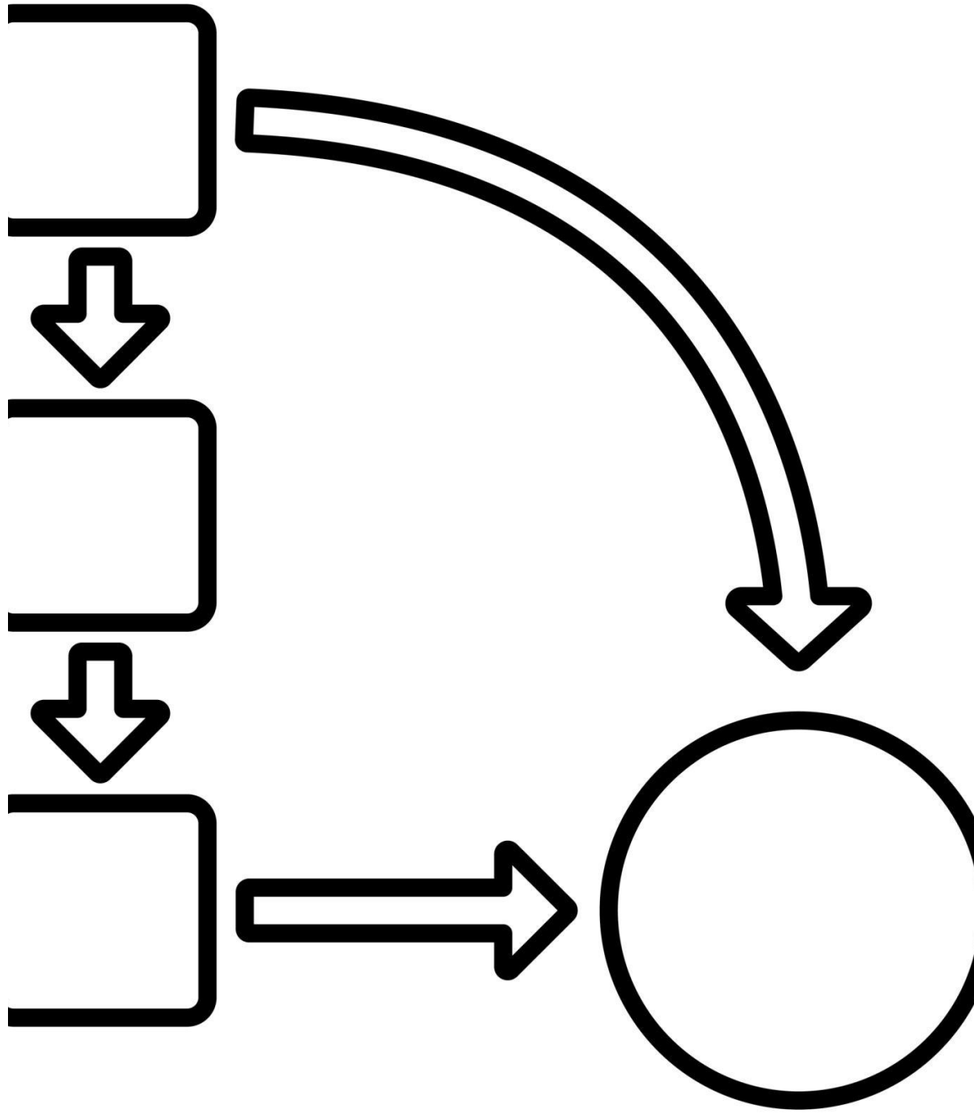
Modelos de Sincronização e Comunicação entre Processos

Explorando os fundamentos da concorrência e comunicação em sistemas computacionais modernos



