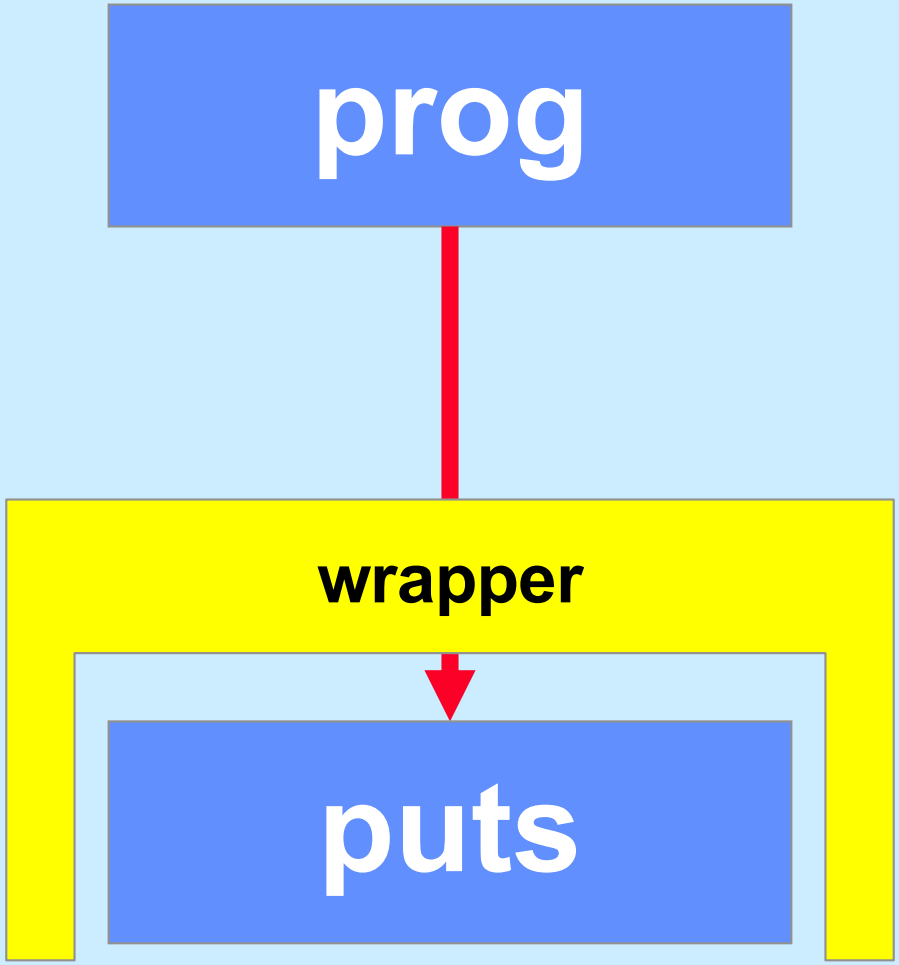
CS 33

Linking and Libraries (2)

Interpositioning



How To ...

```
int __wrap_puts(const char *s) {
  int __real_puts(const char *);

  write(2, "calling myputs: ", 16);
  return __real_puts(s);
}
```


Compiling/Linking It

```
$ cat tputs.c
int main() {
  puts("This is a boring message.");
  return 0;
}
$ gcc -o tputs -Wl,--wrap=puts tputs.c myputs.c
$ ./tputs
calling myputs: This is a boring message.
$
```

How To (Alternative Approach) …

```
#include <dlfcn.h>

int puts(const char *s) {
  int (*pptr)(const char *);

  pptr = (int(*)())dlsym(RTLD_NEXT, "puts");

  write(2, "calling myputs: ", 16);
  return (*pptr)(s);
}
```

What's Going On …

- gcc/ld
  - compiles code
  - does static linking
    - » searches list of libraries
    - » adds references to shared objects
- runtime
  - program invokes *ld-linux.so* to finish linking
    - » maps in shared objects
    - » does relocation and procedure linking as required
  - *dlsym* invokes *ld-linux.so* to do more linking
    - » RTLD_NEXT says to use the next (second) occurrence of the symbol

