y

COMP

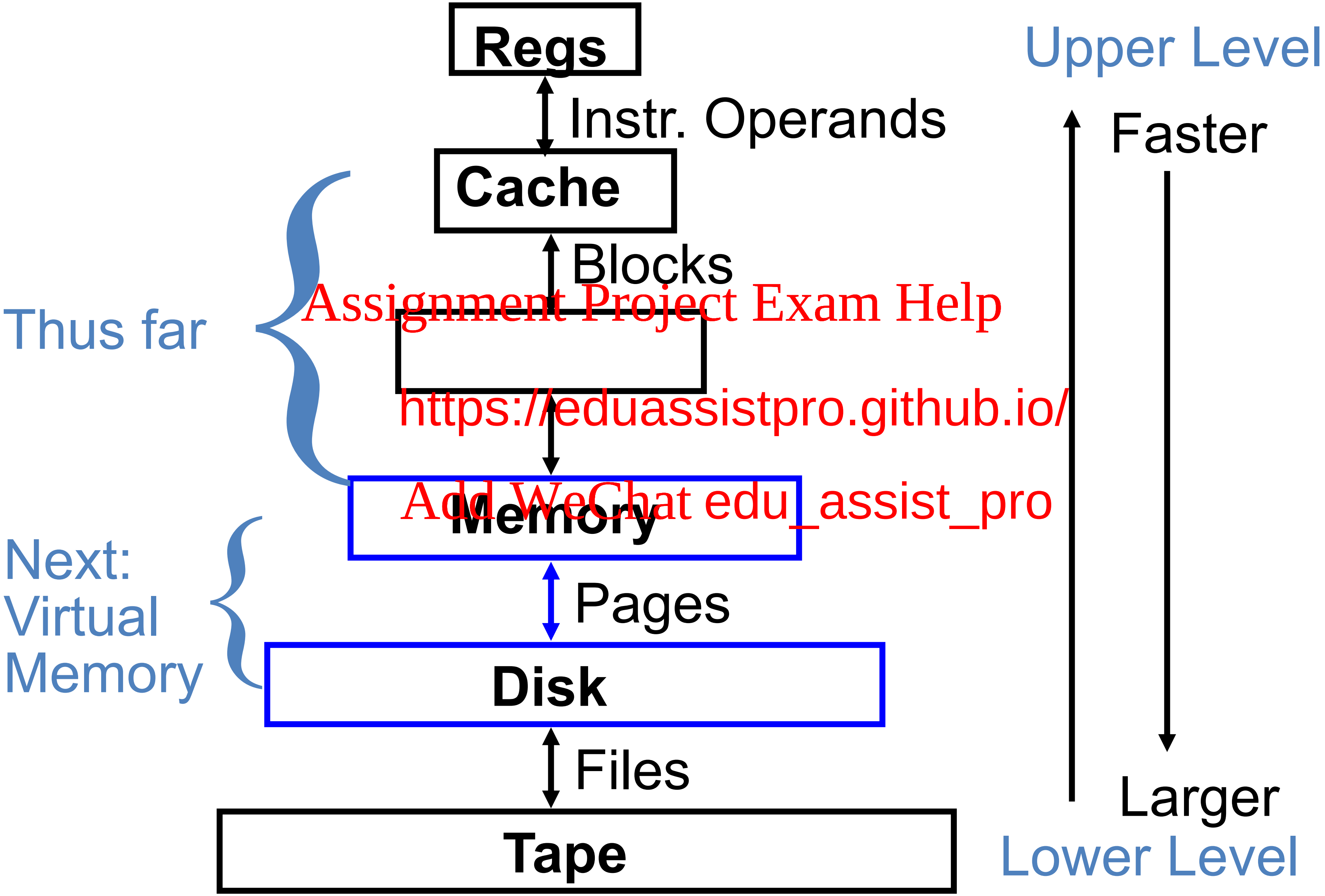
Review (1/2)

- Caches are NOT mandatory:
  - Processor performs arithmetic
  - Memory stores data
  - Caches simply make t [illegible]
- Each level of memory hierarchy is [illegible] set of next higher level
- Caches speed up due to temporal locality: store data used recently
- Block size > 1 word speeds up due to spatial locality: store words adjacent to the ones used recently

Review (2/2)

- Cache design choices:
  - size of cache: speed v. capacity
  - direct-mapped v. associative
  - for N-way set assoc:
  - block replacement policy
  - 2nd level cache?
  - Write through v. write back?
- Use performance model to pick between choices, depending on programs, technology, budget, ...

