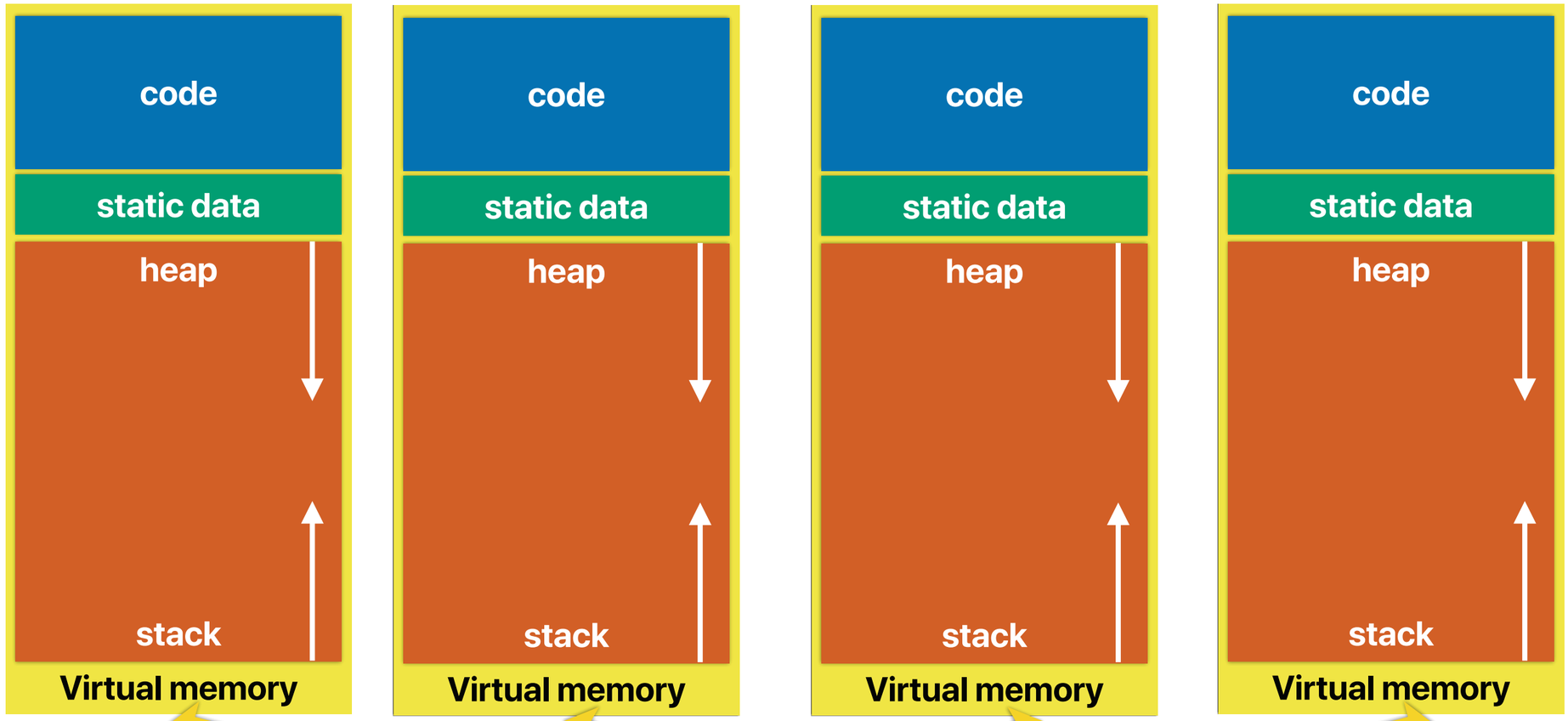
Virtual memory

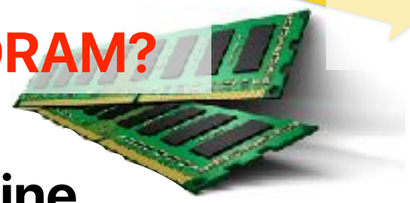
Hung-Wei Tseng

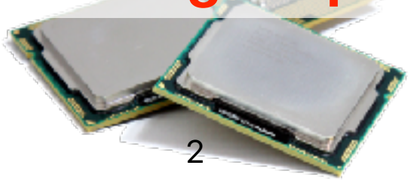
Previously, we talked about virtualization



Previously, we've talked about sharing the processor.

How about sharing DRAM?





The Machine

Reviews: abstractions we've learned so far …

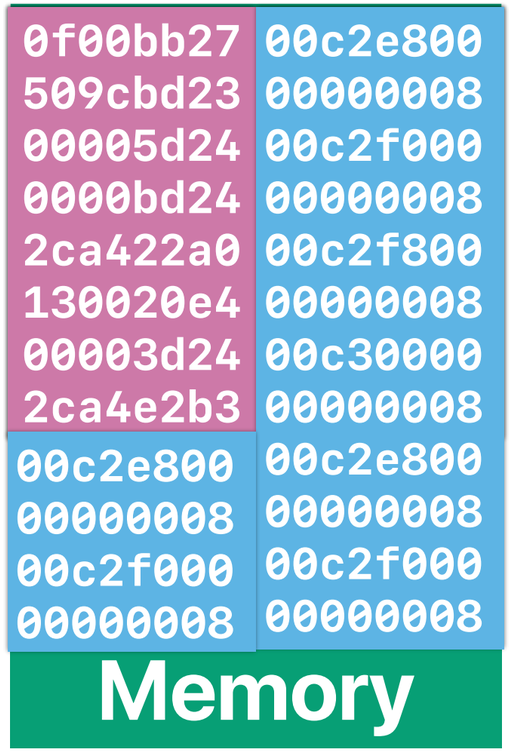
- Process — the abstraction of a **machine**
- Thread — the abstraction of a **processor**
- Virtual memory — the abstraction of the **memory**

Why: If we expose memory directly to the processor (I)

00c2f800
00000008
00c30000
00000008

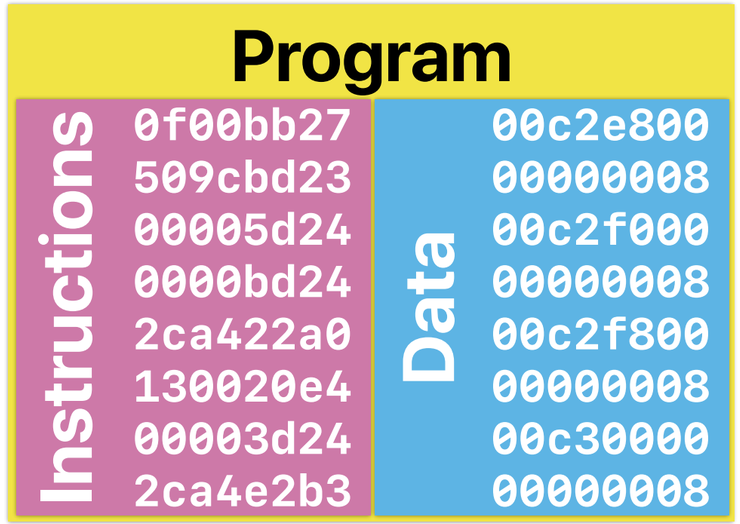
?

What if my program needs more memory?



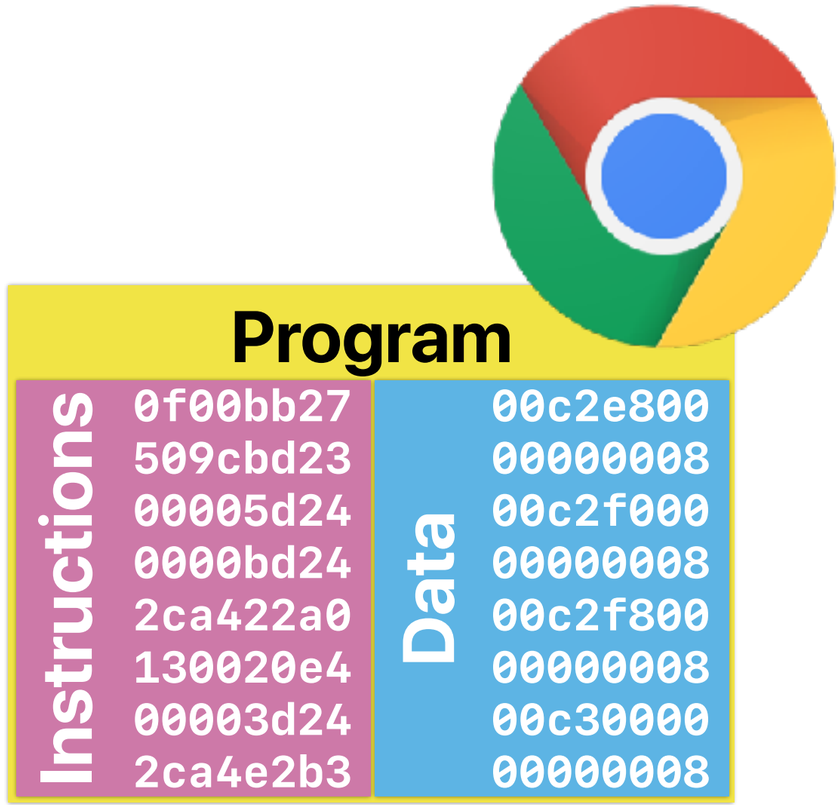
Why: If we expose memory directly to the processor (II)

What if my program runs on a machine with a different memory size?



