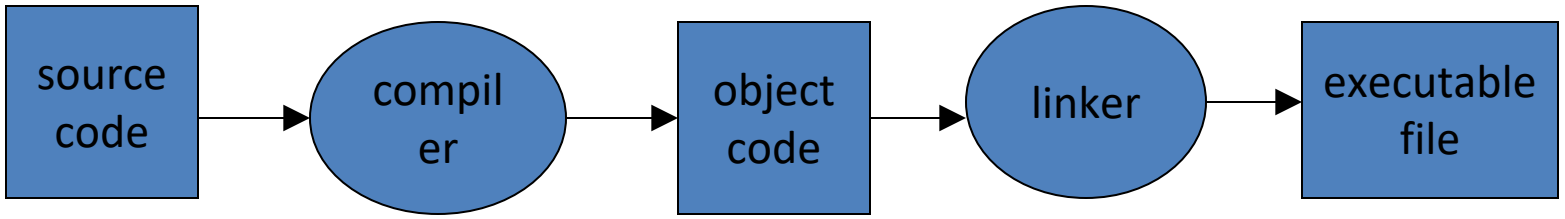
CS35L – Fall 2018

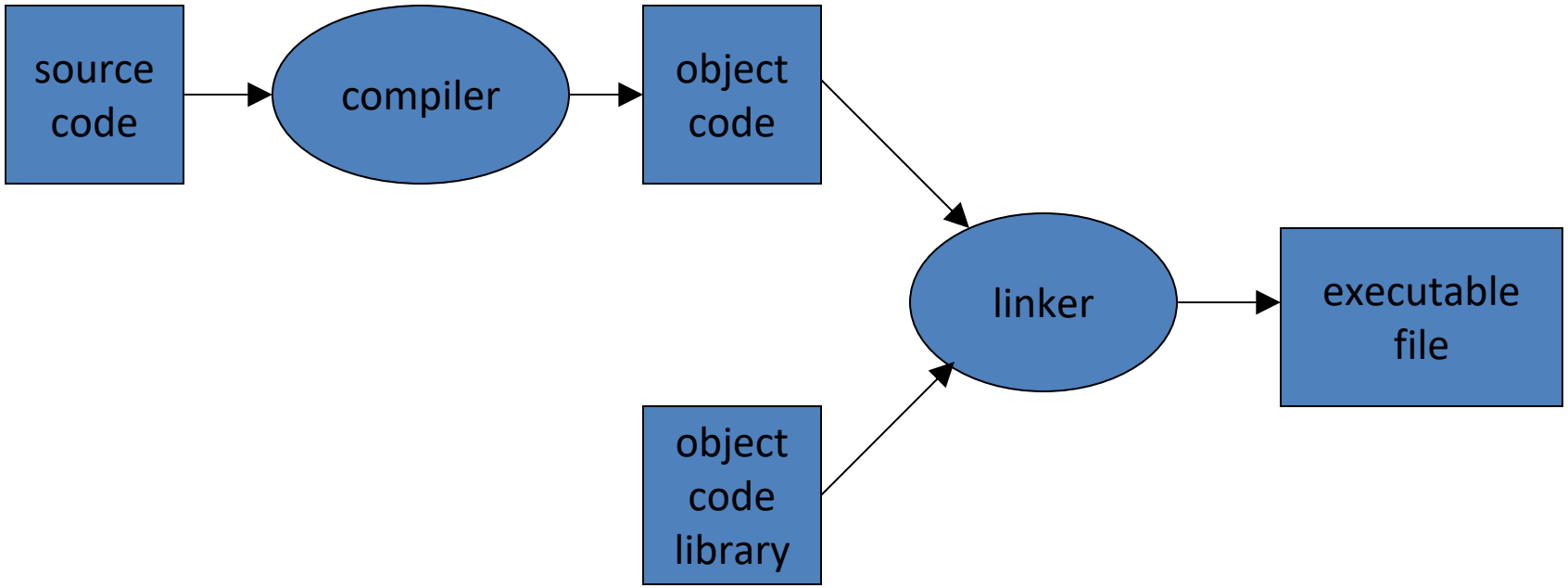
Slide set:	7.1
Slide topics:	Dynamic linking
Assignment:	7

Building an executable file



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

Takes one or more object files generated by a compiler and combines them into a single executable file



