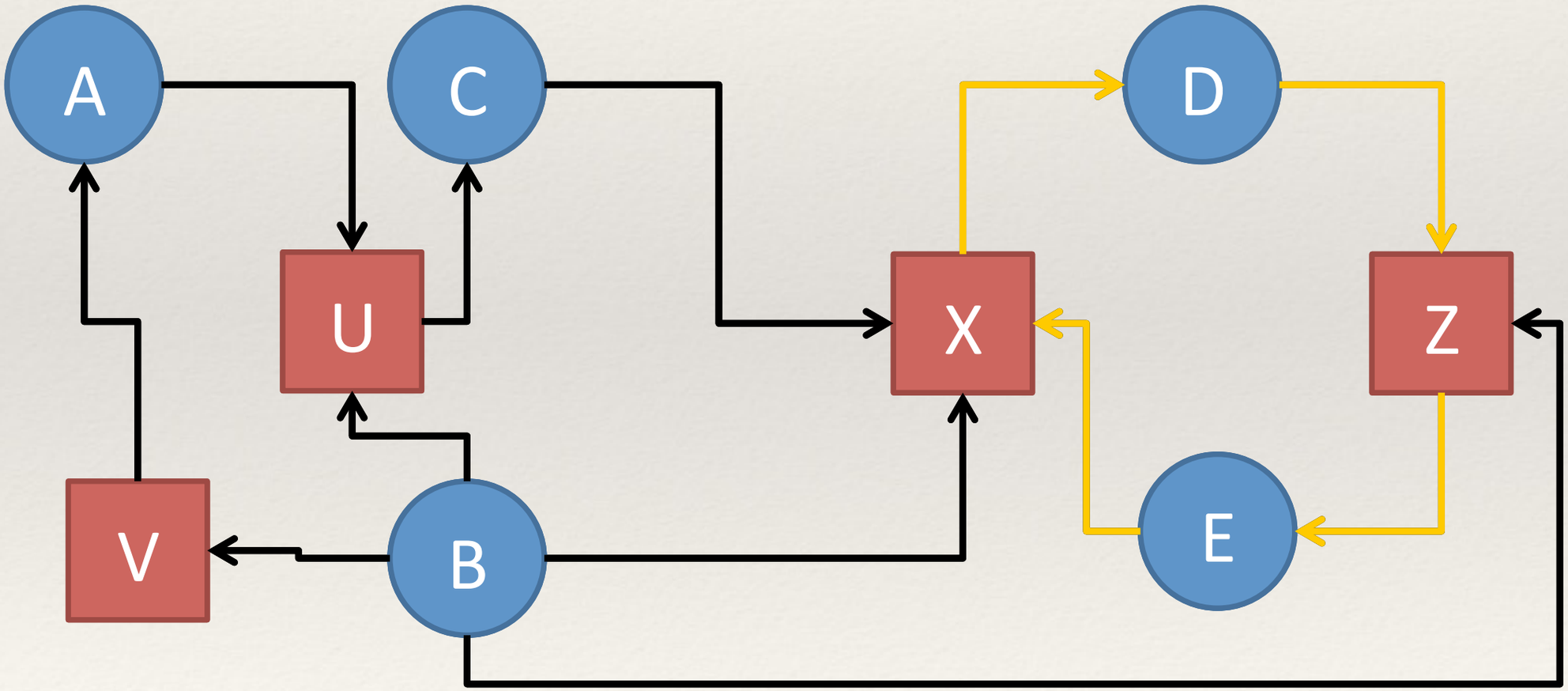
- Sempre que houver um **ciclo**, haverá um *deadlock*



- Não é necessário que todos os processos e recursos estejam envolvidos no ciclo para haver deadlock
- Onde está o deadlock abaixo?



- Não é necessário que todos os processos e recursos estejam envolvidos no ciclo para haver deadlock
- Onde está o deadlock abaixo?



