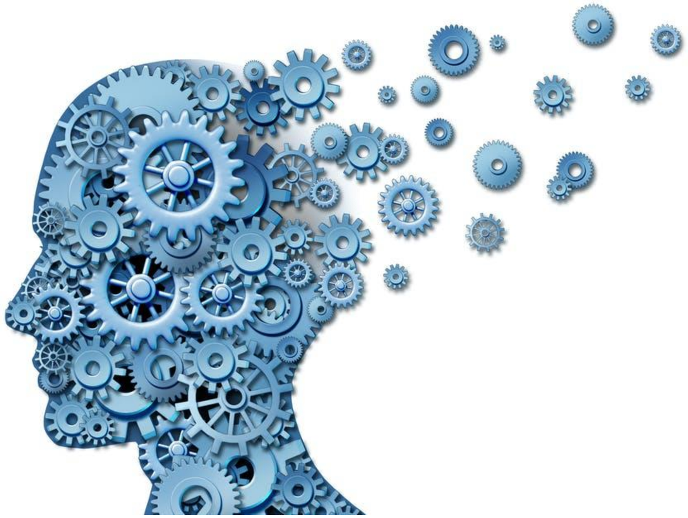
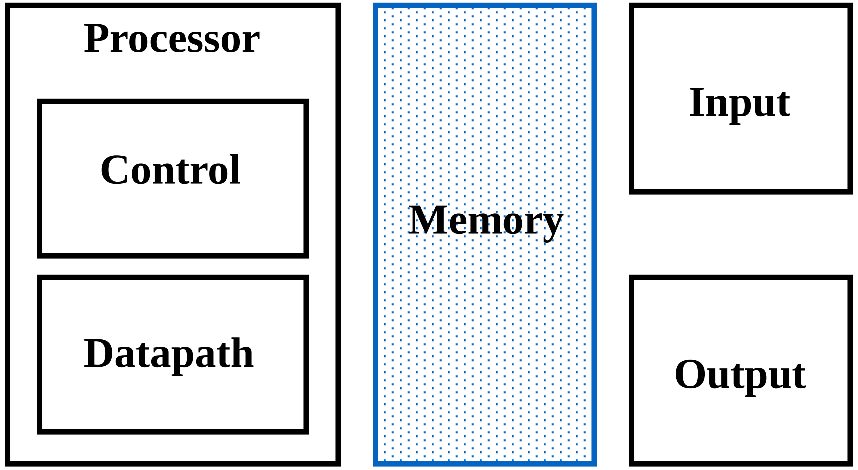
MODULE 3: VIRTUAL MEMORY



Nilanjan Byabarta
University of Engineering and Management, Kolkata

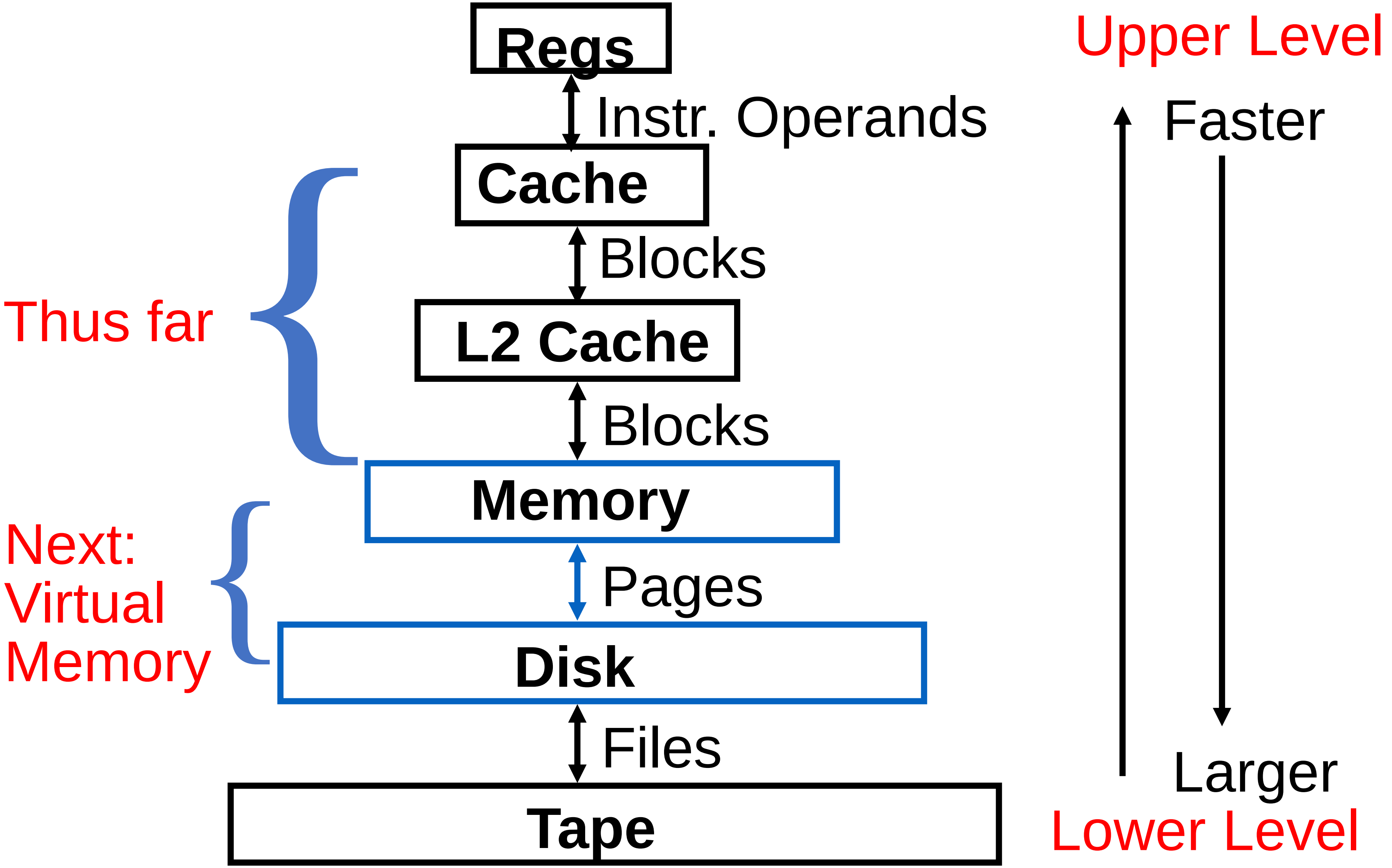
The Big Picture: Where are We Now?

- The Five Classic Components of a Computer



- Today's Topics:
  - Virtual Memory
  - TLB

Another View of the Memory Hierarchy



Memory Hierarchy Requirements

- If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory,
  then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?

- While we're at it, what other things do we need from our memory system?

Memory Hierarchy Requirements

- Share memory between multiple processes but still provide protection – don't let one program read/write memory from another
- Address space – give each program the illusion that it has its own private memory
  - Suppose code starts at address 0x00400000. But different processes have different code, both residing at the same address. So each program has a different view of memory.

Virtual Memory

- Called "Virtual Memory"
- Also allows OS to share memory, protect programs from each other
- Today, more important for protection vs. just another level of memory hierarchy
- Historically, it predates caches

What is virtual memory?