Another View of the Memory Hierarchy



Regs

Instr. Operands

Cache

Blocks

Thus far

Memory

Pages

Next:
Virtual
Memory

Disk

Files

Tape

Upper Level

Faster

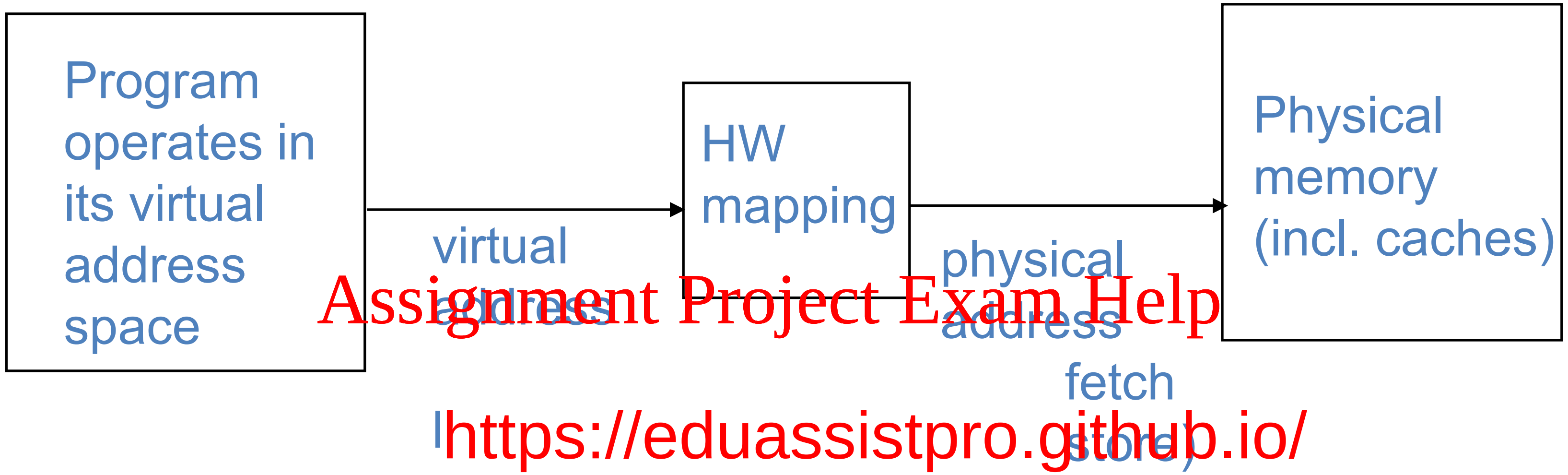
Larger

Lower Level

Virtual Memory

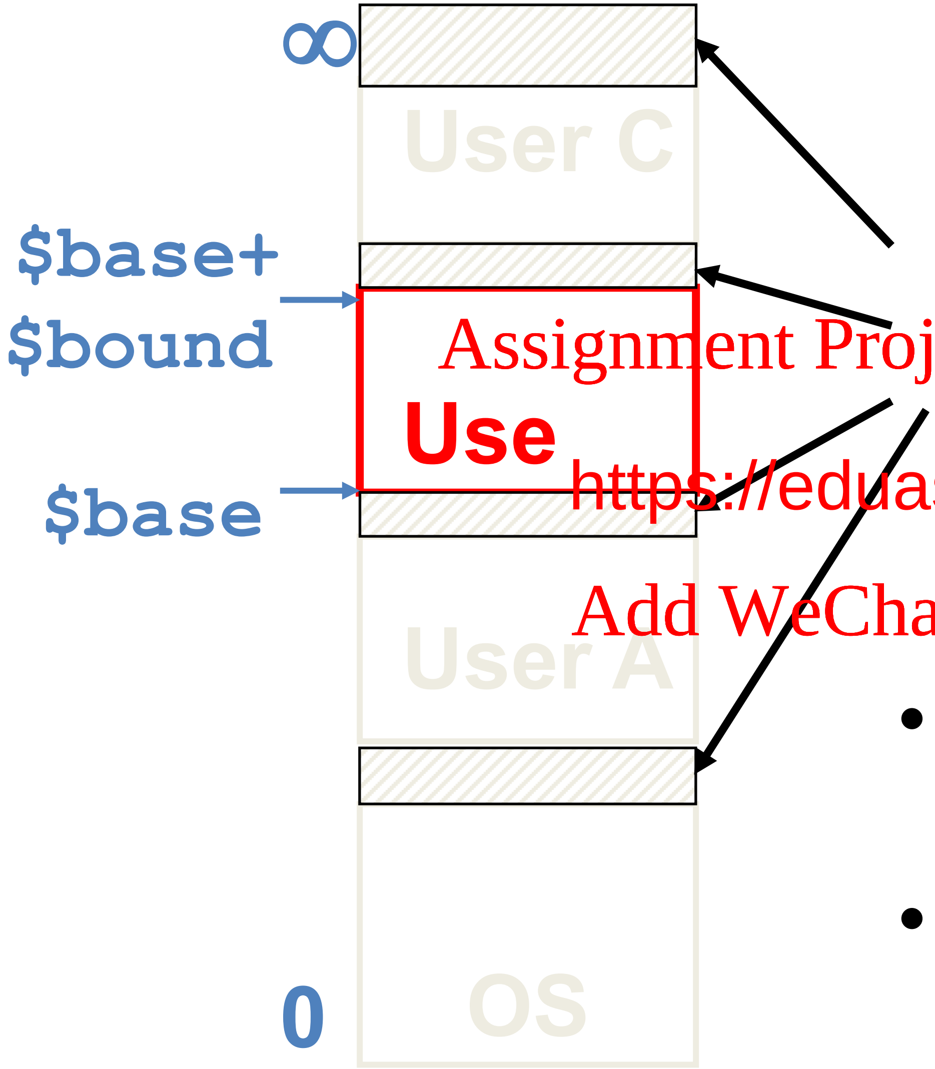
- If Principle of Locality allows caches to offer (usually) speed of cache memory with size of DRAM memory, then recursively why not use at next level to give speed of DRAM memory, size of Disk
- Called "Virtual Memory"
  - Also allows OS to share memory, protect programs from each other
  - Today, more important for protection vs. just another level of memory hierarchy
  - Historically, it predates caches

Virtual to Physical Addr. Translation


Program operates in its virtual address space
virtual address
HW mapping
physical address
fetch
store)
Physical memory (incl. caches)

- Each program operates in its virtual address space; as if it were the only program running
- Each "process" is protected from the other
- OS can decide where each goes in memory
- Hardware (HW) provides virtual -> physical mapping

Simple Example: Base and Bound Reg



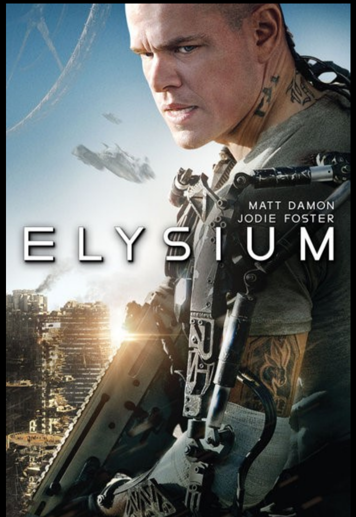
Enough space for User D, but discontinuous ("fragmentation problem")

isconti

- **Also want Process size >> memory size**
- **Base/Bound trick is not enough!**

