ILLINOIS TECH

College of Computing

CS 450 Operating Systems
Processes

Yue Duan

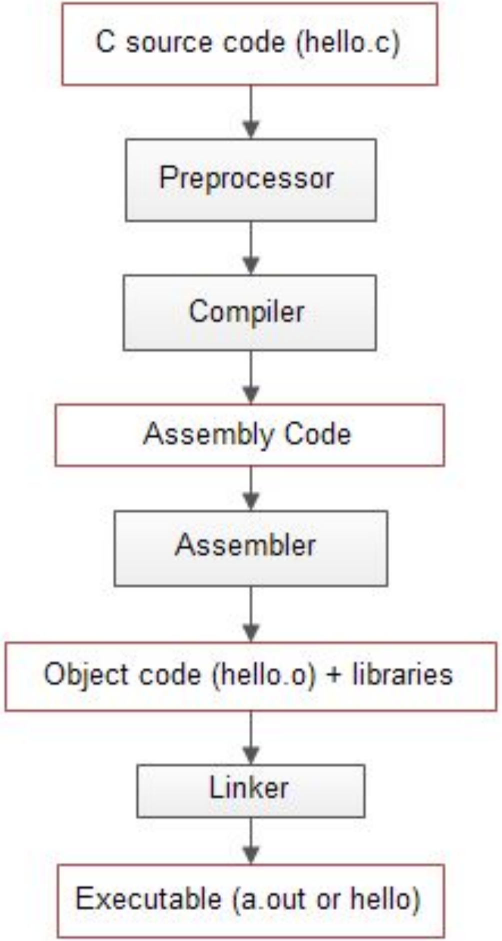
Process vs Program

- A program consists of code and data
  - specified in some programming language, such as C
- Typically stored in a file on disk
- “Running a program” = creating a process
- you can run a program multiple times!
  - one after another or even concurrently

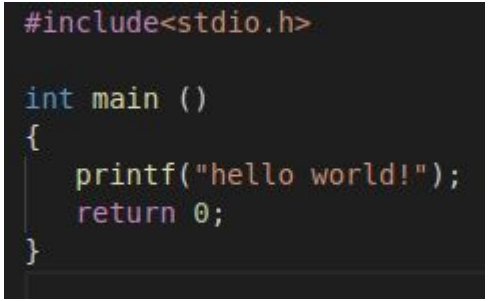
Executable

- An executable is a file containing:
  - executable code
    - CPU instructions
  - data
    - information manipulated by these instructions
- Obtained by compiling a program
  - and linking with libraries

Executable

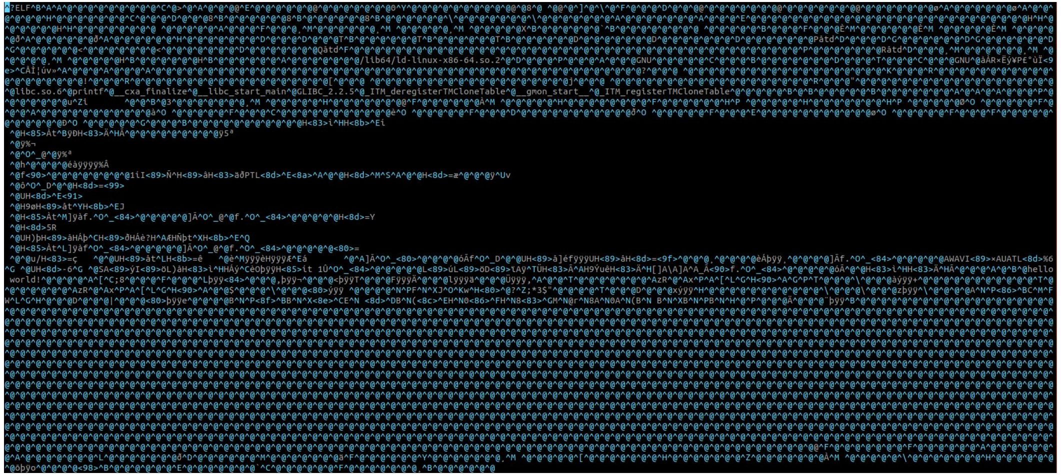

C source code (hello.c)
Preprocessor
Compiler
Assembly Code
Assembler
Object code (hello.o) + libraries
Linker
Executable (a.out or hello)

Executable



```
#include<stdio.h>

int main ()
{
    printf("hello world!");
    return 0;
}
```

