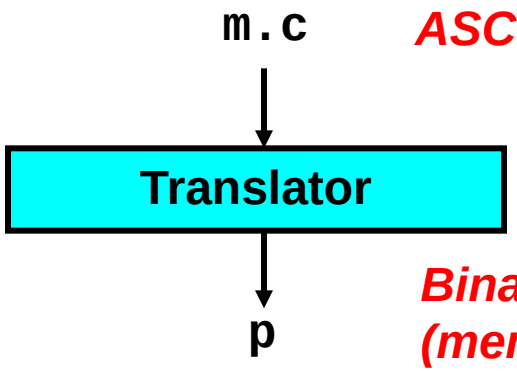
What software engineers need to know about linking and a few things about execution

(Extract from the slides by
Terrance E. Boult
http://vast.uccs.edu/~tboult/)

A Simplistic Program Translation Scheme



ASCII source file

Binary executable object file (memory image on disk)

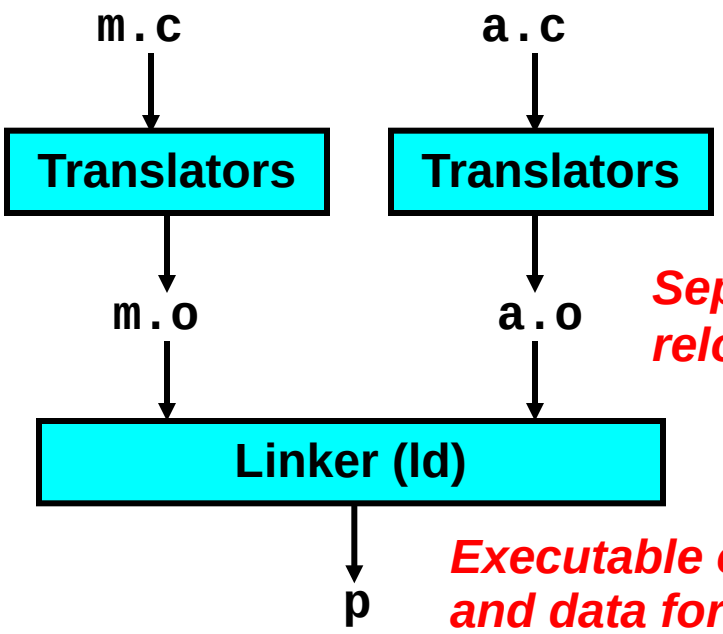
Problems:

- **Efficiency: small change requires complete recompilation**
- **Modularity: hard to share common functions (e.g. `printf`)**

Solution:

- ***Static linker (or linker)***

A Better Scheme Using a Linker



Separately compiled relocatable object files

Executable object file (contains code and data for all functions defined in `m.c` and `a.c`)

Translating the Example Program

- *Compiler driver* coordinates all steps in the translation and linking process.
  - Typically included with each compilation system (e.g., `gcc`)
  - Invokes preprocessor (`cpp`), compiler (`cc1`), assembler (`as`), and linker (`ld`).
  - Passes command line arguments to appropriate phases
- Example: create executable `p` from `m.c` and `a.c`:

```
bass> gcc -O2 -v -o p m.c a.c
cpp [args] m.c /tmp/cca07630.i
cc1 /tmp/cca07630.i m.c -O2 [args] -o /tmp/cca07630.s
as [args] -o /tmp/cca076301.o /tmp/cca07630.s
<similar process for a.c>
ld -o p [system obj files] /tmp/cca076301.o /tmp/cca076302.o
bass>
```

What Does a Linker Do?

- Merges object files
  - Merges multiple relocatable (`.o`) object files into a single executable object file that can loaded and executed by the loader.
- Resolves external references
  - As part of the merging process, resolves external references.
    - *External reference*: reference to a symbol defined in another object file.
- Relocates symbols
  - Relocates symbols from their relative locations in the `.o` files to new absolute positions in the executable.
  - Updates all references to these symbols to reflect their new positions.
    - References can be in either code or data
      - code: `a();  /* reference to symbol a */`
      - data: `int *xp=&x;  /* reference to symbol x */`

