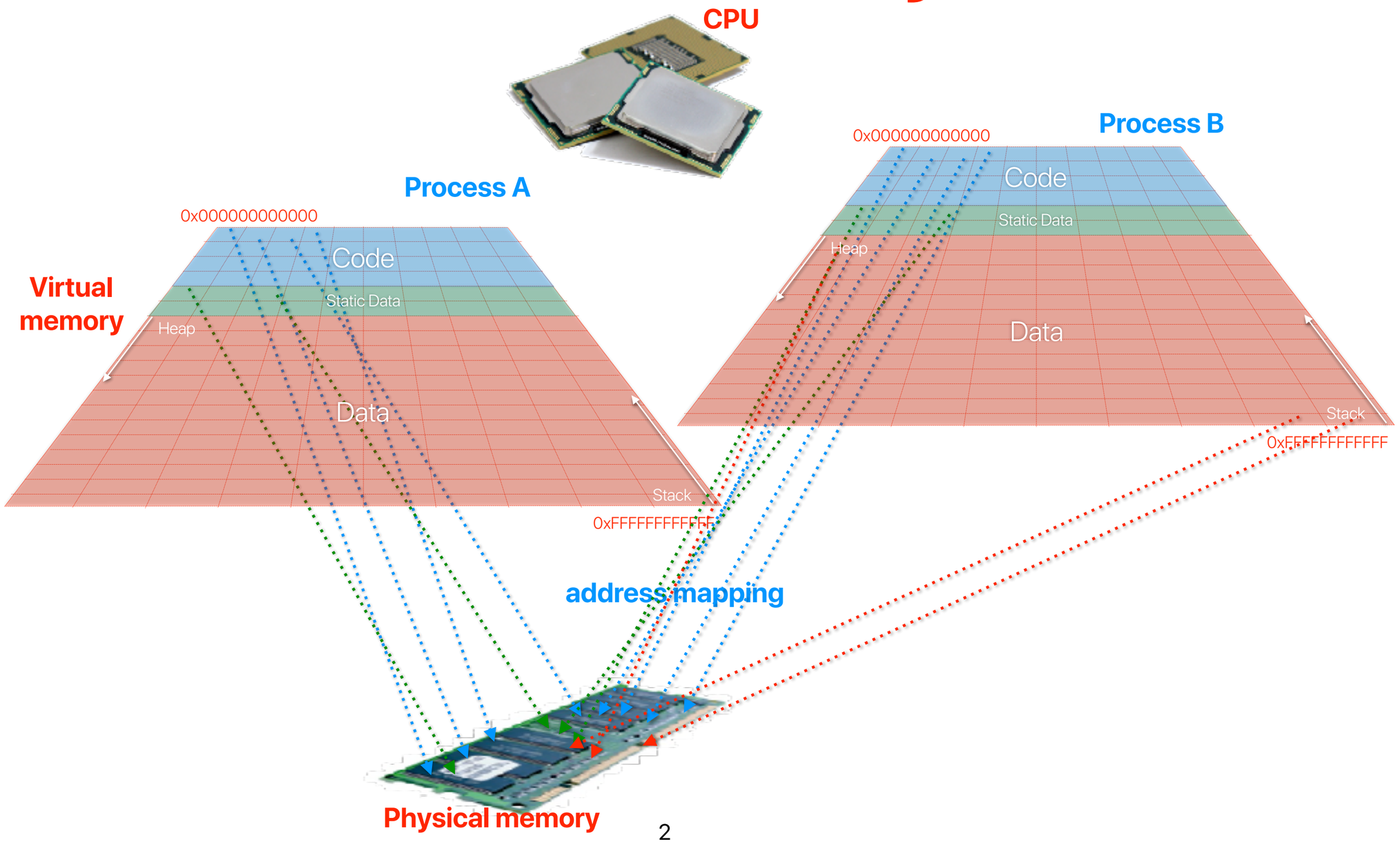
Virtual memory design in operating systems

Hung-Wei Tseng

Virtual Memory



Recap: Demand paging

instruction
0x0

data
0x80000000

Page fault!

Virtual Address Space for Chrome

Page fault!

instruction
0x0

Page fault!

data
0x80008000

Page fault!

Virtual Address Space for Apple Music

Program

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

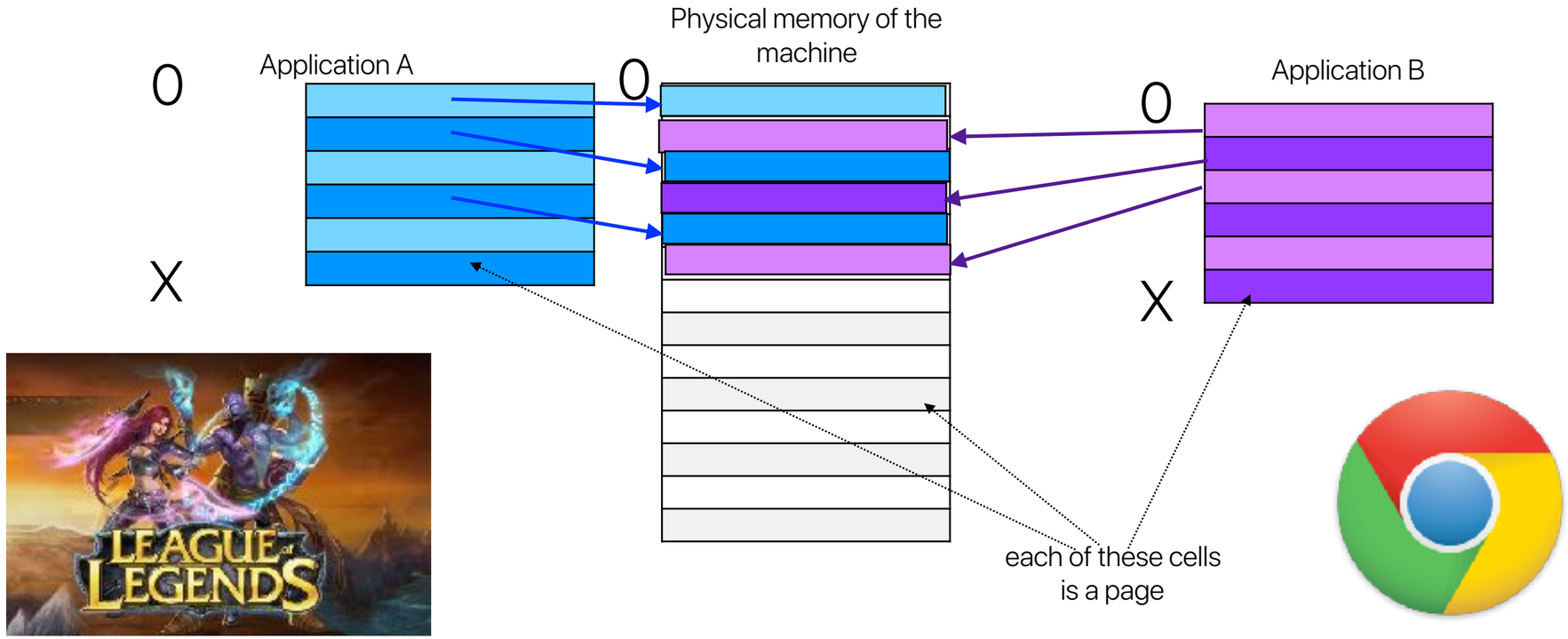
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
0f00bb27	00c2f800
509cbd23	00000008
00005d24	00c30000
0000bd24	00000008

