Développement Sécurisé

IIA - M1

janvier 2017

- Avoir une bonne hygiène de développement
- Connaître les failles de sécurités principales d'un programme
- Apprendre la programmation défensive

Les règles générales

- connaître le langage qu'on utilise
- commenter son code
- utiliser un gestionnaire de version
- tester, tester et tester de nouveau !

Conventions de code

Objectifs

- Homogénéiser les développements
- Garantir un niveau de qualité de code

Conventions de code

Quelques règles de code propre

- nommage explicite $\rightsquigarrow$ code source plus facile à lire et à comprendre
- pas de code mort
- compilation sans erreur ni warning
- une seule instruction par ligne
- `default` obligatoire dans un switch
- pas d'opérateur ternaire
- cast explicite
- attention à la portabilité 32/64bits

Utilisation des accolades pour les conditionnelles et les boucles

ko

```
if (0 == x)
    printf(" x = 0\n");
    /* ajouter <ICI> une instruction
     * qui sera toujours executee
     */

if (0 != x)
{
    printf(" x != 0\n");
    // <code>
    // ...
}
```

ok

```
if (0 == x)
{
    printf(" x = 0\n");
}
else
{
    printf(" x != 0\n");
    // <code>
    // ...
}
```

Définition des options de compilation

- Utilisation d'un générateur de projet (CMake)

Structuration des codes de retour

Les codes d'erreur doivent être porteurs d'informations $\rightsquigarrow$ éviter le `return 1`

- chaque fonction doit retourner un code d'erreur structuré

Définition d'un point de sortie unique pour un programme C

- pas d'appel à la fonction `exit()` en plein milieu d'une fonction

Documentation de sources

Il faut documenter **avant** et **pendant**, *pas après*

- documenter les API
- documenter les algorithmes
- documenter les points *tricky*

Documentation de sources

Exemple

Gestion de version

Utilisation de **git**

Configuration via *bash*

```
$ git config -global user.name "Your Name"
$ git config -global user.email your.name@localhost
$ git config -global core.editor vim
```

Configuration via *.gitconfig*

```
[user]
        name = Your Name
        email = ...

[core]
        editor = vim
        ...
```

Gestion de version

Scénario de base

1. création d'un dépot
2. ajout d'un fichier
3. modification et sauvegarde
4. consultation de l'historique

Création d'un dépot

```
$ mkdir project1; cd project1
$ git init


$ mkdir project1.git; cd project1.git
$ git init -bare
```

Copie d'un dépot

```
$ git clone git://project1.remote.repo project1_local_repo
```

Ajout d'un fichier

```
$ vim hello.c
$ git add hello.c
$ git commit
```

Modification et sauvegarde

```
$ vim hello.c
$ git status -s
⇝ M    hello.c
$ git commit hello.c
```

Consultation de l'historique

```
$ git log
$ git log -oneline
```

1. création d'une branche
2. modification sur la branche
3. merge

1. Création d'une branche

```
$ git branch devel
$ git branch
⇝   devel
⇝ * master
```

2. Modification sur *master*

```
$ vim hello.c
$ git commit hello.c
```

3. Modification sur la branche

```
$ git checkout devel
$ git branch
⇝ * devel
⇝   master
$ vim hello.c
$ git commit hello.c
```

4. Merge

```
$ git checkout master
$ git merge devel
$ cat hello.c
```

Gestion de version

Remote server

1. clone
2. fetch/pull
3. push

Gestion de version

Remote server

1. clone

```
$ git clone <url>
$ git remote -v
⇝ origin <url fetch>
⇝ origin <url pull>
```

2. fetch/pull

```
$ git fetch [branch]
⇝ branch FETCH_HEAD
$ git merge FETCH_HEAD

$ git pull [branch]
```

3. push

```
$ git push

$ git pull
$ git push
```

Gestion de version

.gitignore

```
*.o
*.bak

/doc/html
```

en vrac

```
$ git diff [file]
$ git reset -hard
$ git checkout -b <branch>
```

Outils

- gitk
- git-gui
- atlassian

Référence

- http ://git-scm.com/
- http ://git-scm.com/book/
- http ://gitref.org/
- https ://www.atlassian.com/git/tutorial/

Génération de projets - CMake

- outil de génération multi-plateforme et multi-compilateur
- Objectif :
  - homogénéiser les chaînes de compilation
  - simplifier la gestion des projets multi-plateforme

description du projet (CMakeLists.txt) -> CMake -> Fichier de générations spécifiques à la plateforme (vcproj, makefile, ...) -> outils spécifique plateforme (VisualStudio, make, nmake) -> exe/lib

Génération de projets - CMake

CMakeLists.txt

- fichiers de description du projet
- entrée de CMkake pour générer les fichiers de génération spécifiques à la plateforme
- contient des directives CMkake permettan de déclarer les différens éléments du projet
- 1ère ligne de chaque fichier : déclarer la version minimum de CMake

```
cmake_minimum_required (VERSION 2.8)
```

- déclarer un projet (ensemble d'exécutables et de lib)

```
project <project name>
```

- déclarer un projet (ensemble d'exécutables et de lib)

```
add_executable(<executable name>
        <source file 1>
        <source file 2>
        <header file 1>
        ... )
```

- ajouter une librairie statique au projet

```
add_library(<lib name> STATIC
        <source file 1>
        <source file 2>
        <header file 1>
        ... )
```

- ajouter une librairie dynamique au projet

```
add_library(<lib name> SHARED
        <source file 1>
        <source file 2>
        <header file 1>
        ... )
```

- lier un exécutable à une bibliothèque

```
target_link_library(<executable name>
        <lib 1>
        <lib 2>
        ... )
```

Lister les générateurs

```
$ cmake -h
⇝   Visual Studio 11 Win64
⇝   Unix MakeFiles
⇝   CodeBlocks
```

Généreration

```
$ mkdir build_dir
$ cd build_dir
$ cmake -G "The Generator You Want" <source
dir>
```

Compilation

- nettoyer, compiler en *Release*

```
$ cmake -build <build dir> -clean-first -config Release
```

- compiler en *Debug*

```
$ cmake -build <build dir> -target "name" -config Debug
```

- utiliser les outils spécifique à la plateforme (VStudio, make, CodeBlocks, ...)

Génération de projets - CMake

Ressources

- `cmake -help-full` ou `cmake -help-html`
- http ://www.cmake.org/cmake/help/help.html
- http ://www.cmake.org/cmake/help/cmake_tutorial.html
- man google / man stackoverflow

Tests unitaires

Objectifs

- vérifier qu'une brique unitaire (fonction, classe, méthode de classe) rend le service attendu
- anticiper les bugs
- assurer la non-régression
- avoir *confiance* dans les briques utilisées

Tests unitaires

Bonnes pratiques

- spécifier l'interface de la brique
- spécifier le comportement de l'implémentation
- associer des tests à chaque méthode
- Au moins deux tests par exigence
  - entrées valides $\rightsquigarrow$ résultat attendu
  - entrées invalides $\rightsquigarrow$ pas de crash
- maximiser la couverture de test

Tests unitaires

Frameworks

C	Cunit, xUnit, check
C++	CppUnit, xUnit++, Boost, Google test
Java	Junit
Python	unittest, pytest

Tests unitaires

Cunit - http://cunit.sourceforge.net

Assertions

- `CU_ASSERT(int expression)`
- `CU_ASSERT_PTR_NOT_NULL(value)`
- `CU_ASSERT_STRING_NOT_EQUAL(actual, expected)`

Déclarer d'une suite de tests

- `CU_pSuite CU_add_suite(const char * strName, CU_InitializeFunc pInit, CU_CleanupFunc pClean)`

Déclarer un cas de test

- `CU_pTest CU_add_test(CU_pSuite pSuite, const char* strName, CU_TestFunc pTestFunc)`

Éxecuter les tests

- `void CU_automated_run_tests(void)`

Test unitaires

CUnit - exemple

```c
int max(int a, int b)
{
    if (a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}
```

```c
void test_max(void)
{
    CU_ASSERT(maxi(0,2) == 2);
    CU_ASSERT(maxi(0,-2) == 0);
    CU_ASSERT(maxi(2,2) == 2);
}
```

Droits et permissions Unix

Droits

```
drwxr-x--- 2 oliv oliv 4096 juil. 16 08:31 dir_01
drwxr-x--- 2 oliv oliv 4096 juil. 16 08:26 dir_02
lrwxrwxrwx 1 oliv oliv   17 juil. 16 08:31 doc.txt -> dir_01/my_doc.txt
-rw-r----- 1 oliv oliv   14 juil. 16 08:26 README.txt
```

type fichier normal (`-`), répertoire (`d`), lien symbolique (`l`)

u/g/o **r**eadable | **w**ritable | e**x**ecutable

Changement de droits

`chmod [u g o a] [+ - =] [r w x]`

```
--- 0 -> 000 -----> aucun droit
--x 1 -> 001 -----> execution
-w- 2 -> 010 -----> ecriture
-wx 3 -> 011 -----> ecriture-execution
r-- 4 -> 100 -----> lecture
r-x 5 -> 101 -----> lecture-execution
rw- 6 -> 110 -----> lecture-ecriture
rwx 7 -> 111 -----> lecture-ecriture-execution
```

`sticky`

Un utilisateur ne peut supprimer que les fichiers qui lui appartiennent

`setuid`

Le binaire qui est **setuid** sera exécuté avec les droits de son propriétaire

`setgid`

Le binaire qui est **setgid** sera exécuté avec les droits du groupe auquel il appartient

Changement de droits

- `chmod [u g] +s file`
- `chmod [1/2/4]mode file` (setuid=4, setgid=2, sticky=1)

Droits et permissions

démo time

- `command_wrapper`

ID

UID

real UID Identifiant **réel** du propriétaire du process (*login de session*)

effective UID Identifiant sous lequel est lancé le process

saved UID Copie de *effective UID*

```
uid_t real_uid;       // real UID
uid_t effective_uid;  // effective UID
uid_t saved_uid;      // saved set-user-ID

getresuid(&real_uid, &effective_uid, &saved_uid);
```

GID

La même notion existe pour les GID

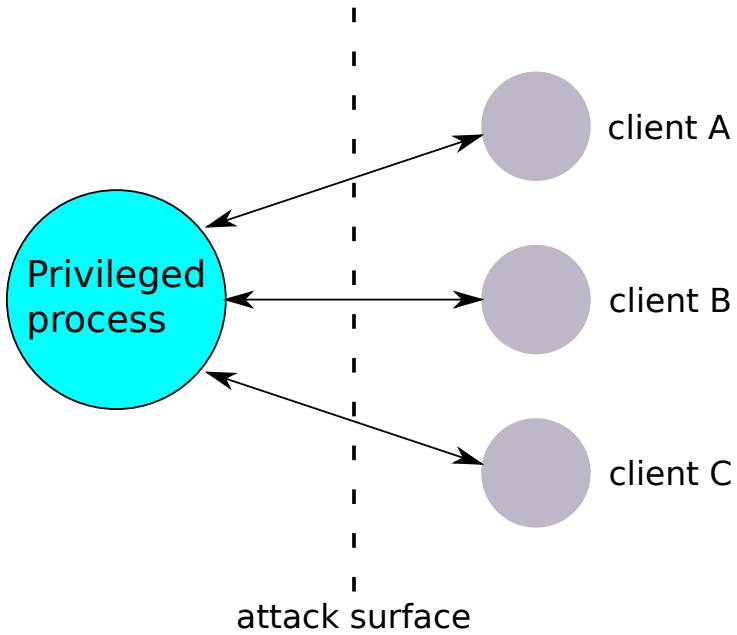
```
gid_t real_gid;       // real GID
gid_t effective_gid;  // effective GID
gid_t saved_gid;      // saved set-group-ID

getresgid(&real_gid, &effective_gid, &saved_gid);
```

Séparation des privilèges

Principe

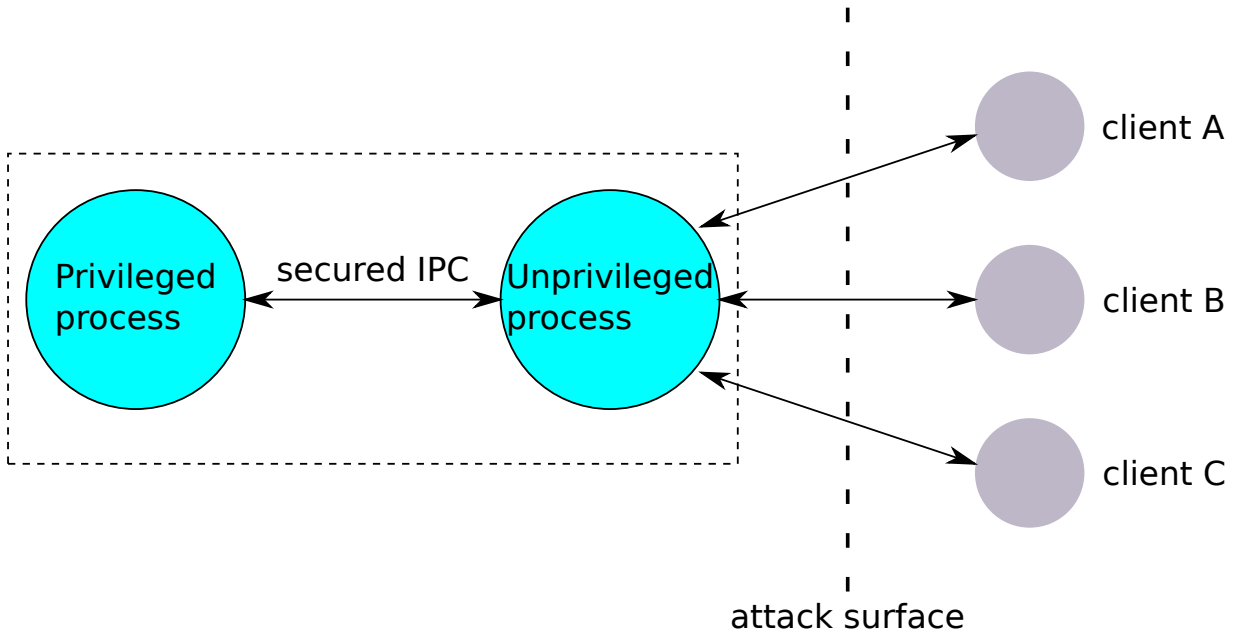
- Minimiser la partie du programme qui requiert des privilèges élevés



Séparation des privilèges

Principe

- Minimiser la partie du programme qui requiert des privilèges élevés



Séparation des privilèges

Perte des droits privilégiés

- obtenir l'UID et le GID d'un utilisateur restreint (*nobody*)
- on s'enferme dans un répertoire vide (*/var/empty*)
- perdre les droits liés au groupe puis à l'utilisateur

Séparation des privilèges

Séparation de privilèges

- `fork()`
- établissement d'une *IPC* entre le père et le fils
- le père : ne gère que la partie sensible
- le fils : *chroot* + perte des droits privilégiés

Séparation des privilèges

Bibliothèques existantes

- privman
- imsg
- ...

CWE (Common Weakness Enumeration)

- `https://cwe.mitre.org`
- Classification des failles (langage, popularité, ...)

```
1    printf("<title>Blissfully Ignorant, Inc.</title>");
2    ftype =  Get_Query_Param("MessageType");
3    strcpy(fname, "/home/cwe/");
4    strcat(fname, ftype);
5    strcat(fname, ".dat");
6    handle = fopen(fname, "r");
7    while(fgets(line, 512, handle)) {
8        if (strncmp(line,"<script>",8)) {
9            printf(line); } }
10   return(200);
```

Combien de failles identifiez-vous ?

