Computer Architecture: Fundamentals, Tradeoffs, Challenges

Chapter 7: Virtual Memory

Yale Patt

The University of Texas at Austin

Austin, Texas

Spring, 2025

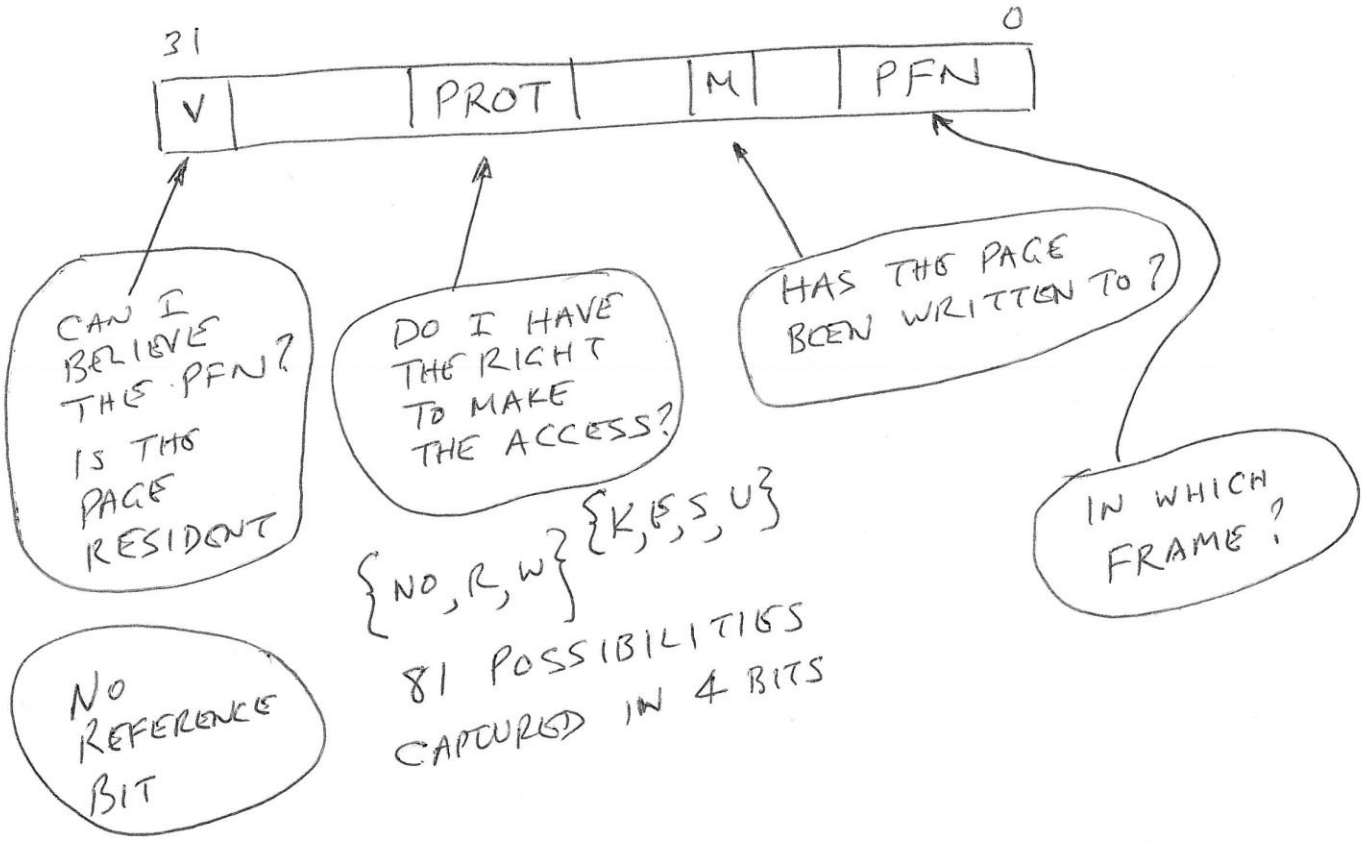
Outline

- *Virtual Memory Characteristics*
- *Pages and Page Tables*
- *Access Control and Translation*
- *Case 1: Process Page Table in physical memory*
- *Case 2: Process Page Table in system virtual memory*
- *Layout of VAX Virtual Memory*
- *How do we process: LD R1, X*
- *A complete example*
- *The Translation Lookaside Buffer (TLB)*
- *Granularity (i.e., page size) of memory that is transferred*

Virtual Memory

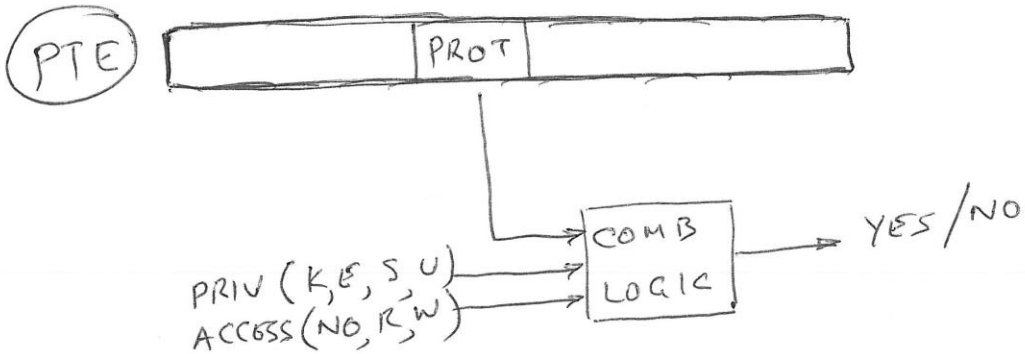
- ***ISA has a large virtual address (VA) space***
  - ***Allows the user program to uniquely identify all objects***
  - ***Required memory space partitioned into pages***
- ***Physical memory is usually smaller***
  - ***It is shared among all processes in the Balance Set***
  - ***Granularity is the frame. One resident page occupies one frame***
- ***Virtual memory management does two things***
  - ***Access Control and Translation***
- ***Requires cooperation of Architecture and O/S***
  - ***Microarchitecture provides the structures***
    - ***And executes the actual protection and translation code***
  - ***Operating System manages the memory***
    - ***What is resident (i.e., in physical memory), what is on the disk***
    - ***What gets kicked out of memory to handle page faults***

