Modelagem física

Baseado em Arndt von Staa

Especificação

- Objetivo dessa aula
  - Apresentar os conceitos de modelagem conceitual e física
  - Apresentar uma linguagem gráfica para a modelagem física de estruturas de dados
  - Apresentar conceitos complementares que visam assegurar qualidade
- Referência básica:
  - Capítulo 9.4
- Referência complementar
  - Silva, R.P.; *UML2 em Modelagem Orientada a Objetos*; Florianópolis, SC: Visual Books; 2007
  - Wells, D.; *Extreme Programming: A Gentle Introduction*; 2006; Buscado em: 06/ fevereiro/2008; URL: http://www.extremeprogramming.org/index.html

Sumário

Como eliminar causas de retrabalho inútil?

- Vamos nos fixar na causas de retrabalho inútil:
  - desenvolver algo e descobrir que não era isso que se queria ou que se precisava
- O que fazer para reduzir ou evitar de vez esse risco?
  - produzir uma boa especificação do que se quer que seja feito
  - produzir uma arquitetura – organização da solução – adequada ao problema a resolver
    - modelar o problema a resolver ➔ modelagem conceitual
    - modelar a solução ➔ modelagem física
  - especificar detalhes da implementação
    - especificação de funções, assertivas

O que é um modelo?

- Um modelo é uma abstração a partir da qual podem-se avaliar, de forma racional, as propriedades ou o comportamento futuro de reificações em conformidade com o modelo
  - Reificação: no processo de abstração, o momento em que a característica de ser uma "coisa" se torna parte da realidade objetiva. [Aurélio]
- Exemplo:
  - $e = \frac{1}{2} at^2 + v_0 t + d_0$
    - é um modelo que descreve o espaço percorrido por um corpo em um movimento uniformemente acelerado em condições ideais
      - sem atrito de contato
      - sem atrito do ar (ou outro fluido)
      - aceleração constante

Modelo conceitual

- Qualquer modelo deve descrever o conjunto de todas as instâncias possíveis do conceito modelado
- Modelo conceitual
  - descreve um conceito sem se preocupar com a forma de sua implementação
- Exemplo
  - um grafo dirigido é uma tupla < **V** , **A** , **O** > formada por
    - um conjunto de vértices **V**,
    - um conjunto de arestas dirigidas **A** e
    - um conjunto de origens **O** $\subseteq$ **V**.
  - cada aresta **a** $\in$ **A** é um par < $\mathbf{v_1}$, $\mathbf{v_2}$ > em que $\mathbf{v_1}$, $\mathbf{v_2}$ $\in$ **V** .
    - a direção da aresta é de $\mathbf{v_1}$ para $\mathbf{v_2}$.

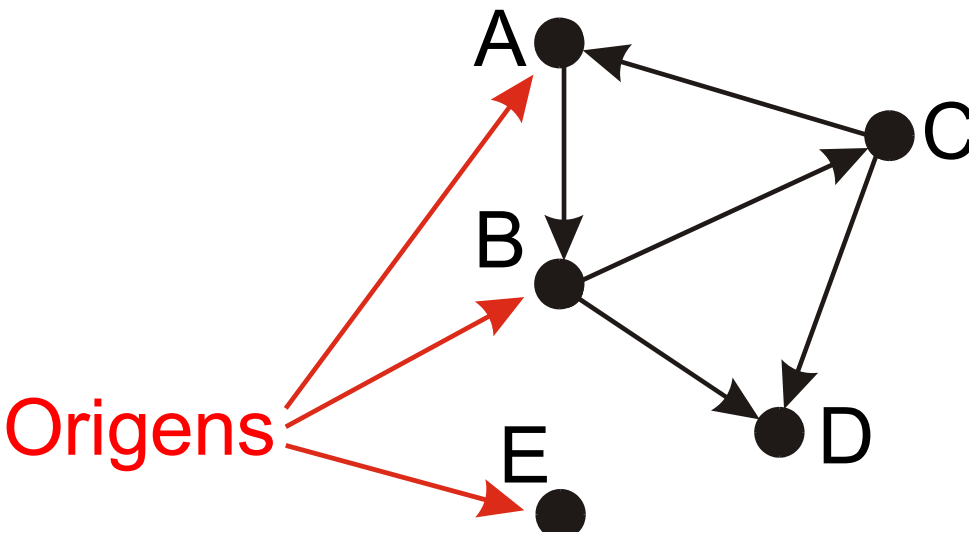
Exemplo conceitual

- Um exemplo conceitual é uma das possíveis instâncias de um modelo conceitual
  - dito de outra forma, um exemplo conceitual é um conjunto de elementos concretos que satisfaz o modelo conceitual
- Exemplo de um grafo
  - **V** = { A , B , C , D , E }
  - **A** = { <A,B> , <B,C> , <B,D> , <C,A> , <C,D> }
  - **O** = { A , B , E }

- Entenderam o exemplo?
  - o grafo é conexo?
  - a partir das origens podem-se alcançar todos os vértices do grafo?

