Writing in a Cache

- On a write hit, should we write:
  - In the cache only (write-back) policy
  - In the cache and main memory (or higher level cache) (write-through) policy
- On a write miss, should we
  - Allocate a block as in a read (write-allocate*)*
  - Write only in memory *(*write-around*)*

The Main Write Options

- Write-through (aka store-through)
  - On a write hit, write both in cache and in memory
  - On a write miss, the most frequent option is write-around
  - Pro: consistent view of memory (better for I/O); no ECC required for cache
  - Con: more memory traffic (can be alleviated with write buffers)
- Write-back (aka copy-back)
  - On a write hit, write only in cache (requires dirty bit)
  - On a write miss, most often write-allocate (fetch on miss) but variations are possible
  - Pro-con reverse of write through

Classifying the Cache Misses: The 3 C's

- Compulsory misses (cold start)
  - The first time you touch a line. Reduced (for a given cache capacity and associativity) by having large lines
- Capacity misses
  - The working set is too big for the ideal cache of same capacity and line size (i.e., fully associative with optimal replacement algorithm). Only remedy: bigger cache!
- Conflict misses (interference)
  - Mapping of two lines to the same location. Increasing associativity decreases this type of misses.
- There is a fourth C: coherence misses (cf. multiprocessors)

Example of Cache Hierarchies

Processor	I-Cache	D-Cache	L2
Alpha 21064	(8KB, 1, 32)	(8KB, 1, 32) WT	Off-chip (2MB,1,64)
Alpha 21164	(8KB, 1, 32)	(8KB, 1, 32) WT	(96KB,3,64) WB
Alpha 21264	(64KB,2,64)	(64KB,2,64)	0ff-chip(16MB,1,64)
Pentium	(8KB,2,32)	(8KB,2,32) WT/WB	Off-chip up to 8MB
Pentium II	(16KB,2,32)	(16KB,2,32) WB	“glued”(512KB,8,32) WB
Pentium III	(16KB,2,32)	(16KB,4,32) WB	(512KB,8,32) WB
Pentium 4	Trace cache 96KB	(16KB,4,64) WT	(1MB, 8,128) WB

Cache Performance

- CPI $_{\text{contributed by cache}}$ = CPI$_c$

  = miss rate * number of cycles to handle the miss
- Another important metric

  Average memory access time = cache hit time * hit rate

  + Miss penalty * (1 - hit rate)

