# Summary: what we covered so far

- virtual memory has many uses
  - pretend we have lots of RAM
  - provide isolation between processes
  - provide sharing between processes
  - mapping of IO peripherals into CPU address space
- the page table maps virtual to physical pages
  - provides a fully associative mapping via table in RAM
- must translate virtual to physical on ***every*** access
  - all user-space addresses are virtual
  - even kernel-space addresses are virtual

# TLB topics and common framework for memory hierarchy

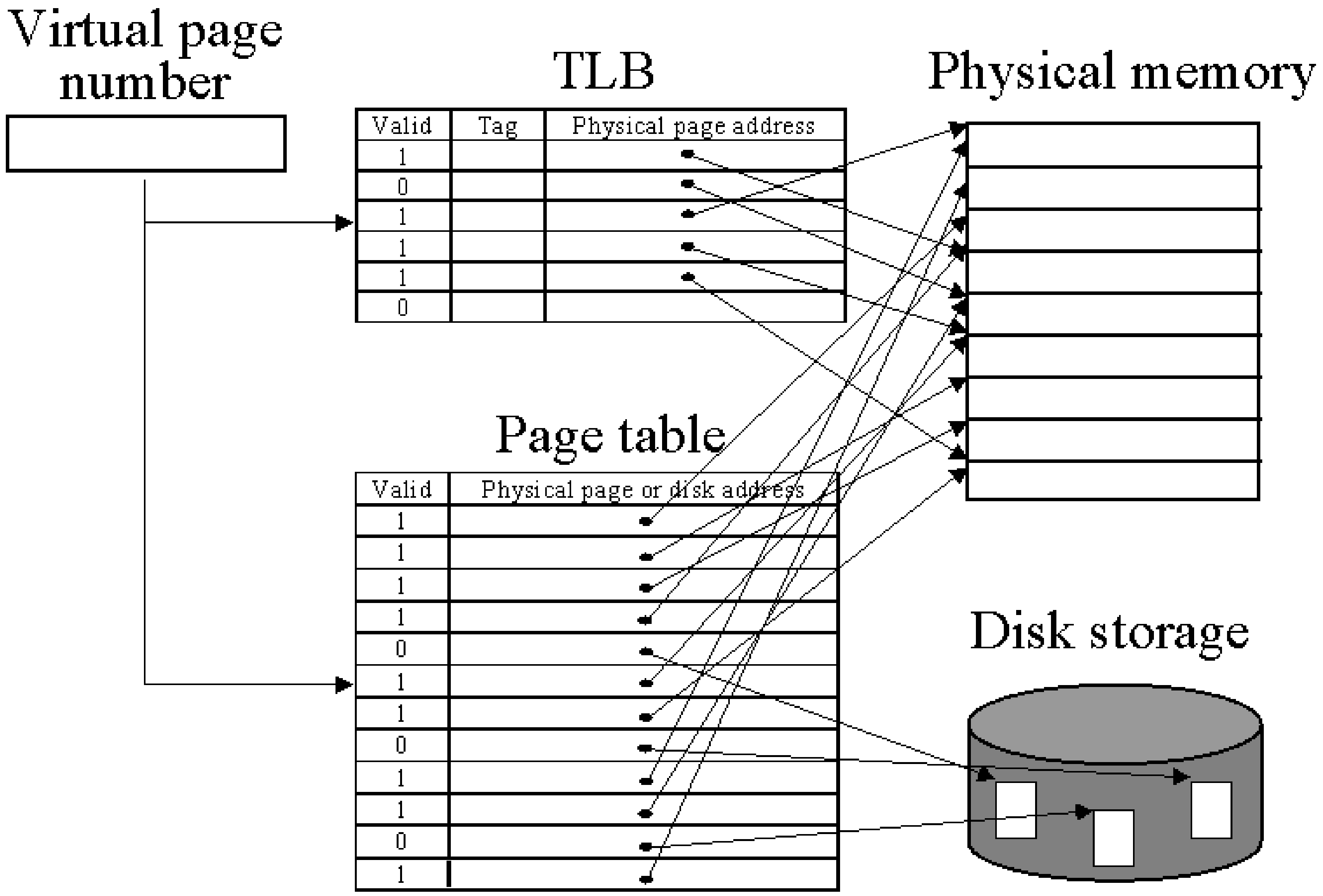
(3rd Ed: p.527-555, 4th Ed: 502-534)

