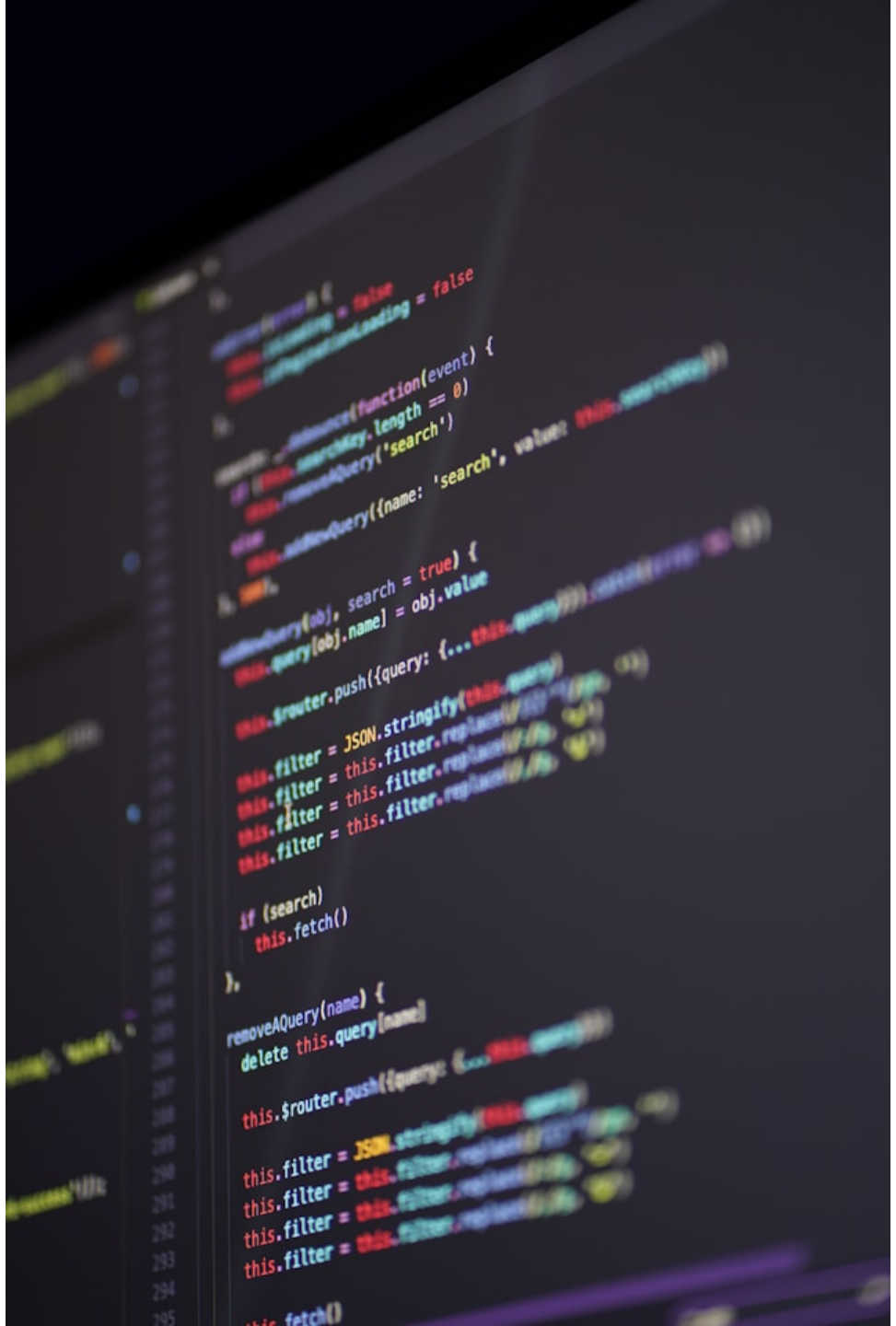


5954025 - Sistemas Distribuídos

Aula 11a - Exclusão Mútua

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

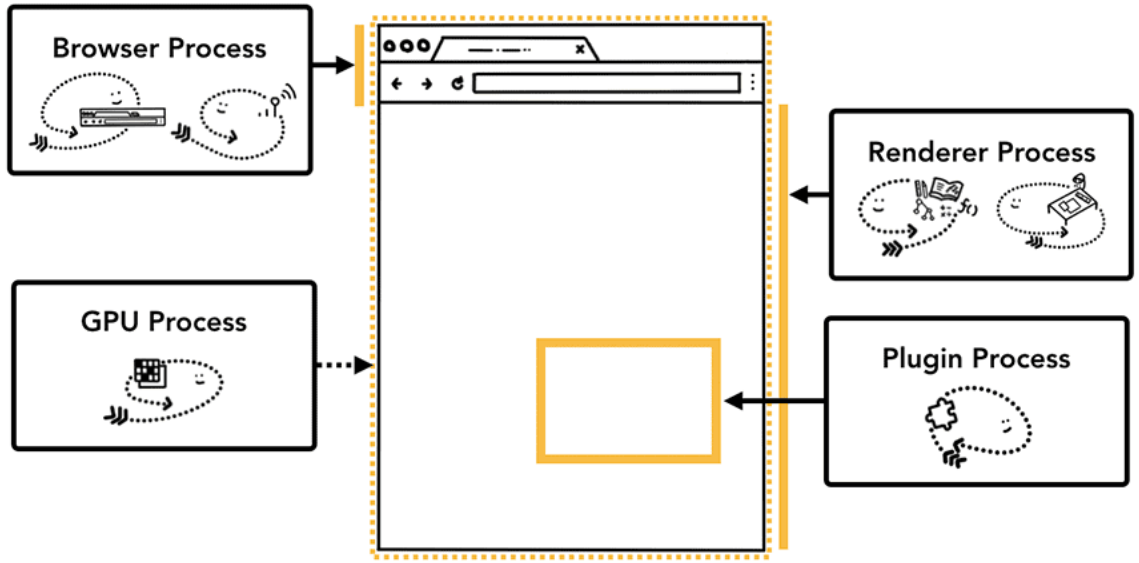
Objetivos de Aprendizagem

- Entender por que múltiplos processos rodando simultaneamente causam desafios de sincronização.
- Diferenciar concorrência em um único computador e em sistemas distribuídos.
- Compreender a ideia de seção crítica e exclusão mútua.
