Langages et Compilation

T. Goubier
L3A
2019/2020

Langages et Compilation

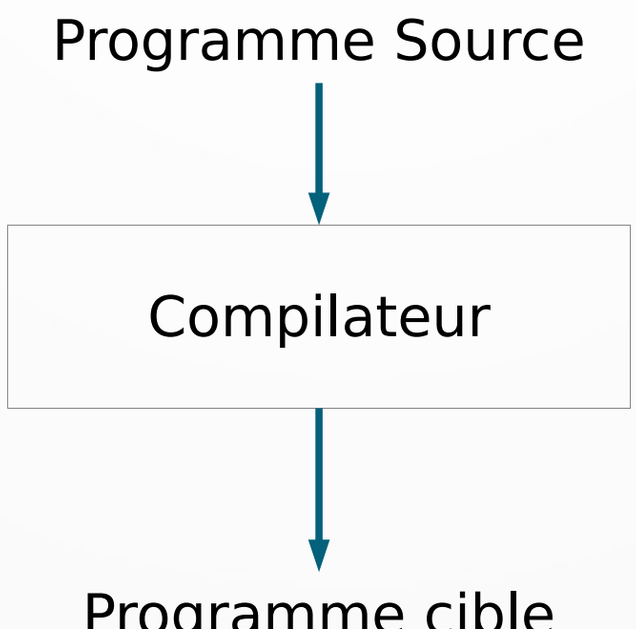
- Implémentation et concepts de langages de programmation
- Structure d’un compilateur
- Techniques du front-end d’un compilateur
  - Analyse lexicale, syntaxique et interprétation
- Pratique: réaliser un interpréteur

L1: Introduction

- Notion d'exécution d'une application dans un système informatique
- Code source :
  - représentation orientée humain de ce qu'une application doit faire
  - N'est pas forcément du texte...
- Définitions d'un compilateur, interpréteur, exécutable

L1: Introduction

- Compilateur
  - Traduit d'un langage source vers un autre langage



C, C++, Java, etc..
Mais aussi représentations graphiques

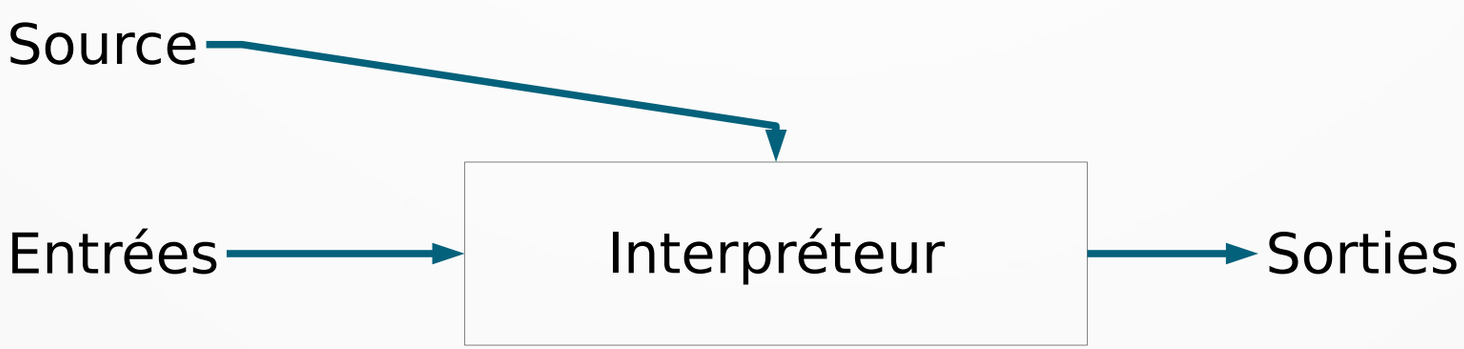
assembleur, bytecode, C

L1: Introduction

- Exécutable
  - Désigne un programme qui peut s’exécuter sur une machine
  - Peut nécessiter un runtime (un OS?)
  - A des entrées associe des sorties après exécution

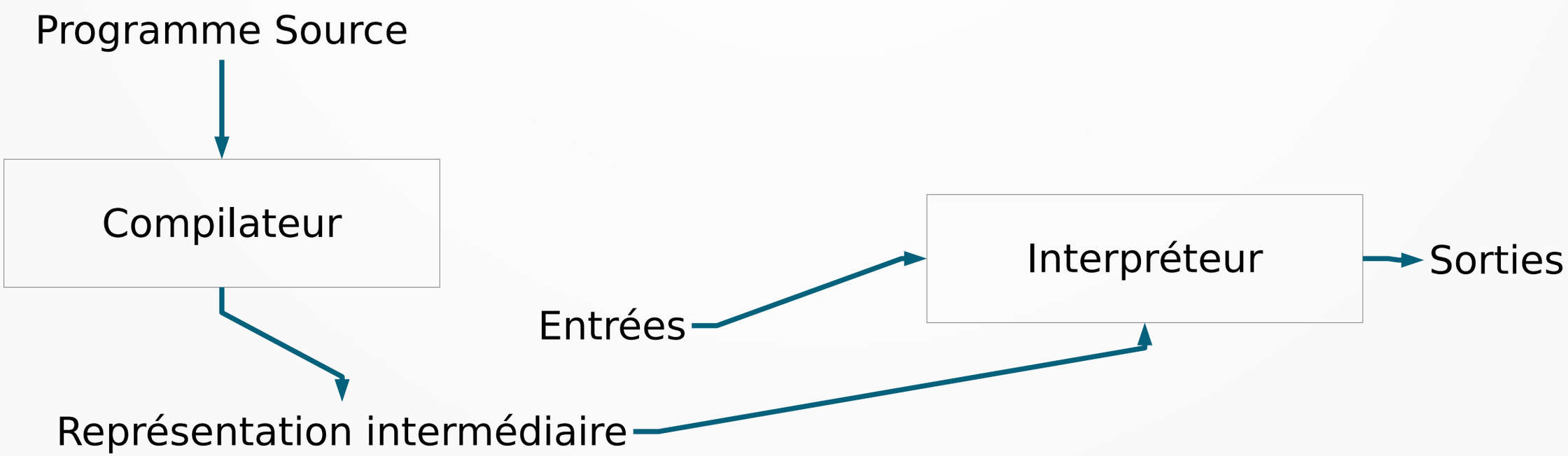
L1: Introduction

- Interpréteur
  - Au lieu de traduire le programme cible, prend un programme source, des entrées et produit directement un résultat
  - Pour chaque exécution, retraite le source



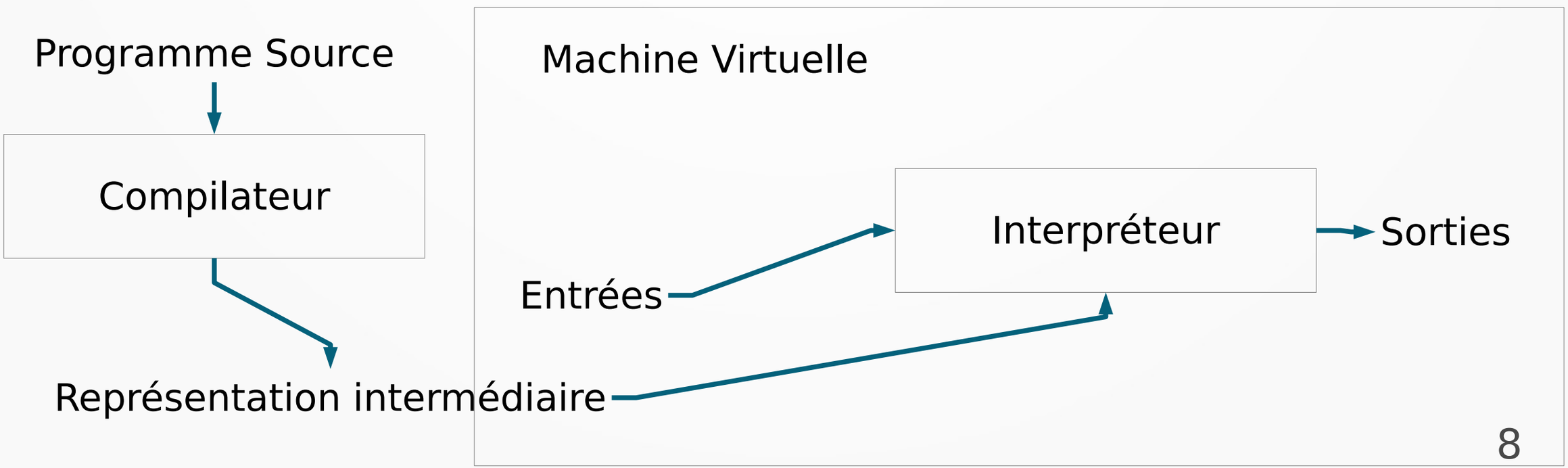
L1: Introduction

- Hybride
  - Compilation vers des formes intermédiaires compilées ou interprétées



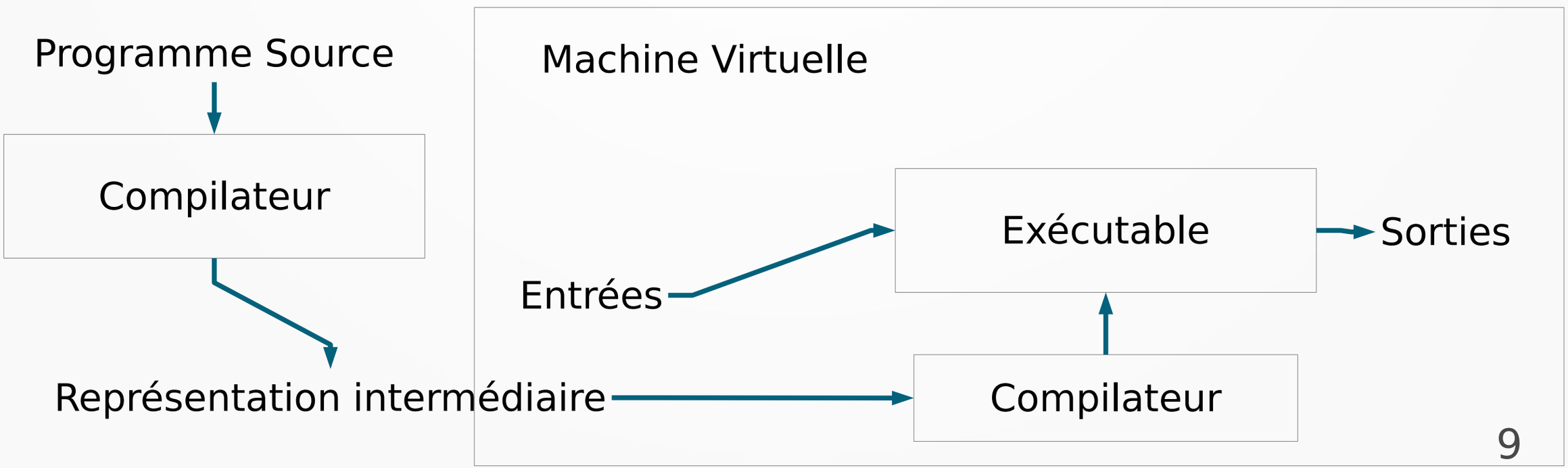
L1: Introduction

- Hybride (2)
  - Compilation vers des formes intermédiaires compilées ou interprétées – exécutées dans une machine virtuelle



