CS356 Unit 8

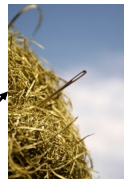
Memory

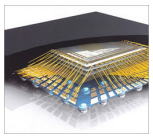
Performance Metrics

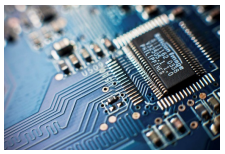
- Latency: Total time for a _____ to complete
  - Often hard to improve dramatically
  - Example: Takes roughly 4 years to get your bachelor's degree
  - From perspective of an _____
- Throughput/Bandwidth: _____ per operation
  - Usually much easier to improve by applying parallelism
  - From perspective of the _____
  - Example: A university can graduate more students per year by hiring more instructors or increasing class size

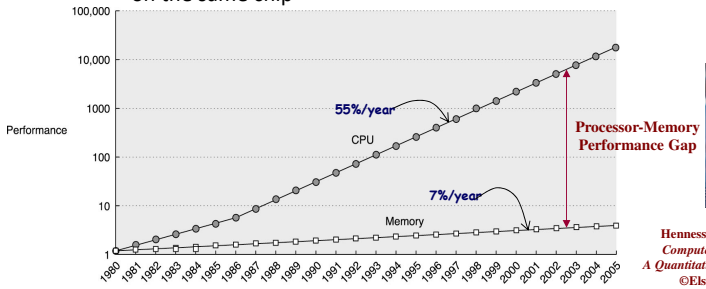
The Memory Wall

- Problem: The Memory Wall
  - Processor speeds have been increasing much faster than memory access speeds (Memory technology targets density rather than speed)
  - Large memories yield large address decode and _____ times
  - Main memory is physically located on separate chips and sending signals between chips takes a _____ than on the same chip









Hennessy and Patterson, *Computer Architecture – A Quantitative Approach* (2003) ©Elsevier Science

Options for Improving Performance

- Focus on latency by improving the _____
  - Can we improve the physical design of the basic memory circuits (i.e. the circuit that remembers a single bit) to create faster RAMs?
    - This is _____
  - Can we integrate memories on the _____ as our processing logic?
- Focus on _____ by improving the architecture/organization
  - Within a single memory, can we organize it in a more efficient manner to improve throughput
    - DRAM organization, DDR SDRAM, etc.
  - Can we use a _____ of memories to make the most expensive accesses far more rare
    - _____
  - These are generally _____ to do than latency improvements

Principle of Locality

- Most of the architectural improvements we make will seek to _____ the Principle of Locality
  - Explains why caching with a hierarchy of memories yields improvement gain
- Works in two dimensions
  - _____ **Locality**: If an item is referenced, items whose addresses are _____ will tend to be referenced _____
    - Examples: _____ and _____
  - _____ **Locality**: If an item is referenced, it will tend to be _____
    - Examples: _____, repeatedly called _____, setting a variable and then reusing it many times
    - 90/10 rule: Analysis shows that usually 10% of the _____ instructions account for 90% of the _____ instructions

Program Code

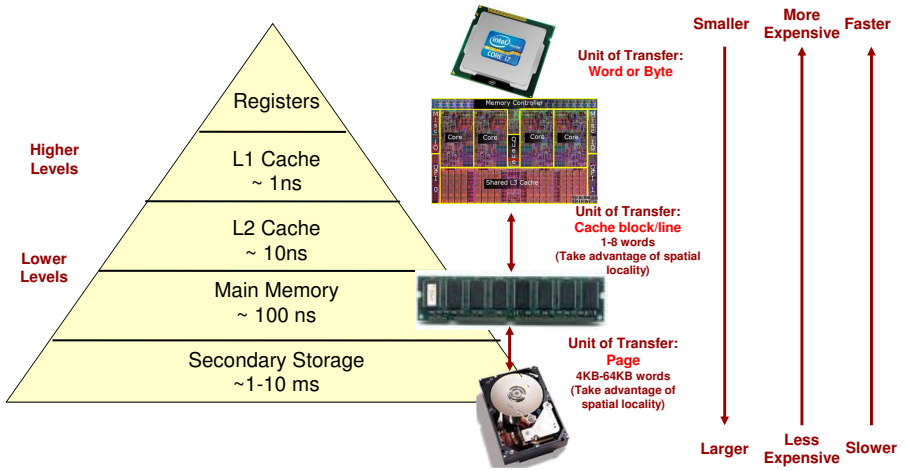
```
func4:
   movl  (%rdi), %eax
   movl  $1, %edx
   jmp   .L2
.L4:
   movslq  %edx, %rcx
   movl  (%rdi,%rcx,4), %ecx
   cmpl    %ecx, %eax
   jle     .L3
   movl    %ecx, %eax
.L3:
   addl    $1, %edx
.L2:
   cmpl    %esi, %edx
   jl      .L4
   ret
```

Arrays

data[5]	0000 0002	0x00214
data[4]	0000 0001	0x00210
data[3]	0000 0002	0x0020c
data[2]	0000 0001	0x00208
data[1]	0000 0002	0x00204
data[0]	0000 0001	0x00200

Memory Hierarchy & Caching

- General approach is to use several levels of faster and faster memory to hide delay of lower levels



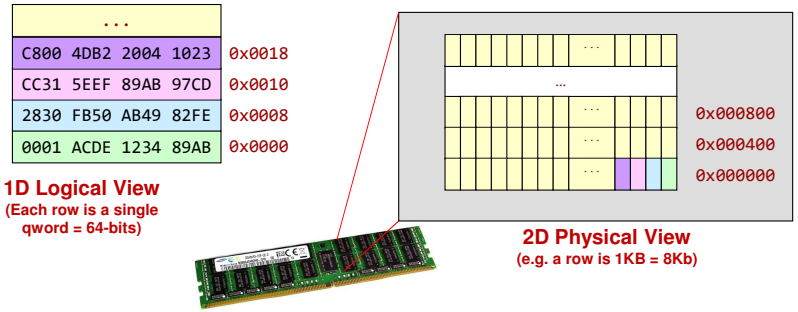
Hierarchy Access Time & Sizes

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

MEMORY ORGANIZATION

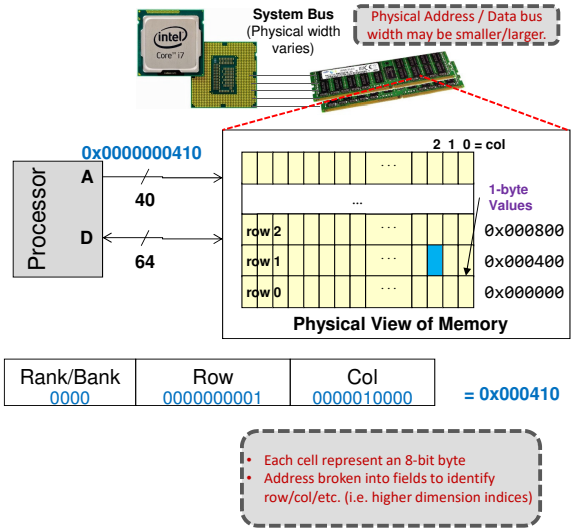
Memory Array

- Logical View = 1D array of rows (Dwords or Qwords)
  - Already this is 2D because each qword is 64-bits (i.e. (64) 1-bit columns)
- Physical View = 2D array of rows and columns
  - Each row may contain 1000's of columns (bits) though we have to access at least 8- (and often 16-, 32-, or 64-) bits at a time



Translating Addresses to 2D Indices

- While the programmer can keep their view of a linear (1D) address space, the hardware will translate the address into several indices (row, column, etc.) by _____ the address bits into _____
- Analogy: When you check into a hotel you receive 1 number but portions of the number represent _____ _____ (e.g. 612)
  - Floor: 6
  - Aisle: 1
  - Room: 2

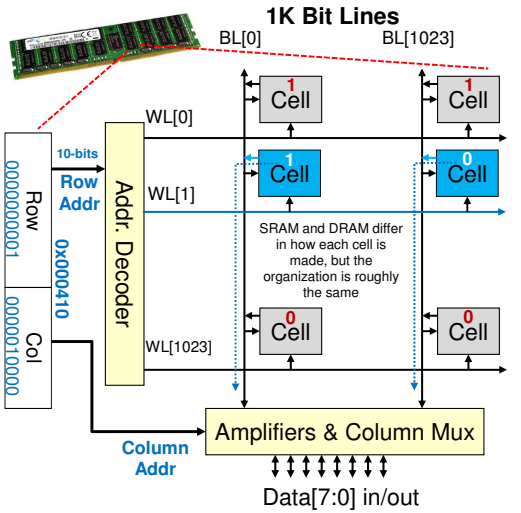


Main memory organization

DRAM TECHNOLOGIES

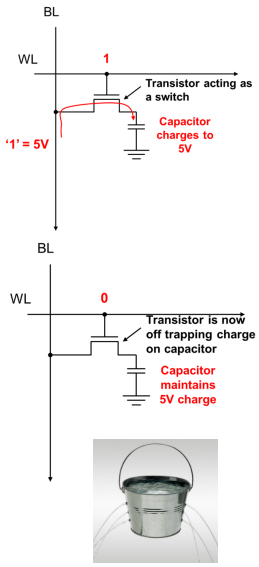
Memory Chip Organization

- Memory technologies share the same layout but differ in their cell implementation
  - _____
  - _____
- Memories require the row bits be sent first and are used to select one row (aka "____ line")
  - Uses a hardware component known as a decoder
- All cells in the selected row access their data bits and output them on their respective "_____"
- The column address is sent next and used to select the desired 8 bit lines (i.e. 1 byte)
  - Uses a hardware component known as a mux



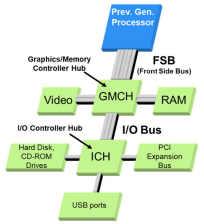
SRAM vs. DRAM

- Dynamic RAM (DRAM) Cells (store 1 bit)
  - Will _____if not refreshed periodically every few _____ [i.e. dynamic]
  - Extremely small (_____ & a capacitor)
    - Means we can have very high density (GB of RAM)
  - Small circuits require more time to access the bit
    - _____
  - Used for _____
- Static RAM (SRAM) Cells (store 1 bit)
  - Will retain values as long as _____ [i.e. static]
  - Larger (___ transistors)
  - Larger circuitry can access bit FASTER
  - Used for _____ memory

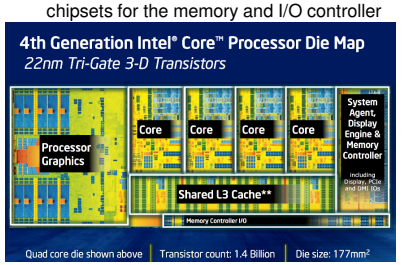


Memory Controller

- DRAMs require non-trivial hardware controller (aka memory controller)
  - To split up the address and send the row and column address as the right time
  - To periodically refresh the DRAM cells
  - Plus more…
- Used to require a separate chip from the processor
- But due to scaling (i.e. Moore's Law) most processors integrate the controller on-chip
  - Helps reduce access time since fewer hops



Legacy architectures used separate chipsets for the memory and I/O controller



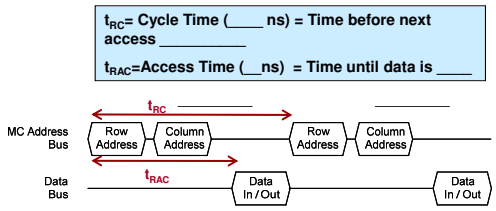
Current general-purpose processors usually integrate the memory controller on chip.

