# From C to C++

- Version 1: Dr. Ofir Pele
- Version 2: Dr. Miri Ben-Nissan
- Version 3: Dr. Erel Segal-Halevi

## Why C++ is much more fun than C (C++ FAQ)?

1. Classes & methods - OO design
2. Generic programming - Templates allow for code reuse
3. Stricter type system (e.g. function args)
4. Some run-time checks & memory control

**A common and mature language that gives you high level and low level control**

Have fun ♪

## Why C++ is much more fun than C (C++ FQA)?

1. Tons of corner cases
2. Duplicate features
3. Cryptic syntax
4. Undecidable syntax (uncompilable progarms!)
5. No two compilers agree on it

**Probably one of the hardest computer languages to master.**

Have fun ♪

## C++ vs. Java

**Java is much simpler to program -**
removes many ambiguous and duplicate featues.

*So why use C++ at all?*

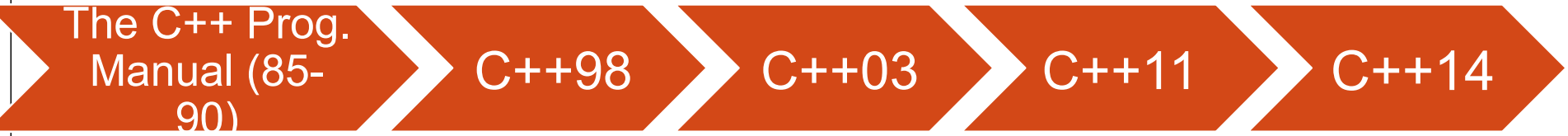
1. Tight memory management – important in embedded systems.
2. Tight time management – important in real-time systems.
3. Creating high-performance libraries that can be linked from other languages, e.g, Python.

