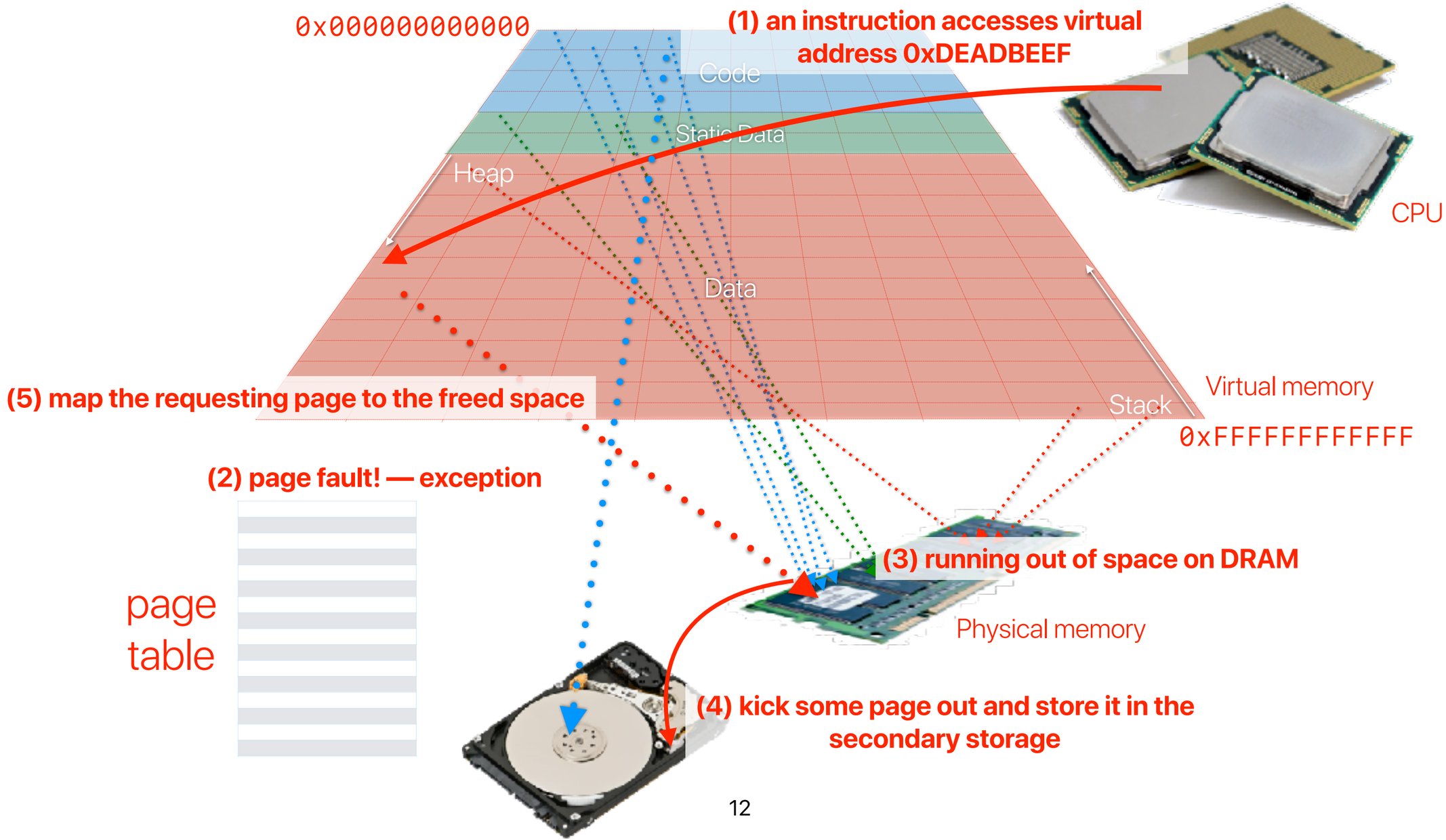
Virtual memory (III): System Architecture and Design

Hung-Wei Tseng

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)

