COS 318: Operating Systems

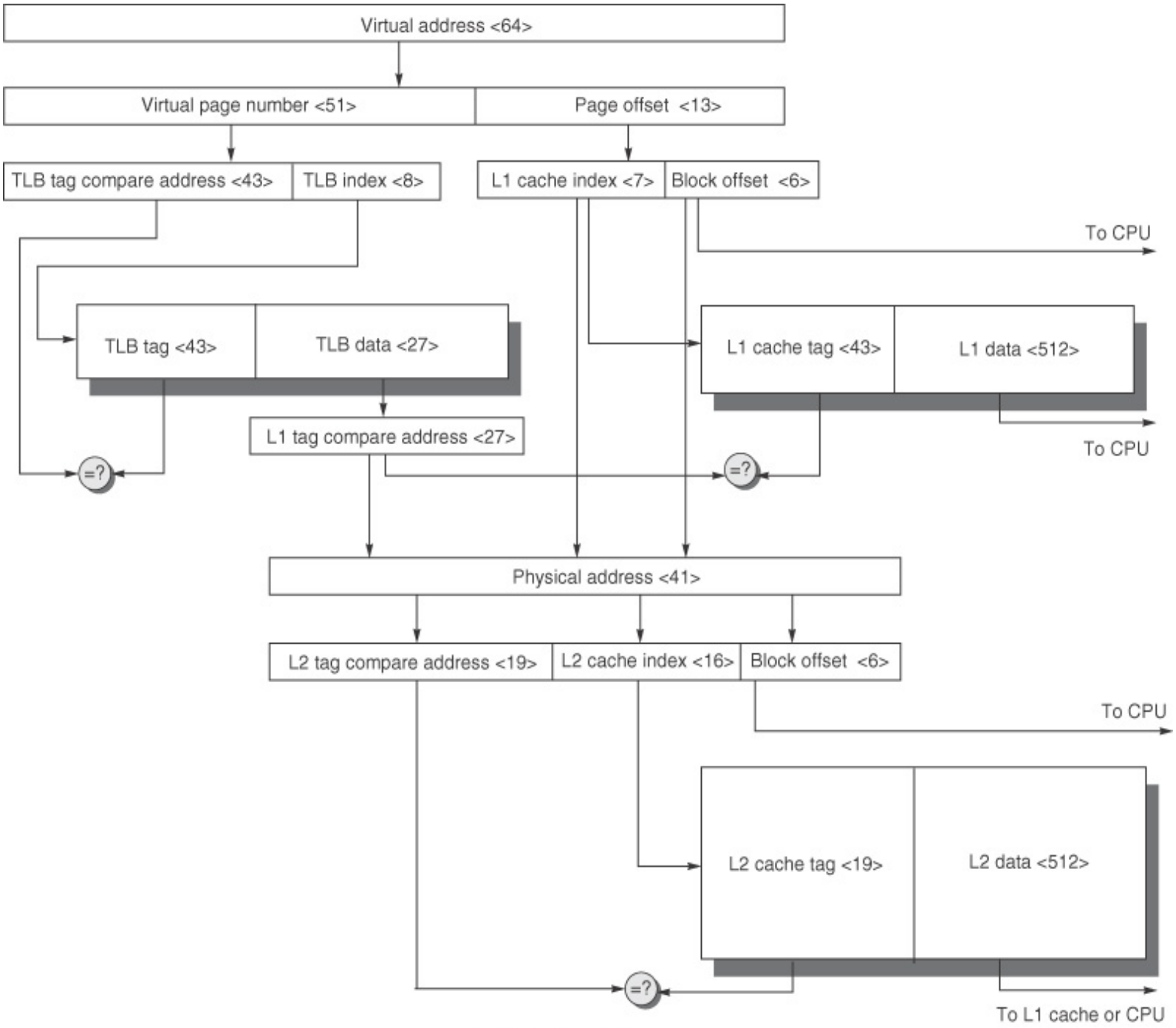
Virtual Memory Paging

Prof. Margaret Martonosi
Computer Science Department
Princeton University

http://www.cs.princeton.edu/courses/archive/fall11/cos318/

Today's Topics

- Paging mechanism
- Page replacement algorithms

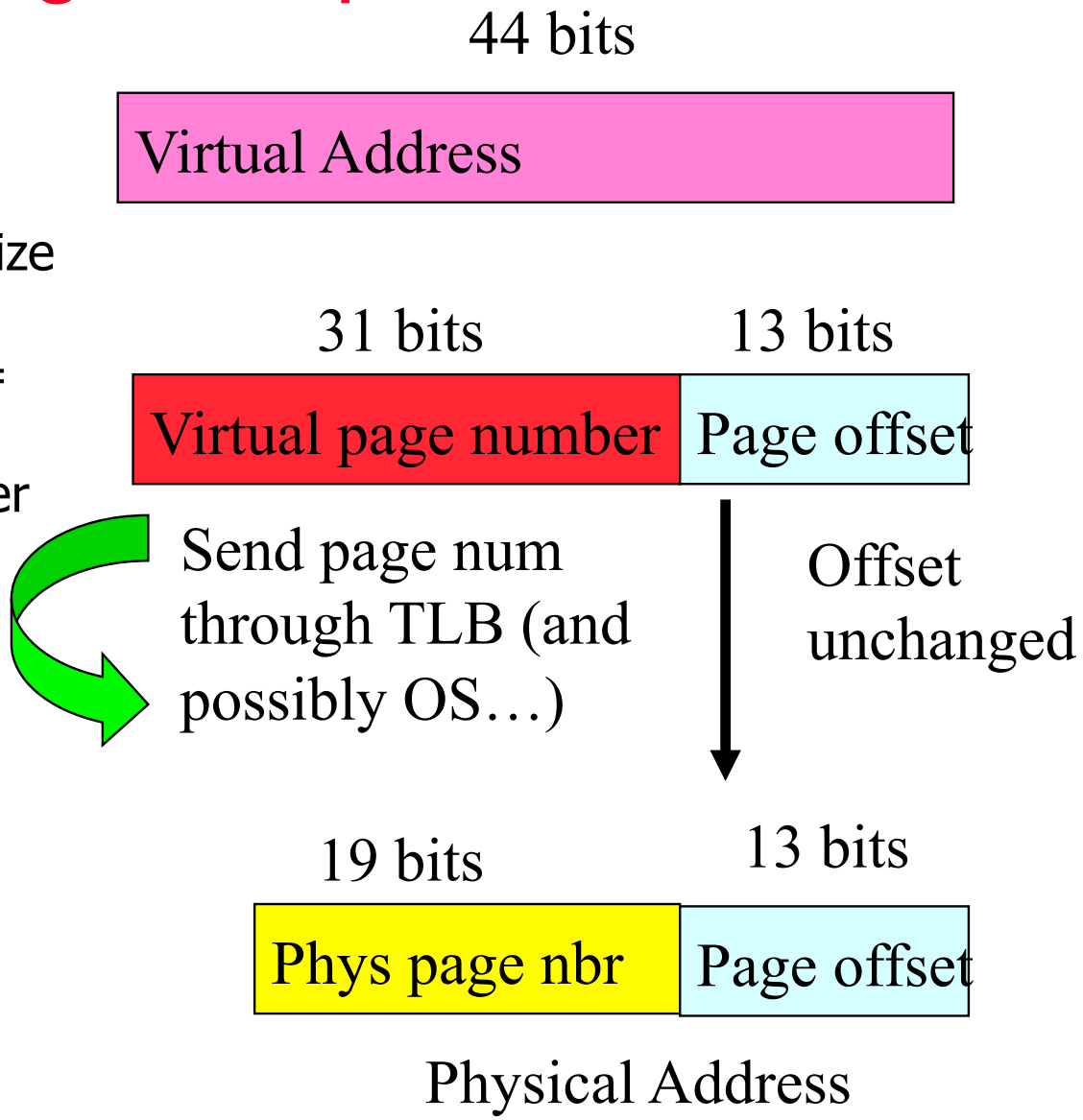


A long example…

- The facts…
- OS & Addressing
  - OS page size = 8K
  - Virtual address space size = 44 bits Virtaddr
  - Physical address limit = 4GB = 32 bits
- Translation Lookaside Buffer (TLB)
  - 64 entries
  - 16-way associativity
- L1 Data Cache
  - Parallel TLB & Cache lookup
  - 32KB (what constrains this?)
  - 4way set associative
  - 64 byte lines