- Virtual memory => treat the main memory as a cache for the disk
- Motivations:
  - Allow efficient and safe sharing of memory among multiple programs
    - Compiler assigns virtual address space to each program
    - Virtual memory maps virtual address spaces to physical spaces such that no two programs have overlapping physical address space
  - Remove the programming burdens of a small, limited amount of main memory
    - Allow the size of a user program exceed the size of primary memory
- Virtual memory automatically manages the two levels of memory hierarchy represented by the main memory and the secondary storage

Issues in Virtual Memory System Design

What is the size of information blocks that are transferred from secondary to main storage (M)? ⇒ ***page size***

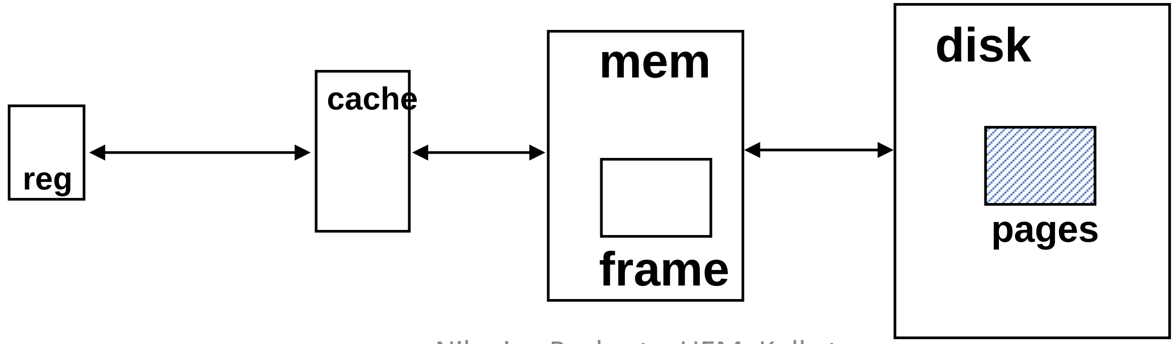
Which region of M is to hold the new page ⇒ ***placement policy***

How do we find a page when we look for it? ⇒ ***page identification***

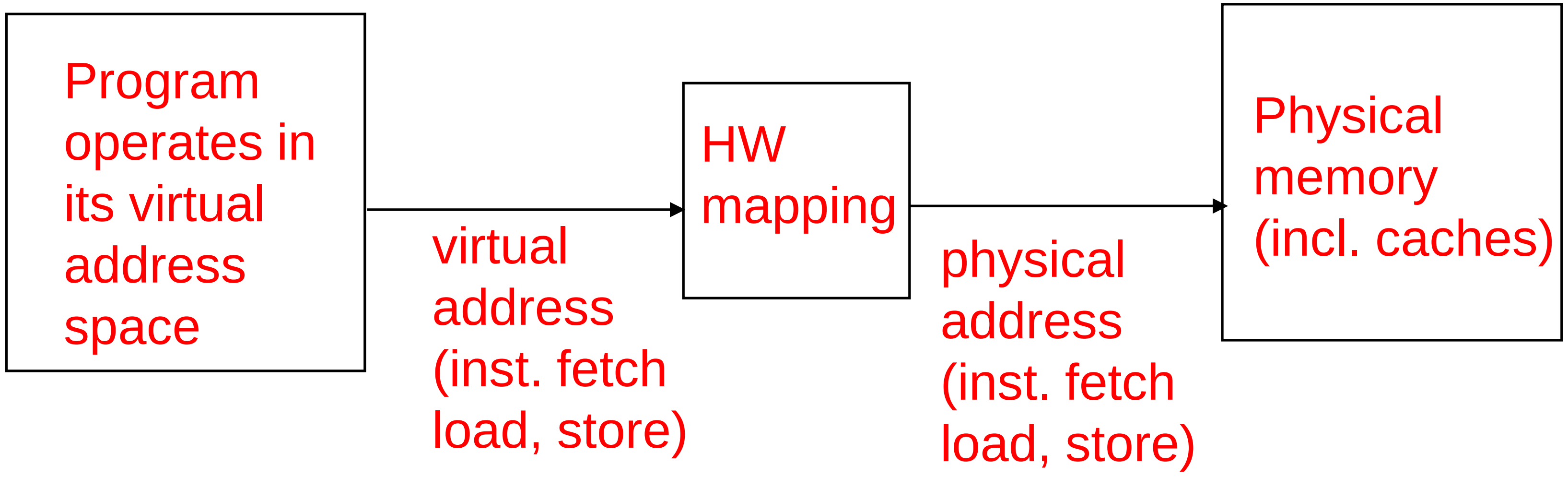
Block of information brought into M, and M is full, then some region of M must be released to make room for the new block ⇒ ***replacement policy***

What do we do on a write? ⇒ ***write policy***

Missing item fetched from secondary memory only on the occurrence of a fault ⇒ ***demand load policy***