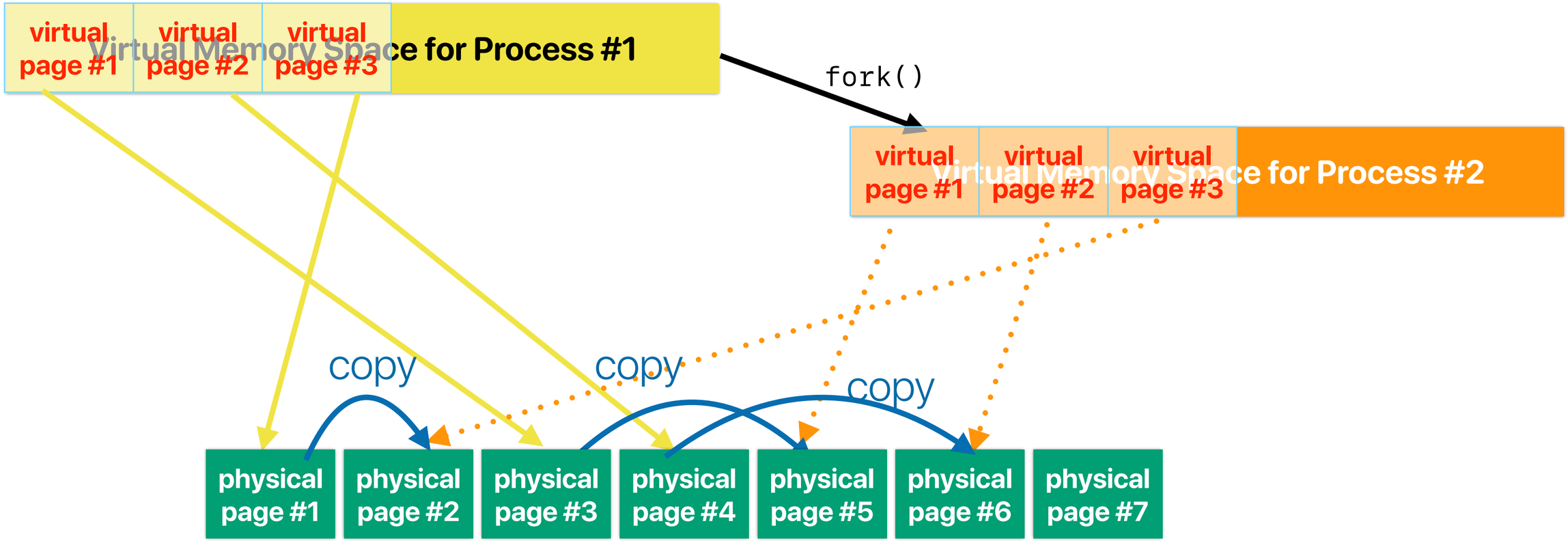
Operations	Latency (ns)	Latency (us)	Latency (ms)	
L1 cache reference	0.5 ns			~ 1 CPU cycle
Branch mispredict	3 ns			
L2 cache reference	4 ns			14x L1 cache
Mutex lock/unlock	17 ns			
Send 2K bytes over network	44 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with 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

Virtual Memory Management in the VAX/VMS Operating System

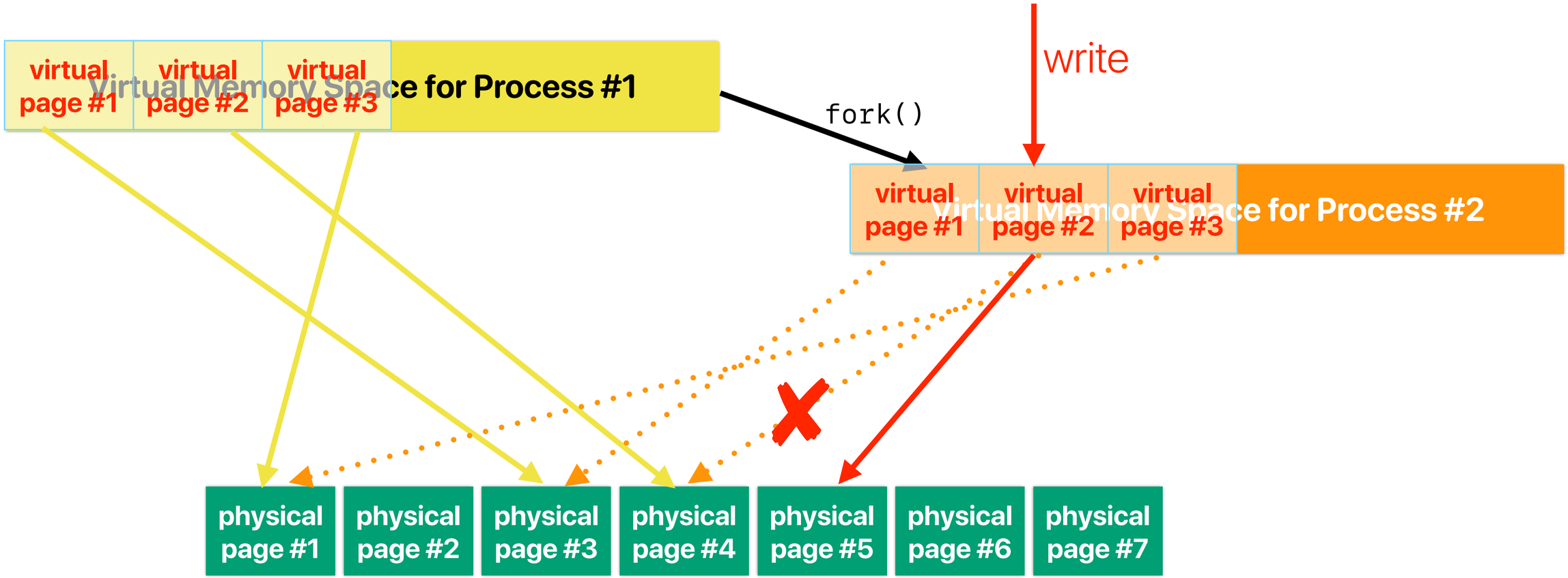
H. M. Levy and P. H. Lipman
Digital Equipment Corporation

