FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

T569 –SISTEMAS DE TEMPO REAL

Aula 13

Prof. Marcelo Sousa

Agenda

- Compartilhamento de Recursos Entre Tasks de Tempo Real
- Inversão de Prioridade
- *Priority Inheritance Protocol* (PIP)

Resource Sharing

- Independência entre Tasks
  - Algoritmos de escalonamento assumiam que as tasks eram independentes entre si.

- Dependências entre Tasks
  - Em sistemas reais em sua maioria há dependência entre Tasks

Resource Sharing

- Exemplos:
  - Sistemas Fly-by-wire
    - Velocidade
    - ➔ Cálculo de Posição ➔ Correção de posição
    - Aceleração

Resource Sharing

- Premissas assumidas no capítulo passado
  - As tasks poderiam ser interrompidas e iniciadas a qualquer tempo sem nenhum implicação no *correctness* (corretude) do dado computado.
- Premissas atuais:
  - Uma tarefa que trabalha com dados críticos não pode ser interrompida a qualquer momento da utilização do recurso compartilhado sem risco de afetar a computação final.

Resource Sharing

- Conclusão:
  - Os algoritmos estudados no capítulo anterior não são ideais para escalonar o acesso de tarefas de tempo real aos recursos compartilhados
  - Novos métodos são necessários.

Resource Sharing

- Neste capítulo:
  - Como os recursos críticos podem ser compartilhados
  - Problemas com compartilhamento
  - Protocolos de compartilhamento de recursos
  - Como os escalonadores estudados anteriormente podem ser melhorados para inserir dependência entre tasks

Compartilhamento de recursos entre tarefas de tempo real

- Muitas aplicações necessitam de compartilhar recursos entre elas
- Alguns recursos necessitam ser utilizados de maneira exclusiva por uma determinada task
- Ou seja, uma task não pode liberar esse recurso de maneira arbitrária a qualquer momento, somente após liberar este recurso
- Caso isso ocorra, o recurso pode estar corrompido

Compartilhamento de recursos

- Estes recursos são denominados :
  - *Non preemptable resources*
  - *Critical resources*

- Exemplo:
  - Arquivos
  - Dispositivos
  - Estruturas de dados

Compartilhamento de recursos

- A CPU é um exemplo de outro tipo de recurso, denominado *Serially Reusable Resource*
  - Características:
    - A task que possui acesso ao recurso utiliza-o de maneira exclusiva
    - O acesso da task pode ser interrompido se retomada sem nenhuma perda de informação

- Acesso a *non preemptable resources* também é exclusivo

Compartilhamento de recursos

- Problema:
  - Uma task que utiliza um *non preemptable resource* não pode ser interrompida enquanto estiver utilizando o recurso.
  - Ocorre inconsistência e pode levar o sistema à falha.
- Conclusão:
  - Algoritmos comumente utilizados para recursos serializados (EDF e RMA) podem não atender de maneira satisfatória ao compartilhamento de *non preemptable resources*

Inversão de Prioridade

- Em sistemas tradicionais:
  - Técnicas: Semáforos, locks, mutexes e monitores.

- Utilização destas técnicas pode ocasionar inversão de prioridades

Inversão de Prioridade

- Conceito de inversão simples de prioridade:
  - Quando uma tarefa de baixa prioridade já está acessando e travando o recurso, uma tarefa de maior prioridade necessitando acessar o mesmo recurso precisa esperar a liberação por parte de uma tarefa de menor prioridade.
  - A tarefa de maior prioridade se mantém bloqueada enquanto a tarefa de menor prioridade utiliza o recurso
  - Neste caso é dito que a tarefa está com uma **inversão simples de prioridade**

