Software Engineering

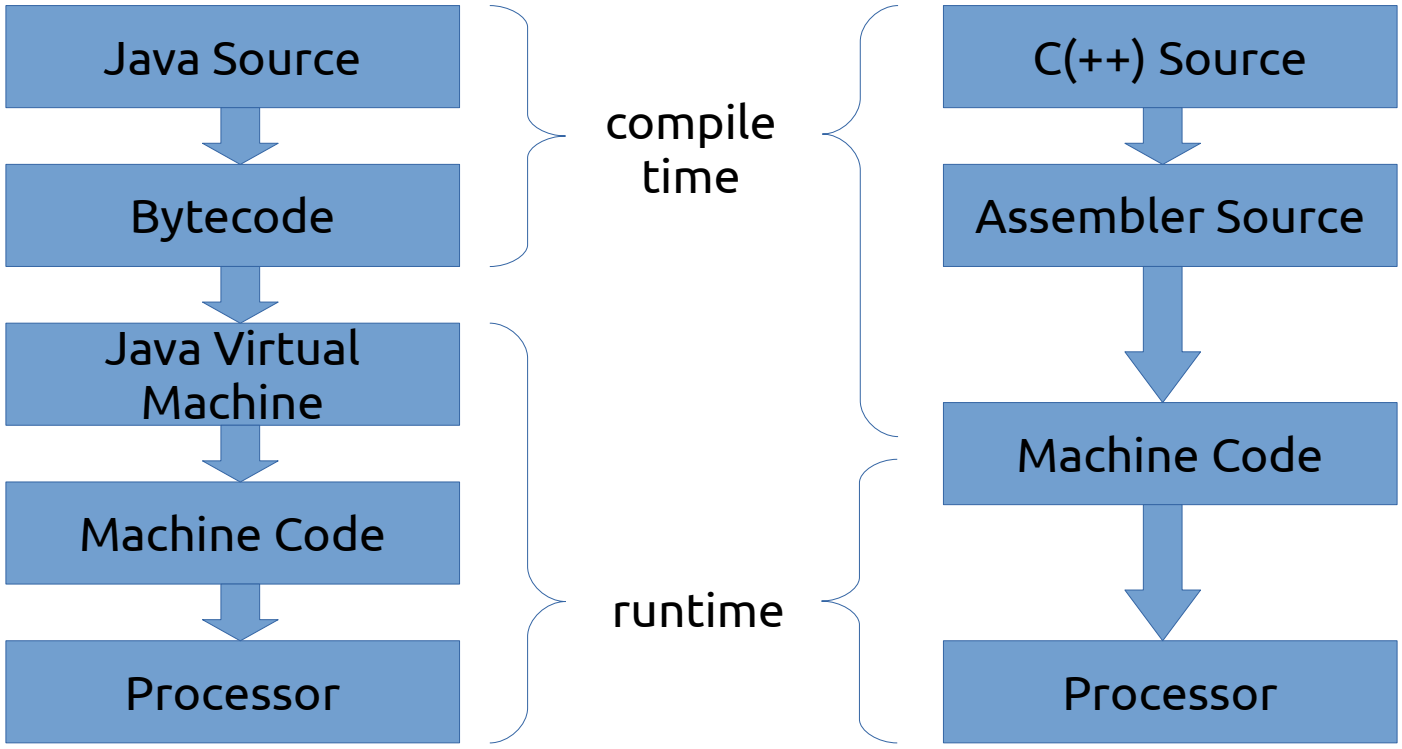
Lecture 09 – Build Process

© 2015-19 Dr. Florian Echtler
Bauhaus-Universität Weimar
<florian.echtler@uni-weimar.de>

Today's topics

- From code to binary
  - Compilation
  - Static & dynamic linking
- Inside a Hello World program

Preface: Java vs. C(++)



