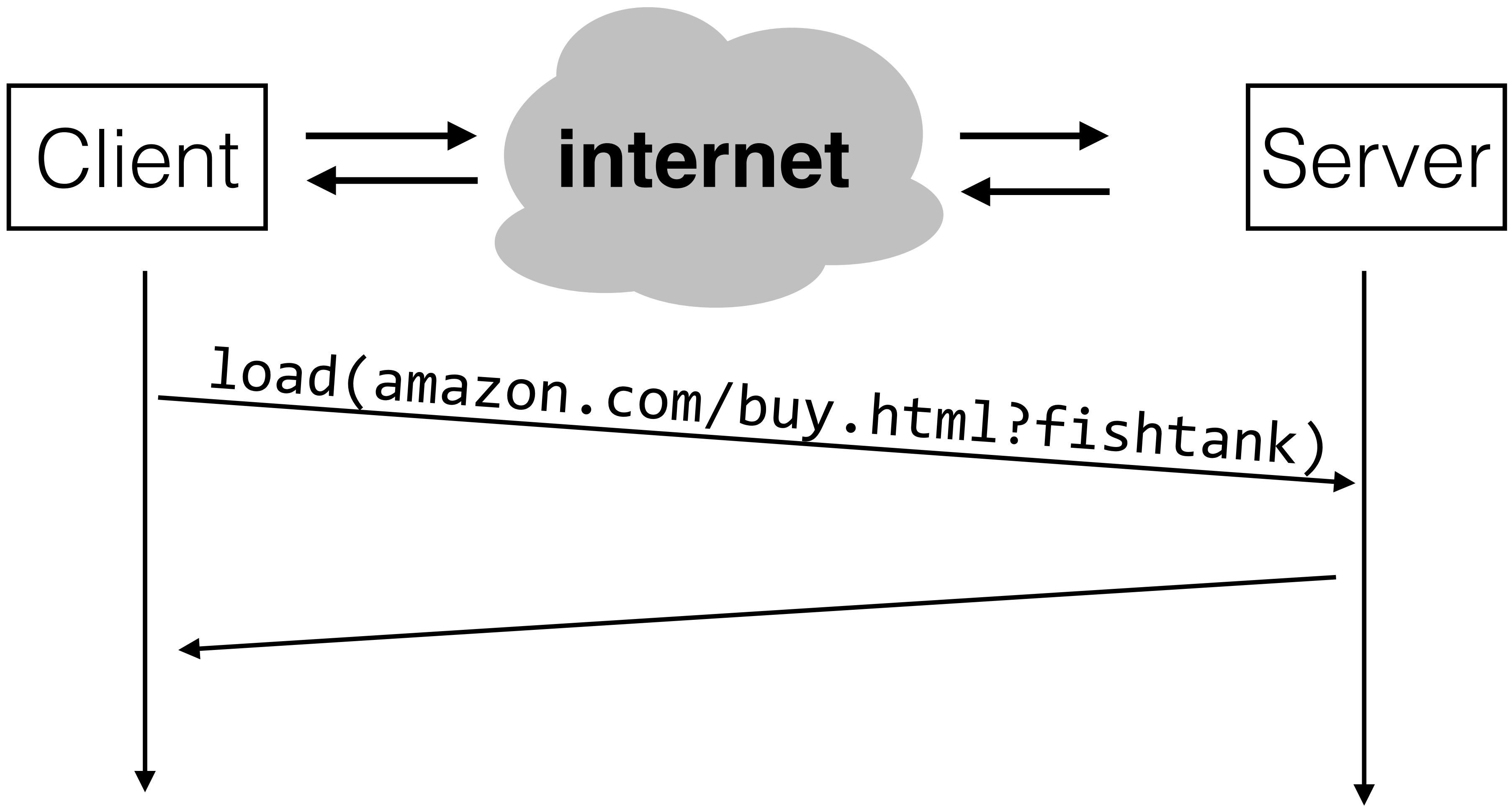
6.033 Spring 2015

Lecture #4

- Operating systems
- Virtual memory
- OS abstractions

Lingering Problem



what if we don't want our modules to be on entirely separate machines? how can we **enforce modularity on a single machine**?