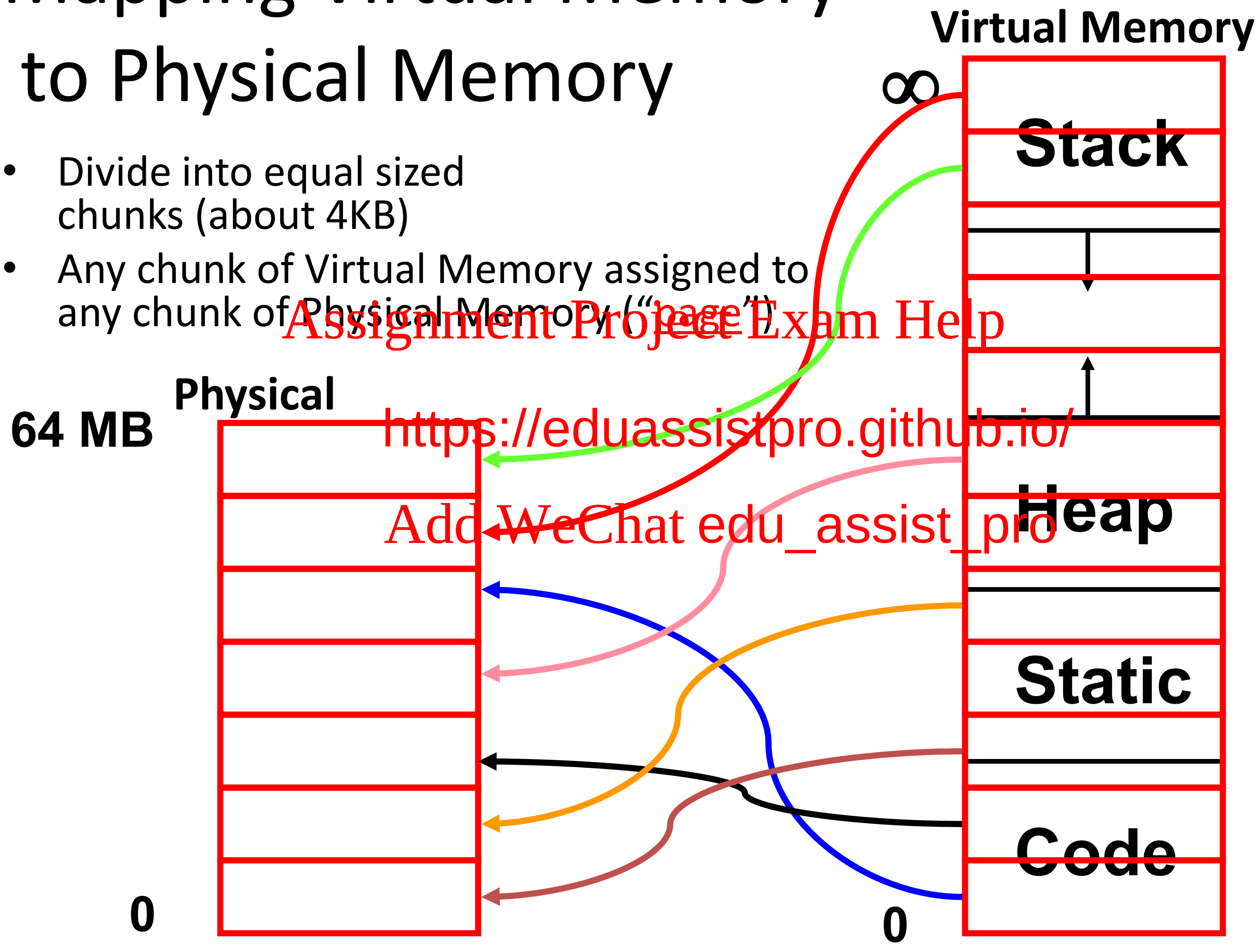
=> use Indirection!

GDT: Global descriptor table
On 286 machines, it allowed for base and bound, while indirection and virtual memory was not introduced to x86 chips until the 386 (in 1986)



Mapping Virtual Memory to Physical Memory

- Divide into equal sized chunks (about 4KB)
- Any chunk of Virtual Memory assigned to any chunk of Physical Memory (“page”)



