Three function of Virtual Memory

- Translation
  - A program is given consistent view of memory, thus a programmer can write a program without knowing the physical layout of memory.
  - Address space can be continuous while it can grow
  - Uses table lookup to translate virtual address into physical address
  - Each process has its own address space => Each process has its own mapping.
- Protection
  - Different processes protected from each other by having different mapping between different processes.
  - Kernel data protected from User programs
- Sharing
  - Two processes can have a page table entry that is mapped to the same physical page.

Virtual Memory

Translation:

- Program can be given consistent view of memory (virtual memory), no matter how the physical memory is organized.
- 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
- Uses table lookup to translate virtual address into physical address

Virtual Memory

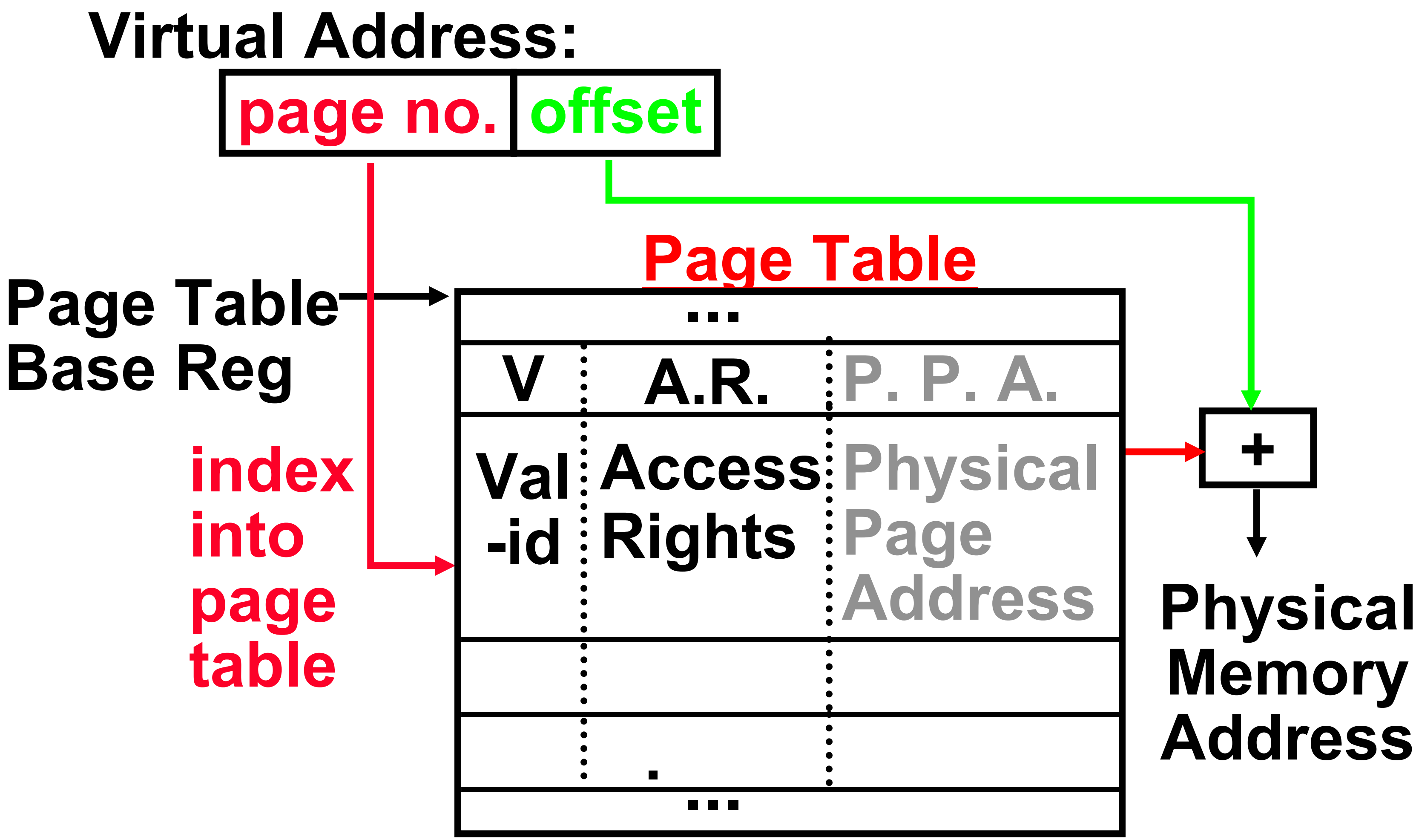
Protection:

- Each process has its own page table.
- OS changes page tables by changing contents of Page Table Base Register
- Different processes protected from each other
- Kernel data protected from User programs

Sharing:

- Two processes can have a page table entry that is mapped to the same physical page.