What happens on a `fork`?



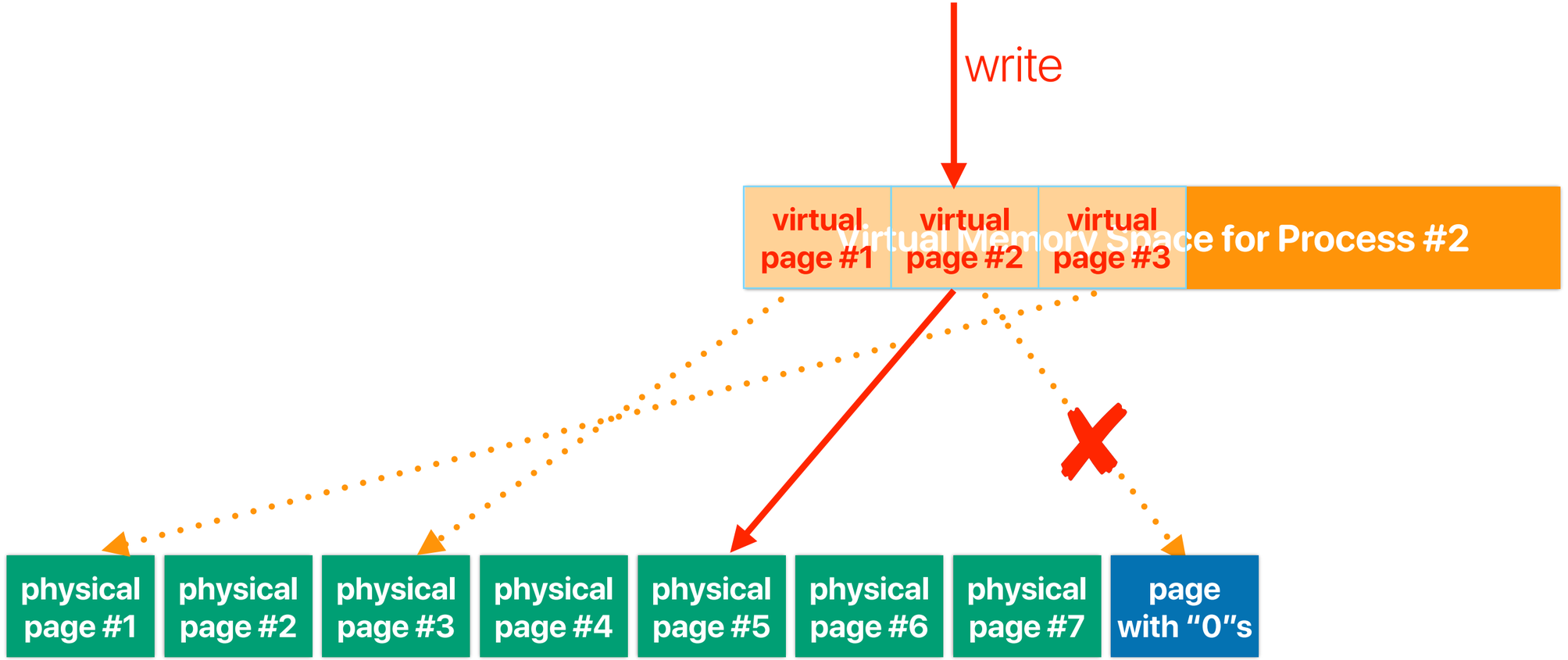
- **Copy the page content to different locations before the new process can start**