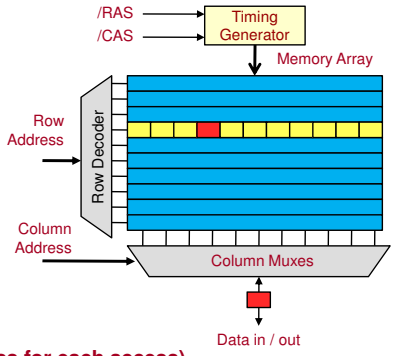
Implications of Memory Technology

- Memory latency of a single access using current DRAM technology will be slow
- We must improve bandwidth
  - Idea 1: Access _____ a single word at a time (to exploit spatial locality)
  - Technology: Fast Page Mode, DDR SDRAM, etc.
  - Idea 2: Increase number of accesses serviced in _____
  - Technology: Banking

Legacy DRAM Timing

- Can have only a single access "in-flight" at once
- Memory controller must send row and column address portions for each access

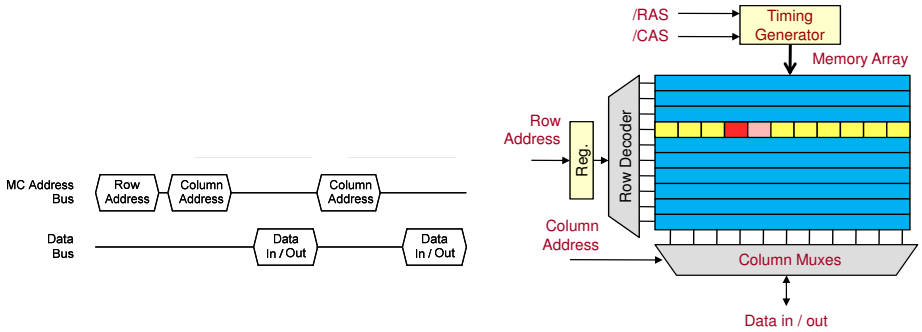




Legacy DRAM
(Must present new Row/Column address for each access)

Fast Page Mode DRAM Timing

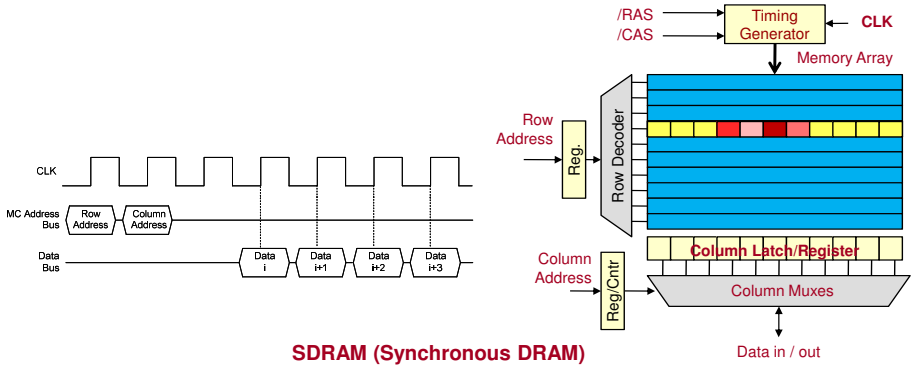
- Can provide _____ addresses with only one row address



Fast Page Mode
(Future address that fall in same row can pull data from the latched row)

Synchronous DRAM Timing

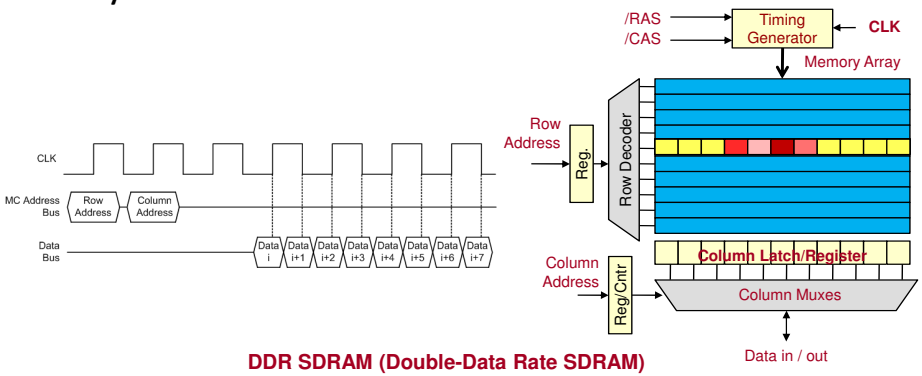
- Registers the column address and automatically increments it, accessing *n* sequential data words in *n* successive clocks called _____… n=_____ usually)



SDRAM (Synchronous DRAM)
Addition of clock signal. Will get up to 'n' consecutive words in the next 'n' clocks after column address is sent

DDR SDRAM Timing

- Double data rate access data every _____ clock cycle



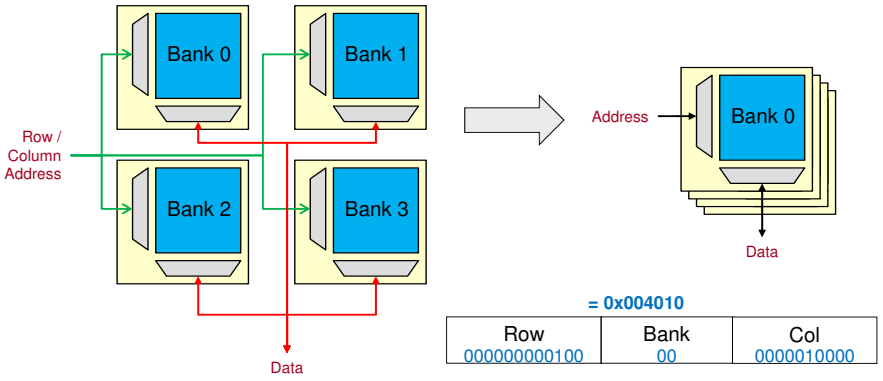
DDR SDRAM (Double-Data Rate SDRAM)
Addition of clock signal. Will get up to '2n' consecutive words in the next 'n' clocks after column address is sent

Key Point About Main Memory

- Time to access a sequential chunk of bytes in RAM (main memory) has two components
  - Time to find the _____ of a chunk (this is _____)
  - Time to access each _____ byte (this is _____)
- Accessing a chunk of N _____ bytes is far faster than N _____ bytes

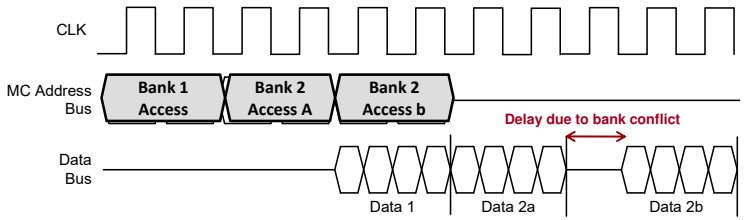
Banking

- Divide memory into "banks" duplicating row/column decoder and other peripheral logic to create _____ memory arrays that can access data in _____
  - uses a _____ of the address to determine which bank to access



Bank Access Timing

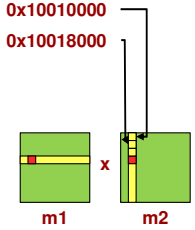
- Consecutive accesses to different banks can be _____ and hide the time to access the row and select the column
- Consecutive accesses within a bank (to different rows) _____ the access latency



Access 1 maps to bank 1 while access 2a maps to bank 2 allowing parallel access. However, access 2b immediately follows and maps to bank 2 causing a delay.

