PROGRAMAÇÃO MODULAR

Prof. Dr. Benedito Ferreira - FACOMP/UFPA

ABSTRAÇÃO

Processo intelectivo pelo qual nos concentramos em **ideias** e não nas **manifestações específicas** dessas ideias.

ABSTRAÇÃO

Processo intelectivo pelo qual nos concentramos em **ideias** e não nas **manifestações específicas** dessas ideias.

Questão decisiva em ciência da computação: concentração nos aspectos essenciais de um problema.

ABSTRAÇÃO

Processo intelectivo pelo qual nos concentramos em **ideias** e não nas **manifestações específicas** dessas ideias.

Questão decisiva em ciência da computação: concentração nos aspectos essenciais de um problema.

detalhes são considerados no momento oportuno

ABSTRAÇÃO

Na programação: distinção entre **o que** uma unidade faz e **como** ela faz isso.

ABSTRAÇÃO

Na programação: distinção entre **o que** uma unidade faz e **como** ela faz isso.

Separação de interesses (concentração, ocupação) entre programadores que usam uma unidade de programa e aqueles que a implementam.

```
#include <stdio.h>
#include <math.h>


...

seno = sin(a);
printf("...");
raizQuadr = sqrt(x);

...
```

```
#include <stdio.h>
#include <math.h>


...

seno = sin(a);
printf("...");
raizQuadr = sqrt(x);

...
```

```
. . .
void ordenaVet(int v[ ],n)
{
   for(int i = 0; i < n-1; i++)
      for(j = i+1; i < n; j++)
         if (...)
      ...
}
...
int main() {
   ...
   // leitura das notas
   for(int i = 0; i<n; i++)
      scanf("%f”,vetNotas[i]);
   ordenaVet(vetNotas,n);
   printf(vetNotas[0]);
   ...
```

Na e uma
un

So ão, ocupação)
en unidade de
pr

ABSTRAÇÃO

É possível usar unidades de baixo nível para implementar outras de mais alto nível.

ABSTRAÇÃO

É possível usar unidades de baixo nível para implementar outras de mais alto nível.

… e usar estas últimas para implementar unidades de nível ainda mais alto: vários níveis de abstração.

ABSTRAÇÃO

É possível usar unidades de baixo nível para implementar outras de mais alto nível.

… e usar estas últimas para implementar unidades de nível ainda mais alto: vários níveis de abstração.

→ *essencial* na construção de grandes sistemas de software.

É
im
...
de
ara
nidades
bstração.

Níveis mais altos (de abstração):
remetem detalhes
para níveis mais baixos.

(camadas intermediárias)

Níveis mais baixos: maior tratamento
de detalhes de implementação.

Abstração de procedimento:

Conceber a funcionalidade do programa a partir de unidades mais simples.

Abstração de procedimento:

Conceber a funcionalidade do programa a partir de unidades mais simples.

Abstração de dados:

Conceber o programa a partir de:

- tipos abstratos de dados,
- classes,
- encapsulamento,
- subclasses/herança, etc.

Abstração de procedimento:

Conceber a funcionalidade do programa a partir de unidades mais simples.

Abstração de dados:

Conceber o programa a partir de:

- tipos abstratos de dados,
- classes,
- encapsulamento,
- subclasses/herança, etc.

nosso assunto.

Programação modular

Metodologia para projetar programas como um conjunto de unidades individuais inter-relacionadas

→ sub-programas ou módulos.

Decomposição:

Fundamental no desenvolvimento de programas de médio/grande porte.

VANTAGENS DA MODULARIZAÇÃO

Facilita o desenvolvimento

Combinação de soluções de subproblemas (menos complexos).

Facilita o desenvolvimento

Um módulo que realiza uma função simples e bem definida:

+ facilmente → compreendido

Facilita o desenvolvimento

Um módulo que realiza uma função simples e bem definida:

+ facilmente
  - compreendido
  - desenvolvido

Facilita o desenvolvimento

Um módulo que realiza uma função simples e bem definida:

+ facilmente
  - compreendido
  - desenvolvido
  - modificado

Facilita o desenvolvimento

Um módulo que realiza uma função simples e bem definida:

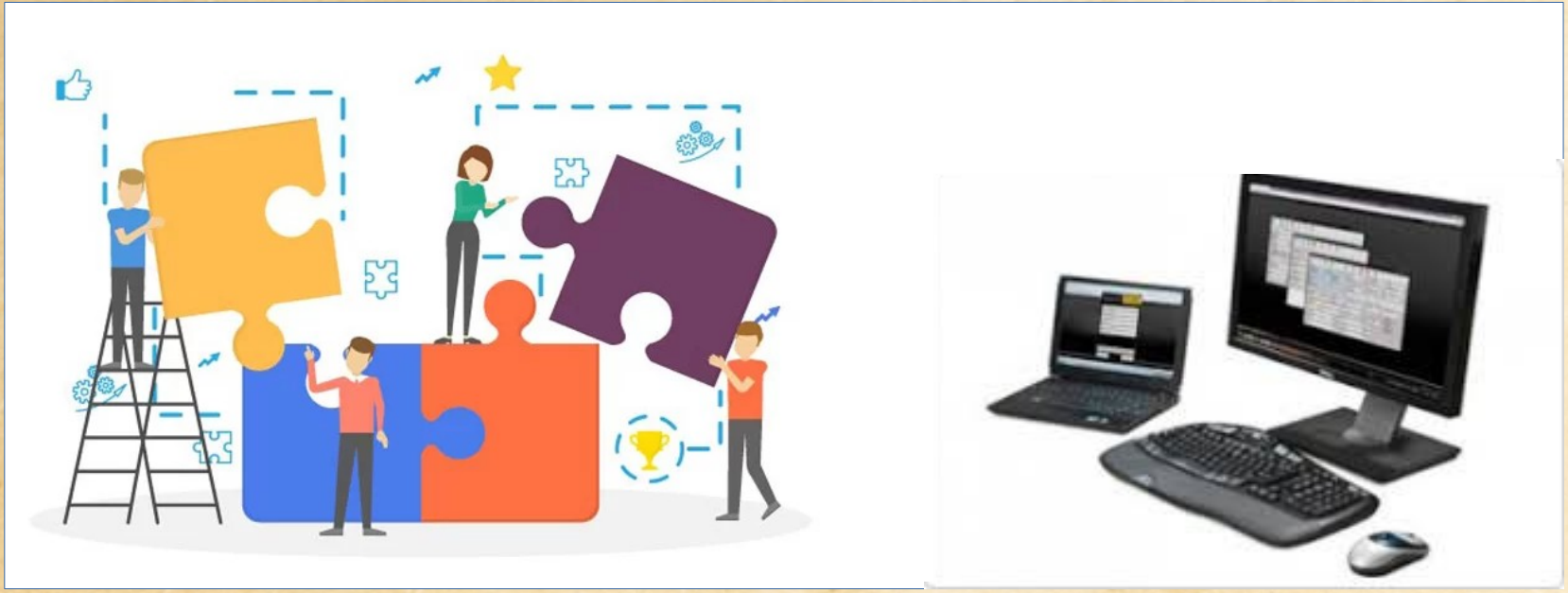
+ facilmente { compreendido, desenvolvido, modificado }

Foco do programador na lógica das partes, *isoladamente*:

ABSTRAÇÃO!

Flexibilidade:

→ alocação de recursos para o desenvolvimento.



Padronização e reúso:

→ é comum que certos módulos sejam necessários em vários programas.

Padronização e reúso:

→ é comum que certos módulos sejam necessários em vários programas.

Bibliotecas de funções:

Evita-se repetição de esforços;

Garante-se padronização.

Planejamento:

Com pequenas unidades, pode-se estimar com maior precisão a quantidade de trabalho envolvido.

→ recursos, prazos, etc.

Manutenção:

É mais fácil
- -localizar erros
- -entender um pequeno módulo
- -fazer alterações

Escalabilidade:

Melhores condições para o aumento da dimensão e da utilização do sistema computacional.

Escalabilidade:

Melhores condições para o aumento da dimensão e da utilização do sistema computacional.

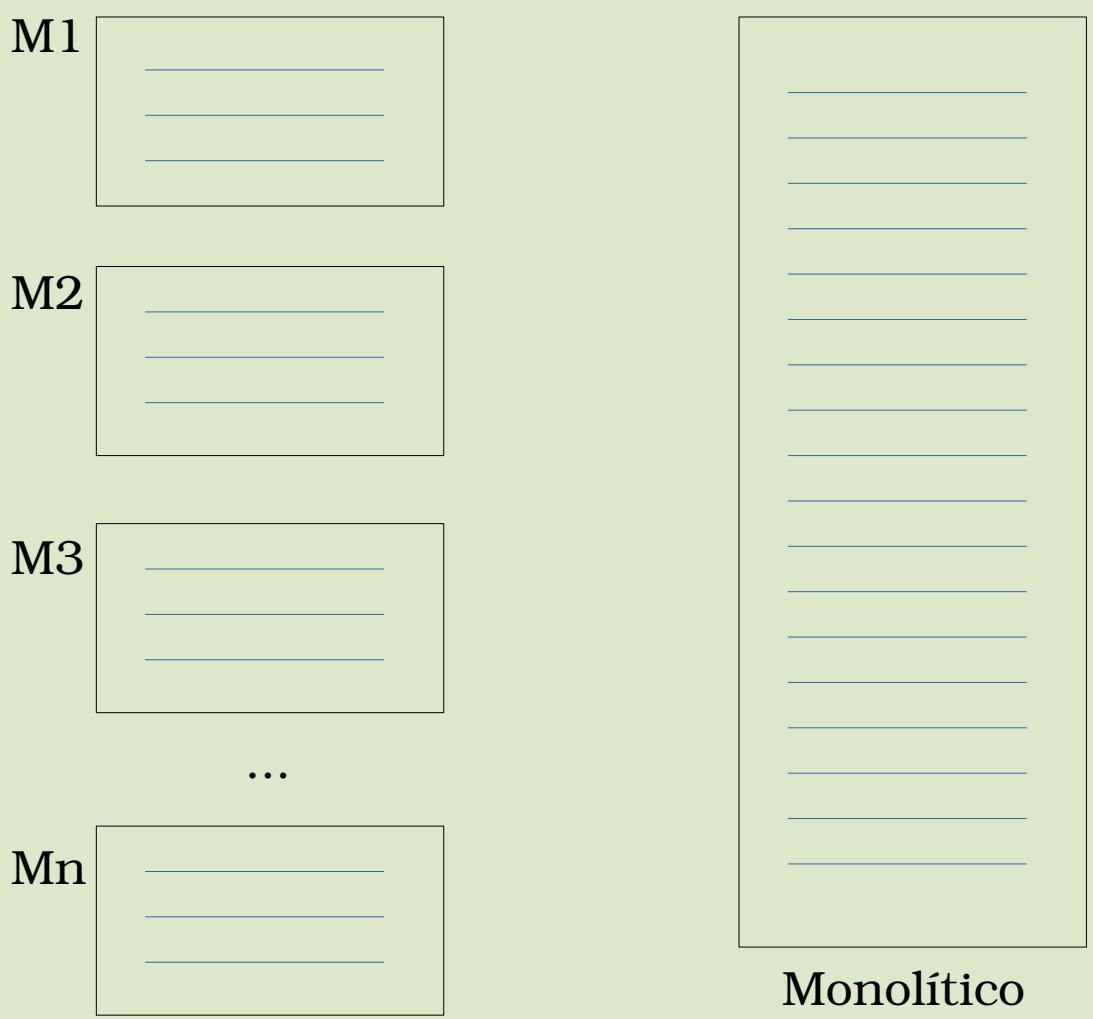
→ Em comparação ao desenvolvimento *monolítico*.

Es

M
e

→
m

ensão



RECOMENDAÇÃO:

Os módulos devem vincular-se a uma **tarefa (ou função) muito bem definida.**

RECOMENDAÇÃO:

Os módulos devem vincular-se a uma **tarefa (ou função) muito bem definida.**

Definir o que um módulo faz em poucas palavras (sem “e”, “ou“, “se“, etc.)

OBS:
A técnica de modularizaçao é adotada em vários ramos de desenvolvimento, como uma forma de lidar com a complexidade.

OBS:
A técnica de modularizaçao é adotada em vários ramos de desenvolvimento, como uma forma de lidar com a complexidade.

→ Não há um "fabricante de computadores". Há fabricantes de discos rígidos, de processadores, etc.

OBS:
A técnica de modularizaçao é adotada em vários ramos de desenvolvimento, como uma forma de lidar com a complexidade.

→ Não há um "fabricante de computadores". Há fabricantes de discos rígidos, de processadores, etc.

→ Não há um "fabricante de carros". Há montadoras que recebem de terceiros motor, câmbio, pneus, condicionador de ar, etc.

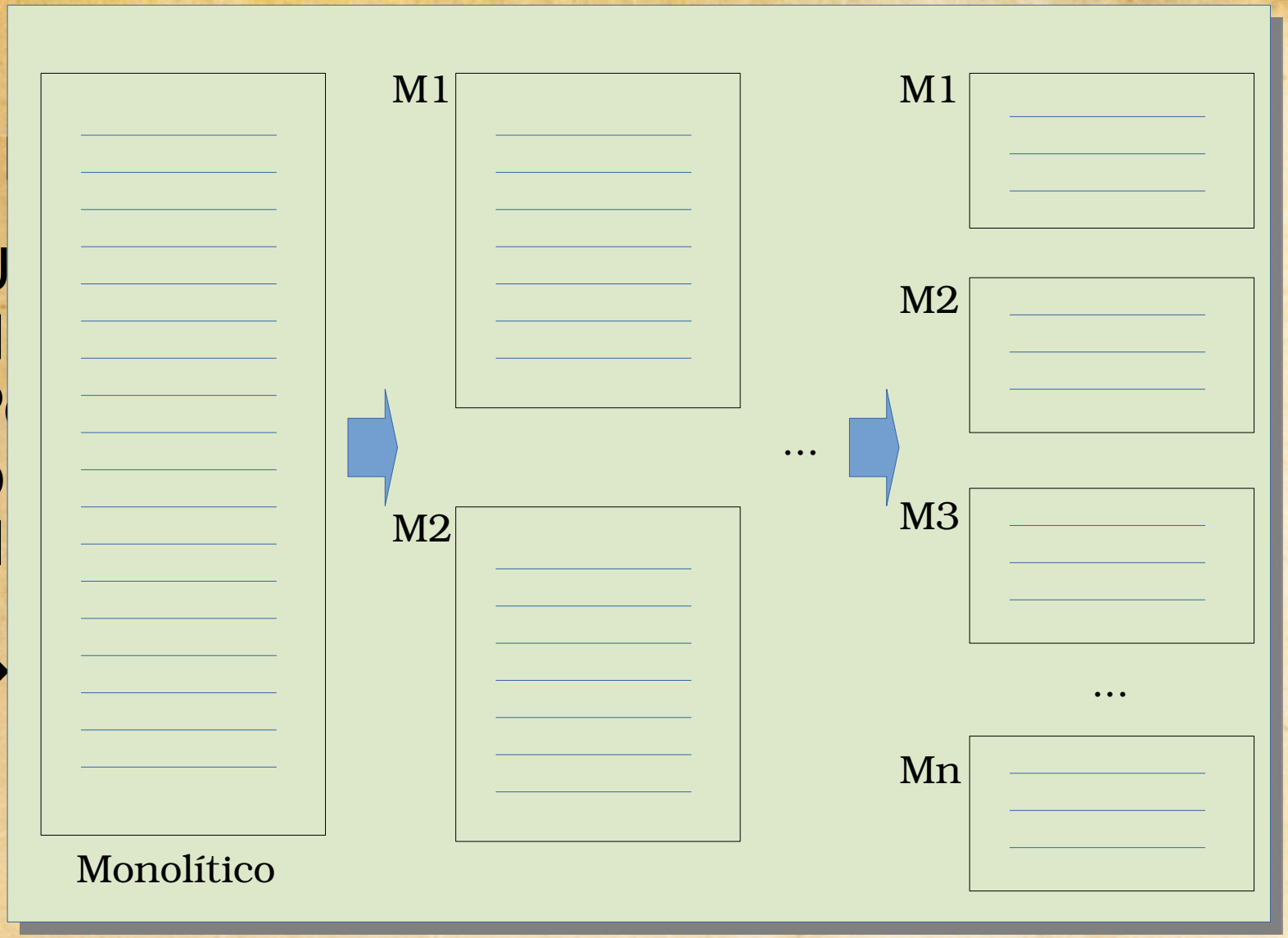
OBS:

Um módulo, por sua vez, pode ser decomposto em outros mais simples. Por exemplo, o fabricante de motor pode terceirizar a fabricação das velas de ignição, pistons, etc.

OBS:

Um módulo, por sua vez, pode ser decomposto em outros mais simples. Por exemplo, o fabricante de motor pode terceirizar a fabricação das velas de ignição, pistons, etc.

→ processo de **decomposição sucessiva**.


M1
M2
...
M1
M2
M3
...
Mn
Monolítico

TIPOS DE MÓDULO

Procedimento:

Realiza **uma tarefa**.
Ao final não há retorno de valores ao módulo que o ativou.

Função:

Realiza algum processamento e retorna uma informação de saída.

Ex: Pascal

```
procedure <nome> (<list_parâm>);
...
var
  {variáveis locais}
begin
  comando 1;
  comando 2;
    ...
  comando n
end;
```

Ex: Pascal

```
function <nome> (<list_parâm>): <tipo_ret>;
...
var
   {variáveis locais}
begin
   comando 1;
   comando 2;
      ...
   comando n;
   <nome> := <express_tipo_ret>
end;
```

Ex: C/C++

```
void <nome> (<list_parâm>) {
...

  // variáveis locais

  comando 1;
  comando 2;
     ...
  comando n;
}
```

Ex: C/C++

```
<tipo_ret> <nome> (<list_parâm>) {
...

  // variáveis locais

  comando 1;
  comando 2;
     ...
  comando n;
  return <express_tipo_ret>;
}
```

Ex: Python

```
def <nome> (<list_parâm>):
  comando 1
  comando 2
  comando n
```

Ex: Python

```
def <nome> (<list_parâm>):
   comando 1
   comando 2
   comando n
   return <express>
```

Dinâmica de ativação/retorno:

A execução de um módulo M1 é ativada pela referência ao seu nome.

Dinâmica de ativação/retorno:

A execução de um módulo M1 é ativada pela referência ao seu nome.

O módulo em execução M2 será suspenso, e o controle passa ao módulo M1.

Dinâmica de ativação/retorno:

A execução de um módulo M1 é ativada pela referência ao seu nome.

O módulo em execução M2 será suspenso, e o controle passa ao módulo M1.

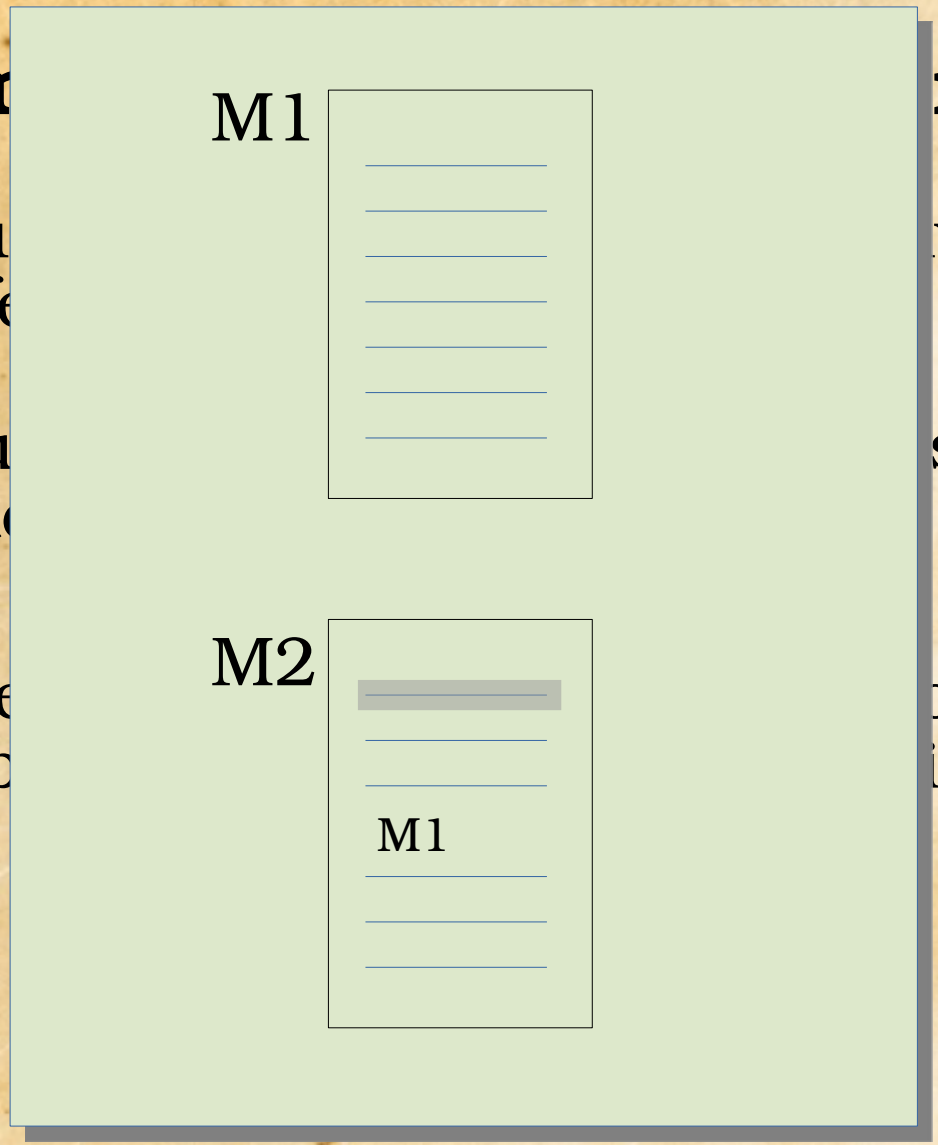
Após a execução, ocorrerá o retorno do controle para M2 no ponto de chamada (ou imediatamente após).

Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

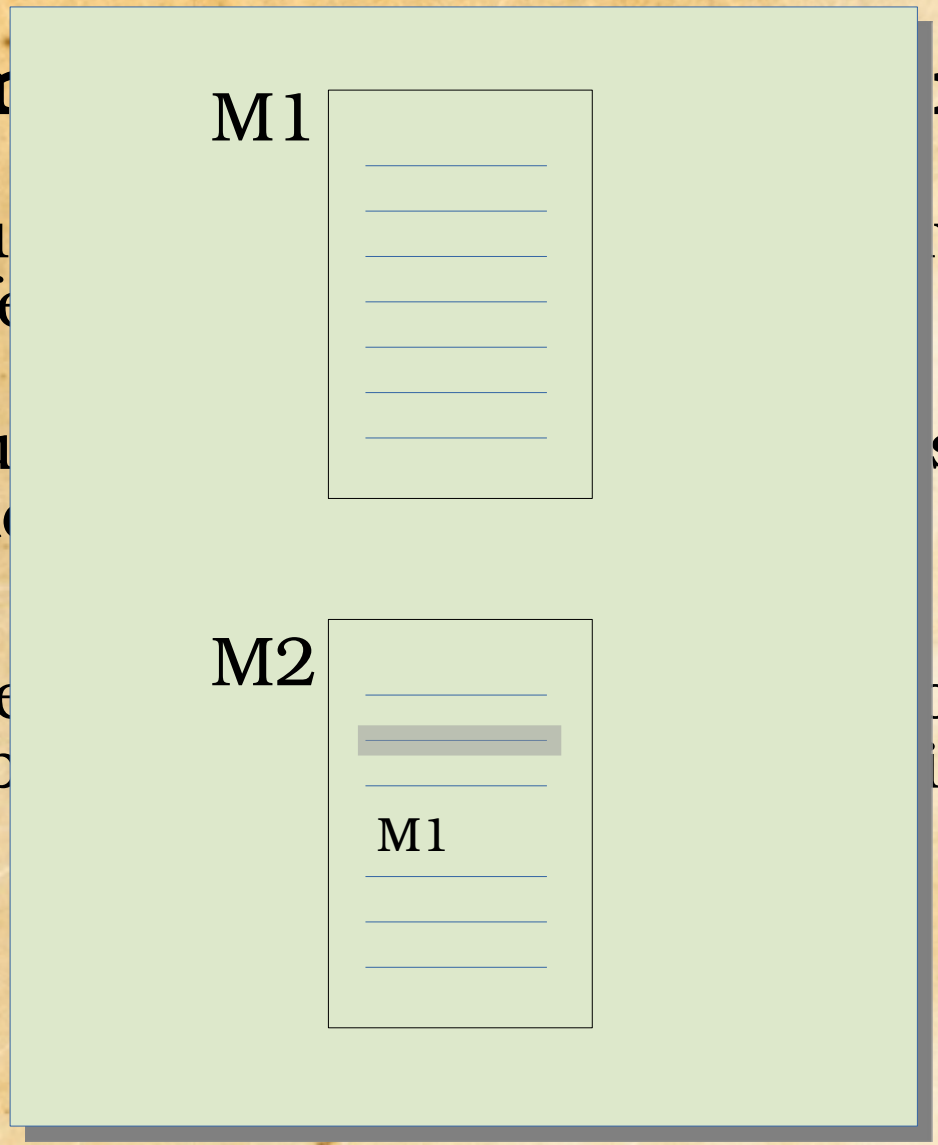


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

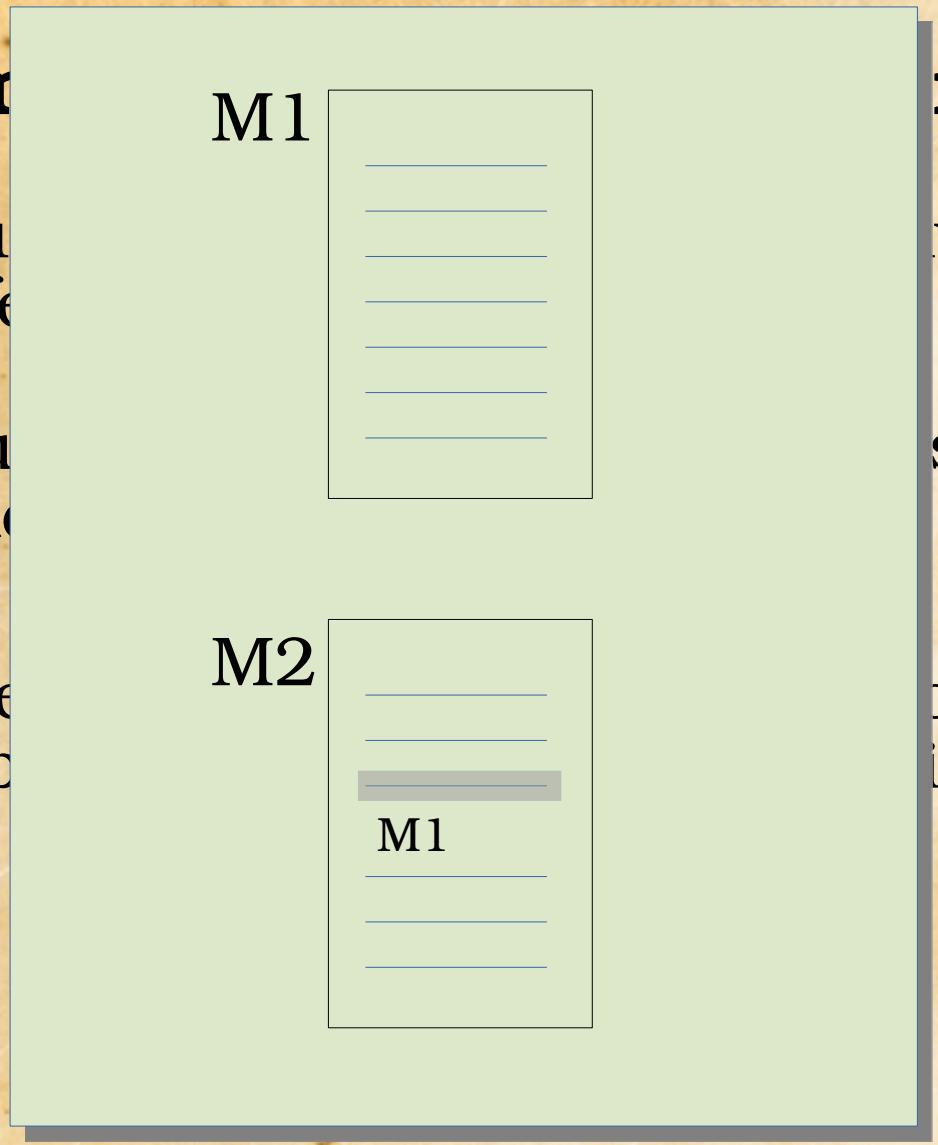


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

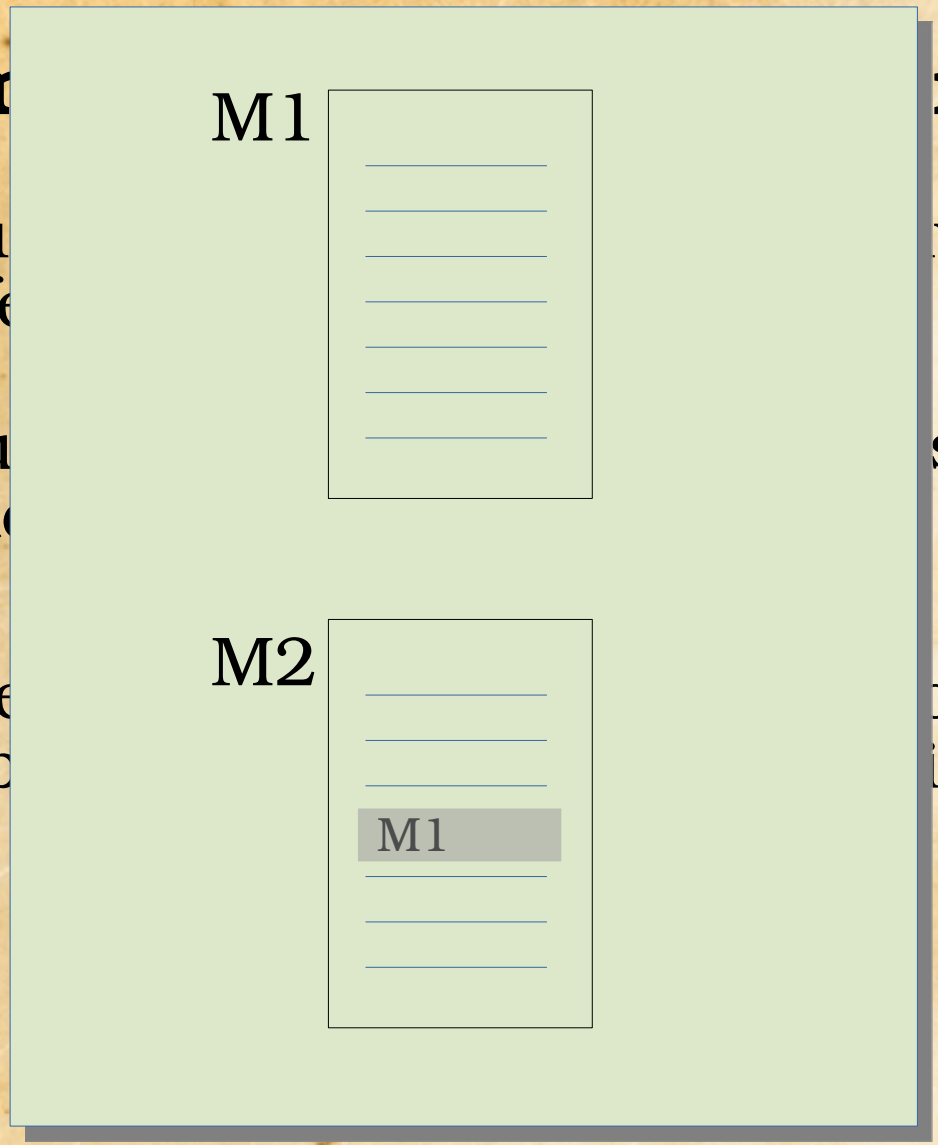


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

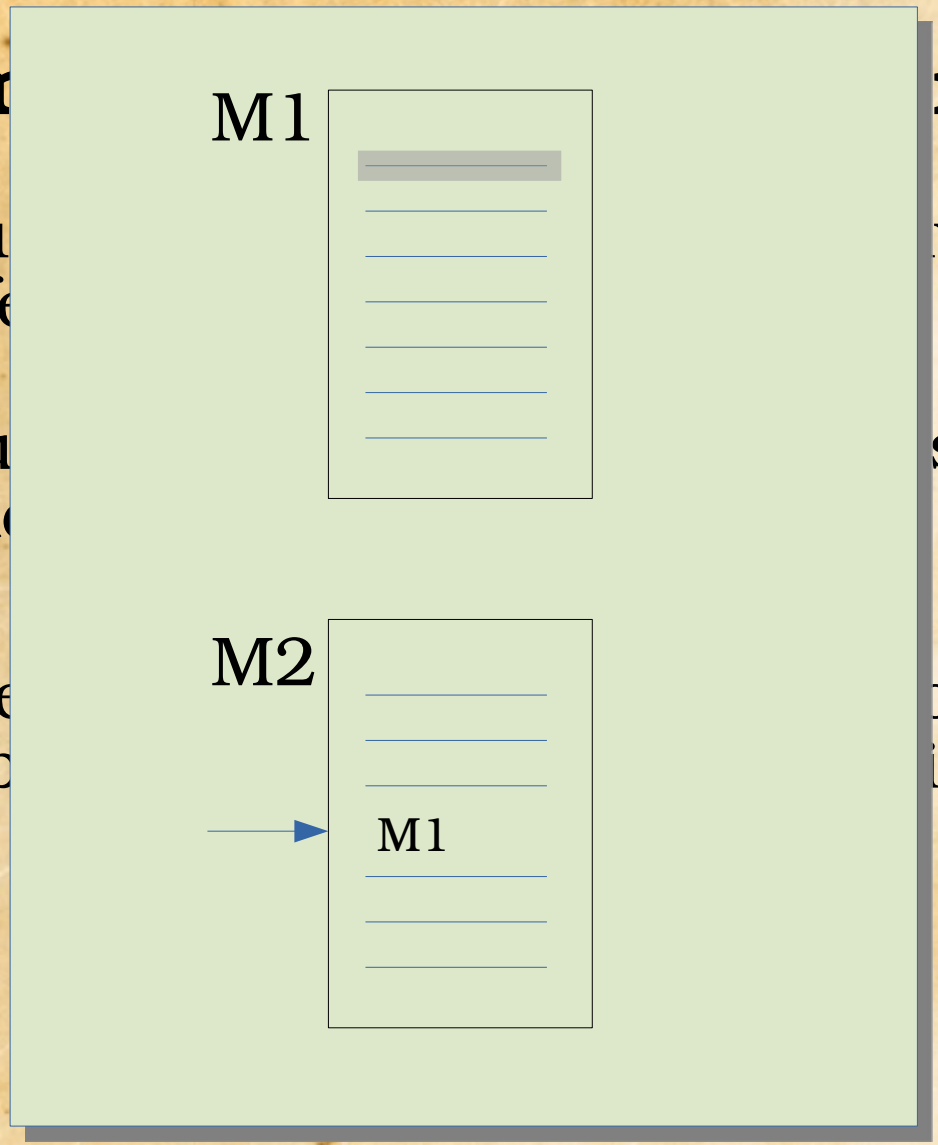


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

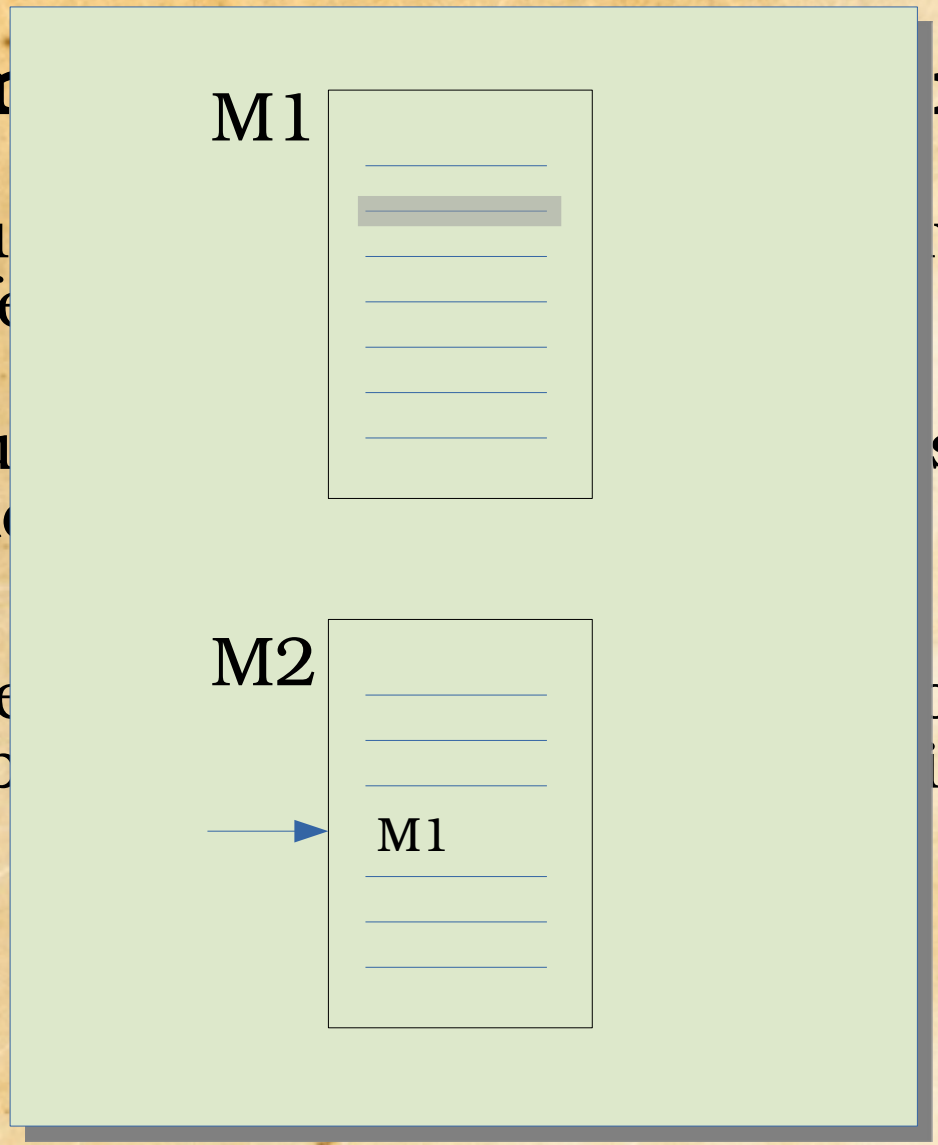


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

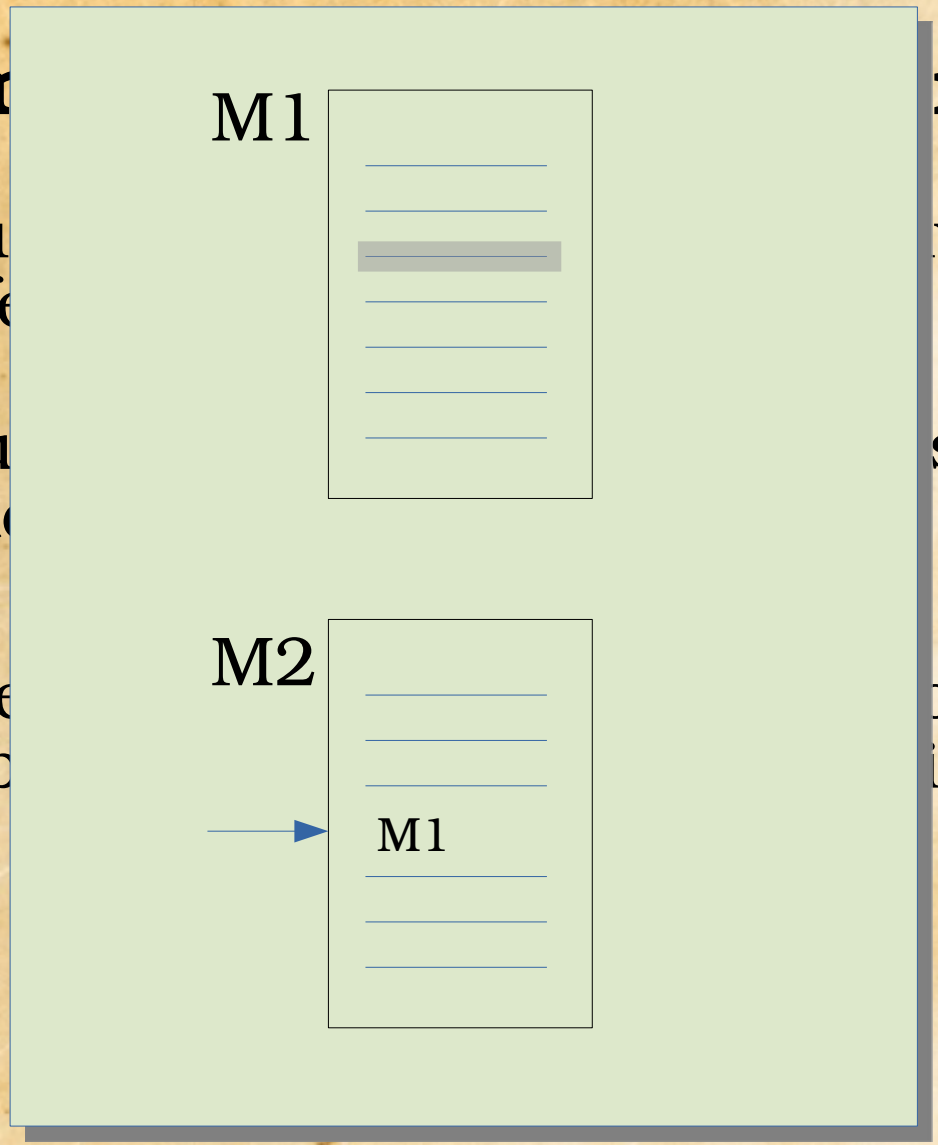


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

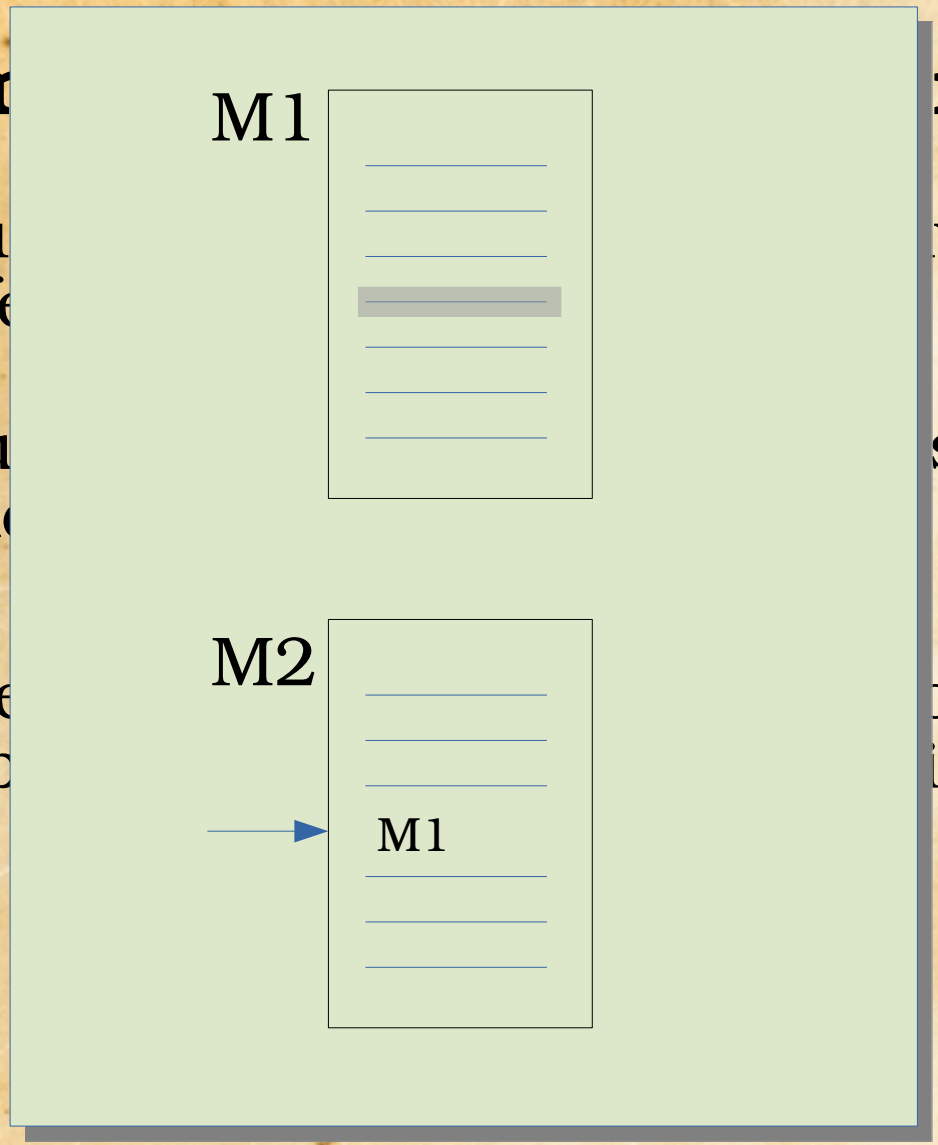


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

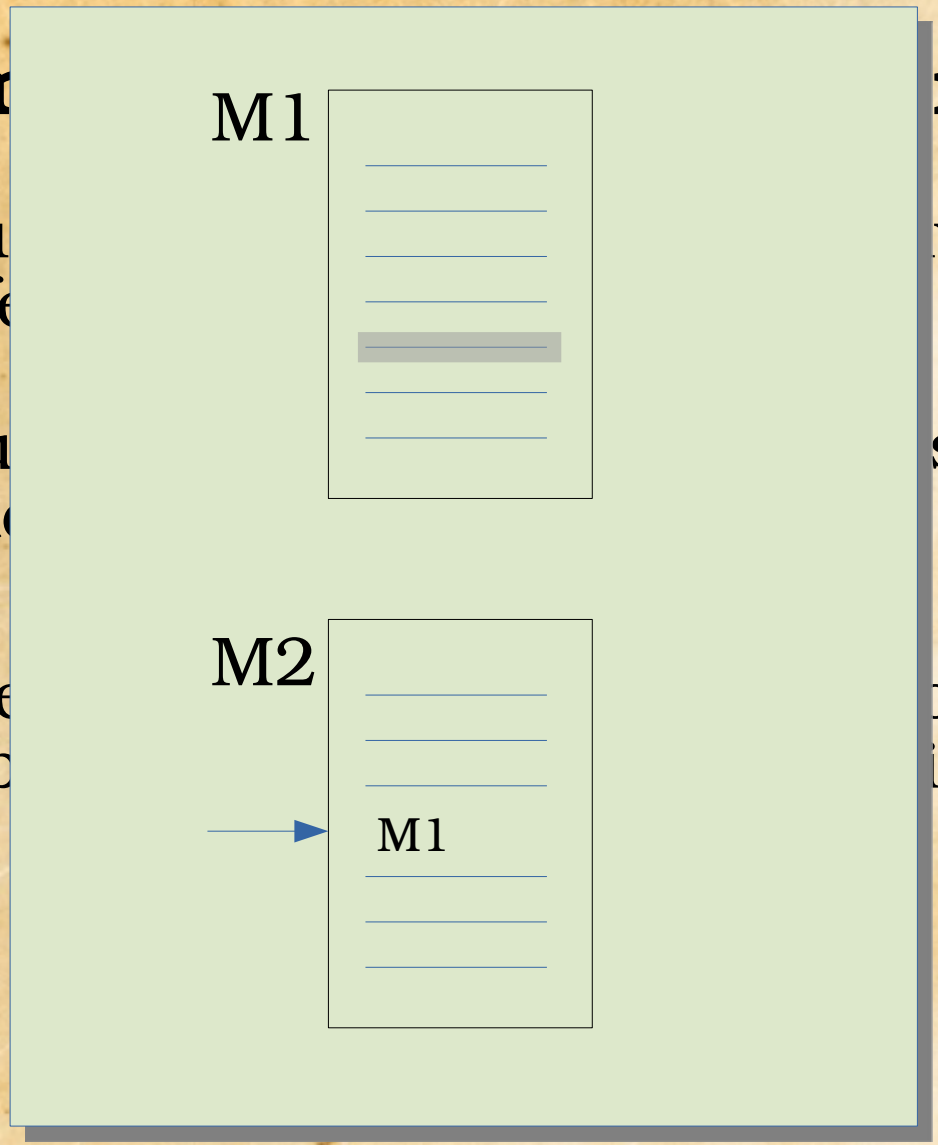


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

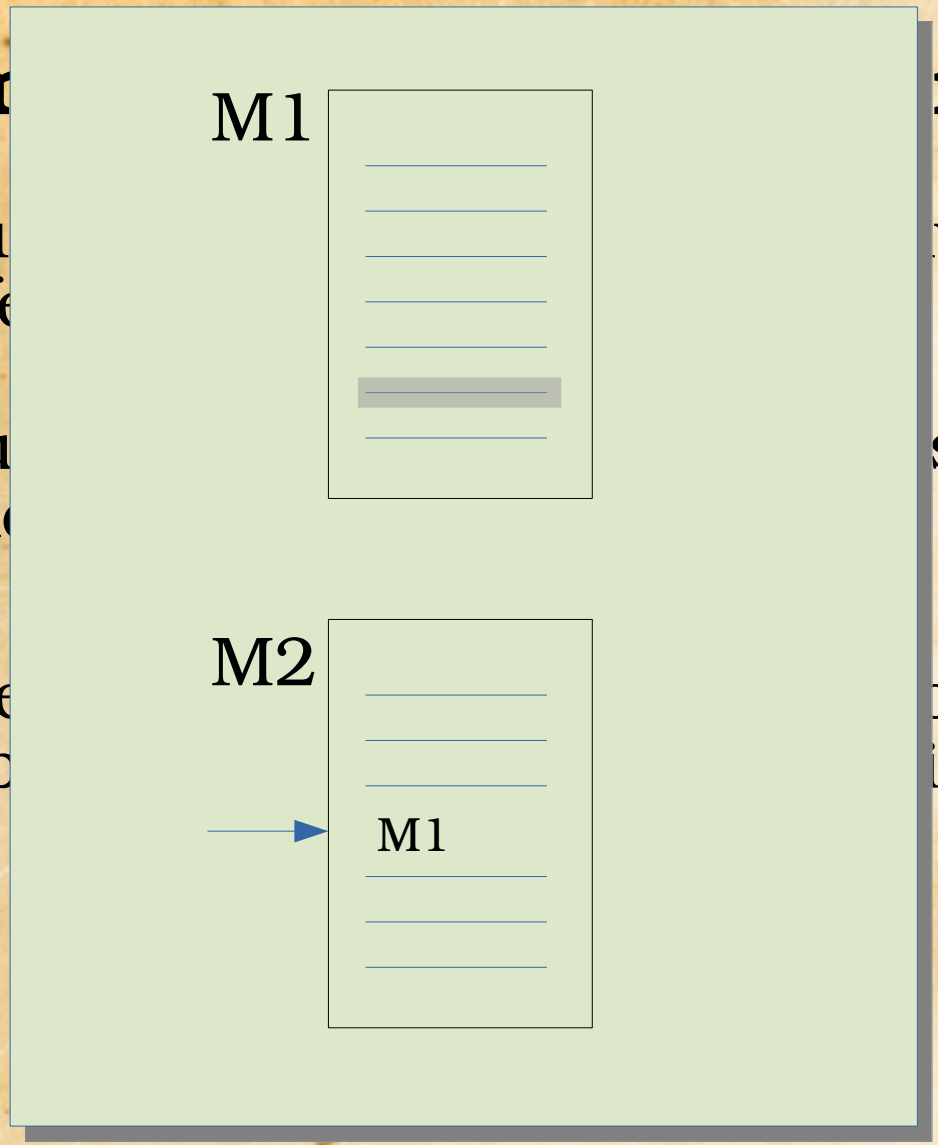


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

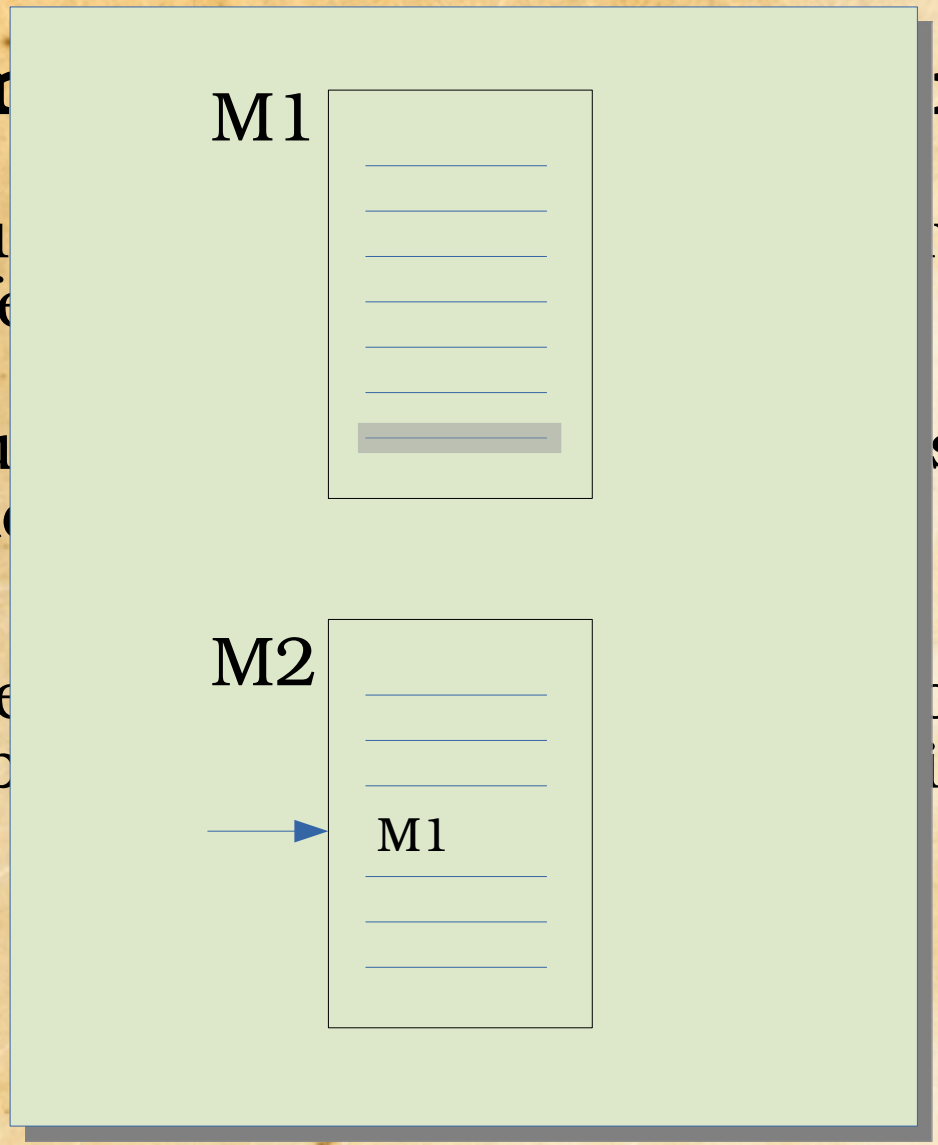


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

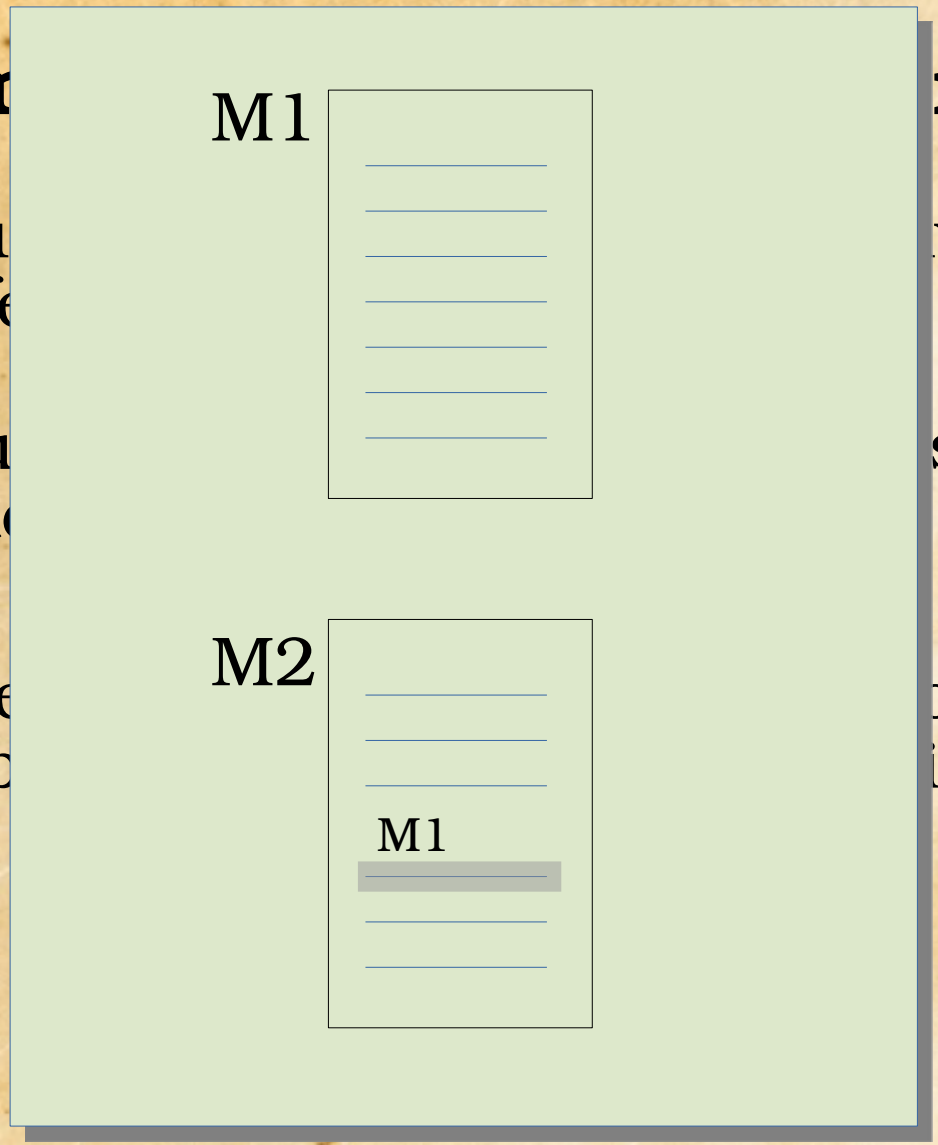


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

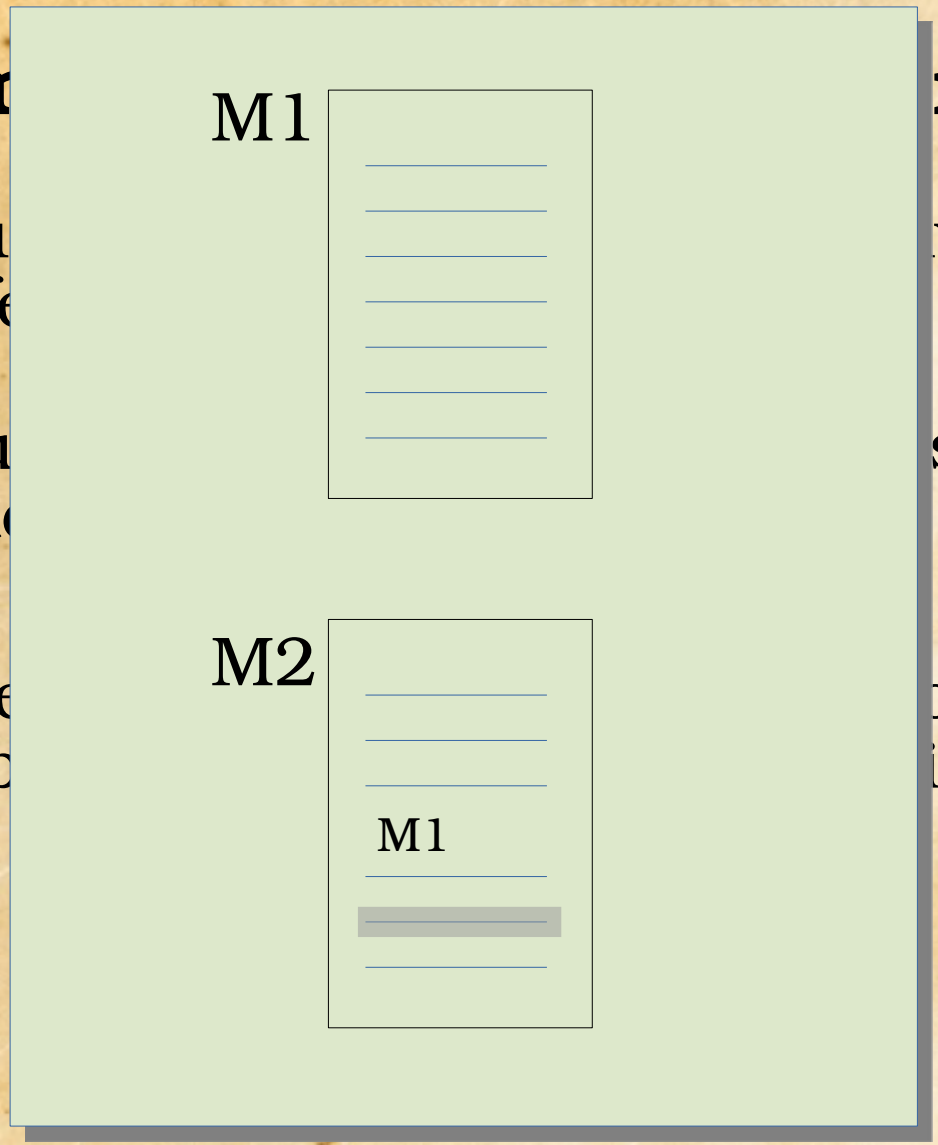


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).

