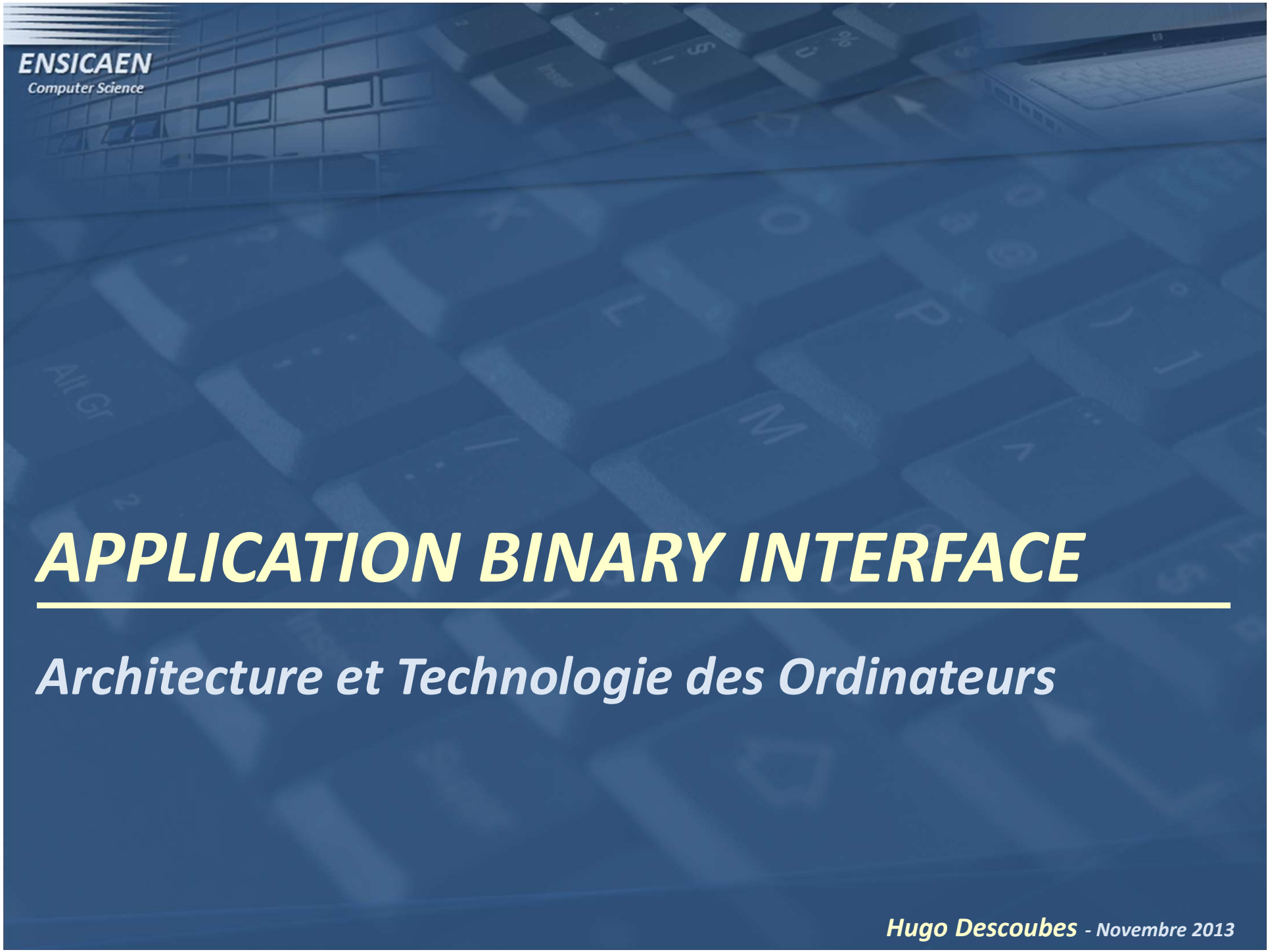
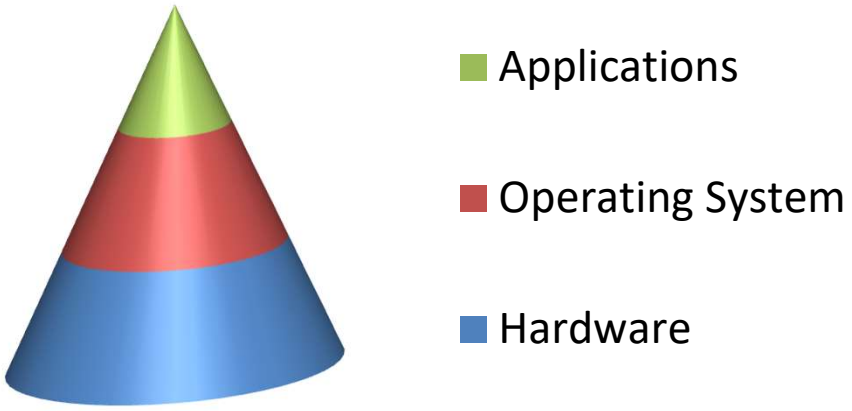


APPLICATION BINARY INTERFACE

Architecture et Technologie des Ordinateurs

Hugo Descoubes - Novembre 2013

Un système travaillant sur une architecture à CPU peut très souvent être découpé en couche. Il s'agit d'une solution mixte logicielle (OS et couches applicatives) et matérielle. Un développeur logiciel dit ''bas niveau'' travaille dans les couches basses de ce modèle :



Observons en quelques chiffres, la répartition des marchés des systèmes d'exploitation sur quelques grands domaines d'application :

- ***Windows de Microsoft :*** *~91% du marché des ordinateurs personnels en 2014 (56,3% pour W7 et 13,5% pour W8/8.1), ~2,5% du marché des Smartphones en 2014, 55% des serveurs en 2014*

- ***UNIX et UNIX-like (GNU/Linux, iOS, MAC OS X, Android …):*** *90% du marché des Smartphones en 2014 (Android ~47%), 67% des serveurs en 2014, GNU/Linux ~97% des superordinateurs en 2014*


sources : *StatCounter, NetApplications*

Vous aurez un enseignement dédié aux systèmes d'exploitation en 2A.

