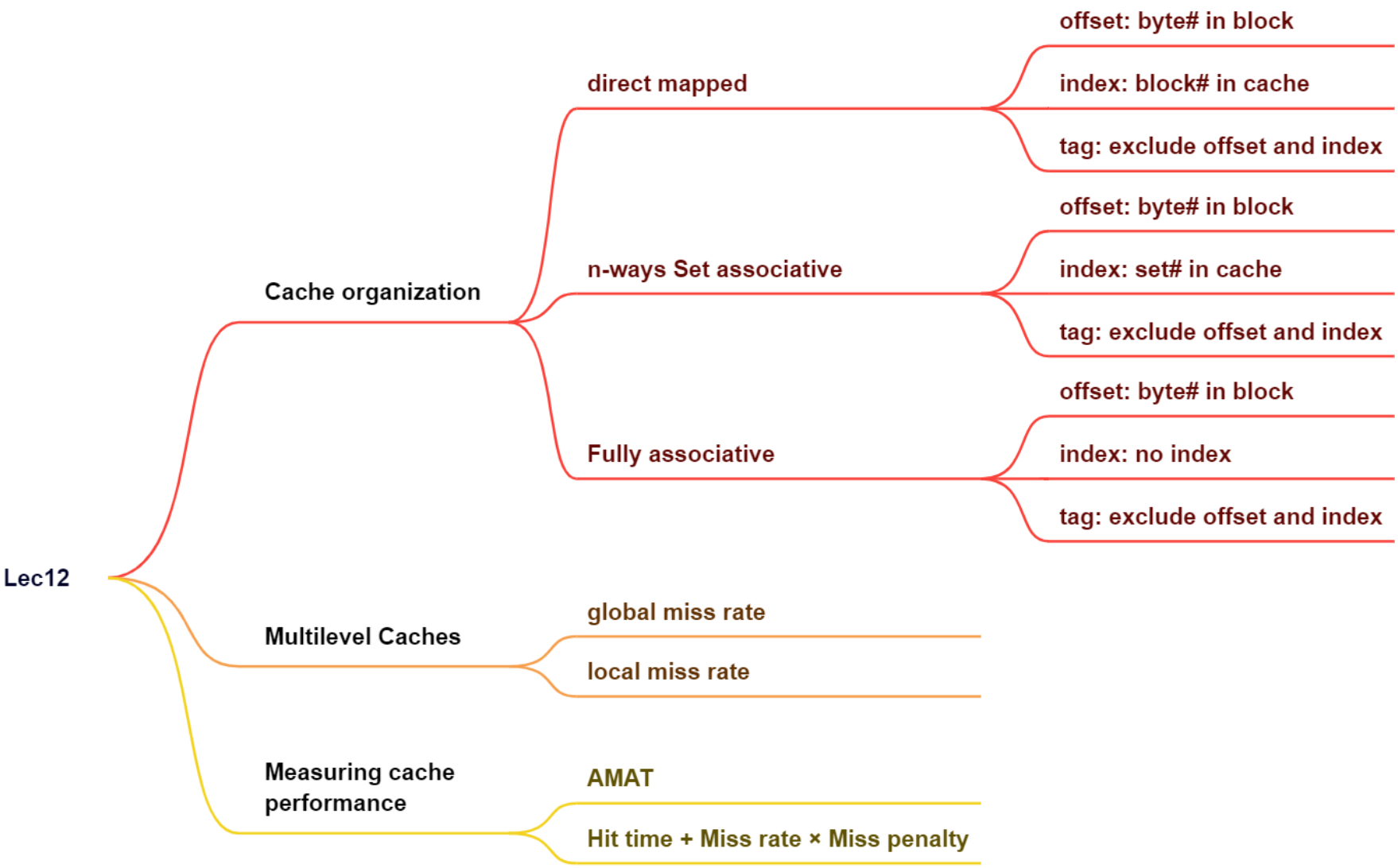
COMPUTER ORGANIZATION

Lecture 13
Memory Hierarchy (3)

2024 Spring

Recap

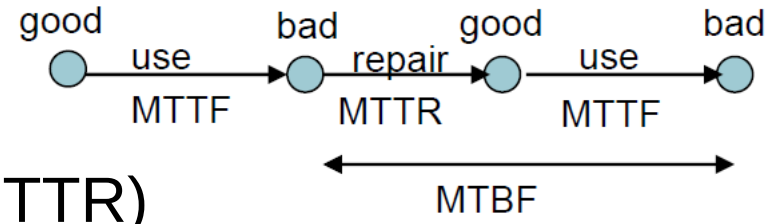

Lec12
Cache organization
direct mapped
offset: byte# in block
index: block# in cache
tag: exclude offset and index
n-ways Set associative
offset: byte# in block
index: set# in cache
tag: exclude offset and index
Fully associative
offset: byte# in block
index: no index
tag: exclude offset and index
Multilevel Caches
global miss rate
local miss rate
Measuring cache performance
AMAT
Hit time + Miss rate × Miss penalty

Outline

- **Hamming code**
- Virtual memory
- Memory hierarchy summary
- Virtual machine

Dependability Measures

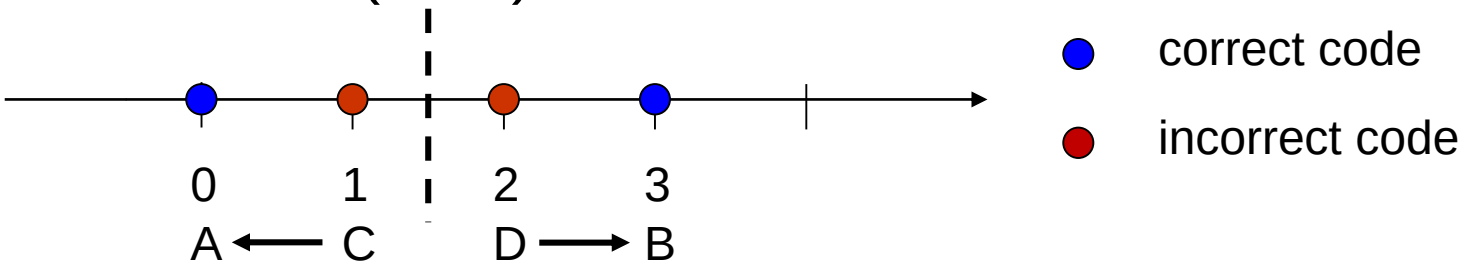
- Reliability: mean time to failure (MTTF)
  - e.g. MTTF of some disks is around 1,000,000-hour (114 years)
  - e.g. MTTF of fan is around 70,000-hour (8 years)
- Mean time to repair (MTTR)
  - the average time required to repair a failed system
- Mean time between failures
  - MTBF = MTTF + MTTR