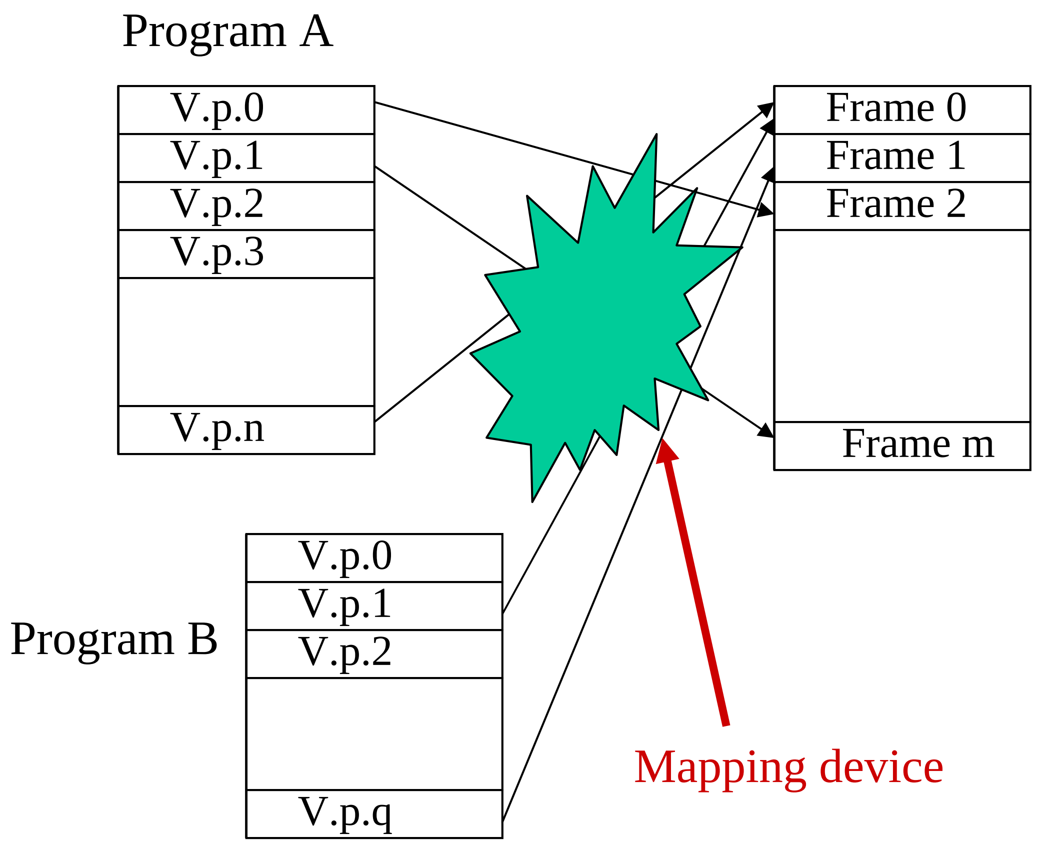
Improving Cache Performance

- To improve cache performance:
  - Decrease miss rate without increasing time to handle the miss (more precisely: without increasing average memory access time)
  - Decrease time to handle the miss w/o increasing miss rate
- A slew of techniques: hardware and/or software
  - Increase capacity, associativity etc.
  - Hardware assists (victim caches, write buffers etc.)
  - Tolerating memory latency: Prefetching (hardware and software), lock-up free caches
  - O.S. interaction: mapping of virtual pages to decrease cache conflicts
  - Compiler interactions: code and data placement; tiling

Improving L1 Cache Access Time

- Processor generates virtual addresses
- Can cache have virtual address tags?
  - What happens on a context switch?
- Can cache and TLB be accessed in parallel?
  - Need correspondence between page size and cache size + associativity
- What about virtually addressed physically tagged caches?

