Topic 13

Memory Hierarchy

- Virtual Memory (2)

A Problem

- Page table is located in main memory
- Address translation requires an extra main memory access
  - One to access the page table
  - Then the actual memory access

Fast Translation Using a TLB

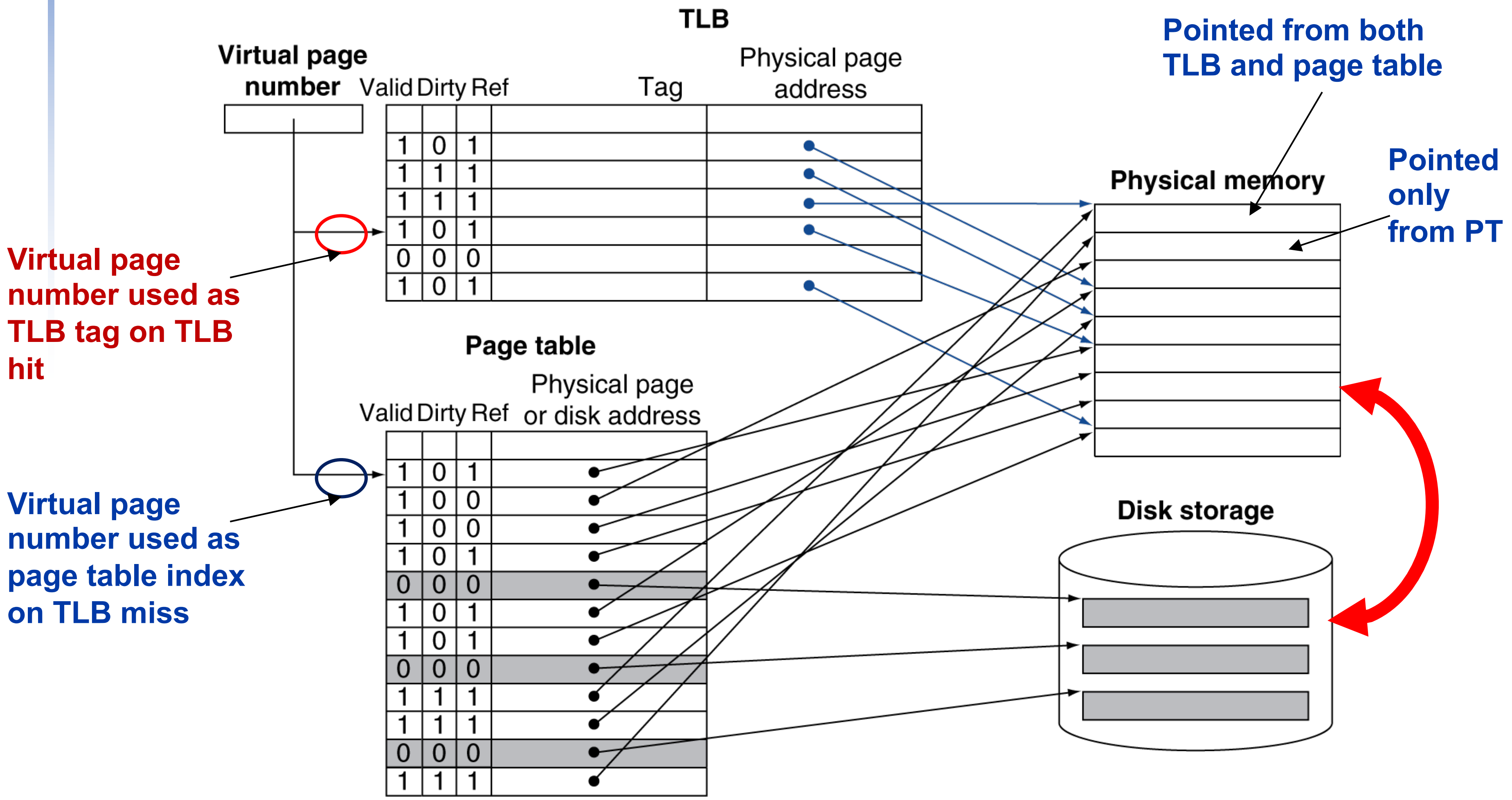
- Access to page tables has good locality
  - So use a fast **cache** of page tables within the CPU to store a subset of the page table
  - Called a **Translation Look-aside Buffer (TLB)**
  - (TLB ← page table) = (cache ← main memory)

TLB

- Load part of the Page Table in cache called TLB
- In TLB, one translation per entry
- Typical: 16–512 entries, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
- Status bits
  - Valid: empty or not
  - Dirty: target memory is changed
  - Reference: target memory is used
- Full Associativity
  - Lower miss rate
  - Small TLB, so access time not a major concern
- LRU or random replacement

Fast Translation Using a TLB

Virtual page number is compared with Tag field of TLB because of full associativity of TLB



TLB
Virtual page number
Valid Dirty Ref
Tag
Physical page address
Pointed from both TLB and page table
Physical memory
Pointed only from PT
Virtual page number used as TLB tag on TLB hit
Page table
Valid Dirty Ref
Physical page or disk address
Disk storage
Virtual page number used as page table index on TLB miss

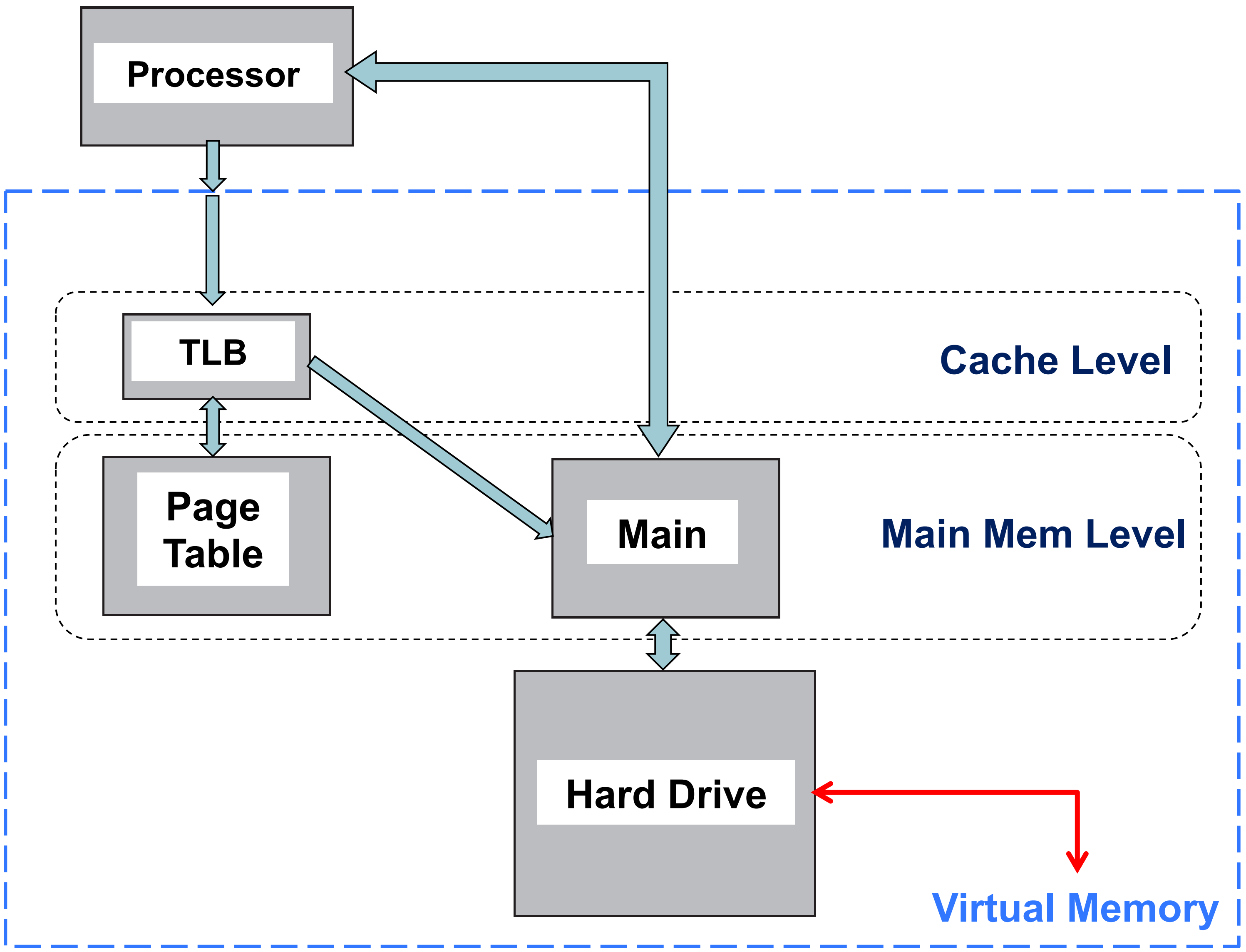
TLB:

Valid	Dirty	Ref
1	0	1
1	1	1
1	1	1
1	0	1
0	0	0
1	0	1

Page table:

Valid	Dirty	Ref
1	0	1
1	0	0
1	0	0
1	0	1
0	0	0
1	0	1
1	0	1
0	0	0
1	1	1
1	1	1
0	0	0
1	1	1

Access Mem through TLB



TLB Hit

- Virtual address found in TLB – hit
  - Provide physical address
  - Reference bit on
  - Dirty bit on if physical address used for write (might not be needed if dirty bit maintained in physical memory)

TLB Miss

- Translation not in TLB – TLB miss
  - If page is in memory (page table valid bit = 1)
    - Load the page table entry from memory to TLB
      - Why not just use page table since it’s accessed anyway?
    - Could be handled in hardware or in software
  - If page is not in memory (page fault)
    - Page fault exception – OS handles fetching the page and updating the page table and TLB
    - Then restart the faulting instruction

