Code Generation

NI(E)-GEN, Spring 2021

https://courses.fit.cvut.cz/NI-GEN

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FIT

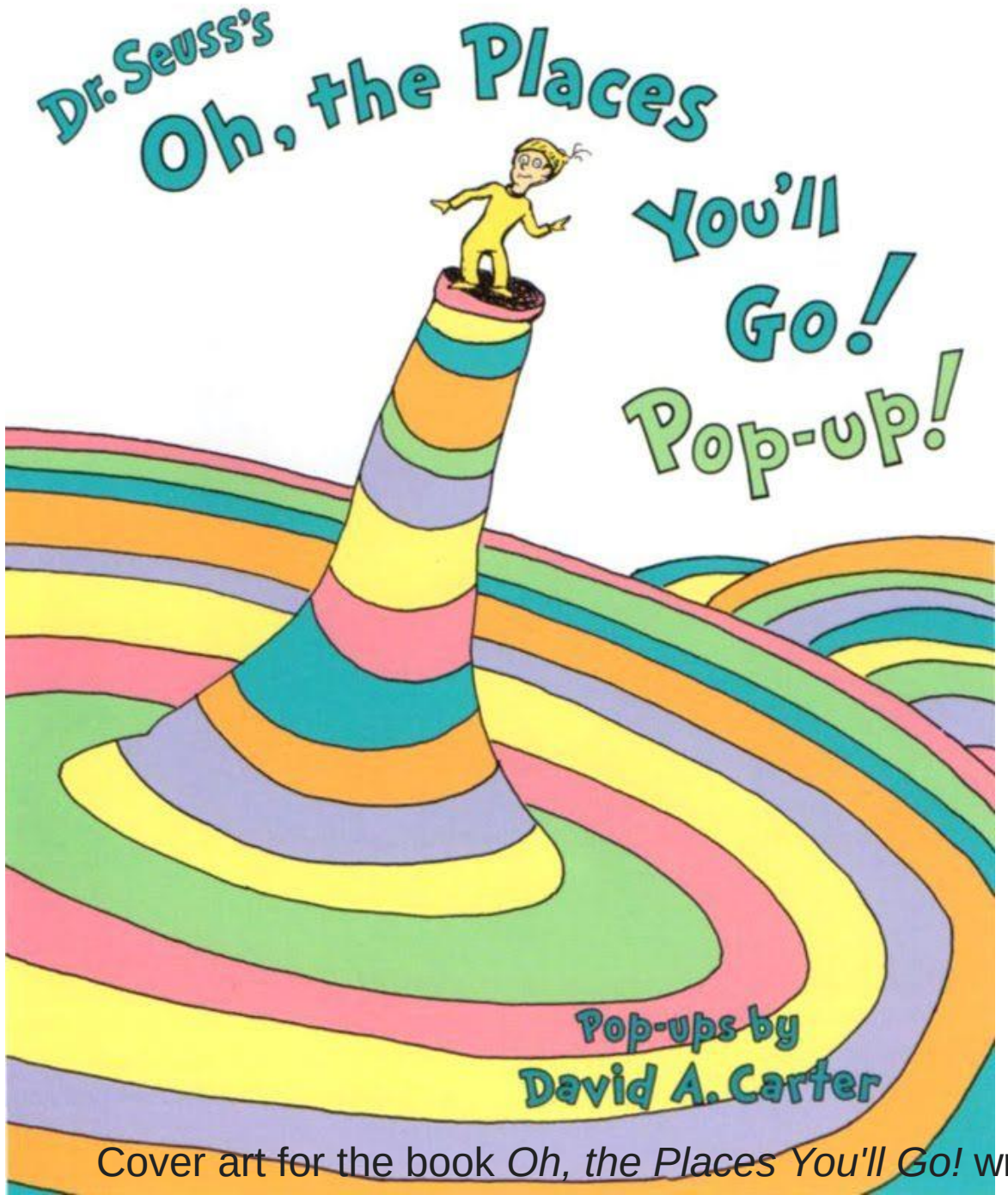
Trivia

- Petr Maj, majpetr@fit.cvut.cz

- Lectures, MON, on teams
- Tutorials, TUE, on teams

- Course information on teams and coursepages

- Grades in grades (and KOS)



Cover art for the book *Oh, the Places You'll Go!* written by Blais, Jacqueline; et al.

What to expect?

- some theory and science
- a lot of practice and engineering

What to expect?

- you will learn in reasonable detail what & how compiler does
- you will write your own compiler for a non-trivial (but still rather simple) programming language
- all the way down to machine code (simplified)
- you will have to work a lot on your own (but we are here to help)

What to expect?

- “irrelevant” details, since in compilers history repeats itself
- some overlap with programming languages design

Grades

- course project:
  - compiler for a small C-like language
  - reasonably large piece of work
  - due at the last tutorial, extensions possible upon previous request
  - 60 points max
- exam
  - on paper, covered theory and algorithms
  - 40 points max
- grade: > 90: A, 80..90 : B, 70..80: C, 60..70: D, 50..60: E, <50: F

Course Project

- compiler (middle & back end) implementation for a small c-like language
- target a tiny86 VM, which is a simplified model of a PC architecture based on x86
- code generation for higher level language constructs (condition, functions, etc.)
- optimizations (inlining, constant propagation, peepholer, etc.)
- register allocation

```
int main() {
    // allocate
    int numbers[100];
    // initialize
    for (int i =0; i < 100; ++i) {
        numbers[i] = i + 2;
    }
    // iterate
    for (int i = 0; i < 100; ++i) {
        if (numbers[i] == 0)
            continue;
        // we have a prime
        print(numbers[i]);
        // remove all that are divisible
        for (int j = i + 1; j < 100; ++j) {
            if (numbers[j] == 0)
                continue;
            if (numbers[j] % numbers[i] == 0)
                numbers[j] = 0;
        }
    }
}
```

And will you succeed?
Yes! You will, indeed!
(98 and 3/4 percent guaranteed.)

KID, YOU'LL MOVE MOUNTAINS!

So...
Be your name Buxbaum or Bixby or Bray
Or Mordecai Ali Van Allen O'Shea,
You're off to Great Places!
Today is your day!
Your mountain is waiting.
So...get on your way!

by Dr Seuss, from Oh The Places You'll Go


JAMES MONTGOMERY FLAGG

The Anatomy of a Compiler

NI(E)-GEN

https://courses.fit.cvut.cz/NI-GEN

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FIT

In the beginning…

- there were very few very big computers
- there were no programming languages
- there were no compilers

Prom. Nr. 2186

CALCULATRICES DIGITALES

DU DÉCHIFFRAGE DE FORMULES LOGICO-MATHÉMATIQUES PAR LA MACHINE MÊME DANS LA CONCEPTION DU PROGRAMME

THÈSE

PRÉSENTÉE

À L'ÉCOLE POLYTECHNIQUE FÉDÉRALE, ZURICH,

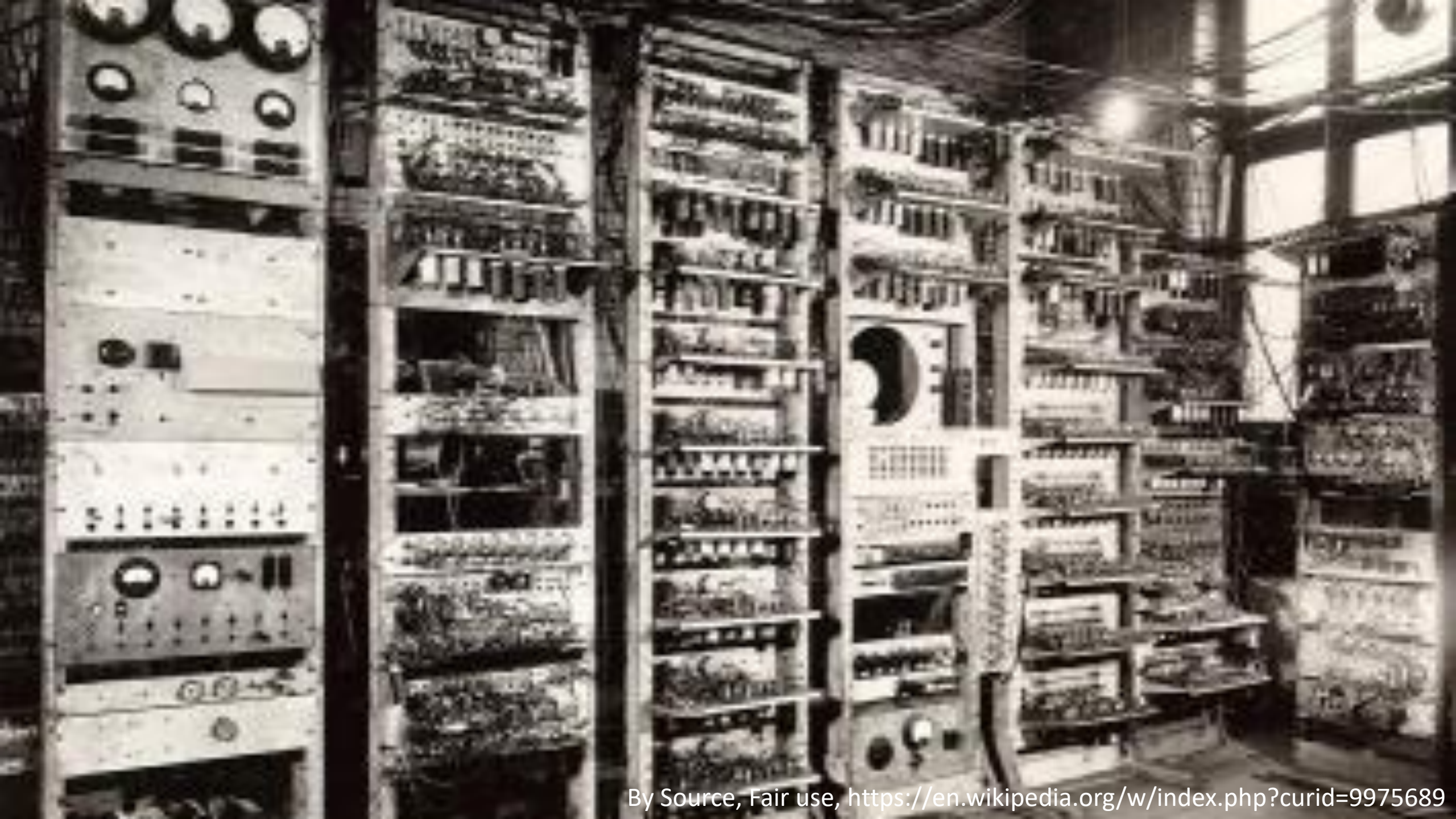
POUR L'OBTENTION DU

GRADE DE DOCTEUR ÈS SCIENCES MATHÉMATIQUES

PAR

CORRADO BÖHM, ing. électr. dipl. EPUL
de Milan (Italie)

Rapporteur : Prof. Dr. E. STIEFEL
Co-rapporteur : Prof. Dr. P. BERNAYS



By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=9975689


My computer is
BIGGER, BETTER and
FASTER than yours!

IBM, We Have a Problem

Bigger Computers Run Bigger Software

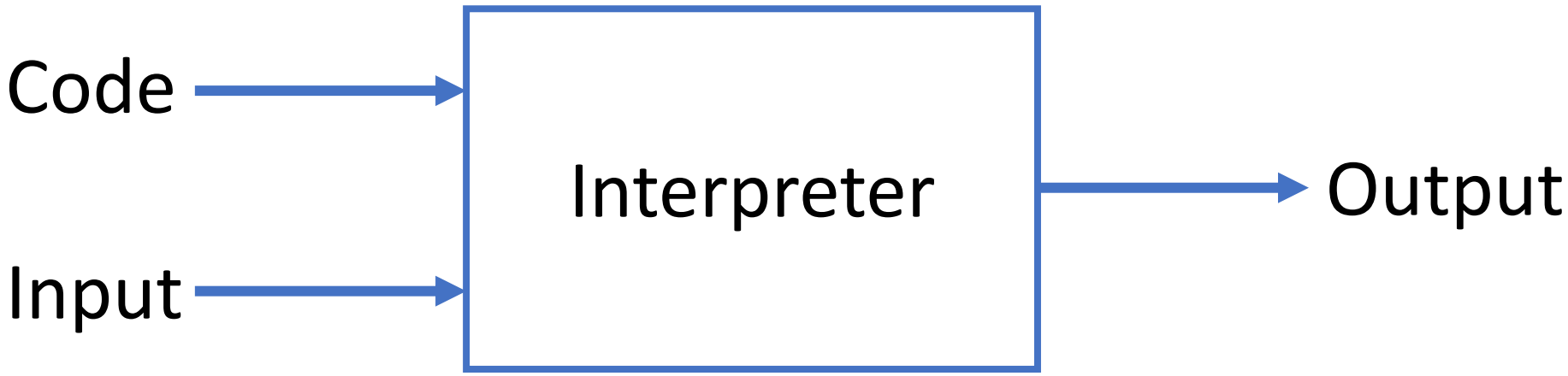
- early computers were super expensive
- but soon developing the software they executed was even costlier
- all of it written in assembly
- enter speedcoding

Actual “high-level” programming language

Speedcoding

- when an arithmetic operation was found in the source code, corresponding routine was called

- designed to ease the burden on programmers
- not for speed (up to 20x slower than assembly)
- occupied about 300 bytes in RAM


Code
Input
Interpreter
Output

High Level Languages Are Great!

but...