- Analisar algoritmos descentralizados: votação e token.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Relembrando: Multitarefa

- Os sistemas operacionais permitem que múltiplas computações ocorram **concorrentemente** em um único sistema computacional.
- Processo como unidade de **gerenciamento** e **proteção**.



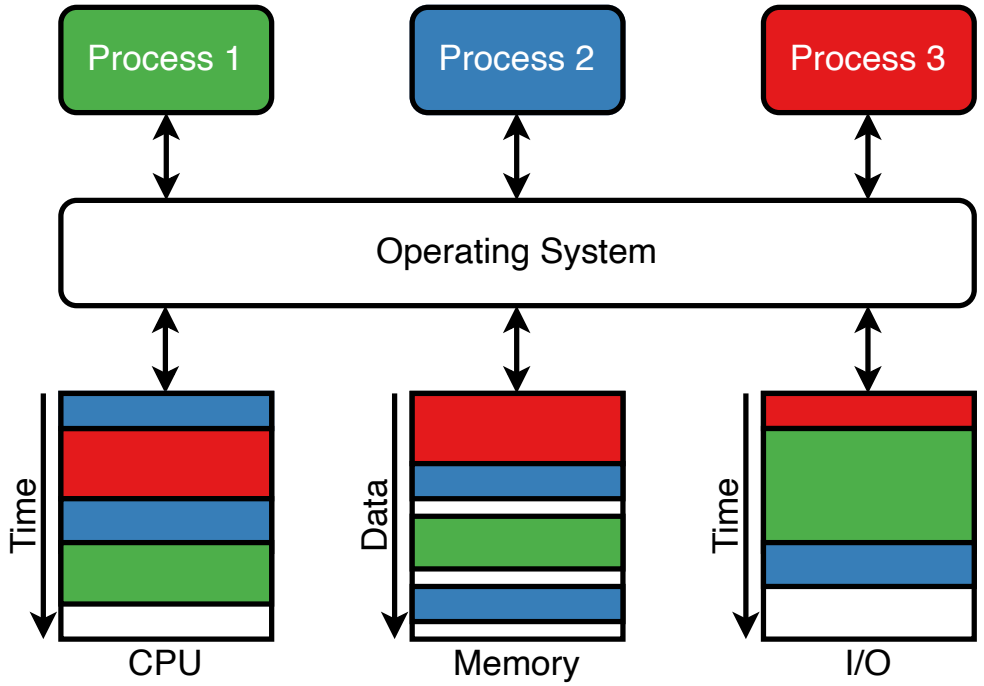
> Na imagem: Processos apontando para diferentes partes da interface do usuário (UI) do navegado.
>
> Fonte: Google Developers

