Review

- Last Class:
  - Review memory hierarchy
  - Direct mapped, set associative caches
- Today's class:
  - Basic cache optimizations
  - Address translation with caches
  - Advance cache optimizations
- Announcement and reminder
  - Reading assignment R1 dues tonight

24

Cache Performance (§B.2)

$CPUtime = IC \times (CPI_{execution} + Misses\ per\ instruction \times Miss\ penalty) \times Clock\ cycle\ time$

IC = Instruction Count

Misses per instruction = Memory accesses per instruction x Miss rate

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

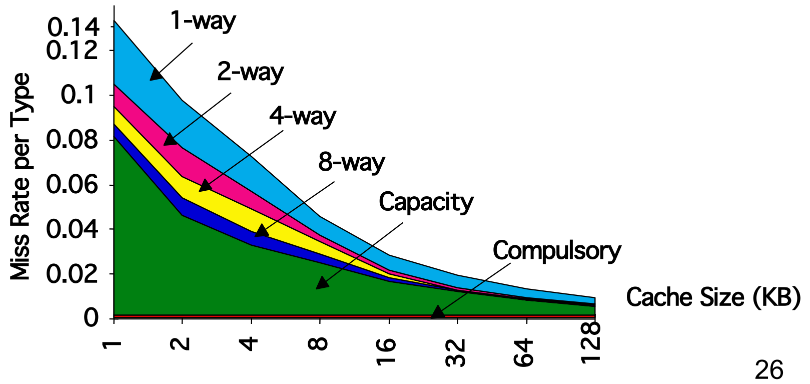
Improving Cache Performance:

1. Reduce the miss rate,
2. Reduce the miss penalty,
3. Reduce the time to hit in the cache.

25

Classification of cache misses

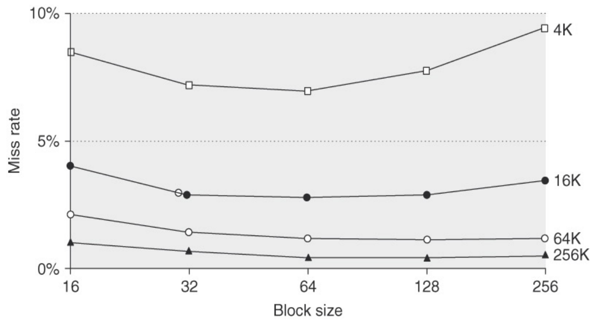
– Compulsory Misses: Sad facts of life. Example: cold start misses.
– Capacity Misses: Increase cache size
– Conflict Misses: Increase cache size and/or associativity.
  Nightmare Scenario: ping pong effect!



26

Basic ways to reduce miss rate (§ B.3)

1. Larger Block Size



- Block size explores spatial locality,
- Large blocks are only useful if penalty to fetch a block of K words is less than K times the penalty to fetch one word
- Why can a large block size help and why can it hurt?
- Which of the three C's are affected by the block size?

27

2. Larger caches

- Which of the three C's are affected by the cache size?
- Why can't we increase the size of the cache arbitrarily?

3. Higher Associativity

- Which of the three C's are affected by associativity?
- 8-way set associativity is as good as full associativity
- 2:1 Cache Rule:
  - Miss Rate DM cache size N = Miss Rate 2-way cache size N/2
- Beware: Execution time is the only final measure!
  - Will hit latency (possibly clock cycle time) increase?
- Effect on power consumption?
- Example: If higher associativity increases the hit time by 20% but decreases the miss rate from 5% to 2.5%, would you go with higher associativity? - Will your decision depend on other factors?

28

Basic ways to reduce the Miss Penalty

4. Multi-level cache

AMAT = Acc Time$_{L1}$ + *Miss Rate*$_{L1}$ *x (Acc Time*$_{L2}$ + *Miss Rate*$_{L2}$ *x Miss Penalty*$_{L2}$*)*

Local miss rate— misses in this cache divided by the total number of memory accesses *to this cache* (Miss rate$_{L2}$)

Global miss rate—misses in this cache divided by the total number of memory accesses *generated by the CPU* (Miss Rate$_{L1}$ x Miss Rate$_{L2}$)

- L2 is larger (fewer misses) and slower (larger hit time) than L1
- Since hits are few in L2, may target miss penalty reduction
- In L1, may target access time reduction.
- May use different organizations and block sizes (easy??)
- Multilevel inclusion is desirable.
- **Danger**: time to DRAM will grow with multiple levels in between
- Out-of-order CPU can hide L1 data cache miss (3–5 clocks), but stall on L2 miss (40–100 clocks)?

