Virtual Memory

Modern Operating Systems, by Andrew Tanenbaum Chap 3

Operating Systems: Three Easy Pieces Chap 13-24

https://en.wikipedia.org/wiki/Page_(computer_memory)

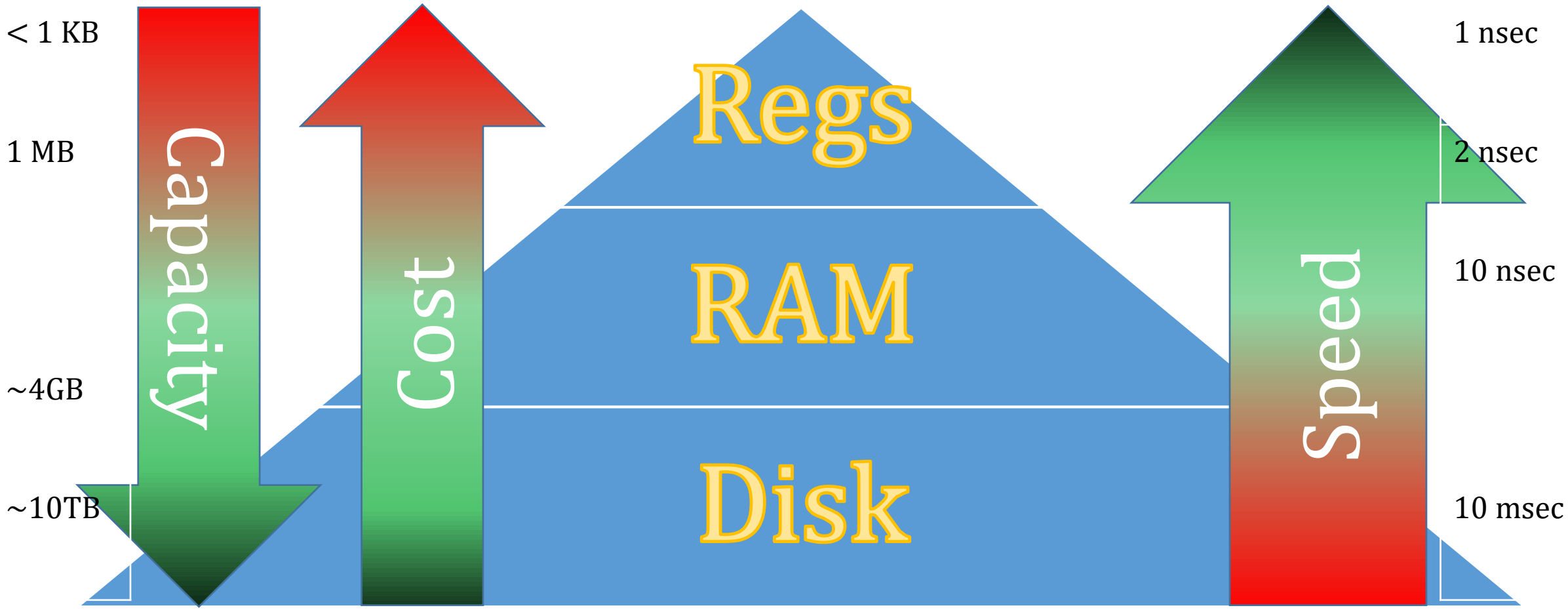
https://en.wikipedia.org/wiki/Page_table

https://en.wikipedia.org/wiki/Virtual_memory

Memory Management

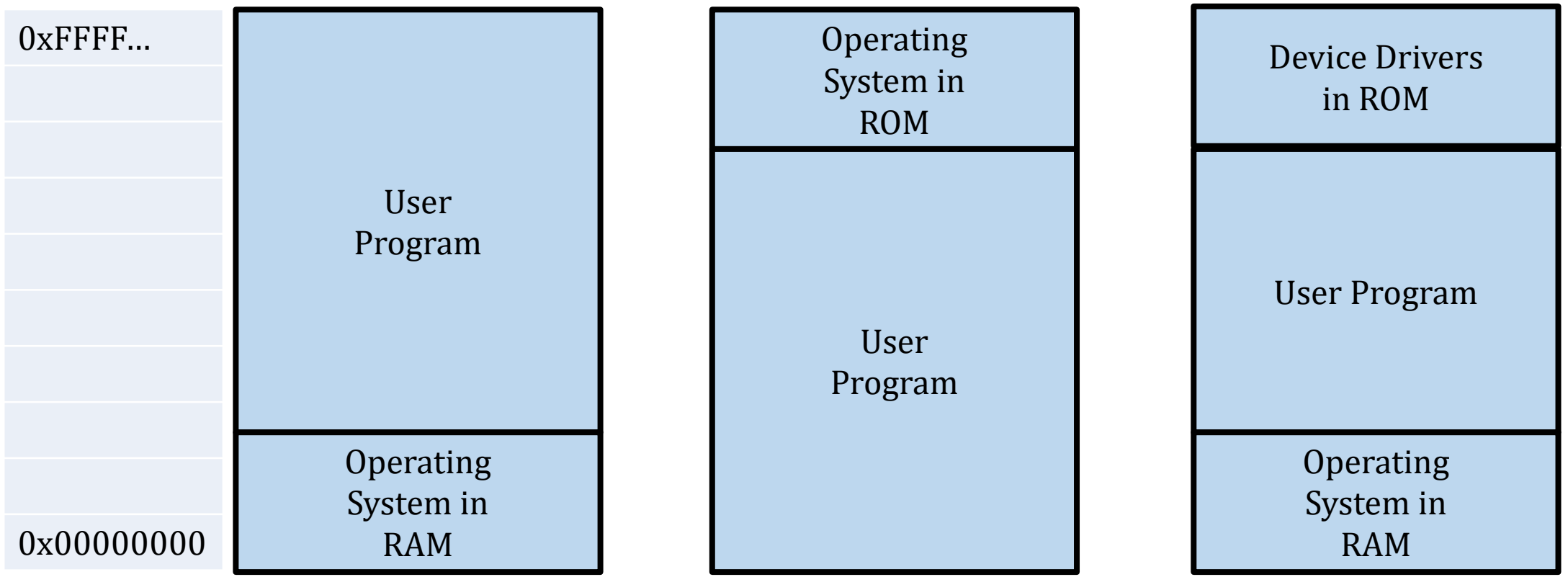
- Ideally, programmers want memory that is:
  - Large
  - Fast
  - Persistent (non-volatile)