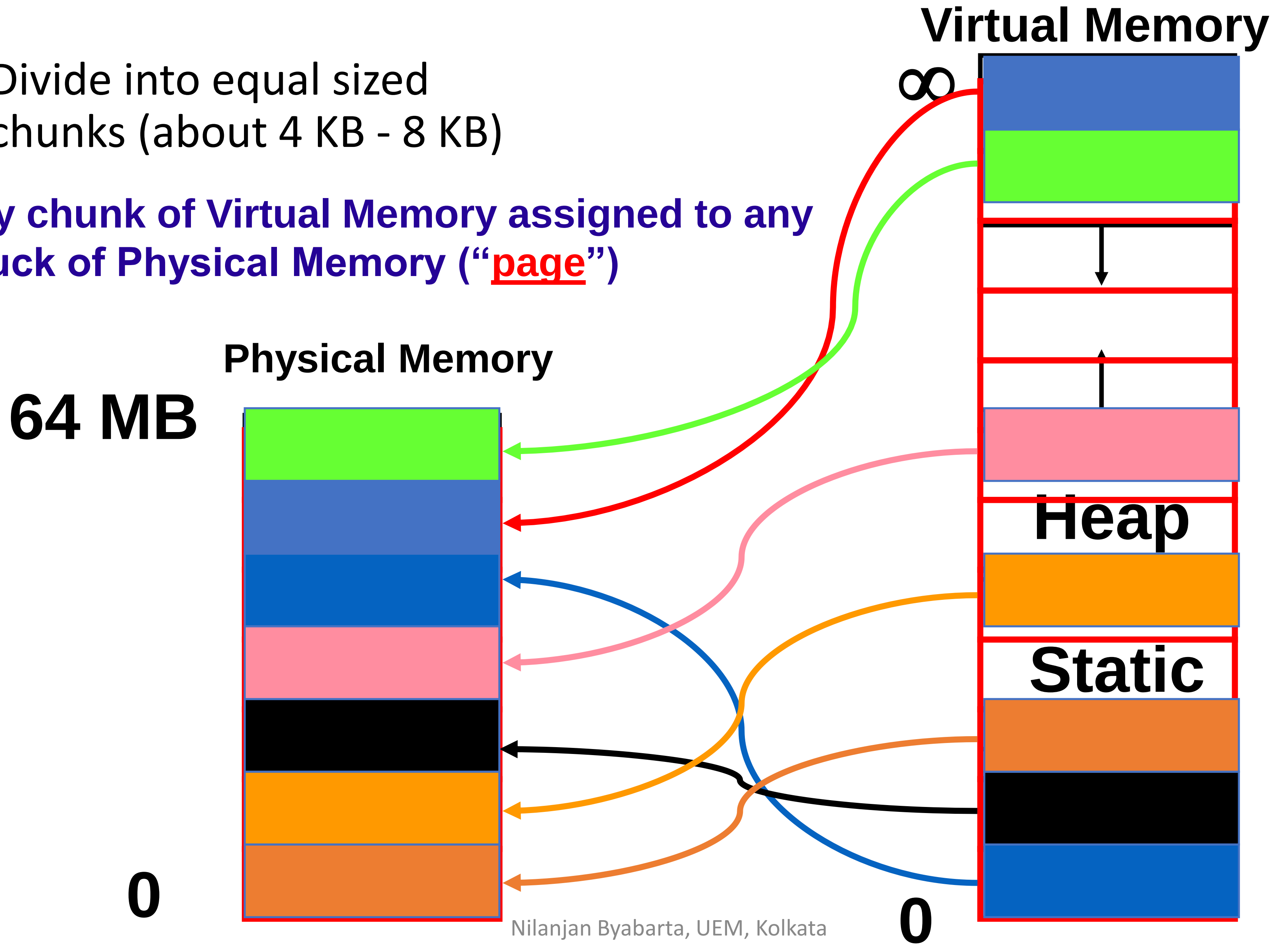
Virtual to Physical Addr. Translation

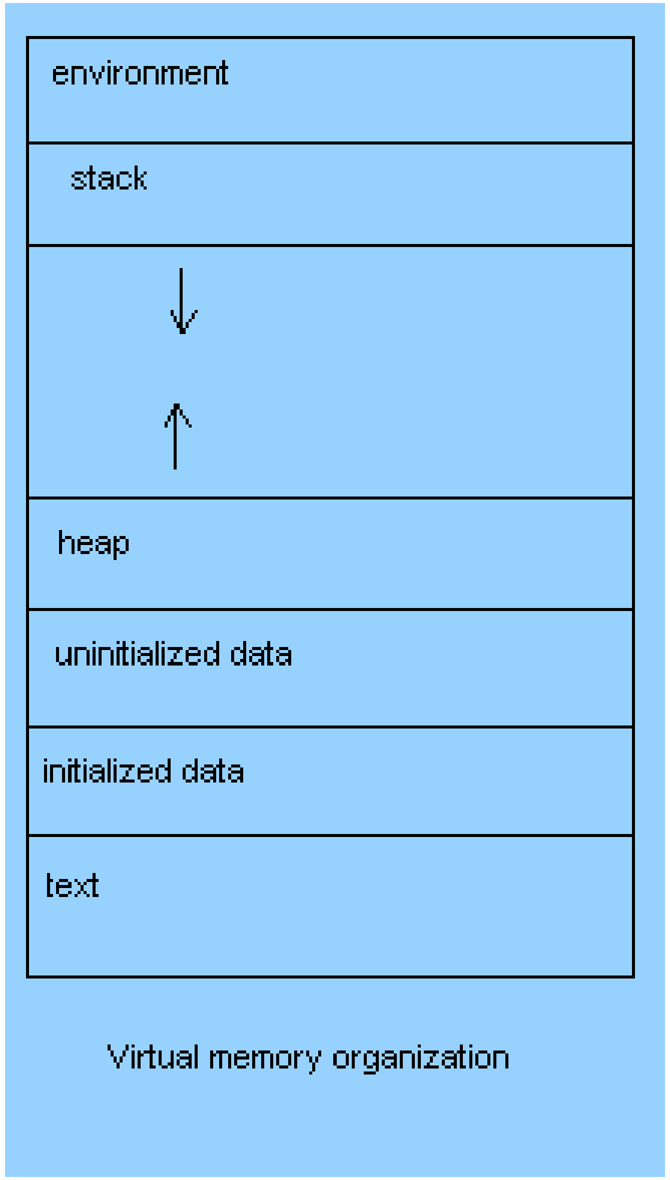


- Each program operates in its own virtual address space; ~only program running
- Each is protected from the other
- OS can decide where each goes in memory
- Hardware (HW) provides virtual $\Rightarrow$ physical mapping

Mapping Virtual Memory to Physical Memory

- Divide into equal sized chunks (about 4 KB - 8 KB)
- **Any chunk of Virtual Memory assigned to any chuck of Physical Memory ("page")**





Mapping Virtual Memory to Physical Memory

Each process's address space is typically organized in 6 sections that are illustrated in the picture:

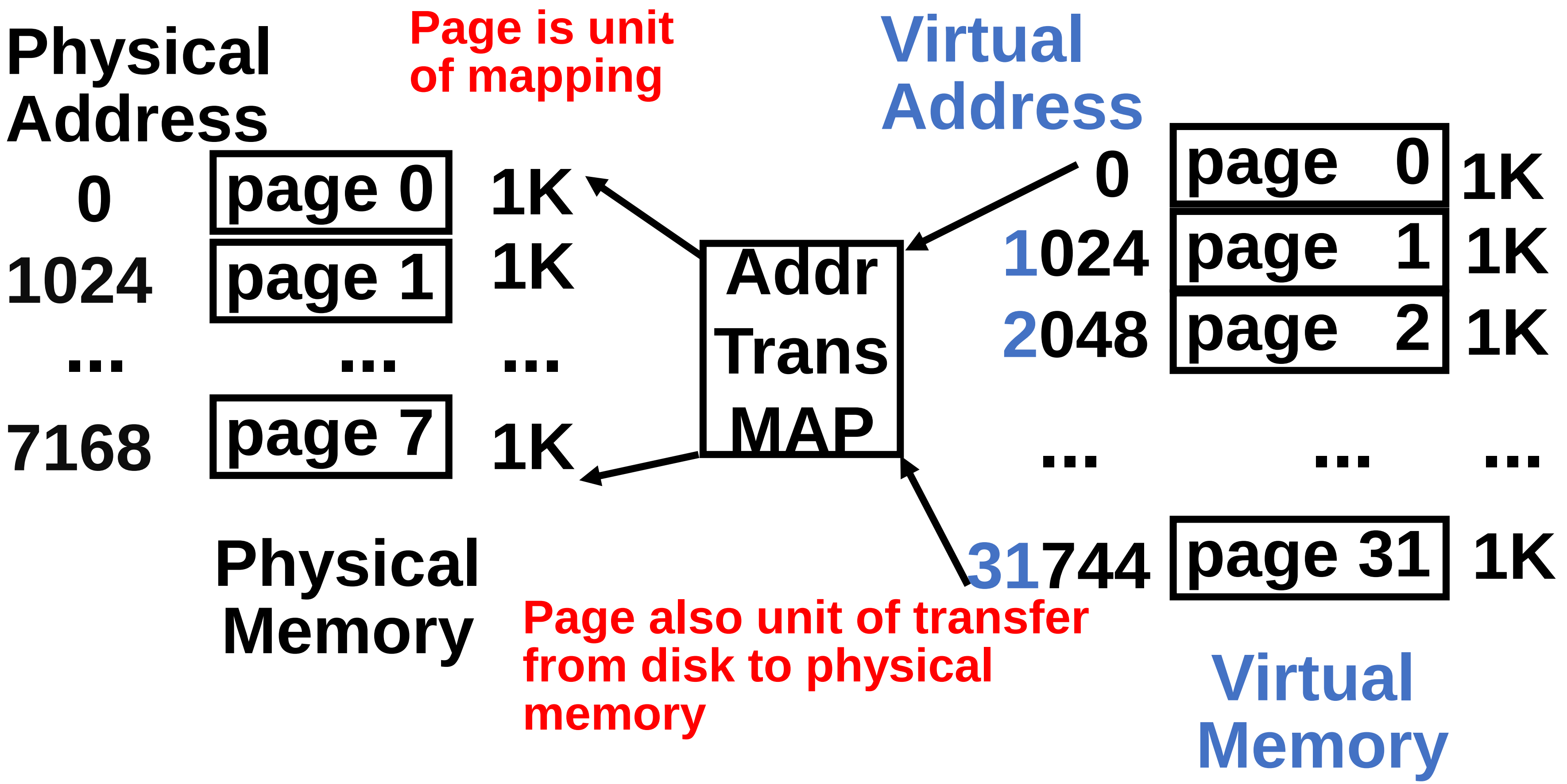
environment section – used to store environment variables and command line arguments;