CWE-120 Classic Buffer Overflow (lines 2, 4, 10)
CWE-23 Relative Path Traversal (lines 2, 4, 6)
CWE-79 Failure to Preserve Web Page Structure (XSS) (lines 7, 9)
CWE-134 Uncontrolled Format String (lines 7, 9)
CWE-476 NULL Pointer Dereference (lines 6, 7)
CWE-20 Improper Input Validation (lines 2, 8)
CWE-116 Improper Encoding or Escaping of Output (lines 7, 9)
CWE-73 External Control of File Name or Path (lines 2, 4, 6)
CWE-404 Improper Resource Shutdown or Release (lines 6, 10)
CWE-252 Unchecked Return Value (lines 2, 4, 6)

Failles de sécurité

Règles du jeu

- développeur : joue en défense, essaie de prévoir tous les comportements possibles de son programme (...)
- attaquant : tous les coups sont permis (une seule erreur lui suffit)

Terrains de jeu

flot d'information : changer ou modifier les données

flot d'exécution : modifier ou contrôler le processus

déni de service : saturer les ressources disponibles

Registres du processeur x86

- registres généraux (ou de travail)
- registres d'offset (de déplacement, de pointeur, ou d'index)
- registres de segment
- registres de flag (d'états, des indicateurs, d'exception ou de drapeaux)

Registres du processeur x86

Les registres de travail (32 bits)

- EAX : Extended Accumulator Register
- EBX : Extended Base Register
- ECX : Extended Counter Register
- EDX : Extended Data Register

Les registres de segments (16 bits)

- CS : Code segment
- DS : Data segment
- SS : Stack segment
- ES : Extra segment
- FS : à partir du 386
- GS : à partir du 386

Registres du processeur x86

Les registres d'index (32 bits) : opérations sur chaines de caractères

- ESI : Extended Source Index : source
- EDI : Extended Destination Index : destination

Les registres d'offset (32 bits)

- EBP : Extended Base Pointer : Pointeur de référence
- ESP : Extended Stack Pointer : Pointeur sur le dernier élément de la pile
- EIP : Extended Instruction Pointer : Pointeur sur la prochaine instruction

Les registres speciaux

- EFLAGS : Extended Flags : Registre de drapeaux

Registres du processeur x86

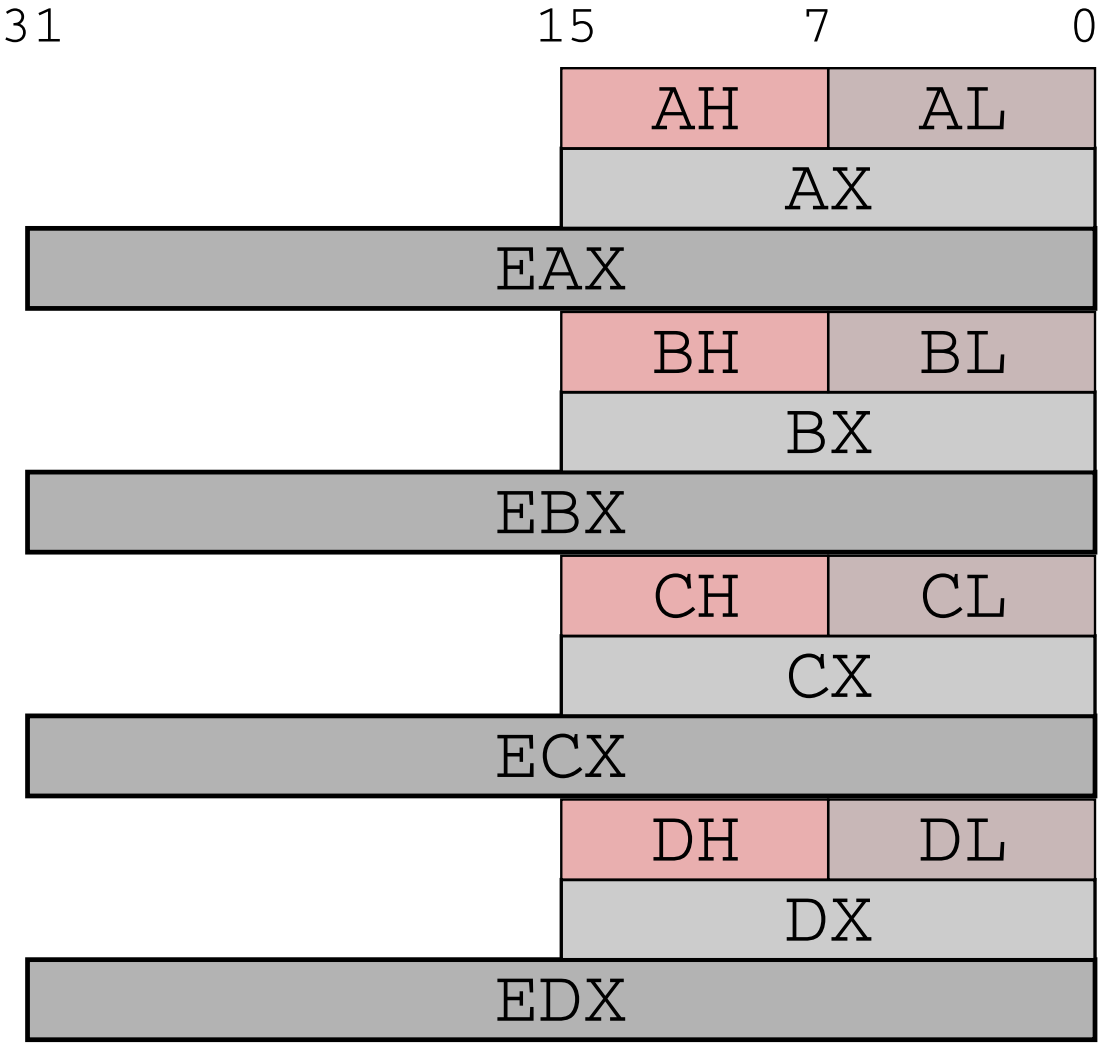


Figure – Registres generaux

Registres du processeur x86

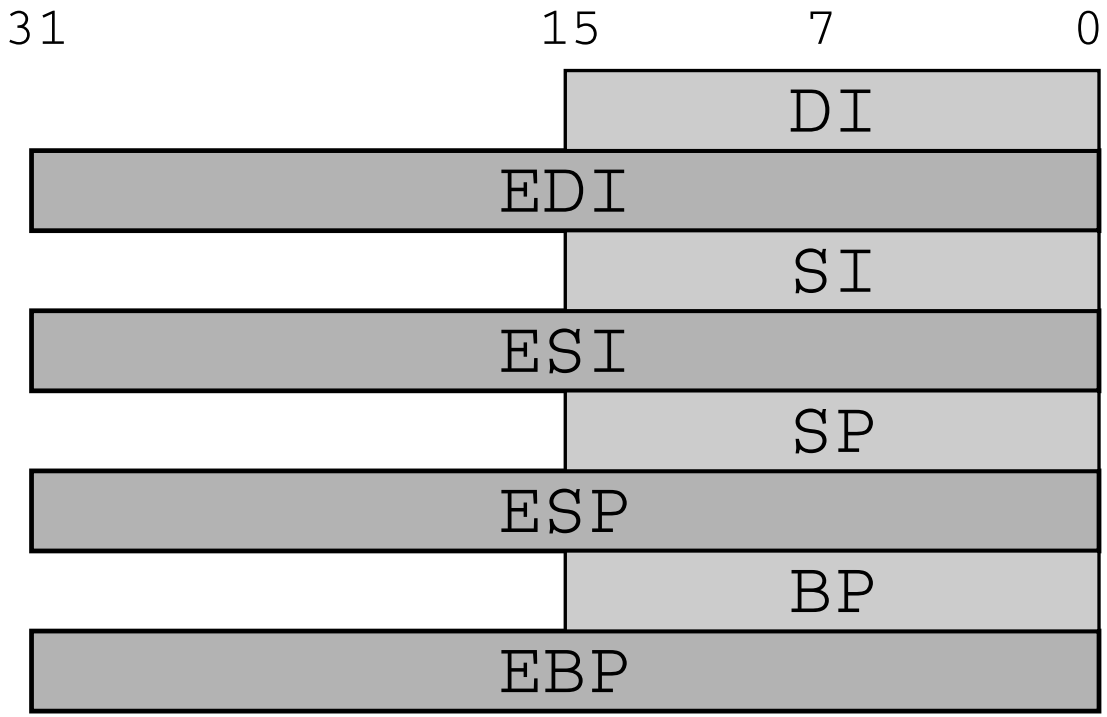


Figure – Registres d'offset

Registres du processeur x86

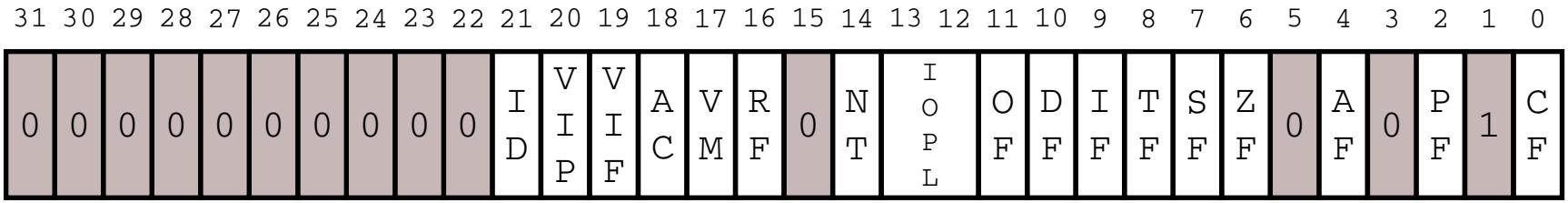


Figure – Structure du registre EFLAGS

ID	ID Flag	System
VIP	Virtual Interrupt Pending	System
VIF	Virtual Interrupt Flag	System
AC	Aligment Check	System
VM	Virtual 8086 Mode	System
RF	Resume Flag	System
NT	Nested Task	System
IOPL	IO Privilege Level	System
OF	Overflow Flag	System
DF	Direction Flag	Control
IF	Interrupt Enable Flag	System
TF	Trap Flag	System
SF	Sign Flag	Status
ZF	Zero Flag	Status
AF	Auxilliary Carry Flag	Status
PF	Parity Flag	Status
CF	Carry Flag	Status

Format des instructions x86

```
mnemonic [operand [operand]]
```

```
inc eax              ; Incremente la valeur stockee dans eax
mov eax, 0           ; Place la valeur 0 dans eax
ret                  ; Sortie de fonction
```

Les opérandes

- opérande registre (8/16/32 bits) :

```
inc eax
add eax, ebx
```

- opérande immediate (8/16/32 bits) :

```
mov eax, 0xaabbccdd
```

- opérande memoire (8/16/32 bits) :

```
mov eax, [ebx]
mov eax, [ebx+0x40000]
mov eax, [ebx+ecx*3]
mov eax, [ebx+ecx*3+0x400000]
```

La pile

Notion de pile

- Méthode de gestion de la memoire en LIFO
- Utilisée pour stocker temporairement des données

La pile en x86

- la pile se remplit vers les adresses décroissantes
- l'adresse du dernier element de la pile pointe par registre `esp`
- les instructions spéciales pour manipuler la pile :

Instruction	Equivalent
push val32	sub esp, 4 mov dword [esp], val32
pop reg	mov reg, dword [esp] add esp, 4

Appel de fonctions

1. charger l'adresse de la procédure dans `eip`
2. éxecuter la procédure
3. revenir après l'appel

Appel de fonctions

1. charger l'adresse de la procédure dans `eip`
2. éxecuter la procédure
3. revenir après l'appel

Convention d'appel

convention d'appel = contrat en l'appelant et l'appelé

- responsable de la correction de la pile (appelant, appelé)
- espace de stockage pour les arguments (registres, pile, ...)
- espace de stockage de la valeur de retour (registre, pile, ...)
- registres préservés (par l'appelé)

Conventions d'appel (x86)

cdecl

- la fonction **appelante** nettoie la pile
- les arguments sont passés par la pile (empilés du dernier au premier)
- la valeur de retour est stockée dans le registre `eax`
- registres préservés : `ebx`, `esi`, `edi`, `ebp`

stdcall

- la fonction **appelée** nettoie la pile
- les arguments sont passés par la pile (empilés du dernier au premier)
- la valeur de retour est stockée dans le registre `eax`
- registres préservés : `ebx`, `esi`, `edi`, `ebp`

fastcall

- la fonction appelante nettoie la pile
- les 2 premiers arguments sont enregistrés dans `ecx`, `edx`, les autres sont passés par la pile (empilés du dernier au premier)
- la valeur de retour est stockée dans le registre `eax`
- registres préservés : `ebx`, `esi`, `edi`, `ebp`

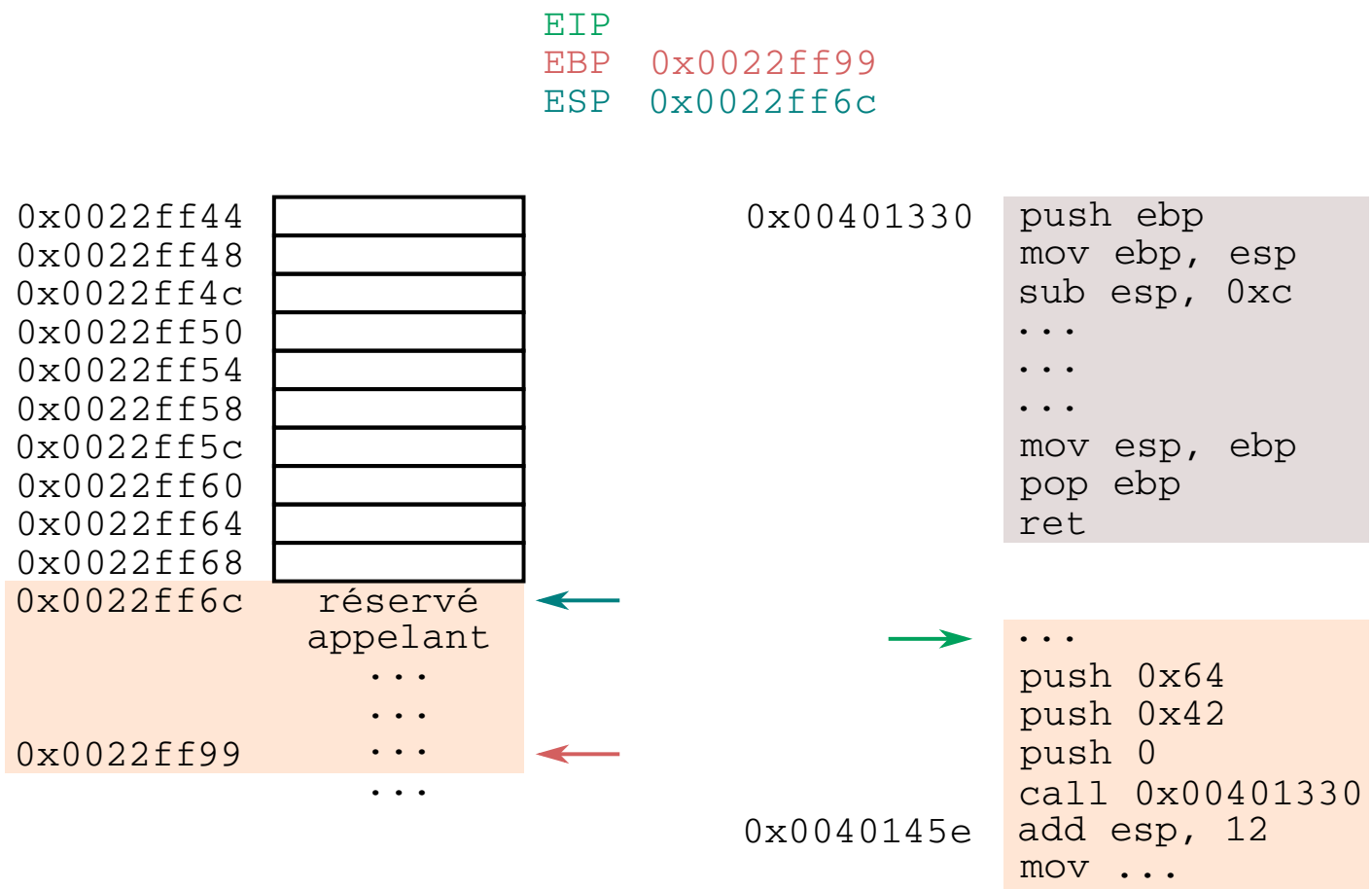
Conventions d'appel (x86)

`pascal`

- la fonction **appelée** nettoie la pile
- les arguments sont passés par la pile (empilés du premier au dernier)
- la valeur de retour est stockée dans le registre `eax`
- registres préservés : `ebx`, `esi`, `edi`, `ebp`