Address Mapping: Page Table



Page Table located in physical memory

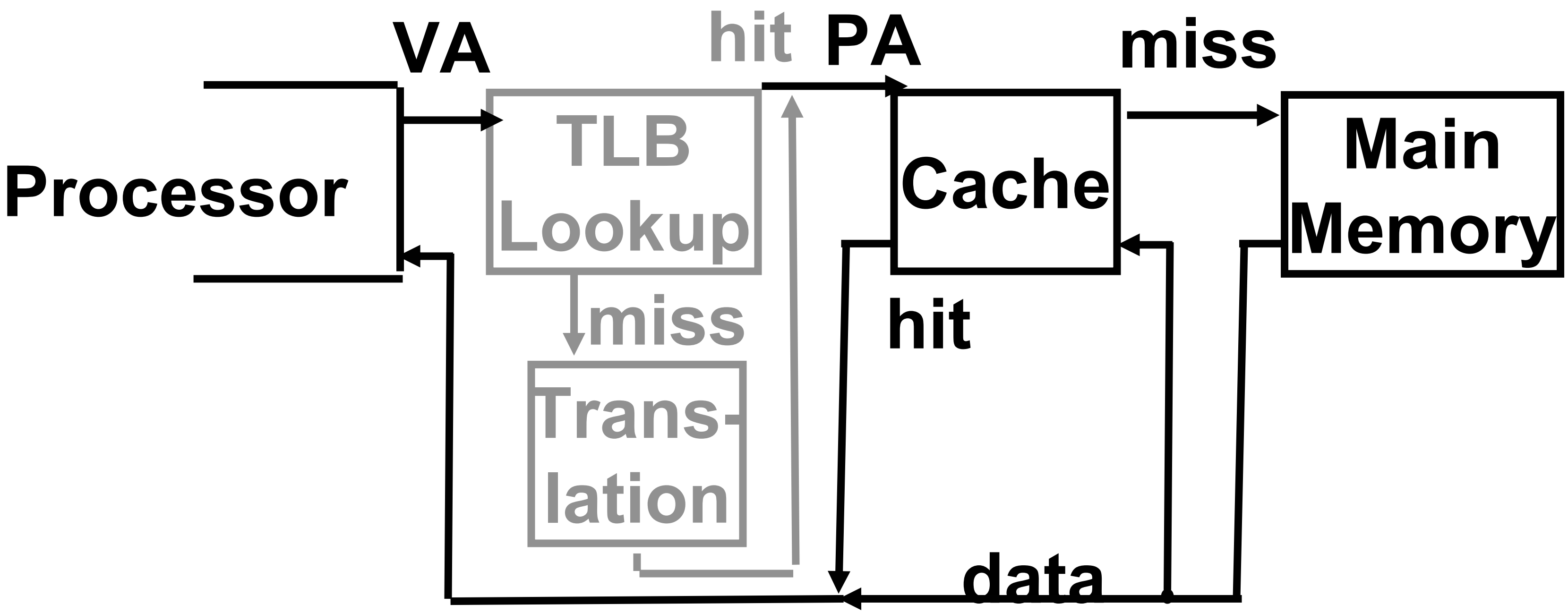
Virtual Memory Problem #1

- Map every address ⇒ 1 indirection via Page Table in memory per virtual address ⇒ 1 virtual memory accesses = 2 physical memory accesses ⇒ SLOW!
- Observation: since locality in pages of data, there must be locality in virtual address translations of 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, 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



On TLB miss, get page table entry from main memory

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
- MIPS follows Option 2: Hardware knows nothing about page table

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 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

