# 実行時環境と手続き呼び出し

# 概要

- 実行時環境
- 手続き呼び出し
  - 活性レコード
  - 手続き呼び出しの実現
  - 変数の参照

# 大域変数と関数定義

## 大域変数と関数機能

(演習問題5より)

```
<PROGRAM>::= <DECLS> <MAIN>
<DECLS>::= empty
        |  <DECLS> <INTDECL>
        |  <DECLS> <FUNCDECL>
<INTDECL>::= 'int' <IDENTLIST> ';'
<FUNCDECL>::= 'int' <IDENT> '(' <FORMALLIST> ')' <BLOCK>
<FORMALLIST>::= empty
             |  <IDENTLIST>
<IDENTLIST>::= <IDENT>
             |  <IDENT> '[' <NUMBER> ']'
             |  <IDENTLIST> ',' <IDENT>
             |  <IDENTLIST> ',' <IDENT> '[' <NUMBER> ']'
<MAIN>::= 'int' 'main'  '(' ')' <BLOCK>
```

**<FORMALLIST>　formal parameter(仮引数）のリスト**

# 関数呼び出し・リターン

```
<STATEMENT>::=
              ….
        |  'return' <EXPRESSION> ';'

<FACTOR>::= <IDENT>
        |  <IDENT> '[' <EXPRESSION> ']'
        |  <NUMBER>
        |  <IDENT> '(' <ACTUALLIST> ')'
        |  '(' <EXPRESSION> ')'
<ACTUALLIST>::= empty
            |  <EXPRESSIONLIST>
<EXPRESSIONLIST>::= <EXPRESSION>
                |  <EXPRESSIONLIST> ',' <EXPRESSION>
```

**<ACTUALLIST>　実引数のリスト**

# 関数呼び出しとリターン(hsm)

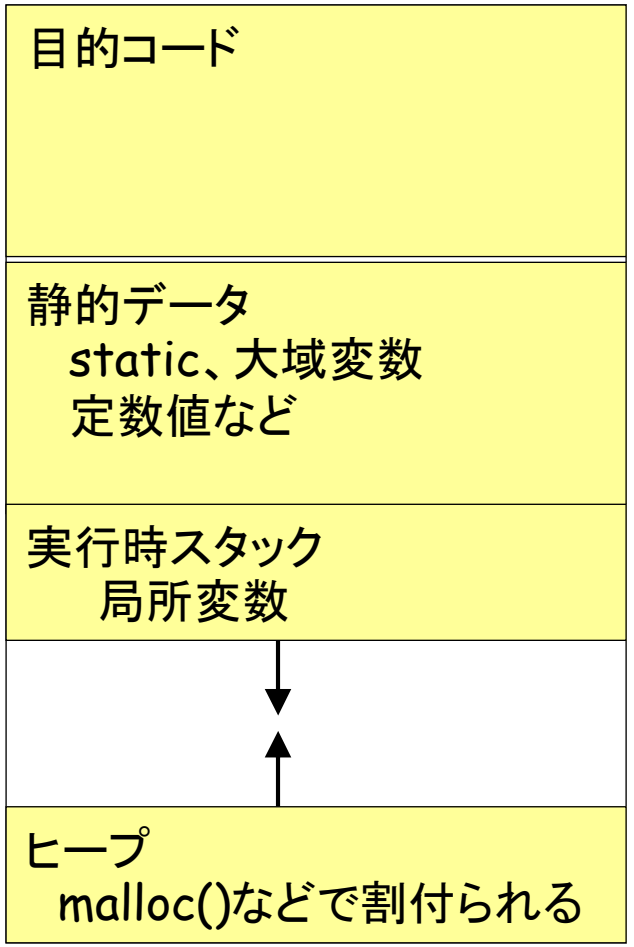
CALL p, q　　関数呼び出し
　　S[sp+1]=b; S[sp+2]=pc; S[sp+3]=base(p); b=sp+1; pc=q;

EF 0, q　　関数の終了
　　pc=S[b+1]+1;b=S[b]; S[b-q]=S[t]; sp=b-q;

b: ベースポインタ(フレームポインタ)
base(p): pレベル下のベースポインタを返す。
　e.g. base(1) 1レベル下のベースポインタを取り出す。
　　base(0) 自分のベースポインタを返す

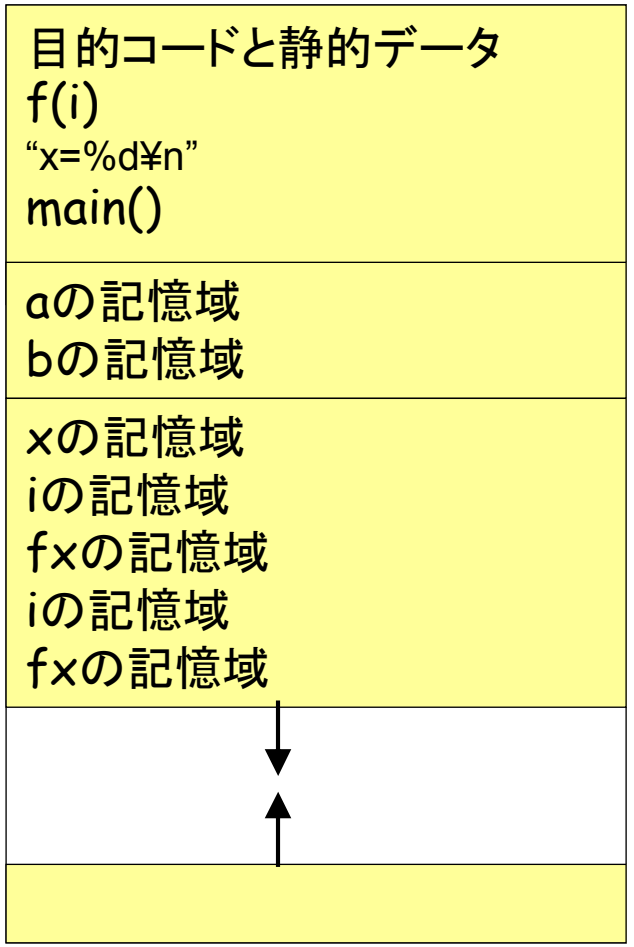
# 実行時環境

- 実行時記憶域のうちわけ
  - 目的コード
  - 静的データ
  - 実行時スタック
  - ヒープ



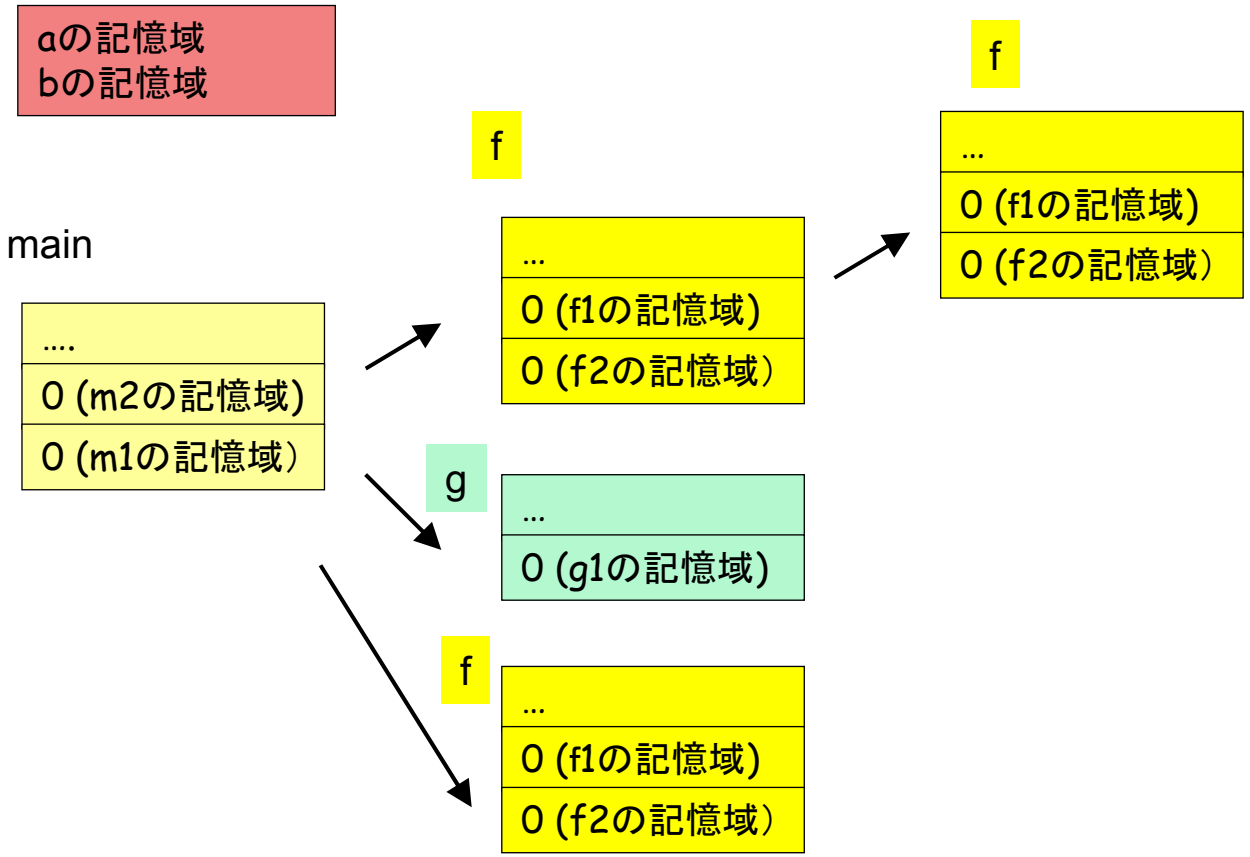
# C言語の例

```
#include <stdio.h>
int a;
int b;
int f(int i)
{
  int fx;
  if (i > 0)
    fx = i* f(i-1);
  else
    fx = 1;
  return fx;
}

int main()
{
  int x;
  a = 0;
  x = f(10);
  printf("x=%d¥n", x);
}
```



