ELF Relocatable Object File Format

ELF header (16 B)		Bootstrapping information for file
`.text`		Machine code of compiled module
`.rodata`		Read-only data (e.g., `printf` format strings, jump tables)
`.data`		Initialized global / static variables
`.bss`		Uninitialized static variables + those initialized to 0
`.symtab`		Symbol table
`.rel.text`		List of `.text` locations that need to be modified
`.rel.data`		List of `.data` locations that need to be modified
`.debug`	optional	Debugging symbol table
`.line`	optional	Mapping between source line #s and `.text` instructions
`.strtab`		String table for symbols in `.symtab`, `.debug`, and section names
Section Header Table		Fixed-size entries describing each section

The Need for Relocation

- When a source file is compiled into an object module, it has a “local coordinate system” (aka “module addresses” or “link-time addresses”) for both `.text` and `.data`.

The Need for Relocation

- When a source file is compiled into an object module, it has a “local coordinate system” (aka “module addresses” or “link-time addresses”) for both `.text` and `.data`.
- When multiple object modules are linked, the multiple `.text` sections need to be combined into a single `.text` section (a “global coordinate system”, aka “run-time addresses”) in the output file. Likewise for other sections.

The Need for Relocation

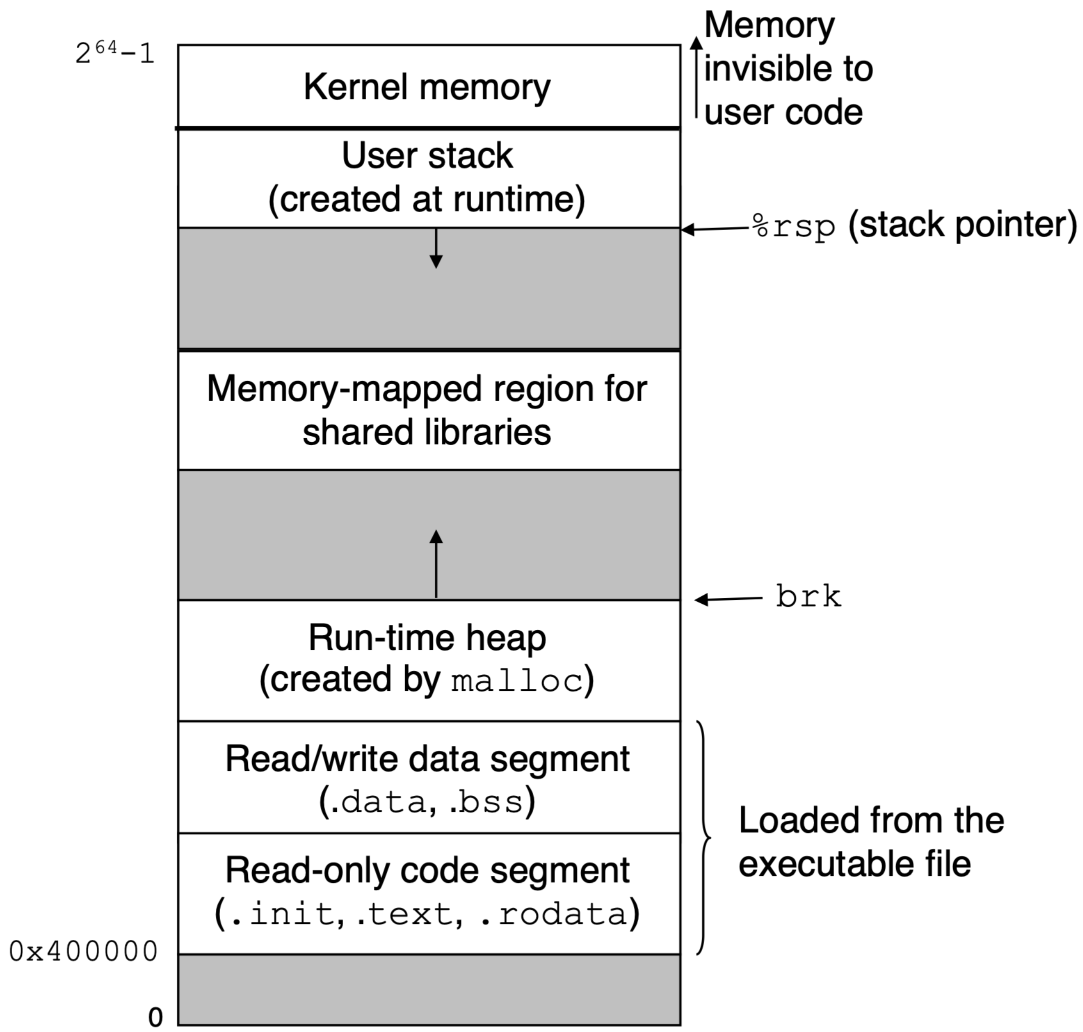
- When a source file is compiled into an object module, it has a "local coordinate system" (aka "module addresses" or "link-time addresses") for both `.text` and `.data`.
- When multiple object modules are linked, the multiple `.text` sections need to be combined into a single `.text` section (a "global coordinate system", aka "run-time addresses") in the output file. Likewise for other sections.
- This requires three separate steps:
  1. Relocating sections to their correct RT addresses.
  2. Computing the correct RT addresses for all symbol *definitions*.
  3. Modifying ("patching") symbol *references* so that they point to the correct RT addresses of the symbol definitions to which they have been resolved during symbol resolution.

