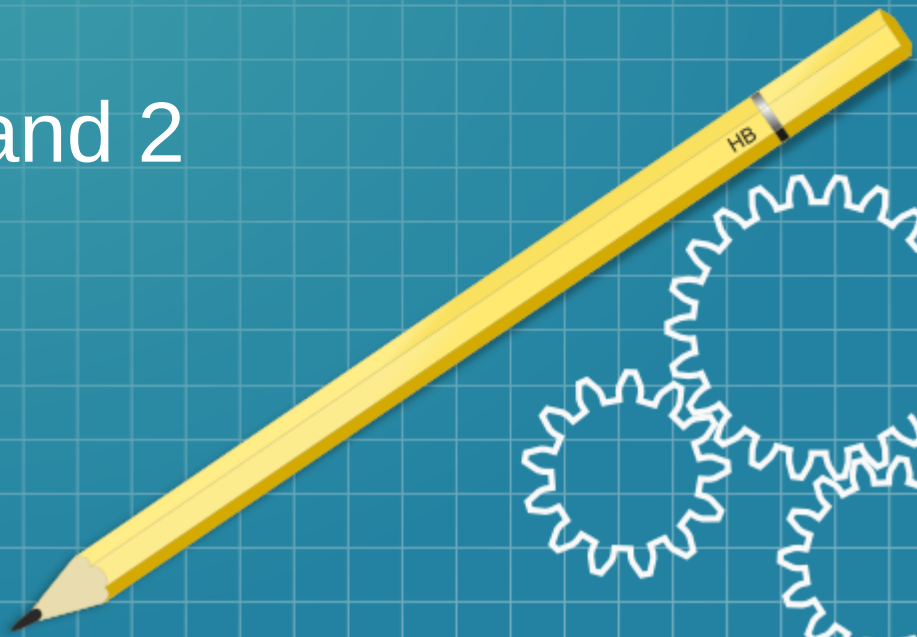
Introduction to C++

Chapters 1 and 2



Python and C++

- Python translation process:

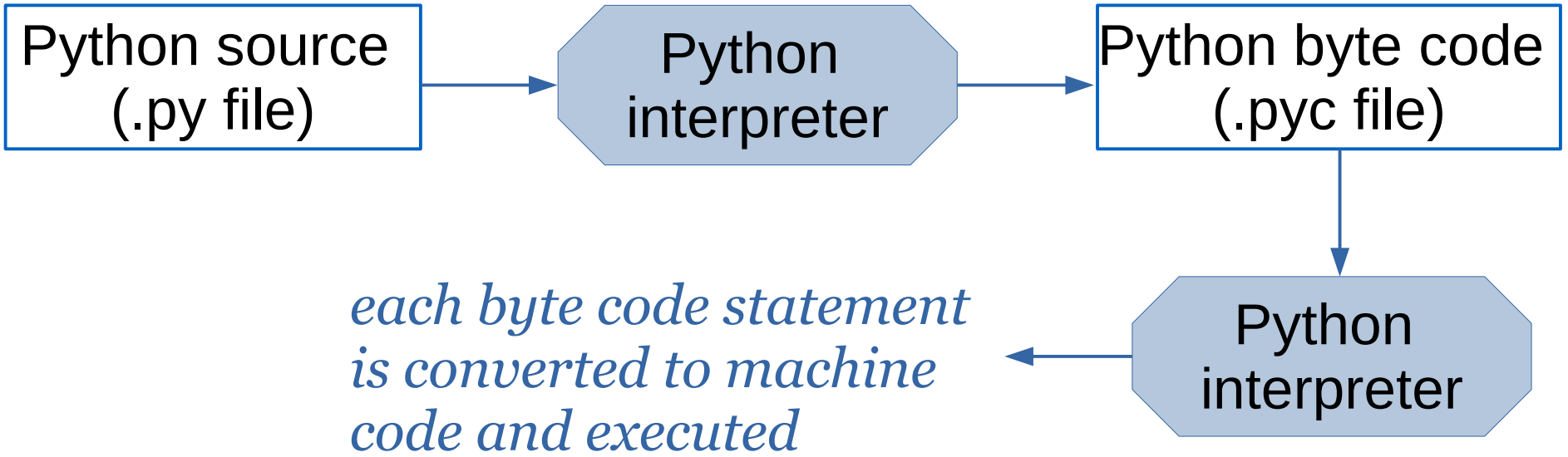
Python source
(.py file)

Python and C++

- Python translation process:

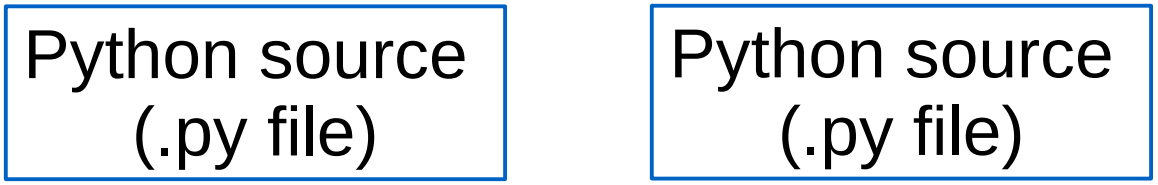
Python and C++

- Python translation process:



