(MIS)CONFIGURING PAGE TABLES

BY ARTEM SHISHKIN

DISCLAIMER

I am presenting the contents of this presentation in my personal capacity. The views expressed are solely my own and do not reflect the views of Intel Corporation or its affiliates.

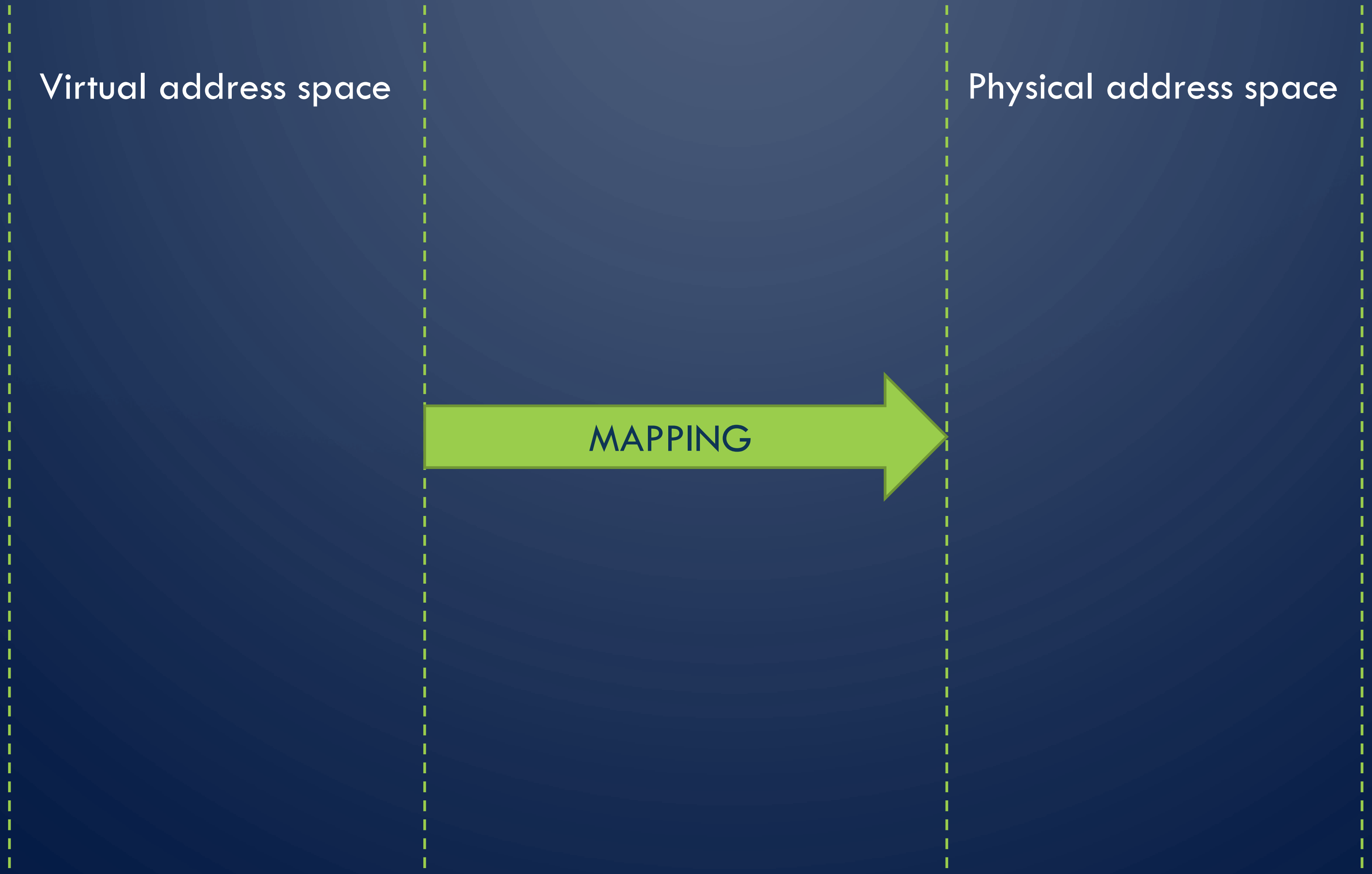
WHOAMI

- Security researcher at Intel Corporation
- Intel STORM team member
- Windows kernel enthusiast