29

Example

- Miss rate of L1 = 4%, acc time is one cycle
- If L2 is direct mapped, local miss rate = 25% and acc time = 10 cycles
- If L2 is set-associative, local miss rate = 20% and acc time = 11 cycles
- Miss penalty for L2 is 100 cycle.

AMAT with direct mapped L2 = 1 + 0.04 (10 + 0.25 * 100) = 2.4 cycles
AMAT with set-associative L2 = 1 + 0.04 (11 + 0.20 * 100) = 2.24 cycles

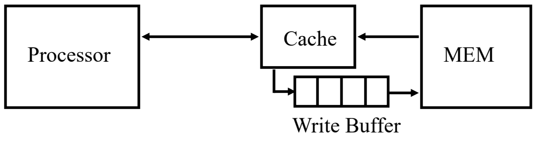
- Compare with a system with larger L1 and no L2, and improved L1 miss rate of 2%

AMAT with large L1 and no L2 = 1 + 0.02 (100) = 3 cycles

- Taking into consideration the effect of writing back replaced dirty blocks complicates slightly the analysis.

30

5. Read Priority over Write on Miss



- If write through, write buffers avoid stalling
- May causes RAW conflicts with reads on cache misses
- Waiting for write buffer to empty will increase read miss penalty
  - Solution: Check write buffer contents before read;
    if no conflicts, let the memory access continue

- If write Back (replace dirty block on a miss)
  – CPU stalls less since restarts as soon as the read completes
  – Cache coherence problem in multi-core execution. Stay tuned.

31

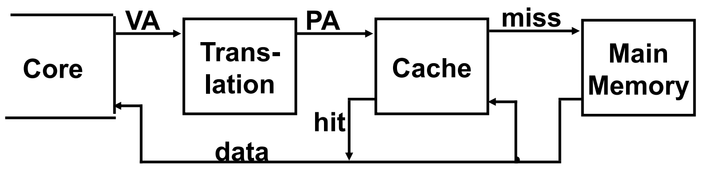
Review: Virtual Memory Concepts

- Use main memory as a "cache" for secondary memory
  - Allows efficient and safe sharing of main memory among multiple processes/threads (running programs)
    - Each program is compiled into its own private virtual address space
  - Provides the ability to run programs and data sets larger than the size of physical memory
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code/data can be loaded in main memory anywhere the OS can find space for it)
- The core and OS work together to translate virtual addresses to physical addresses
  - A virtual memory miss (i.e., when the page is not in physical memory) is called a page fault
- What makes it work efficiently? – the Principle of Locality
  - Programs tend to access only a small portion of their address space over long portions of their execution time

32

Virtual Addressing with a Cache

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



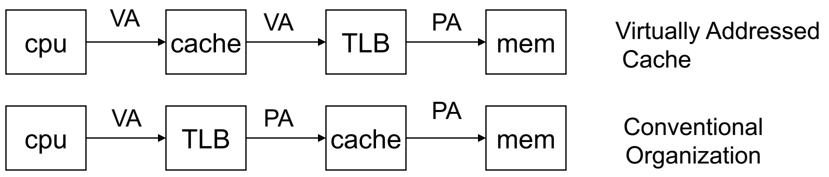
- This makes memory (cache) accesses very expensive (if every access is really *two* accesses)
- The hardware fix is to use a Translation Lookaside Buffer (TLB) – a fast, small read-only **cache** that keeps track of recently used **address mappings** to avoid having to do a page table lookup in memory
- Typical TLBs – 16 to 512 PTEs, 0.5 to 2 cycle for a hit, 10-100 cycles for a miss, 0.01% to 1% miss rate