Why: If we expose memory directly to the processor (III)

What if both programs need to use memory?



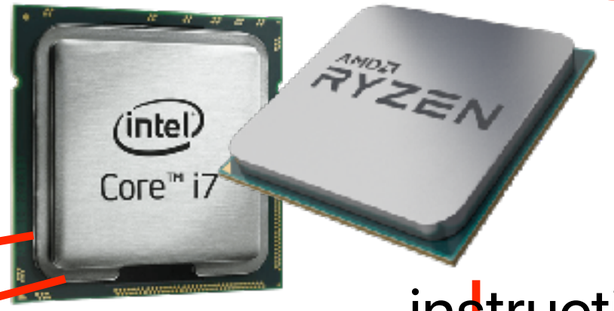
?

Outline

- The virtual memory abstraction
- Segmentation
- Demand paging
- Making demand paging efficient
- Swapping

The Virtual Memory Abstraction

Virtual memory



instruction
0x0

data
0x80000000

instruction
0x0

data
0x80008000

Virtual Memory Space

Virtual Memory Space

Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
0f00bb27	00c2f800
509cbd23	00000008
00005d24	00c30000
0000bd24	00000008

Memory

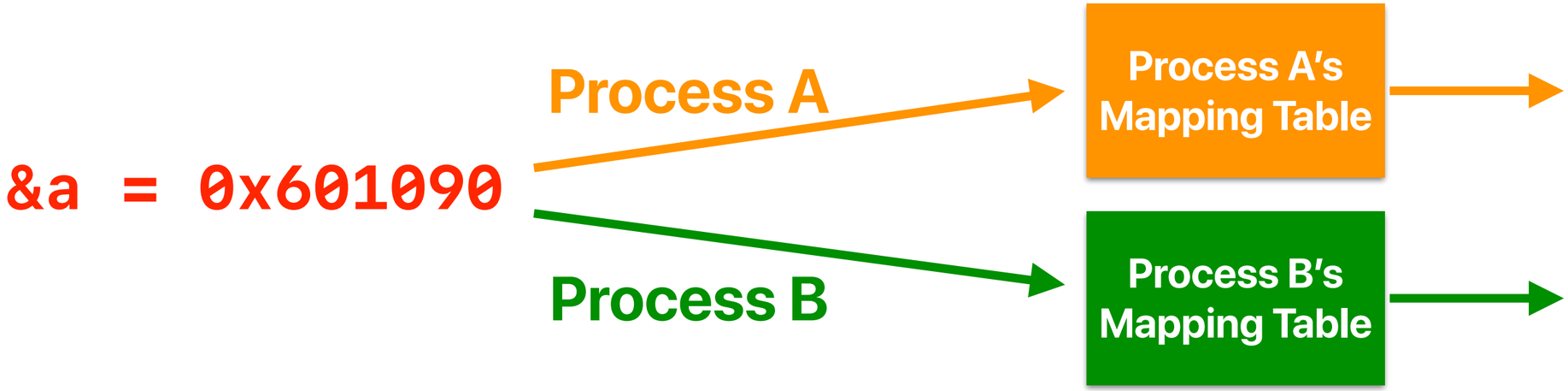
Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

Virtual memory

- An **abstraction** of memory space available for programs/software/programmer
- Programs execute using virtual memory address
- The operating system and hardware work together to handle the mapping between virtual memory addresses and real/physical memory addresses

Demo revisited



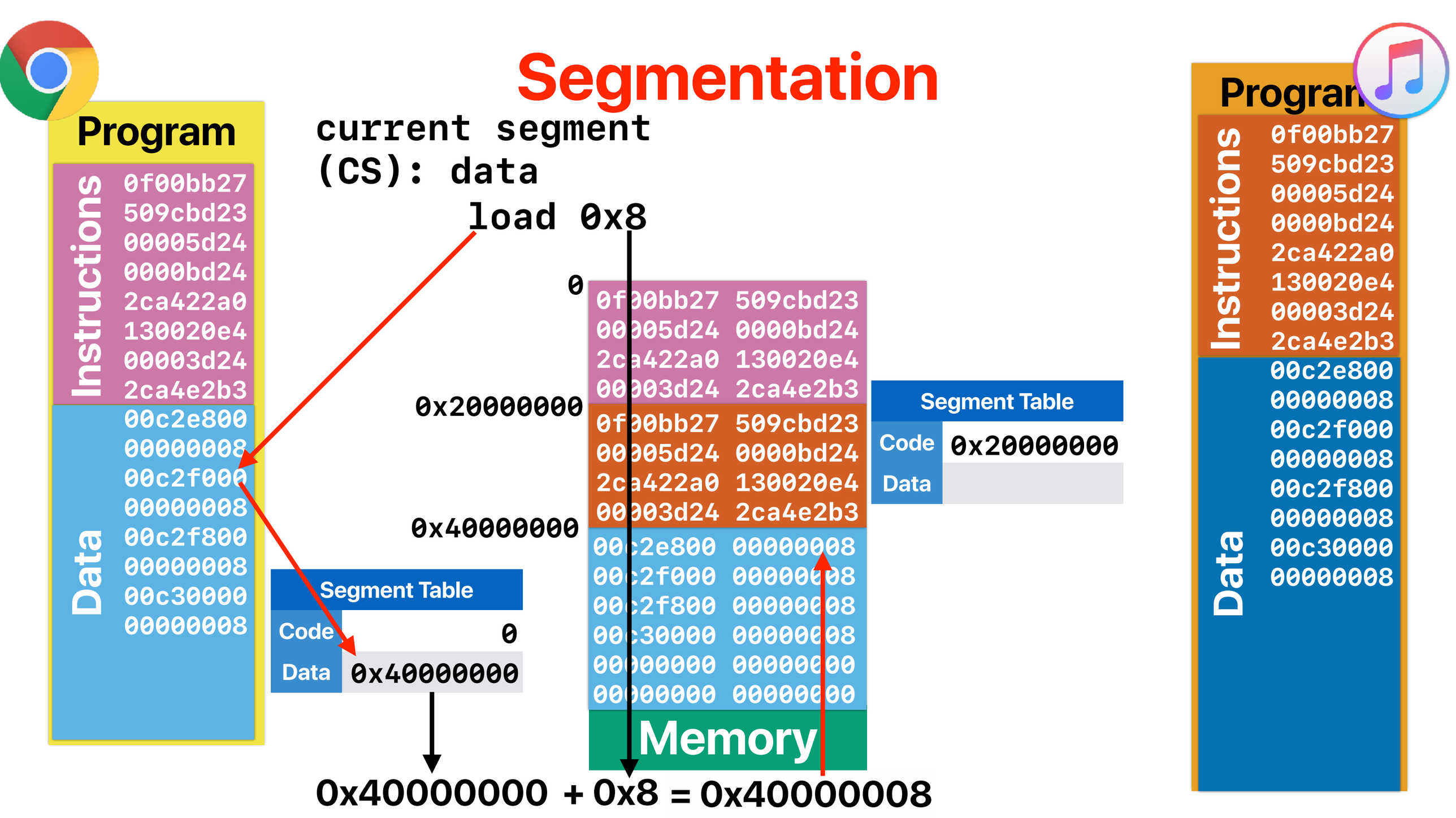

```c
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sched.h>
#include <sys/syscall.h>
#include <time.h>

double a;

int main(int argc, char *argv[])
{
    int i, number_of_total_processes=4;
    number_of_total_processes = atoi(argv[1]);
    for(i = 0; i< number_of_total_processes-1 && fork(); i++);
    srand((int)time(NULL)+(int)getpid());
    fprintf(stderr, "\nProcess %d is using CPU: %d. Value of a is %lf and address of a is %p\n",getpid(), cpu, a, &a);
    sleep(10);
    fprintf(stderr, "\nProcess %d is using CPU: %d. Value of a is %lf and address of a is %p\n",getpid(), cpu, a, &a);
    return 0;
}
```