the stack - used to store memory for function arguments, return values, and automatic variables;
the heap (free store) used for dynamic allocation,

two data sections (for initialized and uninitialized static and global variables)

text section where the actual code is kept

Paging Organization (assume 1 KB pages)



Virtual Memory Mapping Function

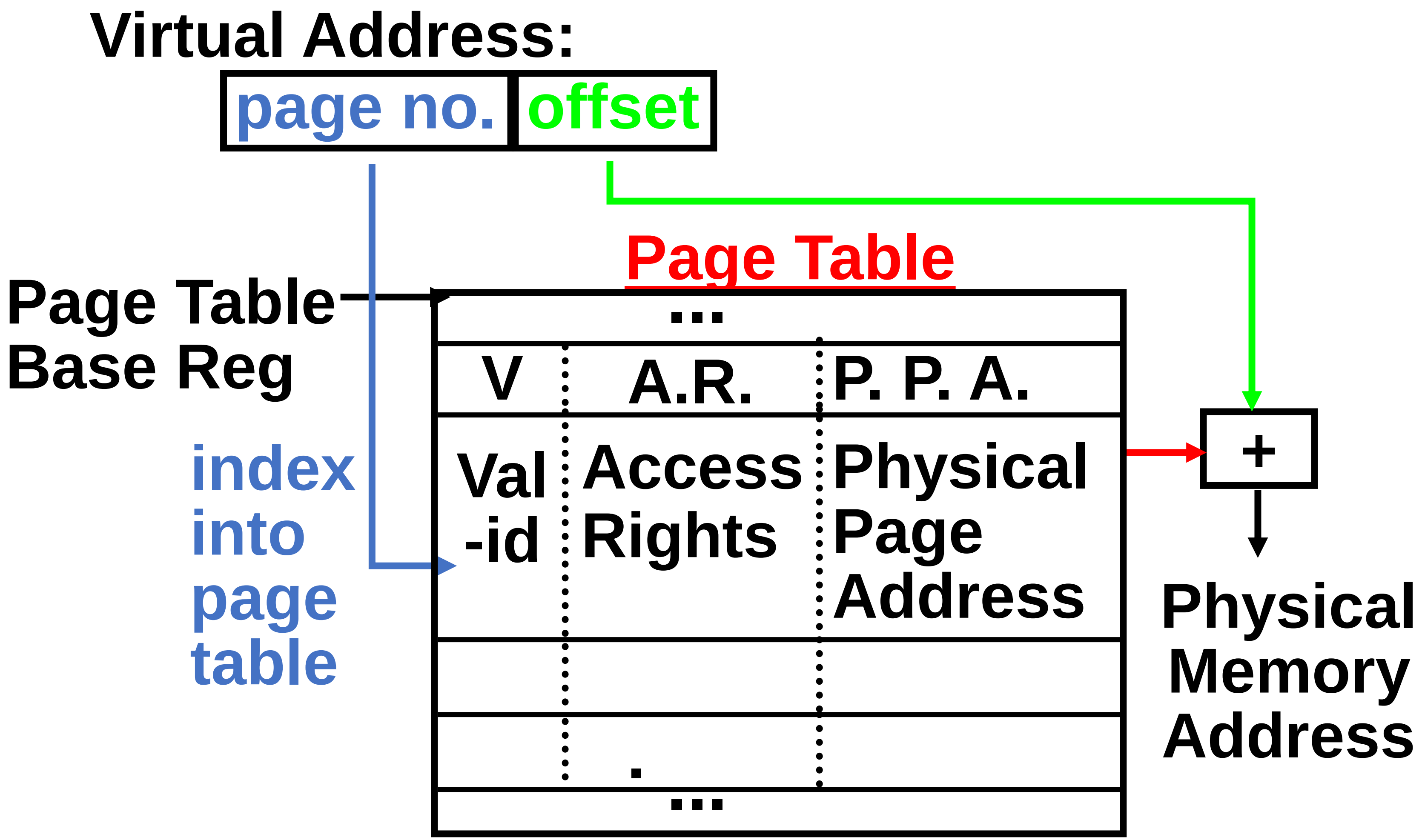
- Cannot have simple function to predict arbitrary mapping
- Use table lookup of mappings

Page Number	**Offset**

- **Use table lookup ("Page Table") for mappings: Page number is index**
- **Virtual Memory Mapping Function**
  - Physical Offset = Virtual Offset
  - Physical Page Number = Page Table[Virtual Page Number]

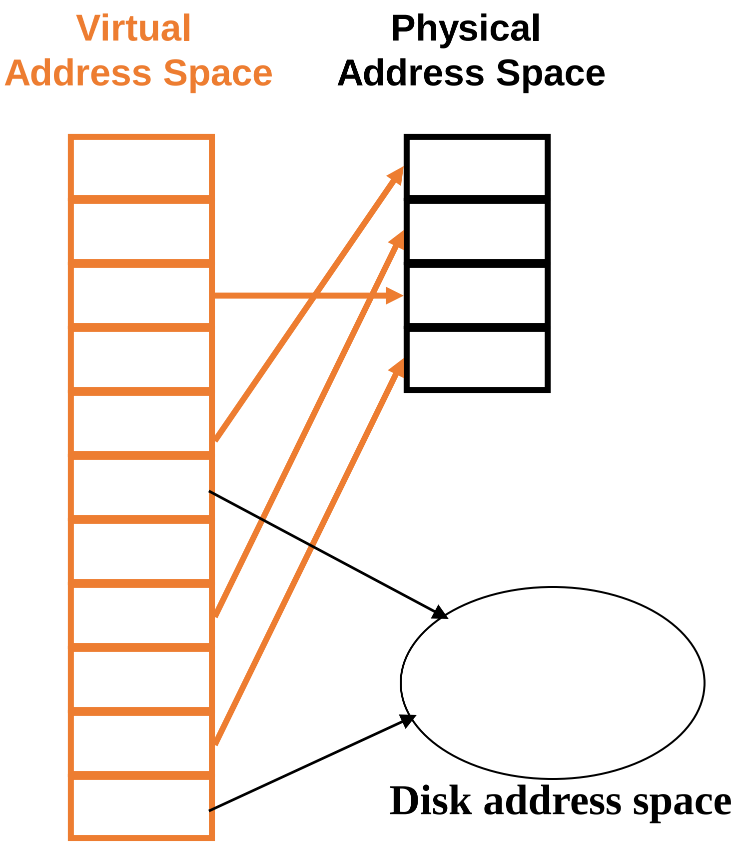
  (P.P.N. also called "Page Frame")

