Béatrice Creusillet <bcreusillet@quarkslab.com>
Juan Manuel Martinez <jmmartinez@quarkslab.com>

Compilation pour le reverseur

Table of Contents

Pré-requis

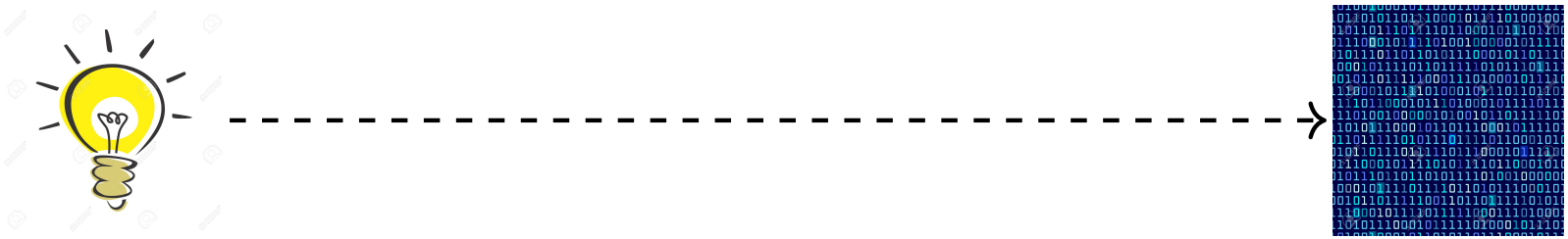
- Linux avec gcc, clang, ida (version gratuite)
- ou Machine virtuelle:
  - Vagrant
  - VirtualBox
  - Fichier Vagrantfile envoyé précédemment
  - `vagrant up`
  - quitter la VM
  - `vagrant up`
  - Si ida n'est pas chargé: `./idafree70_linux.run --mode unattended`
  - En cas de pb de clavier, dans un terminal de la VM

    `sudo dpkg-reconfigure keyboard-configuration`
- TDs: `git clone https://github.com/bac-qb/compil.git`

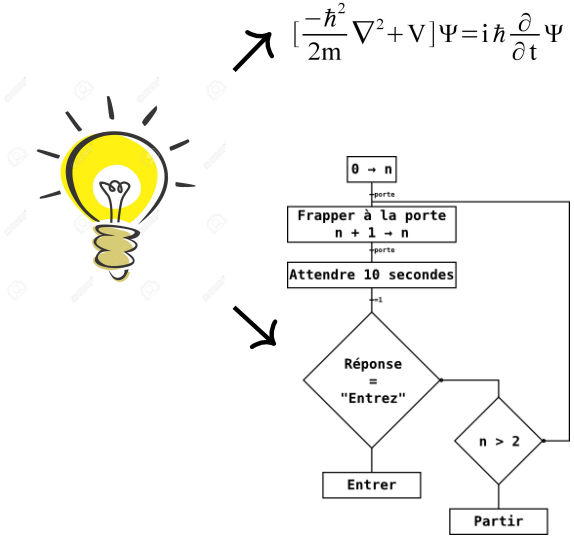
Table of Contents

Table of Contents

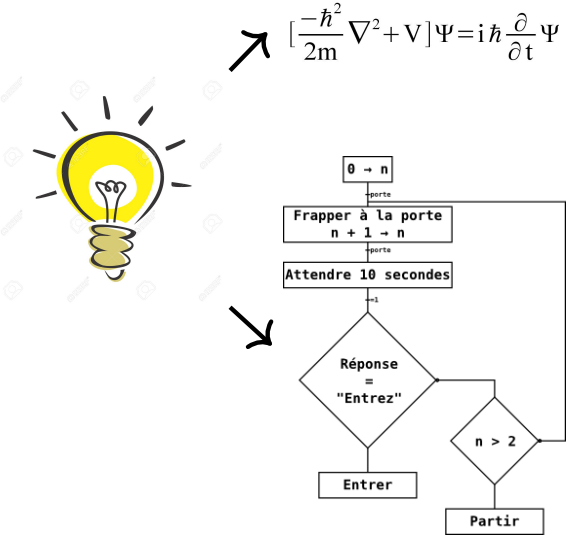
Comment peut-on être binaire ?

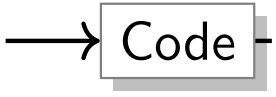


Comment peut-on être binaire ?

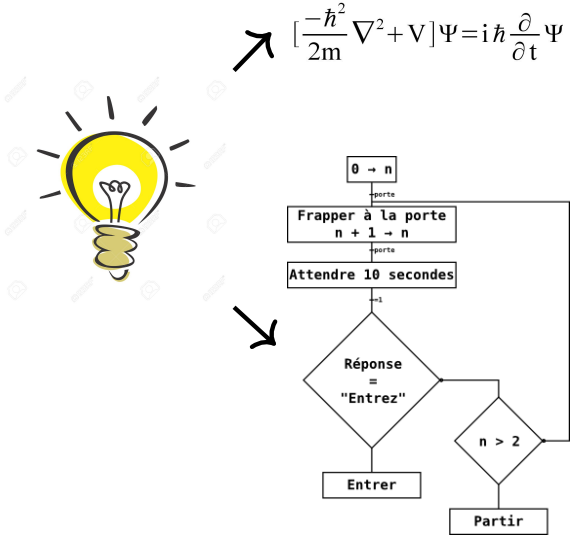


Comment peut-on être binaire ?