The Page Table Entry (PTE)
(A descriptor for each page)

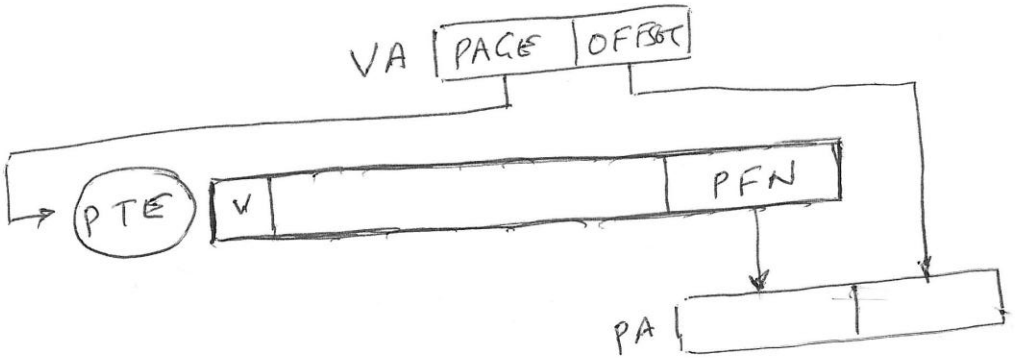


Access Control and Translate

- ***Access Control***



- ***Translation***

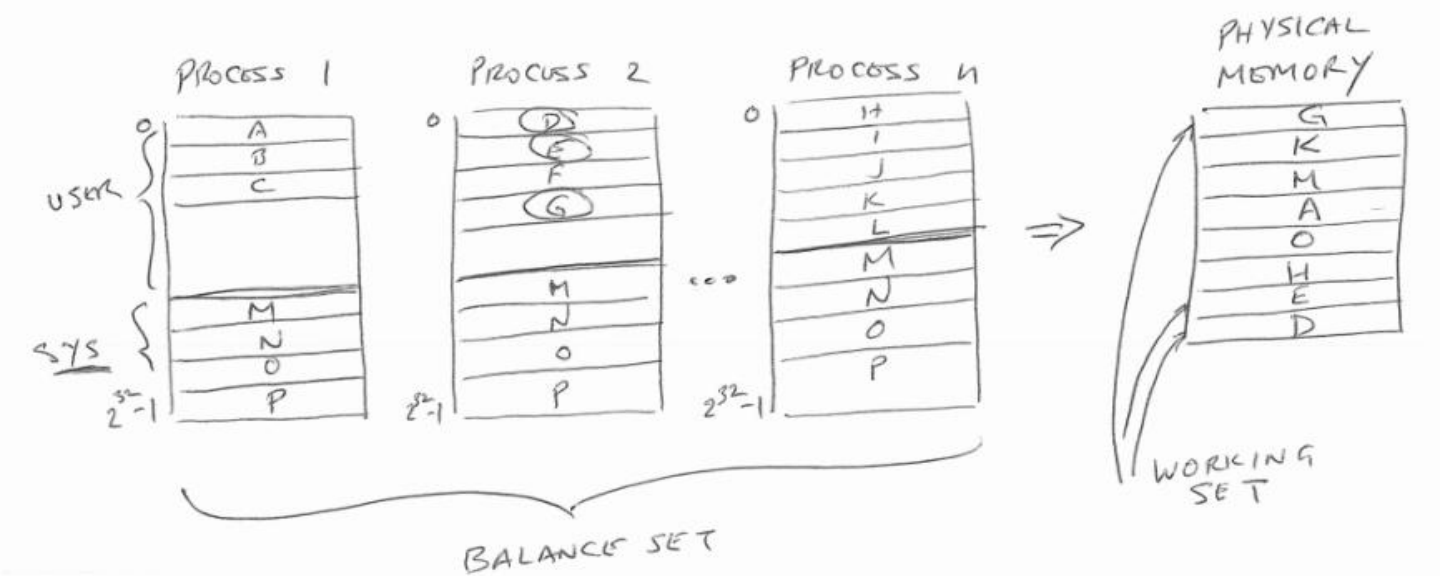


Three Concepts

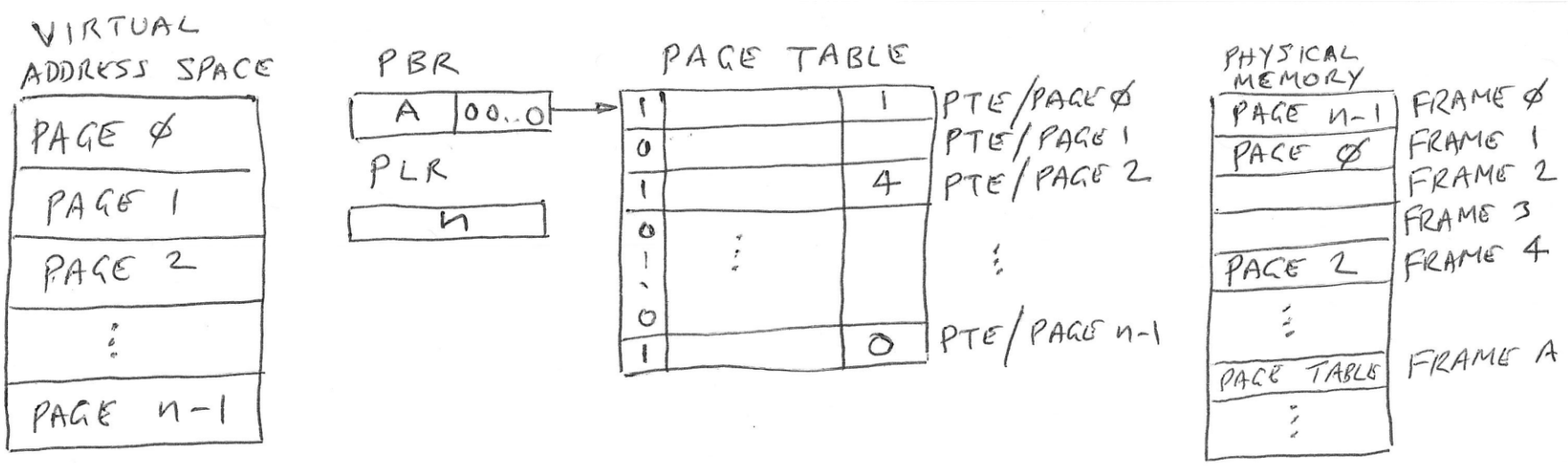
- ***The Process***
  - ***Granularity of process space is a page. Process consists of n pages.***
  - ***All are on the disk, some (the working set) are in physical memory***
    - ***A page of virtual memory occupies a frame of physical memory***
- ***The Process Context***
  - ***State information specific to a process (Intel: Task State Segment)***
    - ***Includes GPRs, Stack pointers, PC, PSL, Memory Management Registers***
  - ***Loaded when turning control of the computer over to a process***
  - ***Saved when removing a process' control of the computer***
  - ***Includes registers specific to the memory management system***
- ***The Page Table***
  - ***Consists of n page table entries (PTE)***
  - ***Each page has a PTE (The PTE is the descriptor for that page)***
    - ***The PTE is used for translation (What frame is the page occupying)***
    - ***The PTE is used for access control (Does the process have the right to make the desired access on this page)***

Layout of Virtual and Physical Memory

- ***N processes share physical memory***
- ***Virtual memory partitioned into user space and privileged space***
- ***Virtual memory pages mapped to physical memory frames***
- ***Balance set is the set of all processes alive in the system***
- ***Working set is the no. of resident pages for a productive process***
  - ***Pages D, E, and G comprise the working set of Process 2***



Case 1: The Page Table is in Physical Memory



- ***The process consists of n pages, three are resident***
