inst.eecs.berkeley.edu/~cs61c

UC Berkeley CS61C : Machine Structures

Lecture 34 – Virtual Memory I

2007-04-13

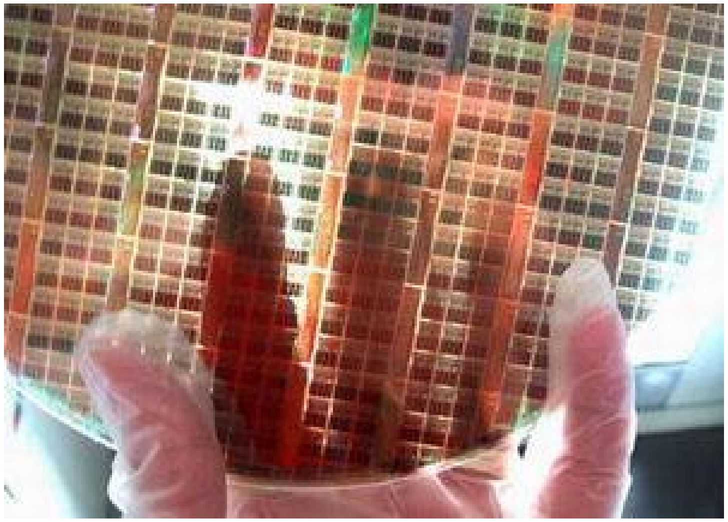
Do the reading!
Caches and VM are hard!

Dan's OH
today 3-4 pm

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

3D chips from IBM ⇒



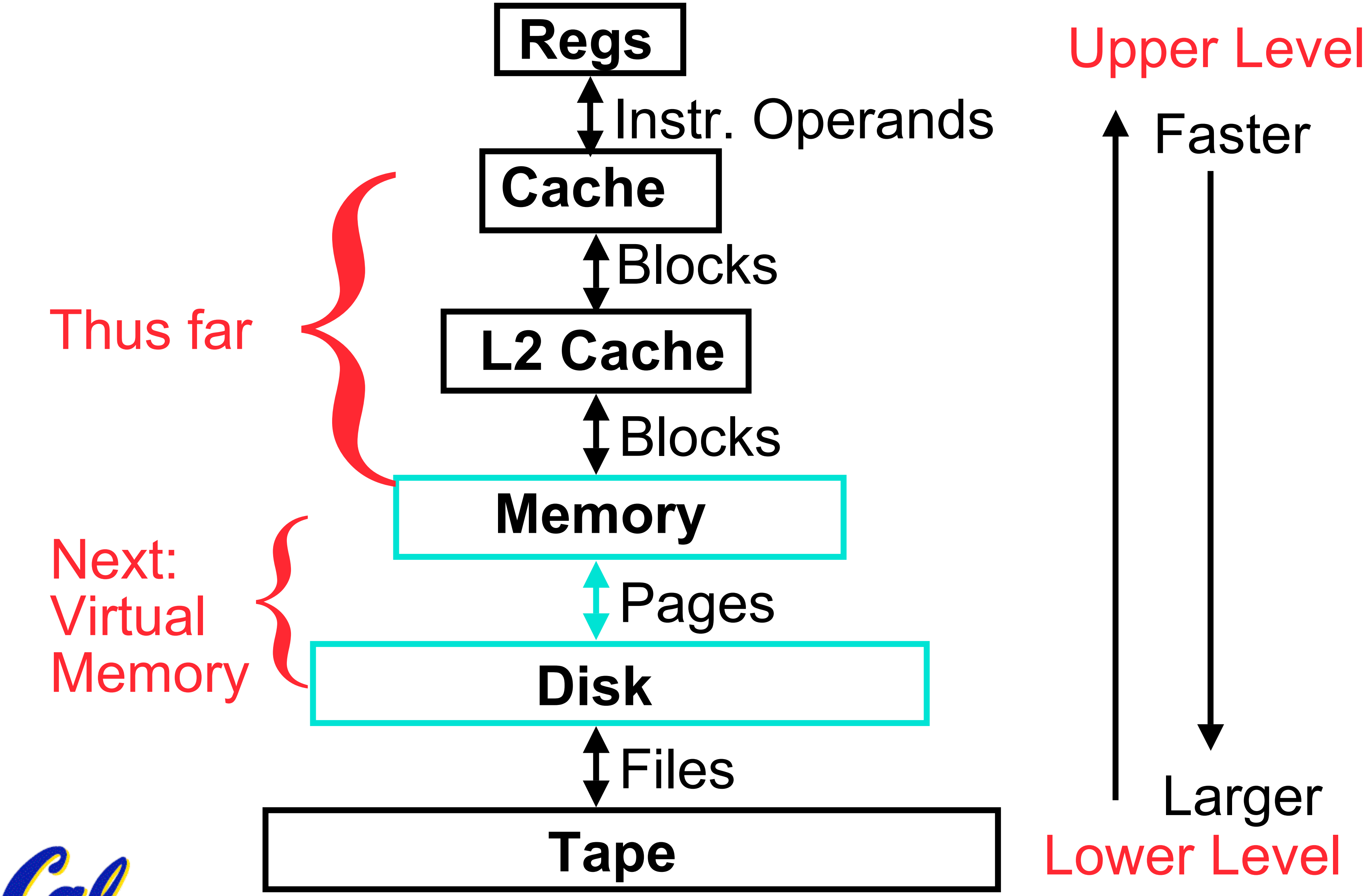
IBM claims to have found a way to build 3D chips by stacking them together. This "through-silicon via" technology is claimed to be the saving grace that extends Moore's law for now!