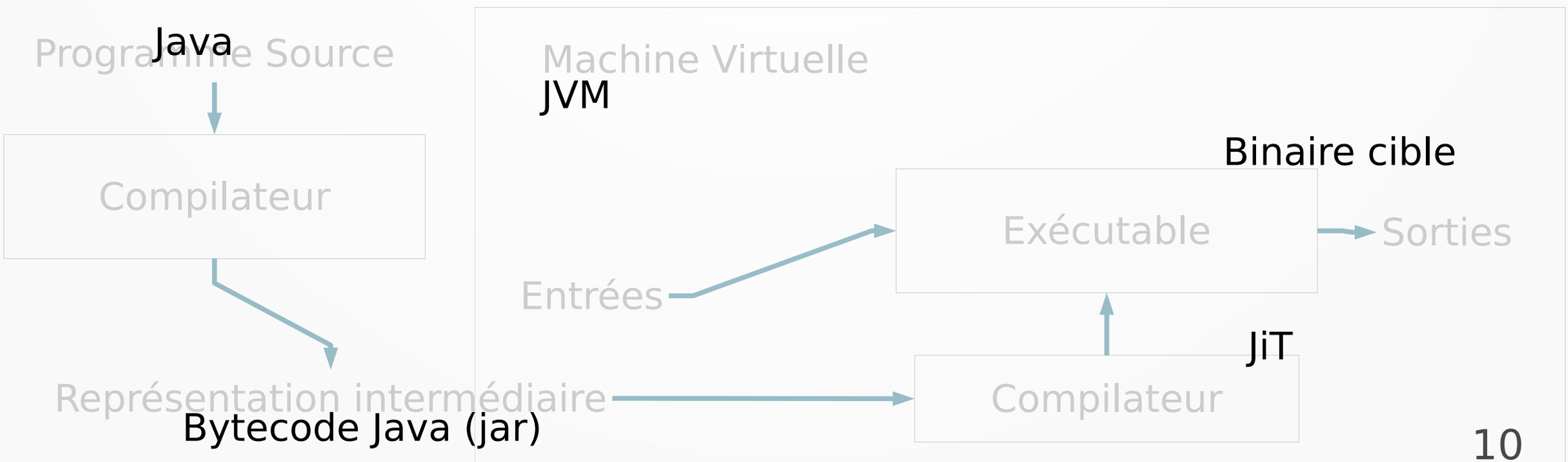
L1: Introduction

- Hybride (3)
  - Compilation vers des formes intermédiaires compilées ou interprétées – exécutées dans une machine virtuelle



L1: Introduction

- Hybride (3)
  - Compilation vers des formes intermédiaires compilées ou interprétées – exécutées dans une machine virtuelle



L1: Introduction

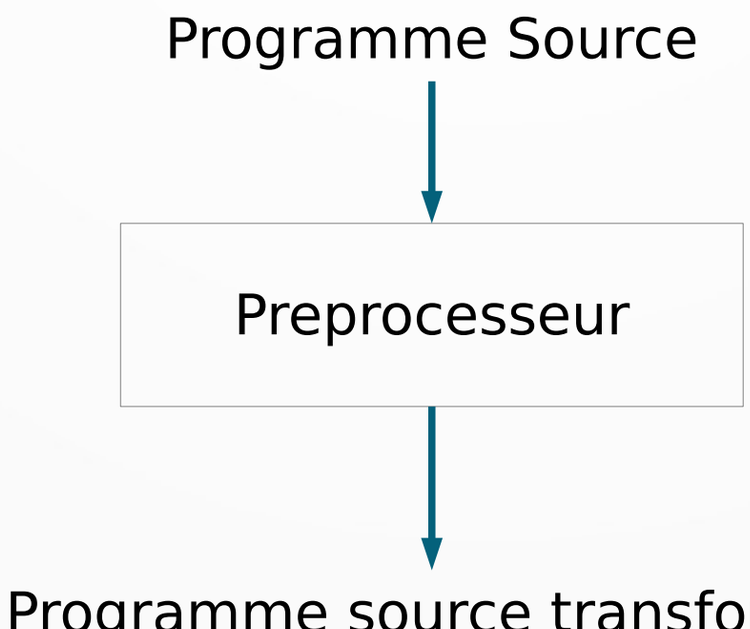
- JiT
  - Just in Time
    - Ne compile que le code réellement utilisé, une fois sur la cible d'exécution
  - L'application est conservée en représentation intermédiaire dans la mémoire
  - Sur exécution du code,
    - compilation et ajout à un cache de code natif
    - exécution à partir du cache
  - Intérêt:
    - bytecode est compact, portable.
    - exécutable utilise seulement une partie de son code
  - Défaut:
    - Coût de la compilation / difficulté à optimiser globalement

L1: Introduction

- AoT
  - Ahead of Time
    - Pour éviter la surcharge du JiT / de la VM
    - Compiler en avance de phase
    - Produire un exécutable (ex: android: à l'installation)
  - Mais
    - Certaines optimisations en peuvent pas être faites
    - Un JiT est capable d'optimiser en fonction de l'usage

L1: Introduction

- Préprocesseur / macros
  - Traduit d'un programme source vers un autre programme source transformé

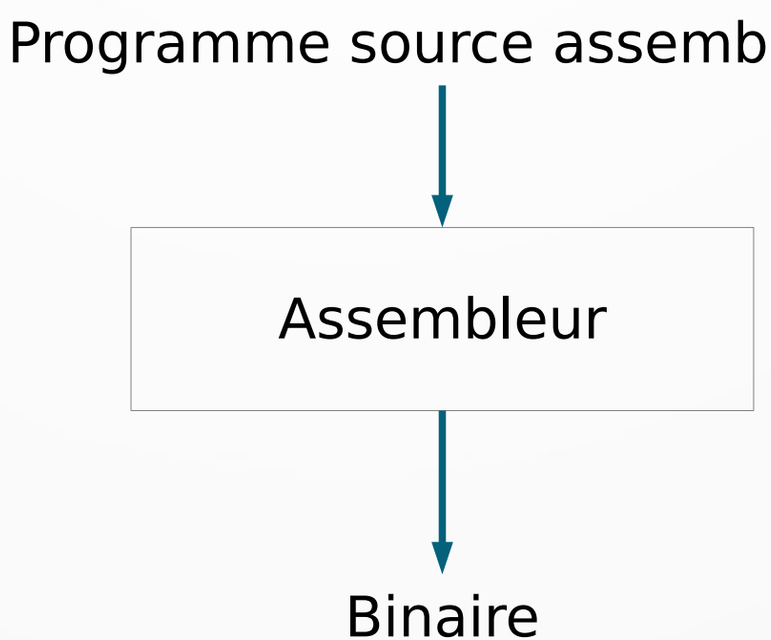


C, C++ avec #include / #if

C, C++ avec #line

L1: Introduction

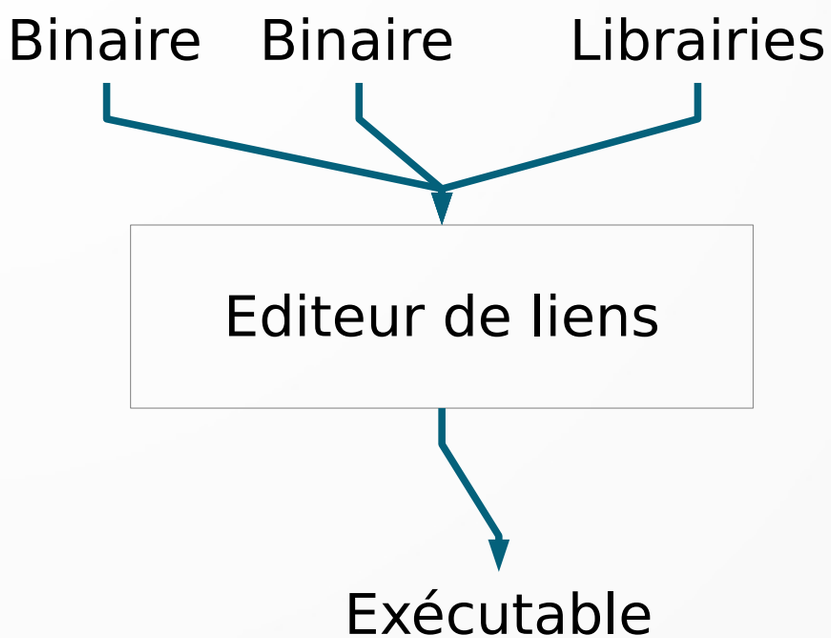
- Assembleur
  - Converti un code source assembleur en un binaire
  - Le binaire peut être incomplet (symboles non définis)



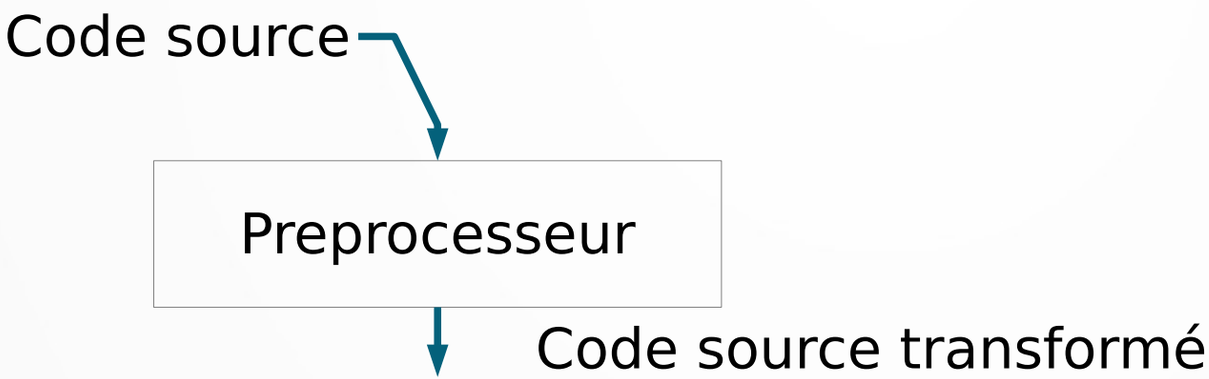
L1: Introduction

- Editeur de liens
  - Converti un groupe de binaires en un exécutable
  - Résout les symboles non définis (fonctions présentes dans d'autres binaires)
  - Cible statique ou dynamique
    - Ajoute le code de chargement des librairies dynamiques
  - Trouve un point de démarrage (main)
    - Ou émet une erreur

