# Informatique Embarquée
# M2 / 2017

# Chaîne de compilation, Compilation croisée, ELF...

# Chaîne de production

- Ensemble d'outils permettant de passer du code source au code binaire machine exécutable (ou dans certains cas, interprétable)
  - Compilateur
  - Assembleur
  - Éditeur de liens (*linker*)
  - Chargeur (*loader*)
- Pour un ou plusieurs (beaucoup de fichiers sources)

# Makefile

- Automatise la production
  - (Re)compile les seuls fichiers sources
    - Qui ont été modifiés depuis la dernière compilation
    - Ou qui dépendent de fichiers qui ont été modifiés depuis la dernière compilation
- Permet de se simplifier la vie en y mettant les règles de production nécessaires au projet
- Fichier `Makefile` ou `makefile` dans le répertoire courant

# Exemple : hello (sans makefile)

```
fa$ touch hello.c
fa$ make hello
cc     -o hello hello.c
fa$ rm hello
fa$ make hello
cc     -o hello hello.c
fa$ make hello
`hello' is up to date.
fa$ touch hello.c
fa$ make CC=gcc CFLAGS=-Wall hello
gcc -Wall    -o hello hello.c
fa$
```

# Makefile

- Un Makefile contient des règles du type :

```
cible: dépendance1 dépendance2...
tabulation commande1
tabulation commande2
tabulation ...
```

# Makefile colimacon

```
all: colimacon

colimacon: main.o libcoli.o
    gcc -o colimacon main.o libcoli.o

main.o : main.c coli.h
    gcc -c main.c -Wall

libcoli.o : libcoli.c coli.h
    gcc -c libcoli.c -Wall
```

# Makefile autres cibles

```
.PHONY: clean mrproper

clean:
	rm *~ *.o # et d'autres encore !

mrproper: clean
	rm colimacon

# on peut aussi mettre des cibles autres

pkg: mrproper
	tar -zcvf ../colimacon.tgz .
```

# Makefile : variables

- Il existe des variables prédéfinies
  - CC : le compilateur
  - CFLAGS : les flags de compilation
  - LD : l'éditeur de liens
  - LDFLAGS : flags pour l'éditeur de liens
- Mais aussi :
  - $@ : la cible
  - $^ : les dépendances
  - $< : la première dépendance
  - $* : le nom de la cible sans suffixe

# Makefile colimacon

```
CC=gcc
CFLAGS=-Wall -ansi
LDFLAGS=
EXEC=colimacon

all: $(EXEC)

colimacon: main.o libcoli.o
    $(CC) -o $@ $^ $(LDFLAGS)

main.o : main.c coli.h
    $(CC) -c $< $(CFLAGS)

libcoli.o : libcoli.c coli.h
    $(CC) -c $< $(CFLAGS)
```