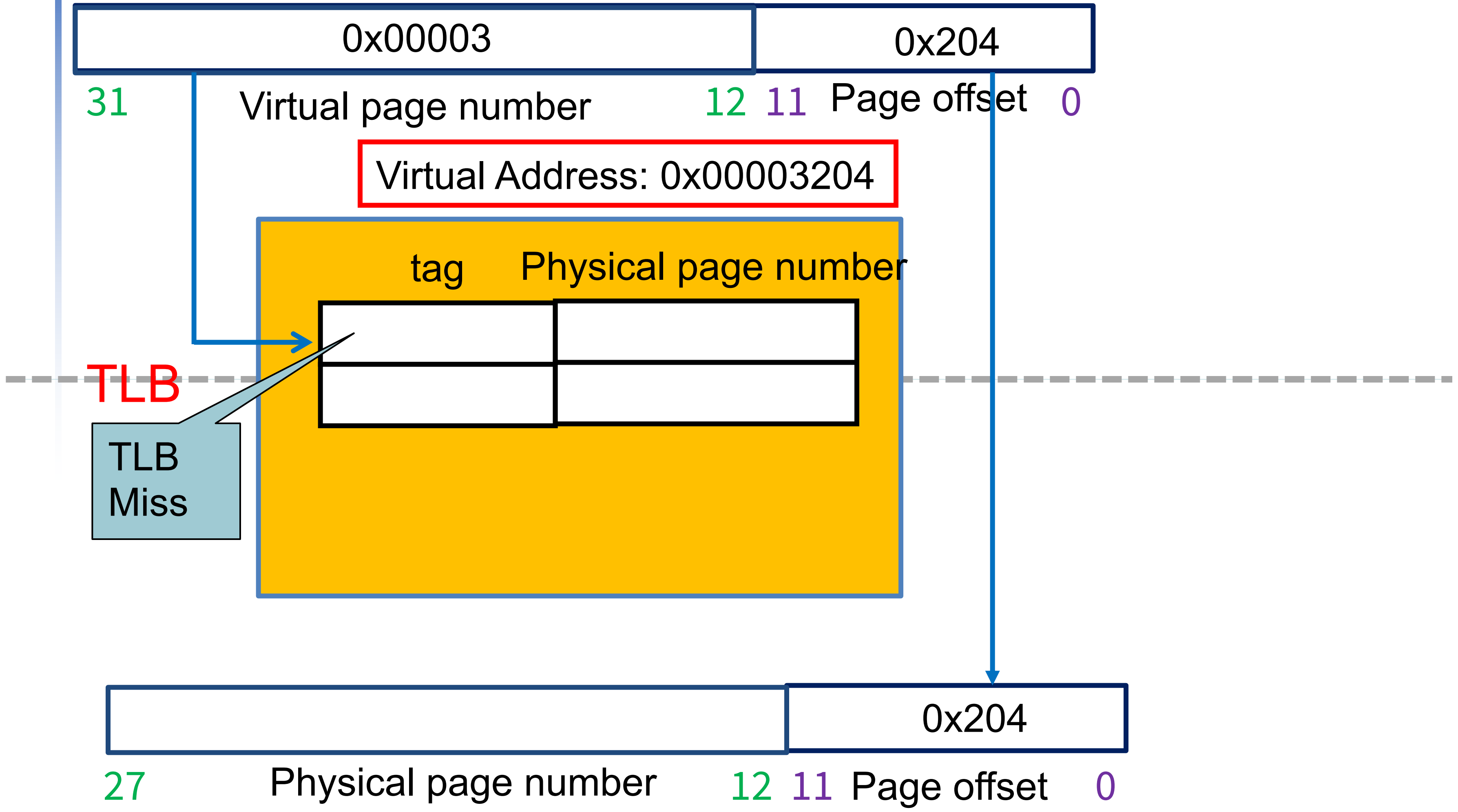
TLB Miss

- Recognize a TLB miss or page fault
  - TLB miss – by TLB tag and TLB valid bit
  - Page fault – by page table valid bit

Handling TLB Miss

- Copy page table entry to TLB
- If TLB is full, replace a TLB entry
  - Ref bit and dirty bit of replaced entry should be copied back to page table
    - These bits need to be copied back to page table only when entry is replaced – write back
  - Not for valid bit – why?
  - Different meaning of Ref bit – why?
  - Other techniques used to keep track of ref/dirty, no need to write back

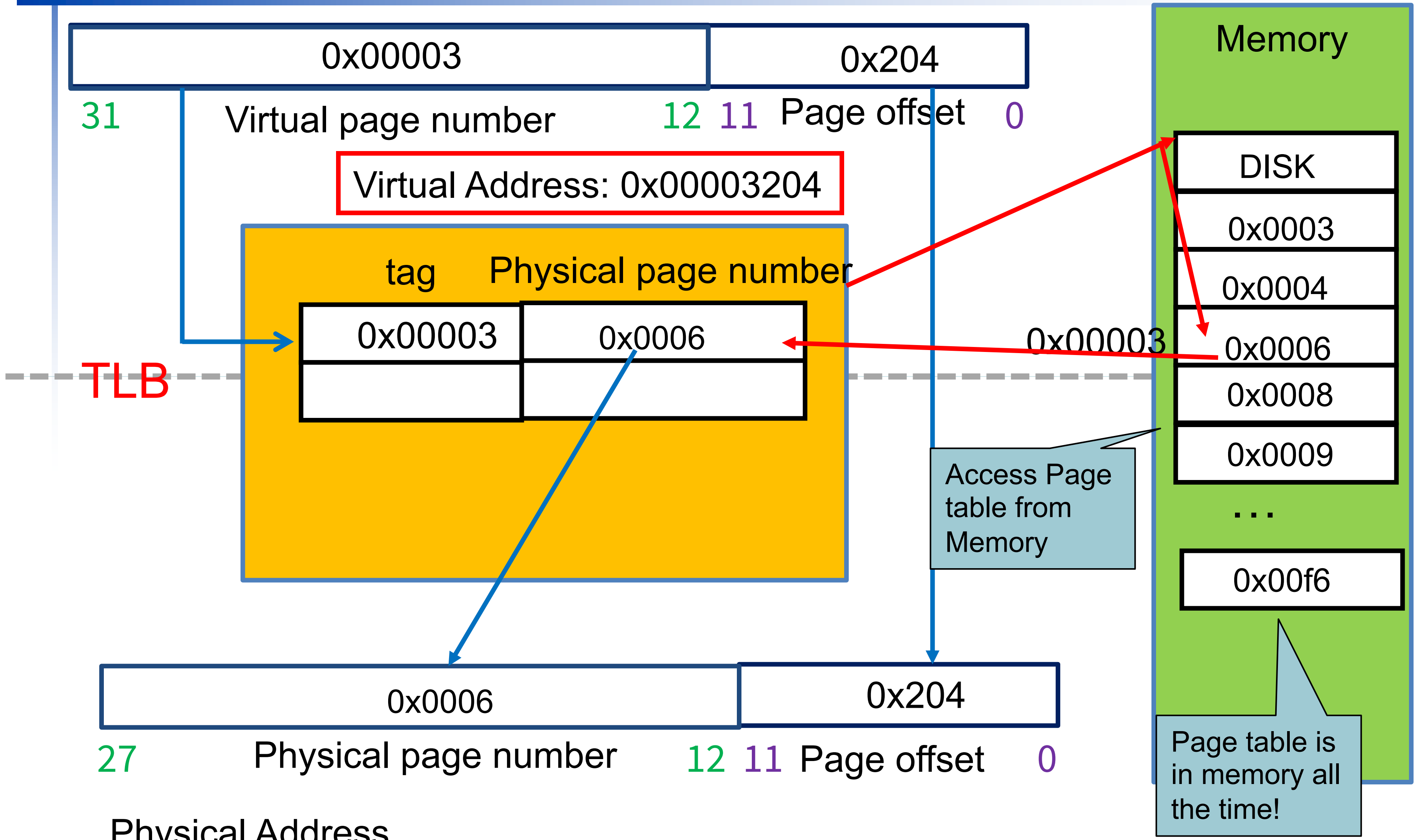
TLB Example (assume 32 bit address)



Physical Address

*Valid, dirty, reference bits omitted here.

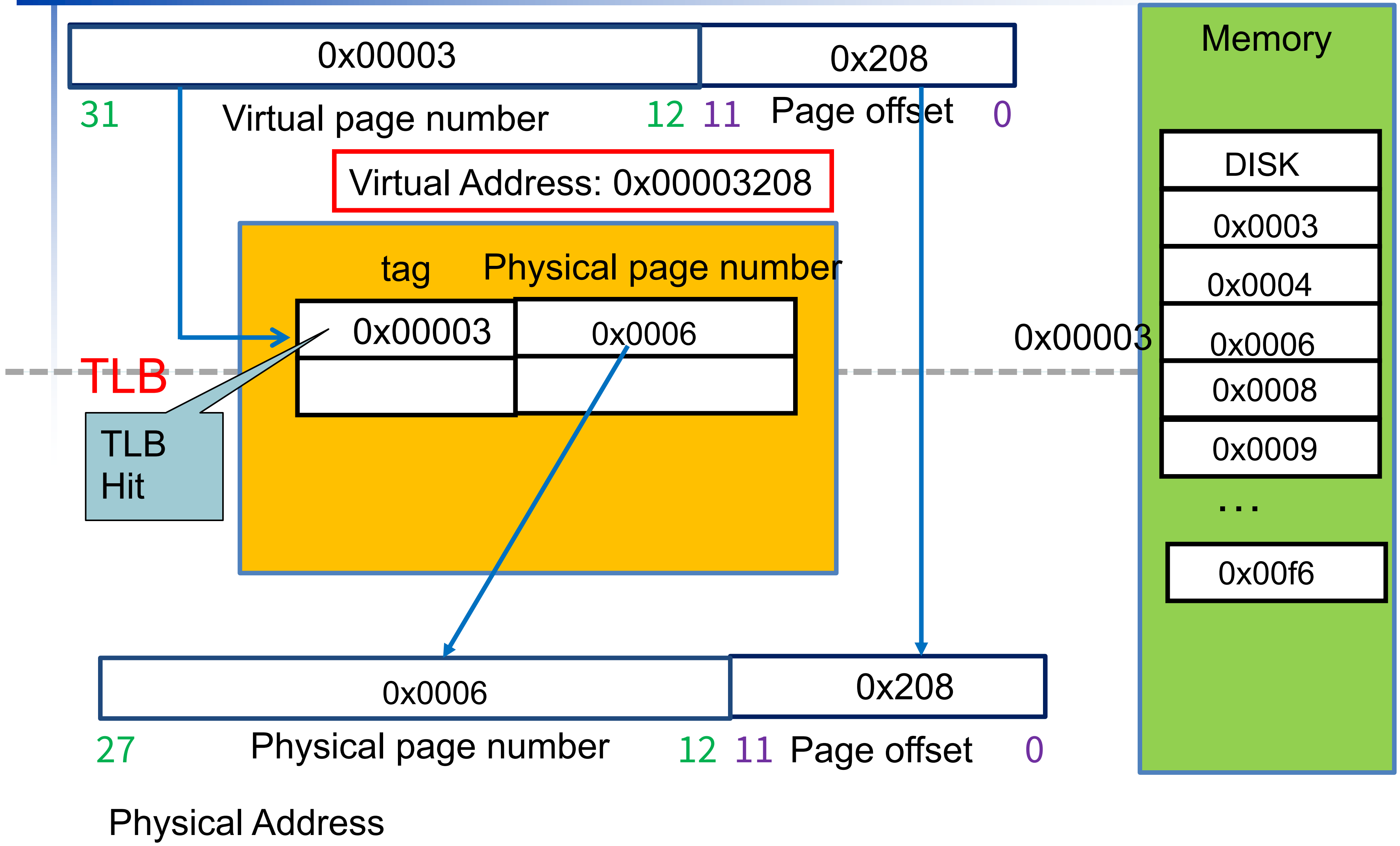
TLB Example (cont'd)



