CS202 (003): Operating Systems
Scheduling (cont.), Virtual Memory

Instructor: Jocelyn Chen

Most of the materials covered in this slide come from the lecture notes of Mike Walfish's CS202

Last Time

Another way of optimization: fair-share scheduler

Try to guarantee that each job obtain a certain percentage of CPU time

Lottery scheduling

Tickets: the share of a resource that a process should receive

The percent of tickles that a process has represents its share of the system resources

Hold a lottery to determine which process should get to run next, every now and then

Let p_i has t_i tickets

Let T be total # of tickets, $T = \sum_i t_i$

Chance of winning the next quantum $= \frac{t_i}{T}$

Control long-term average proportion of CPU for each process!

Lottery scheduling

Hold a lottery to determine which process should get to run next, every now and then

Advantages

- Deals with starvation (if you have ticket, you will make progress)
- Don’t worry that adding one high priority job will starve all others
- Adding/deleting jobs affects all jobs proportionally
- Can transfer tickets between processes
- Flexible by using ticket as a currency

Disadvantages

- Latency is unpredictable
- Expected error somewhat high

Follow-up work to reduce randomness -> Stride Scheduling (see textbook for details)

What Linux does: completely fair scheduler (CFS)

It aims to distribute CPU time fairly among all runnable processes using a virtual runtime metric.

CFS organizes processes in a red-black tree and selects the one with the lowest virtual runtime to run next. This approach balances fairness, efficiency, and interactivity.

See the textbook for more details

Scheduling, lesson learned

Write down your goals (**policy**) before picking the scheduling algorithm (**mechanism**)

Start from/Compare with the optimal solution, even though it cannot be built

Many schedulers in the system that interact:

mutex, interrupt, disk, network, …

Let’s take a step back…

Process

What is a process?

Core abstraction inside a process

How does process communicate with low-level resources?

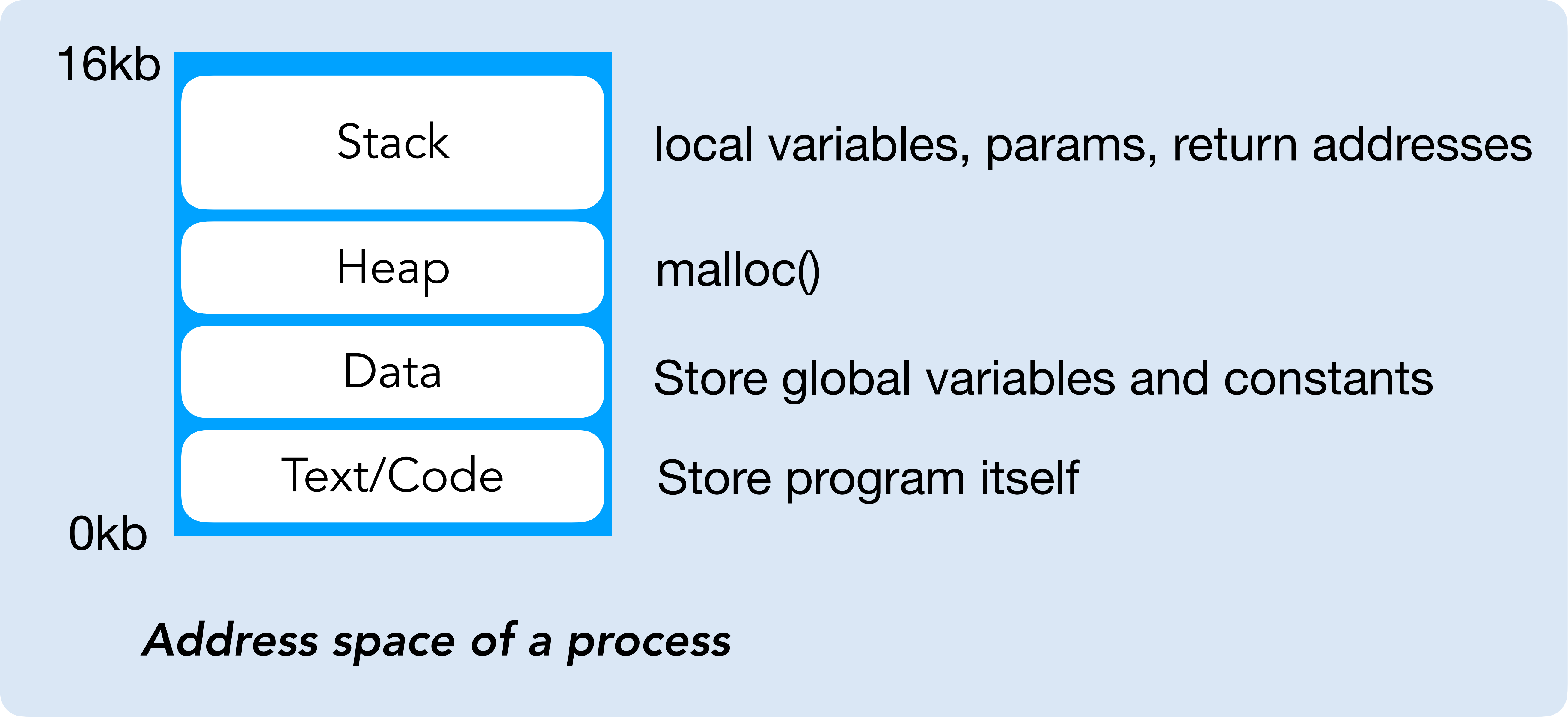
How can we run multiple process/ threads at the same time?

