Virtual Memory

Hung-Wei Tseng

Recap: Let’s dig into this code

```
int main(int argc, char *argv[])
{
    int i,j;
    double **a;
    double sum=0, average;
    int dim=32768;
    if(argc < 2)
    {
        fprintf(stderr, "Usage: %s dimension\n",argv[0]);
        exit(1);
    }
    dim = atoi(argv[1]);
    a = (double **)malloc(sizeof(double *)*dim);
    for(i = 0 ; i < dim; i++)
        a[i] = (double *)malloc(sizeof(double)*dim);
    for(i = 0 ; i < dim; i++)
        for(j = 0 ; j < dim; j++)
            a[i][j] = rand();
    for(i = 0 ; i < dim; i++)
        for(j = 0 ; j < dim; j++)
            sum+=a[i][j];
    average = sum/(dim*dim);
    fprintf(stderr,"average: %lf\n",average);
    for(i = 0 ; i < dim; i++)
        free(a[i]);
    free(a);
    return 0;
}
```

```
#include <stdlib.h>
#include <assert.h>
#include <sched.h>
#include <sys/syscall.h>
#include <time.h>

//#define dim 32768
//#define dim 49152
int main(int argc, char *argv[])
{
    int i,j;
    double **a;
    double sum=0, average;
    int dim=32768;
    if(argc < 2)
    {
        fprintf(stderr, "Usage: %s dimension\n",argv[0]);
        exit(1);
    }
    dim = atoi(argv[1]);
    a = (double **)malloc(sizeof(double *)*dim);
File memory_allocation.c not changed so no update needed
BunnyMACProRetina [/Users/bunny/Dropbox/CSE203/GitHub/demo/virtual_memory] -bunny- ./memory_allocation

BunnyMACProRetina [/Users/bunny/Dropbox/CSE203/GitHub/demo/virtual_memory] -bunny-
```

Recap: Let’s dig into this code

```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]);
    // Create processes
    for(i = 0; i< number_of_total_processes-1 && fork(); i++);
    // Generate rand seed
    srand((int)time(NULL)+(int)getpid());
    a = rand();
    fprintf(stderr, "\nProcess %d. Value of a is %lf and address of a is %p\n”,getpid(), a, &a);
    sleep(10);
    fprintf(stderr, "\nProcess %d. Value of a is %lf and address of a is %p\n”,getpid(), a, &a);
    return 0;
}
```

Demo revisited

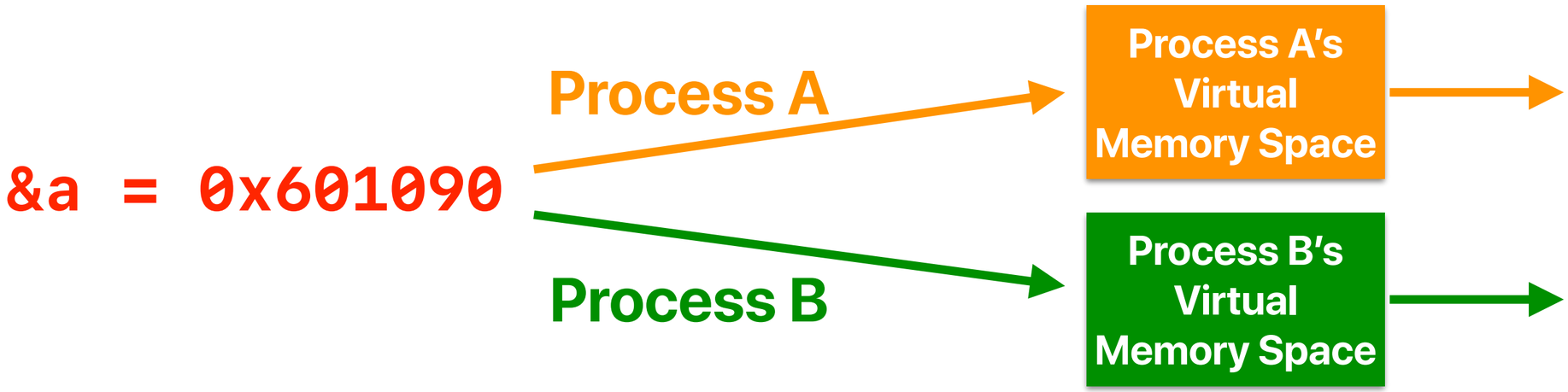
```
    sleep(10);
    fprintf(stderr, "\nProcess %d: Value of a is %lf and address of a is %p\n",(int)getpid(), a, &a);
    return 0;
}
File virtualization.c not changed so no update needed
BunnyMACProRetina [/Users/bunny/Dropbox/CSE203/GitHub/demo/virtual_memory] -bunny- make
gcc -O3 virtualization.c -o virtualization
BunnyMACProRetina [/Users/bunny/Dropbox/CSE203/GitHub/demo/virtual_memory] -bunny- ./virtualization 4

Process 19719: Value of a is 1671139616.000000 and address of a is 0x104967050

Process 19720: Value of a is 1671156423.000000 and address of a is 0x104967050

Process 19718: Value of a is 1671122809.000000 and address of a is 0x104967050

Process 19721: Value of a is 1671173230.000000 and address of a is 0x104967050

Process 19719: Value of a is 1671139616.000000 and address of a is 0x104967050

Process 19721: Value of a is 1671173230.000000 and address of a is 0x104967050

Process 19720: Value of a is 1671156423.000000 and address of a is 0x104967050

Process 19718: Value of a is 1671122809.000000 and address of a is 0x104967050
BunnyMACProRetina [/Users/bunny/Dropbox/CSE203/GitHub/demo/virtual_memory] -bunny-
```

Different values

Different values are preserved

The same memory address!

Recap: Demo revisited



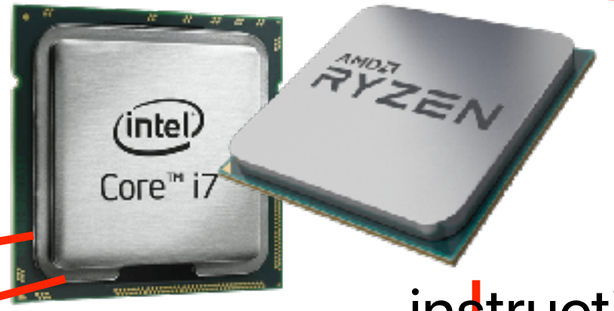

```c
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sched.h>
#include <sys/syscall.h>
#include <time.h>

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. Value of a is %lf and address of a is %p\n",getpid(), a, &a);
    sleep(10);
    fprintf(stderr, "\nProcess %d. Value of a is %lf and address of a is %p\n",getpid(), a, &a);
    return 0;
}
```

Outline

- Virtual memory
- Architectural support for virtual memory

Virtual Memory

Virtual memory



instruction
0x0

data
0x80000000

0x0

$2^{64}-1$

Virtual Memory Space

instruction
0x0

data
0x80008000

$2^{64}-1$

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

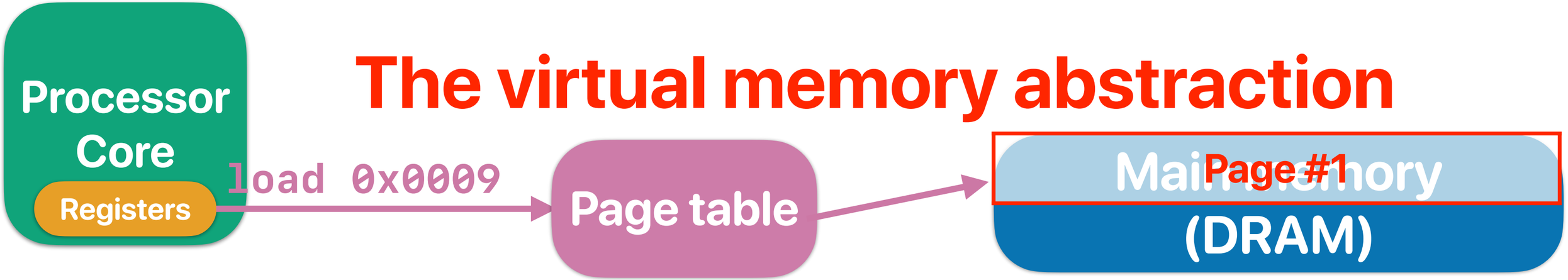
Program

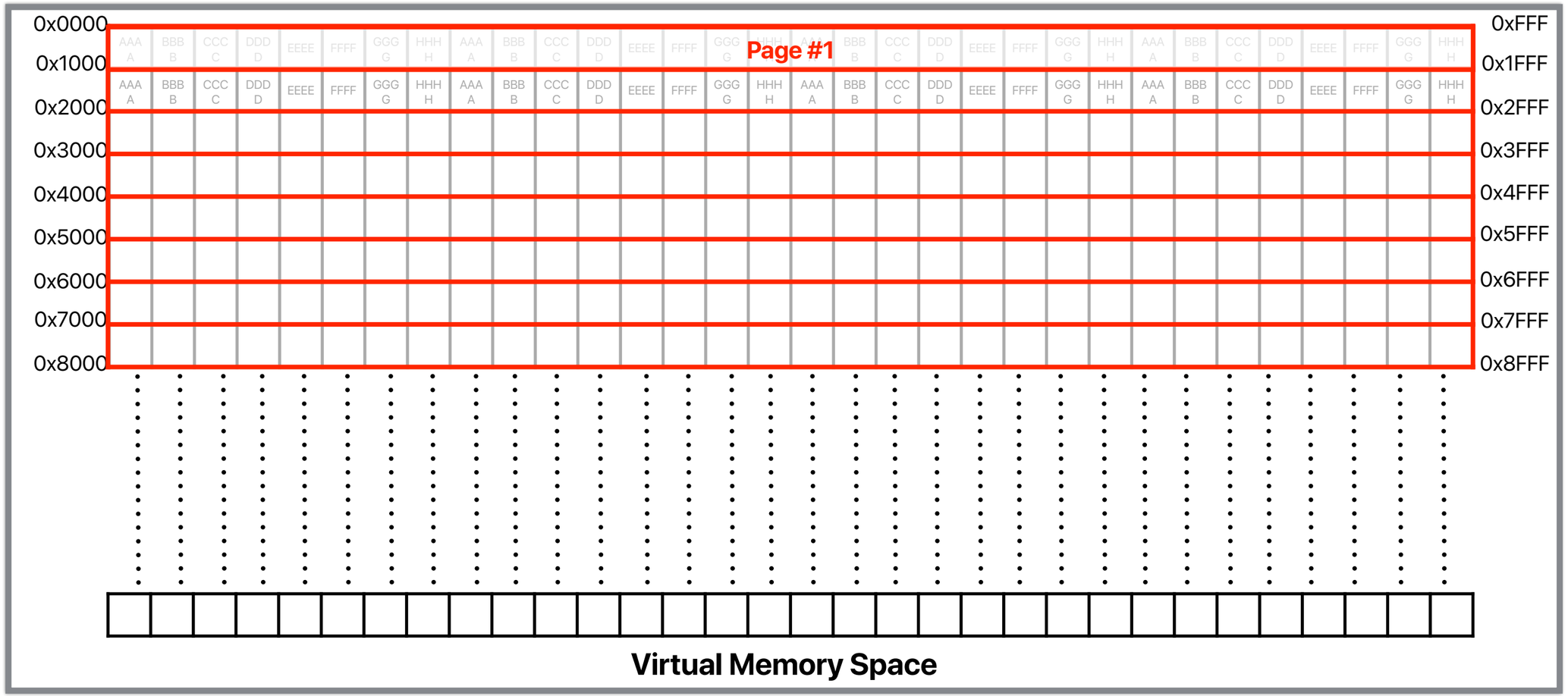
Instructions		Data	
0f00bb27		00c2e800	
509cbd23		00000008	
00005d24		00c2f000	
0000bd24		00000008	
2ca422a0		00c2f800	
130020e4		00000008	
00003d24		00c30000	
2ca4e2b3		00000008	

