Introduction to Compiler Design

Compiling a C Program

prog1.c

```
int f(int a, int b) {
    return a + b ;
}
```

prog2.c

```
int f(int,int) ;

int main() {
    printf("%d\n", f(3,4)) ;
}
```

```
$ gcc -o prog prog1.c prog2.c

$ prog
7
```

Compiling a C Program

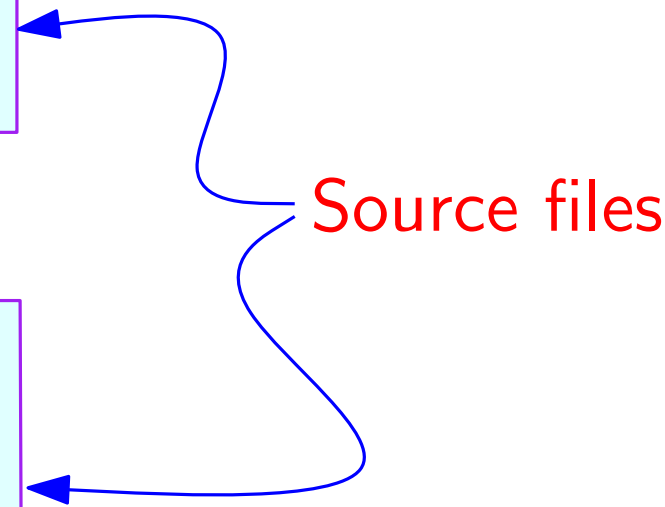
prog1.c

```
int f(int a, int b) {
    return a + b ;
}
```

prog2.c

```
int f(int,int) ;

int main() {
    printf("%d\n", f(3,4)) ;
}
```




```
$ gcc -o prog prog1.c prog2.c

$ prog
7
```

Compiling a C Program

prog1.c

```
int f(int a, int b) {
    return a + b ;
}
```

prog2.c

```
int f(int,int) ;

int main() {
    printf("%d\n", f(3,4)) ;
}
```

Source files

Compilation command

```
$ gcc -o prog prog1.c prog2.c

$ prog
7
```

Compiling a C Program

prog1.c

```
int f(int a, int b) {
    return a + b ;
}
```

prog2.c

```
int f(int,int) ;

int main() {
    printf("%d\n", f(3,4)) ;
}
```

Source files

Compilation command

```
$ gcc -o prog prog1.c prog2.c
```

Executable

```
$ prog
7
```

The Hidden Side

GCC is just a wrapper!!!

The Hidden Side

GCC is just a wrapper!!!

```
$ gcc -v -o prog prog1.c prog2.c
```

```
$ cc1 prog1.c -o /tmp/xyz.S
$ cc1 prog2.c -o /tmp/wxz.S
$ as /tmp/xyz.S -o xyz.o
$ as /tmp/wxz.S -o wxz.o
$ collect2 -o prog -dynamic-linker /lib/ld-linux.so \
    /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/crtn.o \
    /usr/lib/gcc/i686-redhat-linux/4.6.0/crtbegin.o \
    /usr/lib/gcc/i686-redhat-linux/4.6.0/crtend.o \
    -L/usr/lib/gcc/i686-redhat-linux/4.6.0/ \
    -L/usr/lib -lgcc_s -lc -lgcc xyz.o wxz.o
```

The Hidden Side

GCC is just a wrapper!!!

Verbose: print all the commands issued in the background!

```
$ gcc -v -o prog prog1.c prog2.c
```

Compile

Assemble

Link

```
$ cc1 prog1.c -o /tmp/xyz.S
$ cc1 prog2.c -o /tmp/wxz.S
$ as /tmp/xyz.S -o xyz.o
$ as /tmp/wxz.S -o wxz.o
$ collect2 -o prog -dynamic-linker /lib/ld-linux.so \
    /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/crtn.o \
    /usr/lib/gcc/i686-redhat-linux/4.6.0/crtbegin.o \
    /usr/lib/gcc/i686-redhat-linux/4.6.0/crtend.o \
    -L/usr/lib/gcc/i686-redhat-linux/4.6.0/ \
    -L/usr/lib -lgcc_s -lc -lgcc xyz.o wxz.o
```

The Hidden Side

GCC is just a wrapper!!!

Verbose: print all the commands issued in the background!

```
$ gcc -v -o prog prog1.c prog2.c
```

Temporary assembly language output

Compile

Assemble

Link

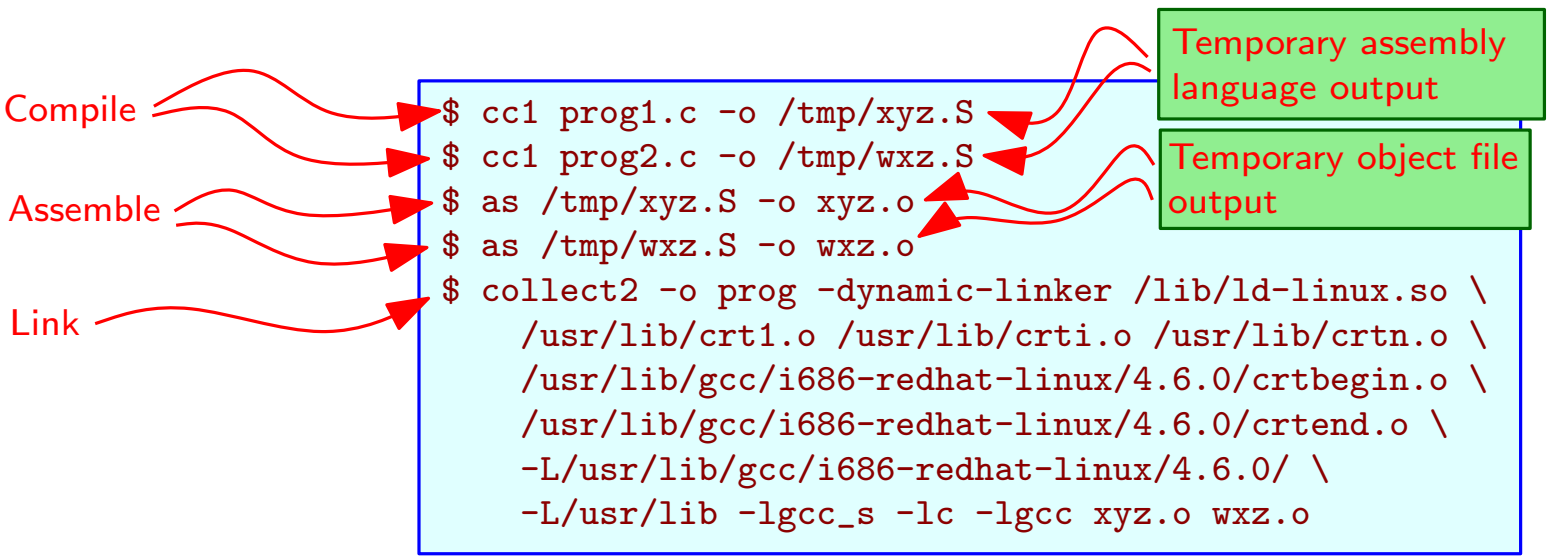
```
$ cc1 prog1.c -o /tmp/xyz.S
$ cc1 prog2.c -o /tmp/wxz.S
$ as /tmp/xyz.S -o xyz.o
$ as /tmp/wxz.S -o wxz.o
$ collect2 -o prog -dynamic-linker /lib/ld-linux.so \
    /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/crtn.o \
    /usr/lib/gcc/i686-redhat-linux/4.6.0/crtbegin.o \
    /usr/lib/gcc/i686-redhat-linux/4.6.0/crtend.o \
    -L/usr/lib/gcc/i686-redhat-linux/4.6.0/ \
    -L/usr/lib -lgcc_s -lc -lgcc xyz.o wxz.o
```