operating systems: enforce modularity on a single machine via **virtualization**

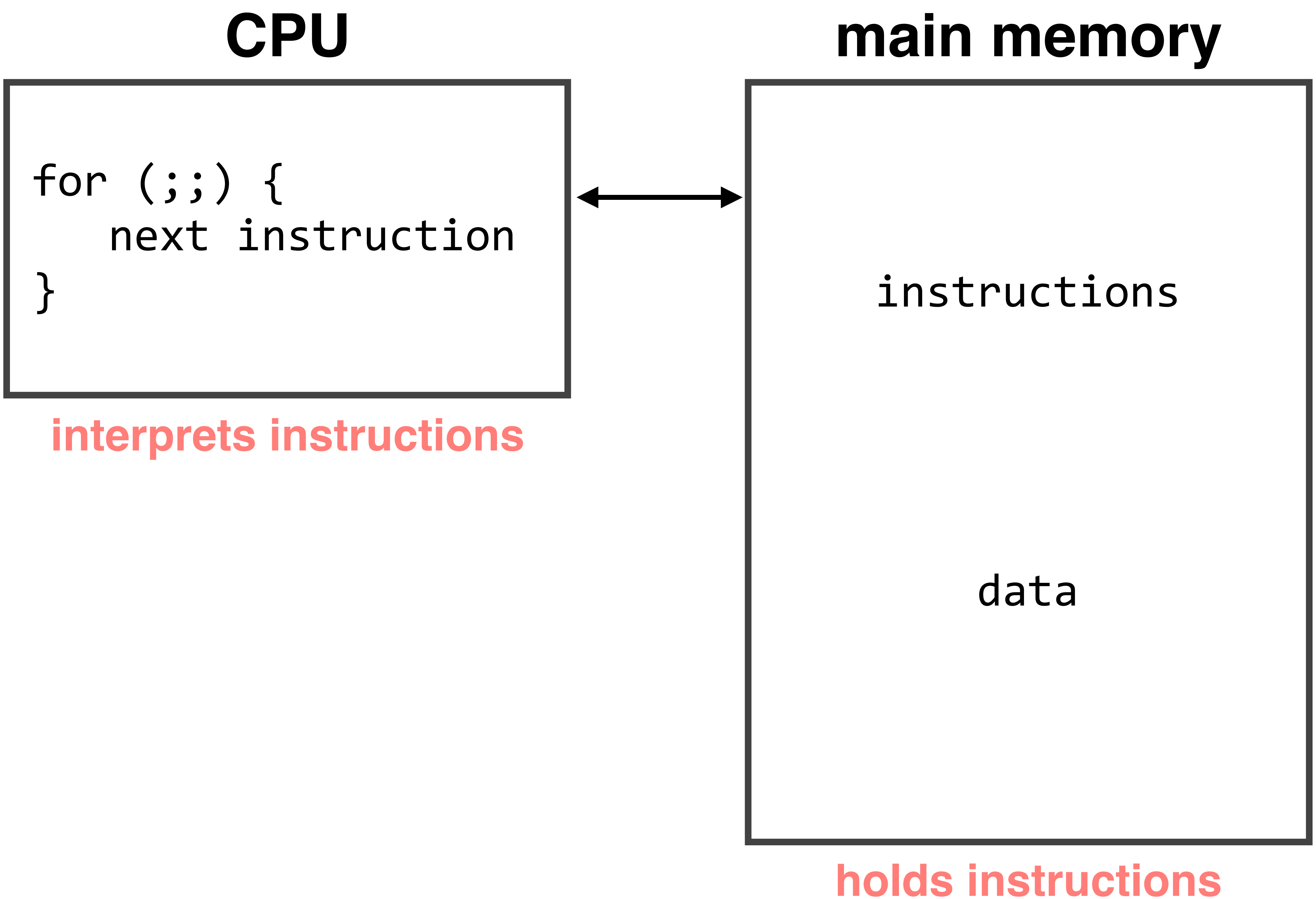
Enforcing Modularity via Virtualization

in order to enforce modularity + build an effective operating system

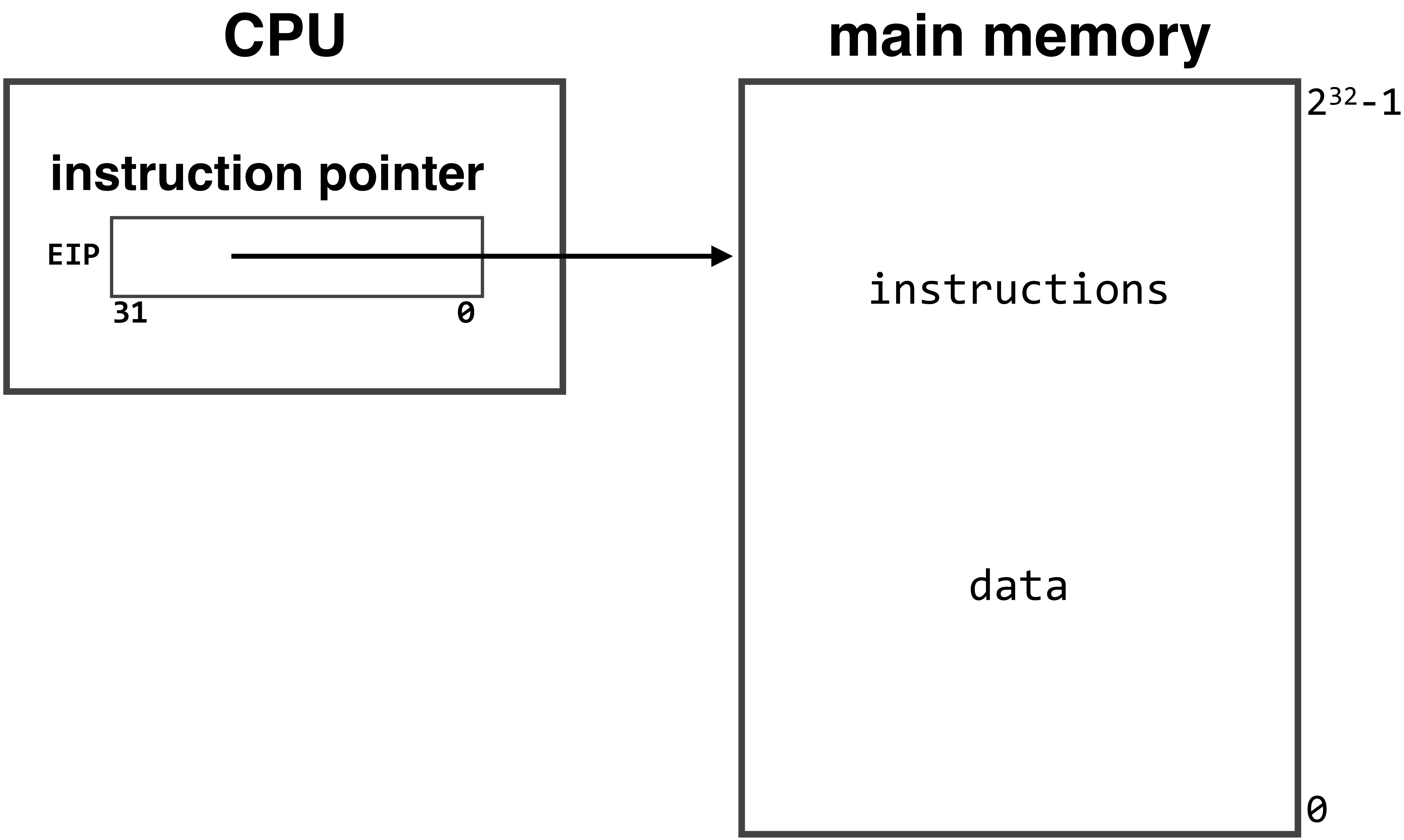
1. programs shouldn't be able to refer to (and corrupt) each others' **memory**	→	**virtual memory**
2. programs should be able to **communicate**	→	assume that they don't need to (for today)
3. programs should be able to **share a CPU** without one program halting the progress of the others	→	assume one program per CPU (for today)

today's goal: virtualize memory so that programs cannot refer to each others' memory

