Lec 20: Virtual Memory

Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

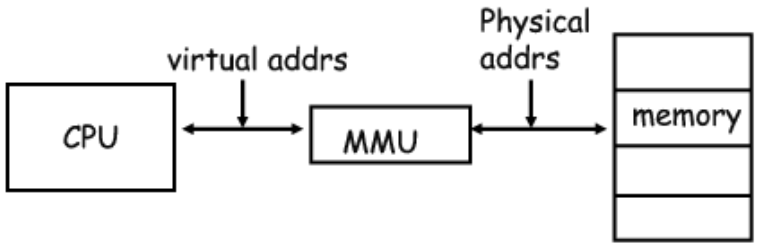
Announcements

- HW 3 out: cache simulation
  - Recitations this week on debugging C
  - Due Nov 6th
- HW 4: smash the stack
  - Out Nov 6th
  - Due Nov 14th
- PA 3 out Nov 14th
  - Due Nov 25th (feel free to turn it in early)
  - Demos and pizza party: Dec 1st or 2nd
- Prelim 2: Dec 4th
- Final project: distributed multicore ray tracer
  - Due exam week

© Kavita Bala, Computer Science, Cornell University

How to make it work?

- Challenge: Virtual Memory can be slow!
- At run-time: virtual address must be translated to a physical address
- MMU (combination of hardware and software)



© Kavita Bala, Computer Science, Cornell University

Address Translation

- How to translate addresses?
  - Per word? Much too expensive
  - Per block? Sure, but what is block size?
- Costs dictate granularity of translation
  - Cost to disk is very large
  - Block size has to be large too
    - Amortization

© Kavita Bala, Computer Science, Cornell University

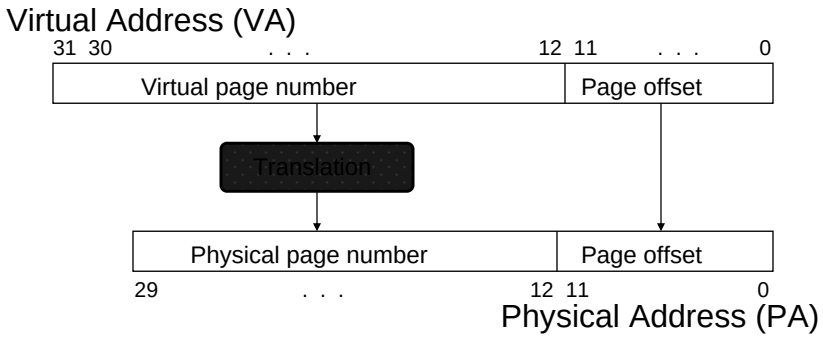
Page Table

- Each process has separate mapping of virtual to physical pages
- Page Table: stores this translation
  - Basically a huge array of translations

© Kavita Bala, Computer Science, Cornell University

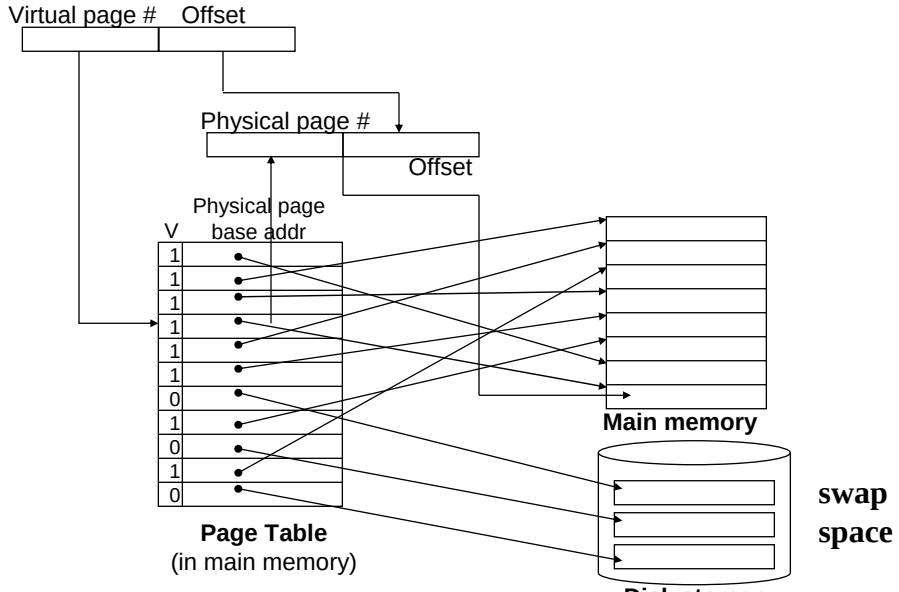
Address Translation

- So each memory request *first* requires an address translation from the virtual space to the physical space



