

Семинар по C++ №4

Компилятор

Компилятор

Компилятор - программа, переводящий написанный на одном языке программирования текст, в код на другом языке программирования

Примеры:

g++: C++ -> assembly

JDK: Java -> Java Byte Code

Стадии компиляции

- **Препроцессинг** - текстовая подстановка (-E)
 - **Компиляция** - преобразования кода в ассемблер (-S)
 - **Ассемблирование** - перевод ассемблерного файла в машинный код - объектный файл (-c)
 - **Линковка**
-
- Сохранить все промежуточные файлы: **--save-temps**

Ассемблерный код

Как посмотреть:

- На сайте godbolt.org
- Скомпилировать исходники:
gcc -S main.cpp
- Дизассемблировать исполняемый файл:
objdump -d -M intel a.out > disasm

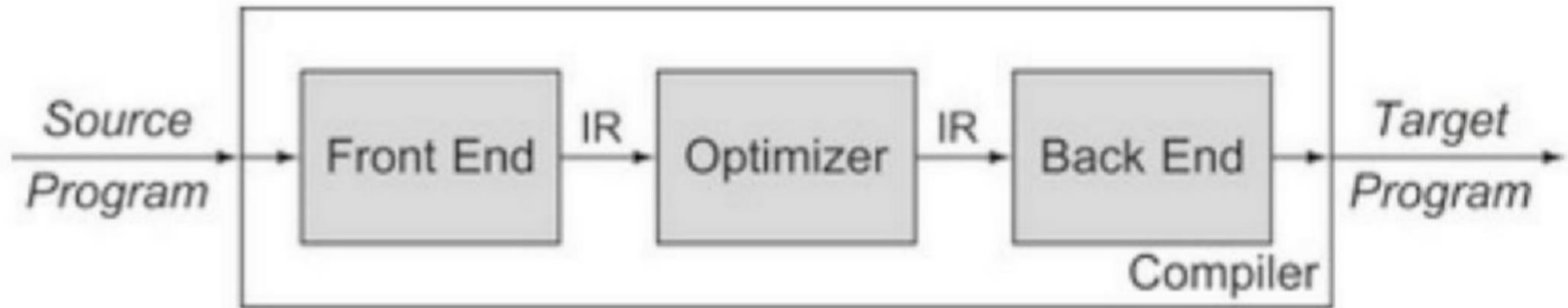
Как разбивать программу на файлы?

1. Объявления в `.hpp`
2. Реализация в `.cpp`
3. **`#pragma once` or `#ifndef`**
4. Линковать в один исполняемый файл:
 `g++ main.cpp my_lib.cpp`
 или
 `g++ main.o my_lib.o`

Интерпретатор

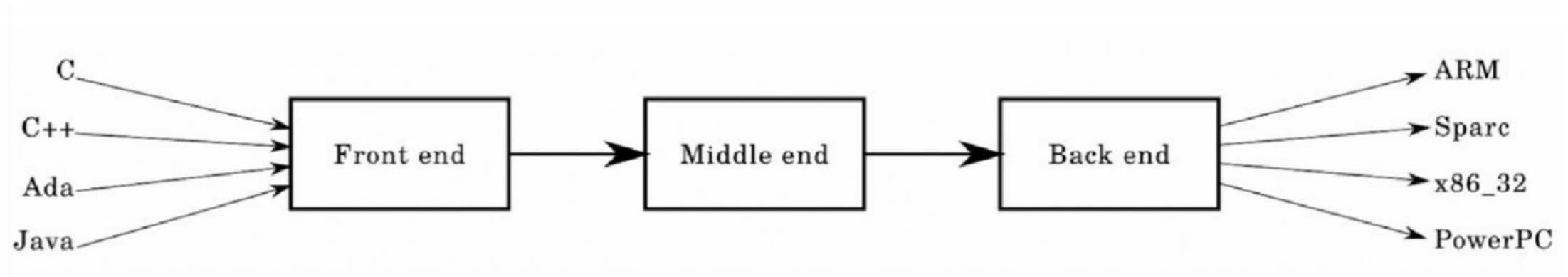
- Интерпретатор - компьютерная программа, которая выполняет инструкции без компиляции в машинный язык
- Примеры:
- Python, Bash, Make

Структура компилятора



- **Frontend** - трансляция исходного кода в промежуточное представление (**IR** = **I**ntermediate **R**epresentation)
- **Optimizer** - оптимизация промежуточного представления кода
- **Backend** - трансляция промежуточного представления в исполняемый код

Преимущество такой декомпозиции



Frontend



Preprocessing

- `g++ -E main.cpp` - Only preprocess the file
- `g++ -DDEBUG main.cpp` - define something
- It expands:
 - `#include`
 - `#define / #ifdef - #else - #endif / #if`

