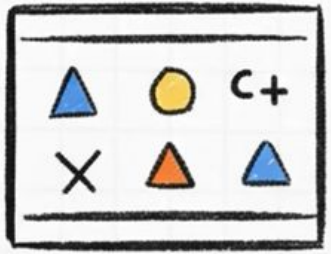
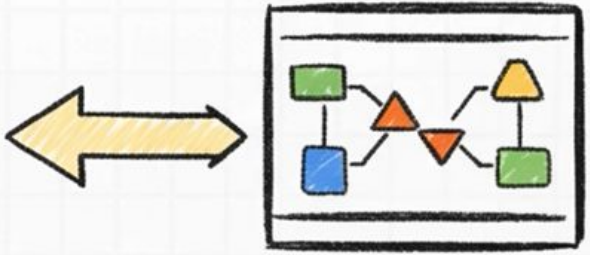
Week 7

Controlling your Code

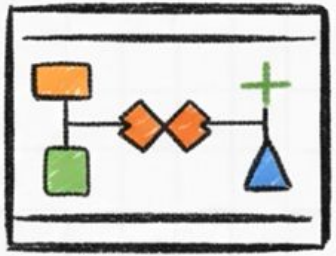


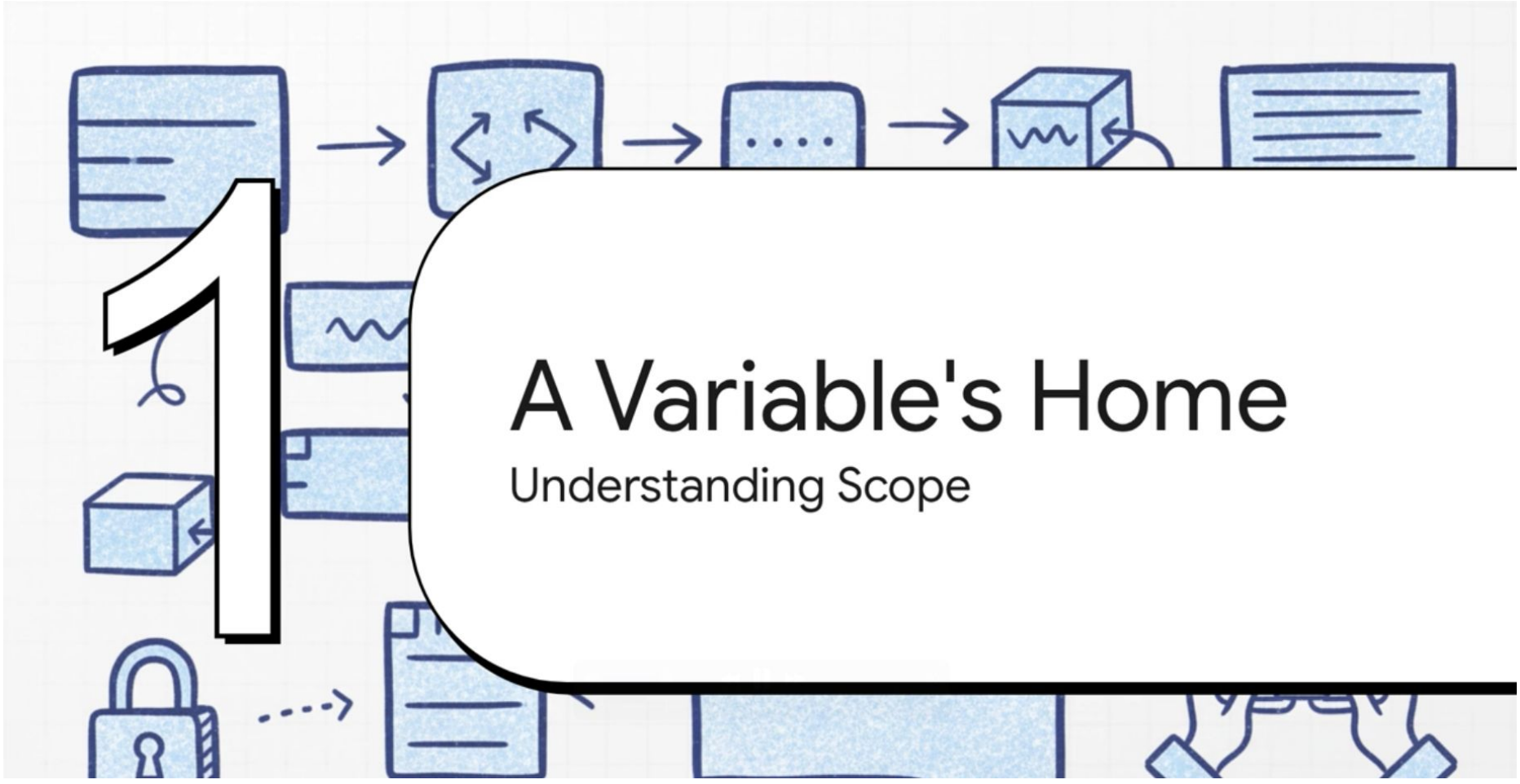


Call by Value

Your functions work. But how do they really talk to each other?

