MC-202 — Aula 29
Gerenciamento de Memória

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

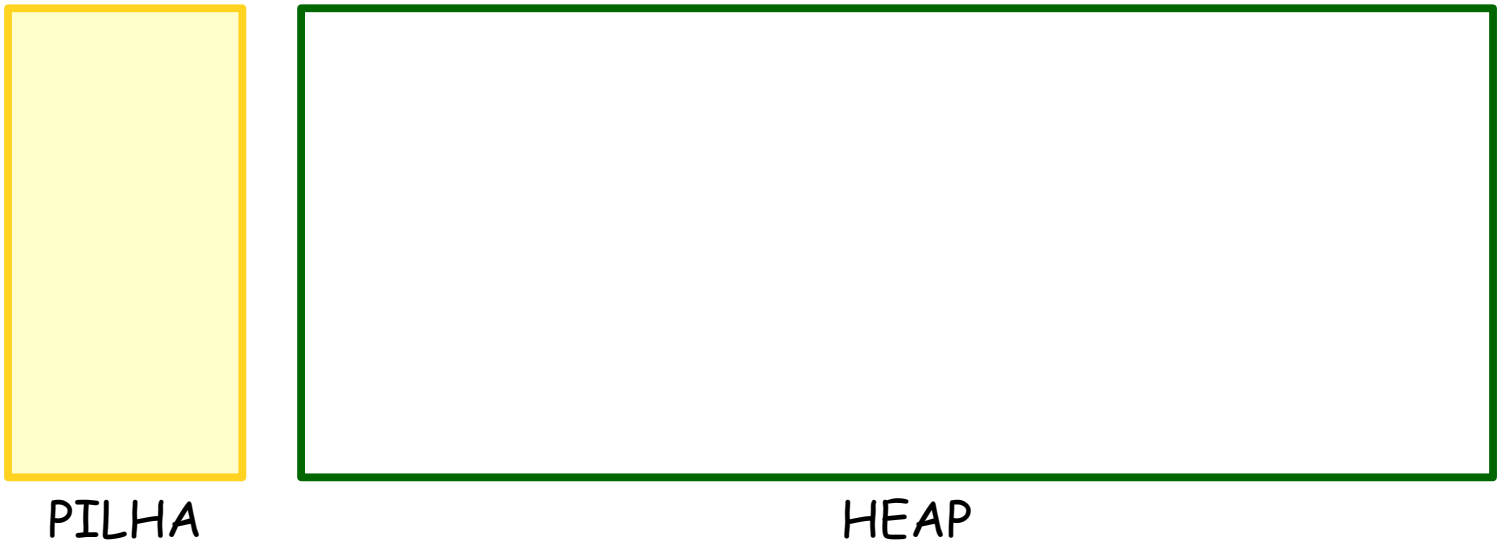
Roteiro

Relembrando

Como a memória está organizada?

Relembrando

Como a memória está organizada?

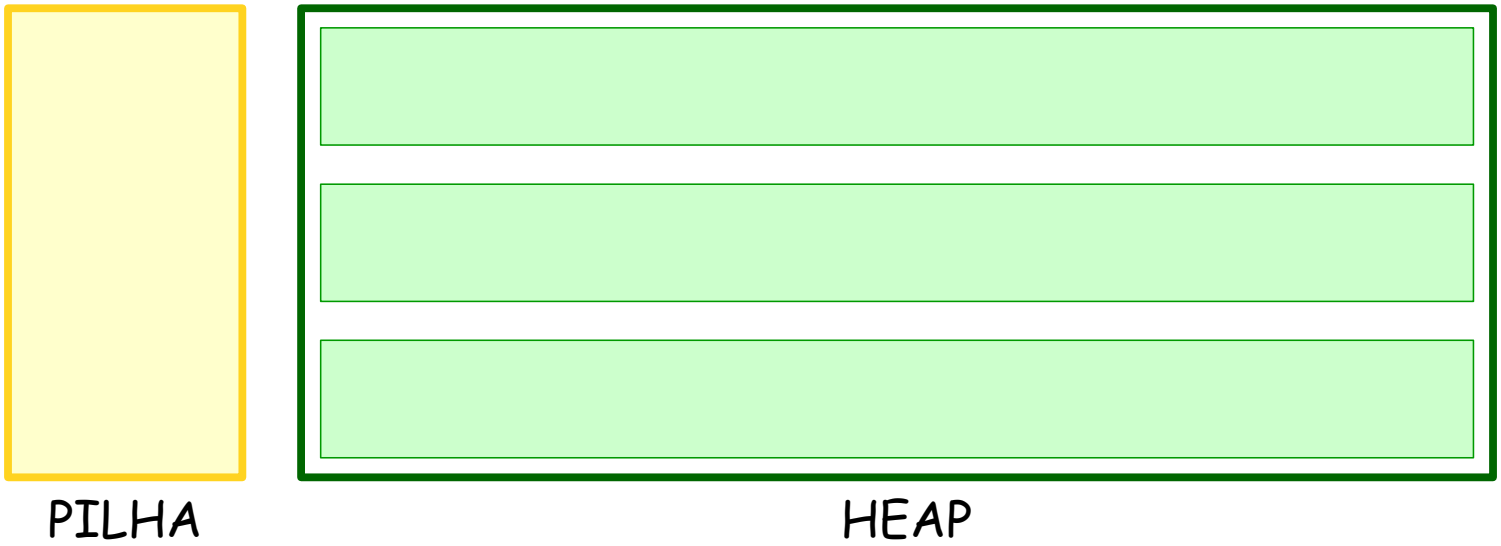


Organização da memória

Dois blocos de memória

Relembrando

Como a memória está organizada?

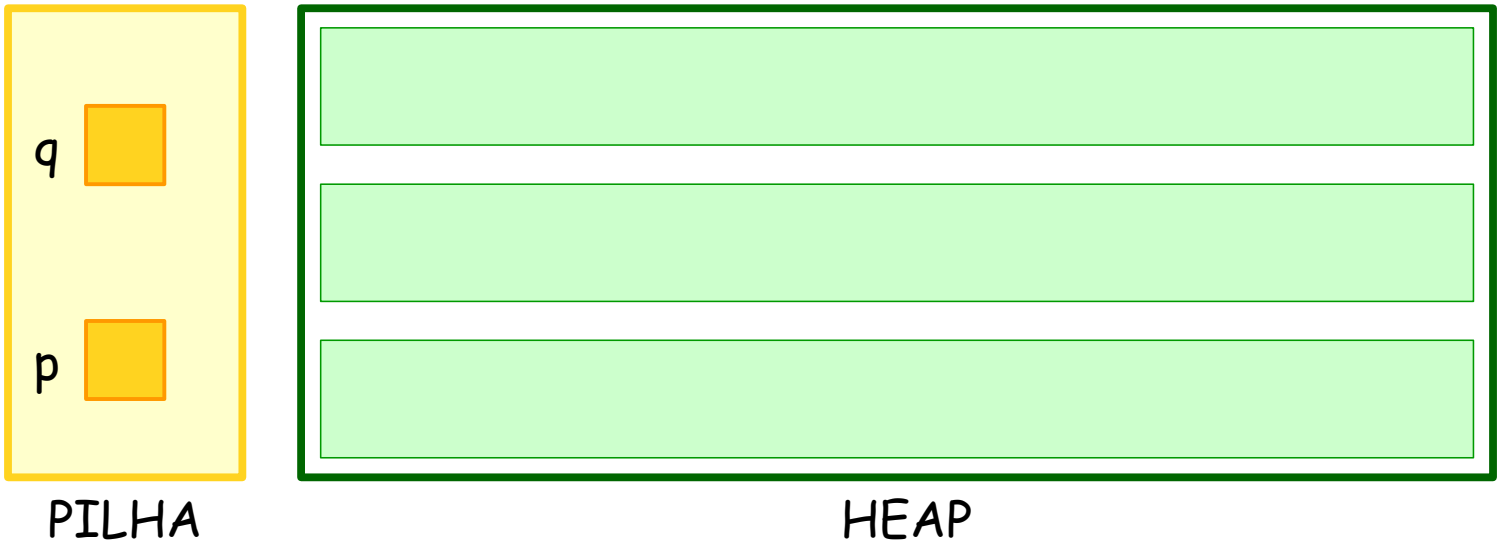


Organização da memória

Dois blocos de memória **lineares**:

Relembrando

Como a memória está organizada?



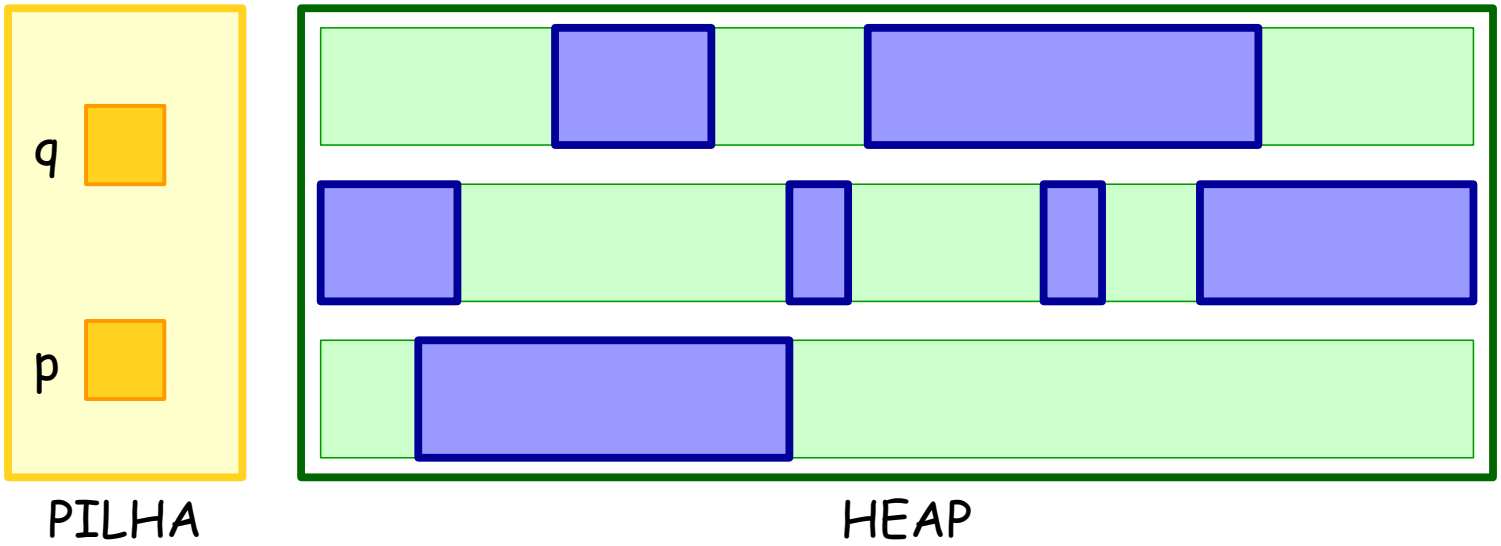
Organização da memória

Dois blocos de memória **lineares**:

- **pilha**: guardamos as variáveis locais

Relembrando

Como a memória está organizada?



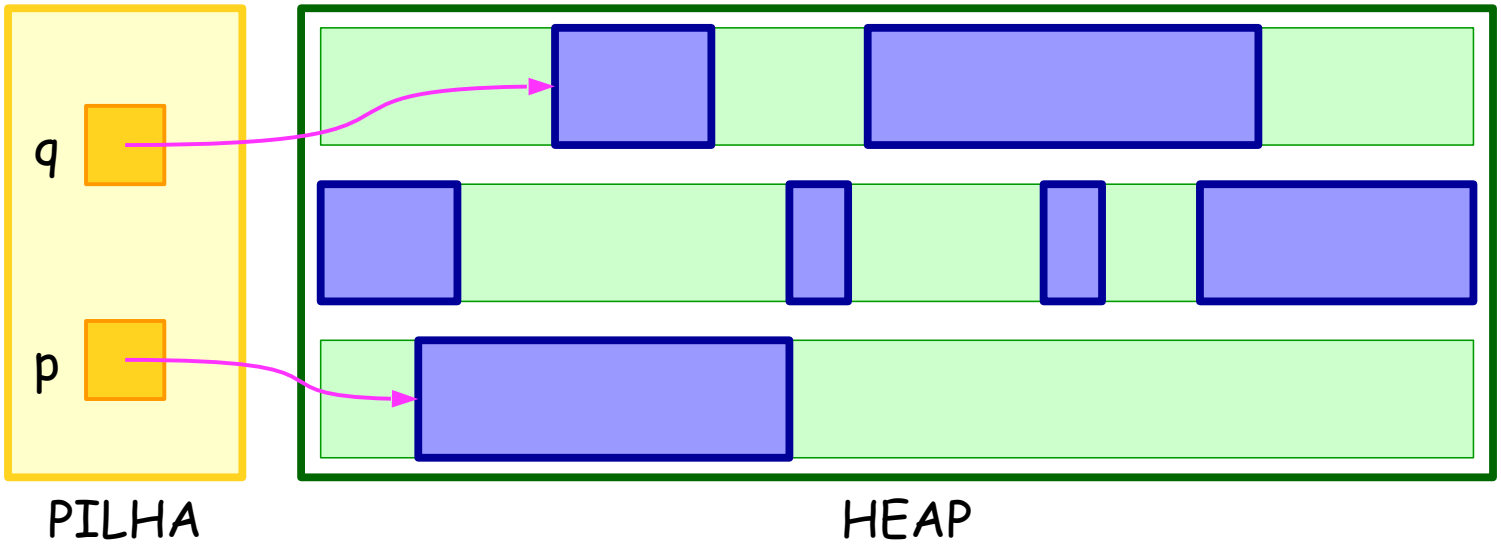
Organização da memória

Dois blocos de memória **lineares**:

- **pilha**: guardamos as variáveis locais
- **heap**: criamos nós dinamicamente

Relembrando

Como a memória está organizada?



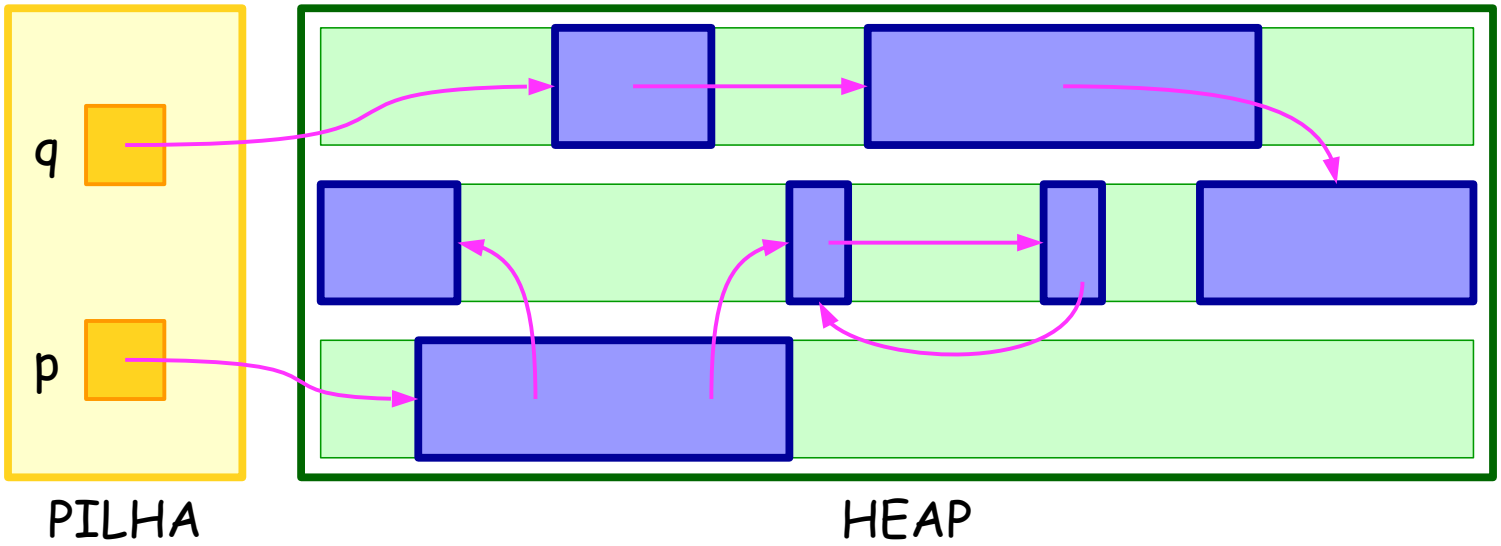
Organização da memória

Dois blocos de memória **lineares**:

- **pilha**: guardamos as variáveis locais (acessamos nós diretamente)
- **heap**: criamos nós dinamicamente

Relembrando

Como a memória está organizada?

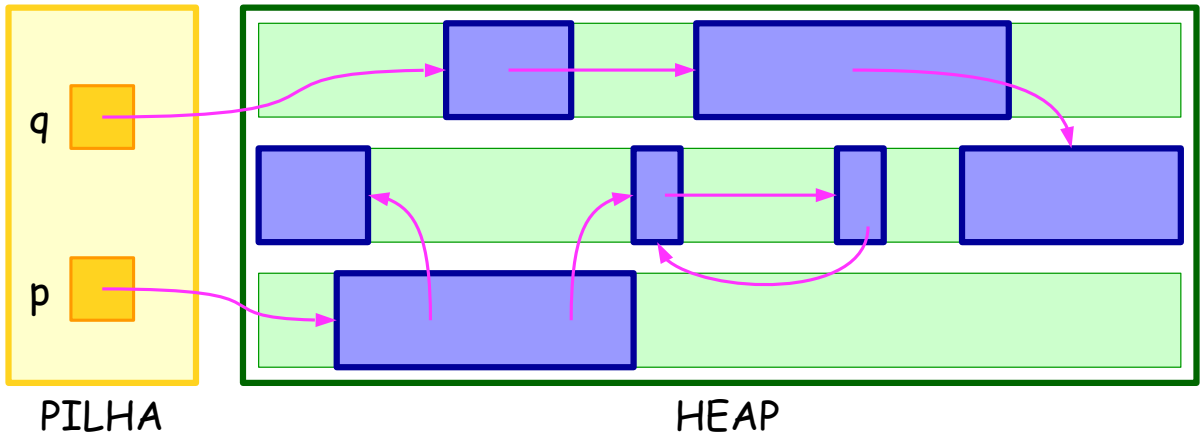


Organização da memória

Dois blocos de memória **lineares**:

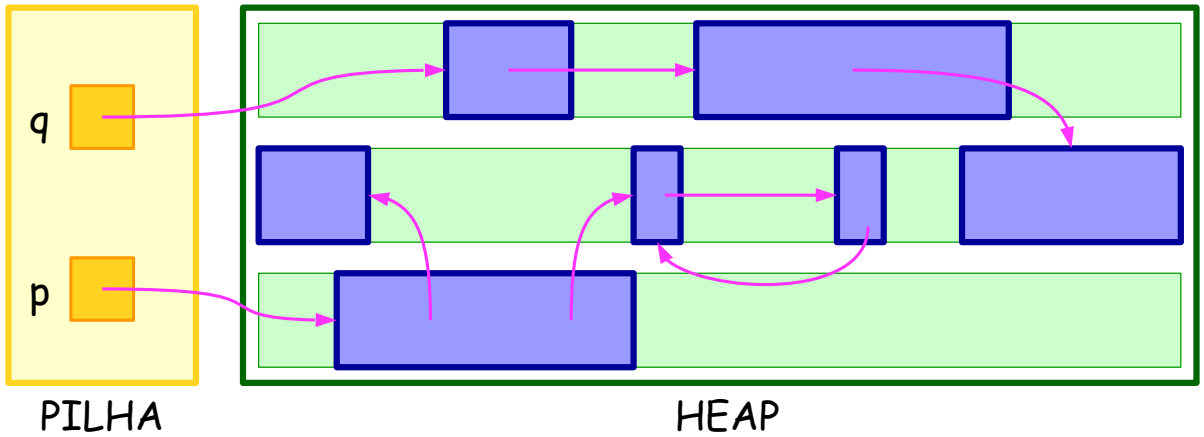
- **pilha**: guardamos as variáveis locais (acessamos nós diretamente)
- **heap**: criamos nós dinamicamente (acessamos nós indiretamente)

Usando o heap



Como usamos o heap?

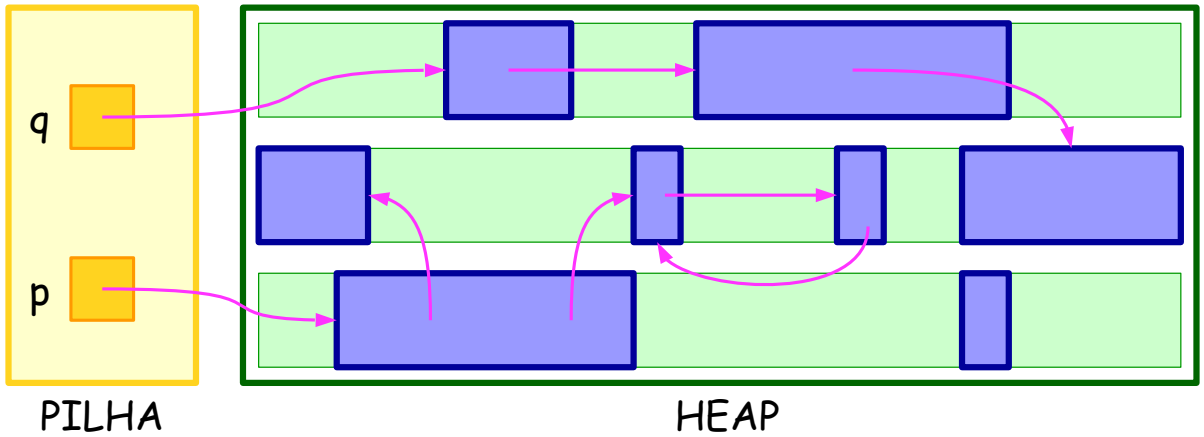
Usando o heap



Como usamos o heap?

- Algumas vezes criamos novos nós

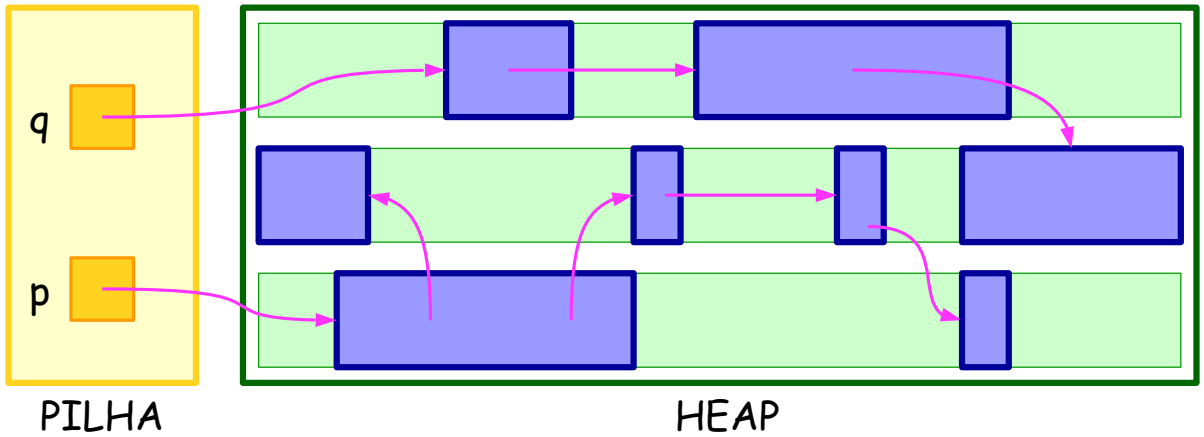
Usando o heap



Como usamos o heap?

- Algumas vezes criamos novos nós

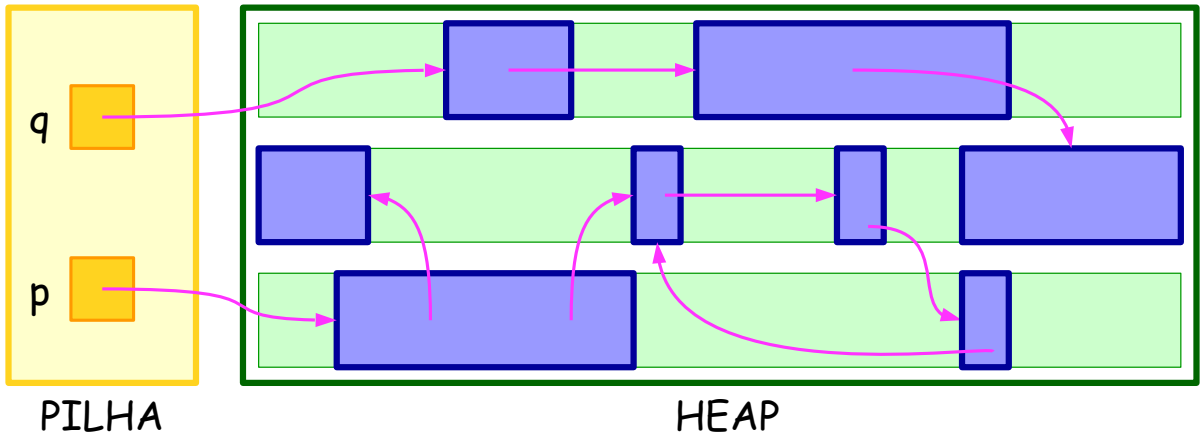
Usando o heap



Como usamos o heap?

- Algumas vezes criamos novos nós

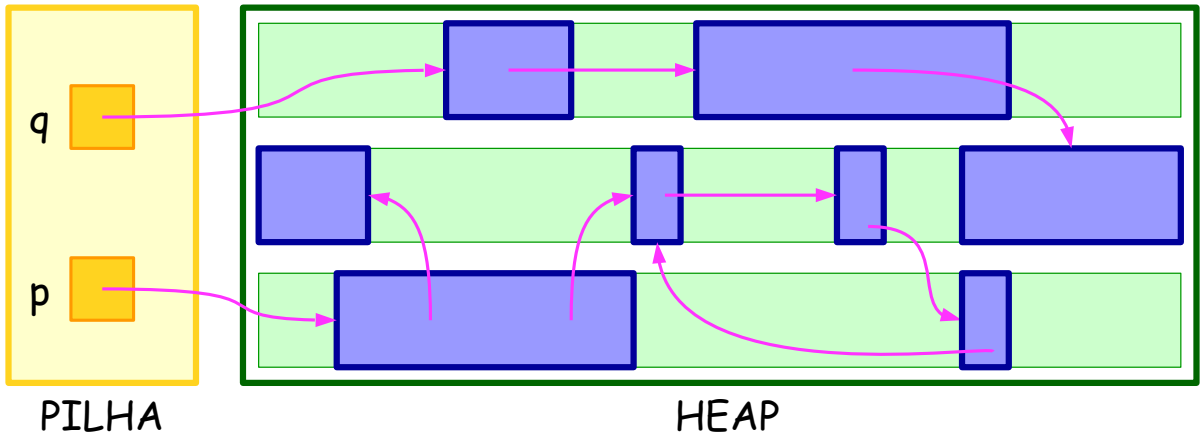
Usando o heap



Como usamos o heap?

- Algumas vezes criamos novos nós

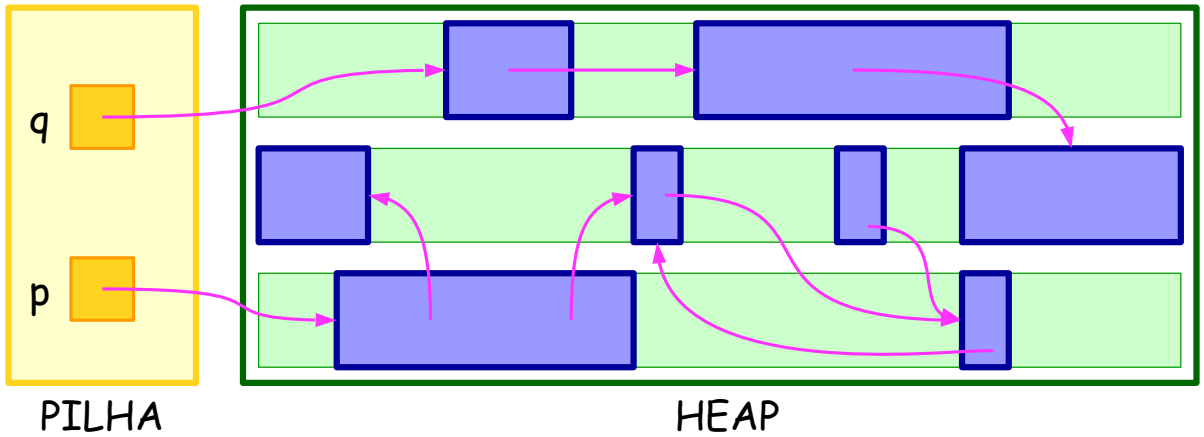
Usando o heap



Como usamos o heap?

- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

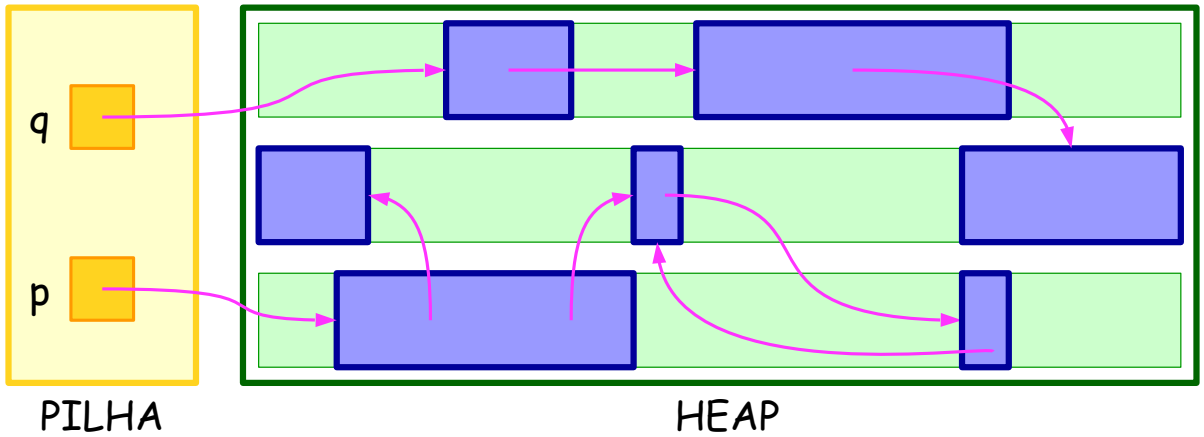
Usando o heap



Como usamos o heap?

- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

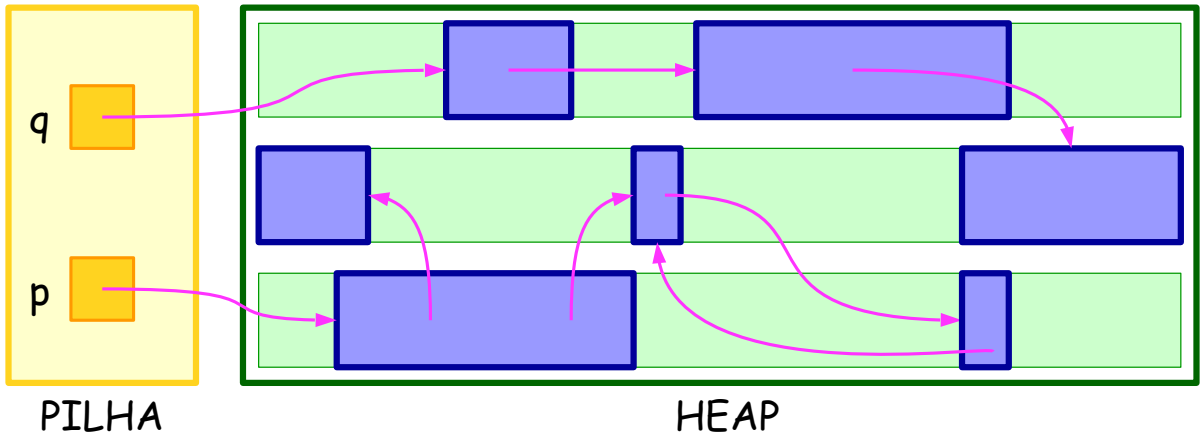
Usando o heap



Como usamos o heap?

- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Usando o heap



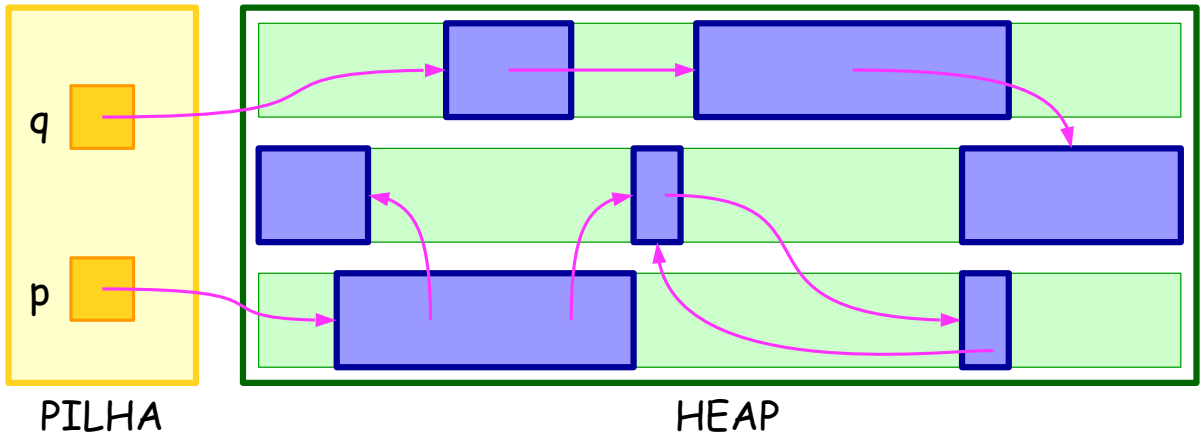
Como usamos o heap?

- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Problema

Organizar a memória

Usando o heap



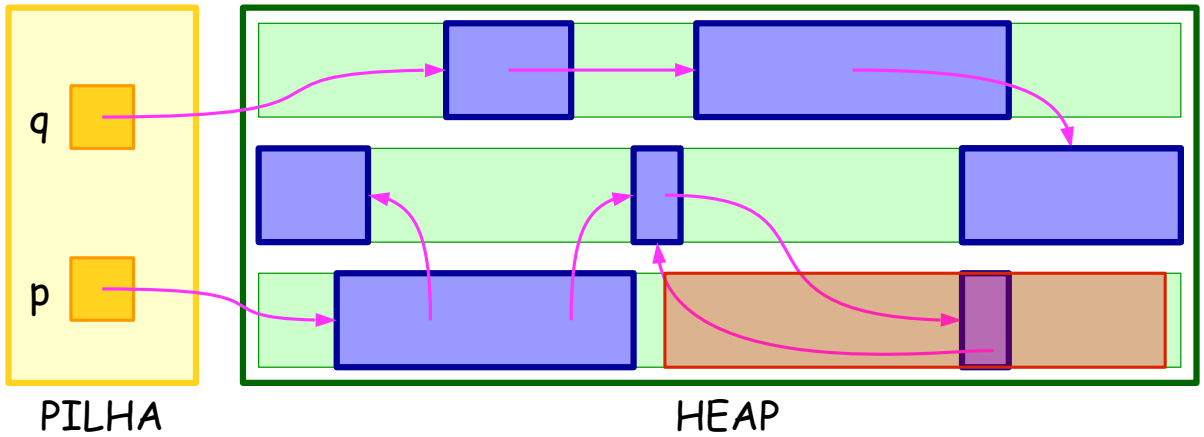
Como usamos o heap?

- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Problema

Organizar a memória: como implementar `malloc` e `free`?

Usando o heap



Como usamos o heap?

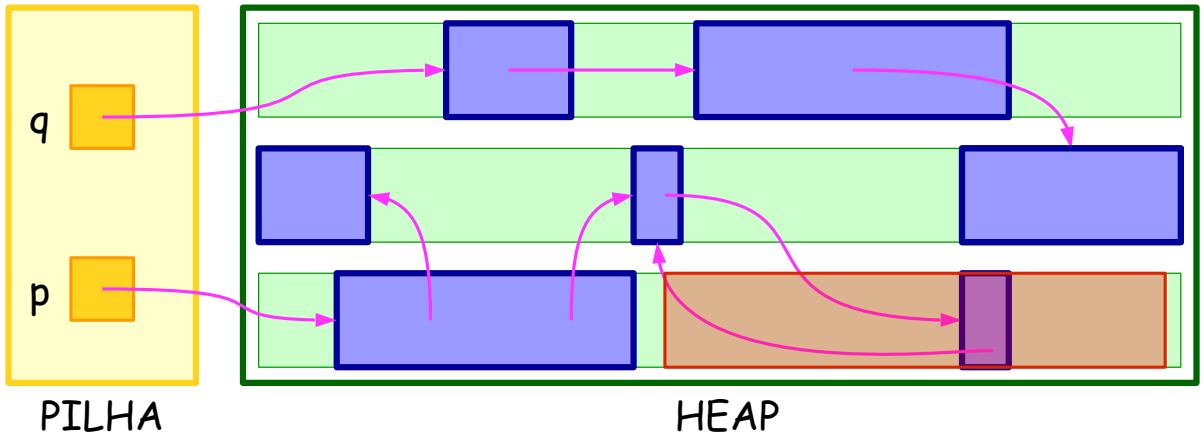
- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Problema

Organizar a memória: como implementar `malloc` e `free`?

- sem sobrescrever nós existentes

Usando o heap



Como usamos o heap?

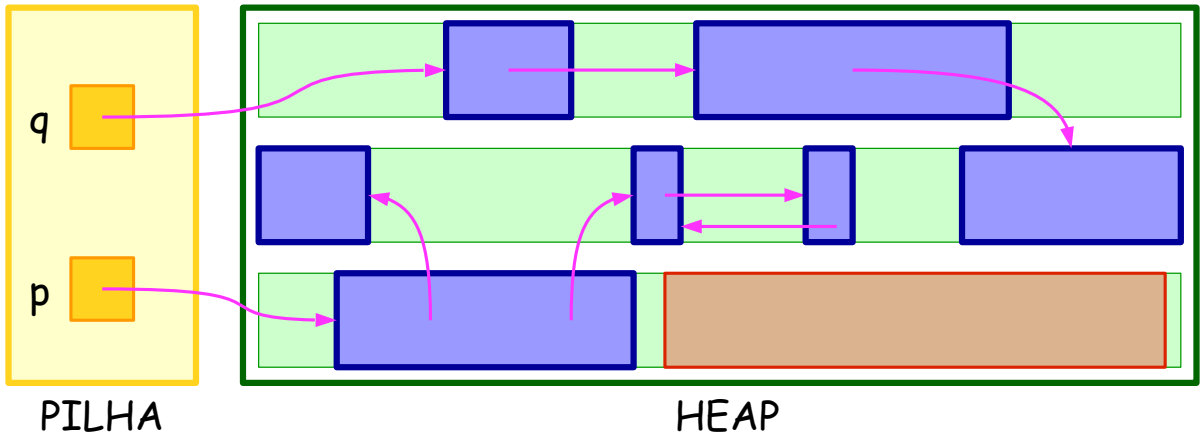
- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Problema

Organizar a memória: como implementar `malloc` e `free`?

- sem sobrescrever nós existentes
- evitando desperdiçar espaço

Usando o heap



Como usamos o heap?

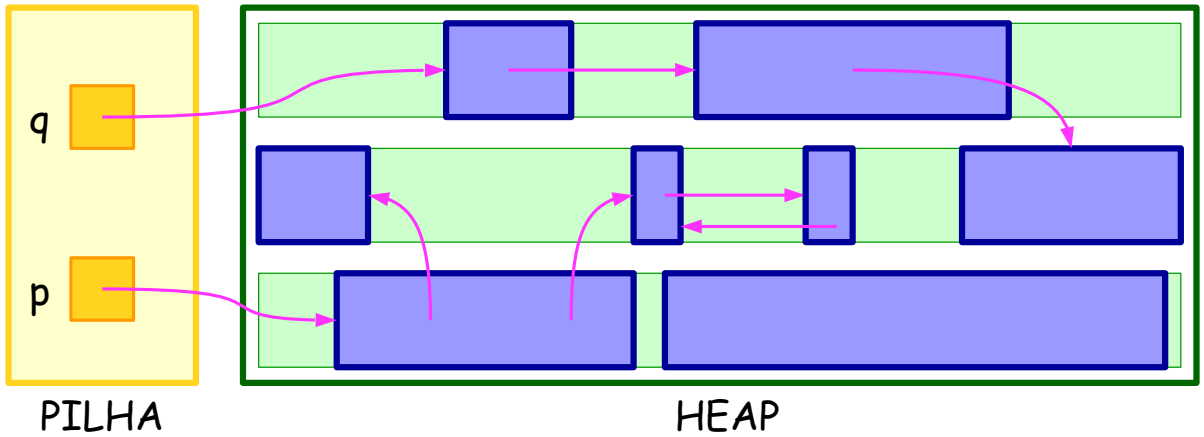
- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Problema

Organizar a memória: como implementar `malloc` e `free`?

- sem sobrescrever nós existentes
- evitando desperdiçar espaço

Usando o heap



Como usamos o heap?

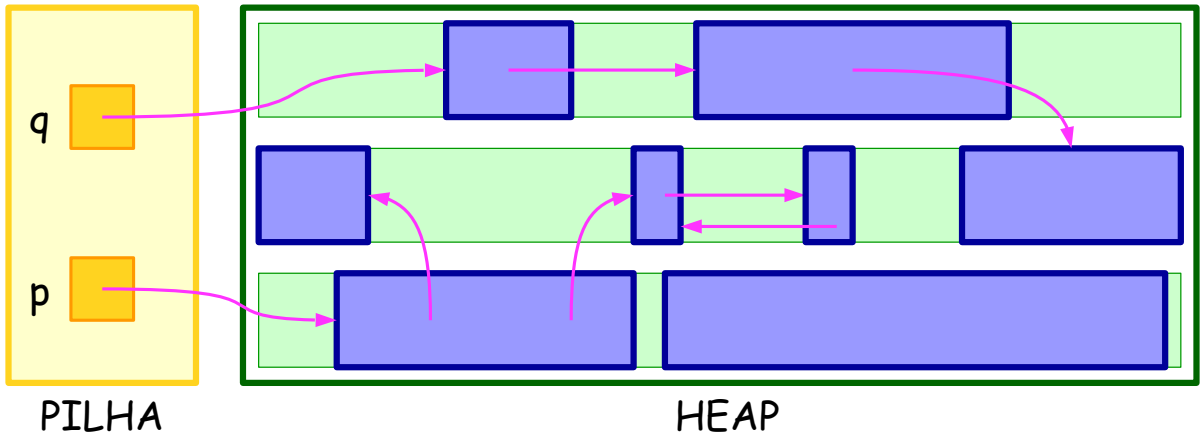
- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Problema

Organizar a memória: como implementar `malloc` e `free`?

- sem sobrescrever nós existentes
- evitando desperdiçar espaço

Usando o heap



Como usamos o heap?

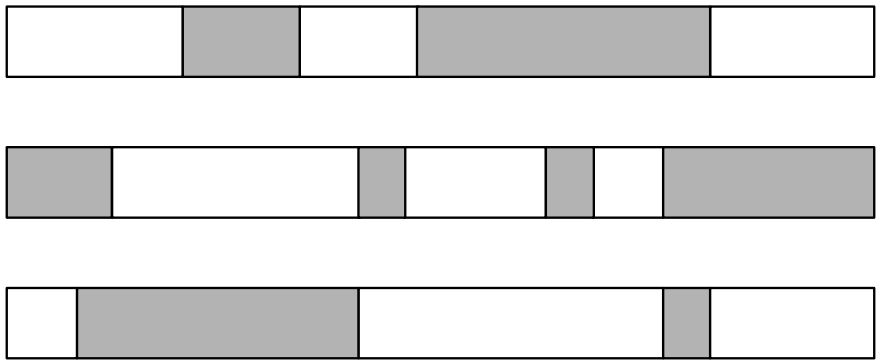
- Algumas vezes criamos novos nós
- Outras vezes removemos nós existentes

Problema

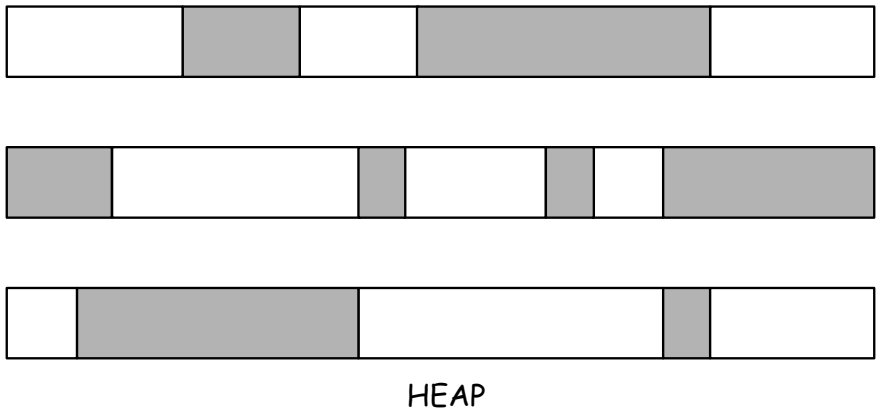
Organizar a memória: como implementar `malloc` e `free`?

- sem sobrescrever nós existentes
- evitando desperdiçar espaço
- eficientemente

Uma ideia: lista dos espaços livres

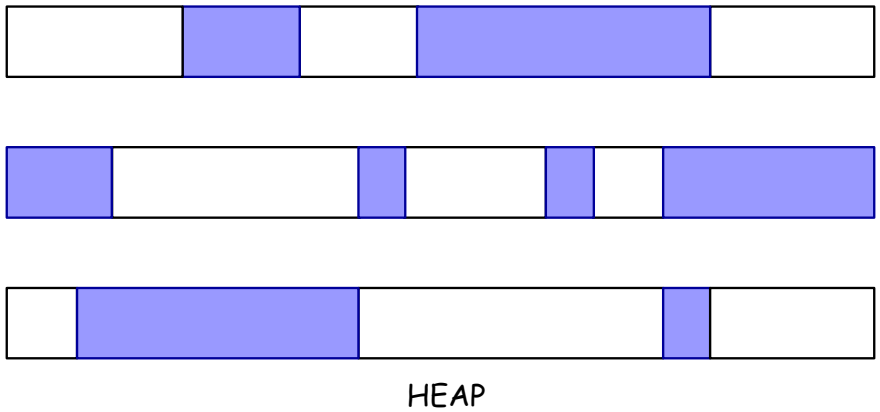


Uma ideia: lista dos espaços livres



Um **bloco** é um espaço contíguo de memória:

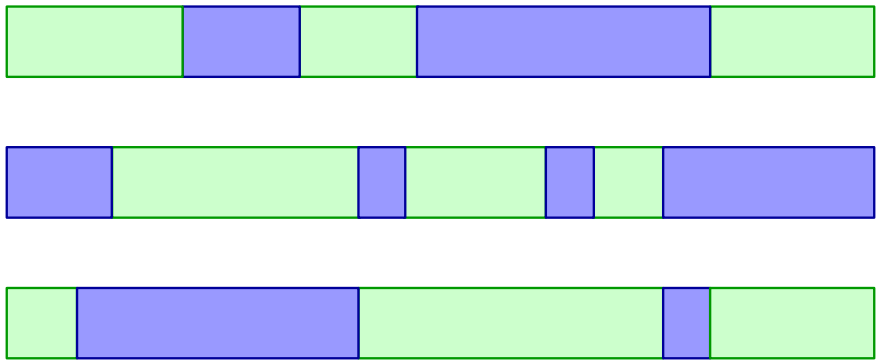
Uma ideia: lista dos espaços livres



Um **bloco** é um espaço contíguo de memória:

- **ocupado**: espaço já reservado para um nó

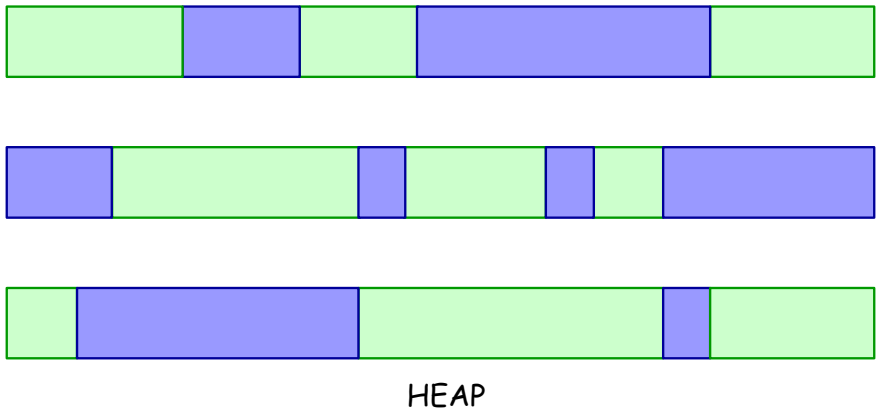
Uma ideia: lista dos espaços livres



Um **bloco** é um espaço contíguo de memória:

- **ocupado**: espaço já reservado para um nó
- **livre**: espaço disponível para criar novos nós; ou

Uma ideia: lista dos espaços livres



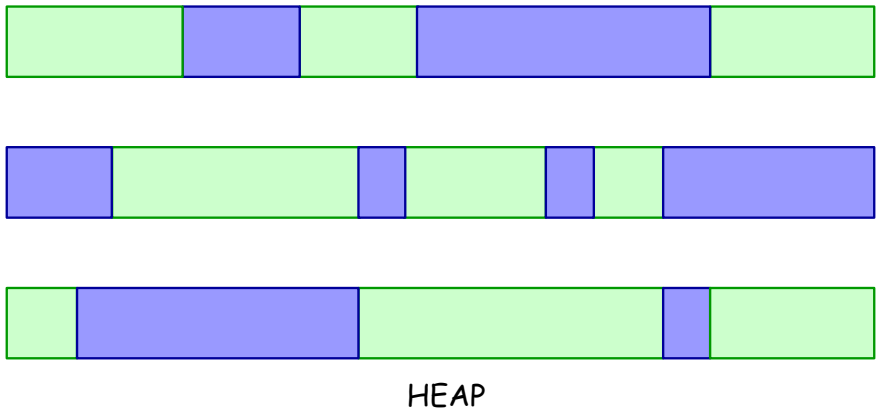
Um **bloco** é um espaço contíguo de memória:

- **ocupado**: espaço já reservado para um nó
- **livre**: espaço disponível para criar novos nós; ou

Lista de blocos livres

É uma lista ligada contendo os blocos **livres** do heap.

Uma ideia: lista dos espaços livres



Um **bloco** é um espaço contíguo de memória:

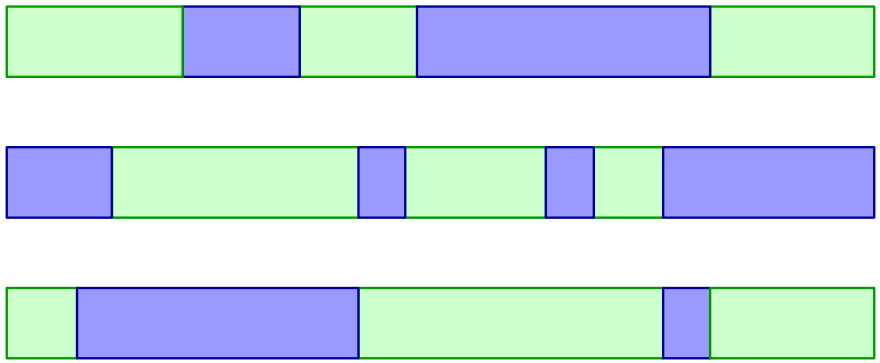
- **ocupado**: espaço já reservado para um nó
- **livre**: espaço disponível para criar novos nós; ou

Lista de blocos livres

É uma lista ligada contendo os blocos **livres** do heap.

Problema: Não podemos alocar memória

Uma ideia: lista dos espaços livres



Um **bloco** é um espaço contíguo de memória:

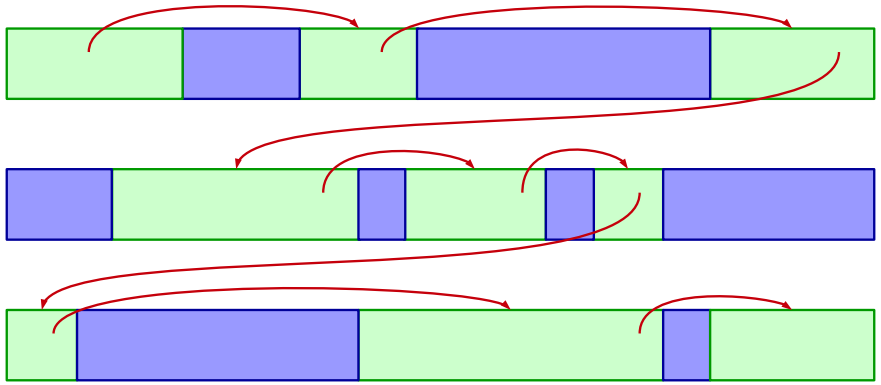
- **ocupado**: espaço já reservado para um nó
- **livre**: espaço disponível para criar novos nós; ou

Lista de blocos livres

É uma lista ligada contendo os blocos **livres** do heap.

Problema: Não podemos alocar memória; onde guardar os ponteiros?

Uma ideia: lista dos espaços livres



Um **bloco** é um espaço contíguo de memória:

- **ocupado**: espaço já reservado para um nó
- **livre**: espaço disponível para criar novos nós; ou

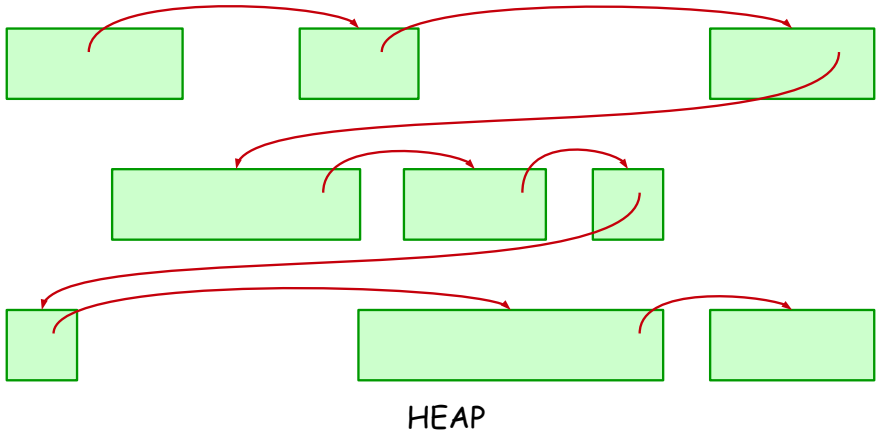
Lista de blocos livres

É uma lista ligada contendo os blocos **livres** do heap.

Problema: Não podemos alocar memória; onde guardar os ponteiros?

$\Rightarrow$ Nos próprios blocos!

Uma ideia: lista dos espaços livres



Um **bloco** é um espaço contíguo de memória:

- **ocupado**: espaço já reservado para um nó
- **livre**: espaço disponível para criar novos nós; ou

Lista de blocos livres

É uma lista ligada contendo os blocos **livres** do heap.

Problema: Não podemos alocar memória; onde guardar os ponteiros?

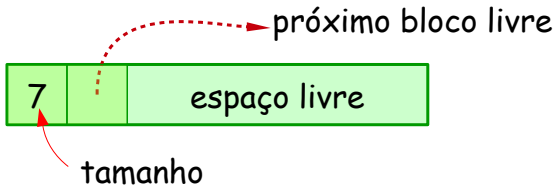
$\Rightarrow$ Nos próprios blocos!

Informações adicionais

Guardamos no início de cada bloco:

Informações adicionais

Guardamos no início de cada bloco:

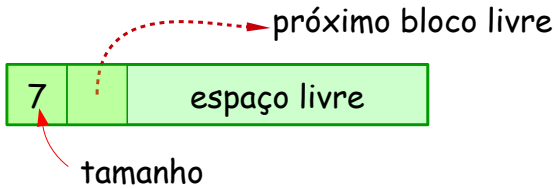


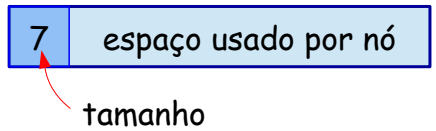
Bloco livre

- tamanho do bloco
- ponteiro para próximo bloco livre

Informações adicionais

Guardamos no início de cada bloco:





Bloco livre

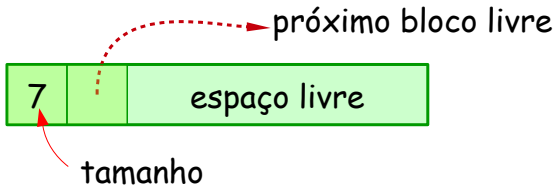
- tamanho do bloco
- ponteiro para próximo bloco livre

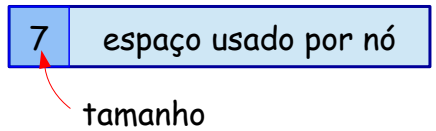
Bloco ocupado:

- tamanho do bloco

Informações adicionais

Guardamos no início de cada bloco:



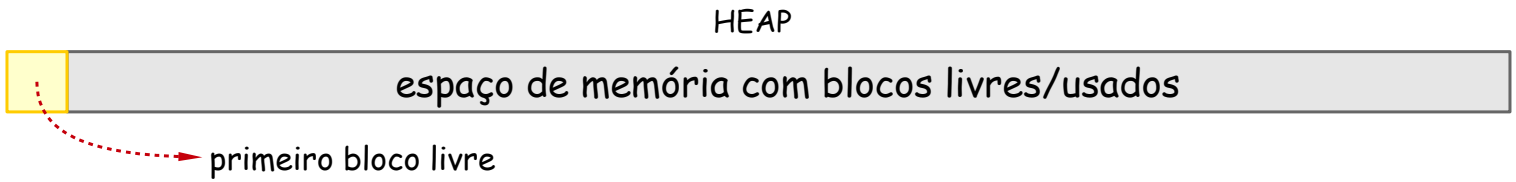


Bloco livre

- tamanho do bloco
- ponteiro para próximo bloco livre

Bloco ocupado:

- tamanho do bloco



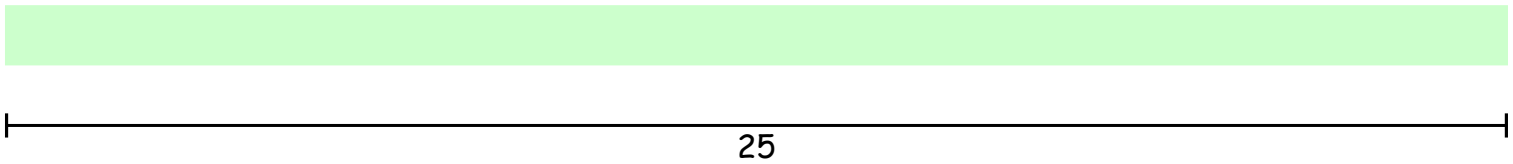
Observação: Também guardamos o ponteiro para o primeiro bloco livre.

Alocando memória

Começamos com um único bloco livre!

Alocando memória

Começamos com um único bloco livre!

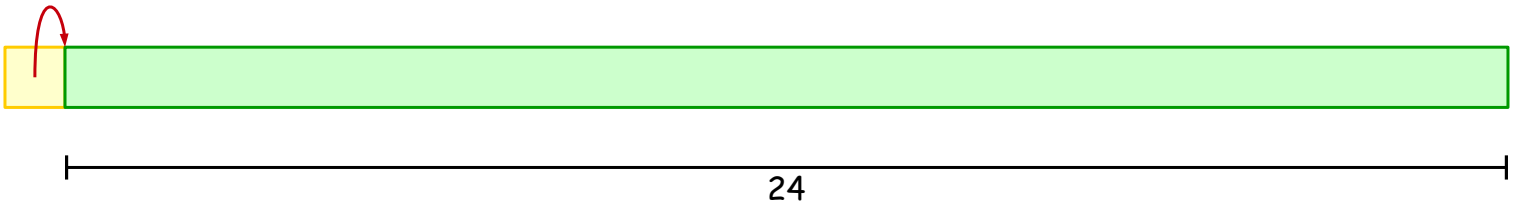


Alocando memória

Começamos com um único bloco livre!

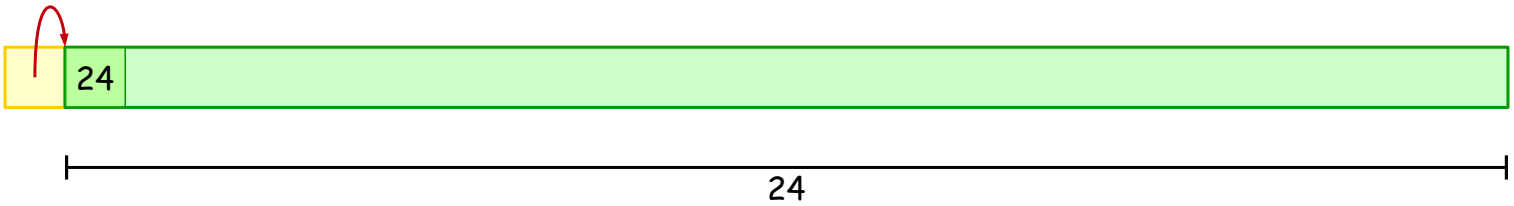
Alocando memória

Começamos com um único bloco livre!



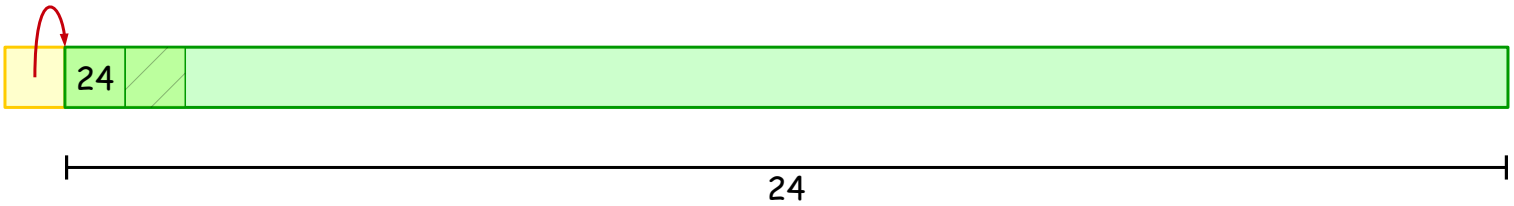
Alocando memória

Começamos com um único bloco livre!



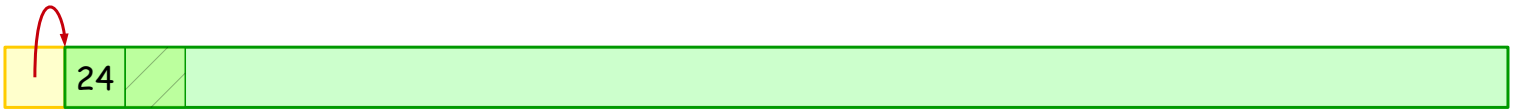
Alocando memória

Começamos com um único bloco livre!



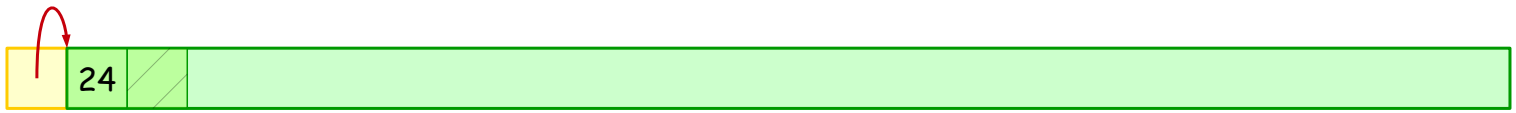
Alocando memória

Começamos com um único bloco livre!



Alocando memória

Começamos com um único bloco livre!

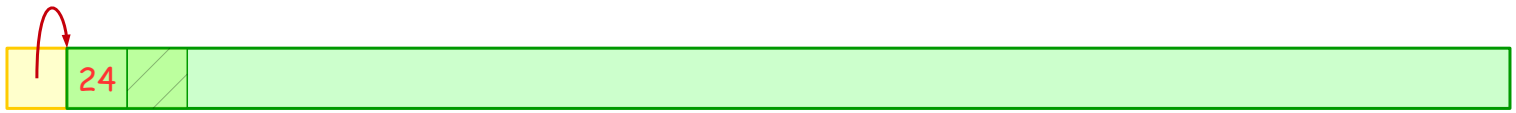


Alocar 4 bytes

Alocando pela primeira vez

Alocando memória

Começamos com um único bloco livre!



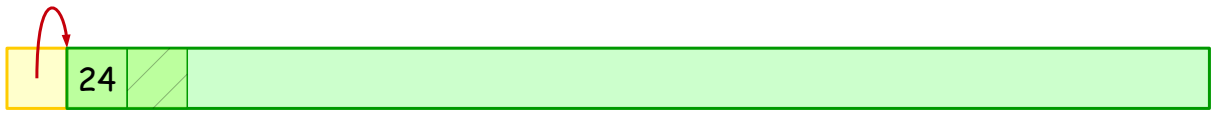
Alocar 4 bytes

Alocando pela primeira vez

1. Procuramos um bloco livre

Alocando memória

Começamos com um único bloco livre!



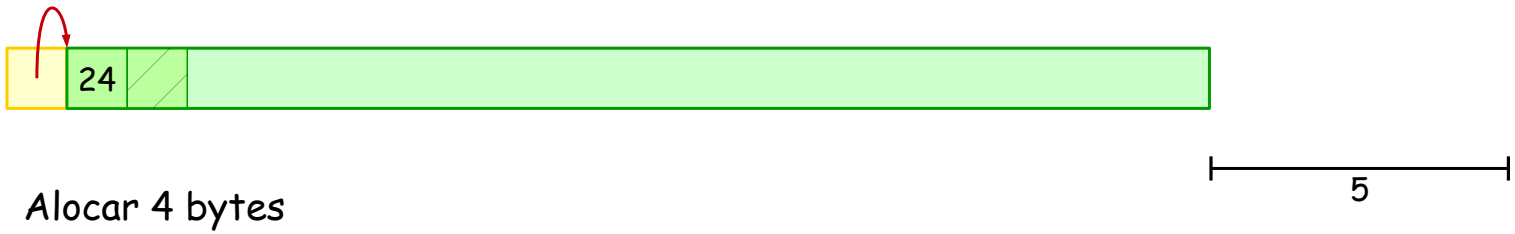
Alocar 4 bytes

Alocando pela primeira vez

1. Procuramos um bloco livre
2. Diminuímos tamanho

Alocando memória

Começamos com um único bloco livre!