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

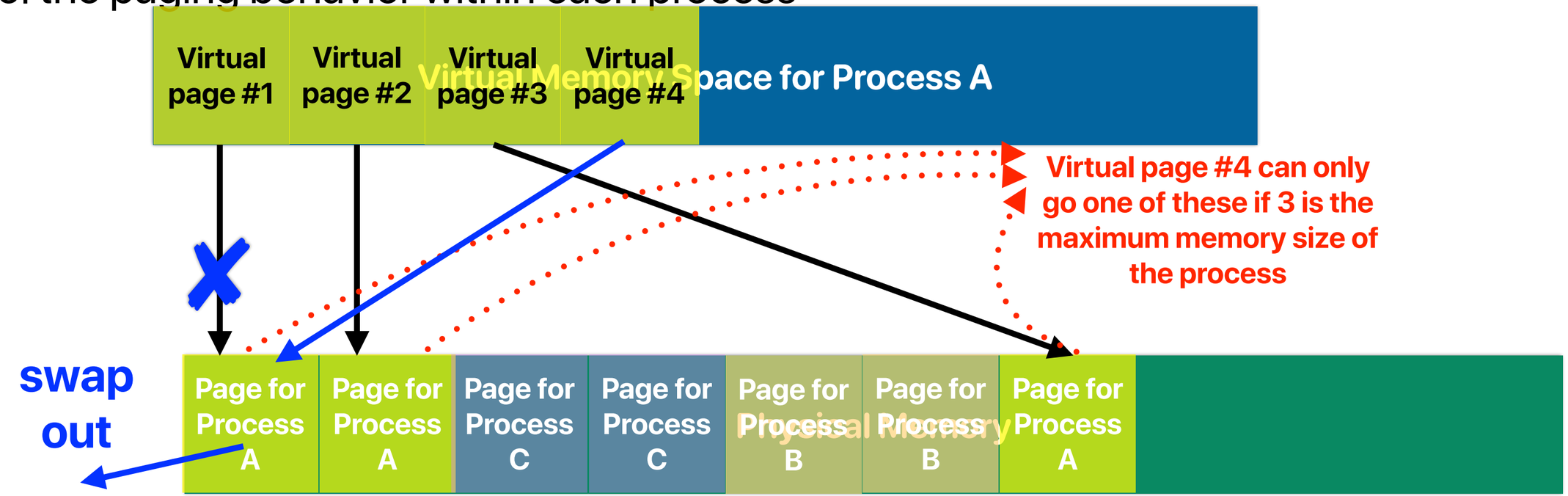
What VAX/VMS proposed to achieve these goals?

- 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



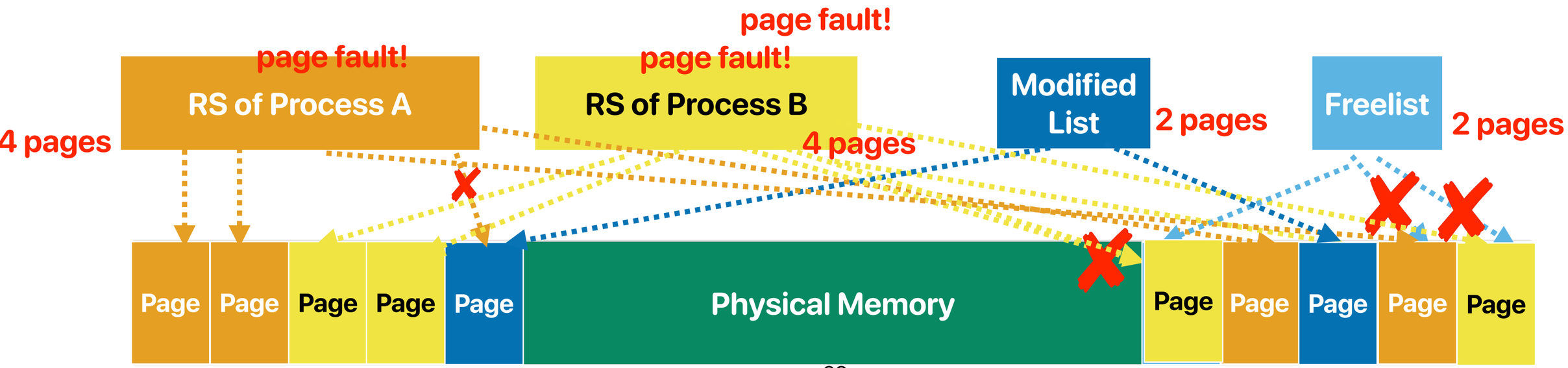
What's the policy? **FIFO!** **Low overhead!**

