UNIVERSITÉ
CÔTE D'AZUR

Stockage, Qualificateurs et Régions Mémoire

Présentation: Stéphane Lavirotte
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
Stéphane Lavirotte, wikibooks

Mail: Stephane.Lavirotte@univ-cotedazur.fr
Web: http://stephane.lavirotte.com/
Université Côte d'Azur

UNIVERSITÉ CÔTE D'AZUR

Rappel: Variables C

- ✓ Classification simple:
  - **Variable globale**:
    - Déclarée en dehors de tout bloc d'instruction
  - **Variable locale**:
    - Déclarée à l'intérieur d'un bloc d'instruction
- ✓ Définitions:
  - *Visibilité* (portée): zone dans laquelle une variable est accessible
  - *Durée de vie:* temps pendant lequel une variable existe
  - *Emplacement:* endroit en mémoire où la variable est stockée
  - *Valeur initiale:* valeur de la variable à la première utilisation
- ✓ Les variables locales et globales ont des durées de vie, une visibilité et des emplacements mémoire différents

UNIVERSITÉ
CÔTE D'AZUR

Classes de stockage

- ✓ La classe de stockage détermine la durée de vie d'une variable
  - Permet de modifier la durée de vie d'une variable
  - Placé avant le type
- ✓ 4 mots clés en C pour définir la classe de stockage:
  - `auto`: pour les variables locales
  - `extern`: déclare une variable sans la définir
  - `register`: demande au processeur de tout faire pour utiliser un registre du processeur pour stocker cette variable
  - `static`: rend une définition de variable persistante

UNIVERSITÉ CÔTE D'AZUR

Classe de stockage `static`

- ✓ **Utilisé dans deux contextes:**
  - **Variable globale ou fonction:**
    - *Durée de vie:* toute la durée d'exécution du programme
    - *Visibilité:* limitée au seul fichier où la variable ou la fonction est déclarée
    - *Intérêt:* Eviter de polluer l'espace de noms
  - **Variable locale à un bloc:**
    - *Durée de vie:* La valeur de la variable est persistante entre les différents appel de la fonction
    - *Visibilité:* à partir de sa déclaration (déclaration (et initialisation) unique au cours du programme)
    - *Intérêt:* garantir une certaine encapsulation (évite de multiples variables globales)

UNIVERSITÉ CÔTE D'AZUR

Classe de stockage `extern`

- ✓ **Déclarer une variable *sans* la définir**
  - ***Durée de vie:*** **toute la durée du programme**
  - ***Visibilité:*** **globale à plusieurs fichiers**
  - ***Intérêt:*** **dans le cas de la compilation multi-fichiers**
    - **définit une variable ou une fonction dans un fichier (.h)**
    - **Permet l'utilisation dans d'autres fichiers**
- ✓ **Toute variable globale et fonction non déclarée (ou définie)** `static` **sont** `extern` **par défaut**

- ✓ **Synthèse:**
  - `static` **= variable et fonction « privées »**
  - `extern` **= variable ou fonction « publics », accessibles depuis d'autres fichiers**

UNIVERSITÉ CÔTE D'AZUR

Classe de stockage `register`

- ✓ Indiquer au compilateur de préférer stocker la variable dans un registre du processeur
  - *Durée de vie:* celle du bloc où la variable est définie
  - *Visibilité:* locale au bloc
  - *Intérêt:* gagner en performance par rapport au stockage de la variable dans un espace mémoire moins rapide (RAM)
    - Utile un contexte de logiciel embarqué
  - Limitation:
    - Le nombre de registre est limité et variable en fonction du processeur (ex: 4 registres pour le stockage sur x86)
  - Utilisation déconseillée sauf cas particulier (le compilateur optimise ce cas souvent mieux que le programmeur)

```
register short i;
for (i = 0; i < 100; i++) {
…
}
```

UNIVERSITÉ CÔTE D'AZUR

Classe de stockage `auto`

- ✓ Définir des variables locales non-statiques dans une fonction
  - *Durée de vie*: celle du bloc où la variable est définie
  - *Visibilité:* locale au bloc
  - *Intérêt:* aucun car comportement par défaut de C
- ✓ Donc inutile en C
  - Ce mot clé est un héritage du langage B dont est issu C