33

A basic way to reduce the Hit time

6. Avoiding Address Translation

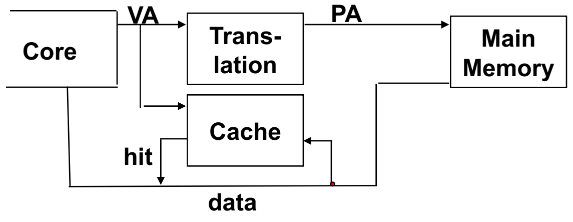
- Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* -- not *Physical Cache.*
  - Every context switch, must flush the cache; otherwise get false hits. Cost is time to flush + "compulsory" misses from empty cache
  - Does not support *aliases* (*synonyms*); Two different virtual addresses mapped to the same physical address.
- To avoid cache flush, we may add *process identifier tag* to cache blocks.


$$\begin{tikzcd}
\boxed{\text{cpu}} \arrow[r, "VA"] & \boxed{\text{cache}} \arrow[r, "VA"] & \boxed{\text{TLB}} \arrow[r, "PA"] & \boxed{\text{mem}} & \text{Virtually Addressed Cache} \\
\boxed{\text{cpu}} \arrow[r, "VA"] & \boxed{\text{TLB}} \arrow[r, "PA"] & \boxed{\text{cache}} \arrow[r, "PA"] & \boxed{\text{mem}} & \text{Conventional Organization}
\end{tikzcd}$$

34

Why Not a Virtually Addressed Cache?

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

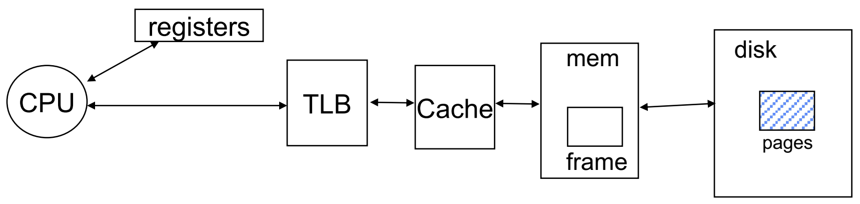


But,

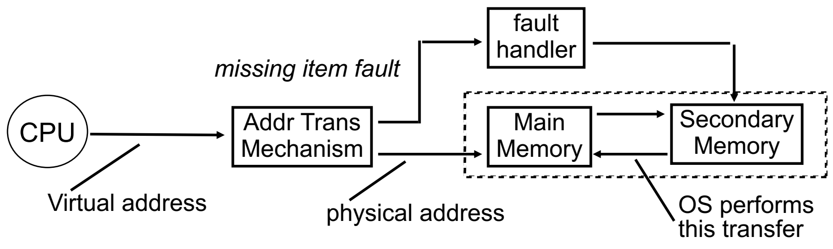
- Two programs which are sharing data will have two different virtual addresses for the same physical address – aliasing – so will have two copies of the shared data in the cache and two entries in the TLB which would lead to *coherence* issues
  - Must update all cache entries with the same physical address or the memory becomes inconsistent

35

Basic Issues in VM System Design (§ B.4)



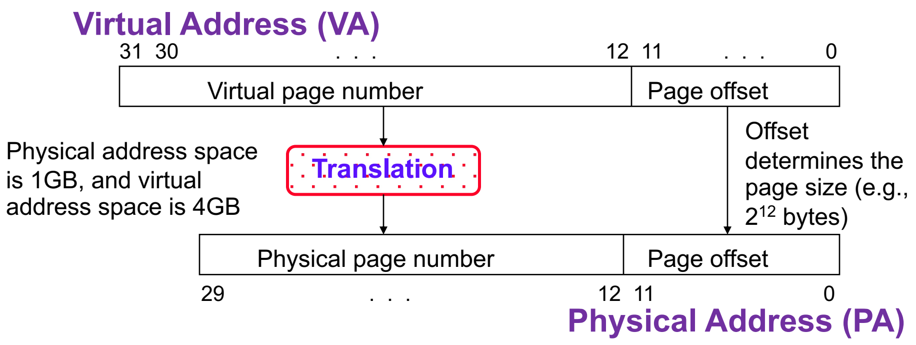
missing item fetched from secondary memory only on the occurrence of a fault --> *load on demand policy*



