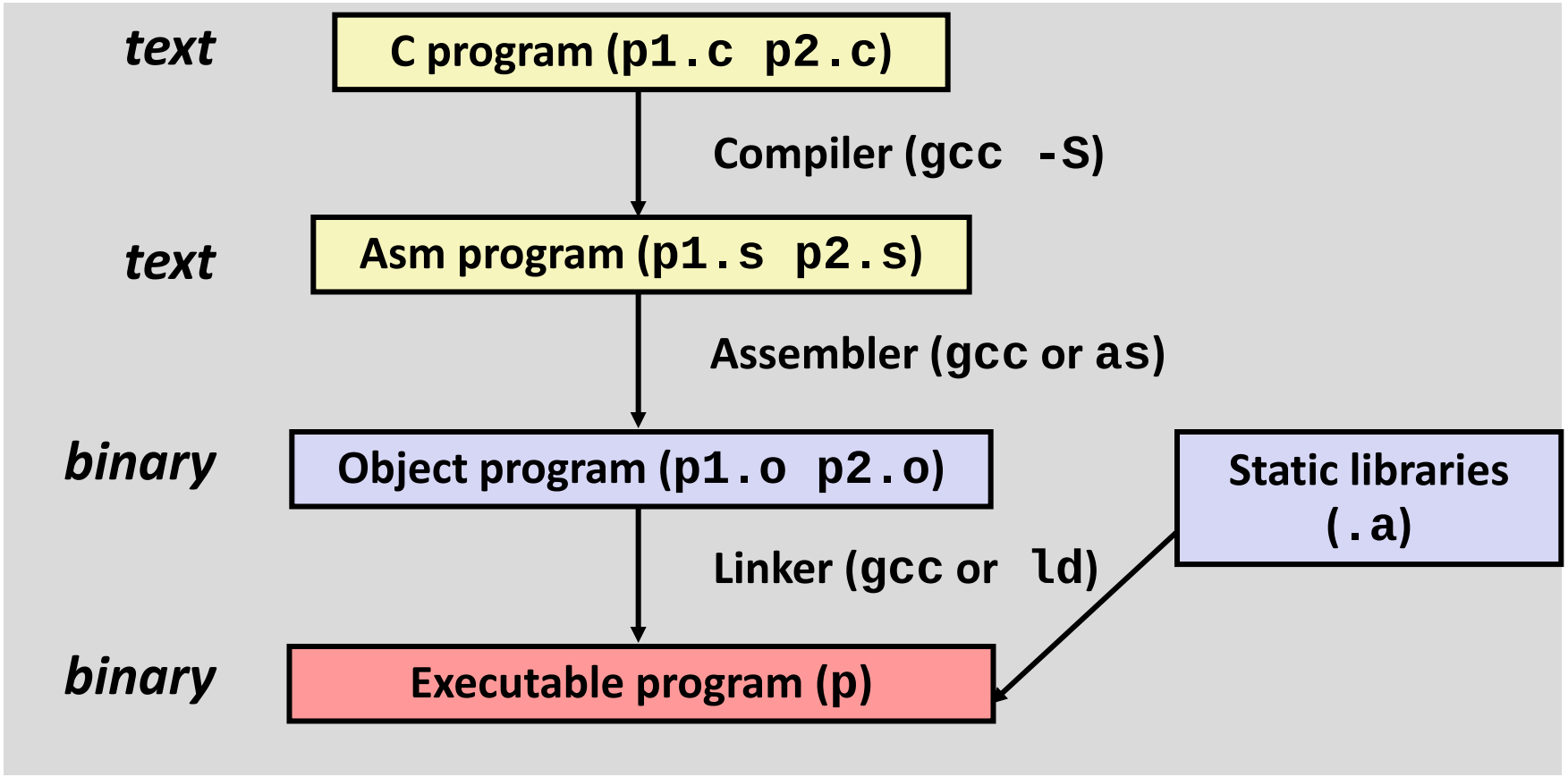
Compilation, Disassembly, and Profiling
(in Linux)

