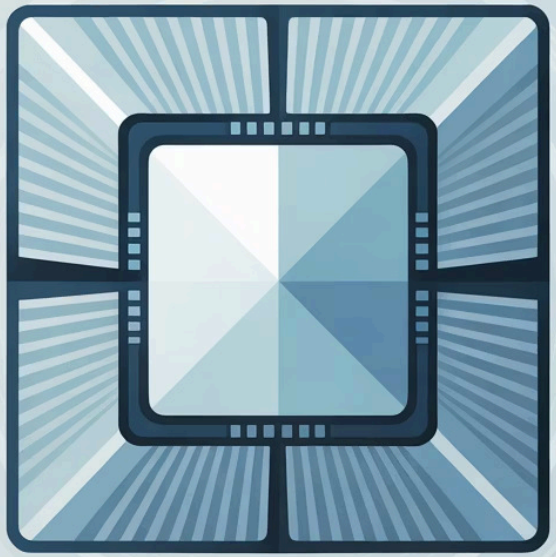


PROCESSOR

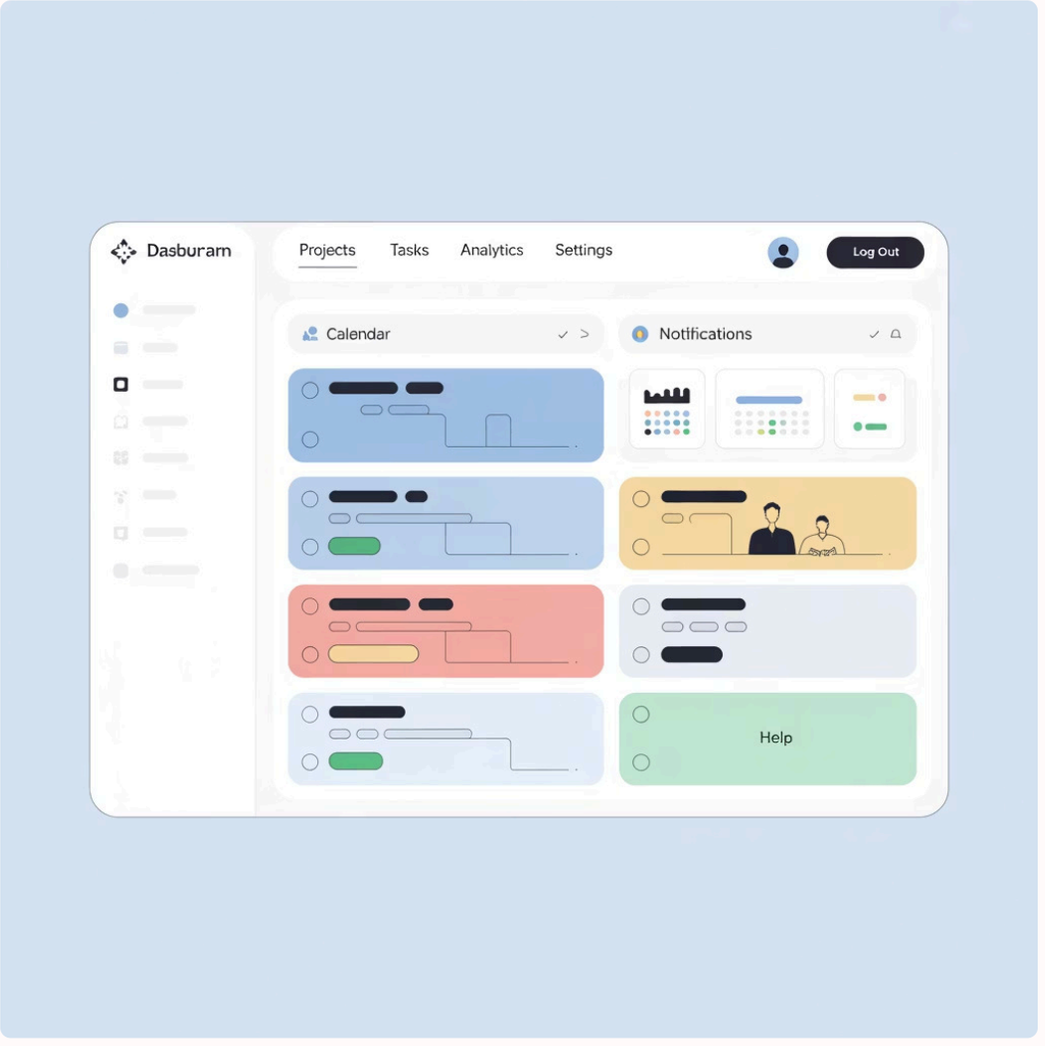
Escalonamento de Tarefas em Sistemas Operacionais

O escalonamento de tarefas é um dos componentes mais críticos dos sistemas operacionais modernos. Através do escalonador de tarefas, o sistema decide a ordem de execução das tarefas prontas, impactando diretamente na eficiência, responsividade e justiça do sistema.

Por que estudar escalonamento?

Impacto no Desempenho

O algoritmo de escalonamento define o comportamento fundamental do sistema operacional, determinando como ele gerencia recursos limitados entre múltiplas tarefas concorrentes.



Diferentes algoritmos são otimizados para cenários específicos: aplicações interativas precisam de baixa latência, processamento de dados necessita de alta eficiência, e sistemas críticos demandam previsibilidade temporal.

Tipos de Tarefas por Comportamento Temporal

Tarefas de Tempo Real

Exigem previsibilidade rigorosa nos tempos de resposta. Utilizadas em sistemas críticos como controle industrial, automação e processamento multimídia.

- Requisitos temporais rígidos
- Previsibilidade obrigatória
- Falhas podem ser catastróficas

Tarefas Interativas

Respondem a eventos externos rapidamente, mas sem os requisitos rígidos do tempo real. Incluem editores, navegadores, jogos e servidores web.

- Resposta rápida ao usuário
- Interface fluida
- Tolerância a pequenos atrasos

Tarefas em Lote (Batch)

Executam sem requisitos temporais específicos, normalmente sem intervenção do usuário. Exemplos: backups, antivírus, cálculos científicos.

