5118007-02 Computer Architecture

Ch. 5 Large and Fast
: Exploiting Memory Hierarchy

4 June 2024

Shin Hong

Memory System Performance

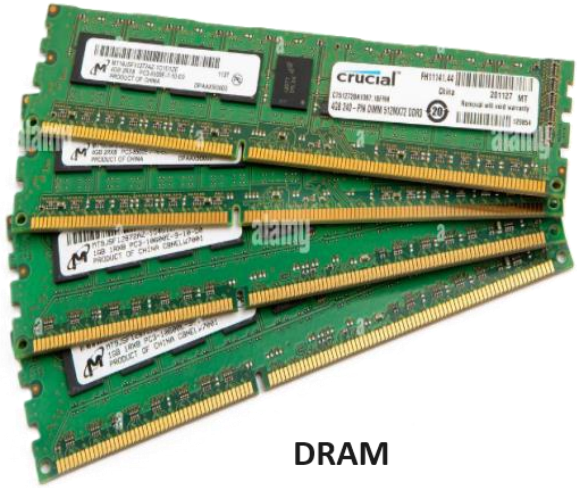
- Memory performance is crucial for system performance because most instructions involve memory accesses

- Challenge: tradeoff of memory capacity and memory speed

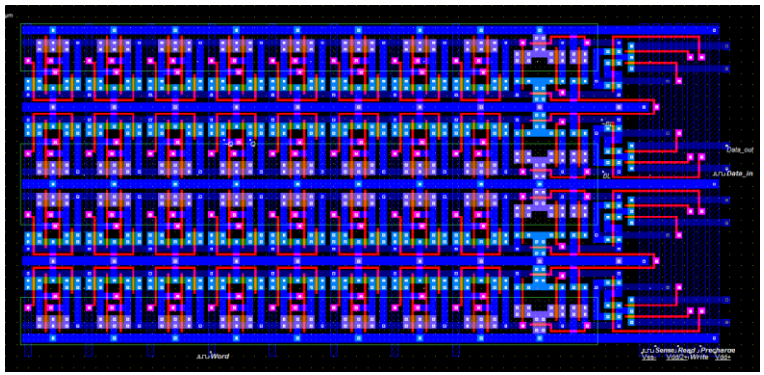
Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

