Aula 9

Deadlocks

1.1 O que é
1.2 Caracterização
1.3 Grafo de dependência
1.4 Métodos de tratamento
1.5 Detecção
1.6 Recuperação

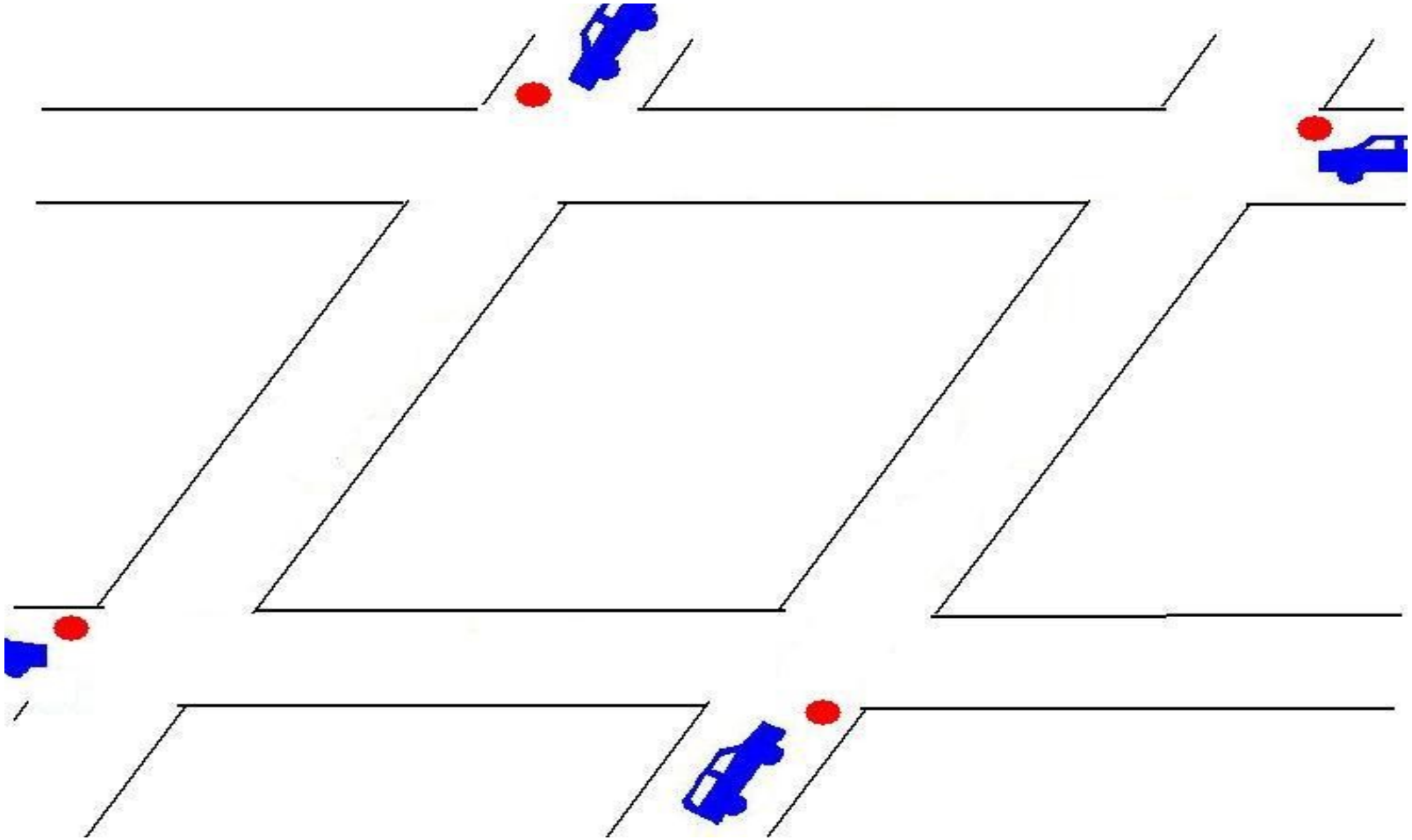
O quê é Deadlock (1)

"Propriedade – indesejável – de um conjunto de processos que estão **bloqueados indefinidamente** esperando uns pelos outros"

Deadlock = Impasse

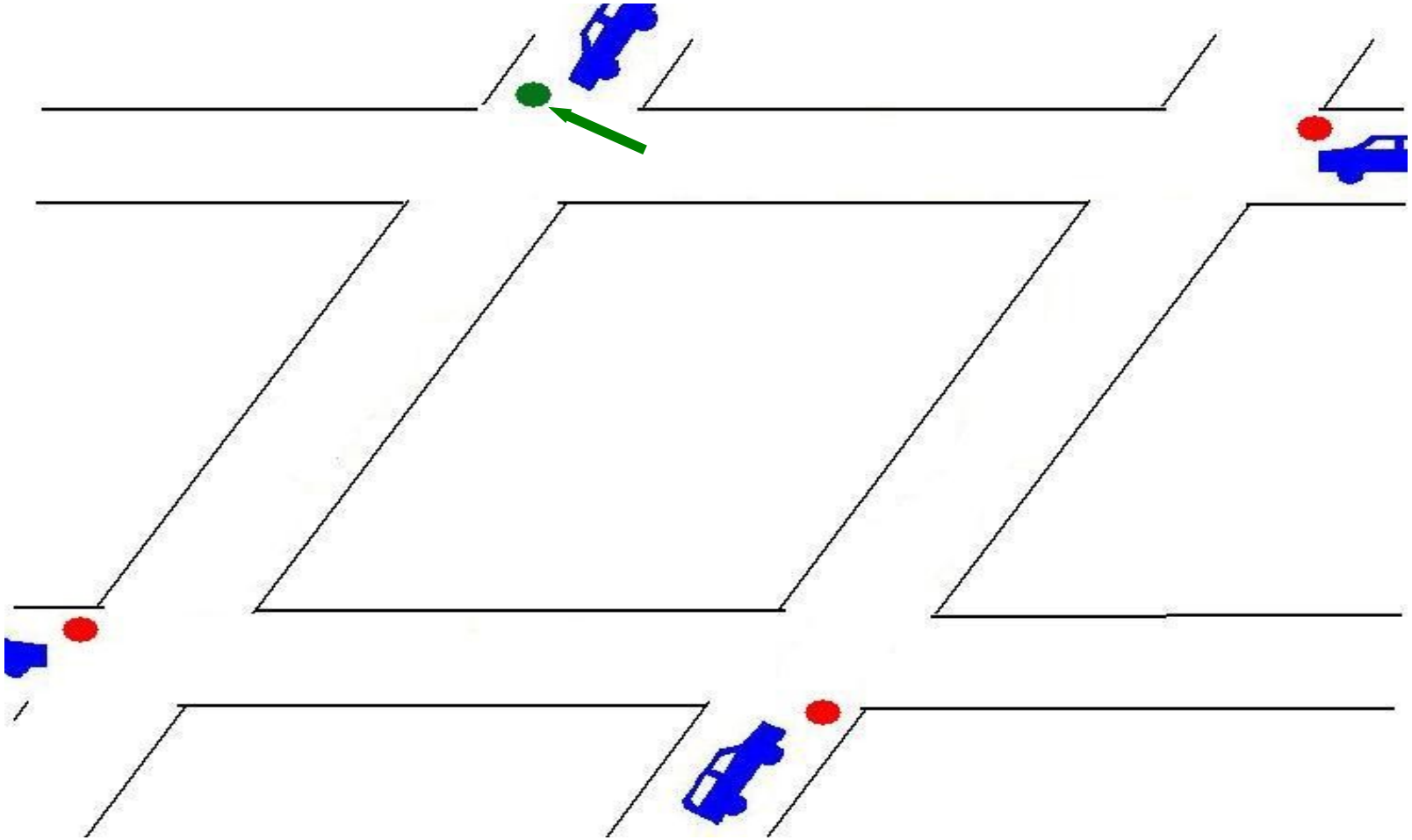
O quê é Deadlock – Exemplo 1 (1)

