Computer Architecture
Chapter 5: Virtual memory

Phạm Quốc Cường

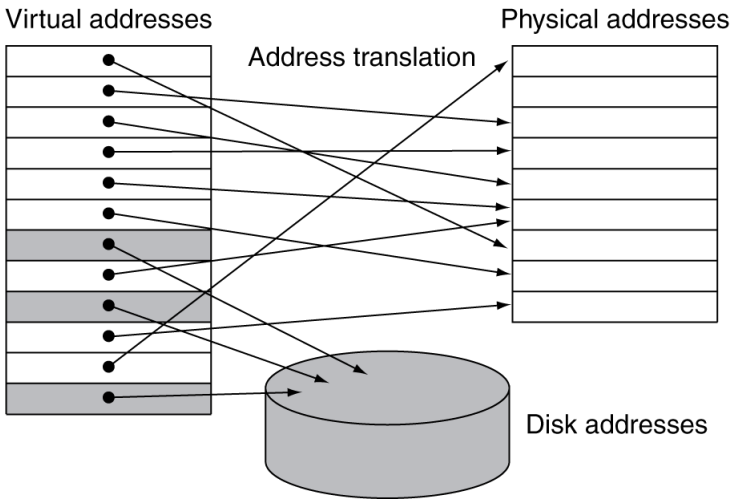
Adapted from Computer Organization the Hardware/Software Interface – 5th

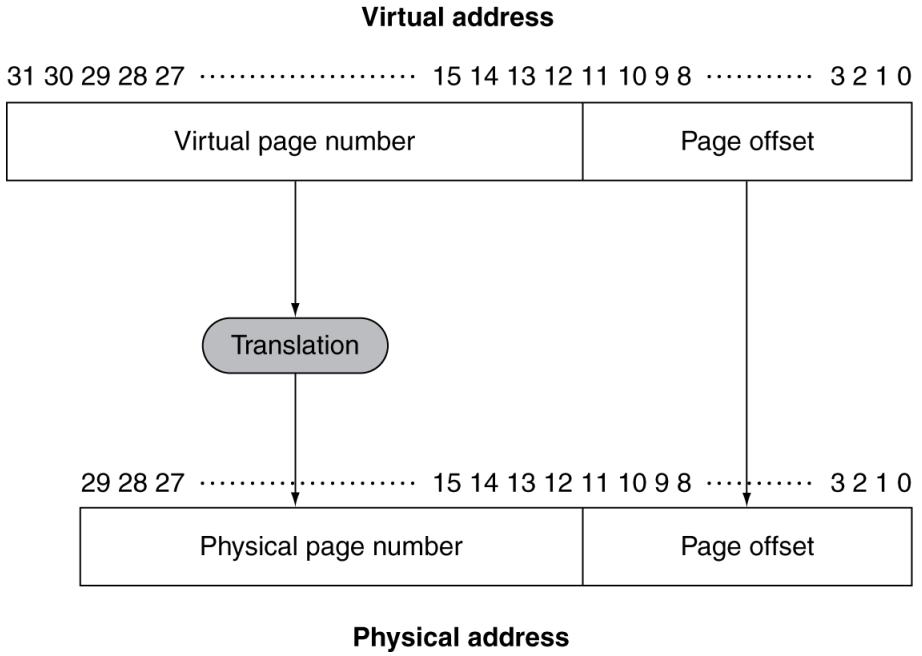
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Address Translation

- Fixed-size pages (e.g., 4K)