UNIVERSITÉ CÔTE D'AZUR

Synthèse des Classes de Stockage

Classe de stockage	Durée de vie	Visibilité (portée)	Lieu de stockage	Valeur initiale
`auto`	Bloc	Bloc	Pile	Non définie
`extern`	Programme	Global à plusieurs fichiers	Segment de données	Zéro
`static`	Programme	Bloc ou fichier	Segment de données	Zéro
`register`	Bloc	Bloc	Registre CPU	Non définie

UNIVERSITÉ CÔTE D'AZUR

Qualificateurs de type

- ✓ C définit 3 qualificateurs influant sur une variable
  - `const`: la **valeur** ne change pas
  - `restrict`: optimisation pour la gestion des pointeurs
  - `volatile`: variable pouvant être modifiée par une source externe au programme lui-même

- ✓ Une variable (ou paramètre de fonction) peut avoir jusqu'à trois qualificateurs
  - Même si certaines combinaisons ont peu de sens

UNIVERSITÉ CÔTE D'AZUR

Qualificateur const 1/2

- ✓ **Attention: ne déclare pas une vraie constante**
  - **Indique au compilateur que la valeur est non modifiable**
  - **Impératif d'assigner une valeur à la déclaration**
  - **Toute tentative ultérieure de modification entraîne une erreur du compilateur**

```
const int i = 0;
i = 1; /* erreur */
```

- ✓ **Utilisation avec les pointeurs**
  - **Indique que la valeur pointée ne peut être modifiée**

```
void fonction(const char * ptr) {
    ptr[0] = 0;   /* erreur */
    ptr++;        /* ok */
}
```

UNIVERSITÉ CÔTE D'AZUR

Qualificateur const 2/2

- ✓ **Attention aux subtilités (placement du `const`)**
  - **Indique que la valeur elle-même du pointeur est constante**

```
char * const ptr = "Salut tout le monde !";
ptr = "Hello world !"; /* erreur*/
```

- ✓ **Encore plus subtile:**
  - **On peut mélanger les deux !**

```
const char * const ptr = "Salut tout le monde !";
ptr = "Hello world !"; /* erreur*/
ptr[0] = 0;            /* erreur*/
```

- ✓ **Pour autant `const` n'est pas une protection réelle**

```
const char lettre = 'A';
memset(& lettre, 'B', 1);
putchar(lettre);       /* affiche B */
```

UNIVERSITÉ CÔTE D'AZUR

Qualificateur `volatile`

- ✓ Spécifier que la variable peut être modifiée à l'insu du compilateur
  - Annule toute optimisation du compilateur
  - Oblige le compilateur à procéder à chaque lecture écriture de la mémoire où est stockée la variable
- ✓ Exemples d'utilisation
  - Les coordonnées du pointeur de souris (modifié par un autre programme)
  - La gestion des signaux (voir cours ProgSys)
  - La communication entre différents fils d'exécution (voir thread)
  - Des registres matériel accédés depuis un prog C (horloge interne, par exemple)
- ✓ Variable que le prog ne peut modifier, mais dont la valeur change quand même (de manière externe au programme)

```
extern const volatile int horloge_temps_reel;
```

UNIVERSITÉ CÔTE D'AZUR

Qualificateur `restrict`

- ✓ Uniquement pour les déclarations de pointeur
- ✓ Responsabilité du programmeur:
  - Certifie que le pointeur est le seul à pointer sur une zone mémoire
  - S'il « ment » => problèmes à l'exécution
- ✓ Permet au compilateur des optimisations impossibles sans ce mot clé
- ✓ Exemple:

```
int* restrict zone;
```