- Availability = MTTF /(MTTF +MTTR)
  - increase MTTF
  - reduce MTTR
- “nines of availability” per year
  - One nine: 90% → 36.5 days of repair/year (2.4h/day)
  - Two nines: 99% → 3.65 days of repair/year (14.1 min/day)
  - Thee nines: 99.9% → 526 minutes of repair/year (1.4 min/day)

The Hamming SEC Code

- Hamming distance
  - minimum number of bits that are different between any two correct bit patterns
  - E.g.use 111 to represent 1, use 000 to represent 0, hamming distance(d) is 3, d = 3
- Minimum distance = 2 provides single bit error detection
  - E.g. parity code: 10→101, 11→110, d = 2
- Minimum distance = 3 provides **Single Error Correction** (SEC)



Encoding SEC

- To calculate Hamming code:
  - Number bits from 1 on the left
  - All bit positions that are a power of 2 are parity bits (bit 1 2 4 8 are parity bits)
  - Each parity bit checks certain data bits and:
    - p1 checks bits where rightmost bit of address 1
    - p2 checks bits where 2nd bit to the right in the address is 1
    - p4 checks bits where 3rd bit to the right in the address is 1
    - p8 checks bits where 4th bit to the right in the address is 1
  - Even parity

0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
		0	1	1	1	0	0	1	0	1	0	1	0
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

Decoding SEC

- Value of parity bits indicates which bits are in error
  - Use numbering from encoding procedure
  - Example: detect error in sequence 011100101110
    - Check parity bits' correctness
    - if parity bits checking result: 0000 indicates no error
    - In the example, parity bits checking result: 0101 indicates bit position 10 was flipped → corrected sequence: 011100101010
      - Pay attention to the bit position in SEC

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
		0	1	1	1	0	0	1	0	1	1	1	0
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
		0	0		1				1				
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

√ 0 means correct (p1 is 0)

X 1 wrong (p2 is 0 but not 1)

√ 0 correct (p4 is 1)

X 1 wrong (p8 is 1 but not 0)

SEC/DED Code

- Add an additional parity bit for the whole word (p_n)
- Make Hamming distance = 4, single error correction (SEC), 2 bit / double error detection (DED)
- Decoding:
  - Parity for p1 p2 p4 p8 correct, p_n correct
  → no error
  - Any of parity for p1 p2 p4 p8 incorrect, p_n incorrect
  → single bit error
  - Parity for p1 p2 p4 p8 correct, p_n incorrect
  → p_n bit error
  - Any of parity for p1 p2 p4 p8 incorrect, p_n correct
  → double bit error
- ECC DRAM uses SEC/DED with 8 bits protecting each 64 bits

Summary

- Cache Performance
  - Mainly depends on miss rate and miss penalty
- To improve cache performance:
  - Fully associative cache
  - Set associative cache
  - Replacement policy
  - Multilevel cache
- Dependability
  - MTTF, MTTR, reliability, availability
  - Hamming code: SEC/DED code

Outline

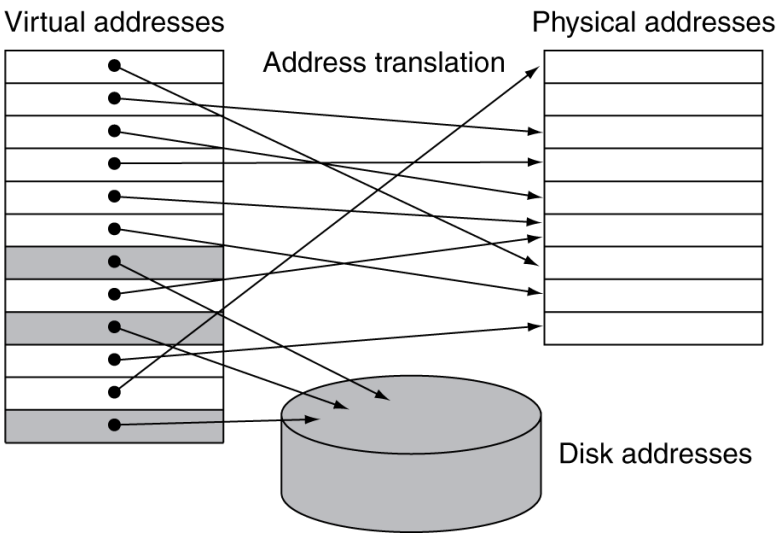
- Hamming code
- **Virtual memory**
- Memory hierarchy summary
- Virtual machine

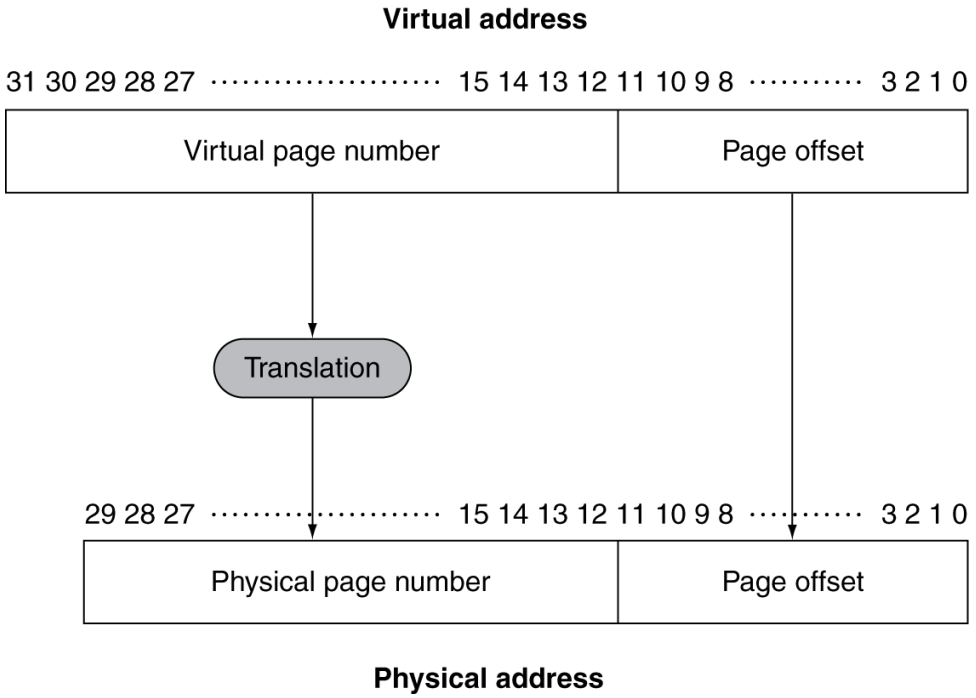
Virtual Memory

- 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)