ELF Executable Object File Format

ELF header (16 B)	NL	Bootstrapping information for file; includes program entry point.
Segment Header Table		Maps contiguous file sections into run-time memory segments.
`.init`	RO	Defines a small function `_init` that is called at program init.
`.text`		Machine code of compiled module
`.rodata`		Read-only data (e.g., `printf` format strings, jump tables)
`.data`	RW	Initialized global / static variables
`.bss`		Uninitialized static variables + those initialized to 0
`.symtab`	NL	Symbol table
`.debug`		Debugging symbol table
`.line`		Mapping between source line #s and `.text` instructions
`.strtab`		String table for symbols in `.symtab`, `.debug`, and section names
Section Header Table		Fixed-size entries describing each section

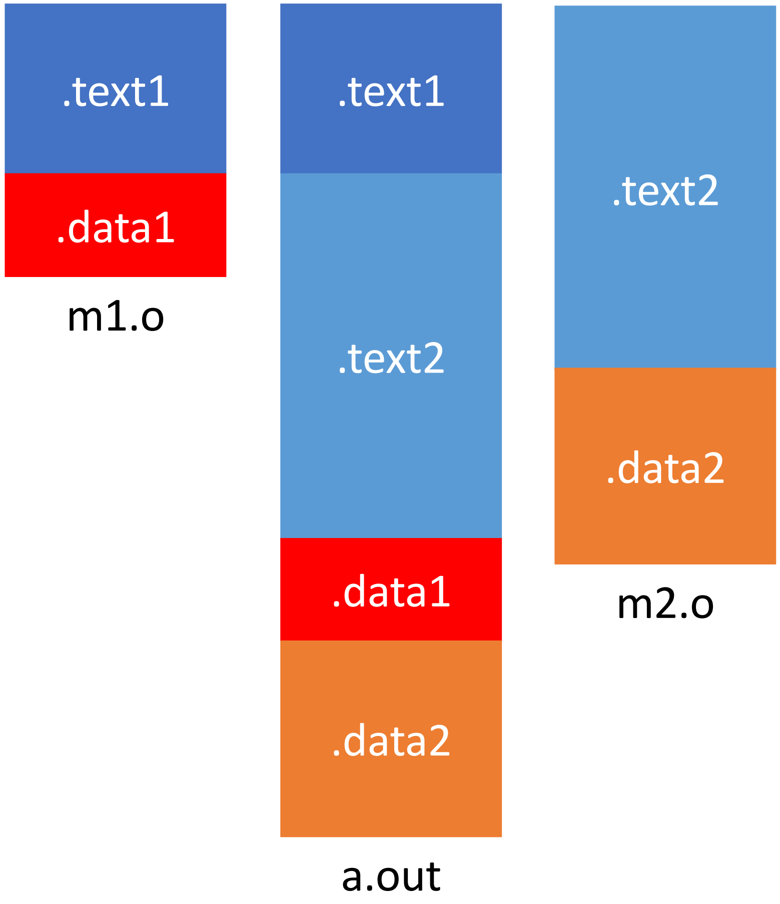
RO: Read-Only segment.
RW: Read/Write segment.
NL: Not Loaded.