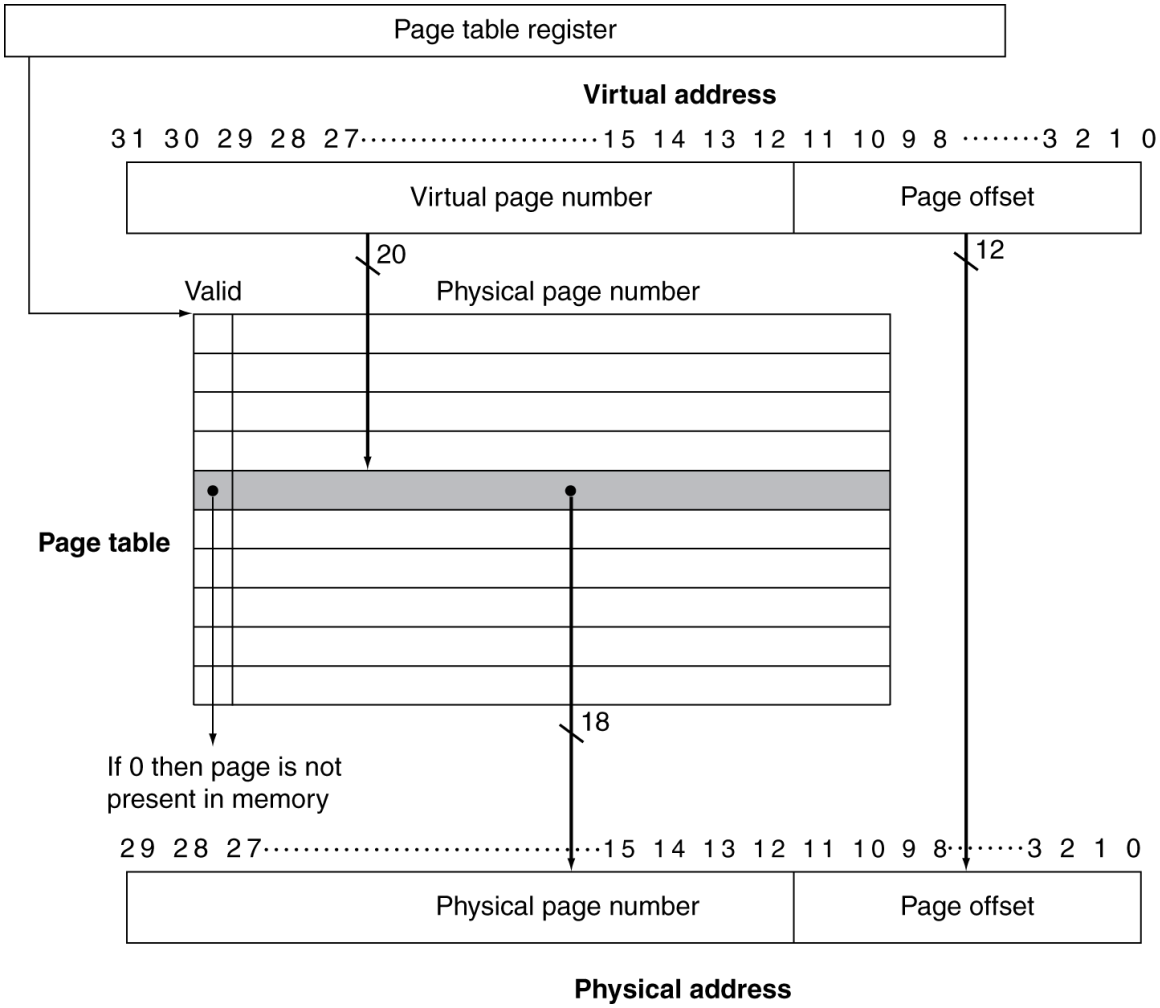
Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