www.physorg.com/news95575580.html

Review: Caches

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

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?

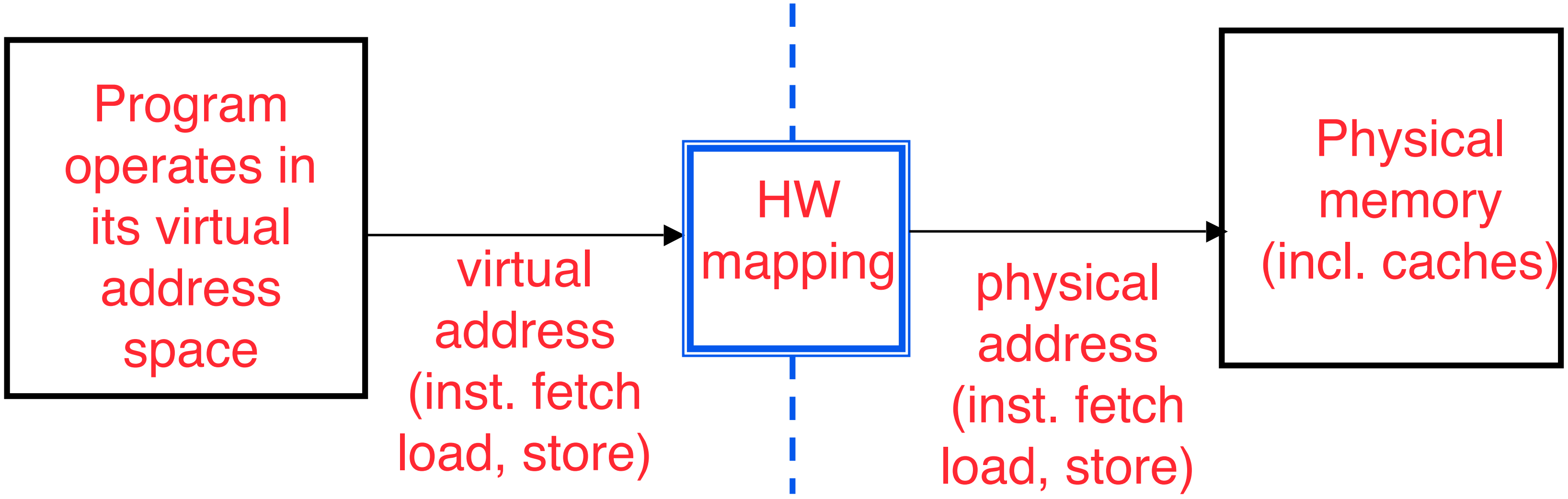
Memory Hierarchy Requirements

- Allow multiple processes to simultaneously occupy memory and 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 0x40000000. 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"
- Next level in the memory hierarchy:
  - Provides program with illusion of a very large main memory:
  - Working set of "pages" reside in main memory - others reside on disk.
- Also allows OS to share memory, protect programs from each other
- Today, more important for protection vs. just another level of memory hierarchy
- Each process thinks it has all the memory to itself
- (Historically, it predates caches)