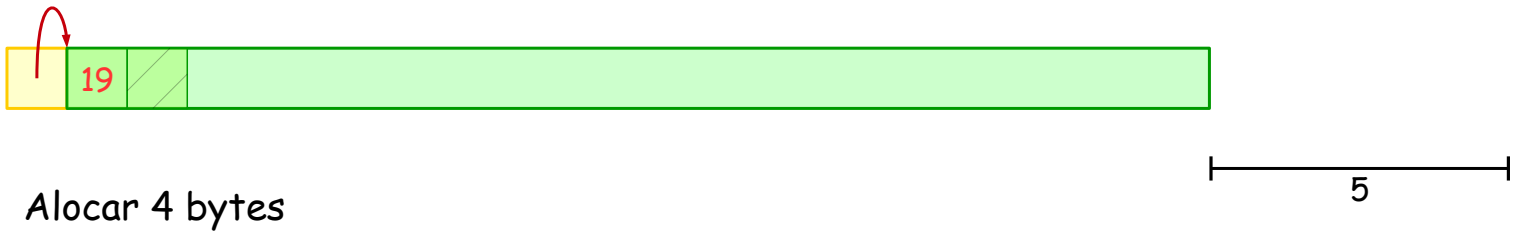


Alocando pela primeira vez

1. Procuramos um bloco livre
2. Diminuímos tamanho

Alocando memória

Começamos com um único bloco livre!

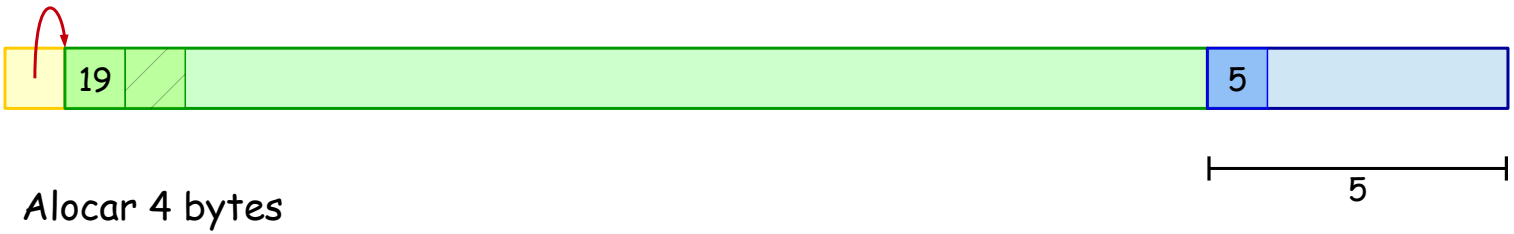


Alocando pela primeira vez

1. Procuramos um bloco livre
2. Diminuímos tamanho

Alocando memória

Começamos com um único bloco livre!

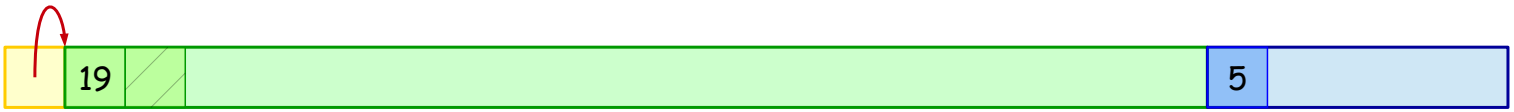


Alocando pela primeira vez

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco (informação adicional + espaço)

Alocando memória

Começamos com um único bloco livre!

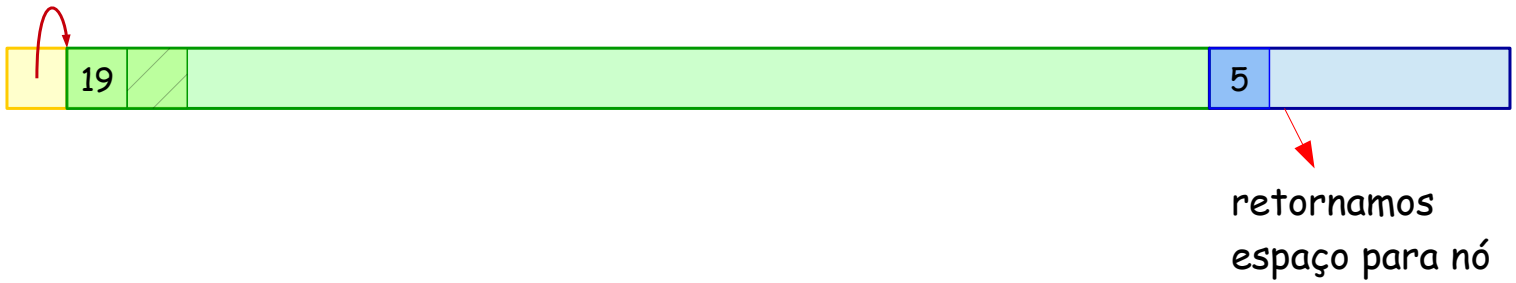


Alocando pela primeira vez

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco (informação adicional + espaço)

Alocando memória

Começamos com um único bloco livre!

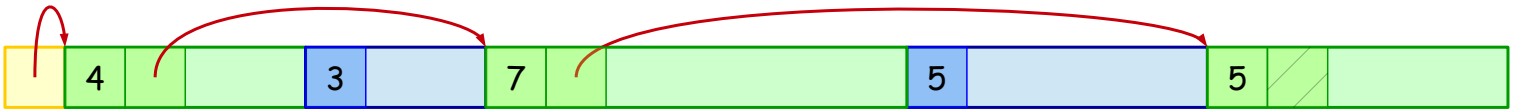


Alocando pela primeira vez

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco (informação adicional + espaço)
4. Retornamos espaço para o nó

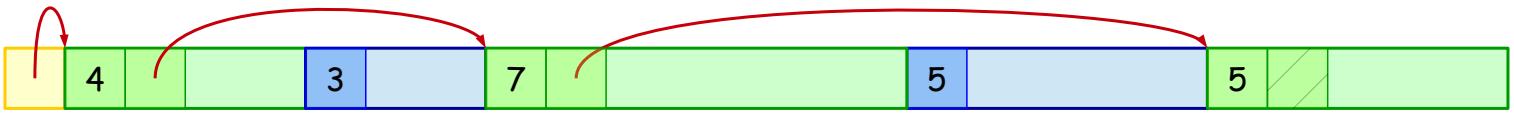
Alocando memória: removendo da lista livre

Um exemplo mais complicado:



Alocando memória: removendo da lista livre

Um exemplo mais complicado:

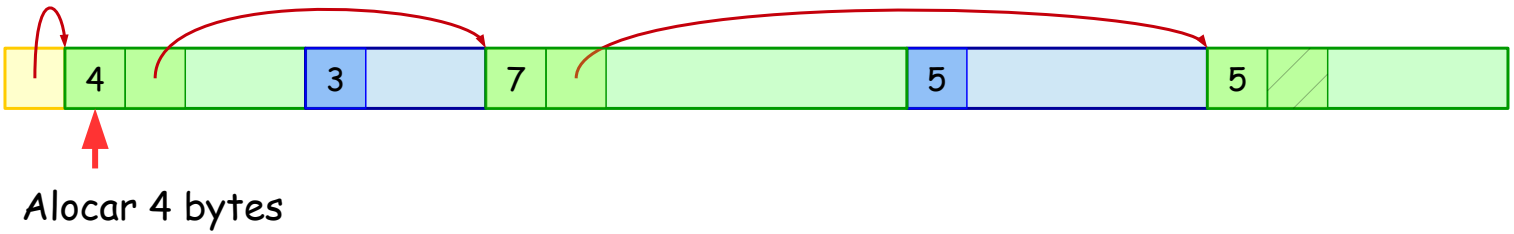


Alocar 4 bytes

Alocando em bloco justo

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

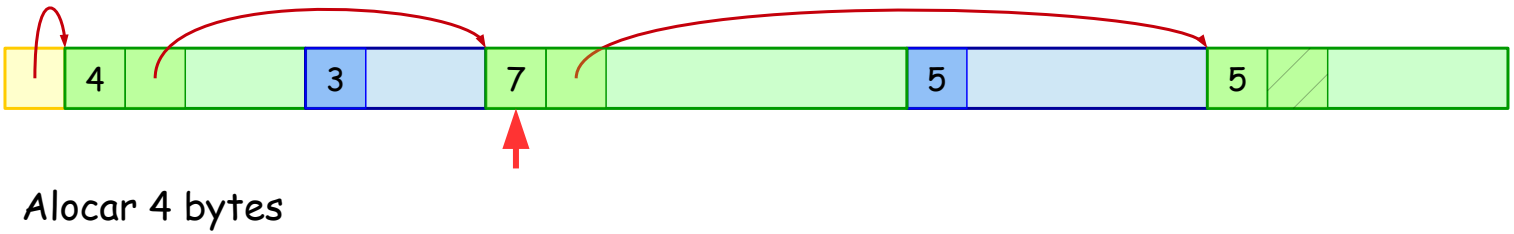


Alocando em bloco justo

1. Procuramos um bloco livre

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

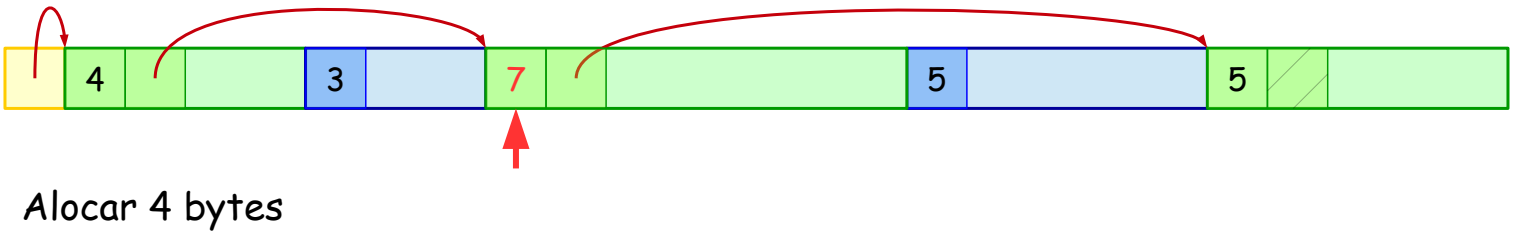


Alocando em bloco justo

1. Procuramos um bloco livre

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

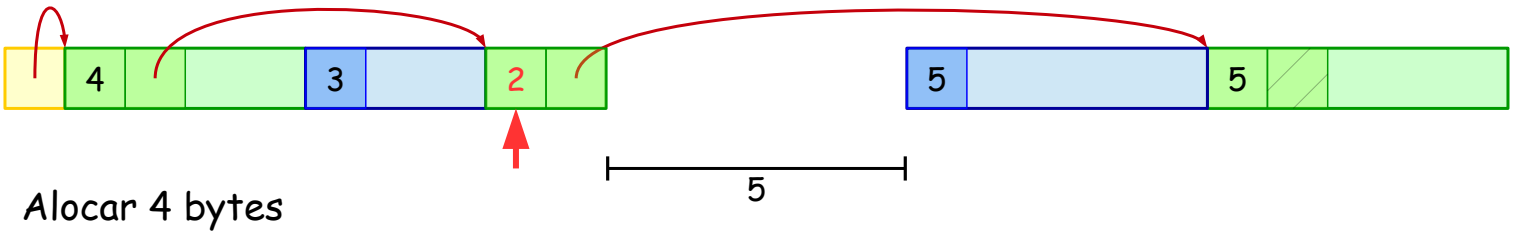


Alocando em bloco justo

1. Procuramos um bloco livre

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

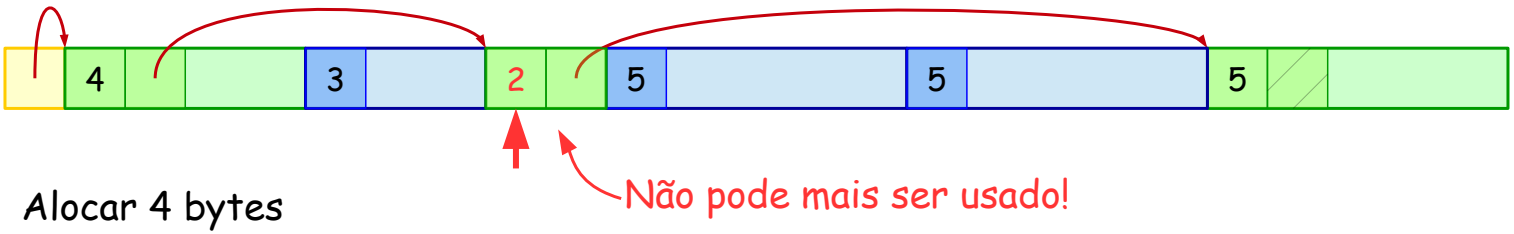


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

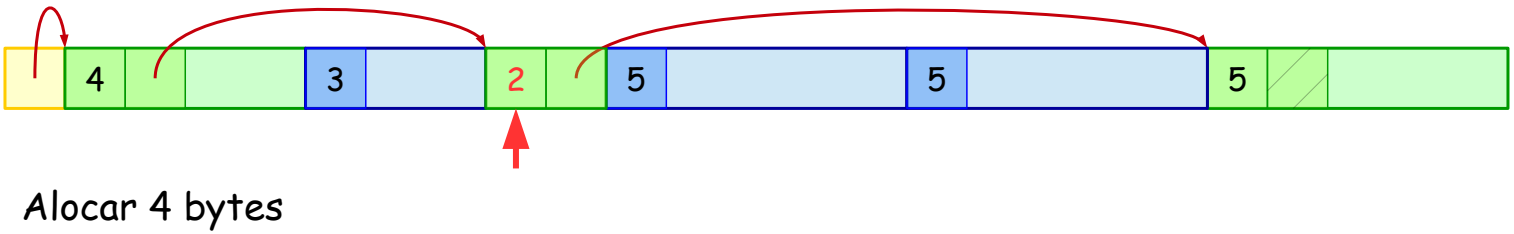


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

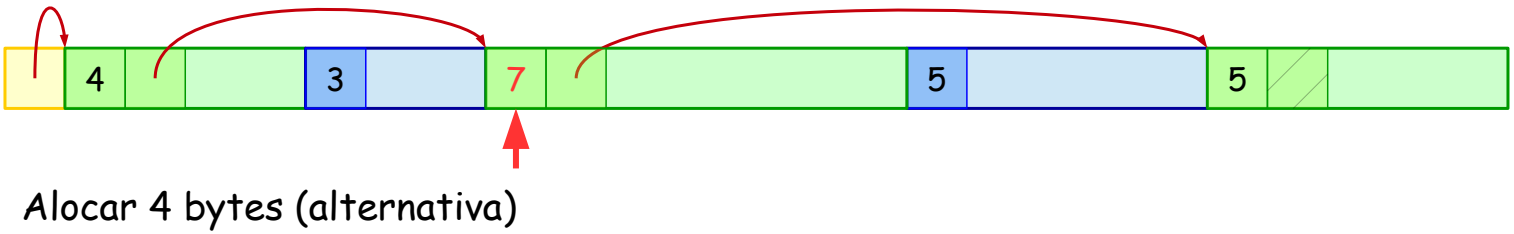


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

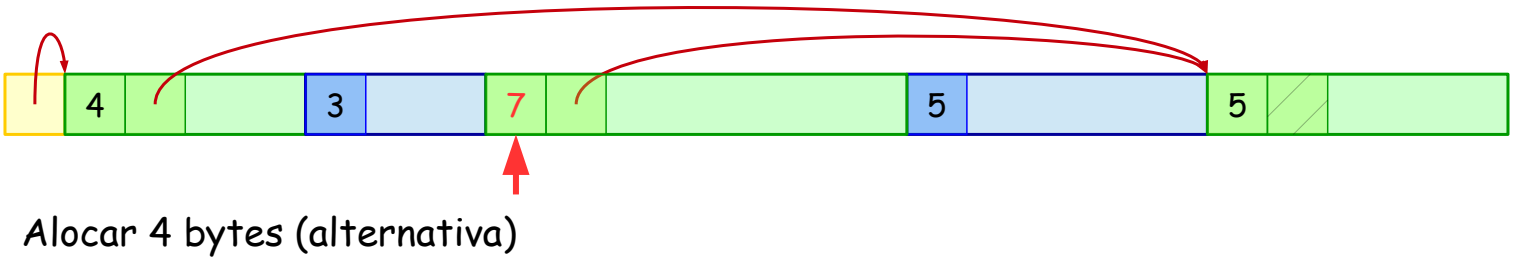


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco
4. Removemos bloco livre da lista quando for muito pequeno

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

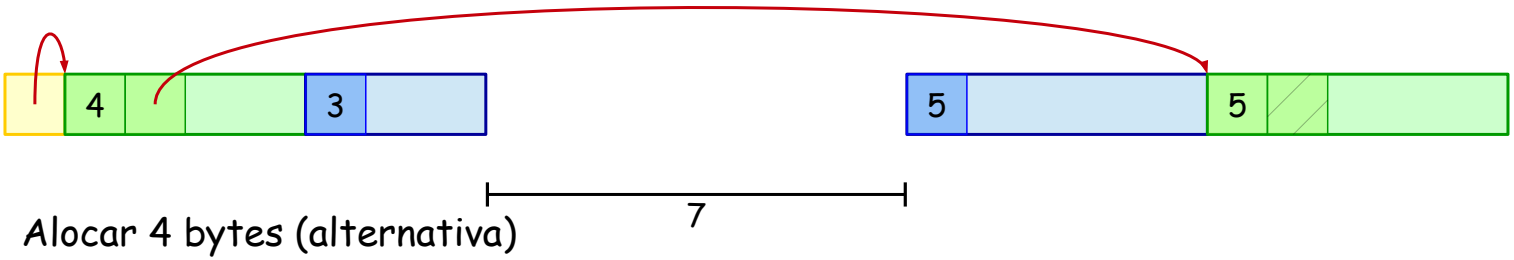


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco
4. Removemos bloco livre da lista quando for muito pequeno

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

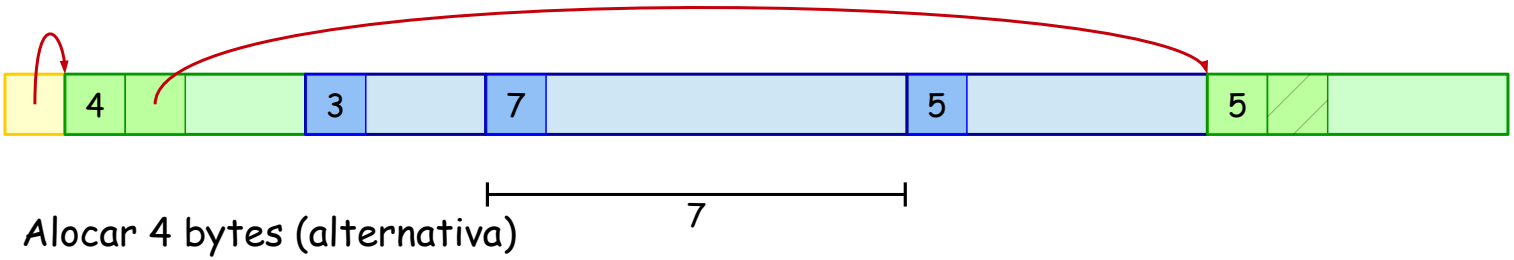


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco
4. Removemos bloco livre da lista quando for muito pequeno

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

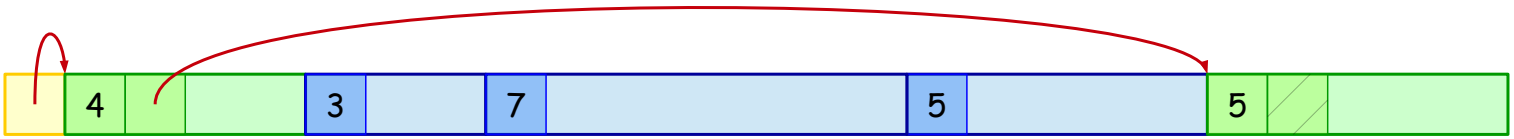


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco
4. Removemos bloco livre da lista quando for muito pequeno

Alocando memória: removendo da lista livre

Um exemplo mais complicado:

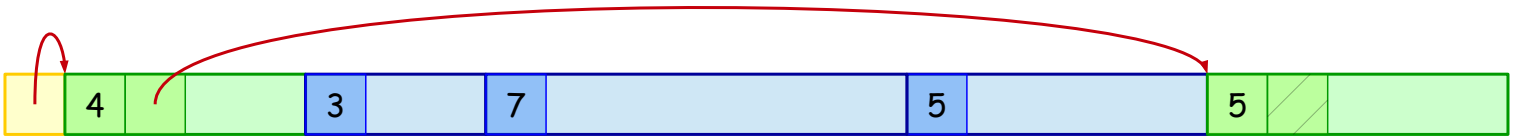


Alocando em bloco justo

1. Procuramos um bloco livre
2. Diminuímos tamanho
3. Alocamos novo bloco
4. Removemos bloco livre da lista quando for muito pequeno

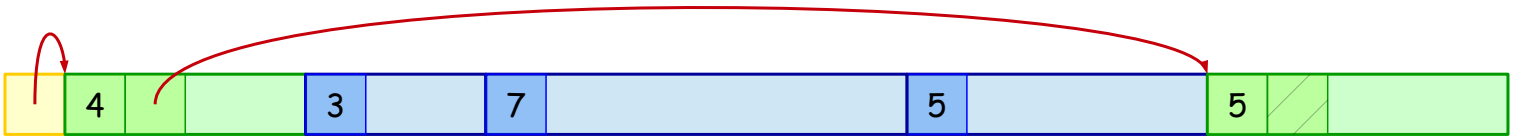
Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:



Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:

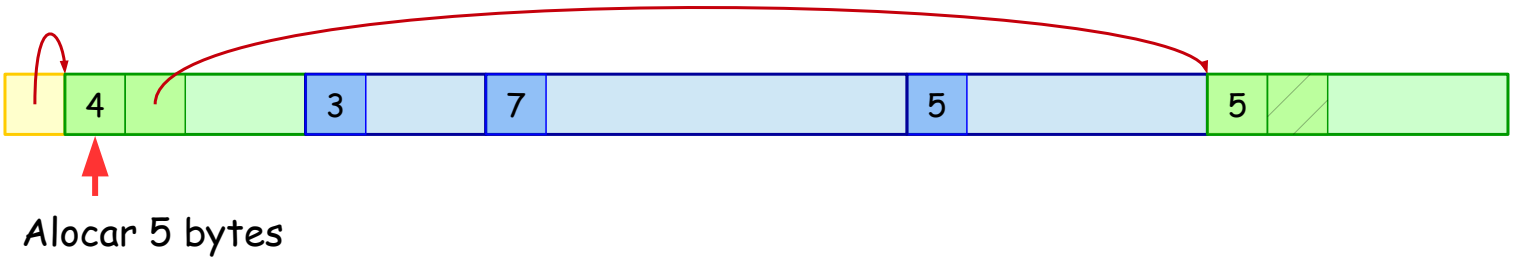


Alocar 5 bytes

Tentando alocar

Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:

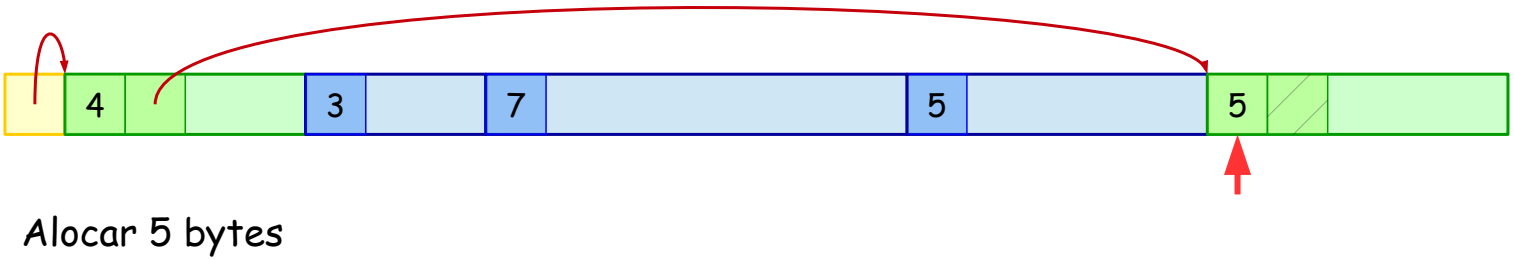


Tentando alocar

1. Procuramos um bloco livre

Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:

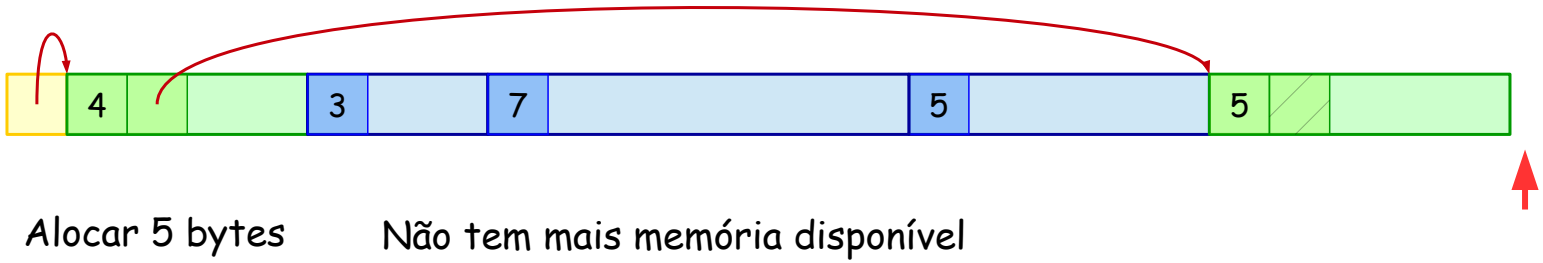


Tentando alocar

1. Procuramos um bloco livre

Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:

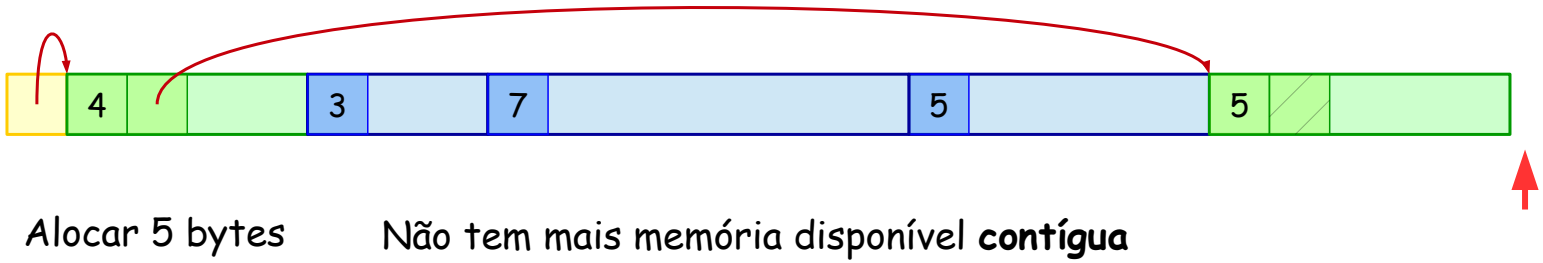


Tentando alocar

1. Procuramos um bloco livre
2. Não encontramos bloco

Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:

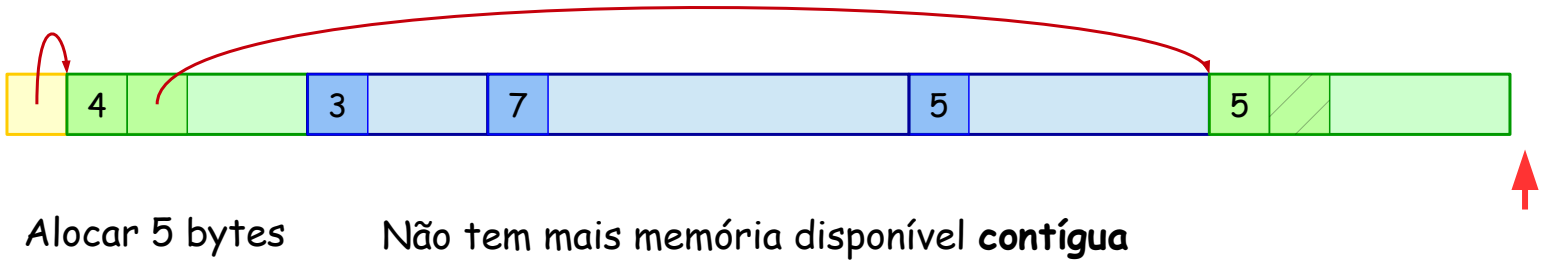


Tentando alocar

1. Procuramos um bloco livre
2. Não encontramos bloco

Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:



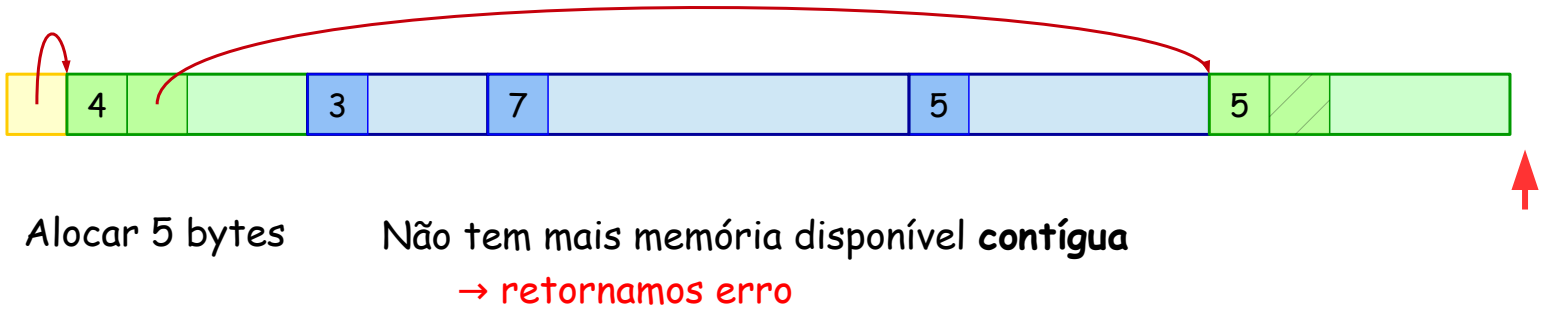
Tentando alocar

1. Procuramos um bloco livre
2. Não encontramos bloco

Observação: a memória está fragmentada!

Alocando memória: faltando memória

Pode ser que nenhum bloco livre é grande o suficiente:



Tentando alocar

1. Procuramos um bloco livre
2. Não encontramos bloco
3. Retornamos algum código de erro (e.g., retornamos `NULL`)

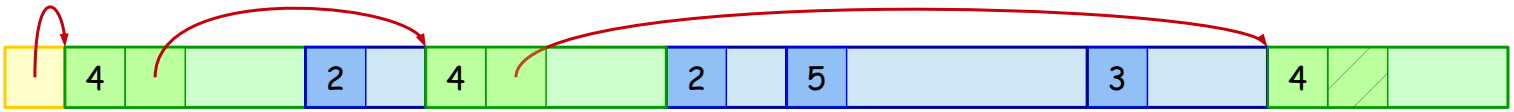
Observação: a memória está fragmentada!

