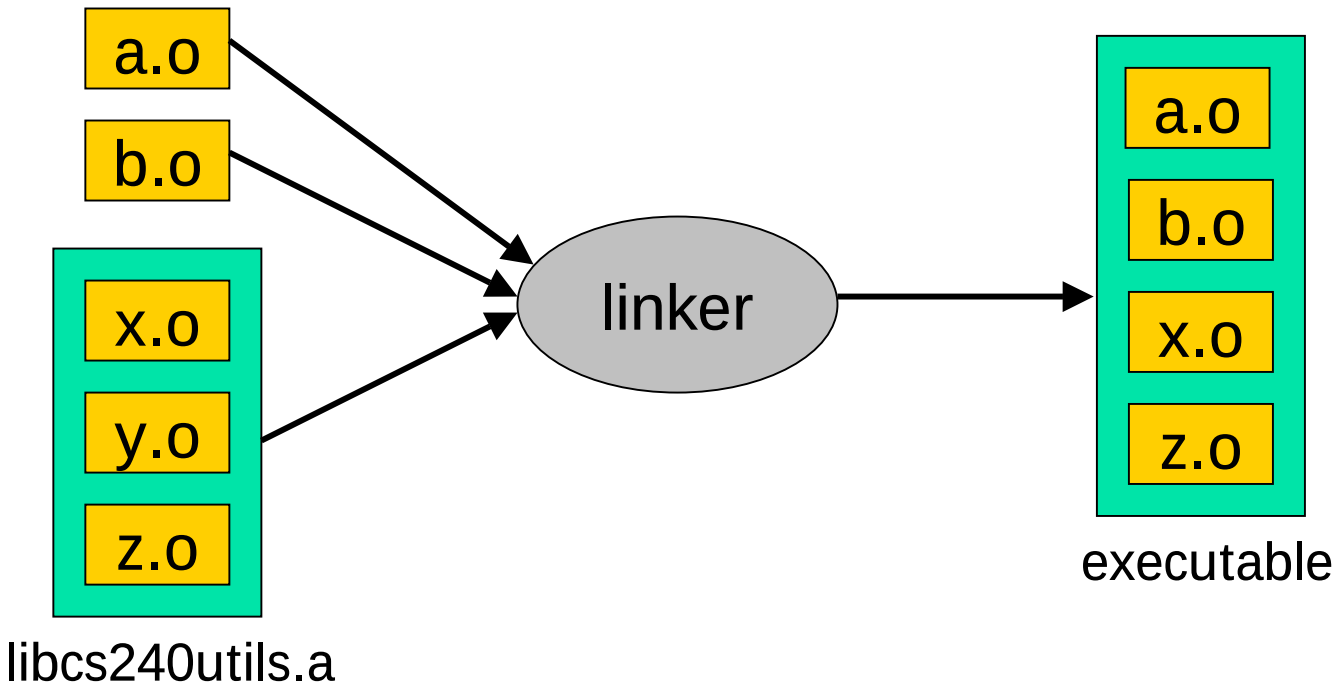
Shared Libraries

Review: Static Libraries

- A static library is just a bunch of `.o` files that are stored together in an archive file
- Creating a static library
  - `ar -rcs ../lib/libcs240utils.a *.o`
  - `libcs240utils.a`
  - Always use `lib` at the front and `.a` at the end:
- Linking with a static library
  - `g++ -o ../bin/crawler *.o ../lib/libcs240utils.a`
  - OR
  - `g++ -o ../bin/crawler *.o -L../lib -lcs240utils`