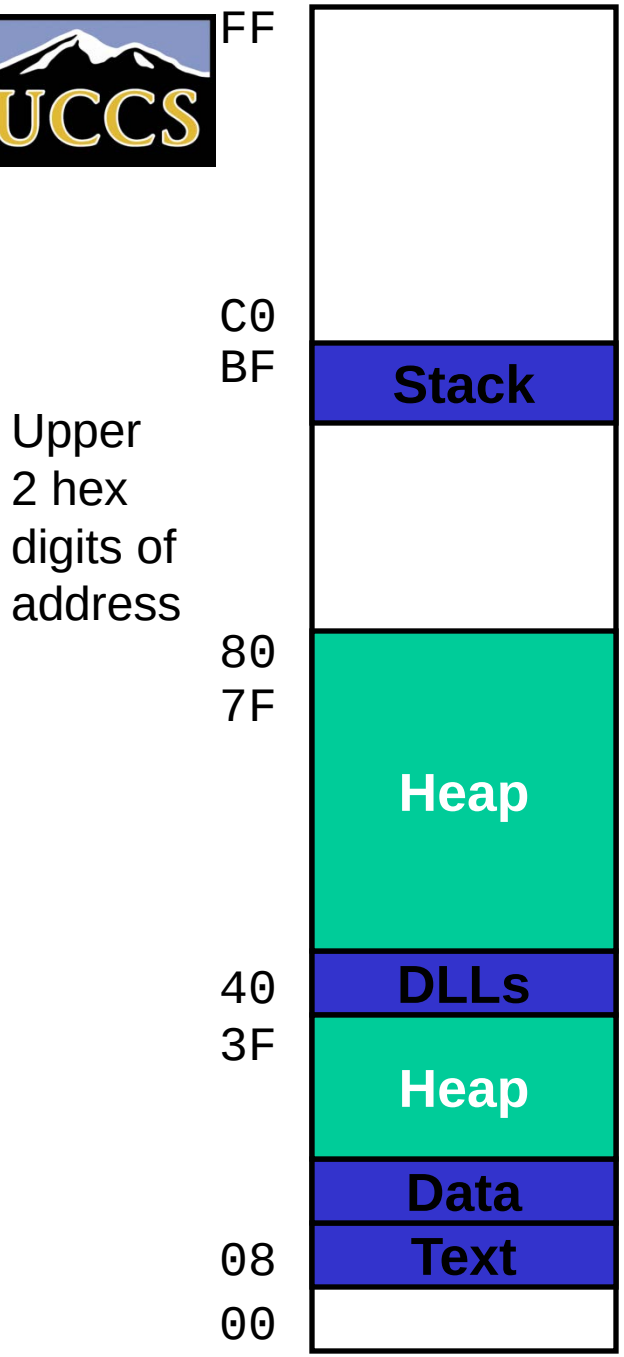
Why Linkers?

- Modularity
  - Program can be written as a collection of smaller source files, rather than one monolithic mass.
  - Can build libraries of common functions (more on this later)
    - e.g., Math library, standard C library
- Efficiency
  - Time:
    - Change one source file, compile, and then relink.
    - No need to recompile other source files.
  - Space:
    - Libraries of common functions can be aggregated into a single file...
    - Yet executable files and running memory images contain only code for the functions they actually use.

Executable and Linkable Format (ELF)

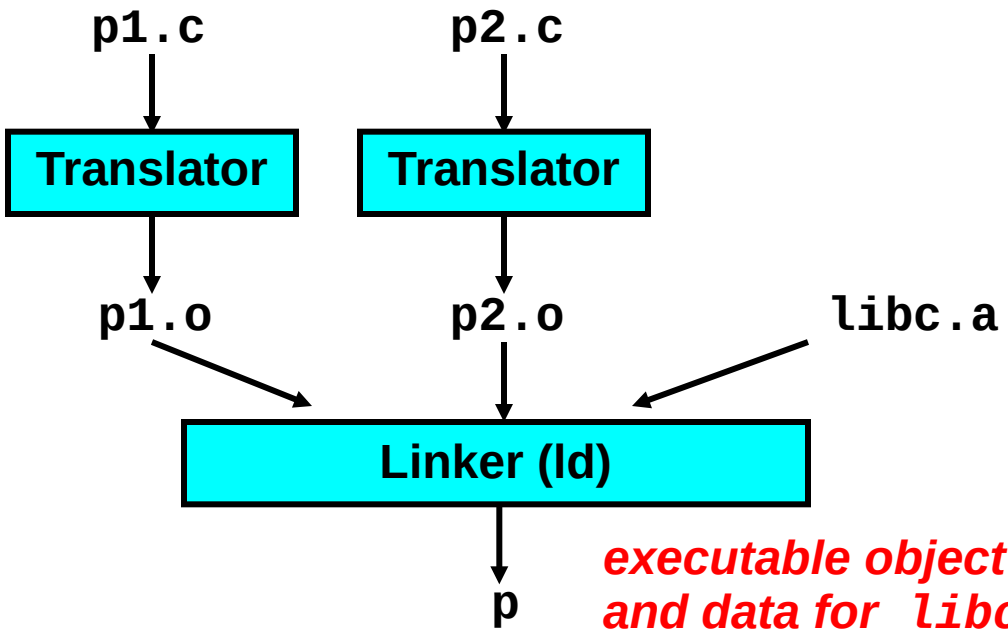
- Standard binary format for object files
- Derives from AT&T System V Unix
  - Later adopted by BSD Unix variants and Linux
- One unified format for
  - Relocatable object files (`.o`),
  - Executable object files
  - Shared object files (`.so`)
- Generic name: ELF binaries
- Better support for shared libraries than old `a.out` formats.