Representação gráfica do exemplo conceitual

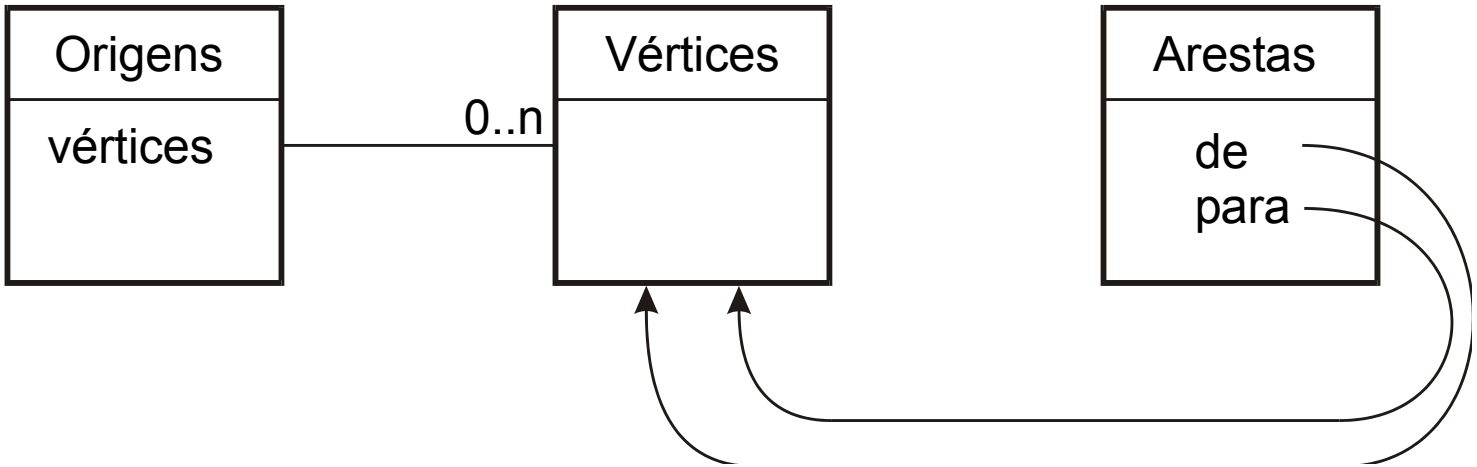
- Figuras valem por mil palavras
  - uma representação gráfica é muito melhor para a compreensão humana
    - para exemplos utilizam-se em geral representações (ad hoc) que mais nos convierem
  - a forma simbólica matemática tende a ser melhor para verificar através de algum algoritmo se o exemplo satisfaz propriedades desejadas



Representação gráfica do modelo

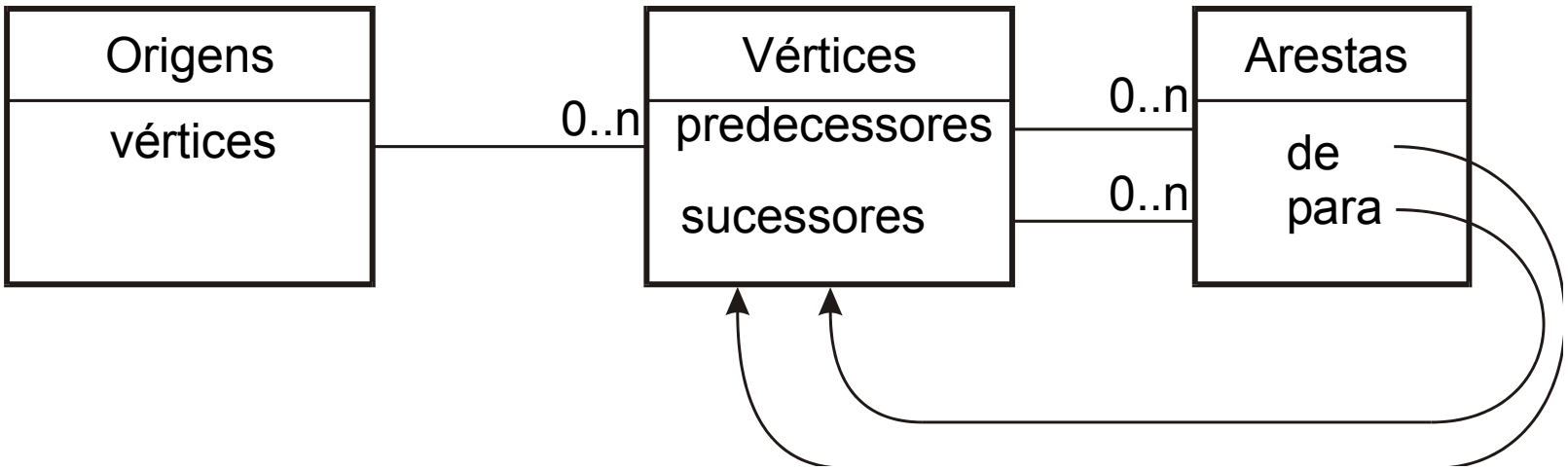
- Modelos devem ser redigidos em uma linguagem de representação precisamente definida
  - facilita o entendimento
  - permite o tratamento matemático
- Para modelos conceituais utilizaremos diagramas de classes da linguagem UML2
  - Conjuntos correspondem a classes e são representados por caixas
  - Relacionamentos entre classes (ex. aresta relaciona um par de vértices) são representados por arestas
  - Os terminadores das arestas informam propriedades (restrições) que os relacionamentos devem satisfazer

