

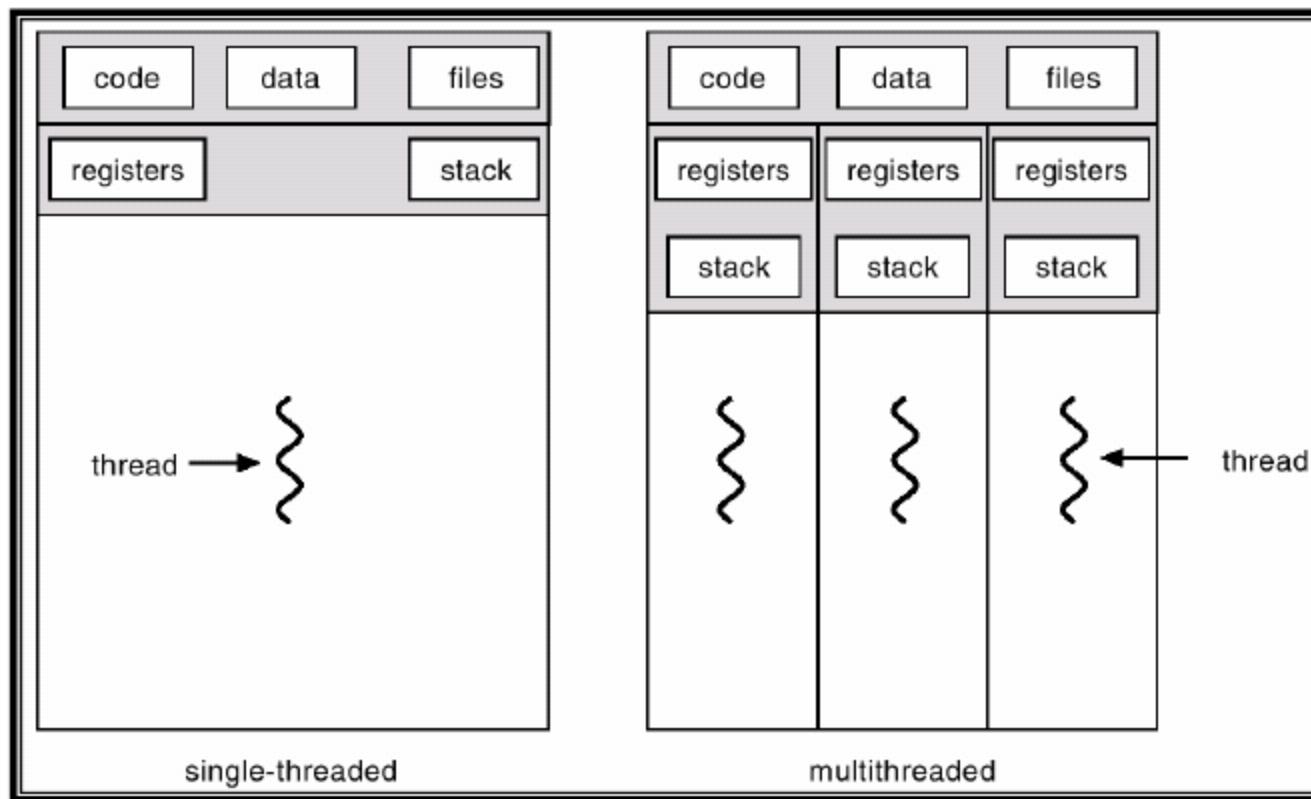
Sistemas Hardware-Software

Aula 22 – Programação concorrente II

2020 – Engenharia

Igor Montagner, Fábio Ayres [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br)

Processos e threads



Processos e threads

- Processos
 - Comunicação entre processos
 - **Possível distribuir em várias máquinas**
- Threads
 - Mais barato de criar e destruir
 - Sempre pertencem a um único processo
 - **Sincronização para acessar recursos compartilhados**

Troca de contexto ocorre de maneira igual nos dois casos!