Malheureusement, développement bas niveau ne veut pas dire développement simple. Un ingénieur travaillant dans ce domaine doit notamment être compétent sur les points suivants :

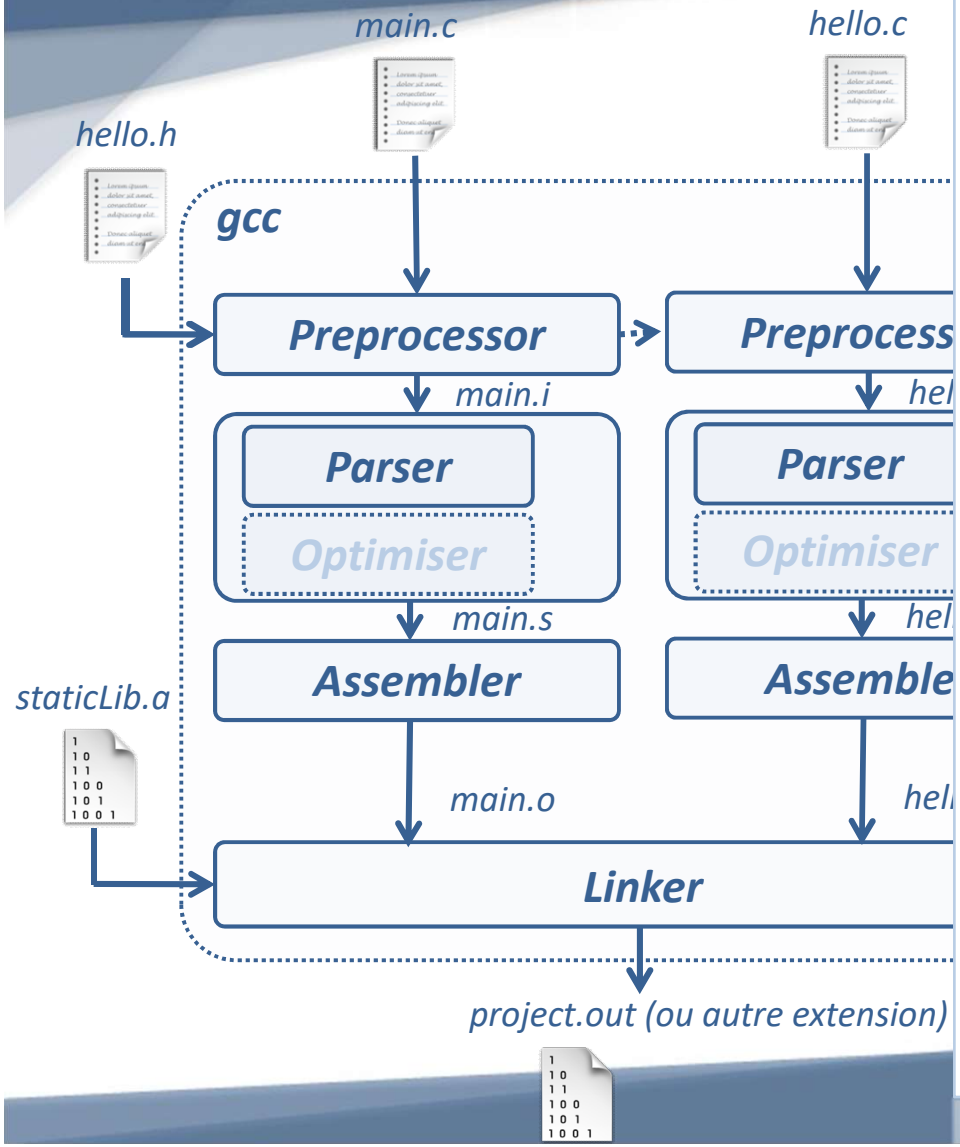
- ***Architectures matérielles*** *(CPU, hiérarchie et gestion mémoire, gestion périphériques, mécanismes d'optimisations …)*
- ***Langages de programmation*** *(essentiellement C/C++ et assembleur)*
- ***Outils de Développement Logiciel*** *(IDE, chaîne de compilation C, outils de debuggage et de profilage, programmation concurrente, programmation parallèle …)*

Effectuons quelques rappels sur une chaîne de compilation C (C toolChain ou C toolSuite). ***Les slides qui suivent sont à savoir par cœur****.*

Les exemples suivants sont donnés sous la chaîne de compilation GCC (GNU Compilation Collection, ***http://gcc.gnu.org/****). L'architecture est la même que toute autre toolChain C, cependant les formats et extensions des fichiers intermédiaires ne sont pas standardisées et peuvent changer d'une chaîne à une autre ou d'une plateforme matérielle à une autre.*

Cet enseignement s'appuie sur les compétences enseignées dans les enseignements "Outils de Développement Logiciel" et "Programmation et langage C".