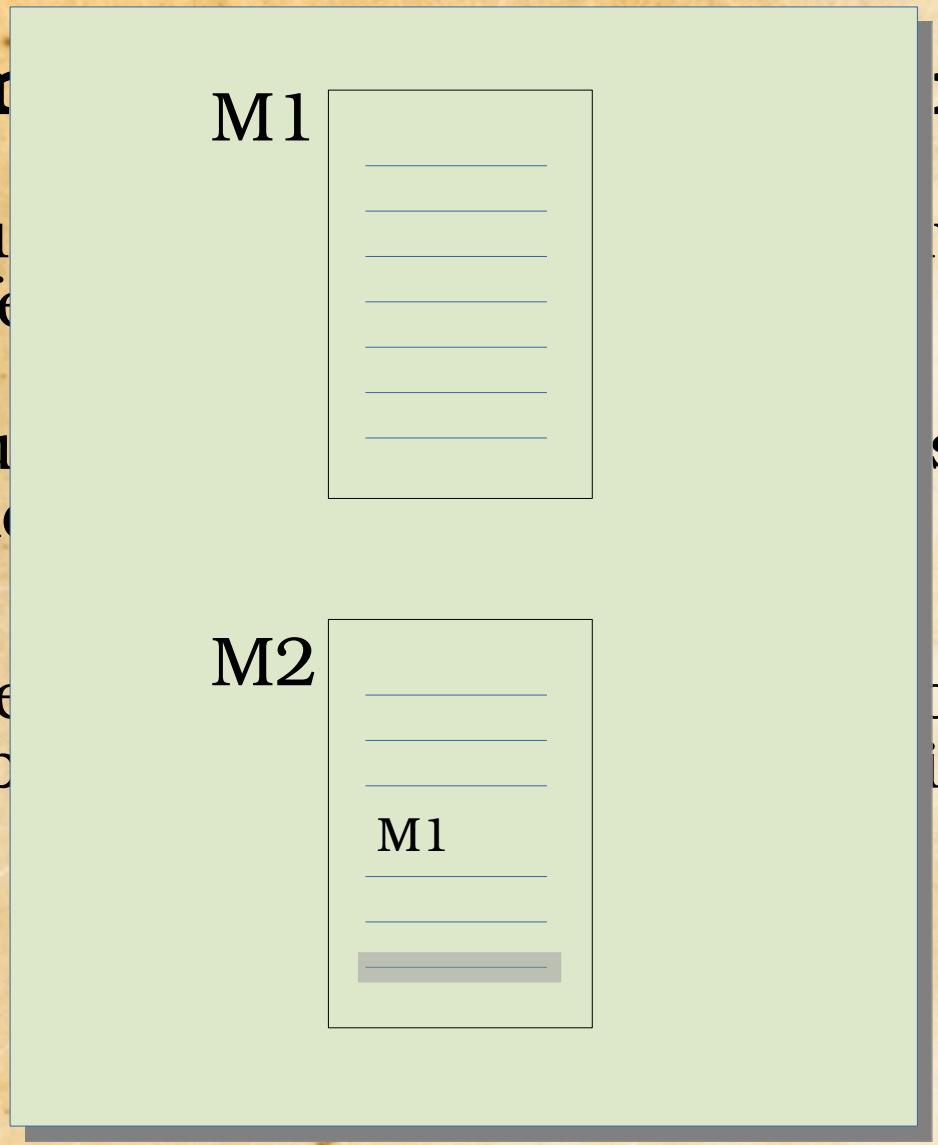


Dinâm **rno:**

A execu vada
pela refe

O módu spenso, e o controle
passa a

Após a e o do controle para
M2 no p iatamente após).



ESCOPO E TEMPO DE VIDA

Prof. Dr. Benedito Ferreira - FACOMP/UFPA

Escopo: área do programa onde um item (variável, tipo, objeto, etc.) pode ser referenciado.

Escopo: área do programa onde um item (variável, tipo, objeto, etc.) pode ser referenciado.

Quanto ao escopo, pode haver itens:

Globais: são definidos fora de qualquer módulo/classe, e podem ser referenciadas em qualquer parte do programa.

Escopo: área do programa onde um item (variável, tipo, objeto, etc.) pode ser referenciado.

Quanto ao escopo, pode haver itens:

Globais: são definidos fora de qualquer módulo/classe, e podem ser referenciadas em qualquer parte do programa.

Locais: só podem ser referenciados dentro da sua respectiva unidade.

```
...
int x1,x2;

void P() {
  float y1,y2,x2;
  ...
}

void Q() {
  int z,y1;
  ...
}

int main( ) {
  float r,s;

}
```

```
...
int x1,x2;

void P() {
  float y1,y2,x2;
  ...
}

void Q() {
  int z,y1;
  ...
}

int main( ) {
  float r,s;

}
```

GLOBAIS

LOCAIS

```
...
int x1,x2;

void P() {
  float y1,y2,x2;
  ...
}

void Q() {
  int z,y1;
  ...
}

int main( ) {
  float r,s;

}
```

A variável global x2 não será visível em P (prevalece a local)

```
...
int x1,x2;

void P() {
  float y1,y2,x2;
  ...
}

void Q() {
  int z,y1;
  ...
}

int main( ) {
  float r,s;

}
```

...quanto a y1,
não há conflito
(escopos distintos)

Tempo de vida:é o intervalo de tempo desde a criação da variável (alocação de espaço) até a sua destruição.

Tempo de vida:é o intervalo de tempo desde a criação da variável (alocação de espaço) até a sua destruição.

- O tempo de vida de uma variável *global* é o tempo total de execução do programa.

Tempo de vida:é o intervalo de tempo desde a criação da variável (alocação de espaço) até a sua destruição.

- O tempo de vida de uma variável *global* é o tempo total de execução do programa.

- Já o de uma variável *local*, é o período durante o qual está ativo o módulo a que ela pertence.

Tempo de vida:é o intervalo de tempo desde a criação da variável (alocação de espaço) até a sua destruição.

- O tempo de vida de uma variável *global* é o tempo total de execução do programa.

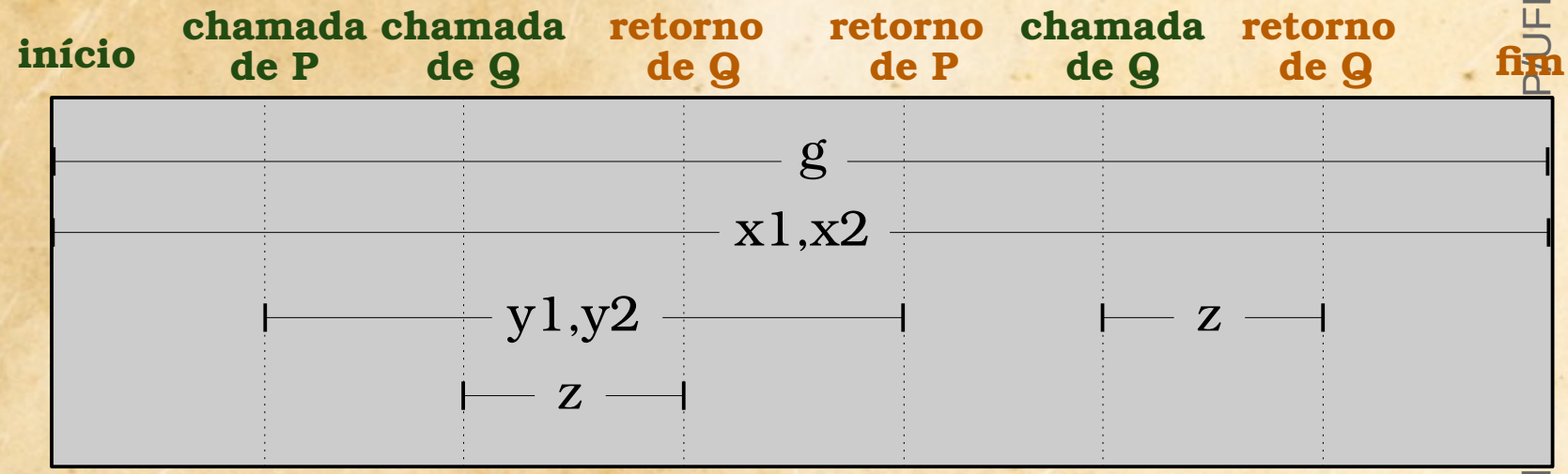
- Já o de uma variável *local*, é o período durante o qual está ativo o módulo a que ela pertence.

O conteúdo de uma variável local se perde após a execução do módulo correspondente.

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

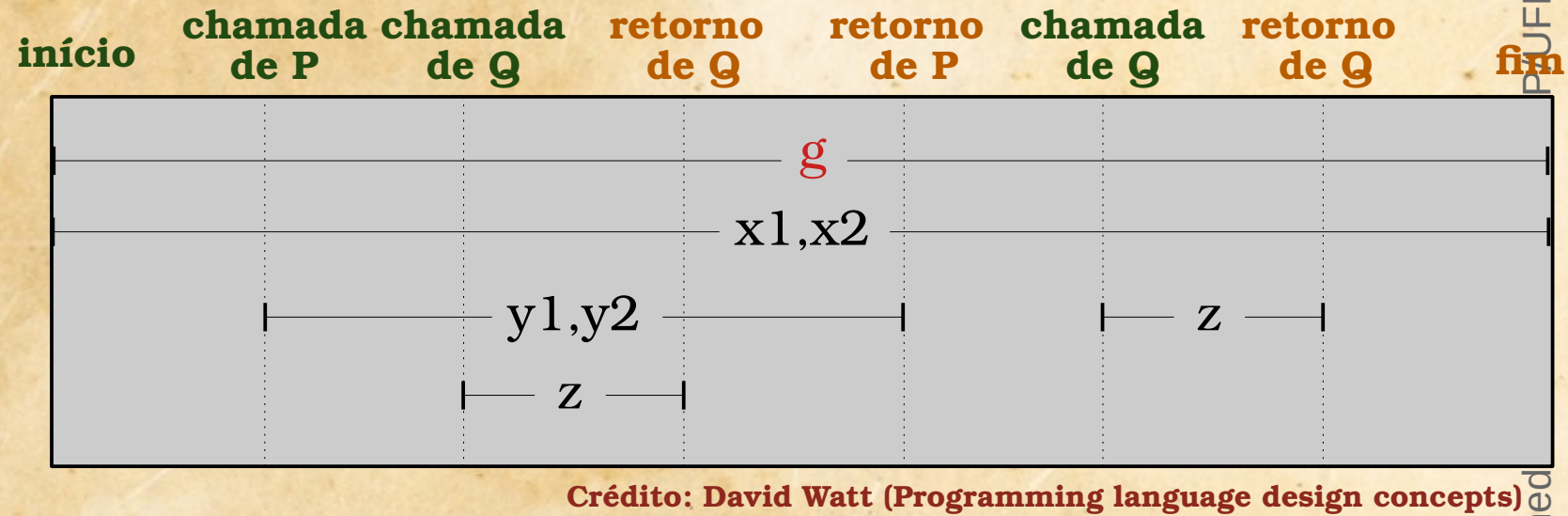
Tempo de vida



Crédito: David Watt (Programming language design concepts)

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida

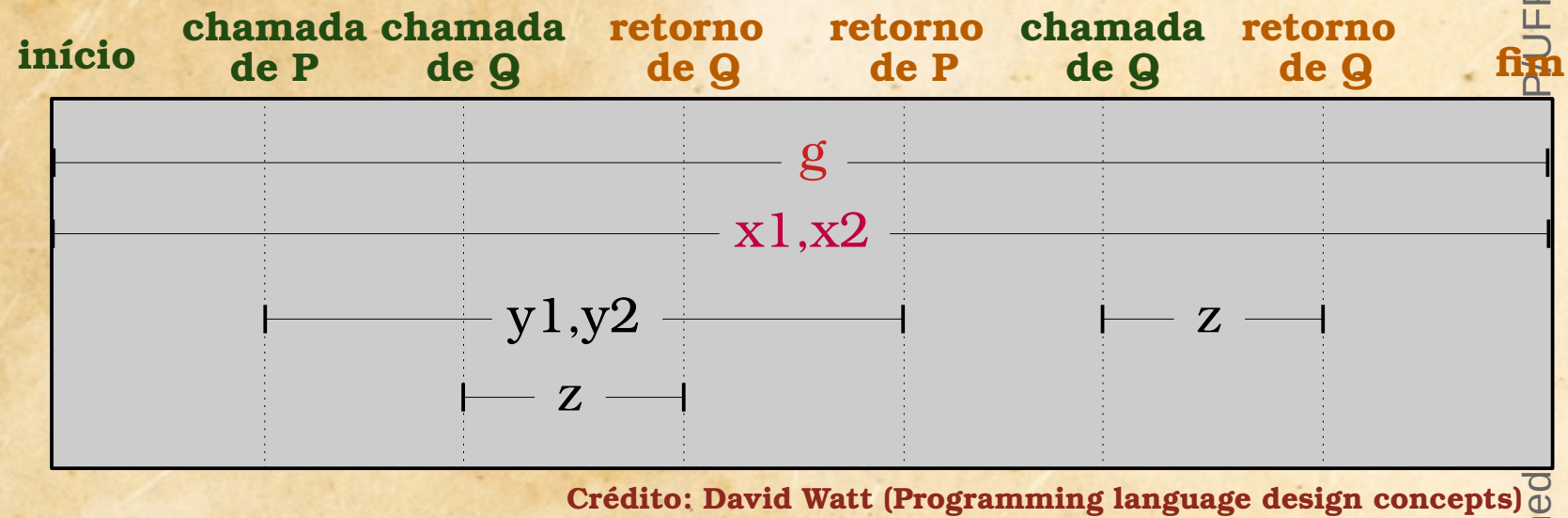


Crédito: David Watt (Programming language design concepts)

Pilha de alocação

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



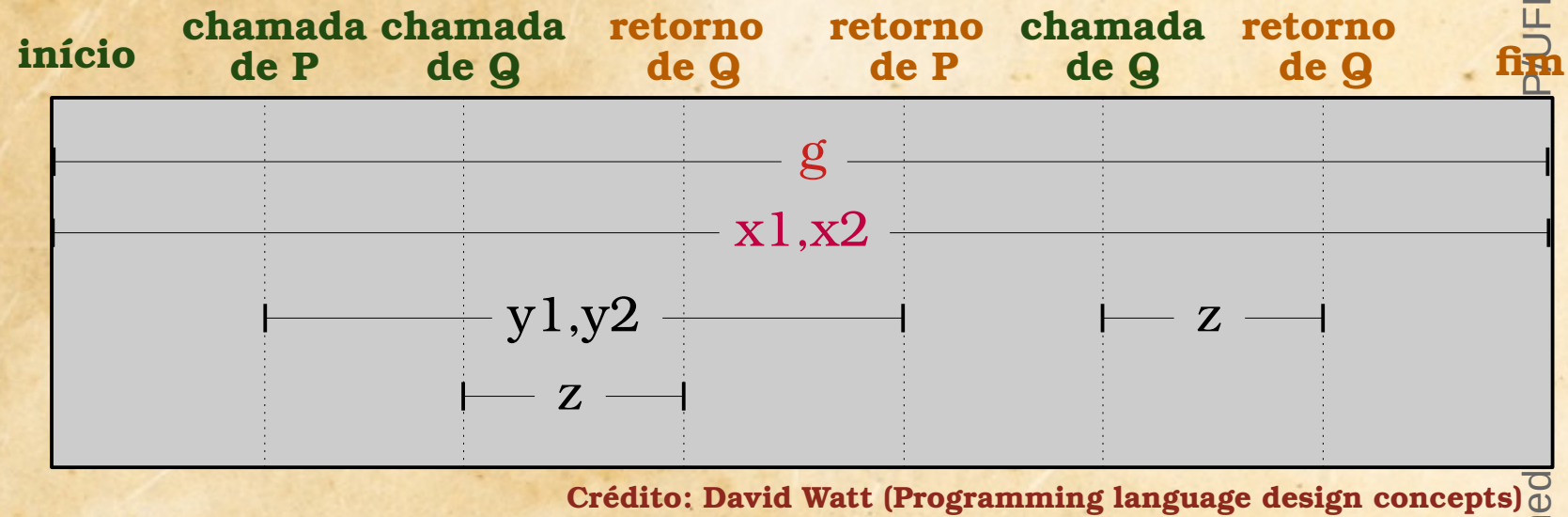
Crédito: David Watt (Programming language design concepts)

Pilha de alocação

g x1 x2

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



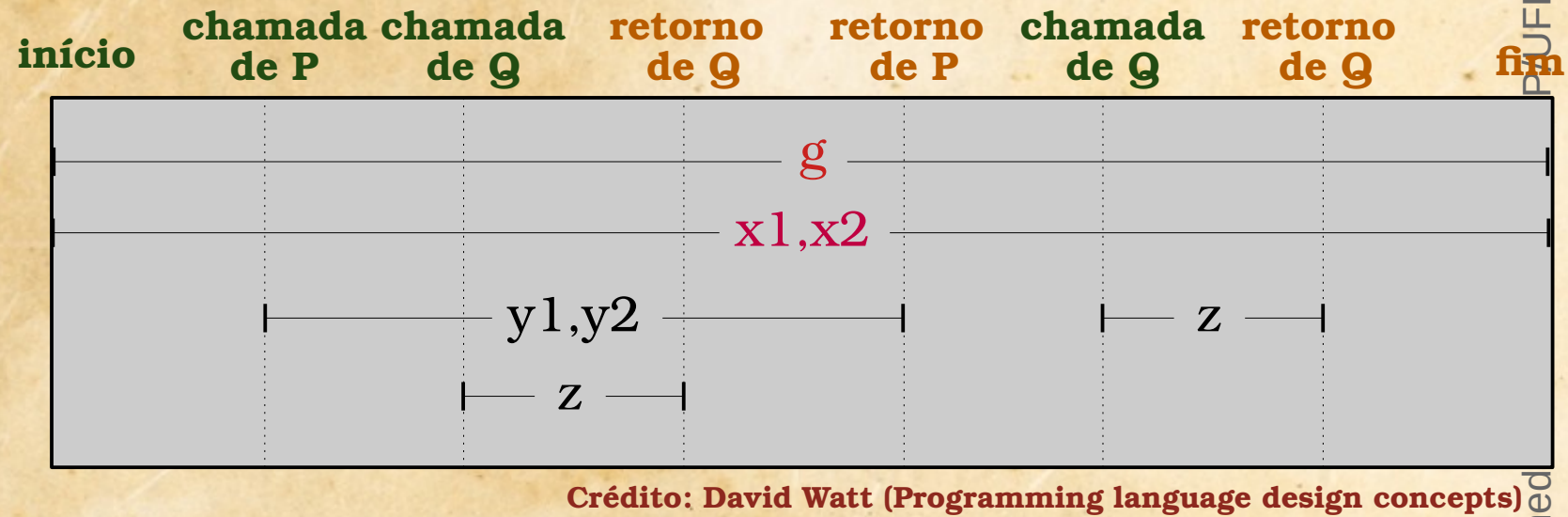
Crédito: David Watt (Programming language design concepts)

Pilha de alocação

g x1 x2

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



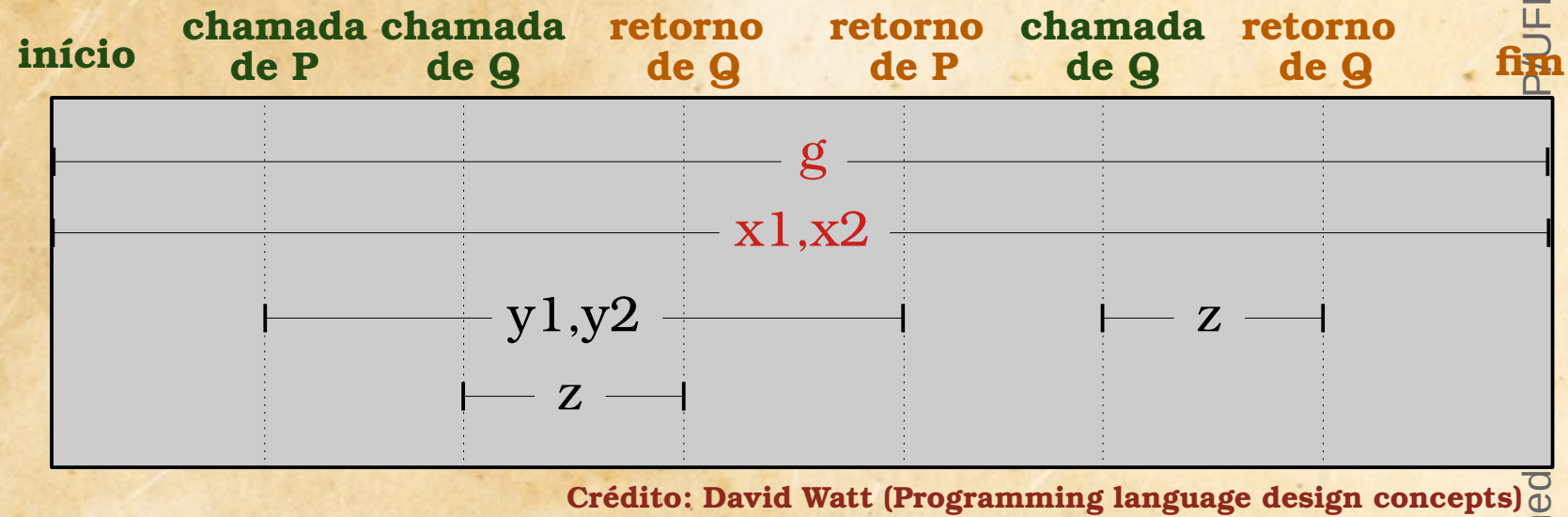
Crédito: David Watt (Programming language design concepts)

Pilha de alocação

g x1 x2

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



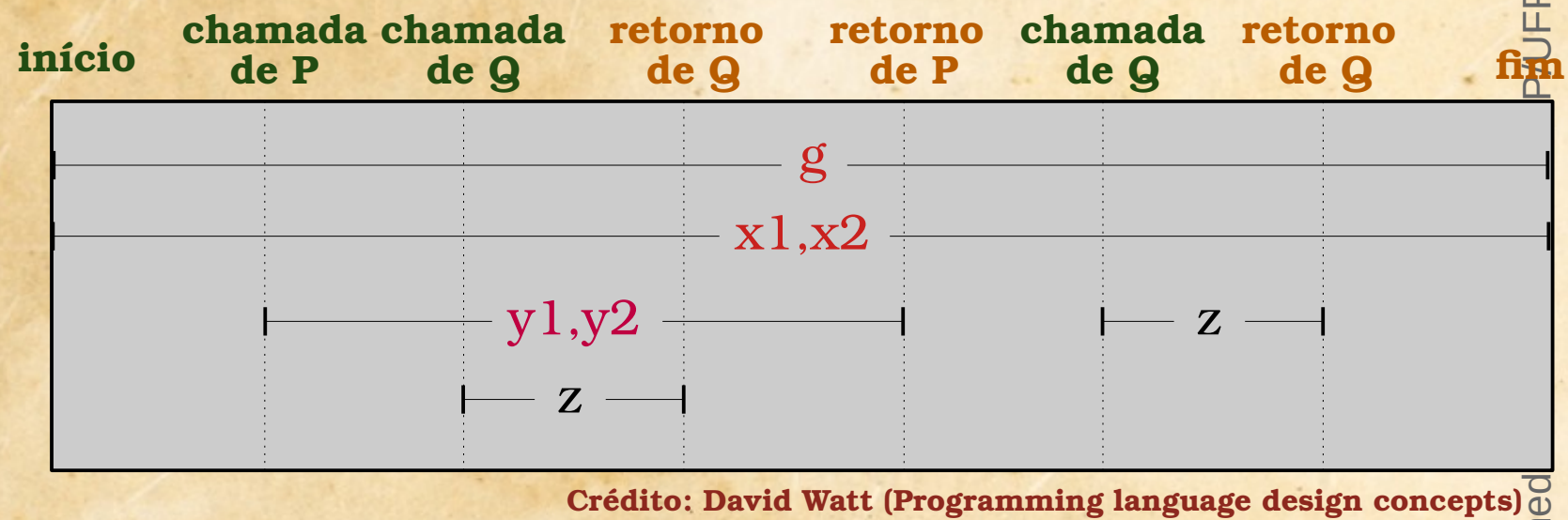
Crédito: David Watt (Programming language design concepts)

Pilha de alocação

g x1 x2

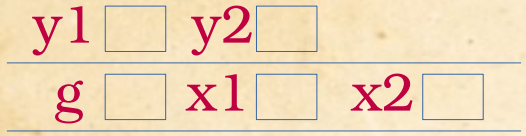
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



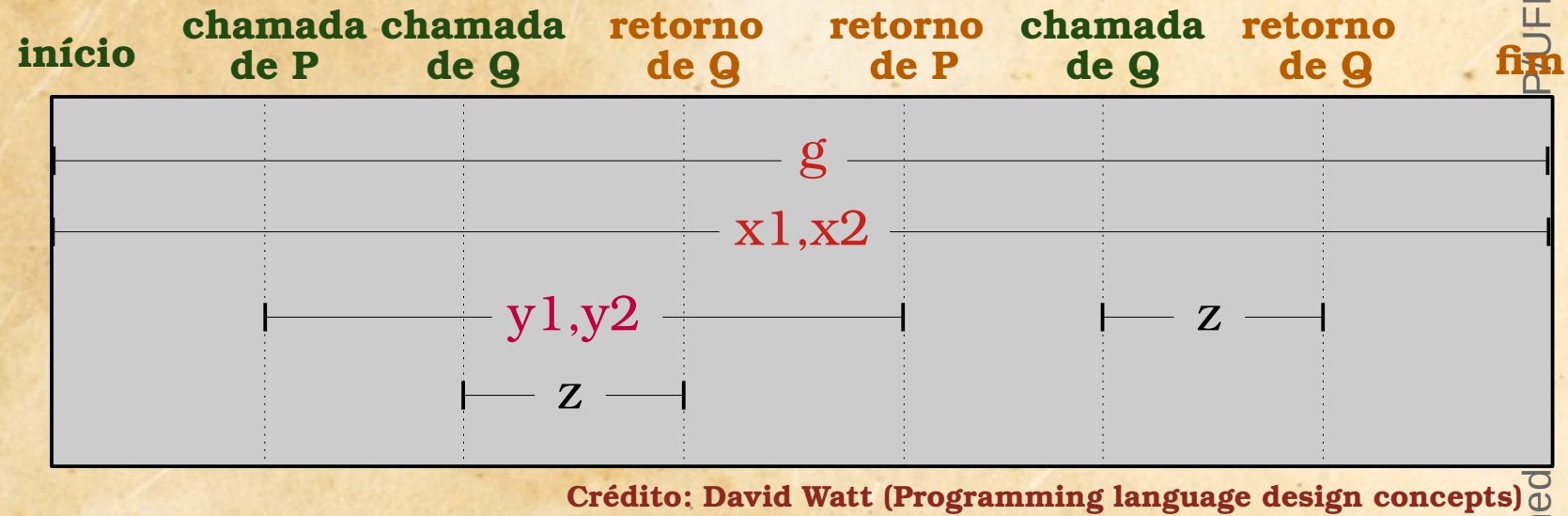
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



