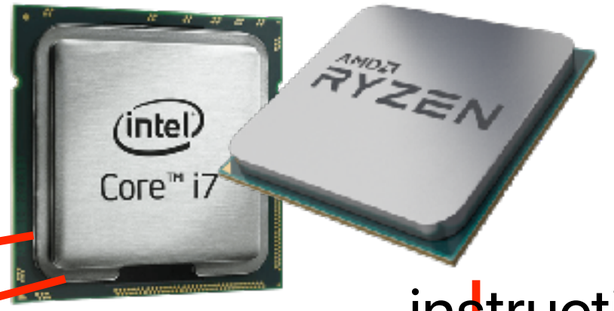
Virtual memory (II): System Architecture and Design

Hung-Wei Tseng

Recap: 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

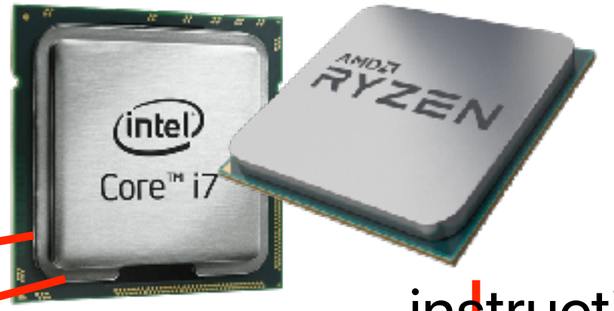
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
0f00bb27	00c2f800
509cbd23	00000008
00005d24	00c30000
0000bd24	00000008

Memory

Program

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

Recap: 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

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: Segmentation v.s. demand paging

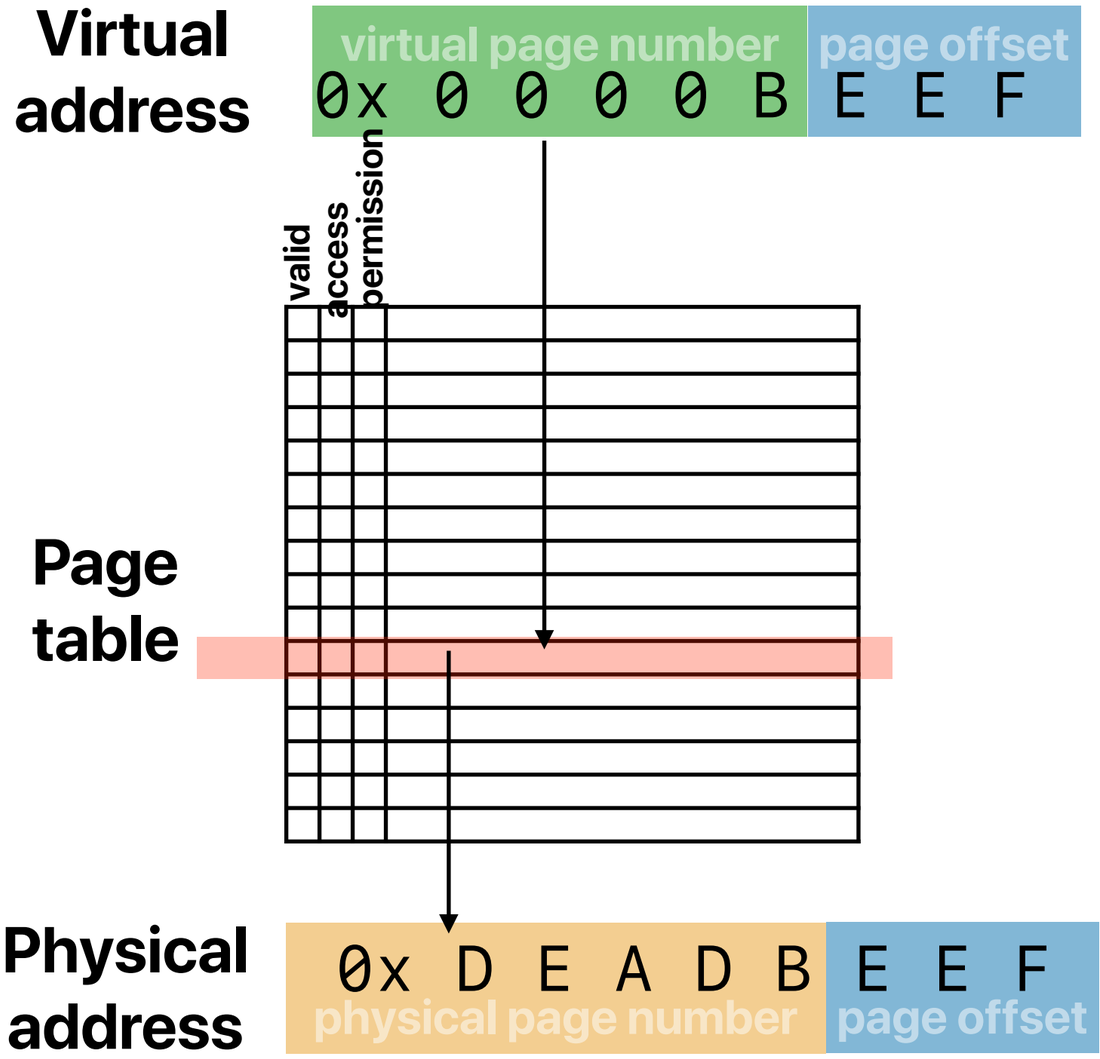
- How many of the following statements is/are correct regarding segmentation and demand paging?
  - ① Segments can cause more external fragmentations than demand paging — the main reason why we love paging!
  - ② Paging can still cause internal fragmentations — within a page
  - ~~③~~ The overhead of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

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



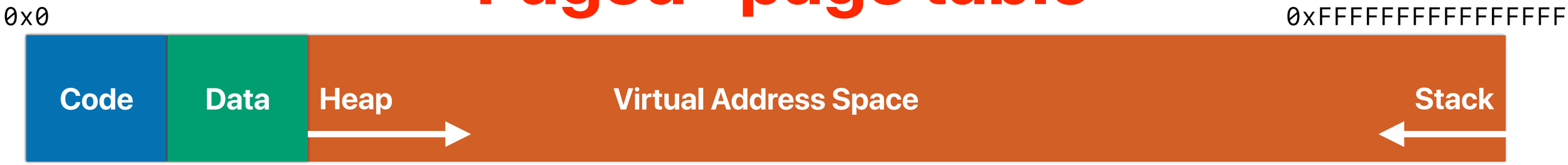
Recap: Size of page table

- 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

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

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

"Paged" page table

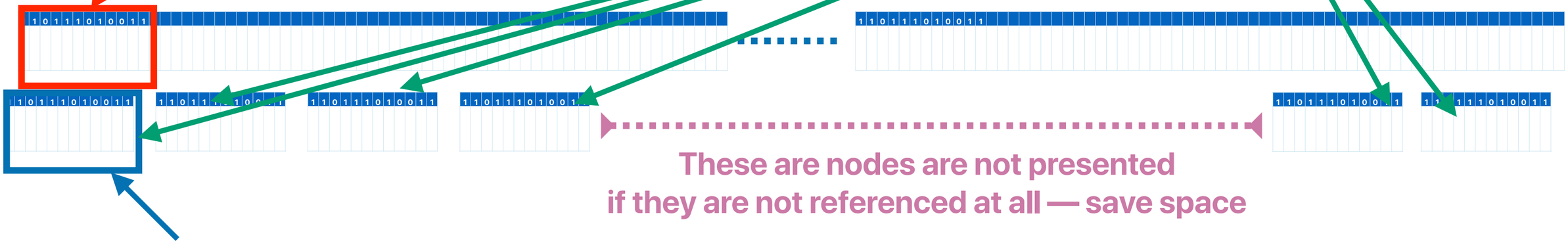


Break up entries into pages!
Each of these occupies exactly a page

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

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

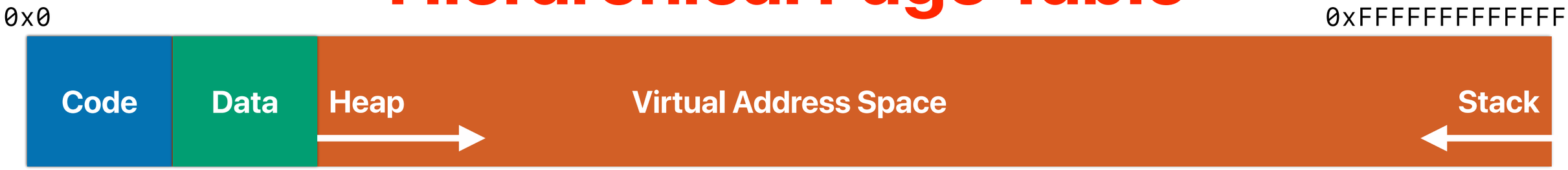
Otherwise, you always need to find more than one consecutive pages — difficult!



