inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture #25 Virtual Memory I

2007-8-7

Scott Beamer, Instructor

inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture #25 Virtual Memory I
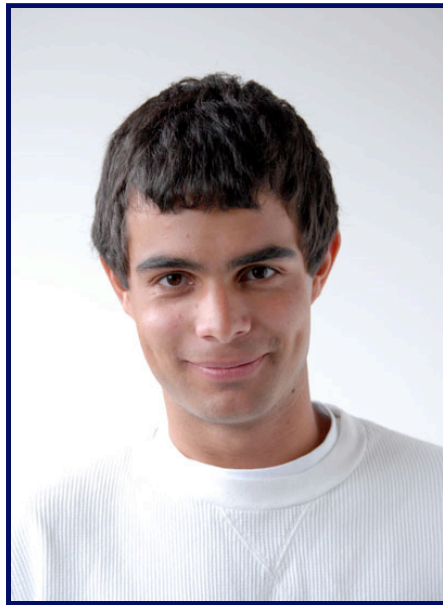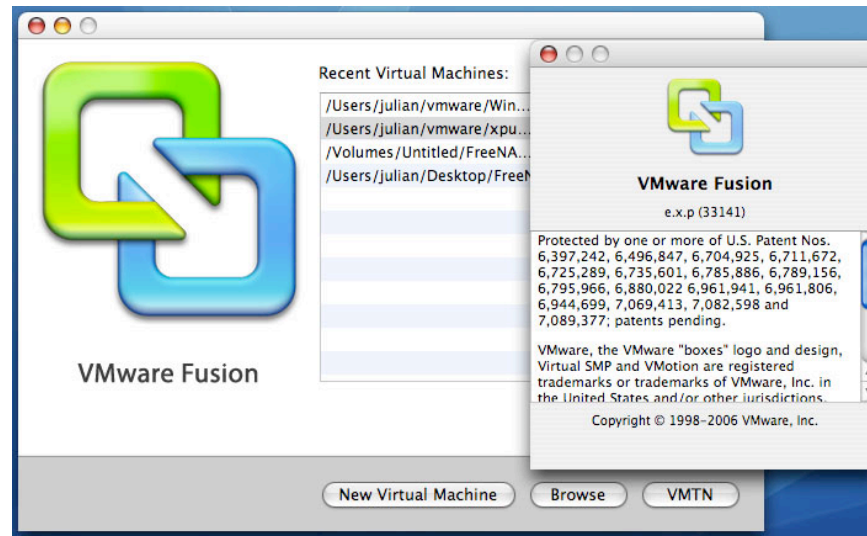
2007-8-7