# History

Default in g++

auto, lambdas, threading, …

Auto return type, generic lambdas, binary literals,…



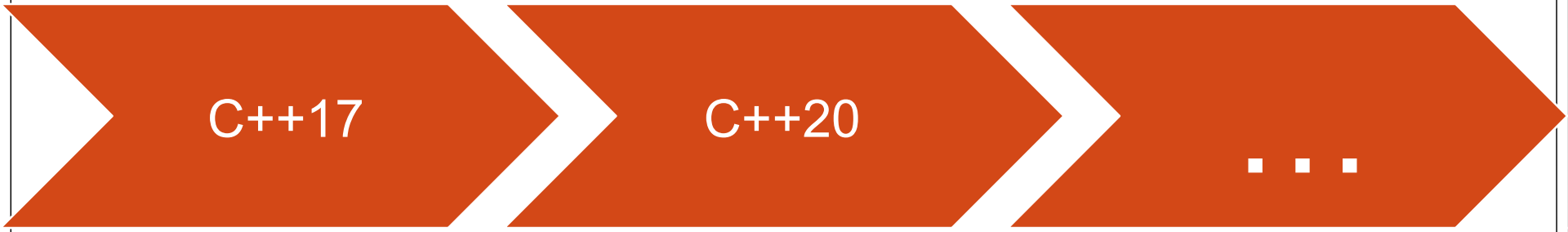

Minor changes

Major changes

Minor changes

**We'll learn parts of C++-11, 14, 17,**
**Mostly parts that makes C++ more "pythonic" while keeping it efficient**

# Future

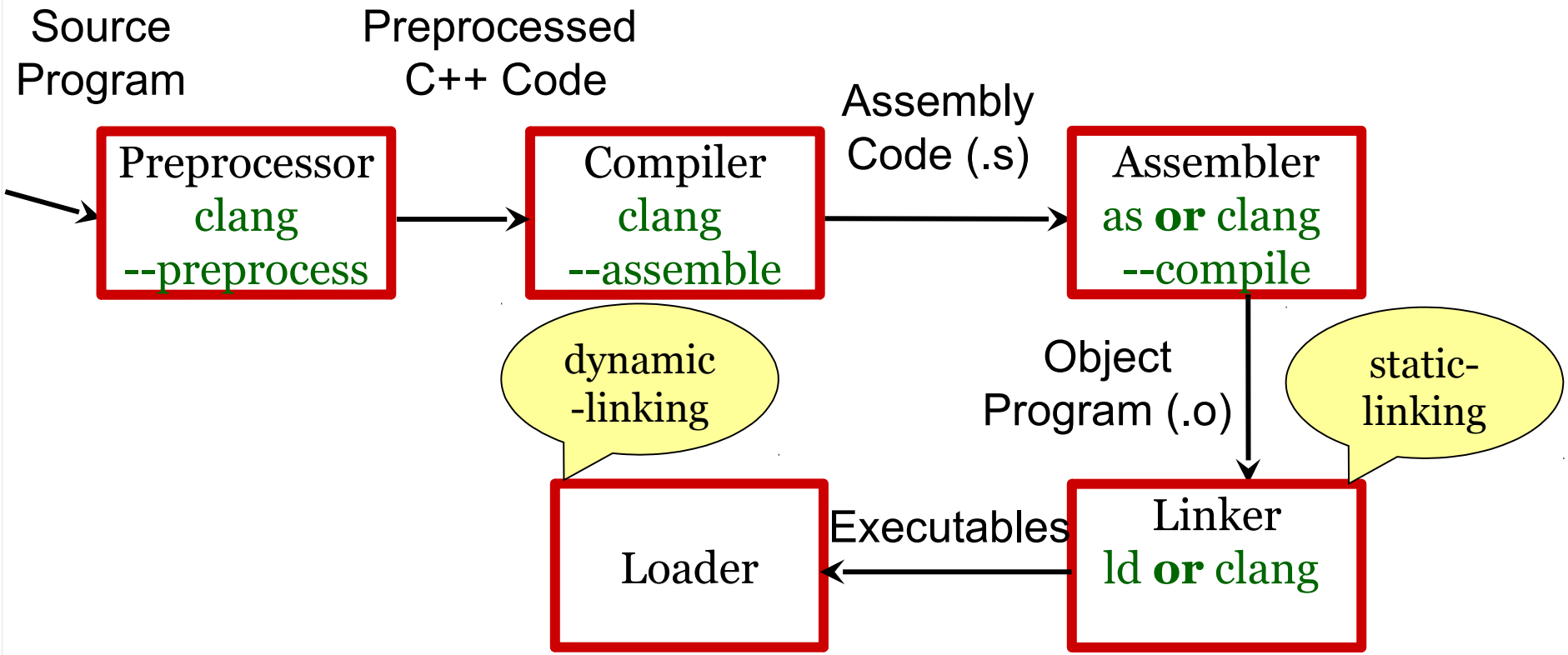


# Basic Unix commands

ls
ls -latr
cd [dirname]
mkdir [dirname]
cat [filename]
nano [filename]
grep [word]
source [filename]
rm [filename]

grep … < input.txt
grep ... > output.txt
ls -latr | grep bash

su
exit
sudo apt install ...


Source Program
Preprocessor
clang
--preprocess
Preprocessed C++ Code
Compiler
clang
--assemble
Assembly Code (.s)
Assembler
as or clang
--compile
Object Program (.o)
Linker
ld or clang
Executables
Loader
dynamic -linking
static- linking

# The missing types

Fill in missing types from C,
in somewhat crude way

# strings in C++

```
#include <iostream>
#include <string>
int main()
{
  std::string str;
  int a;
  double b;
  std::cin >> str >> a >> b;
  if(std::cin.fail())
  {
    std::cerr << "input problem \n";
    return 1;
  }
  std::cout << "I got: " << str << ' '
  << a << ' ' << b << std::endl;
}
```

:More about string functions

http://www.cppreference.com/cppstring

# Boolean variables

```
#include <iostream>
int main()
{
  int a = 5;
  bool isZero = (a == 0);
  // same conditions
  if(!isZero && isZero== false &&
  isZero != true && !!!isZero && a )
  {
    std::cout << "a is not zero\n";
  }
}
```

Good style

# C++-11 enum class

```
enum class Season : char {
   WINTER, //= 0 by default
   SPRING, //= WINTER + 1
   SUMMER,  //= WINTER + 2
   AUTUMN //= WINTER + 3
};

Season curr_season;
curr_season= Season::AUTUMN;
curr_season= SUMMER; //won't compile! (good)
curr_season= 19; //won't compile! (good)
int prev_season= Season::SUMMER; //won't compile!
(good)
```

# Overloading

**Understand and remember.**

- More than syntactic sugar.
- This is how a lot of stuff works under the hood (e.g. inheritance)

# Function overloading - C

```c
#include <stdio.h>
void foo()
{
  printf("foo()\n");
}
void foo(int n)
{
  printf("foo(%d)\n",n);
}
int main()
{
  foo(12);
  foo();
  return 0;
}
```

Compilation output:

**Error: Multiple definition of foo**

# Function overloading – C++

```
#include <iostream>
void foo() {
  std::cout << "foo()\n";
}
void foo(int n) {
  std::cout << "foo(" << n << ")\n";
}
int main() {
  foo(12);
  foo();
}
```

Output:

Compile, and print:

**foo(12)**
**foo()**

# Default parameters

```
#include <iostream>

void foo(int n=5)
{
  std::cout<< n;
}

int main()
{
    foo();
}
```

Output:

Compile, and print:
**foo(5)**

# Overload resolution

1. Find all functions with same name “candidates”. Let’s call them O1.
2. Find O2 subset of O1 which have the correct number of arguments - “viable candidates”
3. Find O3 subset of O2 with best matching arguments.

```
if |O3|=1
    use that function.
else (0 or more than 1):
    emit compilation error.
```