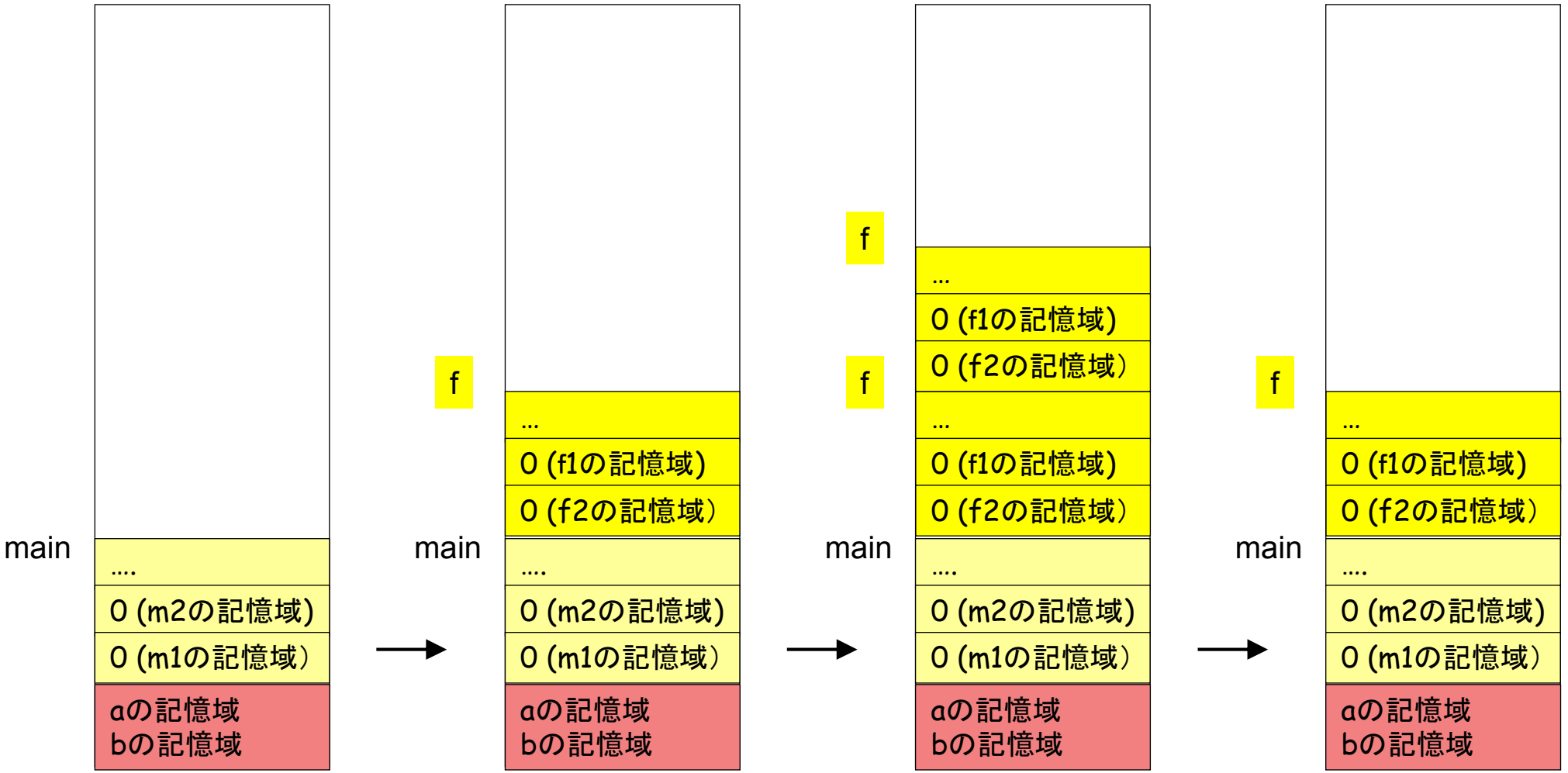
# 手続き呼び出し(概要)

- 手続き呼び出し(procedure call)
  関数呼び出し(function call)、
  メソッド呼び出し(method call)
- サブルーチンの呼び出し(副プログラム)

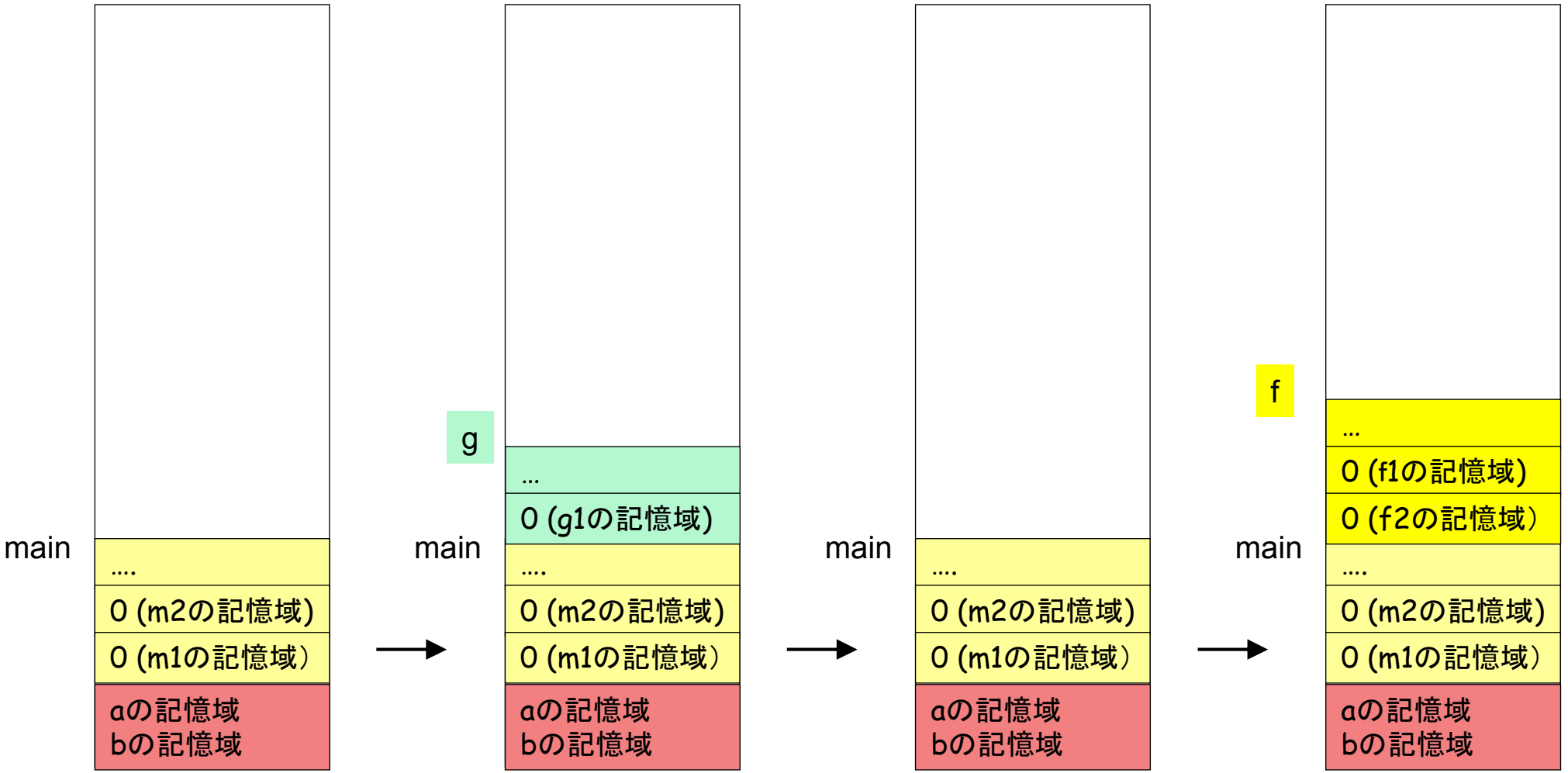



```
int a;
int b;
int f(int i){
  int f1, f2;

  .. f(i-1);
}

int g(){
  int g1;
  ...
}
int main(){
  int m1;
  int m2;
  a = f(2);
    ..
  b = g(2);
   ..
  m2 = f(3) ;
}
```