A Brief Review of Paging



Physical memory

Note: In general n, q >> m

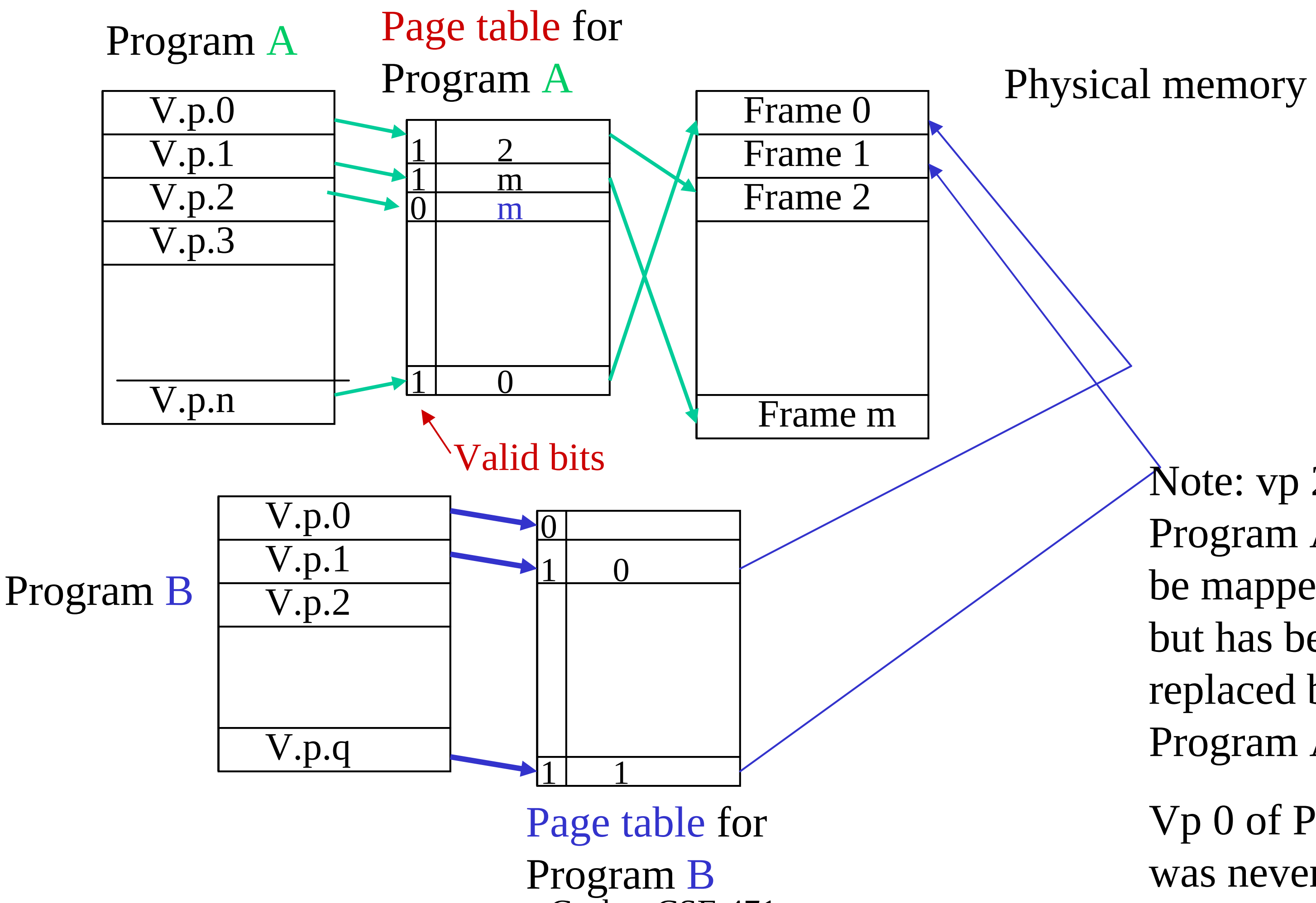
Not all virtual pages of a program are mapped at a given time

In this example, programs A and B share frame 0 but with different virtual page numbers

Mapping Device: Page Tables

- Page tables contain page table entries (PTE):
  - Virtual page number (implicit/explicit), physical page number,valid, protection, dirty, use bits (for LRU-like replacement), etc.
- Hardware register points to the page table of the running process
- Earlier system: contiguous (in virtual space) page tables; Now, multi-level page tables
- In some systems, inverted page tables (with a hash table)
- In all modern systems, page table entries are cached in a TLB