Virtual to Physical Address Translation



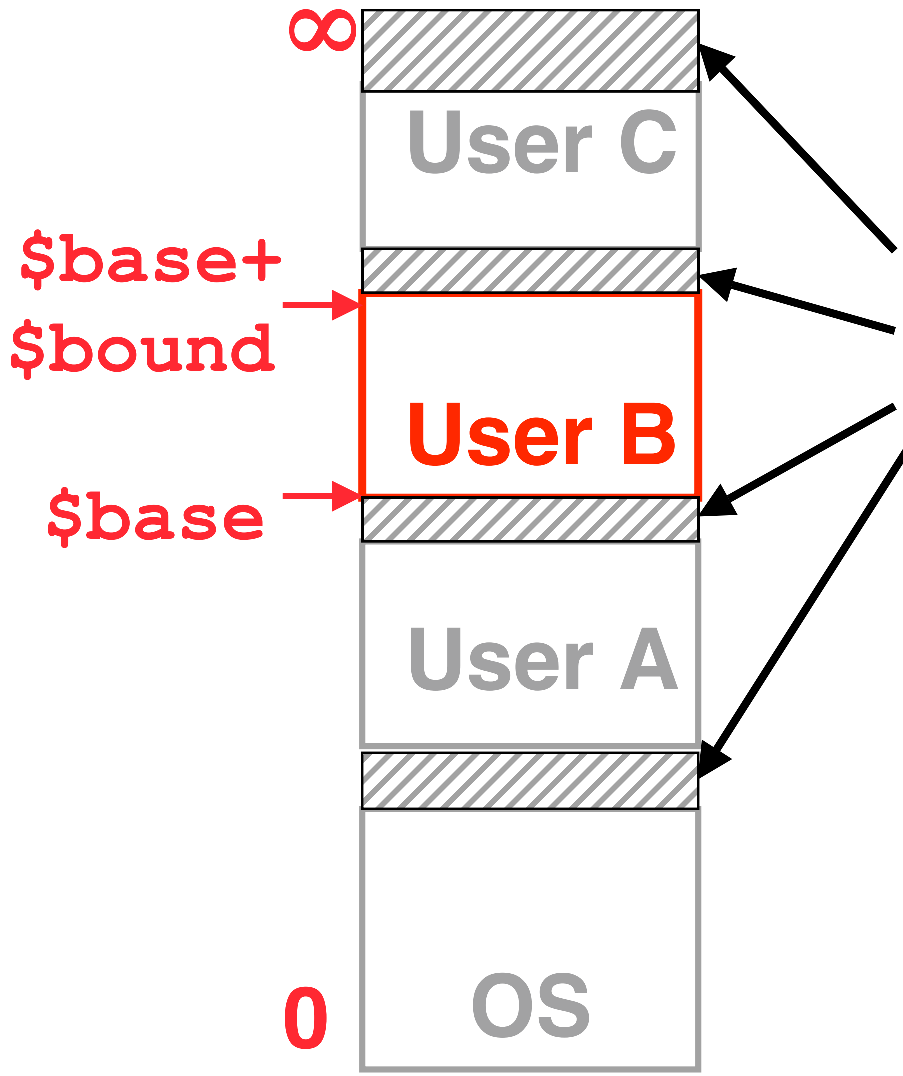
Program operates in its virtual address space
virtual address (inst. fetch load, store)
HW mapping
physical address (inst. fetch load, store)
Physical memory (incl. caches)

- **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 ⇒ physical mapping**

Analogy

- Book title like virtual address
- Library of Congress call number like physical address
- Card catalogue like page table, mapping from book title to call #
- On card for book, in local library vs. in another branch like valid bit indicating in main memory vs. on disk
- On card, available for 2-hour in library use (vs. 2-week checkout) like access rights

