CSE 351: Week 8

Tom Bergan, TA

Today

- What happens when a program starts running?
- Address spaces
- Virtual memory

Let’s start a program

$ ./bufbomb -u tbergan

Goal: execute main() in ./bufbomb

```
int main(int argc, char *argv[]) {
    ...
}
```

Where

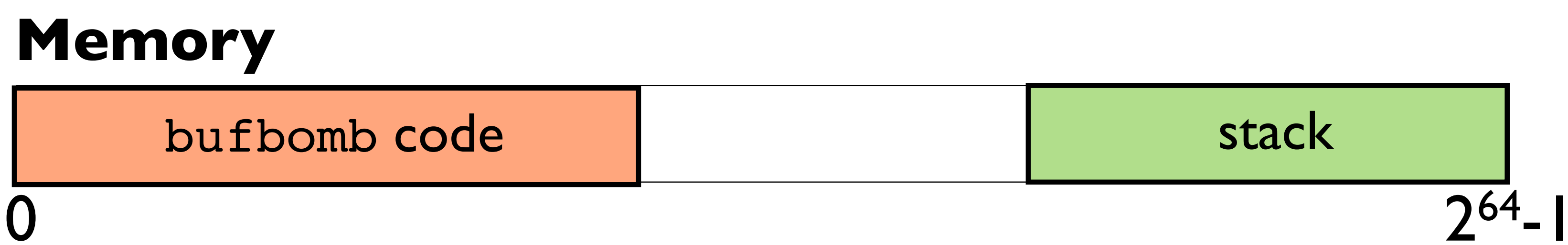
```
argc = 3
argv[0] = “./bufbomb”
argv[1] = “-u”
argv[2] = “tbergan”
```

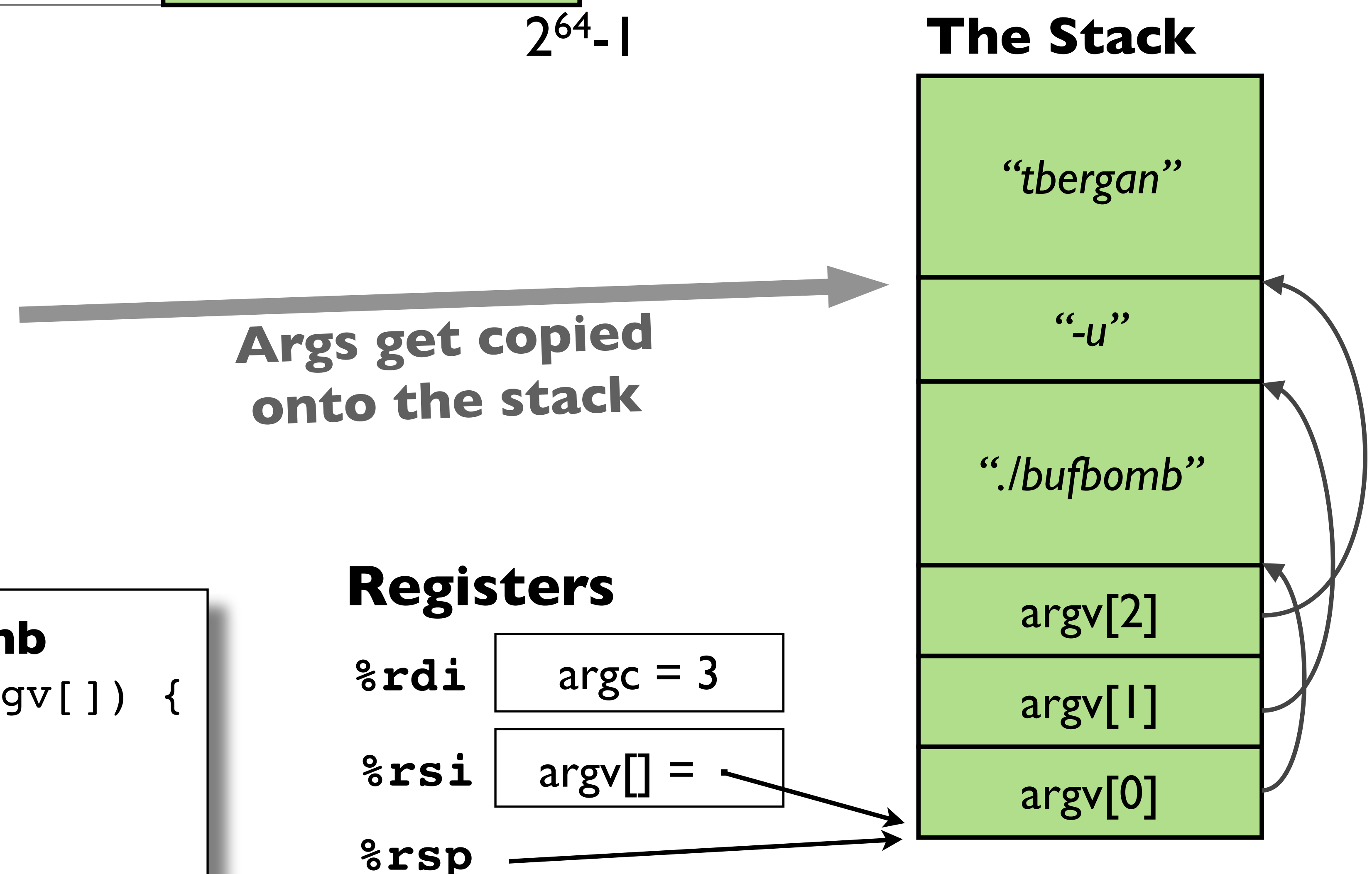
The shell executes this code:

execl(“./bufbomb”, “-u”, “tbergan”, NULL);

How does exec() work?

What happens on exec()?





Steps to exec:

1. Load program executable
2. Copy the args into memory
3. Setup the registers
4. Jump to main()

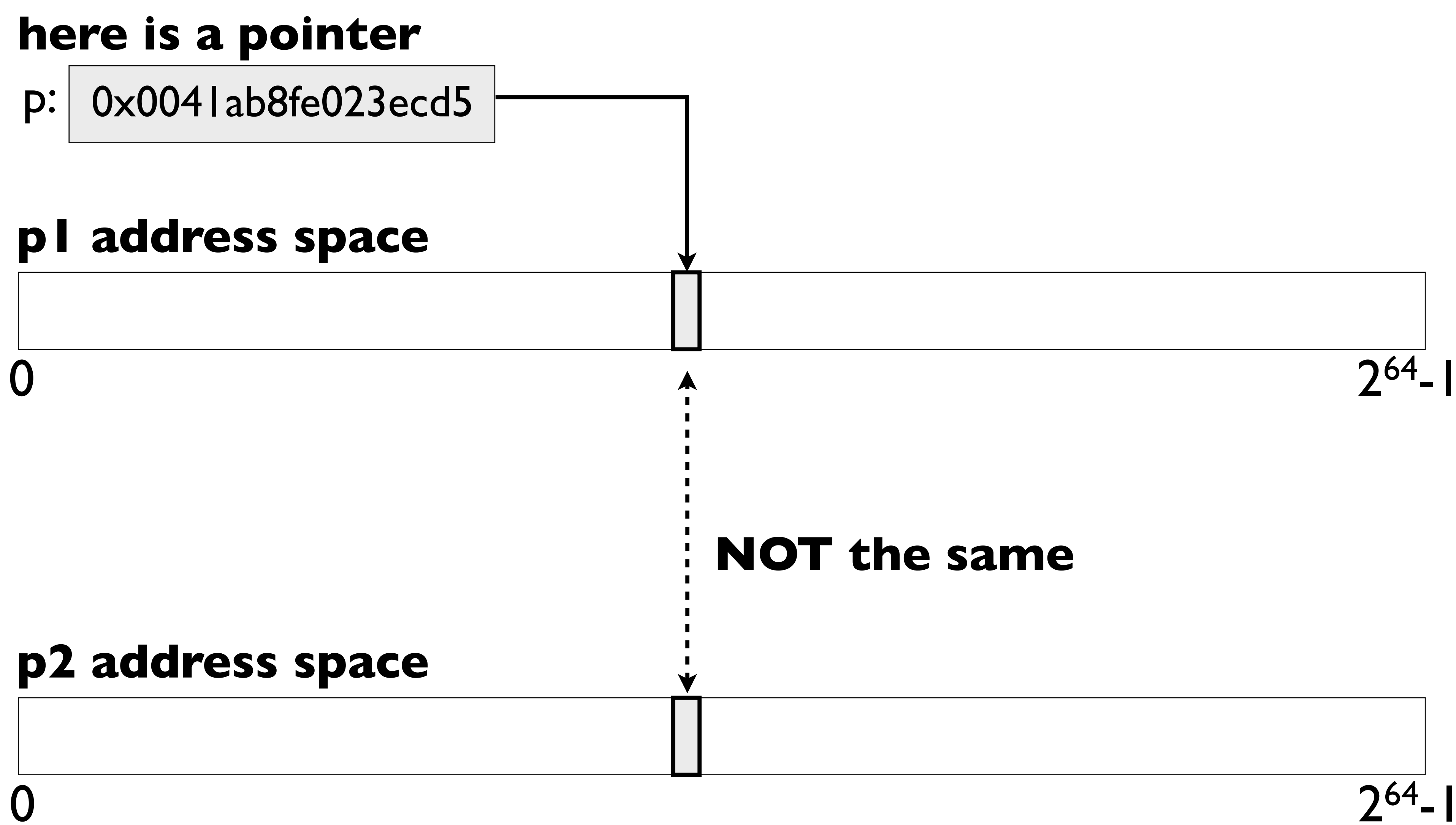
Goal: execute main() in ./bufbomb

```
int main(int argc, char *argv[]) {
    ...
}
```

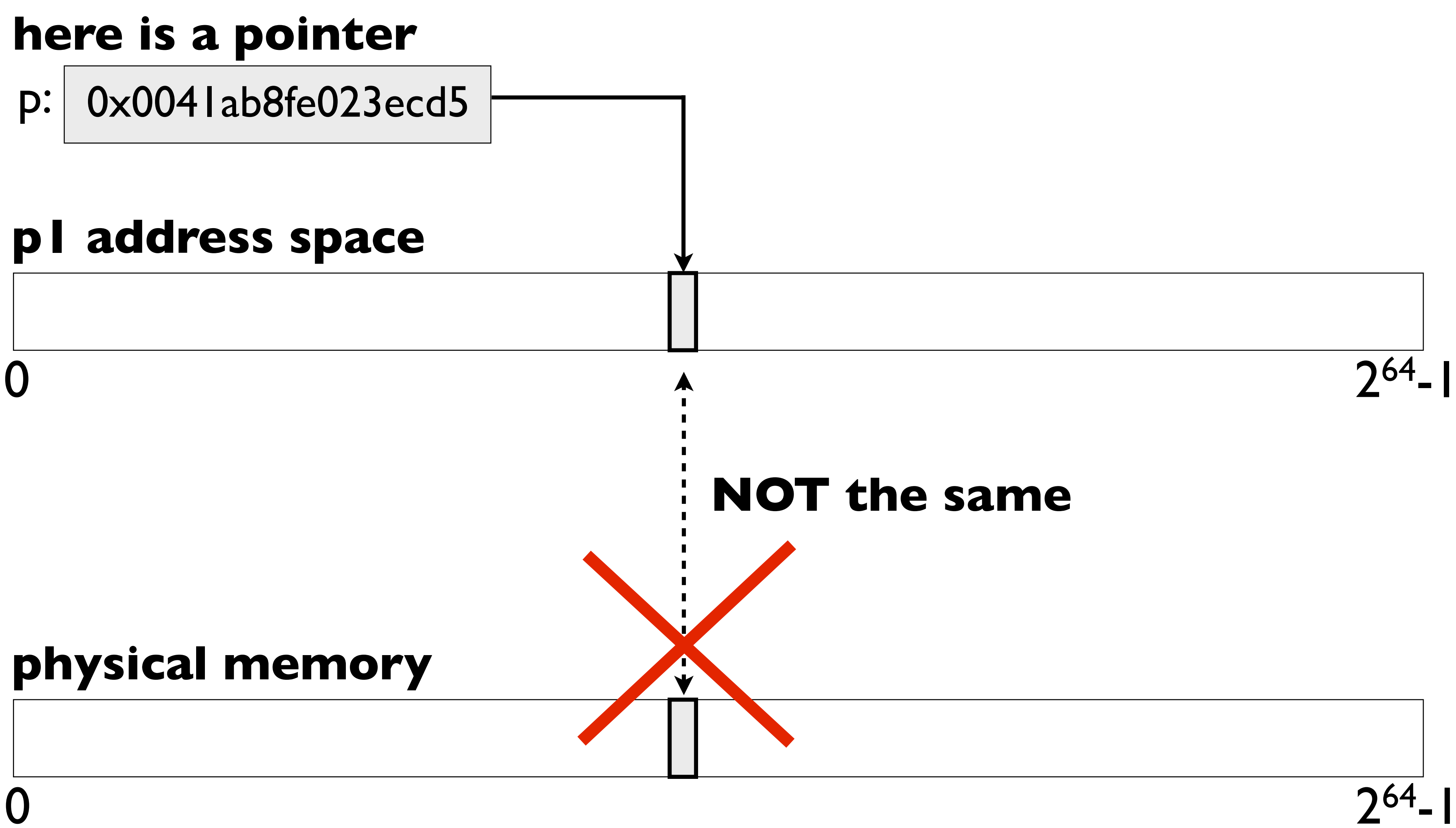
Where

```
argc = 3
argv[0] = "./bufbomb"
argv[1] = "-u"
argv[2] = "tbergan"
```

