SHARE
Technology • Connections • Results

Linux on zSeries ABI and Linkage Format SHARE 102 Session 9236

Dr. Ulrich Weigand
Linux on zSeries Development, IBM Lab Böblingen
Ulrich.Weigand@de.ibm.com

Agenda

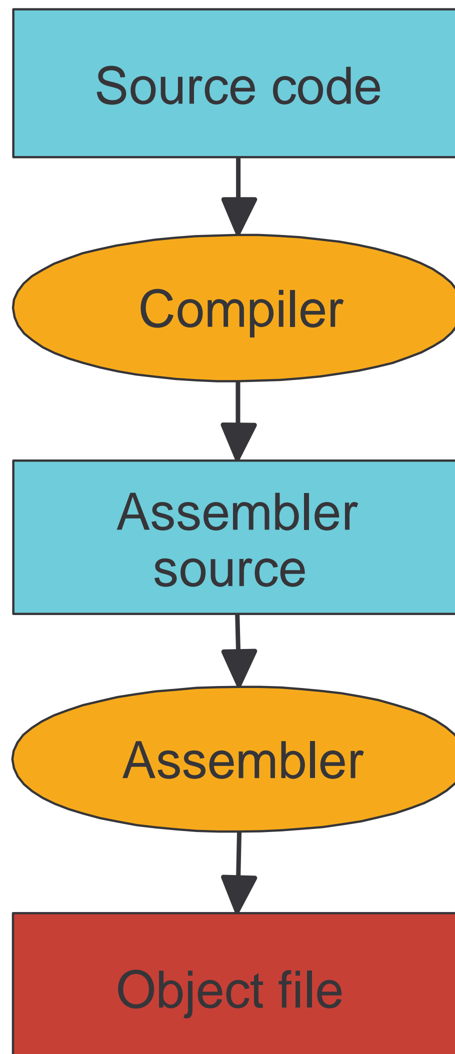
- Compiling, linking, and loading
- Function calling sequence
- Executable and Linking Format (ELF)
- Program loading and dynamic linking
- Debugging and exception handling

The Standard Example

```
$ cat hello.c

#include <stdio.h>
int main (void)
{
  printf ("Hello, world!\n");
}

$ gcc -O3 -o hello hello.c

$ ./hello
Hello, world!
```

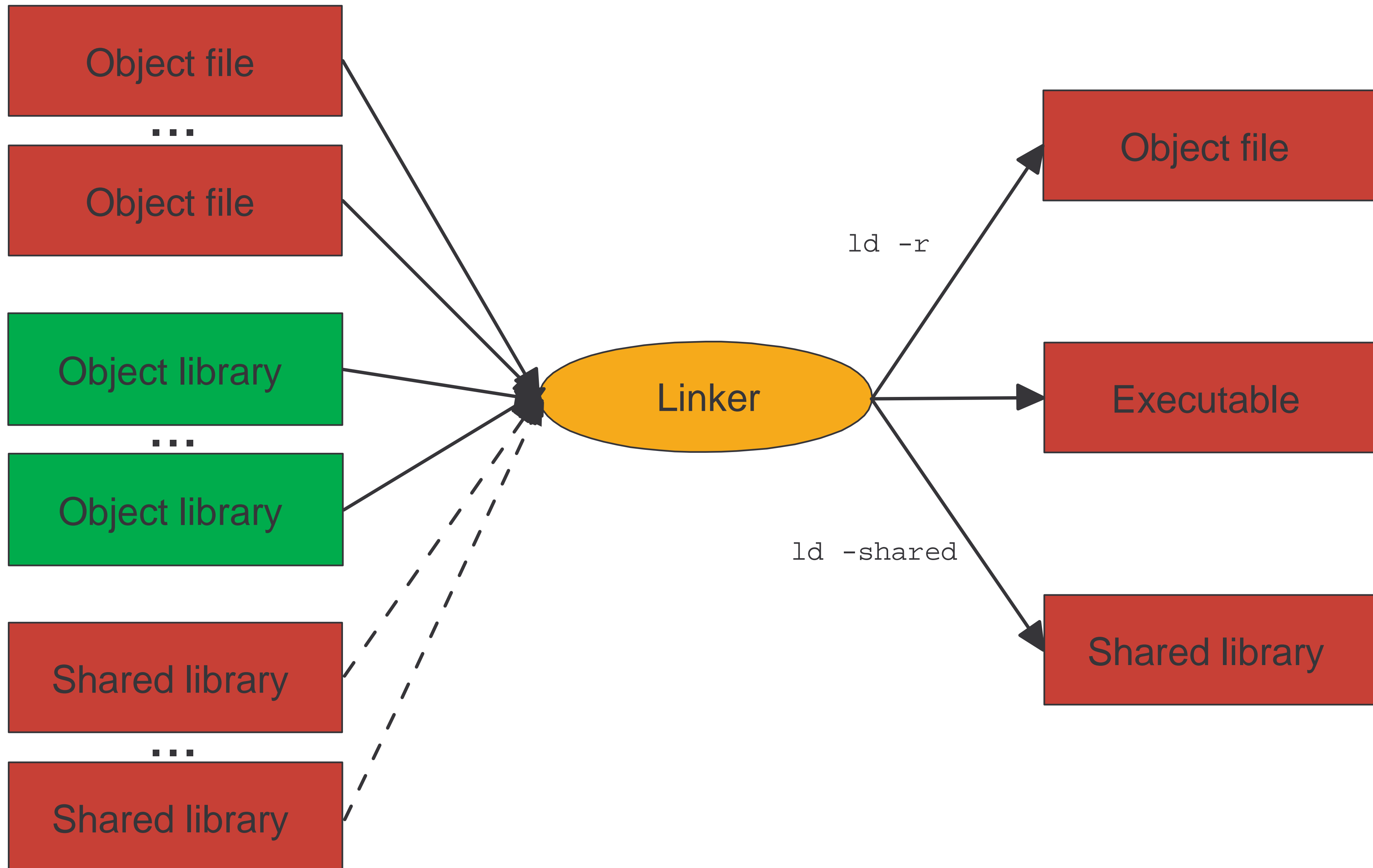
Compiling ...