Compiler, assembler, linker



```
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^C^@>^@^A^@^@^@@^E^@^@^@^@^@^@@^@^@^@^@^@^@0^Y^@^@^@^@^@^@^@^@^@^@@^@8^@ ^@@^@^]^@^\^@^F^@^@^@^D^@^@^@@^@^@^@^@^@^@^@@^@^@^@^@^@^@^@@^@^@^@^@^@^@^@ø^A^@^@^@^@^@^@ø^A^@^@^
@^@^@^@^H^@^@^@^@^@^@^C^@^@^@^D^@^@^@8^B^@^@^@^@^@^@8^B^@^@^@^@^@^@8^B^@^@^@^@^@^@^\^@^@^@^@^@^@^\^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^A^@^@^@^E^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^H^H^@
^@^@^@^@^@H^H^@^@^@^@^@^@^@^@ ^@^@^@^@^@^A^@^@^@^F^@^@^@¸^M^@^@^@^@^@^@¸^M ^@^@^@^@^@¸^M ^@^@^@^@^@X^B^@^@^@^@^@^@`^B^@^@^@^@^@^@^@ ^@^@^@^@^@^B^@^@^@^F^@^@^@È^M^@^@^@^@^@^@È^M ^@^@^@^@^@È^M ^@^@^@^@
^@ð^A^@^@^@^@^@^@ð^A^@^@^@^@^@^@^H^@^@^@^@^@^@^@^D^@^@^@^D^@^@^@T^B^@^@^@^@^@^@T^B^@^@^@^@^@^@T^B^@^@^@^@^@^@D^@^@^@^@^@^@^@D^@^@^@^@^@^@^@^D^@^@^@^@^@^@^@Påtd^D^@^@^@^D^G^@^@^@^@^@^@^D^G^@^@^@^@^@^@^D
^G^@^@^@^@^@^@<^@^@^@^@^@^@<^@^@^@^@^@^@^@^D^@^@^@^@^@^@^@Qåtd^F^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^P^@^@^@^@^@^@^@Råtd^D^@^@^@¸^M^@^@^@^@^@^@¸^M ^@
^@^@^@^@¸^M ^@^@^@^@^@H^B^@^@^@^@^@^@H^B^@^@^@^@^@^@^A^@^@^@^@^@^@^@/lib64/ld-linux-x86-64.so.2^@^D^@^@^@^P^@^@^@^A^@^@^@GNU^@^@^@^@^@^C^@^@^@^B^@^@^@^@^@^@^@^D^@^@^@^T^@^@^@^C^@^@^@GNU^@åÀR×Ëý¥P£°ùÏ<9
e>^CÅÌ¦úv»^A^@^@^@^A^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@?^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@K^@^@^@R^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@!^@^@^@R^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@[^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@j^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@R^@^@^@"^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@libc.so.6^@printf^@__cxa_finalize^@__libc_start_main^@GLIBC_2.2.5^@_ITM_deregisterTMCloneTable^@__gmon_start__^@_ITM_registerTMCloneTable^@^@^@^@^@^B^@^B^@^@^@^@^@^B^@^@^@^@^@^@^A^@^A^@^A^@^@^@^P^@
^@^@^@^@^@^@u^Zi       ^@^@^B^@3^@^@^@^@^@^@^@¸^M ^@^@^@^@^@^H^@^@^@^@^@^@@^F^@^@^@^@^@À^M ^@^@^@^@^@^H^@^@^@^@^@^@^@^F^@^@^@^@^@^@^H^P ^@^@^@^@^@^H^@^@^@^@^@^@^@^H^P ^@^@^@^@^@Ø^O ^@^@^@^@^@^F^@
^@^@^A^@^@^@^@^@^@^@^@^@^@^@à^O ^@^@^@^@^@^F^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@è^O ^@^@^@^@^@^F^@^@^@^D^@^@^@^@^@^@^@^@^@^@^@ð^O ^@^@^@^@^@^F^@^@^@^E^@^@^@^@^@^@^@^@^@^@^@ø^O ^@^@^@^@^@^F^@^@^@^F^@^@^@^@^@^
@^@^@^@^@^@Ð^O ^@^@^@^@^@^G^@^@^@^B^@^@^@^@^@^@^@^@^@^@H<83>ì^HH<8b>^Eí
 ^@H<85>Àt^BÿÐH<83>Ä^HÃ^@^@^@^@^@^@^@^@ÿ5ª
 ^@ÿ%¬
 ^@^O^_@^@ÿ%ª
 ^@h^@^@^@^@éàÿÿÿÿ%Â
 ^@f<90>^@^@^@^@^@^@^@^@1íI<89>Ñ^H<89>âH<83>äðPTL<8d>^E<8a>^A^@^@H<8d>^M^S^A^@^@H<8d>=æ^@^@^@ÿ^Uv
 ^@ô^O^_D^@^@H<8d>=<99>
 ^@UH<8d>^E<91>
 ^@H9øH<89>åt^YH<8b>^EJ
 ^@H<85>Àt^M]ÿàf.^O^_<84>^@^@^@^@^@]Ã^O^_@^@f.^O^_<84>^@^@^@^@^@H<8d>=Y
 ^@H<8d>5R
 ^@UH)þH<89>åHÁþ^CH<89>ðHÁè?H^AÆHÑþt^XH<8b>^E^Q
 ^@H<85>Àt^L]ÿàf^O^_<84>^@^@^@^@^@]Ã^O^_@^@f.^O^_<84>^@^@^@^@^@<80>=
 ^@^@u/H<83>=ç   ^@^@UH<89>åt^LH<8b>=ê   ^@è^MÿÿÿèHÿÿÿÆ^Eá       ^@^A]Ã^O^_<80>^@^@^@^@óÃf^O^_D^@^@UH<89>å]éfÿÿÿUH<89>åH<8d>=<9f>^@^@^@¸^@^@^@^@èÁþÿÿ¸^@^@^@^@]Ãf.^O^_<84>^@^@^@^@^@AWAVI<89>×AUATL<8d>%6
^G ^@UH<8d>-6^G ^@SA<89>ýI<89>öL)åH<83>ì^HHÁý^CèOþÿÿH<85>ít 1Û^O^_<84>^@^@^@^@^@L<89>úL<89>öD<89>ïAÿ^TÜH<83>Ã^AH9ÝuêH<83>Ä^H[]A\A]A^A_Ã<90>f.^O^_<84>^@^@^@^@^@óÃ^@^@H<83>ì^HH<83>Ä^HÃ^@^@^@^A^@^B^@hello
 world!^@^@^@^@^A^[^C;8^@^@^@^F^@^@^@^Lþÿÿ<84>^@^@^@,þÿÿ¬^@^@^@<þÿÿT^@^@^@FÿÿÿÄ^@^@^@lÿÿÿä^@^@^@Üÿÿÿ,^A^@^@^T^@^@^@^@^@^@^@^AzR^@^Ax^P^A^[^L^G^H<90>^A^G^P^T^@^@^@^\^@^@^@àýÿÿ+^@^@^@^@^@^@^@^@^@^@^T^@
^@^@^@^@^@^@^AzR^@^Ax^P^A^[^L^G^H<90>^A^@^@$^@^@^@^\^@^@^@<80>ýÿÿ ^@^@^@^@^N^PF^N^XJ^O^Kw^H<80>^@?^Z;*3$"^@^@^@^@^T^@^@^@D^@^@^@xýÿÿ^H^@^@^@^@^@^@^@^@^@^@^\^@^@^@\^@^@^@zþÿÿ^\^@^@^@^@A^N^P<86>^BC^M^F
W^L^G^H^@^@^@D^@^@^@|^@^@^@<80>þÿÿe^@^@^@^@B^N^P<8f>^BB^N^X<8e>^CE^N <8d>^DB^N(<8c>^EH^N0<86>^FH^N8<83>^GM^N@r^N8A^N0A^N(B^N B^N^XB^N^PB^N^H^@^P^@^@^@Ä^@^@^@¨þÿÿ^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@@^F^@^@^@^@^@^@^F^@^@^@^@^@^@^A^@^@^@^@^@^@^
@^A^@^@^@^@^@^@^L^@^@^@^@^@^@ð^D^@^@^@^@^@^@^M^@^@^@^@^@^@ä^F^@^@^@^@^@^@^Y^@^@^@^@^@^@¸^M ^@^@^@^@^@^[^@^@^@^@^@^@^H^@^@^@^@^@^@^Z^@^@^@^@^@^@À^M ^@^@^@^@^@^\^@^@^@^@^@^@^H^@^@^@^@^@^@
^@õþÿo^@^@^@^@<98>^B^@^@^@^@^@^E^@^@^@^@^@^@`^C^@^@^@^@^@^F^@^@^@^@^@^@¸^B^@^@^@^@^@
```

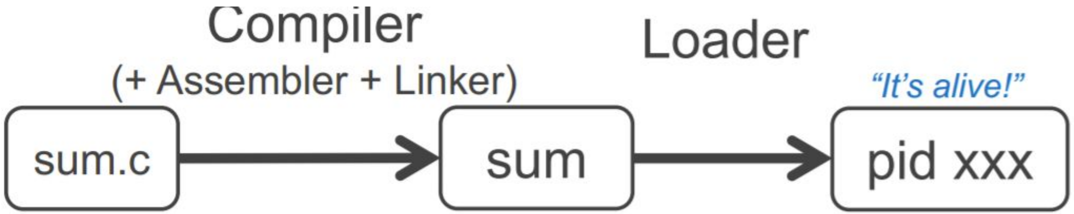
Process vs Program

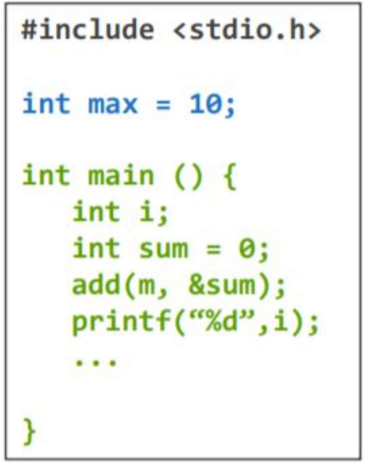
- An executable running on an abstraction of a computer:
  - Address Space (memory) + Execution Context (registers incl. PC and SP)
    - manipulated through machine instructions
  - Environment (clock, files, network, ...)
    - manipulated through system calls
- Current state is called “image” in Thompson/Ritchie paper
- A good abstraction:
  - is portable and hides implementation details
  - has an intuitive and easy-to-use interface
  - can be instantiated many times