Programming Considerations

- For memory configuration given earlier, accesses to the same bank but different row occur on an 32KB boundary
- Now consider a matrix multiply of 8K x 8K integer matrices (i.e. 32KB x 32KB)
- In code below…m2[0][0] @ 0x10010000 while m2[1][0] @ 0x10018000



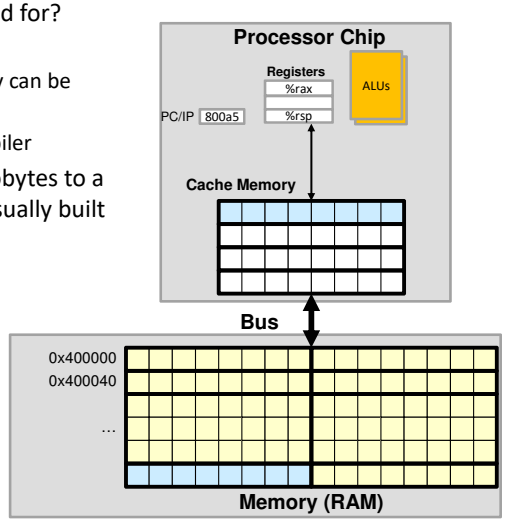
Unused	Row	Bank	Col.
A31-A29	A28…A15	A14,A13	A12…A0
00	1 0000 0000 0001 0	00	0000000000000
00	1 0000 0000 0001 1	00	0000000000000

```
int m1[8192][8192], m2[8192][8192], result[8192][8192];
int i,j,k;
...
for(i=0; i < 8192; i++){
  for(j=0; j < 8192; j++){
    result[i][j]=0;
    for(k=0; k < 8192; k++){
      result[i][j] += matrix1[i][k] * matrix2[k][j];
} } }
```


CACHING

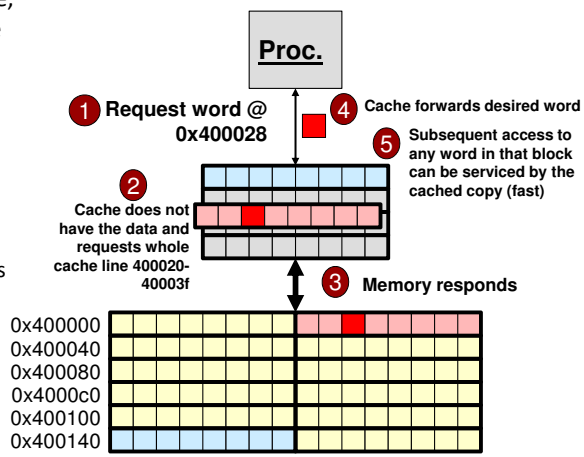
Cache Overview

- Remember what registers are used for?
  - Quick access to copies of data
  - Only a few (32 or 64) so that they can be accessed really quickly
  - Controlled by the software/compiler
- Cache memory is a small-ish, (kilobytes to a few megabytes) "fast" memory usually built onto the processor chip
- Will hold _____ of the latest data & instructions accessed by the processor
- Managed by the _____
  - _____ to the software



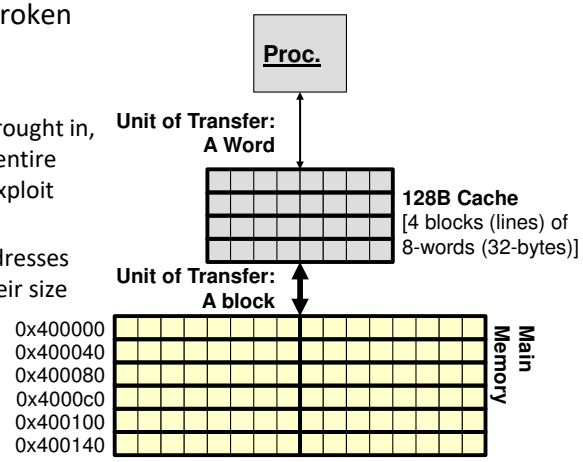
Cache Blocks/Lines

- Whenever the processor generates a read or a write, it will first check the cache memory to see if it contains the desired data
  - If so, it can get the data quickly _____
  - Otherwise, it must go to the _____ main memory to get the data (but subsequent accesses can be serviced by the cache)



Cache Blocks/Lines

- To exploit spatial locality, cache memory is broken into "_____" or "_____"
  - Any time data is brought in, it will bring in the entire block of data (to exploit spatial locality)
  - Blocks start on addresses _____ of their size



Cache and Locality

- Caches take advantage of locality
- Spatial Locality
  - Caches do not store individual words but blocks of words (a.k.a. "cache line" or "cache block")
  - Caches always bring in a block or line of sequential words because if we access one, we are likely to access the next
  - Bringing in blocks of sequential words takes advantage of _____ (i.e. _____, etc.)
- Temporal Locality
  - Leave data in the cache because it will likely be accessed again

Examples of Caching Used

- What is caching?
  - Maintaining copies of information in locations that are faster to access than their primary home
- Examples
  - Data/instruction caches
  - TLB
  - Branch predictors
  - VM
  - Web browser
  - File I/O (disk cache)
  - Internet name resolutions

IMPLEMENTATION ISSUES

Cache Definitions

- **Cache _____** = Desired data is ____ current level of cache
  - Can be further distinguished as read hit vs. write hit
- **Cache _____** = Desired data is _____ in current level
  - Can be further distinguished as read miss vs. write miss
- When a cache miss occurs, the new block is brought from the lower level into cache
  - If cache is full, a block must be _____
- When CPU writes to cache, we may use one of two policies:
  - **Write Through (Store Through)**: Every write updates both _____ and _____ level of cache to keep them in sync. (i.e. coherent)
  - **Write Back**: Let the CPU keep writing to cache at fast rate, not updating the next level. Only _____ to the next level when it needs to be replaced or flushed

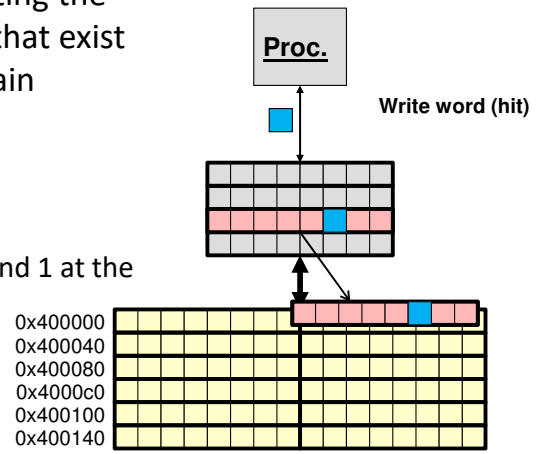
Primary Implementation Issues

- Write Policies
- Replacement algorithms
- Finding cached data (hit/miss)
  - Mapping Algorithms
- Coherency (managing multiple versions)
  - Discussed in future lectures

Write Policies

- On a write-hit how should we handle updating the multiple copies that exist (in cache and main memory)?
- Options:
  - Update both
  - Update 1 now and 1 at the end

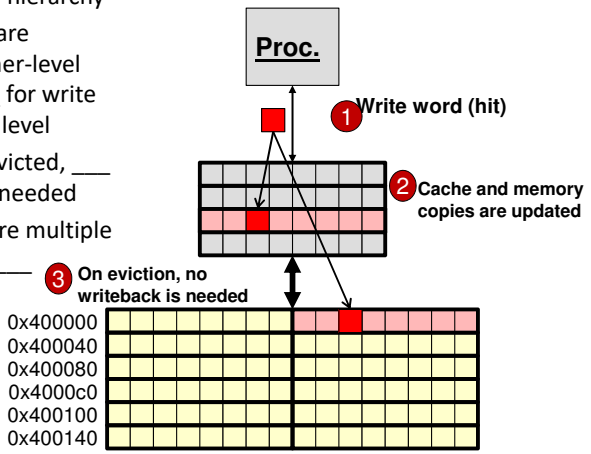
Analogy: A movie star who changes their mind about what to eat for lunch, and the assistant who has to communicate with the chef



Write Through Cache

- Write-through option:
  - Update both levels of hierarchy
  - Depending on hardware implementation, higher-level may have to _____ for write to complete to lower level
  - Later when block is evicted, ___ _____ is needed
  - _____ writes require multiple main memory _____

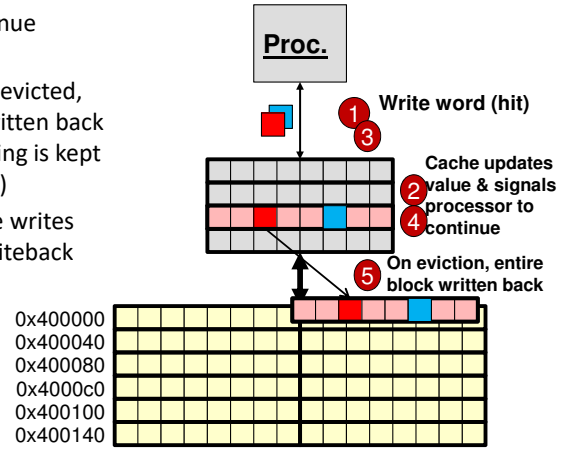
Key Idea: Communicate **EVERY** change to main memory as they happen (keeps both copies in sync)



Write Back Cache

- Write-back option:
  - Update _____ cached copy
  - Processor can continue quickly
  - Later when block is evicted, _____ block is written back (because bookkeeping is kept on a per block basis)
  - Notice that multiple writes only require ___ writeback upon eviction

Key Idea: Communicate **ONLY the FINAL version** of a block to main memory (when the block is evicted)



Write-through vs. Writeback

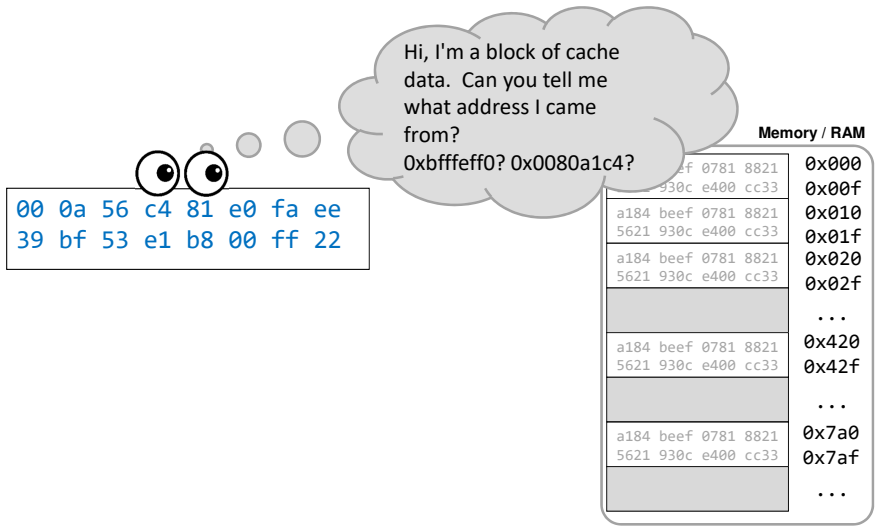
- Write-through
  - Pros: Keep both versions in synch at all times
  - Cons: Poor performance if next level of hierarchy is slow (see virtual memory) or if many, repeated accesses
- Writeback
  - Pros: Fast if many repeated accesses
  - Cons:
    - Coherency issues
    - Slow if few, isolated writes since entire block must be written back
- In practice
  - Writeback must be used for lower levels of hierarchy where the next level is extremely slow
  - Even at higher levels writeback is often used (Most Intel L1 caches are writeback)

Replacement Policies

- On a read- or write-miss, a new block must be brought in
- This requires evicting a current block residing in the cache
- Replacement policies
  - _____
  - _____
  - _____

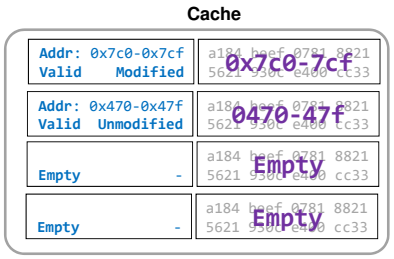
MAPPINGS

Cache Question



Cache Implementation

- Assume a cache of 4 blocks of 16-bytes each
- Must store more than just data!
- What other bookkeeping and identification info is needed?
  - Has the block been _____
  - Is the block _____ or _____
  - _____ of the data: Where did I come from?