Inversão de Prioridade

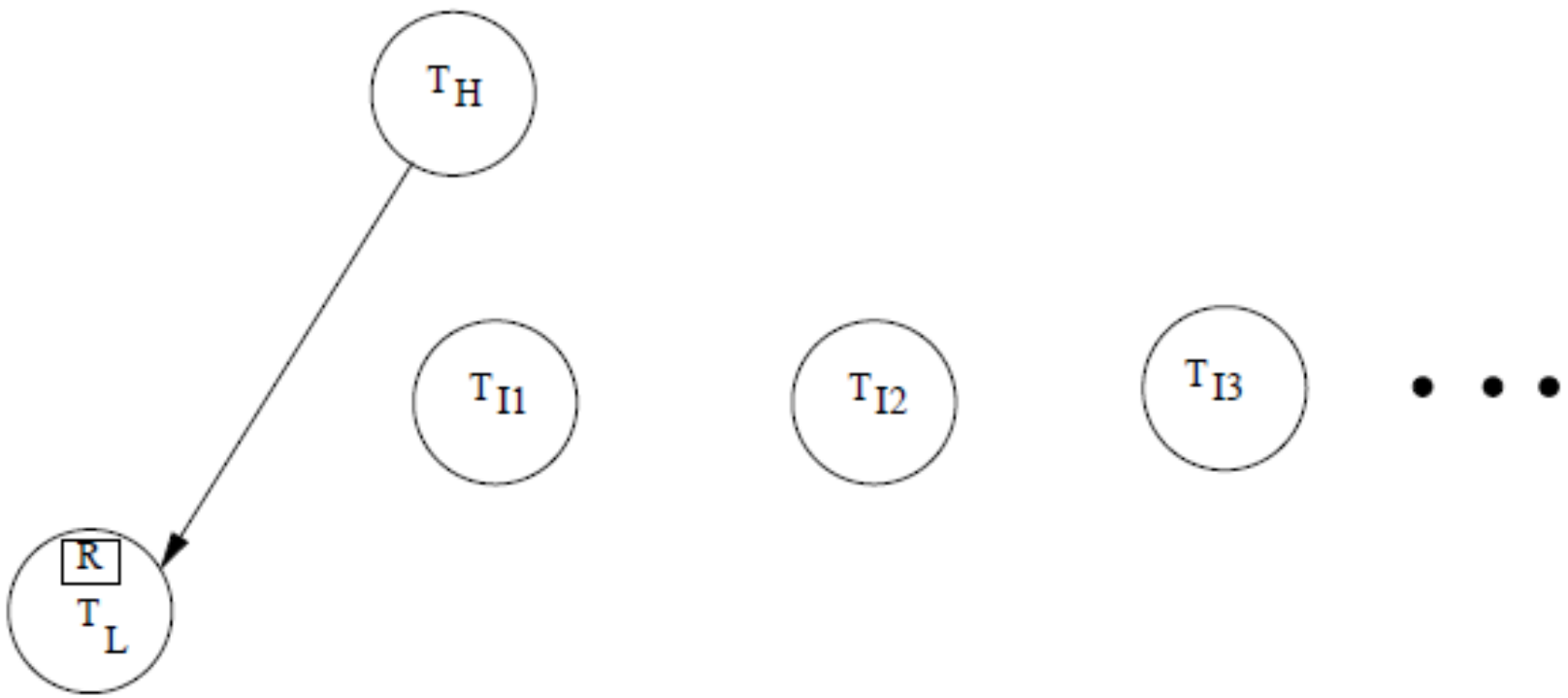
- Algumas Consequências:
  - O impacto de tempo está diretamente relacionado ao tempo de utilização do recurso por parte da task de menor prioridade
  - Maior delay para execução da tarefa de maior prioridade
- Soluções alternativas:
  - Tempo de bloqueio do recurso deve ser o menor possível
  - Boas práticas de programação

Inversão de Prioridade

- Inversão ilimitada de prioridade
  - Uma task de maior prioridade aguarda um recurso que está sendo utilizado por uma task de menor prioridade. Enquanto isso, uma task de prioridade intermediária interrompe a task de menor prioridade da CPU repetidas vezes. Como resultado, a task de menor prioridade não consegue completar a utilização do recursos crítico e a task de maior prioridade aguarda indefinidamente pelo recurso.

