- Front-end
  - Análises
  - Geração RI (representação intermediária)

↓

Código Intermediário

↓

- Back-End
  - Geração de código de máquina
  - Sistema Operacional ? Conjunto de Instruções do processador?

Ambiente de Execução

- O compilador precisa auxiliar o sistema operacional a dar suporte à máquina alvo na execução das abstrações da linguagem fonte. Entre essas abstrações podemos citar:
  - Nomes;
  - Escopos;
  - Tipos de dados;
  - Procedimentos;
  - Operadores;
  - Parâmetros; e
  - Construções de Fluxo de Controle

Ambiente de Execução

- Para isso o compilador cria e gerencia um ambiente em tempo de execução no qual assume que seus programas objeto estão sendo executados.

Ambiente de Execução

- O ambiente de execução trata questões como:
  - Leiaute e alocação de endereços de memória para os objetos nomeados do programa-fonte;
  - Mecanismos de acesso as variáveis;
  - Ligações entre procedimentos;
  - Mecanismos de passagem de parâmetros;
  - Interfaces para sistema operacional, dispositivos de i/o e outros programas.

Gerenciamento de Memória

- Código: espaço definido estaticamente onde o compilador armazena o código executável.
- Estático: espaço definido estaticamente onde o compilador armazena objeto de dados de tamanho fixo, como constantes globais , e dados gerados pelo compilador, como informações para a coleta de lixo.
- Pilha: espaço definido dinamicamente, usado para armazenar Registros de Ativação, uma estrutura de dados geradas durante as chamadas de procedimento.
- Heap: espaço definido dinamicamente, utilizado para armazenar dados de longa duração que podem ser alocados e liberados durante a execução do programa.

Organização de Memória

- Para maximizar a utilização do espaço durante a execução, a pilha e o heap ocupam o restante do espaço de endereços porém cada um em uma extremidade desse espaço restante e crescendo um em direção ao outro.
- Estratégias de Alocação estática e/ou dinâmica de memória influenciam diretamente no uso da pilha e do heap. Muitos Compiladores utilizam uma combinação de alocações estáticas e dinâmicas definindo que:
  - Memória de Pilha: armazena nomes locais de um procedimento
  - Memória de Heap: armazena dados que sobrevivem a chamada do procedimento que os criou.

Pilha

- Linguagens com procedimentos, funções ou métodos gerenciam pelos menos parte de sua memória em pilha.
- Toda vez que um procedimento é chamado seus dados locais são armazenados numa pilha;
- Ao encerrar o procedimento esse espaço é removido da pilha.
- Isso permite
  - O reaproveitamento do espaço de endereços; e
  - Permite compilar código para o procedimento de modo que os endereços relativos de suas variáveis não locais sejam sempre iguais, independente da sequência de chamadas de procedimentos

Seja o seguinte esboço de programa quicksort:

```
int a[11];

void readArray() {/*lê 9 inteiros a[1]...a[9]*/

   int i;

            …

}

int partition(int m, int n) {

/* escolhe um pivô v e particiona a[m..n] de modo que a[m..p-1] sejam menores que v e a[p+1..n] sejam maiores que v. Retorna p*/

            …

}

void quicksort(int m, int n) {

            int i;

            if ( n > m) {

                        i = partition(m,n);

                        quicksort(m, i-1);

                        quicksort( i+1, n)

            }

}


main( ) {

            readArray();

            a[0]=-9999;

            a[10] = 9999;

            quicksort(1,9)

}
```

Ambientes de Execução
Gerenciamento Pilha de Execução

1. Temporários criados pelo compilador
2. Dados locais
3. Estado da máquina. Pode consistir do endereço de retorno (contador do programa) e os registradores da máquina.
4. Um elo de acesso aos dados do procedimento chamador
5. Um elo de controle, aponta para o registro de ativação do procedimento chamador
6. Espaço para o valor de retorno
7. Os parâmetros reais usados pelo procedimento. (o normal é colocá-los em registradores)

Ambientes de Execução
Gerenciamento Pilha de Execução

Registros de Ativação (*frame*)

- é um conjunto de informações armazenadas na pilha de execução toda vez que um procedimento é chamado.

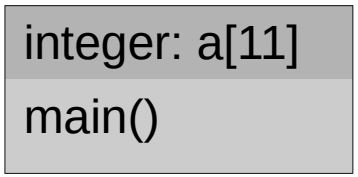
- essas informações variam de acordo com a linguagem implementada. Abaixo tem uma lista de informações que podem aparecer em algum registro de ativação:

Ambientes de Execução
Gerenciamento Pilha de Execução

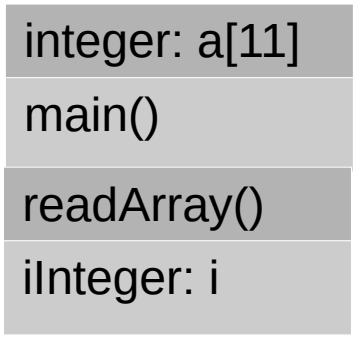
Exemplo de layout e informações armazenadas em um Registro de Ativação

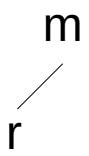
Parâmetros Reais
Valores Retornados
Elo de controle (*dynamic link*)
Elo de acesso (*static link*)
Estado de máquina salvo
Dados Locais
Temporários