Process != Program

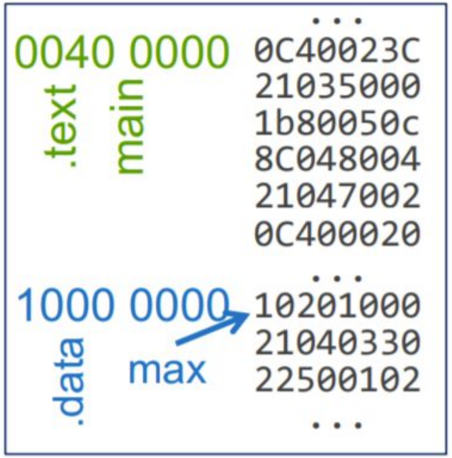
- A program is **passive**:
  - code + data
- A process is **alive**:
  - mutable data + registers + files + ...
- Same program can be run multiple time simultaneously (1 program, 2 processes)
  - > ./program &
  - > ./program &

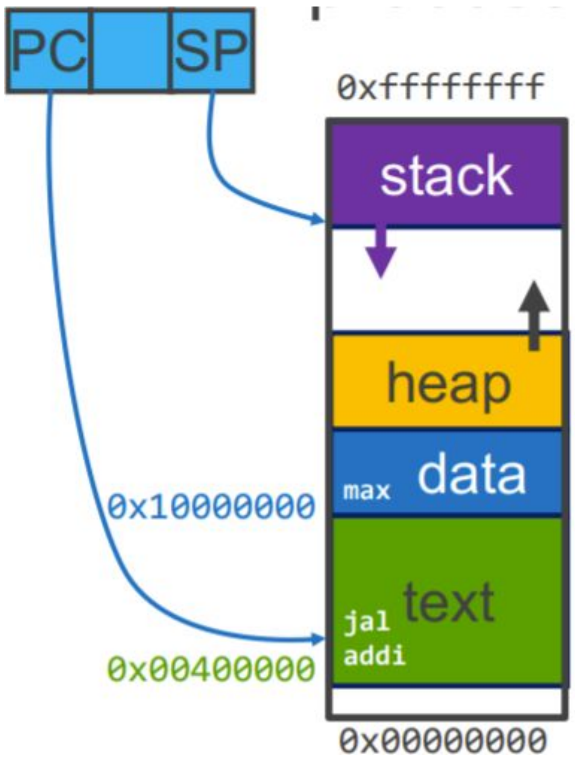
A Day in the Life of a Program



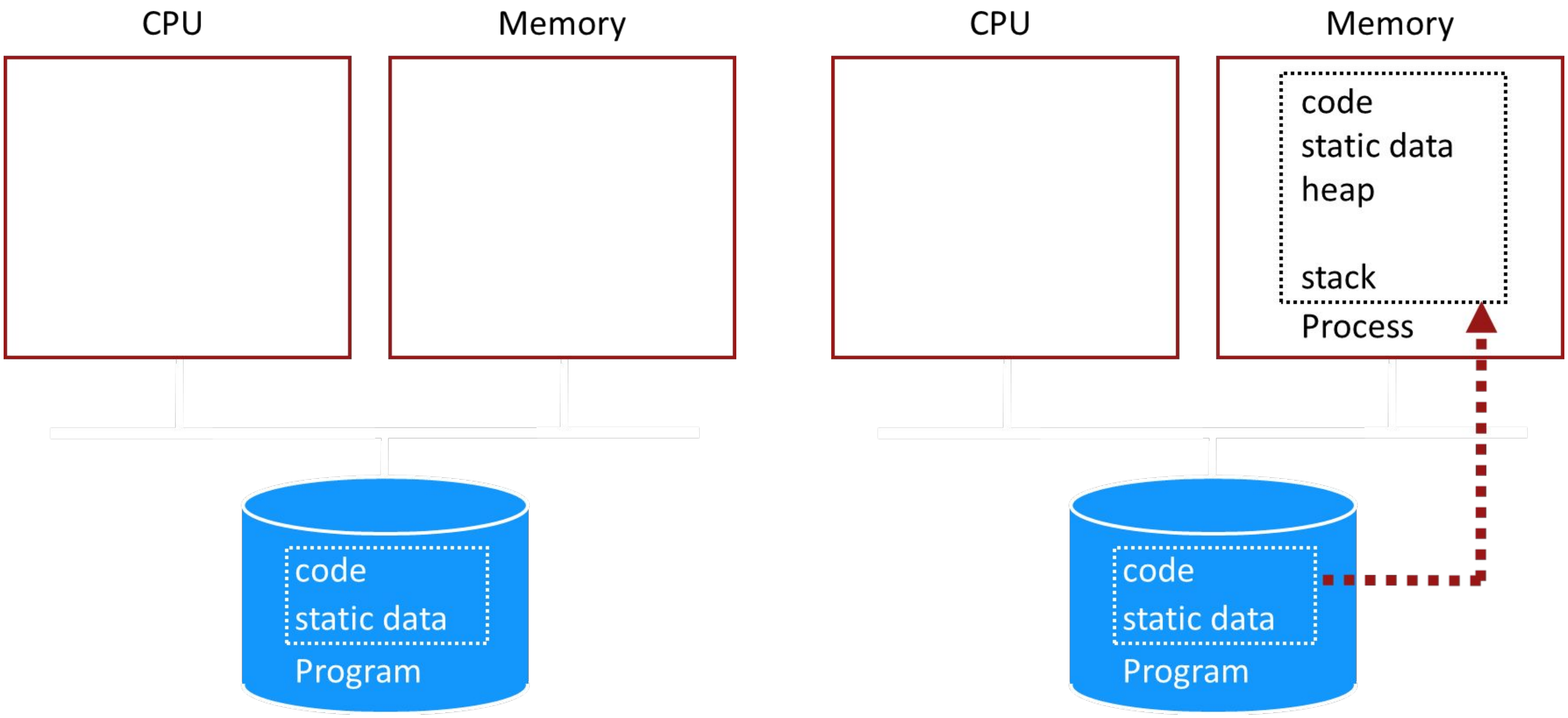


```
#include <stdio.h>

int max = 10;

int main () {
   int i;
   int sum = 0;
   add(m, &sum);
   printf(“%d”,i);
   ...

}
```

