18-447 Lecture 20: Virtual Memory, Protection and Paging

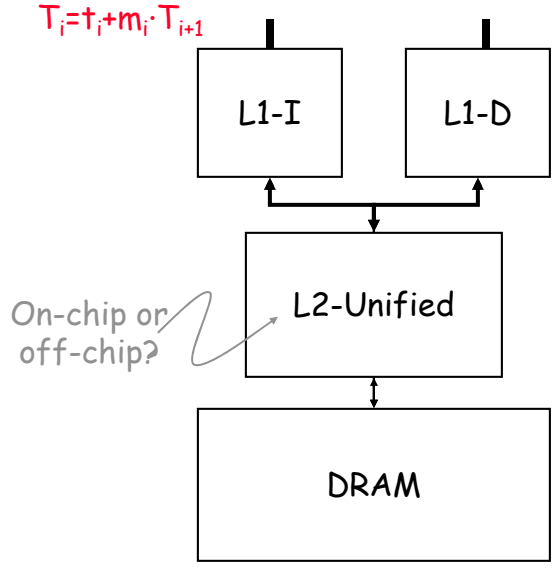
James C. Hoe
Dept of ECE, CMU
April 8, 2009

Announcements: Best class ever, next Monday

Handouts: H14 HW#4 (on Blackboard), due 4/22/09
Assigned Reading
"Virtual memory in contemporary microprocessors."
B. L Jacob and T. N. Mudge. IEEE Micro, July/August 1998.
(on Blackboard)

Multi-Level Caches

$T_i = t_i + m_i \cdot T_{i+1}$



- a few pclk latency
- many GB/sec (random access)

On-chip or off-chip?

intermediate hierarchy is a cheaper way to reduce T_1 then using a faster or bigger L1,

- hundreds of pclk latency
- ~GB/sec (sequential)

Remember, memory hierarchy is also about memory bandwidth

Multi-Level Cache Design

- Upper Hierarchies
  - small C: upper-bound by SRAM access time
  - smallish B: upper-bound by C/B effects and the benefits of fine-grain spatial locality
  - a: required to counter C/B effects
- Lower Hierarchies
  - large C: upper-bound by chip area (or how much you are willing to pay off-chip)
  - large B: to reduce tag storage overhead and to take advantage of coarse-grain spatial locality
  - a: upper bound by complexity (off-chip implementations)

    Very large off-chip caches are either direct-mapped or use on-chip tag-RAM and hit-logic

    Newer on-chip L2s can be highly associative

Inclusion Principle

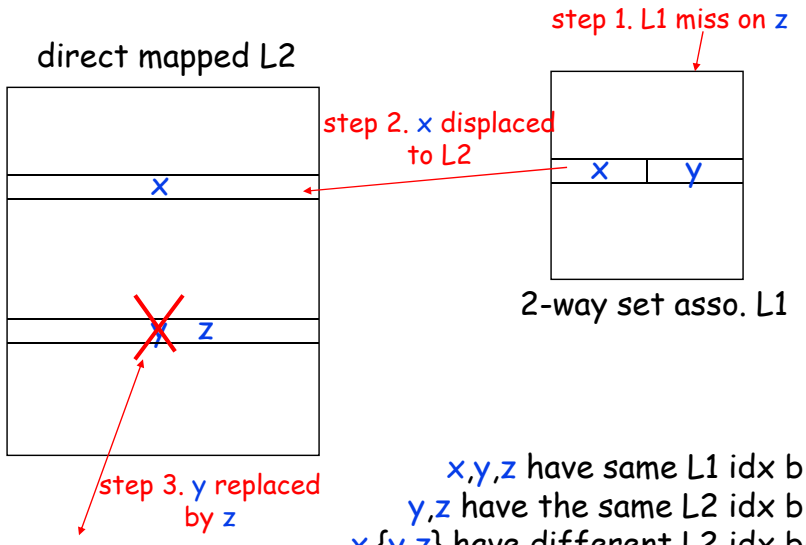
- Traditionally, L_i contents is always a subset of L_{i+1}
  - if a memory location is important enough to be in L_i, it must be important enough to be in L_{i+1}
  - external agents (I/O, other processors) only have to check the lowest level to know if a memory location is cached---do not need to consume L1 bandwidth
- Nontrivial to maintain when L_{i+1} has lower associativity
- E.g. a single L_i miss may trigger multiple L_i evictions
  - suppose L_i has a>1, and L_{i+1} has a=1
  - suppose x, y, z have same L_i index
  - suppose y, z have the same L_{i+1} index and different from x's
  - suppose initially x and y are cached in L_i (and hence L_{i+1})
  - suppose a miss to z evicts x from L_i according to LRU

    z must evict y from L_{i+1} due to collision

    y must also be evicted from L_i to maintain inclusion

  New multicores tend not to maintain inclusion anymore, why?

