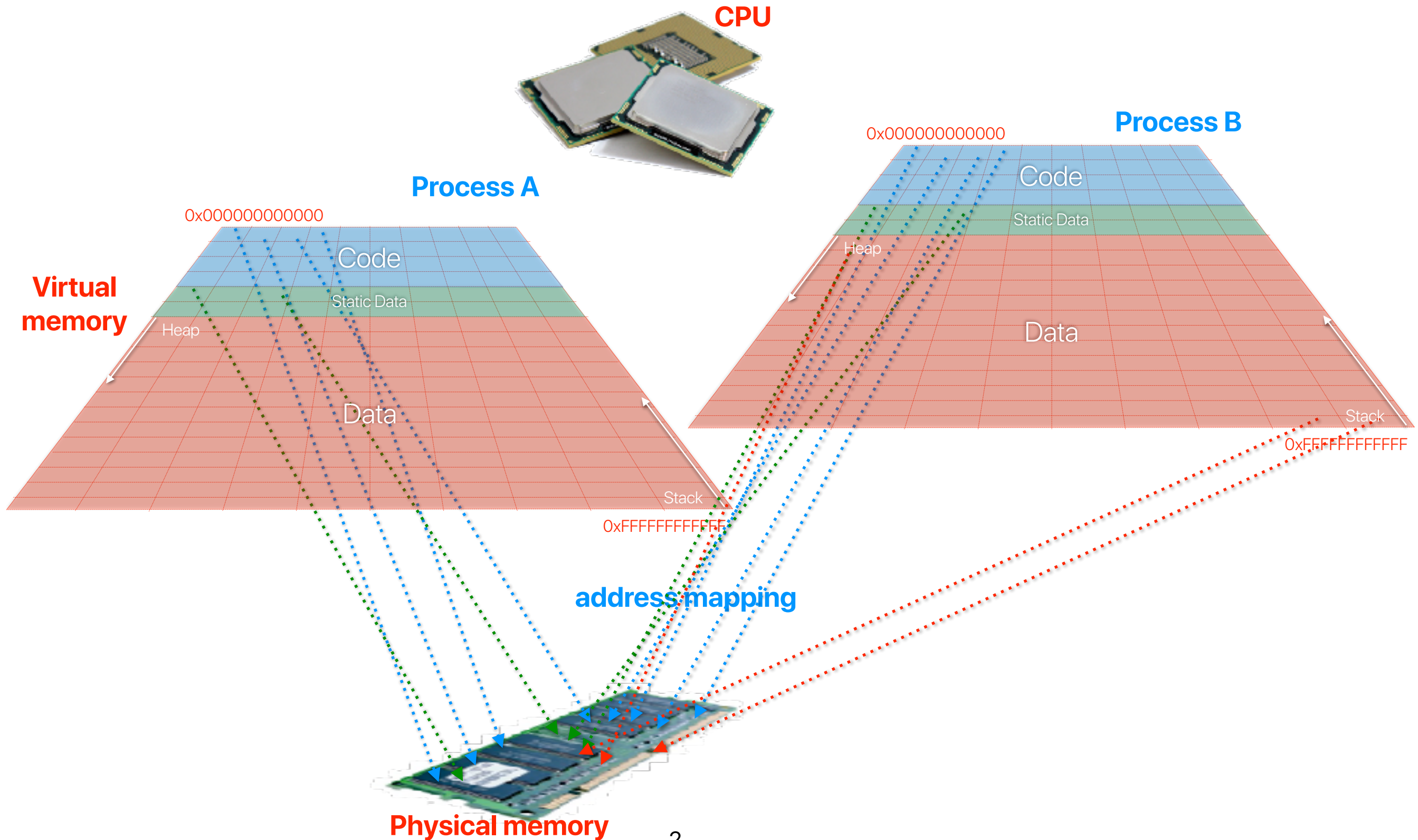


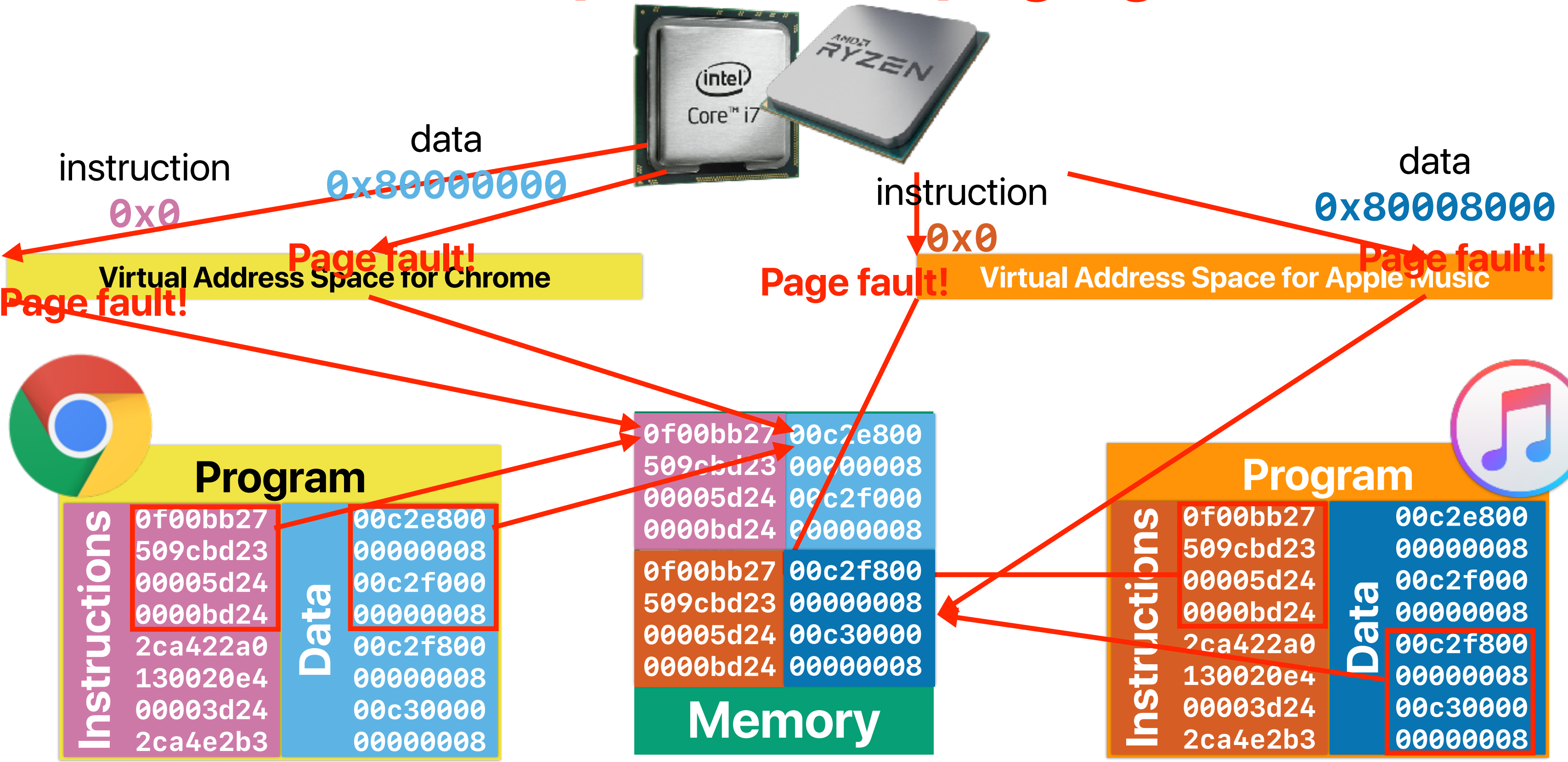
Virtual memory (III): System Architecture and Design