Simple Example: Base and Bound Reg



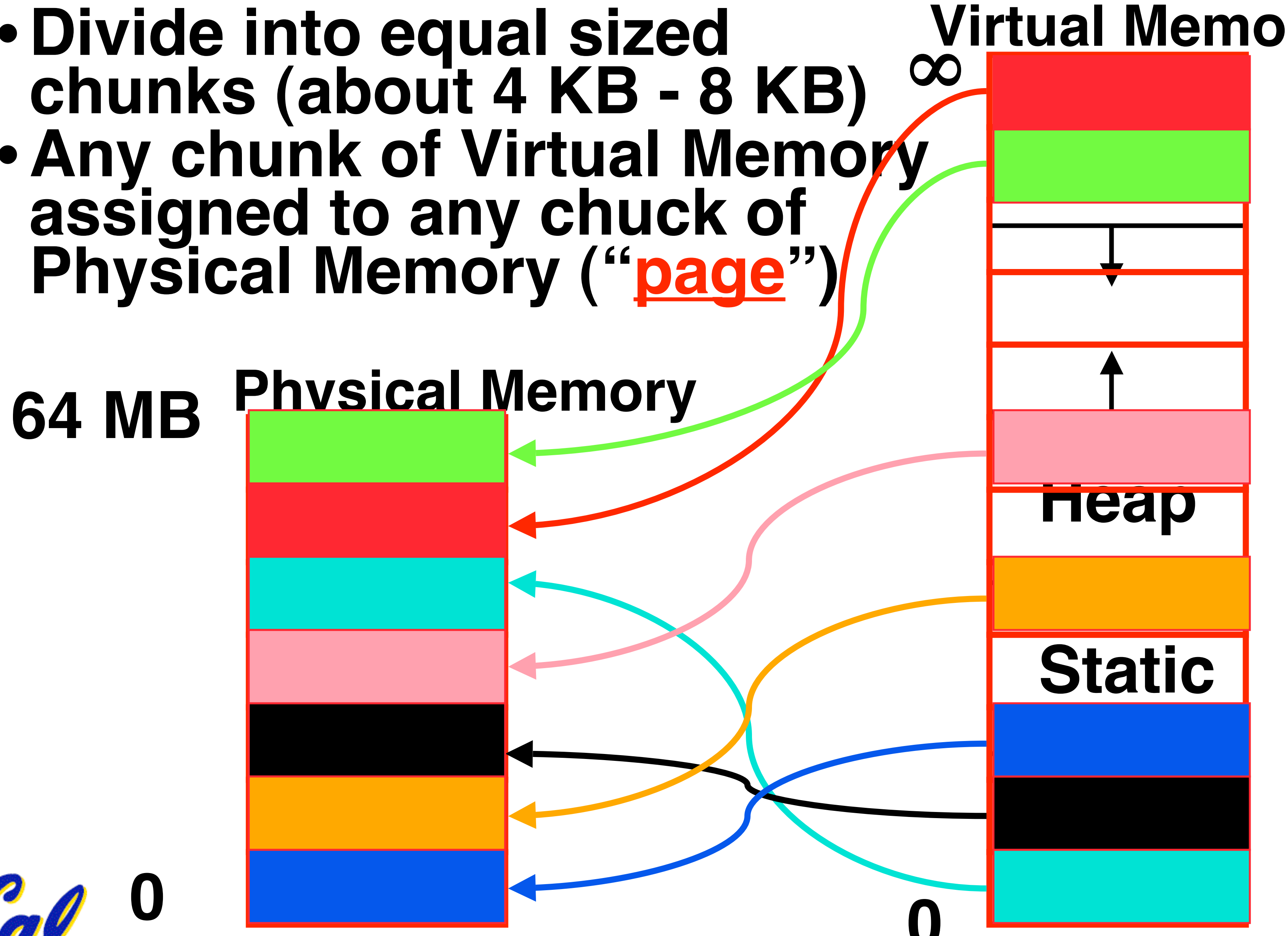
Enough space for User D, but discontinuous ("fragmentation problem")

- Want:
  - discontinuous mapping
  - Process size >> mem
- Addition not enough!

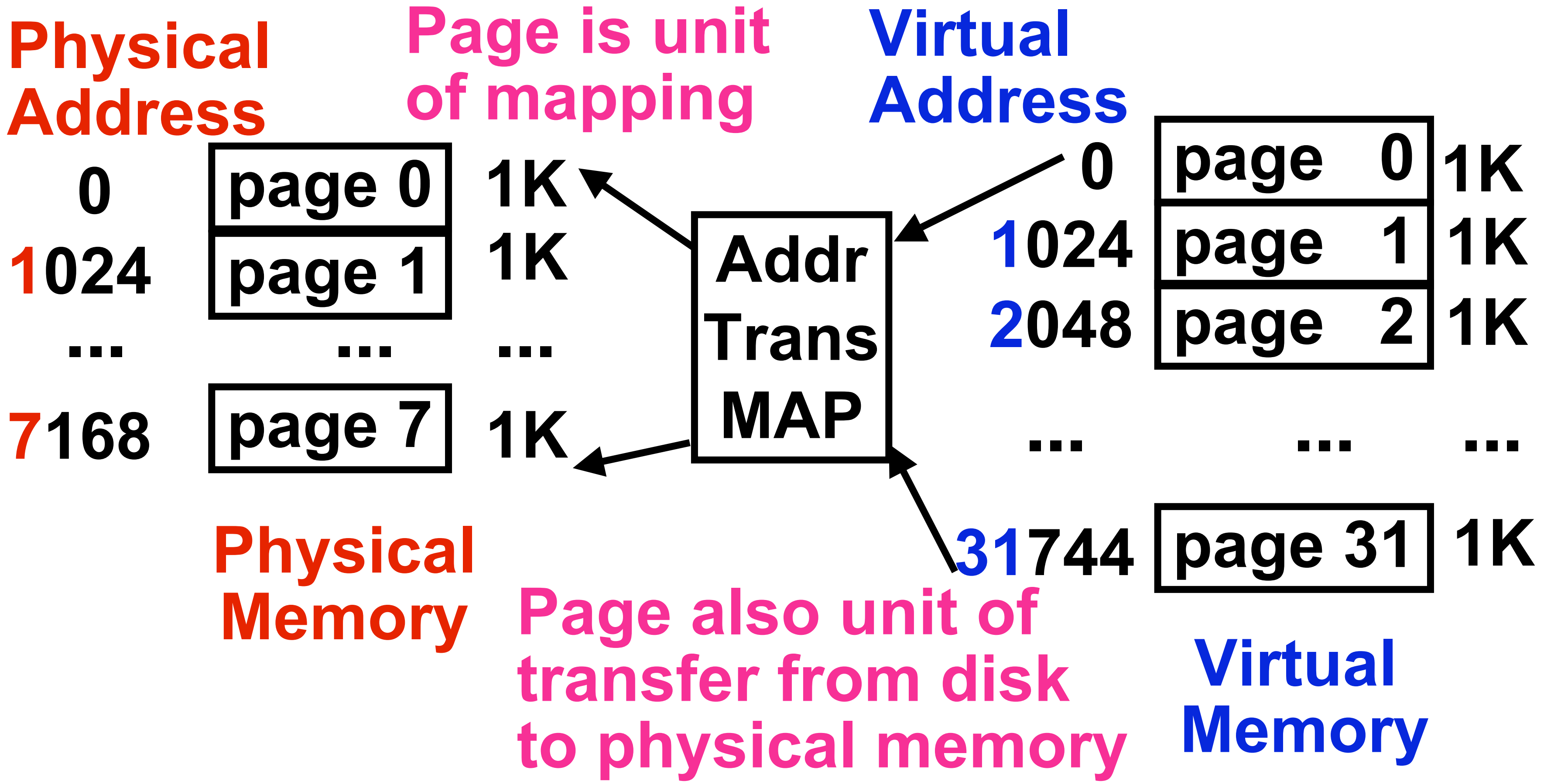
⇒ use Indirection!