SHARE
Technology • Connections • Results



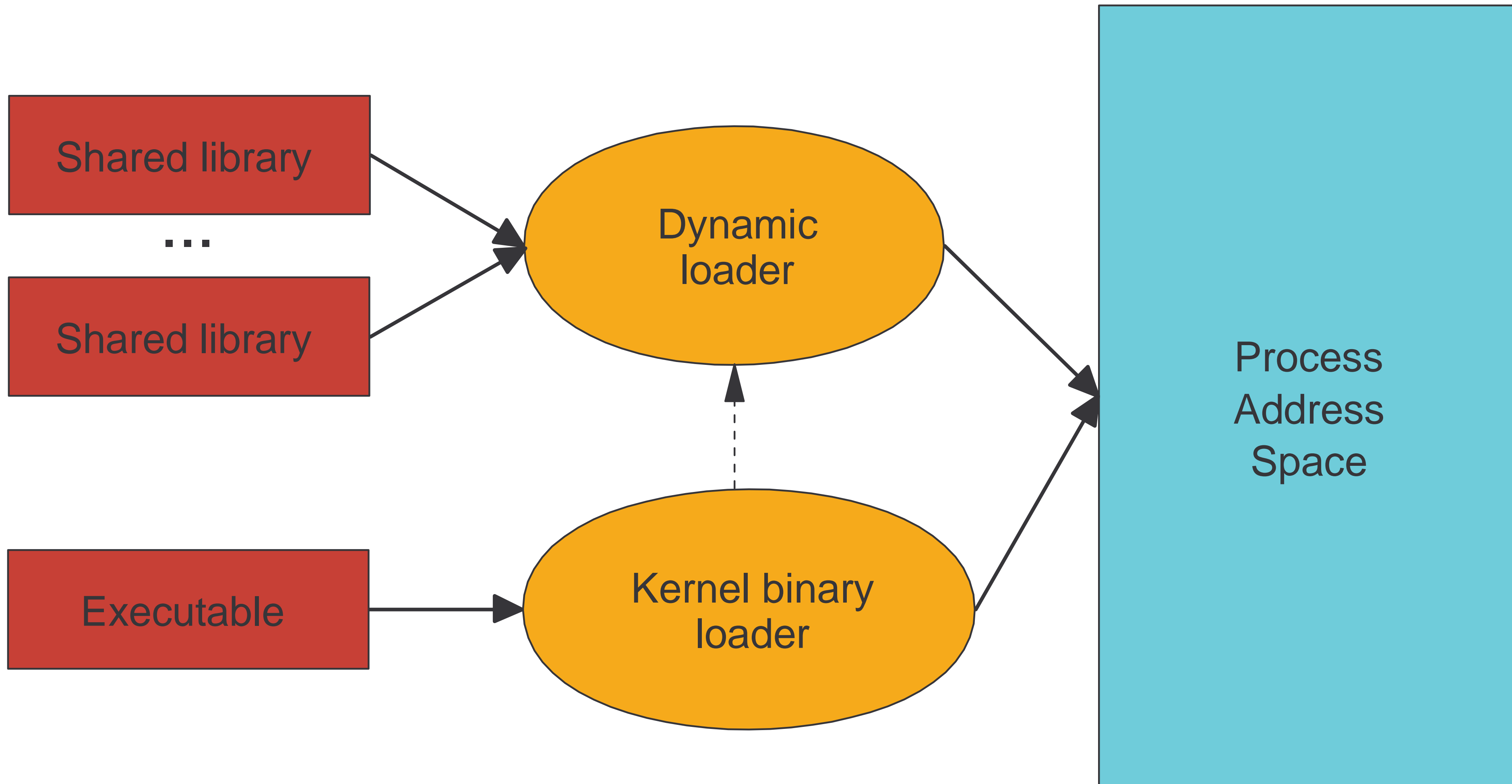

```
$ gcc -O3 -o hello hello.c -v --save-temps

  cc1 -E hello.c -O3 -o hello.i
  cc1 -fpreprocessed hello.i -O3 -o hello.s

  as -o hello.o hello.s

  ld -m elf_s390 -dynamic-linker /lib/ld.so.1 -o hello
    crt1.o crti.o crtbegin.o hello.o
    -lgcc -lgcc_eh -lc -lgcc -lgcc_eh crtend.o crtn.o
```

Linking ...



Loading ...

SHARE
Technology • Connections • Results



Function Calling Sequence

SHARE
Technology • Connections • Results

- Convention between caller and callee
  - Register usage (clobbered/saved)
  - Parameter / return value passing
  - Call linkage / return address
  - Stack layout / register save area
- Design goals
  - Compatible ABI between different compilers/languages
  - Efficient argument passing, avoid 'abstraction penalty'
  - Short function prolog/epilog without pipeline stalls

Function Calling Sequence (cont.)

SHARE
Technology • Connections • Results

- Register usage
  - 16 general purpose registers (32-bit / 64-bit)
    - Saved: %r6-%r15, clobbered: %r0-%r5
    - %r0,%r1 may be modified during inter-module calls!
  - 16 floating point registers (64-bit)
    - [S/390] Saved: %f4,%f6, clobbered: %f0-%f3,%f5,%f7-%f15
    - [zSeries] Saved: %f8-%f15, clobbered: %f0-%f7
  - 16 access registers (32-bit)
    - Clobbered: %a0-%a15
    - %a0 [S/390] / %a0-%a1 [zSeries] reserved for system library use
    - *Note: Linux on zSeries applications never run in AR mode!*

Function Calling Sequence (cont.)

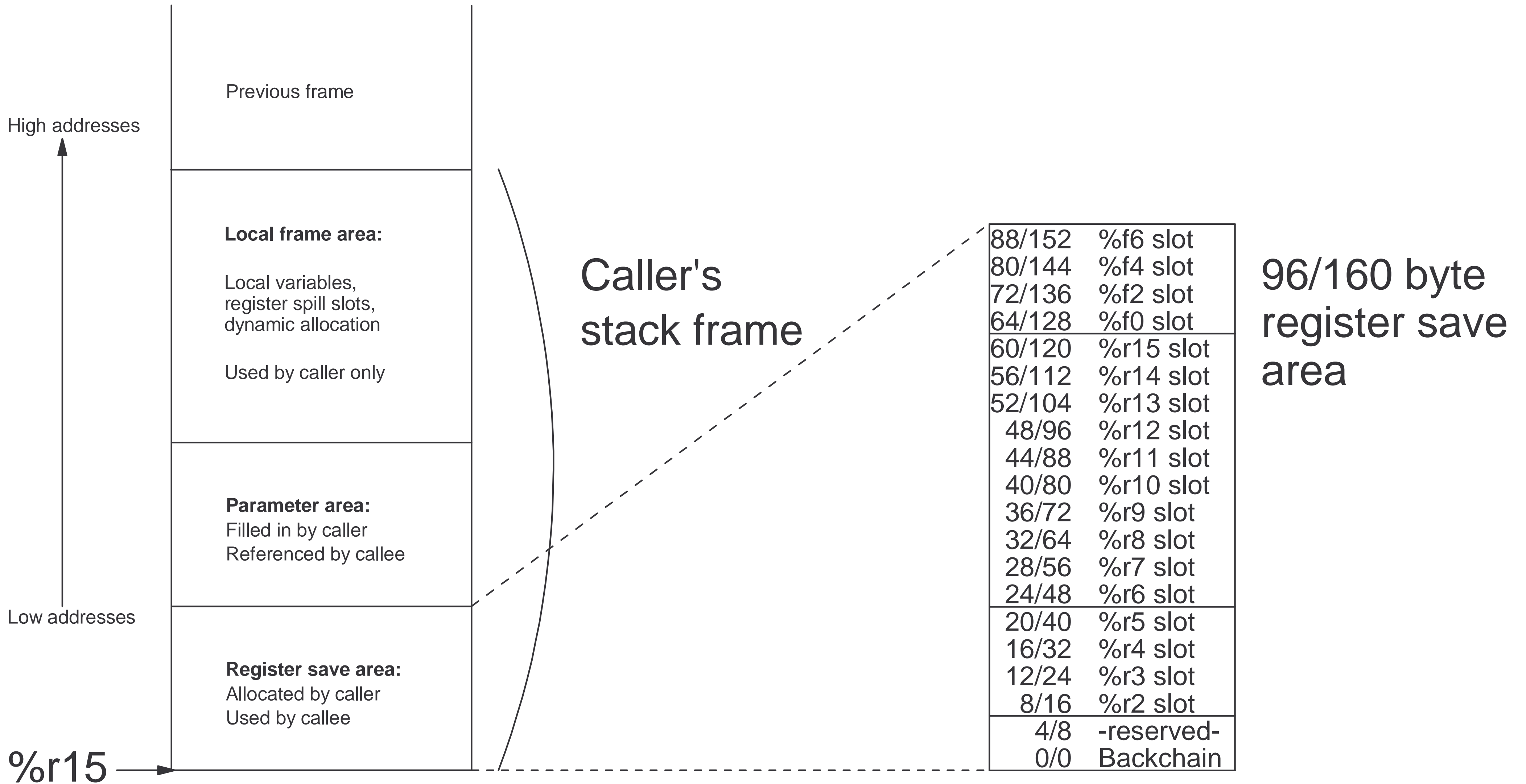
SHARE
Technology • Connections • Results