Virtual Memory Mapping Function

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

Virtual address:	**Page Number**	**Offset**

- Use table lookup of mappings:
  - Page number is index

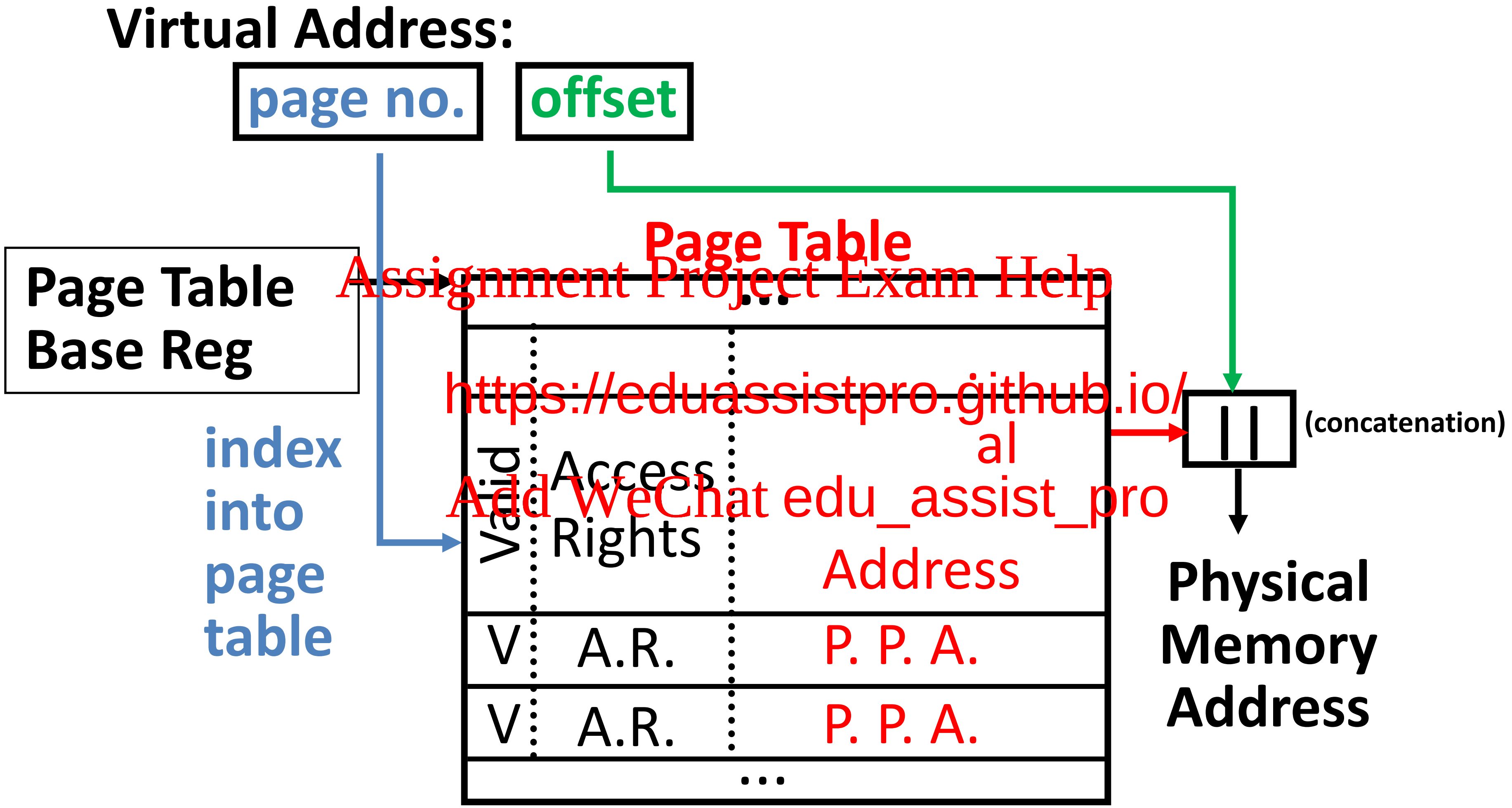
- Virtual Memory Mapping Function
  - Physical Offset = Virtual Offset
  - Physical Page Number
    = PageTable[Virtual Page Number]
  - (P.P.N. also called “Page Frame”)

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 aroun

- Each process running in the ope tem 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