Memory

Program

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

Recap: Demand paging



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 — the main reason why we love paging! — 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

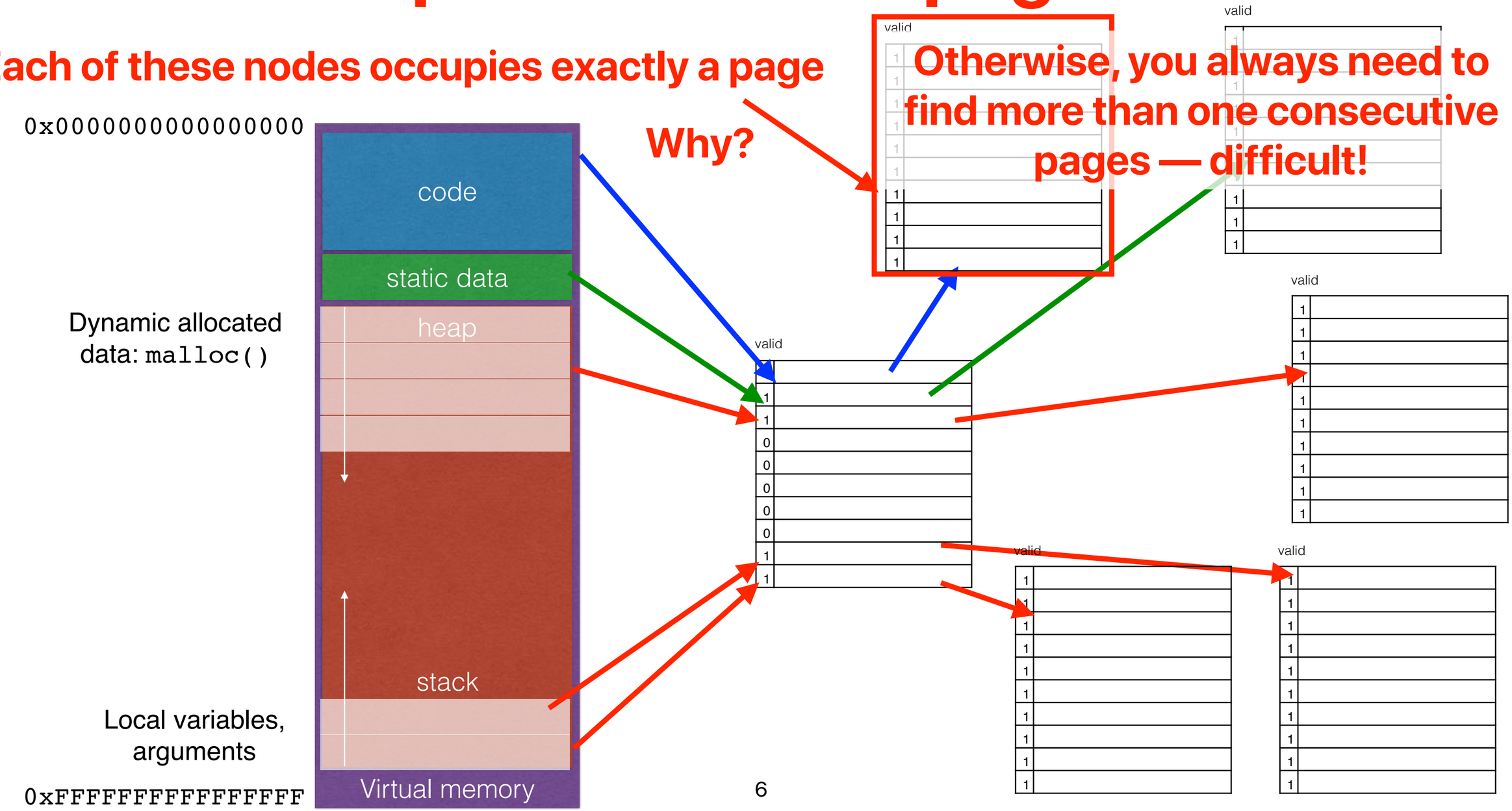
We haven't seen pure/true implementation of segmentations for a while, but we still use segmentation fault errors all the time!

Recap: Hierarchical page table

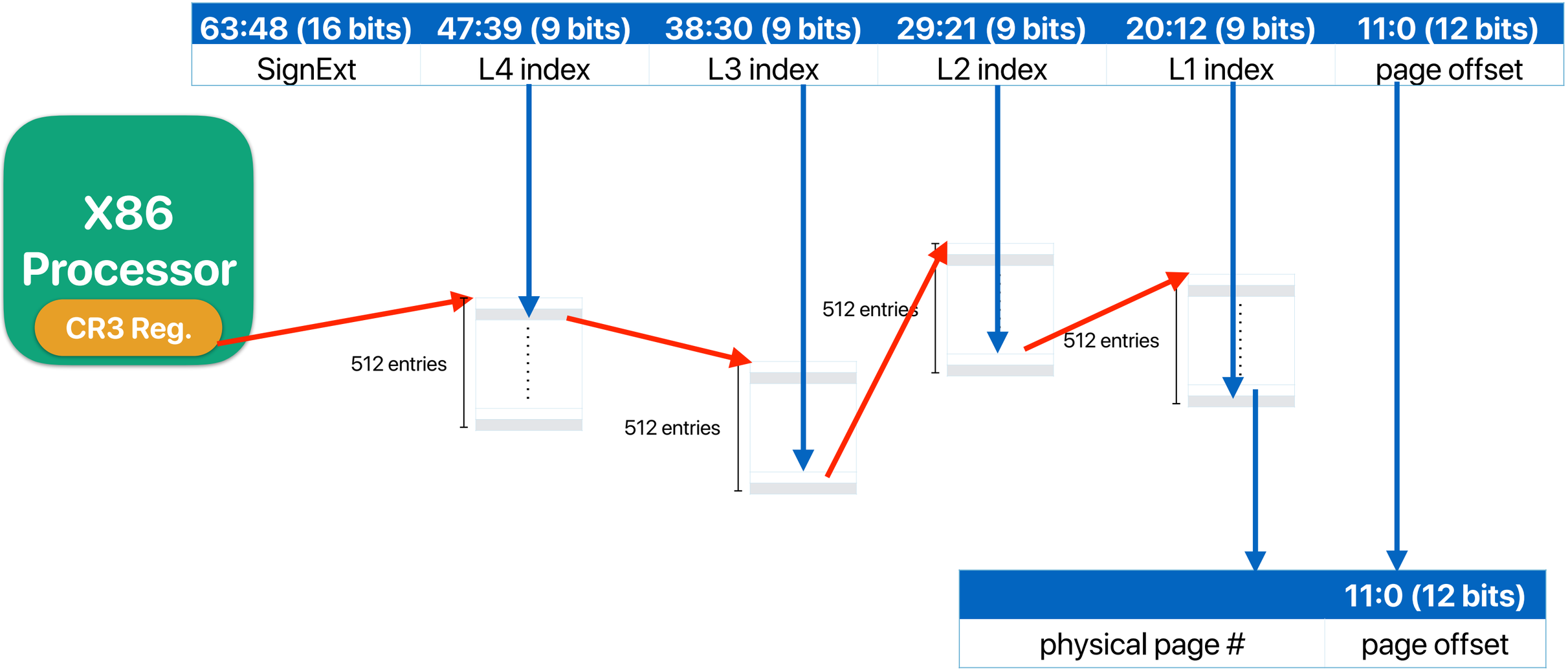
Each of these nodes occupies exactly a page

Why?