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**")



Paging Organization (assume 1 KB pages)



Physical Address

Page is unit of mapping

Virtual Address

Physical Address	Physical Memory	
0	page 0	1K
1024	page 1	1K
...	...	...
7168	page 7	1K

Physical Memory

Addr Trans MAP

Virtual Address	Virtual Memory	
0	page 0	1K
1024	page 1	1K
2048	page 2	1K
...	...	...
31744	page 31	1K

Virtual Memory

Page also unit of transfer from disk to physical memory

Virtual Memory Mapping Function

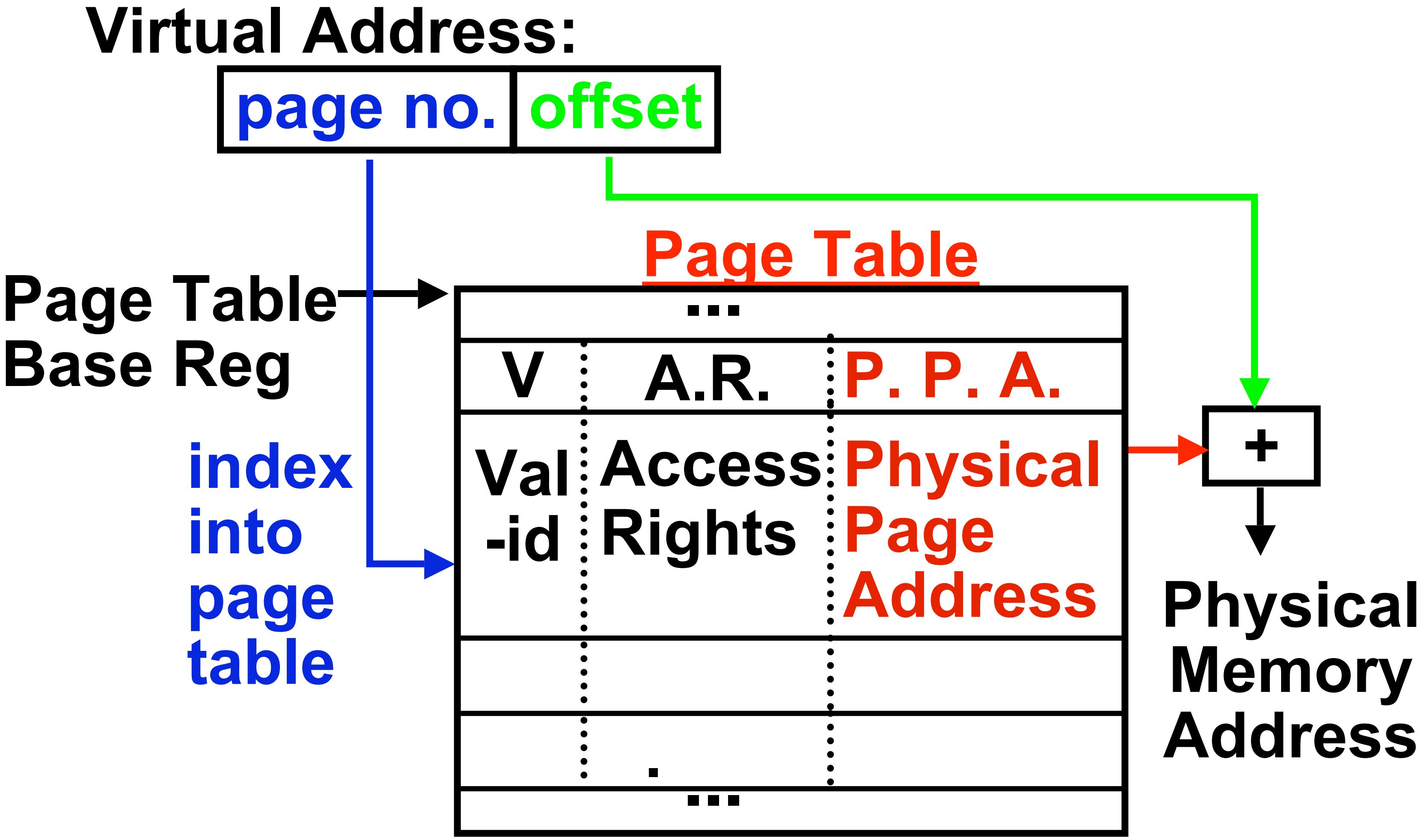
- Cannot have simple function to predict arbitrary mapping
- Use table lookup of mappings

