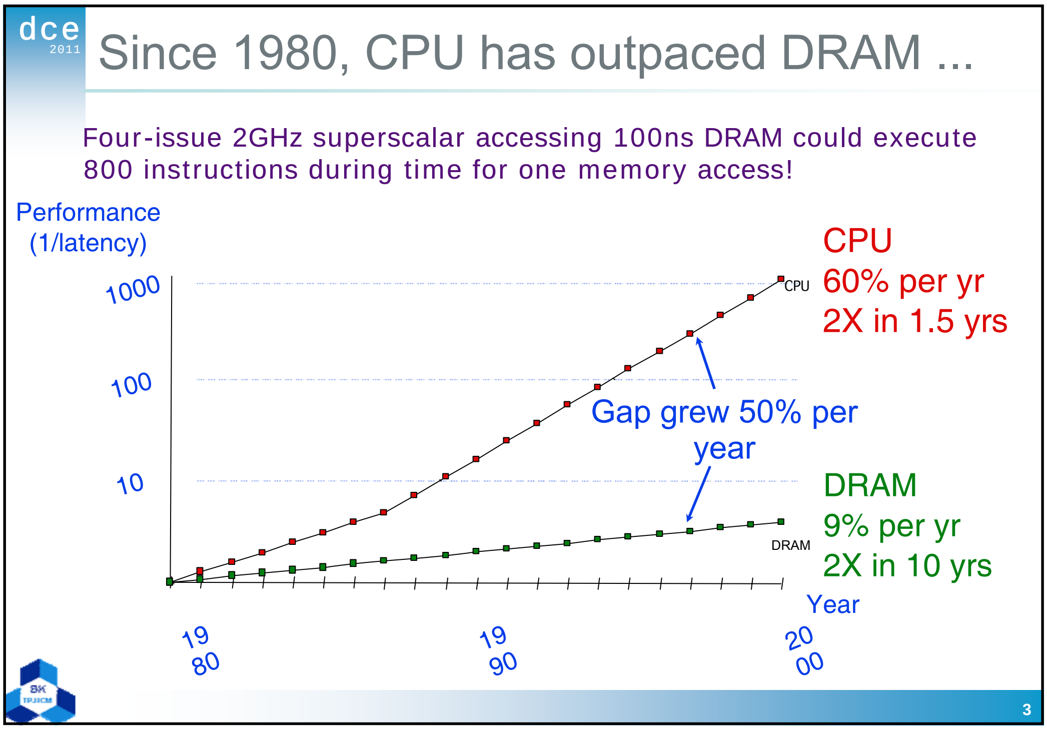
ADVANCED COMPUTER ARCHITECTURE

Khoa Khoa học và Kỹ thuật Máy tính
BM Kỹ thuật Máy tính

Trần Ngọc Thịnh
http://www.cse.hcmut.edu.vn/~tnthinh

Memory Hierarchy Design



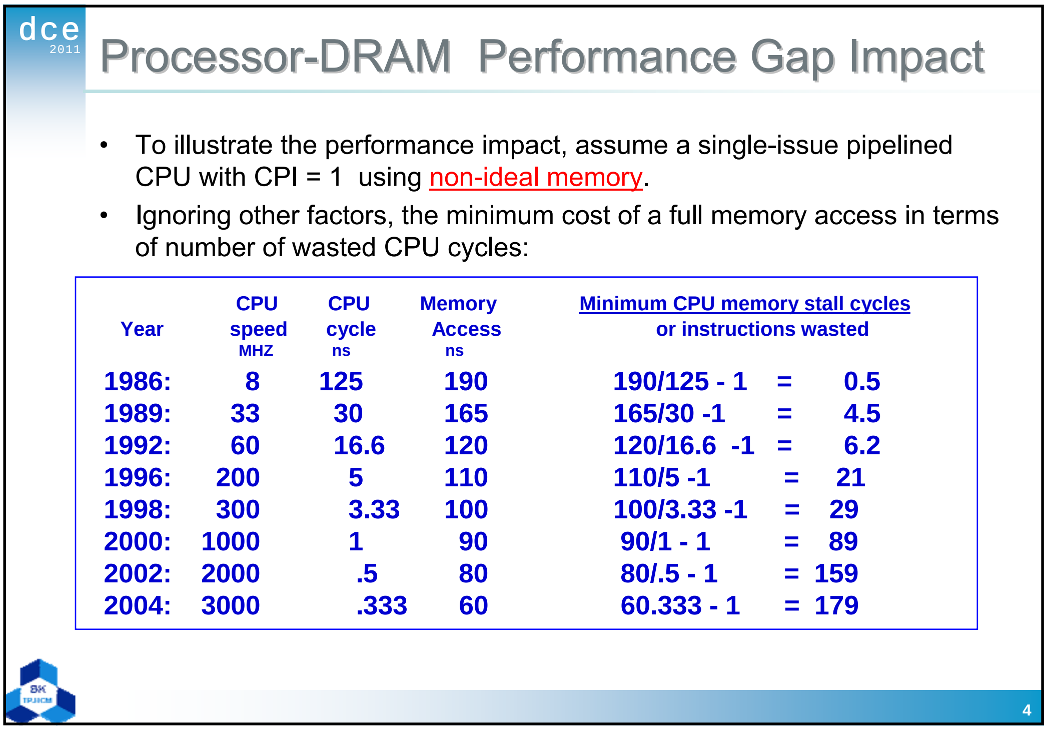
Since 1980, CPU has outpaced DRAM ...

Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

CPU
60% per yr
2X in 1.5 yrs

Gap grew 50% per year

DRAM
9% per yr
2X in 10 yrs



Processor-DRAM Performance Gap Impact

- To illustrate the performance impact, assume a single-issue pipelined CPU with CPI = 1 using non-ideal memory.
- Ignoring other factors, the minimum cost of a full memory access in terms of number of wasted CPU cycles:

Year	CPU speed MHZ	CPU cycle ns	Memory Access ns	Minimum CPU memory stall cycles or instructions wasted		
1986:	8	125	190	190/125 - 1	=	0.5
1989:	33	30	165	165/30 -1	=	4.5
1992:	60	16.6	120	120/16.6 -1	=	6.2
1996:	200	5	110	110/5 -1	=	21
1998:	300	3.33	100	100/3.33 -1	=	29
2000:	1000	1	90	90/1 - 1	=	89
2002:	2000	.5	80	80/.5 - 1	=	159
2004:	3000	.333	60	60.333 - 1	=	179

Levels of the Memory Hierarchy

Capacity
Access Time
Cost

CPU Registers
100s Bytes
<10s ns

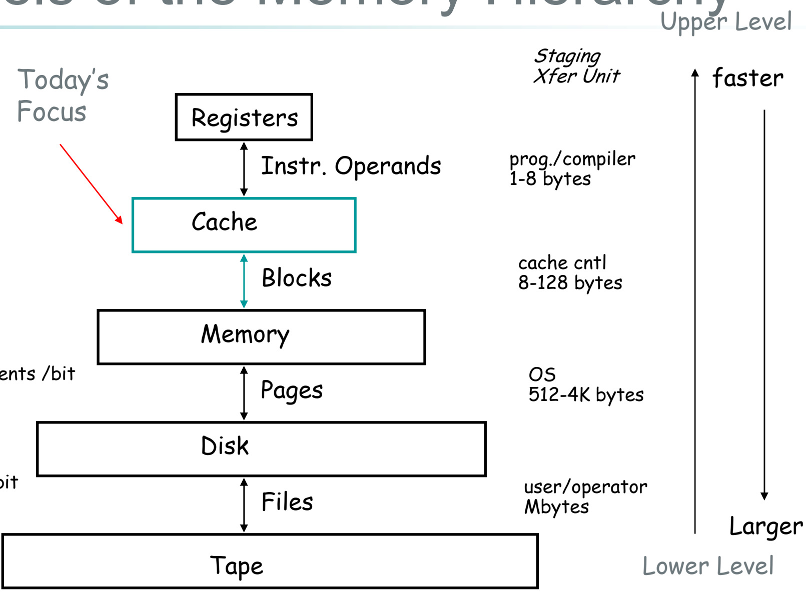
Cache
K Bytes
10-100 ns
1-0.1 cents/bit

Main Memory
M Bytes
200ns- 500ns
$.0001-.00001 cents /bit

Disk
G Bytes, 10 ms
(10,000,000 ns)

10^{-5} - 10^{-6} cents/bit

Tape
infinite
sec-min
10^{-8}



Addressing the Processor-Memory Performance GAP

- Goal: Illusion of large, fast, cheap memory. Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

- Solution: Put smaller, faster "cache" memories between CPU and DRAM. Create a "memory hierarchy".