How to map from virtual to physical? Let’s start from segmentation

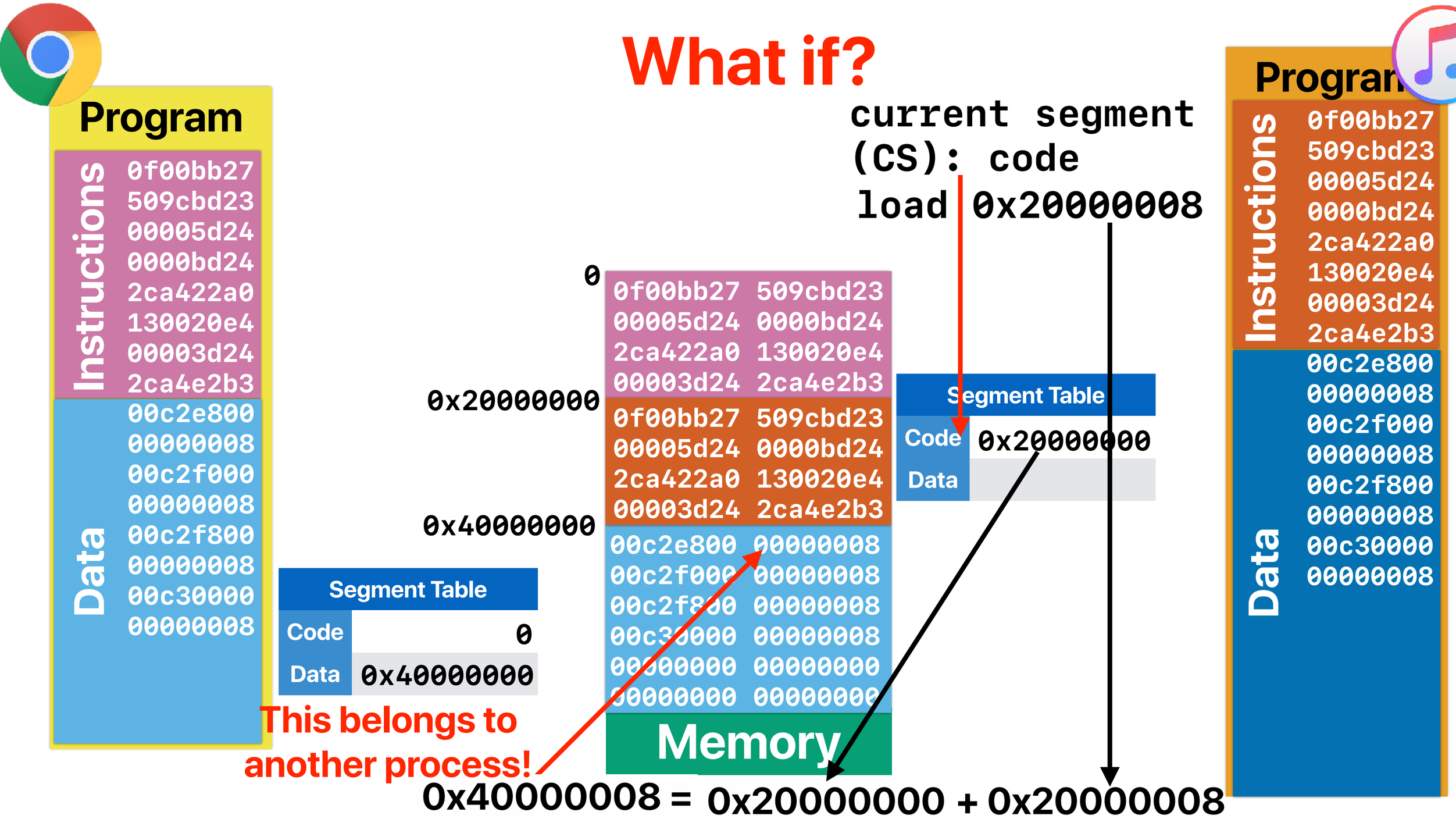
Segmentation

- The compiler generates code using virtual memory addresses
- The OS works together with hardware to partition physical memory space into segments for each running application
- The hardware dynamically translates virtual addresses into physical memory addresses

Segmentation



What if?


Program
Instructions
0f00bb27
509cbd23
00005d24
0000bd24
2ca422a0
130020e4
00003d24
2ca4e2b3
Data
00c2e800
00000008
00c2f000
00000008
00c2f800
00000008
00c30000
00000008
current segment (CS): code
load 0x20000008
0
0x20000000
0x40000000
0f00bb27 509cbd23
00005d24 0000bd24
2ca422a0 130020e4
00003d24 2ca4e2b3
0f00bb27 509cbd23
00005d24 0000bd24
2ca422a0 130020e4
00003d24 2ca4e2b3
00c2e800 00000008
00c2f000 00000008
00c2f800 00000008
00c30000 00000008
00000000 00000000
00000000 00000000
Memory
Segment Table
Code 0x20000000
Data
Segment Table
Code 0
Data 0x40000000
This belongs to another process!
0x40000008 = 0x20000000 + 0x20000008
Program
Instructions
0f00bb27
509cbd23
00005d24
0000bd24
2ca422a0
130020e4
00003d24
2ca4e2b3
Data
00c2e800
00000008
00c2f000
00000008
00c2f800
00000008
00c30000
00000008

What if?

```
current segment
(CS): code
load 0x20000008
```

is offset < bound?

Program

Instructions

```
0f00bb27
509cbd23
00005d24
0000bd24
```

Data

```
00000008
00c2f800
00000008
00c30000
00000008
```

a segmentation fault (often shortened to segfault), raised by hardware with memory protection, when the software has attempted to access a restricted area of memory (a memory access violation).

0

0x40000000

Segment Table		Bound
Code	0	0x20000000
Data	0x40000000	0x30000000

Memory

```
bb27 509cbd23
5d24 0000bd24
22a0 130020e4
3d24 2ca4e2b3
bb27 509cbd23
5d24 0000bd24
22a0 130020e4
00003d24 2ca4e2b3
00c2e800 00000008
00000008
00000008
00000008
00000000
00000000
```

Segment Table		Bound
Code	0x20000000	0x20000000
Data		0x40000000

Yes — proceed

No — segmentation fault!!!

Program

Instructions

```
0f00bb27
509cbd23
00005d24
0000bd24
2ca422a0
130020e4
00003d24
2ca4e2b3
```

```
00c2e800
0008
f000
0008
f800
00000008
00c30000
00000008
```

Data

Now, make sure you login to Poll Everywhere (through the App or the website) with UCRNetID

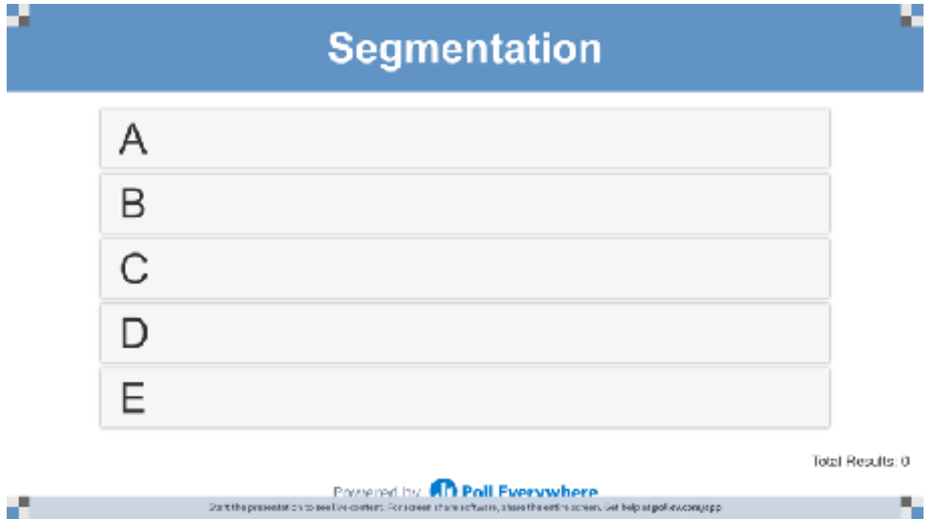
Now, you have about 90 seconds to answer the question!

https://www.pollev.com/hungweitseng close in 1:30

Efficiency of Segmentation

- Regarding segments, how many of the followings are correct?
  - ① Each segment must occupy contiguous physical memory locations
  - ② The system must allocate and reserve the physical memory locations for a segment whenever the program using that segment is scheduled
  - ③ An application can pre-allocate a large segment but turn out not using every byte in the segment
  - ④ The system may not be able to allocate space for a segment even though the total capacity of available physical locations is sufficient

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4



Now, it’s time to discuss with your surroundings (with masks on) — and make sure you vote again after the discussion!