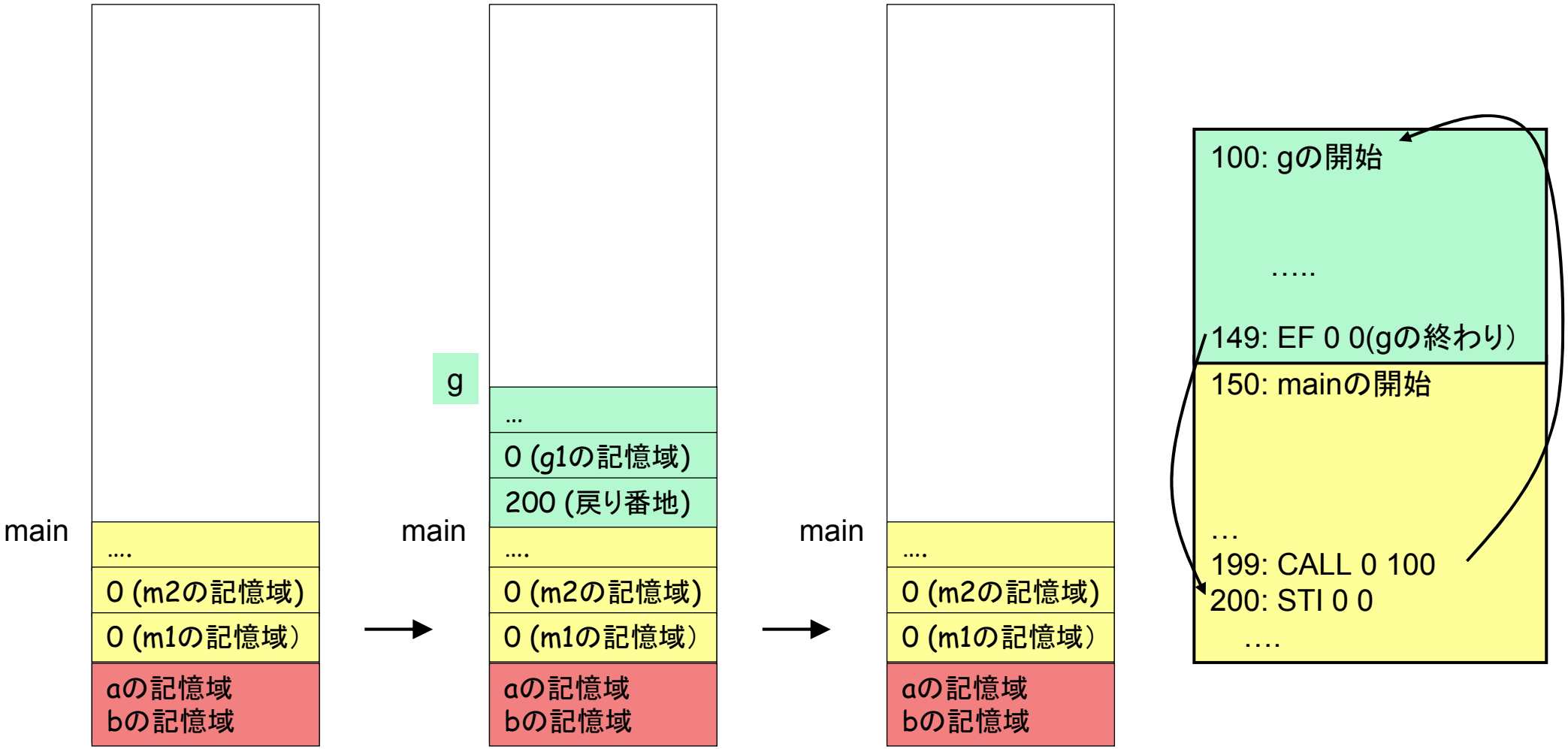
# 実行時スタックの例（1）



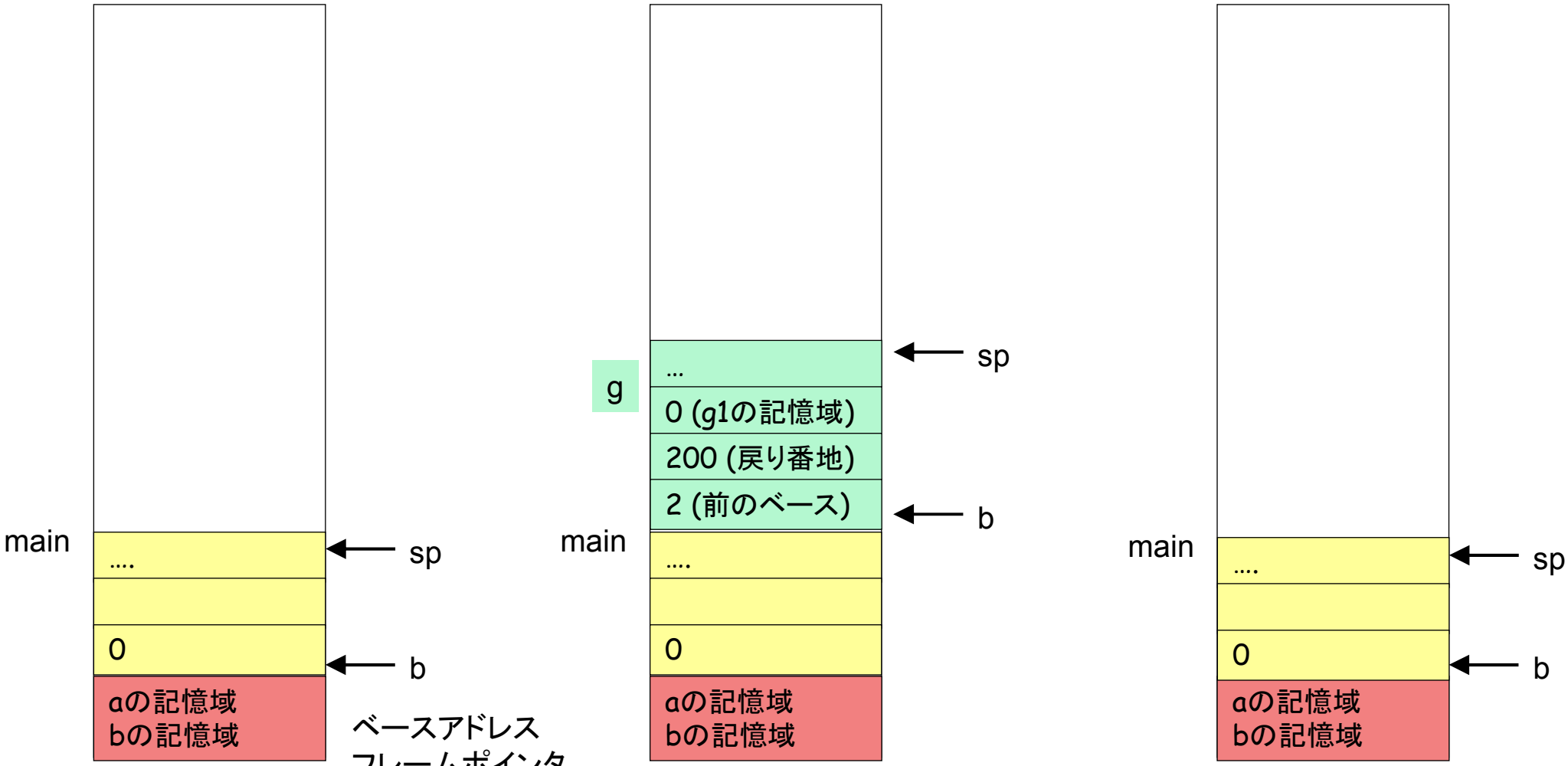
# 実行時スタックの例（2）


main
....
0 (m2の記憶域)
0 (m1の記憶域)
aの記憶域
bの記憶域
g
main
...
0 (g1の記憶域)
....
0 (m2の記憶域)
0 (m1の記憶域)
aの記憶域
bの記憶域
main
....
0 (m2の記憶域)
0 (m1の記憶域)
aの記憶域
bの記憶域
f
main
...
0 (f1の記憶域)
0 (f2の記憶域)
....
0 (m2の記憶域)
0 (m1の記憶域)
aの記憶域
bの記憶域