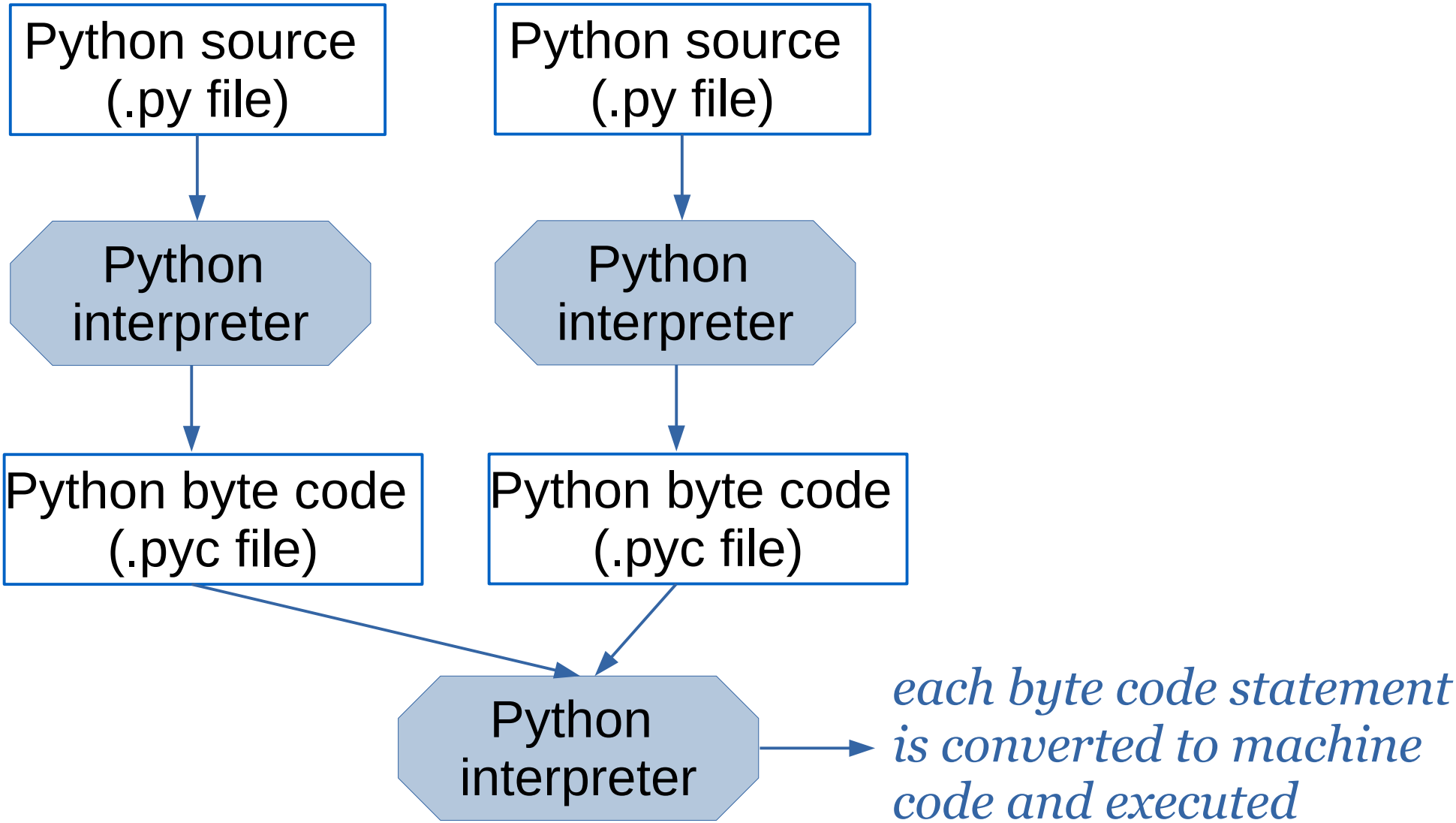
Python and C++

- Python translation process:



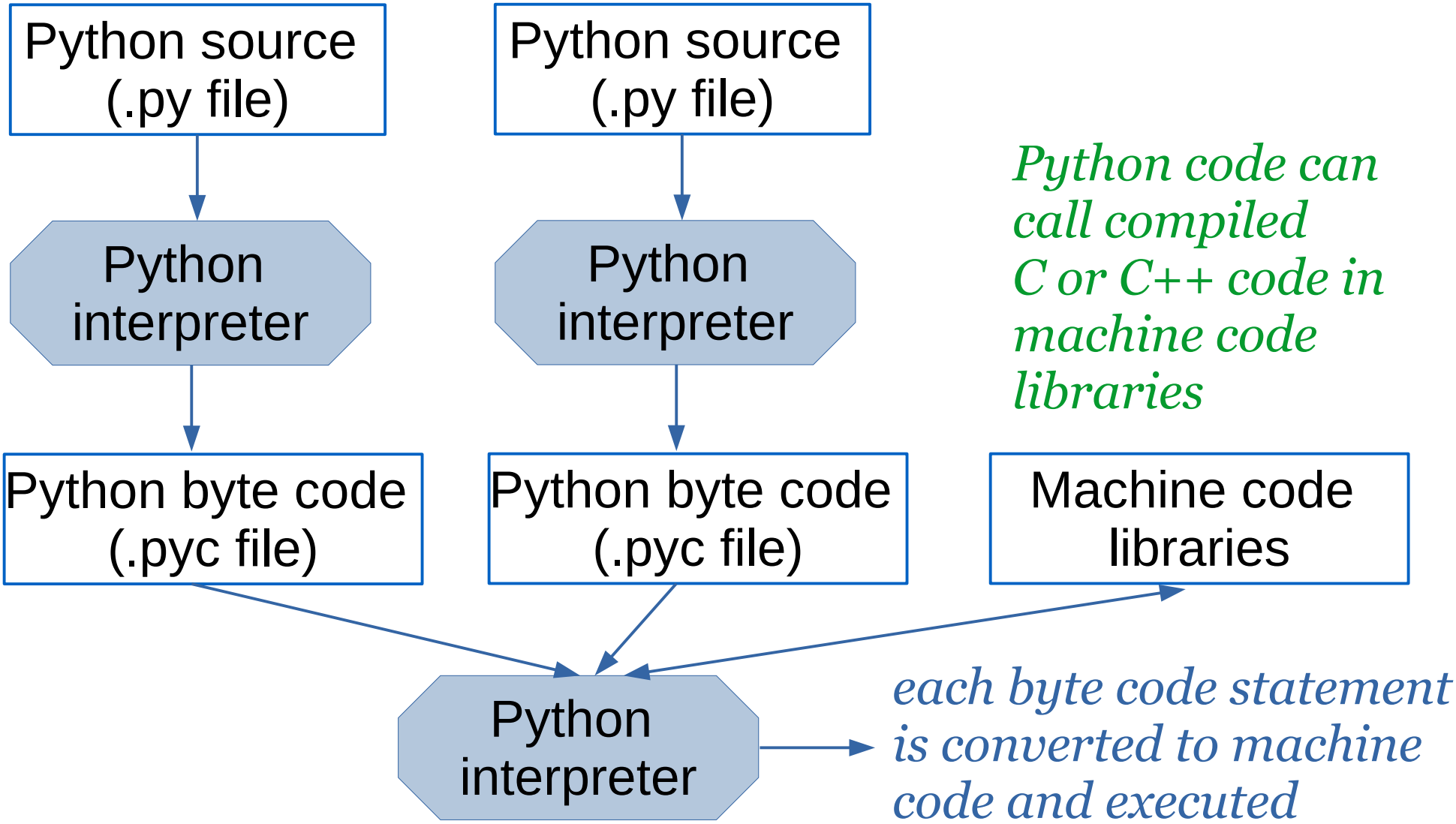
Python and C++

- Python translation process:



Python and C++

- Python translation process:



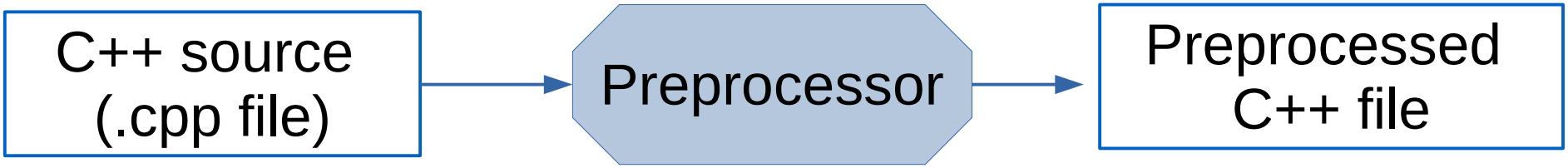
Python and C++

- C++ translation process:

C++ source
(.cpp file)

Python and C++

- C++ translation process:



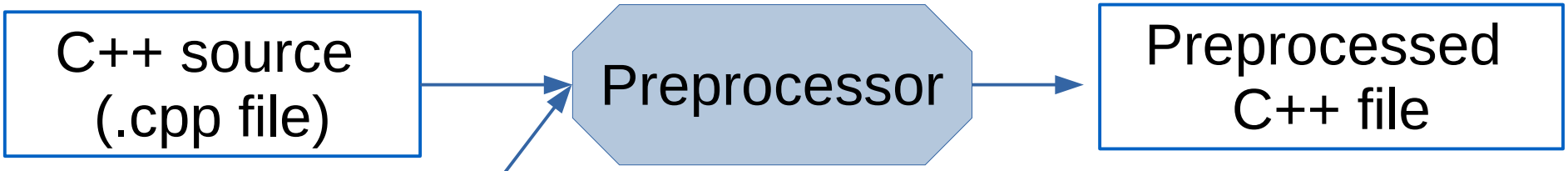
all the lines with # are processed

```
#include<iostream>
```

all the code from iostream file is copied into the C++ source file

Python and C++

- C++ translation process:



all the lines with # are processed

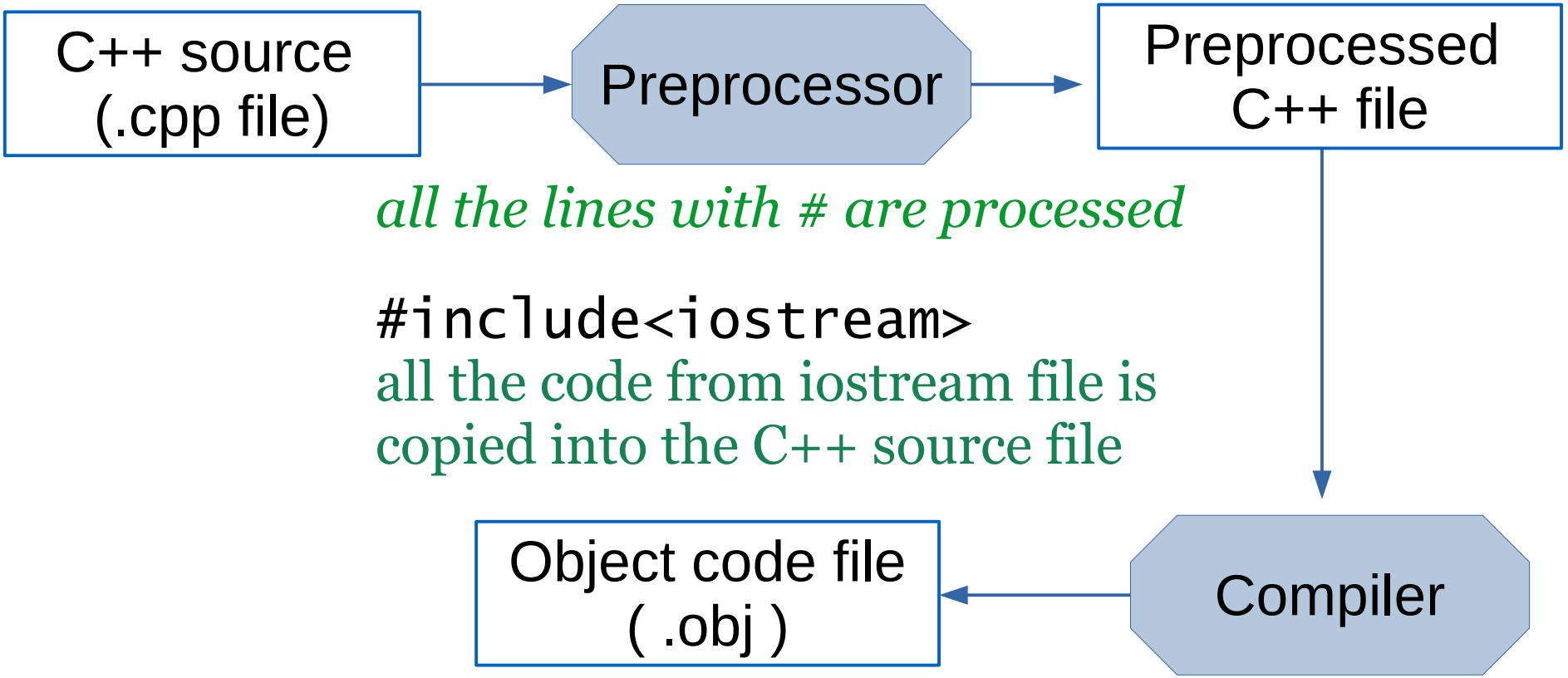
Header file
(.h file)