36

Address Translation

- A virtual address is translated to a physical address by a combination of hardware and software

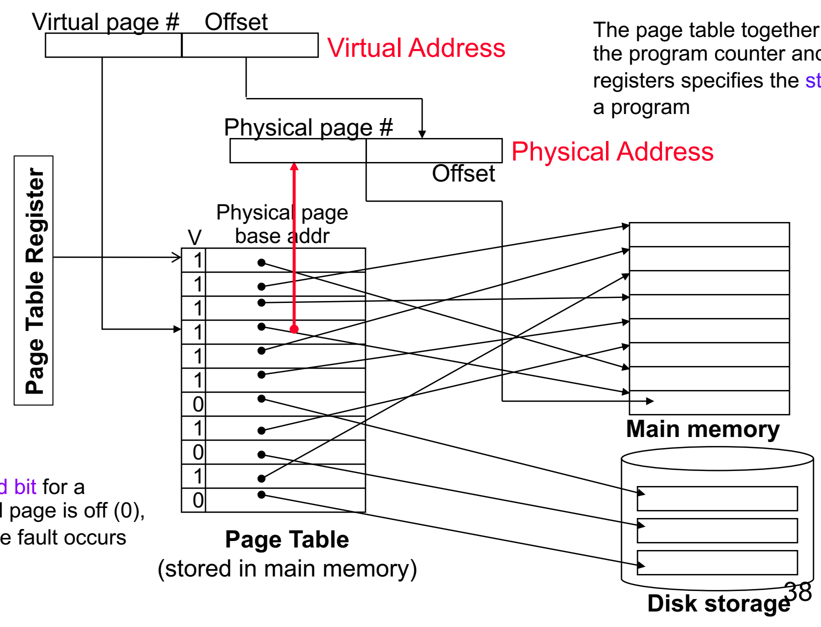


- So, each memory request *first* requires an address translation from the virtual space to the physical space

37

Virtual Address Translation Mechanisms

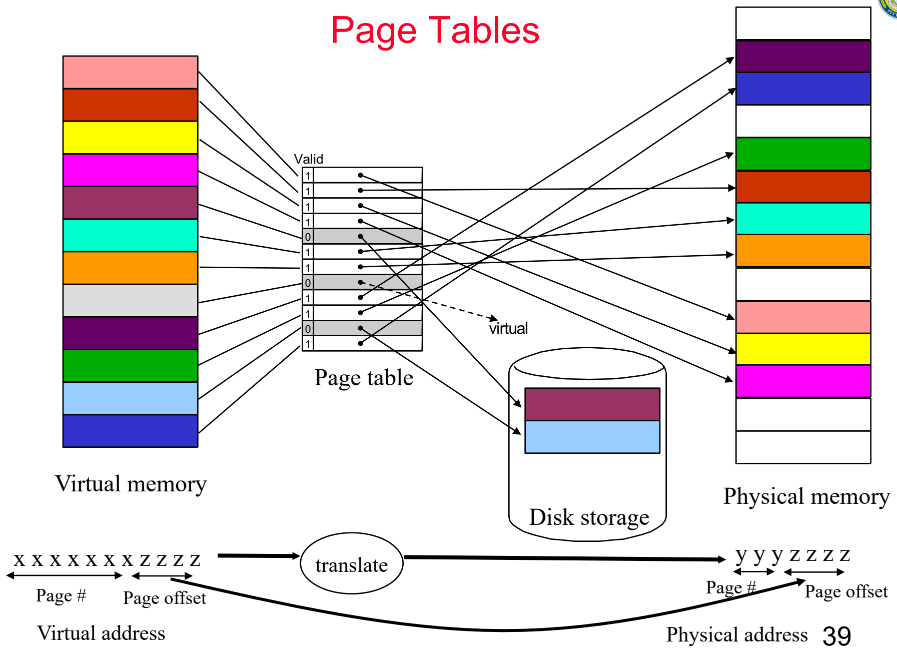
The page table together with the program counter and the registers specifies the state of a program