Internal fragment

Program

Instructions	0f00bb27 509cbd23 00005d24 0000bd24 2ca422a0 130020e4 00003d24 2ca4e2b3
Data	00c2e800 00000008 00c2f000 00000008 00c2f800 00000008 00c30000 00000008

Segment Table		
Code	0	0x20000000
Data	0x40000000	0x30000000

Segment Table		
Code	0x20000000	0x20000000
Data		0x40000000

0	0f00bb27	509cbd23
	00005d24	0000bd24
	2ca422a0	130020e4
	00003d24	2ca4e2b3
0x20000000	0f00bb27	509cbd23
	00005d24	0000bd24
	2ca422a0	130020e4
	00003d24	2ca4e2b3
0x40000000	00c2e800	00000008
	00c2f000	00000008
	00c2f800	00000008
	00c30000	00000008
	00000000	00000000
	00000000	00000000

Memory

We waste some space in the allocated segment

Program

Instructions	0f00bb27 509cbd23 00005d24 0000bd24 2ca422a0 130020e4 00003d24 2ca4e2b3
Data	00c2e800 00000008 00c2f000 00000008 00c2f800 00000008 00c30000 00000008

External fragment

Program

Instructions	0f00bb27 509cbd23 00005d24 0000bd24 2ca422a0 130020e4 00003d24 2ca4e2b3
Data	00c2e800 00000008 00c2f000 00000008 00c2f800 00000008 00c30000 00000008

Segment Table	
Code	0x20000000
Data	0x30000000

Segment Table		
Code	0x20000000	0x20000000
Data		0x40000000

0	0f00bb27 509cbd23 00005d24 0000bd24 2ca422a0 130020e4 00003d24 2ca4e2b3
0x20000000	0f00bb27 509cbd23 00005d24 0000bd24 2ca422a0 130020e4 00003d24 2ca4e2b3
0x40000000	00c2e800 00000008 00c2f000 00000008 00c2f800 00000008 00c30000 00000008 00000000 00000000 00000000 00000000

Memory

0x40000000

Even though we have space, we still cannot map data section!

Program

Instructions	0f00bb27 509cbd23 00005d24 0000bd24 2ca422a0 130020e4 00003d24 2ca4e2b3
Data	00c2e800 00000008 00c2f000 00000008 00c2f800 00000008 00c30000 00000008

Paging

Paging

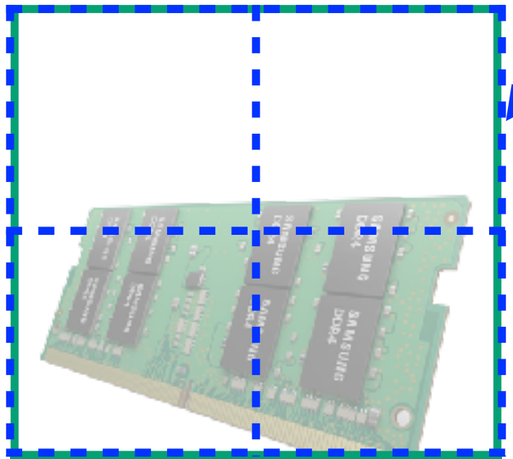
Virtual Address Space for Chrome

Virtual Address Space for Apple Music

each of this is a
fix-sized page

Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008



Memory

Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

Paging

- **Paging**: partition virtual/physical memory spaces into fix-sized pages

Page fault

instruction

0x0

Virtual Address Space for Chrome

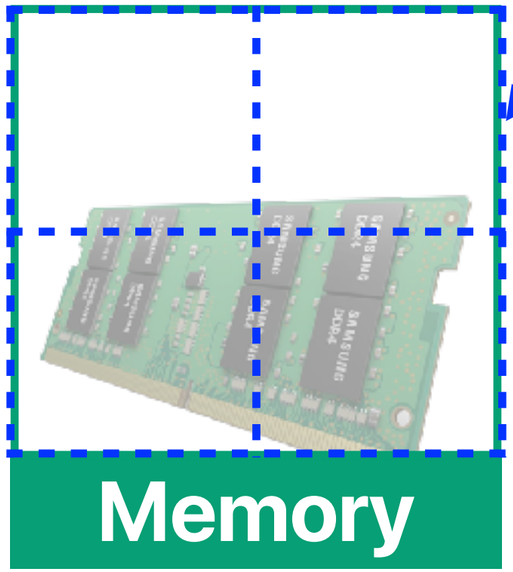
Page fault!

Virtual Address Space for Apple Music

each of this is a fix-sized page

Program

Instructions		Data	
	0f00bb27		00c2e800
	509cbd23		00000008
	00005d24		00c2f000
	0000bd24		00000008
	2ca422a0		00c2f800
	130020e4		00000008
	00003d24		00c30000
	2ca4e2b3		00000008



Program

Instructions		Data	
	0f00bb27		00c2e800
	509cbd23		00000008
	00005d24		00c2f000
	0000bd24		00000008
	2ca422a0		00c2f800
	130020e4		00000008
	00003d24		00c30000
	2ca4e2b3		00000008

Page fault

- **Paging**: partition virtual/physical memory spaces into fix-sized pages
- **Page fault**: when the requested page cannot be found in the physical memory — created the demand of allocating pages!

Demand paging

instruction
0x0

data
0x80000000

instruction
0x0

data
0x80008000

Page fault!

Virtual Address Space for Chrome

Page fault!

Page fault!

Virtual Address Space for Apple Music

Page fault!

each of this is a
fix-sized page

Program

Instructions		Data	
	0f00bb27		00c2e800
	509cbd23		00000008
	00005d24		00c2f000
	0000bd24		00000008
	2ca422a0		00c2f800
	130020e4		00000008
	00003d24		00c30000
	2ca4e2b3		00000008

Memory	
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
0f00bb27	00c2f800
509cbd23	00000008
00005d24	00c30000
0000bd24	00000008

Program

Instructions		Data	
	0f00bb27		00c2e800
	509cbd23		00000008
	00005d24		00c2f000
	0000bd24		00000008
	2ca422a0		00c2f800
	130020e4		00000008
	00003d24		00c30000
	2ca4e2b3		00000008

Demand paging

- **Paging**: partition virtual/physical memory spaces into fix-sized pages
- **Page fault**: when the requested page cannot be found in the physical memory — created the demand of allocating pages!
- **Demand paging**: Allocate a physical memory page for a virtual memory page when the virtual page is needed (page fault occurs)
  - There is also **shadow paging** used by embedded systems, mobile phones — they load the whole program/data into the physical memory when you launch it

https://www.pollev.com/hungweitseng close in 1:30

Segmentation v.s. demand paging

- How many of the following statements is/are correct regarding segmentation and demand paging?
  - ① Segments can cause more external fragmentations than demand paging
  - ② Paging can still cause internal fragmentations
  - ③ The overhead of address translation in segmentation is higher
  - ④ Consecutive virtual memory address may not be consecutive in physical address if we use demand paging

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Segmentation v.s. demand paging

- How many of the following statements is/are correct regarding segmentation and demand paging?
  - ① Segments can cause more external fragmentations than demand paging — the main reason why we love paging!
  - ② Paging can still cause internal fragmentations — within a page
  - ③ The overhead of address translation in segmentation is higher — you need to provide finer-grained mapping in paging — you may need to handle page faults!
  - ④ Consecutive virtual memory address may not be consecutive in physical address if we use demand paging

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

We haven't seen pure/true implementation of segmentations for a while, but we still use segmentation fault errors all the time!

Address translation in demand paging

Demand paging

instruction
0x0

data
0x80000000

Page fault!

Virtual Address Space for Chrome

Page fault!

instruction
0x0

Page fault!

Virtual Address Space for Apple Music

data
0x80008000

Page fault!

each of this is a
fix-sized page

Table!!!

Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
0f00bb27	00c2f800
509cbd23	00000008
00005d24	00c30000
0000bd24	00000008

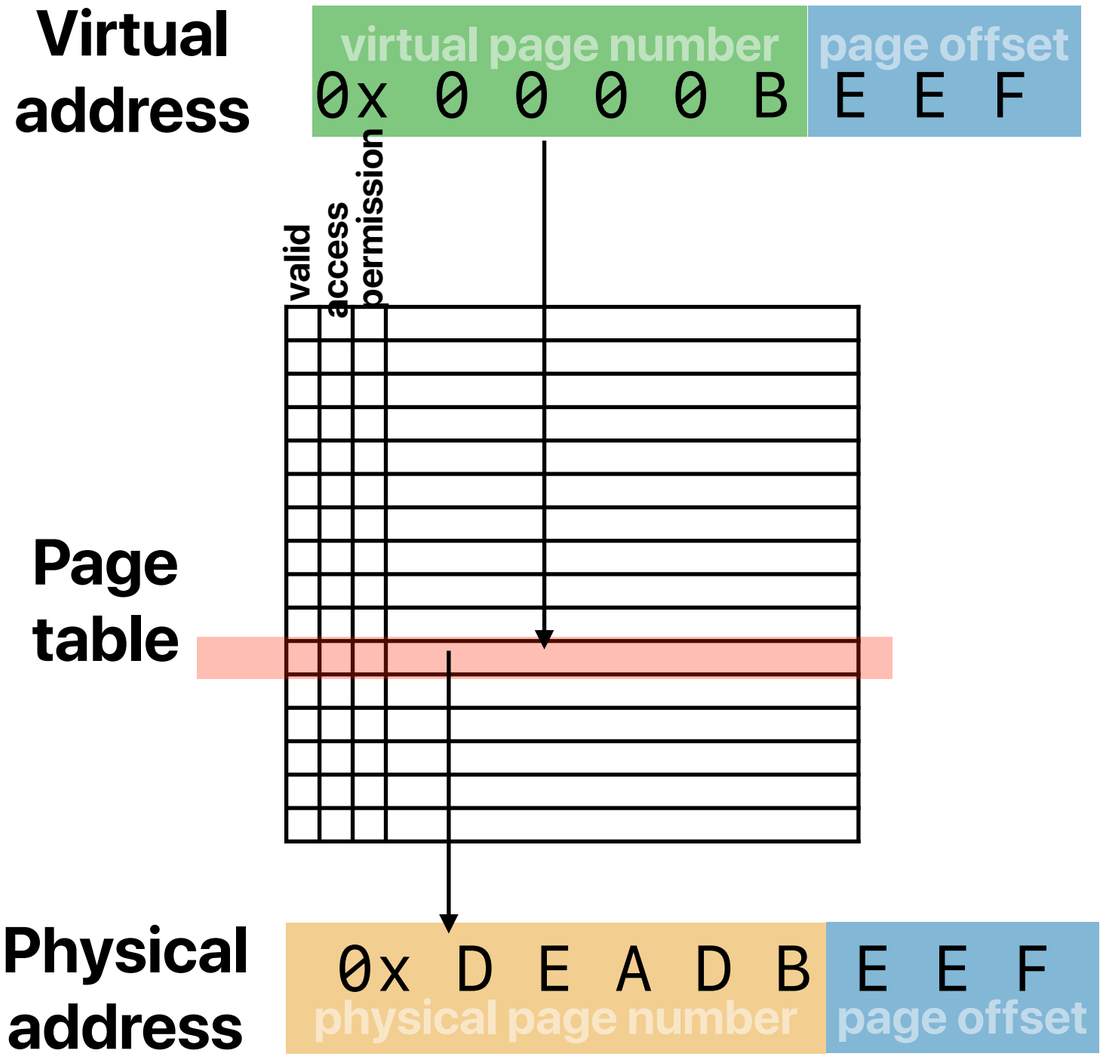
Memory

Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

Address translation

- Processor receives virtual addresses from the running code, main memory uses physical memory addresses
- Virtual address space is organized into "pages"
- The system references the **page table** to translate addresses
  - Each process has its own page table
  - The page table content is maintained by OS
- In addition to valid bit and physical page #, the page table may also store
  - Reference bit
  - Modified bit
  - Permissions



Size of page table

- Assume that we have 32-bit virtual address space, each page is 4KB, each page table entry is 4 bytes, how big is the page table for a process?
  A. 1MB
  B. 2MB
  C. 4MB
  D. 8MB
  E. 16MB

Size of page table

- Assume that we have 32-bit virtual address space, each page is 4KB, each page table entry is 4 bytes, how big is the page table for a process?

A. 1MB
B. 2MB
C. 4MB
D. 8MB
E. 16MB

The size of each entry in the page table

Number of entries in the page table

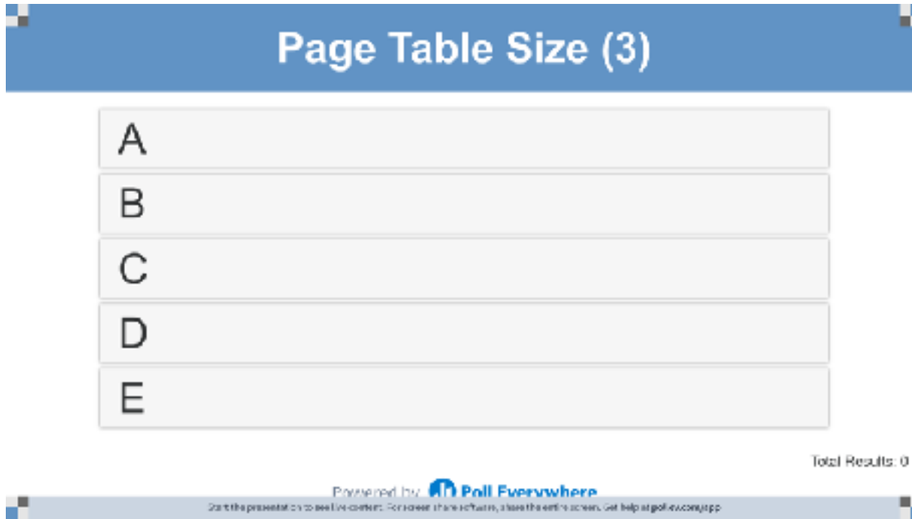
$$4\ bytes \times \frac{4\ GB}{4\ KB} = 4 \times 1\ M = 4\ MB$$

What if we have 16 processes?

4 MB * 16 = 64MB
— we need a separate page table for each process

Size of page table

- Assume that we have **64-bit** virtual address space, each page is 4KB, each page table entry is 8 bytes (64-bit addresses), what magnitude in size is the page table for 32 processes?
  A. MB — 2^{20} Bytes
  B. GB — 2^{30} Bytes
  C. TB — 2^{40} Bytes
  D. PB — 2^{50} Bytes
  E. EB — 2^{60} Bytes



Size of page table

- Assume that we have **64-bit** virtual address space, each page is 4KB, each page table entry is 8 bytes (64-bit addresses), what magnitude in size is the page table for 32 processes?
  A. MB — 2^{20} Bytes
  B. GB — 2^{30} Bytes
  C. TB — 2^{40} Bytes
  D. PB — 2^{50} Bytes
  E. EB — 2^{60} Bytes

$$8\ bytes \times \frac{2^{64}\ B}{4\ KB} = 2^3 B \times \frac{2^{64}\ B}{2^{12}\ B} = 2^{55}\ B = 32\ PB$$

$$32\ PB \times 32 = 2^{60} B = 1\ EB$$

Address translation (cont.)

- Page tables are too large to be kept on the chip (millions of entries)
  — space overhead: surpasses cache capacity
- Instead, the page tables are kept in memory
  — memory access overhead
  — space overhead: can be bigger than physical main memory when address space is large

Smaller page tables

Conventional page table

0x0 0xFFFFFFFFFFFFFFFF

Virtual Address Space

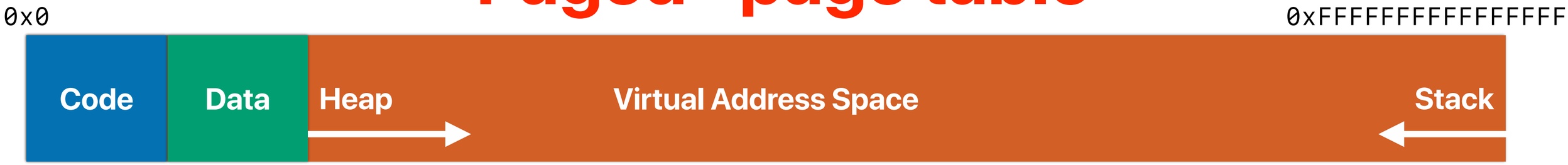
— must be consecutive in the physical memory

— like a big segment! — difficult to find a spot

— simply too big to fit in memory if address space is large!

