CSE 340
Principles of Programming Languages

Programming Languages

Ayan Banerjee

Arizona State University
Adopted from slides by Adam Doupe

What is a programming language?

- A structured way to define computation
- Is this the only definition/purpose?
  - Communicate an algorithm
  - Describe a process
  - Communicate a system to another person
  - Communicate instructions to a machine

How does the computer understand your instructions?

- The CPU understands assembly language

How does the computer understand your instructions?

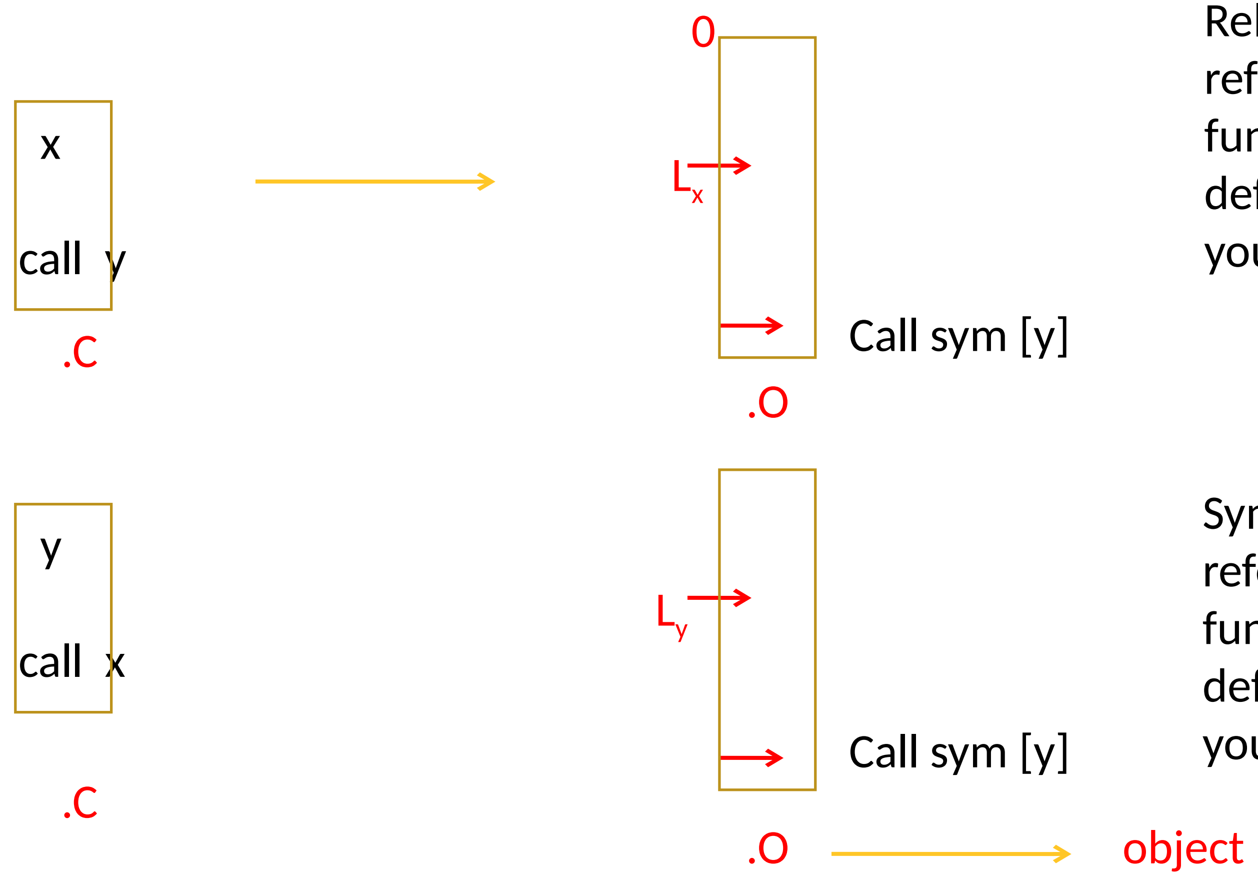
- Programs translate your intentions to the assembly language that the CPU understands
- Compilers
  - Translate programming language to executable binary
- Interpreters
  - Understand a programming language and perform the actual computation
- Transpiler
  - Translate a programming language to another programming language

How do they work?