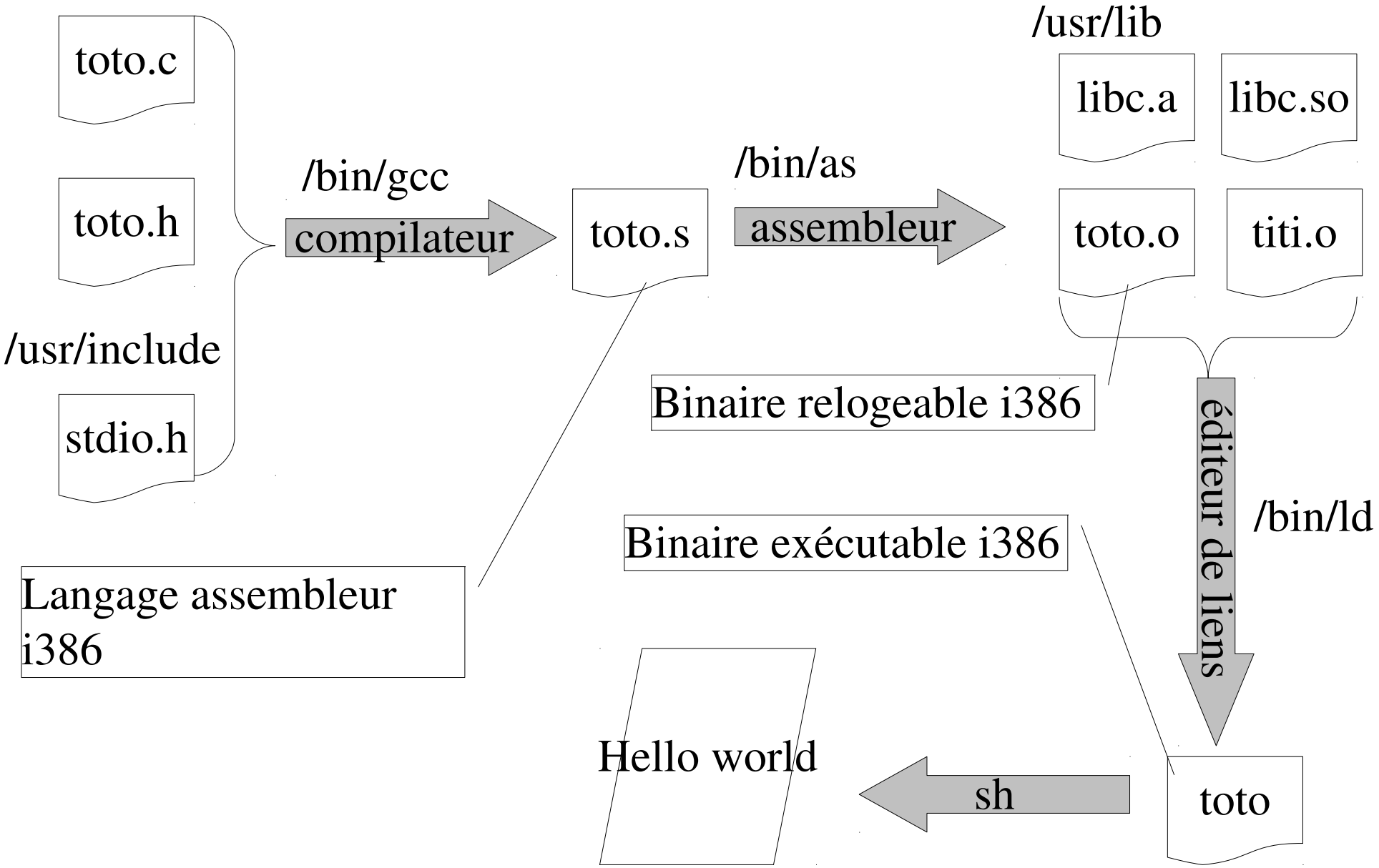
# Makefile règles

```
CC=gcc
CFLAGS=-Wall -ansi
LDFLAGS=
EXEC=colimacon

all: $(EXEC)

colimacon: main.o libcoli.o
    $(CC) -o $@ $^ $(LDFLAGS)

%.o: %.c
    $(CC) -o $@ -c $< $(CFLAGS)
```

# Makefile règles

```
CC=gcc
CFLAGS=-Wall -ansi
LDFLAGS=
EXEC=colimacon
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)

all: $(EXEC)

colimacon: $(OBJ)
    $(CC) -o $@ $^ $(LDFLAGS)

%.o: %.c
    $(CC) -o $@ -c $< $(CFLAGS)
```

# Compilation native sur i386



# Exemple: hello.c

```
#include <stdio.h>
void foo()
{
  printf("Hello World!\n");
}
int main()
{
  foo(); /* Appel procédure locale */
  pause(); /* pour voir /proc...*/
}
```

# Obtenir le fichier .s

```
#gcc -s hello.c  # crée hello.s

foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    $.LC0, (%esp)
        call    puts
        leave
        ret
```

# Obtenir le fichier .s (suite)

```
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $4, %esp
        call    foo
        call    pause
        addl    $4, %esp
        .....
        ret
```

