

# CSE110A: Compilers

April 5, 2023

- *Compiler Overview*

- What is a compiler
- What are the different stages of a compiler
  - Frontend
  - Intermediate
  - Backend

# Announcements

- Friday will be an asynchronous lecture (recorded from last year)
- Homework 1 will be released either Monday or Tuesday
- Piazza is up; please enroll!
- Coming soon:
  - Docker setup instructions on the website
  - TA and tutor office hours





# Announcements

- Did anyone set up a discord?

# Quiz

# Background

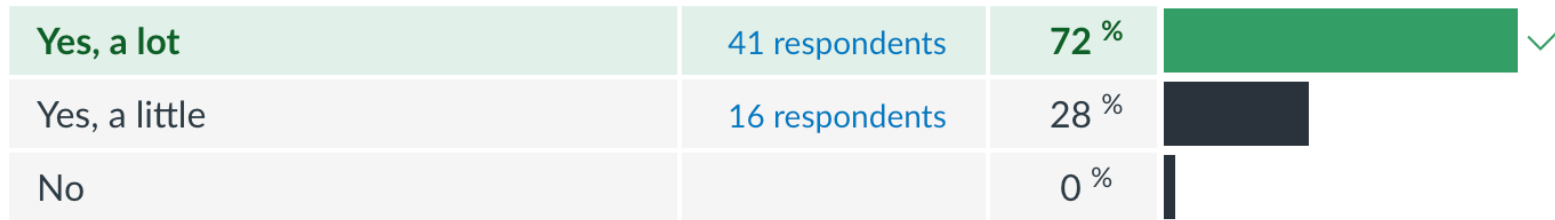
So I can get a better sense of the backgrounds in this class, please select all the classes you have taken:

CSE 103	11 respondents	22 %	
CSE 120	45 respondents	90 %	
CSE 130	29 respondents	58 %	
No Answer	7 respondents	14 %	

2% answered correctly

# Background

Have you ever programmed in Python before?



Discrimination Index



72% answered  
correctly

It is worthwhile to learn!

<https://www.tiobe.com/tiobe-index/>

# What do people hope to get out of this class?

A few answers that I liked:

- “I hope to understand how the compiler works rather than simply accept that it does”
- “learning how my code is translated will make me a better programmer”
- “I’m interested in learning more about docker and python and computer science in general”
- “I want to learn about AI compilers”

# Quiz

- Thank you for all your thoughtful answers!



# Schedule

- Introduction to compilers
- Compiler architecture

# Schedule

- **Introduction to compilers**
- Compiler architecture

*What is a compiler?*

Let's discuss

*What are some of your favorite compilers*

Let's discuss

```

1  ---
2  title: "Fundamentals of Compiler Design"
3  layout: single
4  ---
5
6
7  ### Welcome to **CSE110A:** _Fundamentals of Compiler Design_, Spring 2022 Quarter at UCSC!
8
9  - **Instructor:** \[Tyler Sorensen\](https://users.soe.ucsc.edu/~tsorensen/)
10 - **Time:** Mondays, Wednesdays and Fridays: 4:00 – 5:05 pm
11 - **Location:** Porter 144
12
13 Hello and welcome to the fundamentals of compiler design class!
14
15 In this class you will learn about compiler design and implementation. In the abstract, compilers explore many of the \[foundational problems in computer science\](https://en.wikipedia.org/wiki/Halting_problem). In practice, compilers are \[massive pieces of well-oiled software\](https://www.phoronix.com/scan.php?page=news_item&px=MTg30TQ), and are some of the engineering marvels of the modern world.
16
17 _COVID Note_ : The last few years have been difficult due to the COVID pandemic. Public health concerns and policies remain volatile. The first priority in this class is your health and well-being. We will approach any challenges that arise with compassion and understanding. I expect that you will do the same, both to the teaching staff and to your classmates. We will follow university guidelines and work together to have a productive and fun quarter.
18

```

Home Overview Schedule References

## Fundamentals of Compiler Design

**Welcome to CSE110A: *Fundamentals of Compiler Design*, Spring 2023 Quarter at UCSC!** 🔗

- **Instructor:** [Tyler Sorensen](#)
- **Time:** Mondays, Wednesdays and Fridays: 9:20 – 10:25 AM
- **Location:** Merrill Acad 102

Hello and welcome to the fundamentals of compiler design class!

In this class you will learn about compiler design and implementation. In the abstract, compilers explore many of the

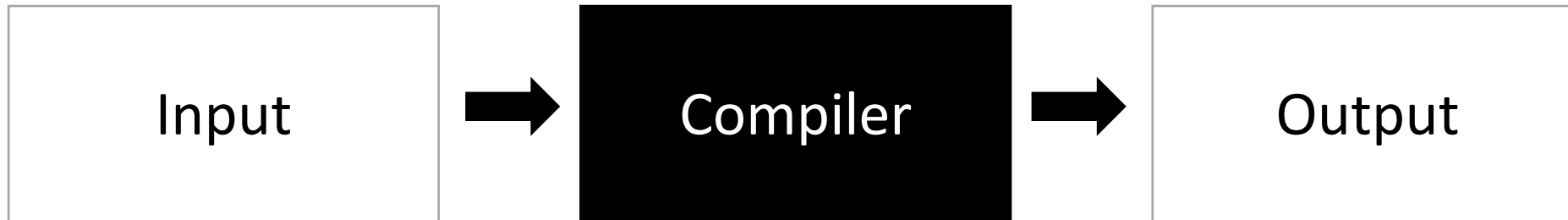
Building this website started with:

- Markdown to describe the page
- compiled with Jekyll to a static webpage
- static webpage is in HTML and javascript

*What is a compiler?*

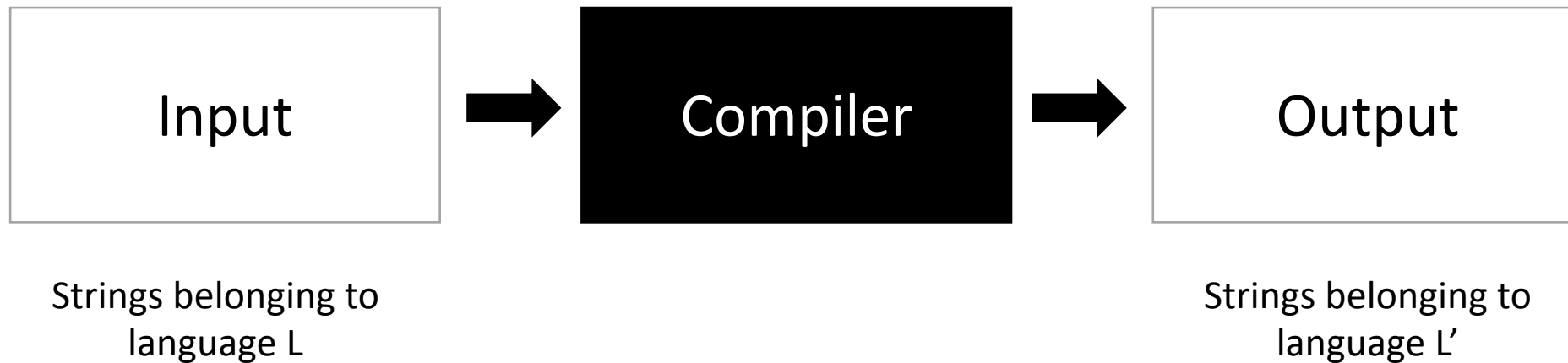


# *What is a compiler?*



This is way too general to be useful  
Any program fits this description.

# *What is a compiler?*



A theoretical answer



```

1  ---
2  title: "Fundamentals of Compiler Design"
3  layout: single
4  ---
5
6
7  ### Welcome to **CSE110A:** _Fundamentals of Compiler Design_, Spring 2022 Quarter at UCSC!
8
9  - **Instructor:** \[Tyler Sorensen\](https://users.soe.ucsc.edu/~tsorensen/)
10 - **Time:** Mondays, Wednesdays and Fridays: 4:00 – 5:05 pm
11 - **Location:** Porter 144
12
13 Hello and welcome to the fundamentals of compiler design class!
14
15 In this class you will learn about compiler design and implementation. In the abstract, compilers explore many of the \[foundational problems in computer science\](https://en.wikipedia.org/wiki/Halting_problem). In practice, compilers are \[massive pieces of well-oiled software\](https://www.phoronix.com/scan.php?page=news_item&px=MTg30TQ), and are some of the engineering marvels of the modern world.
16
17 _COVID Note_ : The last few years have been difficult due to the COVID pandemic. Public health concerns and policies remain volatile. The first priority in this class is your health and well-being. We will approach any challenges that arise with compassion and understanding. I expect that you will do the same, both to the teaching staff and to your classmates. We will follow university guidelines and work together to have a productive and fun quarter.
18

```

Home Overview Schedule References

## Fundamentals of Compiler Design

**Welcome to CSE110A: *Fundamentals of Compiler Design*, Spring 2023 Quarter at UCSC!** [🔗](#)

- **Instructor:** [Tyler Sorensen](#)
- **Time:** Mondays, Wednesdays and Fridays: 9:20 – 10:25 AM
- **Location:** Merrill Acad 102

Hello and welcome to the fundamentals of compiler design class!

In this class you will learn about compiler design and implementation. In the abstract, compilers explore many of the

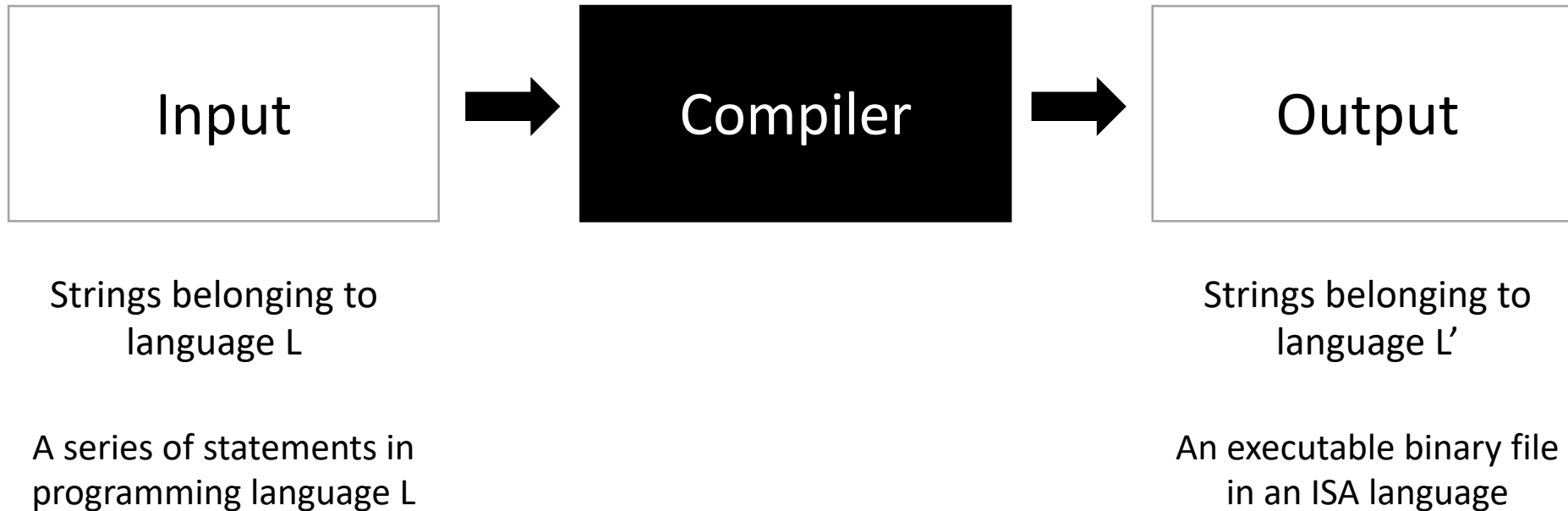
Building this website started with:

- Markdown to describe the page
- compiled with Jekyll to a static webpage
- static webpage is in HTML and javascript

# This would be a compiler

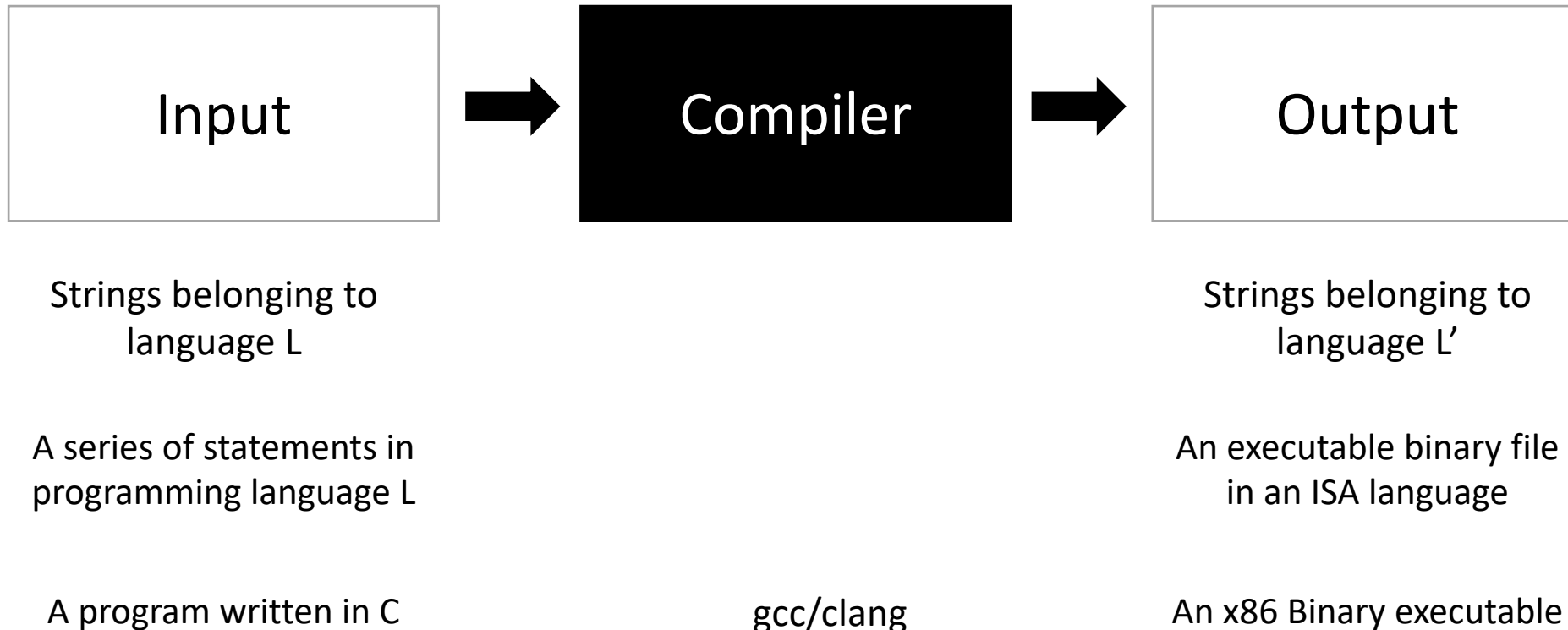
# *What is a compiler?*

A more traditional description  
What are some examples here?



