

# Virtual Memory

**Dr. Tsahi Birk and Dr. Freddy Gabbay**  
**EE Department, Technion**

# Virtual Memory

- **Virtual address: issued by the CPU and the program.**
- **Physical address: address understood by physical memory chips.**
- **Allows each program to use addresses independently of other programs, while guaranteeing that different programs do not step on each other in memory.**
- **Allows multiple processes to share relatively small physical memory.**
- **Allows programs to maintain virtual mem. space > available physical memory**
- **No involvement of the programmer - programs are allocated automatically in the physical memory**
- **Easy to protect information.**
- **May reduce the loading time of programs.**

# Virtual Memory

- **Virtual address ( $2^{32}$ ,  $2^{64}$ ) to Physical Address mapping ( $2^{28}$ )**
- **Virtual memory terms for cache terms:**
  - Cache block → page
  - Cache Miss → page fault
- **How is virtual memory different from caches?**
  - What Controls Replacement
  - Size (transfer unit, mapping mechanisms)
  - Lower level use (I/O)

# Virtual Memory

- **4Qs for VM?**

- **Q1: Where can a block be placed in the upper level?**

*Fully Associative*

- **Q2: How is a block found if it is in the upper level?**

*Tag/Block in page table*

- **Q3: Which block should be replaced on a miss?**

*Random, LRU*

- **Q4: What happens on a write?**

*Write Back or Write Through (with Write Buffer)*

# Virtual → Physical Addr. Mapping

- **Page table:**
  - process id (PID) + virtual address → Physical address
  - table row (entry): PTE
- **Problems:**
  - with one PTE per potential page address, table size is proportional to size of address space · number of processes

Example: 64 bit address, 4KB pages, 1Byte addr. resolution, 8 Bytes per PTE, one process

⇒  $2^{64-12+3}$  Bytes =  $2^{25}$  Gigabytes for page table

  - table does not fit in cache, possibly not even in main memory, so
  - very long hit time in main memory
- **Observations:**
  - use of address space is very sparse
  - locality of reference
- **Solution:** smart representation of table + caching parts of its content (pages of the page table and/or individual PTEs).
- **Remark:** VM applied to main mem is at the HW-SW boundary:
  - translation in hits is handled by hardware
  - page faults and table updates are handled by software (operating system)

# Alpha VM Mapping Mechanism

- “64-bit” address divided into 3 segments
  - seg0 (bit 63=0) user code/heap
  - seg1 (bit 63 = 1, 62 = 1) user stack
  - kseg (bit 63 = 1, 62 = 0) kernel segment for OS
- PTE bits; valid, kernel & user read & write enable (No reference, use, or dirty bit)
- Separation among processes:
  - There is a different region of the table for each process
  - Specified by a base register and a bound on size for each process
  - PID → base register value