Address Mapping: Page Table



Page Table located in physical memory

Page Table



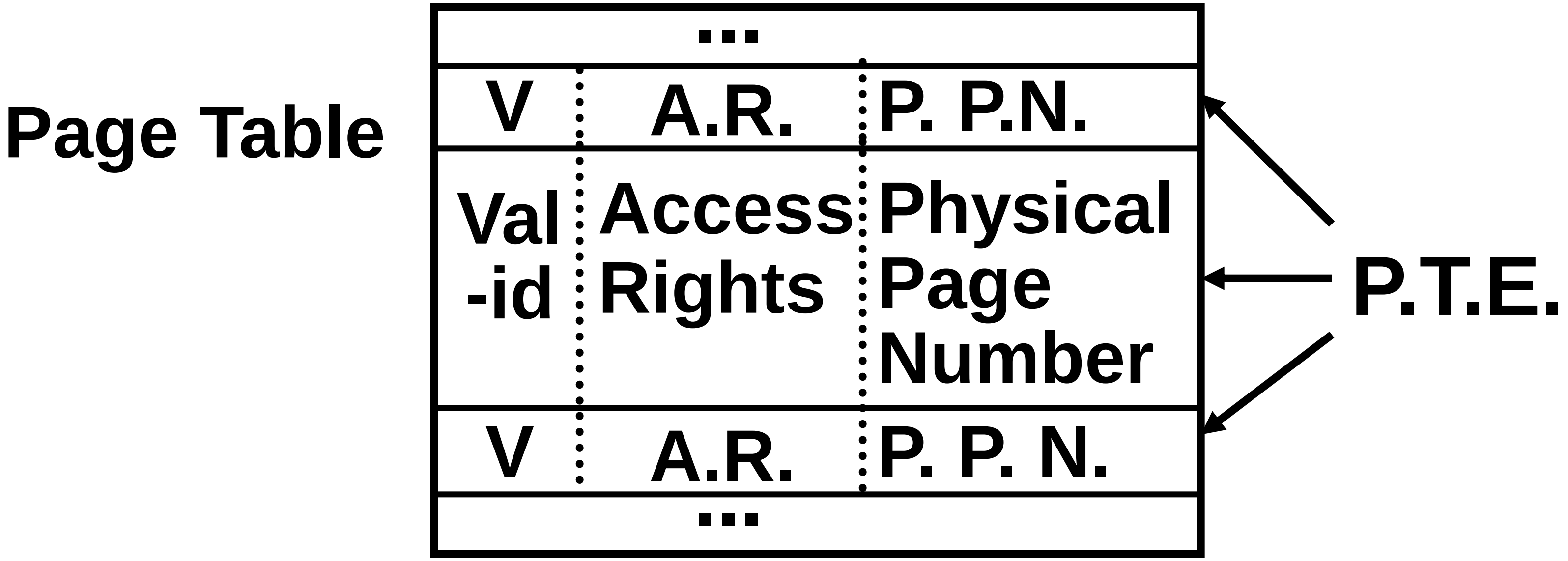
- A page table is an operating system structure which contains the mapping of virtual addresses to physical locations
  - There are several different ways, all up to the operating system, to keep this data around
- Each process running in the operating system has its own page table
  - "State" of process is PC, all registers, plus page table
  - OS changes page tables by changing contents of Page Table Base Register

Requirements revisited

- Remember the motivation for VM:
- Sharing memory with protection
  - Different physical pages can be allocated to different processes (sharing)
  - A process can only touch pages in its own page table (protection)
- Separate address spaces
  - Since programs work only with virtual addresses, different programs can have different data/code at the same address!

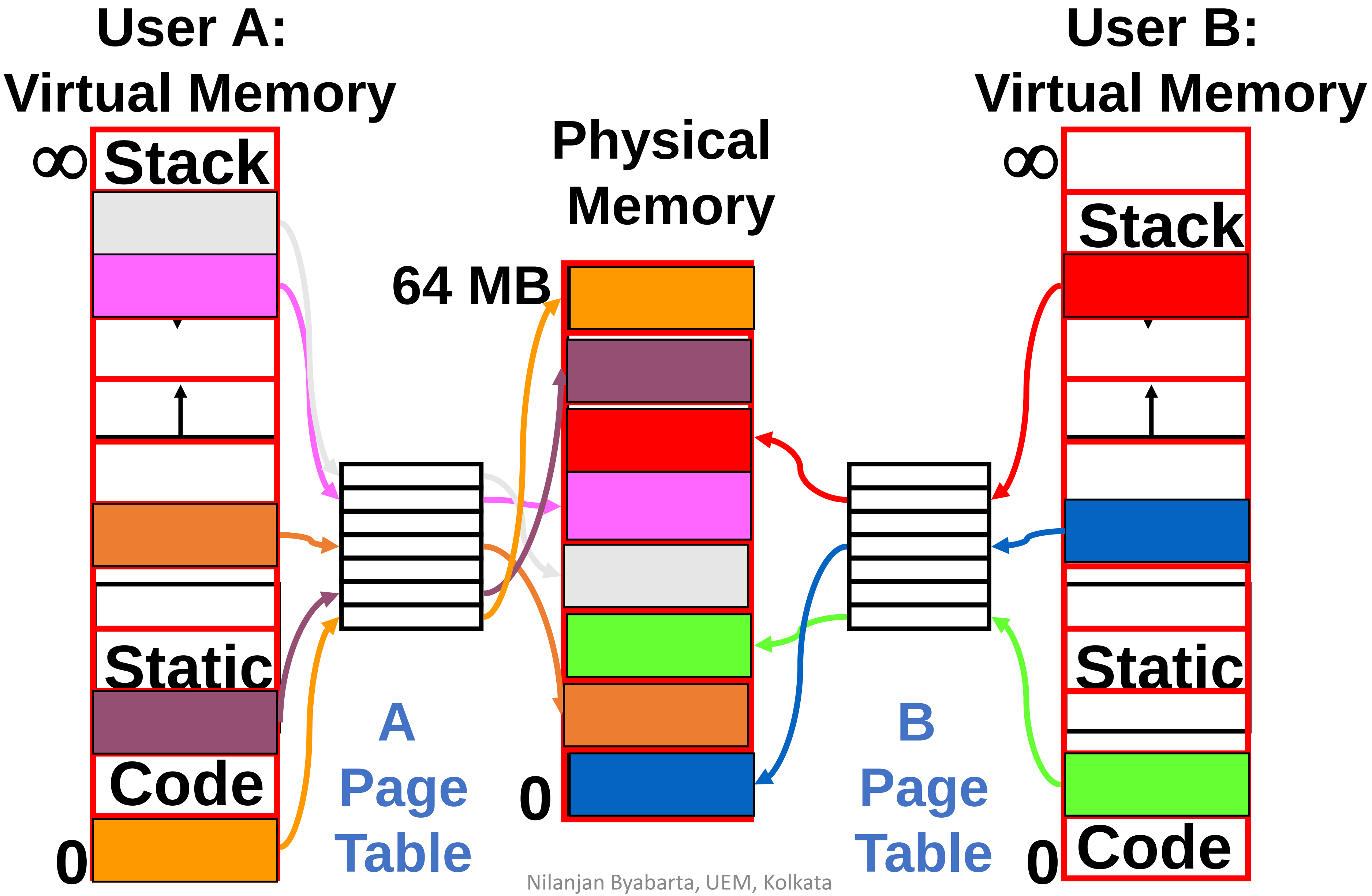
Page Table Entry (PTE) Format

- Contains either Physical Page Number or indication not in Main Memory
- OS maps to disk if Not Valid (V = 0)