Lexer

Source code (string)

"a+b*3+4*2"

Lexer

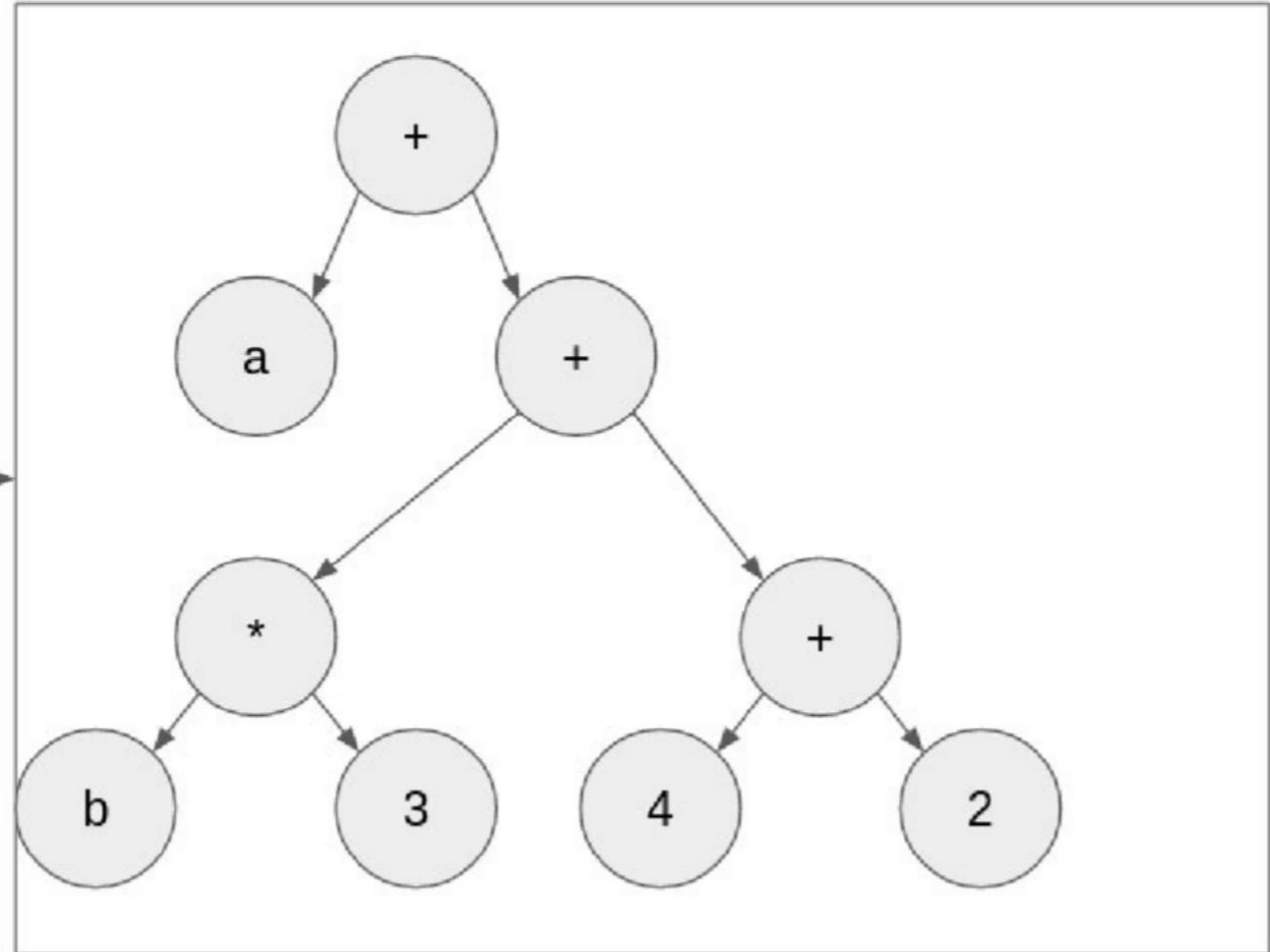
Tokens

[Id "a", Op +, Id b, Op *, Int 3, Op +, Int 4, Op *, Int 2]