Call by Reference→

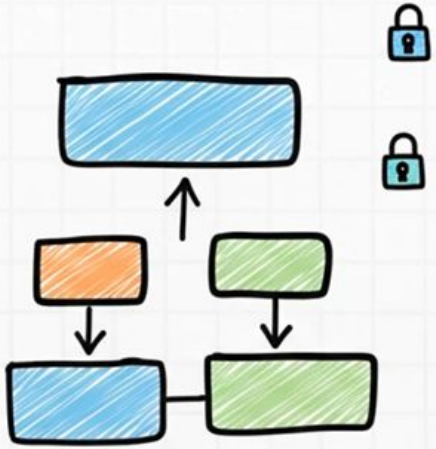


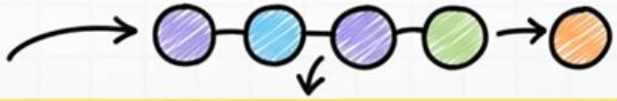


1

A Variable's Home

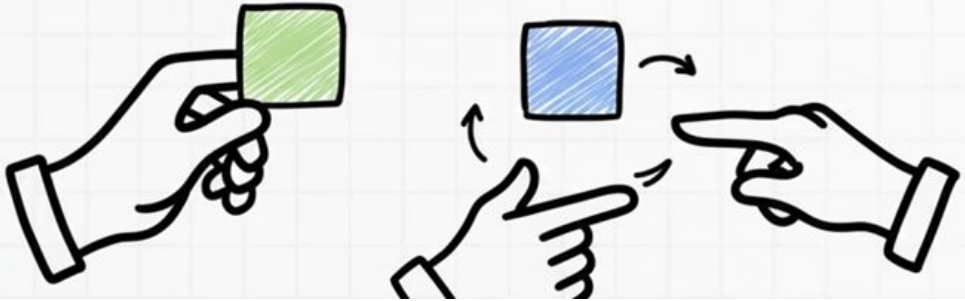
Understanding Scope



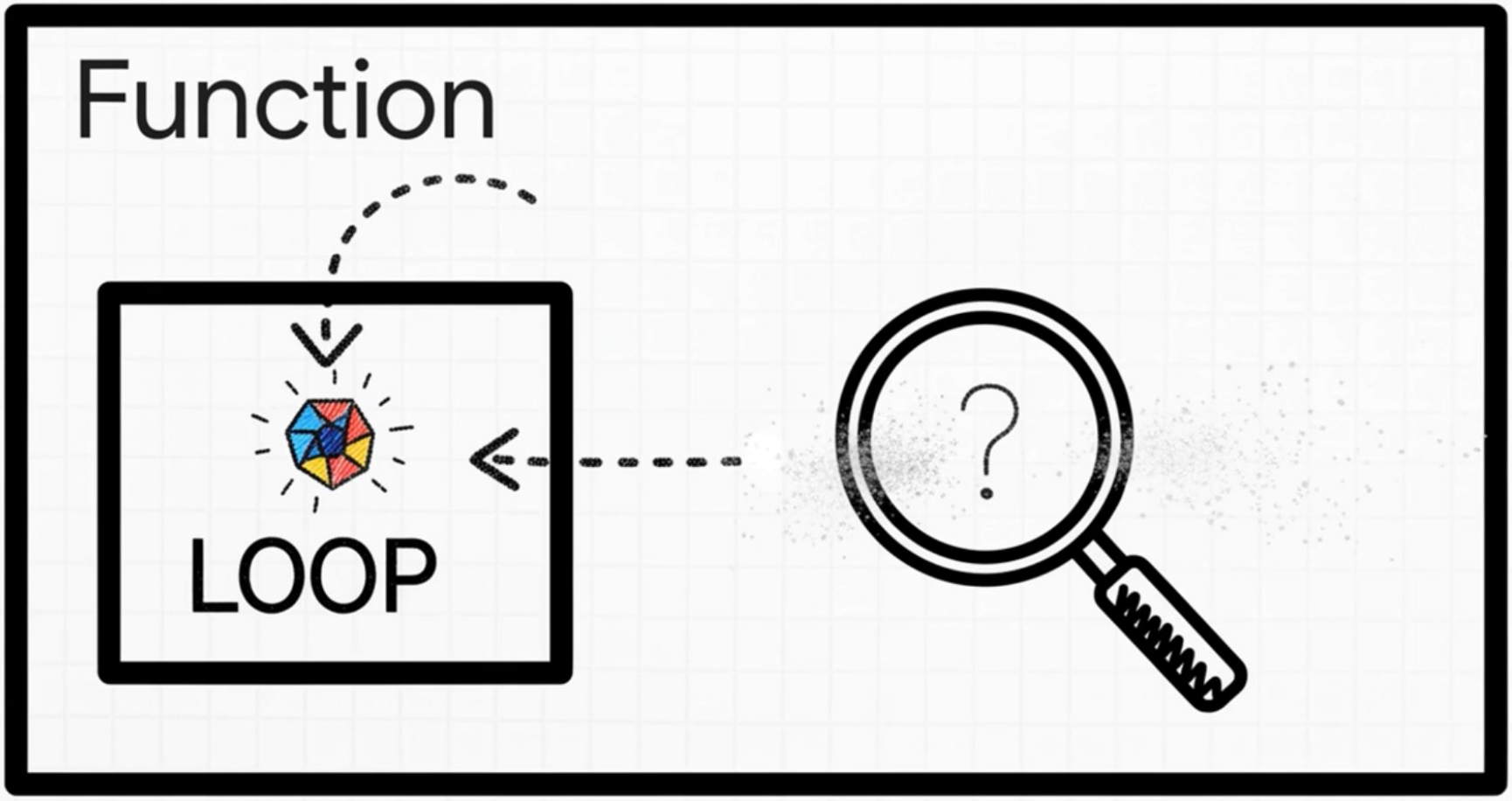


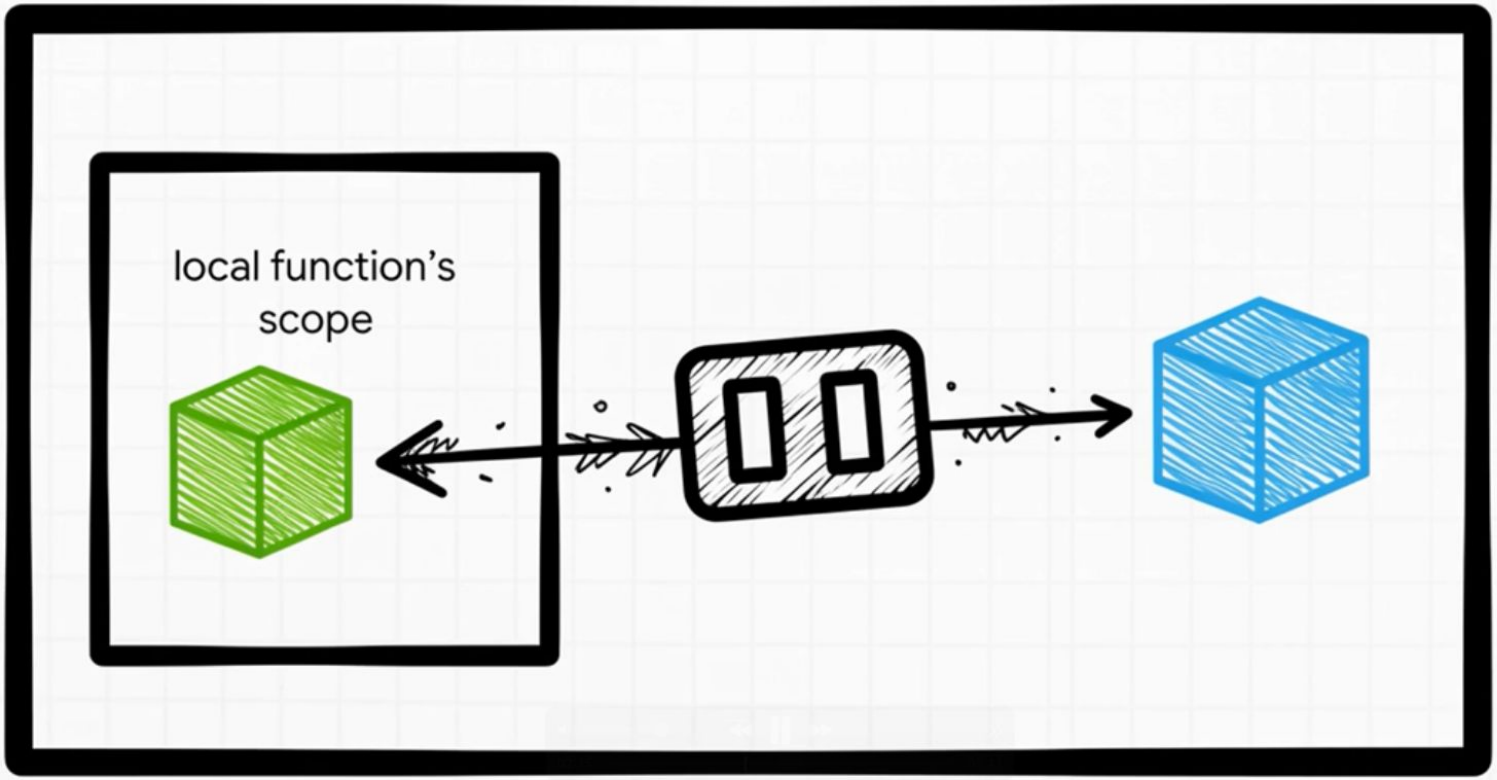
Scope

The region of code where a variable can be legally accessed, defining its 'lifespan' or 'visibility'.



	Local Variables	Global Variables
Declaration	Inside a function or block { }	Outside of all functions
Accessibility	Only within that function/block	Throughout the entire program file
Precedence	Takes priority	Lower priority


Function
LOOP
?


local function's
scope

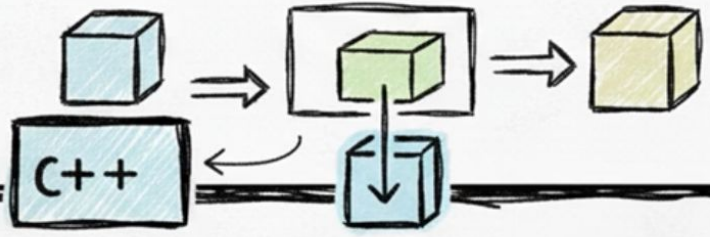
Scope Type	Description	Key Syntax
Local Variables	Variables declared within a block statement or inside a function are local to that block or function [2]. Local variables take precedence when variables share the same name [2].	Declared within `{}` or inside a function definition.
Global Variables	Variables declared outside all functions (ideally at the top) that can be accessed throughout the program or file [2] [3].	Declared at the top of the file, outside of `main` or other functions [2].
Accessing Global	When a local variable has the same name as a global variable, the local variable takes precedence. To access the global variable in the local scope, you must use the **scope resolution operator (`::`)** [2].	`::VariableName` [3]

