CPE 431/531

Chapter 5 – Large and Fast: Exploiting Memory Hierarchy

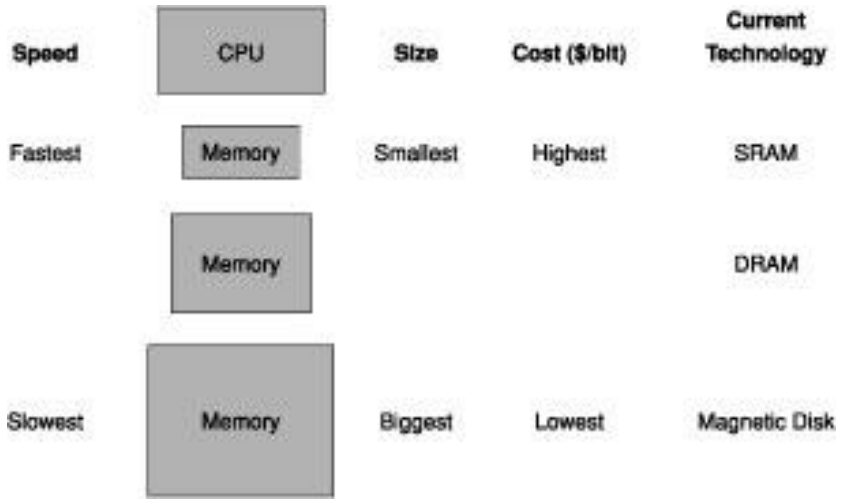
Dr. Rhonda Kay Gaede

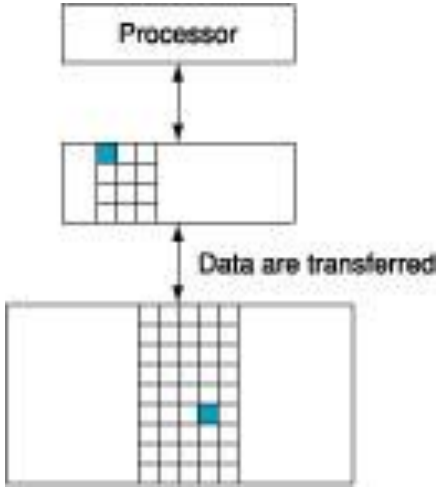
5.1 Introduction

- Programmers always want _____ amounts of _____memory. Caches give that _____
- Principle of Locality
  - Temporal Locality - _____ _____
  - Spatial Locality - _____ _____
- Build a memory _____.

5.1 Cache Terminology

- Data is copied between only ___ levels at a time.
- The minimum data unit is a _____.
- If the data appears in the upper level, this situation is called a ___. The data not appearing in the upper level is called a _____.





5.1 More Terminology

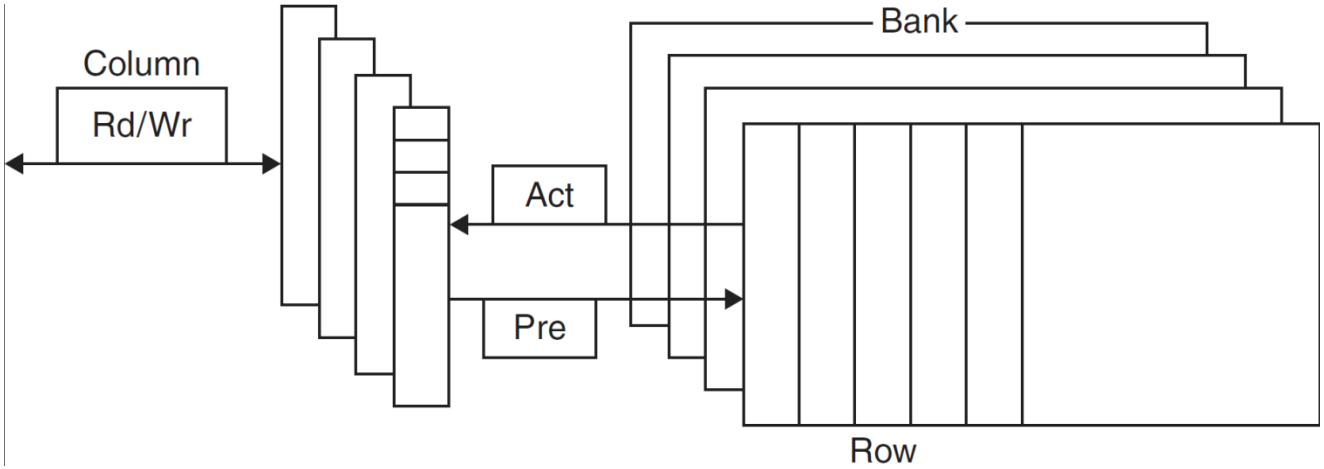
- The _____ is the fraction of memory accesses found in the upper level.
- The _____ is the fraction of memory accesses not found in the upper level.
- The _____ is the time to access the upper level of the memory hierarchy.
- The _____ is the time to replace a block in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the processor.

5.2 Memory Technologies

- _____ RAM (____)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- _____RAM (_____)
  - 50ns – 70ns, $20 – $75 per GB
- _____ disk
  - 5ms – 20ms, $0.20 – $2 per GB
- Ideal memory
  - Access time of _____
  - Capacity and cost/GB of ____

5.2 DRAM Technology

- Data stored as a _____ in a _____
  - Single _____ used to access the _____
  - Must periodically be _____
    - _____ contents and _____ back
    - Performed on a DRAM "____"

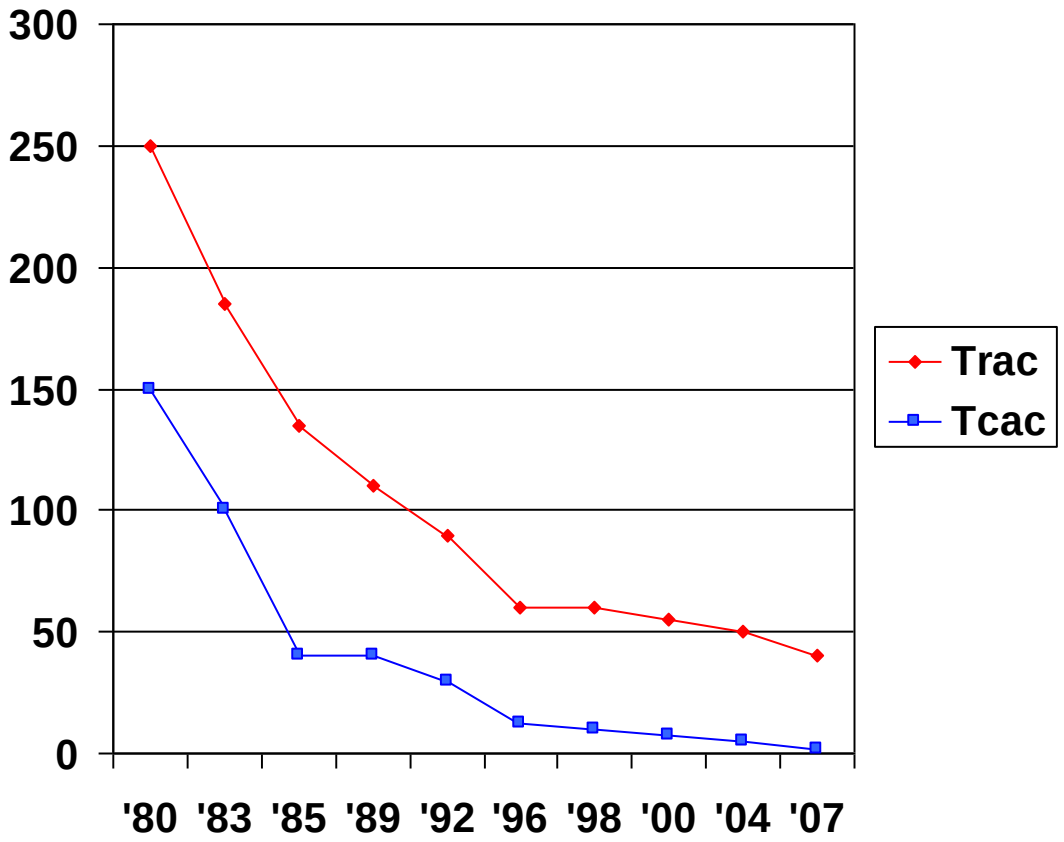


5.2 Advanced DRAM Organization

- Bits in a DRAM are organized as a _____ _____
  - DRAM accesses an _____ _____
  - _____ mode: supply _____ words from a ____ with _____ _____
- _____ data rate (DDR) DRAM
  - Transfer on _____ and _____ clock edges
- _____ data rate (QDR) DRAM
  - Separate DDR _____ and _____

5.2 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

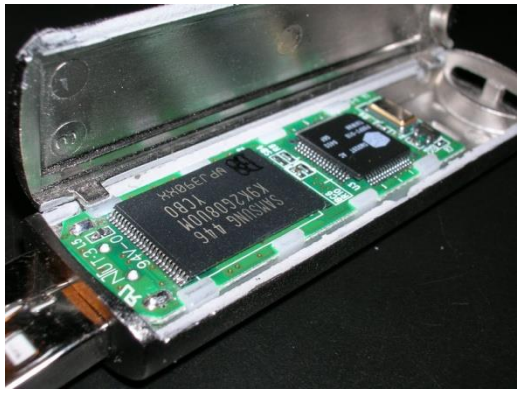


5.2 DRAM Performance Factors

- _____ _____
  - Allows _____ words to be read and refreshed in _____
- _____ _____
  - Allows for _____ accesses in bursts without needing to send _____ _____
  - Improves _____
- _____ _____
  - Allows _____ access to _____ DRAMs
  - Improves _____

5.2 Flash Storage

- _____ semiconductor storage
  - ___× – _____× faster than _____
  - _____, _____ power, more _____
  - But more $/GB (between _____ and _____)

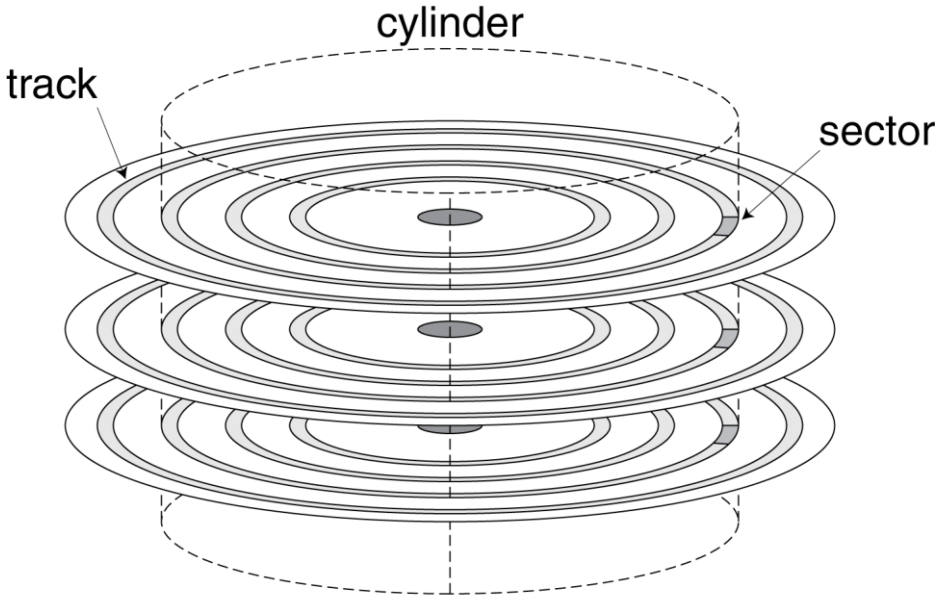


5.2 Flash Types

- ____ flash: bit cell like a ____ gate
  - _____ read/write access
  - Used for _____ memory in _____ systems
- _____ flash: bit cell like a _____ gate
  - _____ (bits/area), but _____ access
  - _____ per GB
  - Used for ____ _____, _____ _____, …
- Flash bits wears out after _____ of accesses
  - Not suitable for direct ____ or _____ replacement
  - _____ _____: _____ data to less used blocks