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 = PageTable[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

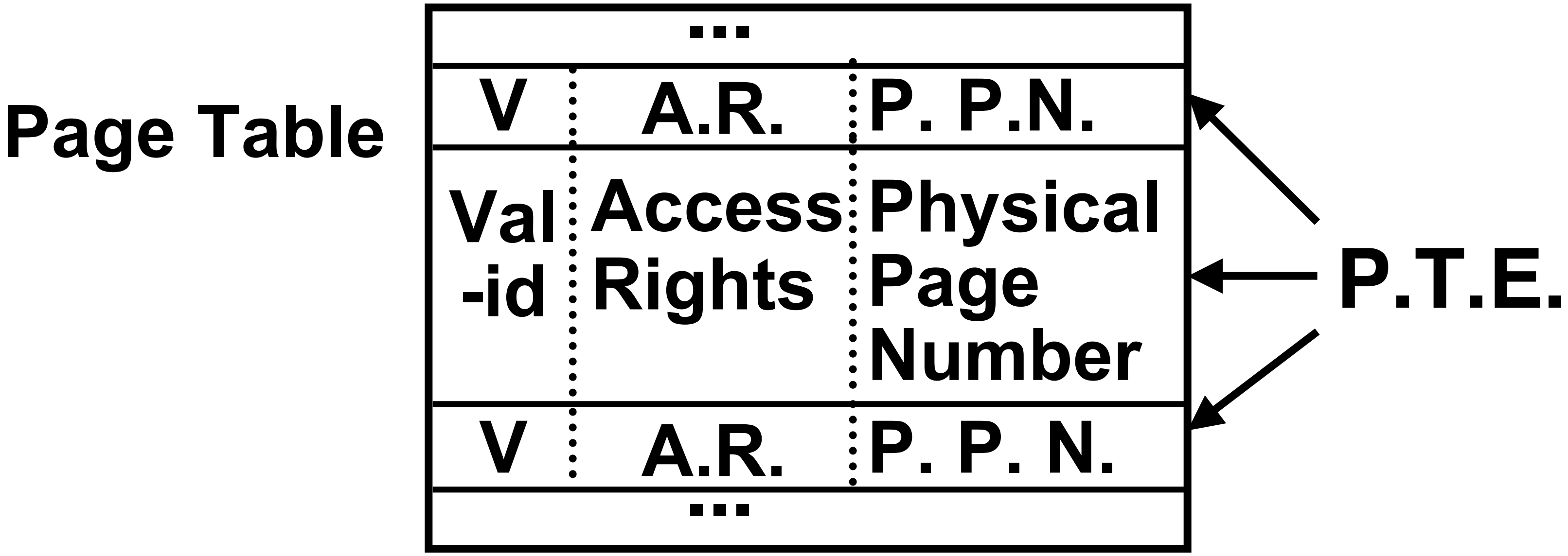
Remember the motivation for VM:

- 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!

What about the memory hierarchy?

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)