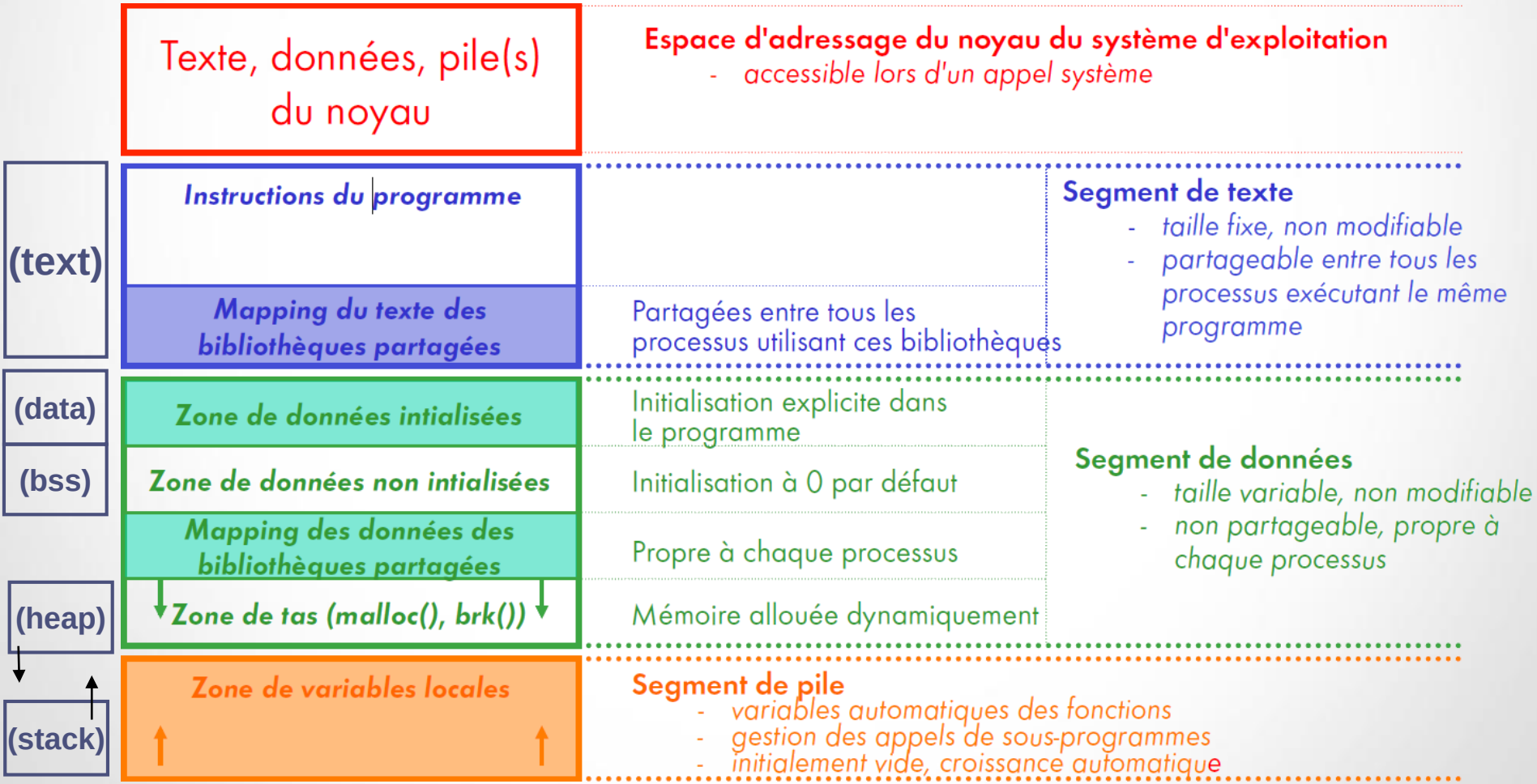
UNIVERSITÉ
CÔTE D'AZUR

Mais où sont rangées ces informations ?

- ✓ Segment mémoire
  - Espace d'adressage indépendant défini par 2 valeurs
    - Adresse où commence le segment (base ou adresse de base)
    - Taille ou décalage (limite ou offset)
  - Place d'adresses continue
- ✓ Type de segment
  - Segment de text: Code Segment (registre CS sur x86)
  - Segment de données: Data Segment (registre DS sur x86)
    - initialisé (data segment)
    - non initialisé (bss)
    - Alloué dynamiquement (heap)
  - Segment de pile: Stack Segment (registre SS sur x86)

UNIVERSITÉ CÔTE D'AZUR

Espace d'Adressage Exemple d'Unix



UNIVERSITÉ CÔTE D'AZUR

Segments

- ✓ Segment de code
  - Contient le code généré par le compilateur
  - Informations pour interpréter sémantiquement les données
- ✓ Segments de données
  - Contient l'espace pour les variables initialisées et non initialisées définies dans votre programme
  - Contient la cartographie des données de bibliothèques partagées
  - Contient la mémoire allouée dynamiquement (`malloc`)
- ✓ Segment de pile
  - Gestion de la chaîne dynamique des appels de fonctions
  - Gestion des variables locales des fonctions
  - Gestion des exceptions