POSIX threads

O padrão POSIX define também uma API de threads (*pthread*) que inclui

- Criação de threads
- Sincronização (usando semáforos)
- Controle a acesso de dados (usando mutex)

Problemas limitados por CPU – Parte 4

- Roda tão rápido quanto a CPU puder
- Otimização de cache vale muito
- Faz pouca entrada/saída
 - Interage pouco com o sistema
- Pode ou não ter partes paralelas

Problemas limitados por CPU – Parte 4

- Dividimos um problemas em partes
- Cada parte é independente (em sua maioria)
- Juntamos os resultados no fim
- Pouca ou nenhuma sincronização

Tarefas paralelas (CPU-bound)

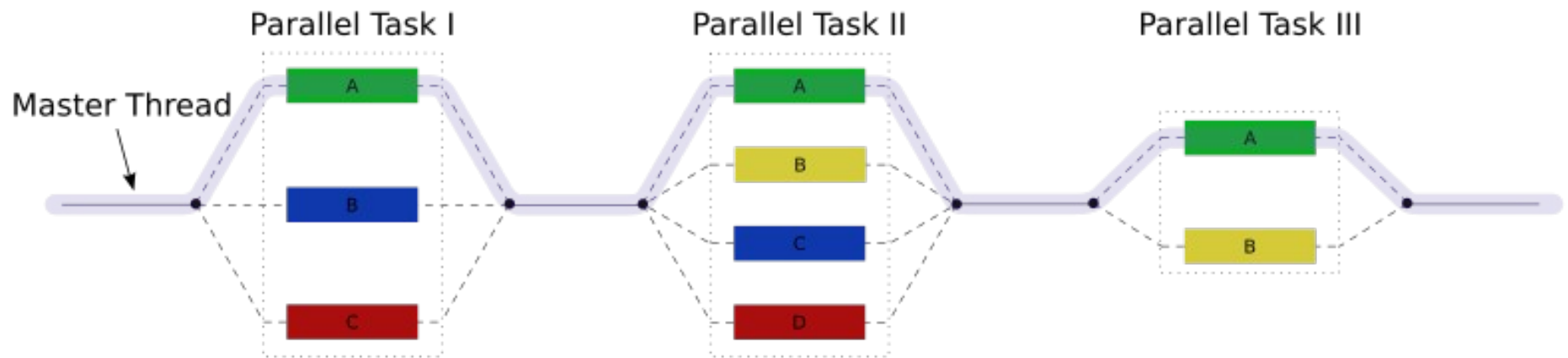
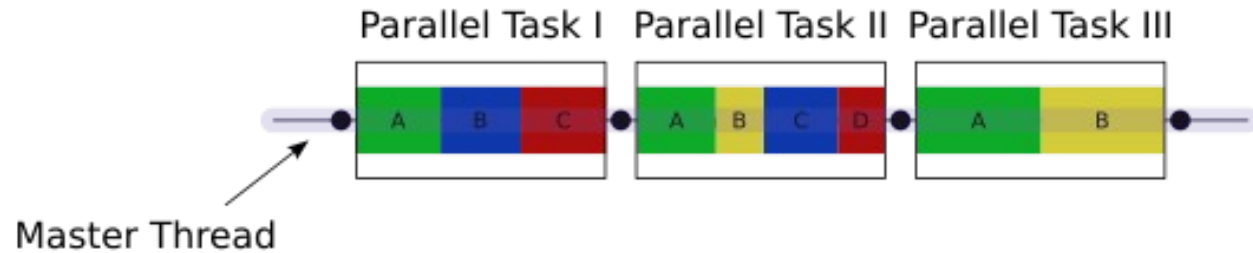


Figura: https://en.wikipedia.org/wiki/File:Fork_join.svg

Atividade prática

- Exercício da parte 0 do handout

Race condition

saída do programa depende da ordem de execução das threads

- Acessos concorrentes a um recurso, com pelo menos uma escrita
- Nossa atividade tem esse problema!