Review: Static Libraries

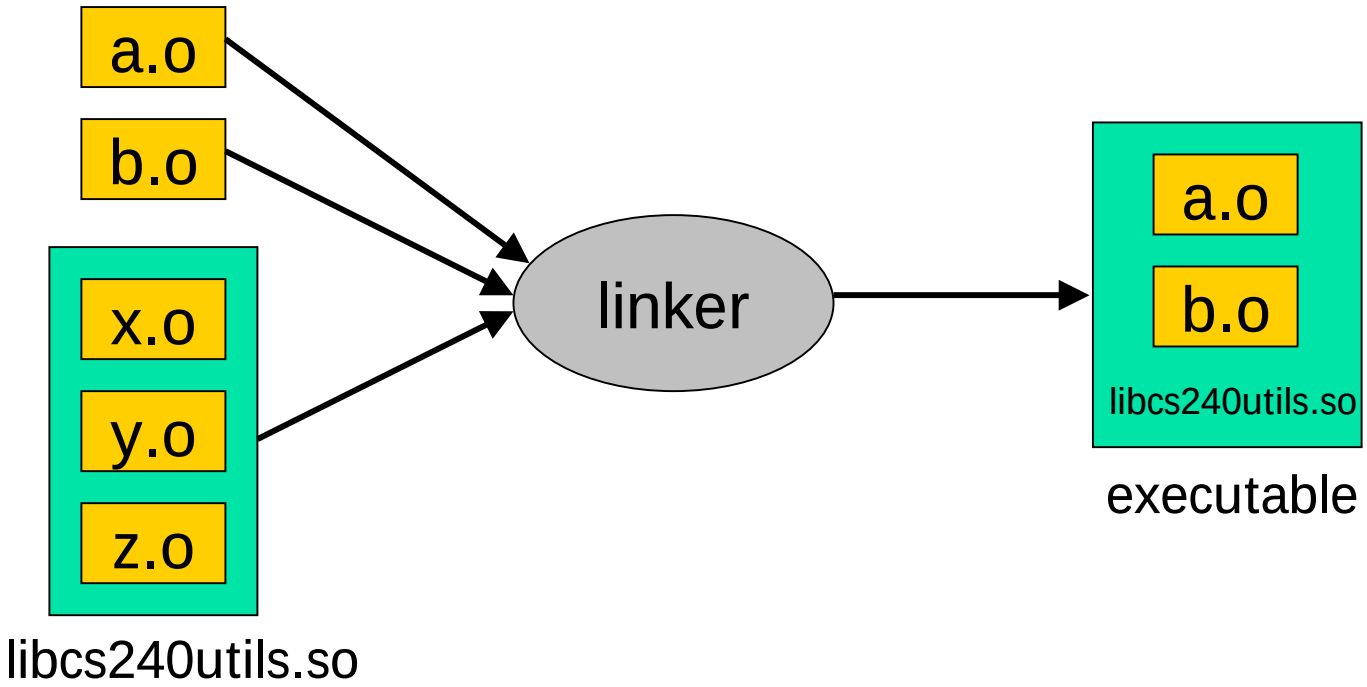
- When you link the executable, the linker copies the code that it needs out of the static library into the executable file
- The executable is stand-alone (it doesn't depend on any other files to run)