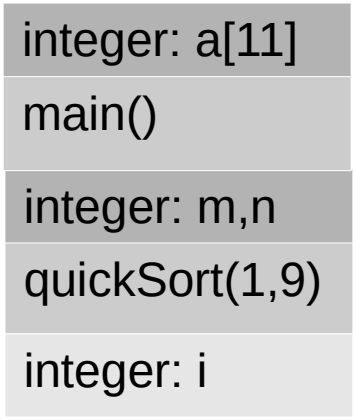
Ambientes de Execução
Gerenciamento Pilha de Execução

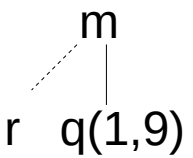


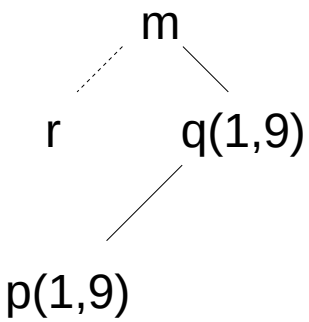
m

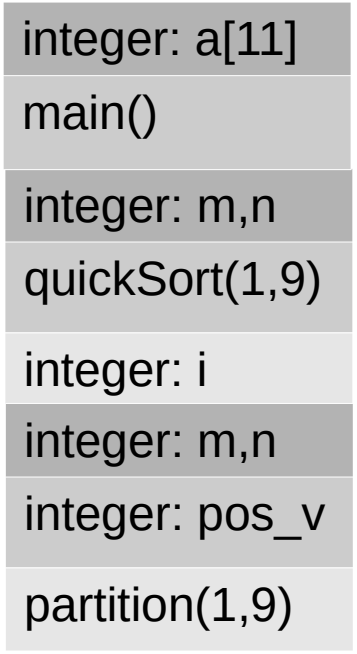


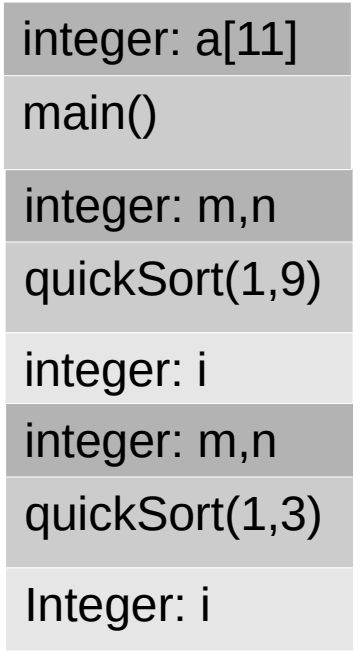


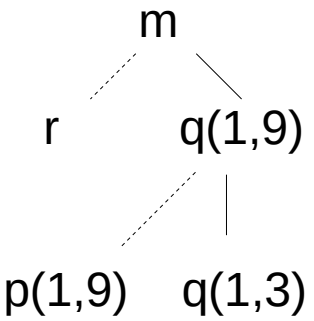












Ambientes de Execução
Gerenciamento Pilha de Execução

- Sequência de chamadas
  - Implementa as chamadas de procedimentos;
  - Consiste no código que aloca um registro de ativação na pilha de execução e entra com as informações dos campos.
- Sequência de retornos
  - Código que restaura o estado da máquina após a execução do procedimento chamado, permitindo o procedimento chamador continuar executando.

Ambientes de Execução
Gerenciamento Pilha de Execução

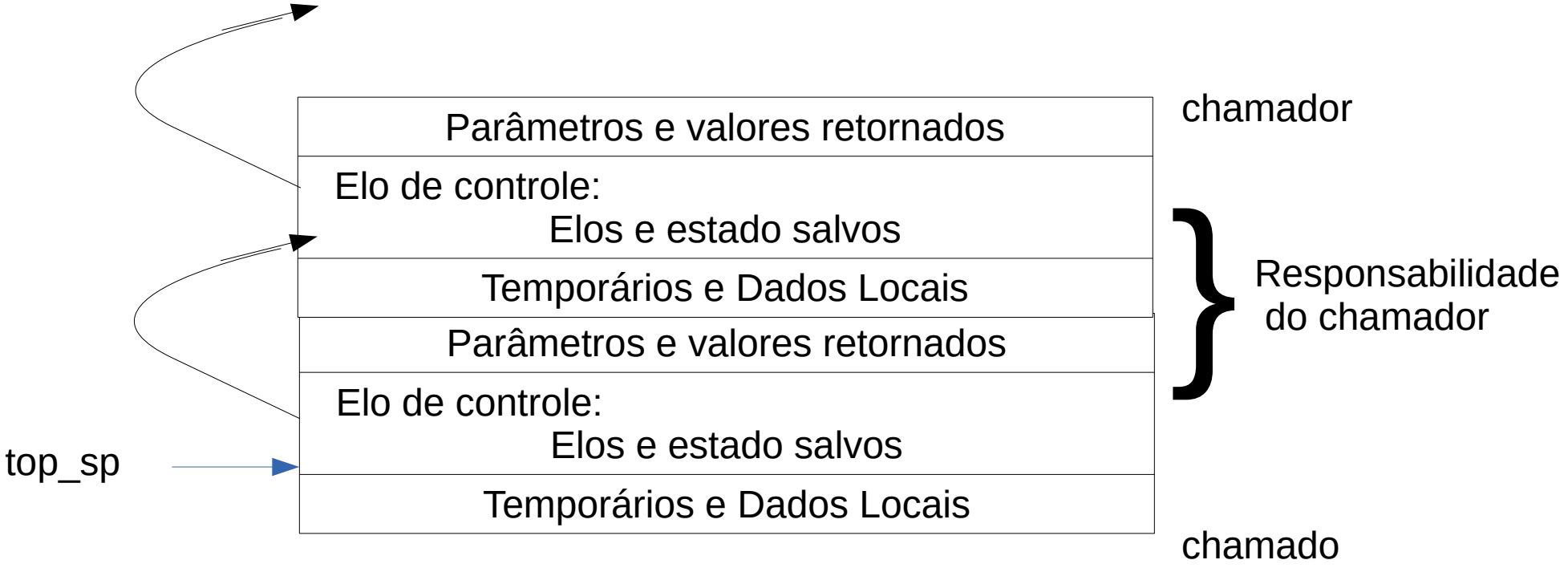
- Ao projetar a sequência de chamadas e o *layout* do registro de ativação os seguintes princípios são uteis:
  - Valores comunicados entre o procedimento chamador e o procedimento chamado são colocados geralmente no inicio do procedimento chamado
  - Os itens de tamanho fixo (elo de controle, elo de acesso e estado da máquina) geralmente são colocados no meio do registro.
  - Os itens cujo tamanho não pode ser conhecido com antecedência são colocados no fim do registro de ativação.