How can one process do multiple tasks concurrently?

How does the operating system manage memory for multiple processes efficiently?

Virtual Memory

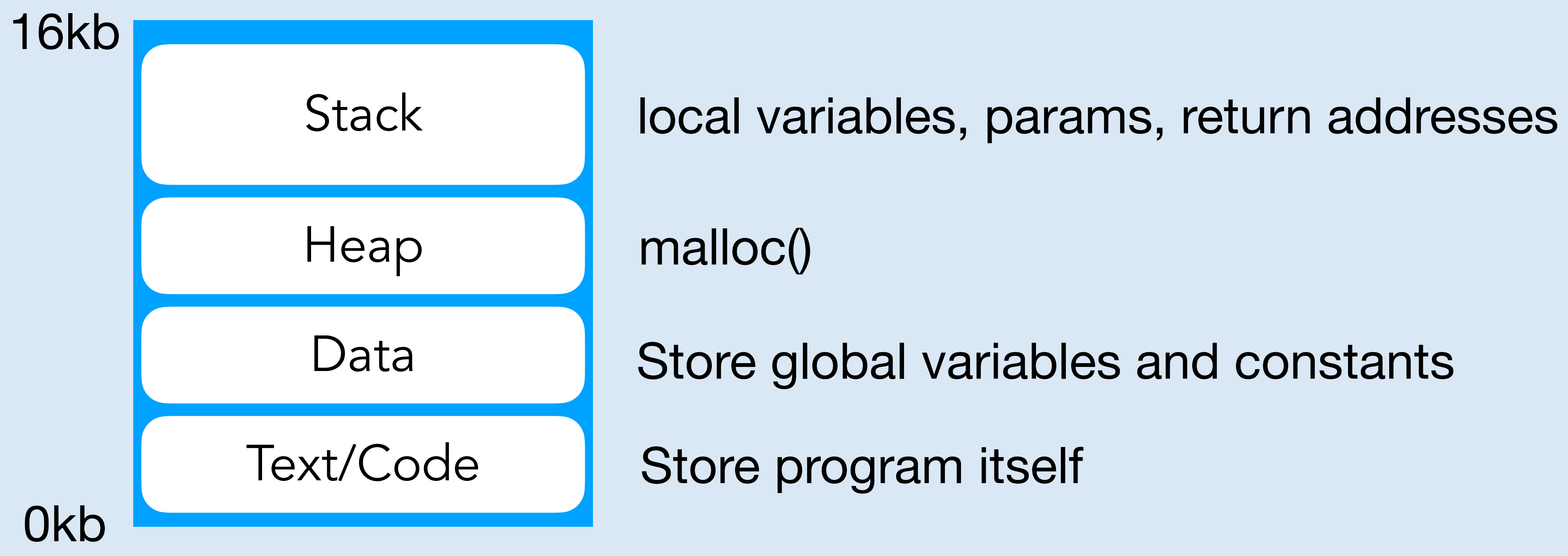
"Each process has its own view of memory"



Does the address space of this program actually at the physical addresses 0 through 16KB?