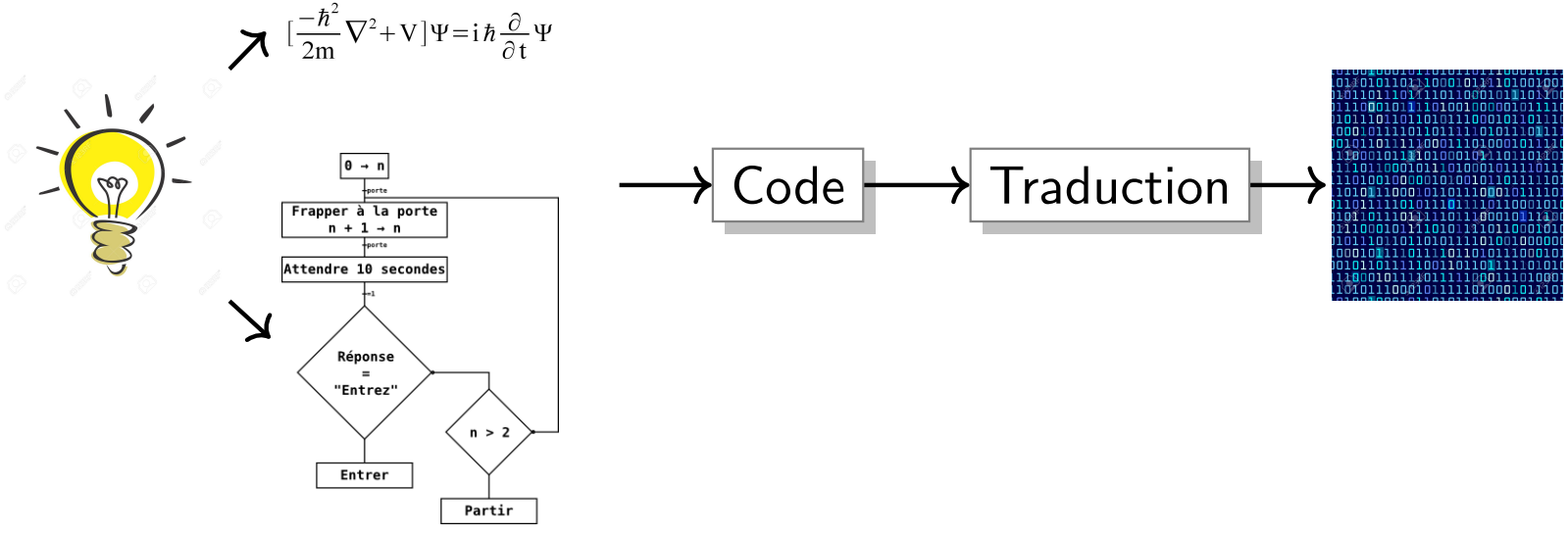
Comment peut-on être binaire ?



→ Code → Traduction →

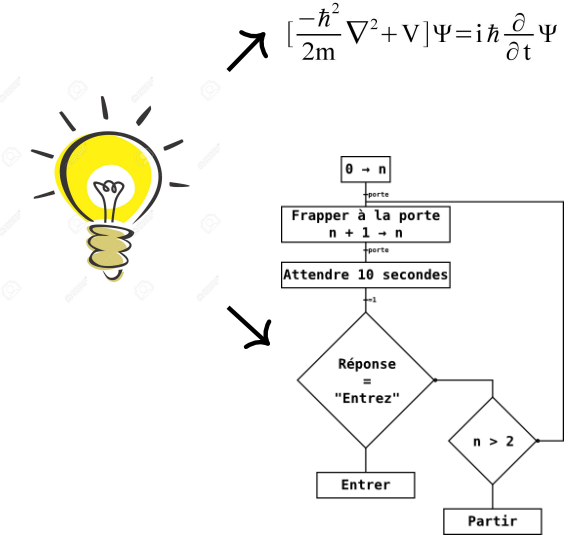
Comment peut-on être binaire ?

Spécialisation



Comment peut-on être binaire ?

Spécialisation



→ Code → Traduction →

Reversing
Généralisation

Table of Contents

Un gradient d'approches

- Compilation : traduction statique en binaire
  - réutilisable
  - code unique qui s'adapte à des inputs variés
  - code adapté à un environnement particulier (windows, linux, osx,...)
  - ex: C, C++, Fortran

Un gradient d'approches

- Compilation : traduction statique en binaire
  - réutilisable
  - code unique qui s'adapte à des inputs variés
  - code adapté à un environnement particulier (windows, linux, osx,...)
  - ex: C, C++, Fortran

- Interprétation : traduction dynamique = à la volée
  - utilisation unique
  - code optimisé pour des inputs particuliers
  - ex: Basic, R, bash

Un gradient d'approches

- Compilation : traduction statique en binaire
  - réutilisable
  - code unique qui s'adapte à des inputs variés
  - code adapté à un environnement particulier (windows, linux, osx,...)
  - ex: C, C++, Fortran
- Pré-compilation : traduction statique vers un 'byte-code'
  - byte-code générique
  - adaptation à l'environnement au run-time
  - ex: Java