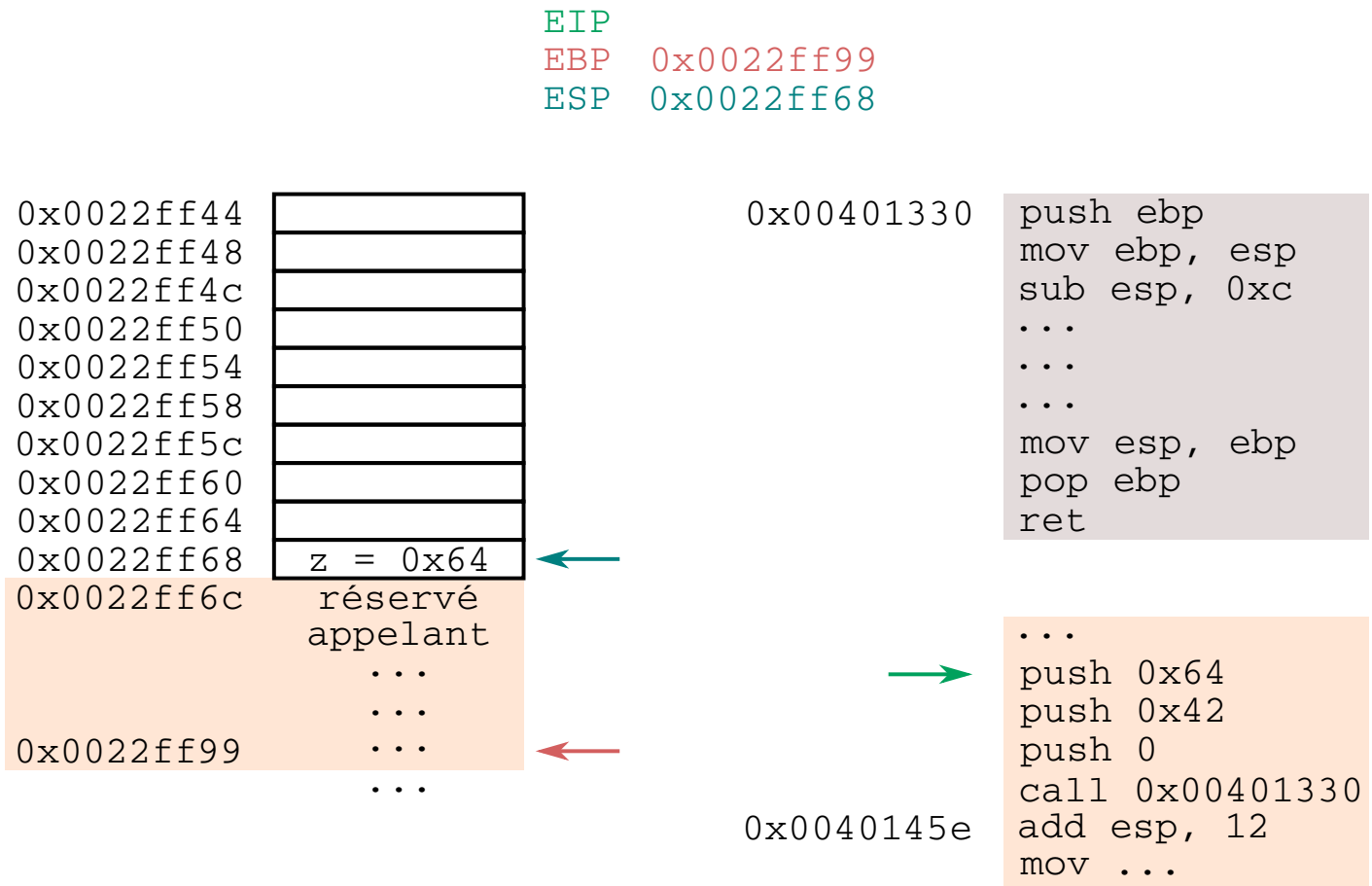
Appel de fonctions

convention CDECL



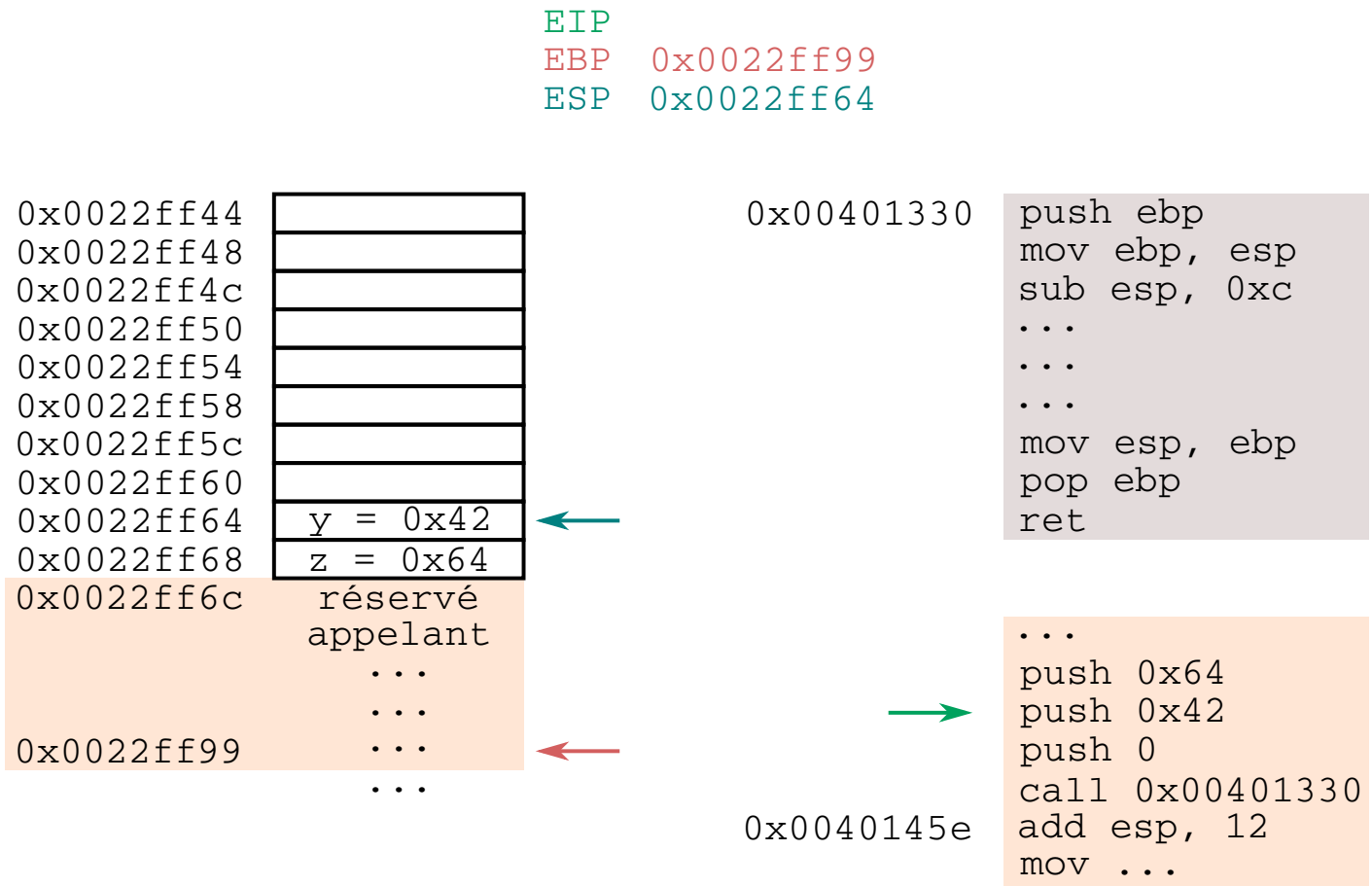
Appel de fonctions

convention CDECL



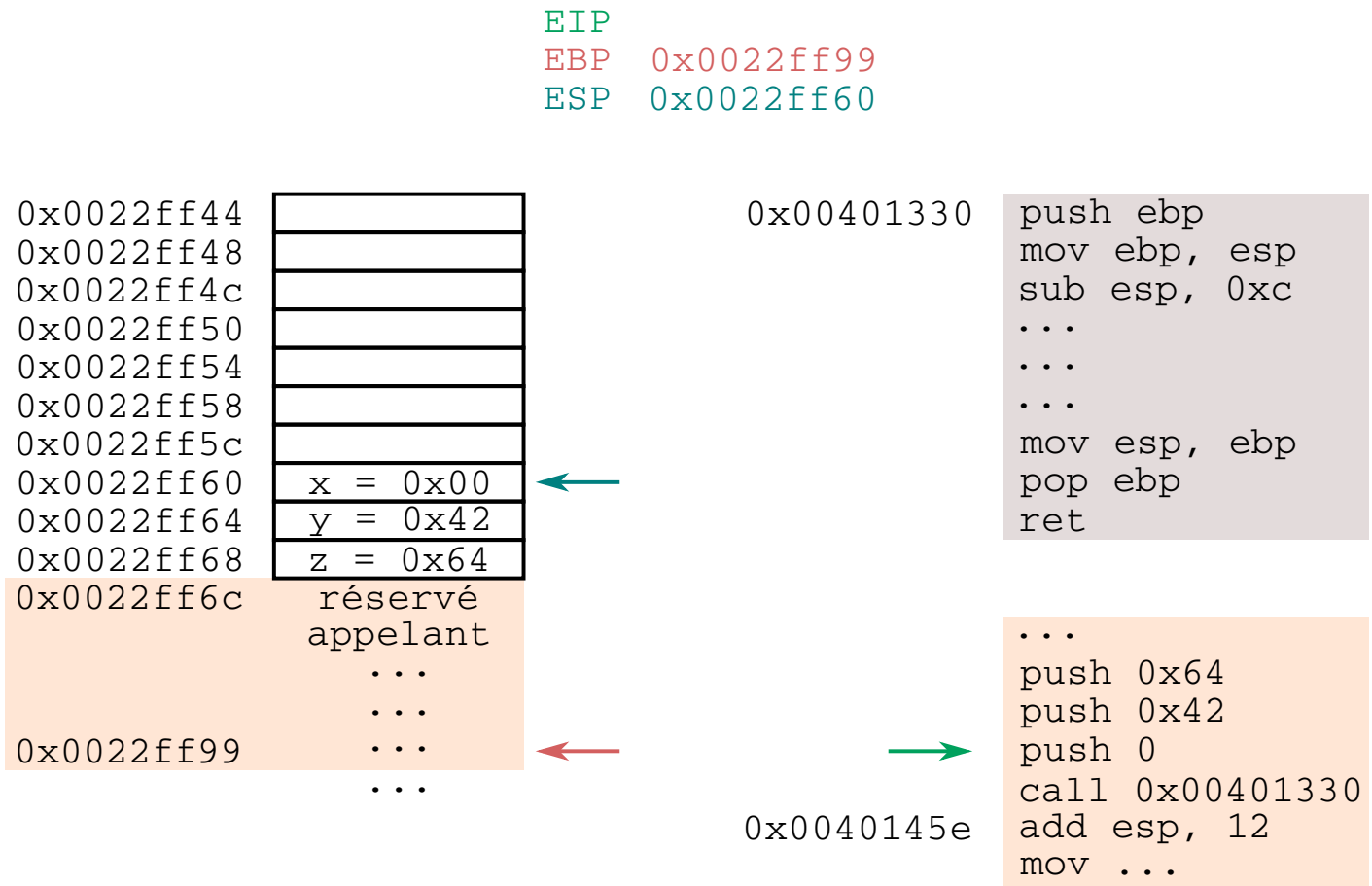
Appel de fonctions

convention CDECL



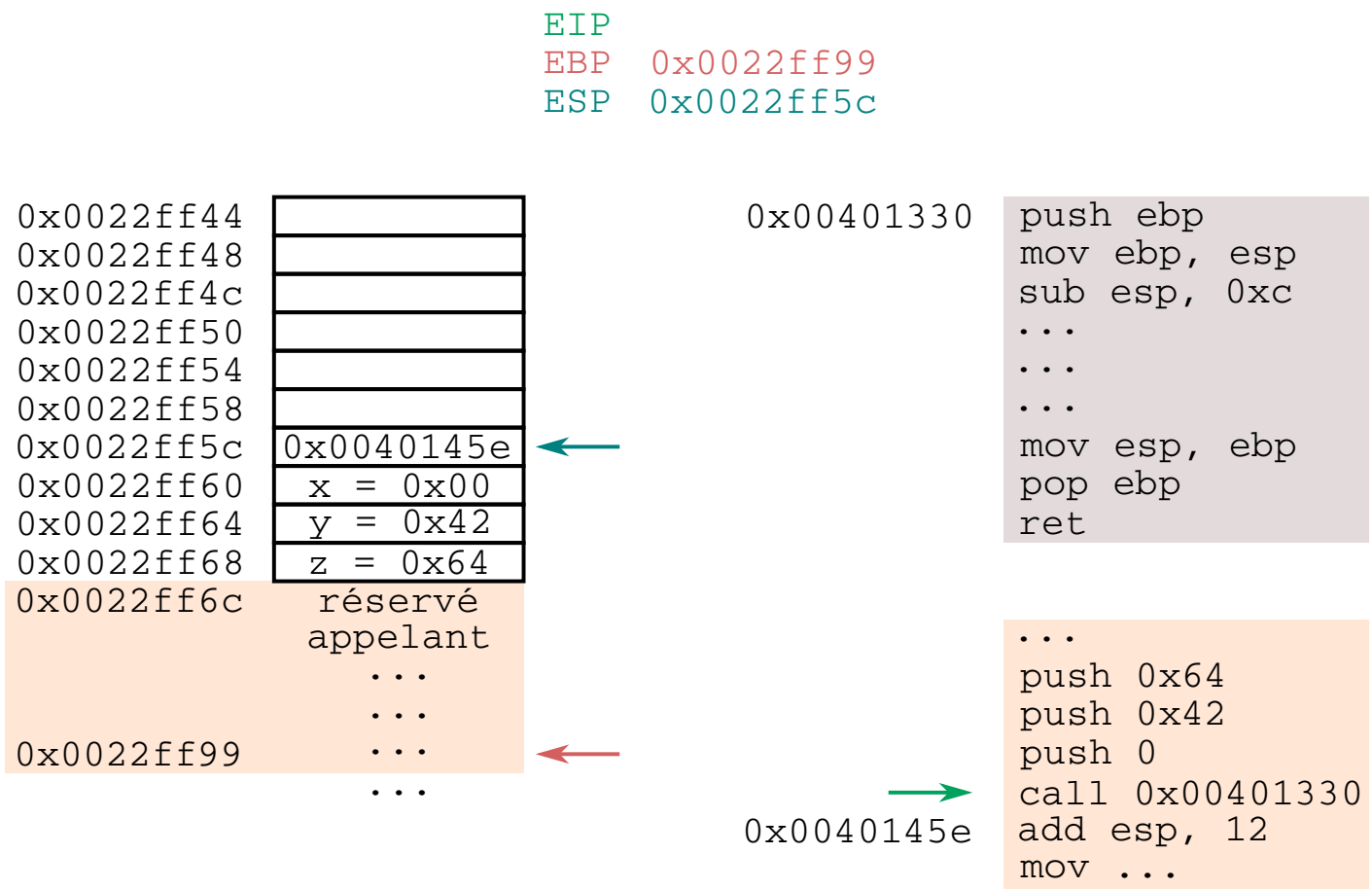
Appel de fonctions

convention CDECL



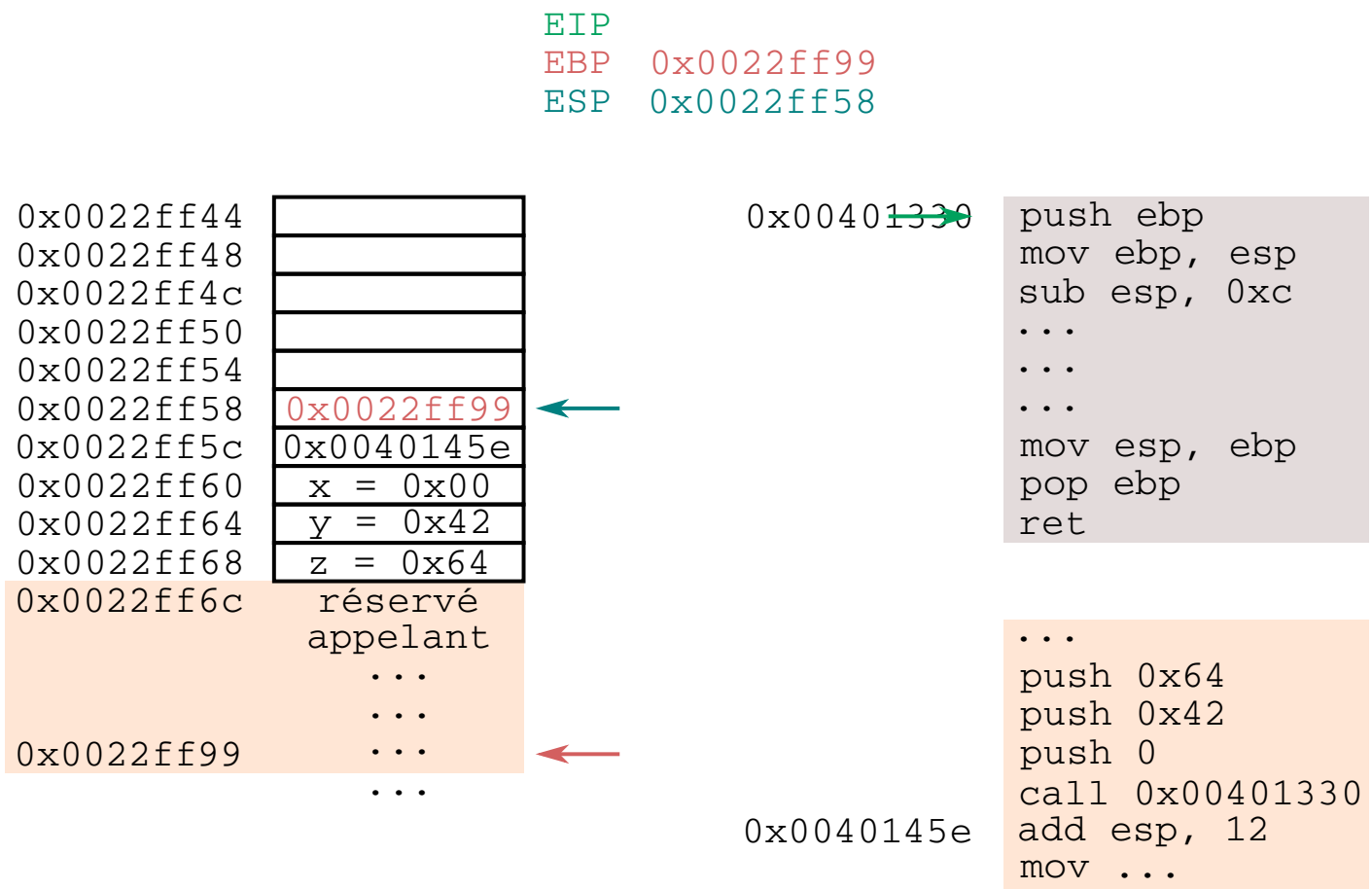
Appel de fonctions

convention CDECL



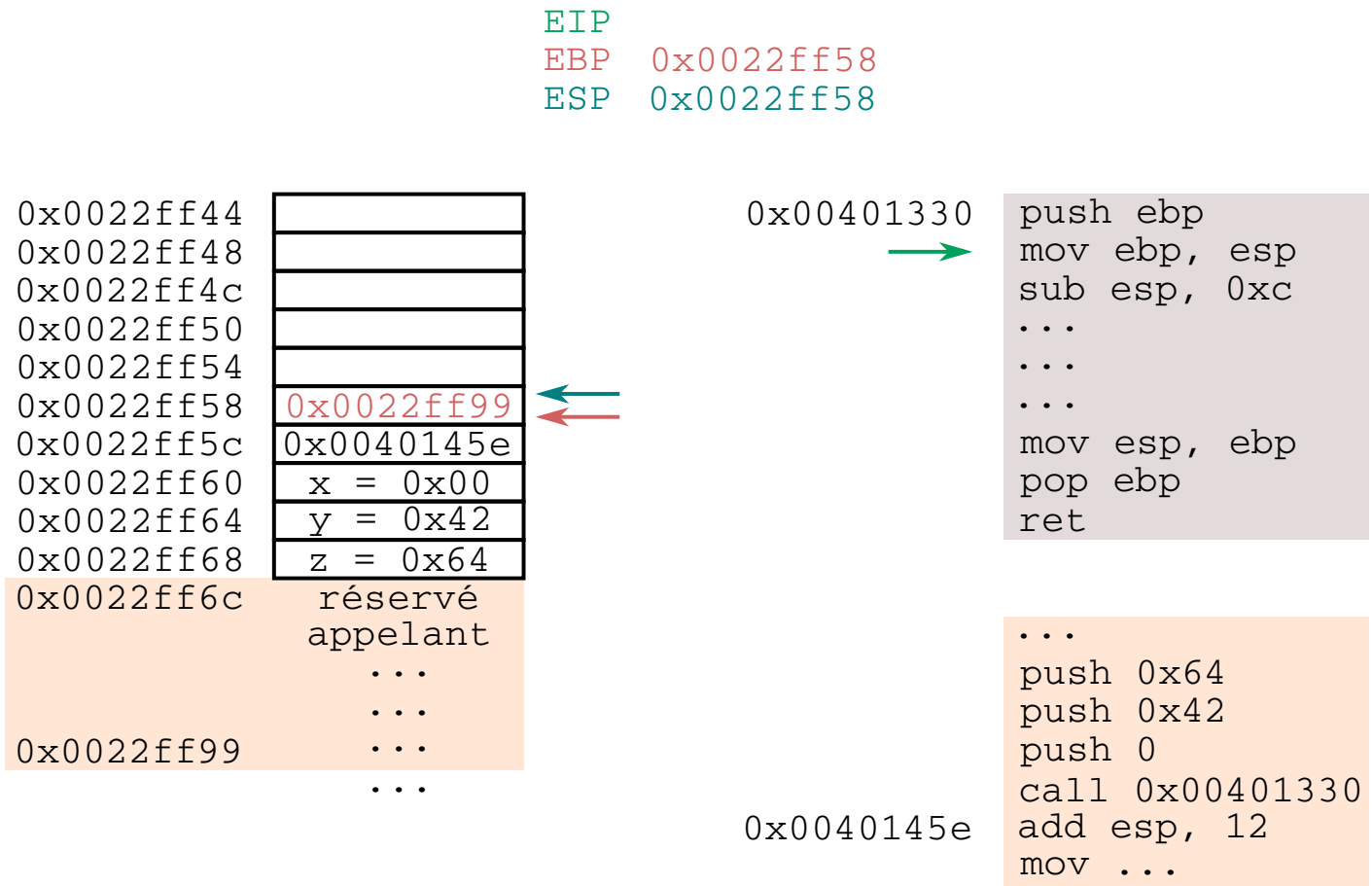
Appel de fonctions

convention CDECL



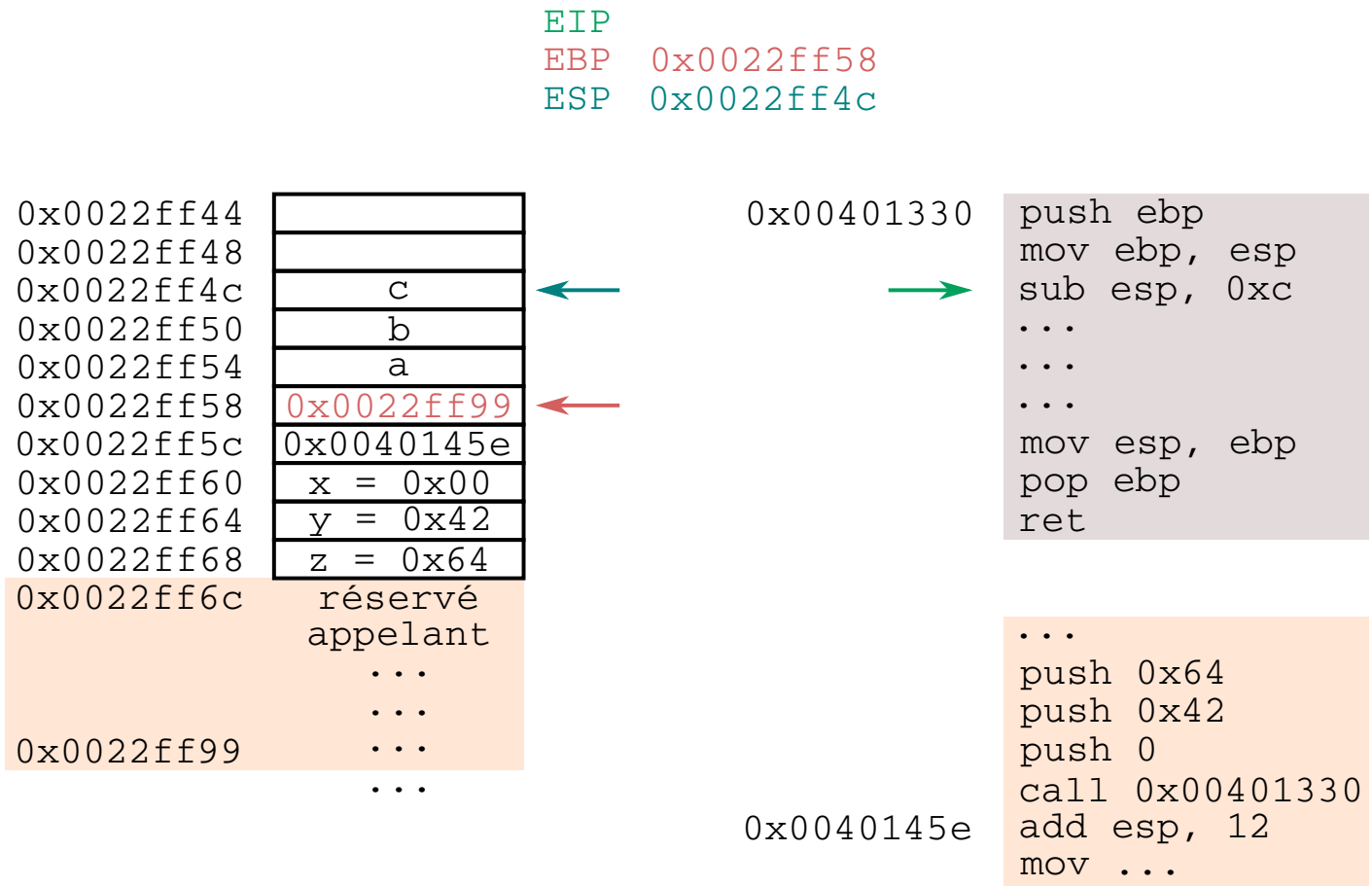