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

Allocate page table entry nodes "on demand"

Hierarchical Page Table



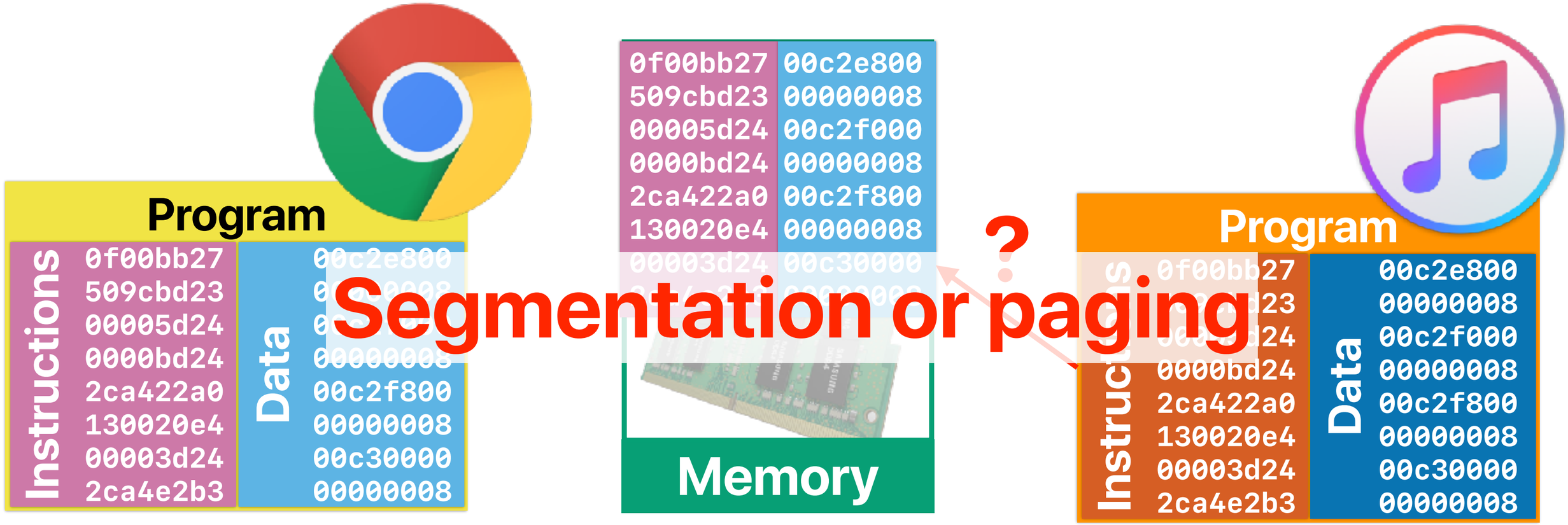
$$\lceil log_{2^9} \frac{2^{64}\ B}{2^{12}\ B} \rceil = \lceil log_{2^9} 2^{52} \rceil = 6 \text{ levels}$$

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)

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

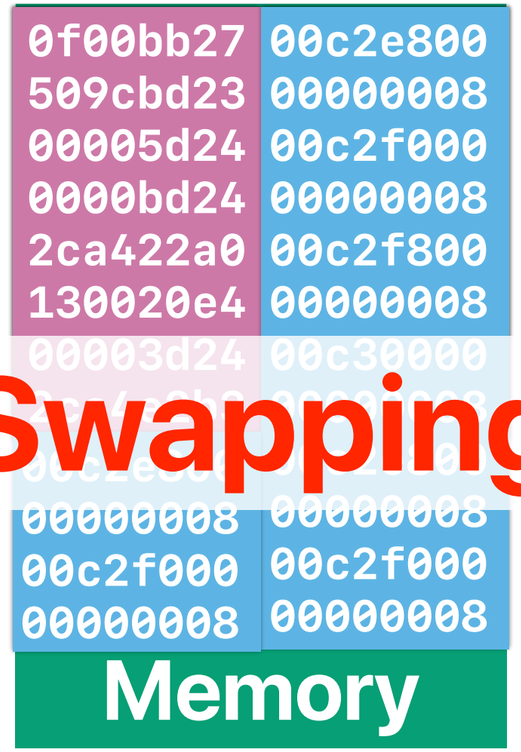
What if both programs need to use memory?



If we expose memory directly to the processor (I)

?

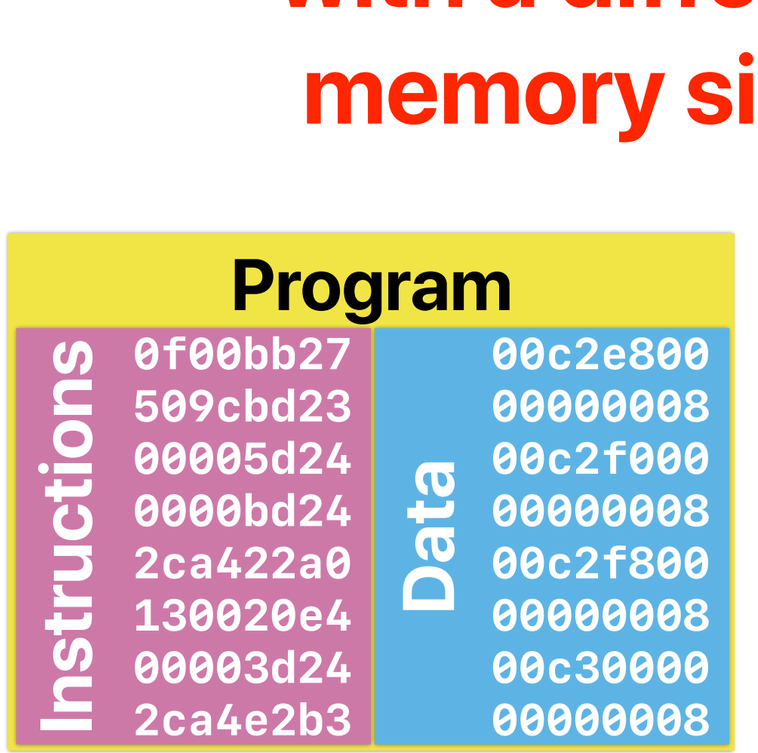
What if my program needs more memory?

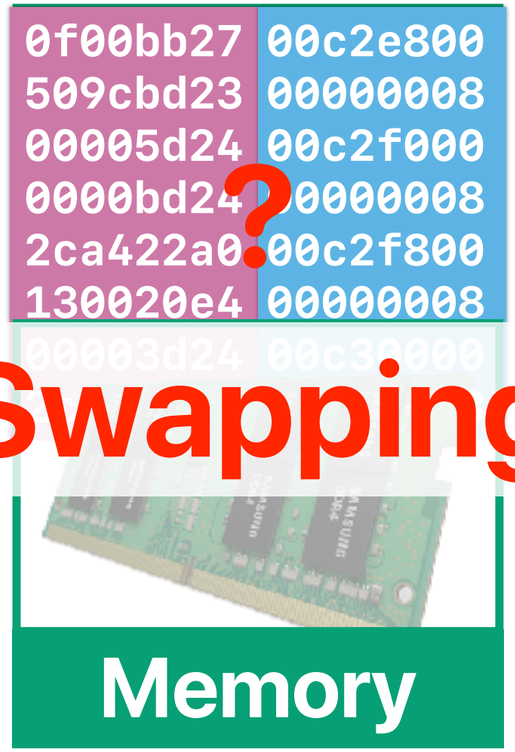


Swapping

If we expose memory directly to the processor (II)

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

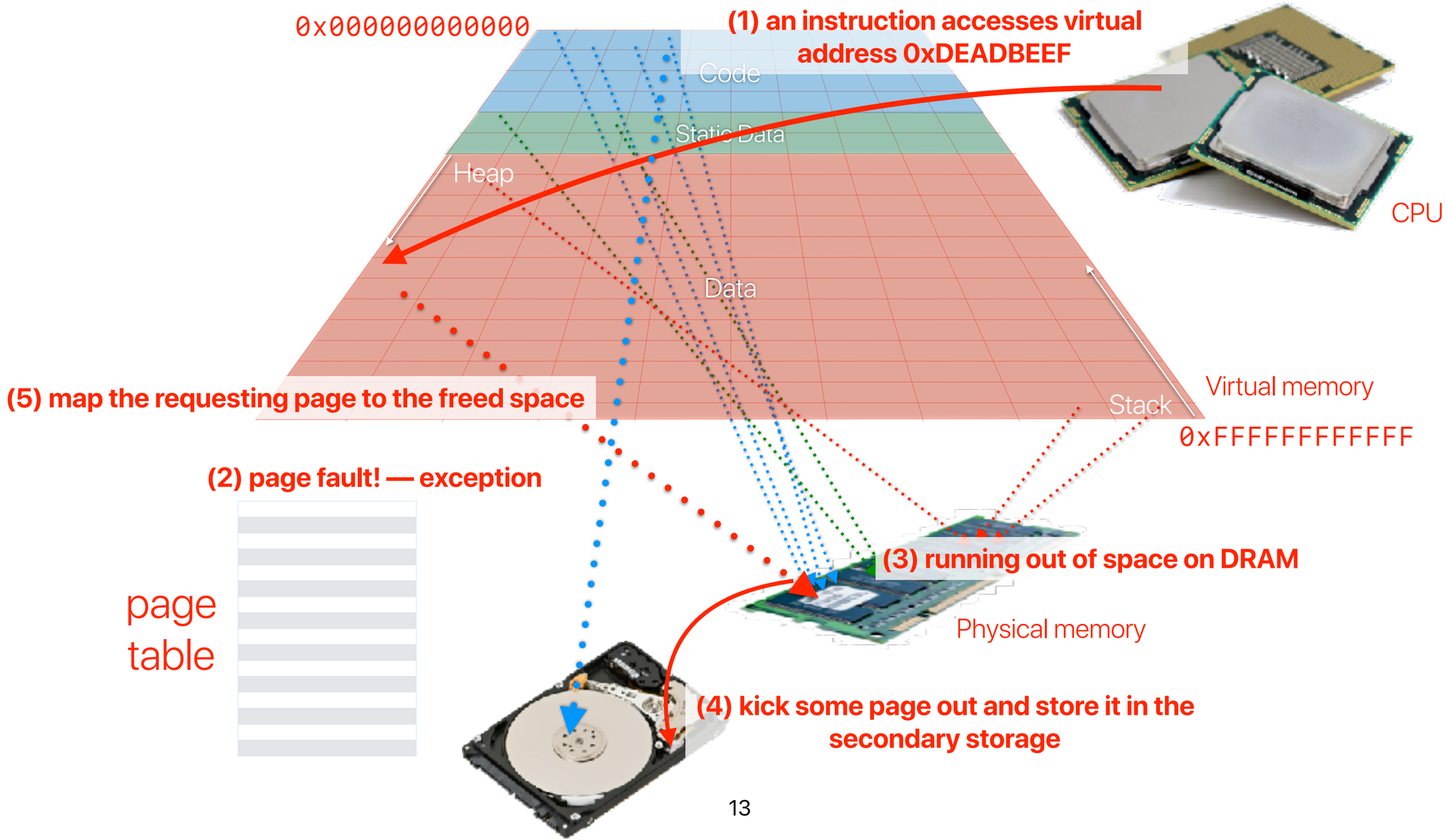




Outline

- 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 (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)
  - 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

