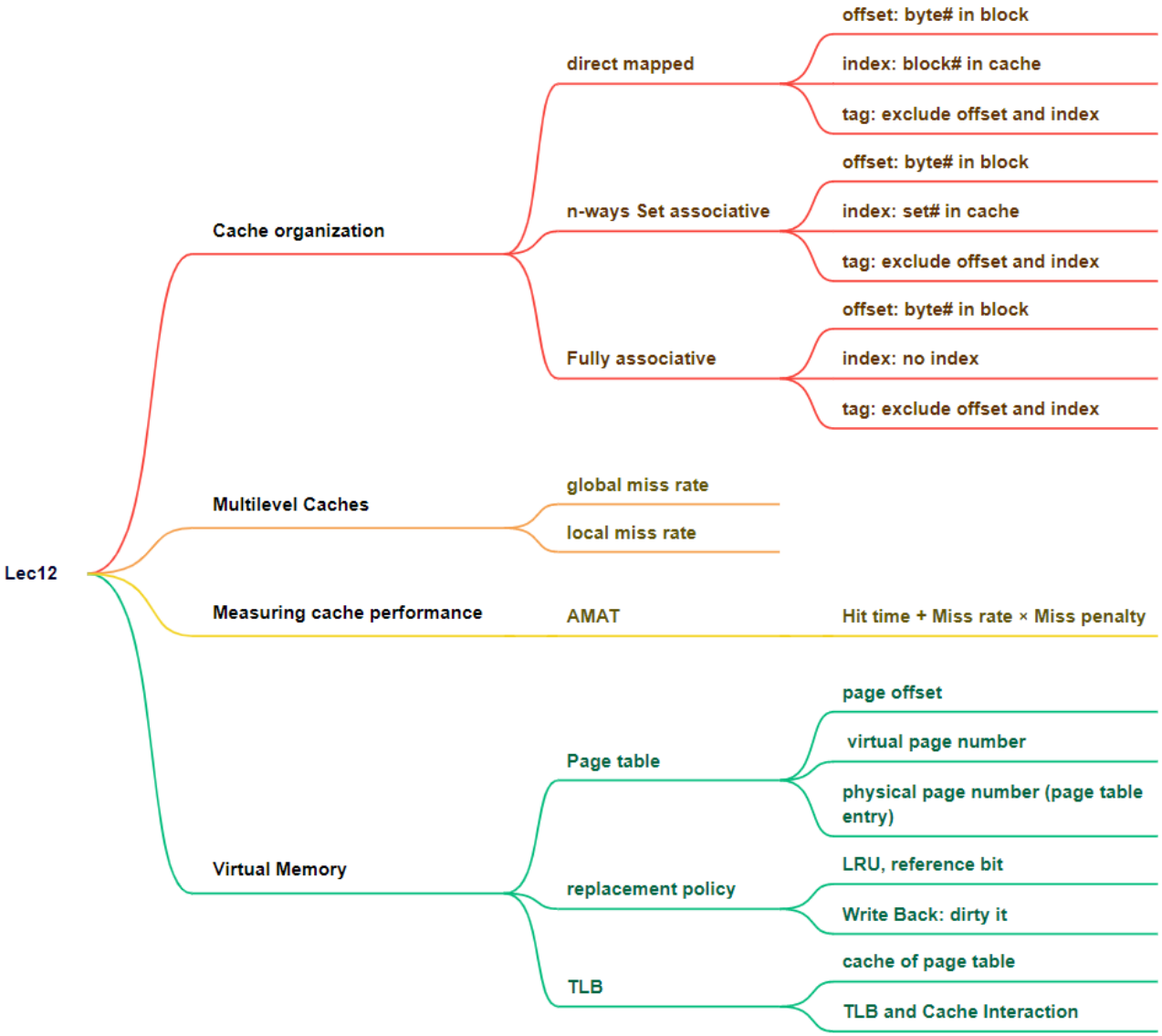
Lecture 13
Memory Hierarchy (3)