Virtual Memory

"Each process has its own view of memory"



Address space of a process

Does the address space of this program actually at the physical addresses 0 through 16KB?

Goals/Benefits of Virtual memory

Programmability

- Program thinks it has a lot of memory, and has its own physical memory
- Compiler and linker don’t have to worry about physical addresses
- multiple instances of the programs can be loaded and not collide

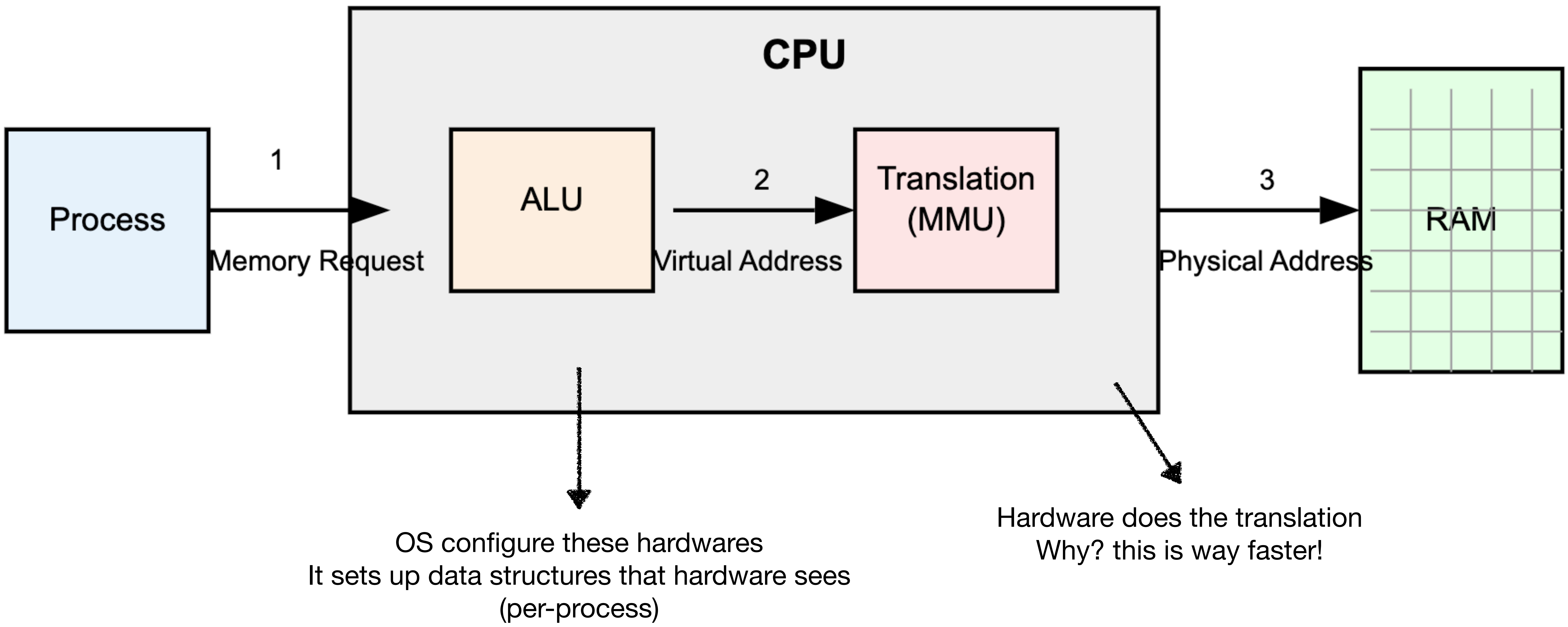
Protection

- Program cannot read/write each other’s memory
- Therefore delivers isolation (prevent bug in one process corrupt with another)

Efficient use of resources

- Programmers don't have to worry that the sum of the memory consumed by all active processes is larger than physical memory

Mapping virtual memory to physical memory



Paging

Divide all memory (physical and virtual) into **fixed-size** chunks

Pages!