- ✓ Exemple de l'exécution d'un programme

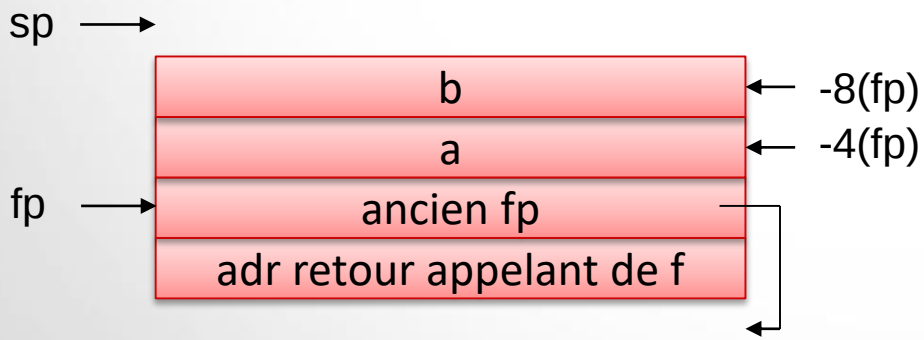
UNIVERSITÉ CÔTE D'AZUR

Exemple de Gestion de la Pile: Situation initiale 1/4

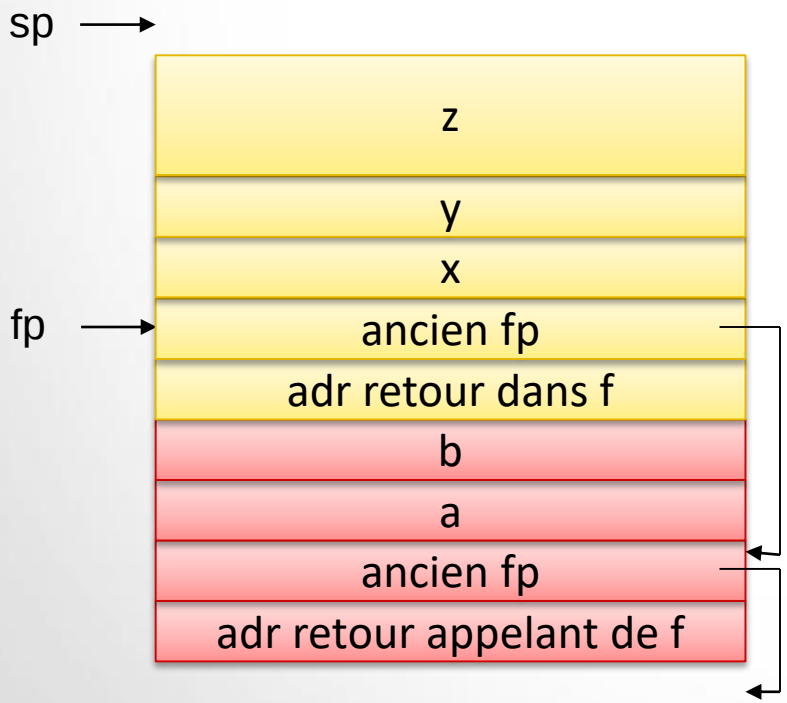
```
void f() {
    int a, b;
    g(a, b);
}
void g(int x, int y)
{
    double z;
    h(z);
}
void h(double a) {
    …
}
```

Exemple de Gestion de la Pile Appel de la fonction f 2/4

```
void f() {
    int a, b;
    g(a, b);
}
void g(int x, int y)
{
    double z;
    h(z);
}
void h(double a) {
    …
}
```

