Architecture des Ordinateurs et Systèmes d'Exploitation

Partie 3 : Compilation
Cours

Fabrice BOISSIER & Elena KUSHNAREVA
2017/2018

fabrice.boissier@gmail.com
elena.kushnareva@malix.univ-paris1.fr

Compilation

Transforme du code source lisible…

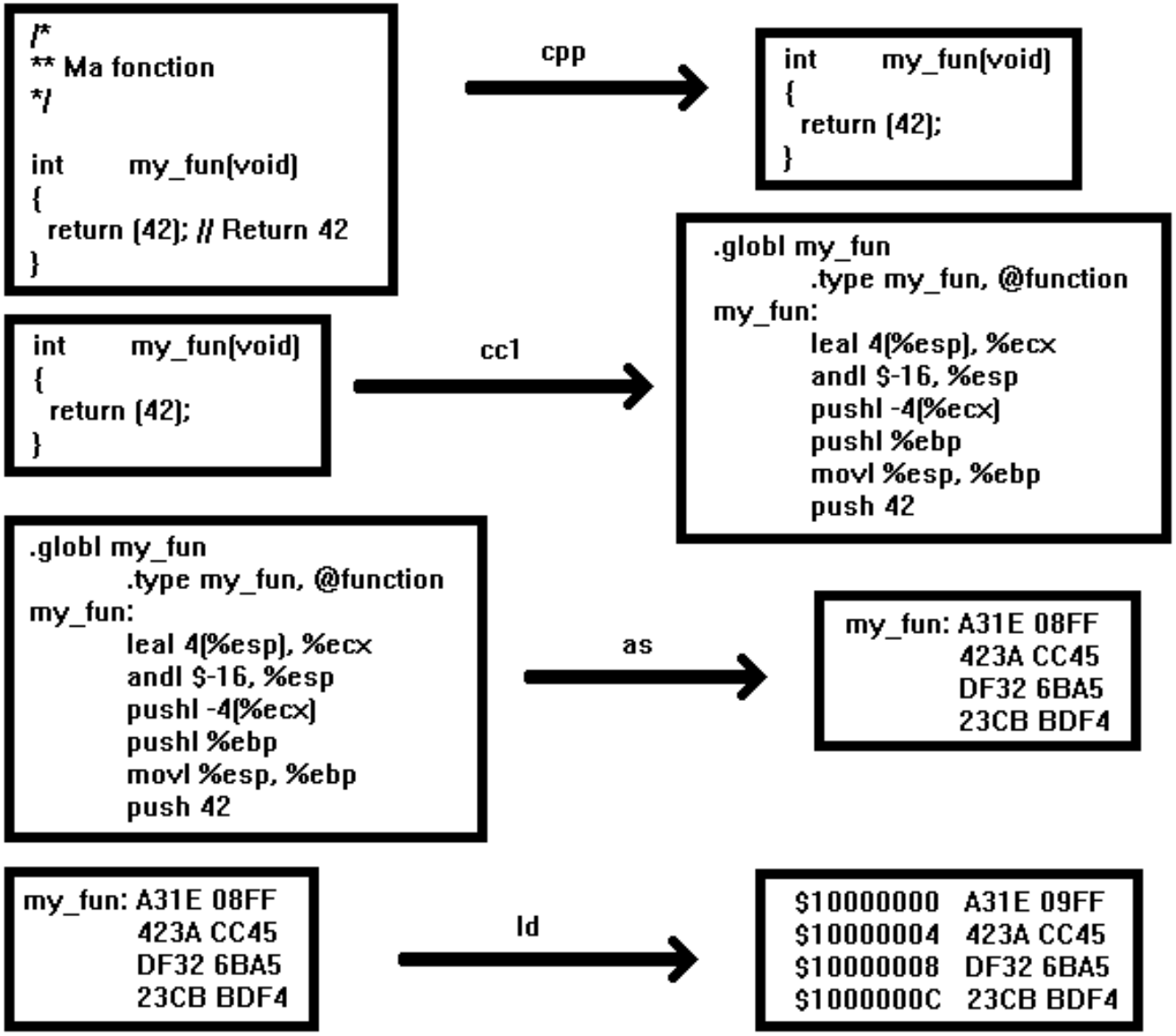
…en langage machine exécutable

1. Pré-Processing
   Pré-Processeur
   cpp / gcc –E

2. Compilation
   cc1 / gcc –S

3. Assembler
   Assemblage
   as / gcc –c

4. Link Edit
   Édition de Lien
   ld / gcc



Références Bibliographiques

- man gcc

- Compilers – Principles, Techniques and Tools (Dragon Book)
  Alfred V.Aho, Ravi Sethi, Jeffrey D.Ullman