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Relembrando: Concorrência

- Recurso Compartilhado: Dados ou equipamentos acessados por vários processos.
- Mesmo em um único CPU core: ilusão de simultaneidade.
- Vários processos executam **simultaneamente** em um sistema.

> Na Imagem: SO gerenciando memória, CPU e I/O para três processos diferentes. Fonte: OER OS



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Relembrando: Condição de Corrida (Race Conditions)

Se não houver um mecanismo de controle ao acesso à variável **next_available_pid**, processos-filhos Processo A e Processo B podem ter o **mesmo** **pid**.

Tempo	Processo A		Processo B
T_1	pid_t child = fork();		pid_t child = fork();
T_2	request pid		request pid
T_3	$\rightarrow$	next_available_pid = 2615	$\leftarrow$
T_4	return 2615		return 2615
T_5	child = 2615		child = 2615

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Problema: Lost Update

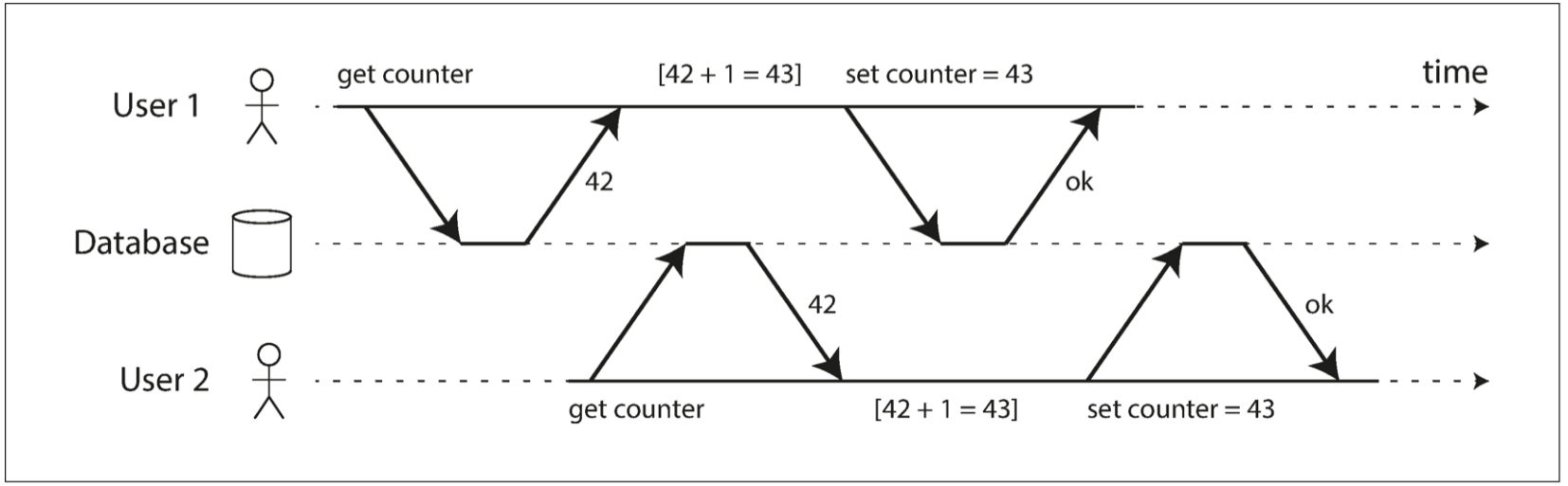


Figure 7-1. A race condition between two clients concurrently incrementing a counter.