Modelo conceitual gráfico do grafo



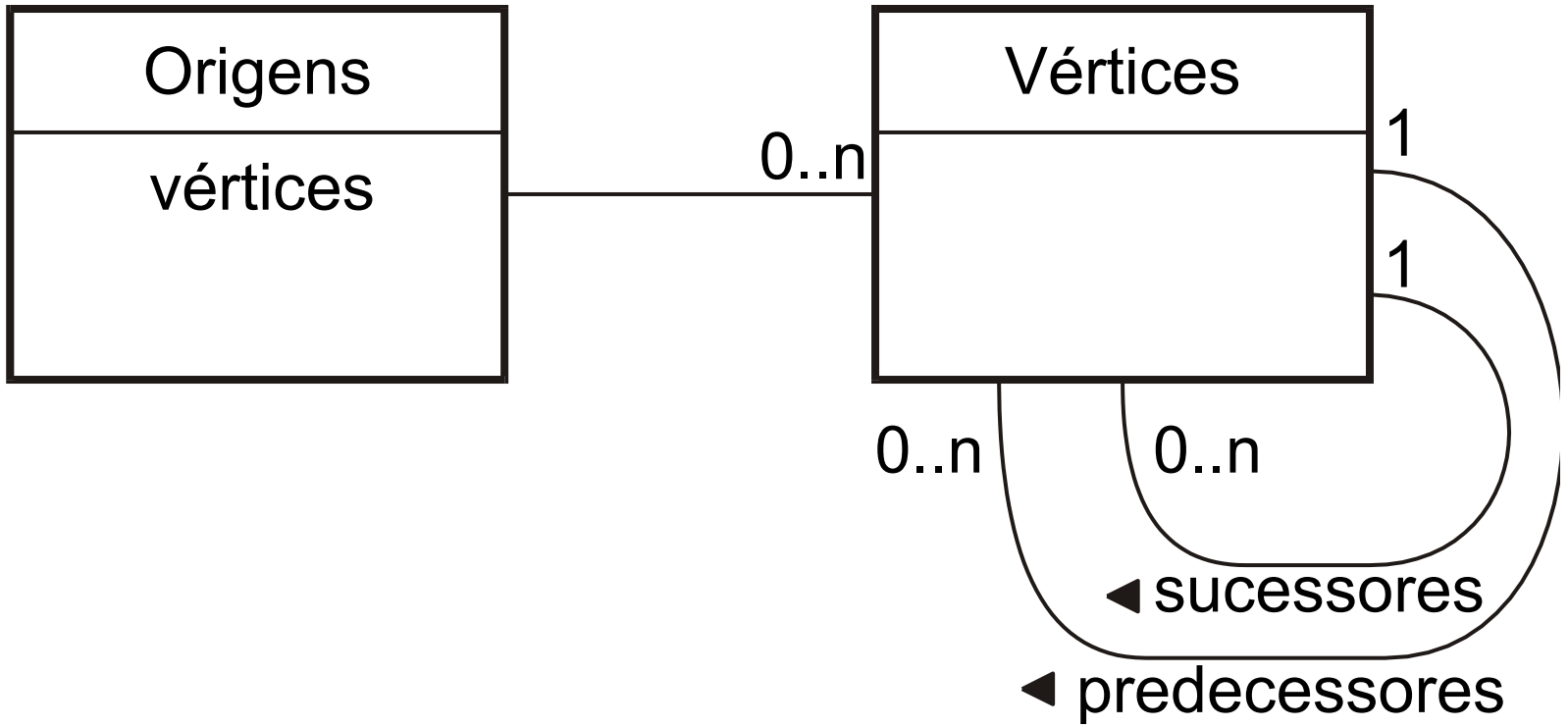
- Este modelo é uma transliteração direta da definição matemática
- Do ponto de vista computacional esse modelo é bom?
  - é fácil descobrir os antecessores e sucessores de um vértice?
    - requer pesquisa no conjunto de Arestas
    - seria possível ter essa informação diretamente?

Modelo conceitual transformado



- Melhorou?
- O conjunto arestas é realmente necessário?
  - conhecendo-se os sucessores, ou os predecessores, pode-se facilmente construir o conjunto de arestas

Modelo conceitual transformado final



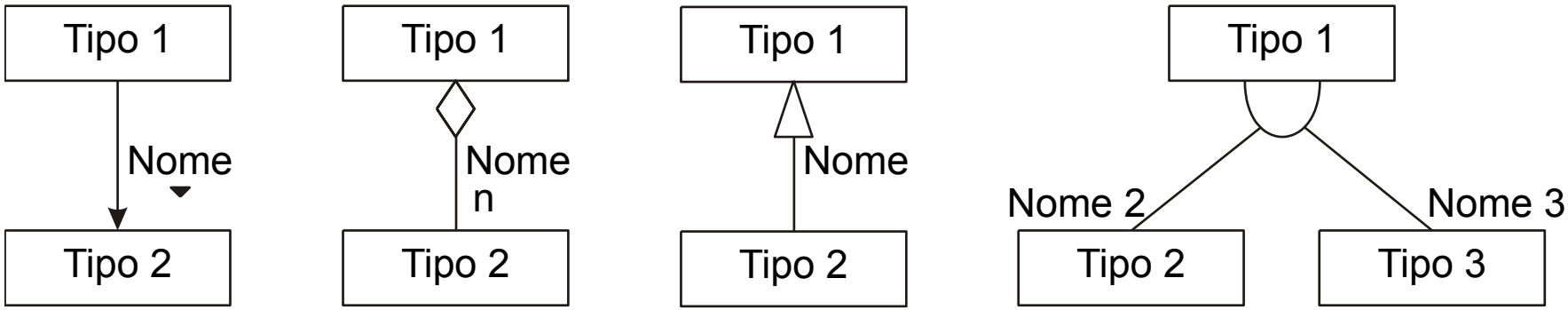
Como implementar isso?

