WELLESLEY

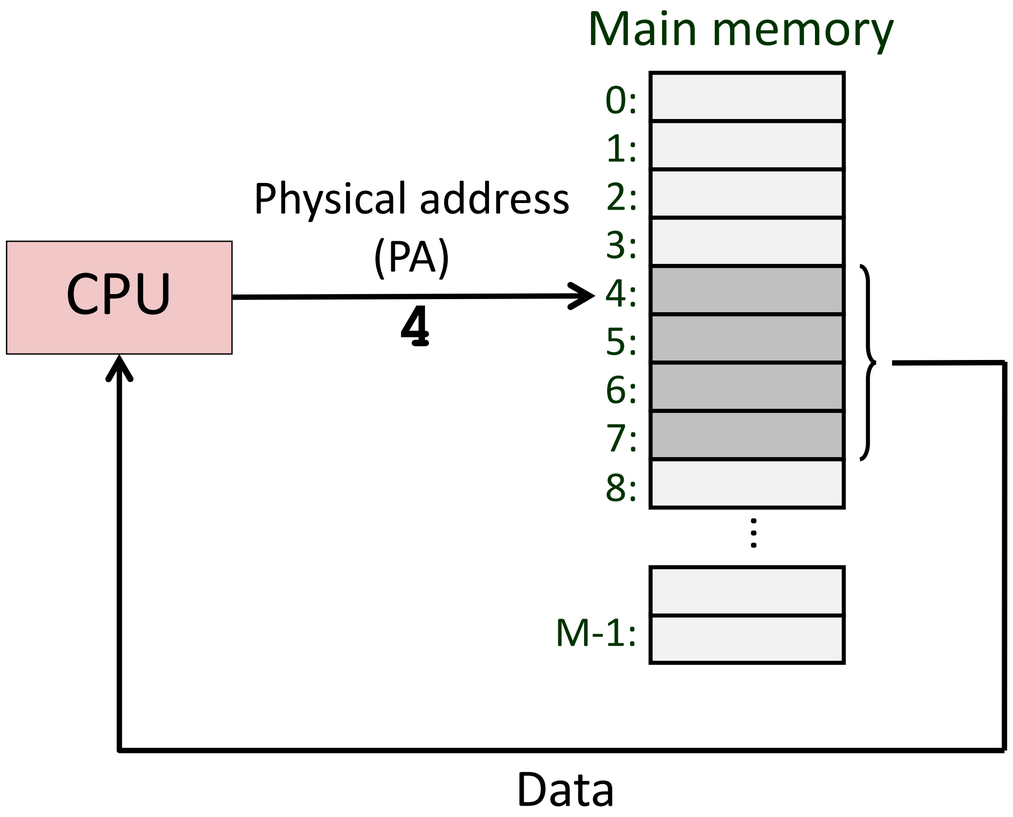
Virtual Memory

Process Abstraction, Part 2: Private Address Space

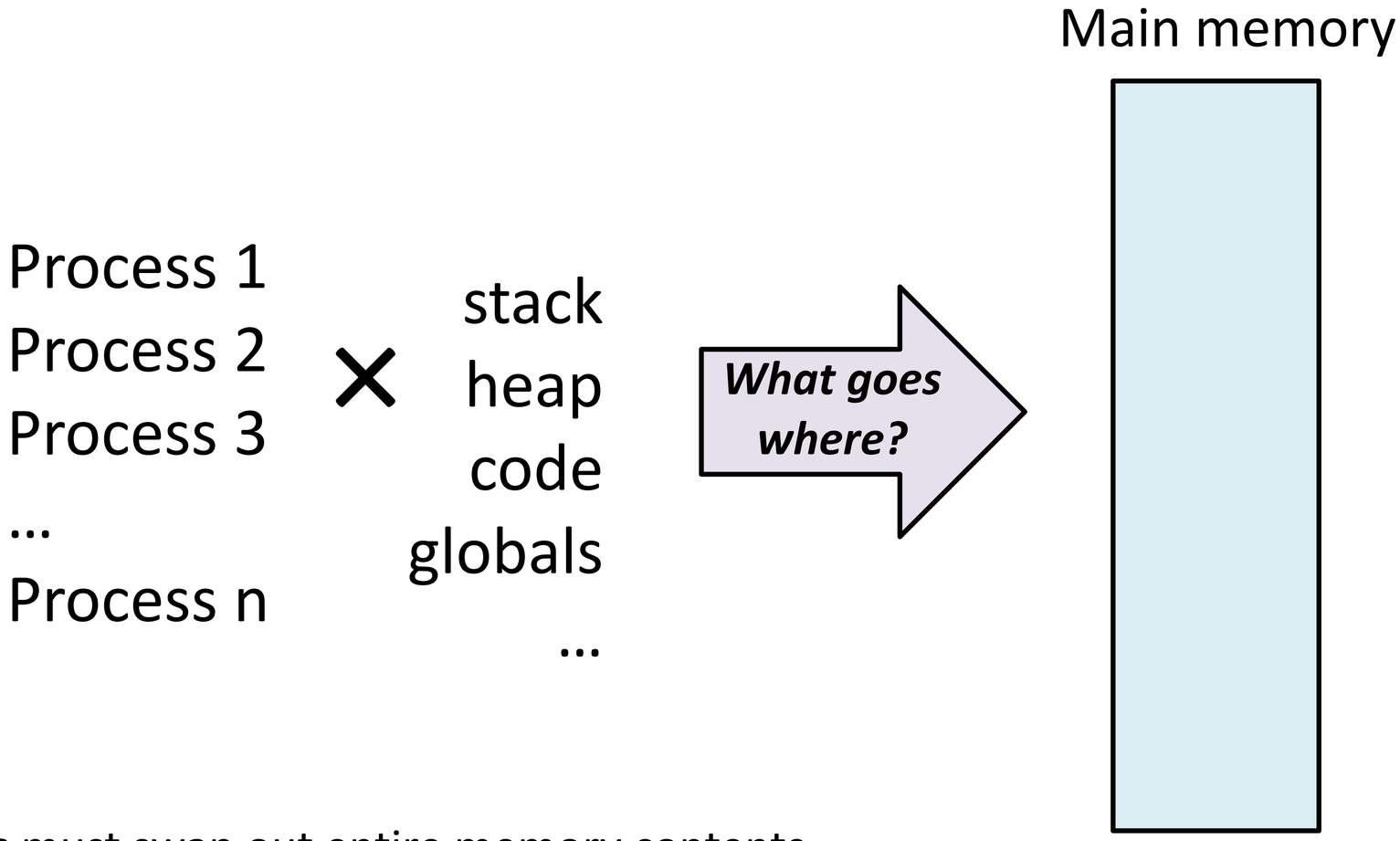
Motivation: why not direct physical memory access?
Address translation with pages
Extra benefits: sharing and protection

Big picture: memory as a contiguous array of bytes is a lie! Why?

Problems with physical addressing



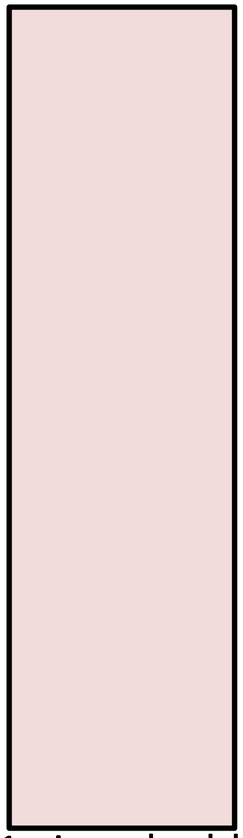
Problem 1: memory management



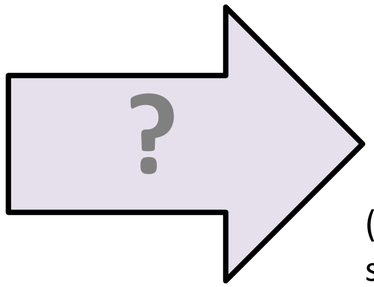
Also:
Context switches must swap out entire memory contents.
Isn't that **expensive**?

Problem 2: capacity

64-bit addresses can address
several exabytes
(18,446,744,073,709,551,616 bytes)



1 virtual address space per process,
with many processes…



Physical main memory offers
~a few dozen gigabytes
(e.g. 8,589,934,592 bytes)

(To scale with 64-bit address space, you can't see it!)

What does this code print? Why/how?