Scott Beamer, Instructor

Vmware Fusion 1.0

VMware Fusion

www.vmware.com
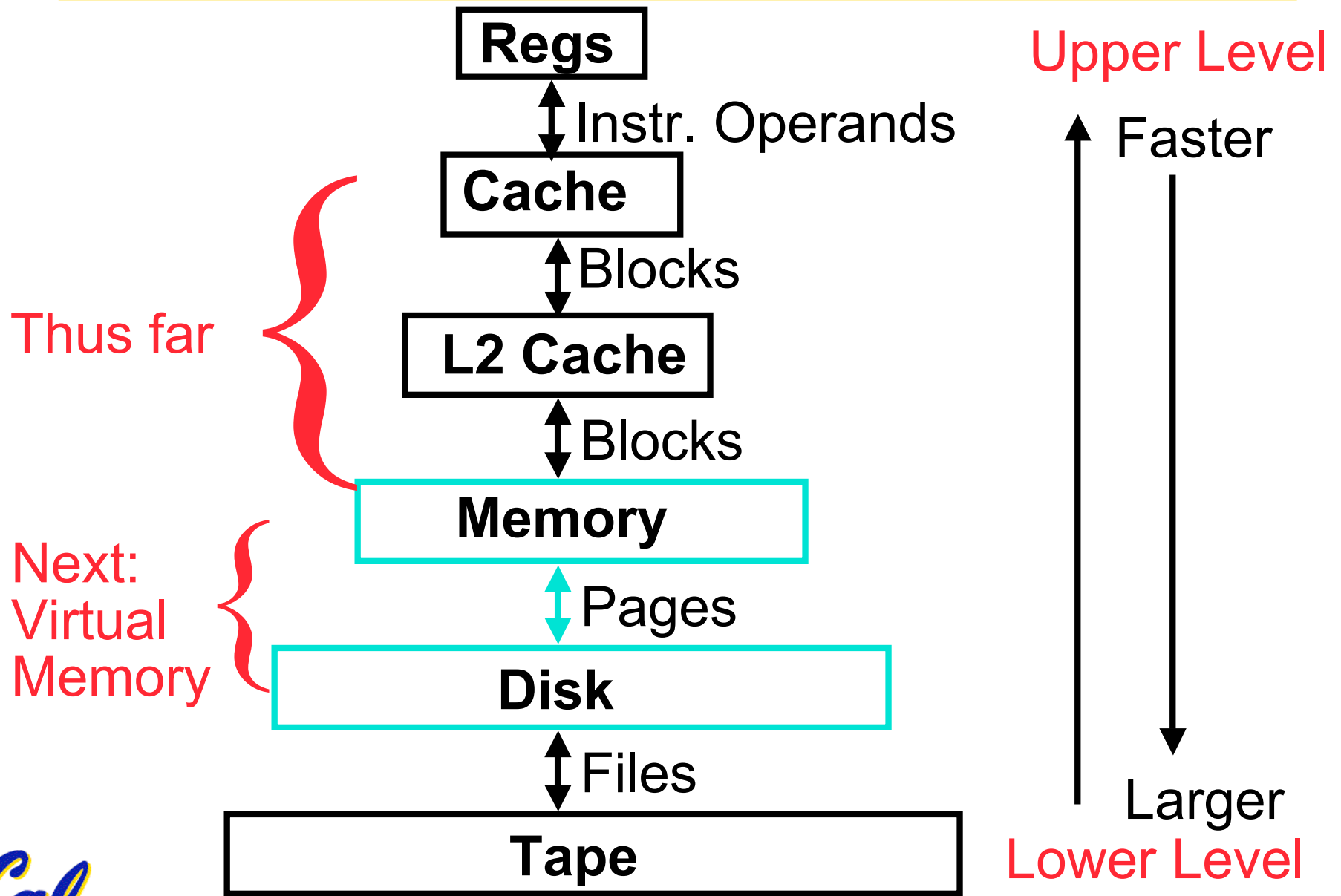
Review: Caches

- **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

Regs

Instr. Operands

Cache

Blocks

Thus far

L2 Cache

Blocks

Memory

Next:
Virtual
Memory

Pages

Disk

Files

Tape

Upper Level

Faster

Larger

Lower Level

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.
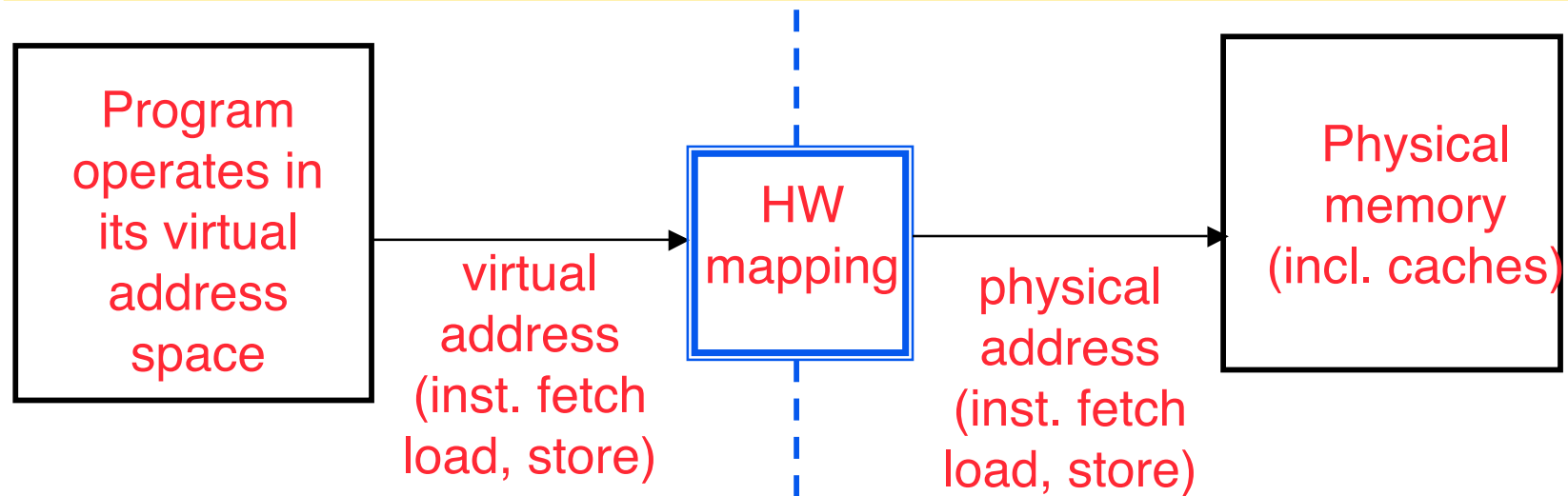
Virtual Memory