- **If valid, also check if have permission to use page: <u>Access Rights</u> (A.R.) may be Read Only, Read/Write, Executable**

Paging/Virtual Memory Multiple Processes



Comparing the 2 levels of hierarchy

Cache Version	Virtual Memory version.
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-8KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

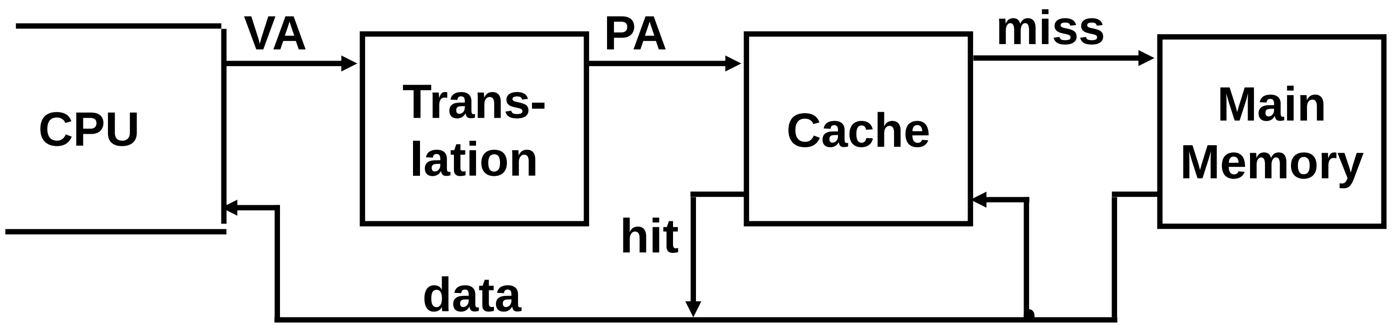
Notes on Page Table

- Solves Fragmentation problem: all chunks same size, and aligned, so all holes can be used
- OS must reserve "Swap Space" on disk for each process
- To grow a process, ask Operating System
  - If unused pages, OS uses them first
  - If not, OS swaps some old pages to disk
  - (Least Recently Used to pick pages to swap)
- Each process has own Page Table
- Will add details, but Page Table is essence of Virtual Memory

Page Table Summary

- Map virtual page number to physical page number
  - Full table indexed by virtual page number
- Minimize page fault
  - Fully associative placement of pages in main memory
- Reside in main memory
- Page fault can be handled in software, why?
- Page size is larger than cache block size, why?
- Each process has its own page table
- Page table base register:
  - The page table starting address of the active process will be loaded to the page table register

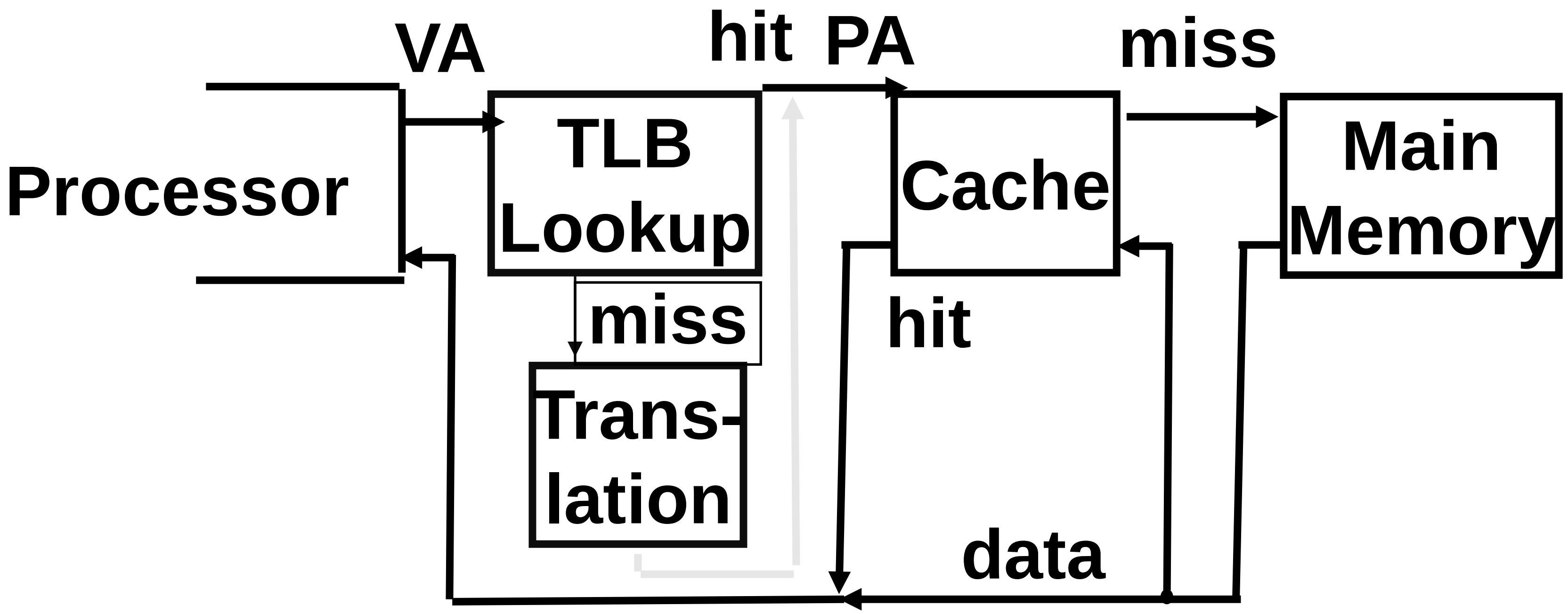
Virtual Memory Problem



- Virtual memory seems to be really slow:
  - we have to access memory on every access – even cache hits!
  - Worse, if translation not completely in memory, may need to go to disk before hitting in cache!
- Solution: Caching! (surprise!) – cache the translation
  - Keep track of most common translations and place them in a "Translation Lookaside Buffer" (TLB)

Translation Look-Aside Buffers (TLBs)

- **TLBs usually small, typically 128 - 256 entries**
- **Like any other cache, the TLB can be direct mapped, set associative, or fully associative**



On TLB miss, get page table entry from main memory

Typical TLB Format

Virtual Address	Physical Address	Dirty	Ref	Valid	Access Rights

- **TLB just a cache on the page table mappings**

- **TLB access time comparable to cache**
  (much less than main memory access time)
- **<u>Dirty</u>: since use write back, need to know whether or not to write page to disk when replaced**
- **<u>Ref</u>: Used to help calculate LRU on replacement**
  - **Cleared by OS periodically, then checked to see if page was referenced**

What if not in TLB?

- Option 1: Hardware checks page table and loads new Page Table Entry into TLB
- Option 2: Hardware traps to OS, up to OS to decide what to do

    follows Option 2: Hardware knows nothing about page table

Page Fault: What happens when you miss?