Common Predictable Patterns

Two predictable properties of memory references:

- Temporal Locality: If a location is referenced, it is likely to be referenced again in the near future (e.g., loops, reuse).

- Spatial Locality: If a location is referenced it is likely that locations near it will be referenced in the near future (e.g., straightline code, array access).

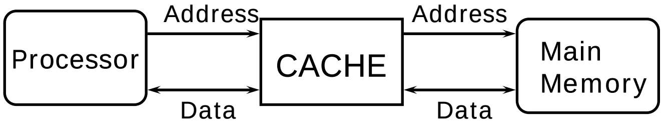
Caches

Caches exploit both types of predictability:

– Exploit temporal locality by remembering the contents of recently accessed locations.

– Exploit spatial locality by fetching blocks of data around recently accessed locations.

Simple view of cache



- The processor accesses the cache first
- Cache hit: Just use the data
- Cache miss: replace a block in cache by a block from main memory, use the data
- The data transferred between cache and main memory is in blocks, and controlled by independent hardware

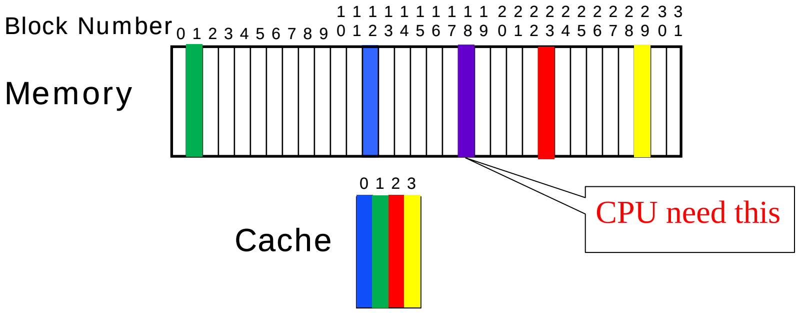
Simple view of cache

- Hit rate: fraction of cache hit
- Miss rate: 1 – Hit rate
- Miss penalty: Time to replace a block + time to deliver the data to the processor

Simple view of cache

- Example: For(i = 0; i < 10; i++) S = S + A[i];
- No cache: At least 12 accesses to main memory (10 A[i] and Read S, write S)
- With Cache: if A[i] and S is in a single block (ex 32-bytes), 1 access to load block to cache, and 1 access to write block to main memory
- Access to S: Temporal Locality
- Access to A[i]: Spatial Locality (A[i])

Replacement

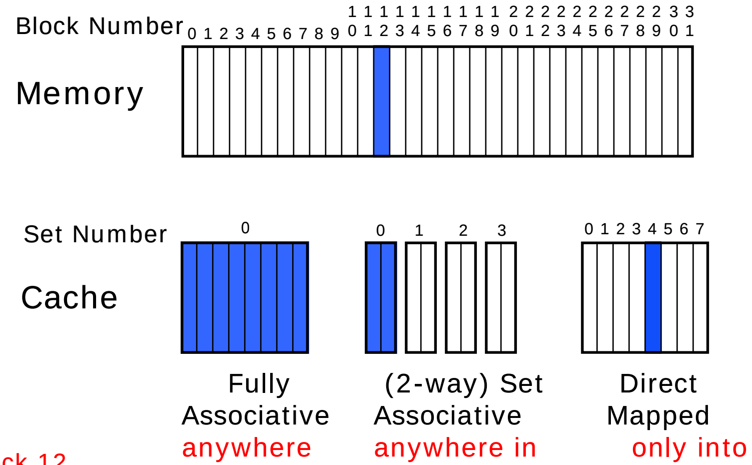


- Cache cannot hold all blocks
- Replace a block by another that is currently needed by CPU

Basic Cache Design & Operation Issues

- Q1: Where can a block be placed cache?
  (Block placement strategy & Cache organization)
  - Fully Associative, Set Associative, Direct Mapped.
- Q2: How is a block found if it is in cache?
  (Block identification)
  - Tag/Block.
- Q3: Which block should be replaced on a miss?
  (Block replacement)
  - Random, LRU, FIFO.
- Q4: What happens on a write?
  (Cache write policy)
  - Write through, write back.

Q1: Where can a block be placed?



