Revisiting OCaml

Lecture 2

Formal Languages and Compilers 2011

Nataliia Bielova

How to run OCaml

- Run the interpreter with
  ```
  ocaml
  ```
- Save the file in “myfile.ml”, let the interpreter run it from file
  ```
  ocaml
  #use “myfile.ml”
  ```
- Compilation of a single module
  ```
  ocamlc –c myfile.ml
  ```
  Results in myfile.cmo
- Then use the compiled file in the interpeter:
  ```
  ocaml
  #load “myfile.cmo”;;
  open Myfile;;
  ```

Value binding and pattern matching

- let (x, y) = (“hi”,(1,2));;
- let (a, (b,c)) = (z, (3,4));;

Value binding and pattern matching

- let (x, y) = (“hi”,(1,2));;
- let (a, (b,c)) = (z, (3,4));;

- let h::t = [4;5;6];;
- let h::t = [4]::[5;6];;

Value binding and pattern matching

- let (x, y) = (“hi”,(1,2));;
- let (a, (b,c)) = (z, (3,4));;

- let h::t = [4;5;6];;
- let h::t = [4]::[5;6];;

- let x = 1 and y = 2 in x*y;;
- let a = 3 and b = 4 in c=a+b;;
- let a = 3 and b=4 in c=a+b in c+2;;

Functions

- fun x -> (x*2, x*4, x*8);;
- let f x = x*2;;
- let y = (f 2) in y*2;;

Functions

- fun x -> (x*2, x*4, x*8);;
- let f x = x*2;;
- let y = (f 2) in y*2;;
- let f x = if x>0 then x

  ```
                 else 0;;
  ```

Functions

- fun x -> (x*2, x*4, x*8);;
- let f x = x*2;;
- let y = (f 2) in y*2;;
- let f x = if x>0 then x

  ```
                  else 0;;
  ```
- String.length;;
- String.contains;;

Lists

- List.rev;;
- List.hd;;
- List.tl;;

Lists

- List.rev;;
- List.hd;;
- List.tl;;
- List.hd [1;2;3];;
- List.hd (List.tl [4;5;6]);;

Lists

- List.rev;;
- List.hd;;
- List.tl;;
- List.hd [1;2;3];;
- List.hd (List.tl [4;5;6]);;
- List.append;;
- the same as list1@list2
- [1;2;3]@[4;5];;

Recursive functions

```
let rec f1  = function
   |0 -> 0
   |n -> n + f1(n-1)

let rec f2 n = match n with
   |0 -> 0
   |n -> n + f2 n-1


let rec f3 n m = match n with
   |0 -> m
   |n -> f3 (n-1) m+n
```

Try an exercise!

- Given a list of string *l*, define a function *find* that builds a new list that contains elements from *l* such that the length of each element is less or equal than 3.
- The order of elements should be preserved.
- For example, if l = ["12"; "abcd"; "www"; "456"]

  then result is [“12”; “www”; “456”]

Compilers and Interpreters

Lection 2

Running OCaml

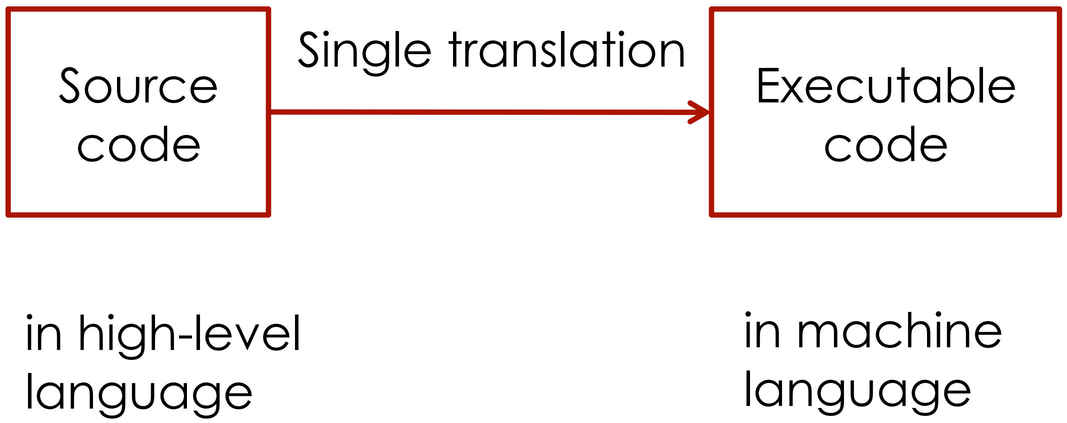
- Run the interpreter with
  - ocaml
- Exit the interpreter:
  - # quit;;
- Compilers:
  - ocamlc compiles in bytecode
- Compilation of a single module
  - ocamlc –c <fileName>.ml
  - Produces <fileName>.cmo