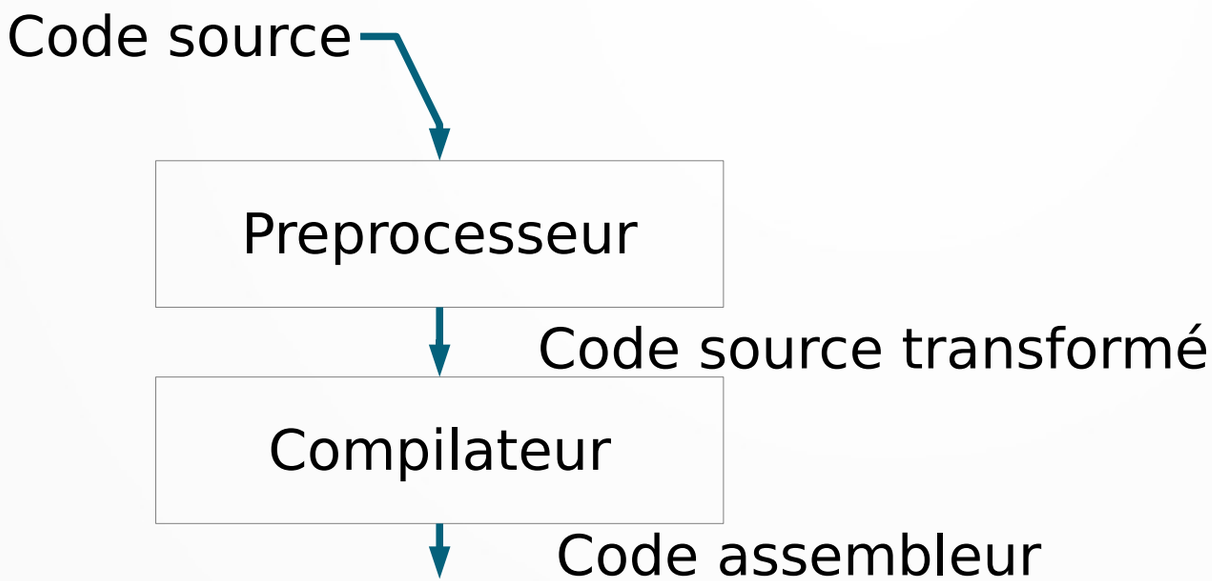
      _start': (.text+0x20): undefined reference to main