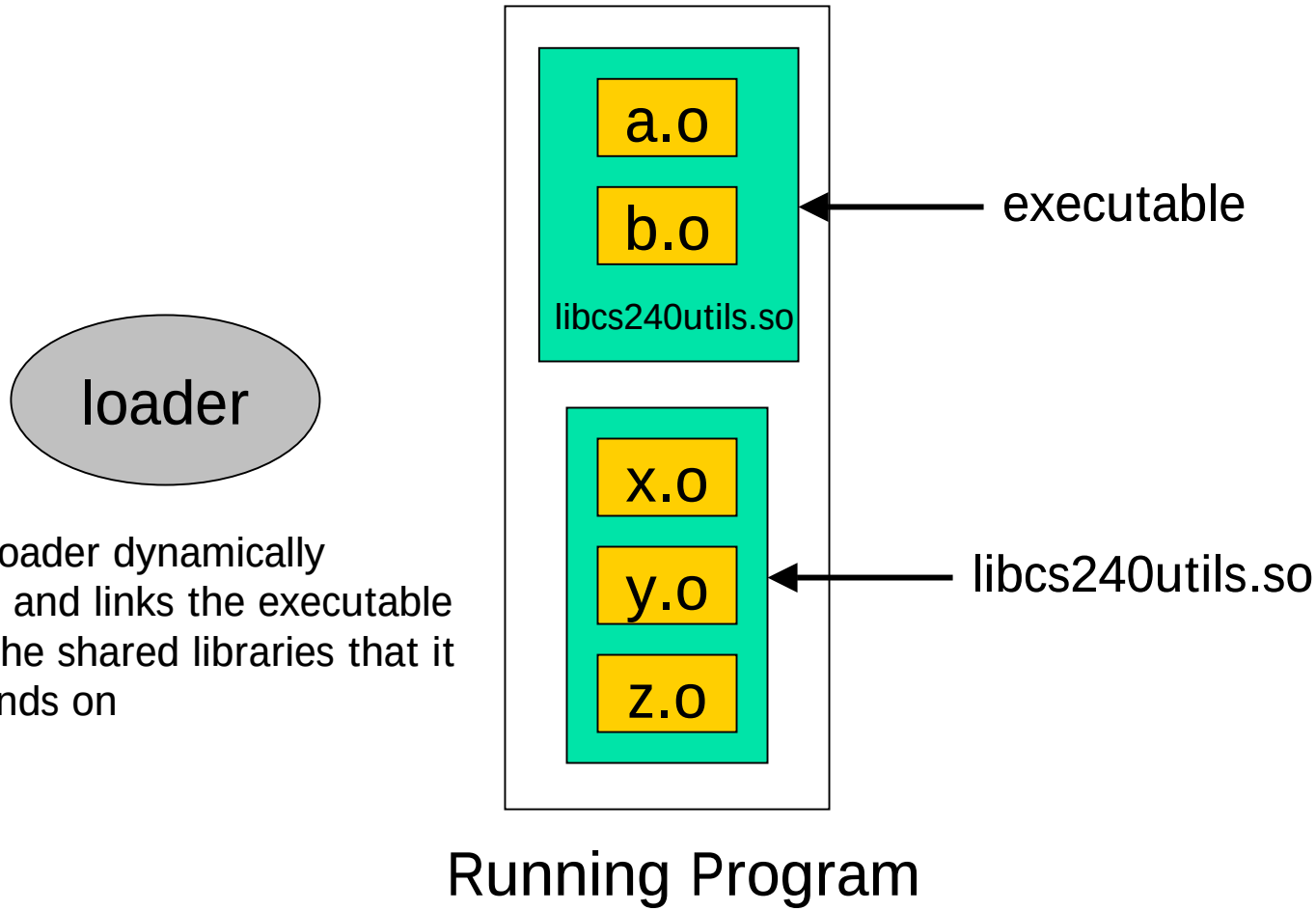
Shared Libraries

- A shared library is similar to a static library, except that the executable does not contain a copy of the library's code
- When the program is run, the loader loads the executable file into memory and all of the shared library files that it depends on
- The loader dynamically links in the shared library code at runtime
- The executable file is not stand-alone because it won't run if the necessary shared libraries are missing
- The same idea as DLLs on MS Windows

Shared Libraries



Shared Libraries



The loader dynamically loads and links the executable and the shared libraries that it depends on

Shared Libraries

- Advantages
  - Saves disk space because every program doesn't have its own copy of the library code
  - Saves memory because all programs that rely on a shared library can share one copy of it in memory
  - Easier to upgrade the library's code; just replace the `.so` file and all programs automatically use the new code
- Disadvantages
  - Executable files are no longer stand-alone
  - Program won't run if a shared library isn't there or can't be found
  - Upgrading shared libraries can break programs that relied on certain behaviors in the old version of the library

Shared Libraries

- Creating a shared library
  - The library `.o` files must be compiled with the `-fPIC` option
  - `g++ -c -fPIC *.cpp`
  - `PIC` stands for "position independent code"
  - The shared library itself is created like this:
  - `g++ -shared -o ../lib/libcs240utils.so *.o`
  - `libcs240utils.so`
  - Always use `lib` at the front and `.so` at the end:
- Linking with a shared library (same as static library)
  - `g++ -o ../bin/chess *.o ../lib/libcs240utils.so`
  - OR
  - `g++ -o ../bin/chess *.o -L../lib -lcs240utils`
- If you use `-L` and `-l`, the linker will look for both `.a` and `.so` files (if both `.a` and `.so` exist, the linker seems to prefer the `.so`)

Shared Libraries

- How does the loader go about finding shared library files at runtime?
  - The loader looks in certain directories for shared libraries (`/lib, /usr/lib`)
  - System administrators can modify the list of directories that are searched using a program named `ldconfig`
  - The `LD_LIBRARY_PATH` environment variable can be set to contain a list of directories that should be searched

Shared Libraries

Interactively

```
$ export LD_LIBRARY_PATH=/users/fred/lib:/users/fred/cs240/lib
$ ./chess
```

Shell Script

```
#!/bin/bash

export LD_LIBRARY_PATH=/users/fred/lib:/users/fred/cs240/lib
./chess
```