Efficiency Matters

- the cost of an interpreter is prohibitive

- John Backus observed that in speedcoding lot of time is spent in the mathematical formulas being translated over and over again every time they are executed

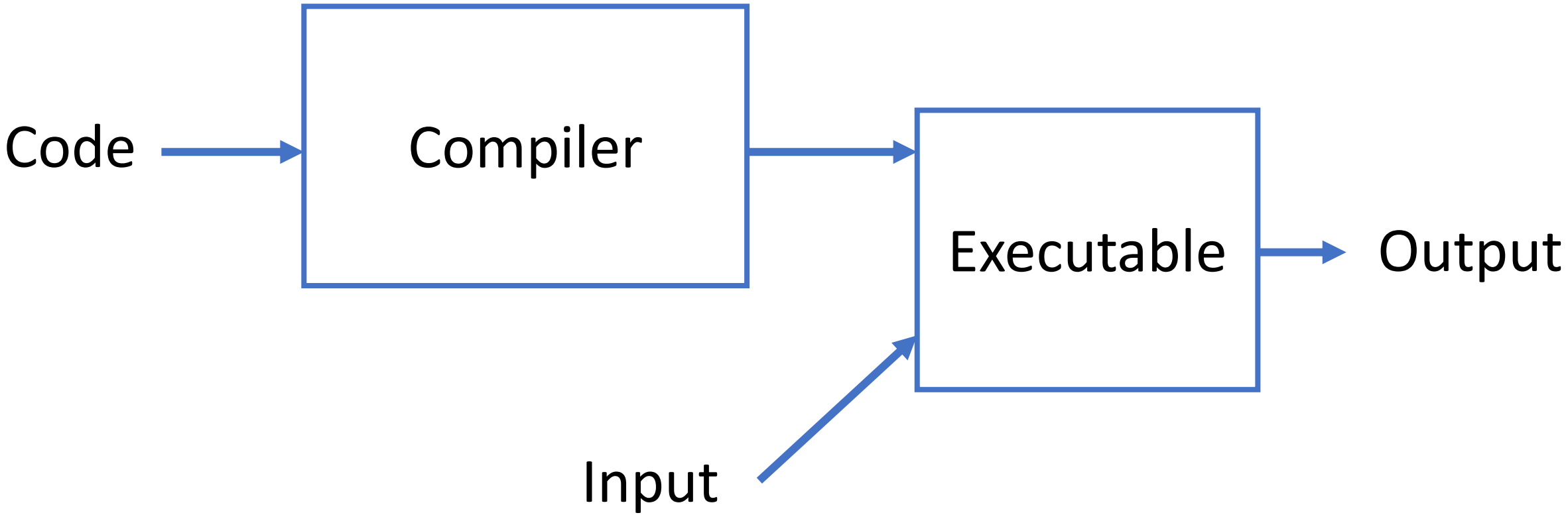
Efficiency Matters

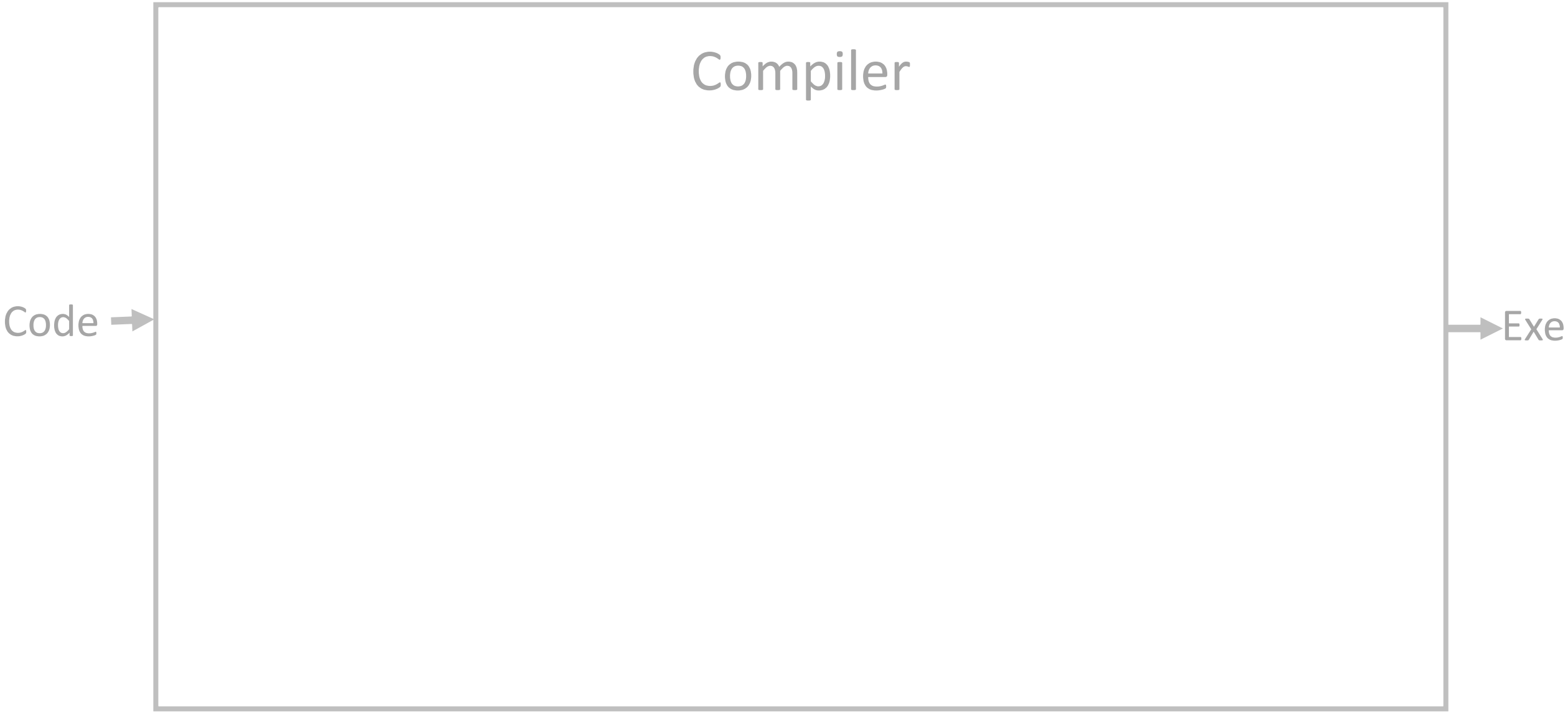
- the cost of an interpreter is prohibitive

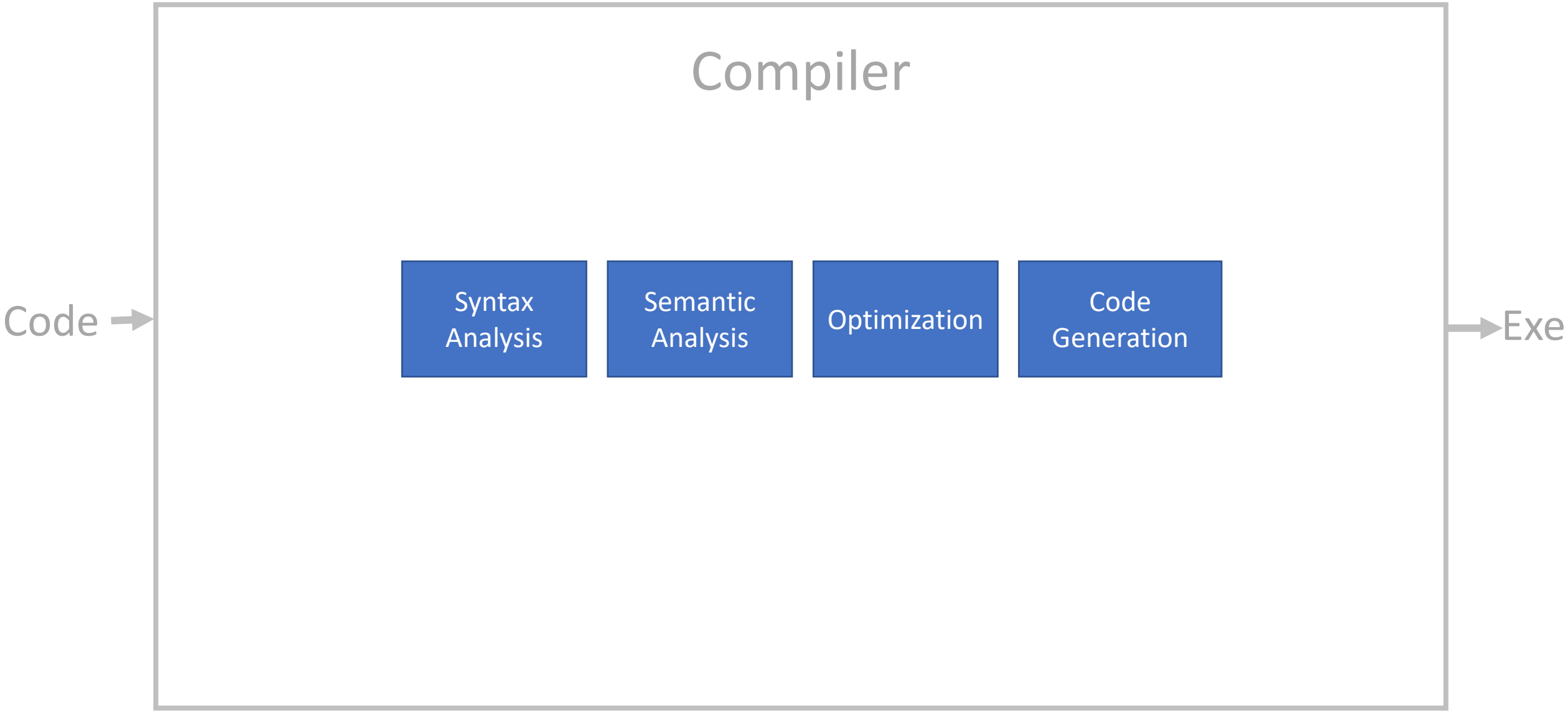
- John Backus observed that in speedcoding lot of time is spent in the mathematical FORmulas being TRANslated over and over again every time they are executed