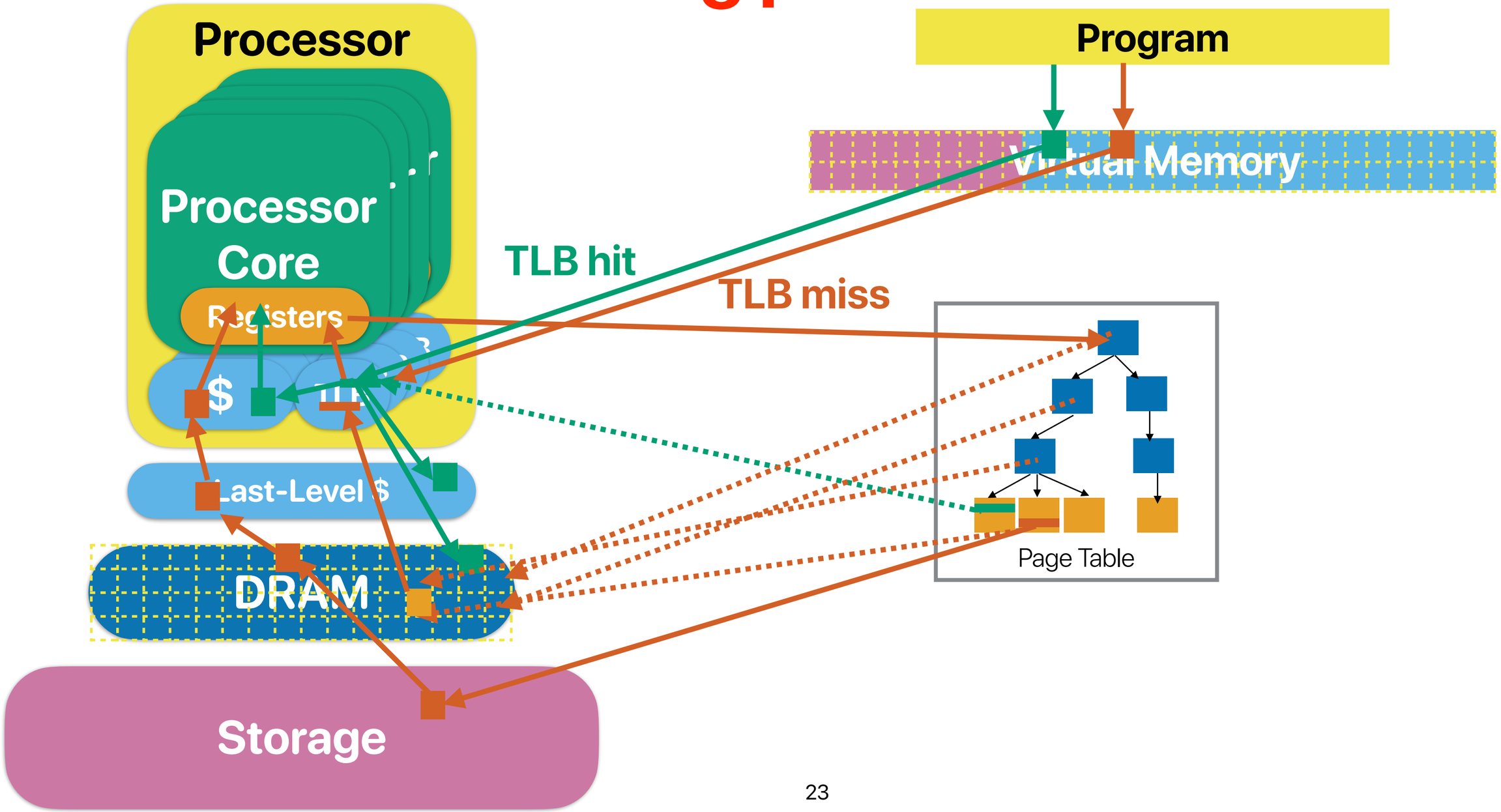
- How much slower (approximately) is your average memory access time in a system when the probability of a page fault/ swapping is 0.1% comparing with the case when there is no page fault/swapping?
  (Assume you swap to a hard disk)
  A. 10x
  B. 100x
  C. 1000x
  D. 10000x
  E. 100000x

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

Big picture


Processor
Processor Core
Registers
$
Last-Level $
DRAM
Storage
Program
Virtual Memory
TLB hit
TLB miss
Page Table

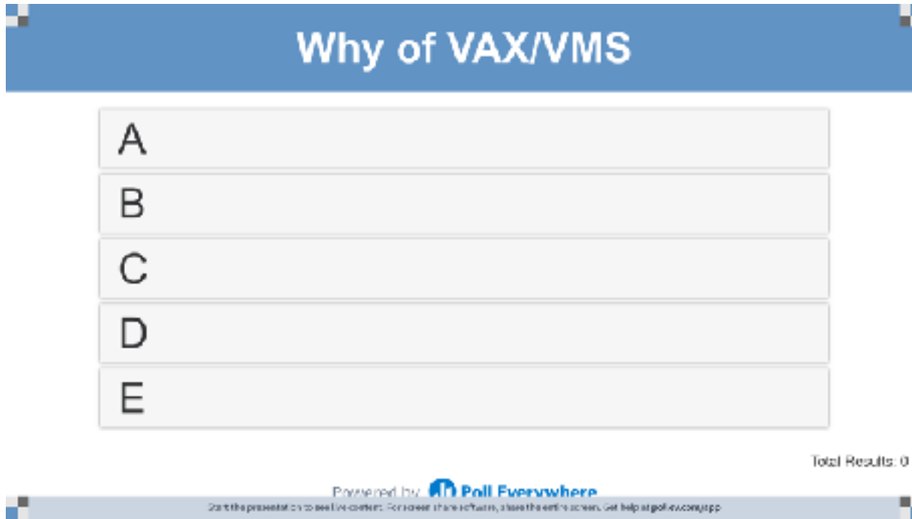
Virtual Memory Management in the VAX/VMS Operating System

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

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

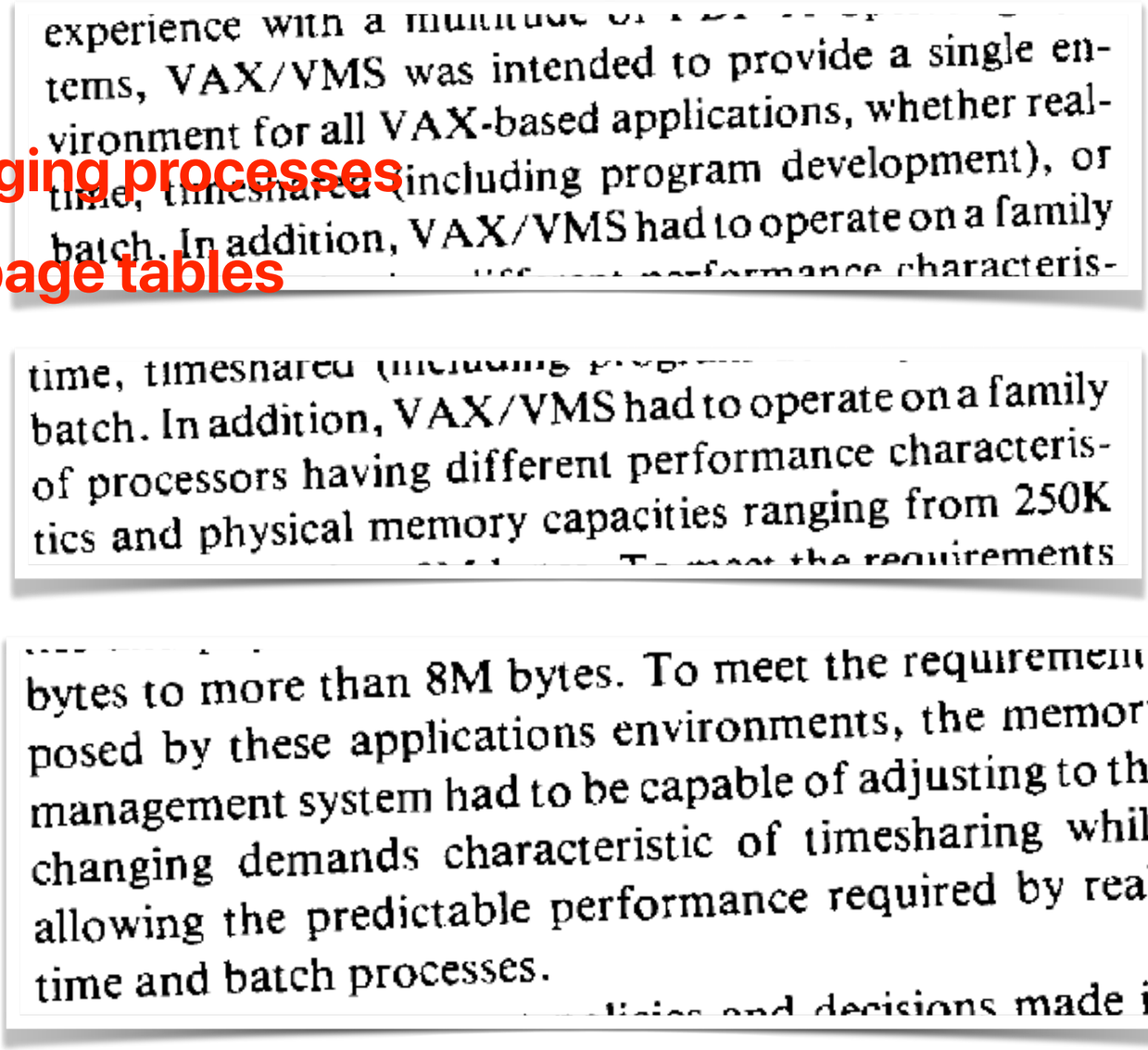
Why: The goals of VAX/VMS

- 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



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?