Bas niveau – C ToolChain – ELF file

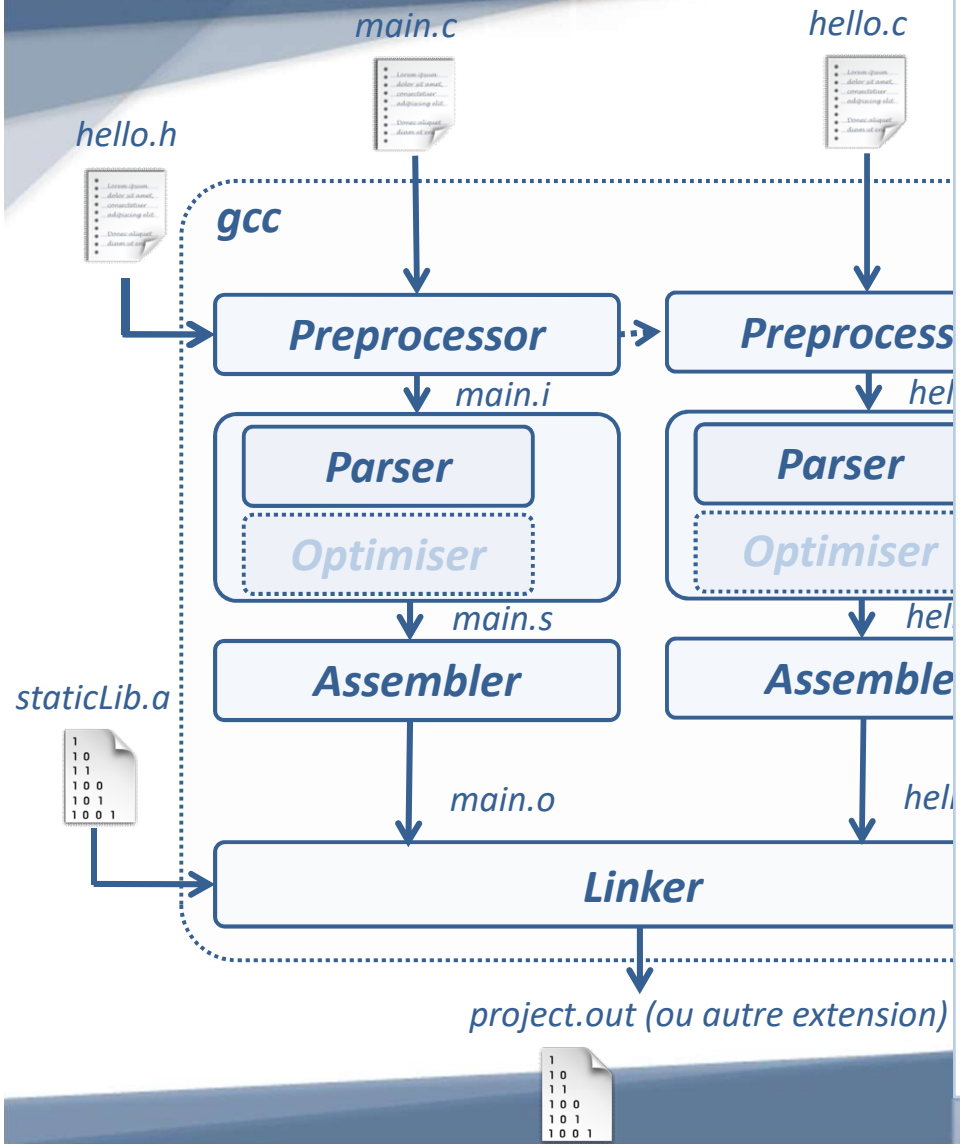


- ***Processus de compilation :*** *Réalisation respective des traitement suivants : Analyse lexicale, pré-traitement, analyse syntaxique, analyse sémantique, génération de code, ''optimisation'', édition des liens*

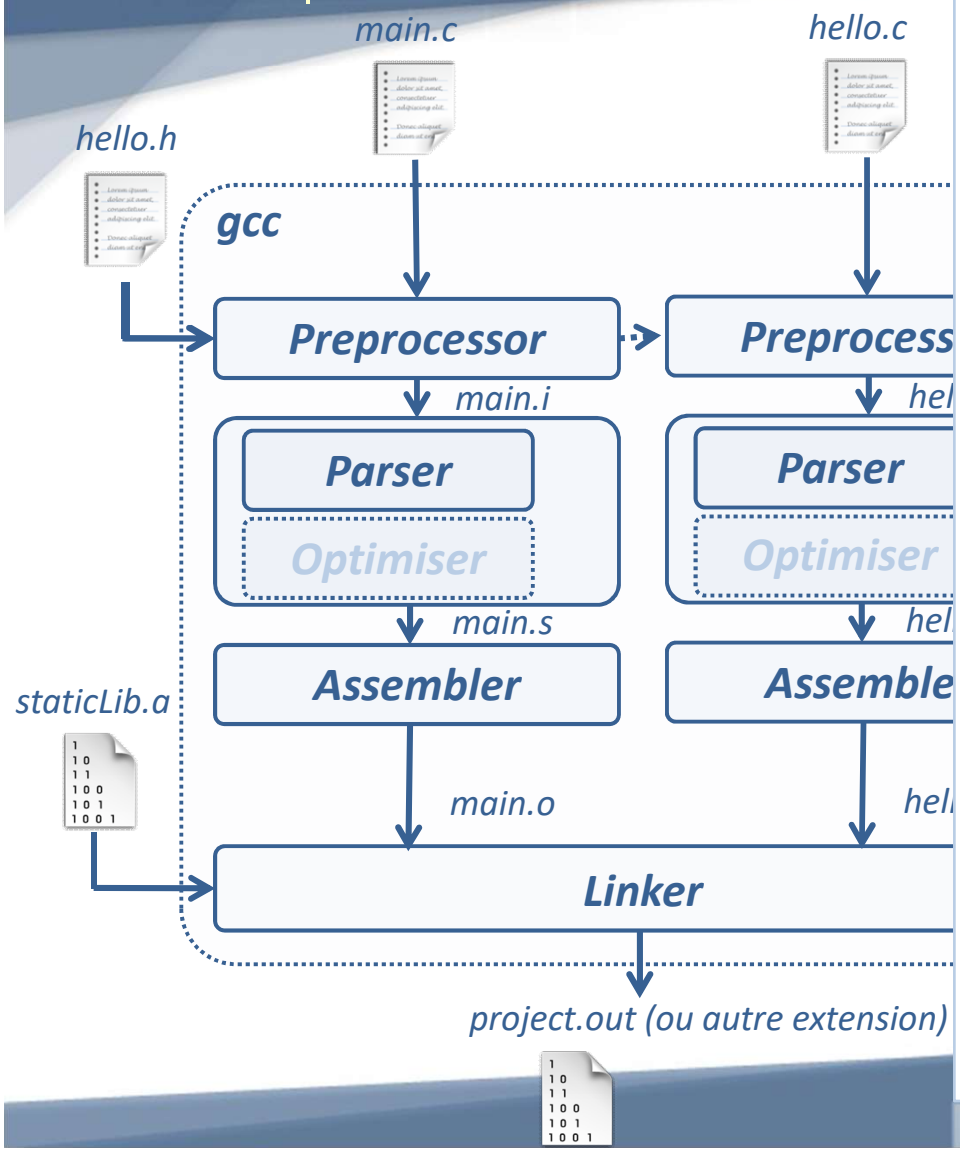
- ***Analyse Lexicale :*** *Elimination commentaires, espaces, détection des mots clés, opérateurs (!=, <= …), chaînes de caractères, constantes numériques*

- ***Préprocesseur :*** *Etapes d'inclusion de code, de substitution de chaînes de caractères et de compilation conditionnelle. Directives de pré-compilation ''#'' et opérateurs ''#pragma'' depuis la norme C99*