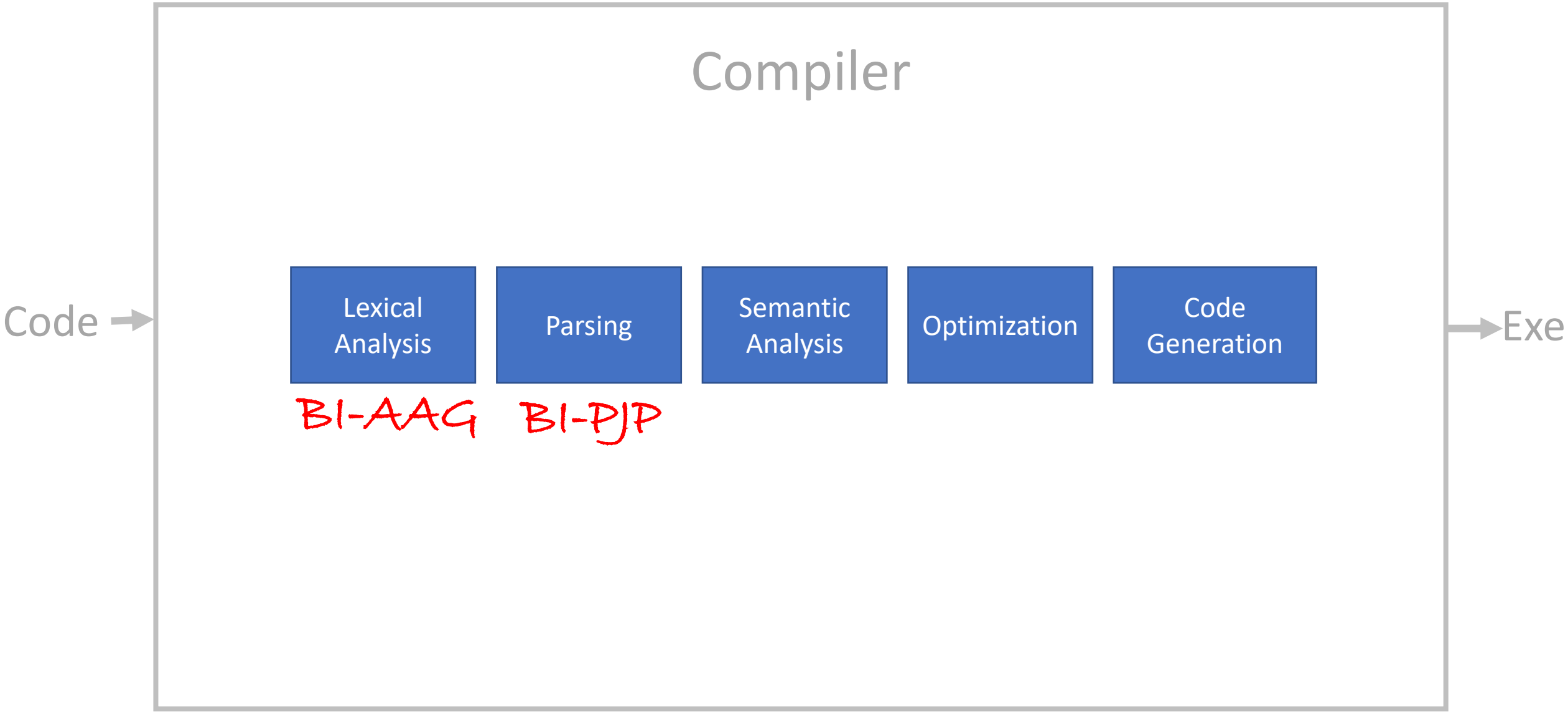
FORTRAN

- what if the operations are translated first once and for all?
- doing this ahead of time also means the compiler can:
  - is not resource constrained
  - can do lots of things (can be slow, as long as generated executable is fast)
- the very first compiler and highly influential compiler


Code
Compiler
Executable
Output
Input


Compiler
Code
Exe


Compiler
Code
Syntax Analysis
Semantic Analysis
Optimization
Code Generation
Exe


Compiler
Code
Lexical Analysis
Parsing
Semantic Analysis
Optimization
Code Generation
Exe
BI-AAG
BI-PJP

```
// a simple function
int min(int a, int b) {
    if (a < b)
        return a;
    else
        return b;
}
```

```
// a simple function
int min(int a, int b) {
    if (a < b)
        return a;
    else
        return b;
}
```

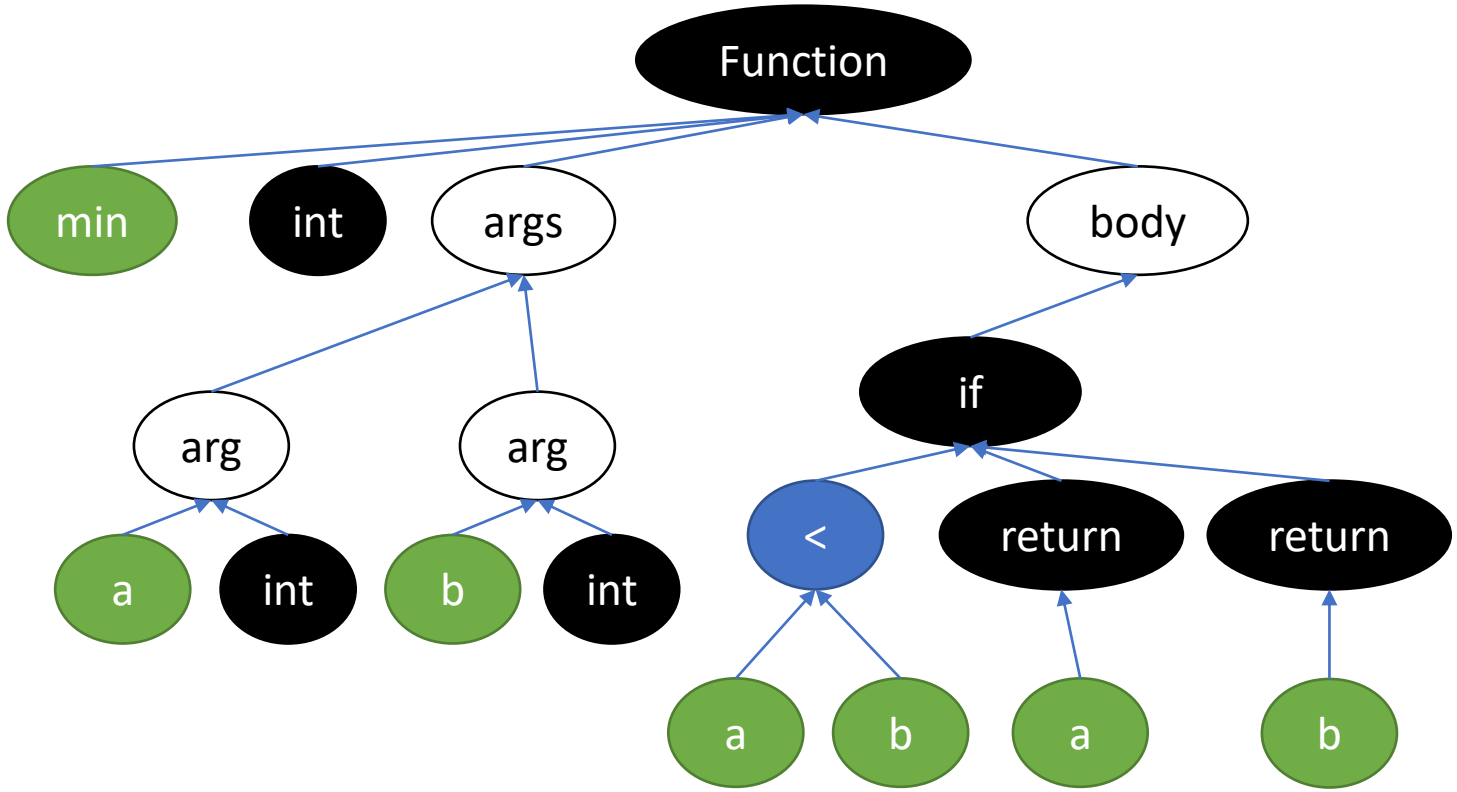
```
// a simple function
int min(int a, int b) {
    if (a < b)
        return ☺;  Invalid character
    else
        return b;
}
```

```
// a simple function
int min(int a, int b) {
    if (a < b)
        return if; valid...
    else
        return b;
}
```

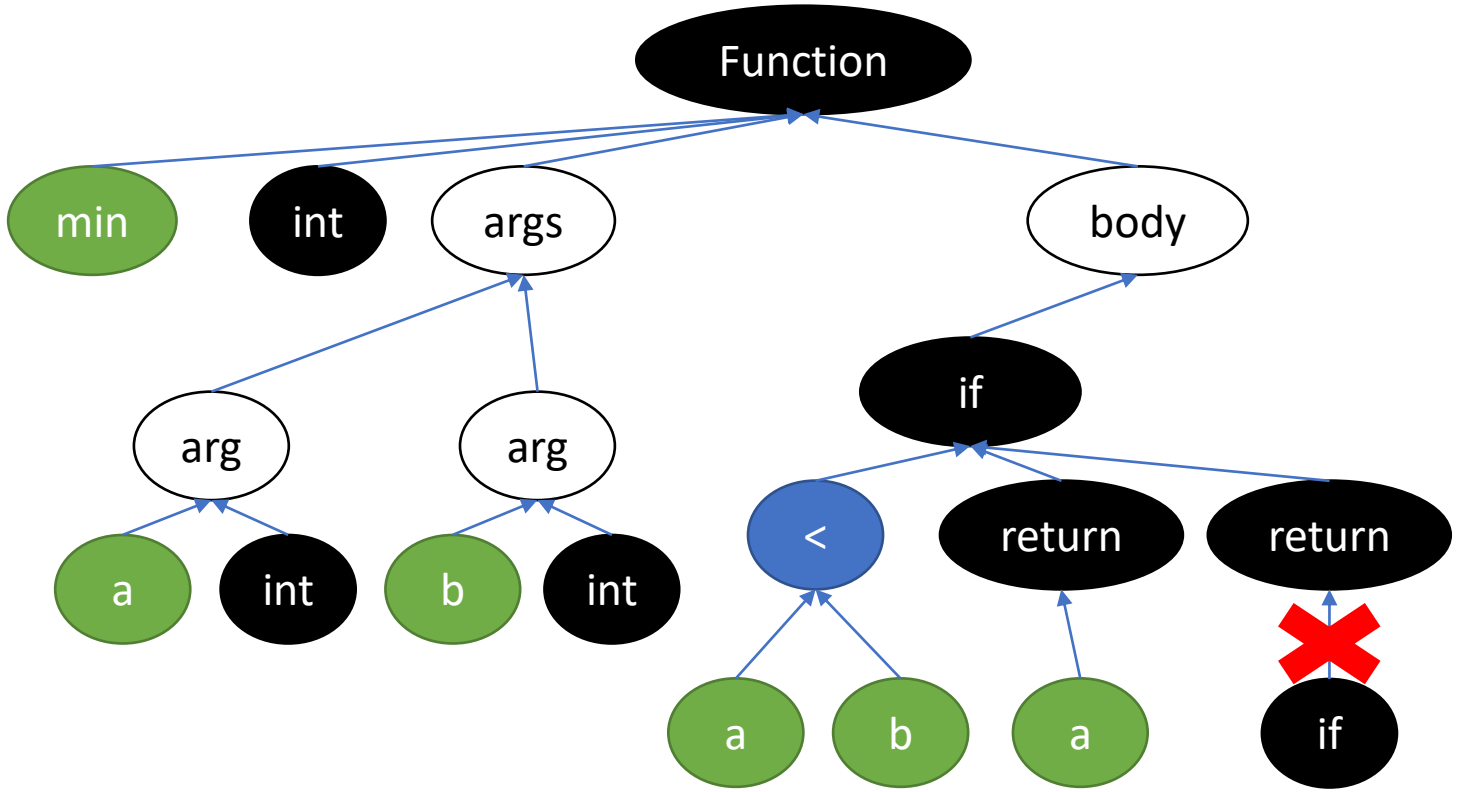
```
int min(int a, int b) {
    if (a < b)
        return a;
    else
        return b;
}
```

```
int min ( int a , int b ) { if ( a < b
    ) return a ; else return b ; }
```