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

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

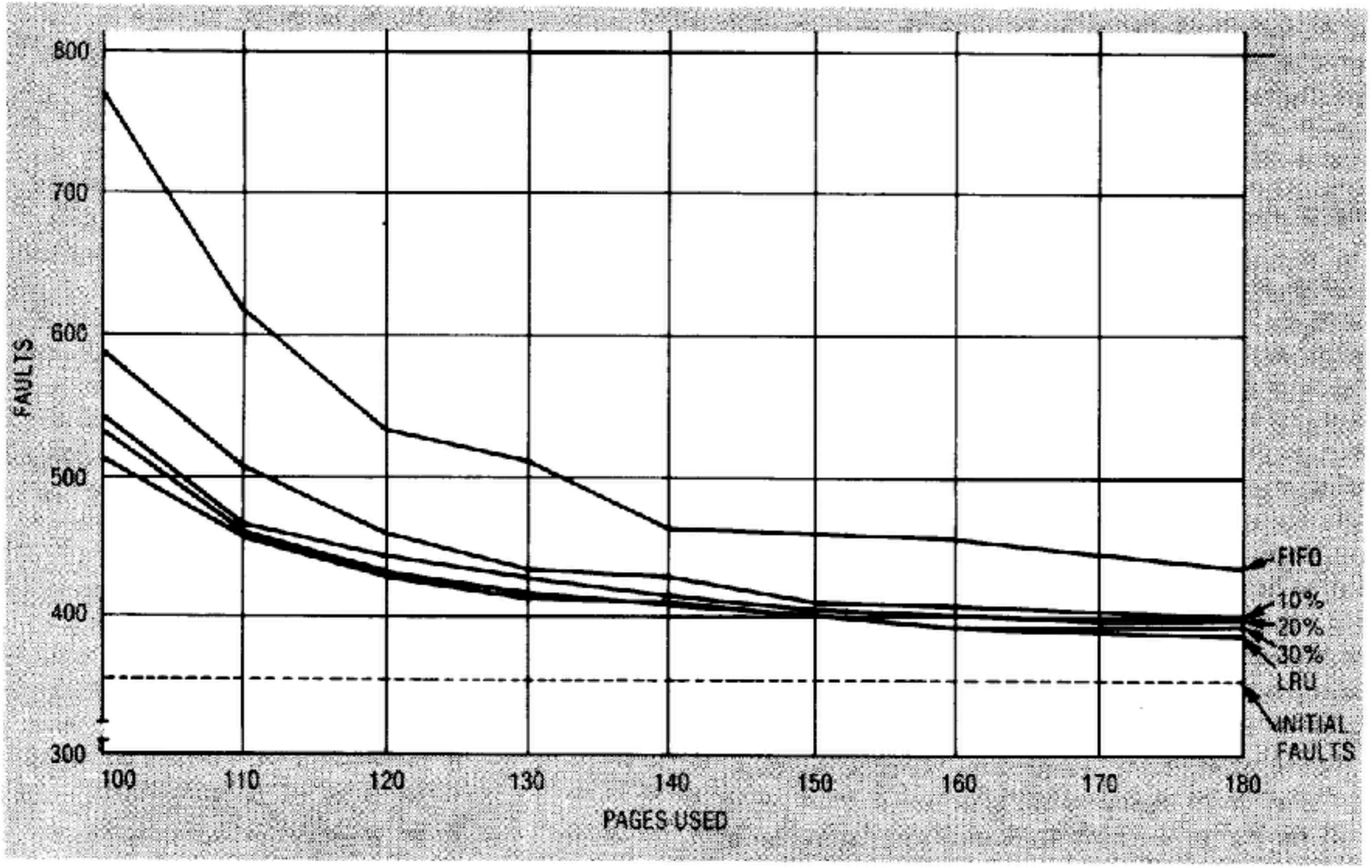
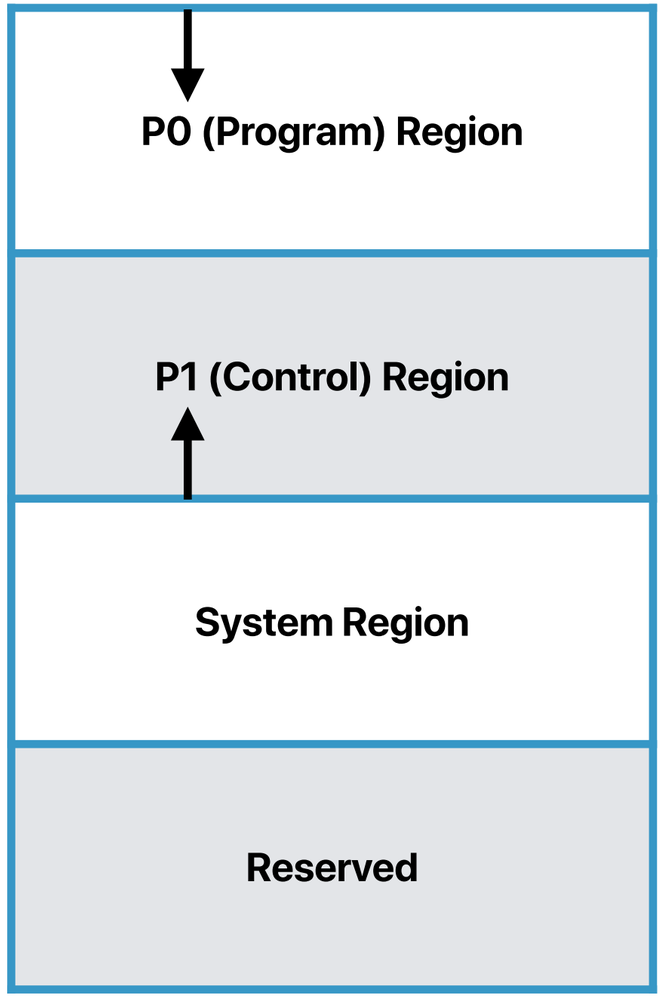