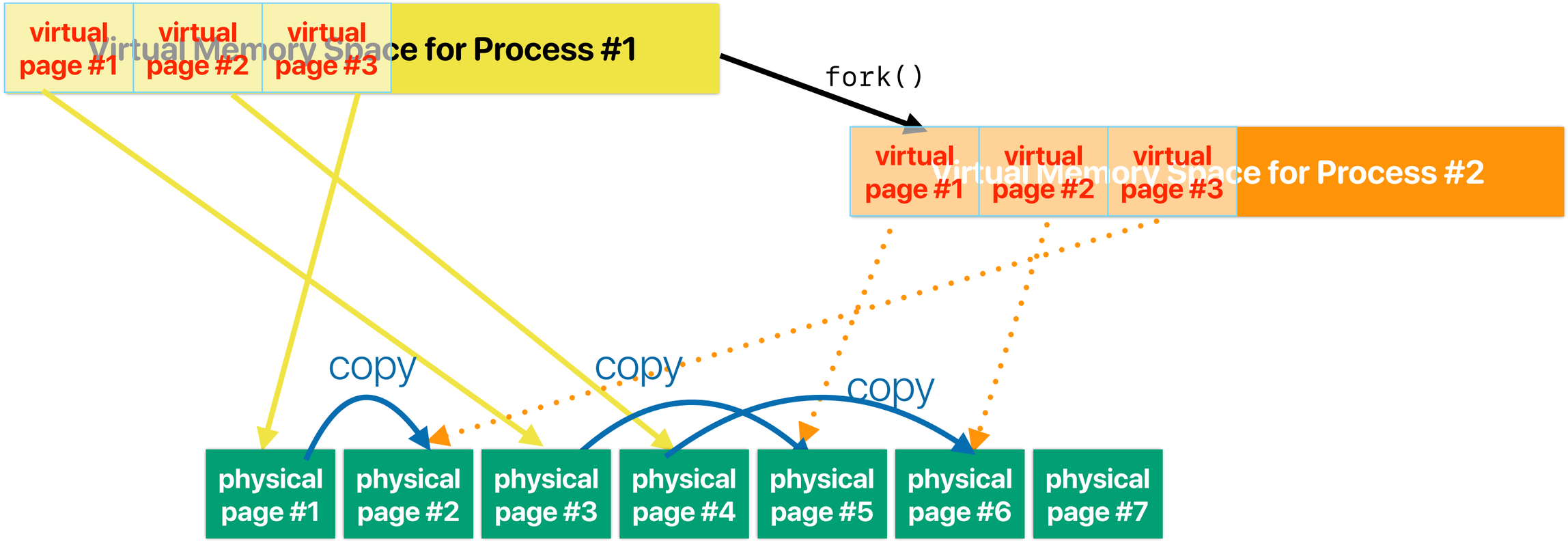
	Goal		Optimization
A	Process startup cost	**W**	Demand-zero & copy-on-refernce
B	Process performance interference	**X**	Process-local replacement
C	Page table lookup overhead	**Y**	Page clustering
D	Paging load on disks	**Z**	Page caching

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?

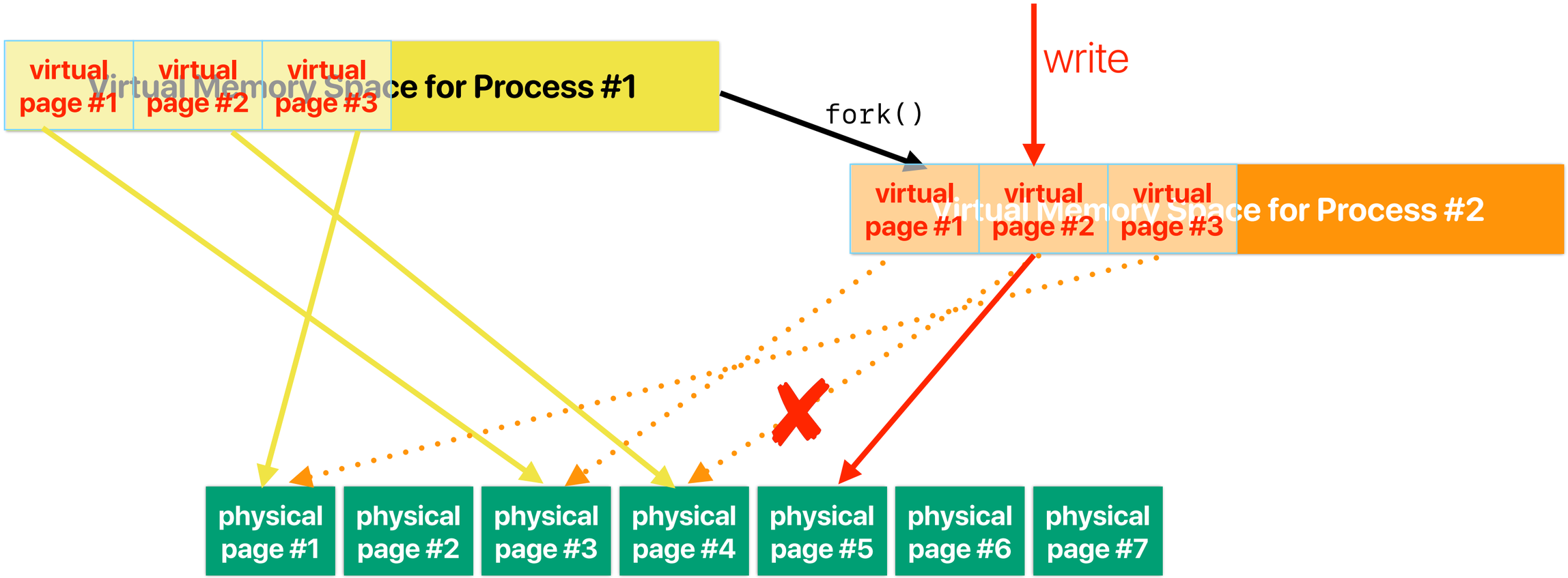
	Goal		Optimization
A	Process startup cost	**W**	Demand-zero & copy-on-refernce
B	Process performance interference	**X**	Process-local replacement
C	Page table lookup overhead	**Y**	Page clustering
D	Paging load on disks	**Z**	Page caching

What happens on a `fork`?



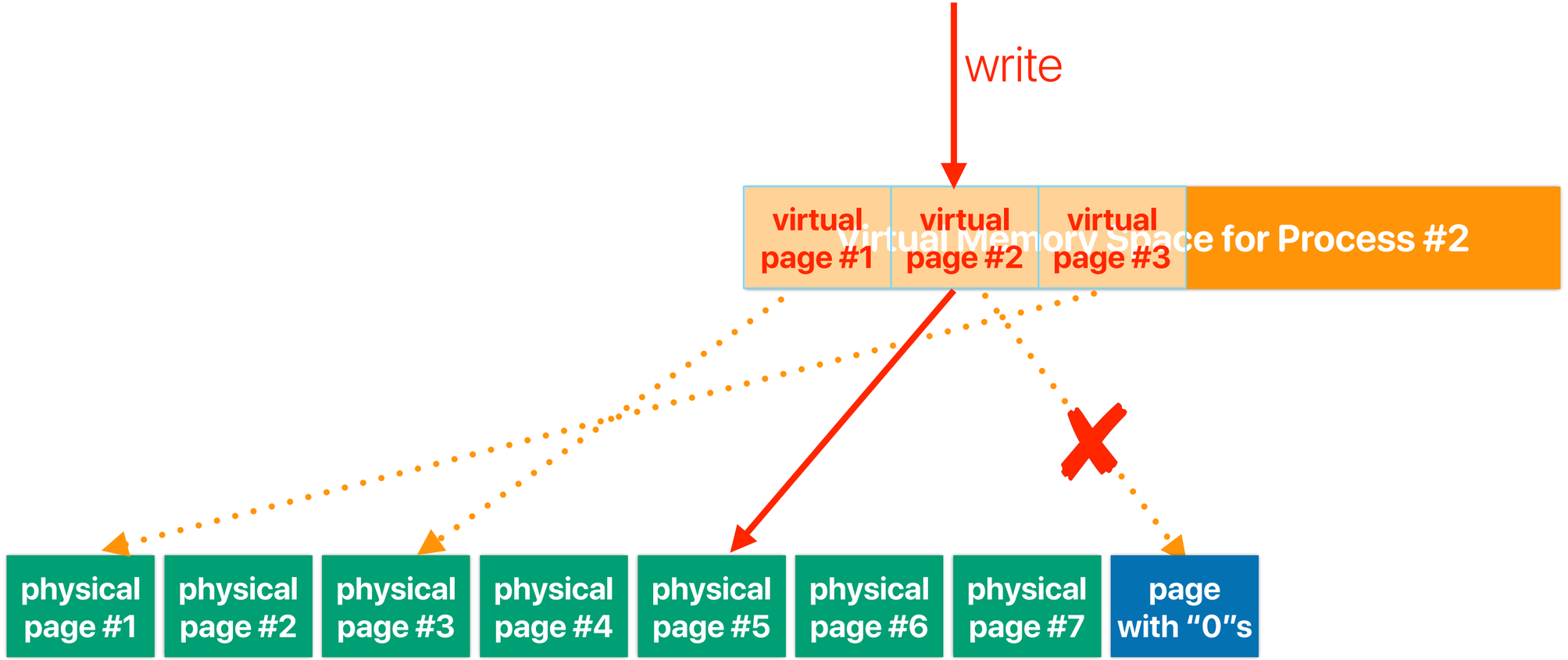
- **Copy the page content to different locations before the new process can start**