- In this class, we will study how these programs are able to translate a high-level programming language into something the computer can understand
- Theory
  - Enables us to define what we can and cannot do when defining a programming language
- Practice
  - Empowers us to develop domain-specific languages, task-specific compilers, static analysis, parsing …

Compiler

Compiler converts high level code into machine language (well not really)



Relative references for functions defined in your code

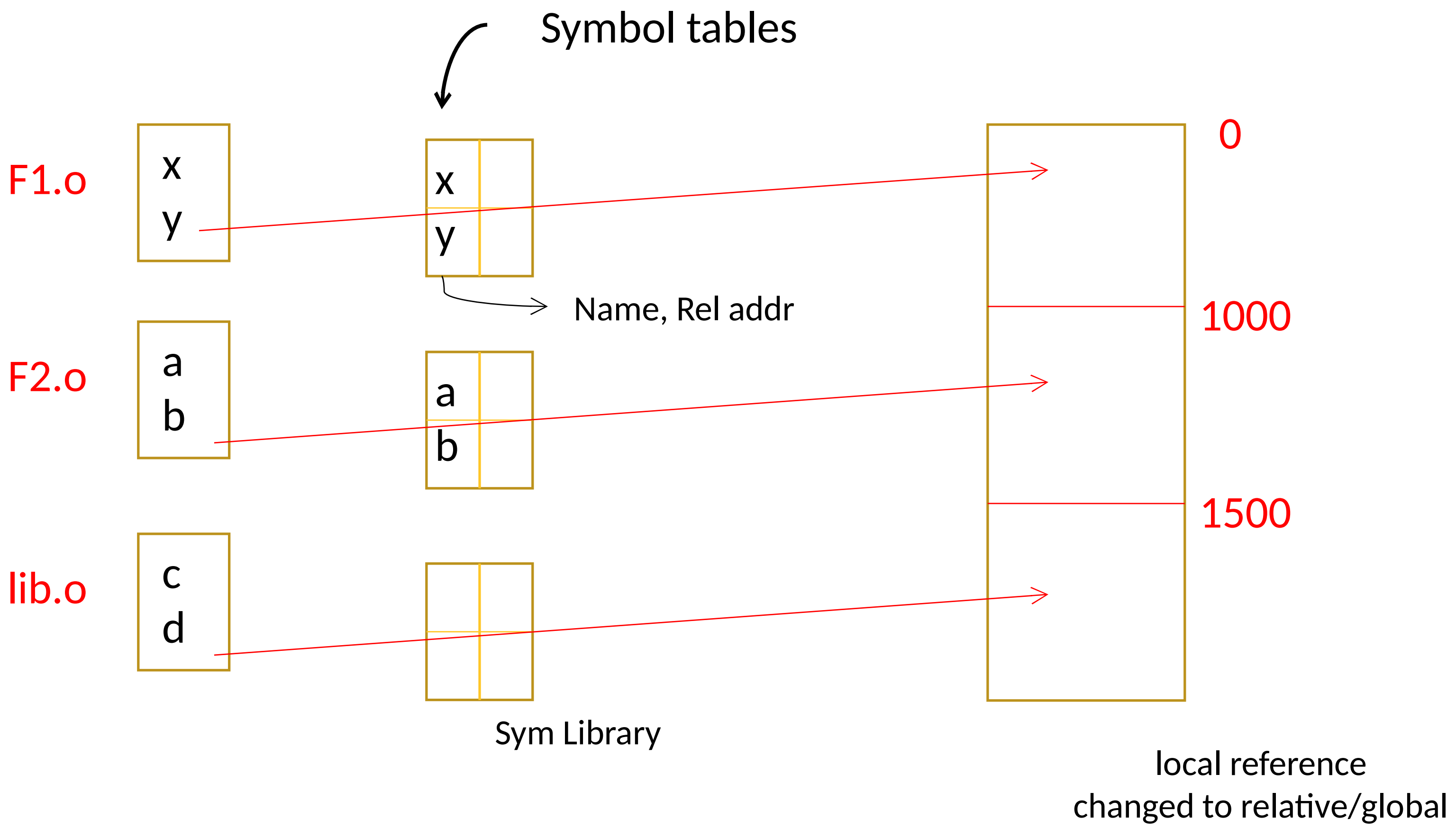
Symbolic references for functions **not** defined in your code

Relocation Table

- Table of pointers to lines of code that need linking to different libraries
  - The symbolic references
- A symbol table is also created in this step, which has information about the symbols and what functions have to be loaded for a given symbol

Linker

Uses relocation table to find lines of code to be replaced and then uses the symbol tables to find which functions have to be used.