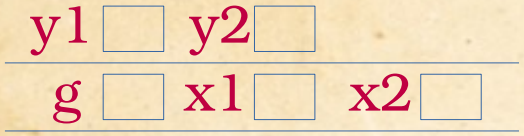
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



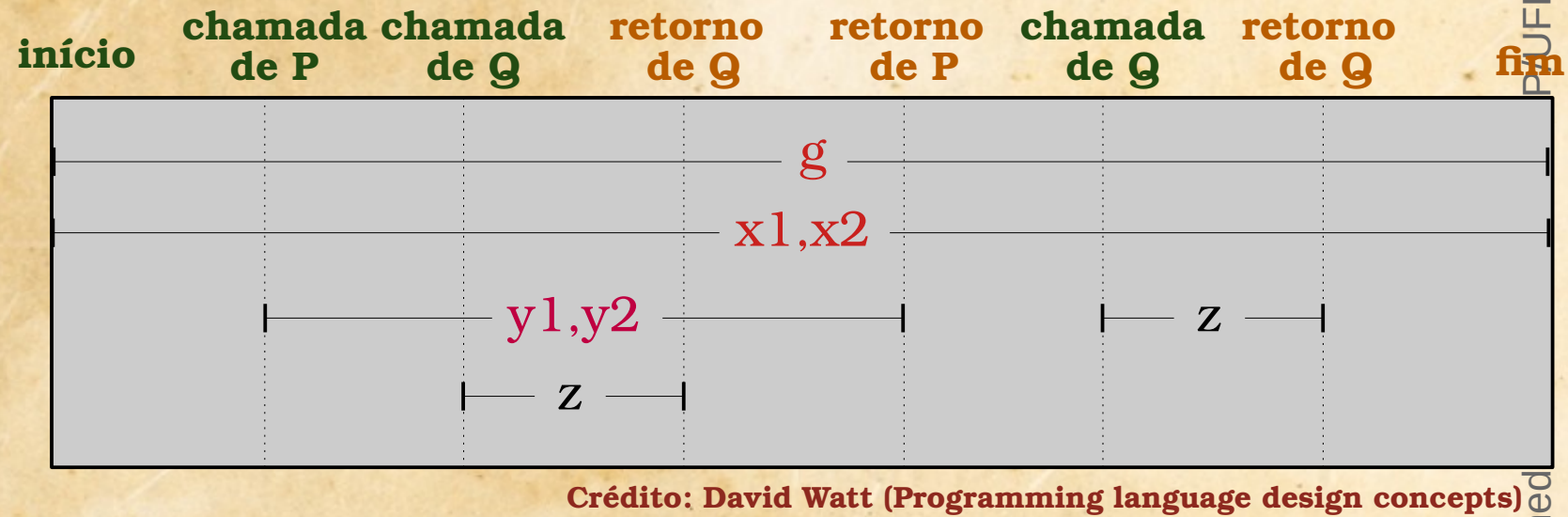
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



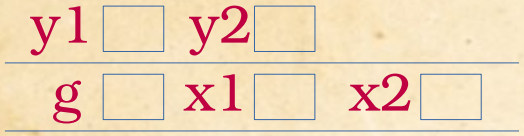
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



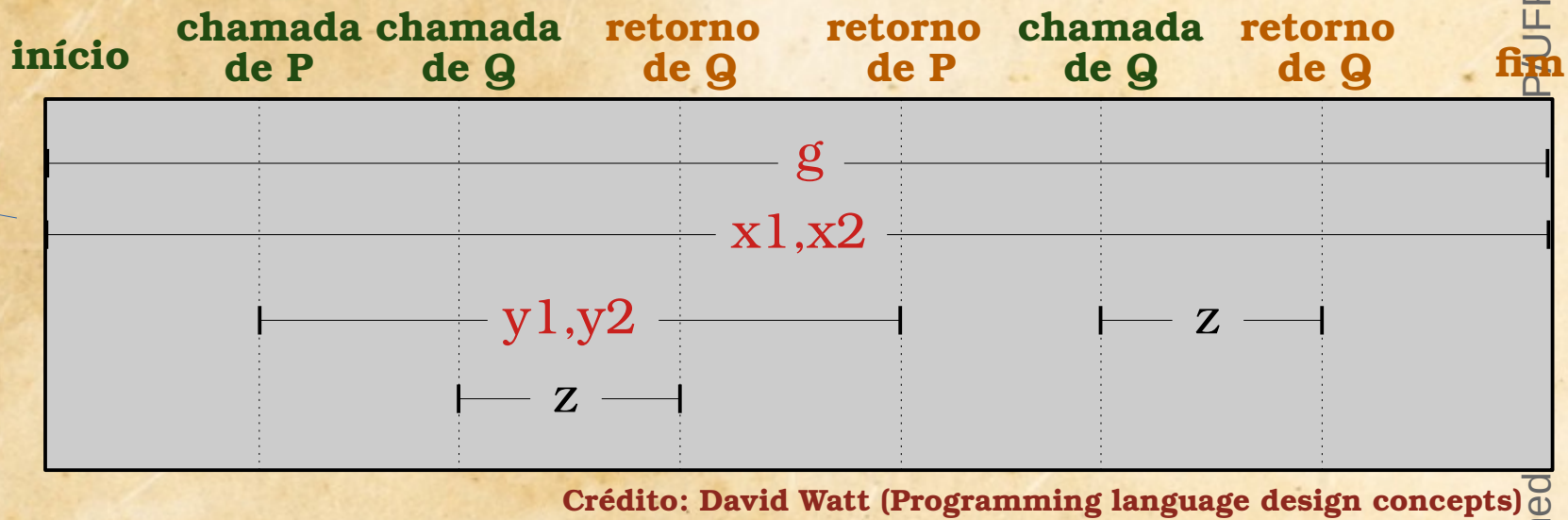
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



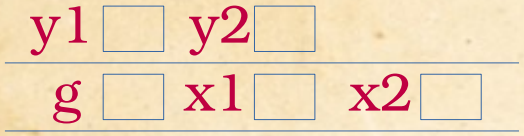
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



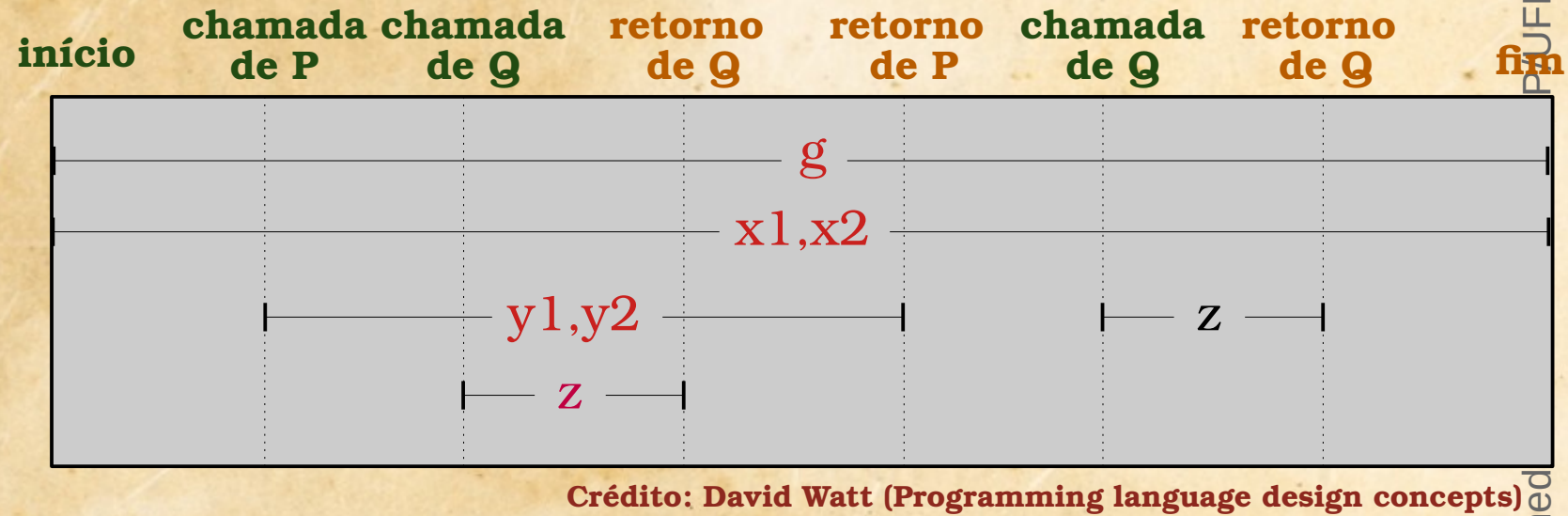
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



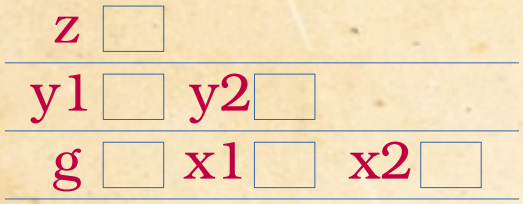
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



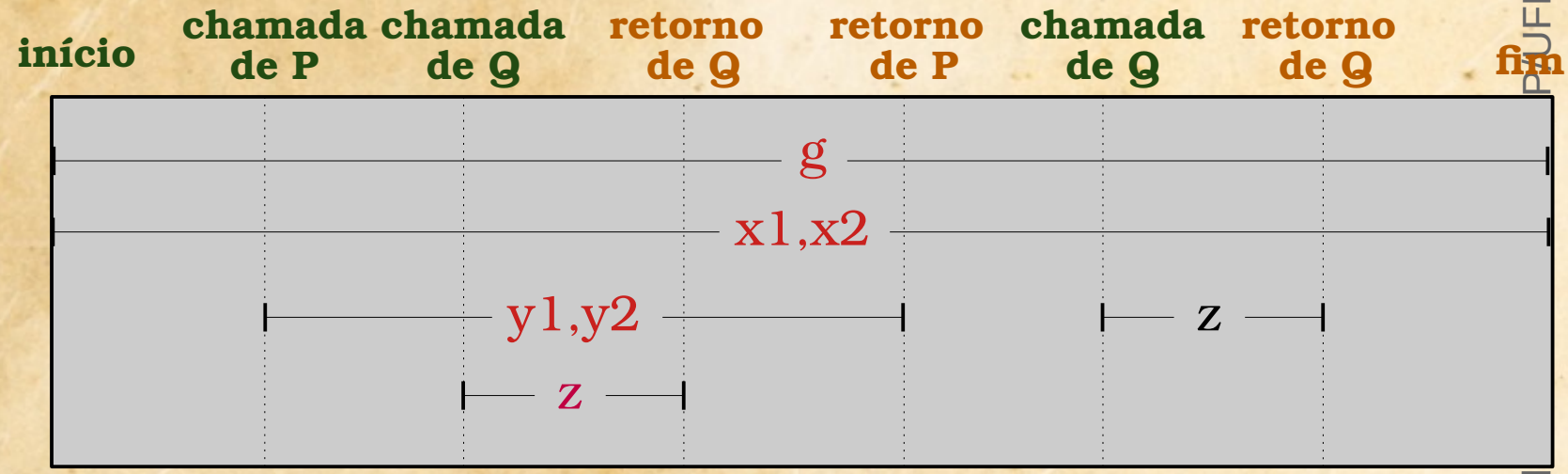
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



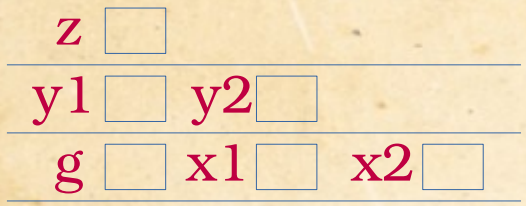
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



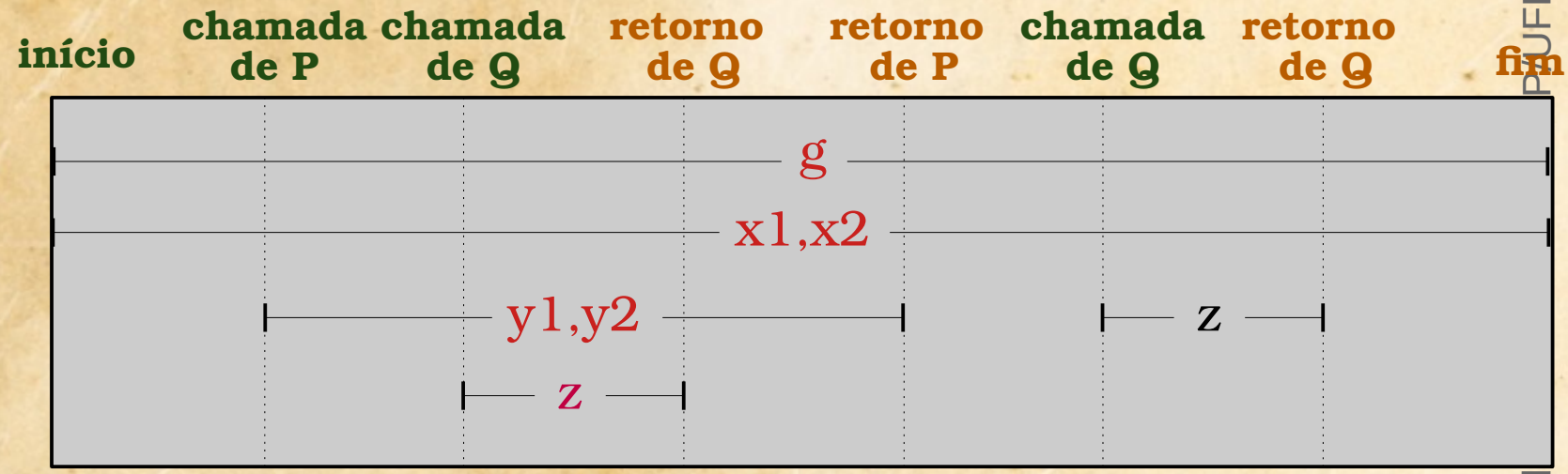
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



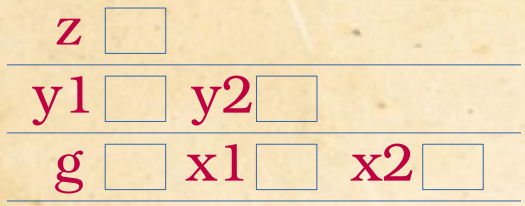
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



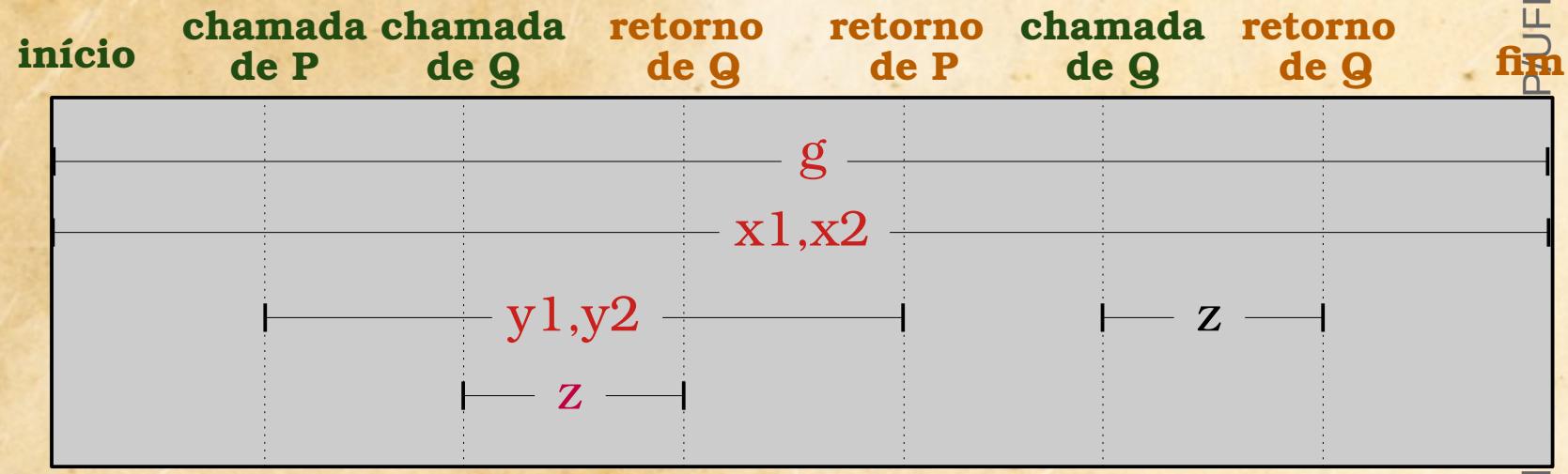
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



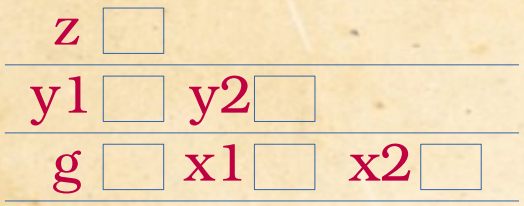
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



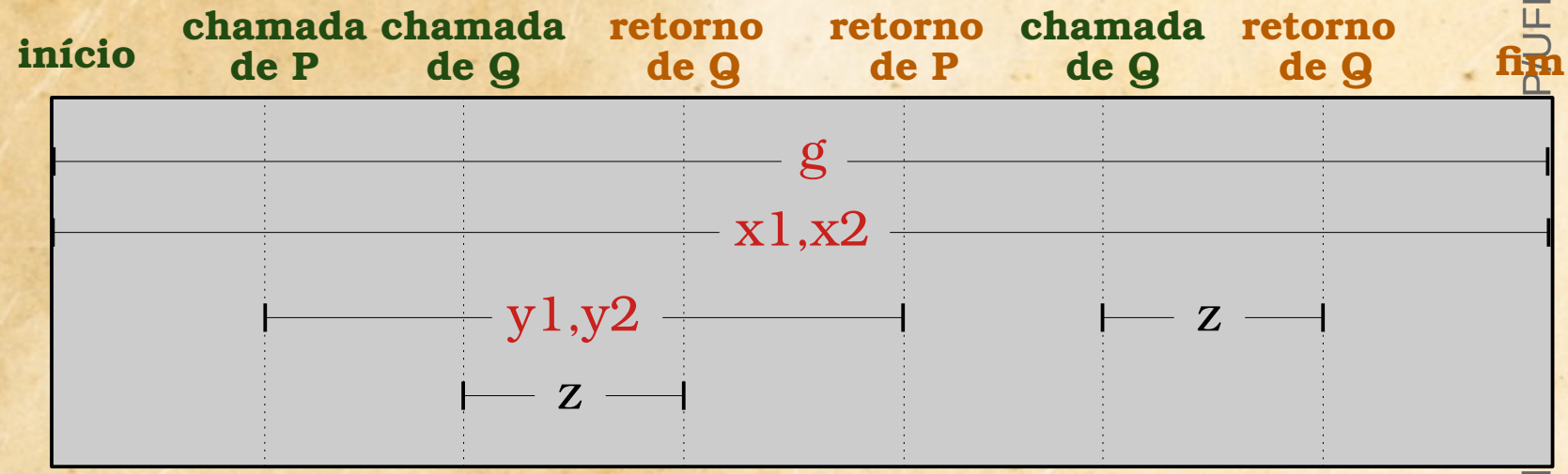
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



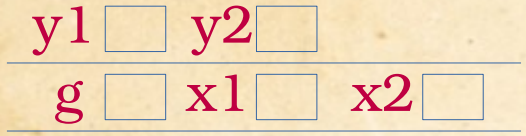
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



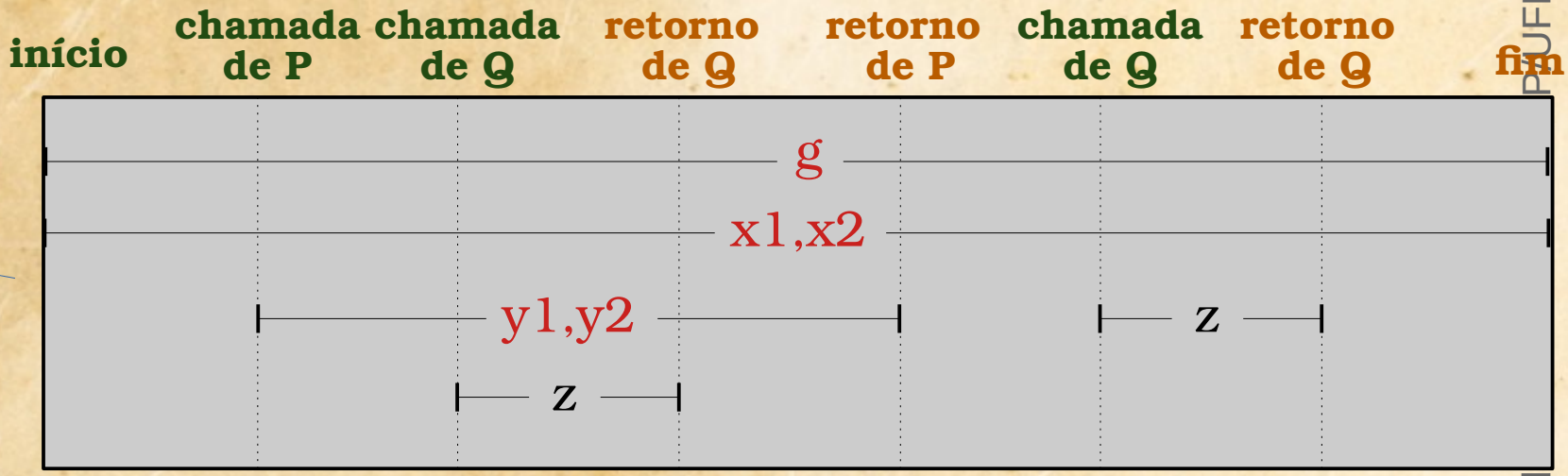
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



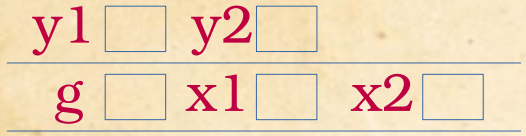
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



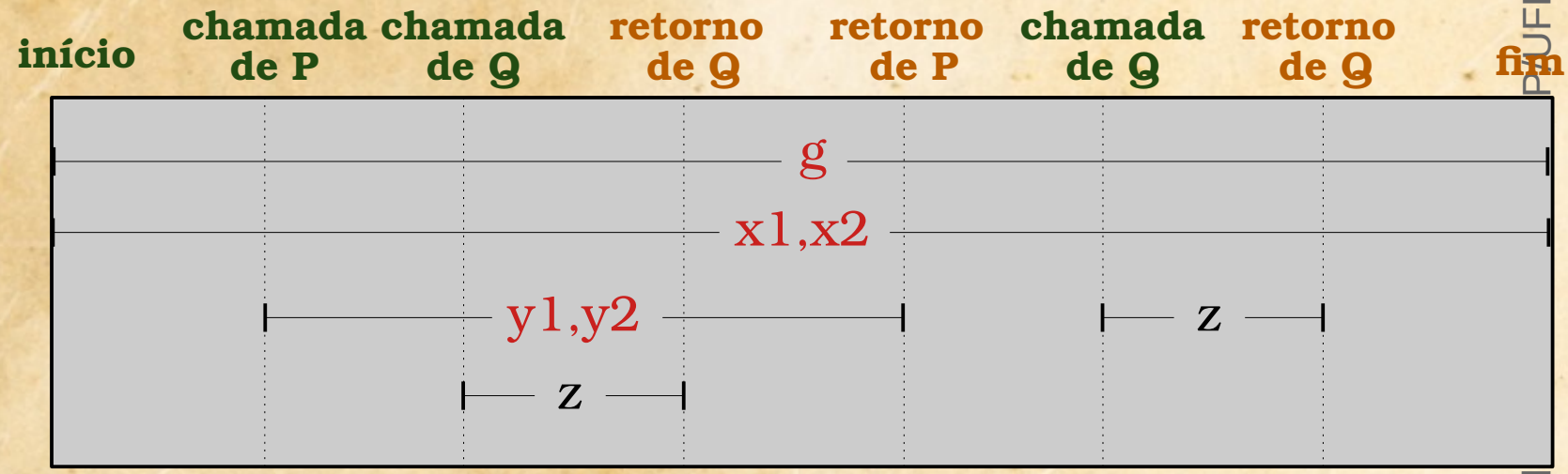
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



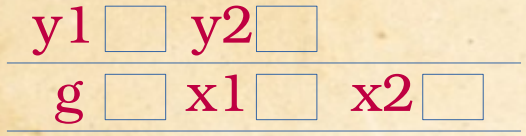
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



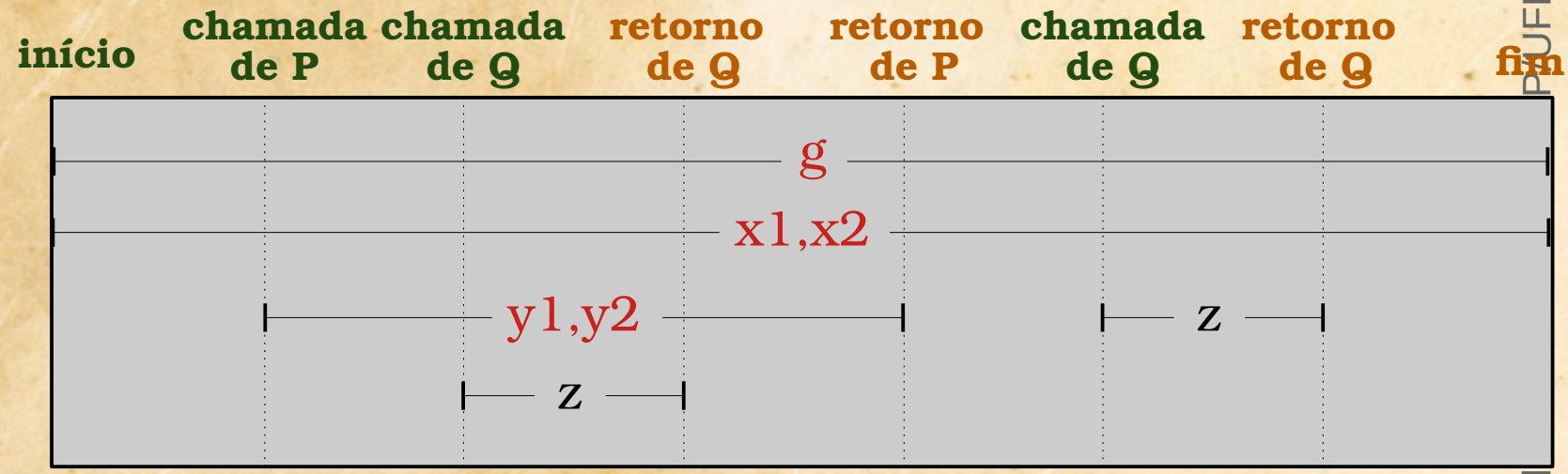
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



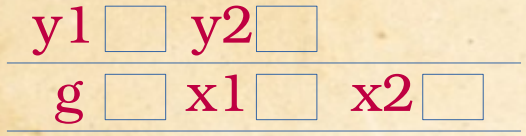
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



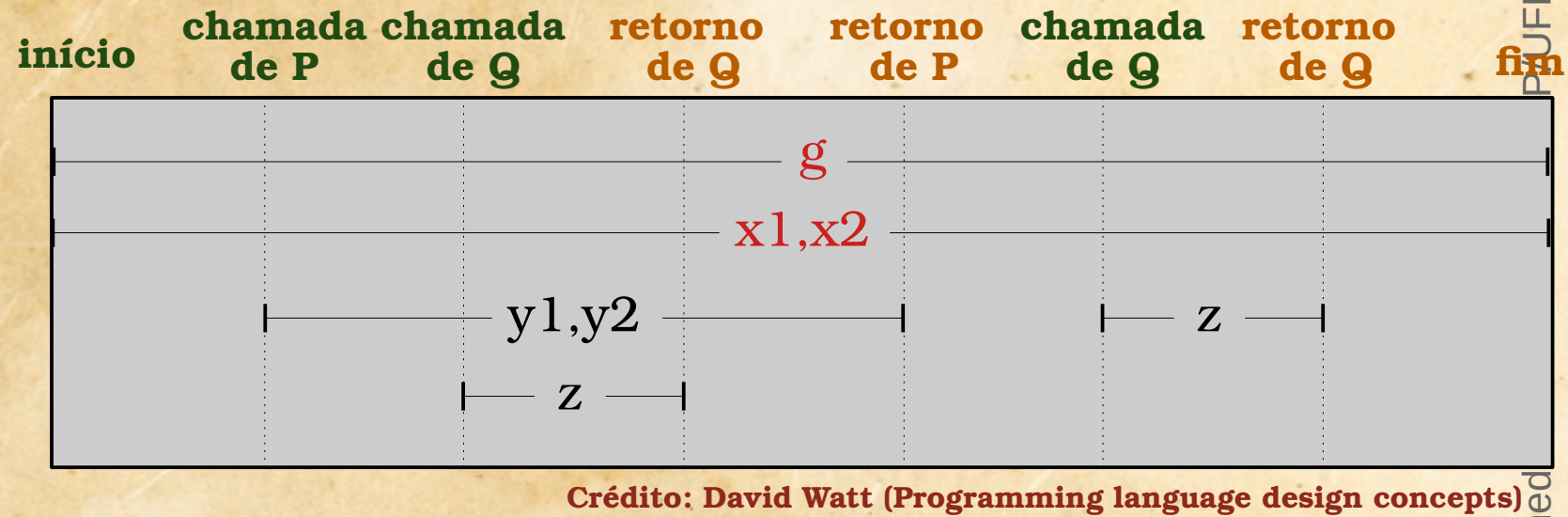
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida

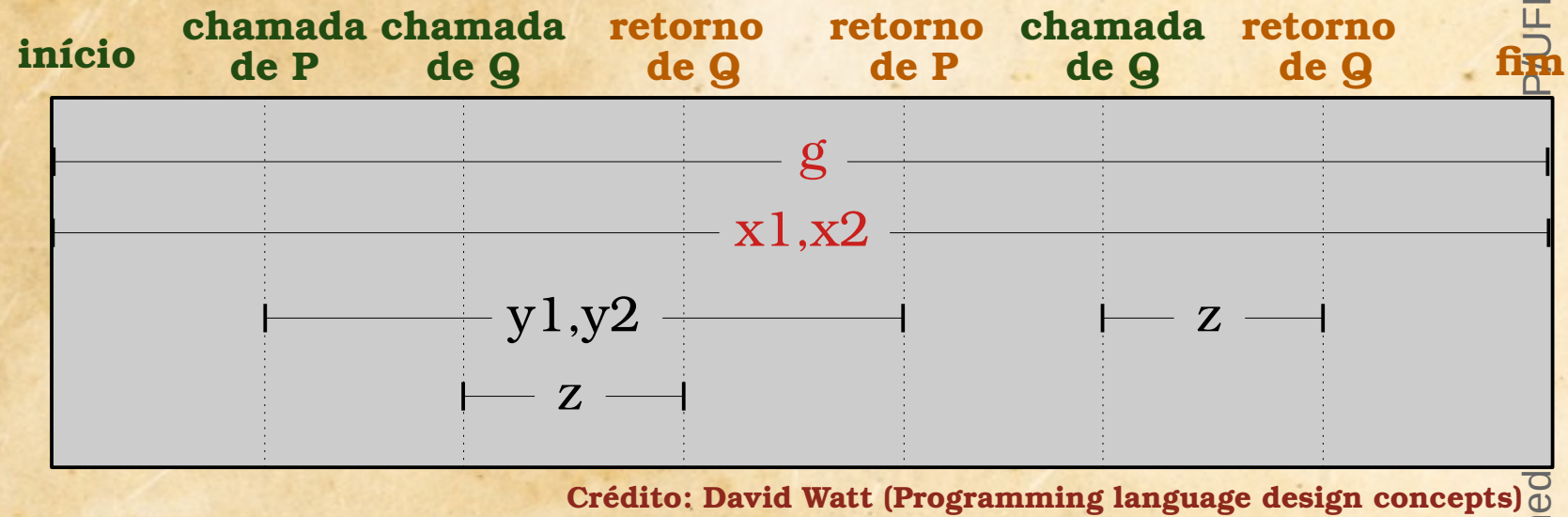


Crédito: David Watt (Programming language design concepts)

Pilha de alocação

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida

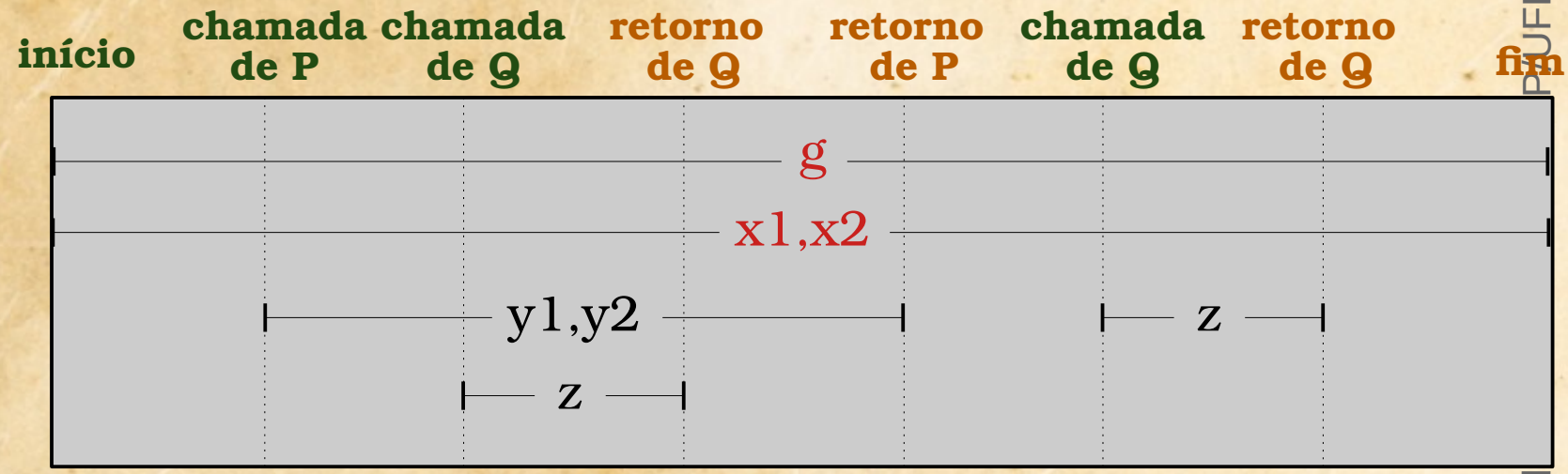


Crédito: David Watt (Programming language design concepts)

Pilha de alocação

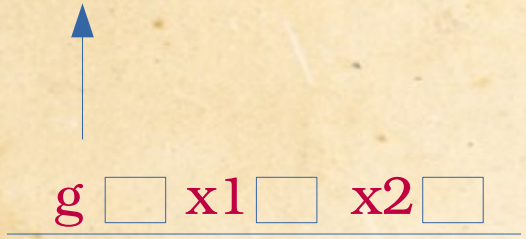
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



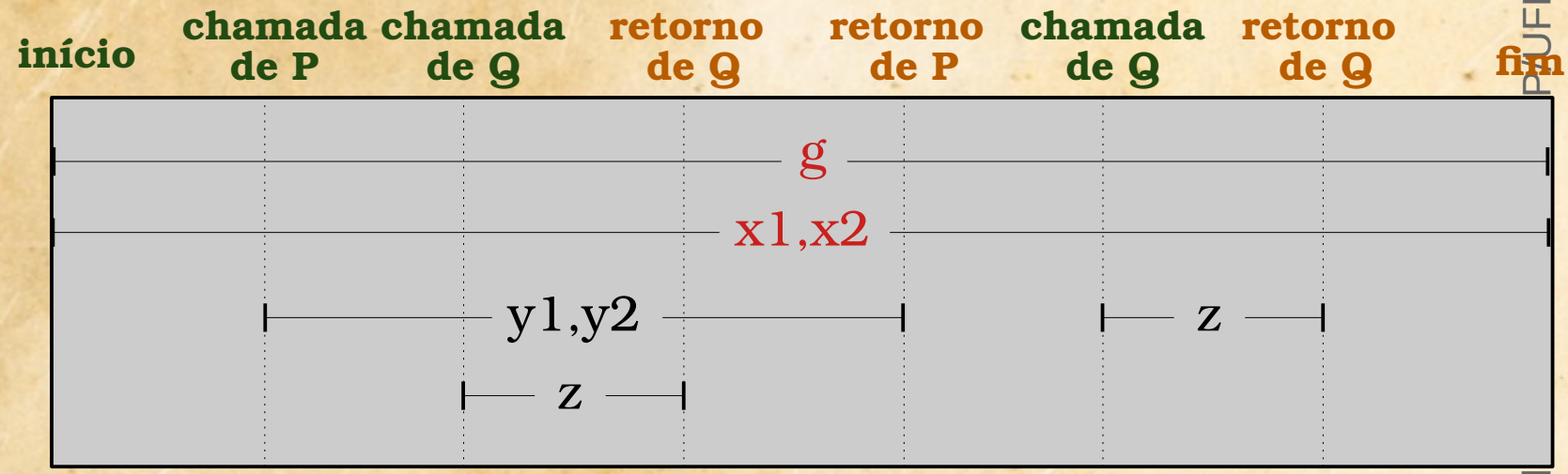
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida

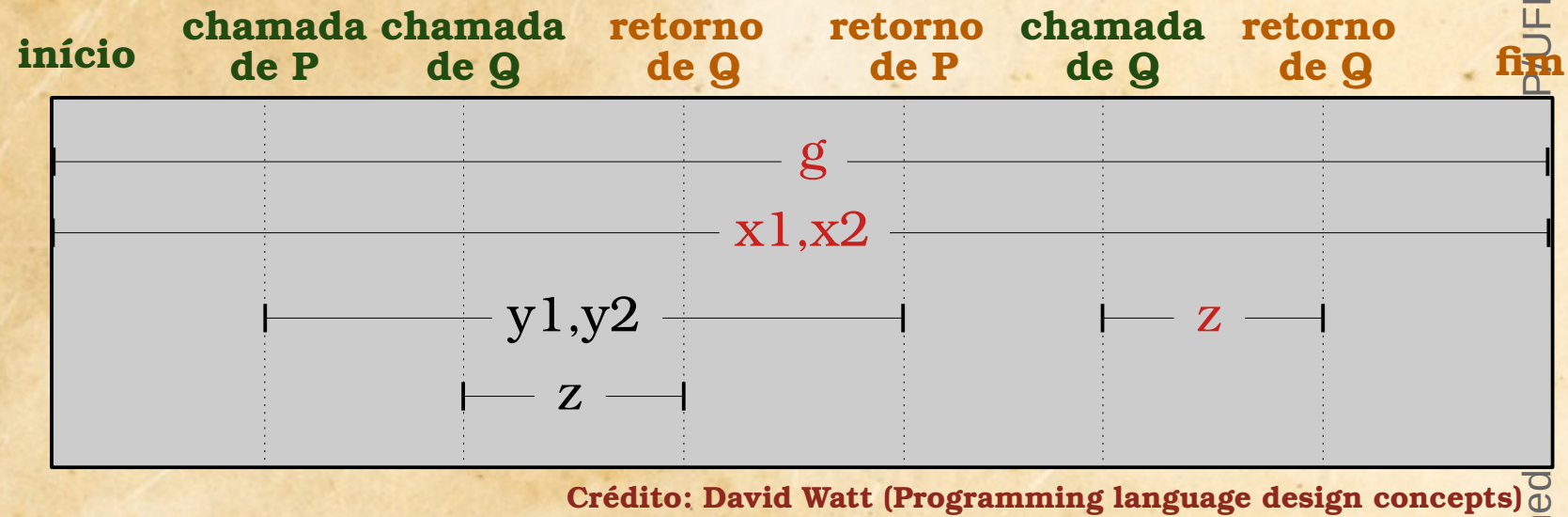


Crédito: David Watt (Programming language design concepts)

Pilha de alocação

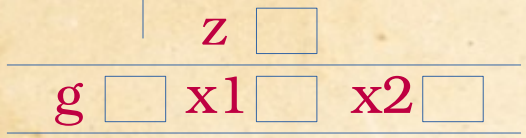
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



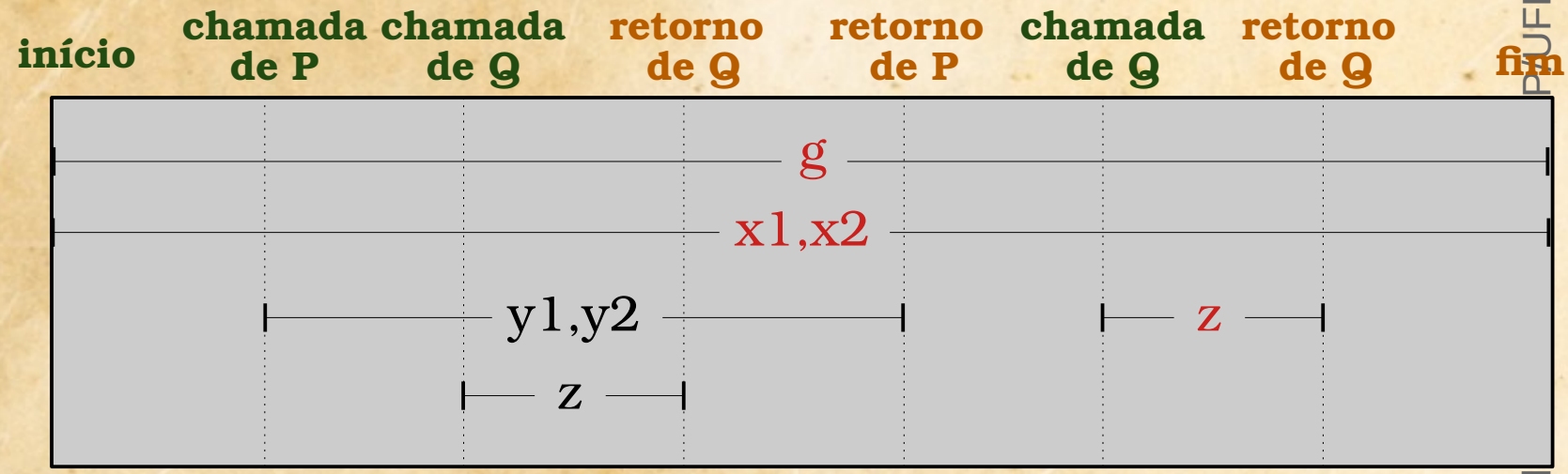
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



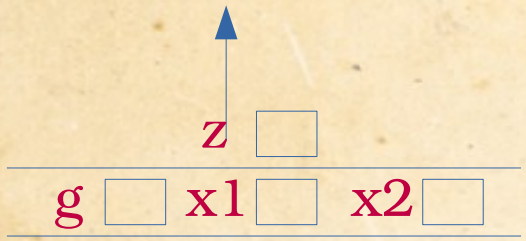
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



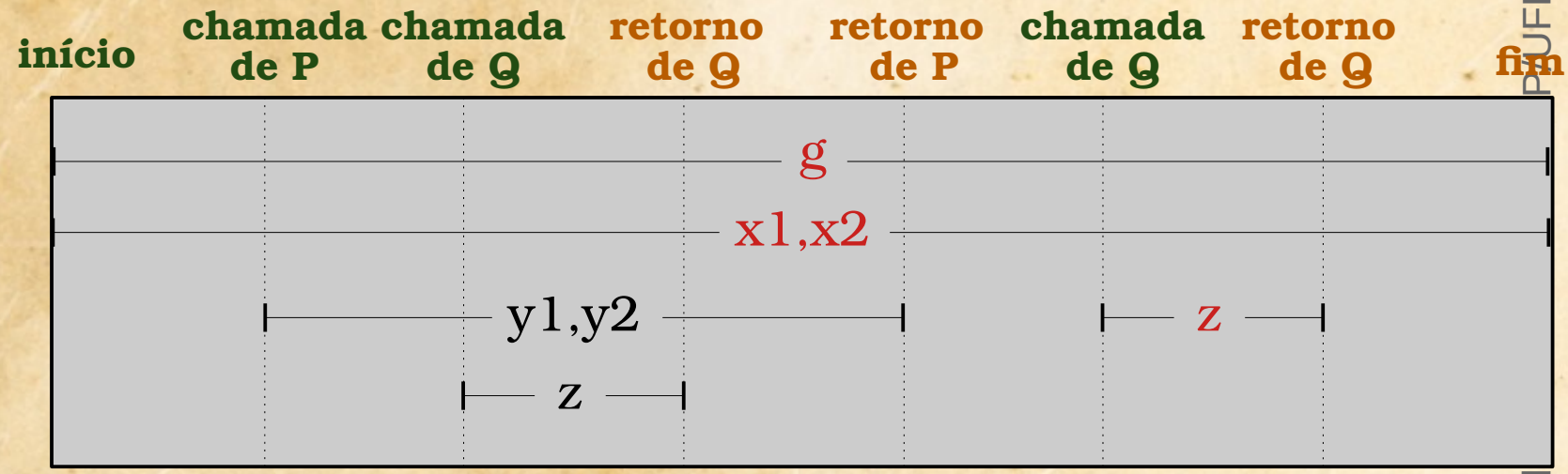
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



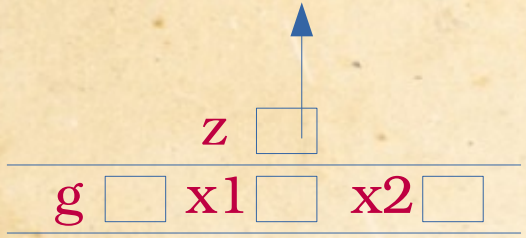
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



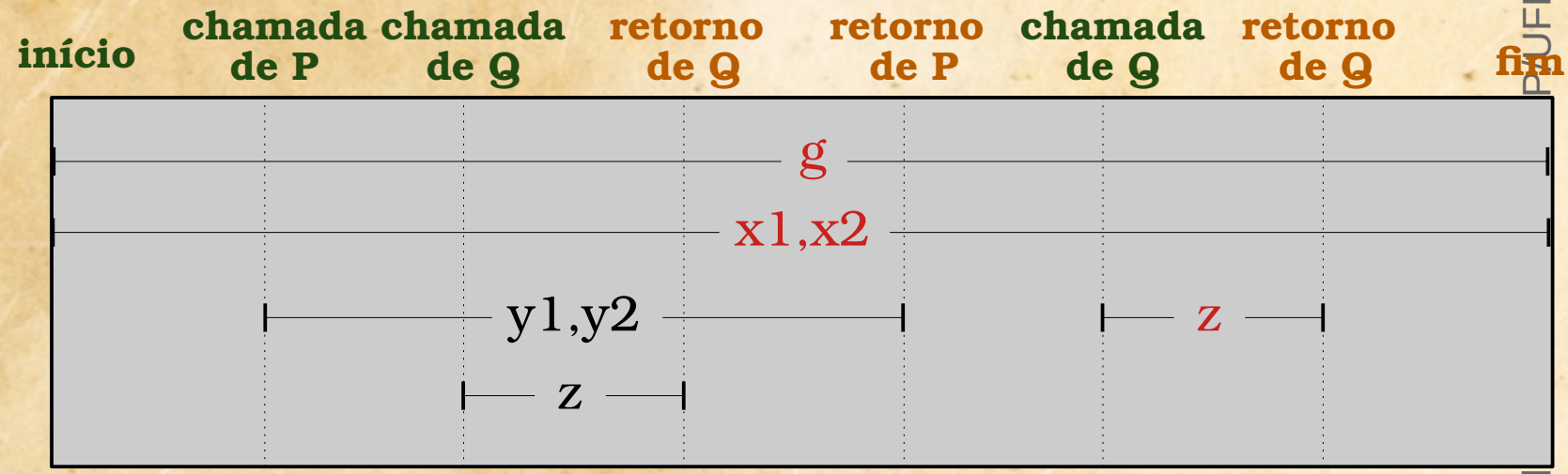
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



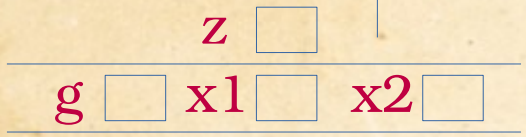
```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



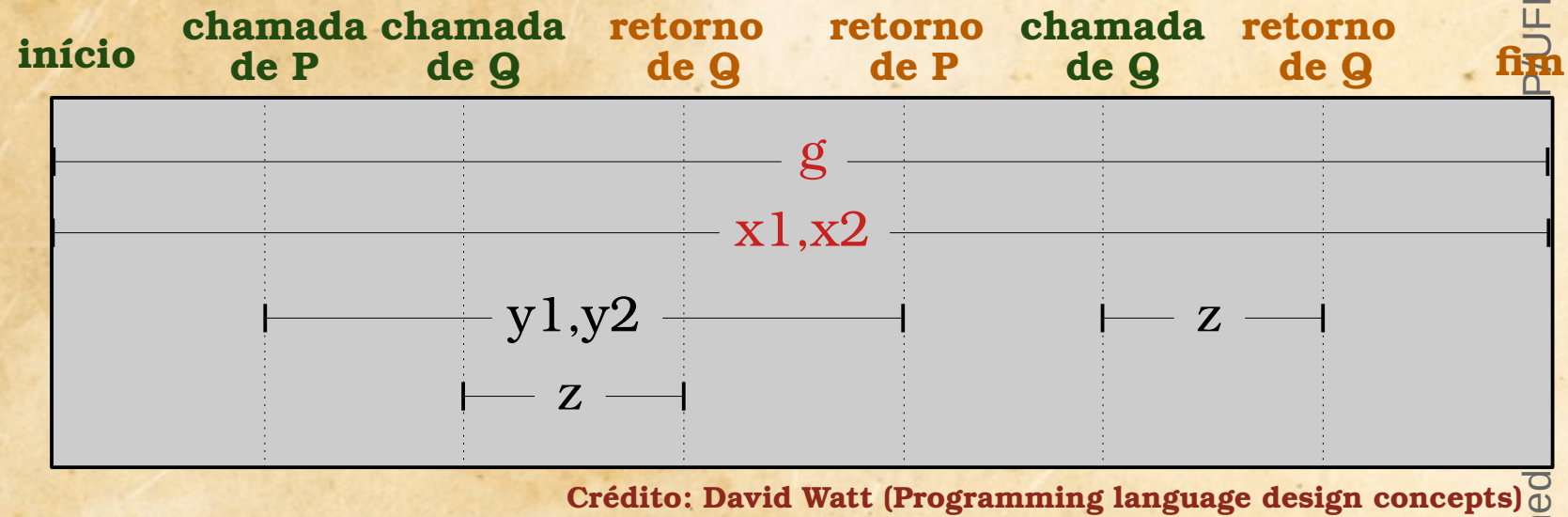
Crédito: David Watt (Programming language design concepts)

Pilha de alocação



```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida

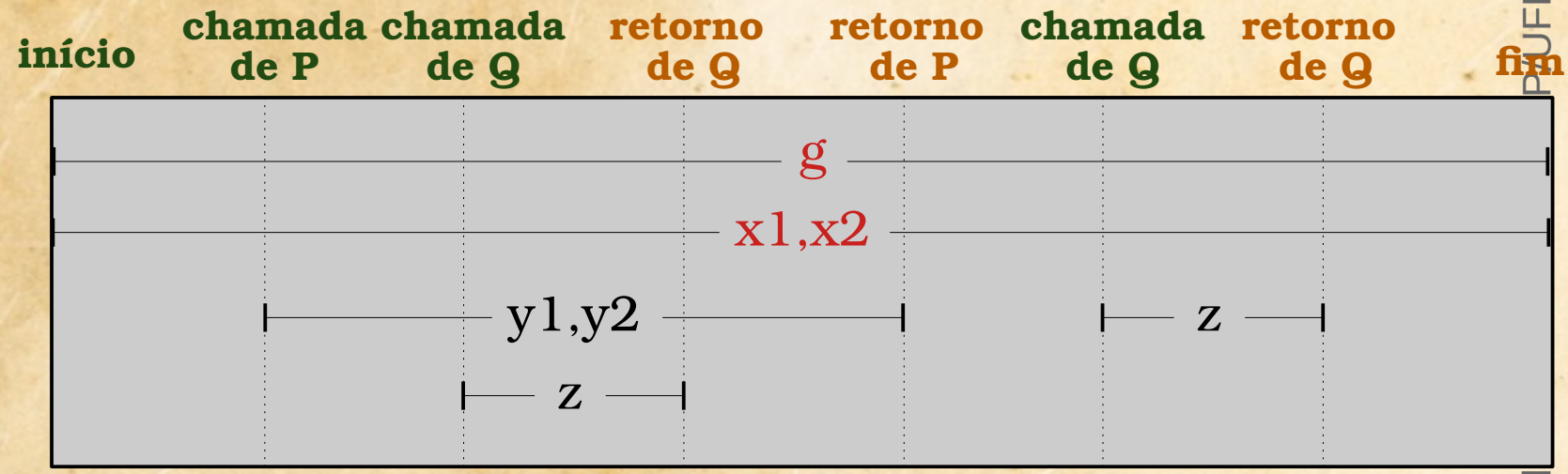


Crédito: David Watt (Programming language design concepts)

Pilha de alocação

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida

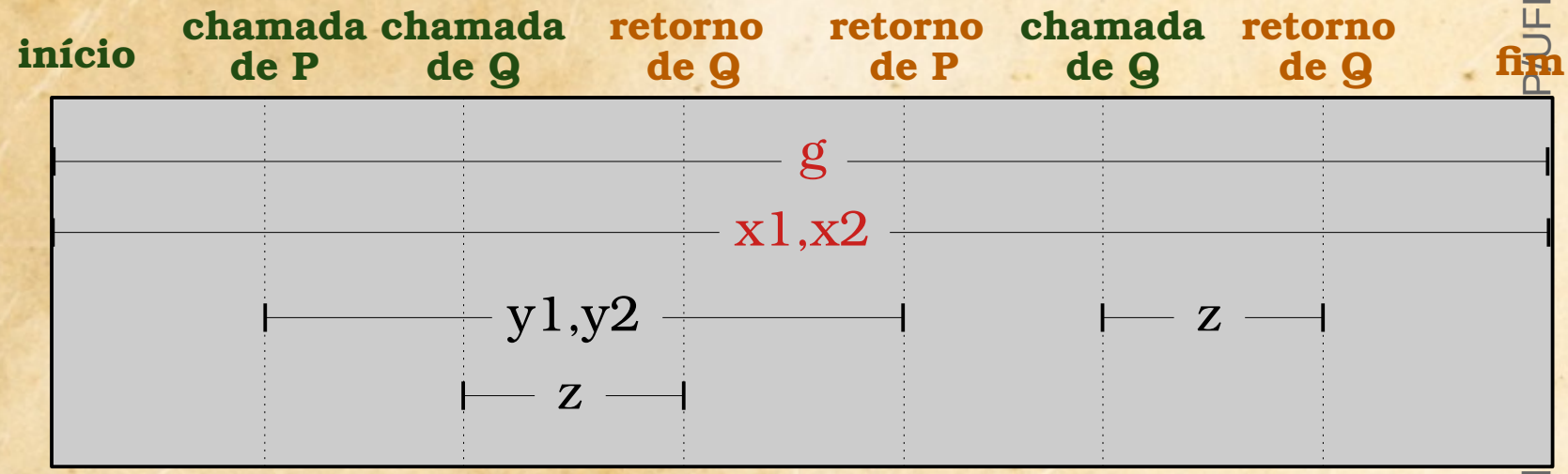


Crédito: David Watt (Programming language design concepts)

Pilha de alocação

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida

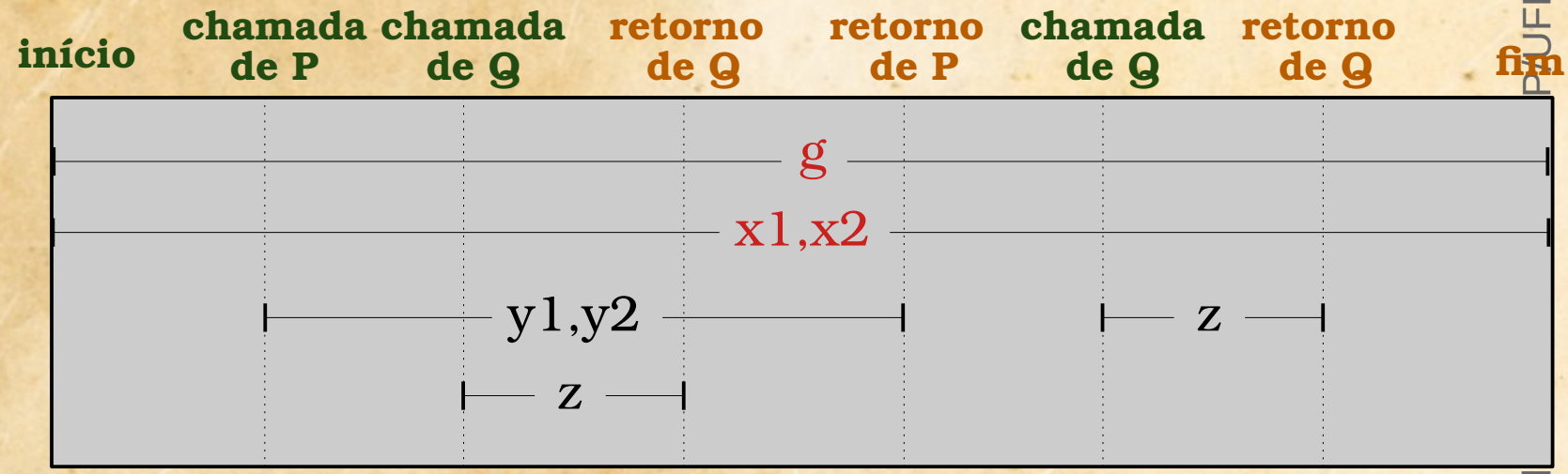


Crédito: David Watt (Programming language design concepts)

Pilha de alocação

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



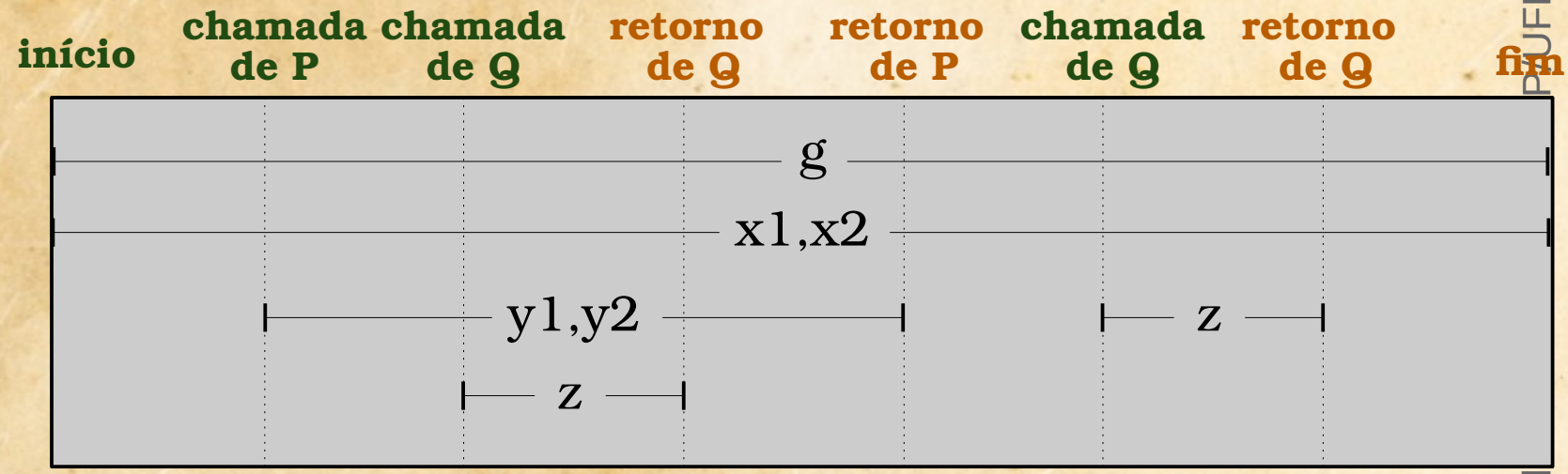
Crédito: David Watt (Programming language design concepts)

Pilha de alocação

g ☐ x1 ☐ x2 ☐

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Tempo de vida



Crédito: David Watt (Programming language design concepts)

Pilha de alocação

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Escopo e tempo de vida são conceitos diretamente relacionados:

→ uma variável não pode ser utilizada fora do seu escopo

→ uma variável só tem *existência efetiva* durante seu tempo de vida.

```
int g;
void Q( ){
    int z;
    ...
}
void P( ){
    float y1;
    int y2:
    ...
    Q( );
    ...
}
int main( ){
   int x1;
   float x2;
   ...
   P( );
   ...
   Q( );
}
```

Entretanto...

x1 e x2 não podem ser referenciadas em P ou Q
(mesmo estando ativas)

Uso de variáveis globais:

Atenção!