Accessing a Global Variable

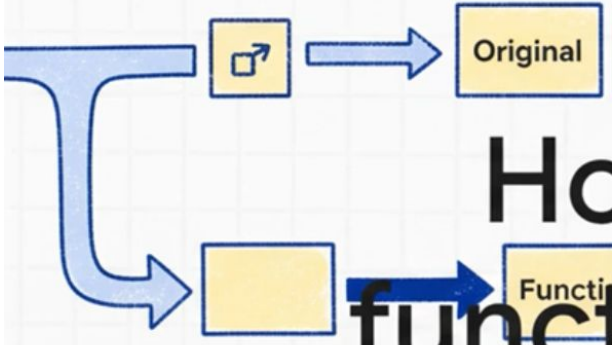
```
// global variable
double PI = 22 / 7.0; // [3]

int main() {
    double PI = 3.14;
    std::cout << ::PI; // accessing the global variable PI [3]
}
```

Passing Parameters

Copies vs. Originals



CALL BY VALUE



C++

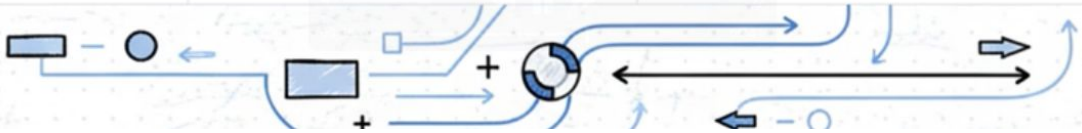
How do you let a function change your original variable?

