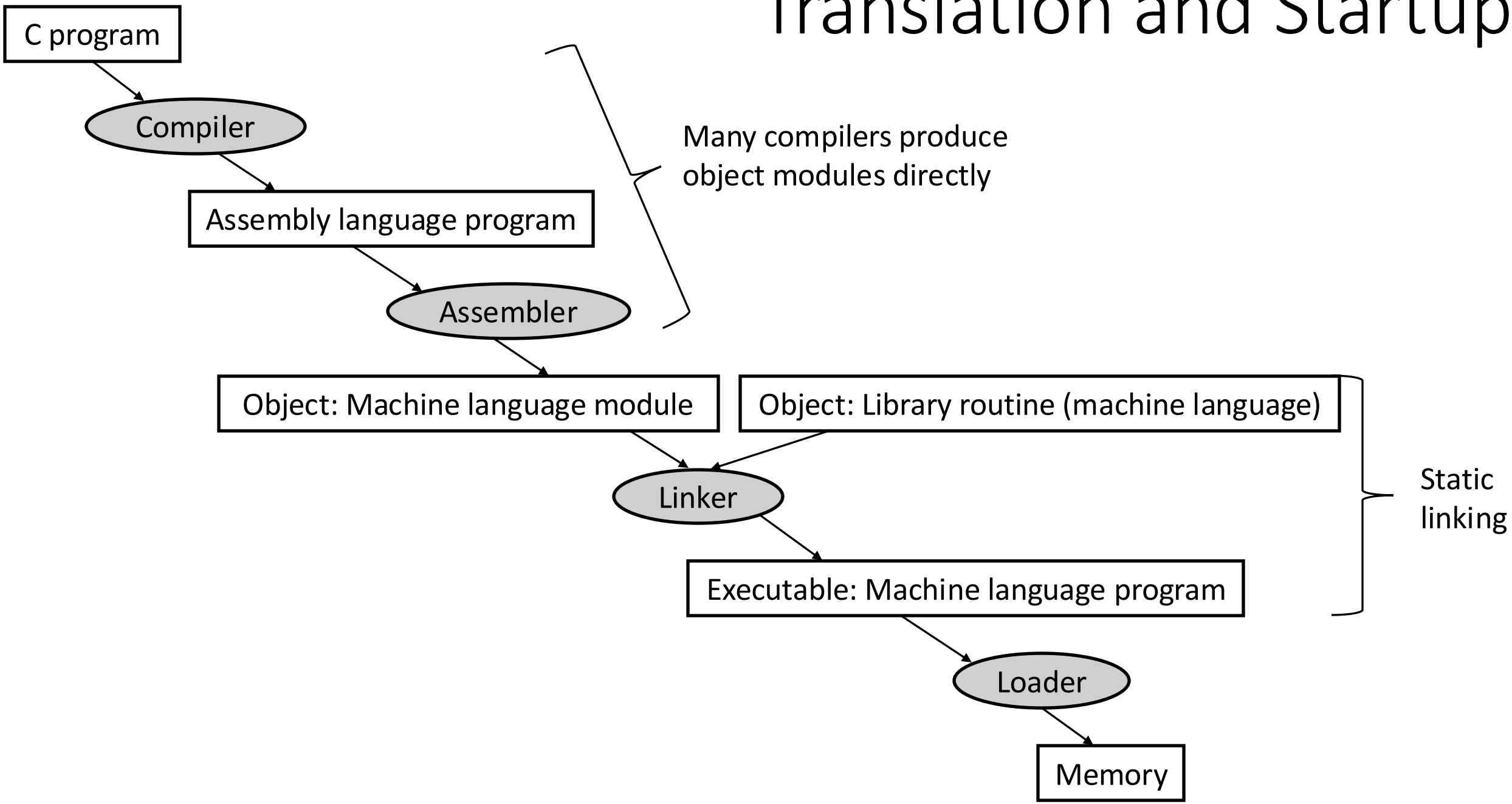
Topic V1F

Compiling, Linking, & Loading

Readings: (Section 2.12)

Translation and Startup



Producing an Object Module

Assembler (or compiler):

- Translates program into machine instructions
- Provides information for building a complete program from the pieces

Object module

Header

Describes content

Text Segment

Instructions

Static Data Segment

Data Allocated for the life of the program

Relocation Info

Used to place content that depends on program’s absolute address

Symbol Table

Global Definitions and external references

Debug Info

Associate instructions with source code

Linking Object Modules

Produces an executable image

1. Merge segments
2. Resolve labels (determine their addresses)
3. Path location-dependent and external references

Could leave location dependencies for fixing by a relocating loader

But with virtual memory, no need to do this

Program can be loaded into an absolute location in virtual memory

Loading a Program

Load from image file on disk into memory

1. Read header to determine segment sizes
2. Create virtual address space
3. Copy text and initialized data into memory
   Or set page table entries so they can be faulted in
4. Set up arguments on stack
5. Initialize registers (including `sp`, `fp`, `gp`)
6. Jump to startup routine
   Copies arguments to `a0`, … and calls main
   When main returns, invoke exit syscall

Dynamic Linking

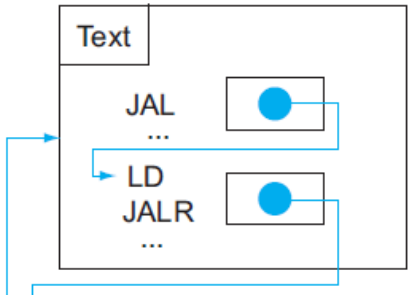
Only link/load a library procedure when it is called

- Requires procedure code to be *rellocatable**
- Avoids image bloat caused by statically linking all (transitively) referenced libraries
- Automatically picks up new library versions

**Rellocatable code* requires a linker to "fix" the code to run in a new location

**Position-independent code* is assembly code that can run anywhere in memory without change

Lazy Linkage



Indirection table

Stub: loads routine ID,
jump to linker/loader

Linker/loader code

Dynamically mapped code

(a) First call to DLL routine (b) Subsequent calls to DLL routine

Dynamic JIT Compilers