If valid bit for a virtual page is off (0), a page fault occurs

38

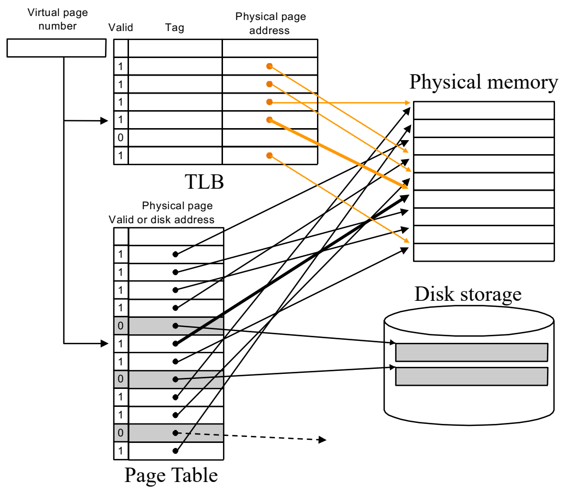
Page Tables



39

Making Address Translation Fast

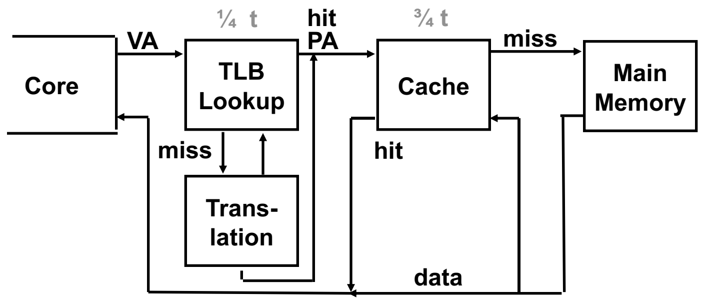
Cache the currently used entries of PT in a *Translation Lookaside Buffer* (TLB).



- What if we get a TLB miss (page table entry is not in TLB)?

40

A TLB in the Memory Hierarchy



- ❑ **A TLB miss – is it a page fault or merely a TLB miss?**
  - ● If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB
    - Takes 100's of cycles to find and load the translation info into the TLB
  - ● If the page is not in main memory, then it's a *true* page fault
    - Takes 1,000,000's of cycles to service a page fault
- ❑ TLB misses are much more frequent than true page faults

41

Handling a TLB Miss

- A TLB miss can indicate one of two possibilities:
  - A page is present in memory, and we need only create the missing TLB entry
  - A page is not present in memory, and we need to transfer control to the operating system to deal with a page fault
- MIPS traditionally handles a TLB miss in *OS*
- Handling a TLB miss or a page fault requires using the exception mechanism to interrupt the active process, transferring control to the operating system, and later resuming execution of the interrupted process
- A TLB miss or page fault exception must be asserted by the end of the same clock cycle that the memory access occurs, so that the next clock cycle will begin exception processing rather than continue normal instruction execution