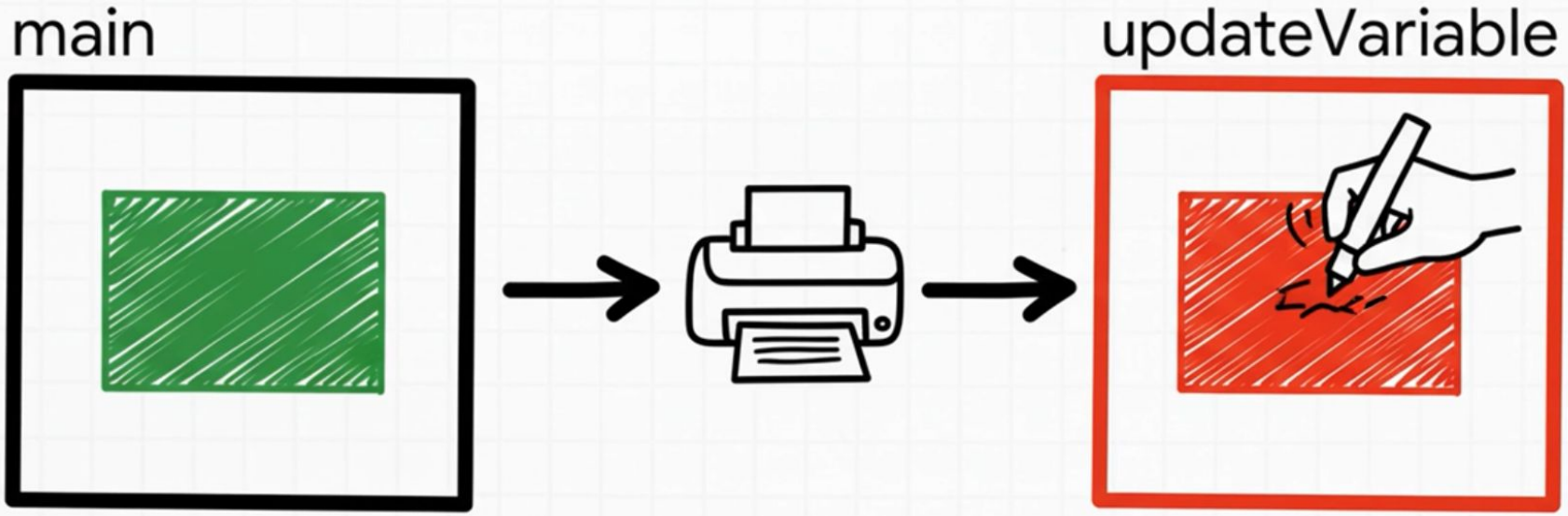
CALL BY REFERENCE

{ }

C++

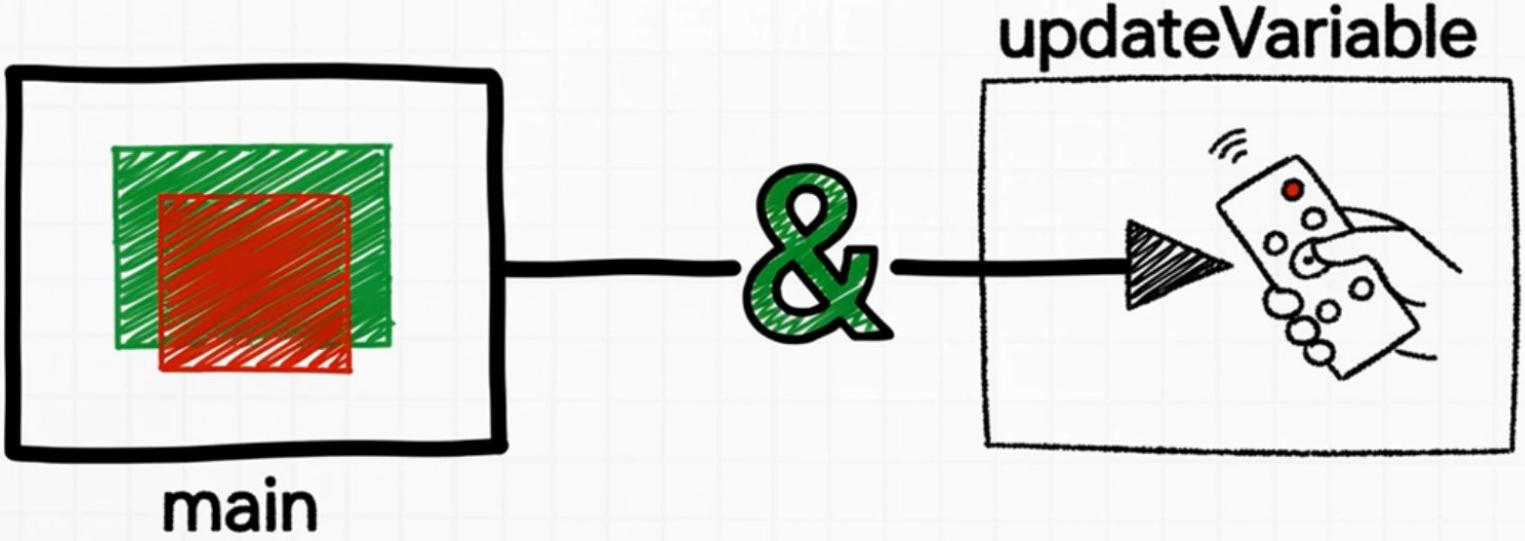
	Call by Value	Call by Reference
Mechanism	Function receives a copy	Function receives the address
Effect	Changes do not affect original	Changes directly modify original
Analogy	Sending a photocopy	Sharing the original document




main
updateVariable

Call by Value

```
void updateVariable(int name); // This is call by value (no &)
int main() {
    int x = 5, y = 6;
    // In call by value, a copy of 'y' (value 6) is passed to the fun
    updateVariable(y);
    // When printed here, 'y' remains 6, because the function modifie
    cout << y << endl;
}
void updateVariable(int name) {
    // 'name' is a local copy of the original argument.
    // Changing 'name' here:
    name = 3;
    // ...only changes the local copy, not the original variable 'y'.
}
```


updateVariable
&
main

Call by Reference

```
void updateVariable(int &);
int main() {
    int x = 5, y = 6;
    cout << &y << endl; // Prints the address of y
    updateVariable(y);
    cout << y << endl;  // Prints the new value of y
}
void updateVariable(int &name) {
    cout << &name << endl; // Prints the address stored in 'name' (wh
    name = 3;              // Modifies the original variable y
}
```

Function Header Syntax (Call by Reference):

```
void updateVariable(int &); // [4]
```

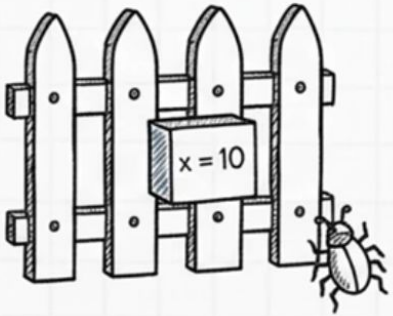
Function Definition Syntax (Call by Reference):

```
void updateVariable(int &name) {
    // code that modifies the original variable
    name = 3;
} // [4]
```