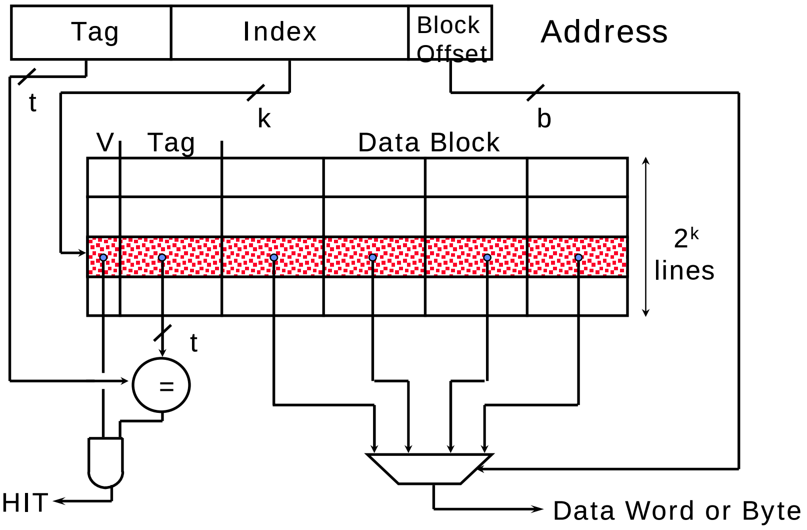
Fully Associative
anywhere

(2-way) Set Associative
anywhere in set 0
(12 mod 4)

Direct Mapped
only into block 4
(12 mod 8)

Block 12 can be placed

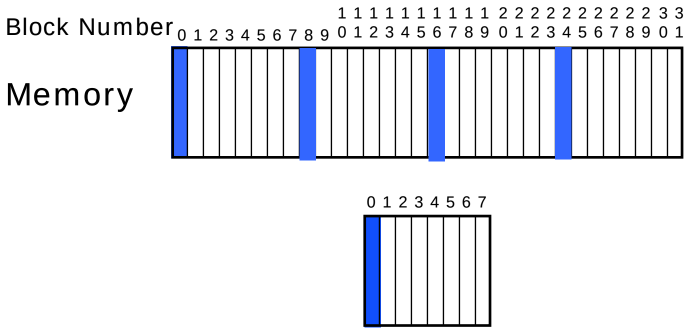
Direct-Mapped Cache



Direct-mapped Cache

- Address: N bits (2^N words)
- Cache has 2^k lines (blocks)
- Each line has 2^b words
- Block M is mapped to the line M % 2^k
- Need t = N-k-bTag bits to identify mem. block
- Advantage: Simple
- Disadvantage: High miss rate
- What if CPU accesses block N0, N1 and N0 % 2^k = N1 % 2^k ?

Direct-mapped Cache



- Access N0, N1 where N0 % 2^k = N1 % 2^k
- Replace a block while there are many rooms available!

4KB Direct Mapped Cache Example

1K = 1024 Blocks
Each block = one word

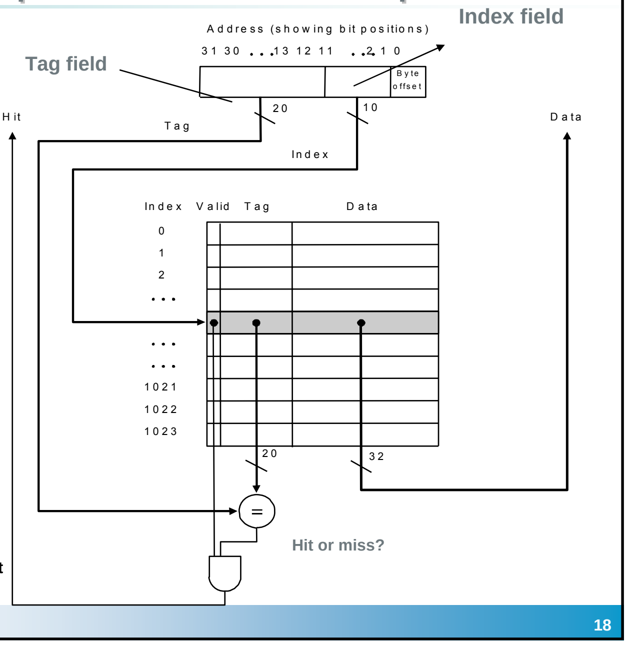
Can cache up to
2^{32} bytes = 4 GB
of memory

Mapping function:

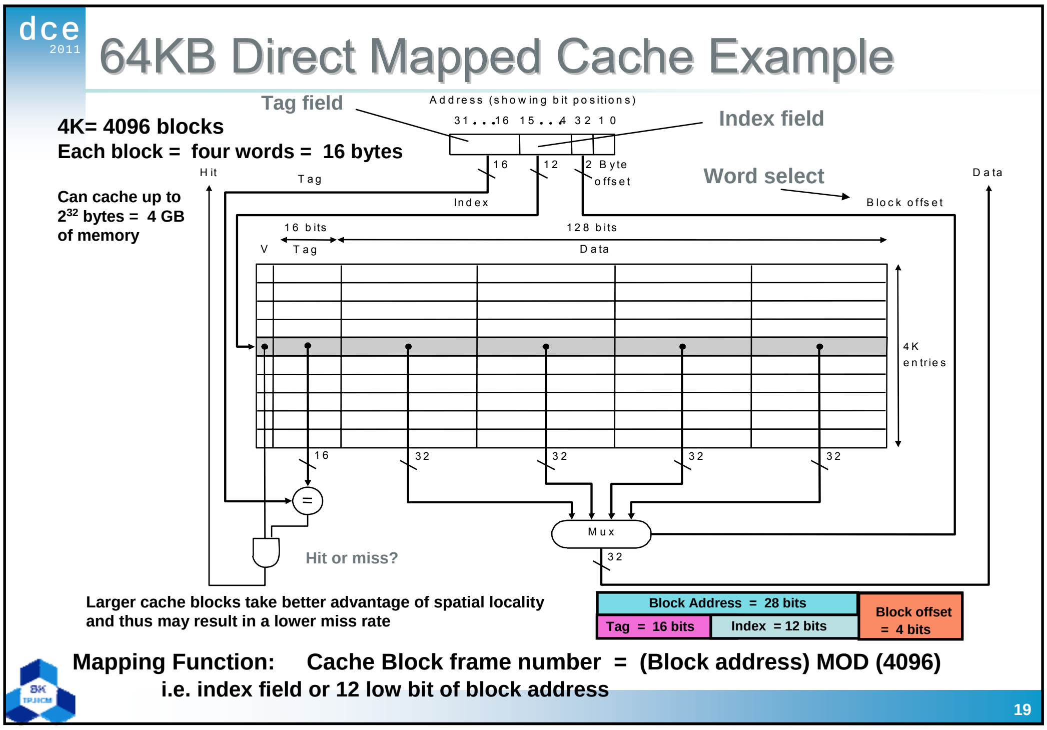
Cache Block frame number = (Block address) MOD (1024)

i.e. index field or
10 low bit of block address

Block Address = 30 bits		Block offset = 2 bits
Tag = 20 bits	Index = 10 bits	



64KB Direct Mapped Cache Example



4K= 4096 blocks

Each block = four words = 16 bytes

Can cache up to 2^{32} bytes = 4 GB of memory

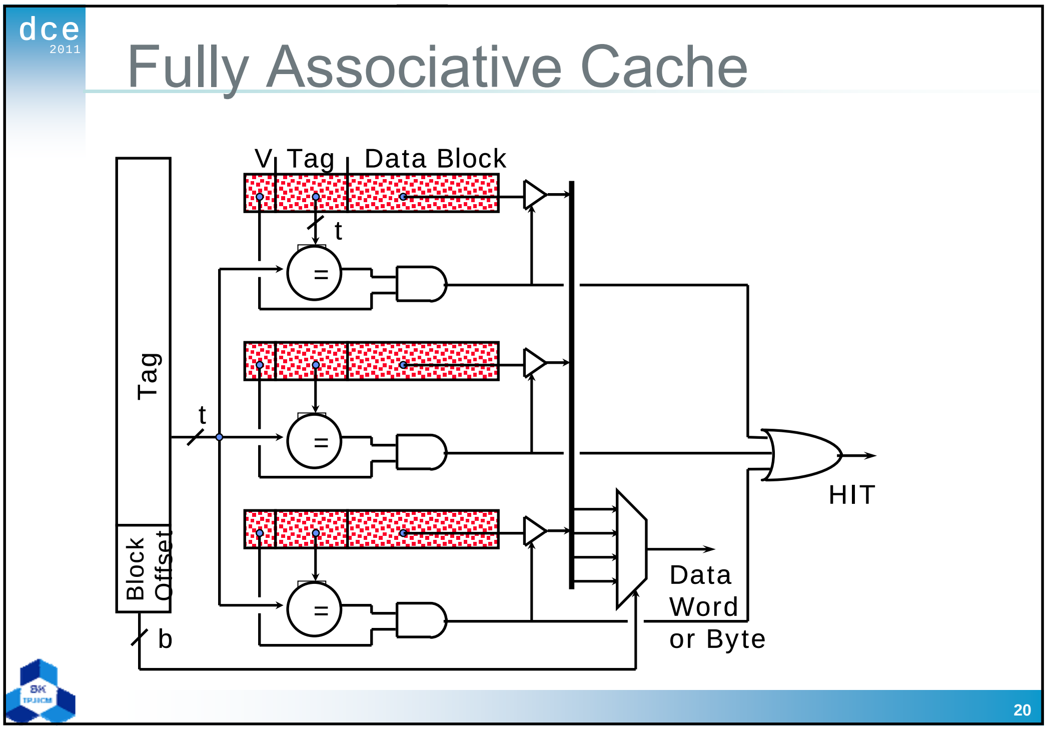
Larger cache blocks take better advantage of spatial locality and thus may result in a lower miss rate