42

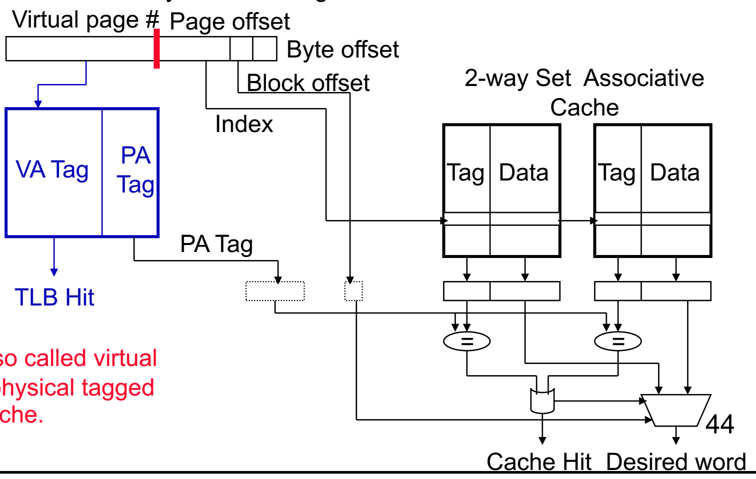
Handling a TLB Miss

- Once the operating system knows the virtual address that caused the page fault, it must complete three steps:
  - Look up the page table entry using the virtual address and find the location of the referenced page on disk
  - Choose a physical page to replace; if the chosen page is dirty, it must be written out to disk before we can bring a new virtual page into this physical page
  - Start a read to bring the referenced page from disk into the chosen physical page
- The last step will take *millions of clock cycles* (so will the second if the replaced page is dirty)
- Accordingly, the operating system will usually select/schedule another process to execute in the processor *until* the disk access completes
- When the read of the page from the disk completes, the operating system can restore the state of the process that originally caused the page fault and execute the instruction that returns from the exception
- The user process (application) then re-executes the instruction that faulted

43

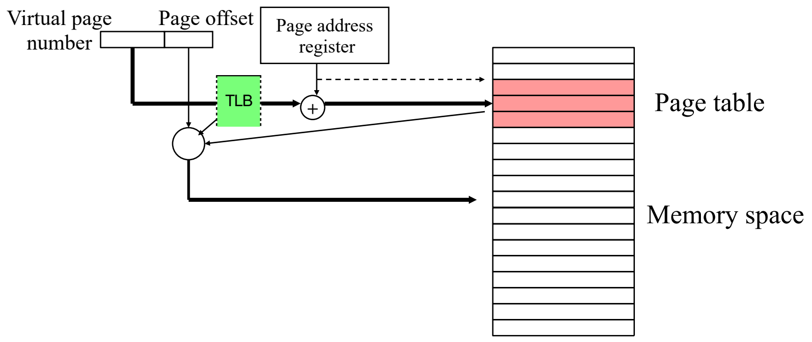
Further Reducing Translation Time

- Can overlap the cache access with the TLB access
- Overlapped access only works as long as the address bits used to index into the cache ***do not change*** as the result of VA translation
- This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache



This is also called virtual indexed physical tagged (VIPT) cache.

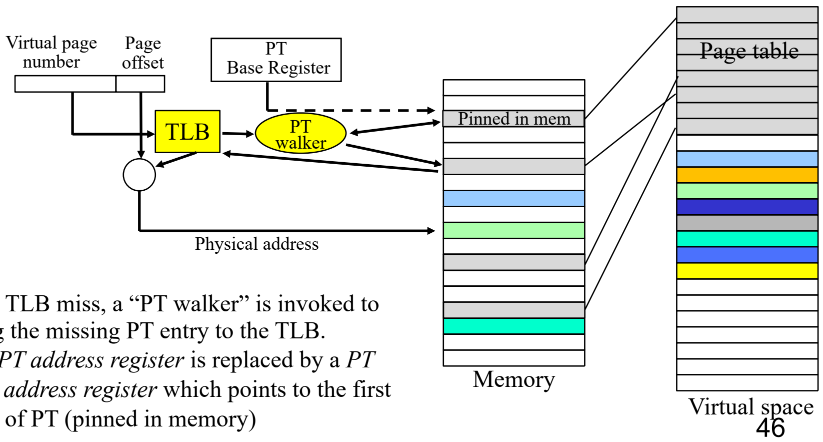
Page Tables



- Each process has its own page table stored in memory starting at a specific address indicated in the *page address register*.
- The page table and page address register are part of the process context
- Each memory reference (hit) requires two memory operations???
- Each page fault requires a memory operation and a disk operation
- The page table can get very large (ex. 32 bit virtual address (4GB), 4KB pages, 4 bytes per PT entry ==> PT size = 4MB = 1024 pages).
- Each process has its own page table => large memory space occupied
- Page table located in reserved memory and operated by OS

The Page Table (PT) is very large

- PT's currently used pages are brought to memory, and like any other page in the virtual space, the location of a page in memory is recorded in PT
- Note that (in our example) the 1024 PT entries corresponding to the 1024 pages of the PT can fit in the first page of the PT – That page is pinned in memory.