- Parameter passing
  - Integer/pointer values (1,2,4,8 bytes) passed in %r2-%r6
    - [S/390] 8-byte integers use register pair (a sole %r6 is skipped)
    - Small values must be zero-/sign-extended to full register width
  - Floating-point values passed in floating-point registers
    - [S/390] Available registers: %f0,%f2
    - [zSeries] Available registers: %f0,%f2,%f4,%f6
  - Some small aggregates are passed by value
    - 'Floating-point equivalent' structs like floating-point values
    - Other aggregates of size 1,2,4,8 bytes like integer values
  - All other values are passed by implicit reference
  - Overflow parameters in defined area in caller's stack frame

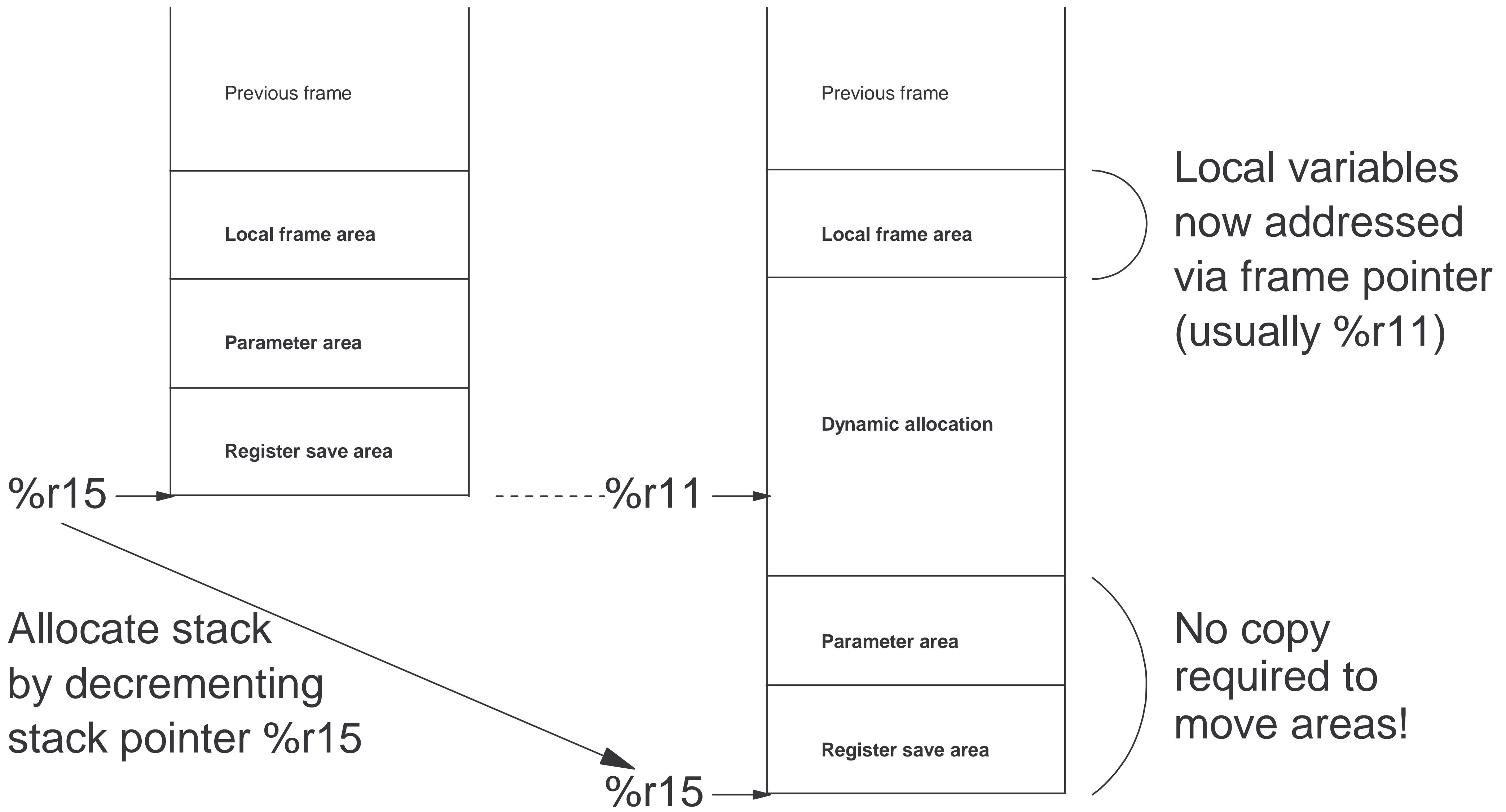
Function Calling Sequence (cont.)

