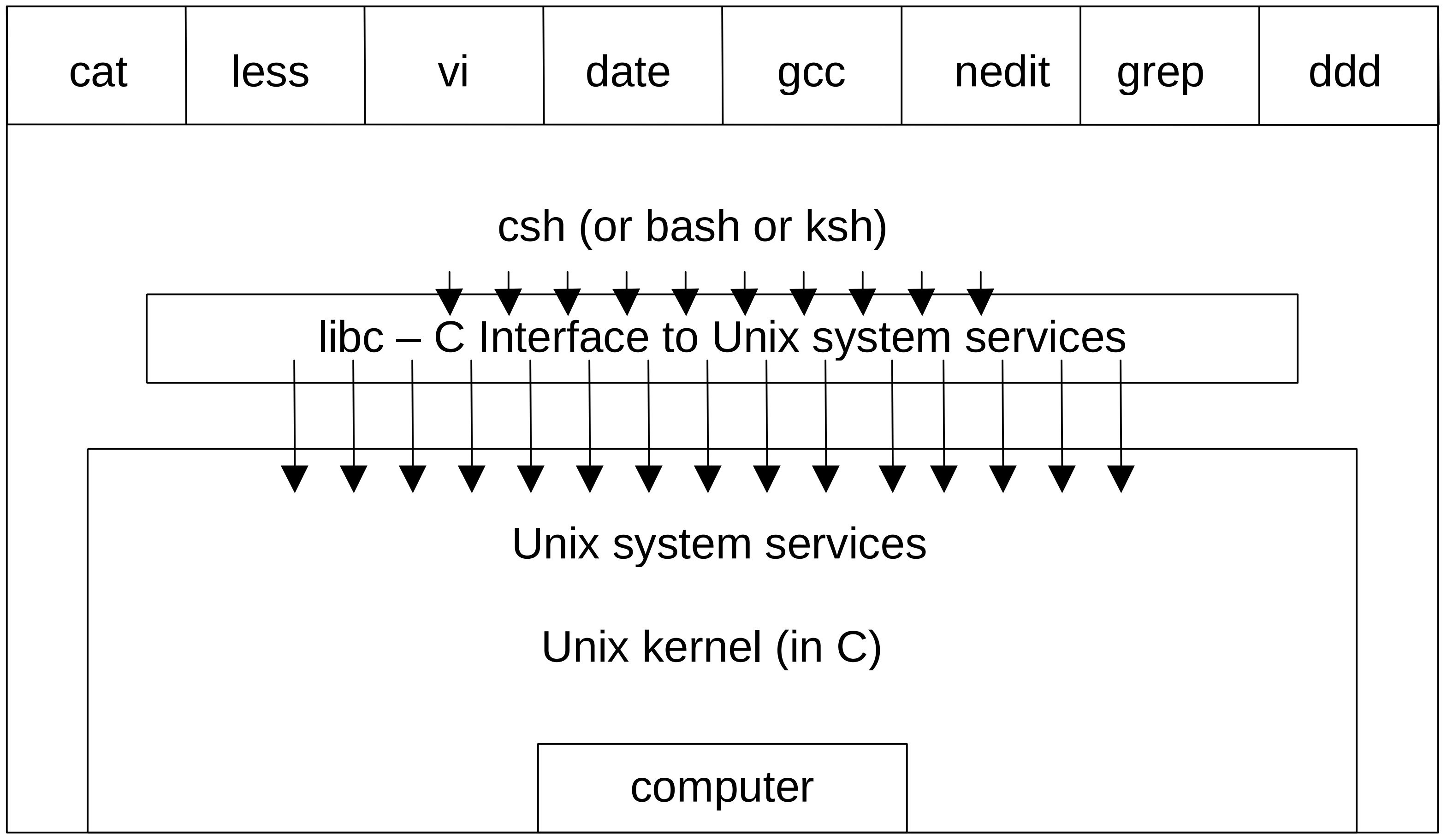
Compilers, Interpreters, Libraries

Comparing compilers and interpreters

Shared vs. non-shared libraries.

Layers of System Software



Compiler vs. Interpreter

- Somehow we need to convert a program into machine code (object code).
- A compiler passes over a whole program before translating it into object code.
- An interpreter reads and executes one line of code at a time.
- An interpreter is a compiled program (often written in C).

C/C++ compiler

- *Preprocessor* does text replacement
  - `#include` replaced by the text of the included file.
  - `#define` macros replaced throughout each file.
- *Compiler* parses the program, performs optimization, and produces assembly code.
- *Assembler* translates assembly code into machine code.
- *Linker* combines object files and libraries into an executable file. It resolves any remaining symbol references.

Java Compiler/Interpreter

- *Compiler* translates program to byte code.
- The *JVM* is a byte code interpreter that translates byte code to machine code.
- Byte codes implement fine grain primitives. They are generic enough that other languages may be compiled to Java byte code.

Shell Interpreter

- The interpreter is a C program!
- The shell interpreter is the program executed when you write

```
#!/bin/sh
```

- Each line of a shell script is input to a C program that parses the line, and determines how to execute it.