Ambientes de Execução
Gerenciamento Pilha de Execução

- Colocar um apontador no topo da pilha, apontando para o fim dos campos de tamanho fixo, permitindo o acesso aos dados através de deslocamentos fixos. Isso também permite o acesso aos dados de tamanho variável.

Ambientes de Execução
Gerenciamento Pilha de Execução

Um exemplo de como os procedimentos chamador e chamado podem cooperar é mostrado na figura abaixo:



Ambientes de Execução
Gerenciamento Pilha de Execução

- O registrador *top_sp* aponta o fim do campo de estado da máquina do registro de ativação do procedimento corrente no topo da pilha, assim é de responsabilidade do procedimento chamador definir o valor de *top_sp* antes de passar o controle para o procedimento chamado.

Ambientes de Execução
Gerenciamento Pilha de Execução

- A sequência de chamada e sua divisão entre o procedimento chamador e o chamado está colocada a seguir:
  - O chamador avalia os parâmetros reais;
  - O chamador armazena o endereço de retorno, valor antigo de *top_sp* no registro de ativação do procedimento chamado, incrementa top_sp (para apontar para o fim do campo com o estado de máquina)
  - O chamado salva os valores do estado da máquina
  - O chamado inicializa os dados locais e começa a execução

Ambientes de Execução
Gerenciamento Pilha de Execução

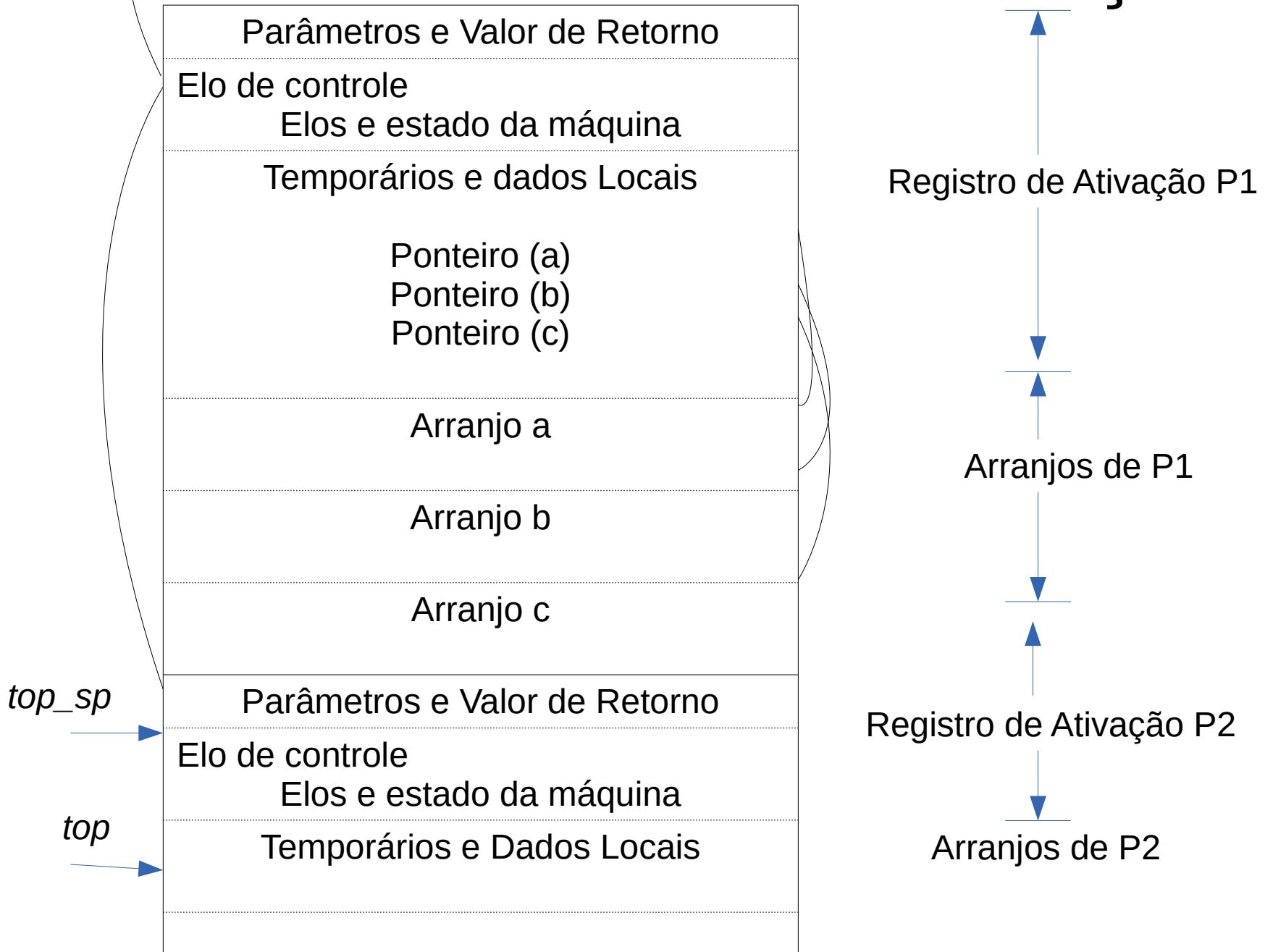
- Uma adequada sequência de retorno correspondente é:
  - O chamado preenche o valor de retorno
  - O chamado restaura o estado máquina salvo
  - O chamado restaura o valor de *top_sp*
  - O chamado desvia o fluxo de execução para o endereço de retorno