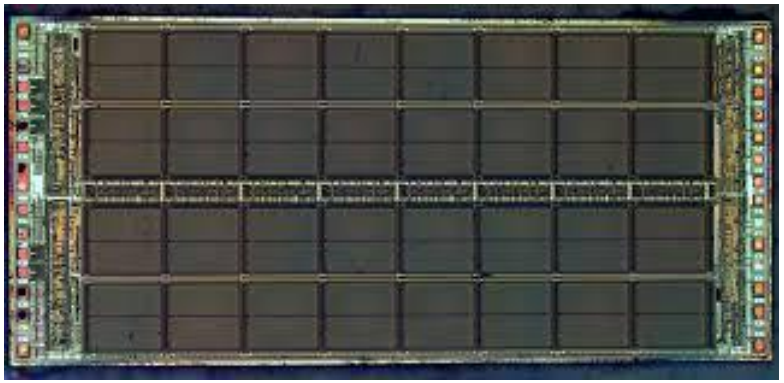
SRAM vs. DRAM

SRAM

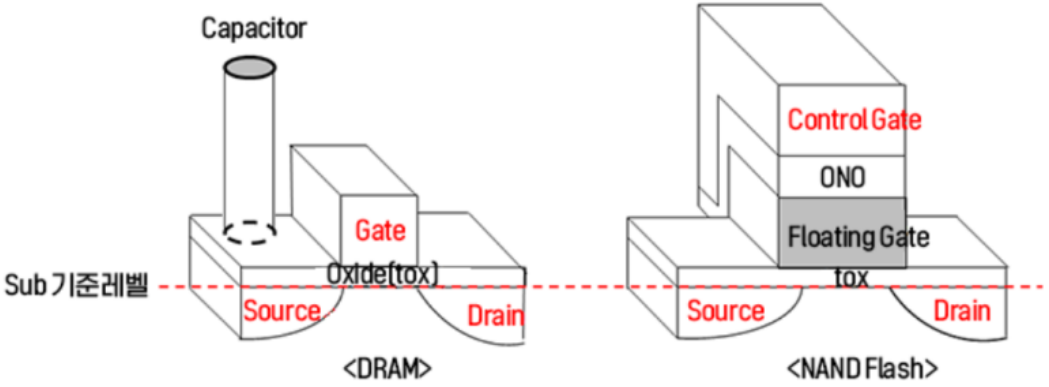


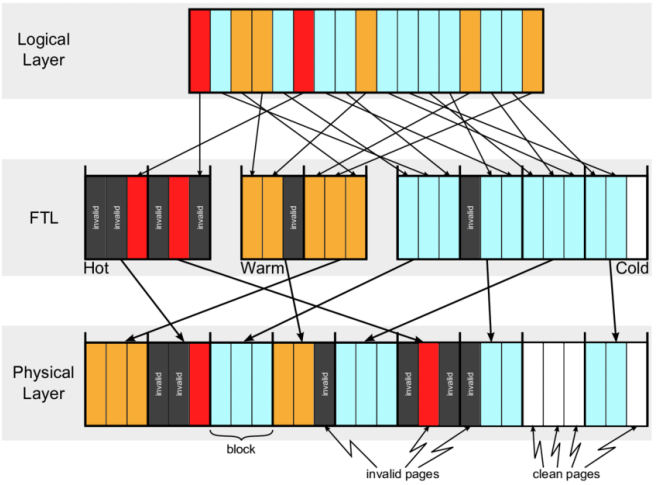
DRAM





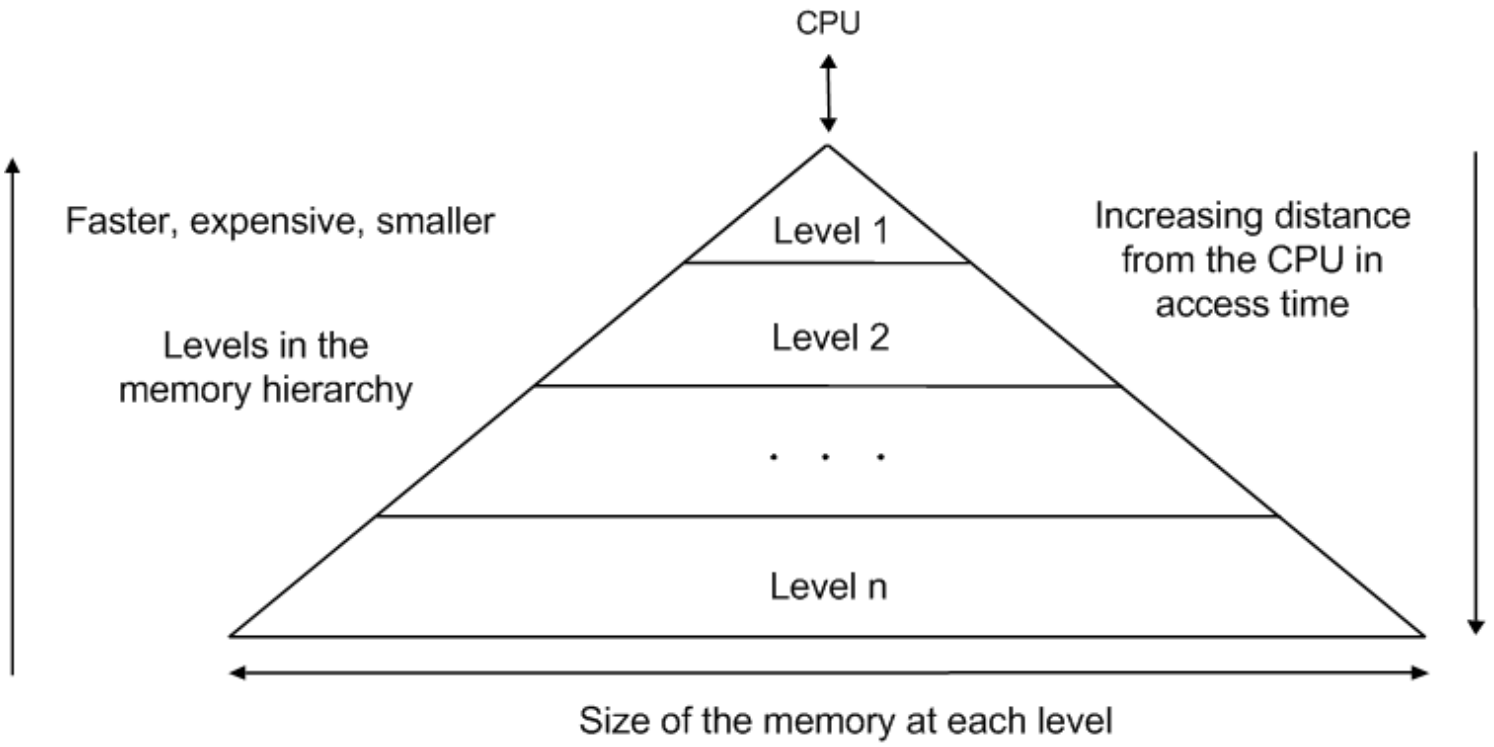
DRAM vs. Flash Memory





Memory Hierarchy

- Have multi-level images (caches) of the same data to exploit the locality in memory access patterns

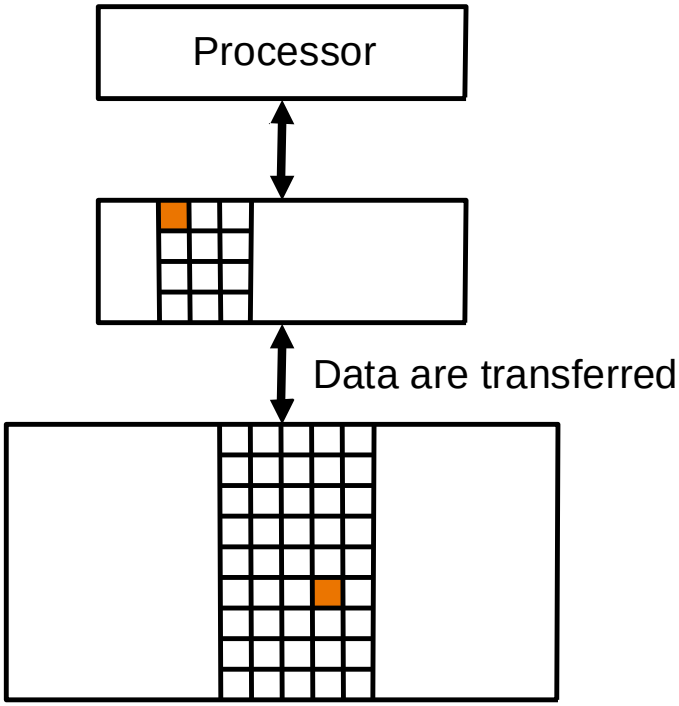


Locality

- Memory access pattern is mostly predictable
- Temporal locality: the same item will tend to be referenced again soon
- Spatial locality: nearby items will tend to be referenced soon
- Memory hierarchy
  - CPU is accessing small, high-speed memory device only
  - Only when cache miss happens, the memory is copied from a lower-level to a higher-level memory device
    - assume that cache miss happens very infrequently

Basic Structure of Memory Hierarchy

- block: data unit
  - one or multiple words
- hit: data requested is presented at the upper level
- miss: data requested is not presented at the upper level
- each pair of levels in the memory hierarchy can be thought of as having an upper and lower level