# Construction hello suite

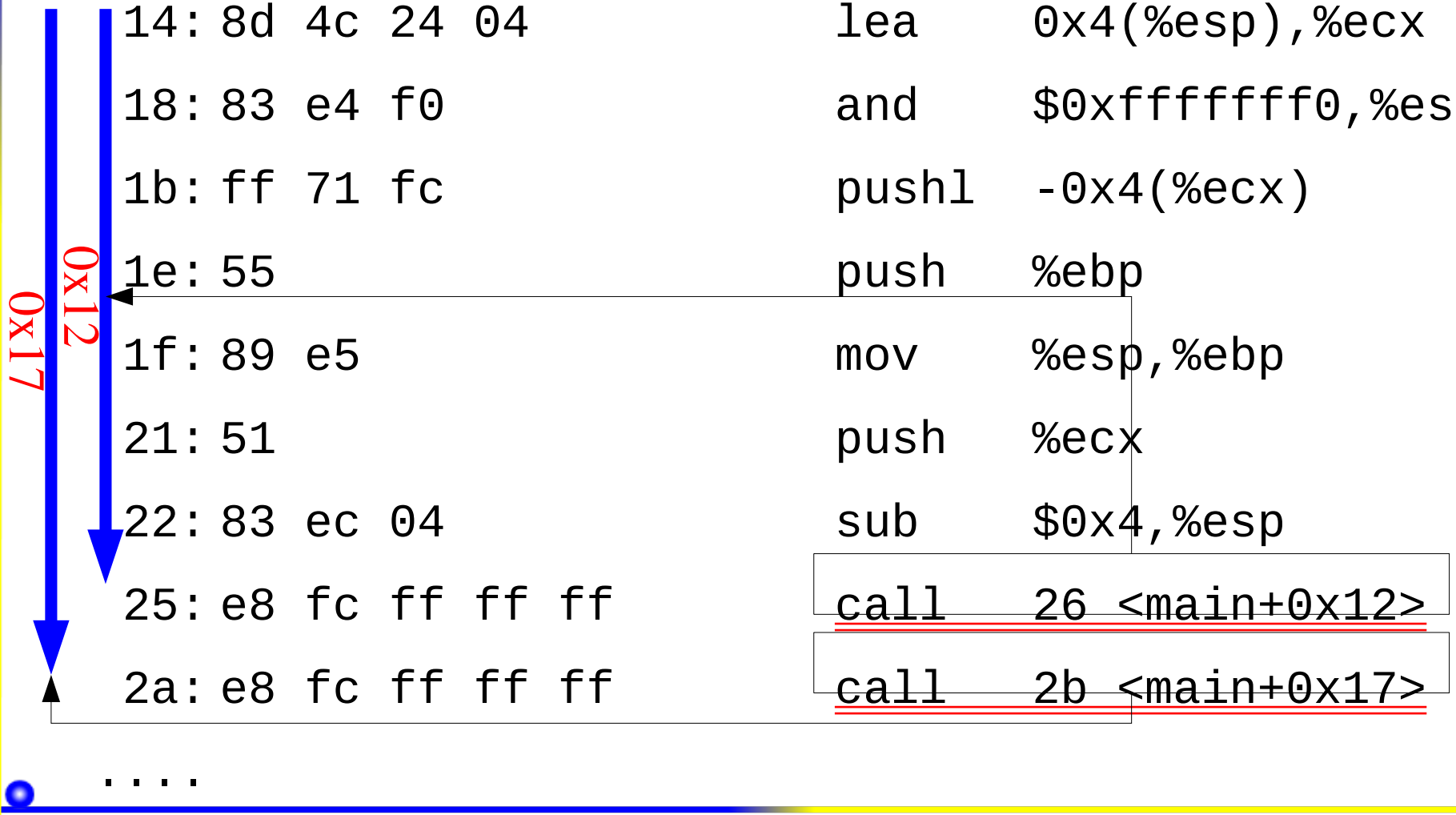
```
#as -o hello.o hello.s
#gcc -o hello hello.o

##Désassembler! => commande objdump
#objdump -D hello.o > hello_o.dis
#objdump -D hello > hello.dis
```

# hello.o désassemblé!

```
00000000 <foo>:
   0:   55                                  push   %ebp
   1:   89 e5                               mov    %esp,%ebp
   3:   83 ec 08                            sub    $0x8,%esp
   6:   c7 04 24 00 00 00 00                movl   $0x0,(%esp)
   d:   e8 fc ff ff ff                      call   e <foo+0xe>
  12:   c9                                  leave
  13:   c3                                  ret
```

# hello.o désassemblé! (suite)

```
00000014 <main>:
  14: 8d 4c 24 04          lea    0x4(%esp),%ecx
  18: 83 e4 f0             and    $0xfffffff0,%esp
  1b: ff 71 fc             pushl  -0x4(%ecx)
  1e: 55                   push   %ebp
  1f: 89 e5                mov    %esp,%ebp
  21: 51                   push   %ecx
  22: 83 ec 04             sub    $0x4,%esp
  25: e8 fc ff ff ff       call   26 <main+0x12>
  2a: e8 fc ff ff ff       call   2b <main+0x17>
  ....
```



# Hello.o « relocation »

```
#objdump -r hello.o > hello_o.reloc
```

```
hello.o:     file format elf32-i386

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000009 R_386_32          .rodata
0000000e R_386_PC32        puts
00000026 R_386_PC32        foo
0000002b R_386_PC32        pause
```

# Hello désassemblé!

```
080483d4 <foo>:
 80483d4:  55                          push   %ebp
 80483d5:  89 e5                       mov    %esp,%ebp
 80483d7:  83 ec 08                    sub    $0x8,%esp
 80483da:  c7 04 24 d0 84 04 08        movl   $0x80484d0,(%esp)
 80483e1:  e8 22 ff ff ff              call   8048308 <puts@plt>
 80483e6:  c9                          leave
 80483e7:  c3                          ret
```