  Embora *top_sp* tenha sido decrementado, o chamador sabe o endereço dos valores de retorno.

Ambientes de Execução
Gerenciamento Pilha de Execução

- Dados de tamanho variável na pilha
  - Linguagens modernas colocam os objetos cujo tamanho não podem ser determinados em tempo de compilação no *heap*, porém os objetos locais podem ser colocados na pilha.
  - A ideia é colocar ponteiros para esses objetos no espaço de dados locais. Esses ponteiros serão alocados após o registro de ativação do procedimento que os contém, deixando assim o espaço ocupado por um procedimento na pilha maior que o espaço de um registro e variável em relação a outro procedimento.

Ambientes de Execução
Gerenciamento Pilha de Execução



Ambientes de Execução
Gerenciamento Pilha de Execução

- Acesso a dados não locais na pilha
  - Para linguagens que não permitem declarações de procedimentos aninhadas, a alocação de memória para as variáveis e o acesso a essas é simples:
    - Variáveis Globais são alocadas em uma área de memória estática, e assim o endereço é fixo e conhecido em tempo de compilação. O acesso é feito usando o endereço fixo.
    - Qualquer outra variável é local a ativação no topo da pilha e o acesso é feito através do ponteiro *top_sp*
  - Linguagens que permitem declarações aninhadas tem acesso muito mais complicado e não serão estudadas por enquanto.

Ambiente de Execução
Gerenciamento do Heap

- Gerenciamento do Heap
  - É uma porção da memória usada para dados que residem indefinidamente, ou até que o programa o exclua explicitamente.
    - Tanto C++ quanto Java disponibilizam a operação *new* para criar um objetos que são passados, ou apontados, de um procedimento para outro. Esses objetos são criados no heap.

Ambiente de Execução
Gerenciamento do Heap

- Gerenciador de Memória
  - Subsistema que aloca e libera espaço dentro do heap
  - Serve de interface entre os programas e o sistema operacional
  - Responsável por implementar a liberação de memória nas linguagens que liberam porções de memória manualmente.
  - Realiza duas funções básicas: alocação e liberação

Ambiente de Execução
Gerenciamento do Heap

- Alocação:
  - Quando um programa solicita memória para um objeto, o gerenciador de memória:
    - Disponibiliza uma porção contígua de memória heap com o tamanho solicitado;
    - Se não existir uma porção no heap que atenda a solicitação, o gerenciador tenta aumentar o tamanho do heap obtendo bytes consecutivos da memória virtual do sistema operacional. Caso os espaços estejam esgotados o gerenciador passa essa informação ao programa.

Ambiente de Execução
Gerenciamento do Heap

- Liberação:
  - O gerenciador de memória retorna o espaço liberado ao repositório de espaço livre.
  - Os gerenciadores de memórias tipicamente não retornam espaço de memória para o sistema operacional, mesmo que o uso do heap do programa diminua.

Ambiente de Execução
Gerenciamento do Heap

- Propriedades do gerenciador de memória:
  - Eficiência de Espaço:
    - Consiste em minimizar o espaço total do heap utilizado por um programa.
    - A eficiência é alcançada minimizando a fragmentação do heap.
  - Eficiência do Programa:
    - O gerenciador deve permitir que o programa execute de forma mais rápida, e isto depende de onde os objetos são colocados na memória primando pelo princípio da localidade.
  - Baixo custo: Minimizar o tempo gasto efetuando alocação e liberação de espaços

Ambiente de Execução
Gerenciamento do Heap

- Hierarquia de Memória de um Computador.
  - A limitação fundamental no tempo de acesso à memória é a tecnologia de hardware.
  - Hoje é impossível implementar uma memória grande (gigabytes) e rápida (nanosegundos).
  - Assim os processadores organizam sua memória em uma hierarquia de memória onde as menores e mais rápidas são colocadas mais próximas e as maiores e mais lentas são colocadas mais distante.

Ambiente de Execução Gerenciamento do Heap

- A memória Física e as Memórias Caches são feitas de memória RAM, porém as caches são feitas de RAM estática e a Física de RAM dinâmica.
- A RAM dinâmica é muito mais simples pois ele “esquece” seus dados
- A RAM estática precisa de um circuíto mais elaborado pois os dados podem permanecer um tempo indeterminado.