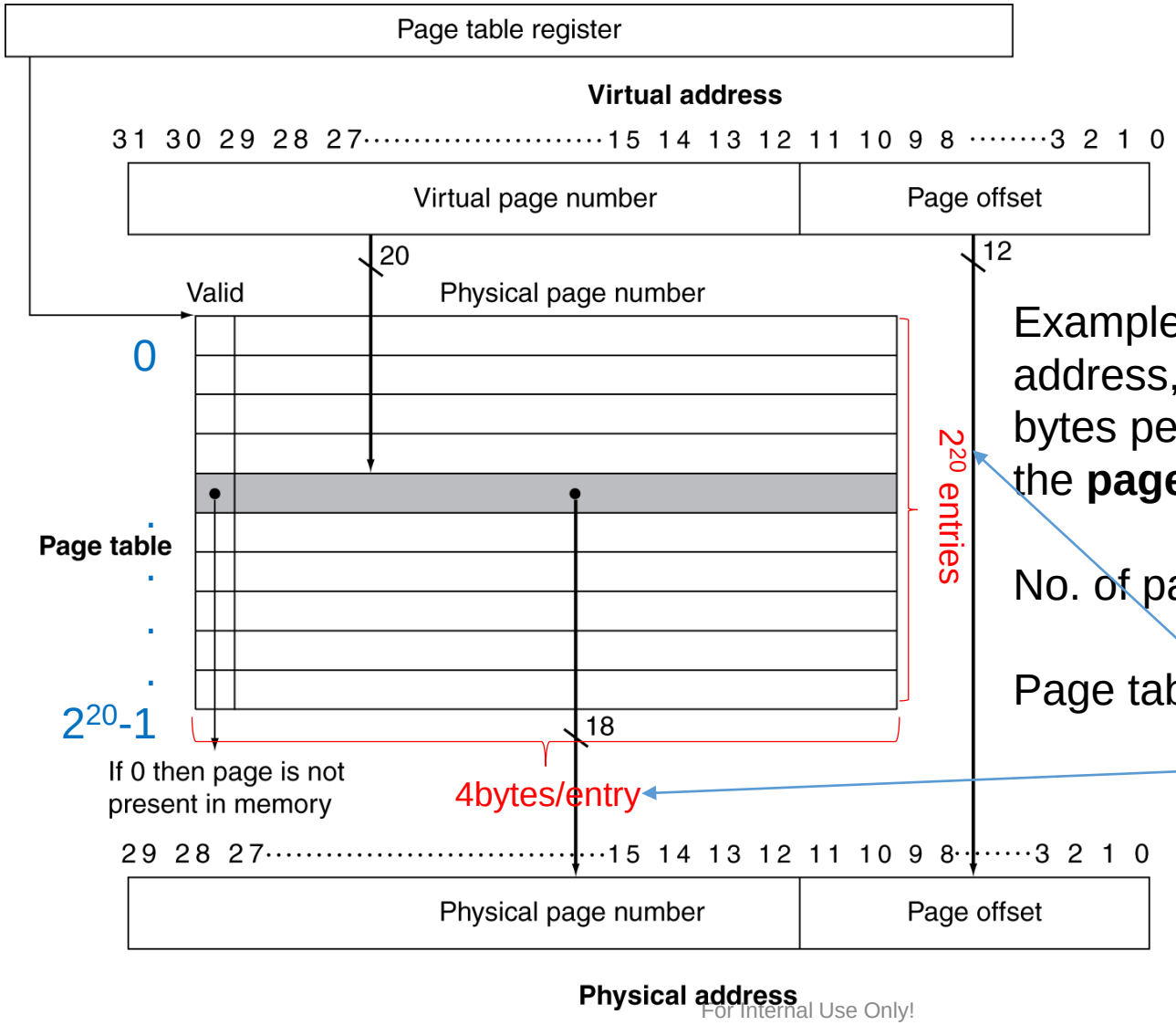
Page Tables

- 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
- Each program has its page table. Page table is in memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced bit, dirty bit, …)
- If page is not present
  - PTE can refer to location in swap space on disk