- TLB organisation: Intel and AMD processors
- TLB misses
- protection with virtual memory
- page faults
- common framework for memory hierarchy

# Accelerating address translation

- page tables large: 10MB-1GB, so in main memory
  - problem: memory reference takes 2 memory accesses, to get physical address and to get data
- special cache for page table mappings: TLB (translation look-aside buffer)
- memory reference: look up virtual page number in TLB
  - if TLB hit: get physical page address
  - if TLB miss: load the relevant part of the page table into TLB and try again

# TLB and page table

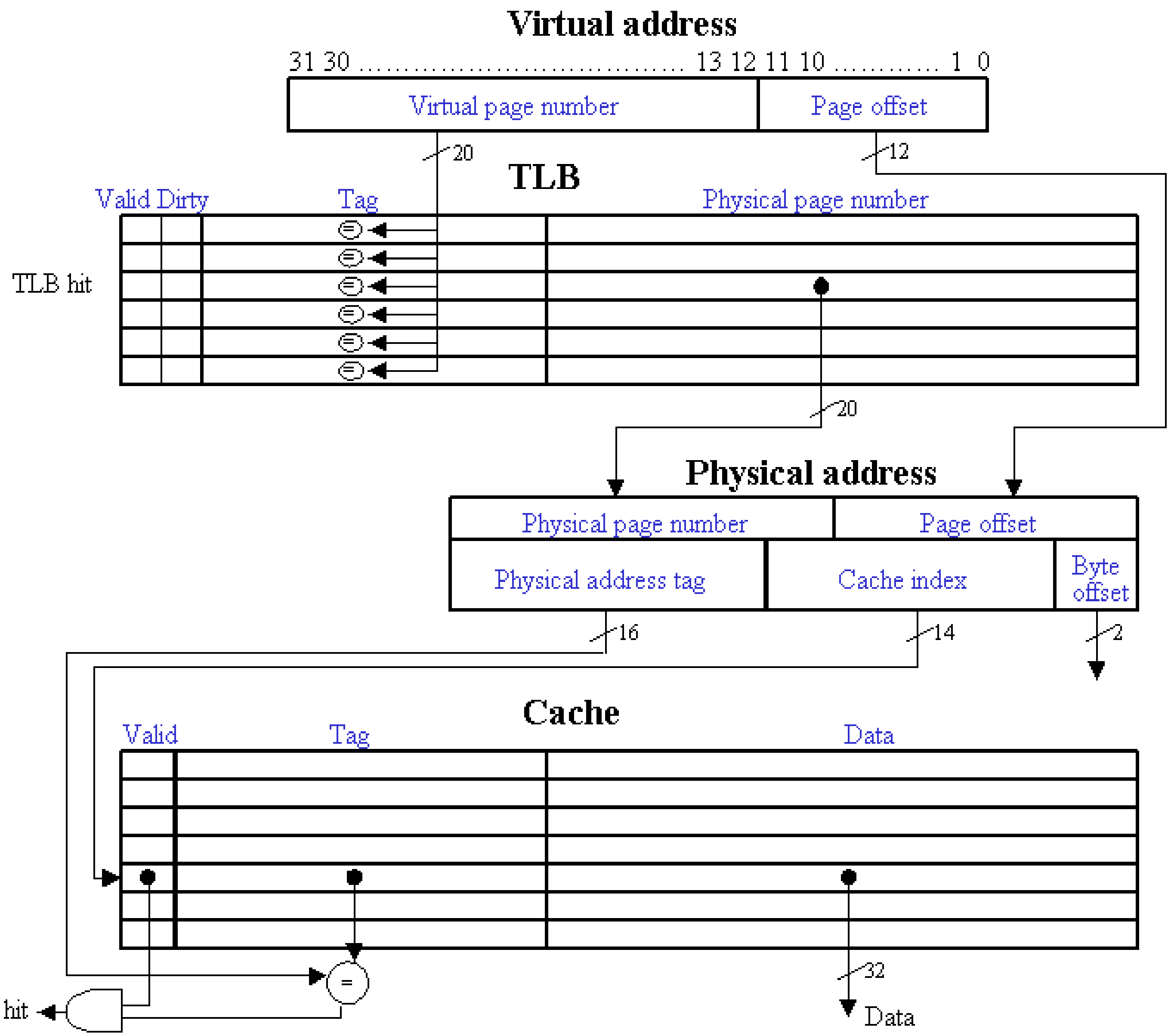


| Valid | Tag | Physical page address |
|---|---|---|
| 1 | | |
| 0 | | |
| 1 | | |
| 1 | | |
| 1 | | |
| 0 | | |

| Valid | Physical page or disk address |
|---|---|
| 1 | |
| 1 | |
| 1 | |
| 1 | |
| 0 | |
| 1 | |
| 1 | |
| 0 | |
| 1 | |
| 1 | |
| 0 | |
| 1 | |

# TLB and cache

TLB:
where is
physical
address?