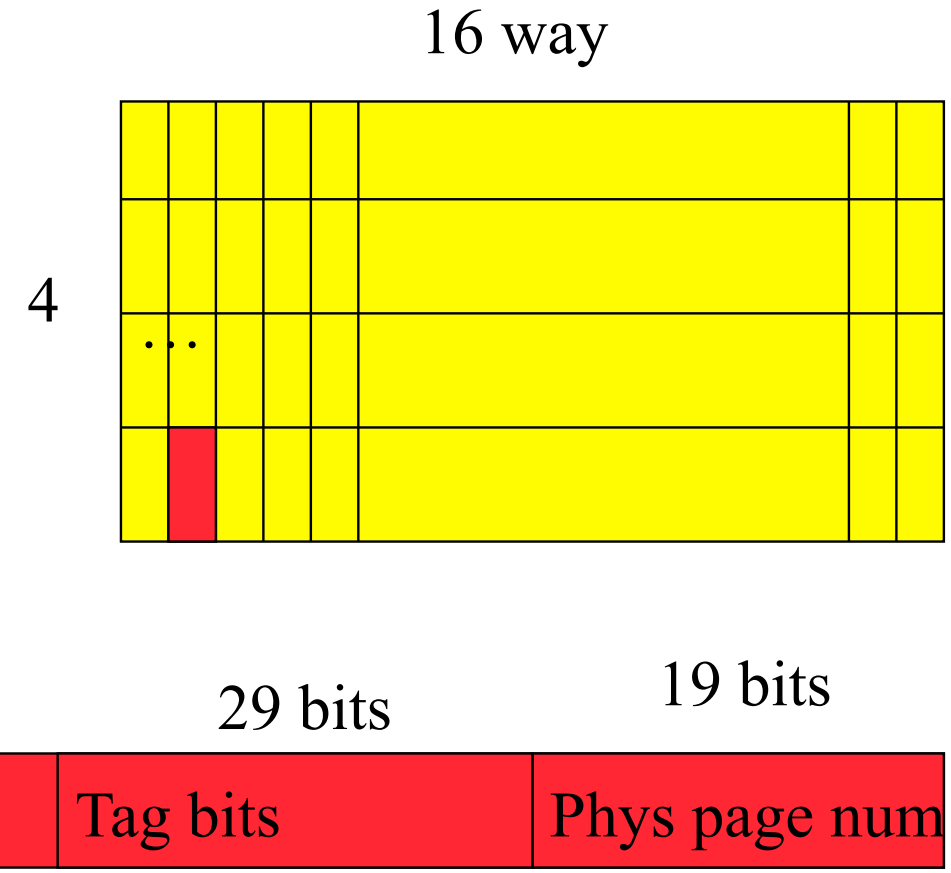
A long example…

- The facts…
- OS & Addressing
  - OS page size = 8K
  - Virtual address space size = 44 bits Virtaddr
  - Physical address limit = 4GB = 32 bits
- Translation Lookaside Buffer (TLB)
  - 64 entries
  - 16-way associativity
- L1 Data Cache
  - Parallel TLB & Cache lookup
  - 32KB (what constrains this?)
  - 4way set associative
  - 64 byte lines



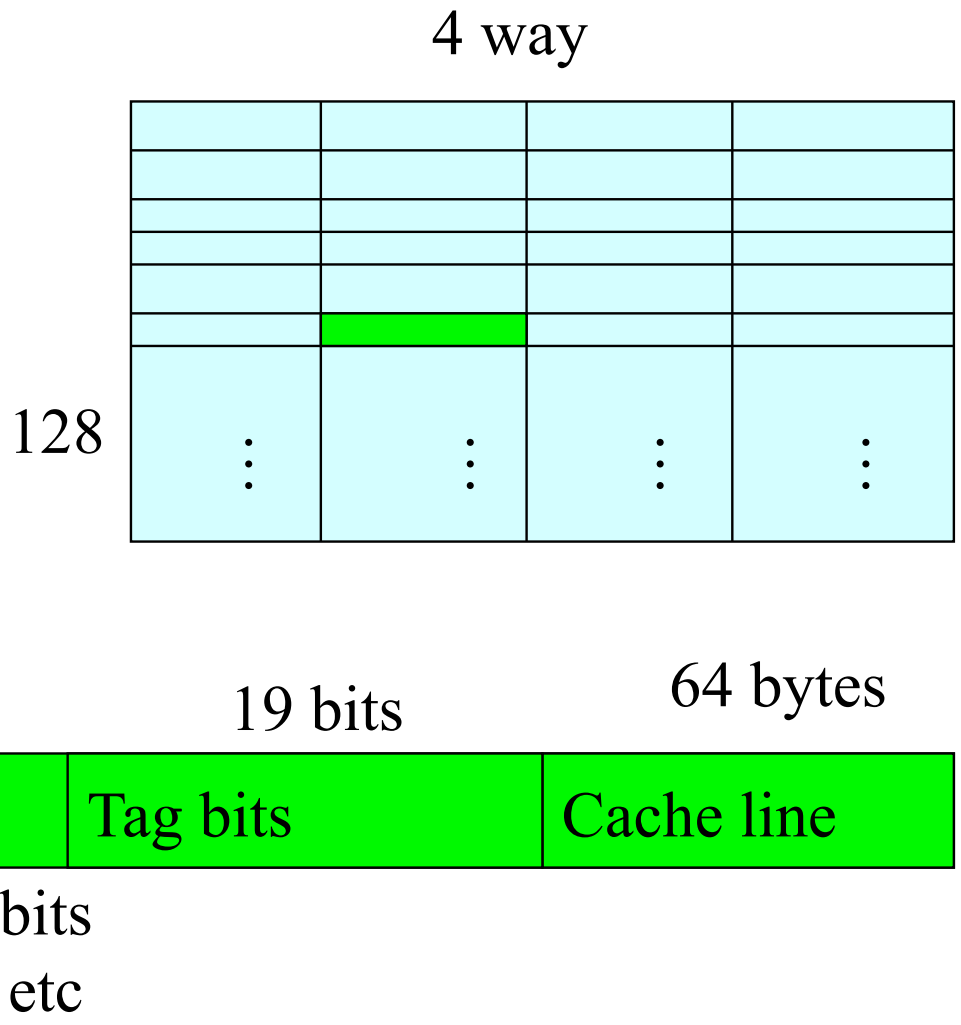
Example step 1: organization of TLB

- It's a 64 entry TLB with 16-way associativity
  - TLB offset=
    - TLBs have no offset! An entry is a page number. You need all of it. Never offset into the middle of it.
  - TLB index= 2 bits
  - TLB tag= everything else
    - = 31 –2 = 29 bits
    - Entry size = size of Phys Page Num = 19 bits



Example step 2: Organization of L1 data cache

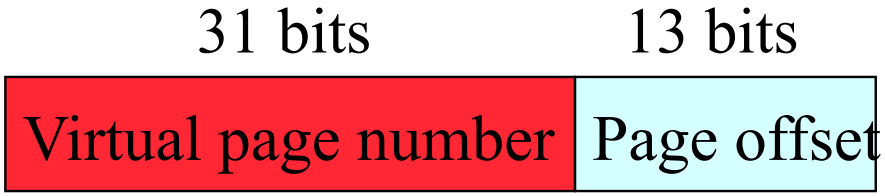
- L1 Data Cache
  - Semi-virtual
  - 32KB
    - what constrains this?
  - 4 way set associative
  - 64 byte lines
- Details:
  - Offset bits = 6 bits
  - Index bits = 7 bits
  - Tag bits = rest = phys address size – 13
  - = 32 –13 = 19 bits



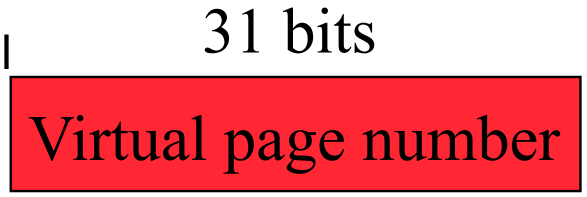
The full path of a data load reference…

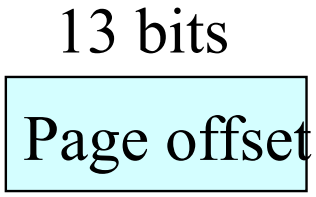
What happens when a processor fetches and executes a load instruction: lw r3, 12(r4)

- Fetch instruction
- Decode instruction
- Fetch register r4. Say the value in r4 is 16000
- Do the address calculation to add 12 to it. We will be fetching from virtual address 16012 (0x3e8c)
- Split 16012 into two parts: Virtual page number and virtual page offset.
  - Say the page size is 8K
  - VPN = 16012 div 8K, V Offset = 16012 mod 8K
  - VPN = 1, Voffset = 7820 (0x1E8C)



