Advanced Programming in the UNIX Environment

Week 11, Segment 2:
Of Linkers and Loaders

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

```
$ readelf -l a.out

Elf file type is EXEC (Executable file)
Entry point 0x400570
There are 7 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x0000000000000188 0x0000000000000188  R      0x8
  INTERP         0x00000000000001c8 0x00000000004001c8 0x00000000004001c8
                 0x0000000000000017 0x0000000000000017  R      0x1
      [Requesting program interpreter: /usr/libexec/ld.elf_so]
```

Linking and Loading

Let’s revisit our lecture on Unix tools and the compile chain:

Linking and Loading

Let's revisit our lecture on Unix tools and the compile chain:

The compiler chain usually performs preprocessing (*e.g.*, via `cpp(1)`)

```
$ cpp crypt.c crypt.i
```

Linking and Loading

Let's revisit our lecture on Unix tools and the compile chain:

The compiler chain usually performs preprocessing (*e.g.*, via `cpp(1)`), compilation (`cc(1)`)

```
$ cpp crypt.c crypt.i
$ cc -S crypt.i
```

Linking and Loading

Let's revisit our lecture on Unix tools and the compile chain:

The compiler chain usually performs preprocessing (*e.g.*, via `cpp(1)`), compilation (`cc(1)`), assembly (`as(1)`)

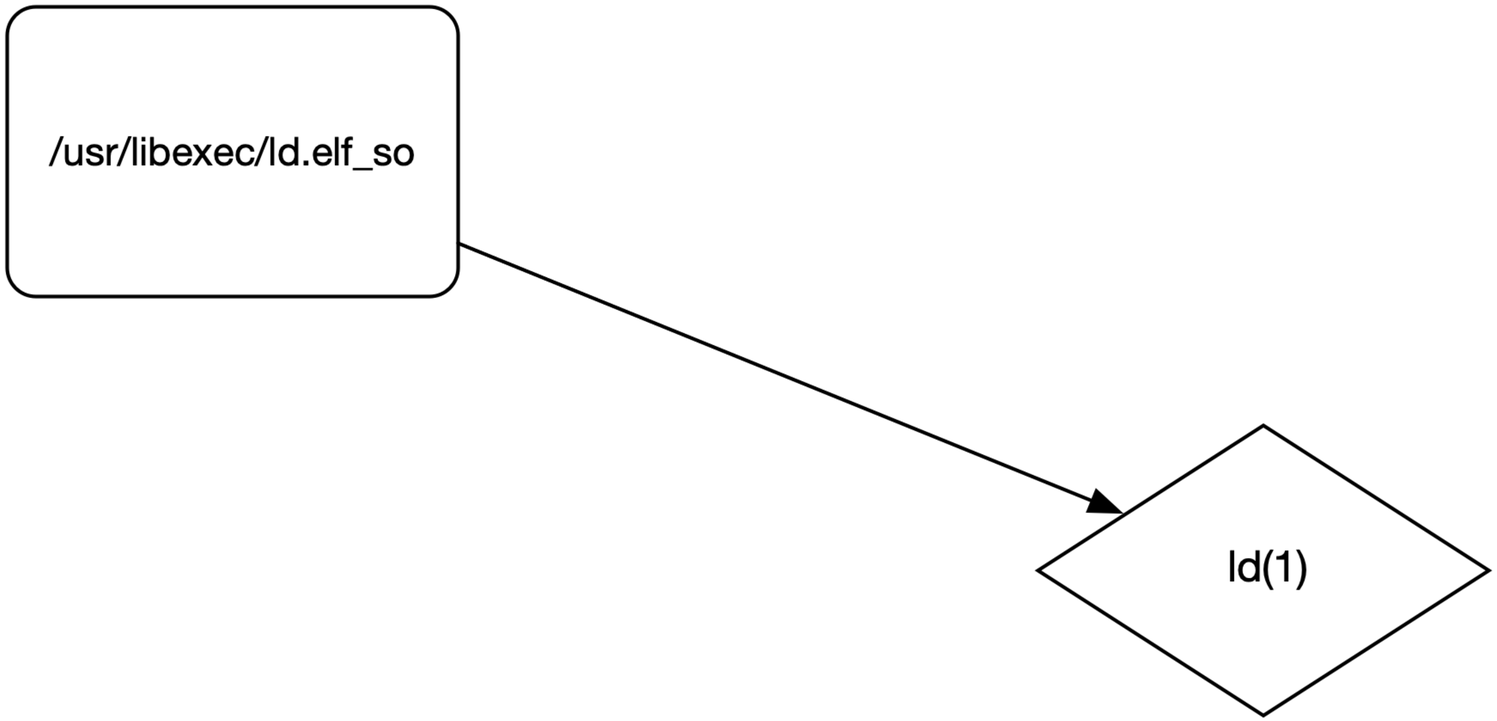
```
$ cpp crypt.c crypt.i
$ cc -S crypt.i
$ as -o crypt.o crypt.s
```