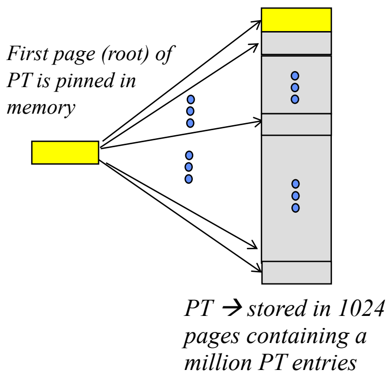


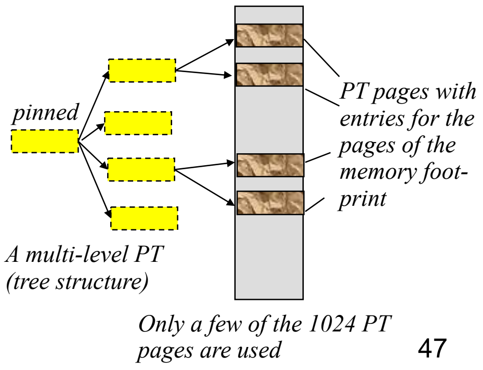
- On a TLB miss, a "PT walker" is invoked to bring the missing PT entry to the TLB.
- The *PT address register* is replaced by a *PT base address register* which points to the first page of PT (pinned in memory)

46

Multi level Page Tables (multi level PT)

- In the example of 4GB and 4KB pages, the PT can be stored in 1024 pages
  - Pages of the PT are brought to memory on demand
  - The first page (root) of the PT keeps track of the locations of PT pages in memory.
- This is a "2-level" PT organization – may generalize to a multi-level PT organization
- Memory foot-print = the part of the VS which is actually used (accessed)
  - A large number of pages in the VS are not allocated or used (empty).
  - Hence a large number of entries of the PT are never accessed


First page (root) of PT is pinned in memory
PT → stored in 1024 pages containing a million PT entries


pinned
A multi-level PT (tree structure)
PT pages with entries for the pages of the memory foot-print
Only a few of the 1024 PT pages are used

47

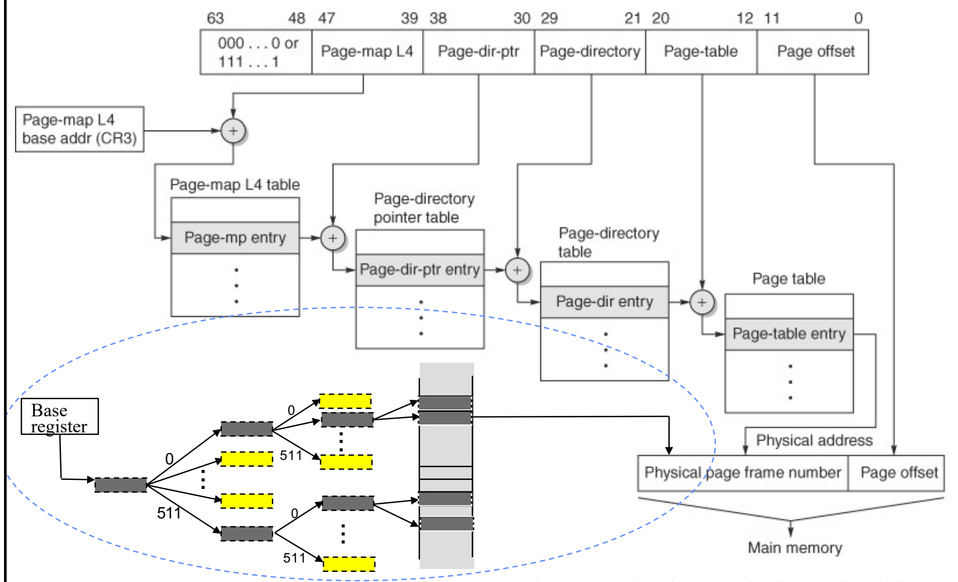
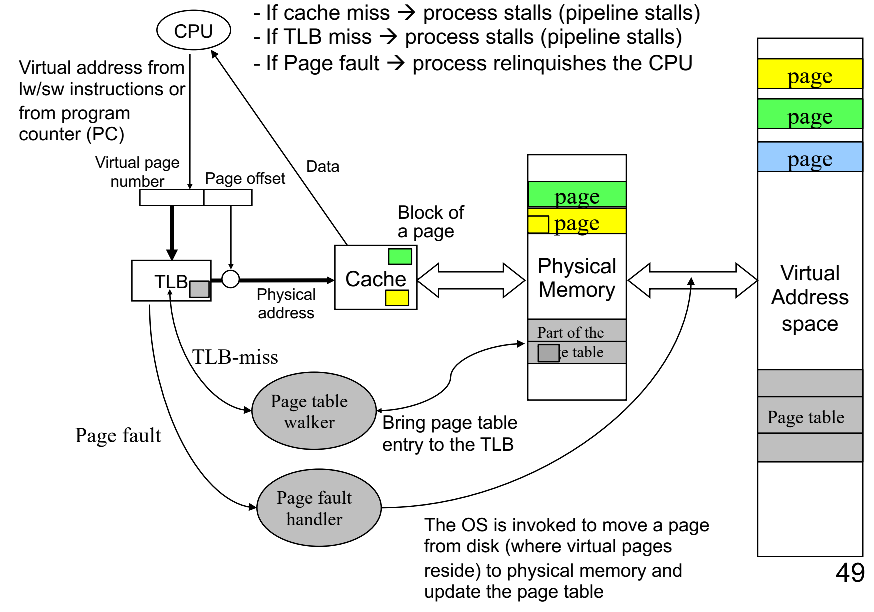