Virtual memory

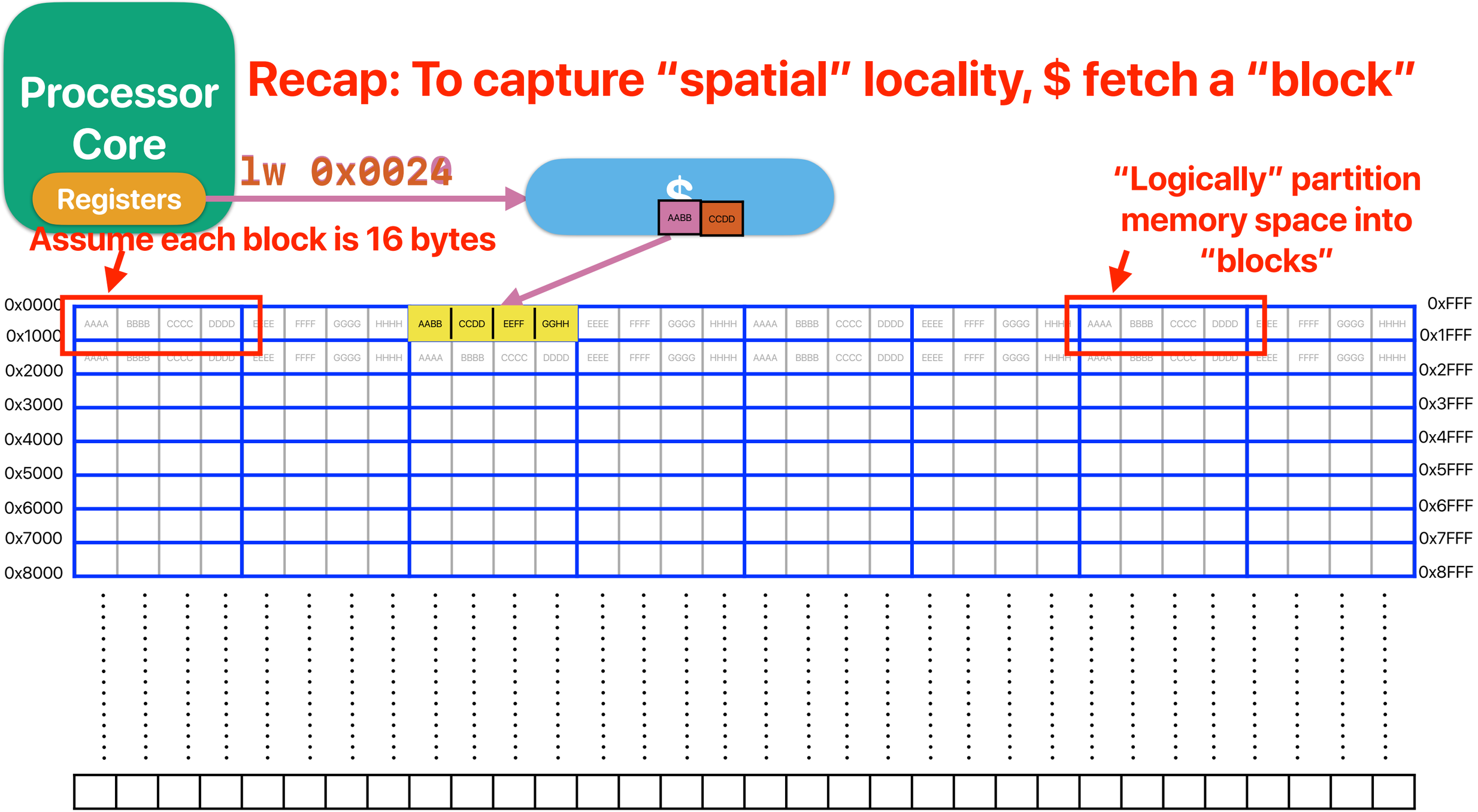
- An **abstraction** of memory space available for programs/software/programmer
- Programs execute using virtual memory address
- The operating system and hardware work together to handle the mapping between virtual memory addresses and real/physical memory addresses
- Virtual memory organizes memory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

The virtual memory abstraction





Recap: To capture "spatial" locality, $ fetch a "block"



Why Virtual memory?

- Allowing multiple applications to share physical main memory
  - Memory protection/isolation among programs/processes is automatically achieved
- Allowing applications to work even the installed physical memory or available physical memory is smaller than the working set of the application
  - Programmer does not need to worry about the physical memory capacity of different machines — make compiled program compatible
  - Multiple programs can work concurrently even through their total memory demand is larger than the installed physical memory

Demand paging

- Treating physical main memory as a “cache” of virtual memory
- The block size is the “page size”
- The page table is the “tag array”
- It’s a “fully-associate” cache — a virtual page can go anywhere in the physical main memory