Physical Address

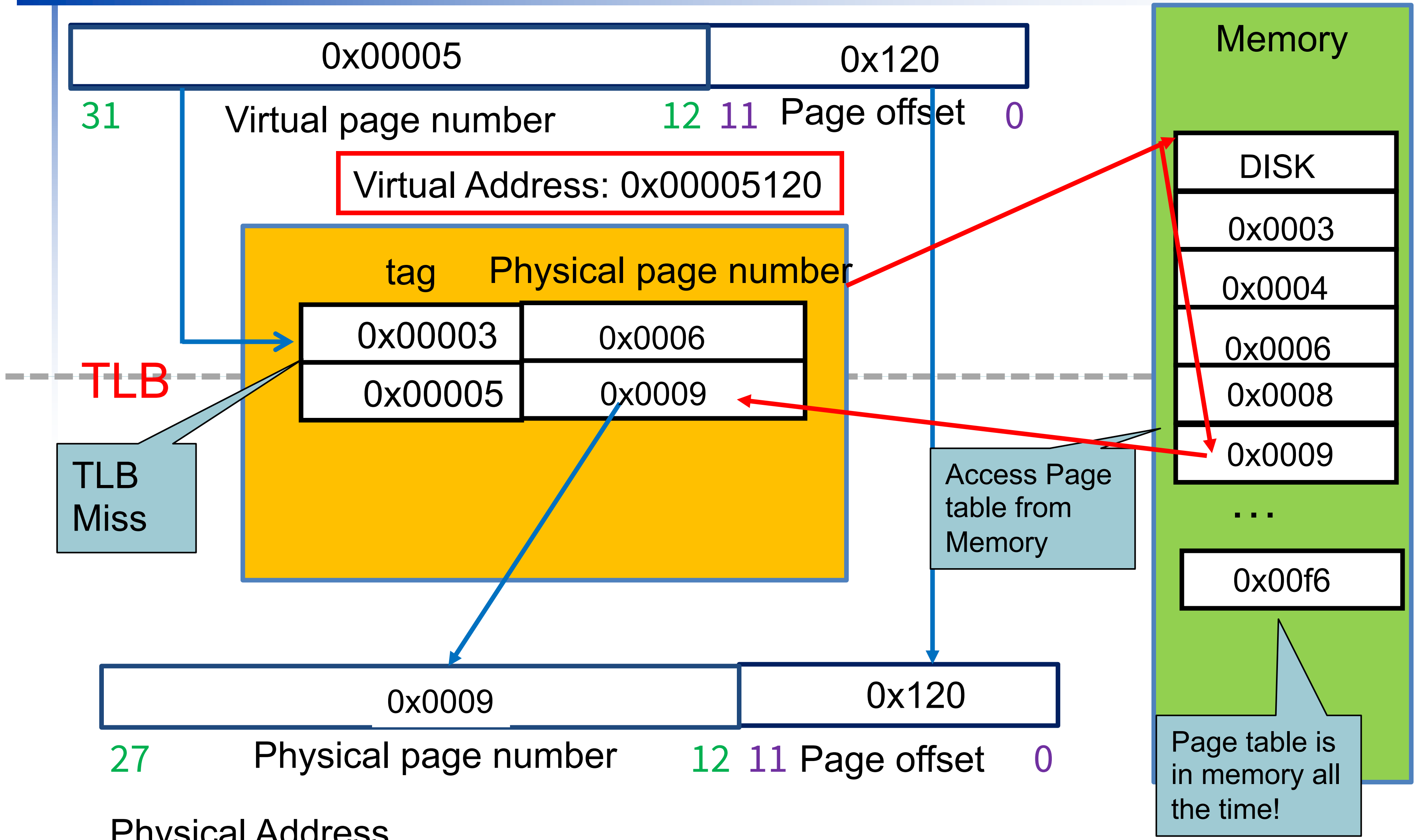
*Valid, dirty, reference bits omitted here.

TLB Example (cont'd)



*Valid, dirty, reference bits omitted here.

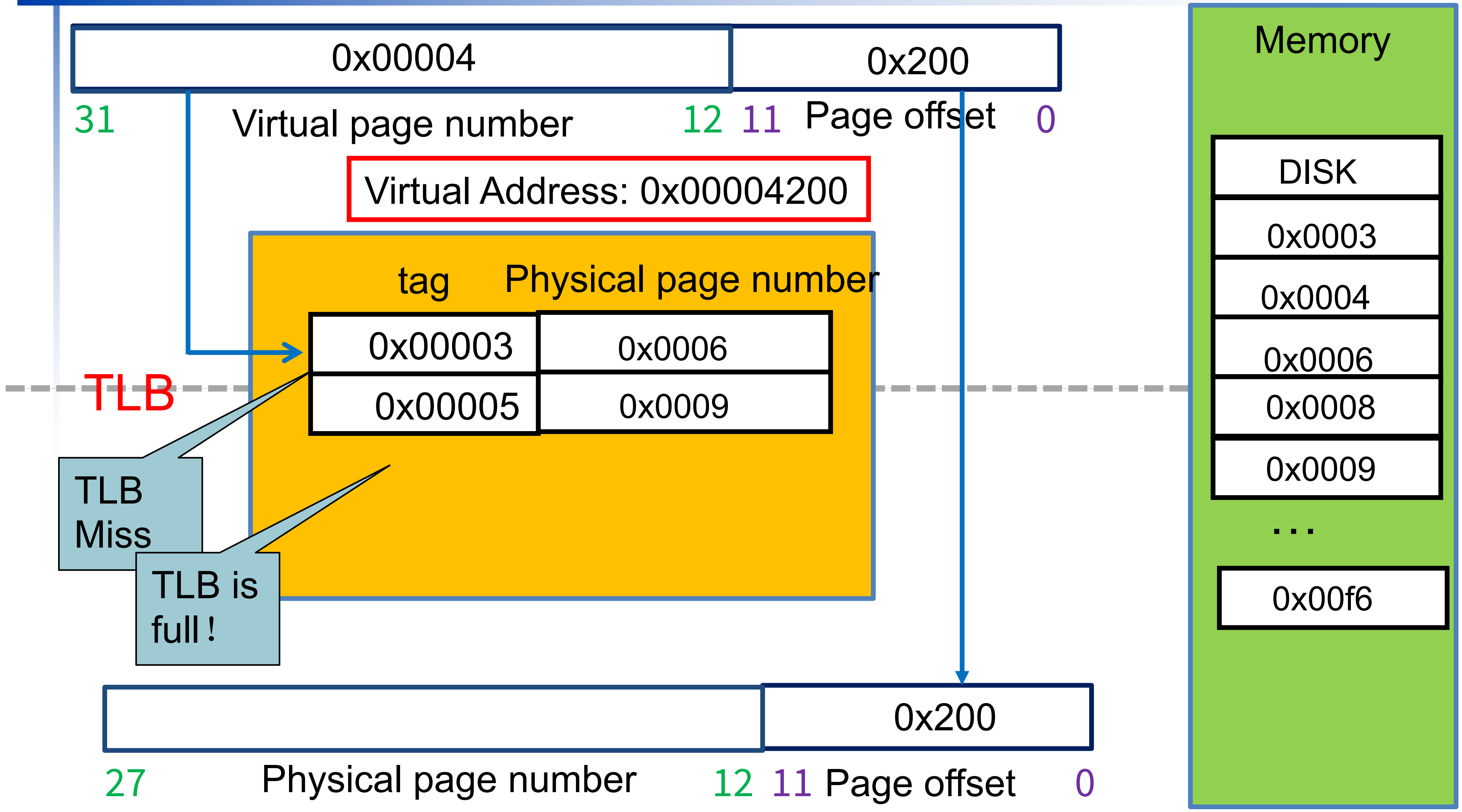
TLB Example (cont'd)



Physical Address

*Valid, dirty, reference bits omitted here.

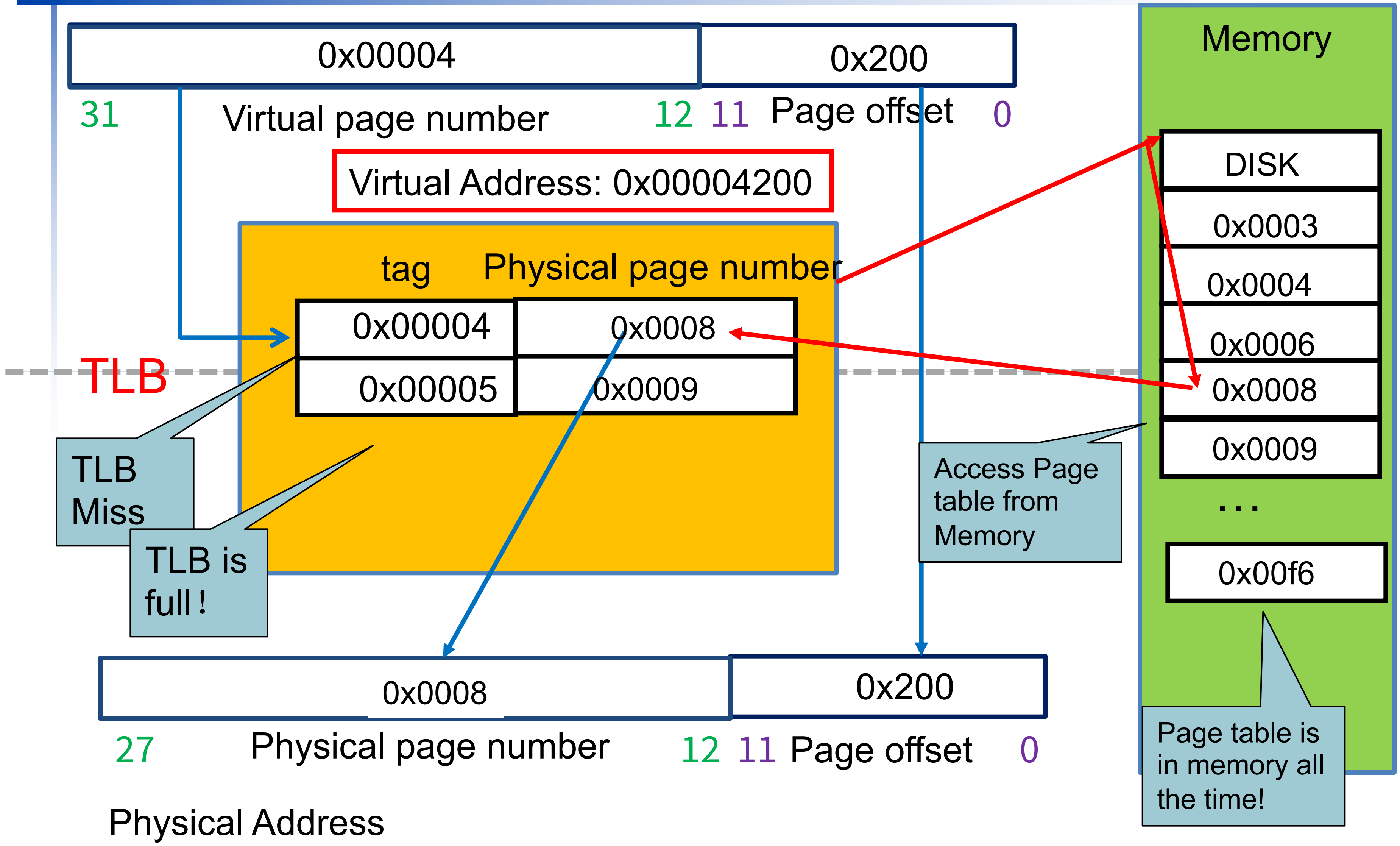
TLB Example (cont'd)