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

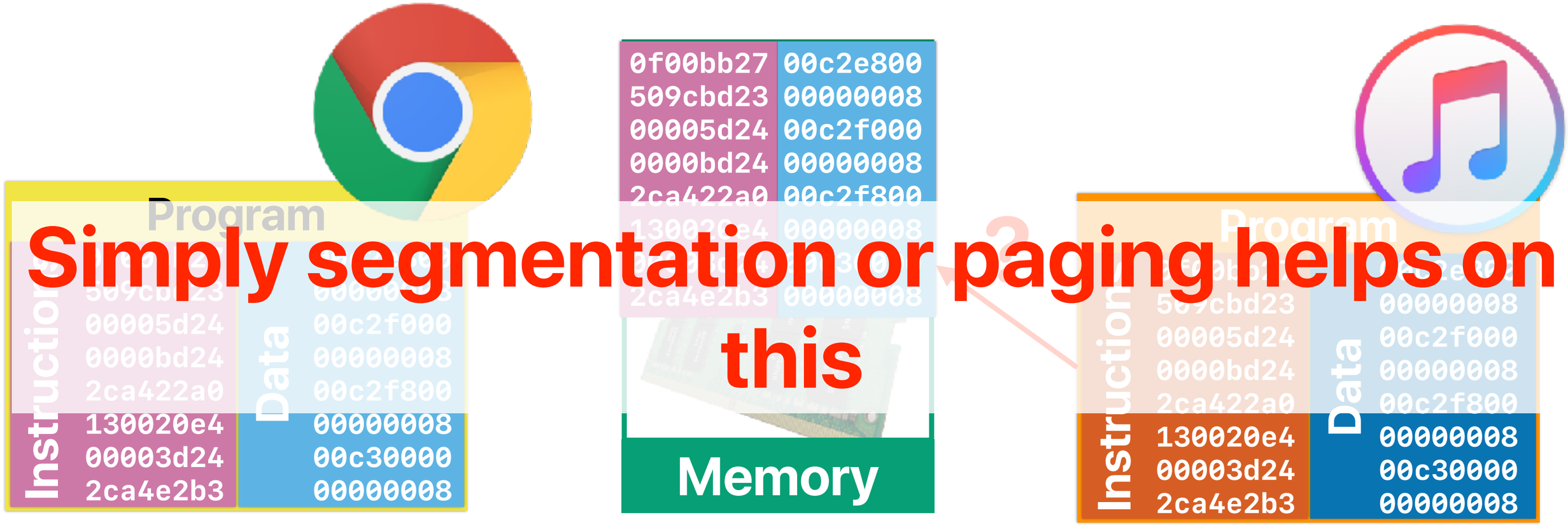


Case study: Address translation in x86-64



Recap: If we expose memory directly to the processor

What if both programs need to use memory?



Simply segmentation or paging helps on this

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

00c2f800
00000008
00c30000
00000008

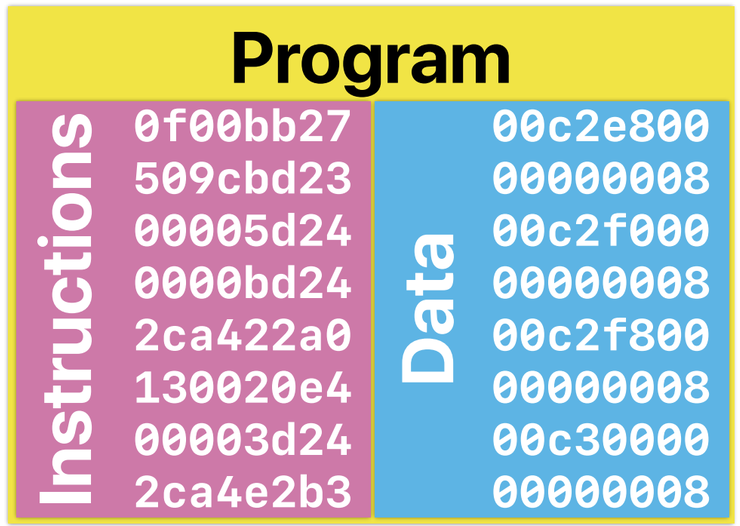
?

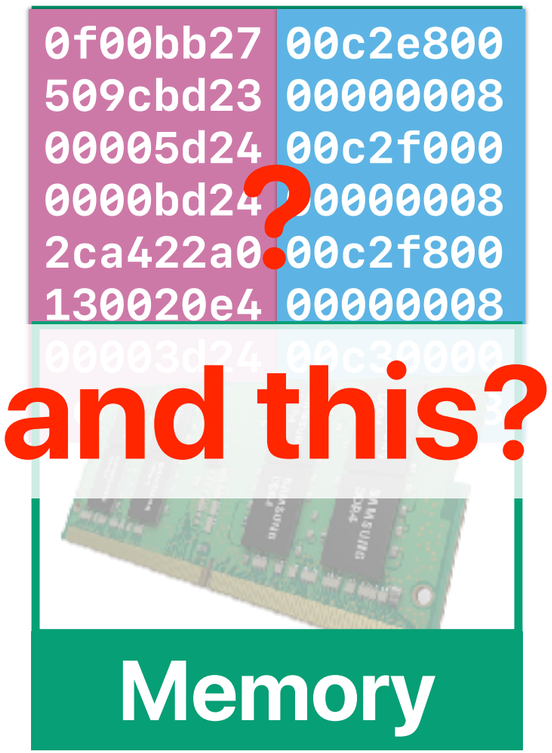
What if my program needs more memory?

But how about this?