- Interprétation : traduction dynamique = à la volée
  - utilisation unique
  - code optimisé pour des inputs particuliers
  - ex: Basic, R, bash

Un gradient d'approches

- Compilation : traduction statique en binaire
  - réutilisable
  - code unique qui s'adapte à des inputs variés
  - code adapté à un environnement particulier (windows, linux, osx,...)
  - ex: C, C++, Fortran
- Pré-compilation : traduction statique vers un 'byte-code'
  - byte-code générique
  - adaptation à l'environnement au run-time
  - ex: Java
- JIT : Just-in-Time compilation
  - traduction dynamique partiellement 'réutilisable'
  - compilation dynamique, parfois limitée aux hot spots
  - ex: Matlab, Julia, Java
- Interprétation : traduction dynamique = à la volée
  - utilisation unique
  - code optimisé pour des inputs particuliers
  - ex: Basic, R, bash

Compilateurs C/C++

Compilateurs commerciaux

- Intel C/C++ compiler (icc) (Windows, linux, MacOs, Intel)
- Visual Studio C++ (Windows, Microsoft)
- IAR C/C++ compiler (Windows, Linux, others, IAR Systems)
- Edison Design Group (Windows, Linux, others, EDG)
- VisualAge C++, XL C/C++ (Windows, LInux, Aix, OS/2, OS/400,... IBM)
- ...

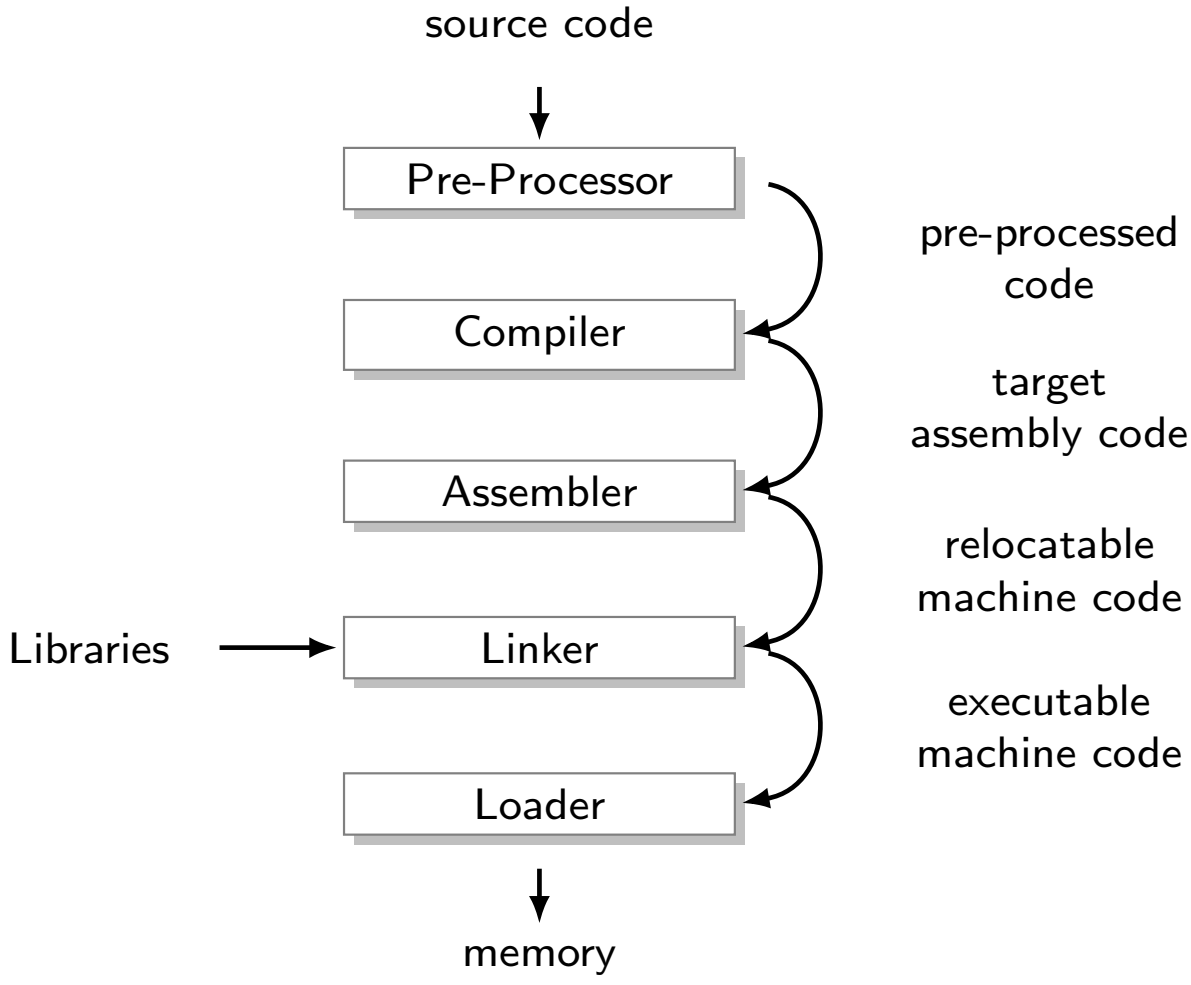
OpenSource ou Freeware

- GNU C/C++ (**gcc**, MinGW) (Windows, linux, MacOS, GNU Project)
- Clang/LLVM (**clang**) (Windows, linux, MacOS, LLVM Project)
- ...