Hung-Wei Tseng

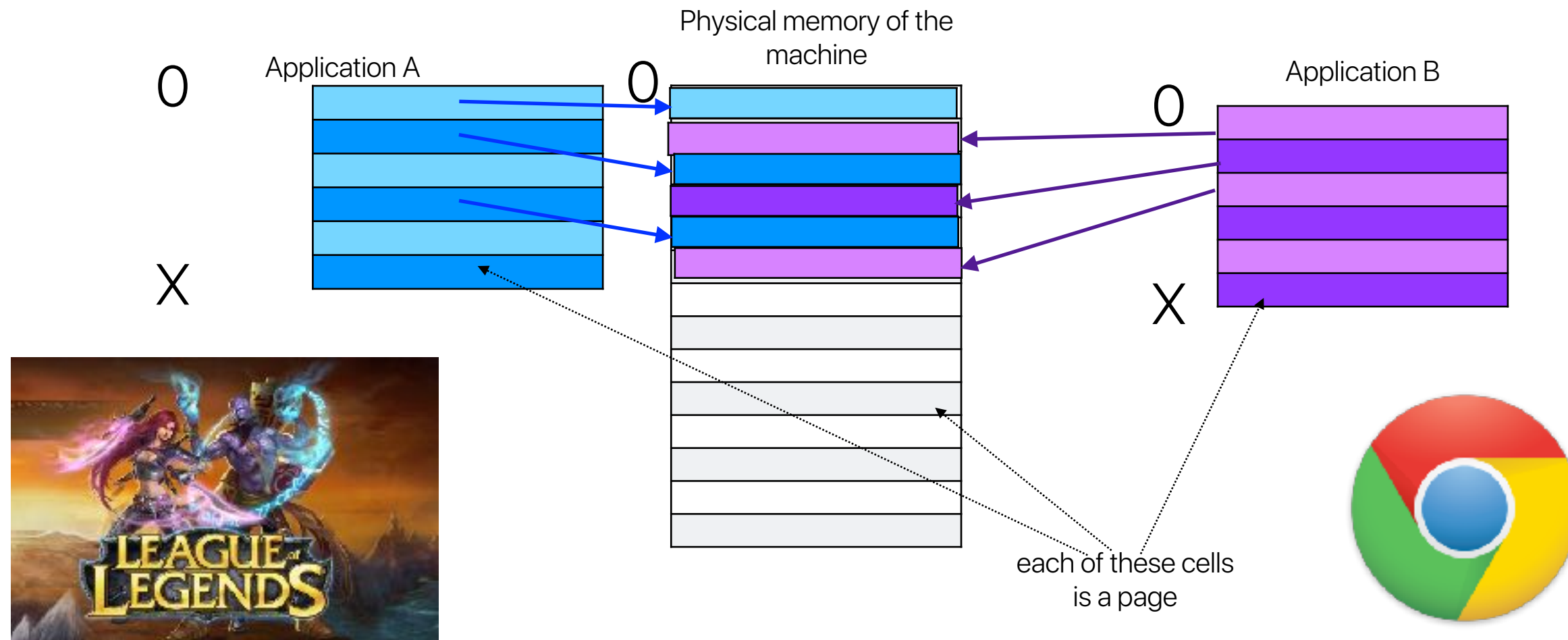
Virtual Memory



Recap: Demand paging



Recap: Demand paging



Segmentation v.s. demand paging

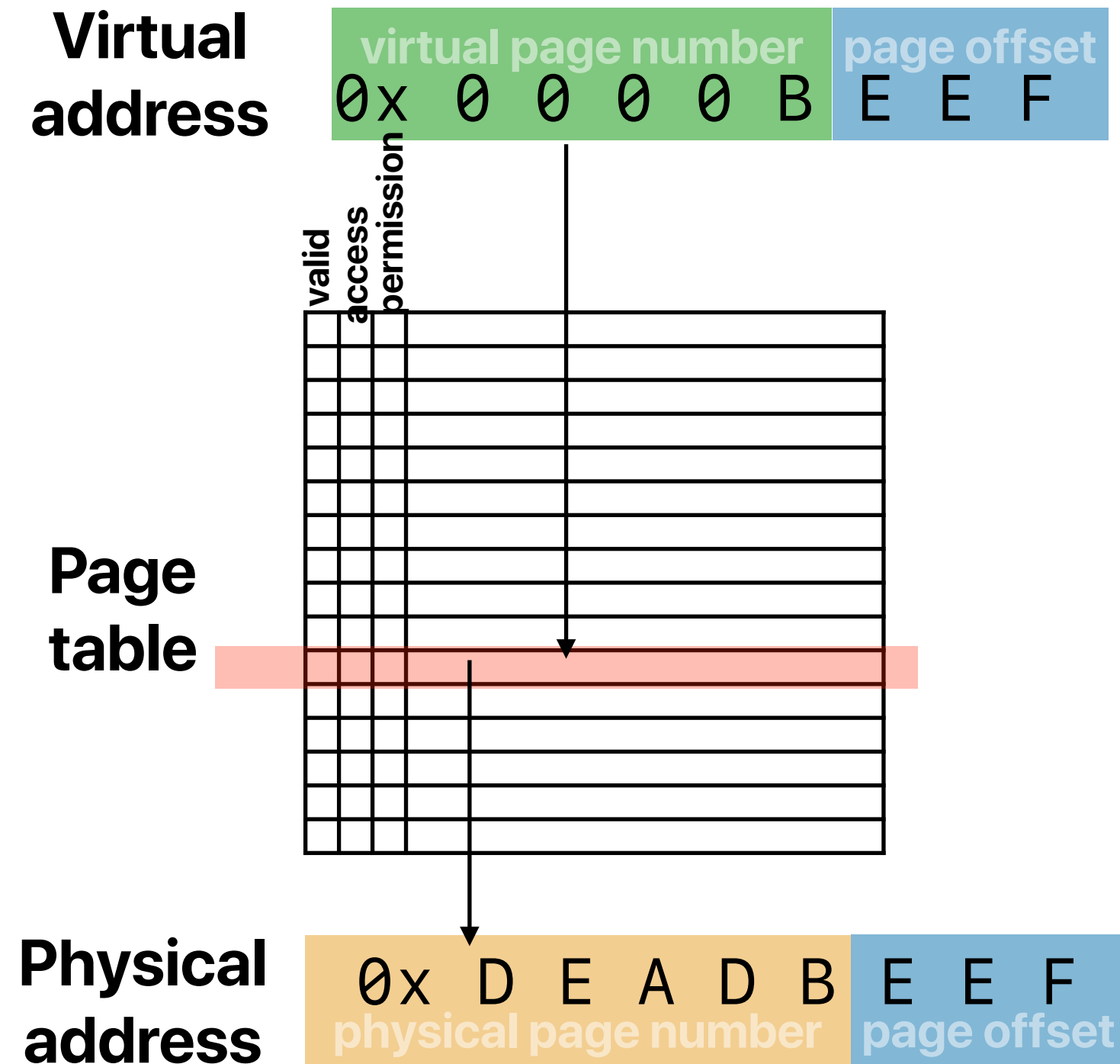
- How many of the following statements is/are correct regarding segmentation and demand paging?
 - ① Segments can cause more external fragmentations than demand paging
 - ② Paging can still cause internal fragmentations — **the main reason why we love paging!**
— **within a page**
 - ☒ ③ The overhead 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: 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



Recap: 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

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

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

Conventional page table

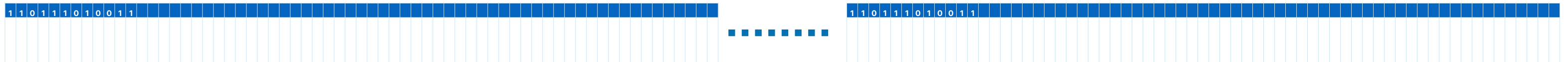
0x0

0xFFFFFFFFFFFFFFFF

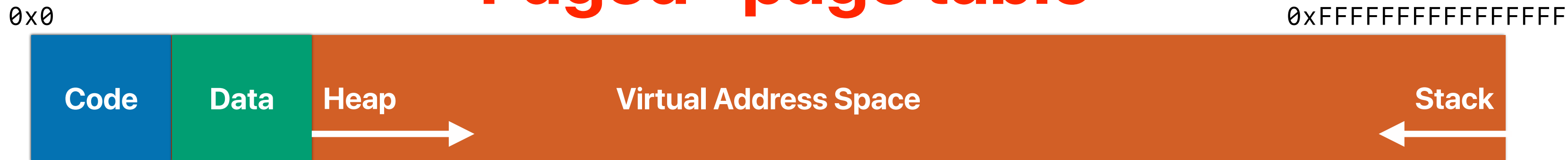
Virtual Address Space

- 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

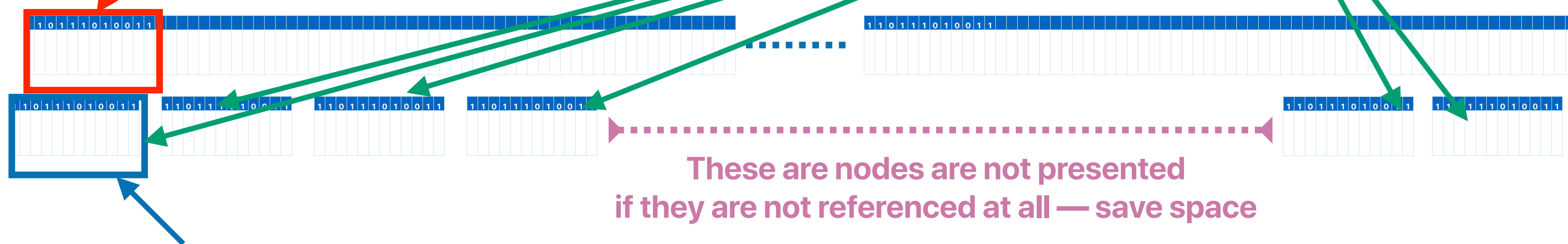


Break up entries into pages!
Each of these occupies exactly a page

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

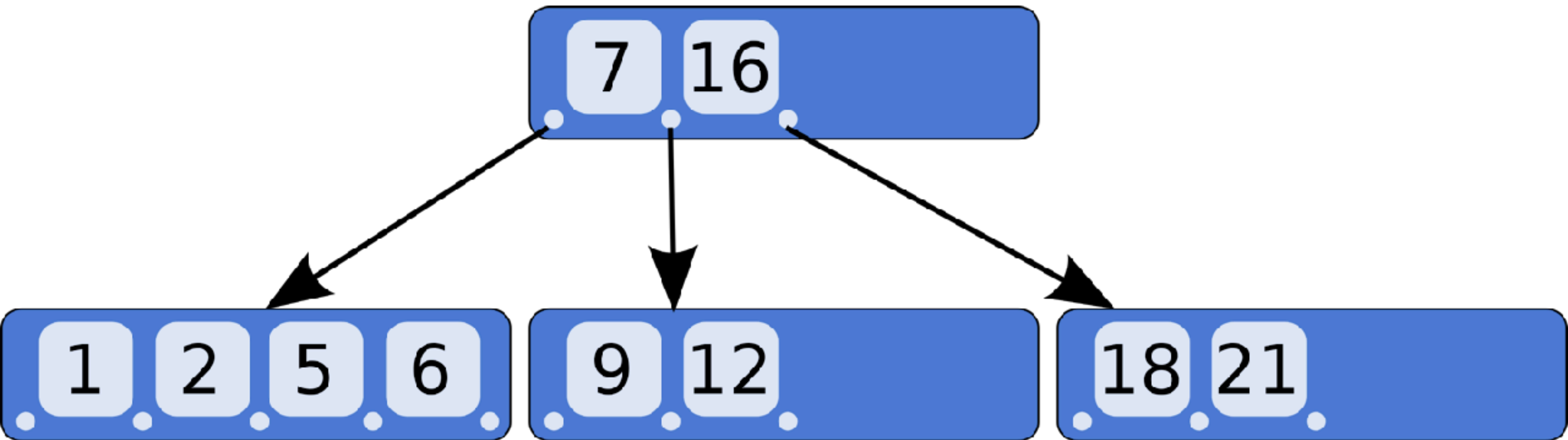
Otherwise, you always need to find more than one consecutive pages — difficult!