- **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

- **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**

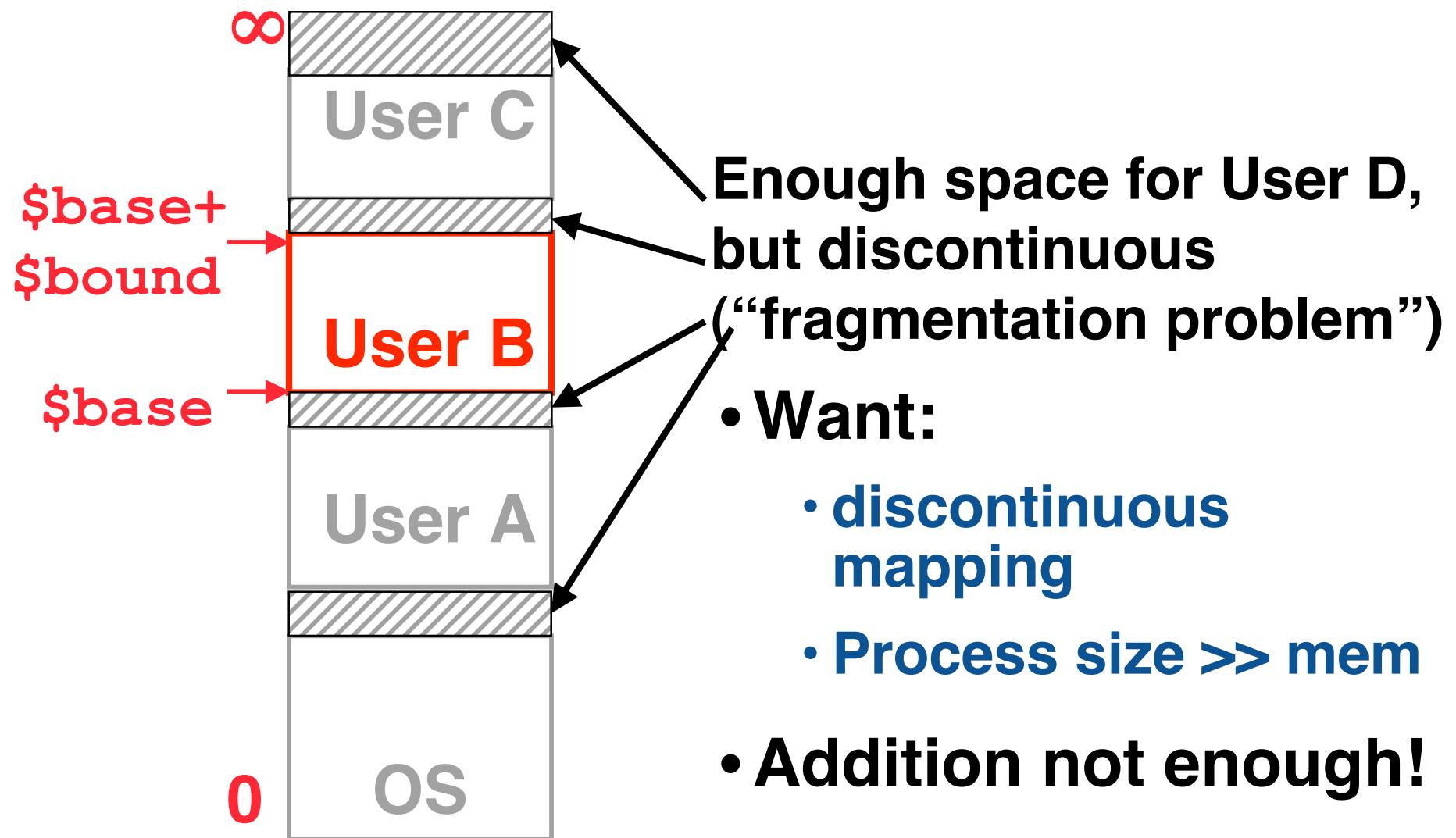# Analogy

- **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

∞

User C

$base+
$bound

User B

$base

User A

0   OS

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

- Want:
  - discontinuous mapping
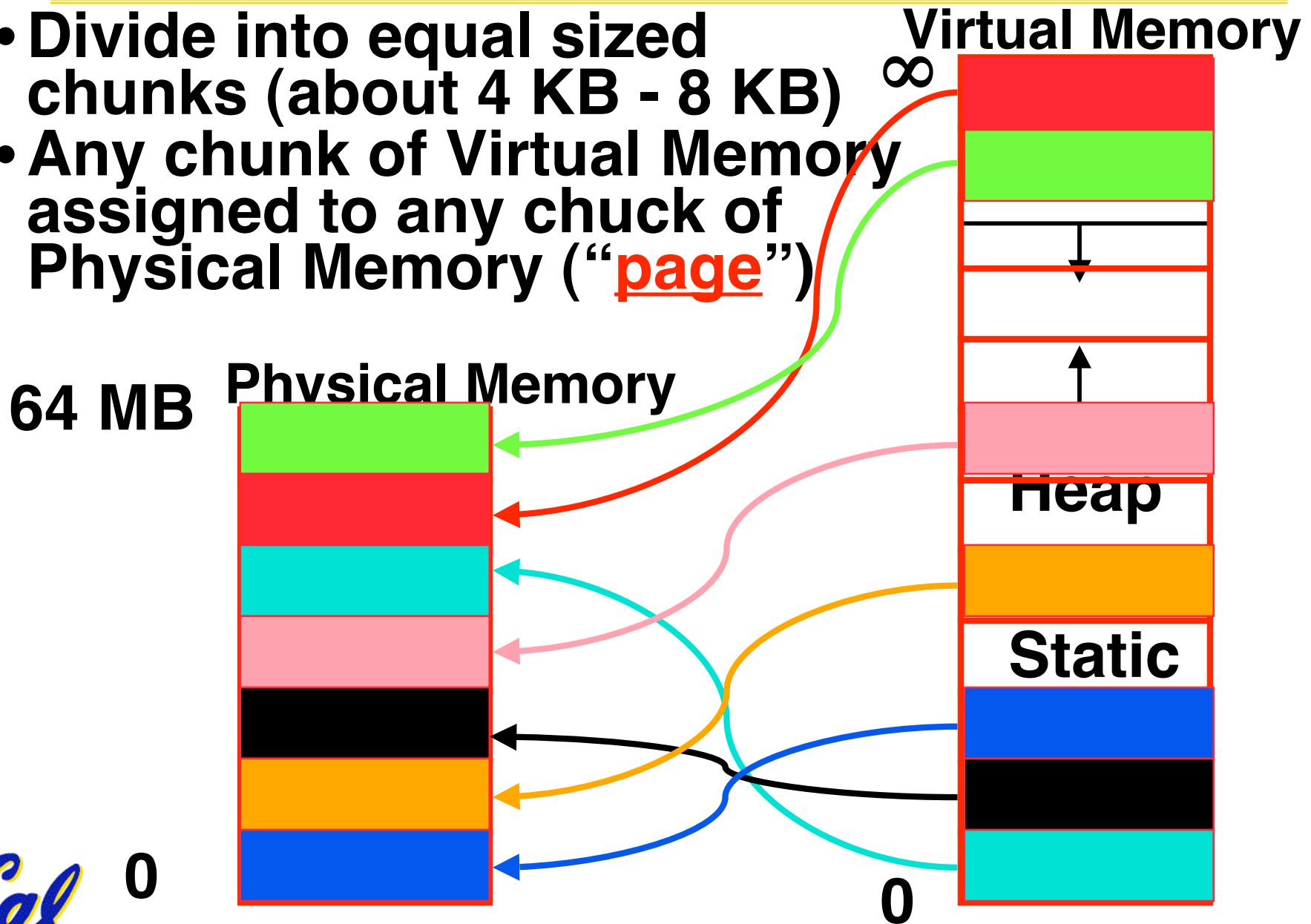 - Process size >> mem

- 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")**

Virtual Memory

∞

Physical Memory

64 MB

Heap

Static

0

0

Paging Organization (assume 1 KB pages)