5.2 Disk Storage

- _____, _____, _____ storage



5.2 Disk Sectors and Access

- Each _____ records
  - Sector ___
  - Data (____ bytes, _____ bytes proposed)
  - _____ correcting code (ECC)
    - Used to hide _____ and recording _____
  - _____ fields and gaps
- Access to a _____ involves
  - _____ delay if other accesses are pending
  - _____: move the heads
  - _____ latency
  - Data _____
  - _____ overhead

5.2 Disk Access Example

- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average read time

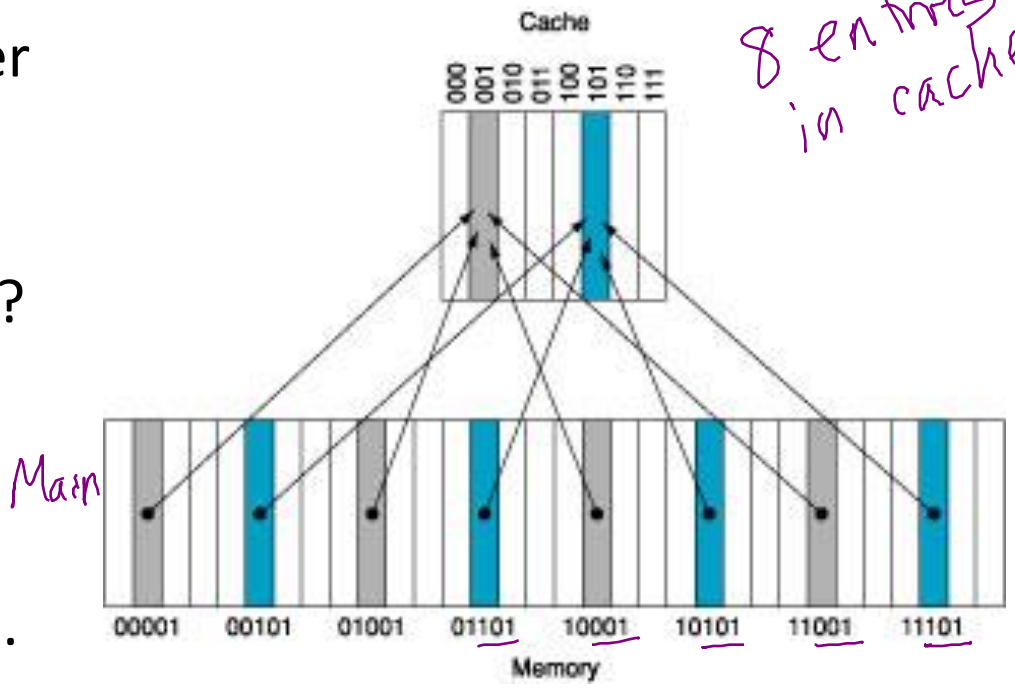
- If actual average seek time is 1ms

5.2 Disk Performance Issues

- Manufacturers quote _____ seek time
  - Based on ____ _____ seeks
  - _____ and ___ _____lead to smaller _____ average seek times
- Smart disk _____ allocate _____ sectors on disk
  - Present _____ sector interface to host
  - SCSI, ATA, SATA
- Disk drives include _____
  - _____ sectors in anticipation of access
  - Avoid _____ and _____ _____

5.3 Burning Question

- How do we know whether a data item is in the cache?
- If it is, how do we find it?
- The simplest scheme is that each item can be placed in exactly one place (direct mapping).
- Mapping



(Block address) Modulo (Number of blocks in cache)

5.3 Accessing a Cache

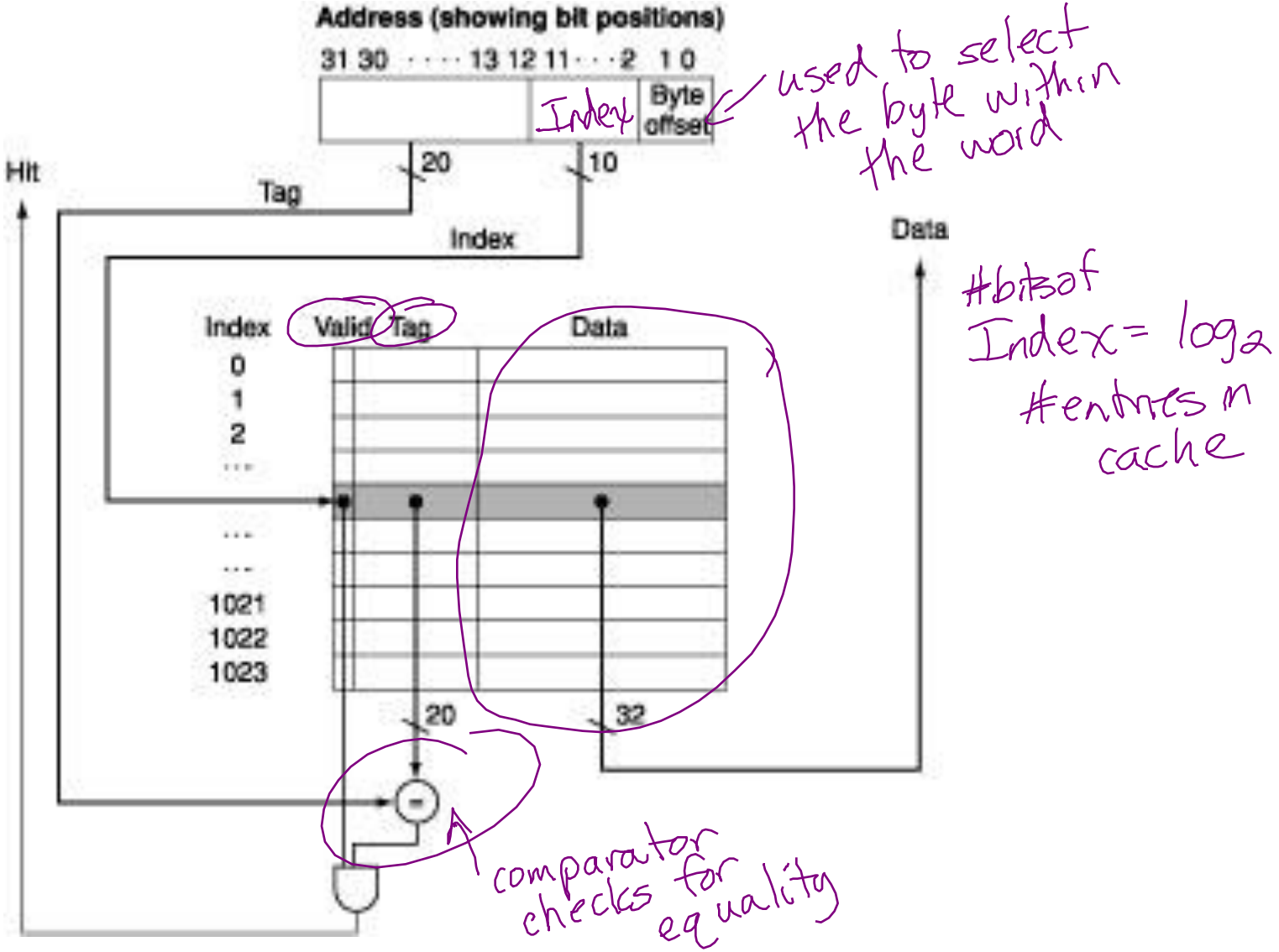
Valid

Index	V	Tag	Data
000	1	10	MEM[16]
001			
010	1	~~11~~, 10	~~MEM[26]~~, MEM[8]
011	1	00	MEM[3]
100			
101			
110	1	10	MEM[22]
111			

10110
22 mod 8 = 6 miss
11010
26 mod 8 = 2 miss
22 mod 8 = 6 hit
26 mod 8 = 2 hit
16 mod 8 = 0 miss
3 mod 8 = 3 miss
16 mod 8 = 0 hit
18 mod 8 = 2 miss
16 mod 8 = 0 hit

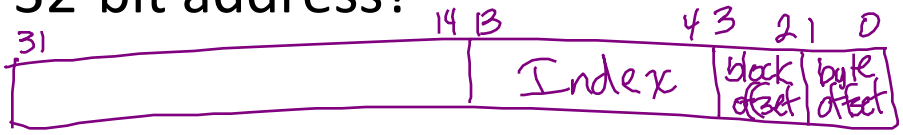
hit rate = 4/9 =
miss rate = 5/9 =

5.3 Mapping Implemented in Hardware



5.3 Total Storage Required

Example: How many total bits are required for a direct-mapped cache with 16 KB of data and four-word blocks, assuming a 32-bit address?



block offset selects the word within the block
how many bits?
2
bits block offset = $\log_2$ # words/block

$16\,kB \times \frac{1\ block}{4\ words} \times \frac{1\ word}{4\ bytes} \times \frac{1\ set}{1\ block} = 1K\ sets$

Index = 10 bits
$= \log_2$ (#sets)

tag bits = 18

$2^{10}\ sets \times \frac{1\ block}{1\ set} \left(\frac{16\ bytes\ data}{1\ block} \times \frac{8\ bits}{1\ byte} \times \left(\frac{18\ bits\ tag + 1\ bit\ valid}{block} \right) \right) =$

$2^{10}(128+19) = 2^{10} \cdot 147 = 147k\ bits$

5.3 Mapping an Address to a Multiword Cache Block

Consider a cache with 64 blocks and a block size of 16 bytes.
What block number does byte address 1200 map to?