Objetivos da Aula

Concorrência

Entender conceitos de concorrência e seções críticas em sistemas computacionais

Mecanismos

Conhecer mecanismos de sincronização: mutex, semáforos e barreiras

Comunicação

Diferenciar memória compartilhada e troca de mensagens entre processos

Implementação

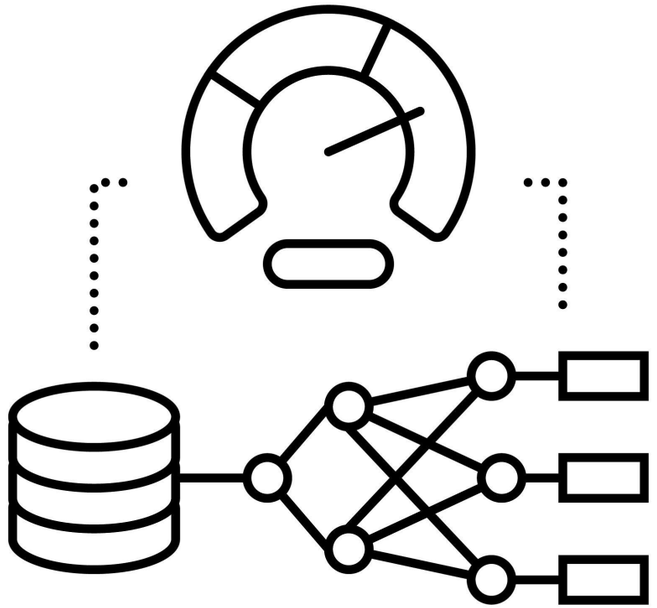
Implementar exemplos simples de comunicação entre processos

Introdução aos Sistemas Paralelos

Sistemas modernos executam vários processos em paralelo, criando a necessidade de coordenação eficiente entre eles. Esta execução simultânea permite:

- Compartilhar recursos como variáveis, arquivos e impressoras
- Trocar informações através de mensagens e dados
- Maximizar a utilização dos recursos do sistema

⊗ **Problema Principal:** Condição de corrida (race condition) → resultados incorretos quando processos acessam o mesmo recurso simultaneamente



Conceitos Básicos

1

Processo

Programa em execução com seu próprio espaço de memória isolado e recursos dedicados

2

Thread

Unidade menor de execução dentro de um processo que compartilha memória com outras threads

