CS 35L- Software Construction Laboratory

Winter 19

TA: Guangyu Zhou

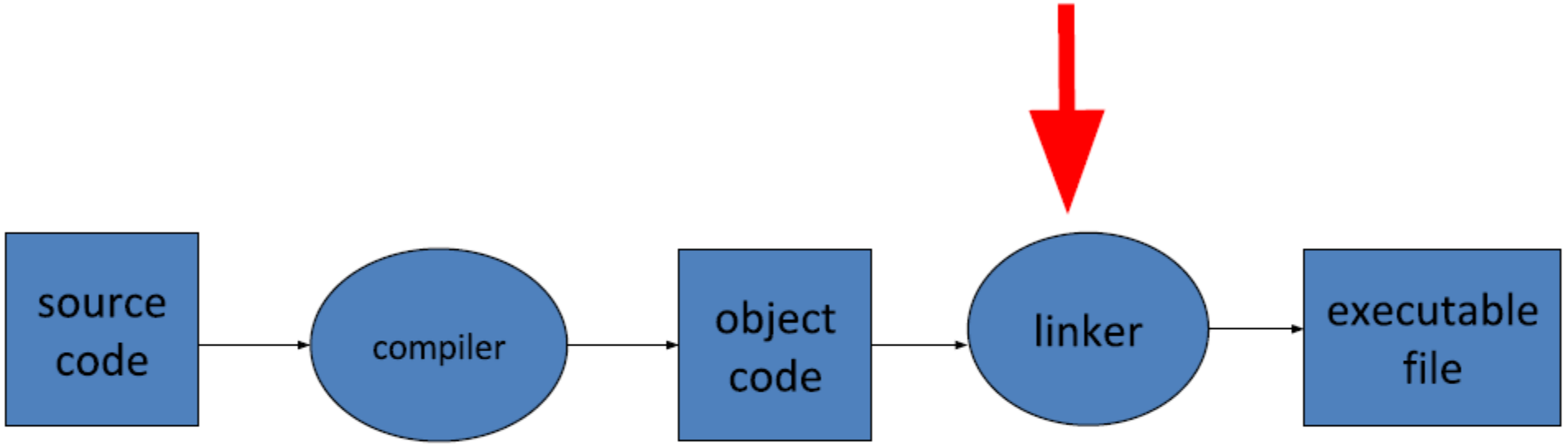
Dynamic Linking

Week 8

Outline

- Dynamic Linking
- GCC options and flags
- Hints for Assignment 7

Building an executable file

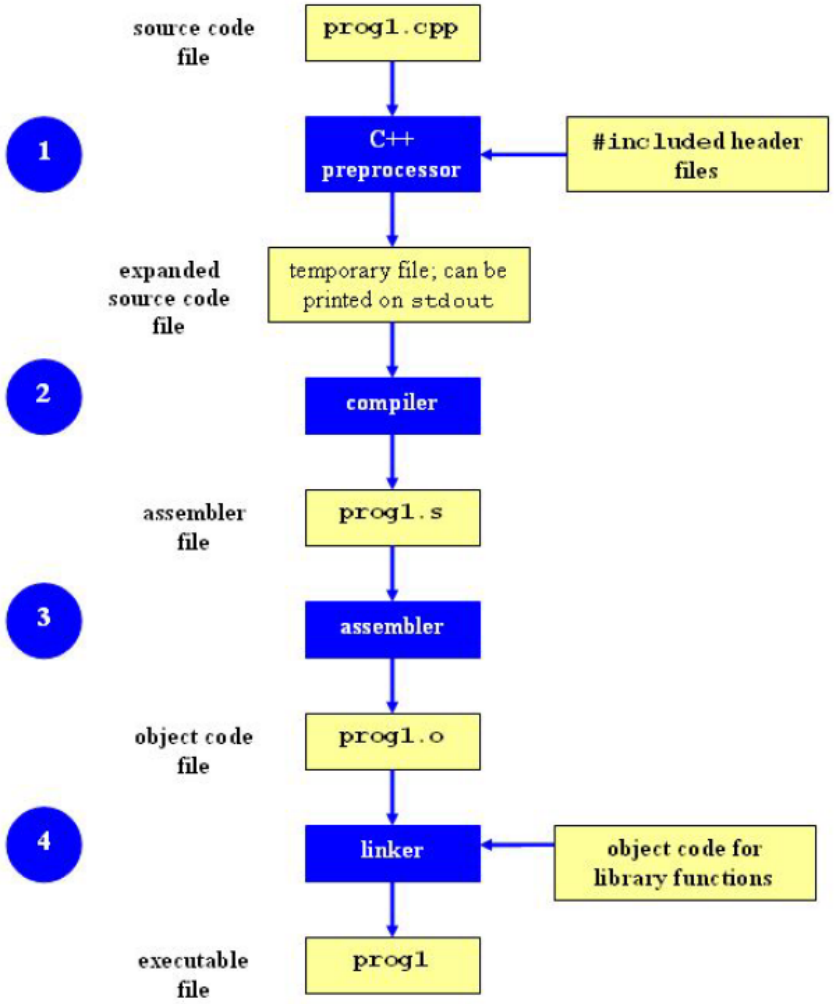


Translates programming language statements into cpu's machine-language instructions