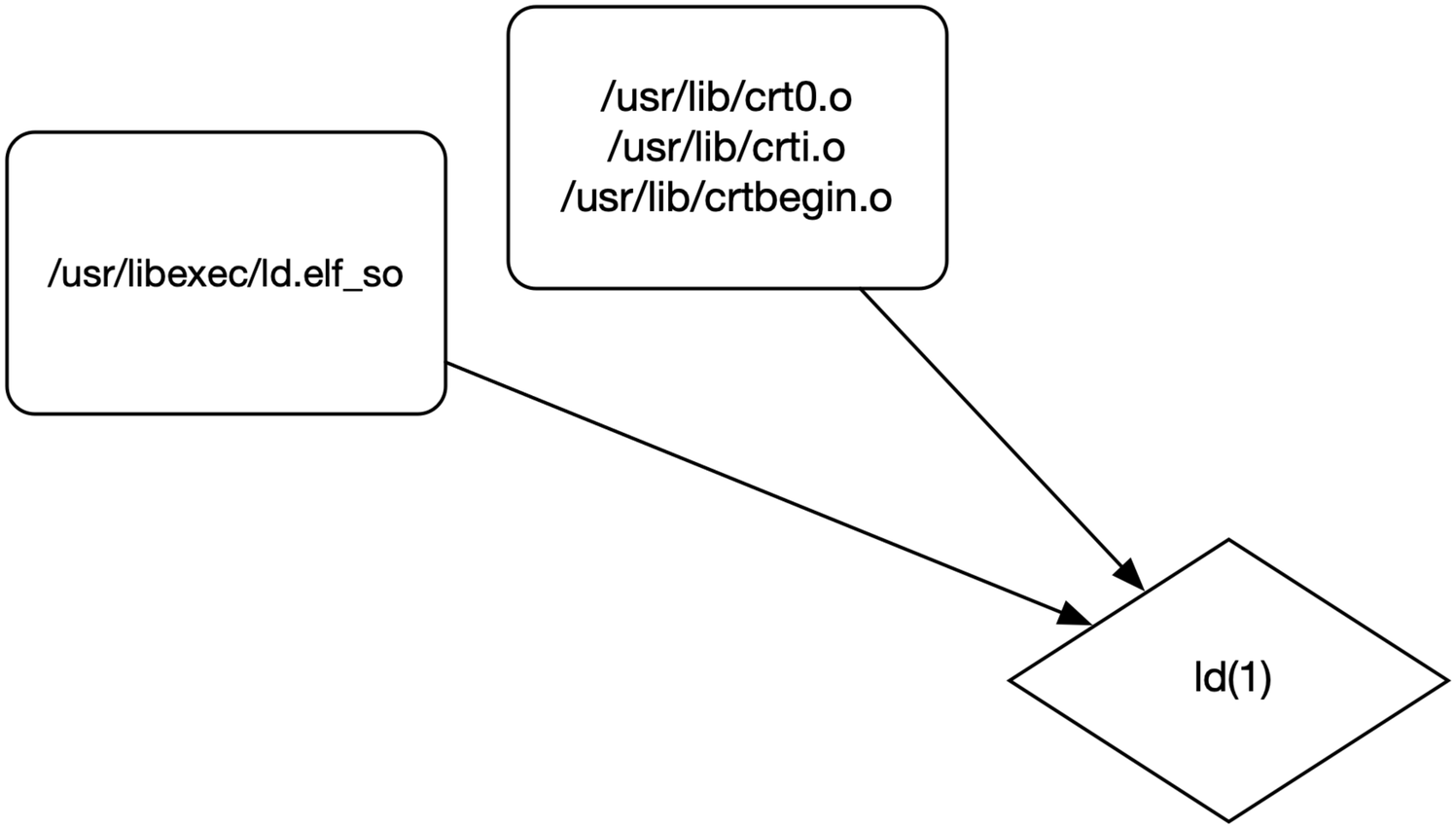
Linking and Loading

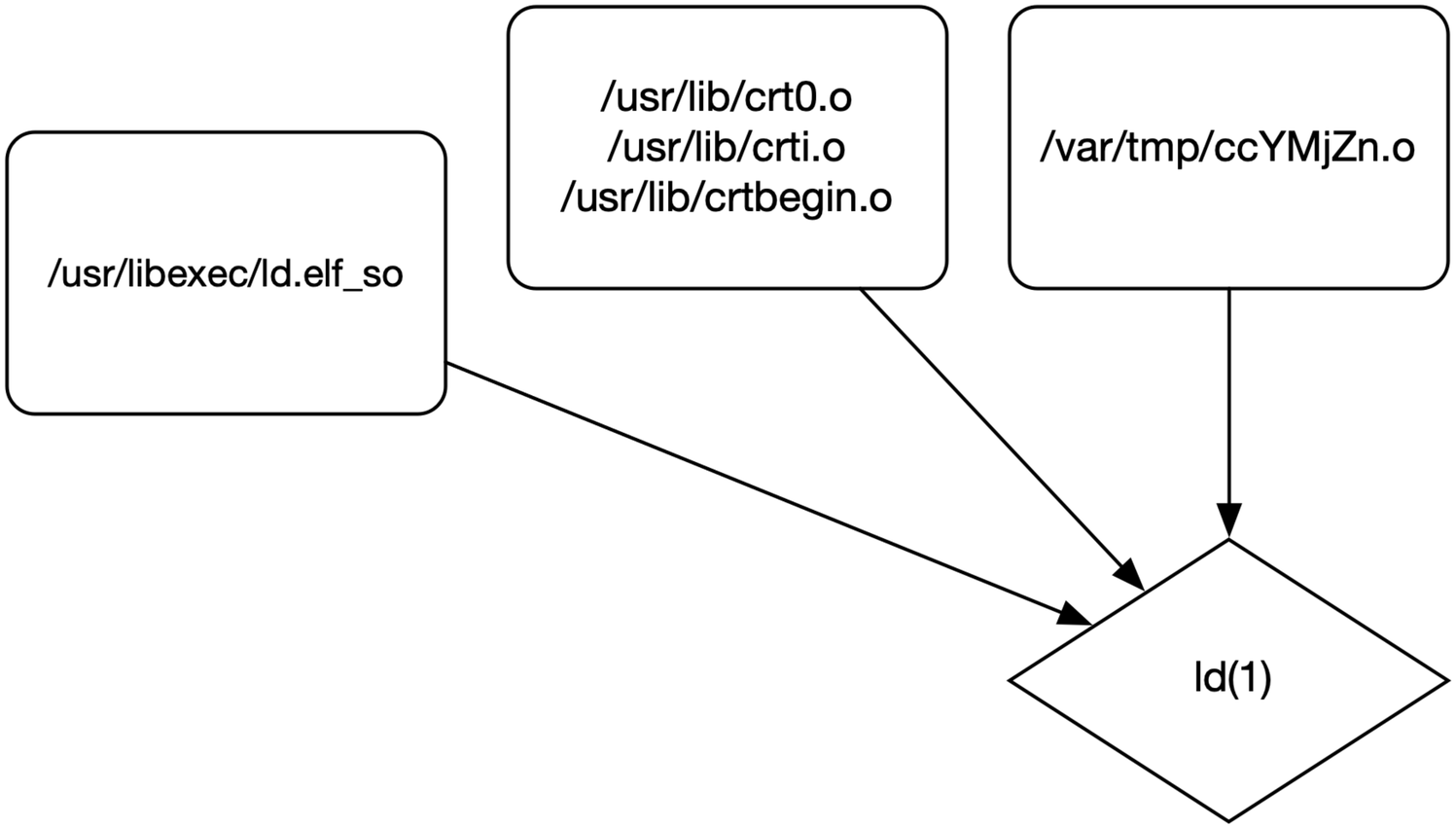
Let's revisit our lecture on Unix tools and the compile chain:

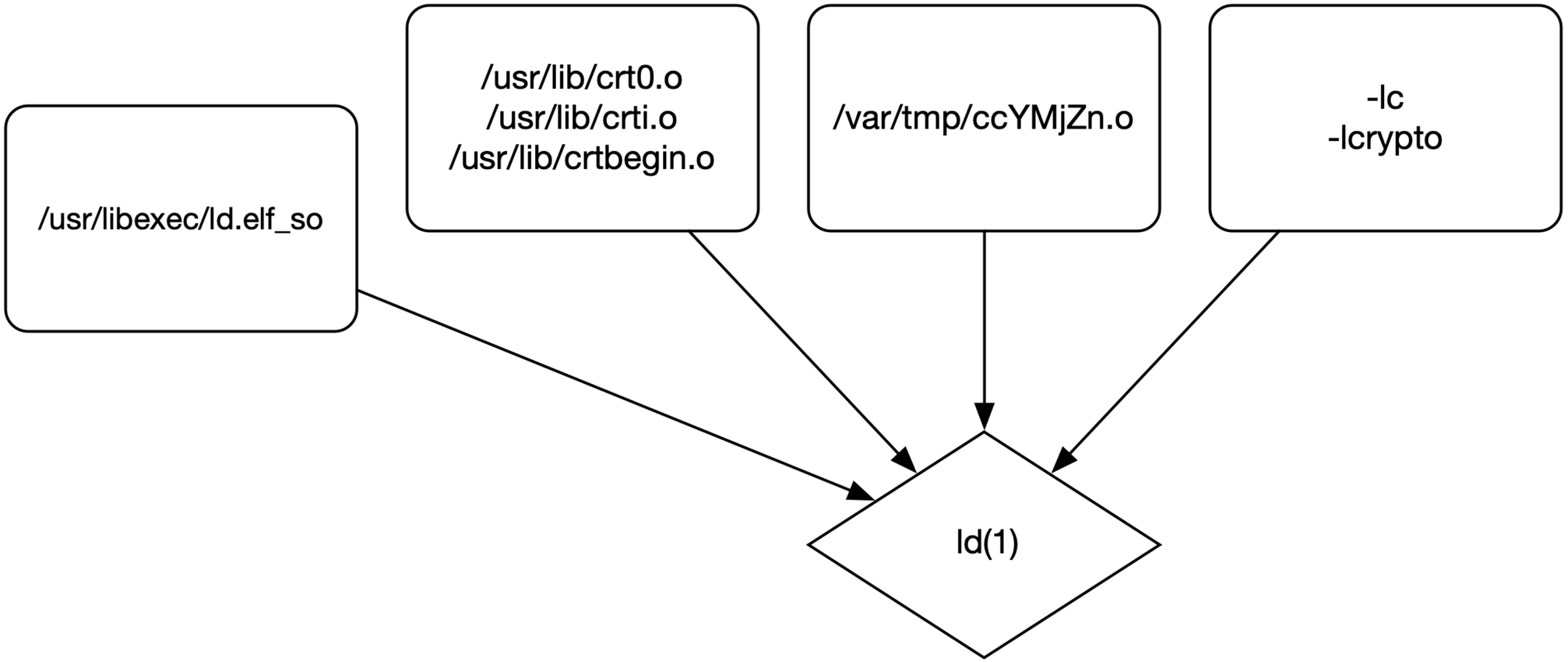
The compiler chain usually performs preprocessing (*e.g.*, via `cpp(1)`), compilation (`cc(1)`), assembly (`as(1)`) and linking (`ld(1)`).