Bas niveau – C ToolChain – ELF file

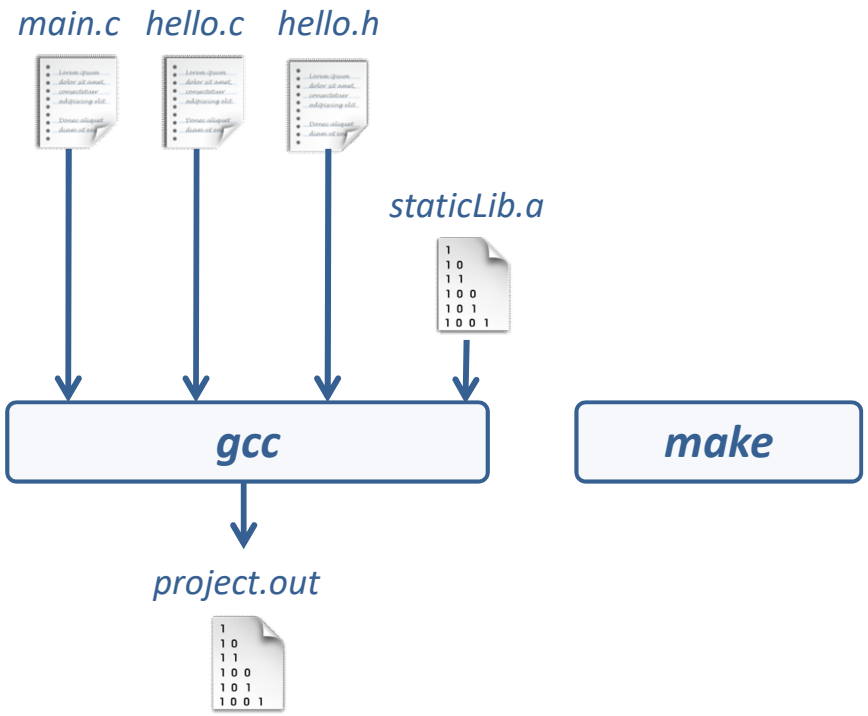


- ***Parser :*** *Analyse séquentielle du code*
  - ***Analyse syntaxique :*** *analyse la structure du code, vérifie le respect des formalismes et syntaxes inhérents au langage compilé*
  - ***Analyse sémantique :*** *résolution des noms, génération de la table des symboles, compatibilité des types …*
- ***Génération du code :*** *Etape dépendant de l'architecture CPU cible (Instruction Set Architecture). Nous parlons de cross-compilation (vs compilation native) lorsque l'architecture cible est différente de celle effectuant la compilation.*
- ***Optimisation (optionnelle) :*** *étage très délicat à développer et dépendant de l'architecture CPU cible.*

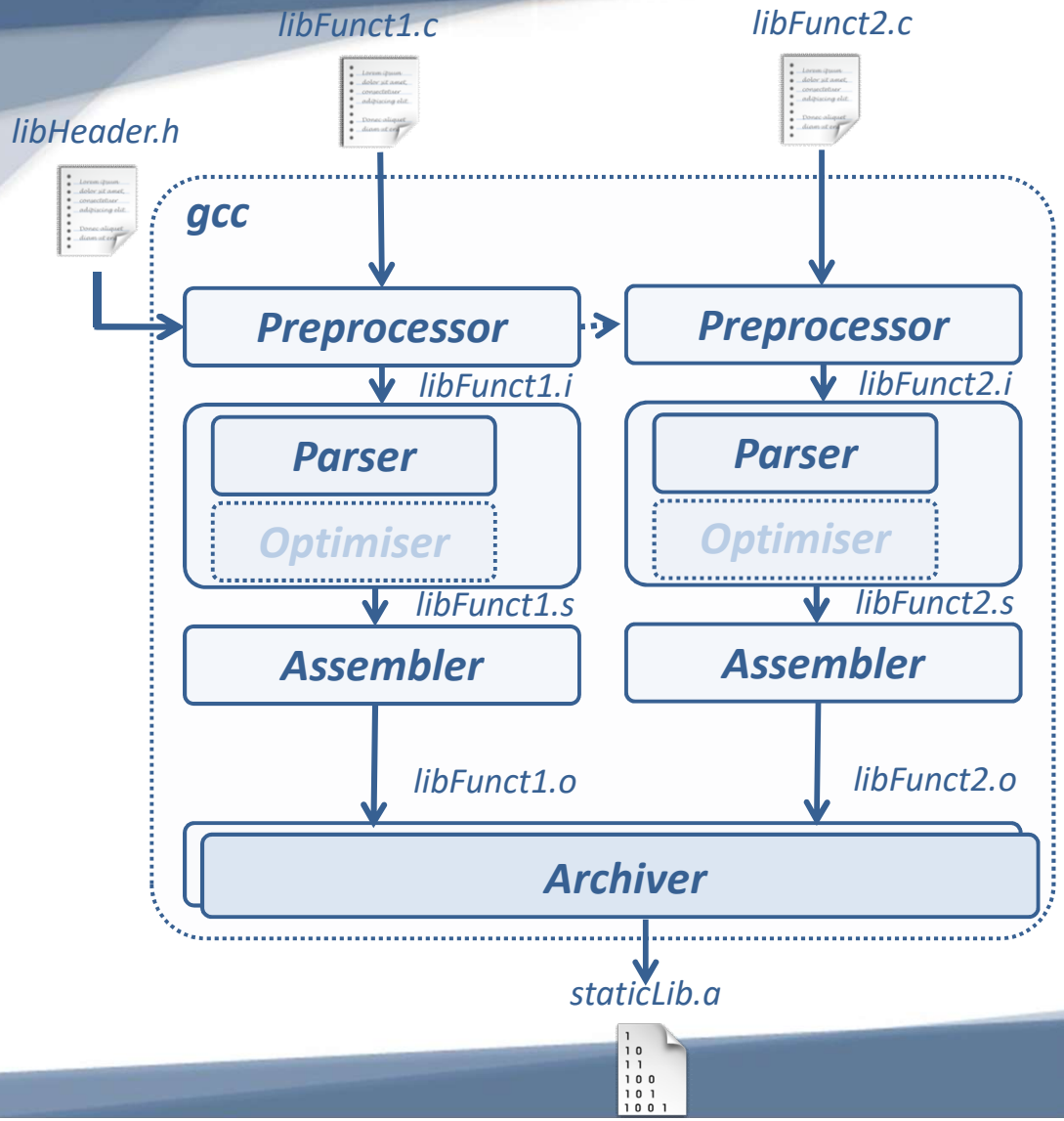


- ***Assembleur :*** *Génération code objet relogeable pour architecture cible (code binaire). Référencement par symboles, aucune dépendance avec le modèle mémoire de l'architecture cible.*