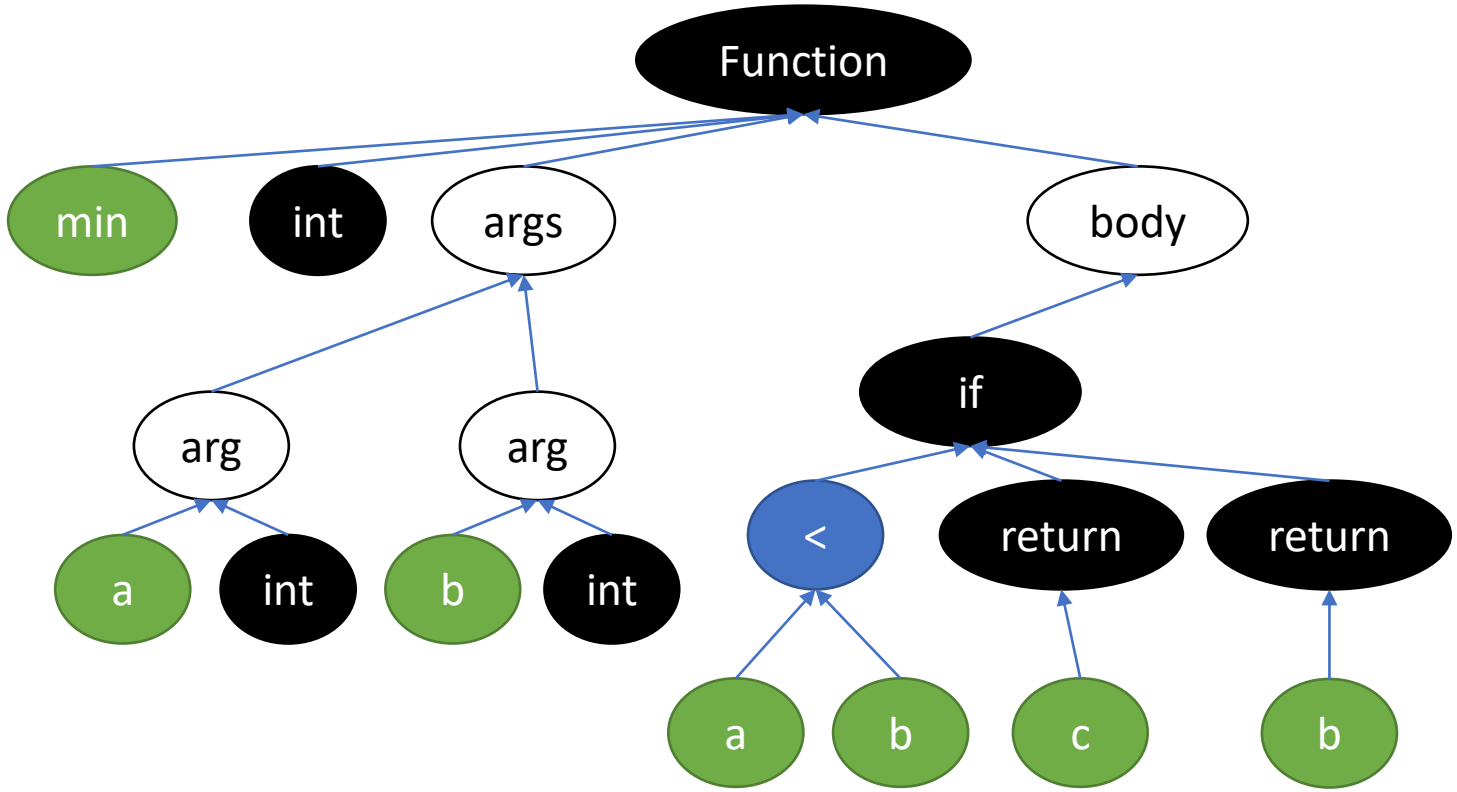
```
int min(int a, int b) {
    if (a < b)
        return a;
    else
        return b;
}
```



```
int min(int a, int b) {
    if (a < b)
        return if;
    else
        return b;
}
```

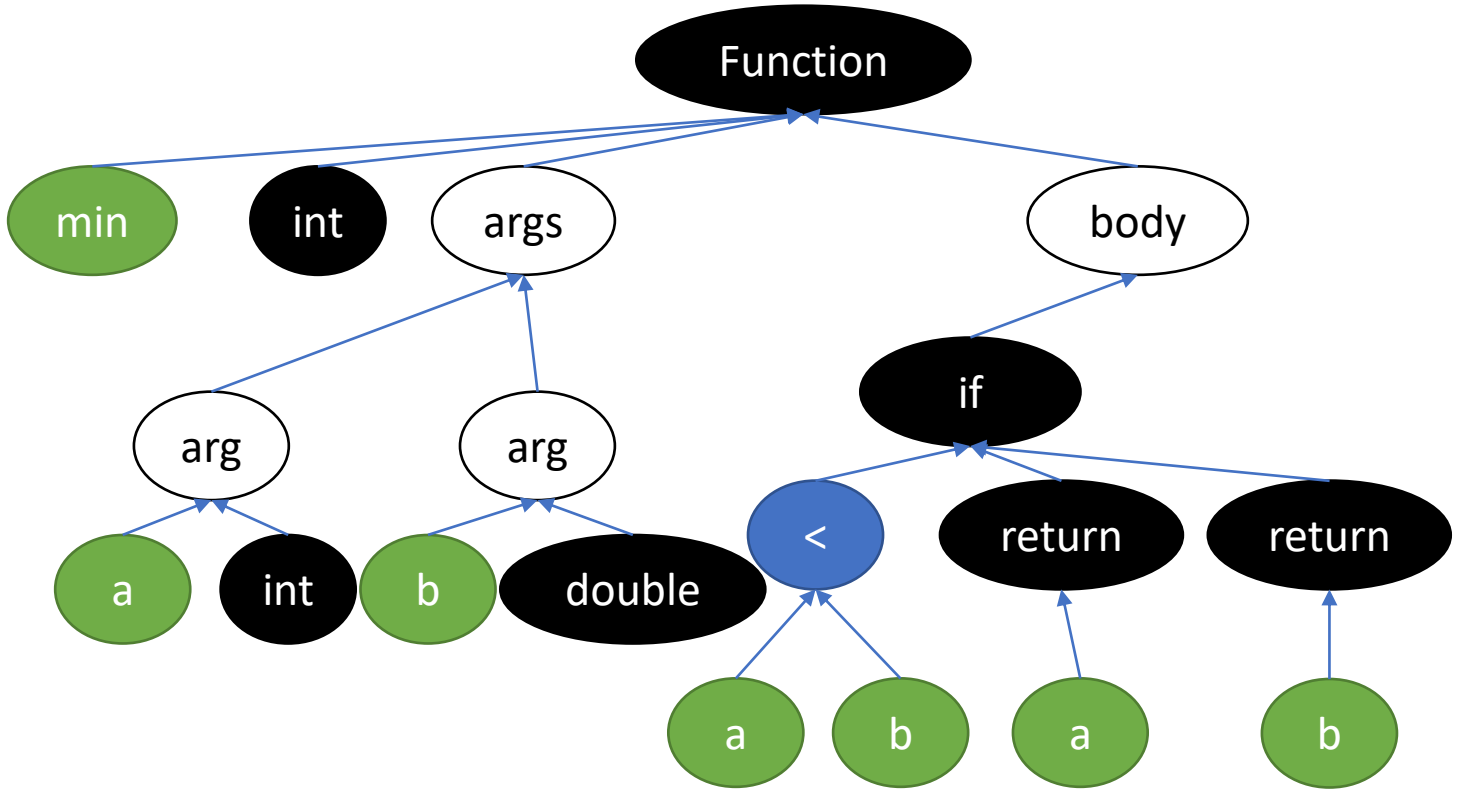


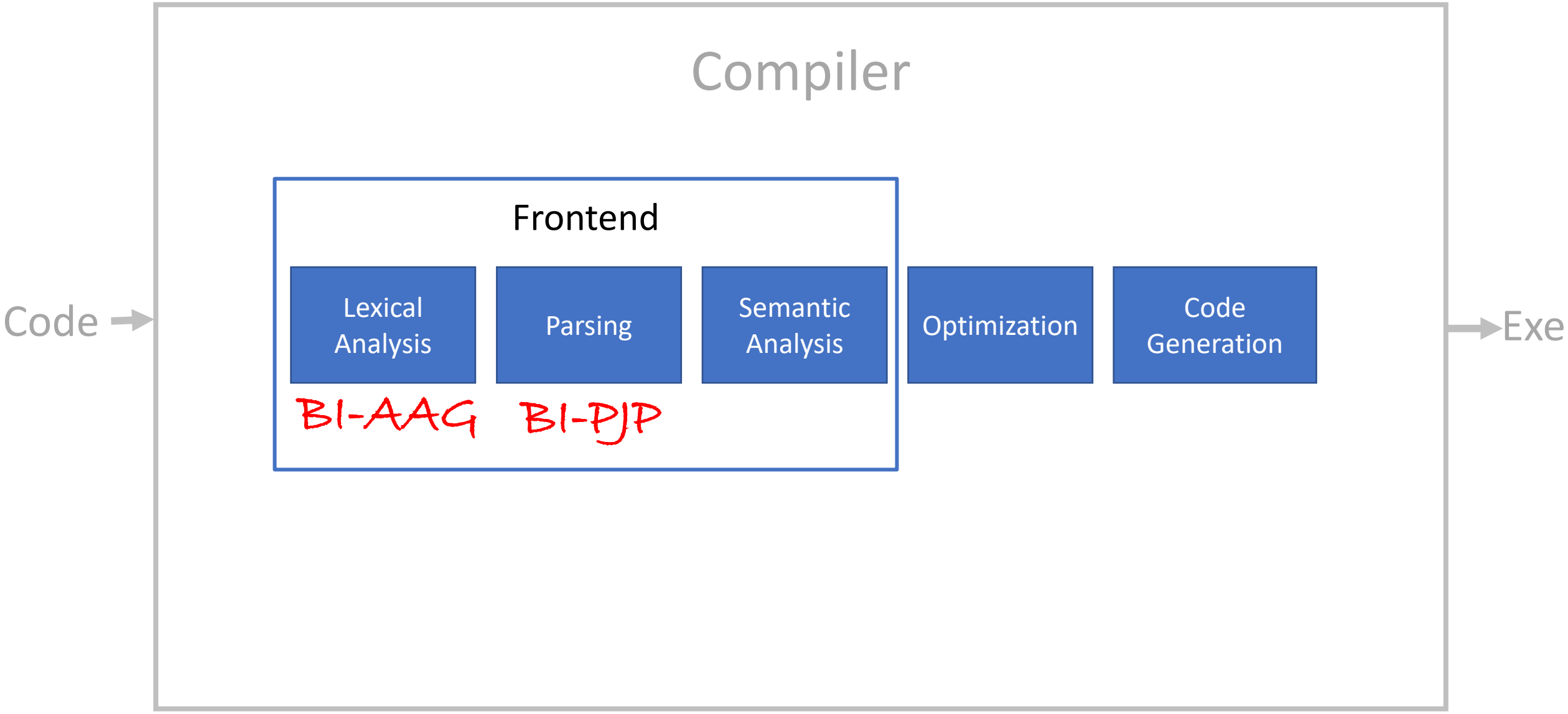
```
int min(int a, int b) {
    if (a < b)
        return c; valid...
    else
        return b;
}
```

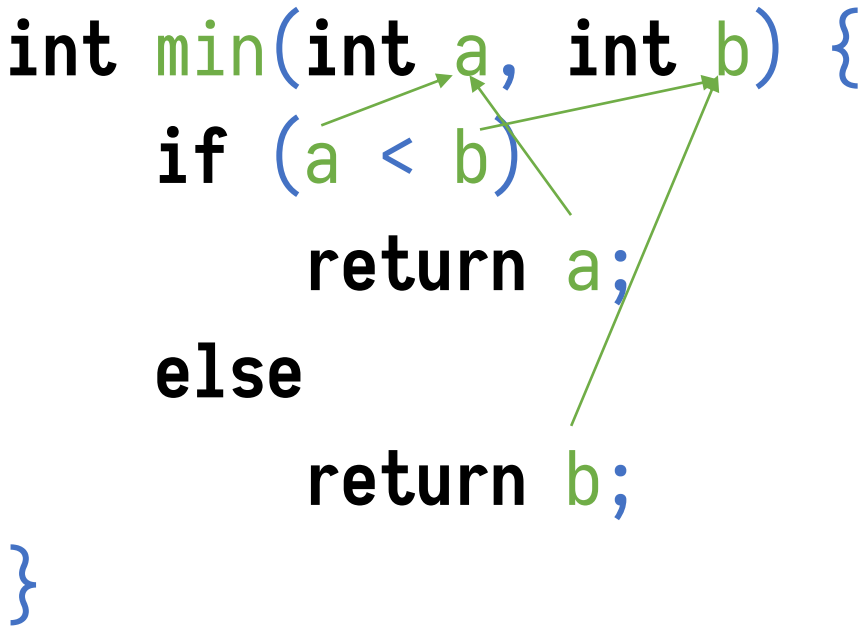


valid...

```
int min(int a, double b) {
    if (a < b)
        return a;
    else
        return b;
}
```



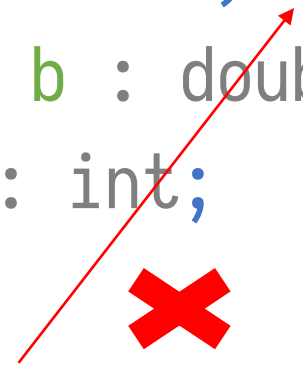

Compiler
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Optimization
Code Generation
Code
Exe
BI-AAG
BI-PJP