Physical Address

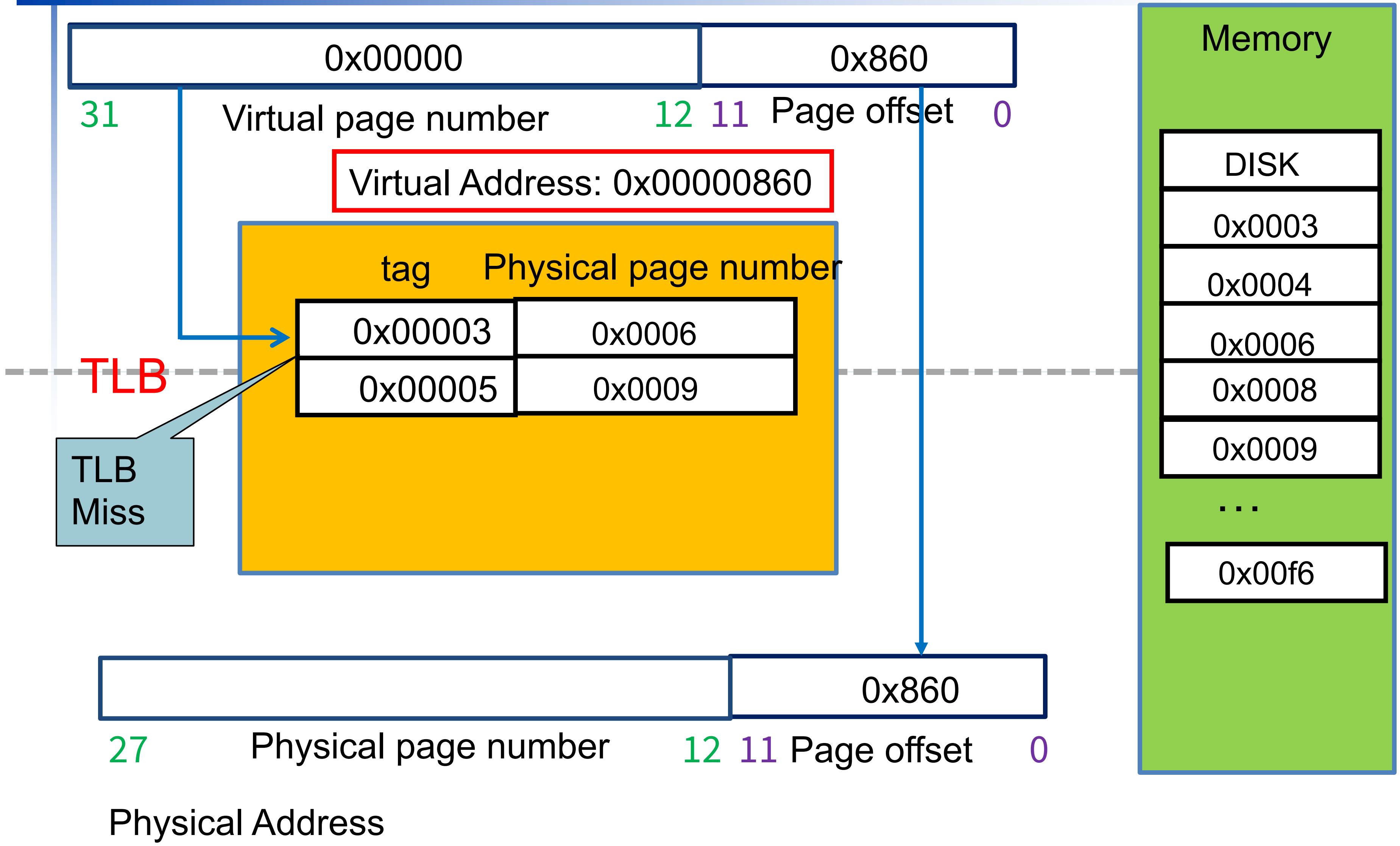
*Valid, dirty, reference bits omitted here.

TLB Example (cont'd)



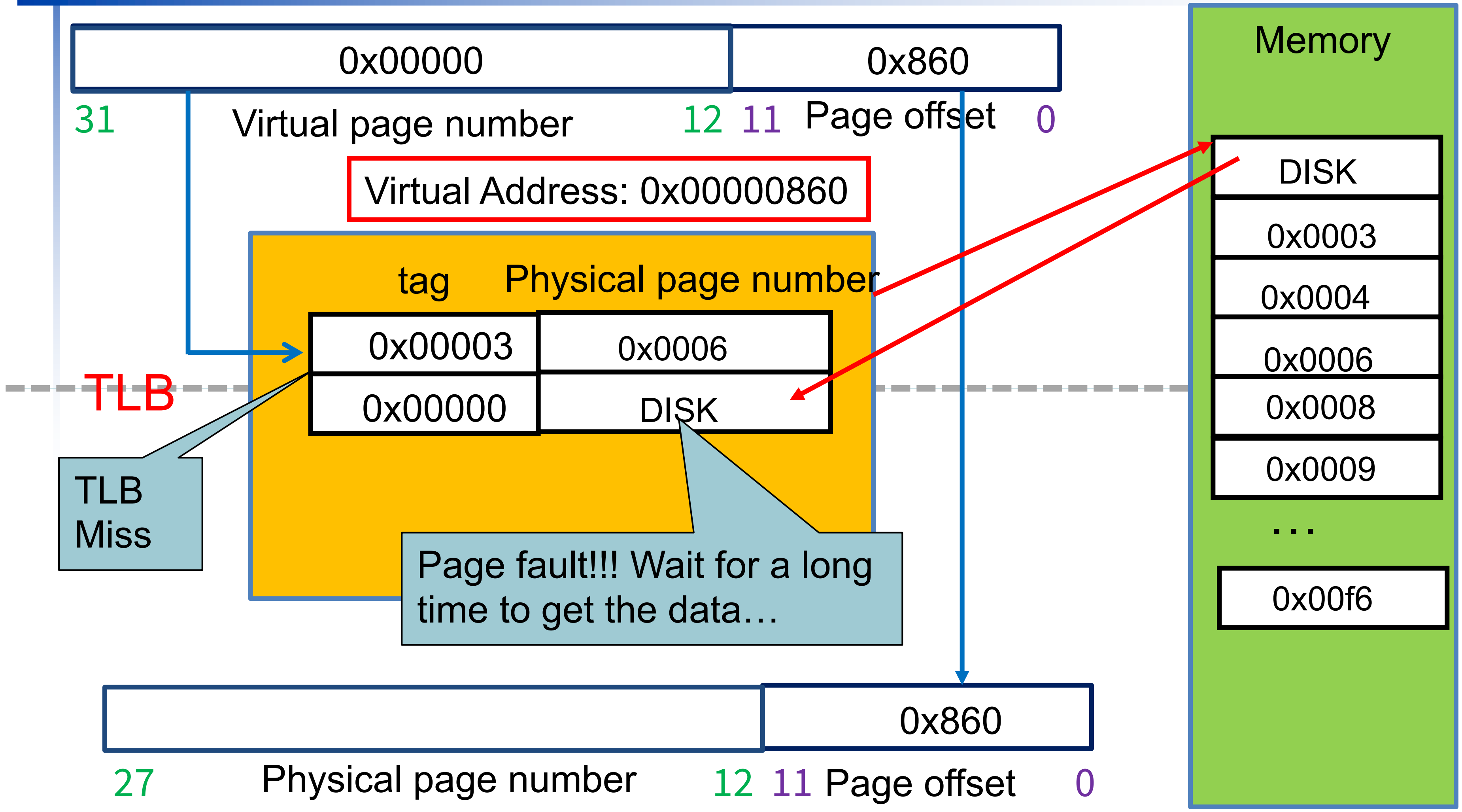
*Valid, dirty, reference bits omitted here.

TLB Example (cont'd)



*Valid, dirty, reference bits omitted here.

TLB Example (cont'd)



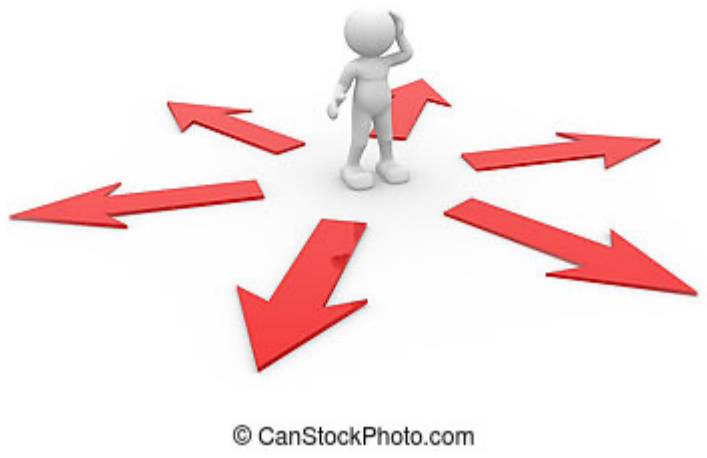
Physical Address

*Valid, dirty, reference bits omitted here.

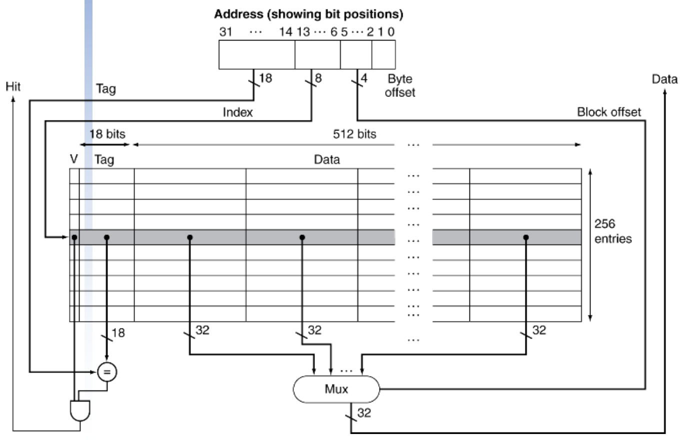
TLB Example (cont’d)

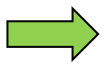
- In reality
  - Need to check valid bit, dirty bit and reference bits before overwriting a block on TLB.
- Page fault is unfortunate, it takes a long time to fetch from the disk

Now, we have learned cache, main memory, virtual memory, page table, TLB, … How do they all work together?