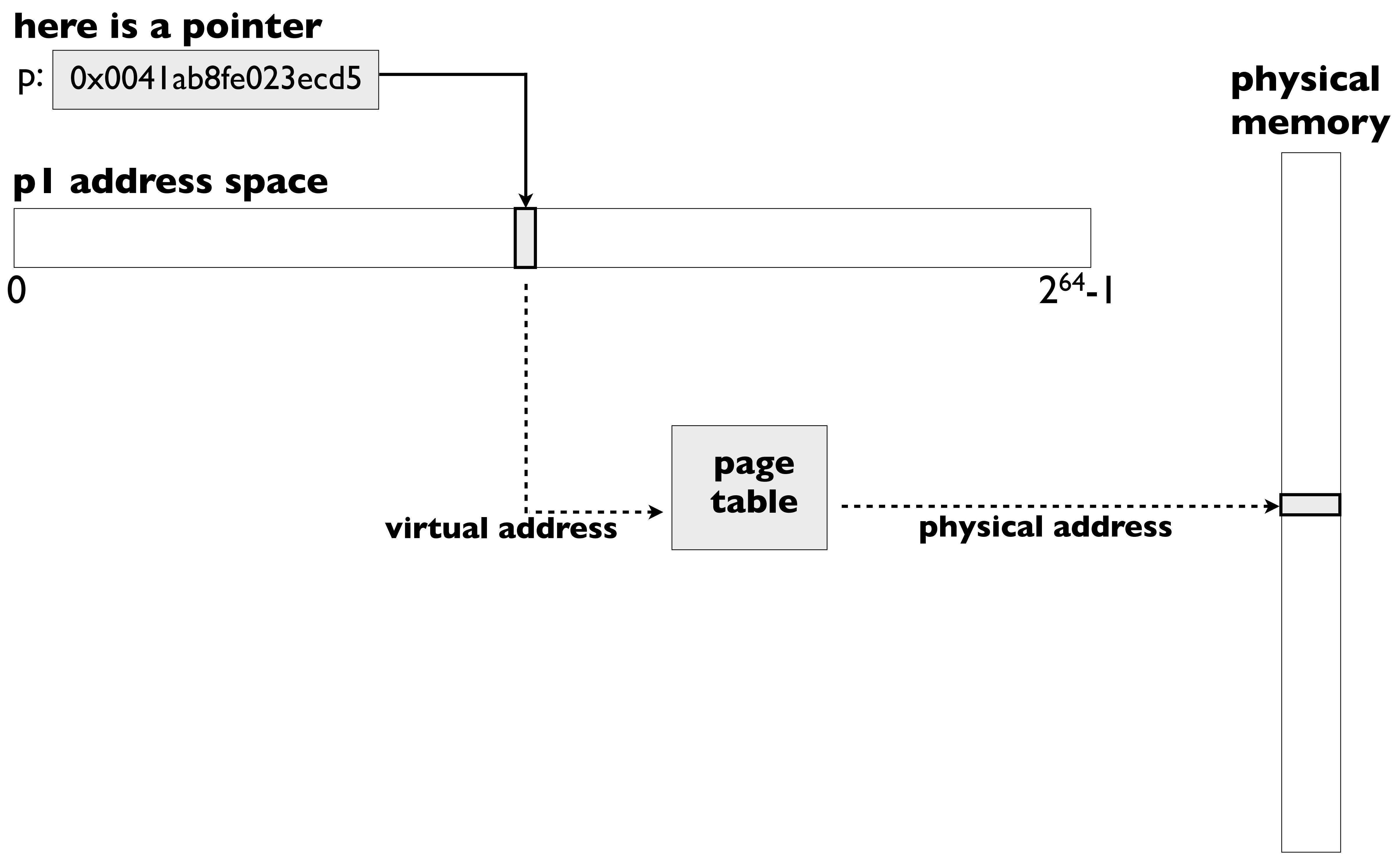
Each process has its own address space



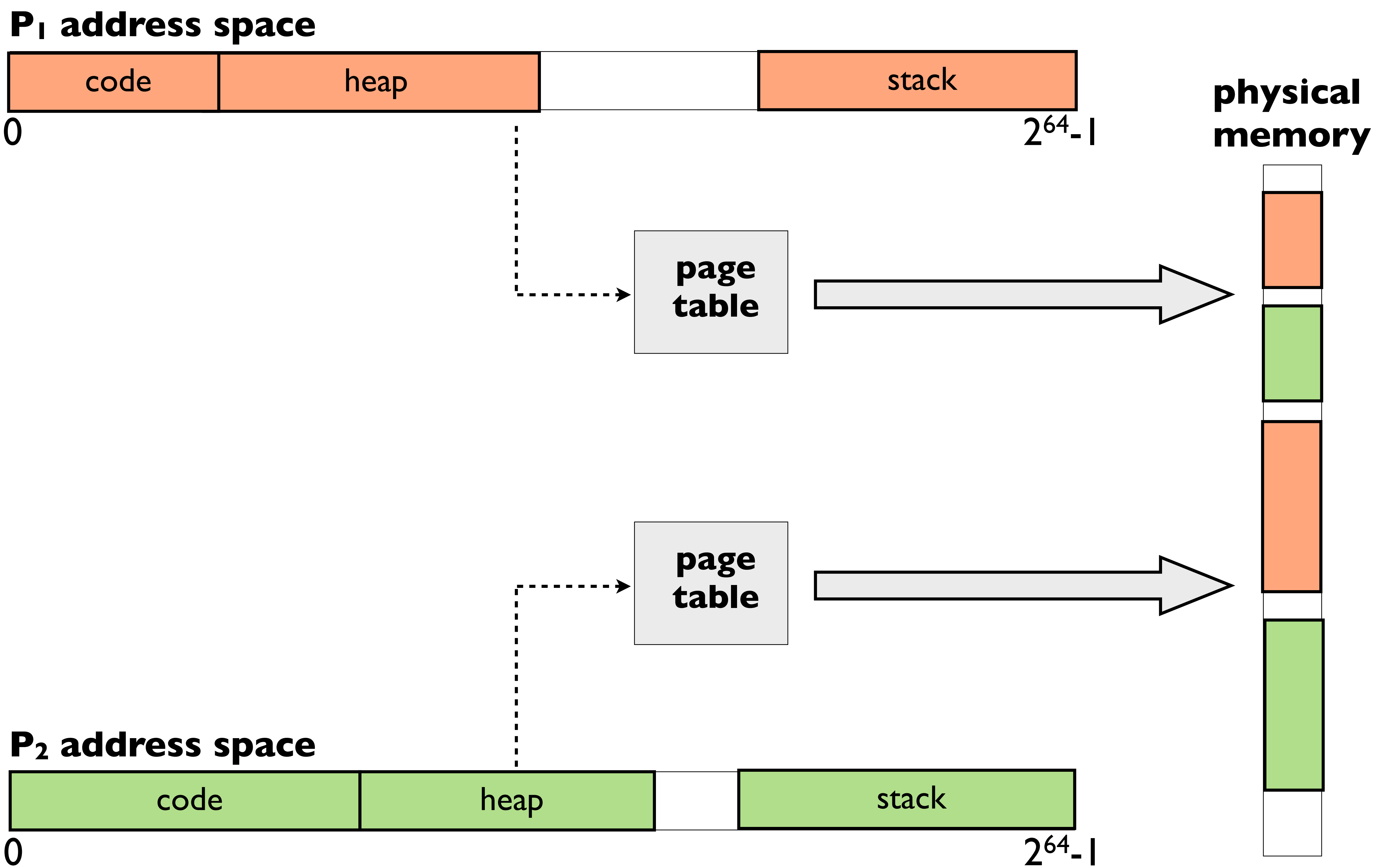
Address spaces are virtual



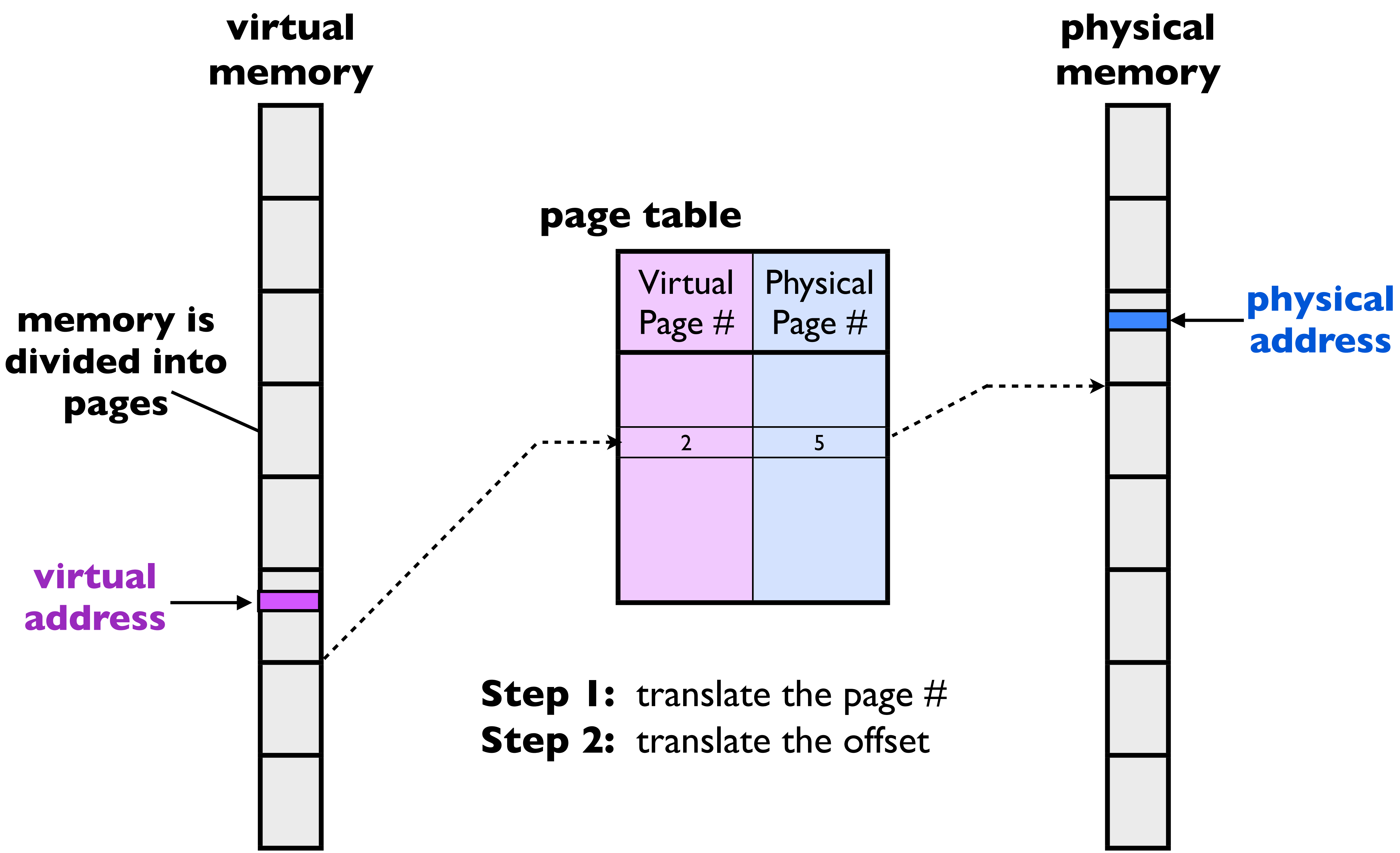