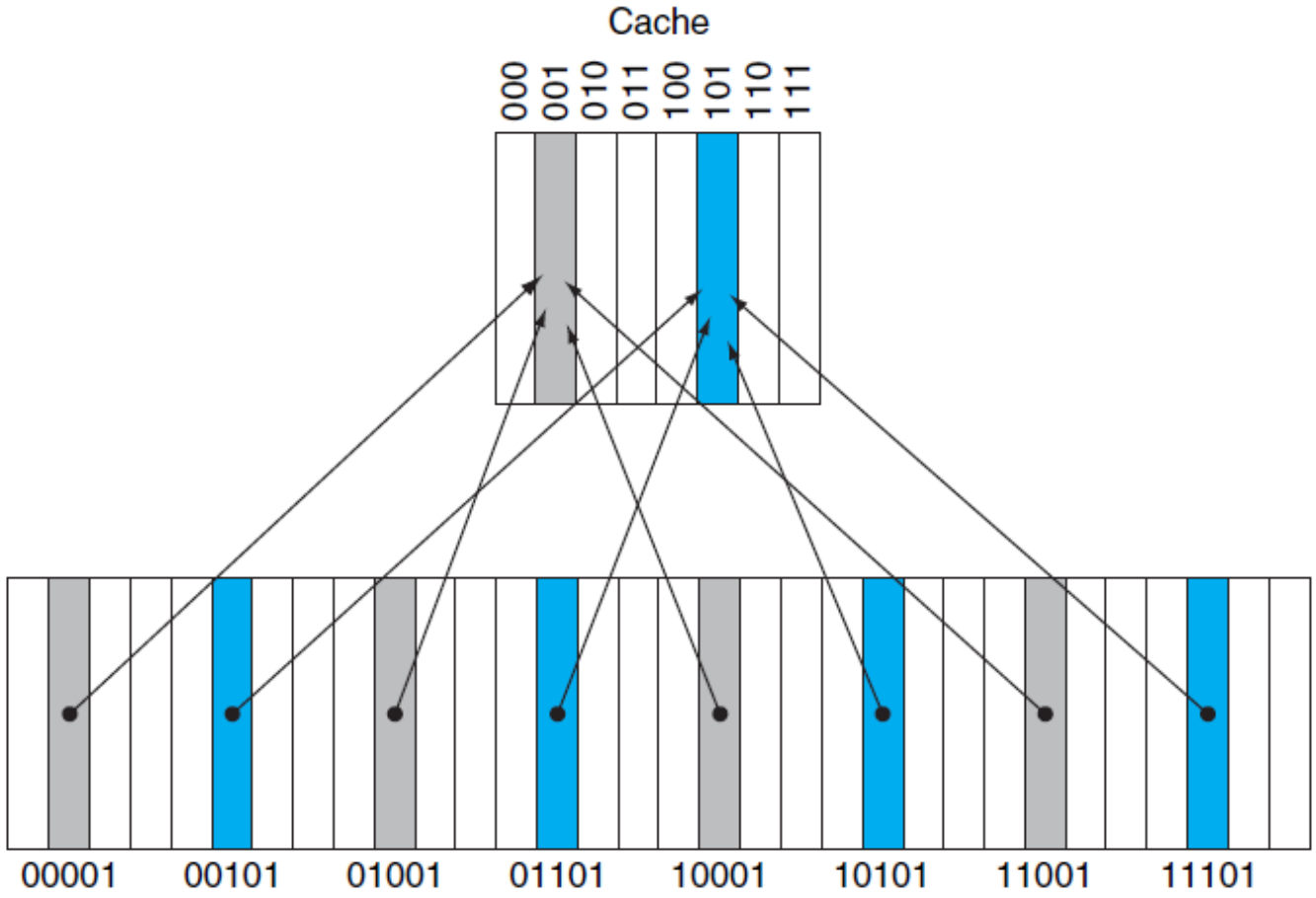
Cache

- A copy of a fraction of data in fast access memory
- Issues
  - how to know the requested data is in cache
  - how to where the data is stored
  - how to update the cached data
  - which data is to be resided in cache