- Page fault means that page is not resident in memory
- Hardware must detect situation
- Hardware cannot remedy the situation
- Therefore, hardware must trap to the operating system so that it can remedy the situation
  - pick a page to discard (possibly writing it to disk)
  - start loading the page in from disk
  - schedule some other process to run

  Later (when page has come back from disk):
  - update the page table
  - resume to program so HW will retry and succeed!

What if the data is on disk?

- We load the page off the disk into a free block of memory, using a DMA (Direct Memory Access – very fast!) transfer
  - Meantime we switch to some other process waiting to be run
- When the DMA is complete, we get an interrupt and update the process's page table
  - So when we switch back to the task, the desired data will be in memory

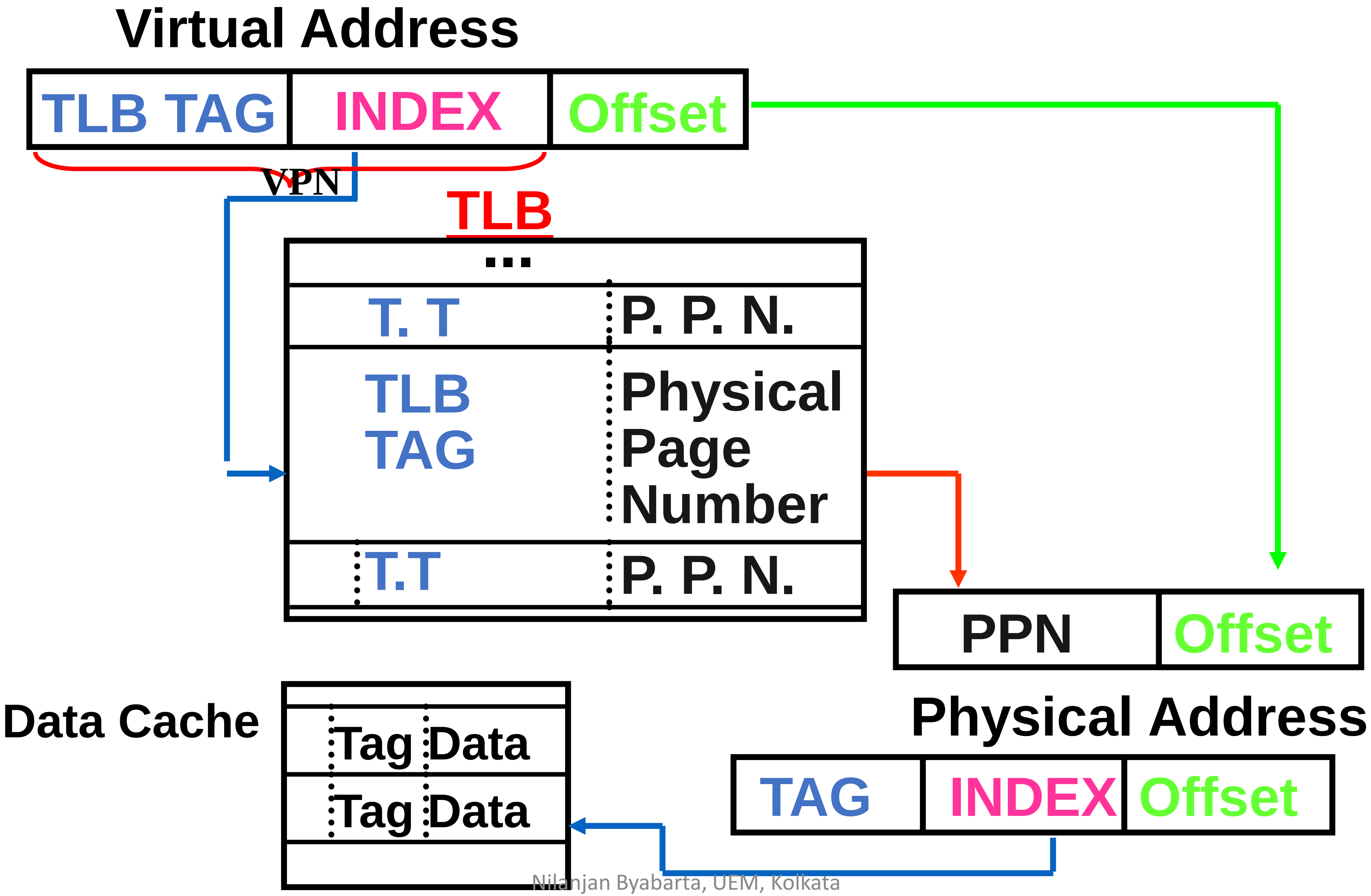
What if we don’t have enough memory?

- We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  - If clean (disk copy is up-to-date), just overwrite that data in memory
  - We chose the page to evict based on replacement policy (e.g., LRU)
- And update that program's page table to reflect the fact that its memory moved somewhere else
- If continuously swap between disk and memory, called Thrashing

And in conclusion…

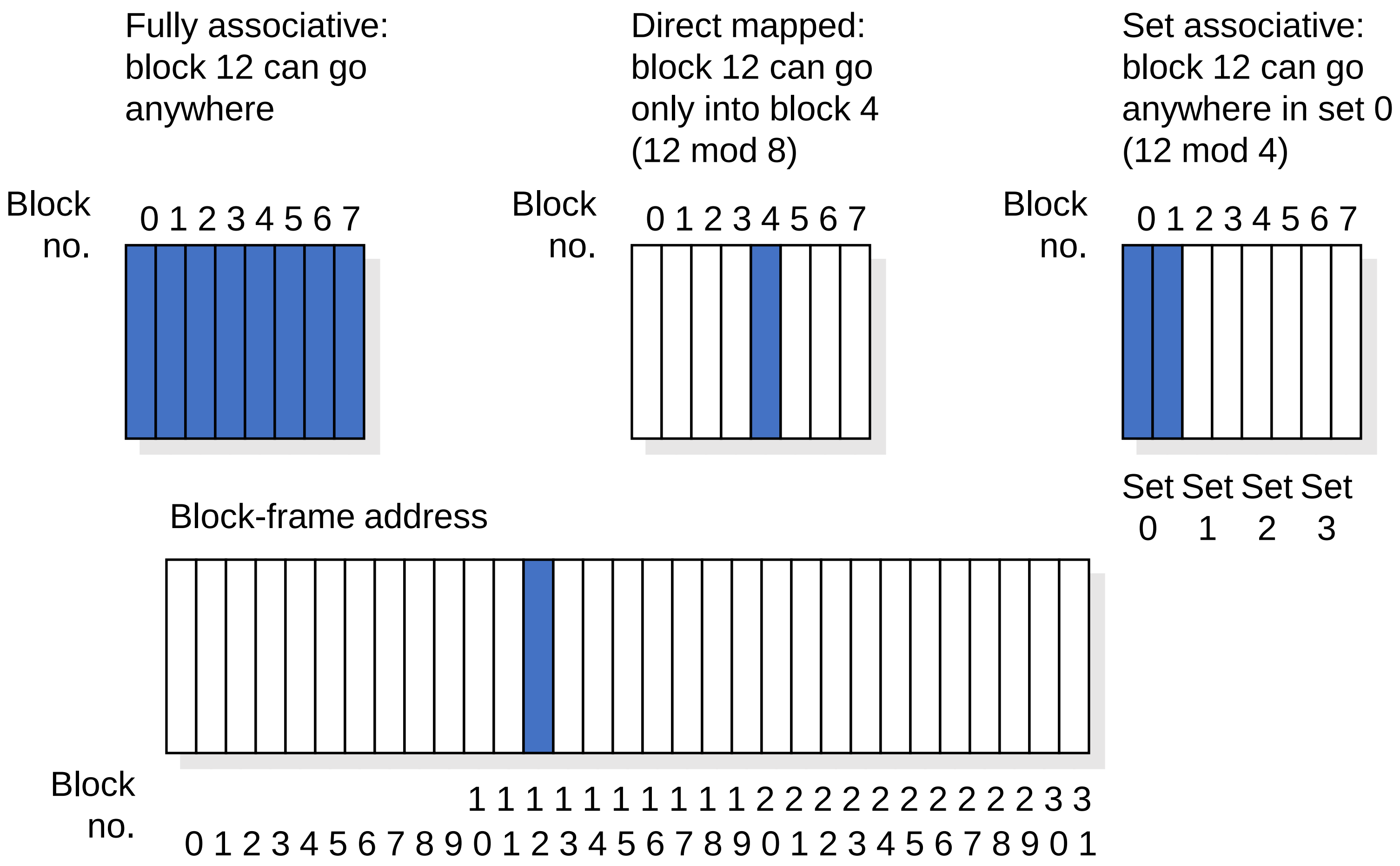
- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each user vs. tag/data in cache
  - TLB is cache of Virtual⇒Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well