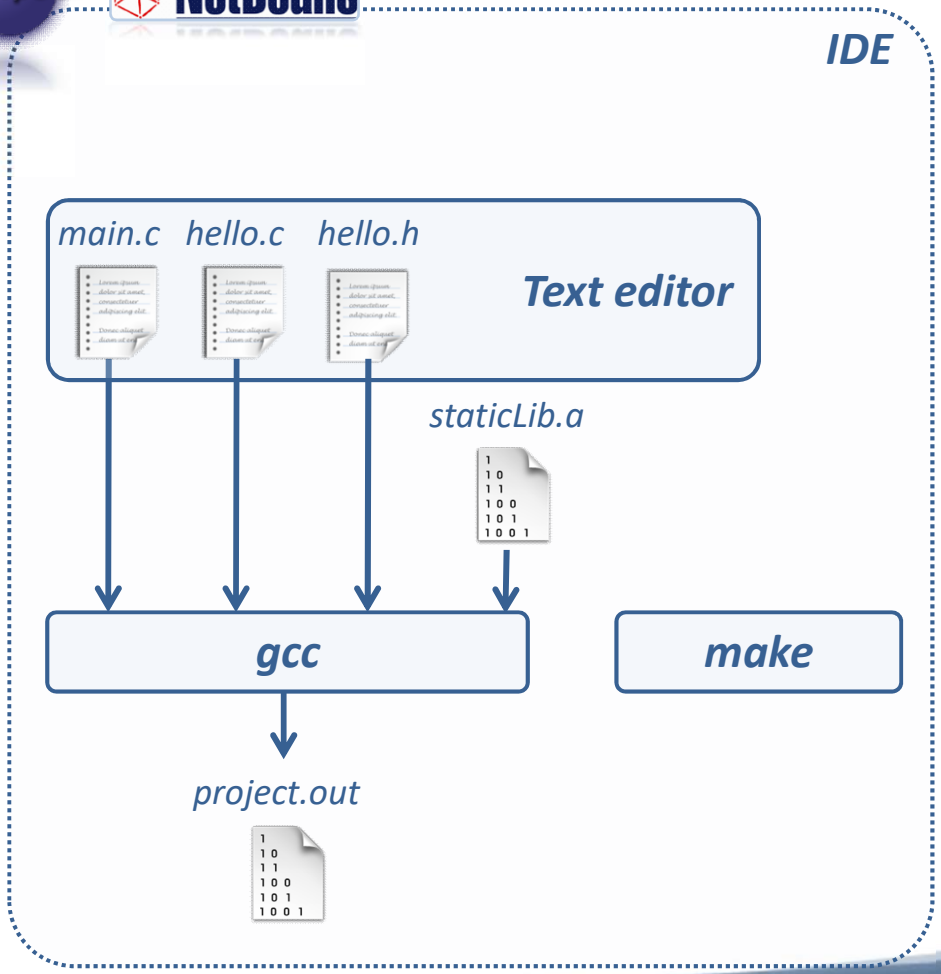
- ***Editeur de liens :*** *Liens entre fichiers objets, bibliothèques statiques, bibliothèques dynamiques chargées à l'exécution. Résolution de la table des symboles et résolution des liens avec le modèle mémoire de l'architecture cible. Génération d'un code binaire exécutable absolu ou relogeable (dépend de l'architecture cible et de la stratégie de chargement du code exécutable)*



- ***make :*** *utilitaire de programmation pour l'automatisation de procédures de compilation*
- ***Archiver :*** *construction de bibliothèques statiques. Archive réalisée à partir de fichiers objets*



- ***make :*** *utilitaire de programmation pour l'automatisation de procédures de compilation*
- ***Archiver :*** *construction de bibliothèques statiques. Archive réalisée à partir de fichiers objets*



- ***make :*** *utilitaire de programmation pour l'automatisation de procédures de compilation*
- ***Archiver :*** *construction de bibliothèques statiques. Archive réalisée à partir de fichiers objets*
- ***Integrated Developement Environment :*** *Aide au développement logiciel. Intègre généralement un éditeur de texte, automatisation procédure de compilation, debugger, utilitaires divers…*

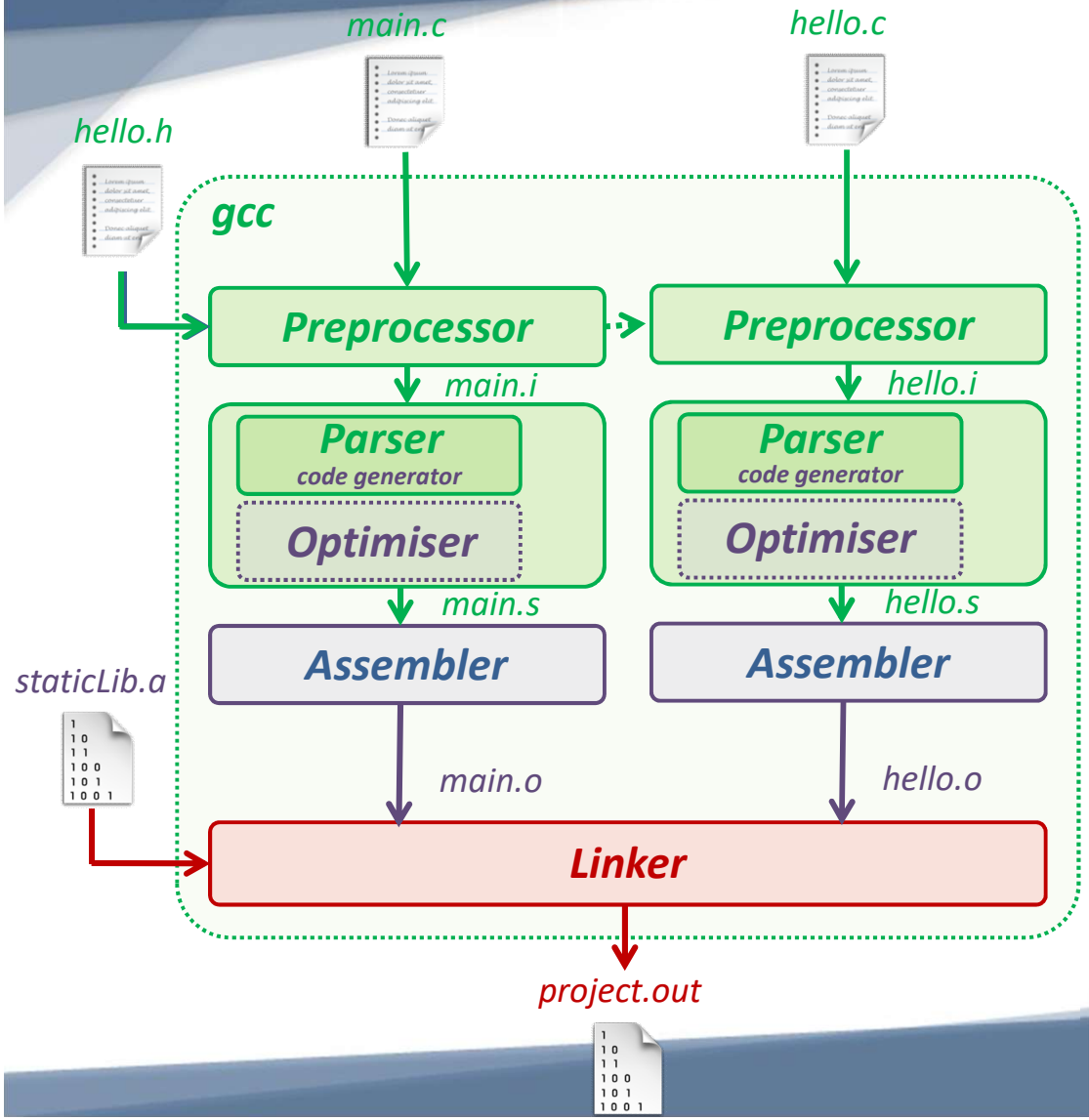
Observons les 3 trois principaux environnements de compilation utilisés sur architecture x86 :

