Exploiting Memory Hierarchy: Memory Bandwidth, Flash, Disk Storage, Cache

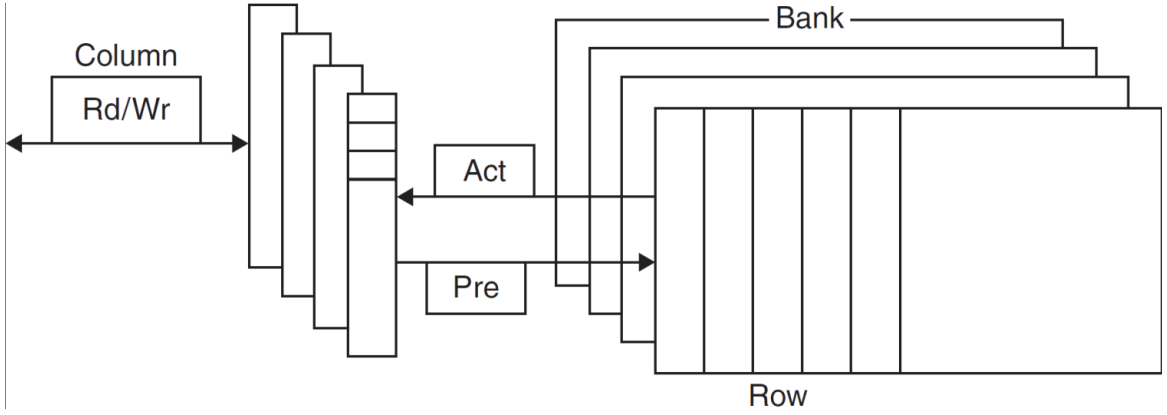
Dr. Vincent C. Emeakaroha

08-03-2017

vc.emeakaroha@cs.ucc.ie

Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
  - Enable double speed
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs



DRAM Generations

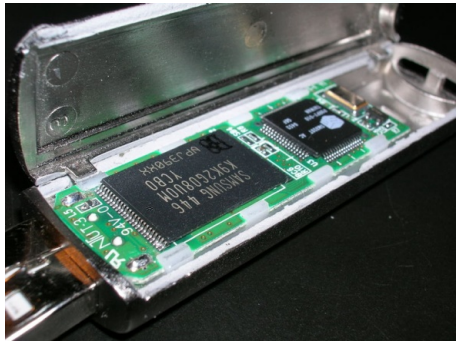
Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50

DRAM Performance Factors

- Row buffer
  - Allows several words to be read and refreshed in parallel
- Synchronous DRAM
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth
- DRAM banking
  - Allows simultaneous access to multiple DRAMs, each with its own row buffer
  - Improves bandwidth

Flash Storage

- A type of
  - Electrically Erasable Programmable Read-Only Memory (EEPROM)
  - Writes can wear out flash memory bits
- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

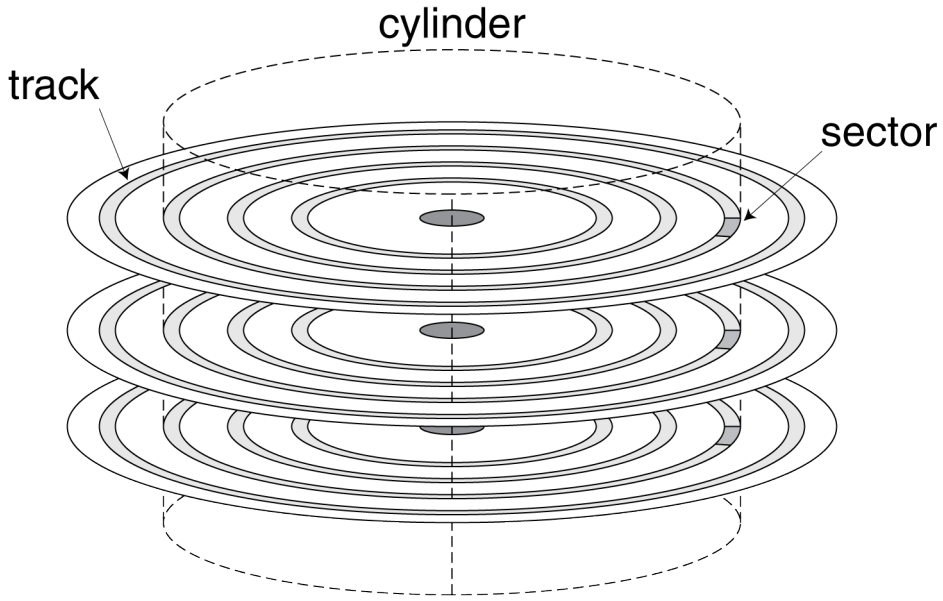


Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear-leveling technique: remap data to less used blocks

Disk Storage

- Nonvolatile, rotating magnetic storage
- Consist of a collection of platter
  - Rotates at 5400 – 15000 revolution per minute



Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps
- Access to a sector involves
  - Three steps
    - Seek: move the heads to the desired track
    - Rotational latency: wait for the sector to rotate under read/write head
    - Data transfer
  - Controller overhead

Disk Performance Comparison

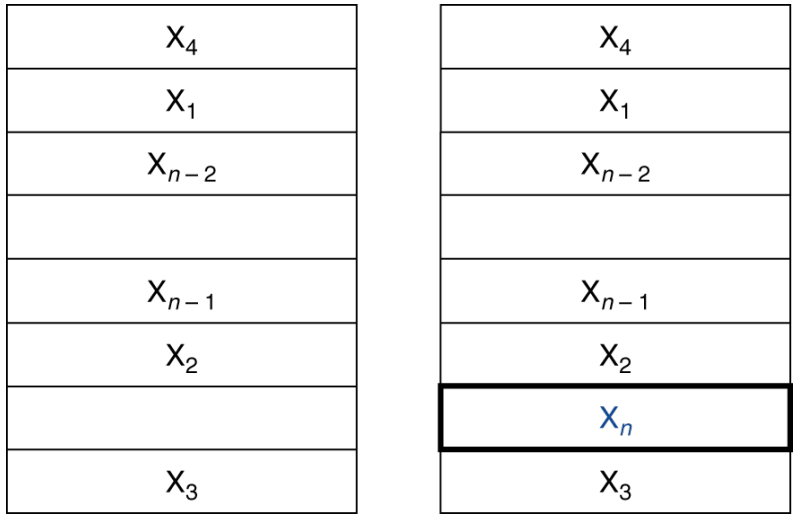
- Slower access time
  - Magnetic device
- But cheap per bit
  - High storage capacity
- Flash is 1000x faster than disk
  - But magnetic disk do not have wear out problem
- DRAM is 100,000x faster than disk

Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given the data X_1, …, X_{n-1} in a cache, processor request X_n
  - Causes a cache miss
  - Block loaded from main memory



a. Before the reference to X_n b. After the reference to X_n

Two key questions:

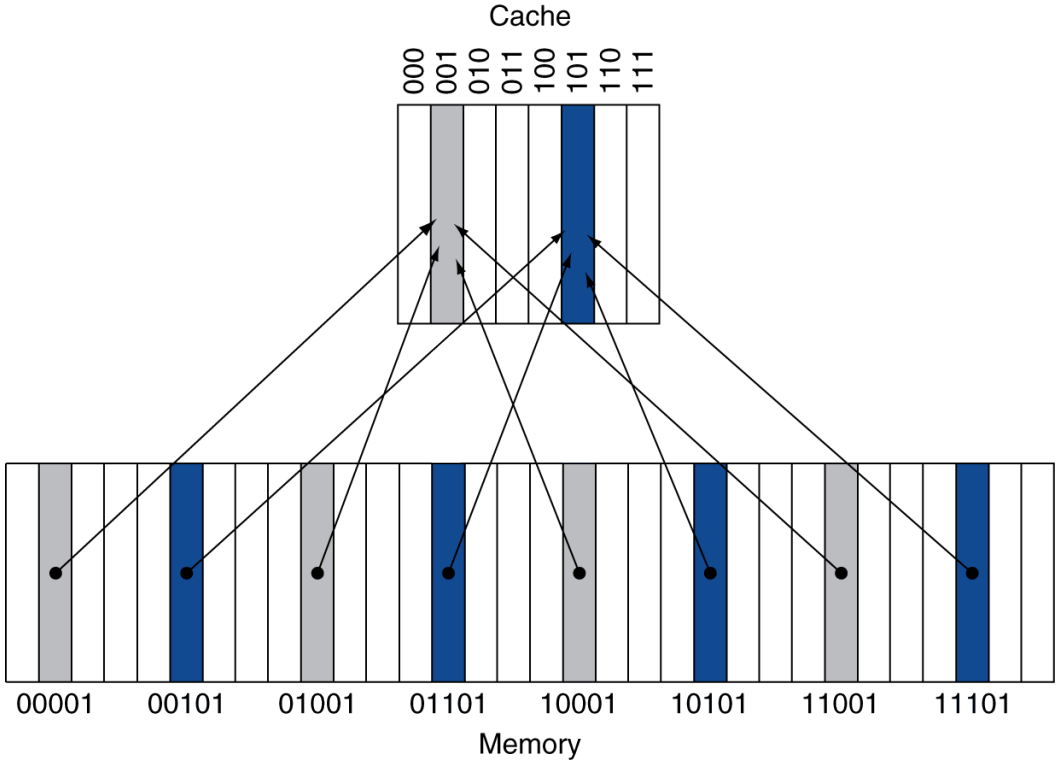
1. How do we know if a data item is in the cache?
2. How do we find it?