The Memory Hierarchy

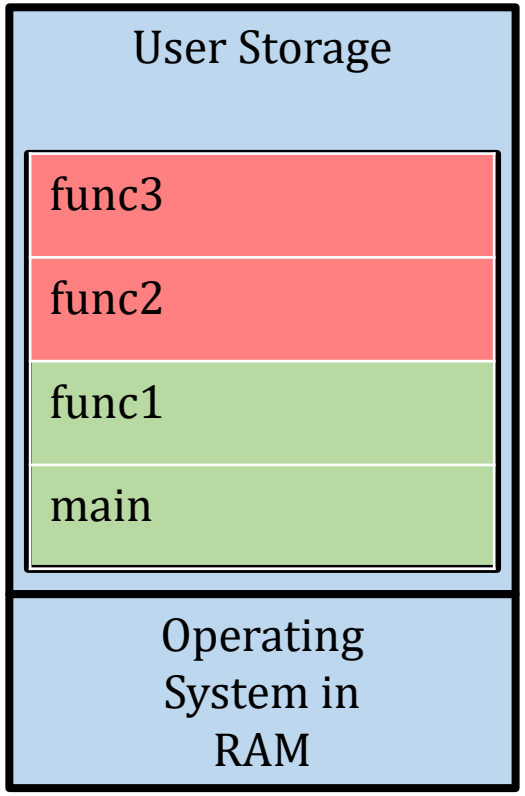


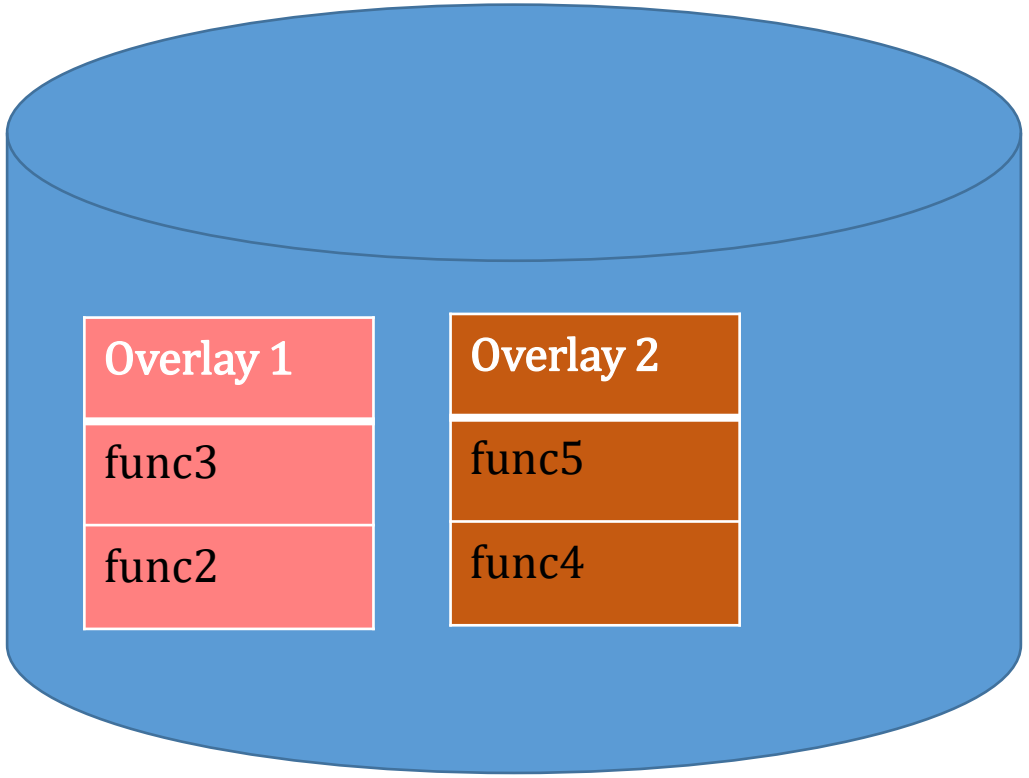
Basic Memory Management

- Assumption: Single user process must share memory with OS

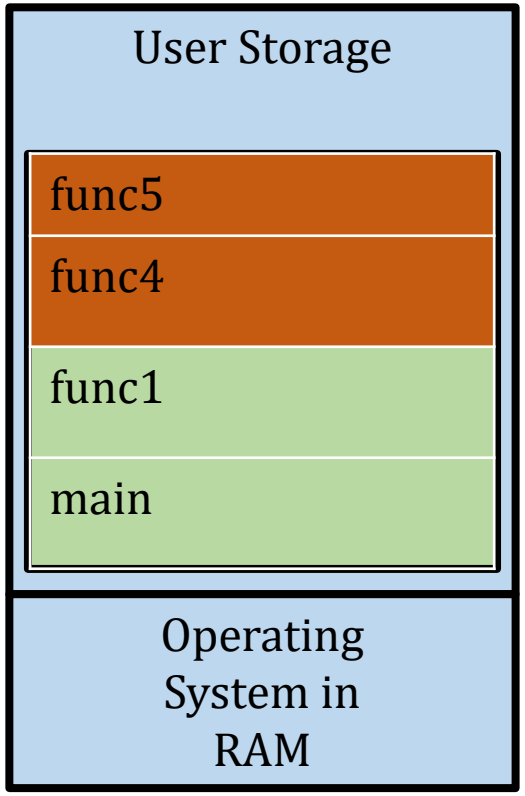


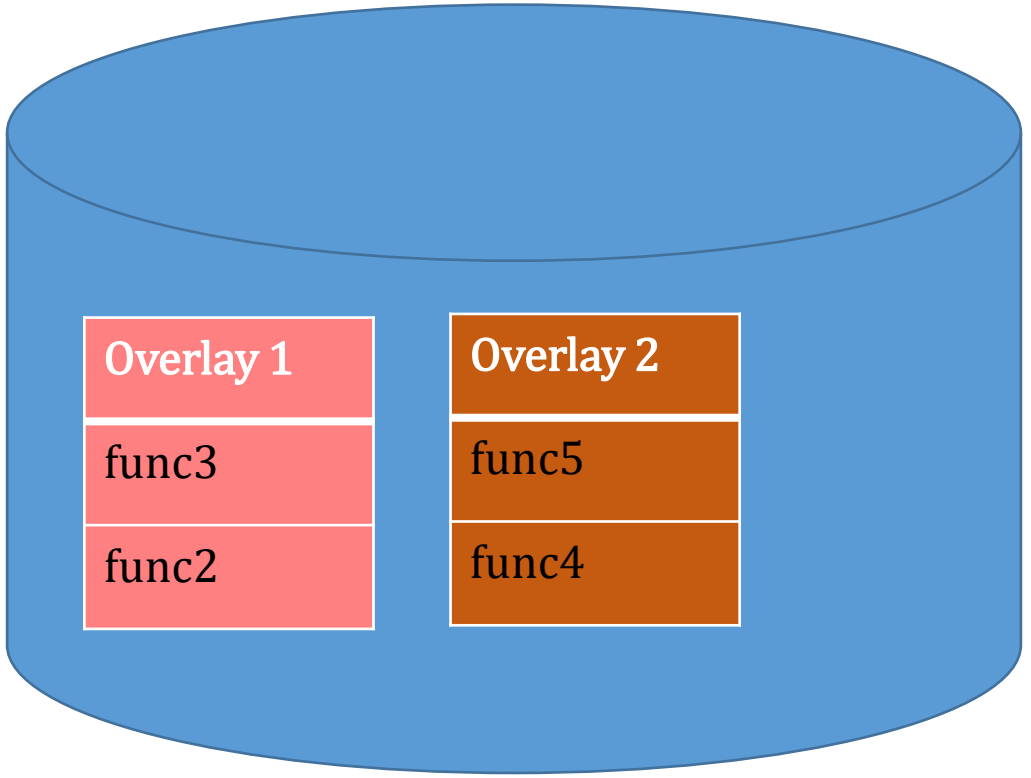
Mainframe Memory Mgmt: Overlays





Mainframe Memory Mgmt: Overlays

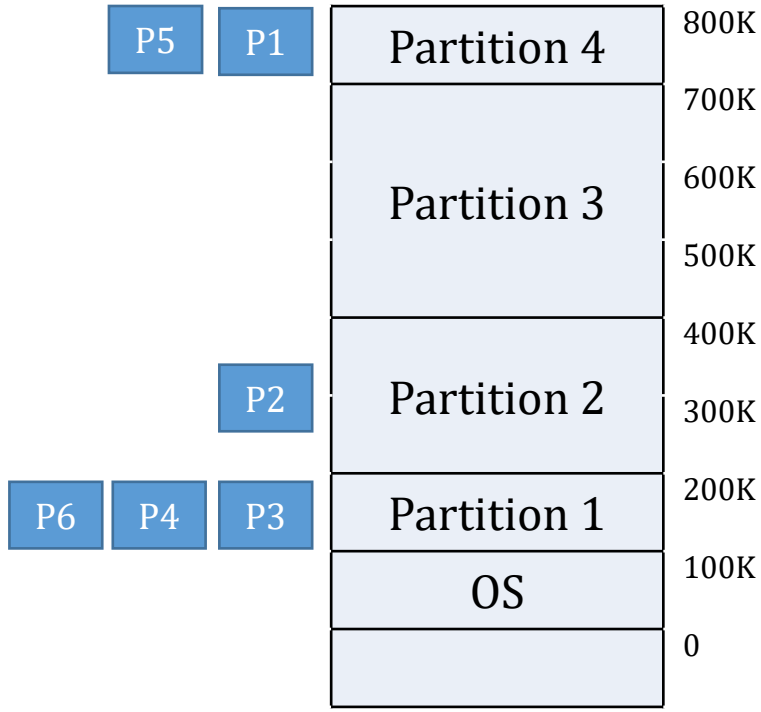




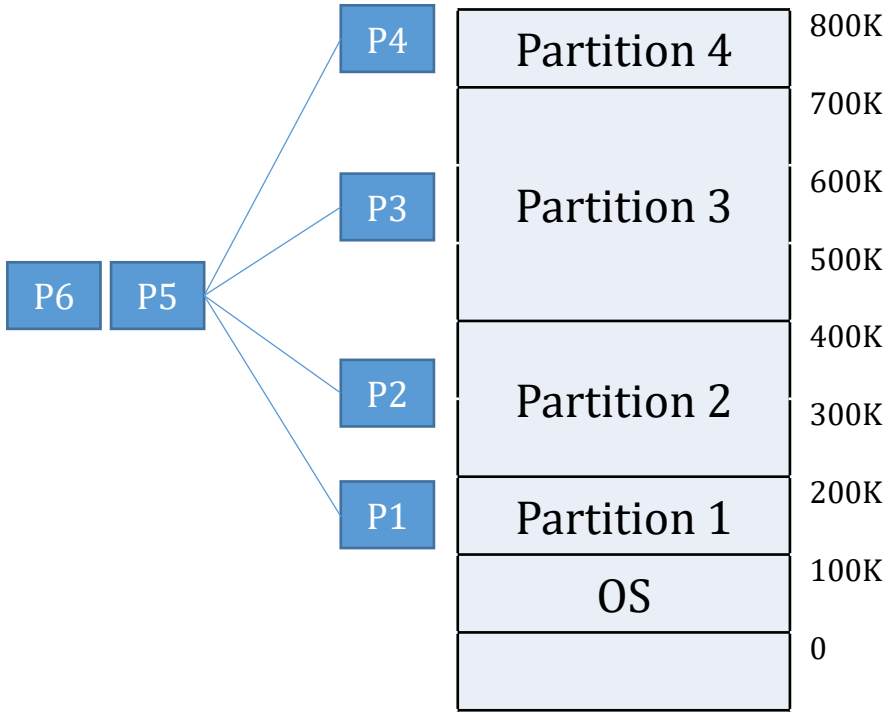
Fixed Partition Memory Management

- Assumes multiple CPUs working on a single memory

Multiple Input Queues (Supermarket Checkout)



Single Input Queue (Dept. Store Checkout)



Relocation

- How do we write programs that can get loaded in different partitions?
  - load Rx,memory_address

