ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

# COMPUTER ORGANISATION
## (TỔ CHỨC MÁY TÍNH)

# Cache Part II

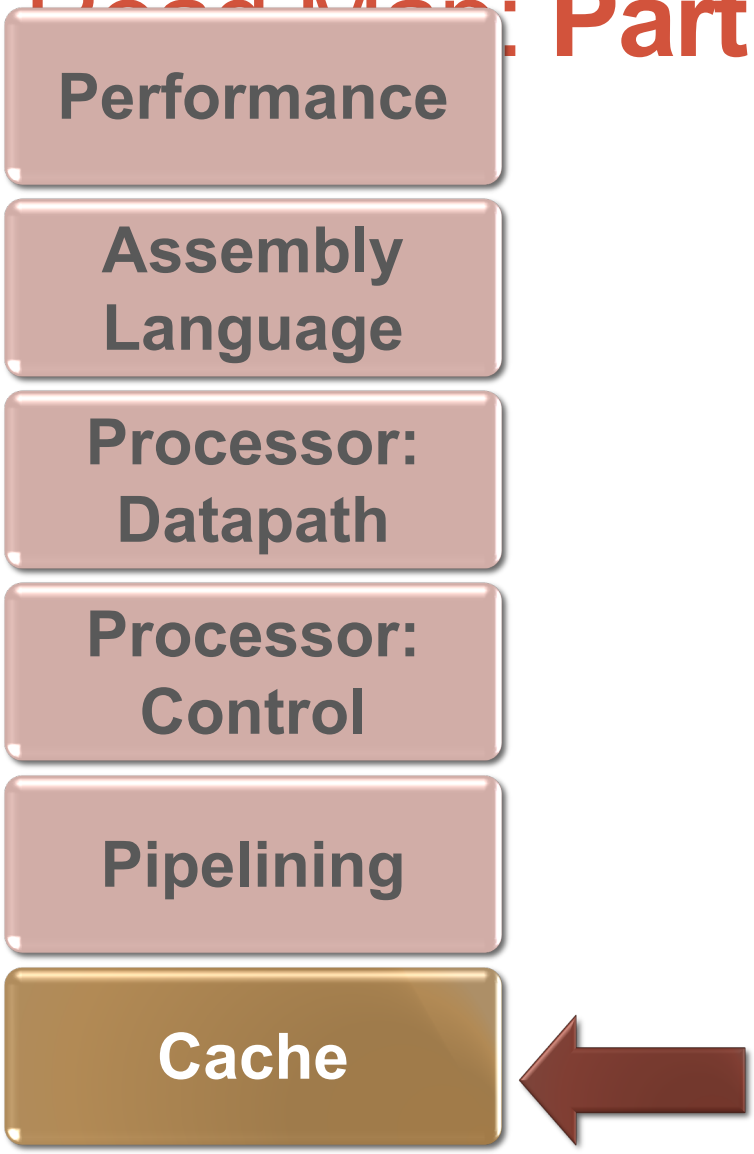
# Acknowledgement


- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

# Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

# Road Map: Part II




- **Cache Part II**
  - Type of Cache Misses
  - Direct-Mapped Cache
  - Set Associative Cache
  - Fully Associative Cache
  - Block Replacement Policy
  - Cache Framework
  - Improving Miss Penalty
  - Multi-Level Caches

# Cache Misses: Classifications

## Compulsory / Cold Miss

- First time a **memory block** is accessed
- Cold fact of life: Not much can be done
- **Solution:** Increase cache block size

## Conflict Miss

- Two or more distinct memory blocks map to the same cache block
- Big problem in direct-mapped caches
- **Solution 1**: Increase cache size
  - Inherent restriction on cache size due to SRAM technology
- **Solution 2**: **Set-Associative caches** (coming next ..)

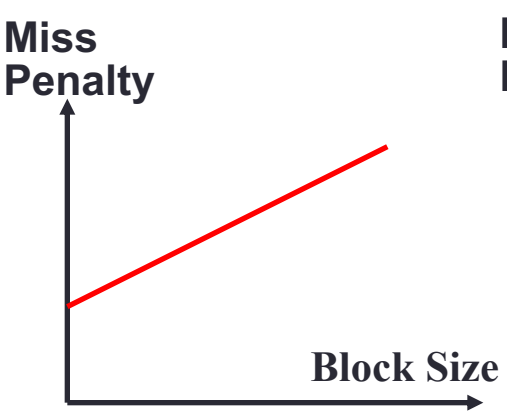
## Capacity Miss

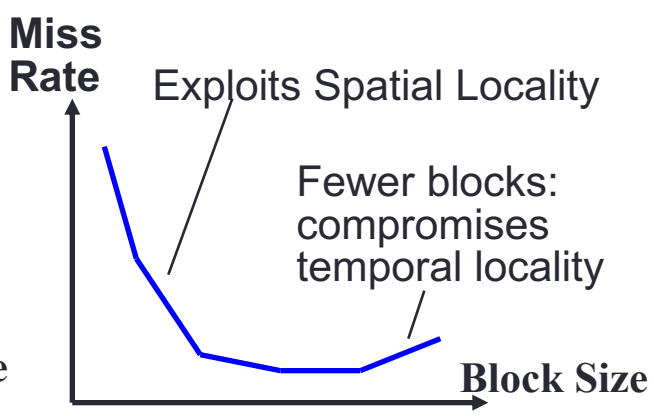
- Due to limited cache size
- Will be further clarified in "fully associative caches" later

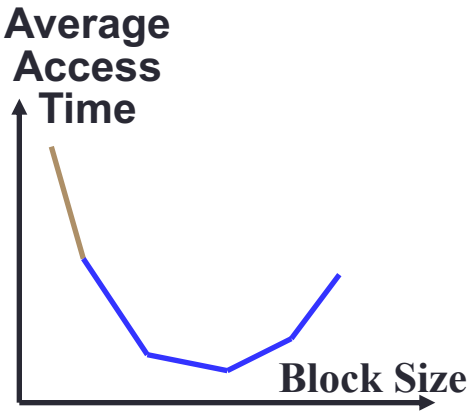
# Block Size Tradeoff (1/3)