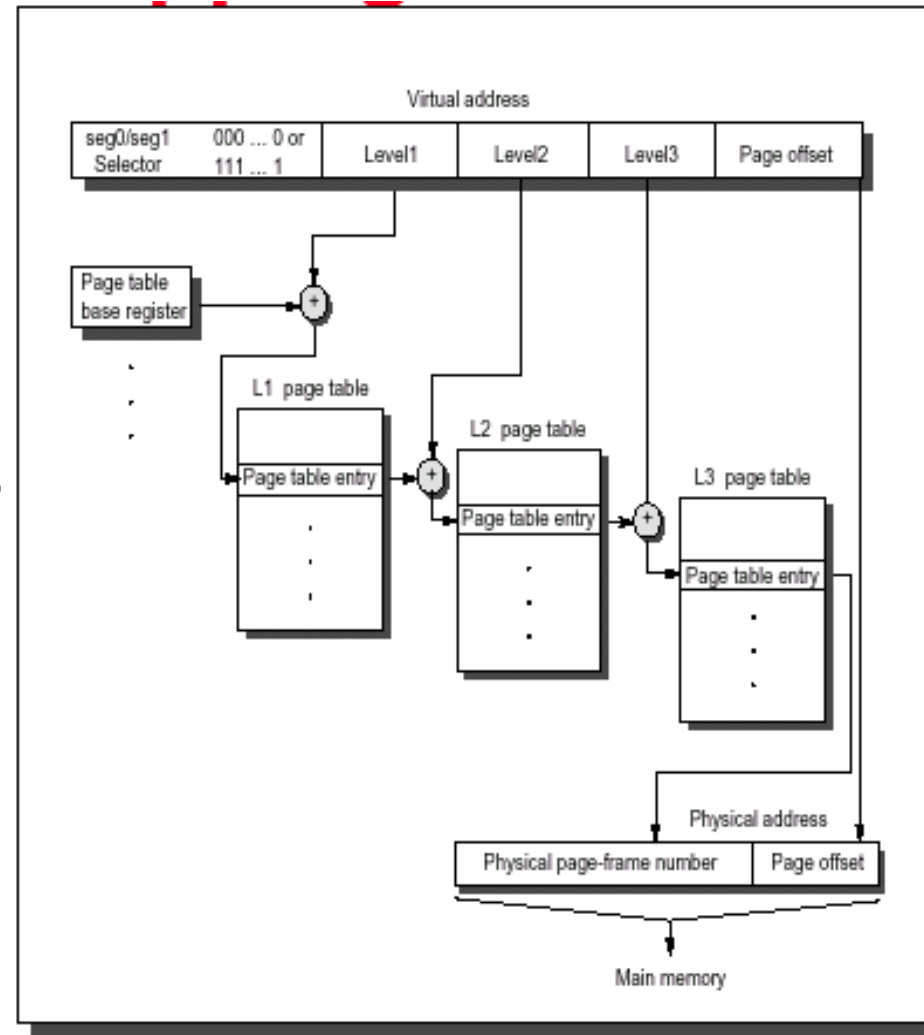


FIGURE 5.43 The mapping of an Alpha virtual address.

# Alpha VM Mapping Mechanism (cont.)

- **Three-level page table and 3-stage translation:**
  - each stage entails access to one page (indexed by 10 bits)
  - for each level-*i* line, there is a page (of mappings) at level *i+1* iff at least one address “covered” by this line is used