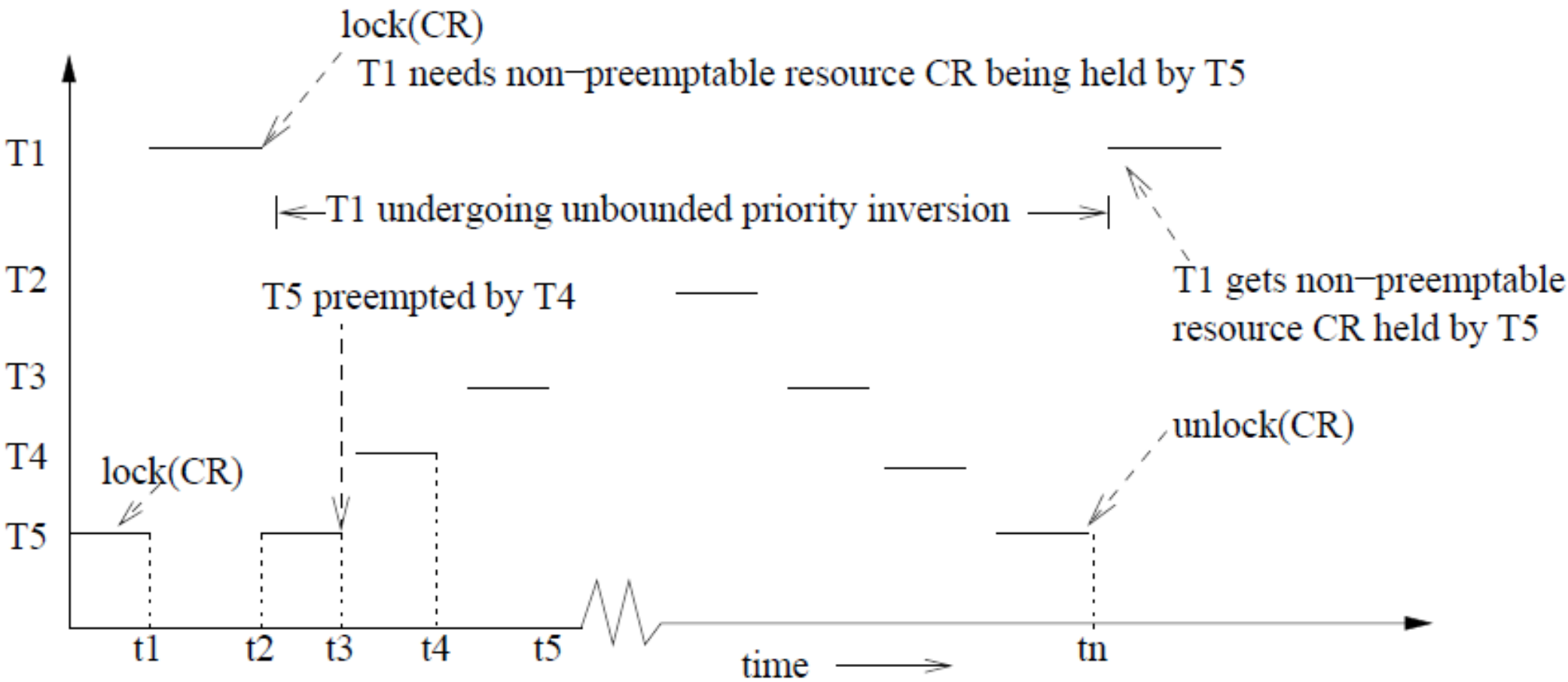
Inversão de Prioridade

- Exemplo



Inversão de Prioridade

- Exemplo



Compartilhamento de recursos

- Métodos para evitar:
  - Aumentar a prioridade da task de menor prioridade até igualar a da task que está aguardando o recurso.
  - É a idéia central do protocolo PIP – *Priority Inheritance Protocol*

Priority Inheritance Protocol - PIP

- Técnica simples de compartilhamento de recursos entre tasks sem incorrer em inversões ilimitadas de prioridades
- Boa parte dos sistemas operacionais de tempo real comerciais possuem suporte a este tipo de protocolo
- Protocolo porposto por *Sha* e *Rajkumar*.

Priority Inheritance Protocol - PIP

- O princípio deste protocolo é que mesmo que uma task sofra de inversão de prioridade, a prioridade da task de menor prioridade, que está travando o recurso, subirá através do mecanismo de herança de prioridade.

- Este método permite que o recurso seja utilizado tão logo seja liberado.