Page Size

In the traditional x86 (and in our labs), the page size will be 4096 B = 4 KB = 2^{12}

2^{10}: kilo
2^{20}: mega
2^{30}: giga
2^{40}: tera

How many pages are there on a 32-bit architecture?

$$\frac{2^{32} \text{ bytes}}{2^{12} \text{ bytes/page}} = 2^{20} \text{ pages}$$

What about if there are 48 bits used to address memory?

$$\frac{2^{48} \text{ bytes}}{2^{12} \text{ bytes/page}} = 2^{36} \text{ pages} = 64 \text{ billion pages}$$

Paging

Each process has a separate mapping

Each page is separately mapped

OS take control on certain (invalid) operations:

- If a process tries to write to a page marked as read-only, it triggers a trap
- If a process tries to access a page marked as invalid, it triggers a trap

After handling a trap, the OS can modify the memory mapping as needed
(load a page from disk, change permissions, ….)

Paging

Each process has a separate mapping

Each page is separately mapped

OS take control on certain (invalid) operations:

- If a process tries to write to a page marked as read-only, it triggers a trap
- If a process tries to access a page marked as invalid, it triggers a trap

After handling a trap, the OS can modify the memory mapping as needed
(load a page from disk, change permissions, ….)

Page Number

Pages are numbered sequentially

page 0: [0,4095]
page 1: [4096, 8191]
page 2: [8192, 12277]
page 3: [12777, 16384]
page 2^{20}-1: [..., 2^{32}-1]

Size of space = $2^{\#}$ of bits

What's the size of space for 32 bits virtual address?

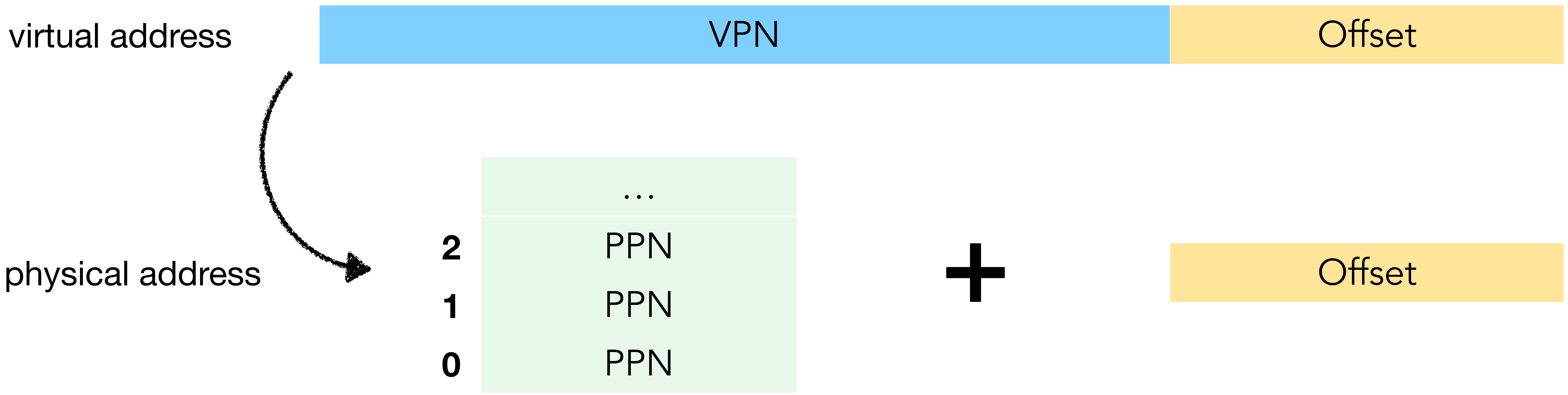
Size of space = 2^{32} bits = 4 GB

Both virtual and physical memory are divided in to pages

VPN (virtual page number)
PPN (physical page number)