Physical Address

Page is unit of mapping

Virtual Address

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

Addr Trans MAP

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

Physical Memory

Page also unit of transfer from disk to physical memory
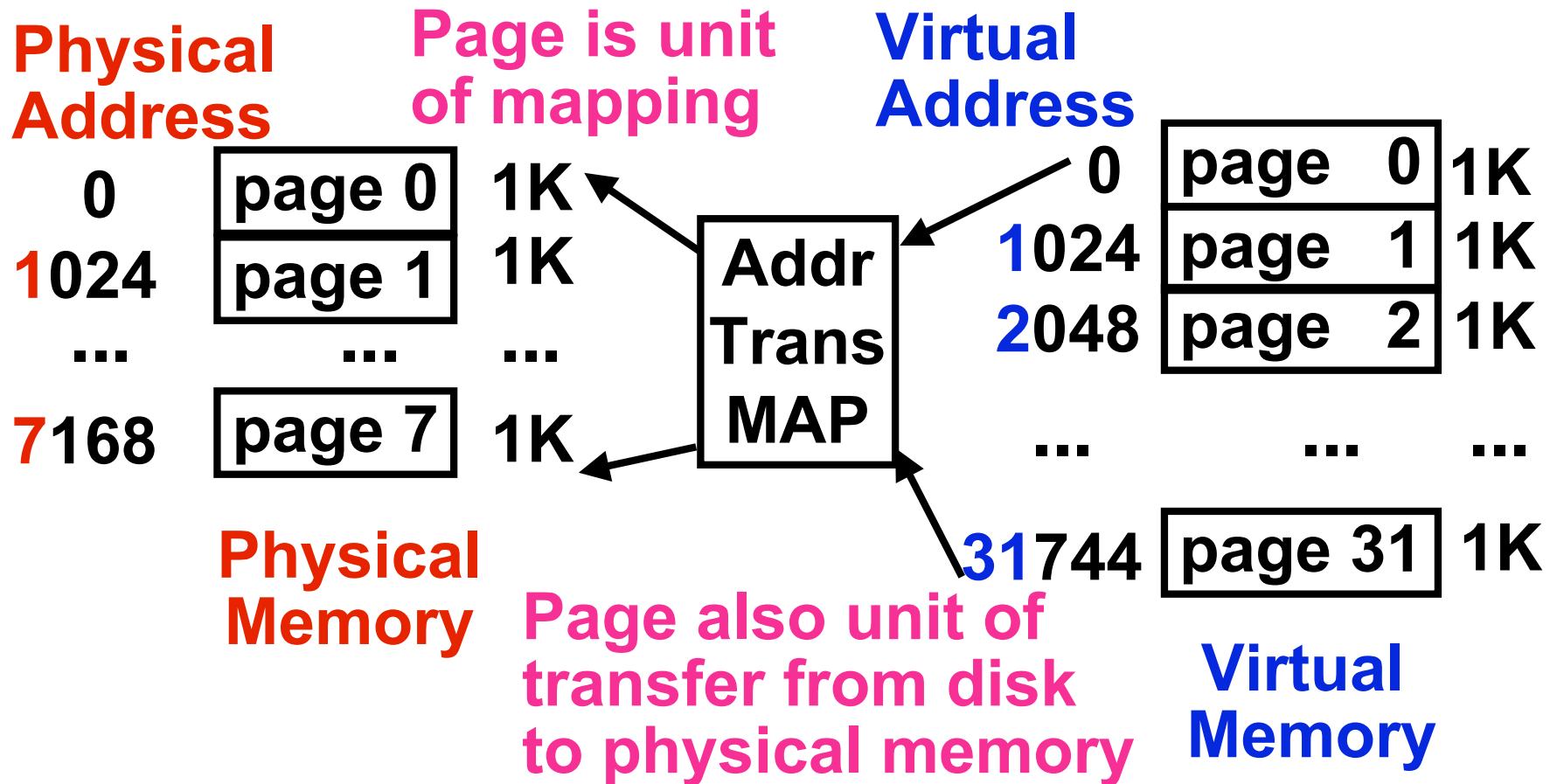
Virtual Memory

Virtual Memory Mapping Function

- **Cannot have simple function to predict arbitrary mapping**

- **Use table lookup of mappings**
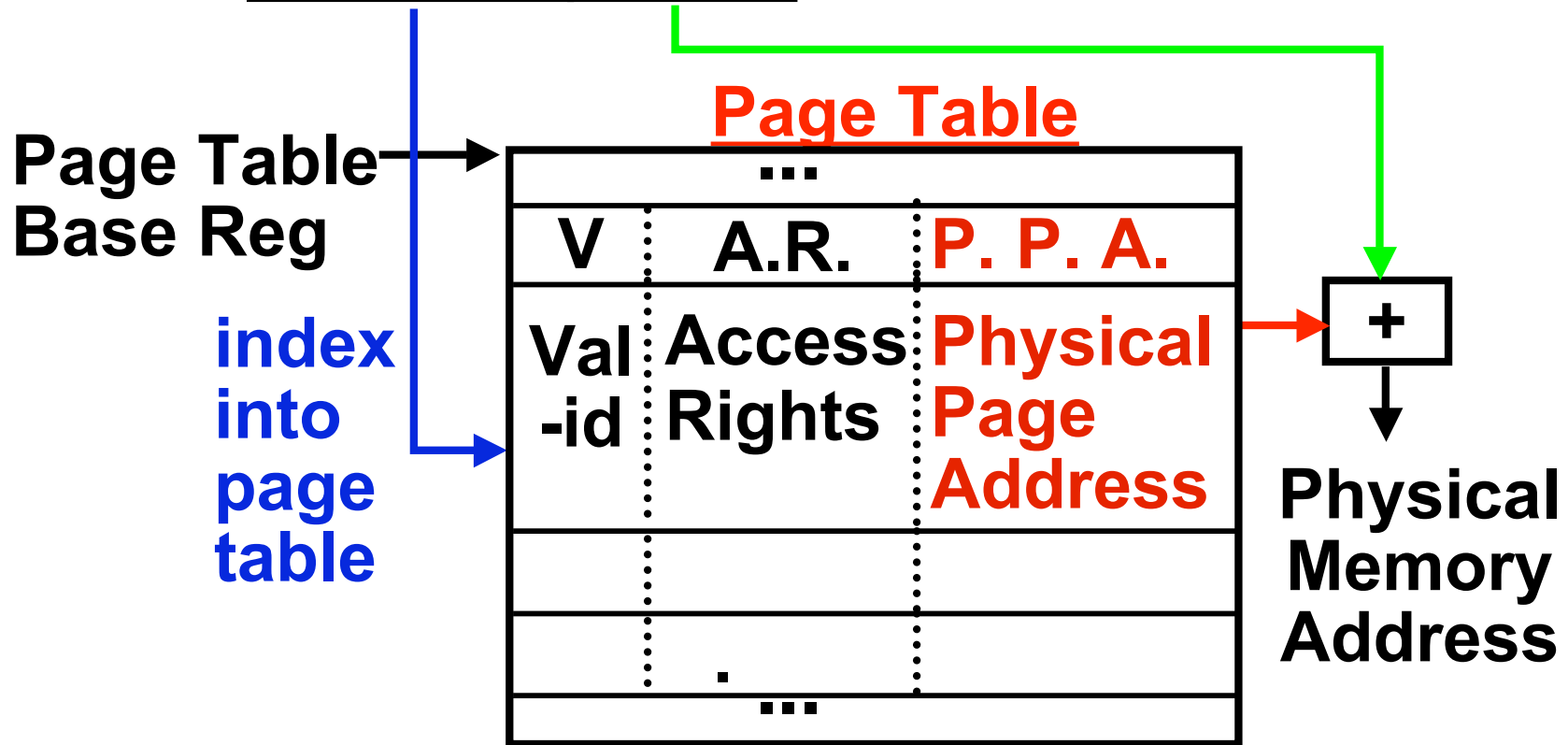
Page Number	Offset