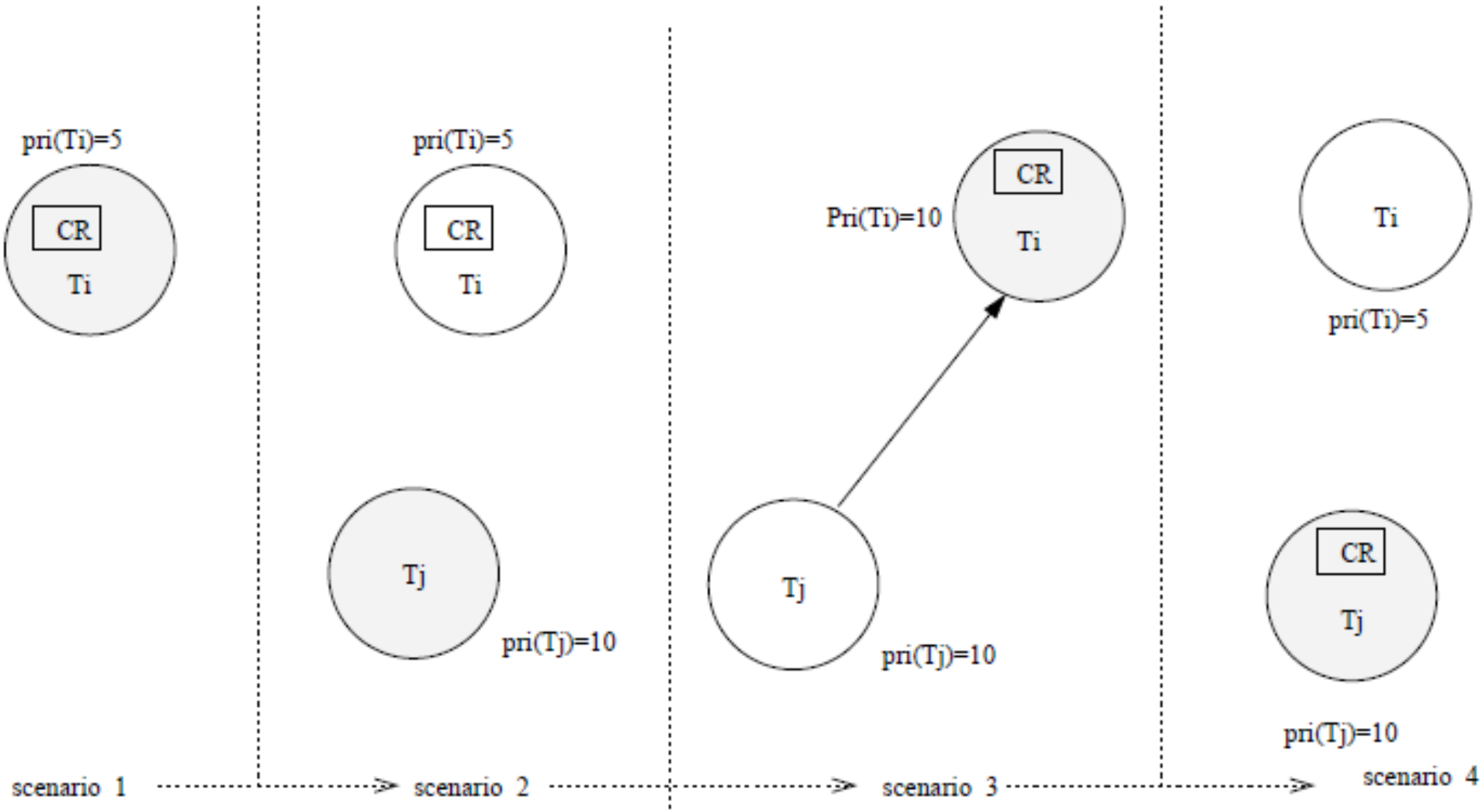
Priority Inheritance Protocol - PIP

- Outras considerações:
  - Caso o recurso seja compartilhado entre inúmeras tasks, todas as tasks irão herdar a prioridade da maior task.
  - Ao liberar o recurso compartilhado a tarefa de menor prioridade retorna ao seu nível anterior.