- Pode-se representar diretamente em memória uma relação n para n, ou 1 para n?
  - não, em memória temos somente ponteiros que são relações 0..1 ➔ { NULL , endereço }
  - para representar um relação de n elementos precisamos de alguma estrutura complementar, em geral uma lista ou uma tabela

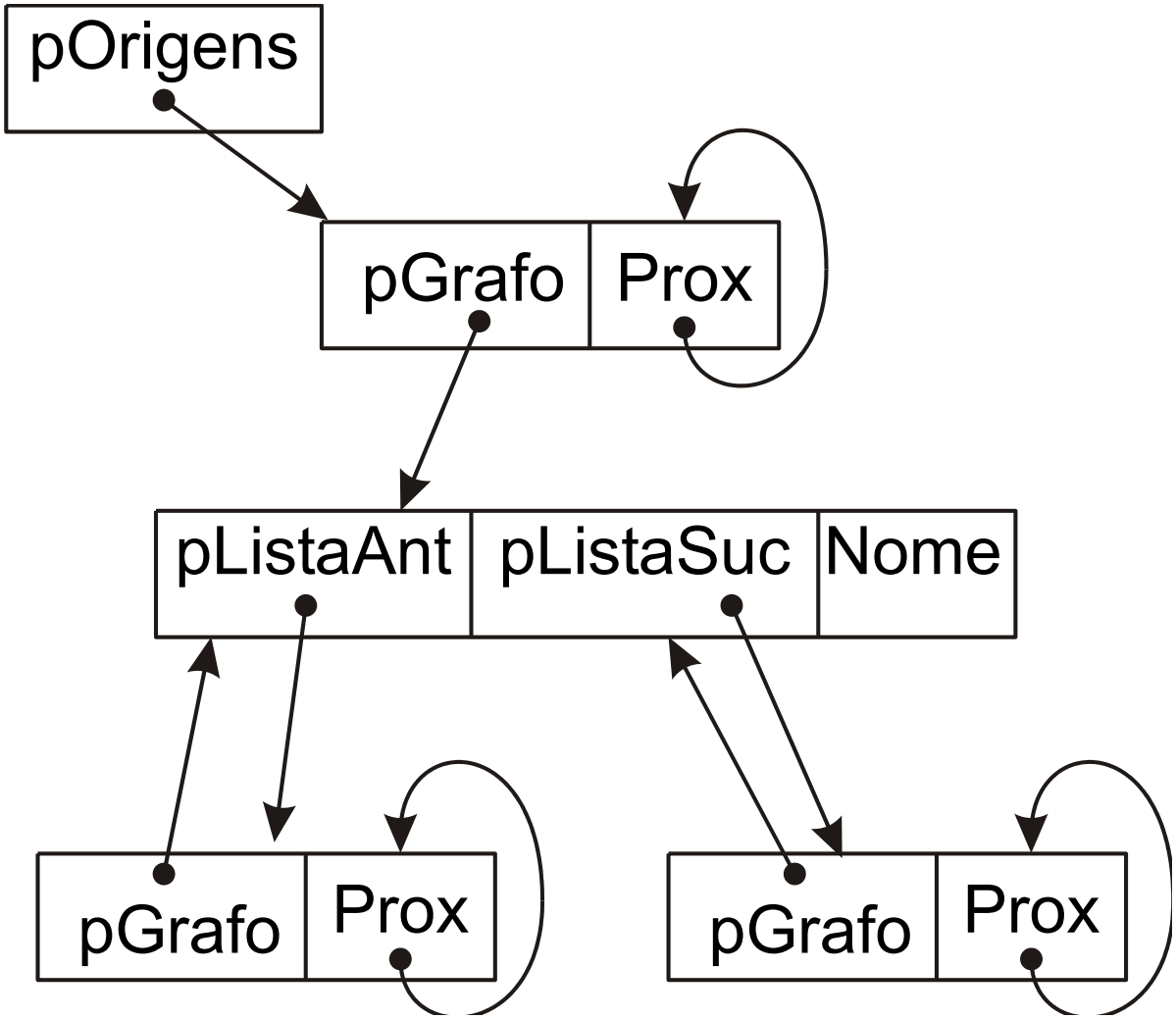
- Um modelo físico de um conceito (em geral uma estrutura de dados) descreve todas as possíveis instâncias desse conceito implementadas em memória física
- Um exemplo físico ilustra exatamente uma dessas possíveis instâncias.