**Average Access Time**
= Hit rate x Hit Time + (1-Hit rate) x Miss penalty

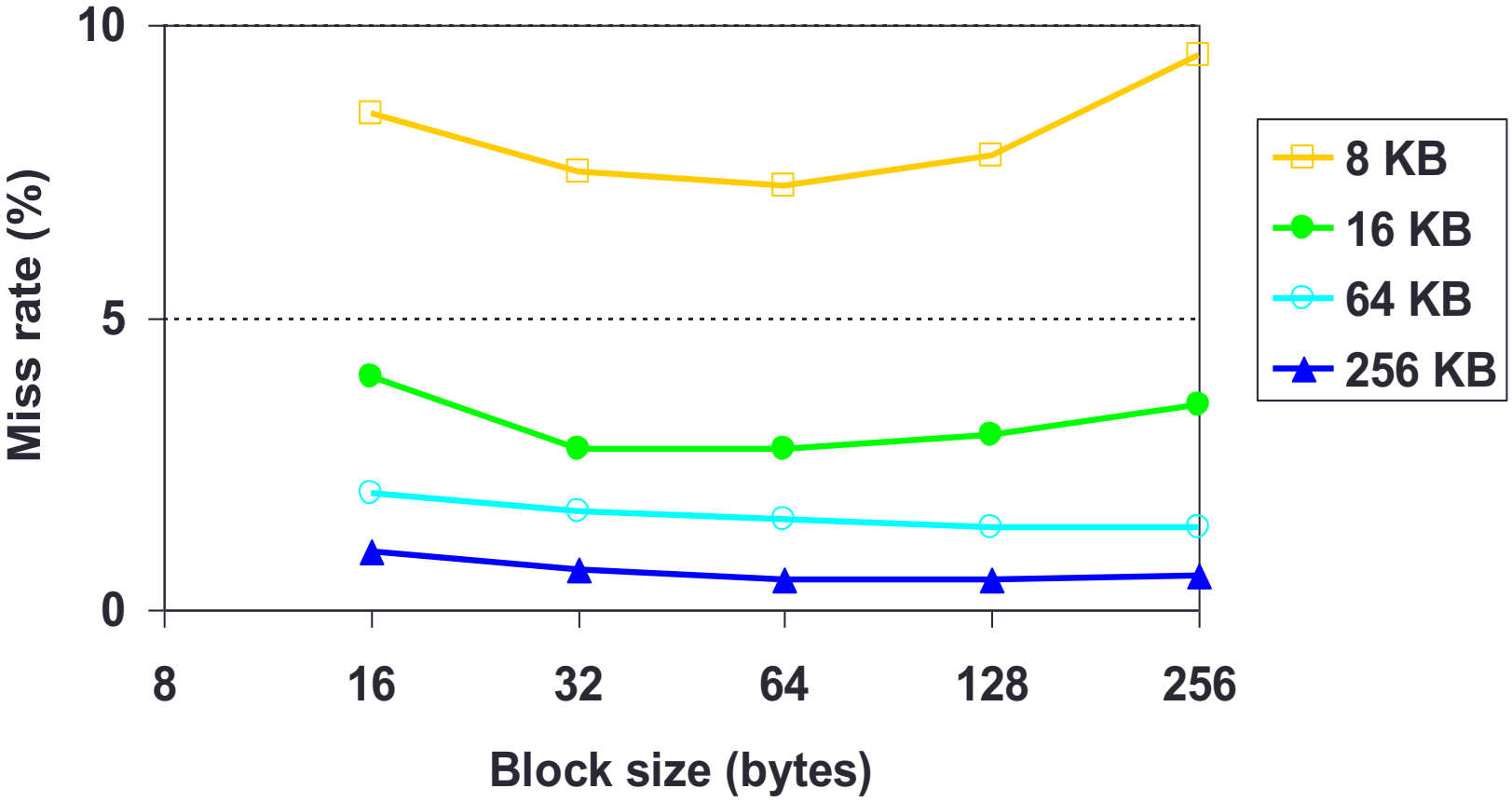
- Larger block size:
  - \+ Takes advantage of spatial locality
  - -- Larger miss penalty: Takes longer time to fill up the block
  - -- If block size is too big relative to cache size
    - ➔ Too few cache blocks ➔ miss rate will go up



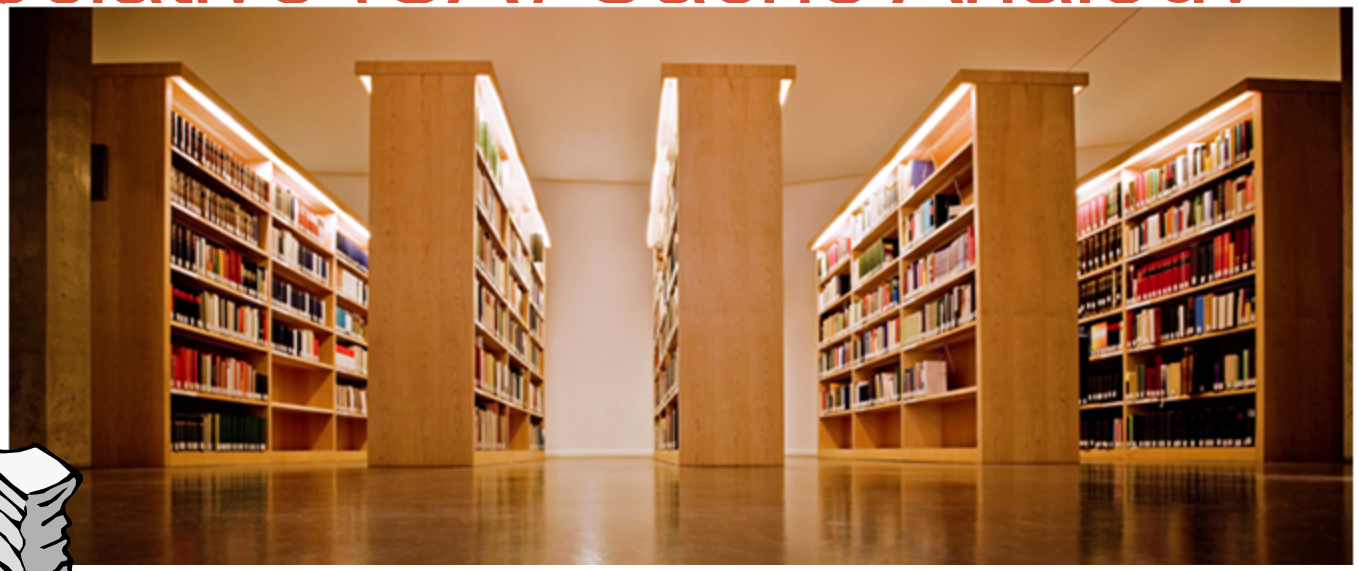
# Block Size Tradeoff (2/2)



# SET ASSOCIATIVE CACHE

Another way to organize the cache blocks

# Set-Associative (SA) Cache Analogy

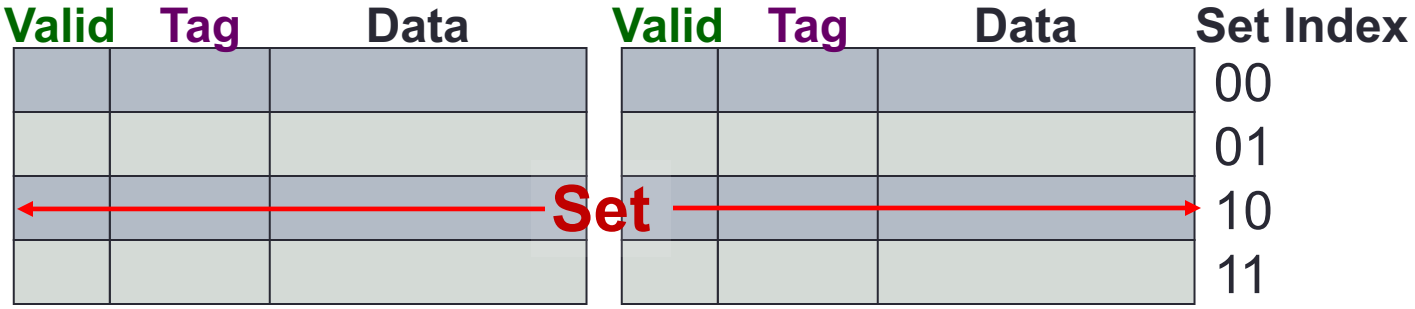


Many book titles start with “T”
➔ Too many conflicts!
Hmm… how about we give more slots per letter, 2 books start with “A”, 2 books start with “B”, …. etc?

# Set Associative (SA) Cache

- **N-way Set Associative Cache**
  - A memory block can be placed in a fixed number of locations (`N > 1`) in the cache
- **Key Idea:**
  - Cache consists of a number of sets:
    - Each set contains **N** cache blocks
  - Each memory block maps to a unique cache set
  - Within the set, a memory block can be placed in **any** element of the set

# SA Cache Structure

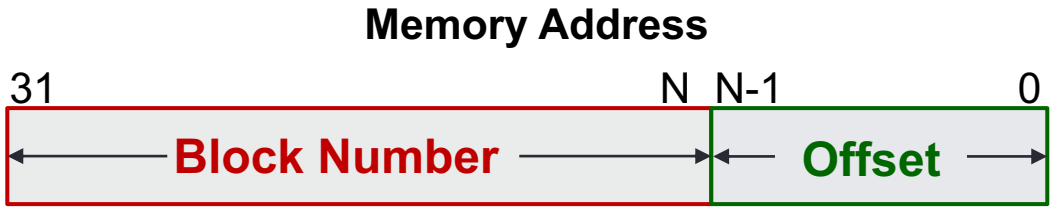



2-way Set Associative Cache

- An example of 2-way set associative cache
  - Each set has two cache blocks
- A memory block maps to an **unique set**
  - In the set, the memory block can be placed in **either of the cache blocks**