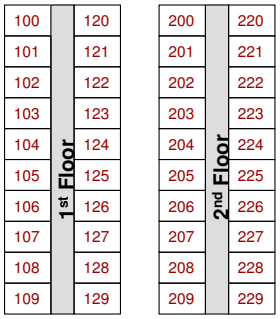


Implementation Terminology

- What bookkeeping values must be stored with the cache in addition to the block data?
- _____ – Portion of the block's address range used to identify the MM block residing in the cache from other MM blocks.
- _____ **bit** – Indicates the block is occupied with valid data (i.e. not empty or invalid)
- _____ **bit** – Indicates the cache and MM copies are "inconsistent" (i.e. a write has been done to the cached copy but not the main memory copy)
  - Used for _____ caches

Identifying Blocks via Address Range

- Possible methods
  - Store start and end address (requires multiple comparisons)
  - Ensure block ranges sit on binary boundaries (upper address bits identify the block with a single value)
    - Analogy: Hotel room layout/addressing



Analogy: Hotel Rooms

1st Digit = Floor
2nd Digit = Aisle
3rd Digit = Room w/in aisle

To refer to the range of rooms on the second floor, left aisle we would just say rooms **20x**

4 word (16-byte) blocks:

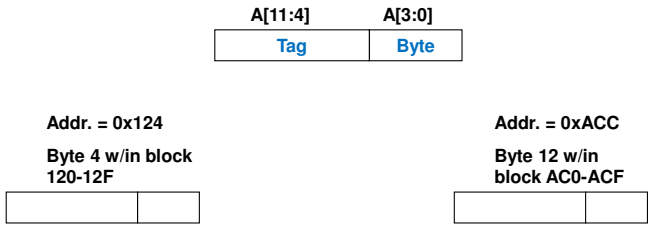
Addr. Range	Binary		
000-00f	0000	0000	0000 - 1111
010-01f	0000	0001	0000 - 1111

8 word (32-byte) blocks:

Addr. Range	Binary		
000-01f	0000	000	00000 - 11111
020-03f	0000	001	00000 - 11111

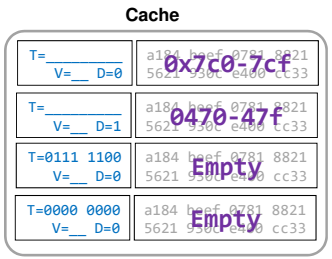
Cache Implementation

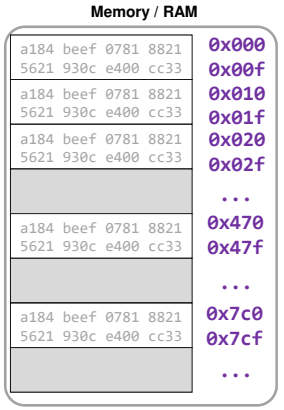
- Assume 12-bit addresses and 16-byte blocks
- Block addresses will range from xx0-xxF
  - Address can be broken down as follows
  - A[11:4] = Tag = Identifies block range (i.e. xx0-xxF)
  - A[3:0] = Byte offset within the cache block



Cache Implementation

- To identify which MM block resides in each cache block, the tags need to be stored along with the "dirty/modified" and "valid" bits



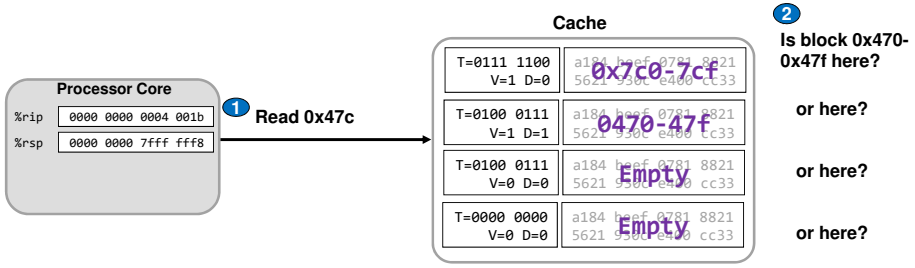


Scenario

- You lost your keys
- You think back to where you have been lately
  - You've been the library, to class, to grab food at campus center, and the gym
  - Where do you have to look to find your keys?
- If you had been home all day and discovered your keys were missing, where would you have to look?
- **Key lesson**: If something can be anywhere you have to search _____
  - By contrast, if we limit where things can be then our search need only look in those _____ places

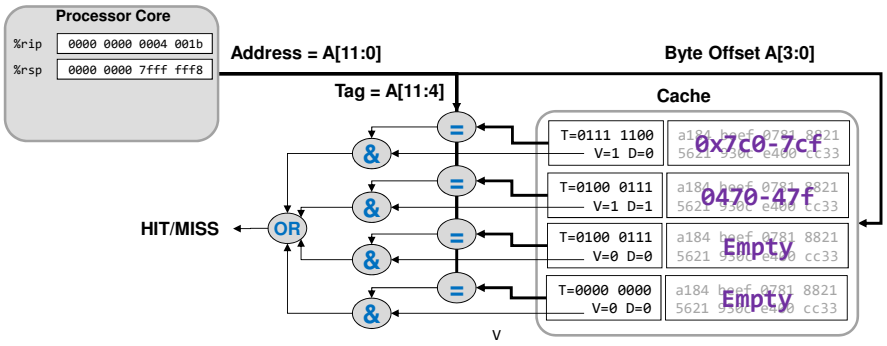
Content-Addressable Memory

- Cache memory is one form of what is known as "content-addressable" memory
  - This means data can be in any location in memory and does not have one particular address
  - Additional information is saved with the data and is used to "address"/find the desired data (this is the "tag" in this case) via a search on each access
  - This search can be very _____!!



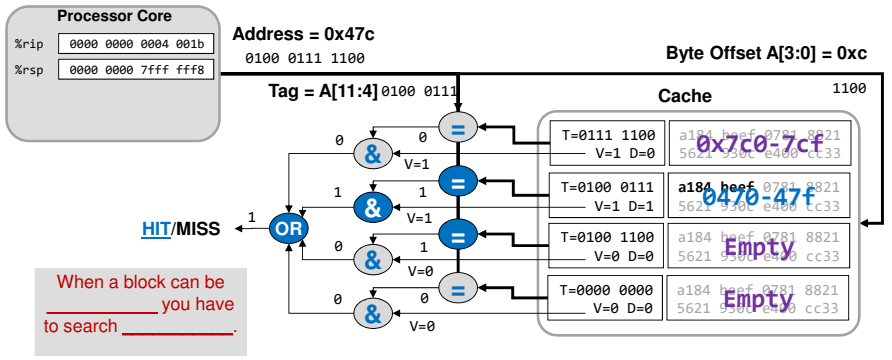
Tag Comparison

- When caches have many blocks (> 16 or 32) it can be expensive (hardware-wise) to check all tags



Tag Comparison Example

- Tag portion of desired address is check against all the tags and qualified with the valid bits to determine a hit

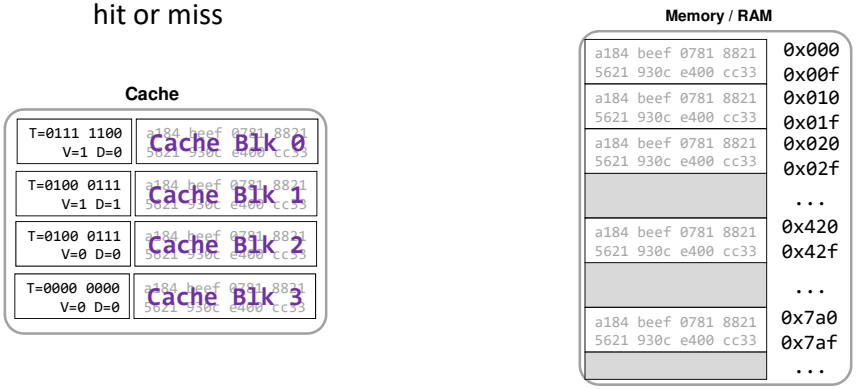


Mapping Techniques

- Determines where blocks can be _____ in the cache
- By reducing number of possible MM blocks that map to a cache block, hit logic (searches) can be done faster
- 3 Primary Methods
  - _____ Mapping
  - _____ Associative Mapping
  - _____-Associative Mapping

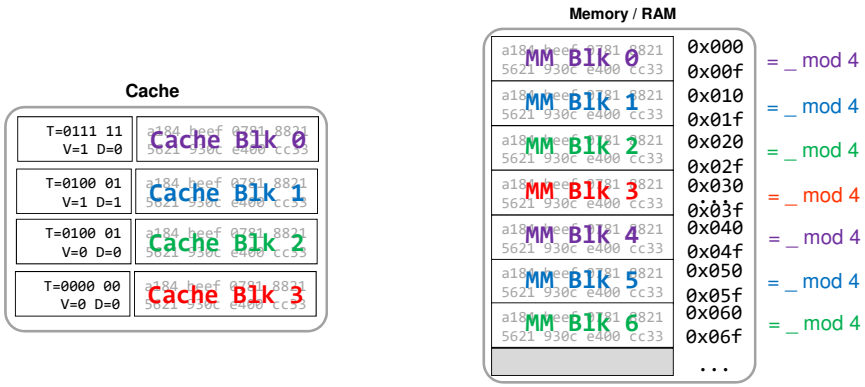
Fully Associative Mapping

- Any block from memory can be put in any cache block (i.e. no restriction)
  - Implies we have to search _____ to determine hit or miss



