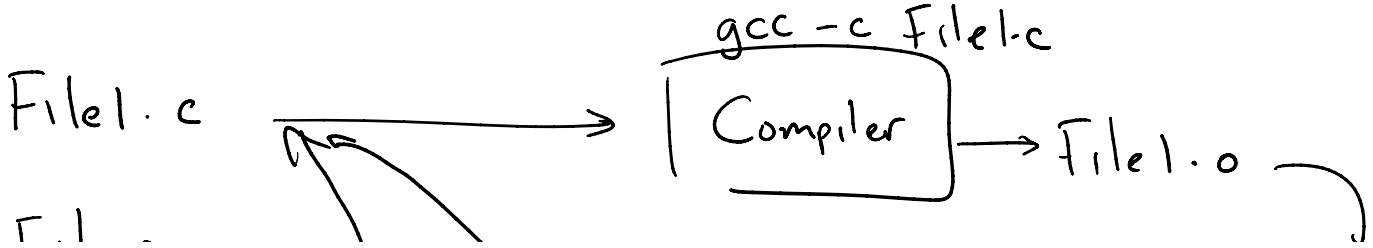
Lecture 11

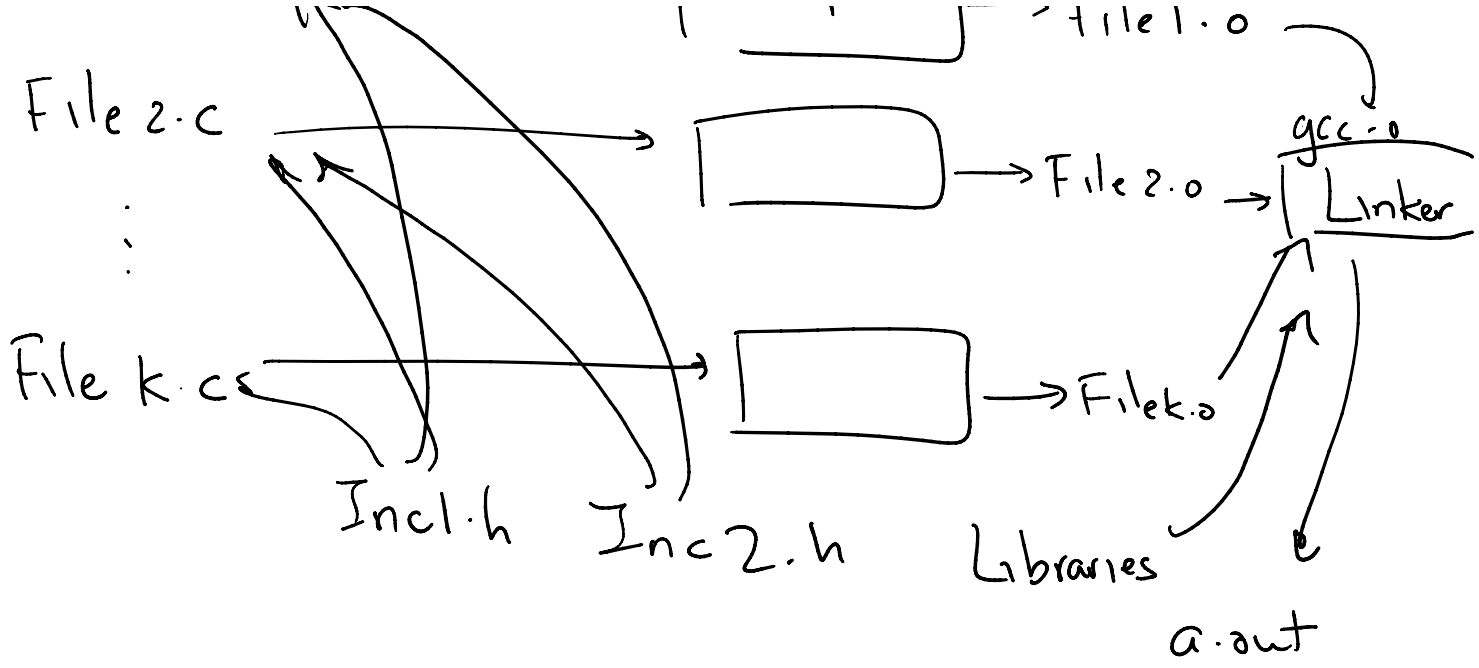
Tuesday, September 28, 2021 1:45 PM

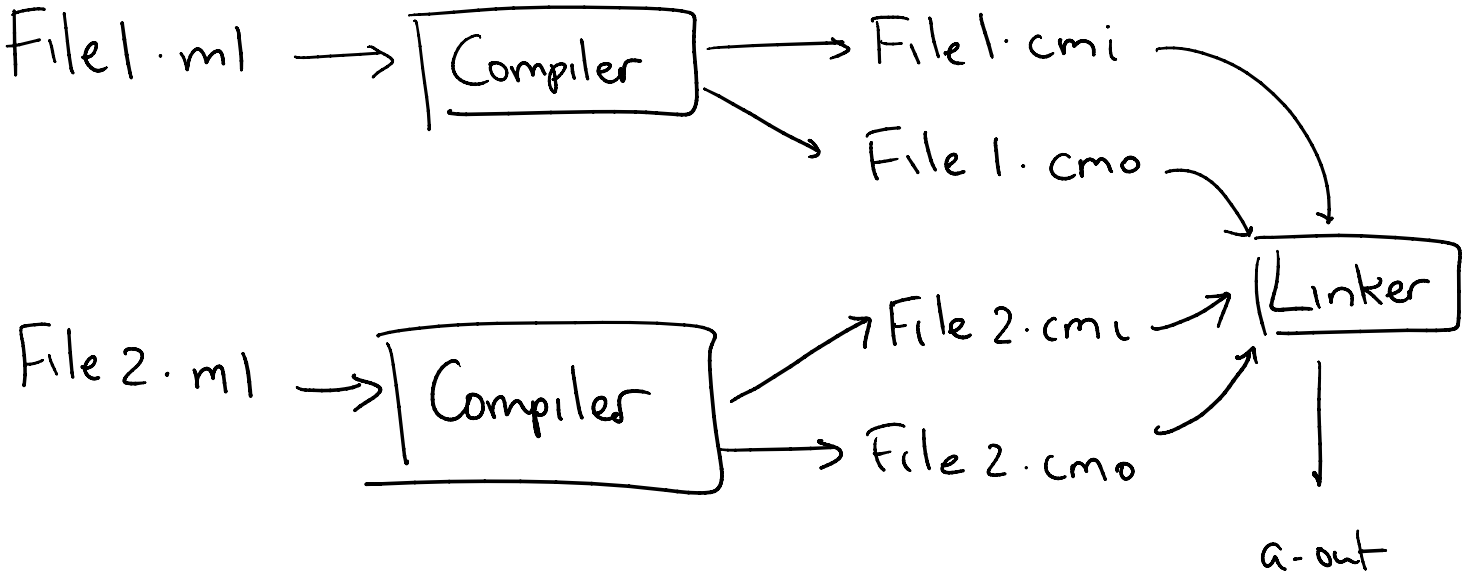
The Ocaml Module System