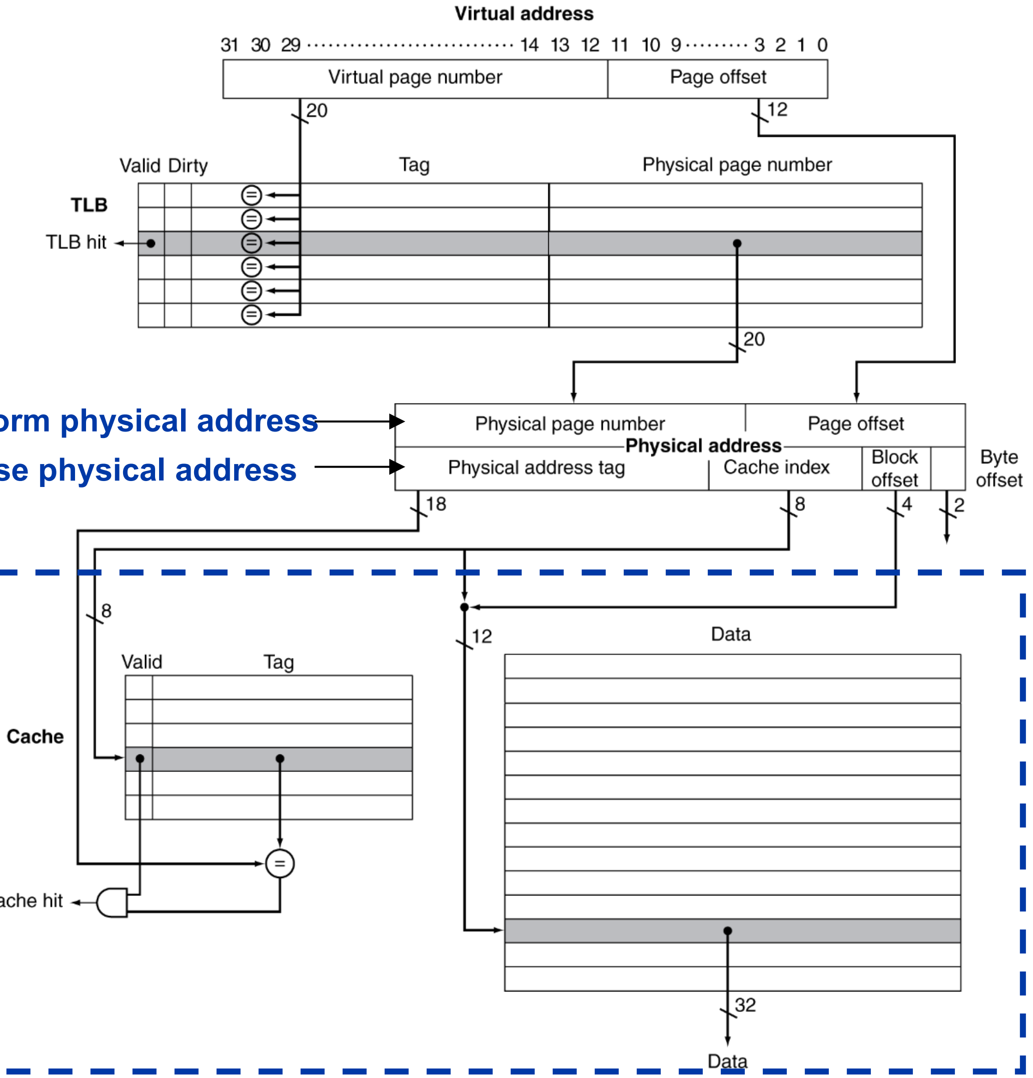


TLB and Cache Interaction





Assuming 4K page size



Form physical address

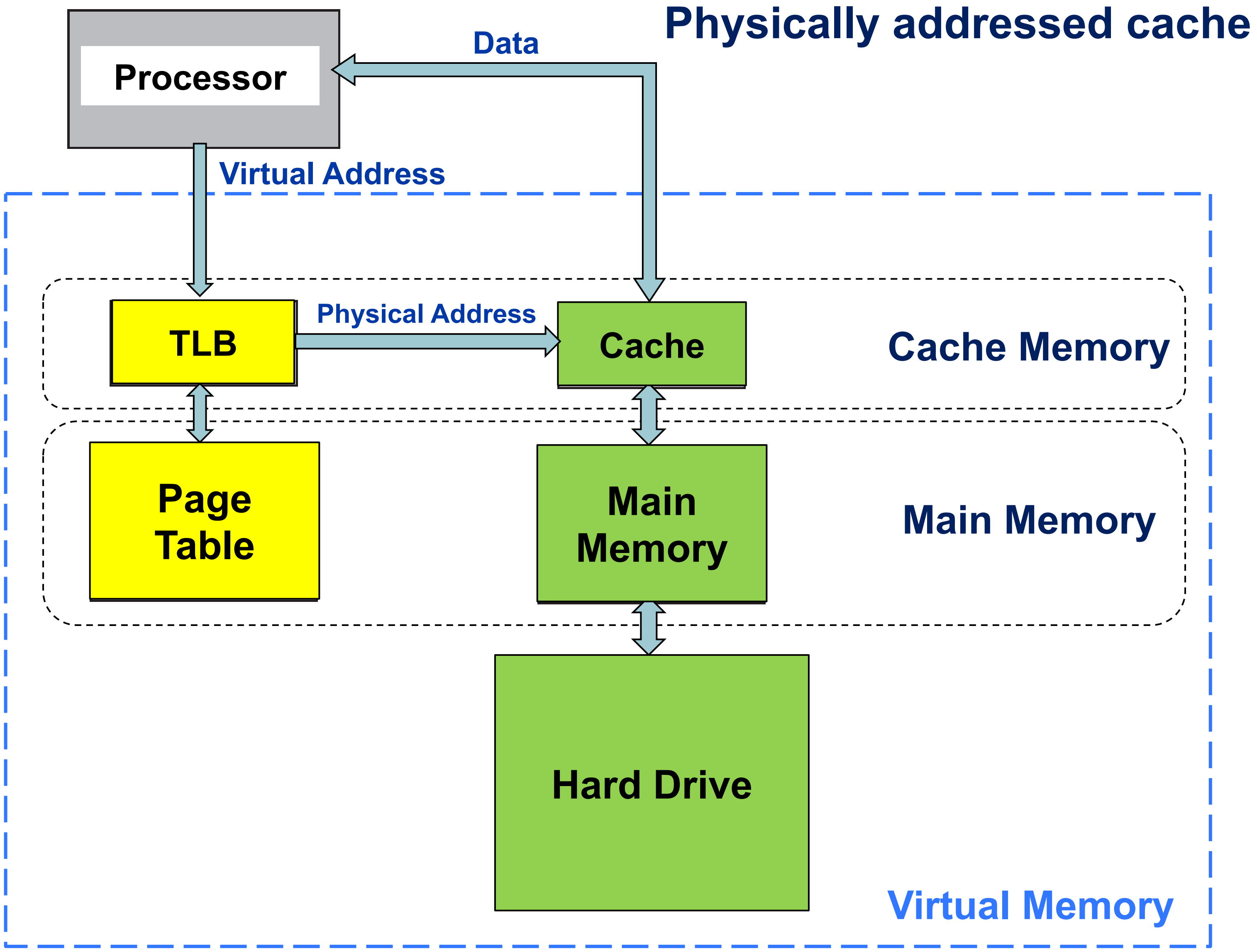
Use physical address

Small RAM for valid and tags

Big RAM aligned in word for data

- Indexed by cache index and block offset
- **No need for big MUX, improved cache access time**

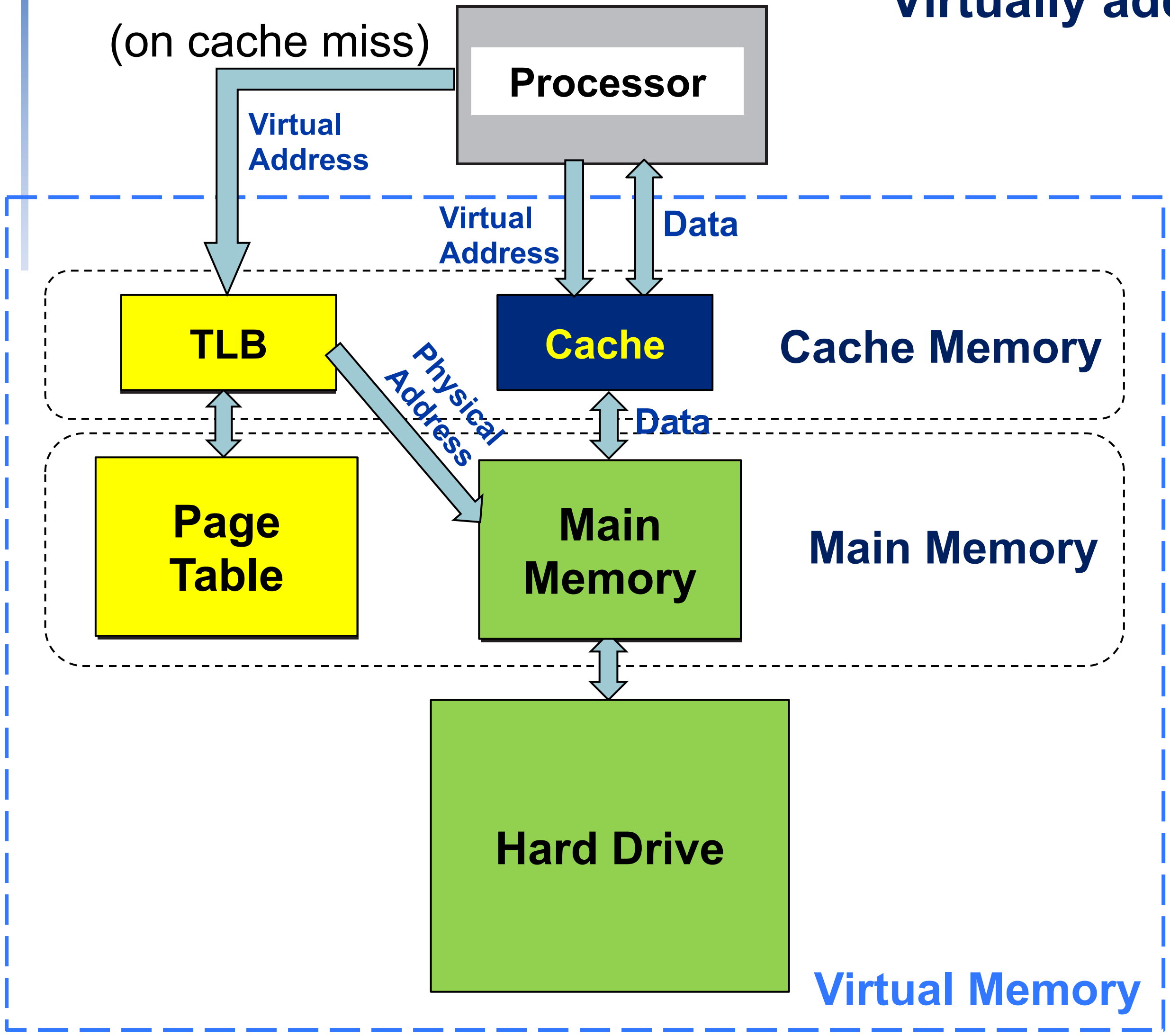
TLB vs. Cache


Physically addressed cache
Processor
Data
Virtual Address
TLB
Physical Address
Cache
Cache Memory
Page Table
Main Memory
Main Memory
Hard Drive
Virtual Memory

TLB and Cache Interaction

- If cache uses physical address
  - Need to translate before access cache
  - Virtual Addr → TLB → cache
  - Having TLB on the critical path, taking longer time
- TLB hit and a cache hit are **independent** events, but a cache hit can only occur after a TLB hit occurs, which means that the data must be present in memory.

TLB vs. Cache

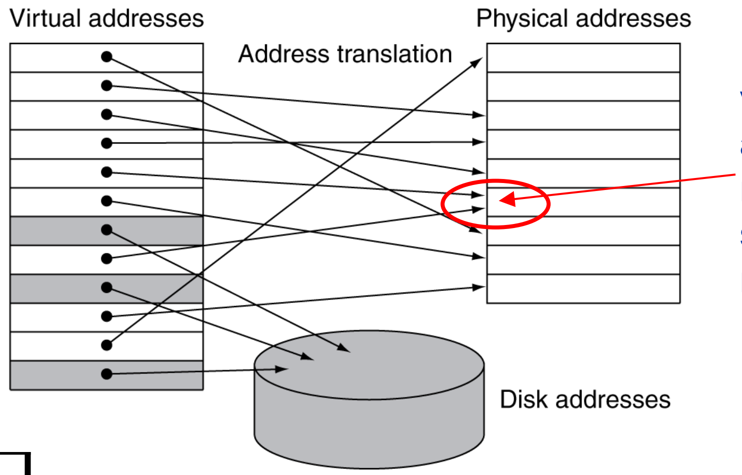


Virtually addressed cache

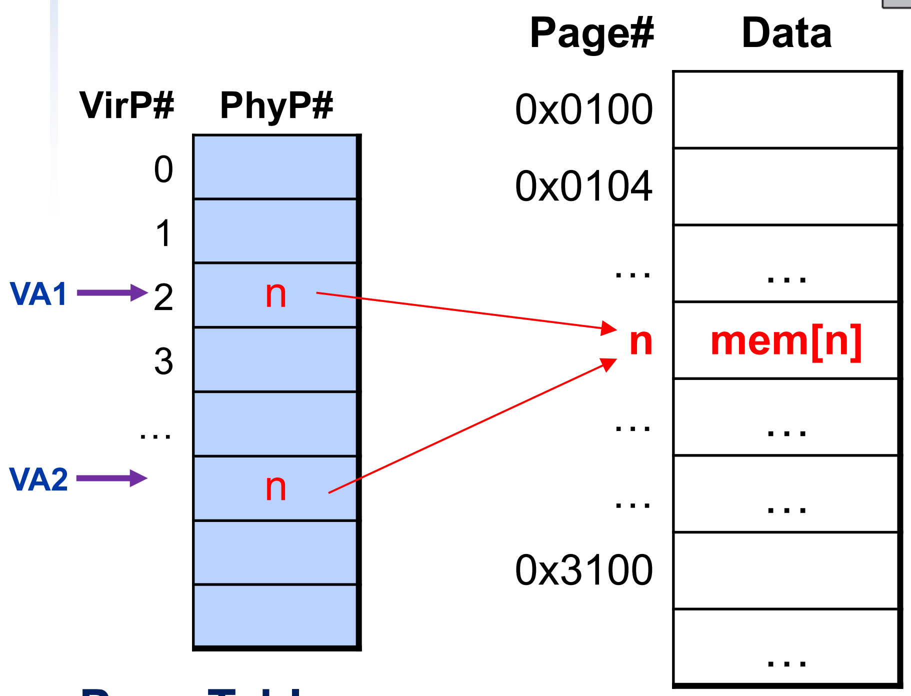
- No need for translation
- Translate when cache miss
- Complications due to aliasing
  - When different virtual spaces map to the same physical main memory location
- Compromise of both
  - Virtually addressed physically tagged cache