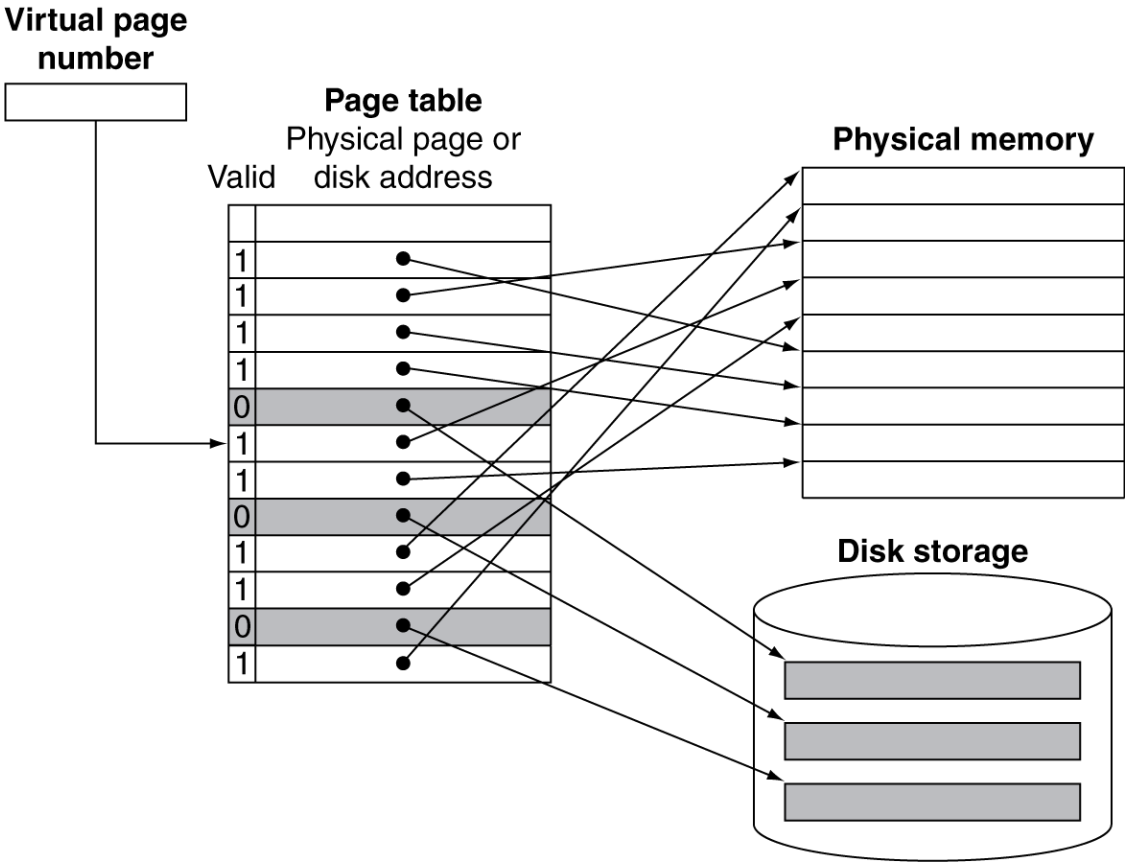
Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk

Translation Using a Page Table



Mapping Pages to Storage



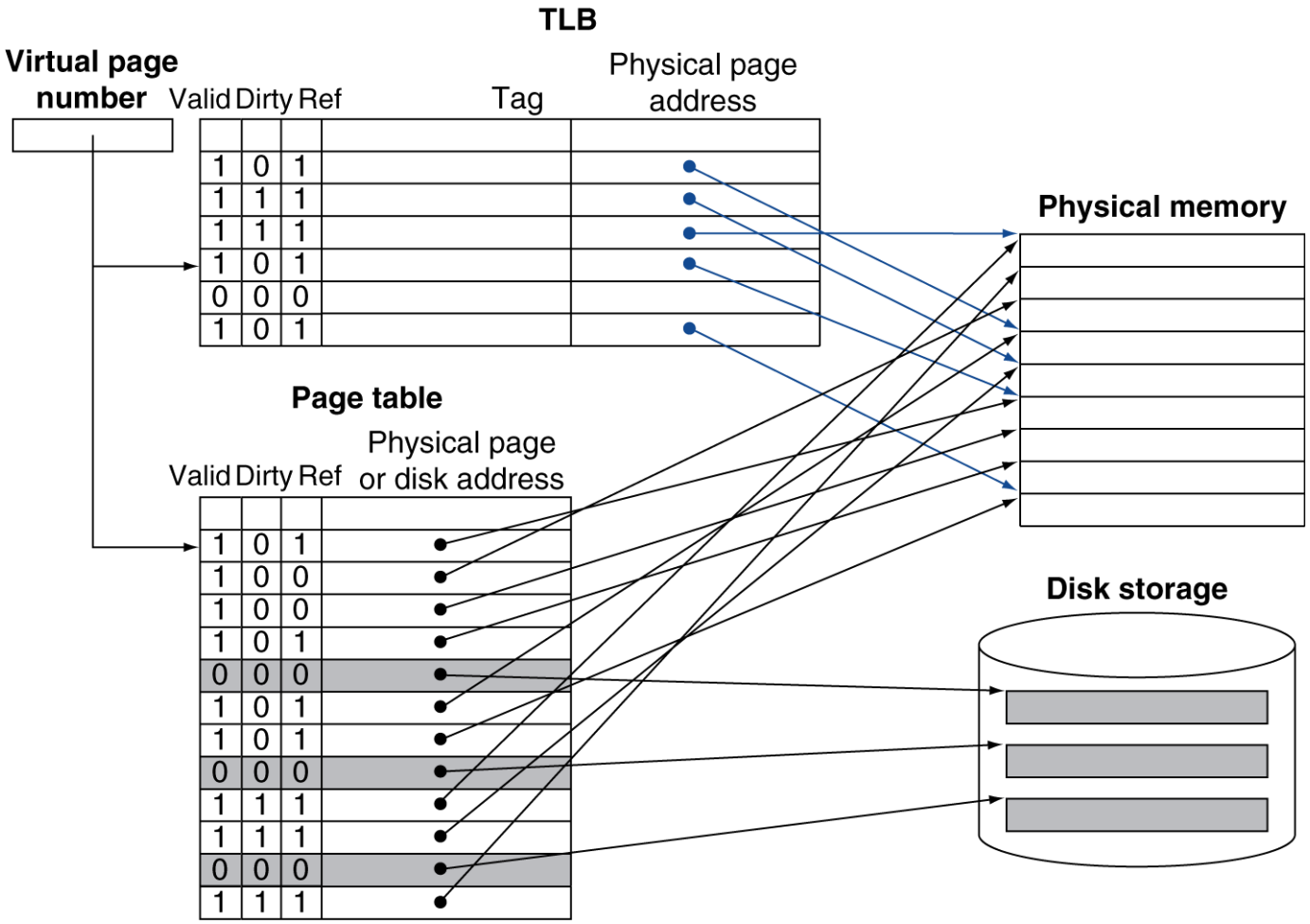
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Fast Translation Using a TLB