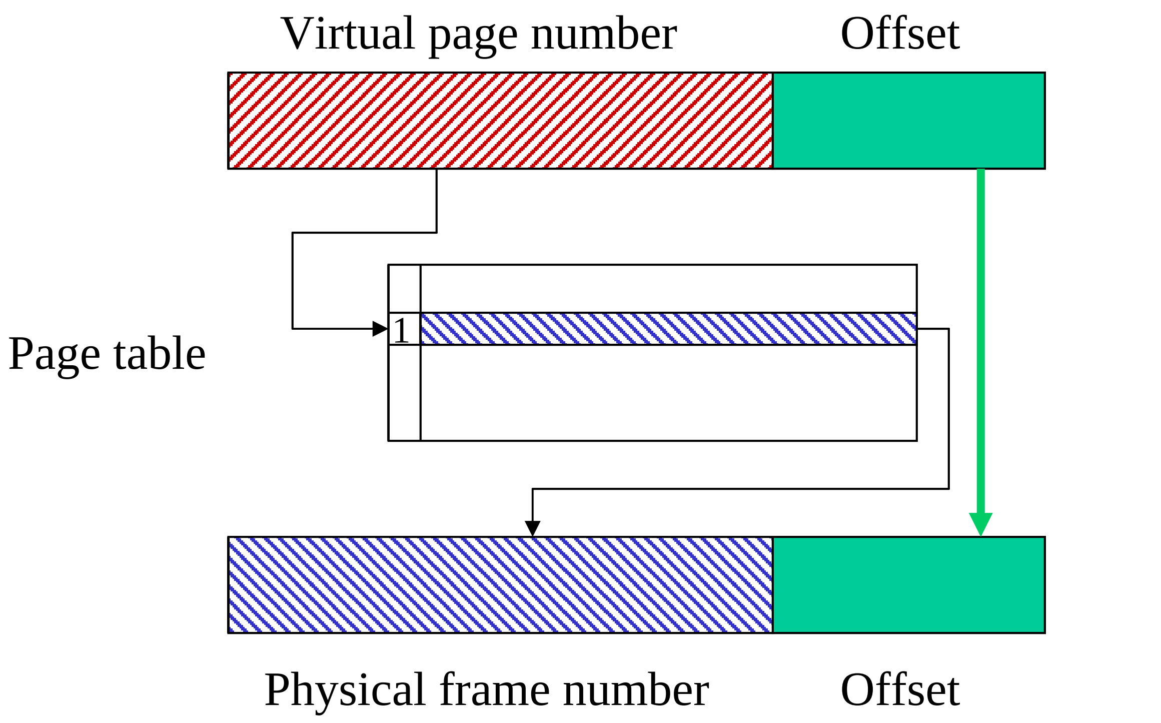
Illustration of Page Table



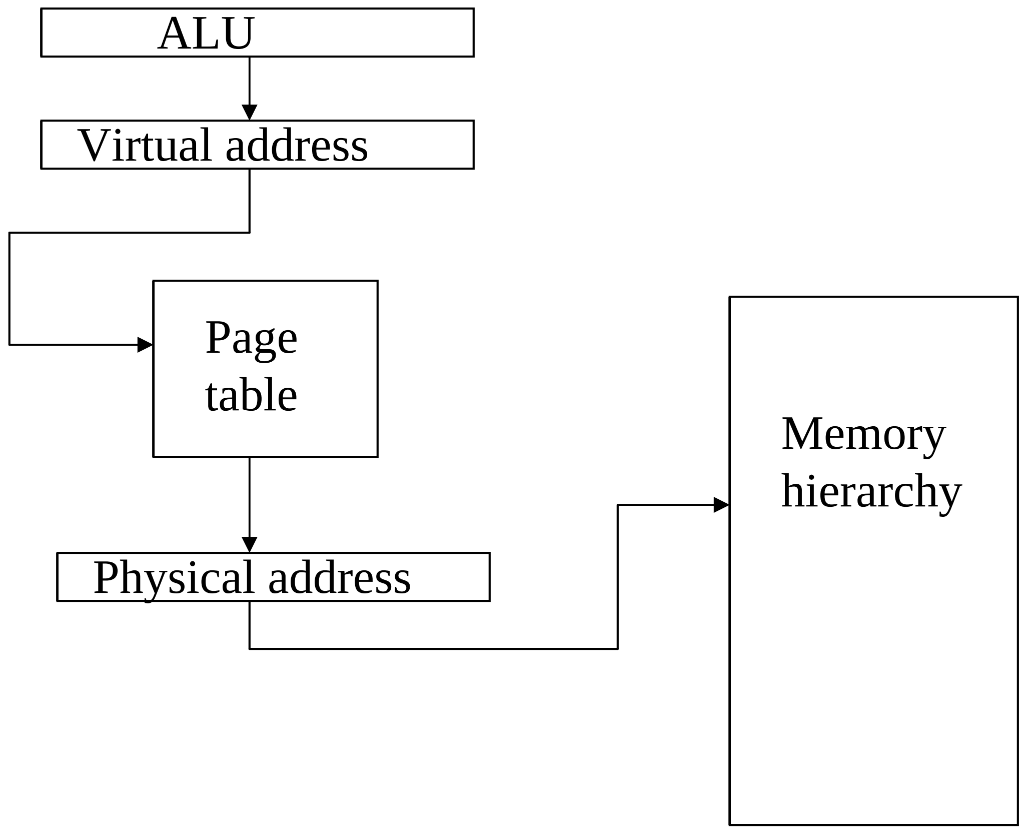
Note: vp 2 of Program A used to be mapped to pp m but has been replaced by vp 1 of Program A;