# 制御の移行と復帰

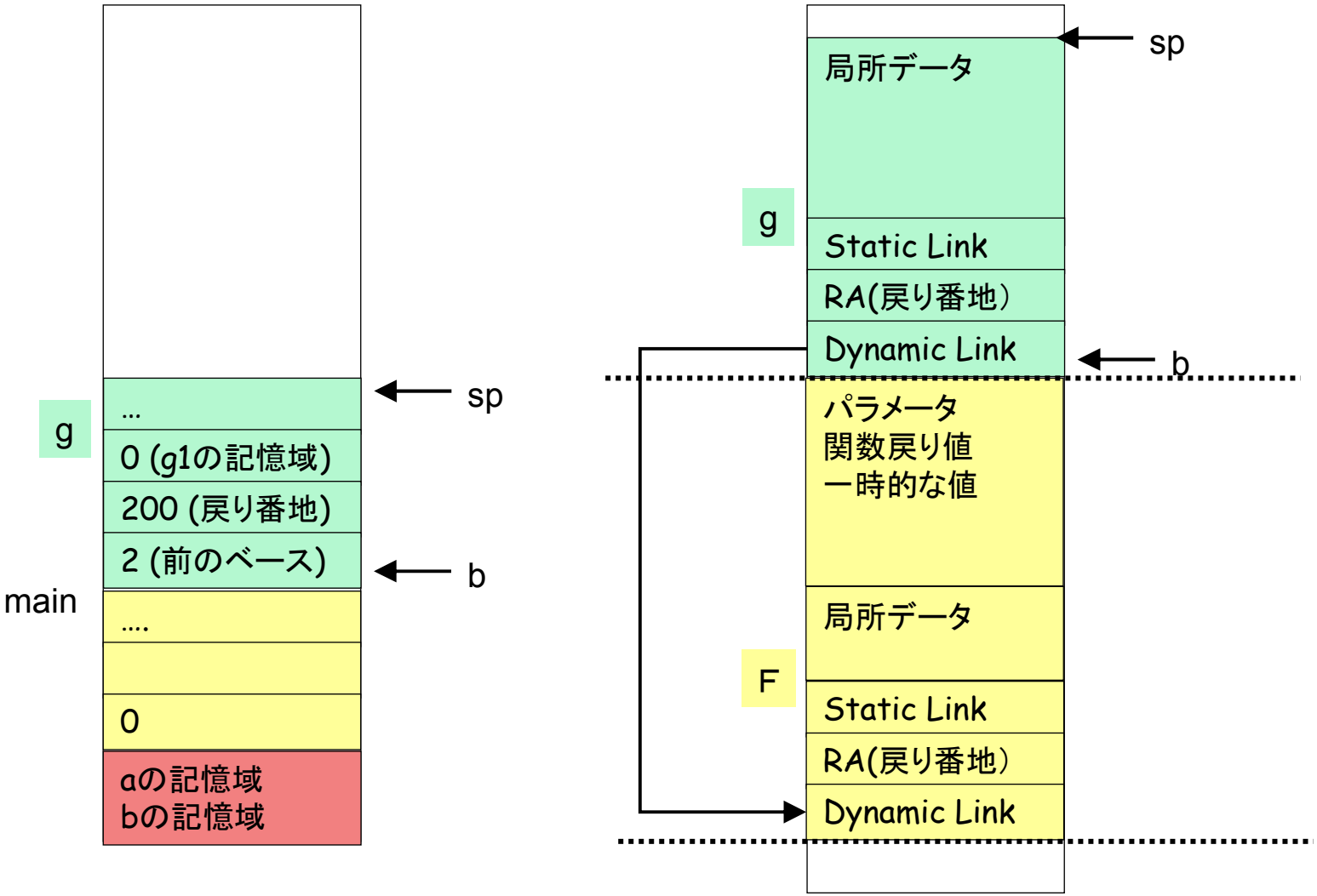

main
....
0 (m2の記憶域)
0 (m1の記憶域)
aの記憶域
bの記憶域
g
main
...
0 (g1の記憶域)
200 (戻り番地)
....
0 (m2の記憶域)
0 (m1の記憶域)
aの記憶域
bの記憶域
main
....
0 (m2の記憶域)
0 (m1の記憶域)
aの記憶域
bの記憶域
100: gの開始
.....
149: EF 0 0(gの終わり)
150: mainの開始
…
199: CALL 0 100
200: STI 0 0
....