block + byte offset combined
4
2 bits byte offset
2 bits block offset

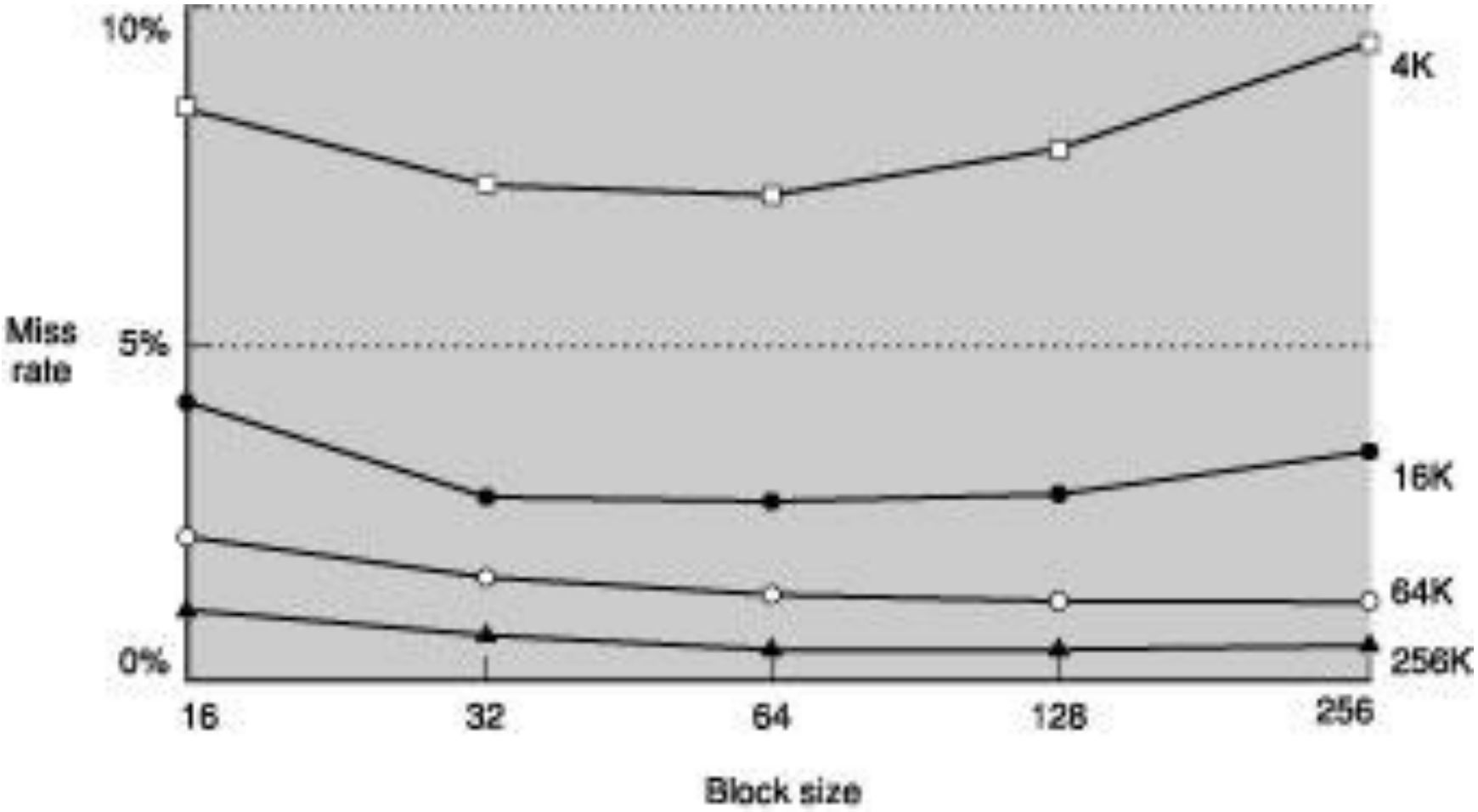
#bits index = $\log_2 64 = 6$

block number = 11

Block number $\left\lfloor \frac{1200}{16} \right\rfloor = 75$

Index 75 modulo 64 = 11

5.3 Miss Rate versus Block Size



5.3 Handling Cache Misses

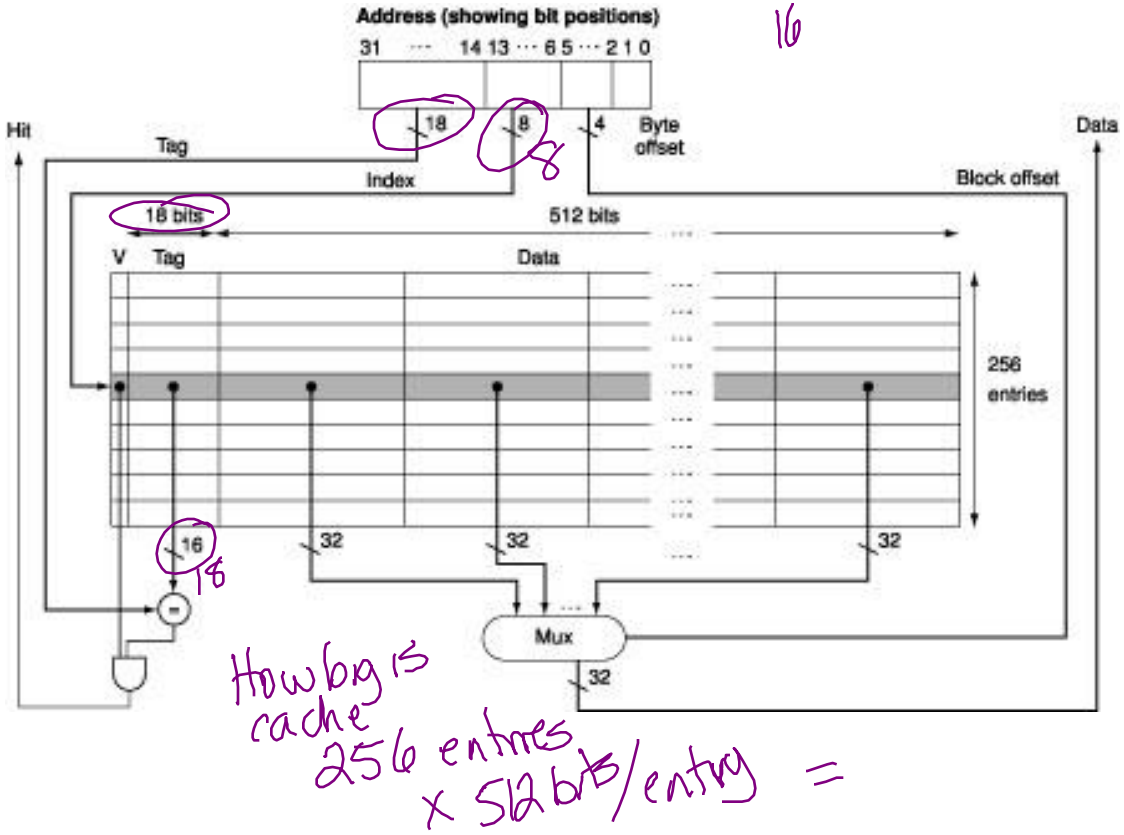
Instruction Cache Miss

1. Send the original PC value (current PC – 4) to the memory.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address into the tag field and turn the valid bit on.
4. Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

5.3 Handling Writes

- Suppose on a store instruction, we wrote the data into only the data cache (and not to main memory).
- Then the cache and main memory are said to be inconsistent
- Solution A: Write everything to both (write through)
  - Problem: bad performance
  - Remediation: write buffer
- Solution B: Only write when evicted from cache (write back)

5.3 An Example Cache



16

How big is cache
256 entries x 512 bits/entry =

The Intrinsity FastMATH Processor is a fast embedded processor that uses the MIPS architecture and a simple cache implementation.

This processor has a 12 stage pipeline.

When operating at peak speed, the processor can request both an instruction word and a data word on every clock cycle.

Separate instruction and data caches are used, each with 4K words and 16-word blocks.

For writes, the FastMATH offers both write-through and write back, letting the OS decide.

5.4 Measuring and Improving Cache Performance

- CPU time = (Read memory misses + Write memory misses) x miss penalty
- Memory-stall clock cycles = Read memory stalls + write memory stalls
- Read-stall cycles = $\frac{\text{Reads}}{\text{Program}}$ x Read miss rate x Read miss penalty
- Write-stall cycles = ($\frac{\text{Writes}}{\text{Program}}$ x Writes miss rate x Write miss penalty) + Write buffer stalls (negligible)
- Memory-stall clock cycles = $\frac{\text{Memory accesses}}{\text{Program}}$ x Miss rate x Miss Penalty
- Calculating Cache Performance
  - i_{miss} = 2 %, d_{miss} = 4 %, $CPI_{perfect}$ = 2, miss penalty = 100 cycles, 36 % loads and stores

What's the speedup/slowdown from perfect?

$$\frac{P_{perfect}}{P_{stalls}} = \frac{ET_{stalls}}{ET_{perfect}} = \frac{I * CPI_{stalls} * CT}{I * CPI_{perfect} * CT} = \frac{5.44}{2} = 2.72$$

instruction miss = $I * 0.02 * 100 = 2I$

data miss = $I * 0.36 * 0.04 * 100 = 1.44I$

$3.44I$

5.4 Impact of Increased Clock Rate

- Suppose the processor in the previous example doubles its clock rate, making the miss penalty 200 cycles.
- Total miss cycles per instruction = $0.36 * 0.04 * 200 + 0.02 * 200$
- Total CPI = $4 + 6.88 = 10.88$ $\quad 2.88 + 4$
- Performance with fast clock compared to performance with slow clock

$$\frac{P_{fast}}{P_{slow}} = \frac{5.44}{10.88}$$

5.4 Average Memory Access Time

- To capture the fact that the time to access data for both hits and misses affects performance, designers sometimes use average memory access time (AMAT) as a way to examine alternative cache designs.
- AMAT = Time for a hit + Miss rate x Miss penalty
- Find the AMAT for a processor with CT = 1 ns, Miss penalty = 20 cycles, Miss rate = 0.05/instruction, Cache access time = 1 cycle. Assume that read and write miss penalties are the same and ignore other write stalls.

AMAT = Time for a hit + Miss rate x miss penalty

= 1 + 0.05 * 20 = 1 + 1 = 2

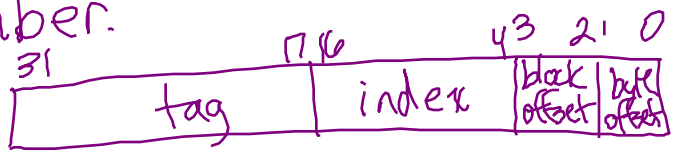
5.4 Flexible Placement Reduces Cache Misses

- One Extreme - direct mapped - one place
- Middle Range - set associative - set of places
- Other Extreme - fully associative - any place
- Set associative mapping

(Block number) Modulo (Number of sets in cache)

Number of sets is a derived number.

2-way means 2 blocks/1 set
4-way means 4 block/1 set



How many sets?

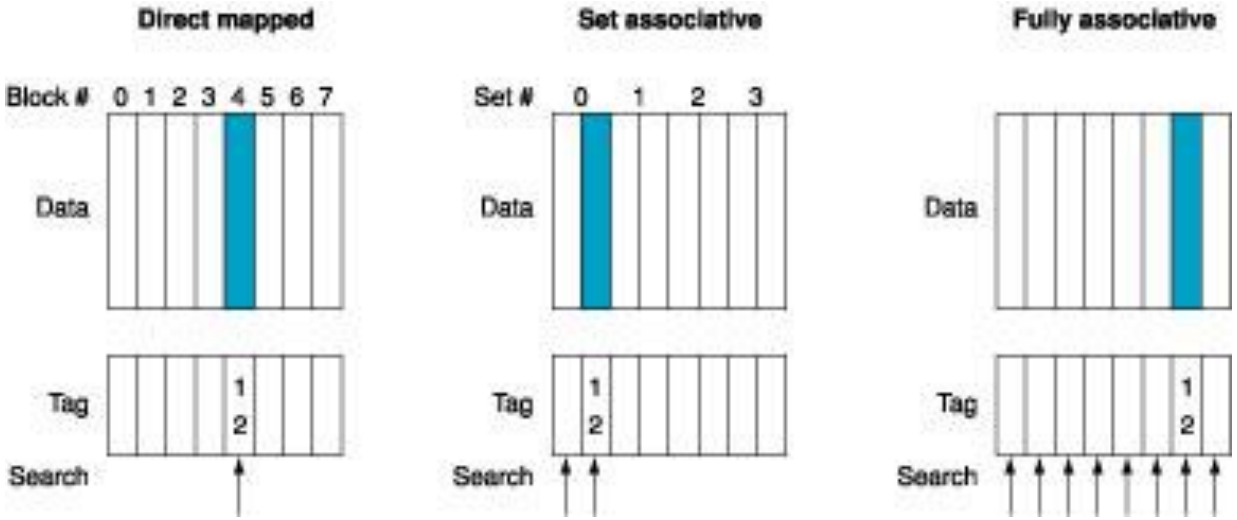
256 KB cache
4 word blocks
2-way set associativity
You have a byte offset

$$256\ KB \times \frac{1\ word}{4\ bytes} \times \frac{1\ block}{4\ words} \times \frac{1\ set}{2\ blocks} = 8k\ sets$$