$\left.\right\}$ Interpeter

$\left.\right\}$ Compiler

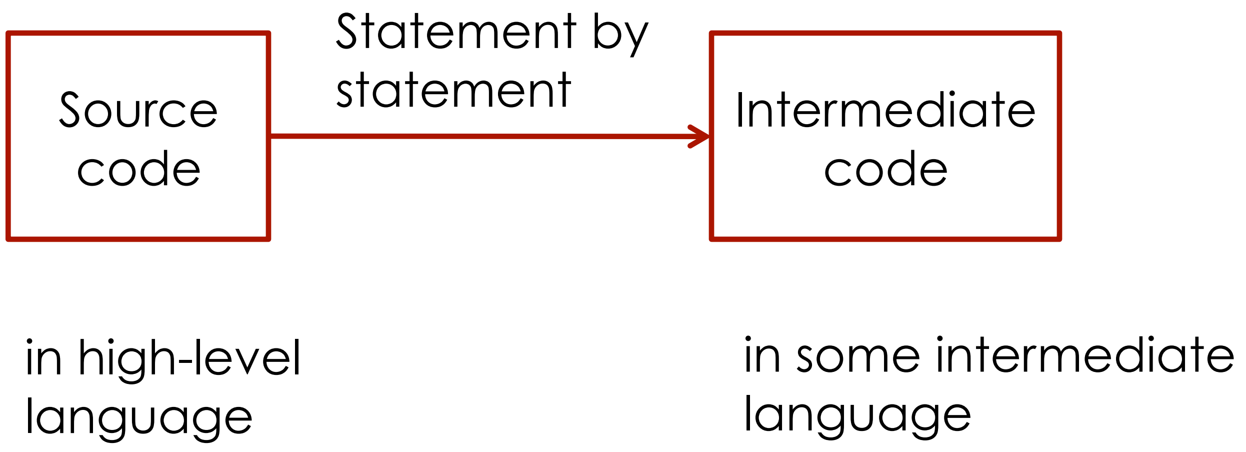
What’ s the difference?

Compiler



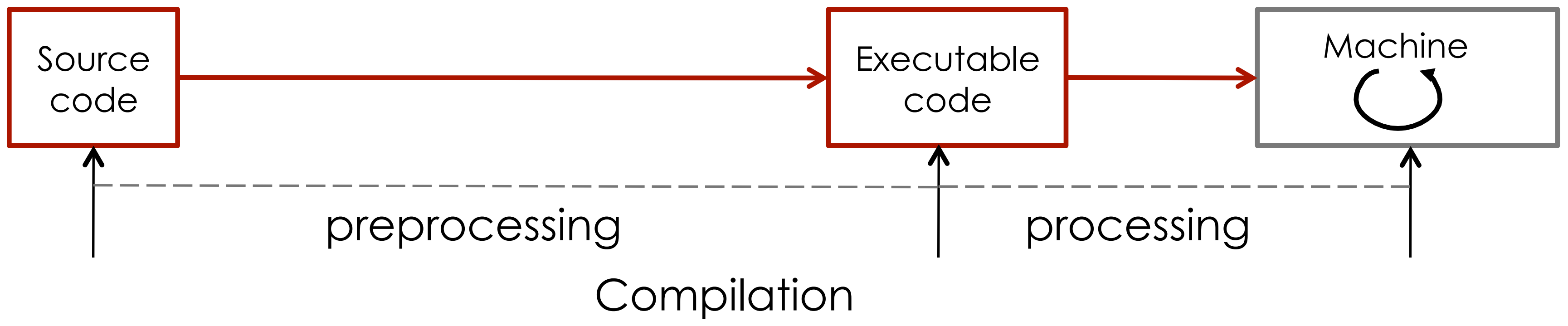
- If an error is found, the source code is not converted