Delayed Wrapping

- LD_PRELOAD
  - environment variable checked by *ld-linux.so*
  - specifies additional shared objects to search (first) when program is started

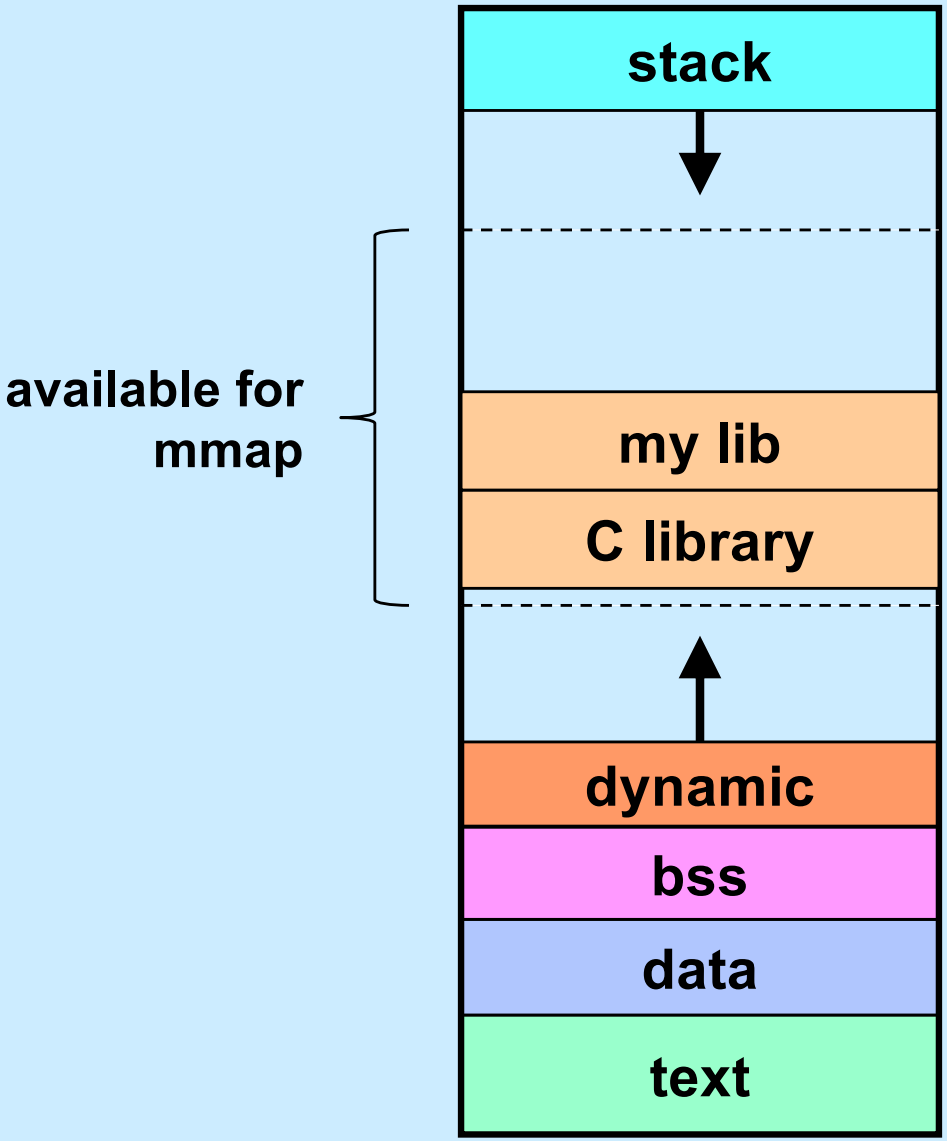
Environment Variables

- **Another form of exec**
  - `int execve(const char *filename, char *const argv[], char *const envp[]);`
- **envp is an array of strings, of the form**
  - **key=value**
- **programs can search for values, given a key**
- **example**
  - **PATH=~/bin:/bin:/usr/bin:/course/cs0330/bin**