- Return value
  - Integer/pointer values in %r2 ([S/390] 8 bytes in %r2/%r3)
    - Small values must be zero-/sign-extended to full register width
  - Floating-point values in %f0
  - Aggregates and all other values via hidden reference
    - Caller needs to allocate buffer
- Call linkage
  - No fixed register defined to hold callee's address
  - Return address passed in %r14
  - Stack pointer (register save area) passed in %r15
  - *Note: Mixed addressing mode applications not supported!*

Stack Frame Layout



Dynamic Stack Allocation

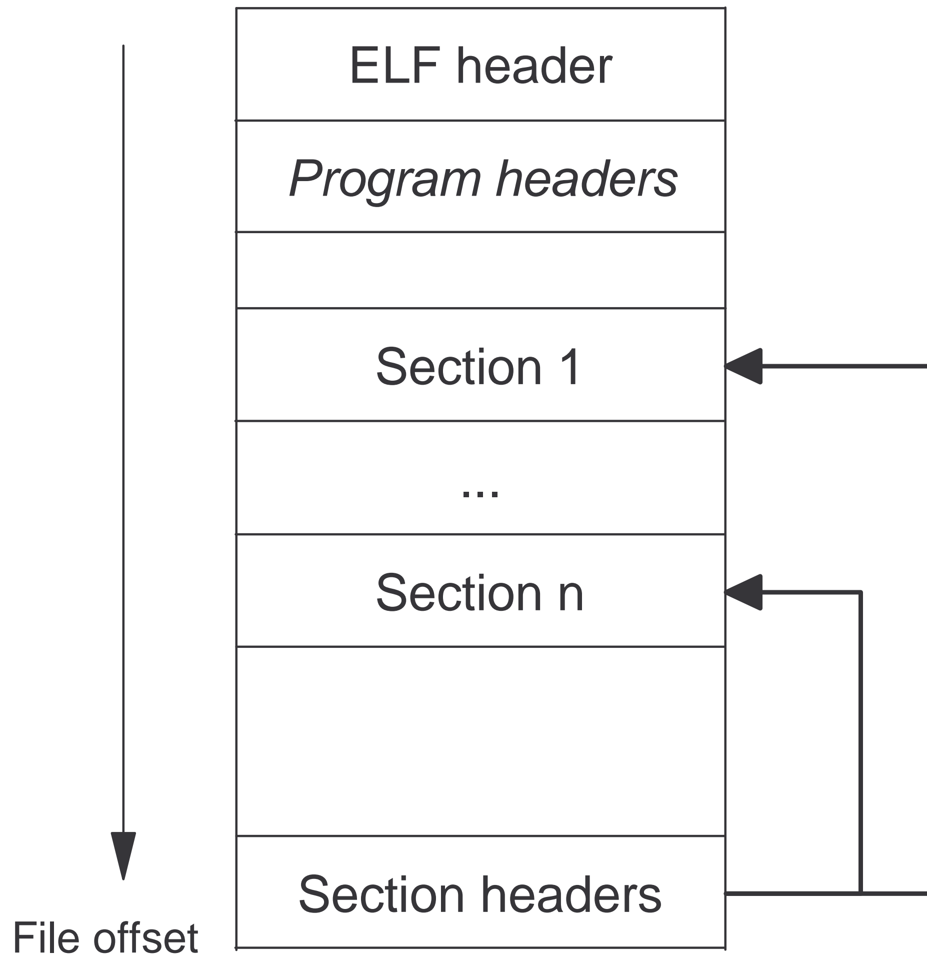


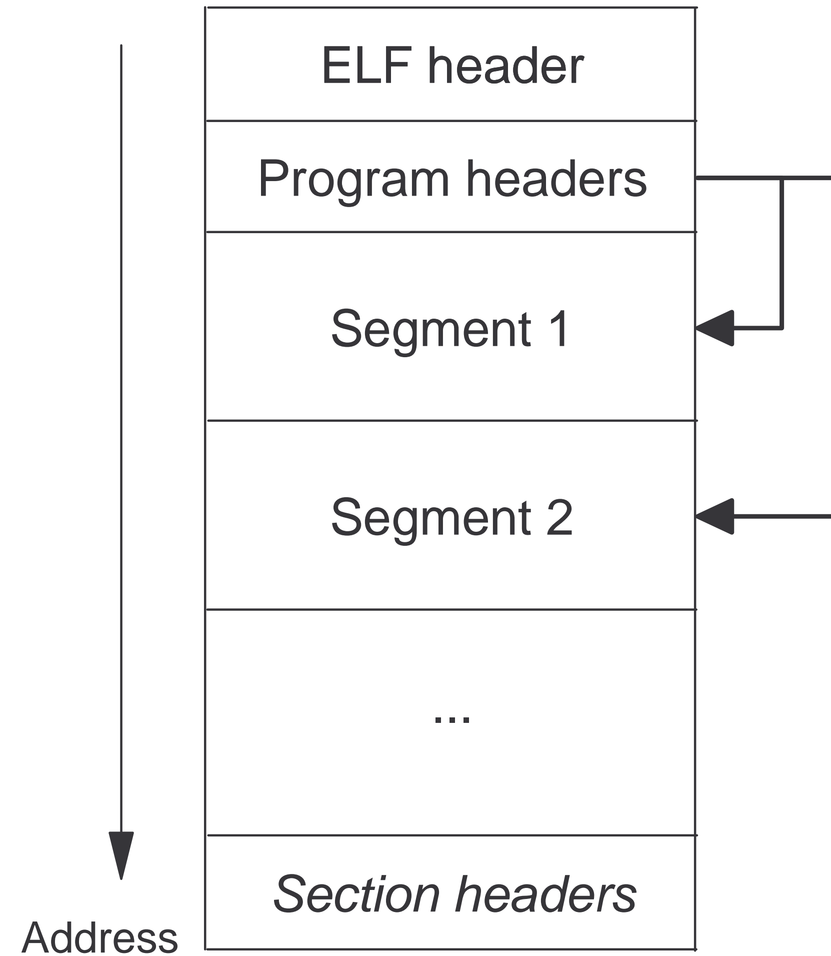
Example (cont.) - Assembler source

```
        .file   "hello.c"
        .section .rodata.str1.2,"aMS",@progbits,1
        .align  2
.LC0:
        .string "Hello, world!"
.text
        .align  4
.globl main
        .type   main, @function
main:
        stm     %r13,%r15,52(%r15)
        basr    %r13,0
.L2:
        l       %r1,.L3-.L2(%r13)
        ahi     %r15,-96
        l       %r2,.L4-.L2(%r13)
        basr    %r14,%r1
        lhi     %r2,0
        l       %r4,152(%r15)
        lm      %r13,%r15,148(%r15)
        br      %r4
```

```
        .align  4
.L4:
        .long   .LC0
.L3:
        .long   puts
        .align  2
        .size   main, .-main
        .section .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.0 (prerelease)"
```

Executable and Linking Format





Sections and Segments

- Sections - the linking view
  - Represent different content types
    - Code, data, linkage information, debug information, ...
  - May be present in file only (e.g. debug info)
  - Link step merges content from many sources *per section*
- Segments - the execution view
  - Represent different load attributes
    - Writable, executable, shared vs. private mapping
  - May be present in memory only (e.g. uninitialized data)
  - Page alignment required