Uso de variáveis globais:

Atenção!

Como elas podem ser alteradas de qualquer ponto do programa, podem ser gerados **efeitos colaterais.**

```
int i;
void M2( ){
   ...
   for(i=0; i<5; i++) {
      ...
   }
}
void M1( ){
    ...
    M2( );
    ...
}
int main( ){
   ...
   for(i=0; i<100; i++) {
      ...
      M1( );
      ...
   }
}
```

```
int i;
void M2( ){
   ...
   for(i=0; i<5; i++) {
      ...
   }
}
void M1( ){
    ...
    M2( );
    ...
}
int main( ){
   ...
   for(i=0; i<100; i++) {
      ...
      M1( );
      ...
   }
}
```

```
int i;
void M2( ){
   ...
   for(i=0; i<5; i++) {
      ...
   }
}
void M1( ){
    ...
    M2( );
    ...
}
int main( ){
   ...
   for(i=0; i<100; i++) {
      ...
      M1( );
      ...
   }
}
```

```
int i;
void M2( ){
   ...
   for(i=0; i<5; i++) { //??
      ...
   }
}
void M1( ){
    ...
    M2( );
    ...
}
int main( ){
   ...
   for(i=0; i<100; i++) {
      ...
      M1( );
      ...
   }
}
```

```
int i;
void M2( ){
   ...
   for(int i=0; i<5; i++) {
      ...
   }
}
void M1( ){
    ...
    M2( );
    ...
}
int main( ){
   ...
   for(int i=0; i<100; i++) {
      ...
      M1( );
      ...
   }
}
```

Tendência das linguagens: desencorajar o uso.

Ex (em Python):

```
def incrVarGlobal( ):
  x=x+1

x=5
incrVarGlobal( )
print(x)
```

Tendência das linguagens: desencorajar o uso.

Ex (em Python):

```
def incrVarGlobal( ):
   x=x+1

x=5
incrVarGlobal( )
print(x)
```

local variable 'x' referenced before assignment

Tendência das linguagens: desencorajar o uso.

Ex (em Python):

```
def incrVarGlobal( ):
   global x
   x=x+1

x=5
incrVarGlobal( )
print(x)      #  6
```

Tendência das linguagens: desencorajar o uso.

Ex (em Python):

```
def atribValVarGlobal( ):
  x=7
  print(x)

x=5
atribValVarGlobal( )
print(x)
```

Tendência das linguagens: desencorajar o uso.

Ex (em Python):

```
def atribValVarGlobal( ):
   x=7
   print(x)   # 7

x=5
atribValVarGlobal( )
print(x)      # 5
```

variáveis distintas

PARÂMETROS E ARGUMENTOS

Prof. Dr. Benedito Ferreira - FACOMP/UFPA

```
int v[100];
...
long soma() {
  long s = 0;
  for(int i=0; i<100; i++)
    s += v[i];
  return s;
}
```

```
int v[100];
...
long soma() {
  long s = 0;
  for(int i=0; i<100; i++)
    s += v[i];
  return s;
}


...
tot = soma();
...
```

soma dos elementos de um vetor em particular **v** (global).

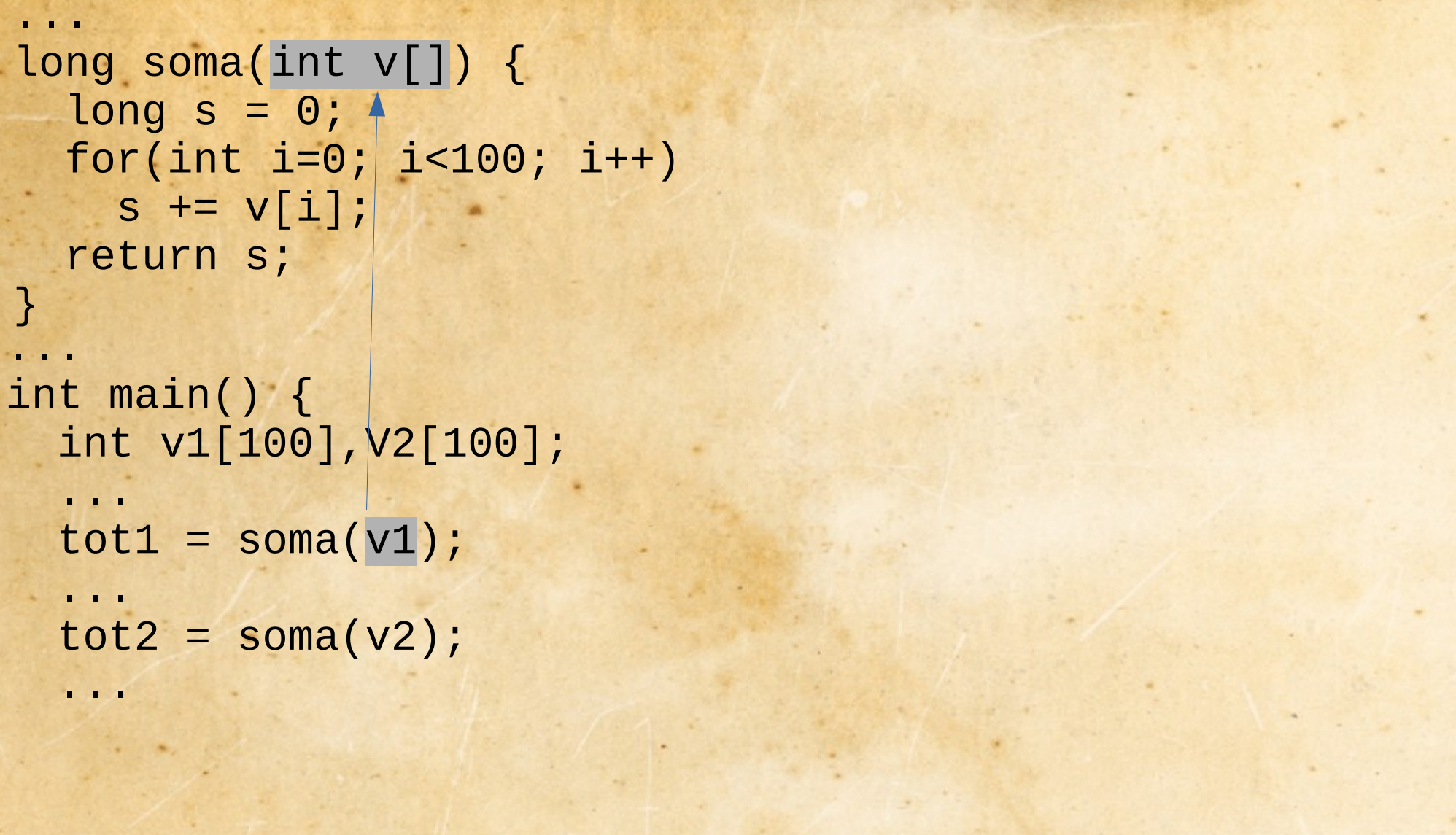
→ utilidade bastante limitada!

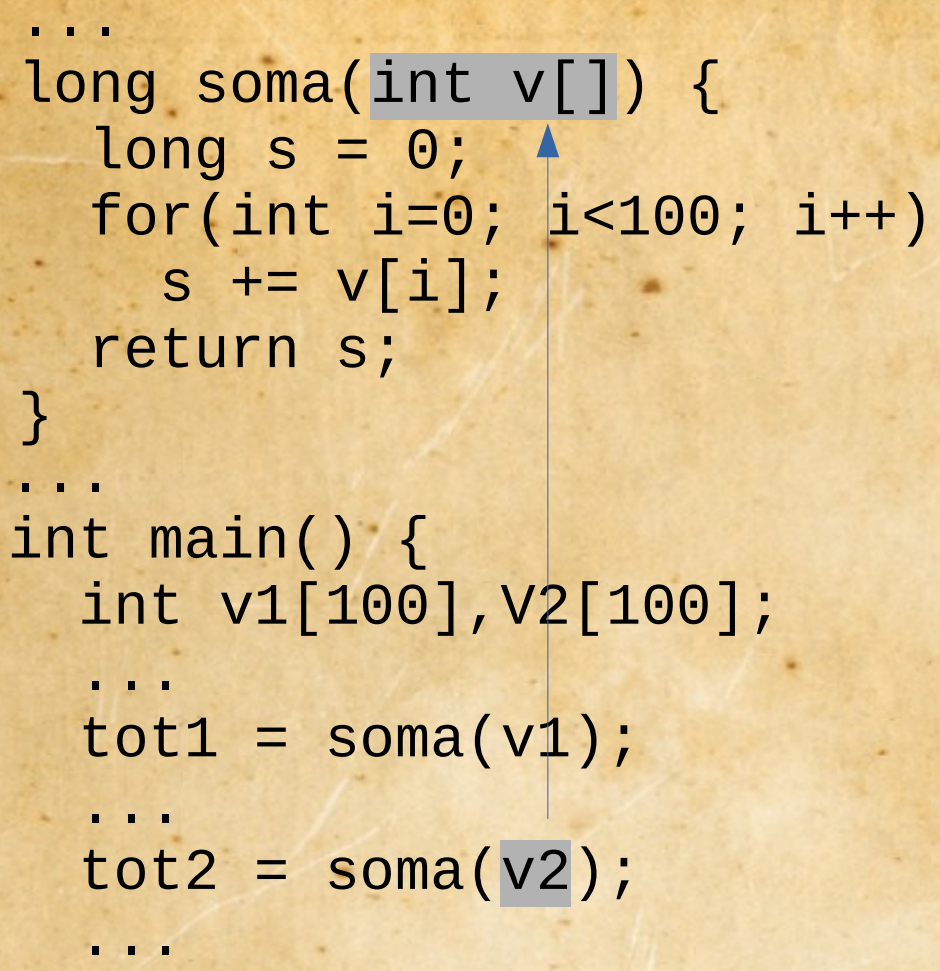
```
...
long soma(int v[]) {
   long s = 0;
   for(int i=0; i<100; i++)
     s += v[i];
   return s;
}
...
int main() {
  int v1[100],V2[100];
  ...
  tot1 = soma(v1);
  ...
  tot2 = soma(v2);
  ...
```

acrescentando um parâmetro.

```
...
long soma(int v[]) {
   long s = 0;
   for(int i=0; i<100; i++)
     s += v[i];
   return s;
}
...
int main() {
  int v1[100],V2[100];
  ...
  tot1 = soma(v1);
  ...
  tot2 = soma(v2);
  ...
```

variáveis locais



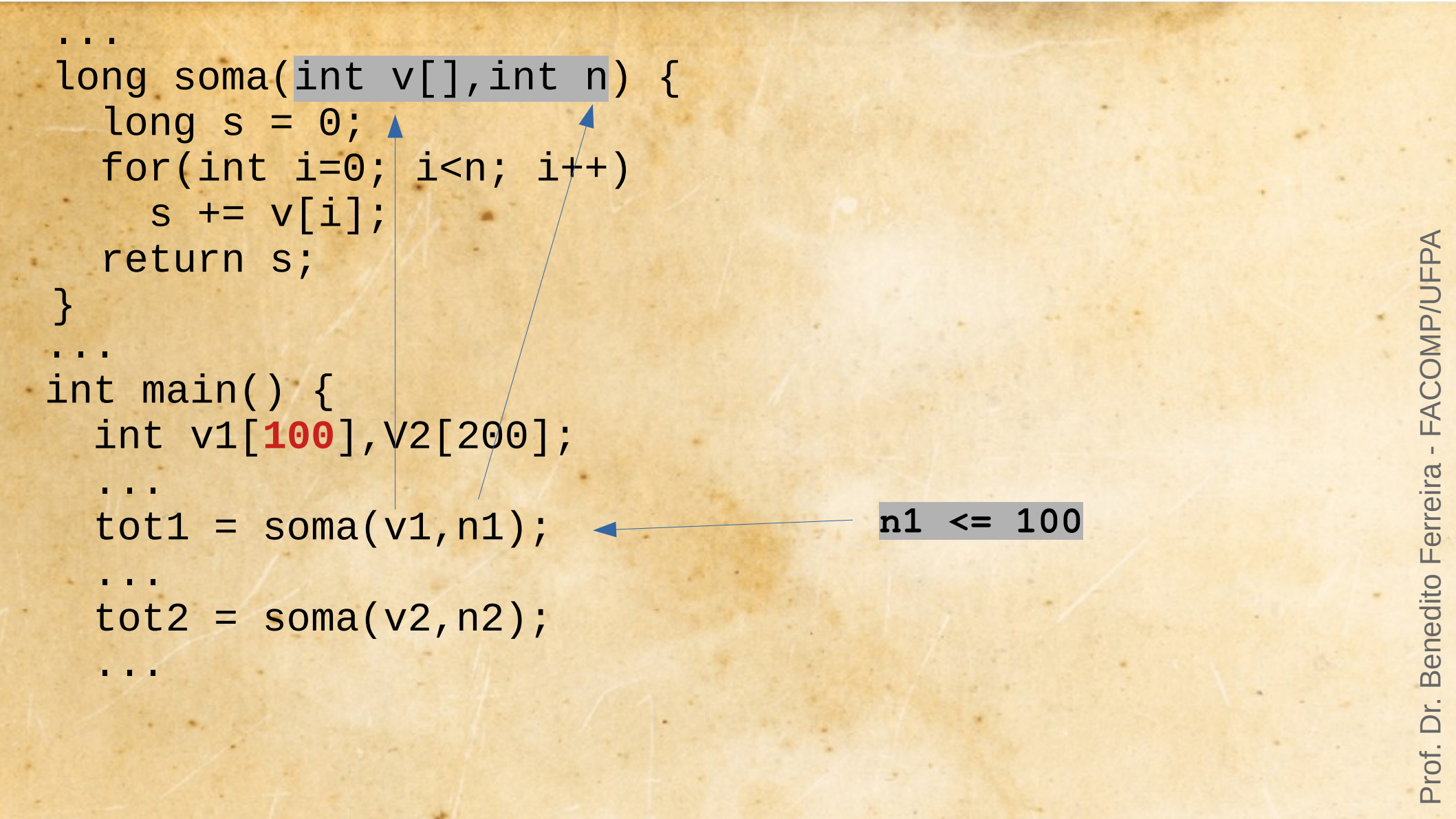


```
...
long soma(int v[]) {
  long s = 0;
  for(int i=0; i<100; i++)
    s += v[i];
  return s;
}
...
int main() {
  int v1[100],V2[100];
  ...
  tot1 = soma(v1);
  ...
  tot2 = soma(v2);
  ...
```

```
...
long soma(int v[],int n) {
   long s = 0;
   for(int i=0; i<n; i++)
     s += v[i];
   return s;
}
...
int main() {
  int v1[100],V2[200];
  ...
  tot1 = soma(v1,n1);
  ...
  tot2 = soma(v2,n2);
  ...
```

novo parâmetro.

```
...
long soma(int v[],int n) {
  long s = 0;
  for(int i=0; i<n; i++)
    s += v[i];
  return s;
}
...
int main() {
  int v1[100],V2[200];
  ...
  tot1 = soma(v1,n1);
  ...
  tot2 = soma(v2,n2);
  ...
```



```
...
long soma(int v[],int n) {
  long s = 0;
  for(int i=0; i<n; i++)
    s += v[i];
  return s;
}
...
int main() {
  int v1[100],V2[200];
  ...
  tot1 = soma(v1,n1);
  ...
  tot2 = soma(v2,n2);
  ...
```

n2 <= 200

```
...
long soma(int v[],int n) {
   long s = 0;
   for(int i=0; i<n; i++)
     s += v[i];
   return s;
}
...
int main() {
  int v1[100],V2[200];
  ...
  tot1 = soma(v1,n1);
  ...
  tot2 = soma(v2,n2);
  ...
```

maior flexibilidade.

```
...
long soma(int v[],int n) {
   long s = 0;
   for(int i=0; i<n; i++)
     s += v[i];
   return s;
}
...
int main() {
  int v1[100],V2[200];
  ...
  tot1 = soma(v1,n1);
  ...
  tot2 = soma(v2,n2);
  ...
```

Parâmetros propiciam:

- comunicação entre os módulos;
- redução de efeitos colaterais.

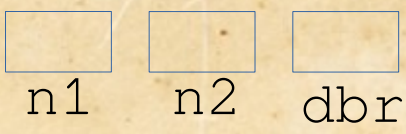
Um **argumento** é um valor (ou outro item) passado para um módulo.

O **parâmetro formal** é um identificador pelo qual um módulo tem acesso a um argumento.

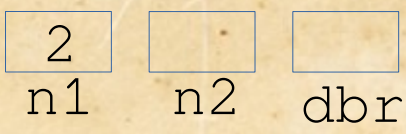
O **parâmetro real** é uma expressão que produzirá o argumento.

Exemplo:

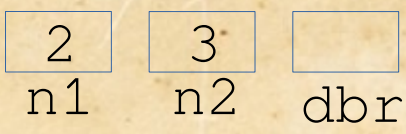
```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```



```
int n1,n2,dbr;
• • •

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

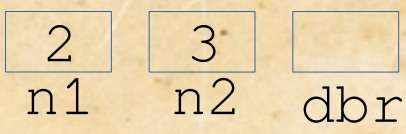


```
int n1,n2,dbr;
• • •

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```



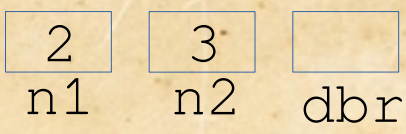
```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

parâmetro formal
(um suporte)

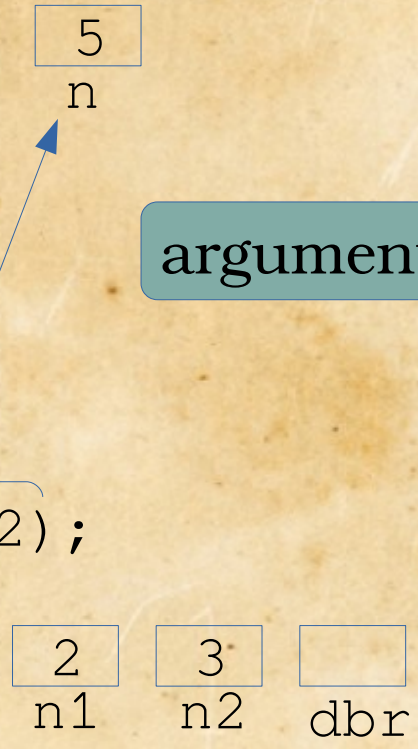


```
int n1,n2,dbr;
• • •

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

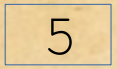
parâmetro real

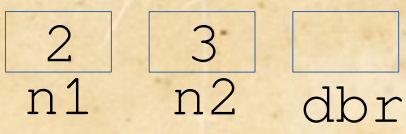


```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

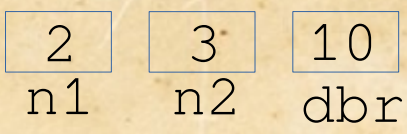


```
int n1,n2,dbr;
• • •

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

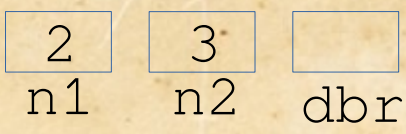




```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```



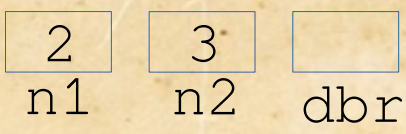
```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```



Outros possíveis parâmetros reais:

```
dbr = dobro(7);
...
dbr = dobro(n1);
```

```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```



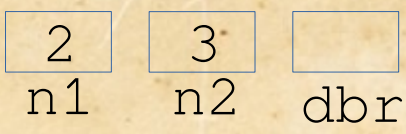
expressões que produzem um valor.

Outros possíveis parâmetros reais:

```
dbr = dobro(7);
...
dbr = dobro(n1);
```

```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

Argumento: 7

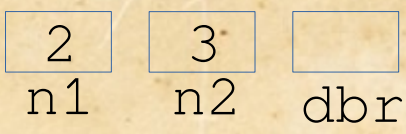


Outros possíveis parâmetros reais:

```
dbr = dobro(7);
...
dbr = dobro(n1);
```

```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

Argumento: 2



Outros possíveis parâmetros reais:

```
dbr = dobro(7);
...
dbr = dobro(n1);
```

```
int n1,n2,dbr;
• • •

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

Deve haver correspondência entre os parâmetros formais e reais quanto a:

- número
- tipo de dados
- posição entre eles

```
int n1,n2,dbr;
. . .

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

Deve haver correspondência entre os parâmetros formais e reais quanto a:

- número
- tipo de dados
- posição entre eles

`n1+n2` (apenas 1 parâmetro)

```
int n1,n2,dbr;
• • •

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

Deve haver correspondência entre os parâmetros formais e reais quanto a:

- número
- tipo de dados
- posição entre eles

`n1+n2` (apenas 1 parâmetro)

`3*n1+5*n2-8` (idem)

```
int n1,n2,dbr;
• • •

int dobro(int n) {
    return 2*n;
}

...
{
    n1 = 2;
    n2 = 3;
    dbr = dobro(n1+n2);
    ...
```

Argumento: 13

Deve haver correspondência entre os parâmetros formais e reais quanto a:

- número
- tipo de dados
- posição entre eles

`n1+n2` (apenas 1 parâmetro)

`3*n1+5*n2-8` (idem)

```
int n1,n2,dbr;
• • •

int d
    re
}

...
{
    n1
    n2
    db
    ..
```

Exceção (argumentos *default*):

C++

```
float volume(float bs,float alt,float prof=2.0);

...

vol = volume(7.1,2.7,4,2);
//  ou...
vol = volume(7.1,2.7);          // prof = 2.0
```

```
int n1,n2,dbr;
. . .

int d
    re
}

...
{
    n1
    n2
    db
    ..
```

Exceção (argumentos *default*):

Python

```
def potencia(base,exp=2):
  return base**exp

...

pot = potencia(2,8);
# ou...
pot = potencia(5);      // exp = 2
```

TIPOS DE PASSAGEM DE PARÂMETROS

Prof. Dr. Benedito Ferreira - FACOMP/UFPA

Por valor: forma mais comum.
→ presente nos exemplos já apresentados.

- o parâmetro formal recebe uma **cópia** do argumento.

- eventual alteração interna no conteúdo do parâmetro formal não terá efeito no parâmetro real.

Por valor: forma mais comum.
→ presente nos exemplos já apresentados.

- o parâmetro formal recebe uma **cópia** do argumento.

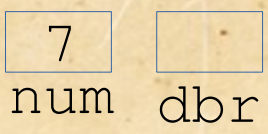
- eventual alteração interna no conteúdo do parâmetro formal não terá efeito no parâmetro real.

C/C++

```
...
int dobro(int x) {
    return 2*x;
}

int main() {
   int num = 7, db;
   ...
   dbr = dobro(num);

   ...
}
```

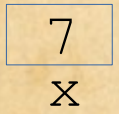
C/C++

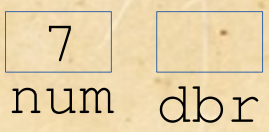
```
...

int dobro(int x) {
    return 2*x;
}


int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```



C/C++

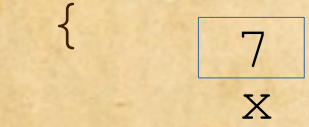
```
...
int dobro(int x) {
    return 2*x;
}

int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```

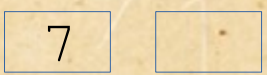




C/C++

```
...
int dobro(int x) {
    return 2*x;
}

int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```





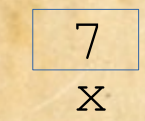
C/C++

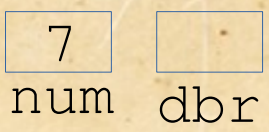
```
...
int dobro(int x) {
    return 2*x;
}

int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```

7	14
num	dbr

C/C++

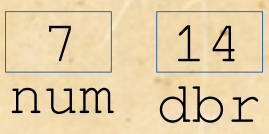
```
...

int dobro(int x) {
    return 2*(x++);   // ??
}


int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```





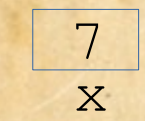
C/C++

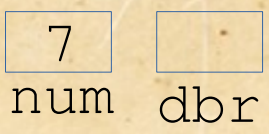
```
...
int dobro(int x) {
    return 2*(x++);   // ??
}

int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```



C/C++

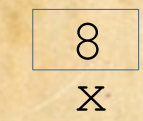
```
...
int dobro(int x) {
    return 2*(++x);   // ????
}

int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```

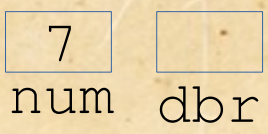




C/C++

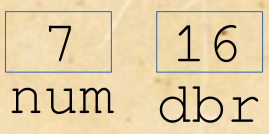
```
...
int dobro(int x) {
    return 2*(++x);   // ????
}

int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```





C/C++

```
...
int dobro(int x) {
    return 2*(++x);   // ????
}

int main() {
    int num = 7, dbr;
    ...
    dbr = dobro(num);

    ...
}
```

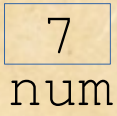


C/C++

```
...
void dobrar(int x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```

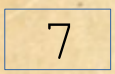
C/C++

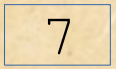
```
...
void dobrar(int x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```



C/C++

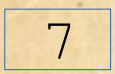
```
...
void dobrar(int x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```





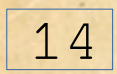
C/C++

```
...
void dobrar(int x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```



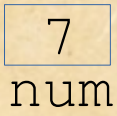
C/C++

```
...
void dobrar(int x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```



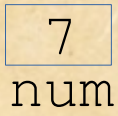
C/C++

```
...
void dobrar(int x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```



C/C++

```
...
void dobrar(int x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```



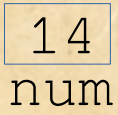
O efeito desejado não foi obtido.

Passagem por referência em C++

```
...
void dobrar(int &x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```

Passagem por referência em C++

```
...
void dobrar(int &x) {
    x = 2*x;
}


int main() {
   int num = 7;
   ...
   dobrar(num);
   ...
}
```

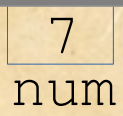


Passagem por referência em C++

```
...
void
    x
}

int r
    in
    .
    do
    .
}
```

Conceito de referência em C++

```
int x =2;
int &y = x;    // outra referência a x

...

x++;  // x vale 3
y++;  // x vale 4
```

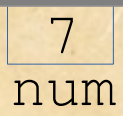


Passagem por referência em C++

```
...
void
    x
}

int r
    in
    .
    do
    .
}
```

Conceito de referência em C++

```
int x =2;
int &y = x;    // outra referência a x

...

x++;   // x vale 3
y++;   // x vale 4
```

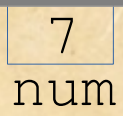


Passagem por referência em C++

```
...
void
    x
}

int r
    in
    .
    do
    .
}
```

Conceito de referência em C++

```
int x =2;
int &y = x;    // outra referência a x

...

x++;   // x vale 3
y++;   // x vale 4
```



Passagem por referência em Pascal

```
var
   num: integer;

procedure dobrar(var x:integer);

begin
   x := 2*x;
end;

begin
   num := 7;
   dobrar(num);
   ...
end.
```

Passagem por referência em Pascal

```
var
   num: integer;

procedure dobrar(x:integer);

begin
   x := 2*x;
end;

begin
   num := 7;
   dobrar(num);  {não haverá o efeito esperado}
   ...
end.
```

default: por valor
(como em C++)

C: somente passagem por valor

C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(int *x) {
    *x = 2*(*x);
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&num);
    cout << "Dobro: " << num << endl;
}
```

C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

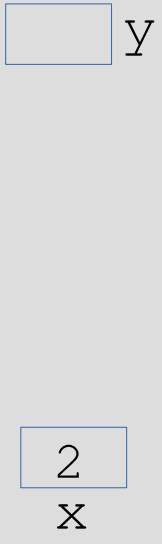
```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y

y = &x;     // endereço de x
```

C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y

y = &x;     // endereço de x
```

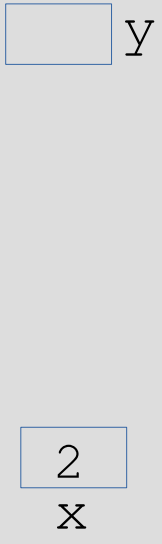


C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y

y = &x;     // endereço de x
```

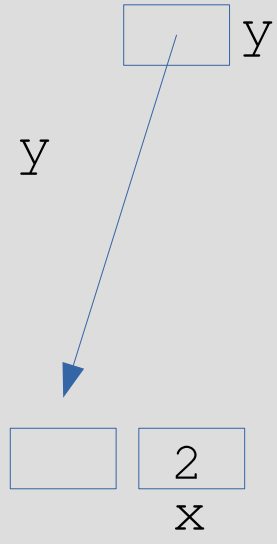


C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y

y = &x;     // endereço de x
```

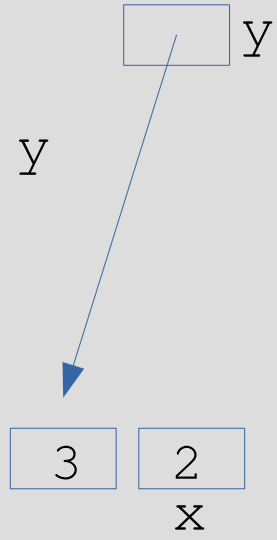


C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y
...
y = &x;     // endereço de x
```

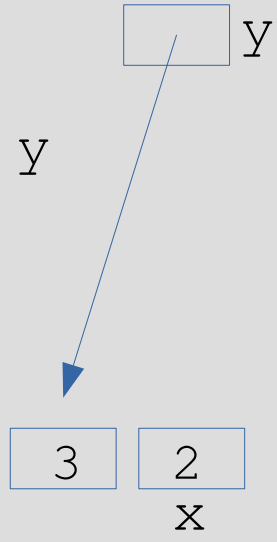


C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y
...
y = &x;      // endereço de x
```

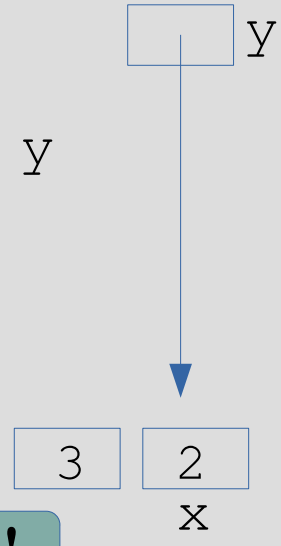


C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y
...
y = &x;     // endereço de x
...
```



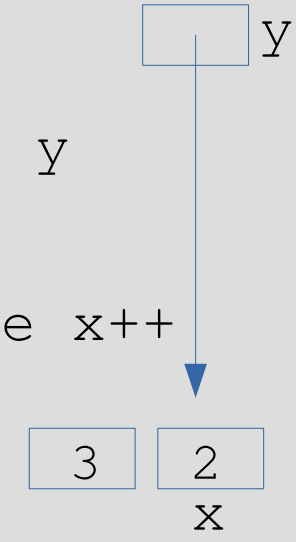
conteúdo perdido!