Figure 3. Faults vs. memory usage in Fortran compilation.

Process memory layout



Code
Heap

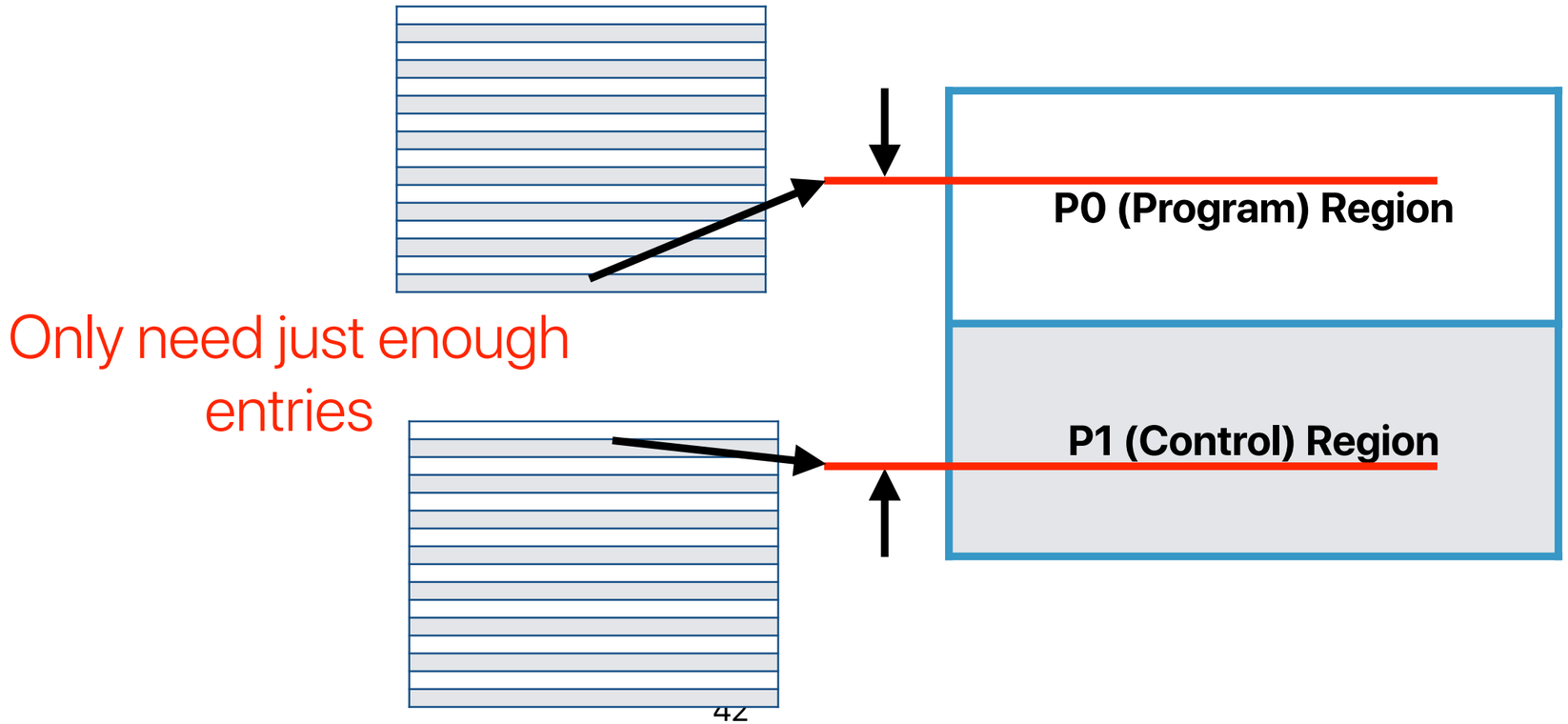
Stack
Other data

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?

