CS 35L Software Construction Lab
Week 8 – Dynamic Linking

Library

- A **library** is a package of code that is meant to be reused by many programs.
- A C++ library comes in two pieces
- Header file : defines the functionality the library is exposing (offering) to the programs using it
- Precompiled binary: contains the implementation of that functionality pre-compiled into machine language

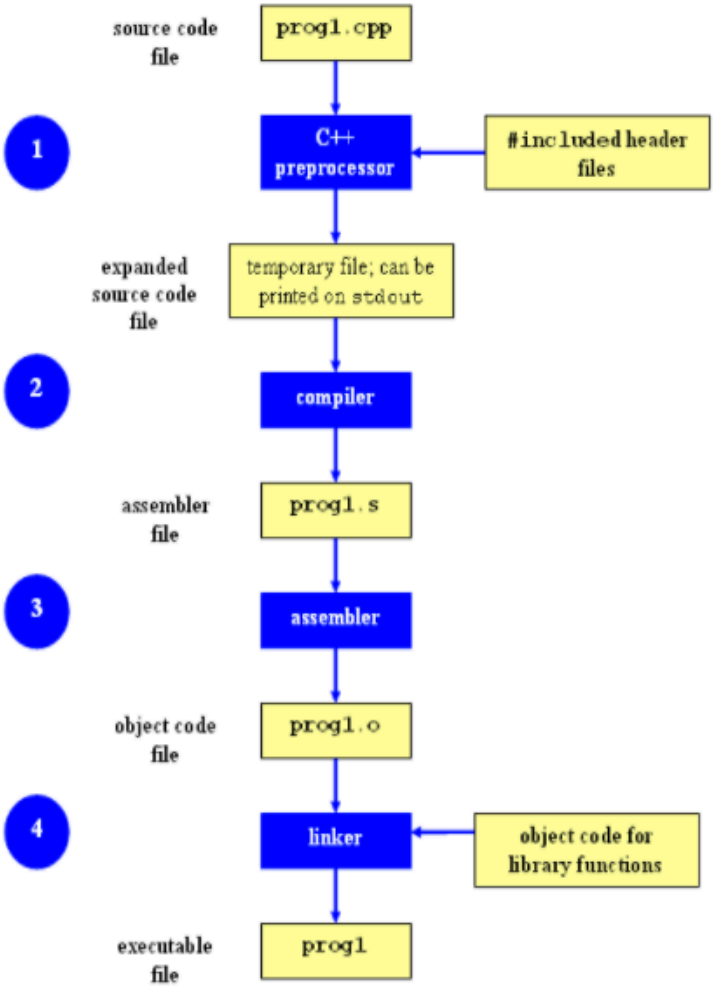
Question: Why do we want library code to be precompiled?

Library

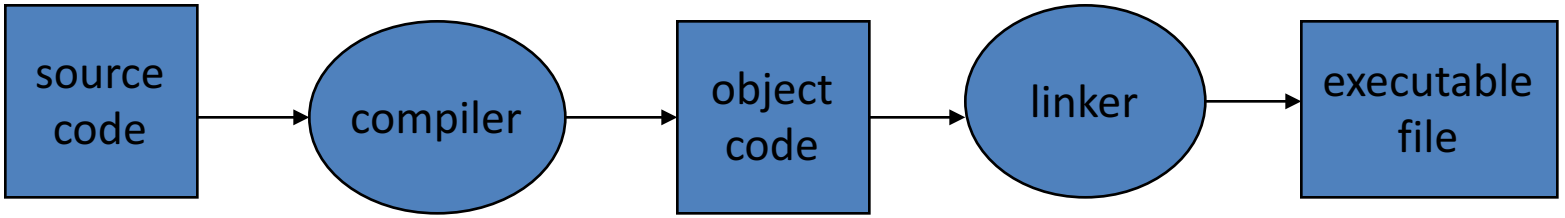
- Two types of libraries
  - static libraries
  - dynamic libraries
- Static Library (archive)
  - Windows uses .lib extension, linux uses .a (archive) extension.
  - Used in static linking
- Dynamic Library (shared object)
  - Windows uses .dll (dynamic link library) extension, Linux uses .so (shared object) extension
  - Used in dynamic linking and dynamic loading