# *What is a compiler?*

## A classic example



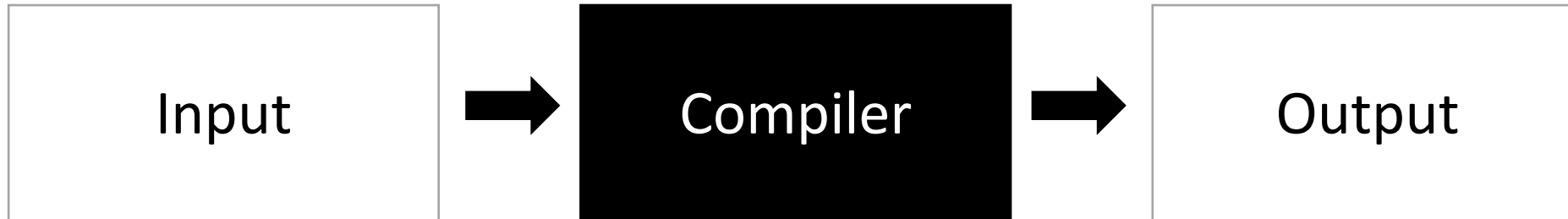
# GCC and Clang

- Two mainstream compiler frameworks
- Similarities and differences?

# *What is a compiler?*

```
int main() {  
    printf("hello world\n");  
}
```

gcc main.c



Strings belonging to  
language L

A series of statements in  
programming language L

A program written in C

Compiler

gcc/clang

Output

Strings belonging to  
language L'

An executable binary file  
in an ISA language

An x86 Binary executable

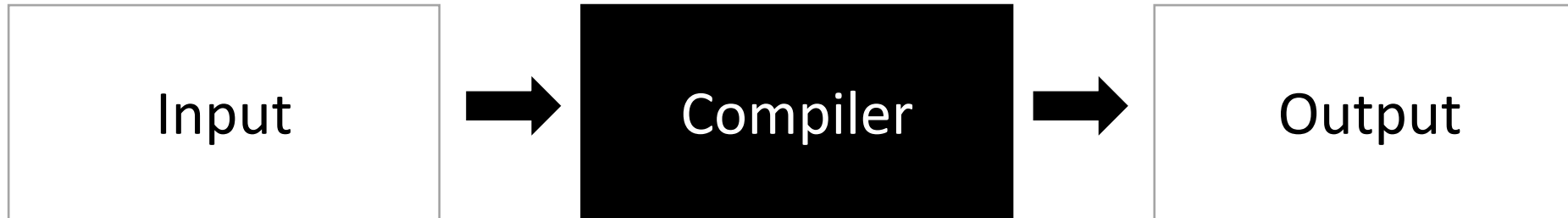
# What is a compiler?

*What is wrong with this picture?*

```
int main() {  
    printf("hello world\n");  
}
```

```
$ ./a.out  
hello CSE 110A
```

gcc main.c



Strings belonging to  
language L

A series of statements in  
programming language L

A program written in C

Strings belonging to  
language L'

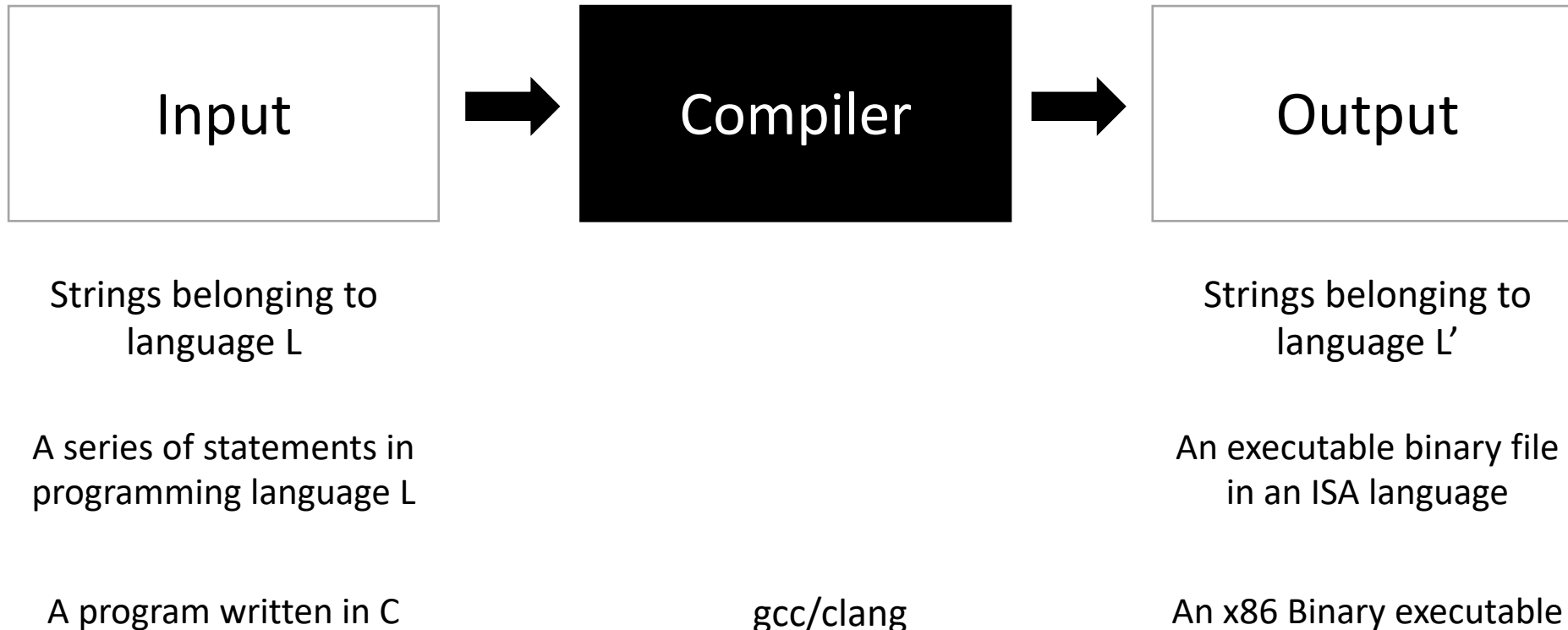
An executable binary file  
in an ISA language

An x86 Binary executable

gcc/clang

# What is a compiler?

**A valid input must have a equivalent valid output.**  
*Semantic equivalence*



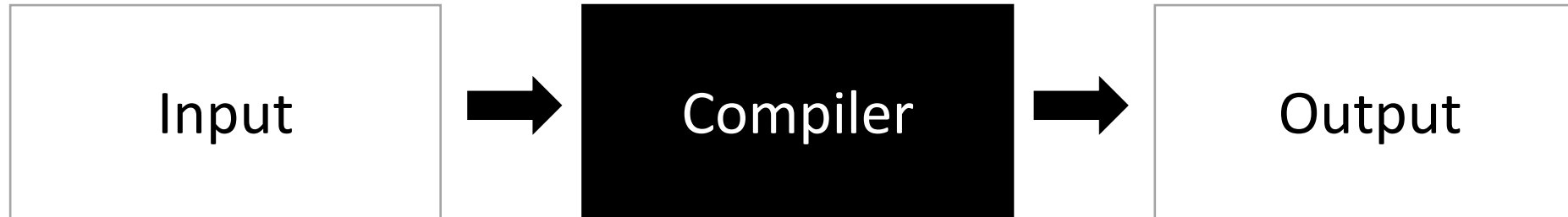
# What is a compiler?

*What is wrong with this picture?*

```
int main() {  
    printf("hello world\n");  
}
```

```
$ ./a.out  
hello CSE 110A
```

gcc main.c



Strings belonging to  
language L

A series of statements in  
programming language L

A program written in C

Strings belonging to  
language L'

An executable binary file  
in an ISA language

An x86 Binary executable

gcc/clang



# What is a compiler?

```
int main() {  
    printf("hello world\n");  
}
```

```
$ ./a.out  
hello world
```

gcc main.c



Strings belonging to  
language L

A series of statements in  
programming language L

A program written in C

Strings belonging to  
language L'

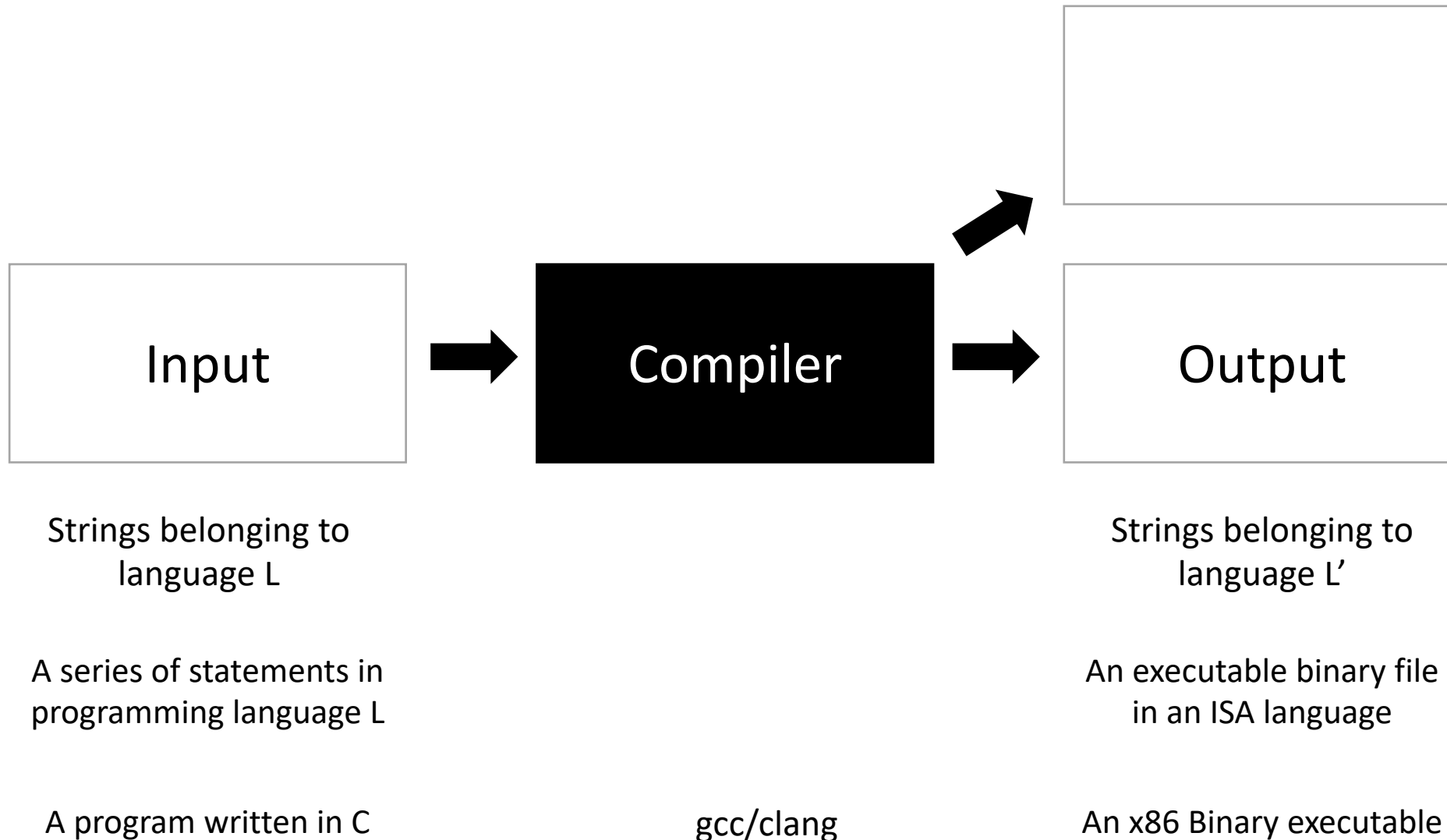
An executable binary file  
in an ISA language

An x86 Binary executable

gcc/clang

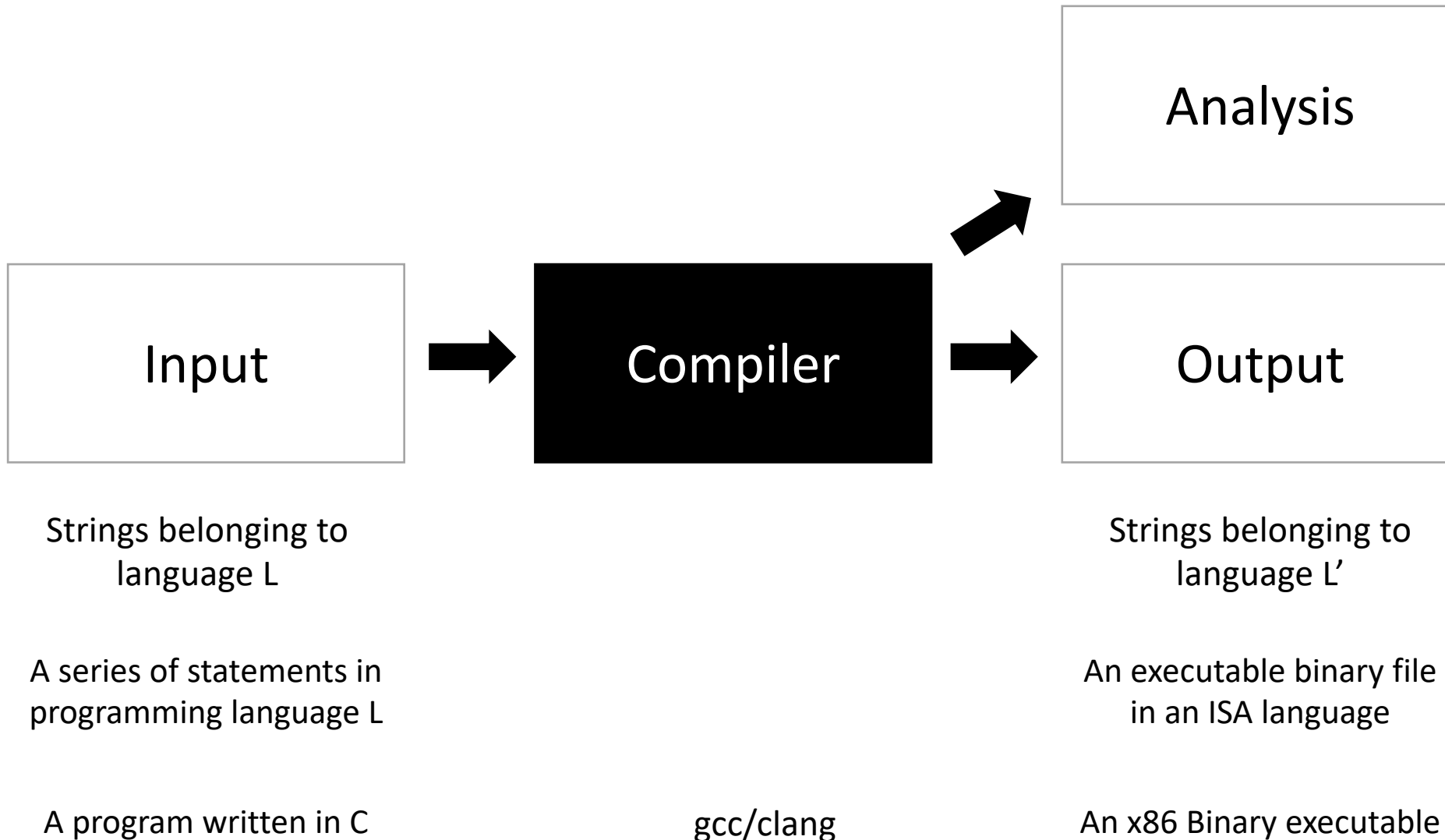
# *What is a compiler?*

What else does a compiler give you?