- Modern Compiler Implementation (in C/Java/ML)
  Andrew W. Appel

Pré-Processeur / Pre-Processing

- `cpp (gcc -E)`

- Exécution des directives de pré-processing :
  - Retrait des commentaires
  - Inclusion des fichiers inclus
  - Transformation des constantes
  - Exécution de tout ordre indiqué dans le langage de macros...

- Résultat :
  Un fichier contenant uniquement du code (moins de logique visible)

```
gcc -E -P file.c -o file_out.c
```

file.h :

```
#ifndef FILE_H_
# define FILE_H_

#define MON_NOMBRE 42

int       my_fun(void);

#endif /* !FILE_H_ */
```

file.c :

```
#include "file.h"

/*
** Ma fonction
*/

int       my_fun(void)
{
  int a = 3;

  if (a == 3)
    return (0); // Return 0
  else
    return (MON_NOMBRE); // Return 42
}
```

cpp

gcc -E -P file.c -o file_out.c

file_out.c :

```
int       my_fun(void);

int       my_fun(void)
{
  int a = 3;

  if (a == 3)
    return (0);
  else
    return (42);
}
```

Compilation

- `cc1 (gcc -S)`

- Transforme le code dans le langage source (C, C++, ... humainement compréhensible) en code "assembleur" dédié au processeur cible (ASM x86 ou x86-64 pour Intel/AMD, ASM 68k pour Motorolla 68000, ASM ARM9 pour ARM9, ...).

- L'assembleur est une suite d'instructions exécutables par un processeur, mais dont la logique "humaine" a été transformée en logique "machine"/processeur.

```
gcc -S file_out.c -o file_asm.s
```

file_out.c :

```c
int	my_fun(void);

int	my_fun(void)
{
  int a = 3;

  if (a == 3)
    return (0);
  else
    return (42);
}
```

cc1

gcc -S file_out.c -o file_asm.s

file_asm.s :

```
	.file	"file_out.c"
	.text
	.globl	my_fun
	.type	my_fun, @function
my_fun:
.LFB0:
	.cfi_startproc
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	movl	$3, -4(%rbp)
	cmpl	$3, -4(%rbp)
	jne	.L2
	movl	$0, %eax
	jmp	.L3
.L2:
	movl	$42, %eax
.L3:
	popq	%rbp
	.cfi_def_cfa 7, 8
	ret
	.cfi_endproc
.LFE0:
	.size	my_fun, .-my_fun
	.ident	"GCC: (Debian 4.9.2-10) 4.9.2"
	.section	.note.GNU-stack,"",@progbits
```

Assembler / Assemblage

- `as (gcc -c)`

- Traduction directe des instructions assembleur en code binaire (ou code objet, aucun rapport avec la POO).

- Chaque fichier assembleur est traduit, et seuls les noms de fonctions restent encore "humainement lisibles", ceci afin de pouvoir créer des bibliothèques (libraries) ou des API.

- Le fichier généré est un fichier objet.

```
gcc -c file_asm.s -o file.o
as file_asm.s -o file.o
```

file_asm.s :

```
        .file   "file_out.c"
        .text
        .globl  my_fun
        .type   my_fun, @function
my_fun:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $3, -4(%rbp)
        cmpl    $3, -4(%rbp)
        jne     .L2
        movl    $0, %eax
        jmp     .L3
.L2:
        movl    $42, %eax
.L3:
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   my_fun, .-my_fun
        .ident  "GCC: (Debian 4.9.2-10) 4.9.2"
        .section        .note.GNU-stack,"",@progbits
```

as

gcc -c file_asm.s -o file.o
(as file_asm.s -o file.o)

file.o :