CS202 2023 Spring

Today's Agenda

- Recap
  - Measuring and improving cache performance
  - Virtual memory
- Context
  - Memory Hierarchy summary
  - Hamming Code
- Reading: Textbook 5.5, 5.6, 5.8
- Self-study: The rest section of Chapter 5

Recap

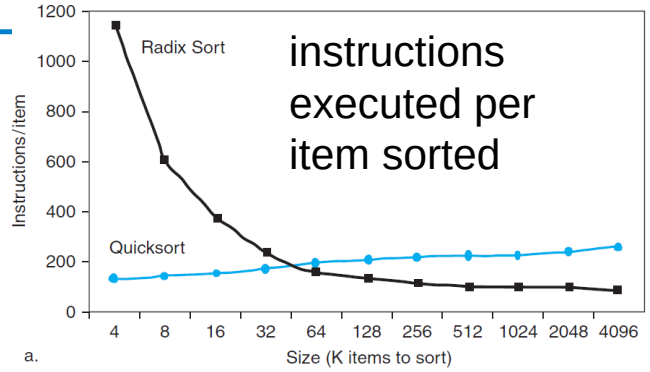

Lec12
Cache organization
direct mapped
offset: byte# in block
index: block# in cache
tag: exclude offset and index
n-ways Set associative
offset: byte# in block
index: set# in cache
tag: exclude offset and index
Fully associative
offset: byte# in block
index: no index
tag: exclude offset and index
Multilevel Caches
global miss rate
local miss rate
Measuring cache performance
AMAT
Hit time + Miss rate × Miss penalty
Virtual Memory
Page table
page offset
virtual page number
physical page number (page table entry)
replacement policy
LRU, reference bit
Write Back: dirty it
TLB
cache of page table
TLB and Cache Interaction

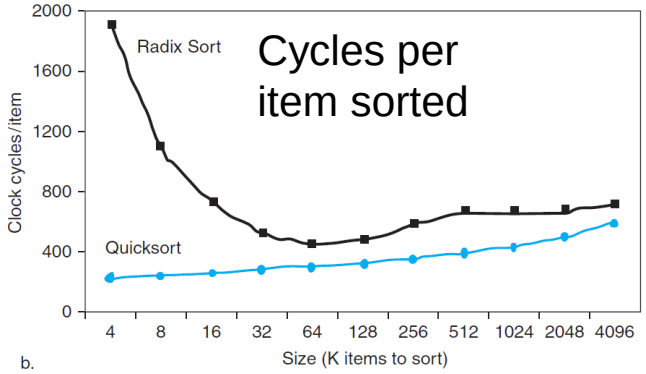
Outline

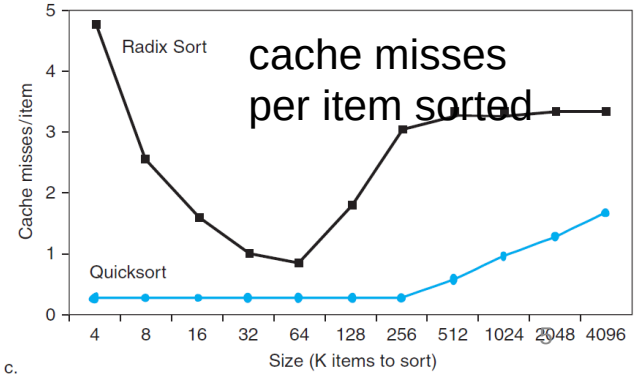
- **Memory hierarchy summary**
- Hamming code
- Virtual machine