Linguagem de representação

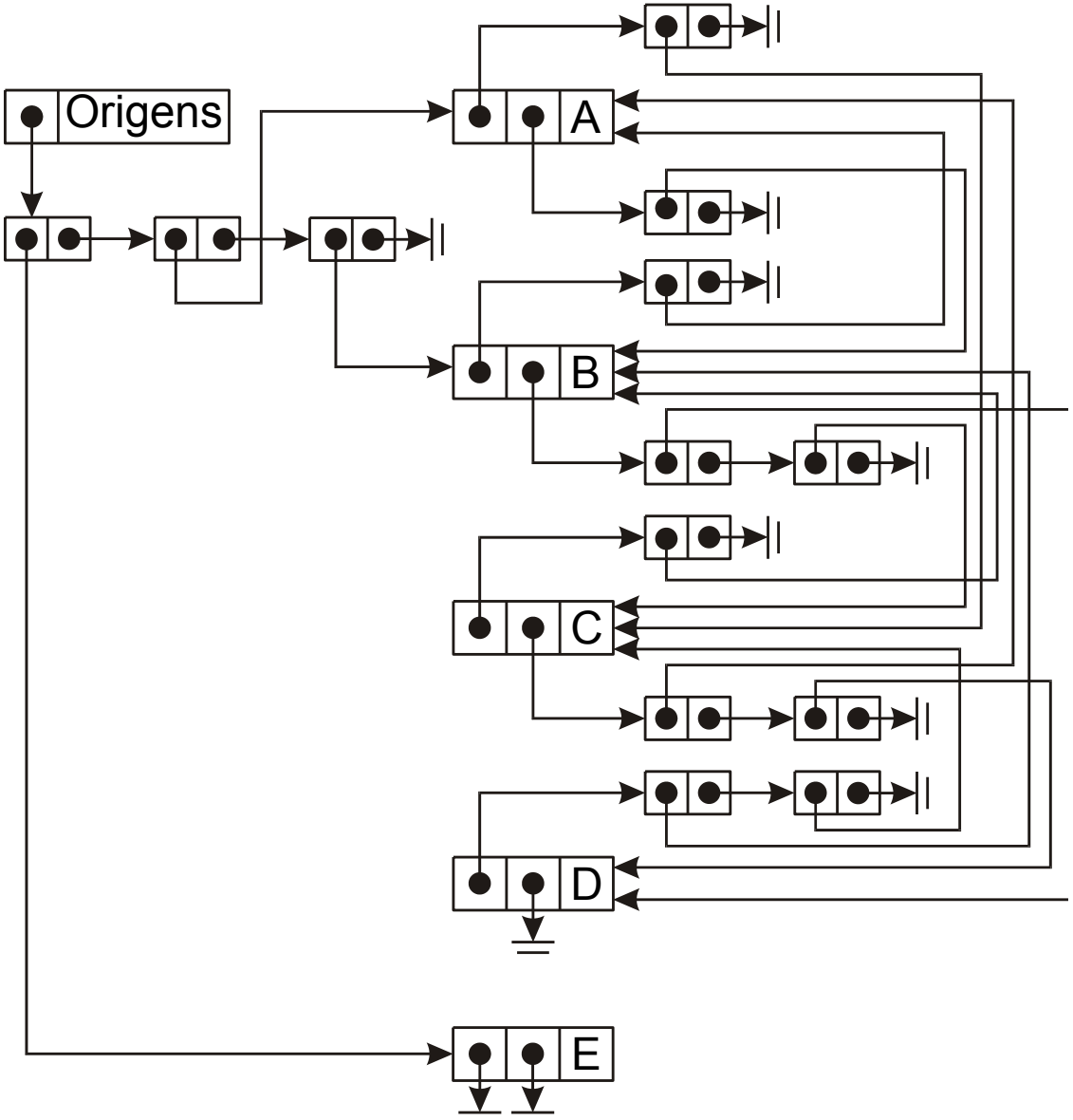
- Estruturas de dados físicas podem ser modeladas graficamente usando uma representação semelhante a dos diagramas de classe da UML2