The Hidden Side

GCC is just a wrapper!!!

Verbose: print all the commands issued in the background!

```
$ gcc -v -o prog prog1.c prog2.c
```

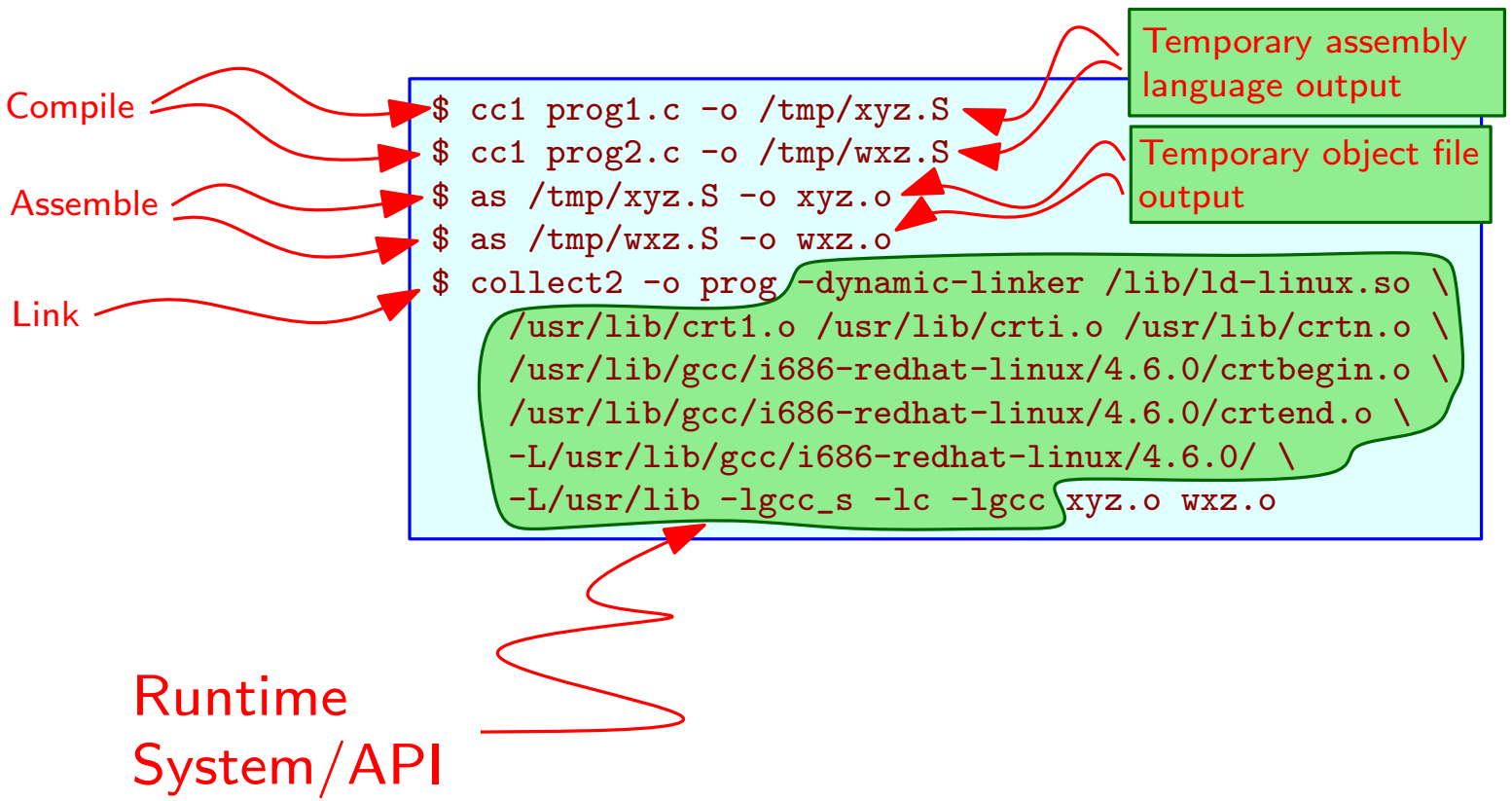


The Hidden Side

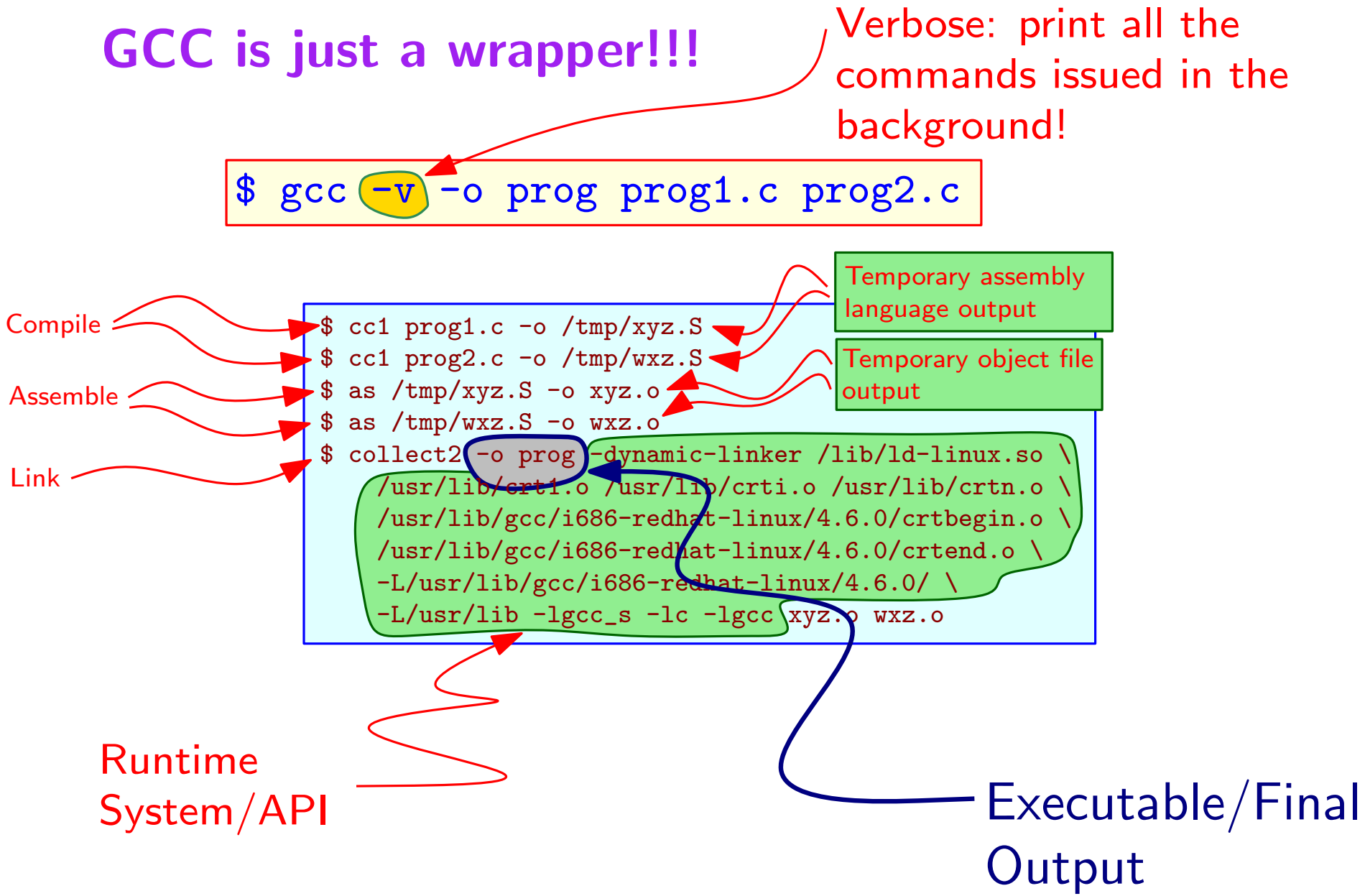
GCC is just a wrapper!!!

Verbose: print all the commands issued in the background!

```
$ gcc -v -o prog prog1.c prog2.c
```



The Hidden Side



From C to Assembly

prog1.c

```
int f(int a, int b) {
    return a + b ;
}
```

```
    .file    "prog1.c"
    .text
    .globl    f
    .type    f, @function
f:
    pushl    %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    movl    8(%ebp), %edx
    addl    %edx, %eax
    popl    %ebp
    ret
    .size    f, .-f
    .section    .note.GNU-stack,"",@progbits
```

xyz.S

prog2.c

```
int f(int,int) ;

int main() {
    printf("%d\n", f(3,4)) ;
}
```

```
    .file    "prog2.c"
    .section    .rodata
.LC0:
    .string    "%d\n"
    .text
    .globl    main
    .type    main, @function
main:
    pushl    %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $4, 4(%esp)
    movl    $3, (%esp)
    call    f
    movl    %eax, 4(%esp)