➔ Need to search both to look for the memory block

# Set-Associative Cache: **Mapping**



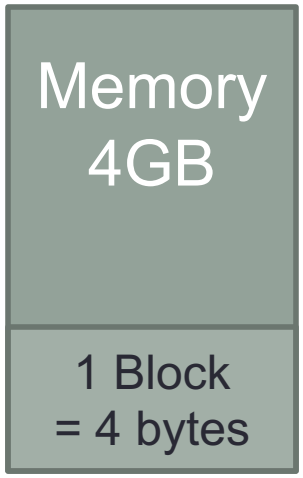

Cache Block size = $2^N$ bytes

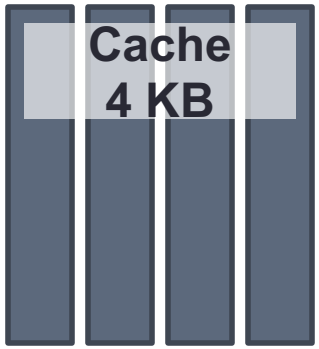
**Cache Set Index**
= (BlockNumber) modulo (NumberOfCacheSets)


Cache Block size = $2^N$ bytes
Number of cache sets = $2^M$
**Offset** = **N bits**
**Set Index** = **M bits**
**Tag** = **32 – (N + M) bits**

**Observation:**
It is essentially unchanged from the direct-mapping formula

# SA Cache Mapping: **Example**




**Memory Address**


**Offset, N = 2 bits**

**Block Number** = 32 – 2 = **30 bits**

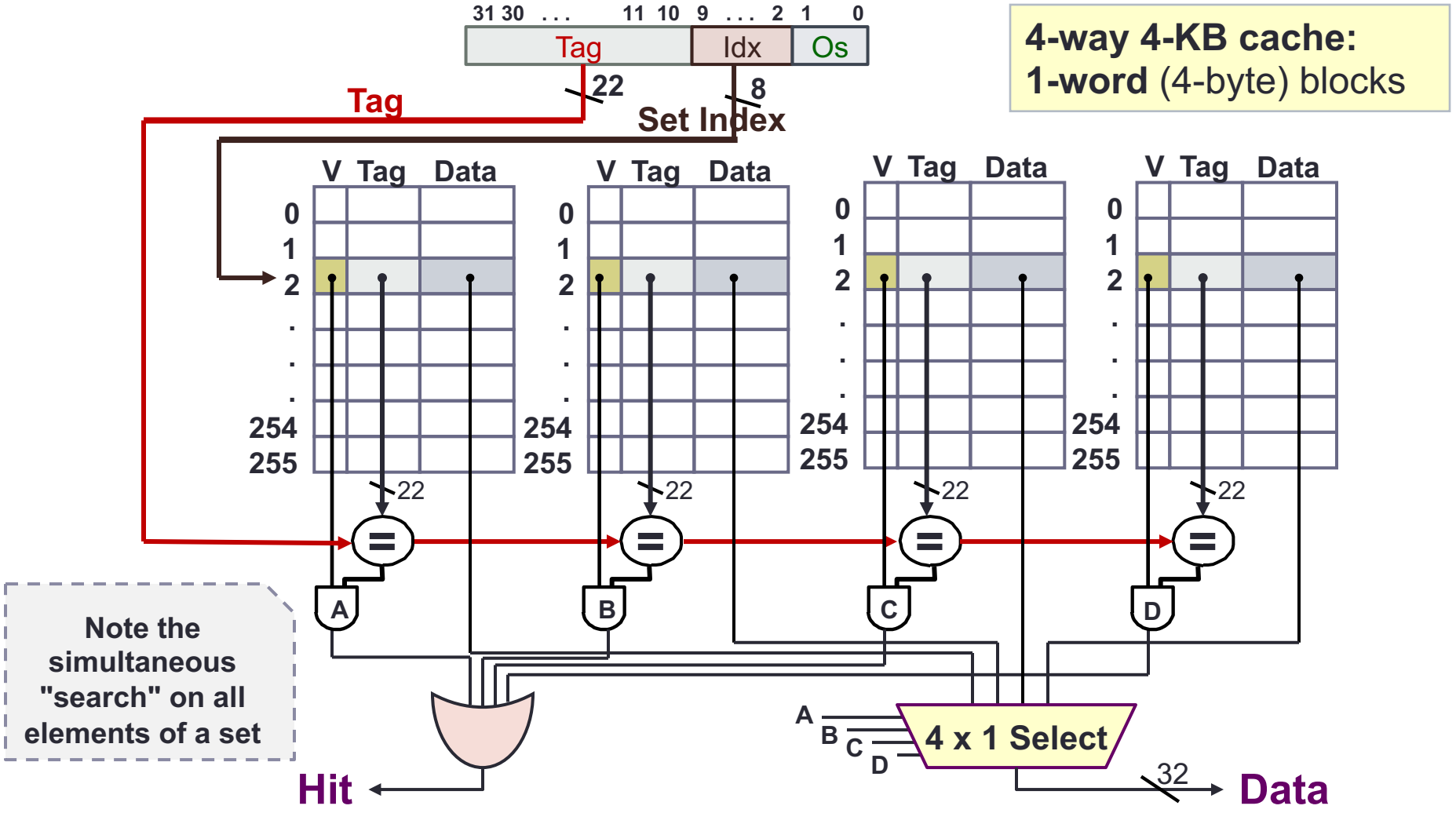
Check: Number of Blocks = $2^{30}$


**Number of Cache Blocks**
= 4KB / 4bytes = 1024 **= $2^{10}$**

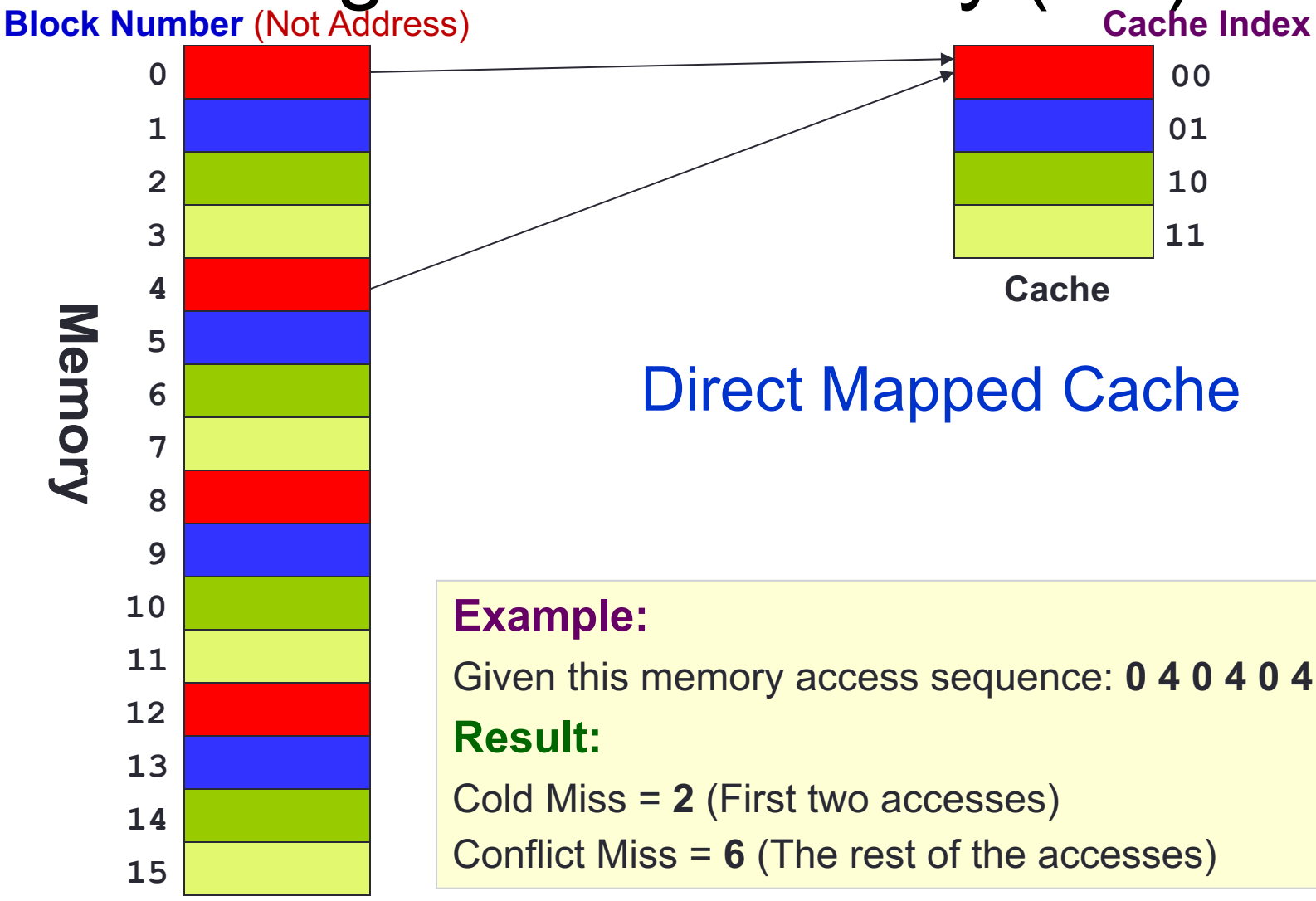
**4-way associative, number of sets**
= 1024 / 4 = 256 **= $2^{8}$**
**Set Index, M = 8 bits**