Appel de fonctions

convention CDECL



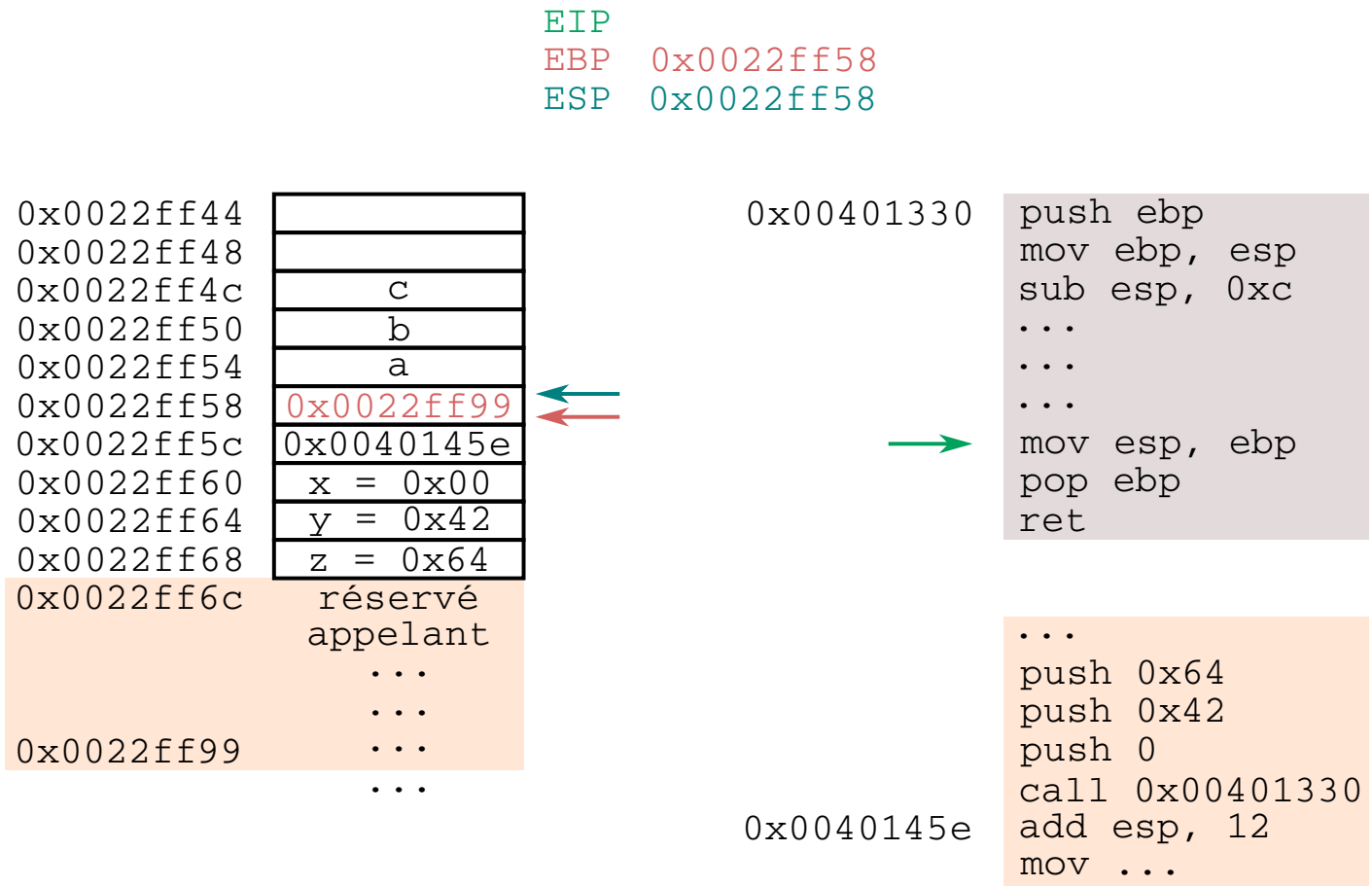
Appel de fonctions

convention CDECL



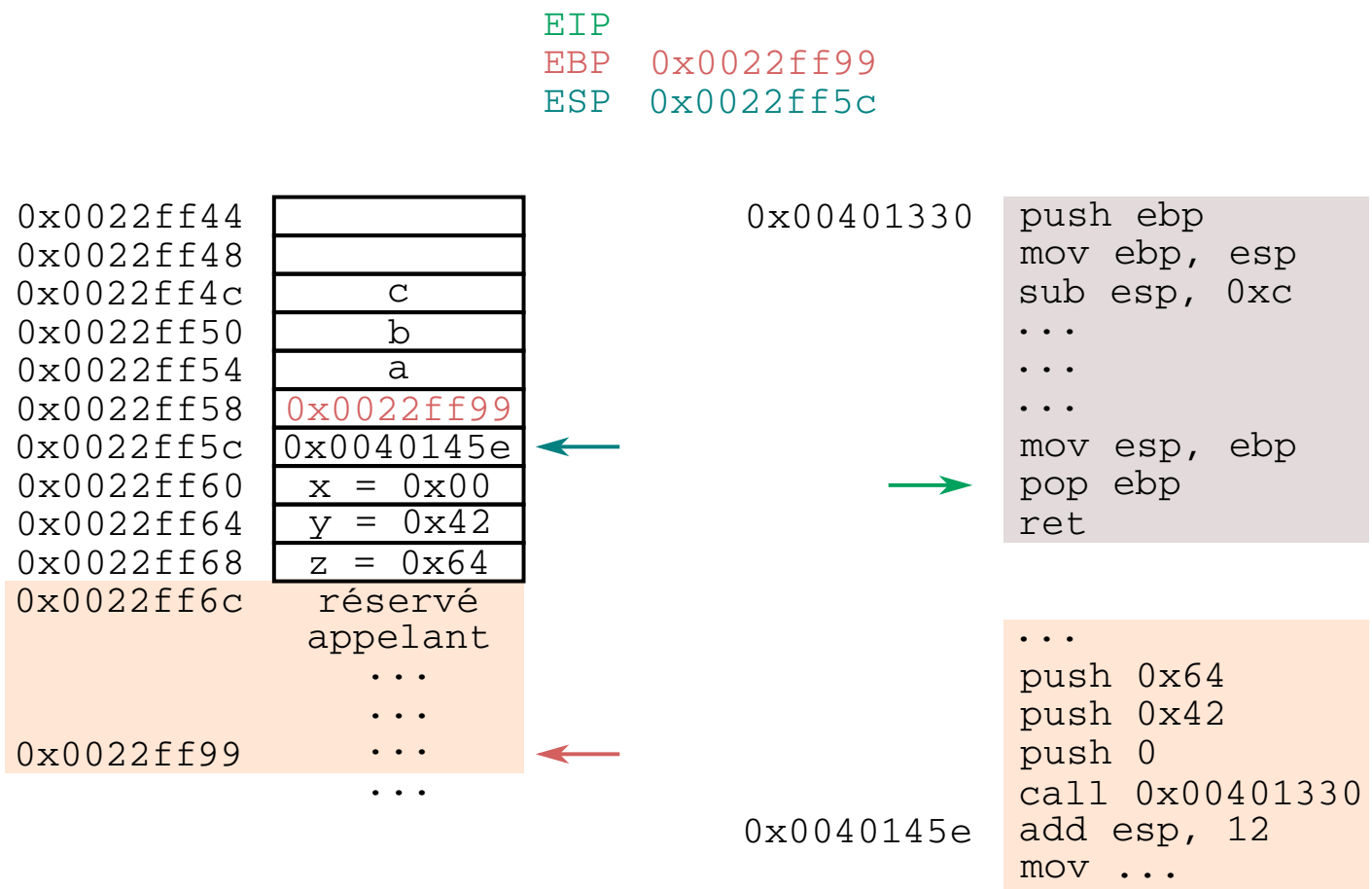
Appel de fonctions

convention CDECL



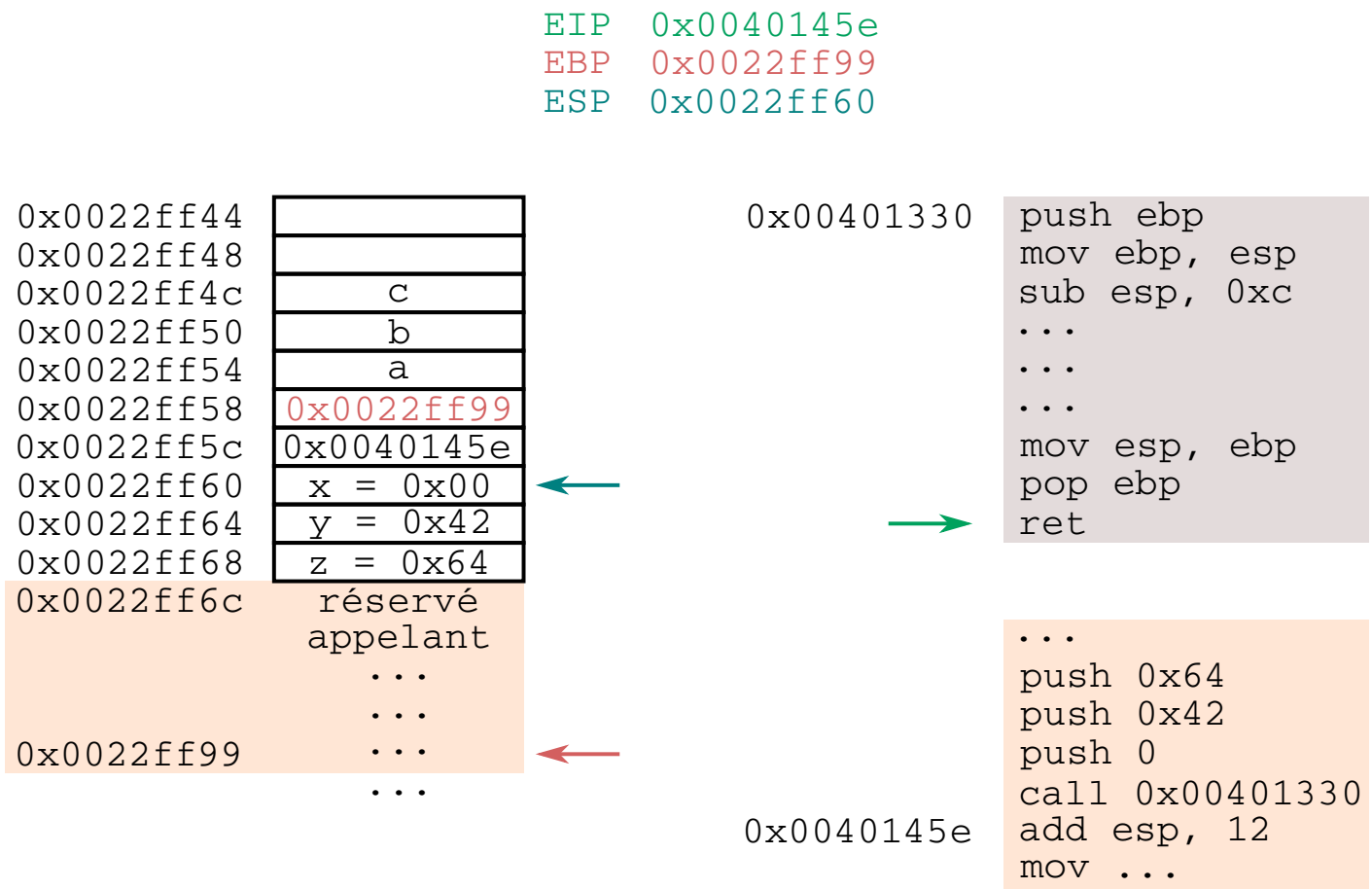
Appel de fonctions

convention CDECL



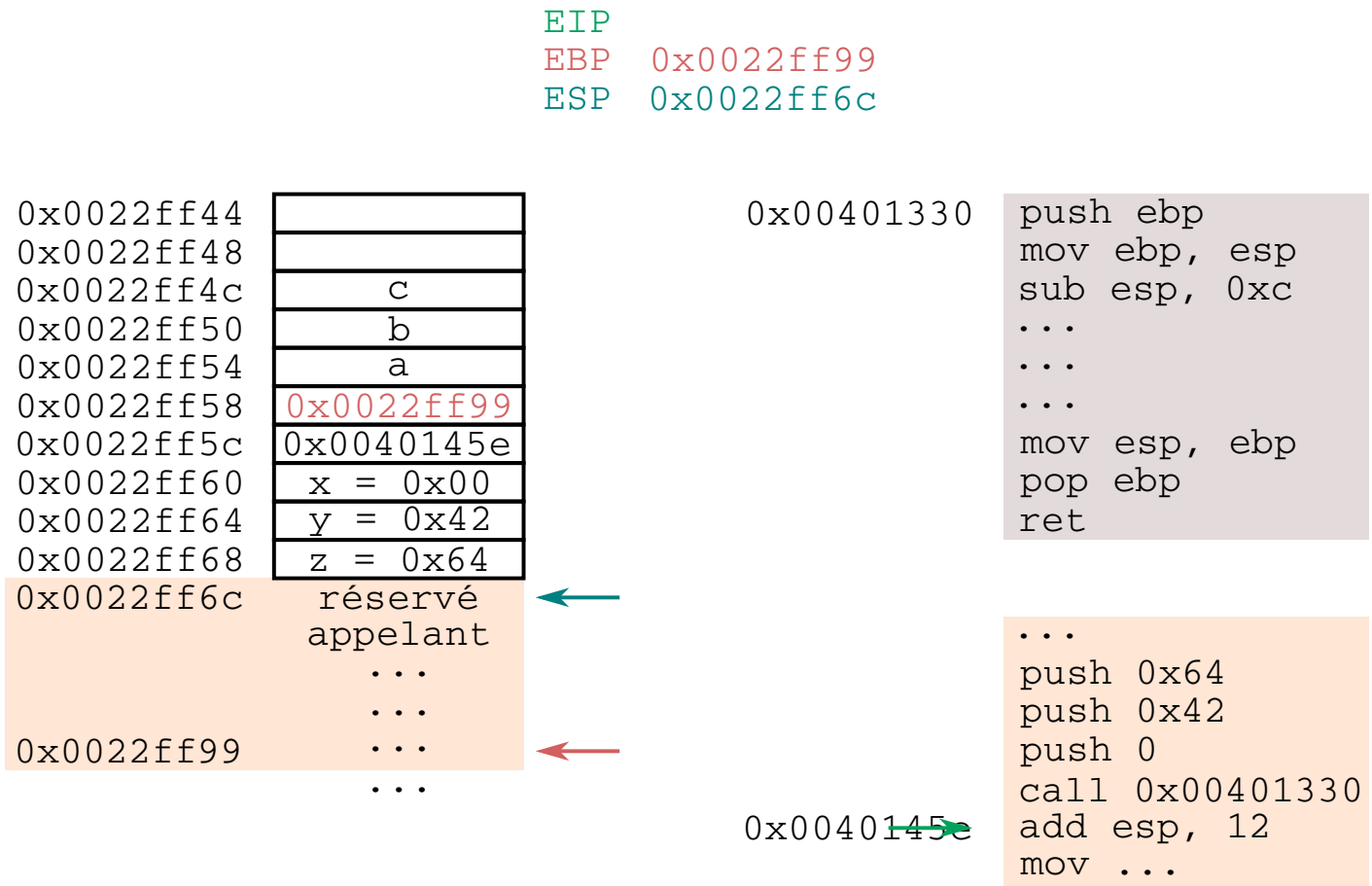
Appel de fonctions

convention CDECL



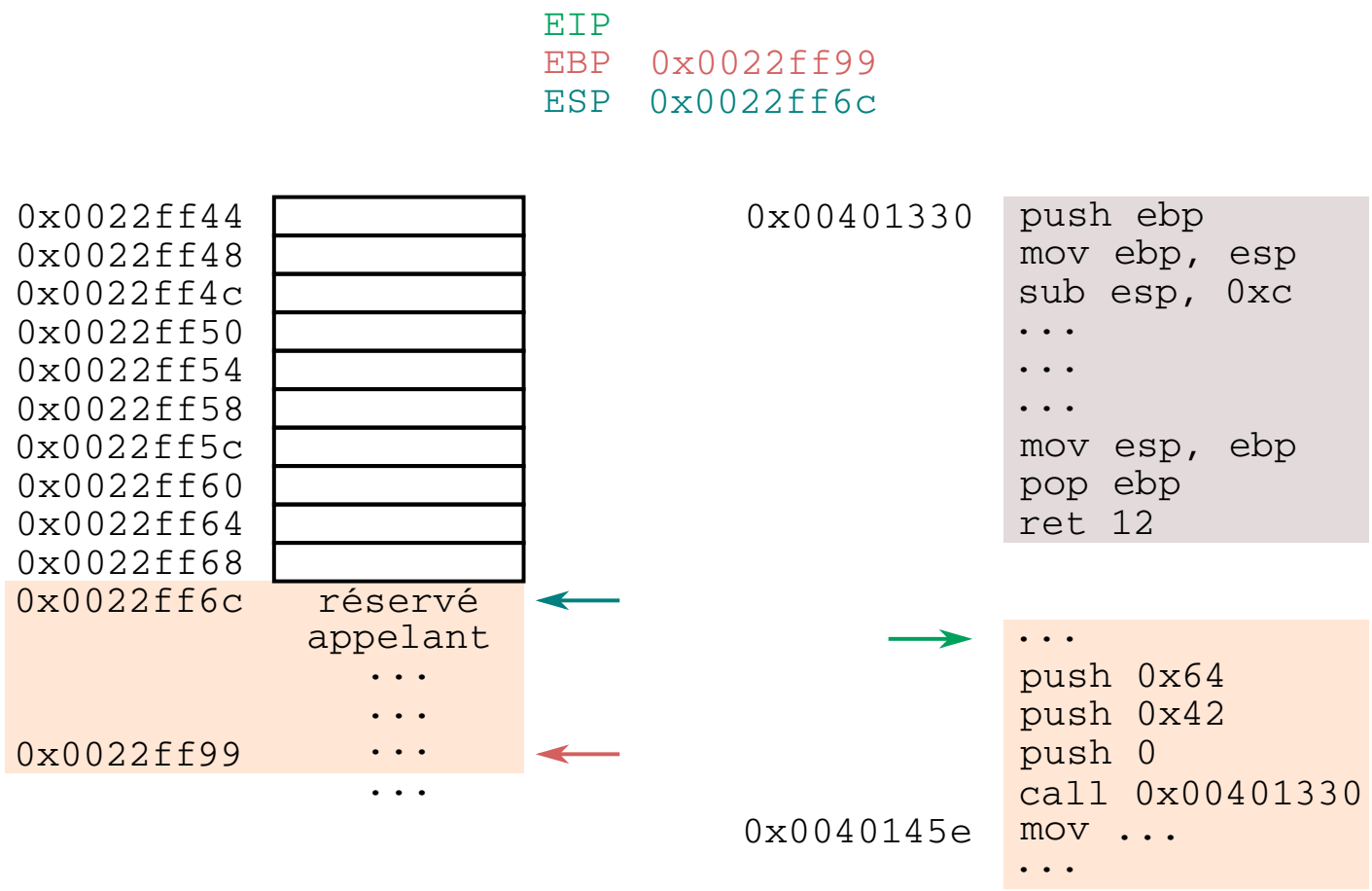
Appel de fonctions

convention CDECL



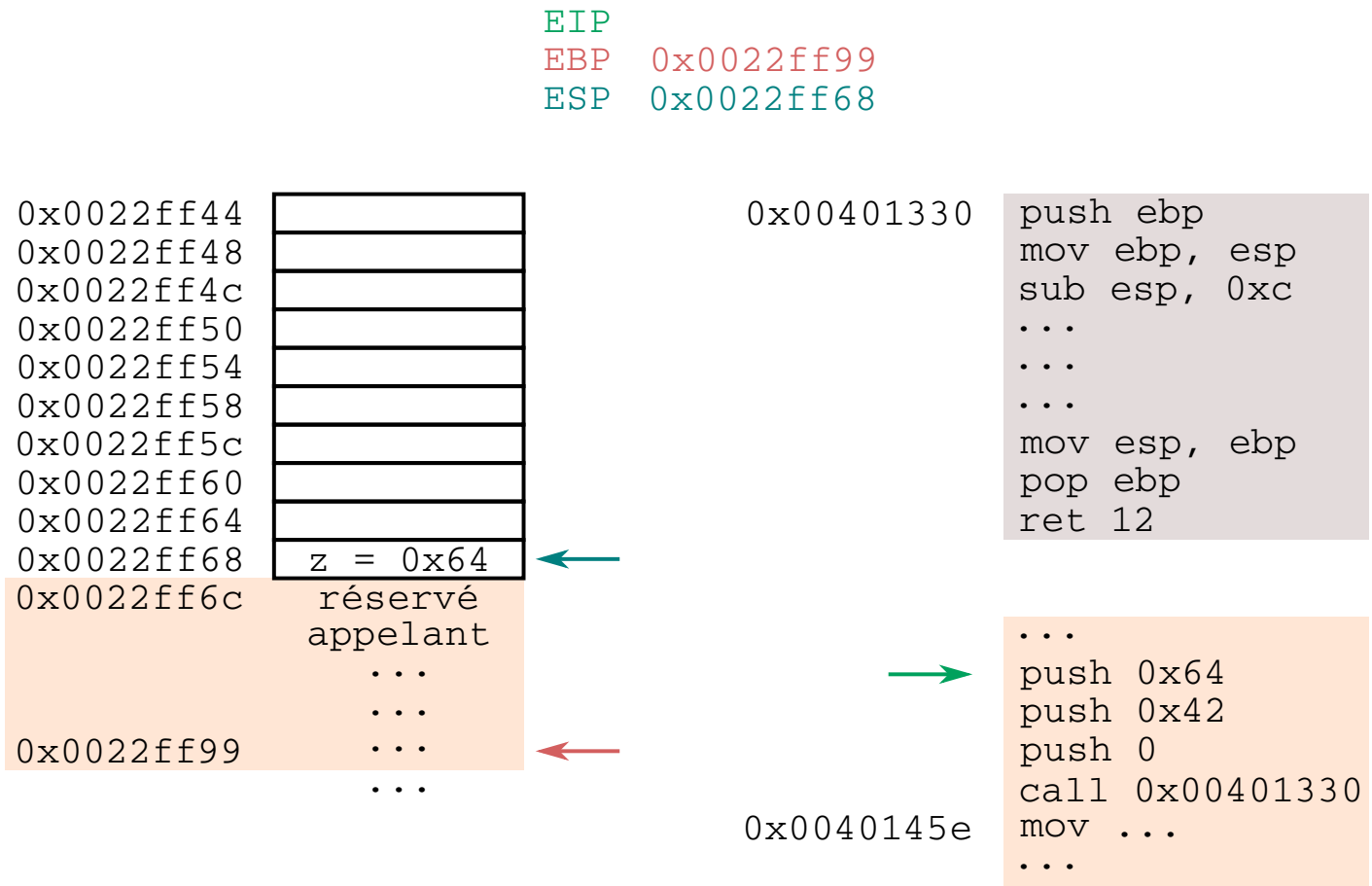
Appel de fonctions