- Processamento em segundo plano
- Sem urgência temporal
- Focadas em eficiência

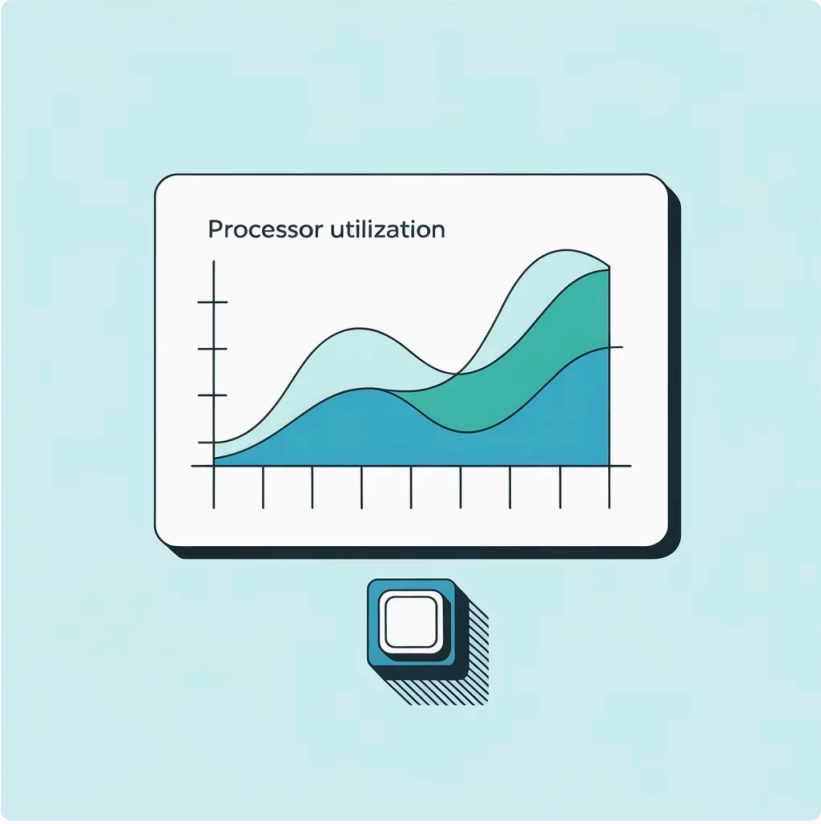
Classificação por Uso de Recursos

CPU-Bound vs I/O-Bound

As tarefas também se classificam pelo padrão de uso de recursos, o que influencia diretamente nas decisões de escalonamento:

- **CPU-Bound:** Usam intensivamente o processador (conversão de vídeo, cálculos matemáticos)
- **I/O-Bound:** Dependem de operações de entrada/saída (editores, compiladores, servidores)

Uma tarefa pode alternar entre comportamentos durante sua execução, como um conversor que alterna entre processamento e leitura/escrita de arquivos.



Padrões de uso típicos de diferentes tipos de tarefas

Objetivos do Escalonamento

Os escalonadores devem equilibrar múltiplos objetivos conflitantes, priorizando diferentes aspectos conforme o contexto de uso do sistema.

Métricas Fundamentais de Avaliação

Tempo de Execução (Turnaround Time)

Tempo total desde a criação até o encerramento da tarefa, incluindo processamento e espera. Métrica crucial para sistemas em lote onde a eficiência global é prioritária.

Tempo de Espera (Waiting Time)

Tempo perdido na fila de tarefas prontas, aguardando o processador. Não inclui esperas por I/O, que são inerentes à aplicação e aos dispositivos utilizados.

Tempo de Resposta (Response Time)

Intervalo entre um evento e sua resposta imediata. Fundamental para sistemas interativos - exemplo: tempo entre pressionar uma tecla e ver o caractere na tela.

Justiça (Fairness)

Distribuição equitativa do processador entre tarefas similares. Tarefas com comportamento e prioridade semelhantes devem receber tratamento proporcional.

Eficiência do Sistema

Medindo a Eficiência

A eficiência (E) indica o grau de utilização do processador na execução de tarefas úteis. Fatores que impactam a eficiência:

- Velocidade da troca de contexto
- Quantidade de tarefas I/O-bound
- Overhead do escalonador
- Fragmentação do tempo de CPU

Tarefas I/O-bound frequentemente liberam o processador antes do fim do quantum, gerando mais trocas de contexto e reduzindo a eficiência global.

Escalonamento: Preemptivo vs Cooperativo

Sistemas Preemptivos

O escalonador pode interromper uma tarefa a qualquer momento por três motivos principais:

- Término do quantum de tempo
- Execução de chamada de sistema
- Interrupção que desperta tarefa mais prioritária

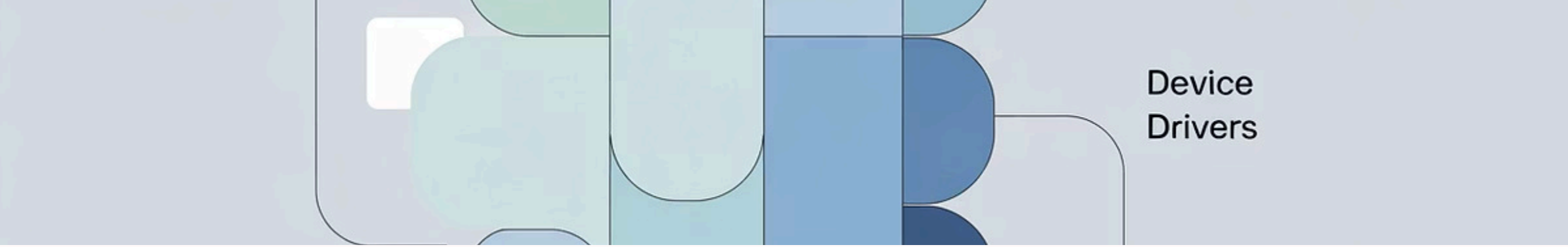
A cada interrupção, o escalonador reavalia todas as tarefas prontas e decide se mantém ou substitui a tarefa atual.

Sistemas Cooperativos

A tarefa executa até decidir liberar o processador voluntariamente:

- Término natural da execução
- Solicitação de operação I/O
- Liberação explícita (sched_yield)

Exigem cooperação das tarefas para funcionamento adequado. Sistemas legados como Windows 3.x operavam assim.