Demand paging

instruction
0x0

data
0x80000000

data
0x80008000

instruction
0x0

data
0x80008000

Page Table for Chrome

Page Table for Apple Music

Program

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

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

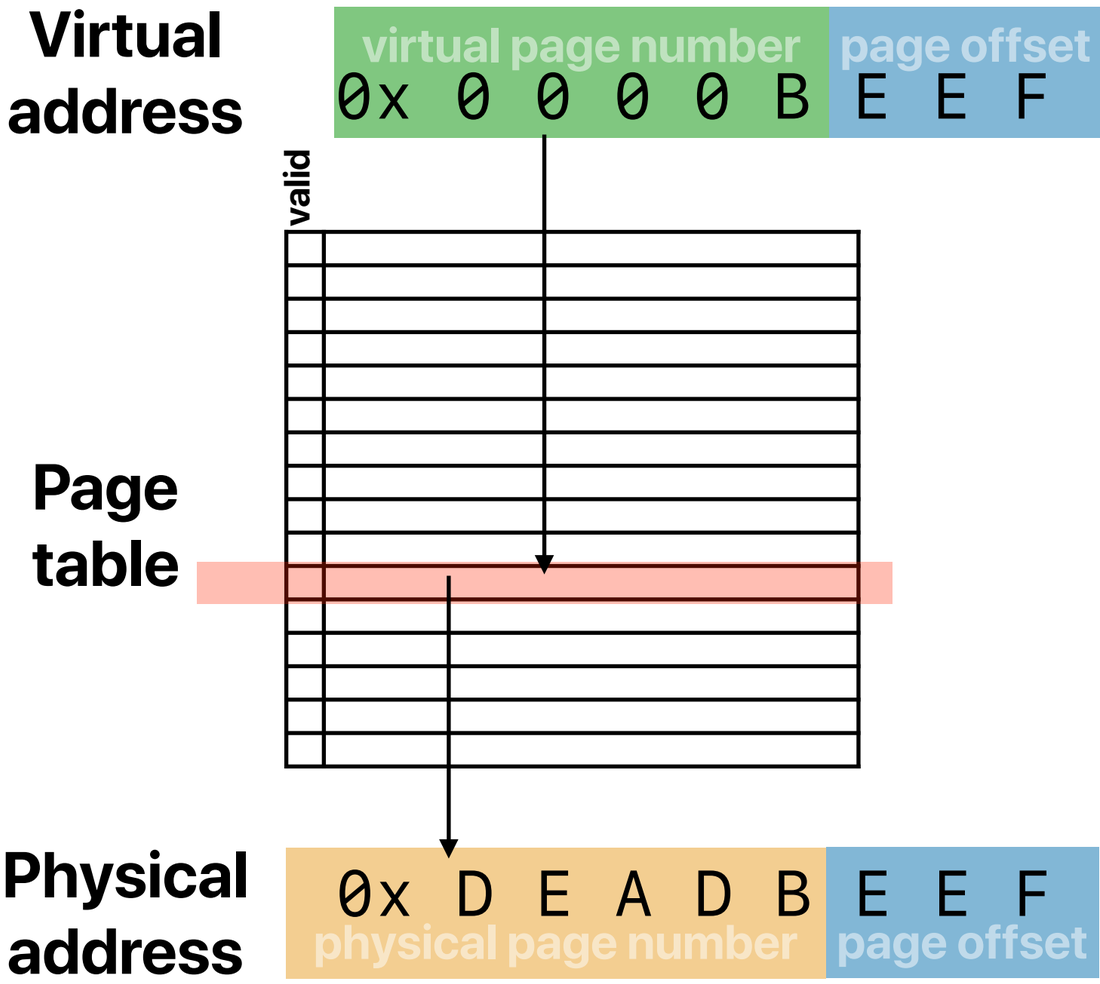
Memory

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

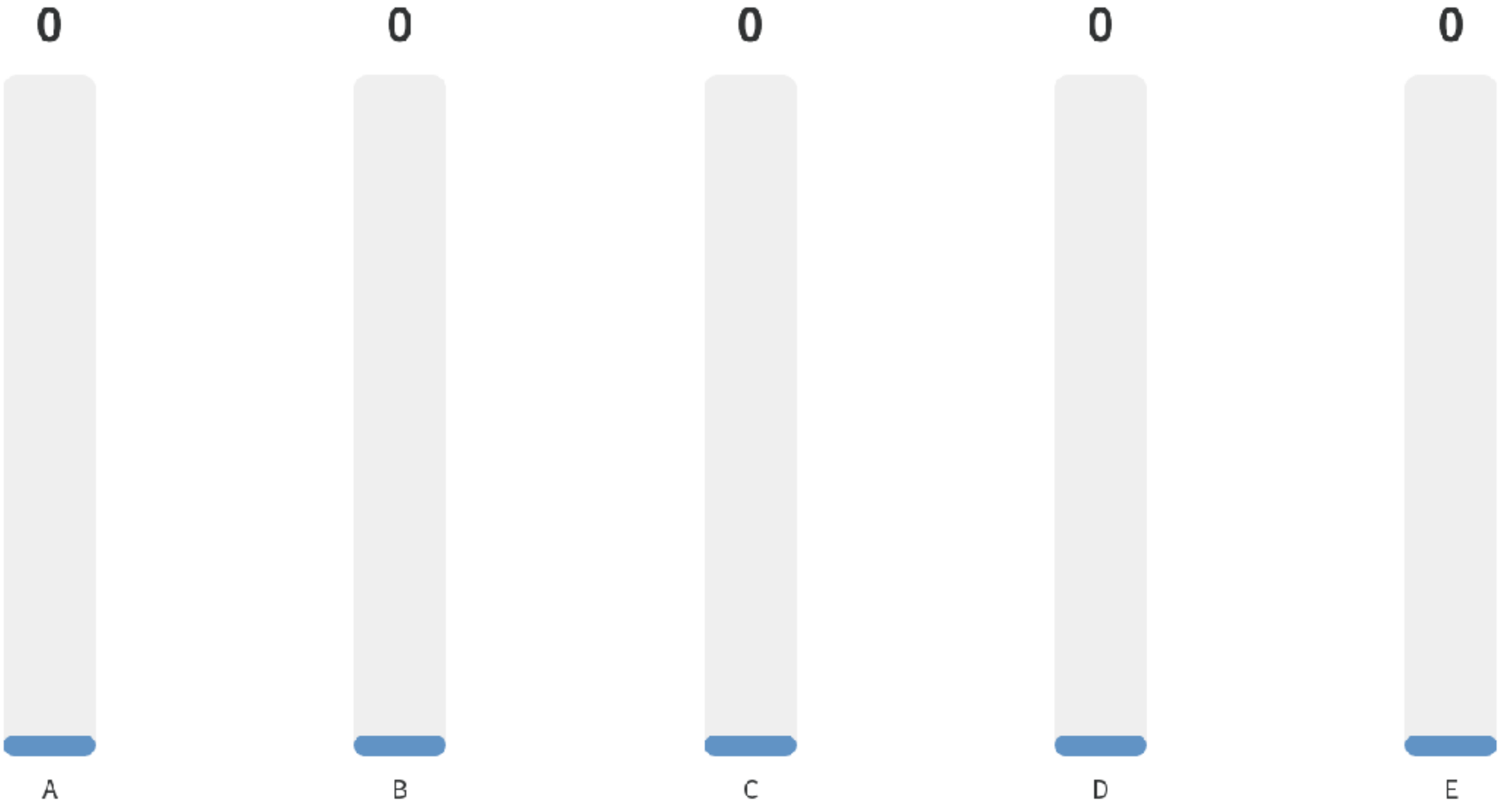


https://www.pollev.com/hungweitseng close in 1:30

Size of page table

- Assume that we have **64-bit** virtual address space, each page is 4KB, each page table entry is 8 Bytes, what magnitude in size is the page table for a process?
  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

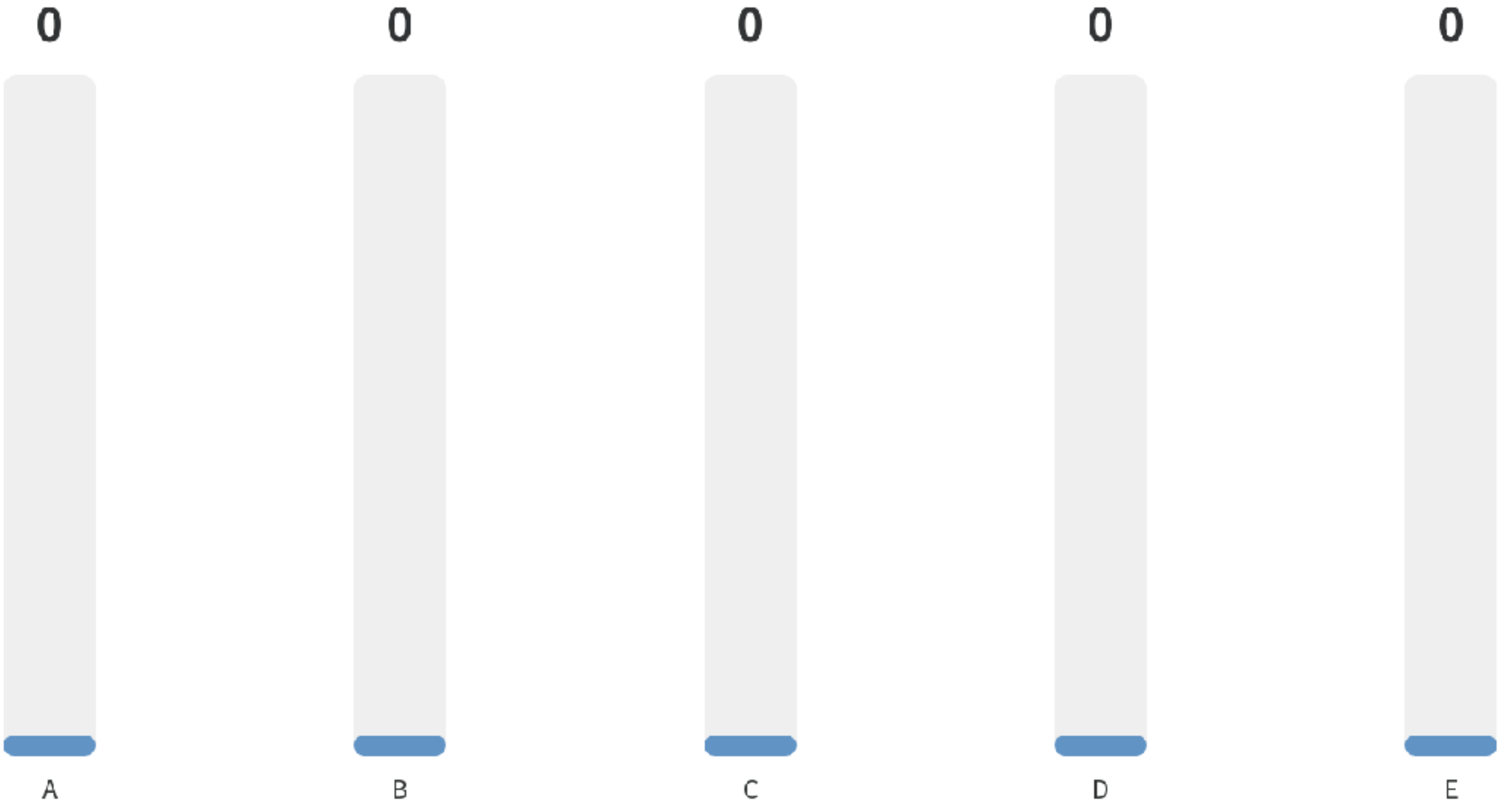
0



Size of page table

- Assume that we have **64-bit** virtual address space, each page is 4KB, each page table entry is 8 Bytes, what magnitude in size is the page table for a process?
  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

0



Size of page table

- Assume that we have **64-bit** virtual address space, each page is 4KB, each page table entry is 8 Bytes, what magnitude in size is the page table for a process?
  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

$$\frac{2^{64}\ Bytes}{4\ KB} \times 8\ Bytes = 2^{55}\ Bytes = 32\ PB$$

If you still don't know why — you need to take CS202

Conventional page table

0x0 0xFFFFFFFFFFFFFFFF

Virtual Address Space

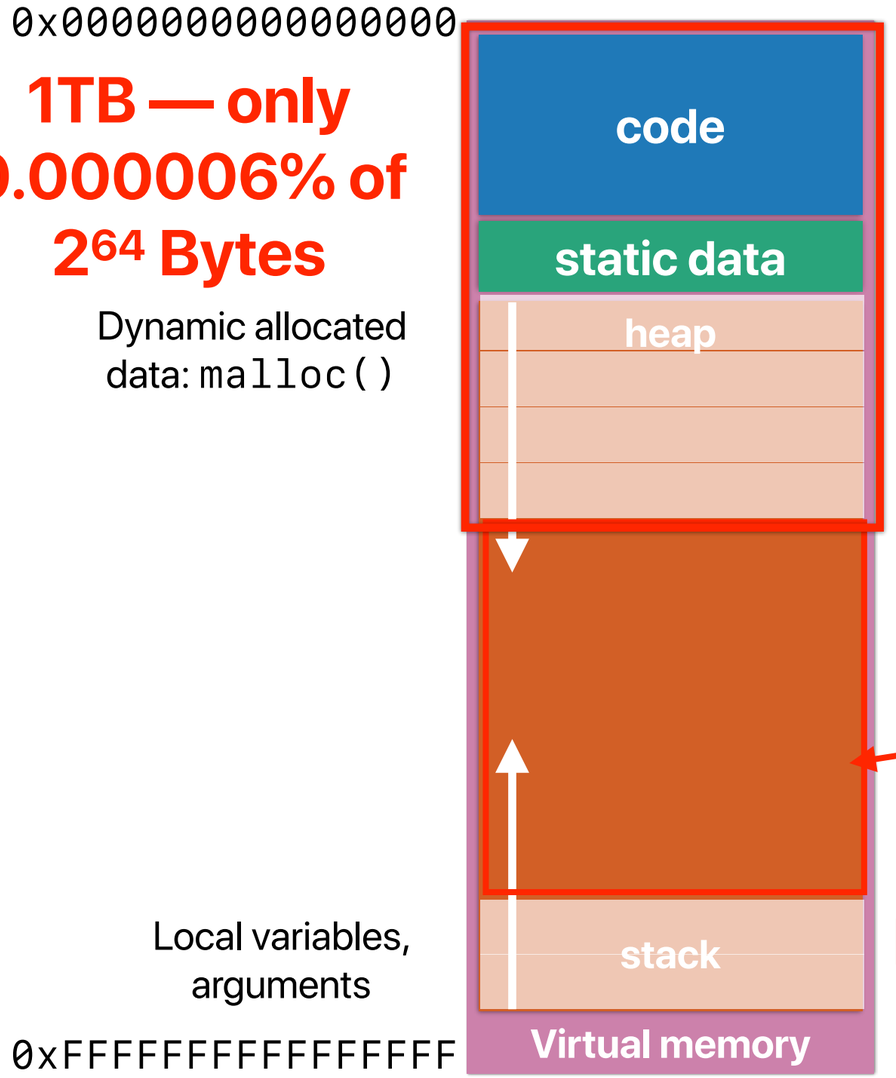
— must be consecutive in the physical memory