Virtually Addressed Cache

Aliasing: the same object has two different names

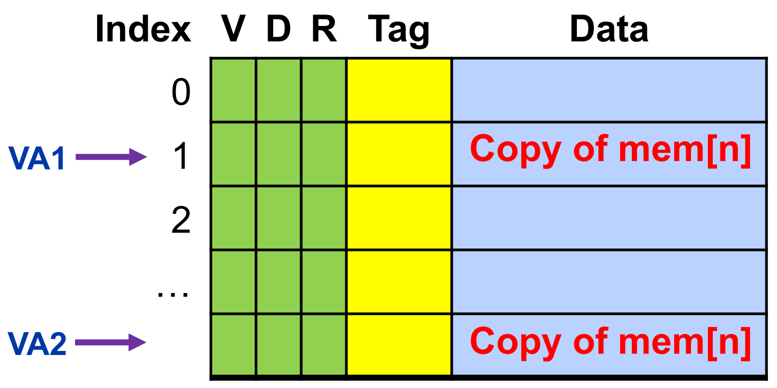


Multiple virtual addresses map to the same physical memory page



Page Table

Main Memory



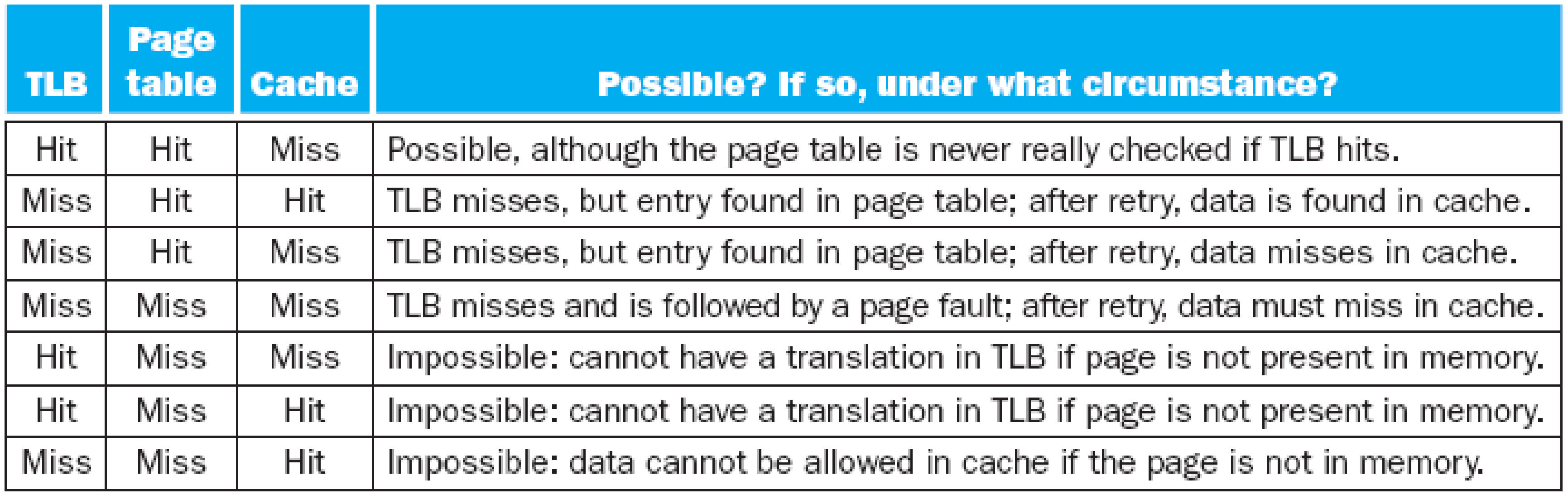
Virtually Addressed Cache

Class Exercise with TLB

- Given
  - 4KB page size, 16KB physical memory, LRU replacement
  - Virtual address: byte addressable, 20 bits (how many bytes?)
  - Page table for program A stored in page #0 of physical memory, starting at address 0x0100, assume only 2 valid entries in page table:
    - Virtual page number 0 => physical page number 1
    - Virtual page number 1 => physical page number 2
- Fully associative TLB, 2 entries
- Show the memory structure, complete following table

Virtual Address	Virtual page number	TLB miss?	Page fault?	Physical Address
0x00F0C				
0x01F0C				
0x20F0C				
0x00100				
0x00200				
0x30000				
0x01FFF				
0x00200				

Relationships in Memory Hierarchy



TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

- Assume cache uses physical addresses
- All data in a memory must also be present in its lower level
  - Impossible to copy data cross levels

Memory Protection

- Different programs can have the same virtual addresses
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions available in supervisor mode
    - System call exception (e.g., syscall in MIPS)
  - Page tables and other status information only updated in supervisor mode by operating system
    - Write enable bit protects memory from being written
  - Different page tables ensure different translations
    - Different translation ensures separate memory locations

Summary of The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Hardware costs
  - Reduce comparisons to reduce cost
- Virtual memory
  - Lookup table full associativity
  - Benefit in reduced miss rate

Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Needs tracking mechanism (reference bit updated and periodically refreshed)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU for high associtivity, easier to implement

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given unaffordable disk write latency

Sources of Misses (3C model)

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for blocks in a set
  - Would not occur in a fully associative cache of the same total size

Cache Design Trade-offs

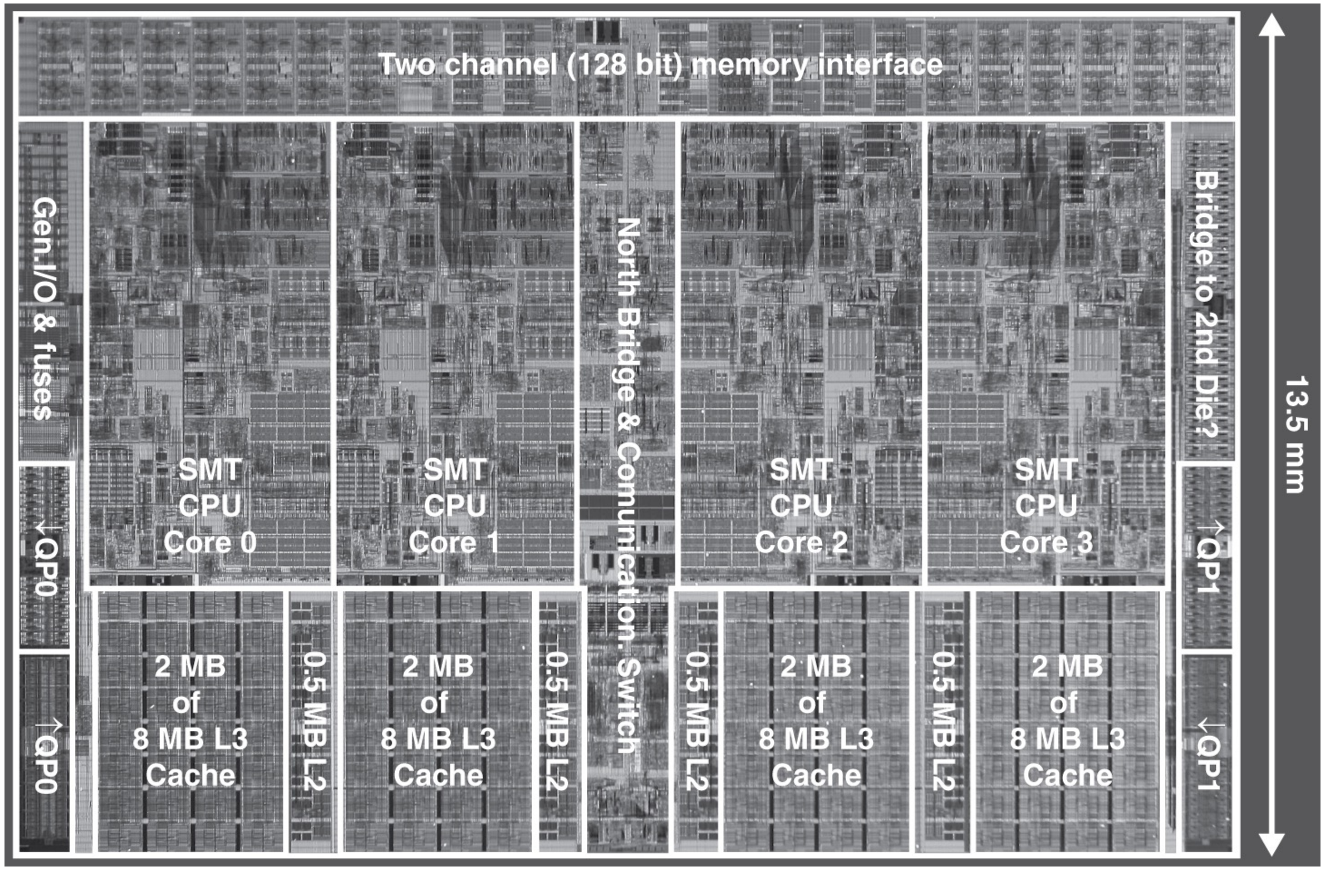
Design change	***Effect on miss rate***	***Negative performance effect***
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	Will increase access time
Increase block size	Decrease compulsory misses May increase conflict misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Miss Penalty Reduction

- Return requested word first
  - Then back-fill rest of block
- Non-blocking miss processing
  - Hit under miss: allow hits to proceed
  - Miss under miss: allow multiple outstanding misses
    - Need parallel processing capability
- bank interleaved cache/main memory
  - multiple concurrent accesses per cycle