Block Address = 28 bits		Block offset = 4 bits
Tag = 16 bits	Index = 12 bits	

Mapping Function: Cache Block frame number = (Block address) MOD (4096)

i.e. index field or 12 low bit of block address

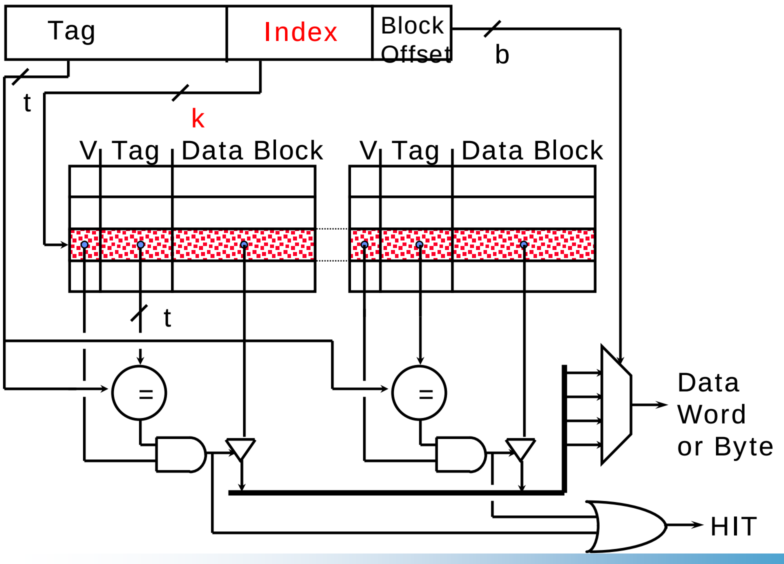
Fully Associative Cache



Fully associative cache

- CAM: Content Addressable Memory
- Each block can be mapped to any lines in cache
- Tag bit: t = N-b. Compared to Tag of all lines
- Advantage: replacement occurs only when no rooms available
- Disadvantage: resource consumption, delay by comparing many elements

Set-Associative Cache



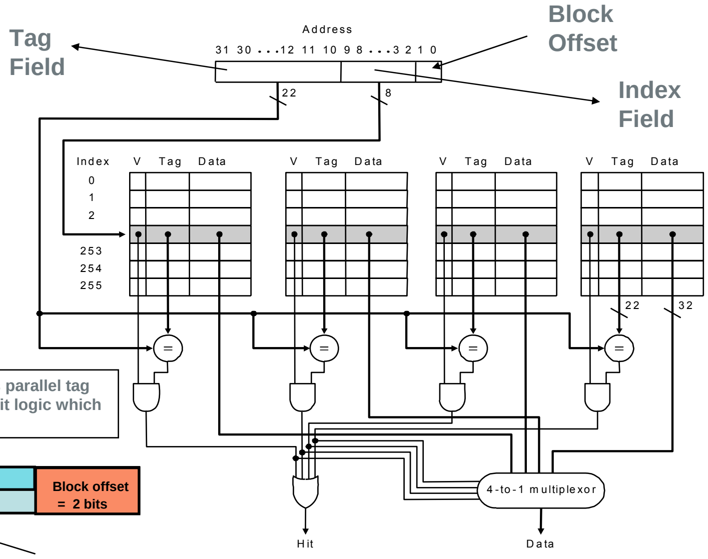
W-way Set-associative Cache

- Balancing: Direct mapped cache vs Fully associative cache
- Cache has 2^k sets
- Each set has 2^w lines
- Block M is mapped to one of 2^w lines in set M % 2^k
- Tag bit: t = N-k-b
- Currently: widely used (Intel, AMD, …)