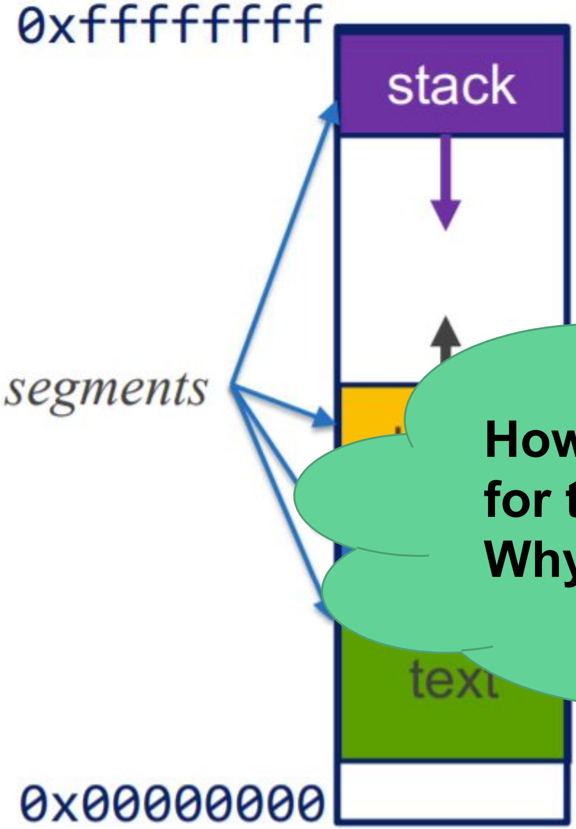




Process Creation



Logical view of process memory



- call stack

n (malloc)

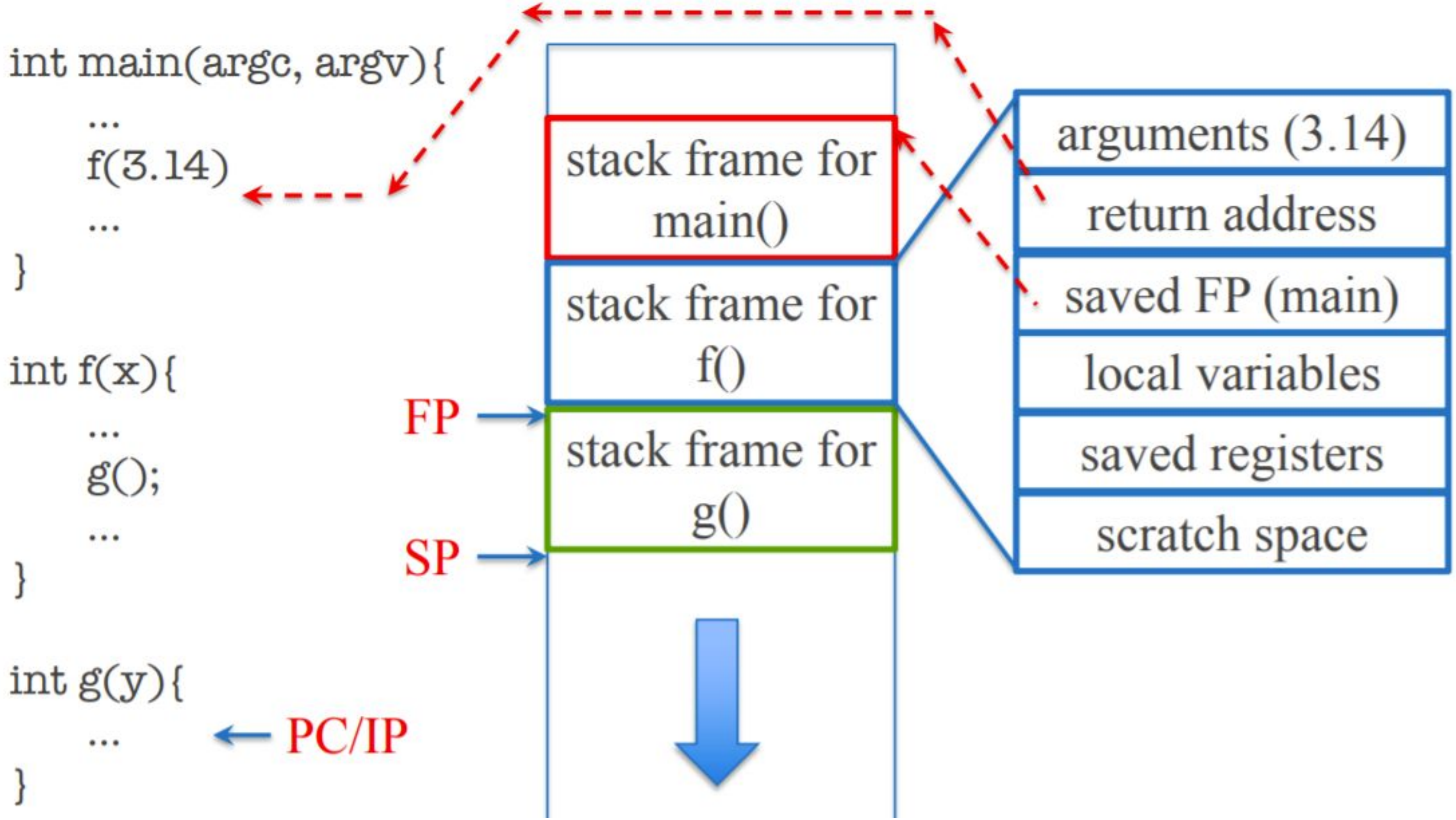
riables

contains code and

onstants

How many bits in an address for this CPU?
Why is address 0 not mapped?

Stack



```
int main(argc, argv){
      ...
      f(3.14)
      ...
}

int f(x){
      ...
      g();
      ...
}

int g(y){
      ...
}
```

PC/IP

FP

SP

stack frame for main()

stack frame for f()

stack frame for g()

arguments (3.14)

return address

saved FP (main)

local variables

saved registers

scratch space

Virtualizing the CPU

- Goal:
  - Give each process impression it alone is actively using CPU
- Resources can be shared in time and space
- Assume single uniprocessor
  - Time-sharing (multi-processors: advanced issue)
- Memory?
  - Space-sharing (later)
- Disk?
  - Space-sharing (later)

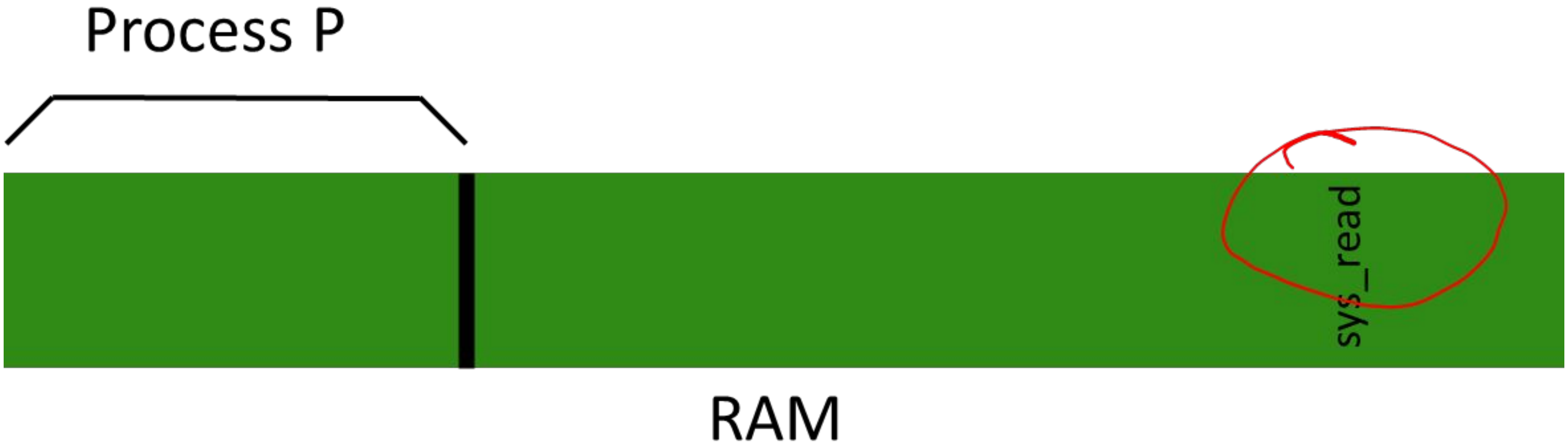
How to Provide Good CPU Performance?