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

Compilation Process

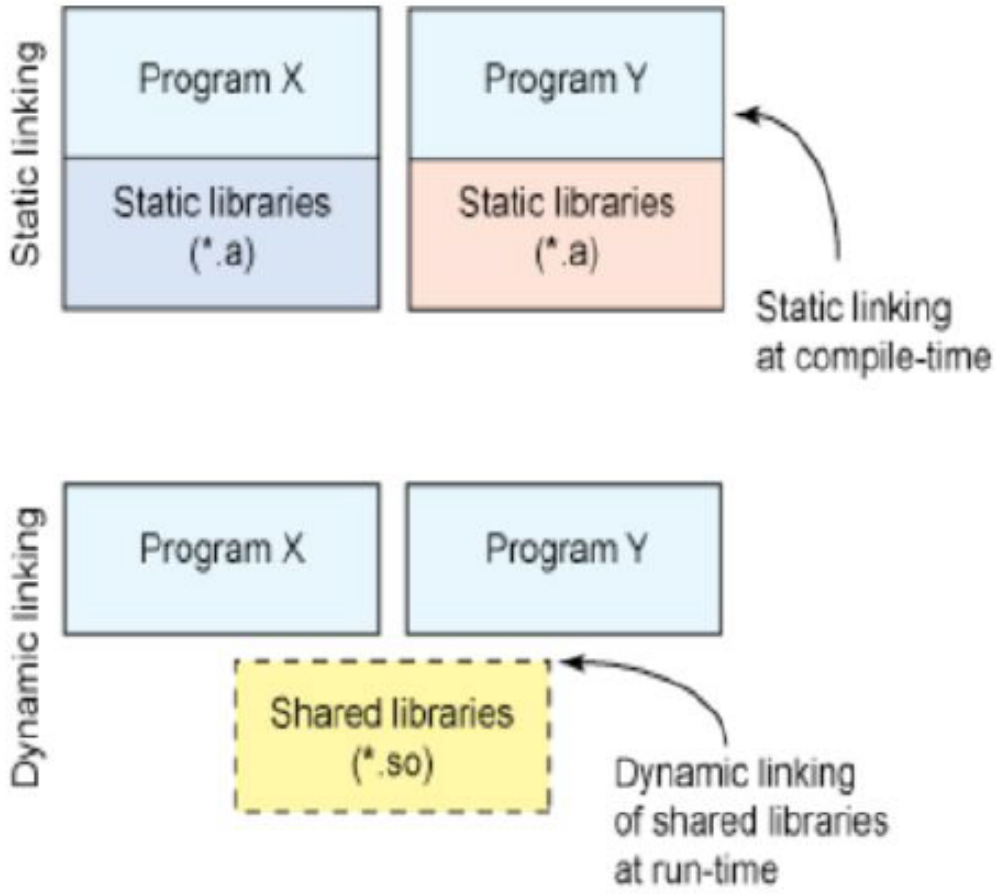
- Preprocessor
  - Expand header includes, macros, etc
- Compiler
  - Generate machine code for certain architecture
- Assembler
  - Create object code
- **Linker**
  - **Link all modules together**
  - **Address resolution**
- Loader
  - Load the executable to memory to start execution



Linux Library

- Static Library
  - Statically linked
  - Every program has its own copy
  - More space in memory
  - Tied to a specific version of the lib. New version of the lib requires recompile of source code
- **Shared Library**
  - Dynamically linking/load
    - Dynamic linking: The OS loads the library when needed. A dynamic linker does the linking for the symbol used.
    - Dynamic load: The program actively loads the library it needs. More control to the program at runtime.
  - Library is shared by multiple programs
  - Lower memory footprint
  - New version of the lib doesn’t require a recompile of source code