> Fonte da imagem: Priyank Verma

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Problema: Lost Update - Exemplo

```c
#include <stdio.h>
#include <pthread.h>

int contador = 0;

void* incrementa_contador(void* thread_id) {
    int tid = (int)(long)thread_id; // Cast para obter o ID da thread
    for (int i = 0; i < 10000; i++) {
        contador++; // Acesso direto à variável global sem proteção
    }
    printf("Thread %d: Contador final = %d\n", tid, contador);
    pthread_exit(0);
}

int main() {
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, incrementa_contador,  (void*)1); // Cria thread1
    pthread_create(&thread2, NULL, incrementa_contador,  (void*)2); // Cria thread2
    pthread_join(thread1, NULL); // Espera a thread1 terminar
    pthread_join(thread2, NULL); // Espera a thread2 terminar

    printf("Valor final do contador: %d\n", contador); // O valor final será imprevisível
    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Seção Crítica

Bloco de código que acessa um recurso compartilhado.

```
entrar na região crítica
    acessar ou modificar recurso compartilhado
sair da região crítica
```

Exemplos de recursos compartilhados:

- Arquivos
- Bancos de dados
- Impressoras
- Variáveis compartilhadas
- Serviços de rede

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exclusão Mútua

Exclusão mútua significa garantir que apenas um processo por vez acesse uma seção crítica.

Objetivos:

- Evitar inconsistência de dados.
- Evitar conflitos no acesso a recursos.
- Garantir previsibilidade na execução.

```
Se Processo A está na seção crítica,
Processo B deve esperar.
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exclusão Mútua: Fluxo

1. Processo quer entrar: **pede permissão**
2. Sistema decide: **concede ou nega**
3. Processo executa: **acessa o recurso**
4. Processo termina: **libera o recurso**

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Propriedades desejadas da exclusão mútua

Todo algoritmo de exclusão mútua deve satisfazer **3 propriedades:**

#	Propriedade	Descrição
1	**Exclusão Mútua**	Apenas 1 processo na seção crítica por vez. Sempre.
2	**Progresso**	Se nenhum está na seção crítica, quem quer entrar deve conseguir.
3	**Espera Limitada**	Nenhum processo espera para sempre. Starvation proibido.

- **Propriedades desejáveis:**
  - **Tolerância a falhas:** se um nó cai, o sistema continua
  - **Desempenho:** poucas mensagens trocadas
- **O que evitar:**
  - **Deadlock:** dois processos esperando um pelo outro infinitamente
  - **Starvation:** um processo nunca consegue entrar na seção crítica

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Concorrência em Sistemas Distribuídos

Processos rodam em **servidores diferentes**, ligados por rede.

- **Sem memória compartilhada**: tudo passa por mensagens
- **Não há relógio global**: difícil ordenar eventos com precisão
- Mensagens podem chegar **fora de ordem ou se perder**
- Máquinas podem falhar de forma independente.

Precisamos de um **protocolo explícito** de coordenação.

> **Exemplo:** Servidor A e Servidor B tentam atualizar o mesmo registro no banco de dados ao mesmo tempo. Qual atualização vale? Quem chegou primeiro?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Como organizar a ordem de eventos em um sistema distribuído?

Relógios de Lamport

Cada processo mantém um relógios lógicos usado em algoritmos de sincronização de relógio baseados na relação *happens-before* definida por Leslie Lamport.