Ambiente de Execução Gerenciamento do Heap

- Os registradores são escassos e seu uso é moldado para cada aplicação de acordo como o compilador gera.
- Os outros níveis são gerenciados automaticamente permitindo o funcionamento do programa em qualquer hierarquia de memória.
- A cada acesso à memória a máquina pesquisa cada nível de memória a partir dos menores para os maiores até localizar os dados
- As memórias caches são gerenciadas pelo hardware, e como os discos são lentos a memória virtual é gerenciada pelo sistema operacional.

Ambiente de Execução
Gerenciamento do Heap

- Os dados são transferidos em blocos de memória contígua.
  - Blocos Maiores são transferidos pelo disco para amortizar o tempo
  - Entre o disco e a memória física são transferidos em páginas (4-64 KB)
  - Entre memória física e as caches são transferidas linhas de caches (32-256 KB)

Ambiente de Execução
Gerenciamento do Heap

- Localidade em Programas
  - Significa que o programa passa a maior parte do tempo executando uma fração relativamente pequena do código, usando uma pequena porção de dados.
  - **Localidade Temporal**: quando os endereços de memória acessados sejam novamente acessados dentro de um curto intervalo de tempo.
  - **Localidade Espacial**: quando os endereços de me mórias vizinhos do endereço acessado também sejam acessados dentro de um curto período de tempo.

Ambiente de Execução Gerenciamento do Heap

- Sabidamente programas gastam 90% do tempo executando 10% do código. Os motivos são:
  - Muitas instruções nunca são executadas, pois apenas uma pequena parte de bibliotecas e componentes é utilizada. Sistemas legados carregam muitas funcionalidades fora de uso.
  - Instruções que assegurem a corretude das entradas de dados, apesar de essenciais, quase nunca são executadas
  - O Programa típico gasta a maior parte do tempo executando laços mais internos e ciclos recursivos de um programa

Ambiente de Execução
Gerenciamento do Heap

- A Localidade nos permite tirar proveito da hierarquia de memória de um computador moderno
  - Colocar a maioria das instruções e dados comuns na memória rápida, porém
  - programas comuns costumam variar mais os dados do que as instruções, então manter os dados mais recentemente na hierarquia mais rápida ajuda muito. (Não funciona para programas com uso intenso de dados – aqueles que percorrem vetores grandes em ciclos.)

Ambiente de Execução Gerenciamento do Heap

- Reduzindo a Fragmentação
- Quando um programa começa a executar seu heap é um único bloco contíguo de memória, e
  - Depois de muitas alocações e liberações o heap passa a ter porções de espaços liberados entre espaços alocados, essas porções de espaços liberados são chamados de buracos e são disponibilizados como espaço livre.
  - Porém ao precisar de um grande espaço alocado pode acontecer de não existir um buraco com tamanho suficiente.
  - Assim o gerenciador de memória precisa evitar fragmentação.

Ambiente de Execução
Gerenciamento do Heap

- Posicionamento de objetos
  - First-fit : tenta alocar no primeiro buraco onde caiba o objeto.
  - Best-fit: tenta alocar no menor buraco onde caiba o objeto.
  - Next-fit: tenta aproveitar o principio de localidade temporal e então tentar alocar o objeto no buraco dividido mais recentemente (onde foi feita a alocação anterior)

Ambiente de Execução
Gerenciamento do Heap

- Uma estratégia muito utilizada quando se usa best-fit para alocação no heap é separar os espaços livre em compartimentos de acordo com seu tamanho, facilitando encontrar o melhor best-fit

Ambiente de Execução
Gerenciamento do Heap

- Gerenciando e unindo espaço livre
  - Quando um objeto é liberado manualmente, o gerenciador de memória deve tornar a porção livre de modo que possa ser alocada.
  - Se os espaços livres estão compartimentados basta colocar a porção na lista de espaços livres segundo seu tamanho, e com uma estratégia simples de mapa de bits é possível encontrar espaços maiores quando necessário, unindo pequenos compartimentos.

Ambiente de Execução
Gerenciamento do Heap

- Tudo se torna mais complexo quando o heap é gerenciado sem compartimentos e as porções liberadas são unidas aos buracos adjacentes.
- Duas estruturas de dados auxiliam nessa tarefa:
  - Rótulos de Fronteira
  - Lista Duplamente Encadeada de porções livres e embutidas.

Ambiente de Execução
Gerenciamento do Heap

- Rótulos de Fronteira: no inicio em fim de cada bloco de espaço livre ou alocado mantemos :
  - Um bit livre/alocado
  - E um contador do número total de bytes no bloco.
- Lista duplamente encadeada de porções livres ou embutidas: os ponteiros para a lista são colocados adjacentes aos rótulos de fronteira, porém a ordem das porções na lista não é especificada. Por exemplo, a lista poderia ser ordenada por tamanho facilitando o best-fit.

Ambiente de Execução
Gerenciamento do Heap

Porção A							Porção B							Porção C						
0	200	↓		↓	200	0	1	100	↓		↓	100	1	1	120	↓		↓	120	1

Liberando Porção B

Porção A							Porção B							Porção C						
0	200	↓		↓	200	0	0	100	↓		↓	100	0	1	120	↓		↓	120	1

Unindo Porção A e Porção B

Porção A							Porção C						
0	320	↓		↓	320	0	1	120	↓		↓	120	1

Ambiente de Execução

- Problemas com a Liberação Manual
  - Linguagens com C e C++ exigem que o programador cuide da liberação de de dados.
  - O ideal é que se remova os endereços que não serão mais acessados e que sejam mantidos os endereços que possam ser referenciados. Quando isto não acontece dois erros podem ocorrer:
    - Memory-Leak (Vazamento de Memória): quando dados que não serão mais acessados não são liberados; e
    - Dangling-Pointer (Ponteiro pendente): quando um endereço referenciado foi liberado
  - A coleta de lixo automática se livra dos vazamentos de memória

Ambiente de Execução
Coleta de Lixo

- **Coleta de Lixo**

  Lixo: dados que não podem ser referenciados.