- ***Visual Studio** proposé par Windows*

- ***Intel C++ Compiler XE** proposé par Intel. Propose des outils très puissants d'optimisation et de profilage pour architecture Intel (Advisor XE, Vtune Amplifier, Inspector XE …)*

- ***GCC (GNU Compiler Collection) avec/sans IDE (Eclipse, Netbeans …)** issu du monde de l'Open Source ayant vocation a être multiplateforme (cross-compilation ARM, MIPS, PPC …). Les deux principaux environnement de compilation rencontrés sous Windows sont Cygwin et MinGW.*

Bas niveau – C ToolChain – ELF file



- *Les 2 premières étapes de la compilation sont architecture agnostique.*

Vous aurez un enseignement dédié à la compilation et l'étude du parser en 2A (graphes et automates)

- ***code generator, optimiser et assembler :*** *dépendance avec l'architecture CPU cible mais pas du modèle mémoire (références symboliques)*

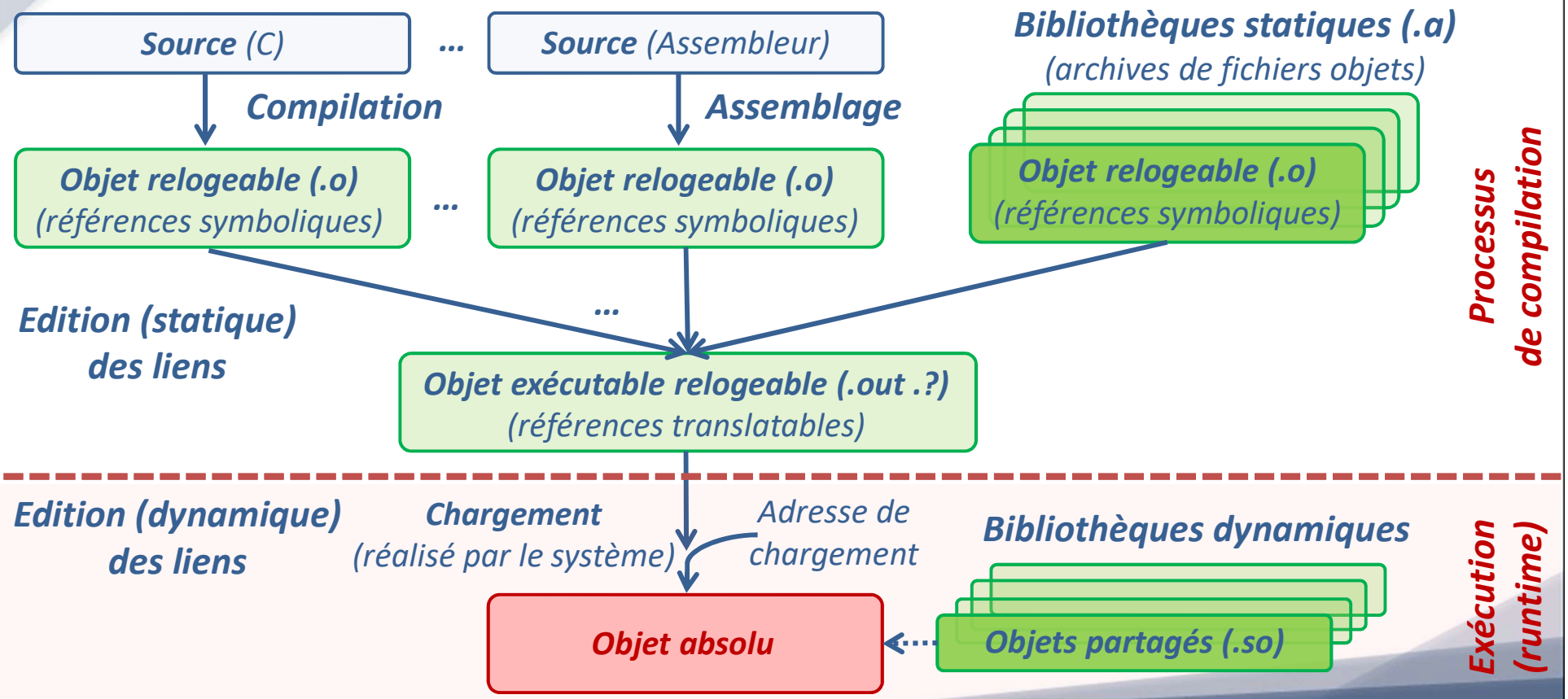
- ***Linker :*** *dépend du modèle mémoire de l'architecture cible*