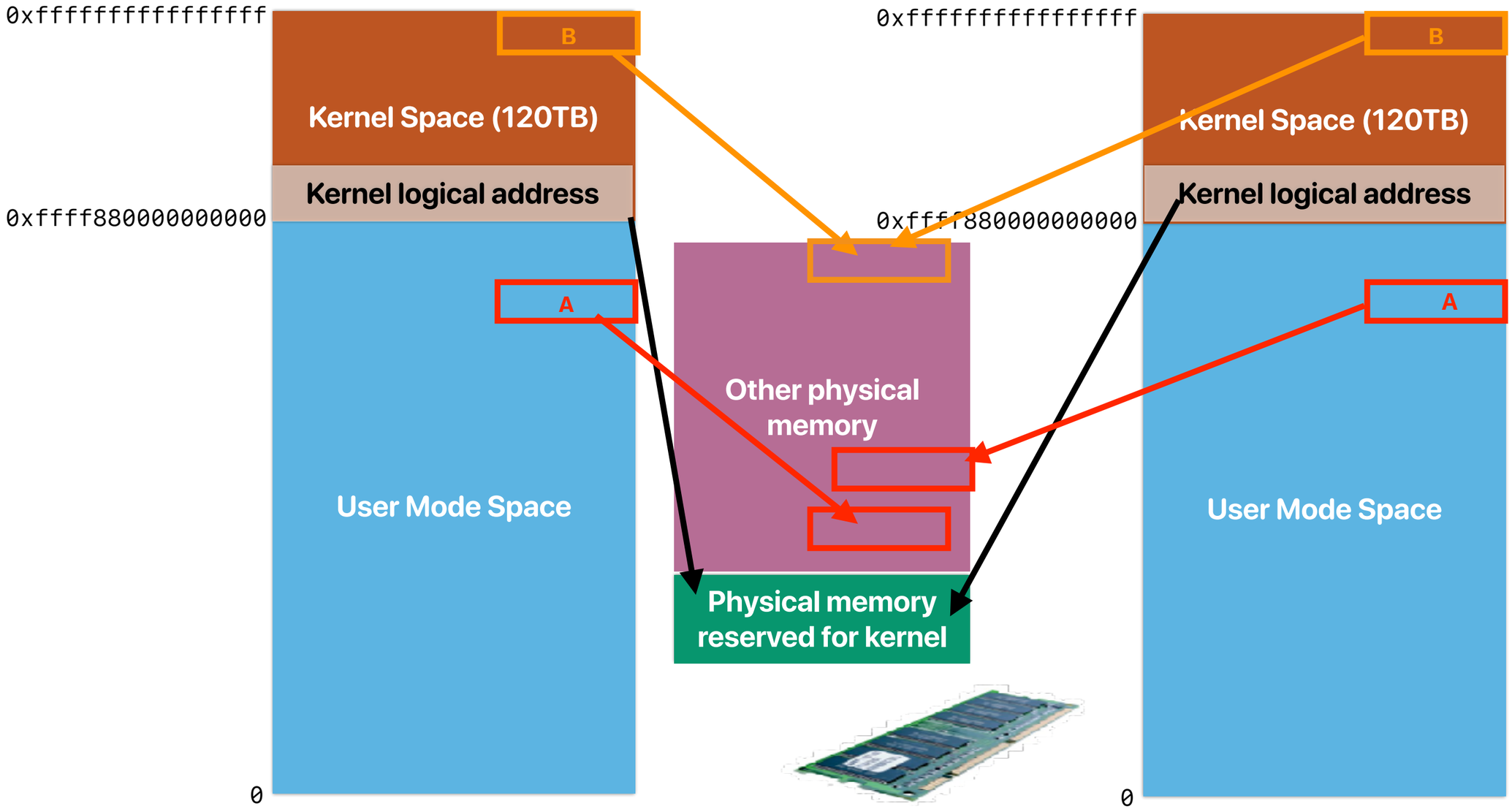
- Each segment has its own page table
- Entries between stack and heap boundaries do not need to be allocated — reduce the size of page table