Single Program



Single Program



Multiple Programs

CPU_1 (used by $program_1$)

```
for (;;) {
   next instruction
}
```

CPU_2 (used by $program_2$)

```
for (;;) {
   next instruction
}
```

main memory

$2^{32}-1$

instructions for $program_1$

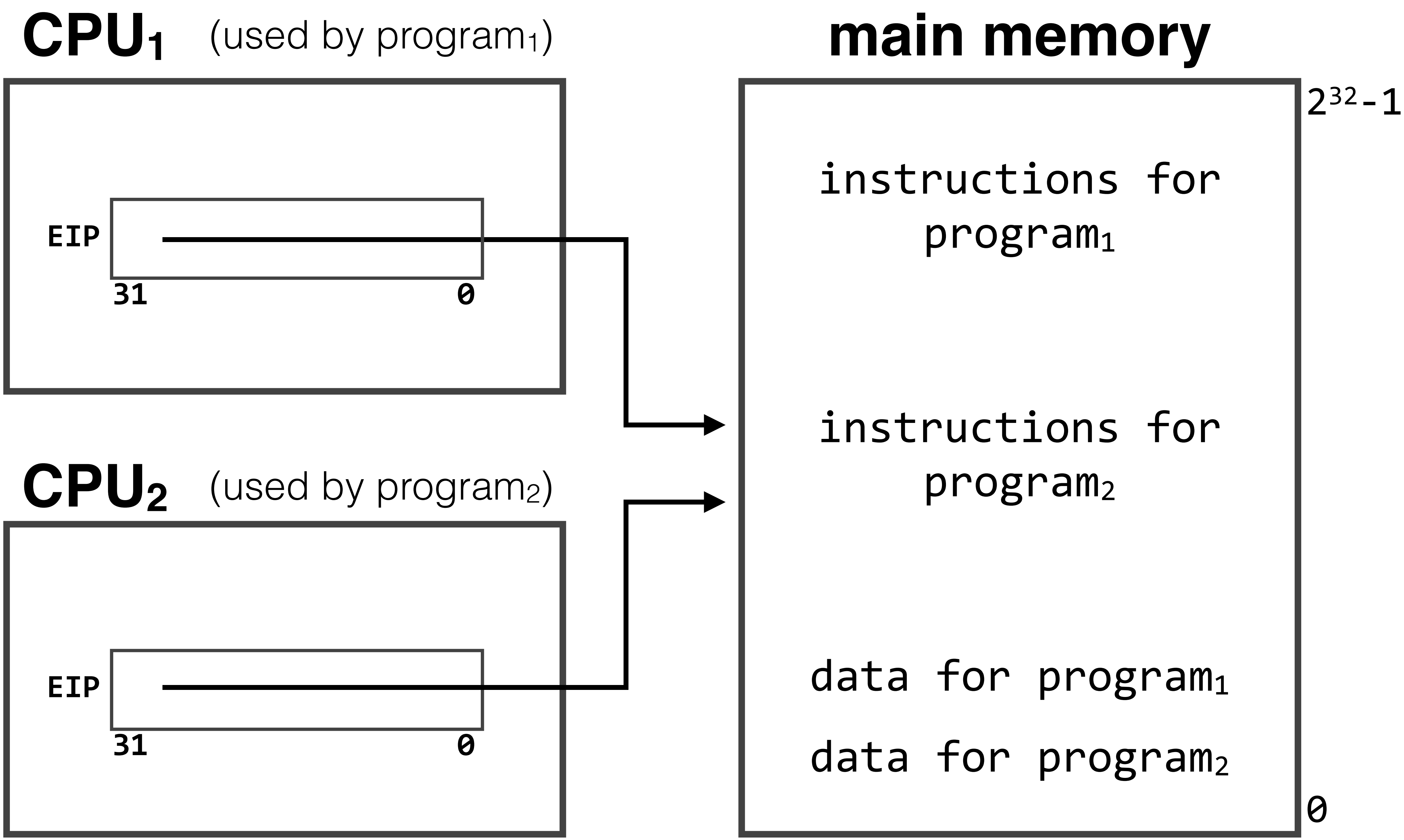
instructions for $program_2$

data for $program_1$

data for $program_2$

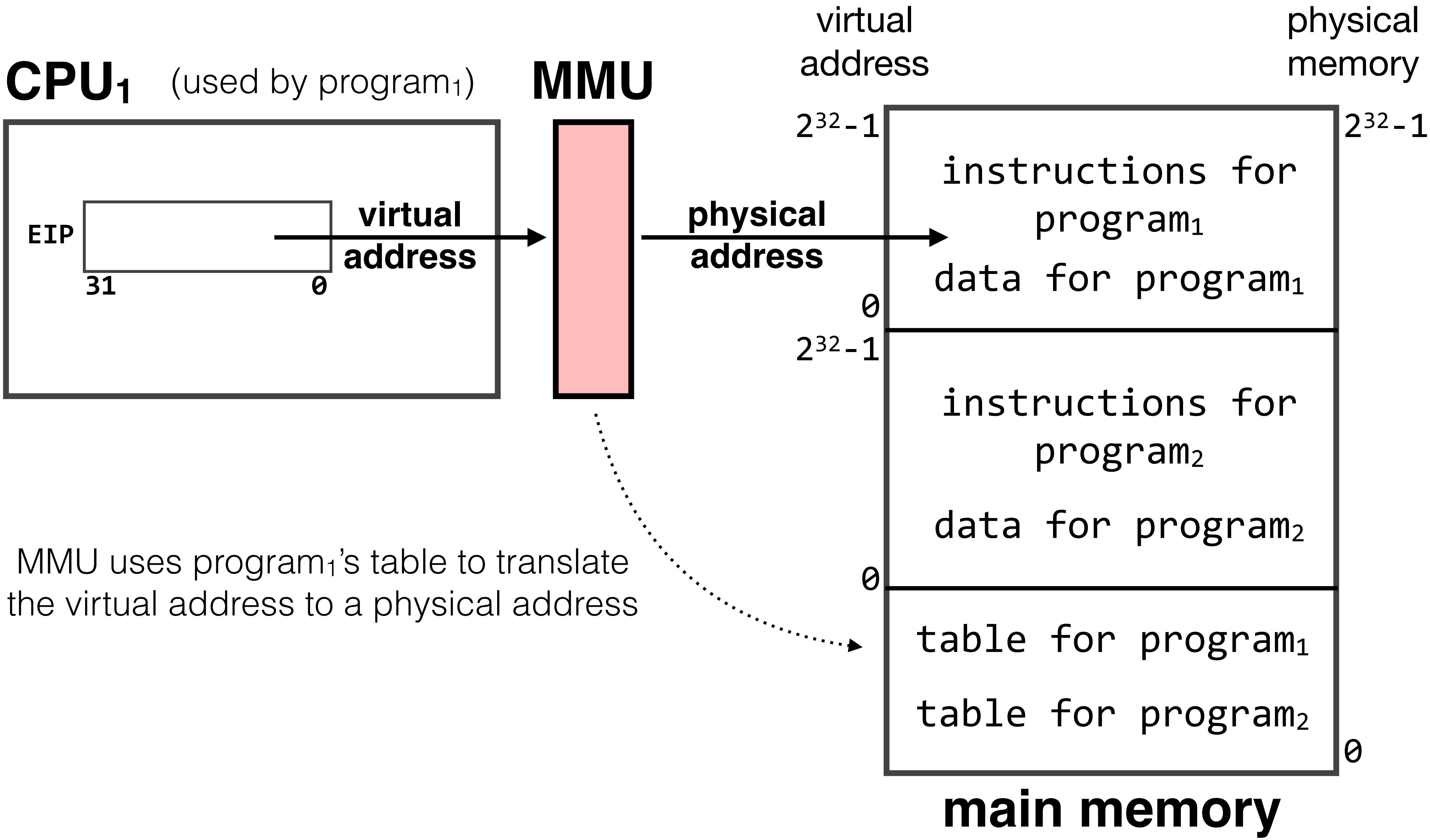