Address Translation & 3 Concept tests

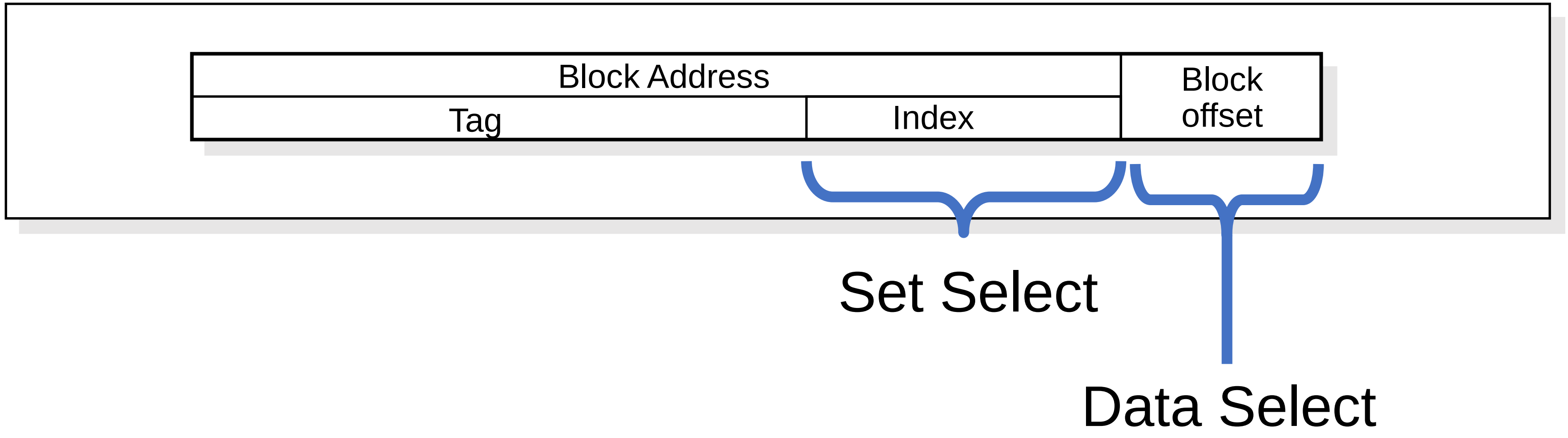


Q1: Where block placed in upper level

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



Q2: How is a block found in upper level?



- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag

Q3: Which block replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Miss Rates

Associativity:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What to do on a write hit?

- Write-through
  - update the word in cache block and corresponding word in memory
- Write-back
  - update word in cache block
  - allow memory word to be “stale”

  => add ‘dirty’ bit to each line indicating that memory be updated when block is replaced

  => OS flushes cache before I/O !!!
- Performance trade-offs?
  - WT: read misses cannot result in writes
  - WB: no writes of repeated writes

Three Advantages of Virtual Memory

1) Translation:

- Program can be given consistent view of memory, even though physical memory is scrambled
- Makes multiple processes reasonable
- Only the most important part of program ("Working Set") must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later

Three Advantages of Virtual Memory

2) Protection:

- Different processes protected from each other
- Different pages can be given special behavior
  - (Read Only, Invisible to user programs, etc).
- Kernel data protected from User programs
- Very important for protection from malicious programs ⇒ Far more "viruses" under Microsoft Windows
- Special Mode in processor ("Kernel more") allows processor to change page table/TLB

3) Sharing:

- Can map same physical page to multiple users ("Shared memory")

Why Translation Lookaside Buffer (TLB)?

- Paging is most popular implementation of virtual memory
- Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection
- Cache of Page Table Entries (TLB) makes address translation possible without memory access in common case to make fast

Virtual Memory Overview (1/4)

- User program view of memory:
  - Contiguous
  - Start from some set address
  - Infinitely large
  - Is the only running program
- Reality:
  - Non-contiguous
  - Start wherever available memory is
  - Finite size
  - Many programs running at a time

Bonus slide: Virtual Memory Overview (2/4)

- Virtual memory provides:
  - illusion of contiguous memory
  - all programs starting at same set address
  - illusion of ~ infinite memory
    (2^{32} or 2^{64} bytes)
  - protection

Virtual Memory Overview (3/4)

- Implementation:
  - Divide memory into “chunks” (pages)
  - Operating system controls page table that maps virtual addresses into physical addresses
  - Think of memory as a cache for disk
  - TLB is a cache for the page table

Virtual Memory Overview (4/4)

- Let's say we're fetching some data:
  - Check TLB (input: VPN, output: PPN)
    - hit: fetch translation
    - miss: check page table (in memory)
      - **Page table hit: fetch translation**
      - **Page table miss: page fault, fetch page from disk to memory, return translation to TLB**
  - Check cache (input: PPN, output: data)
    - hit: return value
    - miss: fetch value from memory

Implementing Protection with VM

- Multiple processes share a single main memory!
  - How to prevent one process from reading/writing over another's data?
    - Write access bit in page table
    - Non-overlapping page tables
    - OS controls the page table mappings
    - Page tables reside in OS's address space
  - How to share information?
    - Controlled by OS
    - Multi virtual addresses map to the same page table

Handling Page Fault and TLB Miss

- TLB Miss
- Page Fault
  - By exception handling mechanism
  - Page fault happens during the clock cycle used to access memory
  - EPC is used to store the address of the instruction that causes the exception
    - How to find the virtual address of the memory unit that holds the data when data page fault happens?
    - Prevent the completion of the faulting instruction – no writing!!
  - Cause register provide page fault reasons
  - OS does the following
    - Find the location of the referenced page on disk
    - Choose a physical page to replace. What about dirty pages?
    - Read the referenced page from disk

And in Conclusion…

- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well
- Virtual Memory allows protected sharing of memory between processes with less swapping to disk

Thank you