- Physical organization — `#use` `#load` Toplevel commands
- Namespaces — `open` `include`
- Abstraction / encapsulation
- Code reuse

Signatures	Structures



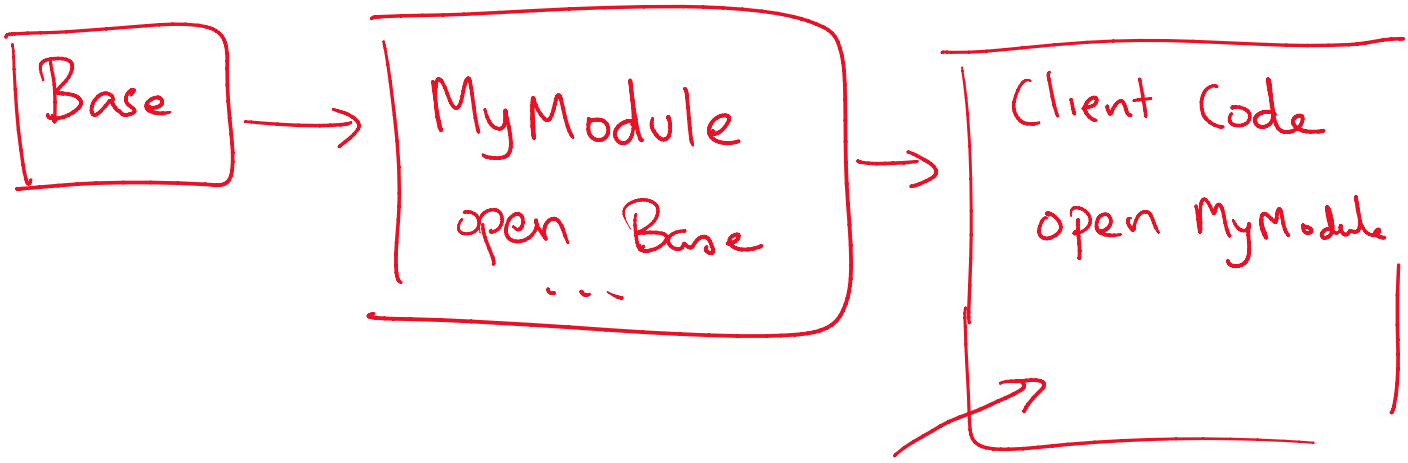




Open Base : "There is a module called Base.

