

EP3 - MAC0422

Simulador de Gerência de Memória

Leonardo Heidi Almeida Murakami - NUSP: 11260186

Variáveis Auxiliares Utilizadas

First Fit

`int current_block_start`

Propósito: Armazena o índice de início do bloco de memória livre que está sendo analisado no momento.

Funcionamento: É inicializada com -1. Quando a primeira unidade livre de um bloco é encontrada, `current_block_start` recebe o índice i daquela unidade. É resetada para -1 ao encontrar uma unidade ocupada, indicando o fim de um bloco livre.

`int current_block_size`

Propósito: Contabiliza o tamanho (em unidades de alocação) do bloco livre atual.

Funcionamento: É incrementada para cada unidade livre contígua encontrada. Quando o seu valor atinge o tamanho requisitado (`size_needed`), a busca termina. É zerada ao encontrar uma unidade ocupada.

Next Fit

`int next_fit_last_search_pos`

Propósito: Esta é a variável de estado mais importante para o Next Fit. Ela retém, entre as diferentes chamadas de alocação, a posição na memória onde a próxima busca deve começar.

Funcionamento: É declarada como static na função main para que seu valor persista. Após uma alocação bem-sucedida no índice `allocation_start_index`, ela é atualizada para $(\text{allocation_start_index} + \text{size_requested}) \% \text{TOTAL_MEMORY_UNITS}$, garantindo que a próxima busca inicie imediatamente após o bloco recém-alocado.

`int current_block_start`

Mesmo funcionamento da variável no First Fit

`int current_block_size`

Mesmo funcionamento da variável no First Fit

Best Fit

`int best_fit_start`

Propósito: Armazena o índice inicial do menor bloco de memória encontrado até o momento que satisfaz a requisição.

Funcionamento: É atualizada sempre que um bloco livre com tamanho menor que `best_fit_size` (mas ainda suficiente) é encontrado.

`int best_fit_size`

Propósito: Guarda o tamanho do menor bloco encontrado até o momento.

Funcionamento: É inicializada com o maior valor possível (`INT_MAX`). Quando um bloco livre com tamanho `current_block_size >= size_needed` é encontrado, seu tamanho é comparado com `best_fit_size`. Se for menor, `best_fit_size` é atualizada com `current_block_size`.

`int current_block_start`

Mesmo funcionamento da variável no First Fit

`int current_block_size`

Mesmo funcionamento da variável no First Fit

Worst Fit

`int worst_fit_start`

Propósito: Armazena o índice inicial do maior bloco de memória encontrado que satisfaz a requisição.

Funcionamento: É atualizada sempre que um bloco livre com tamanho maior que `worst_fit_size` é encontrado.

`int worst_fit_size`

Propósito: Guarda o tamanho do maior bloco encontrado.

Funcionamento: É inicializada com -1. Quando um bloco livre com tamanho `current_block_size >= size_needed` é encontrado, seu tamanho é comparado com `worst_fit_size`. Se for maior, `worst_fit_size` é atualizada com o novo valor.

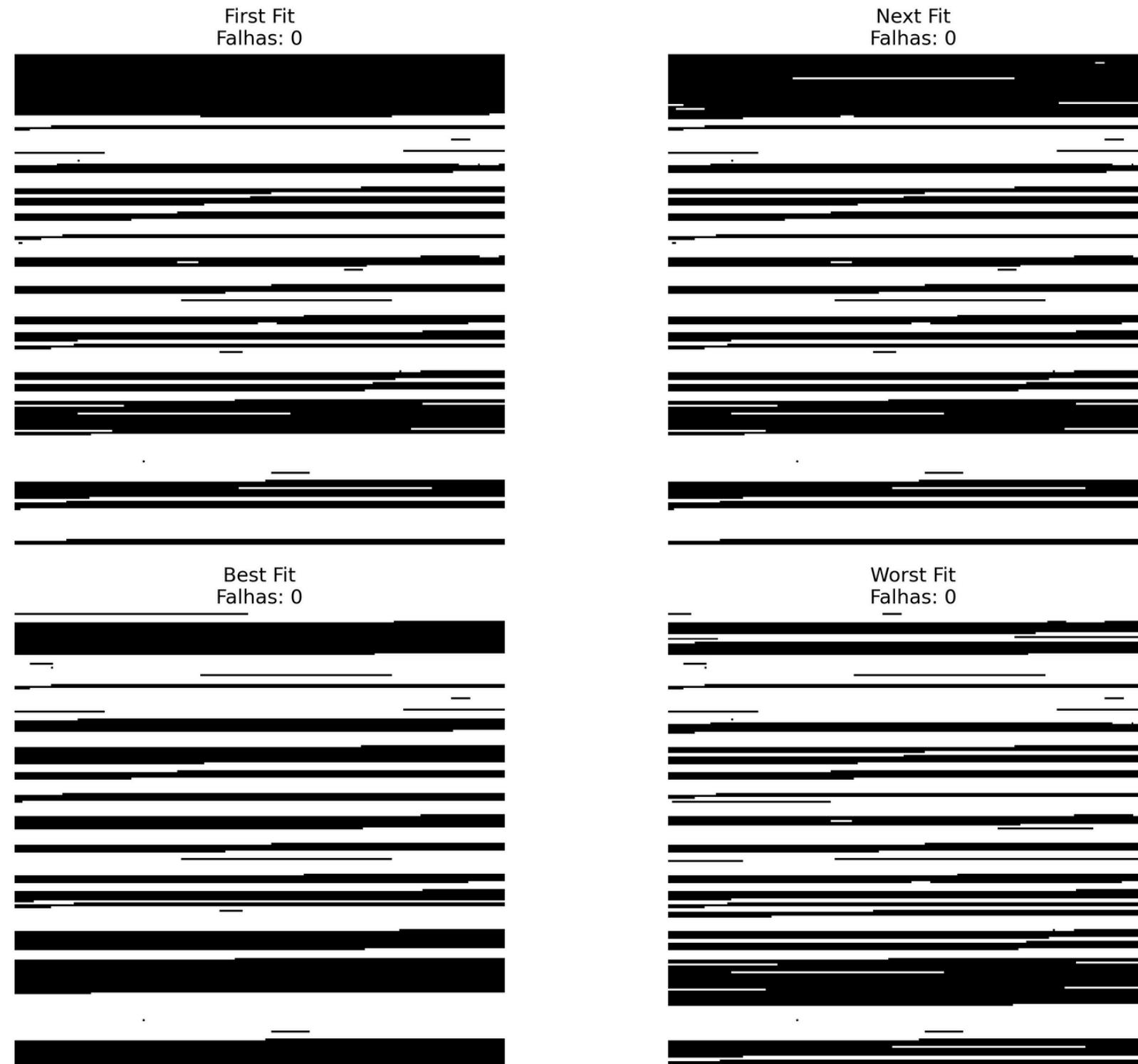
`int current_block_start`

Mesmo funcionamento da variável no First Fit

`int current_block_size`

Mesmo funcionamento da variável no First Fit

trace-firstfit



Por que o trace-firstfit favorece este algoritmo?