# 記憶域の移行と復帰



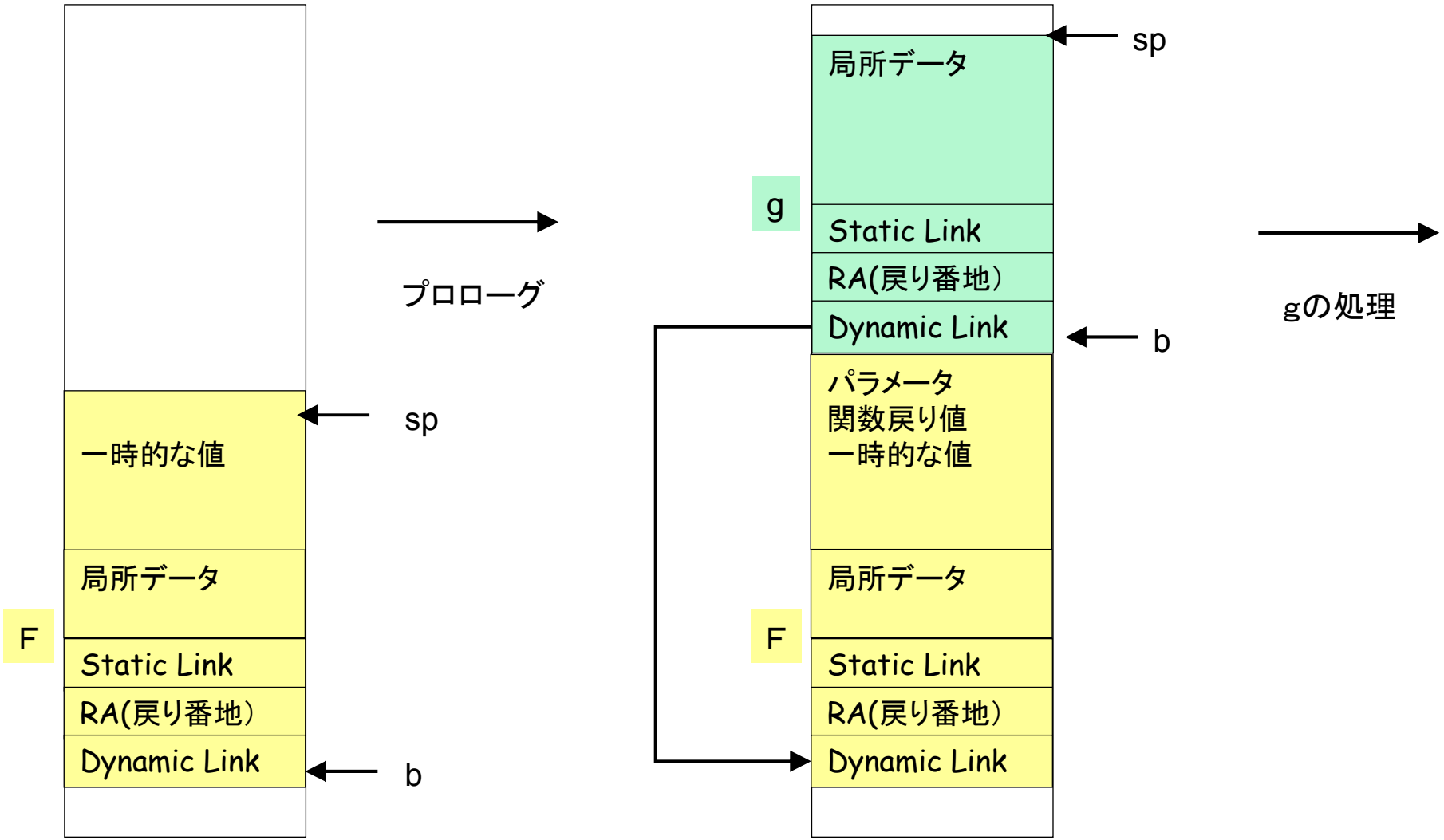

ベースポインタとスタックポインタ(トップ)
の範囲が、関数で用いる局所の記憶域

# 活性レコード(Activation Record)

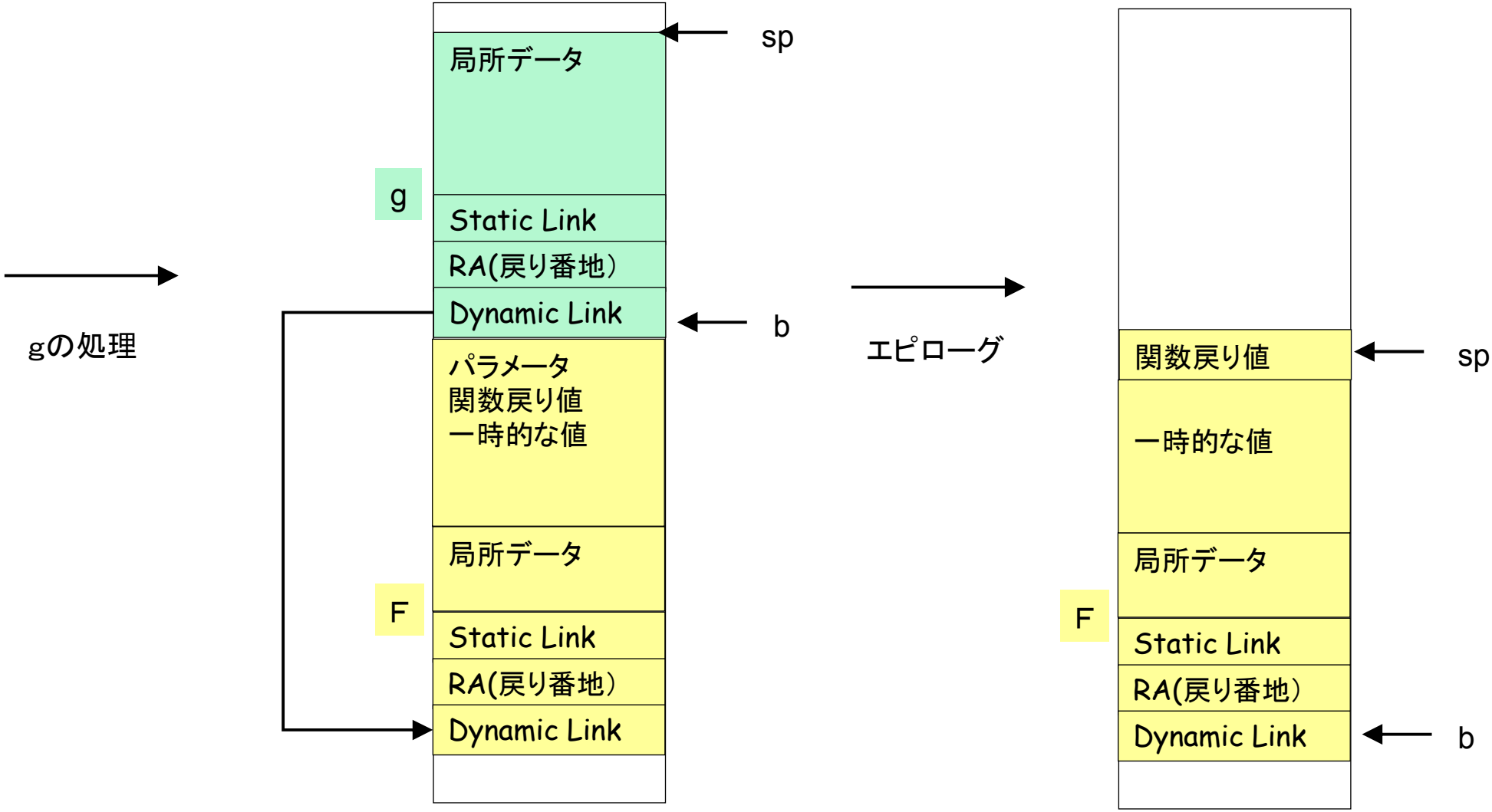



ベースポインタとスタックポインタ(トップ)
の範囲が、関数で用いる局所の記憶域

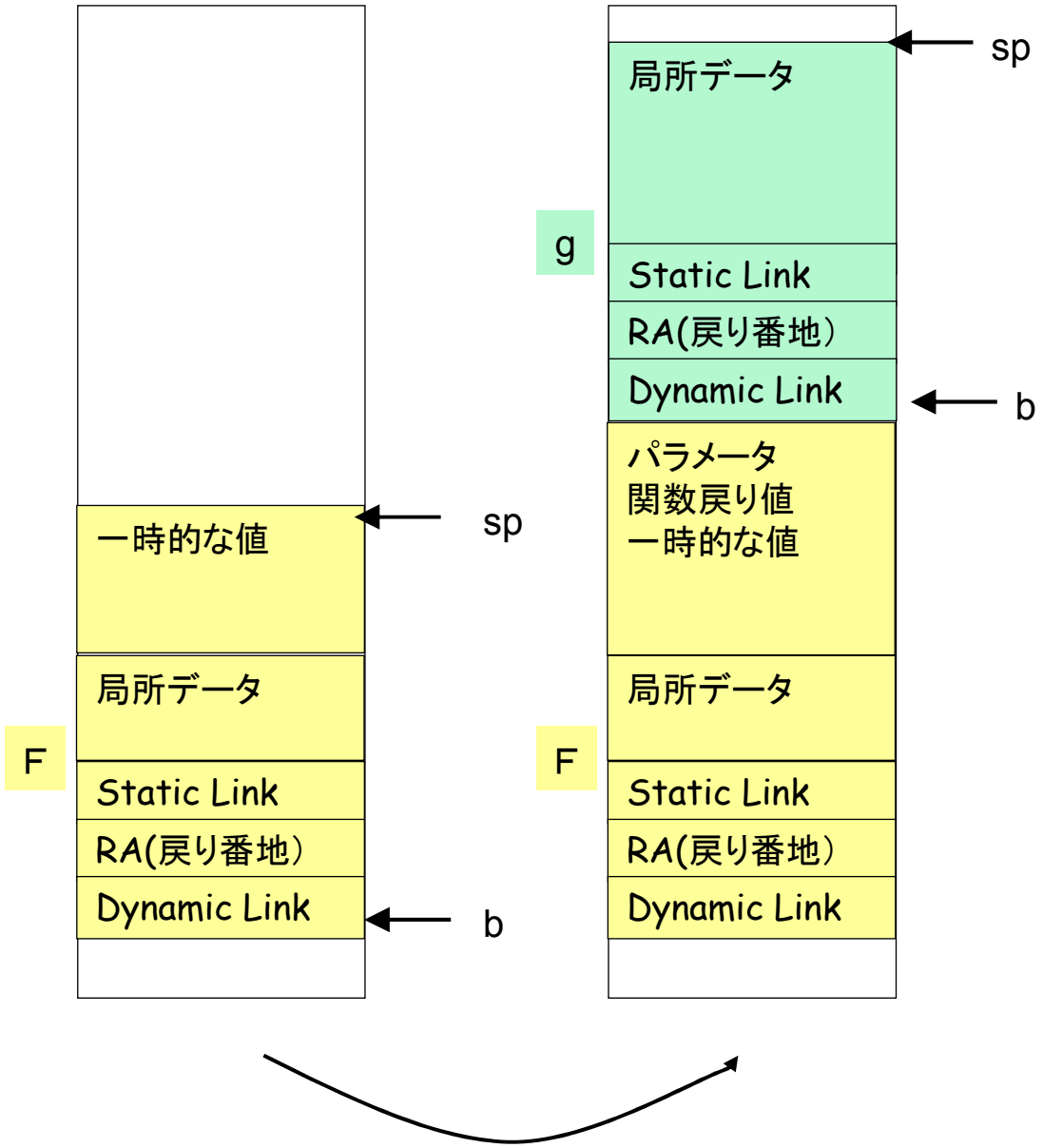
# 呼び出し列(sequence) … プロローグ



# 呼び出し列(sequence) … エピローグ



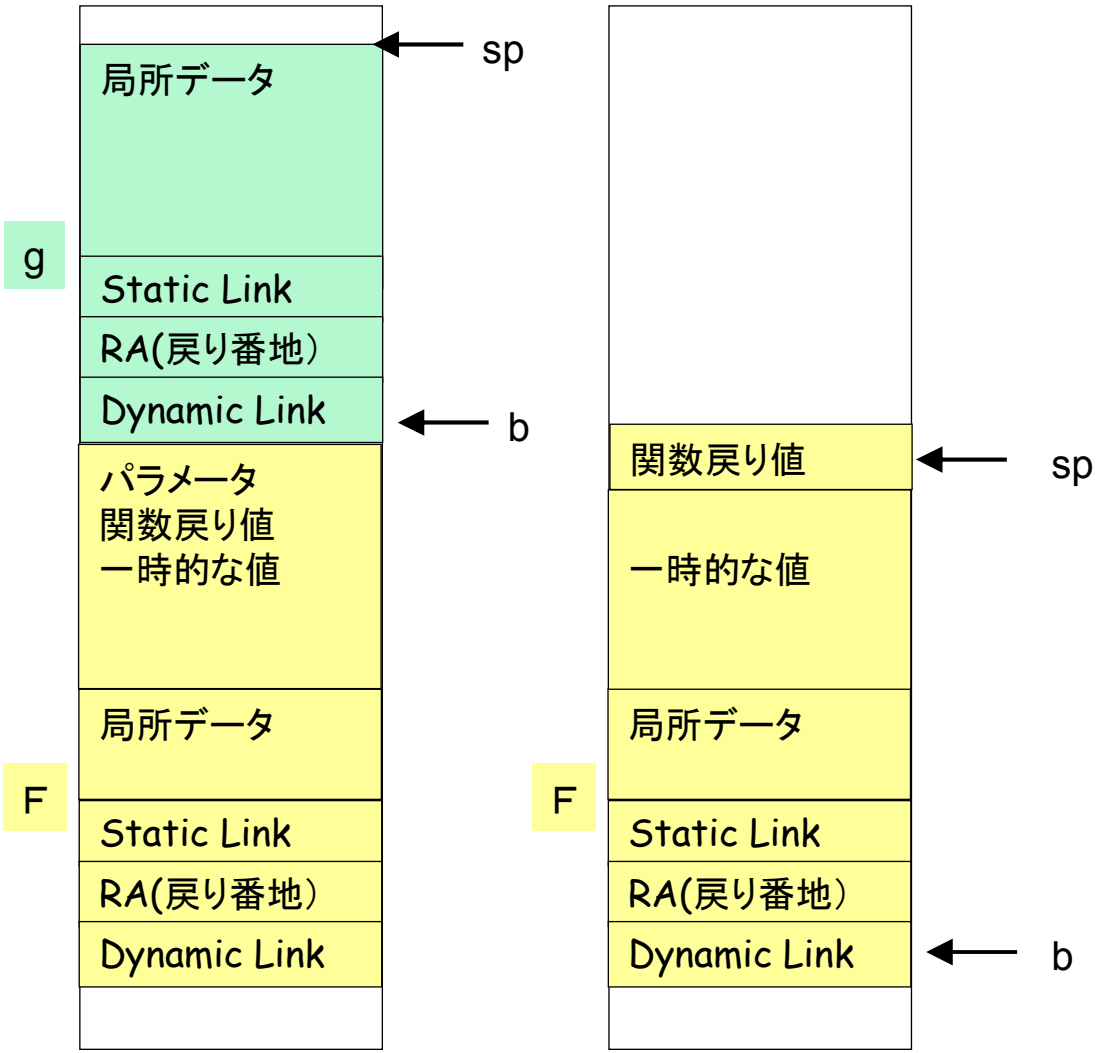
# プロローグの概要




1. 戻り値の場所を確保
2. パラメタの配置
3. Dynamic Link（動的リンク）の設定
4. Return Address（戻り番地）の設定
5. Static Link (静的リンク)の設定
6. Frame Pointer(Base Pointer)の移動
7. PCを関数g先頭番地に設定