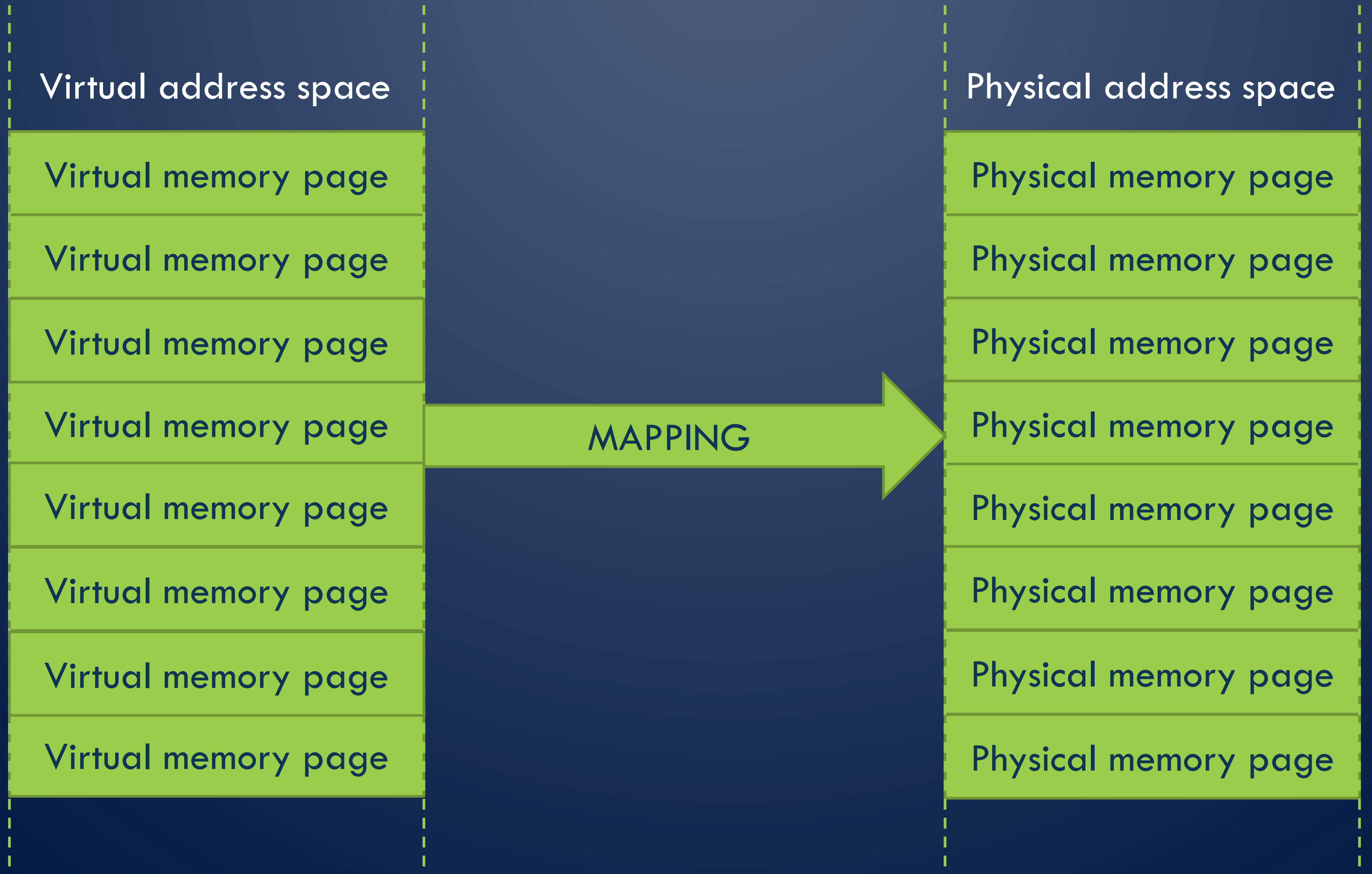
VIRTUAL MEMORY

FROM SCRATCH

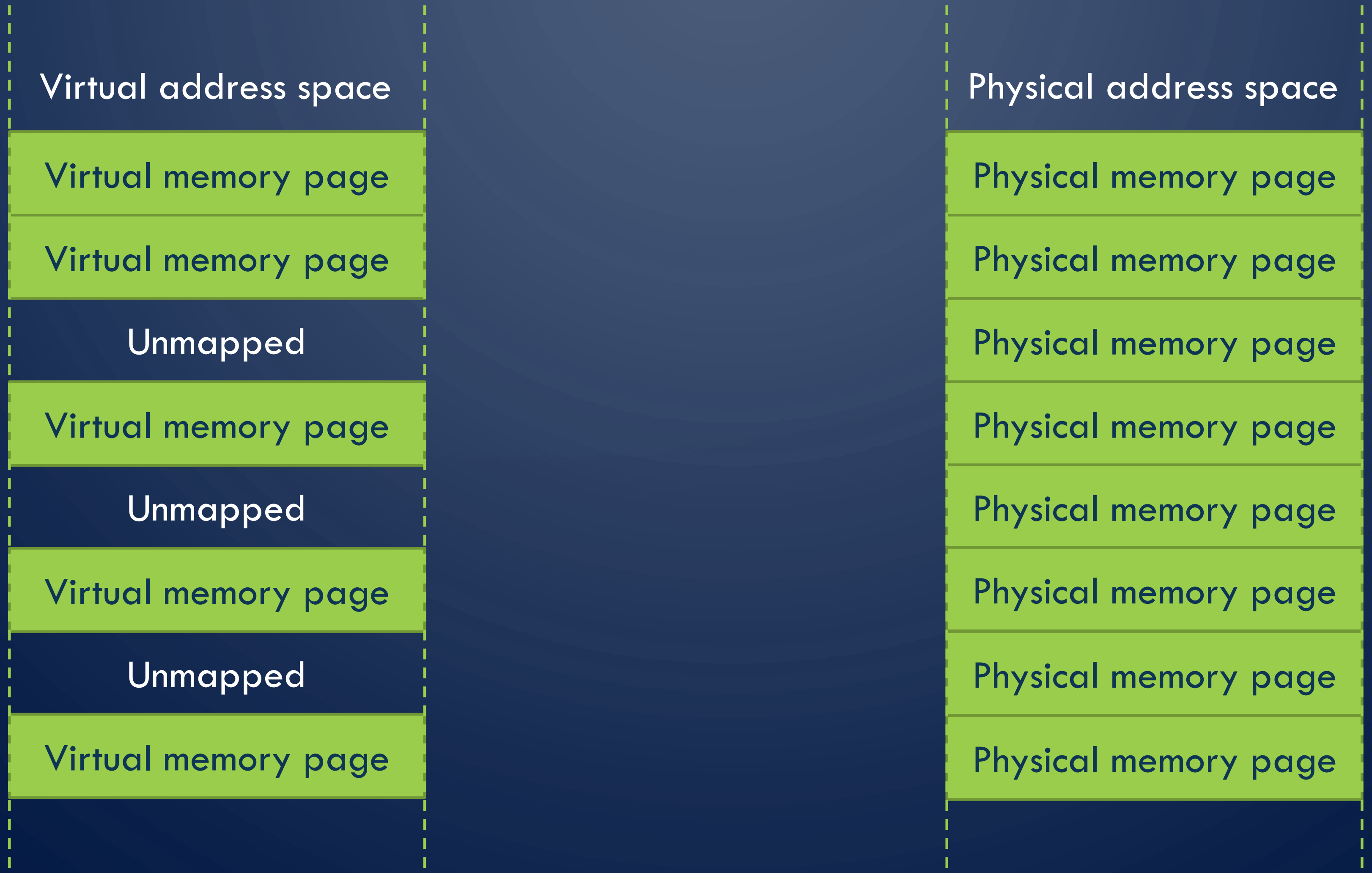
VIRTUAL MEMORY CONCEPT



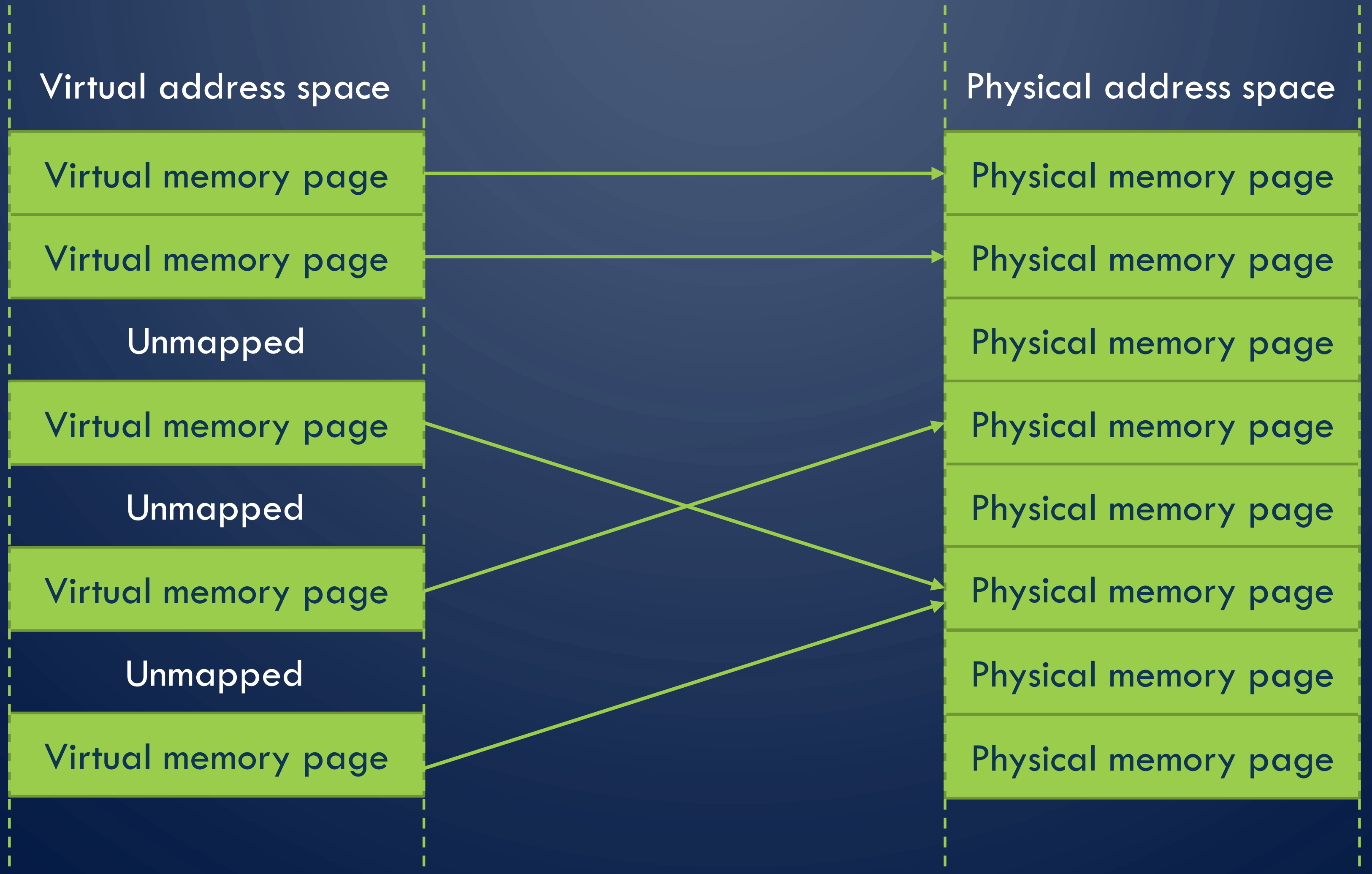
VIRTUAL MEMORY CONCEPT