CS 485: Systems Programming
Spring 2016

Instructor:
Neil Moore

Turning C into Object Code

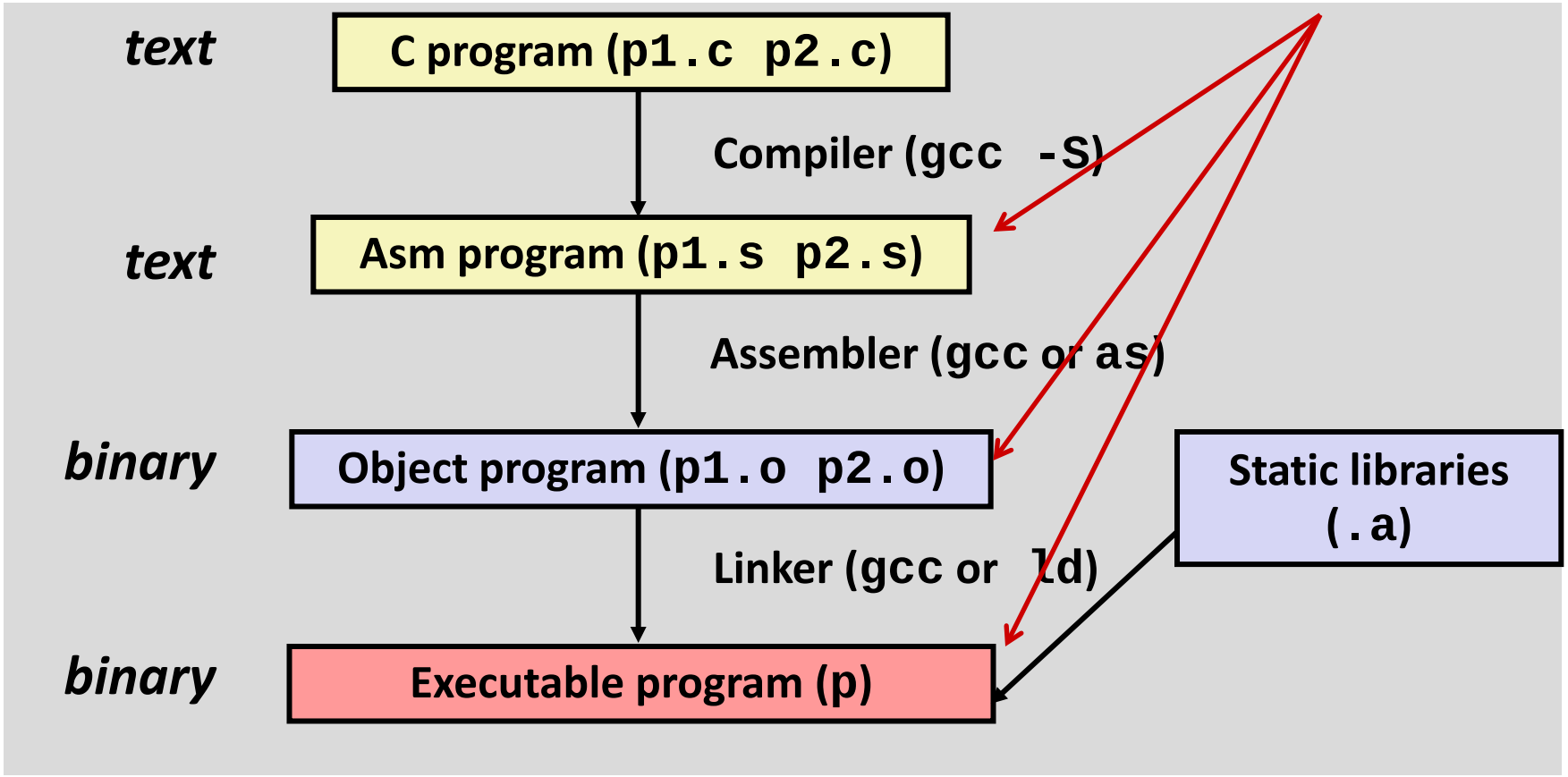
- Code in files `p1.c p2.c`
- Compile with command: `gcc –O1 p1.c p2.c -o p`
  - Use basic optimizations (`-O1`)
  - Put resulting binary in file `p`



Turning C into Object Code

- Code in files `p1.c p2.c`
- Compile with command: `gcc –O1 p1.c p2.c -o p`
  - Use basic optimizations (`-O1`)
  - Put resulting binary in file `p`

Can stop the compilation process at any stage.