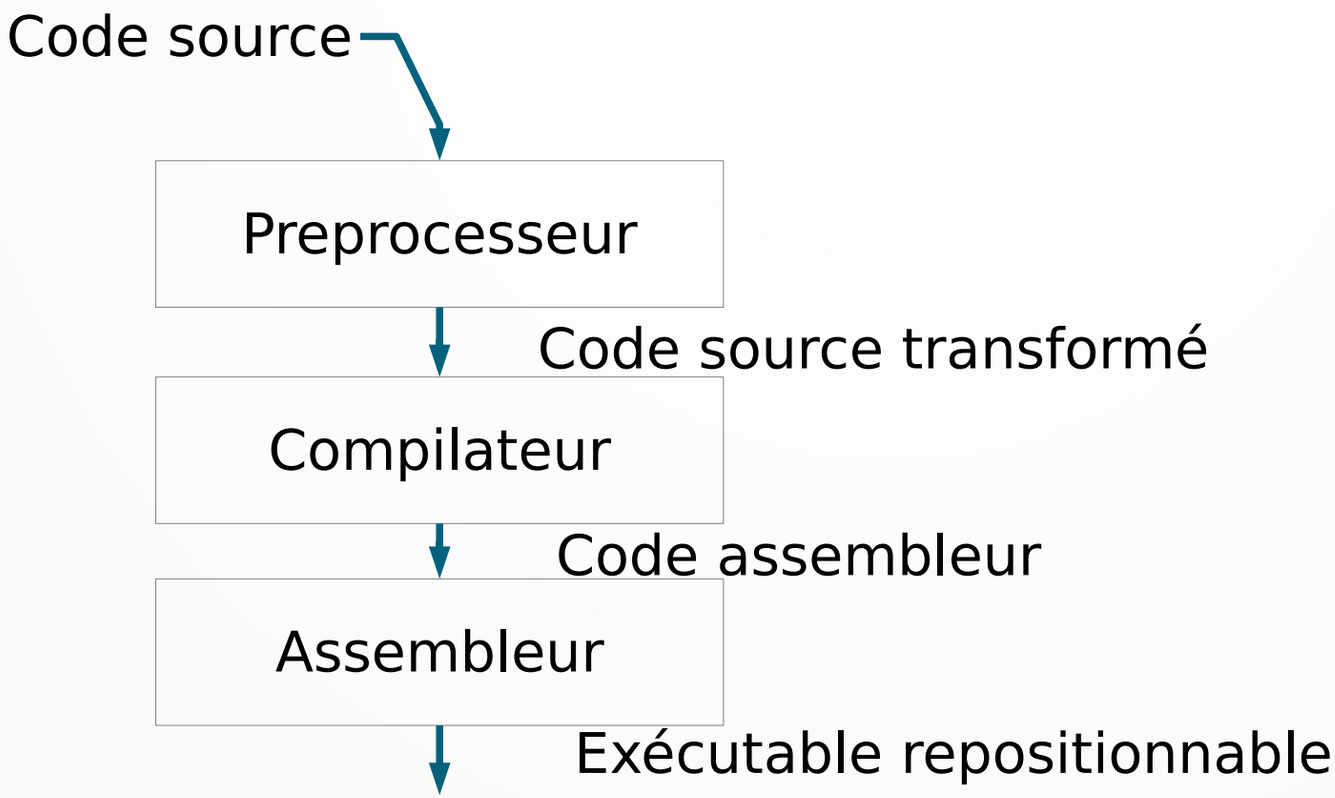
L1: Introduction



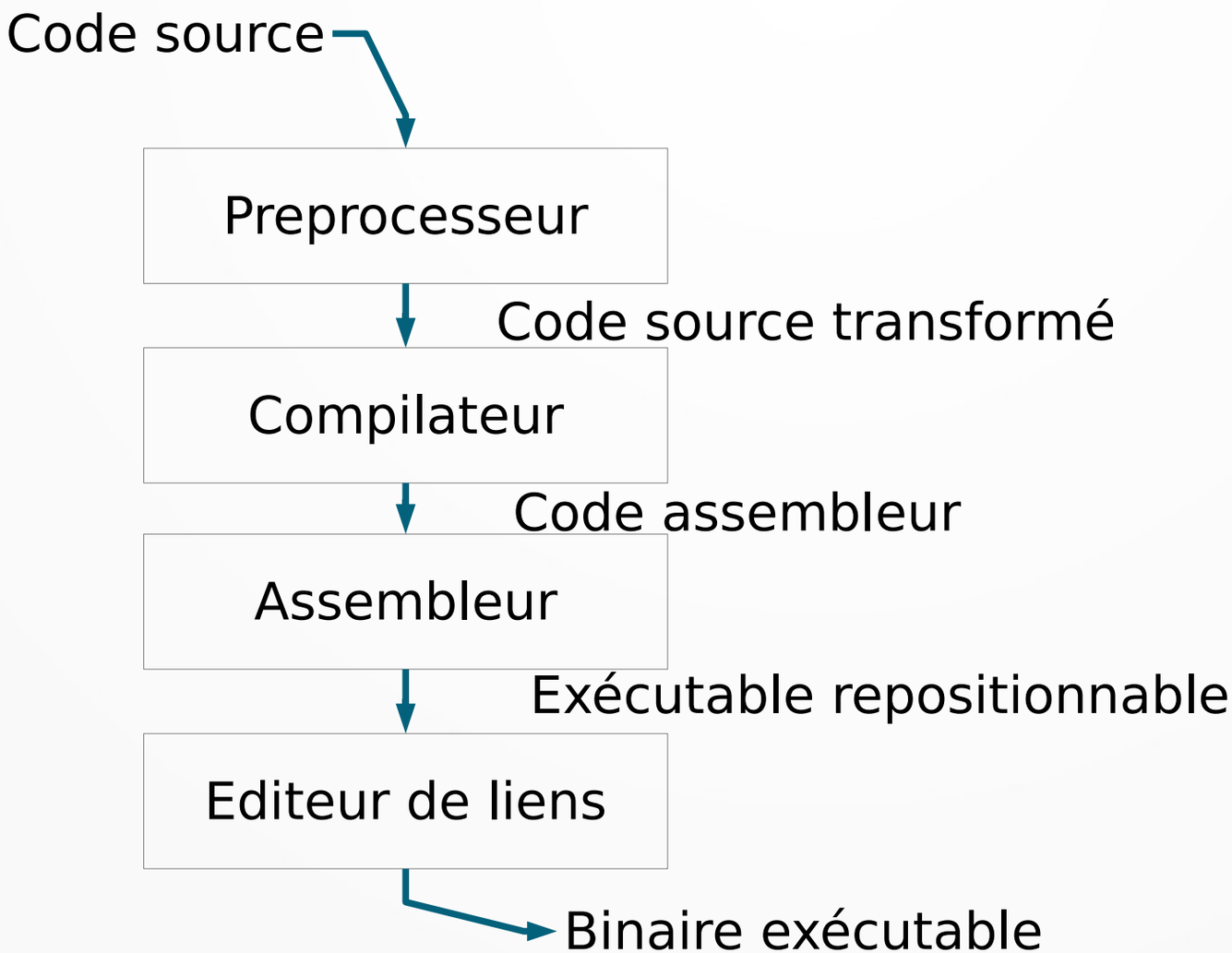
L1: Introduction



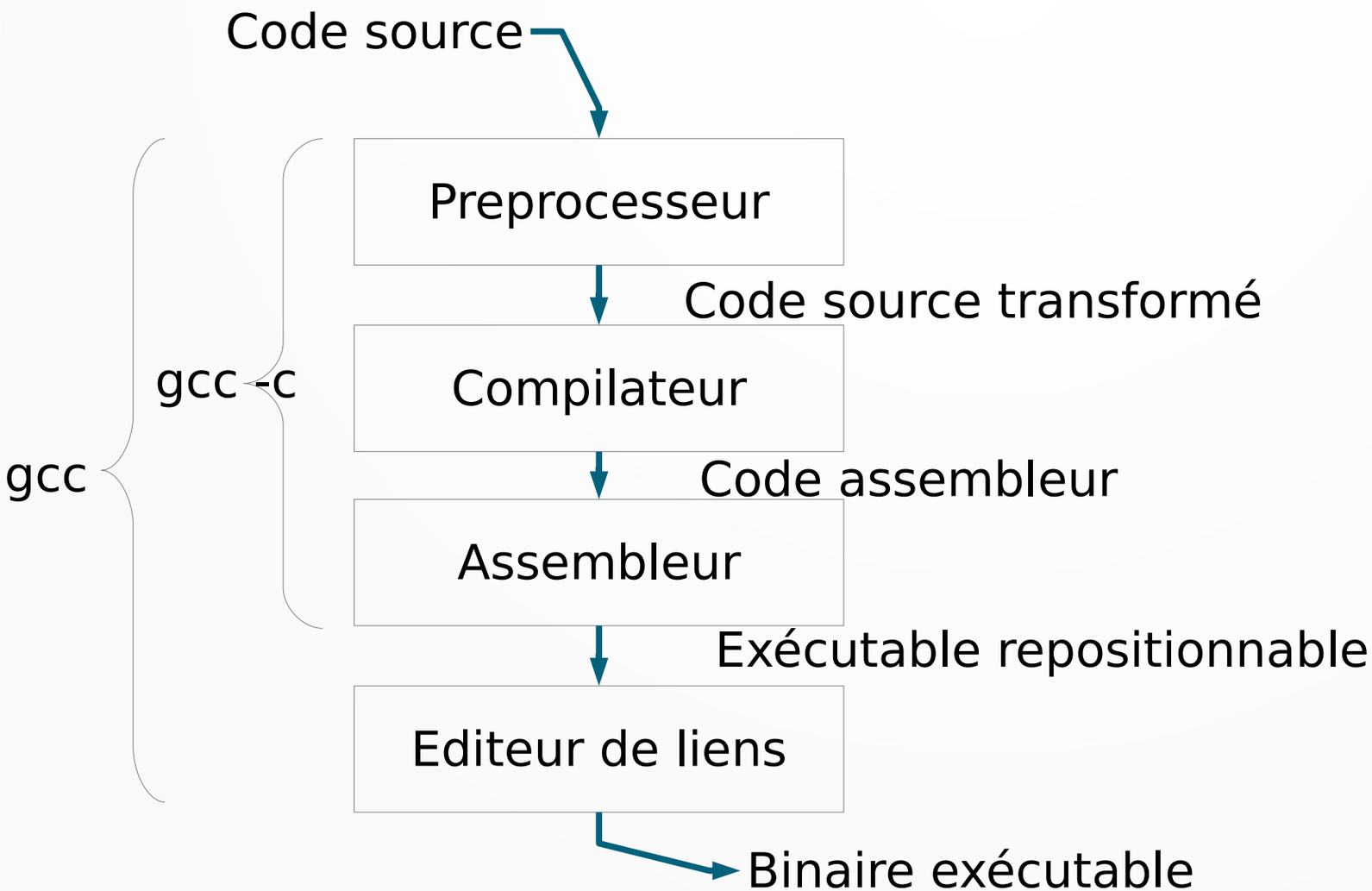
L1: Introduction



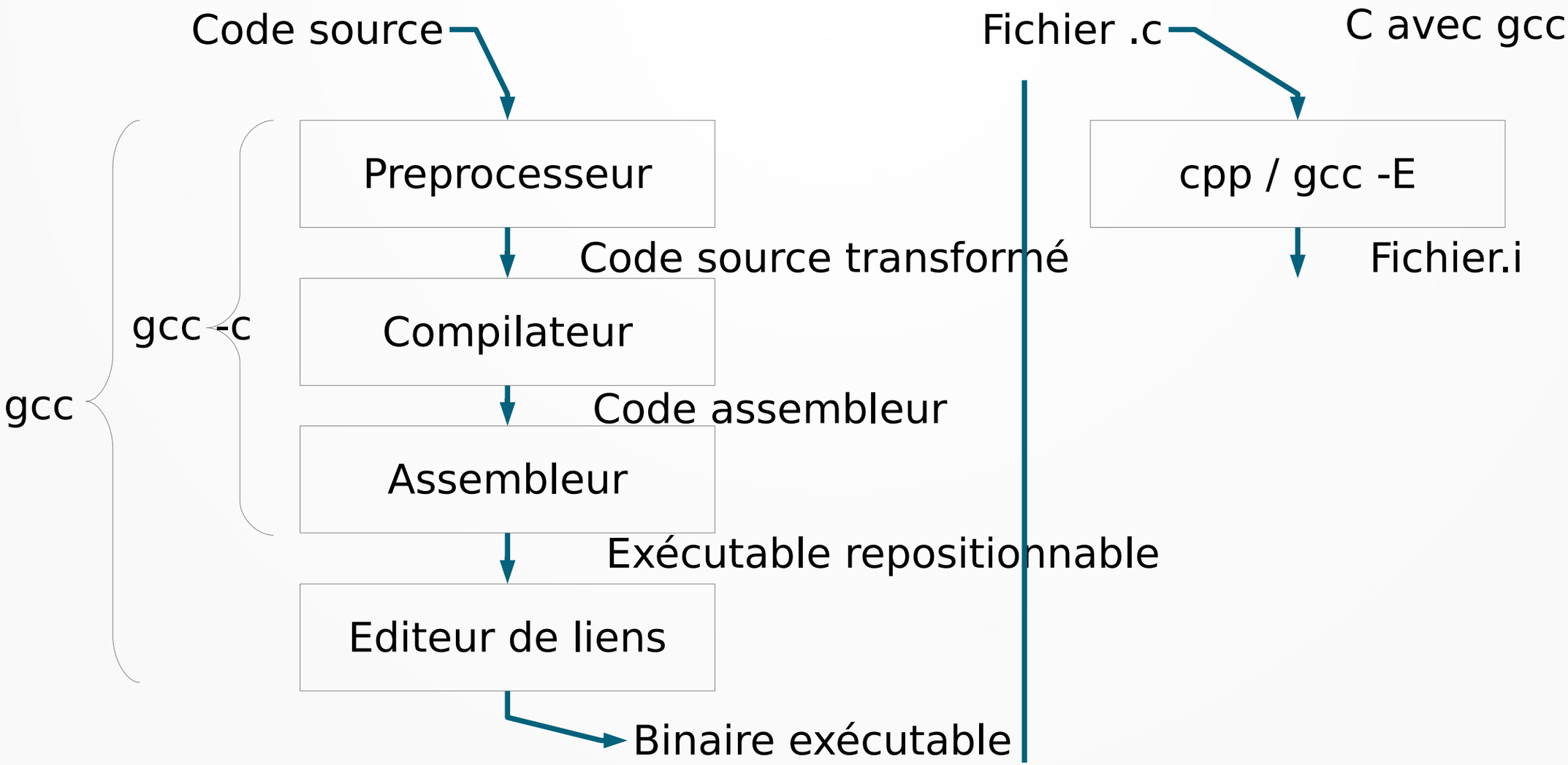
L1: Introduction



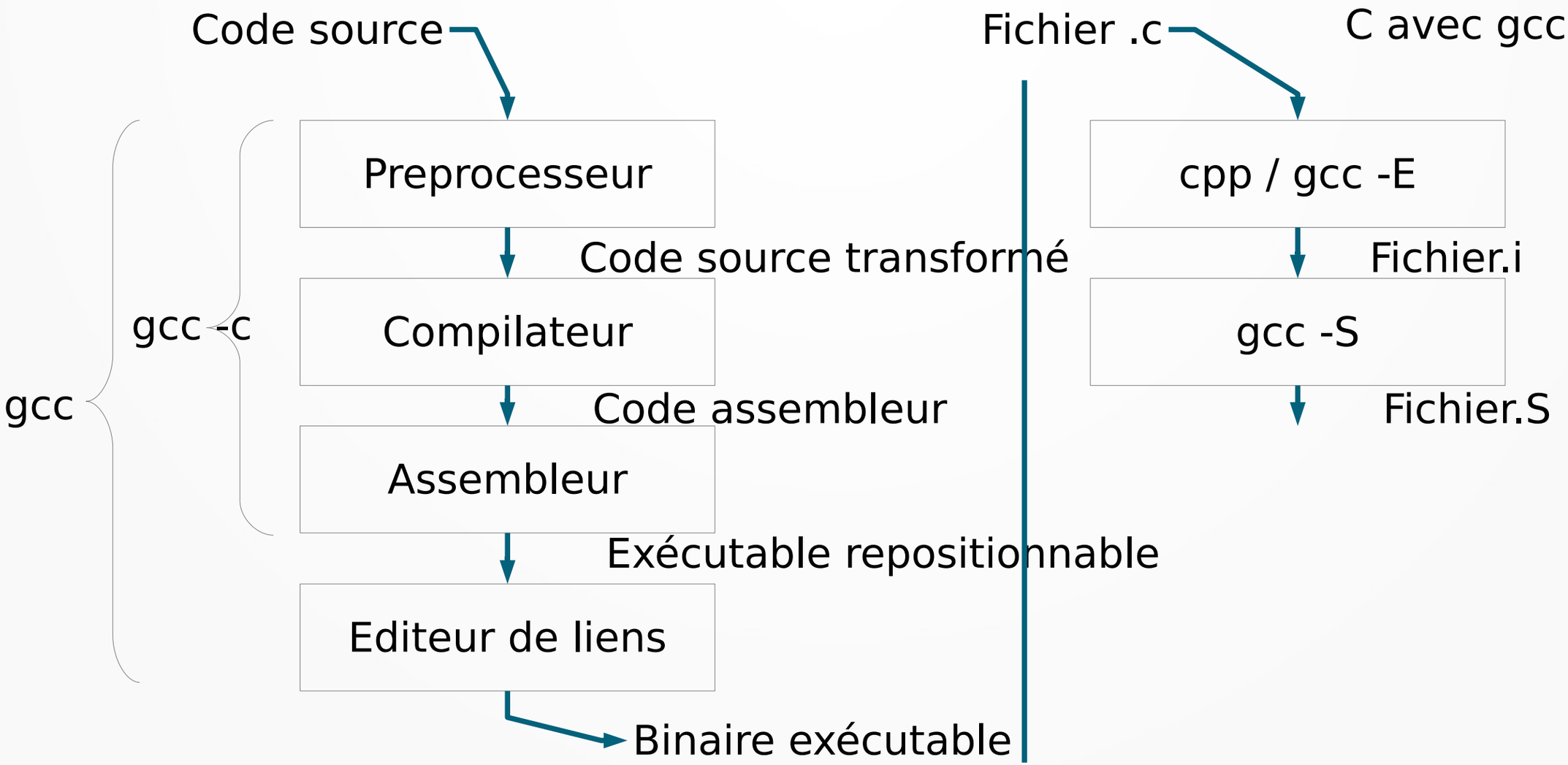
L1: Introduction



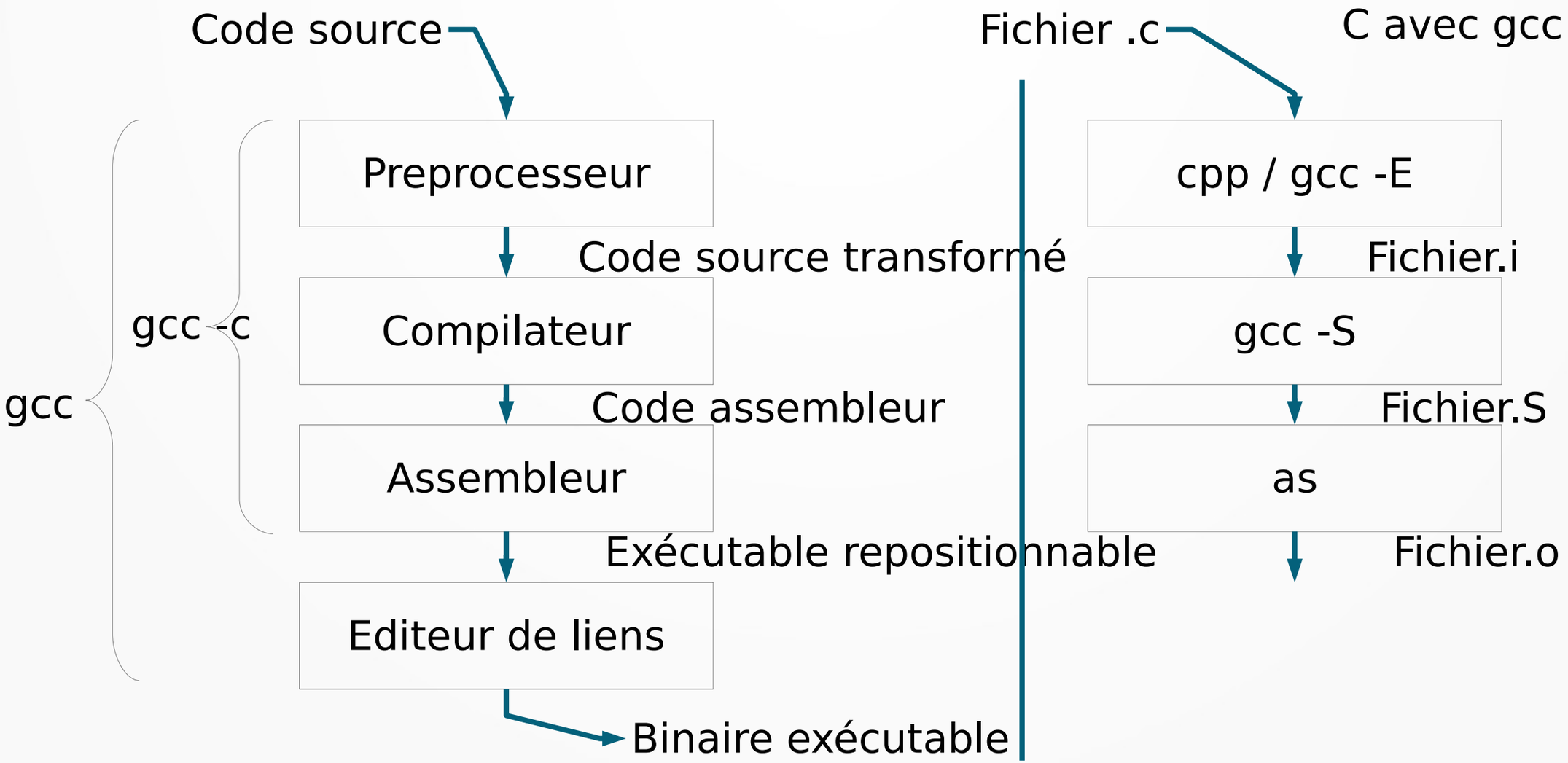
L1: Introduction



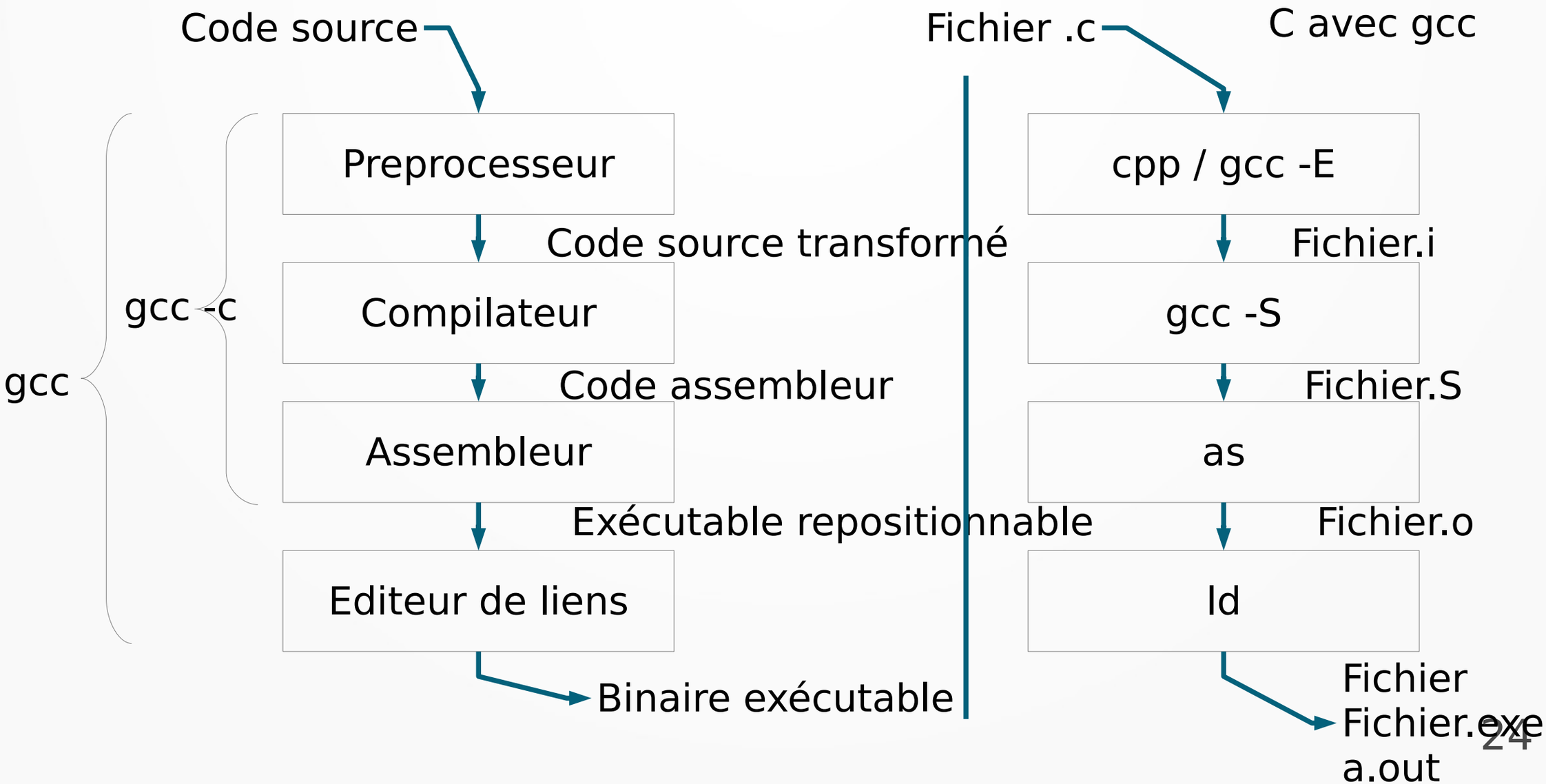
L1: Introduction



L1: Introduction