No início, ninguém detém recursos



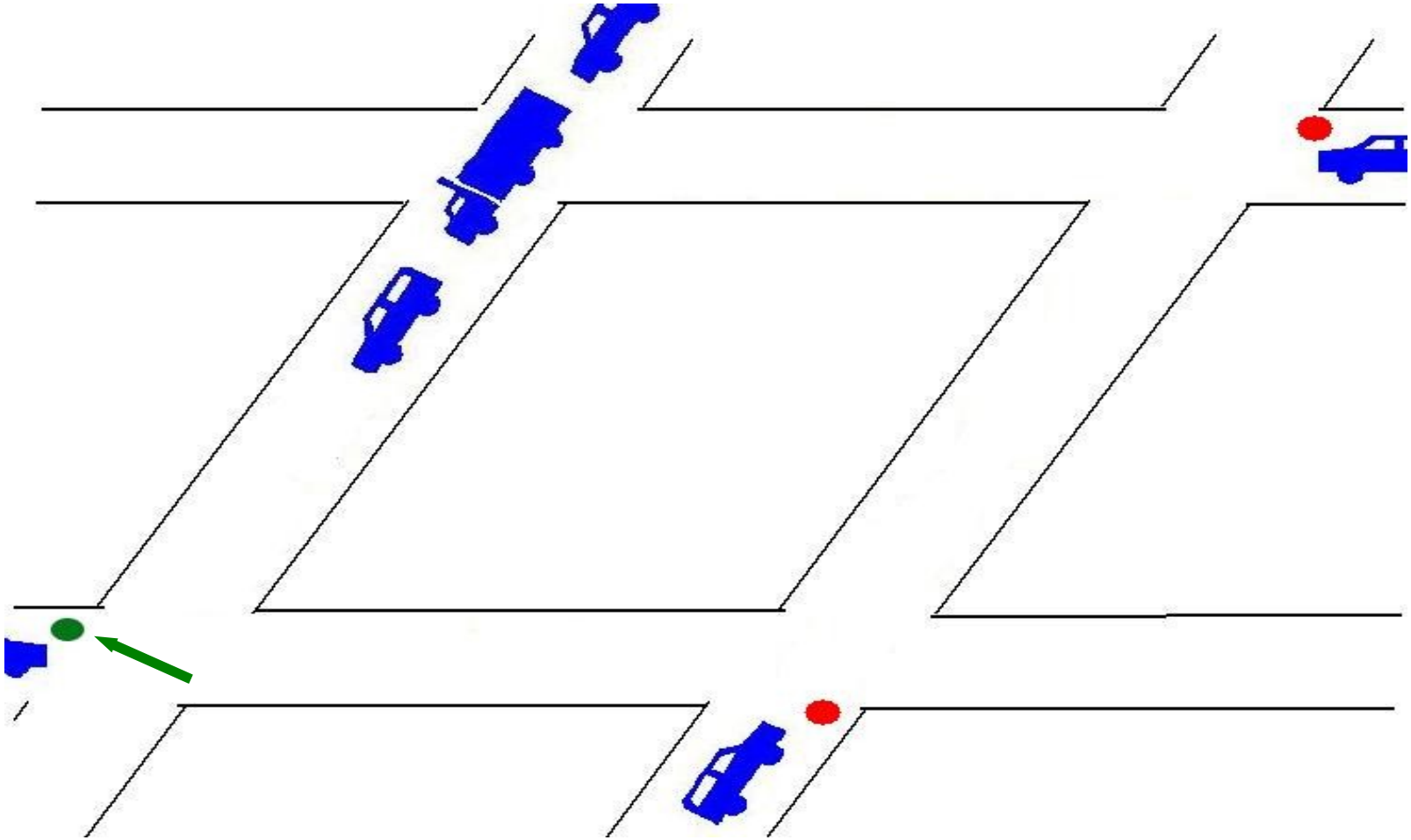
O quê é Deadlock – Exemplo 1 (2)

Agora, começamos a liberar recursos



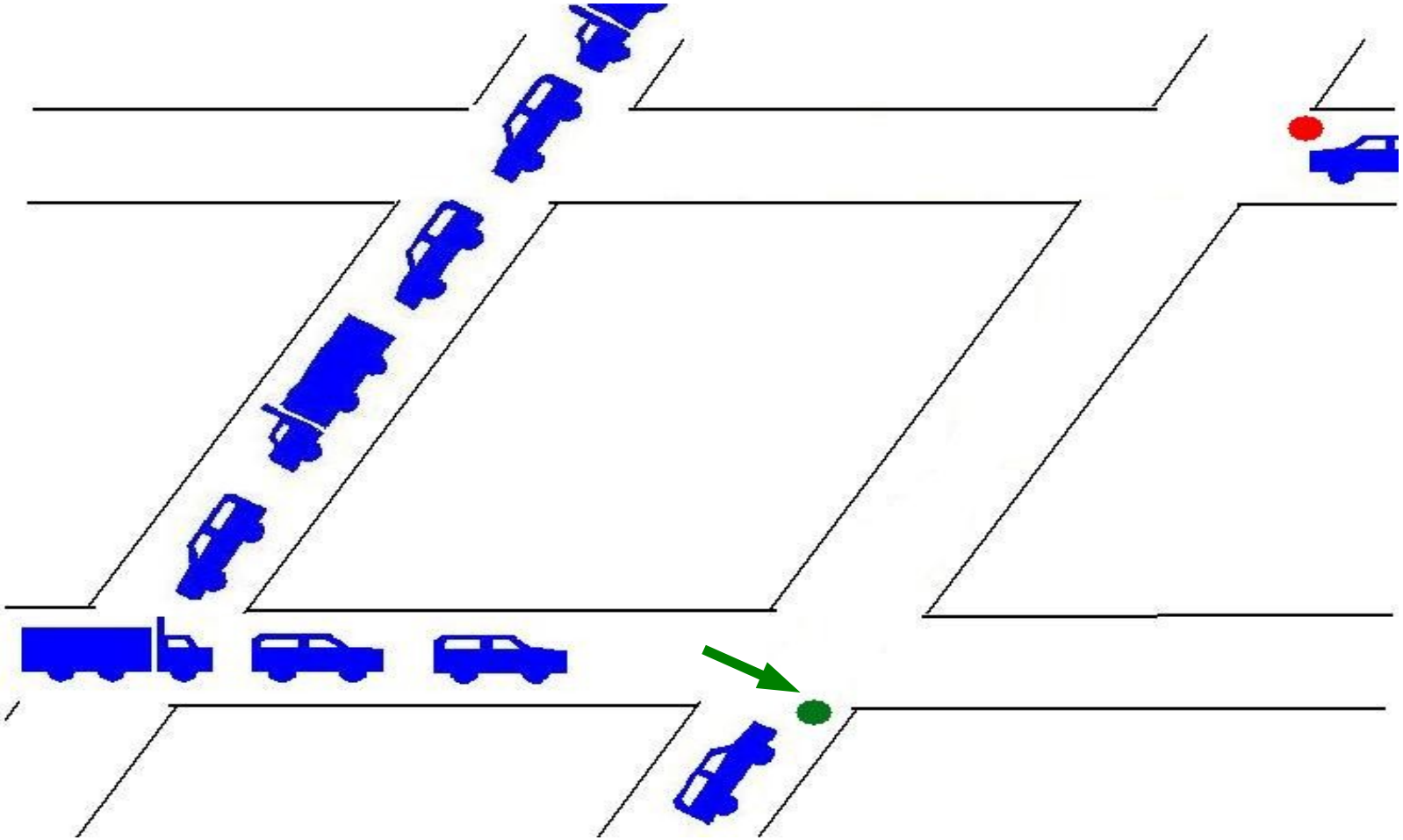
O quê é Deadlock – Exemplo 1 (3)

E vamos liberando...