Região crítica

parte(s) do programa que só podem ser rodadas por uma thread por vez

- Nenhum paralelismo é permitido em regiões críticas
- Evita acessos concorrentes, mas é gargalo de desempenho

Mutex (Mutual Exclusion)

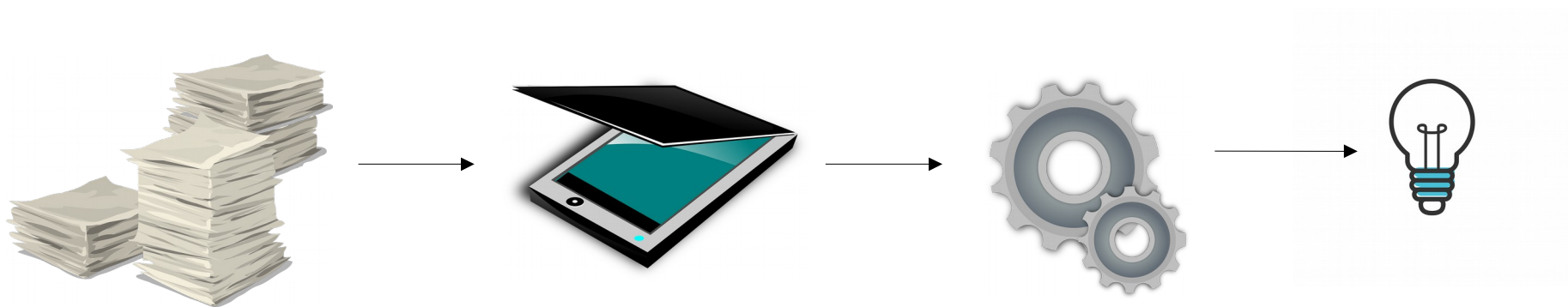
Primitiva de sincronização para criação de regiões de exclusão mútua

- Lock – se estiver destravado, trava e continua
 - – se não espera até alguém destravar
- Unlock – se tiver a trava, destrava
 - – se não tiver retorna erro

Atividade prática (30 minutos)

- Vamos usar mutexes para proteger o acesso a variável de soma.

Problema - leitura de informações



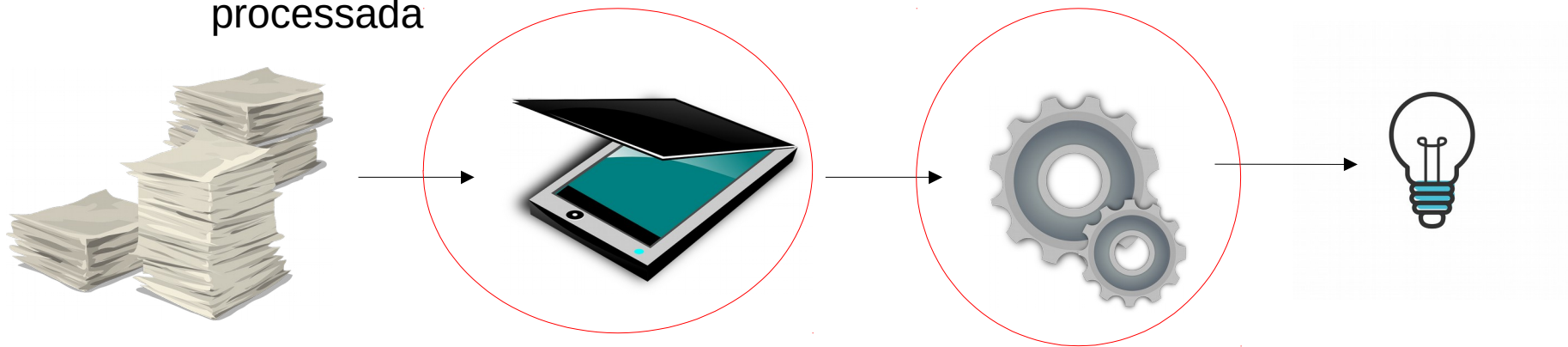
Exemplo 1 – produtor consumidor

Dois conjuntos de threads

- Produzem tarefas a serem executadas
pode depender de um recurso compartilhado
controlar tamanho das tarefas
- Consomem as tarefas e as executam. Cada
consumidor não depende dos produtores nem de
outros consumidores.