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

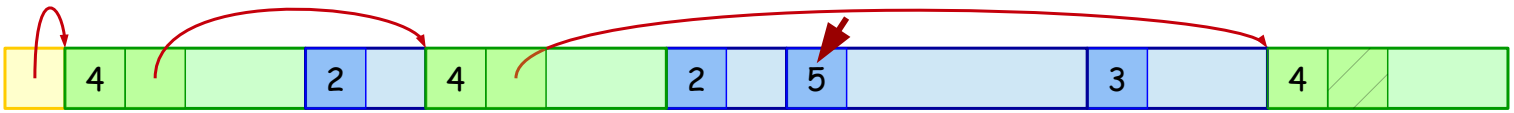
Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:



Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

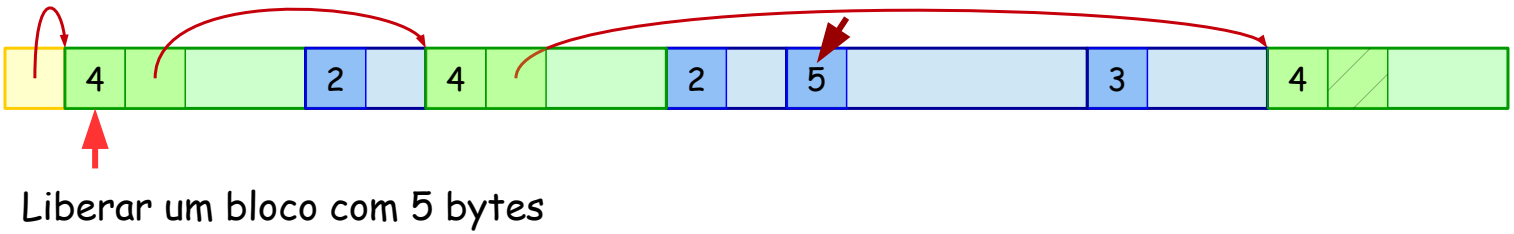


Liberar um bloco com 5 bytes

Liberando

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

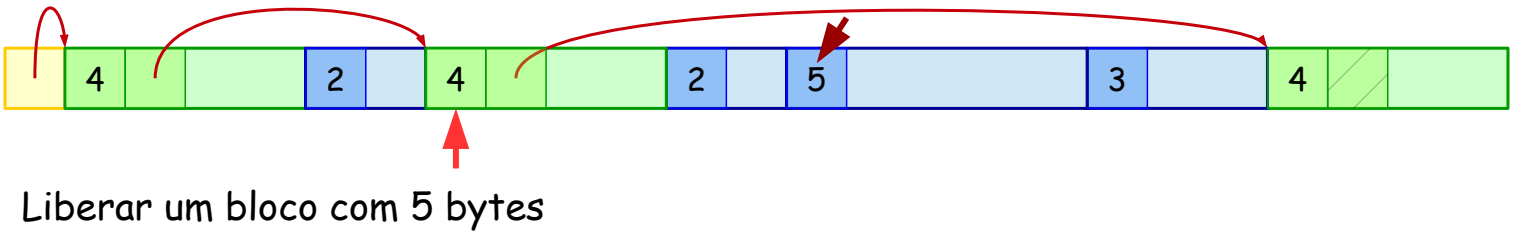


Liberando

1. Procuramos o último bloco livre anterior

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

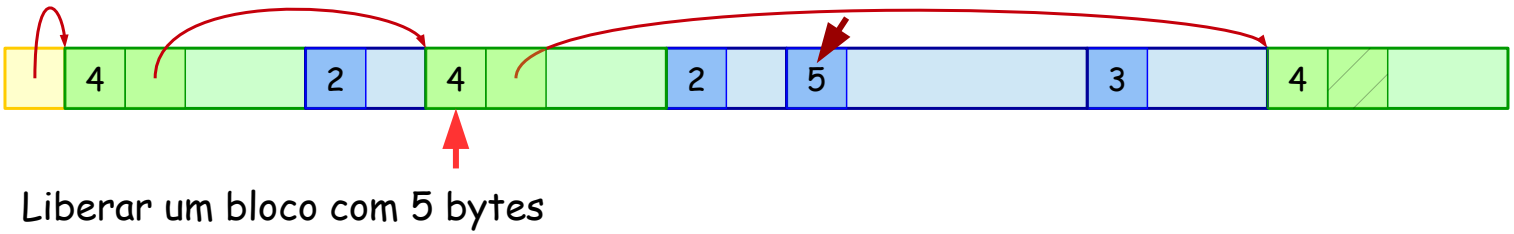


Liberando

1. Procuramos o último bloco livre anterior

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

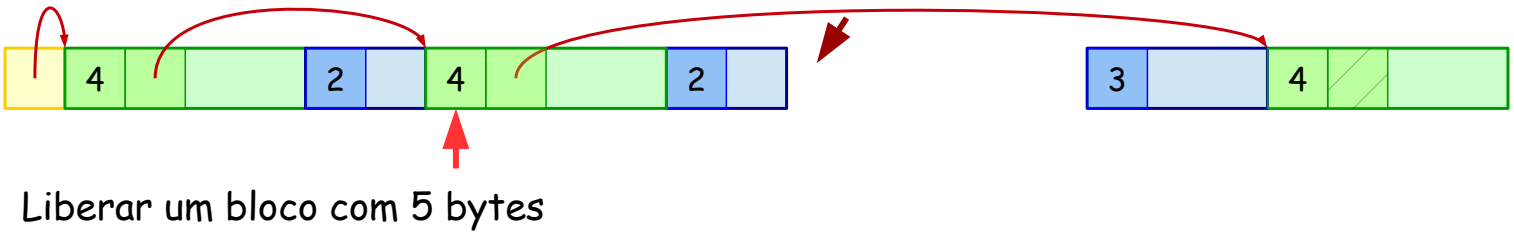


Liberando

1. Procuramos o último bloco livre anterior
2. Criamo um novo bloco livre

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

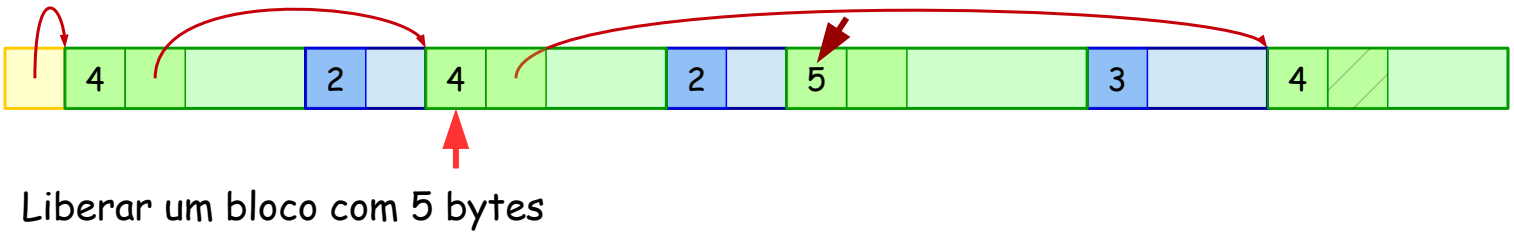


Liberando

1. Procuramos o último bloco livre anterior
2. Criamo um novo bloco livre

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

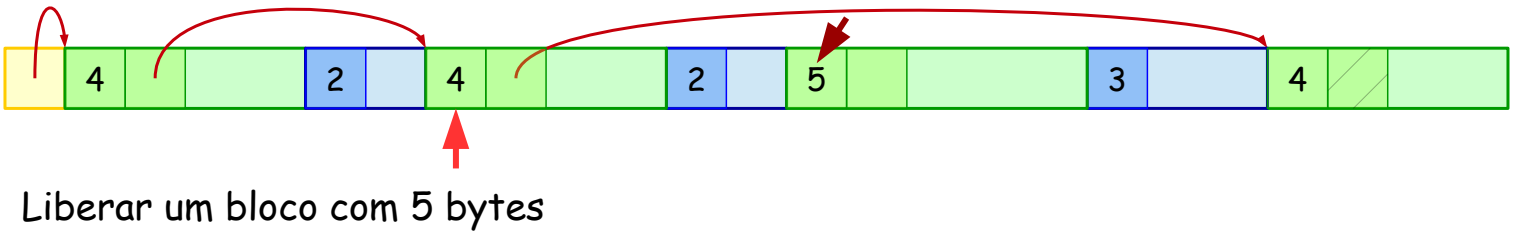


Liberando

1. Procuramos o último bloco livre anterior
2. Criamo um novo bloco livre

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

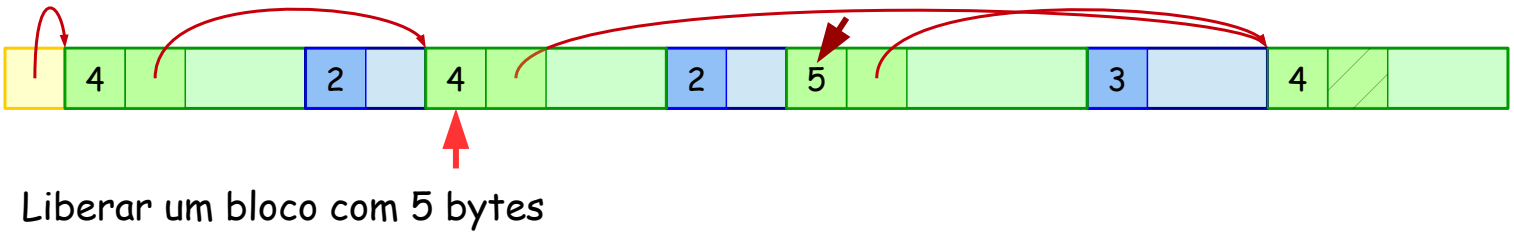


Liberando

1. Procuramos o último bloco livre anterior
2. Criamo um novo bloco livre
3. Inserimos na lista de blocos livres

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

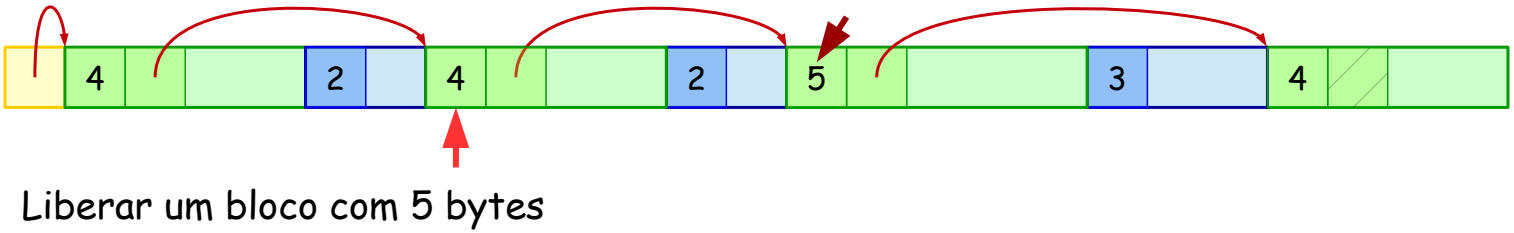


Liberando

1. Procuramos o último bloco livre anterior
2. Criamo um novo bloco livre
3. Inserimos na lista de blocos livres

Liberando memória

Devolvendo um espaço de memória que não vai mais ser usado:

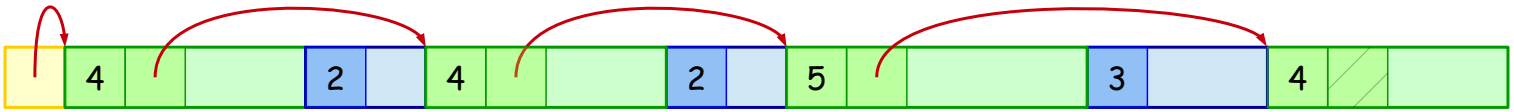


Liberando

1. Procuramos o último bloco livre anterior
2. Criamo um novo bloco livre
3. Inserimos na lista de blocos livres

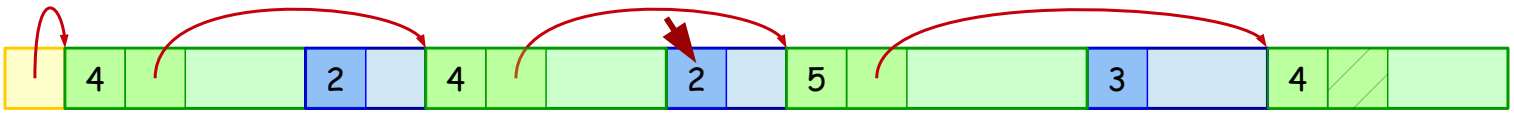
Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:



Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

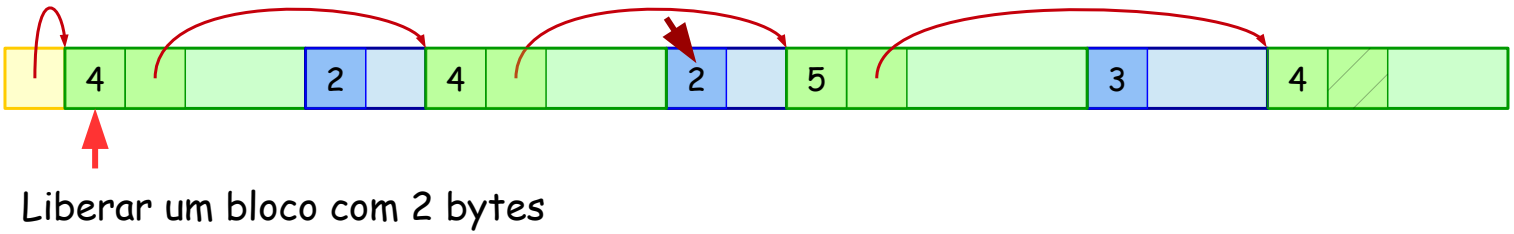


Liberar um bloco com 2 bytes

Liberando e juntando

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

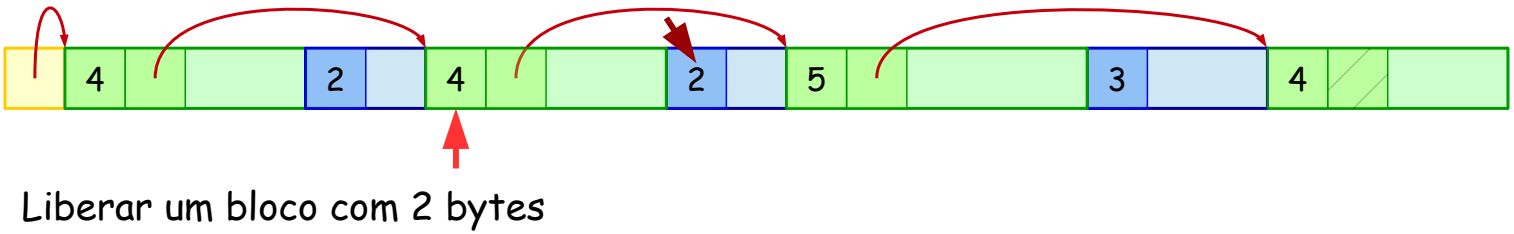


Liberando e juntando

1. Procuramos o último bloco livre anterior

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

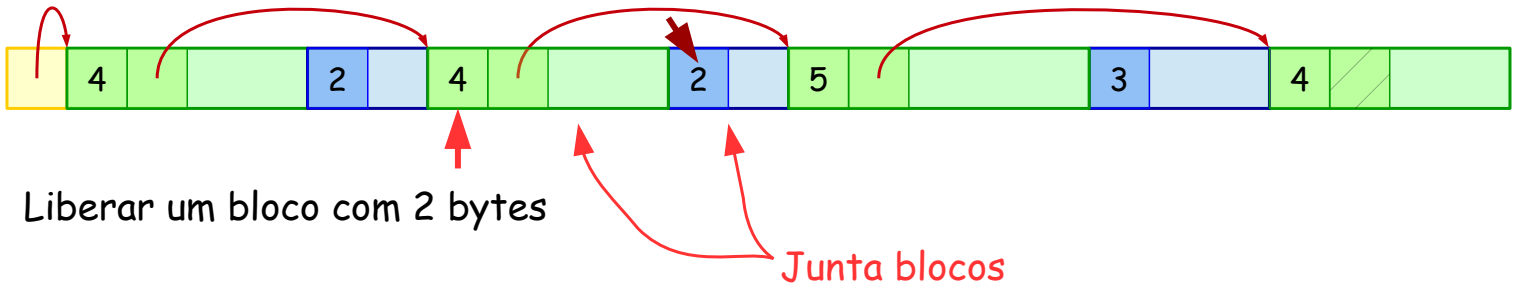


Liberando e juntando

1. Procuramos o último bloco livre anterior

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

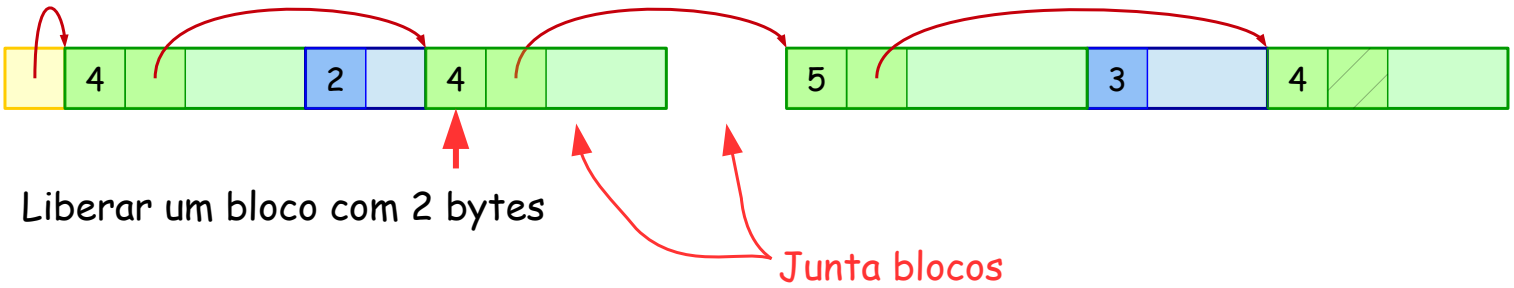


Liberando e juntando

1. Procuramos o último bloco livre anterior
2. Se último for adjacente:

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

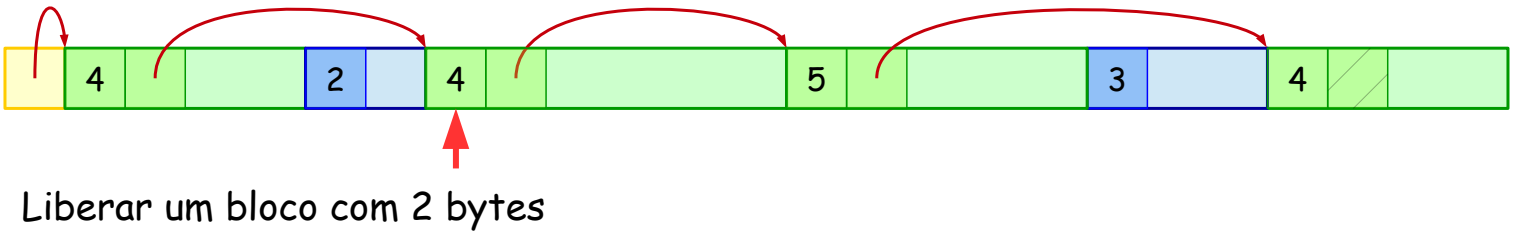


Liberando e juntando

1. Procuramos o último bloco livre anterior
2. Se último for adjacente:
   - aumentamos tamanho do último

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

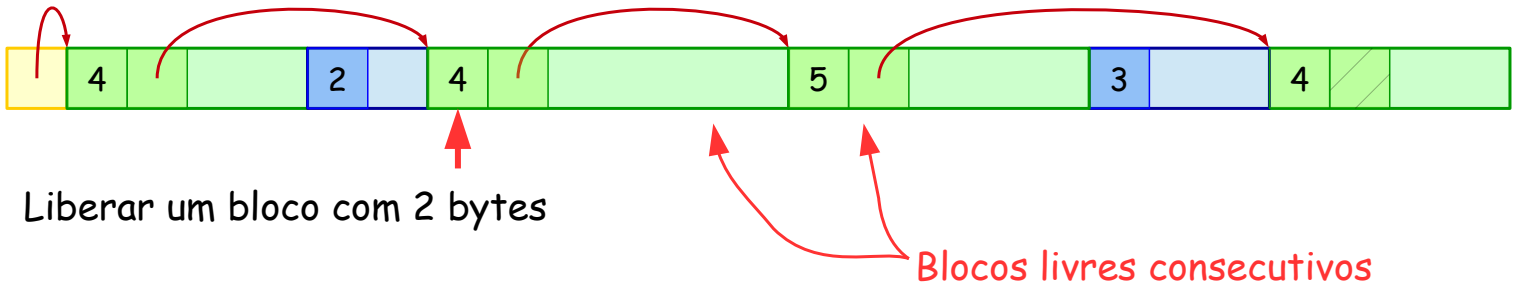


Liberando e juntando

1. Procuramos o último bloco livre anterior
2. Se último for adjacente:
   - aumentamos tamanho do último

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

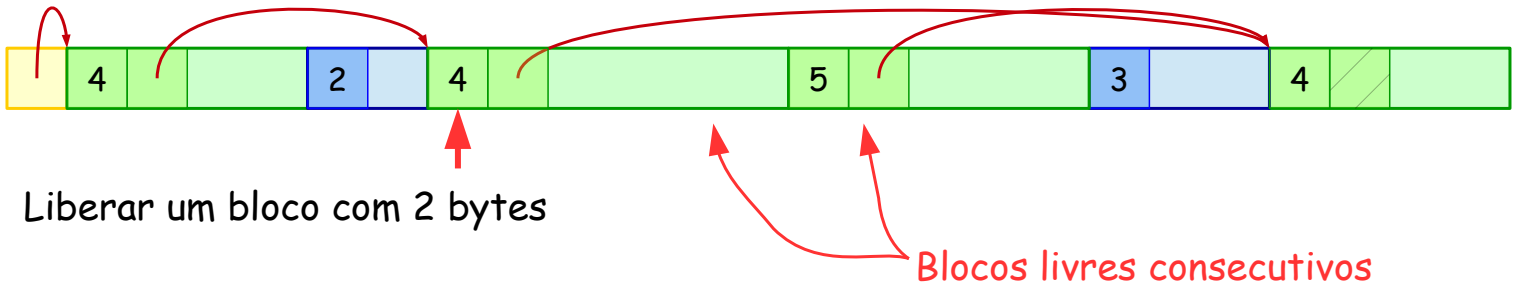


Liberando e juntando

1. Procuramos o último bloco livre anterior
2. Se último for adjacente:
   - aumentamos tamanho do último
3. Se próximo for adjacente:

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

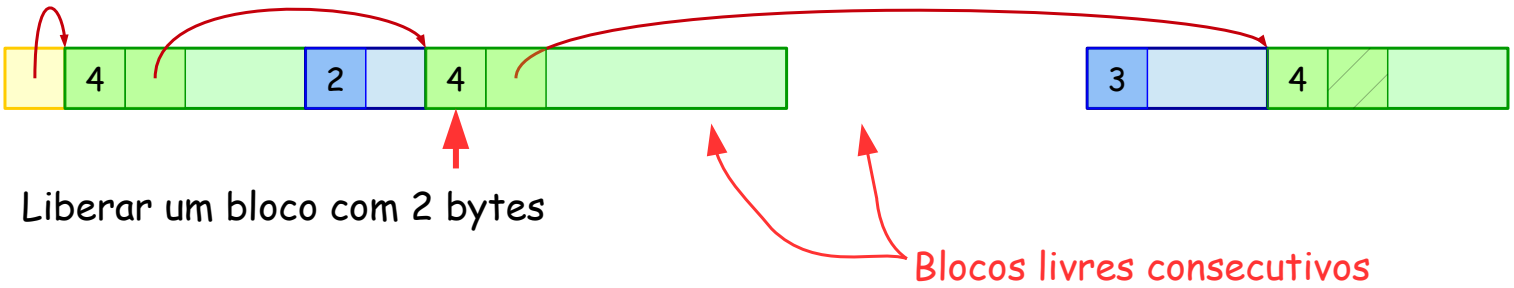


Liberando e juntando

1. Procuramos o último bloco livre anterior
2. Se último for adjacente:
   - aumentamos tamanho do último
3. Se próximo for adjacente:
   - removemos próximo da lista de blocos livres

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:

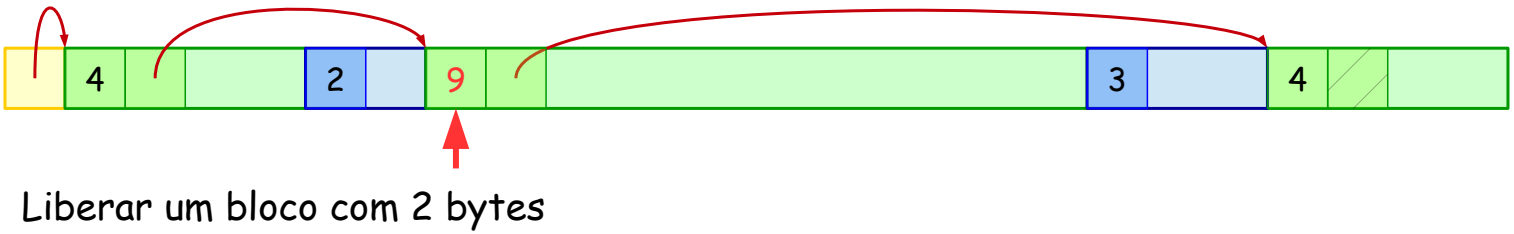


Liberando e juntando

1. Procuramos o último bloco livre anterior
2. Se último for adjacente:
   - aumentamos tamanho do último
3. Se próximo for adjacente:
   - removemos próximo da lista de blocos livres

Liberando memória: combinando blocos livres

Pode ser que haja blocos livres adjacentes:



Liberando e juntando

1. Procuramos o último bloco livre anterior
2. Se último for adjacente:
   - aumentamos tamanho do último
3. Se próximo for adjacente:
   - removemos próximo da lista de blocos livres
   - aumentamos tamanho do bloco corrente

Analisando

- Tempo: podemos percorrer vários blocos livres (toda a lista no pior caso)

Analisando

- Tempo: podemos percorrer vários blocos livres (toda a lista no pior caso)
- Fragmentação: há memória disponível, mas não está contígua

Analisando

- Tempo: podemos percorrer vários blocos livres (toda a lista no pior caso)
- Fragmentação: há memória disponível, mas não está contígua
- Limites: alterar área não alocada pode modificar outros nós ou informações adicionais!

Analisando

- Tempo: podemos percorrer vários blocos livres (toda a lista no pior caso)
- Fragmentação: há memória disponível, mas não está contígua
- Limites: alterar área não alocada pode modificar outros nós ou informações adicionais!

Implicações e práticas recomendadas

- não alocar blocos de memória desnecessariamente
- evitar excesso de nós "pequenos"
- usar boas estimativas para tamanho dos dados (sempre que possível)
- liberar toda a memória alocada
- certificar-se de usar somente o espaço alocado

Analisando

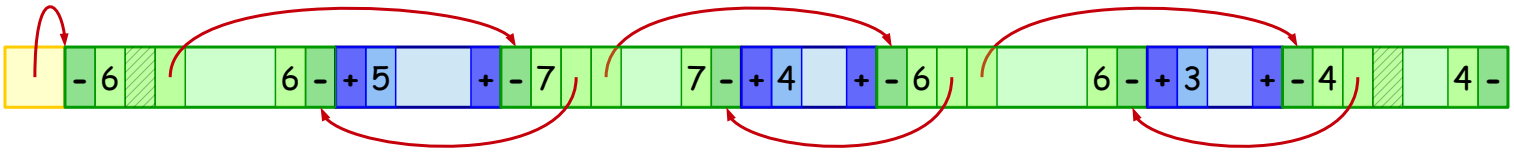
- Tempo: podemos percorrer vários blocos livres (toda a lista no pior caso)
- Fragmentação: há memória disponível, mas não está contígua
- Limites: alterar área não alocada pode modificar outros nós ou informações adicionais!

Implicações e práticas recomendadas

- não alocar blocos de memória desnecessariamente
- evitar excesso de nós "pequenos"
- usar boas estimativas para tamanho dos dados (sempre que possível)
- liberar toda a memória alocada
- certificar-se de usar somente o espaço alocado

Uma pergunta: como melhorar o tempo de alocação/liberação?

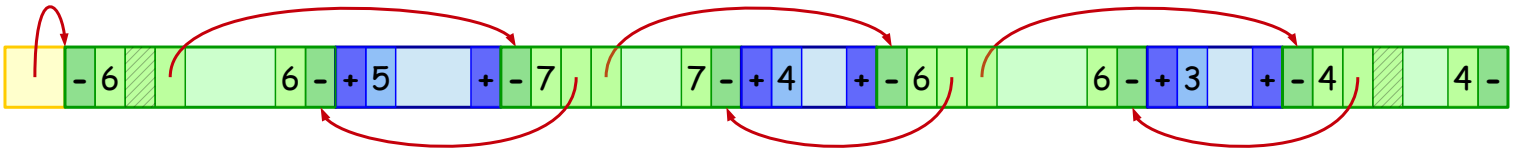
Melhorando um pouco



Marcação de fronteiras

- usamos uma lista duplamente encadeada
- usamos indicadores no início e final de blocos:
- guardamos o tamanho do bloco livre também no final

Melhorando um pouco



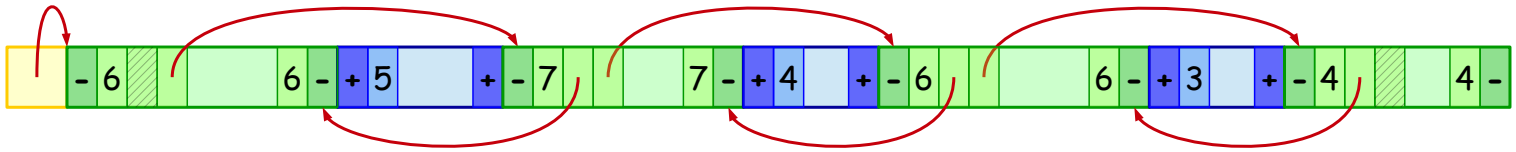
Marcação de fronteiras

- usamos uma lista duplamente encadeada
- usamos indicadores no início e final de blocos:
- guardamos o tamanho do bloco livre também no final

Liberação melhorada

- Se houver bloco livre adjacente: junta
- Se não houver bloco livre adjacente: adiciona no início da lista

Melhorando um pouco



Marcação de fronteiras

- usamos uma lista duplamente encadeada
- usamos indicadores no início e final de blocos:
- guardamos o tamanho do bloco livre também no final

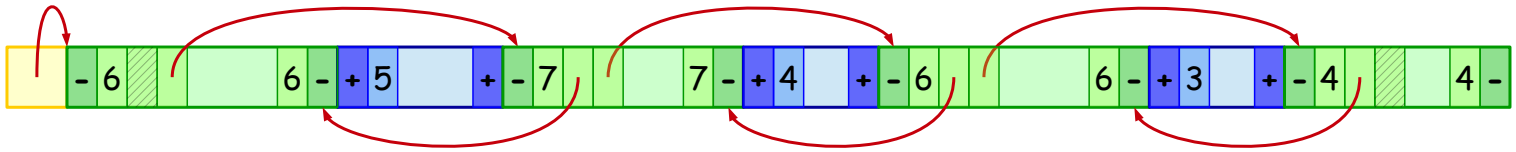
Liberação melhorada

- Se houver bloco livre adjacente: junta
- Se não houver bloco livre adjacente: adiciona no início da lista

Perguntas:

- por que é fácil verificar se há bloco livre adjacente?

Melhorando um pouco



Marcação de fronteiras

- usamos uma lista duplamente encadeada
- usamos indicadores no início e final de blocos:
- guardamos o tamanho do bloco livre também no final

Liberação melhorada

- Se houver bloco livre adjacente: junta
- Se não houver bloco livre adjacente: adiciona no início da lista

Perguntas:

- por que é fácil verificar se há bloco livre adjacente?
- a lista de adjacência será mantida em ordem de "endereço"?

Formas de gerenciamento de memória

Dependendo da linguagem, o gerenciamento pode ser:

- **Explícito:**
  - o programador aloca memória
  - o programador é responsável por liberar a memória
  - Exemplos: C, C++, Pascal

Formas de gerenciamento de memória

Dependendo da linguagem, o gerenciamento pode ser:

- **Explícito:**
  - o programador aloca memória
  - o programador é responsável por liberar a memória
  - Exemplos: C, C++, Pascal

- **Implícito:**
  - o programador "cria" estruturas (de maneira restrita)
  - a linguagem possui mecanismos para liber memória
  - Exemplos: Java, Lisp, Matlab

Liberando memória automaticamente

Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

Liberando memória automaticamente

Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

Pergunta: Como é possível saber que um nó não será mais utilizado?

Liberando memória automaticamente

Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

Pergunta: Como é possível saber que um nó não será mais utilizado?
Resposta: Não é possível!

Liberando memória automaticamente

Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

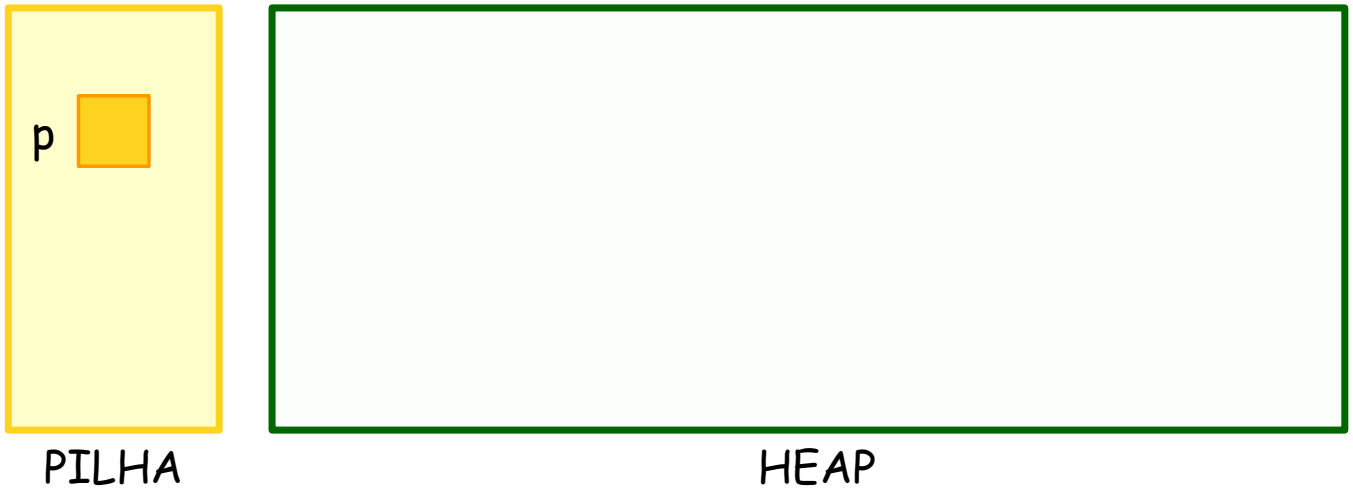
Pergunta: Como é possível saber que um nó não será mais utilizado?
Resposta: Não é possível! Mas podemos deduzir quais são acessíveis.

Liberando memória automaticamente

Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

Pergunta: Como é possível saber que um nó não será mais utilizado?
Resposta: Não é possível! Mas podemos deduzir quais são acessíveis.

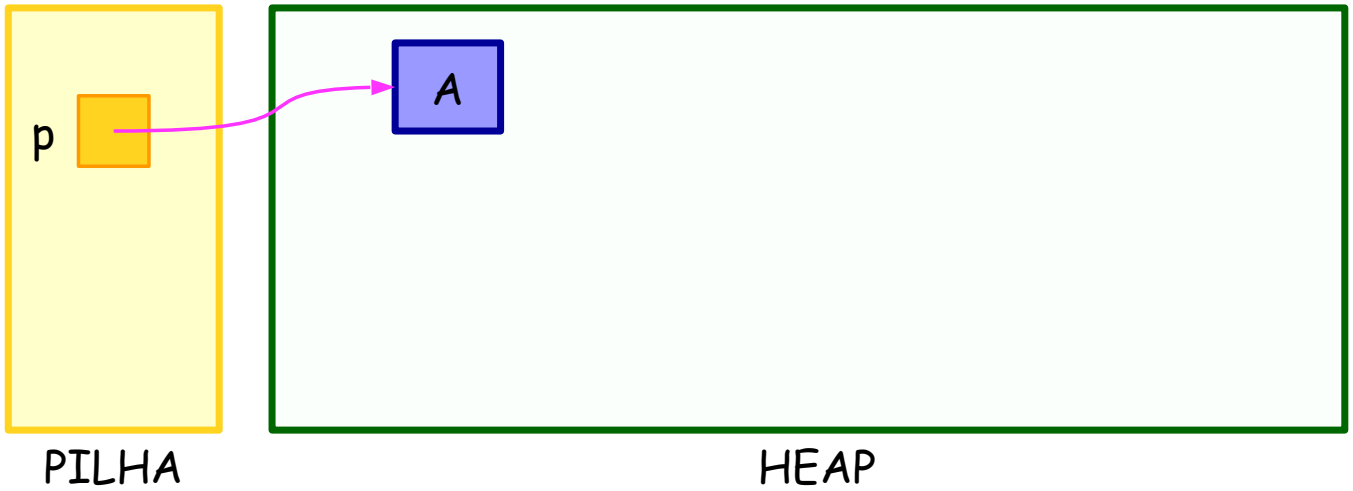


Liberando memória automaticamente

Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

Pergunta: Como é possível saber que um nó não será mais utilizado?
Resposta: Não é possível! Mas podemos deduzir quais são acessíveis.