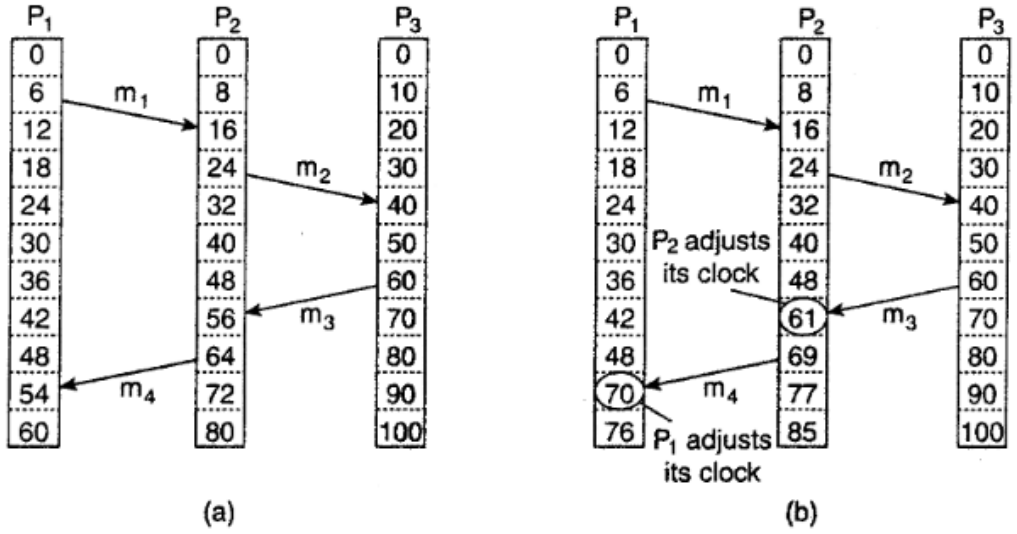


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

Na imagem: (a) Relógios de Lamport são sequências de números ou contadores. (b) O Algoritmo de Lamport corrige os relógios. Fonte: StackOverflow.

Relógios de Lamport

Quando um processo envia uma mensagem, ele inclui o valor do seu contador (relógio) na mensagem.

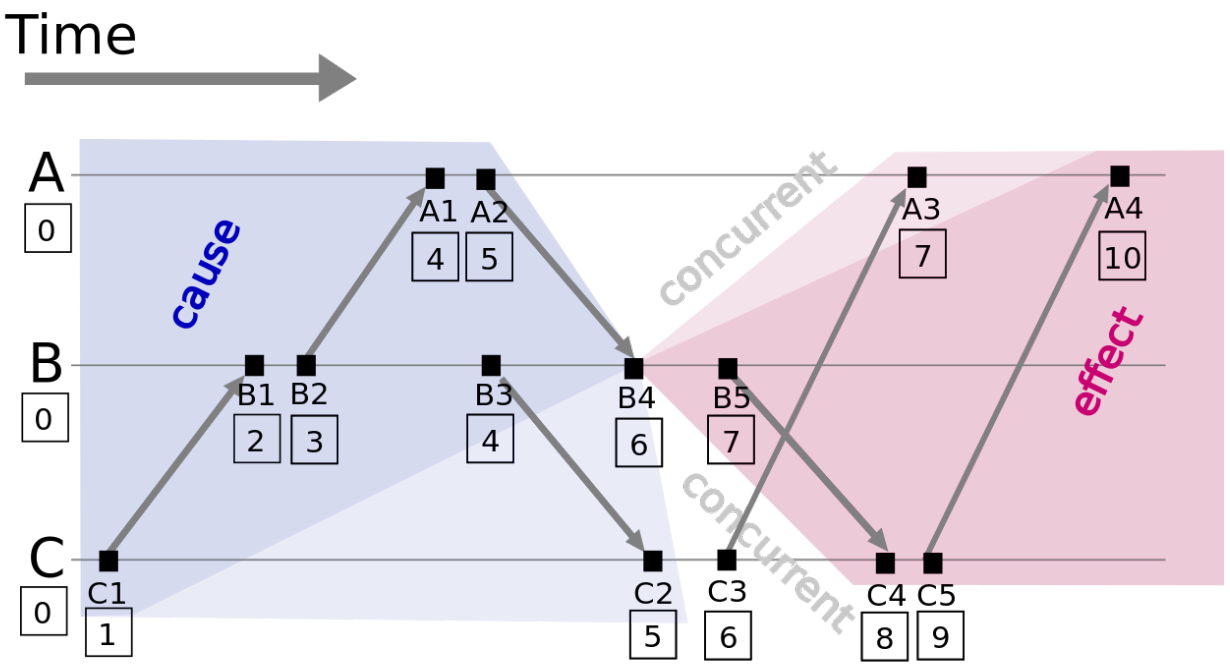


Diagrama de relógios de Lamport. Fonte da imagem: Wikipedia.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Concorrência em Sistemas Distribuídos

Desafio principal: Em um sistema distribuído, como decidir quem pode acessar um recurso compartilhado?