    movl    $.LC0, (%esp)
    call    printf
    leave
    ret
    .size    main, .-main
    .section    .note.GNU-stack,"",@progbits
```

wxy.S

From C to Assembly

prog1.c

```
int f(int a, int b) {
    return a + b ;
}
```

```
    .file    "prog1.c"
    .text
    .globl    f
    .type    f, @function
f:
    pushl    %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    movl    8(%ebp), %edx
    addl    %edx, %eax
    popl    %ebp
    ret
    .size    f, .-f
    .section    .note.GNU-stack,"",@progbits
```

xyz.S

The compiler only translates from C to assembly language!

prog2.c

```
int f(int,int) ;

int main() {
    printf("%d\n", f(3,4)) ;
}
```

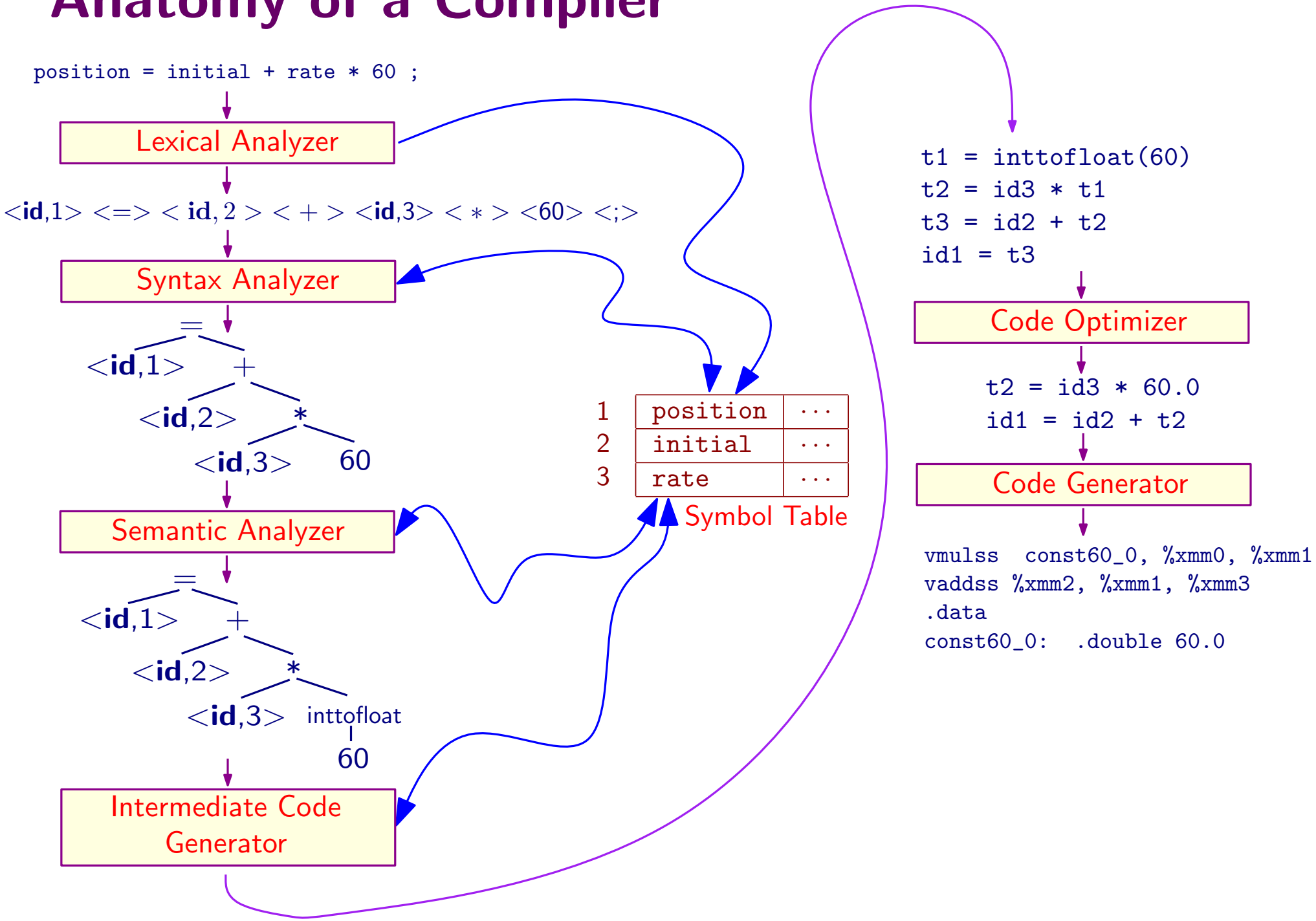
```
    .file    "prog2.c"
    .section    .rodata
.LC0:
    .string    "%d\n"
    .text
    .globl    main
    .type    main, @function
main:
    pushl    %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $4, 4(%esp)
    movl    $3, (%esp)
    call    f
    movl    %eax, 4(%esp)
    movl    $.LC0, (%esp)
    call    printf
    leave
    ret
    .size    main, .-main
    .section    .note.GNU-stack,"",@progbits
```

