UNIVERSITY of WASHINGTON

CSS 422 Hardware and Computer Organization

Memory Hierarchy and Cache

Professor: Yang Peng

The slides are re-produced by the courtesy of
Dr. Arnie Berger, Dr. Ross Ortega and Dr. Wooyoung Kim

Topics

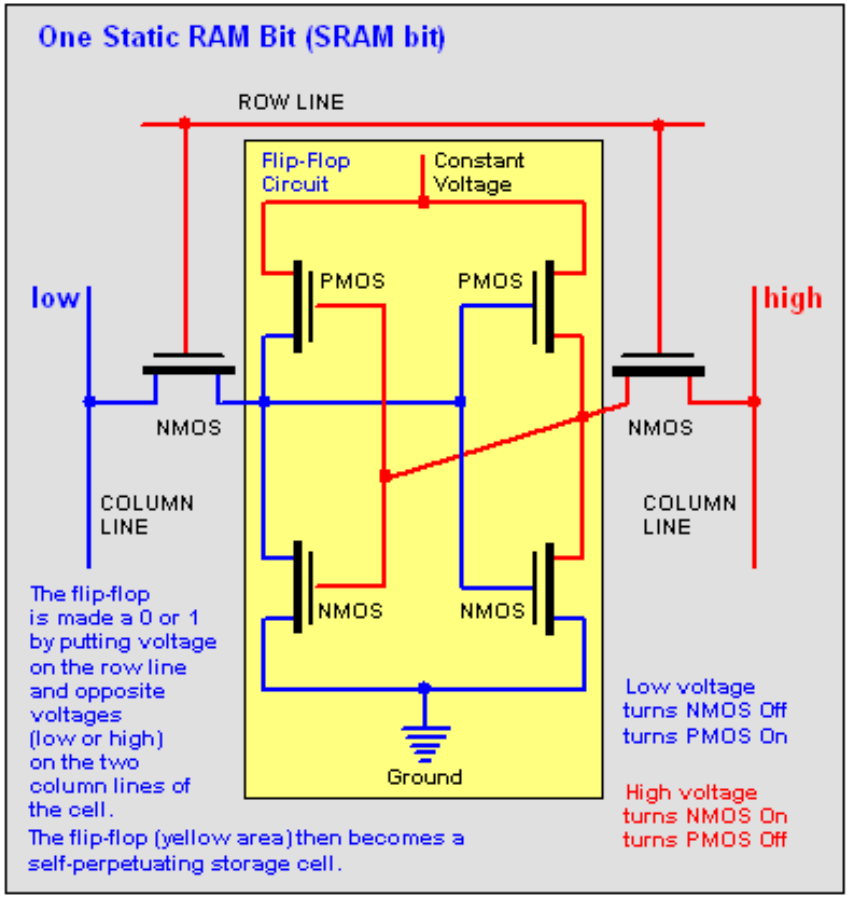
- Memory hierarchy
- Cache memory management
- Cache and performance

- Chapter 6 by Null
- Chapter 14 by Berger(Available online)

Memories: SRAM

From Computer Desktop Encyclopedia
© 2005 The Computer Language Co. Inc.