**Cache Tag** = 32 – 8 – 2 = **22 bits**

# Set Associative Cache Circuitry



# Advantage of Associativity (1/3)




Direct Mapped Cache

**Example:**

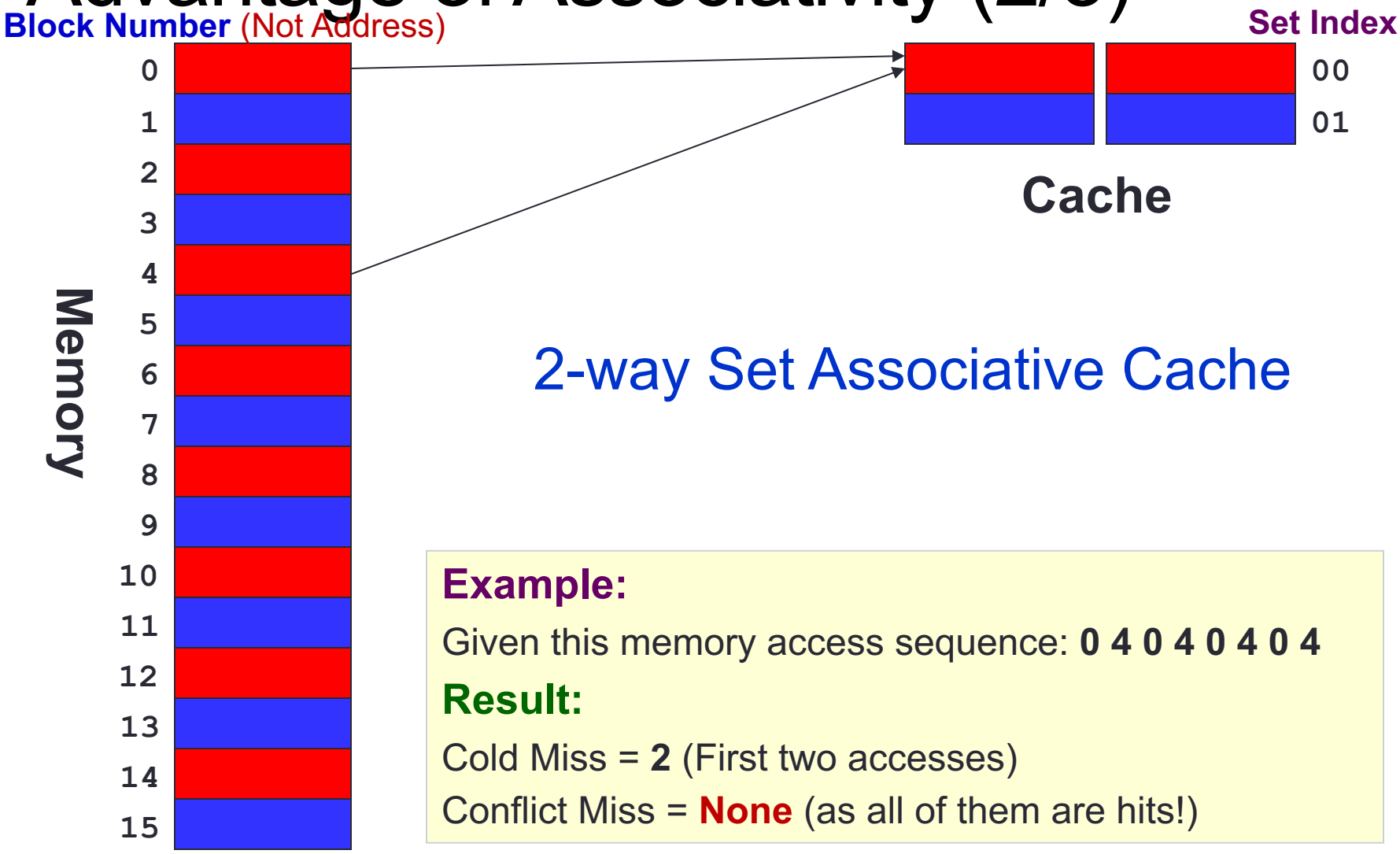
Given this memory access sequence: **0 4 0 4 0 4 0 4**

**Result:**

Cold Miss = **2** (First two accesses)

Conflict Miss = **6** (The rest of the accesses)

# Advantage of Associativity (2/3)




2-way Set Associative Cache

**Example:**

Given this memory access sequence: **0 4 0 4 0 4 0 4**

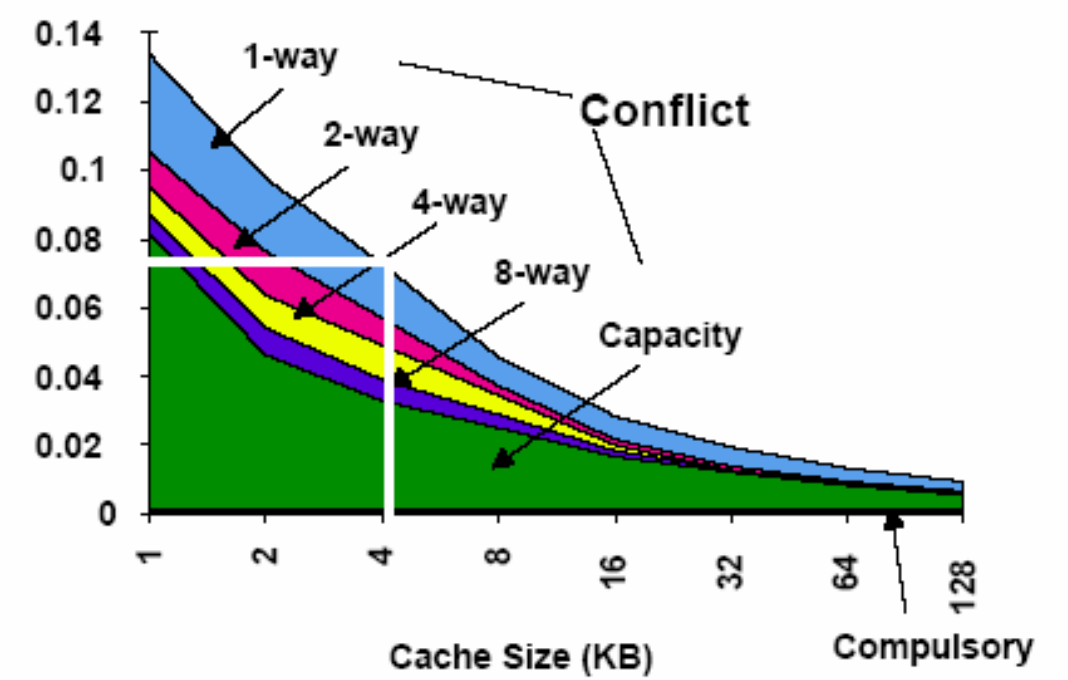
**Result:**

Cold Miss = **2** (First two accesses)

Conflict Miss = **None** (as all of them are hits!)