Virtual Address





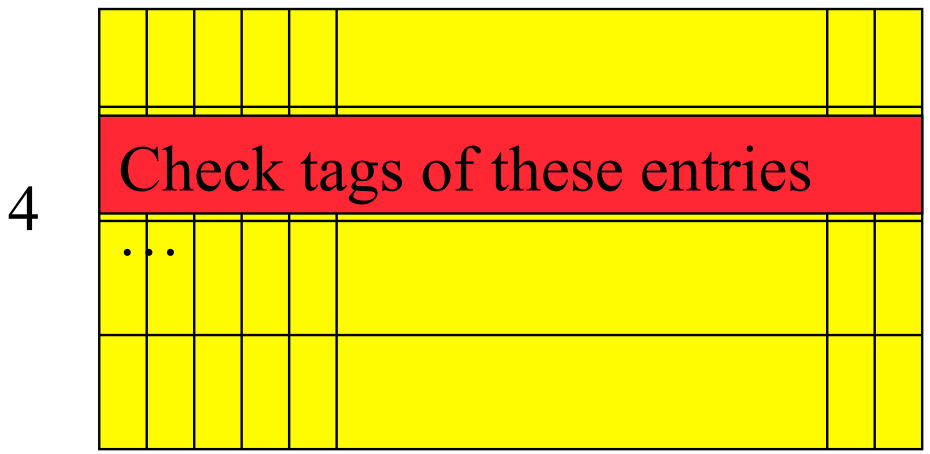
To the TLB for translation

To the cache

Full path of a data load reference, part 2…

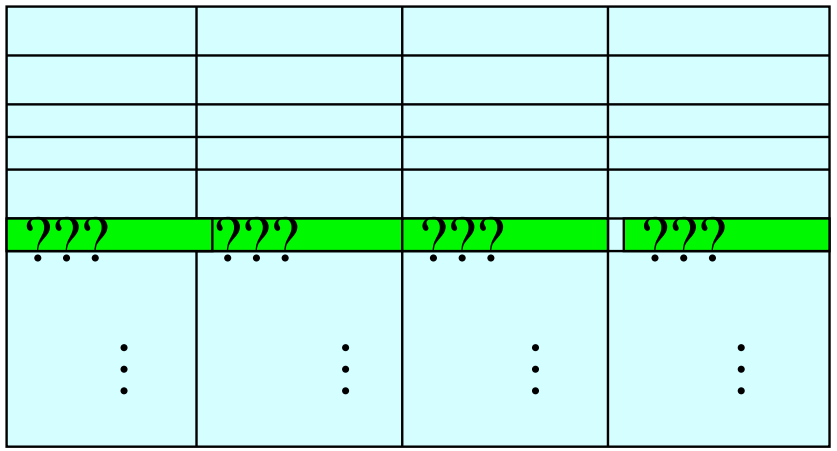
V -> P translation…

- So, VPN = 1…What next?
- Send VPN through translation to get physical page number
- First, try to find the V->P mapping in the TLB



Meanwhile, Data cache lookup…

- Send Page offset to data cache to start cache access
- Page offset = 7820 (0x1E8C)
- L1 Data Cache is 32KB cache, 4-way set associative with 64 byte lines, so…
  index = 61



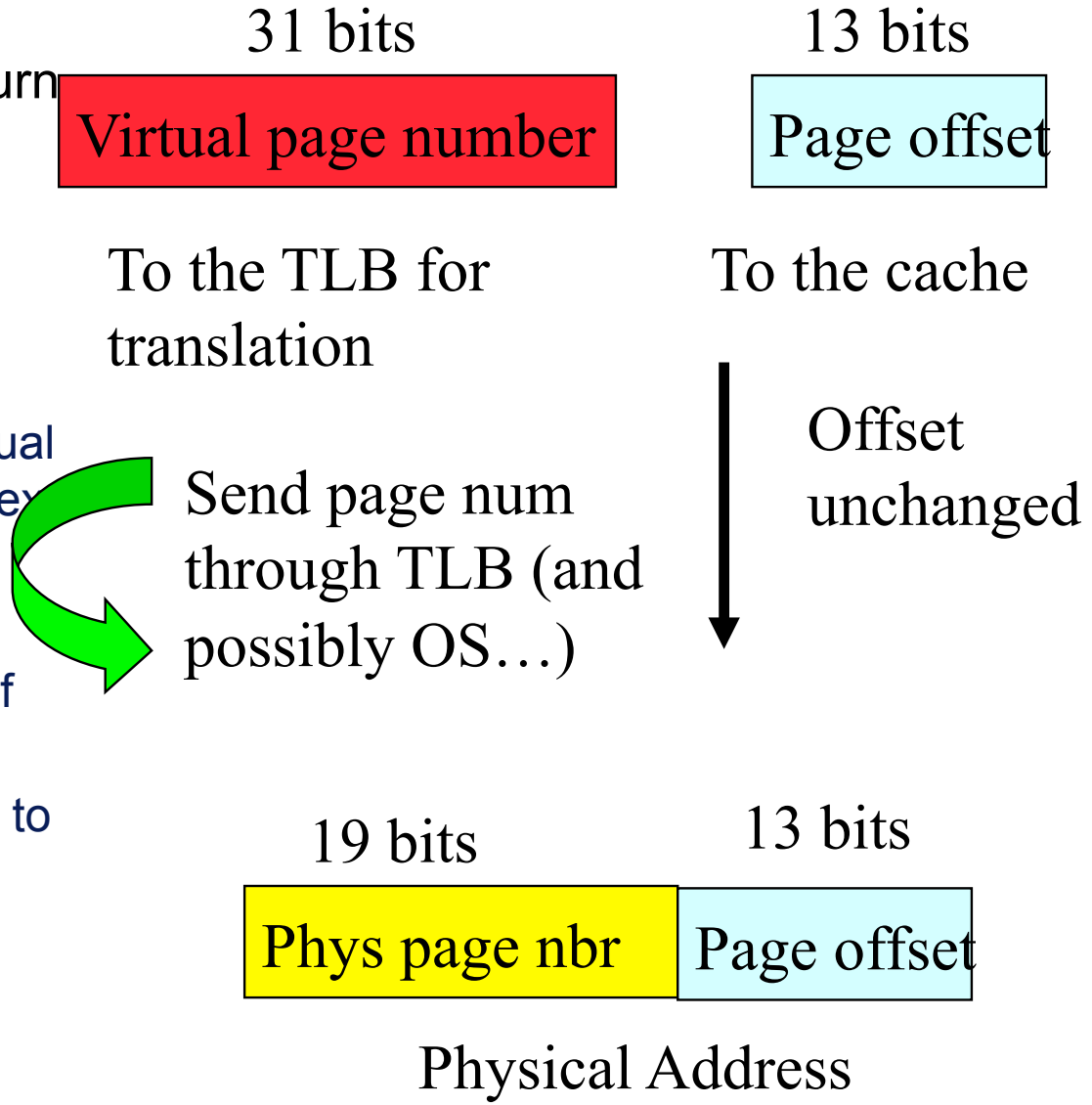
The full path of a data load ref, part 3

V->P translation

If lookup is a hit in the TLB: return PPN to cache controller and continue with Cache lookup sequence

Cache lookup:

- Have used lower portion of virtual address to figure out which index to search. Now what?
- Need tag bits to go any further.
- Tag bits are in the upper part of the address
- Need to wait for V->P mapping to complete.
- If TLB hit, this will be fairly fast.
- If TLB Miss, very slow
- If PPN doesn't arrive within a certain interval, abort and then…



The full path of a data load ref, part 4

V->P translation

- Meanwhile, what's happened to our V->P translation?
- The TLB missed. What's next?

- The TLB is simply a cache of commonly used V->P translations. It is not complete. The OS stores the complete page table (V->P mappings) as a software data structure.
- Assuming this is a machine with a software PT walker, we need to invoke the OS and ask them what the mapping is.
  - Let pipeline drain of any instructions before this load
  - The load cannot complete because we don't know where to reference from.
  - Store away the PC of this load; that's where we'll start from after we ask the OS.

V->P translation, cont'd.

- Enter privileged mode since we want to execute OS code, not user code, now.
- Load the PC with the value of the instruction address for the beginning of the TLB miss handler routine. (software)
- Start executing this software
- It will do a lookup in OS data structures to look for the right mapping.
- Will it find it or not?!

The full path of a data load reference, part 5

V->P translation, cont'd.

- It will do a lookup in OS data structures to look for the right mapping. 2 possible outcomes:
  - If it does NOT FIND a mapping, that means the page is NOT currently in memory. It must be out on disk. Initiate a disk transfer…
  - Else, if it finds mapping, the page is currently in main memory. Just give TLB the info
- Once OS finds the correct mapping, it executes instructions that load the right TLB entry with the PPN info.

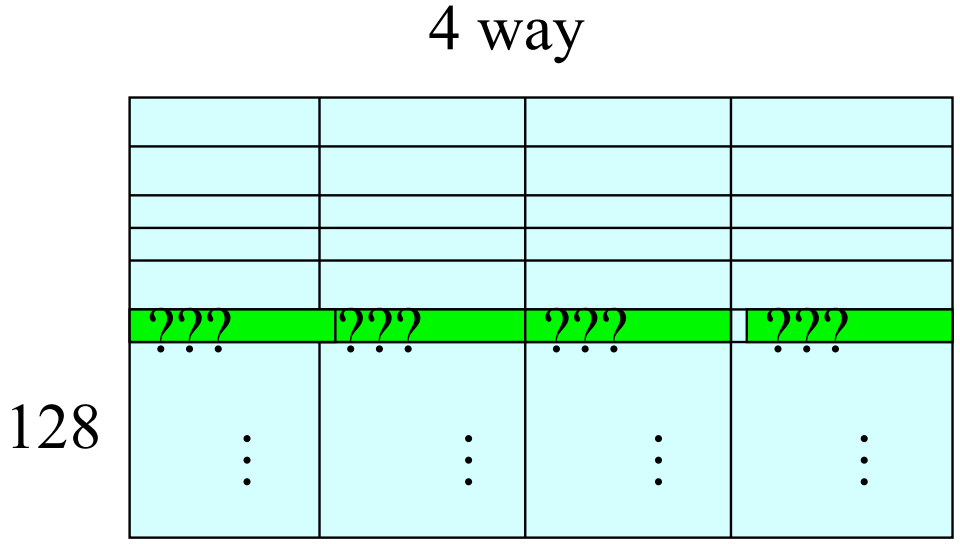
V->P translation, cont'd.