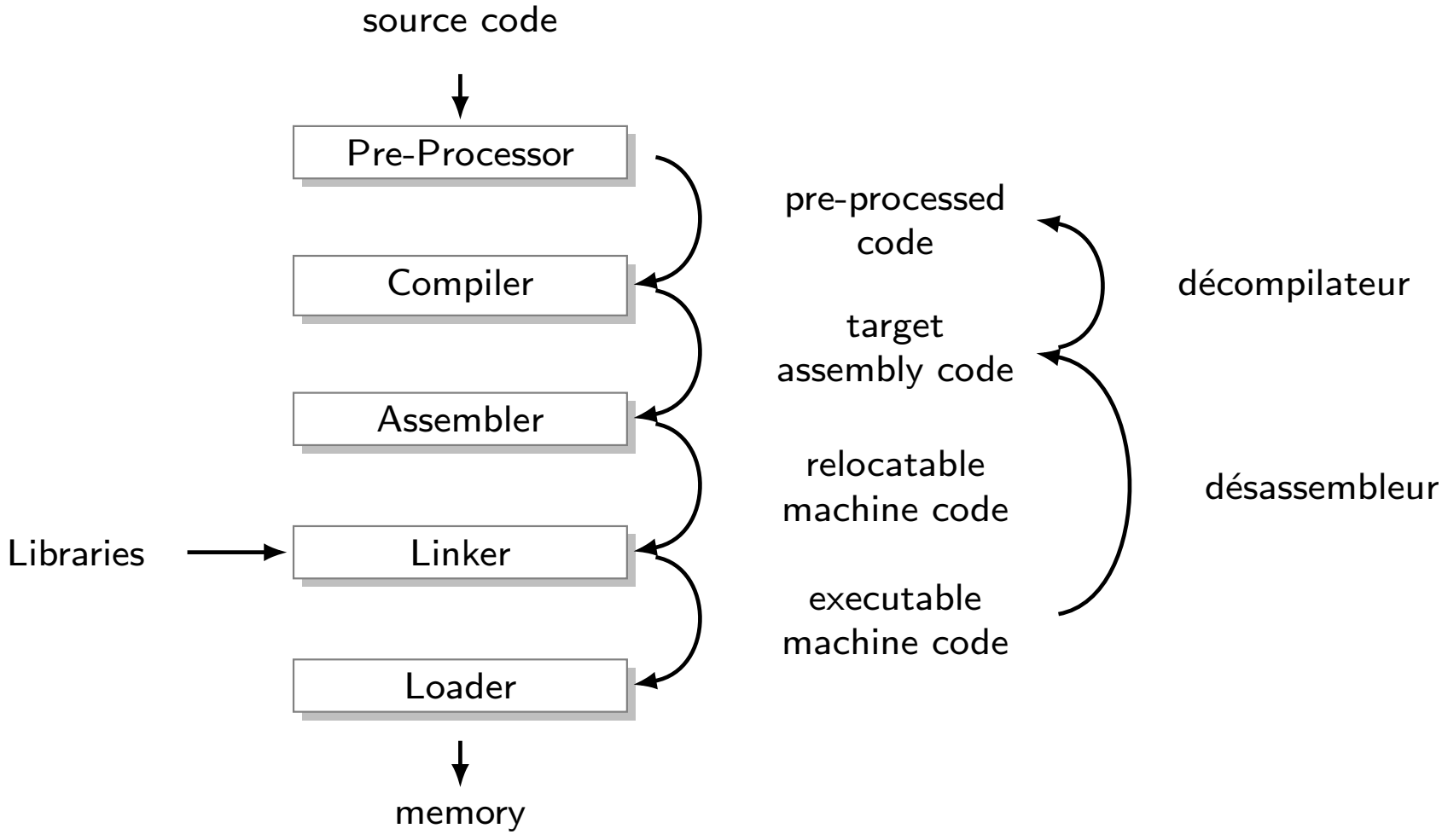
https://en.wikipedia.org/wiki/List_of_compilers

Table of Contents

Une traduction en plusieurs étapes



Une traduction en plusieurs étapes



TD 1 : les étapes de la génération de binaire

welcome.c :

```c
#include <stdio.h>
void welcome(char const* nom) {
  printf("welcome %s!\n", nom);
}
```

main_welcome.c :

```c
#ifndef DEFAULT
#define DEFAULT "everybody"
#endif

extern void welcome(char const*);

int main(int argc, char const* argv[]) {
  if      (argc == 1) { welcome(DEFAULT); }
  else if (argc == 2) { welcome(argv[1]); }
  else { return 1; }
  return 0;
}
```

td1 : exécution

Version par défaut :

```
clang welcome.c main_welcome.c
./a.out
./a.out alice
```

td1 : exécution

Version par défaut :

```
clang welcome.c main_welcome.c
./a.out
./a.out alice
```

```
welcome everybody!
welcome alice!
```

td1 : exécution

Version par défaut :

```
clang welcome.c main_welcome.c
./a.out
./a.out alice
```

```
welcome everybody!
welcome alice!
```