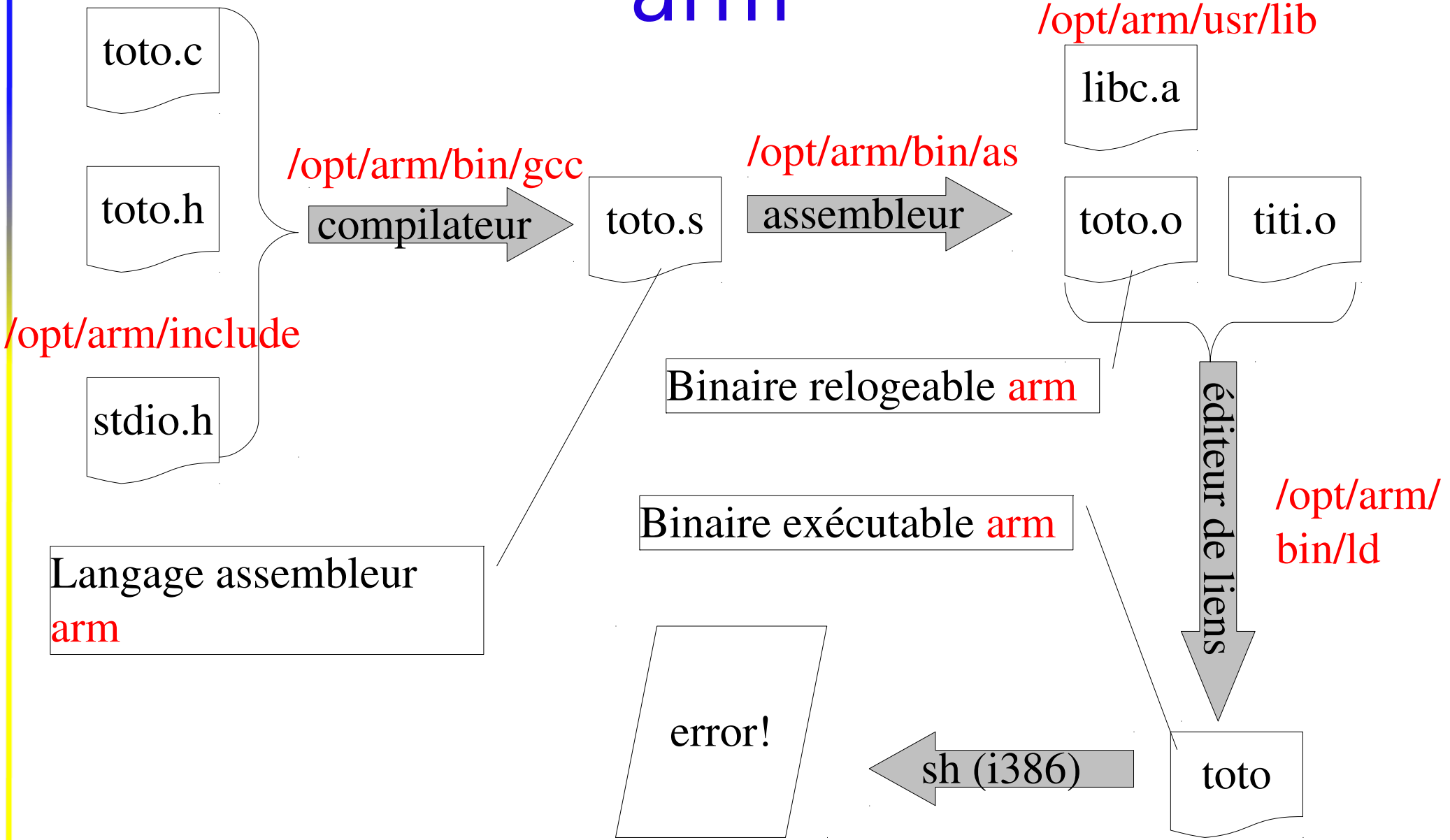
# Hello désassemblé! (suite)

```
080483e8 <main>:
 80483e8:  8d 4c 24 04          lea    0x4(%esp),%ecx
 80483ec:  83 e4 f0             and    $0xfffffff0,%esp
 80483ef:  ff 71 fc             pushl  -0x4(%ecx)
 80483f2:  55                   push   %ebp
 80483f3:  89 e5                mov    %esp,%ebp
 80483f5:  51                   push   %ecx
 80483f6:  83 ec 04             sub    $0x4,%esp
 80483f9:  e8 d6 ff ff ff       call   80483d4 <foo>
 80483fe:  e8 f5 fe ff ff       call   80482f8 <pause@plt>
```