Parser

[Id "a", Op +, Id b, Op *, Int 3,
Op +, Int 4, Op *, Int 2]


parser



Python (lexer + parser)

```
def foo(a):  
    a = 2  
    b = 3  
    return a + b
```

lexer + parser

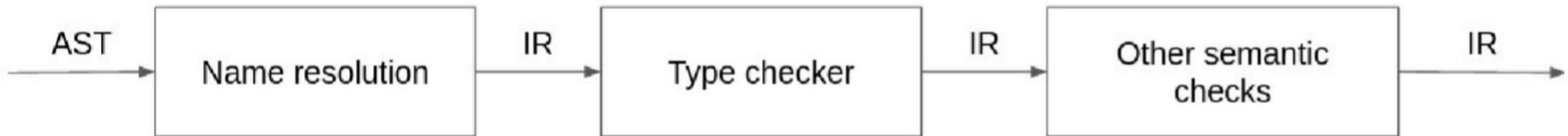


```
FunctionDef(  
    name='foo',  
    args=arguments(  
        posonlyargs=[],  
        args=[  
            arg(arg='a')],  
        kwonlyargs=[],  
        kw_defaults=[],  
        defaults=[]),  
    body=[  
        Assign(  
            targets=[  
                Name(id='a', ctx=Store())],  
            value=Constant(value=2)),  
        Assign(  
            targets=[  
                Name(id='b', ctx=Store())],  
            value=Constant(value=3)),  
        Return(  
            value=BinOp(  
                left=Name(id='a', ctx=Load()),  
                op=Add(),  
                right=Name(id='b', ctx=Load()))],  
            decorator_list=[])
```

Middle-end



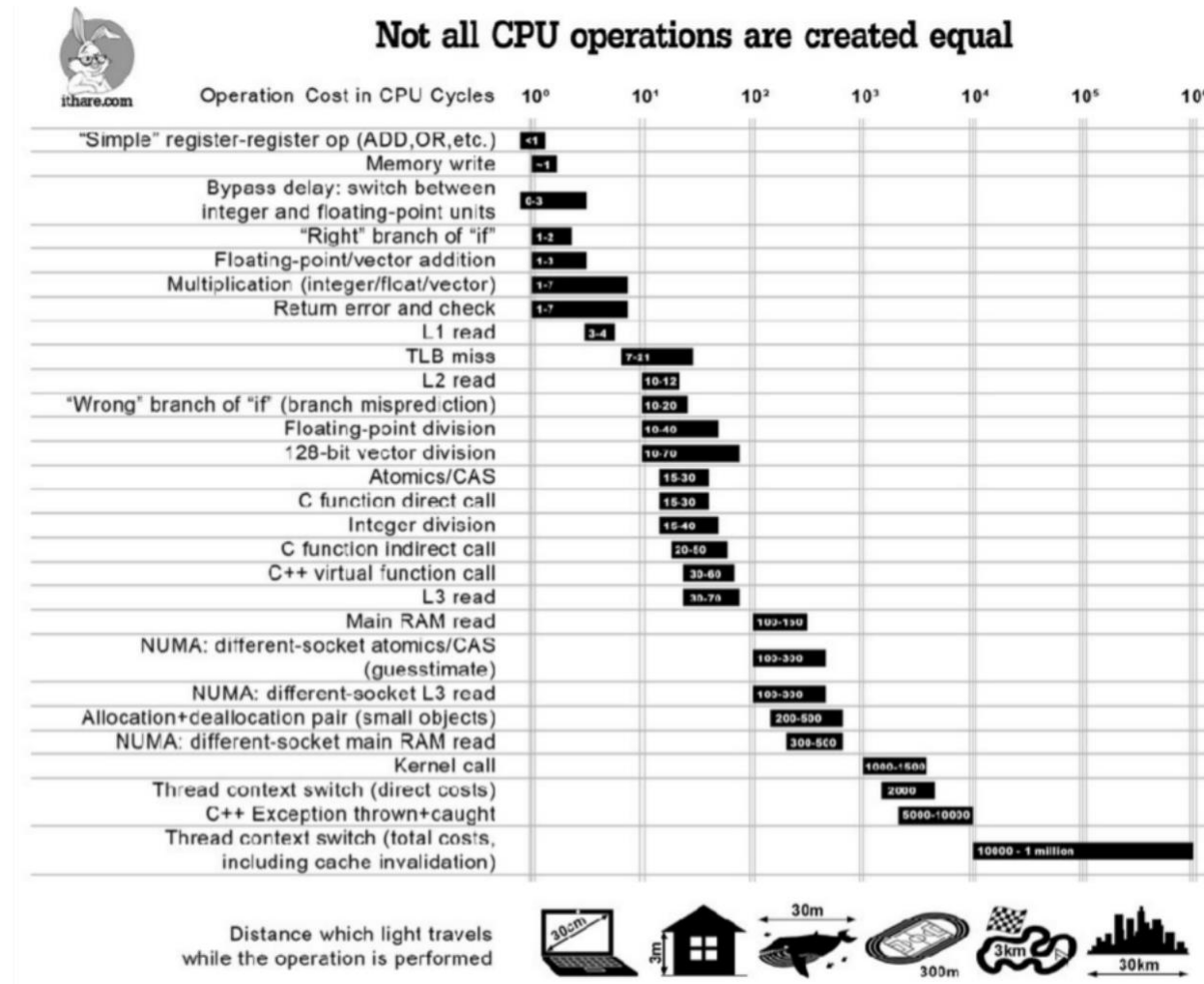
Semantic analysis



Code optimization

- Peephole optimization
- Loop optimizations
- Inlining
- Constant propagation
- Dead-code elimination
- <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>
- -O0, -O1, -O2, -O3, -Ofast, -Og