Virtual Address Spaces



Virtual Address Spaces

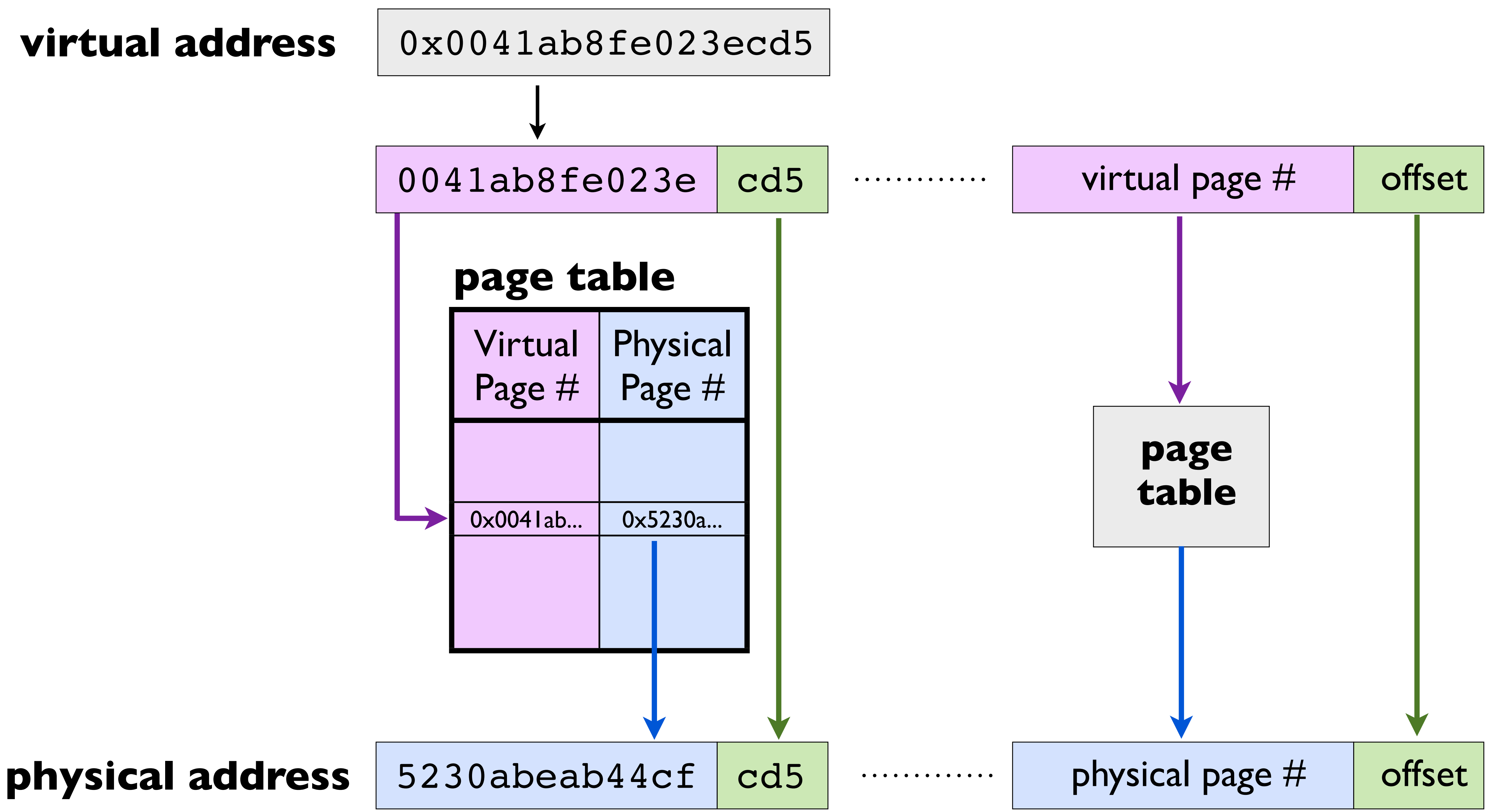


Virtual address translation

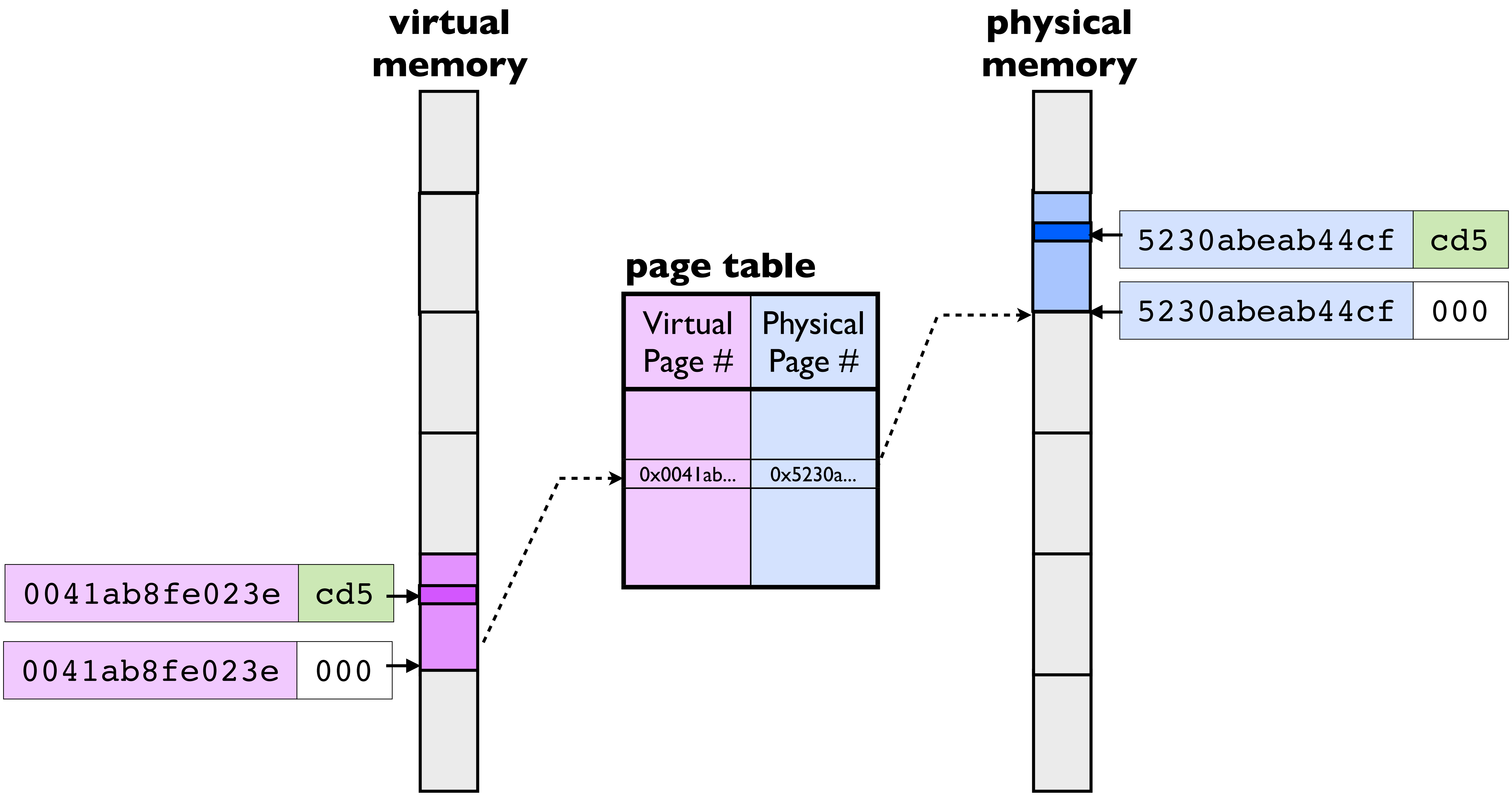


Step 1: translate the page #
Step 2: translate the offset