Exemplo:

```
Nó A quer acessar um arquivo distribuído.
Nó B quer acessar o mesmo arquivo.
Nó C também quer acessar o mesmo arquivo.
```

Sem um controle adequado, os três podem tentar modificar o recurso **ao mesmo tempo**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exclusão Mútua em Sistemas Distribuídos

A exclusão mútua distribuída busca coordenar processos em máquinas diferentes.

O problema é garantir que:

```
Apenas um processo distribuído acesse o recurso crítico por vez.
```

Isso deve ocorrer mesmo sem:

- Memória compartilhada.
- Relógio global confiável.
- Coordenador único obrigatório.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

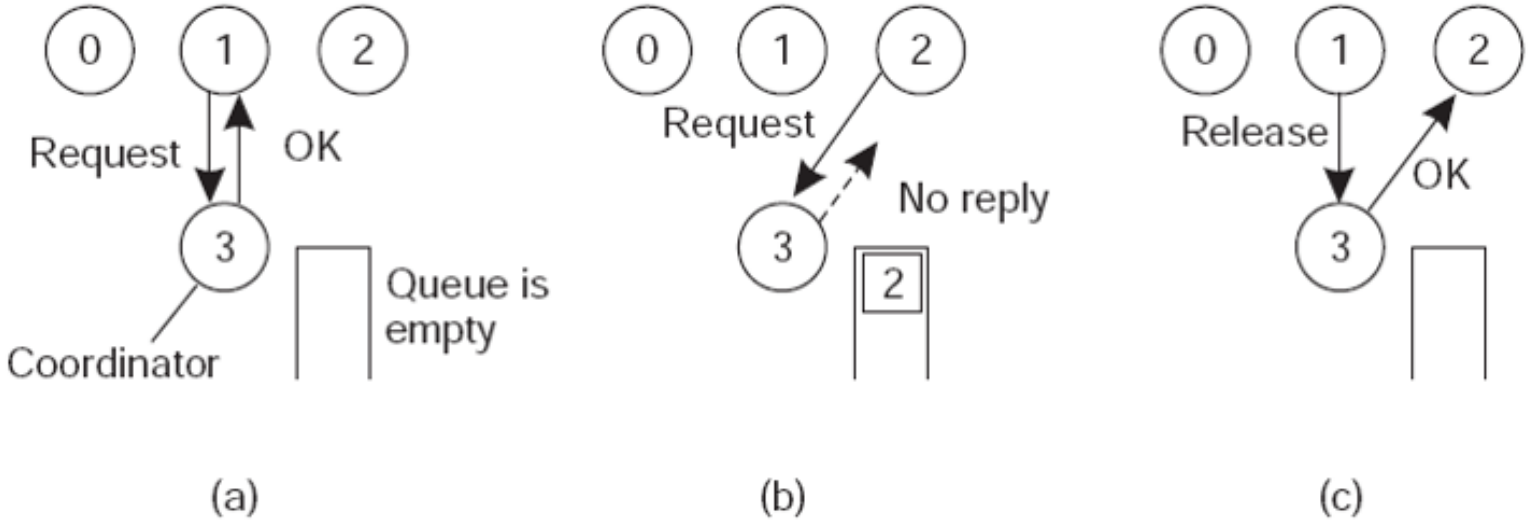
Abordagens para Exclusão Mútua Distribuída

- **Algoritmo Centralizado**: Um nó especial gerencia todos os locks
  - **Vantagem**: Simples de implementar
  - **Desvantagem**: Ponto único de falha
- **Baseado em Permissão**: Processo pede permissão para **todos** os outros
  - **Vantagem**: Tolerante a falhas, totalmente descentralizado
  - **Desvantagem**: custoso, **2(N-1)** mensagens por entrada
- **Baseado em Token**: Um token circula entre os processos
  - Quem tem o token pode entrar na seção crítica
  - **Vantagem**: Baixo overhead em pouca contenda
  - **Desvantagem**: Token pode ser perdido

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo Centralizado

- Um nó especial é eleito **coordenador** (ou servidor de lock)
- Todos os outros nós são **clientes**: nunca se comunicam entre si
- O coordenador mantém uma **fila de espera** e controla quem pode entrar



Fonte da Imagem: quest10.com.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo Descentralizado

Um algoritmo descentralizado não depende de um único coordenador.