Direct-mapped Cache

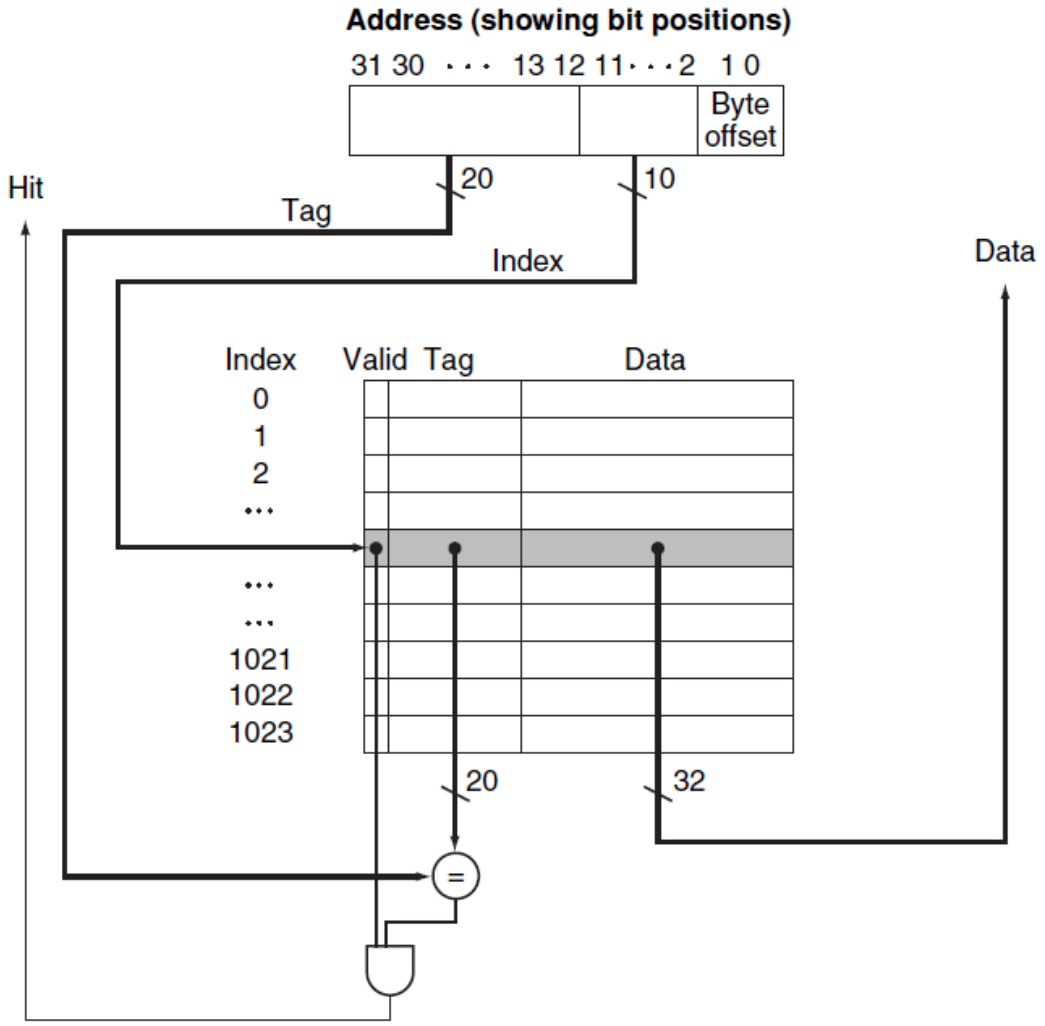
- The processor requests one word at a time, and the cache blocks consist of a single word
- The cache contains a collection of recent references
- The processor requests a word to the memory, if it is not in the cache
- For a given memory access request, the corresponding location in the cache is identified by the memory address
  - Cache location = Address % NumBlocks

Example


Cache
000
001
010
011
100
101
110
111
00001
00101
01001
01101
10001
10101
11001
11101

Cache Entry

- **tag** : to complete the target memory address
- **valid** : a flag bit to indicate whether the entry is used or not
- **data** (cache block) : the copy of the corresponding memory value



Cache Access Example

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.6f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

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

Cache Access Example

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.6f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Cache Access Example

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	miss (5.6c)	(11010_{two} mod 8) = 010_{two}
22	10110_{two}	hit	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	hit	(11010_{two} mod 8) = 010_{two}
16	10000_{two}	miss (5.6d)	(10000_{two} mod 8) = 000_{two}
3	00011_{two}	miss (5.6e)	(00011_{two} mod 8) = 011_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}
18	10010_{two}	miss (5.6f)	(10010_{two} mod 8) = 010_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Cache Access Example

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	miss (5.6c)	(11010_{two} mod 8) = 010_{two}
22	10110_{two}	hit	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	hit	(11010_{two} mod 8) = 010_{two}
16	10000_{two}	miss (5.6d)	(10000_{two} mod 8) = 000_{two}
3	00011_{two}	miss (5.6e)	(00011_{two} mod 8) = 011_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}
18	10010_{two}	miss (5.6f)	(10010_{two} mod 8) = 010_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Cache Access Example

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.6f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Cache Access Example

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	miss (5.6c)	(11010_{two} mod 8) = 010_{two}
22	10110_{two}	hit	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	hit	(11010_{two} mod 8) = 010_{two}
16	10000_{two}	miss (5.6d)	(10000_{two} mod 8) = 000_{two}
3	00011_{two}	miss (5.6e)	(00011_{two} mod 8) = 011_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}
18	10010_{two}	miss (5.6f)	(10010_{two} mod 8) = 010_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Cache Memory Size

- 32-bit address with a direct-mapped cache
- *n* low-order bits are used for indexing cache blocks
  - 2^n cache blocks
- each block consists of 2^m words (2^{m+2} bytes)
  - the cache size (data capacity) is $2^{(n+m+2)}$ bytes
- foreach cache entry, the tag contains (32-*n*-*m*-*2*)-bits and there must be 1 valid bit
  - the number of bits in this direct-mapped cache is