- ***The Process Base Register (PBR)***
  - ***Note that A is a frame number since the Page Table is in Physical Memory; i.e., PBR contains a physical address***
  - ***PBR points to the first address in that frame***
    - ***i.e., the address of the PTE of Page 0 of process space***
- ***The Process Length Register (PLR)***
  - ***PLR contains the number of virtual pages in the process***

The Translation Process (From VA of x to PA of x) Assume x is on virtual page k

- ***Step 1: Is PLR < k, the page number of VA of x?***
  - ***If yes, VA of x is ill formed, access is denied***
- ***Step 2: Get the PTE***
  - ***Since PTEs are 32 bits, address of PTE of page k is PBR + 4 x k***
- ***Step 3: Check protection, verify process' right to access***
  - ***If no, take an access control violation (ACV)***
- ***Step 4: Check the valid bit***
  - ***If 0, page is not resident, take a page fault***
- ***Step 5: Perform translation***

Case 2: The Page Table is in System Virtual Memory

- ***Why do we use this more complicated structure?***
  - ***No need for the whole page table in physical memory at same time***
  - ***2GB of process space yields a 16MB page table for the VAX***
    - ***A 16 MB page table in system virtual space requires 128KB of physical memory if the entire page table is resident, which is almost never necessary.***
- ***The Process Base Register (PBR)***
  - ***Note the page number, indicating PBR contains a virtual address***
  - ***PBR points to the first address on that page***
    - ***Address of the PTE of Page 0 of process space***
  - ***The high bit of PBR is “1” indicating the Page Table is in Sys. Space***
- ***The Process Length Register (PLR)***
  - ***Again, PLR contains the number of pages comprising the process***
- ***The System Page Table is in physical memory***