Symbols and Relocation

- Determining absolute addresses
  - Machine code/data references absolute addresses
  - Addresses will change as result of section merging
  - Only known after final link step (or after loading!)
- Relocation mechanism
  - Code/data sections contain placeholders for addresses
  - Relocation sections contain instructions to update those
  - Relocation types represent methods of computing addresses
- Symbol tables
  - Relocation records refer to addressed by symbolic names
  - Symbol tables associate names with locations

Example (cont.) - ELF Header

SHARE
Technology • Connections • Results

```
$ readelf --file-header hello.o

ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           IBM S/390
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          240 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         11
  Section header string table index: 8
```

Example (cont.) - ELF Header (executable)

SHARE Technology • Connections • Results

```
$ readelf --file-header hello

ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           IBM S/390
  Version:                           0x1
  Entry point address:               0x400330
  Start of program headers:          52 (bytes into file)
  Start of section headers:          2336 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         29
  Section header string table index: 26
```

Example (cont.) - Sections

```
$ readelf --sections hello.o

There are 11 section headers, starting at offset 0xf0:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 00002c 00  AX  0   0  4
  [ 2] .rela.text        RELA            00000000 000370 000018 0c      9   1  4
  [ 3] .data             PROGBITS        00000000 000060 000000 00  WA  0   0  4
  [ 4] .bss              NOBITS          00000000 000060 000000 00  WA  0   0  4
  [ 5] .rodata.str1.2    PROGBITS        00000000 000060 00000e 01 AMS  0   0  2
  [ 6] .note.GNU-stack   PROGBITS        00000000 00006e 000000 00      0   0  1
  [ 7] .comment          PROGBITS        00000000 00006e 000028 00      0   0  1
  [ 8] .shstrtab         STRTAB          00000000 000096 000059 00      0   0  1
  [ 9] .symtab           SYMTAB          00000000 0002a8 0000b0 10     10   9  4
  [10] .strtab           STRTAB          00000000 000358 000018 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Example (cont.) - Sections (executable)

```
[ 1] .interp          PROGBITS        00400134 000134 00000d 00   A  0   0  1
[ 2] .note.ABI-tag    NOTE            00400144 000144 000020 00   A  0   0  4
[ 3] .hash            HASH            00400164 000164 000028 04   A  4   0  4
[ 4] .dynsym          DYNSYM          0040018c 00018c 000050 10   A  5   1  4
[ 5] .dynstr          STRTAB          004001dc 0001dc 00004a 00   A  0   0  1
[ 6] .gnu.version     VERSYM          00400226 000226 00000a 02   A  4   0  2
[ 7] .gnu.version_r   VERNEED         00400230 000230 000020 00   A  5   1  4
[ 8] .rela.dyn        RELA            00400250 000250 00000c 0c   A  4   0  4
[ 9] .rela.plt        RELA            0040025c 00025c 000018 0c   A  4   b  4
[10] .init            PROGBITS        00400274 000274 00005c 00  AX  0   0  4
[11] .plt             PROGBITS        004002d0 0002d0 000060 04  AX  0   0  4
[12] .text            PROGBITS        00400330 000330 000234 00  AX  0   0  4
[13] .fini            PROGBITS        00400564 000564 000038 00  AX  0   0  4
[14] .rodata          PROGBITS        0040059c 00059c 000016 00   A  0   0  4
[15] .eh_frame_hdr    PROGBITS        004005b4 0005b4 00001c 00   A  0   0  4
[16] .data            PROGBITS        004015d0 0005d0 00000c 00  WA  0   0  4
[17] .eh_frame        PROGBITS        004015dc 0005dc 000060 00   A  0   0  4
[18] .dynamic         DYNAMIC         0040163c 00063c 0000c8 08  WA  5   0  4
[19] .ctors           PROGBITS        00401704 000704 000008 00  WA  0   0  4
[20] .dtors           PROGBITS        0040170c 00070c 000008 00  WA  0   0  4
[21] .jcr             PROGBITS        00401714 000714 000004 00  WA  0   0  4
[22] .got             PROGBITS        00401718 000718 000018 04  WA  0   0  4
[23] .sbss            PROGBITS        00401730 000730 000000 00   W  0   0  1
[24] .bss             NOBITS          00401730 000730 000004 00  WA  0   0  4
[25] .comment         PROGBITS        00000000 000730 00010b 00      0   0  1
[26] .shstrtab        STRTAB          00000000 00083b 0000e4 00      0   0  1
[27] .symtab          SYMTAB          00000000 000da8 000460 10     28  2c  4
[28] .strtab          STRTAB          00000000 001208 000225 00      0   0  1
```

Example (cont.) - Segments

SHARE
Technology • Connections • Results

```
$ readelf --segments hello

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00400034 0x00400034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x00400134 0x00400134 0x0000d 0x0000d R   0x1
      [Requesting program interpreter: /lib/ld.so.1]
  LOAD           0x000000 0x00400000 0x00400000 0x005d0 0x005d0 R E 0x1000
  LOAD           0x0005d0 0x004015d0 0x004015d0 0x00160 0x00164 RW  0x1000
  DYNAMIC        0x00063c 0x0040163c 0x0040163c 0x000c8 0x000c8 RW  0x4
  NOTE           0x000144 0x00400144 0x00400144 0x00020 0x00020 R   0x4
  GNU_EH_FRAME   0x0005b4 0x004005b4 0x004005b4 0x0001c 0x0001c R   0x4
  STACK          0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version .gnu.version_r
          .rela.dyn .rela.plt .init .plt .text .fini .rodata .eh_frame_hdr
   03     .data .eh_frame .dynamic .ctors .dtors .jcr .got .bss
   04     .dynamic
   05     .note.ABI-tag
   06     .eh_frame_hdr
   07
```

Example (cont.) - Relocations

SHARE

Technology • Connections • Results

```
$ objdump --disassemble --disassemble-zeroes --reloc hello.o

hello.o:     file format elf32-s390

Disassembly of section .text:

00000000 <main>:
   0:   90 df f0 34             stm     %r13,%r15,52(%r15)
   4:   0d d0                   basr    %r13,%r0
   6:   58 10 d0 22             l       %r1,34(%r13)
   a:   a7 fa ff a0             ahi     %r15,-96
   e:   58 20 d0 1e             l       %r2,30(%r13)
  12:   0d e1                   basr    %r14,%r1
  14:   a7 28 00 00             lhi     %r2,0
  18:   58 40 f0 98             l       %r4,152(%r15)
  1c:   98 df f0 94             lm      %r13,%r15,148(%r15)
  20:   07 f4                   br      %r4
  22:   07 07                   bcr     0,%r7
  24:   00 00 00 00             .long   0x00000000
                        24: R_390_32    .LC0
  28:   00 00 00 00             .long   0x00000000
                        28: R_390_32    puts
```

Example (cont.) - Relocations resolved

SHARE
Technology · Connections · Results