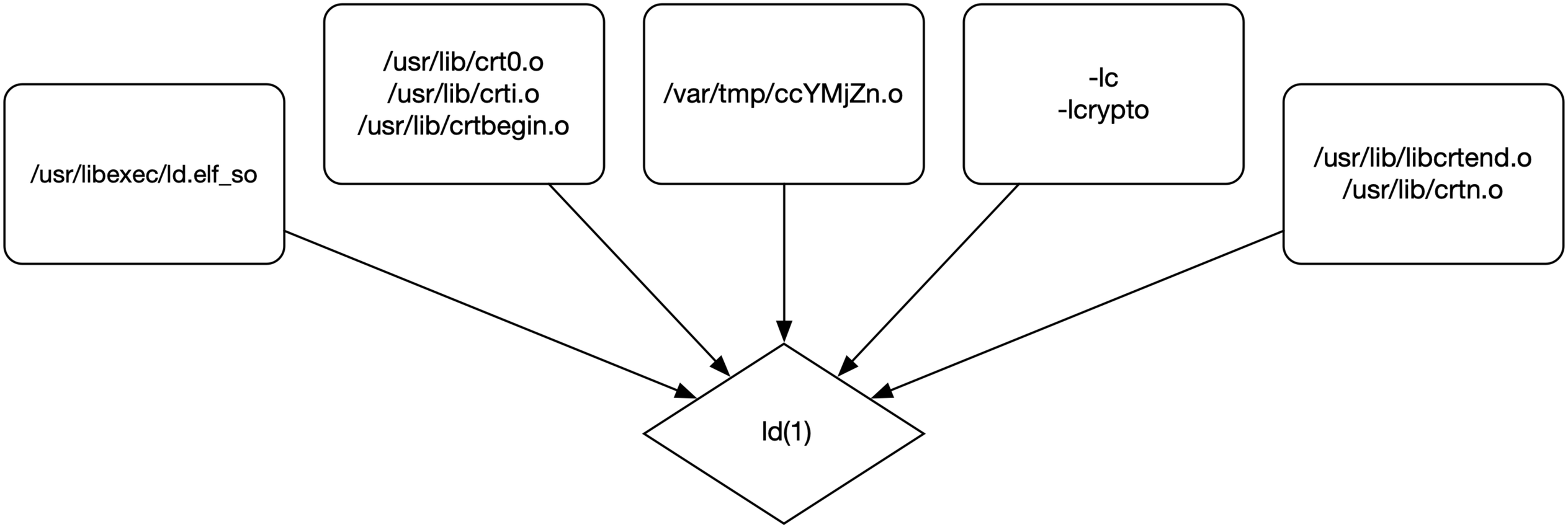
```
$ cpp crypt.c crypt.i
$ cc -S crypt.i
$ as -o crypt.o crypt.s
$ ld -dynamic-linker /usr/libexec/ld.elf_so                    \
        /usr/lib/crt0.o /usr/lib/crti.o /usr/lib/crtbegin.o \
        crypt.o -lcrypto —lc /usr/lib/crtend.o /usr/lib/crtn.o
```