  - uses only 43 unique bits of VA: 3X10 + 13-bit page offset (8KB pages)
  - (future min page size up to 64KB => 55 bits of VA)
- **Table size is now OK, but what about translation time? (hit time!)**

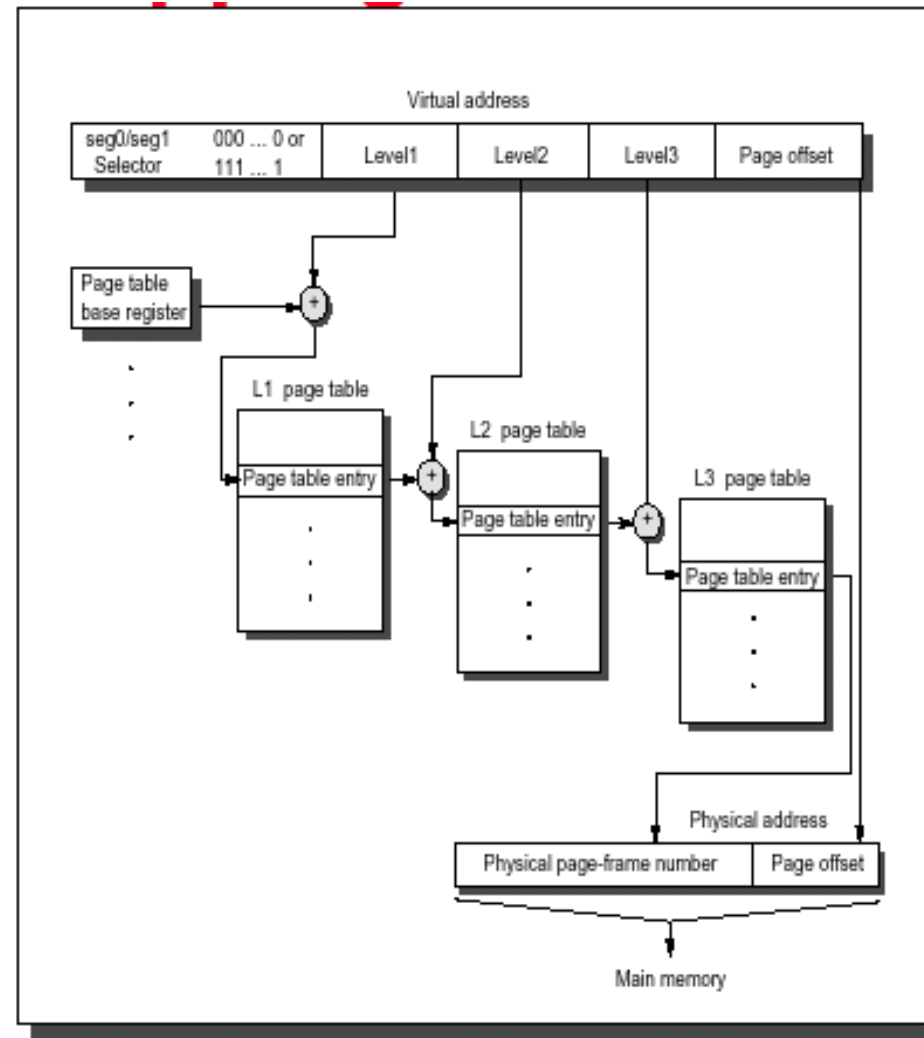
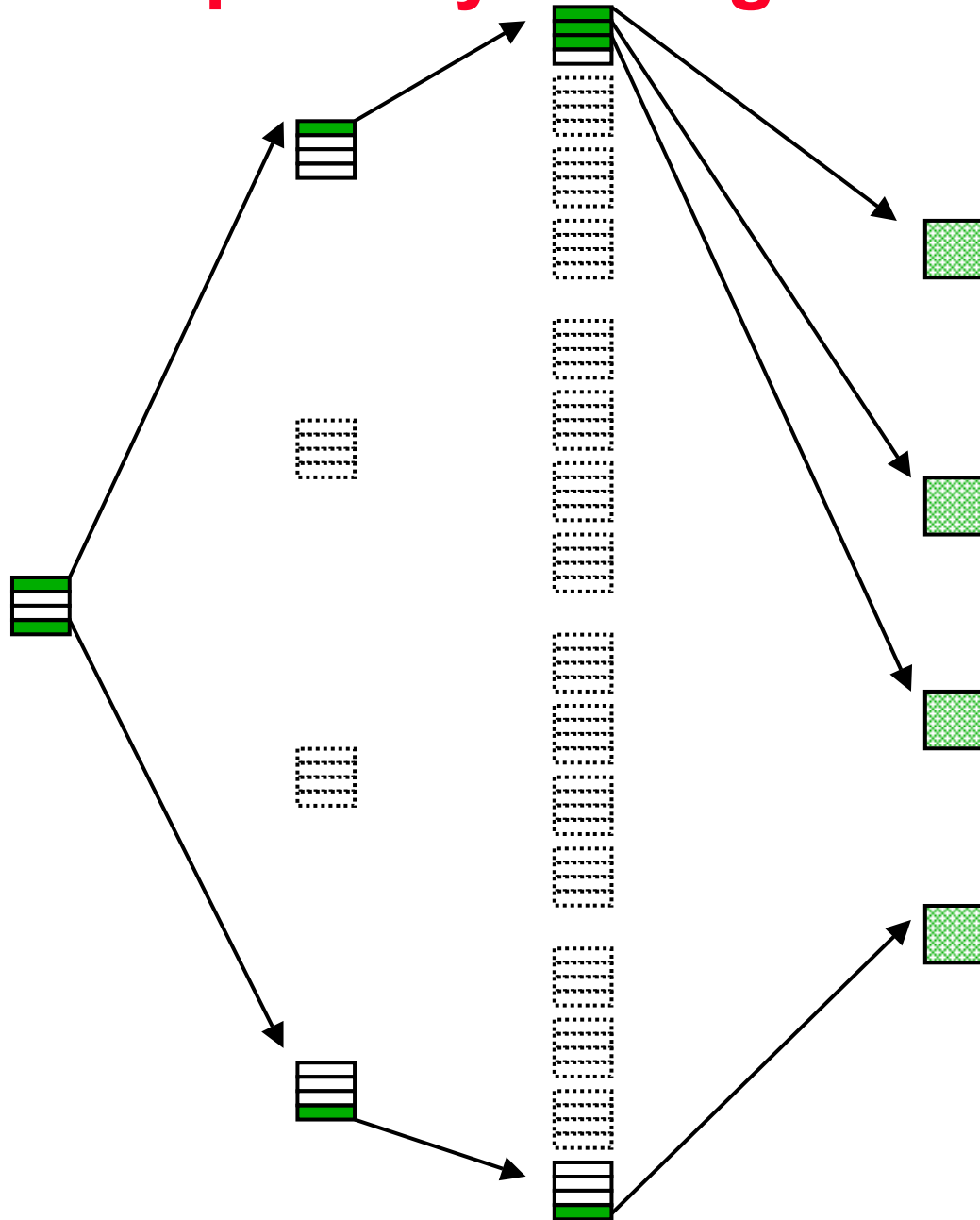