```
$ objdump --disassemble --disassemble-zeroes --reloc hello

hello:     file format elf32-s390
[...]
Disassembly of section .text:
[...]
00400408 <main>:
  400408:       90 df f0 34             stm     %r13,%r15,52(%r15)
  40040c:       0d d0                   basr    %r13,%r0
  40040e:       58 10 d0 22             l       %r1,34(%r13)
  400412:       a7 fa ff a0             ahi     %r15,-96
  400416:       58 20 d0 1e             l       %r2,30(%r13)
  40041a:       0d e1                   basr    %r14,%r1
  40041c:       a7 28 00 00             lhi     %r2,0
  400420:       58 40 f0 98             l       %r4,152(%r15)
  400424:       98 df f0 94             lm      %r13,%r15,148(%r15)
  400428:       07 f4                   br      %r4
  40042a:       07 07                   bcr     0,%r7
  40042c:       00 40 05 a4             .long   0x004005a4
  400430:       00 40 02 f0             .long   0x004002f0
[...]
```

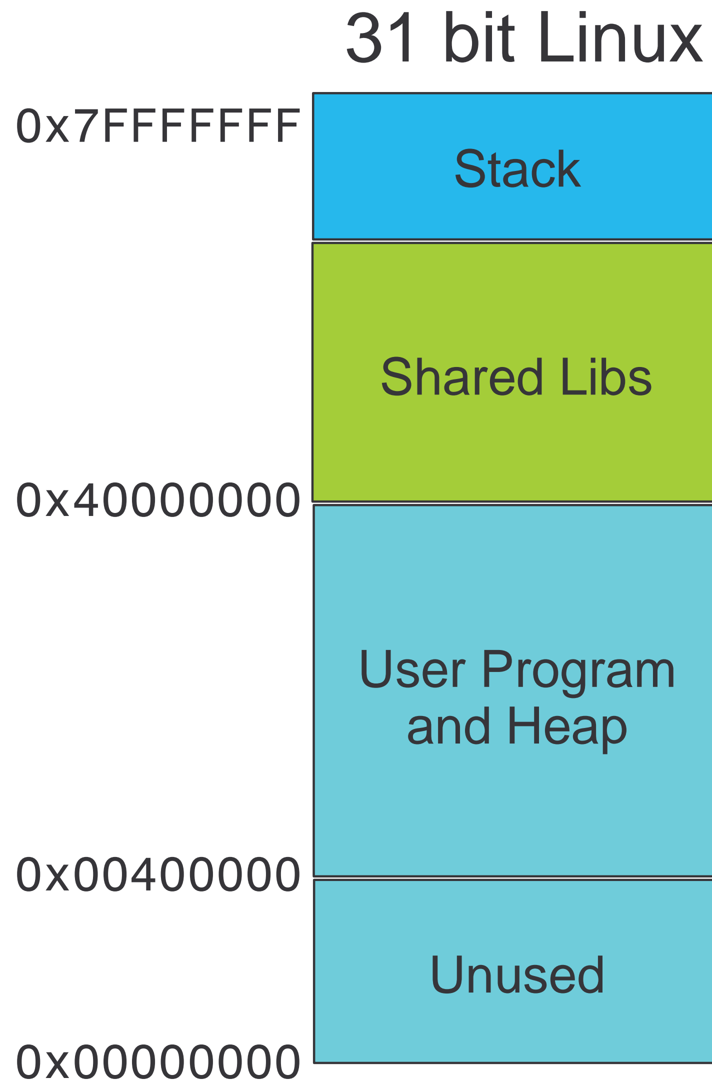
Program Loading

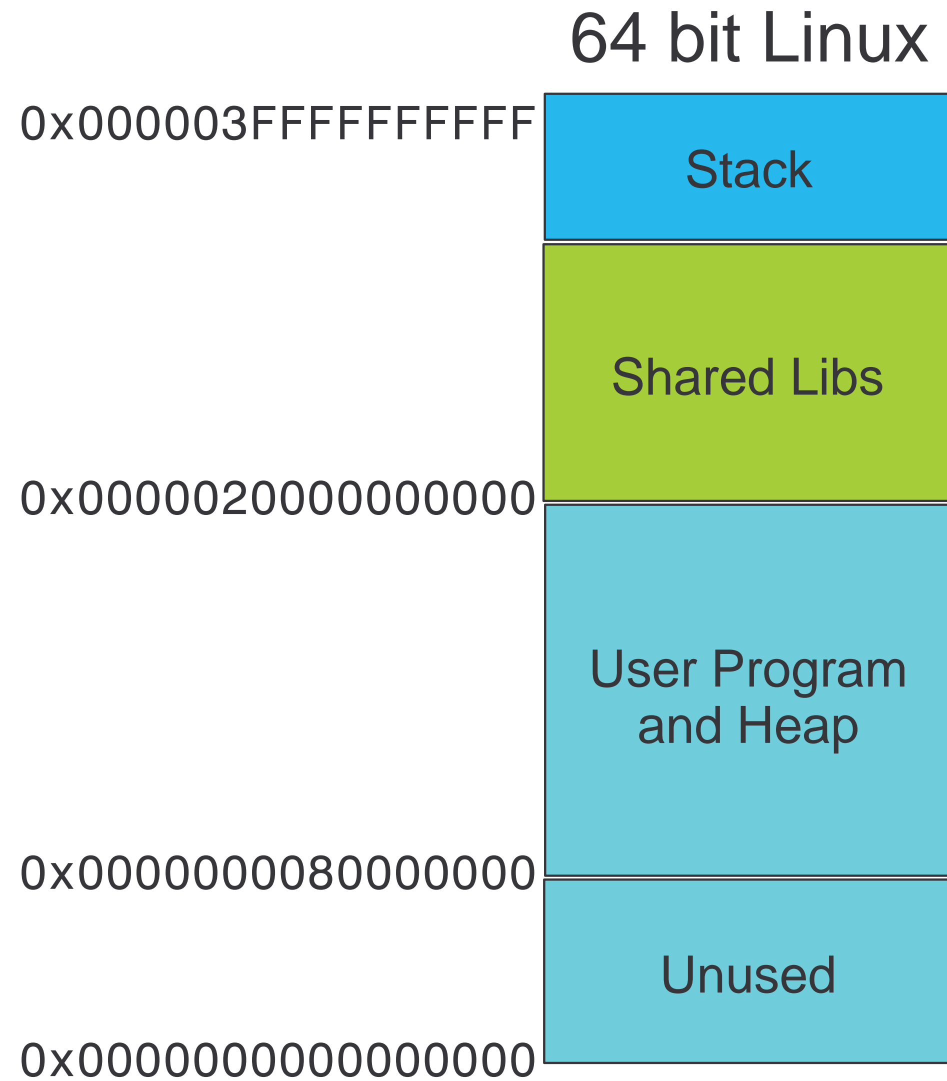
SHARE
Technology • Connections • Results

- Kernel binary loader
  - Gets invoked via the exec..() system call
  - Interprets ELF header and program headers