convention STDCALL



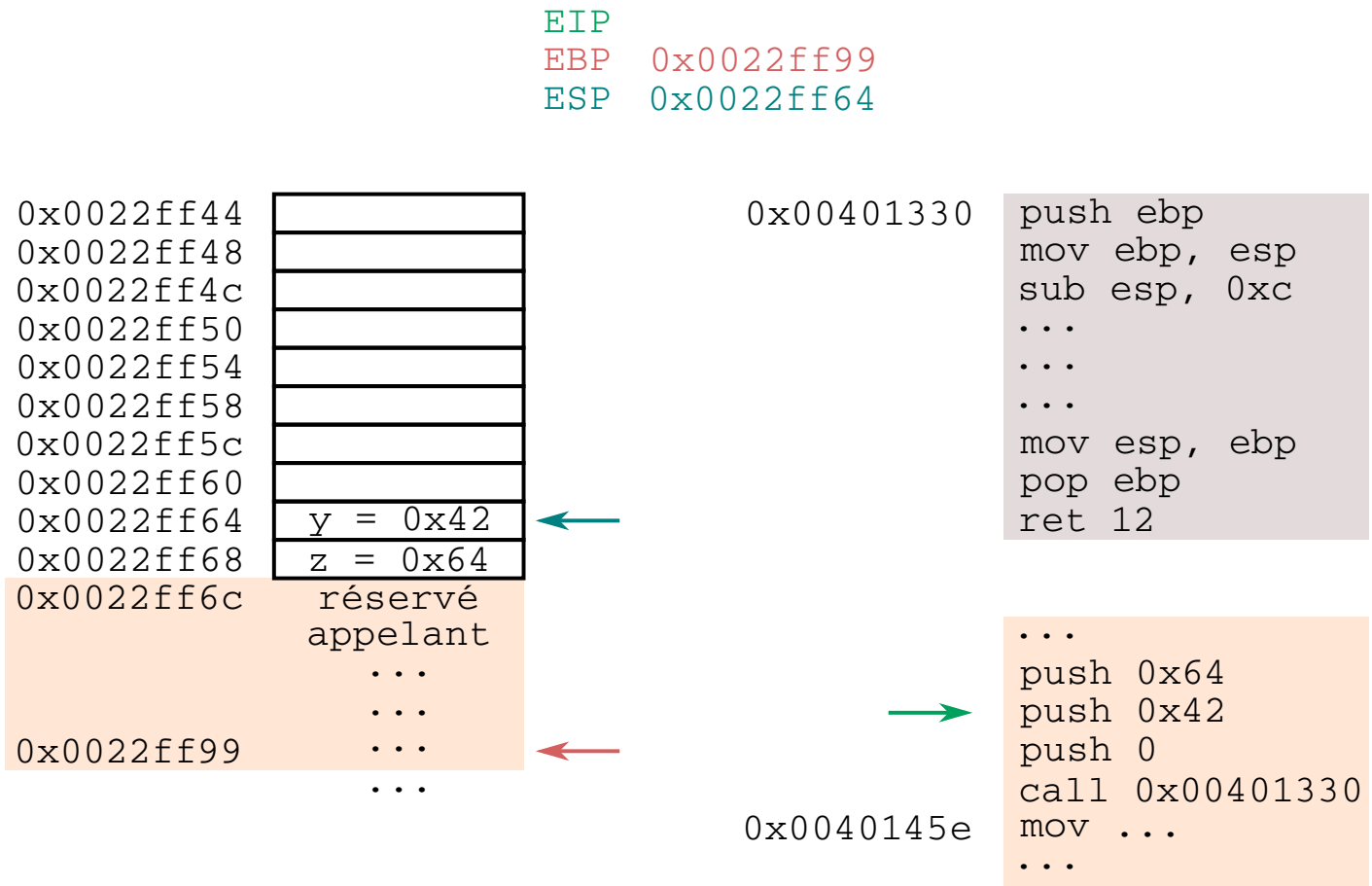
Appel de fonctions

convention STDCALL



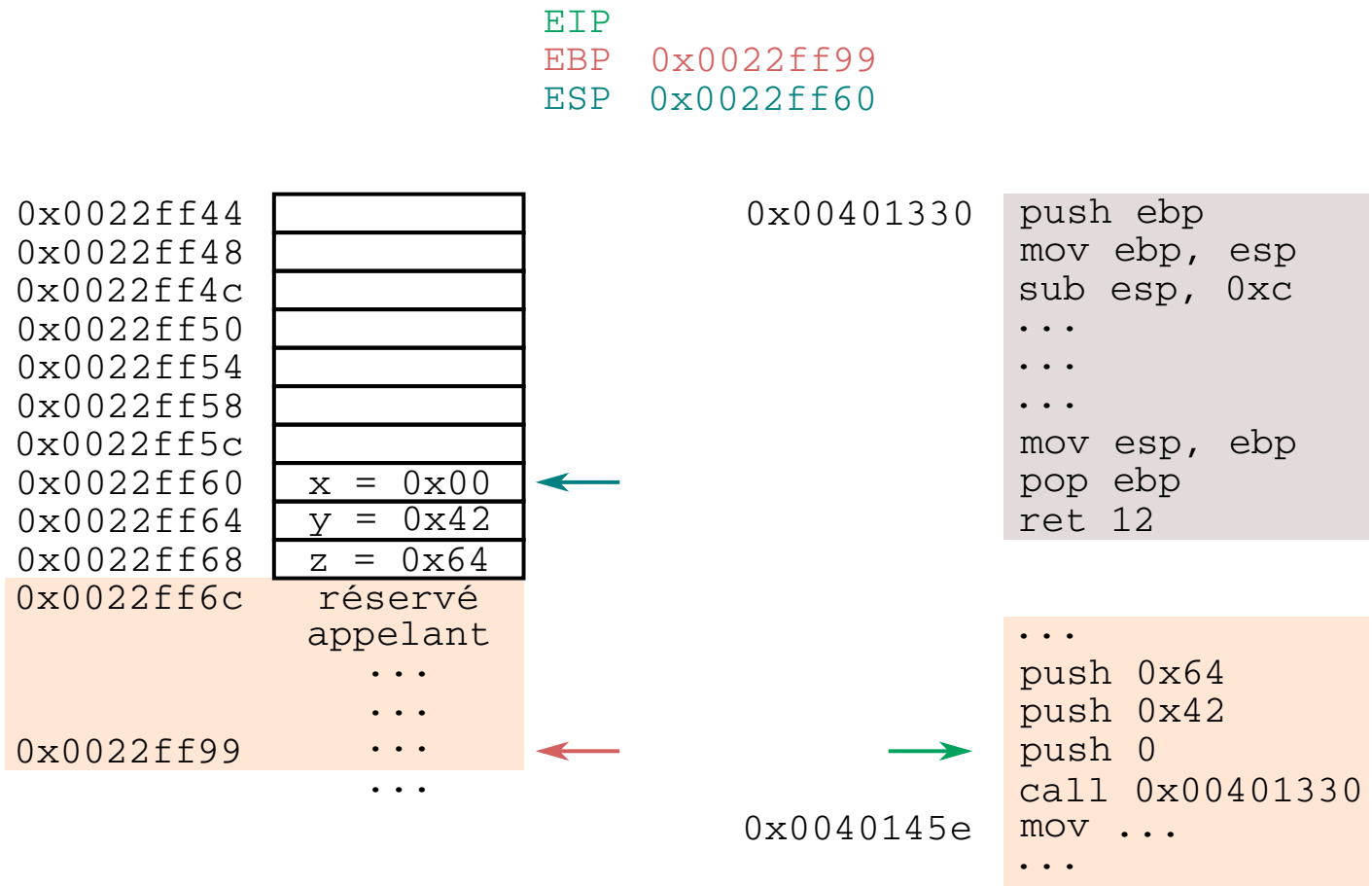
Appel de fonctions

convention STDCALL



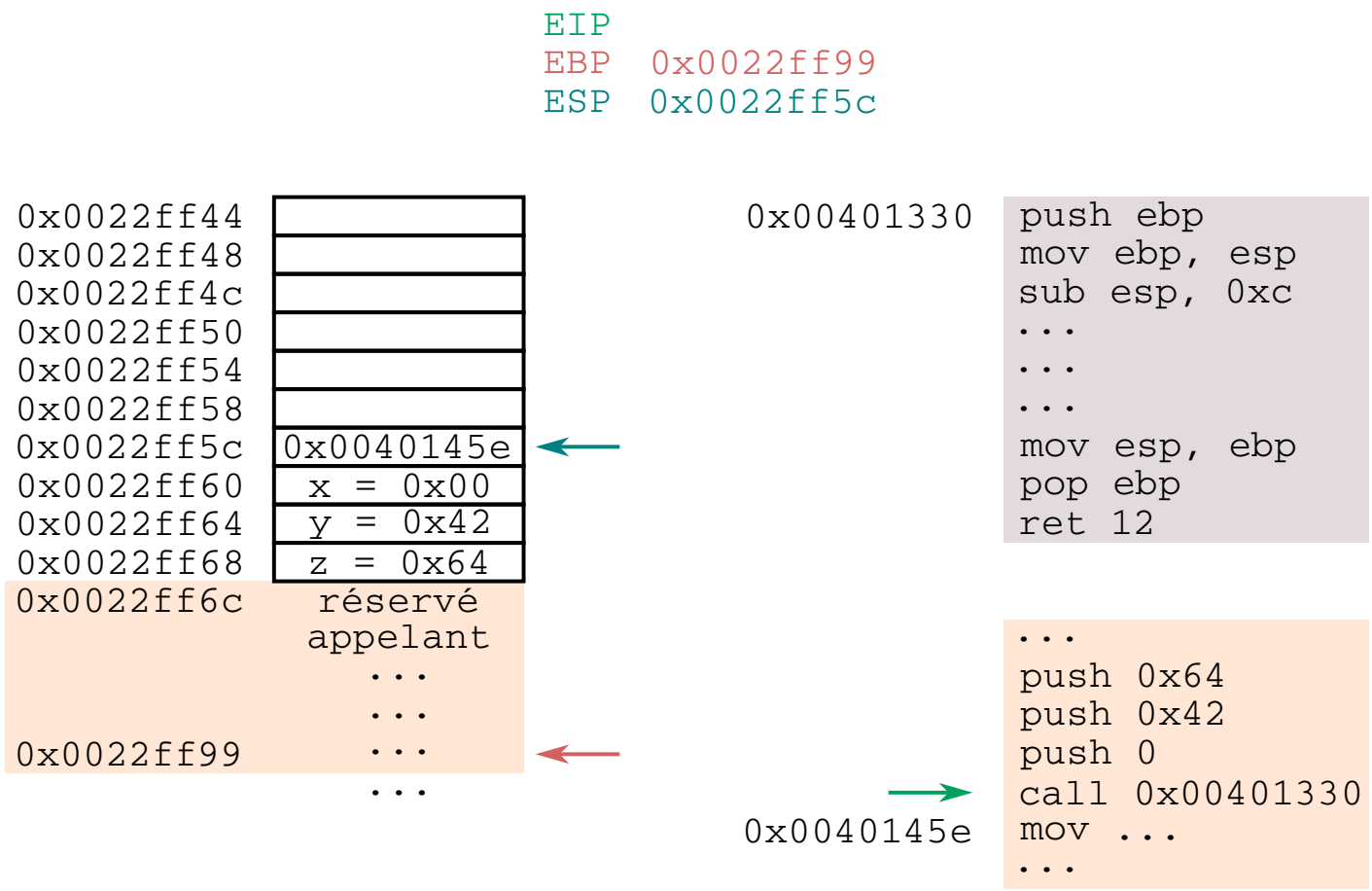
Appel de fonctions

convention STDCALL



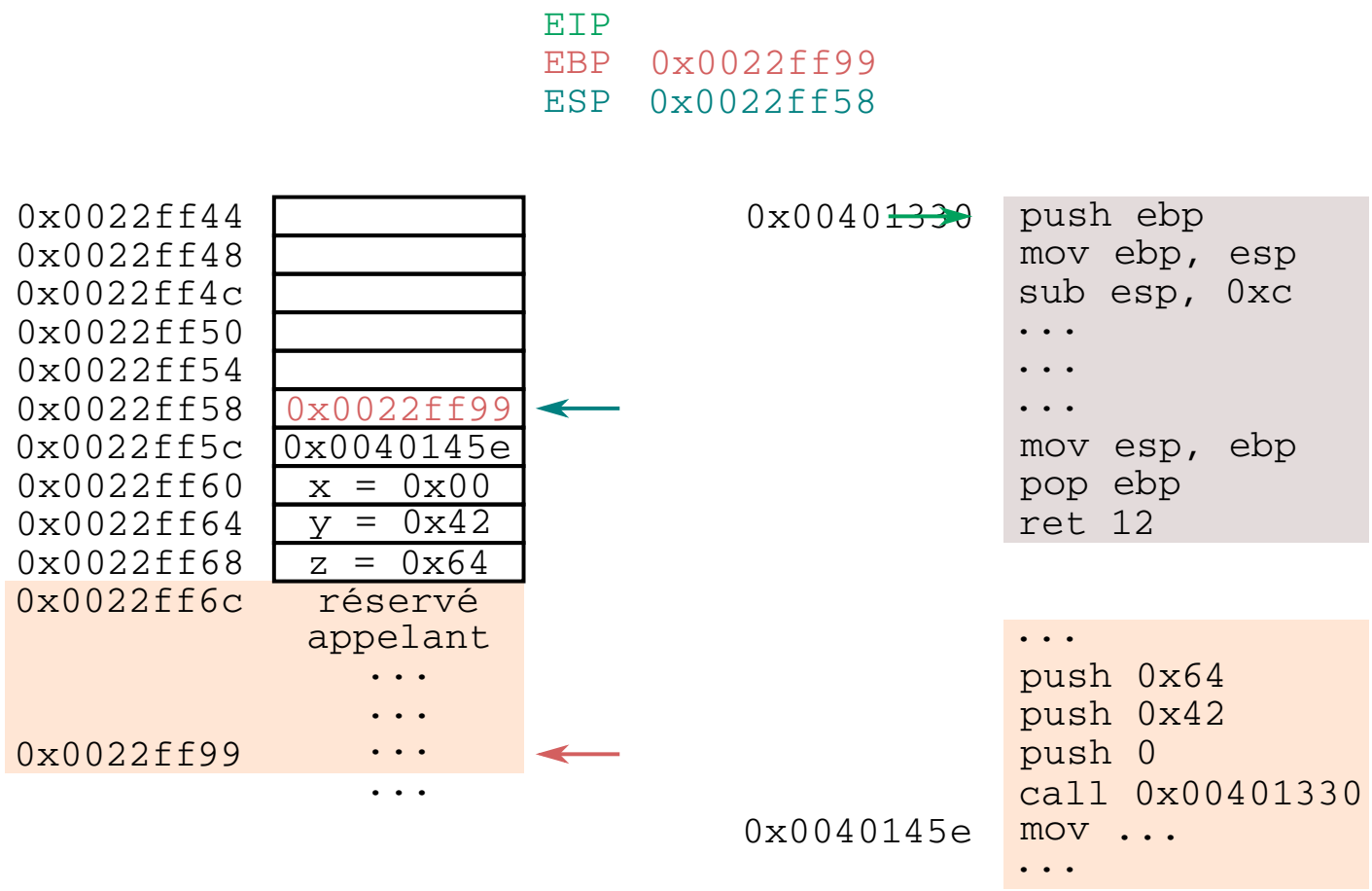
Appel de fonctions

convention STDCALL



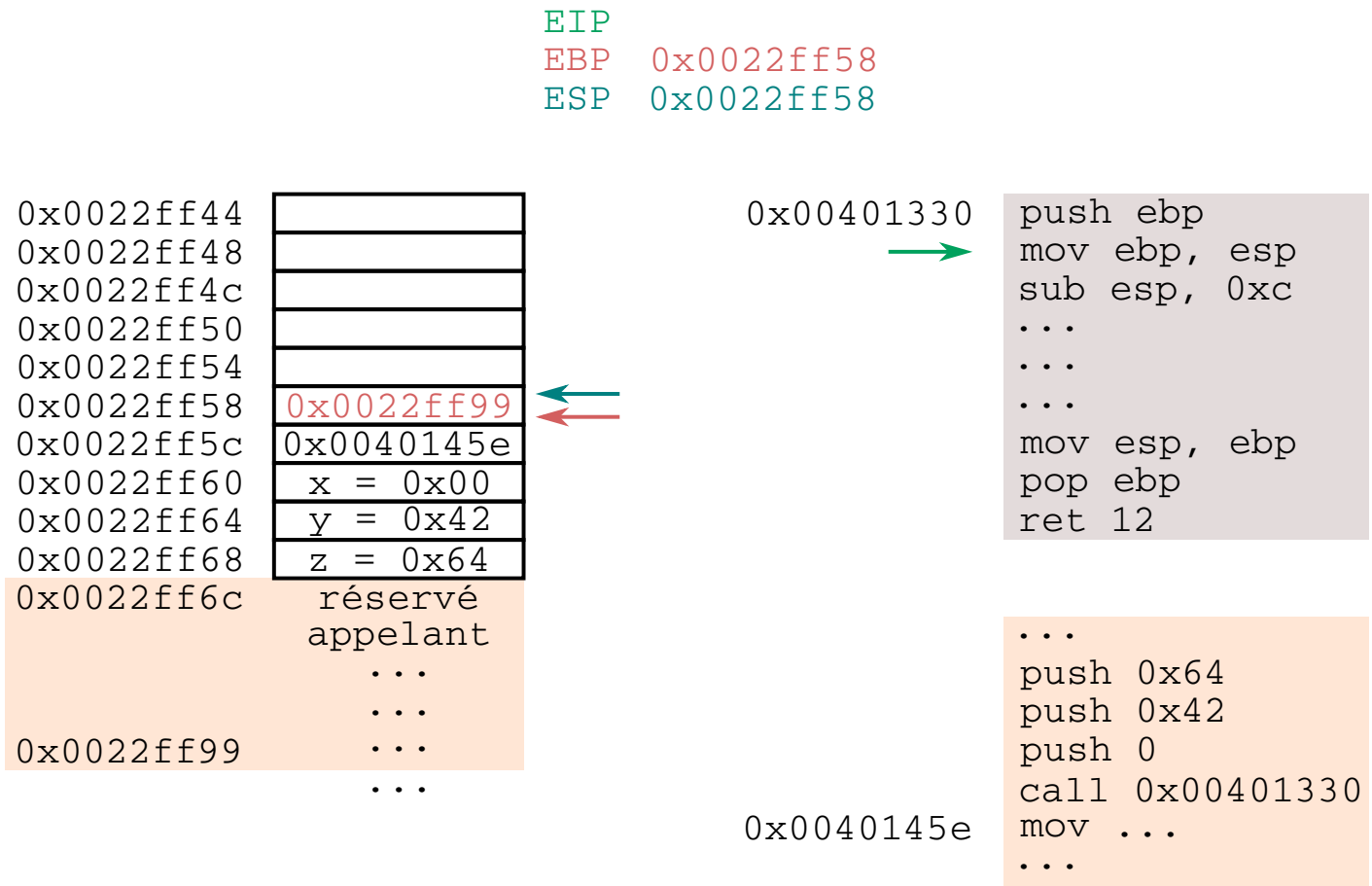
Appel de fonctions

convention STDCALL



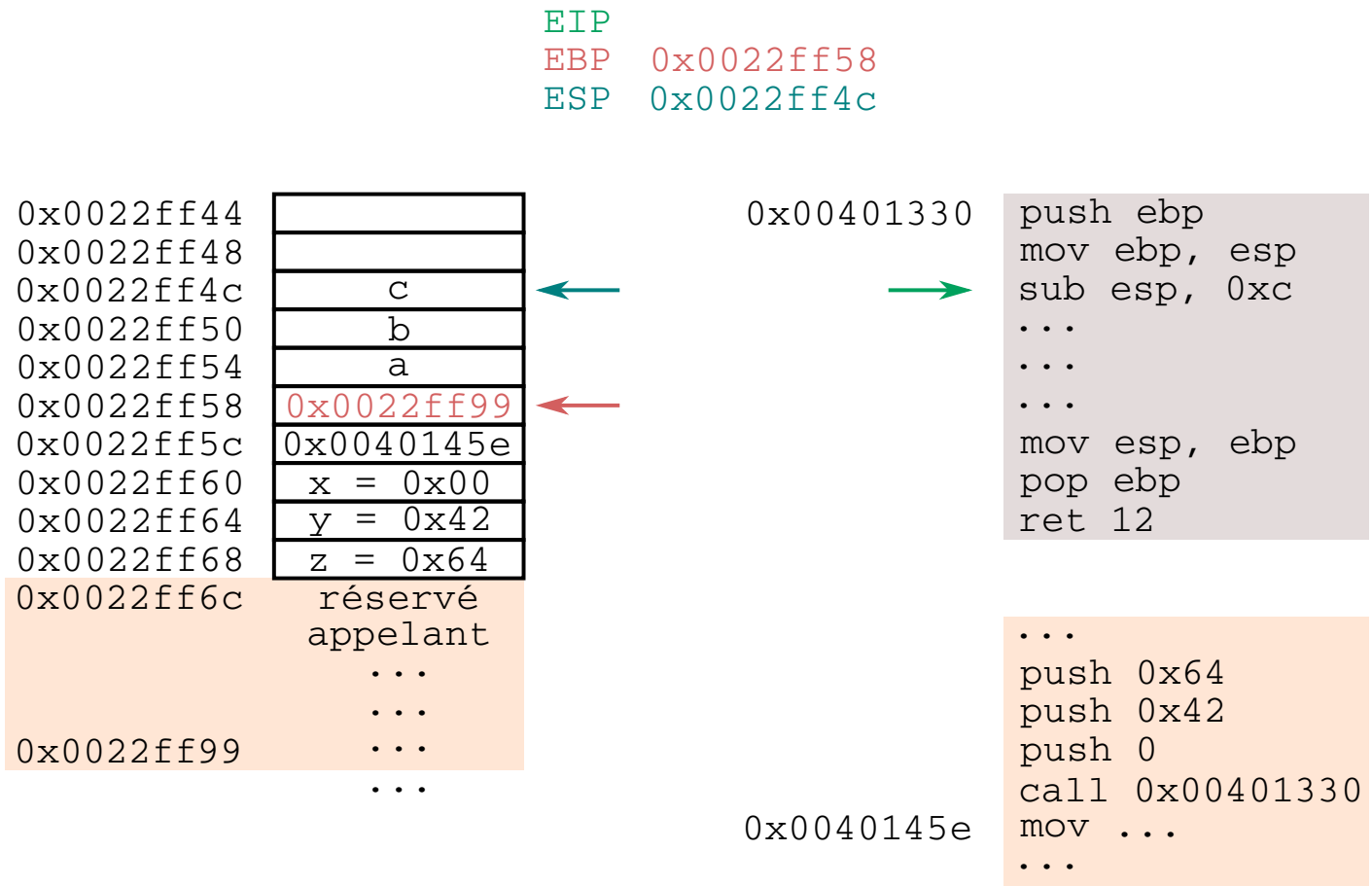
Appel de fonctions