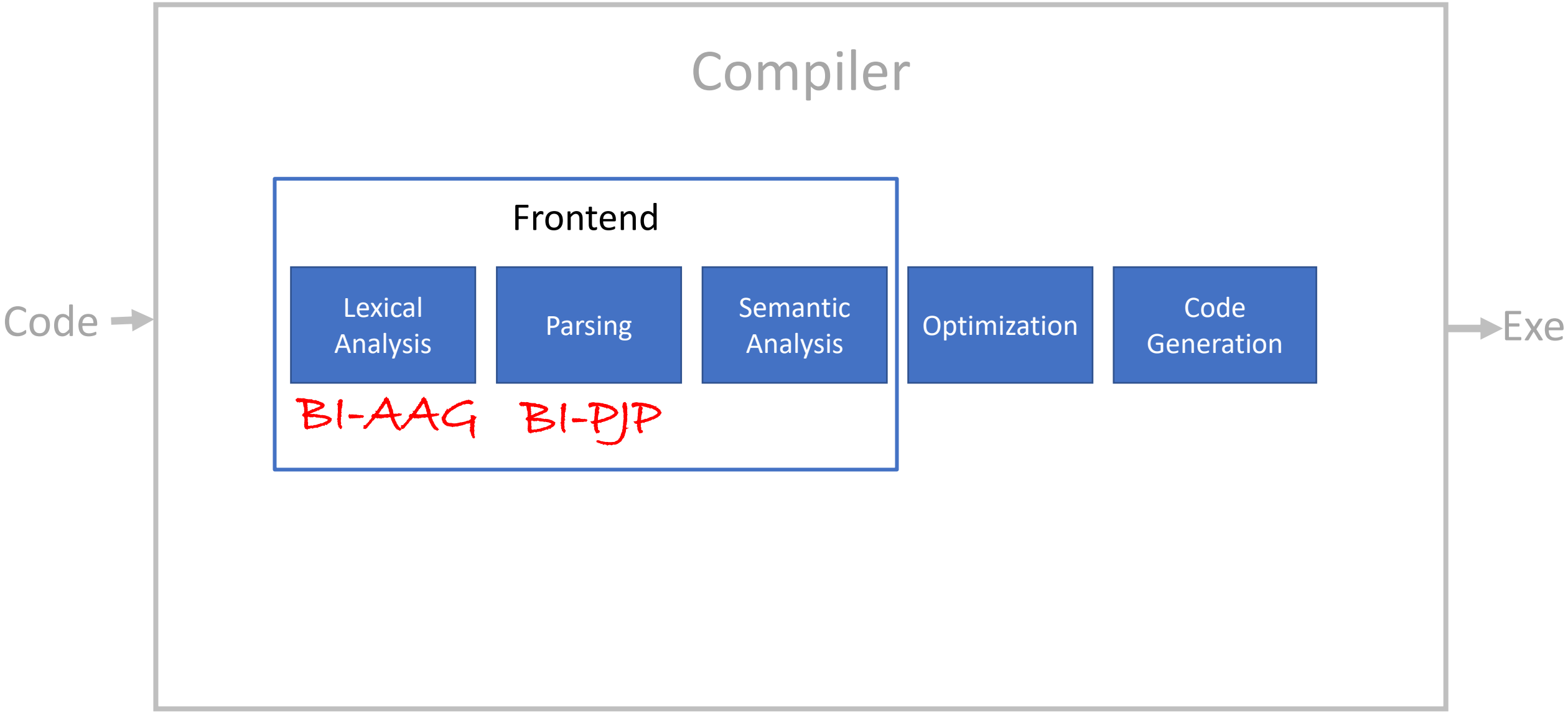

int min(int a, int b) {
if (a < b)
return a;
else
return b;
}

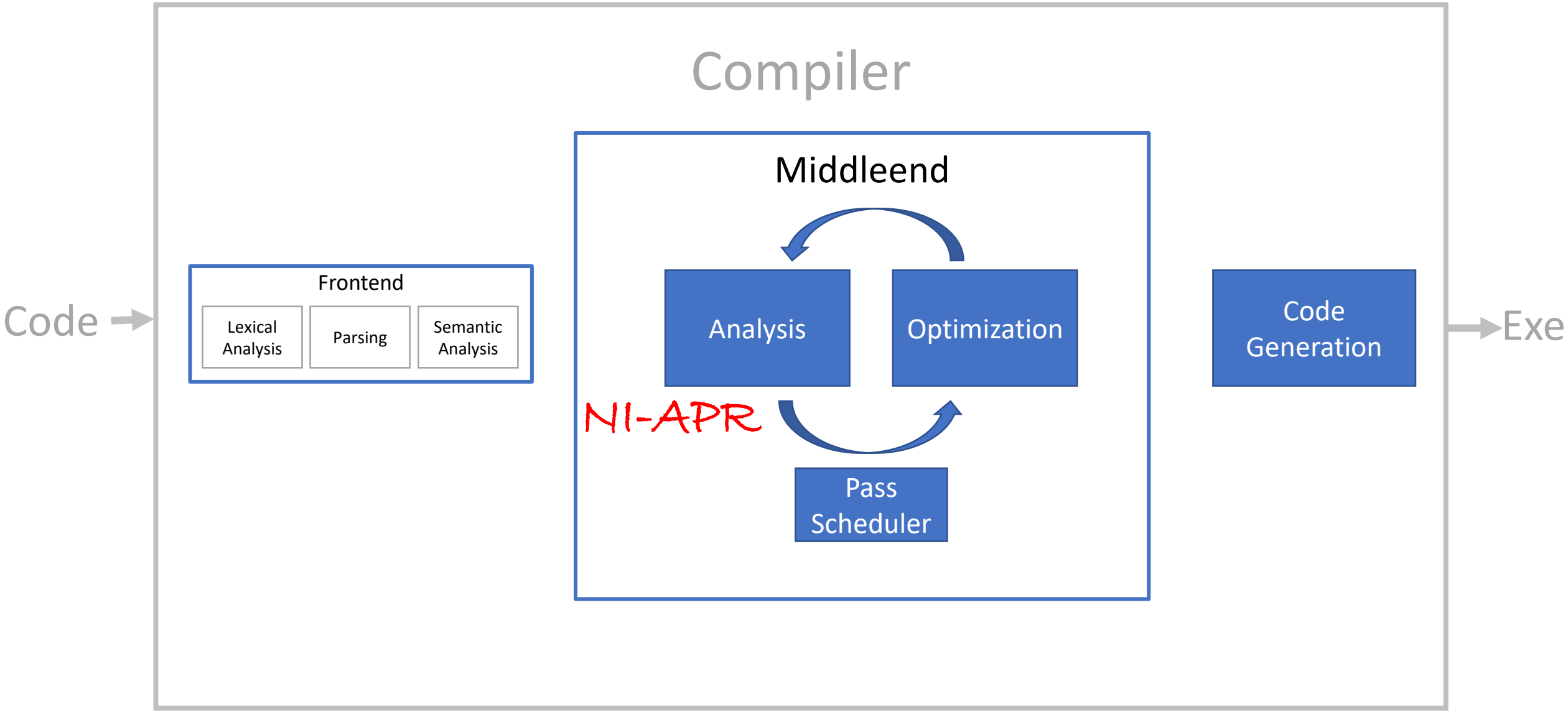
```
int min(int a, int b) {
    if (a < b)
        return a;
    else
        return c;
}
```

```
int min(int a, int b) : int {
    if (a : int < b : int) : bool
        return a : int;
    else
        return b : int;
}
```

```
int min(int a, double b) : int {
    if (a : int < b : double) : bool
        return a : int;
    else
        return b : double;
}
```




Compiler
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Optimization
Code Generation
Code
Exe
BI-AAG
BI-PJP


Compiler
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
NI-APR
Pass Scheduler
Code Generation
Exe

```
int min2(int a, int b) {
    if (a * 2 * 1 < b * 2)
        return a * 2;
    else
        return a * 2;
}
```

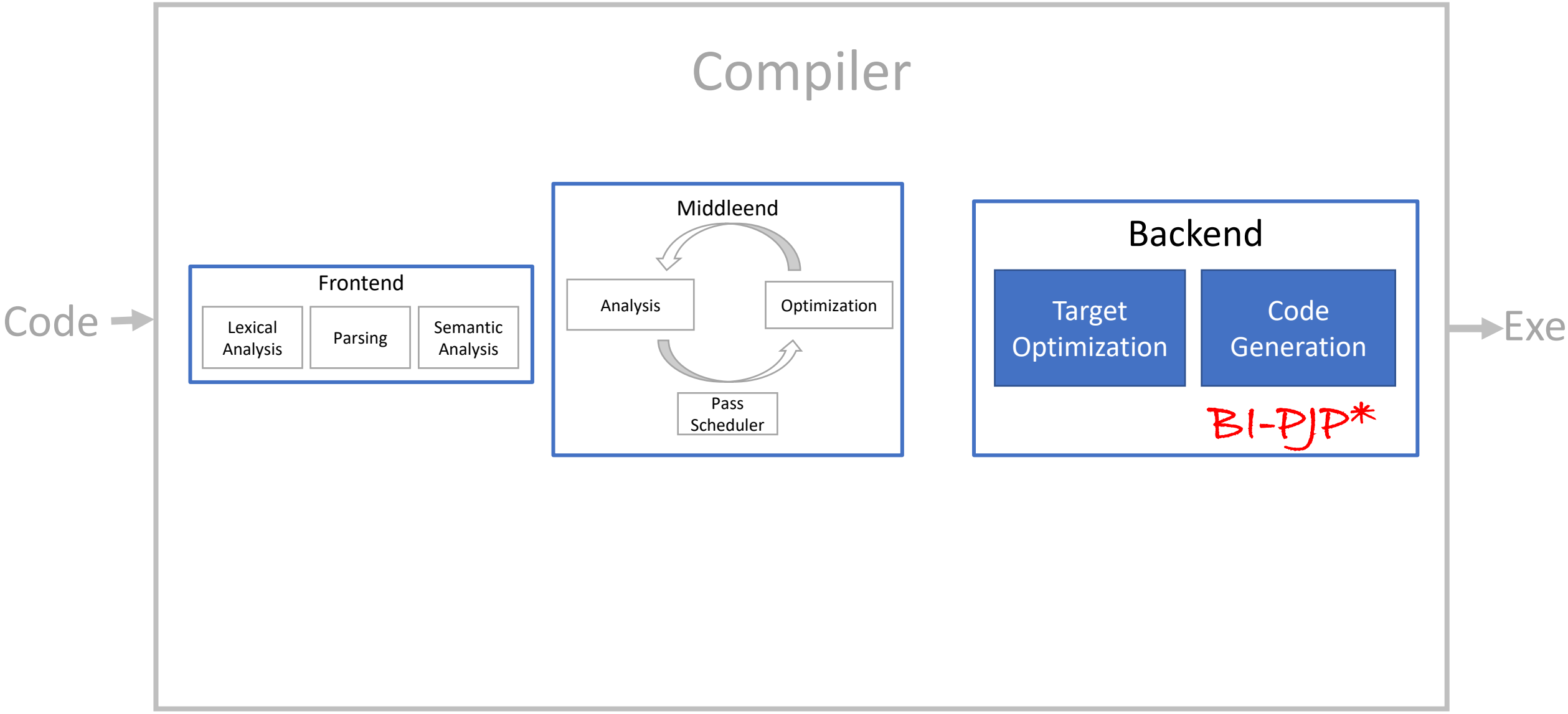
```
int min2(int a, int b) {
    if (a * 2 < b * 2)
        return a * 2;
    else
        return a * 2;
}
```

```
int min2(int a, int b) {
    if (a << 1 < b << 1)
        return a << 1;
    else
        return a << 1;
}
```

```
int min2(int a, int b) {
    int tmp = a << 1;
    if (tmp < b << 1)
        return tmp;
    else
        return tmp;
}
```

```
int min2(int a, int b) {
    int tmp = a << 1;
    return tmp;
}
```