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

Translation Using a Page Table

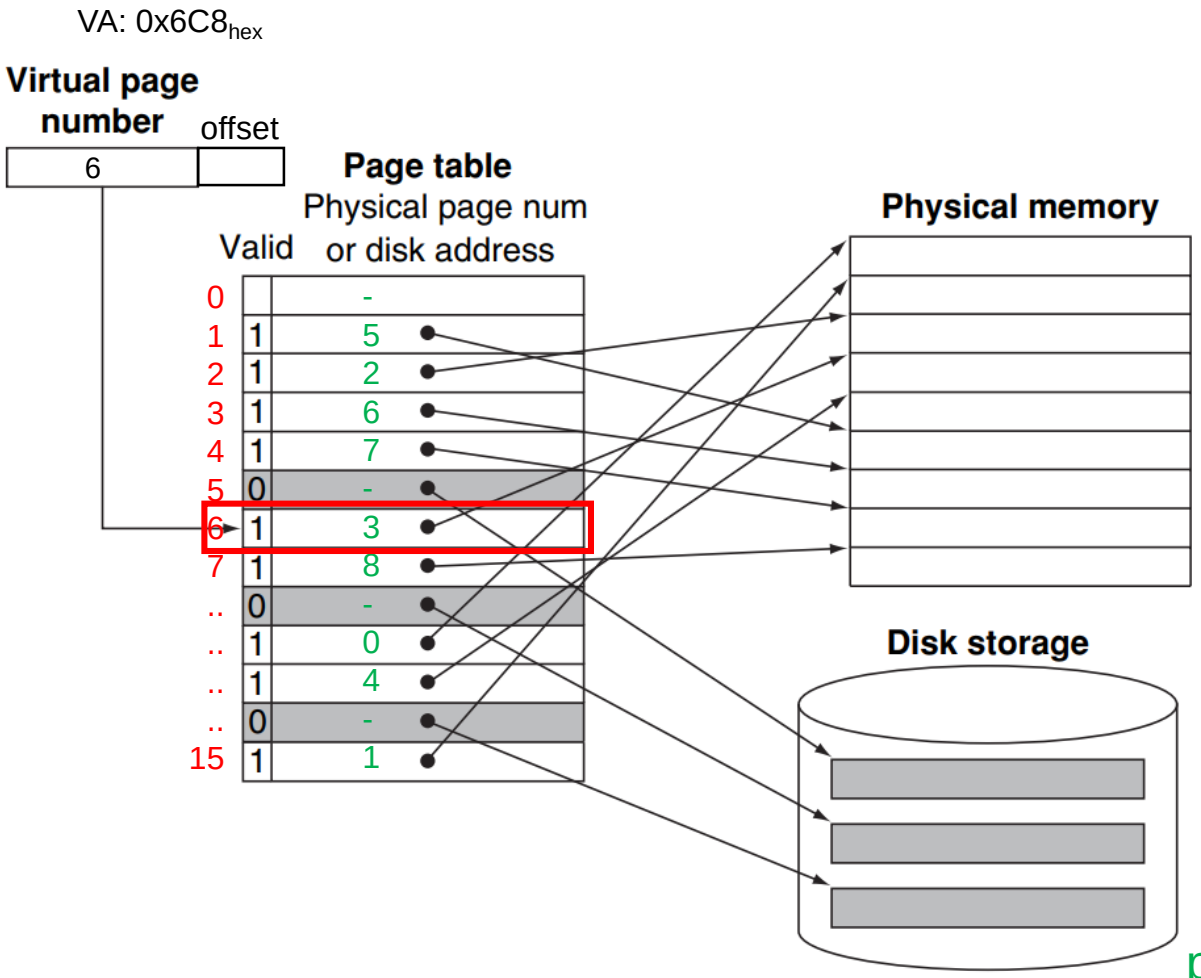


Example: With a 32-bit virtual address, 4 KB page size, and 4 bytes per page table entry, what's the **page table size**?

No. of page table entries
$= 2^{32}/2^{12} = 2^{20}$

Page table size
$= 2^{20} \times 4 = 4\text{MB}$

Mapping Pages to Storage



Example: **16 virtual pages, 256B/page**, virtual address: 12 bits.
To what physical address does virtual address 0x6C8 map to?

page offset: $\log_2(256B) = 8$ bits

$0x6C8_{hex} = 0110\,11001000$

vpn offset

Virtual page number = 6
Physical page number = 3
Physical address =
$1111001000 = 0x3C8_{hex}$

ppn offset

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
  - Use write-back, because write through is impractical
  - Dirty bit in PTE set when page is written

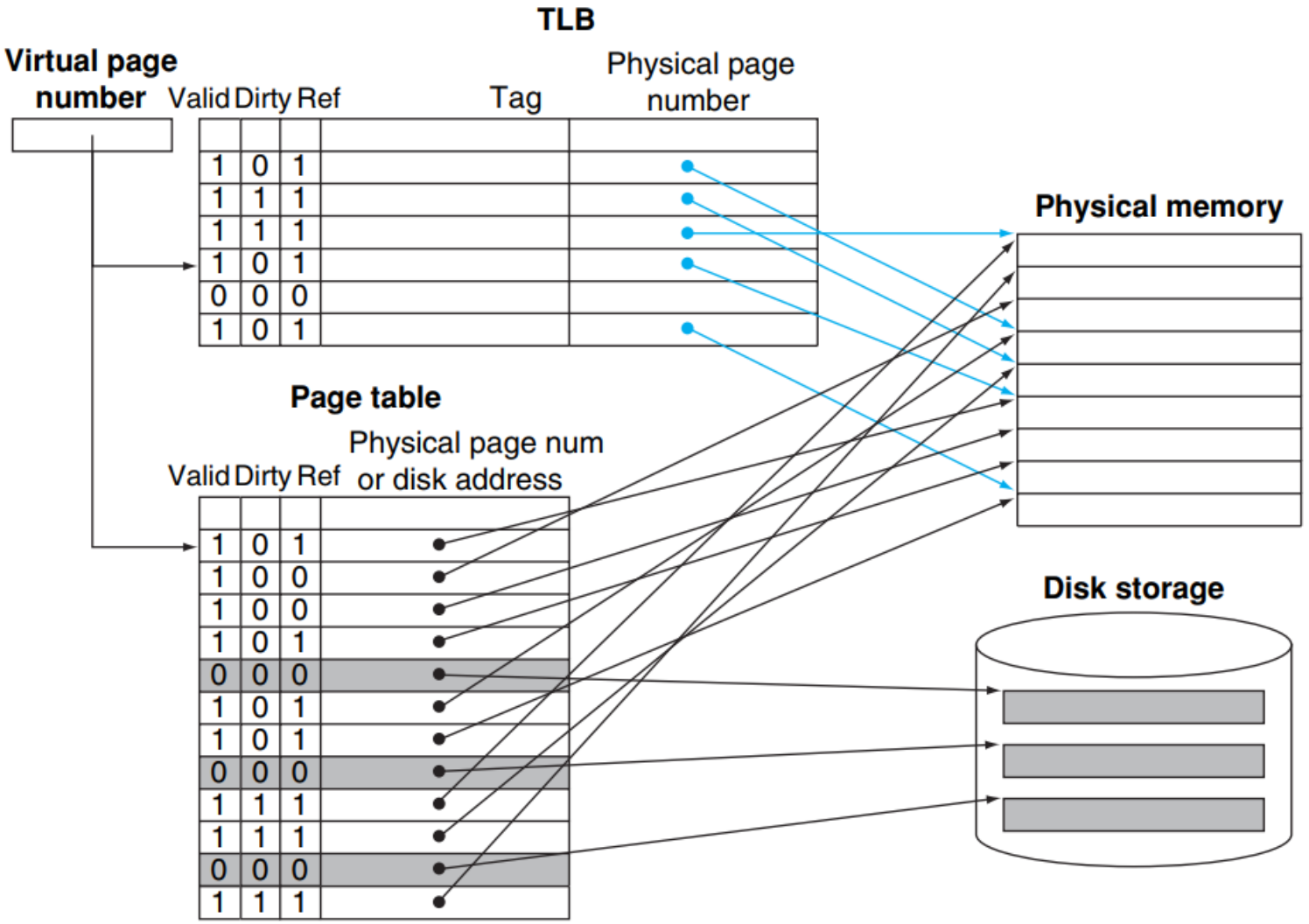
Replacement and Writes

- 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

- Problems of Page Table
  - Access to page table is too slow
    - First access the PTE (one main memory access)
    - Then the actual memory access for data
  - Page table is too big
- 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