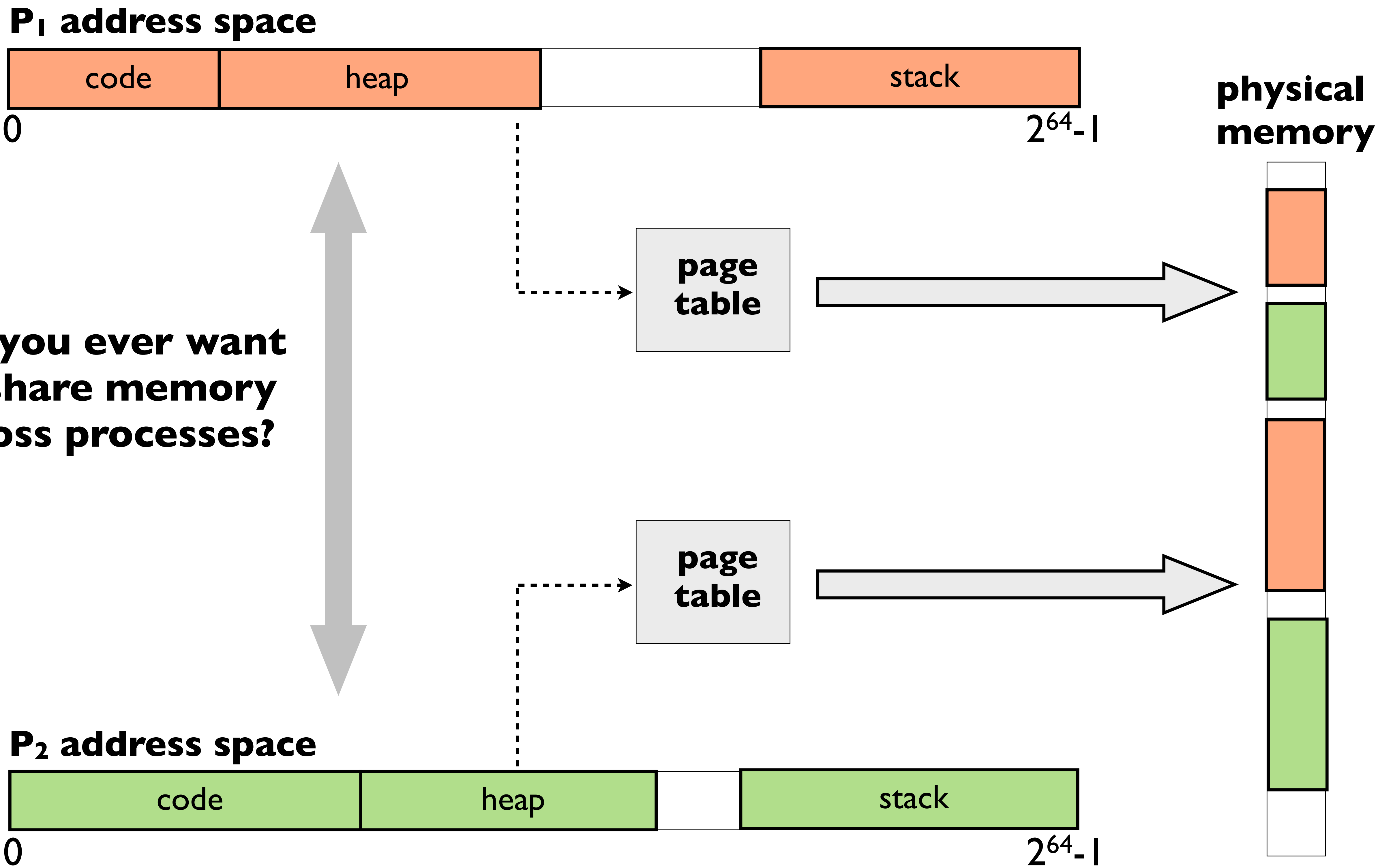
Virtual address translation



Virtual address translation

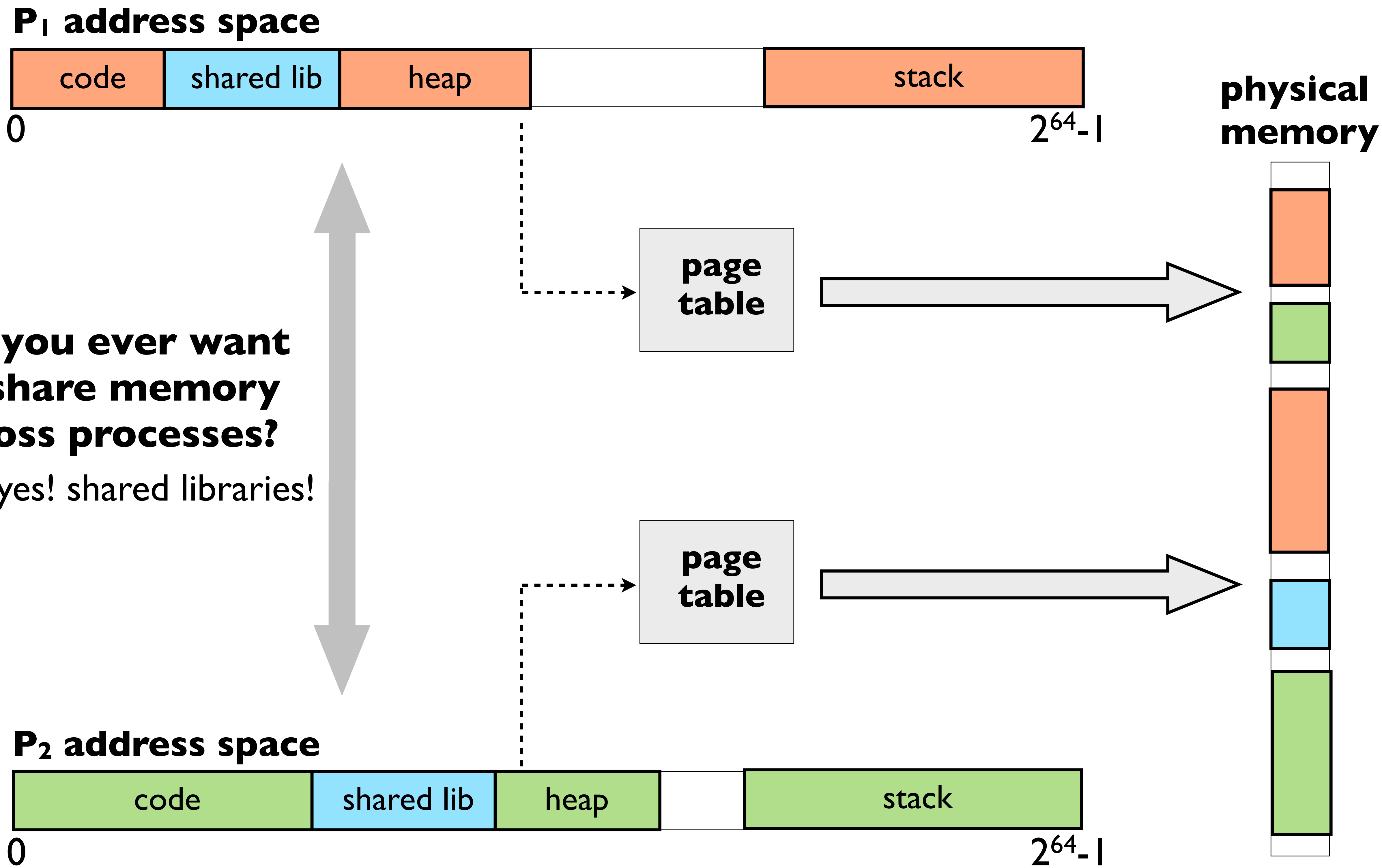


Virtual Address Spaces



Do you ever want to share memory across processes?