Reagindo a Deadlocks

- Há quatro possíveis posturas em relação a *deadlocks*:

  1. Ignorar o problema
  2. Verificação e recuperação
  3. Evitar ao alocar recursos de maneira cuidadosa
  4. Prevenção ao negar alguma das quatro condições mínimas para *deadlocks*

Ignorar o Problema

- Algoritmo de *Ostrich* (Avestruz)

Diz a lenda que quando um avestruz encontra um perigo de vida, como um leão à sua frente, ele enterra a cabeça na areia para fingir que não o viu

- Dependendo do problema, não fazer nada é uma solução aceitável
- A chance de *deadlock* em um sistema específico pode ser de tal magnitude que aconteça um caso a cada cinco anos
- Exemplo:
  - Muitos sistemas atuais deixam o controle de acesso aos dispositivos *CD-ROM* e scanner com seus respectivos *drivers*
  - Os *drivers* quase sempre implementam acesso exclusivo
  - Se um processo acessa o *CD-ROM* e ou o *scanner*, e depois eles tentam acesso trocado, acontecerá um *deadlock*
  - Quase nenhum sistema moderno trata essa situação

Verificação e Recuperação

- A postura do sistema é deixar *deadlocks* acontecerem
- Em intervalos de tempo
  - utilizando a técnica do ciclo em grafo direcionado, o sistema verifica a existência de *deadlocks*
  - após montar o grafo, segue um algoritmo de verificação

Algoritmo de Verificação

1. Para cada vértice, faça os seguintes passos
2. Inicialize uma lista L, vazia, e outra de arestas marcadas, também vazia
3. Adicione o atual vértice a L e verifique se há outro igual a ele na lista. Se houver, há um deadlock
4. Verifique se há um vértice de saída não marcado. Se houver, siga-o e faça passo 5, senão faça passo 6
5. Escolha um dos arcos não marcados de saída, marque-o e torne o seu vértice destino o atual. Volte ao passo 3
6. Se esse vértice for o inicial, o algoritmo termina. Senão, encontrou-se um caminho sem saída. Remova o último vértice de L e faça o seu novo topo o atual vértice. Volte ao passo 3

Recuperação

- Suponha que o algoritmo verificou um *deadlock*. O que fazer?
- **Recuperação por preempção**
  - O usuário escolhe um processo para interromper e ter um recurso retirado
  - Depois tenta voltar a executá-lo
  - Como seria um *deadlock* com impressora?
- **Recuperação por reversão (*rollback*)**
  - Um sistema muito apto a *deadlocks* pode guardar os estados de processos periodicamente
  - Quando ocorrer um *deadlock*, basta reverter um dos processos em *deadlock* para um estado anterior ao acesso ao recurso relevante, e liberá-lo para outro processo
- **Recuperação matando processos**
  - O sistema oferece ao usuário uma lista de processos para que escolha um ou mais a serem terminados