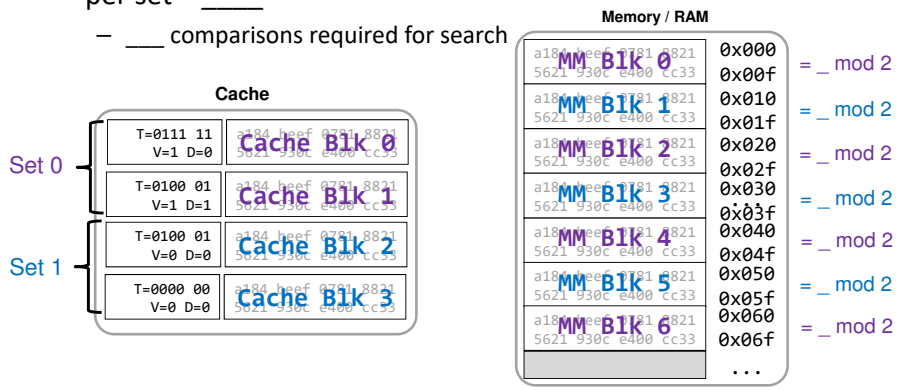
Direct Mapping

- Each block from memory can only be put in one location
- Given n cache blocks,
  MM block i maps to cache block i mod n



K-way Set-Associative Mapping

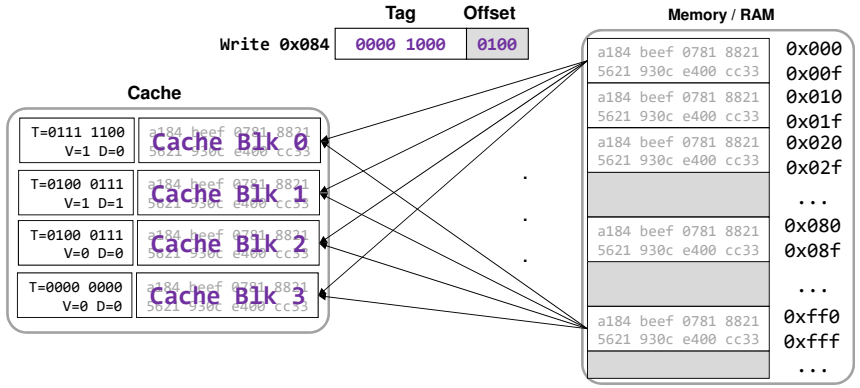
- Given, S sets, block i of MM maps to set i mod S
- Within the set, block can be put anywhere
- Given N=total cache blocks, let K = number of cache blocks per set = ____
  - ___ comparisons required for search



Fully Associative Implementation

- Assume 12 address bits

Offset	B=16 bytes per block _____ offset bits	Determines byte/word within the block
Tag	Remaining bits	Identifies the MM address from where the block came

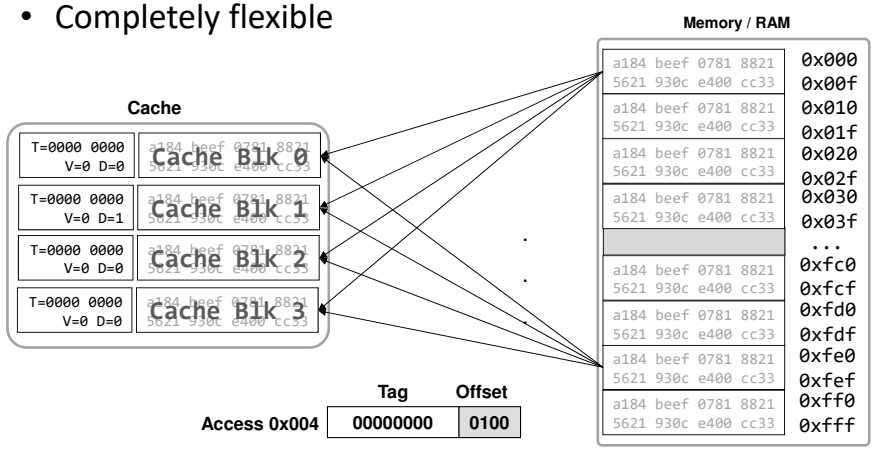


Fully Associative Address Scheme

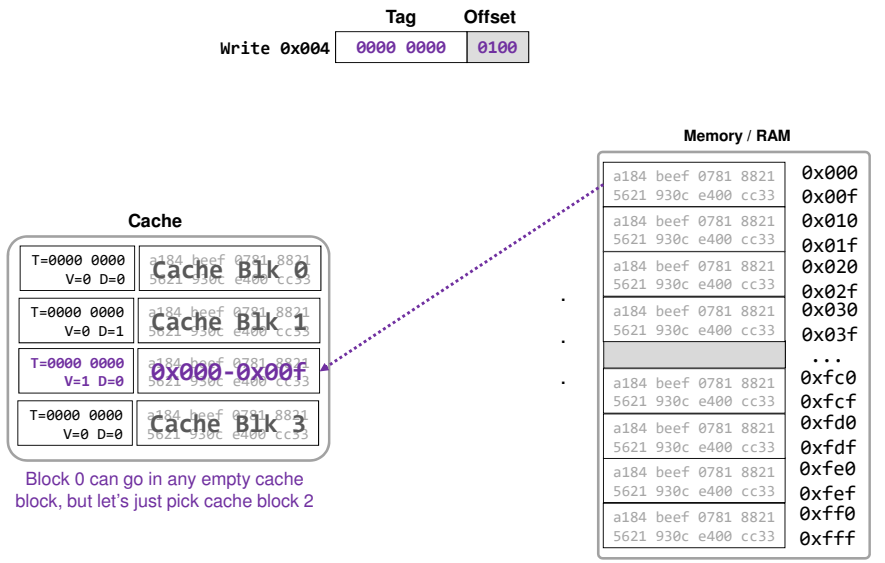
- Byte offset bits = _____ bits (B=Block Size)
- Tag = Remaining bits

Fully Associative Mapping

- Any block from memory can be put in any cache block (i.e. no mapping scheme)
- Completely flexible



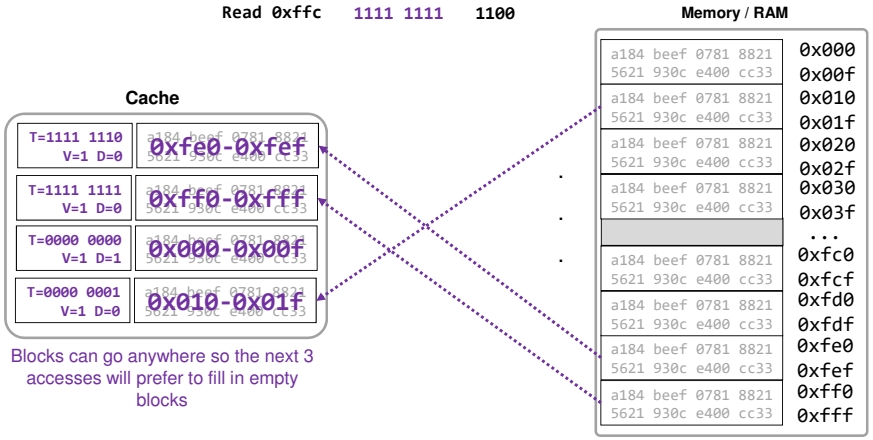
Fully Associative Mapping



Block 0 can go in any empty cache block, but let's just pick cache block 2

Fully Associative Mapping

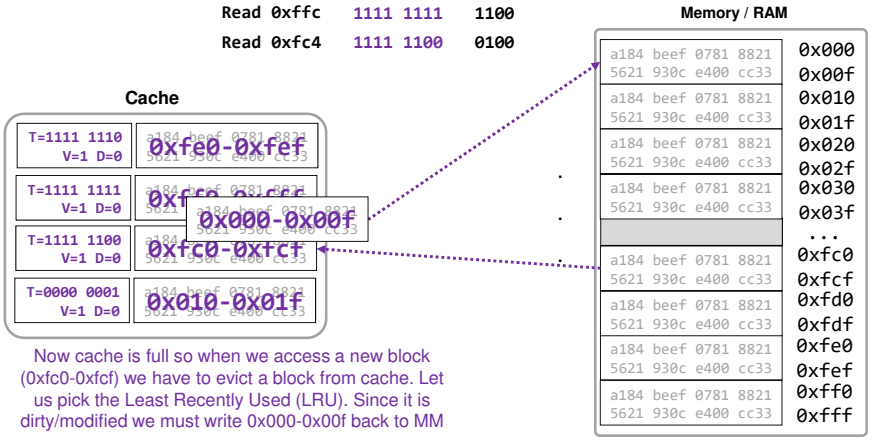
	Tag	Offset
Write 0x004	0000 0000	0100
Read 0x018	0000 0001	1000
Read 0xfe0	1111 1110	0000
Read 0xffc	1111 1111	1100



Blocks can go anywhere so the next 3 accesses will prefer to fill in empty blocks

Fully Associative Mapping

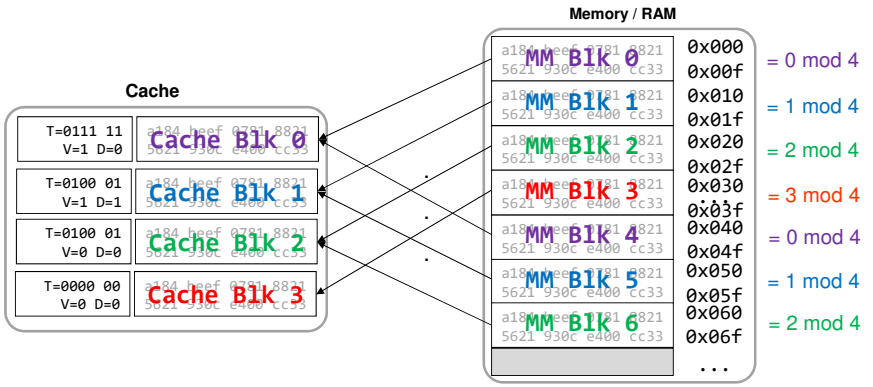
	Tag	Offset
Write 0x004	0000 0000	0100
Read 0x018	0000 0001	1000
Read 0xfe0	1111 1110	0000
Read 0xffc	1111 1111	1100
Read 0xfc4	1111 1100	0100