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

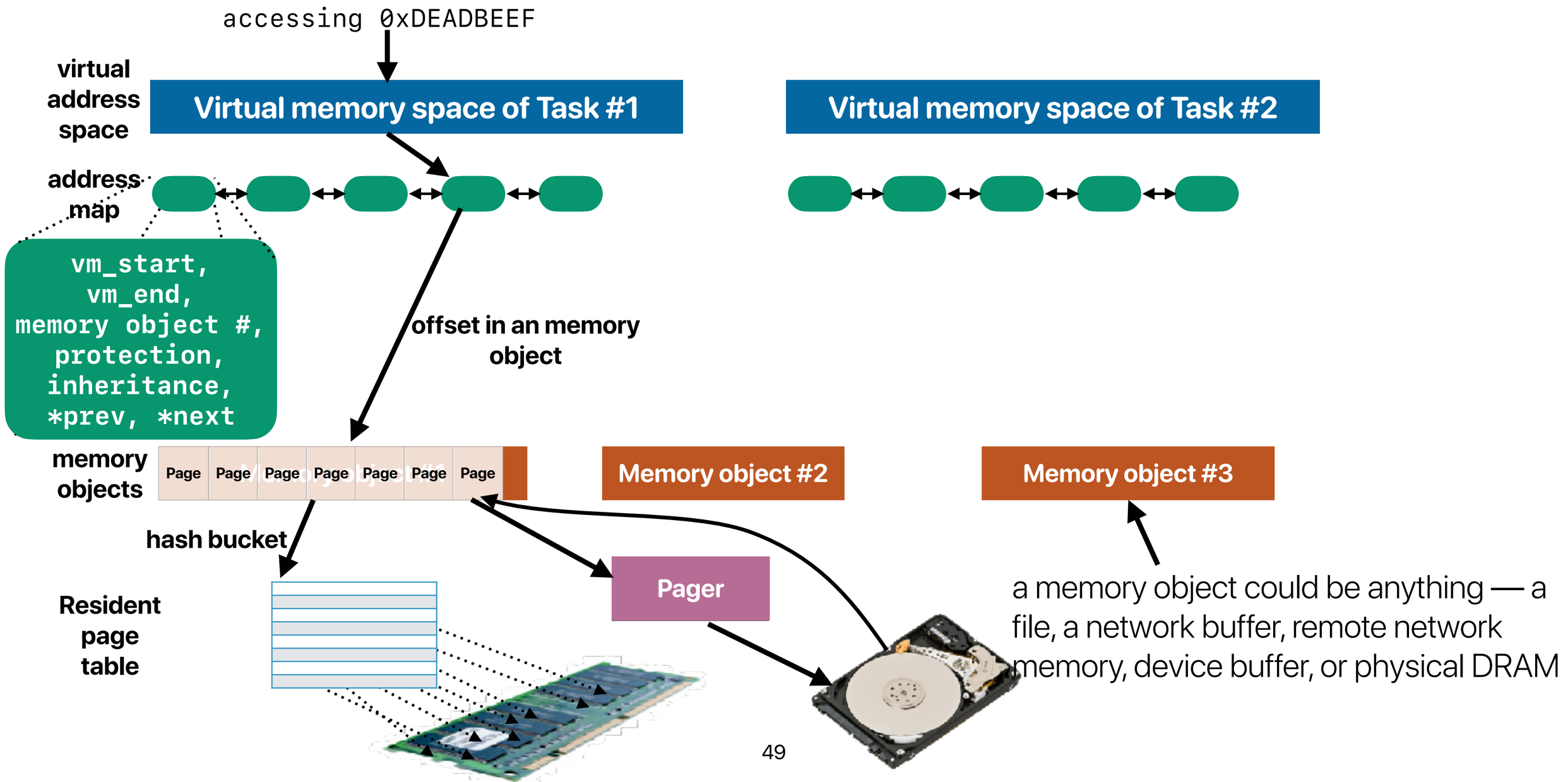
We mentioned previously

- Memory object: data mapped into the address space of a task/process

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 a 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?