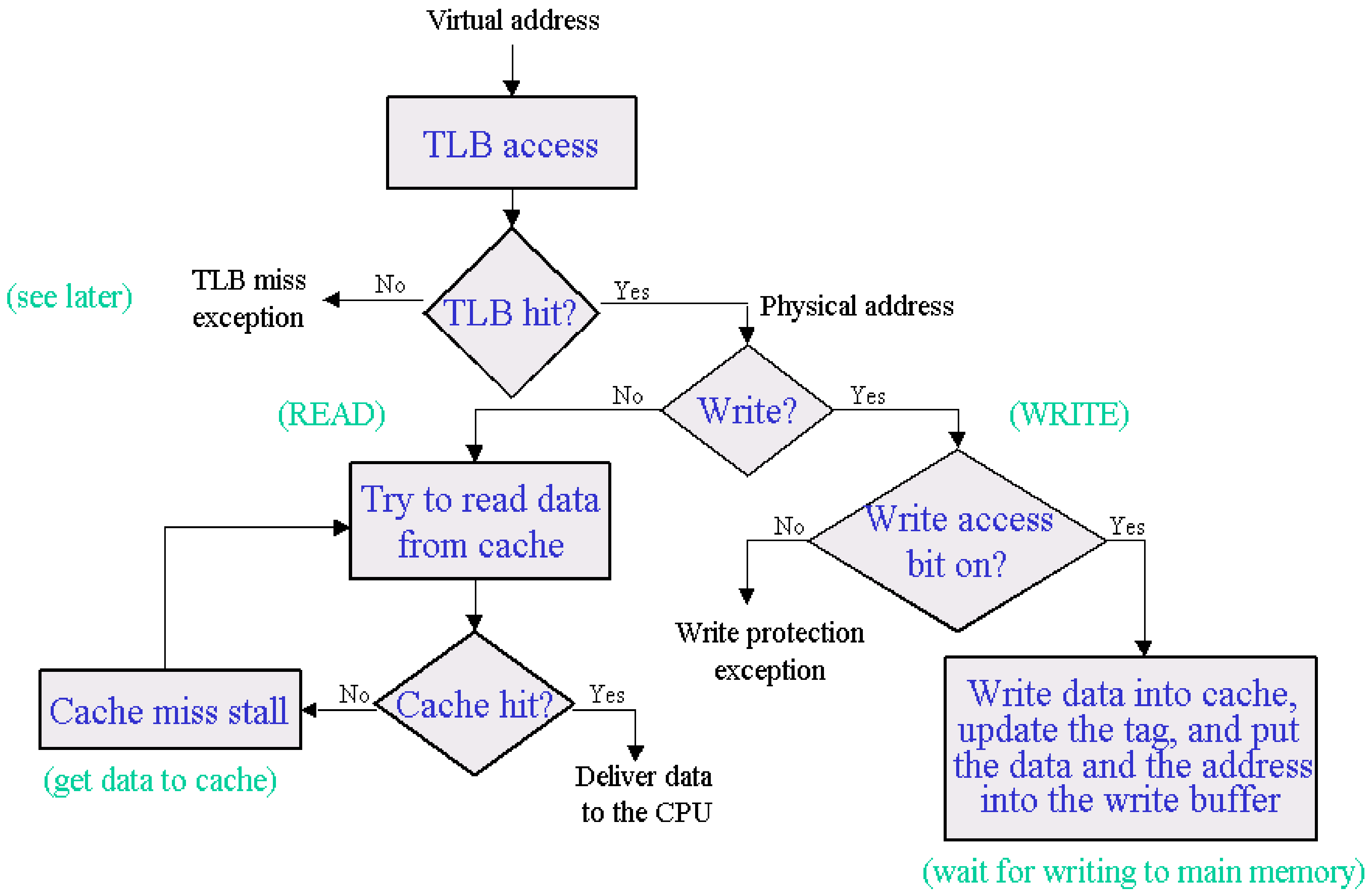


often fully
associative
(see later)

Cache:
where are
data?