8. 現在のレジスタを保存。(callee-save)
9. 局所データのためのメモリ確保
10. スタックポインタspの移動。

# エピローグの概要




1. 戻り値を計算する
2. レジスタを復帰
3. スタックトップ(sp)を復帰
4. Frame Pointer(Base Pointer)を復帰
5. PCを復帰（戻り番地へ戻る）

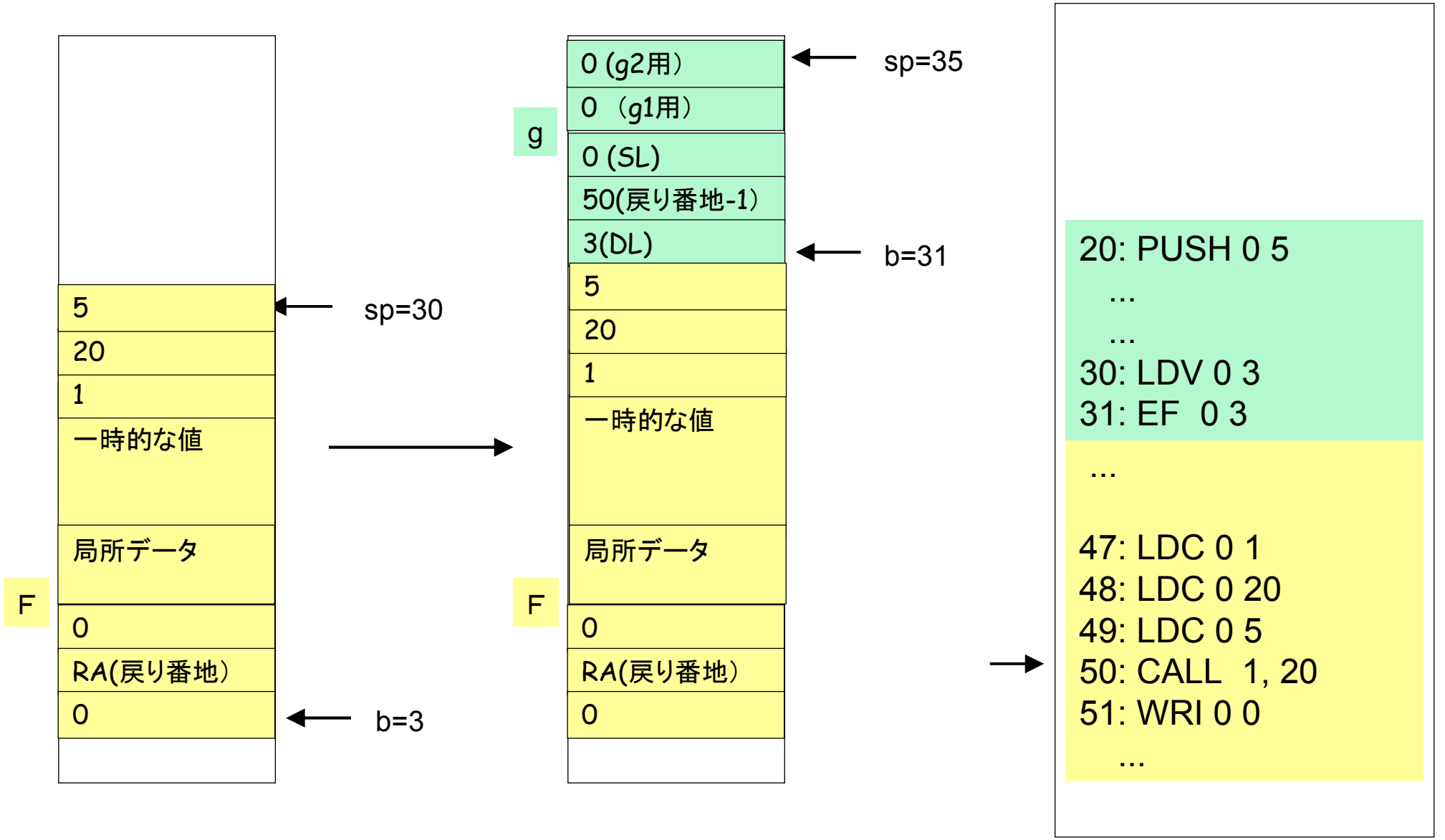
6. 戻り値を格納する

# hsmにおける例

```
int g(int x, y, z){
  int g1;
  int g2;
...
  return g1;
}

int f(){
 ..
putint( g(1,20,5) );
 ..
}
```

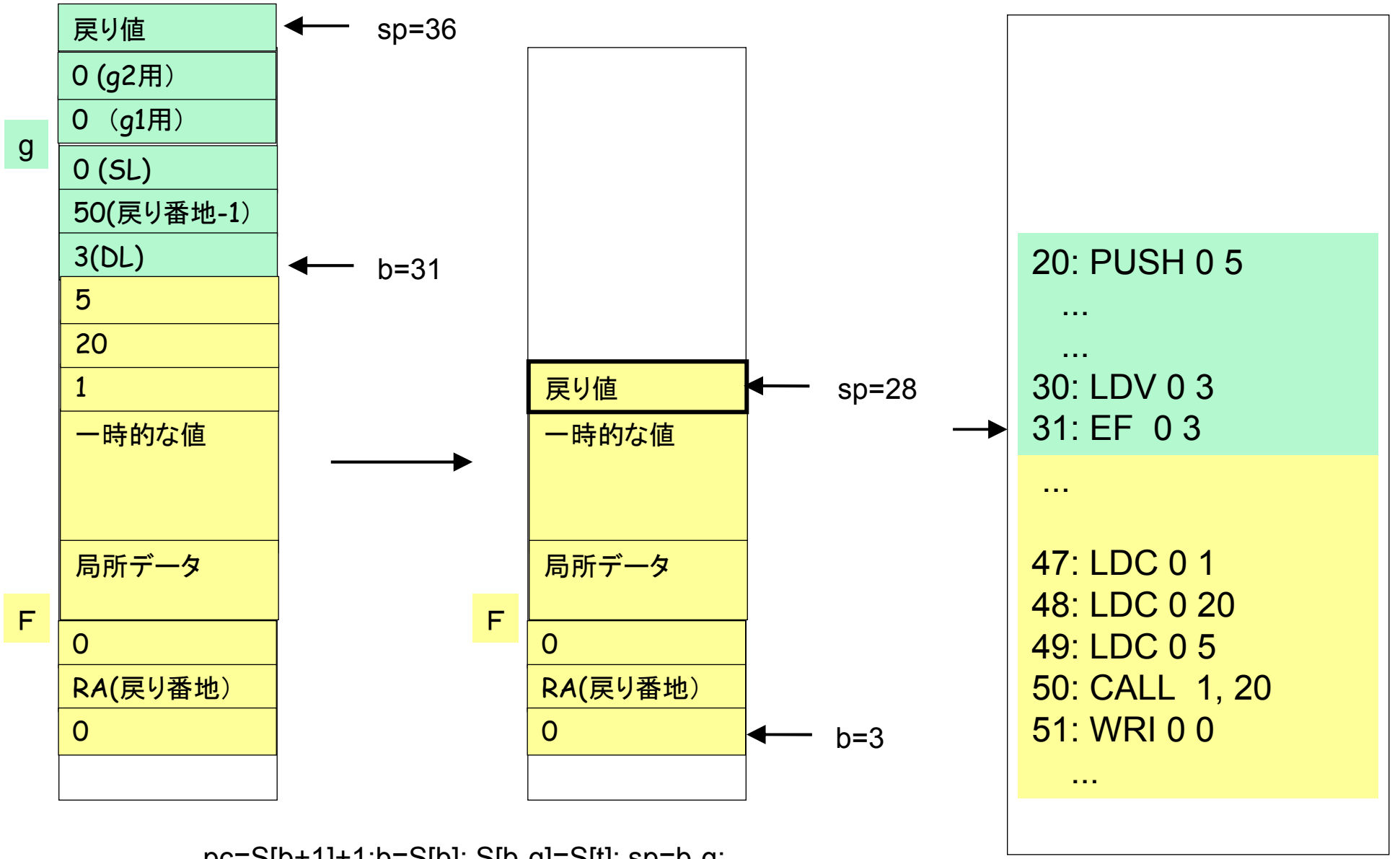
```
20: PUSH 0 5
   ...
   ...
30: LDV 0 3
31: EF  0 3
 ...

47: LDC 0 1
48: LDC 0 20
49: LDC 0 5
50: CALL  1, 20
51: WRI 0 0
    ...
```

# hsmにおけるプロローグの例




S[sp+1]=b; S[sp+2]=pc; S[sp+3]=base(p); b=sp+1; pc=q;