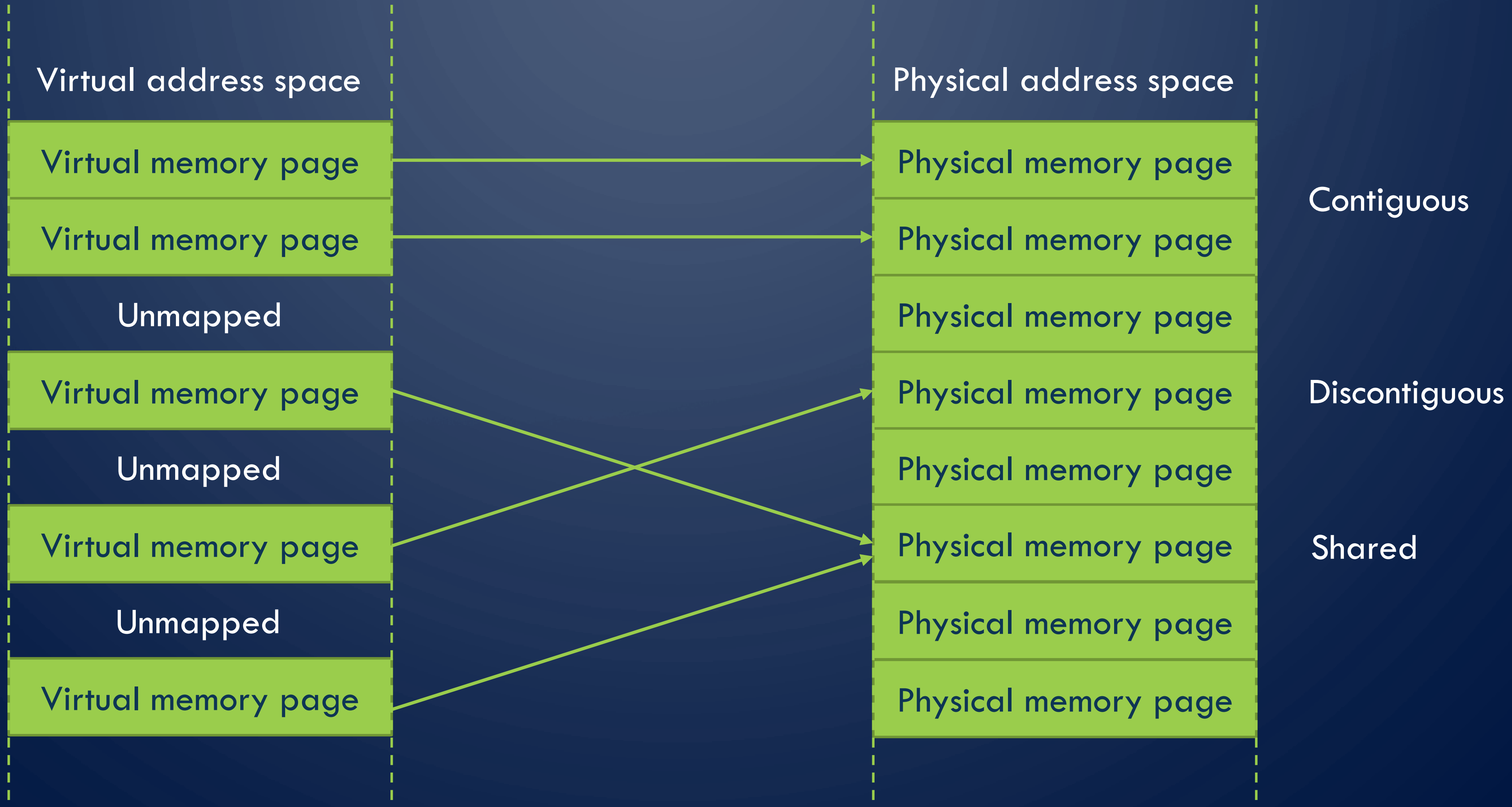
VIRTUAL MEMORY CONCEPT



VIRTUAL MEMORY CONCEPT



VIRTUAL MEMORY CONCEPT



VIRTUAL MEMORY BENEFITS

- Virtualize memory as a resource
  - Pretend we have a lot of it
  - Customize memory layout (like flat)
- Isolate execution environments
- Mitigate memory fragmentation
- Fine grained access control and protection
- A lot more, I'm just saying virtual memory is great