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



Allocate page table entry nodes "on demand"

B-tree

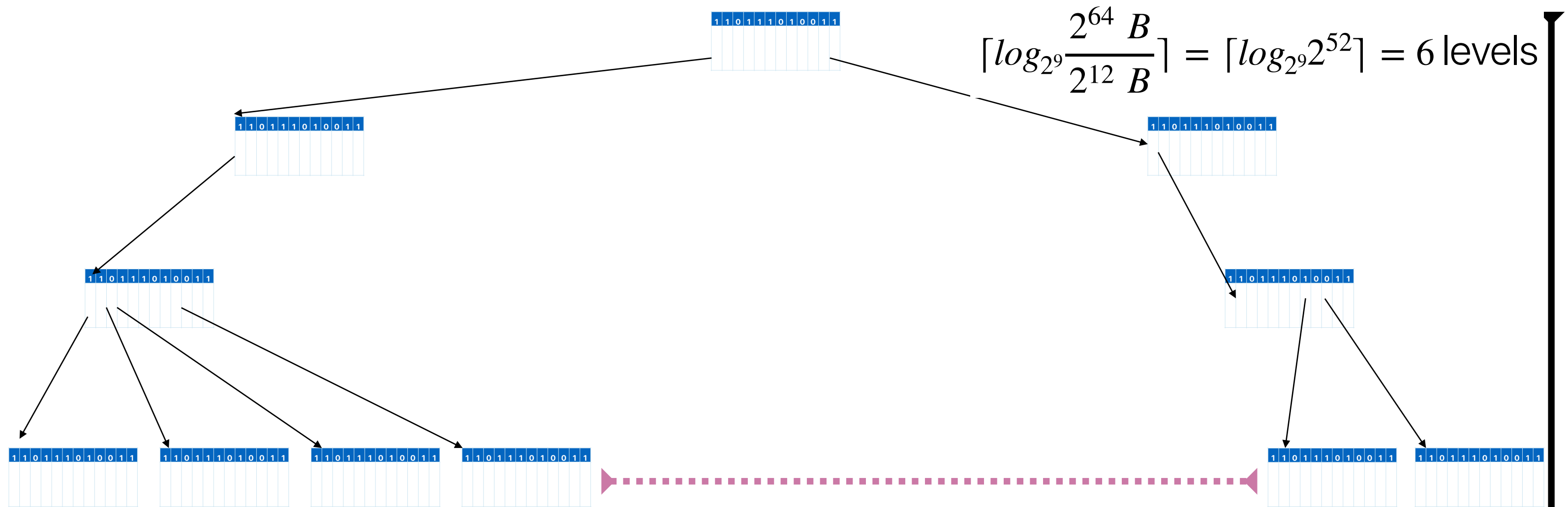
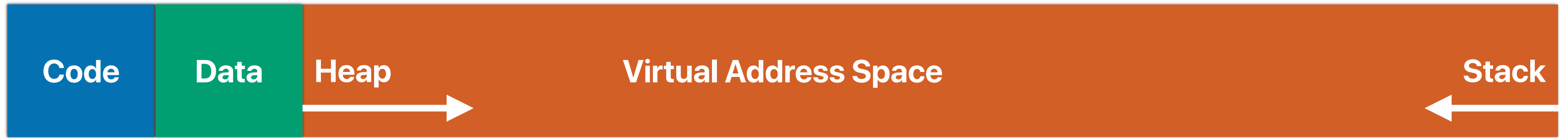


<https://en.wikipedia.org/wiki/B-tree#/media/File:B-tree.svg>

Hierarchical Page Table

0x0

0xFFFFFFFFFFFFFFFF

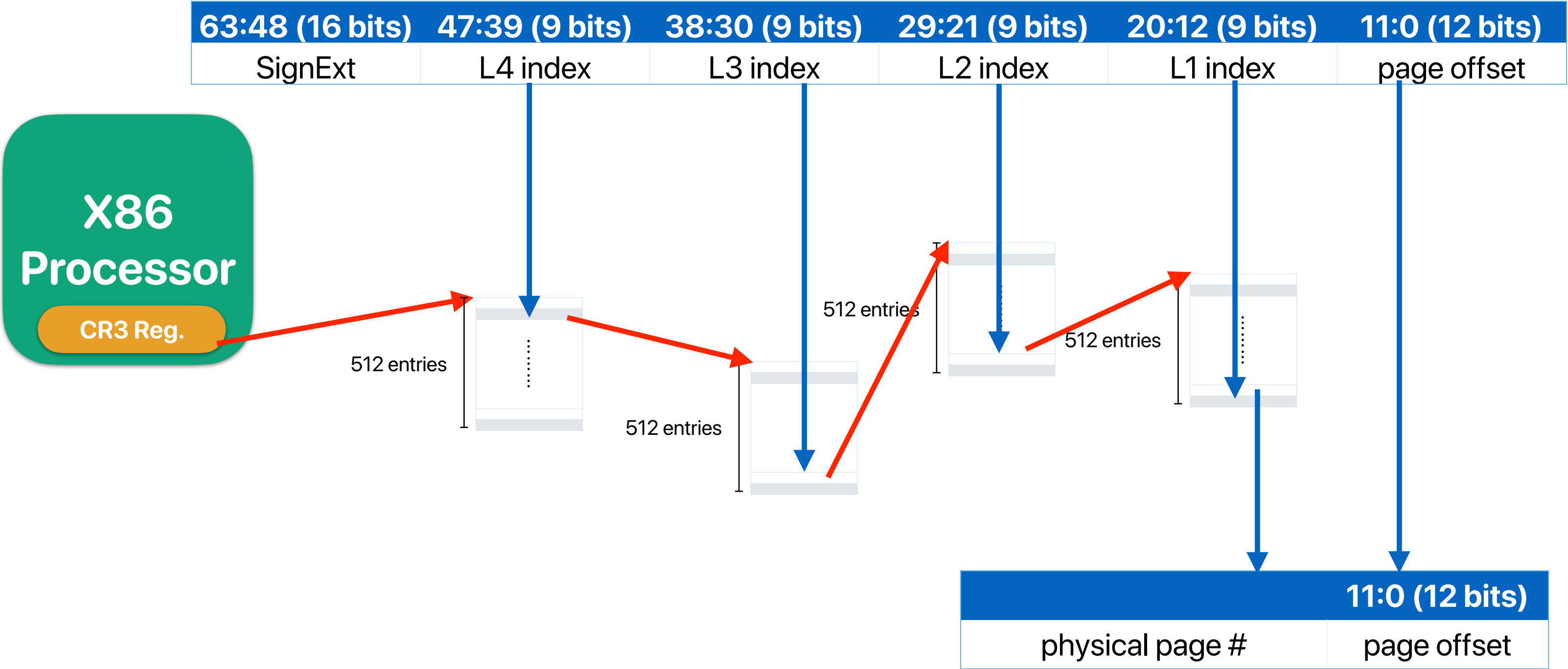


$$\lceil \log_2 \frac{2^{64} B}{2^{12} B} \rceil = \lceil \log_2 2^{52} \rceil = 6 \text{ levels}$$

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

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

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)

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

00c2f800
00000008
00c30000
00000008

?

What if my program
needs more memory?

But how about this?

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

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


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

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

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

?

and this?