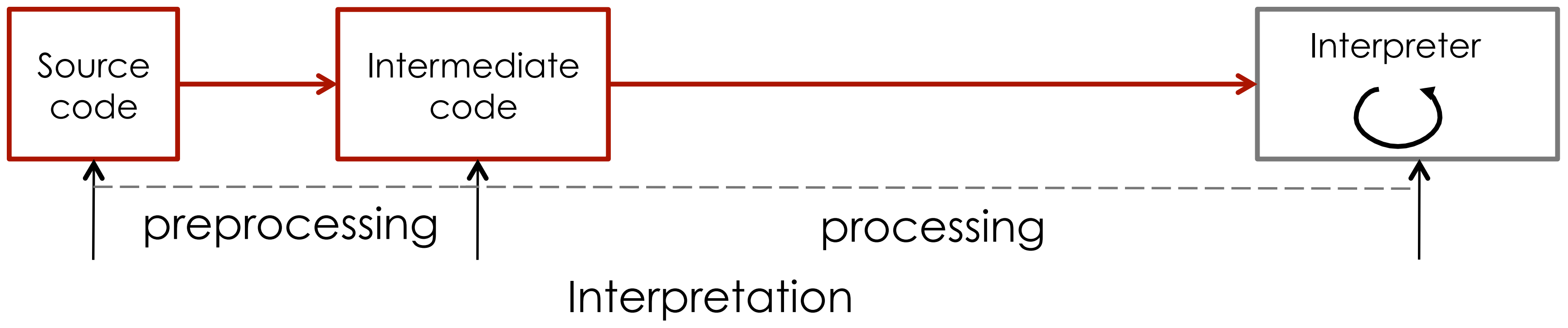
Interpreter



- If an error is found in a statement, the interpreter stops working and shows an error

Compiler vs. Interpreter





Compiler vs. Interpreter

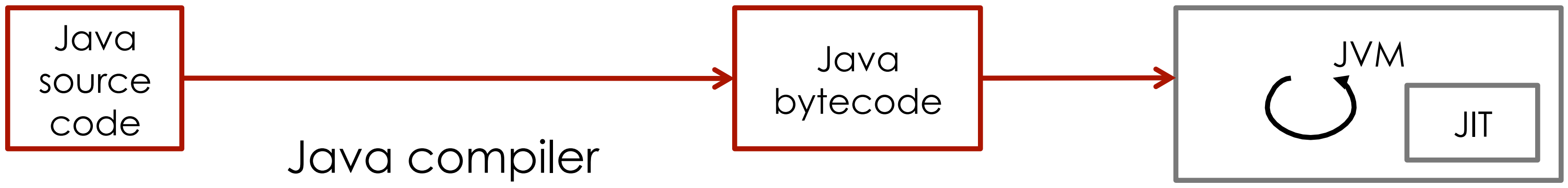
- Compiler characteristics:
  - spends a lot of time analyzing and processing the program
  - the resulting executable is some form of machine- specific binary code
  - the computer hardware interprets (executes) the resulting code
  - program execution is fast

Compiler vs. Interpreter

- Interpreter characteristics:
  - relatively little time is spent analyzing and processing the program
  - the resulting code is some sort of intermediate code
  - the resulting code is interpreted by another program
  - program execution is relatively slow

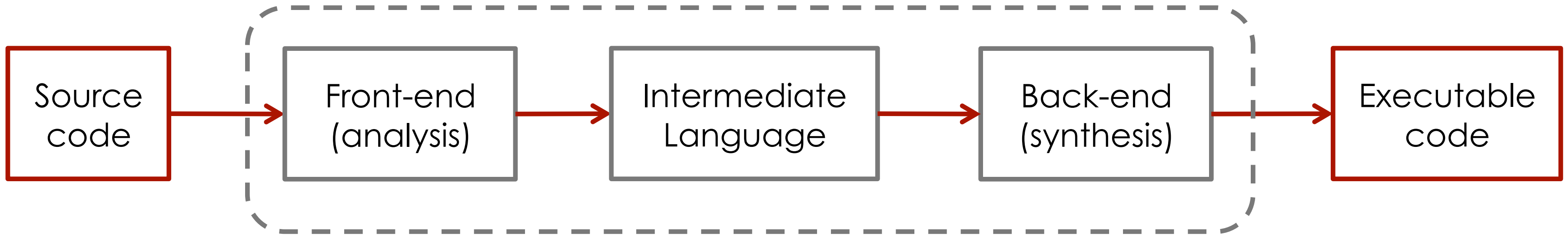
Some real life examples

- C++ compiler
- Java with its Java Virtual Machine (JVM) is something in between, more similar to interpreter