$\frac{2^{64}\ B}{2^{12}\ B}$ page table entries/leaf nodes

"Paged" page table

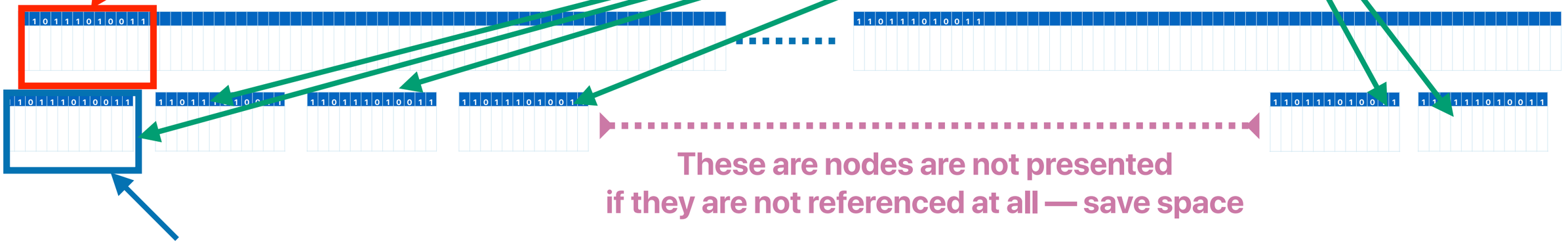


Break up entries into pages!
Each of these occupies exactly a page

$$\frac{2^{12}\ B}{2^{3}\ B} = 2^{9}$$ **PTEs per node**

Question:
These nodes are spread out, how to locate them in the memory?

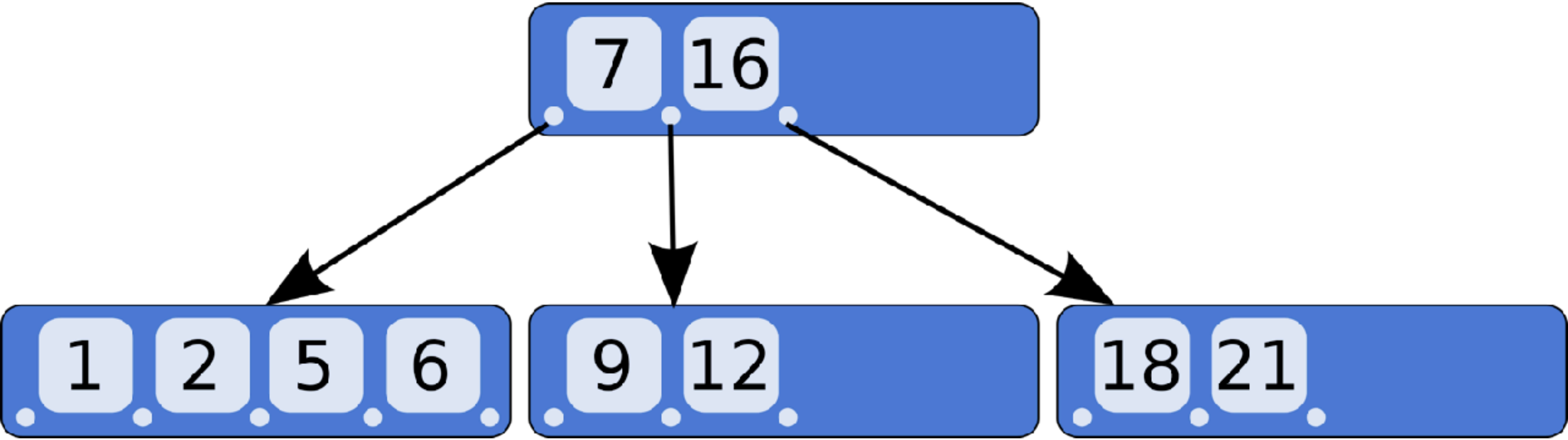
Otherwise, you always need to find more than one consecutive pages — difficult!



These are nodes are not presented if they are not referenced at all — save space

Allocate page table entry nodes "on demand"