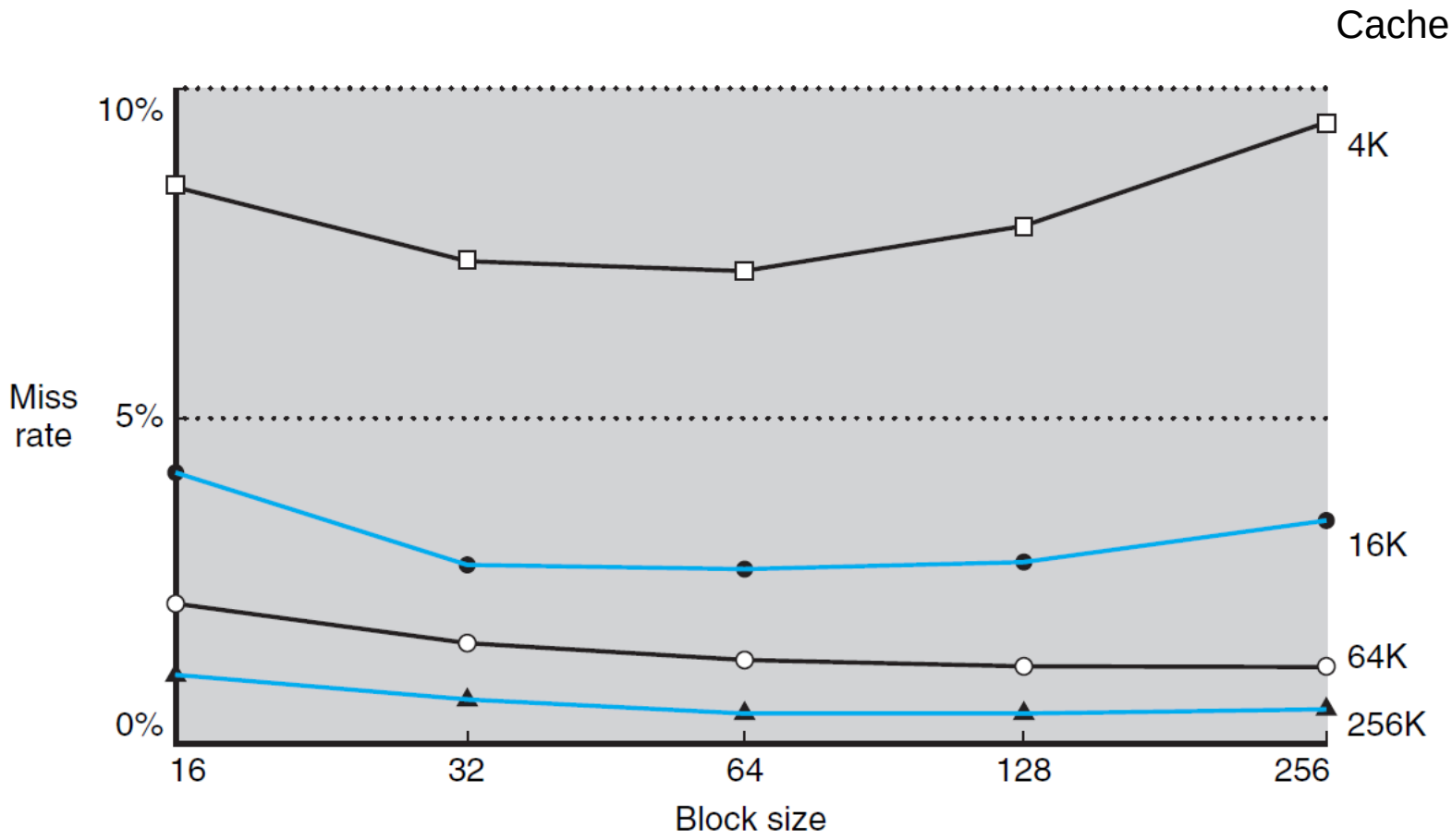
    $2^n \times (2^{m+5} + 32\text{-}n\text{-}m\text{-}1)$ bits

Example

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

- Block addr = Mem addr / Block size
- Cache loc = Block addr % Num Blocks

Miss Rate at Varying Block/Cache Size



Cache Miss

- Read hit
- Read miss
  - Stall the CPU, freezing the contents of all the registers while waiting for memory
  - A separate controller handles the cache miss, fetching the data into cache from memory
  - Once the data is present, Restart the execution

Read Miss at Instruction Cache

- Send PC-4 to memory
- Read block from memory, CPU stalls.
- Write the cache entry (data + tag) and set valid bit on
- Restart the instruction execution
  - fetch with the same memory address again

Read Miss at Data Cache

- Read block from memory, CPU stalls.
- Write the cache entry (data + tag) and set valid bit on
- Restart the instruction execution
  - fetch with the same memory address again

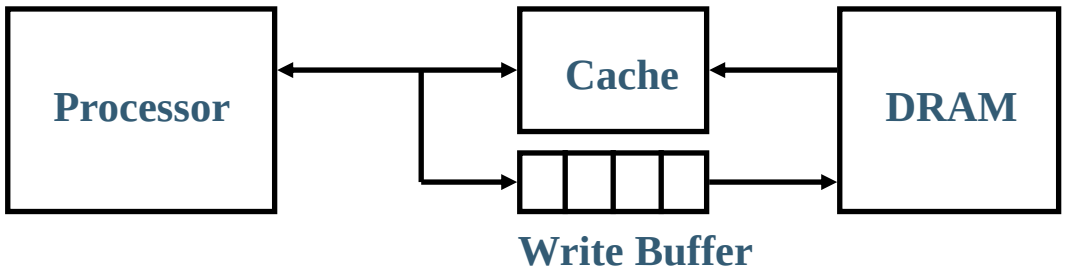
Cache with Store Instruction

- Write-through
  - The information is written to the cache and the main memory
  - the cache and the main memory are always consistent

- Write-back
  - The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - Dirty bit: Is block clean or dirty?
    - Dirty: Cache block was updated after it is loaded into cache. The old block should be written into memory when it is replaced.
    - Clean : Cache block was not updated after it is loaded into cache.