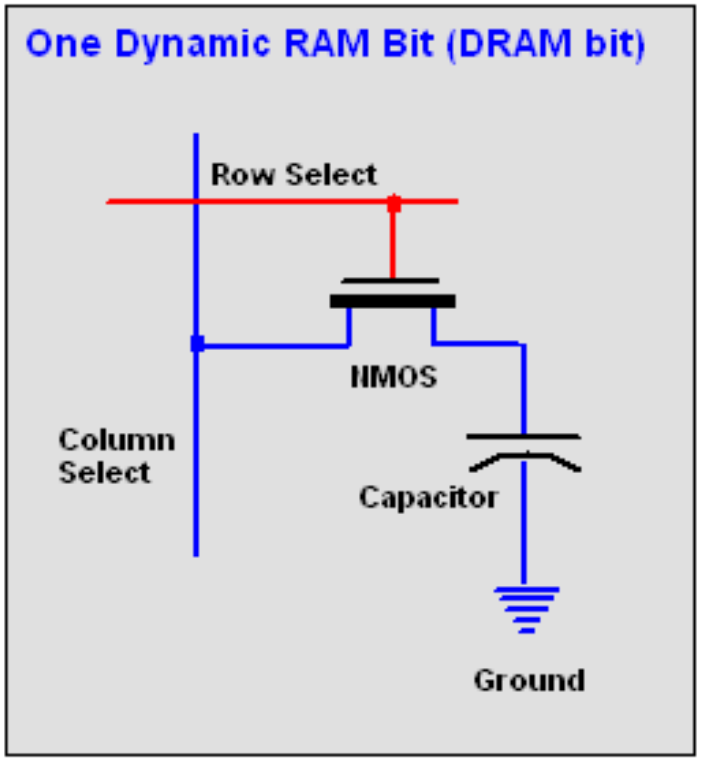
- Static Random-Access Memory
  - Value is stored using a pair of inverting gates
  - Fast, but takes up more space (4 to 6 transistors) than DRAM
  - Expensive



Memories: DRAM

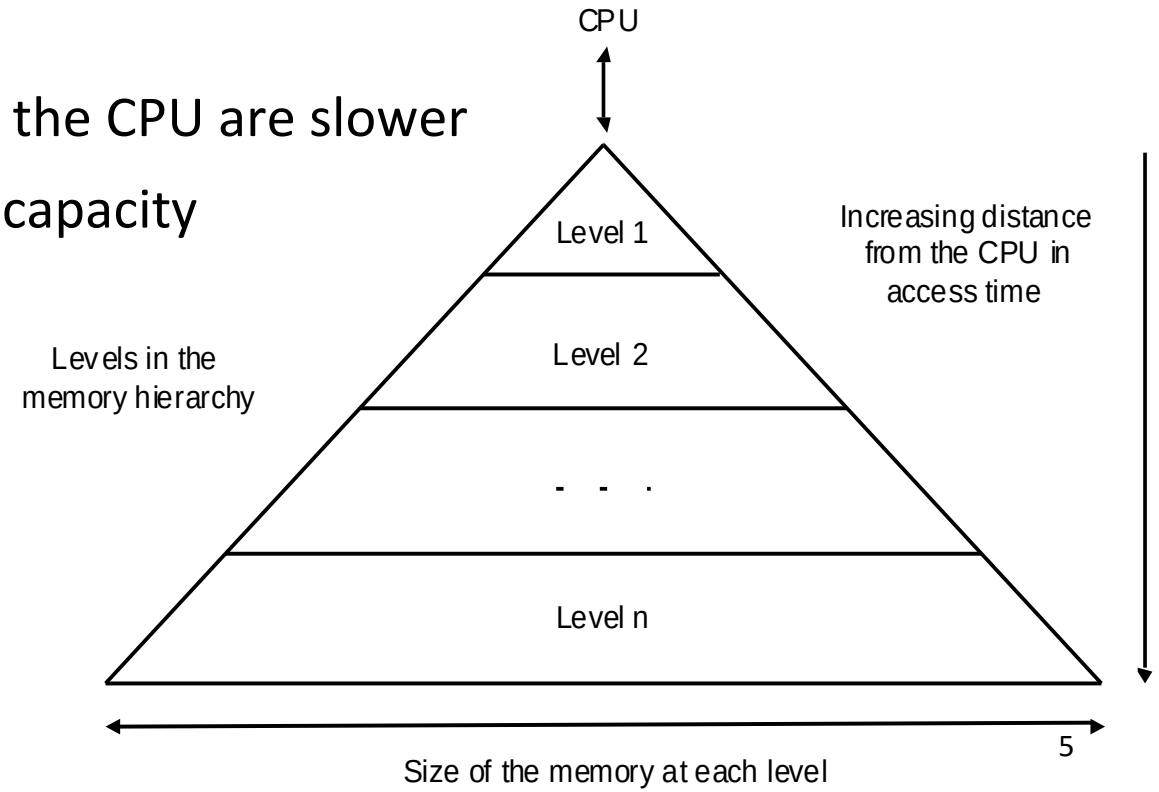
- Dynamic Random-Access Memory
  - Value is stored as a charge on capacitor (must be refreshed)
  - Very small but slower than SRAM (factor of 5 to 10)
  - Cheap