Possible Inclusion Violation



x,y,z have same L1 idx bits
y,z have the same L2 idx bits
x,{y,z} have different L2 idx bits

2 Parts to Modern VM

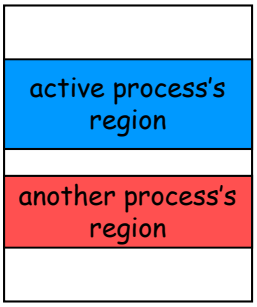
- In a multi-tasking system, VM provides each process with the illusion of a large, private, and uniform memory
- Ingredient A: Naming and Protection
  - each process sees a large, contiguous memory segment without holes ("virtual" address space is much bigger than the available physical storage)
  - each process's memory space is private, i.e. protected from access by other processes
- Ingredient B: demand paging
  - capacity of secondary storage (swap space on disk)
  - at the speed of primary storage (DRAM)

Mechanism: Address Translation

- Protection, VM and demand paging enabled by a common HW mechanism: address translation
- User process operates on "effective" addresses
- HW translates from EA to PA on every memory reference
  - controls which physical locations (DRAM and/or swap disk) can be named by a process
  - allows dynamic relocation of physical backing store (DRAM vs. swap disk)
- Address translation HW and policies controlled by the OS and protected from users, i.e., priviledged

Evolution of Memory Protection

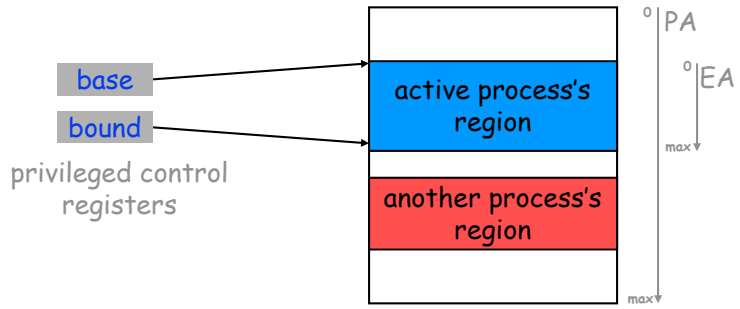
- Earliest machines did not have the concept of protection and address translation
  - no need---single process, single user
    automatically "private and uniform" (but not very large)
  - programs operated on physical addresses directly
    cannot support multitasking protection
- Multitasking 101
  - give each process a non-overlapping, contiguous physical memory region
  - everything belonging to a process must fit in that region
  - how do you keep one process from reading or trashing another process' data?



Base and Bound

- A process's private memory region can be defined by
  - **base**: starting address of the region
  - **bound**: ending address of the region
- User process issue "effective" address (EA) between 0 and the size of its allocated region

private and uniform



Base and Bound Registers

- When switching user processes, OS sets base and bound registers
- Translation and protection check in hardware on every user memory reference
  - PA = EA + **base**
  - if (PA < **bound**) then okay else violation
- User processes cannot be allowed to modify the base and bound registers themselves

  ⇒ requires 2 privilege levels such that the OS can see and modify certain states that the user cannot

  privileged instructions and state

Segmented Address Space

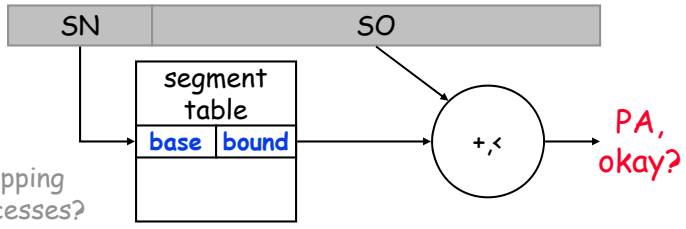
- Limitations of the base and bound scheme
  - large contiguous space is hard to come by after the system runs for a while---free space may be fragmented
  - how do two processes shared some memory regions but not others?
- A "base&bound" pair is the unit of protection

  ⇒ give user multiple memory "segments"
  - each segment is a continuous region
  - each segment is defined by a base and bound pair
- Earliest use, separate code and data segments
  - 2 sets of base-and-bound reg's for inst and data fetch
  - allow processes to share read-only code segments

  became more elaborate later: code, data, stack, etc

Segmented Address Translation

- EA comprise a segment number (SN) and a segment offset (SO)
  - SN may be specified explicitly or implied (code vs. data)
  - segment size limited by the range of SO
  - segments can have different sizes, not all SOs are meaningful
- Segment translation table
  - maps SN to corresponding base and bound
  - separate mapping for each process
  - must be a privileged structure



how to change mapping when swapping processes?