- Leave privileged mode
- Restore context of user program
- Restore program PC to offending load instruction.
- Rerun instruction…
  - Fetch
  - Decode
  - Address calculation
  - Mem access
    - This time, TLB access will be a hit. V->P mapping succeeds!
    - Pass PPN bits to cache ctller…

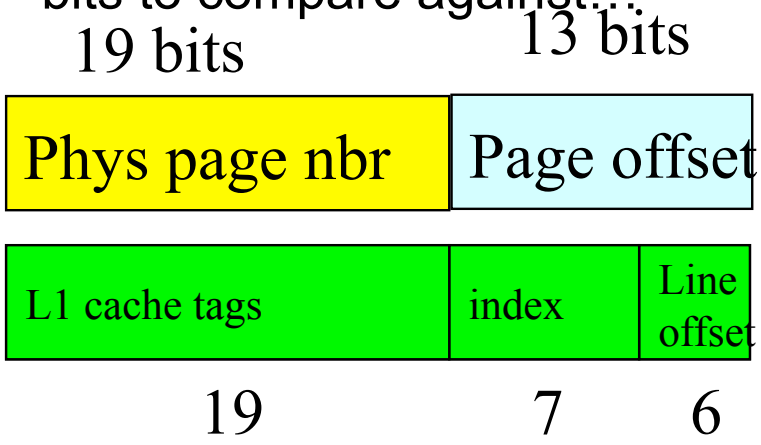
The full path of a data load reference, part 6

Meanwhile, back at the cache…

- Recall that the cache had gotten this far:
  - Have used lower portion of virtual address to figure out which index to search. Now what?
  - Need tag bits to go any further.
  - Tag bits are in the upper part of the address
  - Need to wait for V->P mapping to complete.
  - If TLB hit, this will be fairly fast.
  - If TLB Miss, very slow
  - If PPN doesn't arrive within a certain interval, abort and then…
- This time, instruction is restarted and TLB access succeeds.
  - Q: What does this say about the OS handler code?



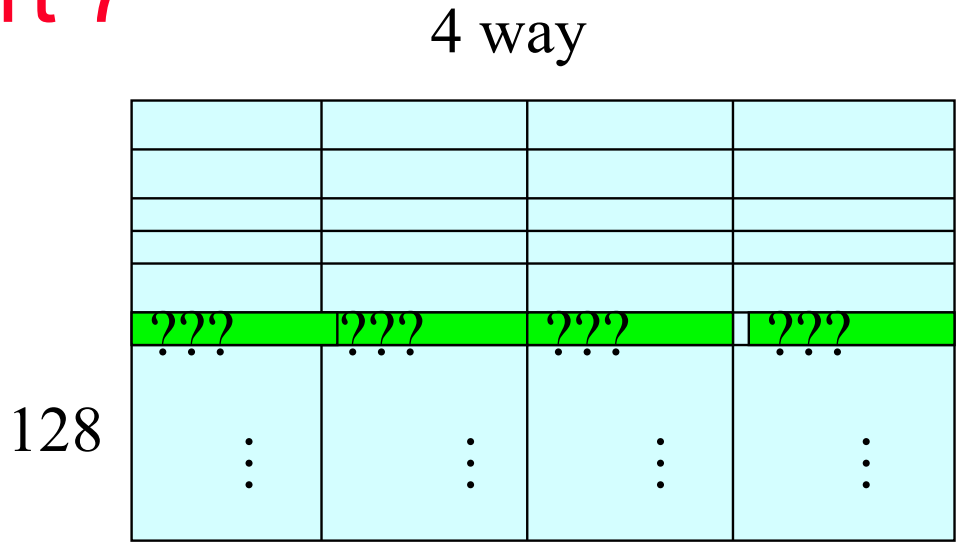
- This time, we have the tag bits to compare against…



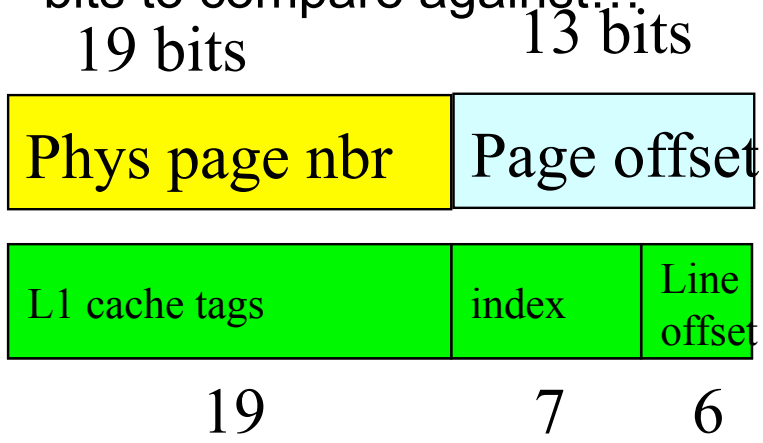
The full path of a load instruction, part 7

Meanwhile, back at the cache…

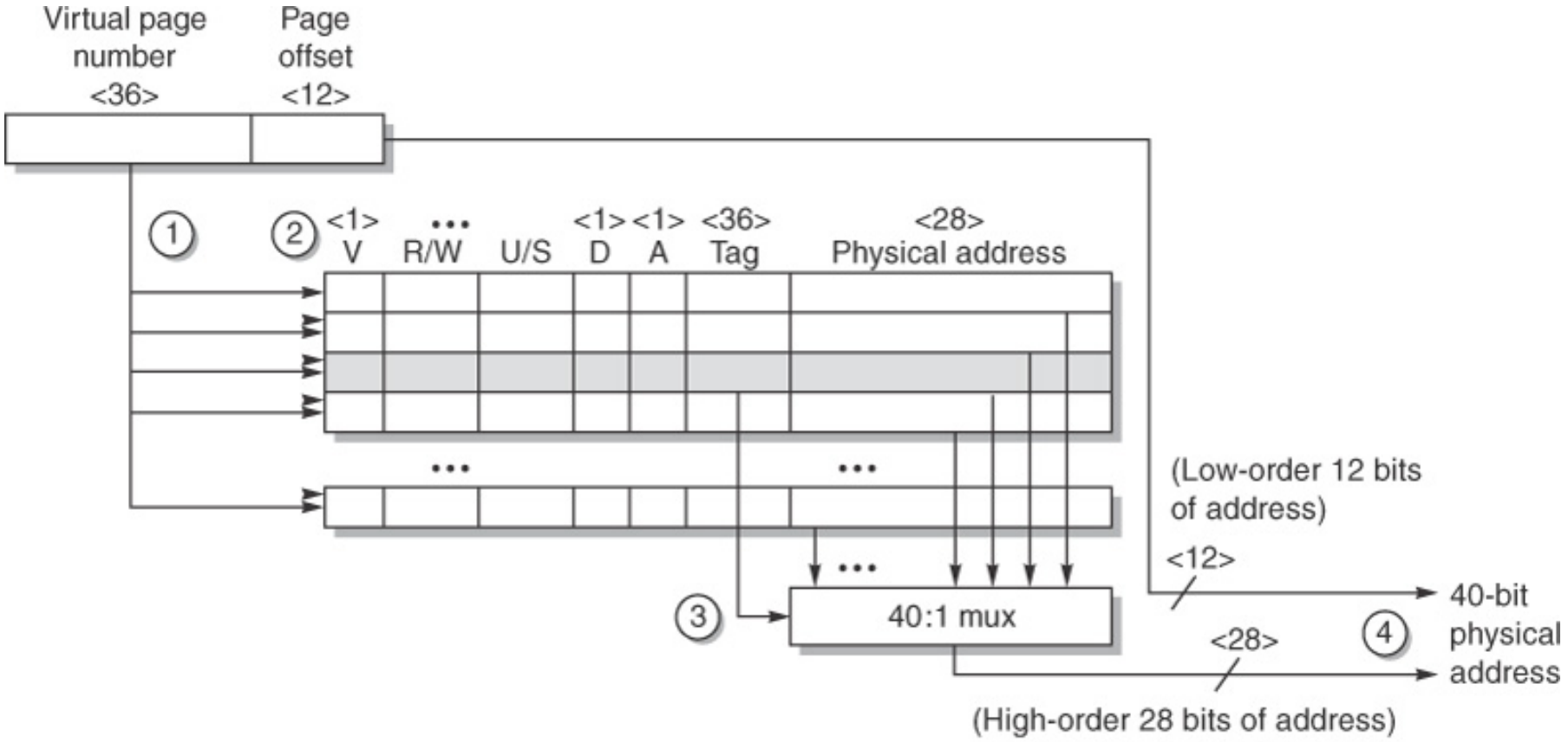
- Compare tag bits from address to tag bits in cache. Find which of 4 sets is match (if any).
- If hit, return data
- If miss……..

