# From Source to Execution

CSE 410, Spring 2004

Computer Systems

http://www.cs.washington.edu/education/courses/410/04sp/

# Readings and References

- Reading
  - » Sections 3.9, A.1 through A.4, *Computer Organization & Design, Patterson and Hennessy*

# Starting a Program

- Two phases from source code to execution
- Build time
  - » compiler creates assembly code
  - » assembler creates machine code
  - » linker creates an executable
- Run time
  - » loader moves the executable into memory and starts the program

# Build Time

- You're experts on compiling from source to assembly and hand crafted assembly
- Two parts to translating from assembly to machine language:
  - » Instruction encoding (including translating pseudoinstructions)
  - » Translating labels to addresses
- Label translations go in the *symbol table*

# Symbol Table

- Symbols are **names** of global variables or labels (including procedure entry points)
- Symbol table associates **symbols** with their **addresses** in the object file
- This allows files compiled separately to be linked

| | |
|---|---|
| `LabelA:` | `0x01031ff0` |
| `bigArray` | `0x10006000` |

# Modular Program Design

- Small projects might use only one file
  - » Any time any one line changes, recompile and reassemble the whole thing
- For larger projects, recompilation time and complexity management is significant
- Solution: split project into modules
  - » compile and assemble modules separately
  - » link the object files

# The Compiler + Assembler

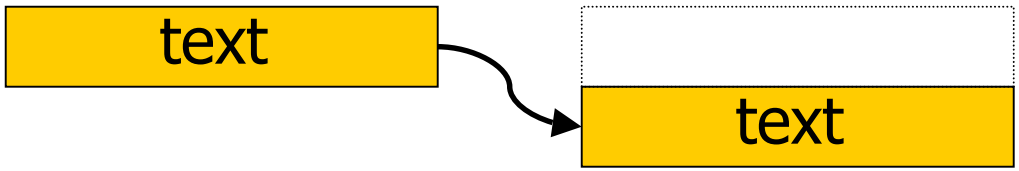
- Translate source files to object files
- Object files
  - » Contain machine instructions (1's & 0's)
  - » Bookkeeping information
    - Procedures and variables the object file defines
    - Procedures and variables the source files use but are undefined (unresolved references)
    - Debugging information associating machine instructions with lines of source code

# The Linker

- The linker's job is to "stitch together" the object files:
  1. Place the data modules in memory space
  2. Determine the addresses of data and labels
  3. Match up references between modules
- Creates an executable file