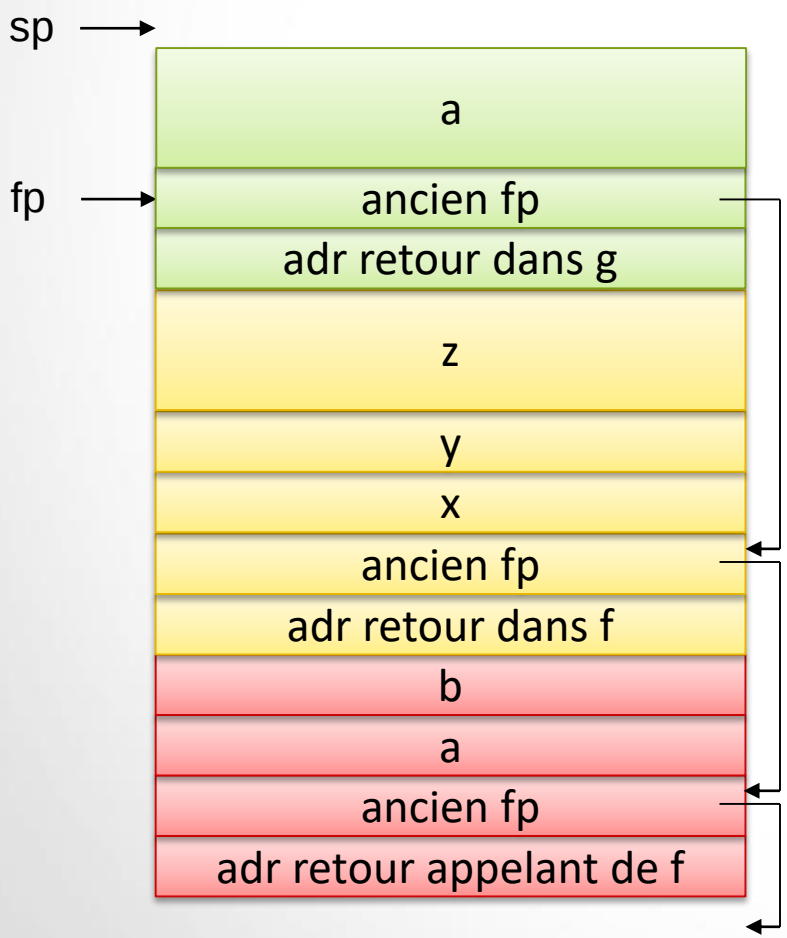


Exemple de Gestion de la Pile
Appel de la fonction g 3/4




```
void f() {
    int a, b;
    g(a, b);
}
void g(int x, int y)
{
    double z;
    h(z);
}
void h(double a) {
    …
}
```

Exemple de Gestion de la Pile
Appel de la fonction h 4/4




```
void f() {
    int a, b;
    g(a, b);
}
void g(int x, int y)
{
    double z;
    h(z);
}
void h(double a) {
    …
}
```

UNIVERSITÉ CÔTE D'AZUR

Exemple Génération de Code

https://godbolt.org/

Code C

```
void h(double a) {
}

void g(int x, int y) {
    double z;
    h(z);
}

void f() {
    int a, b;
    g(a, b);
}
```

Code machine

```
h(double):
        push    rbp
        mov     rbp, rsp
        movsd   QWORD PTR [rbp-8], xmm0
        nop
        pop     rbp
        ret
g(int, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     DWORD PTR [rbp-20], edi
        mov     DWORD PTR [rbp-24], esi
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rbp-32], rax
        movsd   xmm0, QWORD PTR [rbp-32]
        call    h(double)
        nop
        leave
        ret
f():
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     edx, DWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, edx
        mov     edi, eax
        call    g(int, int)
        nop
        leave
        ret
```

UNIVERSITÉ
CÔTE D'AZUR

Quelques questions

Pour vérifier si vous avez compris...

UNIVERSITÉ CÔTE D'AZUR

Quizz : Régions Mémoires

- ✓ Seules les variables globales ou statiques locales sont initialisées à 0 par défaut
- ✓ Une variable non-initialisée locale ou créée sur le tas a une valeur non-définie
- ✓ Dans quelles zones mémoires ces différentes variables sont-elles localisées ?

```
int var1;                                  <-- bss
char var2[] = "buf1";                      <-- data
main() {
    int var3;                              <-- stack
    static int var4;                       <-- bss
    static char var5[] = "buf2";           <-- data
    char * var6;                           <-- stack
    var6 = malloc(512);                    <-- heap
}
```