often direct
mapped

# Memory read and write



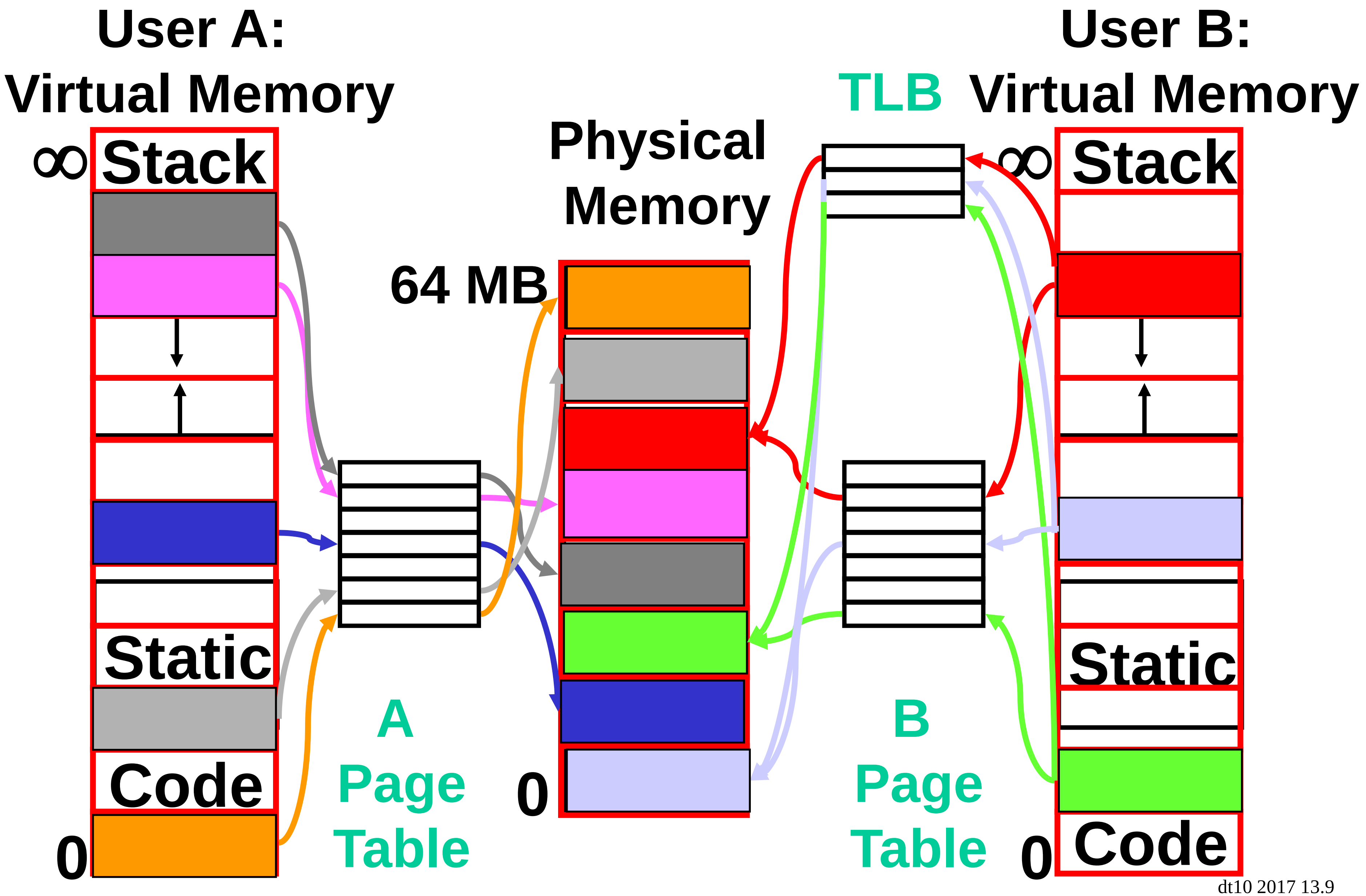
## 2-Level TLB Organization

| | 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<br><br>L1 D-TLB: 64 entries for small pages, 32 for large pages<br><br>Both 4-way, LRU replacement | L1 I-TLB: 48 entries<br><br>L1 D-TLB: 48 entries<br><br>Both fully associative, LRU replacement |
| L2 TLB (per core) | Single L2 TLB: 512 entries<br><br>4-way, LRU replacement | L2 I-TLB: 512 entries<br><br>L2 D-TLB: 512 entries<br><br>Both 4-way, round-robin LRU |
| TLB misses | Handled in hardware | Handled in hardware |

## TLB miss

- no TLB entry matches virtual address
  - either page is in main memory,
    just need to create missing TLB entry
  - or page is not in main memory,
    page fault
- action: check valid bit of page table entry
  - if on, copy physical page address to TLB
  - if off, transfer control to operating system
    (e.g. by exception) to deal with page fault
- quiz: what are possible combinations of hit/miss in cache, TLB and main memory?

# Virtual memory: multiple processes



## Protection with virtual memory

- a user process should not interfere with other users or the operating system
- page table can contain Access Right entries: e.g. *Read Only*, *Read/Write*, *Executable* in addition to the *valid* bit
- hardware supports 2 execution modes: user mode and supervisor mode
- user process runs in users mode: accesses its own page table by reading page table register (but not write page table register), following access rights
- operating system runs in supervisor mode: can change the page tables; change page table register and clear TLB for a new user process

# Handling page faults

1. exception occurs, Cause Register indicates page fault
2. CPU changed to supervisor mode, disables further exceptions until enough saved state to recover
    1. EPC (Exception PC): address of offending instruction
    2. Cause Register: value indicates cause of exception
3. OS saves entire state of current process, e.g. EPC contains restart instruction when OS finishes dealing with page fault

- avoid writing registers/memory - non-recoverable

# Handling page faults (cont.)

4. find virtual address causing page fault:
   - in EPC, if instruction page fault
   - compute from base register and instruction offset, if data page fault (lw or sw)
5. once OS knows the virtual address:
   - look up page table to find disk address
   - choose physical page to replace; save replaced page on disk if dirty
   - copy referenced page from disk to memory