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

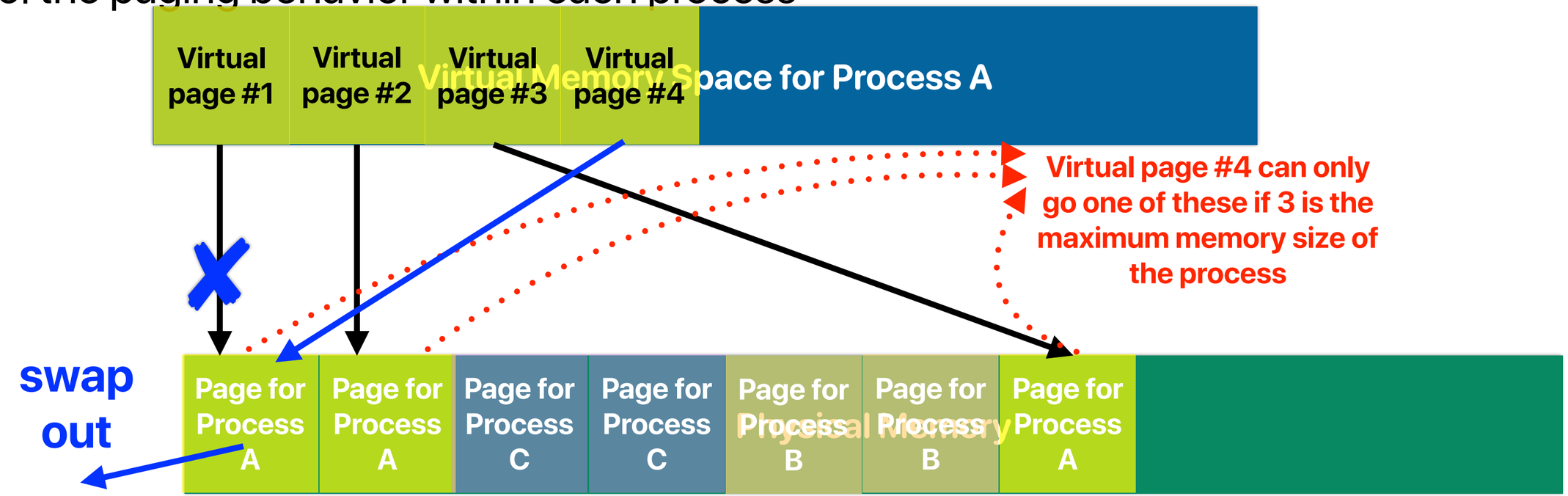
What VAX/VMS proposed to achieve these goals?

- 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 (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	

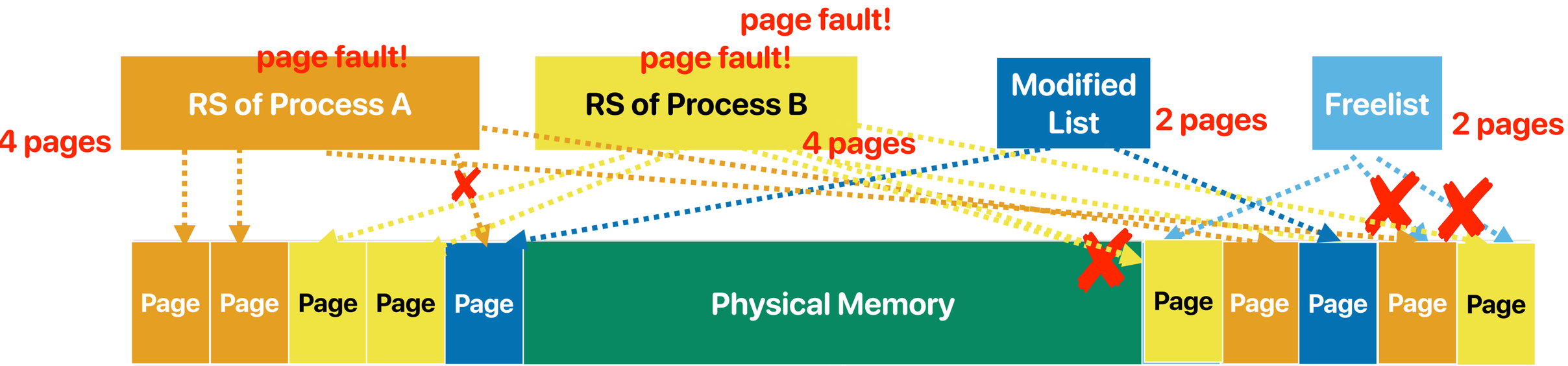
sequential access for a larger block is faster

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

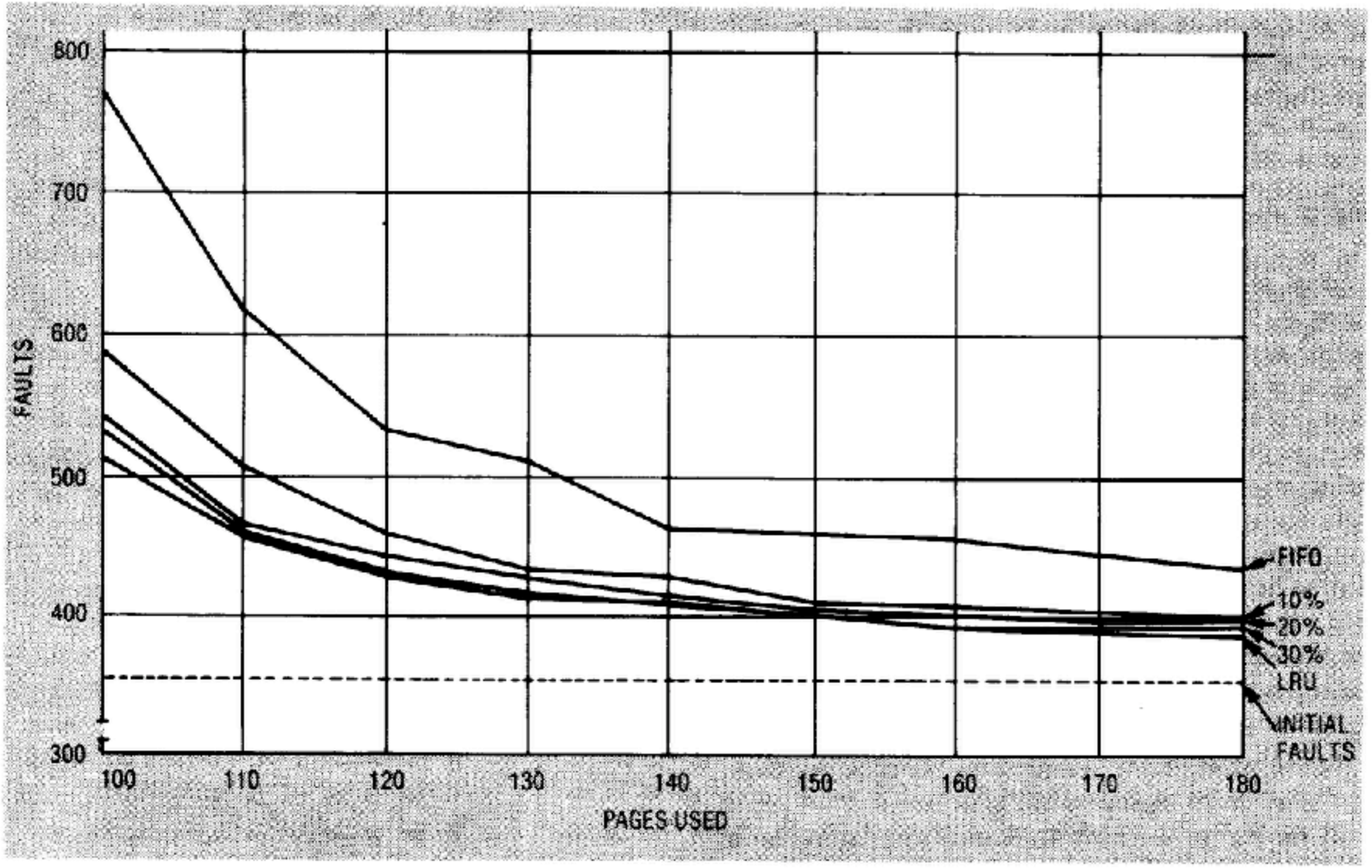


Figure 3. Faults vs. memory usage in Fortran compilation.