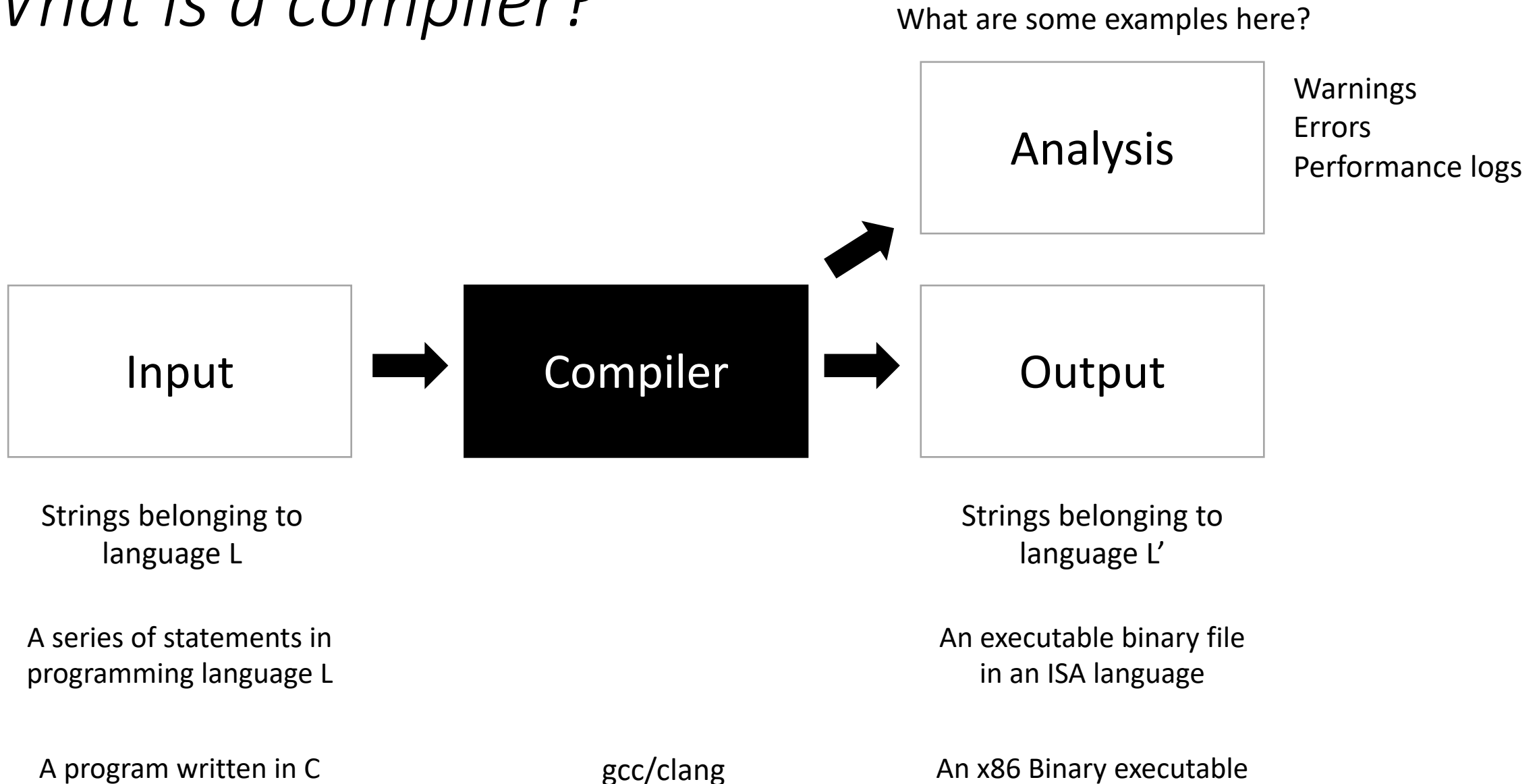


# *What is a compiler?*

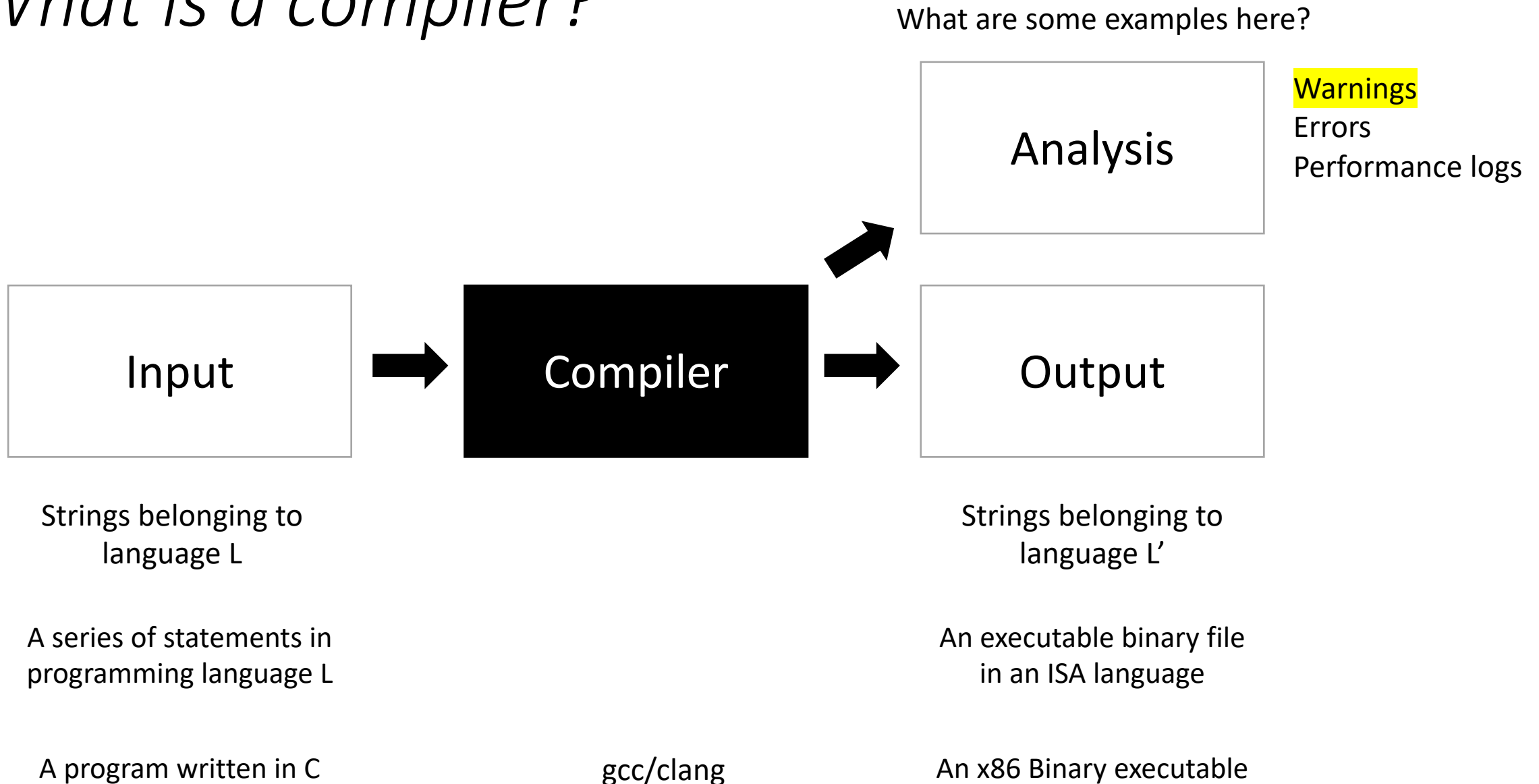
What are some examples here?



# *What is a compiler?*



# *What is a compiler?*



# Demo

- What are some examples of code that might give a warning?

# What can happen when the Input isn't valid?

```
int foo() {  
    int x;  
    int y = x;  
    return y;  
}
```

Try running this through the compiler

# What can happen when the Input isn't valid?

```
int foo() {  
    int x;  
    int y = x;  
    return y;  
}
```

```
int foo(int condition) {  
    int x;  
    if (condition) {  
        x = 5;  
    }  
    int y = x;  
    return y;  
}
```

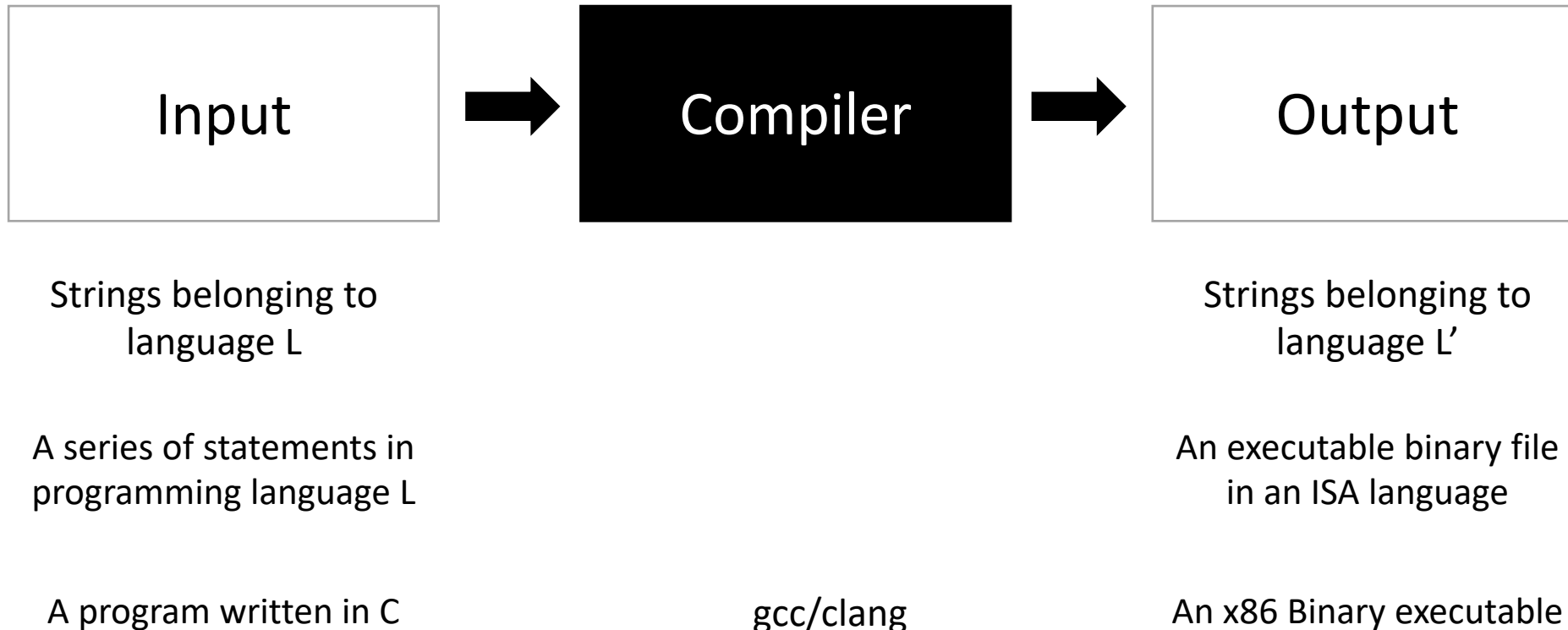
What about this one?

Try running this through the compiler



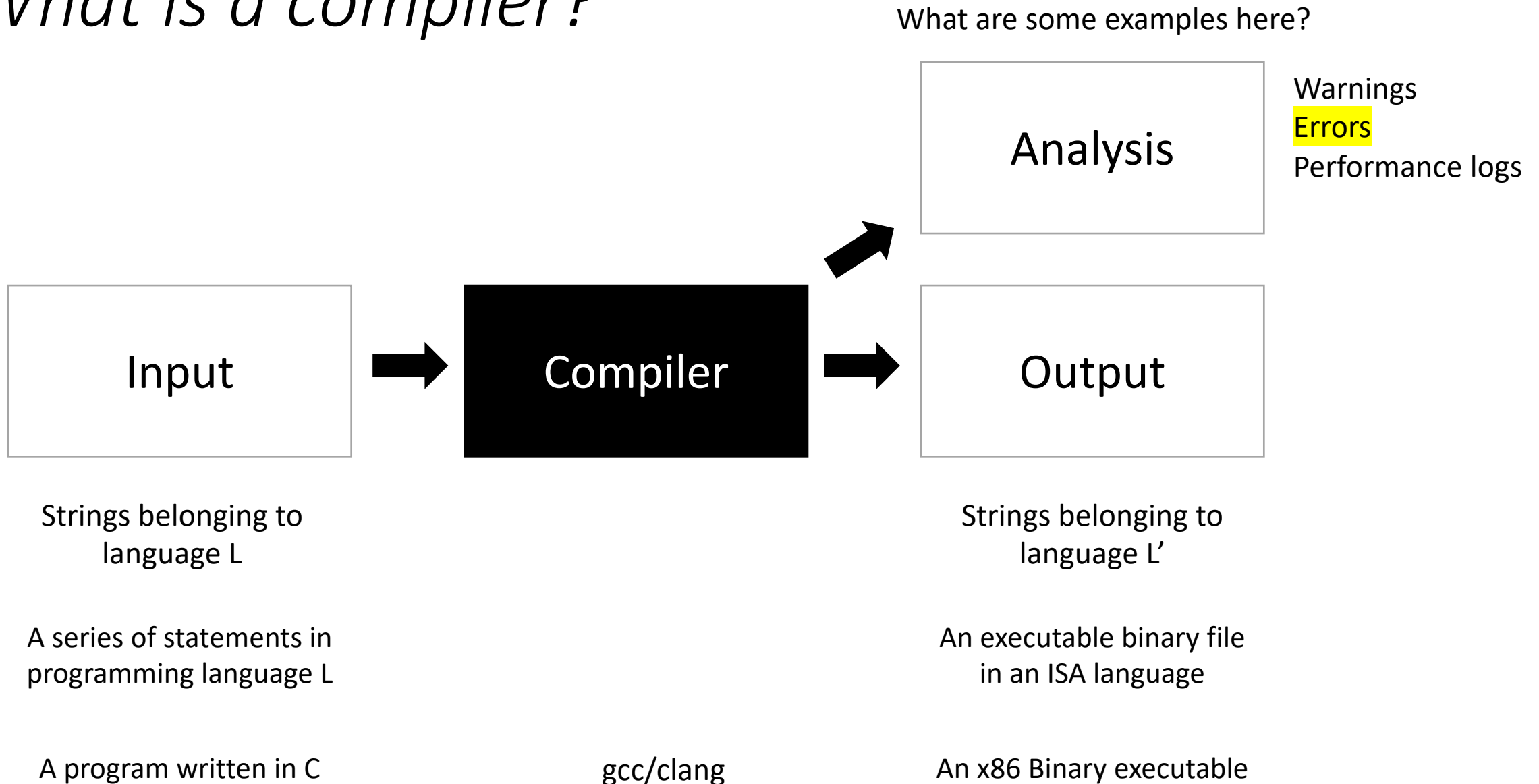
# What is a compiler?