- OS finds another task for CPU during data transfer

# Design parameters

| Feature | Caches L1 / L2 | paged memory | TLB |
|---|---|---|---|
| total size (blocks) | 250-2000 / 15000-50000 | 16000-250000 | 40-1024 |
| total size (kilobytes) | 16-64 / 500-4000 | $10^6$-$10^9$ | 0.25-16 |
| block size (bytes) | 16-64 / 64-128 | 4000-64000 | 4-32 |
| miss penalty (cycles) | 10-25 / 100-1000 | $10^7$-$10^8$ | 10-1000 |
| miss rate (global for L2) | 2-5% / 0.1-2% | 0.00001-0.0001% | 0.01-2% |

## Common framework for memory hierarchy

- planning a block - upper level
  one place, a few places, any place
- finding a block - upper level
  by index, limited search, full search
- replacing a block - by one from lower level
  random, LRU
- writing a block - to lower level
  write through, write back

## Placement schemes

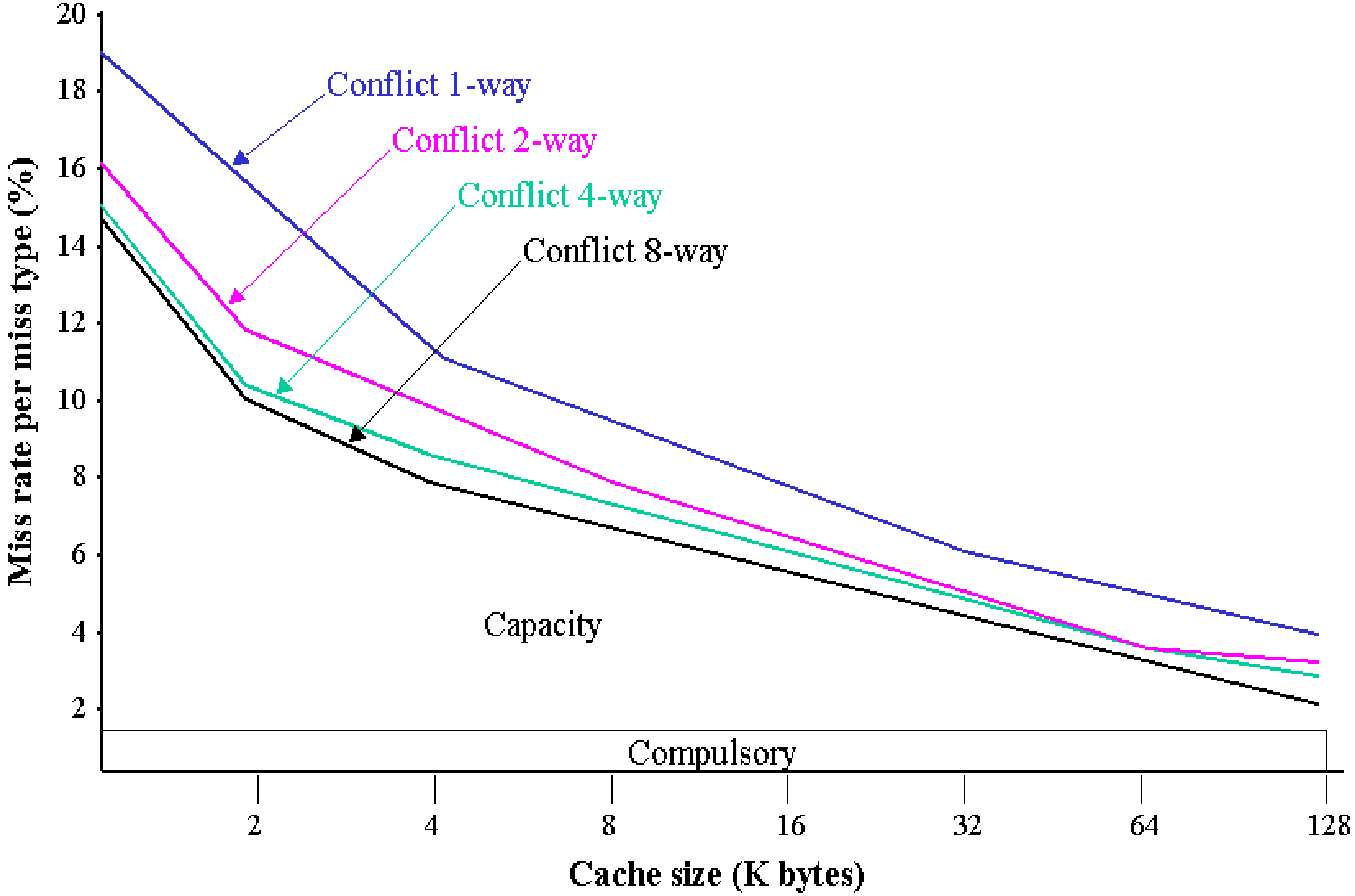
- *unique place* - direct mapped, e.g. cache
  simple, fast access, high miss rate
- *any place* - fully associative, e.g. paged memory
  low miss rate, costly, slow (search everywhere)
- *any n places* - n-way set associative, e.g. TLB
  - contains several sets, each n blocks
  - set with given block = (block number) **mod** (number of sets)
  - for fixed number of blocks, increased associativity leads to more blocks per set