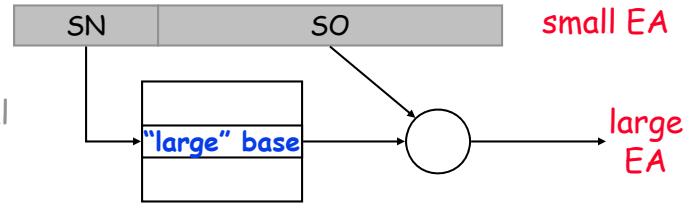
Access Protections

- In addition to naming, finer-grain access protection can be associated with each segment as extra bits in the segment table
- Generic options include
  - readable?
  - writeable?
  - executable?

  also misc. options such as cacheable? which level?
- Normal data pages ⇒ RW(!E)
- Static shared data pages ⇒ R(!W)(!E)
- Code pages ⇒ R(!W)E What about self modifying code?
- Illegal pages ⇒ (!R)(!W)(!E) Why would any one want this?

Another Use of Segments

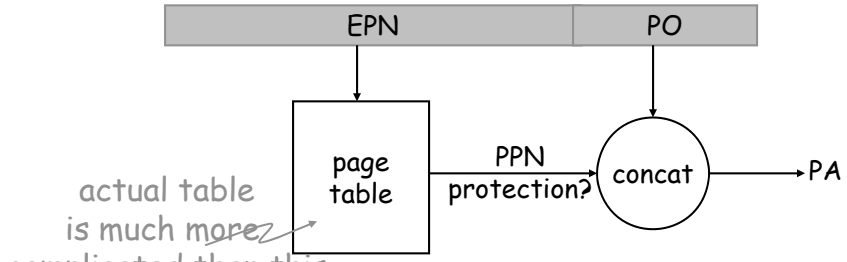
- How to extend an old ISA to support larger addresses for new applications while remain compatible with old applications?
- User-level segmented addressing
  - old applications use identity mapping in table
  - new applications reload segment table at run time with "active segments" to access different regions in memory
  - complications of a "non-linear" address space: dereferencing some pointers (aka long pointers) requires more work if it is not in an active segment



can be orthogonal from protection considerations

Paged Address Space

- Divide PA and EA space into fixed size segments known as "page frames", historically 4KByte
- EAs and PAs are interpreted as page number (PN) and page offset (PO)
  - page table translates EPN to PPN
  - VPO is the same as PPO, just concatenate to PPN to get PA



actual table is much more complicated than this

Fragmentation

- External Fragmentation
  - a system may have plenty of unallocated DRAM, but they are useless in a segmented system if they do not form a contiguous region of a sufficient size
  - paged memory management eliminates external fragmentation
- Internal Fragmentation
  - with paged memory management, a process is allocated an entire page (4KByte) even if it only needs 4 bytes
  - a smaller page size reduces likelihood for internal fragmentation
  - modern ISA are moving to larger page sizes (Mbytes) in addition to 4KBytes Why?

Demand Paging

- Use main memory and "swap" disk as automatically managed levels in the memory hierarchies

  analogous to cache vs. main memory

- Early attempts
  - von Neumann already described manual memory hierarchies
  - Brookner's interpretive coding, 1960
    - a software interpreter that managed paging between a 40KByte main memory and a 640KByte drum
  - Atlas, 1962
    - hardware demand paging between a 32-page (512 word/page) main memory and 192 pages on drums
    - user program believes it has 192 pages

Demand Paging vs. Caching

- Drastically different size and time scale

  ⇒ drastically different design considerations

	L1 Cache	L2 Cache	Demand Paging
Capacity	10~100KByte	MByte	GByte
Block size	~16 Byte	~128 Byte	4K~4M Byte
hit time	a few cyc	a few 10s cyc	a few 100s cyc
miss penalty	a few 10s cyc	a few 100s cyc	10 msec
miss rate	0.1~10%	(?)	0.00001~0.001%
hit handling	HW	HW	HW
miss handling	HW	HW	SW

Hit time, miss penalty and miss rate are not really independent variables!!