3

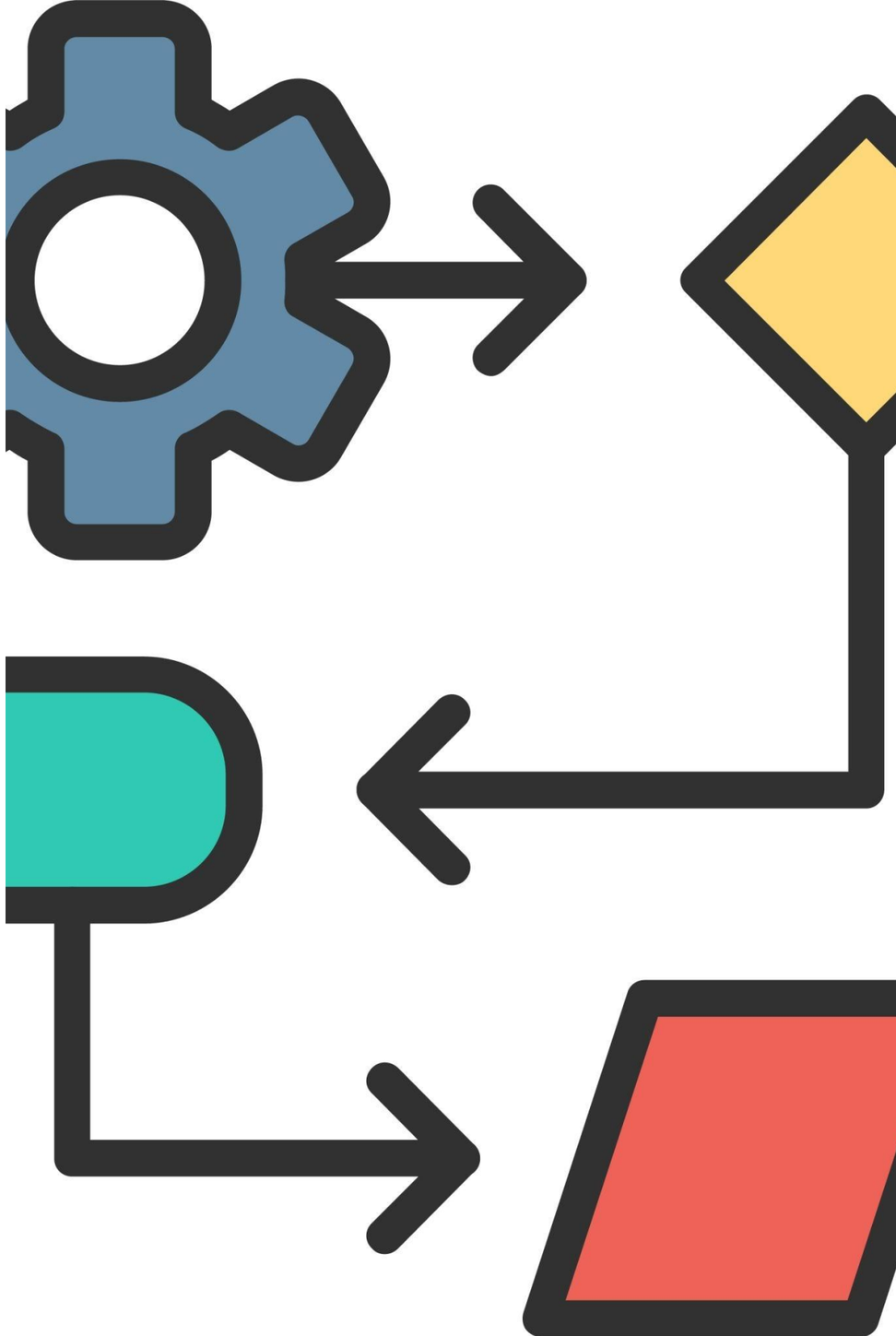
Concorrência

Execução de múltiplos processos de forma paralela ou intercalada no tempo

4

Seção Crítica

Parte do programa onde há acesso a recurso compartilhado que precisa ser protegido



Mecanismos de Sincronização

Mutex / Locks

Garantem exclusão mútua, permitindo que apenas um processo acesse a seção crítica por vez. Fundamental para evitar condições de corrida.

Semáforo

Controla múltiplos acessos a um recurso limitado, funcionando como um contador que permite N processos simultâneos.

Barreiras

Sincronização coletiva onde todos os processos devem chegar a um ponto específico antes de continuar a execução.

Monitores

Abstração de alto nível para gerenciar variáveis compartilhadas com operações atômicas e condições de espera.