# « map » hello

```
08048000-08049000 r-xp 00000000 03:01 278109     /home/debian/TP/hello
08049000-0804a000 rw-p 00000000 03:01 278109     /home/debian/TP/hello
b7e0b000-b7e0c000 rw-p b7e0b000 00:00 0
b7e0c000-b7f61000 r-xp 00000000 03:01 833389     /lib/.../libc-2.7.so
b7f61000-b7f62000 r--p 00155000 03:01 833389     /lib/.../libc-2.7.so
b7f62000-b7f64000 rw-p 00156000 03:01 833389     /lib/.../libc-2.7.so
b7f64000-b7f67000 rw-p b7f64000 00:00 0
b7f72000-b7f75000 rw-p b7f72000 00:00 0
b7f75000-b7f76000 r-xp b7f75000 00:00 0          [vdso]
b7f76000-b7f90000 r-xp 00000000 03:01 824174     /lib/ld-2.7.so
b7f90000-b7f92000 rw-p 0001a000 03:01 824174     /lib/ld-2.7.so
bfb7c000-bfb91000 rw-p bffeb000 00:00 0          [stack]
```

# Compilation croisée i386 => arm



# Compilation, symboles

hello.c

```
main()
{
printf("Hello\n");
}
```

cc

hello.s

```
.LC0:
        .string "Hello\n"
        .text
.globl main
        .type   main,@function
main:
        ....
        subl    $12, %esp
        pushl   $.LC0
        call    printf
         .....
```

as

hello.o

nm

```
00000000 T main
         U printf
```

# Librairies statiques

```
# cc hello.c -o hello -Wl,-static
# file hello
hello: ELF 32-bit LSB executable, Intel
  80386, version 1 (SYSV), for GNU/Linux
  2.2.5, statically linked, not stripped
```

- Utilise libc.a

```
# size hello
text    data bss  dec    hex filename
365785 3820 315  372761 5b019 hello
```

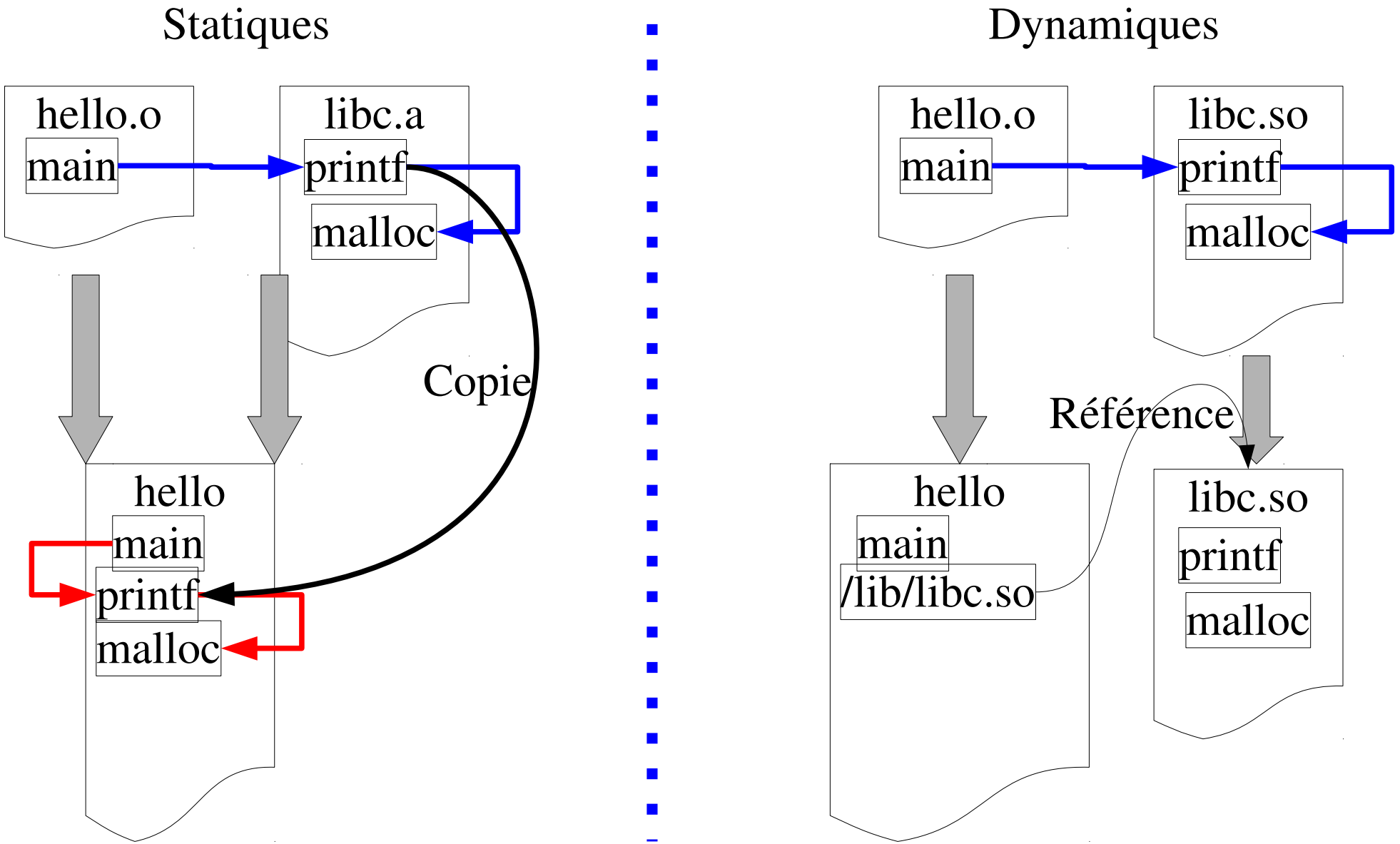
# Librairies dynamiques

```
# cc hello.c -o hello
# file hello
hello: ELF 32-bit LSB executable,
  Intel 80386, version 1 (SYSV), for
  GNU/Linux 2.2.5, dynamically linked
  (uses shared libs), not stripped
```