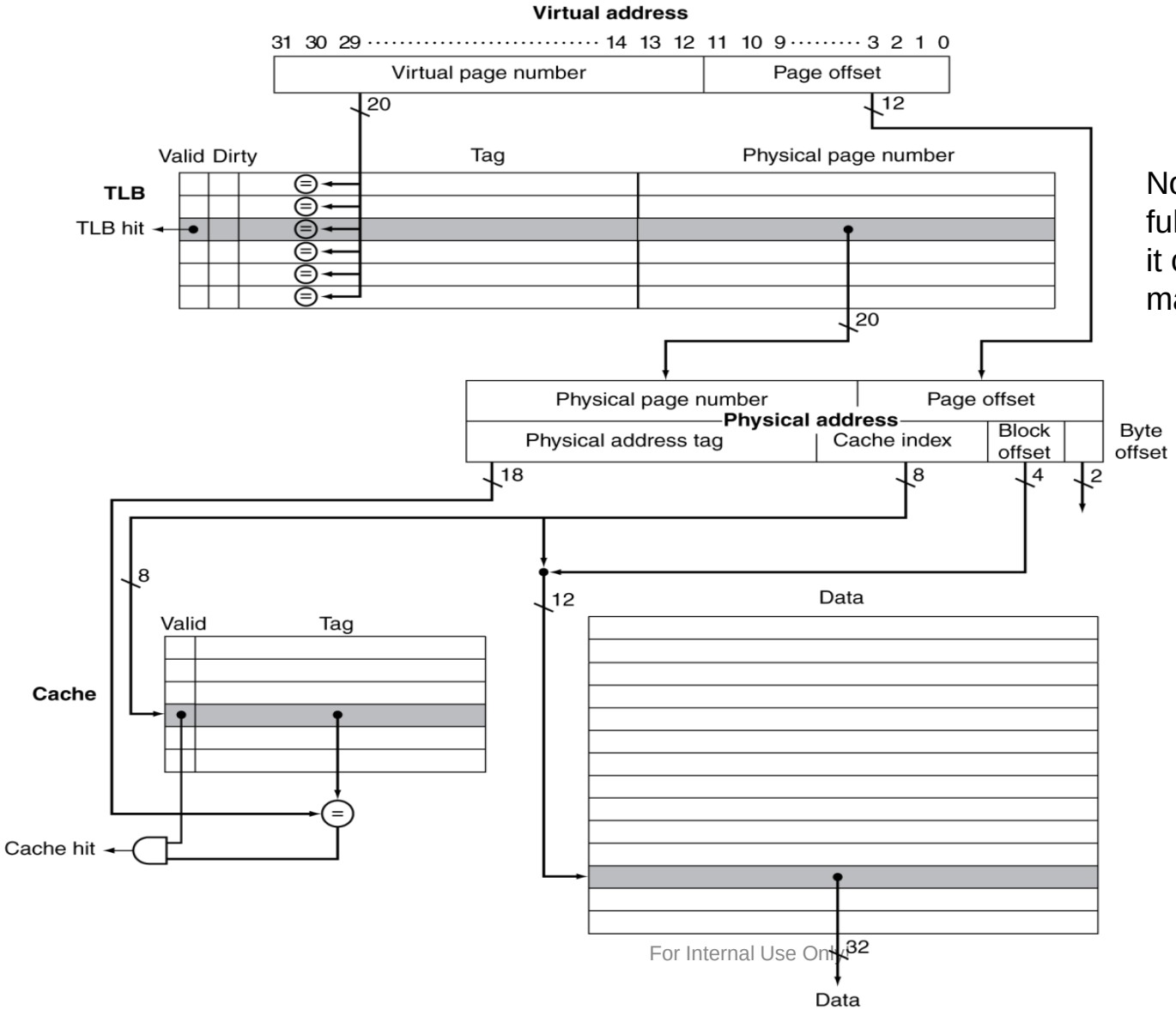
Fast Translation Using a TLB



TLB 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 (software)
  - Then restart the faulting instruction