Vp 0 of Program B was never mapped

Virtual Address Translation



From Virtual Address to Memory Location (highly abstracted)



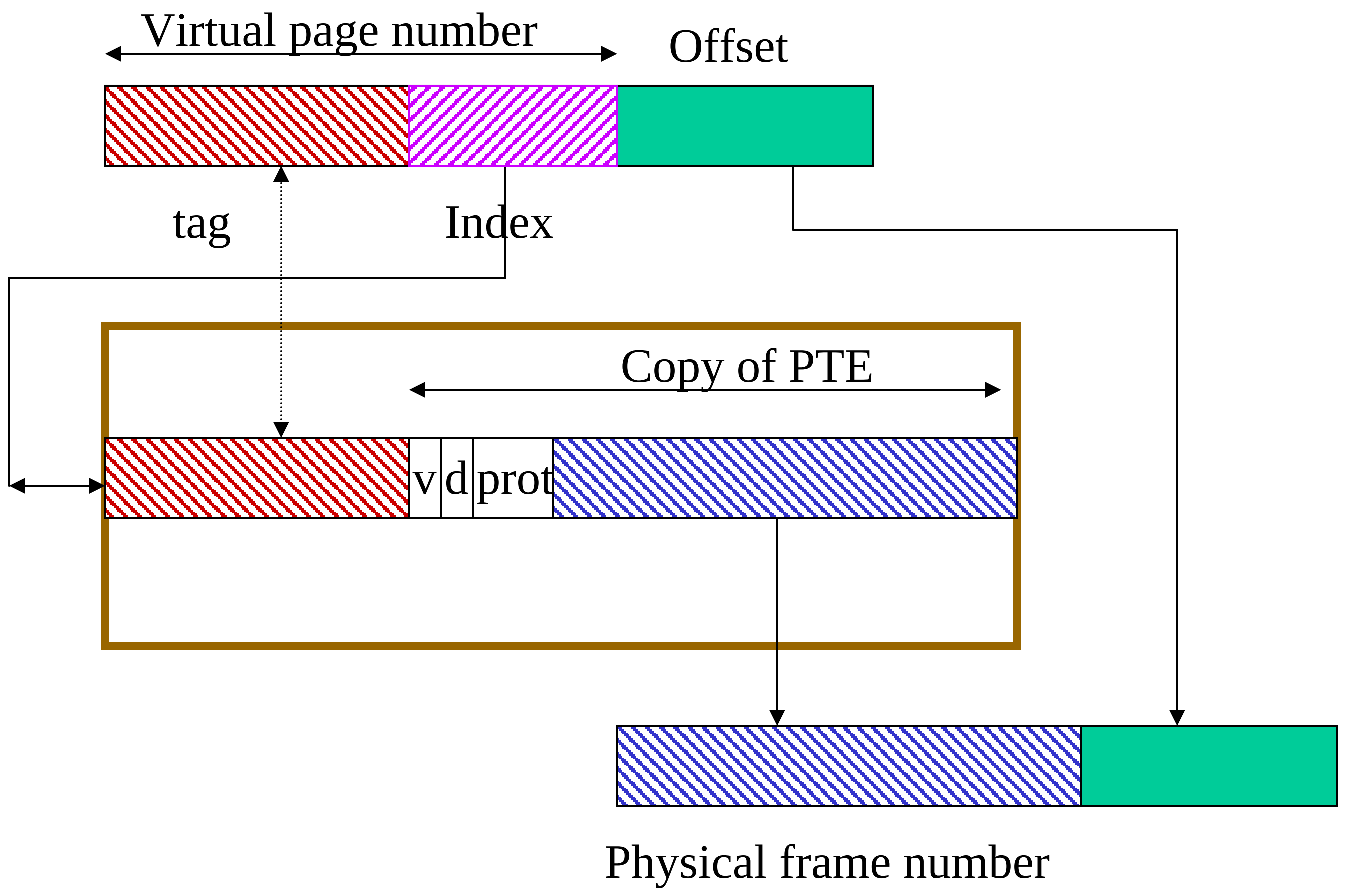
Translation Look-aside Buffers (TLB)