- Utilise libc.so

```
# size hello
text   data    bss    dec    hex filename
818    256     4     1078    436 hello
```

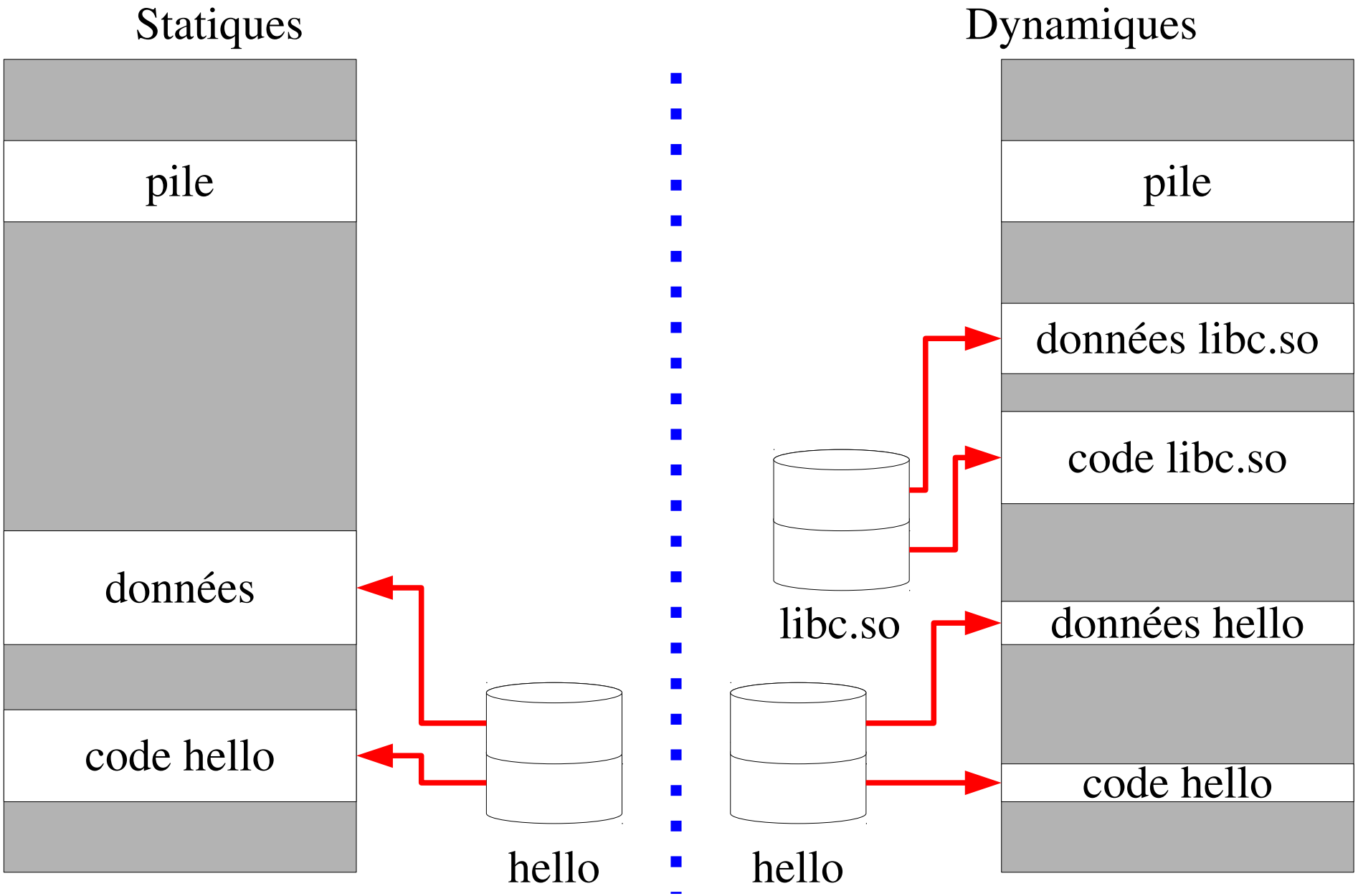
# Librairies


Statiques
hello.o
main
libc.a
printf
malloc
Copie
hello
main
printf
malloc
Dynamiques
hello.o
main
libc.so
printf
malloc
Référence
hello
main
/lib/libc.so
libc.so
printf
malloc