C++
3
Why This Matters
Smarter, Safer Code
ORIGINAL

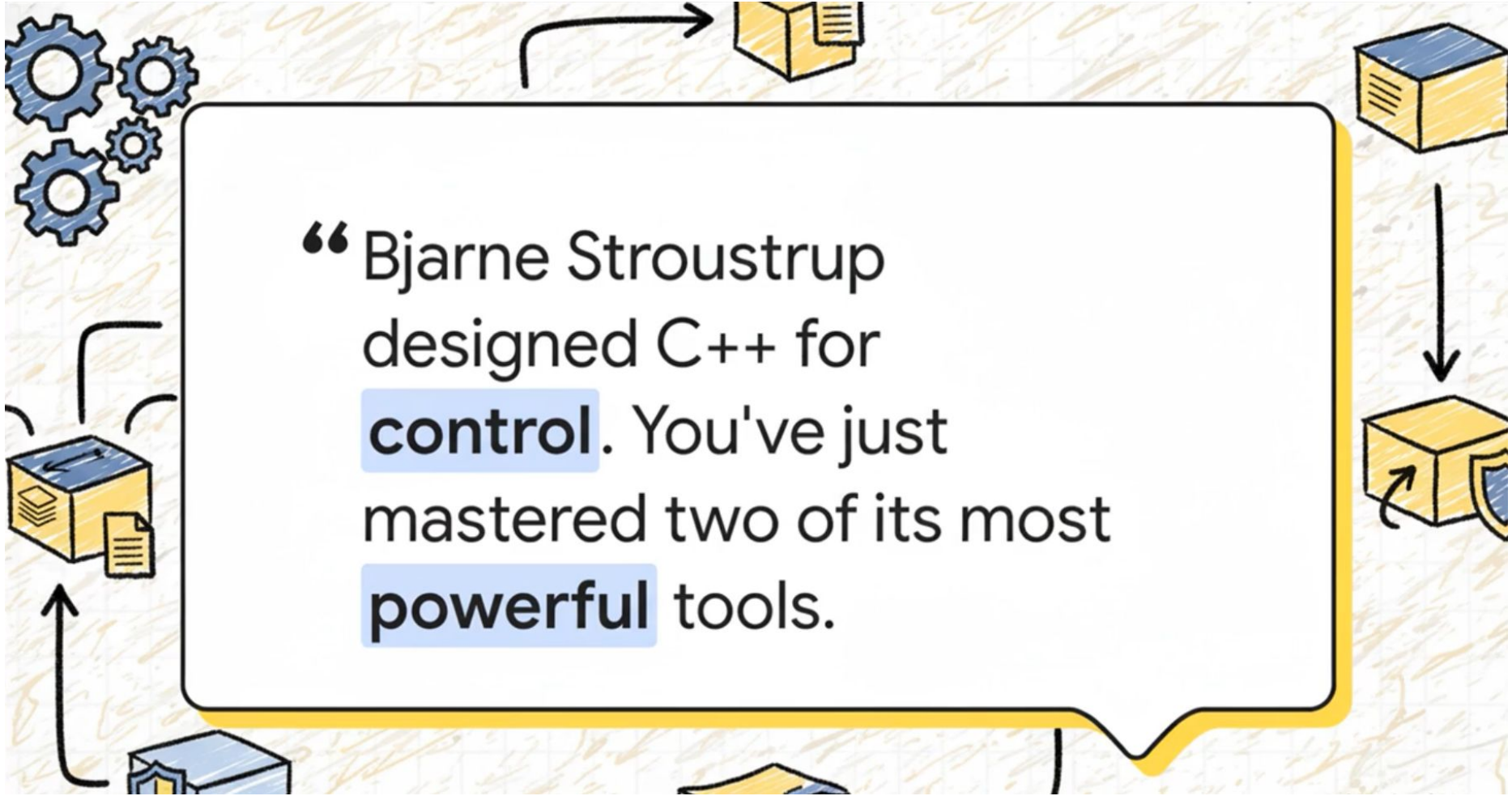
Your C++ Toolkit

Call by Value



Scope

Call by Reference


“Bjarne Stroustrup designed C++ for control. You've just mastered two of its most powerful tools.

Attendance