Programmer doesn't know where the data will be stored!

- Solution: Relocation
  - Program a relative address – from the beginning of the program
  - Operating system must calculate the physical address
    - Program loaded in Partition 2, so base is 200K
    - load RX,+0x3A
    - OS Calculates physical address 200K + 3A = 20003A

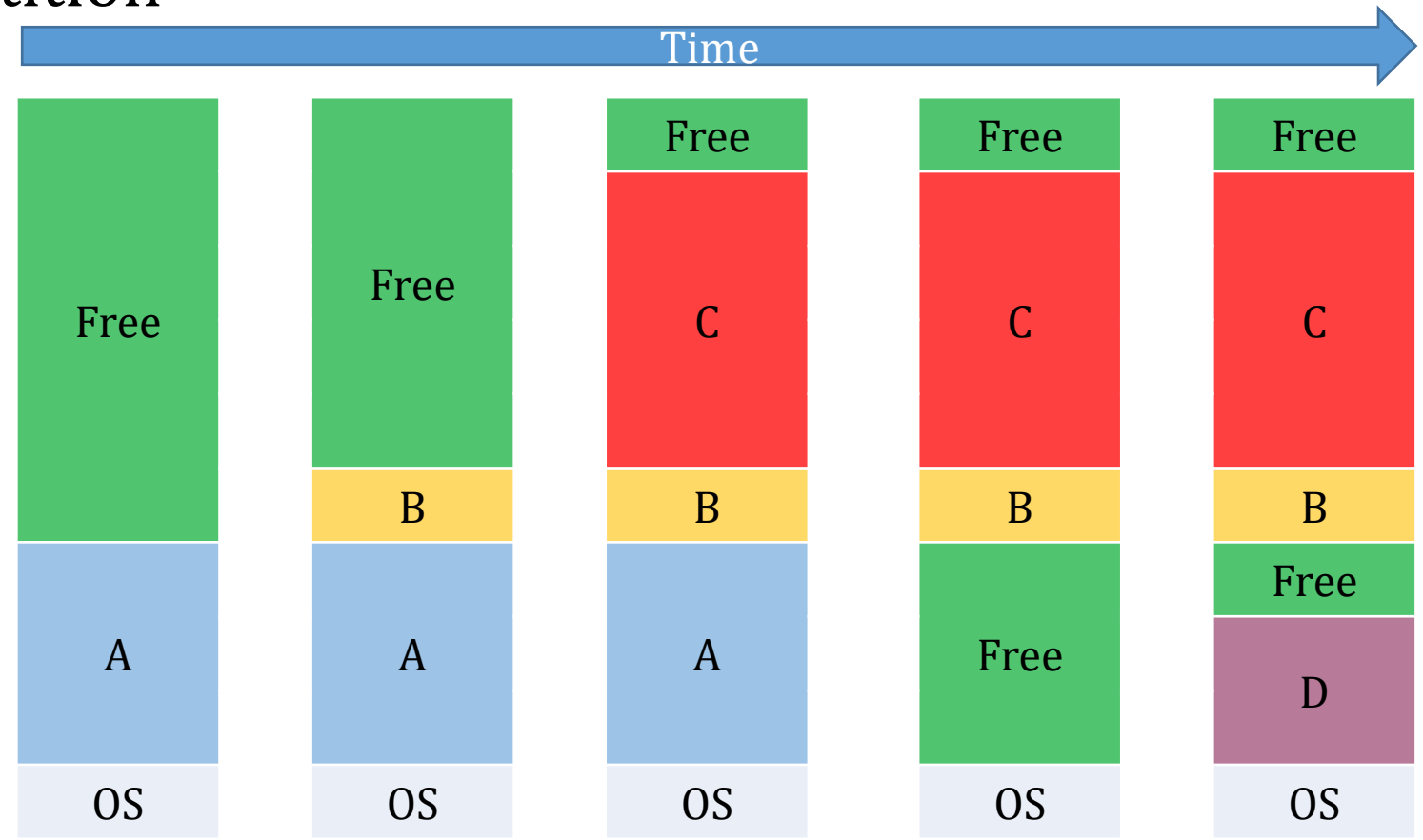
Protecting Relocation

- When program running in partition 2 asks for memory, the OS should prevent getting values from partition 3

- Solution: OS keeps base and limit of the partition where program is loaded
  - e.g. Partition 2 – base is 200K, limit is 400K
  - If physical address is greater than limit, then raise an error (segmentation violation)