O arquivo trace-firstfit foi desenhado com um padrão específico para ressaltar as qualidades do First Fit: muitas alocações pequenas seguidas por uma alocação grande, repetidamente.

Exemplo do Padrão no Trace:

- Primeiro, 10 pedidos de 10 unidades cada.
- Logo depois, um grande pedido de 150 unidades.
- Em seguida, 10 pedidos de 8 unidades.
- E então, um pedido ainda maior de 200 unidades.

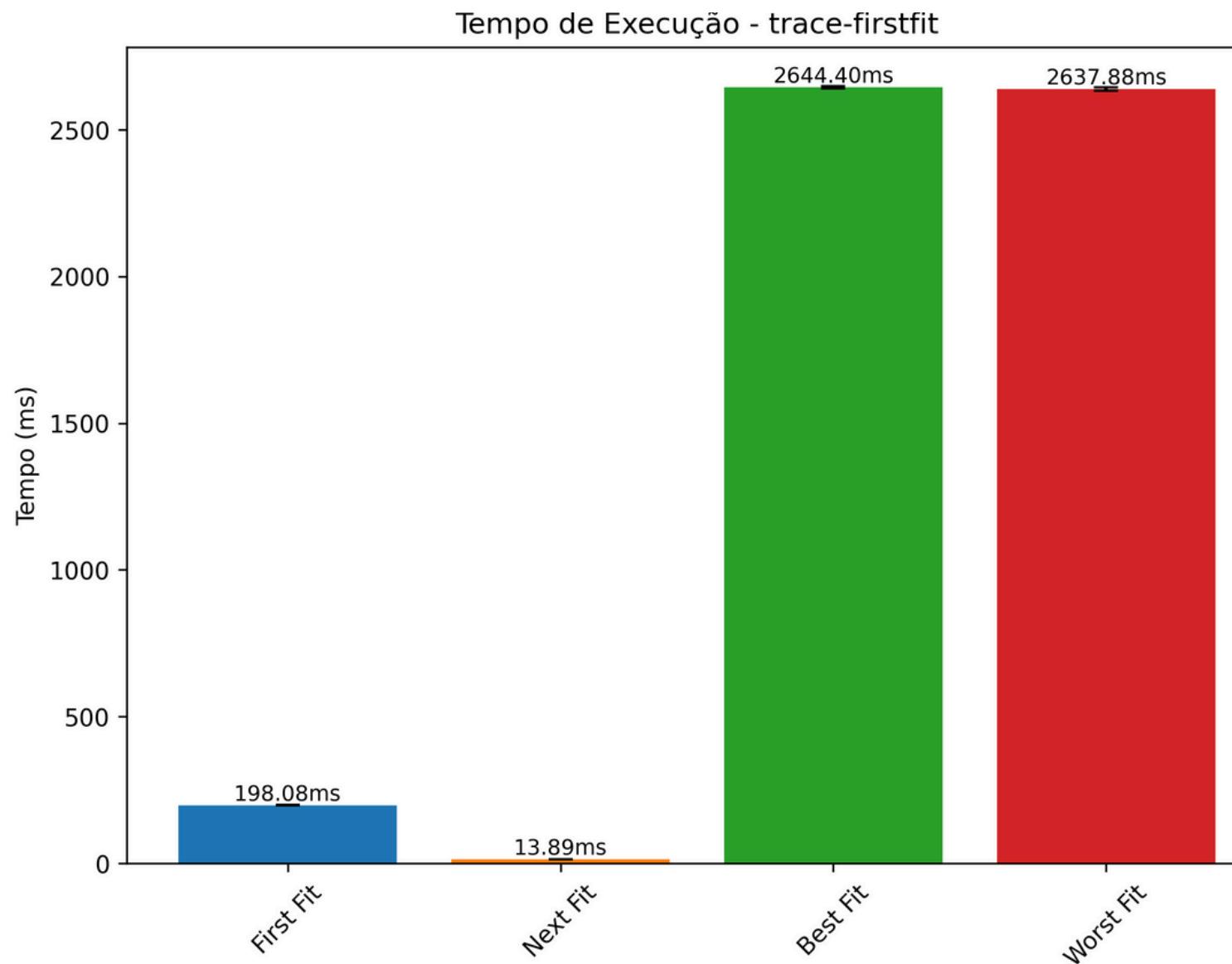
Como o First Fit reage a isso:

1. Ele agrupa todas as alocações pequenas no início da memória, uma após a outra.
2. Quando a requisição grande (de 150 unidades) chega, a busca começa do início, acaba pulando os pequenos blocos já ocupados e encontra rapidamente o enorme espaço livre logo em seguida.
3. O resultado é que tanto os blocos pequenos quanto os grandes são alocados com sucesso e rapidez.
4. O First Fit acaba preservando grandes blocos de memória mesmo sendo bem mais eficiente (em questão de tempo) que o Best Fit

Agiu como esperado?

Podemos assumir que agiu como esperado, alocando todos os seus blocos no começo e tendo uma performance próxima a do Next Fit (se comparado com os outros dois algoritmos), **agiu como esperado**

trace-firstfit



Por que o trace-firstfit favorece este algoritmo?

O arquivo trace-firstfit foi desenhado com um padrão específico para ressaltar as qualidades do First Fit: muitas alocações pequenas seguidas por uma alocação grande, repetidamente.

Exemplo do Padrão no Trace:

- Primeiro, 10 pedidos de 10 unidades cada.
- Logo depois, um grande pedido de 150 unidades.
- Em seguida, 10 pedidos de 8 unidades.
- E então, um pedido ainda maior de 200 unidades.