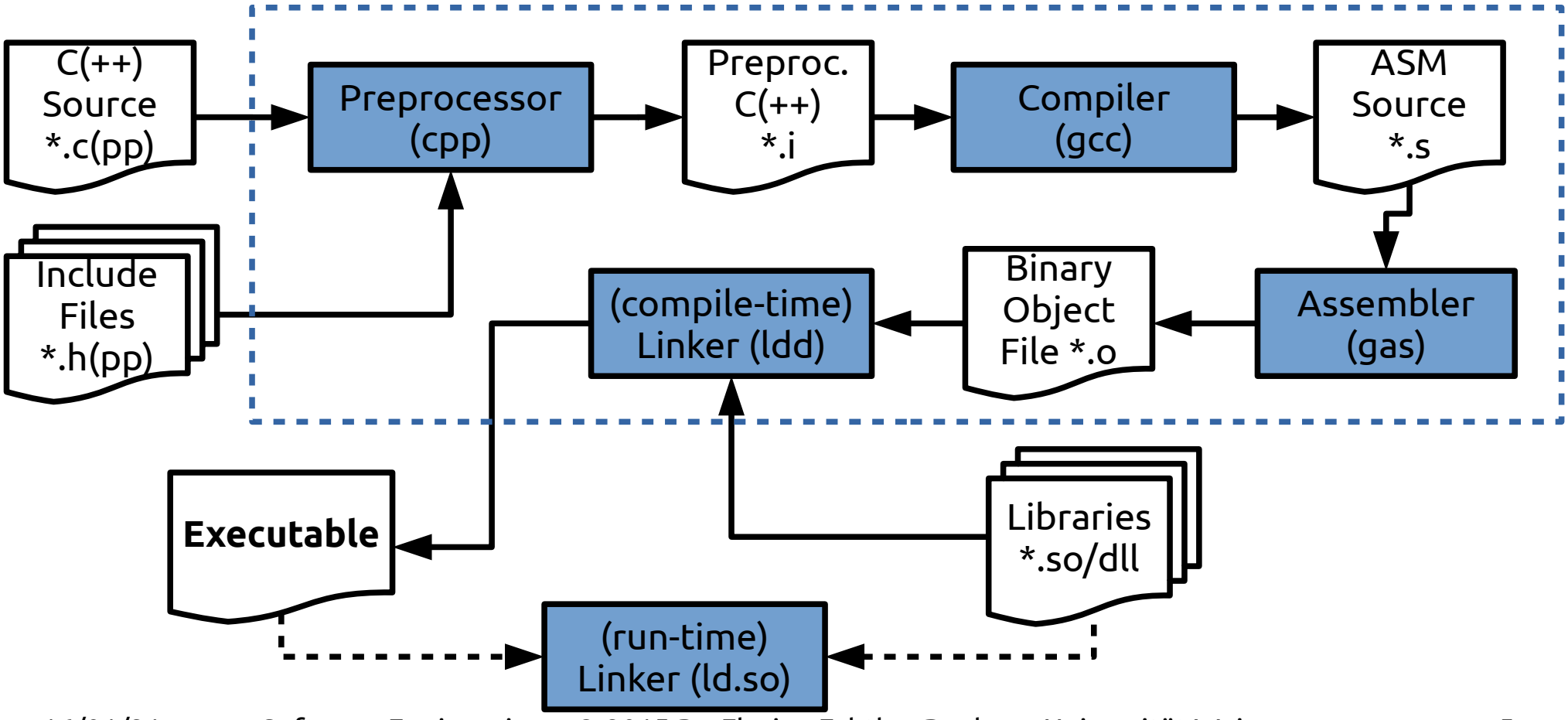
Preface: from code to binary

- What happens when you type

  `gcc -o helloworld helloworld.c?`

- 4 sub-stages:
  - Preprocessing
  - Compilation
  - Assembly
  - Linking (static/dynamic)

Preface: from code to binary (2)