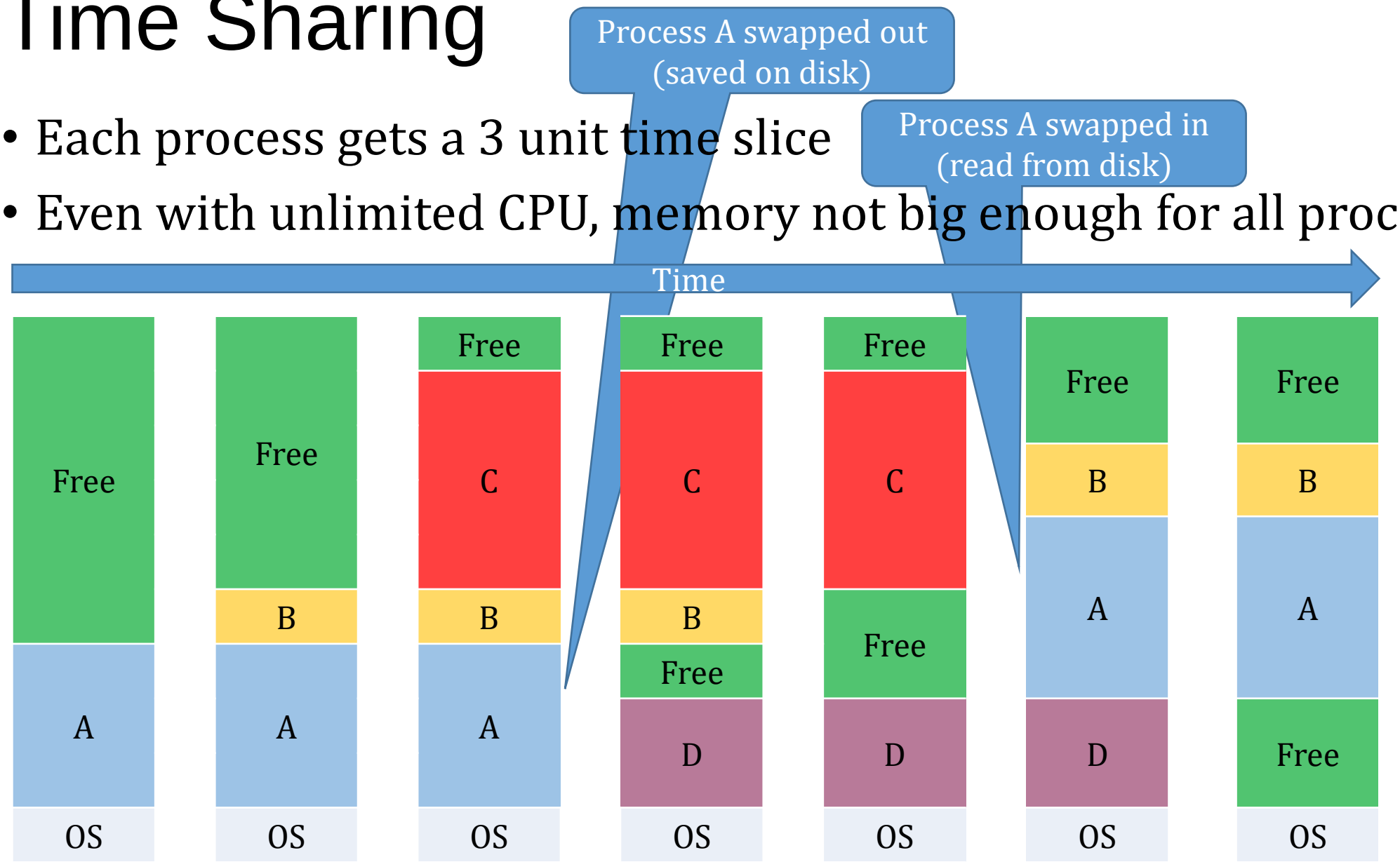
Variable Partition Size

- Make the partition just big enough to hold the process running in that partition



Time Sharing

- Each process gets a 3 unit time slice
- Even with unlimited CPU, memory not big enough for all processes

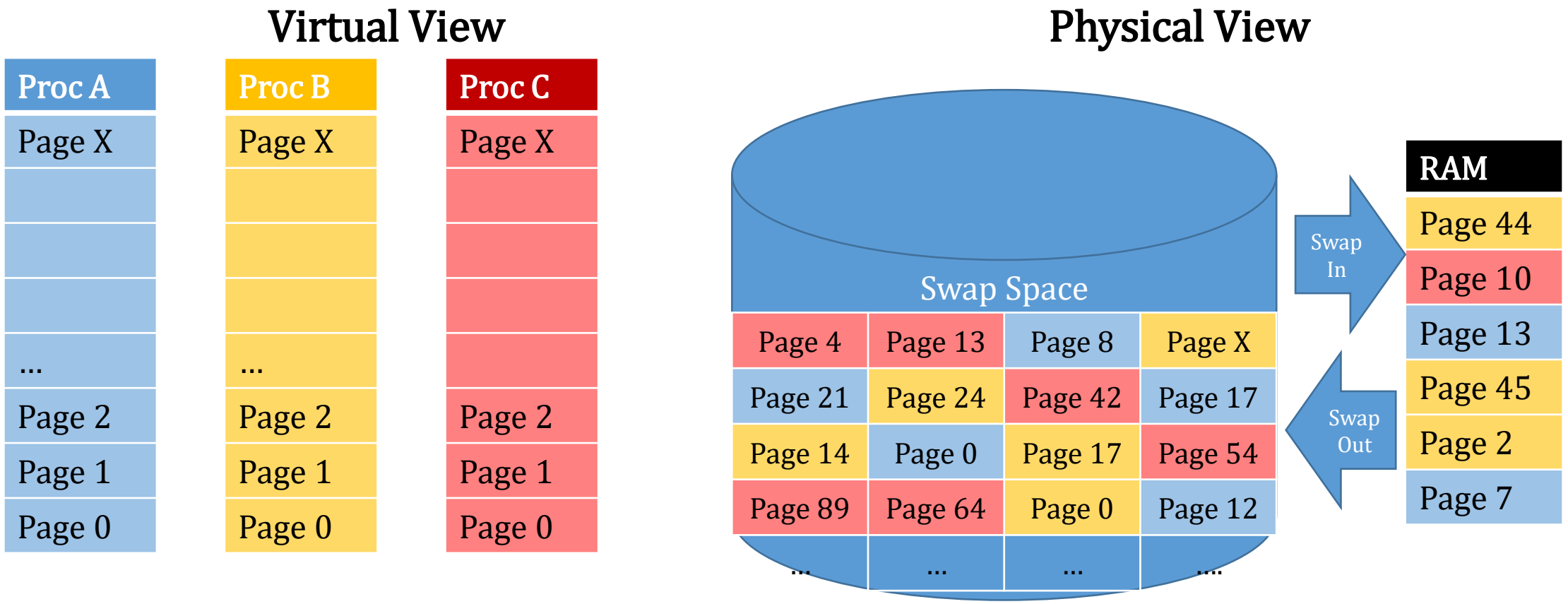


Virtual Memory

- A 64 bit address can select any one of 2^{64} bytes of memory
- Physical Memory is typically much less: 8G=2^{33} bytes of memory
- In fact, a SINGLE PROCESS often does not fit in virtual memory

- Solution: Divide each process memory address space into "pages" of memory
- OS decides which pages stay in real memory, and which get saved on disk
- Each process gets the *illusion* that it has more memory than the physical RAM