© Kavita Bala, Computer Science, Cornell University

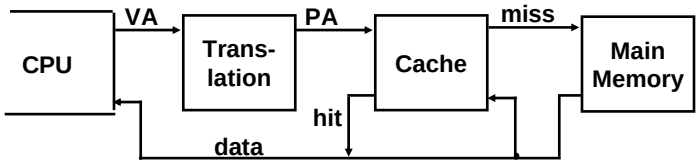
Page Table for Translation



© Kavita Bala, Computer Science, Cornell University

Virtual Addressing with a Cache

- Thus it takes an *extra* memory access to translate a VA to a PA



- This makes memory (cache) accesses very expensive (if every access was really *two* accesses)

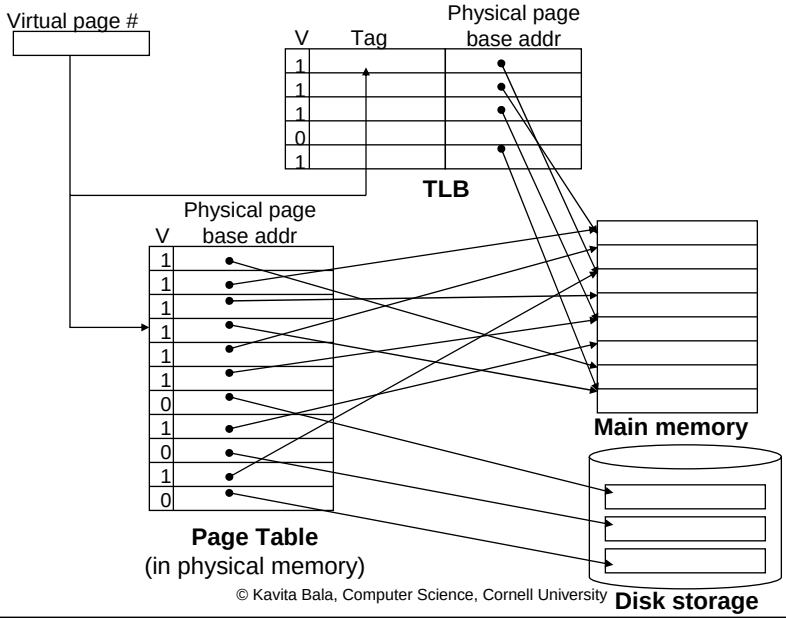
© Kavita Bala, Computer Science, Cornell University

Translation Lookaside Buffer (TLB)

- Page Table has to be in main memory
- But still too slow
- What's our solution? A cache of course
- Hardware: TLB
- A small cache of recently used address mappings
  - TLB hit: avoid a page table lookup

© Kavita Bala, Computer Science, Cornell University

Making Address Translation Fast



© Kavita Bala, Computer Science, Cornell University

Translation Lookaside Buffers (TLBs)

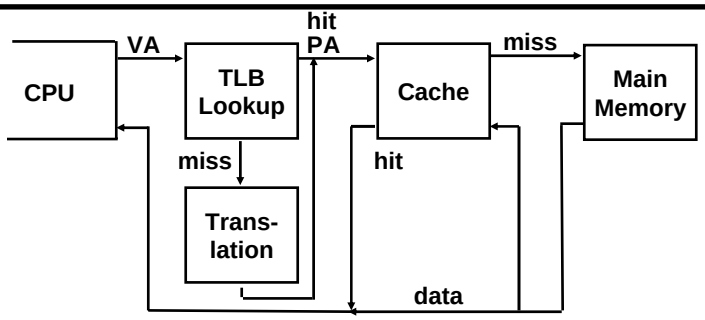
- TLB can be organized as fully associative, set associative, or direct mapped
  - Typically fully associative