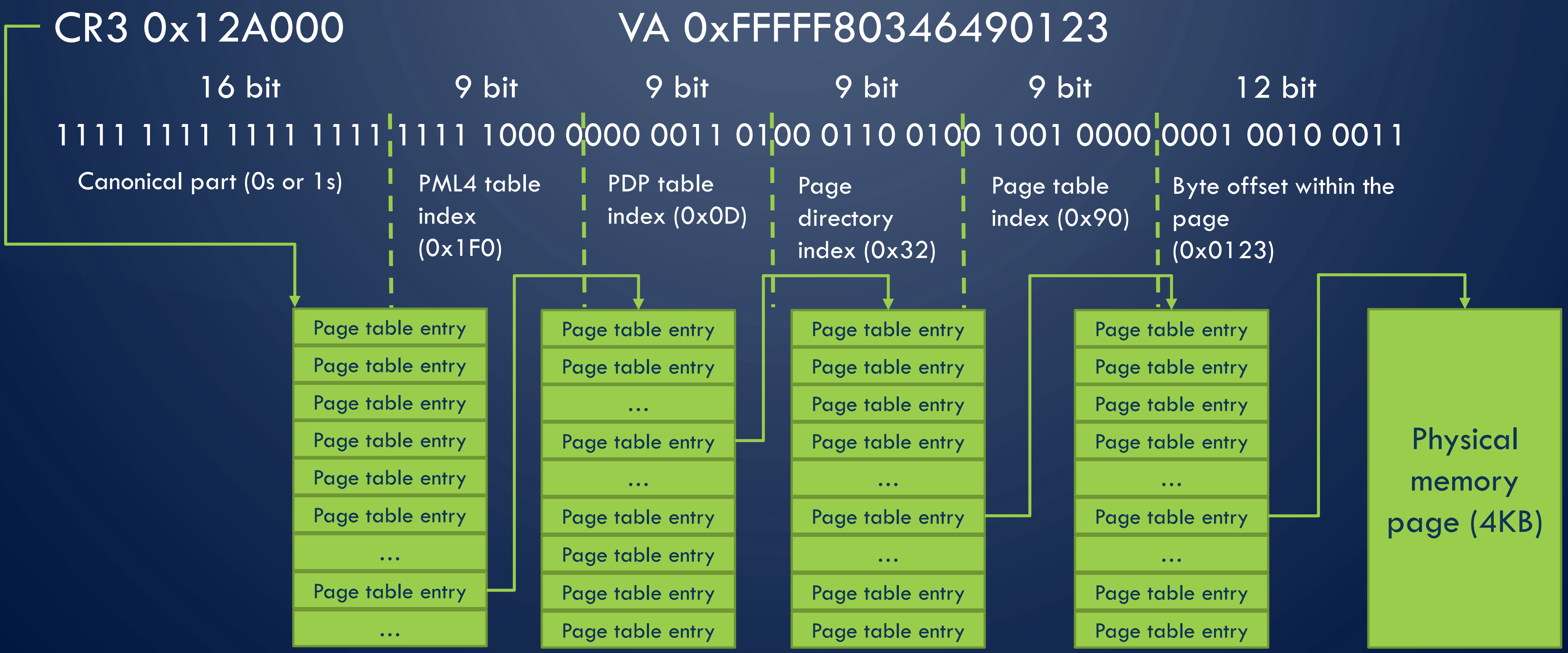
IMPLEMENTATION

IT’S ALWAYS ABOUT THE DETAILS

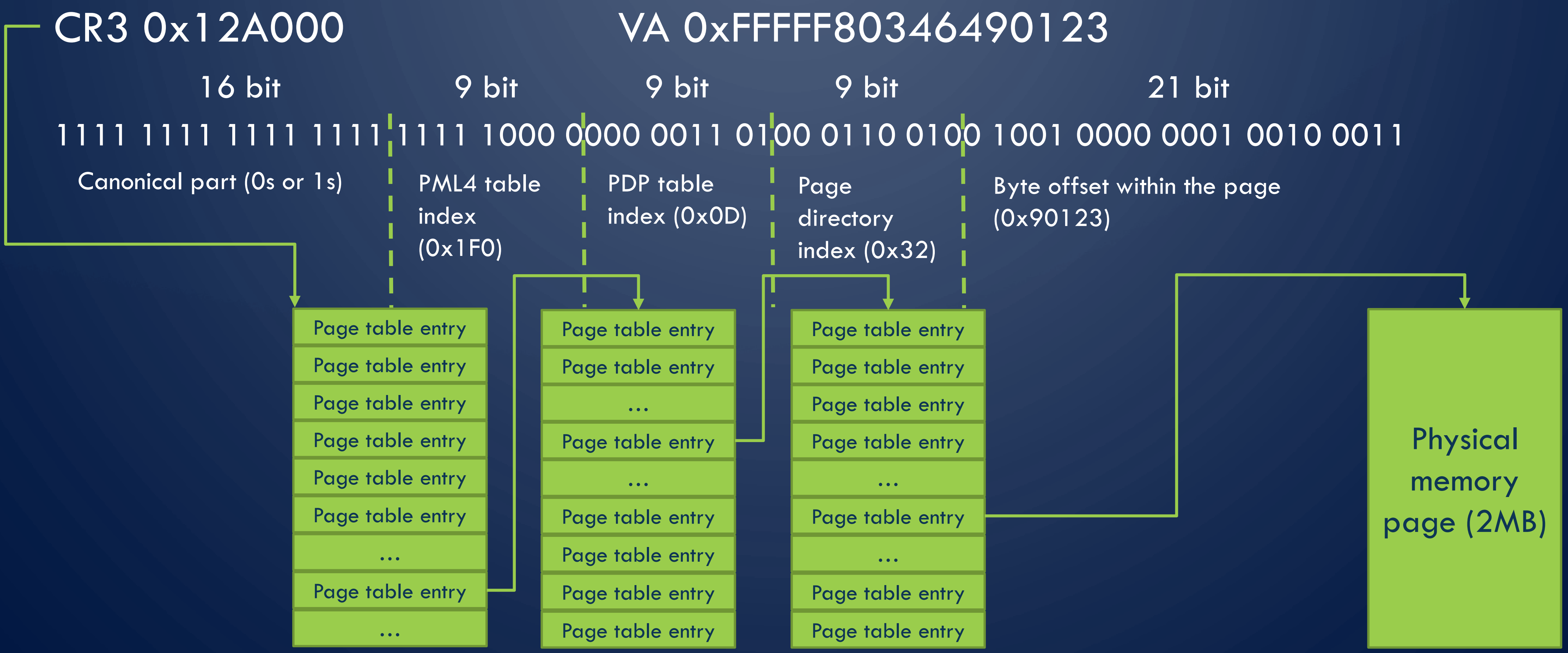
X86-64, LONG MODE PAGING

- Implemented as 4 level page tables (48 bit virtual address)
    - 5 level available recently for long mode (56 bit virtual address)
- Root is stored in a CR3 register
- Pages can map several memory chunk sizes depending on CPU capabilities
    - 1 gigabytes
    - 2 megabytes
    - 4 kilobytes