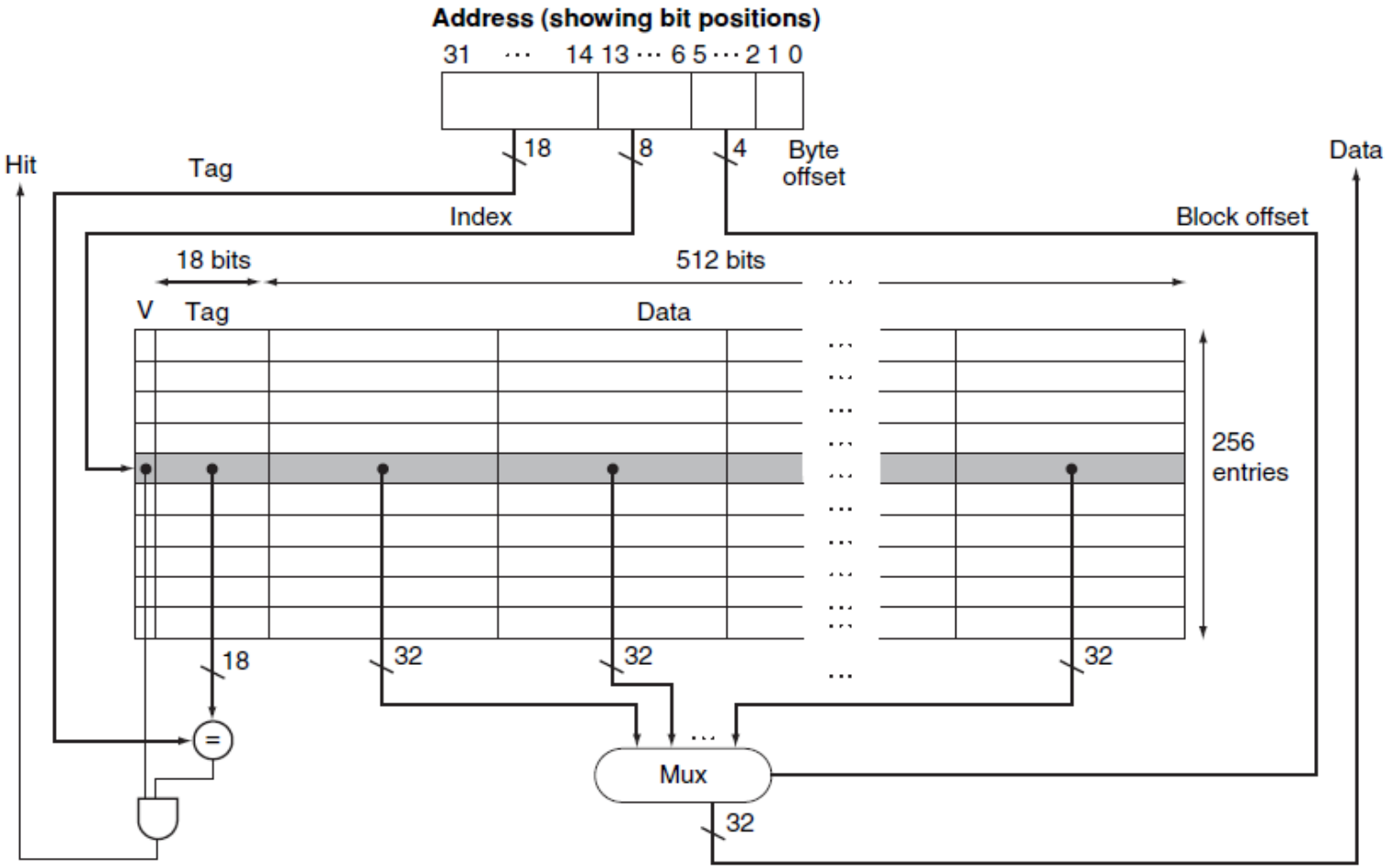
Write-through vs. Write-back

- Pros (and Cons) of each?
  - W-T: Read misses does not result in writes to memory
  - W-B: No repeated writes to memory
- A Write Buffer is located between the Cache and Memory
  - W-T is always combined with write buffers
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory



Ex. Intrinsity FastMATH

- MIPS32 architecture with two 16KB caches
  - instruction and data caches
  - each cache has 256 blocks each of which stores 16 words



Ex. Intrinsity FastMATH (Con'd)

Instruction miss rate	Data miss rate	Effective combined miss rate
0.4%	11.4%	3.2%

FIGURE 5.13 Approximate instruction and data miss rates for the Intrinsity FastMATH processor for SPEC CPU2000 benchmarks. The combined miss rate is the effective miss rate seen for the combination of the 16 KiB instruction cache and 16 KiB data cache. It is obtained by weighting the instruction and data individual miss rates by the frequency of instruction and data references.

- Total cache size: 32 KiB
- Split cache effective miss rate: 3.24%
- Combined cache miss rate: 3.18%

Cache Performance

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

$$\text{Memory-stall clock cycles} = (\text{Read-stall cycles} + \text{Write-stall cycles})$$

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

$$\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} \right) + \text{Write buffer stalls}$$

Cache Performance

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Example

Assume the miss rate of an instruction cache is 2% and the miss rate of the data cache is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.

- CPI = Clock cycle per instruction

$$\text{Instruction miss cycles} = I \times 2\% \times 100 = 2.00 \times I$$

$$\text{Data miss cycles} = I \times 36\% \times 4\% \times 100 = 1.44 \times I$$

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{I \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}} = \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2}$$

Average Memory Access Time (AMAT)

- AMAT = Time for a hit + Miss rate X Miss penalty
- Example

Find the AMAT for a processor with a 1 ns clock cycle time, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls.

The average memory access time per instruction is

$$\begin{aligned} \text{AMAT} &= \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty} \\ &= 1 + 0.05 \times 20 \\ &= 2 \text{ clock cycles} \end{aligned}$$

or 2 ns.

How to Reduce Cache Miss

- Fully-associative cache
  - Let a block can be any cache entry
  - To find a given block, linear search is required
    - make the search in parallel for rapid response

- Set-associative cache
  - Let a block can be placed at a few number of blocks (not one, not all)
  - *n*-way set-associative cache

Examples

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				
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

Example

Assume there are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set-associative, and the third is direct-mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, and 8.