Núcleos Preemptivos Modernos

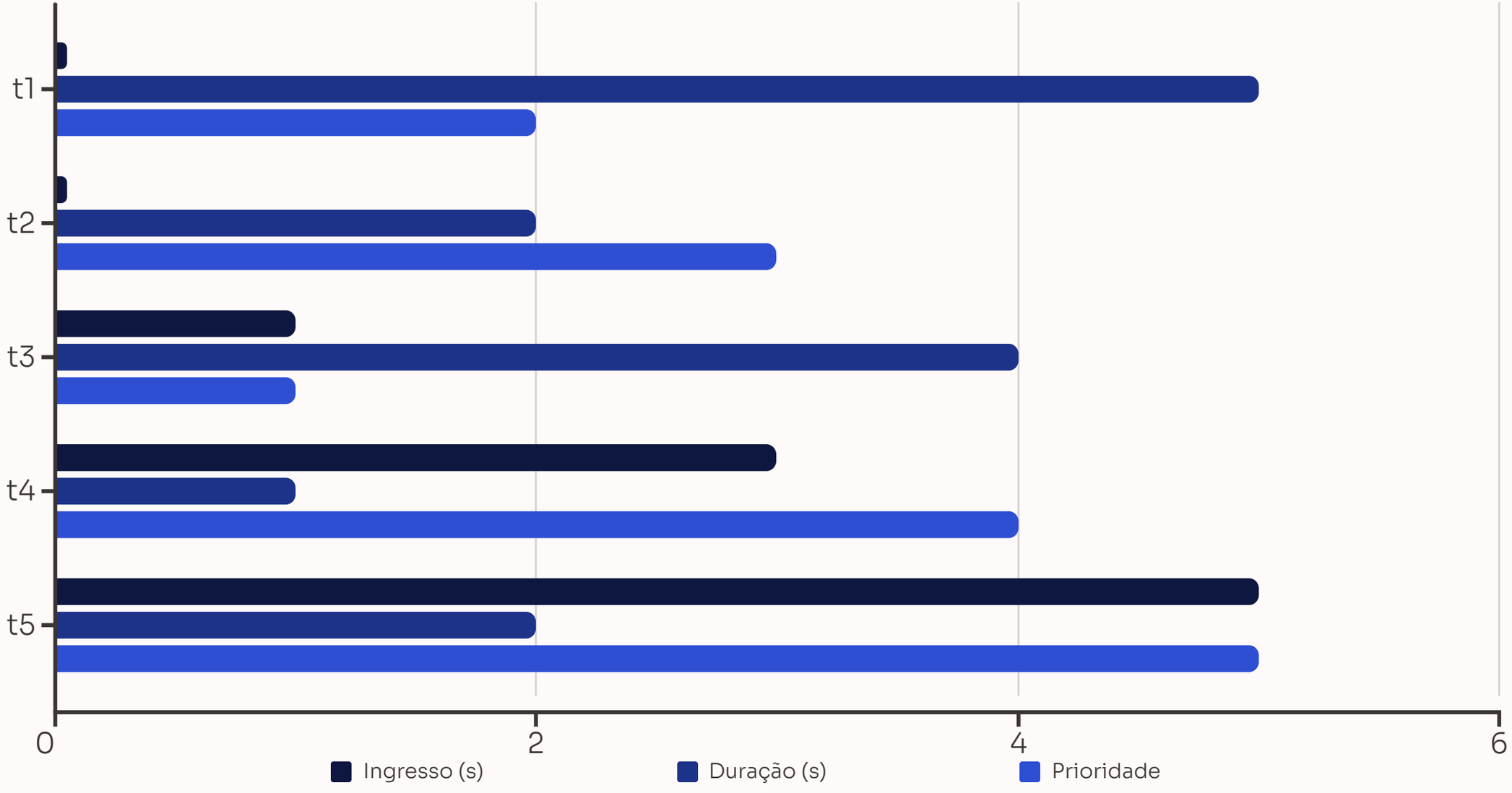
Sistemas simples interrompem tarefas apenas no modo usuário, mantendo threads de núcleo ininterruptas. Sistemas sofisticados implementam preempção também no modo núcleo, permitindo que tarefas de alta prioridade cheguem mais rapidamente ao processador.

Núcleos preemptivos como Solaris, Linux 2.6+ e Windows NT são essenciais para sistemas de tempo real, onde a latência de resposta deve ser minimizada independentemente do estado atual do sistema.

Algoritmos Clássicos

Estudaremos os algoritmos fundamentais que servem de base conceitual para escalonadores modernos, usando um conjunto hipotético de 5 tarefas para demonstrar cada comportamento.

Conjunto de Tarefas para Análise



Para simplificar a análise, consideramos tarefas CPU-bound que não realizam operações I/O. A prioridade utiliza escala positiva (valores maiores = maior prioridade).

First-Come, First-Served (FCFS)

Algoritmo Mais Simples

Atende tarefas sequencialmente conforme chegam ao sistema. Características principais:

- Implementação trivial
- Não requer informações adicionais
- Comportamento previsível
- Pode causar convoys de tarefas lentas

No exemplo com nossas 5 tarefas: Tt médio = 8,0s, Tw médio = 5,2s, com apenas 4 trocas de contexto. Algoritmo eficiente para processamento em lote, mas inadequado para sistemas interativos.

Round-Robin (RR)

FCFS com preempção por tempo cria o algoritmo Round-Robin. Com quantum de 2s em nosso exemplo, observamos comportamento mais complexo: $t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_1 \rightarrow t_4 \rightarrow t_5 \rightarrow \ldots$

A ordem não é sequencial devido à dinâmica da fila de prontas - tarefas retornam à fila conforme seus quanta expiram, intercalando com novas chegadas.

Análise do Round-Robin

8.4s

Tempo Médio de Execução

Ligeiramente superior ao FCFS devido ao overhead de trocas de contexto

5.6s

Tempo Médio de Espera

Maior que FCFS, mas compensado pela melhor distribuição temporal

7

Trocas de Contexto

Quase dobro do FCFS, impactando eficiência mas melhorando responsividade

RR sacrifica eficiência bruta por melhor tempo de resposta, sendo ideal para sistemas interativos onde múltiplas tarefas competem pela atenção do usuário.