# Advantage of Associativity (3/3)

**Rule of Thumb:**
A direct-mapped cache of size **N** has about the same miss rate as a 2-way set associative cache of size **N/2**



# SA Cache Example: **Setup**

- Given:
  - Memory access sequence: **4, 0, 8, 36, 0**
  - 2-way set-associative cache with a total of four 8-byte blocks ➔ **total of 2 sets**
  - Indicate hit/miss for each access


**Offset, N = 3 bits**
**Block Number** = 32 – 3 = 29 bits

**2-way associative, number of sets**= 2 **= $2^1$**
**Set Index, M = 1 bits**

**Cache Tag** = 32 – 3 – 1 = **28 bits**

# Example: LOAD #1

Miss

4. 0 , 8, 36, 0

- Load from **4** ➔

Tag | Index | Offset

00000000000000000000000000 | 0 | 100

**Check:** Both blocks in **Set 0** are invalid **[ Cold Miss ]**

**Result:** Load from memory and place in **Set 0 - Block 0**

| Set Index | Block 0 | | | | Block 1 | | | |
|---|---|---|---|---|---|---|---|---|
| | Valid | Tag | W0 | W1 | Valid | Tag | W0 | W1 |
| 0 | ~~0~~ 1 | 0 | M[0] | M[4] | 0 | | | |
| 1 | 0 | | | | 0 | | | |

# Example: LOAD #2

Miss Hit

4, 0, 8, 36, 0

Tag | Index | Offset

- Load from **0** ➔ 00000000000000000000000000 | 0 | 000

**Result:**

[Valid and Tag match] in **Set 0-Block 0 [ Spatial Locality ]**

| Set Index | Block 0 | | | | Block 1 | | | |
|---|---|---|---|---|---|---|---|---|
| | Valid | Tag | W0 | W1 | Valid | Tag | W0 | W1 |
| 0 | 1 | 0 | M[0] | M[4] | 0 | | | |
| 1 | 0 | | | | 0 | | | |

Miss Hit Miss

4, 0 , 8, 36, 0

# Example: LOAD #3

Tag | Index | Offset

- Load from **8** ➔ 00000000000000000000000000 | 1 | 000

**Check:** Both blocks in **Set 1** are invalid **[ Cold Miss ]**

**Result:** Load from memory and place in **Set 1 - Block 0**

| Set Index | Block 0 | | | | Block 1 | | | |
|---|---|---|---|---|---|---|---|---|
| | Valid | Tag | W0 | W1 | Valid | Tag | W0 | W1 |
| 0 | 1 | 0 | M[0] | M[4] | 0 | | | |
| 1 | ~~0~~ 1 | 0 | M[8] | M[12] | 0 | | | |

Miss Hit Miss Miss

4, 0 , 8, 36, 0

# Example: LOAD #4

Tag | Index | Offset

- Load from **36** ➔ 00000000000000000000000010 | 0 | 100

**Check:** [Valid but tag mismatch] **Set 0 - Block 0**
[Invalid] **Set 0 - Block1 [ Cold Miss ]**

**Result:** Load from memory and place in **Set 0 - Block 1**

| Set Index | Block 0 | | | | Block 1 | | | |
|---|---|---|---|---|---|---|---|---|
| | Valid | Tag | W0 | W1 | Valid | Tag | W0 | W1 |
| 0 | 1 | 0 | M[0] | M[4] | ~~0~~ 1 | 2 | M[32] | M[36] |
| 1 | 1 | 0 | M[8] | M[12] | 0 | | | |

Miss Hit Miss Miss Hit

4, 0 , 8, 36, 0

# Example: LOAD #5

- Load from **0** ➔

Tag | Index | Offset

00000000000000000000000000 | 0 | 000

**Check:** [Valid and tag match] **Set 0-Block 0**
[Valid but tag mismatch] **Set 0-Block1**

**[ Temporal Locality ]**

| Set Idx | Block 0 | | | | Block 1 | | | |
|---|---|---|---|---|---|---|---|---|
| | Valid | Tag | W0 | W1 | Valid | Tag | W0 | W1 |
| 0 | 1 | 0 | M[0] | M[4] | 1 | 2 | M[32] | M[36] |
| 1 | 1 | 0 | M[8] | M[12] | 0 | | | |

# FULLY ASSOCIATIVE CACHE

How about total freedom?

# Fully-Associative (FA) Analogy

Let's not restrict the book by title any more. A book can go into any location on the desk!

# Fully Associative (FA) Cache

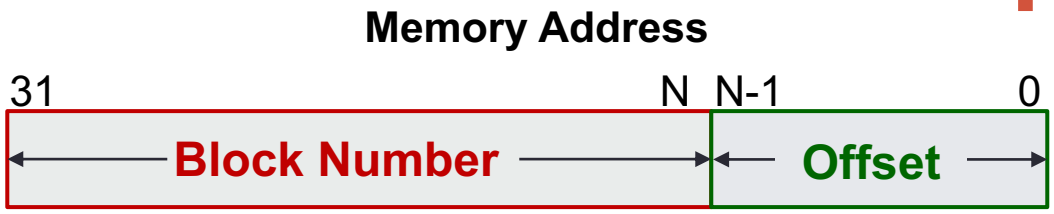
- **Fully Associative Cache**
  - A memory block can be placed in any location in the cache

- **Key Idea:**
  - Memory block placement is no longer restricted by cache index / cache set index

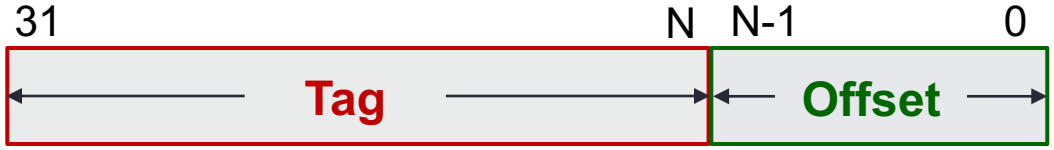
  **++** Can be placed in any location, **BUT**

