Evolution in Memory Management Techniques

- In early days, single program run on the whole machine
  - Used all the memory available
- Even so, there was often not enough memory to hold data and program for the entire run
  - Use of overlays, i.e., static partitioning of program and data so that parts that were not needed at the same time could share the same memory addresses
- Soon, it was noticed that I/O was much more time consuming than processing, hence the advent of multiprogramming

Multiprogramming

- Multiprogramming
  - Several programs are resident in main memory at the same time
  - When one program executes and needs I/O, it relinquishes CPU to another program
  - Relinquishing CPU means a context-switch (in contrast with multithreading where several contexts are provided in the hardware)
- Some important questions from the memory management viewpoint:
  - How does one program ask for (more) memory or free memory
  - How is one program protected from another

Virtual Memory: Basic idea

- Idea first proposed and implemented at the University of Manchester in the early 60's.
- Basic idea is to compile/link a program in a virtual space as large as the addressing space permits
- Then, divide the virtual space in "chunks" and bring those "chunks' on demand in physical memory
- Provide a general (fully-associative) mapping between virtual "chunks" and physical "chunks"

Virtual Memory Implementations

- When the virtual space is divided into chunks of the same size, called pages, we have a paging system
- If chunks are of different sizes, we have segments
  - Segments can correspond to semantic objects (a good thing) but implementation is more difficult (memory allocation of variable size segments; checks for out of bounds etc.)
- Paging (segmented) systems predate caches
  - But same questions (mapping, replacement, writing policy)
- An enormous difference: penalty for a miss
- Requires hardware assists for translation and protection

Paging

- Allows virtual address space larger than physical memory
- Allows sharing of physical memory between programs (multiprogramming) without much fragmentation
  - Physical memory allocated to a program does not need to be contiguous; only an integer number of pages
- Allows sharing of pages between programs (not always simple)

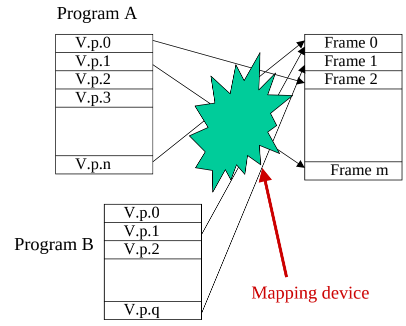
Two Extremes in the Memory Hierarchy

PARAMETER	L1	PAGING SYSTEM
block (page) size	16-64 bytes	4K-8K (also 64K)
miss (fault) time	5-20 cycles (10-50 ns)	Millions of cycles (3-20 ms)
miss (fault) rate	1-10%	0.00001-0.001%
memory size	16K-64K Bytes (impl. depend.)	Gigabytes (depends on ISA)

Other Extreme Differences

- Mapping: Restricted (L1) vs. general (Paging)
  - Hardware assist for virtual address translation (TLB)
- Miss handler
  - Hardware only for caches
  - Software only for paging system (context-switch)
  - Hardware and/or software for TLB
- Replacement algorithm
  - Not important for L1 caches
  - Very important for paging system
- Write policy
  - Always write back for paging systems

Illustration of Paging



Note: In general n, q >> m

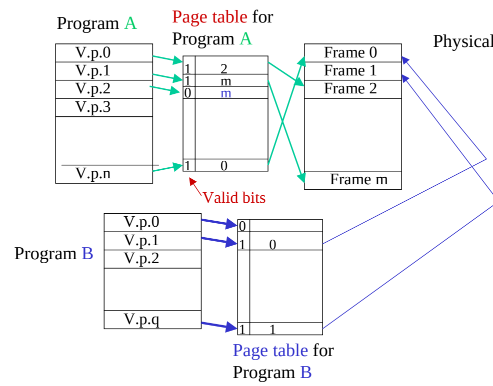
Not all virtual pages of a program are mapped at a given time