- Direct execution
  - Allow user process to run directly on hardware
  - OS creates process and transfers control to starting point (i.e., main())
- Problems with direct execution?
  - Process could do something restricted
    - Could read/write other process data (disk, memory) or restricted device Process could run forever (slow, buggy, or malicious)
  - OS needs to be able to switch between processes
    - Process could do something slow (like I/O)
- OS wants to use resources efficiently and switch CPU to other process

Problem 1: Restricted Operations

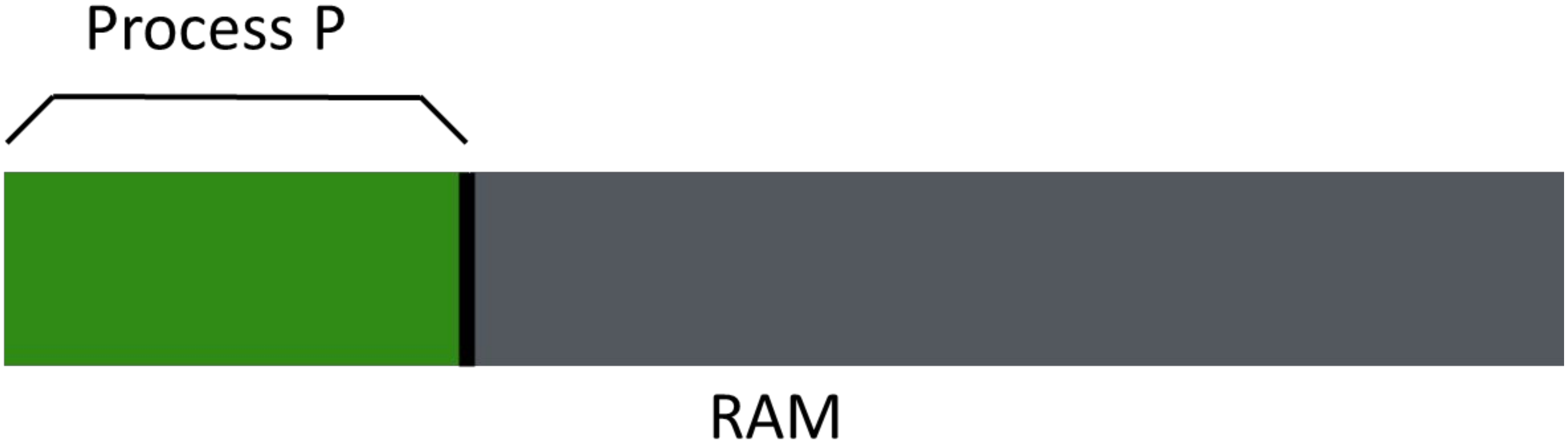
- How can we ensure user process can’t harm others?
- Solution: privilege levels supported by hardware (bit of status)
  - User processes run in user mode (restricted mode)
  - OS runs in kernel mode (not restricted)
    - Instructions for interacting with devices
    - Could have many privilege levels (advanced topic)
- How can process access device?
  - System calls (function call implemented by OS)
  - Change privilege level through system call (trap)

System Call



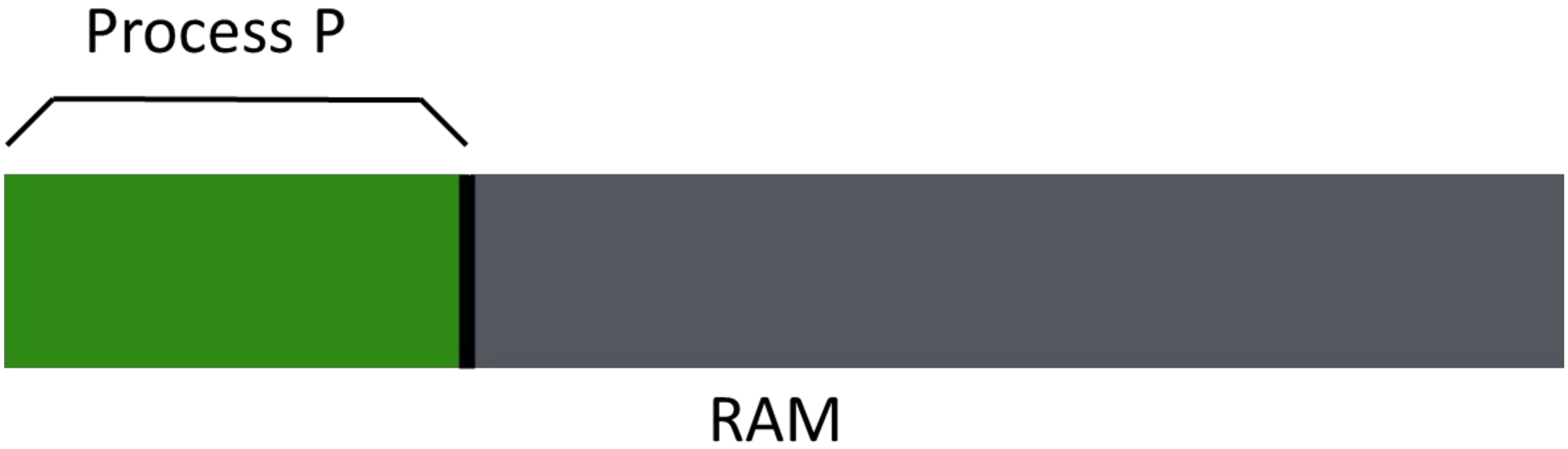
P wants to call read()

System Call



P can only see its own memory because of **user mode** (other areas, including kernel, are hidden)