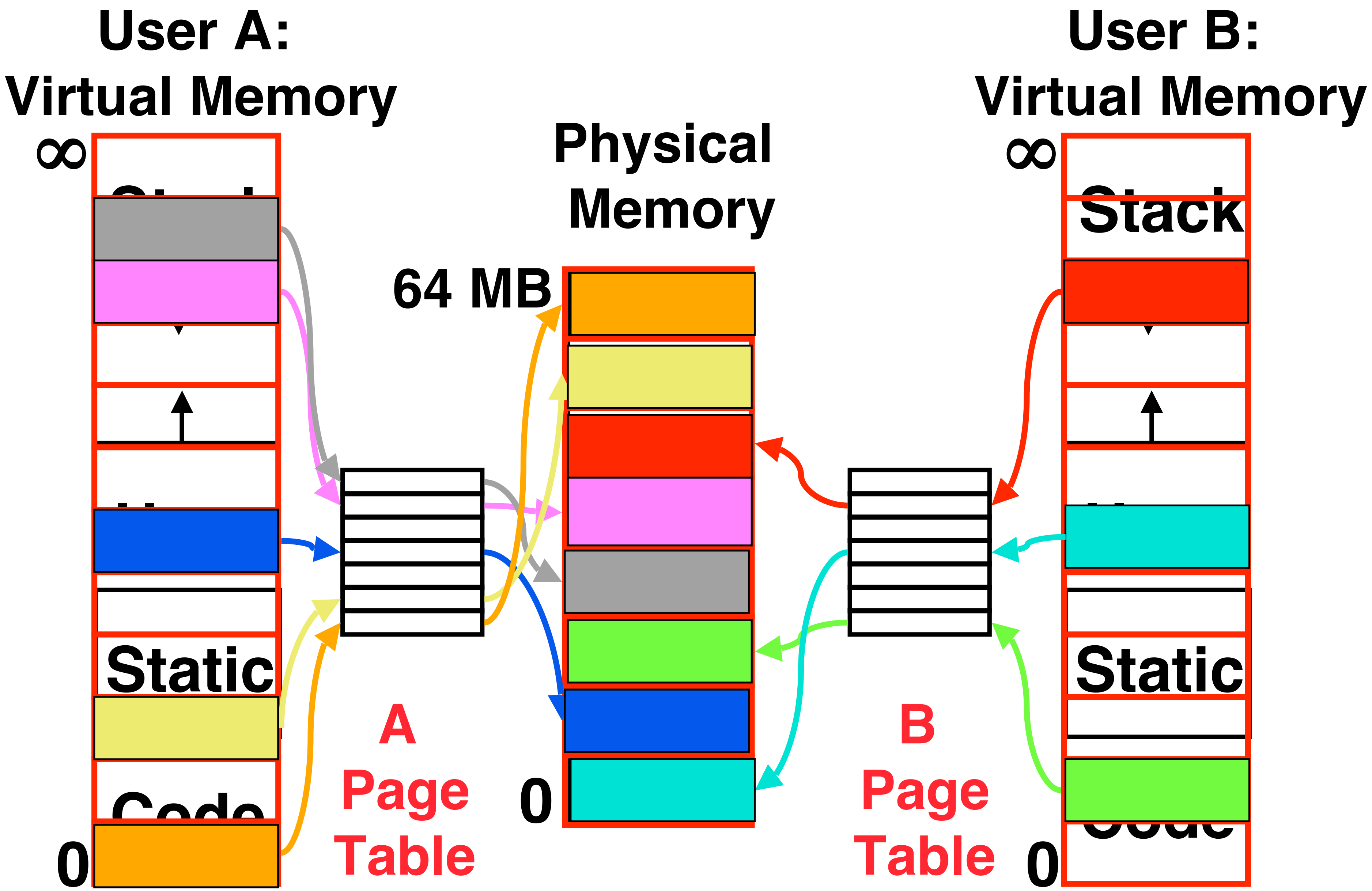
Page Table

...		
V	A.R.	P. P.N.
Val -id	Access Rights	Physical Page Number
V	A.R.	P. P. N.
...		

P.T.E.

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

Paging/Virtual Memory Multiple Processes



User A: Virtual Memory

Physical Memory

User B: Virtual Memory

A Page Table

B Page Table

Comparing the 2 levels of hierarchy

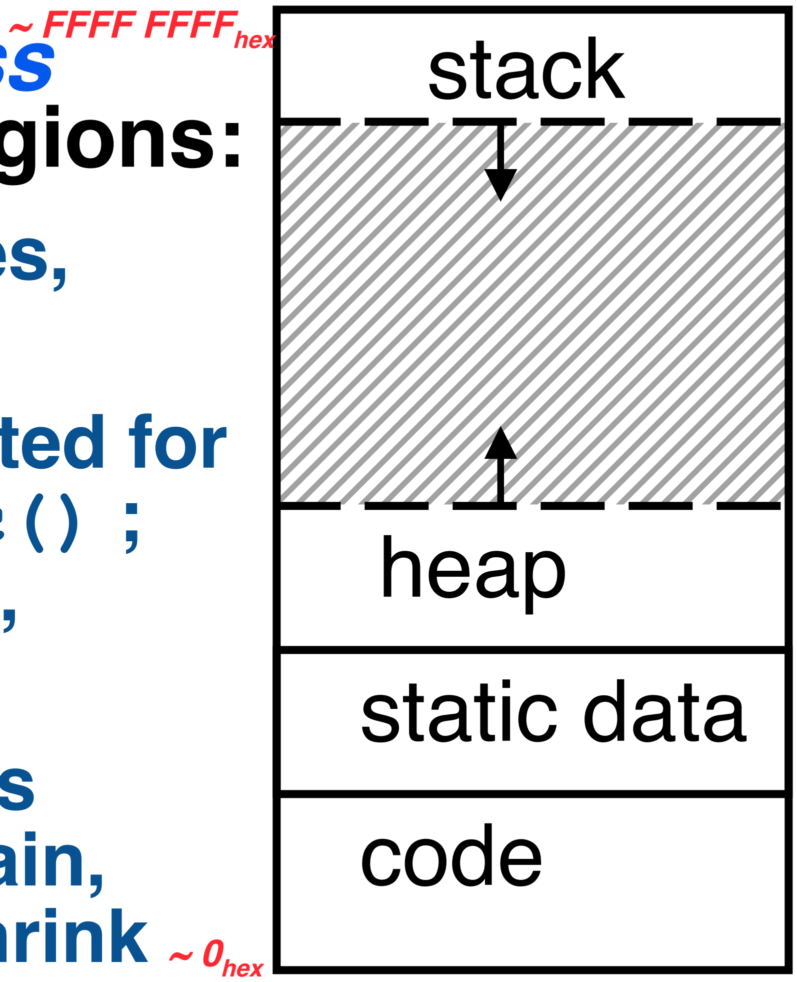
Cache version	Virtual Memory vers.
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-8KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

