CS 3214 lecture #9 "building an executable"

linking & loading

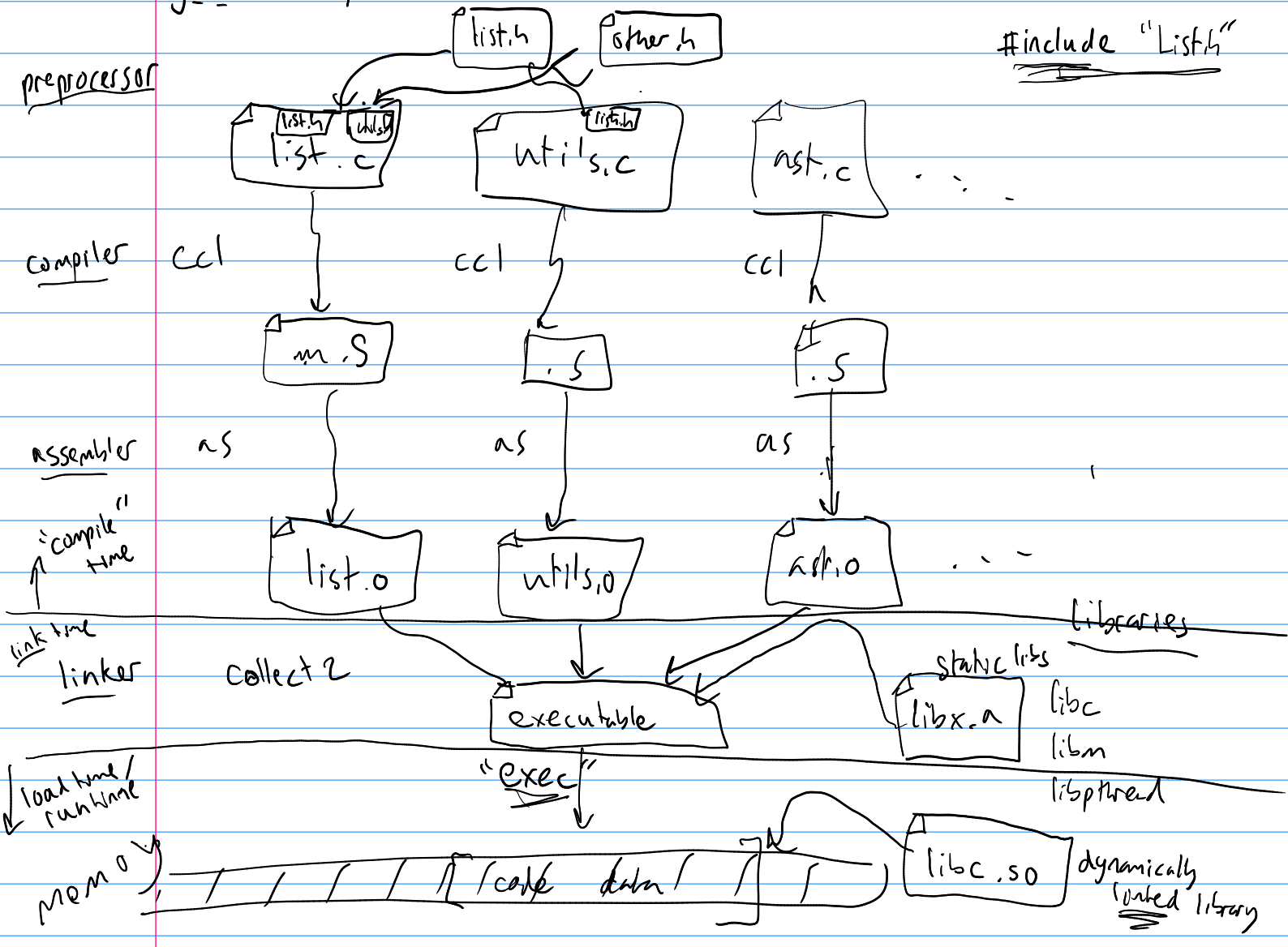
C code { .c, .h } → symbols

"compiler": gcc
- compiler
- linker
- assembler
- preprocessor, ...

{ addresses, constants, no symbols } machine code

symbol resolution

gcc: "compiler driver"



fn names
variable names
globals, ...

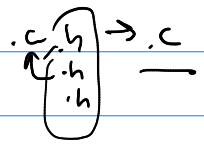
addresses
constants

Back to how do (symbols) get (resolved)?

1. C preprocessor
   - textual insertion
   - text
   TEXT
   gcc -E

   #include "header.h"
   #define GNU_SOURCE
   # define ADD_ONE(x) ((x)+1)
   # define SHIFT(x) ((x)<<16)
   #pragma once
   #ifdef GNU_SOURCE
   #ifndef
   #else

   #define MAXINT
   #define BUFLEN
   # define READ_END 0

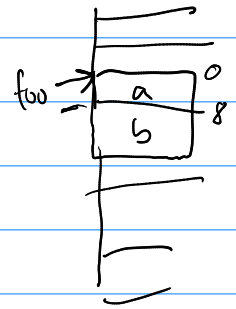


2. Compiler .c → .s

lang → AST → opt → code gen

resolved!
- local variables - registers / stack
- field names in structs
- function parameters — System V calling convention

```
struct foo {
    int a;
    int b;
};
```



Still some placeholders (especially for globals & fn names)

3. assembler .s → .o
   resolves some labels for relative branches

T1: mov %r2, ...
    jmp T2
T2:

4. Linker .o → executable
   - create an in-memory layout for process code & data
   - resolve references & fill in placeholders

ELF

ELF, Mach-O, PE, a.out, .exe

ELF: - all information to load into address space (for execution)
     - enough info for the next tool