0

Multiple Programs



problem: no boundaries

Solution: Virtualize Memory



Storing the Mapping

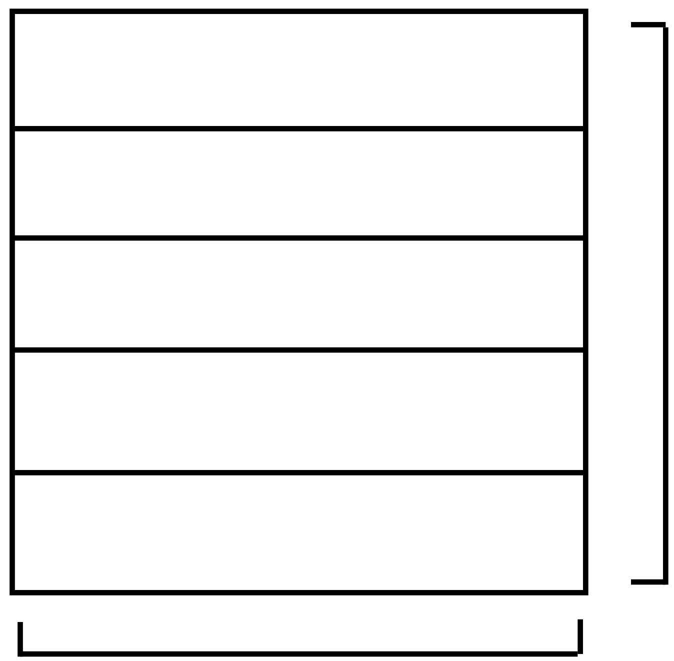
naive method: store every mapping; virtual address acts as an index into the table

= **16GB** to store the table

Storing the Mapping

space-efficient mapping: map to **pages** in memory

one page is (typically) 2^{12} bits of memory.