Modelo físico do grafo

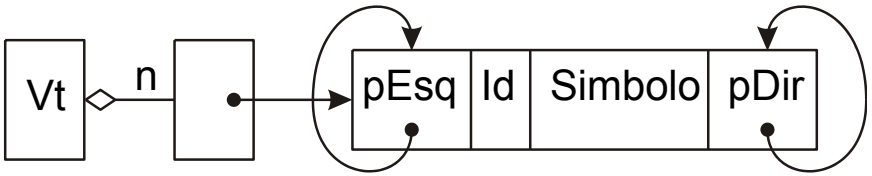


Exemplo físico do grafo

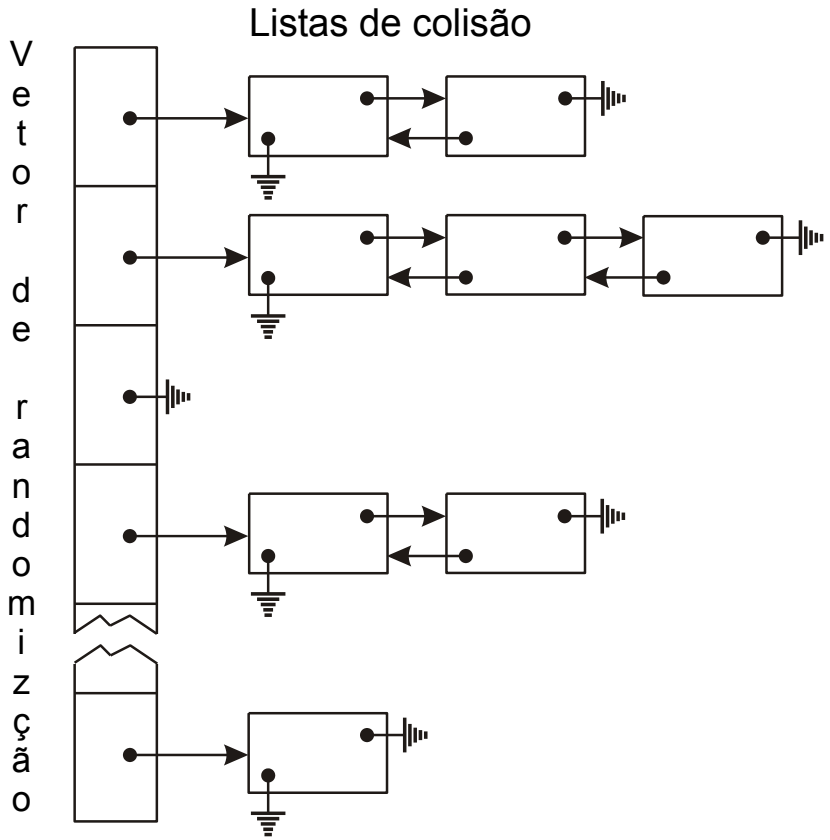


Outro exemplo de modelo e exemplo físico

Modelo

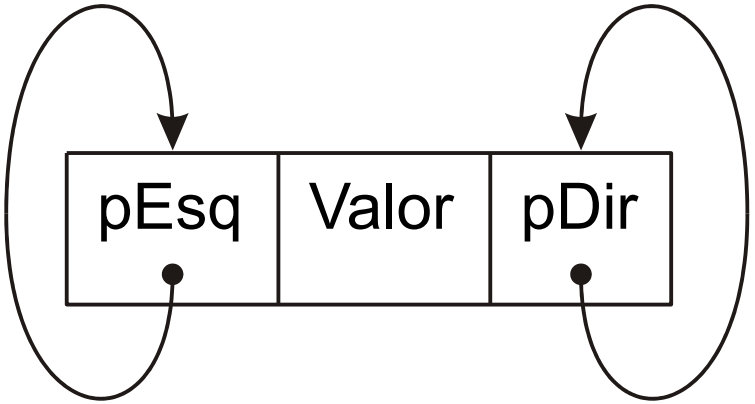


Exemplo

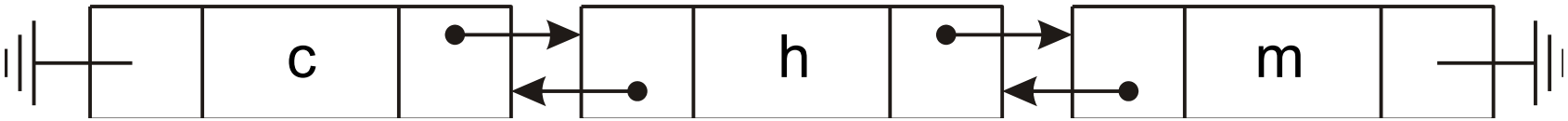


Figuras não são suficientes

Modelo de lista duplamente encadeada

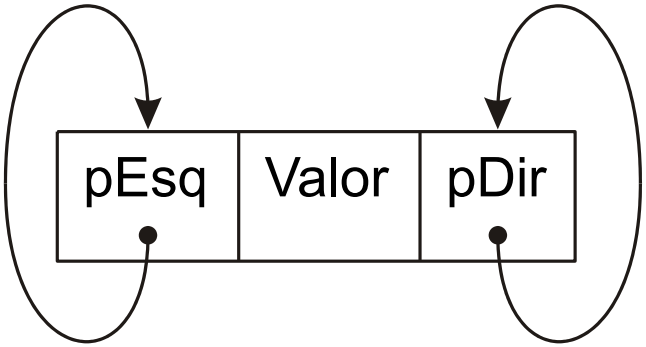


Exemplo de lista duplamente encadeada

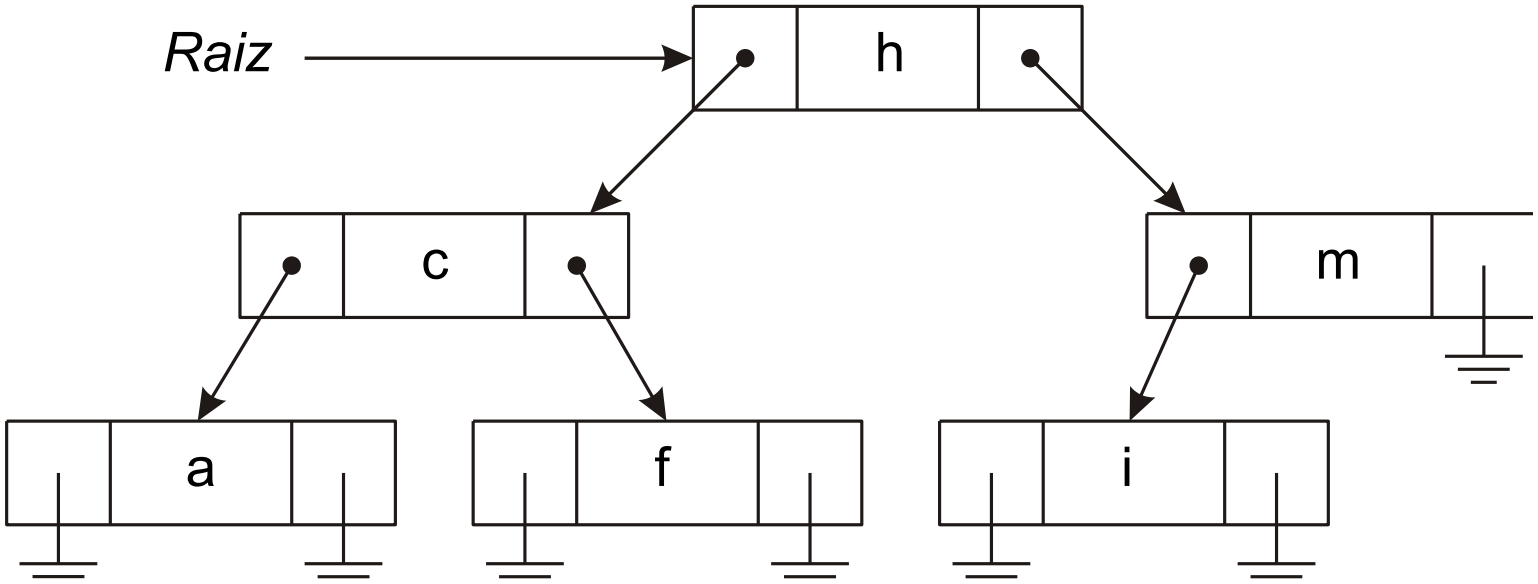


Figuras não são suficientes

Modelo de árvore binária



Exemplo de árvore binária



Assertivas estruturais

- Para resolver o problema da insuficiência de detalhes em figuras utilizam-se assertivas estruturais.
- Exemplo: lista duplamente encadeada
  - Para cada nó N da lista
    - se N->pEsq != NULL então N->pEsq->pDir == N
    - se N->pDir != NULL então N->pDir->pEsq == N
- Exemplo: árvore
  - para cada nó N da árvore
    - a referência para filho à esquerda de um nó tem como destino a raiz da sub-árvore à esquerda
    - a referência para filho à direita de um nó tem como destino a raiz da sub-árvore à direita
    - o conjunto de nós alcançáveis a partir da raiz da sub-árvore à esquerda é disjunto do conjunto de nós alcançáveis a partir da raiz da sub-árvore à direita