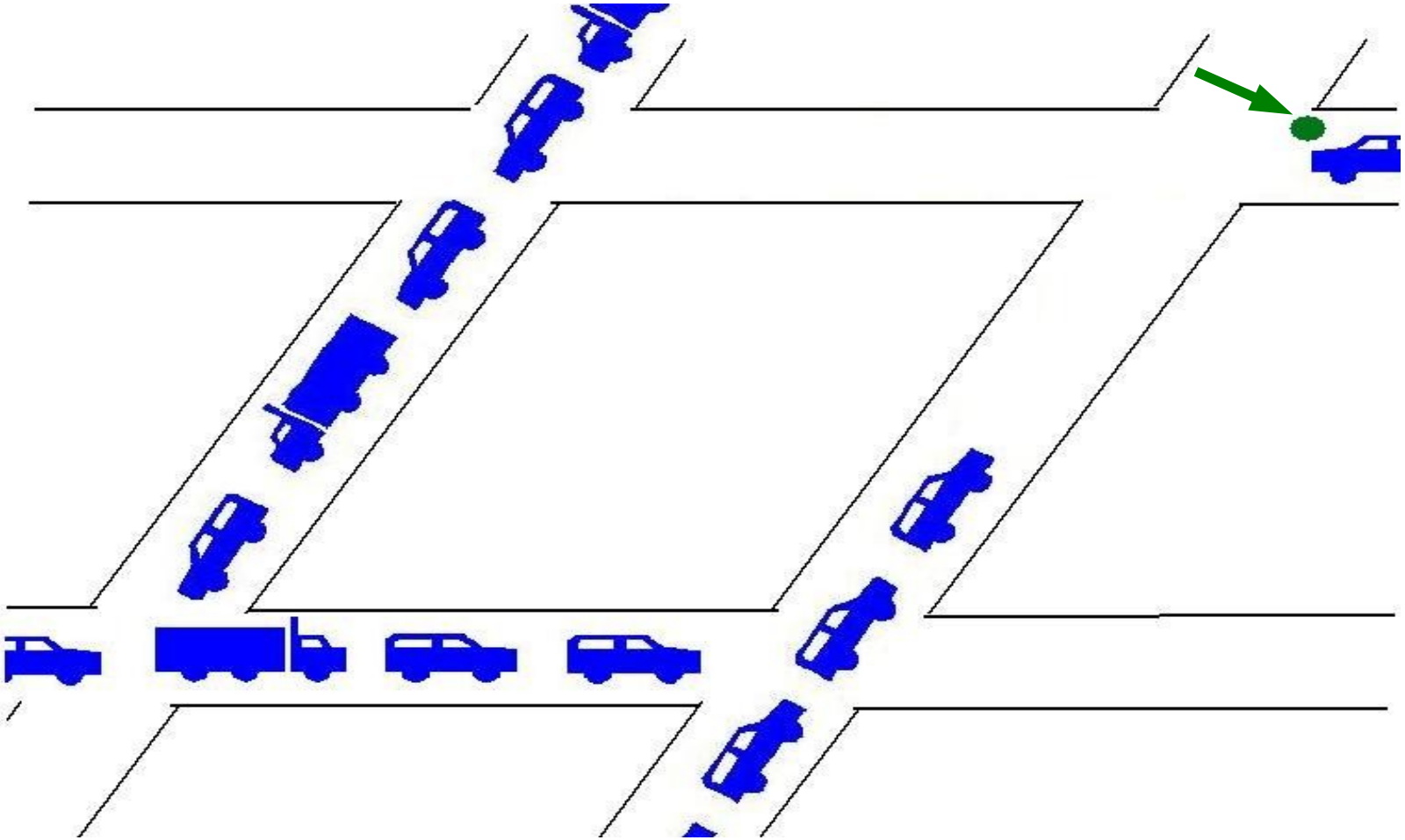
O quê é Deadlock – Exemplo 1 (4)

E vamos liberando...



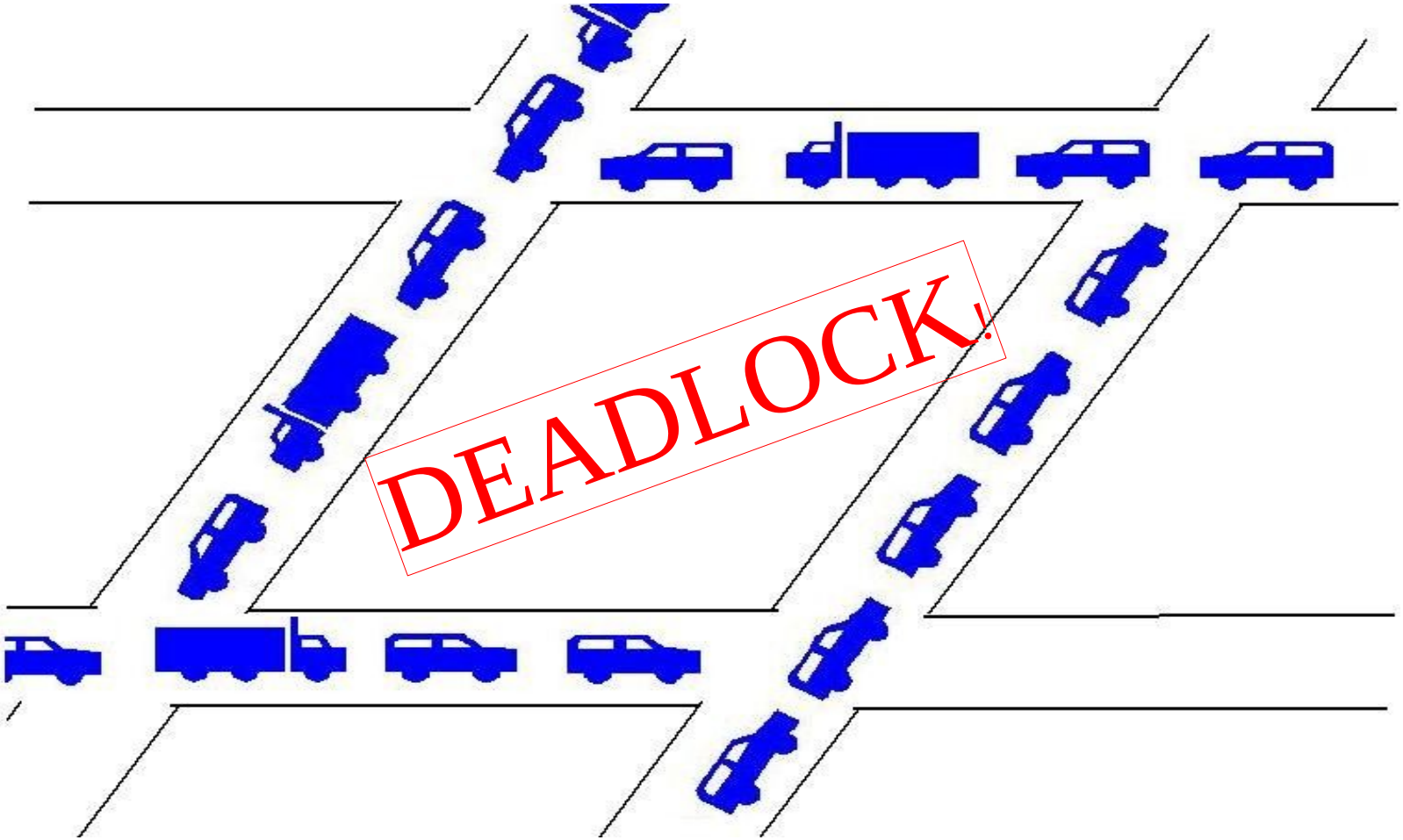
O quê é Deadlock – Exemplo 1 (5)

E vamos liberando...