From Computer Desktop Encyclopedia



Memory Hierarchy

- *Memory Hierarchy*
  - Levels closer to the CPU are faster
    - Relatively small capacity
    - SRAM
  - Levels further from the CPU are slower
    - Relatively large capacity
    - DRAM

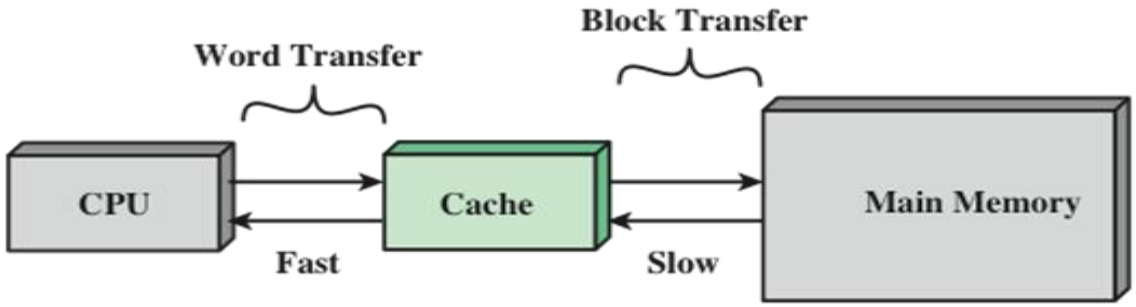


Locality

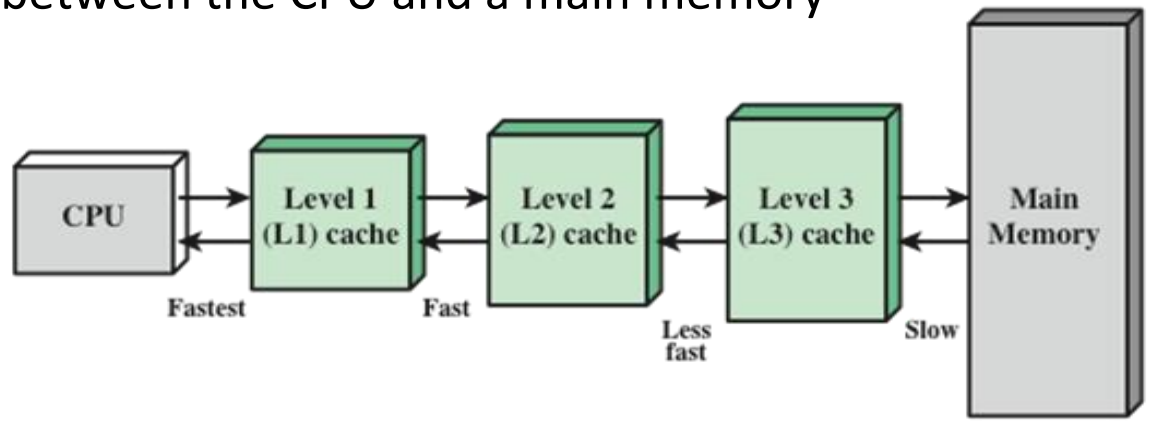
- If an item is referenced (read or write)
  - ***Temporal locality*** says
    - *It will tend to be referenced again soon*
  - ***Spatial locality*** says,
    - *Nearby items will tend to be referenced again soon*
- Our initial focus: two-level hierarchy (upper and lower)
  - **Upper** level is **cache** memory, **lower** level is **main memory**
  - Definitions
    - **Block**: the **minimum unit** of data
    - **Hit**: data requested is **in the cache** (upper level)
    - **Miss**: data requested is **not in the cache** (upper level)