UNIVERSITÉ CÔTE D'AZUR

Quizz: Exemple 1

- ✓ **Quel est le problème dans ce code ?**

```
#include <stdio.h>
#include <stdlib.h>
int * foo(void){
    int t[] = {1, 2, 3};
    return t;
}

int main(void) {
    int * t = foo();
    for (int i = 0; i < 3; i++) {
        printf("%d ", t[i]);
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

UNIVERSITÉ CÔTE D'AZUR

Quizz: Exemple 2

✓ **Que se passe-t-il avec ce code ?**

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

int * foo(void){
    int v[] = {1, 2, 3};
    int *u = v;
    return u;
}
```

```
int main(void) {
    int i;
    int * t = foo();

    for (i = 0; i < 3; i++) {
        printf("%d ", t[i]);
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

UNIVERSITÉ CÔTE D'AZUR

Quizz: Exemple 3

✓ **Que se passe-t-il avec ce code ?**

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

int * foo(void){
    int v[] = {1, 2, 3};
    int *u = v;
    return u;
}

void bar(void) {
    int w[] = {4, 8, 16, 32};
}
```

```
int main(void) {
    int * t = foo();
    bar();

    int i;
    for (i = 0; i < 3; i++) {
        printf("%d ", t[i]);
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

UNIVERSITÉ
CÔTE D'AZUR

Optimiser l'utilisation mémoire

Manipulation de la mémoire à la résolution du bit

Champs de bits

UNIVERSITÉ CÔTE D'AZUR

Manipulation de bits: Opérateurs

- ✓ Six opérateurs de manipulation des bits
  - & : et (ne pas confondre avec && : et logique)
    - Donne 1 si les 2 bits sont à 1
  - | : ou inclusif (ne pas confondre avec || : ou logique)
    - Donne 1 si *au moins* un bit est à 1
  - ^ : ou exclusif
    - Donne 1 si *un seul* bit est à 1
  - ~ : négation ou complément
    - Donne 1 si 0 et donne 0 si 1
  - >> : décalage à droit
    - Revient à diviser par 2^x (x est le décalage)
  - << :décalage à gauche (vers le bit de poids fort)
    - Revient à multiplier par 2^x (x est le décalage) si pas de perte
- ✓ Les opérateurs combinés existent: &= |= ^= <<= >>=

UNIVERSITÉ
CÔTE D'AZUR

Drapeaux & Masques

- ✓ Un masque (*mask*)
  - Un ou plusieurs bits
  - Permet de masquer (supprimer) un ou des bits dans un mot
- ✓ Un drapeau (*flag*)
  - Un bit dans un mot
  - Permet de représenter la présence ou absence d'un état
  - Set Flag: `[flags] |= [bitmask]`
  - Clear Flag: *`[flags] &= ~[bit pattern]`*
  - Check Flag: `[flags] & [bitmask]`
  - *Toogle Flag:* `[flags] ^= [bitmask]`

UNIVERSITÉ CÔTE D'AZUR

Champs de bits

- ✓ Regrouper au plus juste (dans un nombre d'octets moindre) plusieurs valeurs
  - Aucun standard C ne fixe les règles d'implémentation
  - Donc dépendant de votre implémentation
- ✓ Seuls types autorisés
  - `_Bool`, (`short`) `int` et `unsigned` (`short`) `int` (ANSI C)
  - `long` et `char` (`signed` et `unsigned`) (extension Microsoft)

```
typedef struct {
  unsigned short day;
  unsigned short month;
  unsigned short year;
}
```

3 x 16 bits (Linux 32b,64b)