Example

```
$ gcc -o tputs tputs.c
$ ./tputs
This is a boring message.
$ LD_PRELOAD=./libmyputs.so.1; export LD_PRELOAD
$ ./tputs
calling myputs: This is a boring message.
$
```

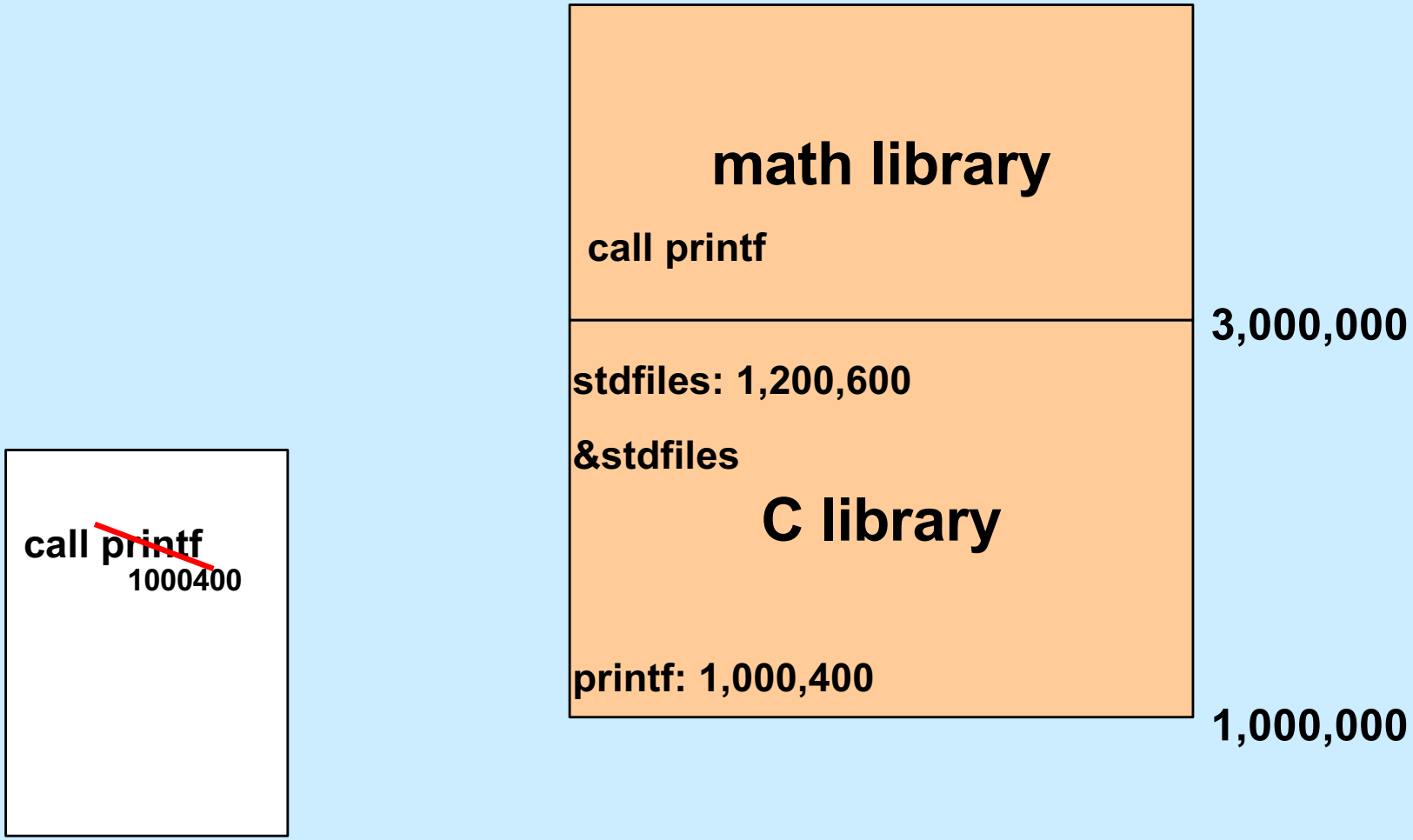
Mmapping Libraries



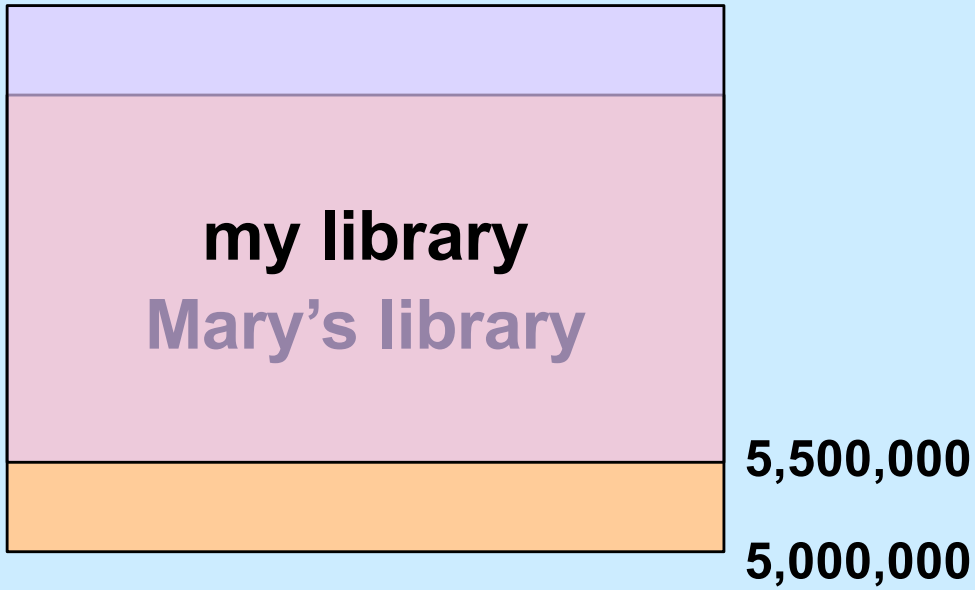