Version avec définition de variable de pré-processing :

```
clang welcome.c main_welcome.c -DDEFAULT='"bob"'
./a.out
./a.out alice
```

td1 : exécution

Version par défaut :

```
clang welcome.c main_welcome.c
./a.out
./a.out alice
```

```
welcome everybody!
welcome alice!
```

Version avec définition de variable de pré-processing :

```
clang welcome.c main_welcome.c -DDEFAULT='"bob"'
./a.out
./a.out alice
```

```
welcome bob!
welcome alice!
```

Le pré-processeur : traitement des macros

```
1  clang -E main_welcome.c
```

Le pré-processeur : traitement des macros

```
clang -E main_welcome.c
```

```
int main(int argc, char const* argv[]) {
  if (argc == 1) { welcome("everybody"); }
  else if (argc == 2) { welcome(argv[1]); }
  else { return 1; }
  return 0;
}
```

Le pré-processeur : traitement des macros

```
clang -E main_welcome.c
```

```
int main(int argc, char const* argv[]) {
  if (argc == 1) { welcome("everybody"); }
  else if (argc == 2) { welcome(argv[1]); }
  else { return 1; }
  return 0;
}
```

```
clang -E main_welcome.c -DDEFAULT='"bob"'
```

Le pré-processeur : traitement des macros

```
clang -E main_welcome.c
```

```
int main(int argc, char const* argv[]) {
  if (argc == 1) { welcome("everybody"); }
  else if (argc == 2) { welcome(argv[1]); }
  else { return 1; }
  return 0;
}
```

```
clang -E main_welcome.c -DDEFAULT='"bob"'
```

```
int main(int argc, char const* argv[]) {
  if (argc == 1) { welcome("bob"); }
  else if (argc == 2) { welcome(argv[1]); }
  else { return 1; }
  return 0;
}
```

Le pré-processeur : inclusion de fichiers *header*

```
1  clang -E welcome.c | wc -l
```

```
1  797
```

Le pré-processeur : inclusion de fichiers *header*

```
1  clang -E welcome.c | wc -l
```

```
1  797
```

```
1  clang -E welcome.c
```

Le pré-processeur : inclusion de fichiers *header*

```
1  clang -E welcome.c | wc -l
```

```
1  797
```

```
1  clang -E welcome.c
```

- les fichiers sont intégralement inclus
- attention à l'expansion de macros !

Le compilateur : génération de code assembleur

```
1  clang -S main_welcome.c
2  more main_welcome.s
```

L'assemblage: génération de code objet

```
1  as main_welcome.s -o main_welcome.o
2  file main_welcome.o
3  nm main_welcome.o
```

L'assemblage: génération de code objet

```
1  as main_welcome.s -o main_welcome.o
2  file main_welcome.o
3  nm main_welcome.o
```

```
1  main_welcome.o: ELF 64-bit LSB relocatable, x86-64,
       version 1 (SYSV), not stripped
2
3                      U _GLOBAL_OFFSET_TABLE_
4  0000000000000000 T main
5                      U welcome
```

L'assemblage: génération de code objet

```
1  as main_welcome.s -o main_welcome.o
2  file main_welcome.o
3  nm main_welcome.o
```

```
1  main_welcome.o: ELF 64-bit LSB relocatable, x86-64,
       version 1 (SYSV), not stripped
2
3                      U _GLOBAL_OFFSET_TABLE_
4  0000000000000000 T main
5                      U welcome
```

```
1  clang -c welcome.c -o welcome.o
2  file welcome.o
3  nm welcome.o
```

L'assemblage: génération de code objet

```
as main_welcome.s -o main_welcome.o
file main_welcome.o
nm main_welcome.o
```

```
main_welcome.o: ELF 64-bit LSB relocatable, x86-64,
    version 1 (SYSV), not stripped

                 U _GLOBAL_OFFSET_TABLE_
0000000000000000 T main
                 U welcome
```

```
clang -c welcome.c -o welcome.o
file welcome.o
nm welcome.o
```

```
welcome.o: ELF 64-bit LSB relocatable, x86-64, version
    1 (SYSV), not stripped

                 U _GLOBAL_OFFSET_TABLE_
                 U printf
0000000000000000 T welcome
```

L'édition de lien: génération de binaire

```
1  ld welcome.o main_welcome.o
```

L'édition de lien: génération de binaire

```
1  ld welcome.o main_welcome.o
```

```
1  ld : avertissement: le symbole d'entree _start est
       introuvable ; utilise par defaut 00000000004000b0
2  welcome.o: Dans la fonction « welcome » :
3  welcome.c:(.text+0x20) : reference indefinie vers «
       printf »
```

L'édition de lien: génération de binaire

```
1  ld welcome.o main_welcome.o
```

```
1  ld : avertissement: le symbole d'entree _start est
       introuvable ; utilise par defaut 00000000004000b0
2  welcome.o: Dans la fonction « welcome » :
3  welcome.c:(.text+0x20) : reference indefinie vers «
       printf »
```

```
1  ld welcome.o main_welcome.o -lc
2  file a.out
```

L'édition de lien: génération de binaire

```
1  ld welcome.o main_welcome.o
```

```
1  ld : avertissement: le symbole d'entree _start est
       introuvable ; utilise par defaut 00000000004000b0
2  welcome.o: Dans la fonction « welcome » :
3  welcome.c:(.text+0x20) : reference indefinie vers «
       printf »
```

```
1  ld welcome.o main_welcome.o -lc
2  file a.out
```

```
1
2  ld:  avertissement: le symbole d'entree _start est
       introuvable; utilise par defaut 00000000004002c0
3
4  a.out: ELF 64-bit LSB executable, x86-64, version 1 (
       SYSV), dynamically linked, interpreter /lib/ld64.so
       .1, not stripped
```