V	Virtual Page #	Physical Page #	Dirty	Ref

- TLB access time is typically smaller than cache access time
  - Because TLBs are much smaller than caches
  - Typically not more than 64 to 512 entries
  - CPU pipeline speed: small/fast

© Kavita Bala, Computer Science, Cornell University

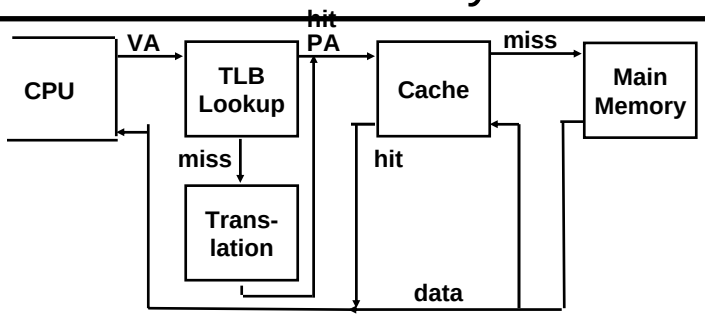
A TLB in the Memory Hierarchy



- A TLB miss:
  - If page in main memory, TLB miss can be handled (in hardware or software)
    - Load from the page table into the TLB
    - Takes 10's of cycles

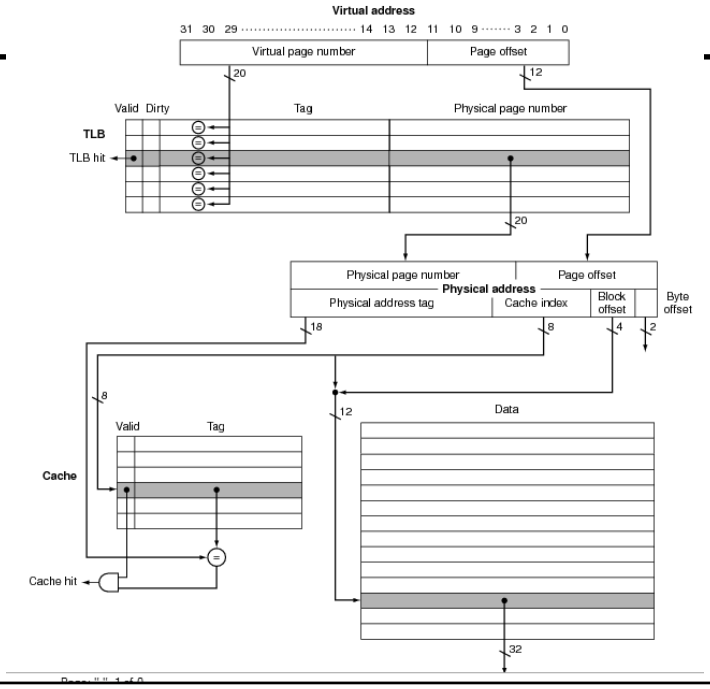
© Kavita Bala, Computer Science, Cornell University