  - Maps ELF segments and stack into address space
  - Loads ELF program interpreter - if present
  - Passes control to entry point (executable/interpreter)
- Dynamic loader (ld.so)
  - Used as program interpreter for dyn. linked applications
  - Maps all required dynamic libraries
  - Resolves dynamic relocations, performs library init calls
  - Passes control to entry point

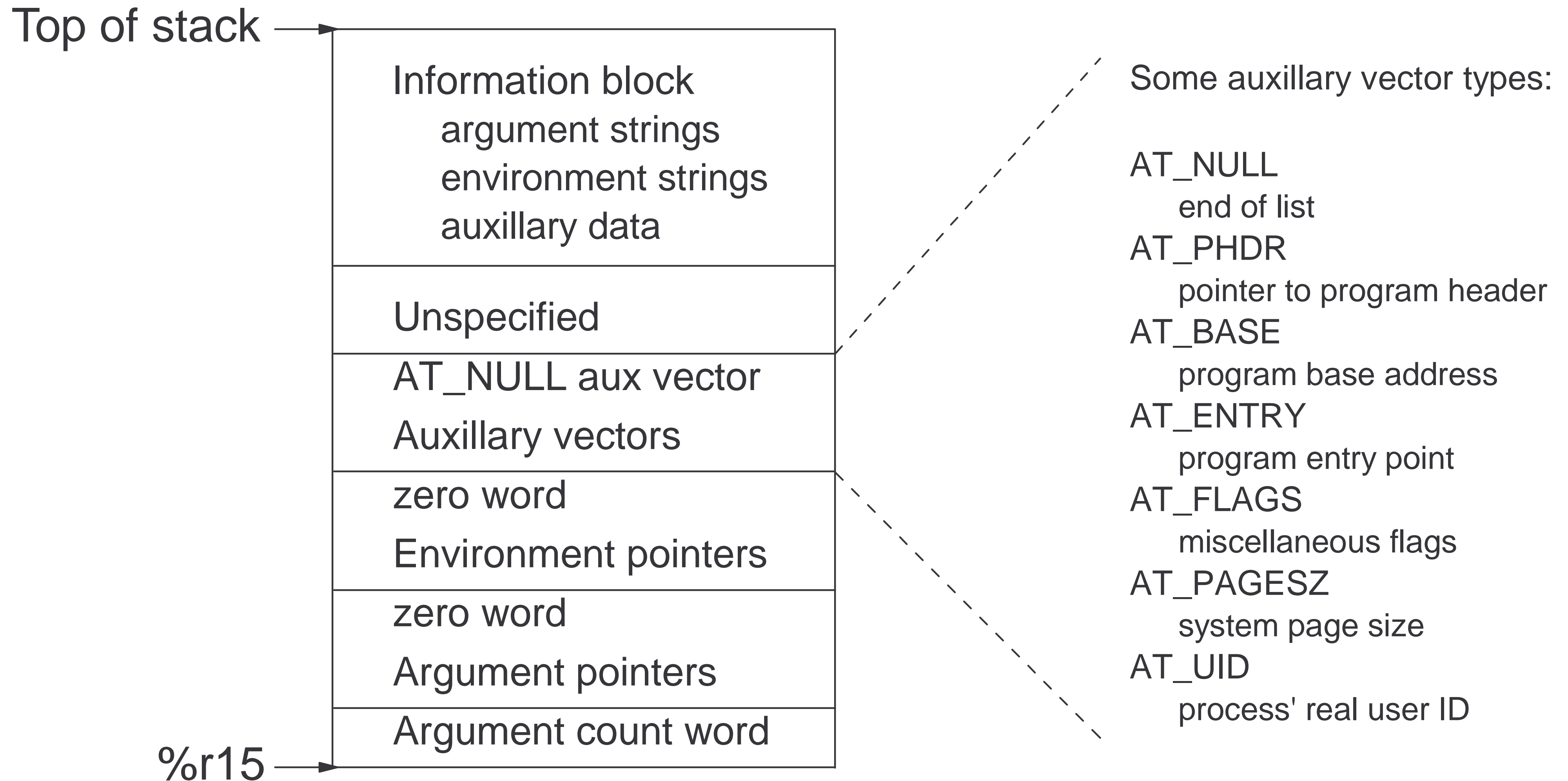
Address Space Layout

SHARE
Technology • Connections • Results





Initial Process Stack



Dynamic Linking

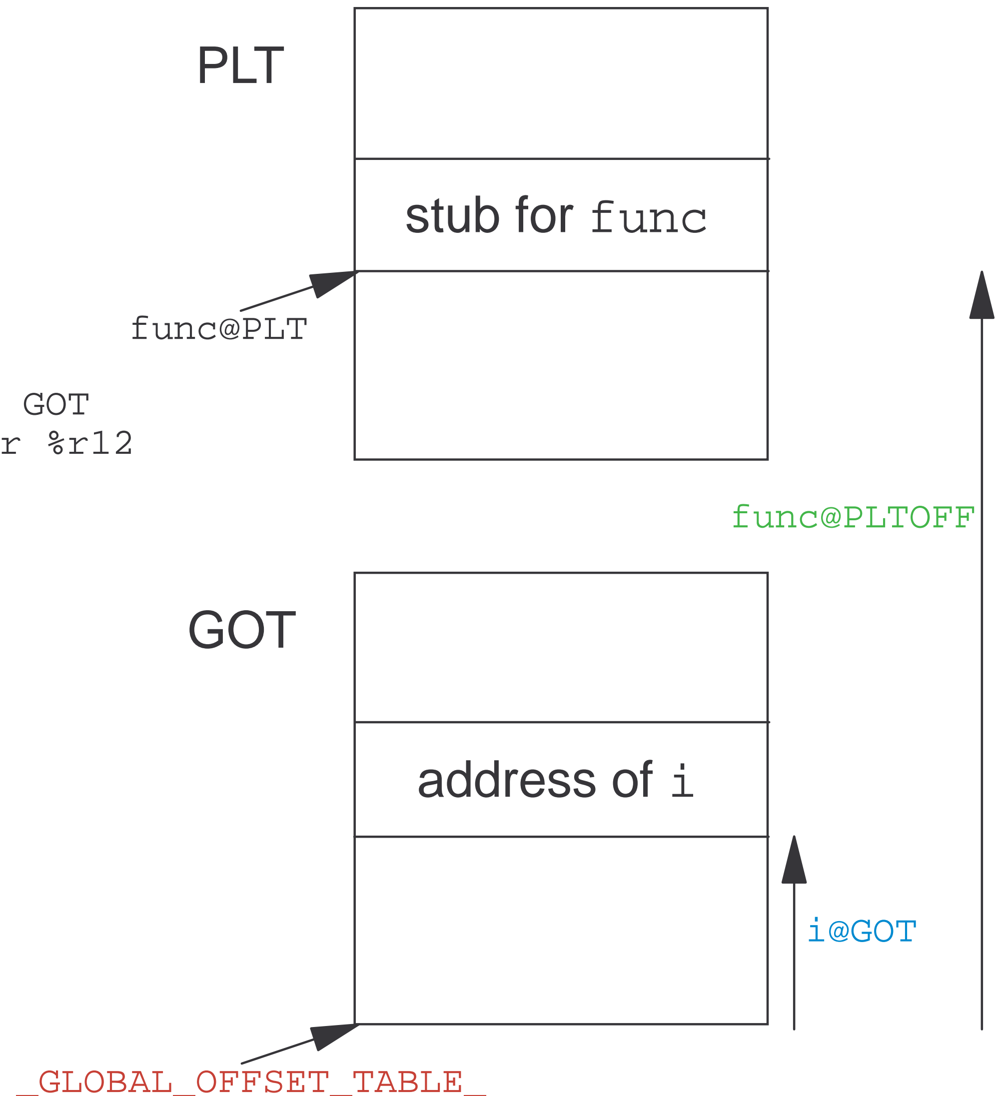
- Special requirements
  - Code/data references resolved at load-time
  - Late binding defers library loading until first use
  - Allow loading library to different virtual addresses
    - Should not make code segment impure
- Solution
  - Compiler generates position-independent code
    - Avoids absolute references to own code/data
  - Linker/loader maintain Global Offset Table
    - Avoids absolute references to foreign data
  - Procedure Linkage Table allows late binding
    - Defers references to foreign code until first use

Position-Independent Code

SHARE
Technology • Connections • Results