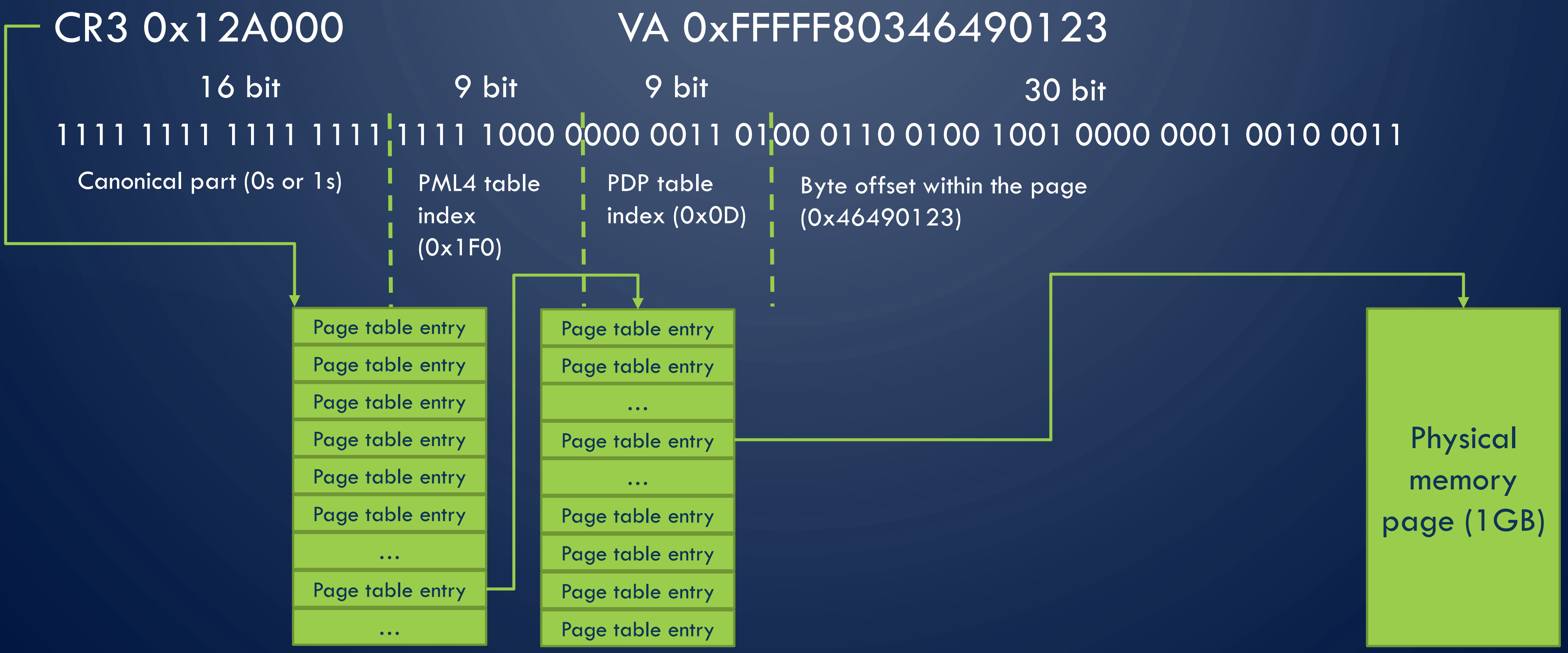
X86-64, LONG MODE PAGING



X86-64, LONG MODE PAGING



X86-64, LONG MODE PAGING



PAGE TABLE ENTRY

- Almost the same for every page table level
- Points to the next page table or a final physical page
- Stronger security attributes control the whole region described by the page table
  - Like R/W bit, disallowing writes for the whole 512 GB region if unset in PML4E
  - Or U/S bit, disallowing usermode access for the whole 1 GB region if unset in PDPTE
  - Or XD bit, disallowing instruction fetches from 2 MB region if set in PDE

PAGE TABLE ENTRY

XD	Prot. Key	Ignored	Rsvd.	Address of a page frame	Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	P	**PTE: 4KB page**

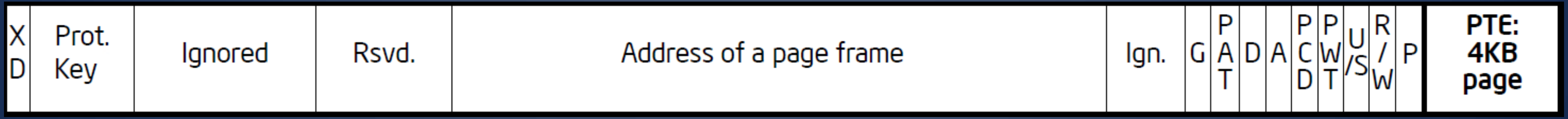
P – Present
R/W – Read / Write
U/S – User / Supervisor
PWT – Page Level Write Through
PCD – Page Level Cache Disable
A – Accessed
D – Dirty
PAT – Page Attribute Table
G – Global
XD – Execute disable

MAPPING THE PAGE

XD	Prot. Key	Ignored	Rsvd.	Address of a page frame	Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	P	**PTE: 4KB page**

P – Present
R/W – Read / Write
U/S – User / Supervisor
PWT – Page Level Write Through
PCD – Page Level Cache Disable
A – Accessed
D – Dirty
PAT – Page Attribute Table
G – Global
XD – Execute disable

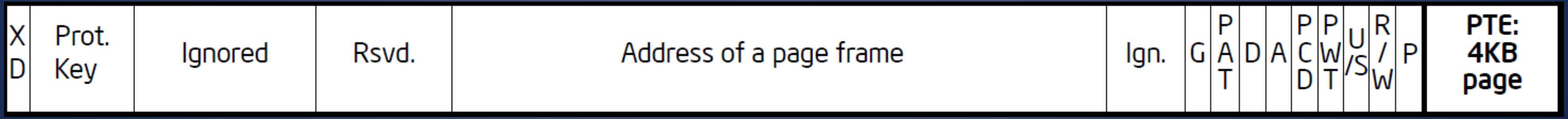
PROTECTION



XD	Prot. Key	Ignored	Rsvd.	Address of a page frame	Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	P	**PTE: 4KB page**

P – Present
R/W – Read / Write
U/S – User / Supervisor
PWT – Page Level Write Through
PCD – Page Level Cache Disable
A – Accessed
D – Dirty
PAT – Page Attribute Table
G – Global
XD – Execute disable

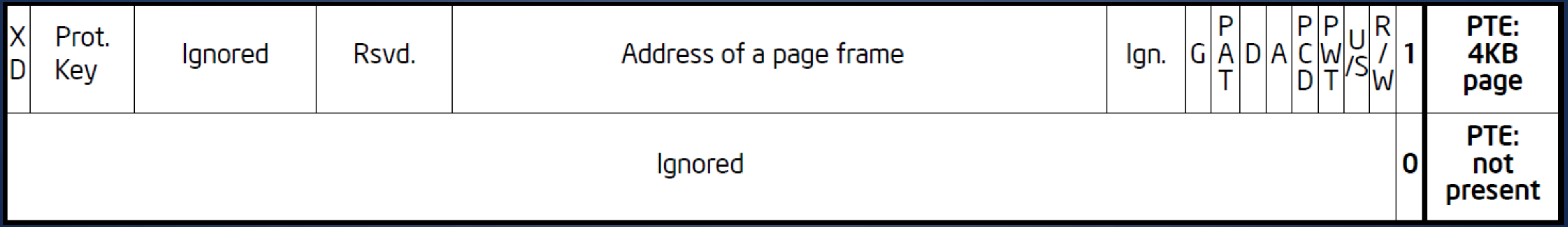
CACHING



XD	Prot. Key	Ignored	Rsvd.	Address of a page frame	Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	P	**PTE: 4KB page**