Exemplo 1 – produtor consumidor

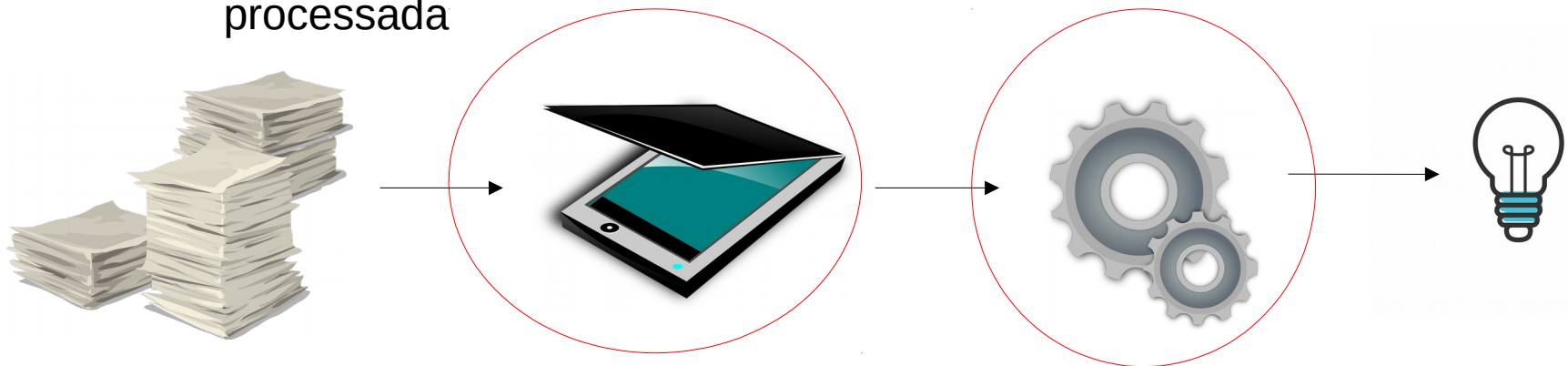
Produtor: Escaneia e
devolve imagem a ser
processada



Consumidor: transforma
imagem em informação

Exemplo 1 – produtor consumidor

Produtor: Escaneia e devolve imagem a ser processada



Consumidor: transforma imagem em informação

Sincronização

1. Consumidor: espera produtor enviar item
2. Produtor: cria item e avisa Consumidor

Semáforos

“Inteiro especial cujo valor nunca pode ser negativo” (man sem_overview)

Duas operações atômicas:

- POST – incrementa o valor.
- WAIT – se maior que zero, decrementa
– se não espera

Atividade prática (30 minutos)

- Vamos usar semáforos para uma situações de sincronização simples:
 - Rendez-vous

Problemas resolvidos por sincronização

- E se precisarmos compartilhar dados?
 - Tarefas preenchendo um vetor
 - Leitura/escrita em variável
- Tarefas heterogêneas
 - Fazem coisas diferentes
 - Mas usam mesmos dados