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

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

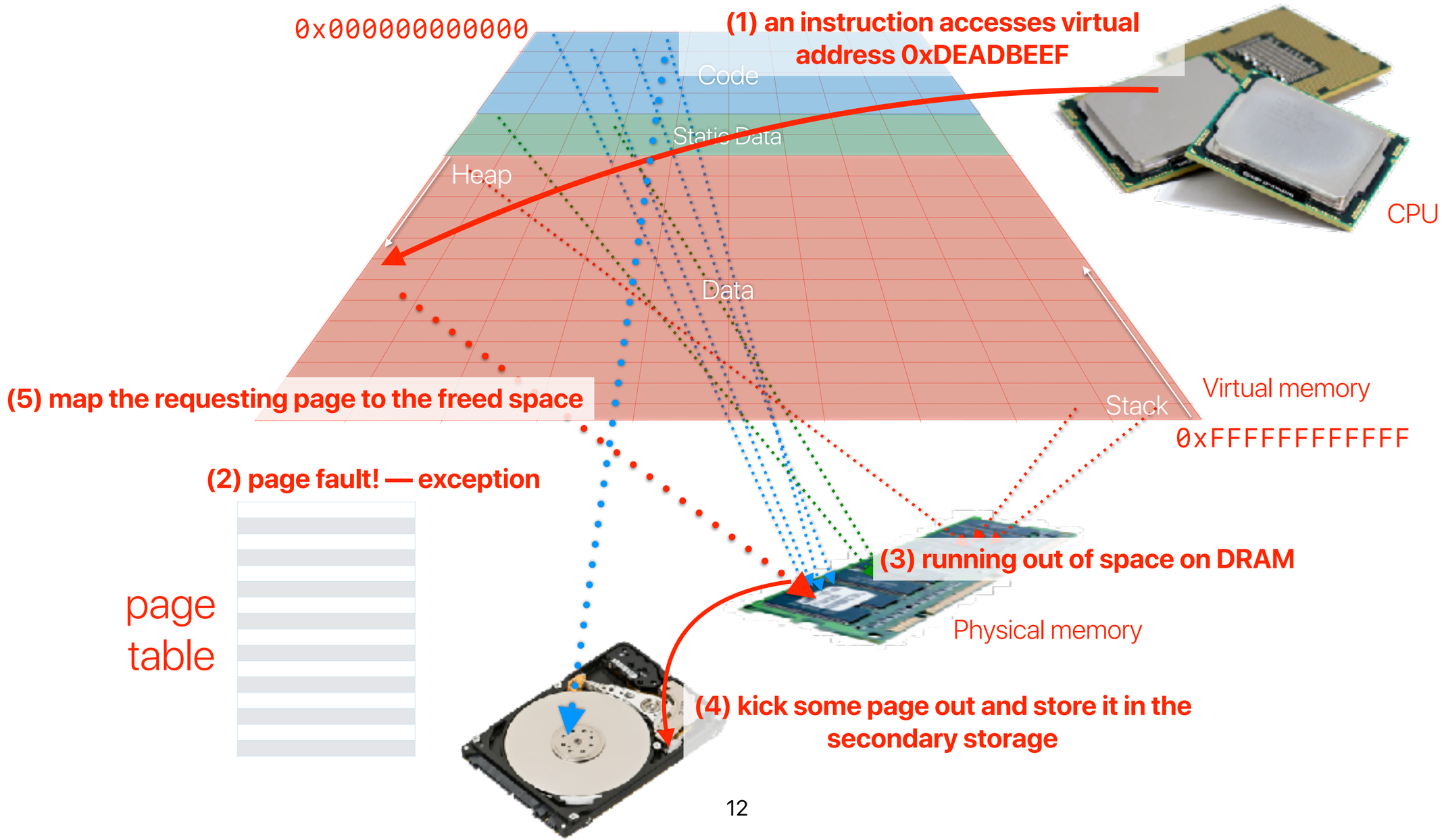




Outline

- Swapping
- VAX/VMS Design
- Mach VM

Demand Paging + Swapping



The mechanism: demand paging + swapping

- Divide physical & virtual memory spaces into fix-sized units — pages
- Allocate a physical memory page whenever the virtual memory page containing your data is absent
- In case if we are running out of physical memory —
  - Reserve space on disks
    - Disks are slow: the access time for HDDs is around 10 ms, the access time for SSDs is around 30us - 1 ms
    - Disks are orders of magnitude larger than main memory
  - When you need to make rooms in the physical main memory, allocate a page in the swap space and put the content of the evicted page there
  - When you need to reference a page in the swap space, make a room in the physical main memory and swap the disk space with the evicted page

Latency Numbers Every Programmer Should Know

Operations	Latency (ns)	Latency (us)	Latency (ms)	
L1 cache reference	0.5 ns			~ 1 CPU cycle
Branch mispredict	5 ns			
L2 cache reference	7 ns			14x L1 cache
Mutex lock/unlock	25 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000 ns	3 us		
Send 1K bytes over 1 Gbps network	10,000 ns	10 us		
Read 4K randomly from SSD*	150,000 ns	150 us		~1GB/sec SSD
Read 1 MB sequentially from memory	250,000 ns	250 us		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from SSD*	1,000,000 ns	1,000 us	1 ms	~1GB/sec SSD, 4X memory
Disk seek	10,000,000 ns	10,000 us	10 ms	20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000 ns	20,000 us	20 ms	80x memory, 20X SSD
Send packet CA-Netherlands-CA	150,000,000 ns	150,000 us	150 ms	

The swapping overhead

- How much slower (approximately) is your average memory access time in a system when the probability of a page fault/swapping is 0.1% comparing with the case when there is no page fault/swapping? (Assume you swap to a hard disk)
  - Memory (i.e. RAM) access time: 100ns
  - Disk access time: 10ms
  - P_f: probability of a page fault
  - Effective Access Time = 100 ns + P_f * 10^7 ns
  - When P_f = 0.001:
    Effective Access Time = 10,100ns
  - When P_f = 0.001, even with an SSD
    Effective Access Time = 100 ns + 10^{-3} * 10^5 ns = 200 ns
  - Takeaway: disk accesses are tolerable only when they are extremely rare

Operations	Latency (ns)
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 1K bytes over 1 Gbps network	10,000 ns
Read 4K randomly from SSD*	150,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Read 1 MB sequentially from SSD*	1,000,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA-Netherlands-CA	150,000,000 ns

The swapping overhead

- How much slower (approximately) is your average memory access time in a system when the probability of a page fault/ swapping is 0.1% comparing with the case when there is no page fault/swapping?
  (Assume you swap to a hard disk)
  A. 10x
  B. 100x
  C. 1000x
  D. 10000x
  E. 100000x

Operations	Latency (ns)
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 1K bytes over 1 Gbps network	10,000 ns
Read 4K randomly from SSD*	150,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Read 1 MB sequentially from SSD*	1,000,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA-Netherlands-CA	150,000,000 ns

Page replacement policy

- Goal: Identify page to remove that will avoid future page faults (i.e. utilize locality as much as possible)
- Implementation Goal: Minimize the amount of software and hardware overhead
  - Example:
    - Memory (i.e. RAM) access time: 100ns
    - Disk access time: 10ms
    - P_f: probability of a page fault
    - Effective Access Time = $10^{-7} + P_f * 10^{-3}$
  - When $P_f = 0.001$:
    Effective Access Time = 10,100ns
  - Takeaway: Disk access tolerable only when it is extremely rare

Virtual Memory Management in the VAX/VMS Operating System

H. M. Levy and P. H. Lipman
Digital Equipment Corporation

The “Why” behind VAX/VMS VM

- The system needs to execute various types of applications efficiently
- The system runs on different types of hardware
- As a result, the memory management system has to be capable of adjusting the changing demands characteristic of time sharing while allowing predictable performance required by real-time and batch processes