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

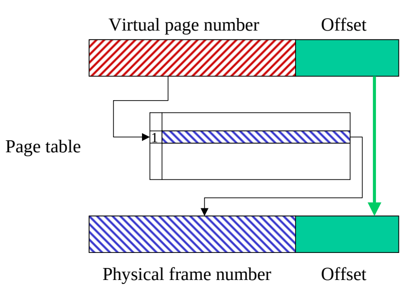
Illustration of Page Table



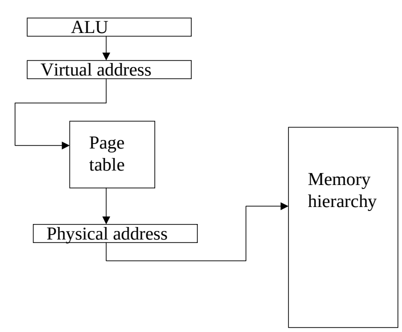
Note: vp 2 of Program A used to be mapped to pp m but has been replaced by vp 1 of Program A;

Vp 0 of Program B was never mapped

Virtual Address Translation



From Virtual Address to Memory Location (highly abstracted)



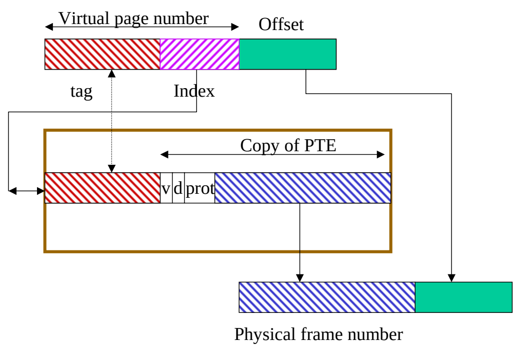
Translation Look-aside Buffers (TLB)

- 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

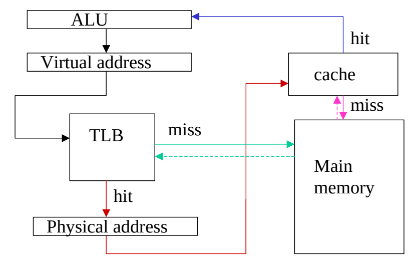
TLB's

- TLB miss handled by hardware or by software (e.g., PAL code in Alpha) or by a combination HW/SW
  - TLB miss 10-100 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)



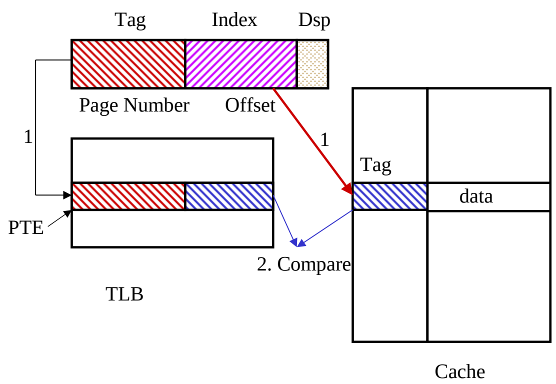
Address Translation

- At each memory reference the hardware searches the TLB for the translation
  - TLB hit and valid PTE the physical address is passed to the cache
  - TLB miss, either hardware or software (depends on implementation) searches page table in memory.
    - If PTE is valid, contents of the PTE loaded in the TLB and back to step above
- In hardware the TLB miss takes a few cycles
- In software takes up to 100 cycles
- In either case, no context-switch
- If PTE is invalid, we have a page fault (even on a TLB hit)

Speeding up L1 Access

- Cache can be (speculatively) accessed in parallel with TLB if its indexing bits are not changed by the virtual-physical translation
- Cache access (for reads) is pipelined:
  - Cycle 1: Access to TLB and access to L1 cache (read data at given index)
  - Cycle 2: Compare tags and if hit, send data to register

Virtually Addressed Cache



"Virtual" Caches

- Previous slide: Virtually addressed, physically tagged
  – Can be done for small L1, i.e., capacity < (page * ass.)
  – Can be done for larger caches if O.S. does a form of page coloring such that "index" is the same for synonyms (see below)
  – Can also be done more generally (complicated but can be elegant)
- Virtually addressed, virtually tagged caches
  – Synonym problem (2 virtual addresses corresponding to the same physical address). Inconsistency since the same physical location can be mapped into two different cache blocks
  – Can be handled by software (disallow it) or by hardware (with "pointers")
  – Use of PID's to only partially flush the cache