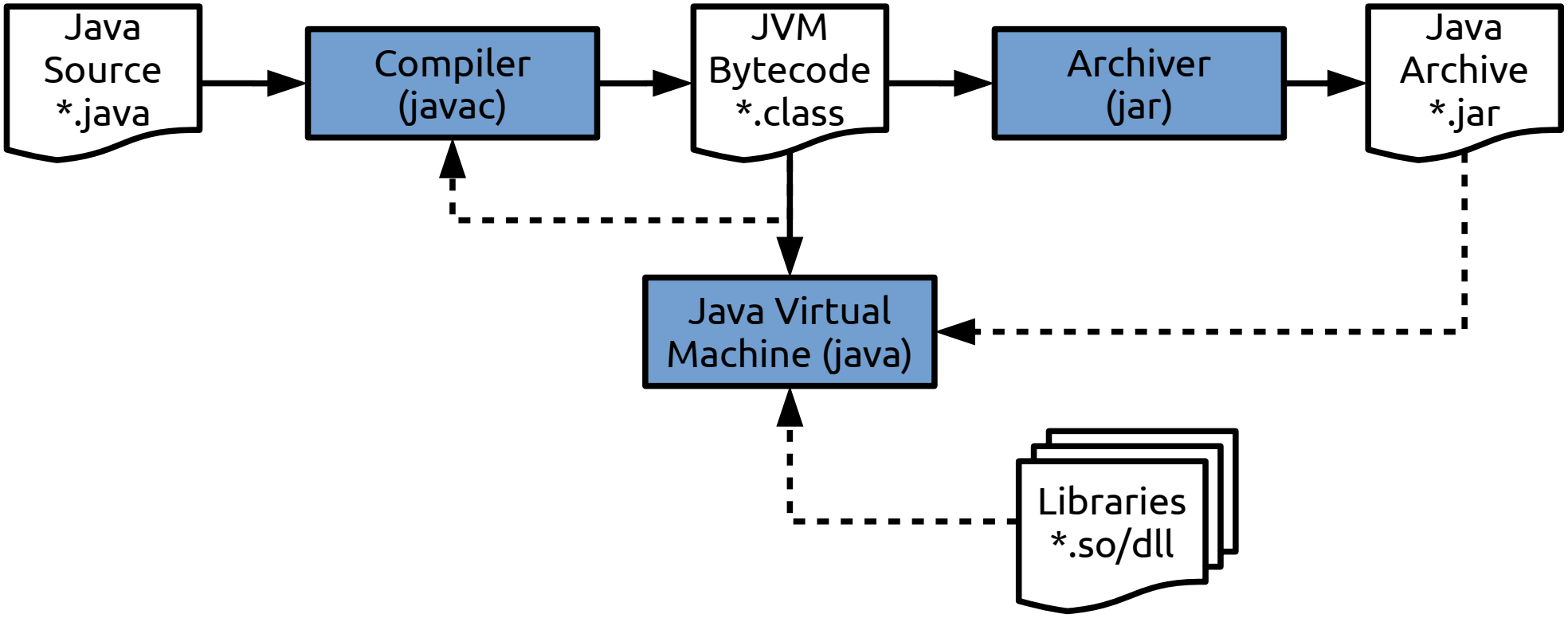


Preface: from code to binary (3)

- What happens when you type

  `javac HelloWorld.java`?
- 2 sub-stages:
  - Compilation & Assembly
  - Linking (static/dynamic)

Preface: from code to binary (4)

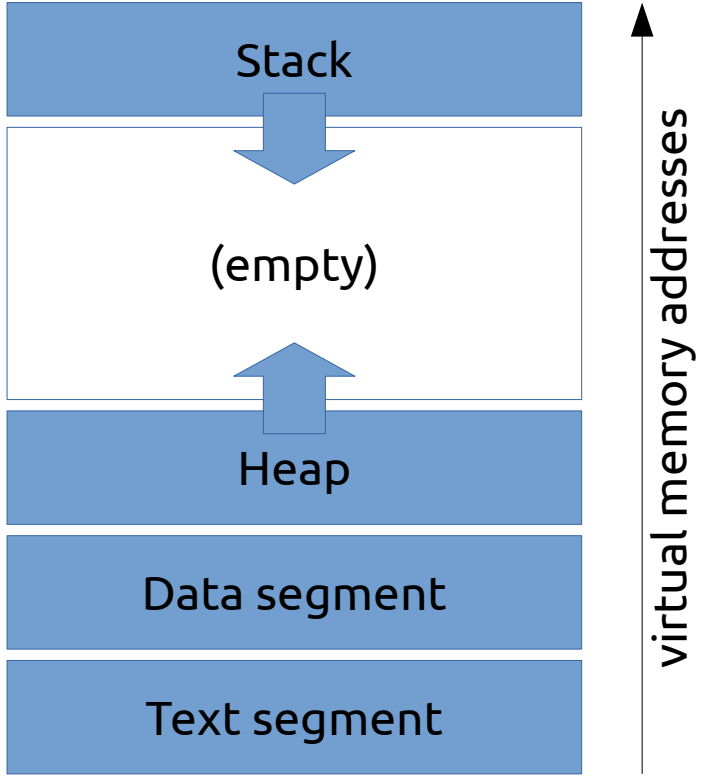


Tasks of the linker

- Combine multiple object files into binary
  - Allows modularization
  - Redundant code can be shared
- Determine address space/memory layout
  - Merge/relocate code blocks from separate objects
- Resolve symbolic references
  - References to library functions
  - References to functions in other objects

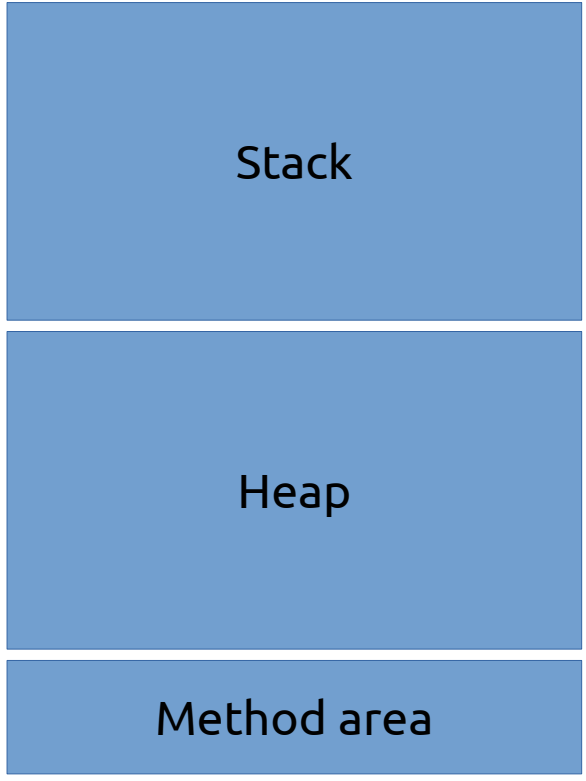
Memory Layout (C)