convention STDCALL



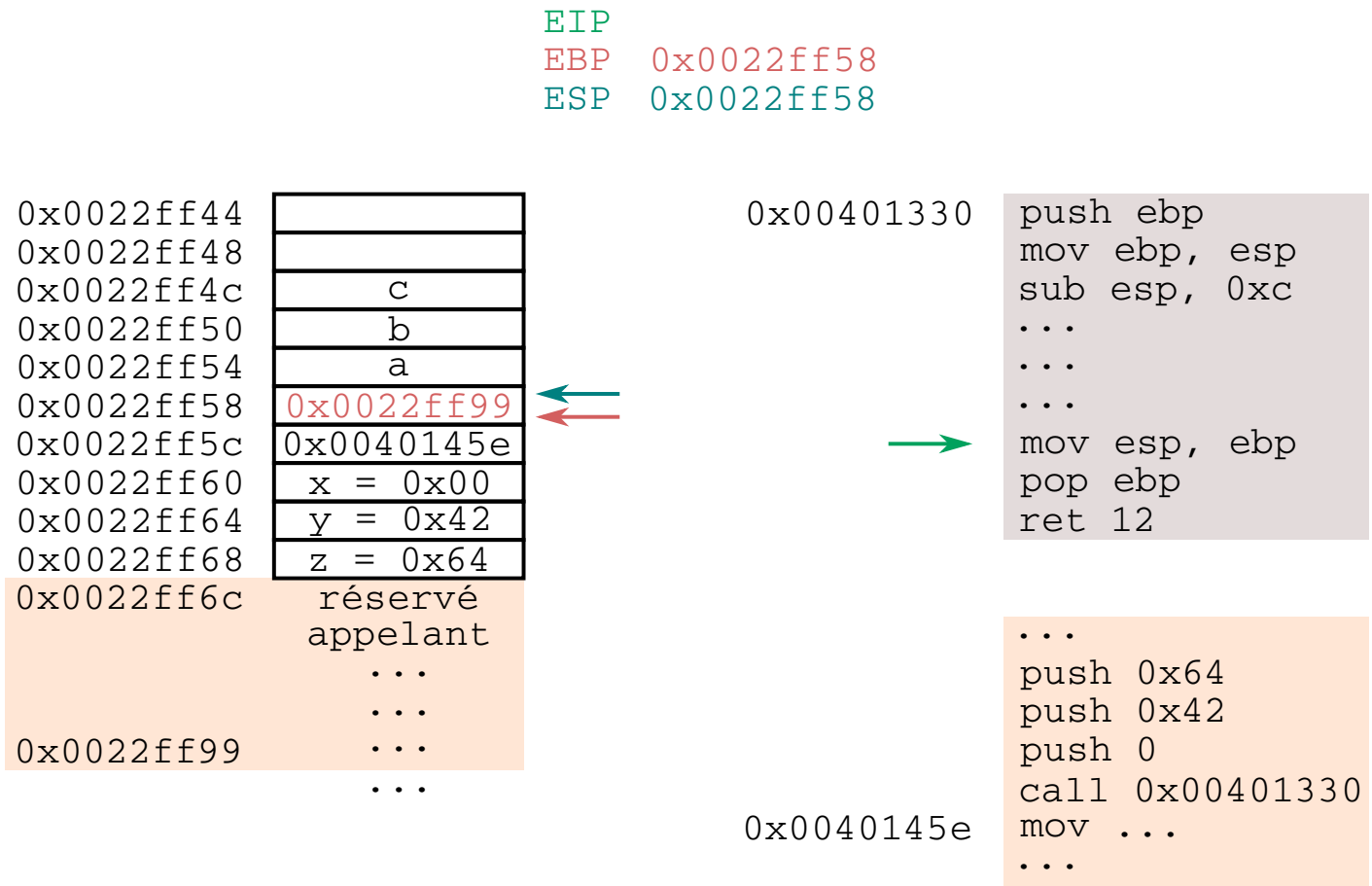
Appel de fonctions

convention STDCALL



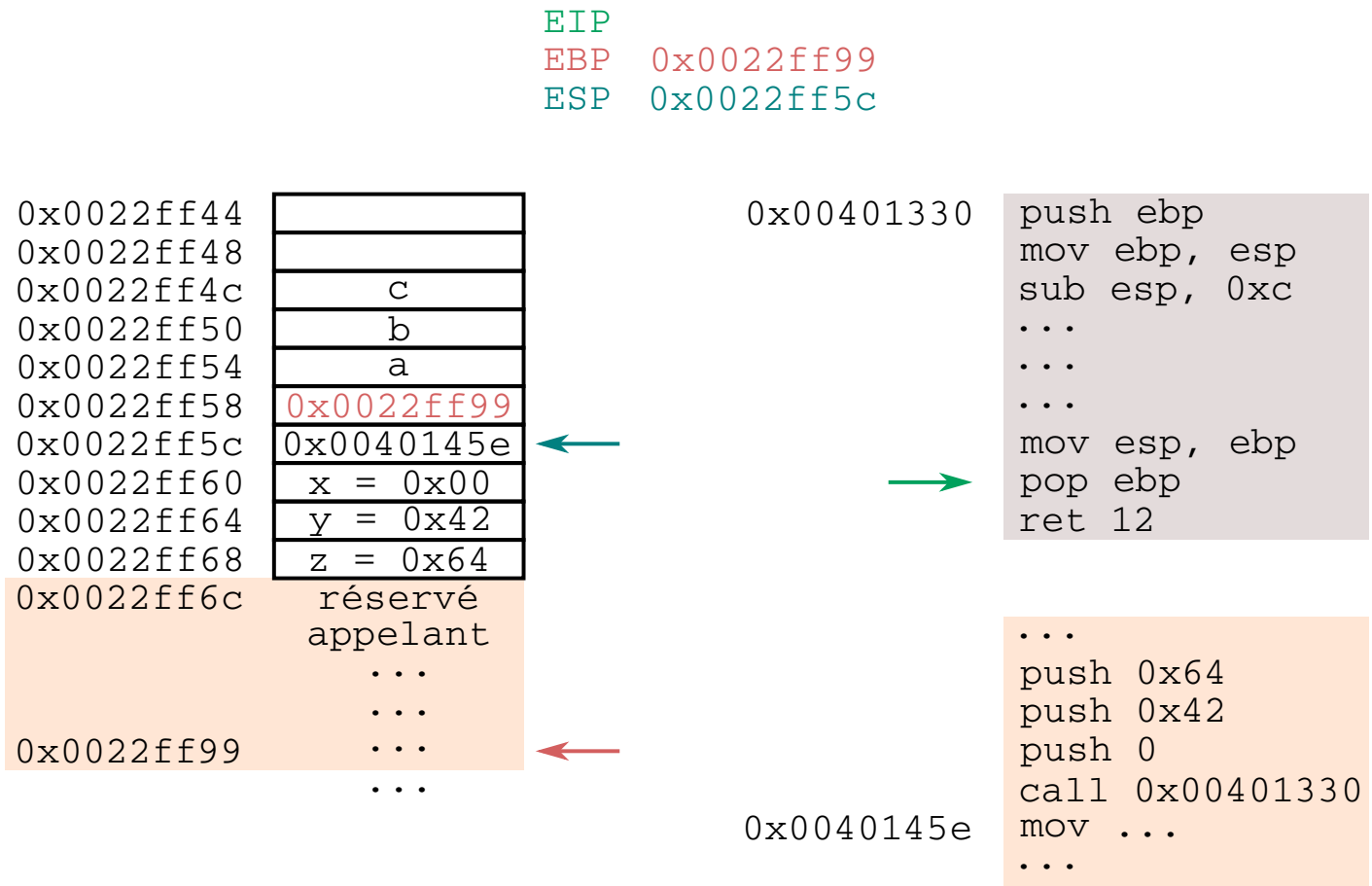
Appel de fonctions

convention STDCALL



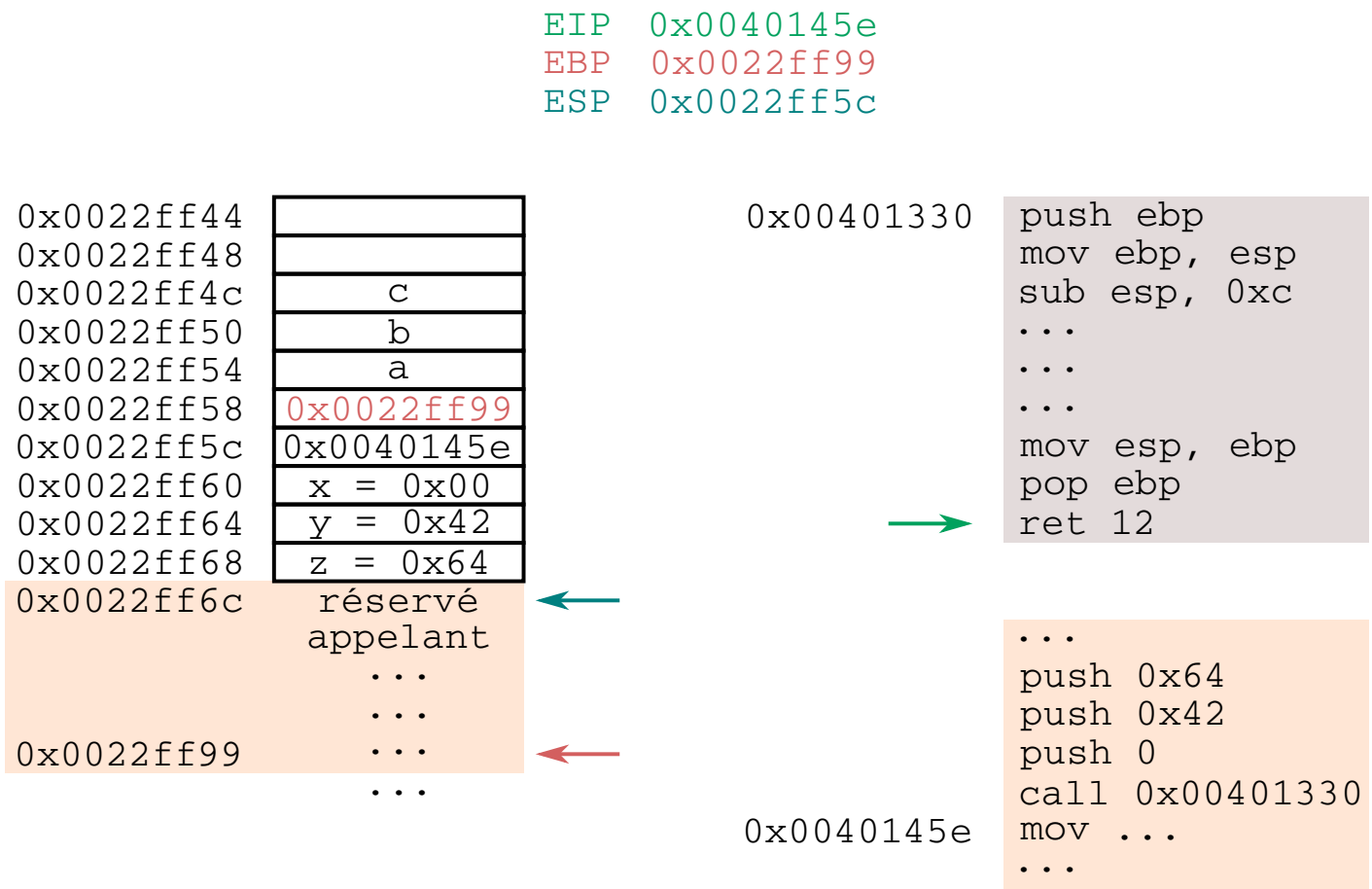
Appel de fonctions

convention STDCALL



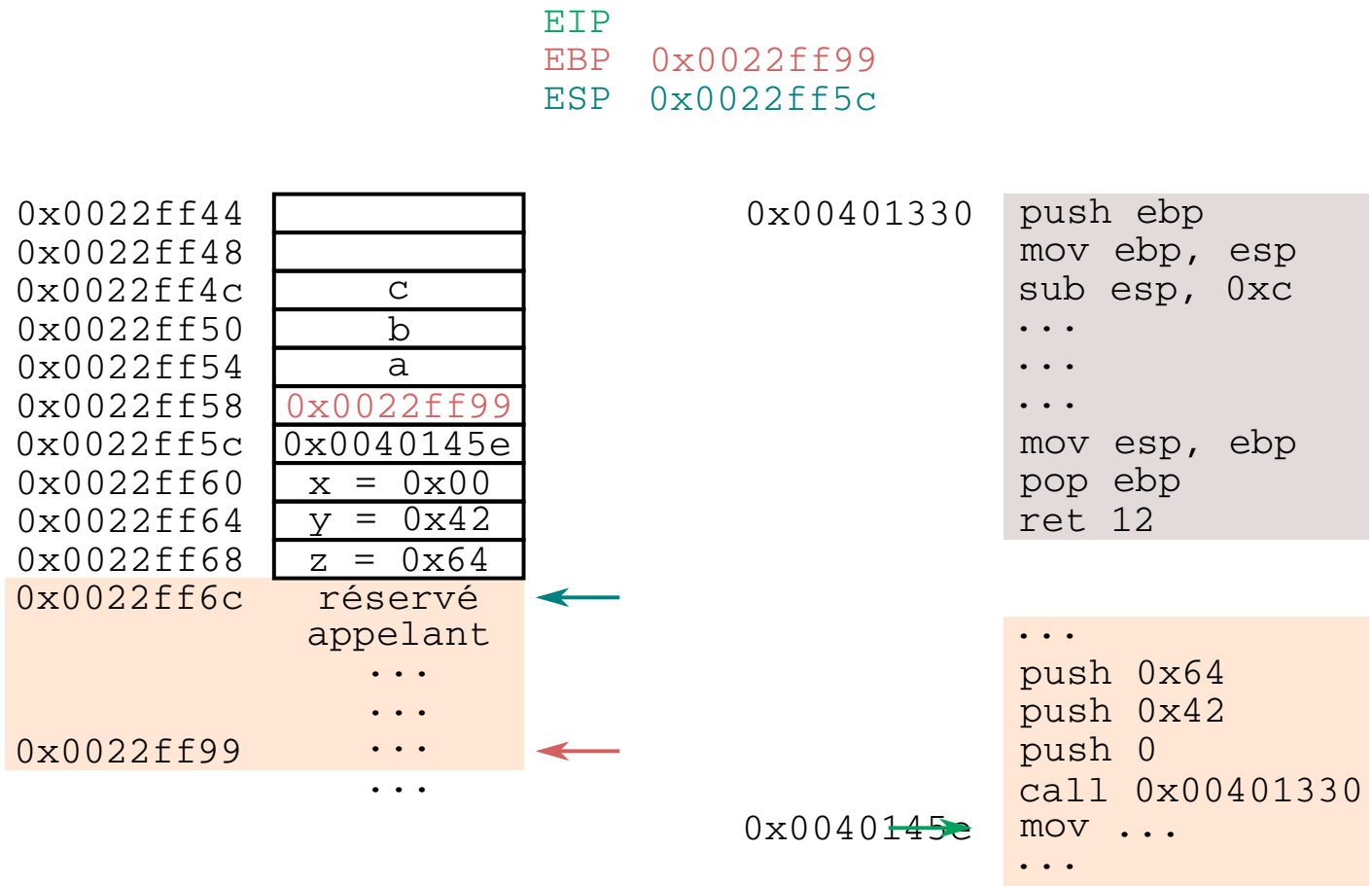
Appel de fonctions

convention STDCALL



Appel de fonctions

convention STDCALL



Appel de fonctions

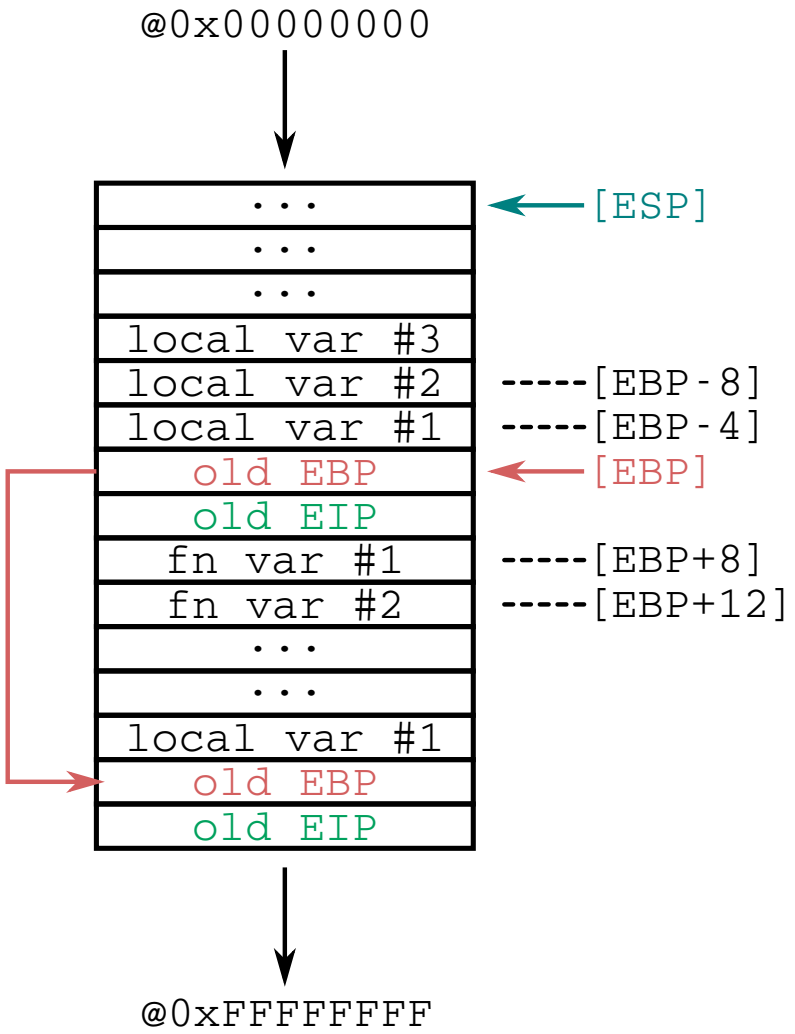


Figure – Etat de la pile

Structure générale d'un fichier exécutable