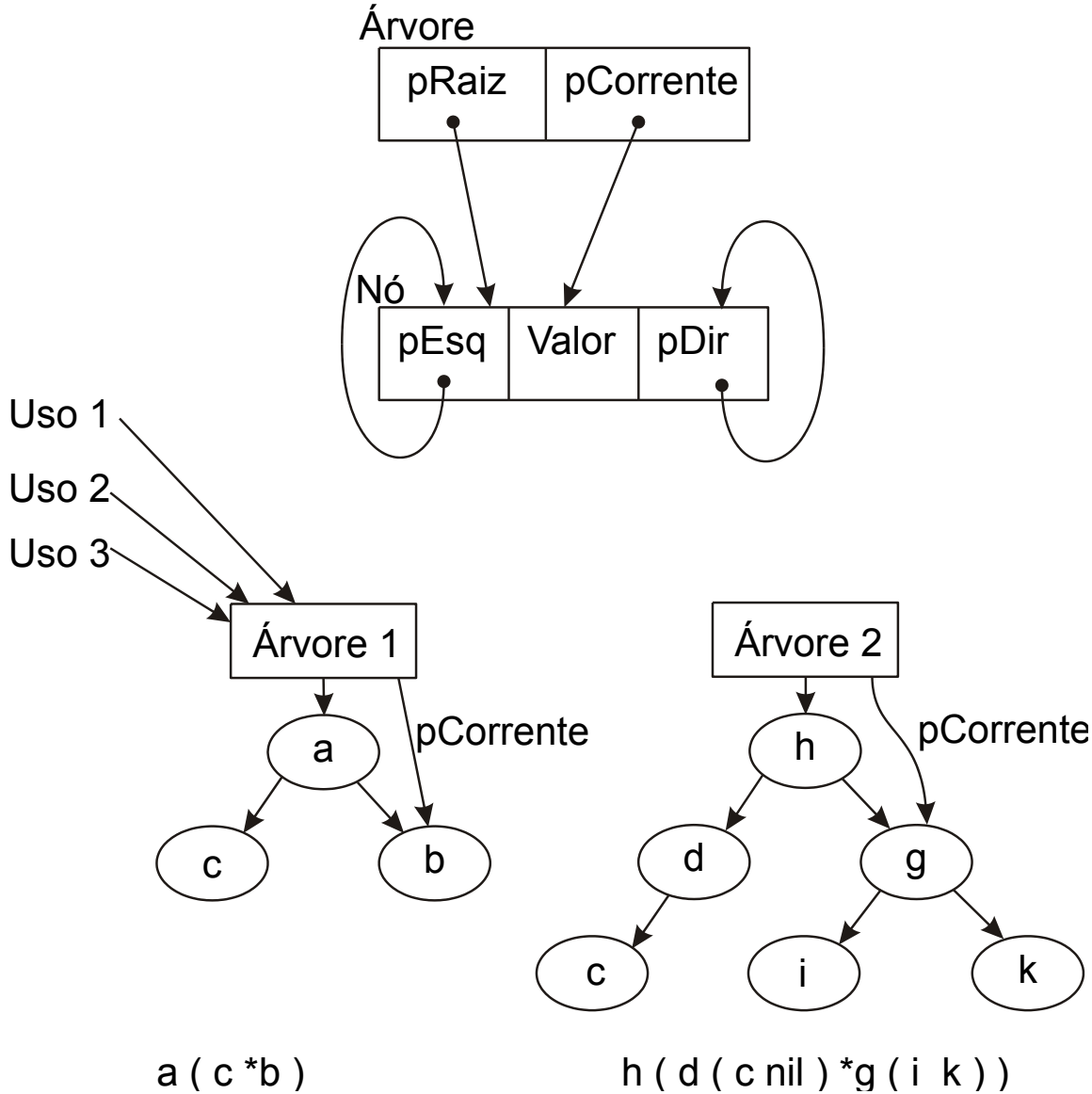
Qualidade de um tipo abstrato

- Um tipo abstrato será tão melhor
  - quanto mais exatamente corresponder a uma entidade mundo real
    - requer um bom entendimento do mundo real
  - quanto mais exatamente corresponder a um único conceito
    - requer uma boa compreensão do conceito
  - quanto menor for a sua interface
    - preferencialmente constituída exclusivamente por funções
    - todas as funções devem ser necessárias para algum cliente
    - o conjunto de funções deve ser suficiente para usar o tipo
    - nenhuma das funções realiza tarefas também realizadas por outras (ortogonalidade)
  - O nome do tipo deve refletir a sua intenção principal
    - não conseguir produzir um nome que capture bem a intenção do tipo é um forte indicador de que está mal definido