L'édition de lien: génération de binaire

```
1  clang -v welcome.o main_welcome.o 2>&1 | grep ld
```

L'édition de lien: génération de binaire

```
clang -v welcome.o main_welcome.o 2>&1 | grep ld
```

```
"/usr/bin/ld" -z relro --hash-style=gnu --eh-frame-hdr
     -m elf_x86_64 -dynamic-linker /lib64/ld-linux-x86
    -64.so.2 -o a.out /usr/bin/../lib/gcc/x86_64-linux
    -gnu/7.3.0/../../../x86_64-linux-gnu/crt1.o /usr/
    bin/../lib/gcc/x86_64-linux-gnu/7.3.0/../../../
    x86_64-linux-gnu/crti.o /usr/bin/../lib/gcc/x86_64
    -linux-gnu/7.3.0/crtbegin.o -L/usr/bin/../lib/gcc/
    x86_64-linux-gnu/7.3.0 -L/usr/bin/../lib/gcc/
    x86_64-linux-gnu/7.3.0/../../../x86_64-linux-gnu -
    L/lib/x86_64-linux-gnu -L/lib/../lib64 -L/usr/lib/
    x86_64-linux-gnu -L/usr/bin/../lib/gcc/x86_64-
    linux-gnu/7.3.0/../../.. -L/usr/lib/llvm-6.0/bin
    /../lib -L/lib -L/usr/lib welcome.o main_welcome.o
     -lgcc --as-needed -lgcc_s --no-as-needed -lc -
    lgcc --as-needed -lgcc_s --no-as-needed /usr/bin
    /../lib/gcc/x86_64-linux-gnu/7.3.0/crtend.o /usr/
    bin/../lib/gcc/x86_64-linux-gnu/7.3.0/../../../
    x86_64-linux-gnu/crtn.o
```

Table of Contents

Piloter son compilateur : les options

Contrôle des inputs/outputs

- fichiers sources : paramètres positionnels
- bibliothèques : `-l<libname>`
- chemins d'accès
  - fichiers headers : `-I<path>`
  - bibliothèques : `-L<path>`
- standard du language : `-std=c++11` par ex.
- warnings : `-W<warning>` (ex: `-Wall`, `-Werror`)
- infos : `--help`, `--version`, `-print-search-dirs`, ...

Passer des options aux différentes étapes

- pré-processeur: `-Wp,<option>`
- assembleur: `-Wa,<option>`
- linker: `-Wl,<option>`

Piloter l'optimiseur

Liste des optimisations :

- gcc/g++:

```
1          g++ --help=optimizers
```

- clang/llvm:

```
1          clang -help
2          opt -help
```

Optimisations globales : `-O<level>`

- `-O0` : pas d'optimisations, pratique pour le debug
- `-O3` : plus fort niveau d'optimisations
- `-Os` : `-O2`, mais minimise la taille du code
- `-Ofast`: `-O3` + `-ffastmath`

TD 2 : Piloter l'optimiseur

```
1  #ifdef BIG_N
2  #define n 100
3  #else
4  #define n 5
5  #endif
6
7  int main() {
8    int a[n];
9    for (int i = 0; i < n; ++i) { a[i] = i+1; }
10
11   int s = 0;
12   for (int i = 0; i < n; ++i) { s +=a [i]; }
13
14   if ( s== (n*(n+1))/2 )       { return s; }
15   return -1;
16 }
```

- avec *#define n 5*
  Comparer les binaires générés avec `-O0` et `-O1`
- avec *#define n 100*
  Comparer les binaires générés avec `-O0` et `-O3`

TD 3 : les options de debug (1)

```c
#include <assert.h>
#define MAX_SIZE 512

int count_char(const char* str) {
  int c = 0;
  while ( (c < MAX_SIZE) && (str[c] != '\0')) { ++c; }
  return c;
}

int main(int argc, char** argv) {
  assert(argc == 2 && "compliant number of arguments");
  return count_char(argv[1]);
}
```

```
clang debug.c -o debug
./debug toto
echo $?
./debug
```

TD 3 : les options de debug (2)

L'impact de `-DNDEBUG`

```
1  clang debug.c -o debug-n -DNDEBUG
2  ./debug-n toto
3  echo $?
4  ./debug-n
```

Comparer les binaires générés avec et sans `-DNDEBUG`

L'impact de `-g`

```
1  clang debug.c -o debug-ng -DNDEBUG -g
```

Comparer les binaires générés avec et sans `-g`

Table of Contents

TD 4 : clang vs. gcc

Reprendre le programme `array.c` du TD 2

Comparer les assembleurs générés par gcc et clang pour `n = 100` et différents niveaux d'option `-O<level>` :

`td4/gcc_vs_clang.sh` :