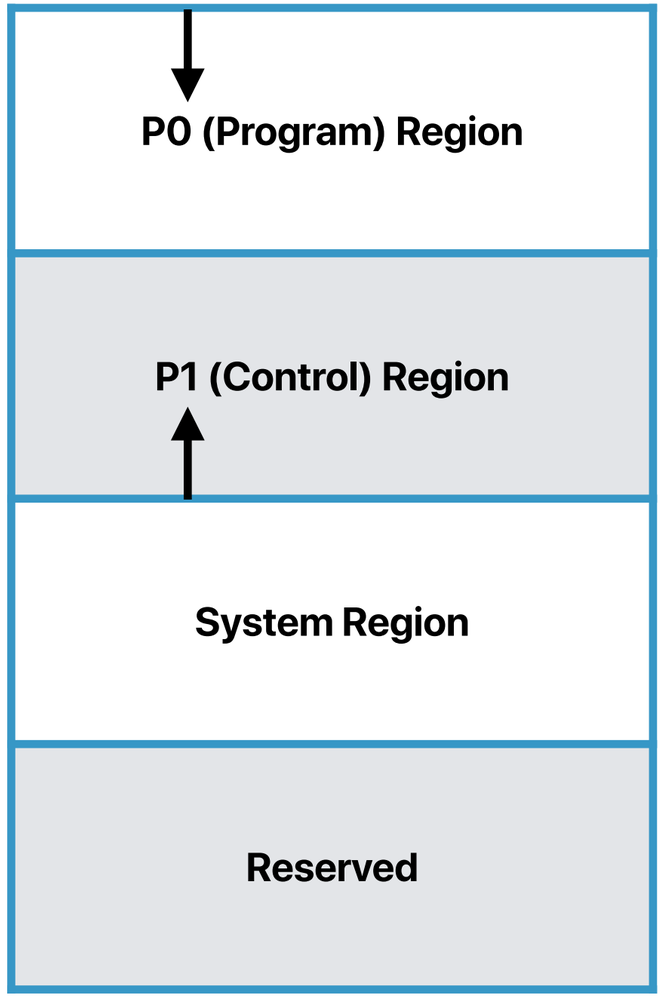
What VAX/VMS proposed to achieve these goals?

- 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-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

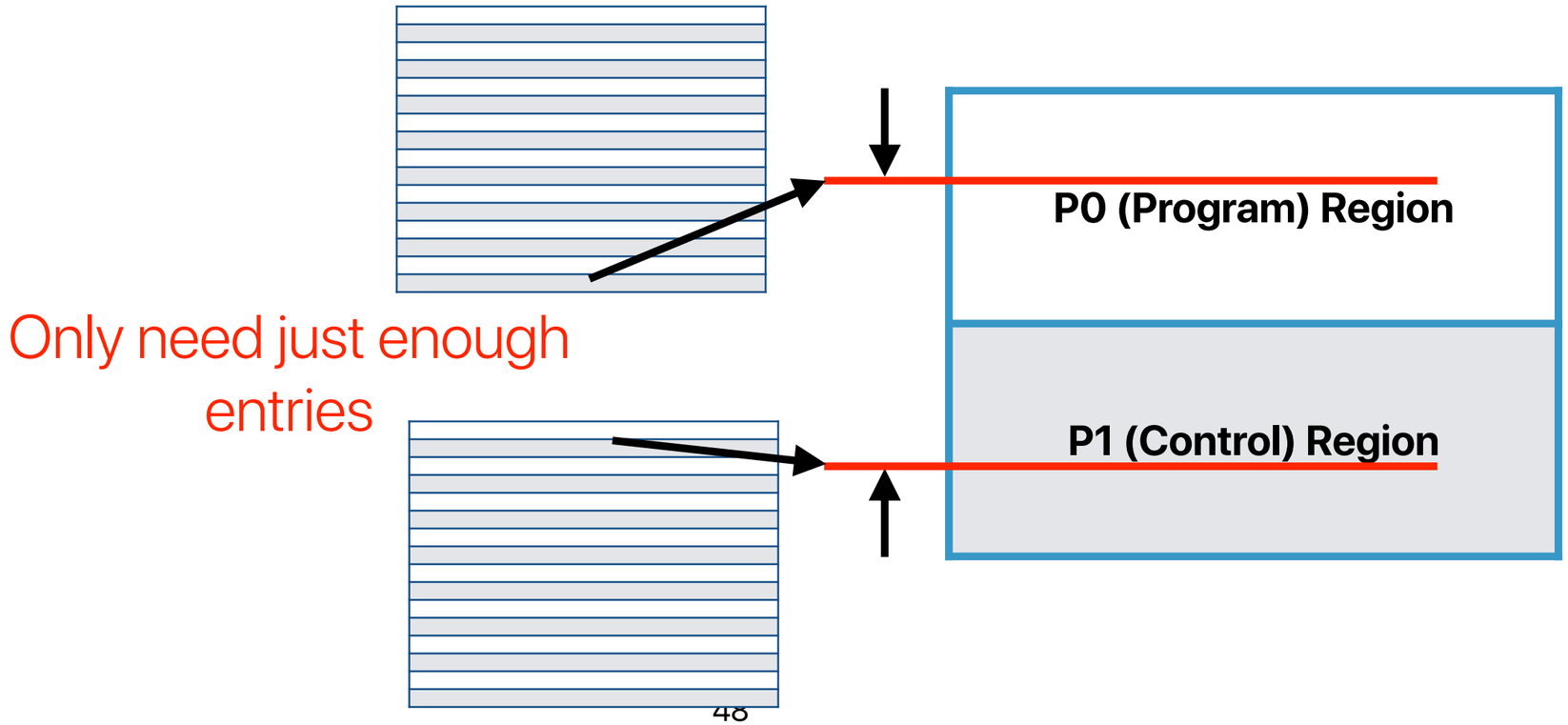
Stack
Other data

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?

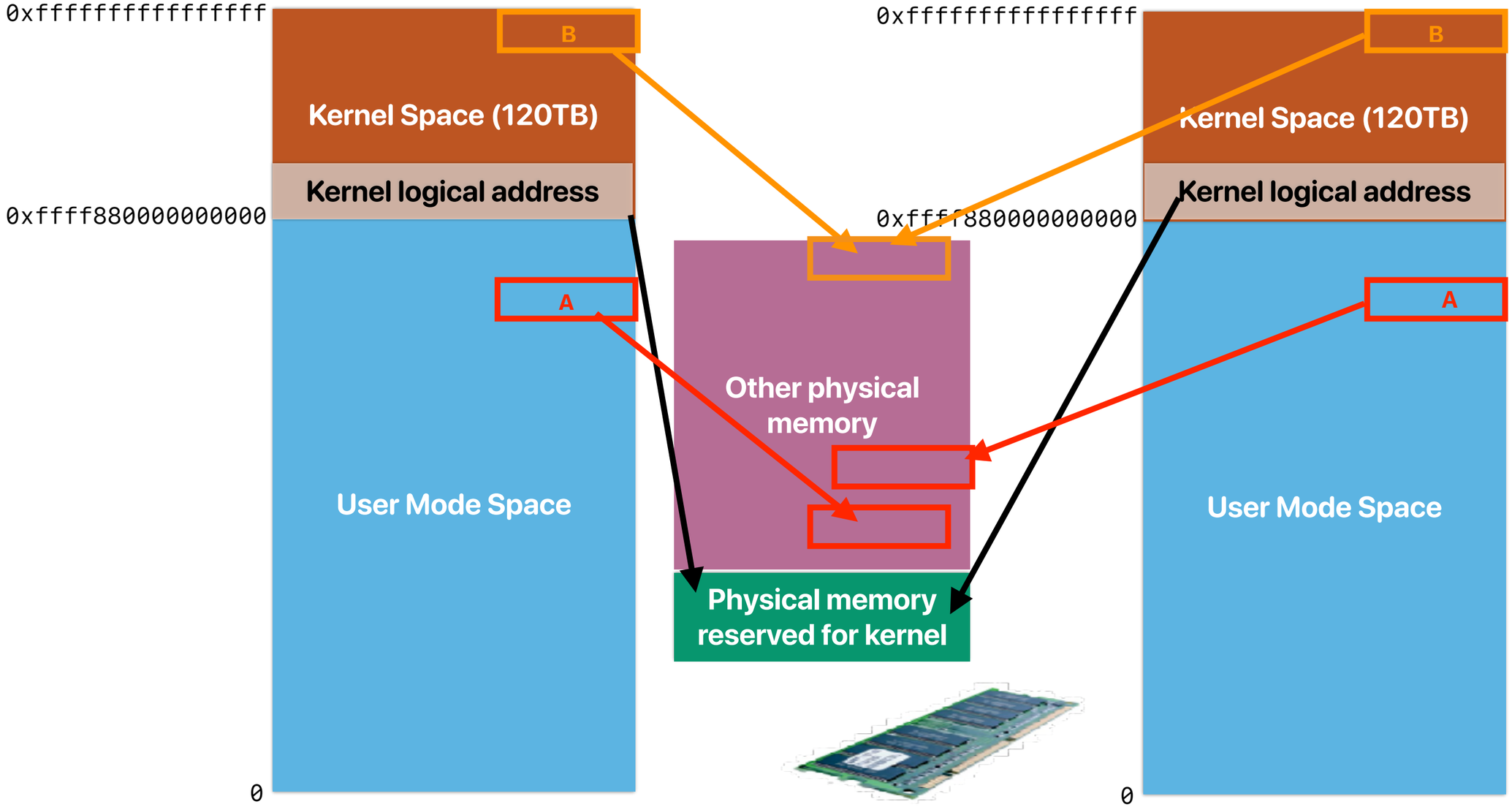
- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