Memory

Current scoreboard



Red

Blue

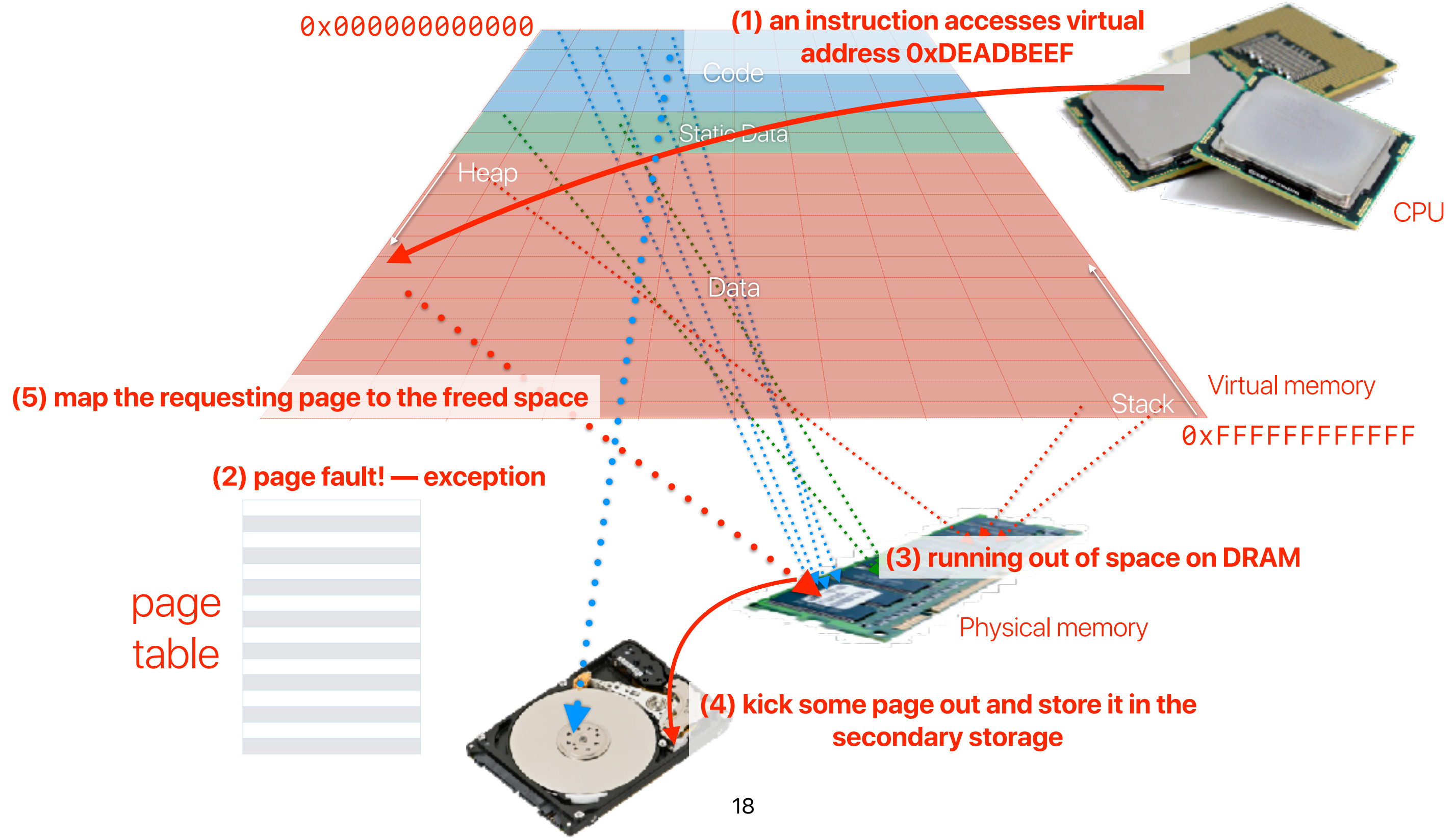
12

13

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 (2020 Version)

Operations	Latency (ns)	Latency (us)	Latency (ms)	
L1 cache reference	0.5 ns			~ 1 CPU cycle
Branch mispredict	3 ns			
L2 cache reference	4 ns			14x L1 cache
Mutex lock/unlock	17 ns			
Send 2K bytes over network	44 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	2,000 ns	2 us		
Read 1 MB sequentially from memory	3,000 ns	3 us		
Read 4K randomly from SSD*	16,000 ns	16 us		
Read 1 MB sequentially from SSD*	49,000 ns	49 us		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from disk	825,000 ns	825 us		
Disk seek	2,000,000 ns	2,000 us	2 ms	4x datacenter roundtrip
Send packet CA-Netherlands-CA	150,000,000 ns	150,000 us	150 ms	

https://colin-scott.github.io/personal_website/research/interactive_latency.html

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

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

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 = $100 \text{ ns} + P_f * 10^7 \text{ ns}$
- When $P_f = 0.001$:
Effective Access Time = 10,100ns
- When $P_f = 0.001$, even with an SSD
Effective Access Time = $100 \text{ ns} + 10^{-3} * 10^5 \text{ ns} = 200 \text{ ns}$
- 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**

Why: 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

Why: 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

The "Why" behind VAX/VMS VM

- The system needs to execute various types of applications efficiently

— Reducing the interference from heavily paging processes

- The system runs on different types of hardware

— Reducing the overhead of page tables

- 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

— Reducing the startup cost of a program

— Reducing the disk load of paging

experience with a multitude of VAX-based systems, VAX/VMS was intended to provide a single environment for all VAX-based applications, whether real-time, timeshared (including program development), or batch. In addition, VAX/VMS had to operate on a family of processors having different performance characteristics and physical memory capacities ranging from 250K to 8M bytes. To meet the requirements

time, timeshared (including program development), or batch. In addition, VAX/VMS had to operate on a family of processors having different performance characteristics and physical memory capacities ranging from 250K to 8M bytes. To meet the requirements

bytes to more than 8M bytes. To meet the requirements posed by these applications environments, the memory management system had to be capable of adjusting to the changing demands characteristic of timesharing while allowing the predictable performance required by real-time and batch processes.

Why: 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 VAX/VMS proposed to achieve these goals?

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

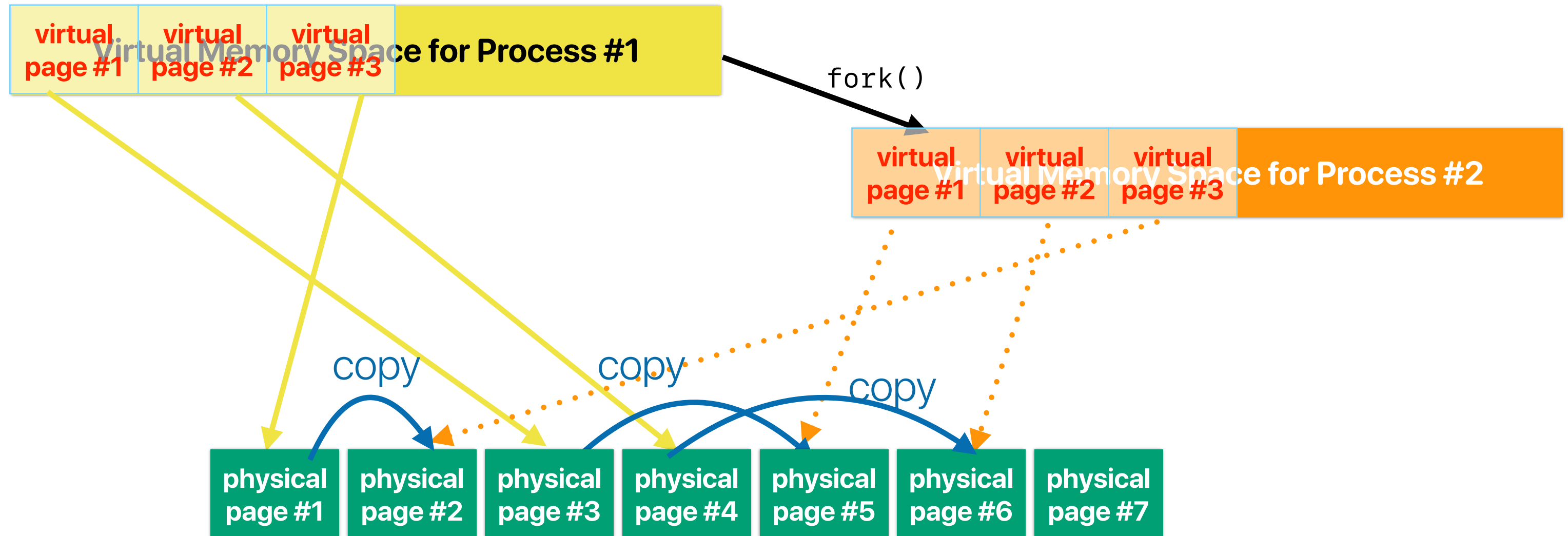
Goal		Optimization	
A	Process startup cost	W	Demand-zero & copy-on-refernce
B	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	Page clustering
D	Paging load on disks	Z	Page caching

What VAX/VMS proposed to achieve these goals?