P – Present
R/W – Read / Write
U/S – User / Supervisor
PWT – Page Level Write Through
PCD – Page Level Cache Disable
A – Accessed
D – Dirty
PAT – Page Attribute Table
G – Global
XD – Execute disable

IGNORED VS RESERVED



XD	Prot. Key	Ignored	Rsvd.	Address of a page frame	Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	**1**	**PTE: 4KB page**
Ignored														**0**	**PTE: not present**

- Reserved – not to be modified
- Ignored – feel free to modify
- Software PTEs for memory manager
- Working set hash table index

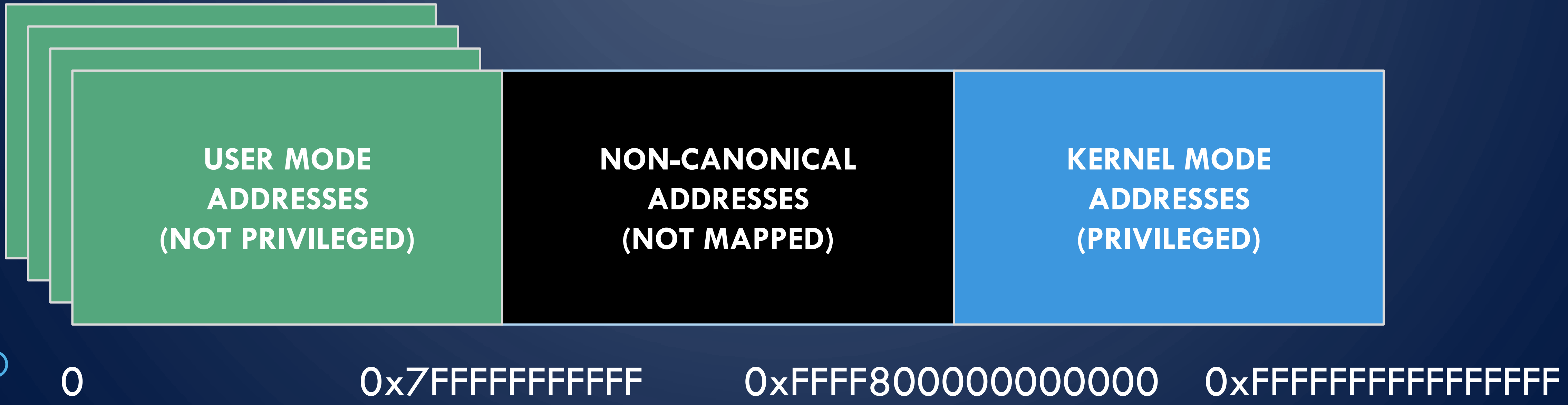
SCANNING PAGE TABLES

- I suggest relying on hardware PTEs
  - This ensures mapping working in hardware
  - Skip memory manager semantics
- Page tables walk
  - Like a tree walk
- Parse PTEs properly
  - Check the page size
  - Check if the page is mapped

STATIC PAGE TABLE ANALYSIS

WHAT DO WE LOOK FOR

USER VS KERNEL


USER MODE ADDRESSES (NOT PRIVILEGED)
NON-CANONICAL ADDRESSES (NOT MAPPED)
KERNEL MODE ADDRESSES (PRIVILEGED)
0
0x7FFFFFFFFFFF
0xFFFF800000000000
0xFFFFFFFFFFFFFFFF

USER VS KERNEL

- Check if U/S bit is set correctly
- Do you think it is a naive idea?
  - Well, check CVE-2018-1038, a single incorrectly set bit led to overwriting page tables from usermode

W^X

- Check if the page is writable and executable at the same time
- Useful for the latest Windows kernel exploits
- Currently some memory is mapped as WX
  - UEFI runtime services
  - Part of ntoskrnl image
  - Custom driver allocations
  - Drivers that use legacy MmMapIoSpace (vs MmMapIoSpaceEx)
- Virtualization-based security prevents W^X at hypervisor level using extended page tables