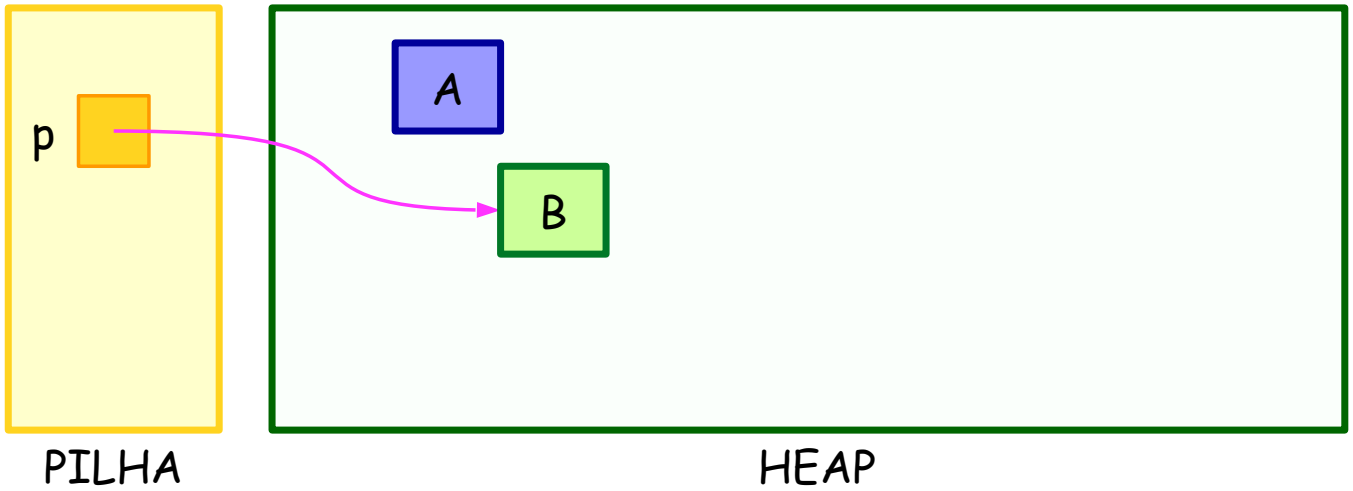


Liberando memória automaticamente

Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

Pergunta: Como é possível saber que um nó não será mais utilizado?
Resposta: Não é possível! Mas podemos deduzir quais são acessíveis.

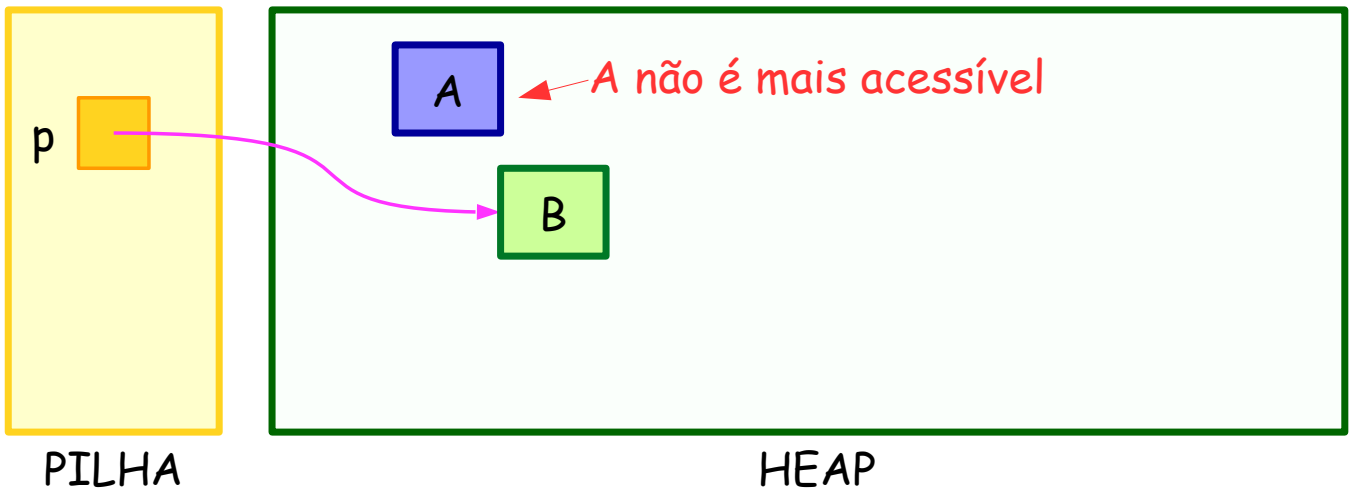


Liberando memória automaticamente

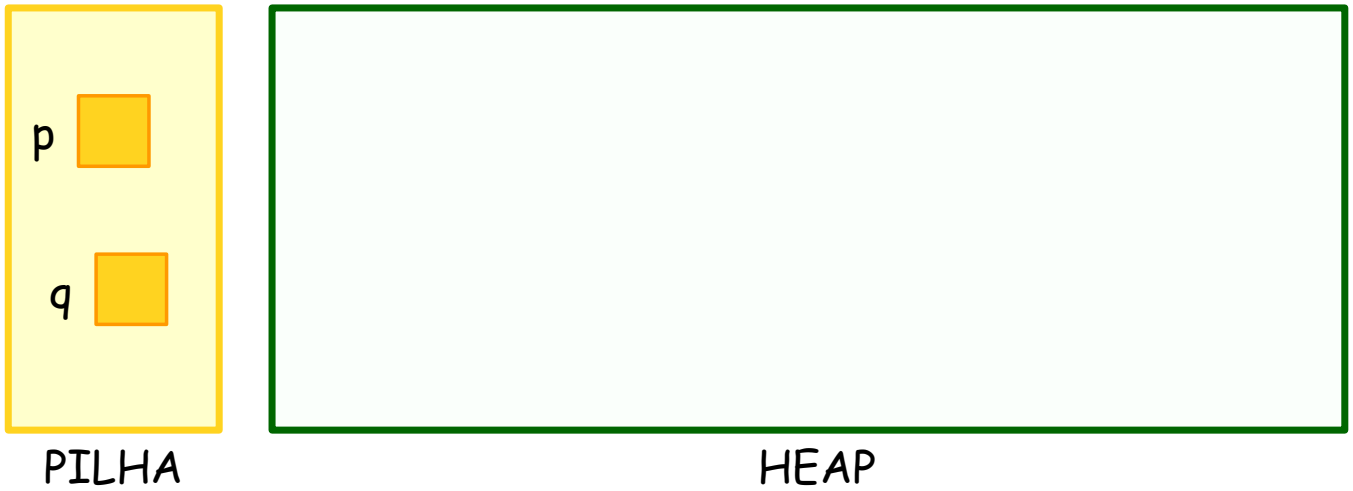
Dificuldade

Só se pode liberar a memória que não será mais utilizada pelo programador!

Pergunta: Como é possível saber que um nó não será mais utilizado?
Resposta: Não é possível! Mas podemos deduzir quais são acessíveis.

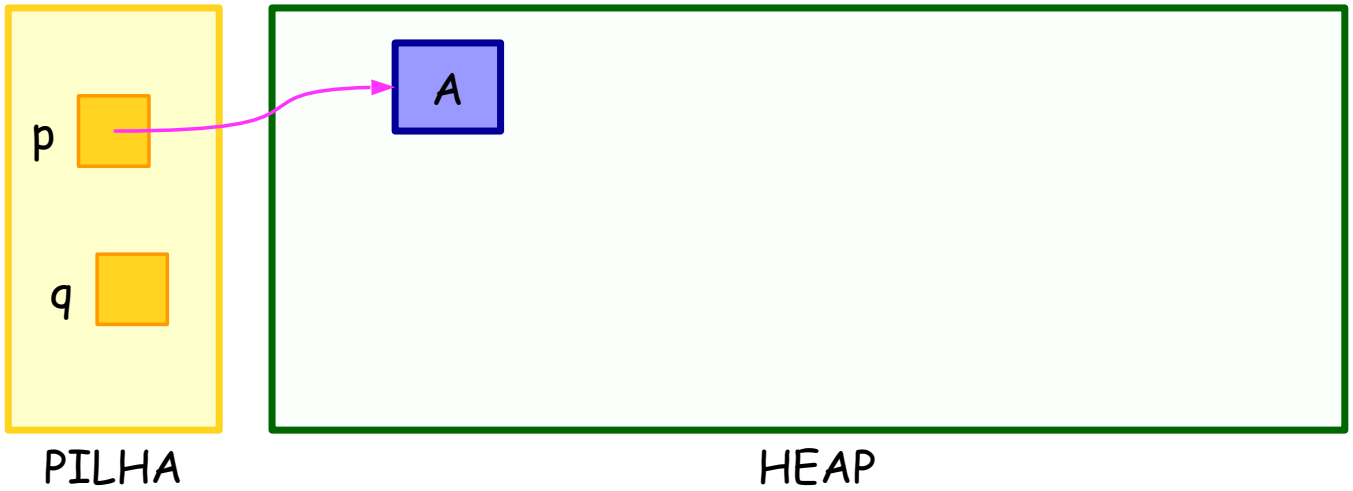


Um exemplo mais complexo



Criando uma lista

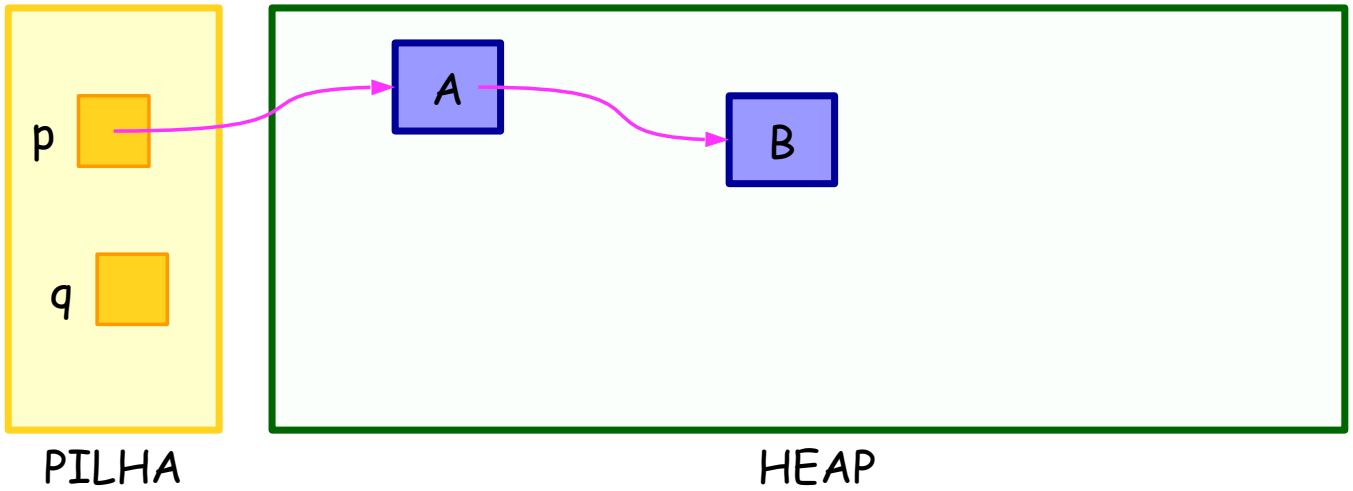
Um exemplo mais complexo



Criando uma lista

1. $p \leftarrow$ cria nó A

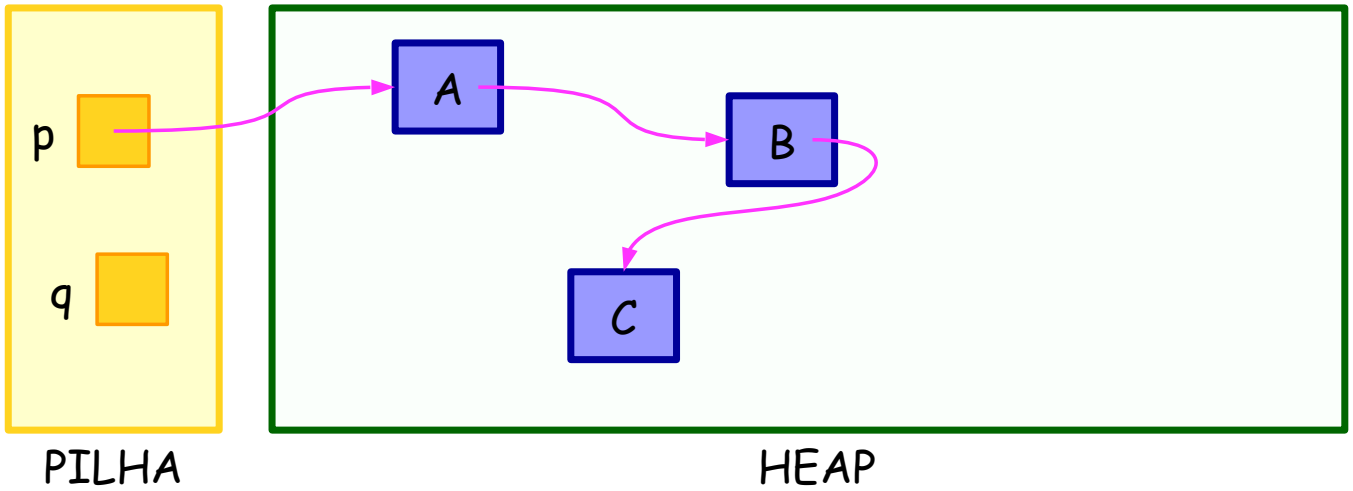
Um exemplo mais complexo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B

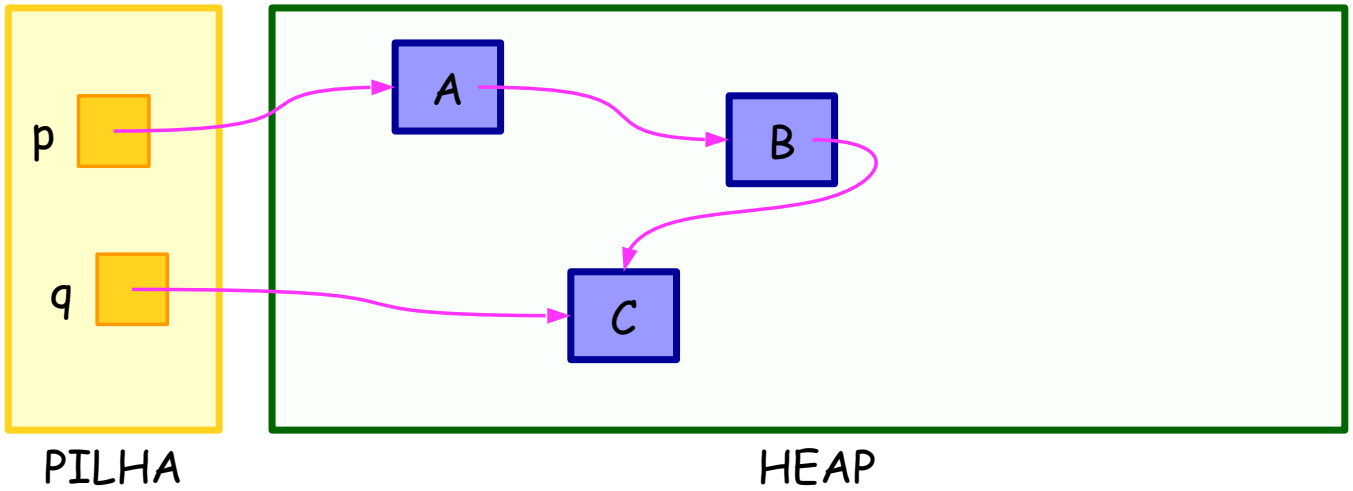
Um exemplo mais complexo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C

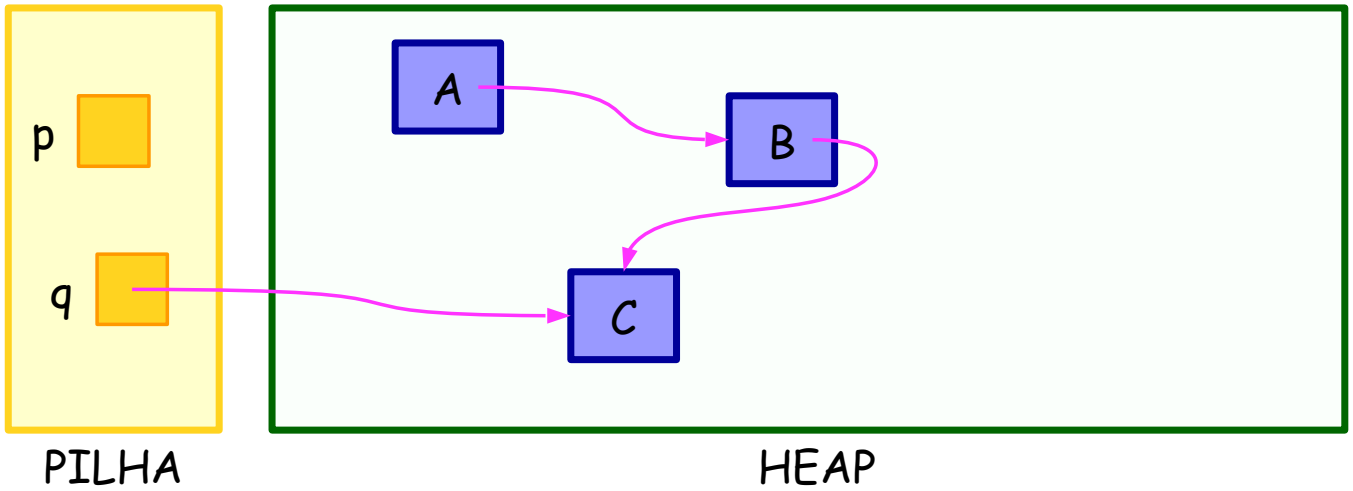
Um exemplo mais complexo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C
4. $q \leftarrow p.prox.prox$

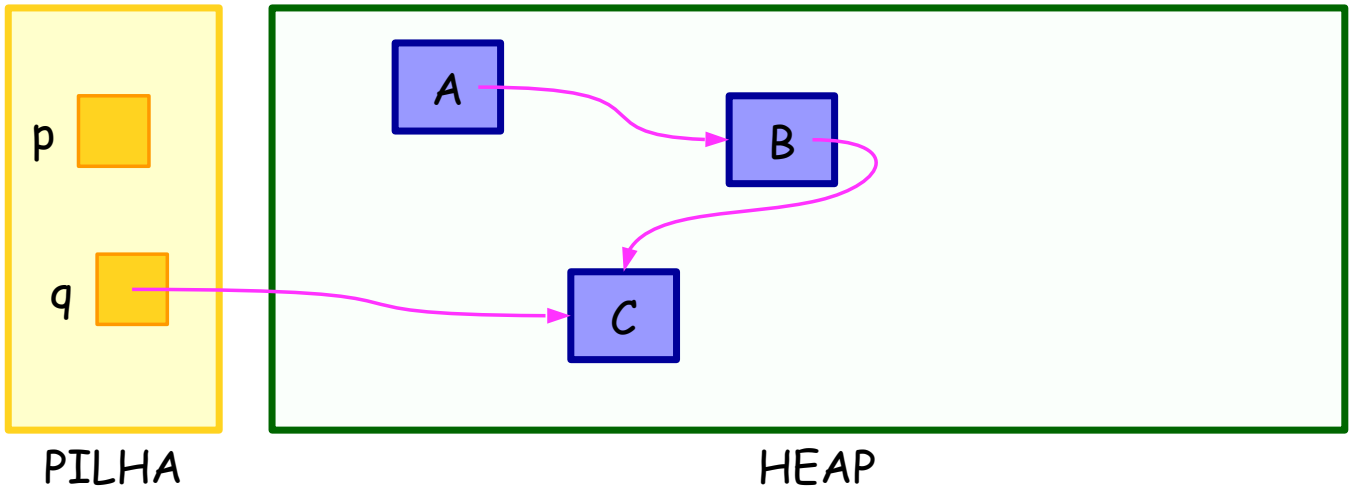
Um exemplo mais complexo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C
4. $q \leftarrow p.prox.prox$
5. $p \leftarrow$ NULL

Um exemplo mais complexo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C
4. $q \leftarrow p.prox.prox$
5. $p \leftarrow$ `NULL`

A partir desse momento, a memória de A e B pode ser liberada.

Coleta de lixo

Coleta de lixo

É o algoritmo executado para liberar a memória dos nós inacessíveis.

Coleta de lixo

Coleta de lixo

É o algoritmo executado para liberar a memória dos nós inacessíveis.

Fases

- Descoberta de nós inacessíveis:
  - marca todos os nós que não podem mais ser acessados
  - pode utilizar uma busca simples em grafo
- Liberação de espaço:
  - devolve todos os blocos marcados à lista de blocos livres
  - pode envolver a compactação dos blocos reservados

Coleta de lixo

Coleta de lixo

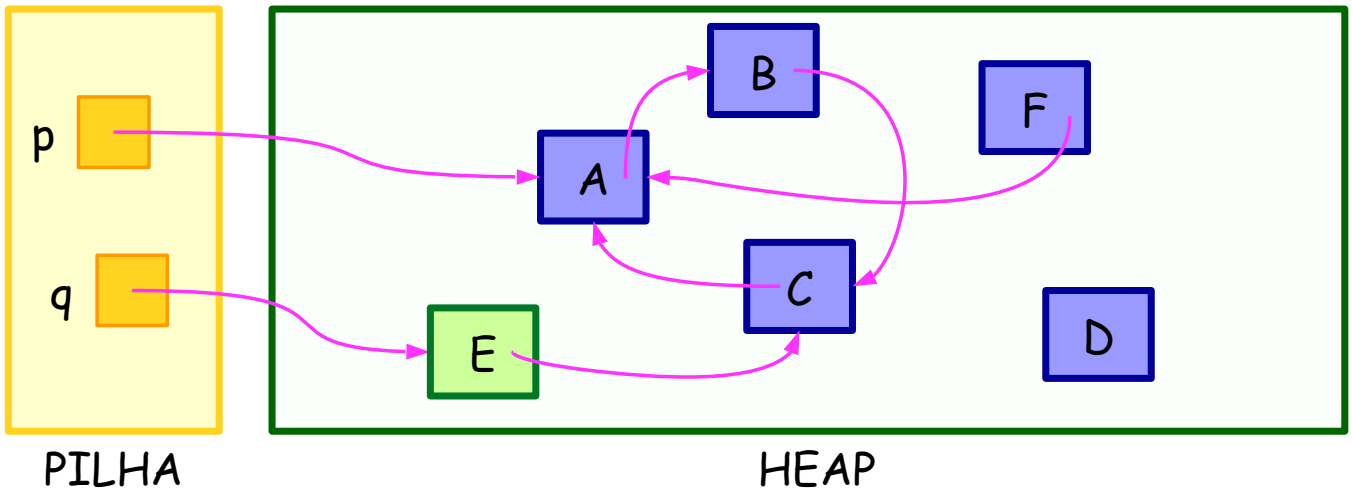
É o algoritmo executado para liberar a memória dos nós inacessíveis.

Fases

- Descoberta de nós inacessíveis:
  - marca todos os nós que não podem mais ser acessados
  - pode utilizar uma busca simples em grafo
- Liberação de espaço:
  - devolve todos os blocos marcados à lista de blocos livres
  - pode envolver a compactação dos blocos reservados

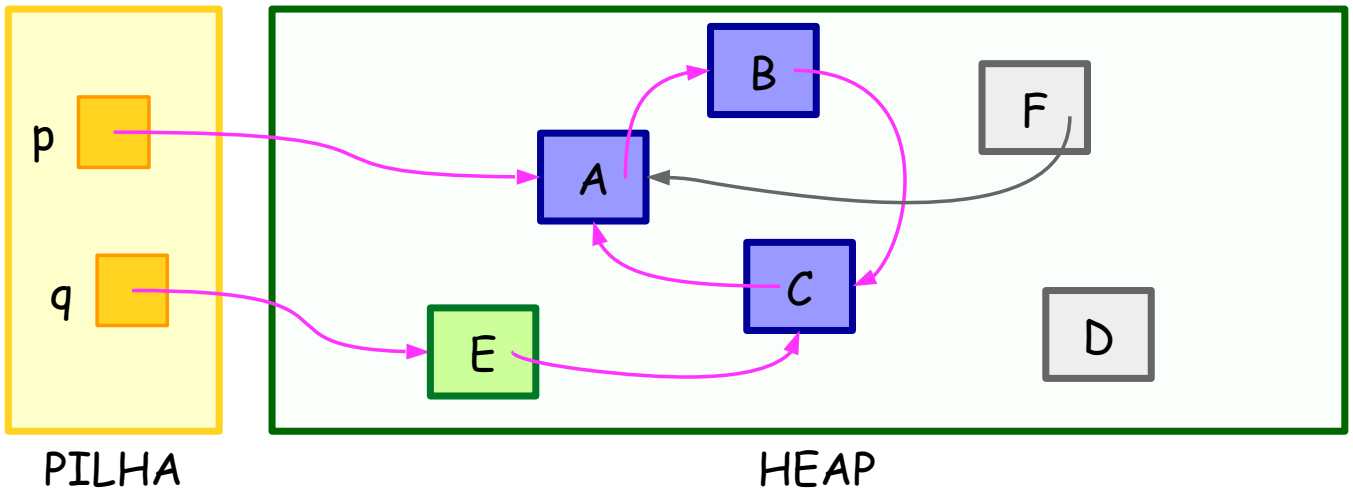
Observação: A coleta de lixo pode ser executada a qualquer momento!

Coleta de lixo: exemplo



Procedimentos

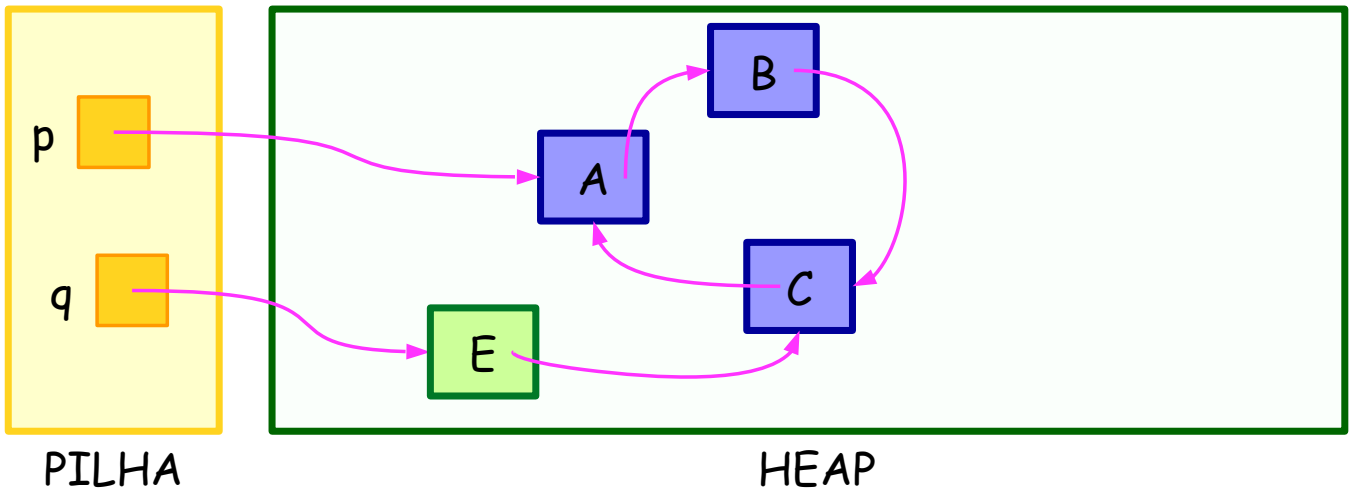
Coleta de lixo: exemplo



Procedimentos

1. Marca nós inacessíveis

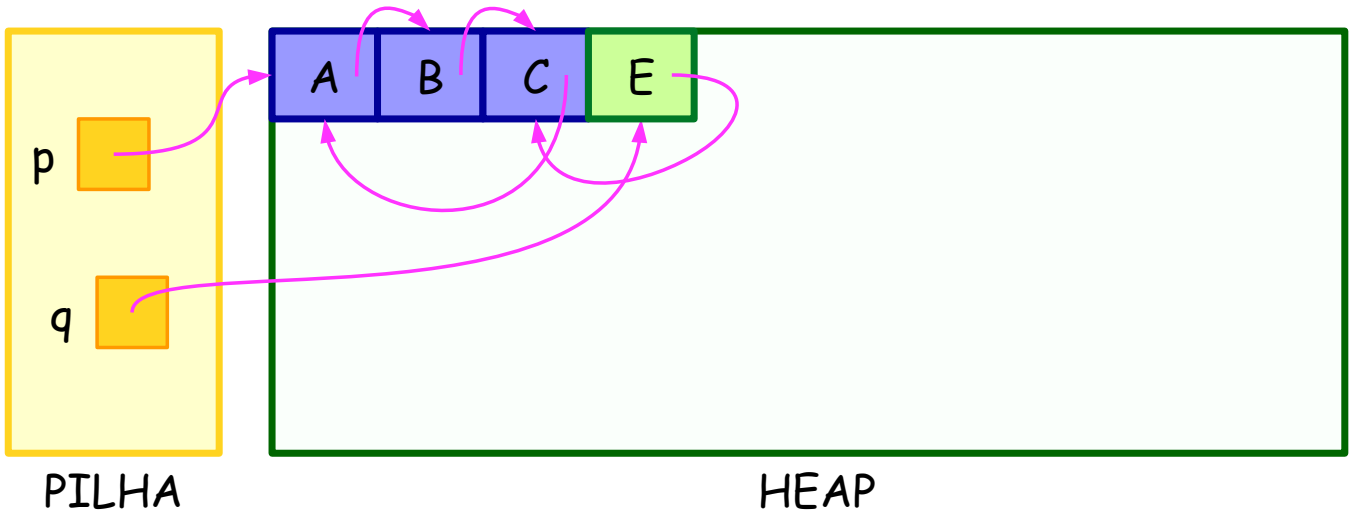
Coleta de lixo: exemplo



Procedimentos

1. Marca nós inacessíveis
2. Libera nós inacessíveis

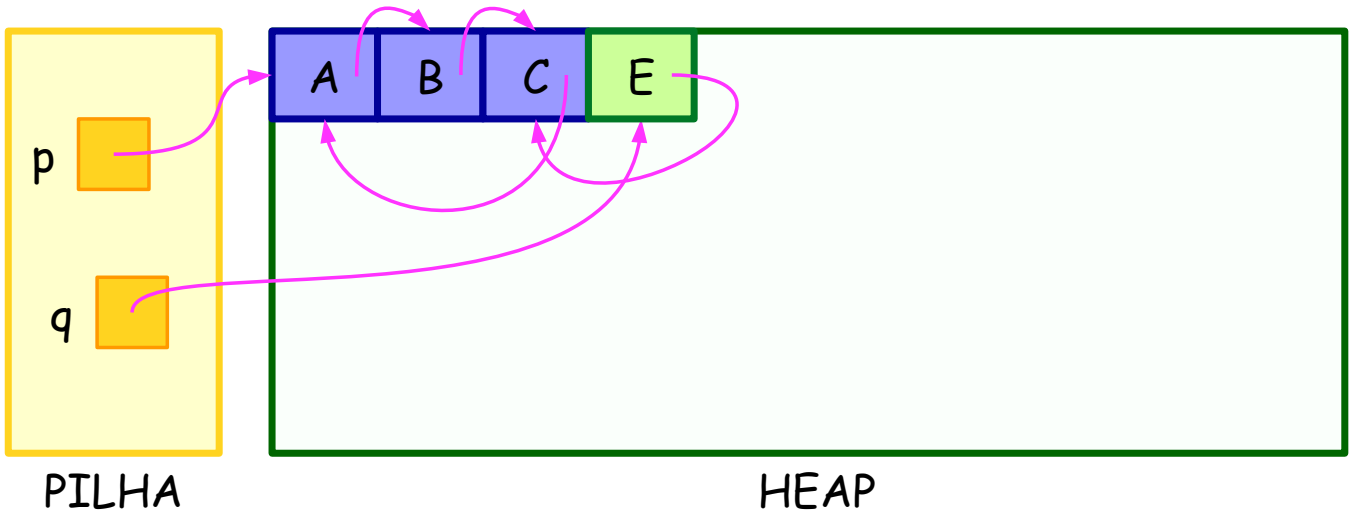
Coleta de lixo: exemplo



Procedimentos

1. Marca nós inacessíveis
2. Libera nós inacessíveis
3. Compacta blocos reservados

Coleta de lixo: exemplo



Procedimentos

1. Marca nós inacessíveis
2. Libera nós inacessíveis
3. Compacta blocos reservados (copia dados e atualiza ponteiros!)

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

- Ao criar um nó:

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

- Ao criar um nó:
  - aloca espaço para o nó
  - inicializa contador com valor 1

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

- Ao criar um nó:
  - aloca espaço para o nó
  - inicializa contador com valor 1
- Ao copiar referência:

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

- Ao criar um nó:
  - aloca espaço para o nó
  - inicializa contador com valor 1
- Ao copiar referência:
  - incrementa contador do nó

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

- Ao criar um nó:
  - aloca espaço para o nó
  - inicializa contador com valor 1
- Ao copiar referência:
  - incrementa contador do nó
- Ao apagar referência:

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

- Ao criar um nó:
  - aloca espaço para o nó
  - inicializa contador com valor 1
- Ao copiar referência:
  - incrementa contador do nó
- Ao apagar referência:
  - decrementa contador do nó

Contagem de referências

Contagem de referências

É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

- Ao criar um nó:
  - aloca espaço para o nó
  - inicializa contador com valor 1
- Ao copiar referência:
  - incrementa contador do nó
- Ao apagar referência:
  - decrementa contador do nó
- Quando contador ative valor 0:

Contagem de referências

Contagem de referências

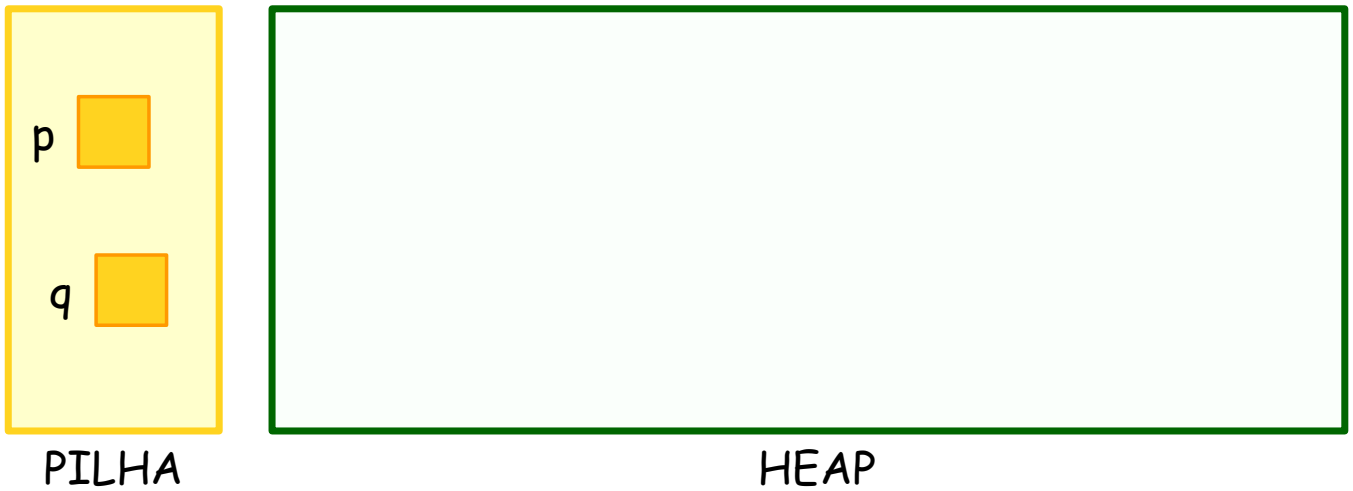
É a estratégia de gerenciamento de memória que libera a memória usada por um nó tão logo não haja mais referências a ele.