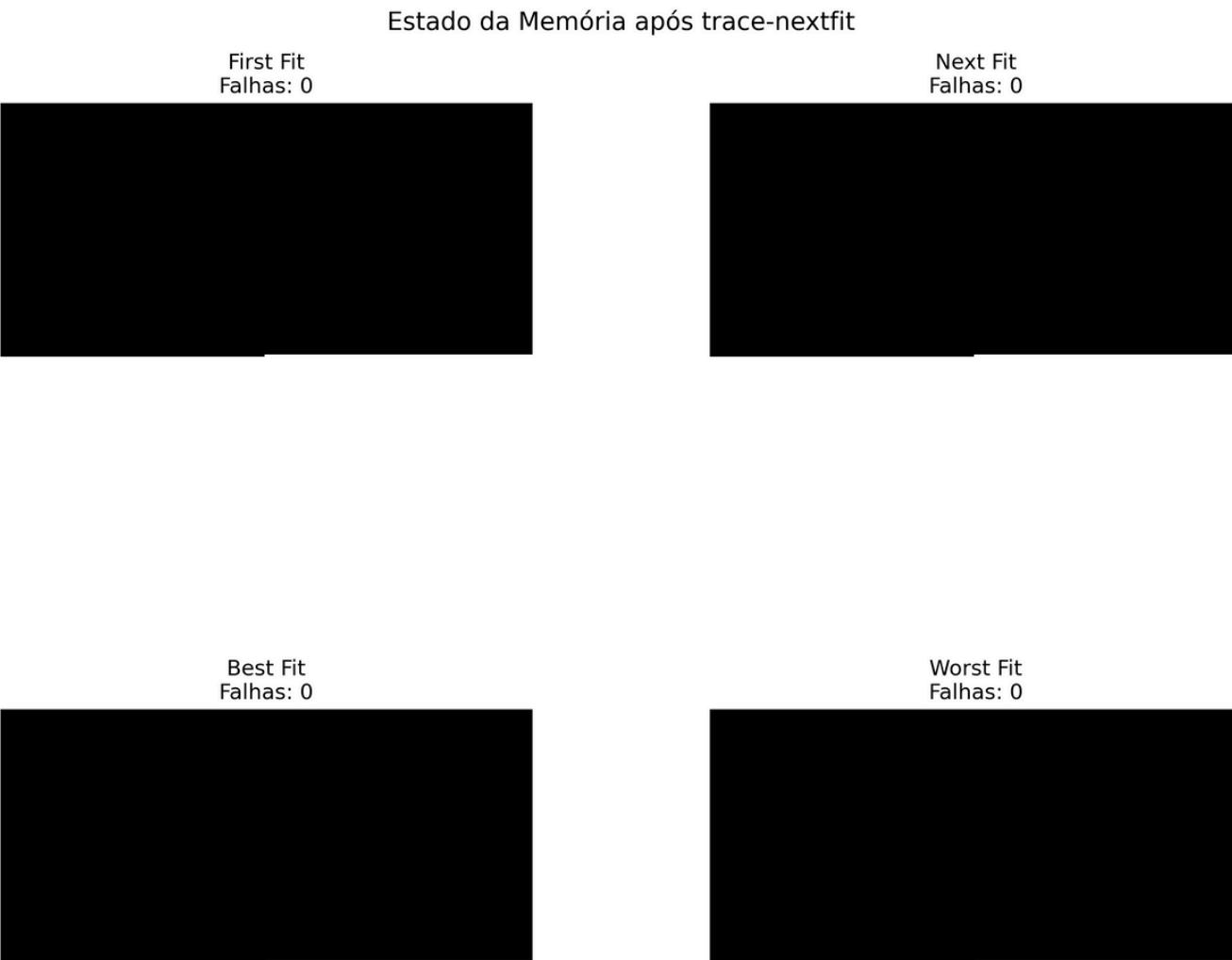
Como o First Fit reage a isso:

1. Ele agrupa todas as alocações pequenas no início da memória, uma após a outra.
2. Quando a requisição grande (de 150 unidades) chega, a busca começa do início, acaba pulando os pequenos blocos já ocupados e encontra rapidamente o enorme espaço livre logo em seguida.
3. O resultado é que tanto os blocos pequenos quanto os grandes são alocados com sucesso e rapidez.
4. O First Fit acaba preservando grandes blocos de memória mesmo sendo bem mais eficiente (em questão de tempo) que o Best Fit

Agiu como esperado?

Podemos assumir que agiu como esperado, alocando todos os seus blocos no começo e tendo uma performance próxima a do Next Fit (se comparado com os outros dois algoritmos), **agiu como esperado**

trace-nextfit



Por que o trace-nextfit favorece este algoritmo?

O trace foi desenhado para simular um fluxo constante de trabalho com requisições de tamanho similar, sem grandes variações.

Tudo isso ocorre após um **COMPACTAR** inicial, que faz com que um grande bloco de memória livre se abra. O algoritmo Next-Fit achará para todos as linhas a seguir, memórias livres mais rapidamente que todos os outros algoritmos mostrando principalmente sua eficiência

Análise do Padrão no Trace:

- **1 COMPACTAR:** O trace começa compactando a memória, isso garante que, após achar a primeira célula livre de memória e começar as suas buscas daquele ponto, será muito rápido achar espaço para qualquer alocação, ganhando uma vantagem muito grande sob qualquer outro algoritmo
- **Fluxo Contínuo:** O arquivo consiste em 200 requisições sequenciais de tamanhos médios (entre 15 e 80). Não há alocações gigantescas nem muito pequenas.