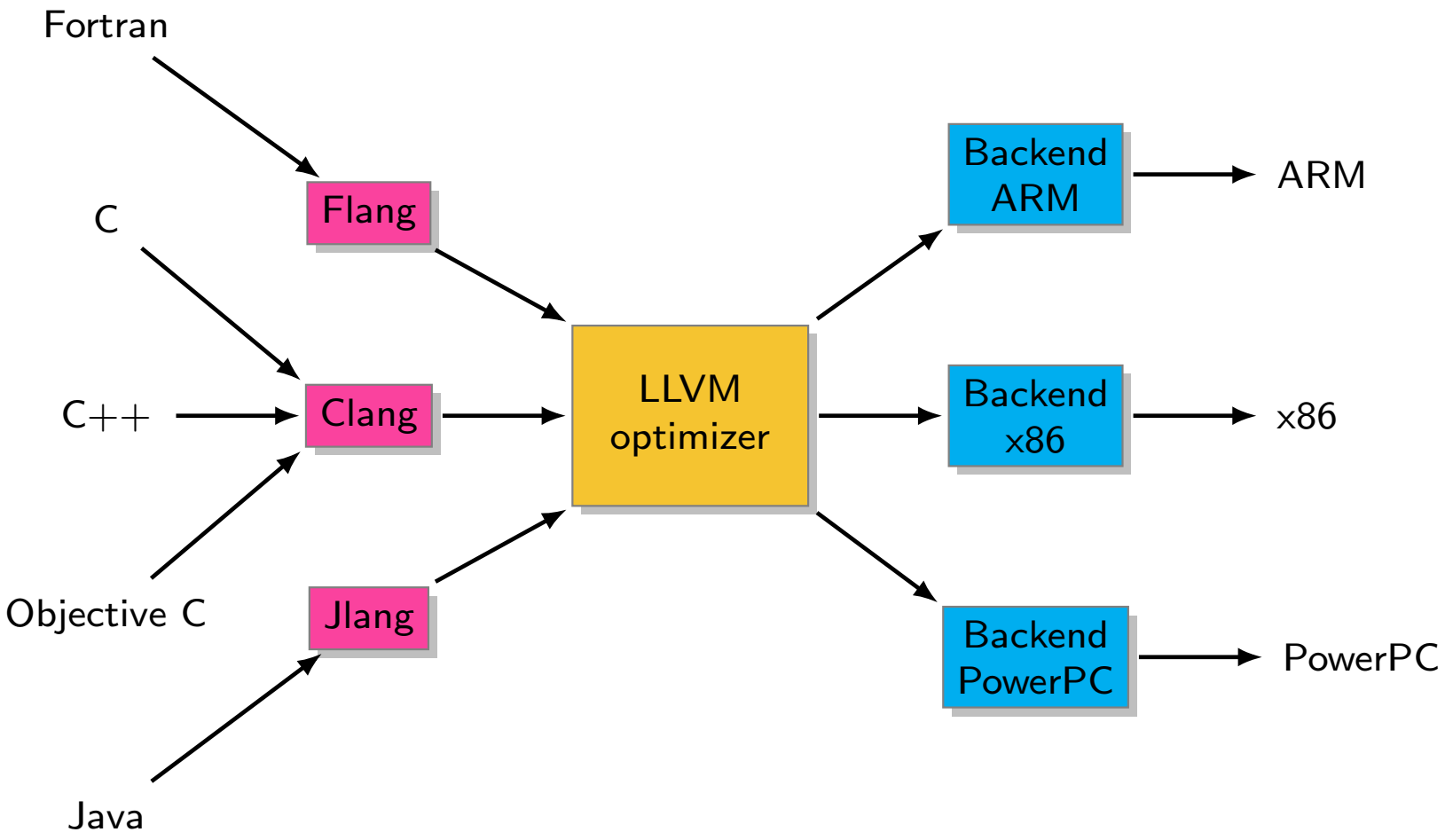
```
1  LEVEL=$1
2
3  # assembleur
4  clang array.c -DBIG_N -O$LEVEL -S -o
       array_clang_O$LEVEL.s
5  gcc array.c -DBIG_N -O$LEVEL -S -o array_gcc_O$LEVEL.s
6
7  # binaire
8  clang array.c -DBIG_N -O$LEVEL -o array_clang_O$LEVEL
9  gcc array.c -DBIG_N -O$LEVEL -o array_gcc_O$LEVEL
```

```
1  bash gcc_vs_clang.sh 1
2  bash gcc_vs_clang.sh 2
3  bash gcc_vs_clang.sh 3
```

Comparer aussi les versions générées par clang avec `-O2` et gcc avec `-O3`.

Table of Contents

Clang/LLVM


Fortran
C
C++
Objective C
Java
Flang
Clang
Jlang
LLVM
optimizer
Backend
ARM
Backend
x86
Backend
PowerPC
ARM
x86
PowerPC

L'IR LLVM

- Pseudo assembleur
- Nombre illimité de registres virtuels = *Values*
- Assignation unique des registres virtuels

```
1  #ifndef DEFAULT
2  #define DEFAULT "everybody"
3  #endif
4
5  extern void welcome(char const*);
6
7  int main(int argc, char const* argv[]) {
8    if       (argc == 1) { welcome(DEFAULT); }
9    else if (argc == 2) { welcome(argv[1]); }
10   else { return 1; }
11   return 0;
12 }
```

```
1  clang main_welcome.c -S -emit-llvm -o -
```