Ideia: Contamos as referências a cada nó.

Funcionamento

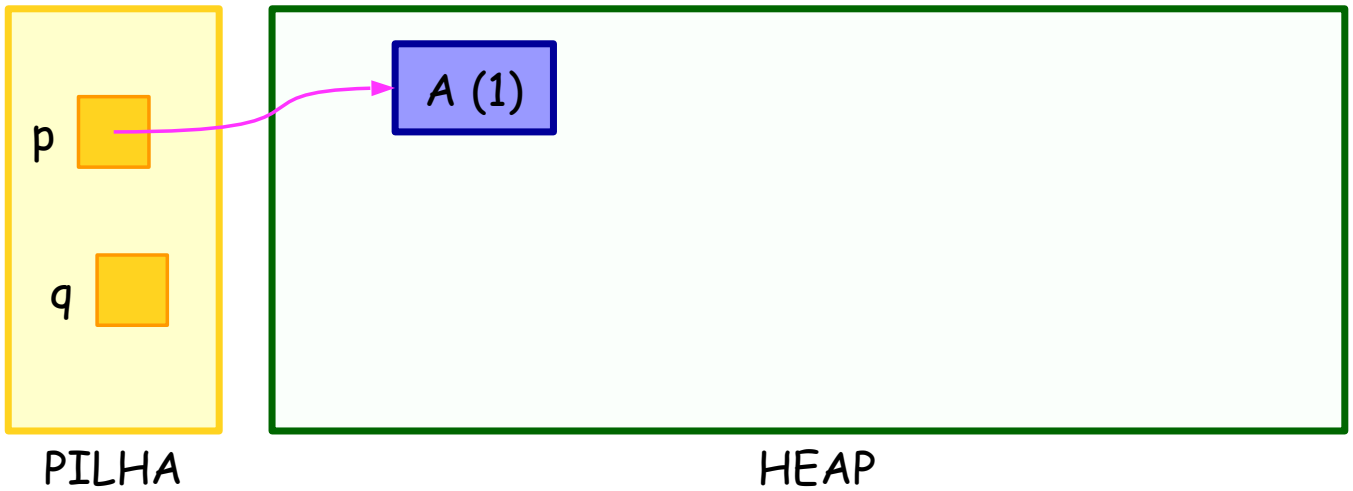
- Ao criar um nó:
  - aloca espaço para o nó
  - inicializa contador com valor 1
- Ao copiar referência:
  - incrementa contador do nó
- Ao apagar referência:
  - decrementa contador do nó
- Quando contador ative valor 0:
  - apaga referências a outros objetos
  - libera a memória utilizada pelo nó

Contagem de referências: exemplo



Criando uma lista

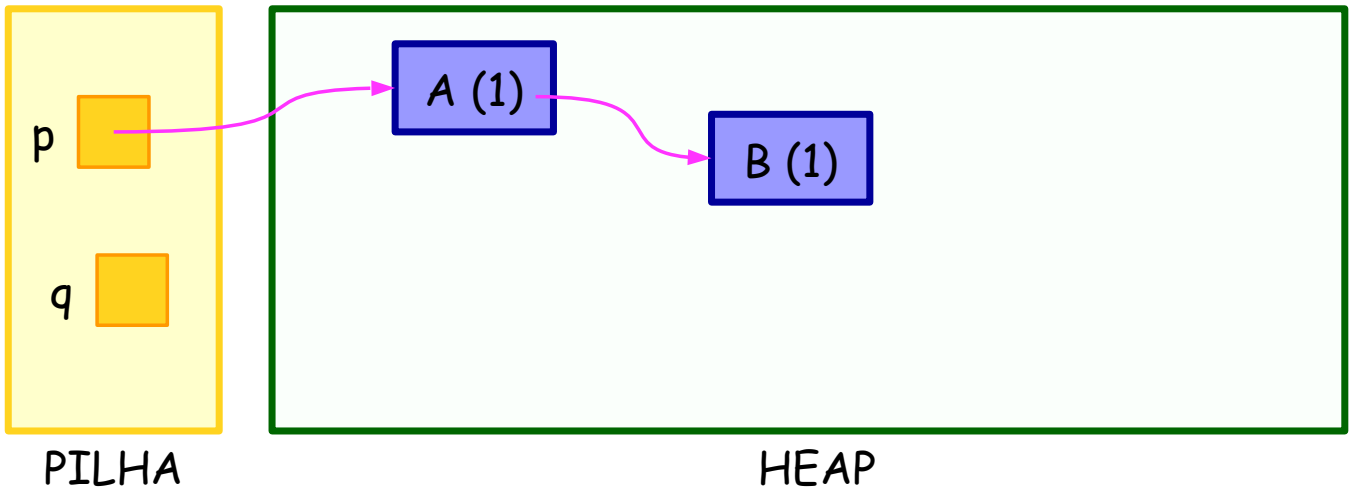
Contagem de referências: exemplo



Criando uma lista

1. $p \leftarrow$ cria nó A

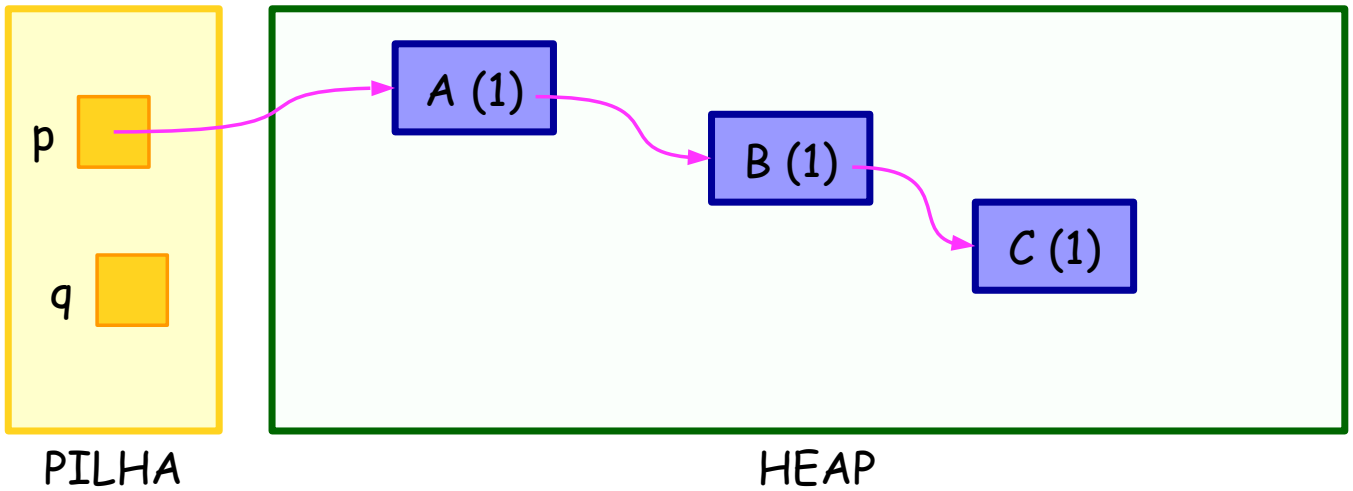
Contagem de referências: exemplo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B

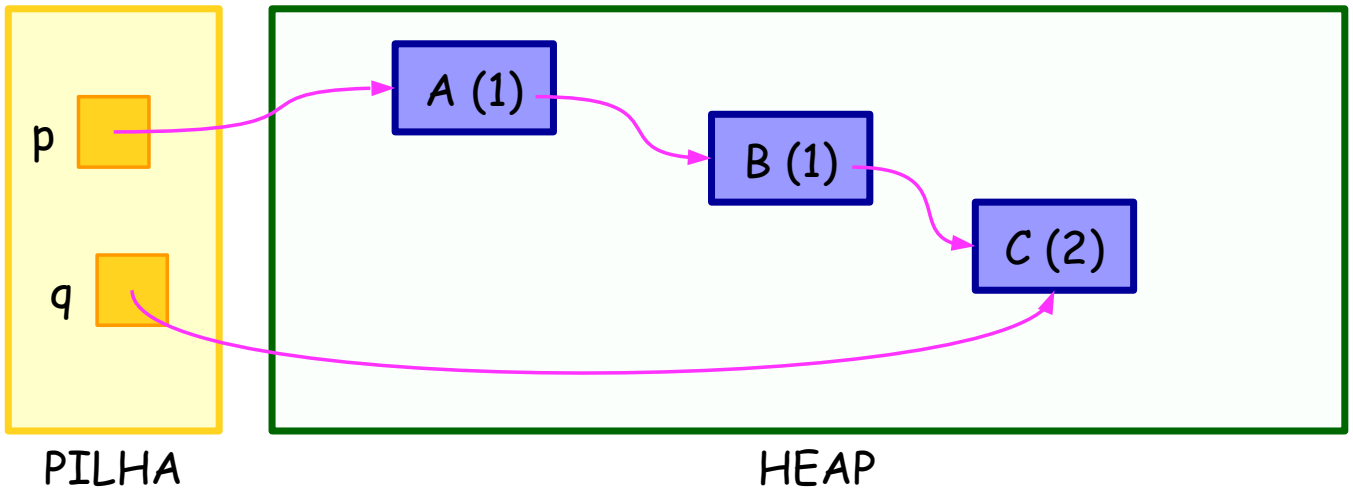
Contagem de referências: exemplo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C

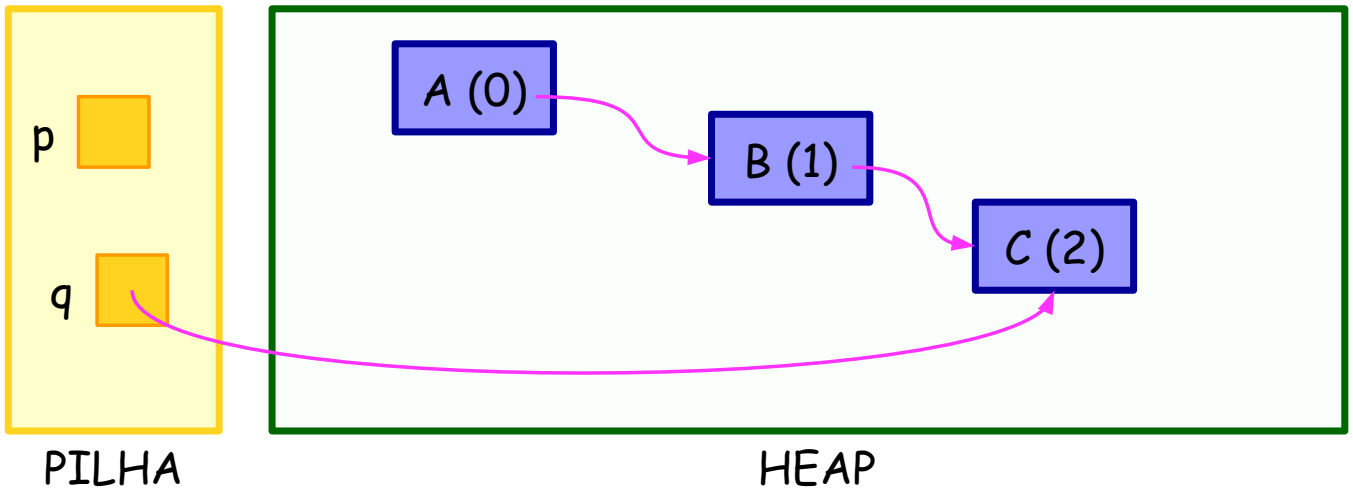
Contagem de referências: exemplo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C
4. $q \leftarrow p.prox.prox$

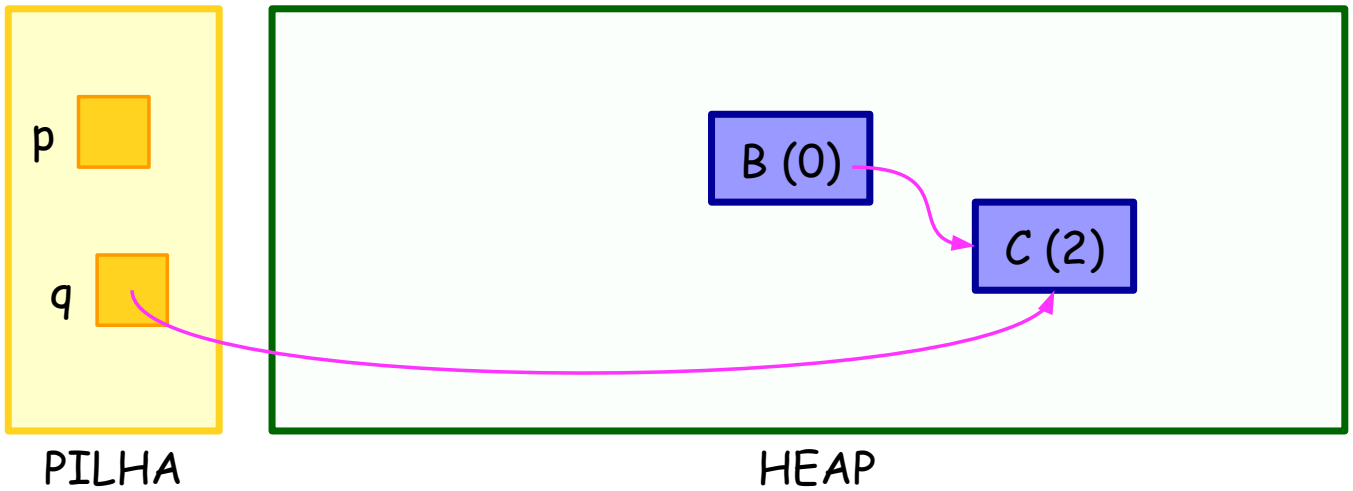
Contagem de referências: exemplo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C
4. $q \leftarrow p.prox.prox$
5. $p \leftarrow$ NULL

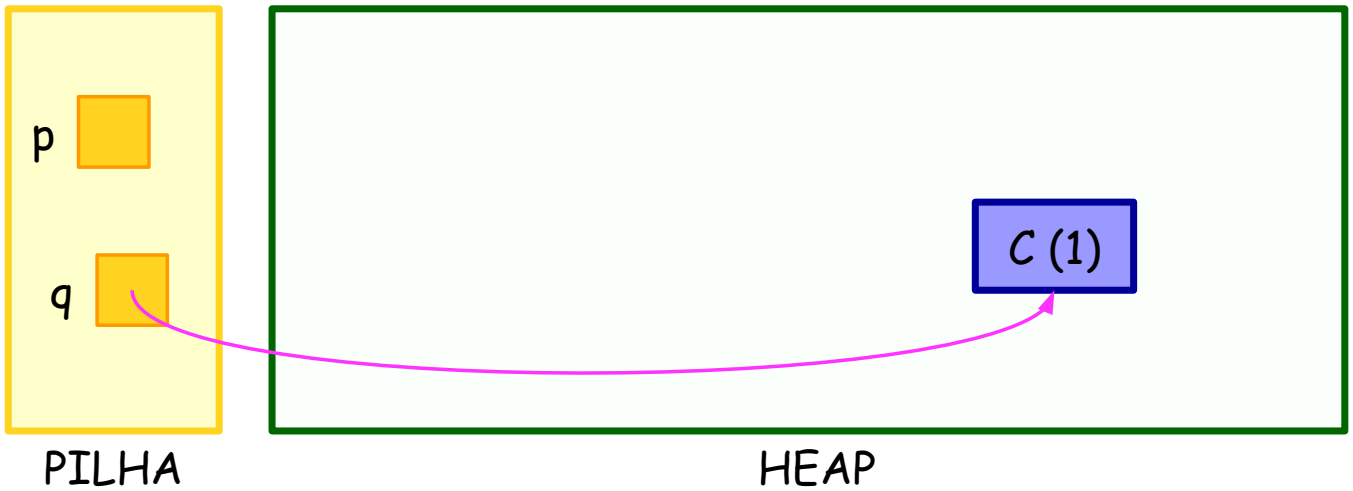
Contagem de referências: exemplo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C
4. $q \leftarrow p.prox.prox$
5. $p \leftarrow$ NULL

Contagem de referências: exemplo



Criando uma lista

1. $p \leftarrow$ cria nó A
2. $p.prox \leftarrow$ cria nó B
3. $p.prox.prox \leftarrow$ cria nó C
4. $q \leftarrow p.prox.prox$
5. $p \leftarrow$ NULL

Exercícios - Alocação de memória

Considere a seguinte situação. Um programa resolve um problema utilizando a técnica de retrocesso. A solução é representada por um vetor de n posições. A primeira versão utilizava recursão explicitamente. Infelizmente, devido ao tamanho das variáveis locais necessárias em cada chamada, o programa só rodava com instâncias pequenas (isso é, para n pequeno). Ou seja, acontecia *stack overflow* e o programa travava.

Para resolver o problema, a função foi reimplementada usando uma pilha com lista encadeada. Curiosamente, embora agora o algoritmo não travava para valores de n grandes, para n pequeno a versão anterior era muito mais rápida.

1. Escreva um pequeno parágrafo definindo o que é uma lista de blocos livres e como e quando ela pode afetar a eficiência de um programa.
2. Explique porque a primeira versão funcionava mais rapidamente que a segunda para n pequeno. Dê sugestões para corrigir esse problema.

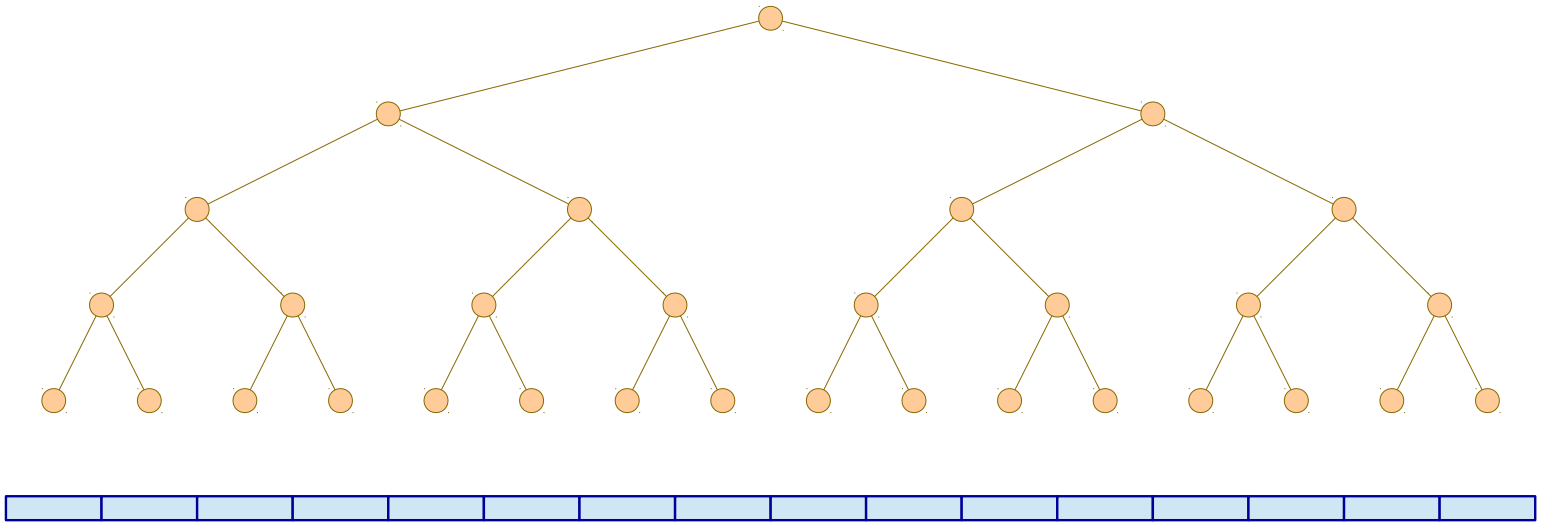
Exercícios - Gerenciamento de memória

1. Coleta de lixo: Uma questão problemática da coleta de lixo é que não há controle sobre quando a coleta é executada. Considerando isso, tente explicar porque a seguinte situação hipotética acontece:
   - Um programa tem como objetivo calcular e devolver o resultado de algumas simulações complexas que dependem dos parâmetros digitados; o programa é interativo e deve devolver a resposta tão logo os parâmetros forem digitados e a tecla ENTER for pressionada. Na maioria das vezes, o programa responde instantaneamente, mas algumas vezes, ele pode demorar alguns instantes (até mesmo segundos) antes de dar uma resposta. Por quê?
2. Contagem de referências: A contagem de referências sozinha não é suficiente para garantir que todos os nós inacessíveis sejam liberados. Para isso é necessário tomar cuidados adicionais
   - verifique essa situação fazendo o seguinte: (i) adicione os contadores de referência no exemplo do slide 18; (ii) faça $p \leftarrow$ `NULL` e $q \leftarrow$ `NULL`; (iii) libere a memória dos nós com contadores zerados
   - pesquise e sugira cuidados adicionais para evitar essa situação

Extra: Sistema de pares

Sistema de pares (*Buddy System*)

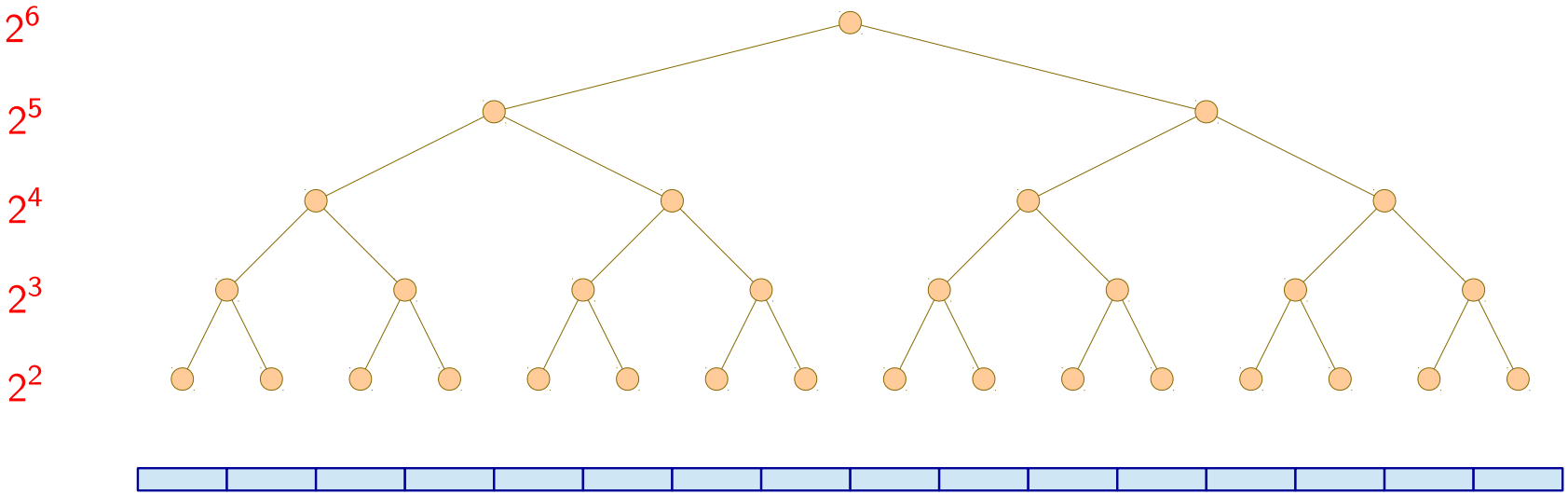
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares

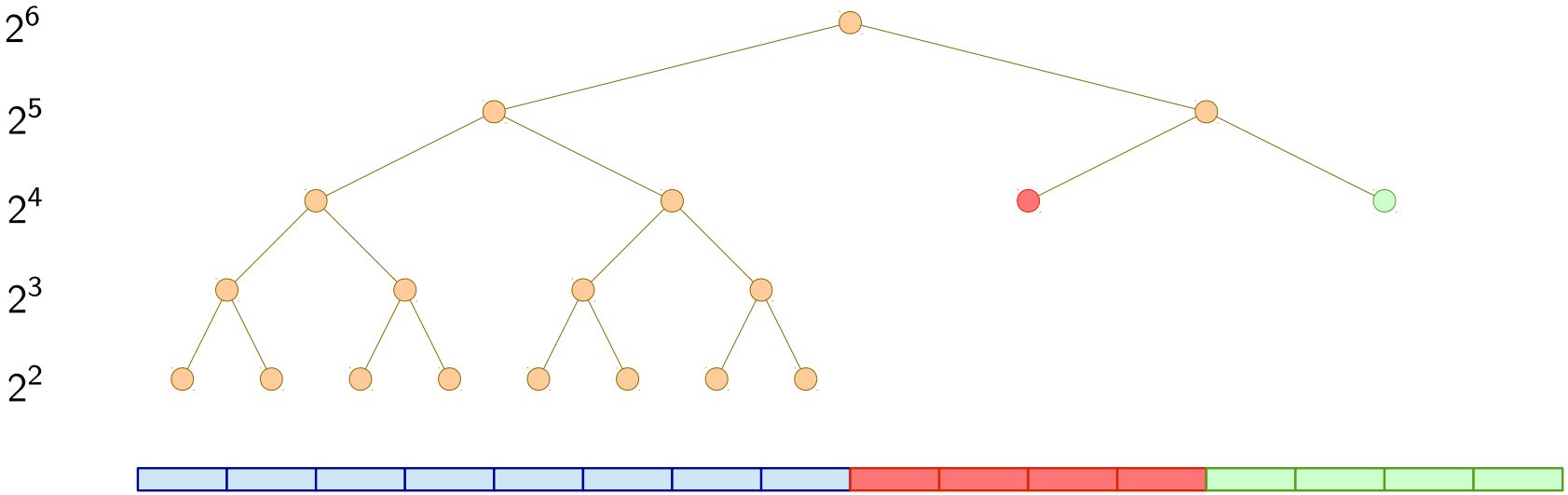
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k

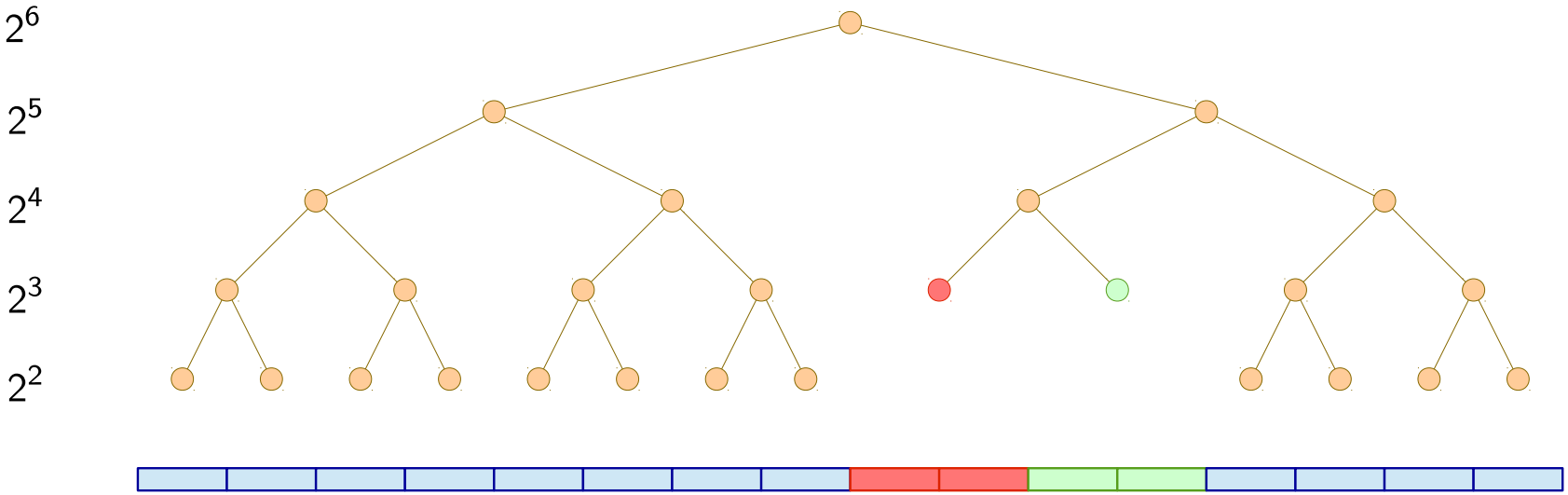
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k
- os pares têm mesmo tamanho e são adjacentes

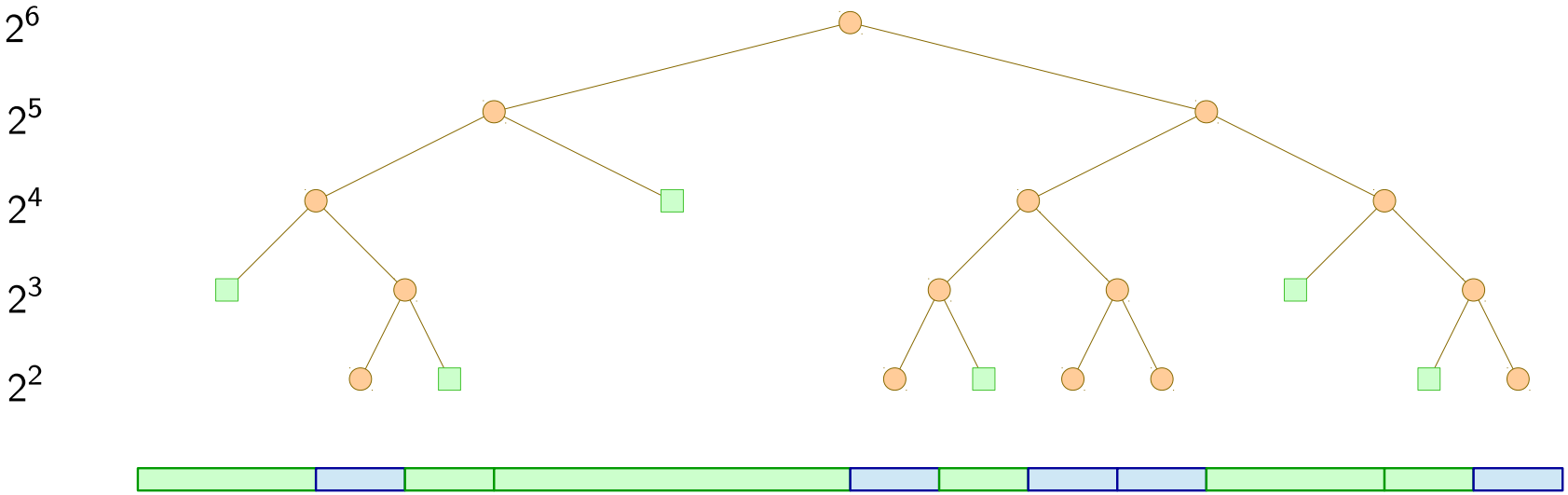
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k
- os pares têm mesmo tamanho e são adjacentes