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

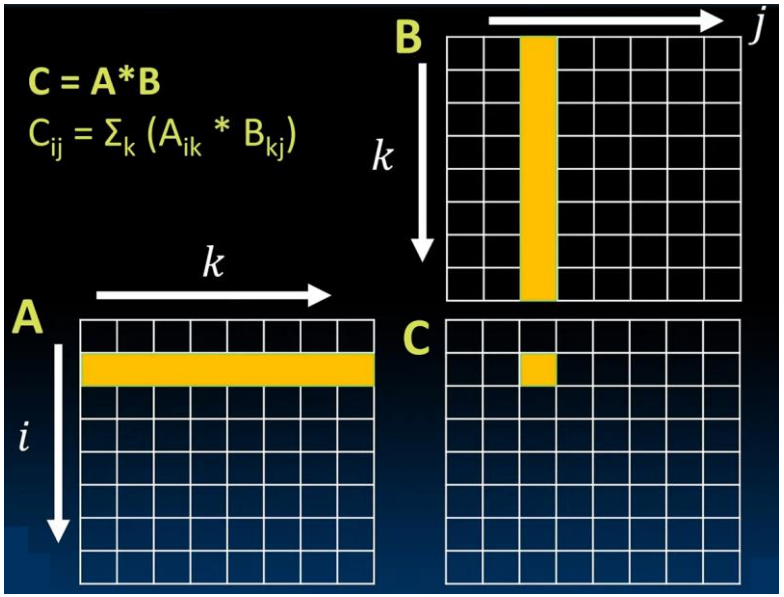






Software Optimization via Blocking

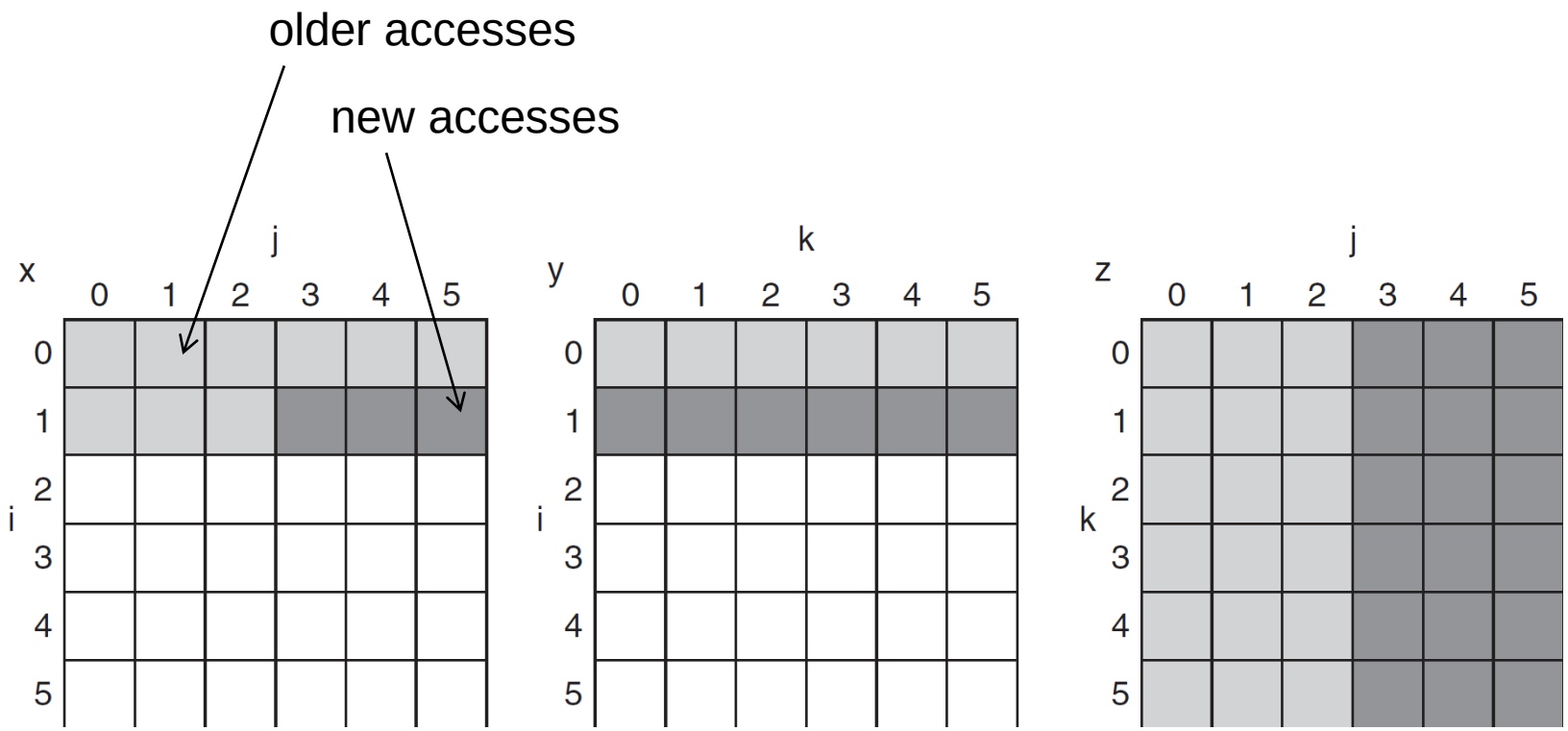
- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:



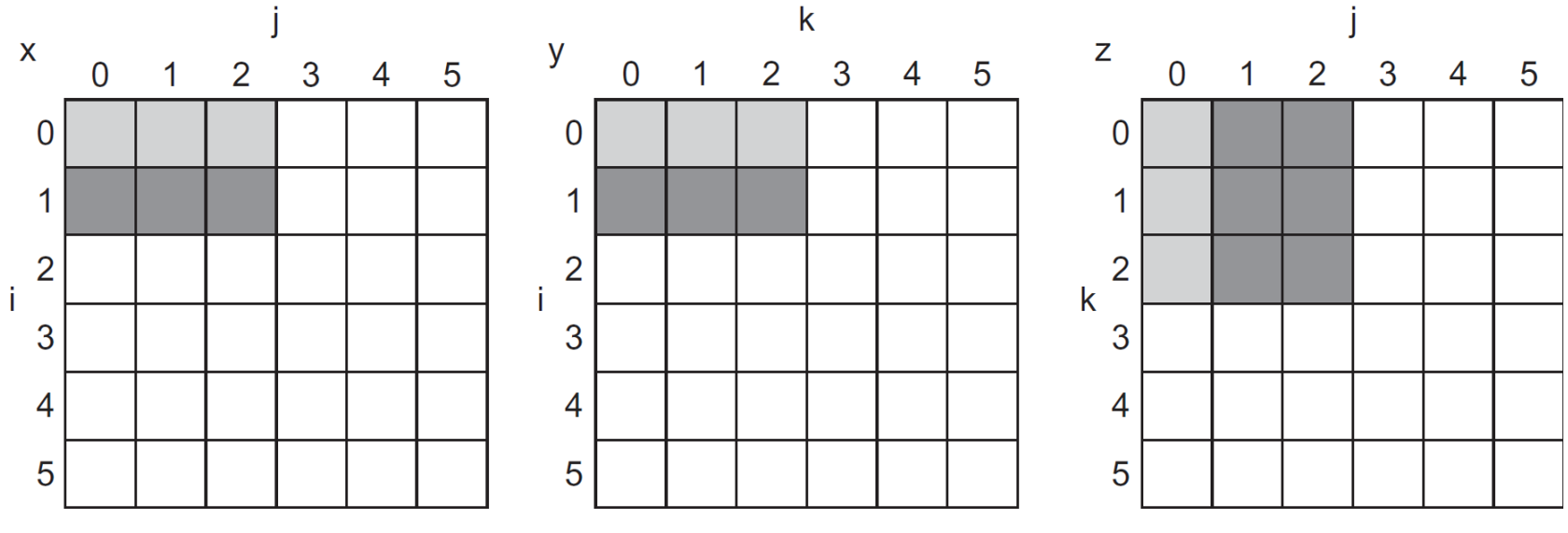

```
for (int j = 0; j < n; ++j)
 {
   double cij = C[i+j*n]; // cij = C[i][j]
   for( int k = 0; k < n; k++ )
     cij += A[i+k*n] * B[k+j*n]; // cij += A[i][j]*B[k][j]
   C[i+j*n] = cij; // c[i][j] = cij
 }
```