Cache and Main Memory

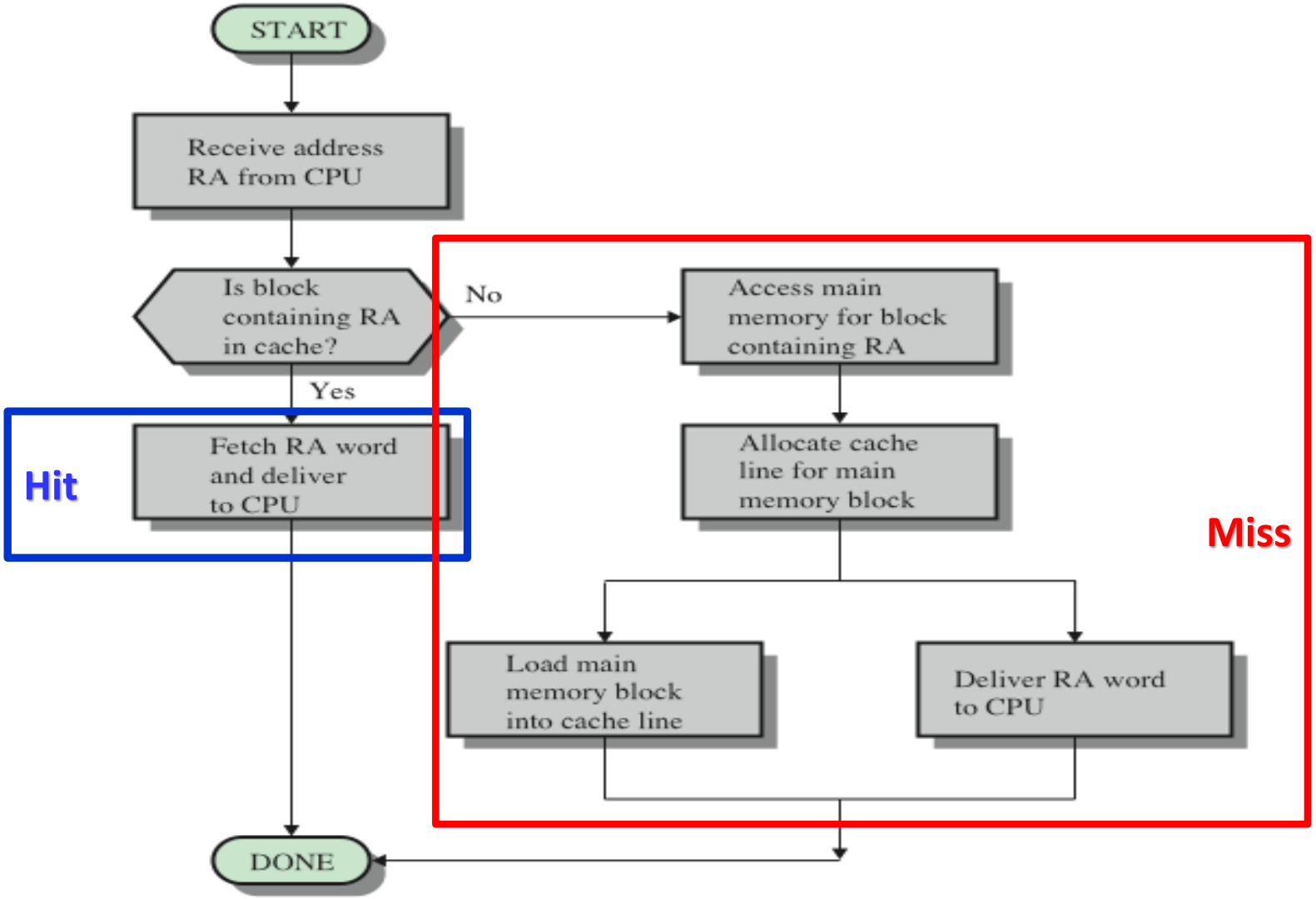
- A computer might have one or more caches between CPU and main memory



  - L1 cache usually means “On-chip” cache
  - L2/ L3 caches between the CPU and a main memory



Cache Read Operation



Effective Execution Time (EET)