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 3e00 0100 0000 0000 0000 0000 0000  ..>.............
00000020: 0000 0000 0000 0000 1802 0000 0000 0000  ................
00000030: 0000 0000 4000 0000 0000 4000 0b00 0800  ....@.....@.....
00000040: 5548 89e5 c745 fc03 0000 0083 7dfc 0375  UH...E......}..u
00000050: 07b8 0000 0000 eb05 b82a 0000 005d c300  .........*...]..
00000060: 4743 433a 2028 4465 6269 616e 2034 2e39  GCC: (Debian 4.9
00000070: 2e32 2d31 3029 2034 2e39 2e32 0000 0000  .2-10) 4.9.2....
00000080: 1400 0000 0000 0000 017a 5200 0178 1001  .........zR..x..
00000090: 1b0c 0708 9001 0000 1c00 0000 1c00 0000  ................
000000a0: 0000 0000 1f00 0000 0041 0e10 8602 430d  .........A....C.
000000b0: 065a 0c07 0800 0000 002e 7379 6d74 6162  .Z........symtab
000000c0: 002e 7374 7274 6162 002e 7368 7374 7274  ..strtab..shstrt
000000d0: 6162 002e 7465 7874 002e 6461 7461 002e  ab..text..data..
000000e0: 6273 7300 2e63 6f6d 6d65 6e74 002e 6e6f  bss..comment..no
000000f0: 7465 2e47 4e55 2d73 7461 636b 002e 7265  te.GNU-stack..re
00000100: 6c61 2e65 685f 6672 616d 6500 0000 0000  la.eh_frame.....
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0100 0000 0400 f1ff  ................
00000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000140: 0000 0000 0300 0100 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 0000 0000 0300 0200  ................
00000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000170: 0000 0000 0300 0300 0000 0000 0000 0000  ................
00000180: 0000 0000 0000 0000 0000 0000 0300 0500  ................
00000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0300 0600 0000 0000 0000 0000  ................
000001b0: 0000 0000 0000 0000 0000 0000 0300 0400  ................
000001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001d0: 0c00 0000 1200 0100 0000 0000 0000 0000  ................
000001e0: 1f00 0000 0000 0000 0066 696c 655f 6f75  .........file_ou
000001f0: 742e 6300 6d79 5f66 756e 0000 0000 0000  t.c.my_fun......
```

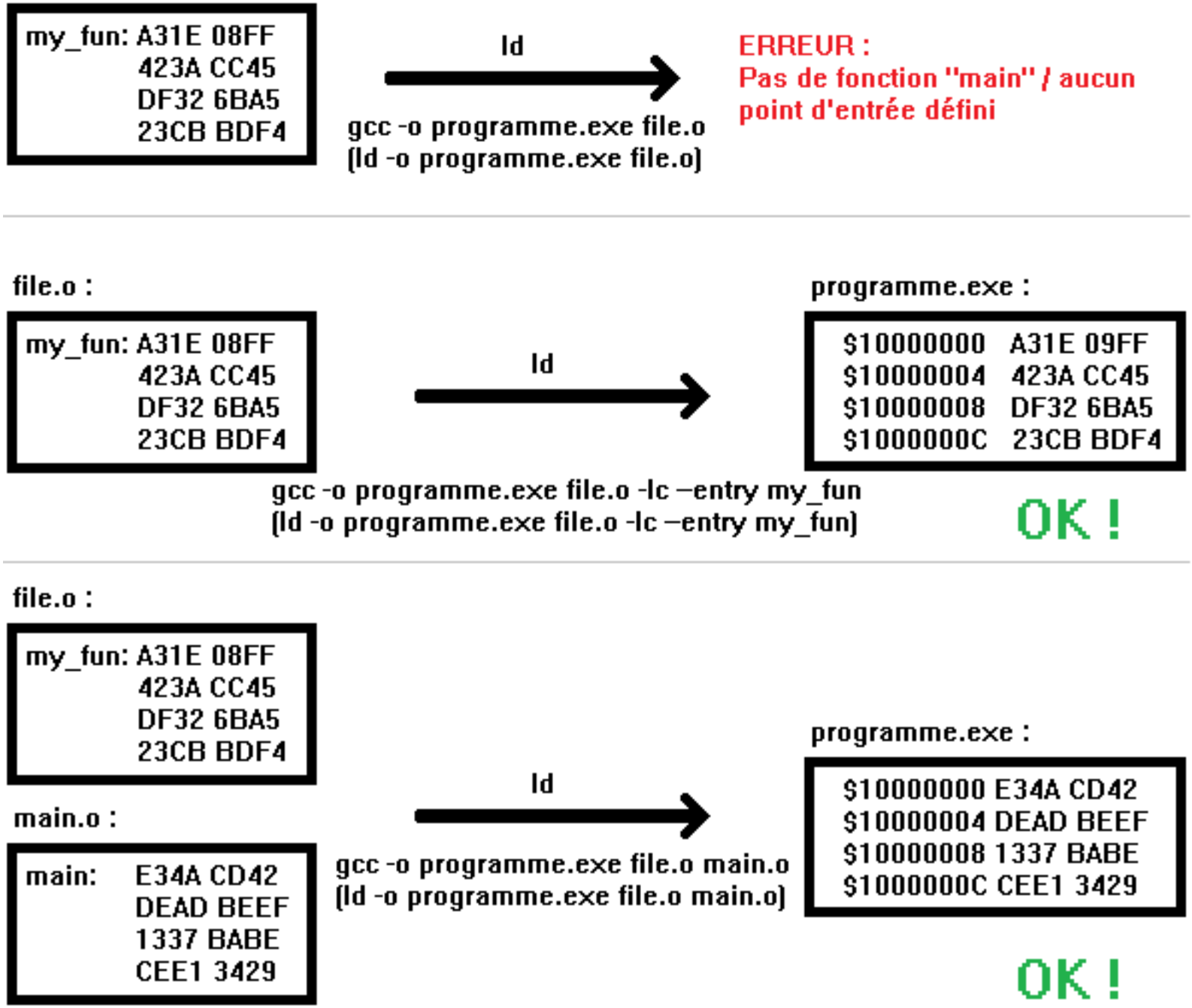
Link Edit / Edition de Lien

- `ld (gcc)`

- Les fichiers objets, dont les fonctions ou variables globales devraient être fournies par d'autres fichiers objets, sont réunis en un seul fichier, et les liens entre les noms des objets sont résolus (les noms de fonctions sont associés à des adresses fixées).

- Cette phase a beaucoup évolué avec le temps, et elle contient beaucoup plus d'étapes aujourd'hui.