```
extern int i;
extern void func (int);
void test (void){ func (i); }

test:
        stm     %r12,%r15,48(%r15)
        basr    %r13,0
.L2:
        l       %r12,.L3-.L2(%r13)
        l       %r3,.L4-.L2(%r13)
        ahi     %r15,-96
        la      %r12,0(%r12,%r13)
        l       %r1,i@GOT(%r12)
        l       %r2,0(%r1)
        bas     %r14,0(%r3,%r12)
        l       %r4,152(%r15)
        lm      %r12,%r15,144(%r15)
        br      %r4
        .align  4
.L4:
        .long   func@PLTOFF
.L3:
        .long   _GLOBAL_OFFSET_TABLE_-.L2
        .align  2
```

Set up GOT pointer %r12



Position-Independent Code (cont.)

SHARE Technology • Connections • Results

```
00000000 <test>:
   0:    90 cf f0 30             stm     %r12,%r15,48(%r15)
   4:    0d d0                   basr    %r13,%r0
   6:    58 c0 d0 2a             l       %r12,42(%r13)
   a:    58 30 d0 26             l       %r3,38(%r13)
   e:    a7 fa ff a0             ahi     %r15,-96
  12:    41 cc d0 00             la      %r12,0(%r12,%r13)
  16:    58 10 c0 00             l       %r1,0(%r12)
                         18: R_390_GOT12 i
  1a:    58 20 10 00             l       %r2,0(%r1)
  1e:    4d e3 c0 00             bas     %r14,0(%r3,%r12)
  22:    58 40 f0 98             l       %r4,152(%r15)
  26:    98 cf f0 90             lm      %r12,%r15,144(%r15)
  2a:    07 f4                   br      %r4
  2c:    00 00 00 00             .long   0x00000000
                         2c: R_390_PLTOFF32      func
  30:    00 00 00 00             .long   0x00000000
                         30: R_390_GOTPC _GLOBAL_OFFSET_TABLE_+0x2a
```

```
Relocation types:

  R_390_GOT12:      Replace low 12 bits with GOT slot offset
  R_390_PLTOFF32:   Replace with offset of PLT entry to GOT
  R_390_GOTPC:      Replace with offset of GOT to current PC
```

Debugging

- Debugging requirements
  - Access current registers (all thread), memory contents
  - Reconstruct function call chain (stack backtrace)
  - Resolve symbolic addresses and data types
  - Live debugging (ptrace): single-step, watch points, ...
  - Post-mortem debugging (core files)
- Generating stack backtraces
  - Old-style: Run-time maintained stack frame backchain
  - New-style: DWARF-2 Call Frame Information metadata
  - glibc backtrace () function handles both

SHARE
Technology • Connections • Results

DWARF-2 Call Frame Information

Common Information Entry

Frame Description Entry

```
        .section        .eh_frame,"a",@progbits
.Lframe1:
        .4byte  .LECIE1-.LSCIE1 # Length of CIE
.LSCIE1:
        .4byte  0x0     # CIE Identifier Tag
        .byte   0x1     # CIE Version
        .ascii "\0"     # CIE Augmentation
        .uleb128 0x1    # CIE Code Alignment
        .sleb128 -4     # CIE Data Alignment
        .byte   0xe     # CIE RA Column
        .byte   0xc     # DW_CFA_def_cfa
        .uleb128 0xf
        .uleb128 0x60
        .align  4
.LECIE1:
```

```
main:
.LFB12:
        # basic block 0
        stm     %r13,%r15,52(%r15)
.LCFI0:
        basr    %r13,0
.L2:
        l       %r1,.L3-.L2(%r13)
        ahi     %r15,-96
.LCFI1:
```

```
.LSFDE1:
        .4byte  .LEFDE1-.LASFDE1   # FDE Length
.LASFDE1:
        .4byte  .LASFDE1-.Lframe1  # FDE CIE offset
        .4byte  .LFB12  # FDE initial location
        .4byte  .LFE12-.LFB12   # FDE address range
        .byte   0x4     # DW_CFA_advance_loc4
        .4byte  .LCFI0-.LFB12
        .byte   0x8f    # DW_CFA_offset, column 0xf
        .uleb128 0x9
        .byte   0x8e    # DW_CFA_offset, column 0xe
        .uleb128 0xa
        .byte   0x8d    # DW_CFA_offset, column 0xd
        .uleb128 0xb
        .byte   0x4     # DW_CFA_advance_loc4
        .4byte  .LCFI1-.LCFI0
        .byte   0xe     # DW_CFA_def_cfa_offset
        .uleb128 0xc0
        .align  4
.LEFDE1:
```

Resources

- GNU Compiler Collection home page
  http://gcc.gnu.org
- Linux on zSeries developerWorks page
  http://www.software.ibm.com/
  developerworks/opensource/linux390/index.html
- Linux on zSeries technical contact address
  linux390@de.ibm.com