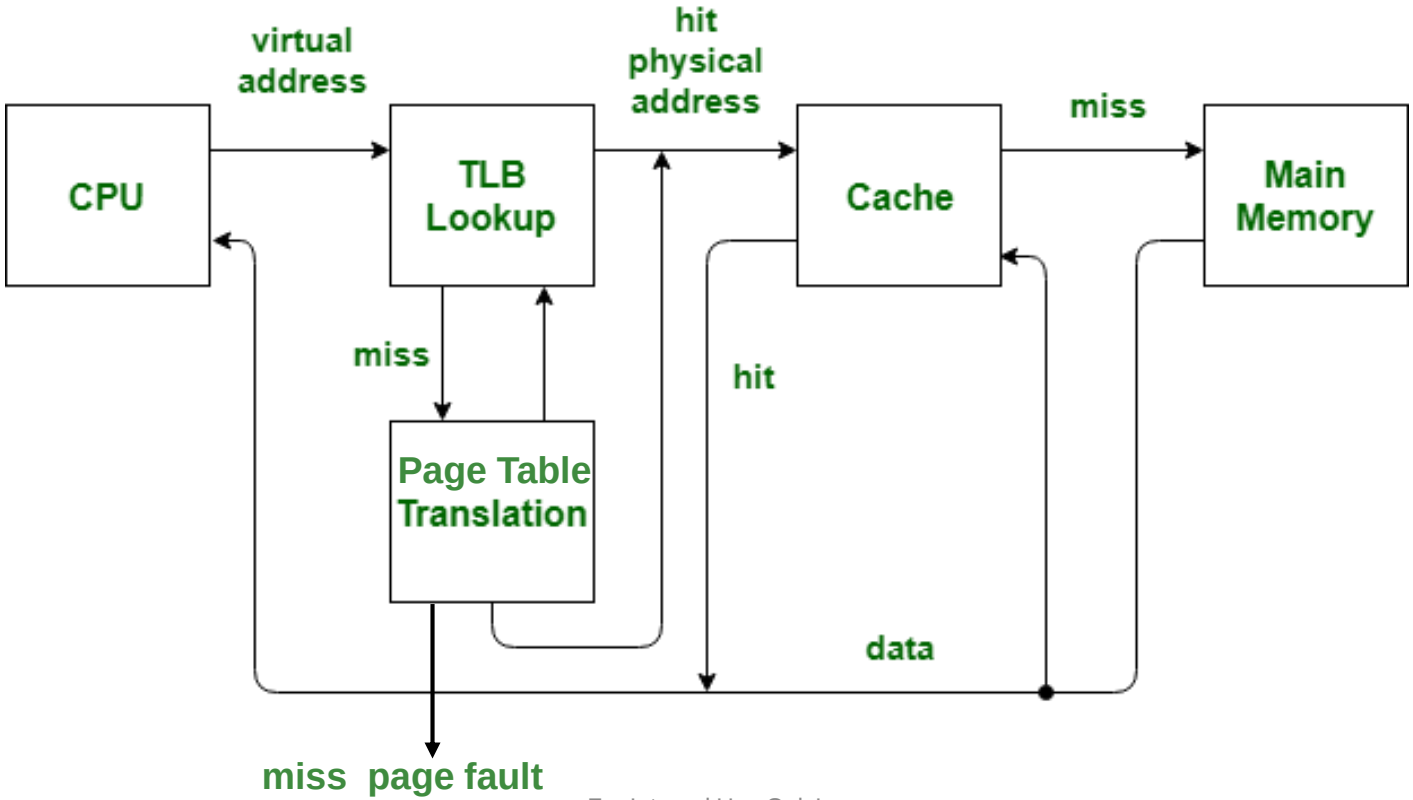
TLB and Cache Interaction



Noted: this example is a fully associative TLB, it can also be direct mapped or set associative

Making Address Translation Practical

- In VM, memory acts like a cache for disk
  - Page table maps virtual page numbers to physical frames
  - Use a page table cache for recent translation
    => Translation Lookaside Buffer (TLB)

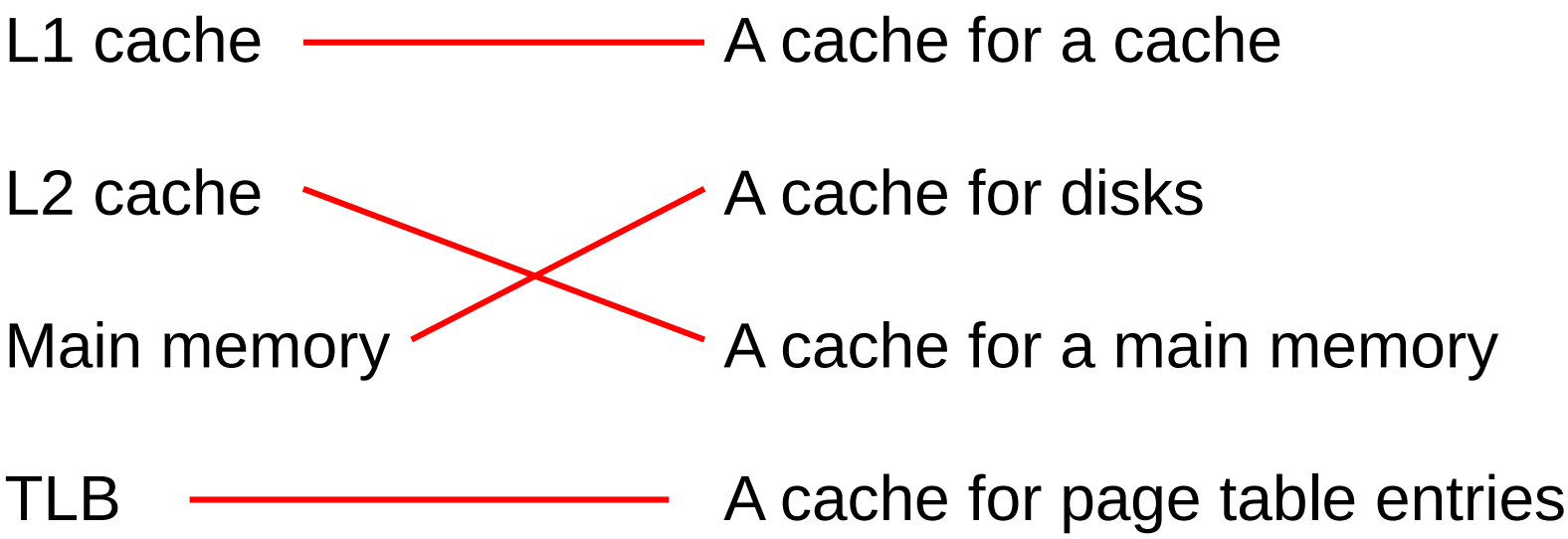


Memory Protection

- 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., ecall in RISC-V)