- ***Fichier ELF (Executable and Linkable Format) :***

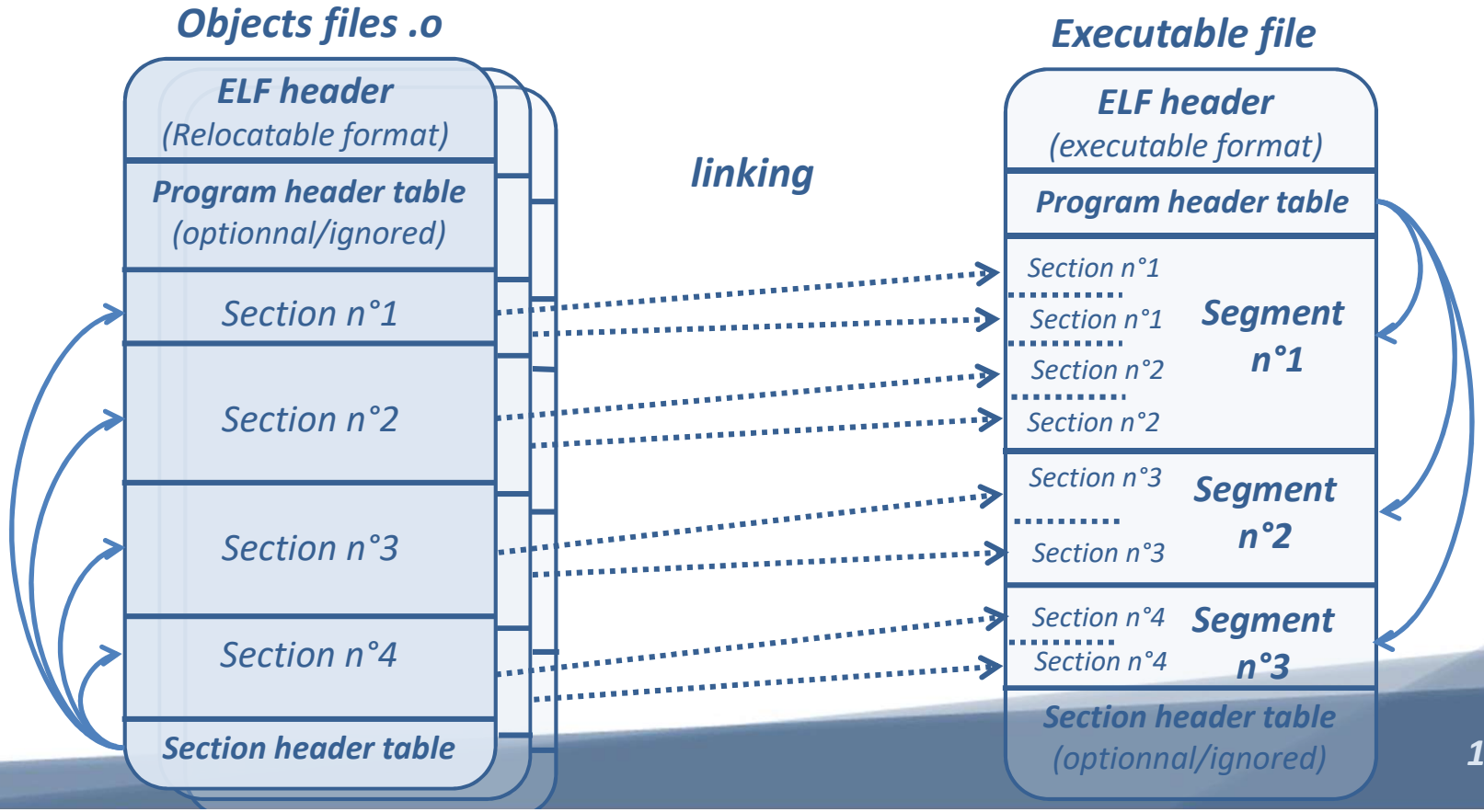
Le format de fichier ELF sert à l'enregistrement de programmes compilés (fichiers objets, exécutables, bibliothèques statiques et dynamiques, modules kernel). Prenons les principales extensions de fichiers compilés rencontrées sous GNU/Linux (tous des fichiers ELF) .o (object), .a (archive de .o), .so (shared object), .ko (kernel object). Ce format plus flexible unifie et remplace les anciens format a.out et COFF.

Le format ELF est extrêmement répandu sur systèmes UNIX-like (GNU/Linux, FreeBSD, Solaris, OpenBSD, Android …) ainsi que sur grand nombre d'autres plateformes (PS-2, PS-3, PSP, Wii, SymbianOS v9 …).

Avant de présenter le format de fichier ELF, présentons le cycle de vie d'un programme (en vert, fichiers ELF) :



*Un fichier ELF est toujours constitué d'une en-tête de fichier (cf. fichier **elf.h**), le reste de la structure diffère en fonction du type de fichier compilé (exécutable, bibliothèque partagée, objet …) :*



*Observons, en utilisant la commande **readelf** proposée avec **binutils** (ou la commande **objdump**), les en-têtes de fichiers ELF respectivement pour un fichier objet .o, exécutable et la bibliothèque standard du C, qui est un objet partagé .so :*

```
vmlinux@vmlinux:~/Desktop/segfault$ readelf -e objdump-minimal.o
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          340 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         13
  Section header string table index: 10
```

```
vmlinux@vmlinux:~/Desktop/segfault$ readelf -e ./a.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x80483f0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          4428 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         30
  Section header string table index: 27
```

```
vmlinux@vmlinux:~/Desktop/segfault$ readelf -e /lib/i386-linux-gnu/libc.so.6
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x19630
  Start of program headers:          52 (bytes into file)
  Start of section headers:          1732720 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         10
  Size of section headers:           40 (bytes)
  Number of section headers:         35
  Section header string table index: 34
```

*Rappelons que pour les **langages compilés**, deux grands types d'allocations mémoire sont rencontrés pour la gestion des variables :*

- ***Allocations statiques :** Allocation mémoire à la compilation (compile-time). Prenons l'exemple des variables globales, locales qualifiées de static … Chaque chaîne de compilation C est très structurée, classe les variables statiques par famille et les range dans des sections spécifiques (sections présentent dans fichier ELF). Observons quelques-unes des principales sections :*
  - ***.bss :** variables statiques non-initialisées. Par définition initialisées à zéro au démarrage*
  - ***.data :** variables statiques initialisées*
  - ***.rodata :** variables statiques en lecture seule (read-only)*