Problem

- How is relocation handled?

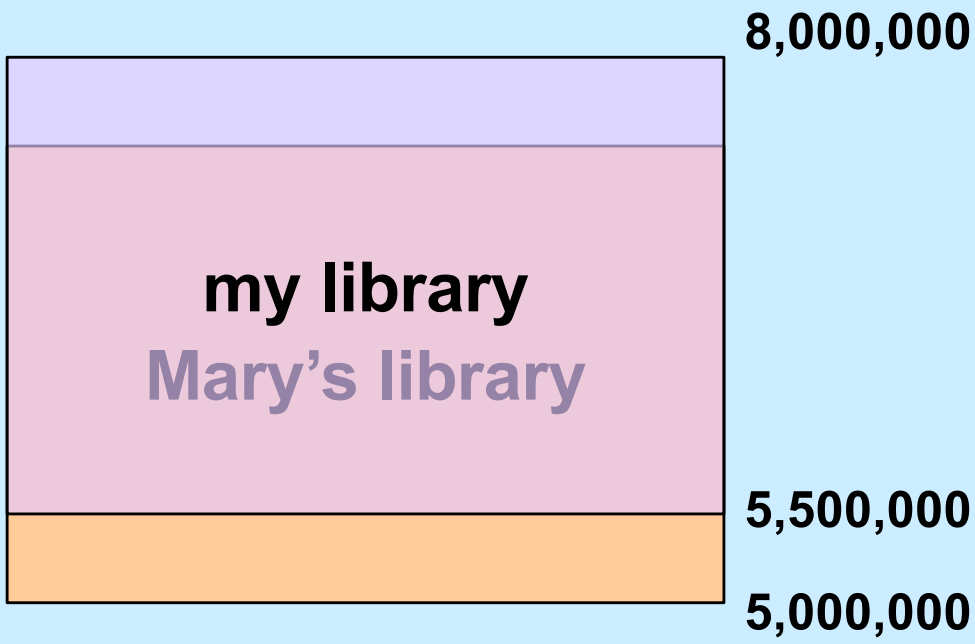
Pre-Relocation



But …



But …



Quiz 1

We need to relocate all references to Mary's library in my library. What option should we give to *mmap* when we map my library into our address space?