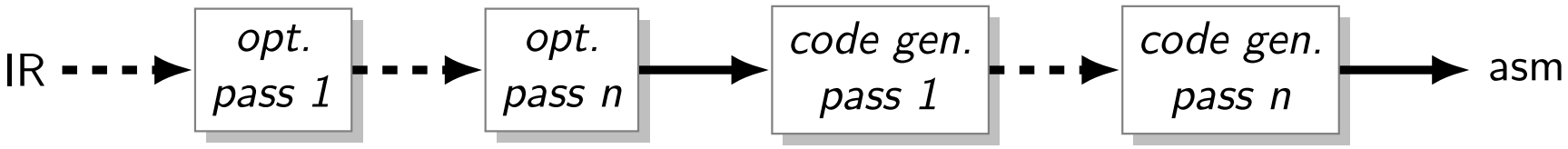
```
1  define i32 @main(i32, i8**) #0 {
2    %3 = alloca i32, align 4
3    %4 = alloca i32, align 4
4    %5 = alloca i8**, align 8
5    store i32 0, i32* %3, align 4
6    store i32 %0, i32* %4, align 4
7    store i8** %1, i8*** %5, align 8
8    %6 = load i32, i32* %4, align 4
9    %7 = icmp eq i32 %6, 1
10   br i1 %7, label %8, label %9
11
12 ; <label>:8:                                     ; preds = %2
13   call void @welcome(i8* getelementptr inbounds ([10 x i8], [10
           x i8]* @.str, i32 0, i32 0))
14   br label %18
15
16 ...
17
18 ; <label>:18:                                    ; preds = %17, %8
19   store i32 0, i32* %3, align 4
20   br label %19
21
22 ; <label>:19:                                    ; preds = %18, %16
23   %20 = load i32, i32* %3, align 4
24   ret i32 %20
25 }
```

```c
#ifdef BIG_N
#define n 100
#else
#define n 5
#endif

int main() {
  int a[n];
  for (int i = 0; i < n; ++i) { a[i] = i+1; }

  int s = 0;
  for (int i = 0; i < n; ++i) { s +=a [i]; }

  if ( s== (n*(n+1))/2 )       { return s; }
  return -1;
}
```

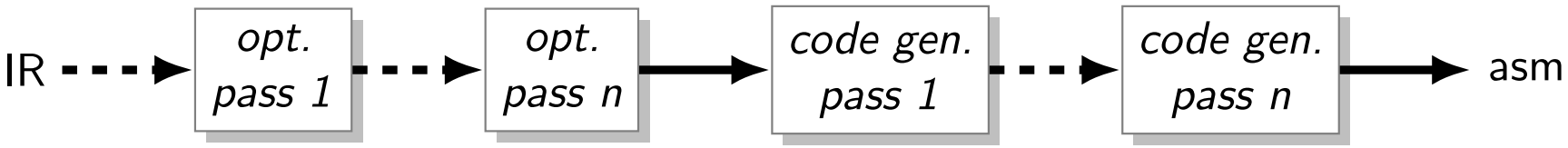
```
clang array.c -S -O1 -emit-llvm -o -
```

```
1  define i32 @main() local_unnamed_addr #0 {
2    ...
3    br label %3
4
5  ; <label>:3:                                     ; preds = %3,
       %0
6    %4 = phi i64 [ 0, %0 ], [ %5, %3 ]
7    %5 = add nuw nsw i64 %4, 1
8    %6 = getelementptr inbounds [5 x i32], [5 x i32]* %1, i64 0,
       i64 %4
9    ...
10   br i1 %8, label %9, label %3
11
12 ; <label>:9:                                     ; preds = %3
13   br label %13
14
15 ; <label>:10:                                    ; preds = %13
16   ...
17   ret i32 %12
18
19 ; <label>:13:                                    ; preds = %9,
       %13
20   %14 = phi i64 [ %19, %13 ], [ 0, %9 ]
21   %15 = phi i32 [ %18, %13 ], [ 0, %9 ]
22   %16 = getelementptr inbounds [5 x i32], [5 x i32]* %1, i64 0,
       i64 %14
23   %17 = load i32, i32* %16, align 4, !tbaa !2
24   ...
25   br i1 %20, label %10, label %13
26 }
```

LLVM : organisation en passes

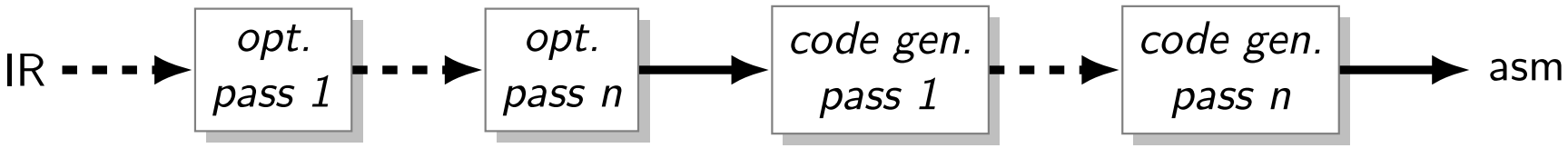


LLVM : organisation en passes



Constants Propagation
Dead Code Eliminationw
Inlining
Loop Unrolling
...

LLVM : organisation en passes



Constants Propagation
Dead Code Eliminationw
Inlining
Loop Unrolling
...

Reg. Allocation
ISel
Peephole Opt.
Target Lowering
...

Table of Contents