O quê é Deadlock – Exemplo 1 (6)

Até ocorrer um impasse!



11455

O quê é Deadlock – Exemplo 2

```
P1 () {
  lock (A);
  lock (B);
  mutex ();
  unlock(B);
  unlock(a);
}
```

```
P2 () {
  lock (B);
  lock (A);
  mutex ();
  unlock(A);
  unlock(B);
}
```

DEADLOCK!

Caracterização: Condições para Deadlock

Condições **necessárias**, mas não suficientes:

- Se existem, **pode** ocorrer deadlock;
- Se não existem, não pode!
  - Exclusão mútua;
  - Sem preempção;
  - Espera circular (os 2 exemplos):

    Existe um conjunto de processos $\{P_0,...P_{n-1}\}$,

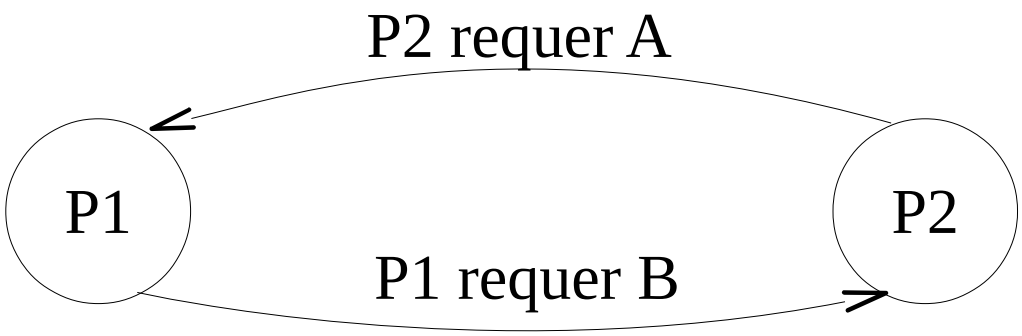
    tal que:

    - P_i precisa de um recurso obtido por $P_{i+1\%n}$

Grafo de Dependências (1)

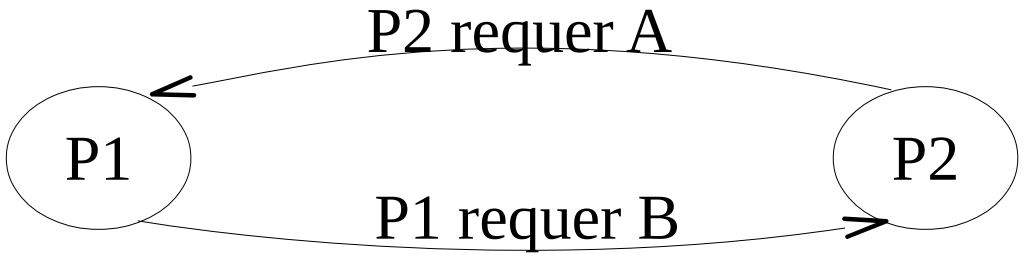
Grafo direcionado que representa as dependências entre os processos do sistema:

- Nodos são processos
- Aresta P_i->P_k existe se P_k tem **algum** recurso que P_i precisa
  - Múltiplos recursos podem ser representados

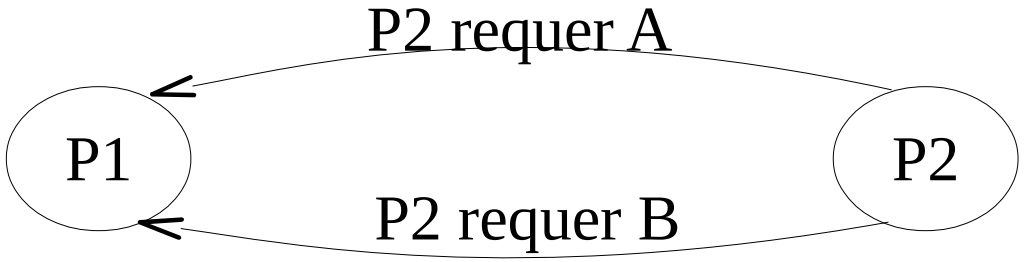


Grafo de Dependências (2)

Deadlocks existem **se e somente se** existem ciclos no grafo de dependências.



Deadlock



No Deadlock

Exceção: recursos com instância múltiplas: ex semáforos não binários.

Métodos de tratamento

- Projetar o protocolo de comunicação de forma que um estado de deadlock não é alcançável
- Permitir que o sistema entre em deadlock e recuperar
- Fingir que o problema não existe

Qual o método mais usado?

- Fingir que o problema não existe! Usado na maioria dos sistemas, inclusive Unix

Prevenção de Deadlock (1)

Uma maneira de se evitar deadlocks é evitar que ao menos uma de suas condições ocorra:

- Exclusão mútua
- Espera circular: Aquisição gradativa de recursos
- Sem preempção

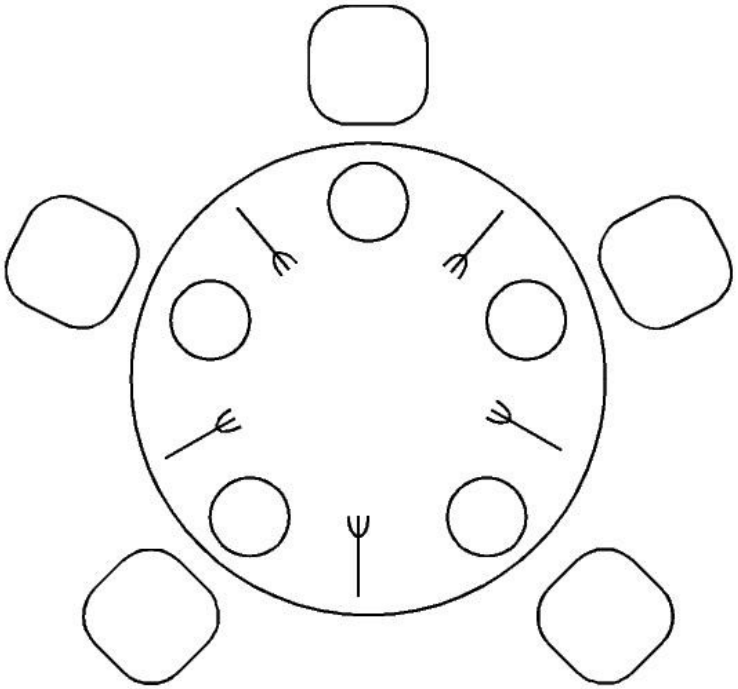
Prevenção de Deadlock (2)

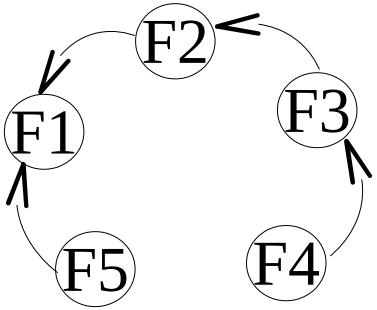
Uma maneira de se evitar deadlocks é evitar que ao menos uma de suas codições ocorra:

- Exclusão mútua: impossível de evitar, em geral
- Aquisição gradativa de recursos: evitável, ver abaixo
- Sem preempção: evitável
  - Adquirir todos os recursos antes de começar a executar
  - Se algum recurso não estiver disponível, liberar todos os recursos que já foram adquiridos
  - Ineficiente

Prevenção de Deadlock (3)

- Espera circular: evitável
  - Para forçar que o grafo de dependências seja acíclico, podemos **ordenar** todos os recursos (numerando-os, por exemplo)
  - Todos os recursos são adquiridos em ordem crescente da ordenação
  - Exemplo: jantar dos filósofos:





Prevenção de Deadlock - Algoritmo do Banqueiro(1)

Usado para determinar se um processo pode executar de maneira segura ou não:

- Todos os processos declaram o máximo de recursos que vão usar durante a execução
- A execução de um novo processo é permitida se a soma dos **recurso requisitados** é menor que a soma dos disponíveis

  Recursos requisitados =

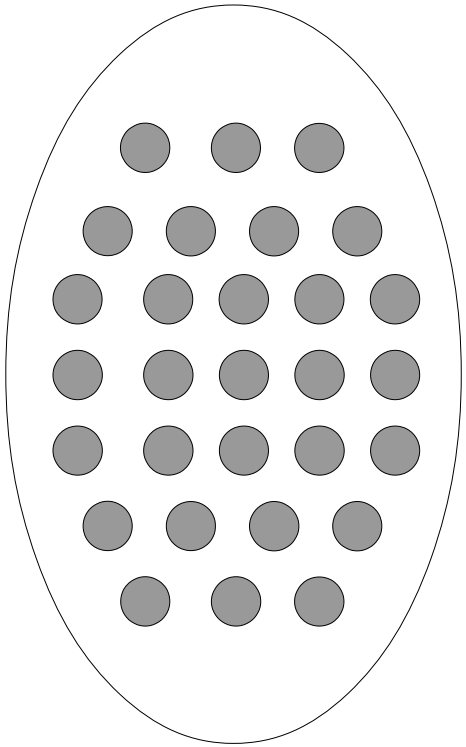
  - Recursos já usados por todos os processos já executando +
  - Recursos exigidos pelo novo processos

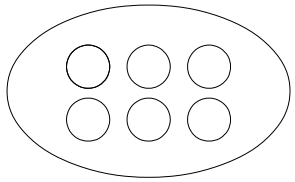
Prevenção de Deadlock - Algoritmo do Banqueiro(2)