- **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

Virtual Address:

page no.	offset

Page Table Base Reg

index into page table

Page Table

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

+

Physical Memory Address

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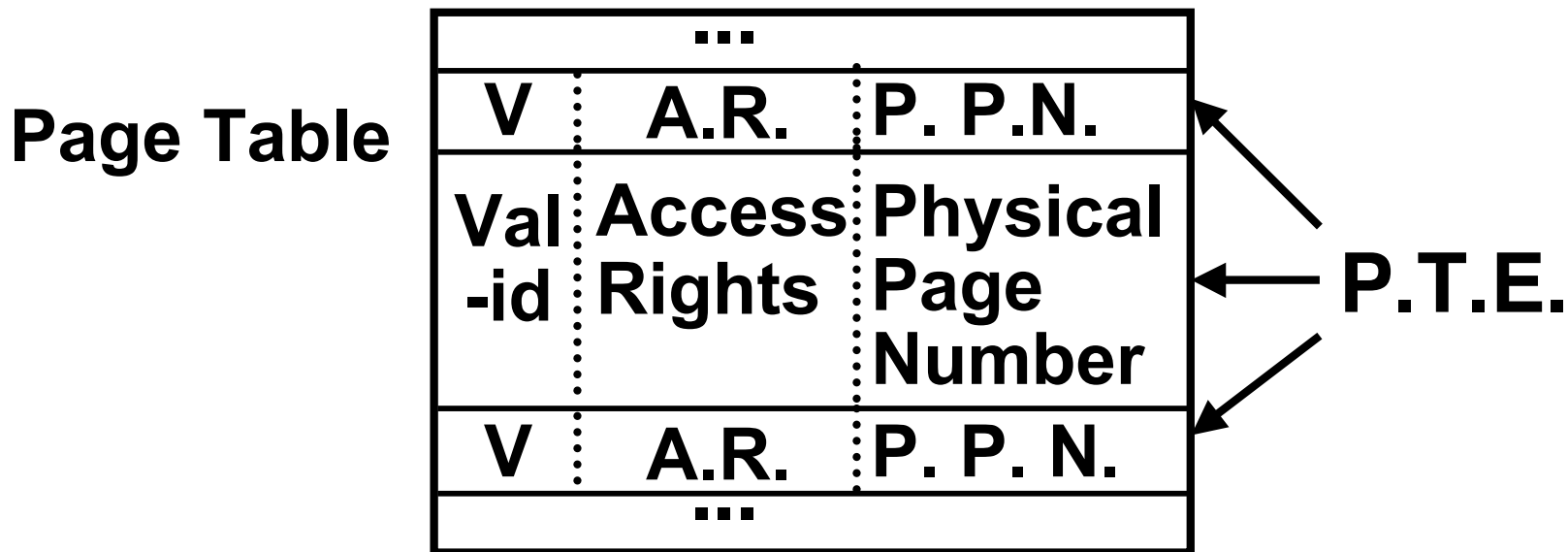
- **If valid, also check if have permission to use page: Access Rights (A.R.) may be Read Only, Read/Write, Executable**