FIGURE 5.43 The mapping of an Alpha virtual address.

# “Alpha-Style” Page Table – Space Savings



Page table size:  
5 pages  
instead of 21 (excluding  
data pages)

-  – Not allocated
-  – Allocated+used
-  – Allocated but not used
-  – Allocated and possibly used



# Fast Translation: Translation Buffer

- Cache of translated addresses
- Alpha 21064 TLB: 32 entries, fully associative
- (TLB: translation look-aside buffer)
- “valid”, “read-permission”, “write-permission” (kernel/user) bits

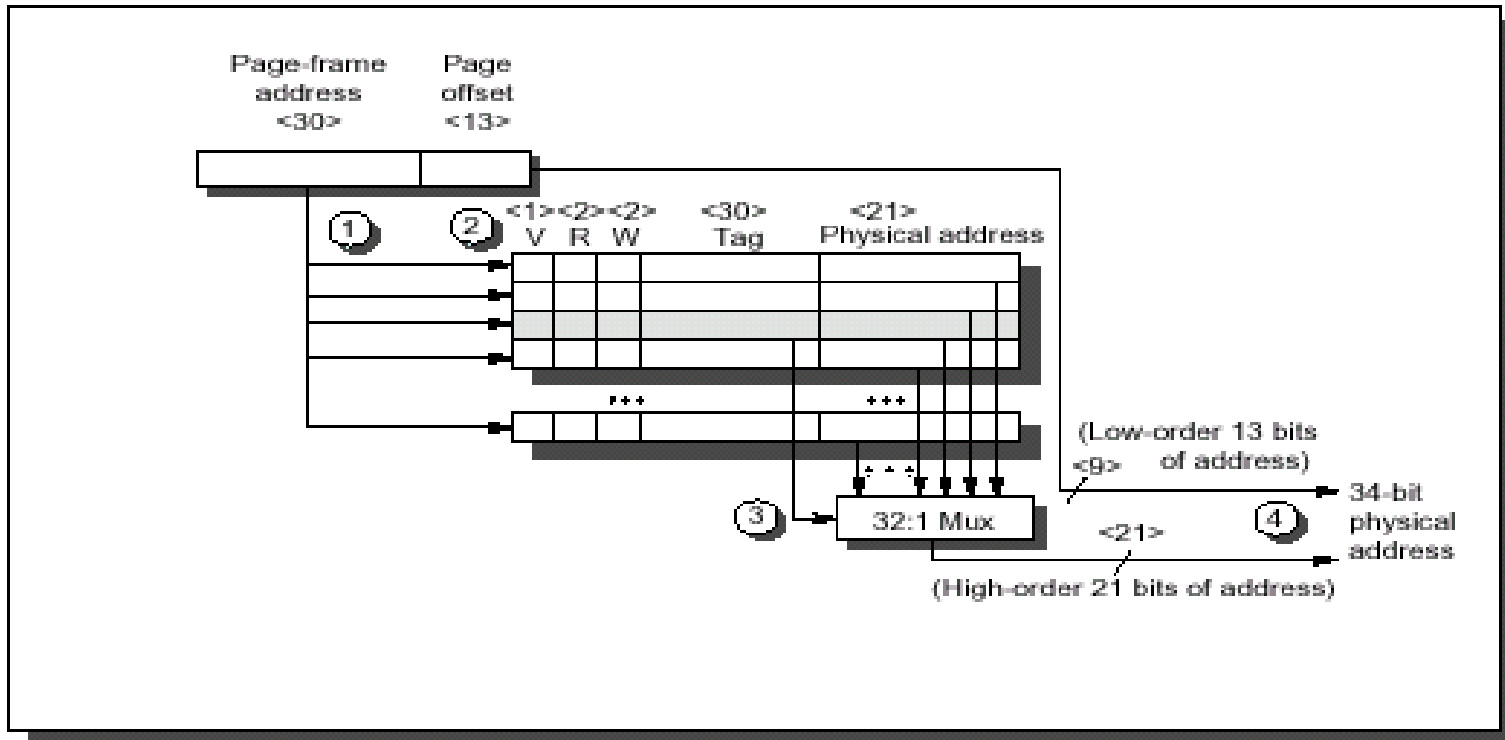


FIGURE 5.41 Operation of the Alpha AXP 21064 TLB during address translation.

# Selecting a Page Size

- **Reasons for larger page size**
  - Page table size is inversely proportional to the page size; therefore memory saved
  - Transferring larger pages to or from secondary storage, possibly over a network, is more efficient
  - Number of TLB entries is restricted by clock cycle time, so a larger page size maps more memory, thereby reducing TLB misses
  - Fast cache hit time easy when cache < page size; bigger page makes it feasible as cache size grows – we will discuss this in the next slides.
- **Reasons for a smaller page size**
  - Fragmentation: don't waste storage; data must be contiguous within page
  - Quicker process start for small processes(??)
- **Hybrid solution: multiple page sizes**
  - Alpha: 8KB, 16KB, 32 KB, 64 KB pages (43, 47, 51, 55 virt addr bits)