Somewhere

Allow me to use the names defined in Base."



Cannot use names in Base without itself opening Base.

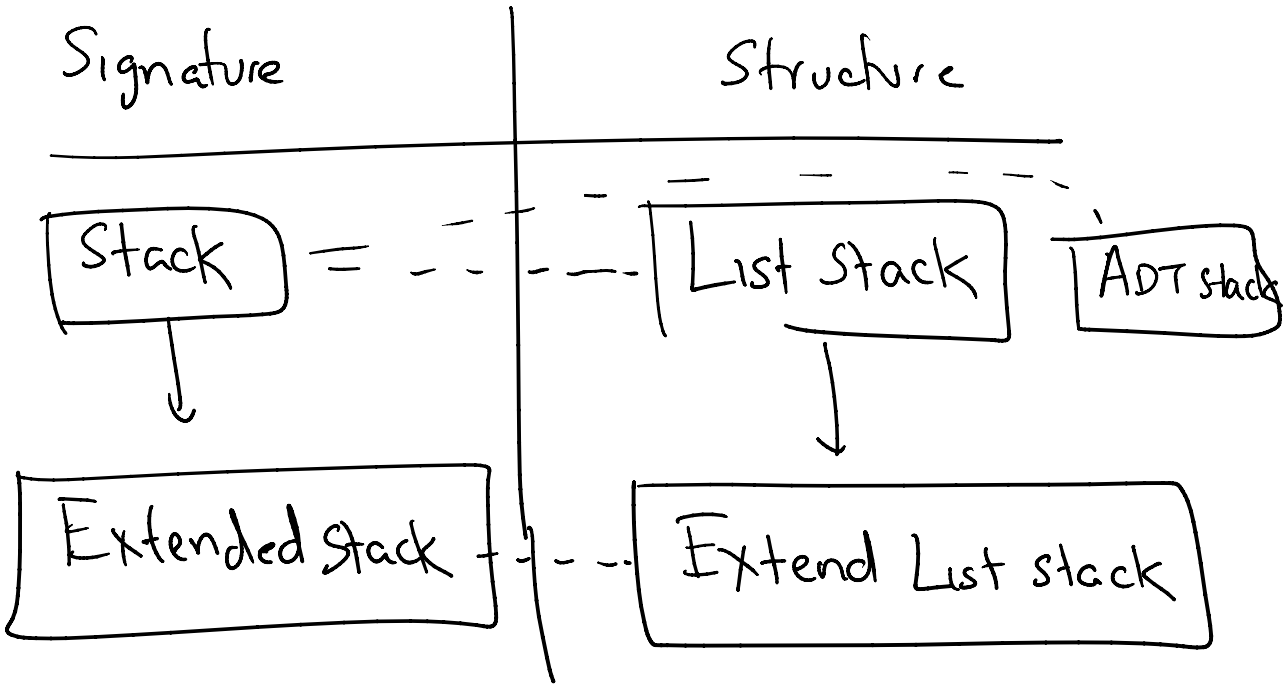
---

include Base

Take all names defined in Base

Bring them in

Re-export them.



Ocaml Base Language:

Values Types Functions

Structure Signature Functor

Module Language

Dynamic Dispatch

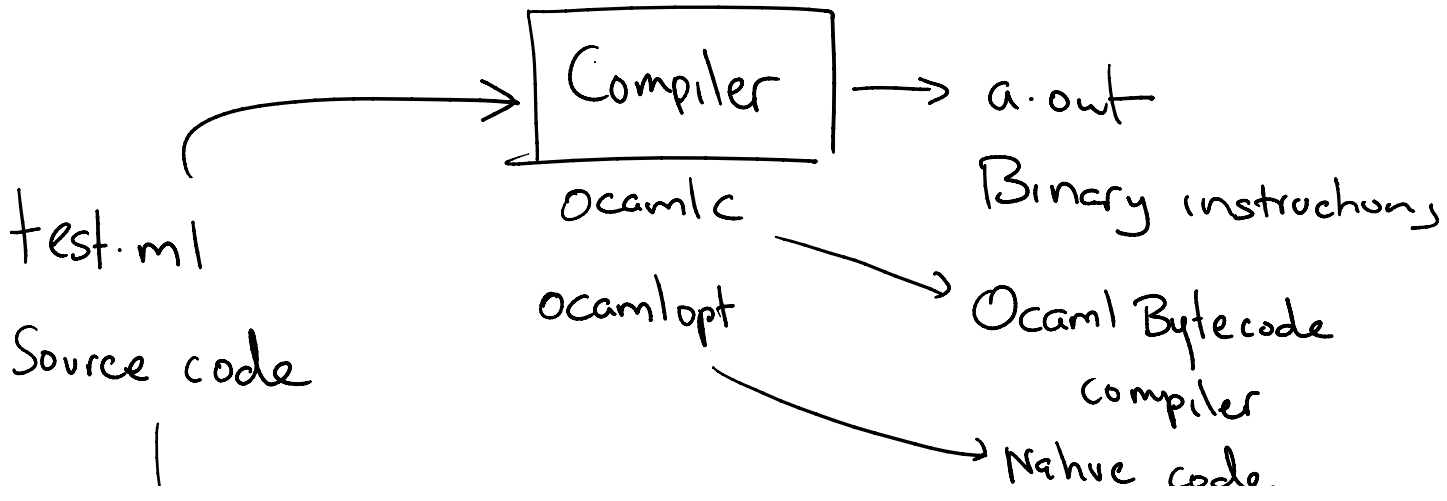
```
abstract class Greeter {
    void greet();
}
class HelloG extends Greeter {
    void greet() { println("Hello"); }
}
class HowdyG extends Greeter {
    void greet(){ println("Howdy!");
}

Void main() }
    Greeter g
    if (input == "formal")
```

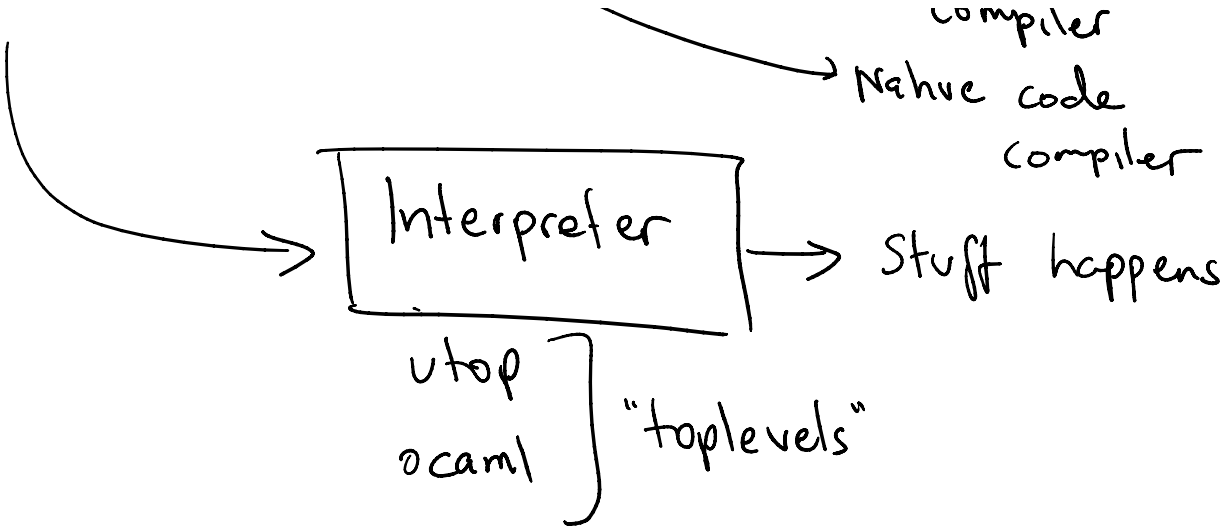
```
        greeter = new Hello ()
    else
        greeter = new Howdy ()
    g.greet()
}
```