Shortest Job First (SJF)

Otimização Matemática

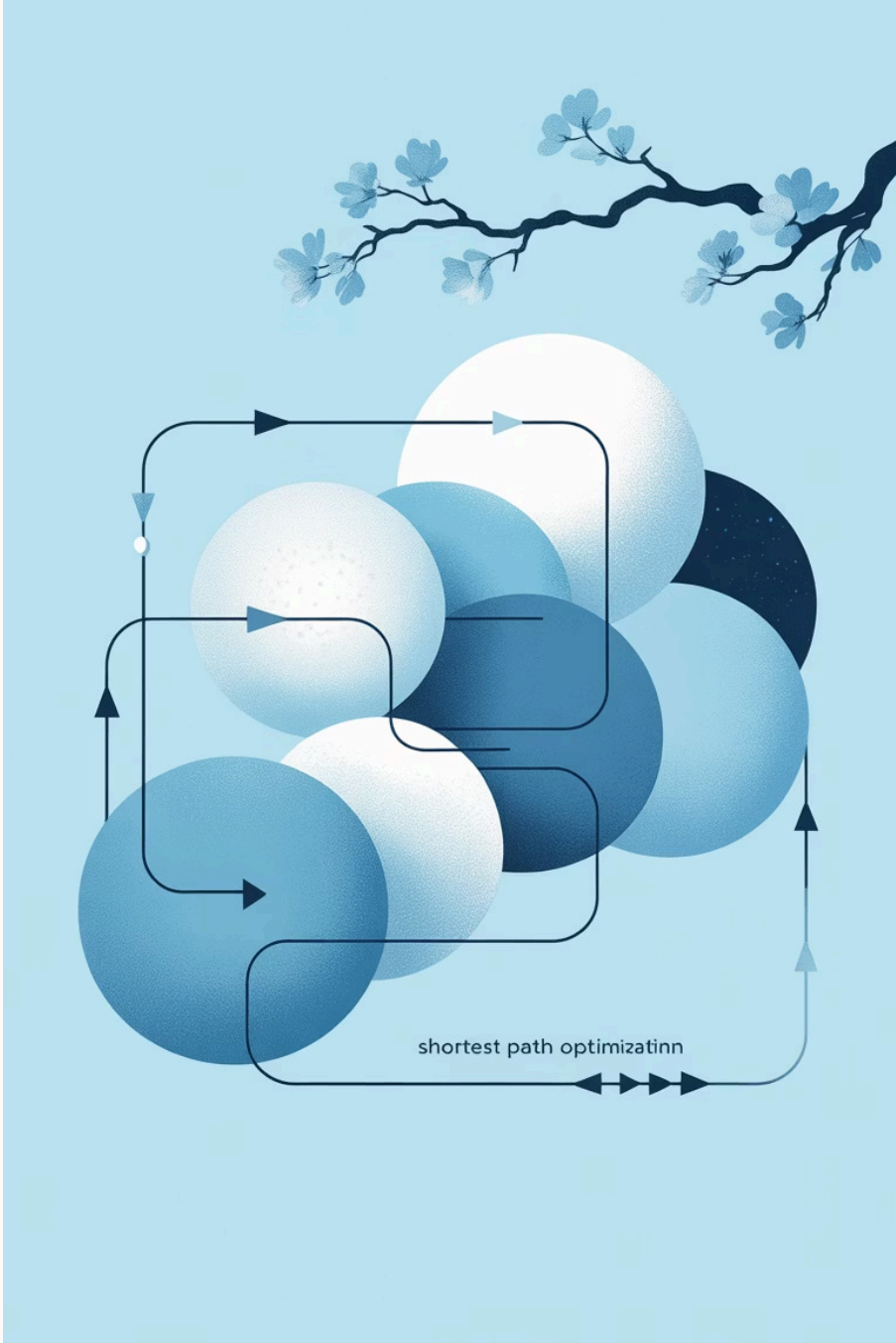
Executa sempre a tarefa mais curta disponível, proporcionando os menores tempos médios teoricamente possíveis:

- Tt = 5,8s (melhor que FCFS e RR)
- Tw = 3,0s (significativamente menor)
- Apenas 4 trocas de contexto

Limitações Práticas

Principais desafios na implementação:

- Como estimar duração futura?
- Risco de inanição para tarefas longas
- Aplicável principalmente a sistemas batch



Estimativa Dinâmica de Duração

Para superar as limitações do SJF, podemos usar histórico recente da tarefa. Em sistemas preemptivos com quantum de 10ms, uma tarefa I/O-bound que utilizou 3ms, 4ms e 4,5ms nos últimos quanta pode ter sua próxima utilização estimada.

01

Coleta de Histórico

Registrar utilização dos últimos 3-5 quanta para capturar padrão recente de comportamento