Ideia geral:

1. Um processo que deseja entrar na região crítica envia pedidos aos demais.
2. Os outros processos respondem autorizando ou negando temporariamente.
3. O processo entra na região crítica quando recebe permissões suficientes.
4. Ao sair, libera o recurso e avisa os demais.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo Descentralizado: pedidos e permissões

```
Processo P1 quer entrar na região crítica.

P1 -> P2: pedido de acesso
P1 -> P3: pedido de acesso
P1 -> P4: pedido de acesso

P2 -> P1: permitido
P3 -> P1: permitido
P4 -> P1: permitido

P1 entra na região crítica.
```

A entrada só ocorre após receber as autorizações necessárias.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo de Ricart-Agrawala (1981)

- Cada processo pede **permissão a todos os outros** antes de entrar
- Entra somente se receber **OK de todos**
- Usa **timestamps de Lamport** para desempatar pedidos simultâneos
- Processo com **menor timestamp** tem prioridade

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo de Ricart-Agrawala (1981)

Passo a passo:

1. P envia **REQUEST(ts, P)** para todos os nós
2. Cada nó responde **OK** — ou guarda na fila se tiver prioridade
3. P coleta OKs de **todos** → entra na seção crítica
4. Ao sair, P envia **OK** para os processos na fila

> **Regra de desempate:** menor timestamp ganha. Em caso de empate de timestamp, menor ID de processo vence.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Ricart-Agrawala: Pseudocódigo

```
Ao querer entrar na seção crítica:
    ts = meu_timestamp + 1
    enviar REQUEST(ts, eu) para TODOS
    aguardar OK de TODOS os outros nós

Ao receber REQUEST(ts_deles, deles):
    SE estou na seção crítica:
        → adicionar à fila (não responder agora)
    SENÃO SE quero entrar E tenho prioridade:
        → (meu_ts < ts_deles)
           OU (empate: meu_id < id_deles)
        → adicionar à fila (não responder agora)
    SENÃO:
        → responder OK imediatamente

Ao sair da seção crítica:
    enviar OK para todos os processos na fila
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo de Token em Anel

Visão Geral

- Os processos formam um **anel lógico**
- Existe exatamente **1 token** no sistema
- O token passa de processo em processo **na ordem do anel**

Funcionamento:

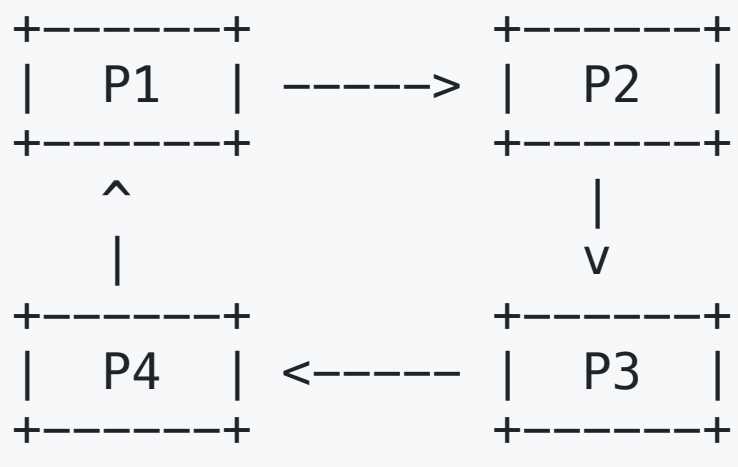
1. Os processos são organizados em um círculo (anel).
2. Apenas quem possui o *Token* pode entrar na Seção Crítica.
3. Ao terminar, o processo deve passar o *Token* para o próximo da fila.
4. Se não precisa acessar o recurso, apenas repassa o *Token*.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo de Token em Anel

O token funciona como uma “chave” que autoriza o uso do recurso compartilhado.