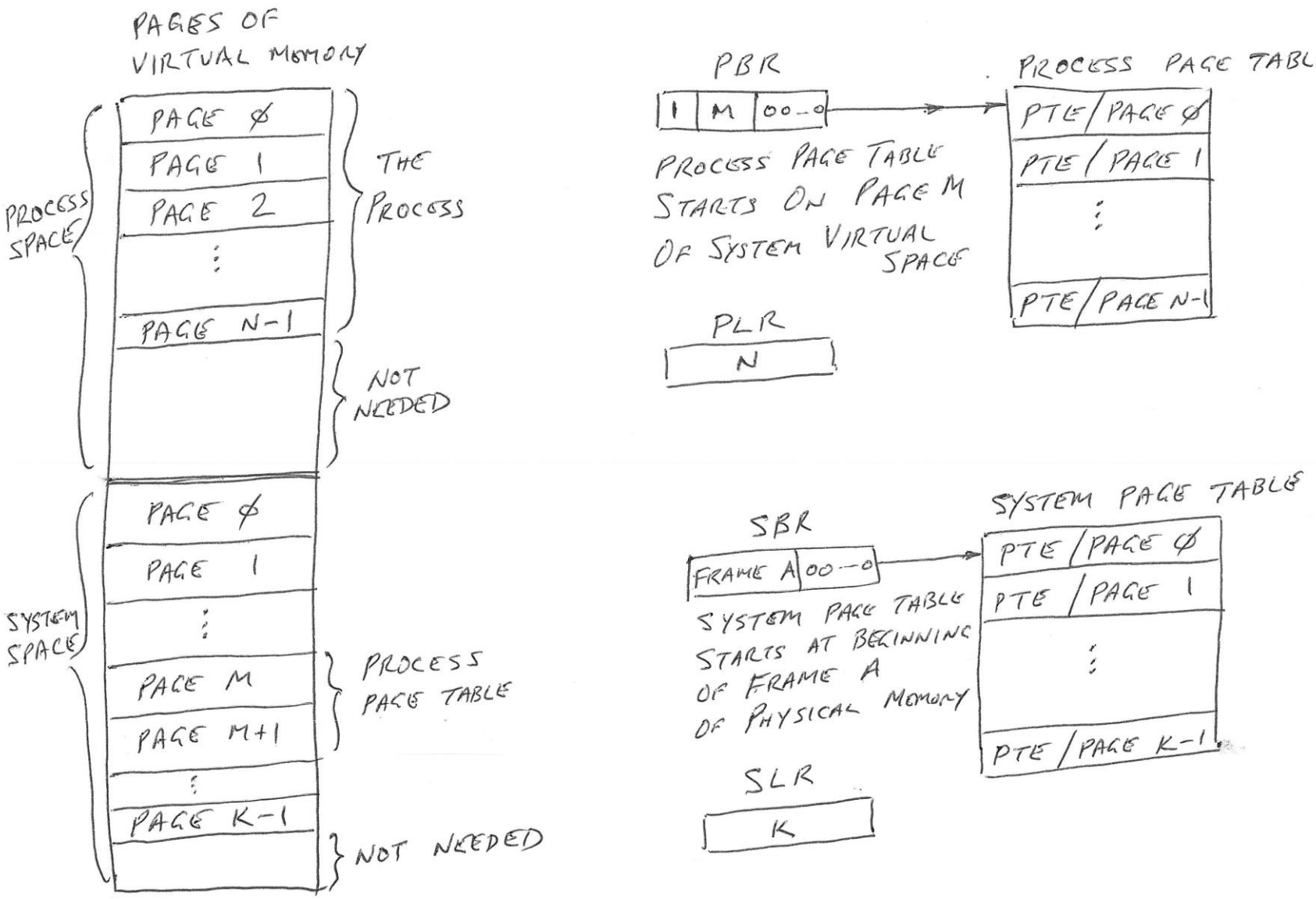
Cost of Page Table in Physical Memory vs Cost of Page Table in System Virtual Memory

- ***Suppose we have available 2^48 bytes of virtual memory***

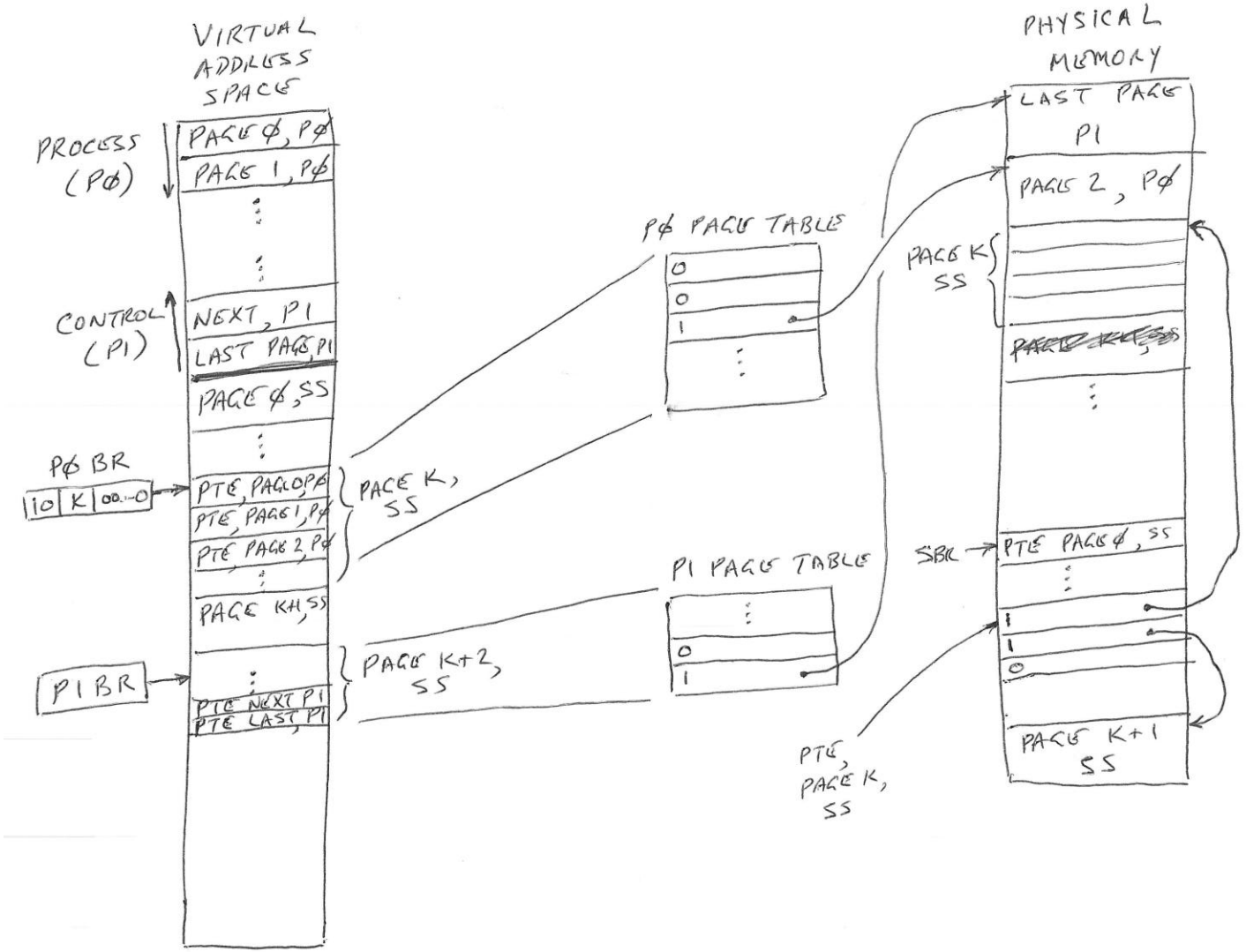
- ***In physical memory:***
  - ***2^48/2^12 = 2^36 pages → 2^38 bytes for the page table***
  - ***= 256GB of physical memory***