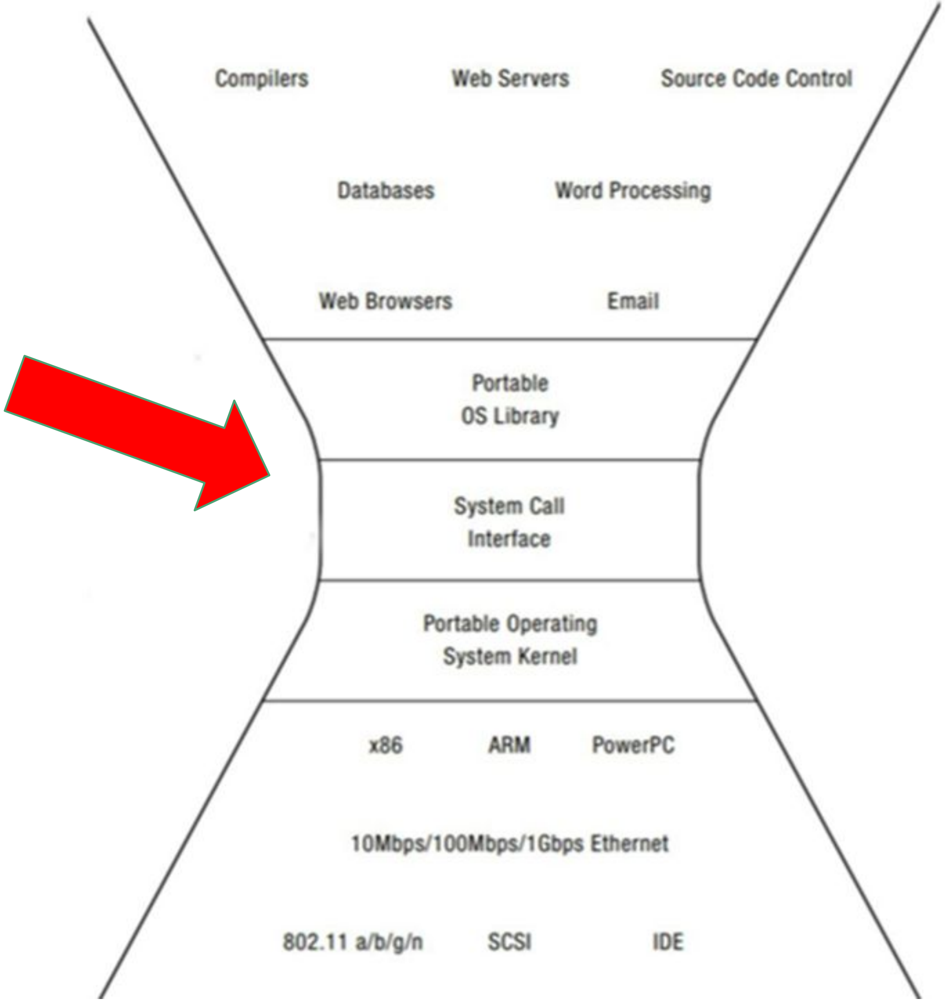
System Call



P wants to call read() but no way to call it directly

System Call

- A process runs on CPU
- Can access O.S. kernel through “system calls”
- Skinny interface - Why?



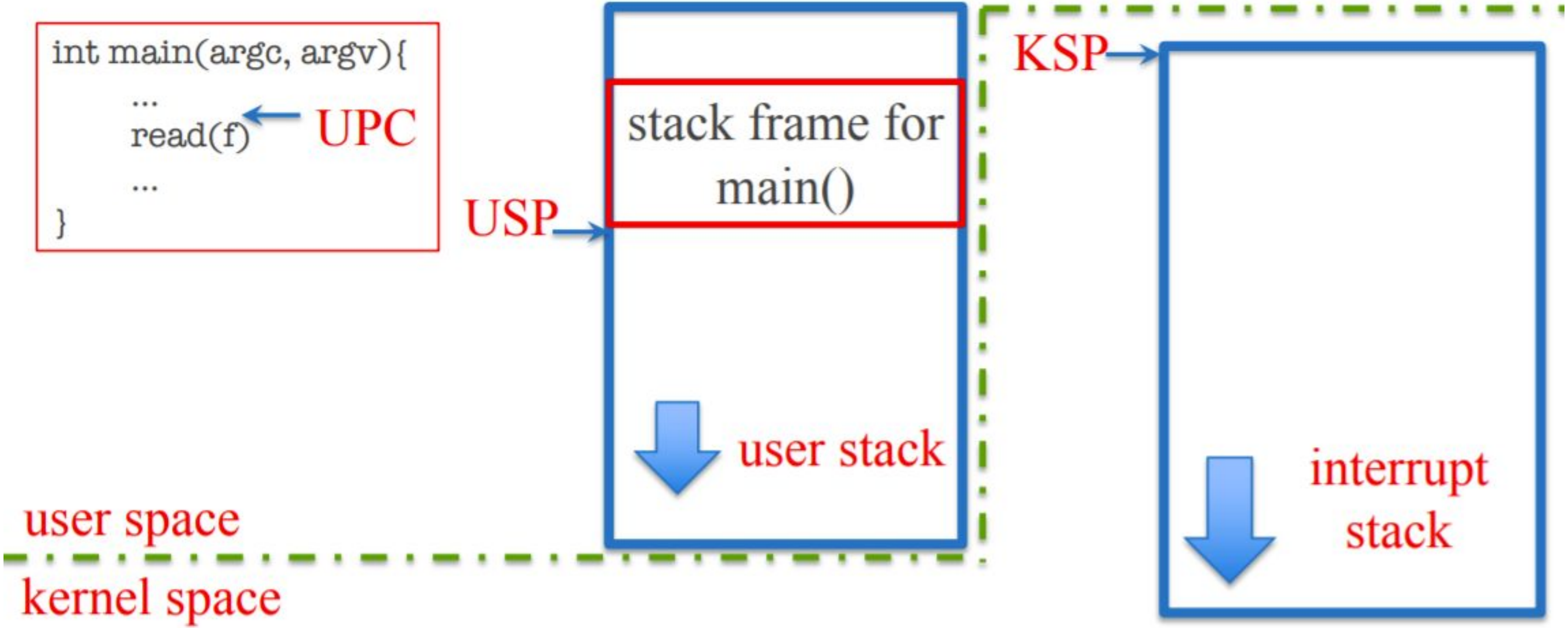
Why a “skinny” interface?

- Portability
  - easier to implement and maintain
  - e.g., many implementations of “Posix” interface
- Security
  - “small attack surface”: easier to protect against vulnerabilities