— need 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

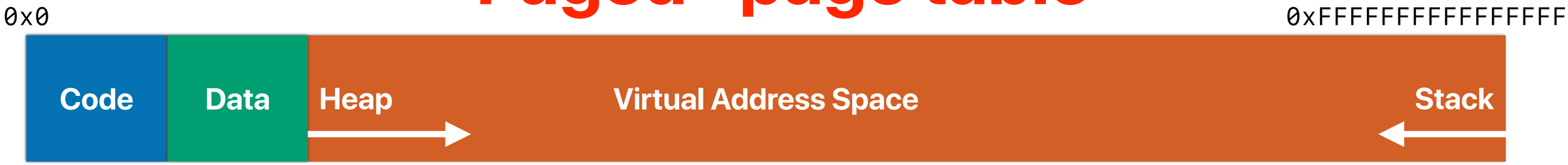
Do we really need a large table?



Your program probably never uses this huge area!

If you still don't know why — you need to take CS202

"Paged" page table

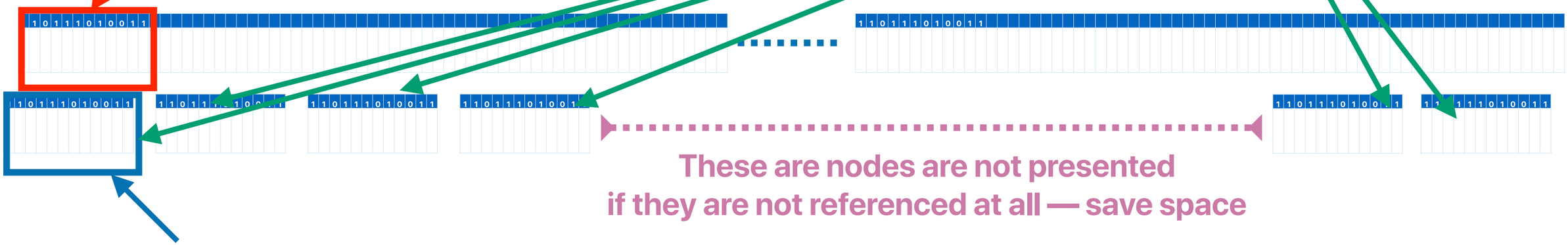


Break up entries into pages!
Each of these occupies exactly a page

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

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

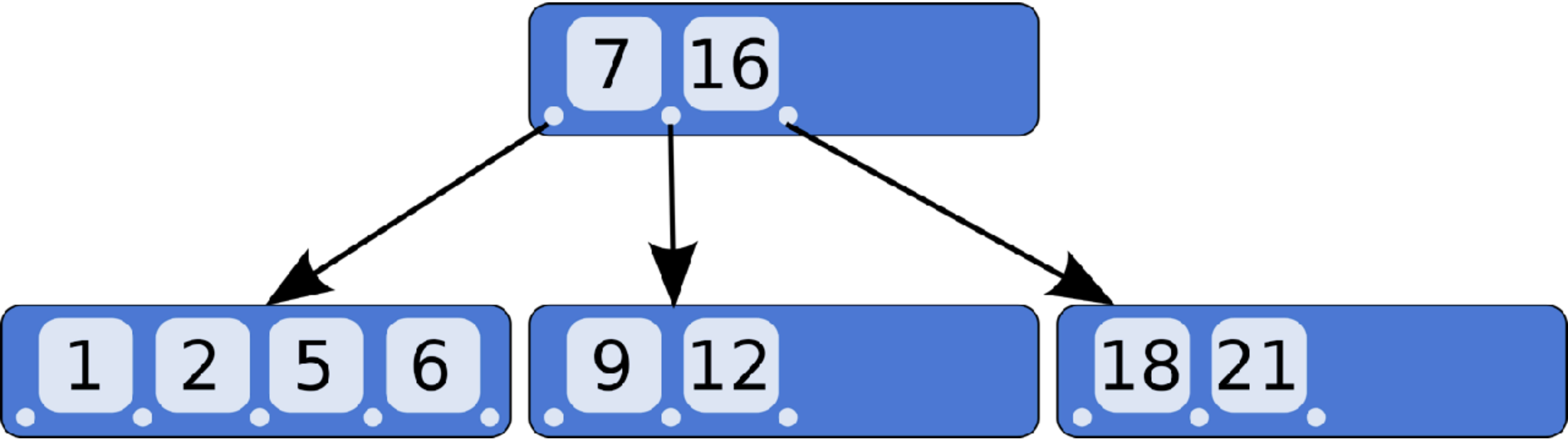
Otherwise, you always need to find more than one consecutive pages — difficult!



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

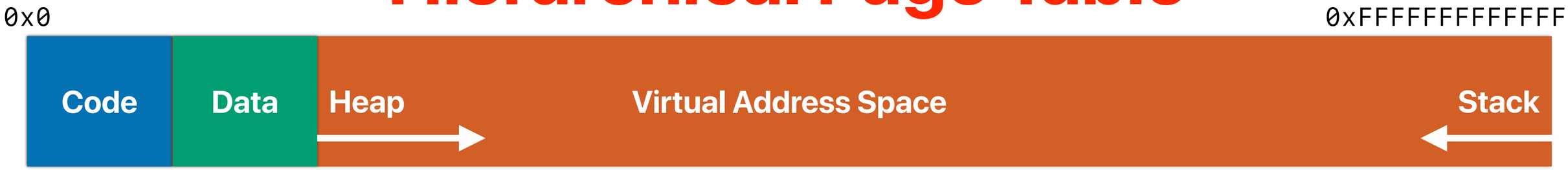
Allocate page table entry nodes "on demand"

B-tree



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

Hierarchical Page Table

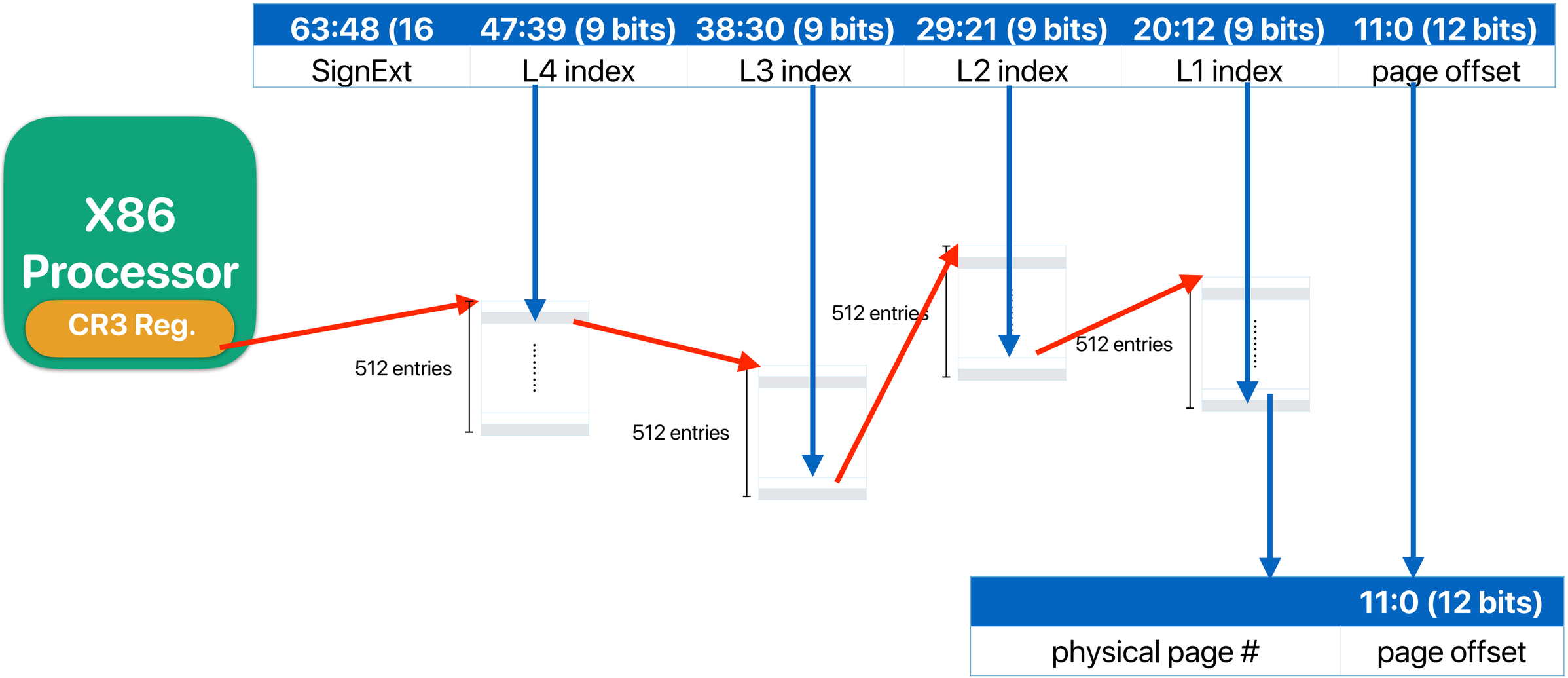


$$\lceil log_{2^9} \frac{2^{64}\ B}{2^{12}\ B} \rceil = \lceil log_{2^9} 2^{52} \rceil = 6 \text{ levels}$$

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)