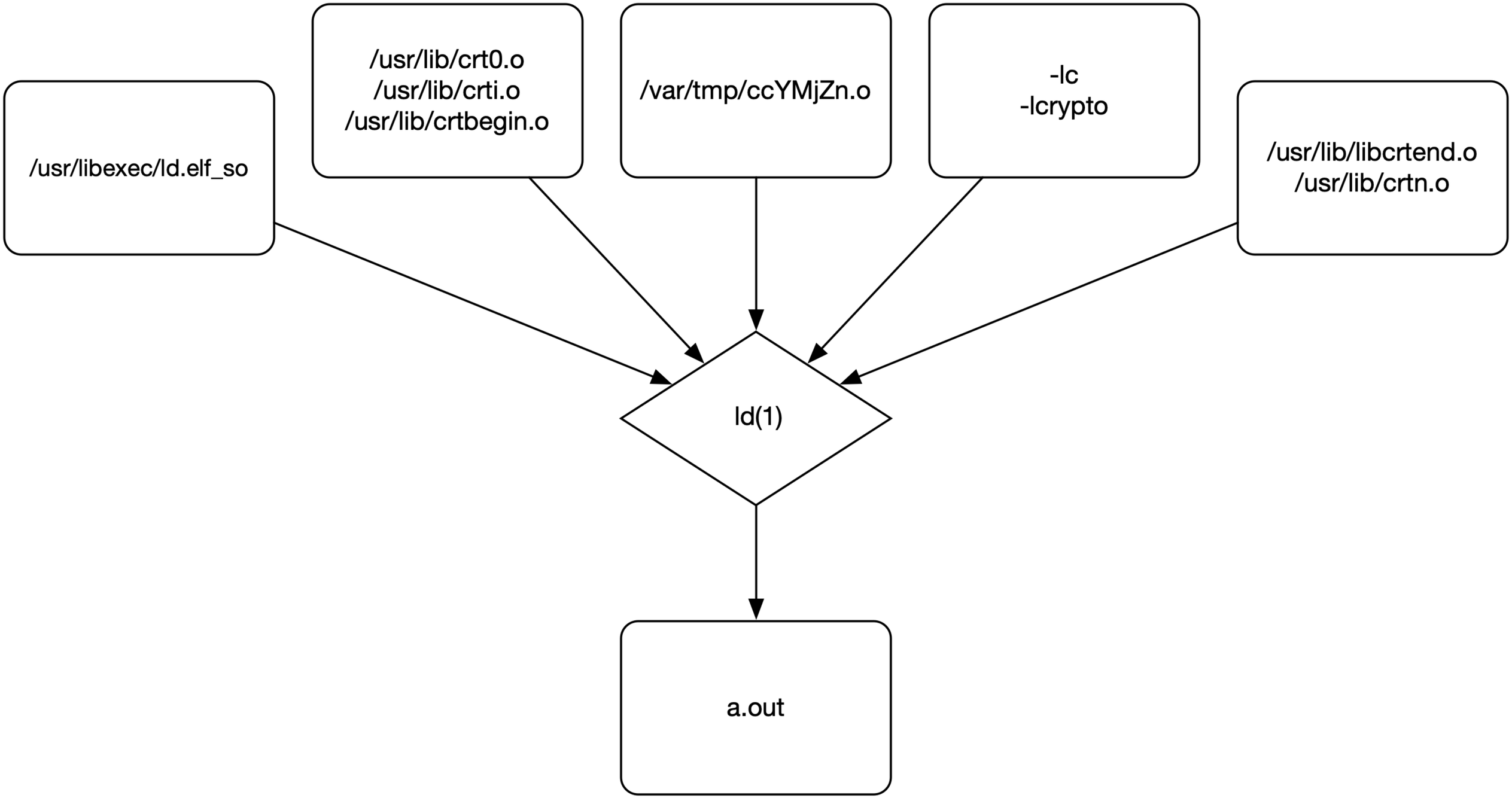

/usr/libexec/ld.elf_so
ld(1)


/usr/lib/crt0.o
/usr/lib/crti.o
/usr/lib/crtbegin.o
/usr/libexec/ld.elf_so
ld(1)

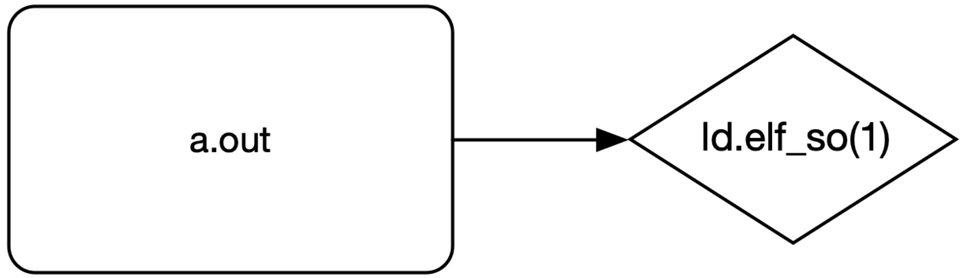

/usr/libexec/ld.elf_so
/usr/lib/crt0.o
/usr/lib/crti.o
/usr/lib/crtbegin.o
/var/tmp/ccYMjZn.o
ld(1)