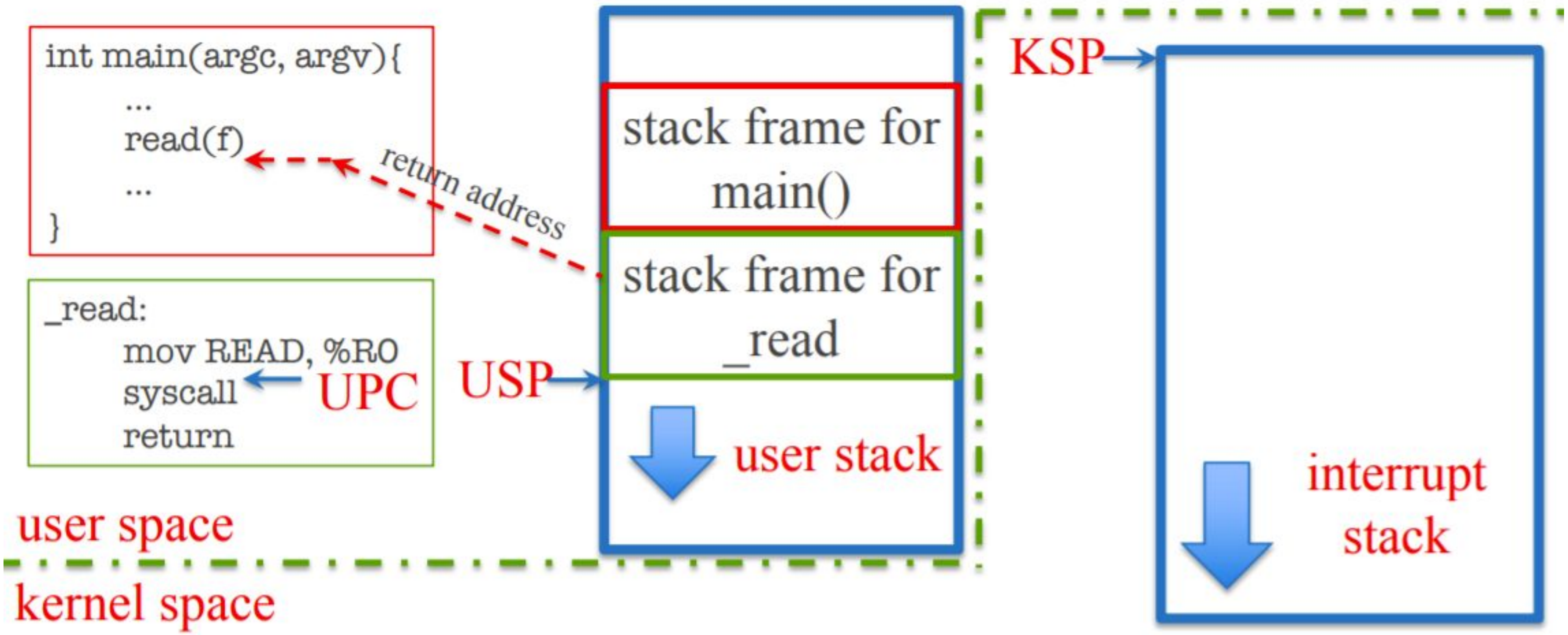
Executing a system call

- Process:
    - 1. Calls system call function in library
    - 2. Places arguments in registers and/or pushes them onto user stack
    - 3. Places syscall type in a dedicated register
    - 4. Executes **syscall** machine instruction
- Kernel:
    - 5. Executes syscall interrupt handler
    - 6. Places result in dedicated register
    - 7. Executes **return_from_interrupt**
- Process:
    - 8. Executes **return_from_function**

Executing read System Call



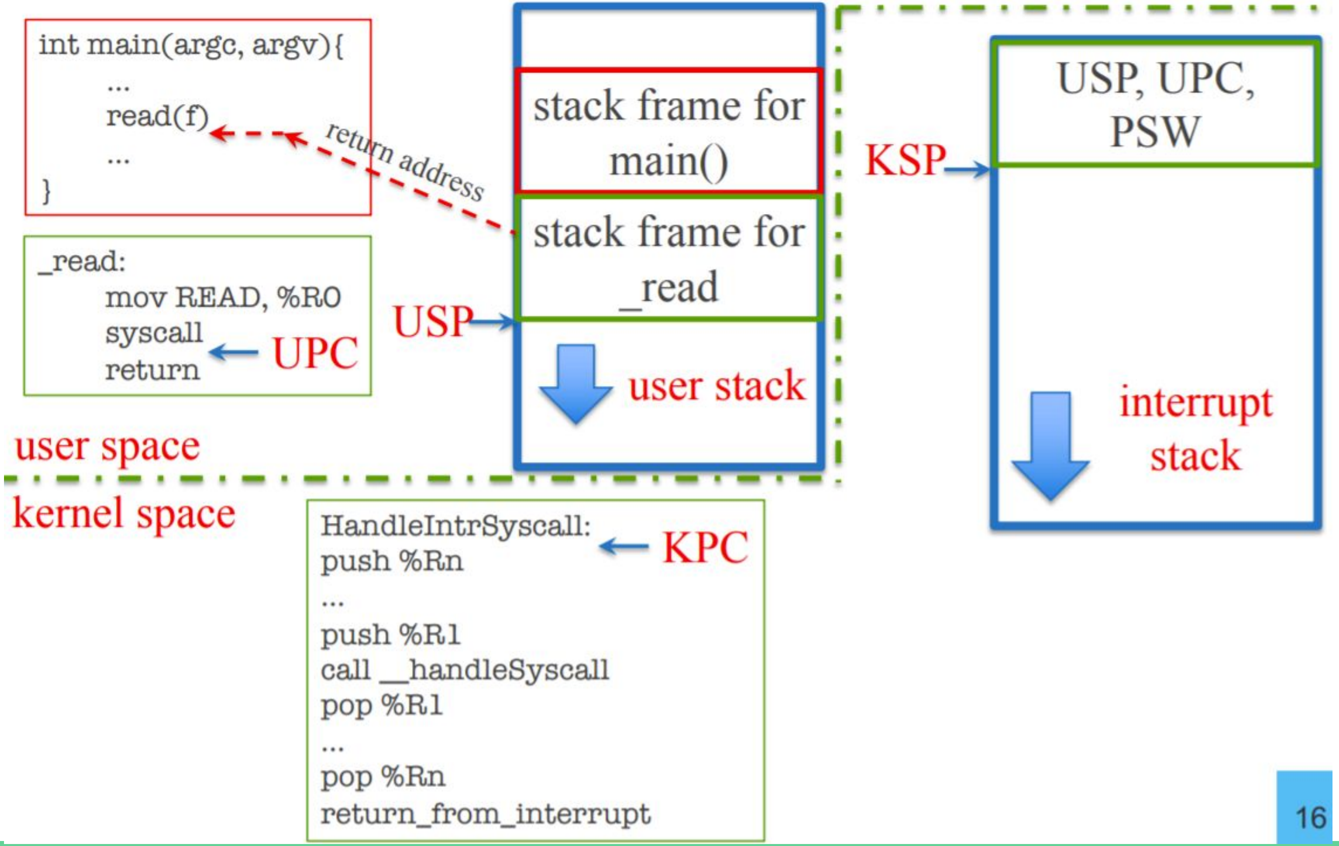
Executing read System Call



```
int main(argc, argv){
        ...
        read(f)
        ...
}
```

```
_read:
        mov READ, %R0
        syscall
        return
```