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  int s;
  int x = 1;
  int *p;
  int child_pid = fork();
  p = &x;
  if (child_pid == 0) {
    *p = 2;
  } else {
    *p = 3;
  }
  printf("Address Ox%lx holds %d\n", (long)p, *p);
}
```

Which is a possible output of this program?

0

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  int s;
  int x = 1;
  int *p;
  int child_pid = fork();
  p = &x;
  if (child_pid == 0) {
    *p = 2;
  } else {
    *p = 3;
  }
  printf("Address 0x%lx holds %d\n", (long)p, *p);
}
```

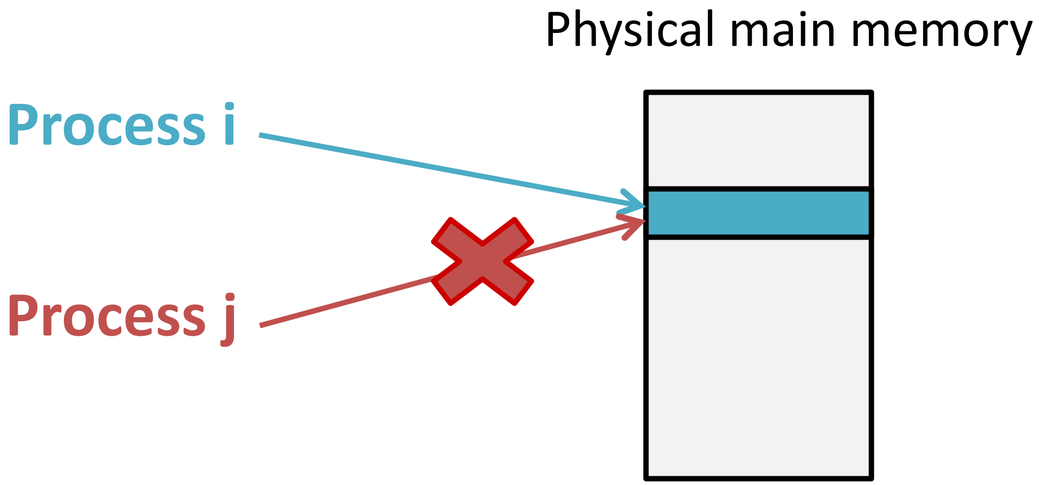
- Addr 0x10 holds 2; Addr 0x10 holds 2
- Addr 0x10 holds 3; Addr 0x10 holds 3
- Addr 0x10 holds 3; Addr 0x10 holds 2
- Addr 0x10 holds 3; Addr 0x20 holds 3
- Addr 0x10 holds 3; Addr 0x20 holds 2
- None of the above

What does this code print? Why/how?

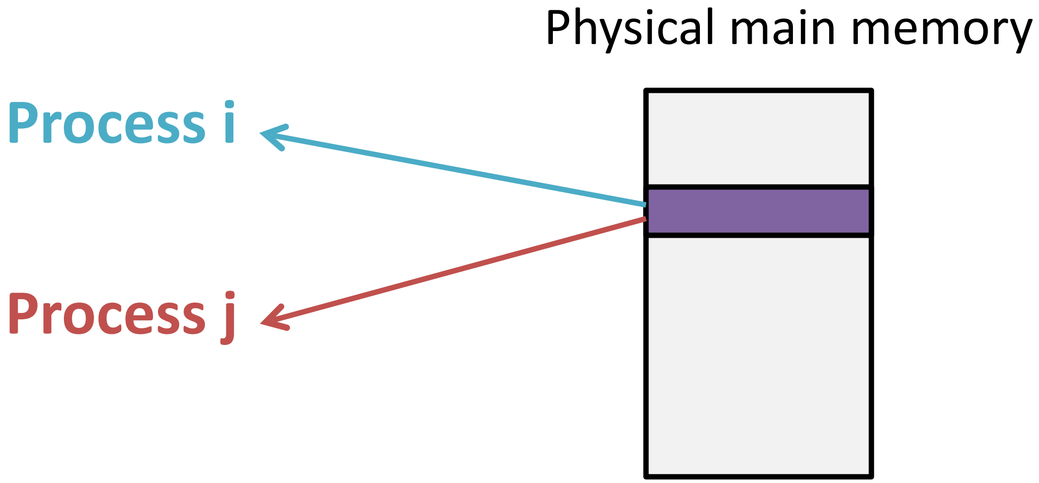
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  int s;
  int x = 1;
  int *p;
  int child_pid = fork();
  p = &x;
  if (child_pid == 0) {
    *p = 2;
  } else {
    *p = 3;
  }
  printf("Address Ox%lx holds %d\n", (long)p, *p);
}
```