Check Yourself

- Match the definitions between left and right

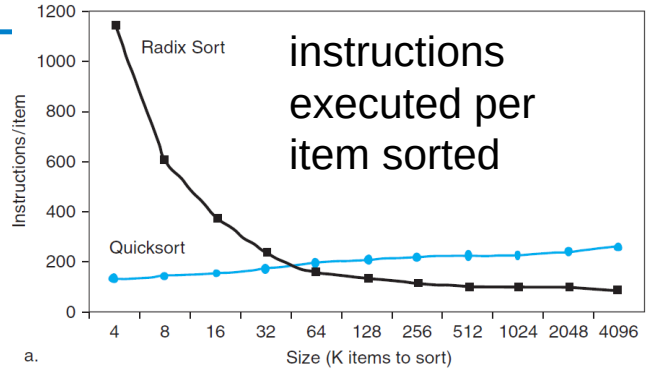


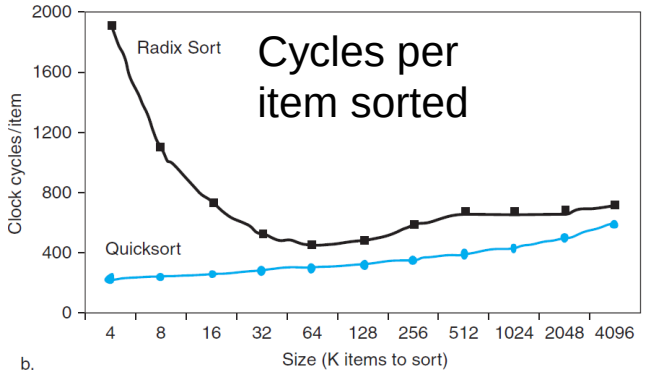
Outline

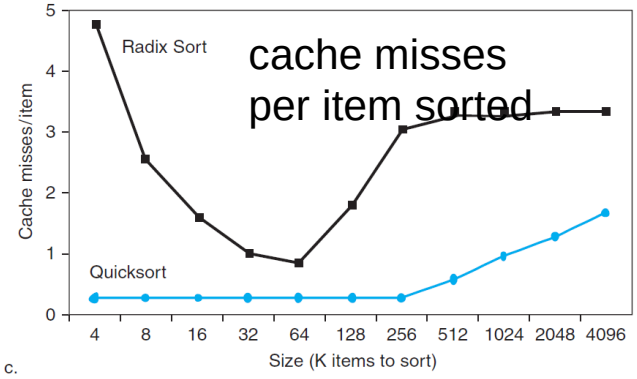
- Hamming code
- Virtual memory
- **Memory hierarchy summary**
- Virtual machine

Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access
- When #items increase,
  - Radix sort has less instructions
  - But quicksort has less clock cycles
  - Because miss rate of radix sort is higher

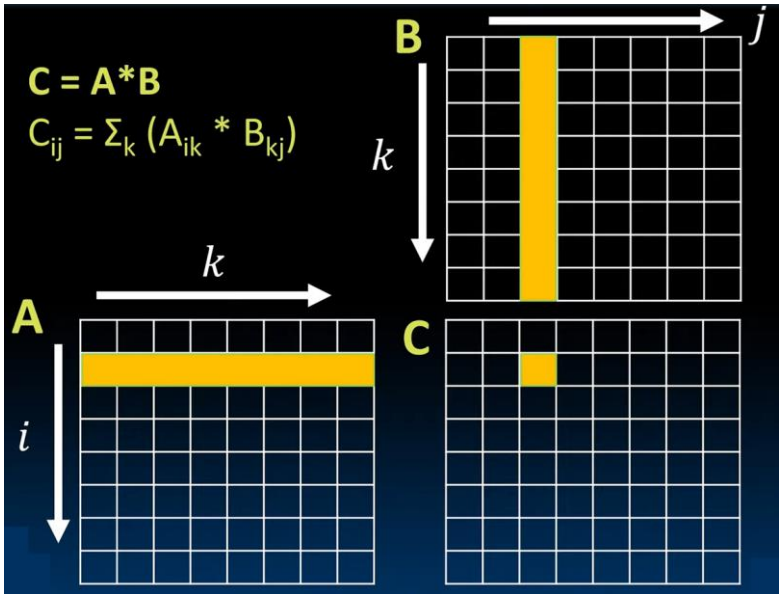






Software Optimization via Blocking

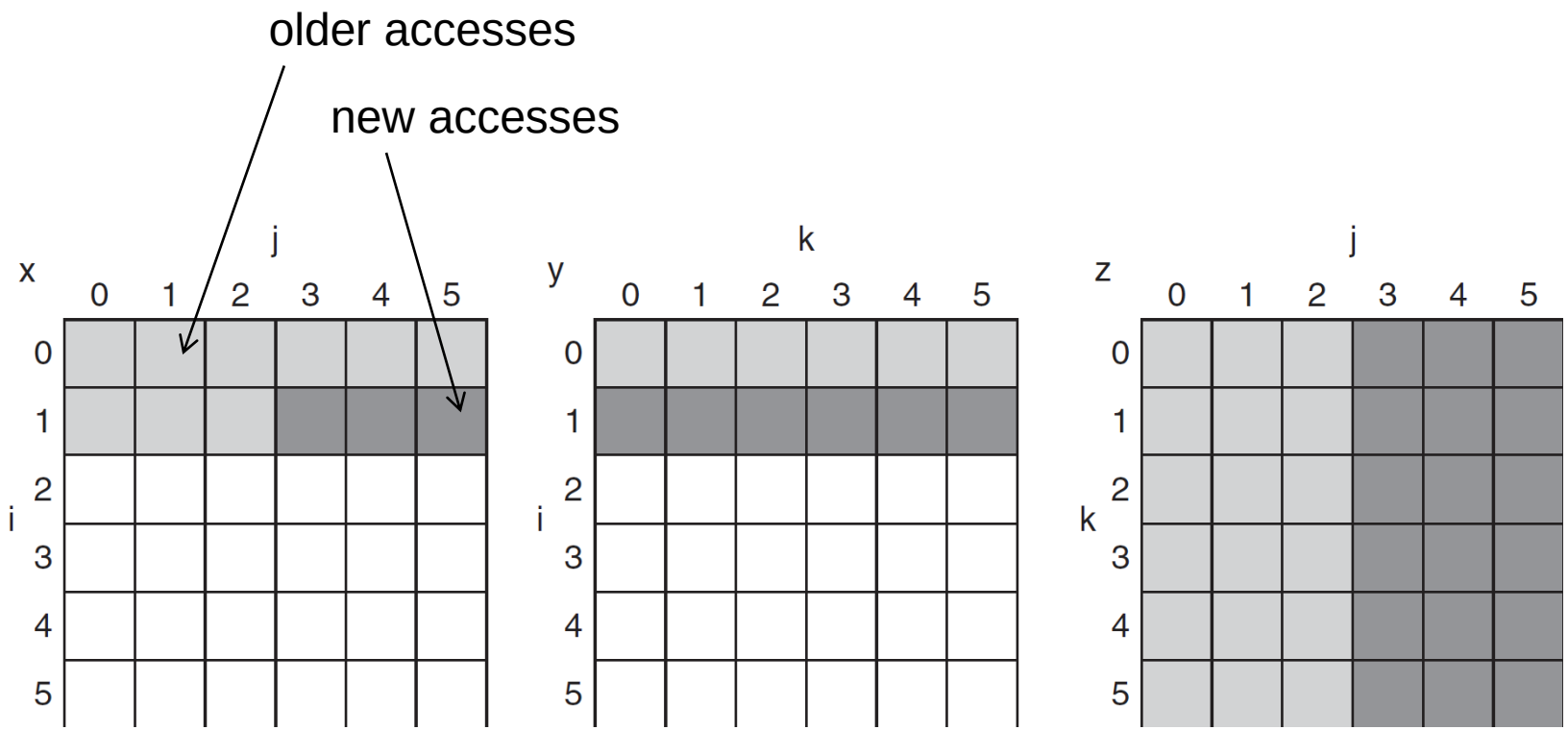
- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:



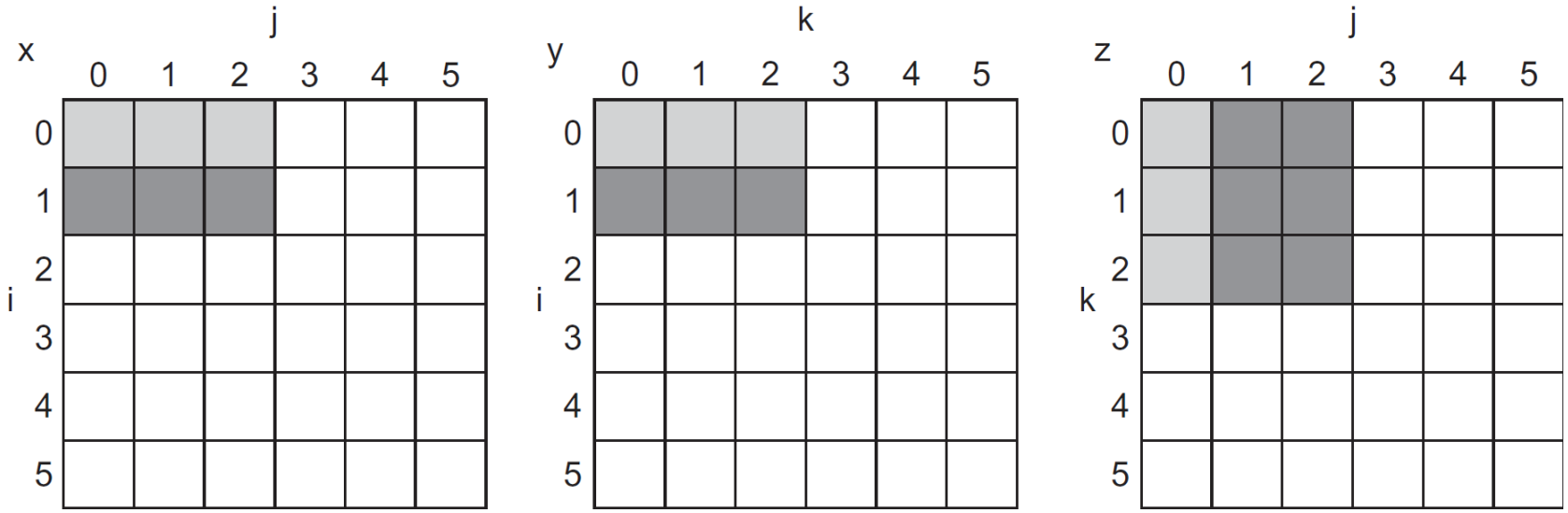

```
for (int j = 0; j < n; ++j)
 {
   double cij = C[i+j*n]; // cij = C[i][j]
   for( int k = 0; k < n; k++ )
     cij += A[i+k*n] * B[k+j*n]; // cij += A[i][j]*B[k][j]
   C[i+j*n] = cij; // c[i][j] = cij
 }
```

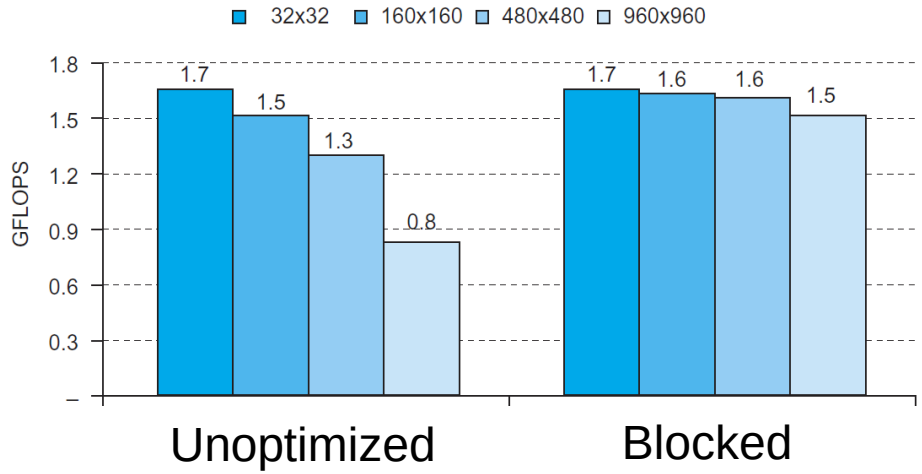
DGEMM Access Pattern

- C, A, and B arrays



Blocked DGEMM Access Pattern





The Memory Hierarchy Summary

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

Replacement

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

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

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	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

TLB, Page Table and Cache

- The possible combinations of events in the TLB, virtual memory system, and physically indexed (tagged) cache.

TLB	Page table	Cache	Possible? Condition?
Hit	Hit	Miss	
Miss	Hit	Hit	
Miss	Hit	Miss	
Miss	Miss	Miss	
Hit	Miss	Miss	
Hit	Miss	Hit	
Miss	Miss	Hit	

TLB, Page Table and Cache

- The possible combinations of events in the TLB, virtual memory system, and physically indexed (tagged) cache.

TLB	Page table	Cache	Possible? Condition?
Hit	Hit	Miss	Possible, but page table never checked if TLB hits
Miss	Hit	Hit	Possible, TLB miss but entry found in page table; after retry, data in cache
Miss	Hit	Miss	Possible, TLB miss but entry found in page table; after retry, data miss in cache
Miss	Miss	Miss	Possible, TLB miss and is followed by a page fault
Hit	Miss	Miss	Impossible, not in TLB if page not in memory
Hit	Miss	Hit	Impossible, not in TLB if page not in memory
Miss	Miss	Hit	Impossible, not in cache if page not in memory

Multilevel On-Chip Caches

Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

2-Level TLB Organization

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

Outline

- Hamming code
- Virtual memory
- Memory hierarchy summary
- **Virtual machine**

Virtual Machines

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources
- Virtualization has some performance impact
  - Feasible with modern high-performance comptuers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC
  - VirtualBox

Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest

Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs

Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
  - Current ISAs (e.g., x86) adapting

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
- Virtual memory and TLB