Direct-mapped Cache

- Block access: 0 8 0 6 8

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

2-set-associative Cache

- Block access: 0 8 0 6 8

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

Fully-associative Cache

- Block access: 0 8 0 6 8

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

Block Replacement Policy

- Least Recently Used (LRU): replace a block that has not been used for the longest time with a new block

- Track when each element was used relative to the other
  - For 2-way set associate cache, keep a single bit to which one of the two was more recently used
  - However, for N-way set associate cache, tracking incurs overhead

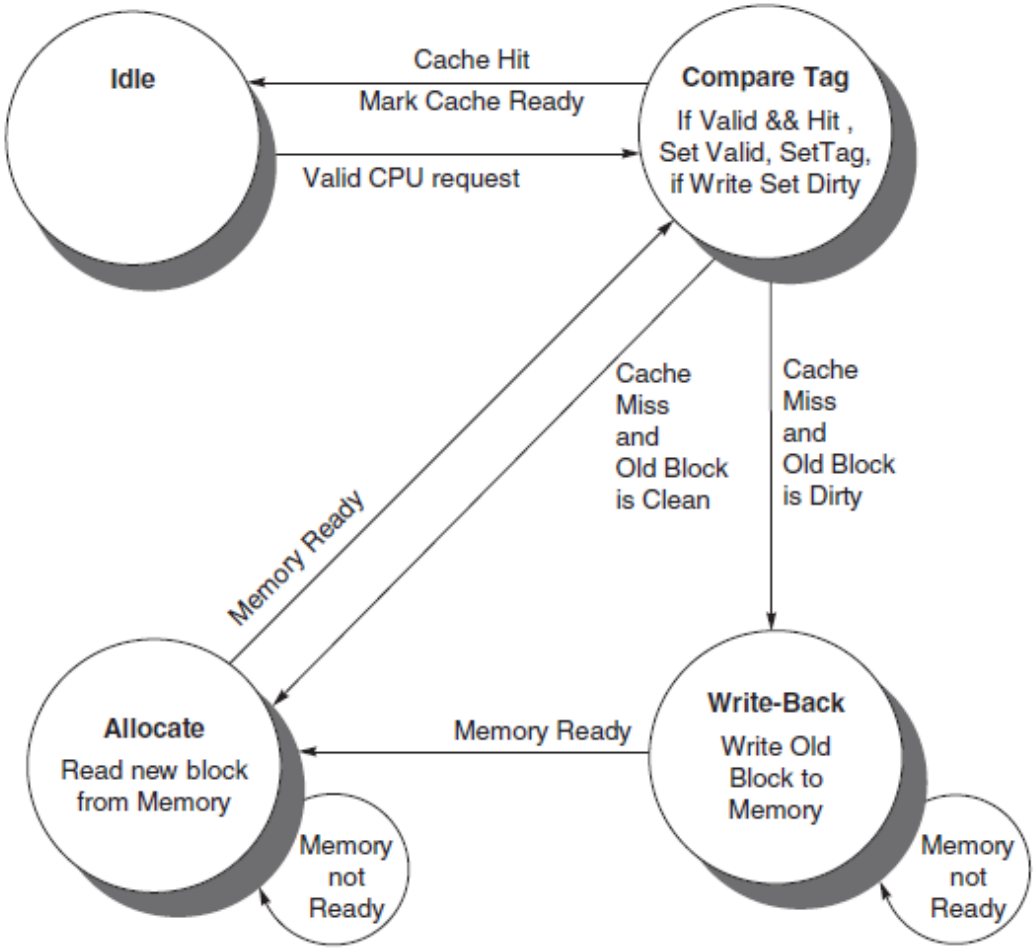
Simple Cache Controller

- Cache spec
  - 32-bit address
  - Direct-mapped cache
  - Write-back using write allocate
  - Block size is 4 words
  - Cache size is 16 KB holding 1024 blocks
  - A cache entry has 1-bit for valid bit and 1-bit for dirty bit

- Interface between processor and cache (cache and memory)
  - 1-bit Read or Write signal
  - 1-bit Valid signal, saying whether there is a cache operation or not
  - 1-bit Ready signal, saying the given cache operation is complete
  - 32-bit address
  - 32-bit data from processor to cache (128-bit data from cache to memory)
  - 32-bit data from cache to processor (128-bit data from memory to cache)