02

Cálculo da Estimativa

Aplicar média simples ou extrapolação ponderada, priorizando dados mais recentes

03

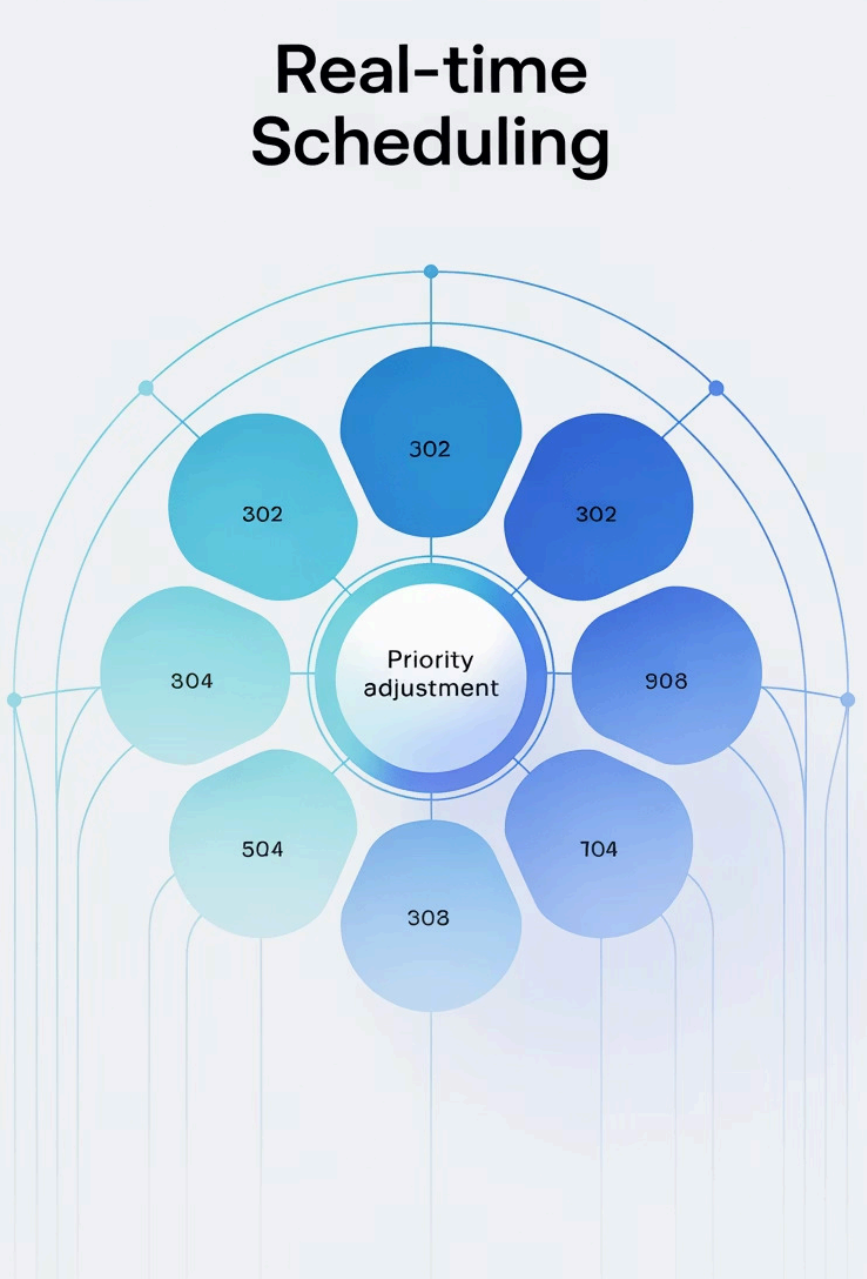
Aplicação do SJF

Usar estimativa como critério de seleção, beneficiando tarefas I/O-bound

Shortest Remaining Time First (SRTF)

Versão preemptiva do SJF que compara tempo restante da tarefa atual com duração de novas tarefas. A cada ingresso no sistema, reavalia se deve manter ou trocar a tarefa executando.

Resultados excepcionais: Tt = 5,4s e Tw = 2,6s (menores valores possíveis). Contudo, intensifica o problema de inanição e aumenta complexidade do escalonador.



Escalonamento por Prioridades

Modelo genérico que permite implementar diversos critérios de importância, desde comportamento da tarefa até classe do usuário.

Prioridades Fixas: Cooperativo vs Preemptivo

PRIOc - Cooperativo

Executa tarefas por ordem de prioridade até conclusão natural:

- Tt = 6,6s
- Tw = 3,8s
- 4 trocas de contexto

Tarefas de menor prioridade podem esperar muito tempo na fila.

PRIOp - Preemptivo

Interrompe tarefa atual quando chega outra mais prioritária:

- Tt = 5,6s
- Tw = 2,8s
- 6 trocas de contexto

Melhores métricas, mas risco severo de inanição.

O Problema da Inanição

Em sistemas com prioridades fixas, tarefas de menor prioridade podem "morrer de fome" se houver fluxo constante de tarefas mais prioritárias. Este problema é especialmente grave em servidores com alta carga.

A solução está no **envelhecimento** (aging): aumentar gradualmente a prioridade de tarefas que aguardam muito tempo, garantindo que eventualmente todas recebam processador.

Prioridades Dinâmicas com Envelhecimento

Algoritmo que associa duas prioridades a cada tarefa: estática (pe, definida externamente) e dinâmica (pd, que evolui internamente).

1

Nova Tarefa

pd = pe (prioridade dinâmica inicia igual à estática)