Paging/Virtual Memory Multiple Processes

User A:
Virtual Memory

∞

Stack

Static

Code

0

Physical Memory

64 MB

0

User B:
Virtual Memory

∞

Stack

Static

Code

0

A Page Table

B Page Table
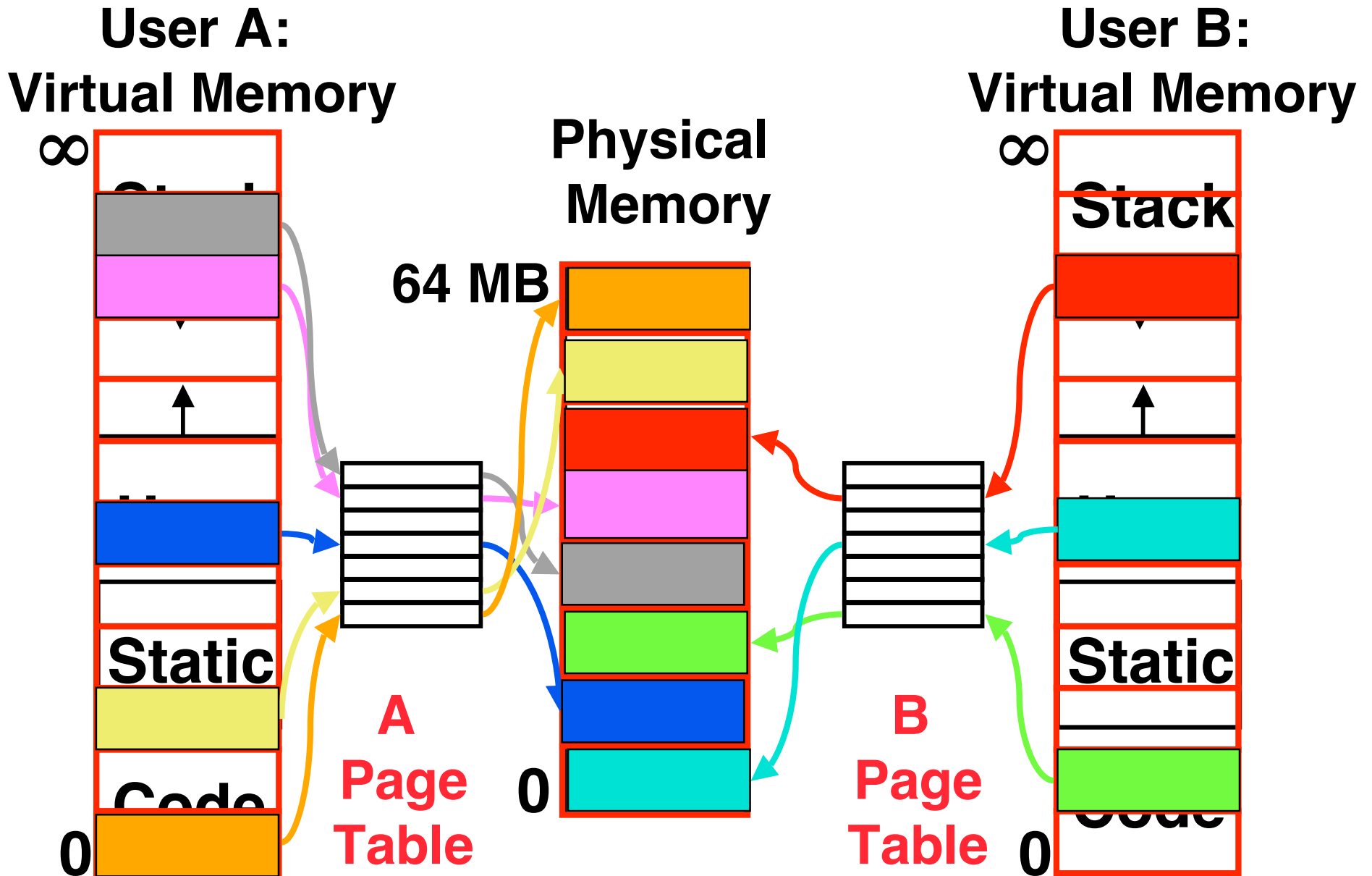
Comparing the 2 levels of hierarchy

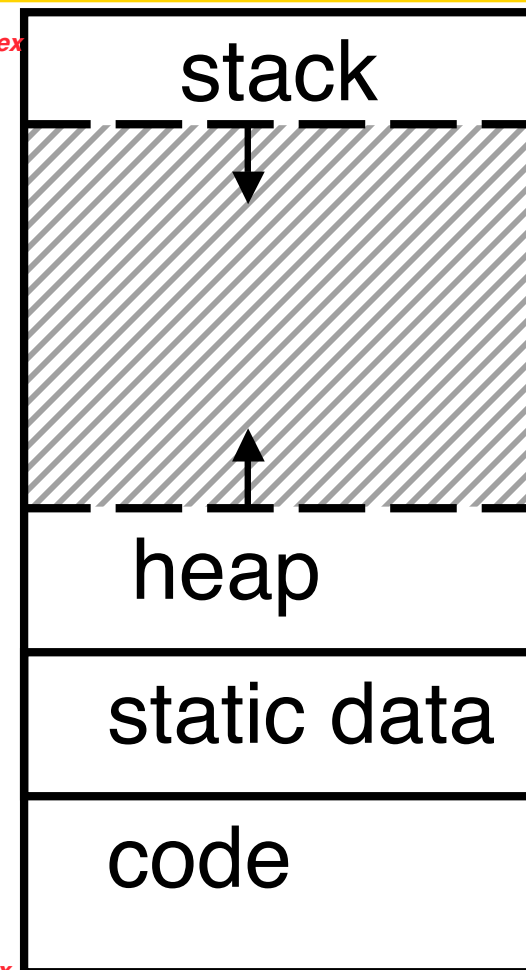| Cache version	Virtual Memory vers.
Block or Line	**Page**
Miss	**Page Fault**
Block Size: 32-64B	Page Size: 4K-8KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

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**

~ FFFF FFFF_hex

~ 0_hex

```
┌──────────────┐
│    stack     │
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌┤
│      ↓       │
│//////////////│
│      ↑       │
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌┤
│     heap     │
├──────────────┤
│  static data │
├──────────────┤
│     code     │
└──────────────┘
```

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

Administrivia

- **Proj4 due Sunday**

- **Proj3 Face-to-Face grading started yesterday**
  - **Make an appointment (and show up)**

- **Final Review Session in works**

- **Course Survey during last lecture**
  - **2 points extra added for taking survey (still anonymous)**
 - **I don't see results until long after grades due**

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 <u>virtual address translations</u> 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 <u>Translation Lookaside Buffer</u>, or <u>TLB</u>**