- **Block**: unit of data transfer (called ***refill line*** as well)
- **Hit Rate**: percentage of accesses found in cache
- **Miss Rate**: percentage of accesses not in cache (*1 - hit rate*)
- **Hit Time**: time to access cache
- **Miss Penalty**: time to replace a block in cache with appropriate data in main memory (replace a whole block, not a single byte)
- **Effective Execution Time (EET):**
  - ***hit rate * hit time + miss rate * miss penalty***
  - **The goal is to achieve the minimum EET!**
  - Decreasing the miss rate (simultaneously increasing hit rate) is an important method

Class Exercise

1. A certain processor has an on-chip cache with the following specifications:
   - Cache **hit rate** for the L1 cache is **98%**
   - Instructions that are located in-cache execute in **1 clock cycle**
   - Instructions that are not found in the on-chip cache will cause the processor to stop program execution and refill a portion of the cache. This operation takes **100 clock cycles** to execute.

Q) What is the effective execution time in nanoseconds for this processor if the clock frequency is 100 MHz?

Class Exercise

1. A certain processor has an on-chip cache with the following specifications:
   - Cache hit rate for the L1 cache is 98%
   - Instructions that are located in-cache execute in 1 clock cycle
   - Instructions that are not found in the on-chip cache will cause the processor to stop program execution and refill a portion of the cache. This operation takes 100 clock cycles to execute.

Q) What is the effective execution time in nanoseconds for this processor if the clock frequency is 100 MHz?

Answer:

Clock cycle time = 1/100MHz = 10ns

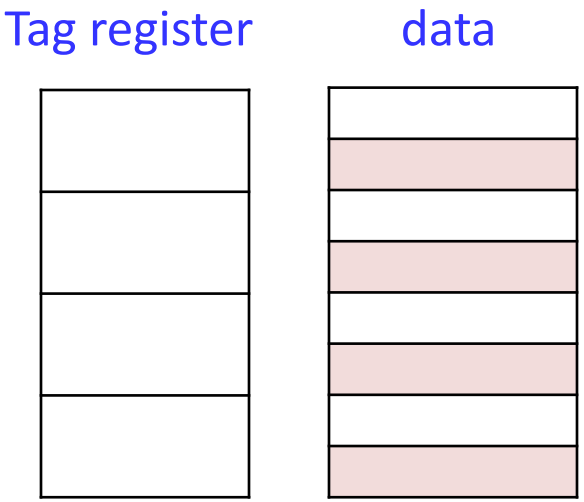
EET = hit rate X hit time + miss rate X miss penalty

= 0.98 X 1 (clock cycle) X 10ns + 0.02 X 100 (clock cycle) X 10 ns

= 9.8 ns + 20 ns = 29.8 ns

Cache Organization

- Cache memory consists of a number of ***cache entries***
  - **Tag register**: Holds the **memory address** and determines if there is a match to a CPU request
  - **Data**: Copies of the **contents** (instructions/data) of main memory



Cache Memory

Cache Design

- Goal: Increase the performance of using Caches
- Issues:
  - How to map the data in memory to the data in cache? How do we know if a data item is in the cache? If it is, how do we find it? (***Mapping Function***)
  - What happens if we change a data value in cache? (***Write Policy***)
  - What to replace when the cache is full? (***Replacement Algorithm***)
  - Is it a physical cache, or virtual cache? (***Cache Address***)
  - How big should be the Cache? (***Cache Size***)
  - How many adjacent data should come together in a cache? (***Line/Block Size***)
  - What is the optimal number of caches? (**Number of Caches**)