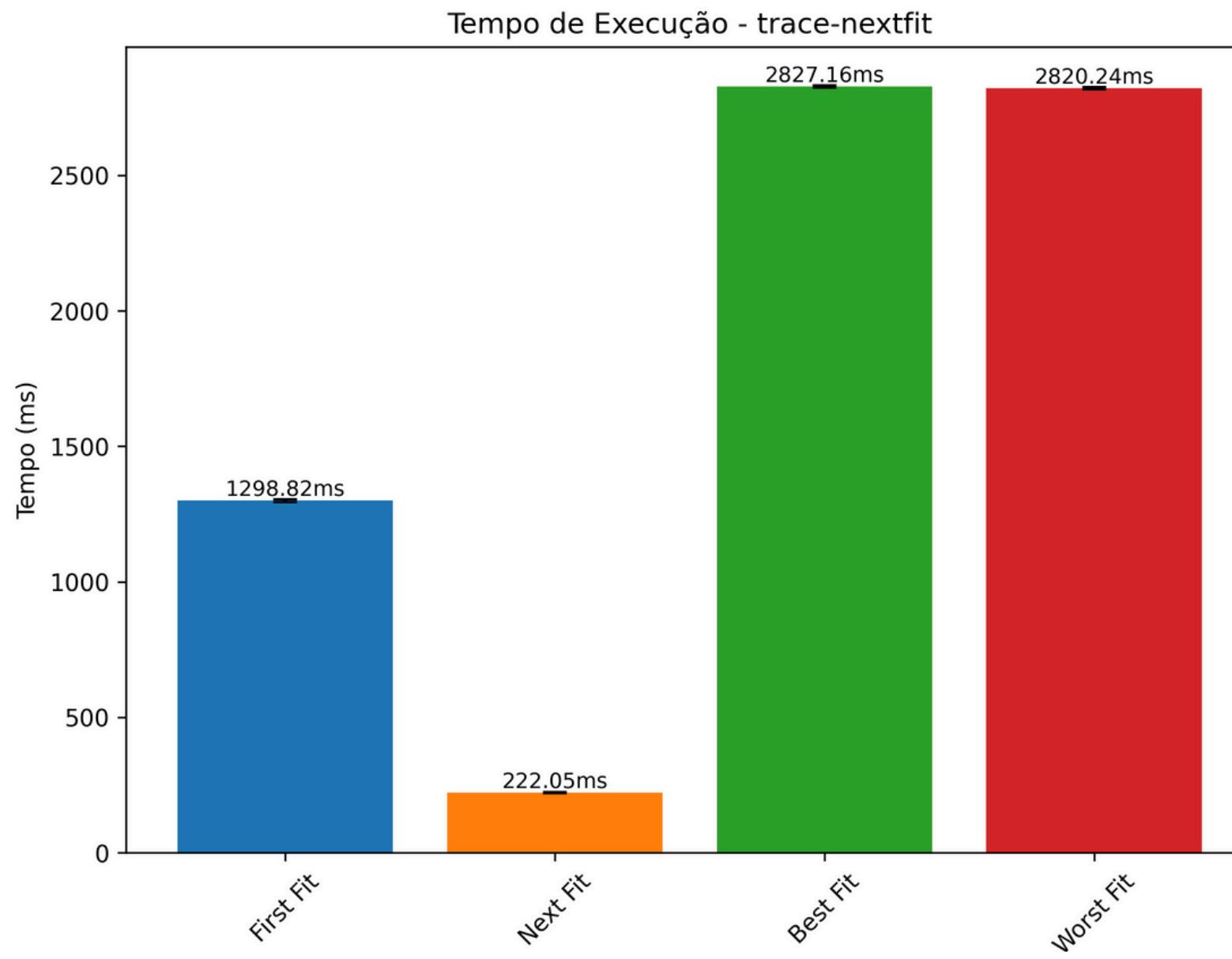
Como o Next Fit reage a isso:

1. Ele aloca o primeiro bloco no início da memória.
2. Para a segunda requisição, ele começa a busca logo após o fim do primeiro bloco, e assim por diante.
3. Ele se comporta como uma "onda" que vai preenchendo a memória de forma progressiva e distribuída. A busca por espaço é quase sempre instantânea, pois o próximo espaço livre está logo adiante.

Agiu como esperado?

Podemos assumir que agiu como esperado, sendo de longe o algoritmo mais eficiente para este caso em específico onde a memória está nada fragmentada (após a compactação), **agiu como esperado**

trace-nextfit



Por que o trace-nextfit favorece este algoritmo?

O trace foi desenhado para simular um fluxo constante de trabalho com requisições de tamanho similar, sem grandes variações.

Tudo isso ocorre após um **COMPACTAR** inicial, que faz com que um grande bloco de memória livre se abra. O algoritmo Next-Fit achará para todos as linhas a seguir, memórias livres mais rapidamente que todos os outros algoritmos mostrando principalmente sua eficiência

Análise do Padrão no Trace:

- **1 COMPACTAR:** O trace começa compactando a memória, isso garante que, após achar a primeira célula livre de memória e começar as suas buscas daquele ponto, será muito rápido achar espaço para qualquer alocação, ganhando uma vantagem muito grande sobre qualquer outro algoritmo
- **Fluxo Contínuo:** O arquivo consiste em 200 requisições sequenciais de tamanhos médios (entre 15 e 80). Não há alocações gigantescas nem muito pequenas.

Como o Next Fit reage a isso:

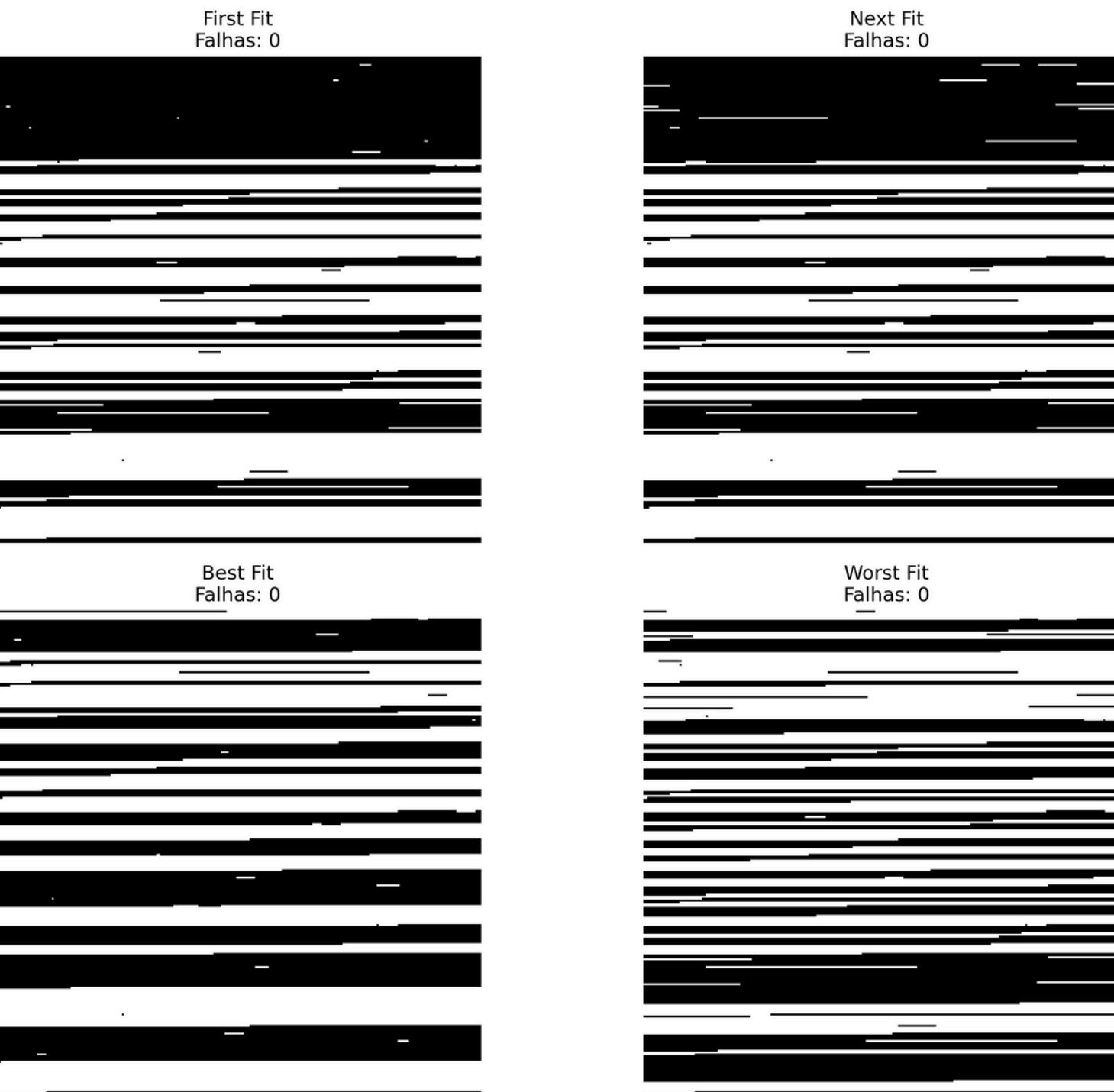
1. Ele aloca o primeiro bloco no início da memória.
2. Para a segunda requisição, ele começa a busca logo após o fim do primeiro bloco, e assim por diante.
3. Ele se comporta como uma "onda" que vai preenchendo a memória de forma progressiva e distribuída. A busca por espaço é quase sempre instantânea, pois o próximo espaço livre está logo adiante.

Agiu como esperado?

Podemos assumir que agiu como esperado, sendo de longe o algoritmo mais eficiente para este caso em específico onde a memória está nada fragmentada (após a compactação), **agiu como esperado**

trace-bestfit

Estado da Memória após trace-bestfit



Por que o trace-bestfit favorece este algoritmo?

Este trace foi desenhado para simular um ambiente caótico e imprevisível, com uma mistura completamente aleatória de requisições pequenas, médias e grandes.

Análise do Padrão no Trace:

- O arquivo contém uma grande variedade de tamanhos: 50, 30, 20, 100, 15, 25, 35, 45, 10, ...
- Não há um padrão claro. É uma simulação realista de um sistema onde diferentes processos com diferentes necessidades de memória competem por recursos.

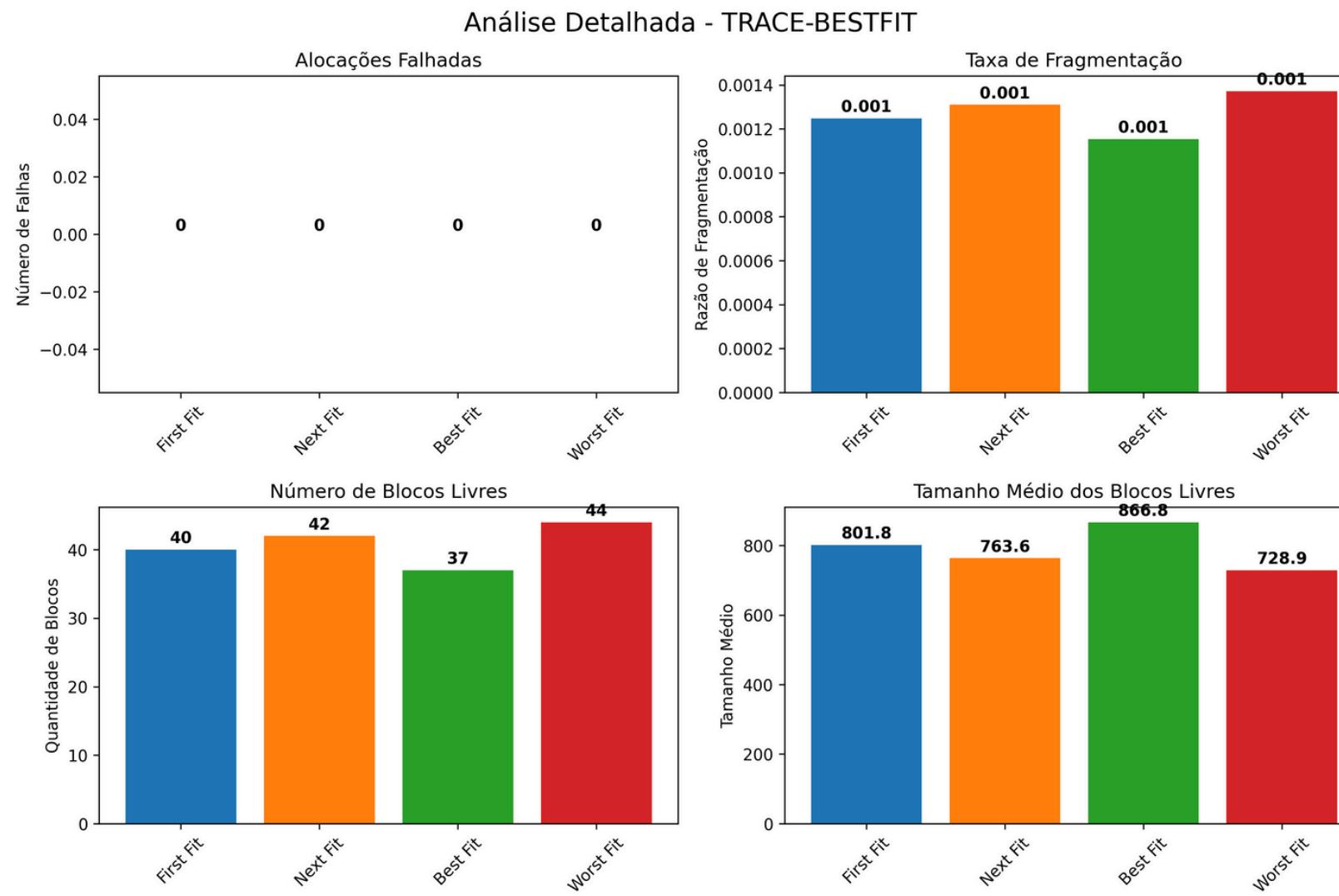
Como o Best Fit reage a isso:

- Para uma requisição de 20 unidades, ele vai procurar por toda a memória e, se encontrar blocos livres de 100, 50 e 22, ele escolherá o de 22.
- Essa decisão deixa um fragmento mínimo (de 2 unidades), mas o mais importante é que os blocos de 100 e 50 permanecem intactos, prontos para futuras requisições grandes.
- Ele gera a maior média de blocos livres, assim como a menor taxa de fragmentação de todos os algoritmos (objetivo principal do algoritmo)

Agiu como esperado?

Podemos assumir que agiu como esperado, pois acabou criando um cenário que condiz com seu objetivo. Evitar uma memória muito fragmentada e espaços muito pequenos de memória disponível, afim de deixar espaço para caso chegue algum processo que precise alocar muita memória, logo **agiu como esperado**

trace-bestfit



Por que o trace-bestfit favorece este algoritmo?

Este trace foi desenhado para simular um ambiente caótico e imprevisível, com uma mistura completamente aleatória de requisições pequenas, médias e grandes.

Análise do Padrão no Trace:

- O arquivo contém uma grande variedade de tamanhos: 50, 30, 20, 100, 15, 25, 35, 45, 10, ...
- Não há um padrão claro. É uma simulação realista de um sistema onde diferentes processos com diferentes necessidades de memória competem por recursos.

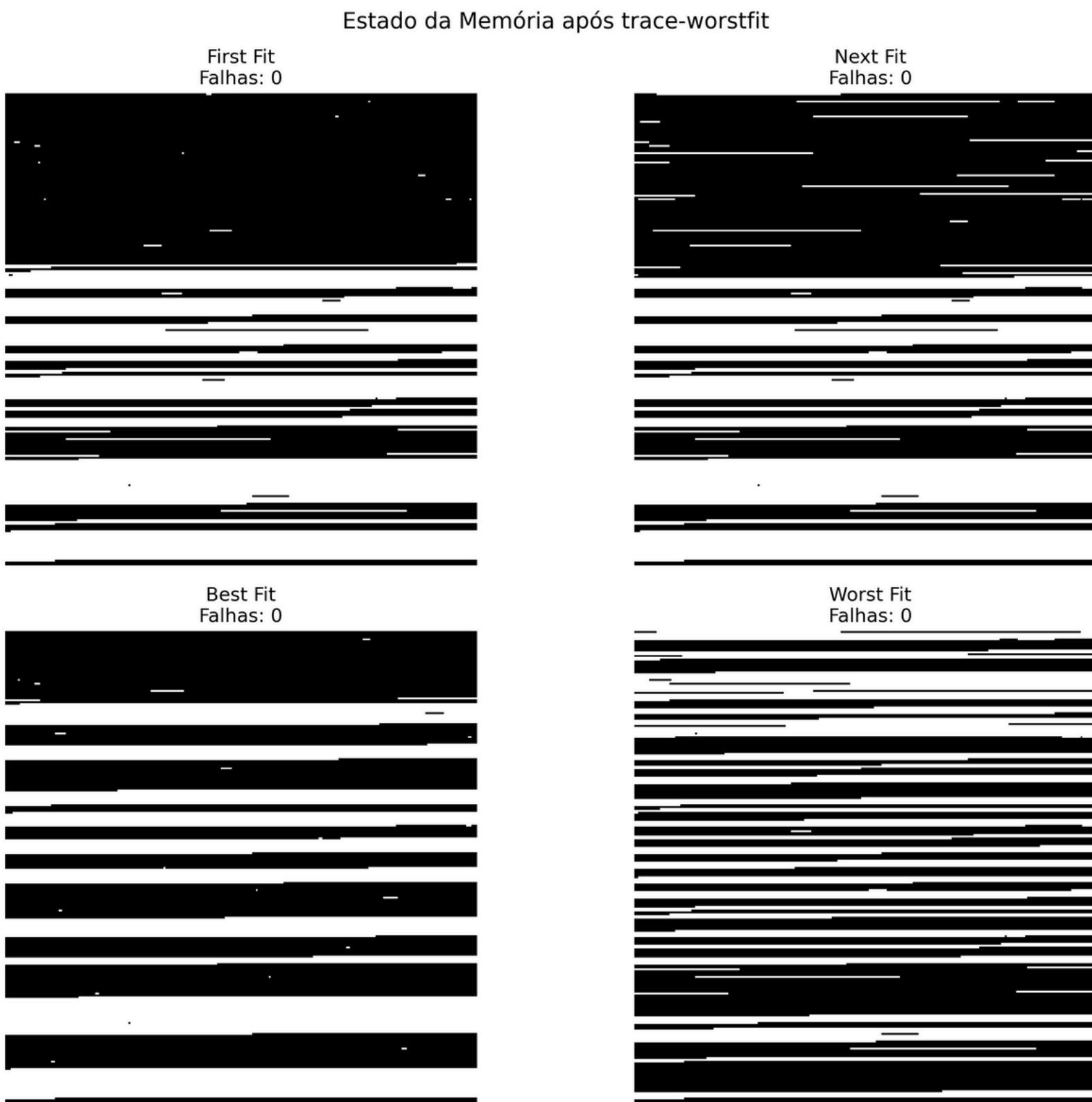
Como o Best Fit reage a isso:

- Para uma requisição de 20 unidades, ele vai procurar por toda a memória e, se encontrar blocos livres de 100, 50 e 22, ele escolherá o de 22.
- Essa decisão deixa um fragmento mínimo (de 2 unidades), mas o mais importante é que os blocos de 100 e 50 permanecem intactos, prontos para futuras requisições grandes.
- Ele gera a maior média de blocos livres, assim como a menor taxa de fragmentação de todos os algoritmos (objetivo principal do algoritmo)

Agiu como esperado?