The goals of VAX/VMS

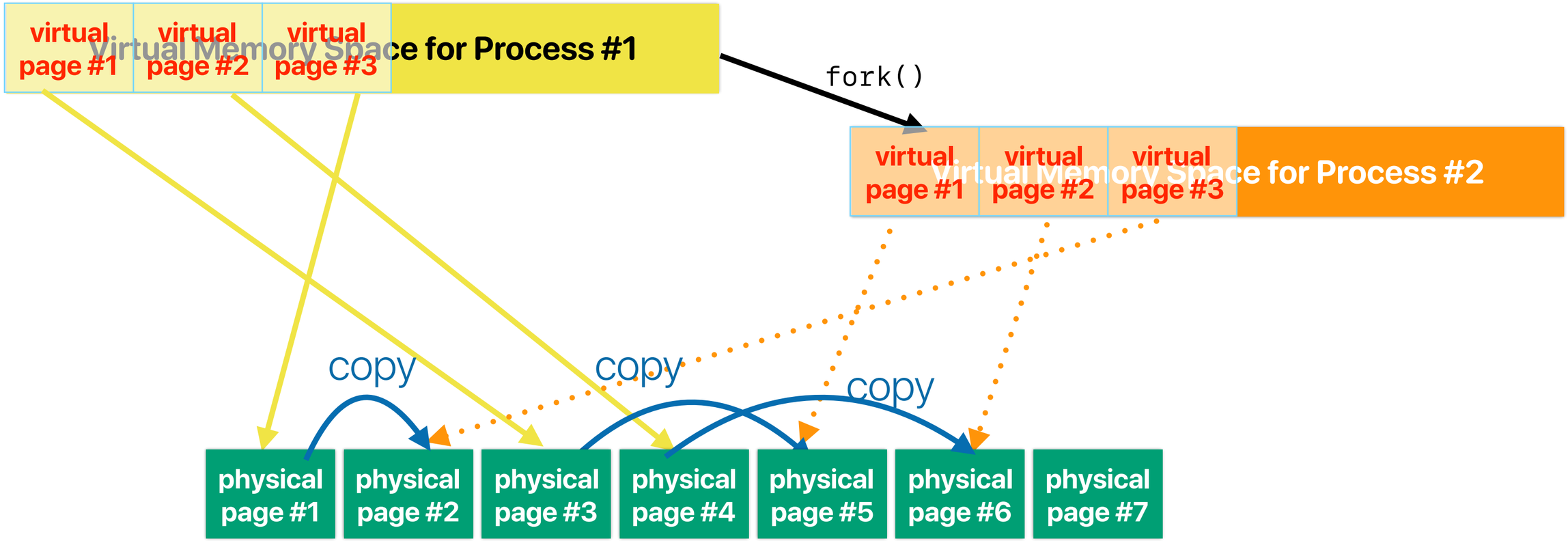
- How many of the following statements is/are true regarding the optimization goals of VAX/VMS?
  - ① Reducing the disk load of paging
  - ② Reducing the startup cost of a program
  - ③ Reducing the overhead of page tables
  - ④ Reducing the interference from heavily paging processes
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

What VAX/VMS proposed to achieve these goals?

- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

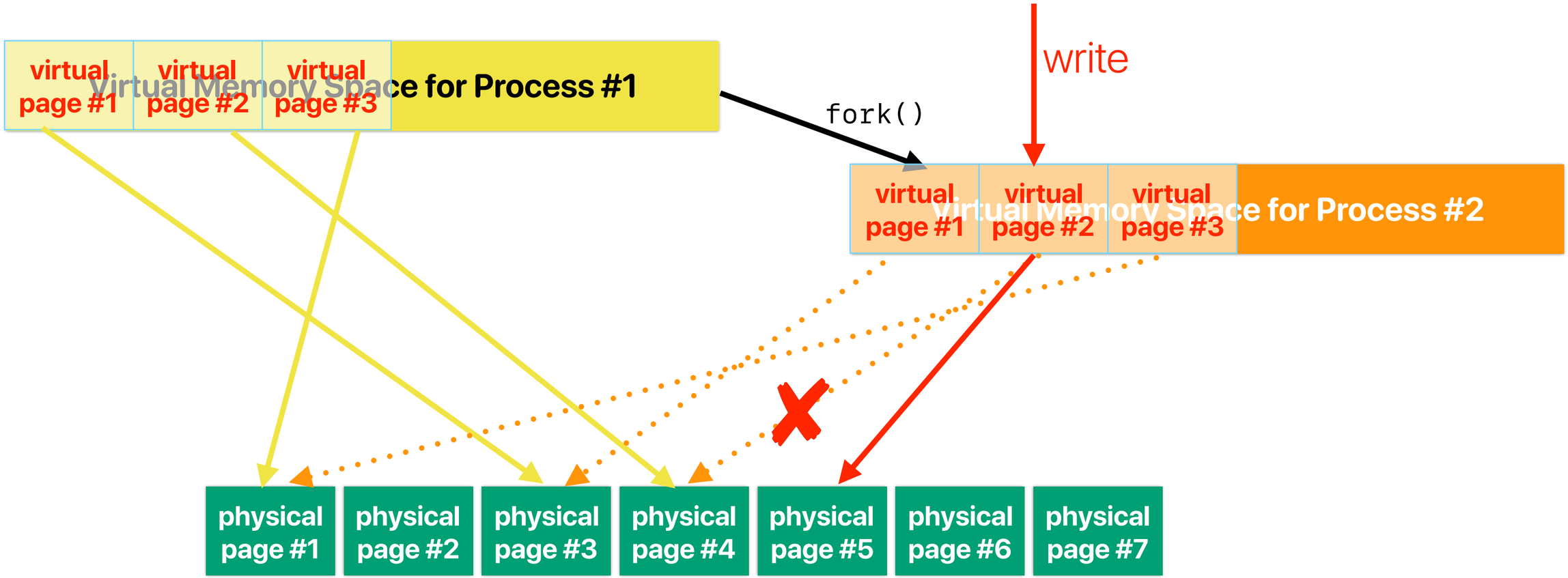
	Goal		Optimization
A	Process startup cost	**W**	Demand-zero & copy-on-refernce
B	Process performance interference	**X**	Process-local replacement
C	Page table lookup overhead	**Y**	Page clustering
D	Paging load on disks	**Z**	Page caching

What happens on a `fork`?



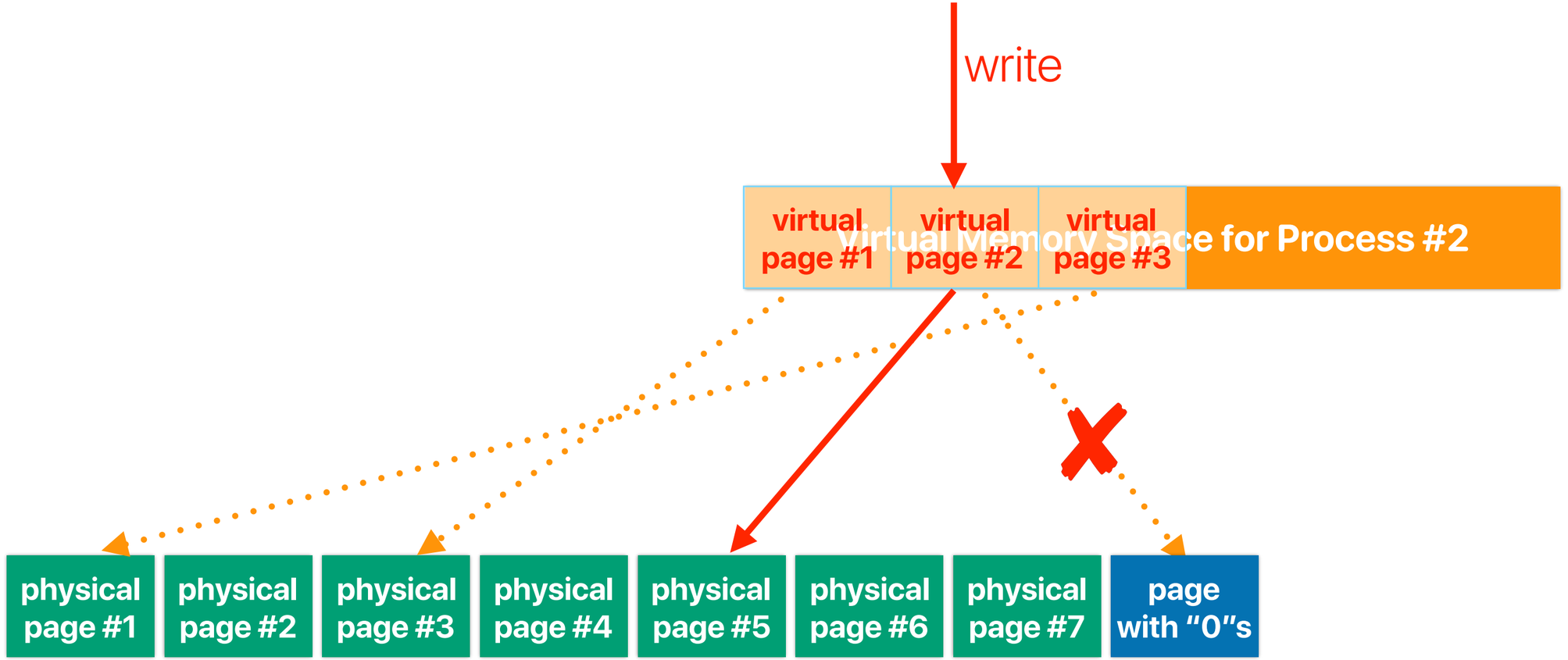
- **Copy the page content to different locations before the new process can start**