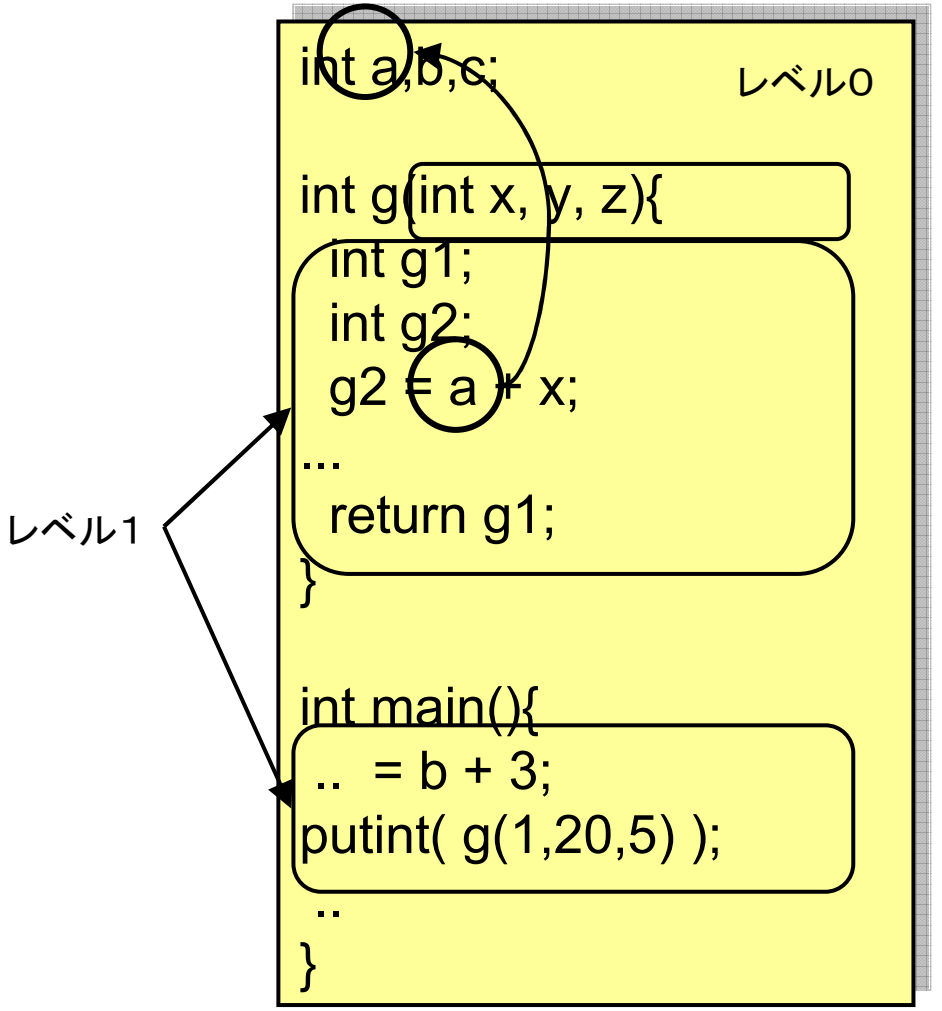
# hsmにおけるエピローグの例




pc=S[b+1]+1;b=S[b]; S[b-q]=S[t]; sp=b-q;

# 変数の参照

- 局所変数の参照
- 非局所変数の参照



# ロード・ストア命令（最終版）

LDV p, q　　ロード命令
　　sp=sp+1; S[sp]=S[base(p)+q]; pc=pc+1;

STV p, q　　ストア命令
　　S[base(p)+q]=S[sp]; sp=sp-1;pc=pc+1;

b: ベースポインタ(フレームポインタ)
base(p): pレベル下のベースポインタを返す。
　e.g. base(1) 1レベル下のベースポインタを取り出す。
　　base(0) 自分のベースポインタを返す

base(p)はStatic Link(SL)を用いて計算する。
→**スタティックリンク法**
そのほか、**Display法**という手法もある。

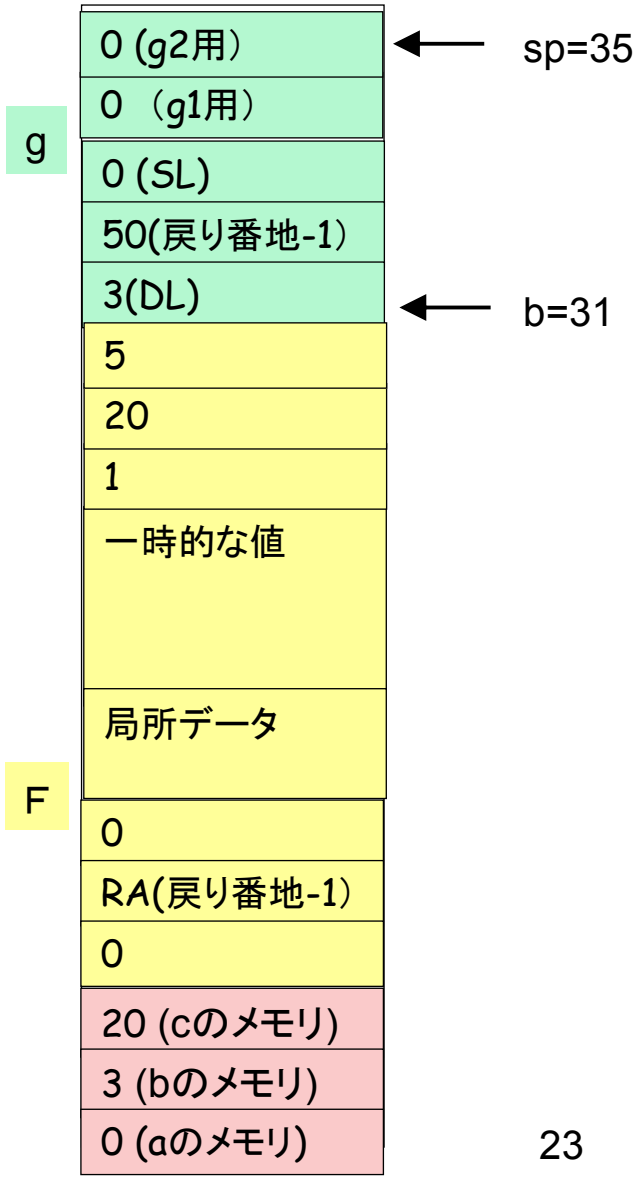
# hsmにおける参照の例(1)

LDV 0,4　-- 局所変数g2の参照
　sp=36; S[36]=S[base(0)+4]=S[31+4]=S[35]; pc++;

LDV 1,2　-- 非局所変数cの参照
　sp=36; S[36]=S[base(1)+2]=S[0+2]=S[2]; pc++;

LDV p, qロード命令
　　sp=sp+1; S[sp]=S[base(p)+q];
　　pc=pc+1;

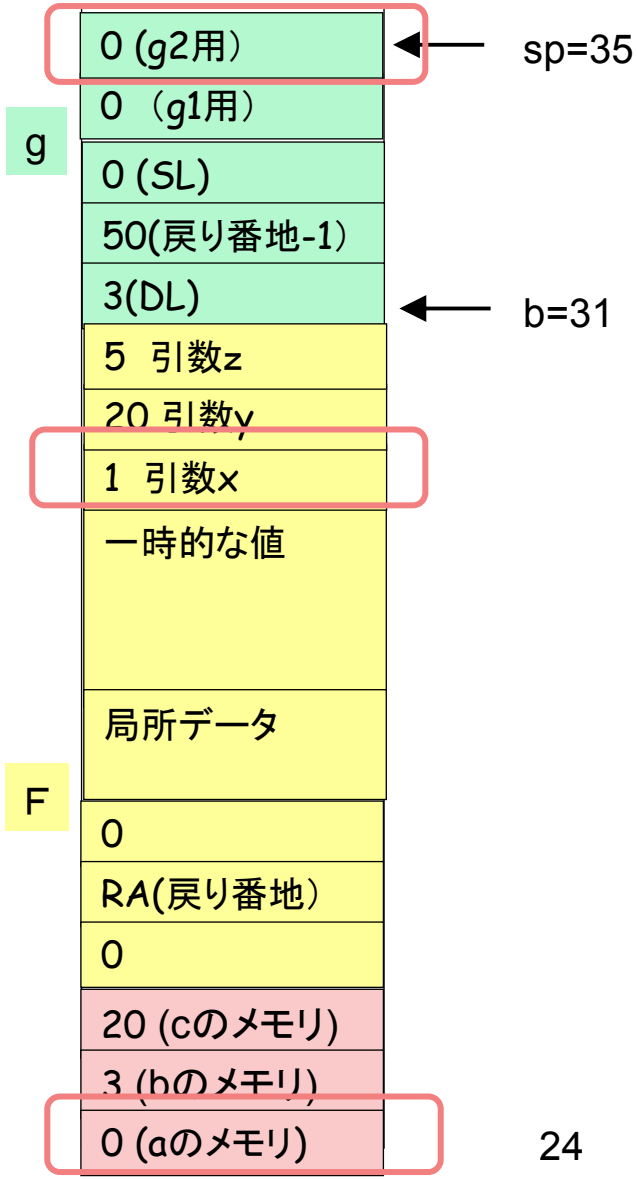
b: ベースポインタ(フレームポインタ)
base(p): pレベル下のベースポインタを返す。
　e.g. base(1) 1レベル下のベースポインタを取り出す。
　　base(0) 自分のベースポインタを返す