A **valid** input must have a equivalent **valid** output.  
*Semantic equivalence*



# Uninitialized variable example

# *What is a compiler?*



# What can happen when the Input isn't valid?

```
int foo() {  
    int my_var = 5;  
    my_var = my_car + 5;  
    return my_var  
}
```

Try running this through a compiler

# What can happen when the Input isn't valid?

```
int foo() {  
    int my_var = 5;  
    my_var = my_car + 5;  
    return my_var  
}
```

Try running this through a compiler

You get an error and a suggestion these days

# What can happen when the Input isn't valid?

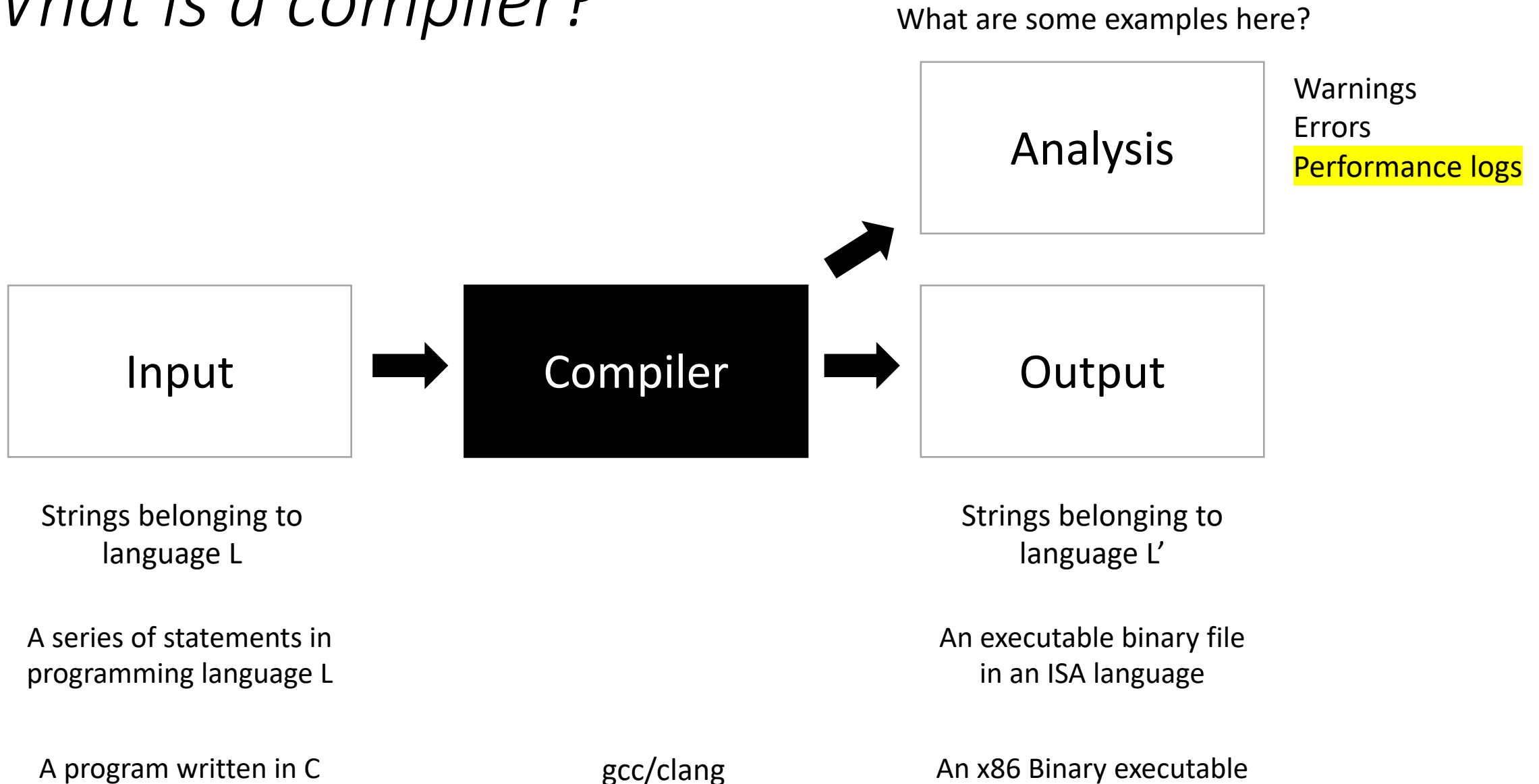
```
int foo() {  
    int *x = malloc(100*sizeof(int));  
    return x[100];  
}
```

What about this one? No error...

What sort of errors are compilers good at catching?

What ones are they not?

# *What is a compiler?*



# How can we know what the compiler is doing?

```
#define SIZE (1024*1024)
int add(int * a, int * b, int * c) {
    for (int i = 0; i < SIZE; i++) {
        a[i] = b[i] + c[i];
    }
    return 0;
}
```

Use the compiler flags

-Rpass-missed=loop-vectorize

-Rpass=loop-vectorize



# Does the compiler need to perform every step?

```
int foo() {  
    int my_var = 0;  
    for (int i = 0; i < 128; i++) {  
        my_var++;  
    }  
    return my_var;  
}
```

# Does the compiler need to perform every step?

```
int foo() {  
    int my_var = 0;  
    for (int i = 0; i < 128; i++) {  
        my_var++;  
    }  
    return my_var;  
}
```

Mentally we probably step through the for loop:

# Does the compiler need to perform every step?

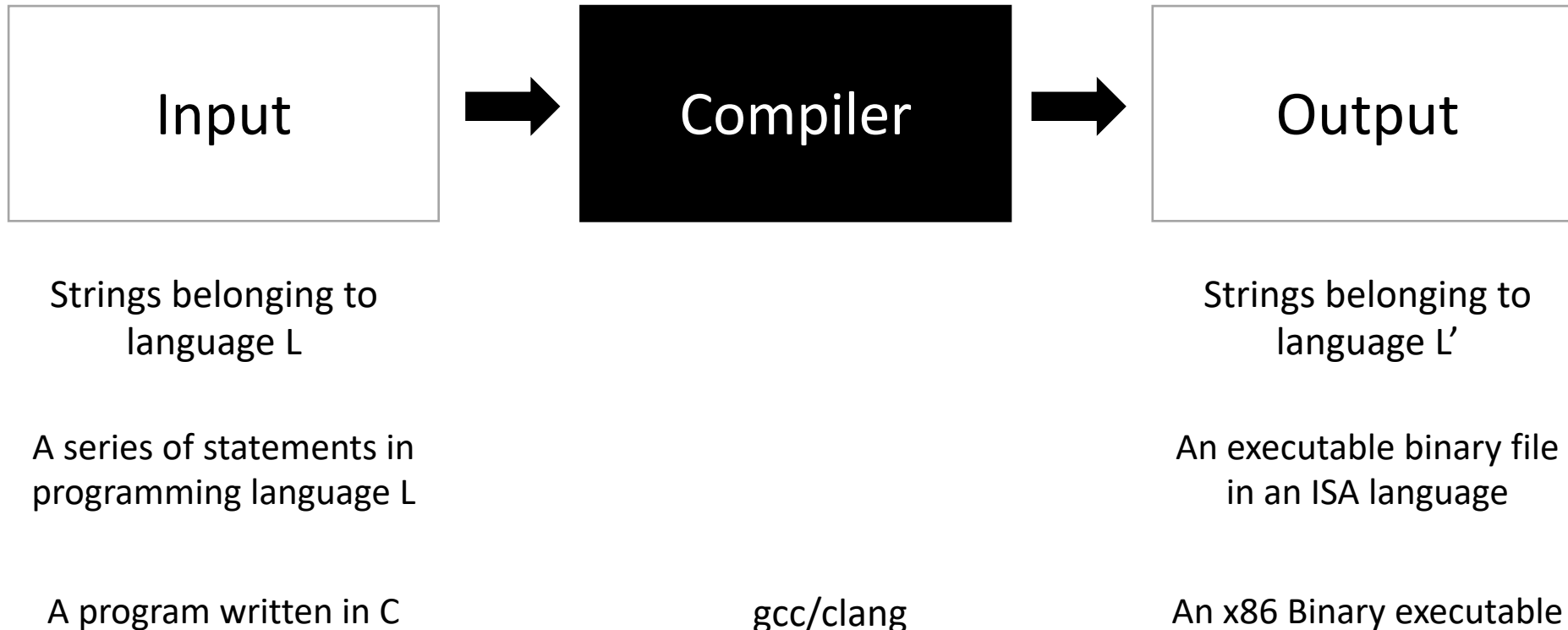
```
int foo() {  
    int my_var = 0;  
    for (int i = 0; i < 128; i++) {  
        my_var++;  
    }  
    return my_var;  
}
```

Mentally we probably step through the for loop:

What does the compiler do?

# What is a compiler?

A valid input must have a **equivalent** valid output.  
*Semantic equivalence*



# Does the compiler need to perform every step?

```
int foo() {  
    int my_var = 0;  
    for (int i = 0; i < 128; i++) {  
        my_var++;  
    }  
    return my_var;  
}
```

```
int foo() {  
    return 128;  
}
```

*are these the same?*

# Does the compiler need to perform every step?

```
int foo() {  
    int my_var = 0;  
    for (int i = 0; i < 128; i++) {  
        my_var++;  
    }  
    return my_var;  
}
```

```
int foo() {  
    return 128;  
}
```

*are these the same?*

**Functionally** - they are the same

**Non-functionally** - they are not

*Most compilers are concerned only with functional equivalence*

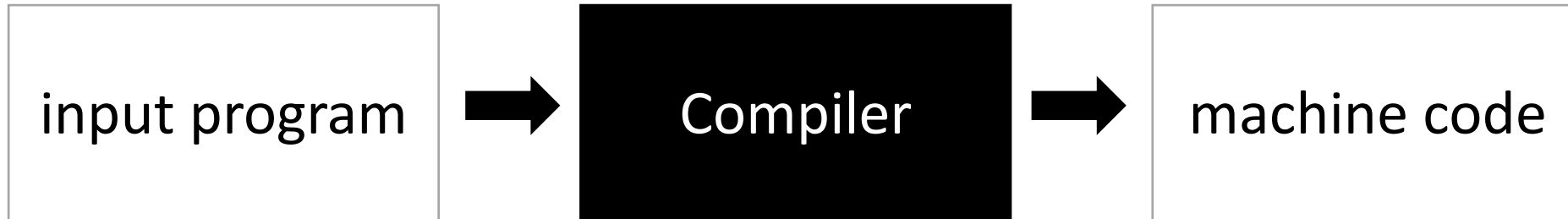
# Schedule

- Introduction to compilers
- **Compiler architecture**

# Compiler Architecture

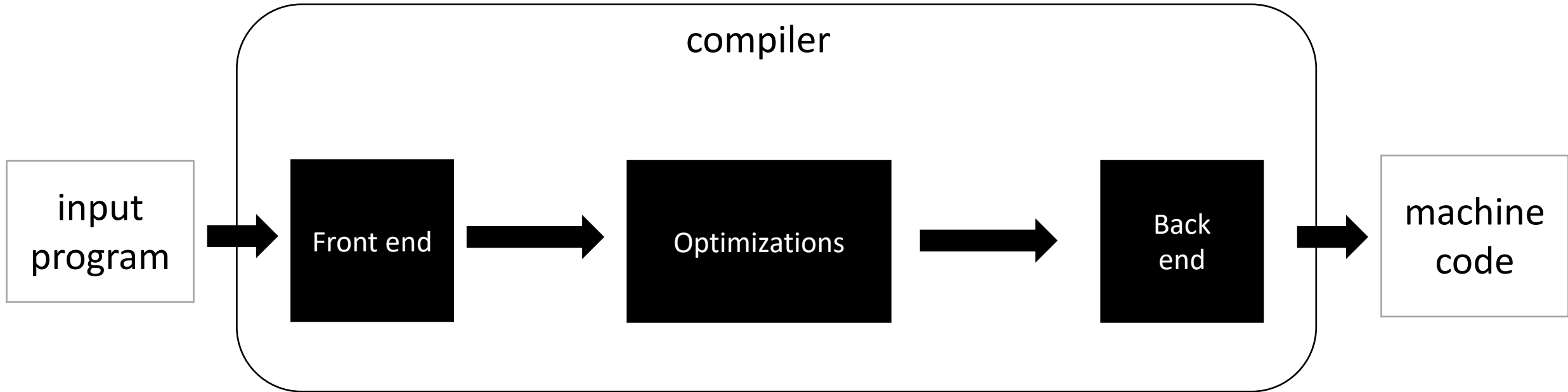


# Compiler Architecture



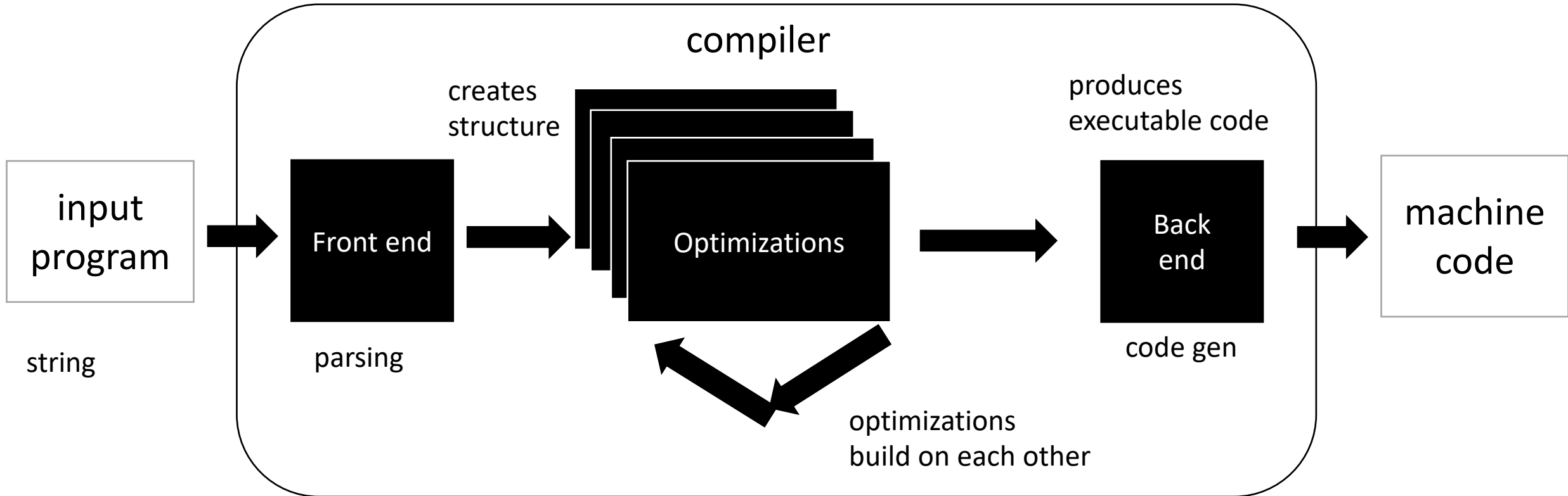
Compilers are complicated and this image is too simple