Now cache is full so when we access a new block (0xfc0-0xfcf) we have to evict a block from cache. Let us pick the Least Recently Used (LRU). Since it is dirty/modified we must write 0x000-0x00f back to MM

Direct Mapping

- Each block from memory can only be put in one location
- Given N total cache blocks,
  MM block i maps to cache block _____



Direct Mapping Address Scheme

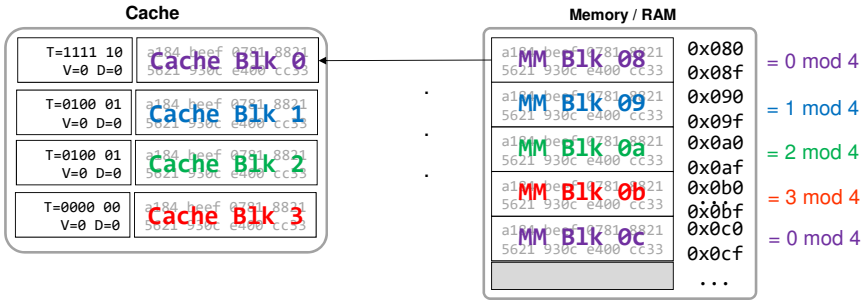
- Byte offset bits = $\log_2 B$ bits (B=Block Size)
- Block bits = _____ (N=# of Cache Blocks)
- Tag = Remaining bits

Direct Mapping Implementation

- Assume 12 address bits

Offset	B=16 bytes per block $\log_2 B = 4$ offset bits	Determines byte/word within the block
Block	N=4 blocks in the cache _____ block bits	Performs hash function (i mod N)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0, 4, 8, …)

	Tag	Block	Offset
Write 0x084			0100

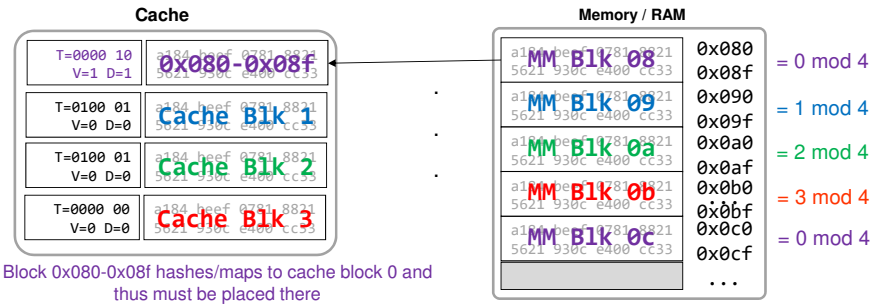


Direct Mapping Implementation

- Assume 12 address bits

Offset	B=16 bytes per block $\log_2 B = 4$ offset bits	Determines byte/word within the block
Block	N=4 blocks in the cache $\log_2 N = 2$ block bits	Performs hash function (i mod N)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0, 4, 8, …)

	Tag	Block	Offset
Write 0x084	0000 10	00	0100



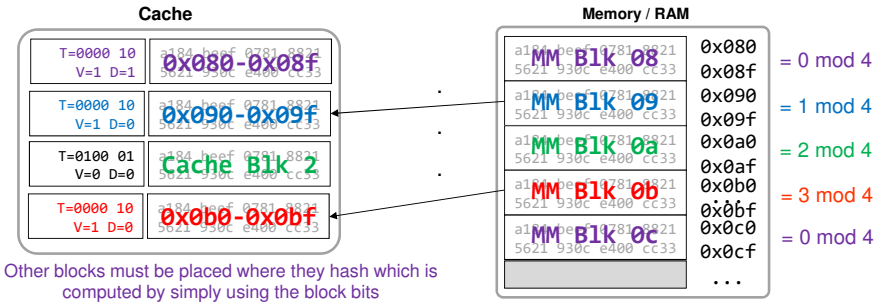
Block 0x080-0x08f hashes/maps to cache block 0 and thus must be placed there

Direct Mapping Implementation

- Assume 12 address bits

Offset	B=16 bytes per block $\log_2 B = 4$ offset bits	Determines byte/word within the block
Block	N=4 blocks in the cache $\log_2 N = 2$ block bits	Performs hash function (i mod N)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0, 4, 8, …)

	Tag	Block	Offset
Write 0x084	0000 10	00	0100
Read 0x09c	0000	__	1100
Read 0x0b8	0000	__	1000



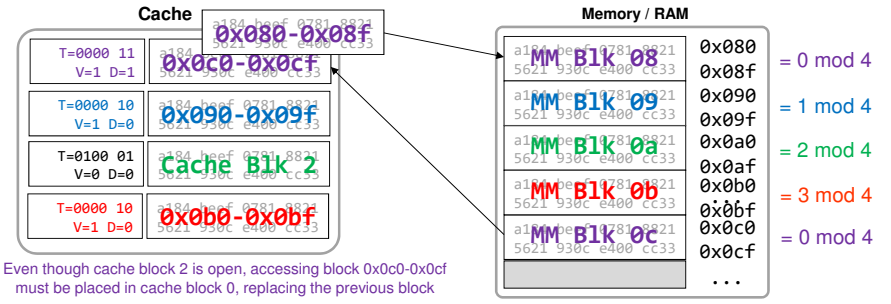
Other blocks must be placed where they hash which is computed by simply using the block bits

Direct Mapping Implementation

- Assume 12 address bits

Offset	B=16 bytes per block $\log_2 B = 4$ offset bits	Determines byte/word within the block
Block	N=4 blocks in the cache $\log_2 N = 2$ block bits	Performs hash function (i mod N)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0, 4, 8, …)

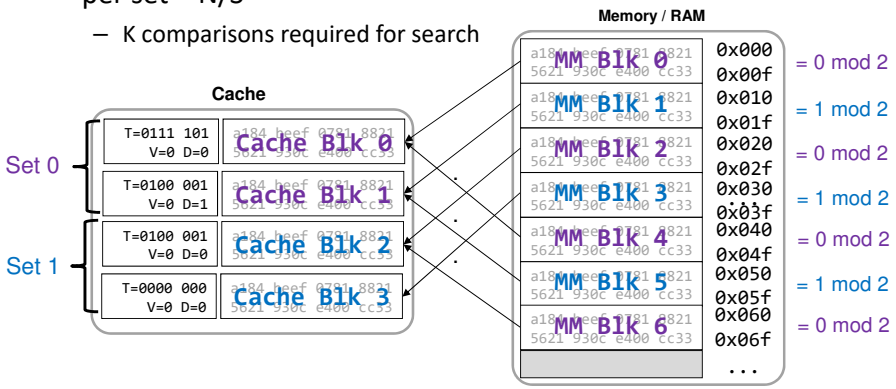
	Tag	Block	Offset
Write 0x084	0000 10	00	0100
Read 0x09c	0000 10	01	1100
Read 0x0b8	0000 10	11	1000
Read 0x0c8	0000	__	1000



Even though cache block 2 is open, accessing block 0x0c0-0x0cf must be placed in cache block 0, replacing the previous block

K-way Set-Associative Mapping

- Given, S sets, block i of MM maps to set i mod s
- Within the set, block can be put anywhere
- Given N=total cache blocks, let K = number of cache blocks per set = N/S
  - K comparisons required for search



K-Way Set Associative Address Scheme

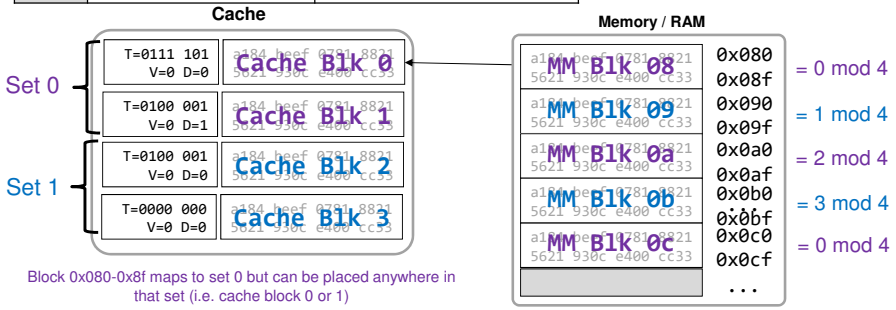
- Byte offset bits = $\log_2$B bits (B=Block Size)
- Set bits = _____ bits (S=# of Cache Sets)
- Tag = Remaining bits

K-way Set-Associative Mapping

- Assume 12-bit addresses

Offset	B=16 bytes per block $\log_2$B = 4 offset bits	Determines byte/word within the block
Set	S=_____ sets _____ set bit(s)	Performs hash function (i mod S)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0x00,..., 0x08, 0x0a, 0x0c, ...)

Write 0x084 | Tag | Set | Offset | 0100 |



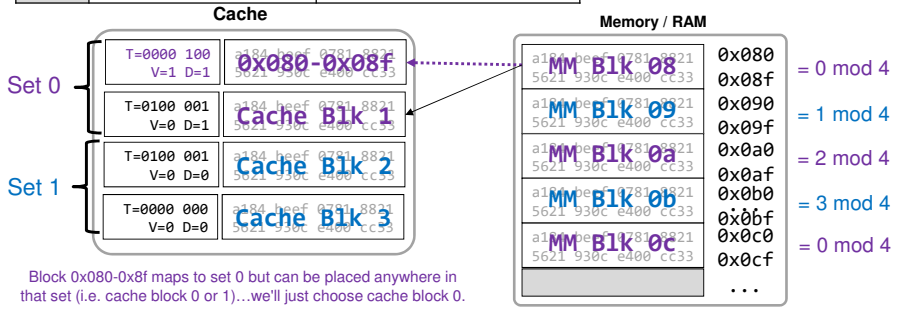
Block 0x080-0x8f maps to set 0 but can be placed anywhere in that set (i.e. cache block 0 or 1)

K-way Set-Associative Mapping

- Assume 12-bit addresses

Offset	B=16 bytes per block $\log_2$B = 4 offset bits	Determines byte/word within the block
Set	S=N/K=2 sets $\log_2$S = 1 set bit	Performs hash function (i mod S)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0x00,..., 0x08, 0x0a, 0x0c, ...)