```
gcc -o programme.exe file.o
ld -o programme.exe file.o
```


my_fun: A31E 08FF
423A CC45
DF32 6BA5
23CB BDF4
ld
gcc -o programme.exe file.o
(ld -o programme.exe file.o)
ERREUR :
Pas de fonction "main" / aucun point d'entrée défini
file.o :
my_fun: A31E 08FF
423A CC45
DF32 6BA5
23CB BDF4
ld
programme.exe :
$10000000 A31E 09FF
$10000004 423A CC45
$10000008 DF32 6BA5
$1000000C 23CB BDF4
gcc -o programme.exe file.o -lc –entry my_fun
(ld -o programme.exe file.o -lc –entry my_fun)
OK !
file.o :
my_fun: A31E 08FF
423A CC45
DF32 6BA5
23CB BDF4
main.o :
main: E34A CD42
DEAD BEEF
1337 BABE
CEE1 3429
ld
gcc -o programme.exe file.o main.o
(ld -o programme.exe file.o main.o)
programme.exe :
$10000000 E34A CD42
$10000004 DEAD BEEF
$10000008 1337 BABE
$1000000C CEE1 3429
OK !

Link Edit / Edition de Lien

```
gcc -o programme.exe file.o
ld -o programme.exe file.o
```

- ERREUR :

Pas de fonction "main" / aucun point d'entrée défini

```
gcc -o programme.exe file.o -lc --entry my_fun
ld -o programme.exe file.o -lc --entry my_fun
```

Résumé

- Ceci est valable pour tout fichier "**file.c**" contenant du C, et un fichier **main.c** préalablement compilé et assemblé contenant une fonction "**int main(void)**".

- L'option **-o** de GCC désigne "*output*", donc le fichier de sortie qui sera écrit.
  Par exemple :

```
gcc -c file_asm.s -o file.o
```

- GCC prend en entrée **file_asm.s**, il doit effectuer une compilation d'après le paramètre **-c**, et il écrira en sortie dans le fichier **file.o**.

- De très très très nombreuses options à GCC existent, vous pouvez les consulter en tapant dans votre terminal :

```
man gcc
```

- Pré-Processeur :

```
gcc –E –P file.c –o file_out.c
```

- Compilation :

```
gcc –S file_out.c –o file_asm.s
```

- Assemblage :

```
gcc –c file_asm.s –o file.o
```

OU

```
as file_asm.s –o file.o
```

- Edition de liens :