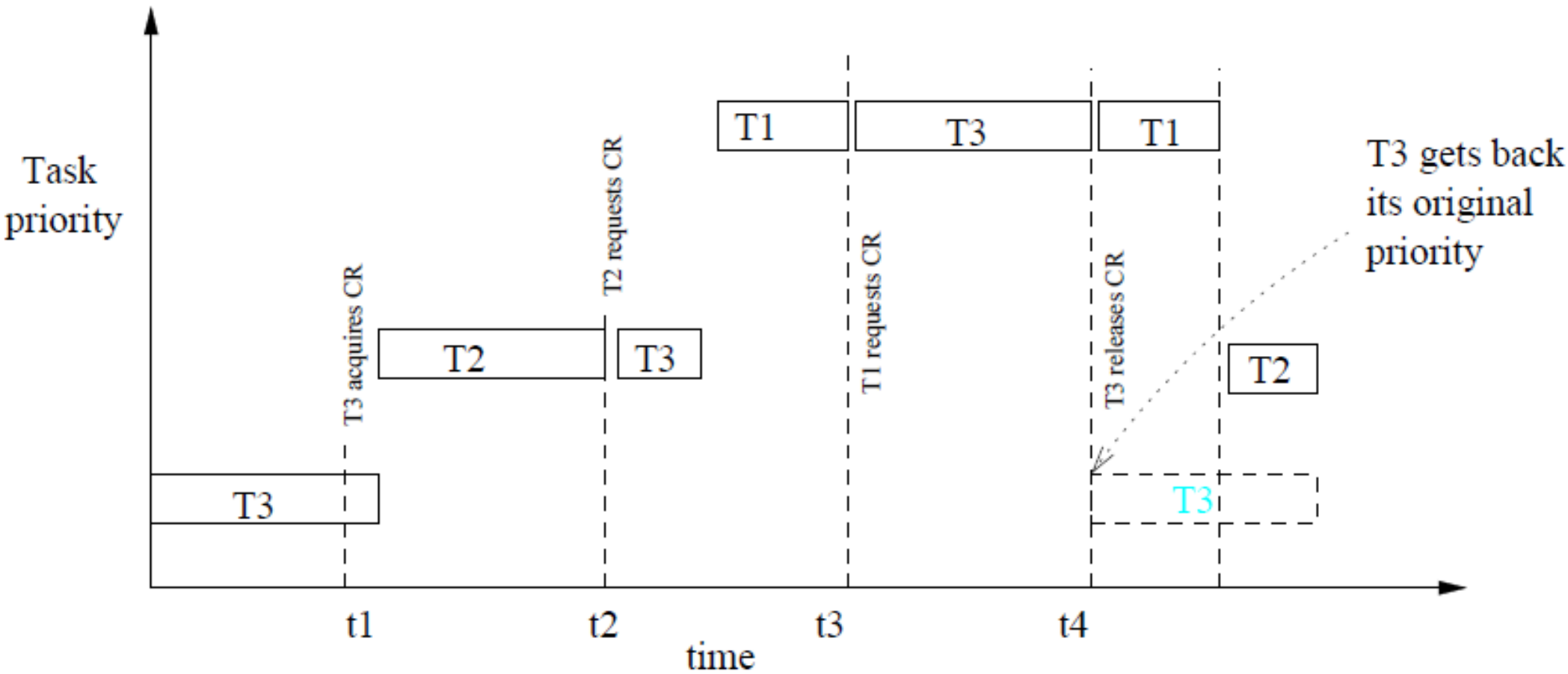
Priority Inheritance Protocol - PIP

- Algoritmo:
  - SE é requerido um recurso E está livre, ENTÃO
    - Pegue-o
  - SE é requerido um recurso E está em uso por maior prioridade, ENTÃO
    - Aguarde o recurso
  - SE é requerido um recurso E está em uso por menor prioridade, ENTÃO
    - Aguarde o recurso
    - Task de menor prioridade que está com o recurso obtém a prioridade desta task.

Priority Inheritance Protocol - PIP



Priority Inheritance Protocol - PIP



Priority Inheritance Protocol - PIP

- Deadlock: O protocolo PIP não realiza nenhum procedimento para evitar o *deadlock*.