# Virtual Memory Impact on Cache Addressing and Hit time

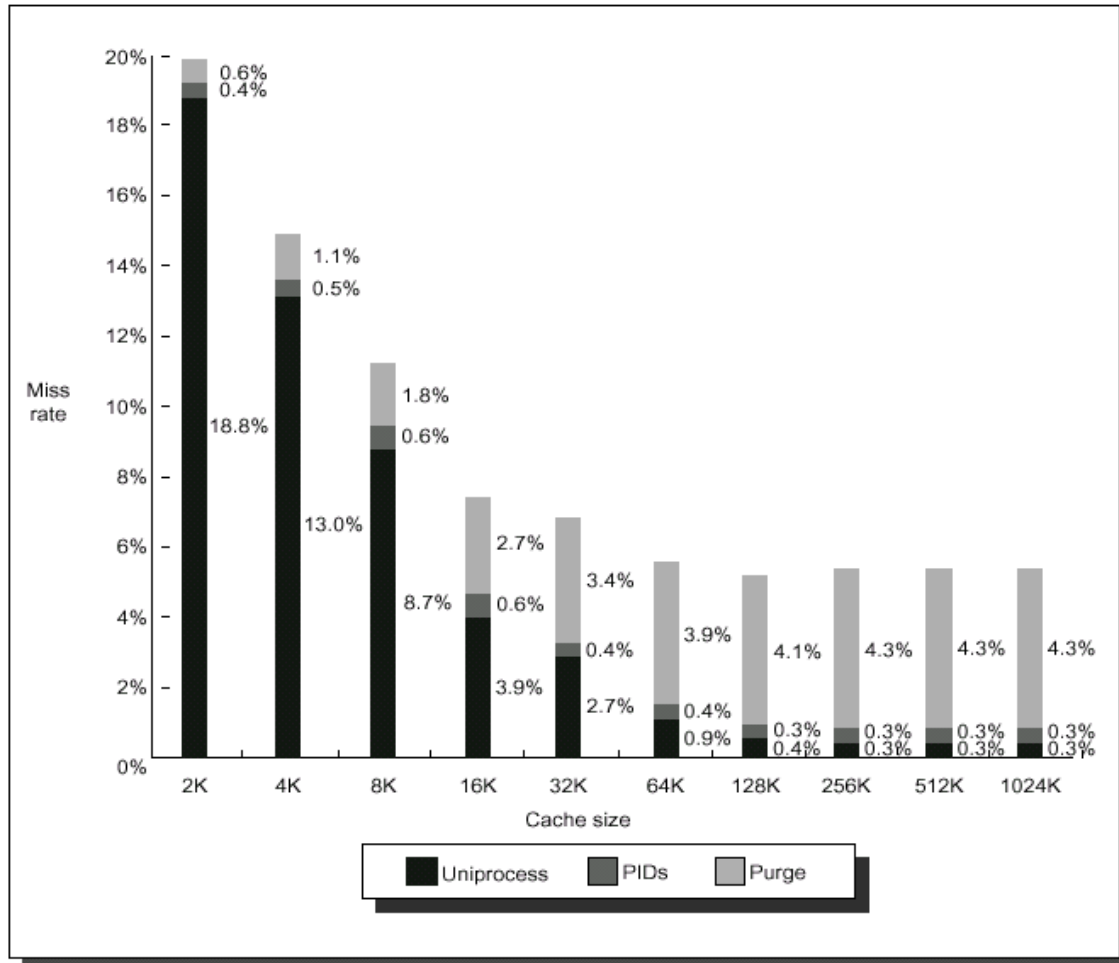
- **Virtually addressed cache:**
  - Cache access is based on virtual address issued by the CPU.
  - TAG and INDEX are extracted from virtual address.
  - Also termed “virtually indexed / virtually tagged”.
  - No impact on hit time - TLB is not in the path to cache hit.
- **Physically addressed cache:**
  - Cache access is based on physical address.
  - TAG and INDEX are extracted from physical address.
  - Also termed “physically indexed / physically tagged”.
  - Virtual address has to be translated first  
⇒ increasing hit time.

# Virtually Addressed Cache implications

- **Every time the OS makes a process-switch the cache content must be flushed, otherwise we get false hits.**
- **Overhead of flushing the cache:**
  - Accumulated locality is lost.
  - Time to flush the data (write back dirty blocks).
  - Reload misses (compulsory misses) for the new process.
- **Solution for flush penalty:**
  - Add a new “process id” (PID) field to each block in the cache.
  - PID can be regarded as a logical extension of the tags.
  - Cache hit  $\Leftrightarrow$  Tag matching and PID matching.
  - Problems with this solution:
    - » Increases associative memory size.
    - » Limits the number of processes that can run concurrently.

# PID Impact on Miss Rate

- **Black is uniprocess**
- **Light Gray is multiprocess when flush cache**
- **Dark Gray is multiprocess when use Process ID tag**
- **Y axis: Miss Rates up to 20%**
- **X axis: Cache size from 2 KB to 1024 KB**



**FIGURE 5.26** Miss rate versus virtually addressed cache size of a program measured three ways: without process switches (uniprocess), with process switches using a process-identifier tag (PIDs), and with process switches but without PIDs (purge).

# Aliasing (Synonym) Problem in Virtually Addressed Caches

- **Aliasing (synonym)** – two (or more) virtual addresses residing in the cache that are mapped to the same physical address in memory (intentionally shared among processes).
- **Consider the following scenario:**
  - Process A and B have shared data in memory for inter-process communication (IPC).
  - Each process views this same memory area as part of its own virtual memory space.
  - Assume we have a big cache to hold the data of the 2 processes.
  - Each block has PID – no need to flush cache in process-switch.
  - The address of the shared data is:
    - » 100 in process A virtual address space.
    - » 200 in process B virtual address space.

# Aliasing (Synonym) Problem in Virtually Addressed Caches (cont.)

- **Consider the following sequence:**
  - Process A runs first and brings the shared data (addr. 100) into the cache.
  - Process A is suspended and process B is scheduled instead.
  - Process B also reads the shared data (addr. 200) into the cache.
  - **At this point we have two different virtual addresses in the cache that are mapped to the same physical address.**
  - Now process B is suspended and process A is resumed.
  - Process A modifies its shared data (addr. 100) – the data is updated in addr. 100 in the cache (and also in main memory if cache is WT), but it is not updated in addr. 200.
  - A is now suspended and B is resumed.
  - B wants to read the new data in addr. 200 but gets the old value instead of the most recent one (updated by process A).