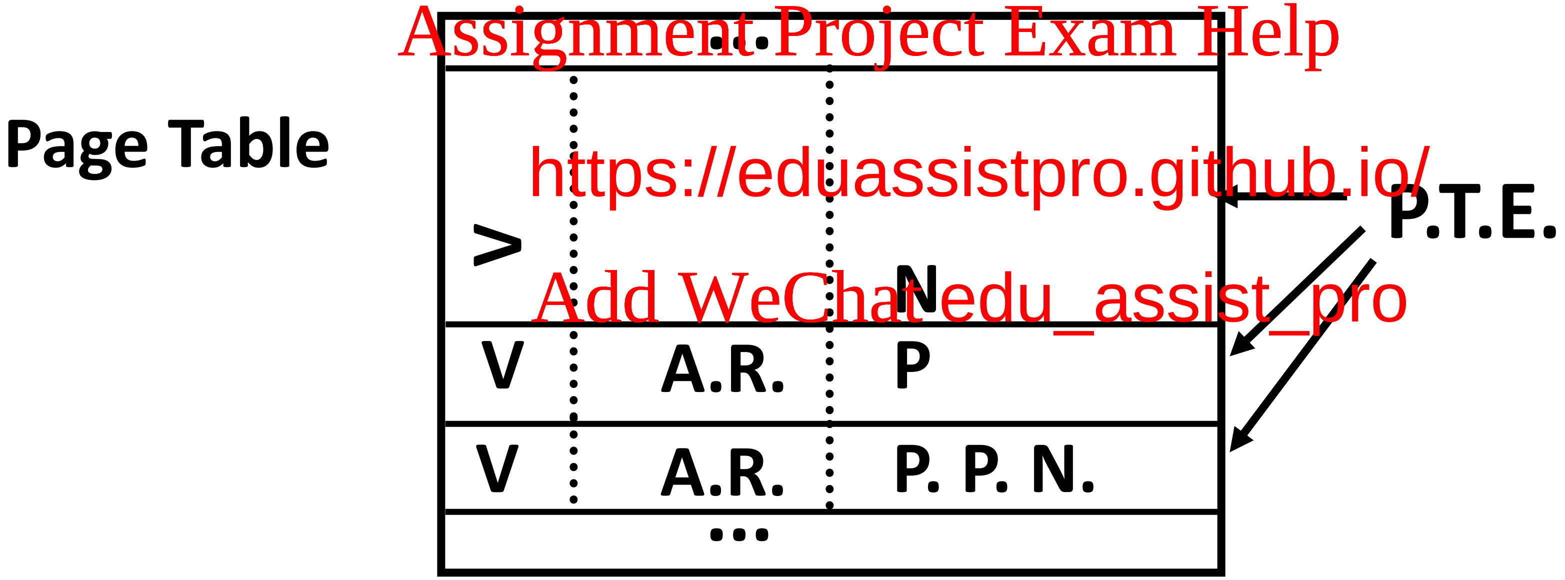
Address Mapping: **Page Table**



Page Table located in physical memory

Page Table Continued

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



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

Notes on Page Table

- Solves Fragmentation problem: all chunks same size, so all holes can be used
- OS must reserve “***Swap Space***” on disk for each process

- To grow a pro tem

  - If unused pa
  - If not, OS swaps some old pa

  - (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

Analogy

- Book title like virtual address “See MIPS Run”
- Call number like physical address QA76.9 A73 S88 2007
- Card catalogue like pa om book title to call number
- On card for book, in local library *or* r branch like valid bit indicating in main memory *or* on disk
- On card, loan restrictions are like access rights, for instance, reserved for 2-hour in library use, or 2-week checkout

Comparing the 2 levels of hierarchy

Cache Version	**Virtual Memory Version**
Block (or Line)	Page
Miss	Page Fault
Block Size: 32-64B	
Placement: Direct Mapped, N-way Set Associative	
Replacement: LRU or Random or…	Least Recently Used (LRU)
Write Thru or Back	Write Back

Virtual Memory Problem #1

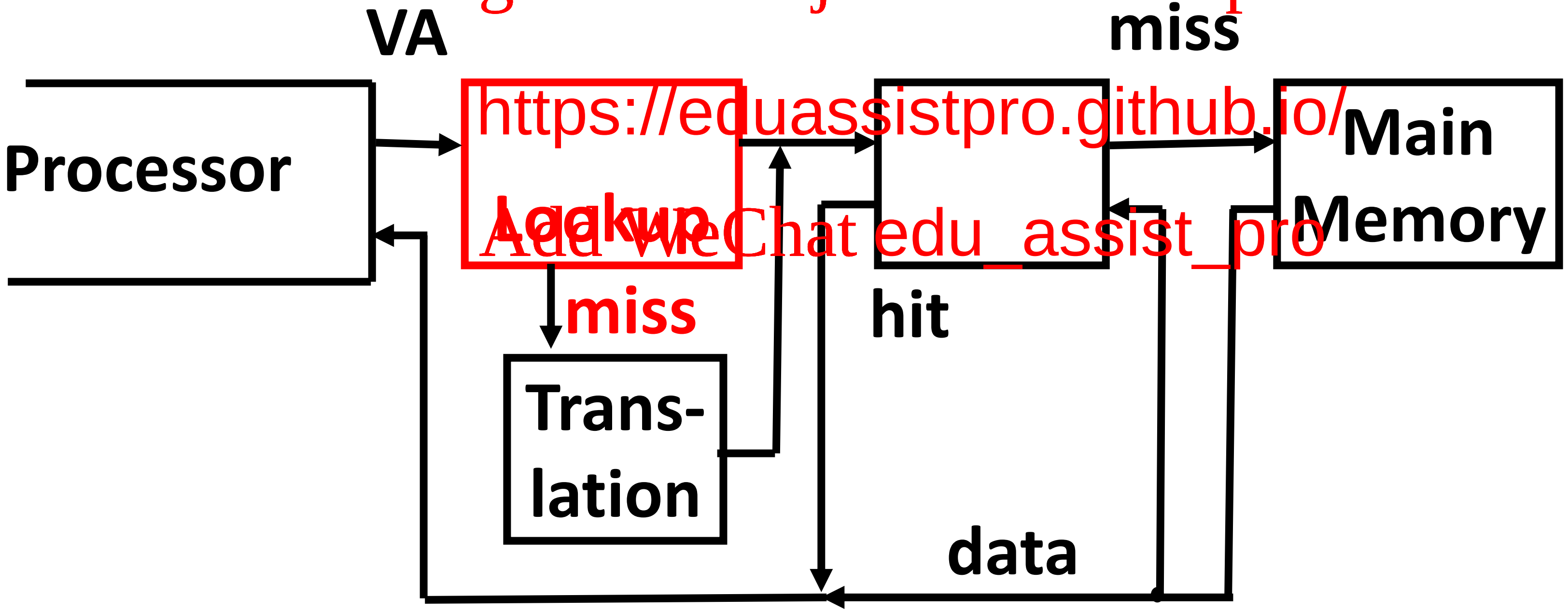
- Map every address ⇒ 1 indirection via Page Table in memory per virtual address ⇒ 1 virtual memory access
  - This implies 2 physic **SLOW!**