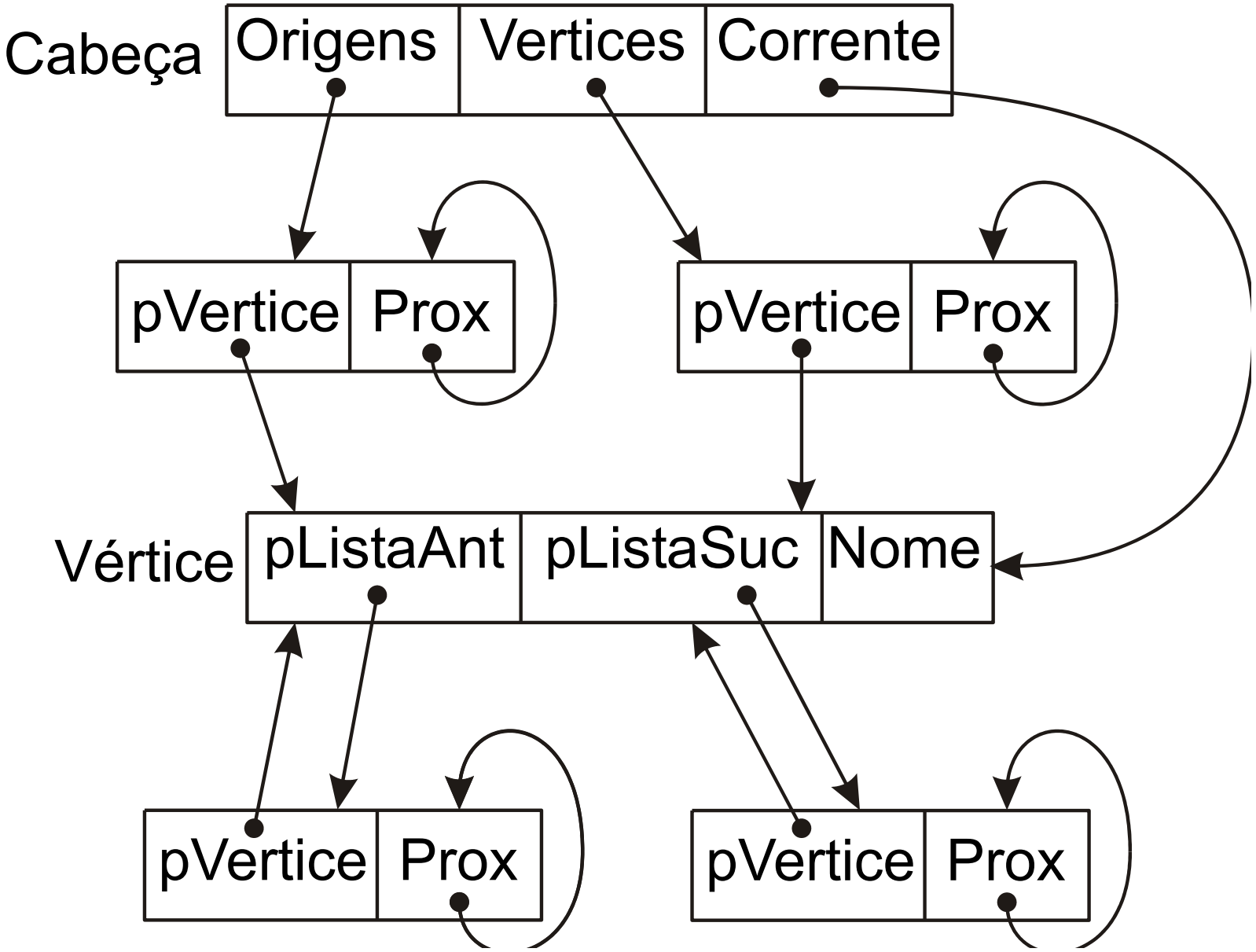
Cabeça de estrutura

- Cada estrutura de dados deve poder ser tratada como se fosse uma unidade
  - independente da sua complexidade e da diversidade de componentes
- Para tal pode-se utilizar uma cabeça de estrutura
  - todas as referências para a estrutura referenciam a cabeça
  - as referências internas à estrutura são desconhecidas ao cliente
- Vantagens
  - melhor encapsulamento da estrutura
    - a cabeça da estrutura passa a ser uma interface de acesso
    - reduz o risco de uso acidental ou deliberadamente incorreto
  - permite tratar estruturas vazias
  - permite mover as estruturas na memória
    - somente a cabeça precisa ficar fixa

Cabeça de estrutura, exemplo



Cabeça de estrutura, exemplo grafo



Vazamento de memória

- Ocorre vazamento de memória quando um espaço de dados alocado deixa de poder ser acessado pelo programa
  - exemplo banal:

```
p = malloc( 10 ) ;
p = NULL ;
```

- Vazamento de memória pode ocorrer somente com espaços de dados alocados dinamicamente (**`malloc`**)

Como prevenir vazamento de memória?

- coleta de lixo (*garbage collection*)
  - no caso geral C e C++ não podem disponibilizar coleta de lixo, pois permitem o cálculo de endereços
  - C e C++ podem disponibilizar coleta de lixo somente se o desenvolvedor estabelecer uma política de alocação rigorosa e fielmente obedecida
    - um caso particular

- cabe observar que existe vazamento de recursos e que não é tratado por coleta de lixo, exemplos
  - arquivos marcados em uso exclusivo, permanecem assim até serem desmarcados
  - janelas criadas permanecem disponíveis até serem destruídas

Âncora

- Espaços acessíveis devem estar ancorados
  - espaços podem estar ancorados a partir de diversos caminhos
- Regra básica:
  - por definição estão ancoradas :
    - variáveis globais
    - variáveis locais.
  - se o espaço E estiver ancorado, então todos os espaços referenciados por E estarão ancorados
  - sempre que uma função estiver para ser terminada devem ser destruídos os espaços (liberados os recursos) que deixarão de estar ancorados após retornar da função
    - ocorre com relação a espaços (recursos) direta ou transitivamente ancorados somente através de uma variável local dessa função
    - importantíssimo em linguagens que permitem exceções

FIM