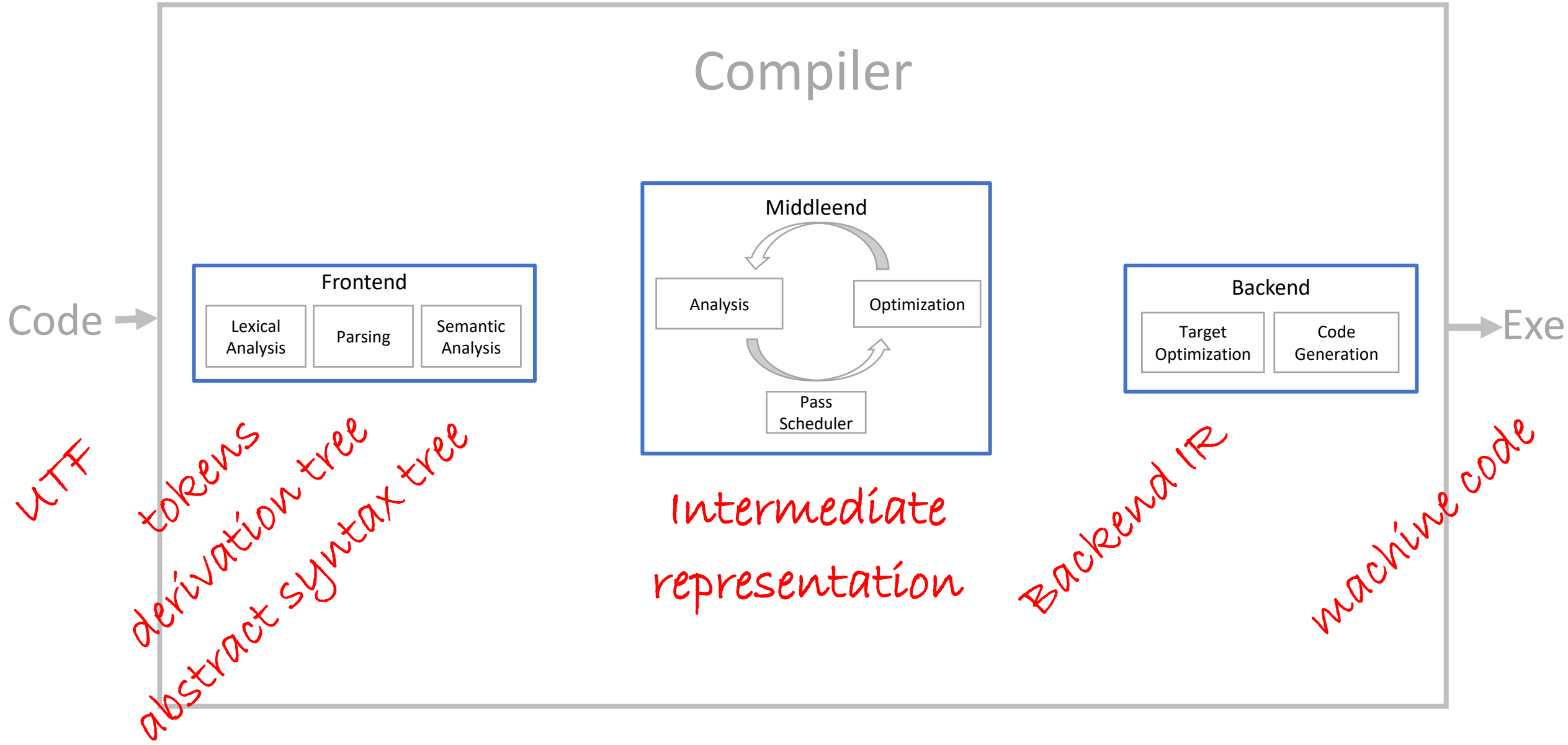
```
int min2(int a, int b) {
    return a << 1;
}
```


Compiler
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
BI-PJP*
Exe

```
bool lessThanZero(int a, int b) {
    return a + b < 0;
}
```

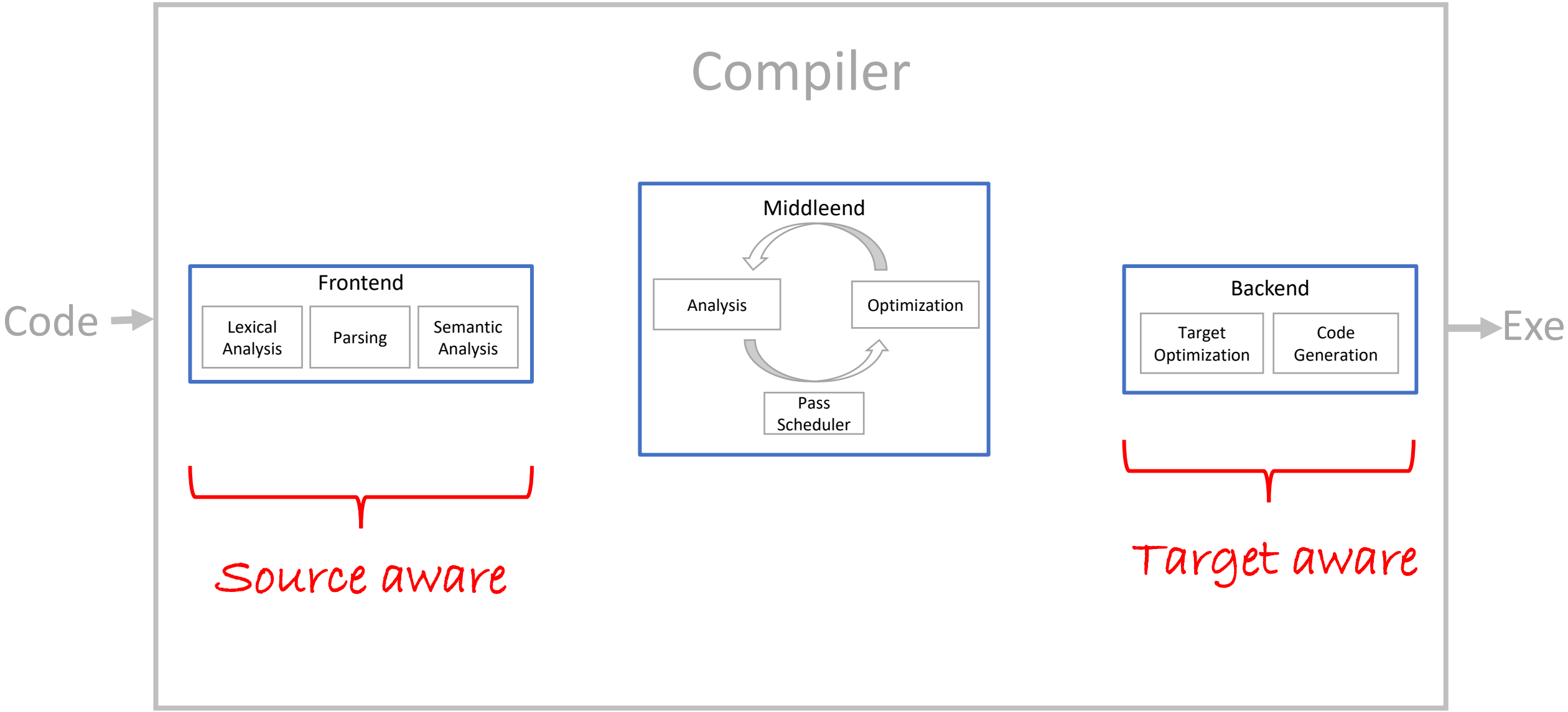
```
lessThanZero:
    mov cx, 0
    add ax, bx ; a + b
    cmp ax, cx ; <
    jl less
    mov ax, 0 ; return false
    ret
less:
    mov ax, 1 ; return true
    ret
```

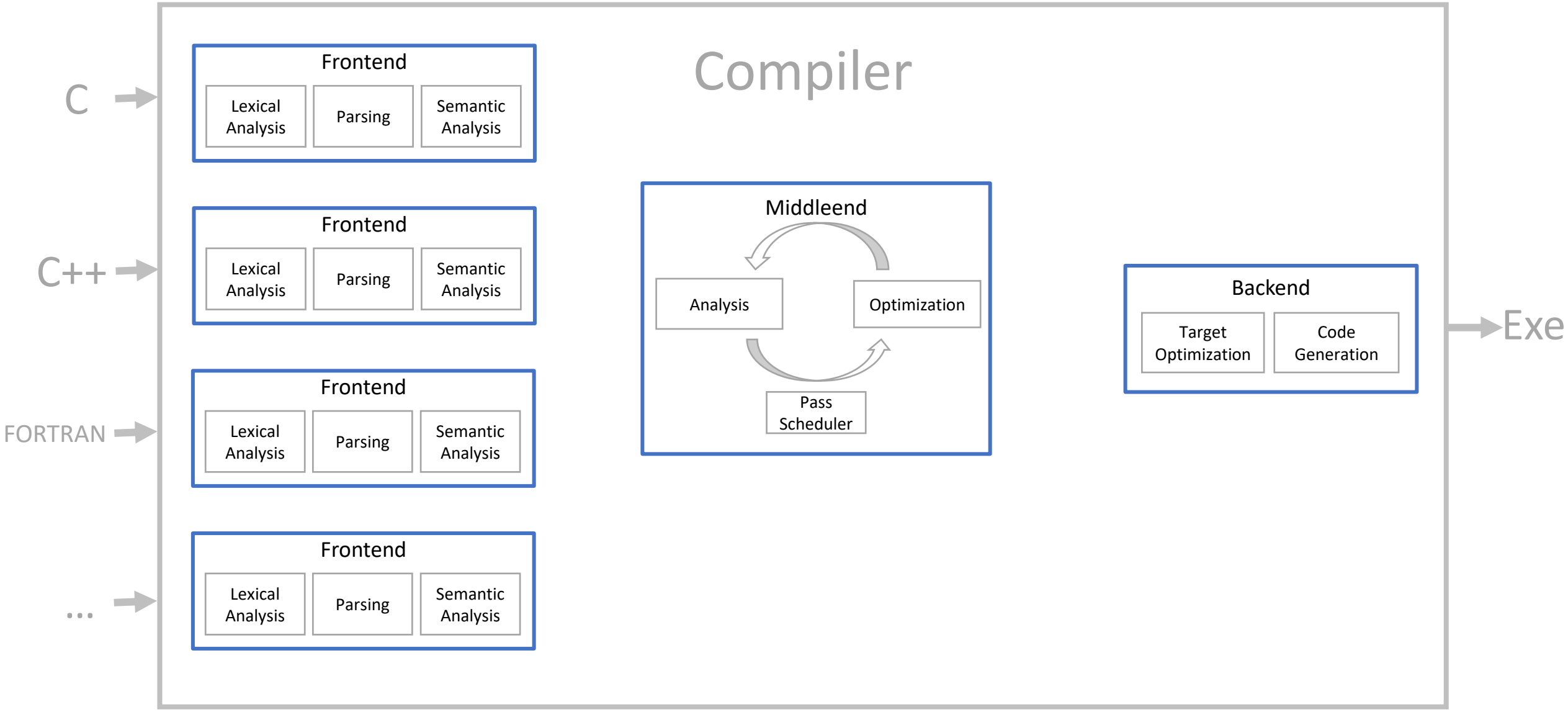
```
lessThanZero:
    mov cx, 0
    add ax, bx ; a + b
    cmp ax, cx ; <
    jl less
    and ax, cx ; (smaller and faster)
    ret
less:
    xor ax, ax ; (smaller and faster)
    ret
```

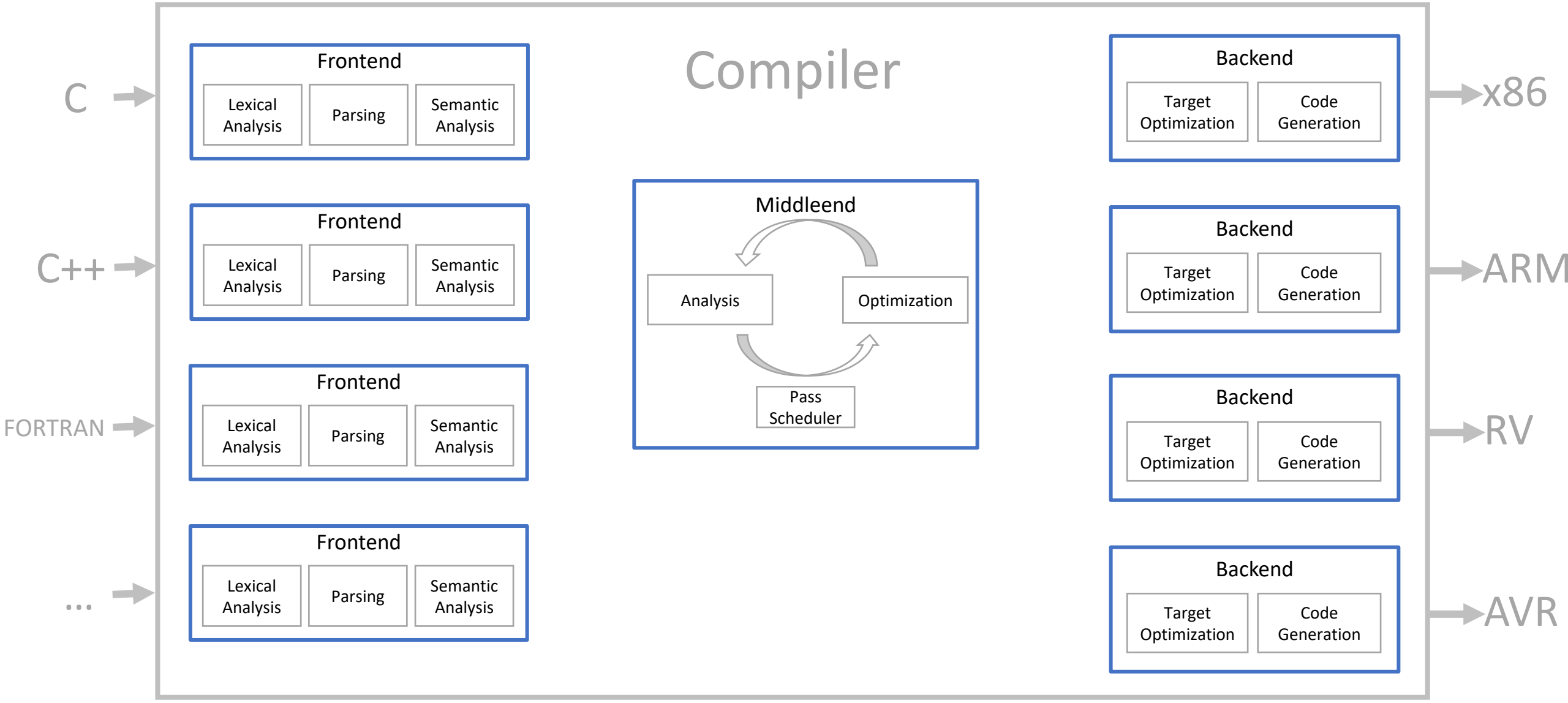

Compiler
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Exe
UTF
tokens
derivation tree
abstract syntax tree
Intermediate representation
Backend IR
machine code