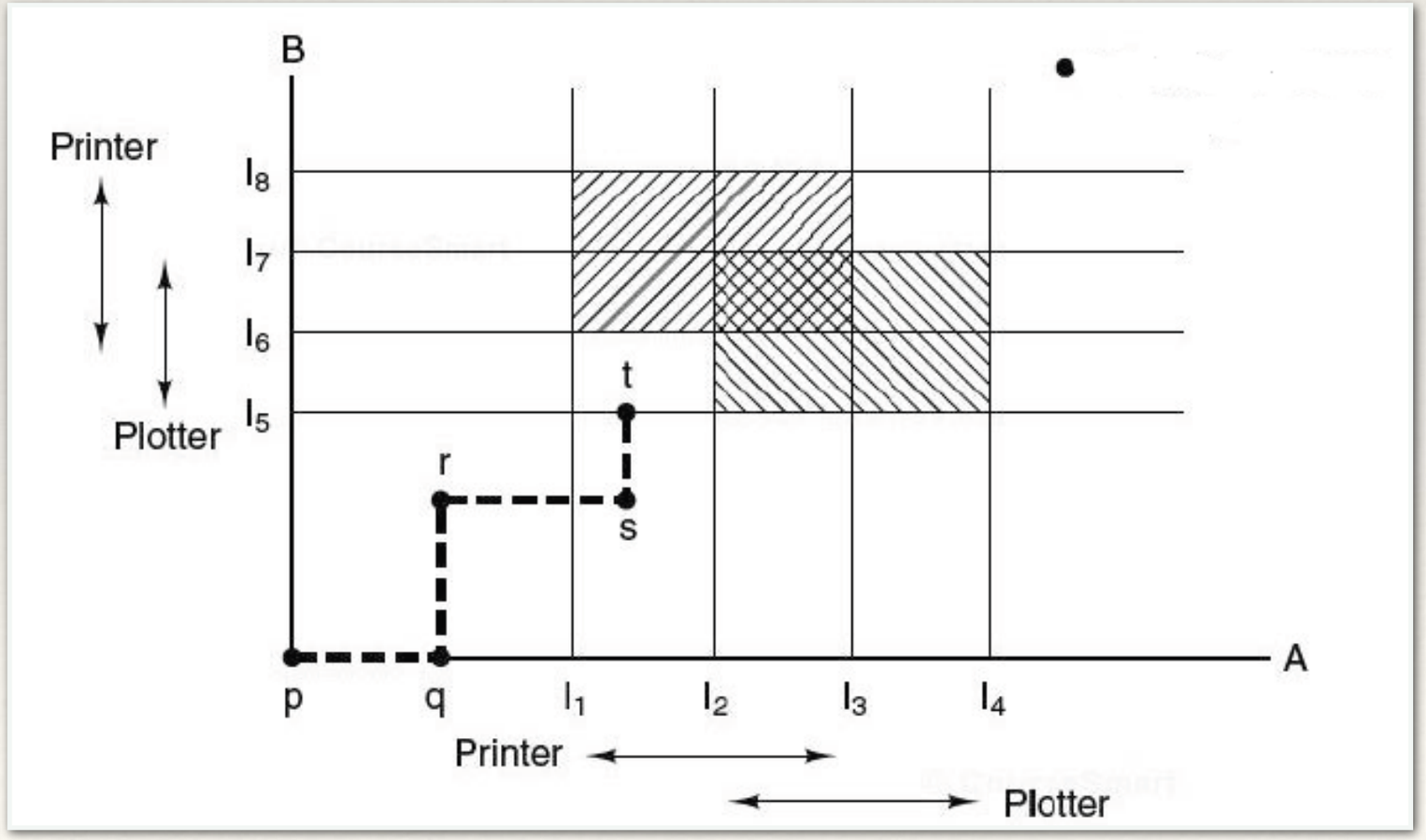
- Discussão:
  - Quais as possíveis desvantagens do processo de verificação?
  - Quais as desvantagens do processo de recuperação?
    - Preempção
    - Reversão
    - Terminando processos

Evitando Deadlocks

- Na maioria dos sistemas, recursos são requisitados um por vez
- É possível evitar *deadlock* ao partilhar recursos de maneira cuidadosa
- Conhecido como **técnica de estados seguros e inseguros**
  - exige dados sobre utilização de recursos pelos processos no momento em que são criados

Estados Seguros e Inseguros

- Considere os processos A e B, que requisitam uso exclusivo de impressora e *plotter*:
  - A requisita **impressora** em I.1 e ***plotter*** em I.2, os liberam em I.3 e I.4
  - B requisita ***plotter*** em I.5 e **impressora** em I.6, os libera em I.7 e I.8



Algoritmo do Banqueiro

- Verifica as necessidades máximas de todos os processos, e quantos recursos há disponível
- No momento de uma requisição, verifica se o sistema entrará em estado inseguro
- Se for o caso, o processo é bloqueado até que outros processos liberem recursos
- Analogia com o sistema bancário de conta corrente e poupança