4K Four-Way Set Associative Cache: MIPS Implementation Example

1024 block frames
Each block = one word
4-way set associative
1024 / 4= 256 sets

Can cache up to
2^{32} bytes = 4 GB
of memory



Tag Field

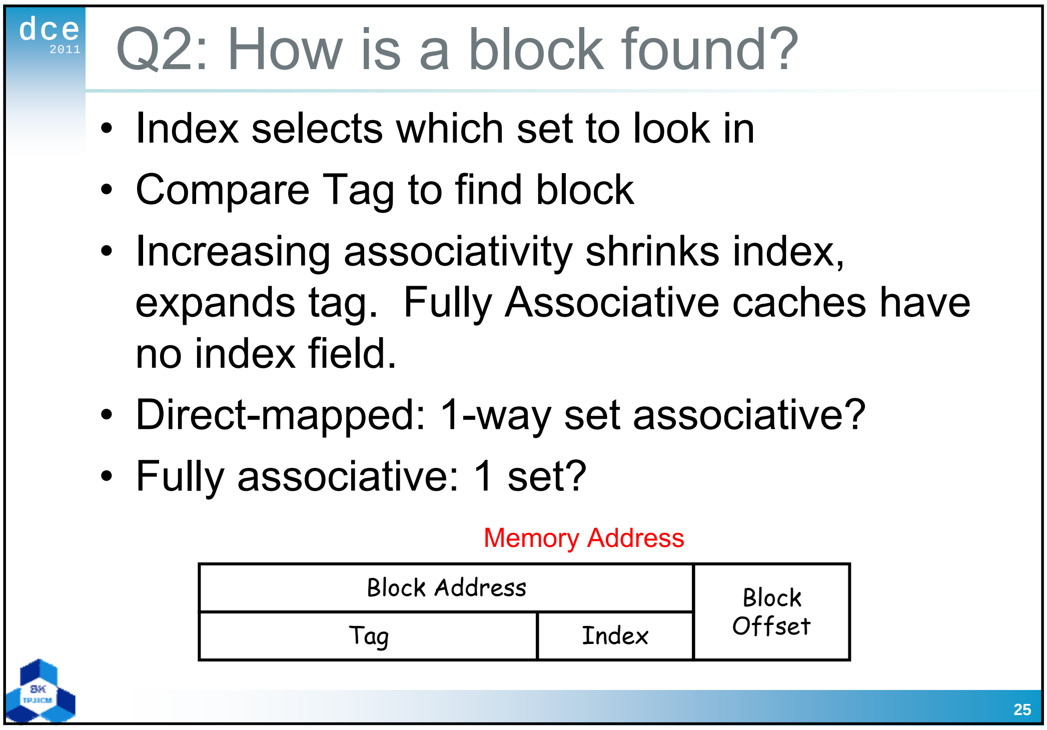
Block Offset

Index Field

Set associative cache requires parallel tag matching and more complex hit logic which may increase hit time

Block Address = 30 bits		Block offset = 2 bits
Tag = 22 bits	Index = 8 bits	

Mapping Function: Cache Set Number = index= (Block address) MOD (256)