Recursos Disponíveis

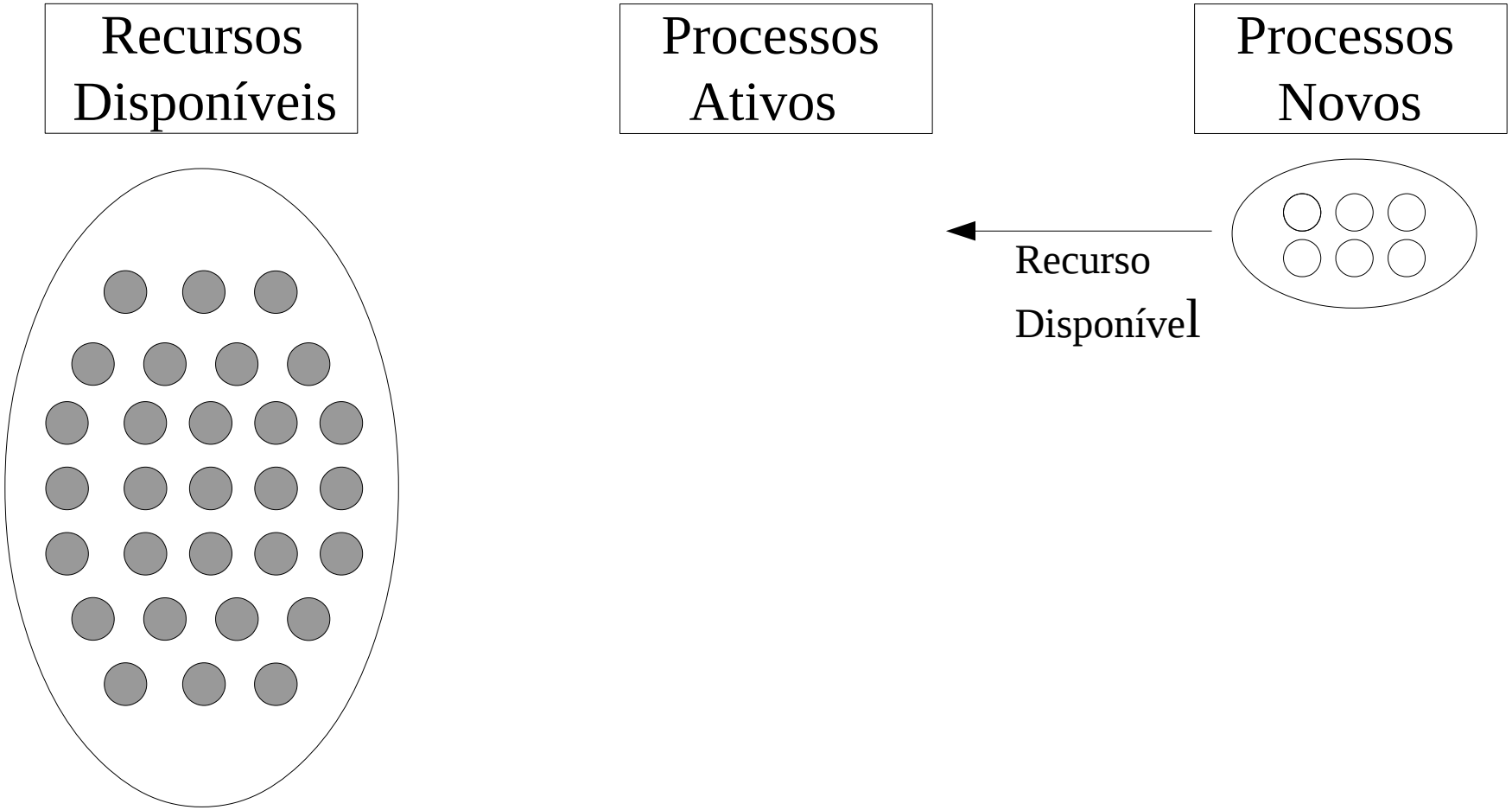
Processos Ativos

Processos Novos

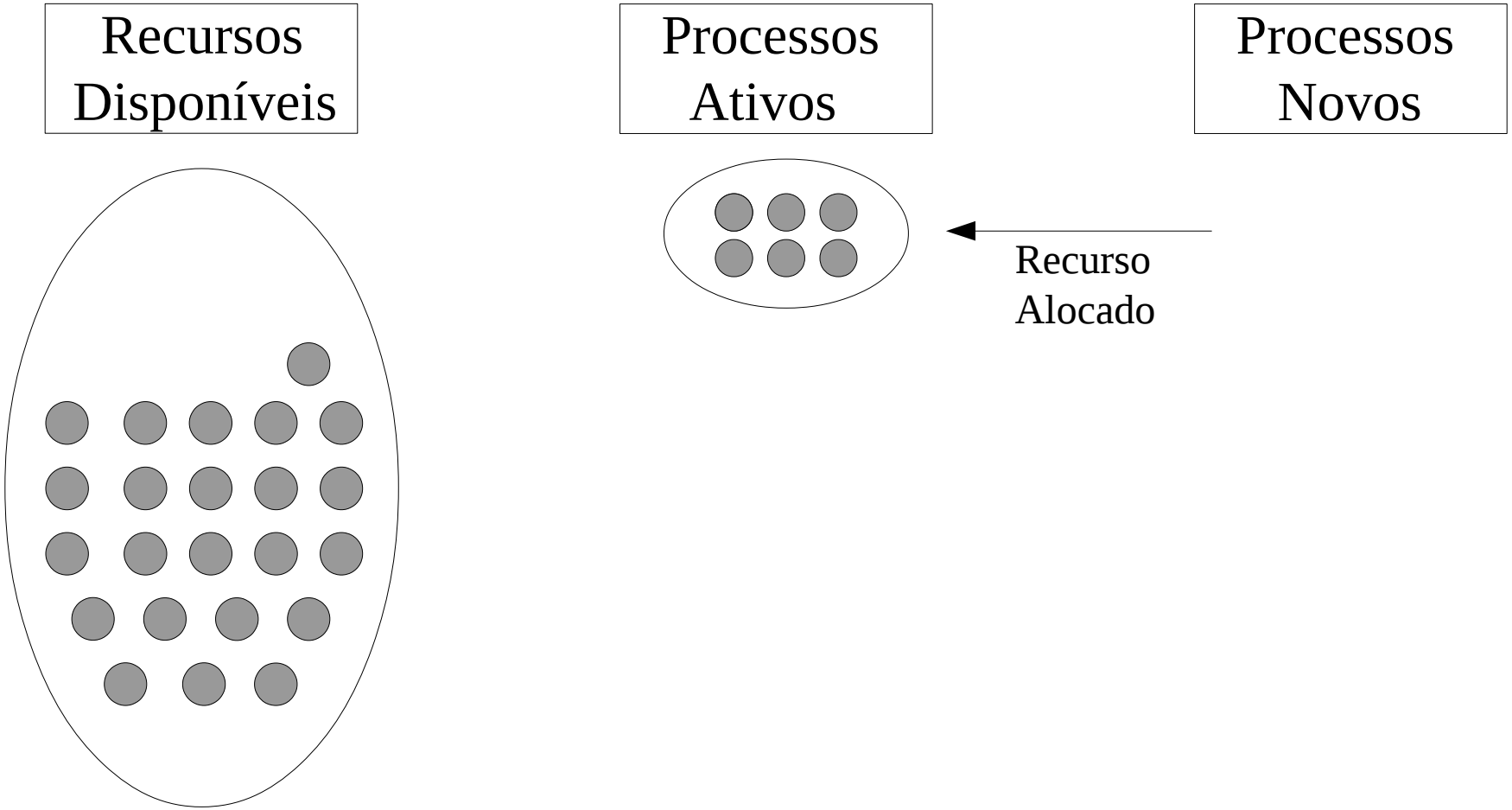




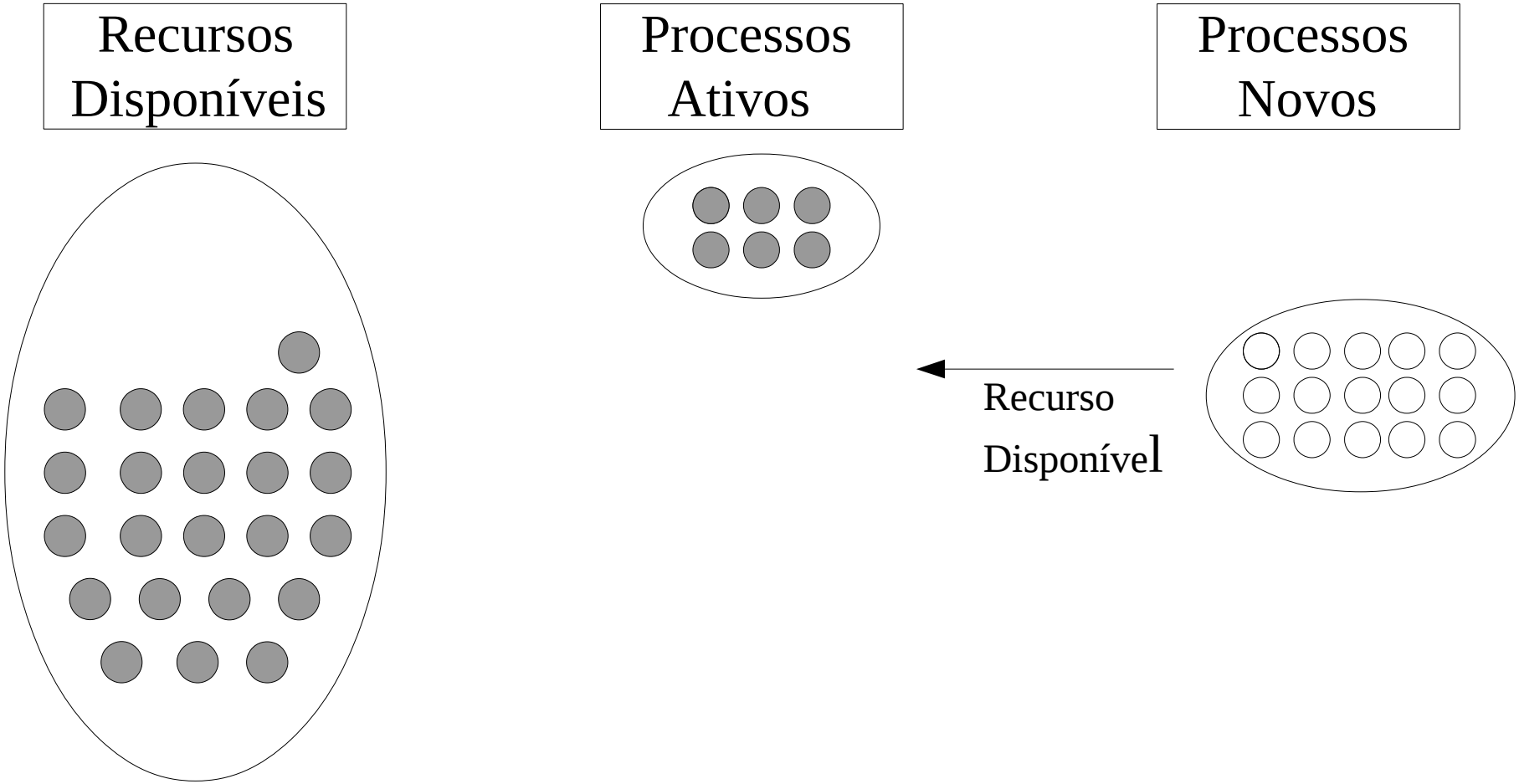
Prevenção de Deadlock - Algoritmo do Banqueiro(3)



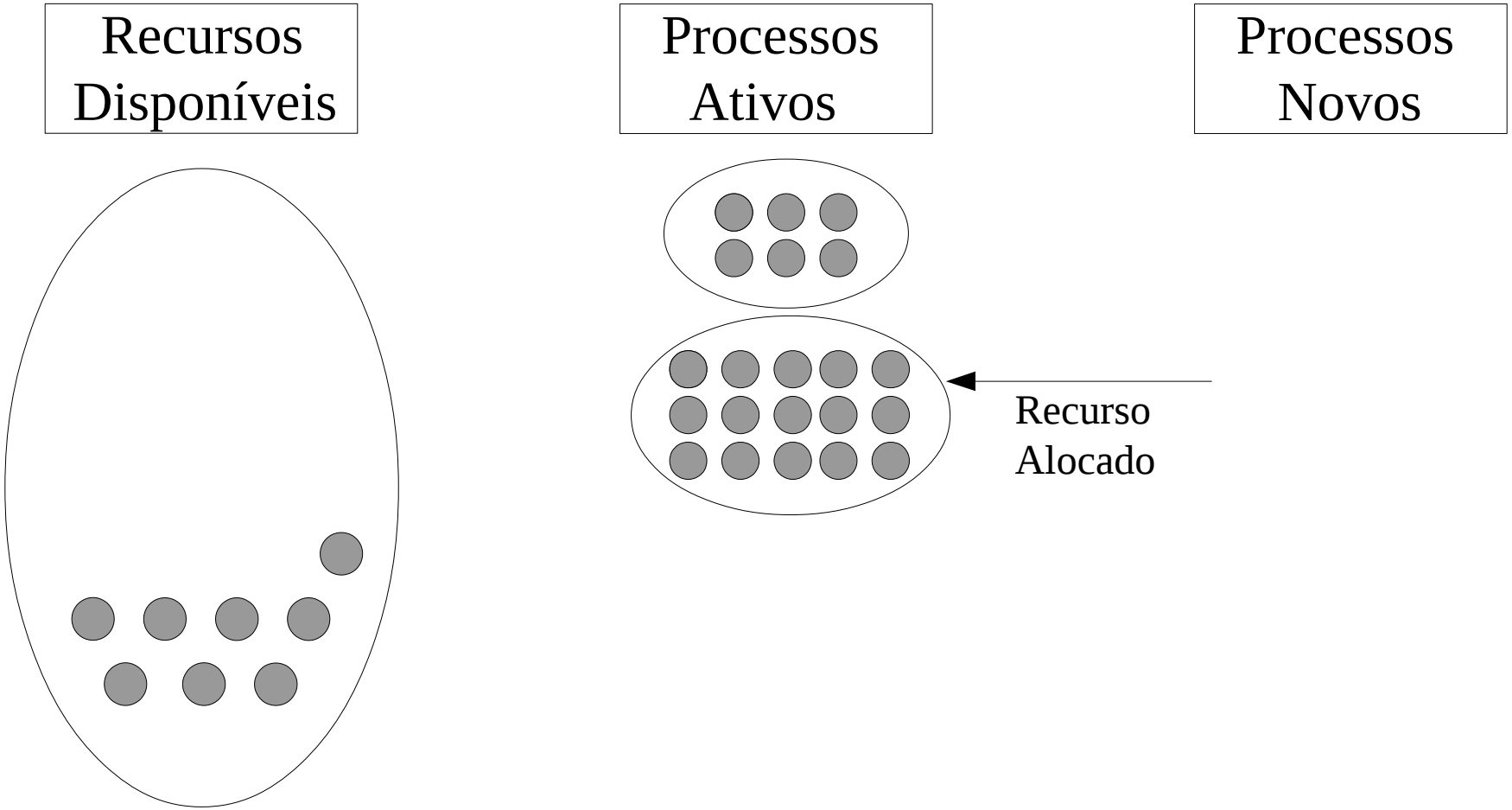
Prevenção de Deadlock - Algoritmo do Banqueiro(4)