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



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

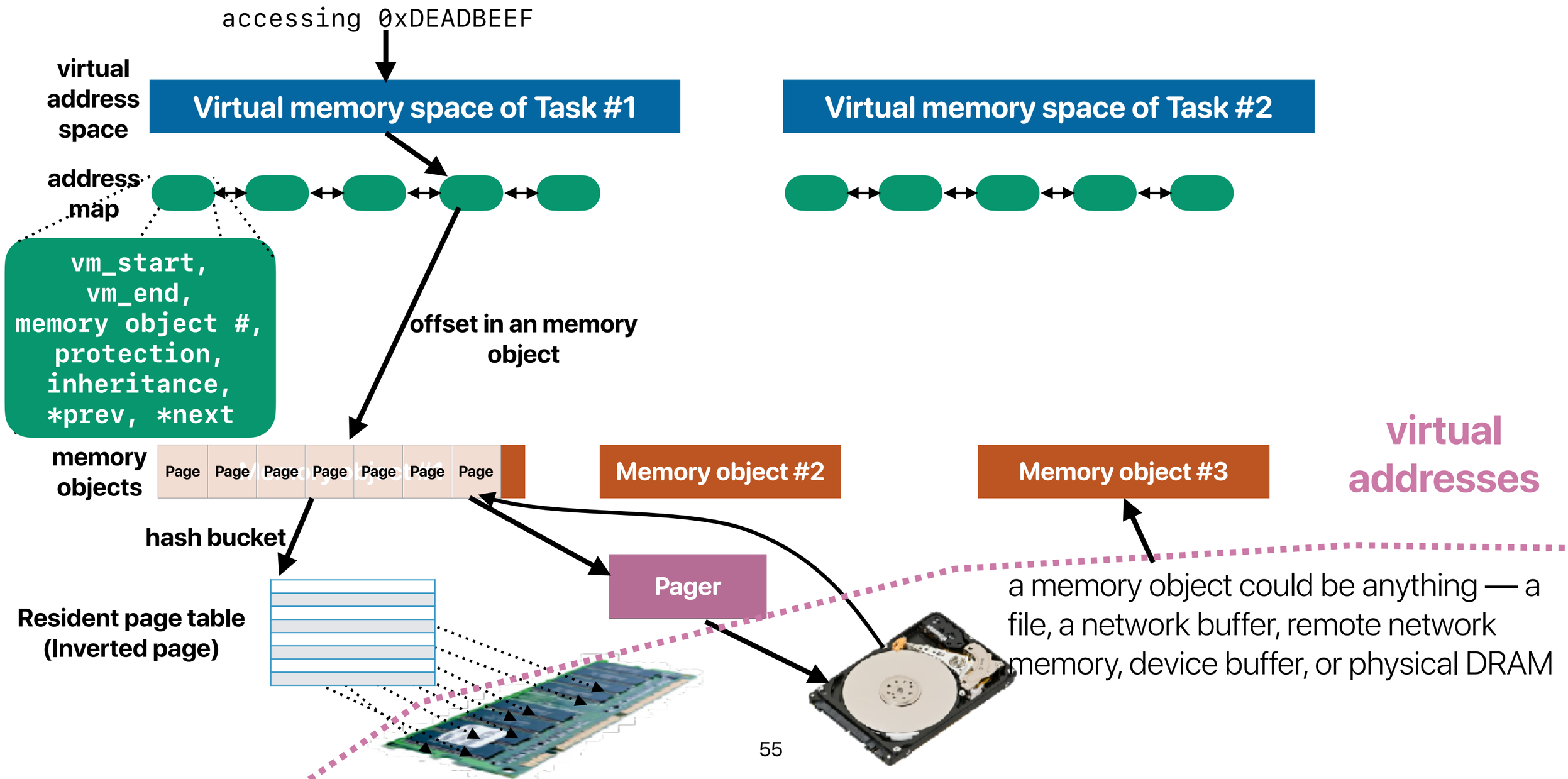
- Task: process in UNIX
- Thread: the basic scheduling identity
- Port: message queues protected by the kernel
- Message: data objects for inter-thread communication
- Memory object: data mapped into the address space of a task/process

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 the physical map (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 a small as several KBs

> VM objects are managed by the machine-independent VM system, with the underlying virtual to physical mappings handled by the machine-dependent *pmap system*. The `pmap` system actually handles page tables, translation lookaside buffers, segments, and so on, depending on the design of the underlying hardware.

https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/vm/vm.html

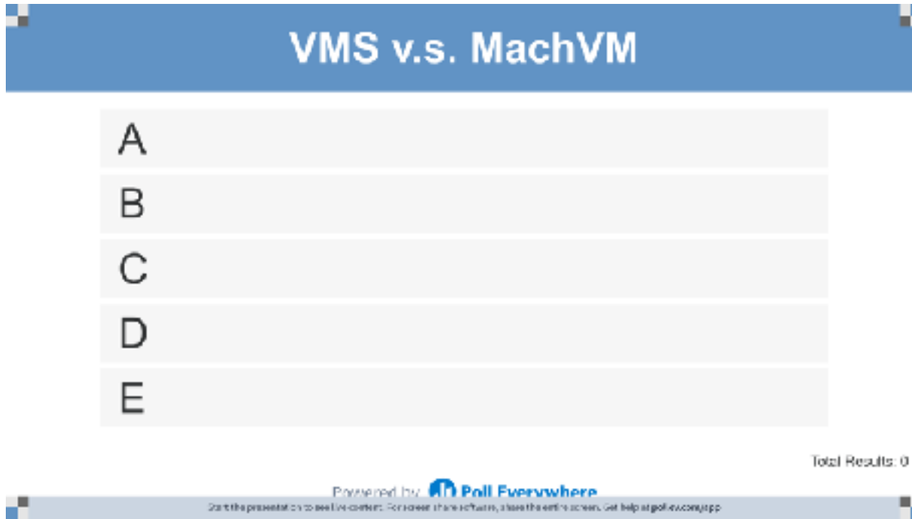
The impact of Mach VM

- 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

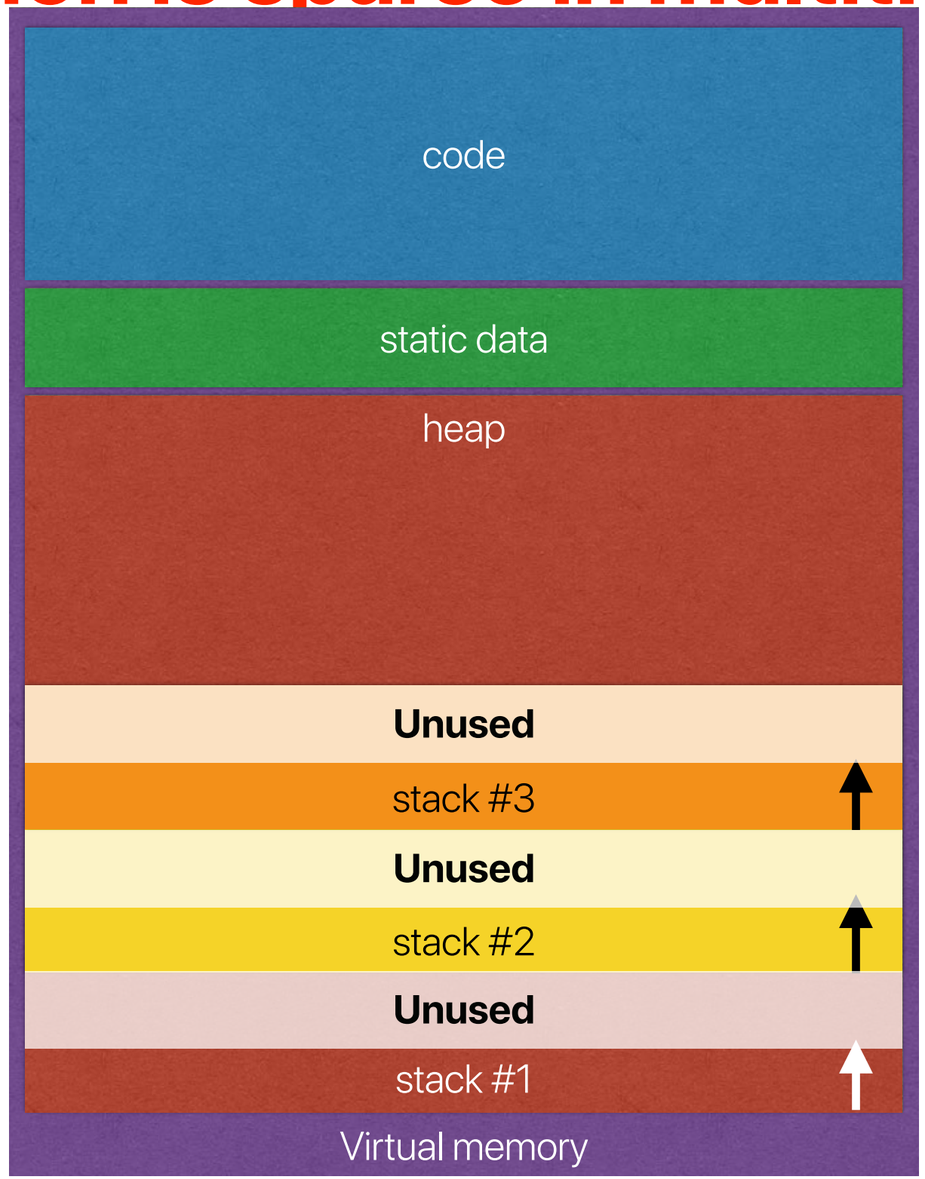


VAX v.s. Mach

- 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 — 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 allow the system more adaptive to various hardware architectures

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

Address allocation is sparse in multithreading model!