18 bits index

5.4 Conceptual View of Set Associativity

- One Extreme - direct mapped -
- Middle Range - set associative -
- Other Extreme - fully associative -
- Set associative mapping



5.4 Pseudo-Implementation View of Set Associativity

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0	A	A	B	B
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

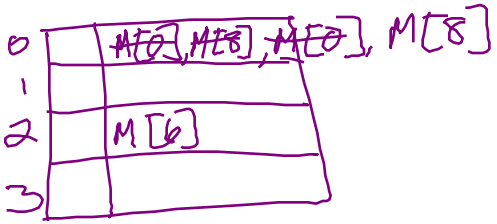
5.4 Misses and Associativity

Look at three small caches (four one word blocks): Address sequence: 0, 8, 0, 6, 8

2-sets, index 1 bit 4 sets, index 2 bits

a. fully associative (1 set) b. two-way set associative c. direct mapped

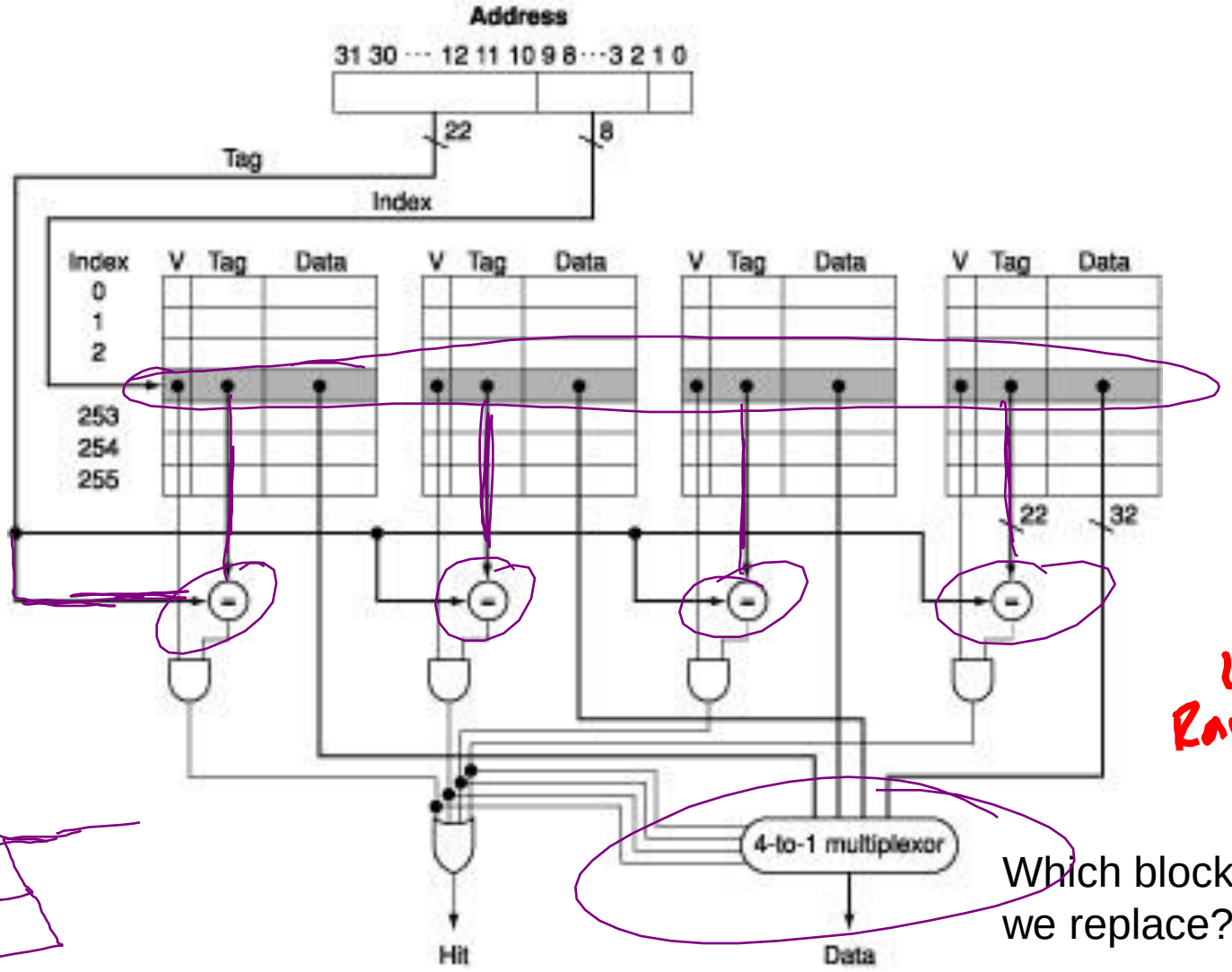
	a. fully associative	b. two-way set associative	c. direct mapped
00000	m	m	m
01000	m	m	m
00000	h	h	m
00110	m	m	m
01000	h	m	m

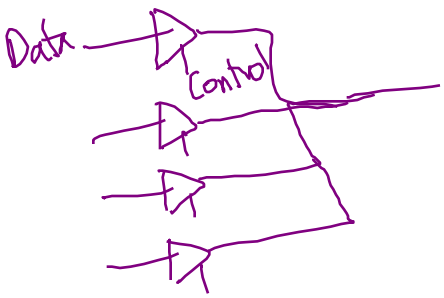


2-way
4 words x 1 block / 1 word x 1 set / 2 blocks = 2 sets

1-way (direct mapped)
4 words x 1 block / 1 word x 1 set / 1 block = 4 sets

5.4 Locating a Block in the Cache





LRU
Random

Which block do we replace?

5.4 Tag Size Considerations

- Size of Tags versus Set Associativity

  For a cache with 4K blocks, a 32-bit address with 0 bits for block and byte offsets, find the #sets, #tag bits for 1, 2, 4 and fully associative organizations

Direct Mapped

$4k \text{ blocks} \times \frac{1 \text{ set}}{1 \text{ block}} = 4k \text{ sets}$ index = 12 bits tag = 20 bits

2-Way

$4k \text{ blocks} \times \frac{1 \text{ set}}{2 \text{ blocks}} = 2k \text{ sets}$ index = 11 bits tag = 21 bits

4-way

$4k \text{ blocks} \times \frac{1 \text{ set}}{4 \text{ blocks}} = 1k \text{ sets}$ index = 10 bits tag = 22 bits

Fully associative

1 set index = 0 bits tag = 32 bits

5.4 Performance of Multilevel Caches

- Example:
  - CPI_{base} = 1.0, CR = 4 GHz
  - Mem_{access} = 100 ns, $L1inst_{miss}$ = 2 %
  - $L2_{access}$ = 5 ns, $L2_{miss}$ per instruction = 0.5 %

Mem access = 400 cycles
global miss rates

Total CPI = CPI_{base} + Memory stalls/instruction

L2 access = 20 cycles

Only L1

CPI_{base} + L1 miss

$= 1.0 + 0.02 \times 400 = 1 + 8 = 9$

L1 and L2

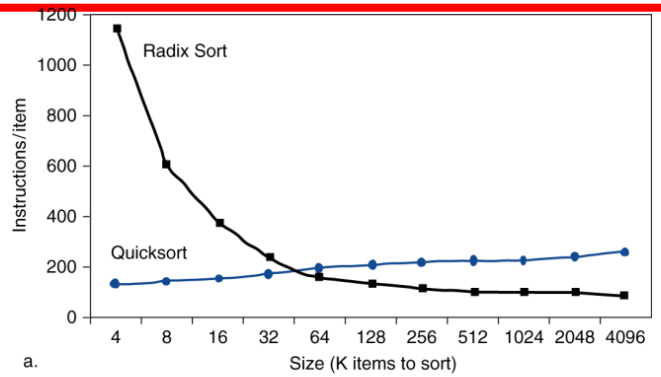
CPI_{base} + L1 miss + L2 miss

$= 1.0 + 0.02 \times 20 + 0.005 \times 400$

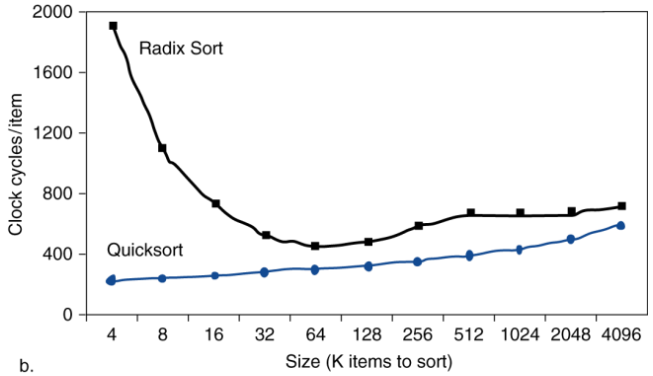
$= 1 + 0.4 + 2 = 3.4$

5.4 Interactions with Software

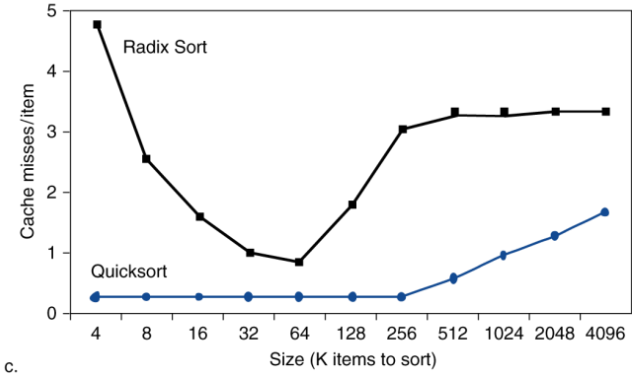
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



a.



b.



c.