a) **the MAP_PRIVATE option**

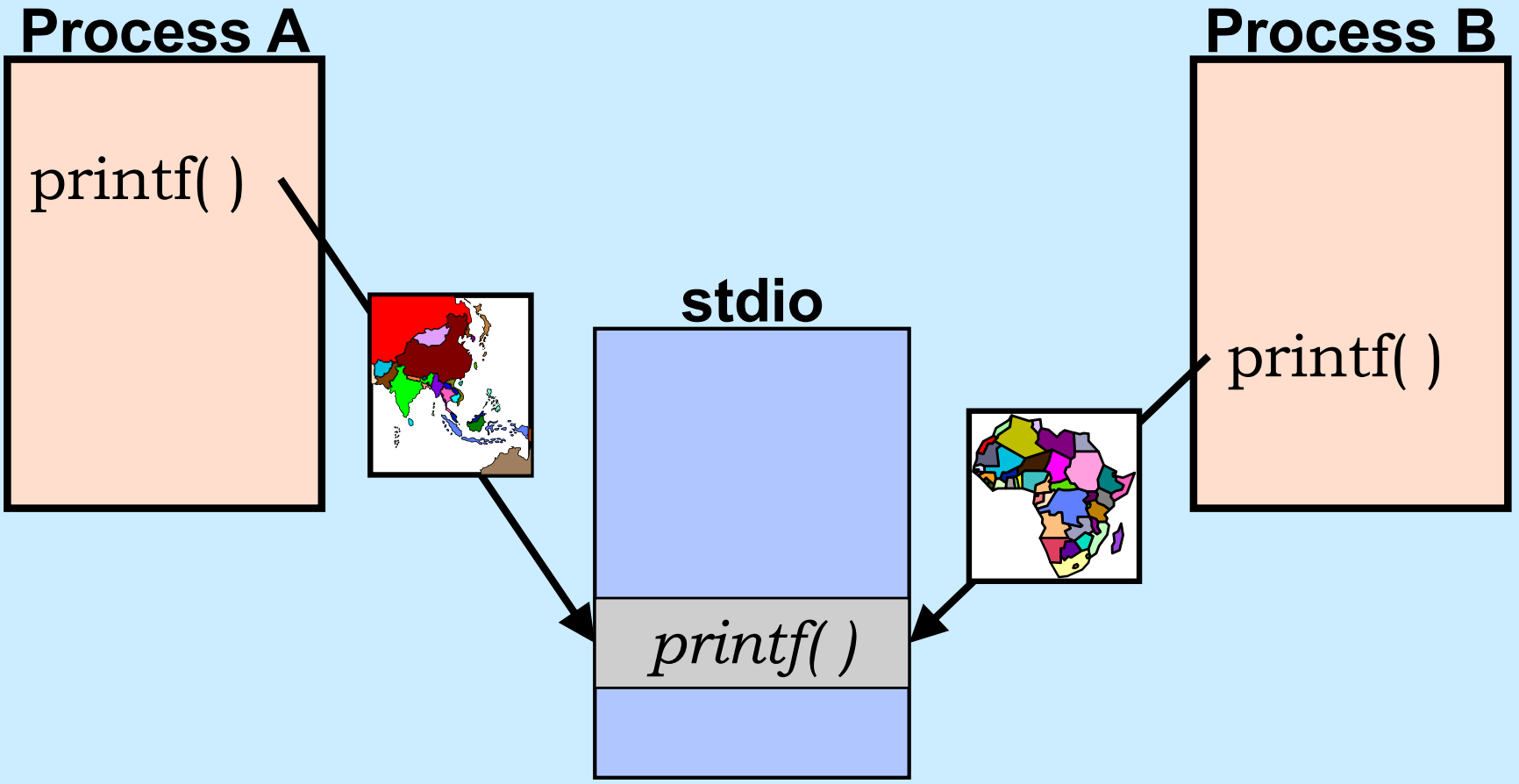
b) **the MAP_SHARED option**

c) **mmap can't be used in this situation**

Relocation Revisited

- Modify shared code to effect relocation
  - result is no longer shared!
- Separate shared code from (unshared) addresses
  - position-independent code (PIC)
  - code can be placed anywhere
  - addresses in separate private section
    - » pointed to by a register