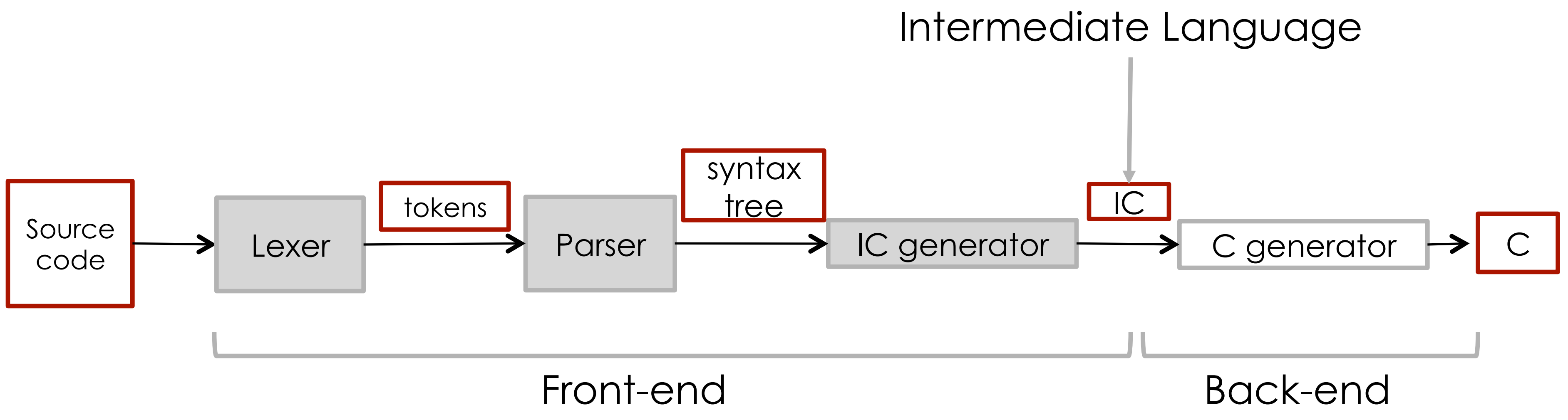


- Java compiler transforms source program to Java bytecode
- JVM is an interpreter of the bytecode
- JIT (Just-In-Time) compiles parts of the bytecode to executable code

Structure of a compiler



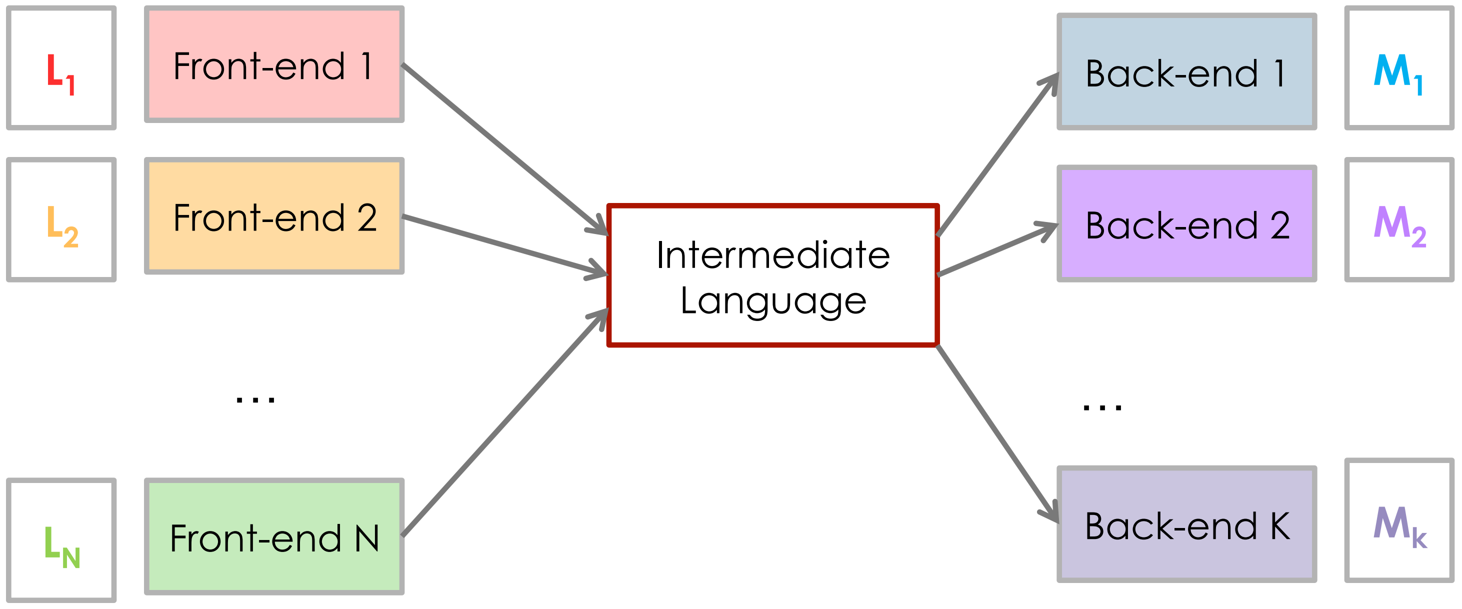
Front-end structure



Back-end structure

- is responsible for emitting the final (executable) version of the source program. Typical parts of the back end are responsible for:
  - instruction selection
  - register allocation
  - memory management
  - instruction scheduling

Front-end and back-end



- Reuse the same front-end for different machines
- Reuse the same back-end for different source languages

References

- CS544: http://web.cs.wpi.edu/~gpollice/cs544-f05/CourseNotes/maps/Class1/Compilervs.Interpreter.html