`include "funcs.h"`

all the code from `funcs.h` *file is copied into the C++ source file*

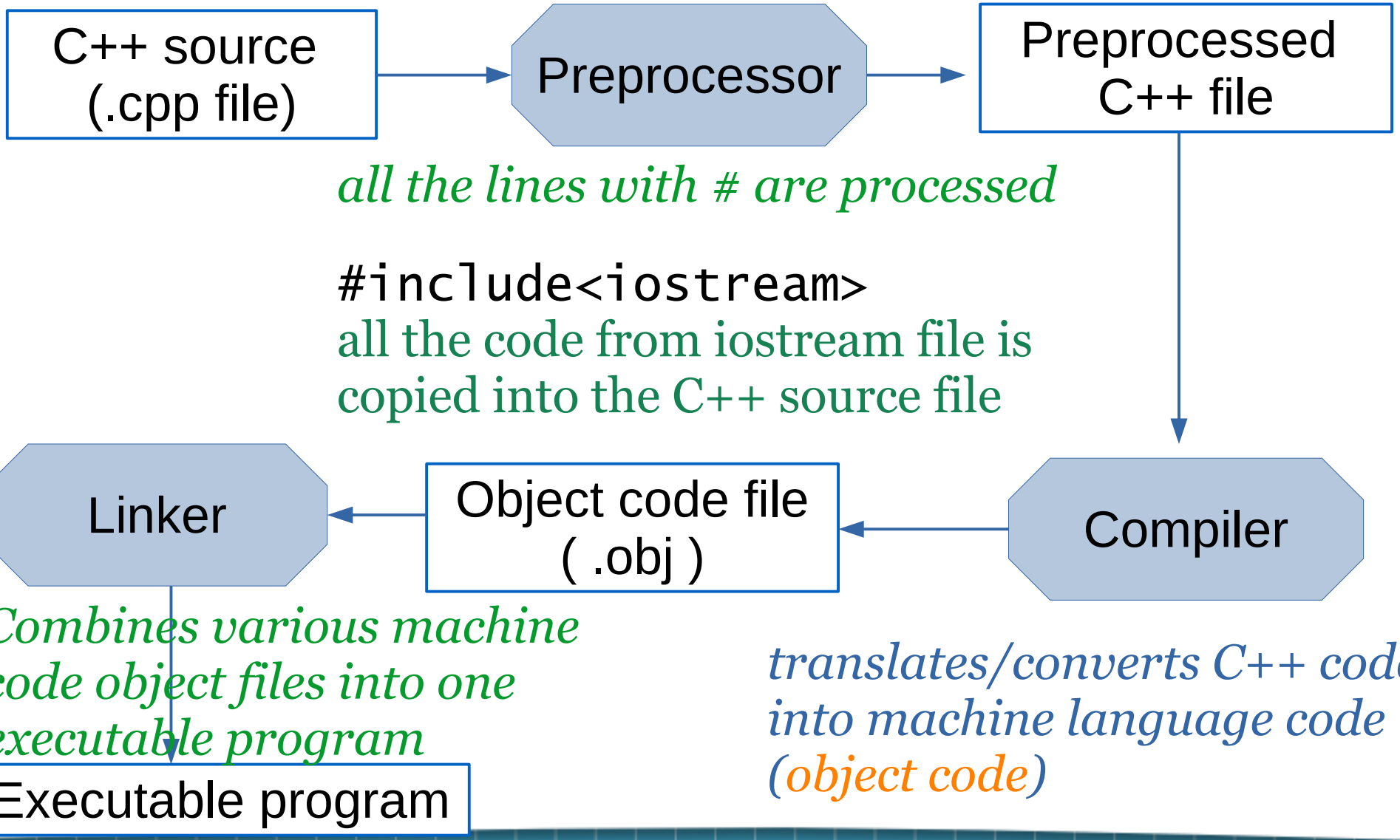
Python and C++

- C++ translation process:



Python and C++

- C++ translation process:



Python and C++

- C++ translation process:

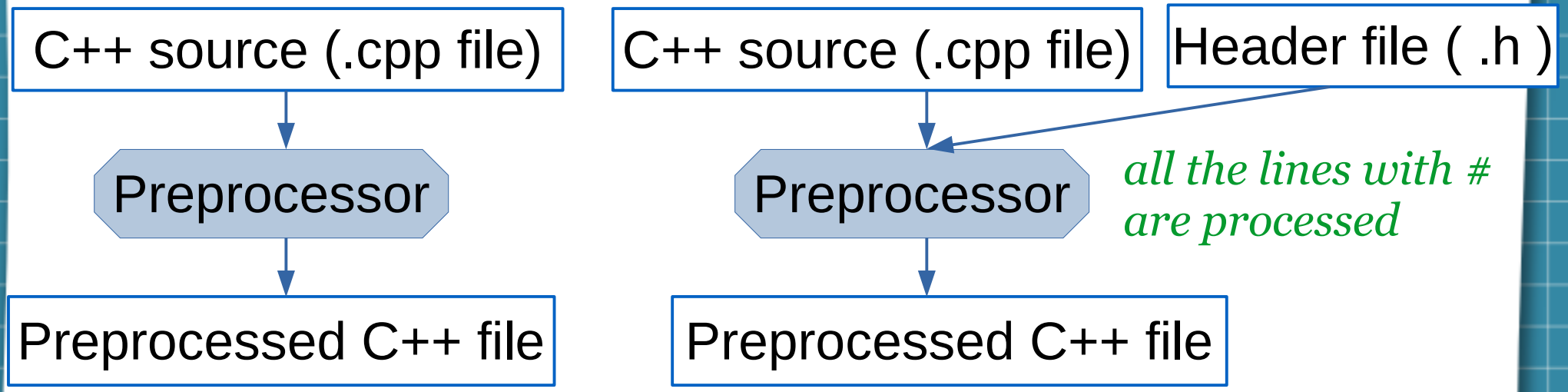
C++ source (.cpp file)

C++ source (.cpp file)

Header file (.h)