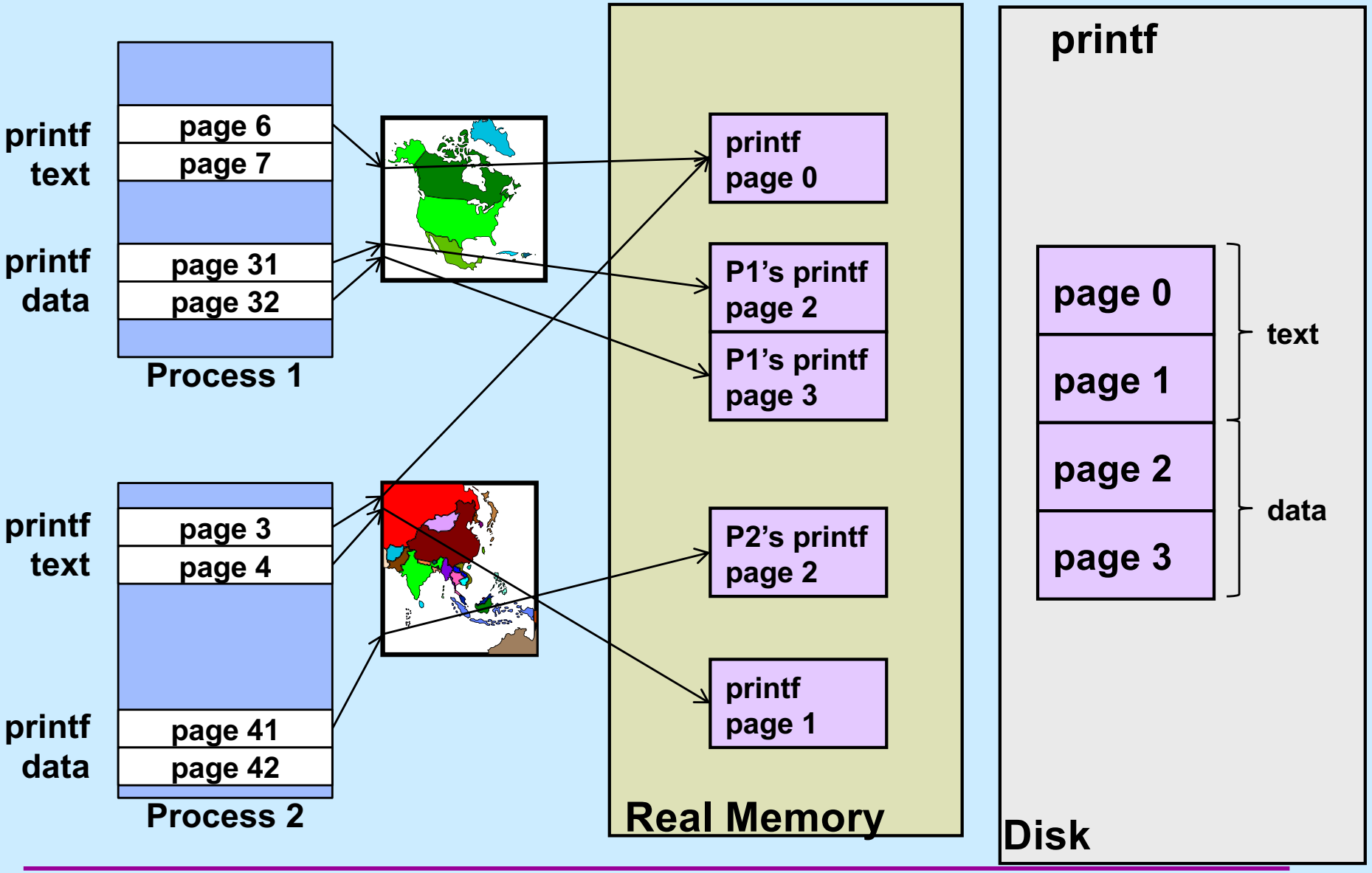
Mapping Shared Objects



Mapping printf into the Address Space

- Printf's text
  - read-only
  - can it be shared?
    - » yes: use MAP_SHARED
- Printf's data
  - read-write
  - not shared with other processes
  - initial values come from file
  - can mmap be used?
    - » MAP_SHARED wouldn't work
      - changes made to data by one process would be seen by others
    - » MAP_PRIVATE does work!
      - mapped region is initialized from file
      - changes are private