Address translation in x86-64



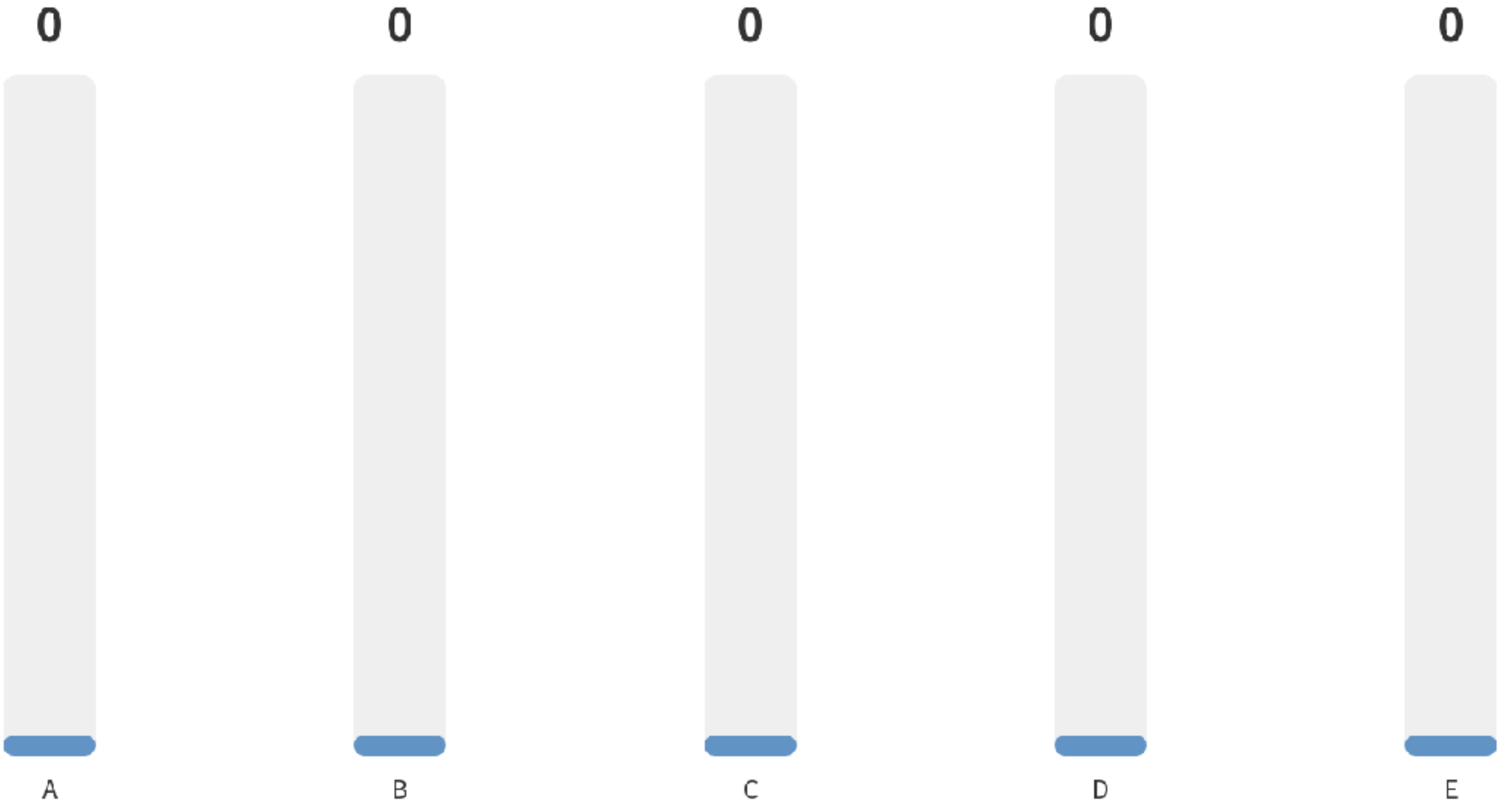
Translation Caching: Skip, Don't Walk (the Page Table)
Thomas W. Barr, Alan L. Cox, Scott Rixner

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

When we have virtual memory...

- If an x86 processor supports virtual memory through the basic format of the page table as shown in the previous slide, how many memory accesses can a **`mov`** instruction that access data memory once incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. 10

0

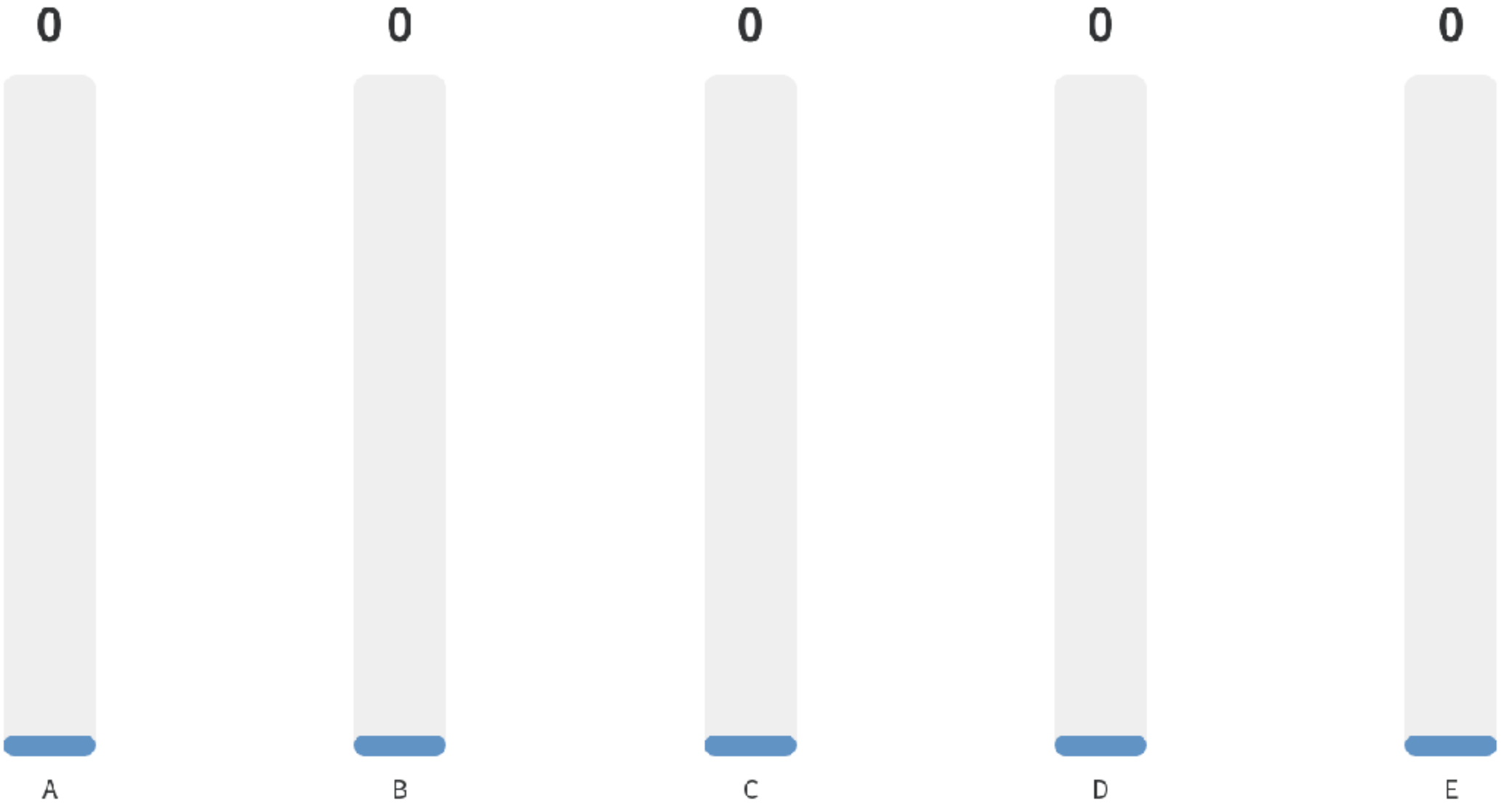


https://www.pollev.com/hungweitseng close in 1:30

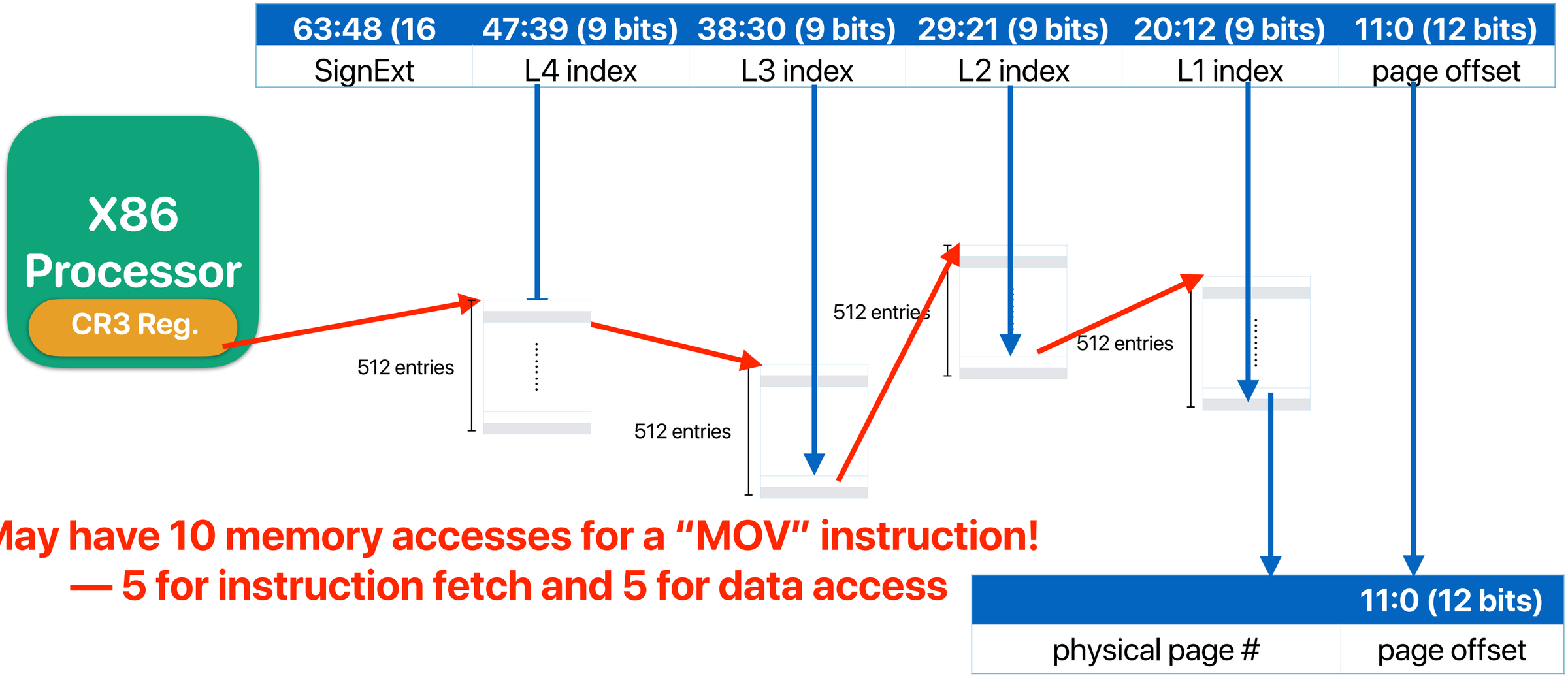
When we have virtual memory...