Key data structure: page table

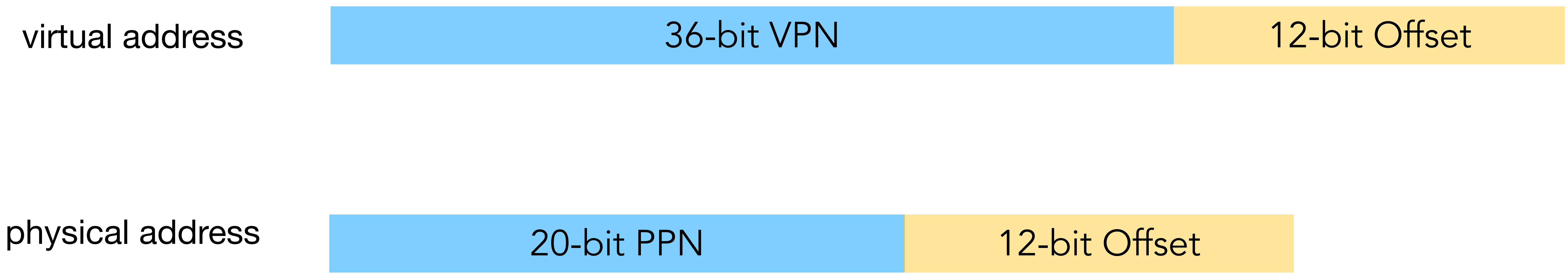
a map from VPN to PPN



Each page table entry expresses a mapping about a contiguous group of address

Another way to look at it

(assuming 48-bit addresses and 4KB pages)



if OS wants a program to be able to use address 0x00402000 to refer to physical address 0x00003000, then the OS conceptually adds an entry:

table[0x00402] = 0x00003

(table[1026] = 3 in decimal)

Create the mapping is hard

Page table can get terribly large!

36-bit VPN => 2^36 translation from VPN to PPN
Assuming each translation is 8 byte => 2^36 * 8 = 512GB
Recall that we are maintaining these mapping per process, 100 process => 51200GB of memory to store address translation!

Most programs only use a small fraction of the available address space,
so it does seem to be a good use of resources

Multi-level page table

Represent a linear page table as a hierarchy of smaller page tables

Each level uses a portion of the virtual address to index into its table

a) The system starts with the root page table.
b) It uses the first part of the address to find an entry in this table.
c) This entry points to a second-level table.
d) The next part of the address is used to index into this second table.
e) This process continues through all levels.
f) The final level provides the actual physical page number.

A virtual address is divided into several parts:

- Multiple segments (often 9 bits each) for indexing each level of tables
- A final segment (often 12 bits) for the offset within the physical page

This tree is space: only fill in parts that are actually in-use!

Multi-level page table

Map 2MB of physical memory at virtual memory 0, …, 2 ^ 21 -1
Let’s say we have 48 bits, and we divide the VPN into 4 9 bits segments
First of all, assuming each physical page is 4 KB, then we have 512 physical pages

The Virtual Address Range: We're mapping addresses from 0 to 2^21 - 1 (2MB).
48-bit Address Structure: (It’s divided as) 9 bits | 9 bits | 9 bits | 9 bits | 12 bits
For the range 0 to 2^21 - 1, **the binary representation** looks like this (X is either 0 or 1):

000000000	000000000	00000XXXX	XXXXXXXXX	XXXXXXXXXXXX
(Level 1)	(Level 2)	(Level 3)	(Level 4)	(Page Offset)

Level 1 (Root):	The first 9 bits are always 000000000 for our entire range. So, we only need one entry in the root table, pointing to the single Level 2 table we'll use.
Level 2:	The next 9 bits are also always 000000000 for our entire range. Again, we only need one entry, pointing to the single Level 3 table.
Level 3:	The next 9 bits start with five 0s, but the last 4 can vary (00000XXXX). However, we only need one entry here because these variations are handled in Level 4. This single entry points to the Level 4 table.
Level 4:	The next 9 bits (XXXXXXXXX) can represent any value from 000000000 to 111111111. This gives us 2^9 = 512 different combinations. That's why we need 512 entries in this level.

Alternatives and tradeoffs

Large/small page size

Large page size: waste actual memory

Small page size: lots of page table entries

Many/few level of mapping

Many level of mapping: Less space spent on page structures when address space is space, but more costly for hardware to walk the page table

Few level of mapping: Need to allocate larger pages, which cost more space, but the hardware has fewer levels of mapping