Copy-on-write



- The modified bit of a writable page will be set when it's loaded from the executable file
- The process eventually will have its own copy of that page

Demand zero



- The linker does not embed the pages with all 0s in the compiled program
- When page fault occurs, allocate a physical page fills with zeros
- Set the modified bit so that the page can be written back

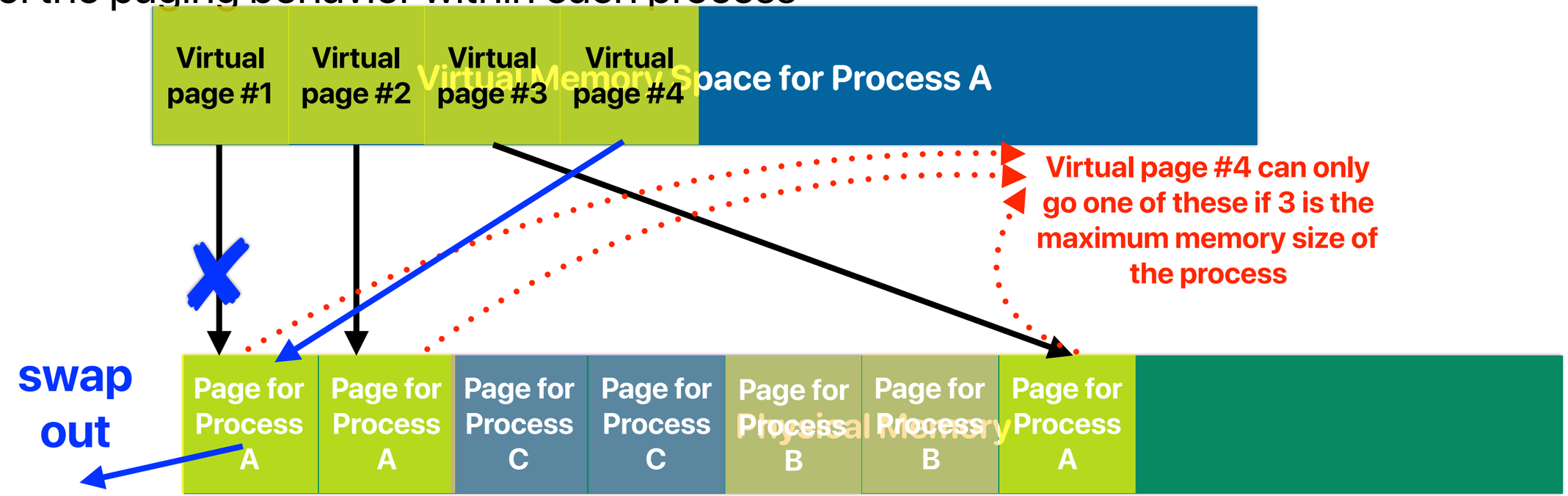
What VAX/VMS proposed to achieve these goals?

- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

Goal		Optimization	
A	Process startup cost	W	Demand-zero & copy-on-refernce
B	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	Page clustering
D	Paging load on disks	Z	Page caching

Local page replacement policy

- Each process has a maximum size of memory
- When the process exceeds the maximum size, replaces from its own set of memory pages
- Control the paging behavior within each process



What's the policy? **FIFO!** **Low overhead!**

What VAX/VMS proposed to achieve these goals?

- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

Goal		Optimization	
A	Process startup cost	W	Demand-zero & copy-on-refernce
B	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	Page clustering
D	Paging load on disks	Z	Page caching

Page clustering

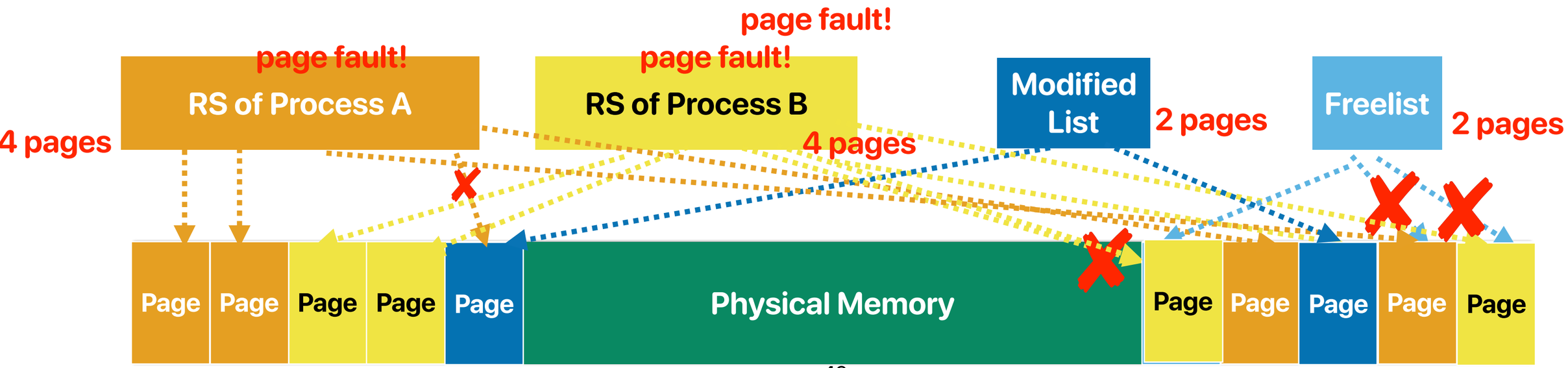
- Read or write a cluster of pages that are both consecutive in virtual memory and the disk
- Combining consecutive writes into single writes

Latency Numbers Every Programmer Should Know

Operations	Latency (ns)	Latency (us)	Latency (ms)	
L1 cache reference	0.5 ns			~ 1 CPU cycle
Branch mispredict	5 ns			
L2 cache reference	7 ns			14x L1 cache
Mutex lock/unlock	25 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cach
Compress 1K bytes with Zippy	3,000 ns	3 us		
Send 1K bytes over 1 Gbps network	10,000 ns	10 us		
Read 4K randomly from SSD*	150,000 ns	150 us		~1GB/sec SSD
Read 1 MB sequentially from memory	250,000 ns	250 us		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from SSD*	1,000,000 ns	1,000 us	1 ms	~1GB/sec SSD, 4X memory
Disk seek for a 512B sector	10,000,000 ns	10,000 us	10 ms	20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000 ns	20,000 us	20 ms	80x memory, 20X SSD
Send packet CA-Netherlands-CA	150,000,000 ns	150,000 us	150 ms	

Page caching to cover the performance loss

- Evicted pages will be put into one of the lists in DRAM
  - Free list: clean pages
  - Modified list: dirty pages — needs to copy data to the disk
- Page fault to any of the page in the lists will bring the page back
  - Reduces the demand of accessing disks