5.4 Software Optimization via Blocking

Goal: maximize accesses to data before it is replaced

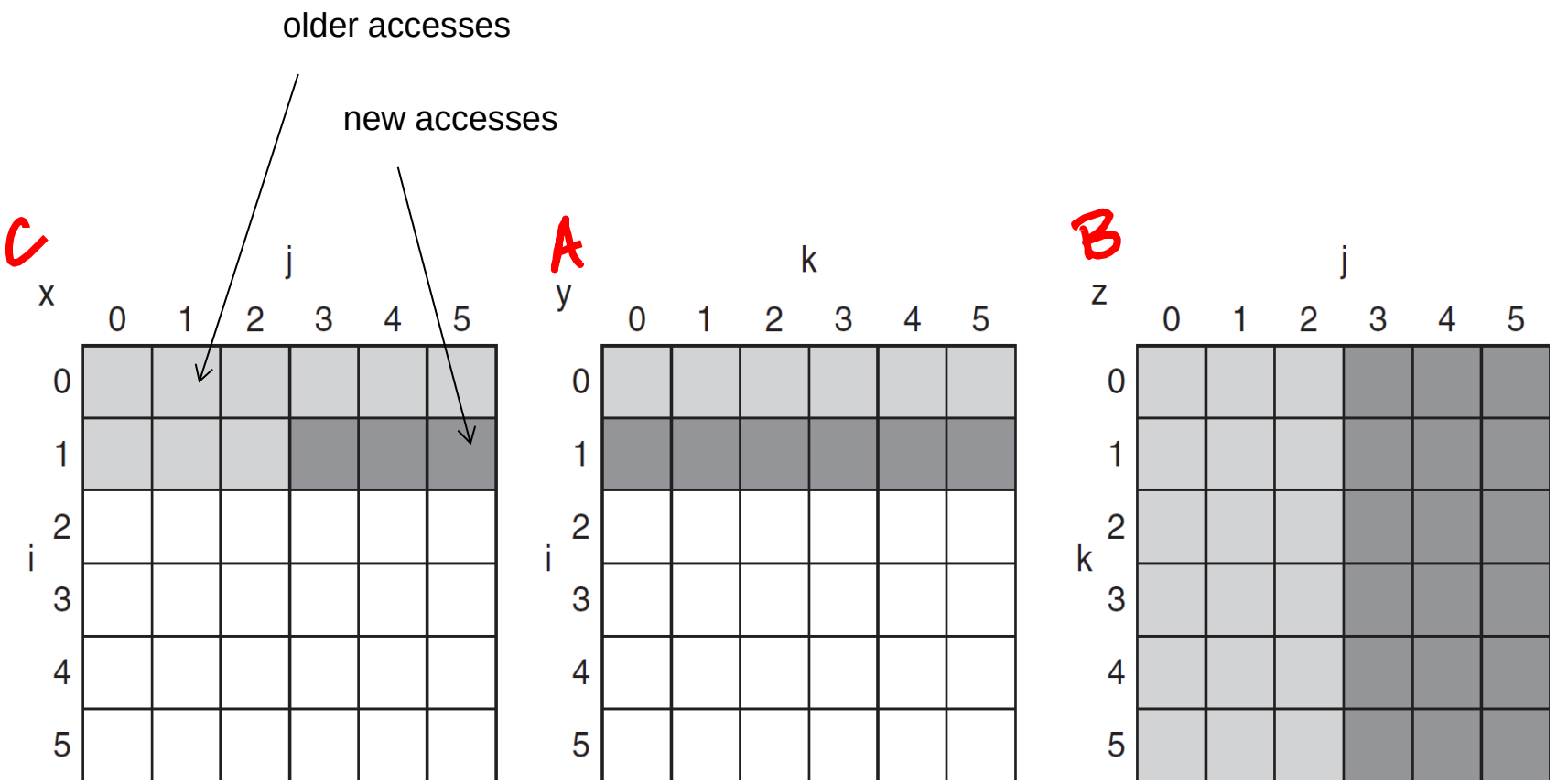
Consider inner loops of DGEMM:

Blocked algorithms operate on submatrices or blocks rather than entire rows or columns of an array

```
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][k]*B[k][j] */
    C[i+j*n] = cij;               /* C[i][j] = cij */
}
```

Outer loop
i

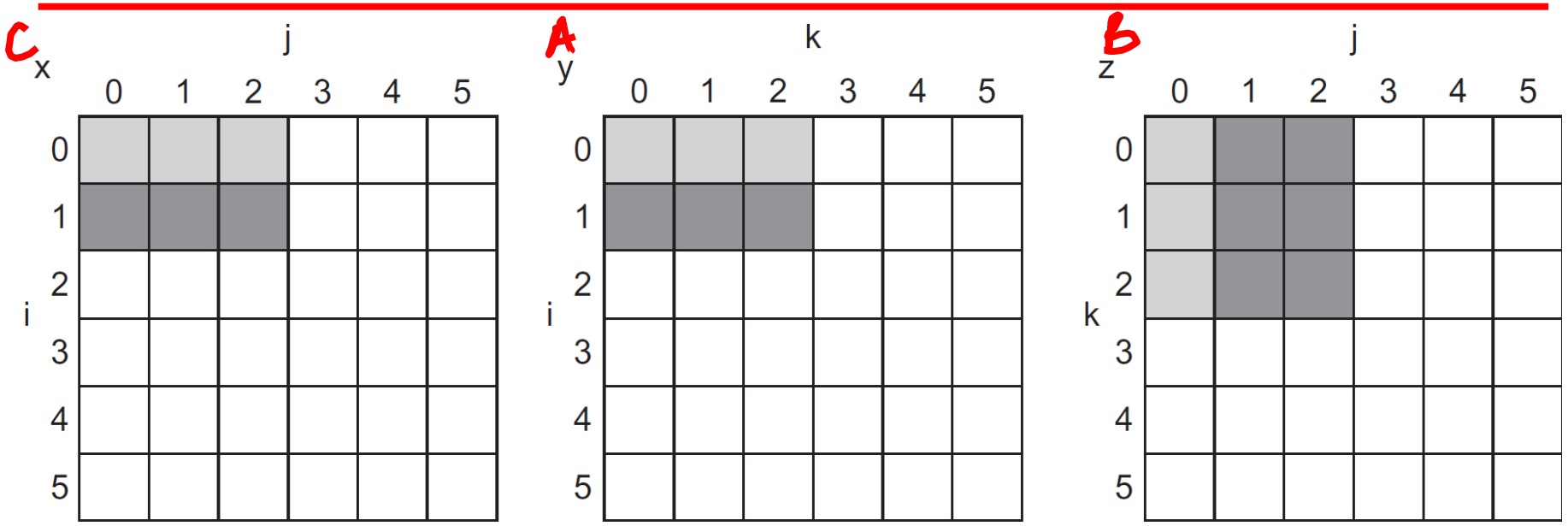
5.4 Array Access Patterns

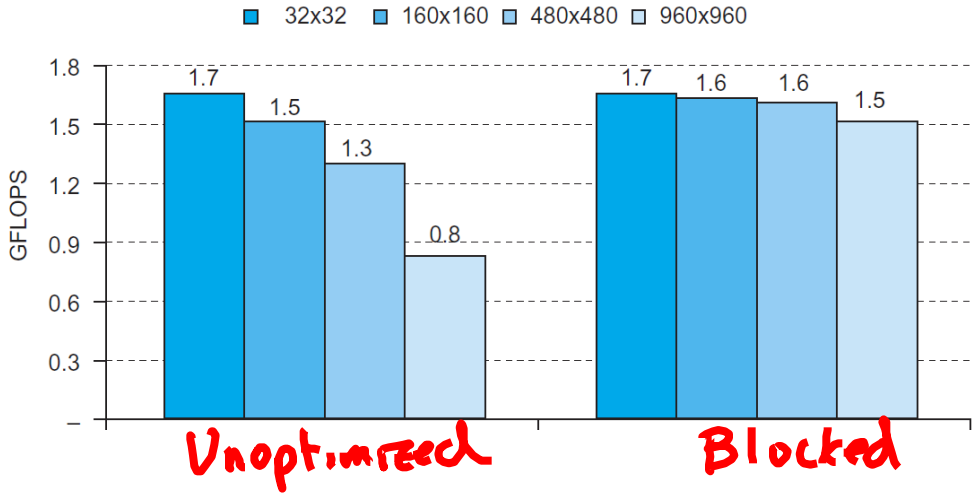


5.4 Cache Blocked DGEMM

```
1 #define BLOCKSIZE 32
2 void do_block (int n, int si, int sj, int sk, double *A,
double
3 *B, double *C)
4 {
5  for (int i = si; i < si+BLOCKSIZE; ++i)
6   for (int j = sj; j < sj+BLOCKSIZE; ++j)
7   {
8    double cij = C[i+j*n];/* cij = C[i][j] */
9    for( int k = sk; k < sk+BLOCKSIZE; k++ )
10    cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
11   C[i+j*n] = cij;/* C[i][j] = cij */
12  }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16  for ( int sj = 0; sj < n; sj += BLOCKSIZE )
17   for ( int si = 0; si < n; si += BLOCKSIZE )
18    for ( int sk = 0; sk < n; sk += BLOCKSIZE )
19     do_block(n, si, sj, sk, A, B, C);
20 }
```

5.4 Blocked DGEMM Access Pattern





5.5 Dependable Memory Hierarchy

- Two states of service
  1. Service accomplishment - service delivered as specified
  2. Service interruption – service different from specified
- Transitions from 1 to 2 are failures, transitions from 2 to 1 are restorations.
- Failures can be permanent or intermittent.
- Reliability is a measure of the continuous service accomplishment, the metric is mean time to failure.
- Availability is a measure of the service accomplishment with respect to the alternation between the two states of accomplishment and interruption.

MTBF = MTTF + MTTR

$$A = \frac{MTTF}{MTTF + MTTR}$$

5.5 MTTF vs. AFR for Disks

- Some disks today are quoted to have a 1,000,000-hour MTTF. As 1,000,000 hours is 114 years, it would seem like they practically never fail. Warehouse scale computers that run Internet services such as Search might have 50,000 servers. Assume each server has 2 disks. Use AFR to calculate how many disks we would expect to fail per year.

One year = 365 x 24 = 8760 hours

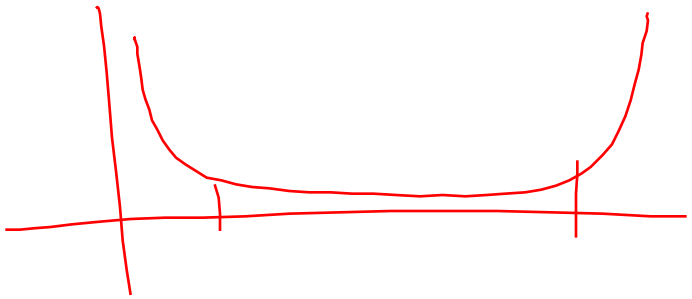
$\frac{8760}{1{,}000{,}000}$ = 0.876% is the annual failure rate

With 100,000 disks, that means 876 fail each year.

5.5 Availability Considerations

- To increase MTTF, you can improve the quality of components or design systems to continue operation in the presence of components that have failed.
- Three techniques
  - Fault avoidance - preventing by construction
  - Fault tolerance - use redundancy
  - Fault forecasting - predicting presence and creation
- We also need to work on decreasing MTTR

fault tolerance

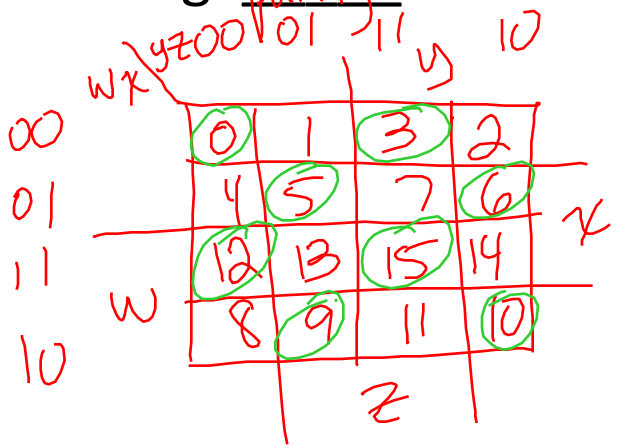


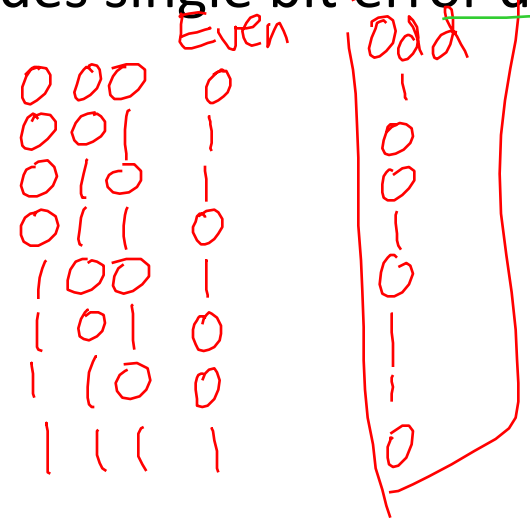
5.5 Single Error Detection - Parity

- Hamming distance
  - Number of bits that are different between two bit patterns