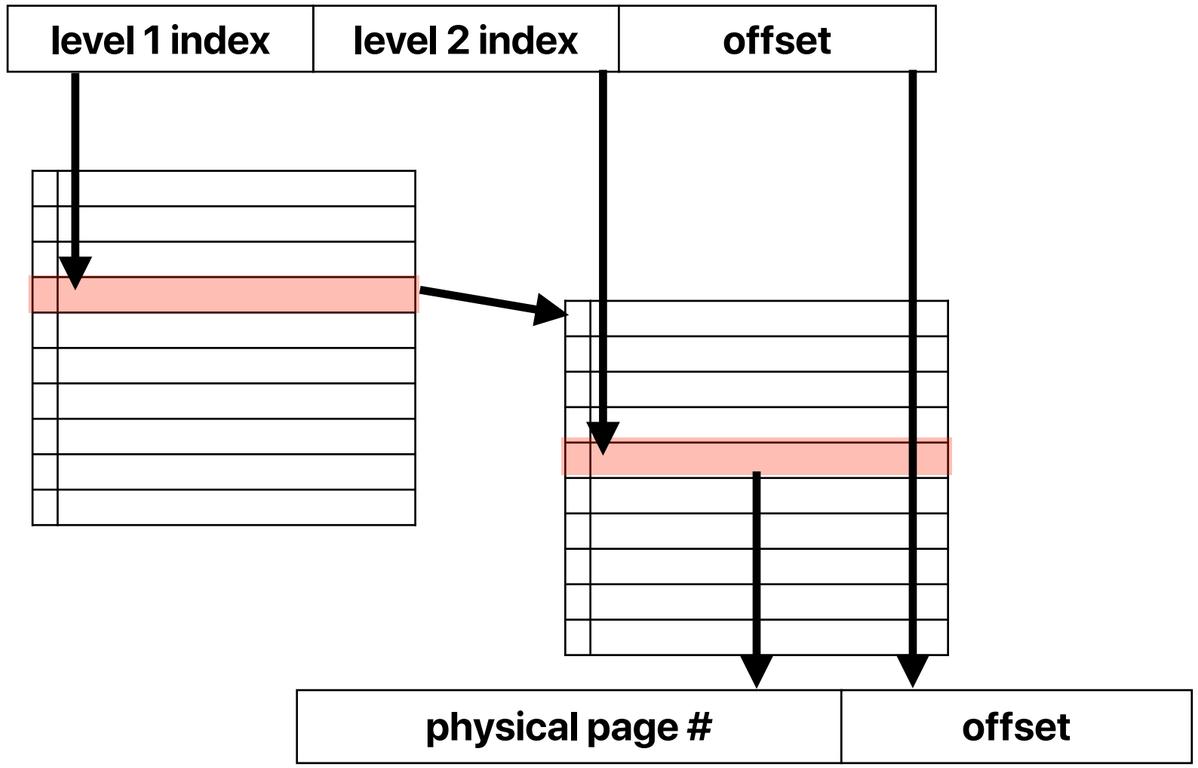
B-tree



https://en.wikipedia.org/wiki/B-tree#/media/File:B-tree.svg

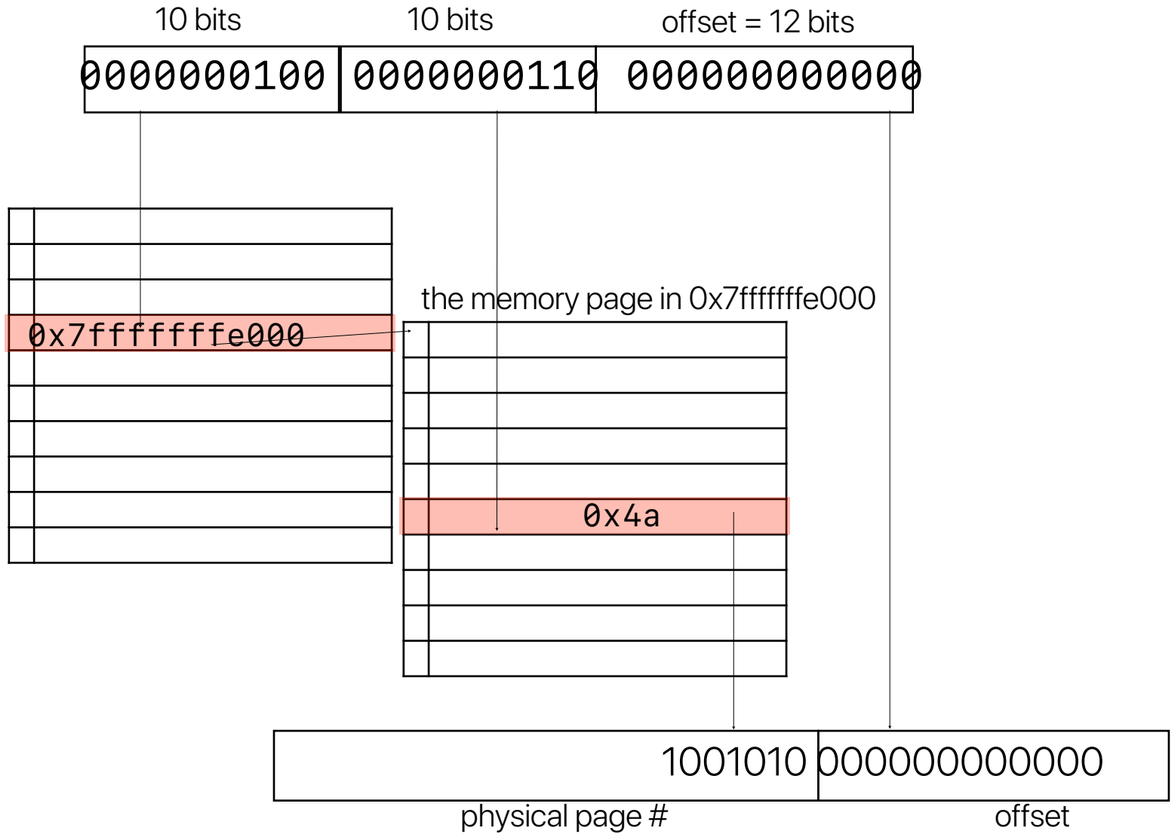
Hierarchical page table

- Break the virtual page number into several pieces
- If one piece has N bits, build an 2^N-ary tree
- Only store the part of the tree that contain valid pages
- Walk down the tree to translate the virtual address

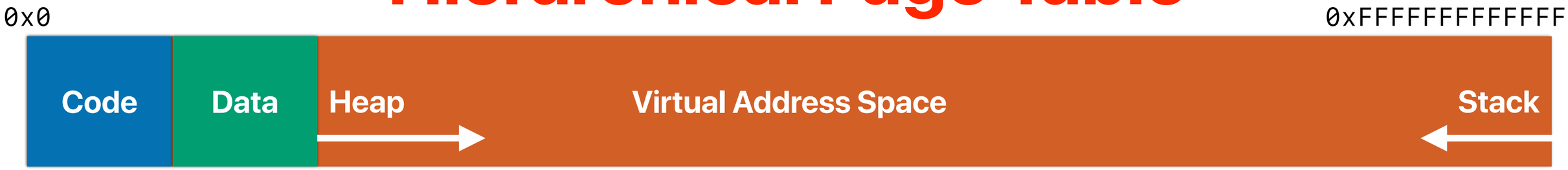


Page table walking example

- Two-level, 4KB, 10 bits index in each level
- If we are accessing 0x1006000 now...



Hierarchical Page Table



$$\lceil log_{2^9} \frac{2^{64}\ B}{2^{12}\ B} \rceil = \lceil log_{2^9} 2^{52} \rceil = 6 \text{ levels}$$

These are nodes are not presented as they are not referenced at all.

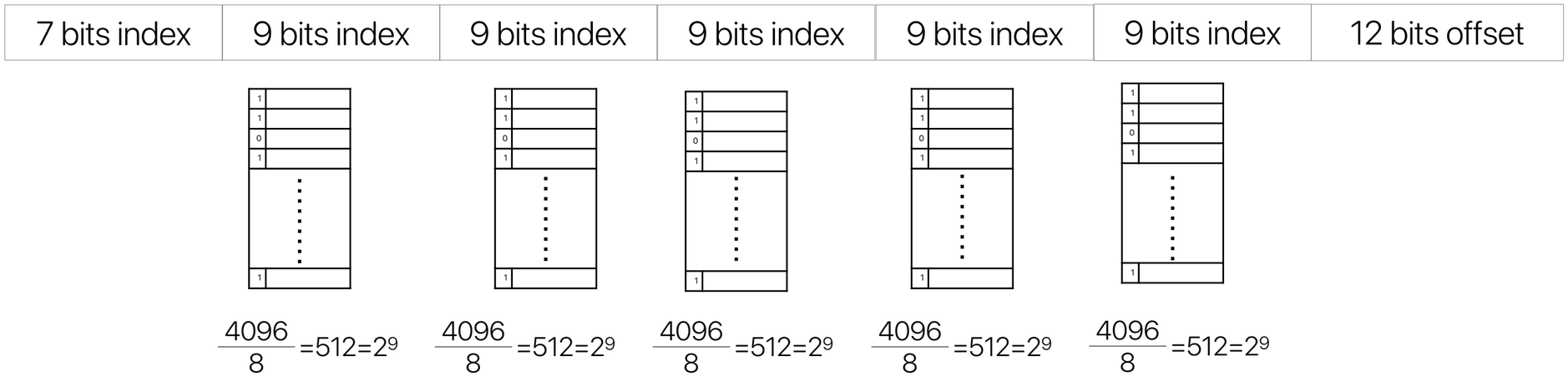
$\frac{2^{64}\ B}{2^{12}\ B}$ page table entries/leaf nodes (worst case)

How many levels do we need?

- Assume that our system uses hierarchical page table with 4KB page size under 64-bit virtual address space and each PTE is 8B in size. How many levels of indexes do we need for the hierarchical page table?
  A. 2
  B. 3
  C. 4
  D. 5
  E. 6

How many levels do we need?

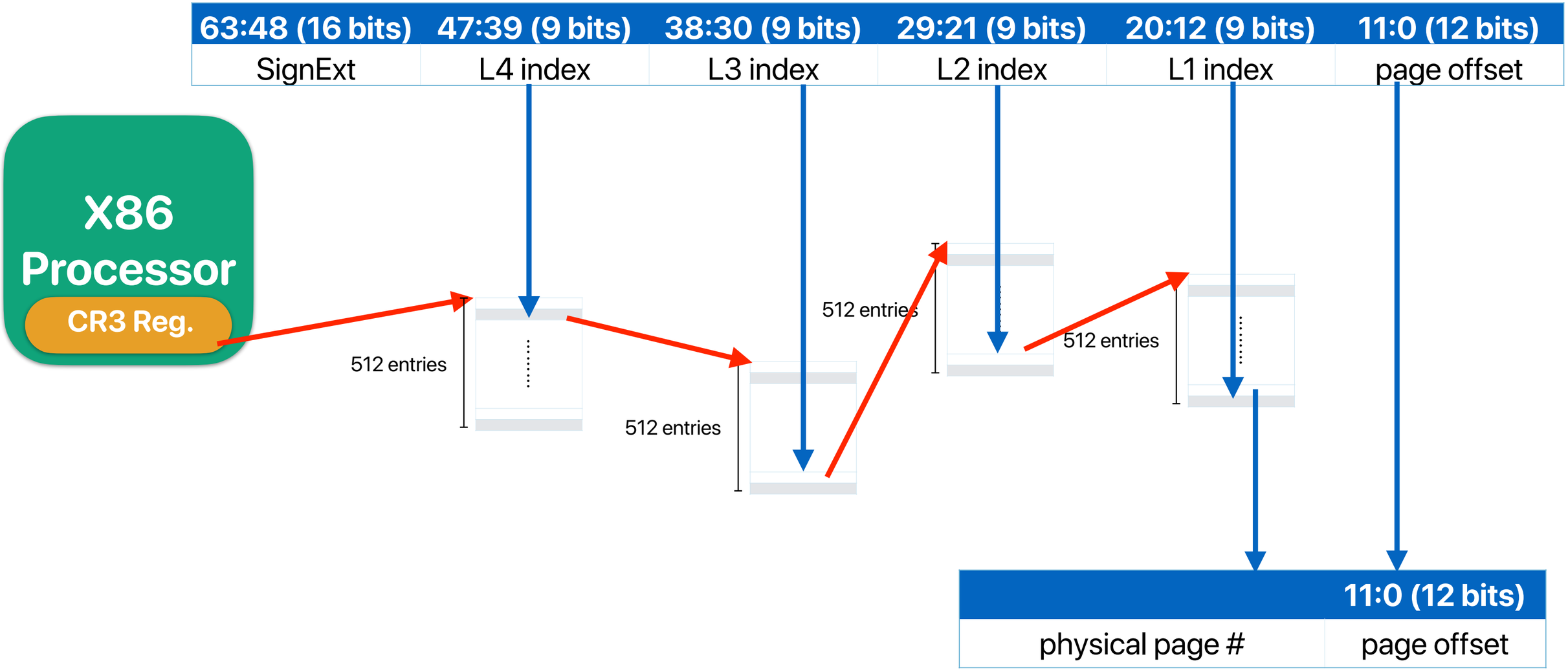
- Assume that our system uses hierarchical page table with 4KB page size under 64-bit virtual address space and each PTE is 8B in size. How many levels of indexes do we need for the hierarchical page table?