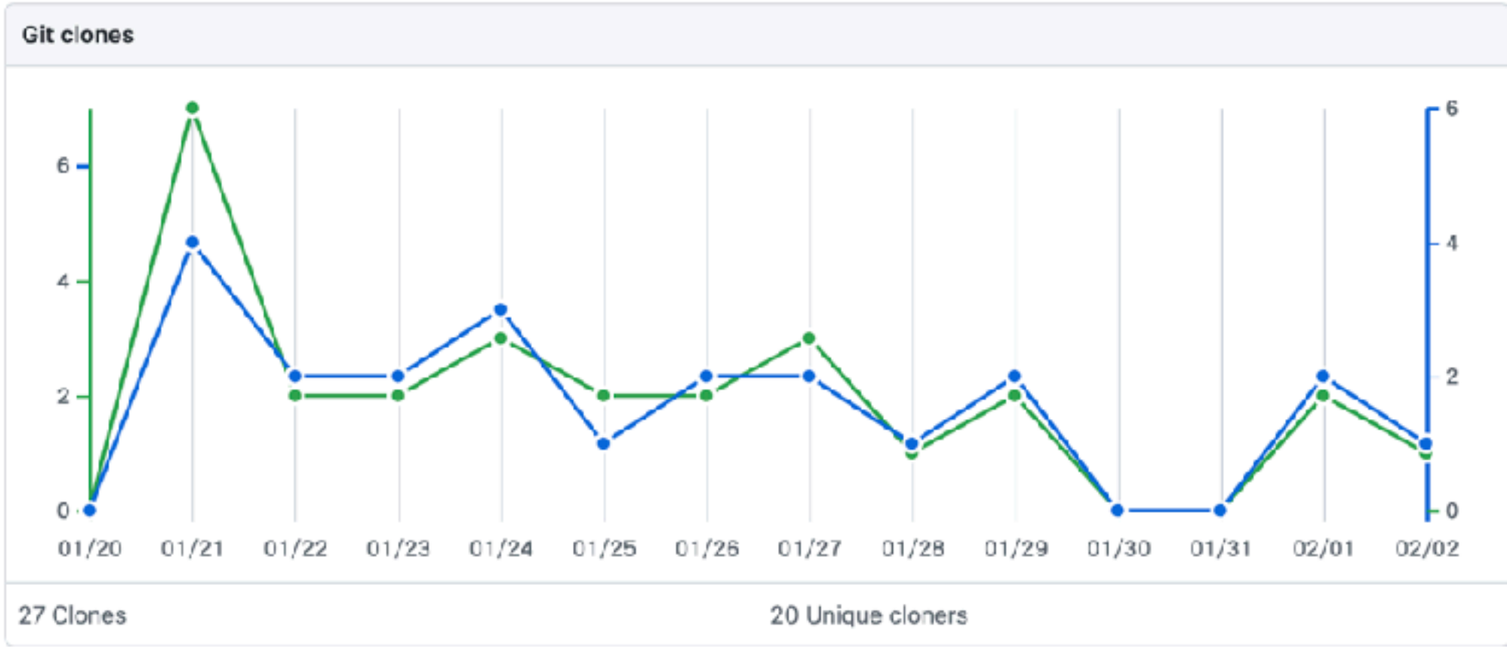
VAX v.s. Mach

- 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 — what's the benefit? — multithreading!
  - ~~③ Mach's VM would make the process creation more efficient~~ — both of them uses copy-on-reference…
  - ④ Mach's VM would allow the system more adaptive to various hardware architectures — what's the title of the paper?

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

Please start your project early!!!

- Groups in 2 (3 **at most**)
  - 25—30 groups expected
  - 18 clones only(?)
    — each of you should work on the code
  - The project is NOT EASY
  - You will meet questions related to the project in exams.
    — No free rider is allowed!



- We have office hours to address each of your need.
- **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**

Announcement

- Reading quizzes due next Tuesday
- 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

Sample Midterm

Disclaimer

- This is just a sample midterm for you to practice.
- Questions listed in multiple choices can be transformed as free answer or short answer questions in the midterm.
  - Same thing for sample free answer questions and short answer questions

Regarding the “true” midterm

- Rules
  - Cheating is not allowed and you will receive an F once we identified that
  - Will release on 2/10/2021 0:00am and due on 2/11/2021 11:59:00pm
  - No lecture on 2/10/2021
  - 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**
- Format (tentatively)
  - 15 multiple choices
  - 4 short answer questions
    — You have to explain everything within 30 words.
  - 3 free answer questions

What OS must track for a process?

- Which of the following information does NOT the OS need to track?
  A. Stack pointer
  B. Program counter
  C. Scheduling information
  D. Registers
  E. None of the above

What do we need for parallel processing

- Which process component(s) must we replicate in order to take advantage of multiple cores/CPUs?
  A. The address space (i.e. memory)
  B. Misc. resources (e.g. open files)
  C. Execution state (e.g. PC, registers, stack pointer)
  D. More than one of the above
  E. None of the above (explain).

Is hierarchical design a good idea?

- How many the following is/are true regarding the proposed hierarchical design in Dijkstra's THE.
  - ① Hierarchical design facilitates debugging
  - ② Hierarchical design makes verification of system components easier
  - ③ Hierarchical design reduces the overhead of running a single process
  - ④ The proposed hierarchical design allows layer 0 to schedule I/O & peripherals

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

What is “system nucleus”

- Regarding “system nucleus”, how many of the following statements are correct?
  - ① The system nucleus is a process
  - ② The system nucleus allows multiple operating systems to execute concurrently
  - ③ The system nucleus provides primitives to load and swap programs
  - ④ Operating systems are user-level processes in the system nucleus architecture
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Processes and threads

- Regarding processes (or say tasks in Mach) and threads, how many of the following statements is/are correct?
  - ① Threads within a process share the same address space
  - ② Threads within a process can communicate and synchronize using shared memory
  - ③ Processes can communicate through messages
  - ④ Two processes may only be able to communicate through messages
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Why not microkernels?

- Although Mach's design strongly influenced modern operating systems, why most modern operating systems do not adopt the design of microkernels?
  A. Microkernels are more difficult to extend than monolithic kernels
  B. Microkernels are more difficult to maintain than monolithic kernels
  C. Microkernels are less stable than monolithic kernels
  D. Microkernels are not as competitive as monolithic kernels in terms of application performance
  E. Microkernels are less flexible than monolithic kernels

What we learned about kernel

- Which of the following is true about kernel?
  A. It executes as a process
  B. It is always executing, in support of other processes
  C. It should execute as little as possible.
  D. A & B
  E. B & C

Interrupt and Trap

- How many of the following statements is/are true regarding interrupt and trap?
  - ① Both interrupt and trap can incur context switch
  - ② Both interrupt and trap are raised from hardware
  - ③ Both interrupt and trap require OS kernel to handle
  - ④ Both interrupt and trap are machine dependent features
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Did they achieve their 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

The role of the OS in virtual memory management

- How many of the following tasks in virtual memory management always requires the assistance of operating system?
  - ① Address translation
  - ② Growth of process address space
  - ③ Tracking free physical memory locations
  - ④ Maintaining mapping tables
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Address translation in x86-64

63:48 (16 bits)	47:39 (9 bits)	38:30 (9 bits)	29:21 (9 bits)	20:12 (9 bits)	11:0 (12 bits)
SignExt	L4 index	L3 index	L2 index	L1 index	page offset

- The above shows the address partition in x86-64. According to this information, how many of the following is/are true?
  - ① x86-64 provides 16EB virtual memory space
  - ② each node in the hierarchical page contains 512 entries
  - ③ the default page size is 4KB
  - ④ if only three level indexes are used, x86-64 can support 2MB page size

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

Free list

- How many of the following statements regarding the “free list” is/are correct?
  - ① It can improve the latency of a page fault
  - ② It can reduce the latency of swapping out a page
  - ③ It can incur disk accesses without page faults
  - ④ It doesn’t allow a page in the list to be used for other purpose
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Disable interrupts?

- Which of the following can disabling interrupts guarantee for for the bounded-buffer example?
  A. At most one process/thread in its critical section
  B. A thread outside of its critical section cannot block another thread from entering its critical section
  C. A thread cannot be postponed indefinitely from entering its critical section
  D. The solution should work regardless the speed of executing threads and the number of processors
  E. None of the above

Wait-free synchronization

- How many of the following statements fulfill the requirements of wait-free synchronization?
  - ① Any operation from a process accessing the shared data structure must be completed within a finite number of steps
  - ② No process can be prevented from completing its operation by failures from other processes
  - ③ The implementation of wait-free synchronization cannot depend on hardware support
  - ④ The implementation of wait-free synchronization should work regardless of the processing speed

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

Answer the following within 30 words

- Differences among 4 types of kernels
- Differences between threads and processes
- The pros and cons between using threads and processes to parallelize an application
- Differences between global and local page replacement policy. Examples of using global/local page replacement policies?
- What is “free list”? How is it used in the system?
- Differences among VAX/VMS and Mach’s VM. Why initial version of UNIX’s VM resembles VAX/VMS?
- Differences between UNIX in the 70’s paper and now. Why?
- What is segmentation fault? What is page fault?
- What is thrashing? What’s saturation?

Think deep

- What problem does multi-level scheduling policy try to address? What they proposed to address the issue?
- How Linux's CFS works? What problem would occur if we carelessly apply CFS to multicore systems?
- What must happen during a context switch? How expensive is a context?
- What must happen when creating a process? How expensive is creating a process? How about creating a thread?
- Can you support threads without bothering the kernel? Pros and cons?

Page replacement policy

- Assume your OS uses LRU policy when handle page faults. Also assume that we have 3 physical memory pages available. How many page faults will you see in the following page reference sequence?

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	9	4	8	7	9	4	8	7	9	4	8	7

- What if we use FIFO?
- What if we use Clock policy?
- Can you propose a policy outperform both FIFO and Clock?

Programming using “RCU”

```
1 void delete(struct el *p)
2 {
3     A;
4     p->next->prev = p->prev;
5     p->prev->next = p->next;
6     B;
7     C;
8 }
```

- What kind of code are you going to replace in A, B, C to perform “deletion” of a node in a linked list using RCU?

More forks

- Consider the following code

```
fork();
printf(“moo\n”);
fork();
printf(“oink\n”);
fork();
printf(“baa\n”);
```

What is the output?

Programming questions in C

- How to implement a simple shell that can launch a user program given from the command line?
- How to implement a shell that can redirect program output to a file?