0000
1111
Hamming distance = 3

- Minimum distance = 2 provides single bit ~~error~~ detection
  - e.g. parity code





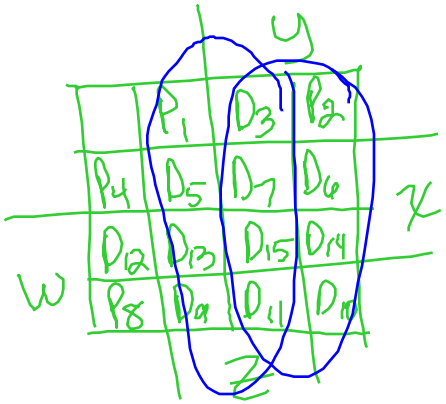
- Minimum distance = 3 provides single error correction, double error detection

5.5 Encoding Single Error Correcting Hamming Code

- To calculate Hamming code:
  - Number bits from 1 on the left
  - All bit positions that are a power of 2 are parity bits
  - Each parity bit checks certain data bits:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
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

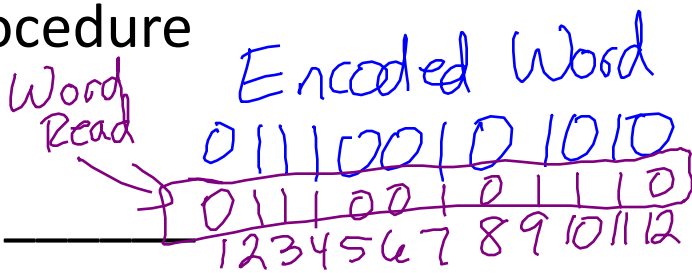
Data 1234 5678
Parity 1111 1111
$P_1 P_2 D_1 P_4 D_2 D_3 D_4 D_5 P_8 D_6 D_7 D_8$



z P_1 x P_4
y P_2 w P_8

5.5 Decoding Single Error Correcting Hamming Code

- Value of parity bits indicates which bits are in error
  - Use numbering from encoding procedure
  - E.g.
    - Parity bits = _____ indicates ___ _____
    - Parity bits = _____ indicates bit ___ was flipped



1 0 0 1 1 0 1 0 Data
1 2 3 4 5 6 7 8 9 10 11 12
P1 P2 1 P4 0 0 1 P8 1 0 1 0

$P_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11}$
$= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0$

$P_2 = D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11}$
$= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1$

$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12}$
$= 0 \oplus 0 \oplus 1 \oplus 0 = 1$

$P_8 = D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12}$
$= 1 \oplus 0 \oplus 1 \oplus 0 = 0$

$C_1 = P_1 \oplus P_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11}$
$= 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0$

$C_2 = P_2 \oplus P_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11}$
$= 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$

$C_4 = P_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_{12}$
$= 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

$C_8 = P_8 \oplus D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12}$
$= 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$

1 0 1 0

5.5 SEC/~~DEC~~ DED Hamming Code

- Add an additional parity bit for the whole word (p_n)
- Make Hamming distance = 4
- Decoding:
  - Let H = SEC parity bits
    - H even, p_n even, no error
    - H odd, p_n odd, a correctable single error
    - H even, p_n odd, error occurred in p_n bit
    - H odd, p_n even, double error
- Note: ECC DRAM uses SEC/DEC with 8 bits protecting each 64 bits

5.6 Virtual Machines Redux

- First developed in the 1960s, they have remained an important part of mainframe computing and have recently gained popularity due to
  - Increasing importance of isolation and security
  - The failures in security and reliability of standard operating systems
  - The sharing of a single computer among many related users
  - The dramatic increase in raw speed of processors
- Broadest Definition
  - Includes basically all emulation methods that provide a standard software interface, like the Java Virtual Machine
- Our Definition
  - Provide a complete system-level environment at the ISA level

5.6 Virtual Machine Basics

- System virtual machines present the illusion that users have an entire computer to themselves, including a copy of the operating system.
- With a VM, multiple OSes all share the hardware resources.
- The software that supports VMs is called a virtual machine monitor (VMM) or hypervisor.
- The underlying hardware is called the host, sharing resources among the guest VMs.

5.6 Virtual Machine Ancillary Benefits

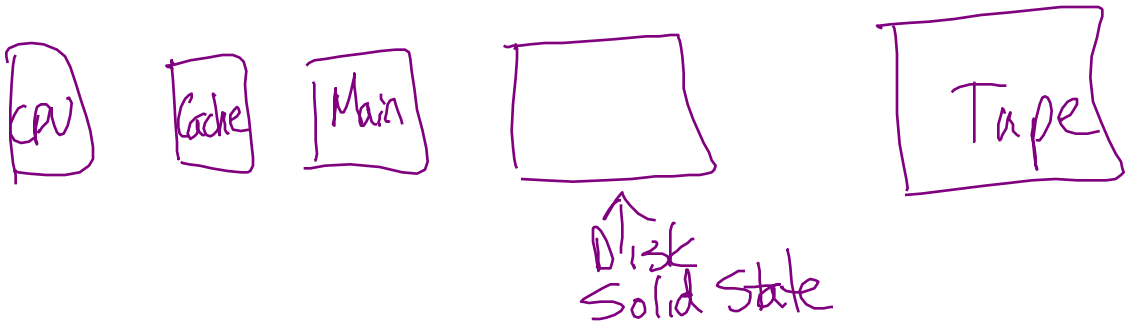
- Our interest is primarily in improving protection
- Other benefits include
  - Managing software: a typical deployment might be some OSes running legacy OSes, many running the current stable OS release, and a few testing the next OS release.
  - Managing hardware: Consolidate the number of servers. Some VMMs support migration of a running VM to a different computer, either to balance load or to evacuate from failing hardware.

5.6 Requirements of a Virtual Machine Monitor

- Guest software should behave on a VM exactly as if it were running on the native hardware, except for performance-related behavior or limitations of fixed resources shared by multiple VMs.
- Guest software should not be able to change allocation of real system resources directly.
- At least two processor modes, system and user.
- A privileged subset of instructions available only in system mode, all system resources must be controllable only via these instructions.

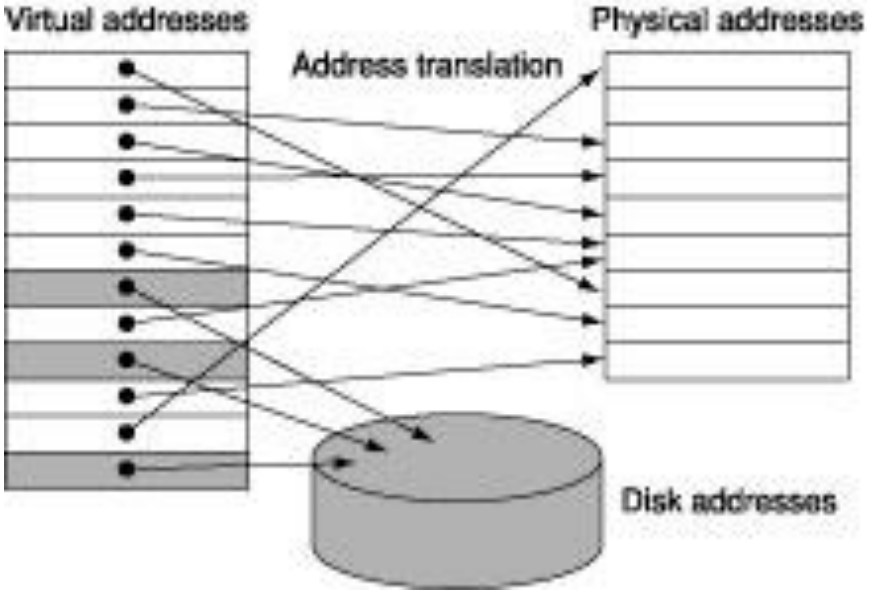
5.7 Virtual Memory

- The main memory can act as a cache for the secondary storage.
- Historically, two motivations for virtual memory
  - remove the burden of small main memory
  - to allow efficient and safe sharing
- Virtual memory implements the translation of a program's address space to physical addresses
- This translation process enforces protection of a program's address space from other programs.



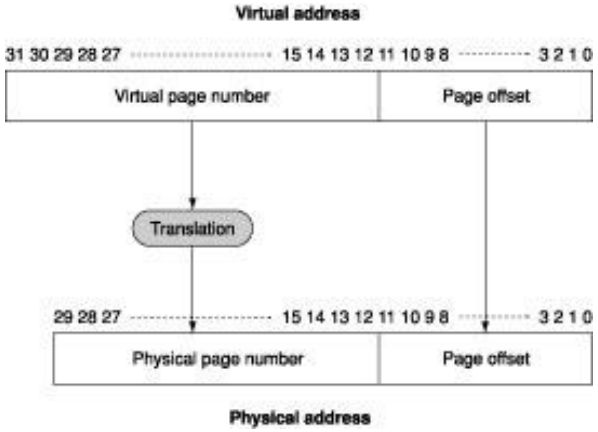
5.7 Virtual Memory Terminology

- A virtual memory ___block___ is called a ___page___.
- A virtual memory ___miss___ is called a ___page fault___.
- Each ___virtual___ address is translated to a ___physical___ address.
- This process is called ___address translation___.



5.7 Virtual Memory Facts

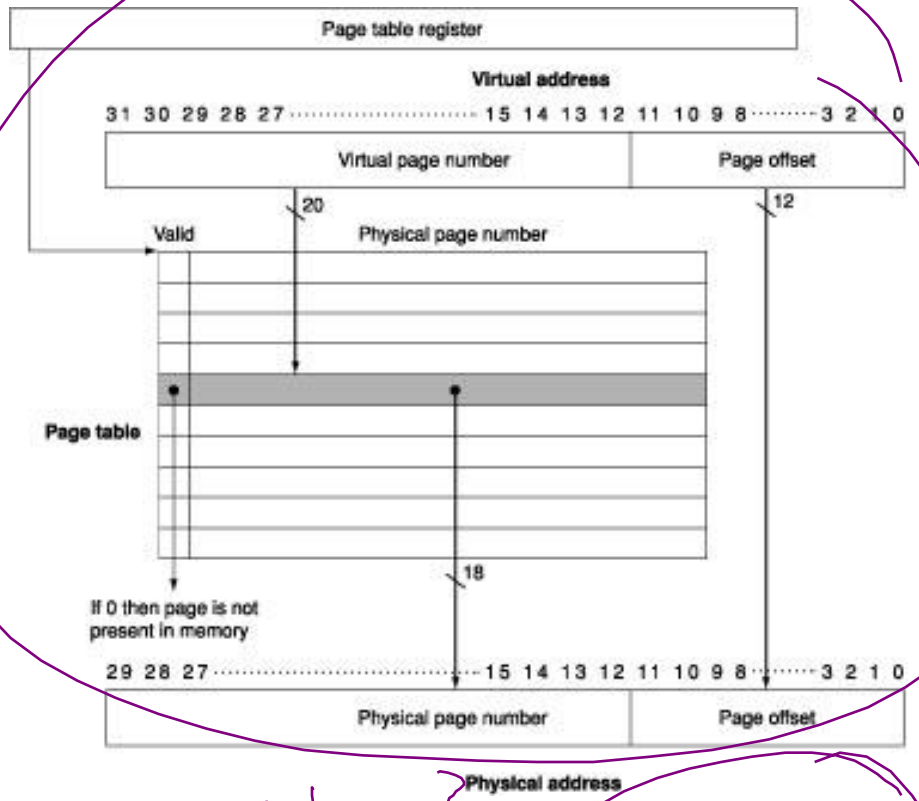
- A virtual address is broken into a virtual page number and a page offset.