How many levels do we need?

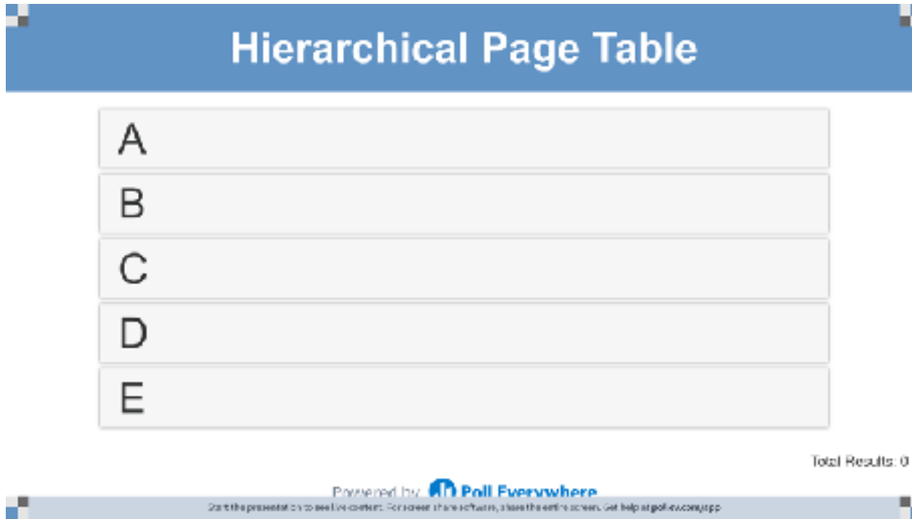
- Assume that our system uses hierarchical page table with 4KB page size under 64-bit virtual address space and each PTE is 8B in size. How many levels do we need for the hierarchical page table?
  - A. 2
  - B. 3
  - C. 4
  - D. 5
  - E. 6

Case study: Address translation in x86-64



Pros and cons of hierarchical page table

- Regarding hierarchical page tables, how many of the following statements is/ are correct?
  - ① Hierarchical page tables can reduce the size of page tables
  - ② Hierarchical page tables can reduce the memory accesses during address translations
  - ③ Hierarchical page tables can reduce the overhead of page fault handling
  - ④ For user-space threads belonging to the same process, these threads will share a hierarchical page table
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4



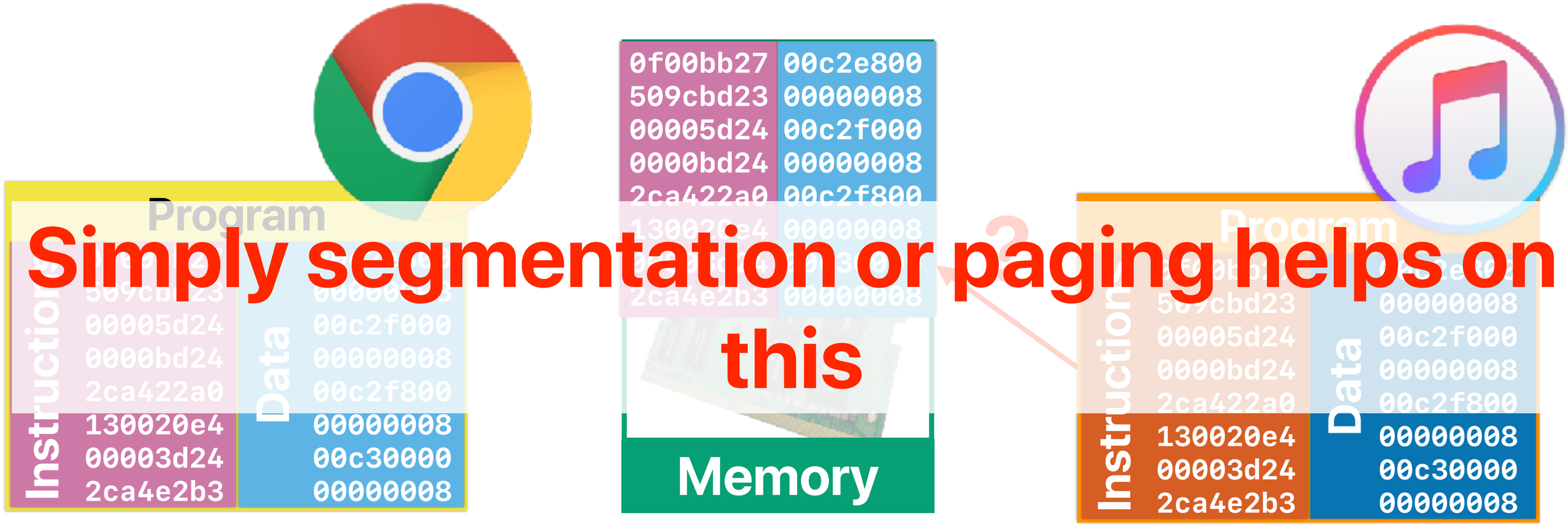
Pros and cons of hierarchical page table

- Regarding hierarchical page tables, how many of the following statements is/are correct?
  - ① Hierarchical page tables can reduce the size of page tables
  - ② Hierarchical page tables can reduce the memory accesses during address translations — create more accesses
  - ③ Hierarchical page tables can reduce the overhead of page fault handling — actually more, as you may need to update the whole tree
  - ④ For user-space threads belonging to the same process, these threads will share a hierarchical page table

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

If we expose memory directly to the processor (III)

What if both programs need to use memory?



Simply segmentation or paging helps on this

If we expose memory directly to the processor (I)

00c2f800
00000008
00c30000
00000008

?

What if my program needs more memory?

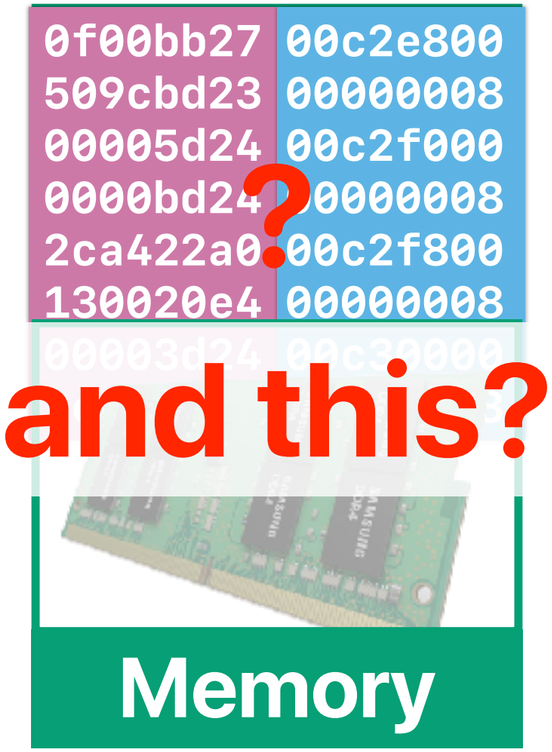
But how about this?

If we expose memory directly to the processor (II)

What if my program runs on a machine with a different memory size?

Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008



Announcement

- Reading quizzes due next Tuesday
- Hung-Wei's office hour shuffle this week (will be back to normal next week)
  - **W 10a-12p**
  - Use the office hour Zoom link, not the lecture one
- Project released
  - Groups in 2 (3 **at most**)
  - **Pull the latest version — had some changes for later kernel versions**
    https://github.com/hungweitseng/CS202-MMA
  - **Install an Ubuntu Linux 20.04 VM as soon as you can!**
  - **Please do not use a real machine — you may not be able to reboot again**
- Midterm
  - Will release on 2/10/2021 0:00am and due on 2/11/2021 11:59:00pm
  - You will have to find a consecutive, non-stop 80-minute slot with this period
  - One time, cannot reinitiate — please make sure you have a stable system and network. We cannot provide service during your exam and we are not obligated to.
  - **No late submission is allowed**

Computer Science & Engineering

202

つづく

新年快樂！虎年虎運！