  Muitas linguagens fazem coleta de lixo, citamos

  - Lisp (desde 1958), Java, Perl, ML, Modula-3, Prolog e Smaltalk

- **Objetivo**: requerer porções de memória contendo objetos que não podem mais ser acessados por um programa.

Ambiente de Execução
Coleta de Lixo

- O requisito básico de uma linguagem para ter um coletor de lixo é ser **segura quanto ao tipo**.
- Uma linguagem segura quanto ao tipo é aquela cujo o tipo de qualquer objeto pode ser determinado tanto em tempo de compilação (ML) como em tempo de execução(Java).
- A partir da informação de tipo de um objeto podemos determinar o tamanho do objeto, e qual dos seus componentes contém referencias a outros objetos(apontadores).

Ambiente de Execução
Coleta de Lixo

- Linguagens como C e C++ são linguagens inseguras quanto ao tipo, pois seus ponteiros podem ser manipulados de forma aleatória.
- um coletor de lixo teoricamente inseguro funciona bem para a maioria dos programas C e C++ que não efetuam essa manipulação dos ponteiros.
- A coleta de lixo é muito dispendiosa e por isso não foi adotada na maioria das linguagens

Ambiente de Execução
Coleta de Lixo

- Alcançabilidade
  - Conjunto Raiz: são todos os dados que podem ser acessados diretamente por um programa, sem ter de seguir qualquer apontador.
  - Um programa pode alcançar qualquer membro do seu conjunto raiz a qualquer momento. Recursivamente, qualquer objeto alcançado através de uma referência feita por um membro do conjunto raiz é um objeto alcançável também
  - A alcançabilidade torna-se mais complexa quando o programa é otimizado pelo compilador. Pois o compilador manipula endereços de memórias em registradores

Ambiente de Execução
Coleta de Lixo

- O conjunto de objetos alcançáveis muda durante a execução do programa. As operações abaixo modificam tal conjunto:
  - Alocações de Objetos (aumenta)
  - Passagem de parâmetros e valores de retorno (propagam)
  - Atribuições referências (terminar alcançabilidade)
  - Retornos de Procedimento

    (terminar alcançabilidade)

Ambiente de Execução
Coleta de Lixo

- Existem duas maneiras básicas para recuperar objetos inalcançáveis.
  - A primeira é capturar o objeto assim que deixa de ser alcançável, e
  - a outra é localizando periodicamente os objetos inalcançáveis.
- O que dá origem a duas técnicas de coleta de lixo
  - Coletores de Lixo por Contagem de Referência
  - Coletores de Lixo baseados em Rastreamento

Ambiente de Execução
Coleta de Lixo

- Coletores de Lixo por Contagem de Referência
  - Todo objeto tem um campo para o contador de referência; e
  - São administrados da seguinte maneira:
    - Alocação do Objeto: o contador passa a valer 1;
    - Passagem de Parâmetro : o contador é incrementado;
    - Atribuições a referência : no comando u = v, onde u e v são referências, o contador do objeto v aumenta e do objeto u diminui uma unidade;

Ambiente de Execução
Coleta de Lixo

- Retorno de procedimento: todas as referências mantidas pelas variáveis locais devem ser decrementadas (o mesmo número de vezes que são referenciadas pelas variáveis locais)
- Perda transitiva de alcançabilidade: sempre que o contador de referência de um objeto chega a zero, toda referencia contida naquele objeto deve ser decrementada.

- A contagem de referência tem desvantagens:
  - Não pode coletar estruturas cíclicas; e
  - É dispendiosa
- A contagem de referência tem por vantagem ser feita de forma incremental.

(Vide Livro para os porquês)

Ambiente de Execução
Coleta de Lixo

- Coleta de Lixo Baseada em Rastreamento
  - Este coletor de lixo é executado periodicamente, geralmente quando não há mais espaços livres ou quando a quantidade destes cai abaixo de um determinado patamar.
  - Alguns algoritmos de coletos baseadas em rastreamento são:
  - Marcar-e-varrer (“*mark- and – sweep*”) (básico)
  - Marcar-e-compactar
  - Coletores de Cópias

Ambiente de Execução
Coleta de Lixo

- Coletor Marcar-e-Varrer Básico
  - O algoritmo visita todos os objetos alcançáveis, a partir do conjunto raiz, e os marca como alcançáveis;
  - Depois percorre todo o heap procurando por objetos, e aqueles que não estiverem marcados como alcançáveis são inseridos na lista de espaço livre.

Ambiente de Execução
Coleta de Lixo

- Abstração Básica
  - Os espaços do heap podem assumir 4 estados:
    - *Free*: espaços pronto para serem alocados - livres
    - *Unreached*: espaço não alcançado (ainda). Sempre que um espaço é alocado é colocado neste estado.
    - *Unscanned*: é um espaço alcançável porém seus apontadores ainda não foram escaneados
    - *Scanned*: é um espaço alcançável cujos apontadores já foram escaneados.