- ***In system virtual memory:***
  - ***2^48/2^12 = 2^36 pages → 2^38 bytes for the page table is System Virtual memory***
  - ***2^38/2^12 = 2^26 pages of System Virtual Memory – 2^28 bytes of Sys Page Table***
  - ***= 256MB of physical memory***

Pages and Page Tables for Case 2 (The Process Page Table is in System Virtual Space)



An example of Case 2: VAX Virtual Memory System (Note:We are ignoring Control (P1) Space in EE460N)



Explanation of the VAX Virtual Memory Layout

- ***The 4GB virtual address space is in 4 regions***
  - ***Bits [31:30] = 00 → P0 Space (user space)***
  - ***Bits [31:30] = 01 → P1 Space (user space)***
  - ***Bits [31:30] =10 → System Space***
  - ***Bits [31:30] = 11 → Reserved for future use***
- ***P0 Page Table is on Page k of System Virtual Space***
  - ***Page 2 of P0 Space is resident in frame 1 of Physical Memory***
    - ***Note: valid bit of PTE of Page 2 is 1***
    - ***Page 0 and 1 of P0 Space are not resident. The valid bits of their PTEs are 0***
- ***P1 Page Table is on Page k+2 of System Space, not resident***
  - ***Last page of P1 space is resident in frame 0 of Physical Memory***
- ***System Page Table is resident in Physical Memory***
  - ***Pages k and k+1 of System Space are resident in Physical Memory***

The flow: LD R1,X

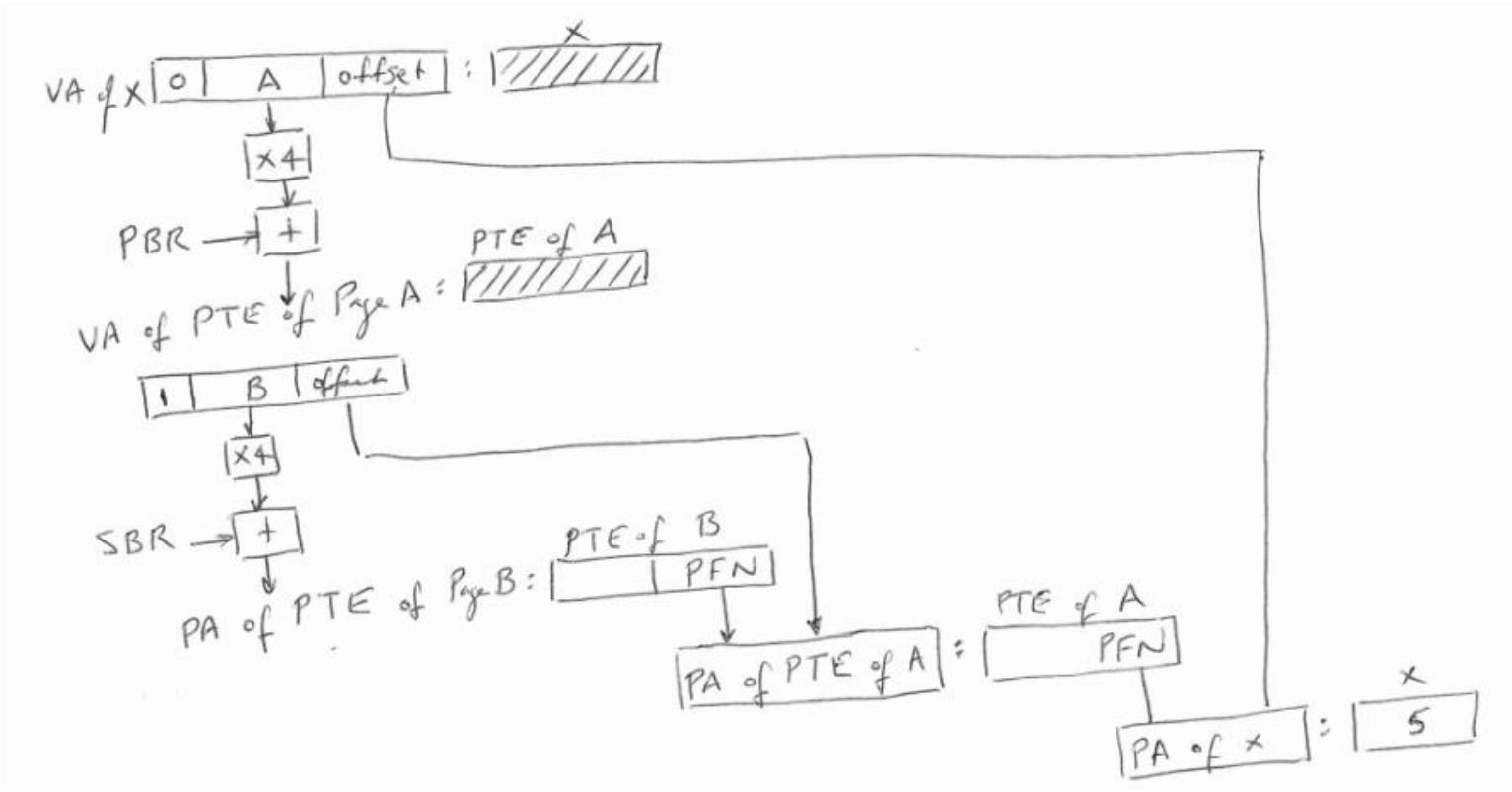
- ***How does the uarch find the physical address (PA) of X***
  - ***Let us say X is on virtual page number A.***
  - ***To get the PA of X, the uarch needs the PTE of page A (VA of X contains: region bits, page no., byte on the page)***
  - ***We compute the virtual address (VA) of the PTE of page A, i.e., PBR + 4 times A. [The Process Page Table is in Sys Space]***
  - ***Let us say the PTE of page A is on page B of system space***
  - ***We can get the PTE of page B of System Space from the System Page Table, which is in Physical Memory [SBR + 4 x B]***
  - ***The PTE of Page B gives us the frame containing the PTE of Page A***
  - ***The PTE of Page A gives us the frame containing X.***
  - ***Since we now have the PA of X, we can access X.***