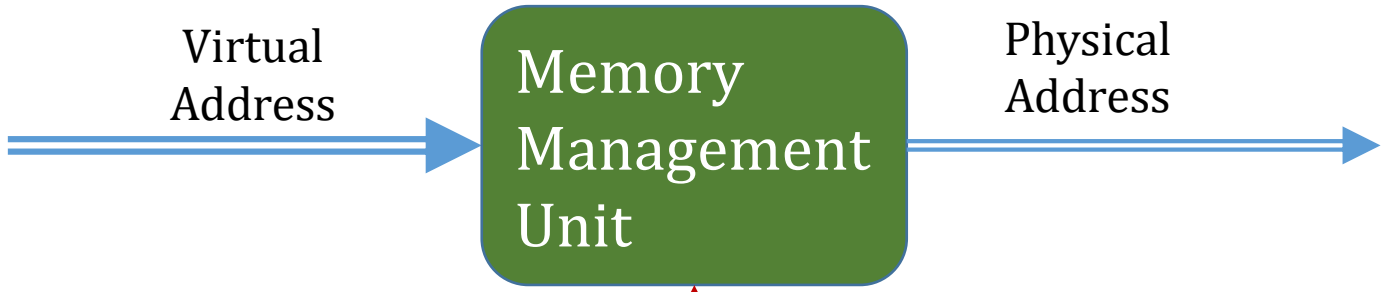
Virtual Memory



Virtual Memory Shell Game

Proc	Page	Slot
B	45	2
B	2	1
A	7	0

Slot	RAM
5	Page 44
4	Page 10
3	Page 13
2	Page 45
1	Page 2
0	Page 7



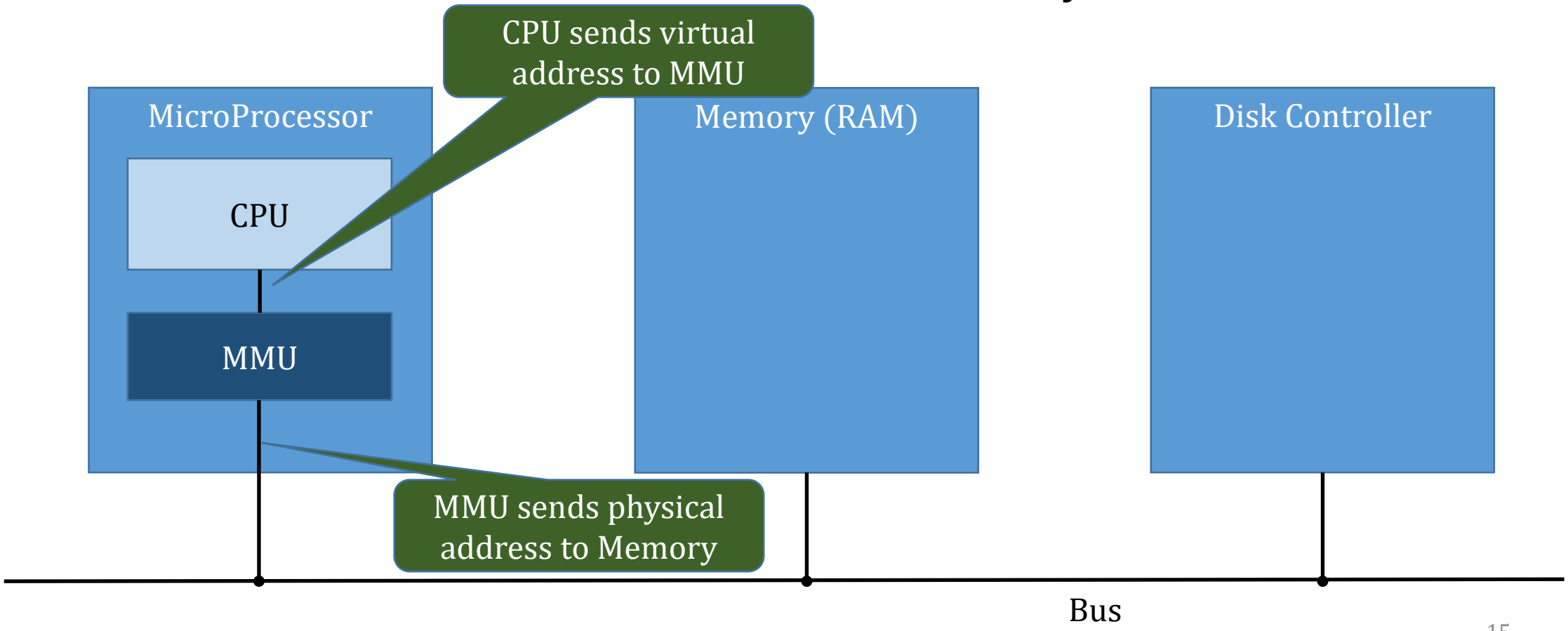
2^{63}	2^{62}	...	2^{12}	2^{11}	2^{10}	...	2^{2}	2^{1}	2^{0}
b_{63}	b_{62}	...	b_{12}	b_{11}	b_{10}	...	b_{2}	b_{1}	b_{0}
Page ID				Page Offset					

2^{32}	2^{31}	...	2^{12}	2^{11}	2^{10}	...	2^{2}	2^{1}	2^{0}
r_{32}	r_{31}	...	r_{12}	b_{11}	b_{10}	...	b_{2}	b_{1}	b_{0}
Real Page Slot				Page Offset					

Page Fault

Memory Management Unit (MMU)