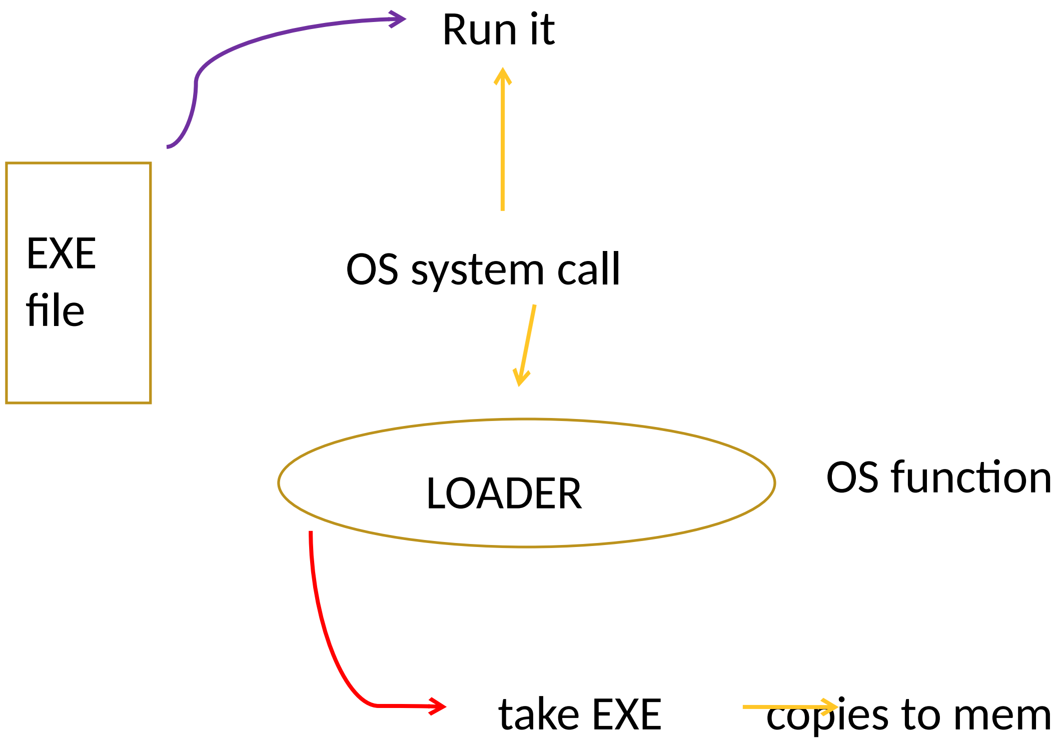


Linker

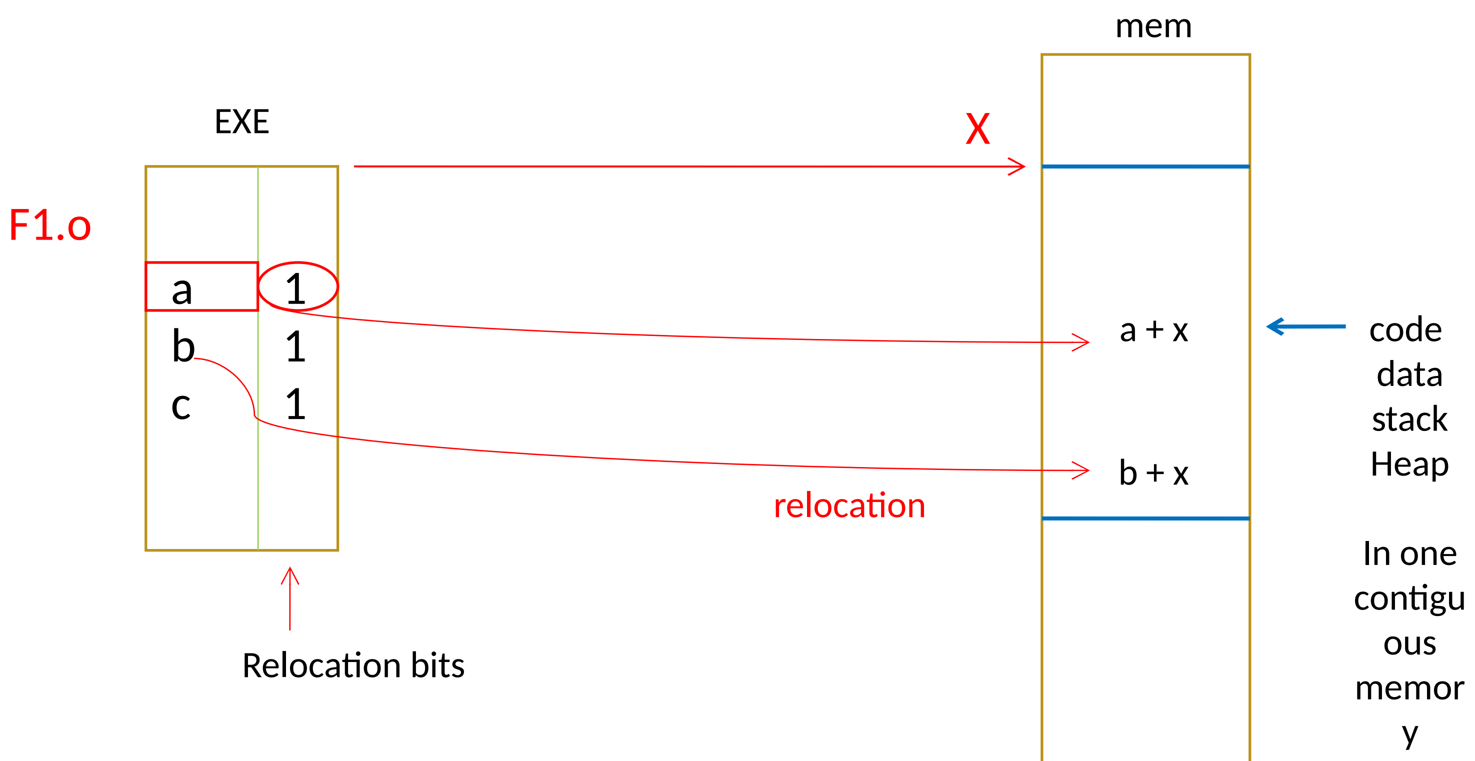
1. Create 1 output binary (merged), start at 0
2. Fix all local references with global relative address
3. Fix all symbolic refs