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

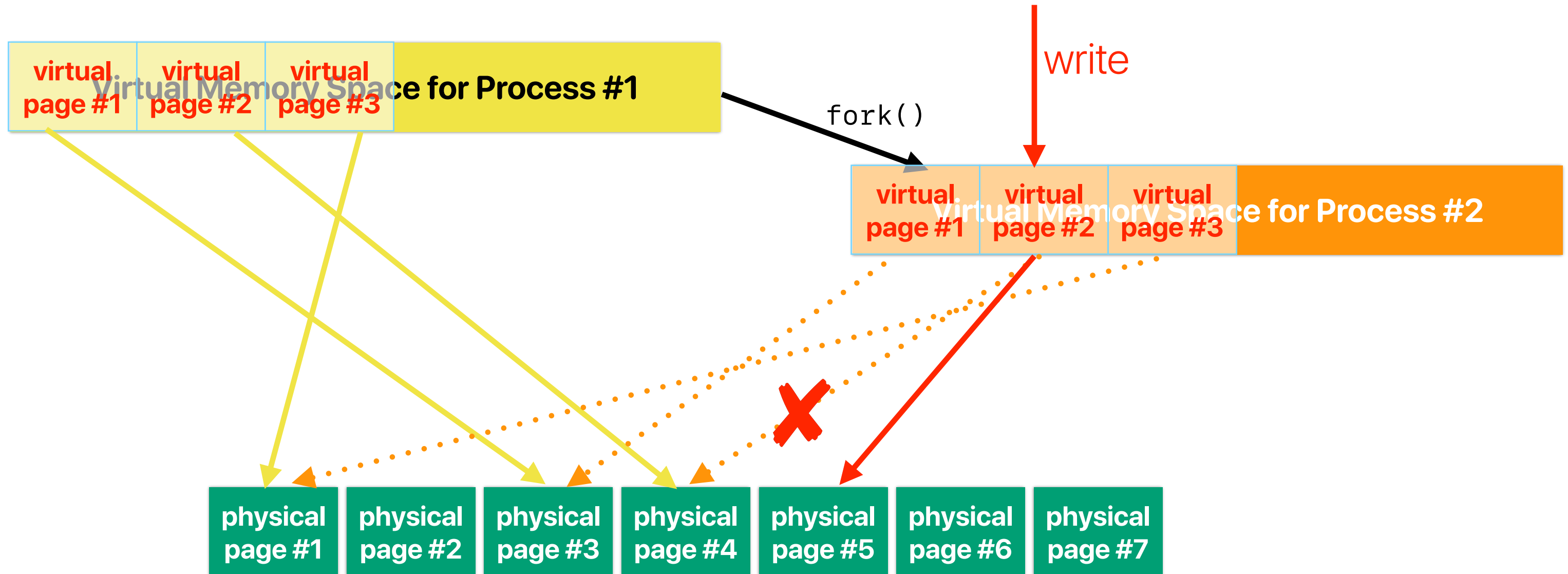
Goal		Optimization	
A	Process startup cost	W	Demand-zero & copy-on-reference
B	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	Page clustering
D	Paging load on disks	Z	Page caching

What happens on a fork?



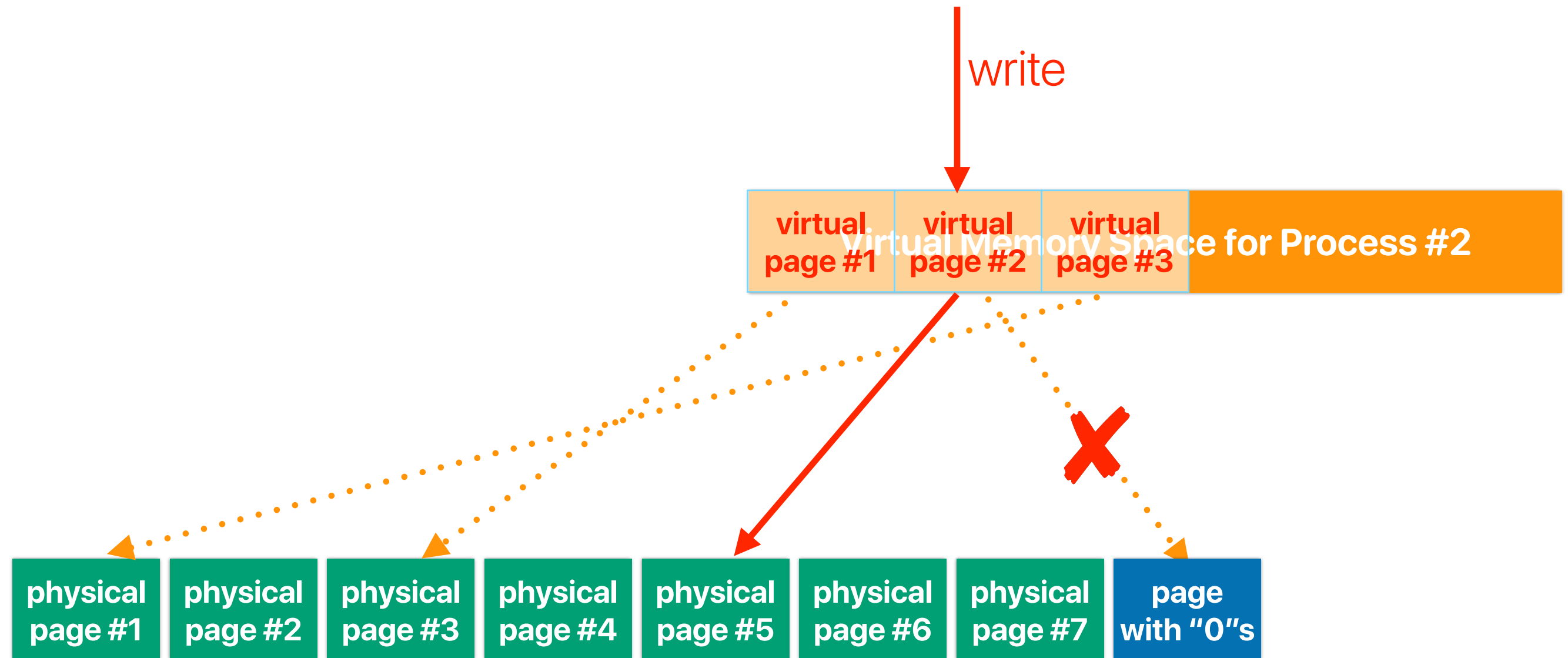
- **Copy the page content to different locations before the new process can start**

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

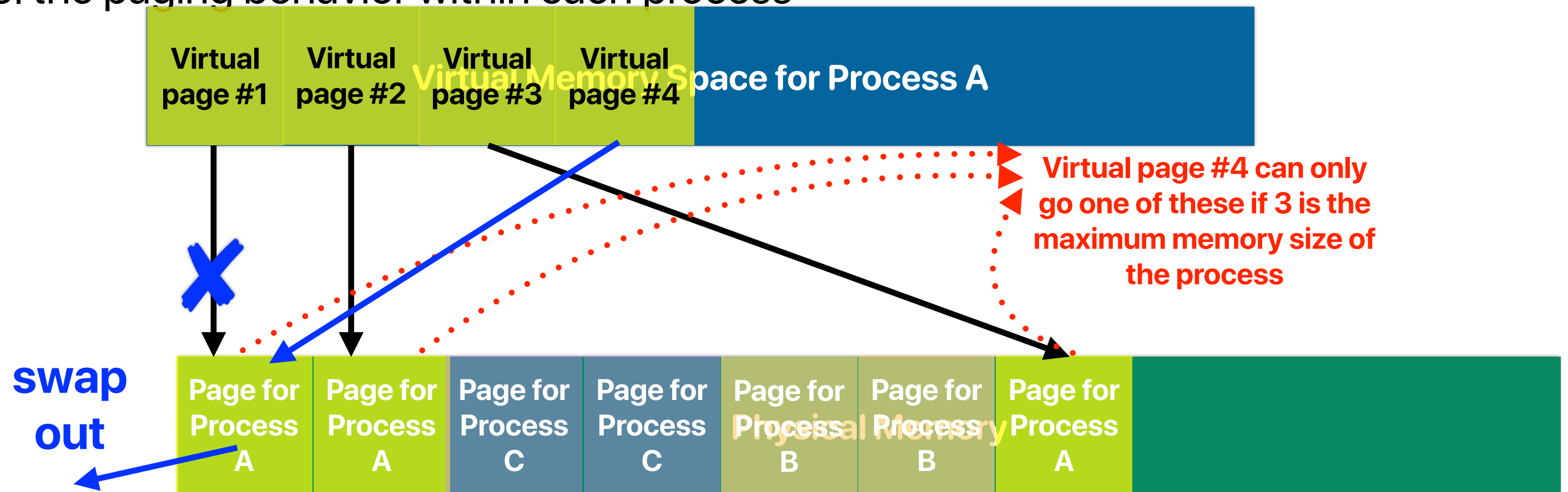
What VAX/VMS proposed to achieve these goals?

- 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-reference
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-reference
 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 (2020 Version)

Operations	Latency (ns)	Latency (us)	Latency (ms)	
L1 cache reference	0.5 ns			~ 1 CPU cycle
Branch mispredict	3 ns			
L2 cache reference	4 ns			14x L1 cache
Mutex lock/unlock	17 ns			
Send 2K bytes over network	44 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	2,000 ns	2 us		
Read 1 MB sequentially from memory	3,000 ns	3 us		
Read 4K randomly from SSD*	16,000 ns	16 us		
Read 1 MB sequentially from SSD*	49,000 ns	49 us		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from disk	825,000 ns	825 us		
Disk seek	2,000,000 ns	2,000 us	2 ms	4x datacenter roundtrip
Send packet CA-Netherlands-CA	150,000,000 ns	150,000 us	150 ms	