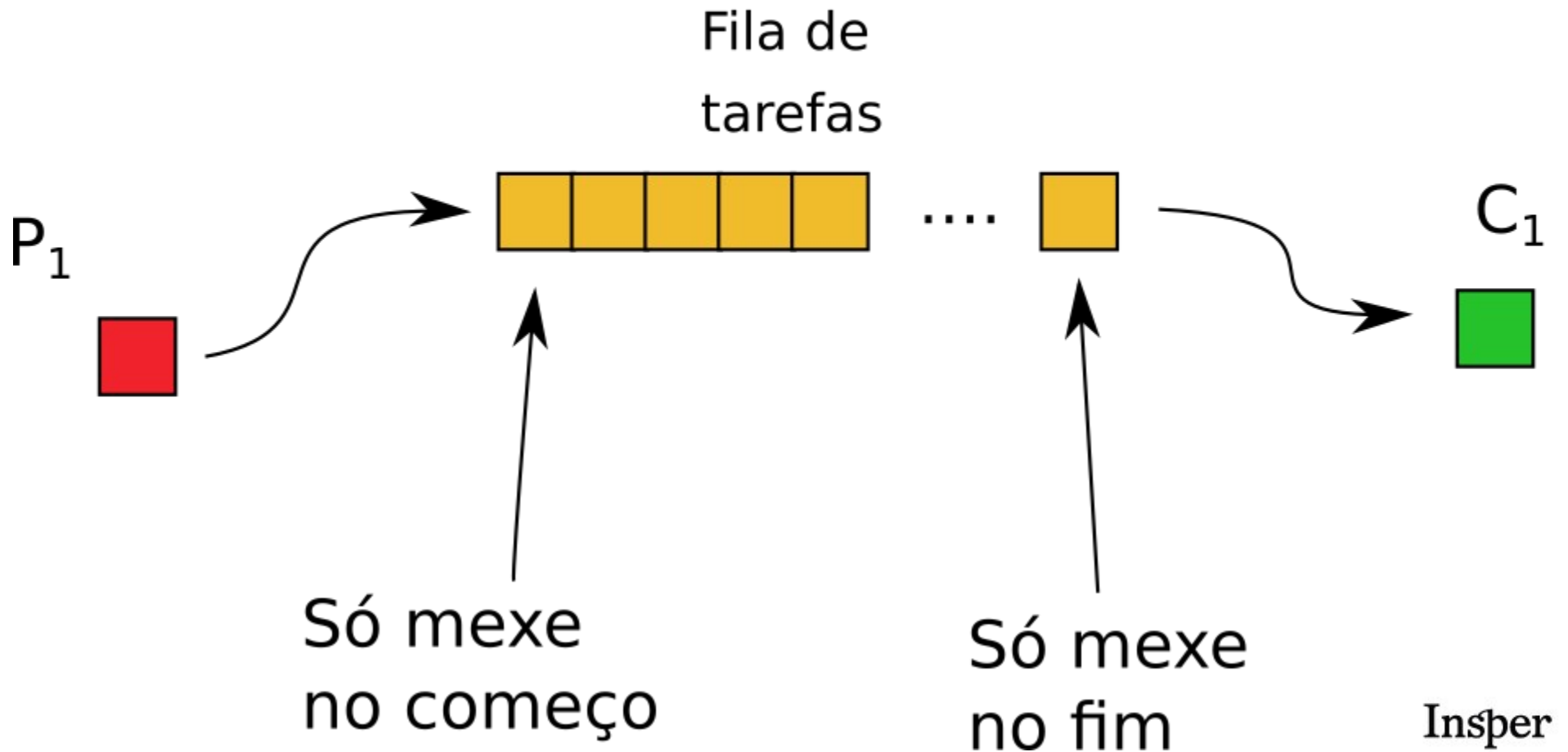
Soluções famosas

- Produtor Consumidor
- Leitores Escritores

Modelo produtor-consumidor 1-1

Produtor

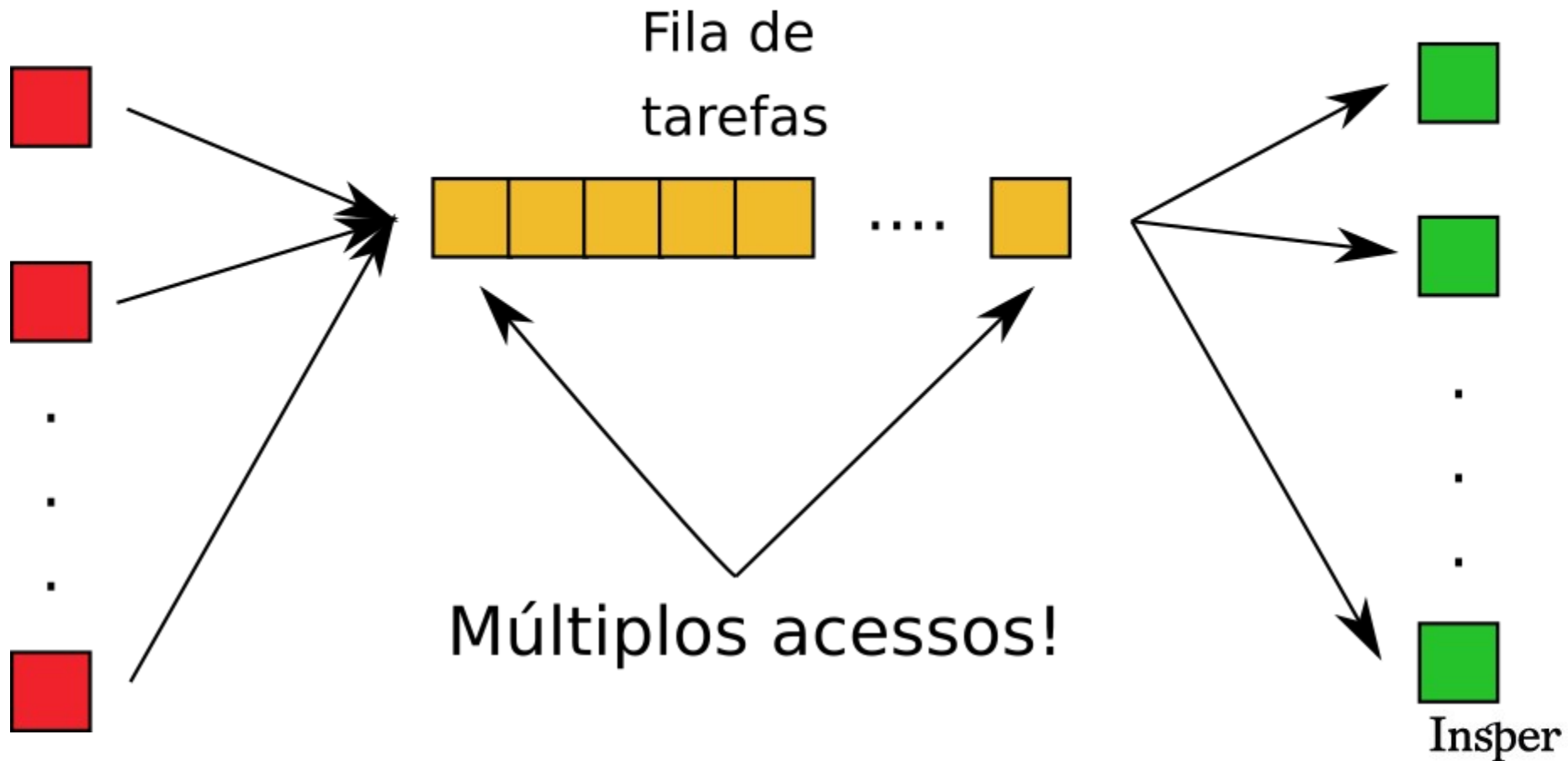
Consumidor