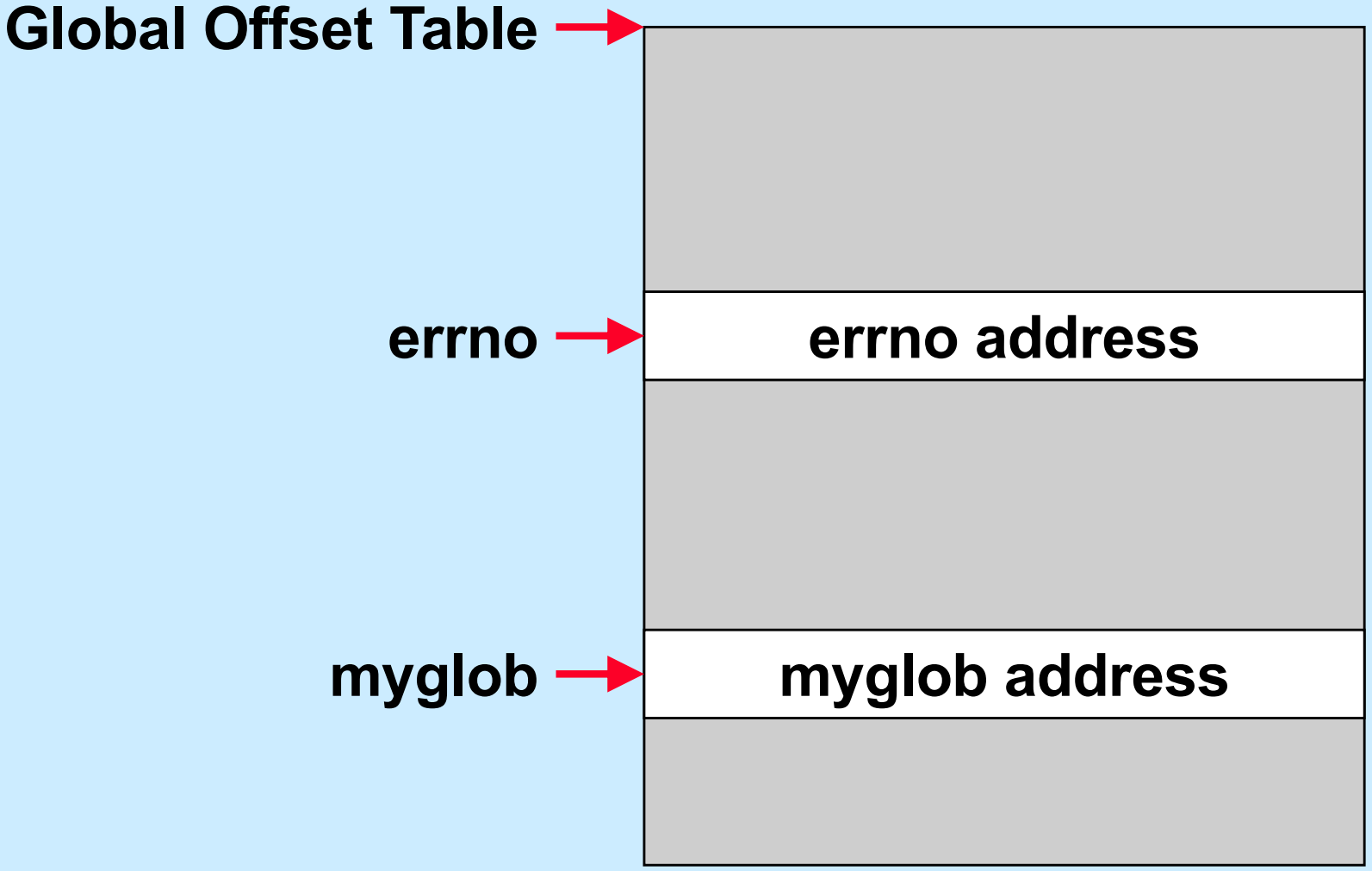
Mapping printf



Position-Independent Code

- Produced by gcc when given the –fPIC flag
- Processor-dependent; x86-64:
  - each dynamic executable and shared object has:
    - » procedure-linkage table
      - shared, read-only executable code
      - essentially stubs for calling functions
    - » global-offset table
      - private, read-write data
      - relocated dynamically for each process
    - » relocation table
      - shared, read-only data
      - contains relocation info and symbol table

Global-Offset Table: Data References



Functions in Shared Objects

- Lots of them
- Many are never used
- Fix up linkages on demand

An Example

```
int main( ) {
   puts("Hello world\n");
   …
   return 0;
}
```

```
0000000000006b0 <main>:
 6b0: 55                          push   %rbp
 6b1: 48 89 e5                    mov    %rsp,%rbp
 6b4: 48 8d 3d 99 00 00 00        lea    0x99(%rip),%rdi
 6bb: e8 a0 fe ff ff              callq  560 <puts@plt>
 …
```


Before Calling puts

```
.PLT0:
  pushq GOT+8(%rip)
  jmp   *GOT+16(%rip)
  nop; nop
  nop; nop
.puts:
  jmp   *puts@GOT(%rip)
.putsnext:
  pushq $putsRelOffset
  jmp   .PLT0
.PLT2:
  jmp   *name2@GOT(%rip)
.PLT2next:
  pushq $name2RelOffset
  jmp   .PLT0
```

Procedure-Linkage Table

```
GOT:
   .quad _DYNAMIC
   .quad identification
   .quad ld-linux.so


puts:
   .quad .putsnext
name2:
   .quad .PLT2next
```

Relocation info:

```
GOT_offset(puts), symx(puts)
```

```
GOT_offset(name2), symx(name2)
```

Relocation Table

After Calling puts

```
.PLT0:
  pushq GOT+8(%rip)