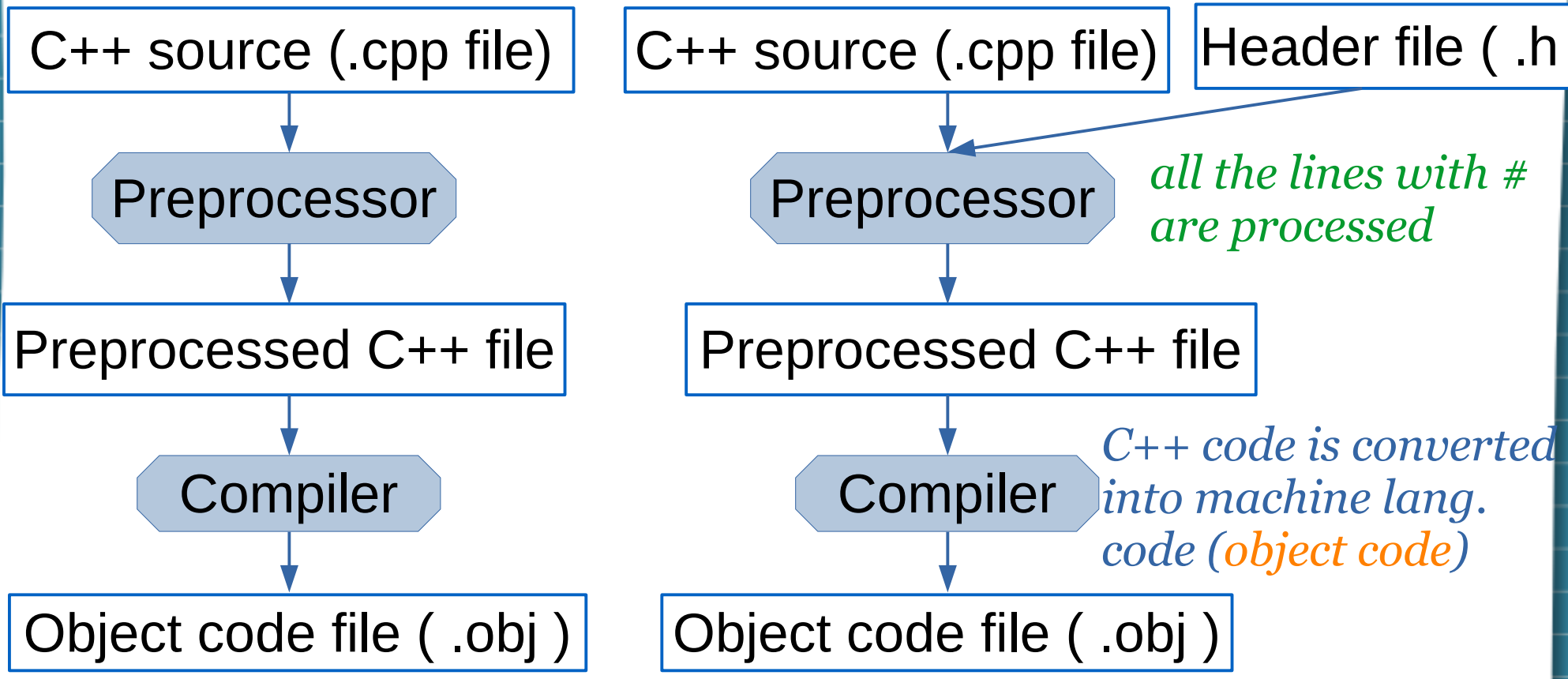
Python and C++

- C++ translation process:



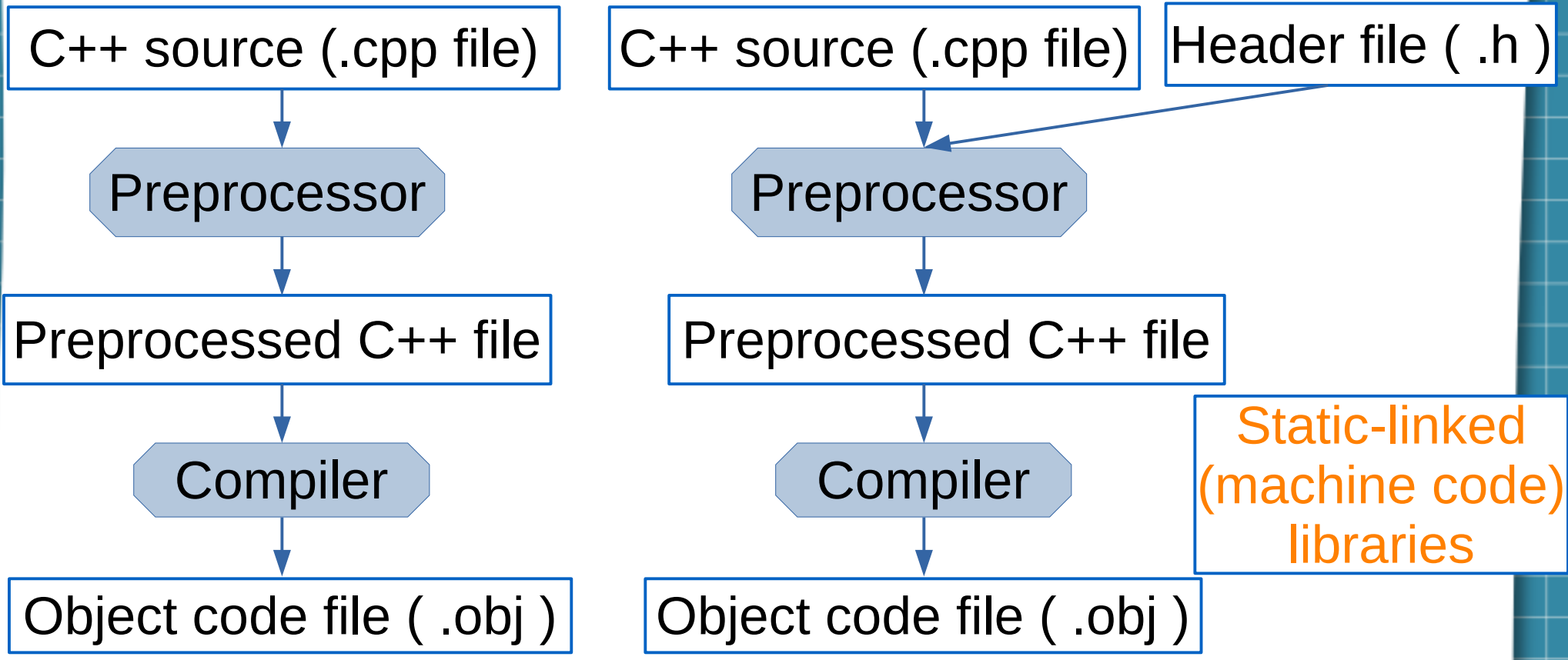
Python and C++

- C++ translation process:



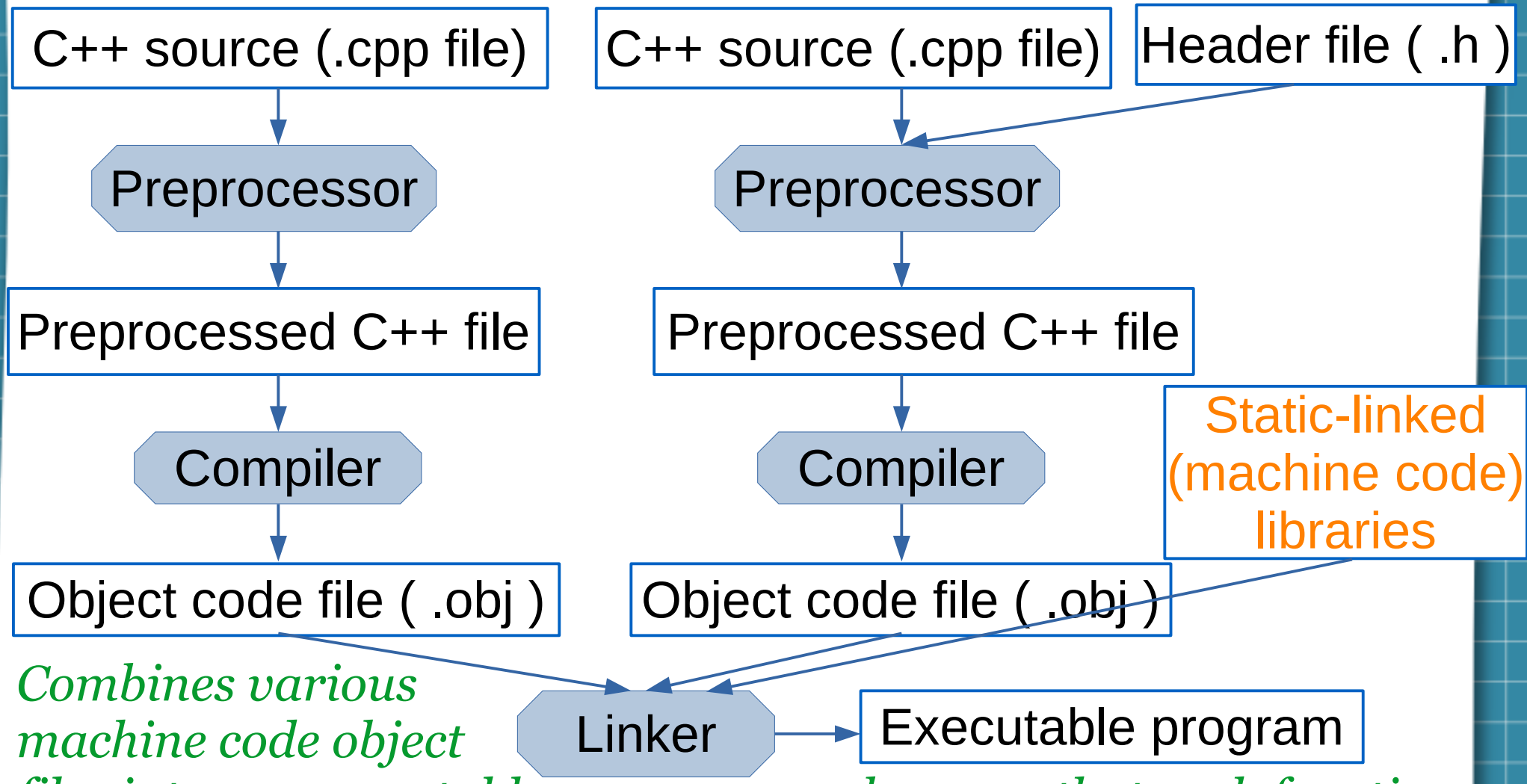
Python and C++

- C++ translation process:



Python and C++

- C++ translation process:



Combines various machine code object files into one executable program; makes sure that each function called exists in exactly one of the object files

Python and C++

- Python code executes slower than compiled C++ code

Python and C++

- Python code executes slower than compiled C++ code
- C and C++ are lower level languages than Python

Python and C++

- Python code executes slower than compiled C++ code
- C and C++ are lower level languages than Python
- C doesn't provide built-in `list` or `dictionary` types

Python and C++

- Python code executes slower than compiled C++ code
- C and C++ are lower level languages than Python
- C doesn't provide built-in `list` or `dictionary` types
- C++ doesn't have any equivalent to Python `list`

Python and C++

- Python code executes slower than compiled C++ code
- C and C++ are lower level languages than Python
- C doesn't provide built-in `list` or `dictionary` types
- C++ doesn't have any equivalent to Python `list`
- There is no single programming language that is a best choice for every problem, since different languages provide different capabilities

C++: Input and Output

```
#include <iostream>

int main() {

std::cout << "Hello, world!";

int a, b, c;
std::cin >> a >> b >> c;

float x{ 1.2 }, z{ 2.9 };

std::cout << a + b + c + x + z;

return 0;

}
```