# Solutions to Aliasing

- **Extra HW that guarantees that every cache block has a unique main-memory physical address**
- **Page coloring:**
  - Use a direct mapped cache +
  - SW/OS always allocates memory for shared data such that they are mapped to the same set in the cache (lower n bits covering the index field must be the same).
  - Since the cache is direct mapped and all shared addresses have the same index field, uniqueness is guaranteed.
  - In the example, the two blocks would bump each other out of the cache:
    - » correct
    - » sub-optimal

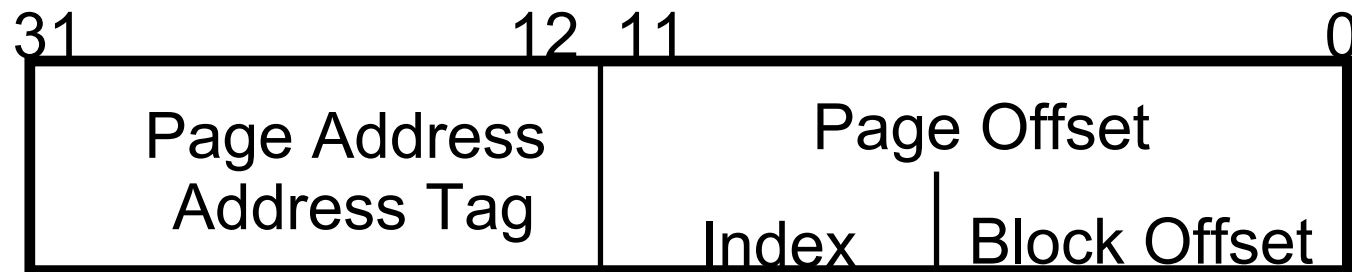


# Physically Addressed Cache implications

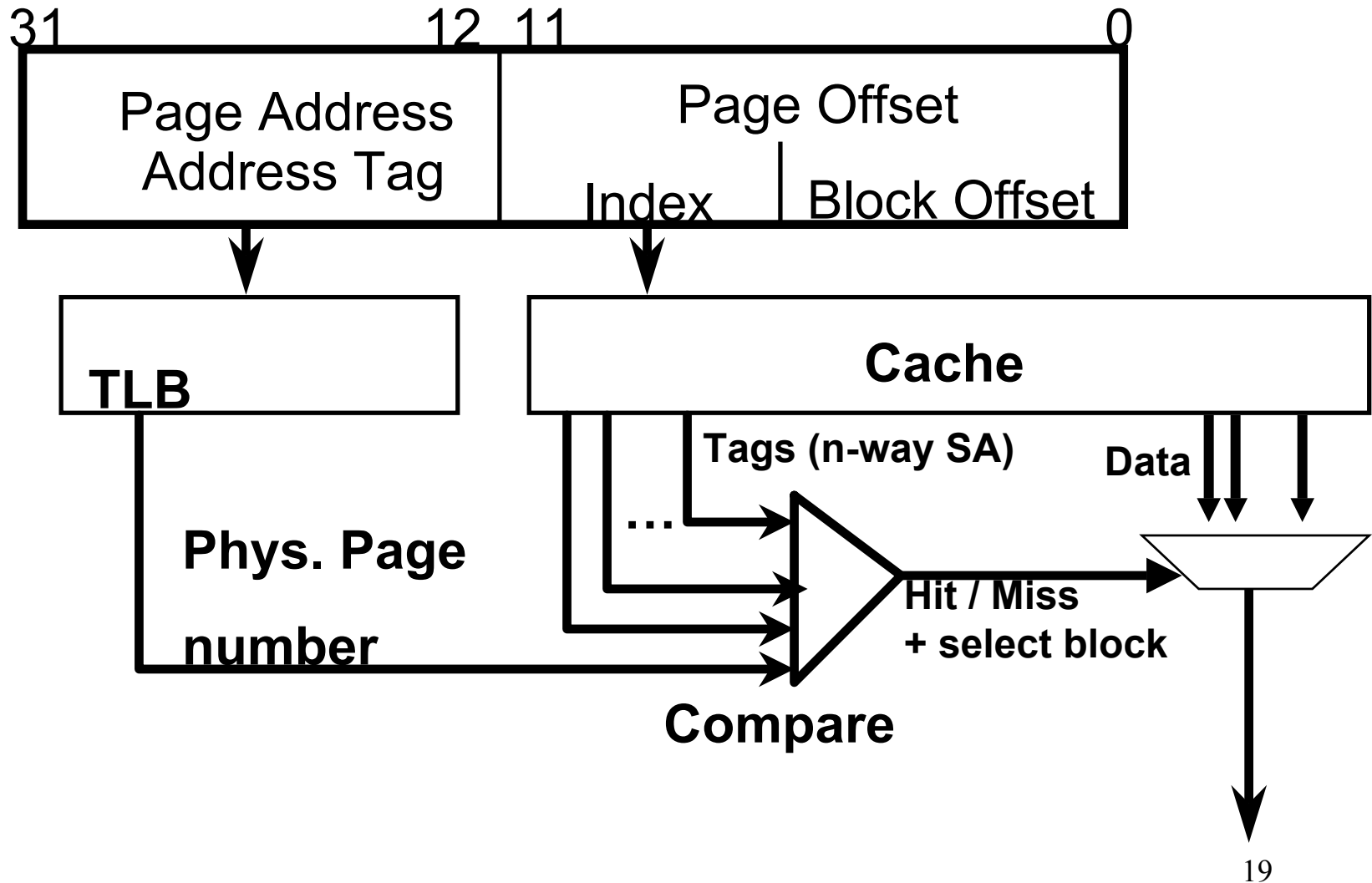
- No need to flush cache content on process-switch.
- Need to flush page data from the cache in page faults.
- No aliasing problems.
- No need for PID field.
- Effective Hit time = TLB hit time + (TLB miss rate)\*(Page table translation time) + Cache hit time