# Compiler Architecture



Medium detailed view

# Compiler Architecture



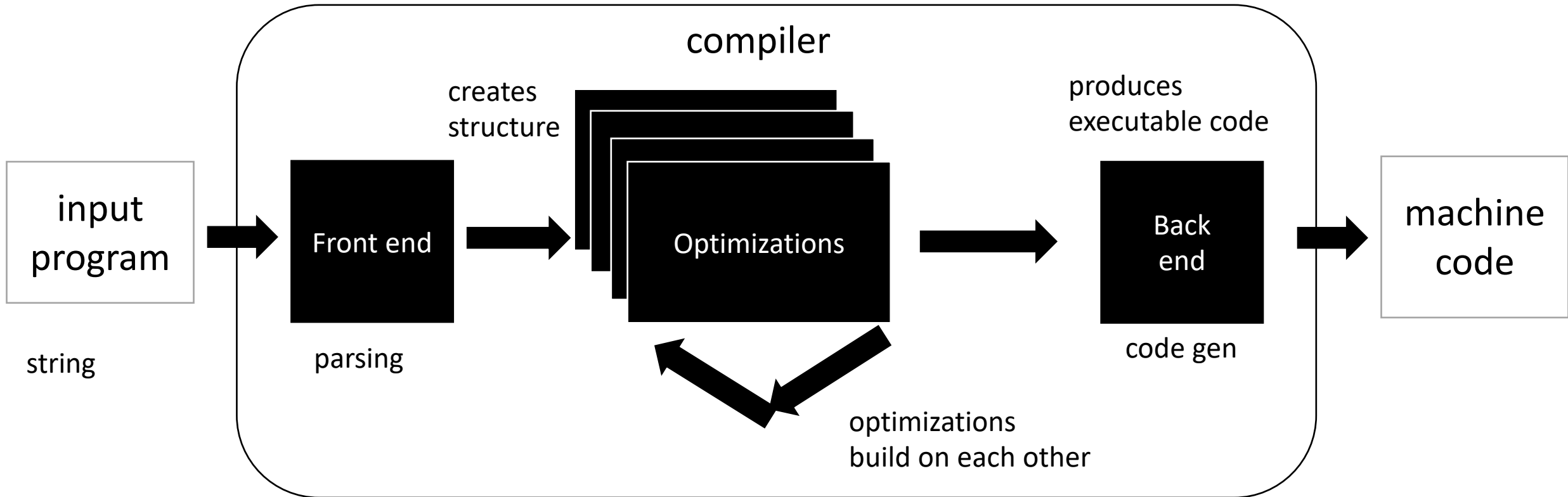
Medium detailed view

more about optimizations: <https://stackoverflow.com/questions/15548023/clang-optimization-levels>

# Compiler Architecture

*What are some of the benefits of this design?*

*What are some of the drawbacks of this design?*



Medium detailed view

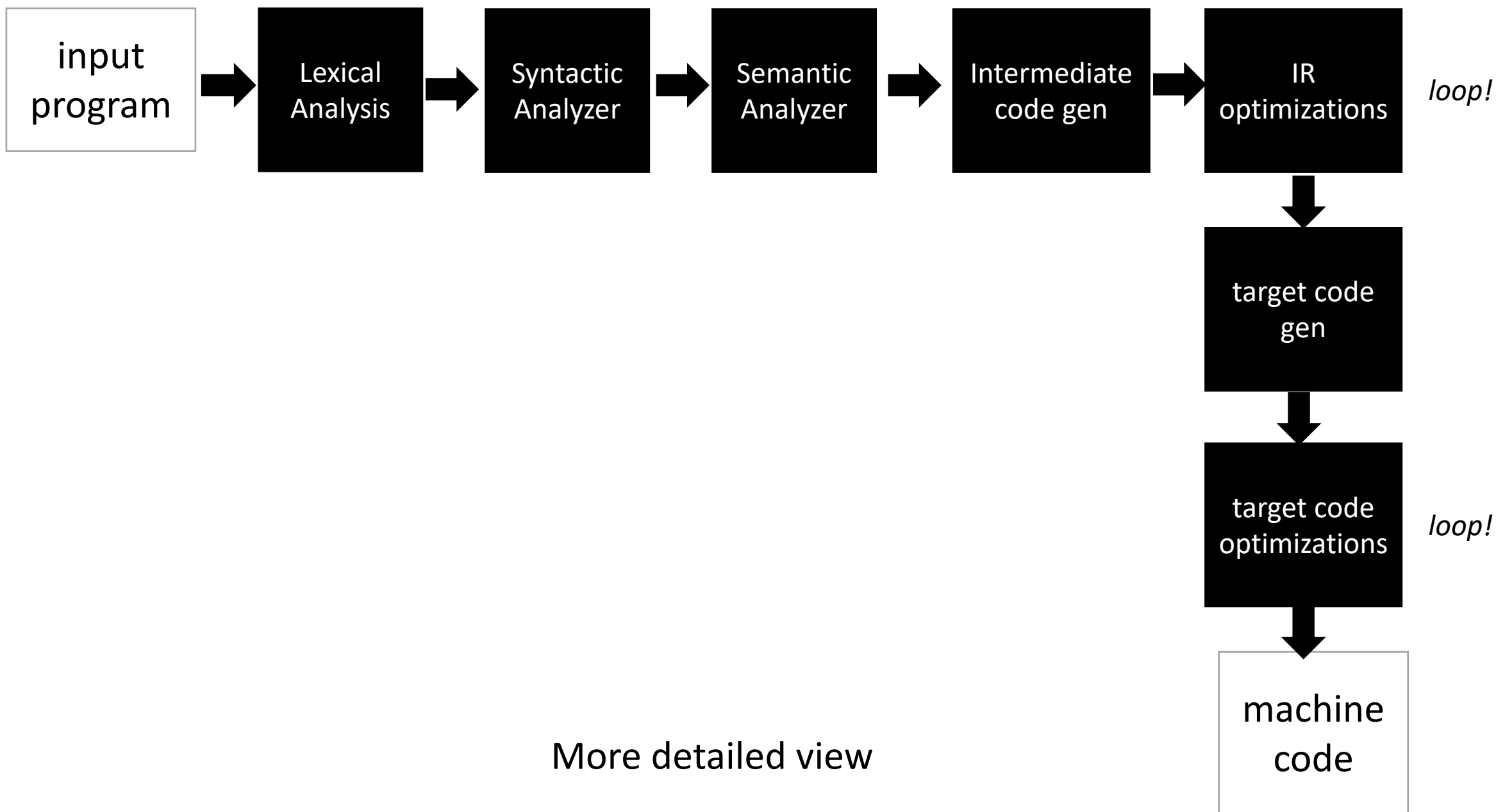
more about optimizations: <https://stackoverflow.com/questions/15548023/clang-optimization-levels>

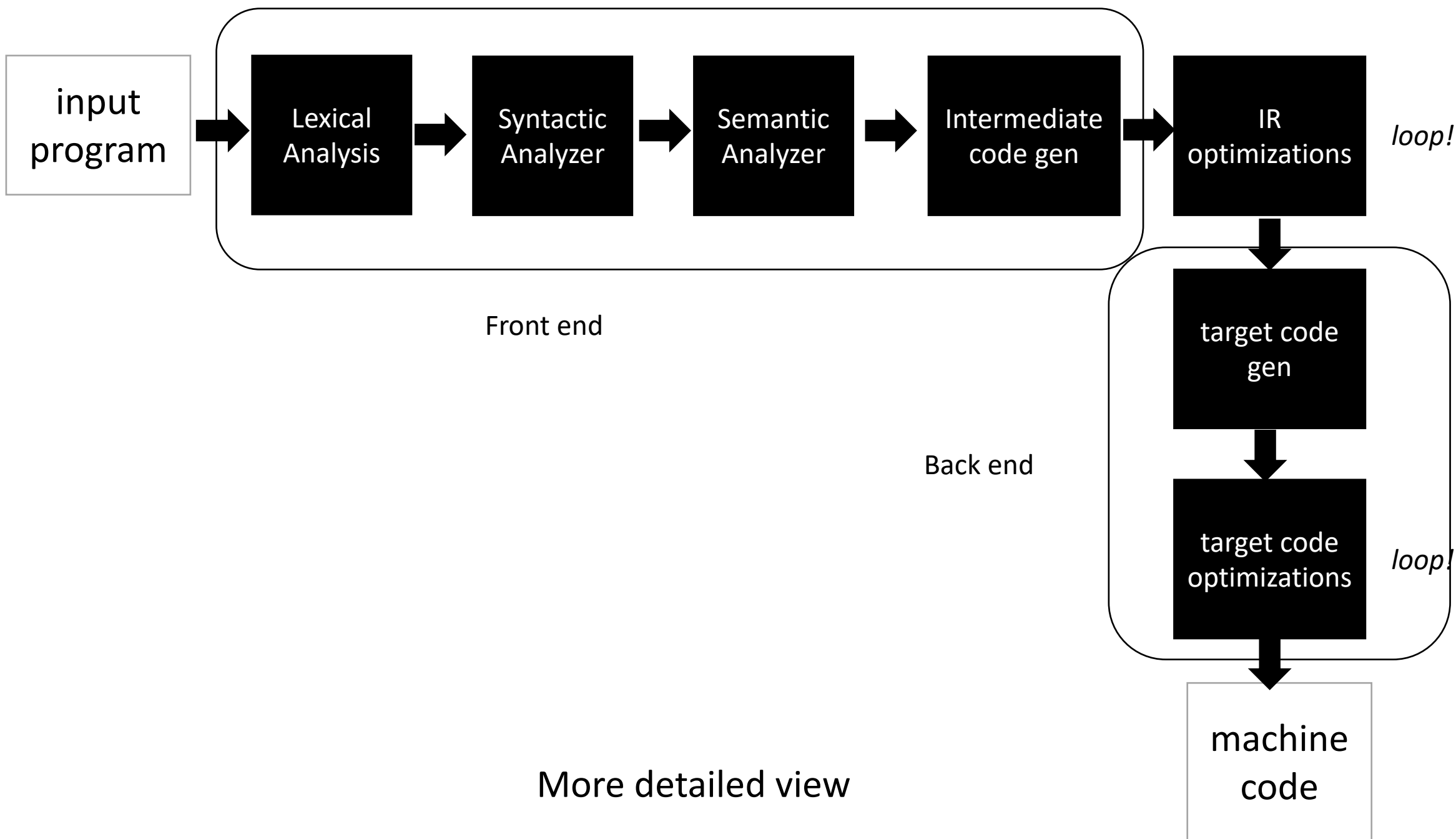
# LLVM compiler infrastructure example

- Front ends:
  - clang - c
  - clang++ - c++
  - Many others (rust, etc.)
- intermediate representation:
  - LLVM byte code
- backends
  - X86
  - ARM
  - M1
  - RISC-V

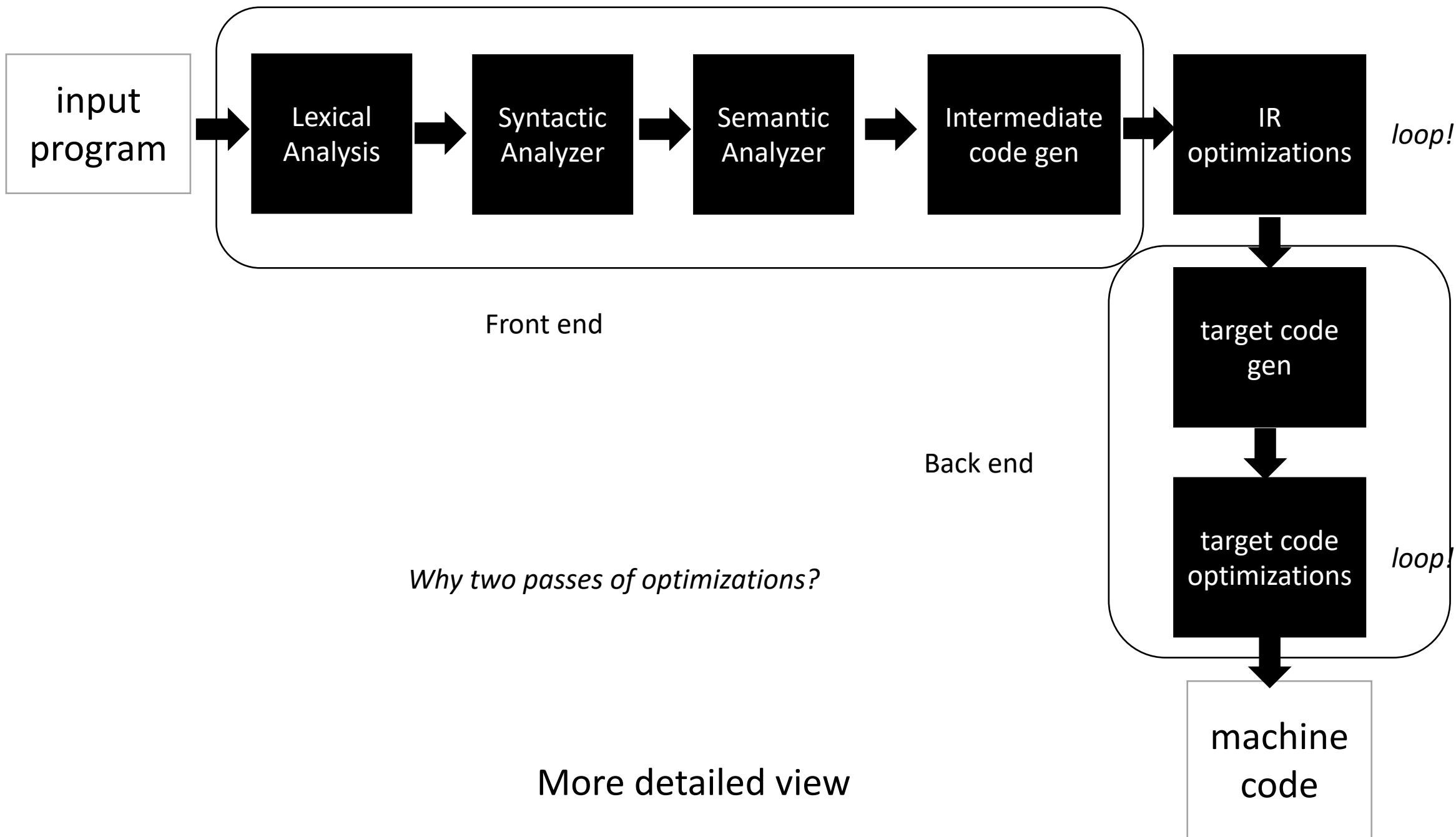
# More detailed compiler view

- Can't fit it nicely on one slide!