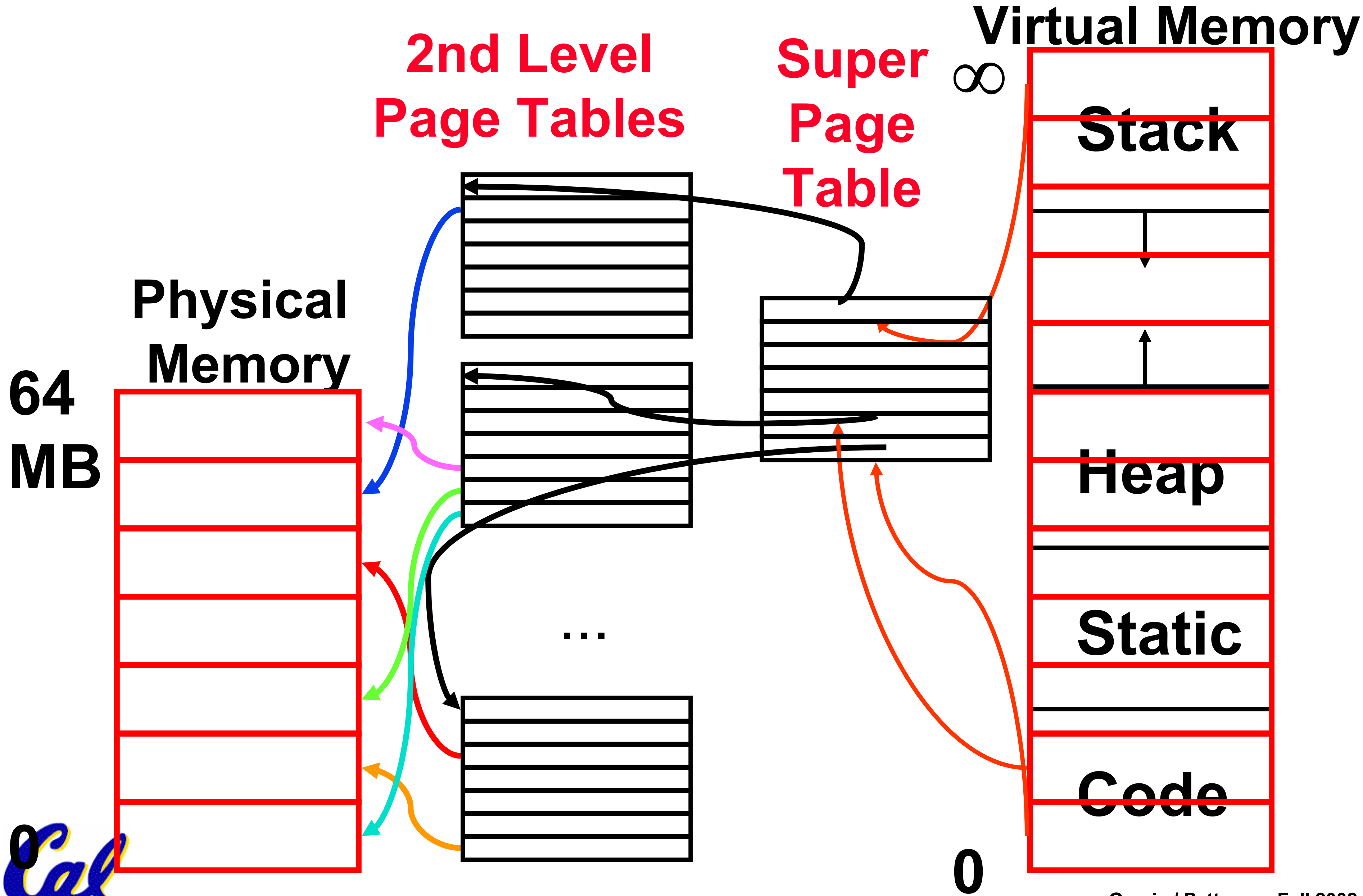
Virtual Memory Problem #2

- **Not enough physical memory!**
  - Only, say, 64 MB of physical memory
  - N processes, each 4 GB (2^{32} B) of virtual memory!
  - Could have 1K virtual pages/physical page!
- **Spatial Locality to the rescue**
  - Each page is 4 KB, lots of nearby references
- **No matter how big program is, at any time only accessing a few pages**
  - "Working Set": recently used pages

Virtual Memory Problem #3

- **Page Table too big!**
  - 4GB Virtual Memory $\div$ 4 KB page $\Rightarrow$ ~ 1 million Page Table Entries $\Rightarrow$ 4 MB just for Page Table for 1 process, 25 processes $\Rightarrow$ 100 MB for Page Tables!
- Variety of solutions to tradeoff memory size of mapping function for slower when miss TLB
  - Make TLB large enough, highly associative so rarely miss on address translation
  - CS 162 will go over more options and in greater depth

2-level Page Table



Page Table Shrink :

- Single Page Table

Page Number	Offset
20 bits	12 bits

- Multilevel Page Table

Super Page No.	Page Number	Offset
10 bits	10 bits	12 bits

- Only have second level page table for valid entries of super level page table

Definitions

- $\text{performance}(x) = \dfrac{1}{\text{execution\_time}(x)}$

" X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)}$$

° CPU execution time for program

= Clock Cycles for a program x Clock Cycle Time

° = $\dfrac{\text{Clock Cycles for a program}}{\text{Clock Rate}}$

Measuring Time using Clock Cycles (2/2)

- One way to define clock cycles:

Clock Cycles for program

= Instructions for a program
(called "Instruction Count")

x Average Clock cycles Per Instruction
(abbreviated "CPI")

- CPI one way to compare two machines with same instruction set, since Instruction Count would be the same