# Classify misses

- compulsory: first access to block after initialisation
- capacity: insufficient cache space
- conflict: rigid placement scheme

| decision | effect on miss rate | possible negative effect |
|---|---|---|
| ↑ size | ↓ capacity misses | ↑ access time, ↑ hardware |
| ↑ associativity | ↓ conflict misses | ↑ access time, ↑ hardware |
| ↑ block size | ↓ compulsory misses | ↑ miss penalty |

# Miss rate classification



# Virtual memory: summary

- motivations: caching; running large programs; running multiple programs; protection
- page table: virtual to physical / disk address
- large page size: reduce miss rate
- place page anywhere: fully associative
- page replacement: LRU, write back, dirty bit
- TLB: page table cache

## Summary: what to do on a write hit?

- write through
  - write to corresponding blocks at both upper and lower level
  - read misses do not require writes to the lower level
  - easy to implement
  - may require write buffer in high-speed systems
- write back
  - only write to upper level: at upper level speed
  - modify lower level block during replacement: minimise access overhead
  - better use of wide lower level, since write entire block
  - preferred choice when speed difference between the 2 levels is large, e.g. main memory and disk
  - more complicated to implement

## Effect of cache size

- r: miss rate, d: data reference per instruction
  p: miss penalty (time) t: clock cycle time
  n: instruction count c: CPI without stall

- CPU time $\tau$ = (CPU execution cycles + memory stall cycles) $\times$ t

$$= ( nc + n \times ( \underset{\text{instruction miss rate}}{r} + \underset{\text{data miss rate}}{rd} ) \times \lceil p/t \rceil ) \times t$$

- 2 caches: large $r_1,t_1$; small $r_2,t_2$ $\qquad r_1<r_2,\ t_1>t_2$

- large cache slower if: $ct_1 + r_1p(1+d) > ct_2 + r_2p(1+d)$

$$c > \frac{p(1+d)(r_2-r_1)}{t_1-t_2}$$

# Summary

| | TLB | L1 cache | virtual memory |
|---|---|---|---|
| hit time | 1 cycle | 1-15 cycles | 10-100 cycles |
| miss penalty | 10-1000 cycles | 10-25 cycles | $10^7$-$10^8$ cycles |
| placement | fully associative or set associative | direct mapped or set associative | fully associative |
| identification | tag / block | tag / block | table |
| replacement | random | direct mapped / random | LRU |
| write | flush on write to page table | write back / write through | write back |