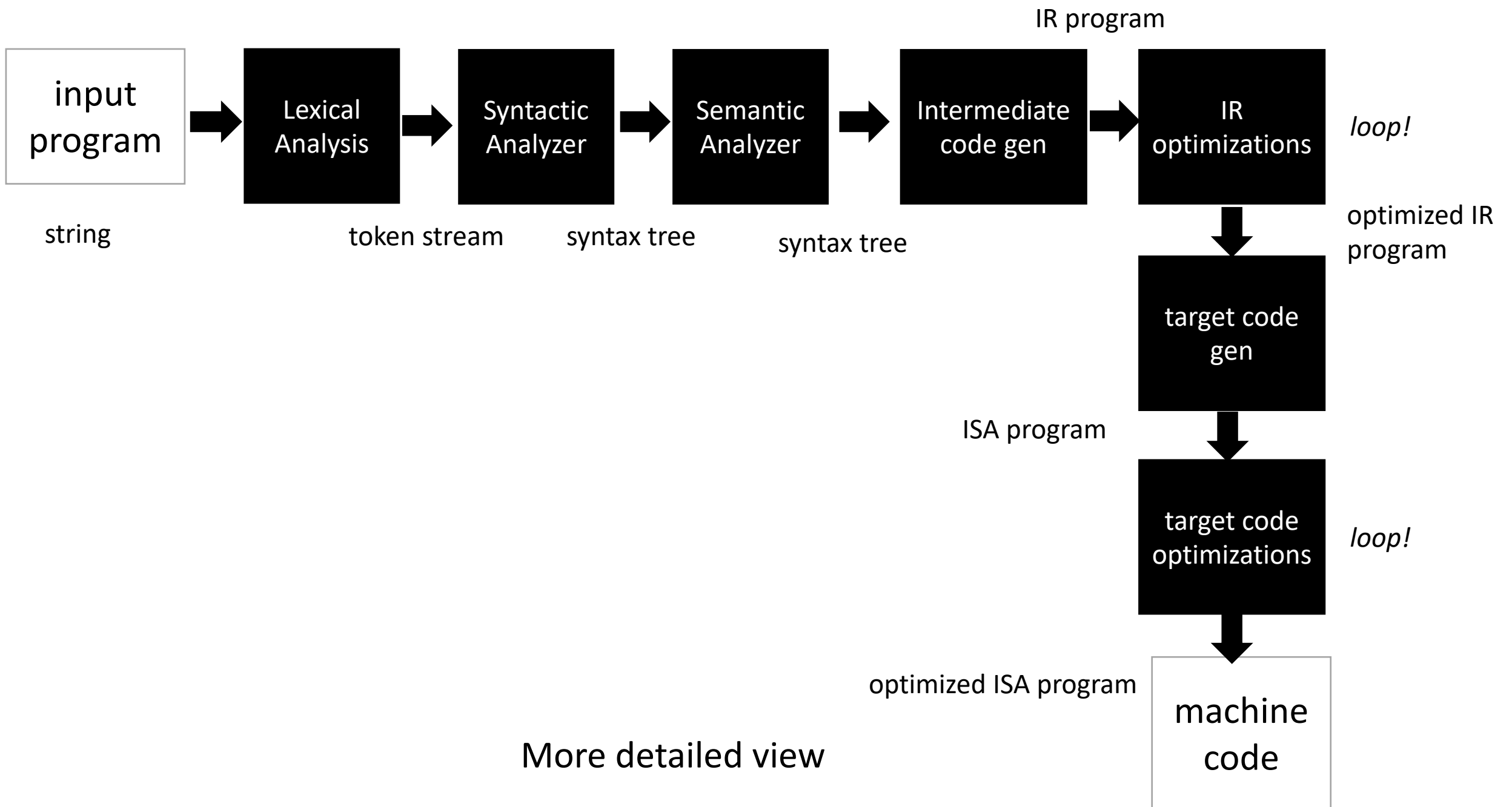


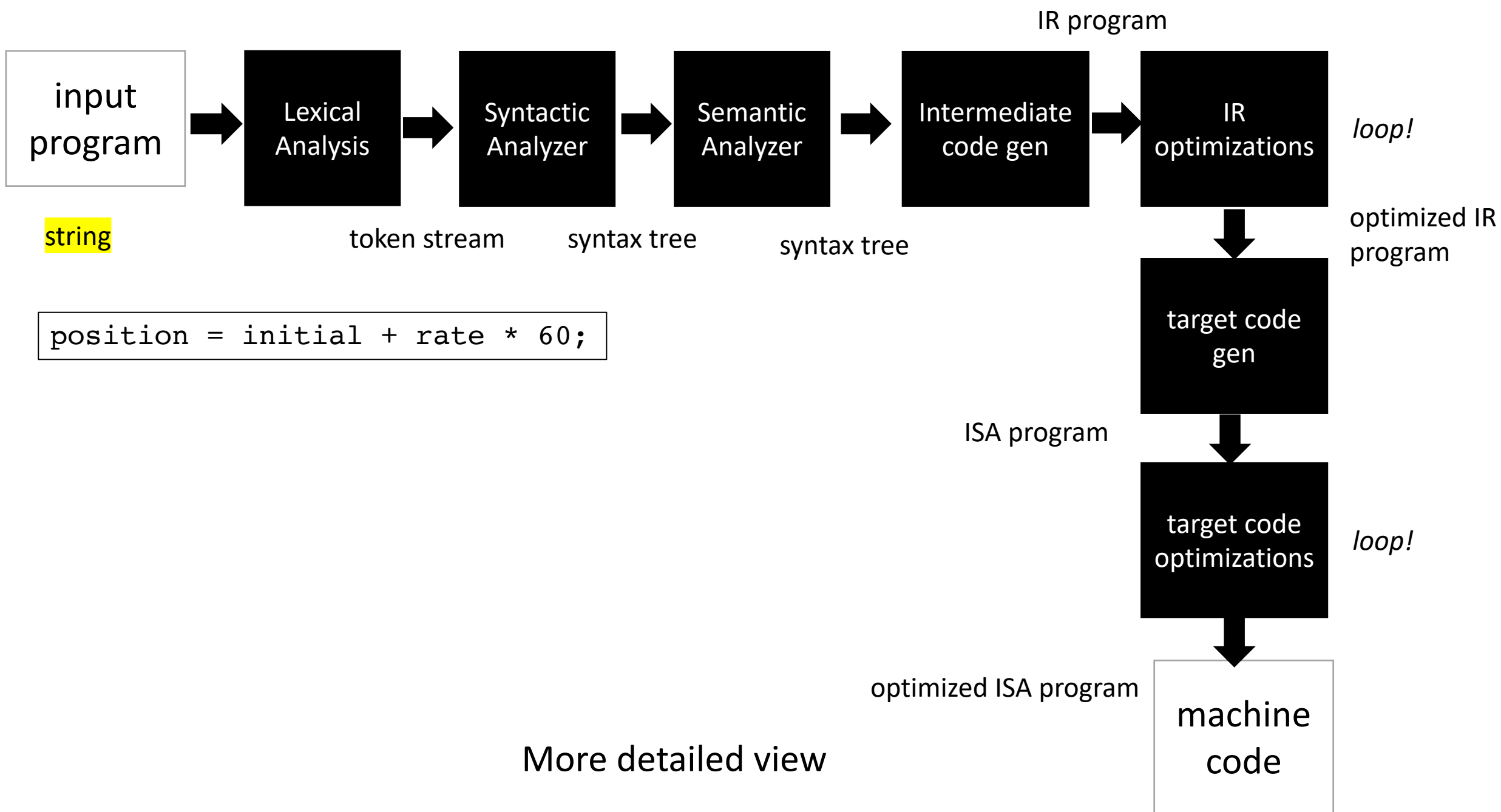


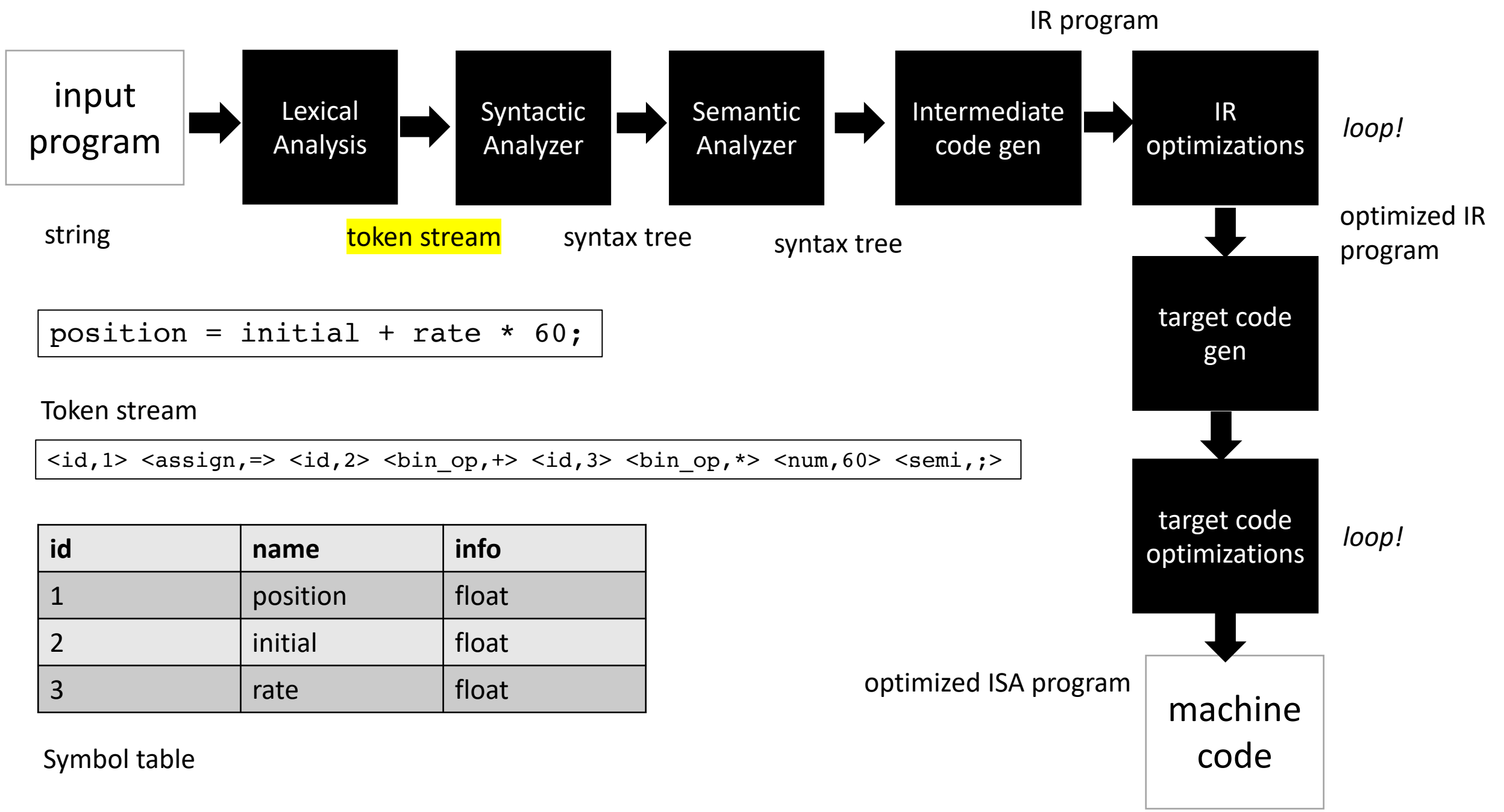
# architecture aware optimizations

- Example

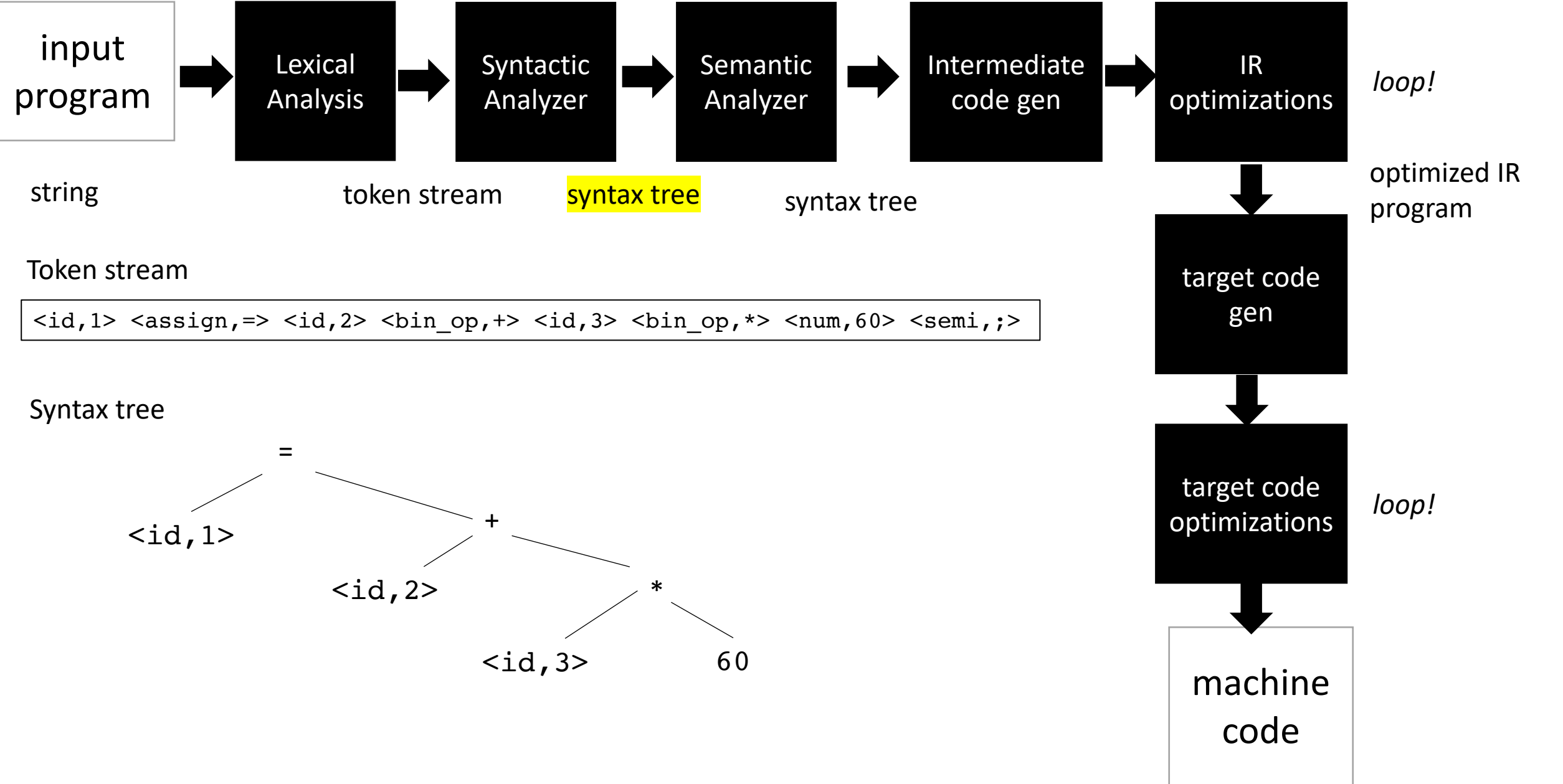
```
int foo(int *x, int *y, int *z) {  
    for (int i = 0; i < 128; i++) {  