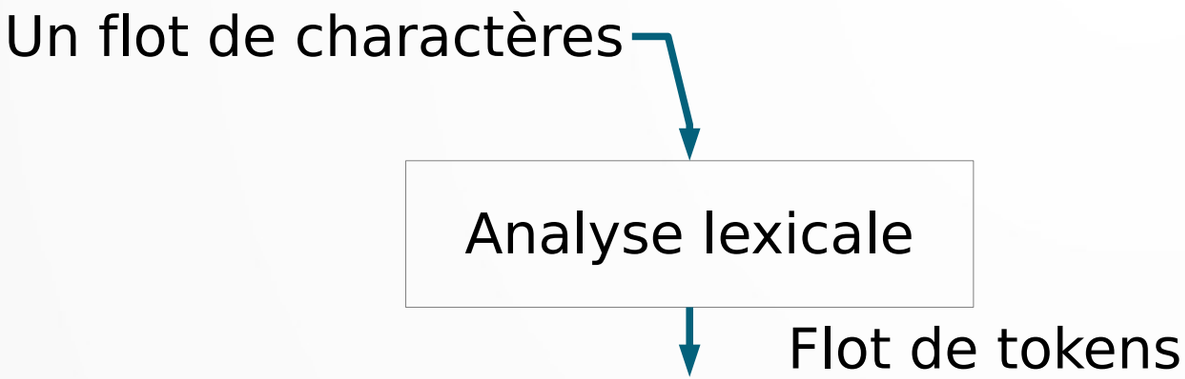
L1: Introduction



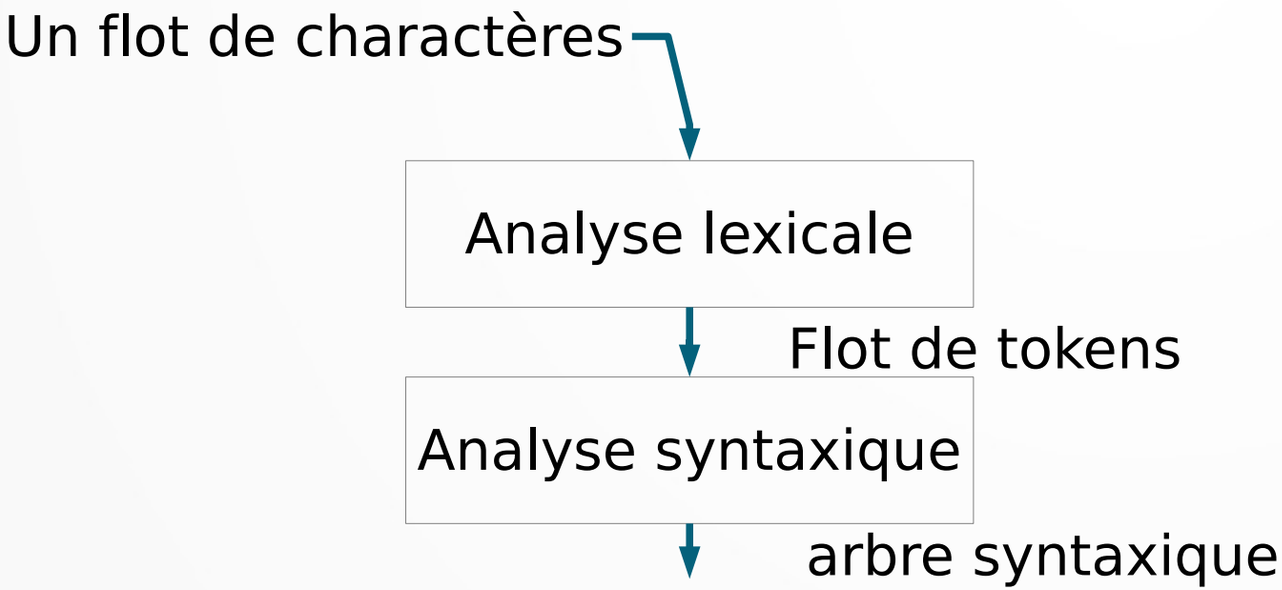
L1: Introduction

- Maintenant nous allons nous intéresser à un des éléments
  - Le compilateur
- Mais chacun des éléments est complexe en soi
  - Le linker
  - Le pré-processeur
  - Un livre entier pour chacun d'eux
    - A vous de savoir quand il faudra aller se documenter sur ces outils
    - Pour résoudre vos problèmes de demain