# Librairies

- Librairies statiques
  - Duplique le code des librairies dans chaque exécutable, disque et mémoire
  - Taille sur disque (de(s) l'exécutable(s)) plus importante
  - Mais pas de dépendances, exécutable "autonome"
  - Chargement en mémoire: simple
- Librairies dynamiques
  - Ne duplique pas le code des librairies (ni sur disque ni en mémoire)
  - Taille sur disque (de l'exécutable) moins importante, mais nécessité d'avoir la librairie
  - Dépendances entre exécutables et librairies
  - Chargement en mémoire plus complexe: besoin d'un "loader"
  - Bibliothèques compilées avec "Position Independent Code"

# Librairies


Statiques
Dynamiques
pile
données
code hello
hello
pile
données libc.so
code libc.so
libc.so
données hello
code hello
hello

# Construire / utiliser une librairie statique

- Générer (compiler) tous les « .o » qui doivent être inclus dans la librairie

```
fa$ ar -cvq libcoli.a coli1.o coli2.o
fa$ gcc -o col main.c libcoli.a
fa$ mkdir lib ; mv libcoli.a lib
fa$ gcc -o t1 tst.c -Llib -lcoli
```

# Construire / utiliser une librairie dynamique

- Générer (compiler) tous les « .o » qui doivent être inclus dans la librairie

```
fa$ gcc -fPIC -c *.c
fa$ gcc -shared -Wl,-soname,libcoli.so.1
        -o libcoli.so.1.0 *.o

fa$ mv libctest.so.1.0 ./lib
fa$ ln -sf lib/libctest.so.1.0
           lib/libctest.so.1
fa$ ln -sf lib/libctest.so.1.0
           lib/libctest.so

fa$ gcc -o t1 tst.c -Llib -lcoli
```

# Taille bibliothèque dynamique

```
#more /etc/redhat-release
Red Hat Enterprise Linux ES release 3 (Taroon)
# ls -l libc.so.6
-rwxr-xr-x 1 root root 1564956 Oct  3  2003 libc.so.6
# size libc.so.6
text    data    bss   dec       hex    filename
1266468 11248   10788 1288504   13a938 libc.so.6
# size /lib/ld-2.3.2.so
text   data     bss   dec     hex   filename
85574  1332     532   87438   1558e /lib/ld-2.3.2.so
# file /lib/ld-2.3.2.so
/lib/ld-2.3.2.so: ELF 32-bit LSB shared object, Intel
 80386, version 1 (SYSV), not stripped
```

# Position Independent Code

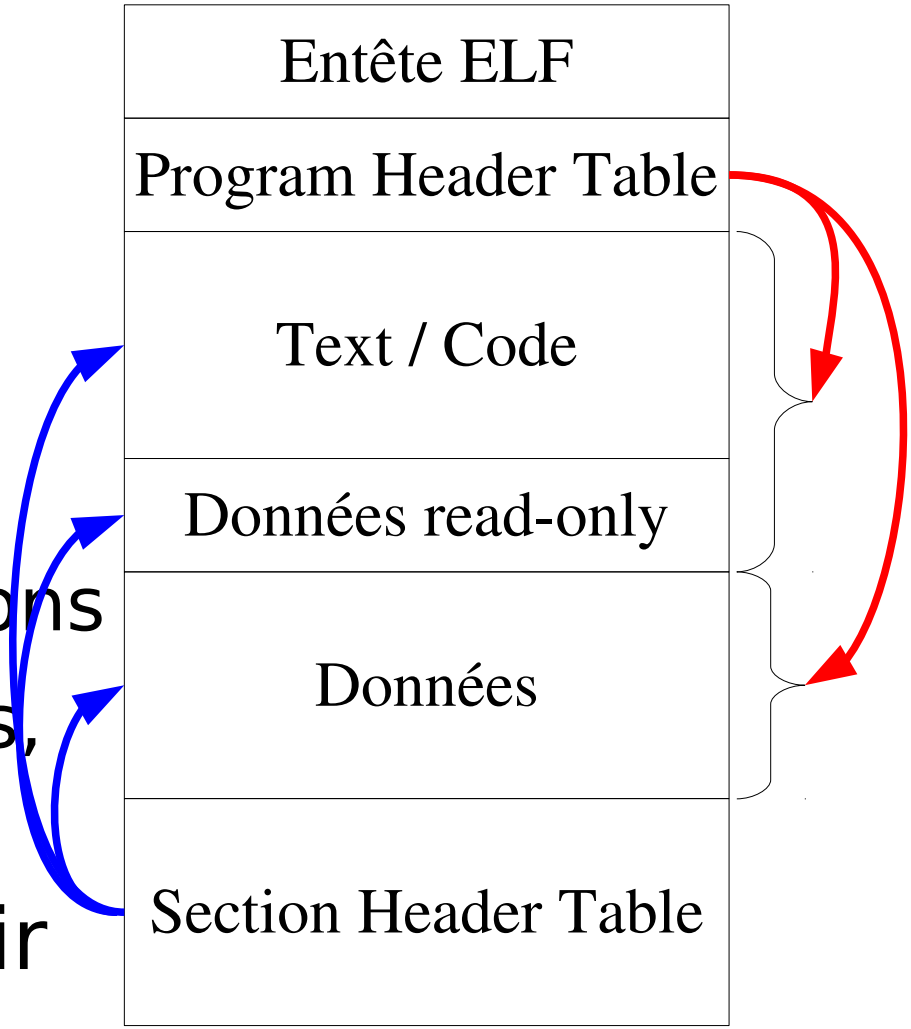
- Permet de placer le code où bon nous semble en mémoire (physique mais aussi virtuelle)
- Appels de procédures par
  - Déplacement relatif à l'instruction courante
  - Par table d'indirection:
    - Les adresses des fonctions sont calculées au chargement de la bibliothèque (pour chaque programme)
- Accès aux données:
  - Par table d'indirection (GOT: Global Offset Table)
- Calcul au chargement ou à la demande (lazy)
  - Impact sur le déterminisme