- If an x86 processor supports virtual memory through the basic format of the page table as shown in the previous slide, how many memory accesses can a **`mov`** instruction that access data memory once incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. 10

0



Address translation in x86-64



May have 10 memory accesses for a "MOV" instruction!

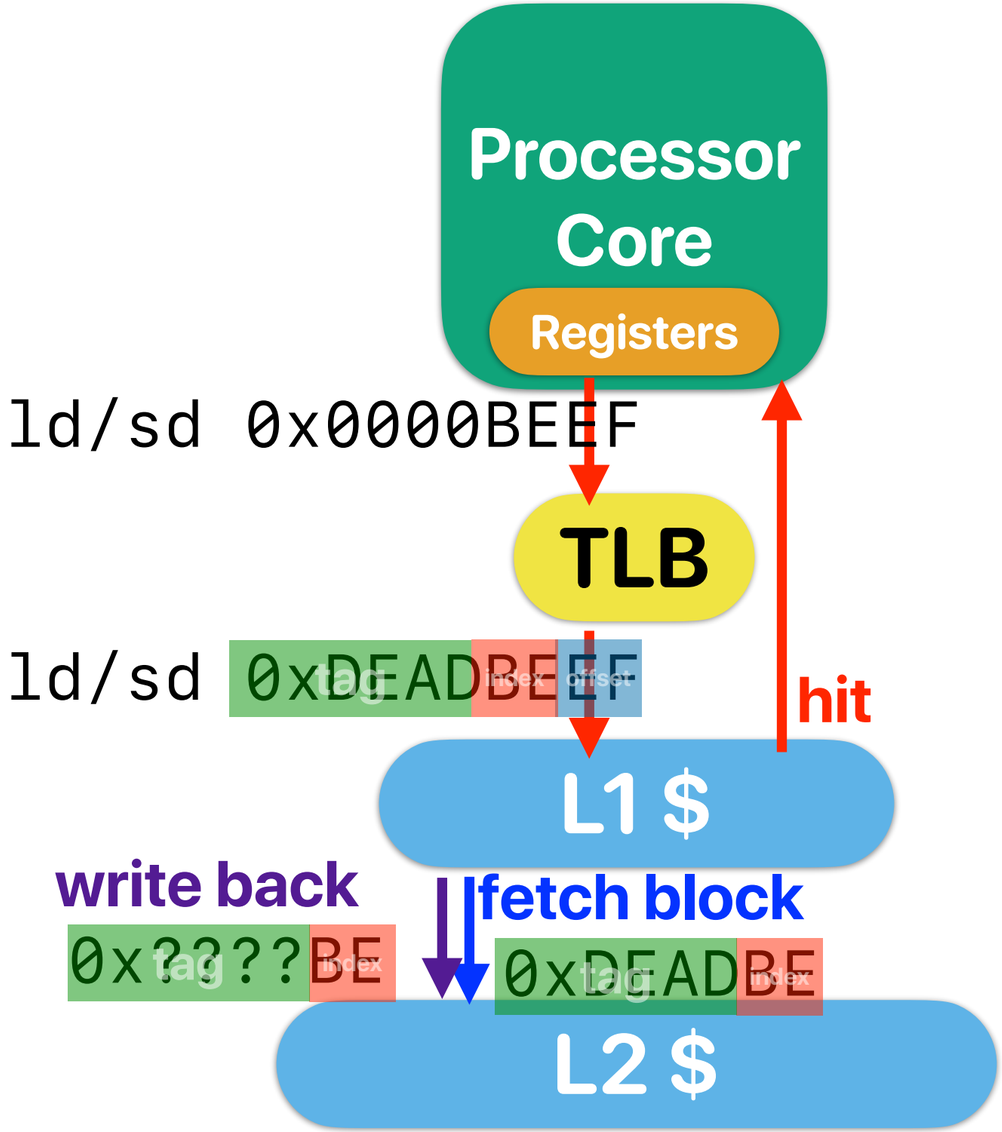
— 5 for instruction fetch and 5 for data access

When we have virtual memory…

- If an x86 processor supports virtual memory through the basic format of the page table as shown in the previous slide, how many memory accesses can a **`mov`** instruction that access data memory once incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. 10

Avoiding the address translation overhead

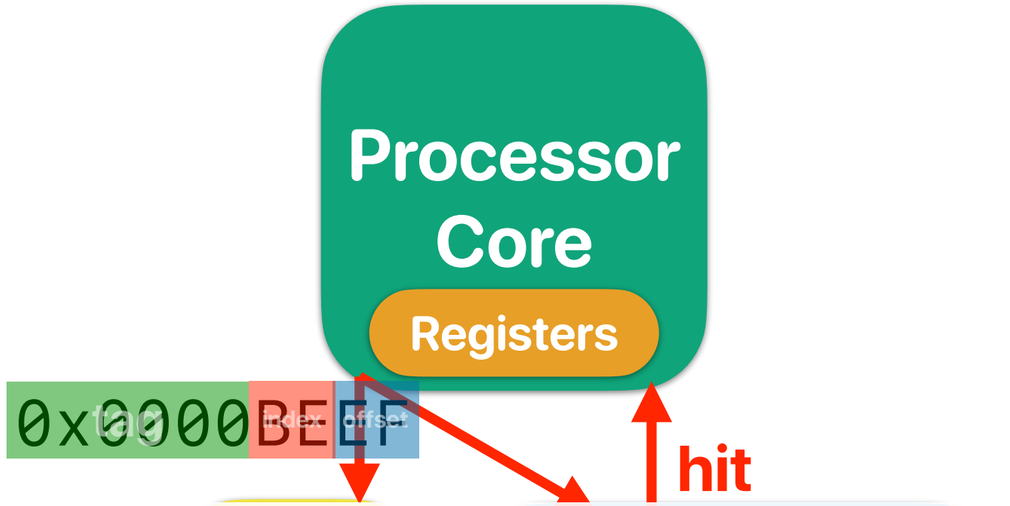
TLB: Translation Look-aside Buffer



- TLB — a small SRAM stores frequently used page table entries
- Good — A lot faster than having everything going to the DRAM
- Bad — Still on the critical path

TLB + Virtual cache

- L1 $ accepts virtual address — you don't need to translate
- Good — you can access both TLB and L1-$ at the same time and physical address is only needed if L1-$ misses
- Bad — it doesn't work in practice
  - Many applications have the same virtual address but should be pointing different **physical addresses**
  - An application can have "aliasing virtual addresses" pointing to the same **physical address**



You really need "physical address" to judge if that's what you want

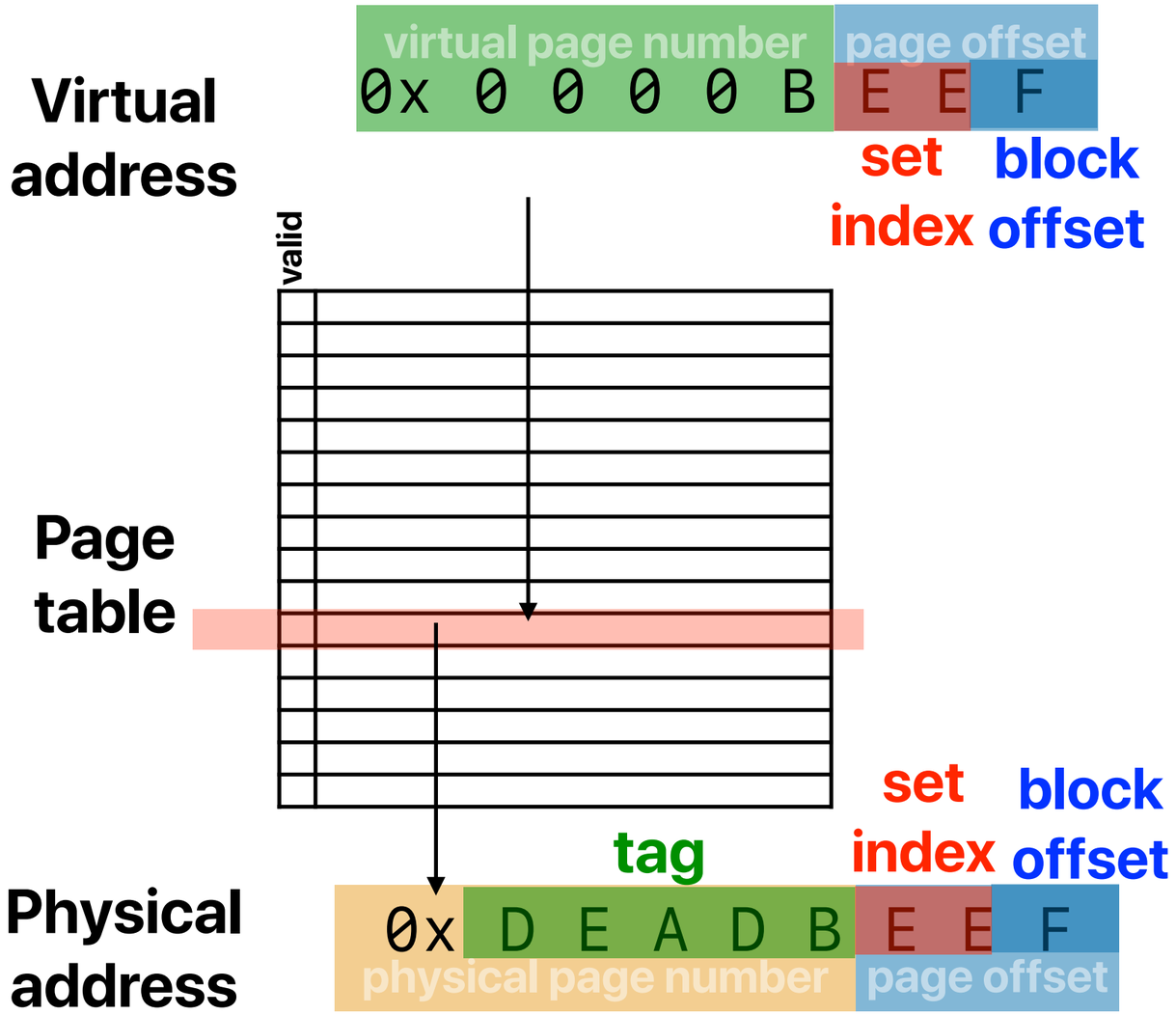
Virtually indexed, physically tagged cache