```
$ ./fork.o
Address 0x16B7E2C04 holds 3
Address 0x16B7E2C04 holds 2
```

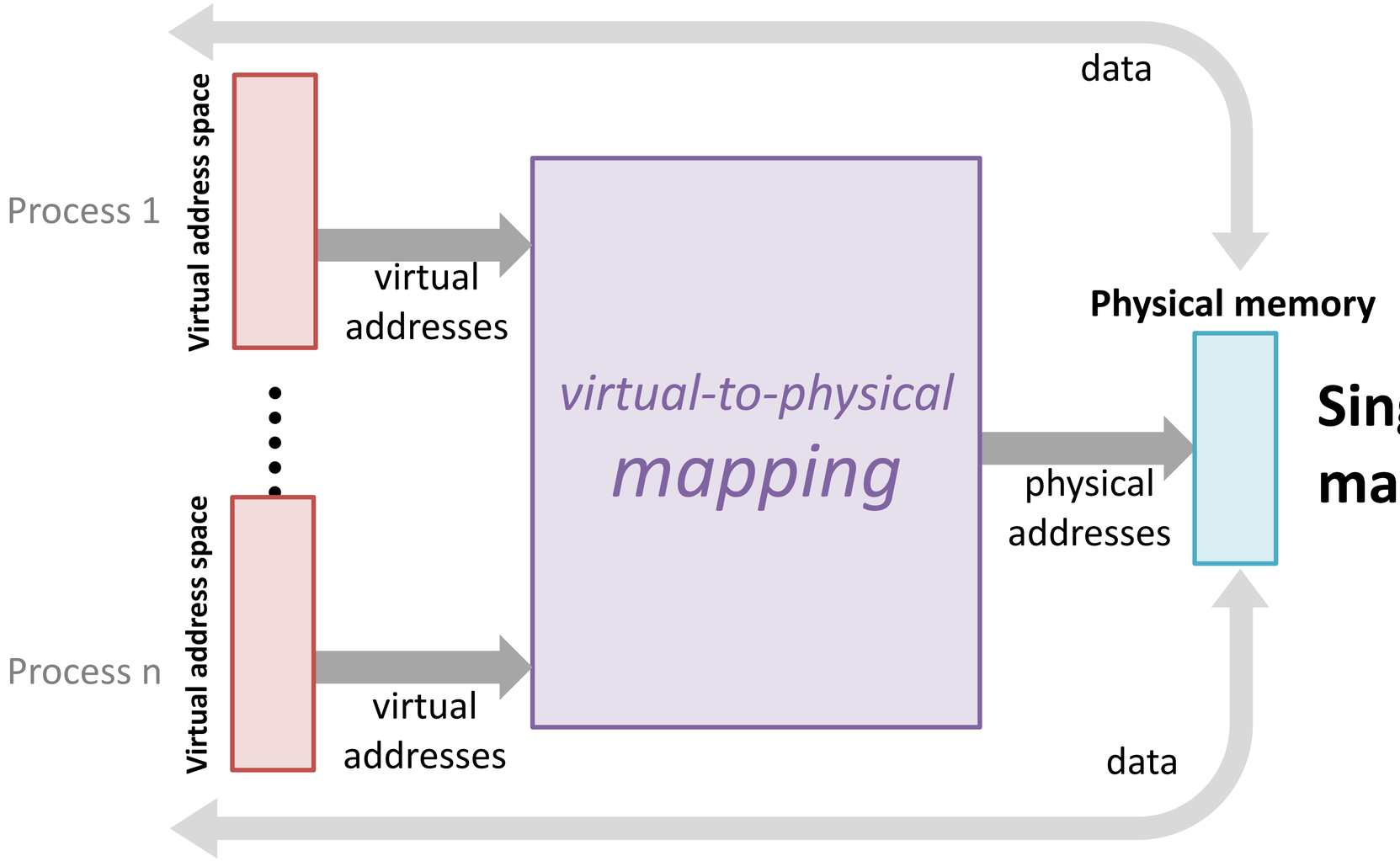
Problem 3: protection



Problem 4: sharing



Solution: Virtual Memory (address *indirection*)



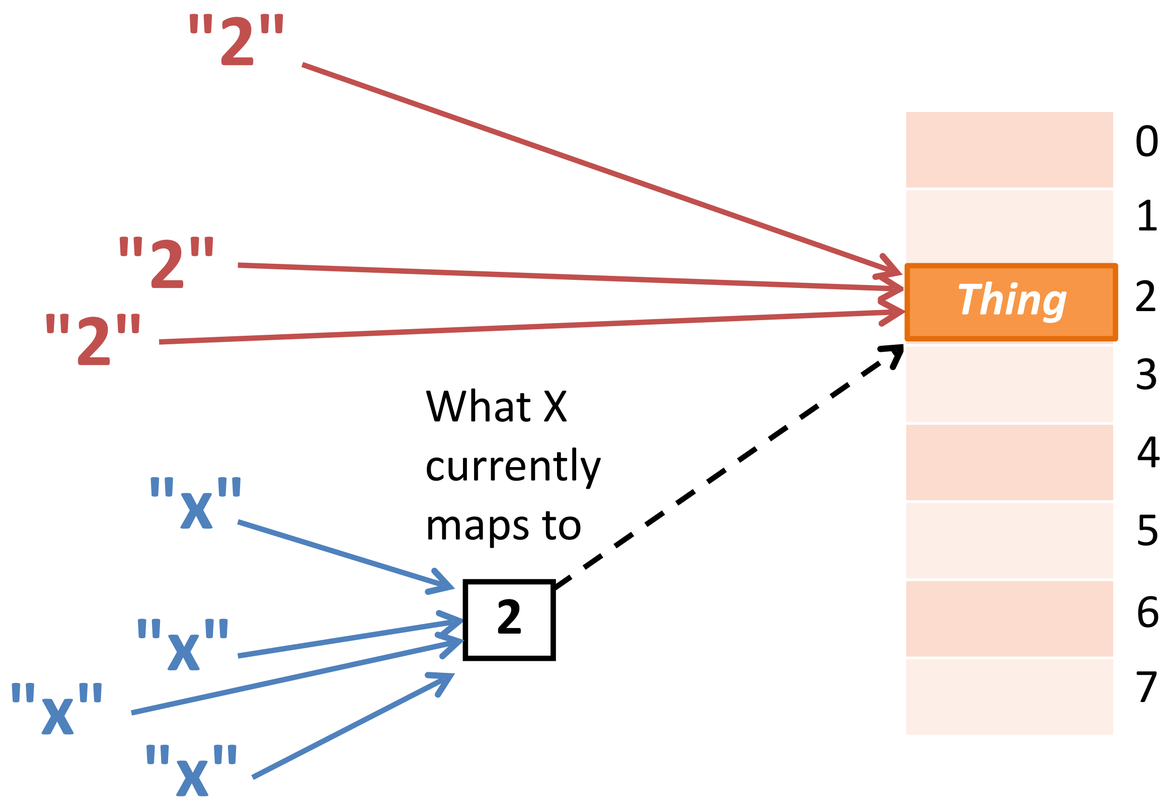
Single physical address space managed by OS/hardware.

Indirection

(it's everywhere!)

Direct naming

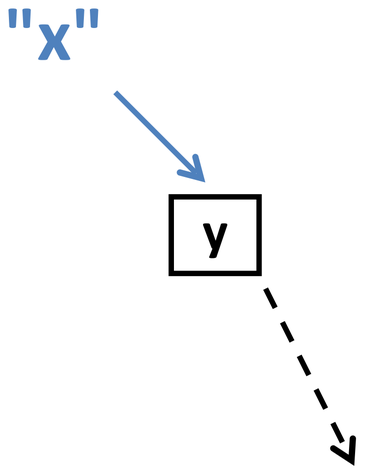
Indirect naming



What if we move ***Thing***?

Tangent: **indirection everywhere**

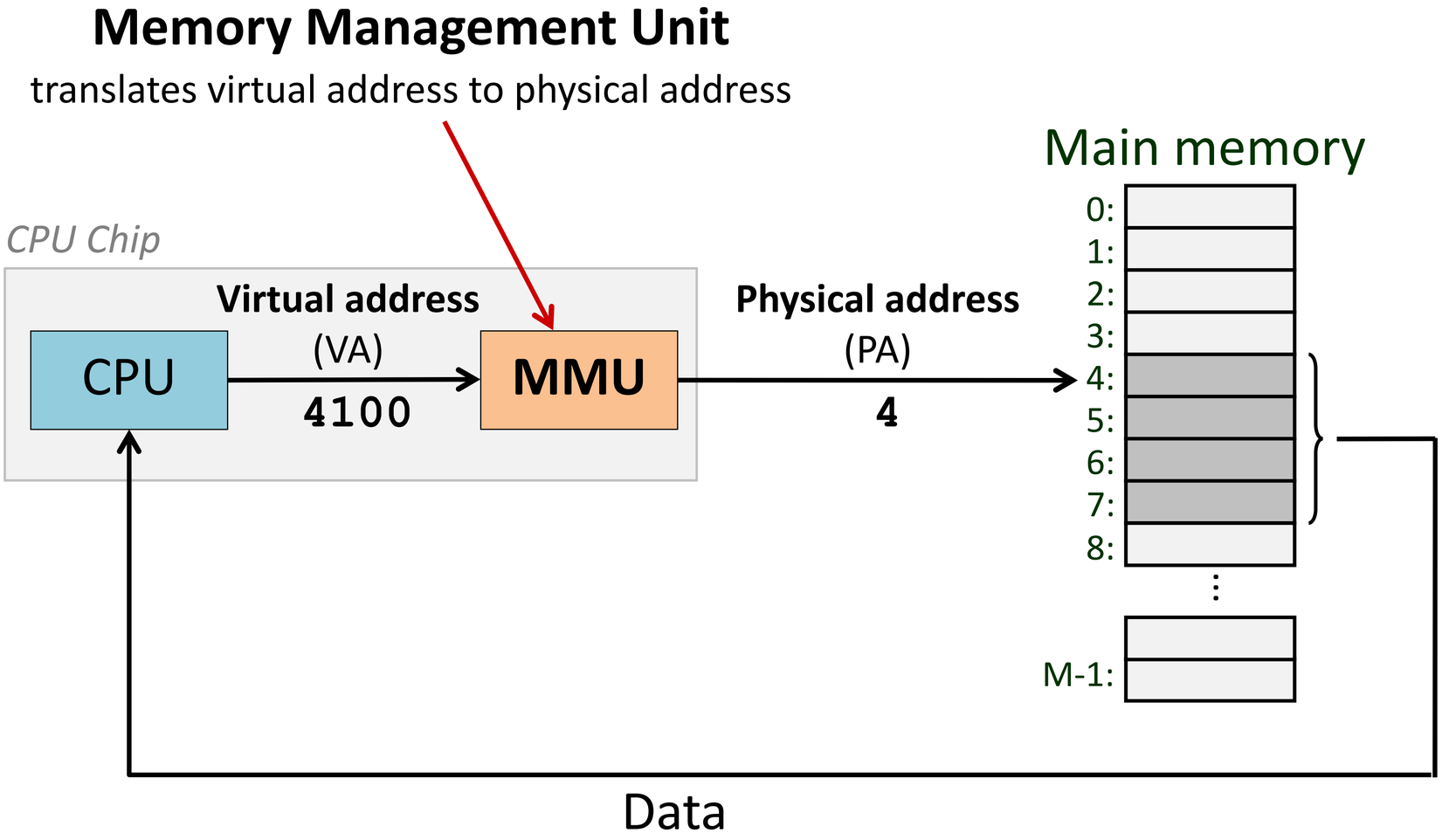