Ambiente de Execução
Coleta de Lixo

- Um ciclo de coleta de lixo baseada em rastreamento estabelece as seguintes mudanças de estados dos espaços:
  - *Free* → *Unreached* : antes do rastreamento
  - *Unreached* → *Unscanned*: quando alcançado a partir do conjunto raíz
  - *Unscanned* → *Scanned*: quando seus ponteiros forem escaneados.
  - *Scanned* → *Unreached*: ao iniciar coleta

Ambiente de Execução
Coleta de Lixo

- Os espaços ainda mudam seus estados através de ações do programa:
  - *Free* → *Unreached* : na alocação de espaço
  - *Unreached* → *Free*: na liberação de espaço
- Com estas abstrações Baker aplicou otimizações sobre o algoritmo Marcar-e-Varrer tornando-o menos dispendioso. (Vide Livro)

Ambiente de Execução
Coleta de Lixo

- Coletores de Lixo Marcar-e-Compactar
  - É um coletor de relocação, move os objetos alcançáveis pelo heap a fim de eliminar a fragmentação.
  - Ou seja, depois de identificar todos os objetos alcançáveis, coloca a todos em uma área contígua do heap.
  - Isto implica em mudar suas referência no conjunto raiz.

Ambiente de Execução
Coleta de Lixo

- O algoritmo marcar-e-compactar tem 3 fases:
  - A primeira fase é a fase de marcação semelhante a fase do algoritmo marcar-e-varrer;
  - Na segunda o algoritmo percorre a porção alocada e define um novo endereço para cada um dos objetos alcançáveis; e
  - Finalmente, o algoritmo copia os objetos para seu novo endereço atualizando todas suas referências.
- Esse algoritmo compacta os objetos imediatamente.

Ambiente de Execução
Coleta de Lixo

- Coletores de Cópia
  - Também é um coletor de relocação
  - Reserva antecipadamente o espaço para onde os objetos podem ser movidos
    - Não precisa então procurar espaços livres
    - O espaço da memória é subdividido em dois subespaços:
      - Um subespaço A para as alocações feitas pelo programa
      - Um subespaço B reservado para a cópia
  - Quando o subespaço A enche, o coletor copia os objetos para o subespaço B e inverte os papéis dos subespaços.

Ambiente de Execução
Coleta de Lixo

- Comparando Custos
  - Marcar-e-Varrer Básico: proporcional ao nº de porções no heap
  - Marcar-e-Varrer Baker: proporcional ao nº de objetos alcançáveis
  - Marca-e-Compactar: proporcional ao nº de porções no heap + o tamanho dos objetos alcançáveis
  - Cópia de Cheney(Vide Livro) : proporcional ao tamanho dos objetos alcançáveis

Ambiente de Execução
Coleta de Lixo

- Coleta de Lixo com Pausa Curta
  - Coletores baseados em rastreamento fazem com que a execução do programa pare e ás vezes longas pausas são necessárias
  - Coletas com Pausas Curtas dividem a tarefa de coletar lixo.
    - Coleta Incremental : divide o trabalho no tempo em coleta de lixo e mutação
    - Coleta Parcial: divide o trabalho no espaço em subconjuntos de lixo.
  - As ideias a partir dessas coletas de pausa curta podem ser adaptadas para criar um coletor em paralelo em um multiprocessador

Ambiente de Execução
Coleta de Lixo

- Coleta de Lixo Incremental
  - Intercala execução da coleta de lixo e do programa (mudador)
  - Primeiro processam o conjunto raiz atomicamente, sem interferência do mudador; e determinam o conjunto inicial de objetos Unscanned;
  - Alterna ações do mudador e passos de rastreamento;
    - Durante esse período as alterações efetuadas pelo mudador são armazenadas em uma lista para posterior atualização quando voltar a execução.

Ambiente de Execução
Coleta de Lixo

- Precisão na Coleta Incremental
  - Uma vez que um objeto deixou de ser alcançavel não volta a ser alcançável;
  - E o conjunto de objetos alcançáveis, durante a coleta de lixo e mutação :
    - Ou cresce, quando um novo objeto é alocado;
    - Ou decresce, quando uma referência é perdida

Ambiente de Execução
Coleta de Lixo

- Assim temos os conjuntos:
  - R : conjuntos de objetos alcançáveis no inicio da coleta
  - New : conjunto de objetos alocados durante passo de rastreamento
  - Lost: conjunto de objetos desreferenciados durante passo de rastreamento
- Por ser caro restabelecer a alcançabilidade cada vez que um objeto é desreferenciado, o coletor incremental não tenta coletar todo o lixo no fim do rastreamento, deixando o lixo flutuante com um subconjunto de Lost.

Ambiente de Execução
Coleta de Lixo

- Rastreamento Parcial Simples
  - Para tratar as alterações feitas pelo mudador um rastreamento incremental faz:
    - Todas as referências que existiam antes da coleta são preservadas
    - Todos os objetos criados são considerados alcançáveis imediatamente e são colocados no estado Unscanned.

Esta implementação é cara pois o algoritmo tem que interceptar todas as operações de escrita e lembrar-se de todas as referências modificadas.

Ambiente de Execução
Coleta de Lixo

- Análise de Alcançabilidade Incremental
  - Ao intercalar mudador e com o algoritmo básico de rastreamento pode acontecer que um invariante do rastreamento pode ser violado por uma ação do mudador, pois
    - Um objeto *scanned* só pode referenciar outro objeto *scanned* ou *unscanned* e num um objeto *unreached*
  - A chave para um rastreamento incremental correto é observar todas as cópias de referências a objetos correntemente não alcançados a partir de um objeto não inspecionado para um que já foi inspecionado.