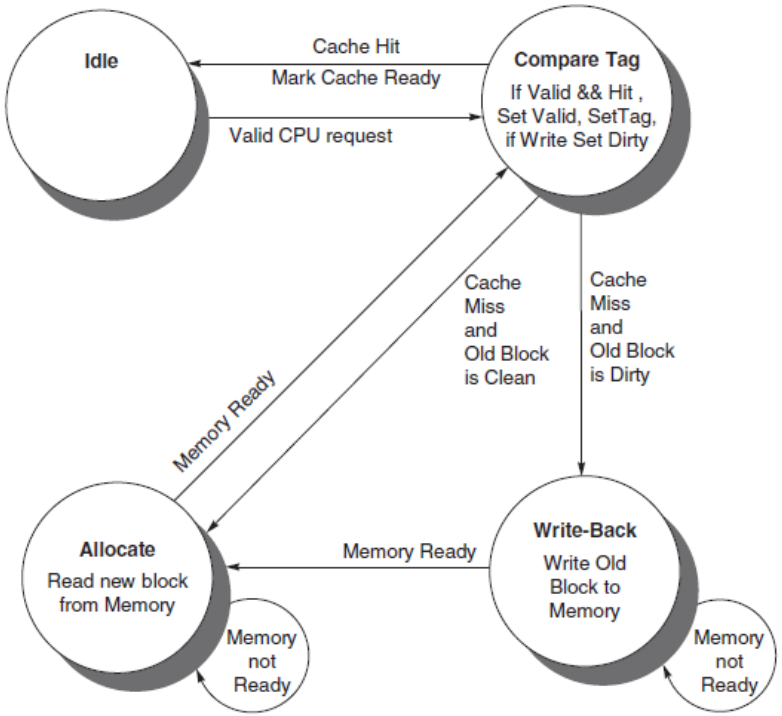
Finite State Machine Controller

- a form of Moore-machine



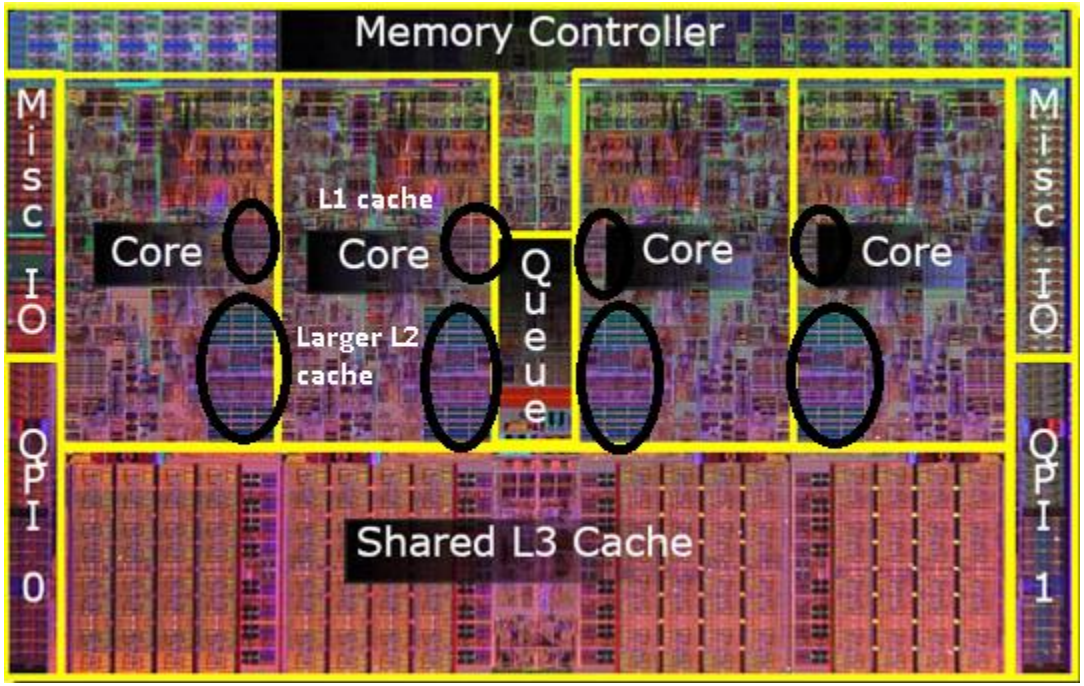
Finite State Machine Controller

- Idle state
  - wait for a Read or Write request from the processor
- Compare-Tag state
  - Check the valid bit at the target index
  - Retrieve the tag at the target index
  - Retrieve the data
- Write-back
  - Send the block at the index to memory
  - Wait for Memory Ready signal
- Allocate
  - Fetch new block from memory
  - Wait for Memory Ready signal



Multi-level Cache

- To further close the gap between the clock rates of processor and DRAM, modern processors support second-level cache (secondary cache)
  - access time of second-level cache is much less than that of main memory



https://superuser.com/questions/196143/where-exactly-l1-l2-and-l3-caches-located-in-computer

Multi-level Cache – Example

- Suppose that there is an architecture of the following properties
  - a base CPI: 1.0
  - a processor clock rate: 4 GHz,
  - a main memory access time: 100 ns
  - the miss rate per instruction at the primary cache: 2.0%
- How much faster will the processor be if a second-level cache with 5 ns access time, which reduces the miss rate to main memory to 0.5%?
  - 4 GHz = 4 x 10^9 cycles / sec, thus one cycle takes 0.25 X 10^{-9} sec = 0.25 ns
  - one main memory access takes 100 ns = 400 cycles
  - Originally, total CPI = Base CPI + Miss Penalty = 1.0 + 400 x 0.02 = 9.0 cycles
  - Secondary cache access takes 5 ns = 20 cycles
  - With secondary cache, Miss Penalty is 20 X 0.02 + 400 X 0.005 = (0.4 + 2.0) cycles, thus total CPI = 3.3 (=1.0 + 2.4) cycles

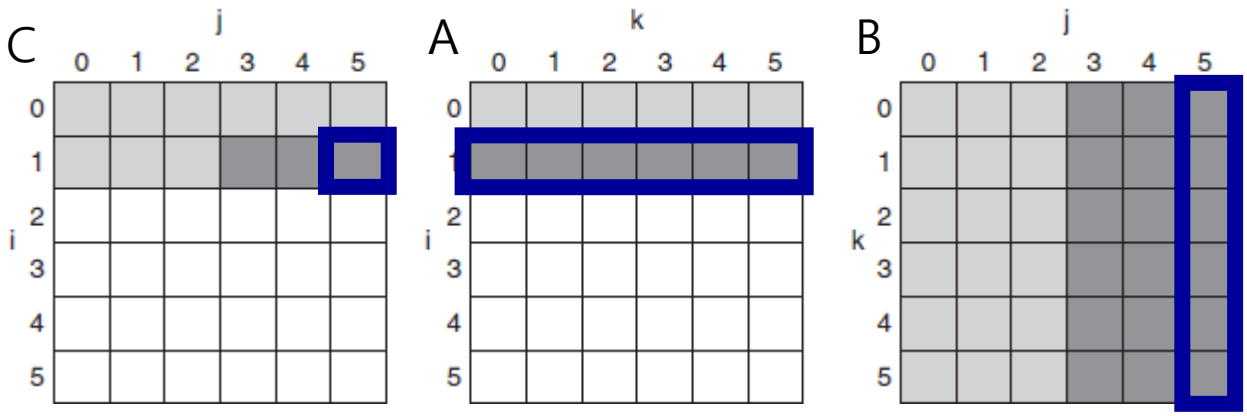
Software Optimization via Blocking

- Blocked algorithm
  - access data block-by-block, rather than on entire array
  - maximize access to the data loaded into the cache before replacement to improve temporal locality

Example. Matrix Multiplication

- conventional algorithm

```
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 */
     }
}
```



Example. Matrix Multiplication

- Blocked version

```
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 }
```