- Pointers
- Constants
- Procedural abstraction
- Domain Name Service (DNS)
- Dynamic Host Configuration Protocol (DHCP)
- Phone numbers
- 911
- Call centers
- Snail mail forwarding
- …



“Any problem in computer science can be solved by adding another level of indirection.”

–David Wheeler, inventor of the subroutine, or Butler Lampson

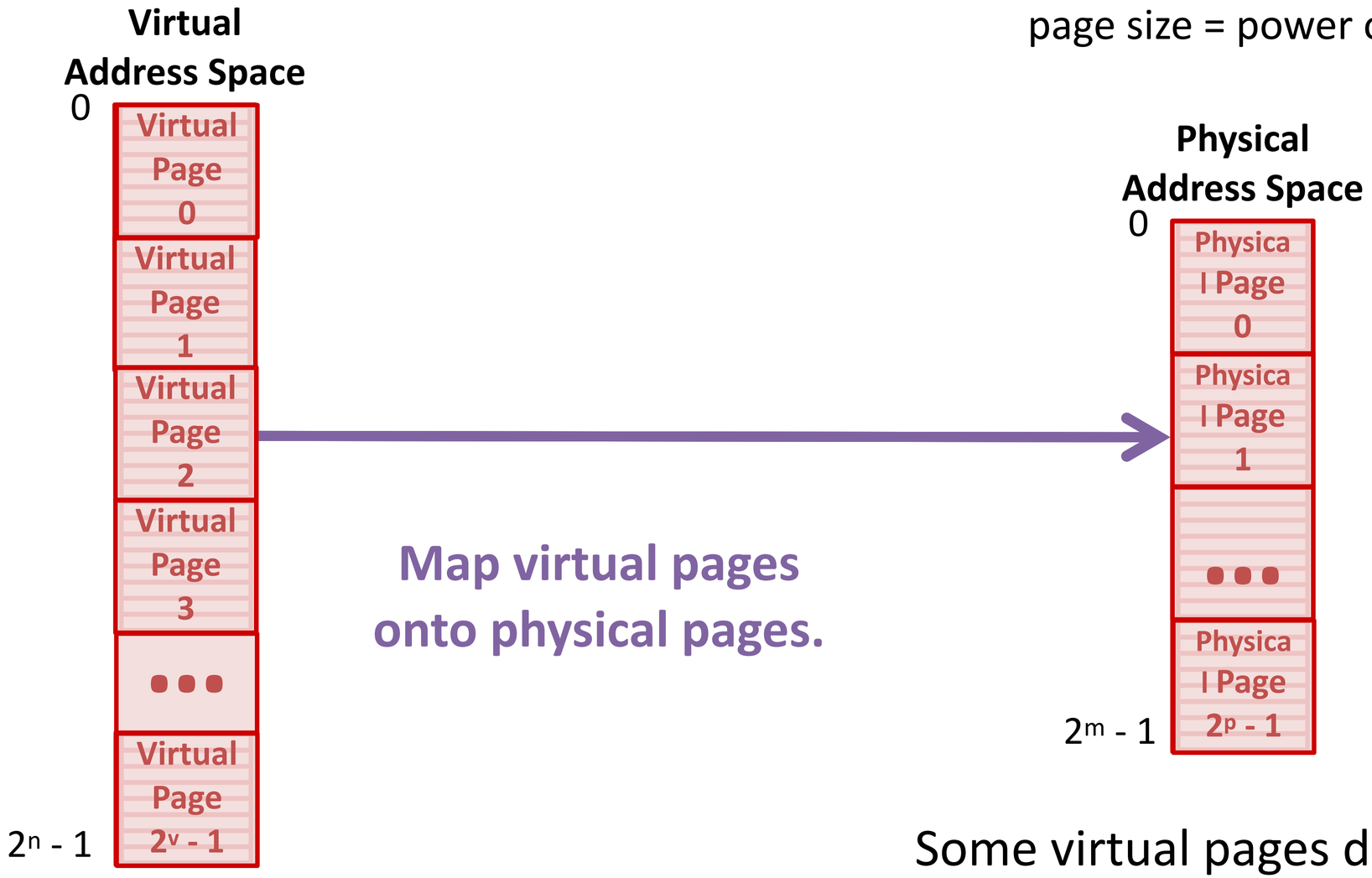
Virtual addressing and address translation



Page-based mapping

fixed-size, aligned ***pages***

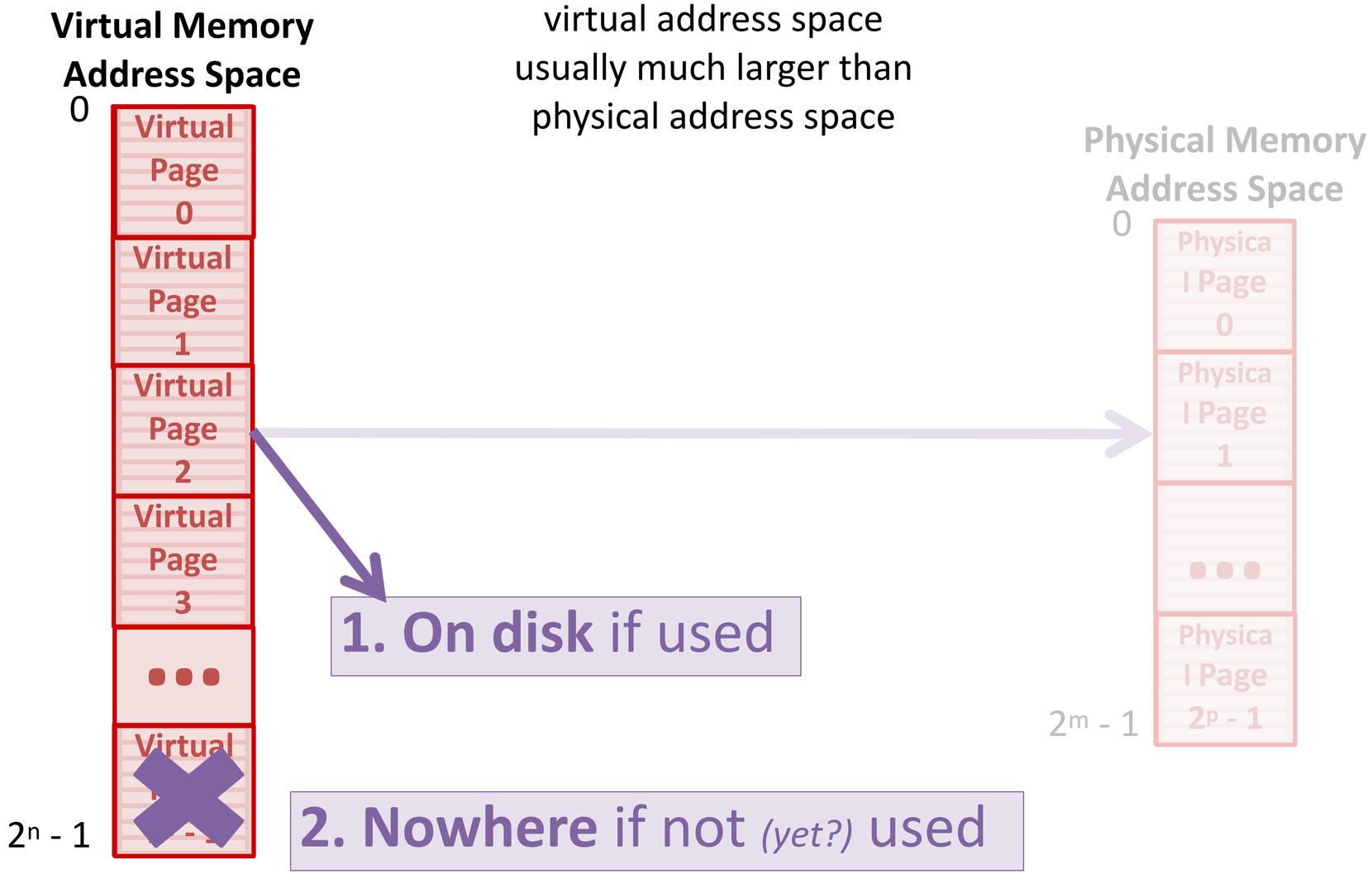