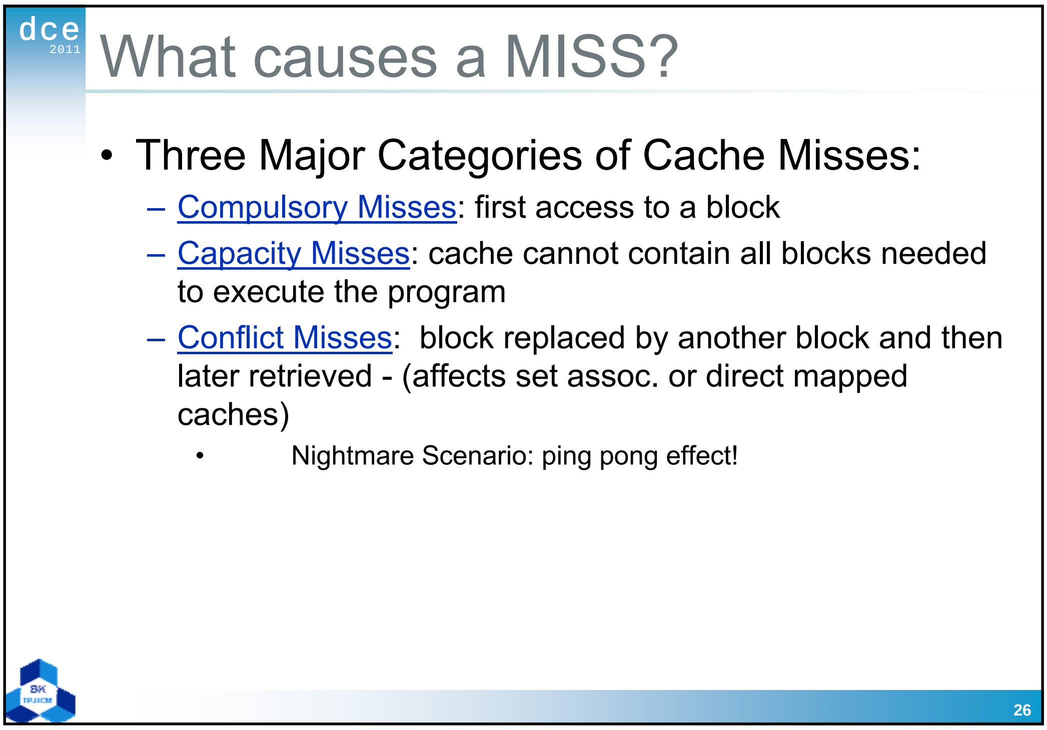


Q2: How is a block found?

- Index selects which set to look in
- Compare Tag to find block
- Increasing associativity shrinks index, expands tag. Fully Associative caches have no index field.
- Direct-mapped: 1-way set associative?
- Fully associative: 1 set?

Memory Address

Block Address		Block Offset
Tag	Index	

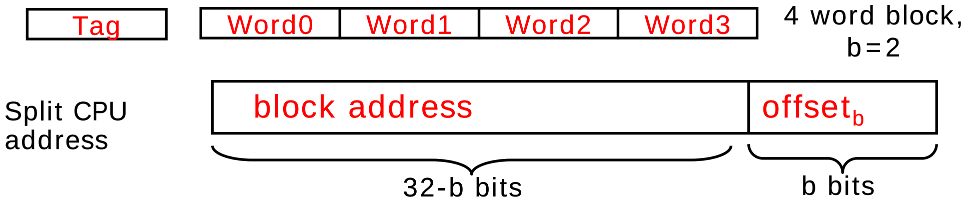


What causes a MISS?

- Three Major Categories of Cache Misses:
  - Compulsory Misses: first access to a block
  - Capacity Misses: cache cannot contain all blocks needed to execute the program
  - Conflict Misses: block replaced by another block and then later retrieved - (affects set assoc. or direct mapped caches)
    - Nightmare Scenario: ping pong effect!

Block Size and Spatial Locality

Block is unit of transfer between the cache and memory



2^b = block size *a.k.a* line size (in bytes)

Larger block size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing block size?

Fewer blocks => more conflicts. Can waste bandwidth.

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - Least Recently Used (LRU)
    - LRU cache state must be updated on every access
    - true implementation only feasible for small sets (2-way, 4-way)
    - pseudo-LRU binary tree often used for 4-8 way
  - First In, First Out (FIFO) a.k.a. Round-Robin
    - used in highly associative caches
- Replacement policy has a second order effect since replacement only happens on misses

Q4: What happens on a write?

- Cache hit:
  - *write through:* write both cache & memory
    - generally higher traffic but simplifies cache coherence
  - *write back:* write cache only (memory is written only when the entry is evicted)
    - a dirty bit per block can further reduce the traffic
- Cache miss:
  - *no write allocate:* only write to main memory
  - *write allocate (aka fetch on write):* fetch into cache
- Common combinations:
  - write through and no write allocate (below example)
  - write back with write allocate (above Example)

Reading assignment 1

- Cache coherent problem in multicore systems
  - Identify the problem
  - Algorithms for multicore architectures
- Reference
  - *eecs.wsu.edu/~cs460/cs550/**cachecoherence**.pdf*
  - …More on internet

Reading assignment 2

- Cache performance
  - Replacement policy (algorithms)
  - Optimization (Miss rate, penalty, …)
- Reference
  - Hennessy - Patterson - Computer Architecture. A Quantitative
  - *www2.lns.mit.edu/~avinatan/research/**cache**.pdf*
  - *… More on internet*