        z[i] = y[i] + x[i];  
    }  
    return 0;  
}
```



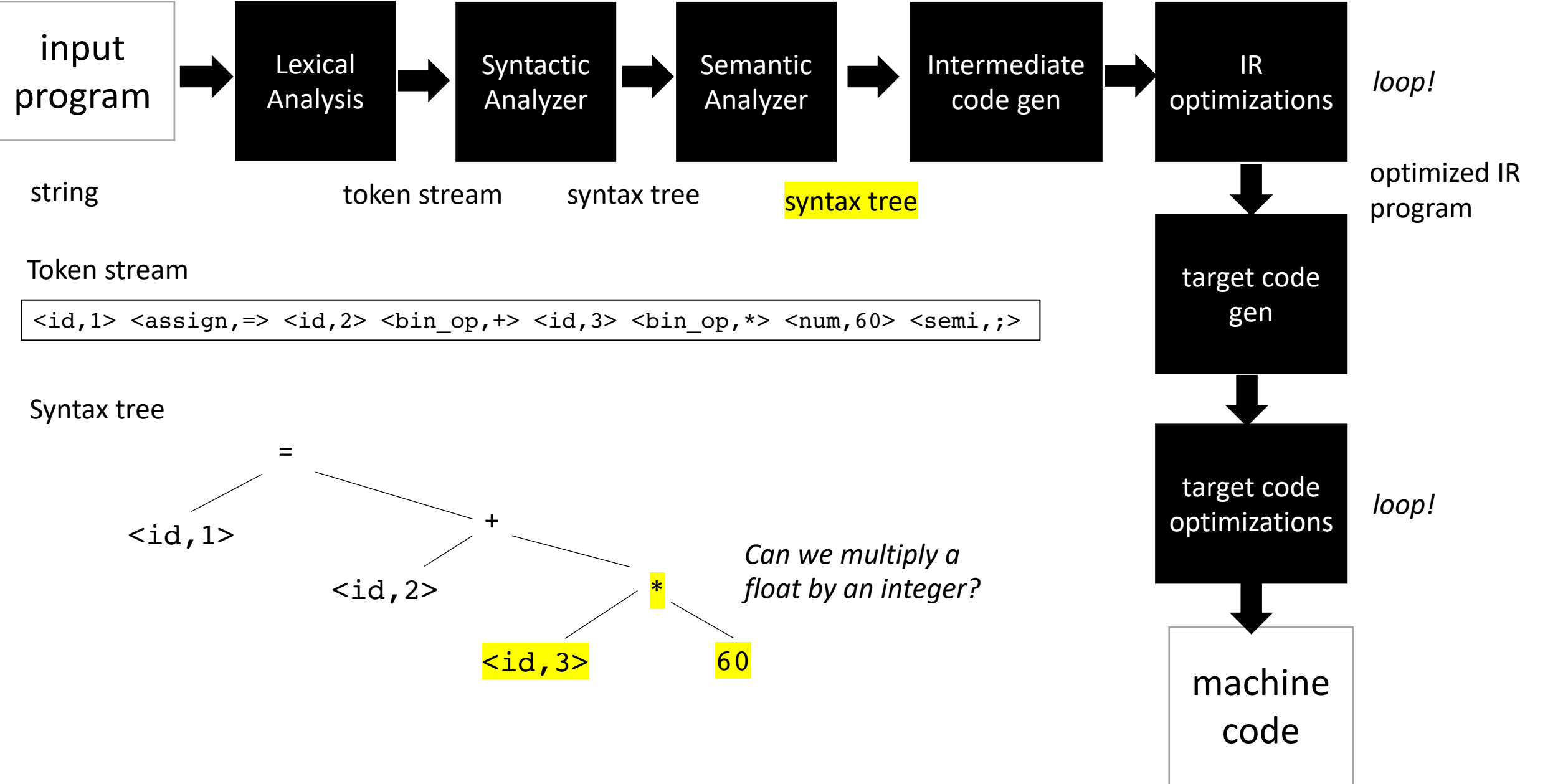




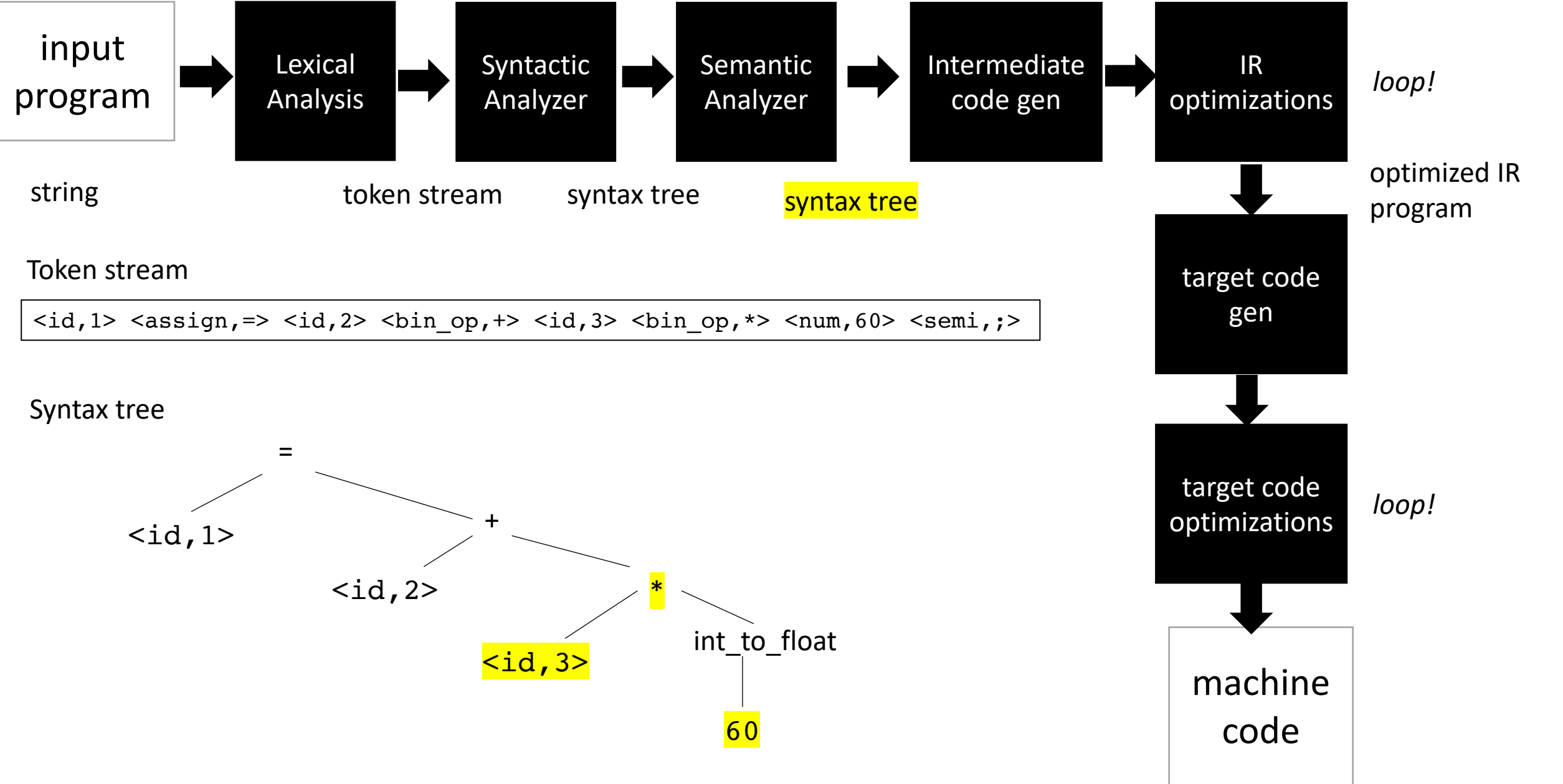
```
position = initial + rate * 60;
```



```
position = initial + rate * 60;
```

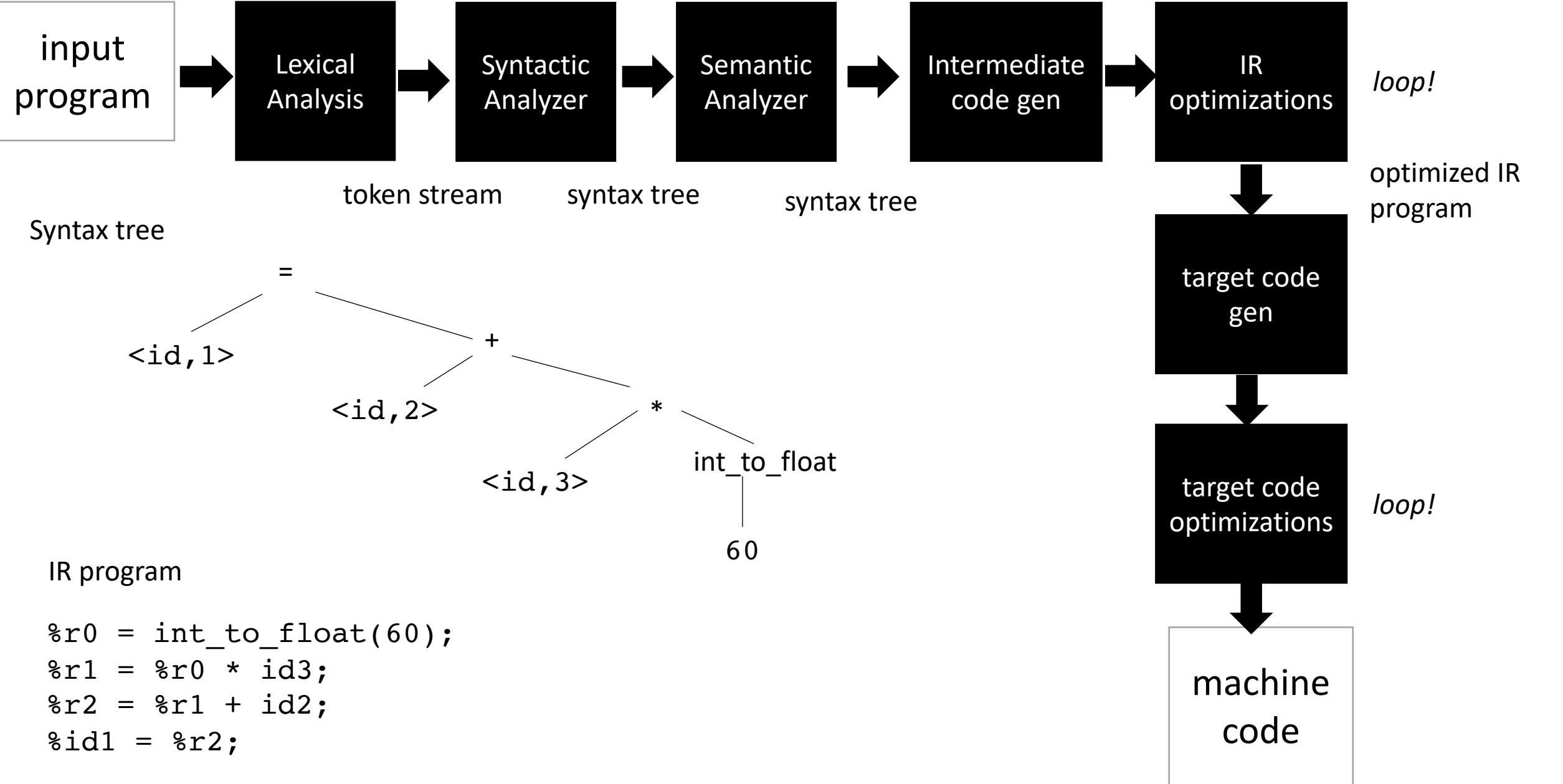


```
position = initial + rate * 60;
```