page size = power of two



Some virtual pages do not fit!

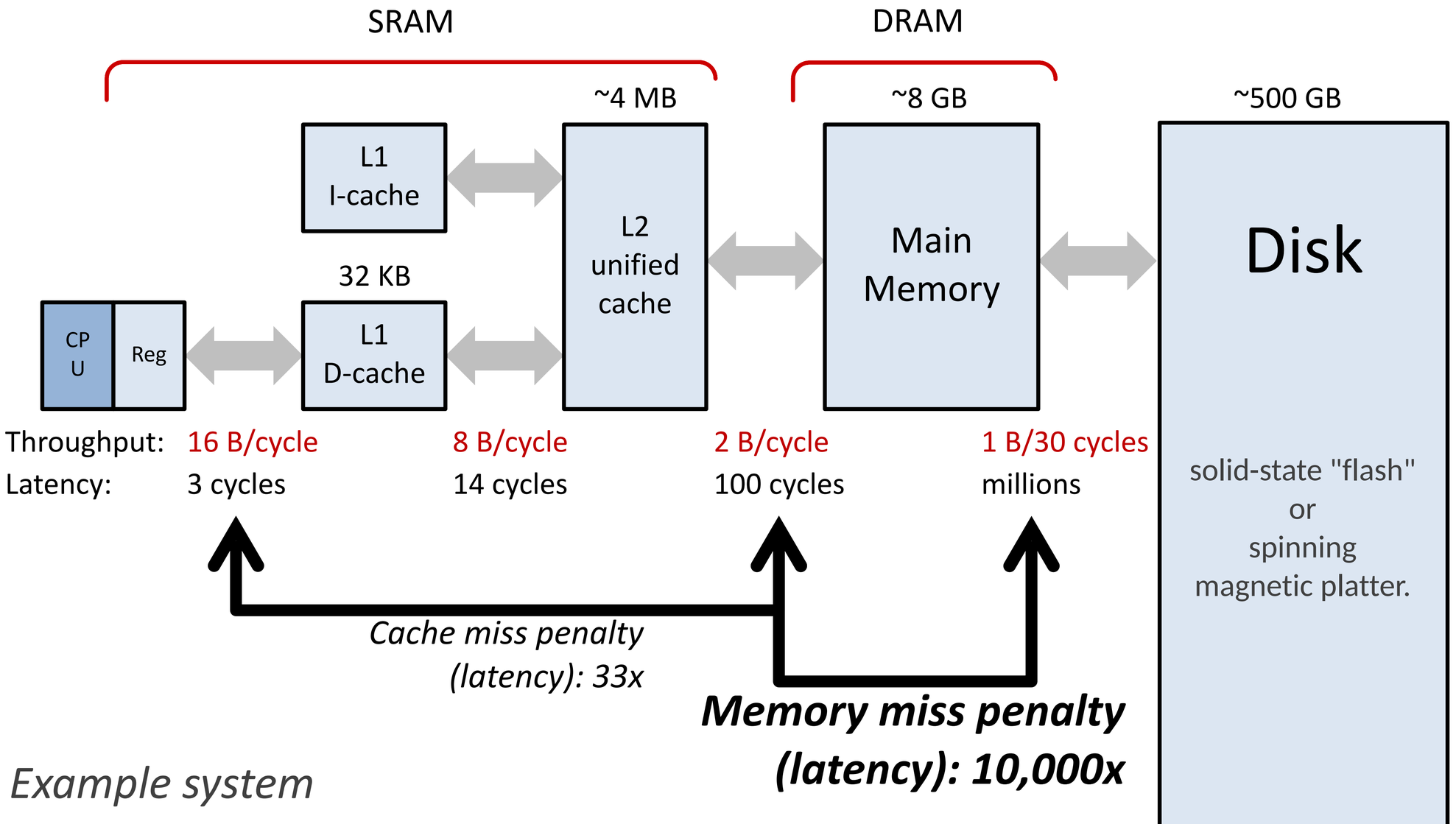
Where are they stored?

Cannot fit all virtual pages! Where are the rest stored?



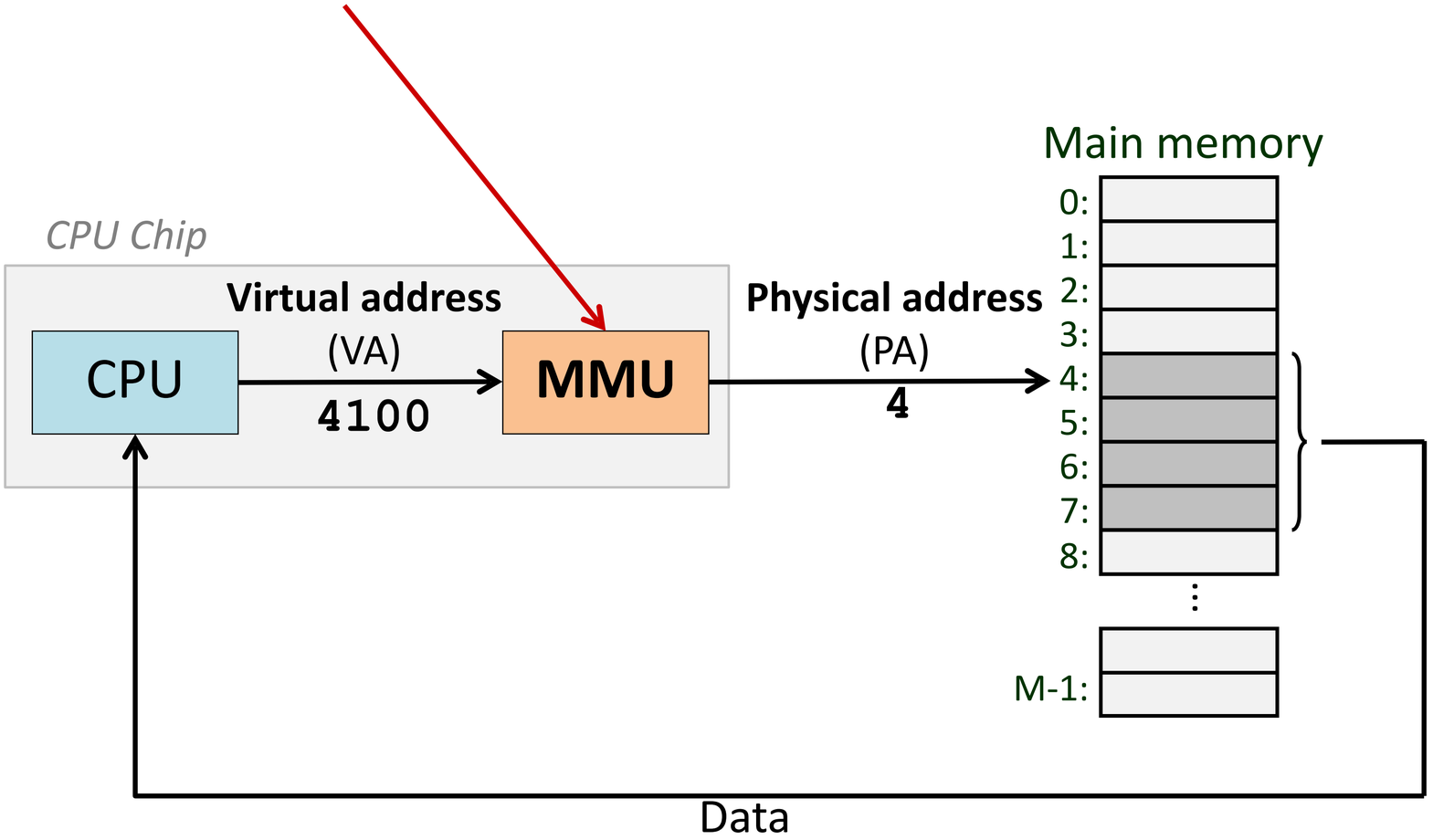
Virtual memory: cache for disk?

Not drawn to scale!



Example system

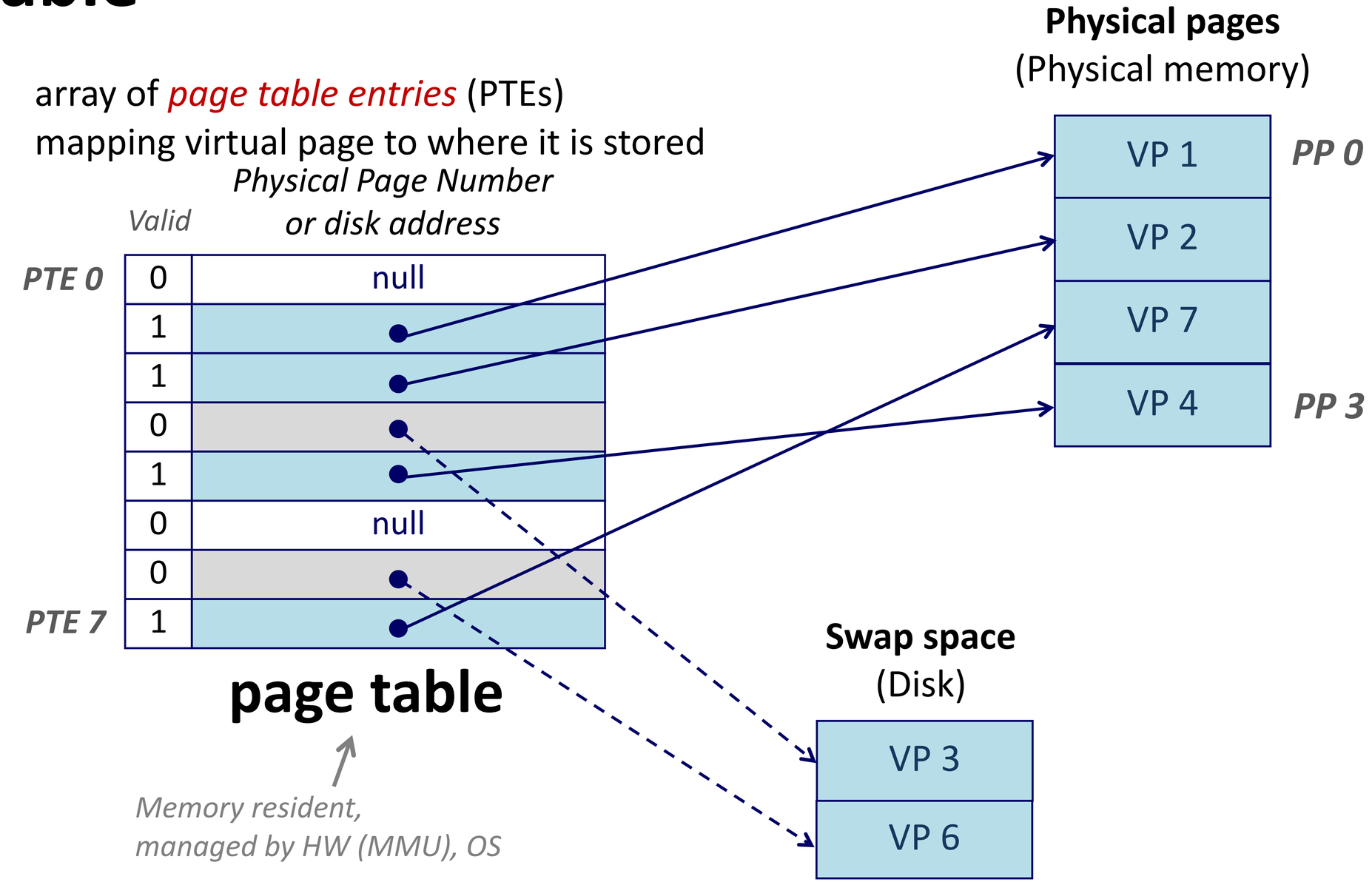
Address translation



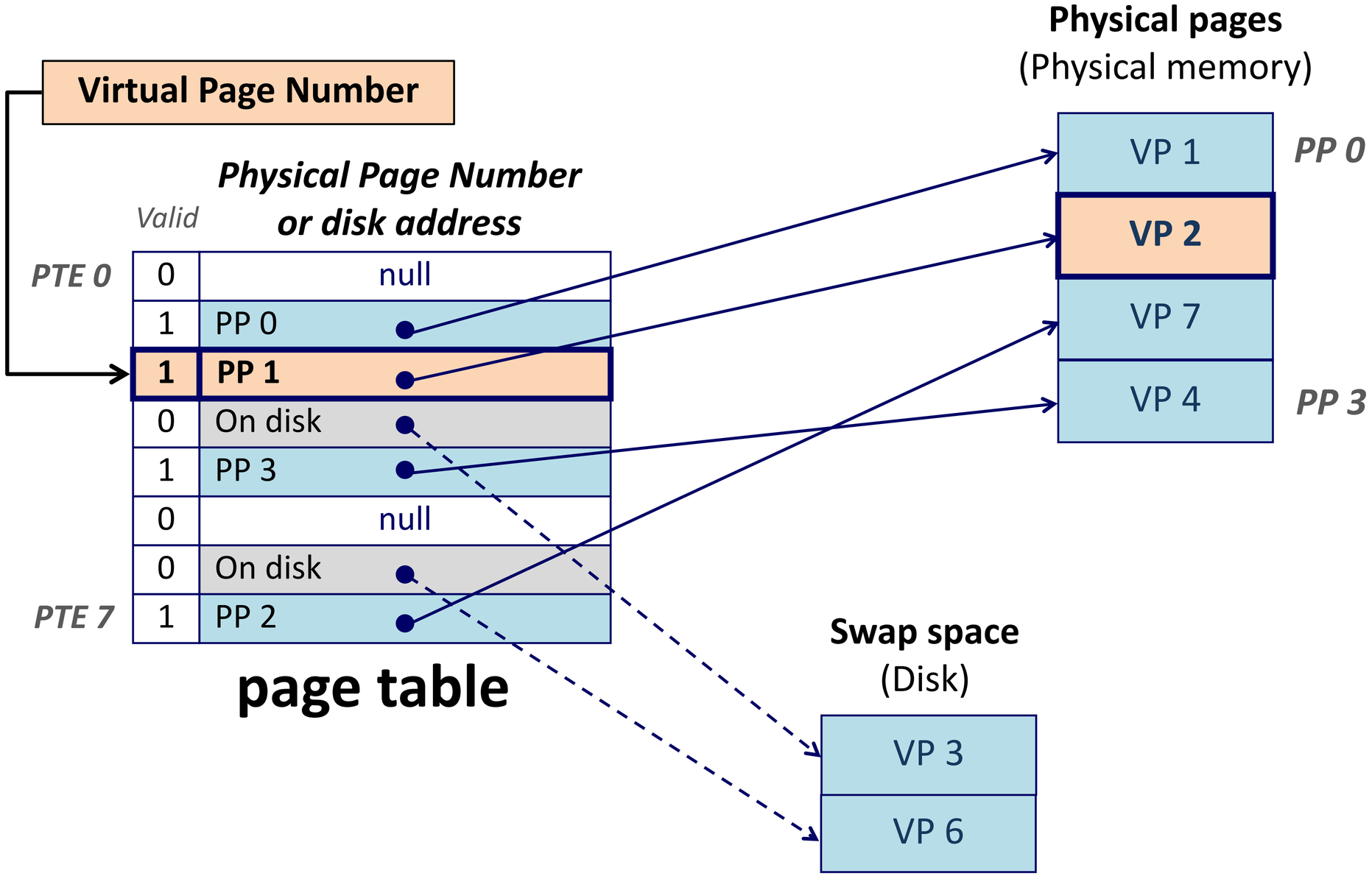
Page table

array of *page table entries* (PTEs)

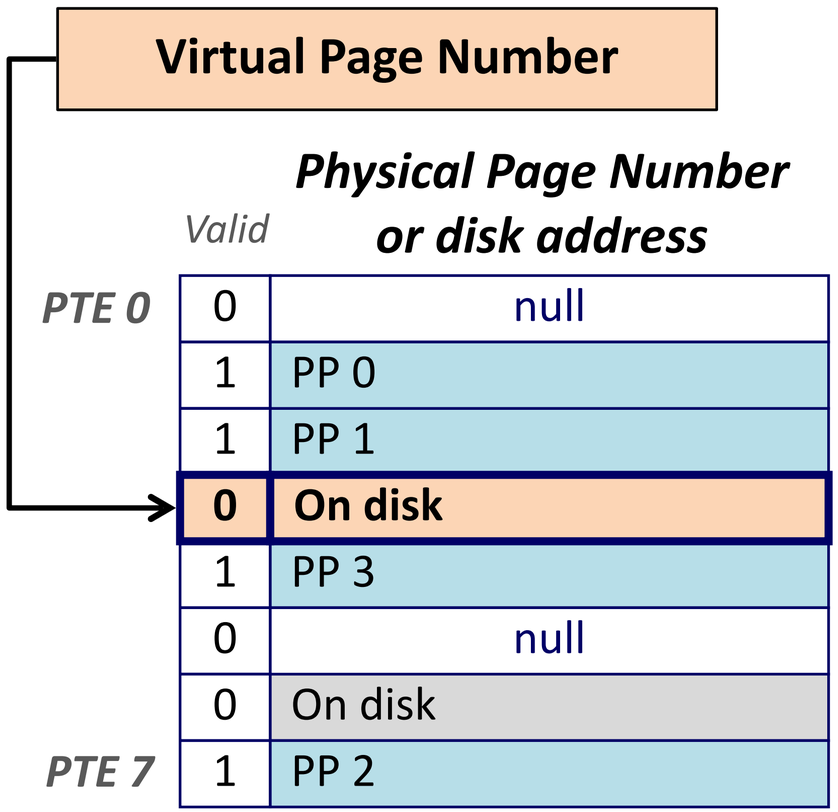
mapping virtual page to where it is stored