Static vs Shared Library

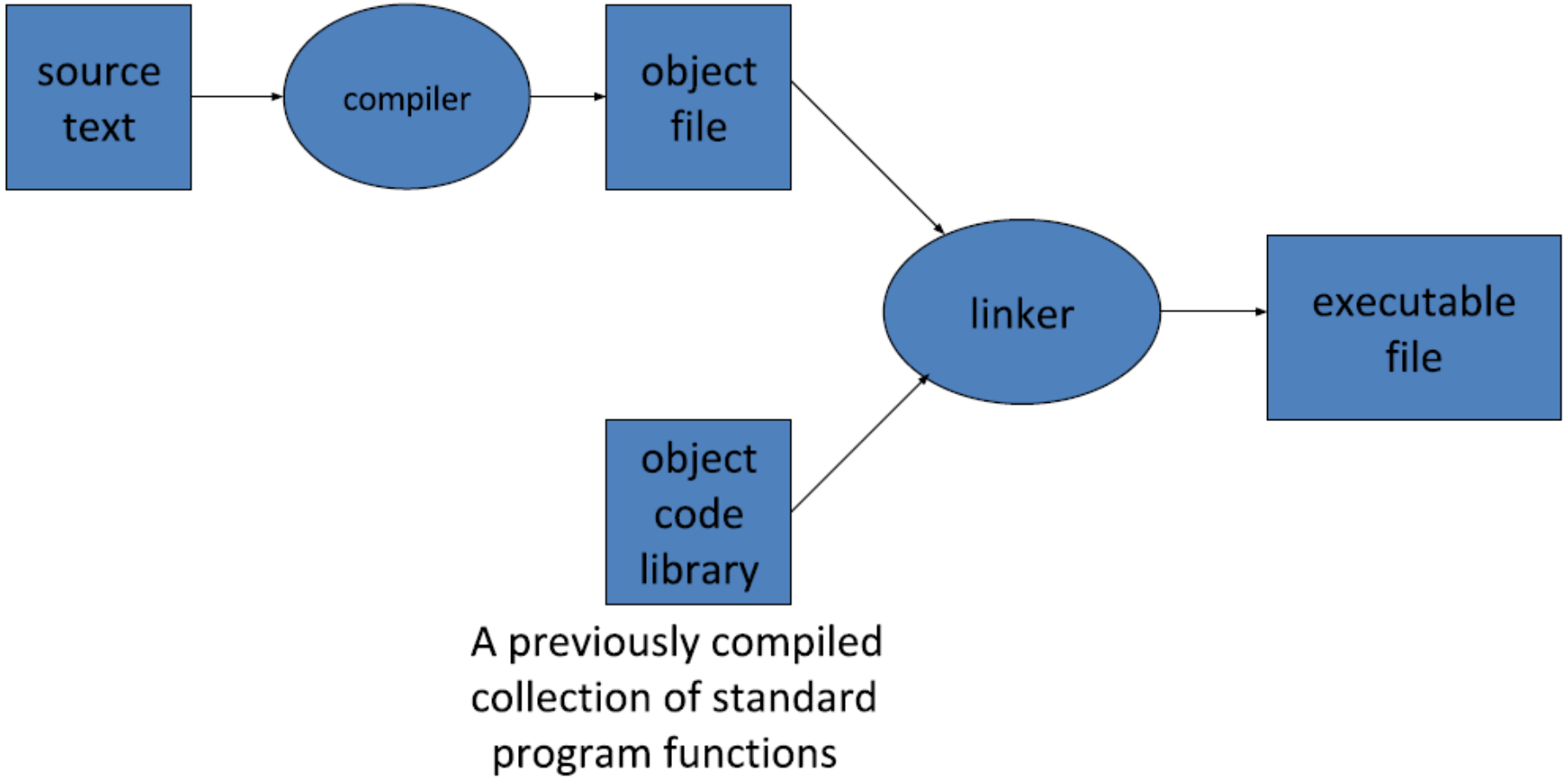


Img Source : http://www.ibm.com/developerworks/library/l-dynamic-libraries/

Static Linking

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

Static Linking



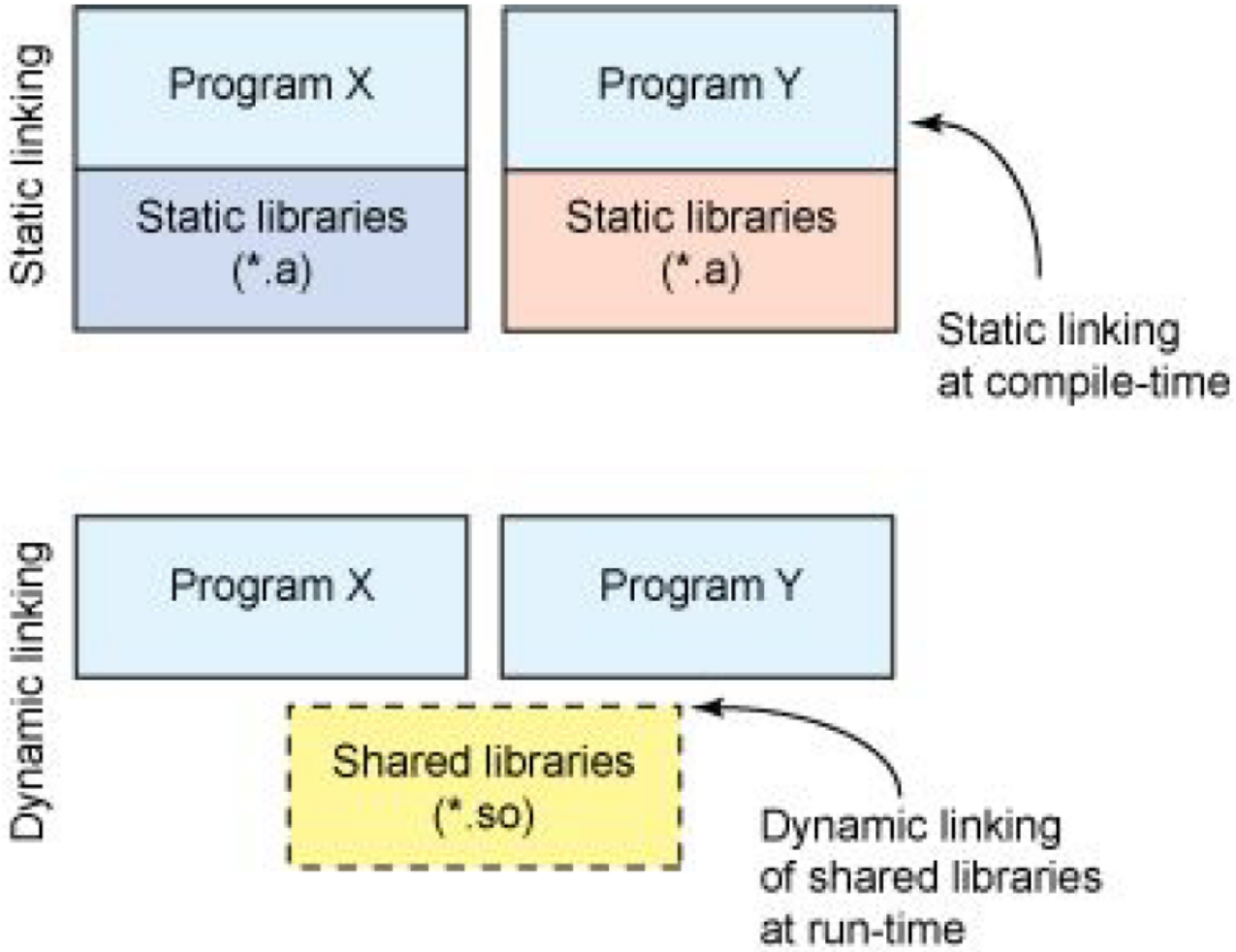
Dynamic Linking

- Allow a process to add, remove, replace or relocate object modules during its execution.
- If shared library are called
  - Only copy a little reference information when the executable file is created
  - Complete the linking during the loading or running time.
- Dynamic libraries are typically denoted by the .so file extension.
  - Like .dll in Windows

Linking and Loading

- Linker collects procedures and links them together 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