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
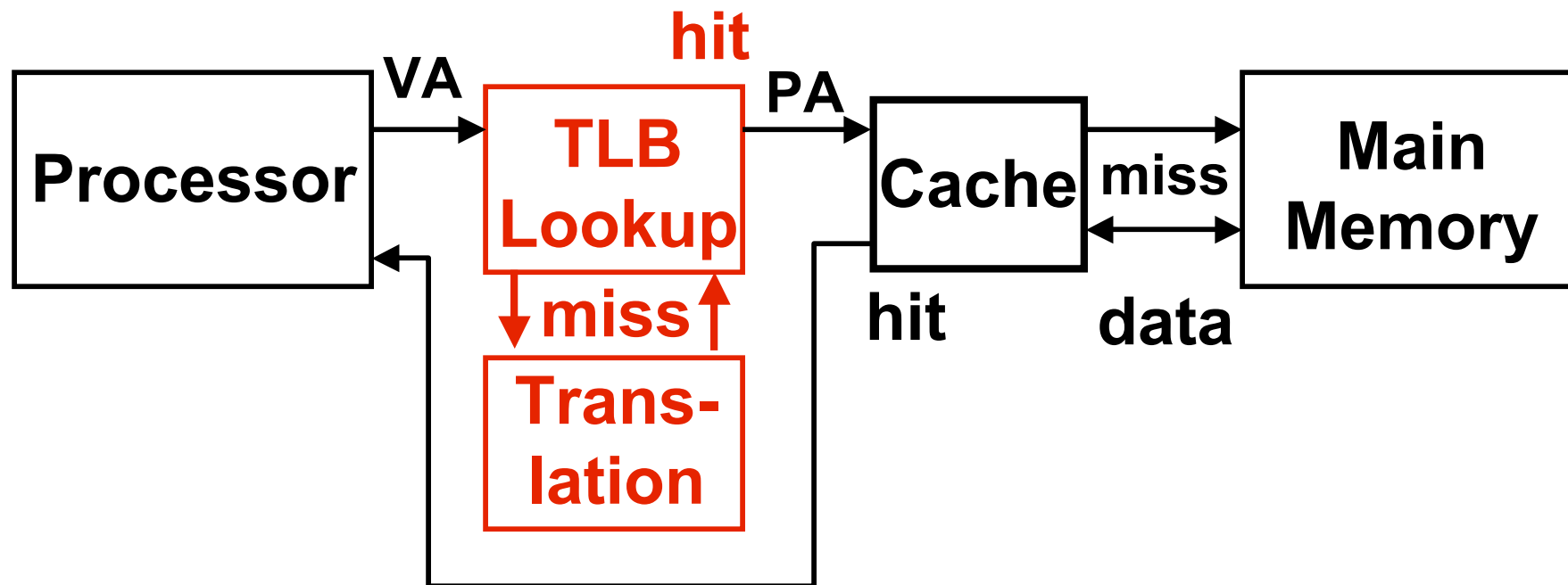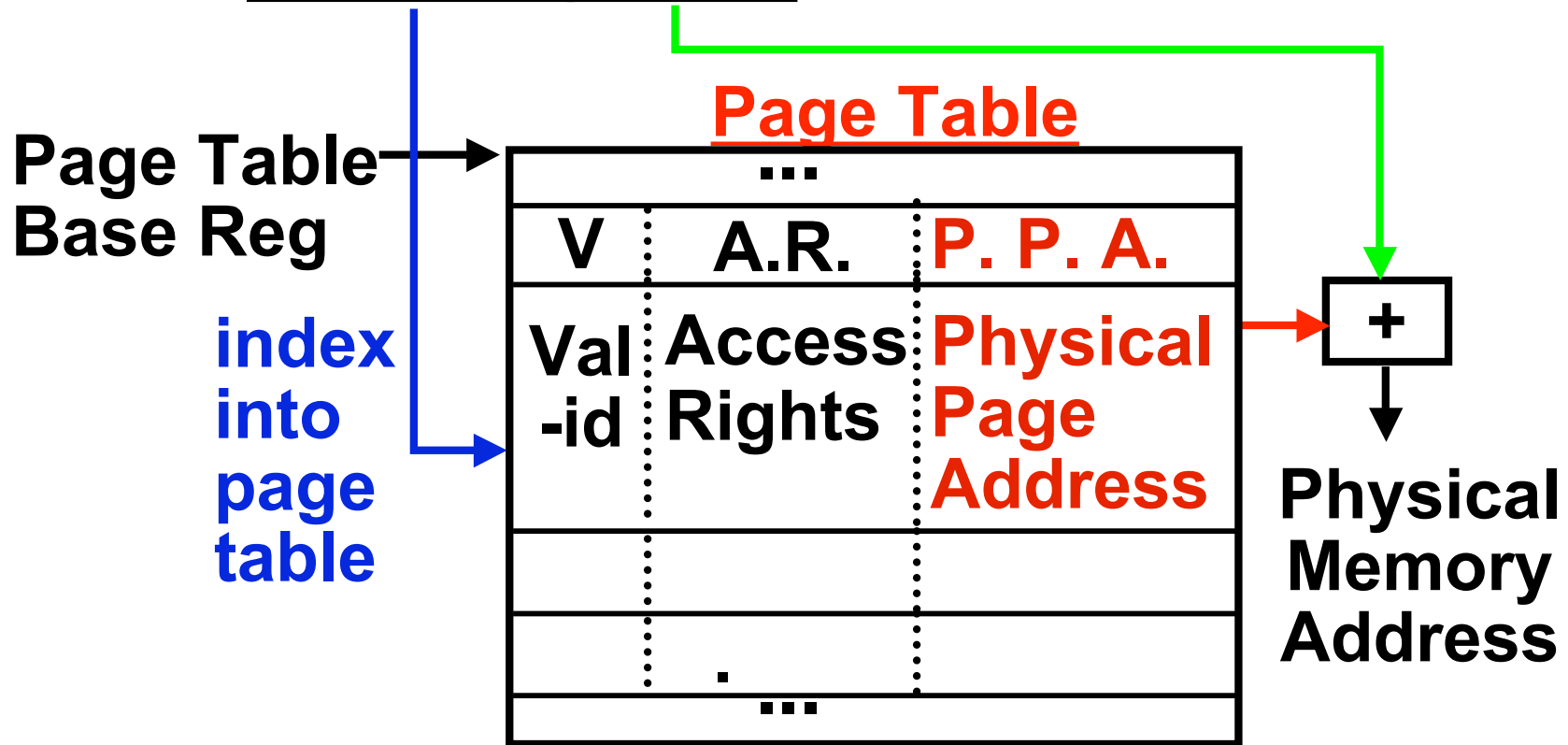