Linux Memory Layout



- Stack
  - Runtime stack
- Heap
  - Dynamically allocated storage
  - When call `malloc`, `calloc`, `new`
- DLLs
  - Dynamically Linked Libraries
  - Library routines (e.g., `printf`, `malloc`)
  - Linked into object code when first executed
- Data
  - Statically allocated data
  - E.g., arrays & strings declared in code
- Text
  - Executable machine instructions
  - Read-only

Static Libraries (archives)



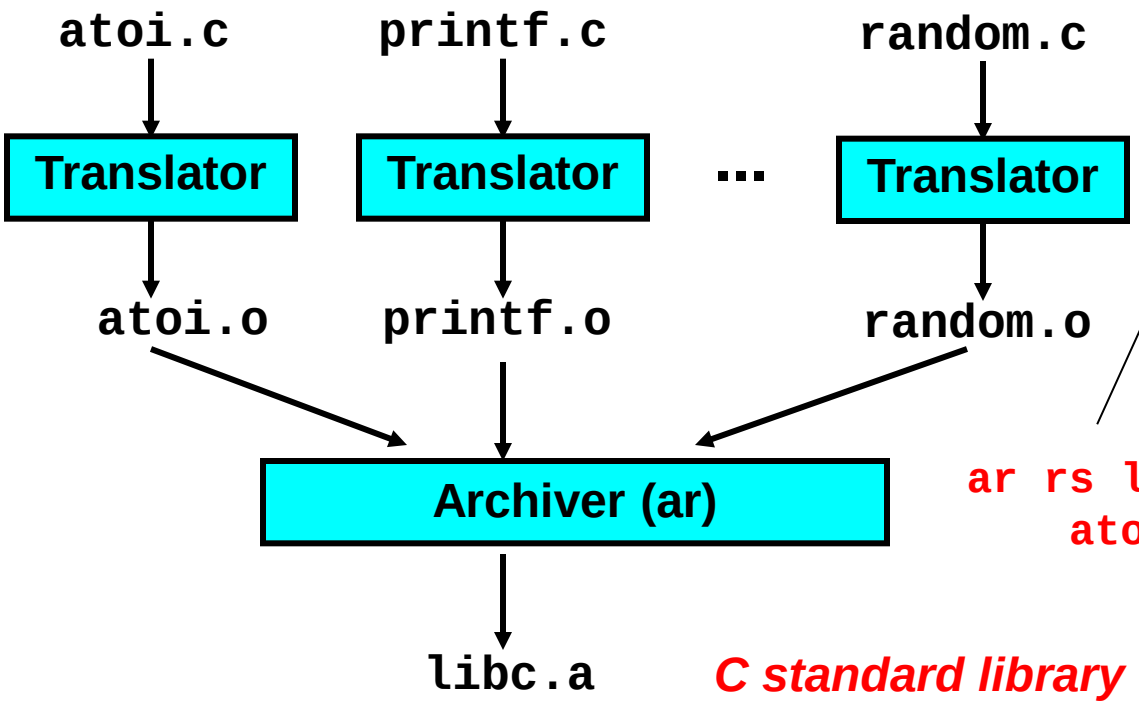
static library (archive) of relocatable object files concatenated into one file.

executable object file (only contains code and data for `libc` functions that are called from `p1.c` and `p2.c`)

Further improves modularity and efficiency by packaging commonly used functions [e.g., C standard library (`libc`), math library (`libm`)]

Linker selectively links only the `.o` files in the archive that are actually needed by the program.