Notes on Page Table

- Solves Fragmentation problem: all chunks same size, 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

Why would a process need to "grow"?

- A program's *address space* contains 4 regions:
  - stack: local variables, grows downward
  - heap: space requested for pointers via `malloc()` ; resizes dynamically, grows upward
  - static data: variables declared outside main, does not grow or shrink
  - code: loaded when program starts, does not change



For now, OS somehow prevents accesses between stack and heap (gray hash lines).

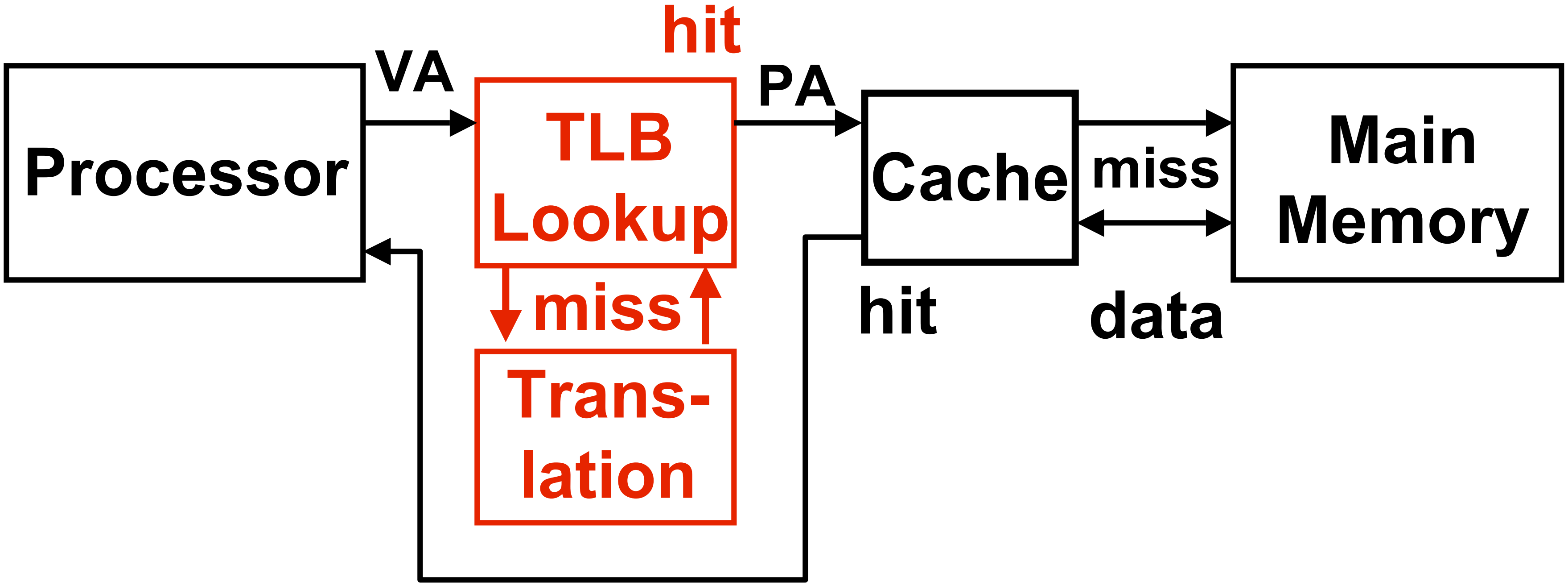
Virtual Memory Problem #1

- 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)

- 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

Peer Instruction

A. Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM

B. Cache management is done by hardware (HW), page table management by the operating system (OS), but TLB management is either by HW or OS

C. VM helps both with security and cost

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT

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