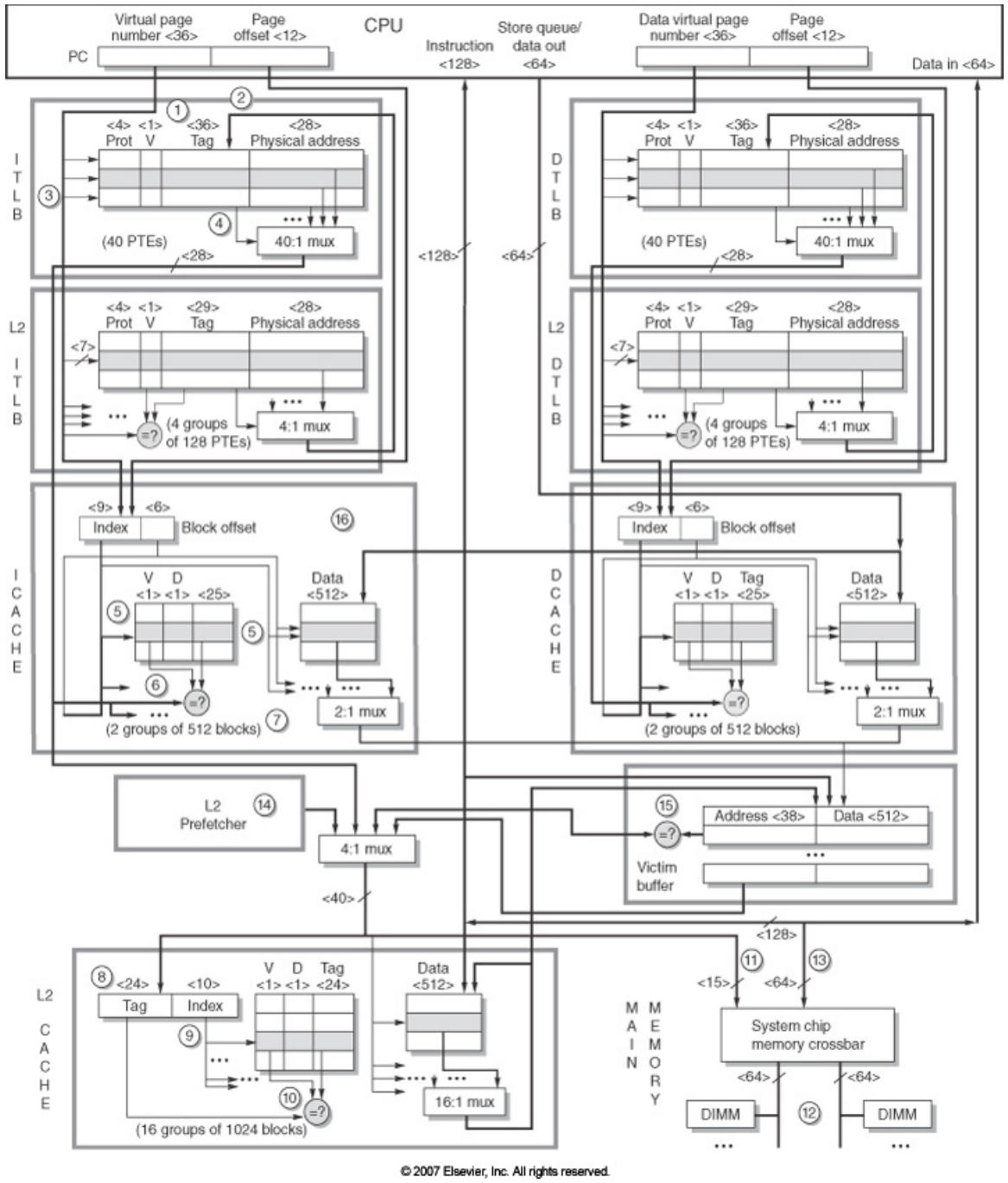


- This time, we have the tag bits to compare against…



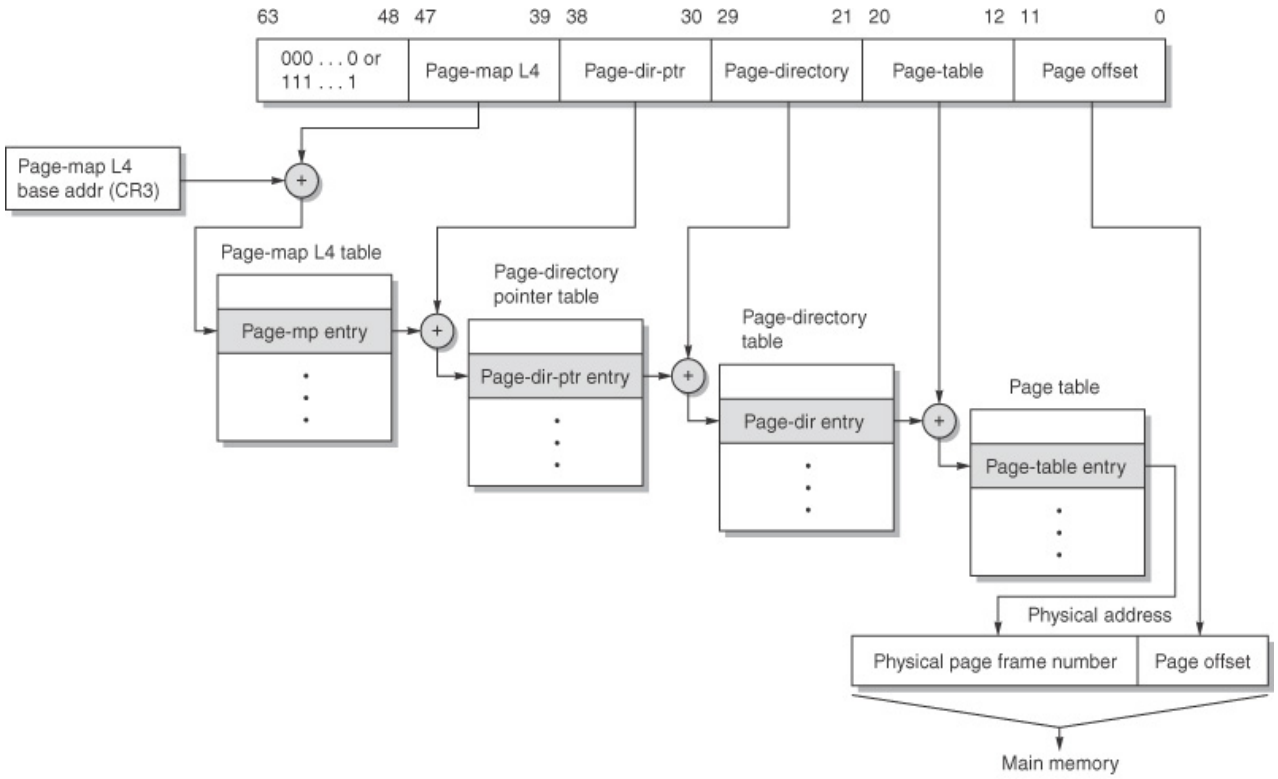
AMD Opteron TLB





More Opteron

AMD Opteron Page Table Organization



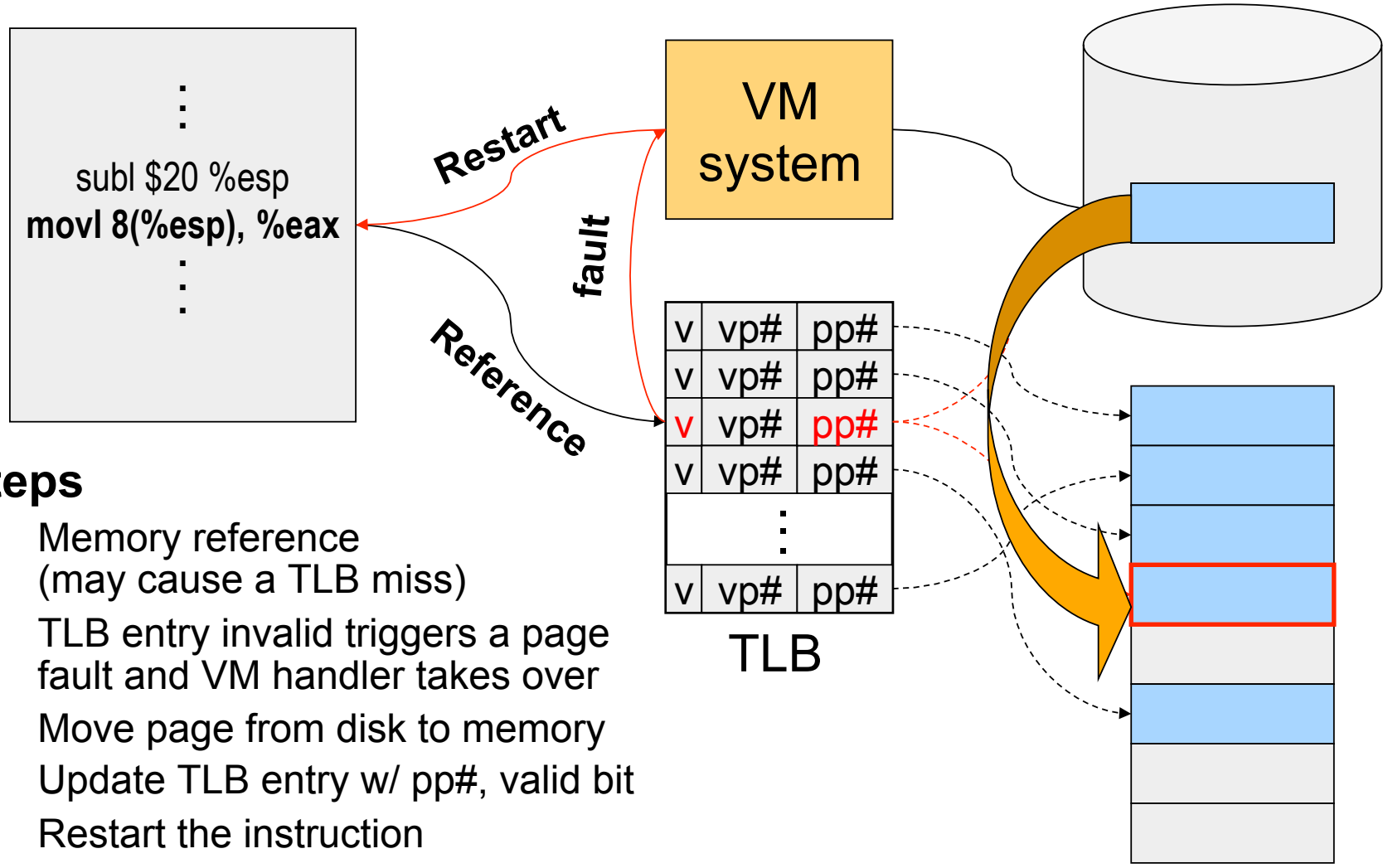
- Sequence of lookups required if TLB misses
- Multi-level page table allows large Virtual Address space to be mapped without devoting lots of space when phys mem requirements are small.

A few more details…

- The OS page table is a large data structure, and a full software traversal is slow. How to optimize?

1) TSB in Software: Software data structure that holds the most commonly used mappings.
   - OS code move mapping entries between it and full page table. “Translation Storage Buffer”. Like a TLB except in software.
   - Goal: TSB entries should at least be hits in L2 cache. 20-50 cycles
2) Hardware Page Table Walks:
   1) When the TLB misses, invoke a **hardware** state machine that traverses the memory that holds the software page table.
   2) Implications: Software page table must have a fixed, unchanging format that hardware knows how to walk.
   3) But this is not free. Still lots of memory fetches to wait for. 50-100 cycles.

VM Paging Steps



Steps

- Memory reference (may cause a TLB miss)
- TLB entry invalid triggers a page fault and VM handler takes over
- Move page from disk to memory
- Update TLB entry w/ pp#, valid bit
- Restart the instruction
- Memory reference again

Policies for Paged Virtual Memory

- The OS tries to minimize page fault costs incurred by all processes, balancing fairness, system throughput, etc.
  - (1) fetch policy: When are pages brought into memory?
    - prepaging: reduce page faults by bring pages in before needed
    - on demand: in direct response to a page fault.
  - (2) replacement policy: How and when does the system select victim pages to be evicted/discarded from memory?
  - (3) placement policy: Where are incoming pages placed? Which frame?
  - (4) backing storage policy:
    - Where does the system store evicted pages?
    - When is the backing storage allocated?
    - When does the system write modified pages to backing store?
    - Clustering: reduce seeks on backing storage

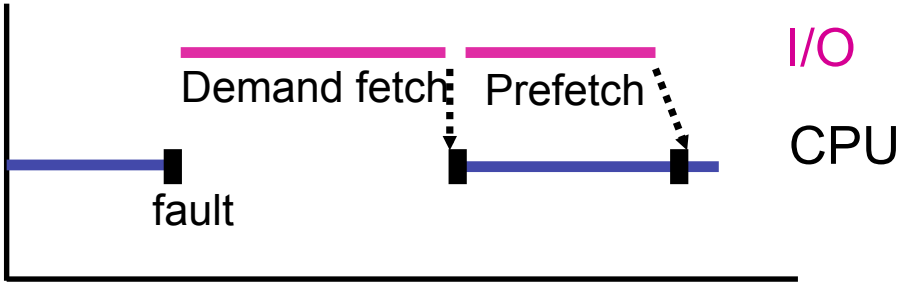
Fetch Policy: Demand Paging

- Missing pages are loaded from disk into memory at time of reference (on demand).