- A page fault takes millions of cycles to process
  - Pages should be large enough to amortize the high access time, though embedded systems are going smaller.
  - Fully associative placement of pages is attractive.
  - Page faults can be handled in software.
  - Virtual memory uses LRU.

5.7 Virtual Memory Mapping

- Pages are _located_ by using a _page table_ that _indexes_ memory.
- Each _program_ has its own _page table_.
- The _page table_ register points to the _start_ of the _page table_.
- A _full_ table is too _expensive_, _hierarchical_ page tables are used.
- The _state_ of a _process_ consists of the _page table register_, _program counter_ and _registers_.



Virtual address 64 bits
Page size is 2^{12} bytes
How big is page table?
How big is physical address? 48 bits

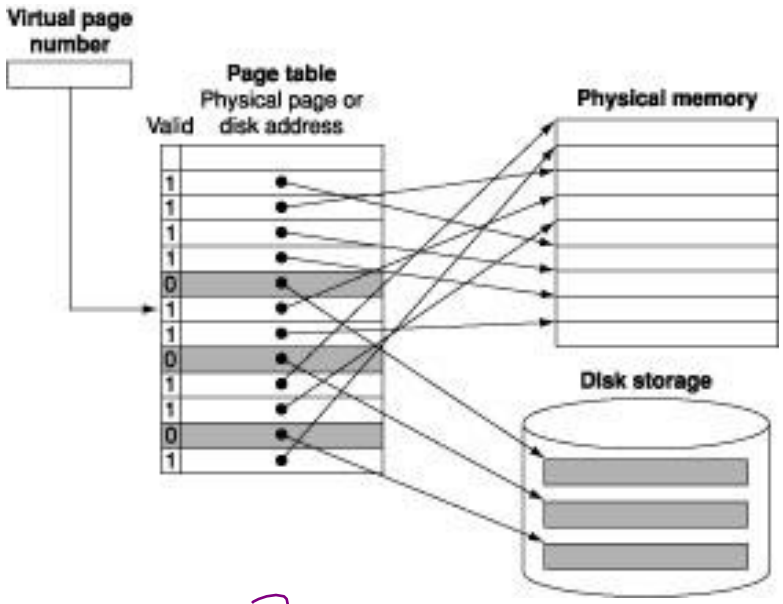
How many virtual pages?
2^{54} x 38 bits
rows columns

Caches

5.7 Page Faults

- The operating system manages page replacement.
- The operating system usually creates the space on disk for all of the pages of a process when it creates the process, this space is called swap space

- A data structure records where each virtual page is stored on disk. Another data structure tracks which processes and which virtual addresses use each physical page.
- On a page fault, the least recently
- used page is evicted.
- Consider 10, 12, 9, 7, 11, 10, then 8



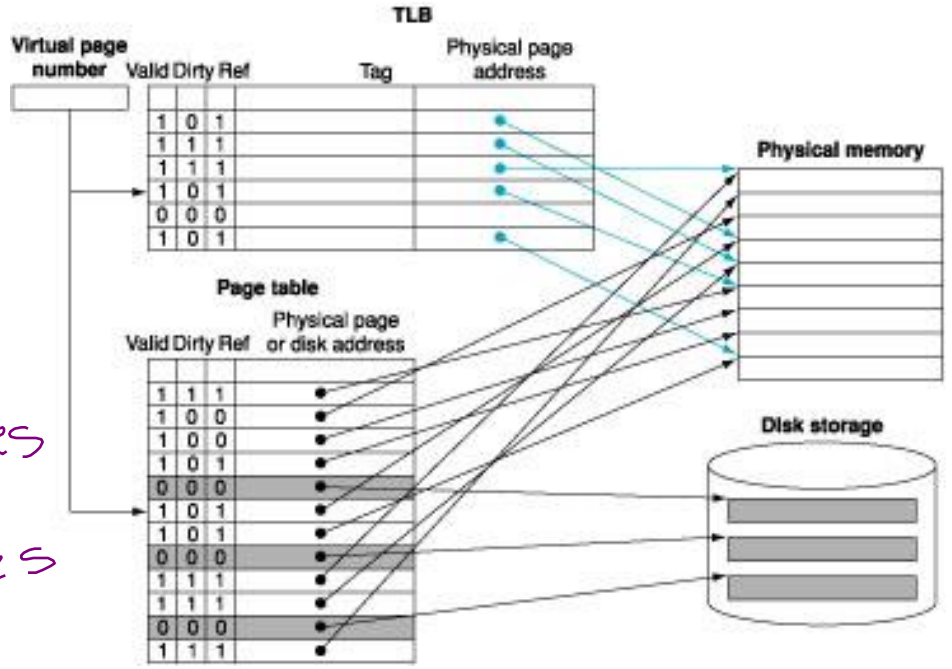
Which evicted ?
12

5.7 Making Address Translation Fast: The TLB

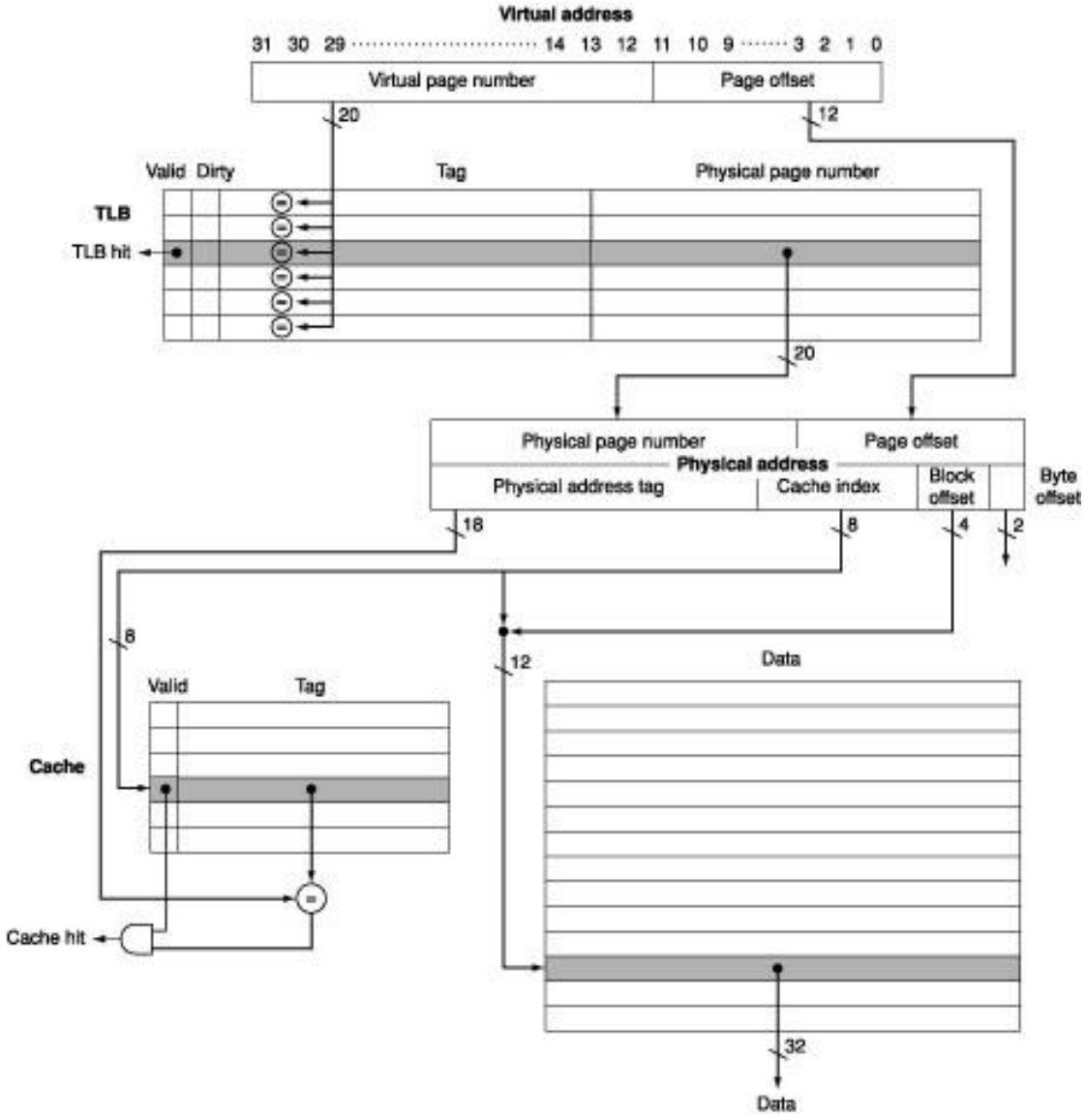
- With virtual memory, you need two memory accesses, one extra for the translation
- Add a cache to keep track of recent translations.
- It's called a translation lookaside buffer (TLB).
- A TLB miss may or may not be a page fault

TLB characteristics

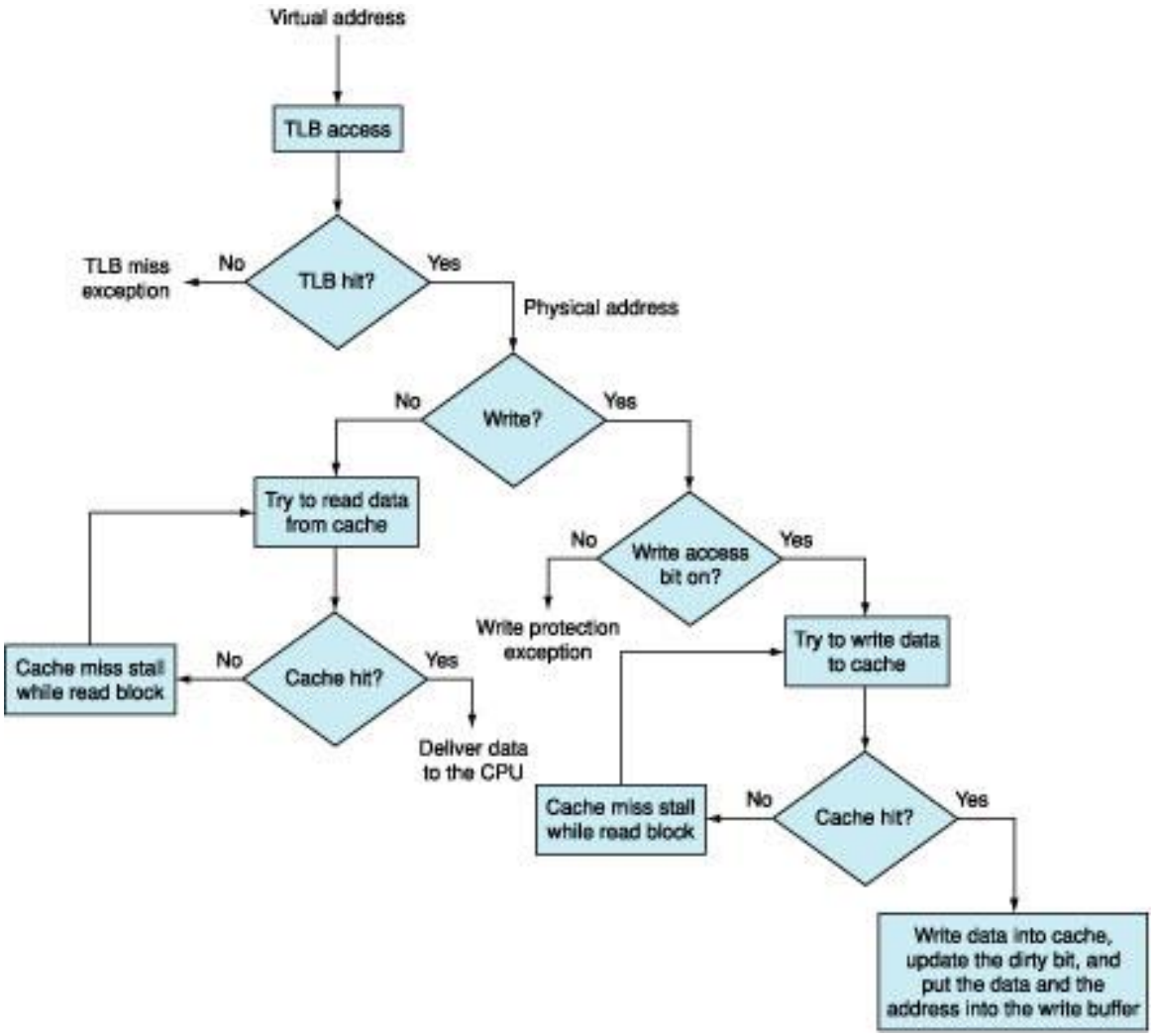
size: 16-512 entries
block size: 1-2 page table entries
hit time: 0.5-1 clock cycle
miss penalty: 10-100 clock cycles
miss rate: 0.01% - 1%