  **---** Need to search all cache blocks for memory access

# Fully Associative Cache: Mapping




Cache Block size = $2^N$ bytes




Cache Block size = $2^N$ bytes

Number of cache blocks = $2^M$

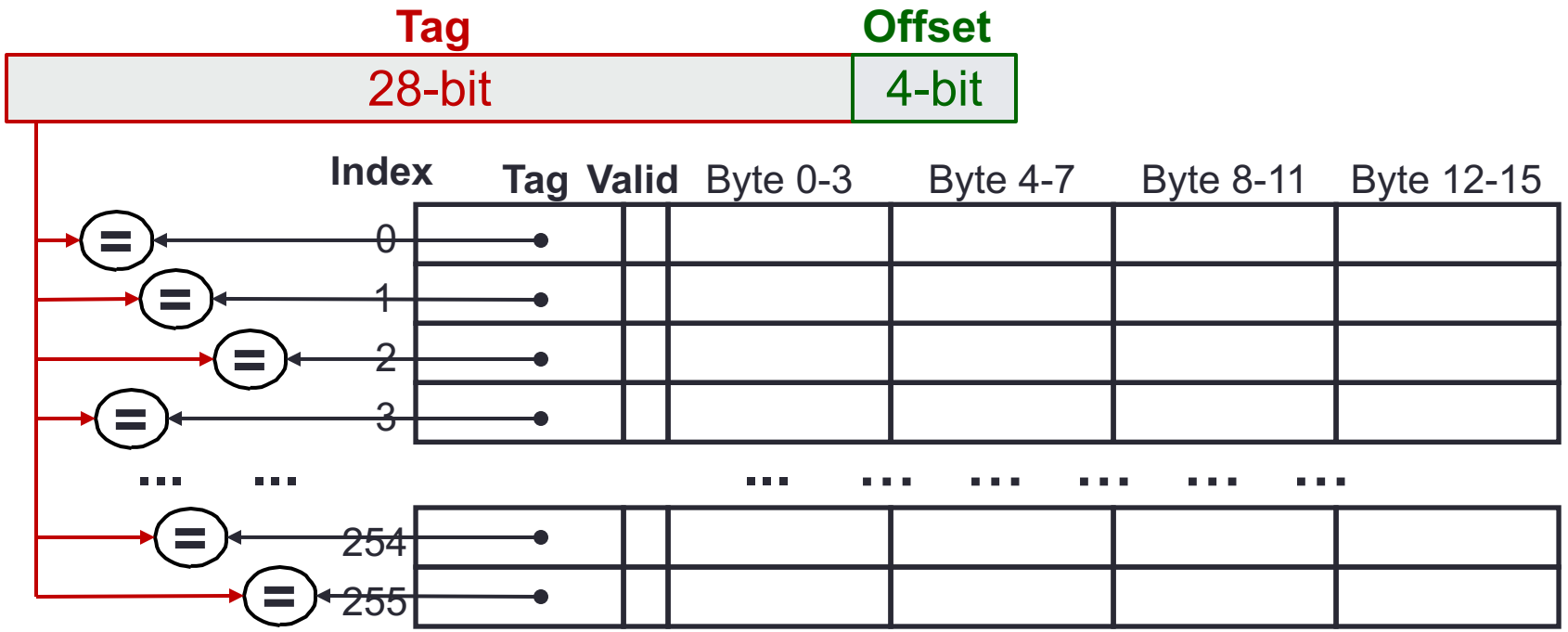
**Offset** = **N bits**

**Tag** = **32 – N bits**

**Observation:**
The block number serves as the tag in FA cache

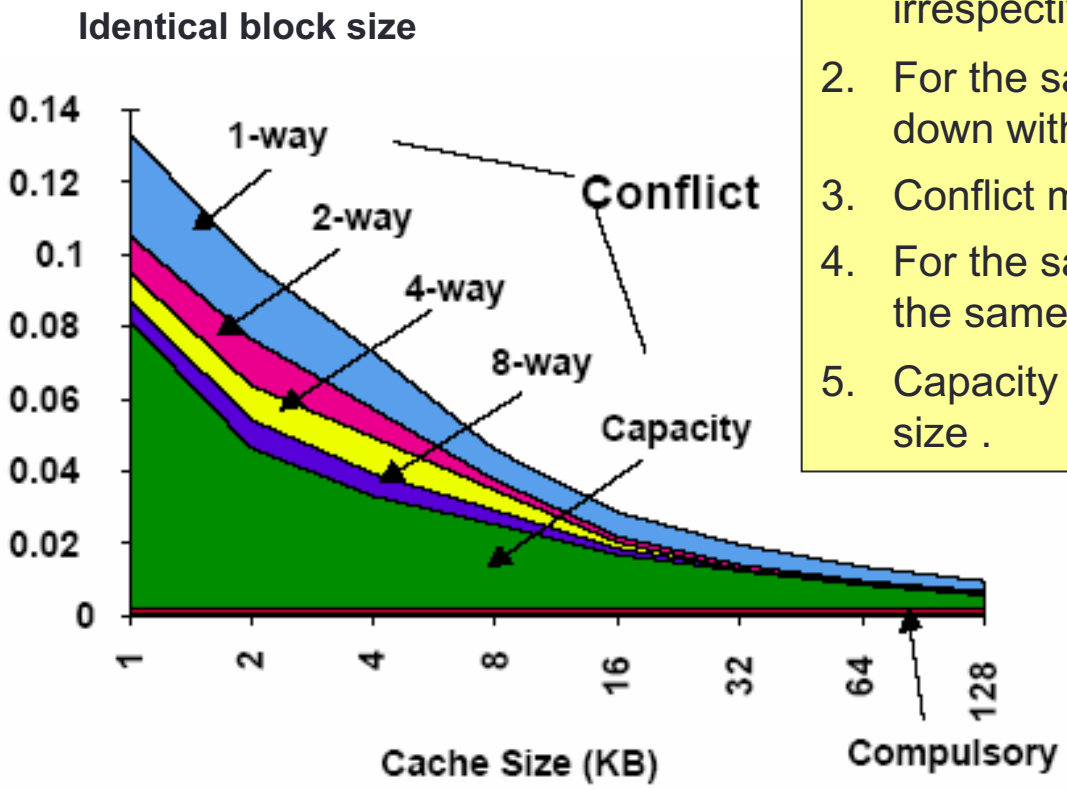
# Fully Associative Cache Circuitry

- Example:
  - 4KB cache size and 16-Byte block size
  - Compare tags and valid bit in parallel




**No Conflict Miss (since data can go anywhere)**

# Cache Performance




**Observations:**

1. Cold/compulsory miss remains the same irrespective of cache size/associativity.
2. For the same cache size, conflict miss goes down with increasing associativity.
3. Conflict miss is 0 for FA caches.
4. For the same cache size, capacity miss remains the same irrespective of associativity.
5. Capacity miss decreases with increasing cache size .

Total Miss = Cold miss + Conflict miss + Capacity miss

**Capacity miss (FA) = Total miss (FA) – Cold miss (FA)**, when Conflict Miss➔0

# BLOCK REPLACEMENT POLICY

Who should I kick out next…..?

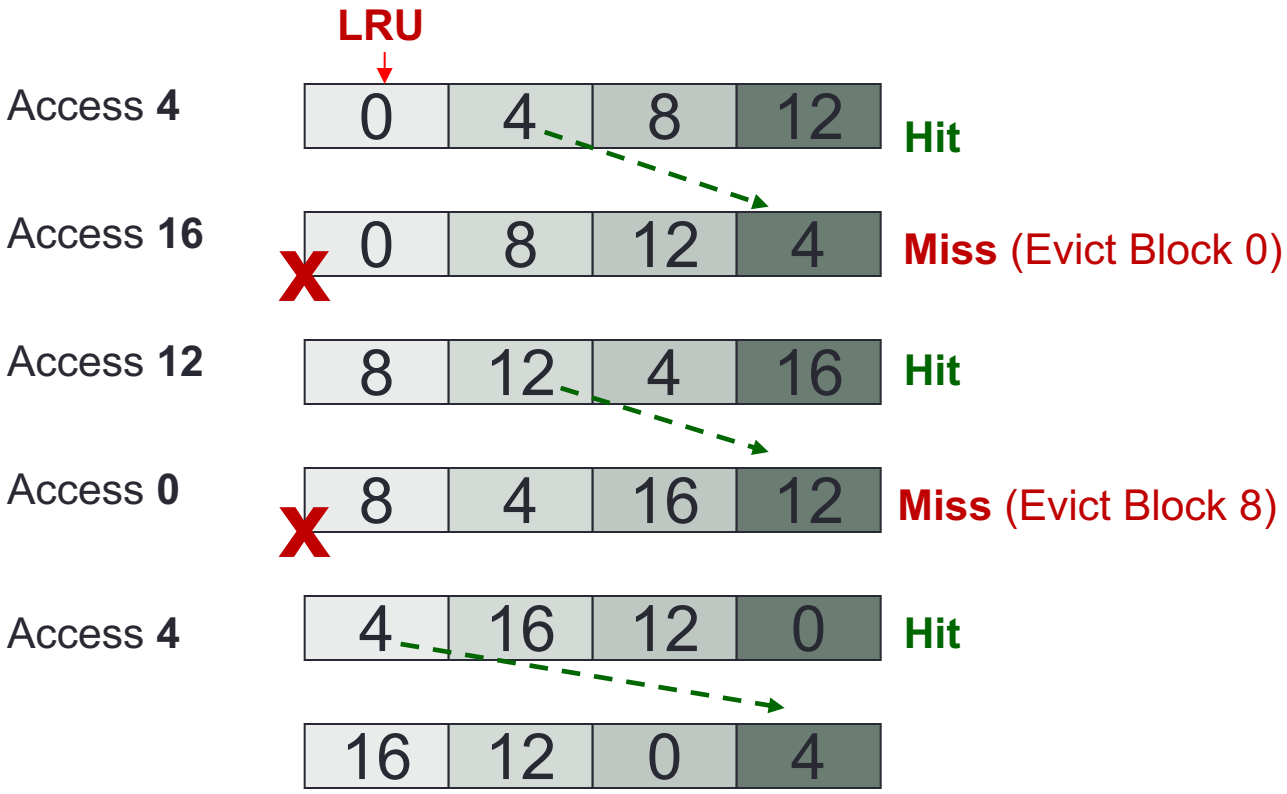
# Block Replacement Policy (1/3)