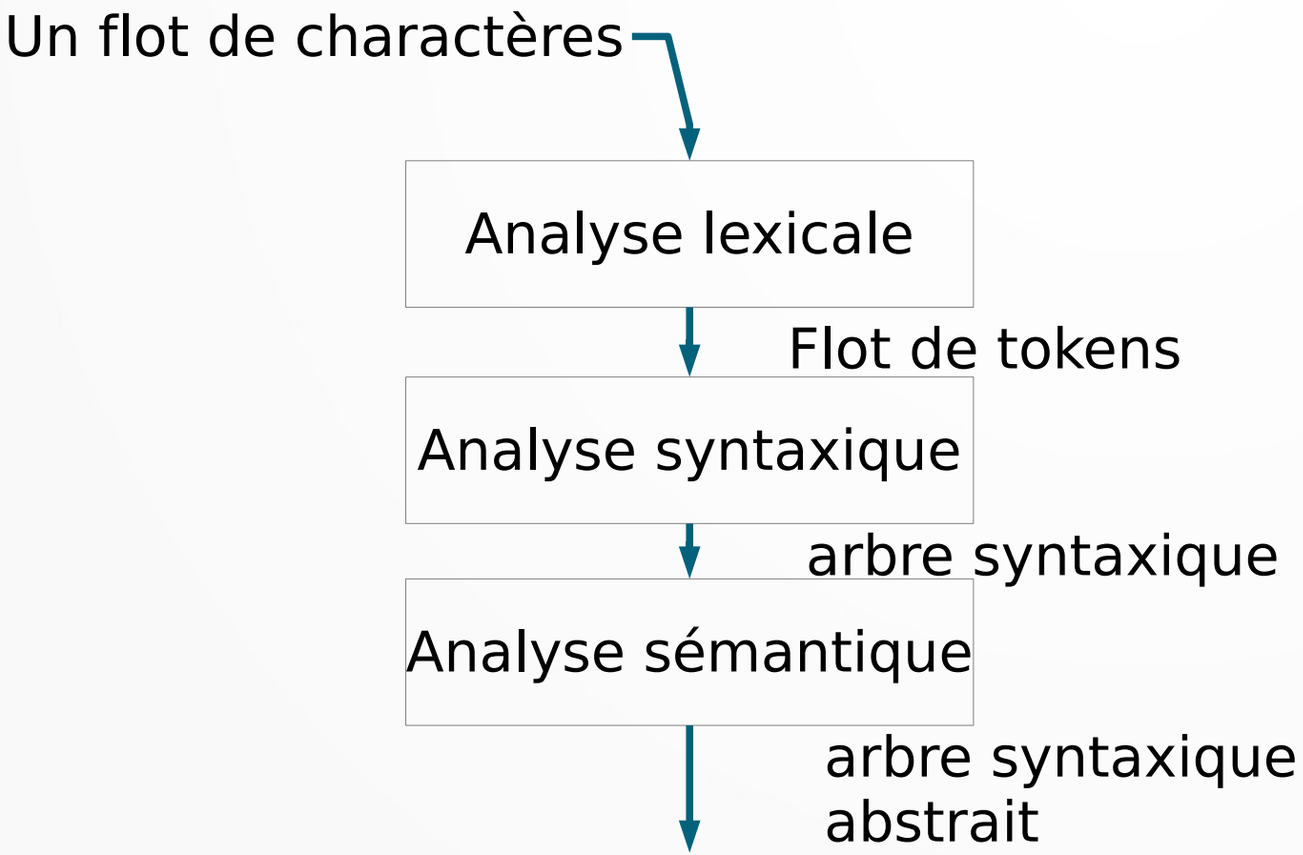
L1: Introduction



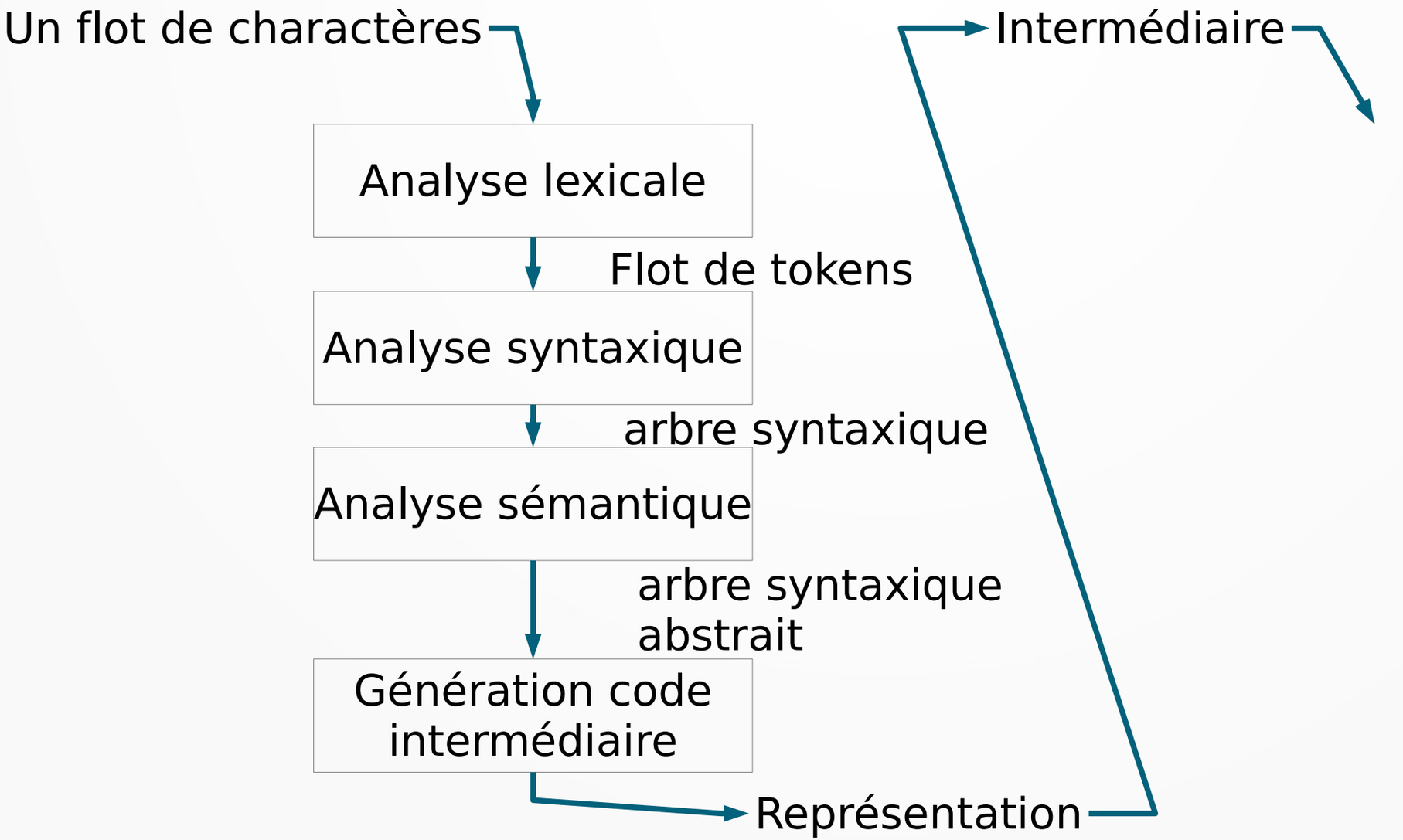
L1: Introduction



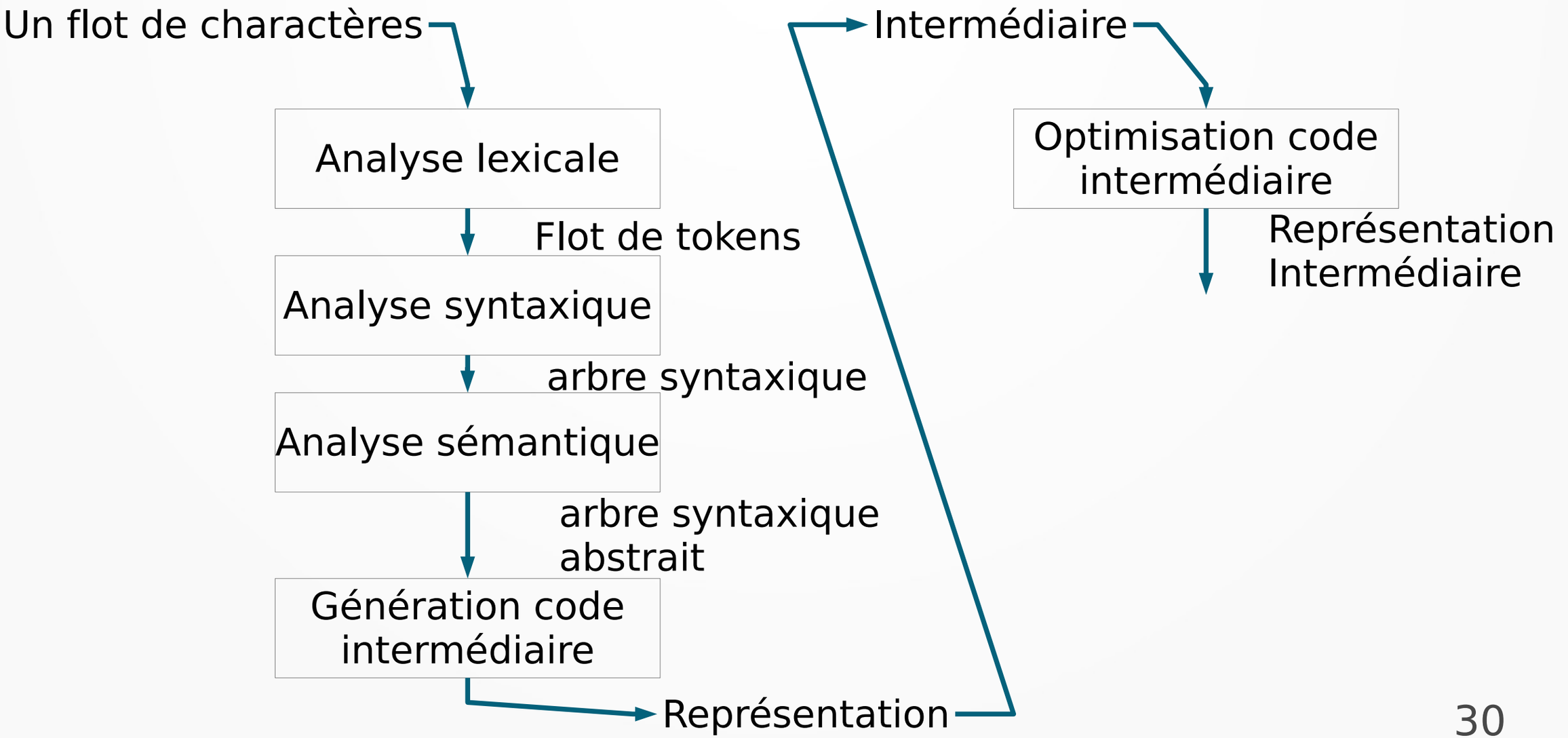
L1: Introduction



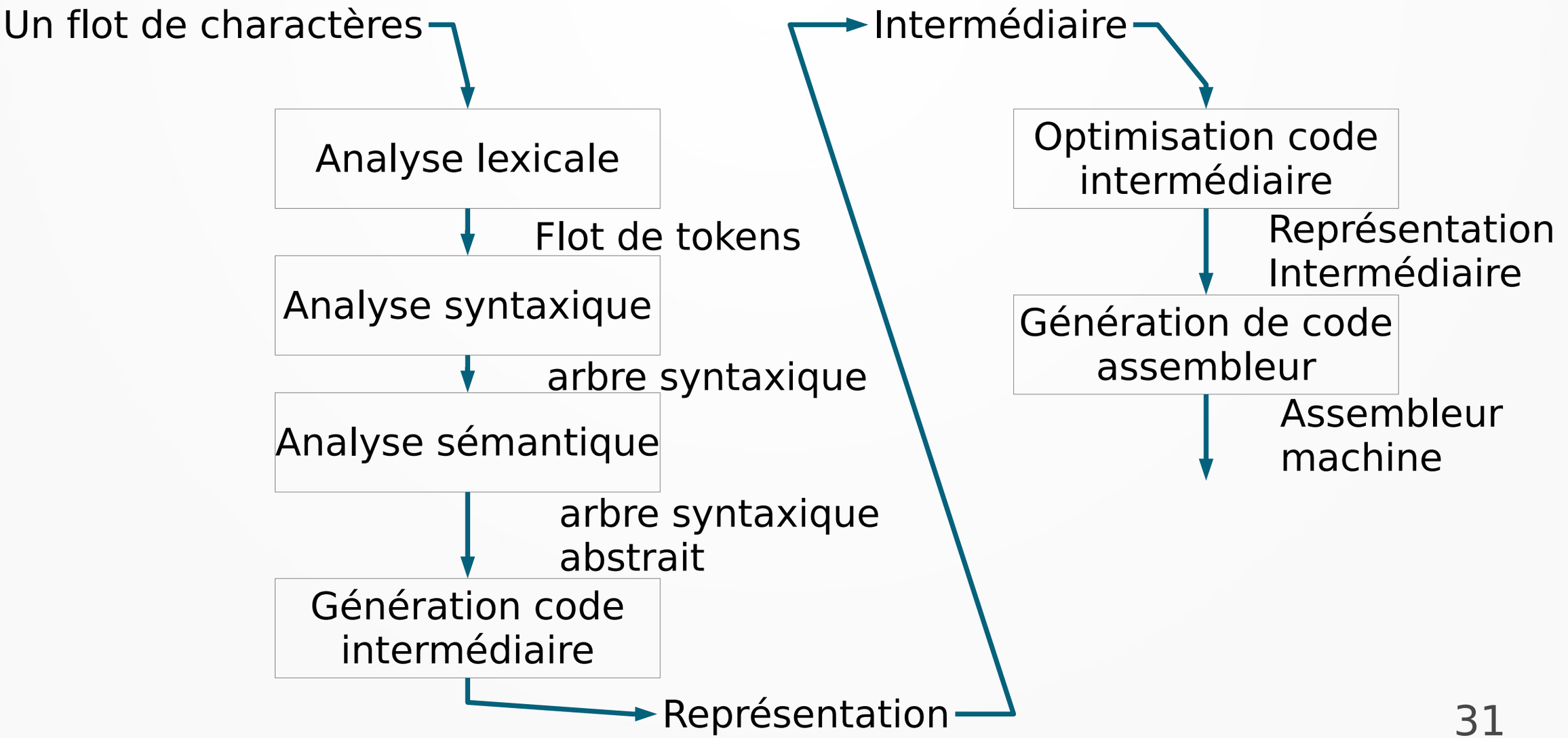
L1: Introduction



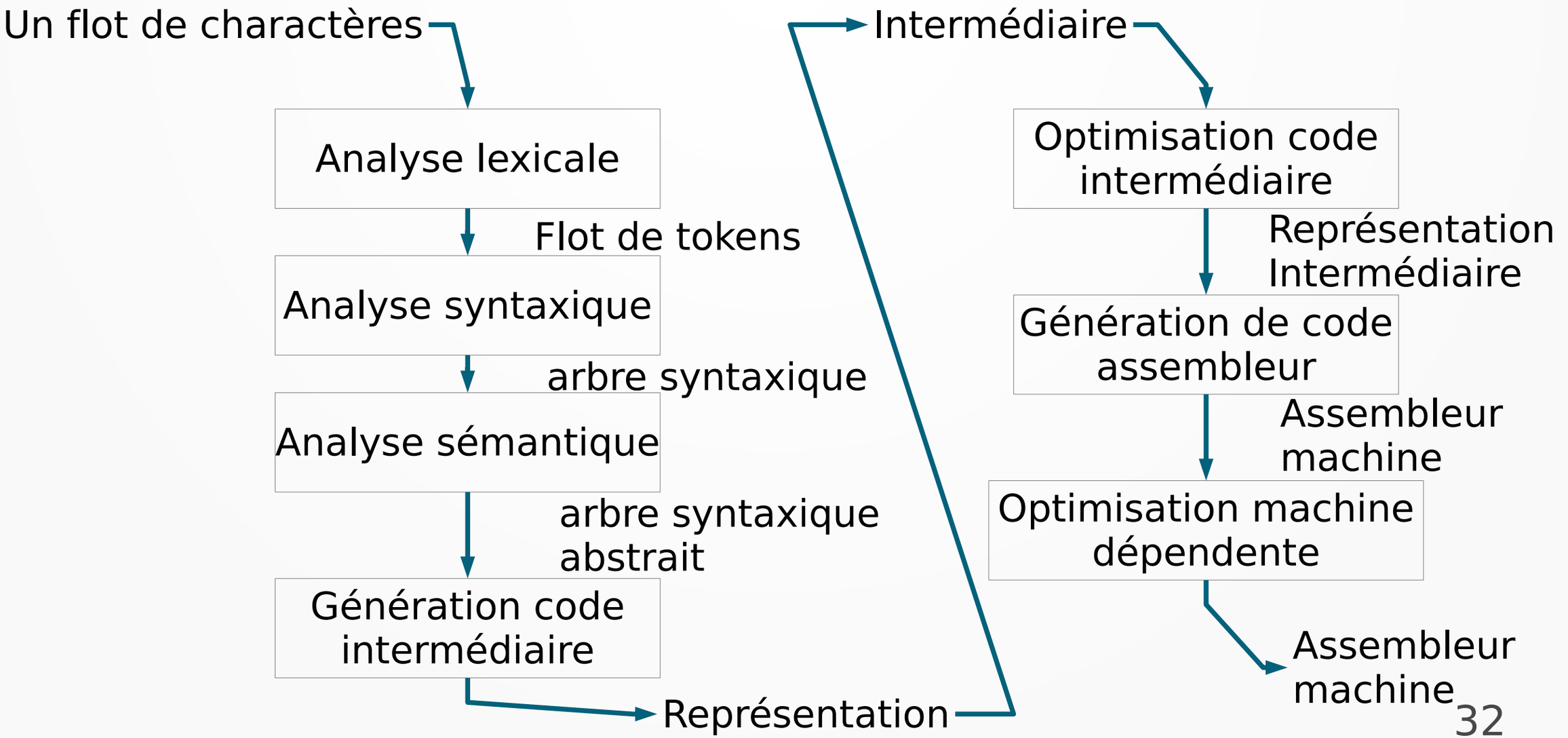
L1: Introduction



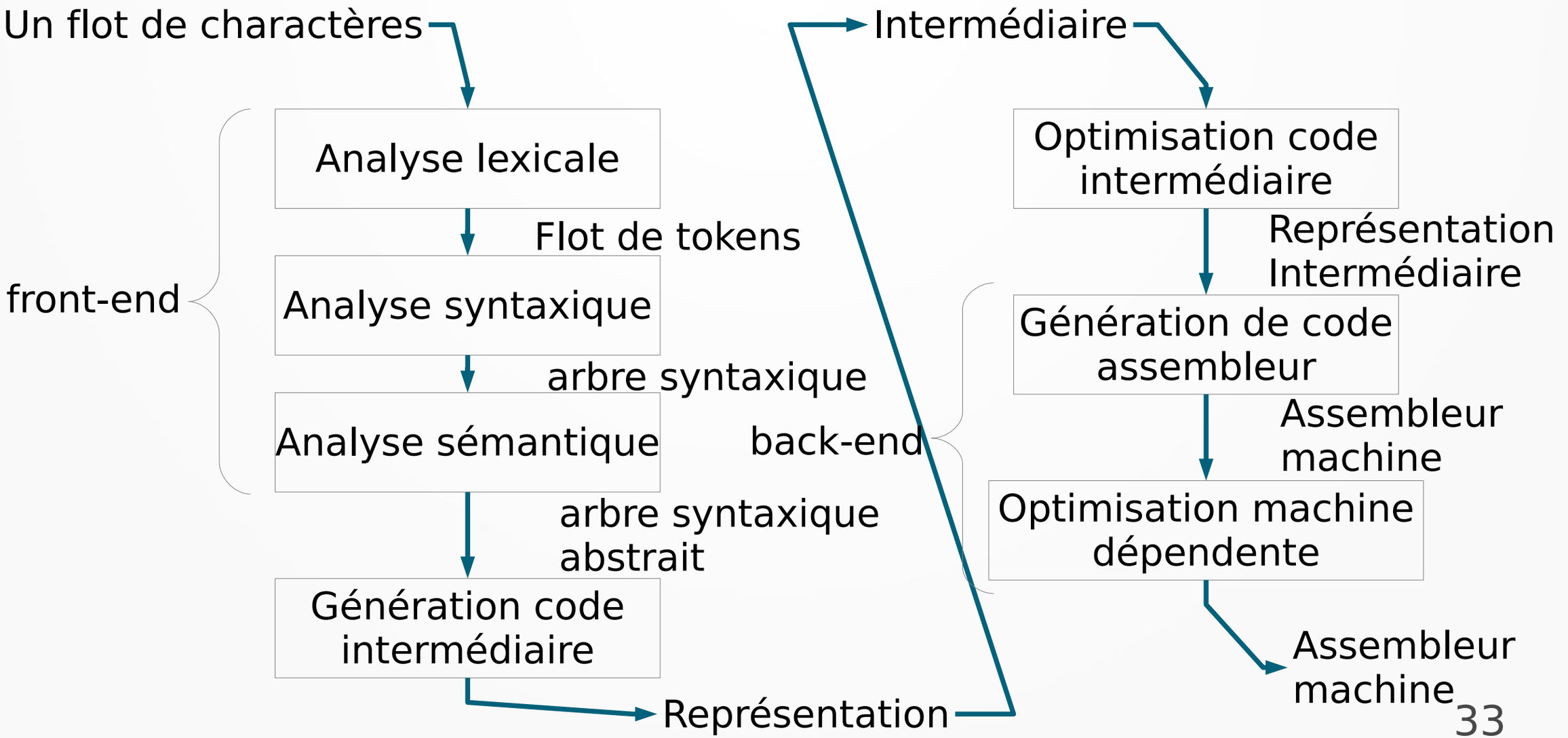
L1: Introduction



L1: Introduction



L1: Introduction



L1: Introduction

- Compilation croisée
  - lorsque la cible est une autre architecture de processeur que celle du poste de travail
    - Exemple compilation pour ARM depuis un PC x86
    - Exécution du binaire / exécutable
      - soit par émulation
        - qemu
      - soit en télécommande d'une carte de développement
        - Série / jtag
  - Orienté embarqué, mobiles, IoT