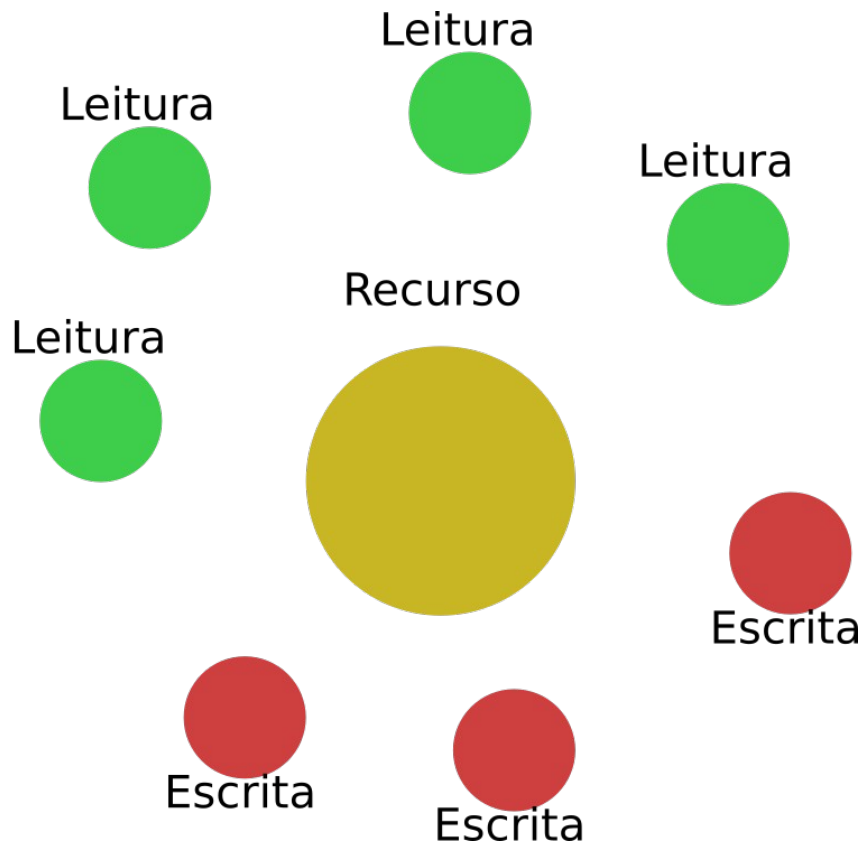
Modelo produtor-consumidor M-N

Produtor

Consumidor



Exemplo 2 - Leitores e Escritores



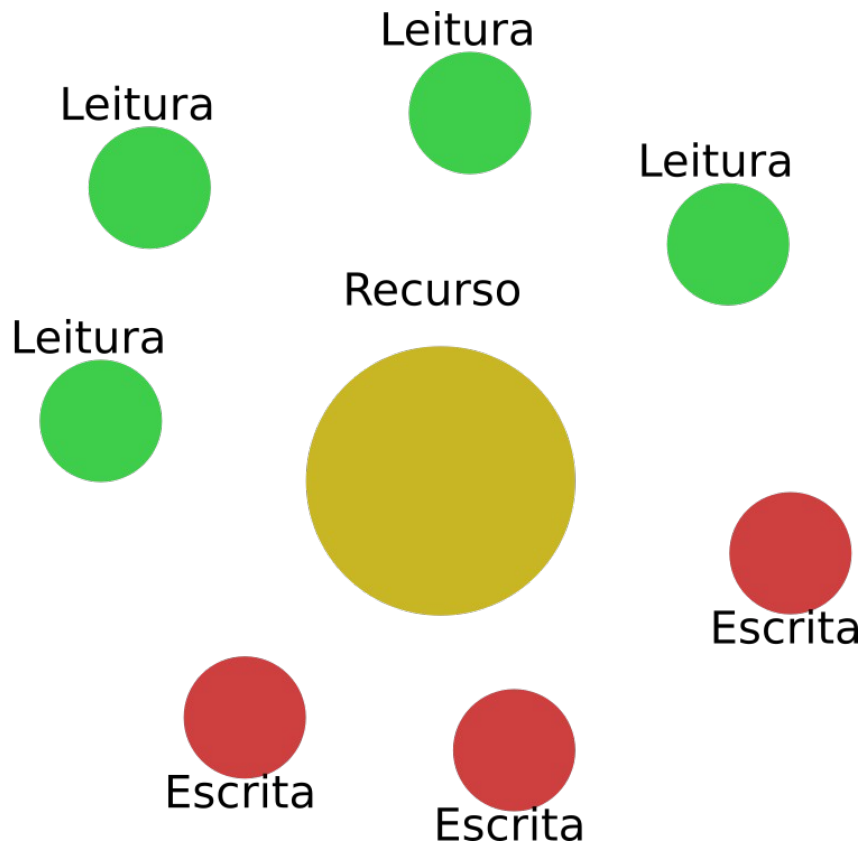
Um recurso compartilhado por vários processos que

- **Leem** o estado do recurso
- **Modificam** o estado do recurso

Com as restrições

- **Leituras** podem ser feitas simultaneamente
- **Escritas** necessitam de acesso exclusivo