A previously compiled collection of standard program functions

Static Linking

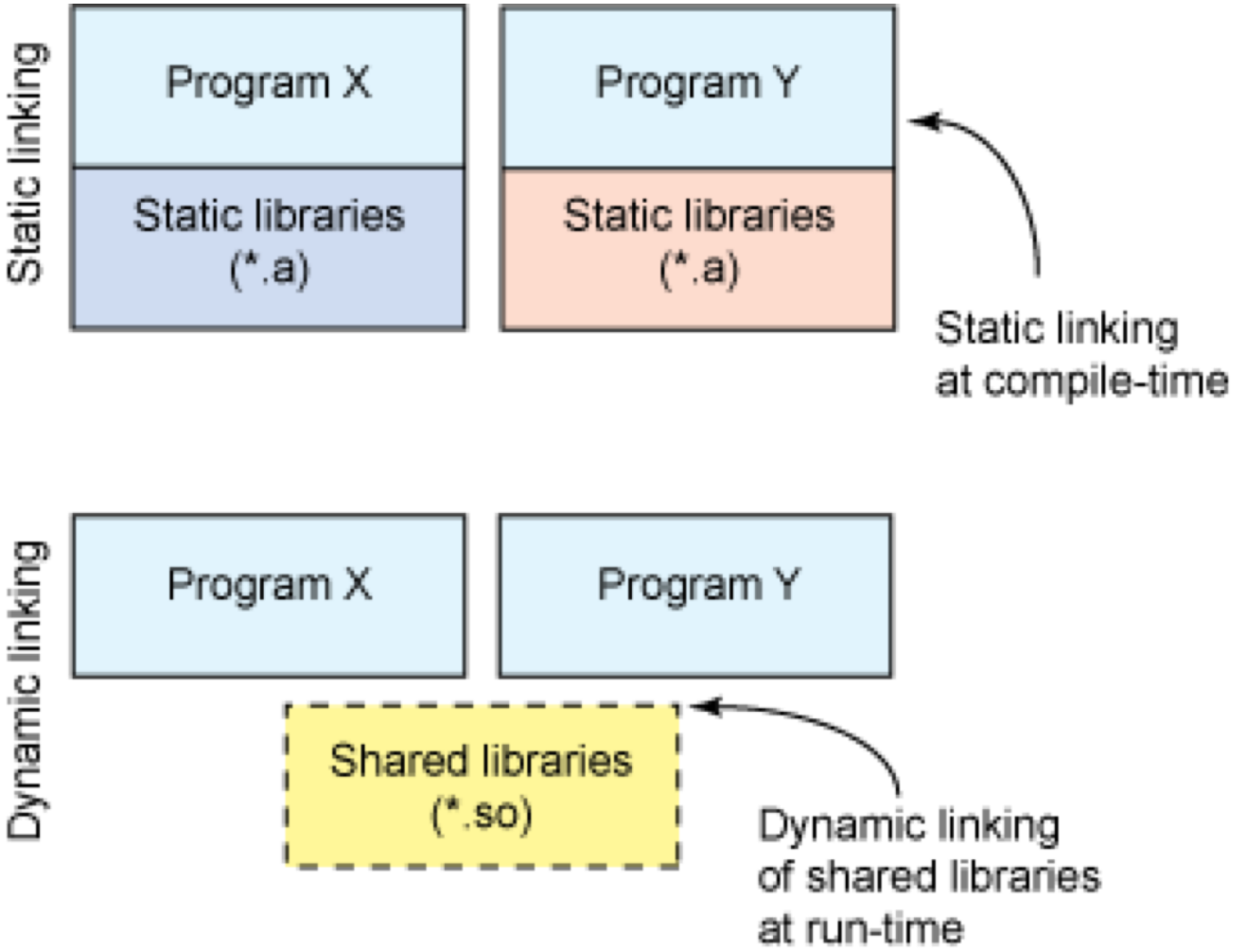
- Carried out only once to produce an executable file
- If static libraries are called, the linker will copy all the modules referenced by the program to the executable
- Static libraries are typically denoted by the .a file extension

Dynamic Linking

- Allows a process to add, remove, replace or relocate object modules during its execution.
- 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

Linking and Loading

- Linker collects procedures and links together the object modules into one executable program
- 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?