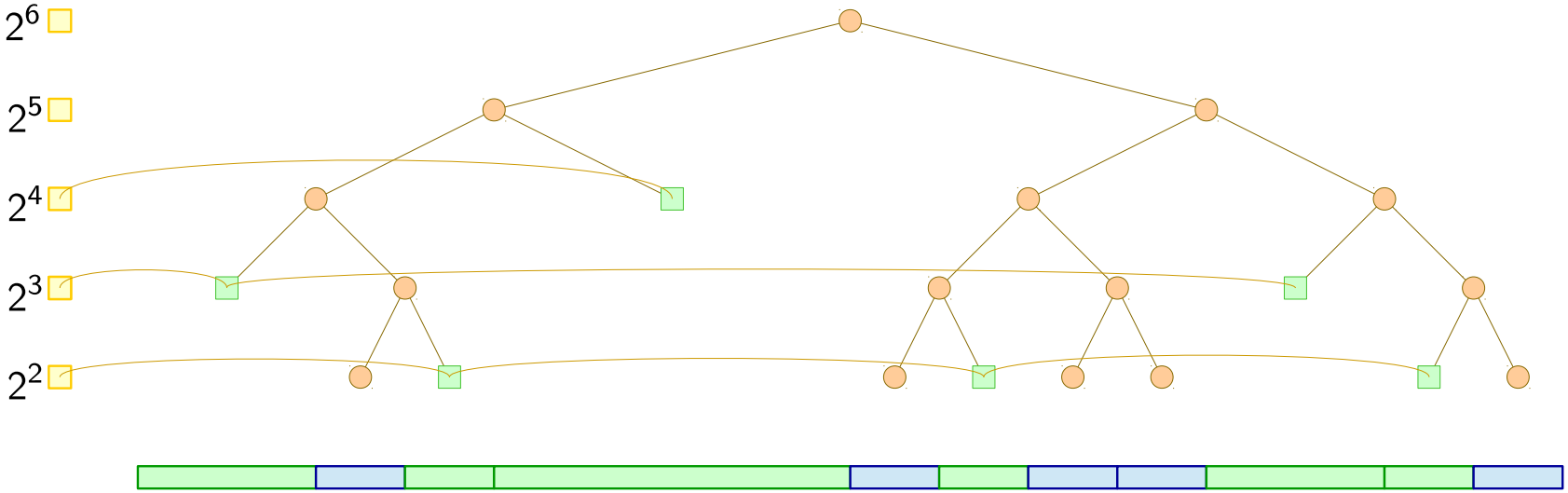
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k
- os pares têm mesmo tamanho e são adjacentes
- blocos podem estar livres ou reservados

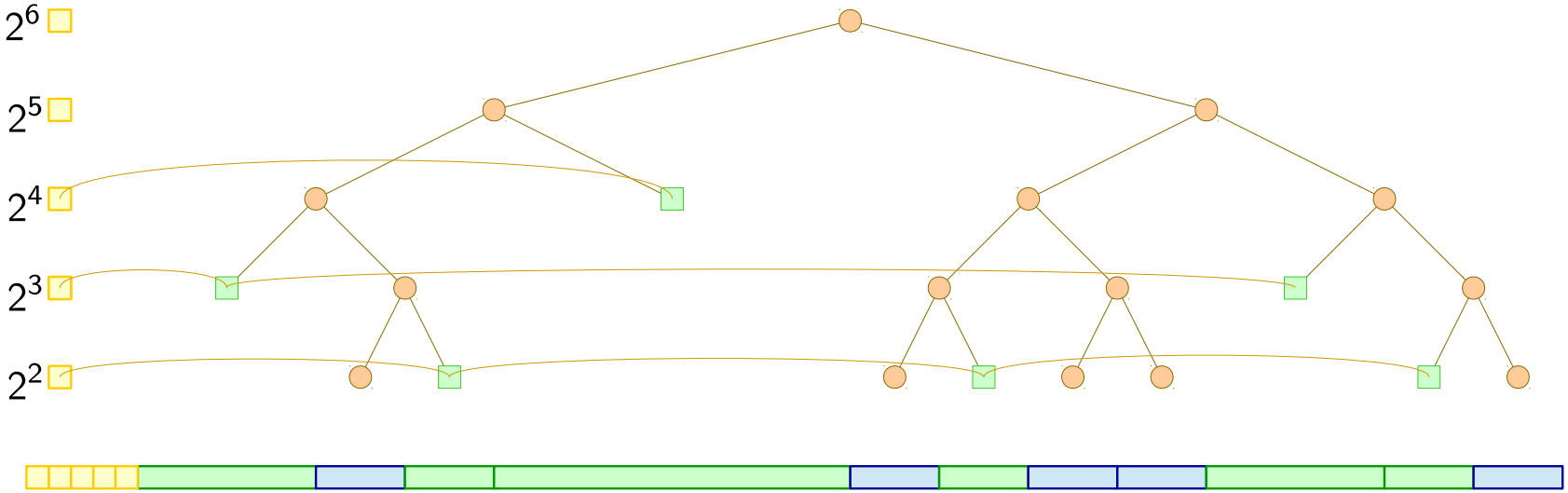
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k
- os pares têm mesmo tamanho e são adjacentes
- blocos podem estar livres ou reservados
- há um lista dupla para blocos livres de tamanho 2^k para cada k

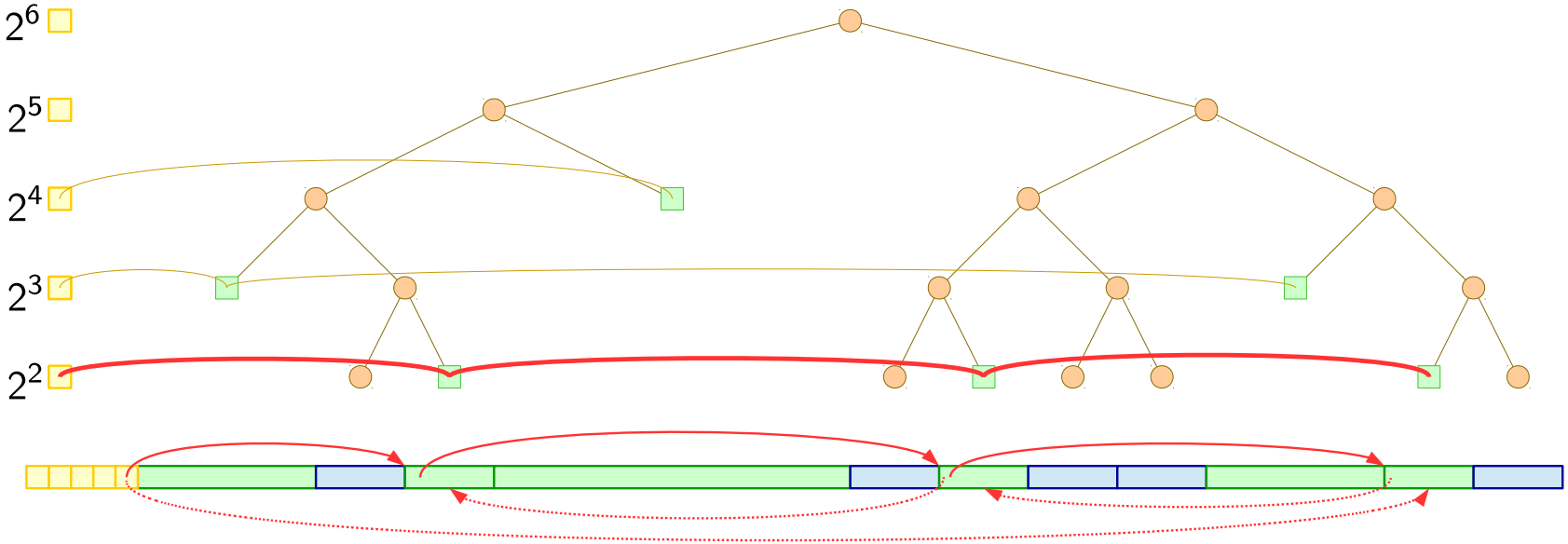
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k
- os pares têm mesmo tamanho e são adjacentes
- blocos podem estar livres ou reservados
- há um lista dupla para blocos livres de tamanho 2^k para cada k
- as cabeças das listas ficam separadas no início da memória

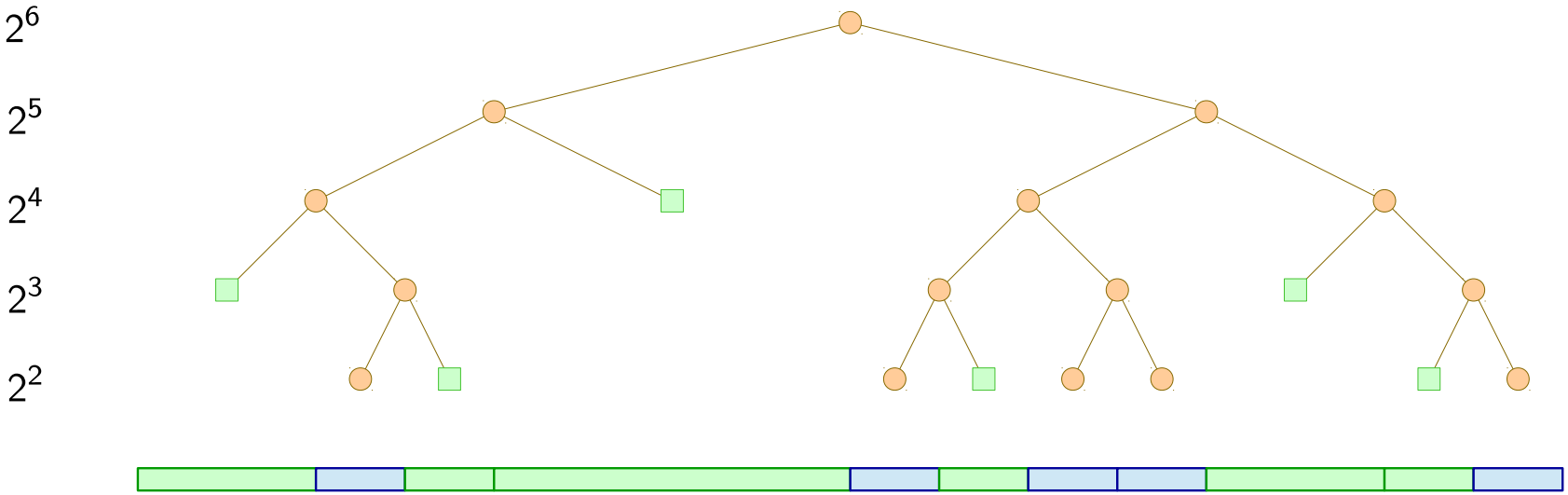
Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k
- os pares têm mesmo tamanho e são adjacentes
- blocos podem estar livres ou reservados
- há um lista dupla para blocos livres de tamanho 2^k para cada k
- as cabeças das listas ficam separadas no início da memória
- o início de cada bloco contém informações: ponteiros, tamanho,se está reservado

Extra: Sistema de pares



Sistema de pares (*Buddy System*)

- a memória é dividida hierarquicamente em pares
- todos blocos têm tamanho 2^k para algum k
- os pares têm mesmo tamanho e são adjacentes
- blocos podem estar livres ou reservados
- há um lista dupla para blocos livres de tamanho 2^k para cada k
- as cabeças das listas ficam separadas no início da memória
- o início de cada bloco contém informações: ponteiros, tamanho,se está reservado

Alocando memória com sistema de pares

Alocando memória com sistema de pares

Alocar 5 bytes

Alocando x bytes

Alocando memória com sistema de pares

Alocar 5 bytes

Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$

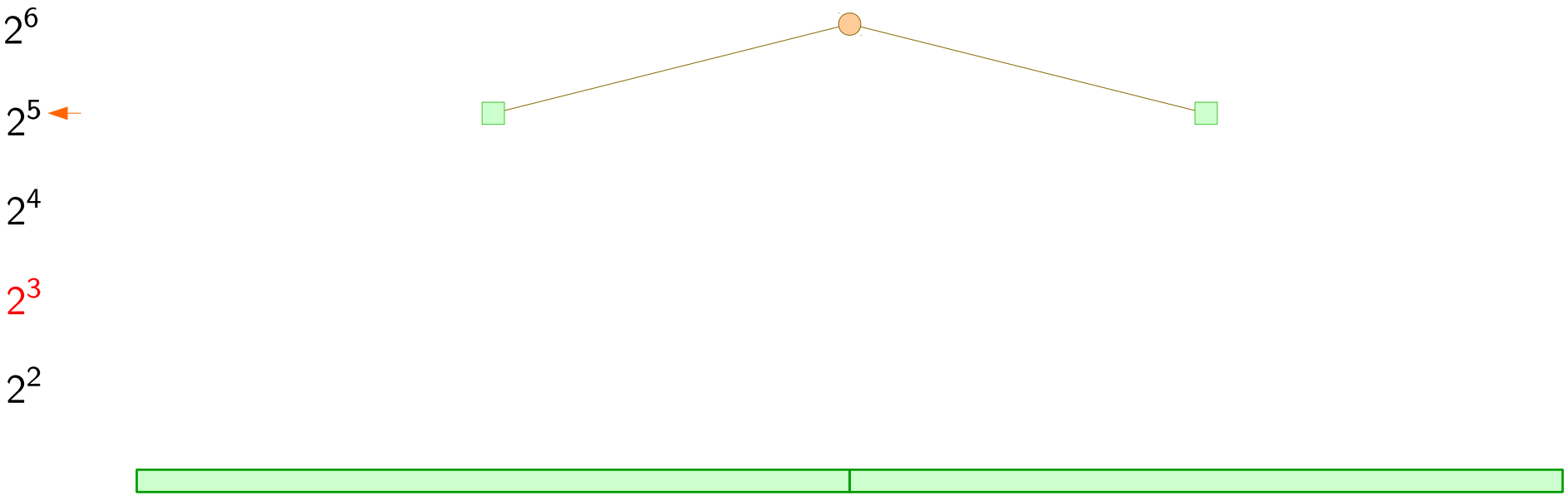
Alocando memória com sistema de pares

Alocar 5 bytes

Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k

Alocando memória com sistema de pares

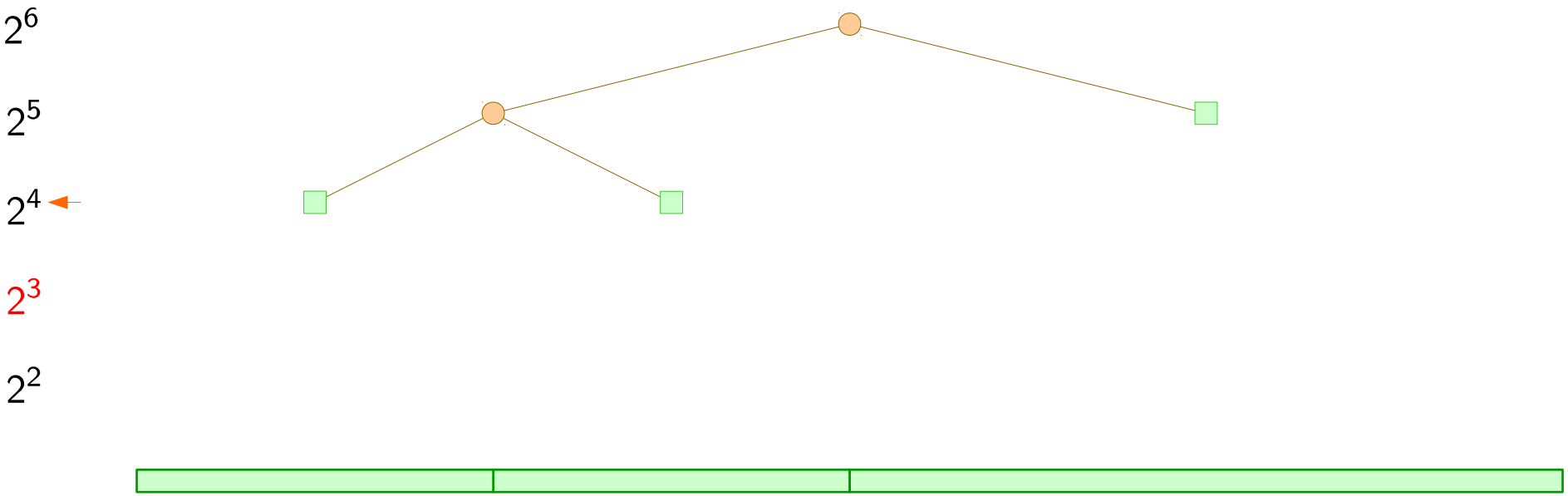


Alocar 5 bytes

Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k

Alocando memória com sistema de pares

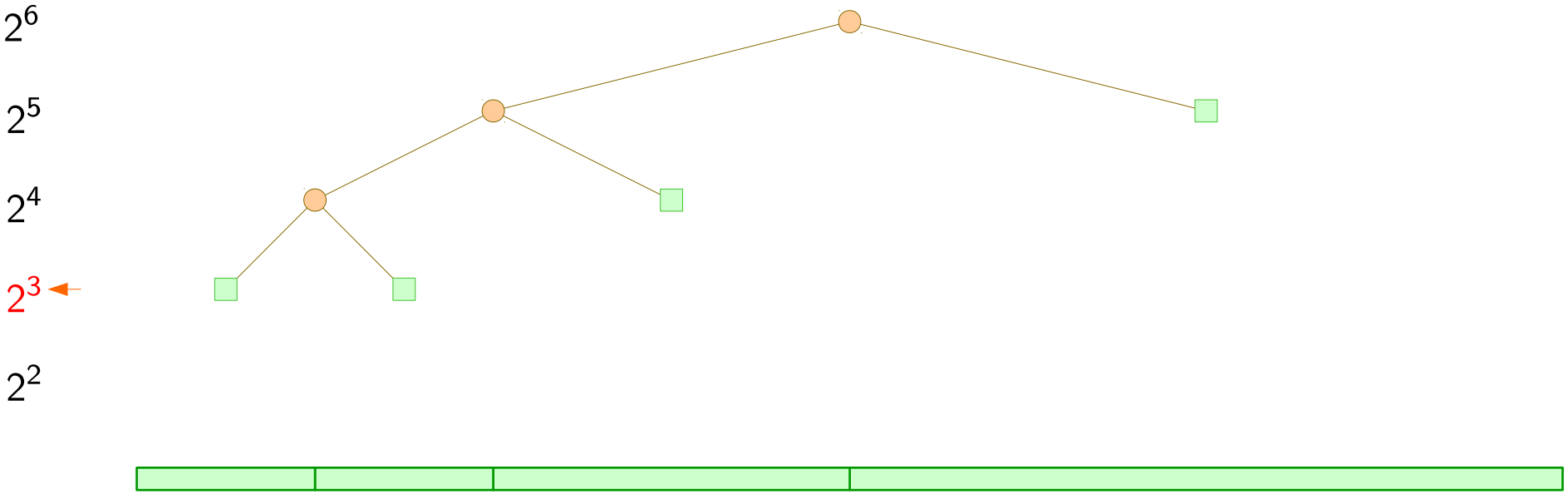


Alocar 5 bytes

Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k

Alocando memória com sistema de pares

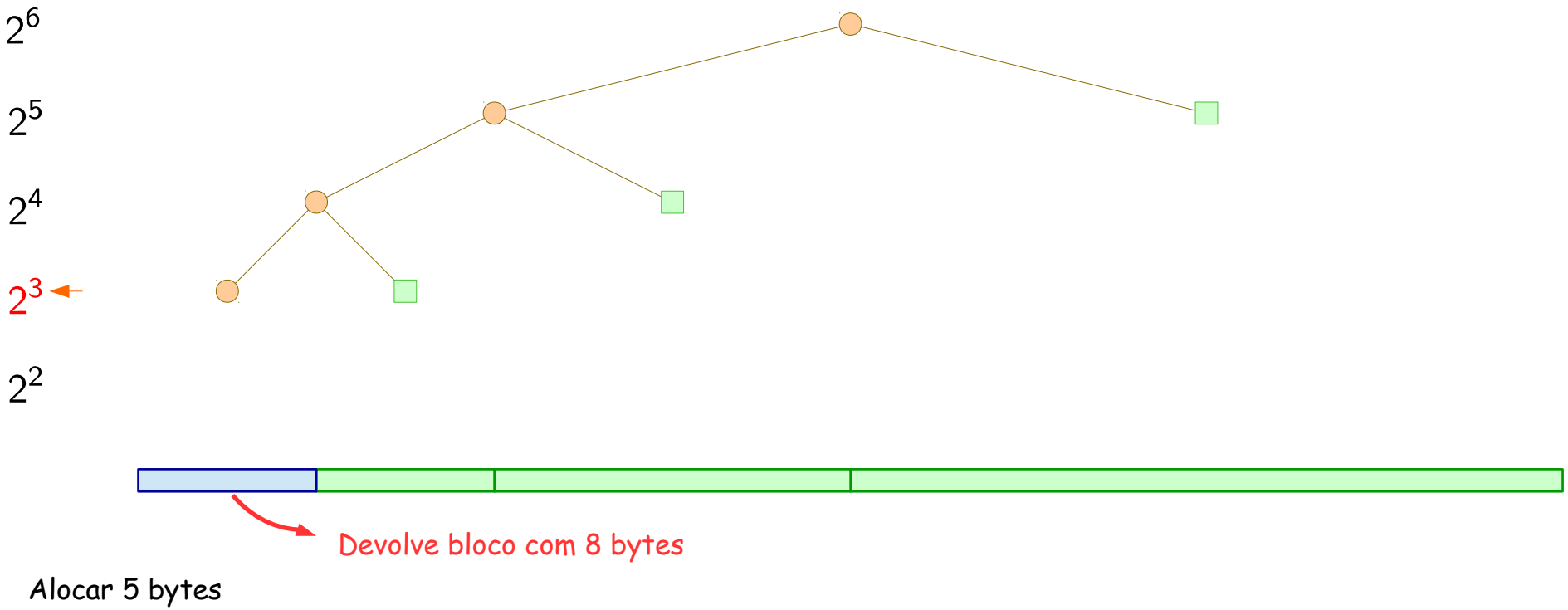


Alocar 5 bytes

Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k

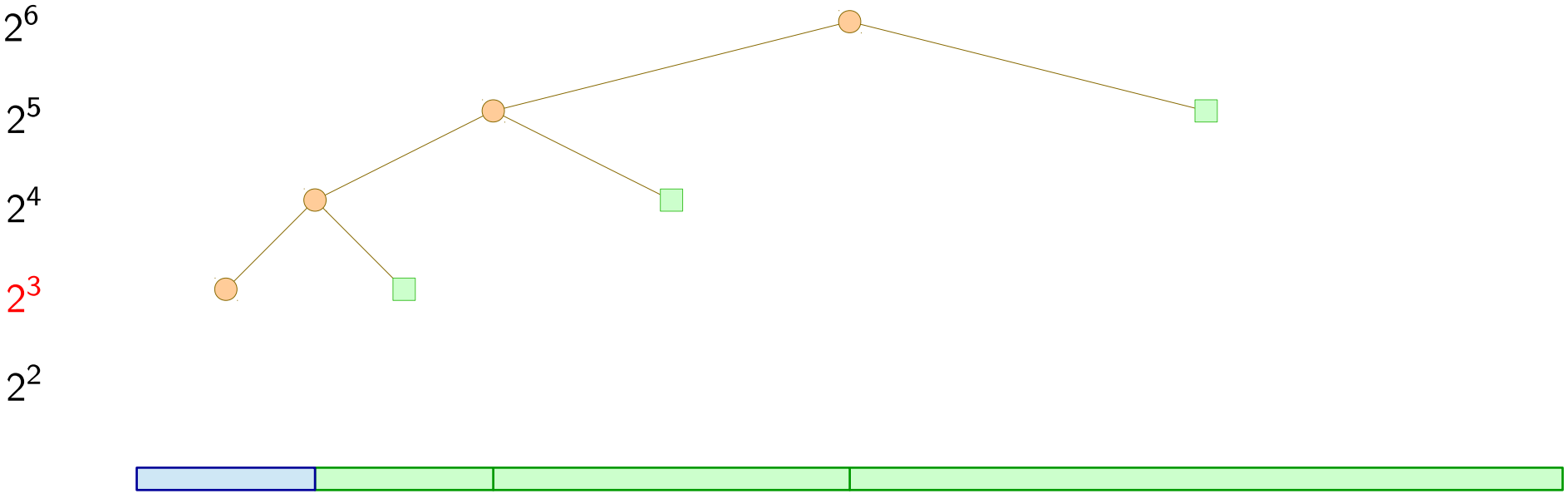
Alocando memória com sistema de pares



Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k
4. Reservar e devolver bloco

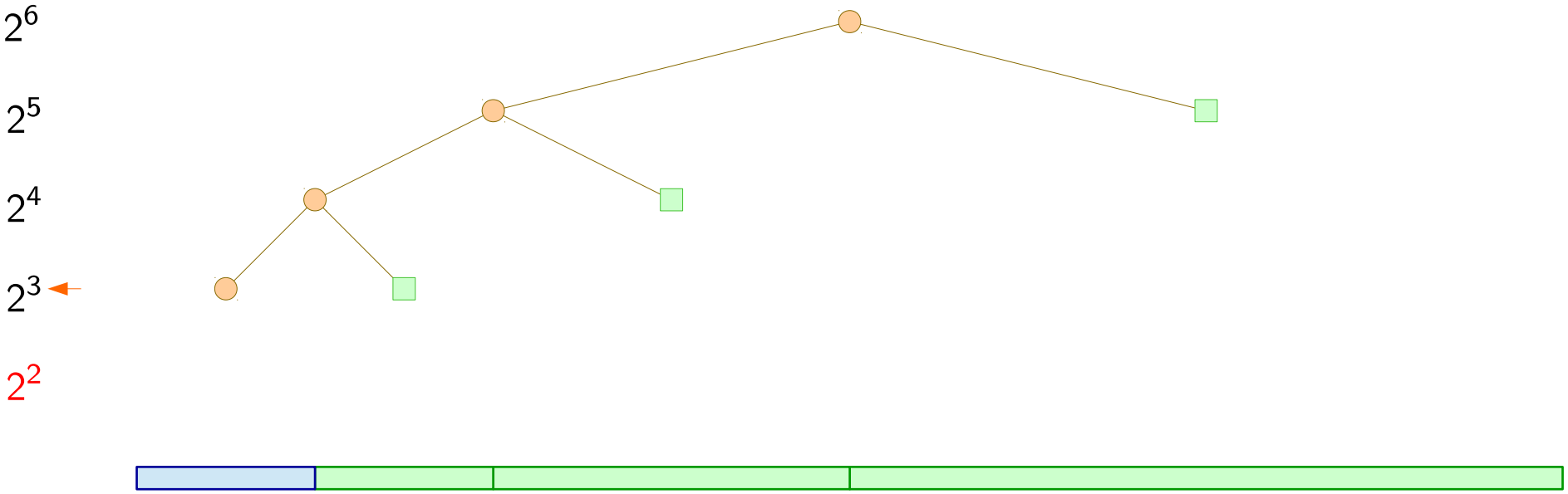
Alocando memória com sistema de pares



Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k
4. Reservar e devolver bloco

Alocando memória com sistema de pares

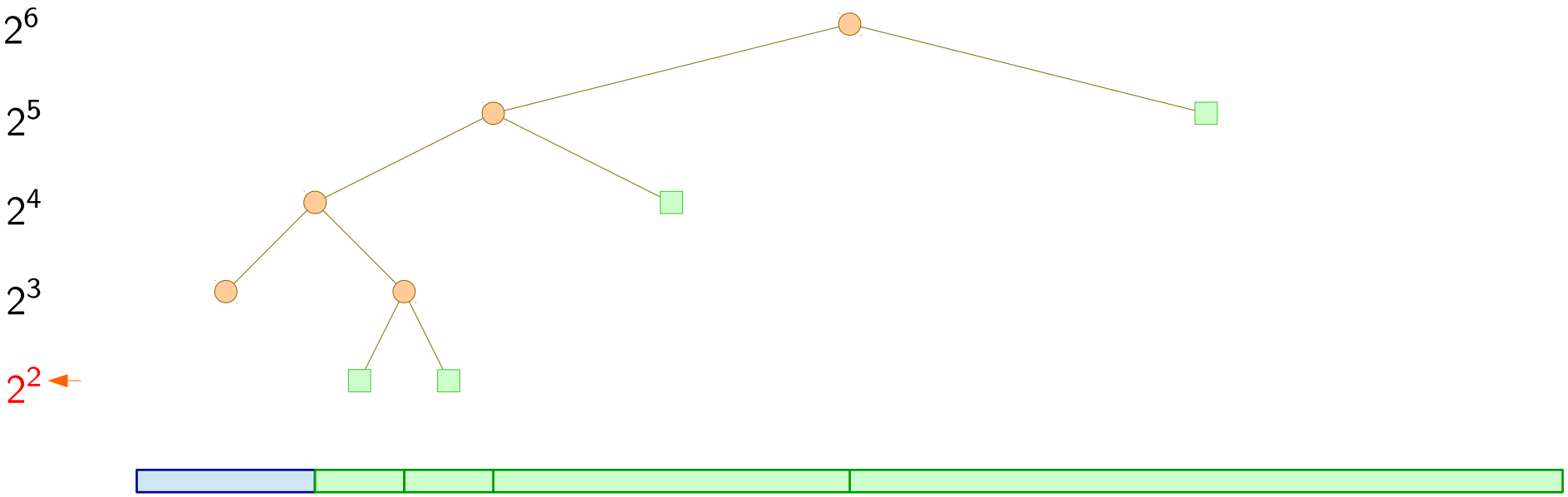


Alocar 4 bytes

Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k
4. Reservar e devolver bloco

Alocando memória com sistema de pares

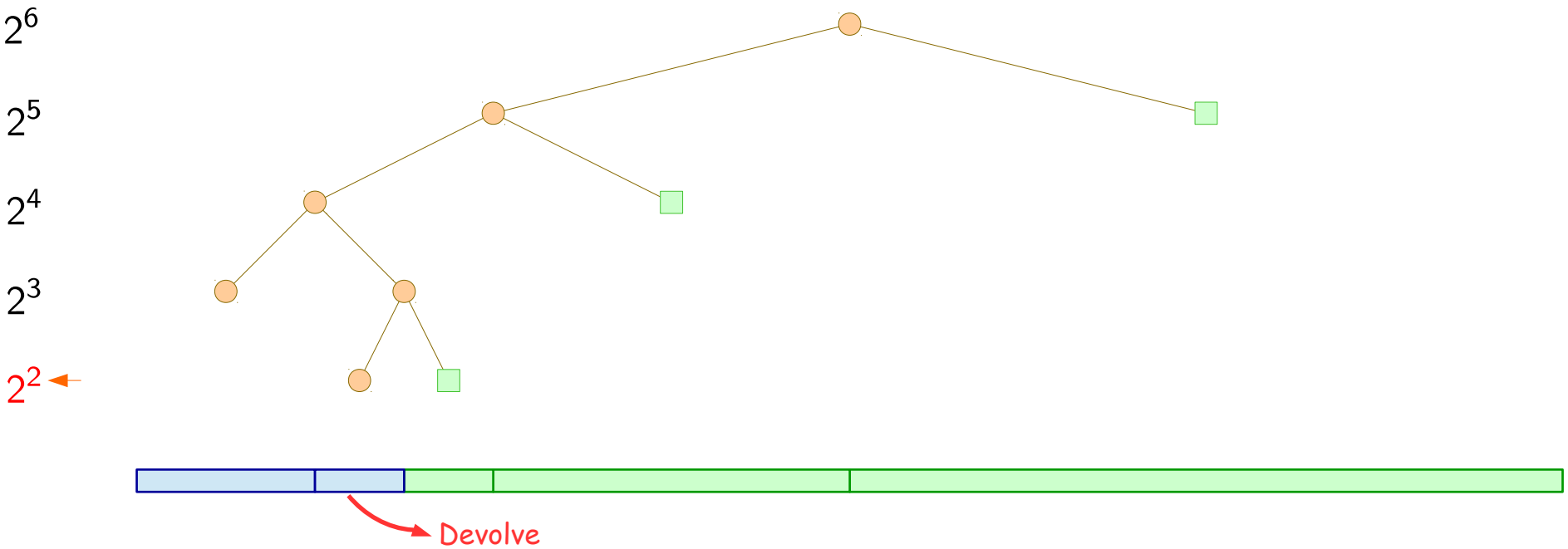


Alocar 4 bytes

Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k
4. Reservar e devolver bloco