Example: Intel Nehalem 4-core processor

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

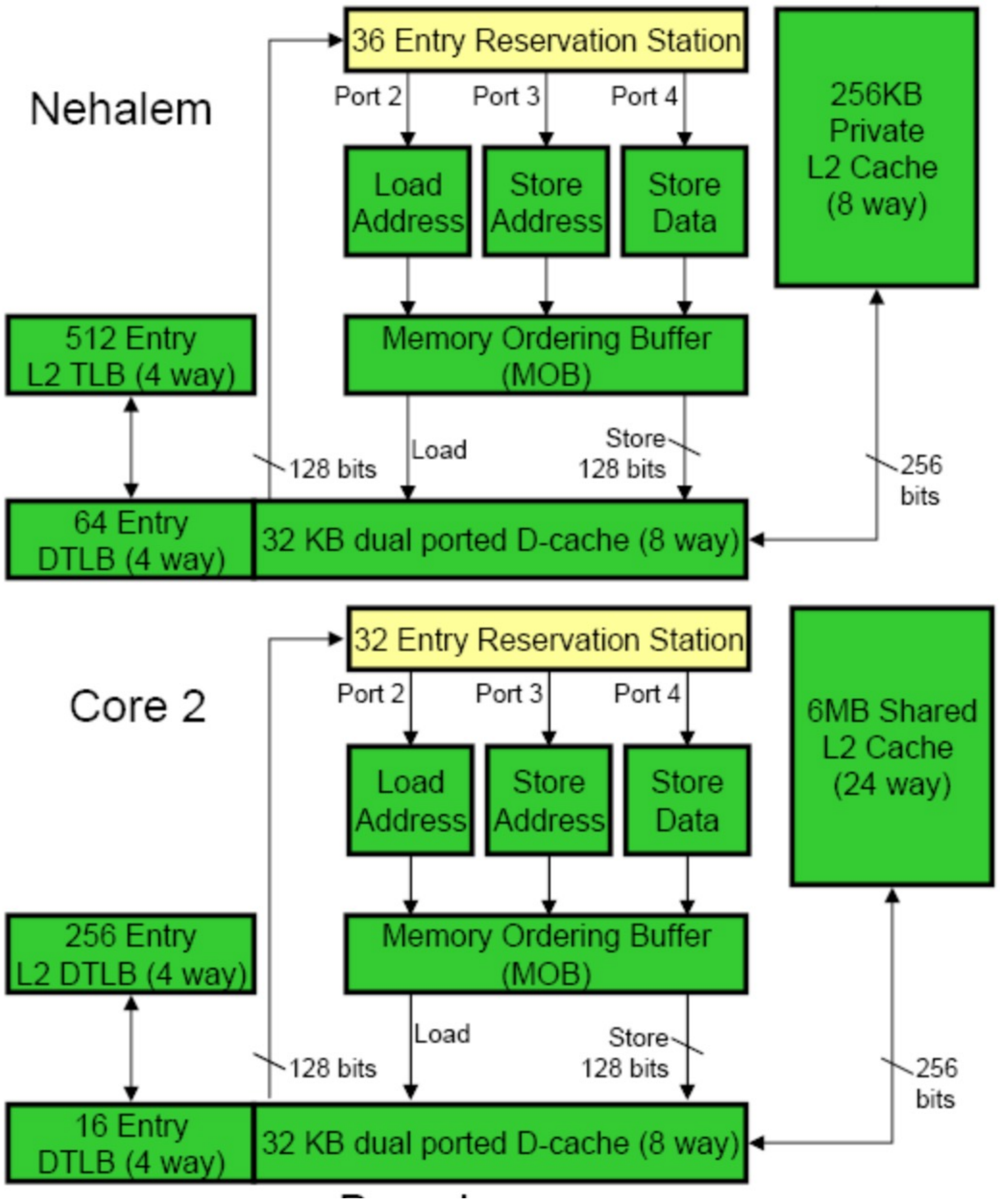
2-Level TLB

	Intel Nehalem	AMD Opteron X4
Virtual addr	48 bits	48 bits
Physical addr	44 bits	48 bits
Page size	4KB, 2/4MB	4KB, 2/4MB
L1 TLB (per core)	L1 I-TLB: 128 entries for small pages, 7 per thread (2×) for large pages L1 D-TLB: 64 entries for small pages, 32 for large pages Both 4-way, LRU replacement	L1 I-TLB: 48 entries L1 D-TLB: 48 entries Both fully associative, LRU replacement
L2 TLB (per core)	Single L2 TLB: 512 entries 4-way, LRU replacement	L2 I-TLB: 512 entries L2 D-TLB: 512 entries Both 4-way, round-robin LRU
TLB misses	Handled in hardware	Handled in hardware

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

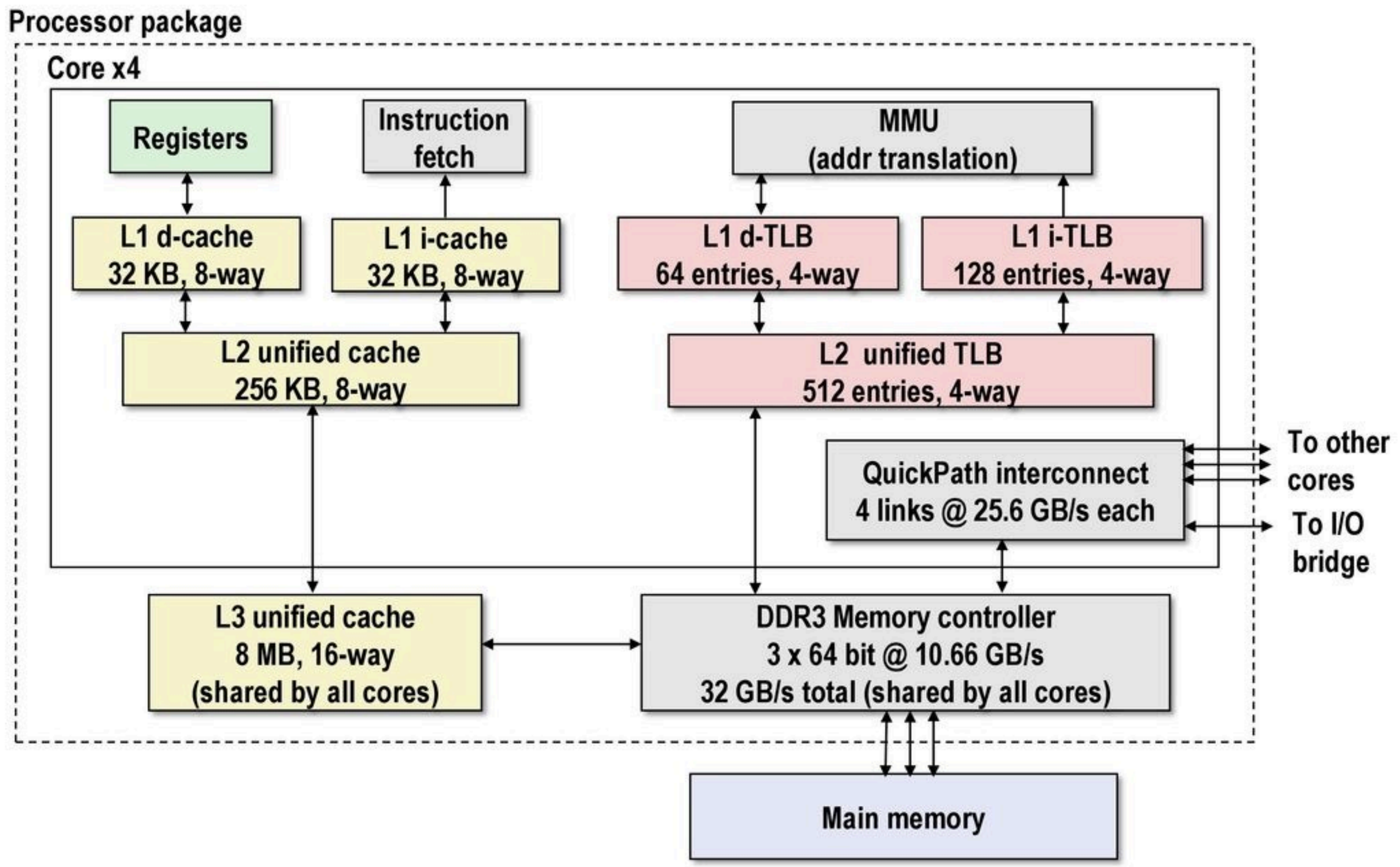
n/a: data not available



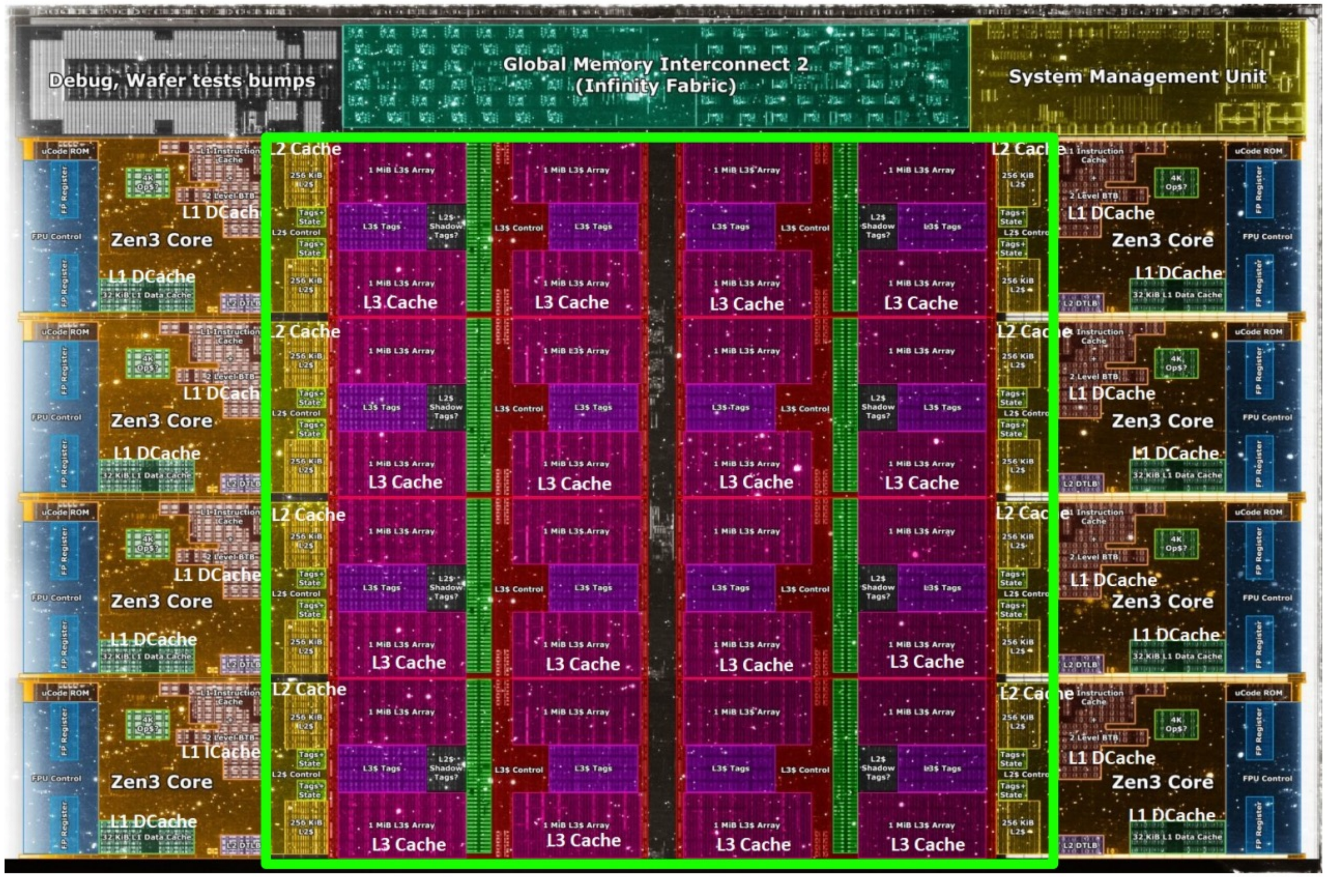
Nehalem
36 Entry Reservation Station
Port 2
Port 3
Port 4
Load Address
Store Address
Store Data
256KB Private L2 Cache (8 way)
512 Entry L2 TLB (4 way)
Memory Ordering Buffer (MOB)
128 bits
Load
Store 128 bits
256 bits
64 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)
Core 2
32 Entry Reservation Station
Port 2
Port 3
Port 4
Load Address
Store Address
Store Data
6MB Shared L2 Cache (24 way)
256 Entry L2 DTLB (4 way)
Memory Ordering Buffer (MOB)
128 bits
Load
Store 128 bits
256 bits
16 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)