wxy.S

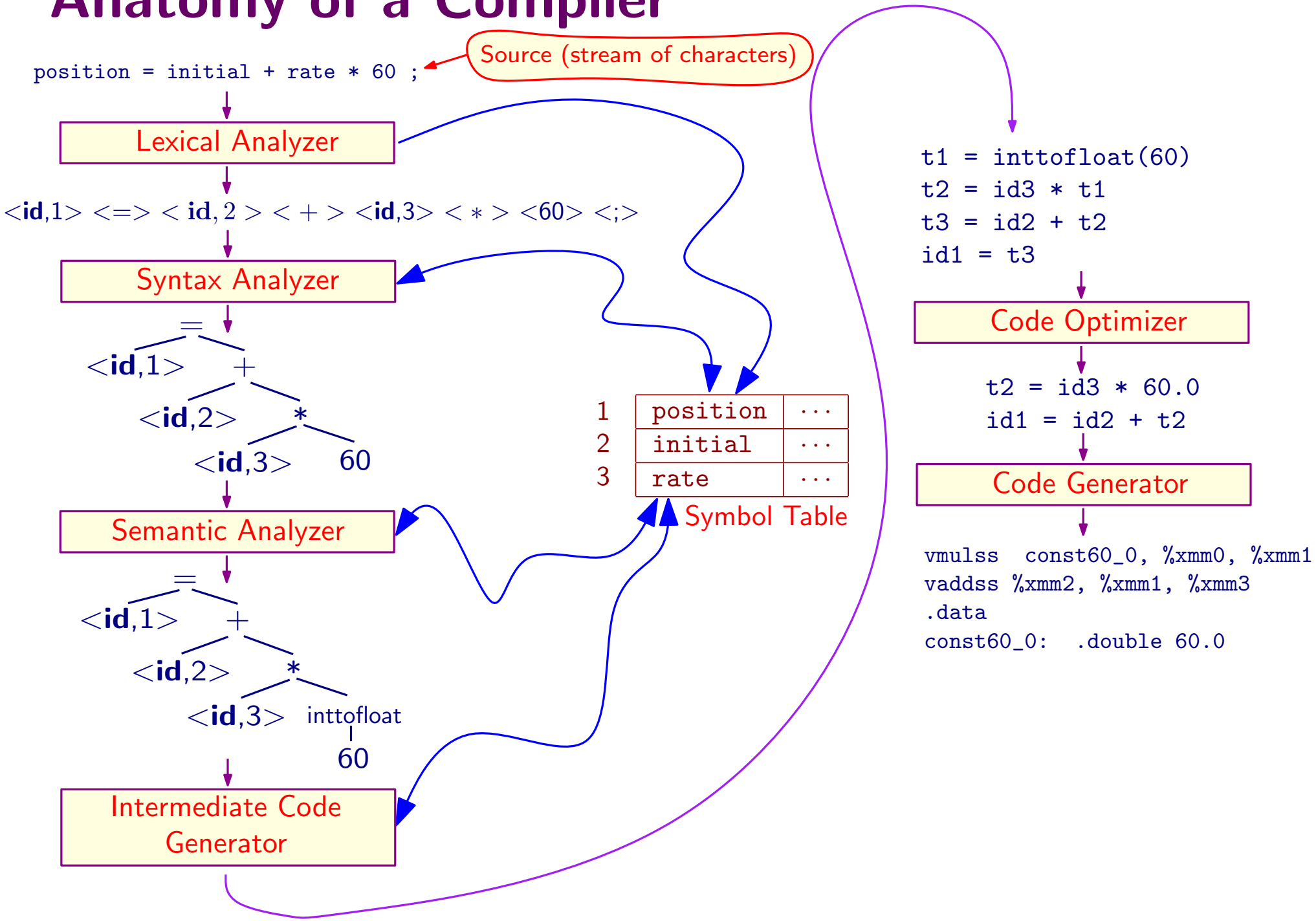
Compilation & Assembly Language Programming

- The compiler is a tool that translates from a high level programming into the assembly language of the system at hand.
- The assembler is a tool usually provided by the processor designer/manufacturer; it is consider part of the programming environment of that language, but not part of the compiler per se.
- The assembly language output could have been written by the programmer by hand
  - no magic; not indispensable in creating executables;
  - increased productivity, portability
- Object/executable file formats are in the realm of computer architecture/operating systems
  - An overview of these topics will be provided as a video recording
- The linker is usually provided by operating system implementer.
  - It usually handles multiple object, library, and executable formats
- To produce executables, the compiler implementer relies on a *runtime system*
  - A system of libraries/system services that the linker can merge into the executable.
  - Sometime dependent on the operating system (example: C on Linux vs Windows)
- Many programming environments for a given language provide an entire suite of compiler, assembler, linker, runtime
  - Easier to maintain the compatibility between components as their versions evolve.
  - Allows a variety of extensions that provide a competitive advantage.