```
typedef struct {
  unsigned short day : 5;
  unsigned short month : 4;
  unsigned short year : 7;
}
```

1 x 16 bits (Linux 32b,64b)

UNIVERSITÉ
CÔTE D'AZUR

Pour aller plus loin

Organisation des informations dans un fichiers

Format ELF

Utilisé sur la plupart des OS Unix (sauf Mac OS X)

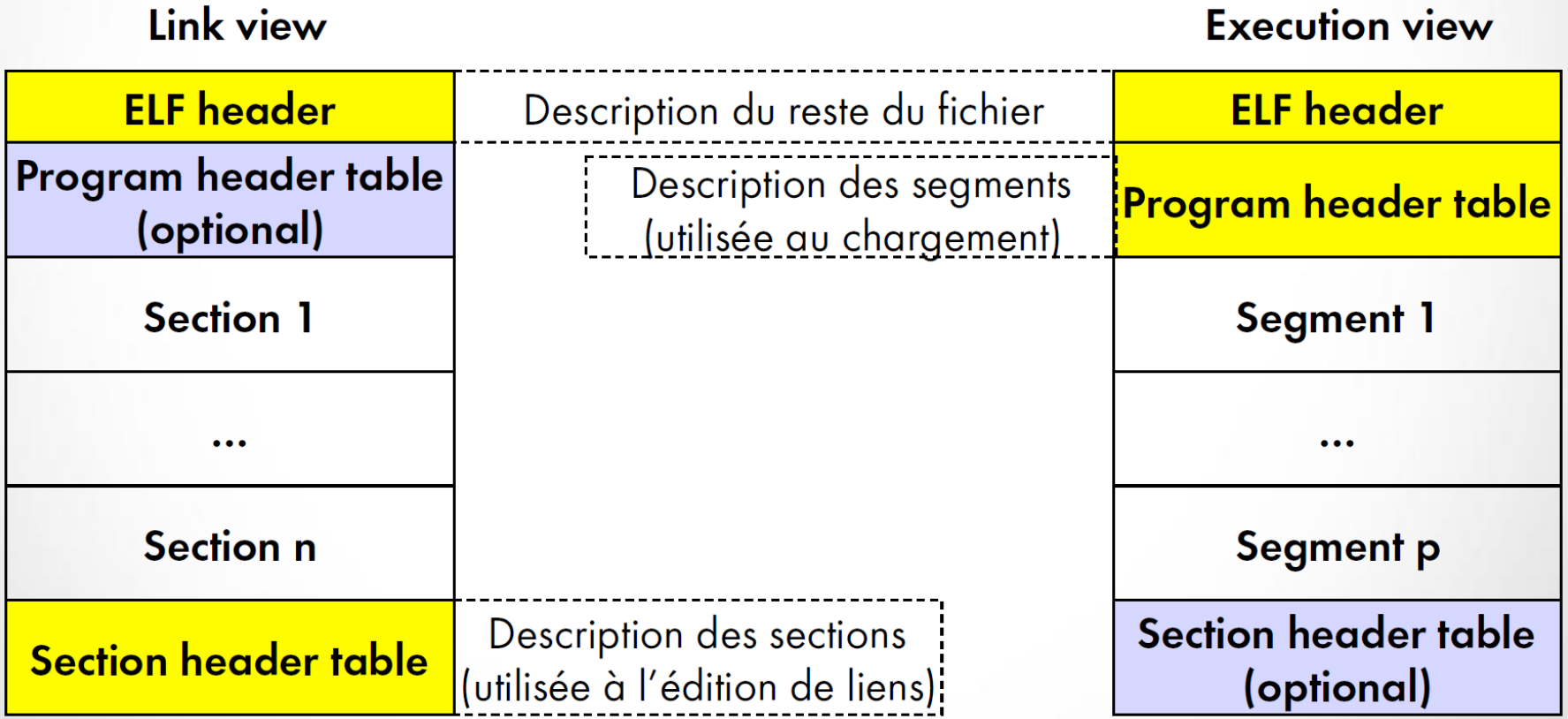
UNIVERSITÉ CÔTE D'AZUR

Format des objets, Bibliothèques et Exécutables

- ✓ Ancien format `a.out` obsolète
- ✓ ELF : Executable and Linking Format
  - Format unique pour fichiers exécutables, objets, et bibliothèques (partagées)
  - Format en principe portable
  - Fichier exécutable (*executable file*)
    - Information nécessaire au SE pour créer l'image mémoire d'un processus
  - Fichier relogeable (*relocatable file*)
    - Fichier destiné à subir une édition de liens avec d'autres fichiers objets pour créer un exécutable ou une bibliothèque partagée
  - Fichier objet partagé (*shared object file*)
    - Information pour l'édition de liens statique ou dynamique
  - Lecture des informations d'un fichier ELF
    - `readelf`
    - bibliothèque `libelf`

UNIVERSITÉ CÔTE D'AZUR

Format ELF 1/3



UNIVERSITÉ
CÔTE D'AZUR

Format ELF 2/3

- ✓ Sections
  - Code, données
  - Tables de chaînes de caractères
  - Information d'édition de liens, de relogement
  - Tables de symboles, informations de « debugging »
  - Commentaires, notes…
- ✓ Segments
  - Groupement des informations éparpillées dans diverses sections afin de construire l'image mémoire du processus

UNIVERSITÉ
CÔTE D'AZUR

Format ELF 3/3