# Determining Addresses

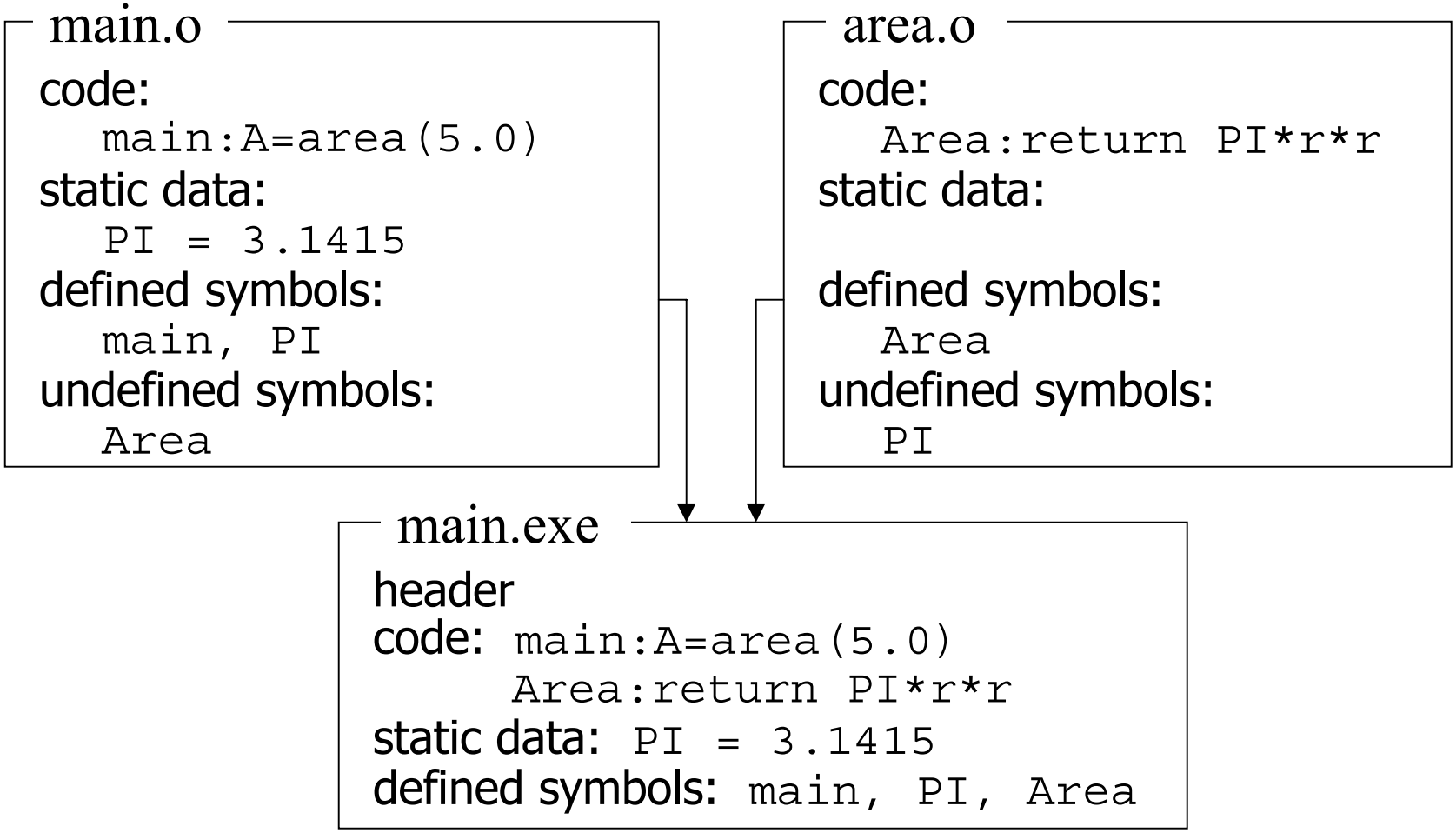
- Some addresses change during memory layout
- Modules were compiled in isolation
- *Absolute* addresses must be *relocated*
- Object file keeps track of instructions that use absolute addresses



# Resolving References

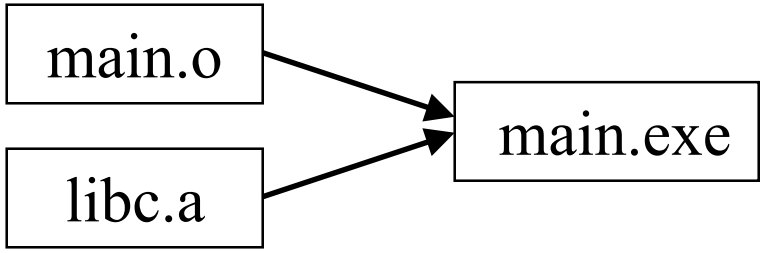
- For example, in a word processing program, an input module calls a spell check module
- Module address is unresolved at compile time
- The linker matches unresolved symbols to locations in other modules at link time
- In SPIM, “main” is resolved when your program is loaded

# Linker Example



# Libraries

- Some code is used so often, it is bundled into *libraries* for common access
- Libraries contain most of the code you use but didn't write: e.g., printf()
- Library code is (often) merged with yours at link time



# The Executable

- End result of compiling, assembling, and linking: the *executable*
  - » Header, listing the lengths of the other segments
  - » Text segment
  - » Static data segment
  - » Potentially other segments, depending on architecture & OS conventions

# Run Time

- When a program is started ...
  - » Some *dynamic linking* may occur
    - some symbols aren't defined until run time
    - Windows' dlls (dynamic link library)
  - » The segments are loaded into memory
  - » The OS transfers control to the program and it runs
- We'll learn a lot more about this during the OS part of the course