Page *hit:* *virtual page is in memory*

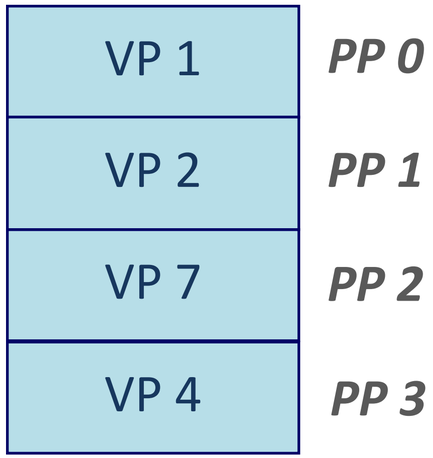


Page *fault:*

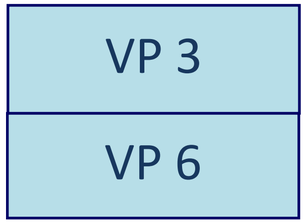


page table

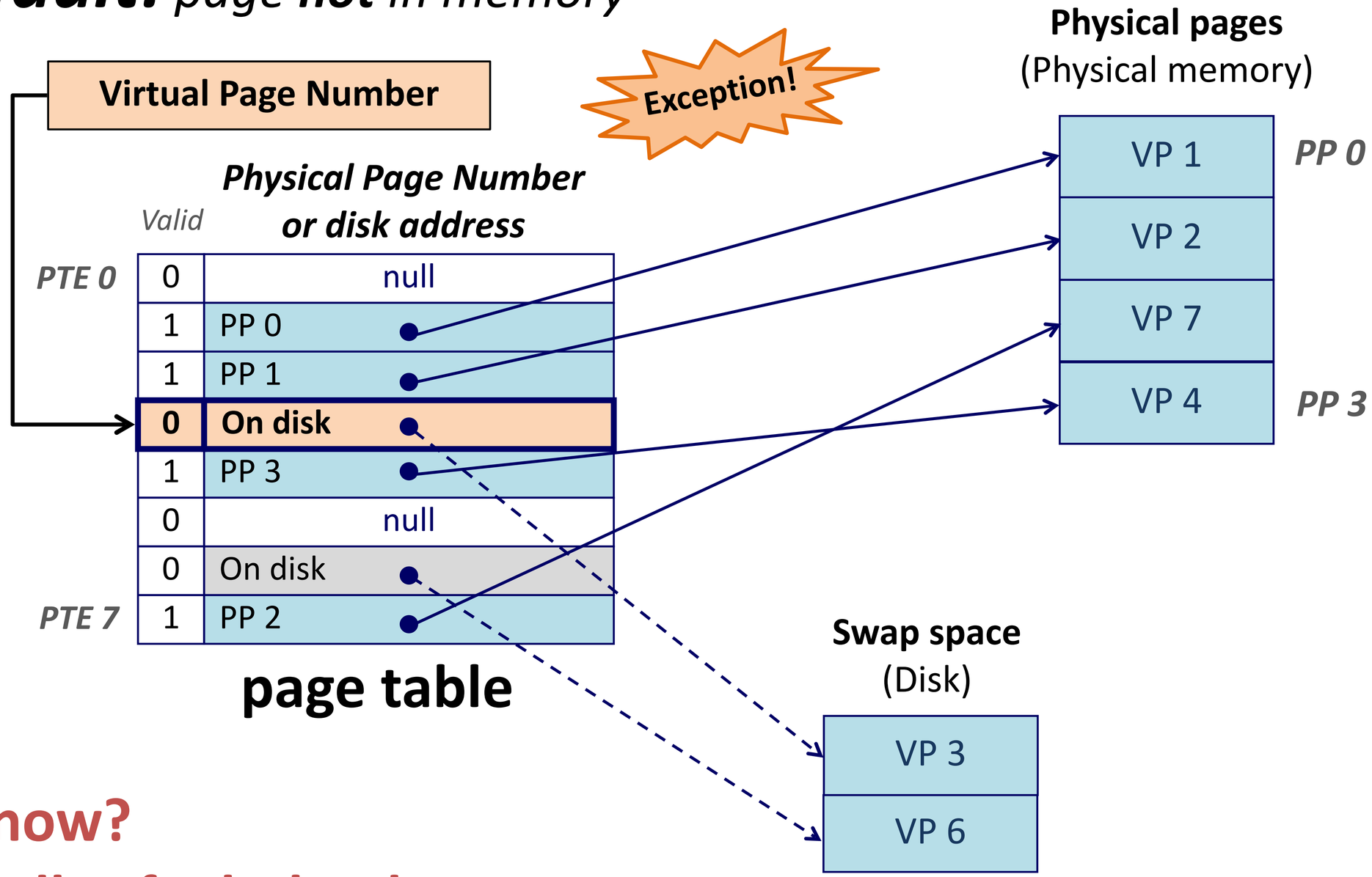
Physical pages
(Physical memory)



Swap space
(Disk)



Page *fault:* *page* ***not*** *in memory*



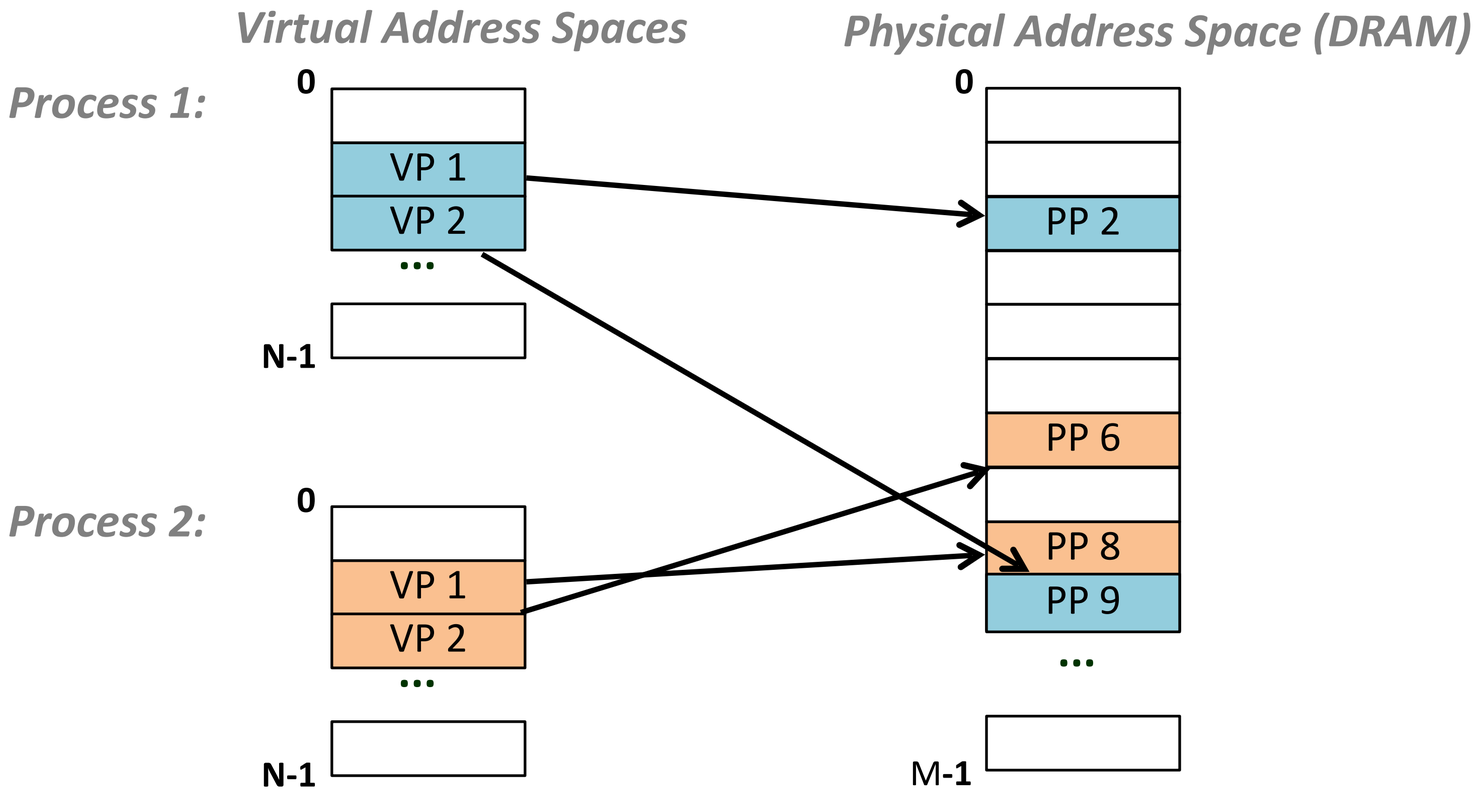
What now?

OS handles fault, loads page

Virtual memory benefits:

Simple address space allocation

Process needs private *contiguous* address space.



Virtual memory benefits:

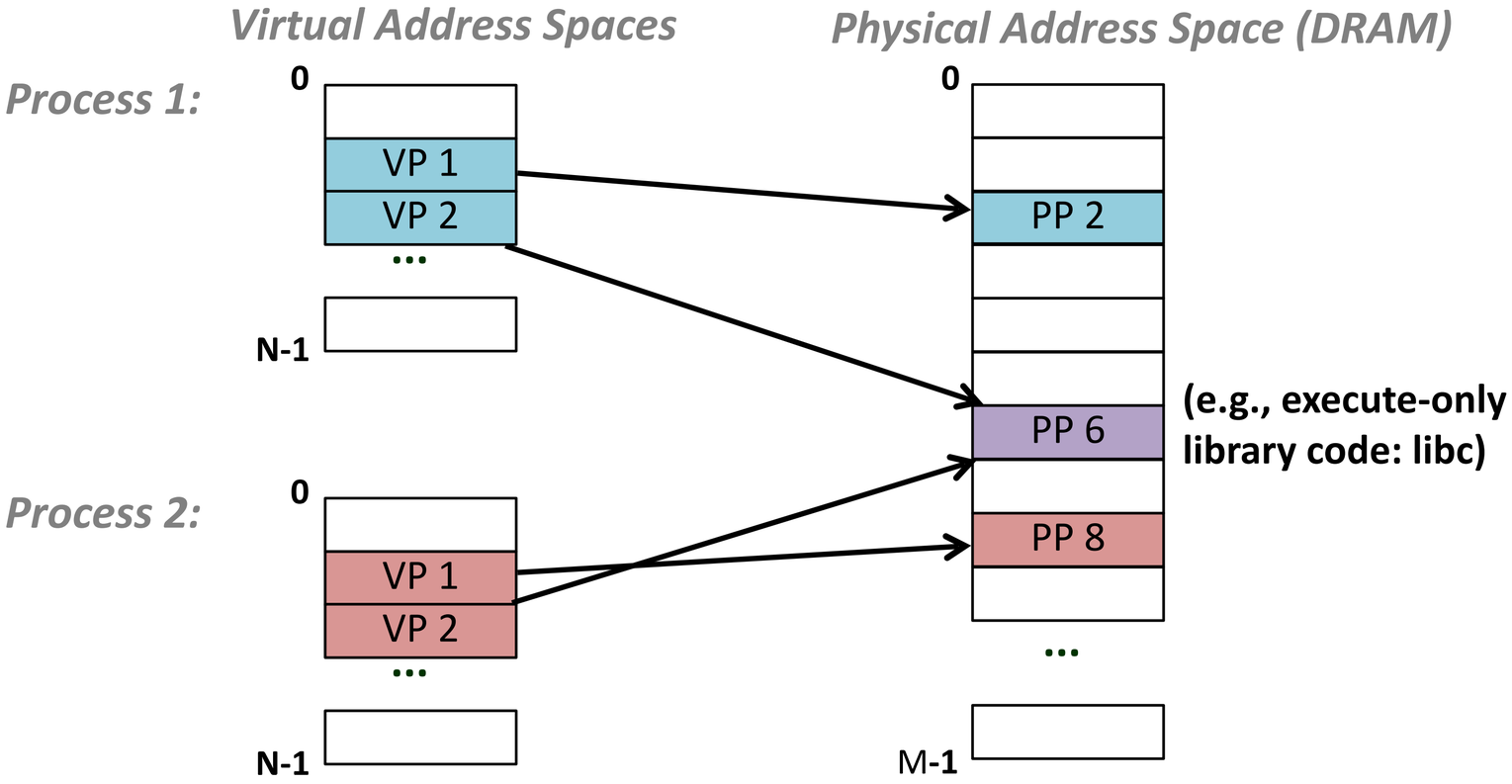
Protection:

All accesses go through translation.

Impossible to access physical memory not mapped in virtual address space.

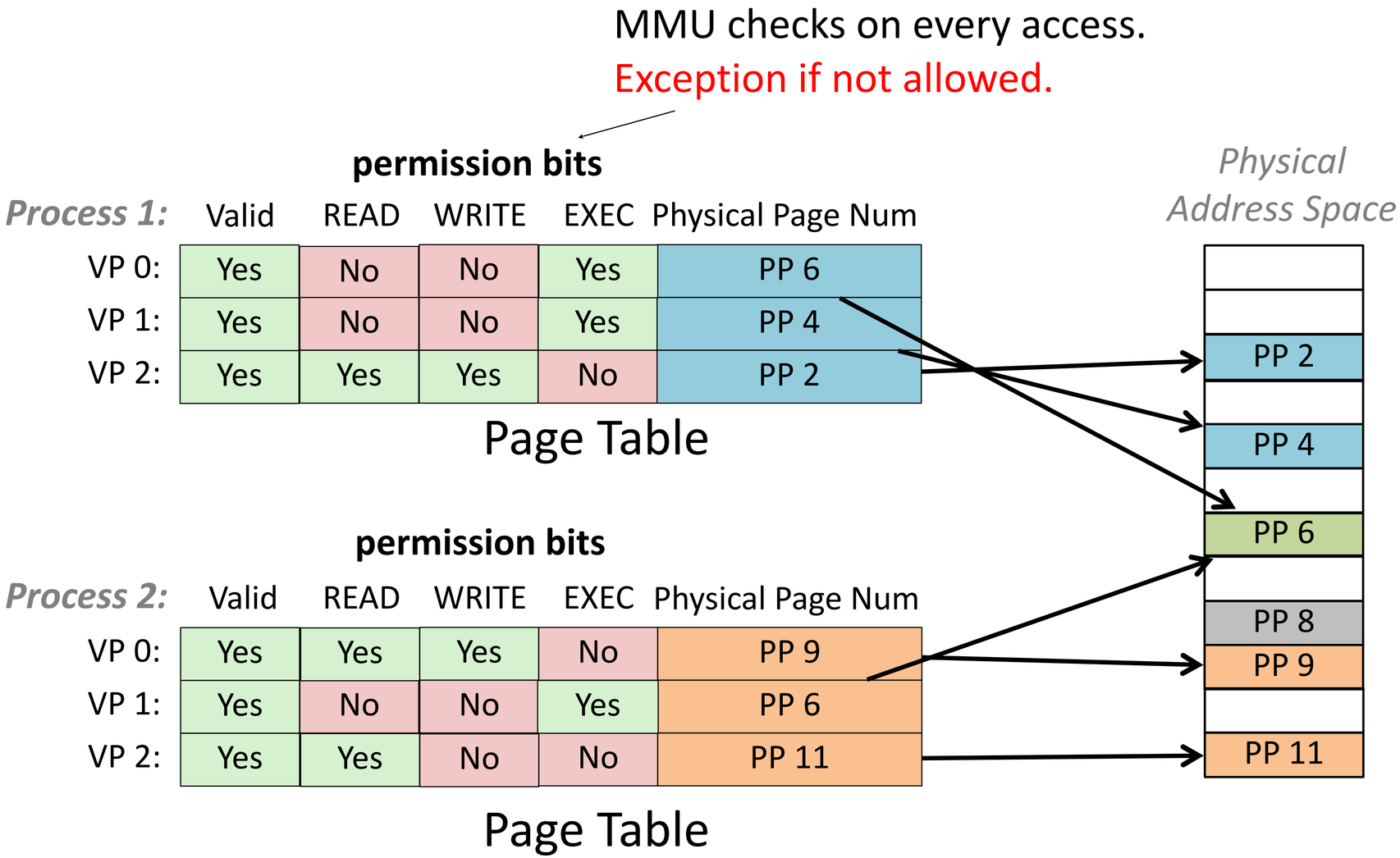
Sharing:

Map virtual pages in separate address spaces to same physical page *(PP 6).*



Virtual memory benefits:

Memory permissions



Summary: **virtual memory**

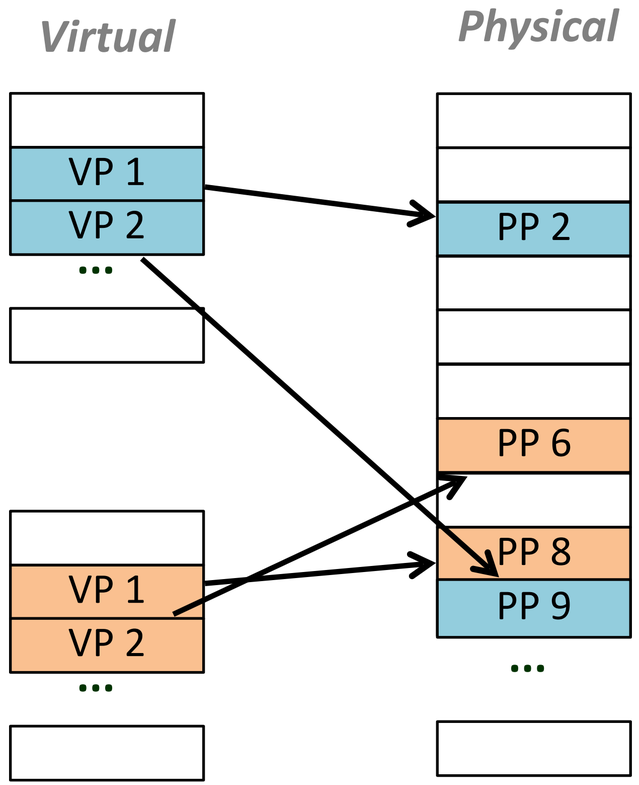
Programmer's view of virtual memory

Each process has its own private linear address space
Cannot be corrupted by other processes