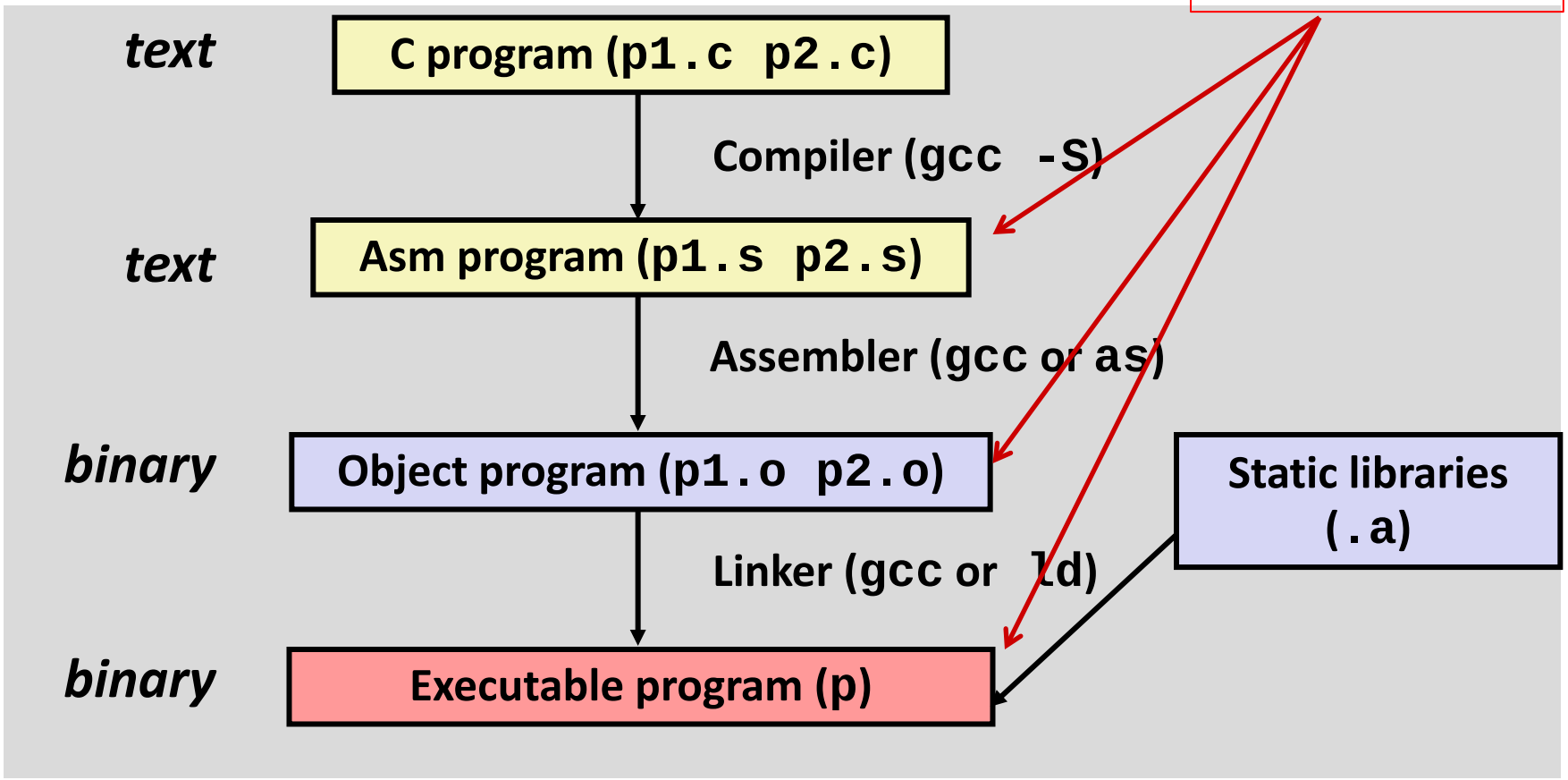
Invoking the compiler

- **We will use the Gnu command line compilers**
  - gcc – compiles C programs
  - g++ - compiles C++ programs (and C programs)
- **Useful command line options**
  - -o filename
    - Defines the output filename
    - Example: gcc –o hello hello.c
      - will create an executable file named “hello”
  - -E
    - Preprocess only – (the same as running the cpp program)
    - Example: gcc –E hello.c > hello.i
      - Will run the preprocessor and process header files to create “hello.i”
  - -c
    - Create an object file (.o) – i.e. Compile/Assemble, but do not link
    - Example: gcc –c hello.c
      - Will create an object file called hello.o
  - -S
    - Create an assembly language file (.s) – i.e., Compile, but do not assemble
    - Example: gcc –S hello.c
      - Will create an assembly language file called “hello.s”

Turning C into Object Code

- Code in files `p1.c p2.c`
- Compile with command: `gcc –O1 p1.c p2.c -o p`
  - Use basic optimizations (`-O1`)
  - Put resulting binary in file `p`

Information is lost at each step of the compilation Process.