Run-Time Layout of Virtual Address Space



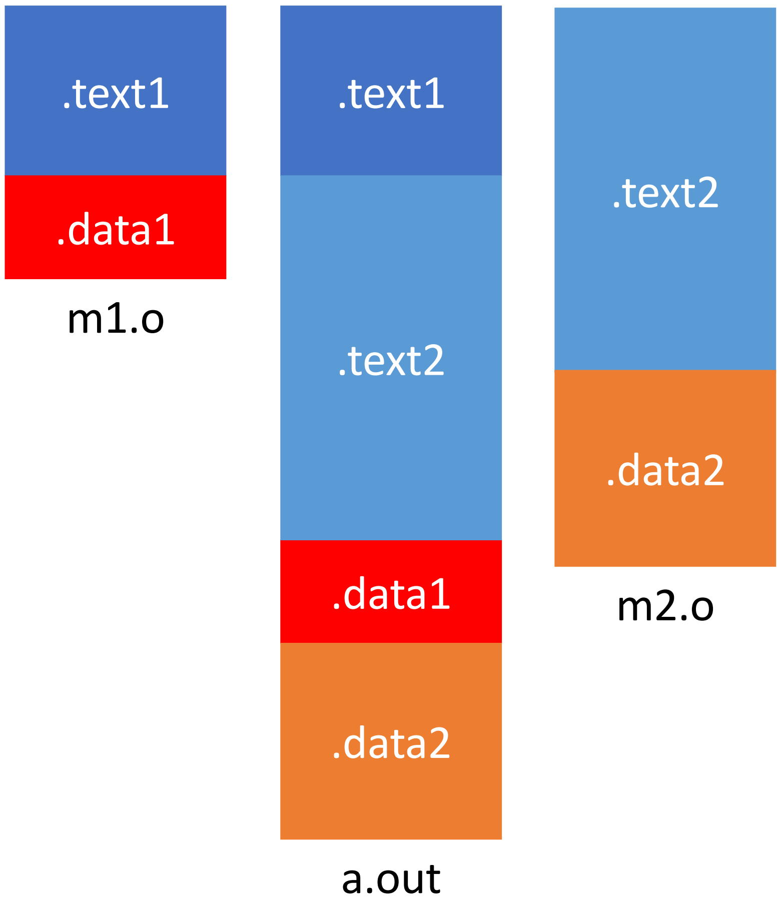
2^64−1
Kernel memory
Memory invisible to user code
User stack (created at runtime)
%rsp (stack pointer)
Memory-mapped region for shared libraries
brk
Run-time heap (created by malloc)
Read/write data segment (.data, .bss)
Read-only code segment (.init, .text, .rodata)
Loaded from the executable file
0x400000
0

Relocating Sections



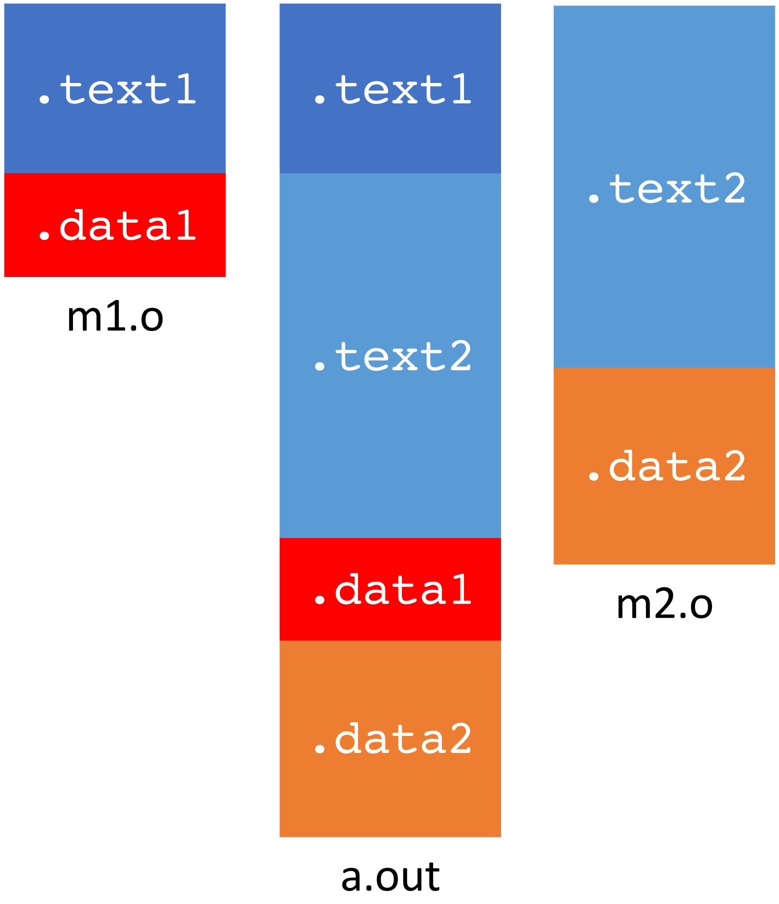
	LT address	Size (B)	RT address	RT offset
`.text1`	0	1000		
`.text2`	0	1400		
`.data1`	0	500		
`.data2`	0	1200		