- Layout identical for each program (virtual memory)
  - Text Segment: executable binary code, read-only
  - Data Segment: global vars
  - Stack: used for local variables & return addrs
  - Heap: used for dynamic allocation (`new()`/`free()`)
  - Beware of memory leaks!



Memory Layout (Java)

- Everything inside the JVM
  - Global heap (shared)
  - One stack per thread
  - Fixed maximum size
  - Method storage
- JVM has *Garbage Collector*
  - Deletes unused objects
  - No memory leaks (in theory)
  - Can impact performance



Linking (static & dynamic)

- *Static* linking: linker tasks performed at *compile* time, results written to executable
  - Good: no incompatibilities possible
  - Bad: program size increases
- *Dynamic* linking: linker tasks performed at *run* time, results loaded into memory
  - Good: program can be kept smaller
  - Bad: library version conflicts (“DLL hell”)

Linking (static & dynamic) (2)

- Both steps usually combined:
  - program objects linked statically
  - libraries linked dynamically
- Examining static linkage:
  - C(++): `objdump -xd` *`object`*
  - Java: `javap -p -c -s` *`classfile`*
- Examining dynamic linkage
  - C(++): `ldd` *`executable`*
  - Java: not applicable (why?)

Hello world: under the hood

Image source (PD): https://en.wikipedia.org/...old_jeep_from_world_war_two.jpg

Hello world (C): under the hood

- `hello.c → hello.i`

```
#include <stdio.h> → [content copy of stdio.h]

void hello() {
    printf("Hello ");
}
```

Hello world (C): under the hood

- `hello.c/hello.i → hello.s`

```
    .section .rodata
.LC0:
    .string  "Hello "

    .text
    .globl   hello
    .type    hello, @function

hello:
    pushq    %rbp
    movq %rsp, %rbp

    movl $.LC0, %edi
    movl $0, %eax
    call printf

    popq %rbp
    ret
```

global read-only data section with “label” .LC0

start of code section with global label “hello”

“setup” code

printf function call with label reference to .LC0

“cleanup” code

Hello world (C): under the hood

- `hello.c/hello.i → hello.s → hello.o`

```
Disassembly of section .text:

0000000000000000 <hello>:
       0:   55                      push   %rbp
       1:   48 89 e5                mov    %rsp,%rbp
       4:   bf 00 00 00 00          mov    $0x0,%edi
            5: R_X86_64_32 .rodata

       9:   b8 00 00 00 00          mov    $0x0,%eax
       e:   e8 00 00 00 00          callq  13 <hello+0x13>
            f: R_X86_64_PC32   printf-0x4

      13:   5d                      pop    %rbp
      14:   c3                      retq
```

start address (not yet linked → zero)

Relocations (final addresses are not yet known, will be filled in by linker)

Hello world (C): under the hood

- `hello_world.dynamic (after linking)`

```
Disassembly of section .text:

00000000004005a2 <hello>:

  4005a2:   55                      push   %rbp
  4005a3:   48 89 e5                mov    %rsp,%rbp
  4005a6:   bf 54 06 40 00          mov    $0x400654,%edi

  4005ab:   b8 00 00 00 00          mov    $0x0,%eax
  4005b0:   e8 ab fe ff ff          callq  400460 <printf@plt>

  4005b5:   5d                      pop    %rbp
  4005b6:   c3                      retq
```

start address (determined by linker)

Former relocations (final addresses filled in by linker)

Hello world (Java): under the hood

- `HelloWorld.java`

```
class HelloWorld {
  public static void main(String args[]) {
    System.out.println("Hello World!");
  }
}
```

Hello world (Java): under the hood

- `HelloWorld.class`

```
public static void main(java.lang.String[]);
 descriptor: ([Ljava/lang/String;)V
 Code:
   0: getstatic     #2  // get static class field
   // Field java/lang/System.out:Ljava/io/PrintStream;
   3: ldc           #3  // get string “Hello World!”
   5: invokevirtual #4  // invoke virtual method
   // Method java/io/PrintStream.println:(Ljava/lang/String;)V
   8: return
```

Reference to a PrintStream object from static field System.out

Invoke method println on PrintStream object with String object as parameter

Reference to a constant String object from local object pool

Questions/Comments?