- Hardware that translates Virtual Address to Physical Address

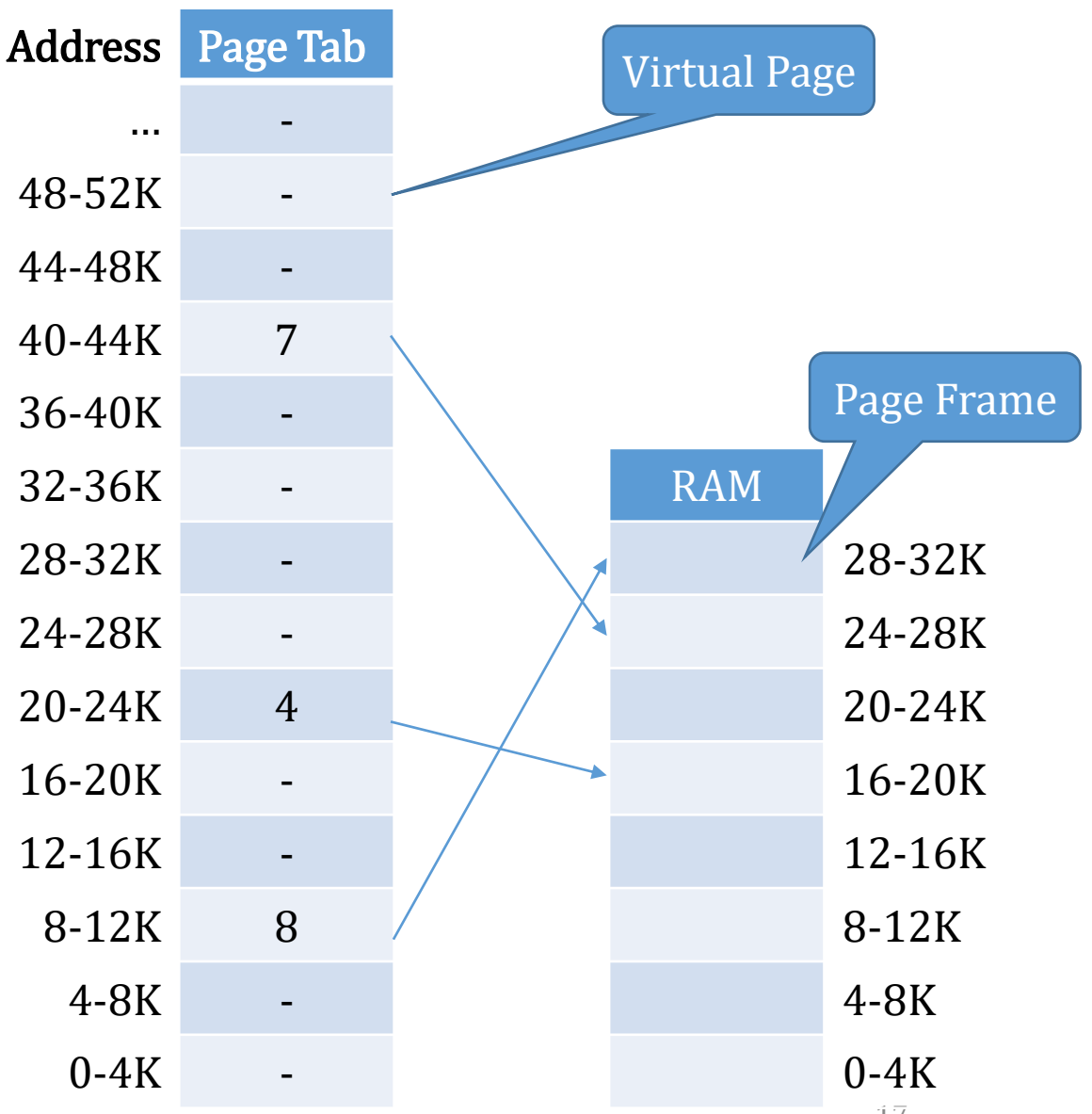


Address / Address Space Size

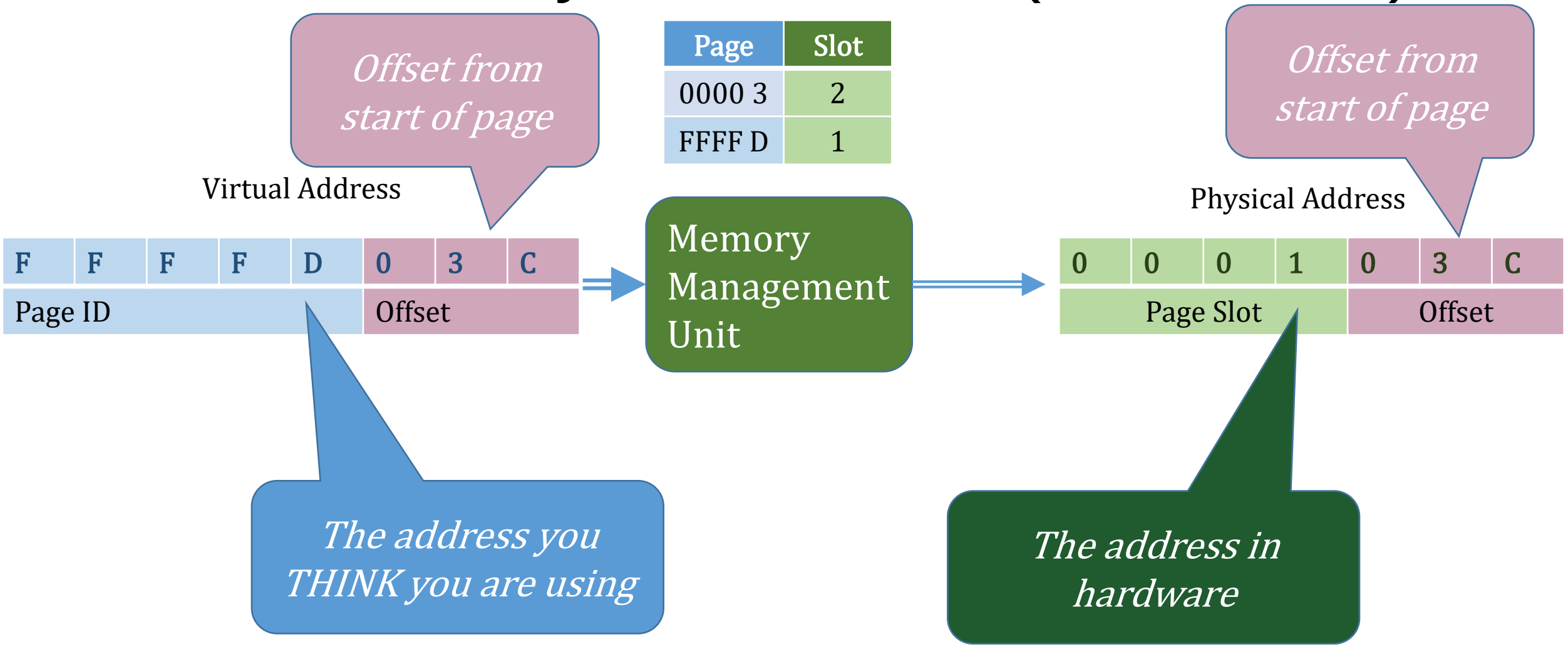
Number of bits in address	Address Space Size
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
10	$2^{10} = 1024 = 1\text{K}$
12	$2^{12} = 4096 = 4\text{K}$
16	$2^{16} = 65{,}536 = 64\text{K}$
32	$2^{32} = 4{,}284{,}481{,}536 = 4\text{G}$
64	2^{64} = very big number = 16EB (Exbibytes)

Page Table

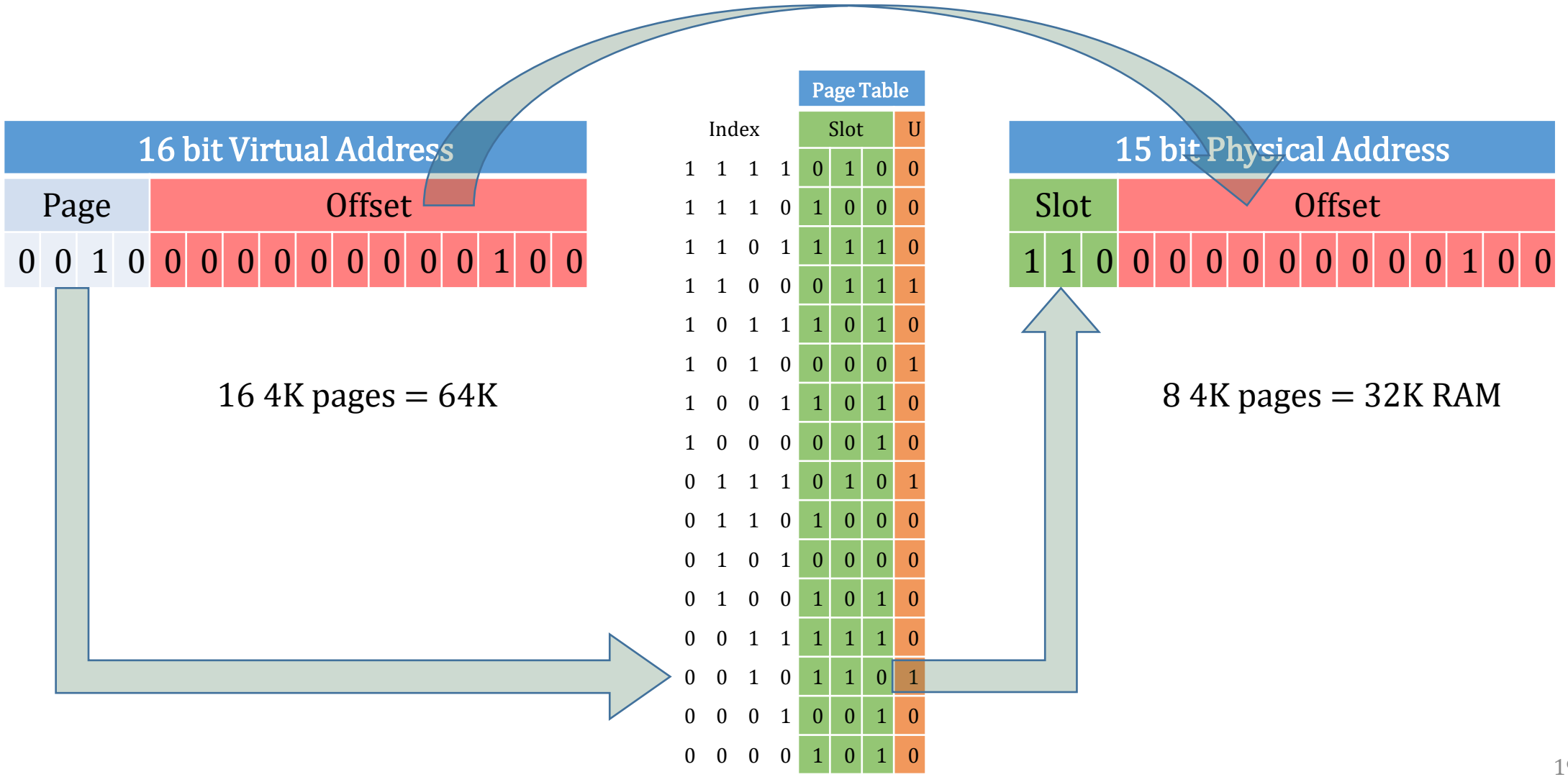
- An array that stores mapping from virtual page numbers to slots in physical RAM
- The OS maintains:
  - One page table per user process,
    and
  - Another page table for kernel memory