C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y
...
y = &x;      // endereço de x
(*y)++;      // mesmo efeito de x++
...
```

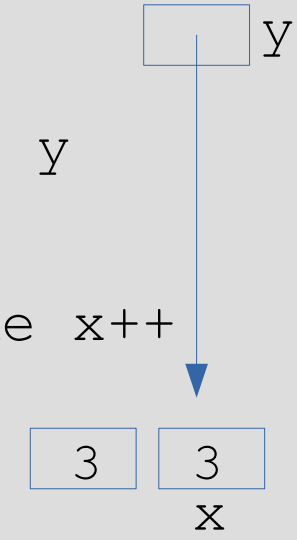


C: somente passagem por valor

→ artifício: uso de ponteiros

```
...
void dobrar(in
    *x = 2*(*x)
}

int main() {
    int num;
    ...
    num = 7;
    dobrar(&nu
    cout << "Do
}
```

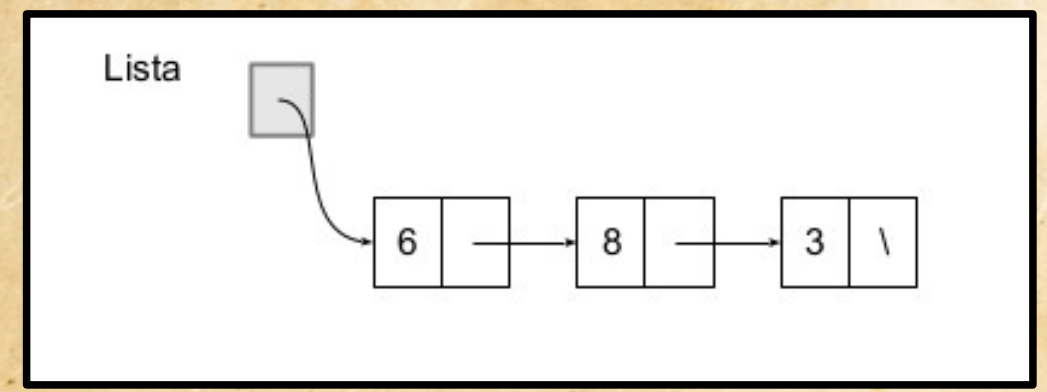
```
int x = 2;   // inteiro
int *y;      // ponteiro para int

...
y = malloc (sizeof(int));
*y = 3;    //obj apontado por y
...
y = &x;     // endereço de x
(*y)++;     // mesmo efeito de x++
...
```



E para passar um **parâmetro ponteiro** por referência??

Ex:

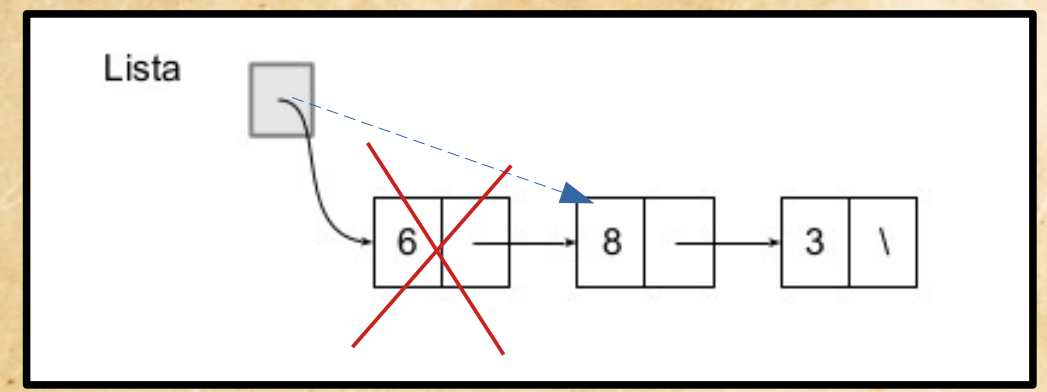
```
void removeInicioLista(No** lista);
```



E para passar um **parâmetro ponteiro** por referência??

Ex:

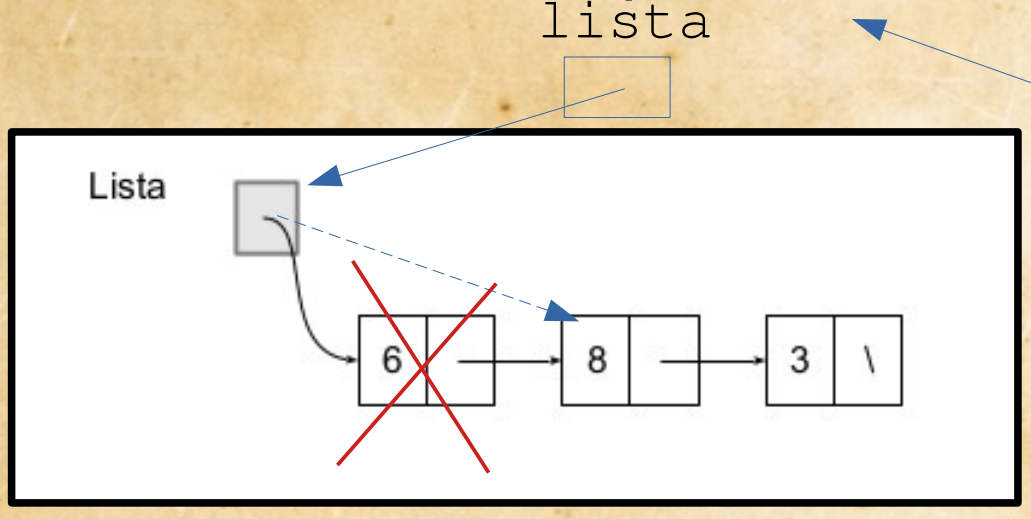
```
void removeInicioLista(No** lista);
```



E para passar um **parâmetro ponteiro** por referência??

Ex:

```
void removeInicioLista(No** lista);
```

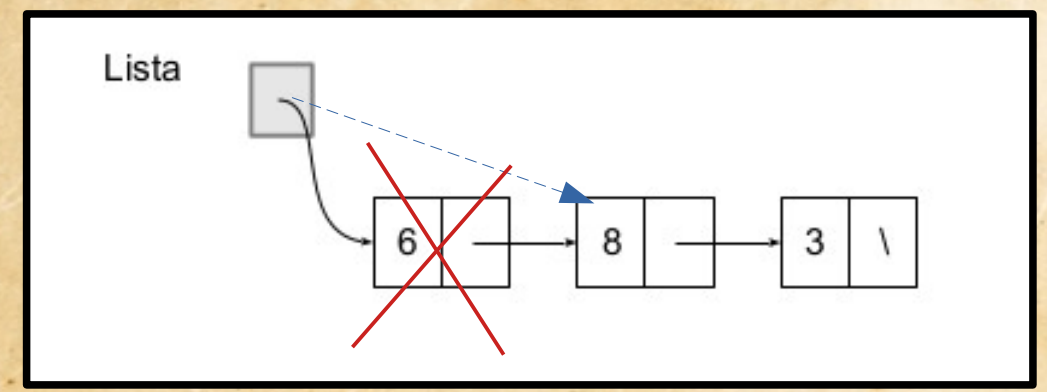


ponteiro para ponteiro!

E para passar um **parâmetro ponteiro** por referência??

Ex:

```
void removeInicioLista(No** lista);
```

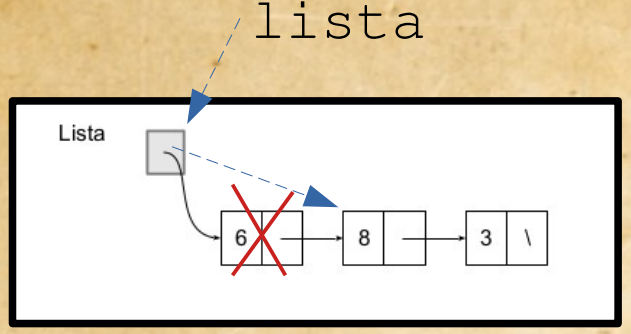


...sintaxe não é das mais amigáveis!

E para passar um **parâmetro ponteiro** por referência??

Ex:

`void removeInici`

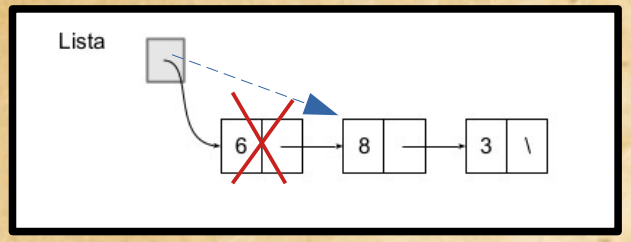



```
void removeIniLst(No ** lista) {

    No *p,*q;
    if (*lista == NULL)
       return;
    p = *lista; q = p->prox;
    free(p);
    *lista = q;
}
```

E para passar um **parâmetro ponteiro** por referência??

Ex:

```
void removeInici
```

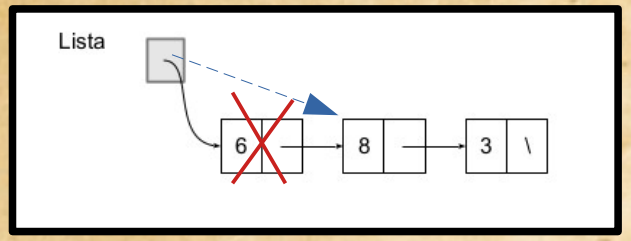



```
No * removeIniLst(No * lista) {

   No *p,*q;
   if (lista == NULL)
      return;
   p = lista; q = p->prox;
   free(p);
   return q;
}
```

E para passar um **parâmetro ponteiro** por referência??

Ex:

```
void removeInici
```




```
No * removeIniLst(No * lista) {

   No *p,*q;
   if (lista == NULL)
      return;
   p = lista; q = p->prox;
   free(p);
   return q;
}
```

Lista = removeIniLst(Lista);

Uma síntese:

C:

- somente passagem por valor;
- uso de ponteiros para alterações nos parâmetros reais;

Uma síntese:

C:
- somente passagem por referência;
- uso de ponteiros para alterações nos parâmetros reais;

C++ e Pascal:
- default: por valor;
- **&** e **var** no caso de passagem por referência.

Uma síntese:

C:
- somente passagem por referência;
- uso de ponteiros para alterações nos parâmetros reais;

C++ e Pascal:
- default: por valor;
- **&** e **var** no caso de passagem por referência.

Em C e C++
arrays – efeito de passagem por referência (são evitadas cópias de grandes objetos)

Observações importantes:

- passagem por valor atende as situações mais comuns em programação;

Observações importantes:

- passagem por valor atende as situações mais comuns em programação;
- passagem por referência traz maiores chances de efeitos colaterais.

Observações importantes:

- passagem por valor atende as situações mais comuns em programação;
- passagem por referência traz maiores chances de efeitos colaterais.

a **alteração** de dados sempre é mais sensível do que sua leitura!

Tendências nas LPs:

- Criação de restrições para o programador, que minimizem a indução ao erro.

Tendências nas LPs:

- Criação de restrições para o programador, que minimizem a indução ao erro.

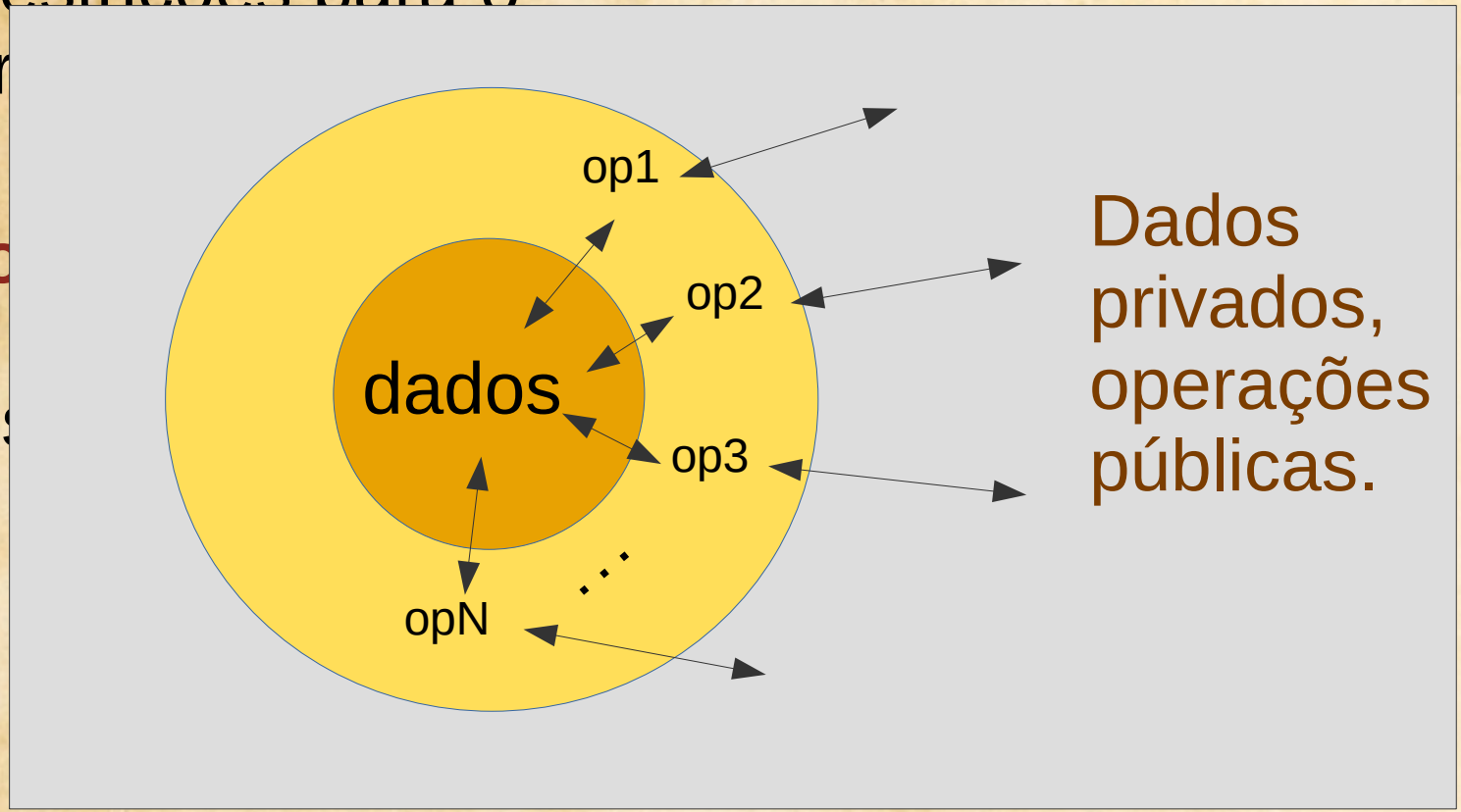
Programação orientada a objetos:

- funções (ou métodos) que manipulam somente os dados internos dos objetos.

Tendências nas LPs:

- Criação de restrições para o programador

Programação
- funções (ou
dados interno



Tendências nas LPs:

- Criação de restrições para o programador, que minimizem a indução ao erro.

Programação orientada a objetos:

- funções (ou métodos) que manipulam somente os dados internos dos objetos.

- eliminação ou minimização (desencorajamento) de práticas problemáticas (uso de variáveis globais, passagem por referência, etc.)

Java:

- Uso muito limitado de variáveis globais

Java:

- Uso muito limitado de variáveis globais

- Tipos primitivos (int, float, etc.): sempre passagem de parâmetro **por valor.**

Java:

- Uso muito limitado de variáveis globais

- Tipos primitivos (int, float, etc.): sempre passagem de parâmetro **por valor.**

- Objetos: é passada a **referência** que dá acesso ao objeto (porém, **métodos** definem disciplinas de acesso).

Java:

- Uso muito limitado de variáveis globais

- Tipos primitivos (int, float, etc.): sempre passagem de parâmetro **por valor.**

- Objetos: é passada a **referência** que dá acesso ao objeto (porém, **métodos** definem disciplinas de acesso).

  → essas **referências** seguem a lógica da passagem por valor: alterações nelas não repercutem externamente.

Python:

- Passagem por valor

- listas/tuplas – passagem por referência

ASPECTOS HISTÓRICOS

Antecedentes: “crise do software”

Anos 1960

- grande desenvolvimento dos equipamentos (memória, processador), sem o correspondente desenvolvimento das técnicas de programação.

Antecedentes: “crise do software”

Anos 1960

- grande desenvolvimento dos equipamentos (memória, processador), sem o correspondente desenvolvimento das técnicas de programação.

- crescimento da dimensão dos sistemas
  → problemas no desenvolvimento monolítico.

Antecedentes: “crise do software”

Anos 1960

- grande desenvolvimento dos equipamentos (memória, processador), sem o correspondente desenvolvimento das técnicas de programação.

- crescimento da dimensão dos sistemas
  → problemas no desenvolvimento monolítico.
  → muito tempo gasto em manutenção.

Antecedentes: “crise do software”

Anos 1960

- grande desenvolvimento dos equipamentos (memória, processador), sem o correspondente desenvolvimento das técnicas de programação.

- crescimento da dimensão dos sistemas
  - → problemas no desenvolvimento monolítico.
  - → muito tempo gasto em manutenção.
  - → dificuldade para localizar erros.

Antecedentes: “crise do software”

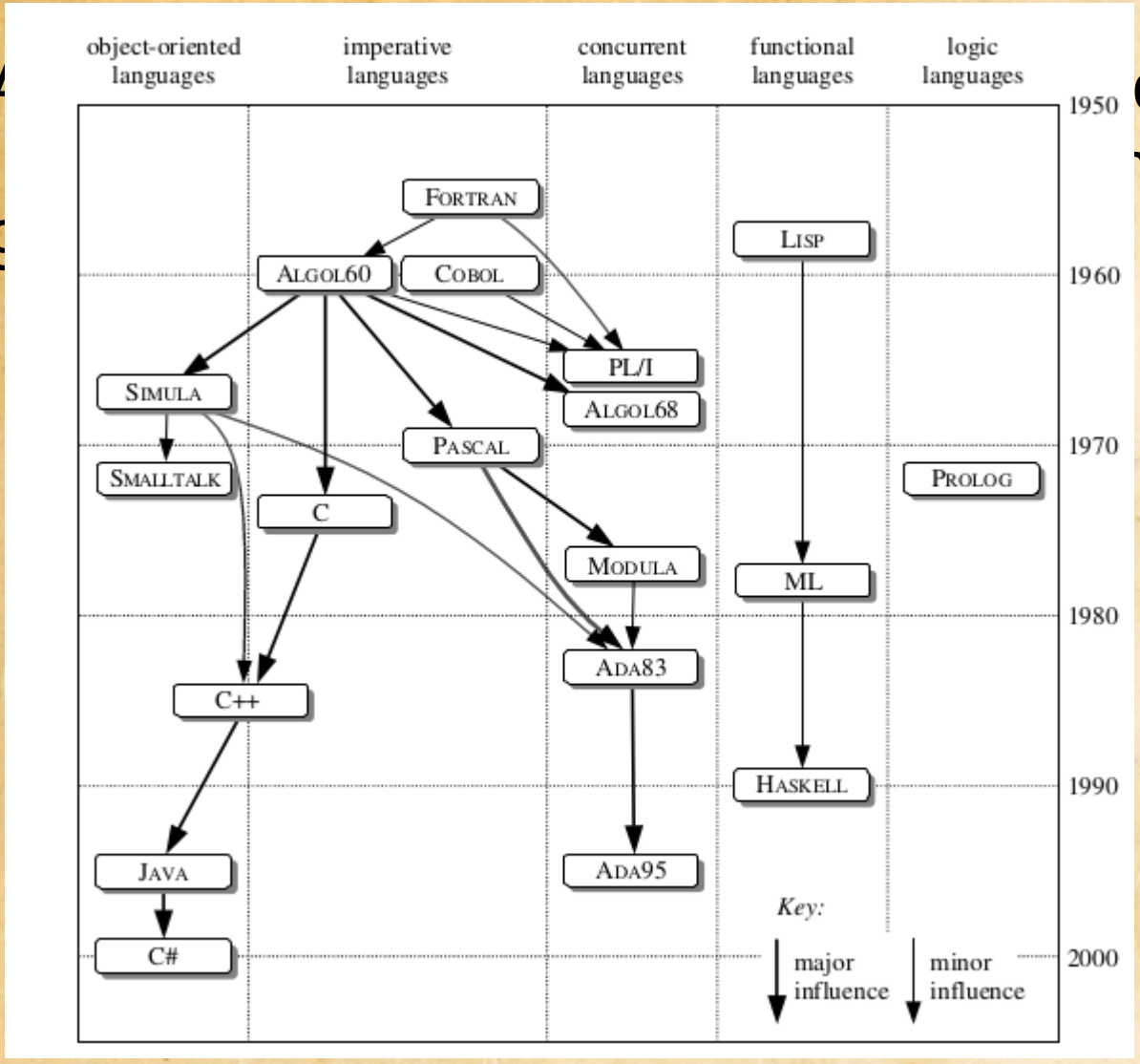
Anos 1960

- grande desenvolvimento dos equipamentos (memória, processador), sem o correspondente desenvolvimento das técnicas de programação.

- crescimento da dimensão dos sistemas
  - → problemas no desenvolvimento monolítico.
  - → muito tempo gasto em manutenção.
  - → dificuldade para localizar erros.
  - → programas com indicação de abandono.

- A técnica da programação modular foi inicialmente concebida no final dos anos 1960.

- A oi
in nos
19



Validade atual da técnica:

A programação modular, *por si:*
→ **foi se tornando também insuficiente**
para o desenvolvimento de grandes sistemas.

Validade atual da técnica:

A programação modular, *por si:*
→ **foi se tornando também insuficiente**
para o desenvolvimento de grandes sistemas.

Porém, não ocorre o abandono dessa técnica, mas sua **incorporação** nos novos paradigmas:
- Tipos abstratos de dados
- Programação orientada a objetos

Validade atual da técnica:

A programação modular, *por si:*
→ **foi se tornando também insuficiente** para o desenvolvimento de grandes sistemas.

Porém, não ocorre o abandono dessa técnica, mas sua **incorporação** nos novos paradigmas:
- Tipos abstratos de dados
- Programação orientada a objetos

Superação/ assimilação

Va

A

→

pa

sis

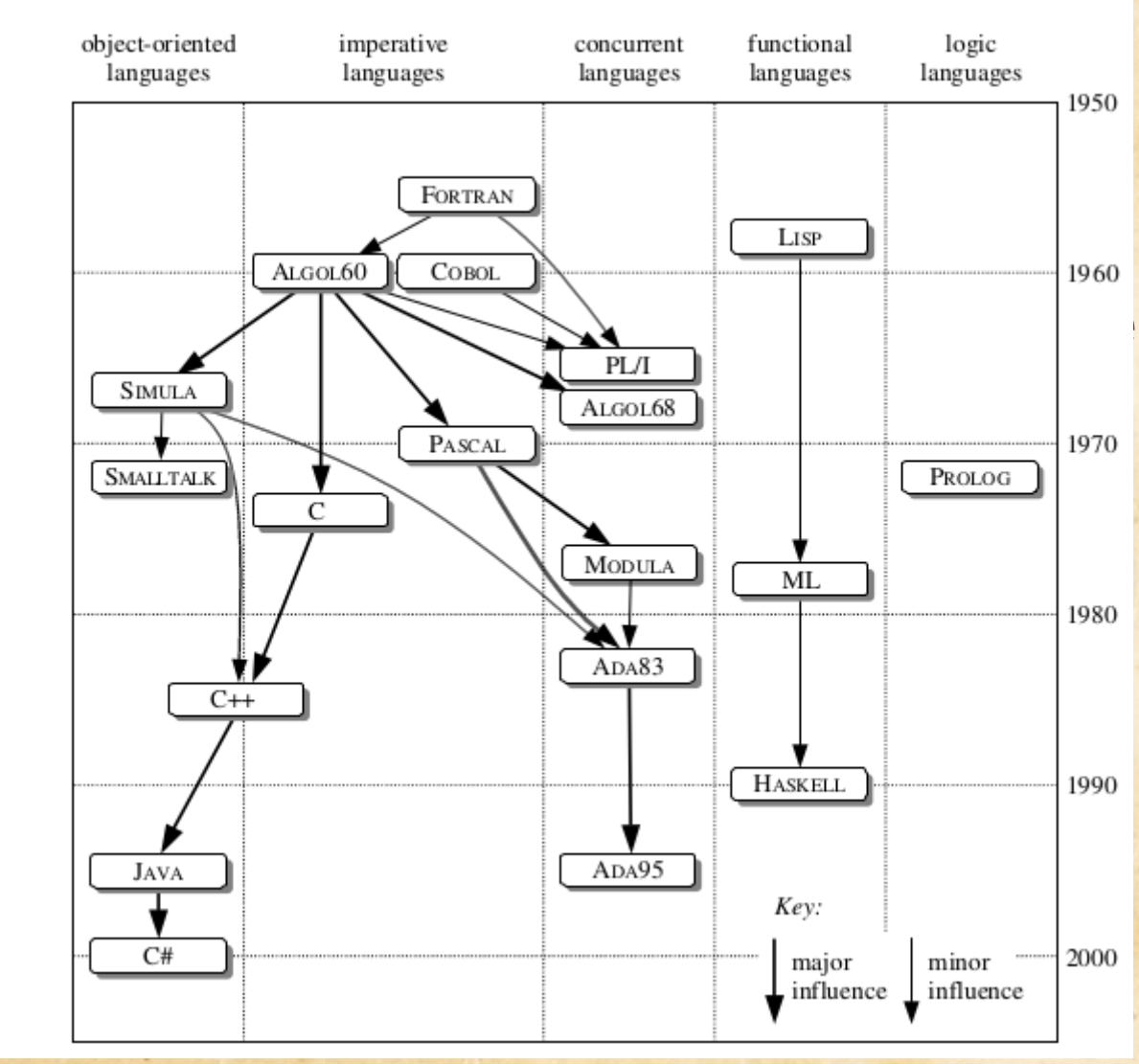
Po

téc

no

- T

- P



nte

Anos 1980/1990

Va

A

→

pa

sis

Po

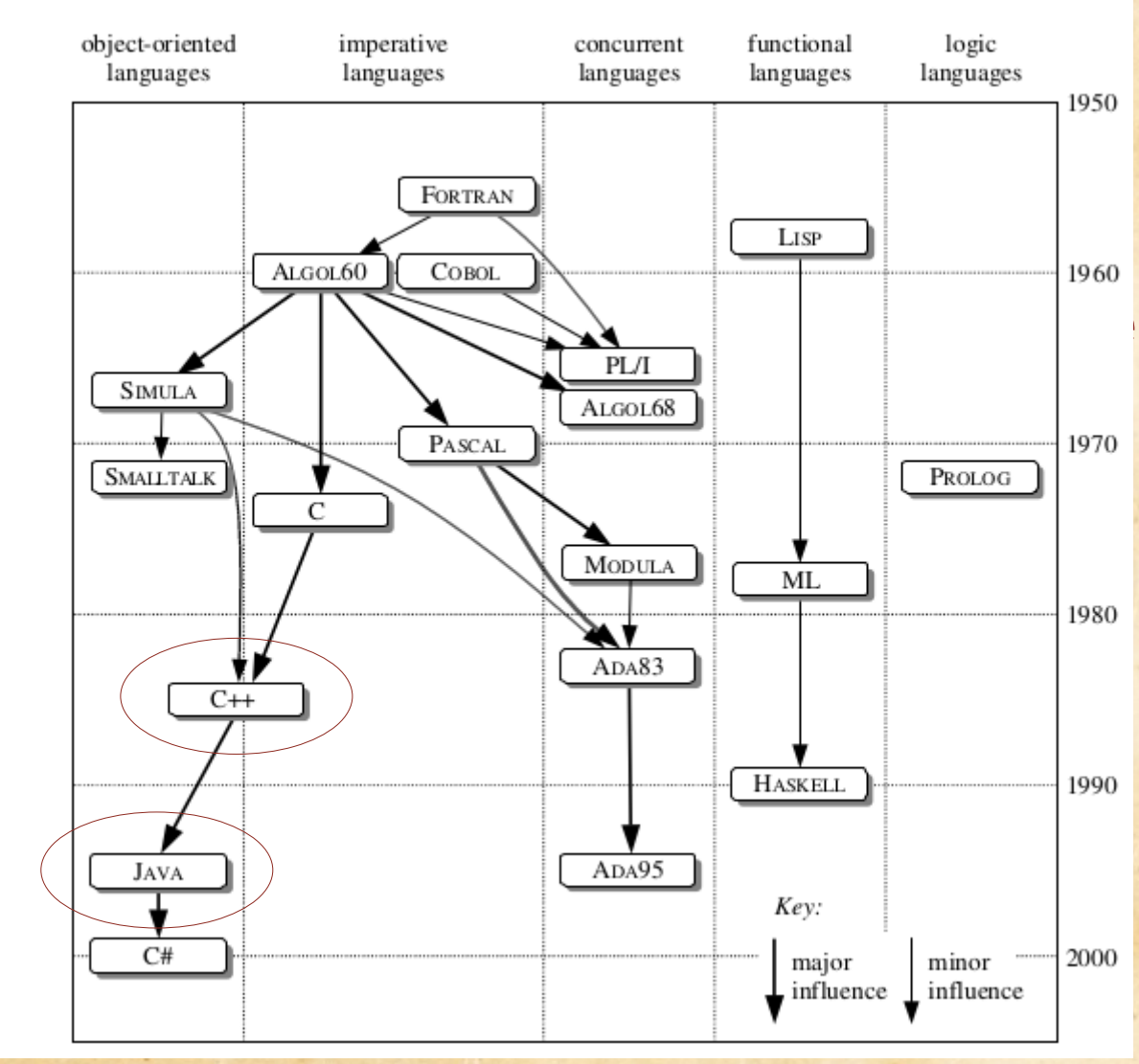
téc

no

- T

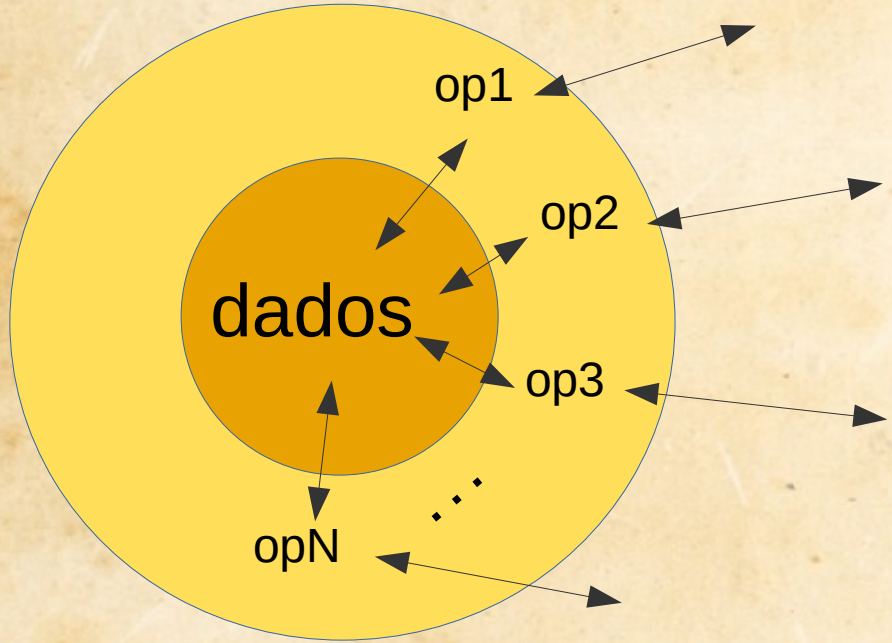
- P

nte



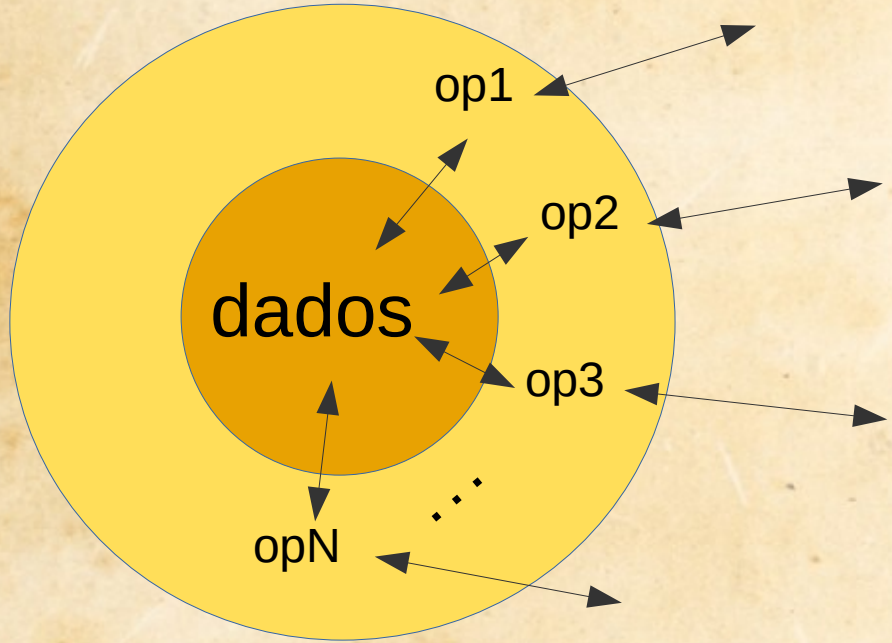
Anos 1980/1990

As operações (métodos) dos objetos não deixam de ser funções (módulos).



As operações (métodos) dos objetos não deixam de ser funções (módulos).

- operação específica
- lista de parâmetros
- tipo de retorno

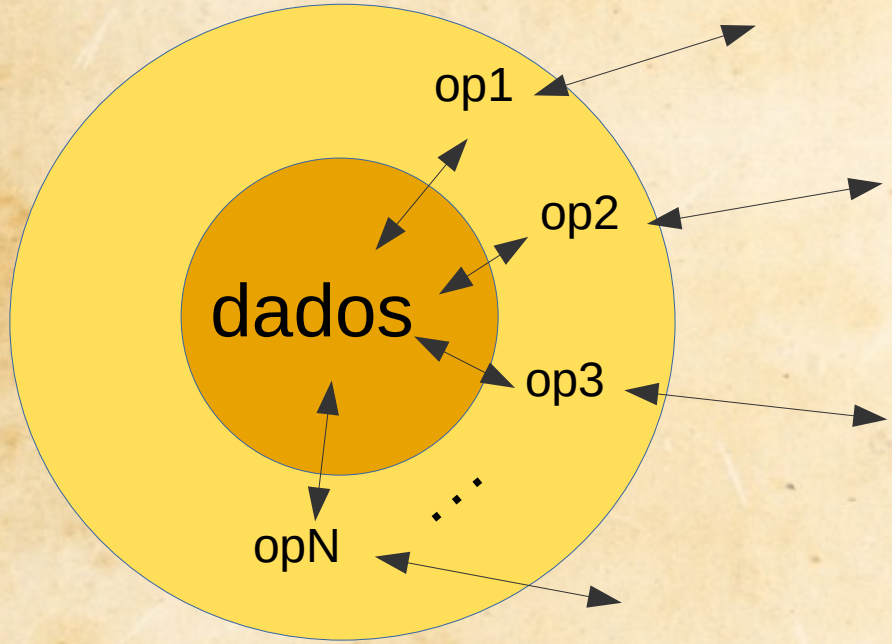


As operações (métodos) dos objetos não deixam de ser funções (módulos).

- operação específica
- lista de parâmetros
- tipo de retorno

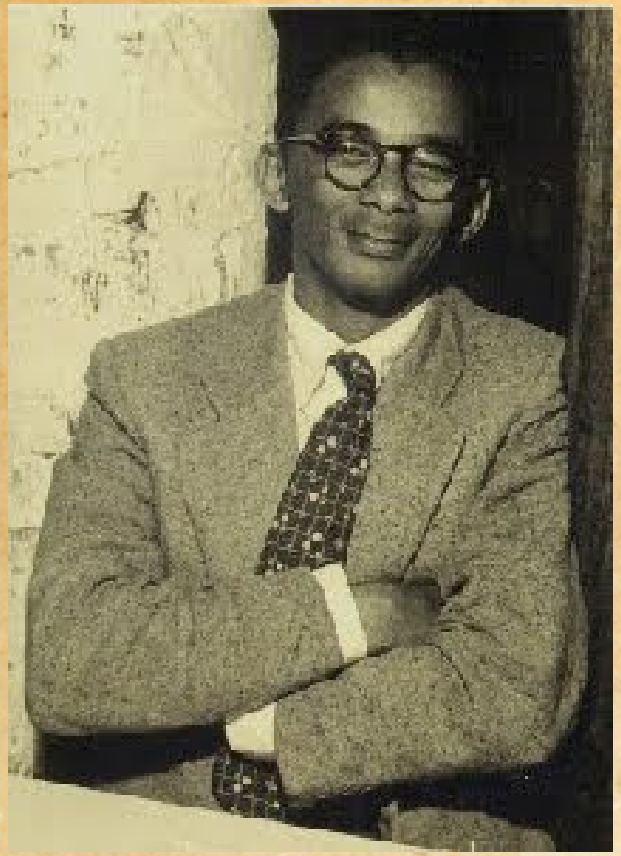
...contudo, dentro de um contexto maior.

- proteção de dados
- controle de sua modificação



DICA CULTURAL

Prof. Dr. Benedito Ferreira - FACOMP/UFPA

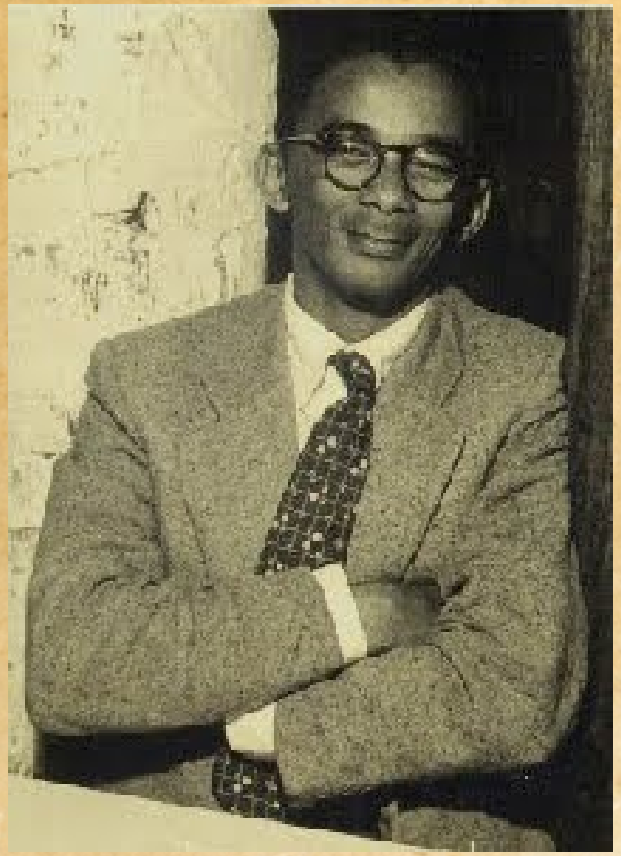


Dalcídio Jurandir

Um dos maiores romancistas da Amazônia e do Brasil.

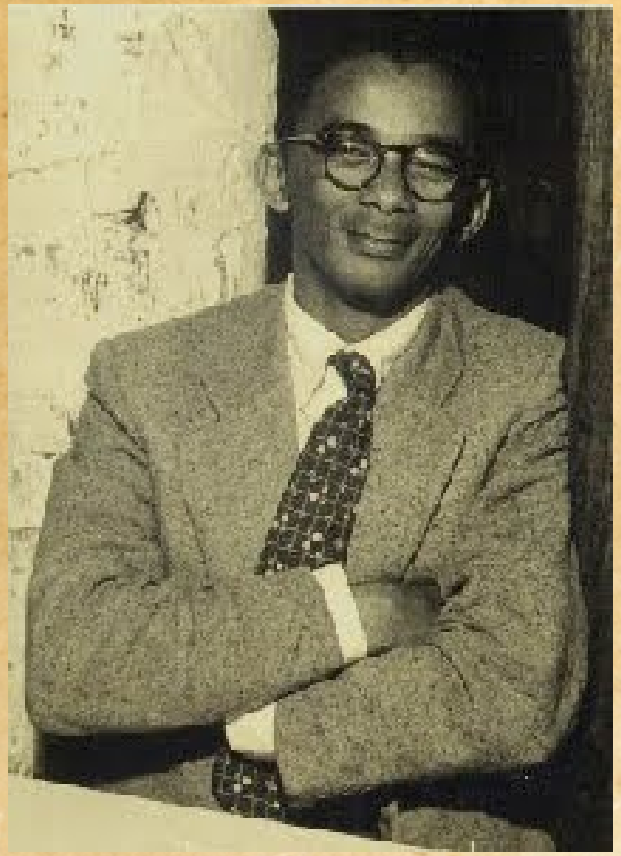
Escritor e jornalista, nascido em Ponta de Pedras (Marajó), em 1909.

Viveu até 1979. Morreu no Rio de Janeiro/RJ.



Dalcídio Jurandir

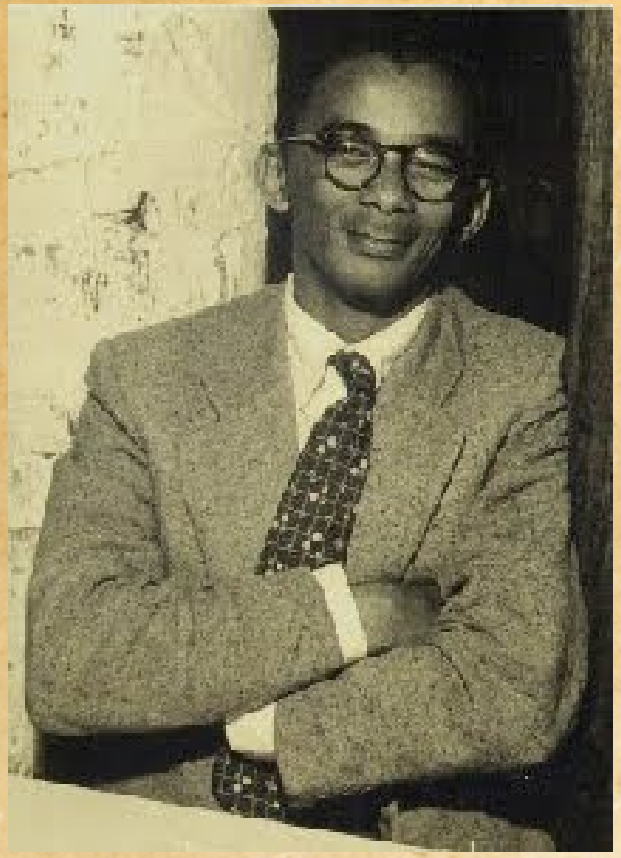
Prêmio Machado de Assis, da ABL, em 1972, pelo conjunto da obra.



Dalcídio Jurandir

Prêmio Machado de Assis, da ABL, em 1972, pelo conjunto da obra.

O **Ciclo do Extremo-Norte**: dez volumes que traçam "um irretocável painel da vida amazônica, envolvendo dramas humanos e questões sociais que permanecem atuais" (Jornal *Beira do Rio*, abril e maio de 2016).



Compõem o ciclo do Extremo-Norte:

Chove nos Campos de Cachoeira (1941),
Marajó (1947),
Três Casas e um Rio (1958),
Belém do Grão Pará (1960),
Passagem dos Inocentes (1967),
Primeira Manhã (1967),
Ponte do Galo (1971),
Os Habitantes (1976),
Chão dos Lobos (1976),
Ribanceira (1978).

"Todo meu romance distribuído, provavelmente, em dez volumes, é feito, da maior parte, da gente mais comum, tão ninguém, que é a minha criaturada grande de Marajó, Ilhas e Baixo Amazonas. Fui menino de beira de rio, do meio do campo, banhista de igarapé. Passei a juventude no subúrbio de Belém, entre amigos, nunca intelectuais, nos salões da melhor linhagem que são os clubinhos de gente da estiva e das oficinas, das doces e brabinhas namoradas que trabalhavam na fábrica. Um bom intelectual de cátedra alta diria: são as minhas essências, as minhas virtualidades. Eu digo tão simplesmente: é a farinha d'agua dos meus bijus (sic). Sou um também daqueles de lá, sempre fiz questão de não arredar pé de minha origem e para isso, ou melhor, para enterrar o pé mais fundo, pude encontrar uma filiação ideológica que me dá razão. A esse pessoal miúdo que tento representar nos meus romances chamo de aristocracia de pé no chão".

Dalcídio Jurandir

Folha do Norte - 23 de outubro de 1960.

Rio de Janeiro, Jornal Dom Casmurro, 3 de agosto de 1940.

FOI ASSIM...

GRANDE CONCURSO «DOM CASMURRO» E «VECCHI EDITOR»

Premiados os srs. Dalcidio Jurandir e Clovis Ramalhete!

O "Premio DOM CASMURRO", de 5:000$000, coube ao romance "Chove nos campos de Cachoeira", de autoria do escritor paraense, enquanto que a "Ciranda", do sr. Clovis Ramalhete, foi conferido o "Premio Vecchi - Editor", de 3:000$000 — Notas sobre os dois premiados — A Ata do Concurso — Como transcorreu a etapa final do disputado certame que interessou vivamente as rodas literárias

A ATA

"AOS VINTE e quatro dias do mês de julho de mil novecentos e quarenta, reuniu-se na redação de DOM CASMURRO, á rua Evaristo da Veiga, 16, 1.º andar, ás 21 horas, a comissão julgadora do Concurso de Romances, instituido por este jornal em combinação com Vecchi-Editor desta capital. O motivo da reunião prendia-se ao julgamento final dos originais inscritos, que se apresentaram em numero de quatro (4), como finalistas ás decisões do juri, e que se intitulavam: — "Chove nos campos de Cachoeira", "Ciranda", "Estrêla do Pastor" e "Marinatambalo".

"Os trabalhos tiveram inicio precisamente ás 21 horas, com a presença dos Srs. Bricio de Abreu, presidente do juri; Alvaro Moreyra, por DOM CASMURRO; Eugênia Alvaro Moreyra, por delegação de Oswald de Andrade; Omer Mont'Alegre, como intelectual e representante de Vecchi-Editor; assistidos por Danilo Bastos, secretário do concurso.

"Tomando a palavra, o sr. Bricio de Abreu expõe que a Sra. Raquel de Queiroz achava-se impossibilitada de comparecer á reunião, por se encontrar enférma, mas que havia comunicado que todos os romances estavam com "notas" consignadas por ela, após os haver lido, sendo de sua opinião que, assim, a sua ausência não prejudicaria a marcha dos trabalhos. Comunicou ainda o sr. Bricio de Abreu que o sr. Wilson de A. Lousada ficara de comparecer á reunião, não o tendo feito até áquela hora, mas que também todos os romances estavam com "notas" por ele dadas. Em seguida pede, afim de esclarecer os debates, proceda o sr. secretário á leitura das notas conseguidas pelos quatro finalistas e se faça a adição. Aprovada a proposta e feita a soma, verificou-se o seguinte resultado:

Marinatambalo . . .	32 pontos
Chove nos campos de Cachoeira	35 "
Ciranda	35 "
Estrêla do Pastor . .	32 "

"Diante dêsse resultado, propõe o sr. Alvaro Moreyra que, em vista do sr. Omer Mont'Alegre ser o membro do Juri de maior responsabilidade no caso, não só em virtude da sua condição de intelectual como também de representante do editor que patrocinava o Prêmio e ainda editaria os livros, expusesse êle aos colegas, em detalhes, a justificação dos seus votos.

"Expõe então claramente o sr. Omer Mont'Alegre as razões dos seus votos, propondo após que só se tivesse em conta para os prêmios os romances "Chove nos campos de Cachoeira" e "Ciranda", uma vez que eram os de maior numero de pontos.

"Em seguida, o sr. Bricio de Abreu propõe que cada um dos presentes fizesse um estudo detalhado e a análise da leitura, sob todos os pontos de vista dos méritos de cada um dos dois romances. Aprovada a proposta, falou primeiro o sr. Omer Mont'Alegre, em seguida Alvaro Moreyra e depois Eugênia Alvaro Moreyra, ficando unanimemente comprovada a superioridade de "Chove nos campos de Cachoeira".

"Prosseguindo, expõe o sr. Bricio de Abreu que tôdas as notas, como vinha de verificar, tinham sido iguais para os dois concorrentes, com exceção de Eugênia Alvaro Moreyra que havia dado 7 a "Chove nos campos de Cachoeira" enquanto a sra. Raquel de Queiroz havia dado 3, ao passo que para "Ciranda" a primeira havia dado 6 e a ultima também 6, obtendo ambos, assim o mesmo numero de votos. Continua com a palavra o sr. Bricio de Abreu que, diante da exposição que cada um dos presentes vinha de fazer dos méritos dos dois romances e de acôrdo com o regulamento do Juri, como Presidente da Comissão, dá o primeiro prêmio ao livro intitulado "Chove nos campos de Cachoeira", de Jagarajó, e o segundo prêmio ao romance "Ciranda", de Matias Pascoal.

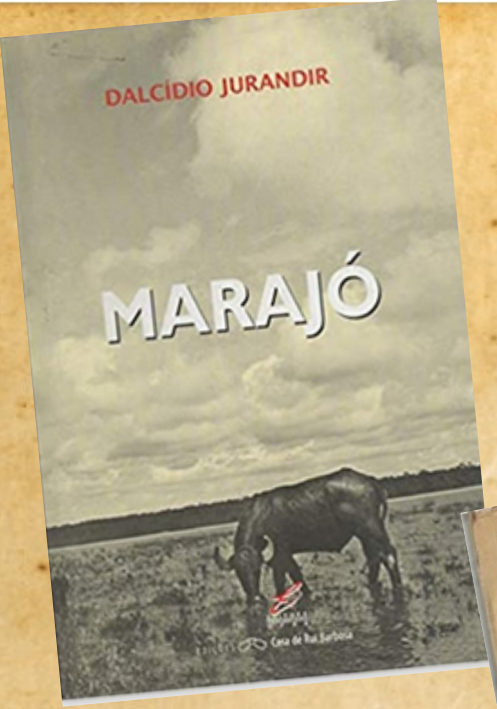
"Pedidos os envelopes que acompanharam os originais, foram êles, na presença de todos, abertos pelo sr. secretário, verificando-se então que o pseudônimo Jagarajó escondia o sr. Dalcidio Jurandir, e o de Matias Pascoal o sr. Clovis Ramalhete, o primeiro de Belém, no Estado do Pará, e o ultimo desta capital".

Dalcidio Jurandir

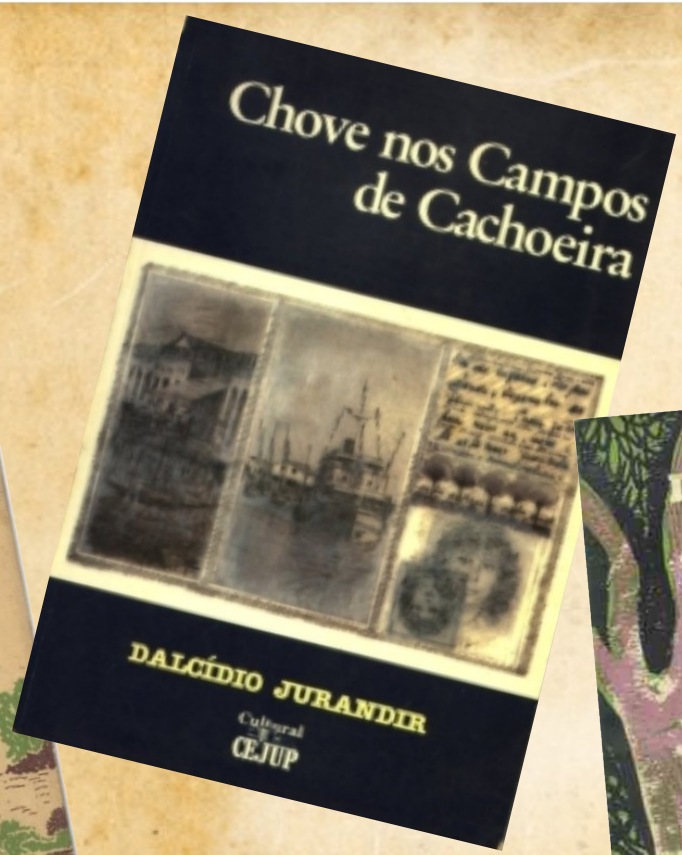
... claros de combate aos temas seculares, ás teorias rotineiras e ao ... ro de Castro, Sandoval Lage, Abguar Bastos e outros.

... o ultimo grau da competição.

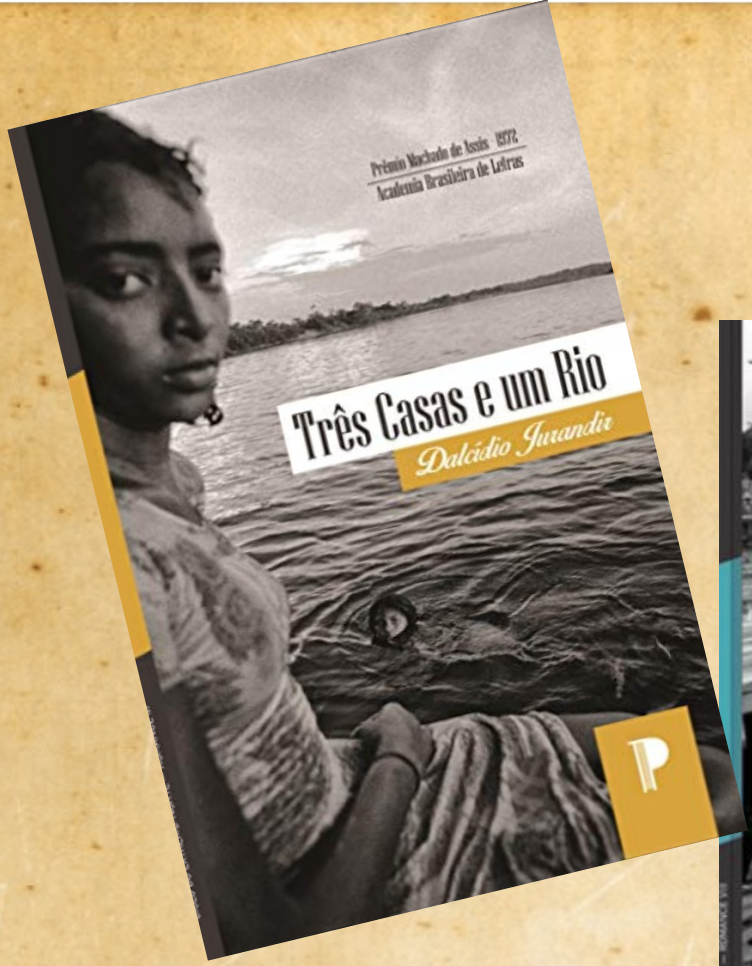
Dalcidio Jurandir foi oficial do ... Suas viagens constantes pelos rios paraenses, em contacto quase ... des fazendas que deram á Ilha característica eminentemente ... tipos e os métodos de vida, para, em seguida, traçar a sua galeria,

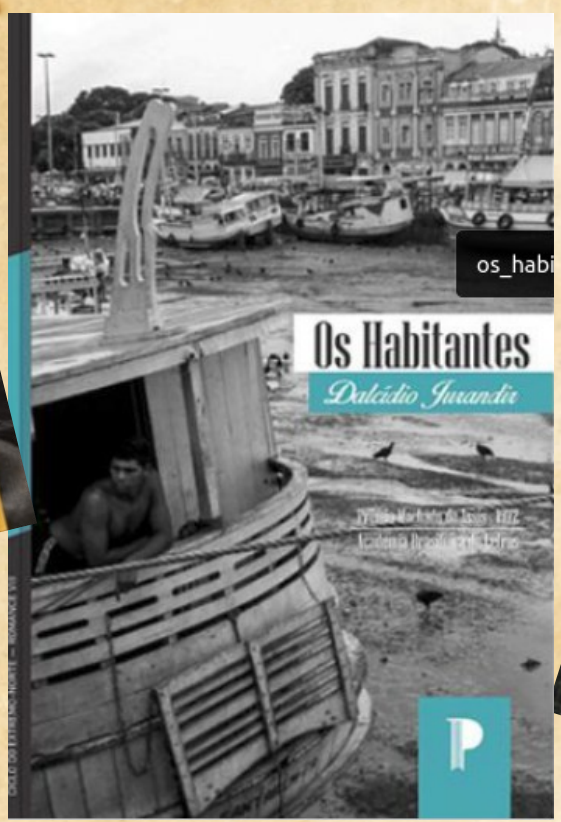

DALCÍDIO JURANDIR
MARAJÓ


DALCIDIO JURANDIR
MARAJÓ
ROMANCE
LIVRARIA JOSÉ OLYMPIO EDITORA


Chove nos Campos de Cachoeira
DALCÍDIO JURANDIR
Cultural
CEJUP


Dalcidio Jurandir
BELÉM DO GRÃO PARÁ
romance
MARTINS


Prêmio Machado de Assis 1977
Academia Brasileira de Letras
Três Casas e um Rio
Dalcídio Jurandir


os_habi
Os Habitantes
Dalcídio Jurandir


DALCÍDIO JURANDIR
PONTE DO GALO
MARTINS — MEC