A simple solution is to assign cache location based on the address of the word in memory

- Direct mapping

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice for cache systems to calculate address
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits
- E.g., 8-block cache uses three lower bits $8 = 2^3$

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

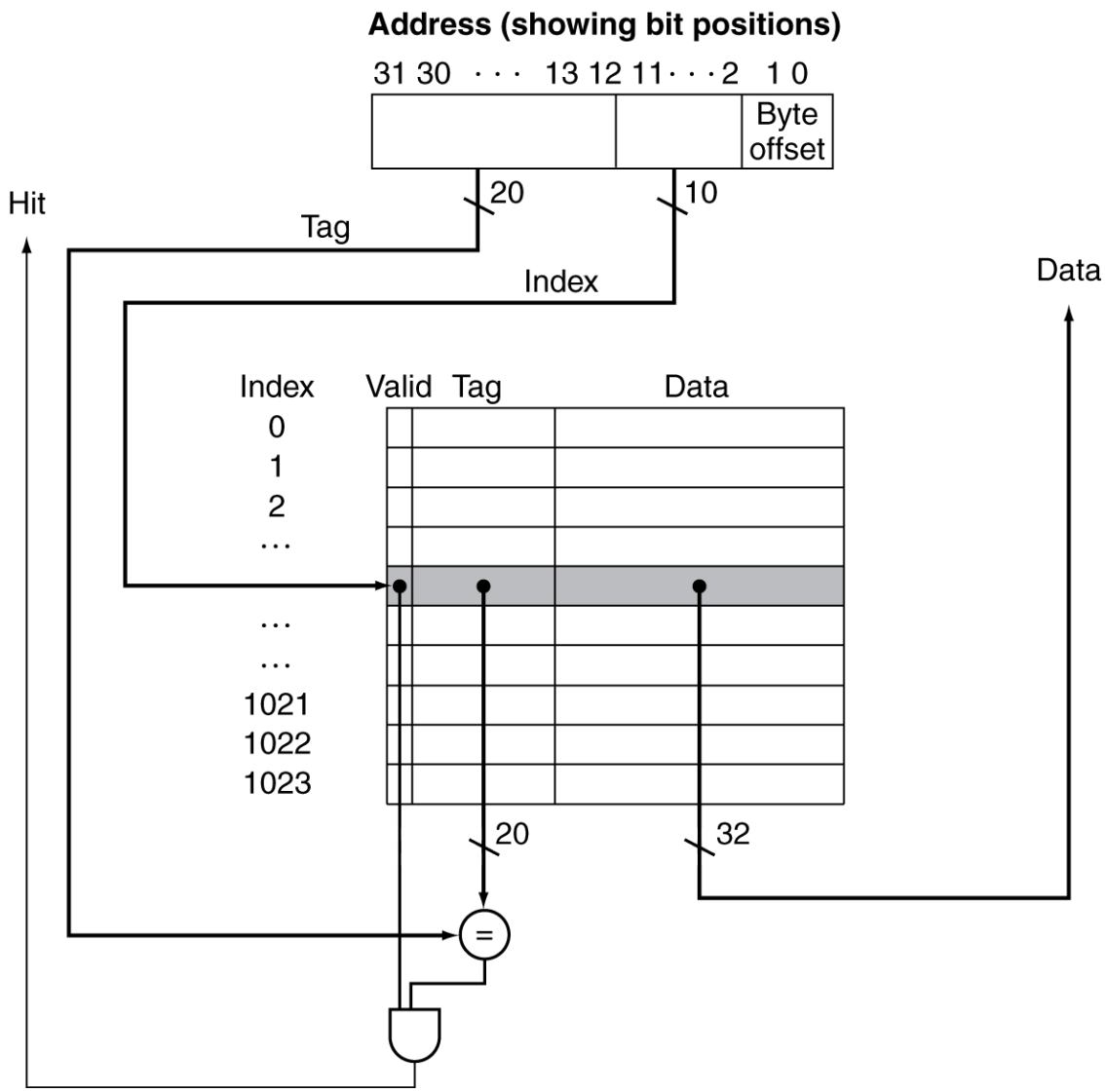
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision



If tag and upper 20 are equal and valid bit set to 1 then we have a hit

Example: Larger Block Size

- 64 blocks cache, 16 bytes/block
  - To what block number does address 1200 map?
- Block address = Byte address/ Byte per block = $\lfloor 1200/16 \rfloor = 75$
- Block number = 75 modulo 64 = 11

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help to reduce the penalty

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access
  - Both instruction and data miss follow the same procedure of fetching block from memory hierarchy

Cache Miss Procedure

- Send original PC value to memory
  - i.e., current PC -4
- Instruct memory to perform a read and wait for memory to complete access
- Write the cache entry
  - Put data from memory into data portion of entry
  - Write upper bits of the address into the tag
  - Turn the valid bit on
- Restart instruction execution
  - Refetch instruction from cache, this time a hit

Cache Write

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write-through technique
  - Update both cache and next lower level memory hierarchy ensuring consistency
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative solution to write buffer
  - On data-write hit, just update the block in cache
    - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first