Priority Inheritance Protocol - PIP

- Exemplo:
  - Duas tasks T1 e T2 necessitam acessas doir recursos compartilhados CR1 e CR2.
  - T1 possui maior prioridade.
  - T1: Lock CR1, Lock CR2, Unlock CR2, Unlock CR2
  - T2: Lock CR2, Lock CR1, Unlock CR1, Unlock CR2

Priority Inheritance Protocol - PIP

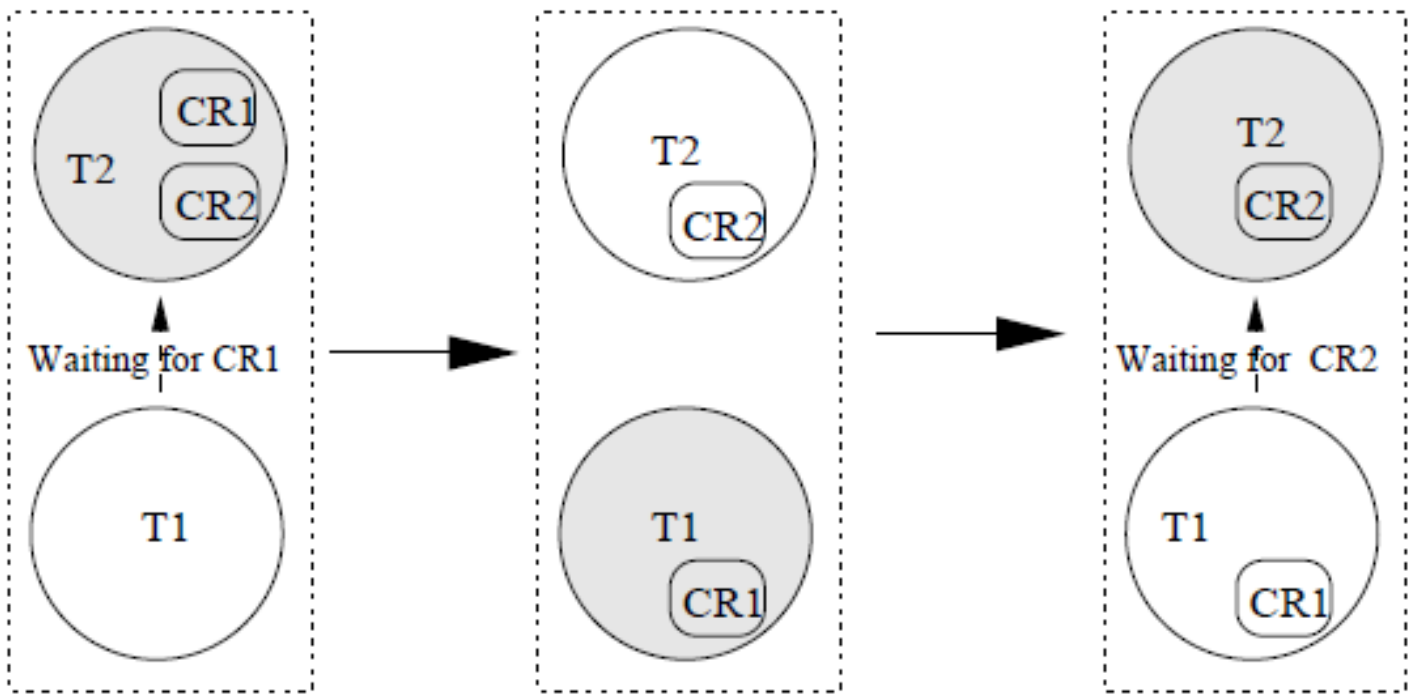
- Exemplo:
  - Fluxo de Execução
  - T2 inicia a execução e trava CR2.
  - T1 interrompe T2 e trava CR1 e tenta travar CR2
  - A prioridade de T2 aumenta e volta a ser executada.
  - T2 tenta travar CR1
  - As duas tasks encontram-se em *deadlock*

Priority Inheritance Protocol - PIP

- Chain Blocking: se a cada momento necessitar de um recurso, ela necessitar sofrer inversão de prioridade.
- Se uma tarefa necessitar de *n* recursos, ela poderá sofrer inversão de prioridade *n* vezes.

Priority Inheritance Protocol - PIP

- Chain Blocking



T2 Executing
T1 Blocked
Snapshot 1

T1 Executing
T2 Blocked
Snapshot 2

T2 Executing
T1 Blocked
Snapshot 3

Próxima Aula

- Highest Locker Protocol (HLP)
- Priority Ceiling Protocol (PCP)
- Gerenciamento de dependências das tasks