Observons le contenu des sections d'un fichier objet élémentaire après compilation. Accès aux variables par adressage relatif aux adresses de base des sections :

```c
/**
* @file objdump-minimal.c
* @author
* @date novembre 2013
*/
#include <stdio.h>

char data[]="Bonjour le Monde\n";

/**
* @fn void main (void)
* @brief program entry point
*/
int main(int argc,char* argv[]){
    char* rodata="Hello World\n";

    printf("%s%s", data, rodata);

return 0;
}
```

```
vmlinux@vmlinux:~/Desktop/segfault$ gcc -c objdump-minimal.c
vmlinux@vmlinux:~/Desktop/segfault$ objdump -s objdump-minimal.o

objdump-minimal.o:     file format elf32-i386

Contents of section .text:
 0000 5589e583 e4f083ec 20c74424 1c000000  U.......  .D$....
 0010 00b80d00 00008b54 241c8954 2408c744  .......T$..T$..D
 0020 24040000 00008904 24e8fcff ffffb800  $.......$.......
 0030 000000c9 c3                          .....
Contents of section .data:
 0000 426f6e6a 6f757220 6c65204d 6f6e6465  Bonjour le Monde
 0010 0a00                                 ..
Contents of section .rodata:
 0000 48656c6c 6f20576f 726c640a 00257325  Hello World..%s%
 0010 7300                                 s.
```

Section .text : *binaire du programme*

Adresses relatives dans les sections *(représentation hexadécimale)*

Contenu des sections *(représentation hexadécimale)*

Contenu des sections *(représentation sous forme de caractères)*

- ***Allocations automatiques :*** *Allocation mémoire à l'exécution (run-time). Vu par la suite.*

  - ***Gestion par la pile (ou stack) :*** *variables locales, paramètres, valeur de retour, adresse de retour et contexte d'exécution de fonction. Nous parlons également souvent d'allocation automatique.*

- ***Allocations dynamiques :*** *Allocation mémoire à l'exécution (run-time). Vu par la suite.*

  - ***Gestion par le tas (ou heap) :*** *fonctions malloc, free et variantes.*

Observons l'en-tête contenant la table des sections pour le programme précédemment compilé (fichier objet).

```c
/**
* @file objdump-minimal.c
* @author
* @date novembre 2013
*/
#include <stdio.h>

char data[]="Bonjour le Monde\n";

/**
* @fn void main (void)
* @brief program entry point
*/
int main(int argc,char* argv[]){
    char* rodata="Hello World\n";

    printf("%s%s", data, rodata);

return 0;
}
```

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000035 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 000444 000020 08     11   1  4
  [ 3] .data             PROGBITS        00000000 00006c 000012 00  WA  0   0  4
  [ 4] .bss              NOBITS          00000000 000080 000000 00  WA  0   0  4
  [ 5] .rodata           PROGBITS        00000000 000080 000012 00   A  0   0  1
  [ 6] .comment          PROGBITS        00000000 000092 00002b 01  MS  0   0  1
  [ 7] .note.GNU-stack   PROGBITS        00000000 0000bd 000000 00      0   0  1
  [ 8] .eh_frame         PROGBITS        00000000 0000c0 000038 00   A  0   0  4
  [ 9] .rel.eh_frame     REL             00000000 000464 000008 08     11   8  4
  [10] .shstrtab         STRTAB          00000000 0000f8 00005f 00      0   0  1
  [11] .symtab           SYMTAB          00000000 000360 0000c0 10     12   9  4
  [12] .strtab           STRTAB          00000000 000420 000024 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

There are no program headers in this file.
```

- ***Section .symtab :*** *cette section, nommée* ***table des symboles*** *est essentielle. La compilation est un processus dépendant du langage et de l'architecture du CPU mais indépendant du mapping mémoire. Les binaires compilés (fichiers objets) travaillent par références symboliques, la table des symboles lie les symboles à des adresses relatives vers différentes sections.*

```c
/**
* @file objdump-minimal.c
* @author
* @date novembre 2013
*/
#include <stdio.h>

char data[]="Bonjour le Monde\n";

/**
* @fn void main (void)
* @brief program entry point
*/
int main(int argc,char* argv[]){
    char* rodata="Hello World\n";

    printf("%s%s", data, rodata);

return 0;
}
```

```
Symbol table '.symtab' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS objdump-minimal.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    8
     8: 00000000     0 SECTION LOCAL  DEFAULT    6
     9: 00000000    18 OBJECT  GLOBAL DEFAULT    3 data
    10: 00000000    53 FUNC    GLOBAL DEFAULT    1 main
    11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

Section .data

Section .text

Adresse dans la section

Nous pouvons également rencontrer une table de relocation (liste de trous à compléter avec la table des symboles). A titre indicatif, les bibliothèques dynamiques sont liées à l'exécution par le chargeur du noyau au démarrage du programme. La section ***.plt (procedure linkage table)*** *effectue une redirection à l'exécution vers l'adresse absolue de la procédure cible (printf dans notre cas).*

```c
/**
* @file objdump-minimal.c
* @author
* @date novembre 2013
*/
#include <stdio.h>

char data[]="Bonjour le Monde\n";

/**
* @fn void main (void)
* @brief program entry point
*/
int main(int argc,char* argv[]){
    char* rodata="Hello World\n";

    printf("%s%s", data, rodata);

return 0;
}
```

```
080483e4 <main>:
main():
 80483e4:       55                      push   %ebp
 80483e5:       89 e5                   mov    %esp,%ebp
 80483e7:       83 e4 f0                and    $0xfffffff0,%esp
 80483ea:       83 ec 20                sub    $0x20,%esp
 80483ed:       c7 44 24 1c f0 84 04    movl   $0x80484f0,0x1c(%esp)
 80483f4:       08
 80483f5:       b8 fd 84 04 08          mov    $0x80484fd,%eax
 80483fa:       8b 54 24 1c             mov    0x1c(%esp),%edx
 80483fe:       89 54 24 08             mov    %edx,0x8(%esp)
 8048402:       c7 44 24 04 14 a0 04    movl   $0x804a014,0x4(%esp)
 8048409:       08
 804840a:       89 04 24                mov    %eax,(%esp)
 804840d:       e8 ee fe ff ff          call   8048300 <printf@plt>
 8048412:       b8 00 00 00 00          mov    $0x0,%eax
 8048417:       c9                      leave
 8048418:       c3                      ret
```

ENSICAEN
Computer Science

Merci de votre attention !