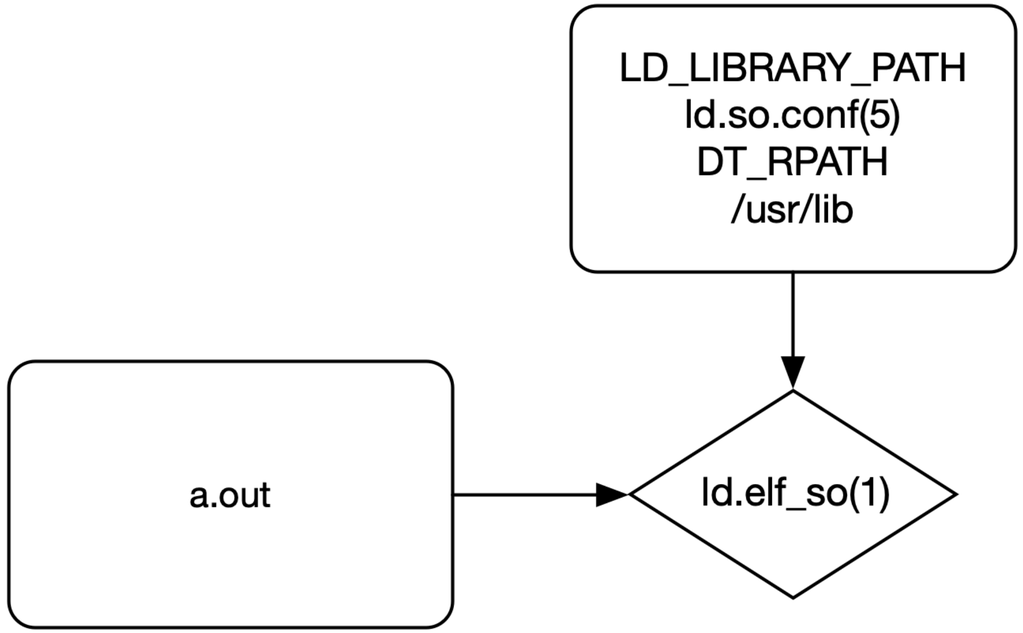

/usr/libexec/ld.elf_so
/usr/lib/crt0.o
/usr/lib/crti.o
/usr/lib/crtbegin.o
/var/tmp/ccYMjZn.o
-lc
-lcrypto
ld(1)

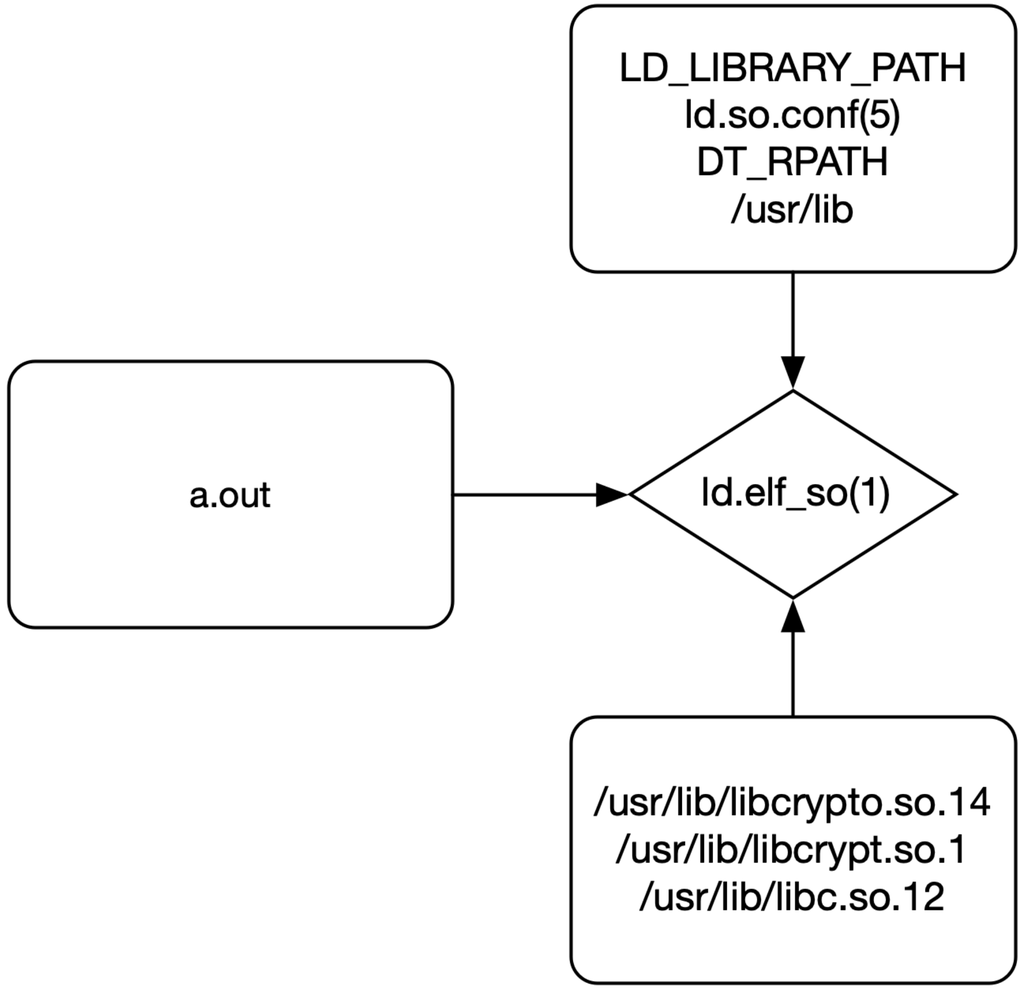

/usr/libexec/ld.elf_so
/usr/lib/crt0.o
/usr/lib/crti.o
/usr/lib/crtbegin.o
/var/tmp/ccYMjZn.o
-lc
-lcrypto
/usr/lib/libcrtend.o
/usr/lib/crtn.o
ld(1)