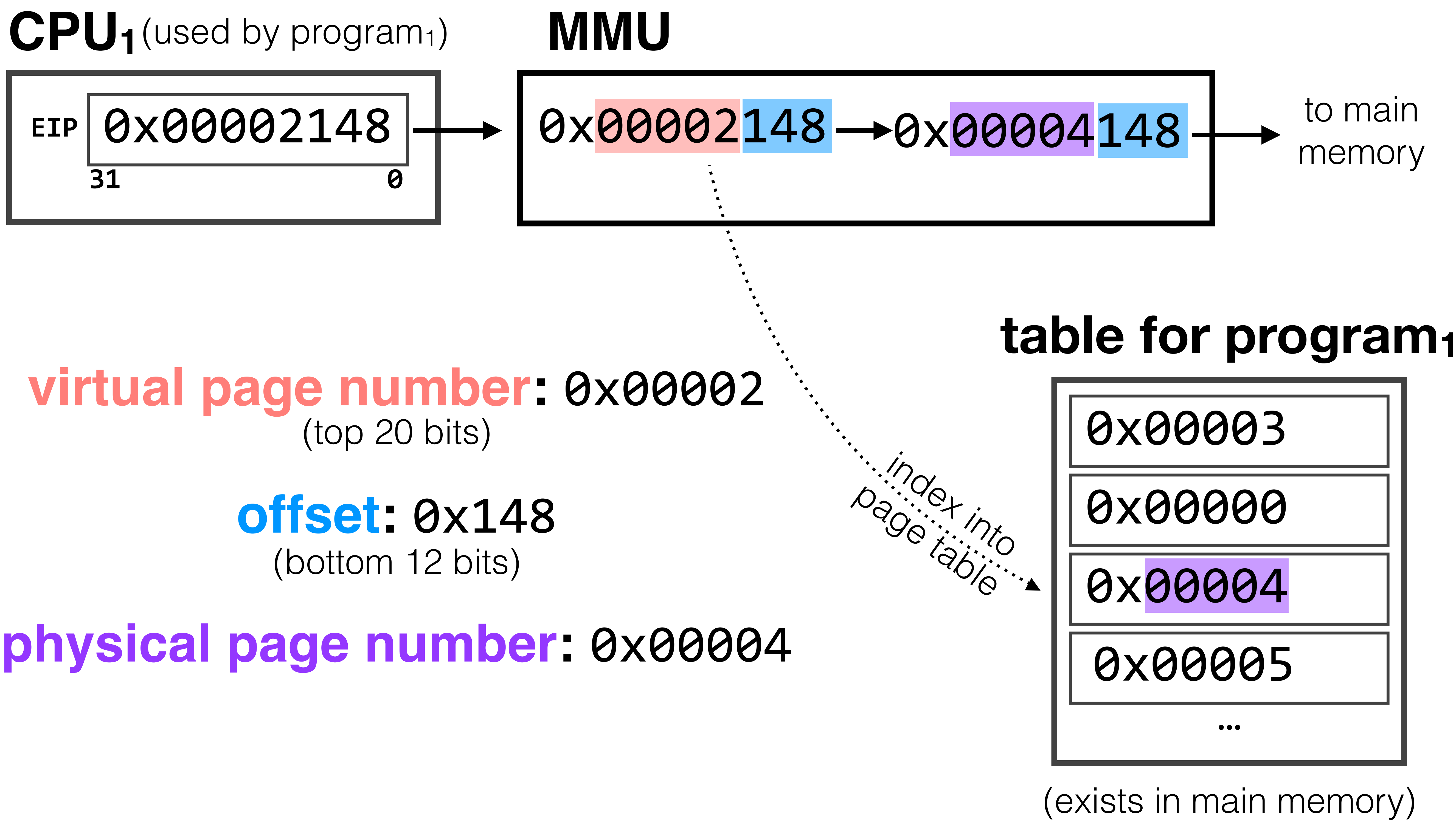
$2^{32-12} = 2^{20}$ entries

32 bits* per entry

= **4MB** to store the table

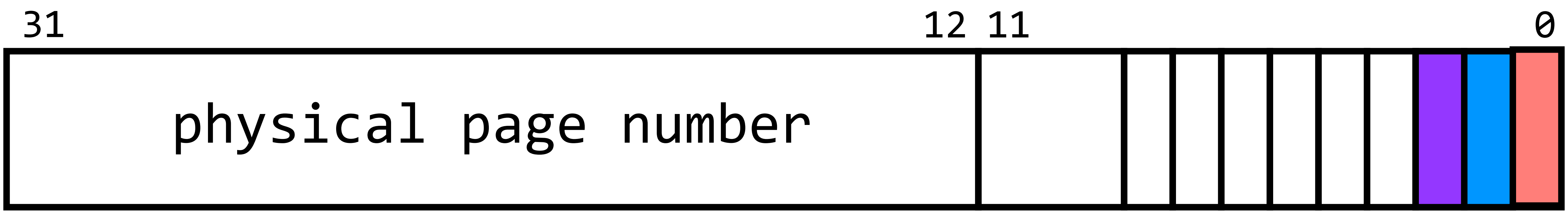
* you'll see why it's not 20 bits in a second

Using Page Tables



Page Table Entries

page table entries are 32 bits because they contain a 20-bit physical page number and 12 bits of additional information



present (P) bit: is the page currently in DRAM?

read/write (R/W) bit: is the program allowed to write to this address?

user/supervisor (U/S) bit: does the program have access to this address?

kernel manages **page faults** and other **interrupts**

operating systems: enforce modularity on a single machine via **virtualization** and **abstraction**

- **Operating systems**
  Operating systems enforce modularity on a single machine via **virtualization** and **abstraction**

- **Virtual memory**
  Virtualizing memory prevents programs from referring to (and corrupting) each other's memory. The **MMU** translates virtual addresses to physical addresses using **page tables**

- **OS abstractions**
  The OS presents abstractions for devices via system calls, which are implemented with interrupts. Using interrupts means the **kernel** directly accesses the devices, not the user