Creating Static Libraries



r: replace existing or insert new file(s) into the archive
s: create an archive index

```
ar rs libc.a \
    atoi.o printf.o … random.o
```

Archiver allows incremental updates:

- **Recompile function that changes and replace .o file in archive.**

Using Static Libraries

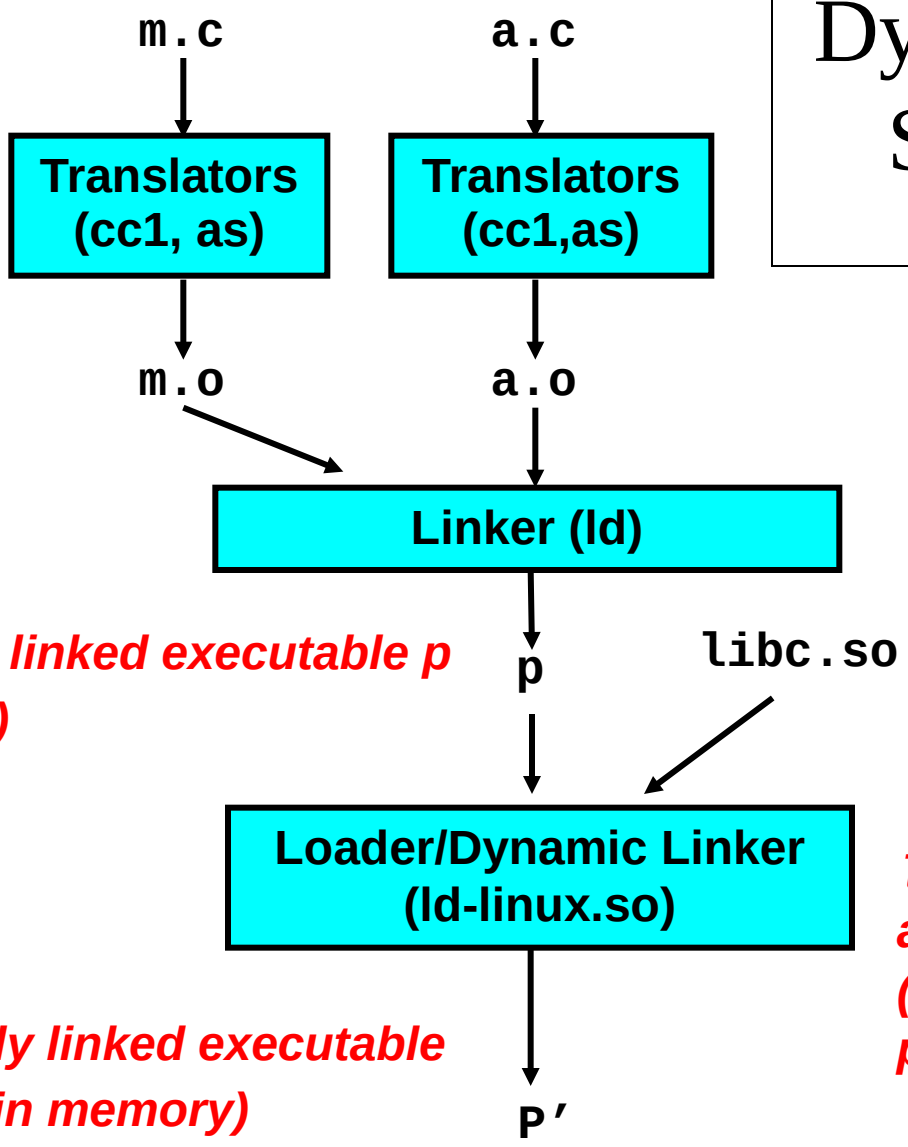
- Linker's algorithm for resolving external references:
  - Scan .o files and .a files in the command line order.
  - During the scan, keep a list of the current unresolved references.
  - As each new .o or .a file obj is encountered, try to resolve each unresolved reference in the list against the symbols in obj.
  - If any entries in the unresolved list at end of scan, then error.
- Problem:
  - Command line order matters!
  - Moral: put libraries at the end of the command line.

```
bass> gcc -L. libtest.o -lmine
bass> gcc -L. -lmine libtest.o
libtest.o: In function `main':
libtest.o(.text+0x4): undefined reference to `libfun'
```

Shared Libraries

- Static libraries have the following disadvantages:
  - Potential for duplicating lots of common code in the executable files on a filesystem.
    - e.g., every C program needs the standard C library
  - Potential for duplicating lots of code in the virtual memory space of many processes.
  - Minor bug fixes of system libraries require each application to explicitly relink
- Solution:
  - *Shared libraries* (dynamic link libraries, DLLs) whose members are dynamically loaded into memory and linked into an application at run-time.
    - Dynamic linking can occur when executable is first loaded and run.
      - Common case for Linux, handled automatically by `ld-linux.so`.
    - Dynamic linking can also occur after program has begun.
      - In Linux, this is done explicitly by user with `dlopen()`.
      - Basis for High-Performance Web Servers.
    - Shared library routines can be shared by multiple processes.

Dynamically Linked Shared Libraries



Partially linked executable p (on disk)

Shared library of dynamically relocatable object files

`libc.so` functions called by `m.c` and `a.c` are loaded, linked, and (potentially) shared among processes.

Fully linked executable p' (in memory)

The Complete Picture