- Observation: since l ata, there must be locality in ***virtual address transl*** those pages

- Since small is fast, why not use a small cache of virtual to physical address translations to make translation fast?
- For historical reasons, this cache is called a ***Translation Lookaside Buffer***, or ***TLB***

Translation Look-Aside Buffers

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



Typical TLB Format

Virtual Address	Physical Address	Dirty	Ref	Valid	Access Rights

- TLB just a cach pings
- TLB access time comparable t
  (much less than main memory access time)

- Dirty: since we use “write back” policy, need to know whether or not to write page to disk when replaced
- Ref: Used to help calculate LRU on replacement
  - Can be cleared periodically by OS, then checked later to see if page was **referenced**

What if page not in TLB?

- Option 1: Hardware checks page table and loads new Page Table Entry into TLB

- Option 2: Hardware S to decide what to do
  - This is a simple and f

- MIPS follows Option 2: Hardwar nothing about page table
  - That is, there is no “page table base register” and instead there is only the TLB

TLB Miss (simple strategy)

- If the address is not in the TLB, MIPS traps to the operating system
  - When in the operating system, we don't do translation (turn off virtual memory)
- The operating syste gram caused the TLB fault, page fault, and knows wh ual address desired was requested
  - So we look the entry up in the page table
  - Then we add the entry to the TLB
  - Then we resume the program again at the instruction that failed (it will not have a TLB miss next time)

What if the data is on disk?

- We load the page off the disk into a free block of memory, using a ***DMA transfer***