# 参照の例(2)

```
int a,b,c;

int g(int x, y, z){
  int g1;
  int g2;
  g2 = a + x;
...
  return g1;
}

int main(){
 ..  = b + 3;
putint( g(1,20,5) );
 ..
}
```

```
…

LDV  1 0
LDV  0 -3
AD   0 0
STV  0 4
…
```



# CALL命令(1)

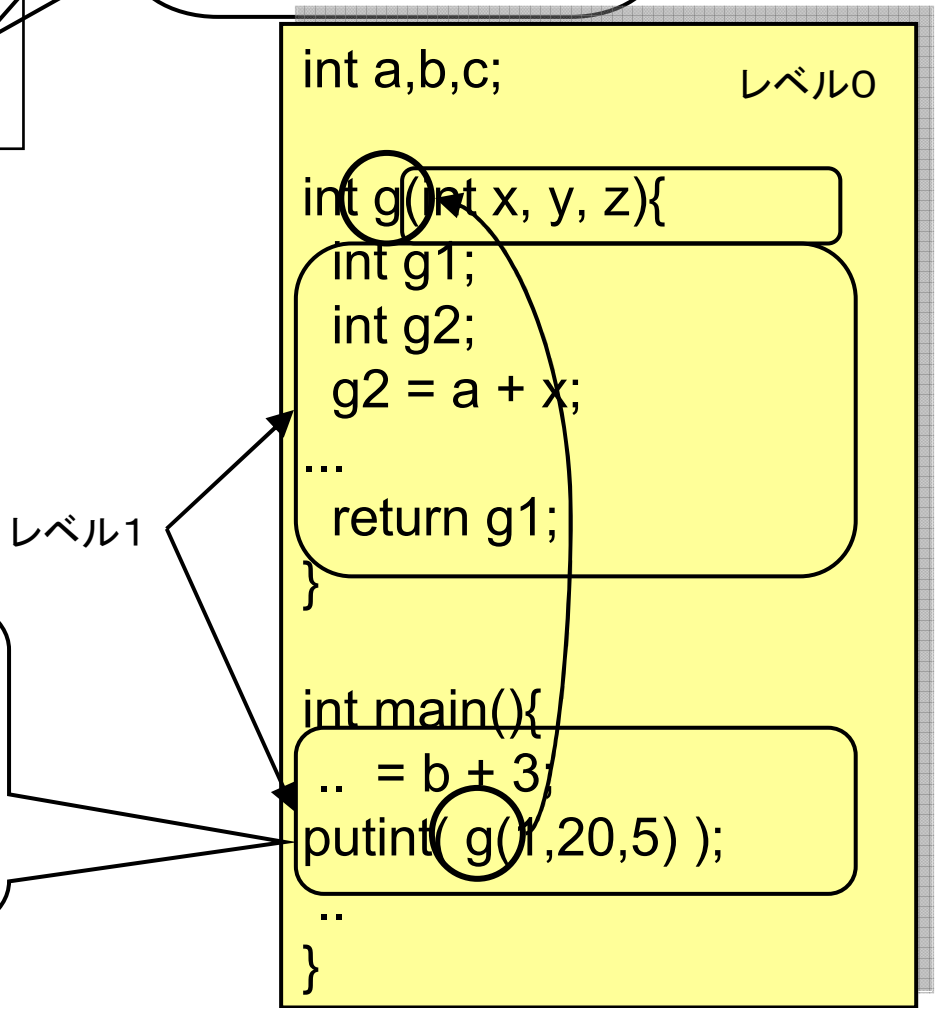
CALL p, q　　関数呼び出し
　　S[sp+1]=b; S[sp+2]=pc;
　　S[sp+3]=base(p);
　　b=sp+1; pc=q;

呼びたい関数定義のレベルが、現在のレベルと、どれだけ離れているかをqで指定する。ここでレベルとは、字面上(lexicalの)のレベル

b: ベースポインタ(フレームポインタ)
base(p): pレベル下のベースポインタを返す。
　e.g. base(1) 1レベル下のベースポインタを取り出す。
　　base(0) 自分のベースポインタを返す

現在のレベル=1
呼ぶ関数gのレベル=0

CALL 1, gのアドレス



# CALL命令(2)

最初にmainを呼ぶ場合は、大域環境でmain()の呼び出しがあると考えられる。

この場合、大域環境はレベル0、main関数の定義もレベル0なので、

CALL 0, mainの開始アドレス

となる。今回は、関数の中でネストして関数定義できないため、これ以外は、必ずレベル1において、レベル0にある関数を呼ぶので、

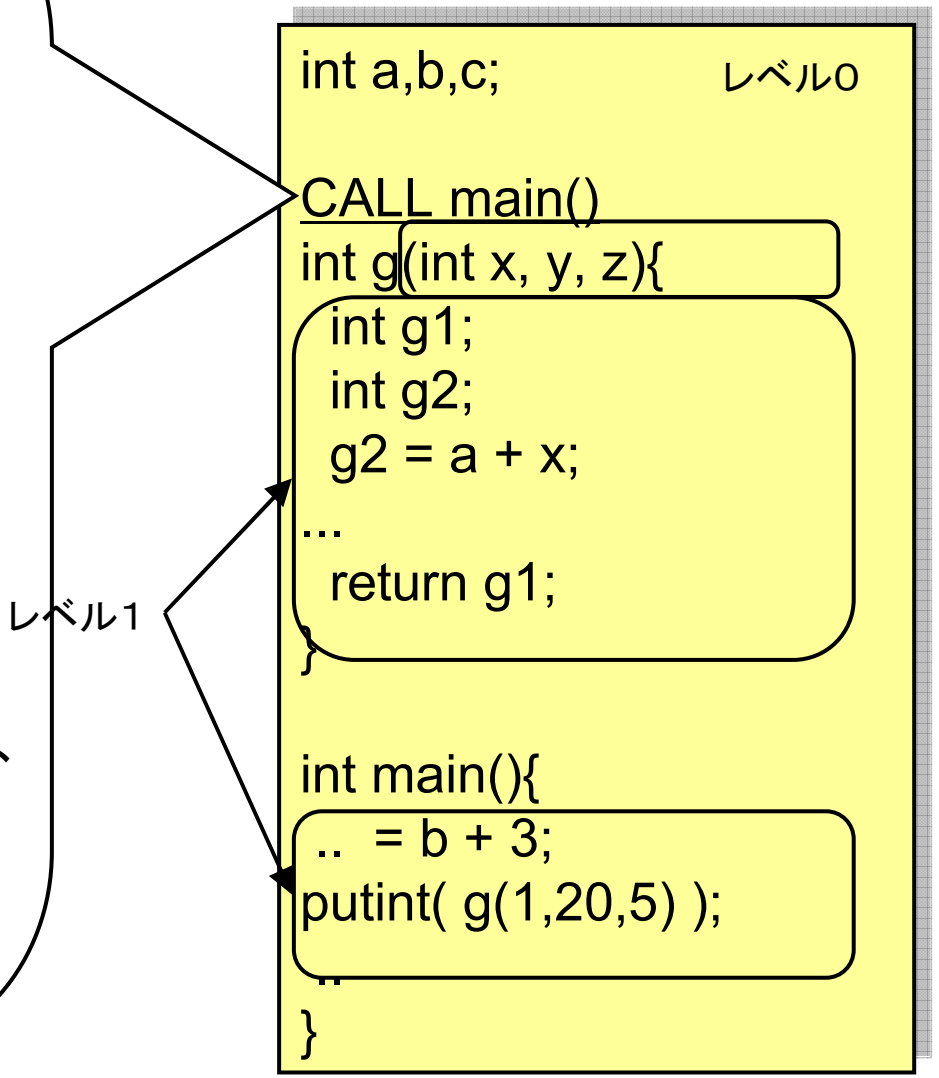
CALL 1, q

の形で関数呼び出しをすることとなる。

同様にロード命令や、ストア命令の形は今回は、次の2通りである。

LDV 0, q (局所変数の場合)

LDV 1, q (大域変数の場合)



# パラメータの渡し方

- 値呼び(call by value)
- 参照呼び(call by reference)

```
int f(int a){
   ...
  a = 20;
}


int main(){
  int x=10;
  ...
   = f(x);

}
```

```
int f(var int a){
   ...
  a = 20
}

int main(){
  int x=10;
  ...
   = f(x);

}
```

f(x)の後では、xは20となる。
参照呼びでは、xの値(10)ではなく、xのアドレス(番地)が渡される。

# 参照呼び(配列引数…余力のある人向け課題)

```
int f(int a[16]){  // Cではa[]とも書ける
   …
}


int main(){
  int array[16];
  …
   = f(array);

}
```

実際には配列のアドレスが渡される。

配列のアドレスを渡す。

引数でない配列変数
場合は
LDA 0, qだった。

関数内でa[2]の参照

```
LDV 0, -1  (引数aのアドレス)
LDC 0, 2   (a[2]の参照)
AD   0, 0
LDI   0, 0
```

関数fの呼び出し

```
,,,,

LDA  0, 3 (arrayのアドレス)
CALL 1, fの番地
…
```