Write 0x084 | Tag: 0000 100 | Set: 0 | Offset: 0100 |



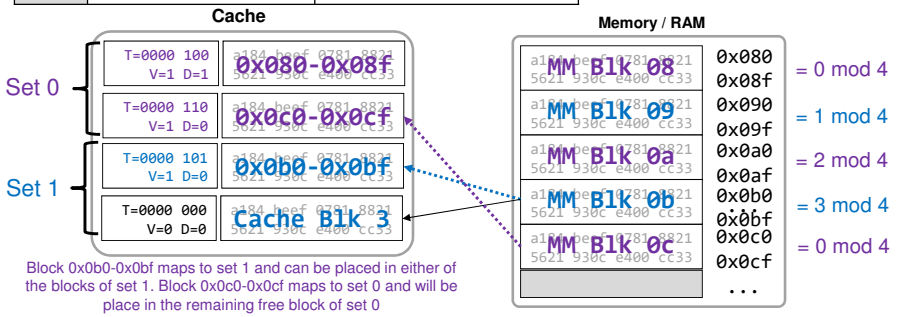
Block 0x080-0x8f maps to set 0 but can be placed anywhere in that set (i.e. cache block 0 or 1)...we'll just choose cache block 0.

K-way Set-Associative Mapping

- Assume 12-bit addresses

Offset	B=16 bytes per block $\log_2 B = 4$ offset bits	Determines byte/word within the block
Set	S=N/K=2 sets $\log_2 S = 1$ set bit	Performs hash function (i mod S)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0x00,…, 0x08, 0x0a, 0x0c, …)

	Tag	Set	Offset
Write 0x084	0000 100	0	0100
Read 0x0b0	0000 ___	_	0000
Read 0x0c8	0000 ___	_	1000



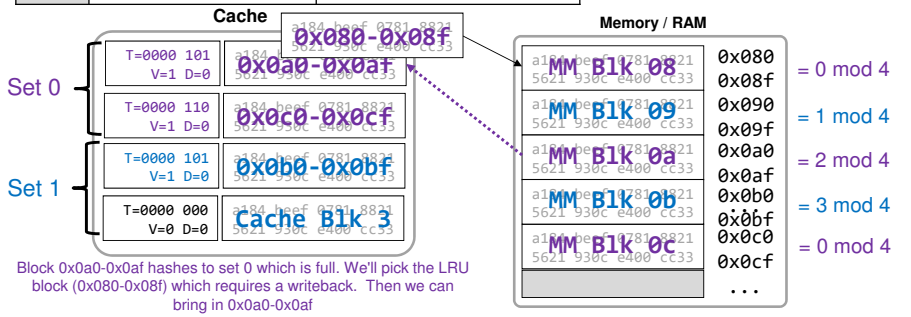
Block 0x0b0-0x0bf maps to set 1 and can be placed in either of the blocks of set 1. Block 0x0c0-0x0cf maps to set 0 and will be place in the remaining free block of set 0

K-way Set-Associative Mapping

- Assume 12-bit addresses

Offset	B=16 bytes per block $\log_2 B = 4$ offset bits	Determines byte/word within the block
Set	S=N/K=2 sets $\log_2 S = 1$ set bit	Performs hash function (i mod S)
Tag	Remaining bits	Identifies blocks that map to the same bucket (block 0x00,…, 0x08, 0x0a, 0x0c, …)

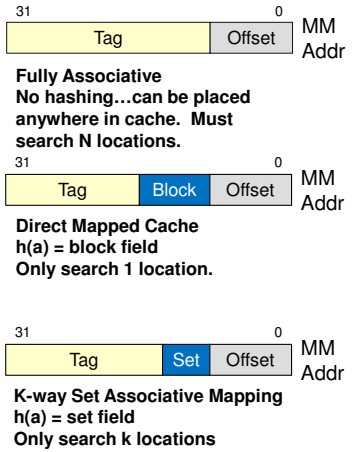
	Tag	Set	Offset
Write 0x084	0000 100	0	0100
Read 0x0b0	0000 101	1	0000
Read 0x0c8	0000 110	0	1000
Read 0x0a4	0000 ___	_	0100



Block 0x0a0-0x0af hashes to set 0 which is full. We'll pick the LRU block (0x080-0x08f) which requires a writeback. Then we can bring in 0x0a0-0x0af

Summary of Mapping Schemes

- Fully associative
  - Most flexible _____
  - _____ search time ____
- Direct-mapped cache
  - Least flexible (more evictions)
  - _____ search time ____
- K-way Set Associative mapping
  - Compromise
    - 1-way set associative = _____
    - N-way set associative = _____
  - Work to search is ___ [k is usually small enough to be done in parallel => O(_)]



Address Mapping Examples

- 16-bit addresses, 2 KB cache, 32 bytes/block
- Find address mapping for:
  - Fully Associative
  - Direct Mapping
  - 4-way Set Associative
  - 8-way Set Associative

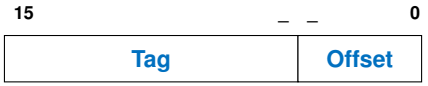
Address Mapping Examples

- First find parameters:
  - B = Block size
  - N = Cache blocks
  - S = Sets for 4-way and 8-way
- B is given as 32 bytes/block
- N depends on cache size and block size
  - N =
- S for 4-way & 8-way
  - $S_{4\text{-way}}$ = N/k =
  - $S_{8\text{-way}}$ = N/k =

Fully Associative

- Offset =
- Tag =

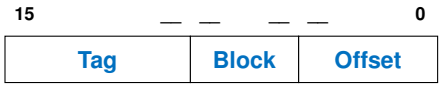
Parameters:
B = 32
N = 64
S4-way = 16
S8-way = 8



Direct Mapping

- Offset =
- Block =
- Tag =

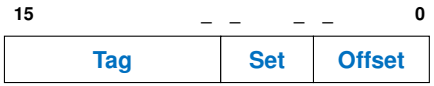
Parameters:
B = 32
N = 64
S4-way = 16
S8-way = 8



4-Way Set Assoc. Mapping

- Offset =
- Set =
- Tag =

Parameters:
B = 32
N = 64
S4-way = 16
S8-way = 8



8-Way Set Assoc. Mapping

- Offset =
- Set =
- Tag =

Parameters:
B = 32
N = 64
S4-way = 16
S8-way = 8

Cache Operation Example

- Address Trace
  - R: 0x00a0
  - W: 0x00f4
  - R: 0x00b0
  - W: 0x2a2c
- Operations
  - Hit
  - Fetch block XX
  - Evict block XX (w/ or w/o WB)
  - Final WB of block XX)

- Perform address breakdown and apply address trace
- 2-Way Set-Assoc, N=4, B=32 bytes/block

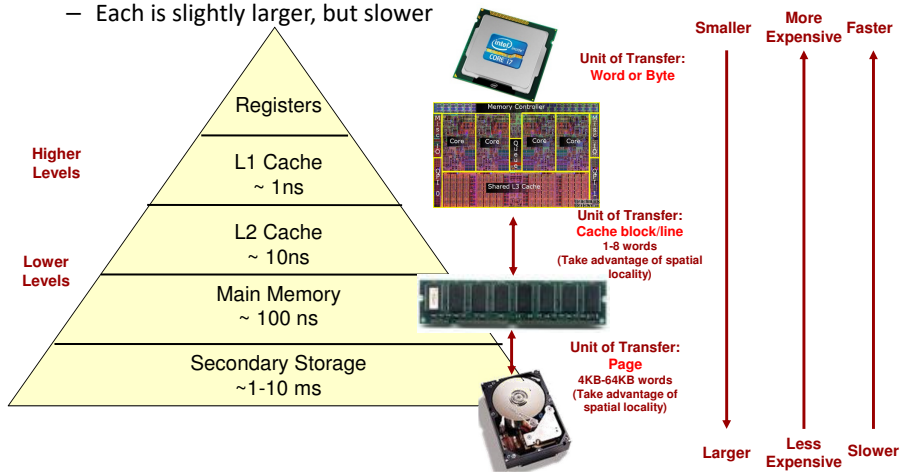
Address	Tag	Set	Byte Offset
0x00a0			
0x00f4			
0x00b0			
0x2a2c			

Processor Access	Cache Operation
R: 0x00a0	
W: 0x00f4	
R: 0x00b0	
W: 0x2a2c	
Done!	

ADDING MULTIPLE LEVELS OF CACHE

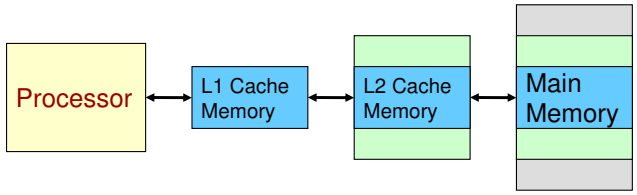
More of a Good Thing

- If one cache was good, more is likely better
  - Add a Level 2 and even Level 3 cache
  - Each is slightly larger, but slower



Principle of Inclusion

- When the cache at level i misses on data that is stored in level k (i < k), the data is brought into all levels j where _____
- This implies that lower levels always contains a _____ of higher levels
- Example:
  - L1 contains most recently used data
  - L2 contains that data + data used earlier
  - MM contains all data
- This make coherence far easier to maintain between levels



Average Access Time

- Define parameters
  - H_i = Hit Rate of Cache Level L_i (Note that 1-H_i = Miss rate)
  - T_i = Access time of level i
  - R_i = Burst rate per word of level i (after startup access time)
  - B = Block Size
- Let us find T_{AVE} = average access time