Podemos assumir que agiu como esperado, pois acabou criando um cenário que condiz com seu objetivo. Evitar uma memória muito fragmentada e espaços muito pequenos de memória disponível, afim de deixar espaço para caso chegue algum processo que precise alocar muita memória, logo **agiu como esperado**

trace-worstfit



O que o Worst Fit faz de melhor?

- Deixa o Maior Fragmento Restante: Ao alocar espaço no maior bloco livre, ele garante que o “buraco” que sobra seja o maior possível. A teoria é que um grande fragmento restante é mais útil para futuras alocações do que um fragmento minúsculo.
- Evita “Buraquinhos” de Memória: O principal “inimigo” do Worst Fit é o Best Fit. O Best Fit procura um encaixe perfeito e, se não o encontra, pode deixar um fragmento de 1 ou 2 unidades (os chamados “buraquinhos”). O Worst Fit, ao usar o bloco maior, evita criar esses pequenos fragmentos inúteis.

Por que o trace-worstfit favorece este algoritmo?

O trace foi desenhado para simular um padrão específico: uma alternância constante entre requisições grandes e pequenas.

Análise do Padrão no Trace:

- O arquivo intercala pedidos: 200 (grande), 10 (pequeno), 180 (grande), 15 (pequeno), 160 (grande), 8 (pequeno), e assim por diante.
- Esse padrão força os algoritmos a decidirem: "Para atender a este pedido pequeno, devo 'gastar' um bloco grande ou usar um bloco pequeno que se encaixe melhor?"

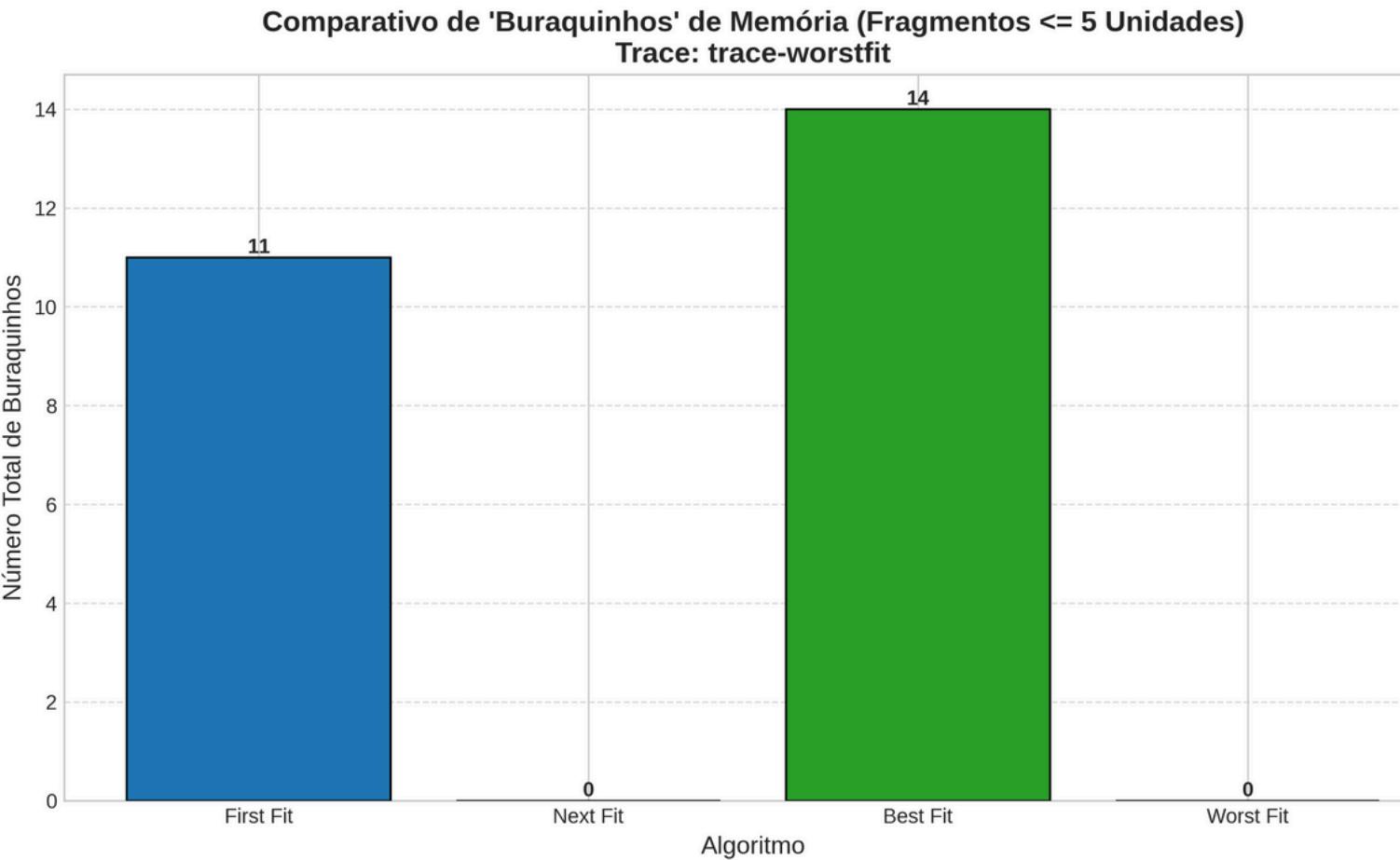
Como o Worst Fit reage a isso (supondo memória já fragmentada):

- Tanto para a requisição de 200 quanto para a de 10, ele irá atacar o maior bloco de memória disponível.
- O efeito colateral disso é que ele deixa todos os blocos de tamanho médio e pequeno intocados. Ele "sacrifica" o maior bloco para proteger os demais.