- Can we find physical address directly in the virtual address

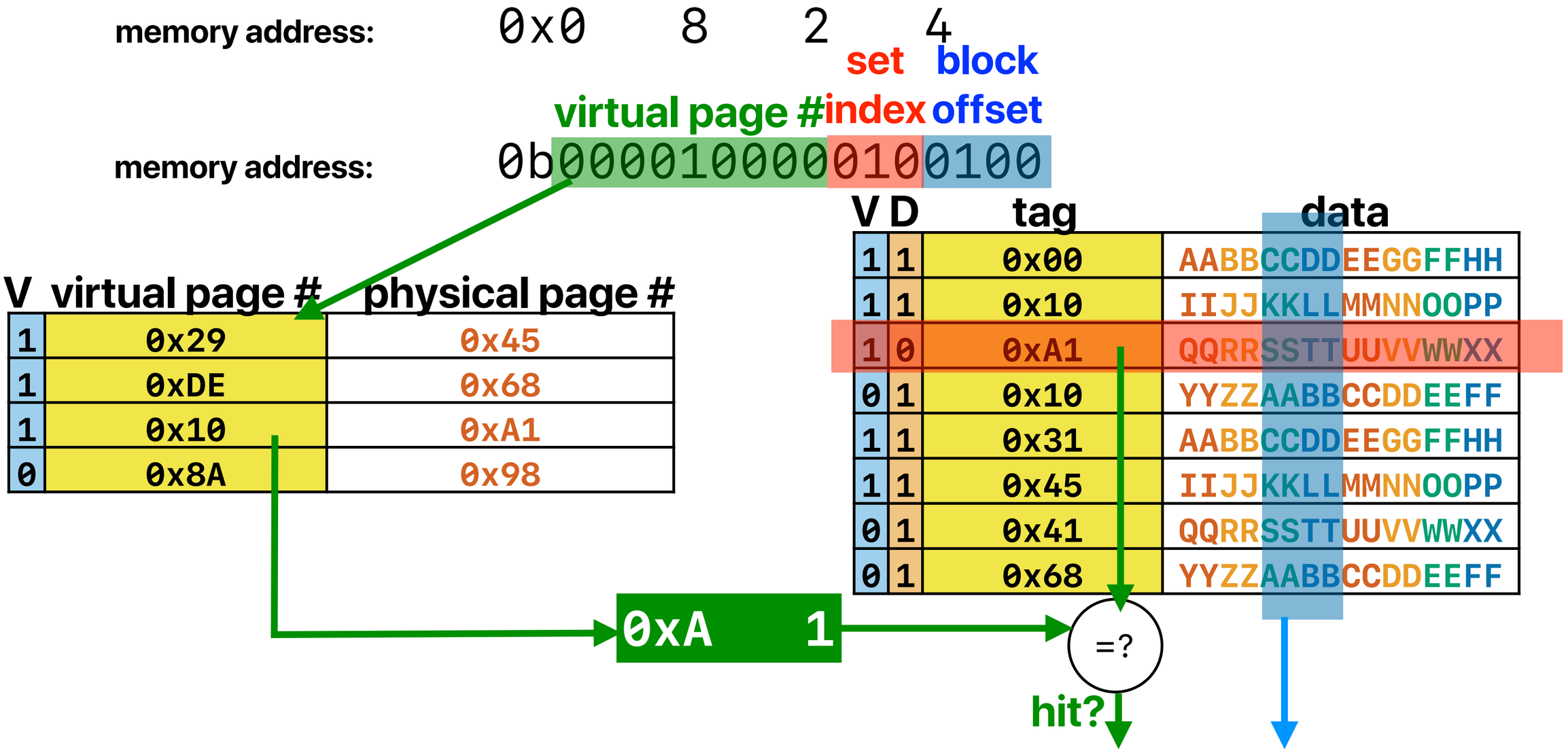
  — Not everything — but the page offset isn't changing!

- Can we indexing the cache using the "partial physical address"?