2

Seleção

Escolhe tarefa com maior pd

3

Envelhecimento

Demais tarefas: pd += α

4

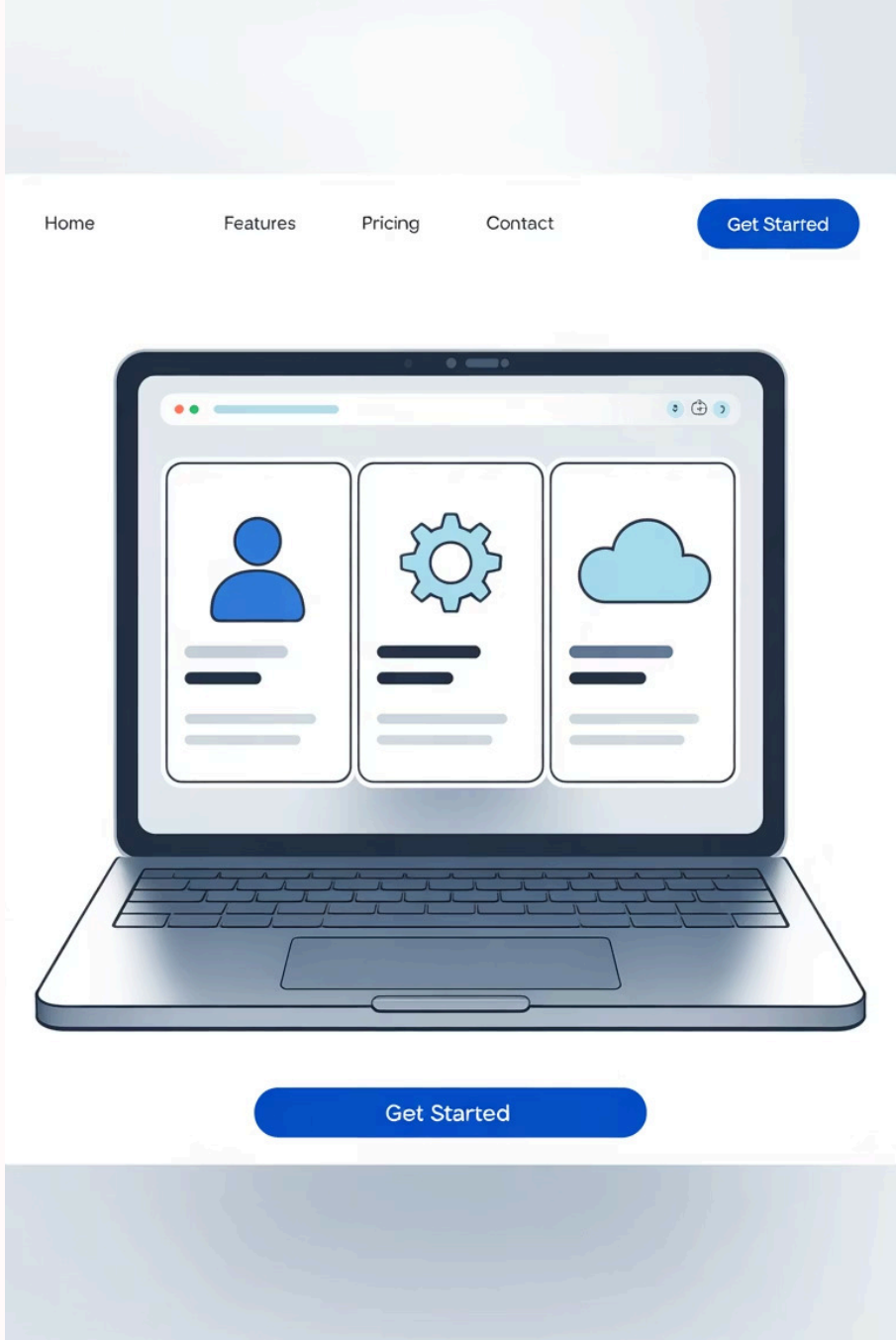
Rejuvenescimento

Tarefa executada: pd = pe

Proporcionalidade no Tempo Compartilhado

Prioridades dinâmicas resolvem problema crucial: em sistemas Round-Robin com prioridades fixas, tarefas executam sequencialmente sem distribuição proporcional. Tarefas t_1(prioridade 1), t_2(prioridade 3) e t_3(prioridade 6) deveriam receber processador proporcionalmente.

Com envelhecimento, t_3 recebe mais tempo que t_2, que recebe mais que t_1, mantendo todas ativas e respeitando a proporcionalidade desejada pelo usuário.



Definindo Prioridades: Fatores Externos e Internos

Fatores Externos

Informações que o escalonador não pode descobrir sozinho:

- Classe do usuário (admin, gerente, estagiário)
- Nível de serviço pago (básico, premium)
- Importância crítica da tarefa
- Políticas organizacionais

Fatores Internos

Dados coletados e estimados pelo próprio sistema:

- Idade da tarefa no sistema
- Duração estimada de execução
- Grau de interatividade observado
- Uso de memória e outros recursos

Escalas de Prioridade: Windows vs Linux

Windows (2000+)

Classes hierárquicas para processos e threads:

- 24: Tempo real
- 13: Alta
- 10: Acima do normal
- 8: Normal
- 6: Abaixo do normal
- 4: Baixa/Ociosa

Janela ativa recebe incremento +1 ou +2

Linux (2.4+)

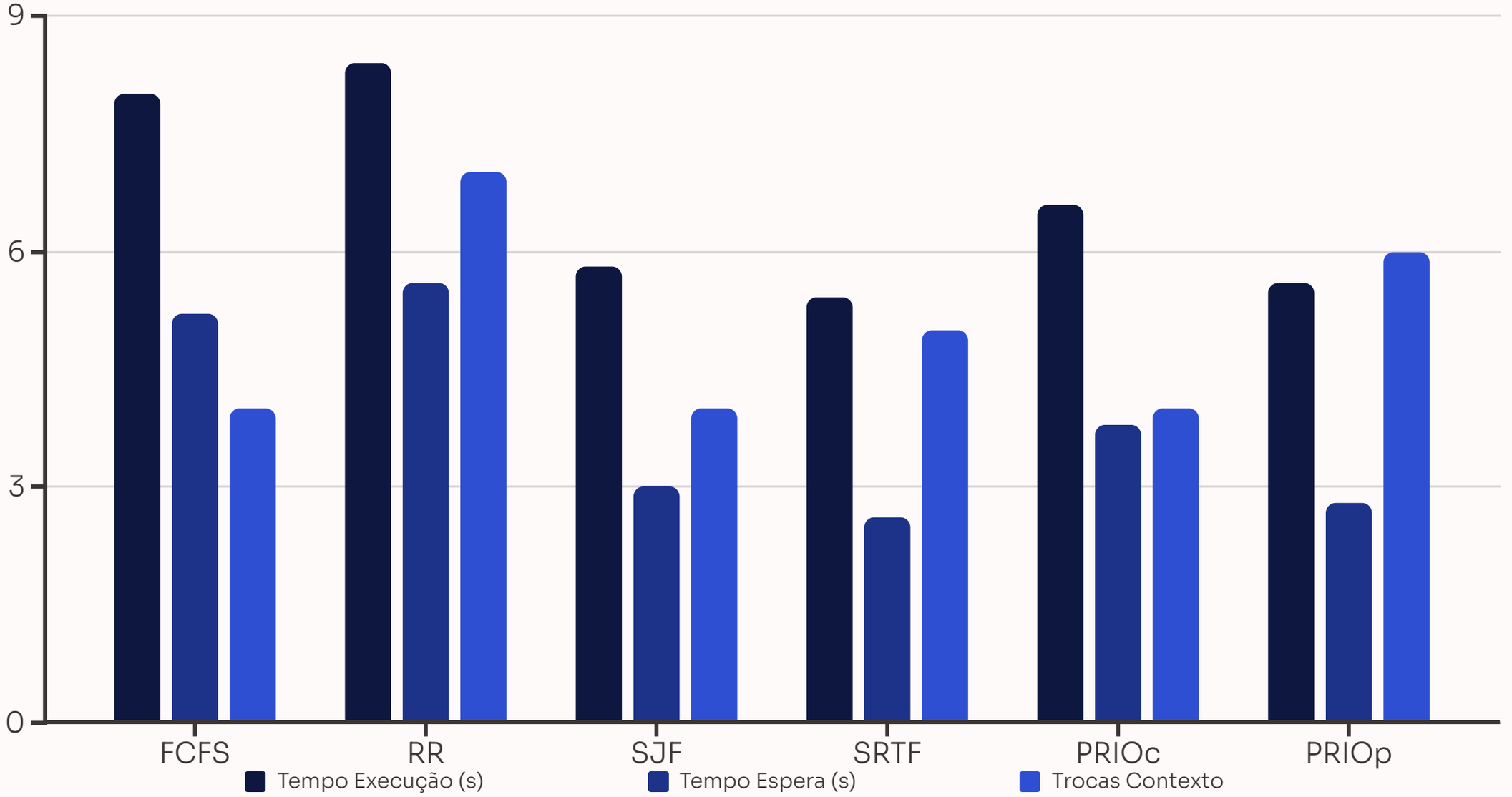
Duas escalas distintas:

Tempo Real: 1-99 positiva (maior valor = maior prioridade)

Nice Level: -20 a +19 negativa (maior valor = menor prioridade)

Comandos: nice e renice para ajuste por usuários

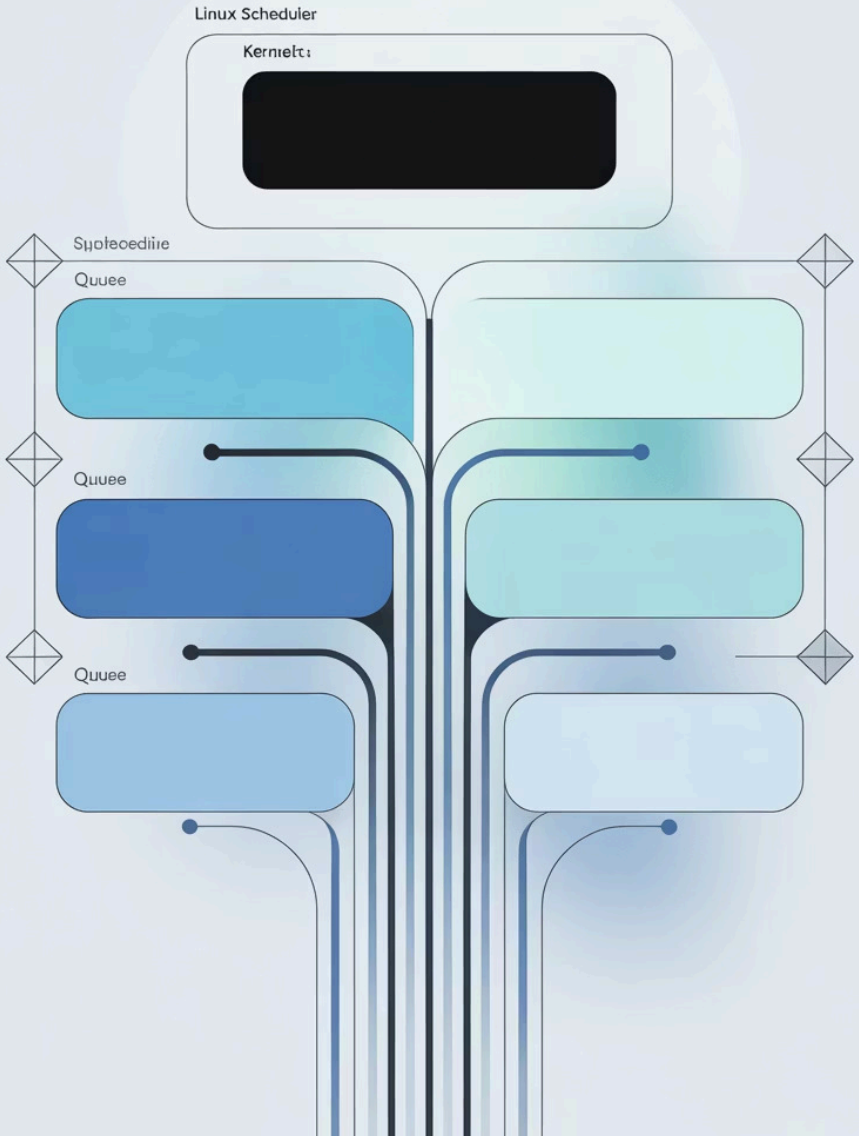
Comparação dos Algoritmos Estudados



SRTF oferece melhores métricas, RR melhor responsividade. Preempção aumenta trocas de contexto mas melhora tempos médios.

Escalonadores Modernos

Sistemas reais combinam múltiplas políticas em arquiteturas sofisticadas, adaptando-se dinamicamente aos diferentes tipos de tarefas.



Arquitetura Linux: Multiple Feedback Queues

O Linux divide tarefas em classes de escalonamento, cada uma com sua fila e algoritmo específico. Esta abordagem permite otimização especializada para diferentes necessidades computacionais.

Hierarquia de precedência: DEADLINE > FIFO > RR > NORMAL/BATCH/IDLE. Tarefas interativas só executam quando não há tarefas de tempo real ativas.

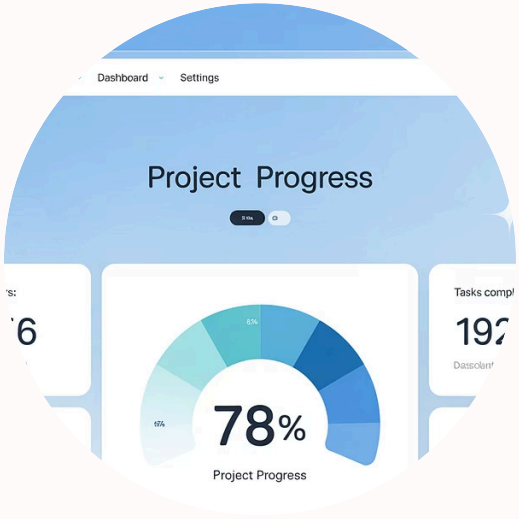
Classes de Escalonamento no Linux

SCHED_DEADLINE

Tarefas de tempo real com prazos específicos. Utiliza algoritmo Earliest Deadline First (EDF) para garantir cumprimento de deadlines.