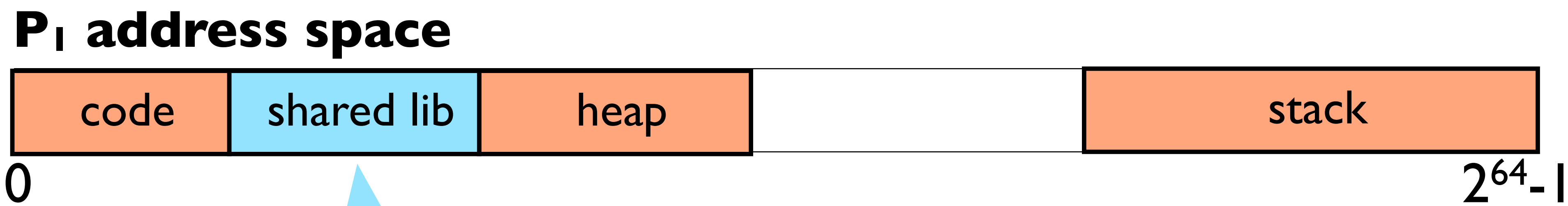
Virtual Address Spaces

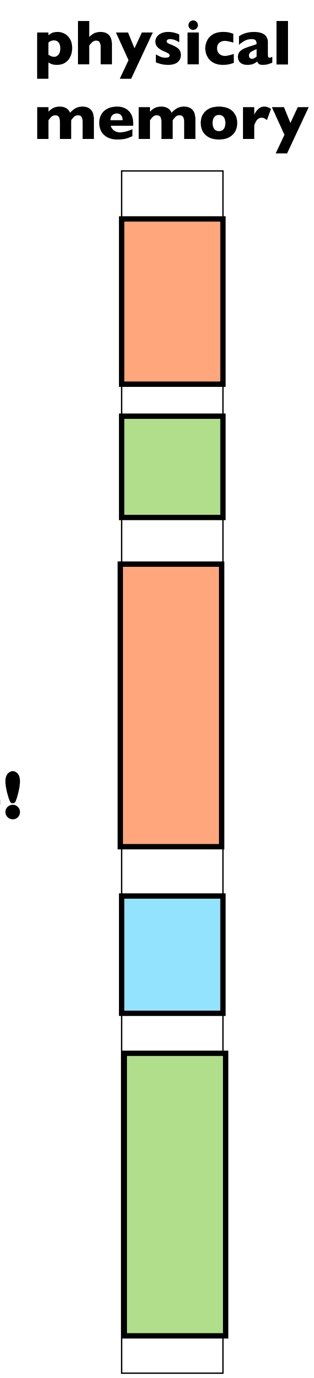


Do you ever want to share memory across processes?

- yes! shared libraries!