This is a lot of work

for the programmer…


Compiler
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Source aware
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Target aware
Exe


C
C++
FORTRAN
...
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Compiler
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Exe


C
C++
FORTRAN
...
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Compiler
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Backend
Target Optimization
Code Generation
Backend
Target Optimization
Code Generation
Backend
Target Optimization
Code Generation
x86
ARM
RV
AVR

This is a lot of work

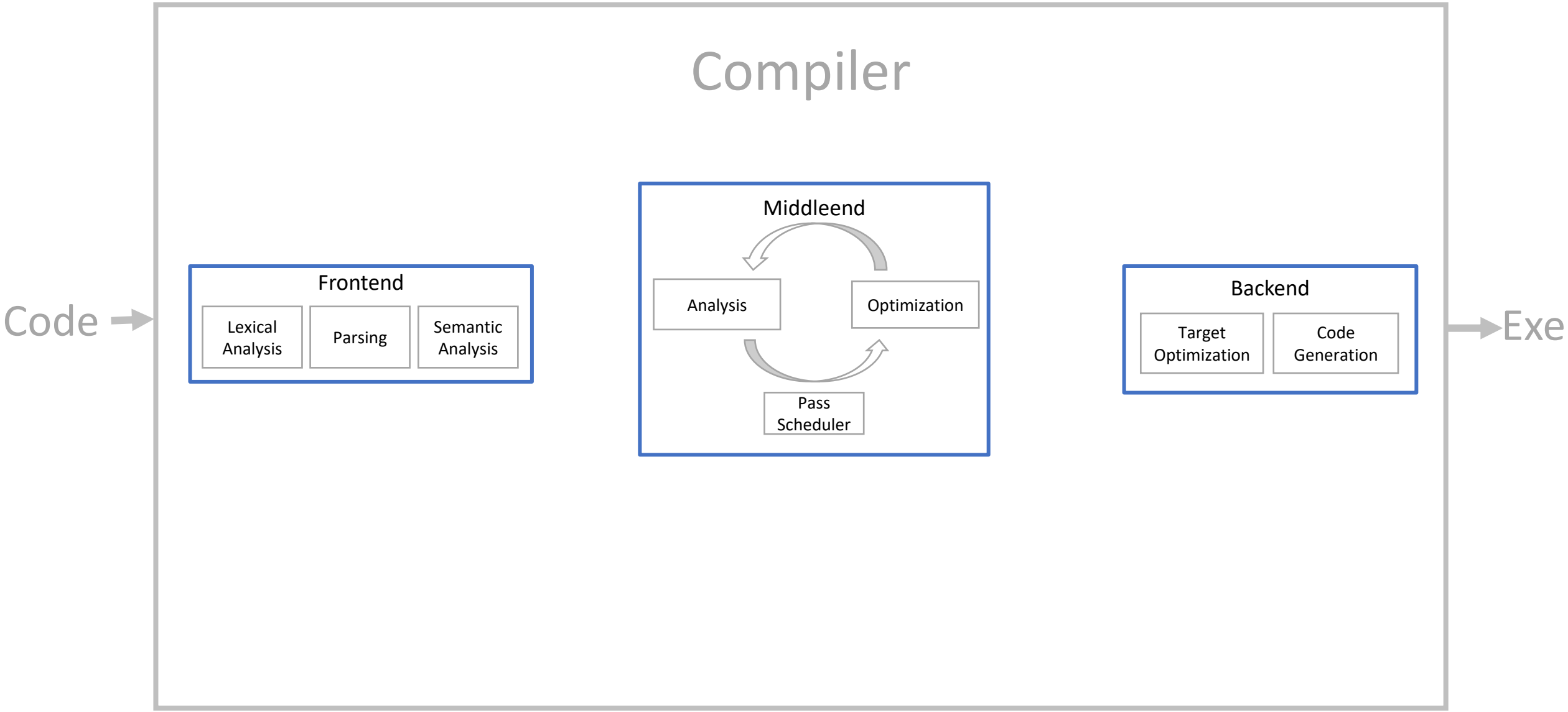
for the machine…

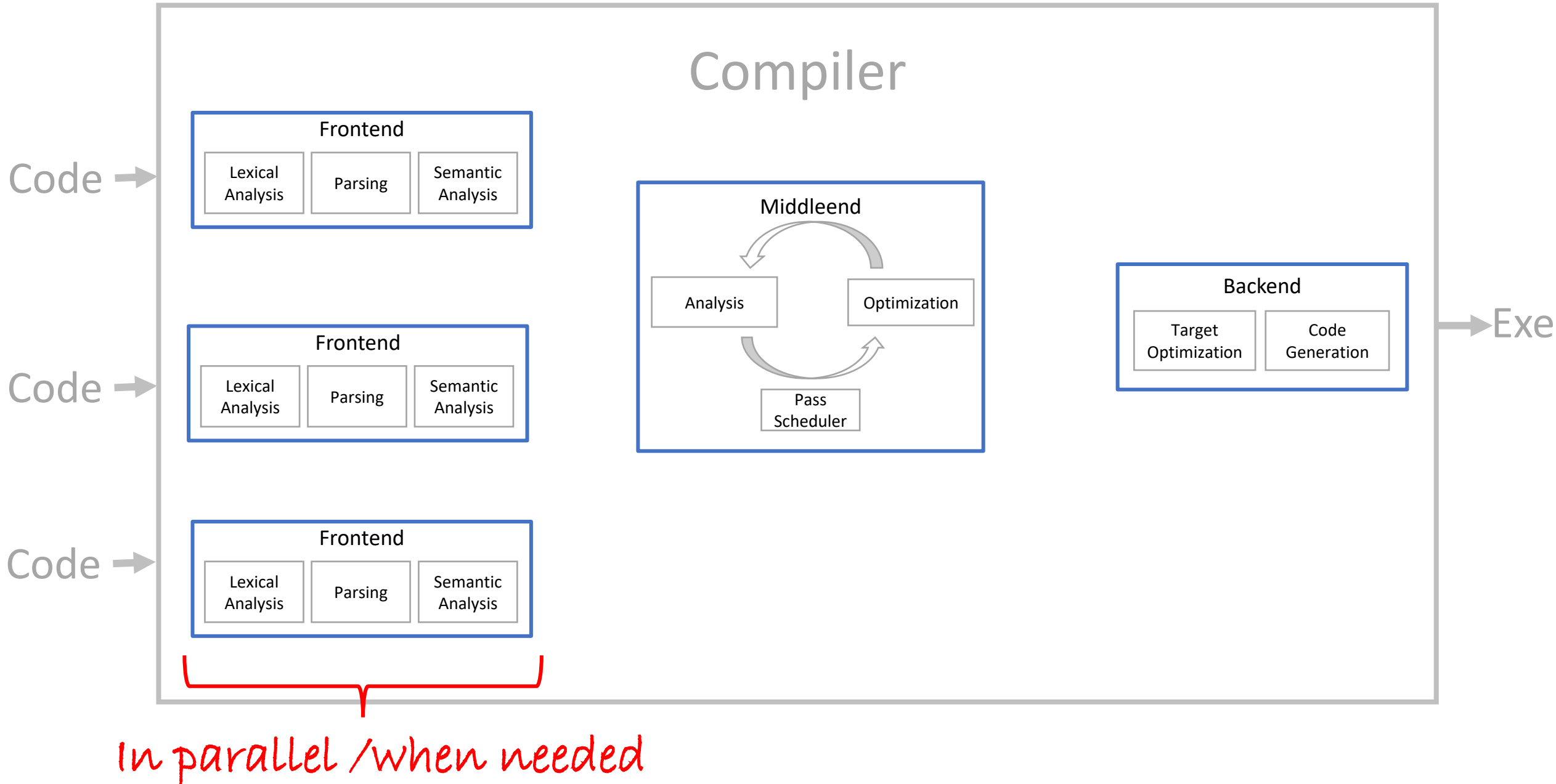
Compiler Efficiency

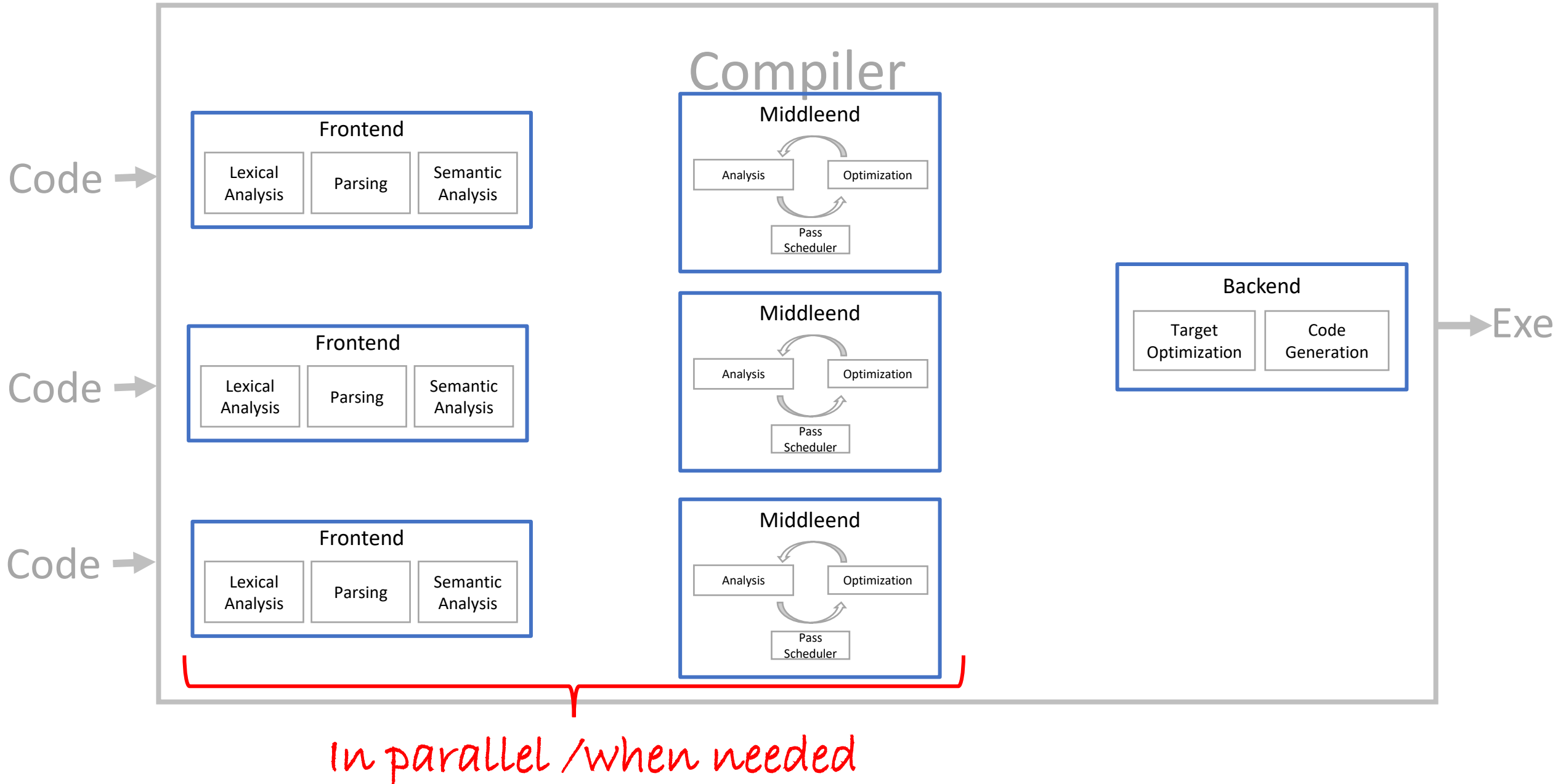
- not really that important, you compile once and then can run as many times as you want