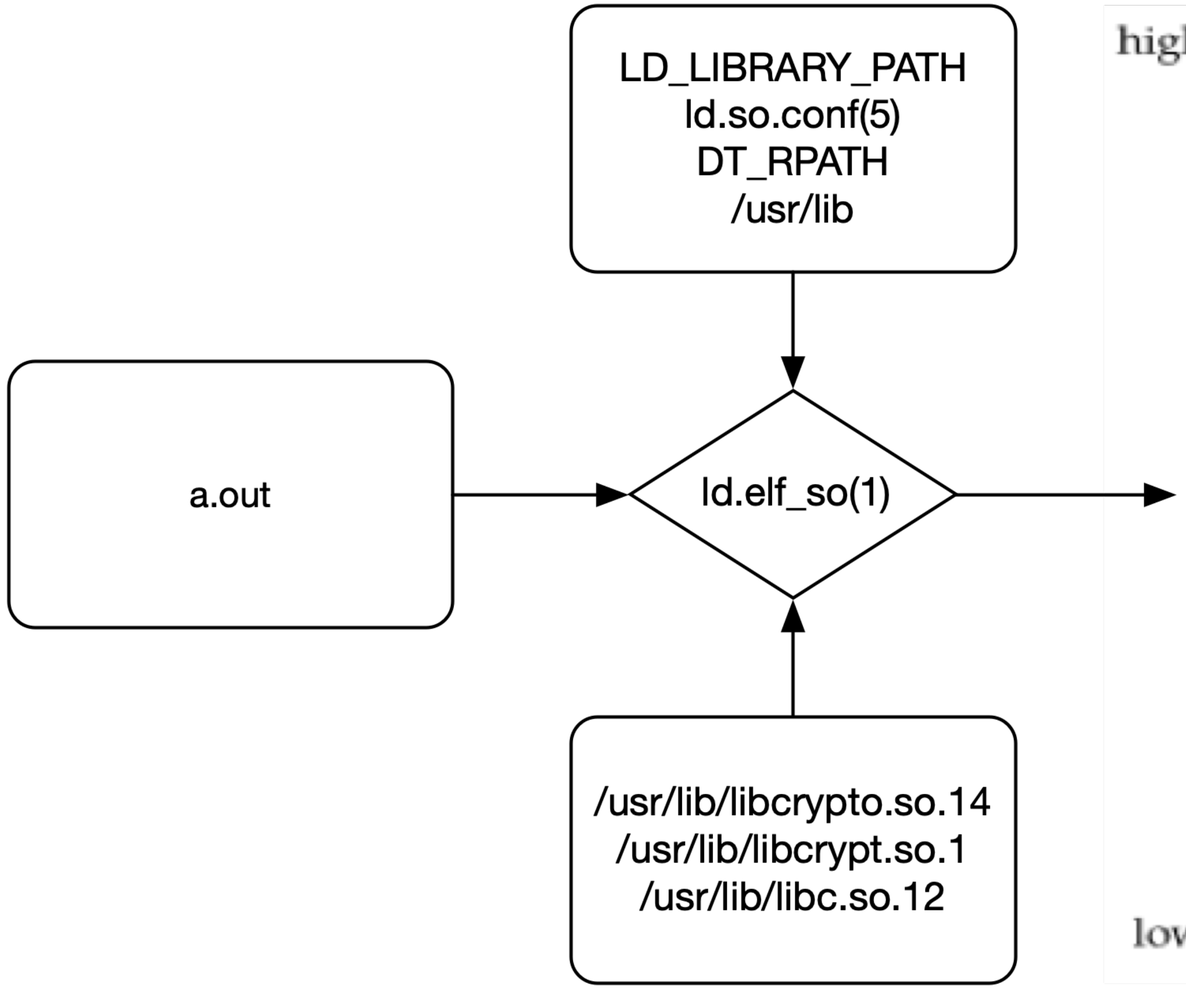

/usr/libexec/ld.elf_so
/usr/lib/crt0.o
/usr/lib/crti.o
/usr/lib/crtbegin.o
/var/tmp/ccYMjZn.o
-lc
-lcrypto
/usr/lib/libcrtend.o
/usr/lib/crtn.o
ld(1)
a.out