Figure B.27 The mapping of an Opteron virtual address. The Opteron virtual memory implementation with four page table levels supports an effective physical address size of 40 bits. Each page table has 512 entries, so each level field is 9 bits wide. The AMD64 architecture document allows the virtual address size to grow from the current 48 bits to 64 bits, and the physical address size to grow from the current 40 bits to 52 bits.

The whole picture

- If cache miss → process stalls (pipeline stalls)
- If TLB miss → process stalls (pipeline stalls)
- If Page fault → process relinquishes the CPU


CPU
Virtual address from lw/sw instructions or from program counter (PC)
Virtual page number
Page offset
Data
TLB
Physical address
Cache
Block of a page
page
page
Physical Memory
Part of the page table
page
page
page
Virtual Address space
Page table
TLB-miss
Page table walker
Bring page table entry to the TLB
Page fault
Page fault handler
The OS is invoked to move a page from disk (where virtual pages reside) to physical memory and update the page table

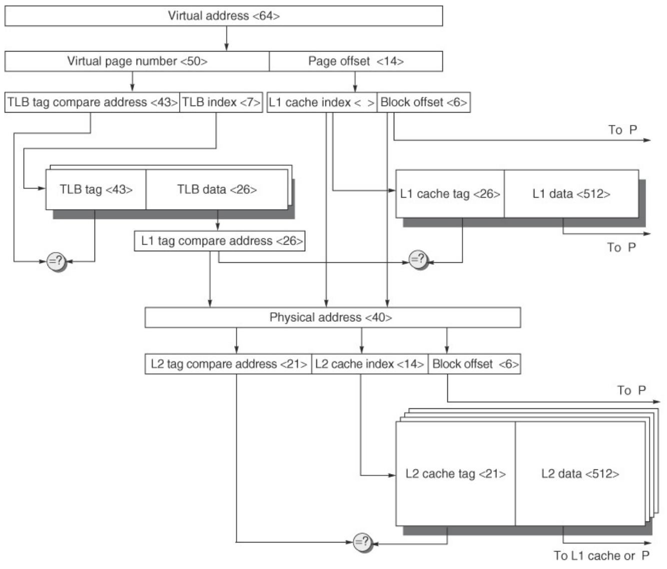


Figure B.17 The overall picture of a hypothetical memory hierarchy going from virtual address to L2 cache access. The page size is 16 KB. The TLB is two-way set associative with 256 entries. The L1 cache is a direct-mapped 16 KB, and the L2 cache is a four-way set associative with a total of 4 MB. Both use 64-byte blocks. The virtual address is 64 bits and the physical address is 40 bits.