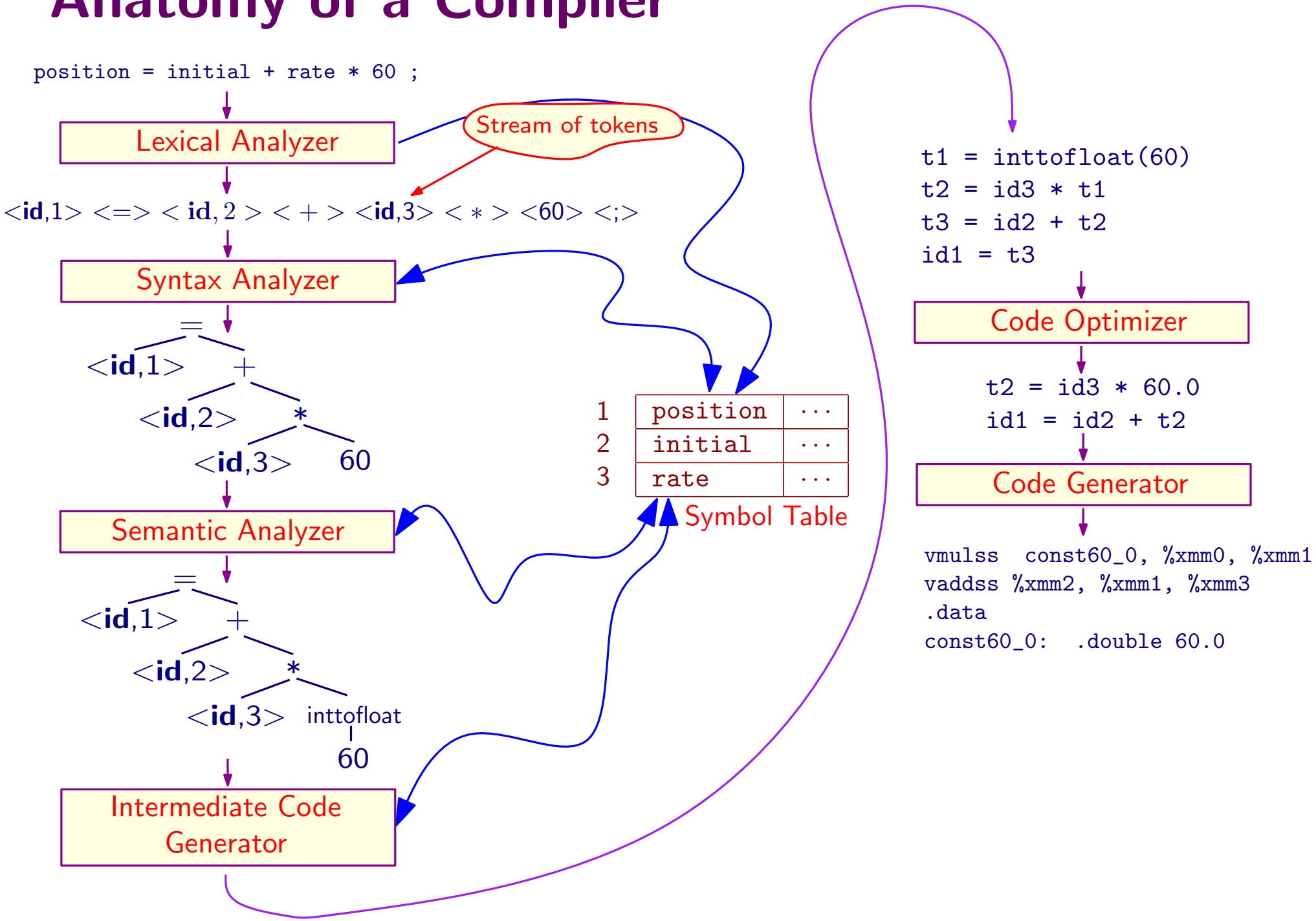
Anatomy of a Compiler



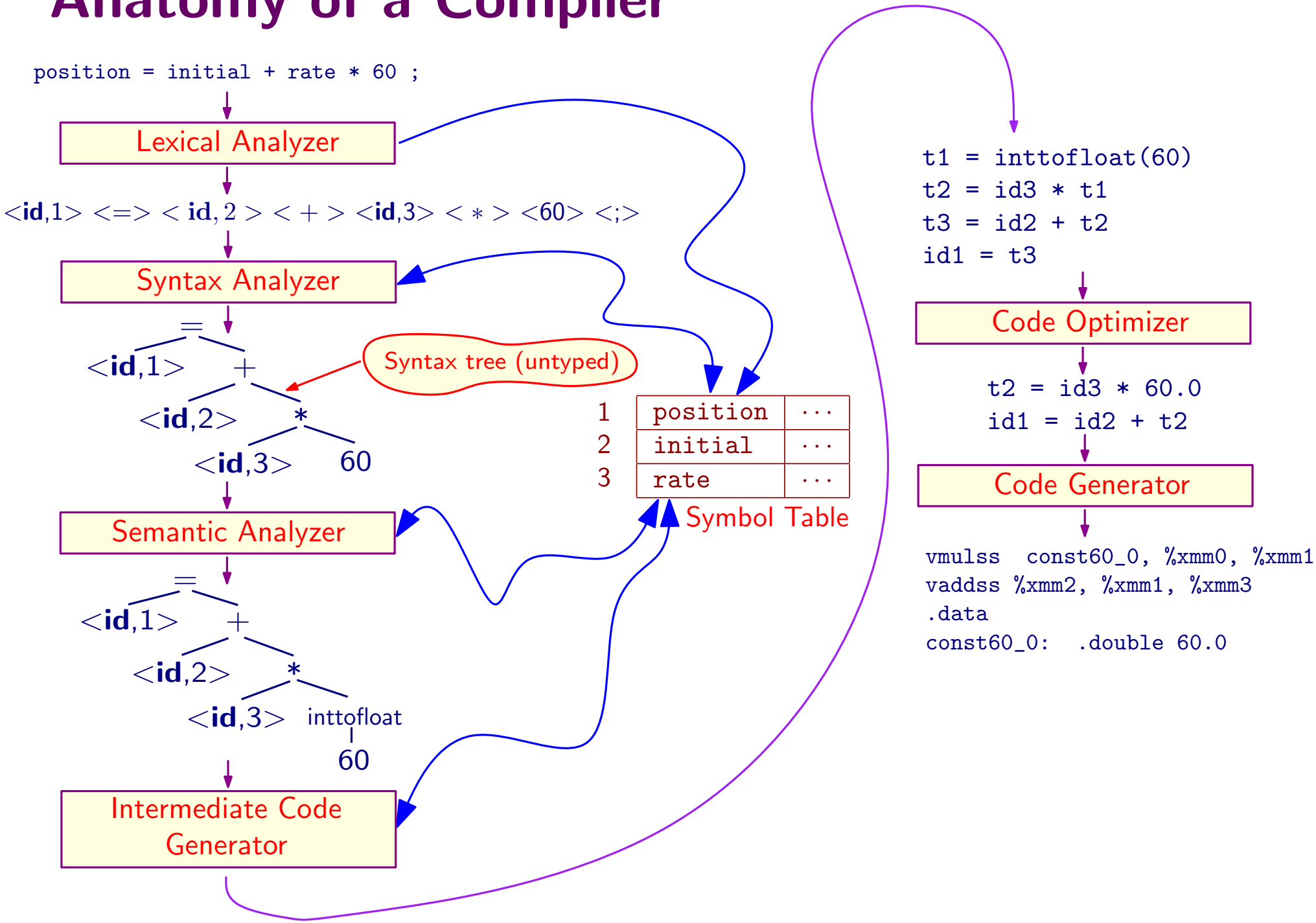
Anatomy of a Compiler



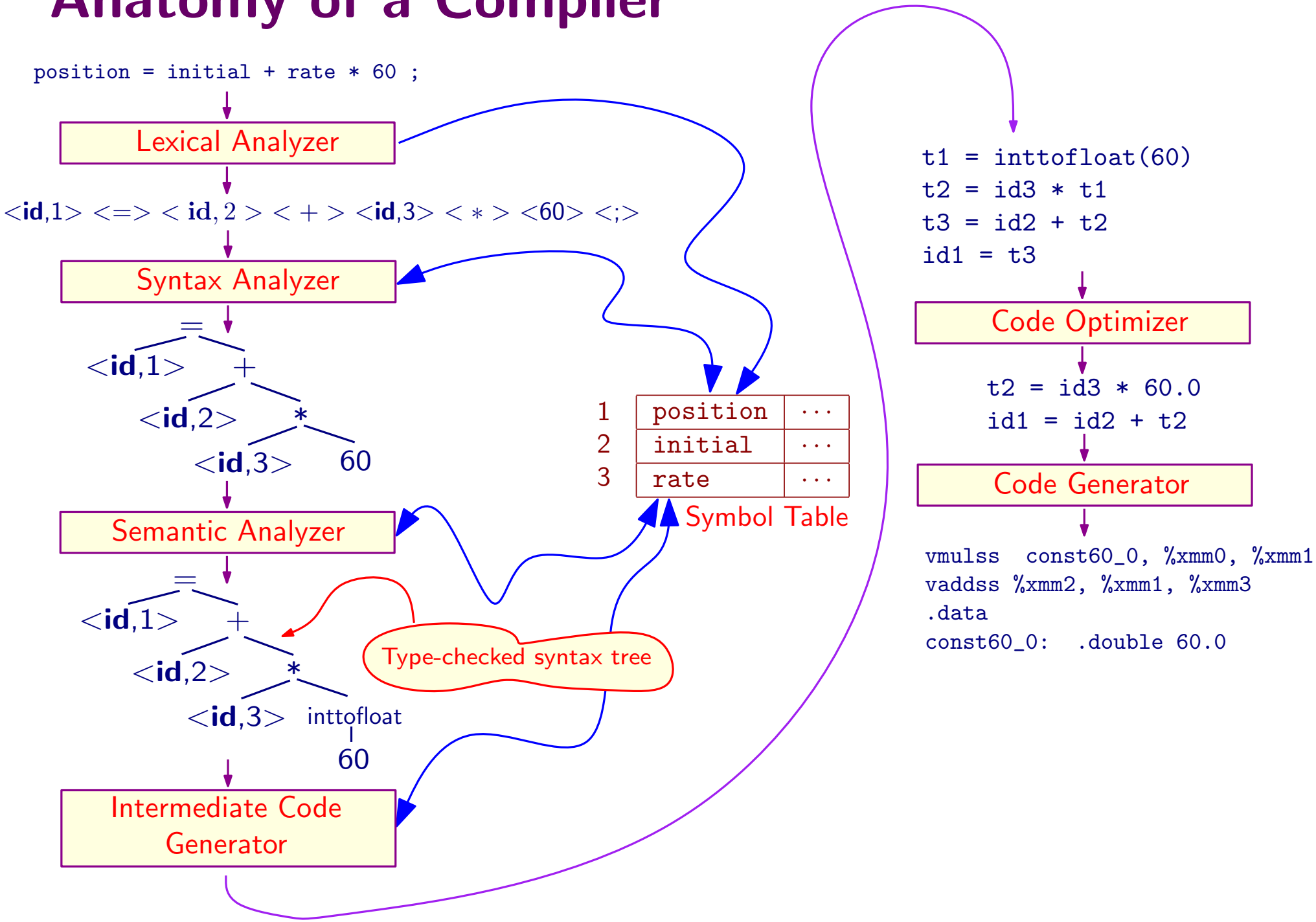
Anatomy of a Compiler



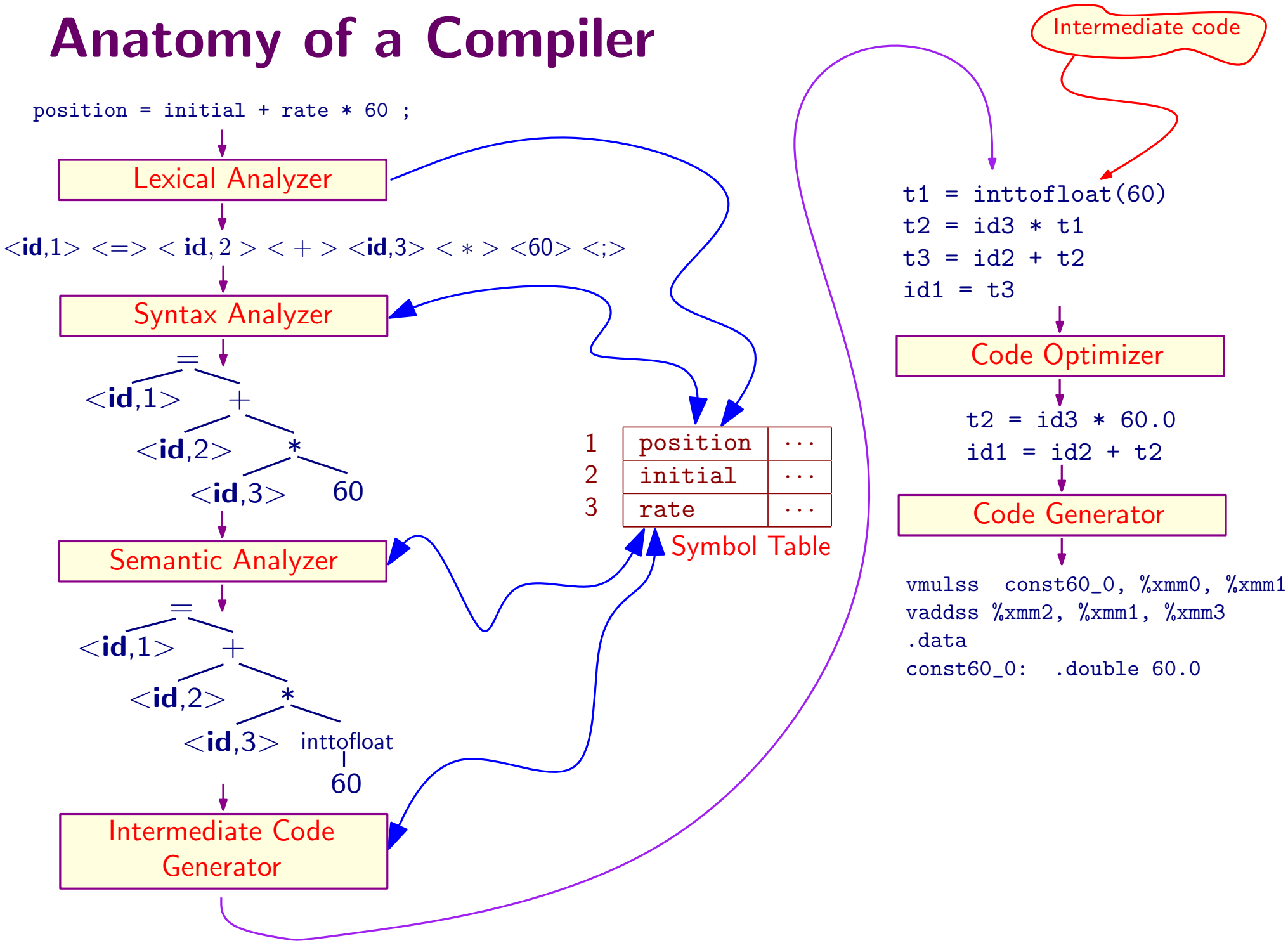
Anatomy of a Compiler



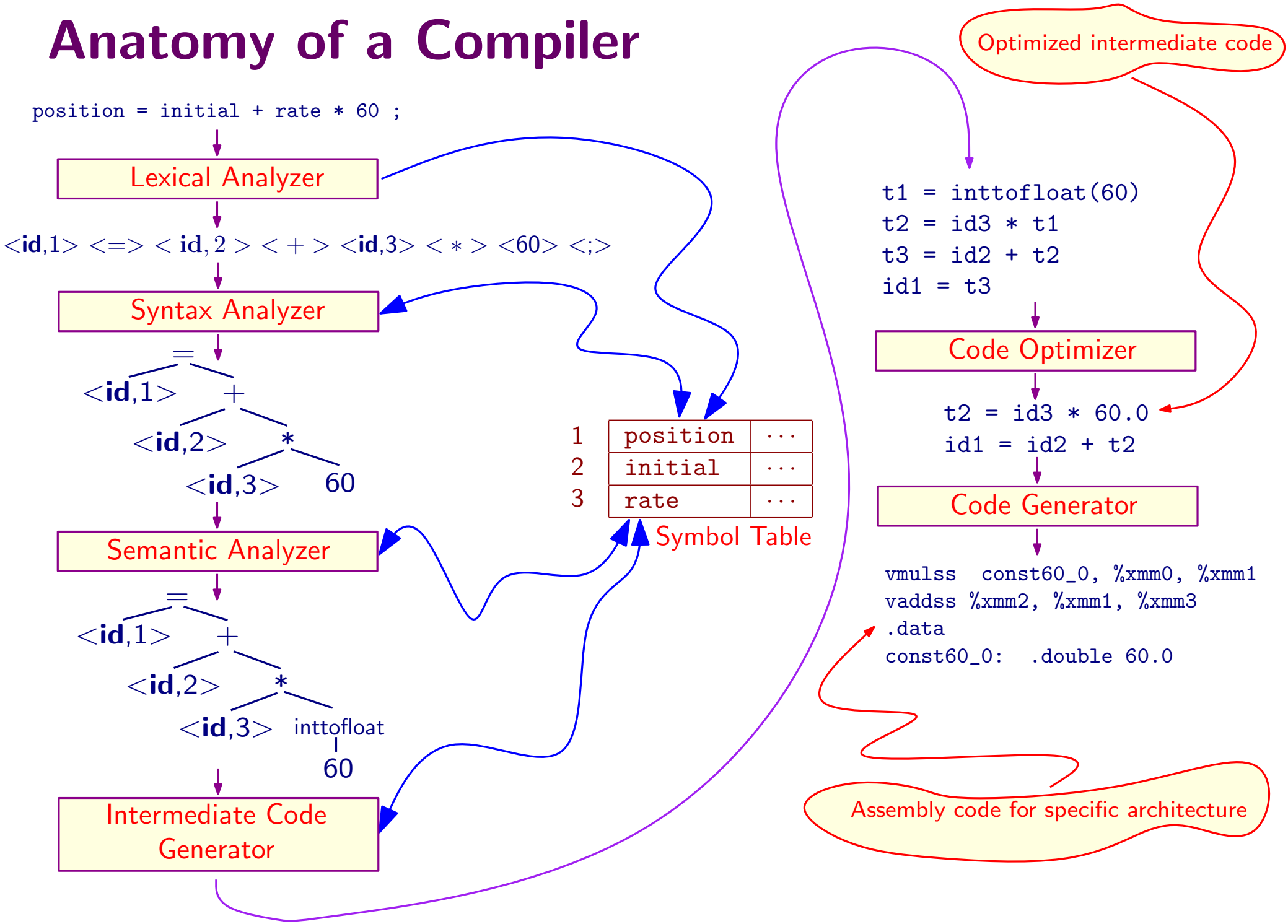
Anatomy of a Compiler