Resolução por estados seguros

	Há	Max
A	3	9
B	2	4
C	2	7
Livres	3	

	Há	Max
A	3	9
B	4	4
C	2	7
Livres	1	

	Há	Max
A	3	9
B	0	-
C	2	7
Livres	5	

	Há	Max
A	3	9
B	2	4
C	7	7
Livres	0	

	Há	Max
A	3	9
B	0	-
C	0	-
Livres	7	

Estado inseguro e *deadlock*

	Há	Max
A	3	9
B	2	4
C	2	7
Livres	3	

	Há	Max
A	4	9
B	2	4
C	2	7
Livres	2	

Inseguro!!

	Há	Max
A	4	9
B	4	4
C	2	7
Livres	0	

	Há	Max
A	4	9
B	-	-
C	2	7
Livres	4	

deadlock…

Vale a pena evitar deadlocks?

- Discuta quais os benefícios e malefícios em adotar uma política de evitar *deadlocks* e o algoritmo do banqueiro
  - Quais as dificuldades em adotar essa postura?
  - Que tipos de sistemas se beneficiam?
  - Em que tipo de sistema é impraticável?

- Se você fosse o projetista de Windows e Linux, adotaria o algoritmo do banqueiro? Por quê?

Prevenindo Deadlocks

- Postura: *"evitar deadlocks é praticamente impossível"*
- Talvez a melhor solução seja trabalhar na negação de uma das quatro condições necessárias ao *deadlock*:
  - **Exclusão mútua** para acesso aos recursos
  - **Acesso e espera**, um processo pode requisitar acesso a mais recursos, mesmo em posse de outros
  - **Recursos não-preemptáveis**, não podem ser retirados a força de um processo
  - **Condição circular de espera**

Negando "exclusão mútua"

- Ao invés de permitir acesso exclusivo a dispositivos, programar um servidor (*daemon*) para gerenciar acessos a eles
- Analogia com o *spooler* de impressora
- Problemas:
  - Como fazer isso com um gravador de DVDs?
  - Mesmo um *spooler* pode ter problemas:
    - No spooler cabem mil páginas
    - Processo A envia 500 páginas para impressão, mas precisa de 600
    - Processo B envia outras 500 páginas, mas precisa de 800
    - *Deadlock!*

Negando "Acesso e Espera"

- Um processo não pode estar segurando um recurso e requisitar outros.
- Duas ideias:
  - Processos são obrigados a requisitar todos os recursos antes de entrarem em execução
  - Processos são obrigados a liberarem todos os recursos que estão segurando e requisitarem tudo de uma vez
- Crítica?

Negando "Recursos não-preemptáveis"

- Eliminar recursos não-preemptáveis do sistema
- Uma maneira de fazê-lo é virtualizar os recursos:
  - Ao imprimir, um processo gera um arquivo impresso virtual, que depois é impresso fisicamente por um *daemon* (fila de impressão)
- Nem todos os recursos podem ser virtualizados dessa forma
  - Operações de escrita a bancos de dados
  - Recursos com tempo crítico de operação (placa de vídeo, por exemplo)

Negando "condição cíclica"

- Evitar a possibilidade de haver condição cíclica no grafo
- Uma maneira é simplesmente proibir processos segurando um recurso de requisitar outros
  - Impossibilita certos tipos de tarefas
- Outra maneira é a enumeração dos dispositivos
  - Se todos os processos requisitarem recursos na mesma ordem, nunca haverá *deadlock*
  - Neste caso, funcionalidades estão limitadas (um gerente de tabelas em um BD pode ser impedido de conseguir a impressora)
  - Pode ser impossível encontrar enumeração que responda por todos os programas existentes no sistema

Conclusão...

- Não há situação perfeita!
- A política em relação a *deadlocks* depende
  - do tipo de sistema desejado
  - dos usuários que vão utilizá-lo
  - dos programas que serão utilizados no sistema