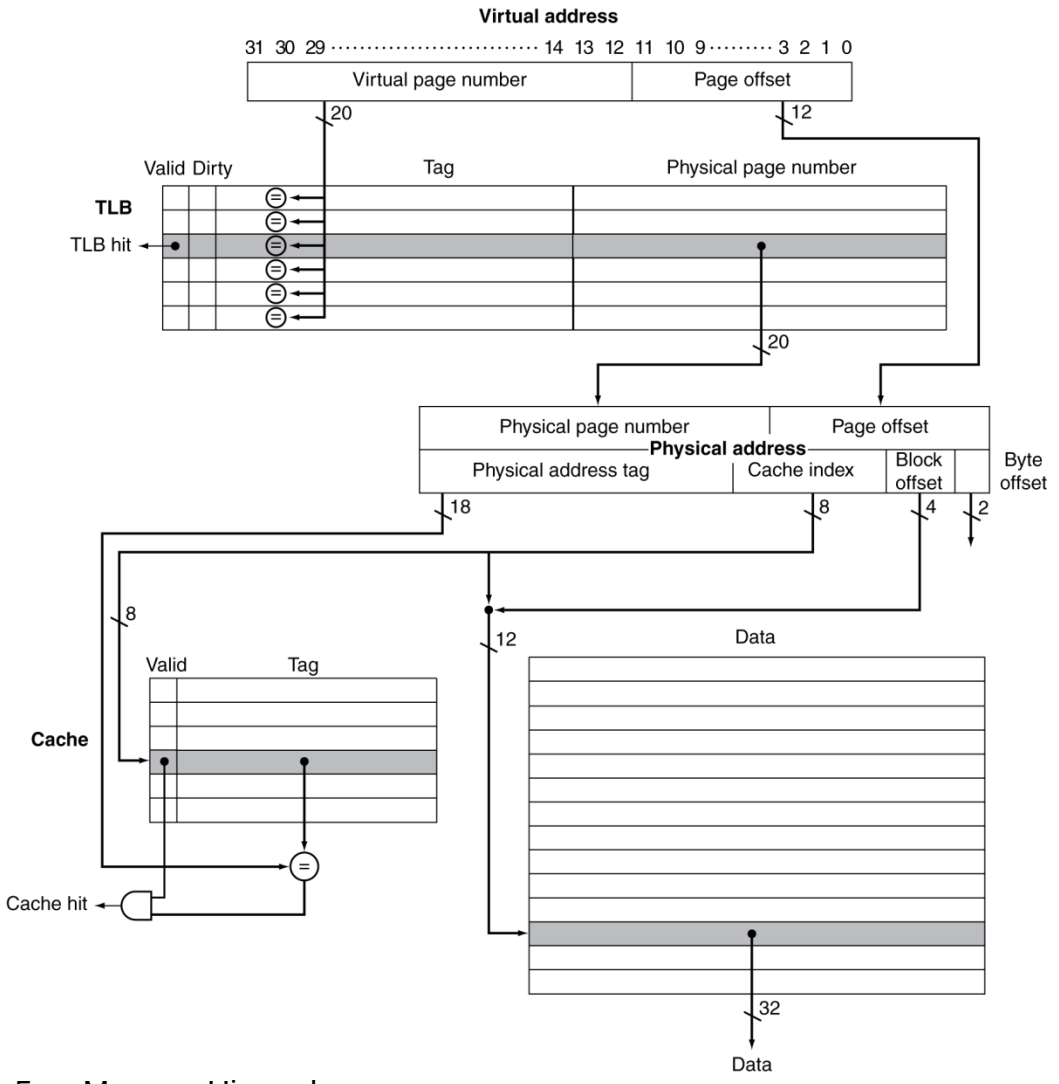
- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