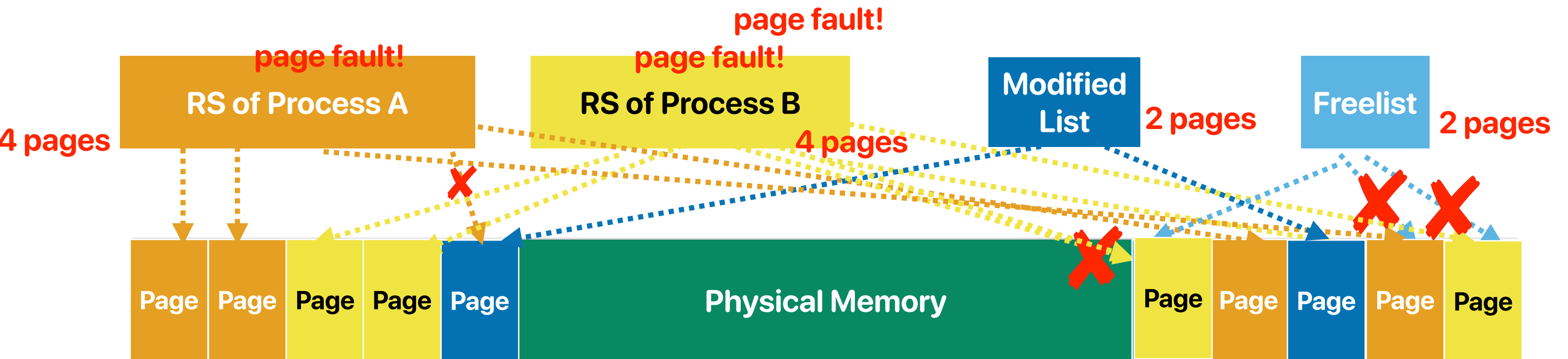
sequential access for a larger block is faster

for a 512B sector

https://colin-scott.github.io/personal_website/research/interactive_latency.html

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

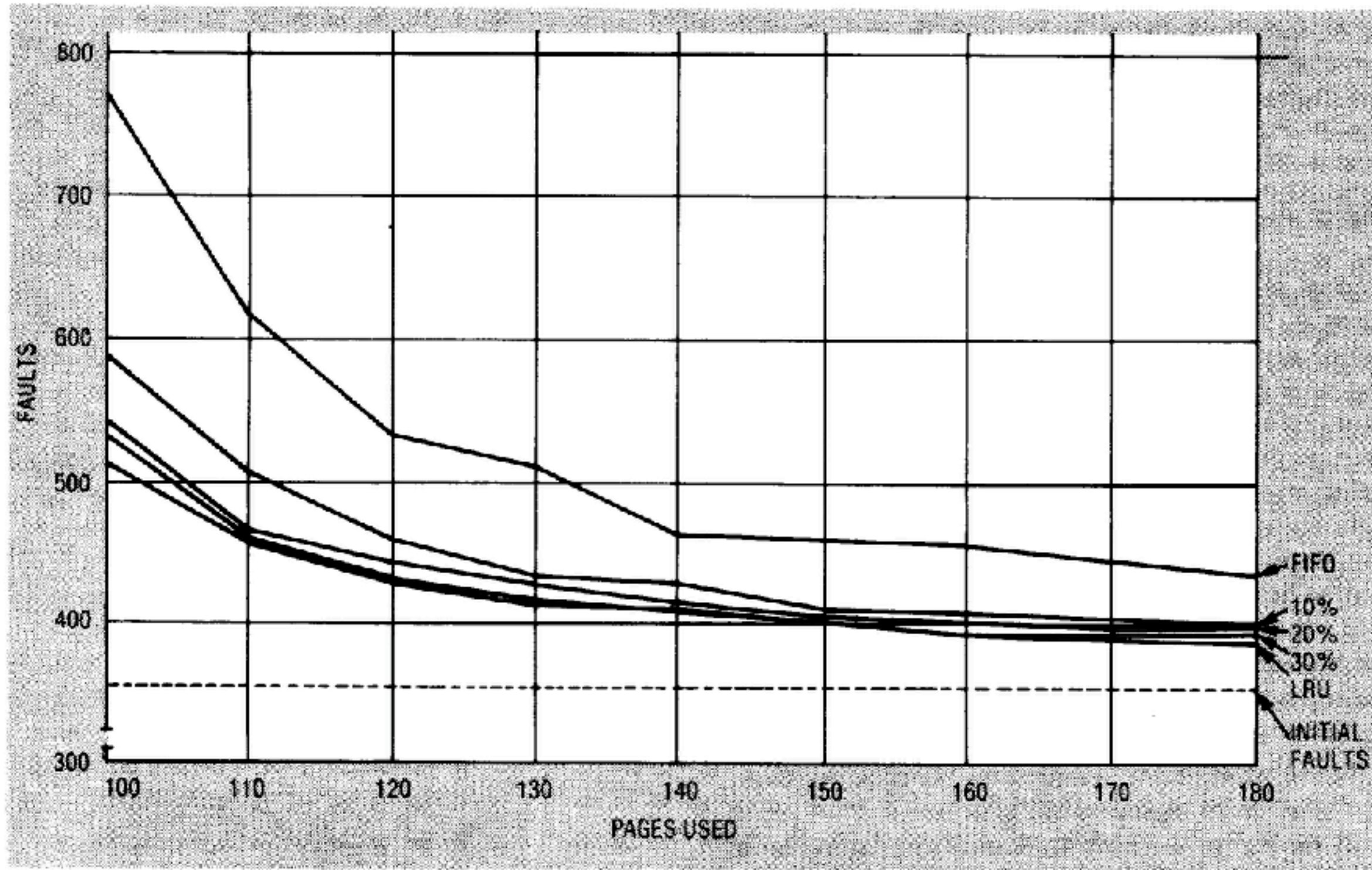





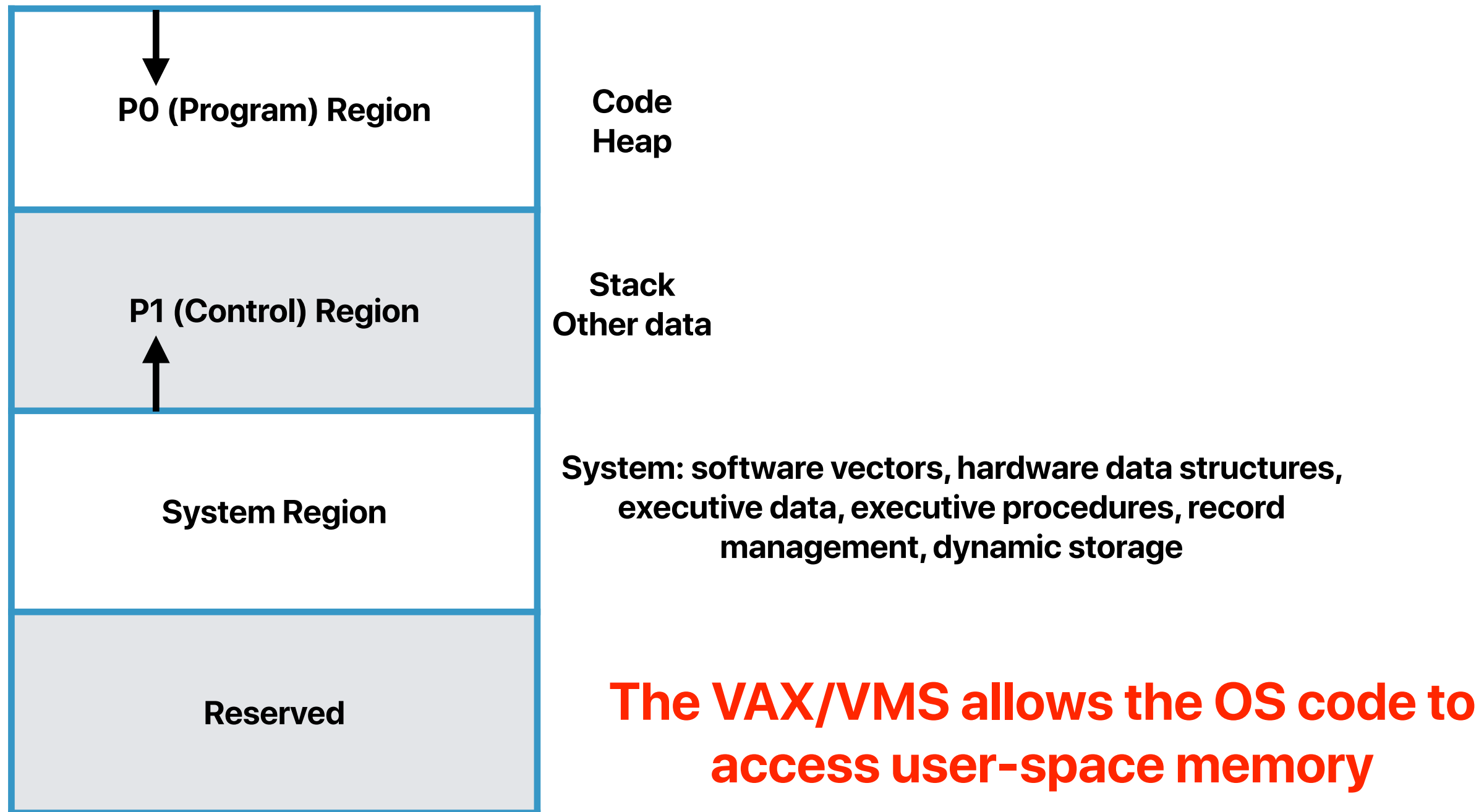
Figure 3. Faults vs. memory usage in Fortran compilation.

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?