- le code du programme
- des données (chaines de caractères, structures, ...) initialisées ou non
- des données utilisées par le loader de l'OS pour créer le processus

Ces différents éléments sont regroupés dans le fichier par *sections*

Exemple de fichier assembleur x86

```
global start

extern printf
extern strlen

section .text

start:
    push ebp
    mov epb, esp
    ...
    ...
    ret
; end of start

section .data

text_usage:     db 'md5sum usage:',0
```

Mapping mémoire

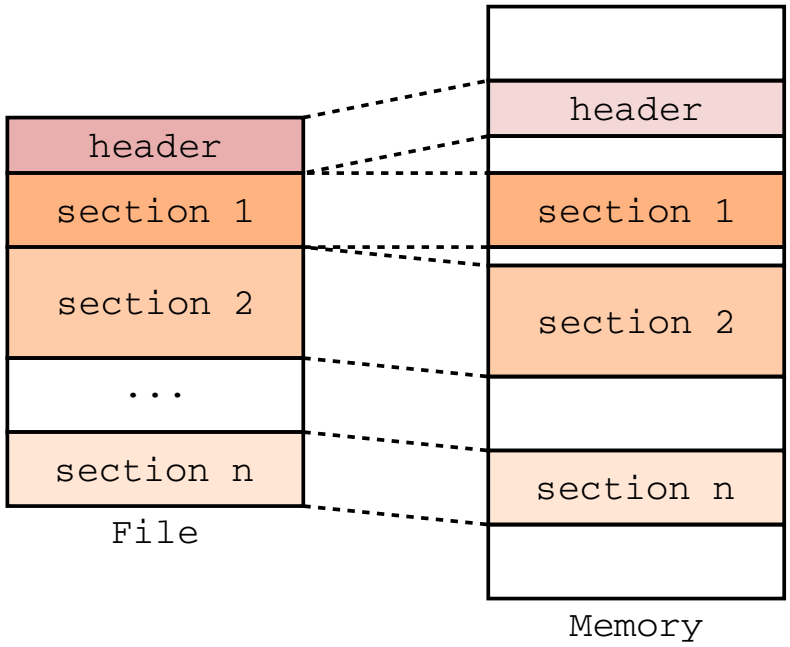


Figure – Mapping des sections en mémoire lorsd'un chargement d'un exécutable

Mapping mémoire

Sections

text/code	code executable
data	variables globales/statiques initialisées ($\neq 0$)
bss	variables globales/statiques non initialisées
...	

Memoire d'un processus

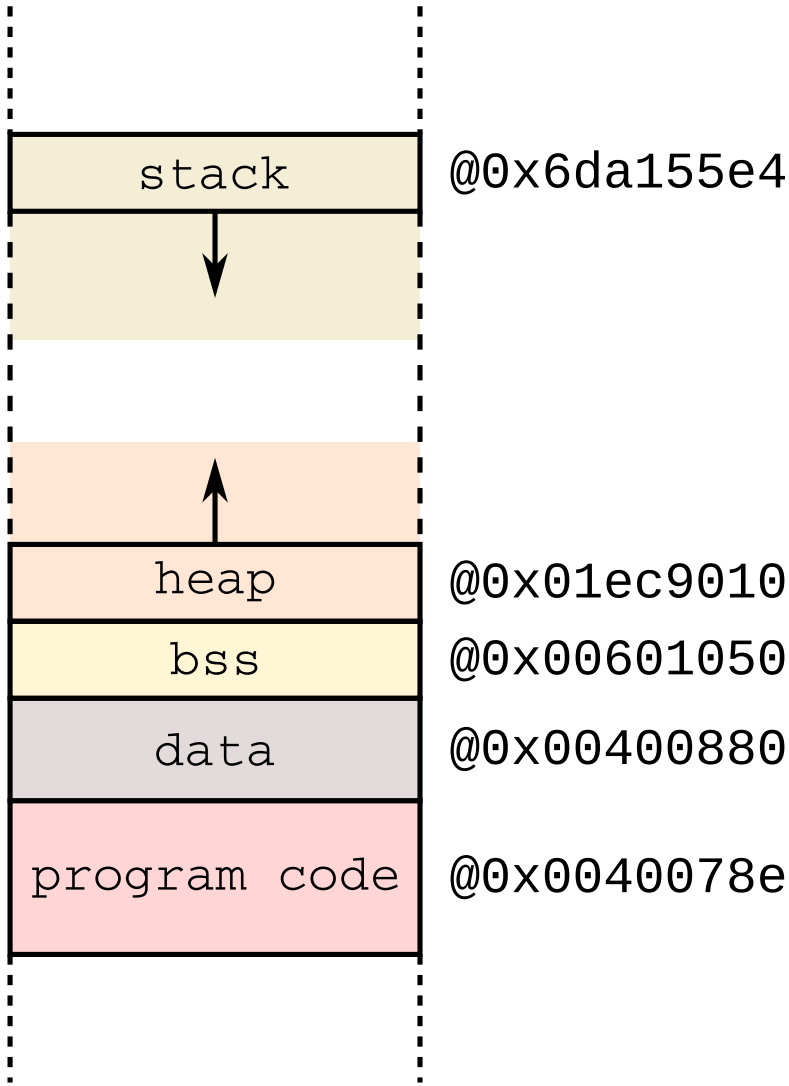


Figure – Mémoire d'un processus sous Linux x64

Memoire d'un processus

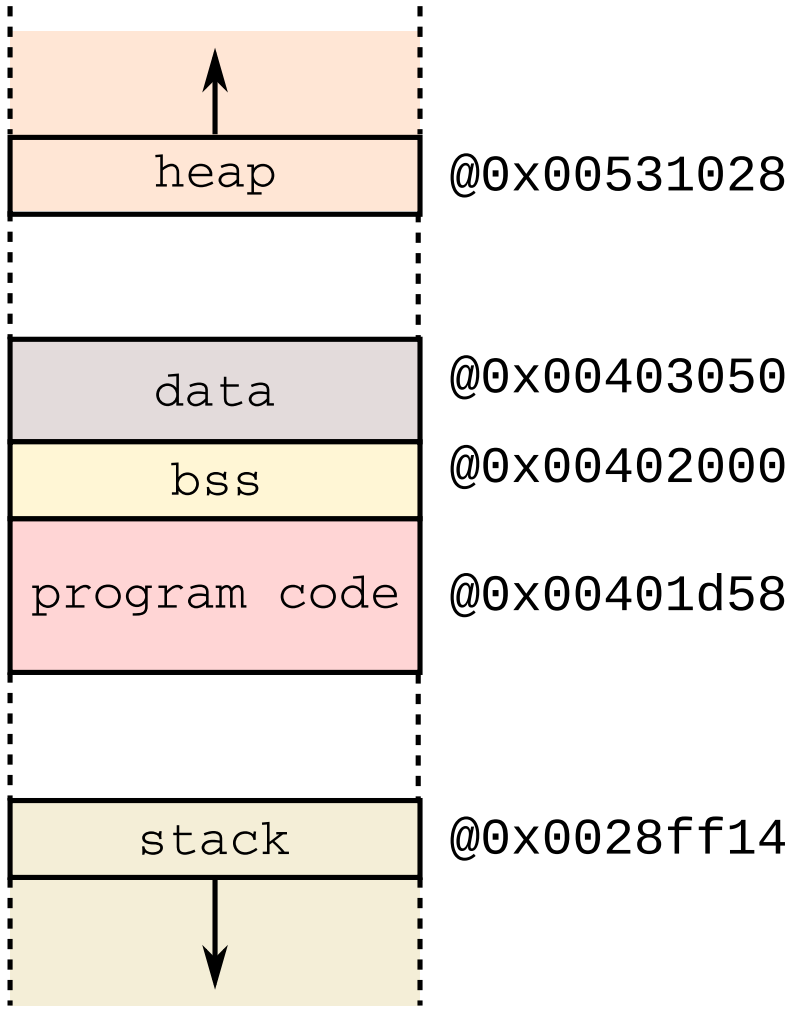


Figure – Mémoire d'un processus sous Windows 7sp1 x64