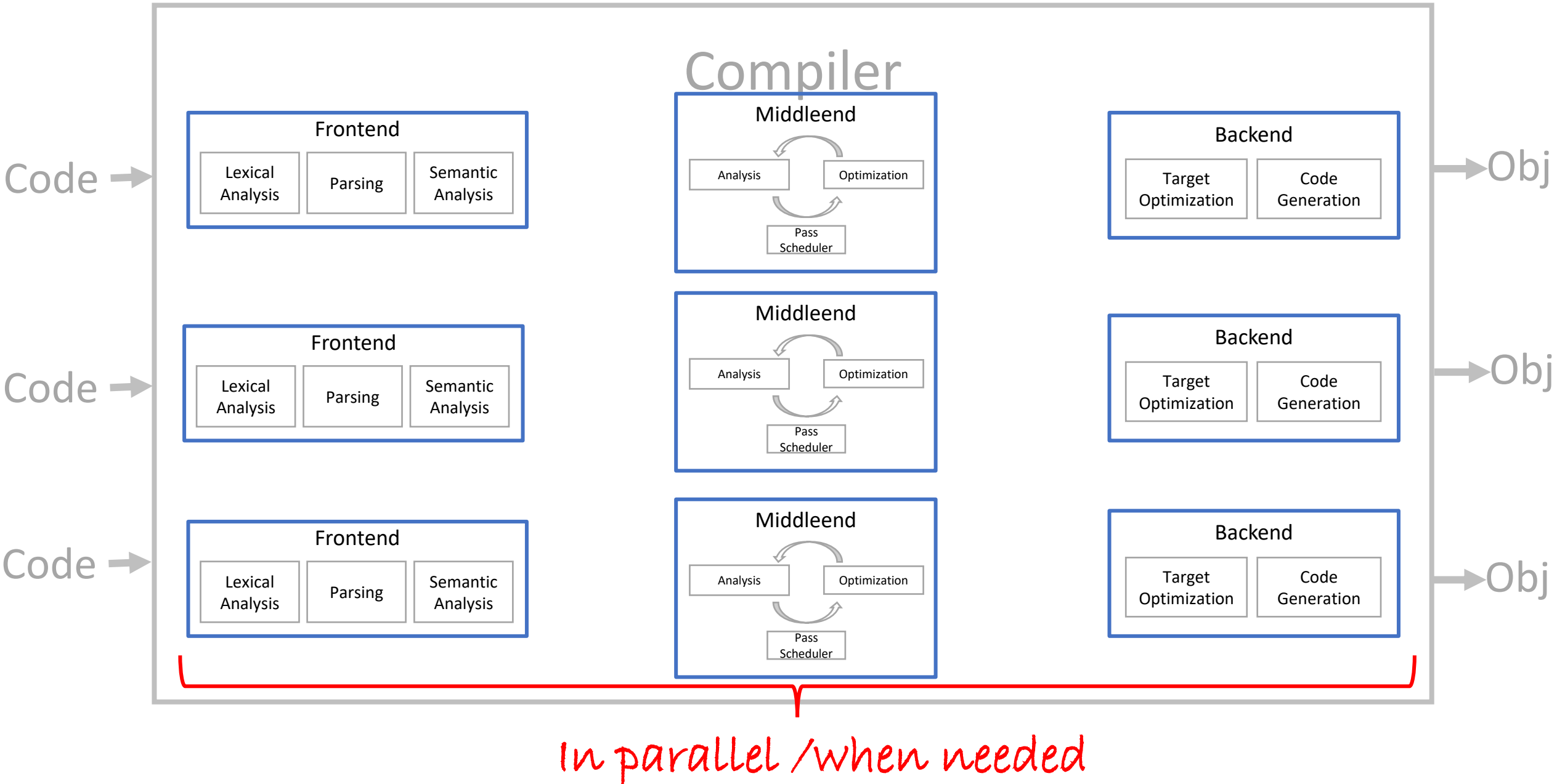
- except when you don't compile just once

we all make mistakes…


Compiler
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Exe


Compiler
Code
Code
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Exe
In parallel /when needed


Compiler
Code
Code
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Middleend
Analysis
Optimization
Pass Scheduler
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Exe
In parallel /when needed


Compiler
Code
Code
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Middleend
Analysis
Optimization
Pass Scheduler
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Backend
Target Optimization
Code Generation
Backend
Target Optimization
Code Generation
Obj
Obj
Obj
In parallel /when needed

```c
// file A

int min(int a, int b) {
    return a < b ? a : b;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

```c
// file B

int min(int a, int b);

int foo(int a) {
    return a + 3;
}

int main() {
    return min(foo(4), 10);
}
```

```c
// file A

int min(int a, int b) {
    return a < b ? a : b;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

```c
// file B

int min(int a, int b);

int foo(int a) {
    return a + 3;
}

int main() {
    return min(4 + 3, 10);
}
```

```c
// file A

int min(int a, int b) {
    return a < b ? a : b;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

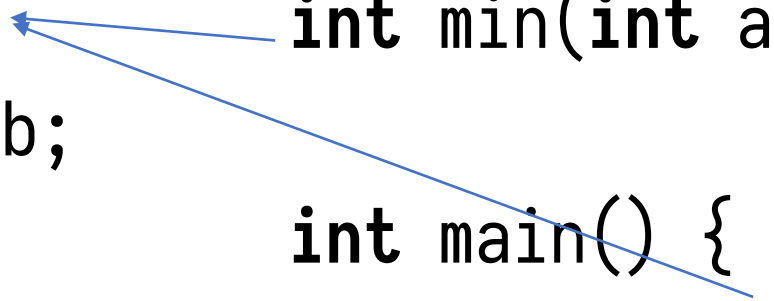
```c
// file B

int min(int a, int b);

int main() {
    return min(7, 10);
}
```

```
// obj A

int min(int a, int b) {
    return a < b ? a : b;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

```
// obj B

int min(int a, int b);

int main() {
    return min(7, 10);
}
```

```
// obj A

int min(int a, int b) {
    return a < b ? a : b;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

```
// obj B

int min(int a, int b);

int main() {
    return min(7, 10);
}
```



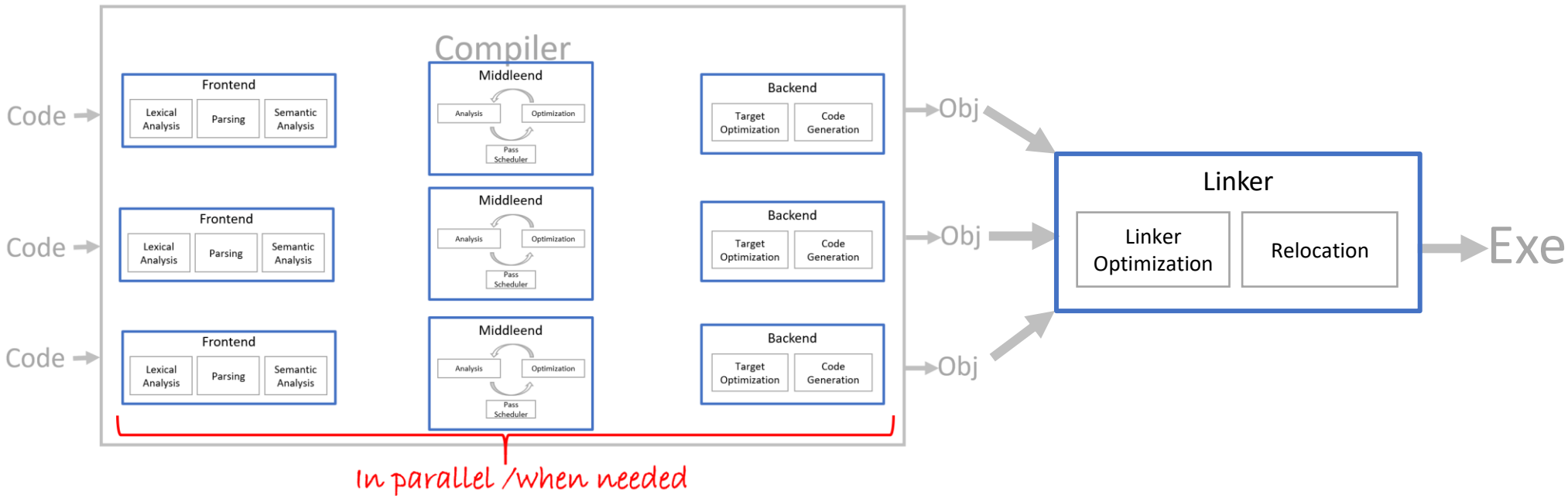
```
// obj A

int min(int a, int b) {
    return a < b ? a : b;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

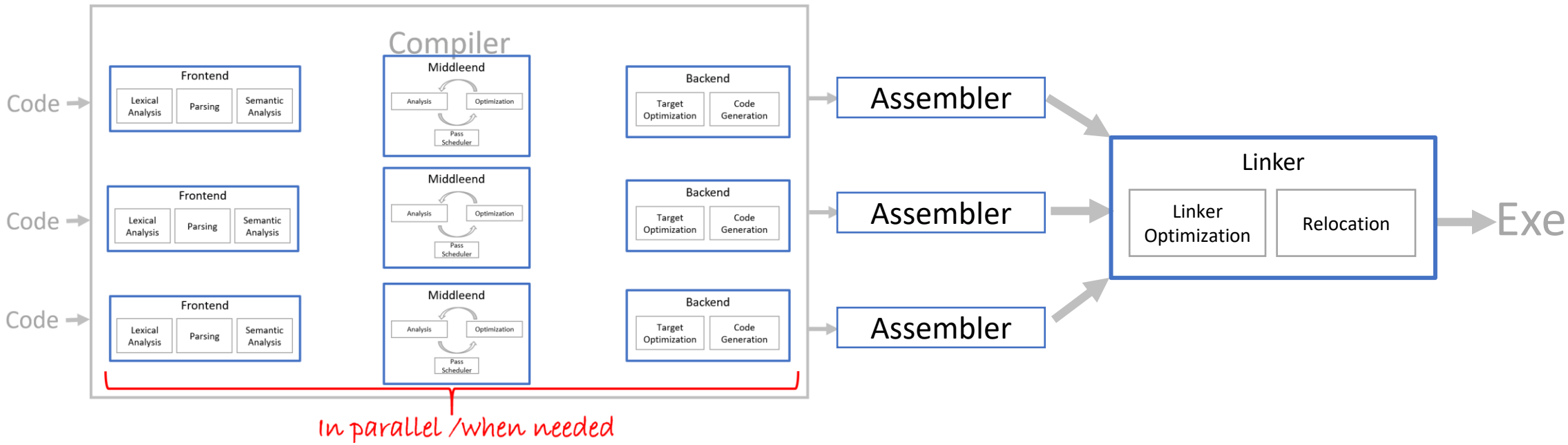
```
// obj B

int min(int a, int b);

int main() {
    return 7 < 10 ? 7 : 10;
}
```

```c
// obj A

int min(int a, int b) {
    return a < b ? a : b;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

```c
// obj B

int min(int a, int b);

int main() {
    return 7;
}
```

```
// executable

int main() {
    return 7;
}
```


Compiler
Code
Code
Code
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Obj
Obj
Obj
Linker
Linker Optimization
Relocation
Exe
In parallel /when needed


Code
Code
Code
Compiler
Frontend
Lexical Analysis
Parsing
Semantic Analysis
Middleend
Analysis
Optimization
Pass Scheduler
Backend
Target Optimization
Code Generation
Assembler
Assembler
Assembler
Linker
Linker Optimization
Relocation
Exe
In parallel /when needed