# Virtually Indexed / Physically Tagged Cache

- Virtual address issued by the CPU consists of two parts:
  - Page offset – not translated.
  - Virtual Page Address – needs translation.
- If the cache index is taken from the address part that is not translated (page offset), then we can start reading Tag and data in parallel with the TLB translation.
- By the time the translation in TLB is over, we have the tags ready for comparison.
- Time:  $\max(\text{TLB translation time, and cache access time})$  instead of their sum.



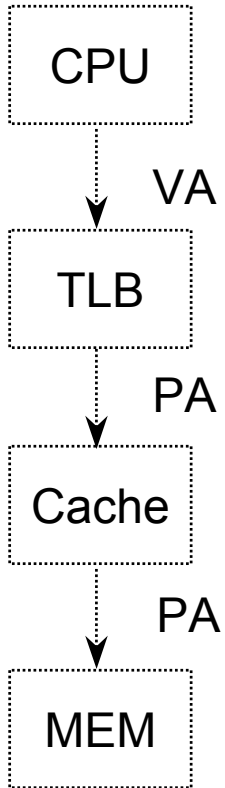
# Virtually Indexed / Physically Tagged Cache



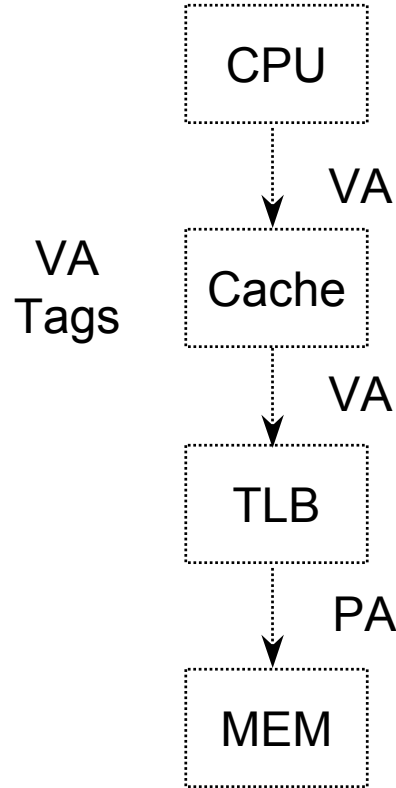
# Virtually Indexed / Physically Tagged Cache: Page-Size Requirement

- **Page offset  $\geq$  Index + block offset**
- **$2^{\text{Page offset}} \geq 2^{\text{Index}} \times 2^{\text{block offset}}$**
- **Page size  $\geq$  number of sets  $\times$  block size**
- **Number of sets = Cache size / (block size  $\times$  Associativity)**
- **Page size  $\geq$  Cache size / Associativity**
- **Requirement:**  
**Page size \* Cache Associativity  $\geq$  Cache size**