Problemas Clássicos de Sincronização

Produtor-Consumidor

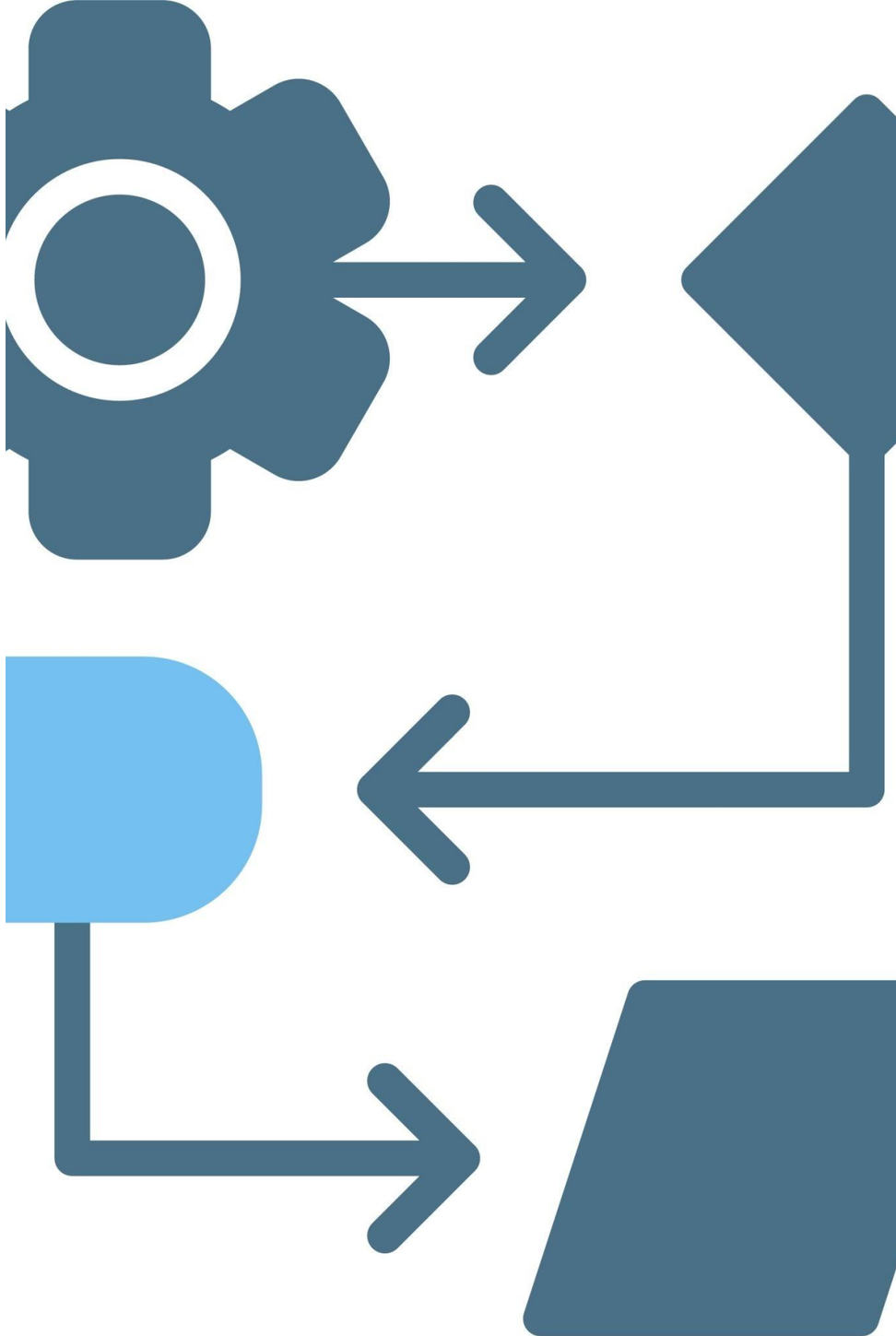
Sincronizar processos que produzem dados com aqueles que os consomem, gerenciando um buffer compartilhado de tamanho limitado.

Filósofos Jantando

Evitar deadlock ao acessar recursos limitados (garfos) onde cada filósofo precisa de dois recursos para comer.

Leitores e Escritores

Múltiplos leitores podem acessar simultaneamente, mas escritores precisam de acesso exclusivo ao recurso compartilhado.