Shared Libraries



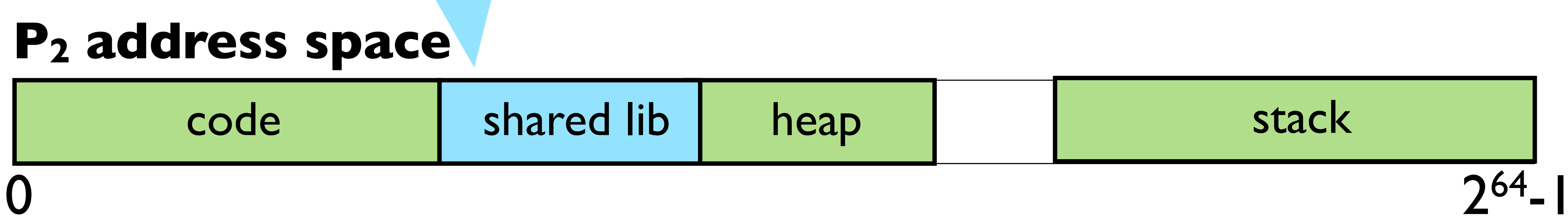


A shared library:

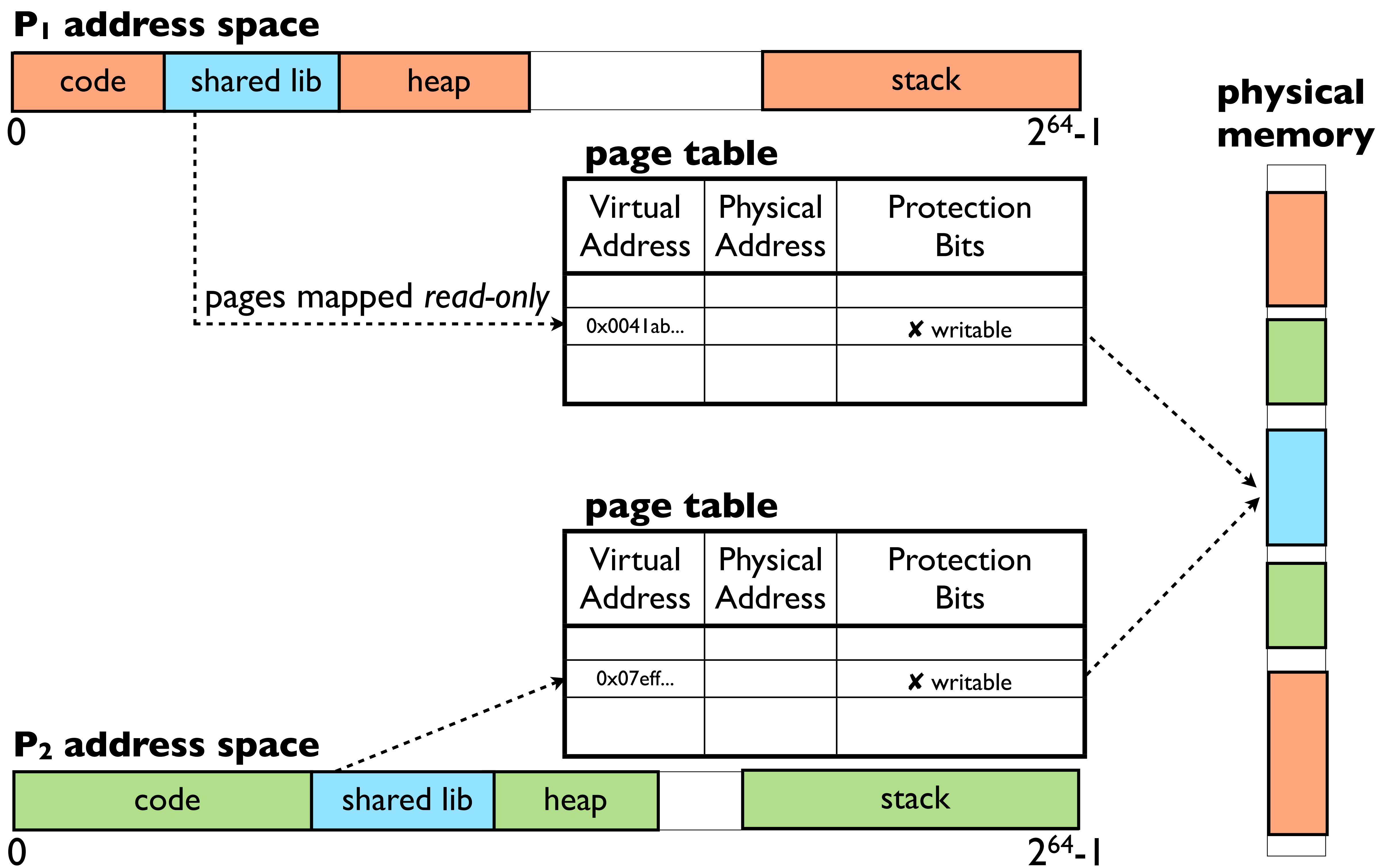
- think printf(): **.so* on linux, **.dll* on windows
- share code pages in multiple address spaces (saves space!)

Problem: can't let P_2 overwrite to P_1's code!

- solution: map pages *read-only*



Shared Libraries



Page table protection bits

(partial list)

- **writable** bit
  - is the page writable?
  - when unset, the page is *read-only*

  Why would you want this?
  - protect code pages (don't accidentally overwrite)
  - read-only data (e.g. constant strings literals: "xyz")

- **executable** bit
  - is the page executable?
  - when unset, code on the page *cannot* be executed

  Why would you want this?
  - protect non-code pages (e.g. prevents buffer overflow exploits)
  - read-only data (e.g. constant strings literals: "xyz")

Shared Libraries

Shared libraries are loaded at *runtime*

New steps to start a program:

1. Load program executable

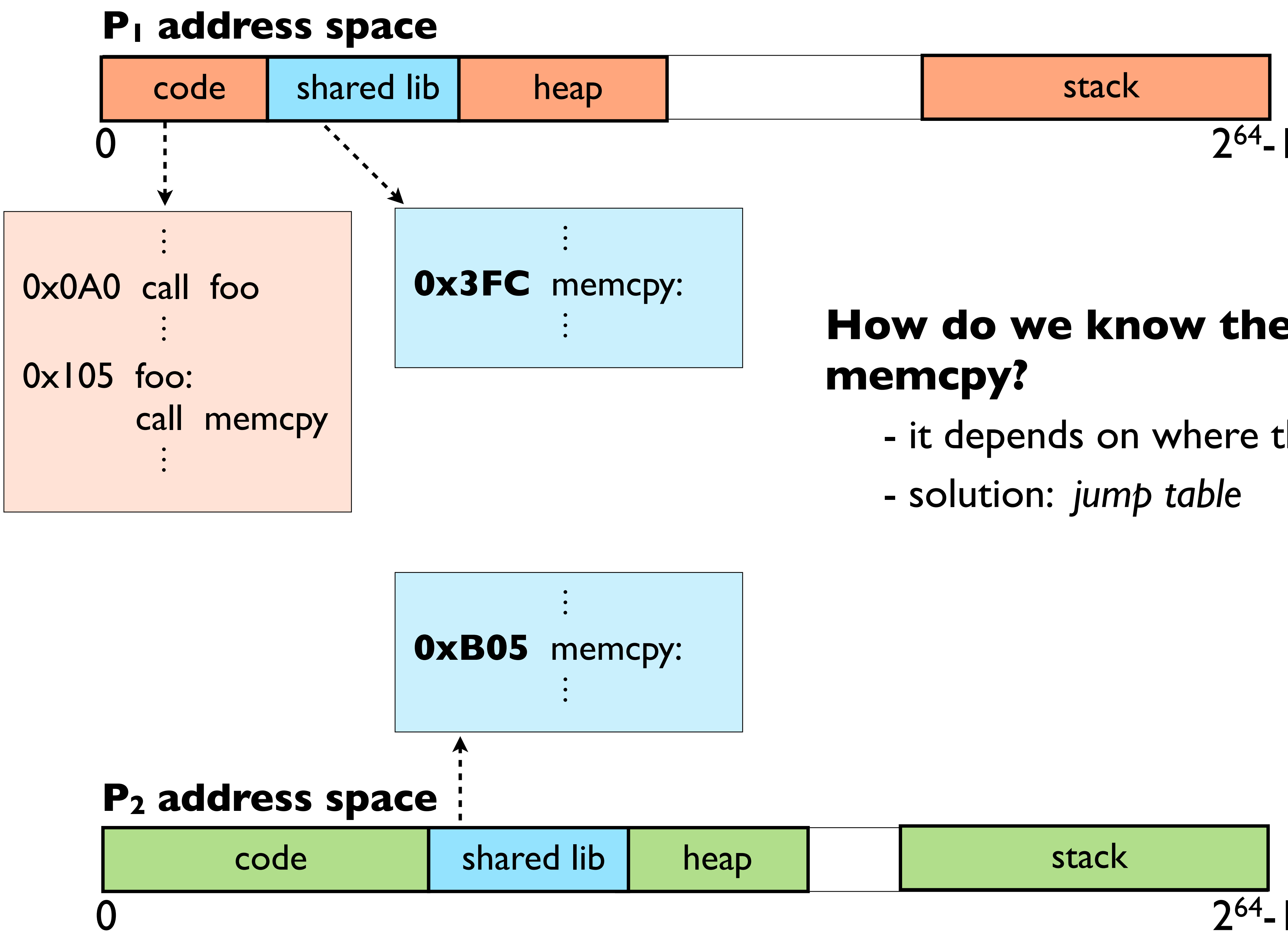
1a. Load shared libraries

2. Copy the args into memory

3. Setup the registers

4. Jump to main()

Shared Libraries

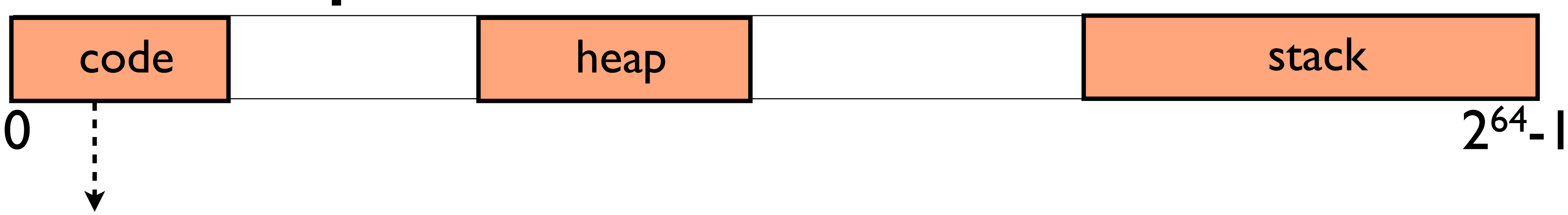


How do we know the address of memcpy?

- it depends on where the lib was loaded
- solution: *jump table*

Shared Libraries

P_1 address space



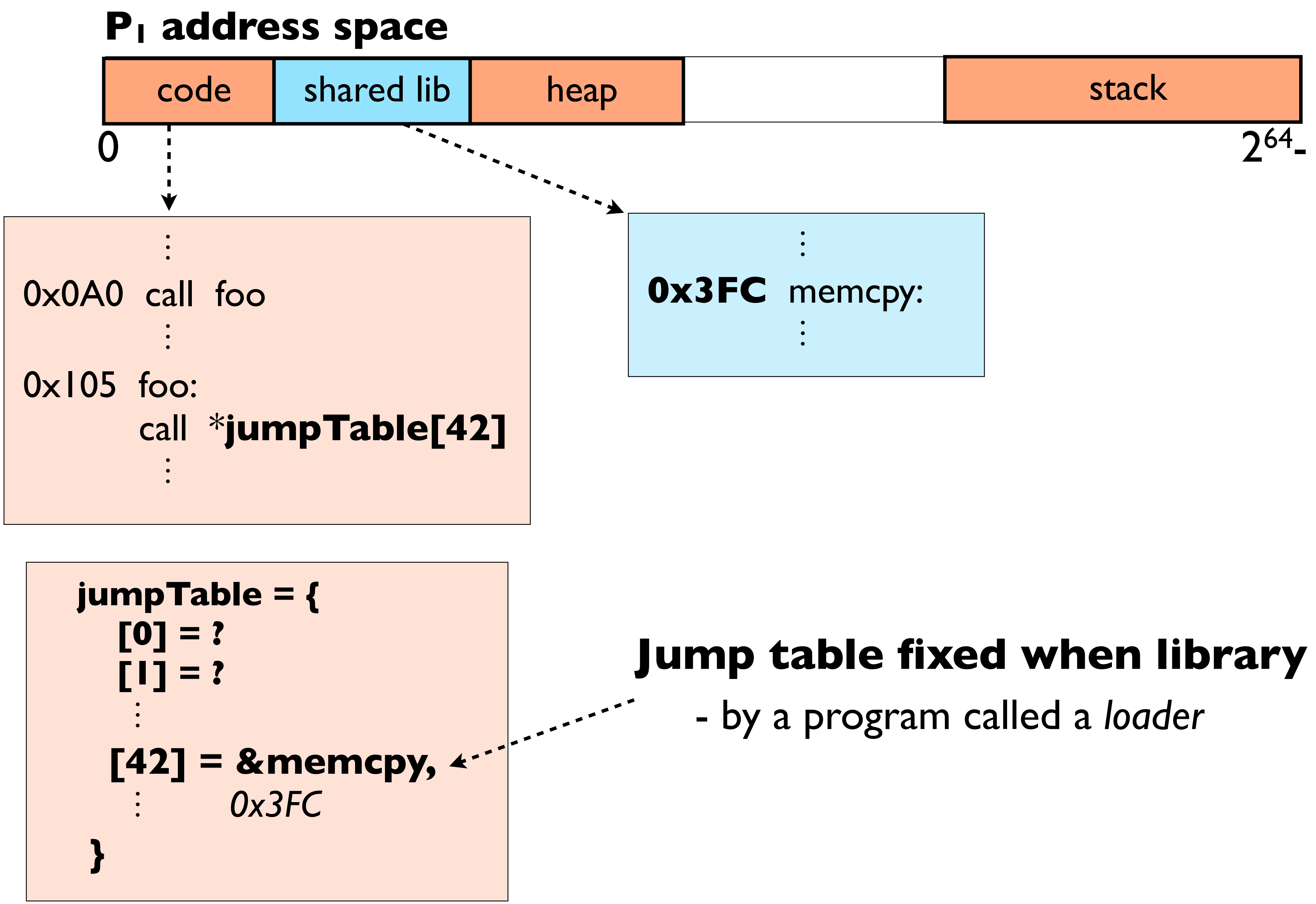

```
        ⋮
0x0A0  call  foo
        ⋮
0x105  foo:
       call  *jumpTable[42]
        ⋮
```

Library call indirects through *jump table*

```
jumpTable = {
   [0] = ?
   [1] = ?
    ⋮
   [42] = ?
    ⋮
}
```

Jump table initially empty

Shared Libraries



Jump table fixed when library is loaded

- by a program called a *loader*