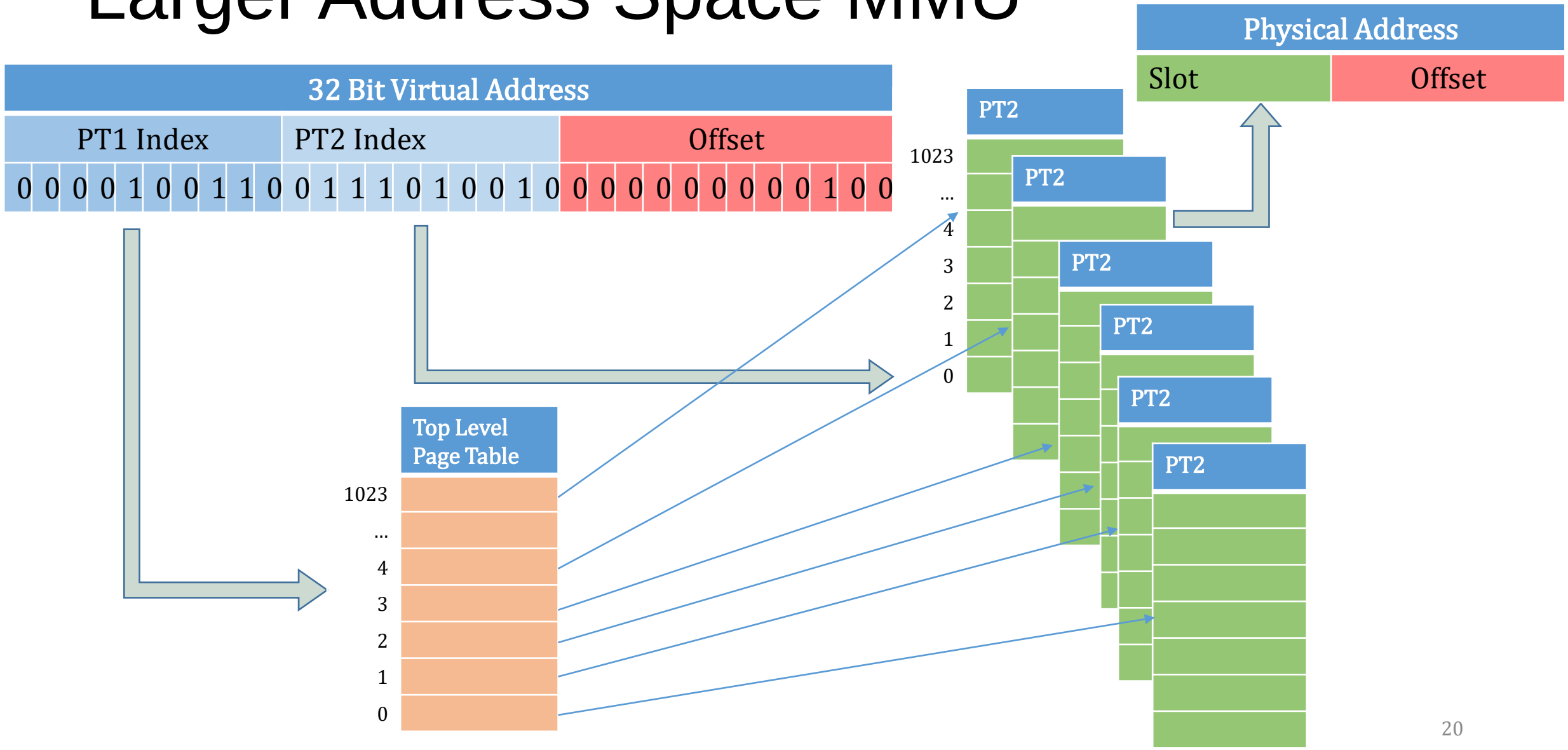
Virtual Memory Shell Game (32 bit addr)



Small Address Space MMU



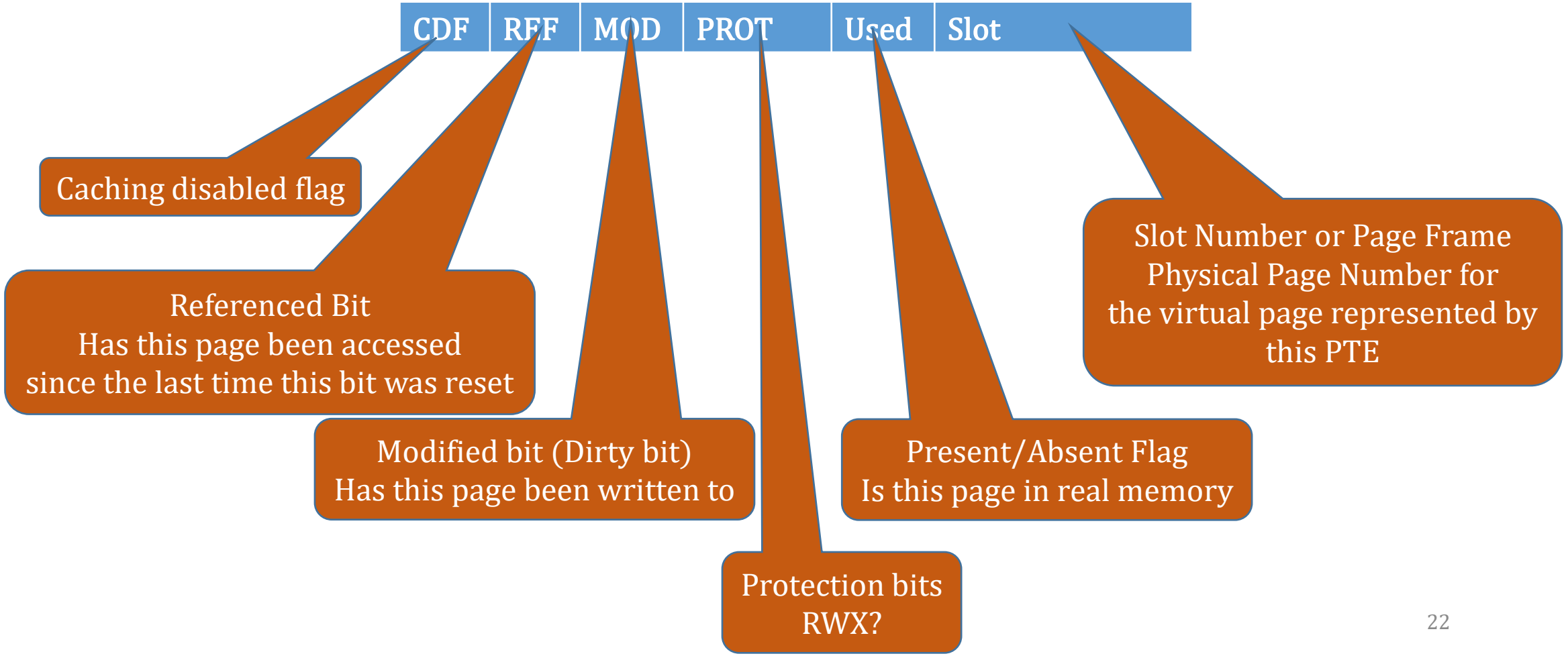
Larger Address Space MMU



Larger Address Space MMU

- Two level page tables
- Page tables are too big to fit in RAM inside the MMU
  - Kept in main memory
- MMU finds Page Table 1 via Registers (CR2 in Intel)