Compilation Process

- Preprocessor
  - Expand Header includes, macros, etc
  - -E option in gcc to show the resulting code
- Compiler
  - Generates machine code for certain architecture
- Linker
  - Linkall modules together
  - Address resolution
- Loader
  - Loads the executable to memory to start execution



Building an executable file



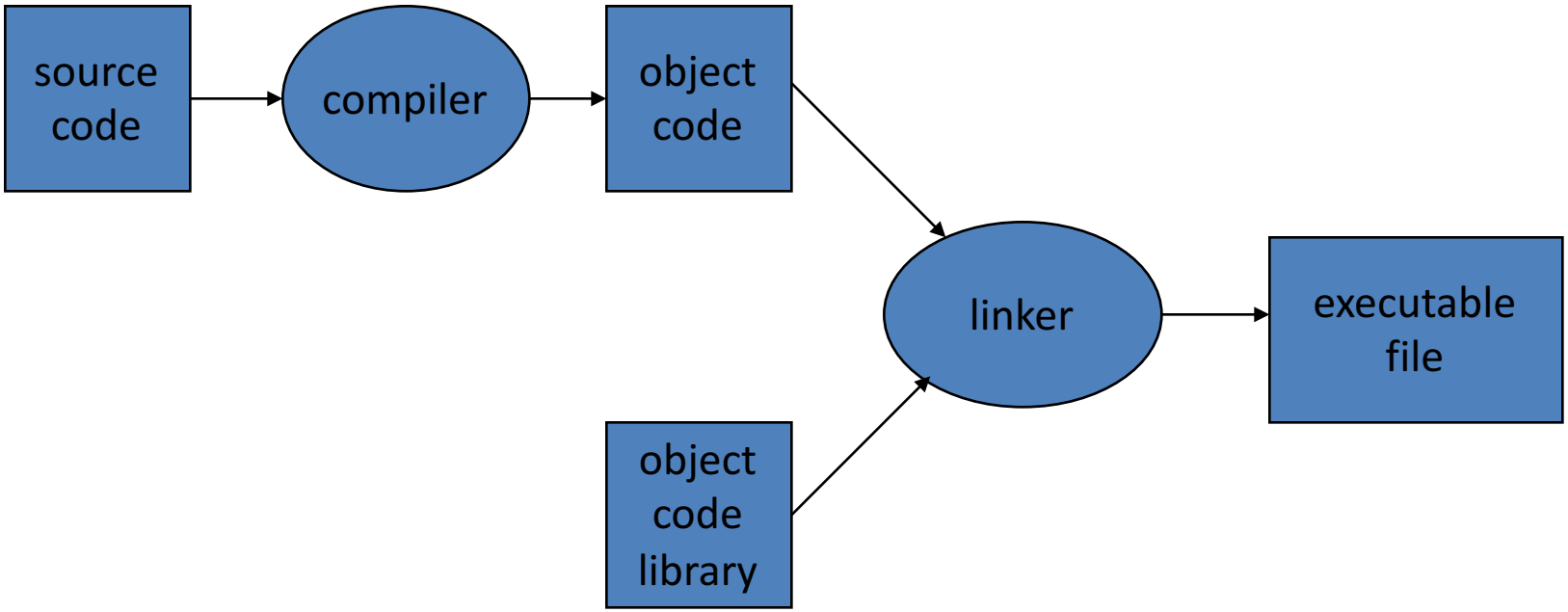
Translates programming language statements into cpu's machine-language instructions

Adjusts any memory references to fit the Operating Sytem's memory model

Static Linking

- Carried out only once to produce an executable file
- If static libraries are called, the linker will copy all the modules referenced by the program to the executable
- Static libraries are typically denoted by the .a file extension