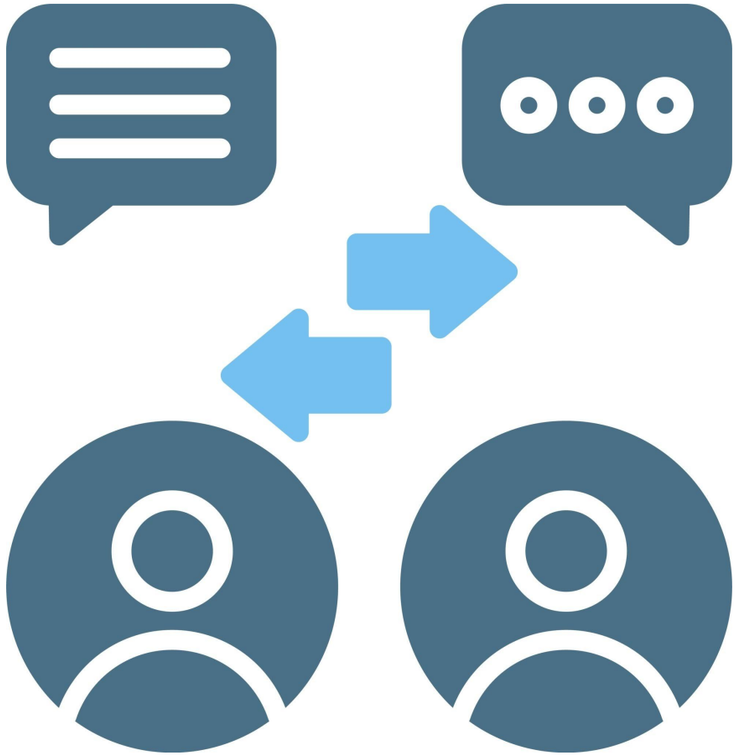
Modelos de Comunicação Entre Processos

Memória Compartilhada

- Comunicação rápida e eficiente
- Exige sincronização cuidadosa
- Processos acessam mesma região de memória
- Ideal para sistemas com alta frequência de comunicação

Troca de Mensagens

- Comunicação mais segura
- Via filas, pipes ou RPC
- Isolamento entre processos
- Melhor para sistemas distribuídos



Problemas Comuns em Sistemas Concorrentes

Deadlock

Processos esperando indefinidamente uns pelos outros, criando um ciclo de dependências que impede o progresso.

Starvation

Processo nunca consegue acessar o recurso necessário devido à priorização constante de outros processos.

Race Condition

Resultado final depende da ordem de execução dos processos, levando a comportamentos imprevisíveis.

⚠ Estes problemas podem causar falhas críticas em sistemas de produção e devem ser cuidadosamente evitados através de design adequado.

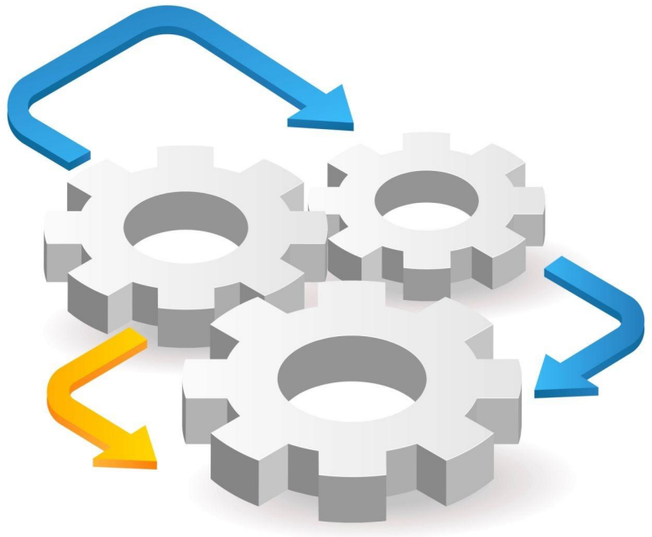
Demonstração: Condição de Corrida

Problema sem Sincronização

```python
import multiprocessing

def incrementa(contador):
    for _ in range(1000):
        contador.value += 1

contador = multiprocessing.Value('i', 0)
p1 = multiprocessing.Process(target=incrementa, args=(contador,))
p2 = multiprocessing.Process(target=incrementa, args=(contador,))
p1.start(); p2.start()
p1.join(); p2.join()
print("Valor esperado: 2000")
print("Valor obtido:", contador.value)
```



Este código demonstra como a falta de sincronização pode levar a resultados incorretos.