5.7 The Intrinsity FastMATH TLB



5.7 Processing a read or write-through



5.4 Overall Operation of a Memory Hierarchy

TLB	Page Table	Cache	Possible? If so, under what circumstance?
miss	miss	miss	TLB misses and is followed by a page fault; after retry, data must miss in cache
miss	miss	hit	Impossible: data cannot be allowed in cache if the page is not in memory
miss	hit	miss	TLB misses, but entry found in page table; after retry, data misses in cache
miss	hit	hit	TLB misses, but entry found in page table; after retry, data is found in cache
hit	miss	miss	Impossible: data cannot be allowed in cache if the page is not in memory
hit	miss	hit	Impossible: data cannot be allowed in cache if the page is not in memory
hit	hit	miss	Possible, though the page table is never really checked if TLB hits

5.7 Implementing Protection with Virtual Memory

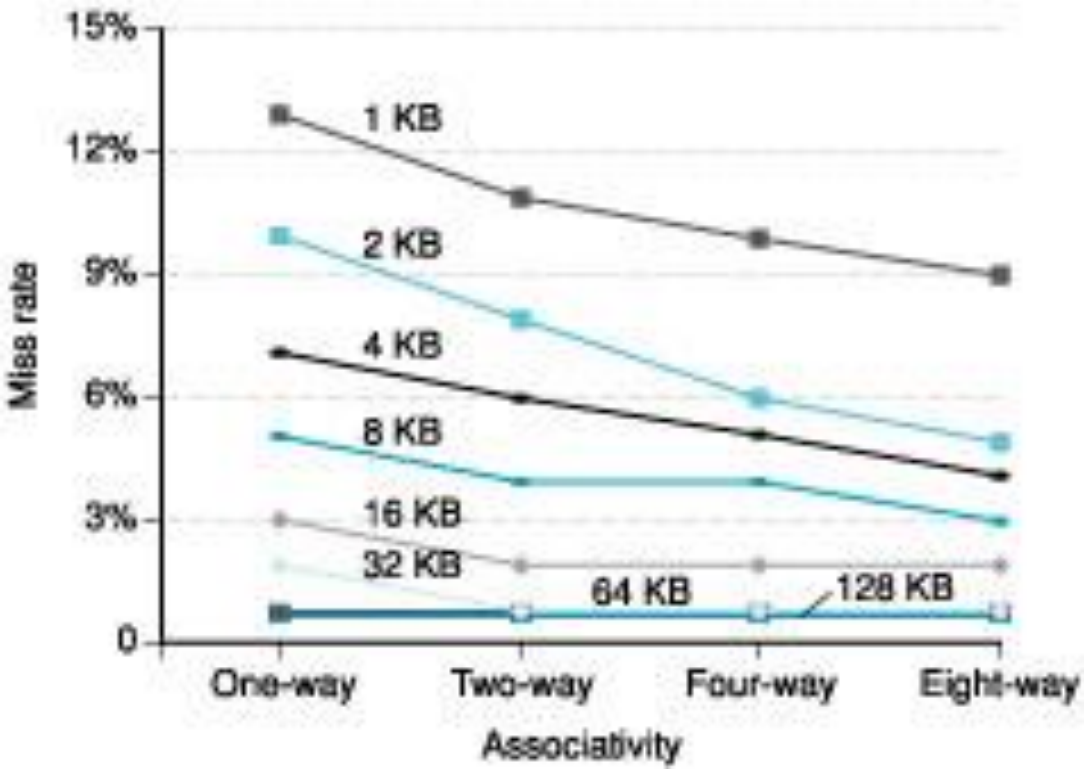
- Hardware Must Provide
  - At least two modes
    - user mode
    - privileged mode
  - Provide a portion of the processor state that a user process can read but not write
    - mode bit, page table pointer, TLB
  - Provide mechanisms whereby the processor can move between modes
    - system call user ⇒ system
    - return from exception system ⇒ user
- Software Can Help
  - Place the page tables in the protected address space of the operating system

5.7 Summary

- Pages are made large to take advantage of spatial locality and reduce the miss rate.
- The mapping between virtual addresses and physical addresses, which is implemented in a page table, is made fully associative so that a virtual page can be placed anywhere in main memory.
- The operating system uses techniques, such as LRU and a reference bit, to choose which pages to replace.

5.8 Where can a Block be Placed?

- A range of Associativities is possible
- Advantage: increasing associativity decreases miss rates.
- Disadvantage: Increasing associativity increases cost and slower access



5.8 How is a Block Found?

- Cache
  - _Small_ degrees of associativity are used because _large_ degrees are _expensive_
- Virtual Memory
  - _Full_ _associativity_ makes sense because
    - Misses are _very_ _expensive_
    - _Software_ can implement _sophisticated_ replacement schemes
    - Full map can be easily _indexed_
    - _Large_ items means small number of _mappings_

5.8 Replace Which Block on a Cache Miss?

- Cache
  - Random
  - LRU
- Virtual Memory
  - LRU

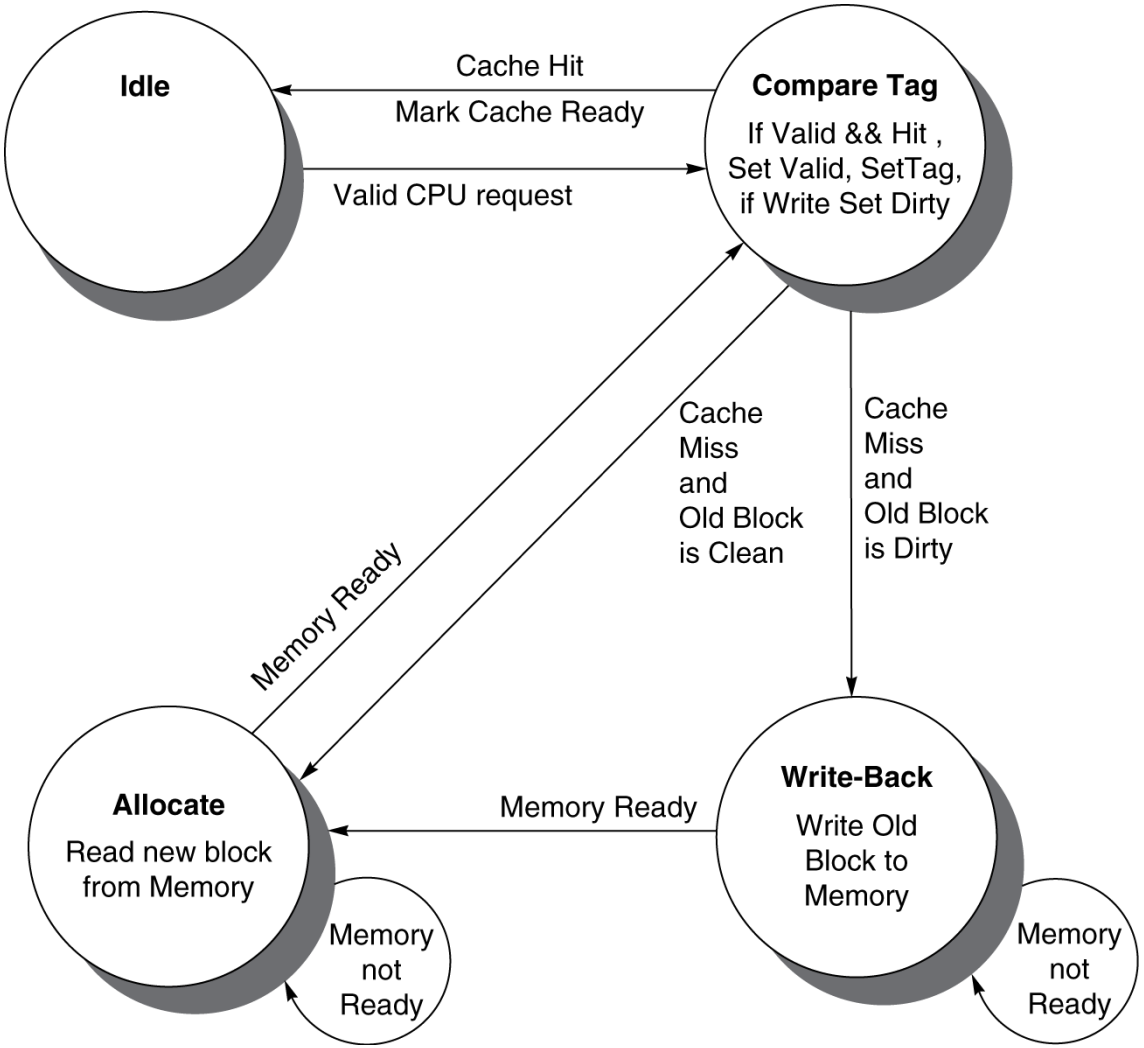
5.8 What Happens on a Write?

- Caches - write-back is the _strategy_ of the _future/present_
- Virtual memory – _Always_ uses write-back

5.8 The Three Cs

- Cache misses occur in three categories:
  - Compulsory misses - cache empty at startup
  - Capacity misses - can't hold them all
  - Conflict misses - map to the same place
- The challenge in designing memory hierarchies is that every change that potentially improves the miss rate can also negatively affect overall performance.

5.9 Finite State Cache Controller



5.10 Cache Coherence

- Multiple processors commonly share an address space
- They may bring data into cache and then write to their local copies
- These local copies are likely
  - Different from each other
  - Different from the main memory value
- Example

Time step	Event	CPU A's cache X	CPU B's cache X	Memory X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

5.10 Snooping

- Every cache has the sharing status of the block along with the block
- The caches are all accessible via some broadcast mechanism (bus or network).
- All cache controllers monitor or snoop on the medium to see whether or not they have a copy of the block requested.
- The writing CPU broadcasts an invalidate of the block.
- Example

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1

5.16 Concluding Remarks

- The difficulty of building a memory system to keep pace with faster processors is underscored by the fact that the raw material for main memory, DRAMs, is essentially the same in all computers, fast or slow.
- Multilevel caches facilitate cache optimizations.
  - different blok sizes
  - lower level cache has time to implement strategies
- Other trends
  - Reorganizing the program to enhance locality
  - Prefetching
    Special cache aware instructions