Fast Translation Using a TLB



TLB and Cache Interaction

- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address



TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not preset
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

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

Exercise

- Given the TLB (fully associative) and the Page table (4KB pages) with LRU replacement
- If pages must be brought from disk, increment the next largest page number

a. Show the final state of the TLB and Page table if virtual address requests are as follow:
   - 4669, 2227, 13916, 34587, 48870, 12608, 49225
   - 12948, 49419, 46814, 13975, 40004, 12707, 52236

b. The same question but 16KB pages instead of 4KB

V	Physical or Disk
1	5
0	Disk
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	3
1	12

V	Tag	Physical
1	11	12
1	7	4
1	3	6
0	4	9

Hint: Analyse the virtual address to extract virtual page number

Solution

- Virtual address (decimal): 4669, 2227, 13916, 34587, 48870, 12608, 49225
- Binary address: 4KB pages => 12-bit page offset
  - 4669 = 1_0010_0011_1101, VPN = 1
  - 2227 = 0_1000_1011_0011, VPN = 0
  - 13916 = 11_0110_0101_1100, VPN = 3
  - 34587 = 1000_0111_0001_1011, VPN = 8
  - 48870 = 1011_1110_1110_0110, VPN = 11
  - 12608 = 11_0001_0100_0000, VPN = 3
  - 49225 = 1100_0000_0100_1001, VPN = 12

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669	1	TLB miss PT hit PF	1	11	12
			1	7	4
			1	3	6
			1 (last access 0)	1	13
2227	0	TLB miss PT hit	1 (last access 1)	0	5
			1	7	4
			1	3	6
			1 (last access 0)	1	13
13916	3	TLB hit	1 (last access 1)	0	5
			1	7	4
			1 (last access 2)	3	6
			1 (last access 0)	1	13
34587	8	TLB miss PT hit PF	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 0)	1	13
48870	11	TLB miss PT hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 4)	11	12
12608	3	TLB hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	11	12
49225	12	TLB miss PT miss	1 (last access 6)	12	15
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	11	12

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669	0	TLB miss PT hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 0)	0	5
2227	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 1)	0	5
13916	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 2)	0	5
34587	2	TLB miss PT hit PF	1 (last access 3)	2	13
			1	7	4
			1	3	6
			1 (last access 2)	0	5
48870	2	TLB hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			1 (last access 2)	0	5
12608	0	TLB hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			1 (last access 5)	0	5
49225	3	TLB hit	1 (last access 4)	2	13
			1	7	4
			1 (last axxess 6)	3	6
			1 (last access 5)	0	5

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

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

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

Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Pseudo-LRU
    - Close to LRU, less costly hardware
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

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 disk write latency

Sources of Misses

- 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 entries in a set
  - Would not occur in a fully associative cache of the same total size

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
- Memory system design is critical for multiprocessors