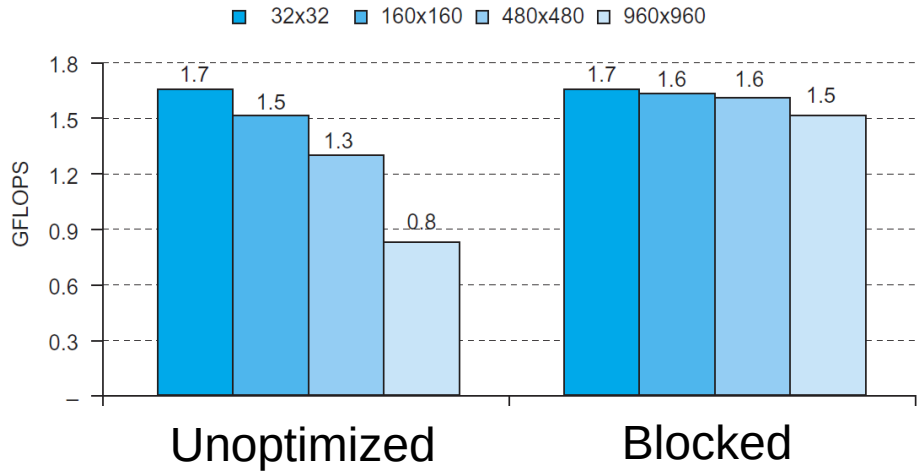
DGEMM Access Pattern

- 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.

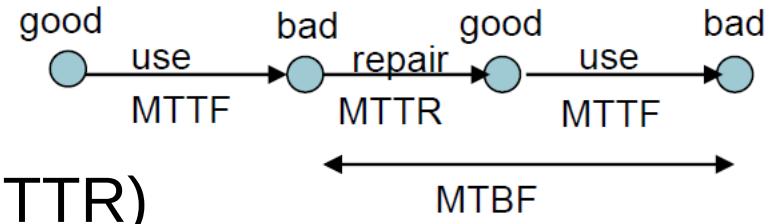
TLB	Page table	Cache	Possible? Condition?
Hit	Hit	Miss	Possible, but page table never checked if TLB hits
Miss	Hit	Hit	Possible, TLB miss but entry found in page table; after retry, data in cache
Miss	Hit	Miss	Possible, TLB miss but entry found in page table; after retry, data miss in cache
Miss	Miss	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

Outline

- Memory hierarchy summary
- **Hamming code**
- 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

- Richard Hamming(1915-1998)
  - Turing award(1968)
  - Automatic coding systems, and error-detecting and error-correcting codes
- Hamming distance
  - minimum number of bits that are different between any two correct bit patterns
- 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 1st bit to the right in the address is 1
    - p4 checks bits where 2nd bit to the right in the address is 1
    - p8 checks bits where 4th bit to the right in the address is 1

	1 **p1**	2 **p2**	3 **d1**
4 **p4**	5 **d2**	6 **d3**	7 **d4**
8 **p8**	9 **d5**	10 **d6**	11 **d7**
12 **d8**	13 **d9**	14 **d10**	15 **d11**

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	
Parity bit coverate p2		X	X			X	X			X	X	
Parity bit coverate p4				X	X	X	X					X
Parity bit coverate 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 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 0	p2 0	d1	p4 1	d2	d3	d4	p8 1	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

√ 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

0 **Pn**	1 **p1**	2 **p2**	3 **d1**
4 **p4**	5 **d2**	6 **d3**	7 **d4**
8 **p8**	9 **d5**	10 **d6**	11 **d7**
12 **d8**	13 **d9**	14 **d10**	15 **d11**

Outline

- Memory hierarchy summary
- Hamming code
- **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

Pitfalls

- Byte vs. word addressing
  - Example: 32-byte direct-mapped cache, 4-byte blocks
    - Byte 36 maps to block 1
    - Word 36 maps to block 4
  - Example: 256-byte cache and 32 bytes per block
    - Byte address 300 maps to block number 1

    0…0 0 0 1 0 0 1 0 1 1 0 0
- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality

```
for (i = 0; …) {
    for (j = 0; …) {
        x[i][j] = x[i][j]+y[i][j] } }
```

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