Exemplo 2 - Leitores e Escritores

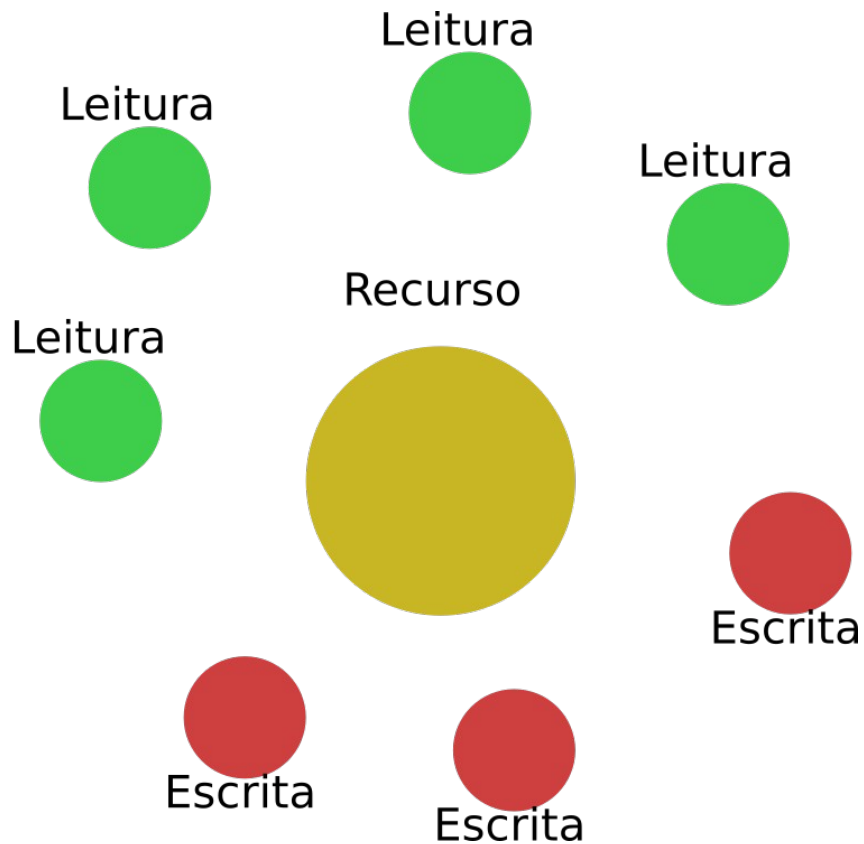


Problemas:

- O quê acontece se a frequência de leitores é alta e a frequência de escritores é baixa?
- E se for o oposto?

Starvation: situação onde uma thread (ou grupo de threads) nunca consegue acesso a um recurso.

Exemplo 2 - Leitores e Escritores



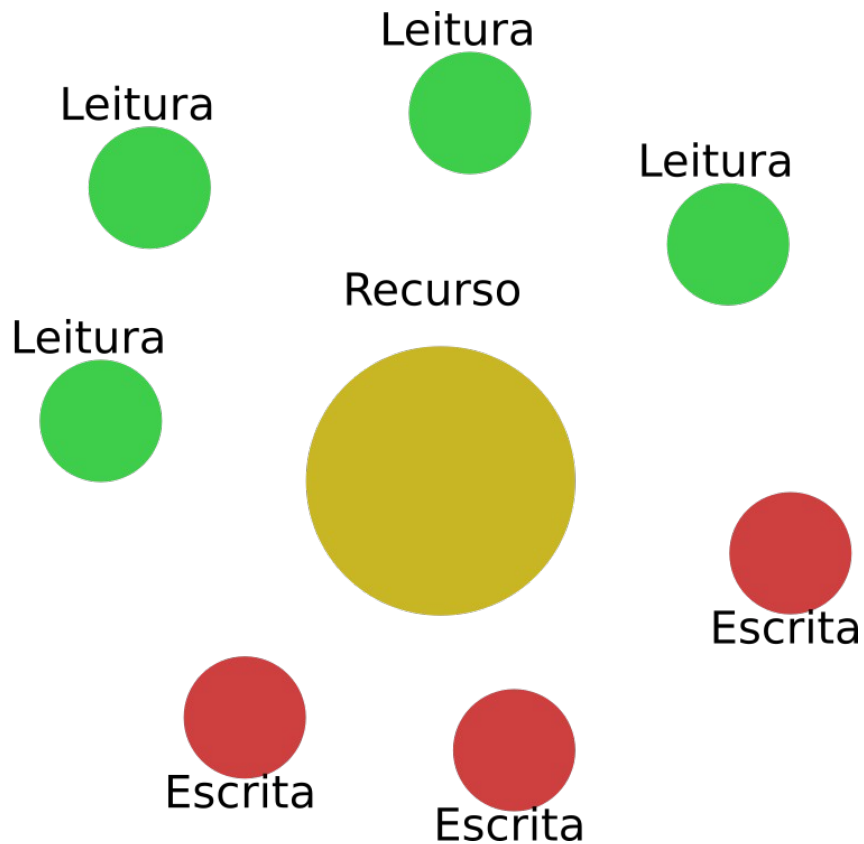
Solução mutex:

1. Trava
2. Lê ou escreve
3. Destrava

Solução **horrível!**

- Só há conflito em escritas, mas trava sempre
- Basicamente sequencial
- Não trata starvation
- Não define prioridade

Exemplo 2 - Leitores e Escritores



- 1) Leitores tem preferência, não há ordem garantida;
- 2) Escritores tem preferência, não há ordem garantida;
- 3) Os acessos são feitos por ordem de chegada, mas se há vários leitores em seguida eles podem executar simultaneamente;

Insper

www.insper.edu.br