  The alternative would be to prefetch into memory in anticipation of future accesses (need good predictions).
- Page fault occurs because valid bit in page table entry (PTE) is off. The OS:
  - allocates an empty frame*
  - initiates the read of the page from disk
  - updates the PTE when I/O is complete
  - restarts faulting process

* Placement and possible Replacement policies

Prefetching Issues

- Pro: overlap of disk I/O and computation on resident pages. Hides latency of transfer.
  - Need information to guide predictions
- Con: bad predictions
  - Bad choice: a page that will never be referenced.
  - Bad timing: a page that is brought in too soon
- Impacts:
  - taking up a frame that would otherwise be free.
  - (worse) replacing a useful page.
  - extra I/O traffic



Page Replacement Policy

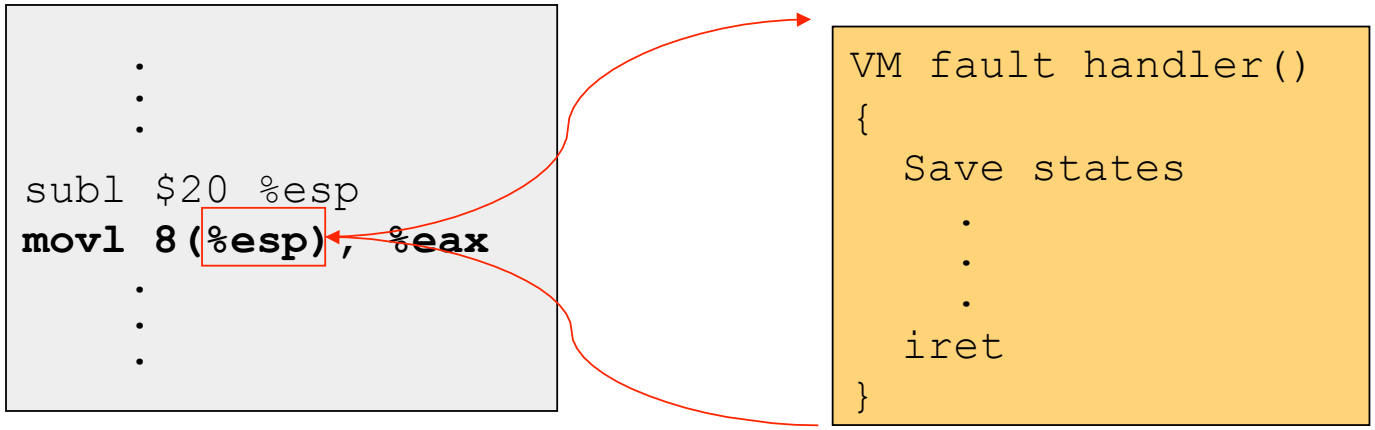
- When there are no free frames available, the OS must replace a page (victim), removing it from memory to reside only on disk (backing store), writing the contents back if they have been modified since fetched (dirty).
- Replacement algorithm - goal to choose the best victim, with the metric for "best" (usually) being to reduce the fault rate.
  - FIFO, LRU, Clock, Working Set…
    (defer to later)

Placement Policy

- Which free frame to chose?
- Are all frames in physical memory created equal?

- Yes, if only considering size. Fixed size.
- No, if considering
  - Cache performance, conflict misses
  - Access to multi-bank memories
  - Multiprocessors with distributed memories

How Does Page Fault Work?



- User program should not be aware of the page fault
- Fault may have happened in the middle of the instruction!
- Can we skip the faulting instruction?
- Is a faulting instruction always restartable?

VM Page Replacement

- Things are not always available when you want them
  - It is possible that no unused page frame is available
  - VM needs to do page replacement
- On a page fault
  - If there is an unused frame, get it
  - **If no unused page frame available,**
    - **Find a used page frame**
    - **If it has been modified, write it to disk**
    - **Invalidate its current PTE and TLB entry**

    Page Replacement
  - Load the new page from disk
  - Update the faulting PTE and remove its TLB entry
  - Restart the faulting instruction
- General data structures
  - A list of unused page frames
  - A table to map page frames to PID and virtual pages, why?