System view of virtual memory

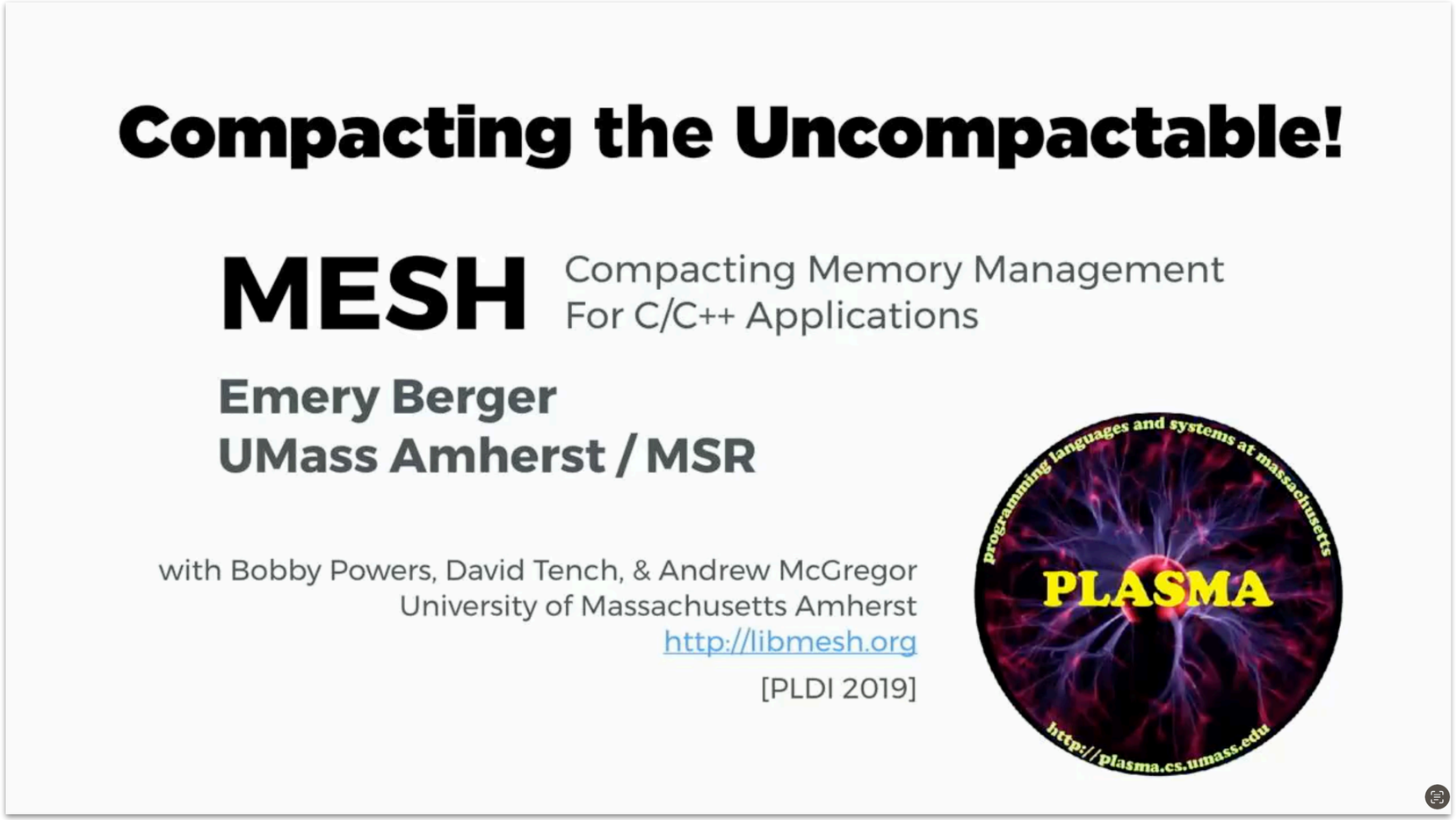
Uses memory efficiently (due to locality) by caching virtual memory pages
Simplifies memory management and sharing
Simplifies protection -- easy to interpose and check permissions
More goodies:

- Memory-mapped files
- Efficient shared pages (e.g., C library code)
- Cheap `fork()` with copy-on-write pages (COW)



Clever real-world application of Virtual Memory

One of Alexa's all time favorite research papers!



https://www.youtube.com/watch?v=xb0mVfnvkp0