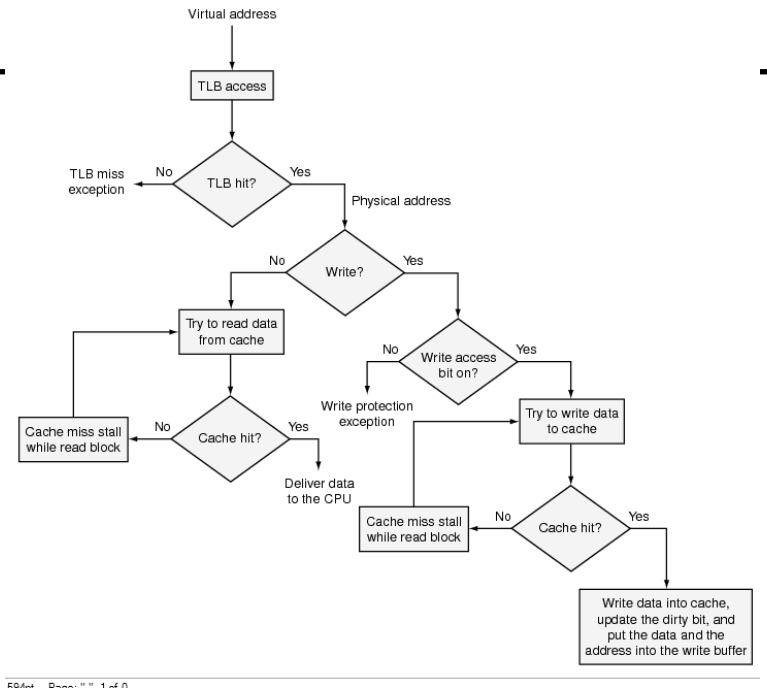
A TLB in the Memory Hierarchy



- A TLB miss:
  - If the page is not in main memory
    - Page fault!
    - Takes 1,000,000's of cycles to service a page fault
- TLB misses are much more frequent than true page faults

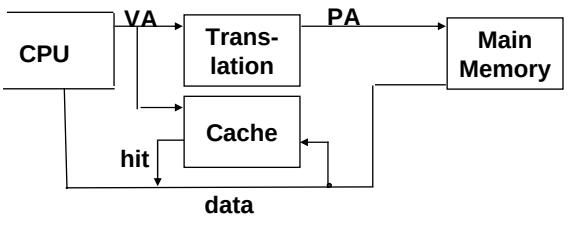
© Kavita Bala, Computer Science, Cornell University





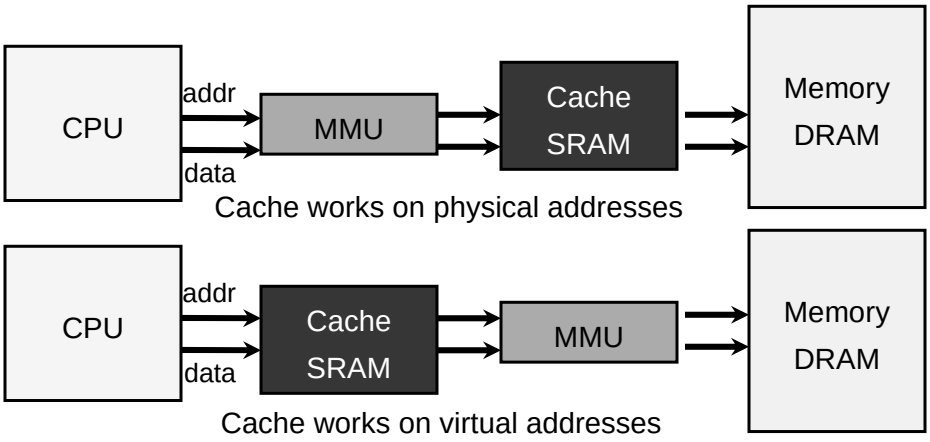
Remove TLB from critical path?

- A virtually addressed cache would only require address translation on cache misses



© Kavita Bala, Computer Science, Cornell University

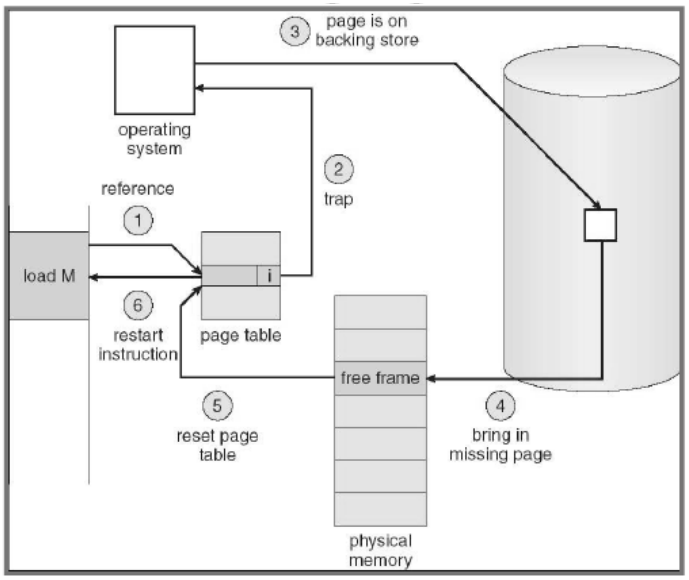
Virtual vs. Physical Caches



- L1 (on-chip) caches are typically virtual
- L2 (off-chip) caches are typically physical

© Kavita Bala, Computer Science, Cornell University

Paging



© Kavita Bala, Computer Science, Cornell University