Standard Libraries

- System calls are not part of the C language definition.
- System calls are defined in libraries (.a .so)
- Libraries typically contain many .o object files.
- To create your own library archive file:

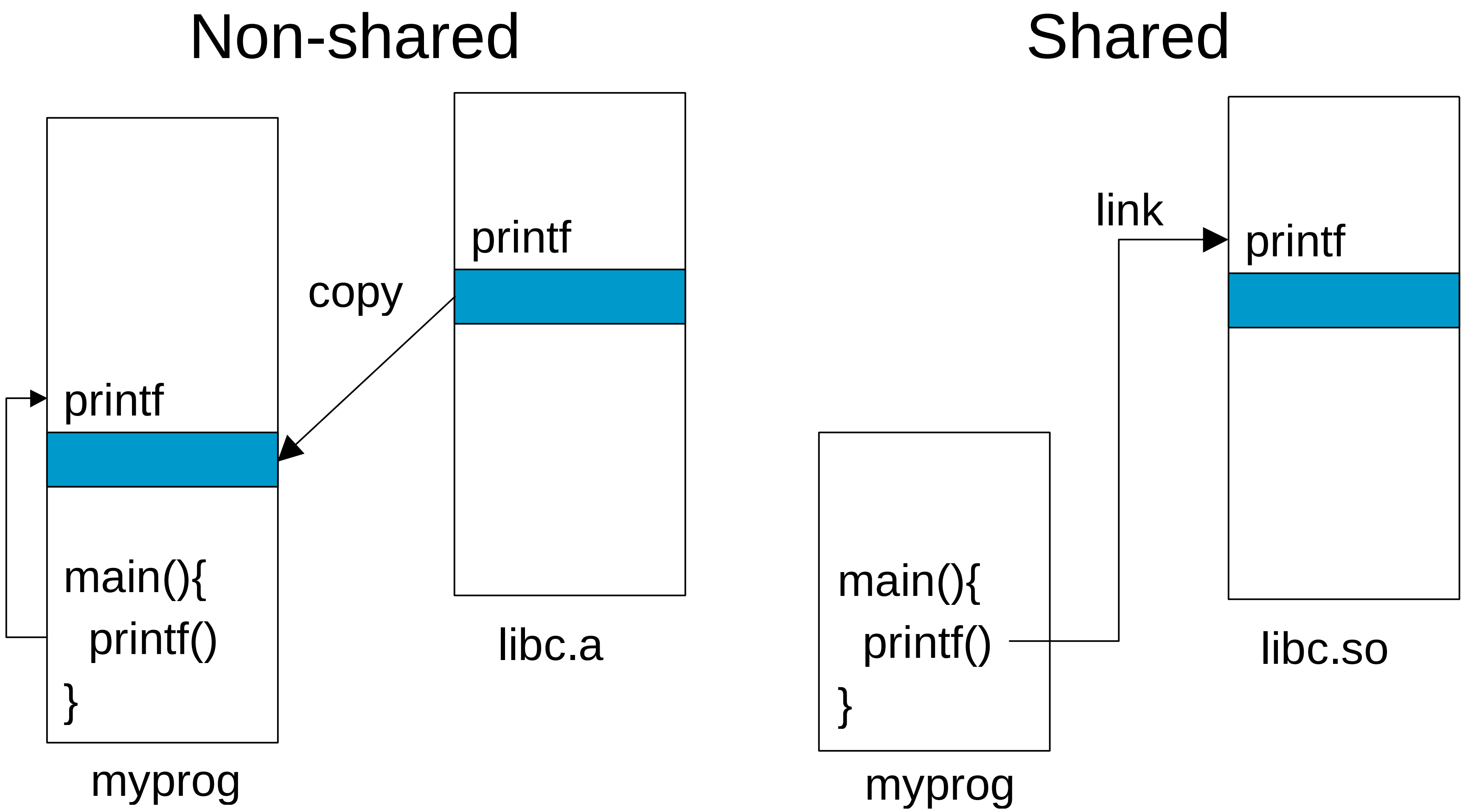
```
ar crv mylib.a *.o
```

- Look in `/usr/lib` and `/usr/local/lib` for system libraries.

Shared Libraries

- `.a` libraries are not shared. The functions used are copied into the executable of your program.
  - size bloat when lots of processes use the same libraries
  - performance and portability are the wins
- `.so` libraries are shared. One copy exists in memory, and all programs using that library link to it to access library functions.
  - reduces total memory usage when multiple processes use the shared library.
  - small performance hit as extra work must be done either when a library function is called, or at the beginning.
  - many tradeoffs and variations between OS's

Shared vs. Non-Shared Libraries



System calls

- Perform a subroutine call directly to the Unix kernel.
- `libc` provides the C interface to system calls
- 4 main categories
  - File management
  - Process management
  - Communication
  - Error handling

Executing a Program

- A special start-up routine (`crt0`) is always linked in with your program.
- This routine reads the arguments and calls main.
- The `libc` library is automatically linked into your program, which is how you have access to many C functions (`printf`, `open`, etc.)
- Your program also calls special functions on exit that close file descriptors and clean up other resources.