Speed of operations

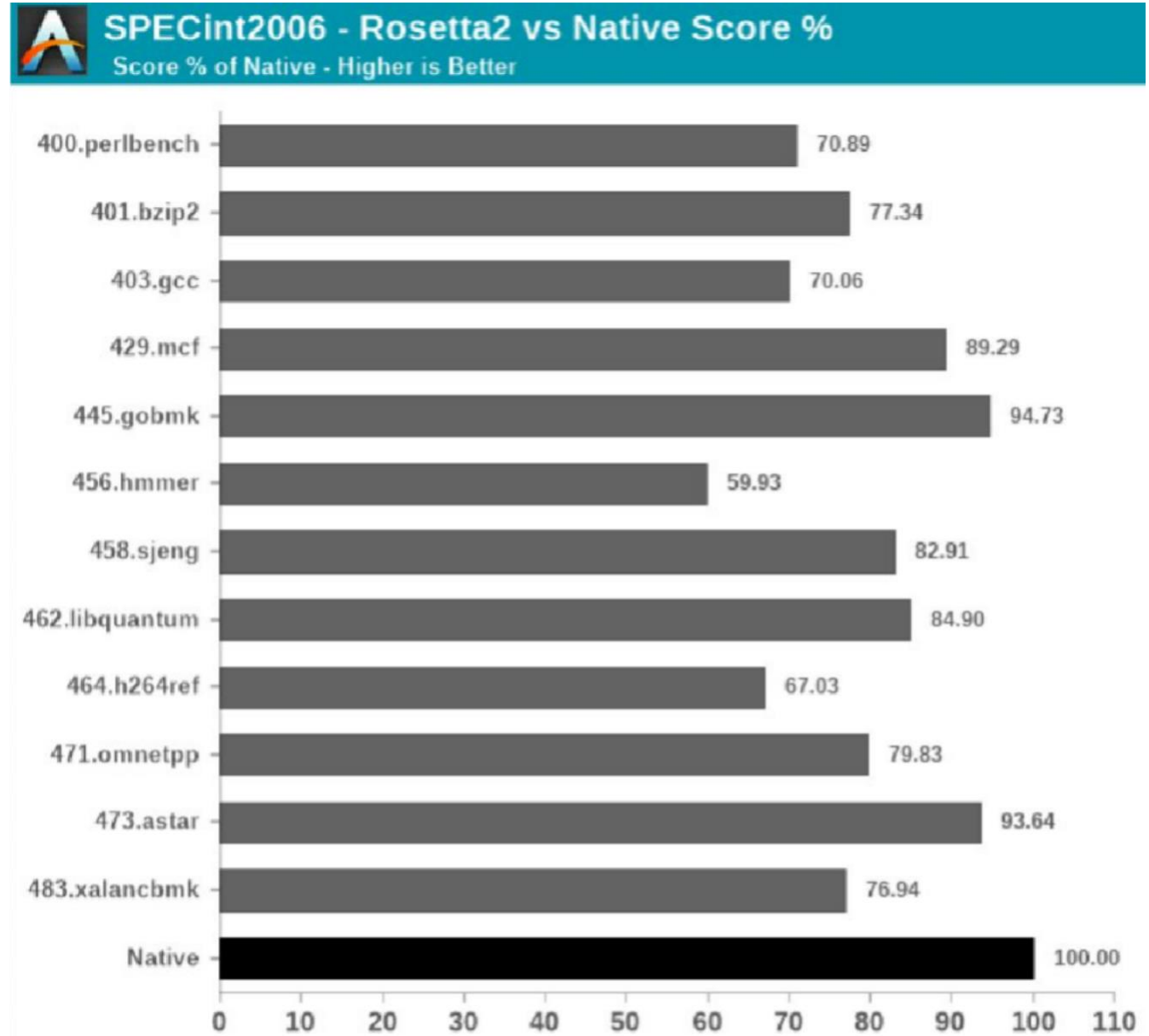


Warnings

- -Wall, -Wextra, -Werror, -Wpedantic
- <https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html>

Binary Translation

- Apple Rosetta
- 2006: PowerPC -> Intel
- 2020: Intel -> ARM (M1)



- Dragon Book
- Engineering a compiler