Page caching

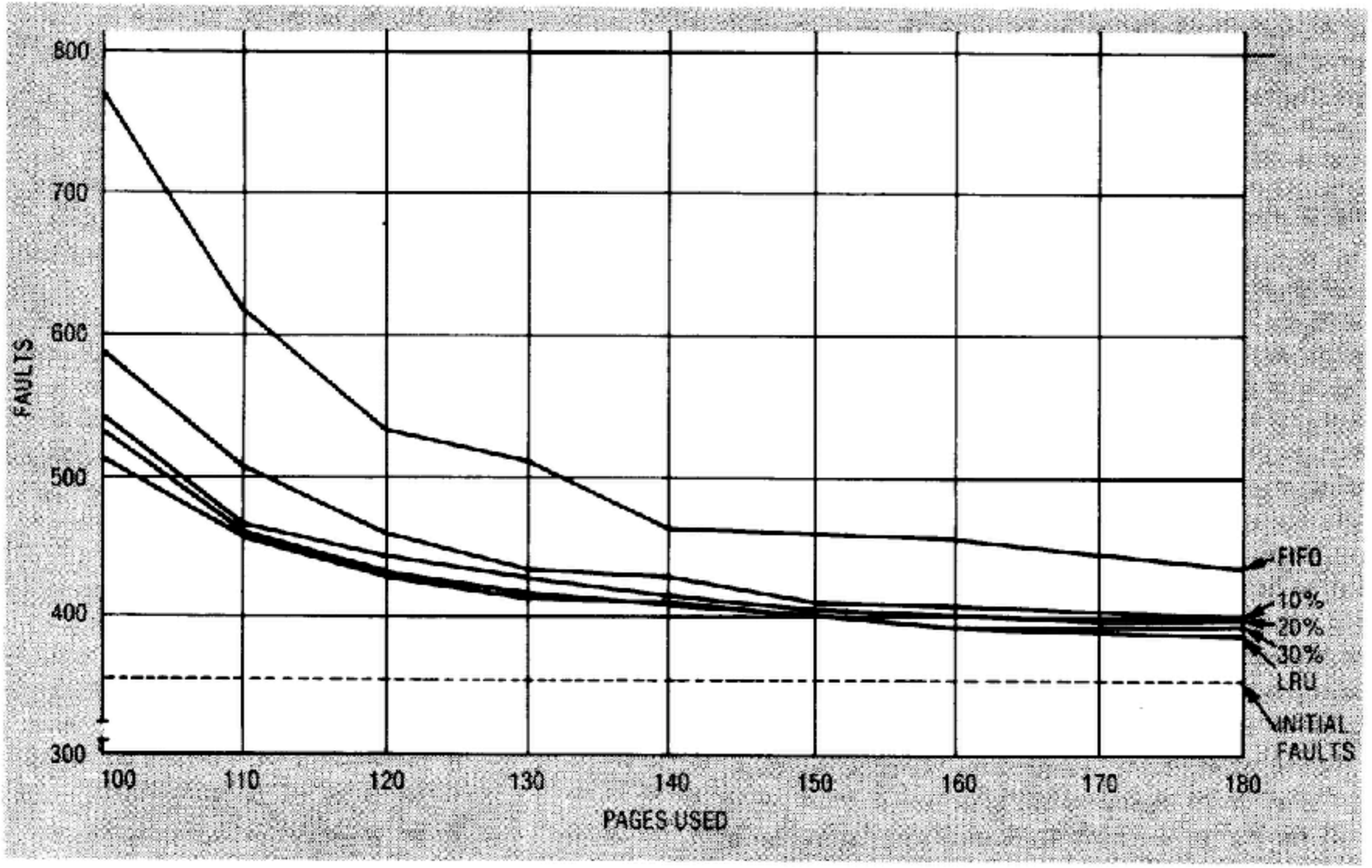


Figure 3. Faults vs. memory usage in Fortran compilation.

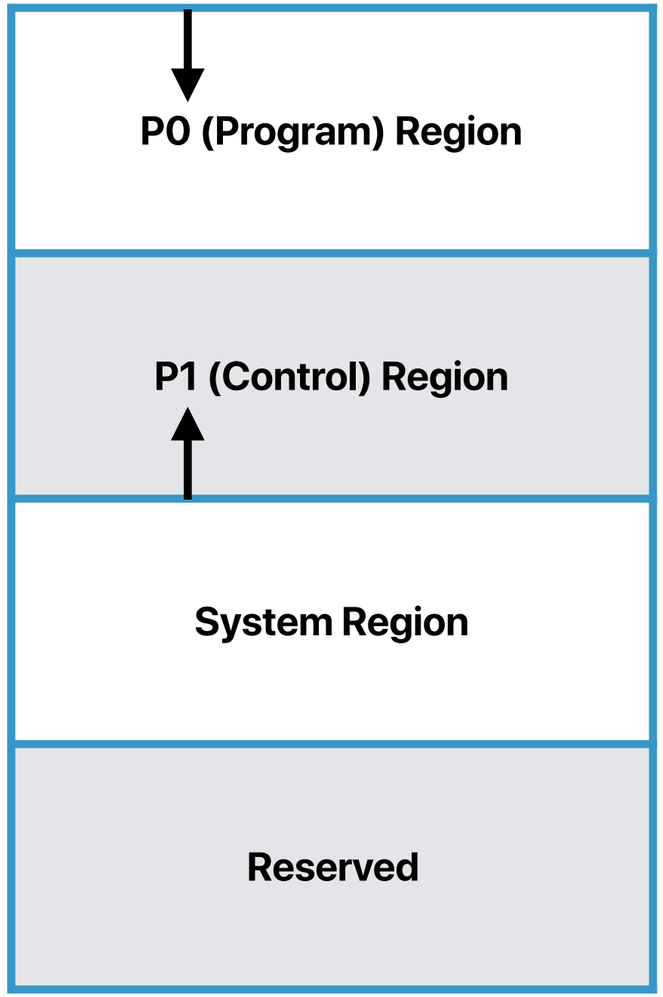
What VAX/VMS proposed to achieve these goals?

- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

Goal		Optimization	
A ✓	Process startup cost	W	Demand-zero & copy-on-refernce
B ✓	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	Page clustering
D ✓	Paging load on disks	Z	Page caching