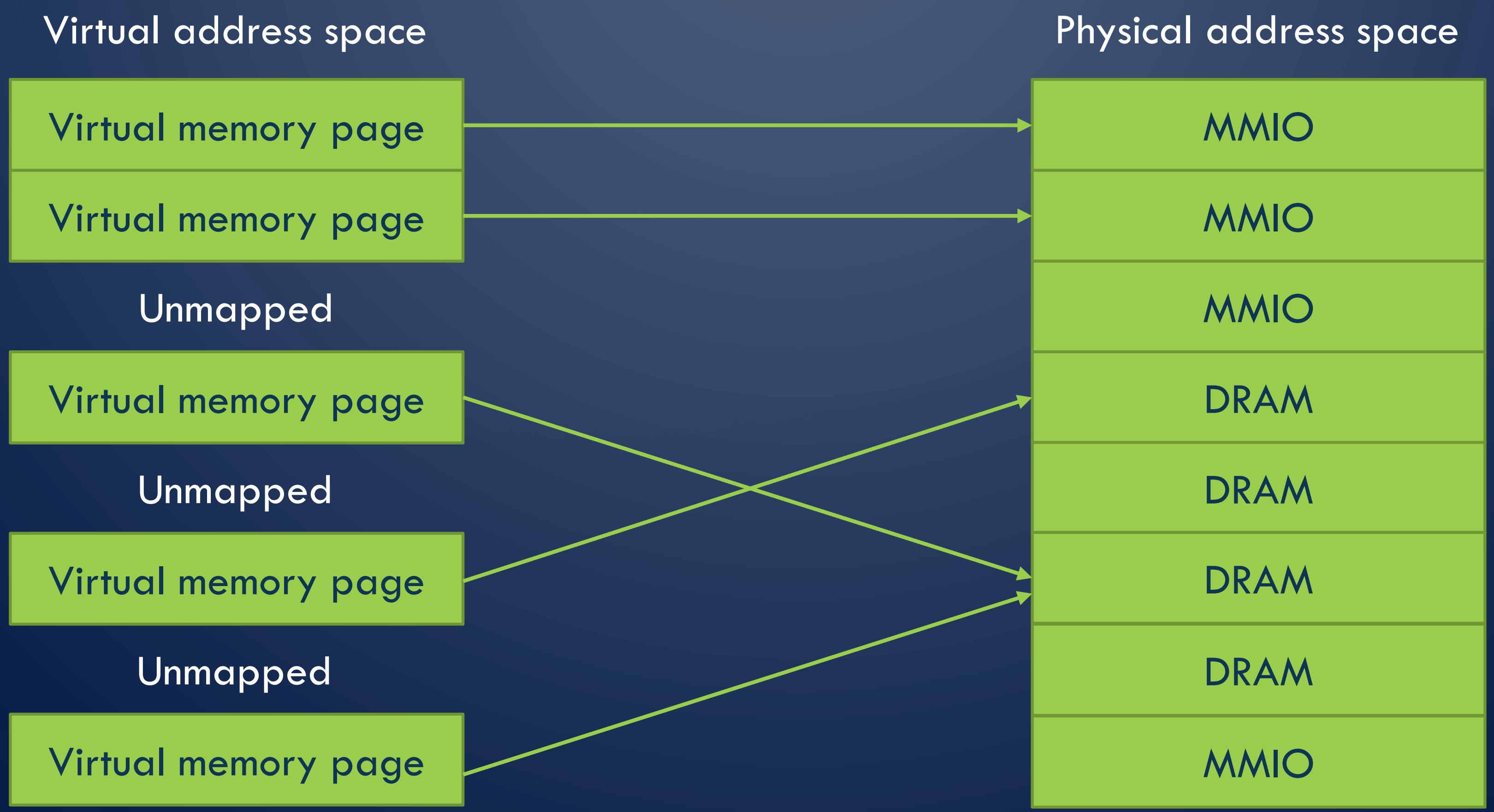
UEFI RANGE

- Basically mapped as RWX
  - Depends on a firmware
  - Modern MS Surface systems support page protection
- Function pointer table at hal!HalEfiRuntimeServicesBlock
  - Can be triggered by reading UEFI variable for example
- Blocked by VBS

SEMANTIC GAP

- PVOID MmMapIoSpace(PHYSICAL_ADDRESS PhysicalAddress, SIZE_T NumberOfBytes, MEMORY_CACHING_TYPE CacheType);
- PVOID MmMapIoSpaceEx(PHYSICAL_ADDRESS PhysicalAddress, SIZE_T NumberOfBytes, ULONG Protect);
- MmMapIoSpace goes to MmMapIoSpaceEx
  - Caching parameter implicitly converted to Protection parameter
  - MmNonCached converted to PAGE_NOCACHE | PAGE_EXECUTE_READWRITE
  - MmCached converted to PAGE_EXECUTE_READWRITE
  - MmWriteCombined converted to PAGE_WRITECOMBINE | PAGE_READWRITE

ACCESS TO THE HARDWARE



ACCESS TO THE HARDWARE

- Check caching settings for the mapping
  - It is common that MMIO is mapped as non-cached
- Drivers can expose MMIOs to unprivileged code
- MMIO controls the hardware directly
- Immediate LPE if DMA engine is exposed
- https://access.redhat.com/security/cve/cve-2010-5313

SHARED MAPPINGS

- Cross-process
- Cross-mode
- Check if less privileged mode has higher privileged mapping
  - User can access sensitive kernel data, which leads to info leak
  - User can unexpectedly modify privileged mapping data, which leads to LPE
- Example: KUSER_SHARED_DATA
  - Read-only for user mode, Read-Write for kernel mode, no sensitive info

DYNAMIC PAGE TABLE ISSUES

WATCH THE CACHES

TLB AND PAGING STRUCTURE CACHES

- TLB - Translation lookaside buffers
- Per CPU thread
- Cache translation information
    - PFN address
    - R/W bit
    - U/S bit
    - XD bit
    - Protection keys
    - Dirty flag
    - Memory type

SHARED RESOURCE

- Page tables occupy a lot of memory
  - Mapping 16 GB of memory requires ~32 MB of memory to map as 4KB pages
  - Often shared between processes / modes
- We have a publicly documented OS APIs to change mapping attributes
- Memory manager is supposed to synchronize memory view across the CPU threads

SHARED RESOURCE

- Synchronization failure leads to race conditions
  - Example: CVE-2018-18281
- Check if caches are flushed and synchronized
  - Not trivial
  - But worth it

REFERENCES

- Intel Software Developers Manual
  - https://software.intel.com/en-us/articles/intel-sdm
- Can you break Windows Page Table Isolation?
  - https://nadav.amit.zone/windows/2018/09/15/windows-pti.html
- Taking a page from the kernel's book: A TLB issue in mremap()
  - https://googleprojectzero.blogspot.com/2019/01/taking-page-from-kernels-book-tlb-issue.html
- Total meltdown
  - http://blog.frizk.net/2018/03/total-meltdown.html

THANKS AND HAPPY FISHING!

QUESTIONS?