Static linking
Program X
Static libraries
(*.a)
Program Y
Static libraries
(*.a)
Static linking
at compile-time
Dynamic linking
Program X
Program Y
Shared libraries
(*.so)
Dynamic linking
of shared libraries
at run-time

Dynamic linking

- Unix systems: Code is typically compiled as a *dynamic shared object* (DSO)
- Dynamic vs. static linking resulting size

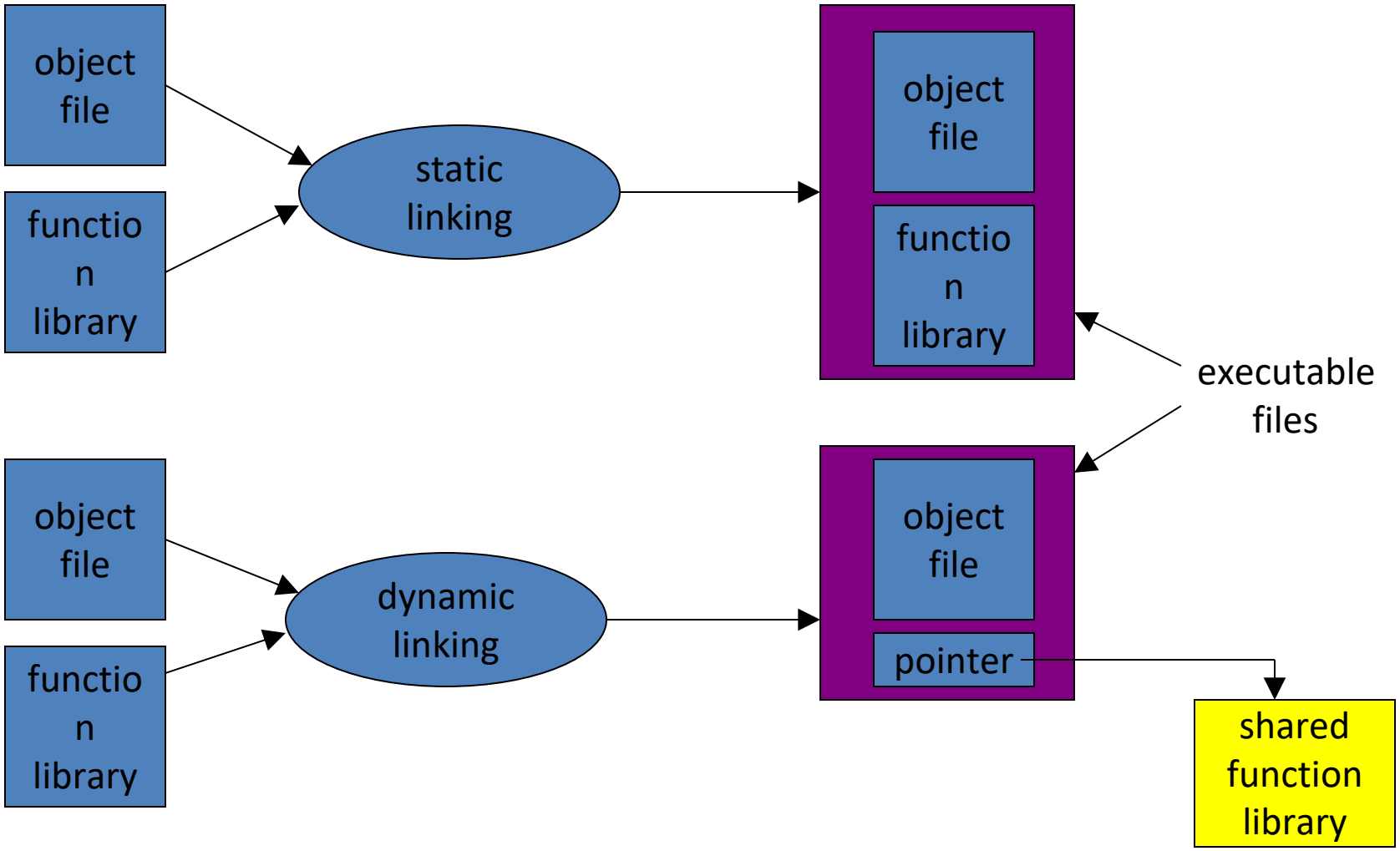
```
$ gcc -static hello.c -o hello-static
$ gcc hello.c -o hello-dynamic
$ ls -l hello
     80 hello.c
  13724 hello-dynamic
1688756 hello-static
```

- If you are the sysadmin, which do you prefer?

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

Smaller is more efficient



Disadvantages of dynamic linking

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

Lab 7

- Write and build simple `cos(sqrt(3.0))` program in C
  - Use `ldd` to investigate which dynamic libraries your cos program loads
  - Use `strace` to investigate which system calls your cos program makes
- Use "`ls /usr/bin | awk 'NR%101==nnnnnnnnn%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 omit the duplicates!