C++: Input and Output

```
#include <iostream>

int main() {

std::cout << "Hello, world!";

int a, b, c;
std::cin >> a >> b >> c;

float x{ 1.2 }, z{ 2.9 };

std::cout << a + b + c + x + z;

return 0;

}
```

preprocessing directive to include the input/output stream header `iostream`

`iostream` defines class templates for input: `basic_istream`, and output, `basic_outstream`, and other objects

C++: Input and Output

```
#include <iostream>

int main() {

std::cout << "Hello, world!";

int a, b, c;
std::cin >> a >> b >> c;

float x{ 1.2 }, z{ 2.9 };

std::cout << a + b + c + x + z;

return 0;

}
```

defining the `main` function, which can return an *integer value*

C++: Input and Output

```
#include <iostream>

int main() {

std::cout << "Hello, world!";

int a, b, c;
std::cin >> a >> b >> c;

float x{ 1.2 }, z{ 2.9 };

std::cout << a + b + c + x + z;

return 0;

}
```

(abbreviation)
standard namespace, or
C++ standard library
that has definitions of
many useful objects.

C++: Input and Output

```
#include <iostream>

int main() {

std::cout << "Hello, world!";

int a, b, c;
std::cin >> a >> b >> c;

float x{ 1.2 }, z{ 2.9 };

std::cout << a + b + c + x + z;

return 0;

}
```

output operator

output object

input operator(s)

input object

C++: Input and Output

```
#include <iostream>

int main() {

std::cout << "Hello, world!";

int a, b, c;
std::cin >> a >> b >> c;

float x{ 1.2 }, z{ 2.9 };

std::cout << a + b + c + x + z;

return 0;

}
```

declare three variables of integer type

declare and *initialize* two variables of float type (decimals)
"list initialization", introduced in C++ 11

C++ latest versions

- C++ 11 was published by ISO/IEC in 2011
- C++ 14 was published by ISO/IEC in 2014
- C++ 17 was published by ISO/IEC in 2017
  - WikiPedia
- C++ 20 is yet to be published
  - https://en.wikipedia.org/wiki/C%2B%2B20
  - https://en.cppreference.com/w/cpp/20

C++ operators and order of precedence
Arithmetic

Operation	Algebraic Expression	Arithmetic operator (C++)	C++ expression
addition	x + 18	+	x + 18
subtraction	5 - y	-	5 - y
multiplication	ax	*	a*x
division	x ÷ y	/	x / y
remainder	y mod 7	%	y % 7
power	x^4	no operator, use function pow()	pow(x,4)

Comments:

1) Be careful with *division* operator:

`5/2 = 2,` whereas `5.0 / 2 = 5 / 2.0 = 2.5`

2) *power* function: you will need to include the library <`math`>

More info: http://www.cplusplus.com/reference/cmath/pow/

C++ operators and order of precedence
Arithmetic

Operation	Algebraic Expression	Arithmetic operator (C++)	C++ expression
addition	x + 18	+	x + 18
subtraction	5 - y	-	5 - y
multiplication	ax	*	a*x
division	x ÷ y	/	x / y
remainder	y mod 7	%	y % 7
power	x^4	no operator, use function pow()	pow(x,4)

Precedence rules: same as in mathematics

1) parentheses are evaluated first,
nested parentheses: innermost first, etc.

2) multiplication, **division**, **remainder** next (*left to right*)

3) addition, **subtraction** last (left to right)

C++ operators and order of precedence
Arithmetic

Few examples of writing algebraic expressions in C++:

Algebraic Expression	C++ expression

$\frac{\sqrt{b^2-4ac}}{2a}$	`#include <math>`
	`sqrt(b*b – 4*a*c) / (2*a)  or`
	`sqrt(pow(b,2) – 4*a*c) / (2*a)`

$\frac{5}{a-b}$	`5 / (a - b)`
$\frac{5}{a}-b$	`5 / a - b`

C++ operators and order of precedence
Equality and Relational Operators

relational / equality	algebraic expression	C++ expression	
greater than	x > 18	x > 18	relation operators
less than	y < 5	y < 5	
greater than or equal	x ≥ 18	x >= 18	
less than or equal	y ≤ 5	y <= 5	
check for equality	(is) x = 8 (?)	x == 8	equality operators
check for “not equal”	(is) x ≠ 8 (?)	x != 8	

Precedence rules: equality operators have the same level of precedence, it is lower than that of the relational operators, and they associate left to right

C++ operators and order of precedence

Operator precedence:

operators	associativity
()	innermost ones; otherwise not specified in the standard
*, /, %	left to right
+ , -	left to right
<<, >>	left to right
<, <=, >=, >	left to right
==, !=	left to right
=	right to left

C++ operators and order of precedence

Let's dive into three examples:

Example1.cpp: it is very similar to the code we saw before.
We are going to:

- use directive `using namespace std;` and discuss how it affects the rest of the code
- use directives

```
using std::cout;
using std::cin;
using std::endl;
```

and discuss the differences between all three versions

Example2.cpp: code with some added relations and logical operators

Example3-operators.cpp: solving quadratic equation; demonstrates use of integers, decimals, characters, while loop, and comparison operators.