The Abstraction

- ***Start with VA, end with PA (6 steps)***
  - ***1.Which page table***
  - ***2. Is Page No. < Proc length register? If no, ACV fault***
  - ***3. Get the PTE (using the page tables)***
  - ***4. Check protection field. If no, ACV fault***
  - ***5. Check V bit, is page resident. If no, TNV fault (Page Fault)***
  - ***6. All cool, access the physical address***

The Step-by-step Translation Process ("Walking" the Page Table)

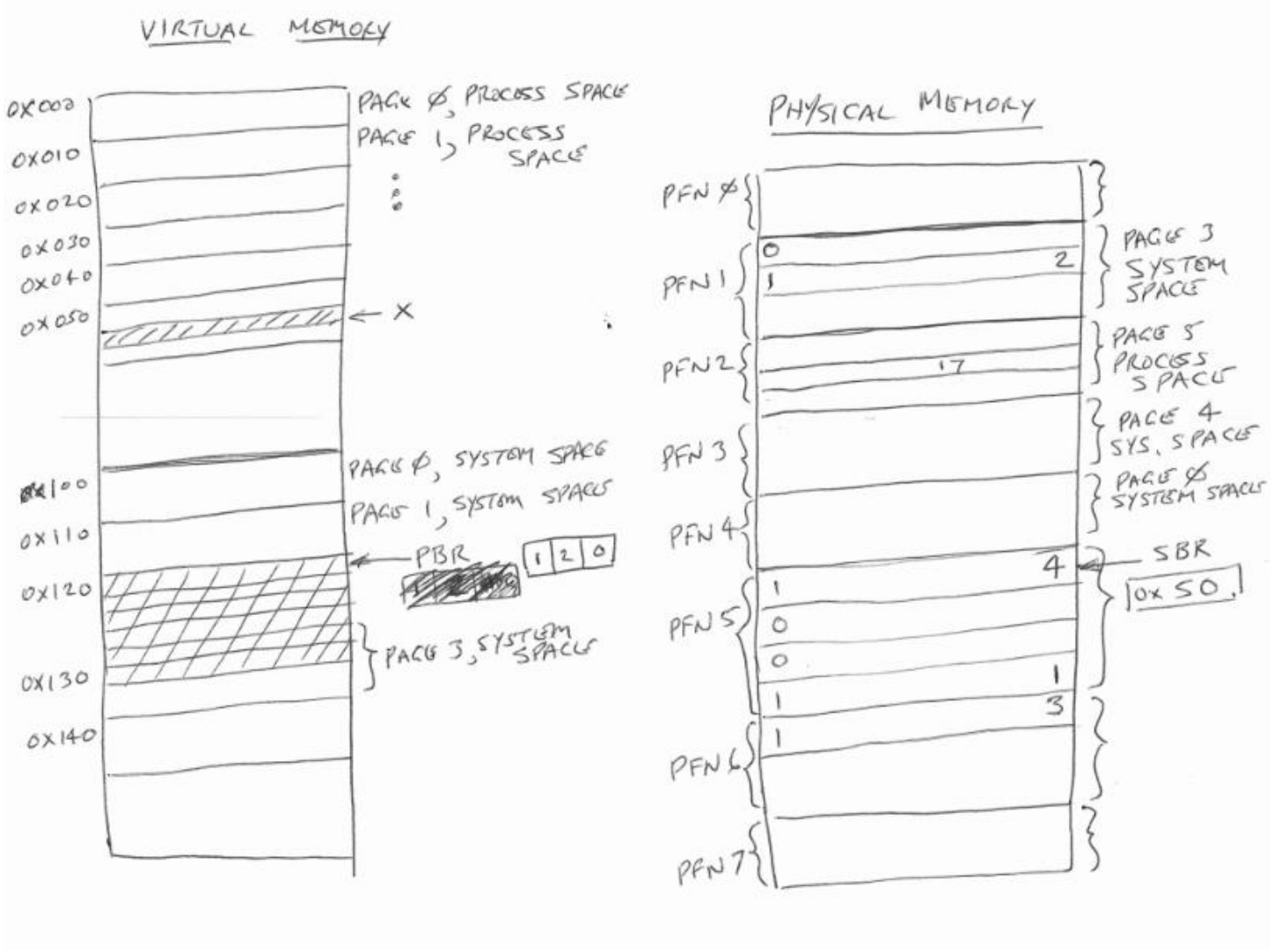
- ***x is in virtual memory on Page A***
- ***The PTE of Page A is in Page B of System Virtual Space***



A Complete Example!