Alocando memória com sistema de pares

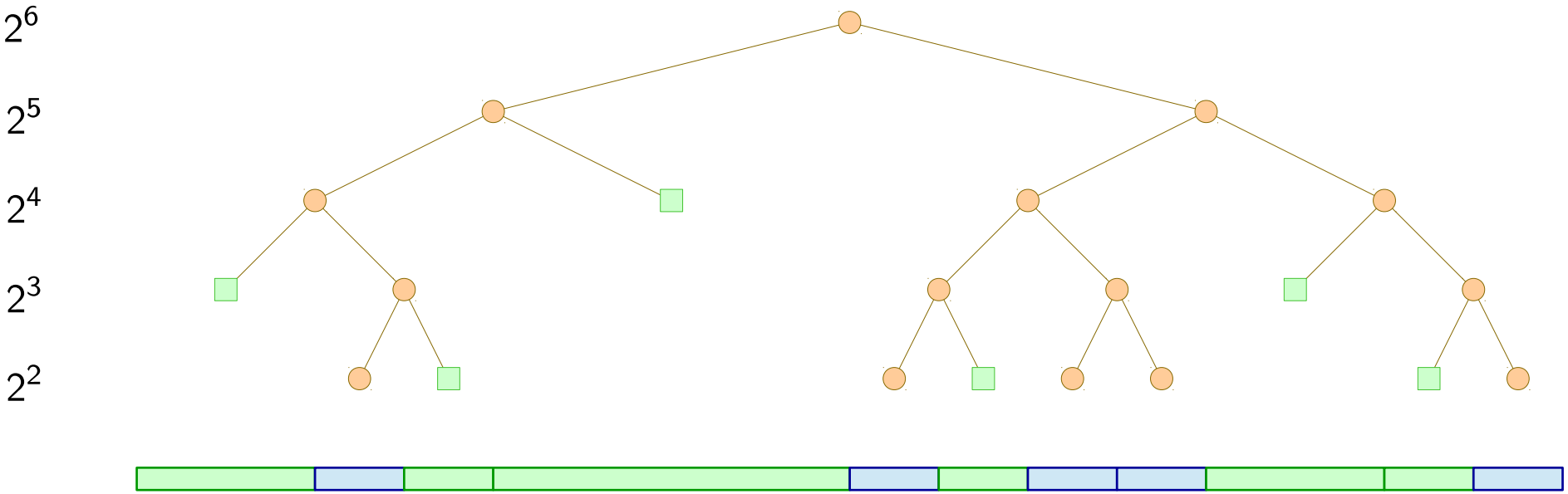


Alocar 4 bytes

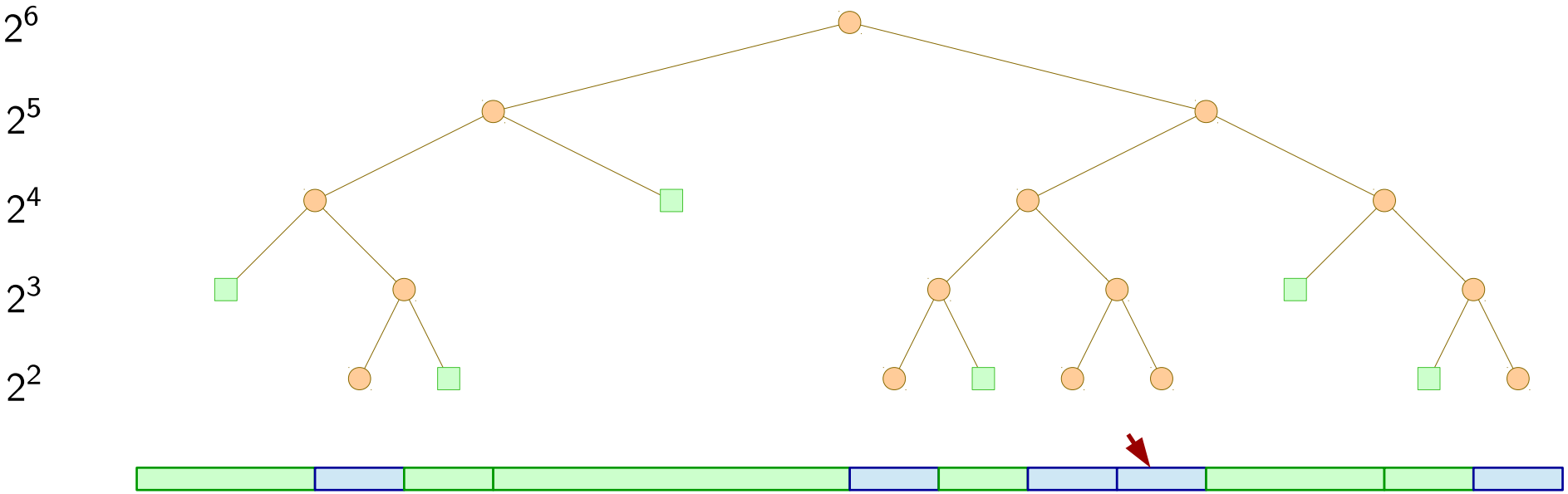
Alocando x bytes

1. Procurar menor k tal que $x \leq 2^k$
2. Procurar menor bloco livre maior ou igual a 2^k
3. Dividir em 2 blocos livres enquanto for maior que 2^k
4. Reservar e devolver bloco

Liberando memória com sistema de pares



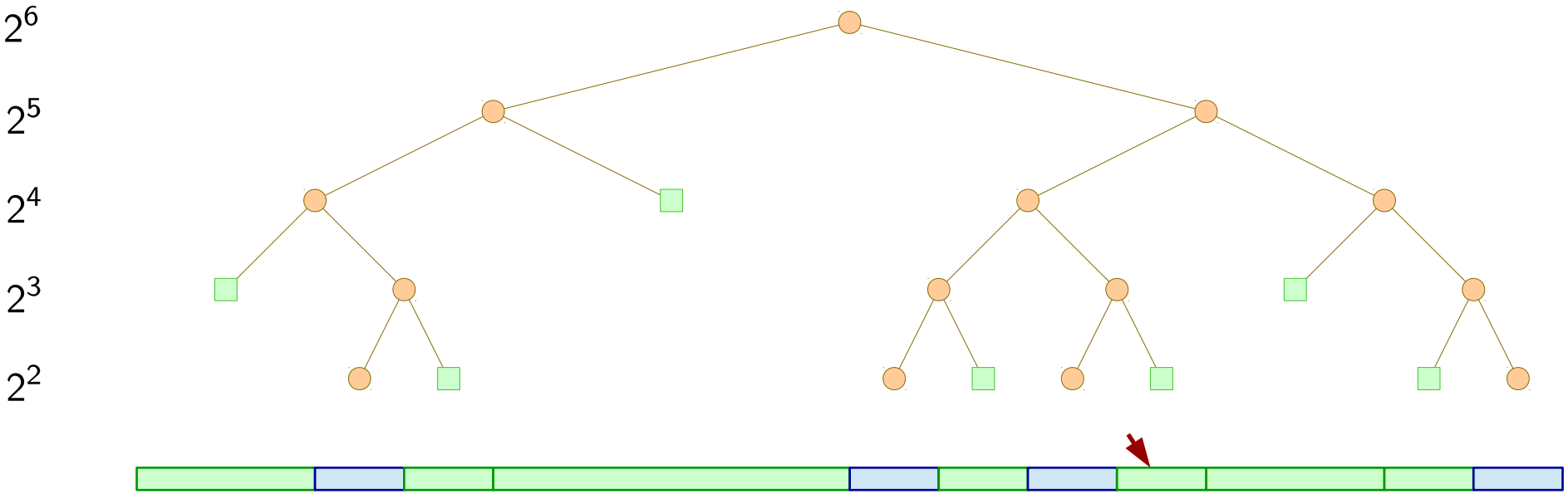
Liberando memória com sistema de pares



Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

Liberando memória com sistema de pares

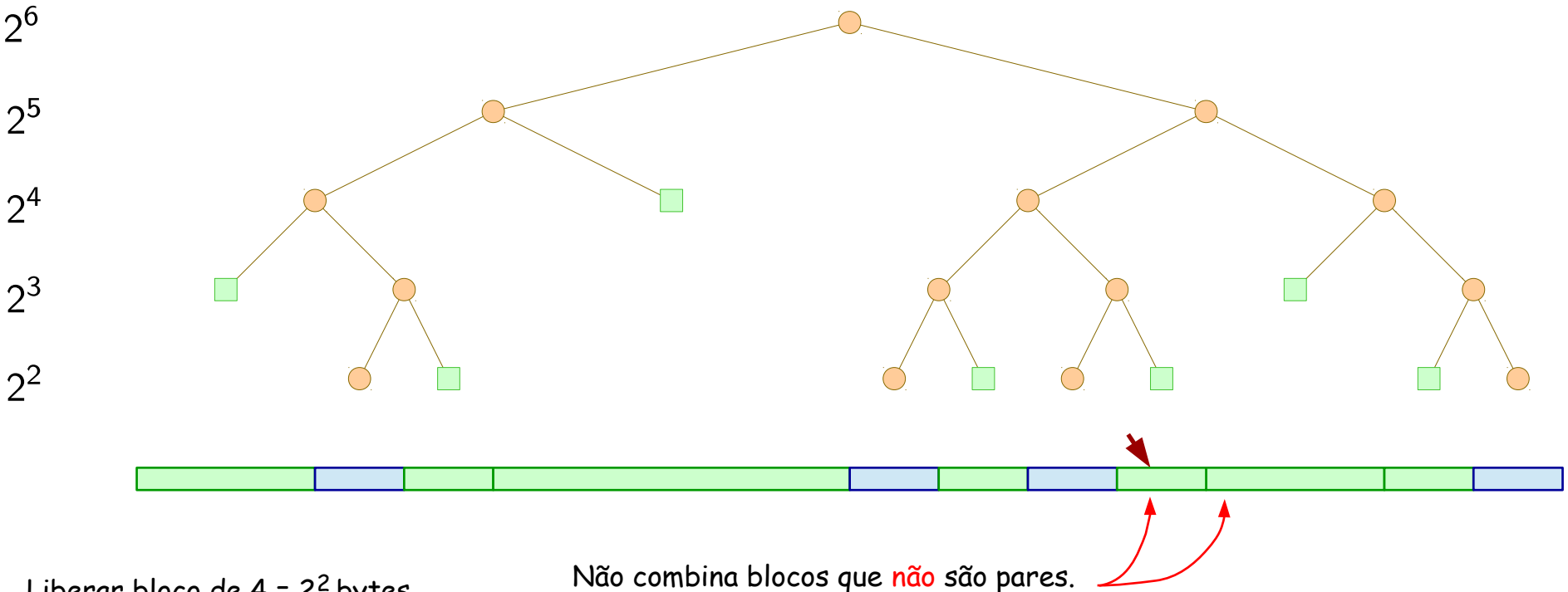


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre

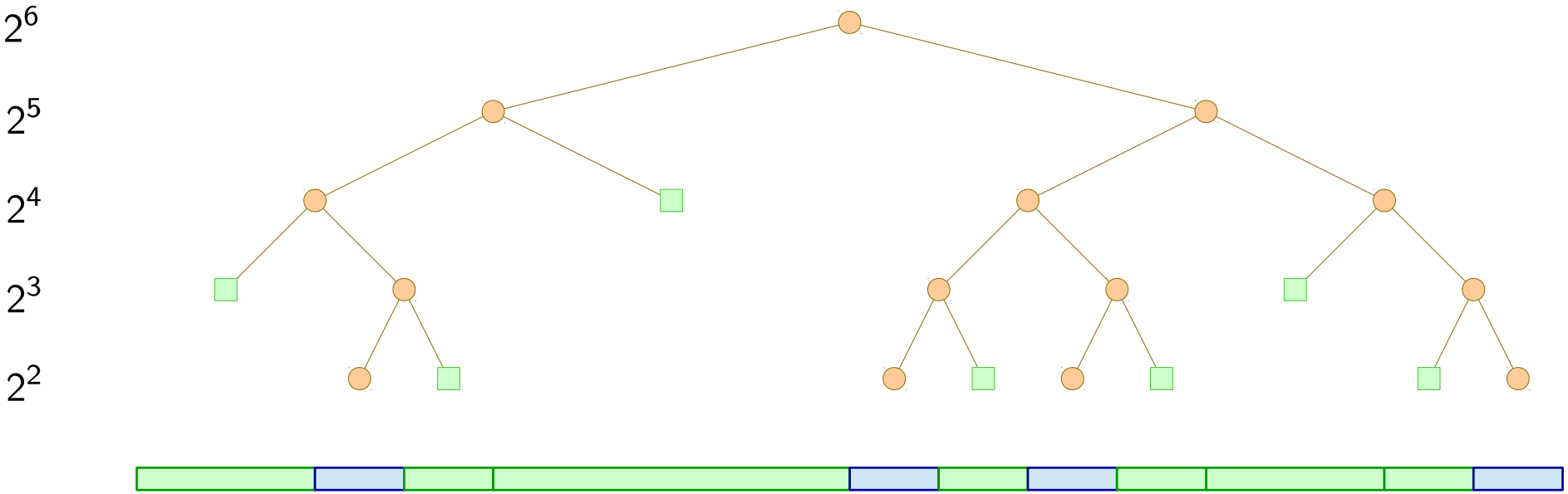
Liberando memória com sistema de pares



Liberar um bloco

1. Inserir na lista de blocos livre

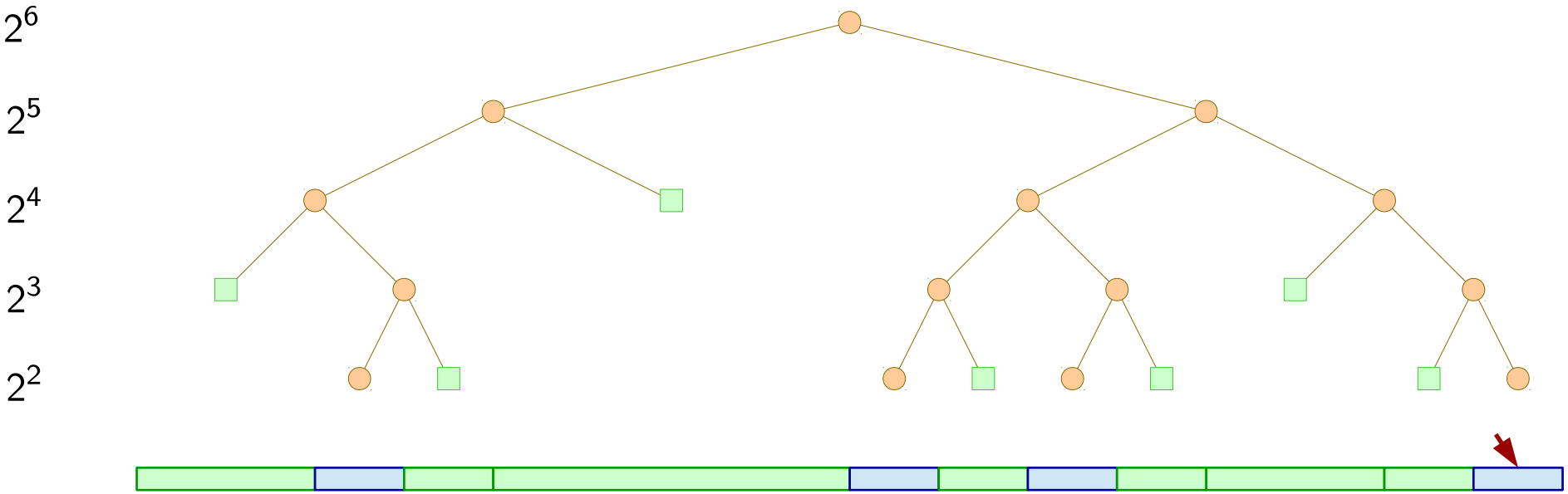
Liberando memória com sistema de pares



Liberar um bloco

1. Inserir na lista de blocos livre

Liberando memória com sistema de pares

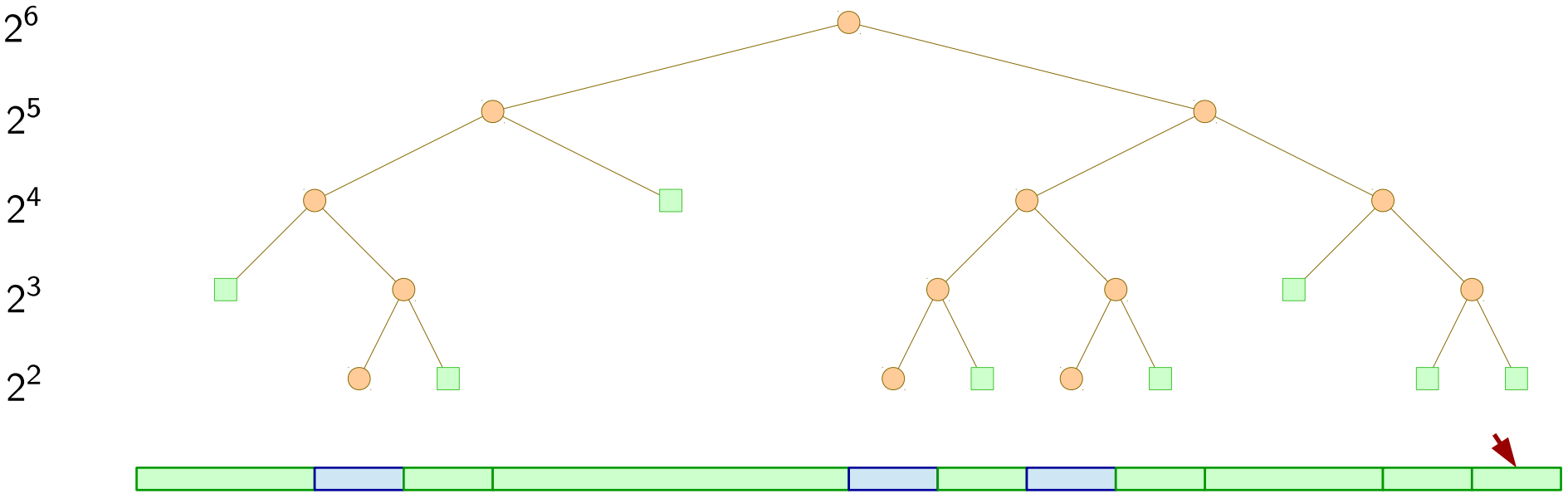


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre

Liberando memória com sistema de pares

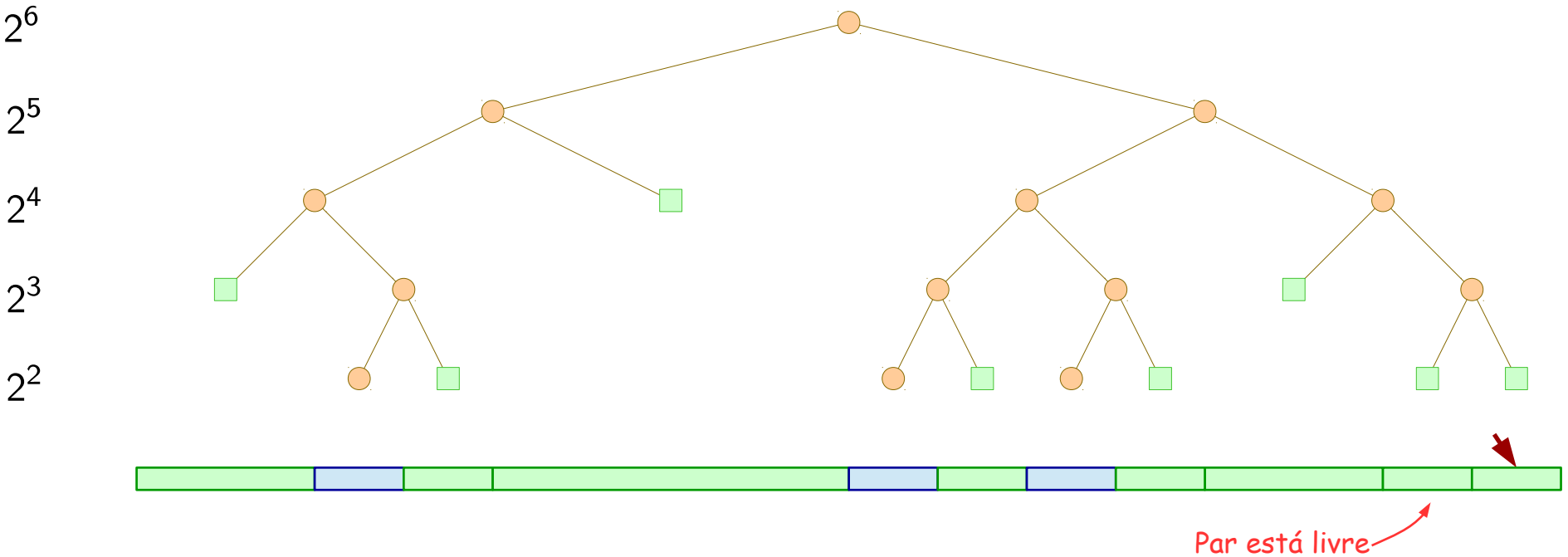


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre

Liberando memória com sistema de pares

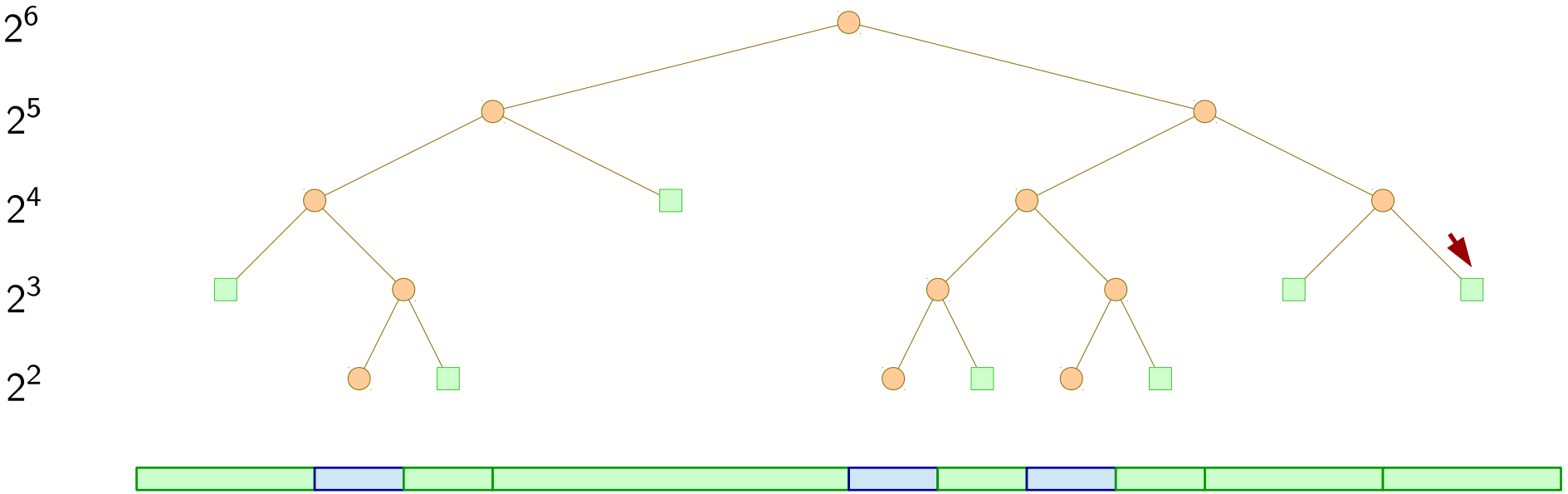


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre
2. Se o par for livre

Liberando memória com sistema de pares

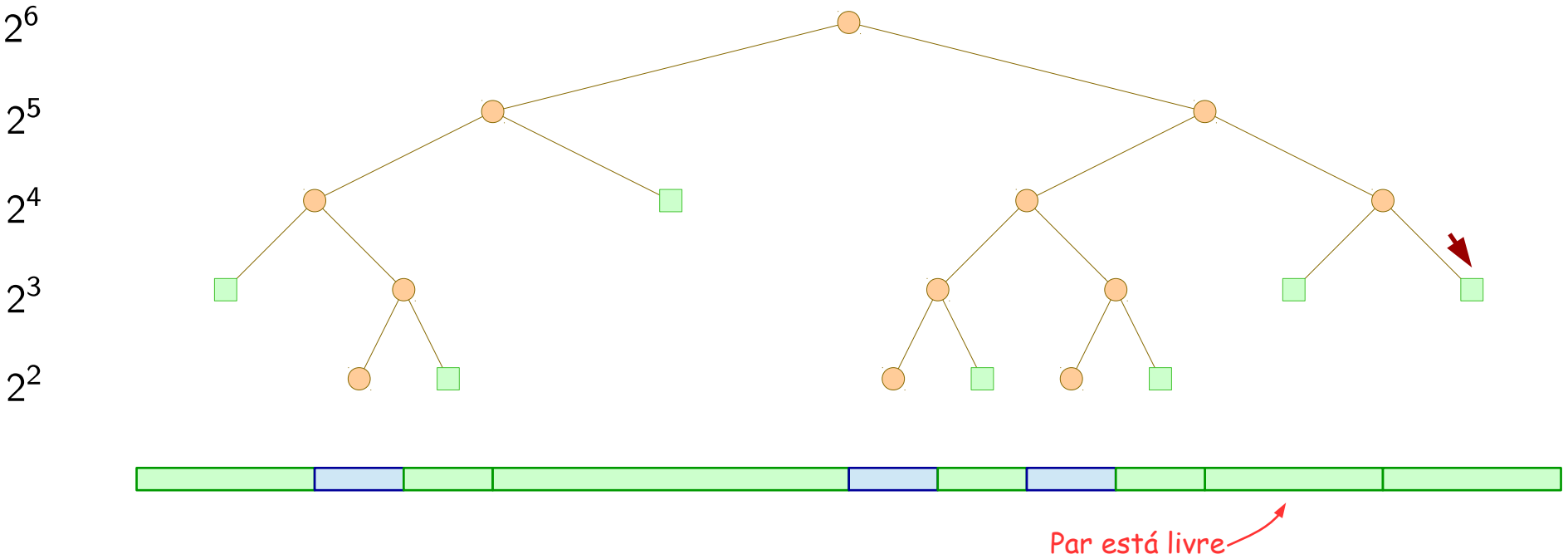


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre
2. Se o par for livre, combinar

Liberando memória com sistema de pares

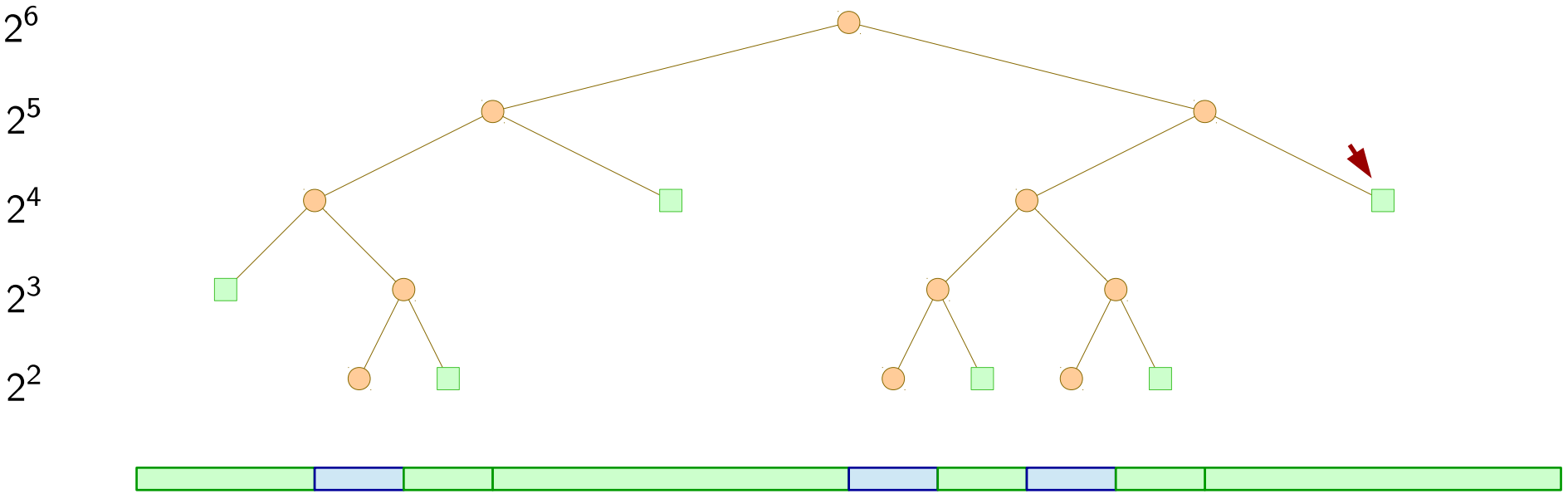


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre
2. Se o par for livre, combinar e repetir

Liberando memória com sistema de pares

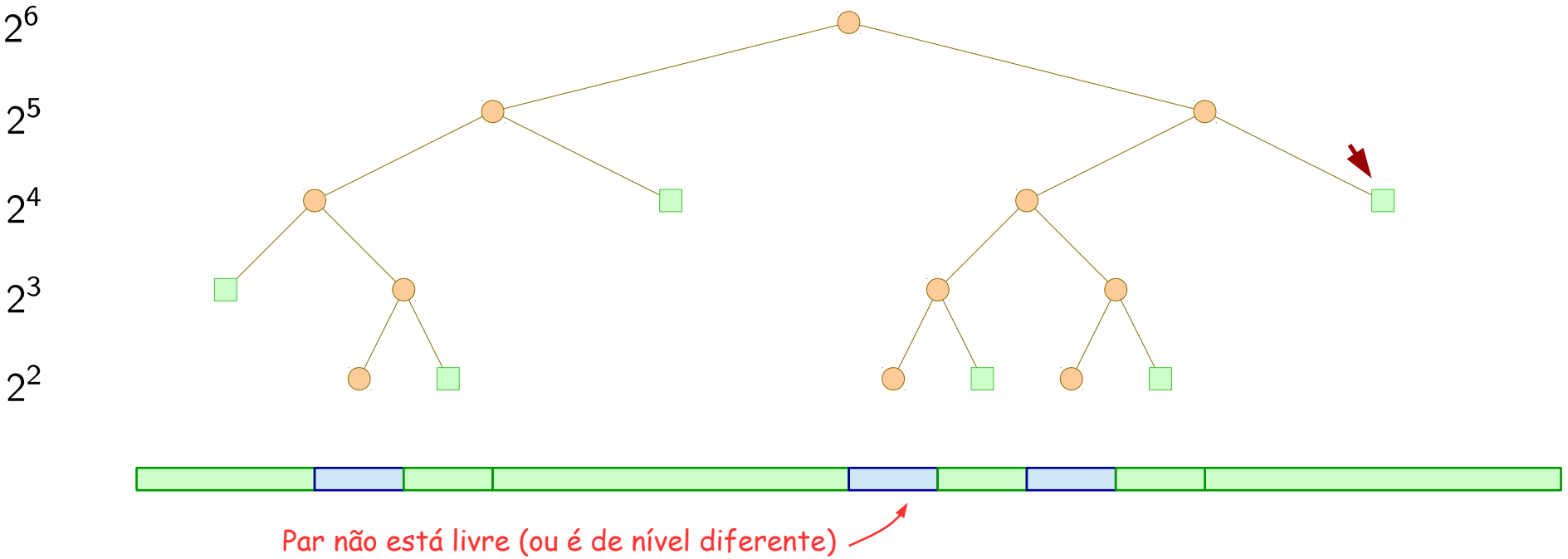


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre
2. Se o par for livre, combinar e repetir

Liberando memória com sistema de pares

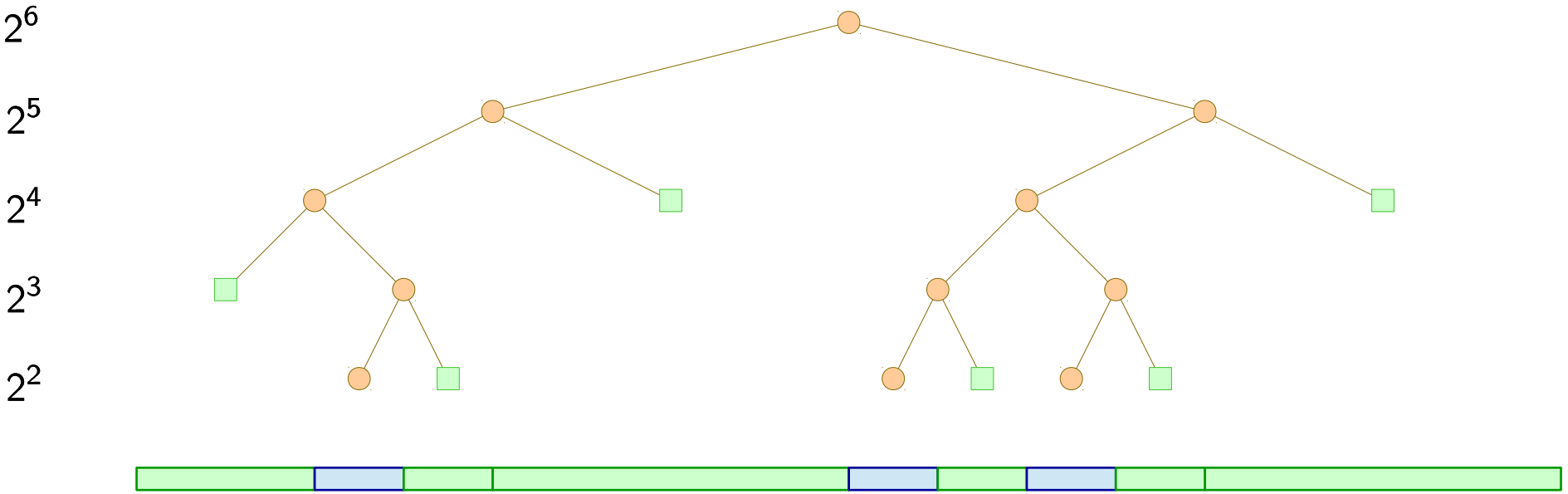


Liberar bloco de 4 = 2^2 bytes

Liberar um bloco

1. Inserir na lista de blocos livre
2. Se o par for livre, combinar e repetir

Liberando memória com sistema de pares



Liberar um bloco

1. Inserir na lista de blocos livre
2. Se o par for livre, combinar e repetir

Exercício - Sistemas de pares

1. Uma das propriedades do sistema de pares é, dado o endereço de um bloco reservado, poder encontrar o endereço de seu par rapidamente. A ideia por trás disso é que o tamanho de cada par é potência de 2, e o par de um bloco tem o mesmo tamanho que ele. Dado um bloco de endereço X e tamanho 2^k, como encontrar o endereço de seu par?
2. Considerando que o tamanho do heap é M, tente estimar o tempo máximo para alocar um nó de tamanho x; ou verificar que não há bloco livre disponível.