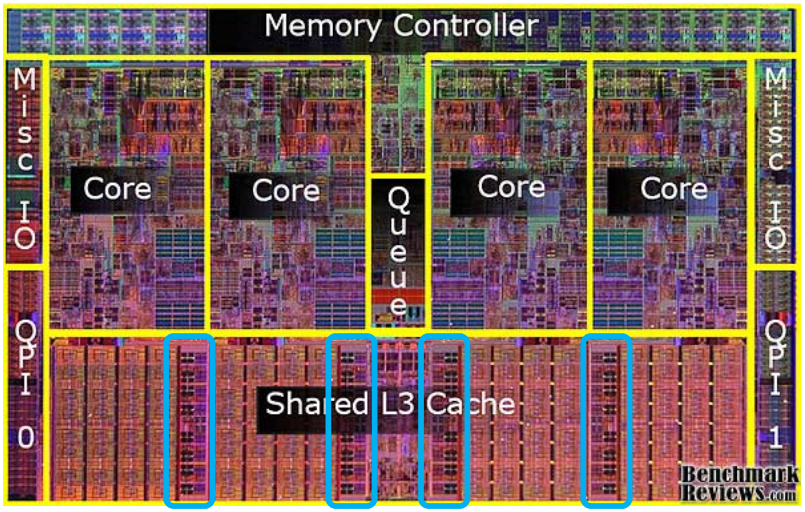
T_{ave} without L2 cache

- 2 possible cases:
  - Either we have a hit and pay only the L1 cache hit time
  - Or we have a miss and read in the whole block to L1 and then read from L1 to the processor
- $T_{ave} = T_1 + \underbrace{(1-H_1)\bullet[T_{MM} + B\bullet R_{MM}]}_{\text{(Miss Rate)*(Miss Penalty)}}$
- For T_1=10ns, H_1 = 0.9, B=8, T_{MM}=100ns, R_{MM}=25ns
  - T_{ave} = 10 + [(0.1) • (100+8•25)] = 40 ns

T_{ave} with L2 cache

- 3 possible cases:
  - Either we have a hit and pay the L1 cache hit time
  - Or we miss L1 but hit L2 and read in the block from L2
  - Or we miss L1 and L2 and read in the block from MM
- T_{ave} = _____
- For T_1 = 10ns, H_1 = 0.9, T_2 = 20ns, R_2 = 10ns, H_2 = 0.98, B=8, T_{MM}=100ns, R_{MM}=25 ns
- T_{ave} =

Intel Nehalem Quad Core



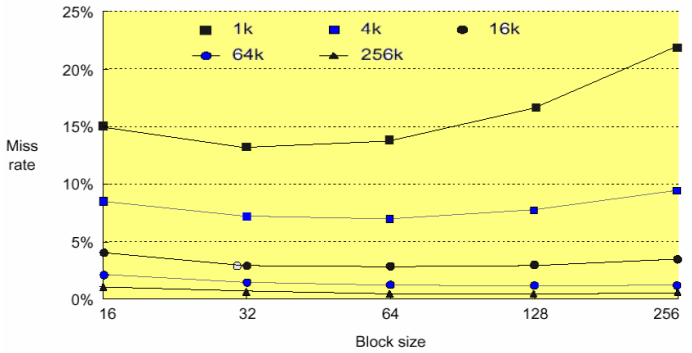
UNDERSTANDING MISSES

Miss Rate

- Reducing Miss Rate means lower T_{AVE}
- To analyze miss rate categorize them based on why they occur
  - _____ Misses
    - _____ to a block will always result in a miss
  - _____ Misses
    - Misses because the cache is _____
  - _____ Misses
    - Misses due to _____ (replacement of direct or set associative)

Miss Rate & Block Size

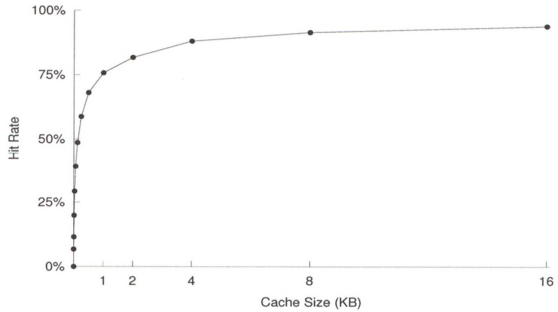
- Block size too small: Not getting _____ to next higher level
- Block size too large: Time is spent getting data you _____ and that data occupies space in the cache that prevents other useful data from being present



Graph used courtesy "Computer Architecture: AQA, 3rd ed.", Hennessey and Patterson

Hit/Miss Rate vs. Cache Size

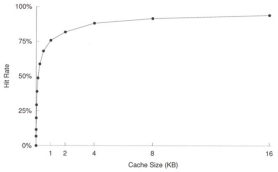
- Capacity is important up to a point
  - Only the data the program is currently working with (aka its "working set") need fit in the cache

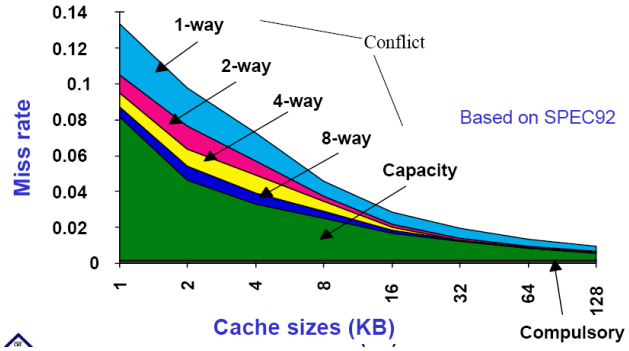


OS:PP 2nd Ed.: Fig. 9.4

Miss Rate & Associativity

- At reasonable cache sizes, associativity above _____-way does not yield much improvement





Graph used courtesy "Computer Architecture: AQA, 3rd ed.", Hennessey and Patterson

Prefetching

- Hardware Prefetching
  - On miss of block i, fetch block _____
- Software Prefetching
  - Special "_____" Instructions
  - Compiler inserts these instructions to give hints ahead of time as to the upcoming access pattern

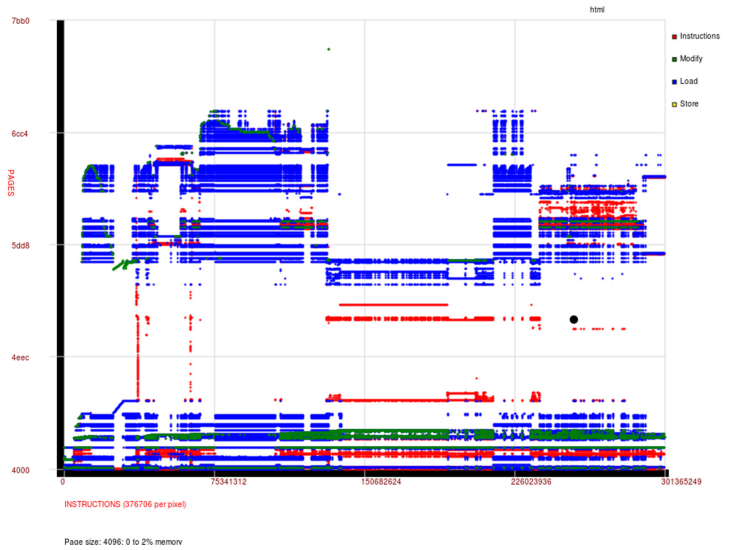
CACHE CONSCIOUS PROGRAMMING

What Makes a Cache Work

- What are the necessary conditions
  - Locations used to store cached data must be faster to access than original locations
  - Some reasonable amount of _____
  - Access patterns must be somewhat _____

Working Sets

- Generally a program works with different sets of data at different times
  - Consider an image processing algorithm akin to JPEG encoding
    - Perform data transformation on image pixels using several weighting tables/arrays
    - Create a table of frequencies
    - Perform compression coding using that table of frequencies
    - Replace pixels with compressed codes
- The data that the program is accessing in a small time window is referred to as its working set
- We want that working set to fit in cache and make as much reuse of that working set as possible while it is in cache
  - Example of performing JPG compression:
    - Keep weight tables in cache when performing data transformation
    - Keep frequency table in cache when compressing



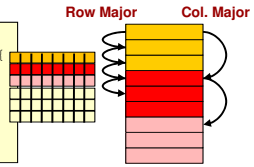
https://cartesianproduct.wordpress.com/tag/working-set/

Cache-Conscious Programming

- Order of array indexing
  - Row major vs. column major ordering
- Blocking (keeps working set small)
- Pointer-chasing
  - Linked lists, graphs, tree data structures that use pointers do not exhibit good spatial locality
- General Principles
  - Keep working set reasonably ____ (temporal locality)
  - Use small _____ (spatial locality)
  - Static structures usually better than dynamic ones

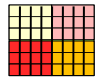
```
for(i=0; i<SIZE; i++) {
  for(j=0; j<SIZE; j++) {
    // Row-major
    A[i][j] = A[i][j]*2;
    // Column-major
    A[j][i] = A[j][i]*2;
} }
```

Example of row vs. column major ordering



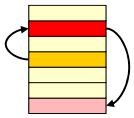
Memory Layout of matrix A

Original Matrix



Blocked Matrix

Linked Lists

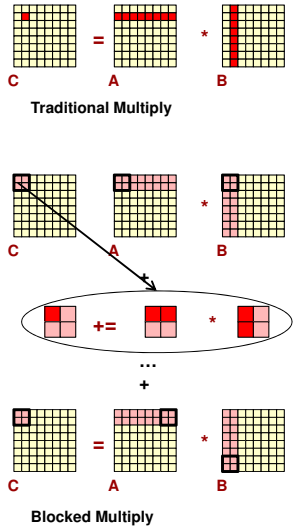


Memory Layout of Linked List

Blocked Matrix Multiply

- Traditional working set
  - 1 row of C, 1 row of A, NxN matrix B
- Break NxN matrix into smaller BxB matrices
  - Perform matrix multiply on blocks
  - Sum results of block multiplies to produce overall multiply result
- Blocked multiply working set
  - Three BxB matrices

```
for(i = 0; i < N; i+=B) {
 for(j = 0; j < N; j+=B) {
  for(k = 0; k < N; k+=B) {
   for(ii = i; ii < i+B; ii++) {
    for(jj = j; jj < j+B; jj++) {
     for(kk = k; kk < k+B; kk++) {
       Cb[ii][jj] += Ab[ii][kk] * Bb[kk][jj];
} } } } } }
```



Blocked Multiply Results

- Intel Nehalem processor
  - L1D = 32 KB, L2 = 256KB, L3 = 8 MB