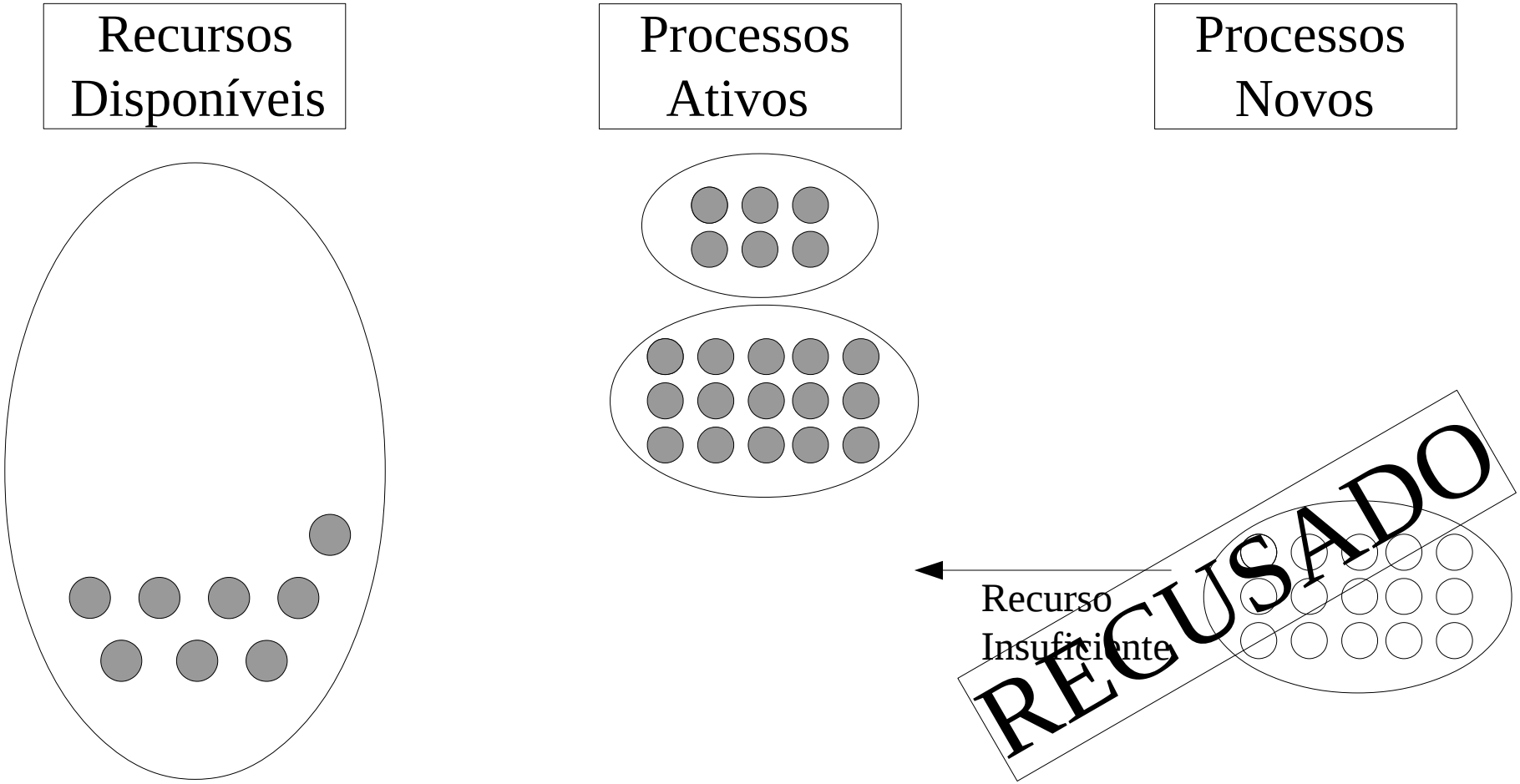
Prevenção de Deadlock - Algoritmo do Banqueiro(5)



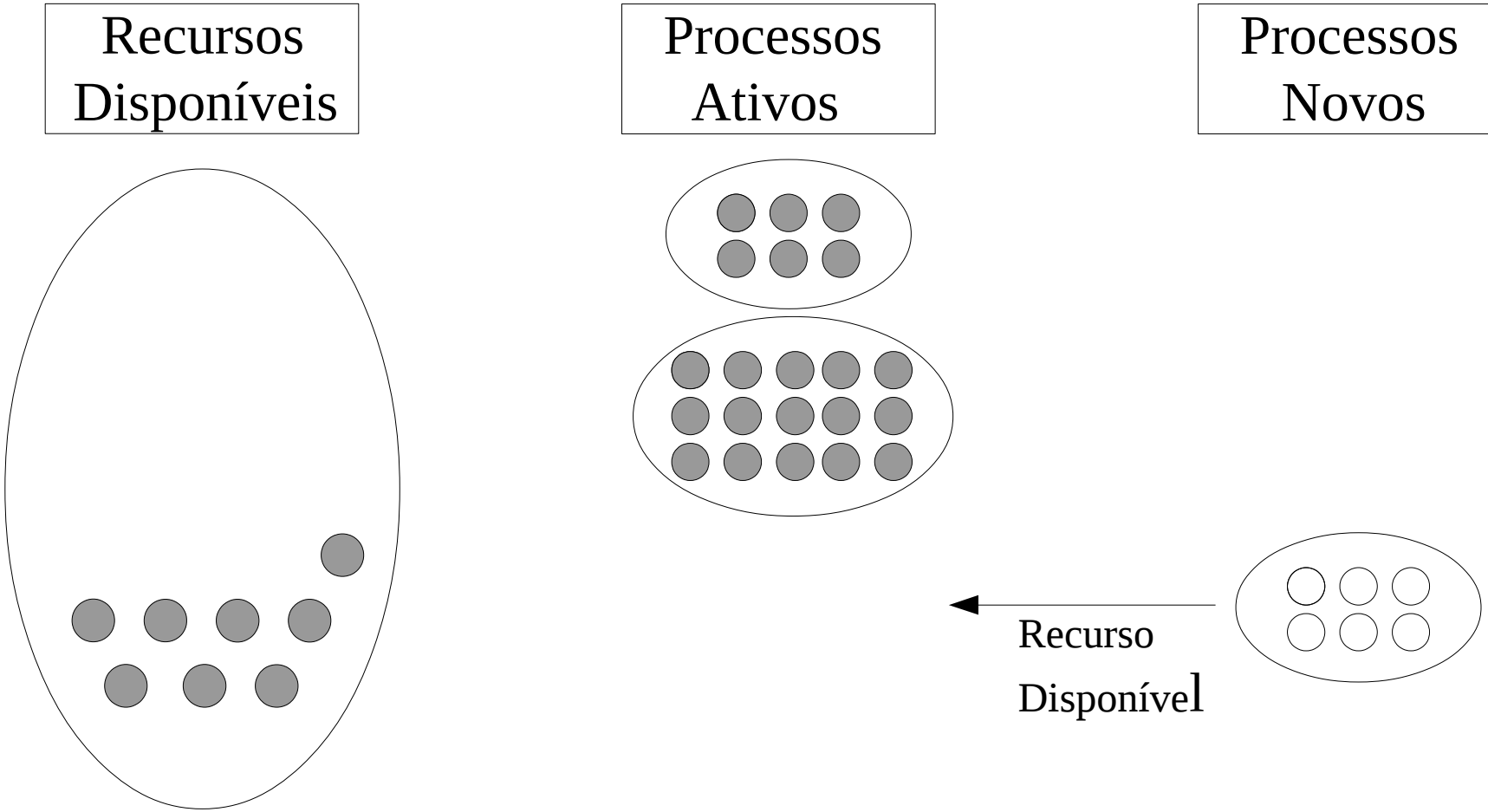
Prevenção de Deadlock - Algoritmo do Banqueiro(6)