Linking libraries

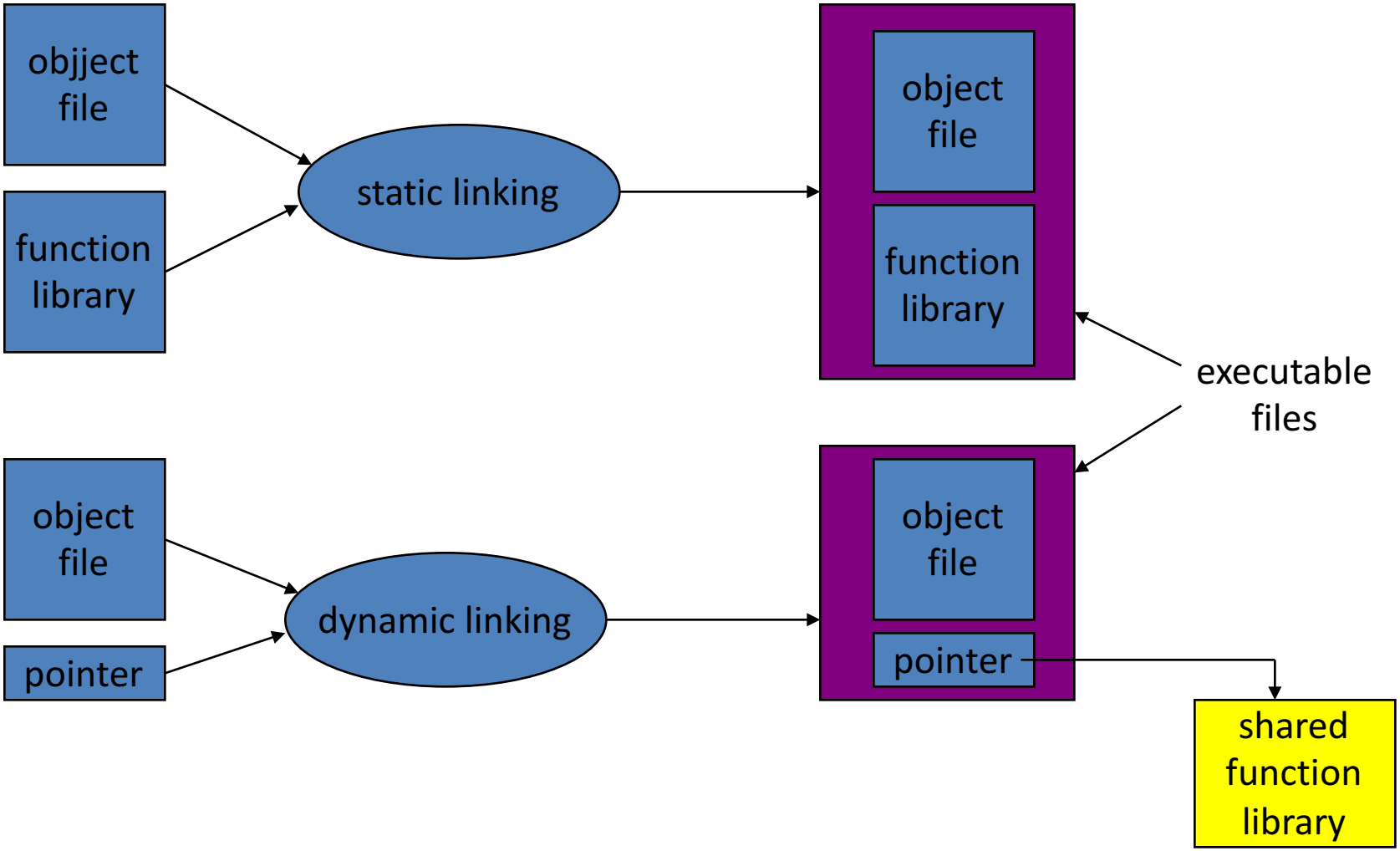


A previously compiled collection of standard program functions

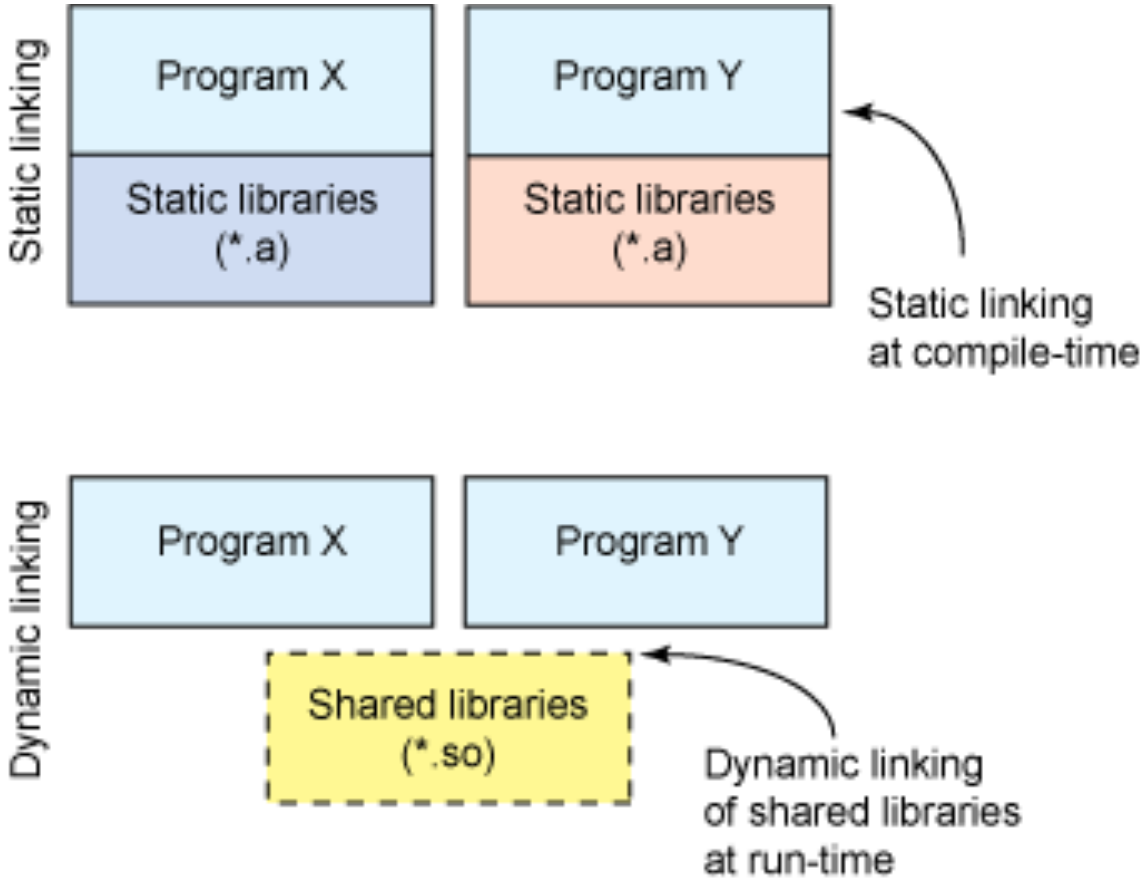
Dynamic Linking

- If shared libraries are called:
  - Only copy a little reference information when the executable file is created
  - Complete the linking during loading time or running time
- Dynamic libraries are typically denoted by the .so file extension
  - .dll on Windows

Smaller is more efficient



Q: What are the pros and cons?



Linking and Loading

- Why isn't everything written as just one **big** program, saving the necessity of linking?
  - Efficiency: if just one function is changed in a 100K line program, why recompile the whole program? Just recompile the one function and relink.
  - Multiple-language programs
  - Other reasons?

Dynamic linking

- Dynamic vs. static linking resulting size

```
$ gcc -static hello.c -o hello-static
$ gcc hello.c -o hello-dynamic
$ ls -l hello
     80 hello.c
  13724 hello-dynamic
    383 hello.s
1688756 hello-static
```

Advantages of dynamic linking

- The executable is typically smaller
- When the library is changed, the code that references it does not usually need to be recompiled
- The executable accesses the .so at run time; therefore, multiple programs can access the same .so at the same time
  - Memory footprint amortized across all programs using the same .so

Disadvantages of dynamic linking

- Performance hit
  - Need to load shared objects (at least once)
  - Need to resolve addresses (once or every time)
- What if the necessary dynamic library is missing?
- What if we have the library, but it is the wrong version?

Lab 8

- Write and build simple “cos(0.5)” program in C
  - Use `ldd` to investigate which dynamic libraries your hello world program loads
  - Use `strace` to investigate which system calls your hello world program makes
- Use “`ls /usr/bin | awk 'NR%101==SID%101'`” to find ~25 linux commands to use `ldd` on
  - Record output for each one in your log and investigate any errors you might see
  - From all dynamic libraries you find, create a sorted list
    - Remember to remove the duplicates!