Loader

Takes the output of linker and copies them to memory.



Loader

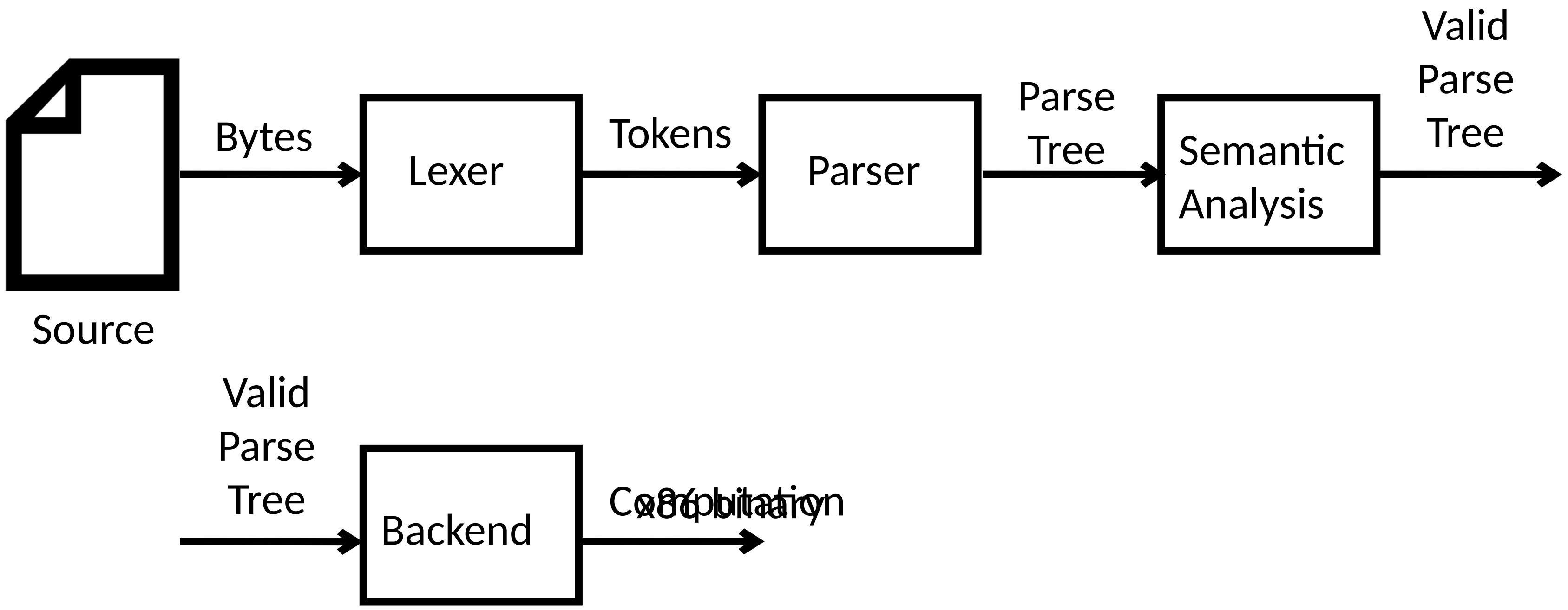

mem
EXE
X
F1.o
a
1
b
1
c
1
a + x
b + x
relocation
Relocation bits
code
data
stack
Heap
In one contiguous memory

Create a Process

- Process Control Blocks (PCB)
  - Process ID
  - Program Counter = start address of code
  - Status Register

- Put the PCB in the Ready Queue

Overview of Program Interpretation



What makes a program valid?

- Syntax
  - What does it mean to look like a valid program?
- Semantics
  - What does it mean for a program to be valid?
- Correctness?
  - Is the program the correct one for the job?