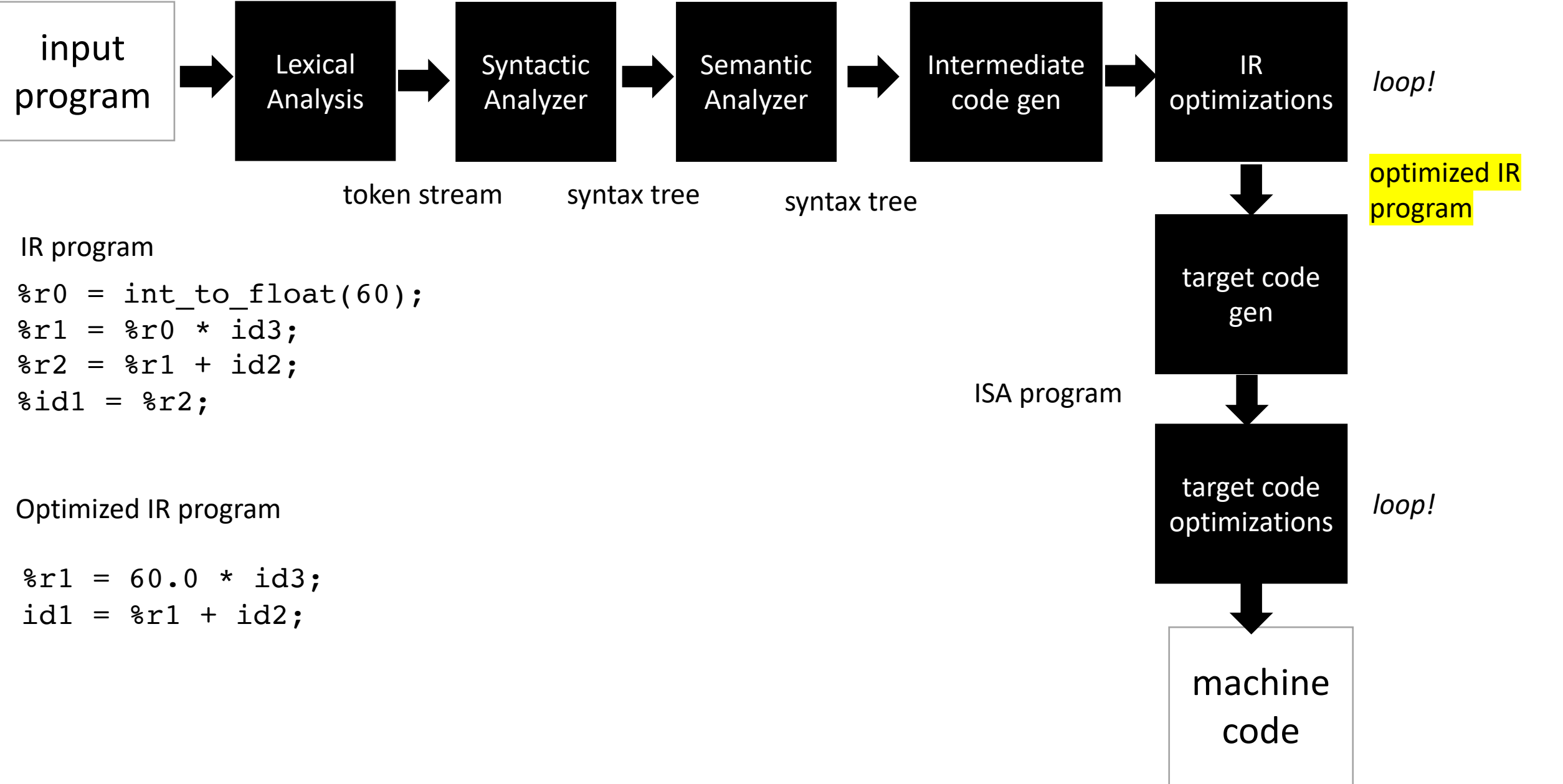




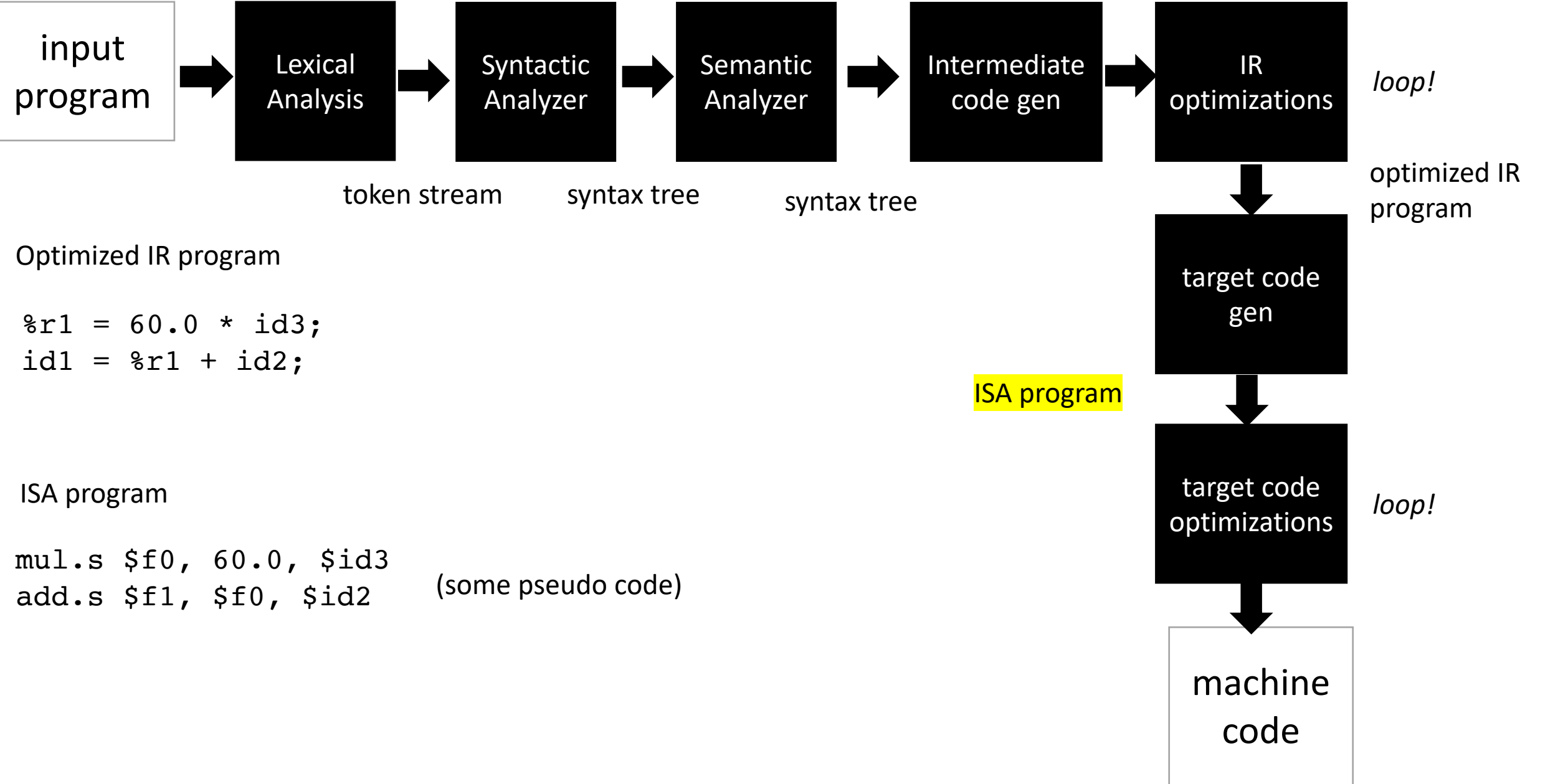
```
position = initial + rate * 60;
```



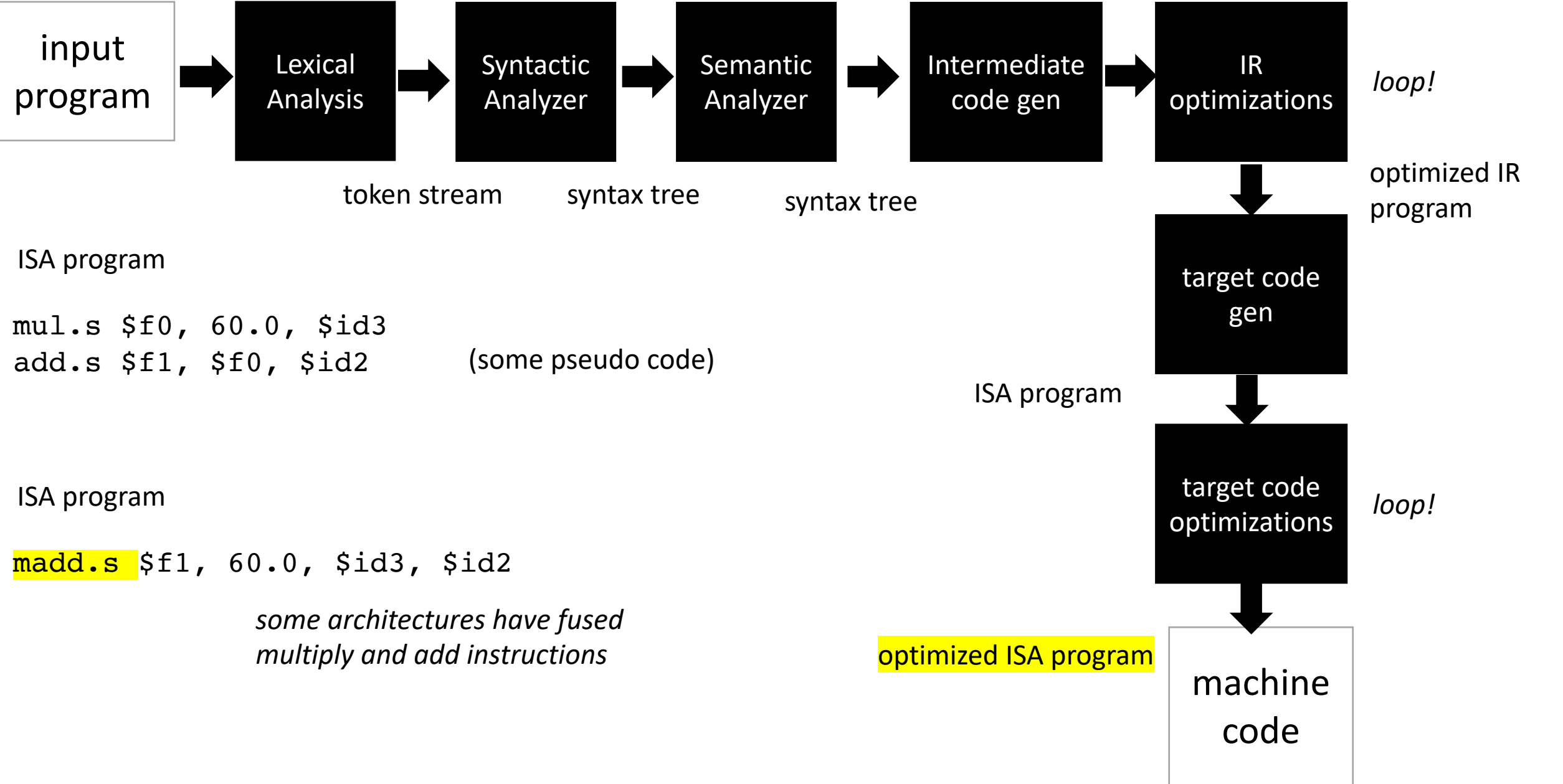
```
position = initial + rate * 60;
```



```
position = initial + rate * 60;
```



```
position = initial + rate * 60;
```

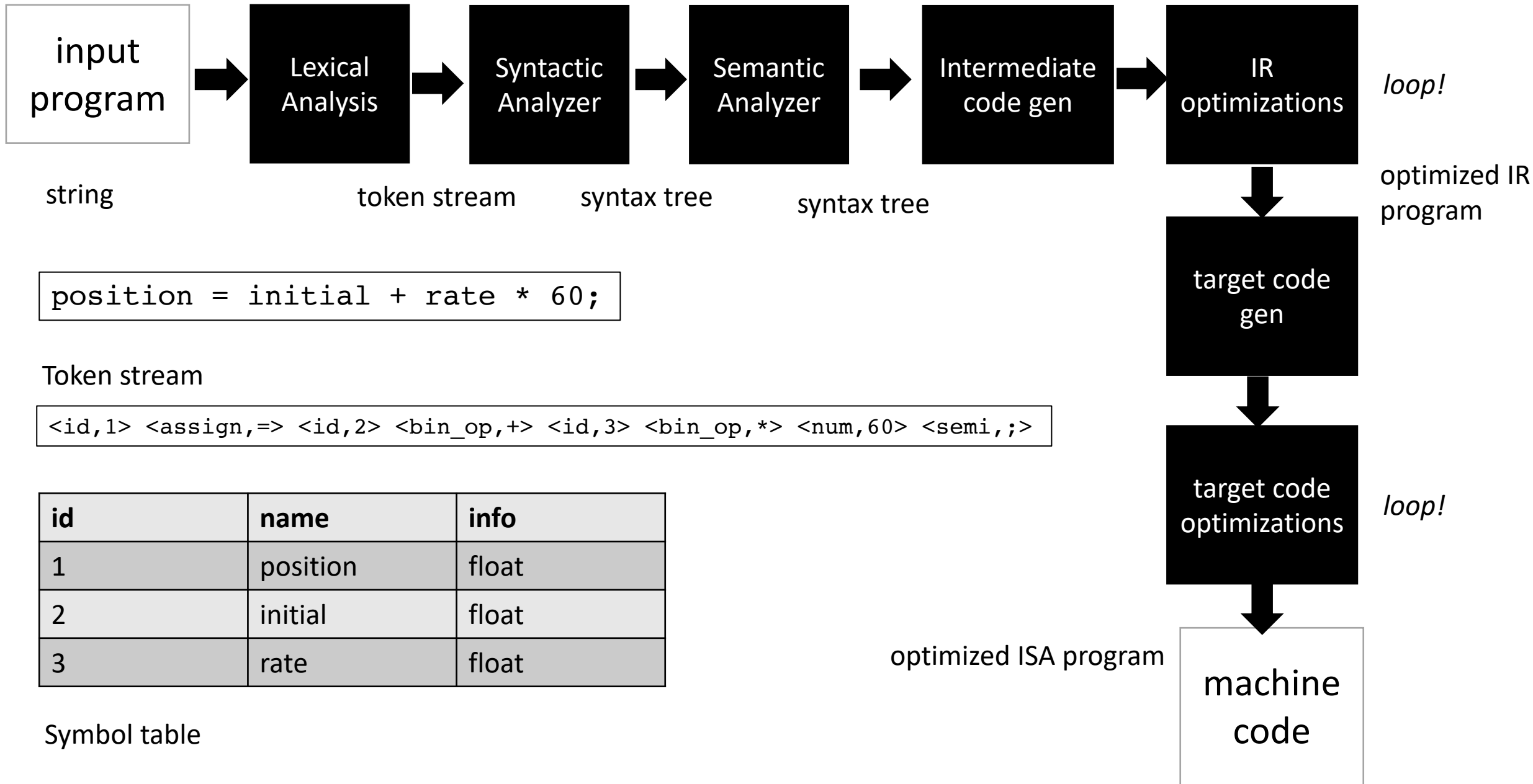


# Compiler Architecture



*Now you've seen a journey through a compiler!*

# First module



# Next Class

- **Lexical Analysis**