Which “Used” Page Frame To Replace?

- Random
- Optimal or MIN algorithm
- NRU (Not Recently Used)
- FIFO (First-In-First-Out)
- FIFO with second chance
- Clock
- LRU (Least Recently Used)
- NFU (Not Frequently Used)
- Aging (approximate LRU)
- Working Set
- WSClock

Optimal or MIN

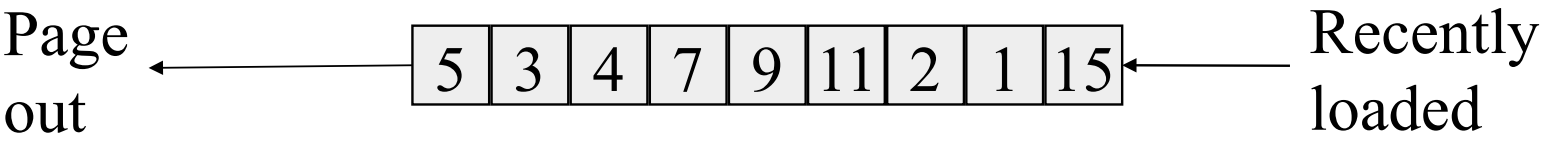
- Algorithm:
  - Replace the page that won't be used for the longest time (Know all references in the future)
- Example
  - Reference string: 1 2 3 4 1 2 5 1 2 3 4 5
  - 4 page frames
  - 6 faults
- Pros
  - Optimal solution and can be used as an off-line analysis method
- Cons
  - No on-line implementation

Not Recently Used (NRU)

- Algorithm
  - Randomly pick a page from the following (in this order)
    - Not referenced and not modified
    - Not referenced and modified
    - Referenced and not modified
    - Referenced and modified
  - Clear reference bits
- Example
  - 4 page frames
  - Reference string
  - 8 page faults

- Pros
  - Implementable
- Cons
  - Require scanning through reference bits and modified bits

First-In-First-Out (FIFO)



- Algorithm
  - Throw out the oldest page
- Example
  - 4 page frames
  - Reference string
  - 10 page faults

- Pros
  - Low-overhead implementation
- Cons
  - May replace the heavily used pages

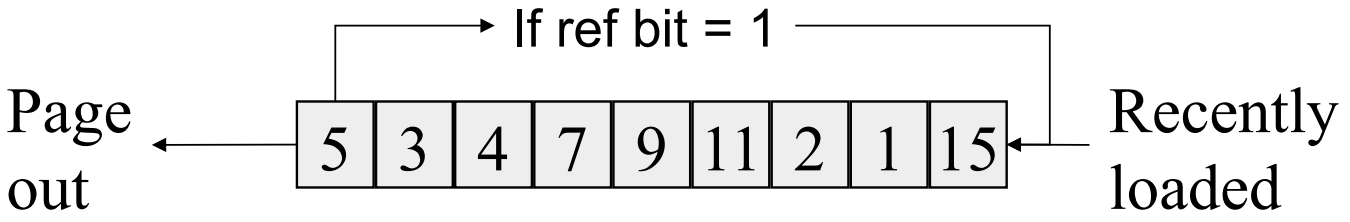
More Frames → Fewer Page Faults?

- Consider the following with 4 page frames
  - Algorithm: FIFO replacement
  - Reference string:
  - 10 page faults

- Same string with 3 page frames
  - Algorithm: FIFO replacement
  - Reference string:
  - **9 page faults!**

- This is so called “Belady’s anomaly” (Belady, Nelson, Shedler 1969)

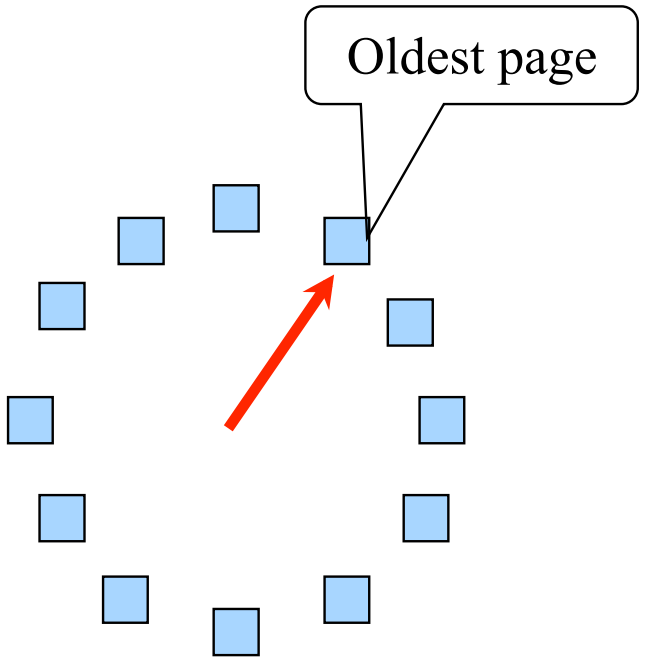
FIFO with 2nd Chance



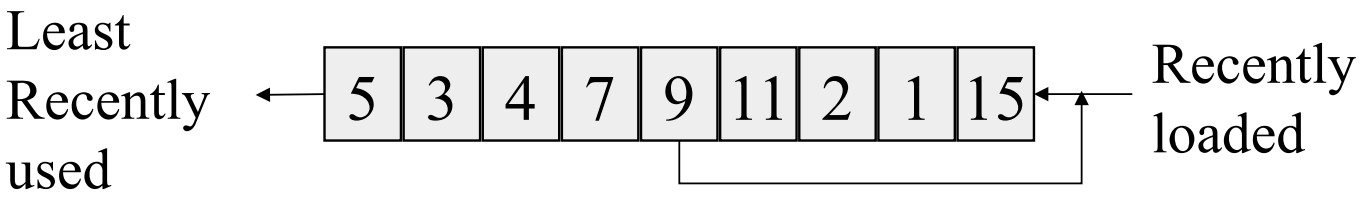
- Algorithm
  - Check the reference-bit of the oldest page
  - If it is 0, then replace it
  - If it is 1, clear the referent-bit, put it to the end of the list,
    and continue searching
- Example
  - 4 page frames
  - Reference string: 1 2 3 4 1 2 5 1 2 3 4 5
  - 8 page faults
- Pros
  - Simple to implement
- Cons
  - The worst case may take a long time

Clock

- FIFO clock algorithm
  - Hand points to the oldest page
  - On a page fault, follow the hand to inspect pages
- Second chance
  - If the reference bit is 1, set it to 0 and advance the hand
  - If the reference bit is 0, use it for replacement
- Compare with the FIFO with 2nd chance
  - What's the difference?
- What if memory is very large
  - Take a long time to go around?



Least Recently Used



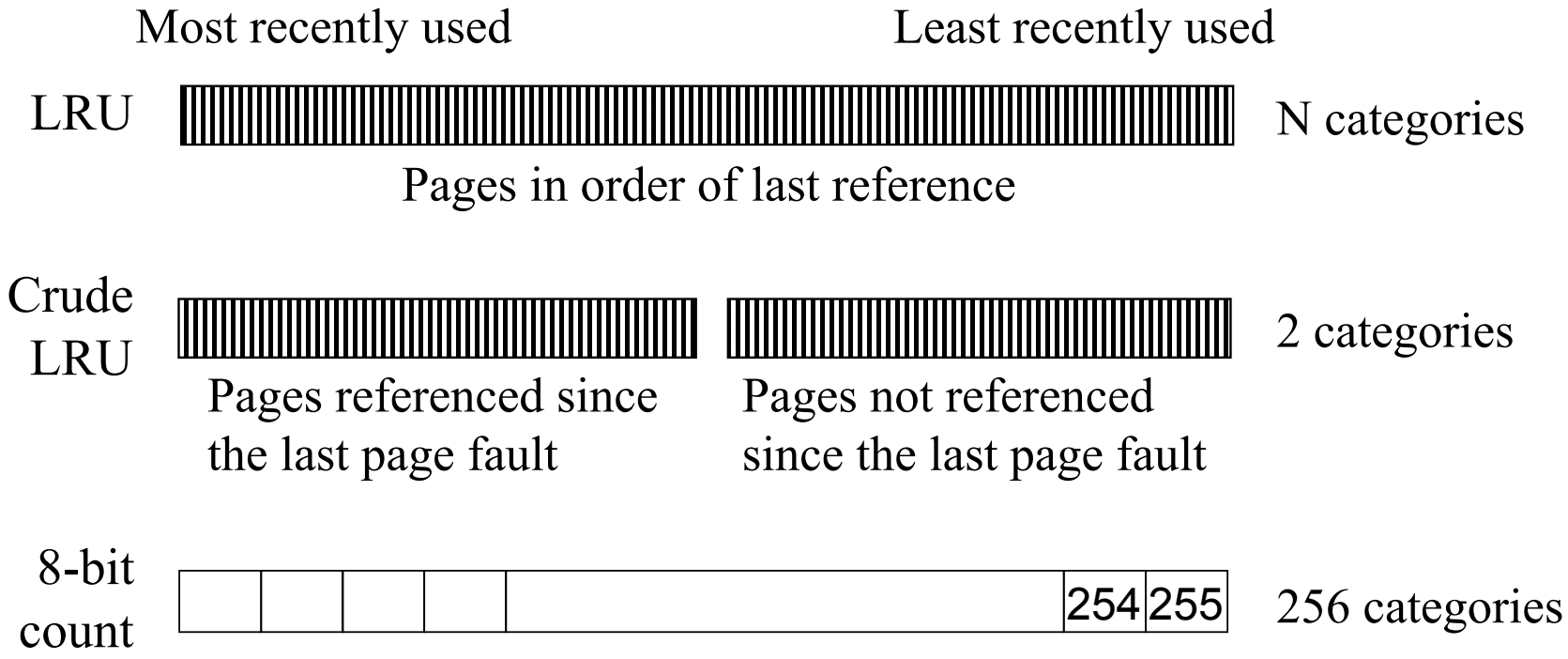
- Algorithm
  - Replace page that hasn't been used for the longest time
    - Order the pages by time of reference
    - Timestamp for each referenced page
- Example
  - 4 page frames
  - Reference string:
  - 8 page faults