L1: Introduction

- Représentation intermédiaire (IR)
  - Représentation abstraite ou texte décrivant les instructions essentielles du langage / d'un exécutable
  - Peut-être commune à de multiple langages de programmation
  - Est indépendante de la machine cible
  - Exemples
    - GIMPLE (gcc)
    - LLVM-IR (LLVM)

L1: Introduction

- LLVM-IR

```
; Declare the string constant as a global constant.
@.str = private unnamed_addr constant [13 x i8] c"hello world\0A\00"

; External declaration of the puts function
declare i32 @puts(i8* nocapture) nounwind

; Definition of main function
define i32 @main() {   ; i32()*
  ; Convert [13 x i8]* to i8*...
  %cast210 = getelementptr [13 x i8], [13 x i8]* @.str, i64 0, i64 0

  ; Call puts function to write out the string to stdout.
  call i32 @puts(i8* %cast210)
  ret i32 0
}

; Named metadata
!0 = !{i32 42, null, !"string"}
!foo = !{!0}
```

L1: Introduction

- Objectif
  - Construire un front-end / interpréteur
    - Pour un langage de programmation simple
- Concepts de langages de programmation
    - A implémenter dans l'interpréteur
  - Scopes
  - Appels et structures de contrôle
  - Récursivité

L1: Introduction

- Exemple

```
static double voltage(long
SviVid) {
    if(SviVid <= 0x7F && SviVid
>= 0x7C)
        return 0;
    else return 1.550 - 0.0125 *
SviVid;
}
```

```
            vidToSet[pstateId] = vid;
            divToSet[pstateId] = div;
            if(verbose) printf("vid 0x%x/%d / %.4fV, div
%.02f to set for pstate %d\n", vid, vid, voltage(vid),
div, pstateId);
            break;
        default:
            if(optind != (argc -1)){
                fprintf(stderr, "Invalid argument %s\n",
argv[optind]);
                usage(argv[0]);
                exit(EXIT_FAILURE);
            }
            else {
                usage(argv[0]);
                exit(0);
            }
    }
```

L1: Introduction

- Scopes
  - La définition d'un symbole est limitée à l'unité dans laquelle il est défini (notion de portée / scope)
  - Les scopes peuvent être imbriqués (module / classe / fonction)
  - Un symbole peut être redéfini dans un scope intérieur

```
int i;
void f() {
    double i;
    ...
}
```

L1: Introduction

- Scopes
  - En fonction du langage, ça peut être contre-intuitif:

```
typedef int i;
void f() {
    i * i;
    ...
}
```

  - Cet exemple est valide...

L1: Introduction

- Scopes
  - Autre exemple

```
typedef int i;
void f(i i) {
    ...
}
```

  - Cet exemple est valide...
  - Que penser de:

```
int x(struct {int x, y;} point) { return point.x; } ;
```

L1: Introduction

- Scopes
  - Autre exemple

```
test.c:2:14: warning: anonymous struct declared inside parameter list
 int x(struct {int x, y;} point) { return point.x; } ;
              ^
test.c:2:14: warning: its scope is only this definition or declaration, which is
probably not what you want
```

  - Que penser de:

```
int x(struct {int x, y;} point) { return point.x; } ;
```

L1: Introduction

- Appels et structures de contrôle
  - Appel de fonction
    - transférer le contrôle (l'exécution) dans un autre bloc de code
    - Trouver ce bloc...
    - passer des paramètres, renommer des variables

```
int f(int x) { ... }

...
    int a = 3;
    y = f(a);
...
```

L1: Introduction

- Appels et structures de contrôle
  - Structure de contrôle
    - Choisir d'exécuter un code ou un autre (sur une condition)

```
if (x > y)
    max = x;
else
    max = y;
```

L1: Introduction

- Récursivité
  - Lorsqu'un code s'appelle lui-même

```
int f(int n) {
    if (n > 1)
        return n *f(n – 1);
    else
        return 1;
}
```

  - Implémenté via les scopes et appels (si ça a été fait correctement)

L1: Introduction

- L'essentiel des langages de programmation est implémentable avec ces fonctionalités
  - Objets
  - Classes
  - Fonctions virtuelles
  - Polymorphisme
- Manquant
  - Les abstractions des langages fonctionnels (closures)

L1: Introduction

- Exemples:
  - Numpy

    pairwise sum
  - C + @var@

```
static @type@
pairwise_sum_@TYPE@(char *a, npy_intp n, npy_intp stride)
{
    ...
    else if (n <= PW_BLOCKSIZE) {
        npy_intp i;
        @type@ r[8], res;
        ...
        r[0] = @trf@(*((@dtype@ *)(a + 0 * stride)));
        ...
        r[7] = @trf@(*((@dtype@ *)(a + 7 * stride)));

        for (i = 8; i < n - (n % 8); i += 8) {
            /* small blocksizes seems to mess with hardware prefetch */
            NPY_PREFETCH(a + (i + 512/(npy_intp)sizeof(@dtype@))*stride, 0, 3);
            r[0] += @trf@(*((@dtype@ *)(a + (i + 0) * stride)));
            ...
            r[7] += @trf@(*((@dtype@ *)(a + (i + 7) * stride)));
        }
       ...
        return res;
    }
    else {
        /* divide by two but avoid non-multiples of unroll factor */
        npy_intp n2 = n / 2;

        n2 -= n2 % 8;
        return pairwise_sum_@TYPE@(a, n2, stride) +
               pairwise_sum_@TYPE@(a + n2 * stride, n - n2, stride);
    }
}
```

L1: Introduction

- Exemples:
  - Numpy

    pairwise sum
  - Une fonction C + @var@
  - Plusieurs fonctions C
  -

```
static npy_float
pairwise_sum_FLOAT(char *a, npy_intp n, npy_intp stride)
{
    ...
}

static npy_double
pairwise_sum_DOUBLE(char *a, npy_intp n, npy_intp stride)
{
    ...
    r[7] = (*((npy_double *)(a + 7 * stride)));

    for (i = 8; i < n - (n % 8); i += 8) {
        /* small blocksizes seems to mess with hardware prefetch */
        NPY_PREFETCH(a + (i + 512/(npy_intp)sizeof(npy_double))*stride, 0, 3);
        r[0] += (*((npy_double *)(a + (i + 0) * stride)));
    ...
}

static npy_longdouble
pairwise_sum_LONGDOUBLE(char *a, npy_intp n, npy_intp stride)
{
    ...
```

L1: Introduction

- Exemples:
  - Numpy

    pairwise sum
  - Une fonction C + @var@
  - Plusieurs fonctions C
  - Présence dans le binaire (.o)

```
00000000000032d0 <pairwise_sum_FLOAT>:
    32d0:       41 57                   push   %r15
    32d2:       41 56                   push   %r14
    32d4:       41 55                   push   %r13
    32d6:       41 54                   push   %r12
    32d8:       49 89 d4                mov    %rdx,%r12
    32db:       55                      push   %rbp
    32dc:       48 89 fd                mov    %rdi,%rbp
    32df:       53                      push   %rbx
    32e0:       48 89 f3                mov    %rsi,%rbx
    32e3:       48 83 ec 18             sub    $0x18,%rsp
    32e7:       48 83 fe 07             cmp    $0x7,%rsi
    32eb:       0f 8e 6f 01 00 00       jle    3460 <pairwise_sum_FLOAT+0x190>
    32f1:       48 81 fe 80 00 00 00    cmp    $0x80,%rsi
    32f8:       0f 8f 12 01 00 00       jg     3410 <pairwise_sum_FLOAT+0x140>
    32fe:       48 8d 04 17             lea    (%rdi,%rdx,1),%rax
    3302:       f3 0f 10 0f             movss  (%rdi),%xmm1
    3306:       48 83 e6 f8             and    $0xfffffffffffffff8,%rsi
    330a:       4f 8d 04 a4             lea    (%r12,%r12,4),%r8
    330e:       f3 0f 10 00             movss  (%rax),%xmm0
    3312:       48 01 d0                add    %rdx,%rax
    3315:       48 8d 14 12             lea    (%rdx,%rdx,1),%rdx
    3319:       f3 0f 10 20             movss  (%rax),%xmm4
```

L1: Introduction

- Exemples:
  - Numpy

    pairwise sum
  - Une fonction C + @var@
  - Plusieurs fonctions C
  - Présence dans le binaire (.o)

```
342e:       e8 9d fe ff ff          callq  32d0 <pairwise_sum_FLOAT>
3433:       49 8d 74 1d 00          lea    0x0(%r13,%rbx,1),%rsi
3438:       4c 89 e2                mov    %r12,%rdx
343b:       f3 0f 11 44 24 0c       movss  %xmm0,0xc(%rsp)
3441:       4a 8d 7c 35 00          lea    0x0(%rbp,%r14,1),%rdi
3446:       e8 85 fe ff ff          callq  32d0 <pairwise_sum_FLOAT>
344b:       f3 0f 58 44 24 0c       addss  0xc(%rsp),%xmm0
3451:       48 83 c4 18             add    $0x18,%rsp
3455:       5b                      pop    %rbx
3456:       5d                      pop    %rbp
3457:       41 5c                   pop    %r12
3459:       41 5d                   pop    %r13
345b:       41 5e                   pop    %r14
345d:       41 5f                   pop    %r15
345f:       c3                      retq
3460:       48 85 f6                test   %rsi,%rsi
3463:       7e 3b                   jle    34a0 <pairwise_sum_FLOAT+0x1d0>
3465:       66 0f ef c0             pxor   %xmm0,%xmm0
3469:       31 c0                   xor    %eax,%eax
346b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
3470:       48 83 c0 01             add    $0x1,%rax
3474:       f3 0f 58 07             addss  (%rdi),%xmm0
3478:       4c 01 e7                add    %r12,%rdi
```