Ambiente de Execução
Coleta de Lixo

- Para evitar transferência problemáticas de referências o algoritmo pode modificar a ação do mudador de uma das seguintes maneiras:
  - Barreira de Escrita: intercepta as escritas de referências para um objeto *scanned* o_1 quando a referência é para um objeto *unreached* ***o***. Neste caso o objeto ***o*** volta a ser *unscanned.* Uma alternativa é colocar o objeto o_1 com *unscanned*.
  - Barreira de Leitura: intercepta as leituras de referências a objetos *unreached* ou *unscanned*. Neste caso sempre que ocorrer a leitura de um objeto ***o*** a uma referência nesses estados, o objeto ***o*** deve ser colocado como *unscanned*.

Ambiente de Execução
Coleta de Lixo

- Barreira de Transferência: intercepta a perda de referência original em um objeto *unreached* ou *unscanned*. Sempre que o mudador modificar a referência em um objeto *unreached* ou *unscanned*, salve a referência sendo modificada e classifique-a como alcançavel e coloque-a no estado *unscanned*

Barreiras de Escritas são mais eficientes

- Barreiras de Leituras são mais caras
- Barreiras de Transferências não são competitivas

(Vide Livra para os porquês)

Ambiente de Execução
Coleta de Lixo

- Coleta Parcial
  - Observações:
    - Usualmente de 80% a 98% dos todos objetos recém-alocados morrem cedo;
    - Normalmente um objeto que resiste a uma coleta, resistirá por muitas outras.
  - Dois algoritmos de coleta parcial:
    - Coleta Generativa: atua mais frequentemente na área do heap que contém os objetos mais novos.
    - Algoritmo do Trem: não gasta uma grande fração de tempo com objetos novos, mas limita as pausas em razão da coleta de lixo

Ambiente de Execução
Coleta de Lixo

- Assim uma boa combinação de estratégias é usar a coleta generativa para objetos novos e quando um objeto tornar-se maduro, “promovê-lo” para um heap separado que é gerenciado pelo algoritmo do trem.

Ambiente de Execução
Coleta de Lixo

- Coleta Parcial trabalha com os seguintes conjuntos de objetos:
  - Conjunto Destino: conjunto de objetos a serem coletados em uma rodada de coleta parcial
  - Conjunto Estável: todos os demais objetos que não estão no conjunto destino
  - Conjunto de Lembrados: objetos estáveis que referenciam um objeto destino por uma ação do mudador.

Ambiente de Execução
Coleta de Lixo

- Coleta de Lixo Generativa
  - O heap é dividido em uma série de partições que são numeradas de 0, 1, até n, onde as partições com menor número contém objetos mais recentes.
  - Assim os objetos criados são colocados na partição 0, quando esta se enche, o seu lixo é coletado e os objetos alcançáveis são colocados na partição 1.
  - A partição 0 torna-se vazia e os objetos são alocados ali novamente, quando se encher novamente a coleta é executa e os objetos colocados alcançáveis colocados na partição 1 juntamente com os objetos que lá estavam

Ambiente de Execução
Coleta de Lixo

- Esse padrão se repete até que a partição 1 fique cheia quando a coleta de lixo será aplicada nas partições 0 e 1.

De uma maneira geral as partições coletadas estão sempre abaixo de uma determina partição i, e as partições de 0 a i formam o conjunto de destino. As partições acima de i formam o conjunto estável.

É preciso lembrar apenas as referencias que objetos estáveis fazem a objetos mais novos para fazer apenas a coleta desses e não todos os objetos estaveis, o que tornaria a coleta muito mais cara.

Ambiente de Execução
Coleta de Lixo

- Algoritmo do Trem
  - Utiliza partições de tamanho fixo chamadas de carro.
  - Os carros são organizados em trens.
  - Há uma ordenação lexicográfica dos trens e seus carros, ou seja são ordenados primeiro pelo número do trem e depois pelo número do carro dentro do trem.
  - Cada carro tem um conjunto de lembrados que são as referencias feitas a seus objetos com origem
    - Em carros de número mais alto no mesmo trem; ou
    - Em trens de números mais altos
  - E cada trem tem uma lista de lembrados com origem em trens com números mais altos.

Ambiente de Execução
Coleta de Lixo

- O núcleo do algoritmo consiste em como coletamos o primeiro carro do primeiro trem durante uma rodada completa de coleta.
  - O conjunto alcançável do carro é formado pelos objetos com referências a partir do conjunto raíz e aqueles com referencia no conjunto lembrado
  - Esses objetos são inspecionados como no algoritmo Marcar-e-varrer, mas somente os objetos no carro(internos)

Ambiente de Execução
Coleta de Lixo

- Os objetos alcançáveis são então movidos para outro lugar segundo as seguintes regras:
  - Se há uma referencia ao objeto no conjunto de lembrados de qualquer outro trem o objeto é movido de preferência para um dos trens de onde vem a referencia e se possível para o mesmo carro, se não hover espaço para um novo último carro;
  - Se não há uma referência de outro trem mas há referencia ou do conjunto raiz ou de outro carro no mesmo trem o objeto é movido, de preferência, para o carro de onde vem a referencia senão para o último carro (ou um novo último carro se não houver espaço)
- Depois de mover todos os objetos do primeiro carro este pode ser excluído.

Ambiente de Execução
Coleta de Lixo

- O objetivo é extrair do primeiro trem todos os objetos que não são lixo cíclico e desta forma ele pode ser coletado na próxima rodada da coleta
- Portanto precisamos criar novos trens, mesmo não havendo limite de carros em um trem.