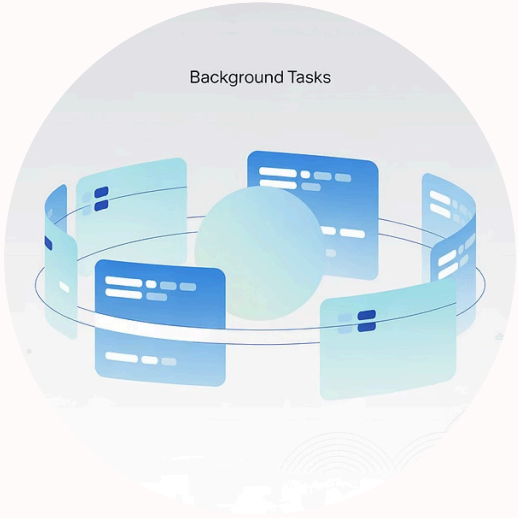
SCHED_FIFO/RR

Prioridades fixas sem/com quantum respectivamente. FIFO executa até bloquear, RR adiciona preempção temporal com quantum proporcional.



SCHED_NORMAL

Classe padrão para tarefas interativas. Implementa Completely Fair Scheduler (CFS) baseado em tempos de processamento realizados.



SCHED_BATCH

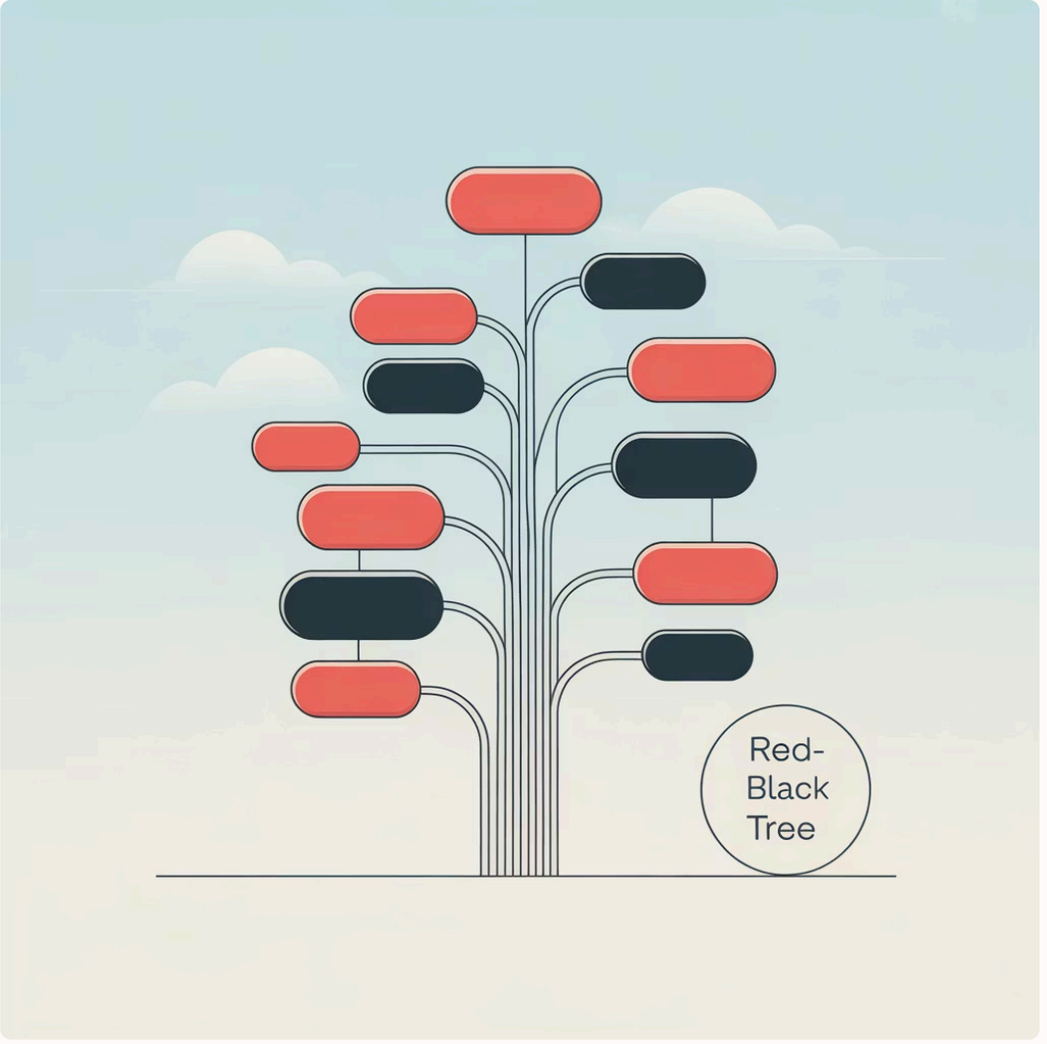
Similar ao NORMAL mas otimizada para tarefas CPU-bound, sendo ativada menos frequentemente que tarefas interativas.

Completely Fair Scheduler (CFS)

Escalonador Justo

O CFS mantém tarefas em árvore rubro-negra ordenada por tempo de processamento realizado:

- Sempre escolhe tarefa com menor tempo acumulado
- Operações O(log n) garantem escalabilidade
- Proporcionalidade baseada em pesos (nice values)
- Adaptação automática a mudanças de carga



A estrutura de árvore rubro-negra permite busca, inserção e remoção eficientes mesmo com milhares de tarefas.

Conclusões e Próximos Passos

Fundamentos Sólidos

Compreendemos os algoritmos clássicos que fundamentam escalonadores modernos, suas vantagens, limitações e trade-offs característicos.

Aplicação Prática

Sistemas reais combinam múltiplas estratégias, adaptando-se dinamicamente aos padrões de carga e requisitos específicos das aplicações.

Evolução Contínua

O desenvolvimento continua com algoritmos para multicore, tempo real e especializações como sistemas embarcados e high-performance computing.