Resultado: O valor final será menor que 2000 devido à condição de corrida!

Soluções Práticas de Sincronização

01

Exclusão Mútua com Lock

0

2

0

3

```
lock = multiprocessing.Lock()
contador = multiprocessing.Value('i', 0)
p1 = multiprocessing.Process(target=incrementa, args=(contador, lock))
p2 = multiprocessing.Process(target=incrementa, args=(contador, lock))
```

Estas implementações em Python demonstram como aplicar os conceitos de sincronização na prática, garantindo execução segura e previsível em ambientes concorrentes.

Soluções Práticas de Sincronização

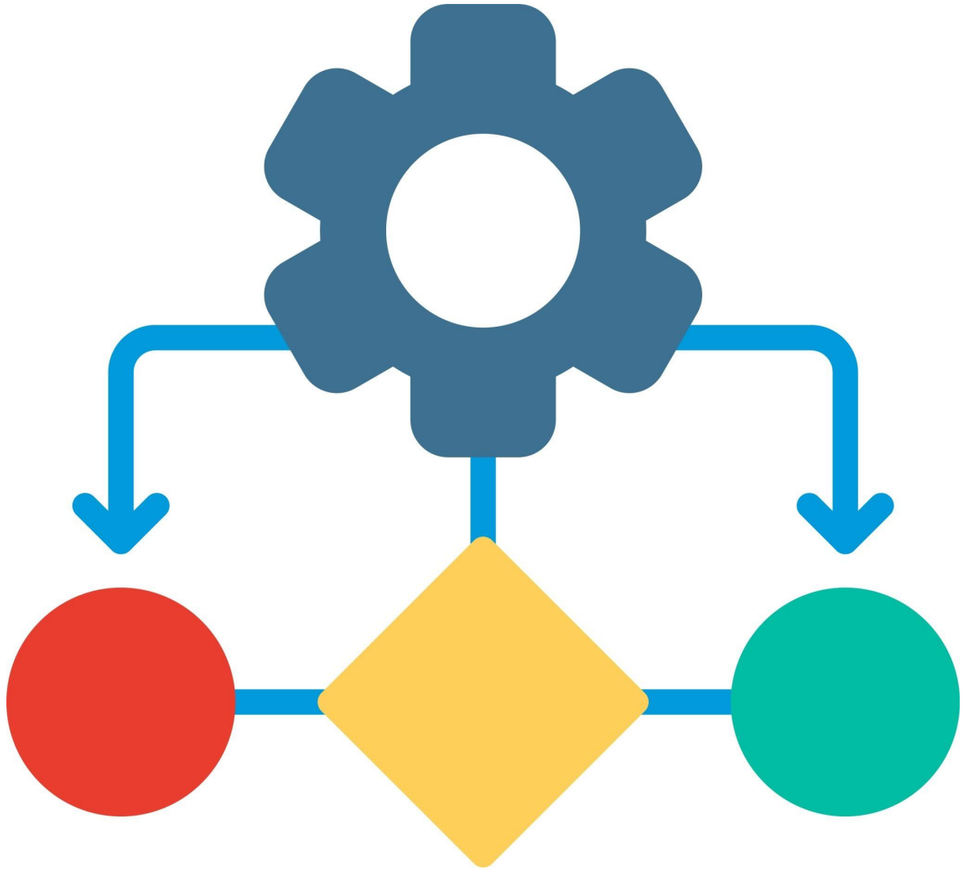
2 Comunicação por Fila

0

3

```
fila = multiprocessing.Queue()
# processos enviam mensagens para a fila
```

Estas implementações em Python demonstram como aplicar os conceitos de sincronização na prática, garantindo execução segura e previsível em ambientes concorrentes.



7.4 Semáforo (acesso limitado)

```
semaforo = multiprocessing.Semaphore(2)  # só 2 processos por vez
```