Relocating Sections



	LT address	Size (B)	RT address	RT offset
`.text1`	0	1000	0	0
`.text2`	0	1400	1000	1000
`.data1`	0	500	2400	0
`.data2`	0	1200	2900	500

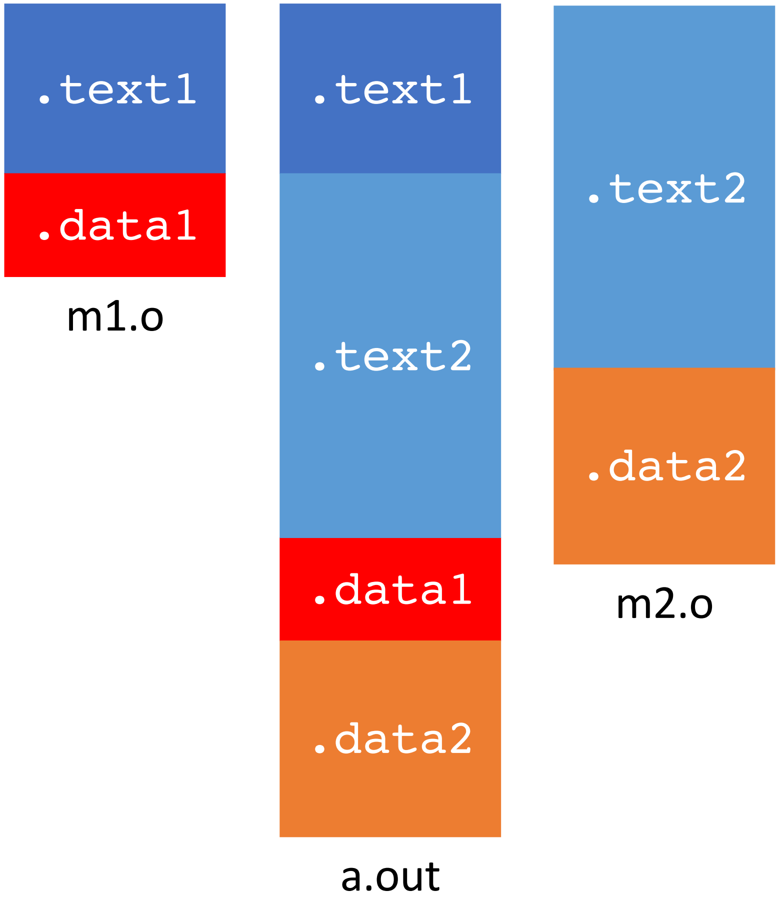
Computing Symbol Definition Addresses



	LT address	Size (B)	RT address	RT offset
.text1	0	1000	0	0
.text2	0	1400	1000	1000
.data1	0	500	2400	0
.data2	0	1200	2900	500

Module	Section	Symbol	LT offset	RT address
m1.o	.text1	foo	100	
	.data1	glob1	100	
m2.o	.text2	bar	100	
	.data2	glob2	100	

Computing Symbol Definition Addresses



	LT address	Size (B)	RT address	RT offset
.text1	0	1000	0	0
.text2	0	1400	1000	1000
.data1	0	500	2400	0
.data2	0	1200	2900	500

Module	Section	Symbol	LT offset	RT address
m1.o	.text1	foo	100	100
	.data1	glob1	100	2500
m2.o	.text2	bar	100	1100
	.data2	glob2	100	3000

Patching Symbol References

- We have a symbol reference $m.s$, i.e., the symbol s being referenced in module m.
    - Symbol resolution has matched it to symbol definition $n.t$.
    - We have computed the RT address of $n.t$.

Patching Symbol References

- We have a symbol reference $m.s$, i.e., the symbol s being referenced in module m.
  - Symbol resolution has matched it to symbol definition $n.t$.
  - We have computed the RT address of $n.t$.
- We just need to deal with two issues.
  - Does the reference $m.s$ even need to be patched?
  - If it does, how does it need to be updated?

Patching Symbol References

- We have a symbol reference $m.s$, i.e., the symbol s being referenced in module m.
  - Symbol resolution has matched it to symbol definition $n.t$.
  - We have computed the RT address of $n.t$.
- We just need to deal with two issues.
  - Does the reference $m.s$ even need to be patched?
  - If it does, how does it need to be updated?
- The relocation records in the `.rel.text` and `.rel.data` sections of module m provide the answers.
  - The symbol references that need to be patched are exactly the ones that are identified in the relocation records (which were generated by the compiler).
  - Part of the record describes the type of relocation needed.
  - ELF defines 32 different relocation types. Two major ones:
    - `R_X86_64_PC32`: Reference using 32-bit PC-relative address.
    - `R_X86_64_32`: Reference using 32-bit absolute address.