Same Fundamentals

- Potentially $M=2^m$ bytes of memory, how to keep the most frequently used ones in C bytes of fast storage where $C << M$
- Basic issues

  (1) where to "cache" a virtual page in DRAM?

  (2) how to find a virtual page in DRAM?

  (3) granularity of management

  (4) when to bring a page into DRAM?

  (5) which virtual page to evict from DRAM to disk to free-up DRAM for new pages?

DRAM in the case of demand paging

(5) is much better covered in an OS course

BTW, architects should take OS and compiler

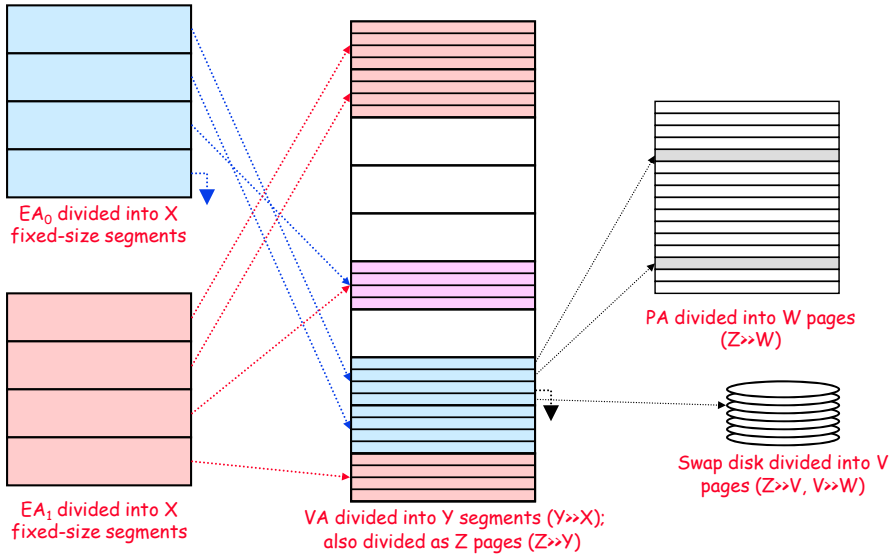
Terminology and Usage

- Physical Address
  - addresses that refer directly to specific locations on DRAM or on swap disk
- Effective Address
  - addresses emitted by user applications for data and code accesses
  - usage is most often associated with "protection"
- Virtual Address
  - addresses in a large, linear memory seen by user app's

    often larger than DRAM and swap disk capacity
  - virtual in the sense that some addresses are not backed up by physical storage (hopefully you don't try to use it)
  - usage is most often associated with "demand paging"

Modern memory system always have both protection and demand paging; usage of EA and VA is sometimes muddled

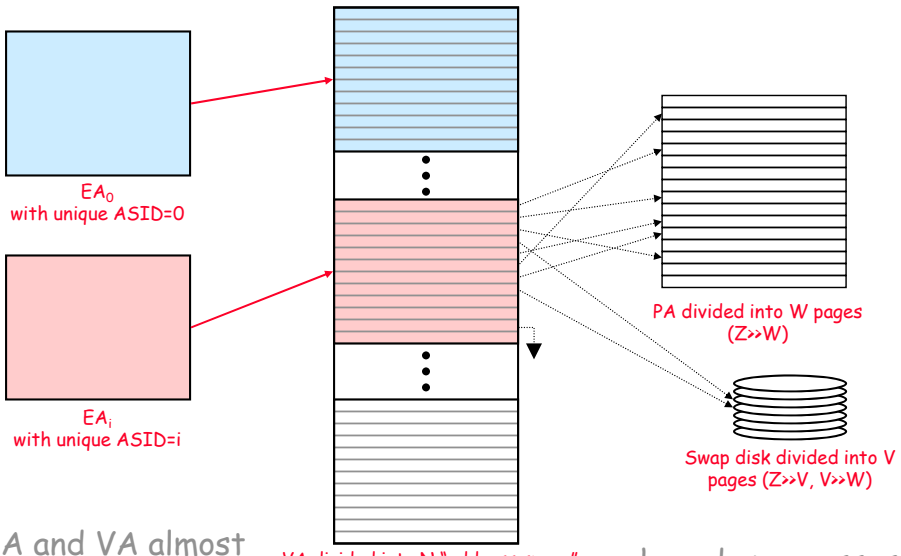
EA, VA and PA (IBM's view)



segmented EA: private, contiguous + sharing

demand paged VA: size of swap, speed of DRAM

EA, VA and PA (almost everyone else)



EA and VA almost synonymous

how do processes share pages?