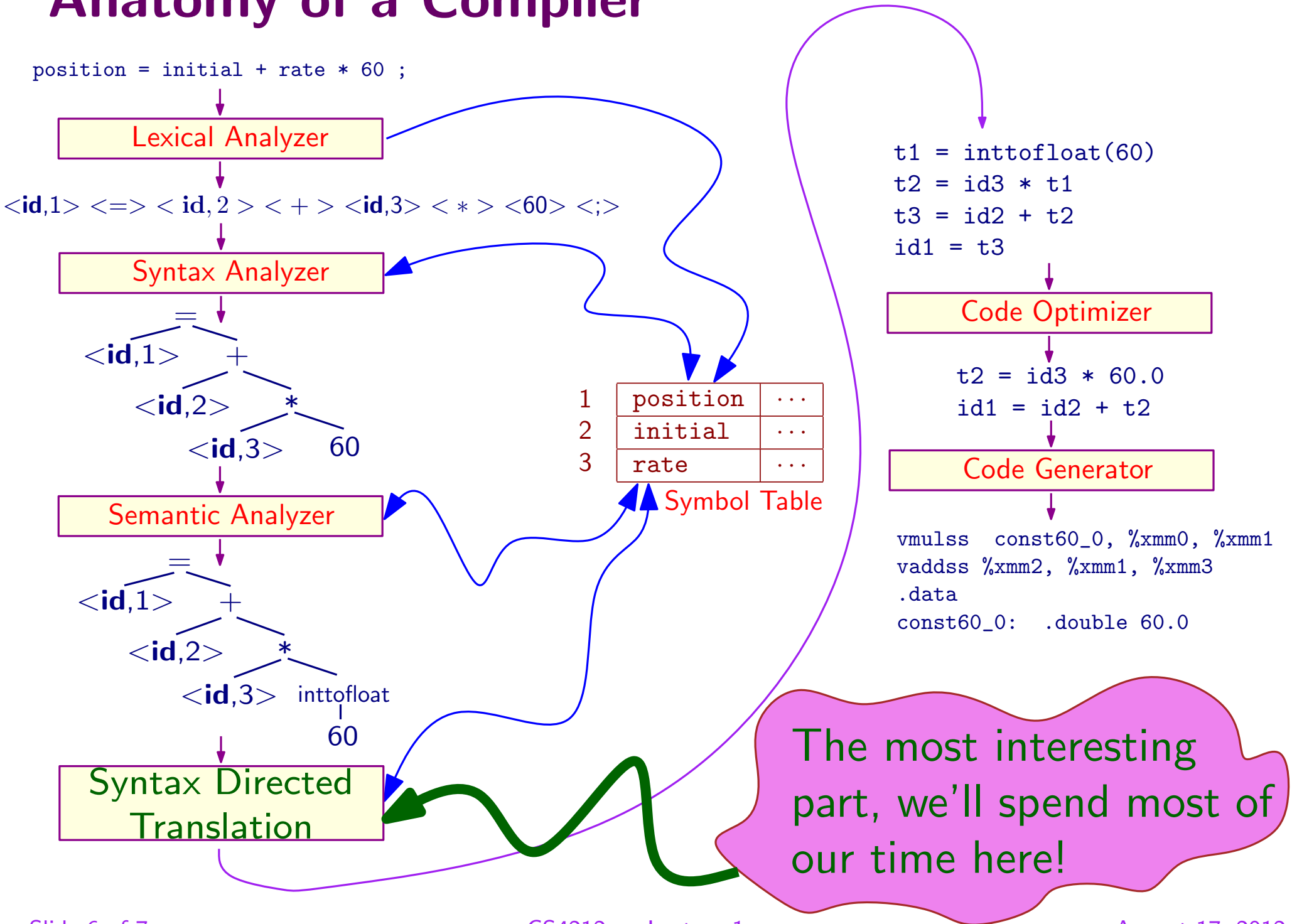
Anatomy of a Compiler



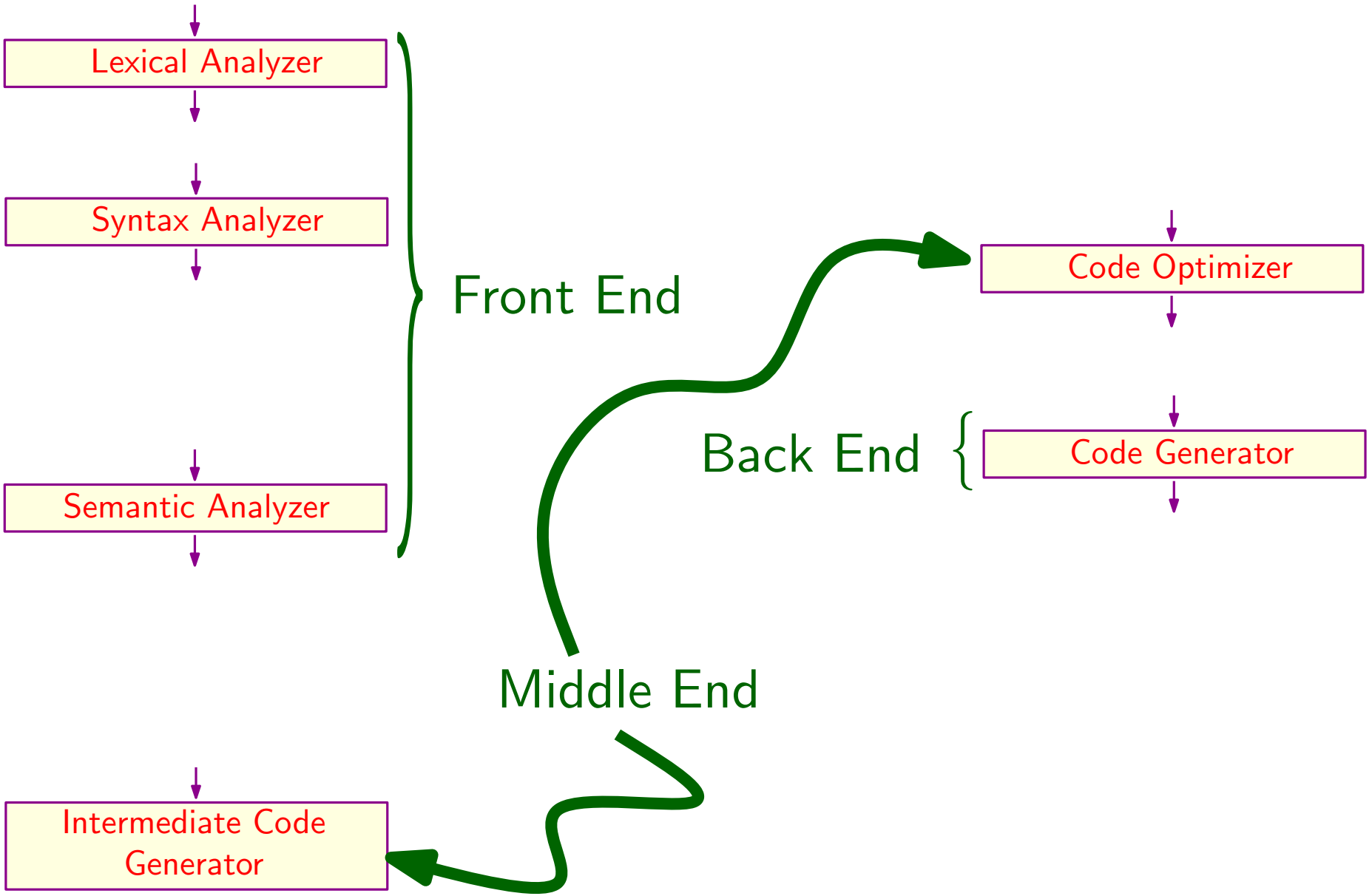
Anatomy of a Compiler



Anatomy of a Compiler



Anatomy of a Compiler



Our Approach

- Circumvent the need of a front end by using Prolog for implementation
  - Will revisit the topic of lexical and syntactic analysis towards the end of the module

- Devote most effort towards understanding the concept of syntax directed translation.
  - The most transferrable skill that you will gain in this module.
  - Develop a toy compiler capable of translating most common programming constructs for procedural and object-oriented languages.

- Generate directly Pentium code (no intermediate code).
  - An overview of the topic of Intermediate Code, and it's use inside GCC, will be given as a video tutorial.

- Output is an AL file that can be assembled into an executable.
  - Allows to check that our output is correct.

- Optimizations will be understood by comparison with GCC, without the need to implement them.