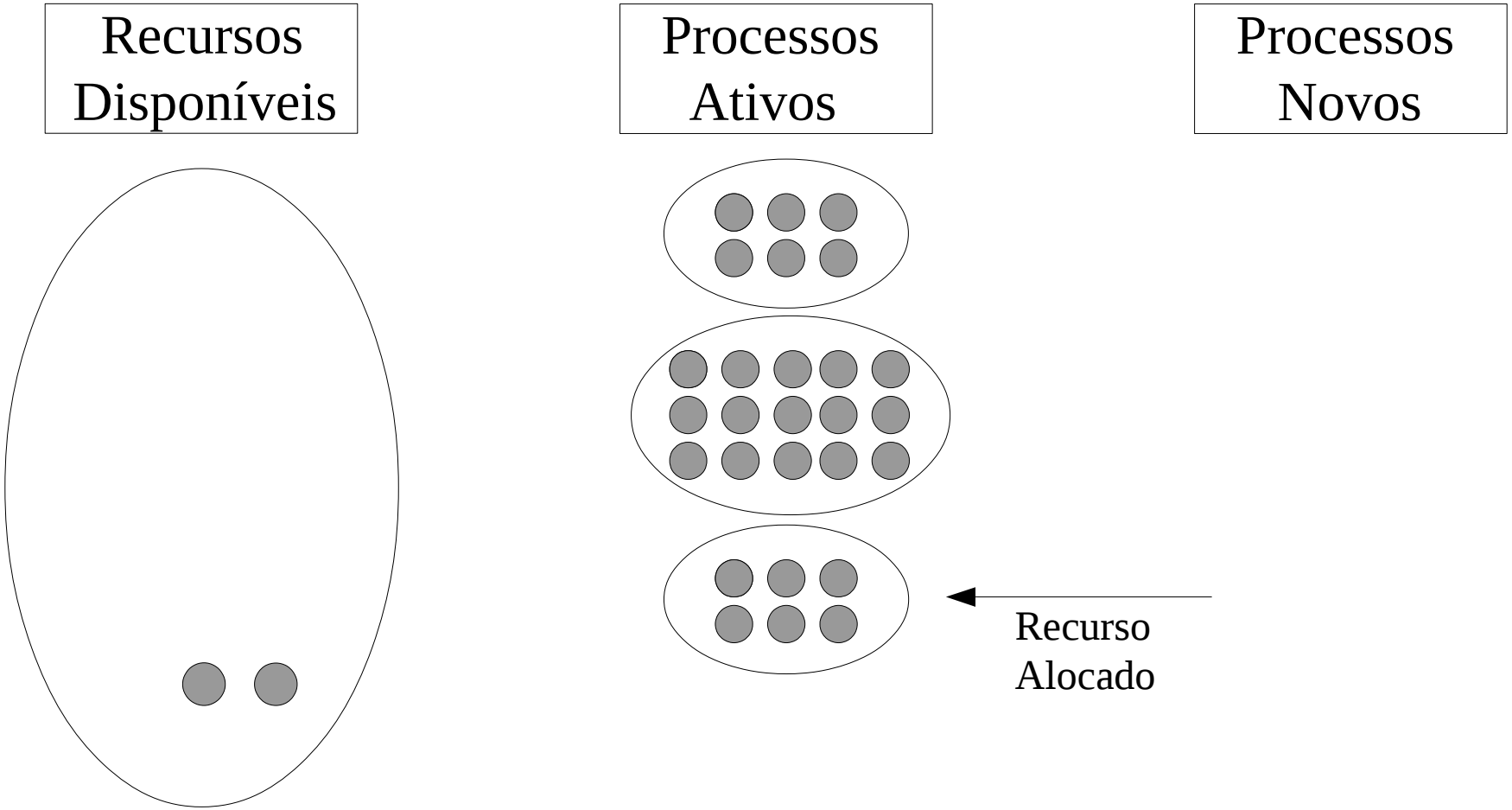
Prevenção de Deadlock - Algoritmo do Banqueiro(7)



Prevenção de Deadlock - Algoritmo do Banqueiro(8)



Prevenção de Deadlock - Algoritmo do Banqueiro(9)



Prevenção de Deadlock - Algoritmo do Banqueiro(10)

Concluindo:

- Conservativo:
  - Pode recusar processos que não causem deadlock
- Mas seguro:
  - Impede a ocorrência de deadlock
- Aplica-se a múltipla instâncias de recursos

Detecção de Deadlock

Difícil: por isso muitos preferem ignorar o problema

- Para recursos com uma instância só: busca em grafos para identificar ciclos: $O(n^2)$
- Para recursos com múltiplas instâncias: mais complicado ainda!

Por causa disto, deve-se chamar o algoritmo de detecção de deadlocks infrequentemente:

- Uma vez por hora?
- Quando a utlização da CPU cair para menos de X%?

Recuperação de Deadlock

Manual: manda o operador “se virar”

Automática: aborta-se processos

- Qual processo escolher?
- Todos?
- Se abortar sempre o mesmo => inanição

Automática: Tomar o recurso de volta do processo

- De qual processo?
- Tem que fazer o “roll back”, voltar o estado do processo – difícil

Solução Texana ?

PRIVATE PROPERTY
NO
TRESPASSING
VIOLATORS
WILL BE PROSECUTED

NO
TRESPASSING
VIOLATORS
WILL BE SHOT
SURVIVORS
WILL BE
PROSECUTED

NO
TRESPASSING
VIOLATORS
WILL BE SHOT
SURVIVORS
WILL BE SHOT
AGAIN

Welcome to Texas
Don't mess with TEXAS
Our citizens have concealed weapons
If you kill someone, We will kill you back
We enjoy gun fights - it's a Texas tradition
We have 120 prisons - enjoy your stay

Resumo – Deaclock

Deadlock - Impasse:

"Propriedade – indesejável – de um conjunto de processos que estão **bloqueados indefinidamente** esperando uns pelos outros"

Tratamento:

- Difícil de dectectar
- Difícil de evitar num contexto amplo
  - Unix e outros sistemas operacionais ignoram
- Possível de evitar num contexto específico
  - No contexto de um serviço ou aplicação
  - Grafo de alocação
  - Algoritmo do banqueiro