- Set Associative or Fully Associative Cache:
  - Can choose where to place a memory block
  - Potentially replacing another cache block if full
  - Need **block replacement policy**

- **L**east **R**ecently Used (**LRU**)
  - **How:** For cache hit, record the cache block that was accessed
    - When replacing a block, choose one which has not been accessed for the longest time
  - **Why:** Temporal locality

# Block Replacement Policy (2/3)

- Least Recently Used policy in action:
  - 4- way Cache
  - Memory accesses: **0 4 8 12 4 16 12 0 4**



# Block Replacement Policy (3/3)

- Drawback for LRU
  - Hard to keep track if there are many choices

- Other replacement policies:
  - First in first out (FIFO)
    - Second chance variant
  - Random replacement (RR)
  - Least frequently used (LFU)

# Additional Example #1

- **Direct-Mapped** Cache:
  - Four 8-byte blocks
- Memory accesses:
  **4, 8, 36, 48, 68, 0, 32**

| Addr: | Tag | Index | Offset |
|---|---|---|---|
| 4: | 00…000 | 00 | 100 |
| 8: | 00…000 | 01 | 000 |
| 36: | 00…001 | 00 | 100 |
| 48: | 00…001 | 10 | 000 |
| 68: | 00…010 | 00 | 100 |
| 0: | 00…000 | 00 | 000 |
| 32: | 00…001 | 00 | 000 |

| Index | Valid | Tag | Word0 | Word1 |
|---|---|---|---|---|
| 0 | ~~0~~ 1 | ~~0~~ 1 | ~~M[0]~~ M[32] | ~~M[4]~~ M[36] |
| 1 | ~~0~~ 1 | 0 | M[8] | M[12] |
| 2 | 0 | | | |
| 3 | 0 | | | |

# Additional Example #2

- **Fully-Associative** Cache:
  - Four 8-byte blocks
  - LRU Replacement Policy
- Memory accesses:
  **4, 8, 36, 48, 68, 0, 32**

| Addr: | Tag | Offset |
|---|---|---|
| 4: | 00…00000 | 100 |
| 8: | 00…00001 | 000 |
| 36: | 00…00100 | 100 |
| 48: | 00…00110 | 000 |
| 68: | 00…01000 | 100 |
| 0: | 00…00000 | 000 |
| 32: | 00…00100 | 000 |

| Index | Valid | Tag | Word0 | Word1 |
|---|---|---|---|---|
| 0 | ~~0~~ 1 | 0 | M[0] | M[4] |
| 1 | ~~0~~ 1 | 1 | M[8] | M[12] |
| 2 | ~~0~~ 1 | 4 | M[32] | M[36] |
| 3 | 0 | | | |

# Additional Example #3

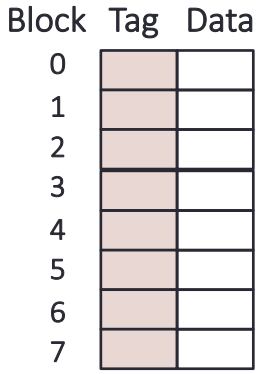
- **2-way Set-Associative** Cache:
  - Four 8-byte blocks
  - LRU Replacement Policy
- Memory accesses:
  **4, 8, 36, 48, 68, 0, 32**

| Addr: | Tag | Index | Offset |
|---|---|---|---|
| 4: | 00…0000 | 0 | 100 |
| 8: | 00…0000 | 1 | 000 |
| 36: | 00…0010 | 0 | 100 |
| 48: | 00…0011 | 0 | 000 |
| 68: | 00…0100 | 0 | 100 |
| 0: | 00…0000 | 0 | 000 |
| 32: | 00…0010 | 0 | 000 |

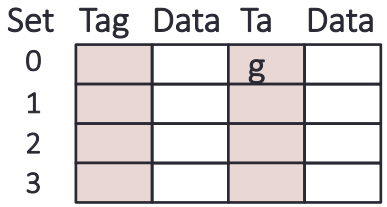
| Set Index | Block 0 | | | | Block 1 | | | |
|---|---|---|---|---|---|---|---|---|
| | Valid | Tag | Word0 | Word1 | Valid | Tag | Word0 | Word1 |
| 0 | ~~0~~ 1 | 0 | M[0] | M[4] | ~~0~~ 1 | 2 | M[32] | M[36] |
| 1 | ~~0~~ 1 | 0 | M[8] | M[12] | 0 | | | |

# Cache Organizations: Summary

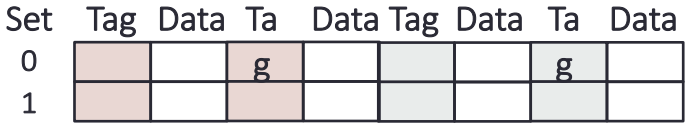
One-way set associative
(direct mapped)



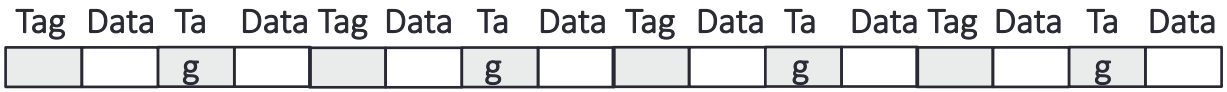

Two-way set associative




Four-way set associative




Eight-way set associative (fully associative)



# Cache Framework (1/2)

**Block Placement:** Where can a block be placed in cache?

| Direct Mapped: | N-way Set-Associative: | Fully Associative: |
|---|---|---|
| • Only one block defined by index | • Any one of the **N** blocks within the set defined by index | • Any cache block |

**Block Identification:** How is a block found if it is in the cache?

| Direct Mapped: | N-way Set Associative: | Fully Associative: |
|---|---|---|
| • Tag match with only one block | • Tag match for all the blocks within the set | • Tag match for all the blocks within the cache |

# Cache Framework (2/2)

**Block Replacement:** Which block should be replaced on a cache miss?

| Direct Mapped: | N-way Set-Associative: | Fully Associative: |
|---|---|---|
| • No Choice | • Based on replacement policy | • Based on replacement policy |

**Write Strategy:** What happens on a write?

Write Policy: Write-through vs. write-back

Write Miss Policy: Write allocate vs. write no allocate

# EXPLORATION

What else is there?