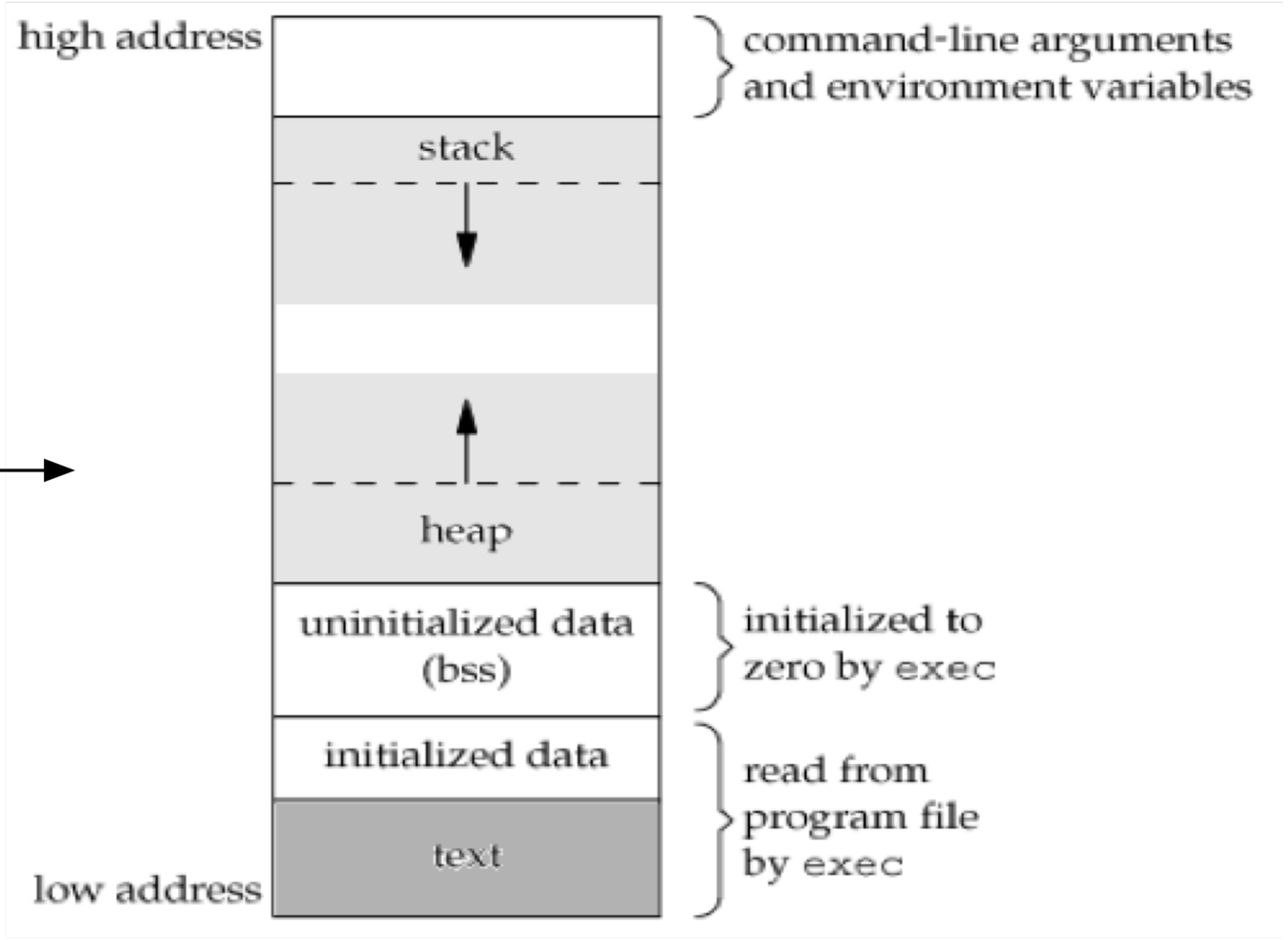
```
                 0x0000000000000190 0x0000000000000190  RW     0x8
  NOTE           0x0000000000000158 0x0000000000400158 0x0000000000400158
                 0x000000000000002c 0x000000000000002c  R      0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .note.netbsd.ident .note.netbsd.pax .hash .dynsym .dynstr .rela.dyn .r
jschauma@apue$ ktrace -i ./a.out foo
[2]   Segmentation fault (core dumped) ktrace -i ./a.out foo
jschauma@apue$ kdump
  1493      1 ktrace   EMUL  "netbsd"
  1493      1 ktrace   CALL  execve(0x7f7fff1378ba,0x7f7fff137318,0x7f7fff137330
)
  1493      1 ktrace   NAMI  "/home/jschauma/11/./a.out"
  1493      1 a.out    EMUL  "netbsd"
  1493      1 a.out    RET   execve JUSTRETURN
  1493      1 a.out    PSIG  SIGSEGV SIG_DFL: code=SEGV_MAPERR, addr=0x0, trap=6
)
  1493      1 a.out    NAMI  "a.out.core"
jschauma@apue$ ld -dynamic-linker /usr/libexec/ld.elf_so /usr/libexec/ld.elf_so
/usr/lib/crt0.o /usr/lib/crti.o crypt.o -lc -lcrypto /usr/lib/crtn.o
jschauma@apue$ ./a.out foo
$1$$n1rTiFE0nRifwV/43bVon/
```


a.out
ld.elf_so(1)


LD_LIBRARY_PATH
ld.so.conf(5)
DT_RPATH
/usr/lib
a.out
ld.elf_so(1)


LD_LIBRARY_PATH
ld.so.conf(5)
DT_RPATH
/usr/lib
a.out
ld.elf_so(1)
/usr/lib/libcrypto.so.14
/usr/lib/libcrypt.so.1
/usr/lib/libc.so.12


LD_LIBRARY_PATH
ld.so.conf(5)
DT_RPATH
/usr/lib
a.out
ld.elf_so(1)
/usr/lib/libcrypto.so.14
/usr/lib/libcrypt.so.1
/usr/lib/libc.so.12


high address
command-line arguments
and environment variables
stack
heap
uninitialized data
(bss)
initialized to
zero by exec
initialized data
text
read from
program file
by exec
low address

Summary

- A linker takes multiple *object files*, resolves symbols to *e.g.*, addresses in *libraries* (possibly relocating them in the process), and produces an *executable*.
- A loader copies a program into main memory, possibly invoking the *dynamic linker* or *run-time link editor* to find the right libraries, resolve addresses of symbols, and relocate them.

- On some systems, the run-time link-editor is itself an *executable*, allowing you to invoke it directly and passing another executable file as an argument. Try it out!

```
$ ld -no-dynamic-linker … file.o …
$ /lib64/ld-linux-x86-64.so ./a.out
```