# Fichiers ELF

- ELF: Executable and Linking Format
- Les fichiers binaires utilisent généralement un format baptisé ELF (imposé par LSB)
  - Binaires relogeables (`.o`)
  - Binaires éxéutables
  - Librairies partagées (`.so`)
  - Core dump
- Largement utilisé (y compris sur téléphones portables cf wikipedia)

# Fichiers ELF

- Program Header Table
  - Décrit un ou plusieurs segments
  - Utile (à run-time) pour le chargement
- Section Header Table
  - Décrit une ou +eurs sections
  - Utile pour éditions de liens, relocations
- Un segment peut contenir +eurs sections



# Manipulation ELF

- Commandes:
  - objdump
  - readelf
- Librairie de manipulation
  - Libelf (FreeBSD)
  - Linux?

# readelf -h (a.out / statique)

```
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:               0x8048100
  Start of program headers:          52 (bytes into file)
  Start of section headers:          373088 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         4
  Size of section headers:           40 (bytes)
  Number of section headers:         21
  Section header string table index: 18
```

# Program Headers

```
Program Headers:
Type   Offset   VirtAddr   PhysAddr   FileSiz MemSiz
  Flg Align
LOAD   0x000000 0x08048000 0x08048000 0x595e8 0x595e8
  R E 0x1000
LOAD   0x05a000 0x080a2000 0x080a2000 0x00eec 0x01b54
  RW  0x1000
NOTE   0x0000b4 0x080480b4 0x080480b4 0x00020 0x00020
  R   0x4
STACK  0x000000 0x00000000 0x00000000 0x00000 0x00000
  RW  0x4
```

# ELF: sections

```
Section Headers:
[Nr] Name
  [ 0]
  [ 1] .init
  [ 2] .text
  [ 3] __libc_freeres_fn
  [ 4] .fini
  [ 5] .rodata
  [ 6] __libc_subfreeres
  [ 7] __libc_atexit
  [ 8] .eh_frame
  [ 9] .data
  [10] .ctors
  [11] .dtors
```

```
[Nr] Name
  [12] .jcr
  [13] .got
  [14] .bss
  [15] __libc_freeres_pt
  [16] .comment
  [17] .note.ABI-tag
  [18] .shstrtab
  [19] .symtab
  [20] .strtab
```

# ELF: segments / sections

```
Section to Segment mapping:

  Segment Sections...
00 .init .text __libc_freeres_fn .fini .rodata
   __libc_subfreeres __libc_atexit .eh_frame
   .note.ABI-tag


01 .data .ctors .dtors .jcr .got .bss
    __libc_freeres_ptrs


02 .note.ABI-tag
```

# ELF: volume méta-données

```
# size hello_stat
text    data bss  dec     hex  filename
365849 3820 3156 372825 5b059 hello
# ls -l hello_stat
-rwxr-x---      412380  hello
```

=> delta: 39 555

```
# size hello_dyn
text  data  bss  dec      hex filename
818   256   4    1078     436 hello
# ls -l hello_dyn
-rwxr-x---        4662  hello
```

=> delta: 3 584