# Improving Cache Penalty

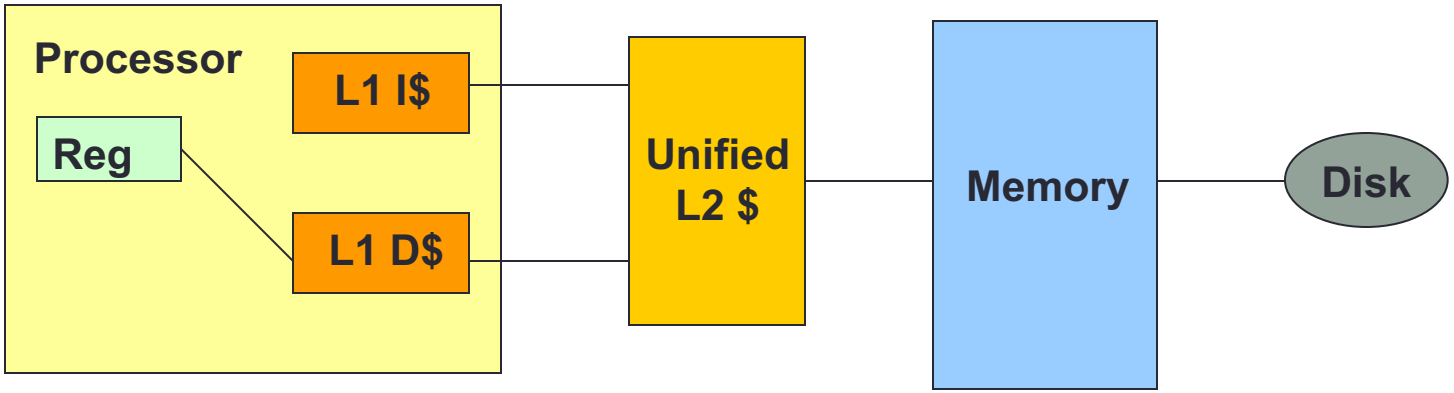
**Average access time** =

Hit rate x Hit Time + (1-Hit rate) x Miss penalty

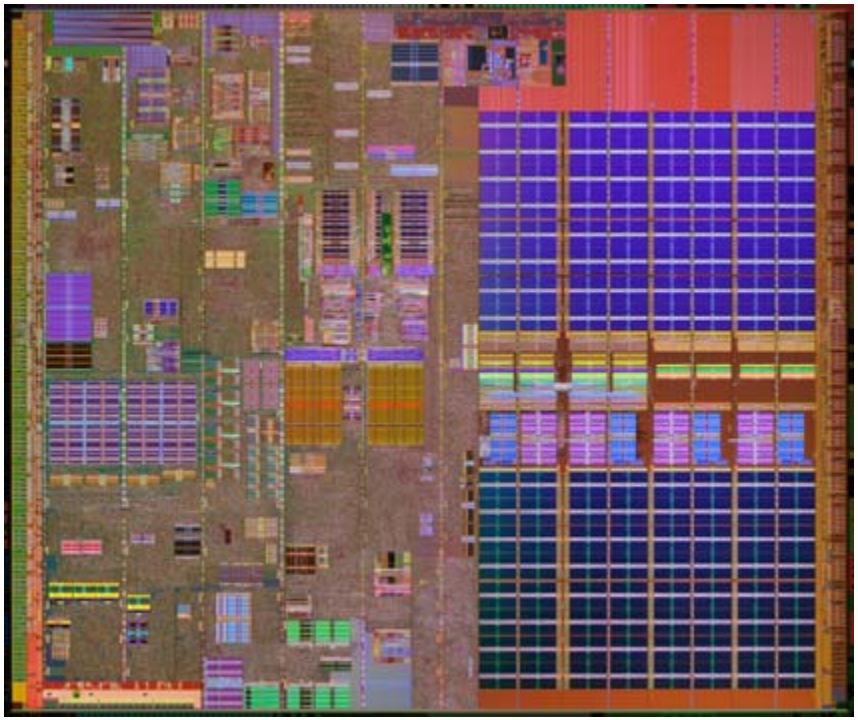
- So far, we tried to improve Miss Rate:
  - Larger block size
  - Larger Cache
  - Higher Associativity
- What about **Miss Penalty**?

# Multilevel Cache: **Idea**

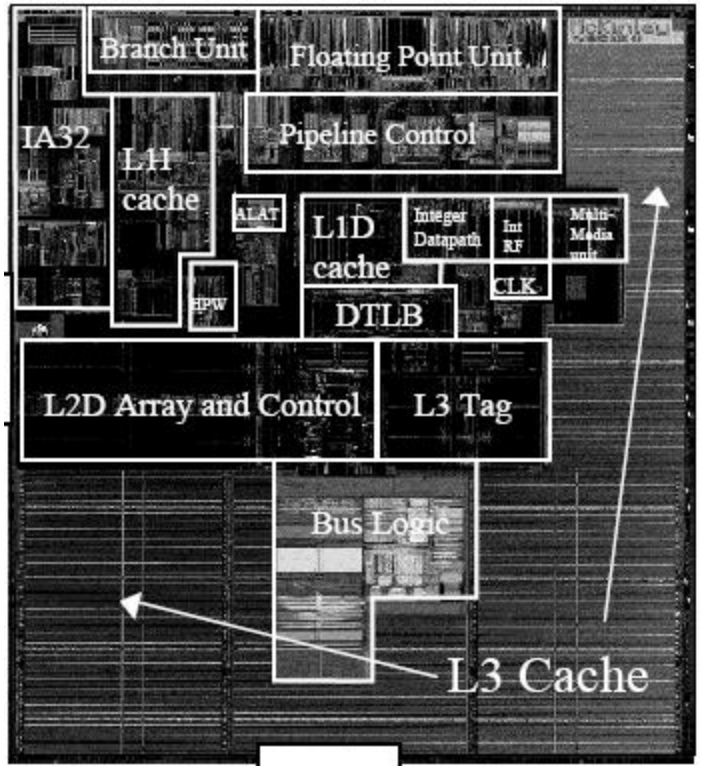
- Options:
  - Separate data and instruction caches or a unified cache
- Sample sizes:
  - **L1**: 32KB, 32-byte block, 4-way set associative
  - **L2**: 256KB, 128-byte block, 8-way associative
  - **L3**: 4MB, 256-byte block, Direct mapped



# Example: Intel Processors



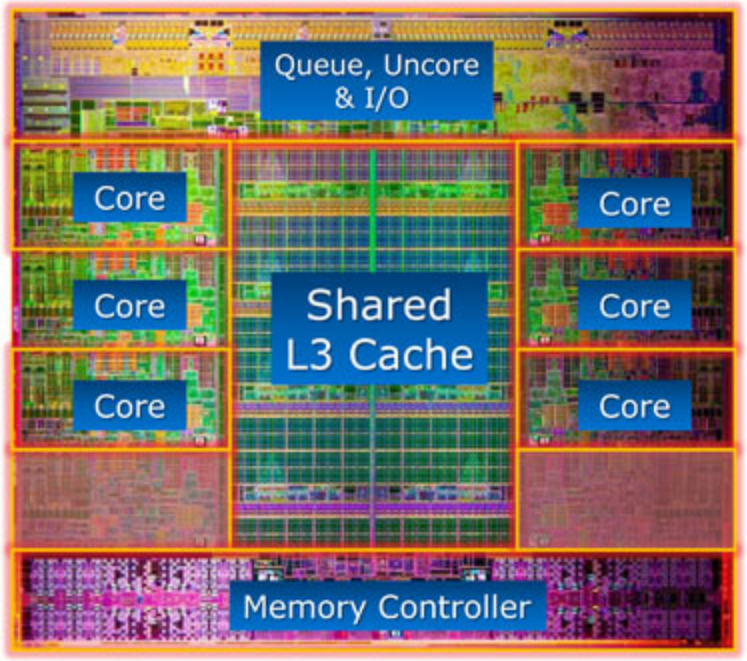
**Pentium 4 Extreme Edition**
L1: 12KB I$ + 8KB D$
L2: 256KB
L3: 2MB




**Itanium 2 McKinley**
L1: 16KB I$ + 16KB D$
L2: 256KB
L3: 1.5MB – 9MB

# Trend: Intel Core i7-3960K

## Intel® Core™ i7-3960X Processor Die Detail




**Intel Core i7-3960K**

**per die:**

-2.27 billion transistors

-15MB shared Inst/Data Cache (LLC)

**per Core:**

-32KB L1 Inst Cache

-32KB L1 Data Cache

-256KB L2 Inst/Data Cache

-up to 2.5MB LLC

# Reading Assignment

- Large and Fast: Exploiting Memory Hierarchy
  - Chapter 7 sections 7.3 (3rd edition)
  - Chapter 5 sections 5.3 (4th edition)

# Q&A