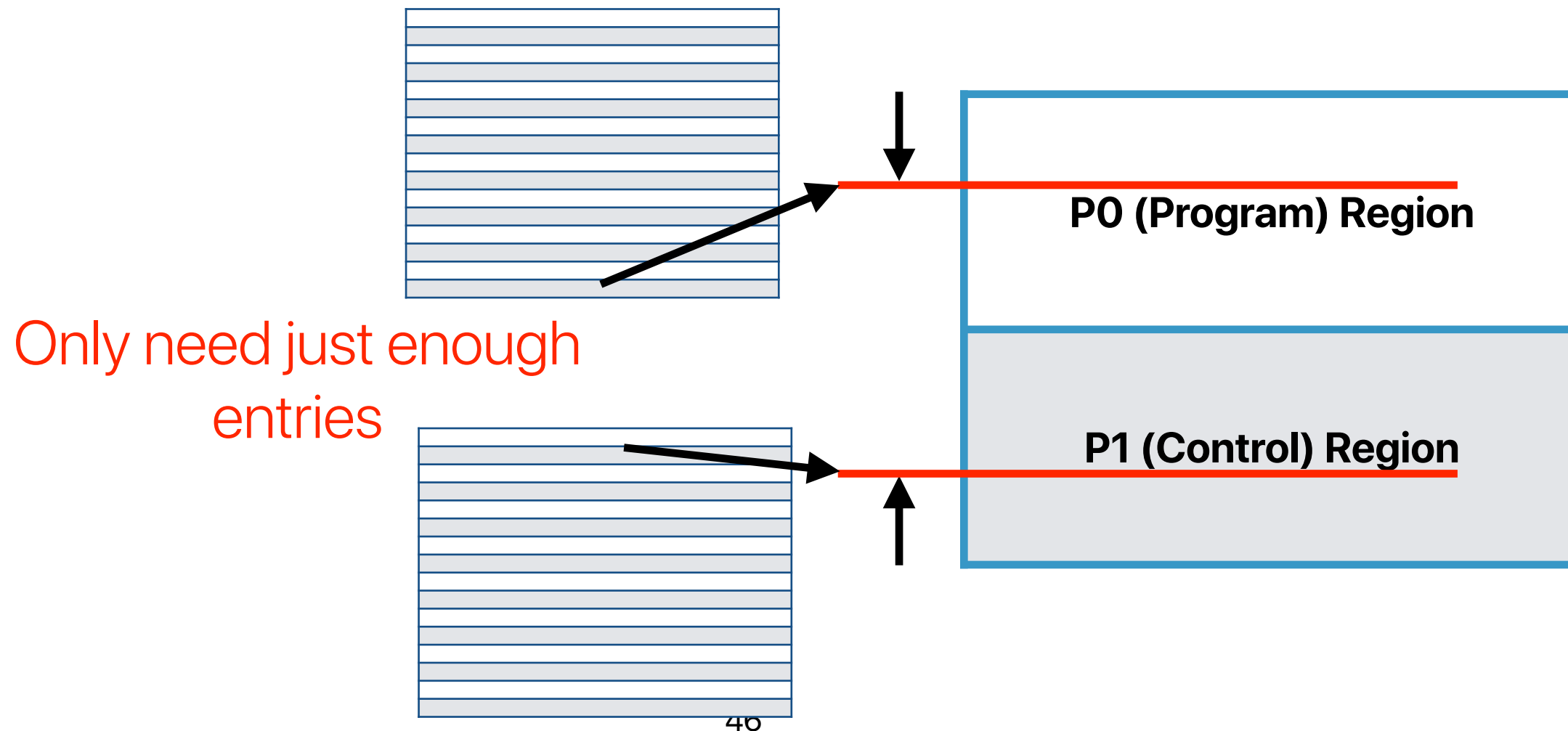
Goal		Optimization	
 A	Process startup cost	W	Demand-zero & copy-on-reference
 B	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	Page clustering also helps reduce disk loads
 D	Paging load on disks	Z	Page caching

Process memory layout



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?

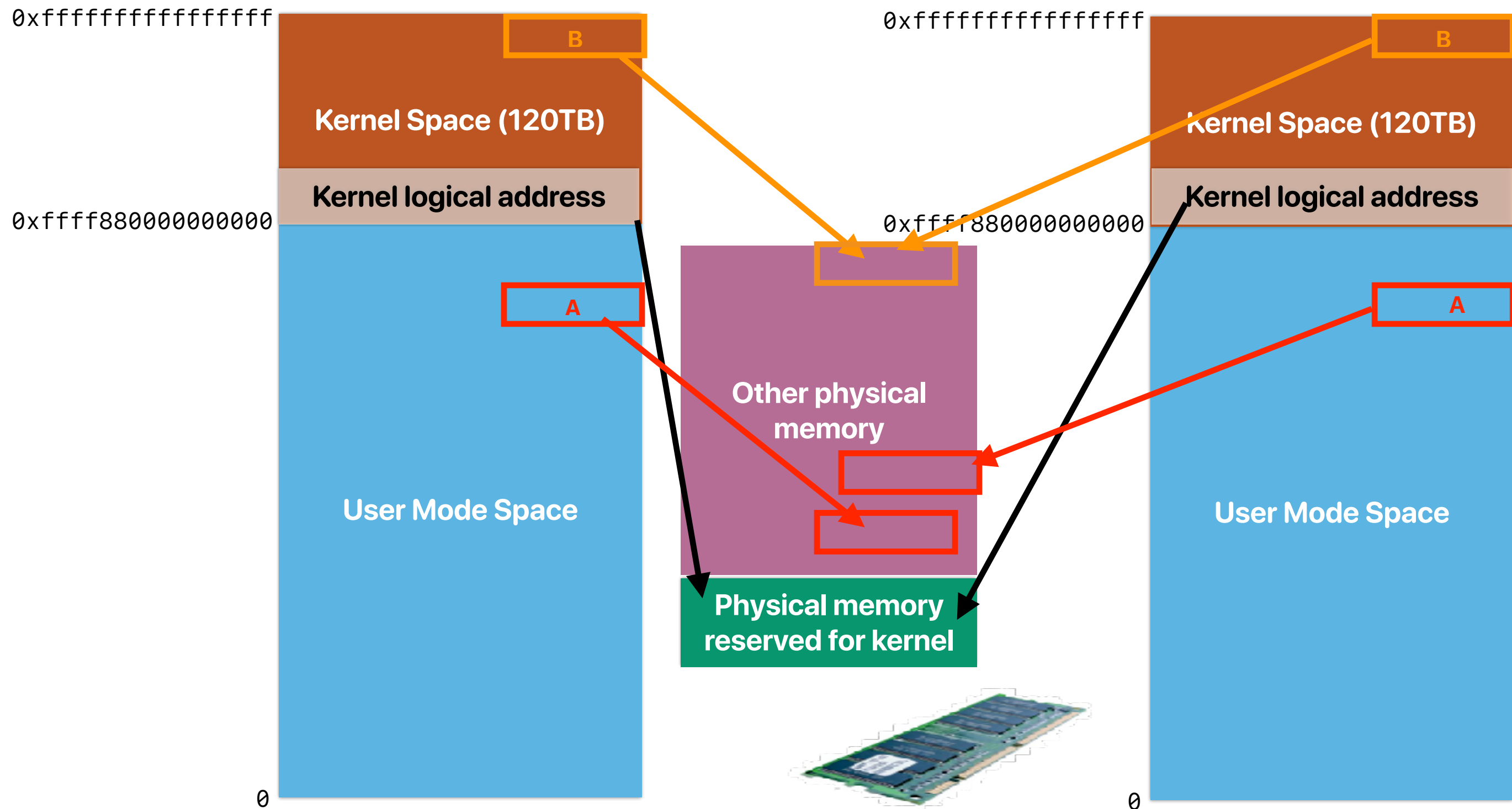
- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

Goal		Optimization	
<input checked="" type="checkbox"/> A	Process startup cost	W	Demand-zero & copy-on-reference
<input checked="" type="checkbox"/> B	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	segmented memory layout
<input checked="" type="checkbox"/> 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



Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures

**Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baron, David Black,
William Bolosky, and Jonathan Chew**

Carnegie-Mellon University, NeXT, University of Rochester

Mach abstractions

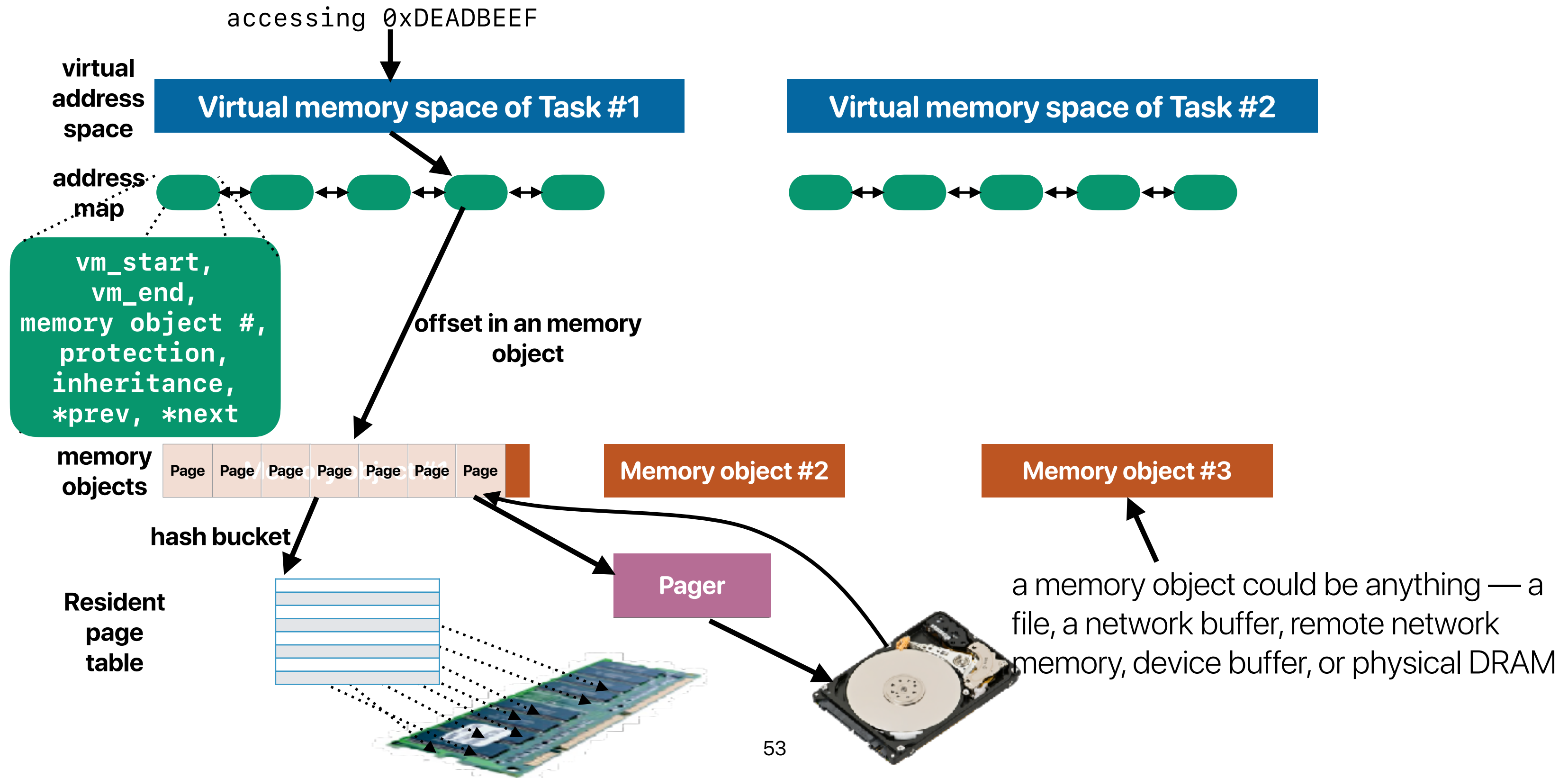
- Task: process in UNIX
- Thread: the basic scheduling identity
- Port: message queues protected by the kernel
- Message: data objects for inter-thread communication
- Memory object: data mapped into the address space of a task/
process

We mentioned previously

What Mach VM proposed?

- Machine-independent virtual memory design by maintaining all VM state in a machine-independent module
- Treat hardware page tables/TLBs as caches of machine-independent information

Overview of Mach's VM



Where is pmap?

- Pmap is just a **cache** of virtual to physical address mapping
- It accelerates address translation by caching the address mapping, but not required
- As a result, it can be as small as several KBs

The impact of Mach VM

- MacOS X uses a “hybrid” kernel — BSD + Mach
- The kernel itself is BSD-based — modular, not microkernel-based
- MacOS X's virtual memory resembles the Mach VM design
 - Why?

VAX v.s. Mach

- MacOS does not adopt the microkernel idea from Mach but takes Mach's VMS design instead of a VAX/UNIX style one. Why?
 - ① Mach's VM would provide better average memory access latency
 - ② Mach's VM would make the page table more efficient for sparse address allocations
 - ③ Mach's VM would make the process creation more efficient
 - ④ Mach's VM would be less dependent on hardware architecture
- A. 0
B. 1
C. 2
D. 3
E. 4

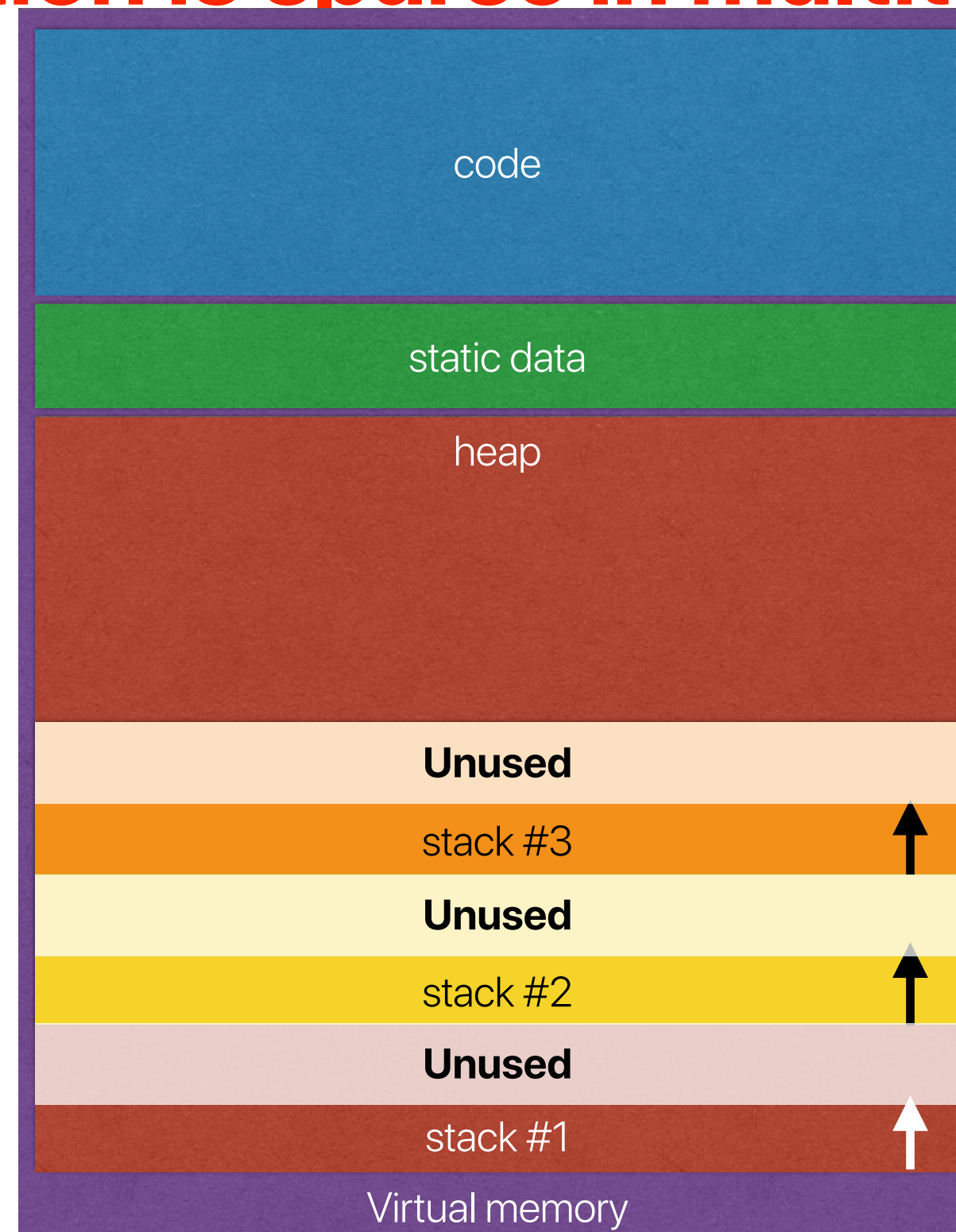
VAX v.s. Mach

- MacOS does not adopt the microkernel idea from Mach but takes Mach's VMS design instead of a VAX/UNIX style one. Why?
 - ① Mach's VM would provide better average memory access latency
 - ② Mach's VM would make the page table more efficient for sparse address allocations
 - ③ Mach's VM would make the process creation more efficient
 - ④ Mach's VM would be less dependent on hardware architecture
- A. 0
B. 1
C. 2
D. 3
E. 4

VAX v.s. Mach

- MacOS does not adopt the microkernel idea from Mach but takes Mach's VMS design instead of a VAX/UNIX style one. Why?
 - ❌ ① Mach's VM would provide better average memory access latency
— both of them uses FIFO by default + Mach has more context switches
 - ② Mach's VM would make the page table more efficient for sparse address allocations
— think about it's linked-list nature
 - ③ Mach's VM would make the process creation more efficient
 - ④ Mach's VM would be less dependent on hardware architecture
- A. 0
B. 1
C. 2
D. 3
E. 4

Address allocation is sparse in multithreading model!



VAX v.s. Mach

- MacOS does not adopt the microkernel idea from Mach but takes Mach's VMS design instead of a VAX/UNIX style one. Why?
 - ☒ ① Mach's VM would provide better average memory access latency
 - both of them uses FIFO by default + Mach has more context switches
 - ② Mach's VM would make the page table more efficient for sparse address allocations
 - think about it's linked-list nature
 - ☒ ③ Mach's VM would make the process creation more efficient
 - what's the benefit? — multithreading!
 - both of them uses copy-on-reference...
 - ④ Mach's VM would be less dependent on hardware architecture
 - what's the title of the paper?
- A. 0
B. 1
C. 2
D. 3
E. 4

Announcement

- Reading quizzes due next Tuesday
- New office hour
 - M 3p-4p and Th 9a-10a
 - Use the office hour Zoom link, not the lecture one
- Project released
 - Groups in 2
 - **Pull the latest version — had some changes for later kernel versions**
<https://github.com/hungweitseng/CS202-ResourceContainer>
 - **Install an Ubuntu Linux 16.04.07 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 2/15/2021 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
 - **No late submission is allowed**

Computer Science & Engineering

202

つづく