- ***We will modify the ISA to make it easier to digest:***
  - ***Page size will be 16 bytes (instead of 512 bytes)***
  - ***VA will be 9 bits (instead of 32 bits), 32 pages of 16 bytes***
  - ***Physical Address will be 7 bits, 8 page frames of 16 bytes***
  - ***PTE will still take 4 bytes***
  - ***Process space consists of 16 pages, our example: 6 pages***
  - ***System space consists of 16 pages, our example: 5 pages***
- ***Our example:***
  - ***Process page table starts (PBR) at VA 0x120***
    - ***6 PTEs times 4 bytes/PTE = 24 bytes = 1.5 pages of system space***
  - ***System page table starts (SBR) at PA 0x50***
    - ***5 PTEs times 4 bytes/PTE = 20 bytes = 1.25 pages of physical memory***
  - ***System page table indicates pages 0,3, 4 are resident***
  - ***Process page table indicates page 5 resident, page 4 not so.***

A Memory Map

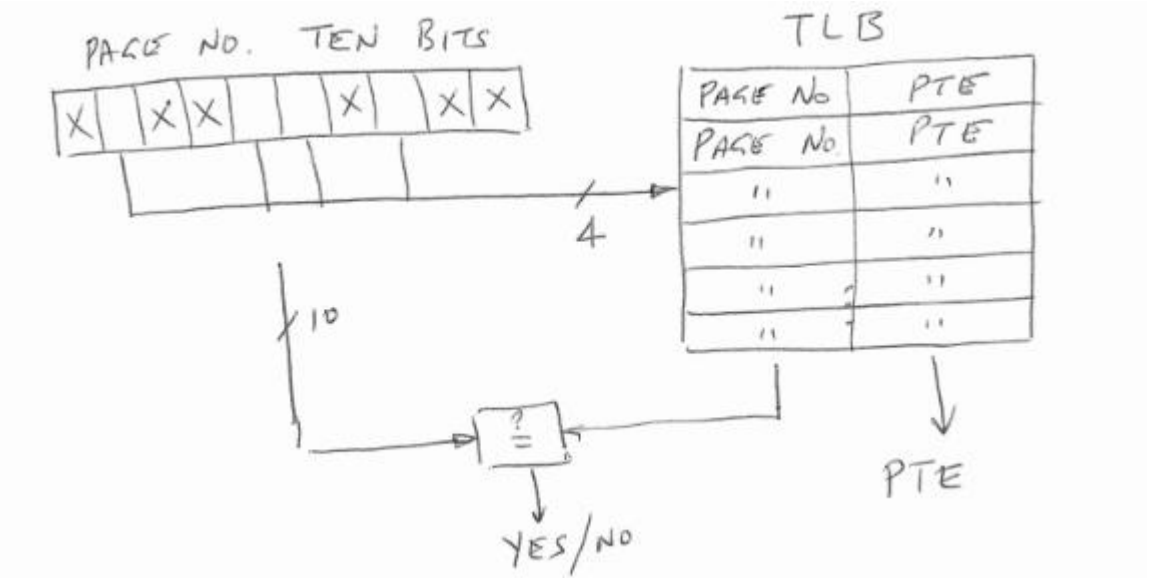

VIRTUAL MEMORY
0X000
0X010
0X020
0X030
0X040
0X050
0X100
0X110
0X120
0X130
0X140
PAGE Ø, PROCESS SPACE
PAGE 1, PROCESS SPACE
X
PAGE Ø, SYSTEM SPACE
PAGE 1, SYSTEM SPACE
PBR
1 2 0
PAGE 3, SYSTEM SPACE
PHYSICAL MEMORY
PFN Ø
PFN 1
PFN 2
PFN 3
PFN 4
PFN 5
PFN 6
PFN 7
0
1
2
PAGE 3 SYSTEM SPACE
17
PAGE 5 PROCESS SPACE
PAGE 4 SYS. SPACE
PAGE Ø SYSTEM SPACE
4
SBR
0x 50
1
0
0
1
1
3
1

Processing the instruction: LD R1, X

- ***x is a VA: 0 0101 1000***
  - ***i.e., byte 8 on page 5 of process space***
- ***We need the PTE of page 5 of process space.***
  - ***Page 5 x 4 bytes/PTE = 0x00010100. Add to PBR.***
  - ***Therefore, Page 5 PTE is at VA = 0x134***
  - ***Note the crosshatch on the figure. We can not read VM.***
- ***To get the physical location of page 5, access SPT***
  - ***SBR + 4x page 3 gives us physical address of PTE of page 3***
  - ***SBR = 0x50, i.e., 0x 1010000; 4 times 3 =12, i.e., 0001100***
  - ***i.e., PA of PTE of page 3 of system space is in 0x5c***
  - ***We read physical memory: PTE of page 3 indicates PFN is 1***
  - ***Since VA of this PTE was 0x134, we add 4 (offset) to PFN***
  - ***yielding 0x14, the address of the PTE of the Page containing x***
  - ***The PTE directs us to PFN 2, yielding 0x28 as PA of x***
  - ***The contents of 0x28 is 17 which we load into R1, and done!***

TLB Structure

- *Example of a Translation Lookaside Buffer (TLB)*
  - *TLB has 16 entries*
  - *Assume 2^10 pages of Virtual Memory*
    - *Page number consists of ten bits*
    - *Index to TLB is a 4-bit hash function (bits 8,5,4,2)*
  - *TLB is a Content addressable memory*
    - *We compare ten bit page number in the entry with page number*
    - *If a match, we can immediately output the PTE (no extra cycles)*