```
funcao Processo_i(Recebeu_Token):
    // 1. Entra na Seção Crítica
    Executar_Tarefa();
    // 2. Passa o Token adiante
    Enviar_Token(Processo_(i+1));
```



O token circula em uma ordem definida.Imagine uma linha de montagem onde o "sinal verde" (token) só é passado

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Algoritmo de Token: Vantagens e Desvantagens

Vantagens:

- Garante exclusão mútua de forma simples.
- Evita conflito direto entre múltiplos pedidos.
- Pode reduzir o número de mensagens em alguns cenários.

Ideia importante: não há disputa simultânea se só existe um token válido.

Desvantagens:

- O token pode ser perdido por falha de um processo.
- Um processo pode demorar a repassar o token.
- É necessário detectar e regenerar o token com cuidado.
- Se houver dois tokens por erro, a exclusão mútua pode ser violada.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Critérios de avaliação dos algoritmos

Ao comparar algoritmos de exclusão mútua distribuída, analisamos:

- Número de mensagens trocadas.
- Tempo de espera para entrar na região crítica.
- Tolerância a falhas.
- Risco de deadlock.
- Justiça entre os processos.
- Escalabilidade com muitos nós.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 1 (discussão em sala)

Considere três processos distribuídos: P1, P2 e P3.

Todos desejam acessar o mesmo arquivo compartilhado.

Perguntas:

1. O que pode dar errado se não houver exclusão mútua?
2. Qual parte da operação representa a região crítica?
3. Como um algoritmo baseado em token resolveria esse problema?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 2 (discussão em sala)

Analise a situação:

```
P1 está com o token.
P2 precisa acessar a região crítica.
P3 também precisa acessar a região crítica.
P1 falha antes de repassar o token.
```

Perguntas:

1. Qual é o problema gerado?
2. Por que não é seguro simplesmente criar um novo token imediatamente?
3. Que tipo de mecanismo adicional seria necessário?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Conclusão

- **Race condition:** dois processos acessam o mesmo dado → resultado incorreto
- **Seção crítica:** trecho de código que acessa recurso compartilhado
- **Exclusão mútua:** garantir que apenas 1 processo acessa por vez
- **3 requisitos:** exclusão mútua, progresso, espera limitada
- **Algoritmo de Ricart-Agrawala:** pede permissão a todos, timestamps para desempatar
- **Algoritmo de Token em Anel:** token circula, quem tem o token pode entrar

> **Pergunta**: O que acontece se o nó que tem o token falhar *durante* a seção crítica? Como o sistema detecta essa falha e regenera o token com segurança?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questões para Estudo

Questão 1

Em um sistema distribuído real, o que é mais difícil de lidar?

- Muitos processos concorrendo?
- Falhas de comunicação?
- Perda do token?
- Falta de relógio global?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão 2

Explique a principal diferença arquitetural e de resiliência entre usar um **Lock Manager Centralizado** versus um sistema baseado em **Passagem de Token**. Em qual cenário você preferiria cada abordagem?

> Descreva o passo a passo (em 4 etapas) que ocorre desde o momento em que o Processo A recebe o Token até o momento em que ele passa o Token para o Processo B, garantindo que nenhum outro processo possa interferir nesse intervalo.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão 3

Considere um sistema distribuído usando Passagem de Token. Se o **Processo C** falhar *depois* de ter executado a Seção Crítica, mas *antes* de passar o Token para o Processo D, qual é o estado do sistema? Como esse problema pode ser mitigado em teoria (sem detalhar algoritmos complexos)?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão 4

Em um ambiente onde os processos são muito numerosos e as operações na Seção Crítica são extremamente rápidas (milissegundos), qual dos mecanismos de exclusão mútua (Centralizado ou Token) tende a gerar mais *overhead* (custo extra de comunicação)? Justifique sua resposta.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br