Source: https://www.realworldtech.com/nehalem/8/

Intel Core i7 Memory System


Processor package
Core x4
Registers
Instruction fetch
MMU (addr translation)
L1 d-cache 32 KB, 8-way
L1 i-cache 32 KB, 8-way
L1 d-TLB 64 entries, 4-way
L1 i-TLB 128 entries, 4-way
L2 unified cache 256 KB, 8-way
L2 unified TLB 512 entries, 4-way
QuickPath interconnect 4 links @ 25.6 GB/s each
To other cores
To I/O bridge
L3 unified cache 8 MB, 16-way (shared by all cores)
DDR3 Memory controller 3 x 64 bit @ 10.66 GB/s 32 GB/s total (shared by all cores)
Main memory

Memory Hierarchy in a Modern System



Core Count:
8 cores/16 threads

L1 Caches:
32 KB per core

L2 Caches:
512 KB per core

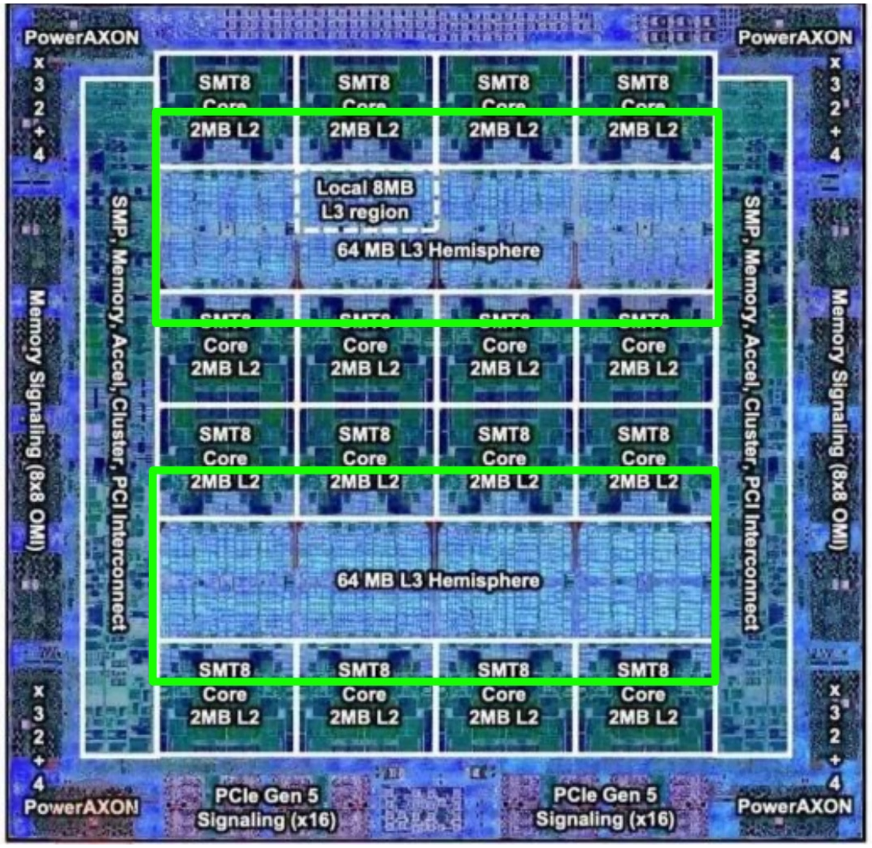
L3 Cache:
32 MB shared

AMD Ryzen 5000, 2020

CPU

Source: https://wccftech.com/amd-ryzen-5000-zen-3-vermeer-undressed-high-res-die-shots-close-ups-pictured-detailed/

Memory Hierarchy in a Modern System


PowerAXON
x 3 2 + 4
SMT8 Core
2MB L2
Local 8MB L3 region
64 MB L3 Hemisphere
SMP, Memory, Accel, Cluster, PCI Interconnect
Memory Signaling (8x8 OMI)
PCIe Gen 5 Signaling (x16)

IBM POWER10, 2020

Cores:
15-16 cores,
8 threads/core

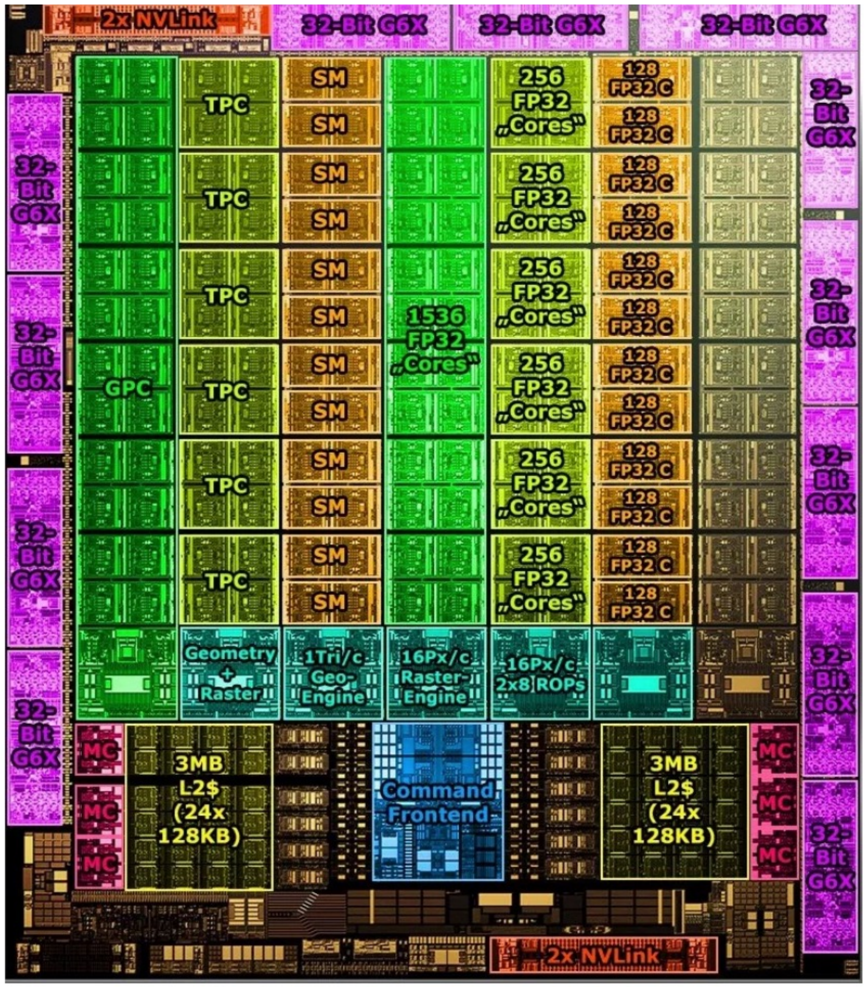
L2 Caches:
2 MB per core

L3 Cache:
120 MB shared

CPU

Source: https://www.it-techblog.de/ibm-power10-prozessor-mehr-speicher-mehr-tempo-mehr-sicherheit/09/2020/

Memory Hierarchy in a Modern System



Nvidia Ampere, 2020

Cores:
128 Streaming Multiprocessors

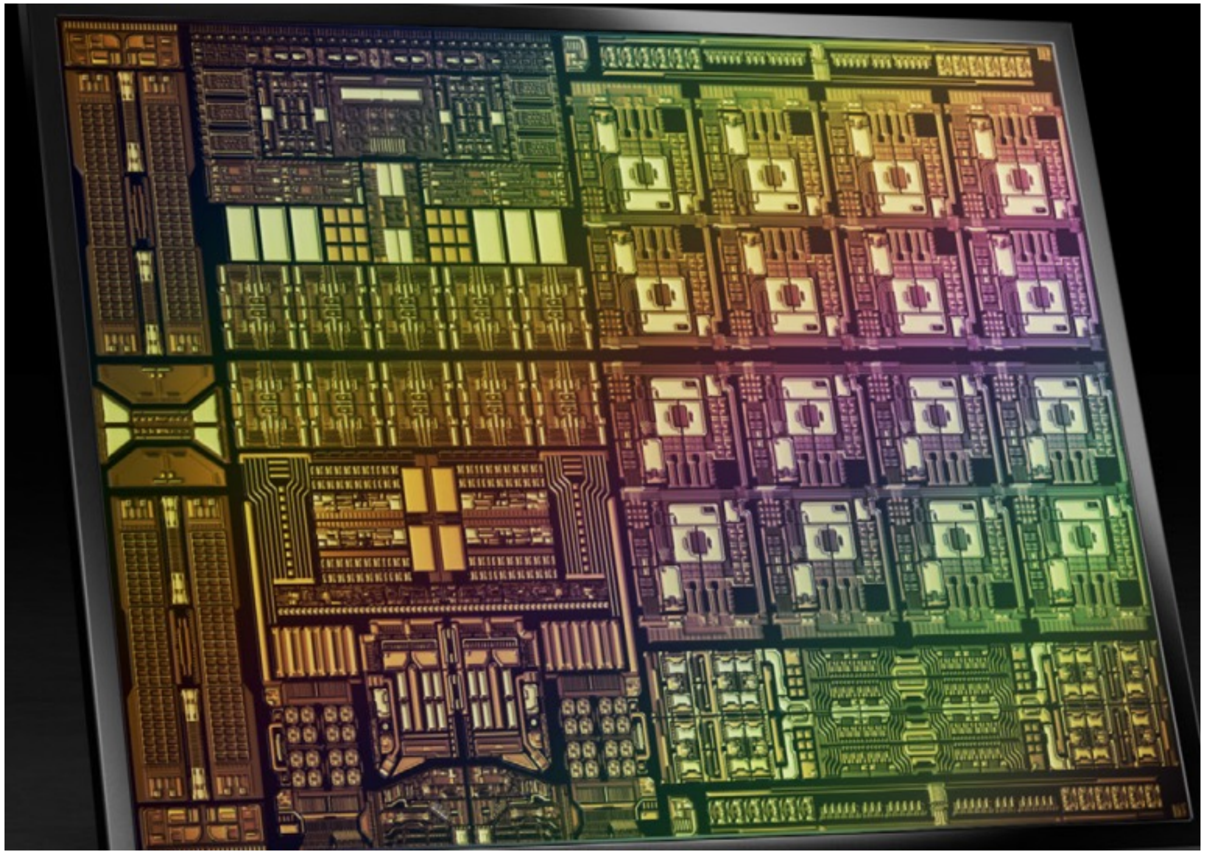
L1 Cache or Scratchpad:
192KB per SM
Can be used as L1 Cache and/or Scratchpad

L2 Cache:
40 MB shared

GPU

Source: https://www.tomshardware.com/news/infrared-photographer-photos-nvidia-ga102-ampere-silicon

Memory Hierarchy in a Modern System



Cores:
16 ARM v8.2

L2 Cache:
8MB

LLC system cache:
16MB

NVIDIA BLUEFIELD 3 DPU, 2021

DPU

Source: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
  - Vitual space → L1 TLB → L2 TLB → (page table) → physical space
- Memory system design is critical for multiprocessors