On TLB miss, get page table entry from main memory

Review Address Mapping: Page Table

Virtual Address:

page no.	offset

Page Table Base Reg

index into page table

Page Table

...		
V	**A.R.**	**P. P. A.**
Val -id	**Access Rights**	**Physical Page Address**
	.	
...		

+

Physical Memory Address

Page Table located in physical memory

Address Translation using TLB

Virtual Address

VPN

TLB Tag	INDEX	Offset

TLB

...

TLB Tag	P. P. N.
(Tag used just like in cache)	Physical Page Number
TLB Tag	P. P. N.

PPN	Offset

Physical Address

Tag	INDEX	Offset

Data Cache

Tag	Data
Tag	Data
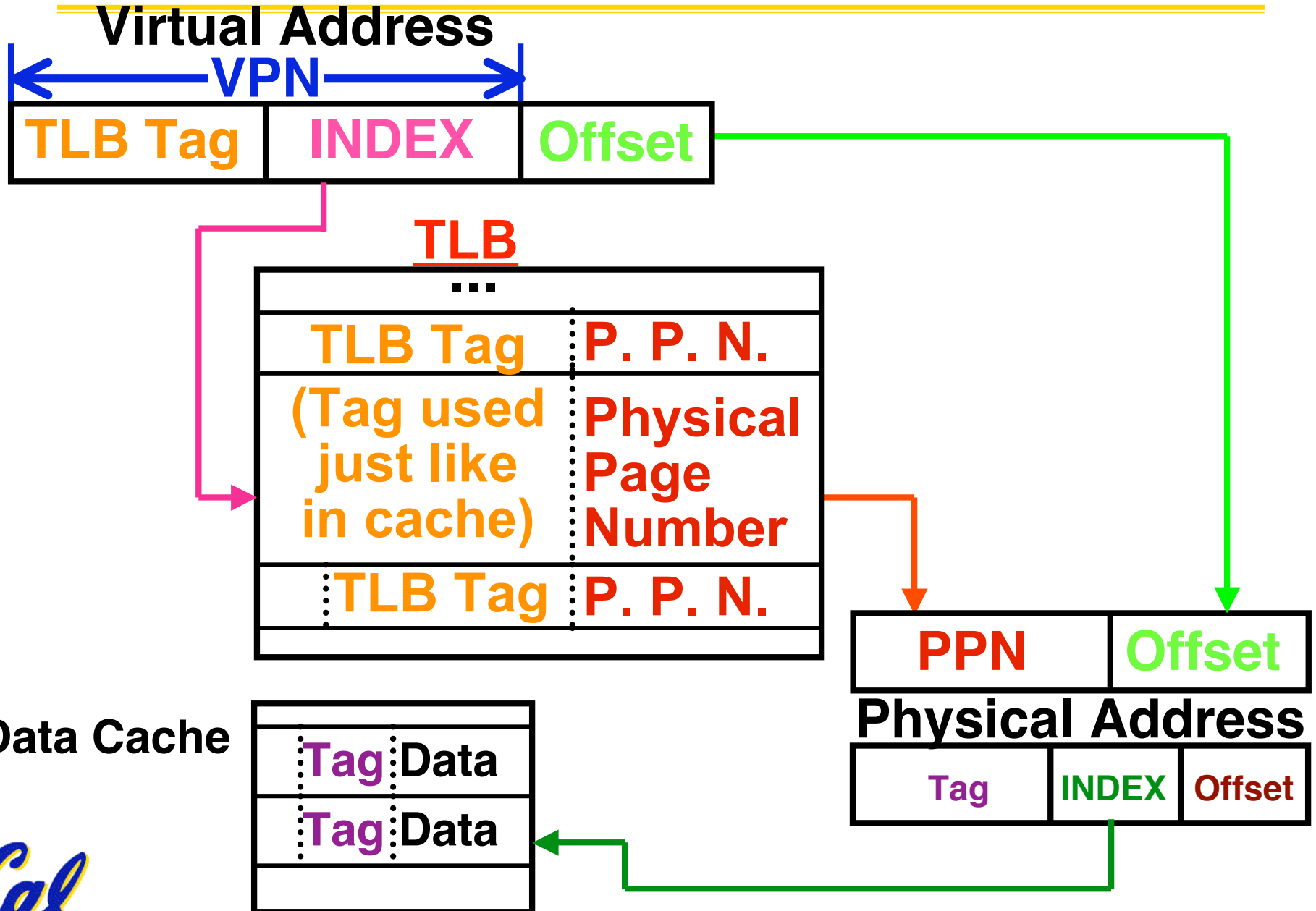
Typical TLB Format

Tag	Physical Page #	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)**
- **Dirty: since use write back, need to know whether or not to write page to disk when replaced**
- **Ref: 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**

  - **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 (Direct Memory Access – special hardware support to avoid processor)**

  - **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**

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 $\Rightarrow$ Far more "viruses" under Microsoft Windows
- Special Mode in processor ("Kernel mode") allows processor to change page table/TLB

3) Sharing:

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

Peer Instruction Answer

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

FALSE

TRUE

TRUE

A. No. Both for VM <u>and</u> cache

B. Yes. TLB SW (MIPS) or HW ($ HW, Page table OS)

C. Yes. Protection <u>and</u> a bit smaller memory

	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**