Compiling Into Assembly

For example:
Variable names are lost
Parameters are lost

C Code

```
int triplesum(int x, int y)
{
    int t = x+y;
    return 3*t;
}
```

Generated IA32 Assembly

```
triplesum:
    addl %edi, %esi
    leal (%rsi,%rsi,2), %eax
    ret
```

Upon entry:
- x stored in %edi register
- y stored in %esi register

Then:
- t replaces y in %esi
- (%rsi is 64-bit %esi)
- return value goes to %eax

Also some directives like "`.cfi_startproc`" (used for debugging), and labels like "`.LFB0:`"

Obtain with command

```
gcc –O1 -S tsum.c
```

Produces file `tsum.s`

Object Code

Just binary bytes. Most assembly language is lost

Code for `sum`

```
0000 <triplesum>:
    0x01
    0xfe
    0x8d
    0x04
    0x76
    0xc3
```

- **Total of 6 bytes**
- **Each instruction here is 1, 2, or 3 bytes (but can be much longer)**
- **Placeholder address `0x0000`; actual address assigned by linker.**

- **Assembler**
  - Translates `.s` into `.o`
  - Binary encoding of each instruction
  - Nearly-complete image of executable code
  - Missing linkages between code in different files
- **Linker**
  - Resolves references between files
  - Combines with static run-time libraries
    - E.g., code for **`malloc`**, **`printf`**
  - Some libraries are *dynamically linked*
    - Linking occurs when program begins execution

Invoking the compiler

- **We will use the Gnu command line compilers**
  - gcc – compiles C programs
  - g++ - compiles C++ programs (and C programs)
- **Useful command line options**
  - -o filename
    - Defines the output filename
    - Example: gcc –o hello hello.c
      - will create an executable file named “hello”
  - -E
    - Preprocess only – (the same as running the cpp program)
    - Example: gcc –E hello.c > hello.i
      - Will run the preprocessor and process header files to create “hello.i”
  - -c
    - Create an object file (.o) – i.e. Compile/Assemble, but do not link
    - Example: gcc –c hello.c
      - Will create an object file called hello.o
  - -S
    - Create an assembly language file (.s) – i.e., Compile, but do not assemble
    - Example: gcc –S hello.c
      - Will create an assembly language file called “hello.s”
  - -g
    - Add symbol information to the file
    - Useful when debugging and disassembling programs
  - -pg
    - Add profiling information to the file
    - Useful when profiling performance
  - -O*level*
    - Optimize the code using the specified level’s optimizations
    - *Level* 0 is the fewest optimizations*, Level* 3 is the most optimizations
    - *Level* g (-Og) optimizes as much as possible without hurting debugging.
    - Example: gcc –O3 hello.c

Profiling Code

- Compile with the –pg option to gcc
- gprof – commonly installed and used profiling tool for unix-based systems
- Valgrind – more advanced tool that also comes with graphical user interfaces to visualize a program's performance and call graph

Gprof concepts

- **Step 1: Add profiling information to the program**
  - gcc –pg –o myprog myprog.c
- **Step 2: Run the program to create *gmon.out* (profile info)**
  - ./myprog
- **Step 3: Analyze the performance information**
  - View time spent in each procedure
    - gprof –p ./myprog
  - View call graph
    - gprof –q ./myprog

Disassembling Code

- There are a variety of tools that can be used to look at compiled code.
- Some are useful for seeing the assembly language code/instructions
  - objdump -d
  - gdb – using the disassemble command
- Some provide information about the data/variables
  - nm
- Some are basics tools that can give hints about what is in the file
  - strings
  - od
- Some are graphical front ends
  - dissy

Disassembling Object Code

Disassembled

```
000000000040055d <triplesum>:
 40055d:  01 fe       add    %edi,%esi
 40055f:  8d 04 76    lea    (%rsi,%rsi,2),%eax
 400562:  c3          ret
```

- **Disassembler**

  `objdump -d filename`

  - Useful tool for examining object code
  - Analyzes bit pattern of series of instructions
  - Produces approximate rendition of assembly code
  - Can be run on either `a.out` (complete executable) or `.o` file

Alternate Disassembly

Object

```
0x40055d:
    0x01
    0xfe
    0x8d
    0x04
    0x76
    0xc3
```

Disassembled

```
Dump of assembler code for function triplesum:
0x..40055d <+0>:     add    %edi,%esi
0x..40055f <+2>:     lea    (%rsi,%rsi,2),%eax
0x..400562 <+5>:     ret
```

- **Within gdb Debugger**
  - **`First run "gdb filename"`**
  - **`Then inside gdb type "disassemble sum"`**
    - Disassemble procedure
  - **`x/6xb sum`**
    - E**x**amine the 6 **b**ytes starting at `sum`, as he**x**adecimal