- Keeping page tables in memory defeats the purpose of caches
  - Needs one memory reference to do the translation
- Hence, introduction of caches to cache page table entries; these are the TLB's
  - There have been attempts to use the cache itself instead of a TLB but it has been proven not to be worthwhile
- Nowadays, TLB for instructions and TLB for data
  - Some part of the TLB's reserved for the system
  - Of the order of 128 entries, quite associative

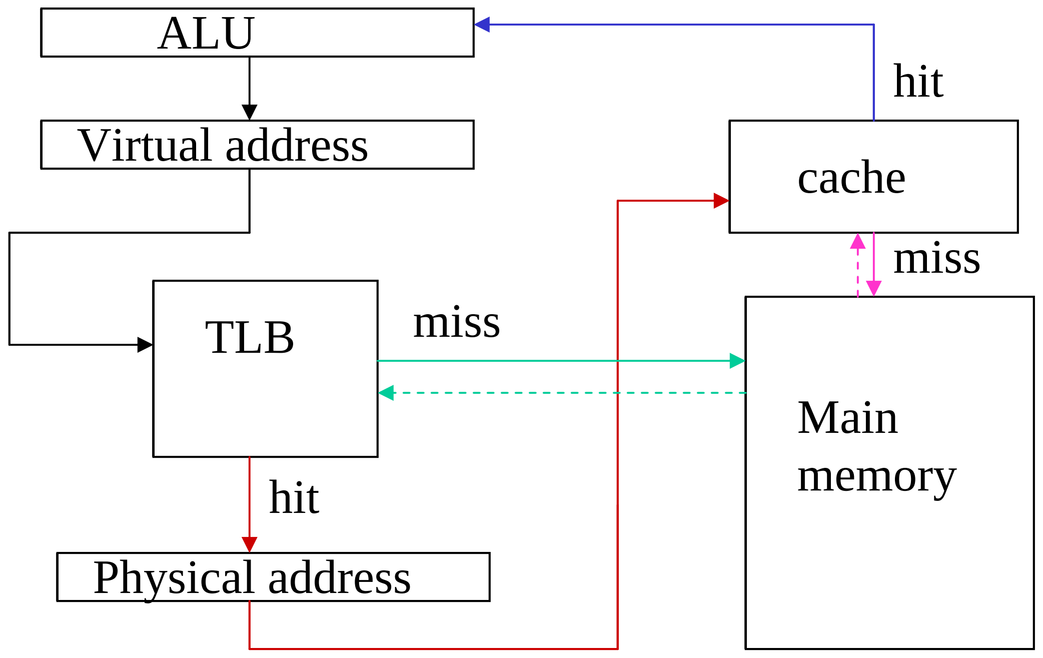
TLB’s

- TLB miss handled by hardware or by software (e.g., PAL code in Alpha) or by a combination HW/SW
  - TLB miss 10’s-100’s cycles -> no context-switch
- Addressed in parallel with access to the cache
- Since smaller, goes faster
  - It’s on the critical path
- For a given TLB size (number of entries)
  - Larger page size -> larger mapping range

TLB organization



From Virtual Address to Memory Location (highly abstracted; revisited)


ALU
Virtual address
TLB
miss
hit
Physical address
cache
hit
miss
Main memory