Typical Page Table Entry (PTE)



Translation Lookaside Buffers (TLBs)

- A TLB is a small cache that contains some Page Table Entries
- Used to speed up translation of virtual to physical addresses
- TLB is part of the MMU hardware (packaged on CPU chip)
- TLB is NOT a data or instruction cache
- On older x86 processors, TLB "flushed" every context switch
  - No field in TLB to identify process
  - Tagged TLB's contain process to reduce this overhead

Valid	Virt.Page	Mod	Prot	Page Frame
1	0x0C2	1	RW	0x1F
1	0x014	0	R X	0x26
1	0x0B4	1	RW	0x1D
1	0x0B3	1	RW	0x3E
1	0x013	0	R X	0x32
1	0x015	0	R X	0x2D

Cold Start Penalty

- Cost of repopulating TLB (and other caches) when context switches
- After a context switch, many (or all) TLB entries are invalidated
  - Older TLB without process tags… entire TLB flushed
- New memory accesses by the new process probably cause a TLB miss
- MMU must then lookup the page table in main memory to populate the missing TLB entry
  - Takes much more time than a cache hit

Tagged TLB

- A "tag" in each TLB entry identifies the process/thread context to which the TLB belongs
- TLB entries for multiple processes/threads can be stored simultaneously in the TLB
  - TLB lookup checks the tag as well as the virtual page number
- With tags, context switch no longer requires TLB flush
  - Reduces cold start penalty

Memory Translation Architectures

Architected Page Tables

- Page table interface defined by ISA and understood by MMU
- E.g. x86 architecture
- TLB miss handled by MMU in hardware
- Page faults handled by OS in software
- ISA specifies page table format

Architected TLBs

- TLB interface defined by ISA and understood by MMU
- E.g. alpha architecture
- TLB miss handled by OS in software
- Page faults handled by OS in software
- ISA does not specify page table format

Page Size impact on Page Tables

Small Page size

- Less internal fragmentation
- Page swap less expensive
- More pages
  - bigger page table
  - Smaller "TLB coverage"

Large Page Size

- More internal fragmentation
- Page swap more expensive
- Fewer Pages
  - Smaller page table
  - Larger "TLB Coverage"

TLB Coverage

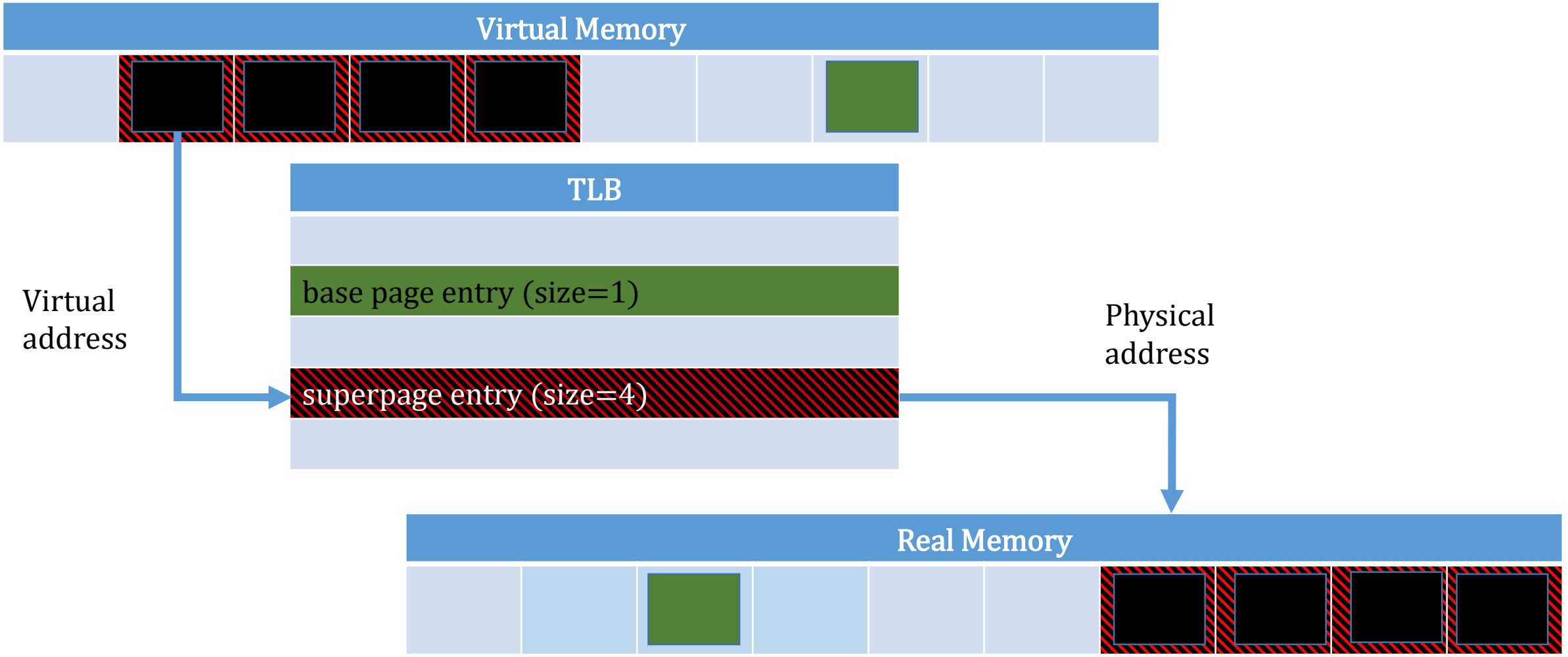
- Maximum amount of memory mapped by the TLB
  - Maximum amount of memory that can be accessed without TLB misses
- TLB Coverage = N x P bytes
  - N = Number of entries in the TLB
  - P = Page size in bytes
- N is fixed by hardware constraints… to increase coverage, we must increase P

P	Cov	Comments
1	N	Miss Miss Miss
4K	Nx4K	Pretty Low
2^{64}	$Nx2^{64}$	N address spaces with no misses! but impossible to swap!

Superpages

- Memory pages of larger sizes than the standard 4K pages
  - supported by most modern CPUs
- Superpage size = $2^{?}$ x base page size (usually 4K)
- Only one TLB entry per superpage
  - Multiple (identical) page table-entries – one per base (4K) page
- Constraints:
  - Contiguous physically and virtually
  - Aligned physically and virtually
  - Uniform protection attributes
  - One reference bit, one dirty bit

Superpage TLB



Ungraded Quiz

Consider a machine that has a 32 bit virtual address space and 8K page size

1. What is the total size (in bytes) of the virtual address space for each process?
2. How many bits in the 32 bit address are needed to determine the page number of the addres?
3. How many bits in the 32 bit address represent the byte offset into the page?
4. How many page table entries are present in the page table?

Quiz Answers

1. Total size of the address space = 2^{32}
   - 2^{10} =1024, so $2^{32} = 2^{22}$ K or 2^{12}M or 2^{2} G = 4G
2. How many bits for page numbers?
   - Number of pages = 2^{32} / 2^{3} * 2^{10} = 2^{19}, $\log_2(2^{19}) = 19$
3. How many bits for offsets?
   - 32-19=13 or
   - $\log_2(8K)=\log_2(2^{3}*2^{10})=13$
4. How many page-table entries?
   - Same as number of pages… 2^{19}