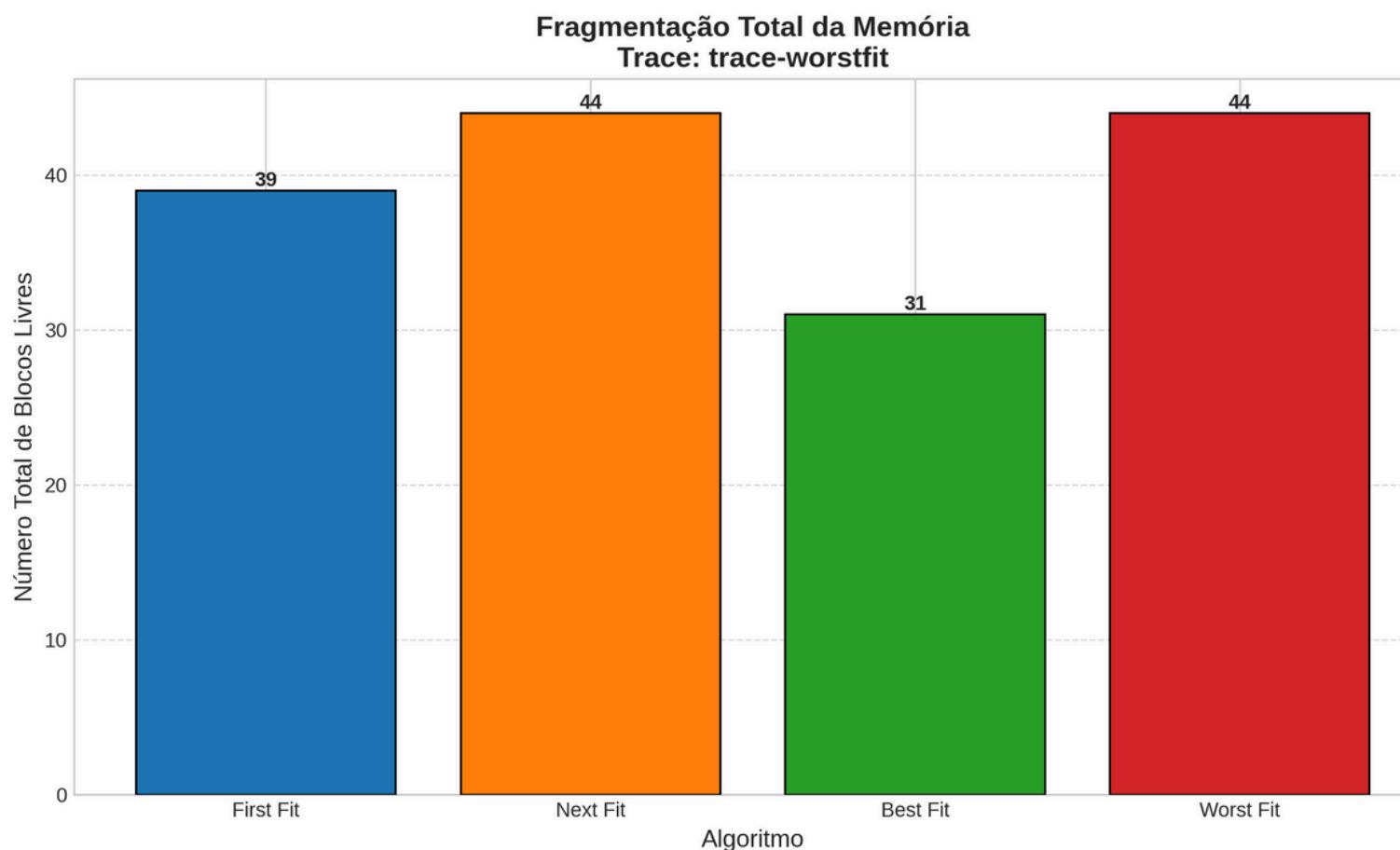
Agiu como esperado?

Podemos enxergar, na visualização da memória que todos os outros algoritmos deixam pequenos espaços praticamente inutilizáveis em seus blocos de memórias maiores enquanto, apesar de bem fragmentado visualmente, o Worst Fit mantém apenas espaços grande o suficiente para armazenar a memória de processos, logo **agiu como esperado**

trace-worstfit



apesar de possuir a mesma quantidade de fragmentos de memoria livre, os fragmentos do Worst Fit são maiores e mais úteis que o do Next Fit



O que o Worst Fit faz de melhor?

- Deixa o Maior Fragmento Restante: Ao alocar espaço no maior bloco livre, ele garante que o "buraco" que sobra seja o maior possível. A teoria é que um grande fragmento restante é mais útil para futuras alocações do que um fragmento minúsculo.
- Evita "Buraquinhos" de Memória: O principal "inimigo" do Worst Fit é o Best Fit. O Best Fit procura um encaixe perfeito e, se não o encontra, pode deixar um fragmento de 1 ou 2 unidades (os chamados "buraquinhos"). O Worst Fit, ao usar o bloco maior, evita criar esses pequenos fragmentos inúteis.

Por que o trace-worstfit favorece este algoritmo?

O trace foi desenhado para simular um padrão específico: uma alternância constante entre requisições grandes e pequenas.

Análise do Padrão no Trace:

- O arquivo intercala pedidos: 200 (grande), 10 (pequeno), 180 (grande), 15 (pequeno), 160 (grande), 8 (pequeno), e assim por diante.
- Esse padrão força os algoritmos a decidirem: "Para atender a este pedido pequeno, devo 'gastar' um bloco grande ou usar um bloco pequeno que se encaixe melhor?"

Como o Worst Fit reage a isso (supondo memória já fragmentada):

- Tanto para a requisição de 200 quanto para a de 10, ele irá atacar o maior bloco de memória disponível.
- O efeito colateral disso é que ele deixa todos os blocos de tamanho médio e pequeno intocados. Ele "sacrifica" o maior bloco para proteger os demais.

Agiu como esperado?

Podemos enxergar, na visualização da memória que todos os outros algoritmos deixam pequenos espaços praticamente inutilizaveis em seus blocos de memórias maiores enquanto, apesar de bem fragmentado visualmente, o Worst Fit mantém apenas espaços grande o suficiente para armazenar a memória de processos, logo **agiu como esperado**