  - Meantime we switc s waiting to be run

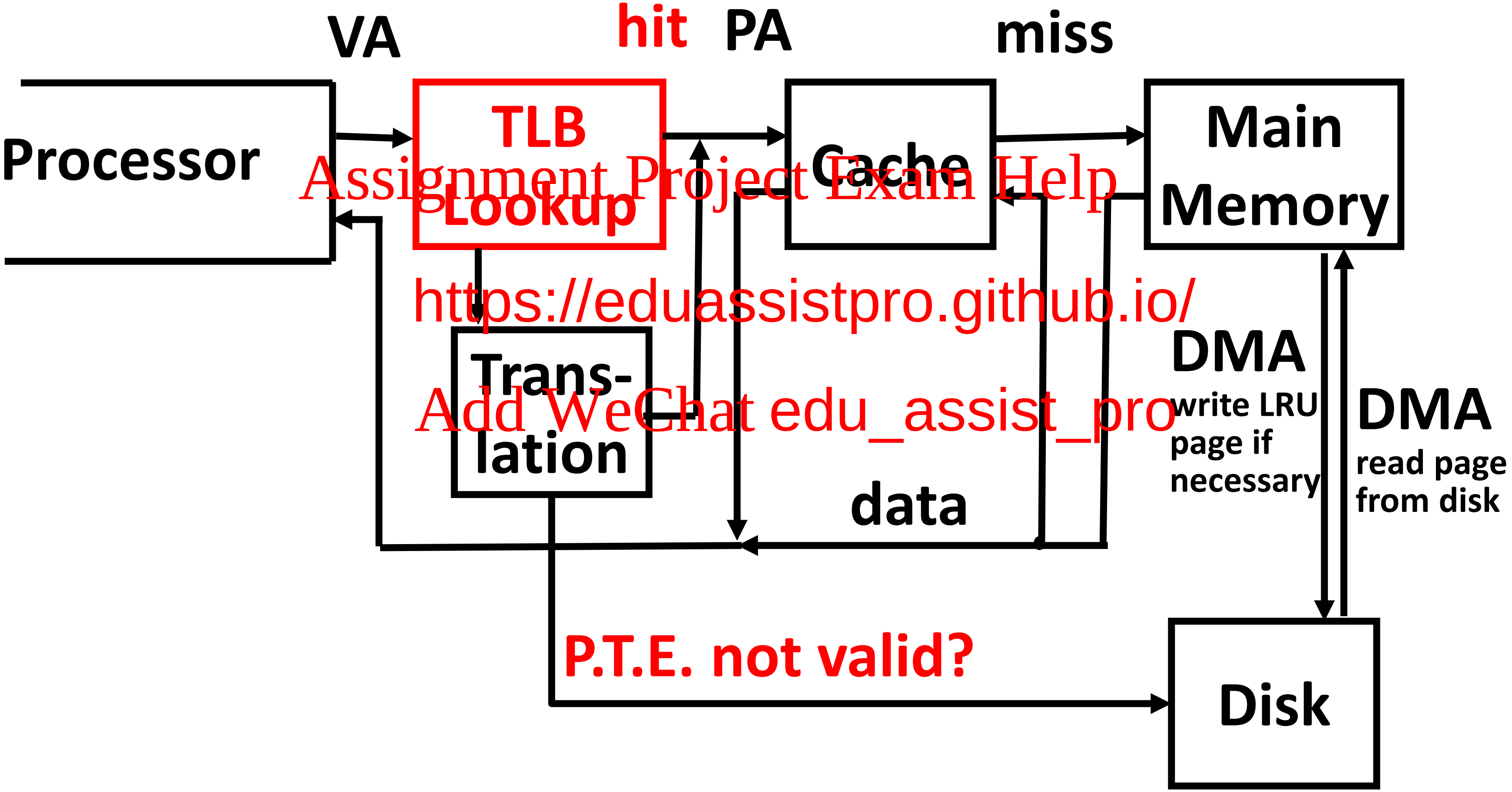
- When the DMA is co interrupt, and can then update the process's page table

  - So when we switch back to the task, the desired data will be in memory

What if we don't have enough memory?

- Chose some physical page belonging to a program,
  - We chose the physical page to evict based on replacement policy (e.g., LRU)
  - If chosen pa nto the disk
  - If chosen pa up-to-date), then we can simply overw ta in memory
- The program previously using the chosen page must have its page table updated to reflect the fact that its memory moved somewhere else.
- Finally, the OS can update our program’s page table to use this physical page

Data on Disk?


VA
hit
PA
miss
Processor
TLB
Lookup
Cache
Main
Memory
Trans-
lation
data
DMA
write LRU
page if
necessary
DMA
read page
from disk
P.T.E. not valid?
Disk

Virtual Memory Problem #2

- Not enough physical memory! Suppose 4KB pages and…
  - Have only 64 MB (2^{26} B) of physical memory
  - N processes, each given 4 GB (2^{32} B) of virtual memory
  - How many virtual pag N=1 process?
  - For what N will we ha hysical page?
- Spatial Locality to the rescue
  - Each page is 4 KB, lots of nearby references
- Even for huge programs, typically only accessing a few pages at any given time
  - The "Working Set" of recently used pages

Virtual Memory Problem #3

- Page Table too big!
  - 4 GB Virtual Memory ÷ 4 KB page
    - ⇒ approximately 1 million Page Table Entries, each taking up 1
    - ⇒ 4 MB just for Pa ,
      25 processes **will need 100 M** Tables!

- Variety of solutions to tradeoff memory size of *mapping function* for *slower TLB misses*
  - Make TLB large enough, highly associative so rarely miss on address translation
  - *Alternative mapping functions are not in the scope of this course*

Things to Remember 1/2

- Apply Principle of Locality Recursively
- Reduce Miss Penalty? add a (L2) cache
- Manage memory to e
  - Originally included protection as b protection is critical
  - Use Page Table of mappings vs. tag/data in cache
- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB

Things to Remember 2/2

- Virtual Memory allows protected sharing of memory between processes with less swapping to disk, less fragmentation than always-swap, or base/bound
- Spatial and Temporal king Set of Pages is all that must be in memory for process to well
- TLB to reduce performance cost of VM
- Need a more compact representation to reduce memory size cost of simple 1-level page table (*especially when using a 64-bit address space*)

Review and More Information

- Textbook 5th edition 5.7, Virtual Memory