```
gcc –o programme.exe file.o main.o
```

OU

```
ld –o programme.exe file.o main.o –lc –entry main
```

Makefile

- Les lignes de commande pour compiler sont longues.
  Comment accélérer cela ?

- En automatisant les lignes de compilation avec des outils spéciaux.
  Exemple :
  - script shell générant les paramètres utiles (architecture cible, flags, ...)
  - Makefile qui regroupe les noms de fichiers et les lignes de commande
  - génération automatique de Makefiles
  - génération automatique de packages contenant les programmes
  - ...

- Plus loin encore pour développer :
  - écriture automatique du code à partir de modèles graphiques
  - pas d'architecture cible, mais une "VM" dédiée au code (bytecode)
  - ...

tabulation

```
all:
	make preproc
	make preassemble
	make assemble
	make linkedit

preproc:
	gcc -E -P file.c -o file_out.c

preassemble:
	gcc -S file_out.c -o file_asm.s

assemble:
	gcc -c file_asm.s -o file.o

linkedit:
	gcc -c main.c -o main.o
	gcc -o programme.exe file.o main.o

help:
	echo 'TARGET: preproc preassemble assemble linkedit'

clean:
	rm -f *~

distclean: clean
	rm -f file_out.c
	rm -f file_asm.s
	rm -f file.o
	rm -f main.o
	rm -f programme.exe
```

cible(s) à exécuter avant de lancer les commandes

Makefile

- Cibles définies :
  all, preproc, preassemble, assemble, linkedit, help, clean, distclean
  (attention : tabulation nécessaire au début des lignes de commandes)
  (distclean appelle d'abord la cible clean avant de s'exécuter)

- Appel des cibles (dans shell) :
  make help
  make distclean
  make preproc
  make
  make clean

- Possibilité de définir des variables, ne pas recompiler ce qui a déjà été compilé, etc...

Interpréteur de Commandes / Scripts

- La compilation transforme un code source (langage haut niveau) en instructions machines (bas niveau)
- Les instructions seront exécutées par le processeur lorsque le programme sera lancé

- Qu'est-ce que l'interprétation ?
- Qu'est-ce qu'un interpréteur de commandes ?

Interpréteur de Commandes / Scripts

Qu'est-ce que l'interprétation ?

- Les scripts ne sont pas « compilés »
- Ils sont lus par un programme qui « interprète » les mots/lignes du script, et exécute des opérations qu'il comprend

Interpréteur de Commandes / Scripts

Qu’est-ce qu’un interpréteur de commandes ?

- Le programme qui lit les scripts et interprète les mots/lignes pour les exécuter est appelé un « interpréteur de commandes »
- Pour chaque langage de script, il existe un ou des interpréteurs

Interpréteur de Commandes / Scripts

- Langages interprétés célèbres :
  - Les Shells (sh, bash, tcsh, …)
  - Perl
  - Python
  - PHP
  - JavaScript
  - tcl/tk
  - m4, Make
  - Ruby
  - OCaml
  - R

Scripts & ByteCode

- Interpréteurs connus produisant du bytecode :
  - Perl (.pl script, programme exécutable)
  - Python (.py script, .pyc bytecode)
  - OCaml (.ml source, .cmo objet bytecode, .opt programme natif)
  - R
  - Tcl

Scripts & ByteCode

Qu’est-ce que le ByteCode ?

- Le ByteCode est un « langage machine » pour une machine virtuelle
- Les instructions ByteCode seront exécutées par un programme au lieu du processeur directement (il y a donc une légère surcouche entre le processeur et le ByteCode)

Scripts & ByteCode

Qu’est-ce que le ByteCode ?

- Les « machines virtuelles » qui exécutent le ByteCode sont des programmes qui prennent souvent comme noms :
  - Virtual Machine
  - Runtime Environment

Interpréteur de Commandes / Scripts

- Langages générant du ByteCode :
  - Perl
  - Python
  - OCaml
  - R
  - Tcl
  - Java (JRE ou JVM)
  - C# (CLR – Common Language Runtime)
  - Clang/LLVM

Interpréteur de Commandes / Scripts

Qu'est-ce que le « JIT » ?

- « Just In-Time » ou « dynamic translation »
- Compilation à la volée/lors de l'exécution
- Un programme analyse le code et décide de compiler ce qui est nécessaire, ou d'interpréter ce qui est suffisant
- Stratégie entre l'interprétation et la compilation
- Utilise un programme similaire aux VM/RE