  — Yes — Just make set index + block set to be exactly the page offset



Virtually indexed, physically tagged cache



Virtually indexed, physically tagged cache

- If page size is 4KB —

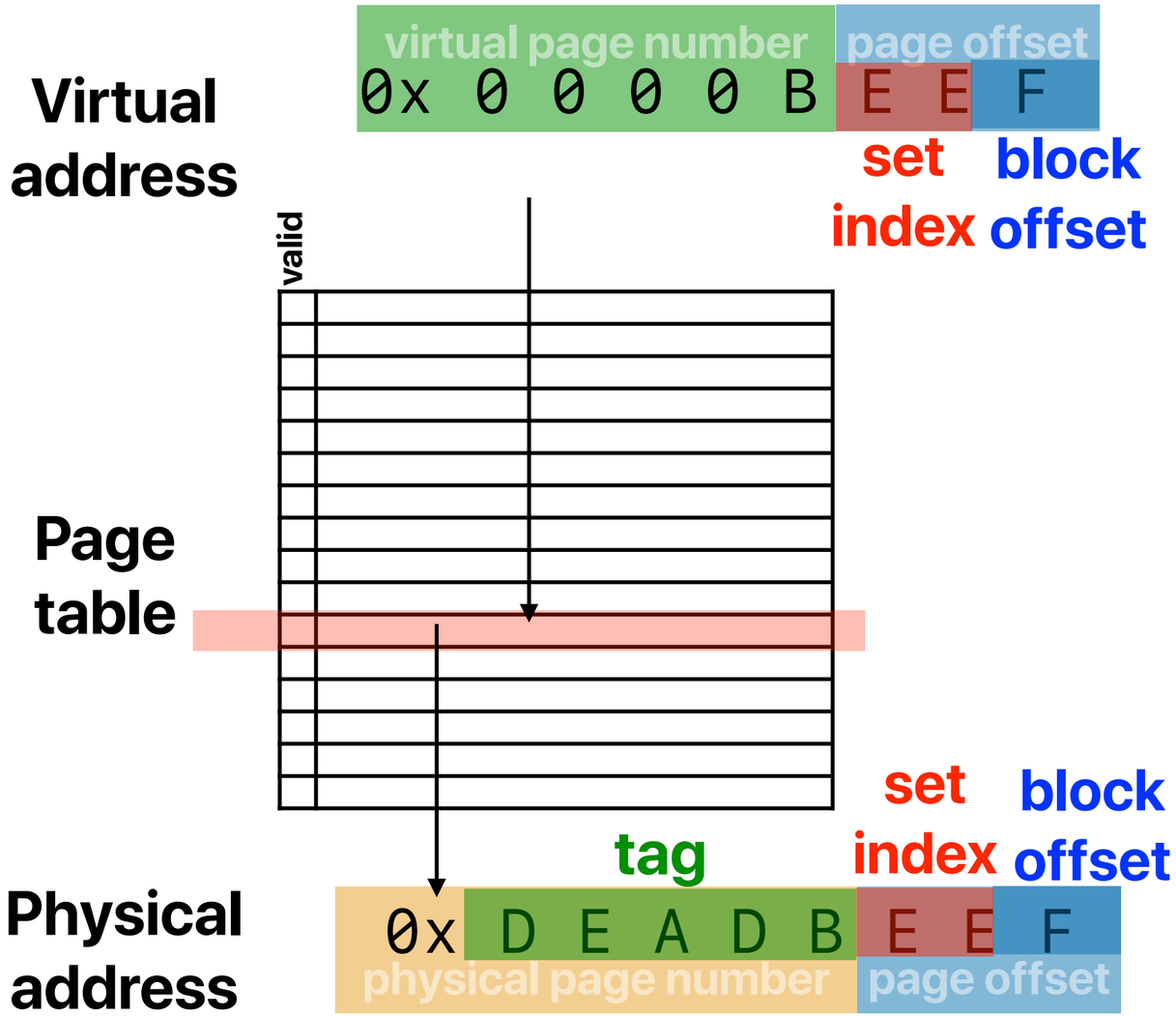
$$lg(B) + lg(S) = lg(4096) = 12$$

$$C = ABS$$

$$C = A \times 2^{12}$$

$$if\ A = 1$$

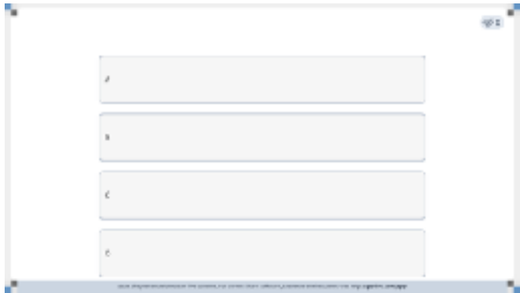
$$C = 4KB$$



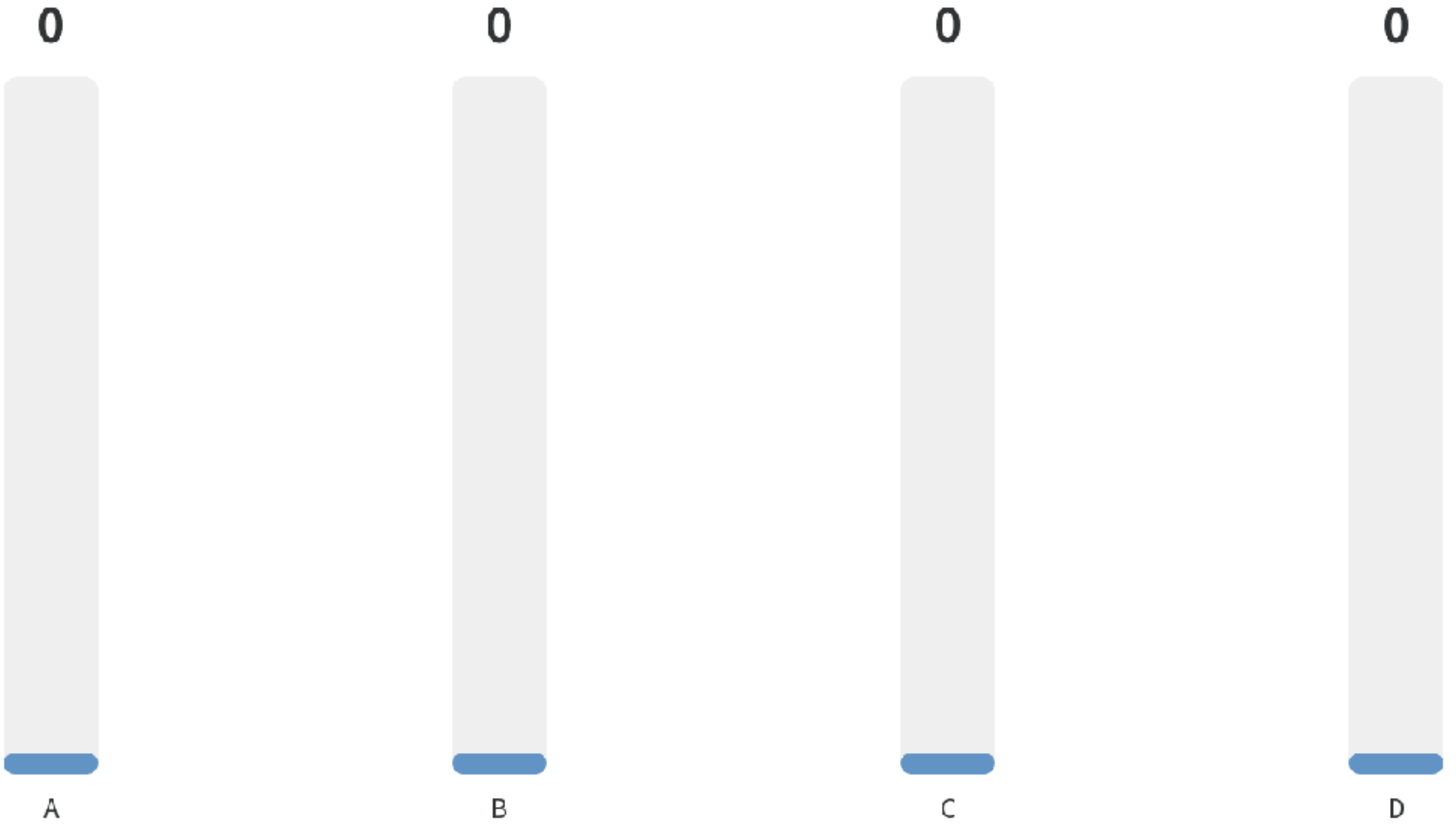
https://www.pollev.com/hungweitseng close in 1:30

Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the operating system use 4K pages.
  A. 32B blocks, 2-way
  B. 32B blocks, 4-way
  C. 64B blocks, 4-way
  D. 64B blocks, 8-way



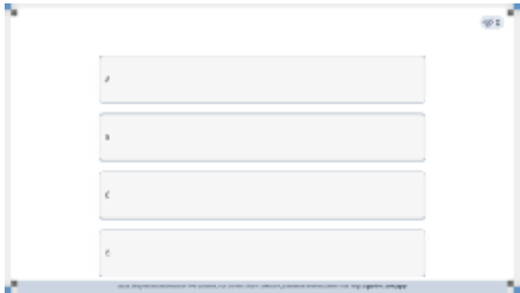
0



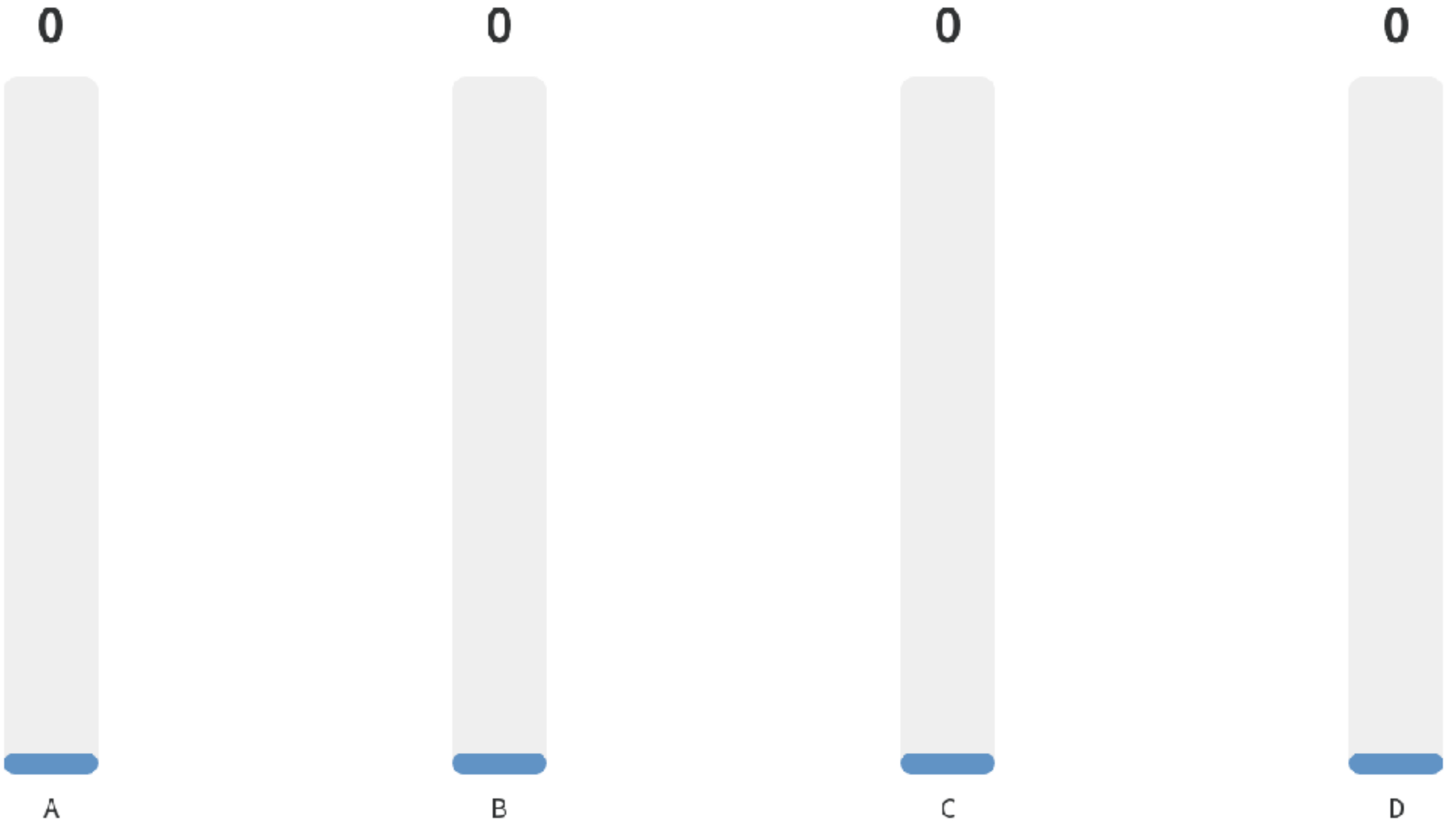
https://www.pollev.com/hungweitseng close in 1:30

Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the operating system use 4K pages.
  A. 32B blocks, 2-way
  B. 32B blocks, 4-way
  C. 64B blocks, 4-way
  D. 64B blocks, 8-way



0



Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the operating system use 4K pages.

A. 32B blocks, 2-way
B. 32B blocks, 4-way
C. 64B blocks, 4-way
D. 64B blocks, 8-way

Exactly how Core i7 9th generation configures its own cache

$$lg(B) + lg(S) = lg(4096) = 12$$

$$C = ABS$$

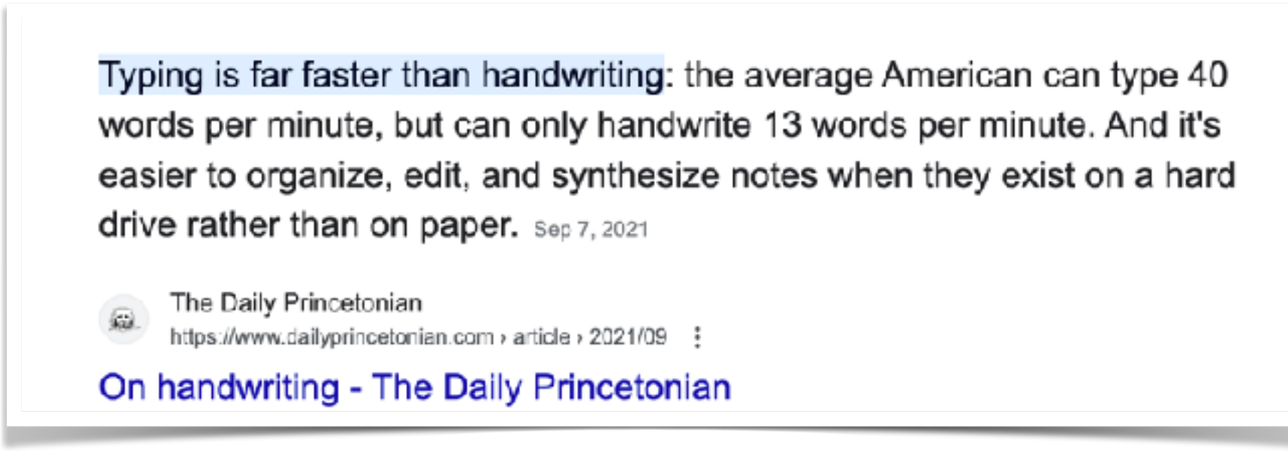
$$32KB = A \times 2^{12}$$

$$A = 8$$


Q & A

Announcement

- Reading quiz due this Wednesday before the lecture
- Midterm tomorrow — 2p-3:20p, no exceptions
  - Be sure to watch the midterm review lecture
  - Sample midterm slides are already released with the last lecture — no solution should be provided as questions are mostly from previous lectures
  - In-person examine
    - Closed-book, closed notebook. No cheat sheet allowed
  - Online examine will have 1.5x questions compared to in-person ones
    - Since typing is faster — based on my teaching experience and scientific research
    - Learning LaTeX equation editing is part of the course — frequently used in scientific editing and becoming the standard in MS Office and Mac
    - It's an open-book/note test and self-proctored — you can preset some frequently used formula and we don't have many in this class
    - Discussion/communication with any other during the examine is prohibited



Computer
Science &
Engineering

142

つづく