1 2 3 4 1 2 5 1 2 3 4 5

- Pros
  - Good to approximate MIN
- Cons
  - Difficult to implement

Approximation of LRU

- Use CPU ticks
  - For each memory reference, store the ticks in its PTE
  - Find the page with minimal ticks value to replace
- Use a smaller counter



Aging: Not Frequently Used (NFU)

- Algorithm
  - Shift reference bits into counters
  - Pick the page with the smallest counter to replace

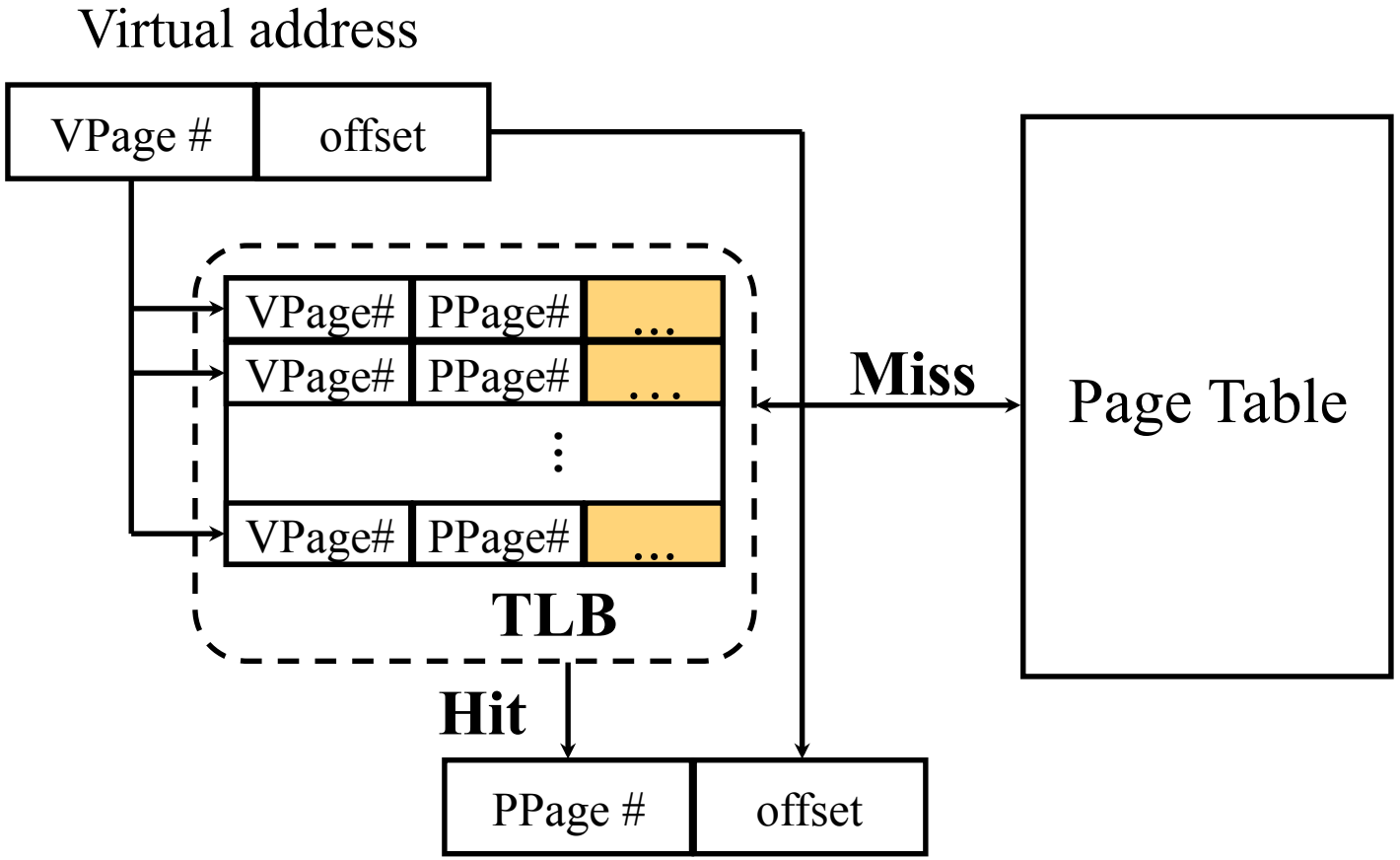
00000000	00000000	10000000	01000000	10100000
00000000	10000000	01000000	10100000	01010000
10000000	11000000	11100000	01110000	00111000
00000000	00000000	00000000	10000000	01000000

- Old example
  - 4 page frames
  - Reference string:
  - 8 page faults

1 2 3 4 1 2 5 1 2 3 4 5

- Main difference between NFU and LRU?
  - NFU has a short history (counter length)
- How many bits are enough?
  - In practice 8 bits are quite good

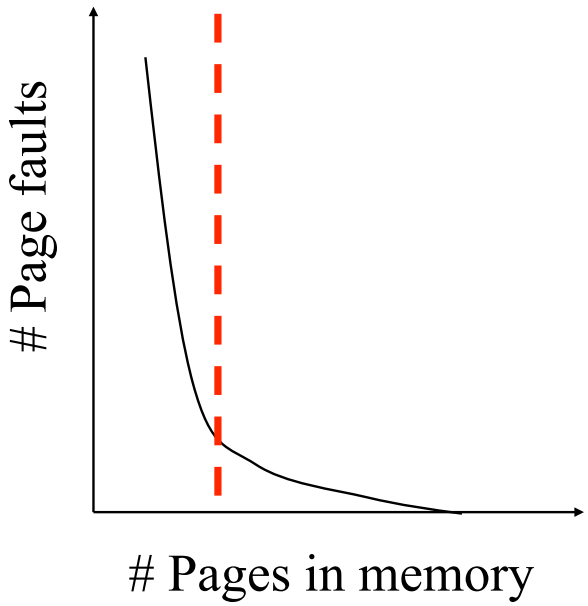
Revisit TLB and Page Table



- Important bits for paging
  - **Reference**: Set when referencing a location in the page
  - **Modify**: Set when writing to a location in the page

Program Behavior (Denning 1968)

- 80/20 rule
  - > 80% memory references are within <20% of memory space
  - > 80% memory references are made by < 20% of code
- Spatial locality
  - Neighbors are likely to be accessed
- Temporal locality
  - The same page is likely to be accessed again in the near future



Working Set

- Main idea (Denning 1968, 1970)
  - Define a working set as the set of pages in the most recent K page references
  - Keep the working set in memory will reduce page faults significantly
- Approximate working set
  - The set of pages of a process used in the last T seconds
- An algorithm
  - On a page fault, scan through all pages of the process
  - If the reference bit is 1, record the current time for the page
  - If the reference bit is 0, check the “time of last use,”
    - If the page has not been used within T, replace the page
    - Otherwise, go to the next
  - Add the faulting page to the working set

WSClock

- Follow the clock hand
- If the reference bit is 1
  - Set reference bit to 0
  - Set the current time for the page
  - Advance the clock hand
- If the reference bit is 0, check “time of last use”
  - If the page has been used within δ, go to the next
  - If the page has not been used within δ and modify bit is 1
    - Schedule the page for page out and go to the next
  - If the page has not been used within δ and modify bit is 0
    - Replace this page

Replacement Algorithms

- The algorithms
  - Optimal or MIN algorithm
  - NRU (Not Recently Used)
  - FIFO (First-In-First-Out)
  - FIFO with second chance
  - Clock
  - LRU (Least Recently Used)
  - NFU (Not Frequently Used)
  - Aging (approximate LRU)
  - Working Set
  - WSClock
- Which are your top two?

Summary

- VM paging
  - Page fault handler
  - What to page in
  - What to page out
- LRU is good but difficult to implement
- Clock (FIFO with 2nd hand) is considered a good practical solution
- Working set concept is important