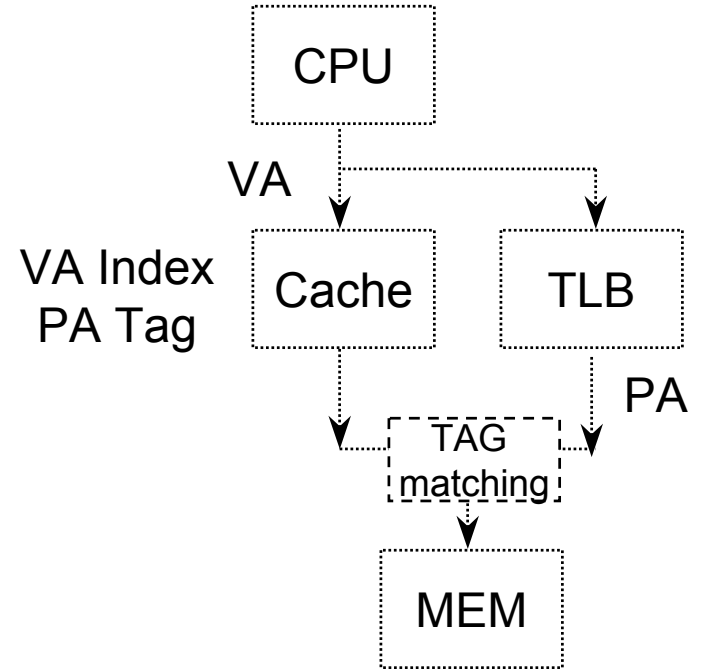
# Cache Addressing Summary



Physically Addressed  
Cache

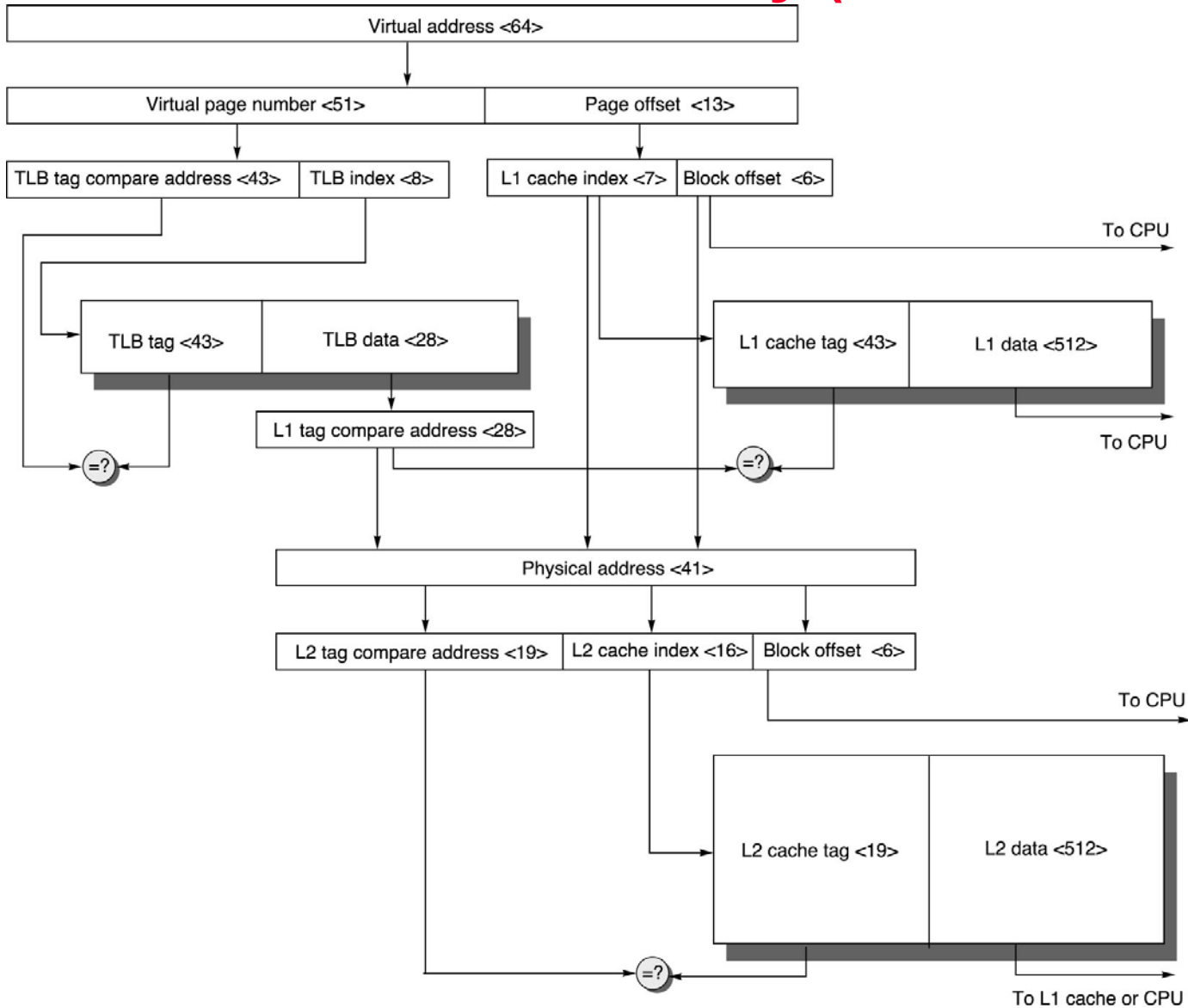


Virtually Addressed  
Cache



Virtually indexed /  
Physically tagged  
cache

# Overall Mem Hierarchy (H&P3e 5.37)



# Alpha 21064

- Separate Instr & Data TLB & Caches
- TLBs fully associative
- TLB updates in SW (“Priv Arch Lib”)
- Caches 8KB direct mapped
- Critical 8 bytes first
- Prefetch instr. stream buffer
- 2 MB L2 cache, direct mapped (off-chip)
- 256 bit path to main memory, 4 x 64-bit modules