Linking and Load



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

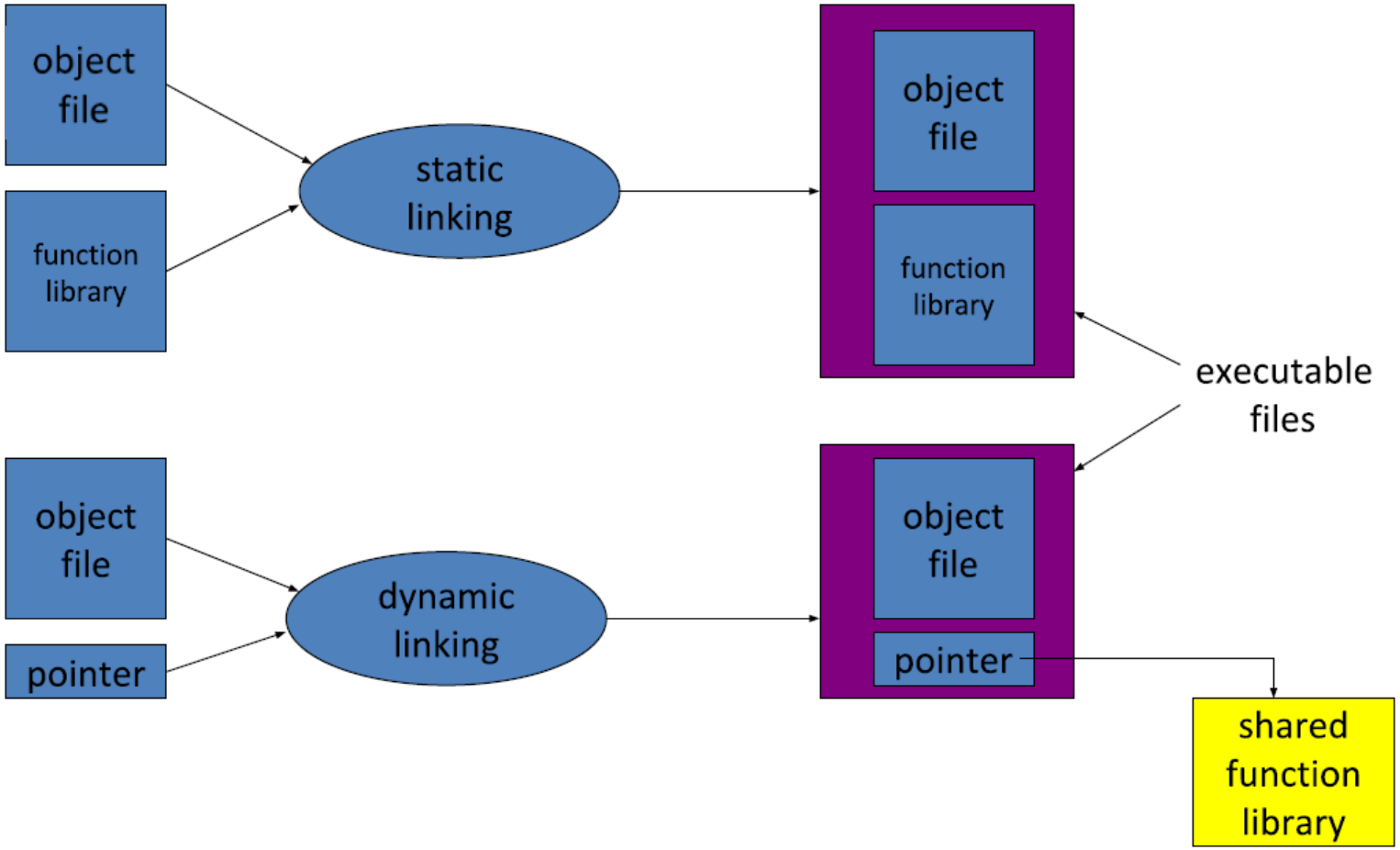
    383 hello.s

    1688756 hello-static
- If you are the system admin, 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?

How are libraries dynamically linked?

Table 1. The DI API

Function	Description
dlopen	Makes an object file accessible to a program
dlsym	Obtains the address of a symbol within a `dlopen`ed object file
dlerror	Returns a string error of the last error that occurred
dlclose	Closes an object file

GCC Options

- -c: compile and create object files
- -o: name of output file
- -I(upper-case i): additional folders to search for header files
- -L: additional folders to search for libraries to link with
- -shared: create shared libraries
- -l(lower case L): Name of additional library to link with
- -fpic: Output position independent code. Required for shared libraries.

Assignment 7 is available

- Visit:

http://web.cs.ucla.edu/classes/winter19/cs35L/assign/assign7.html

More References of Dynamic Linking: check supplement materials

Piazza Note:

- The homework part of the assignment 7 requires special hardware.
- Please do, build and test your assignment only on lnxsrv09. The assignment will also be graded under lnxsrv09.
- You can ssh into lnxsrv09 by doing ssh username@lnxsrv09.seas.ucla.edu.

Lab 7: Who's linked to what?

- Write and build simple math program in C
  - Compute cos(sqrt(3.0)) and print it using the format “%.17g”
  - Use **ldd** to investigate which dynamic libraries your program loads
  - Use **strace** to investigate which system calls your program makes
- Use “ls /usr/bin | awk ‘NR%101==UID%101’” to find the 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 (remove duplicate)

Lab 7

```
#!/bin/bash

for x in "$(ls /usr/bin | awk 'NR%101==your_uid%101'
$1)"; do
  y=`which $x`
  ldd $y
done
```

example run, unique sort, need to omit addresses at end:

```
./ldd_run  | grep so | sort -u
```