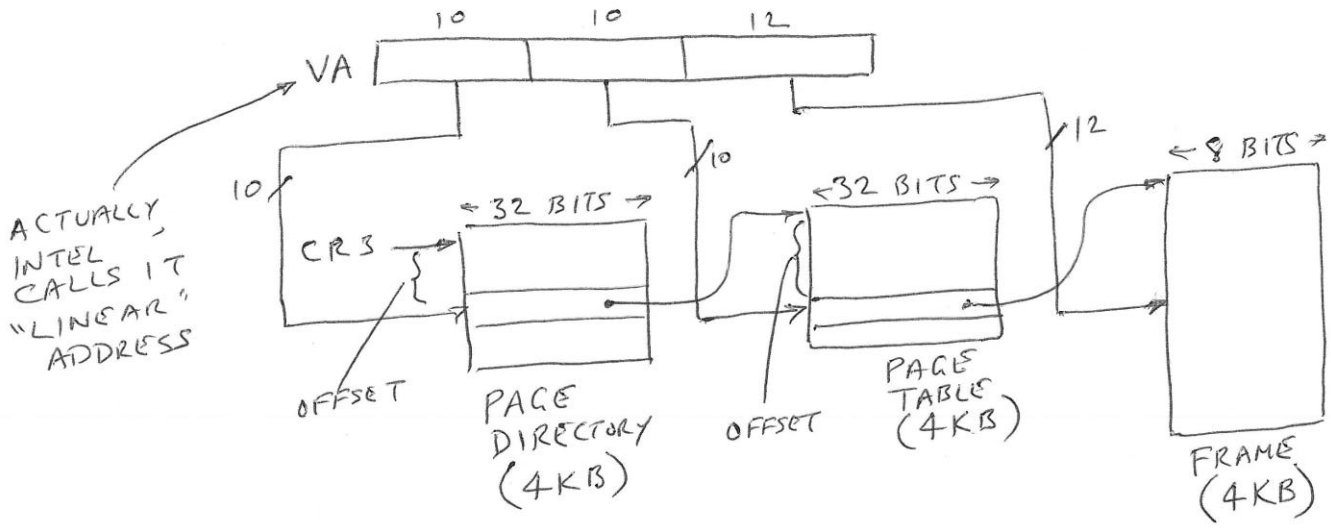


Cost of translation with a TLB (VAX-11/780)

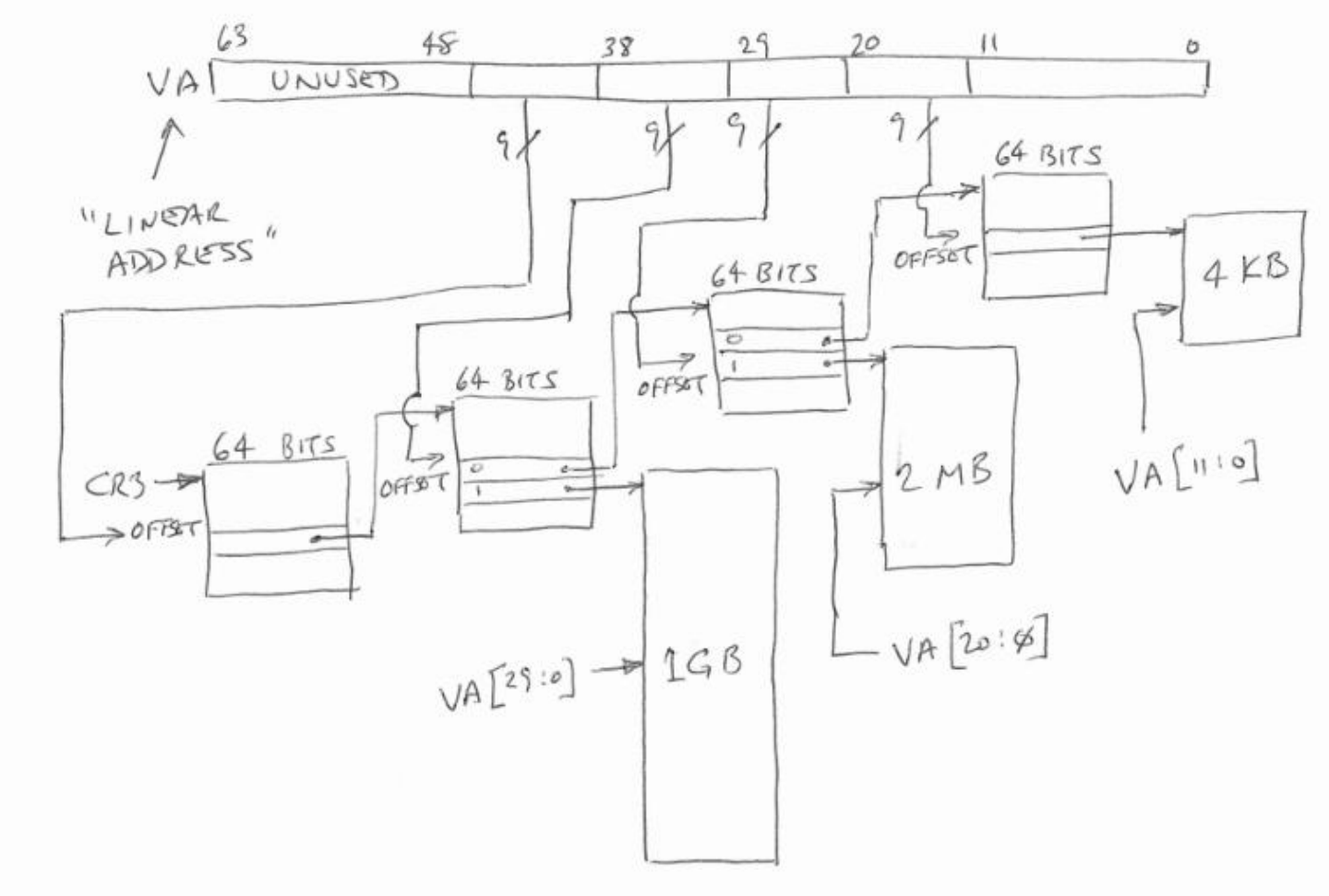
- ***Page Walk: 22 cycles***
- ***TLB hit: 0 cycles***
- ***TLB hit ratio: 95%***
- ***Therefore: 0.05 x 22 cycles + 0.95 x 0 cycles = 1 cycle***

The 4KB Page Size

- *Original size for x86 ISA*
- *Still the only page size for almost all ISAs today*
  - *Including RISCV*
  - *Not x86 – three page sizes: 4KB, 2MB, 1GB*
  - *Not Arm – three page sizes: 4KB, 64KB, 1MB*
  - *Not Faruk – 4KB, 8KB, 16KB, 32KB, … 1GB*
- *The original layout for x86 pages*



x86 Extension to 3 Page Sizes



Gracias!