Unit 2

Implementing a Language

Interpreter





"Hosted" implementations

Python — CPython — C

HotSpot JVM — C++

Javascript — Spider Monkey — Rust

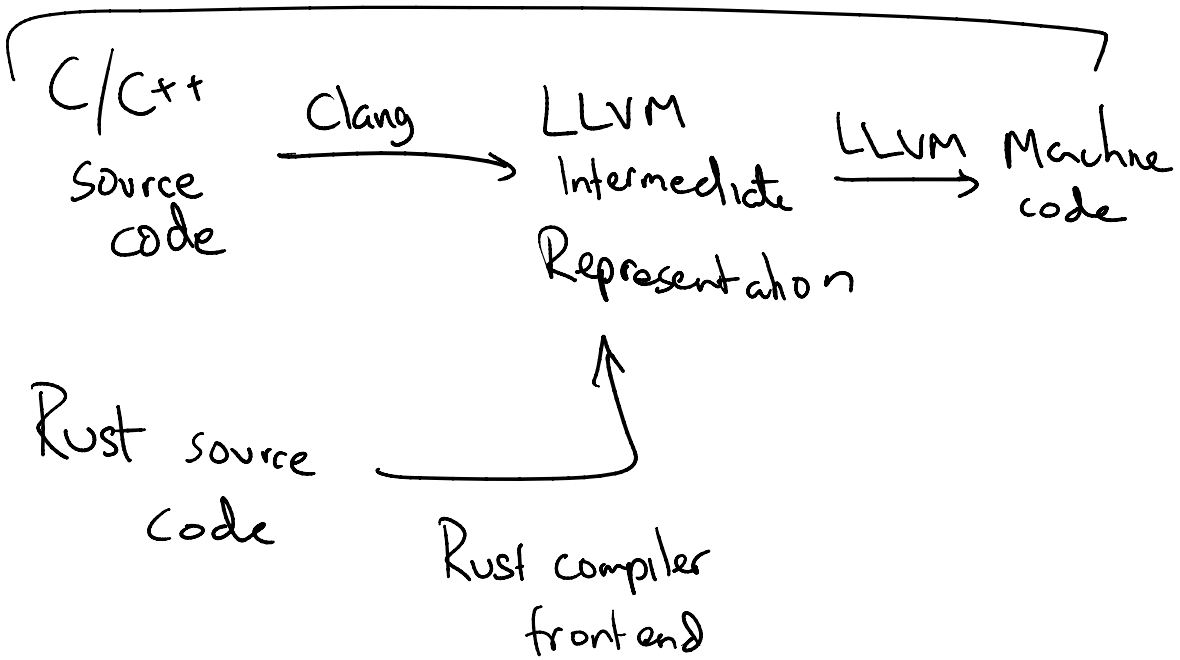
V8 — C++

"Bootstrapped" implementations

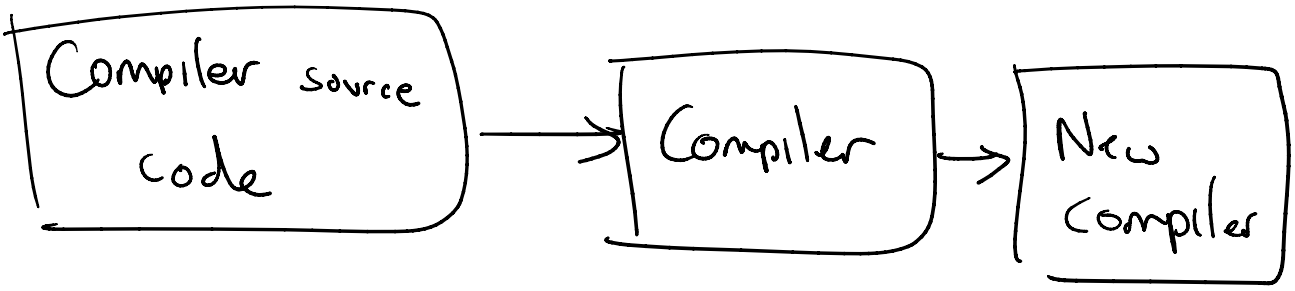
Compiler written in its own language

gcc ——— C++

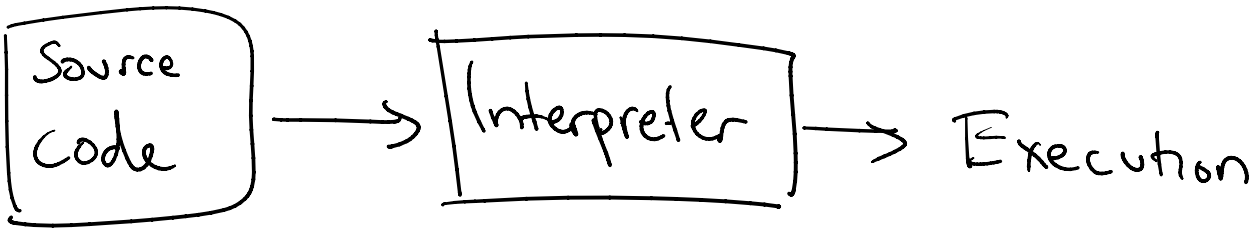
Clang/LLVM ——— C++



Ocamlopt ——— Ocaml



Bootstrapped Interpreter



Pypy Rikes JVM