Executing read System Call

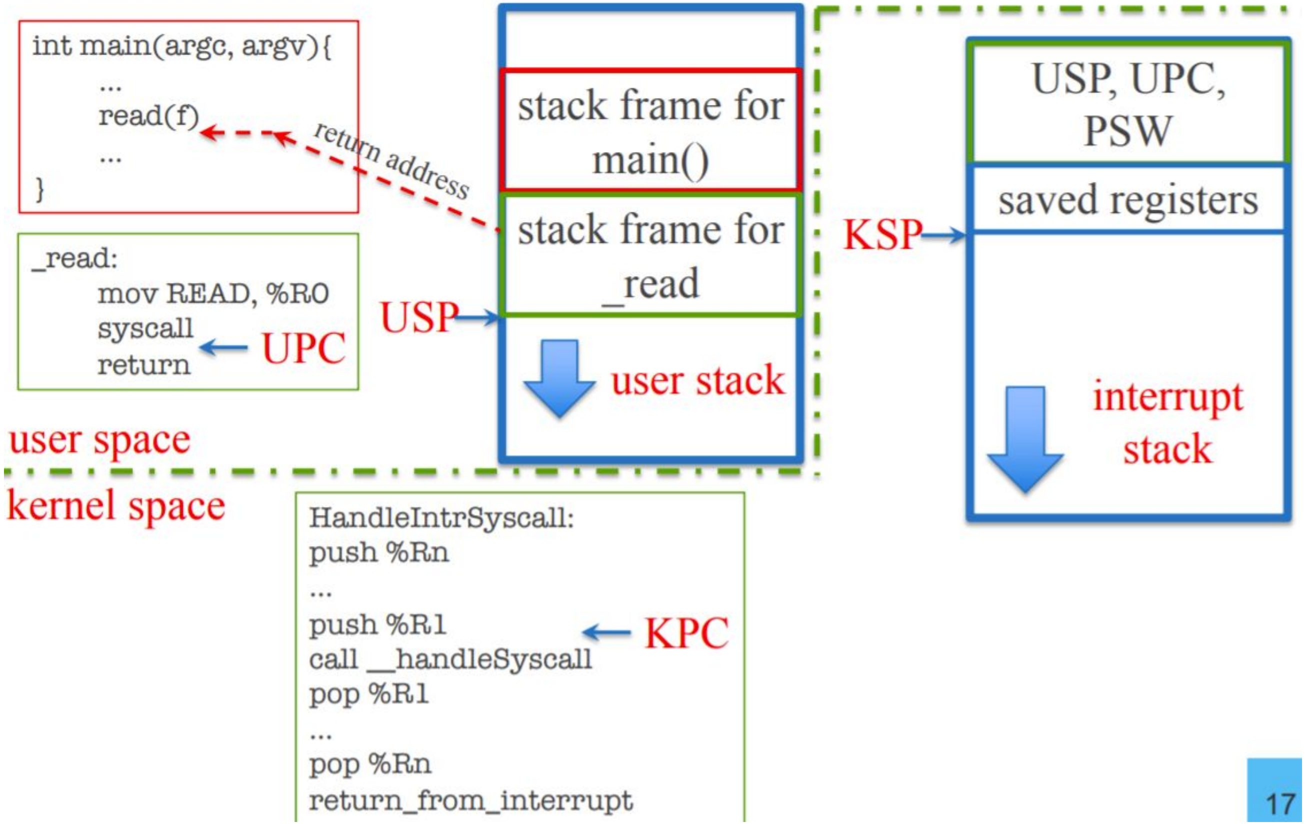


```
int main(argc, argv){
        ...
        read(f)
        ...
}
```

```
_read:
        mov READ, %R0
        syscall
        return
```

```
HandleIntrSyscall:
push %Rn
...
push %R1
call __handleSyscall
pop %R1
...
pop %Rn
return_from_interrupt
```

Executing read System Call

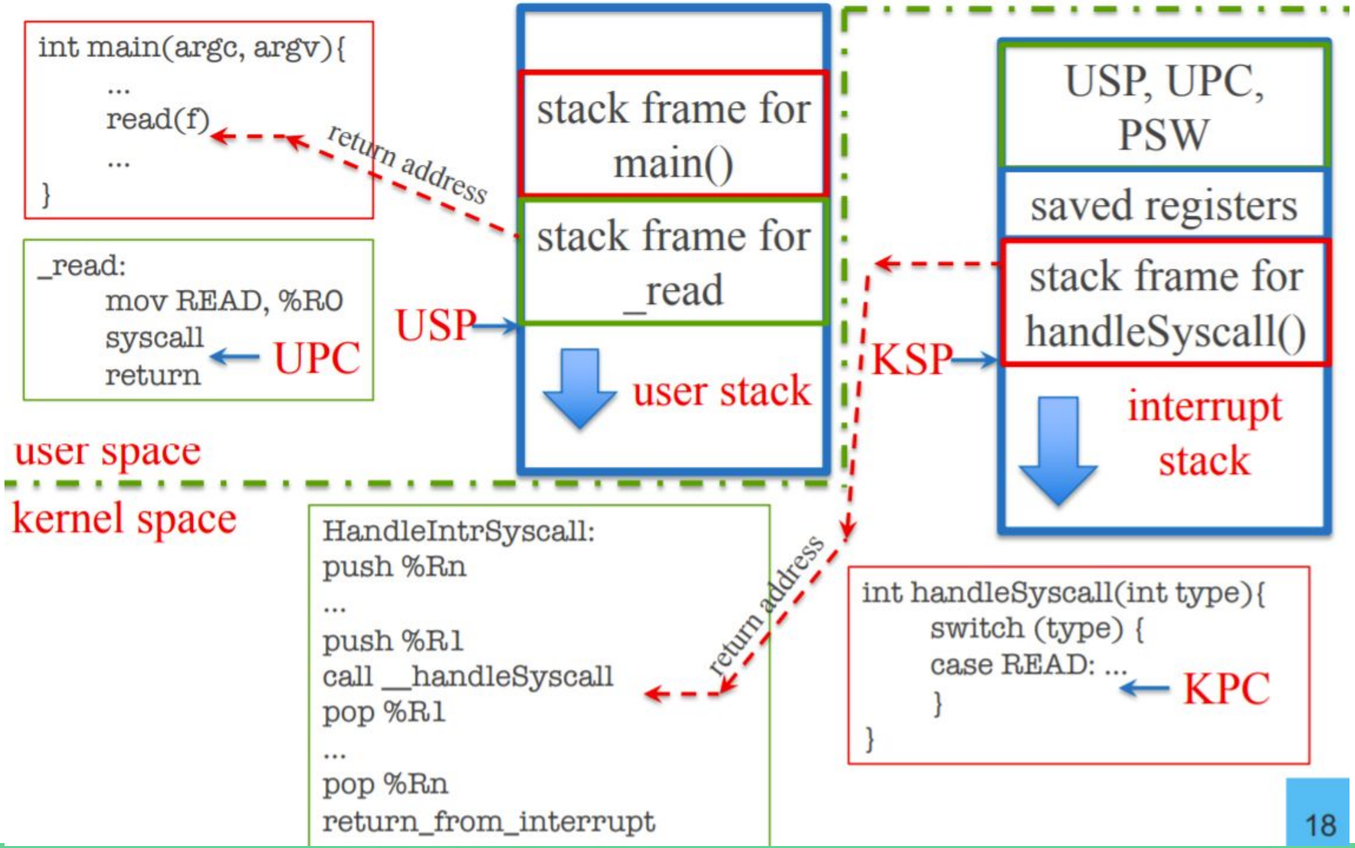


```
int main(argc, argv){
        ...
        read(f)
        ...
}
```

```
_read:
        mov READ, %R0
        syscall
        return
```

```
HandleIntrSyscall:
push %Rn
...
push %R1
call __handleSyscall
pop %R1
...
pop %Rn
return_from_interrupt
```

Executing read System Call



```
int main(argc, argv){
        ...
        read(f)
        ...
}
```

```
_read:
        mov READ, %R0
        syscall
        return
```

```
HandleIntrSyscall:
push %Rn
...
push %R1
call __handleSyscall
pop %R1
...
pop %Rn
return_from_interrupt
```

```
int handleSyscall(int type){
        switch (type) {
        case READ: ...
        }
}
```

What if read needs to “block”?

- read may need to block if
    - reading from terminal
    - reading from disk and block not in cache
    - reading from remote file server
    - etc

should run another process!

THANK YOU!