  jmp   *GOT+16(%rip)
  nop; nop
  nop; nop
.puts:
  jmp   *puts@GOT(%rip)
.putsnext:
  pushq $putsRelOffset
  jmp   .PLT0
.PLT2:
  jmp   *name2@GOT(%rip)
.PLT2next:
  pushq $name2RelOffset
  jmp   .PLT0
```

Procedure-Linkage Table

```
GOT:
   .quad _DYNAMIC
   .quad identification
   .quad ld-linux.so


puts:
   .quad puts
name2:
   .quad .PLT2next
```

Relocation info:

```
GOT_offset(puts), symx(puts)
```

```
GOT_offset(name2), symx(name2)
```

Relocation Table

Quiz 2

On the second and subsequent calls to *puts*

a) control goes directly to *puts*
b) control goes to an instruction that jumps to *puts*
c) control still goes to ld-linux.so, but it now transfers control directly to *puts*

You'll Soon Finish CS 33 …

- You might
  - celebrate

  - take another systems course
    - » 320
    - » 1380
    - » 1660
    - » 1670
    - » 1680

  - become a 300 TA

Systems Courses Next Semester

- CS 320 (Intro to Software Engineering)
  - you've mastered low-level systems programming
  - now do things at a higher level
  - learn software-engineering techniques using Java, XML, etc.
- CS 1380 (Distributed Systems)
  - you now know how things work on one computer
  - what if you've got lots of computers?
  - some may have crashed, others may have been taken over by your worst (and smartest) enemy
- CS 1660/1620/2660 (Computer Systems Security)
  - liked buffer?
  - you'll really like 1660
- CS 1670/1690/2670 (Operating Systems)
  - still mystified about what the OS does?
  - write your own!

The End

Well, not quite …

Database is due on 12/13

Happy Coding and Happy Holidays!