also helps reduce disk loads

Process memory layout



Code
Heap

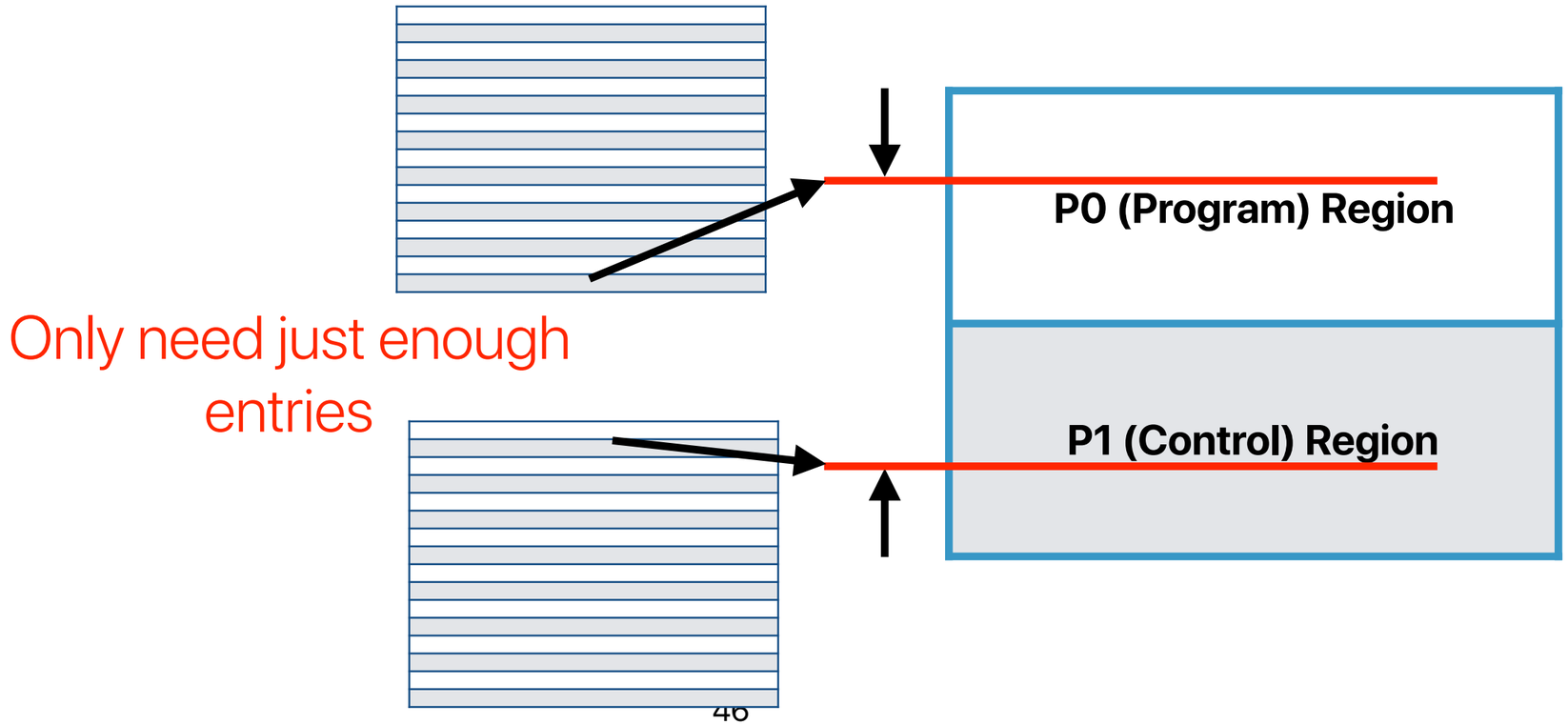
Stack
Other data

System: software vectors, hardware data structures, executive data, executive procedures, record management, dynamic storage

The VAX/VMS allows the OS code to access user-space memory

Why segmented layout?

- Each segment has its own page table
- Entries between stack and heap boundaries do not need to be allocated — reduce the size of page table



What VAX/VMS proposed to achieve these goals?

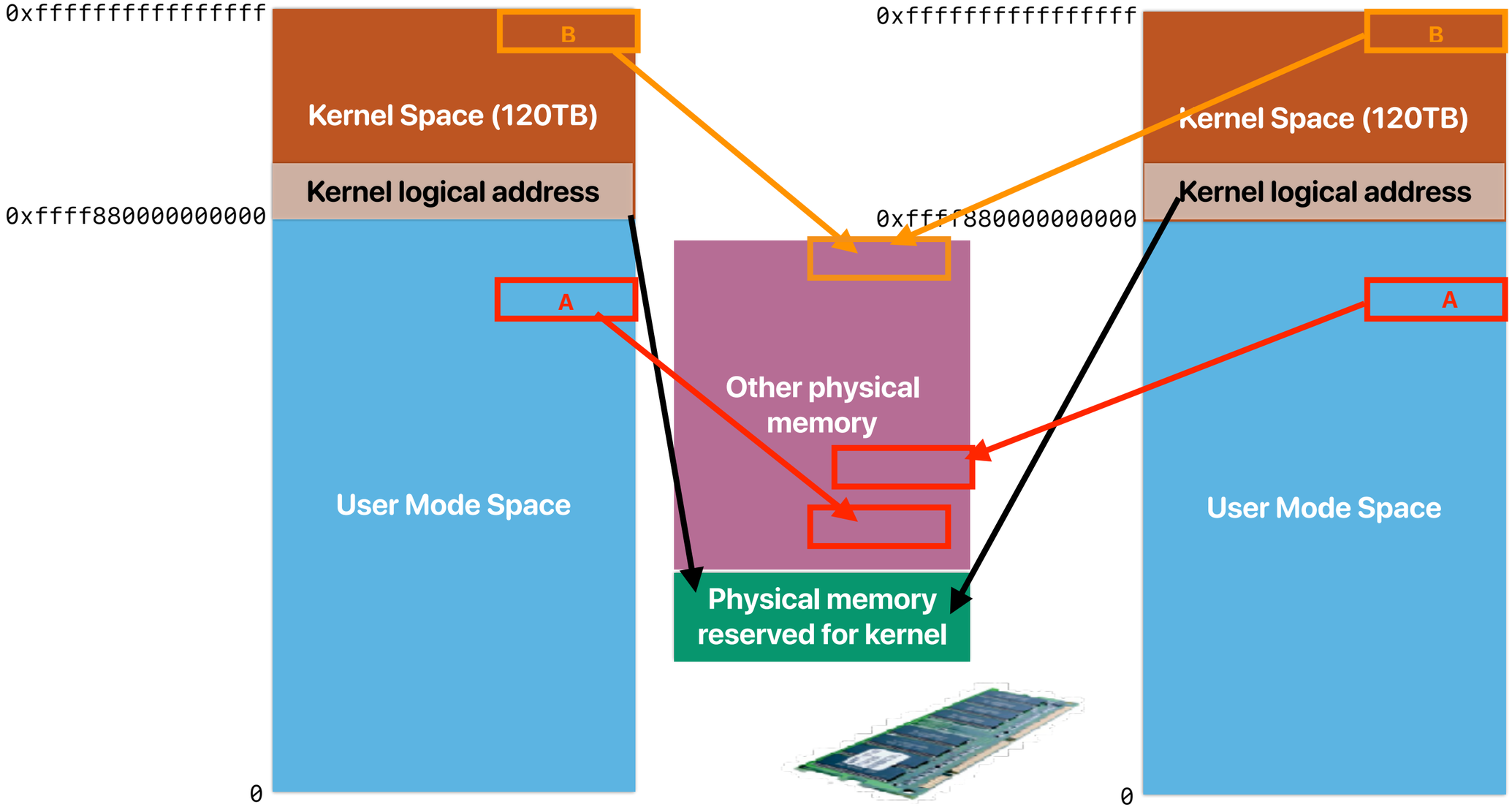
- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

Goal		Optimization	
A	Process startup cost	W	Demand-zero & copy-on-refernce
B	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	**segmented memory layout**
D	Paging load on disks	Z	Page caching

The impact of VAX/VMS

- VAX is popular in universities and UNIX is later ported to VAX — a popular OS research platform
- Affect the UNIX virtual memory design
- Affect the Windows virtual memory design

64-bit Linux process memory layout



Announcement

- Reading quiz due next Tuesday
- Project due 3/3
  - We highly recommend you to fresh install a Ubuntu 16.04.6 Desktop version within a VirtualBox
    - Virtual box is free
    - If you crash the kernel, just terminate the instance and restart virtual box
  - Use office hours to discuss projects