Mapping Function

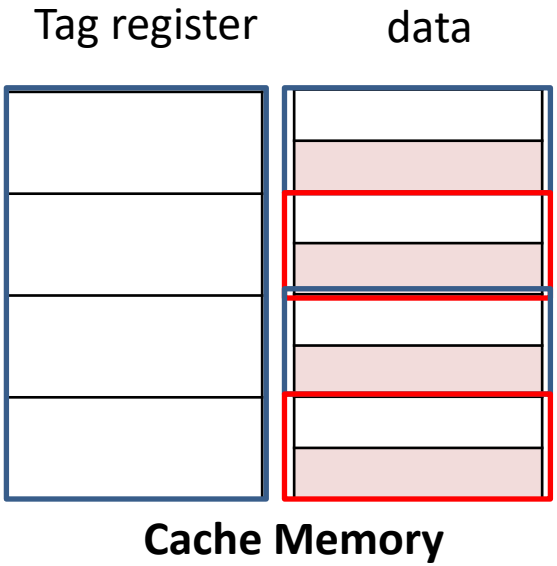
- Processor can access the data/instruction by its **address in main memory**

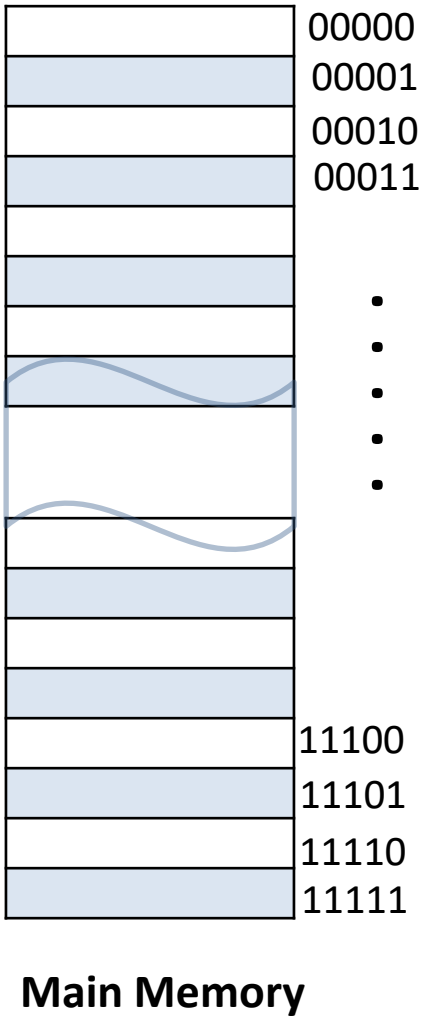
- But, data in cache **DOES NOT** have an address!

- How to map the address of memory to some data in cache?
- Mapping Schemes
  1. Direct Mapping
  2. Associative Mapping
  3. Set-Associative Mapping

Direct-Mapping

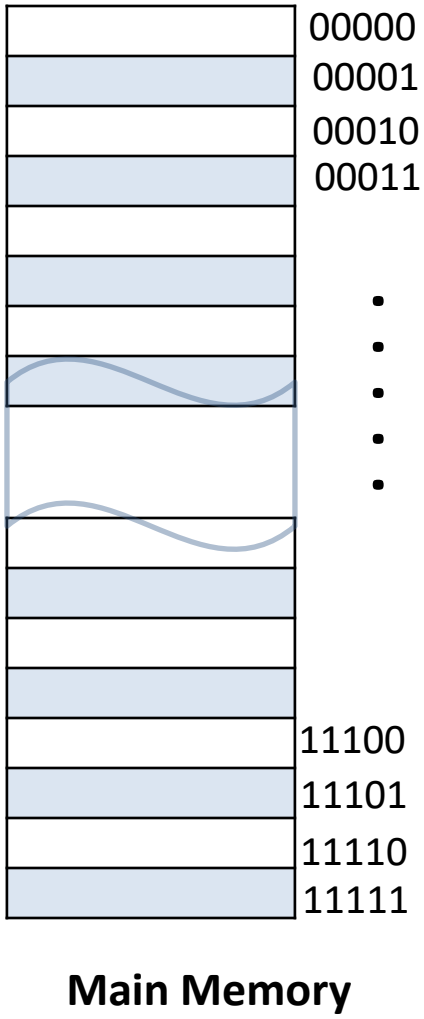
- Suppose a main **memory system** has **32-byte capacity**
- The system is **byte-addressable**
- A **direct-mapped cache** has **8-byte capacity**
- A **block** (refill line) has **2 bytes**





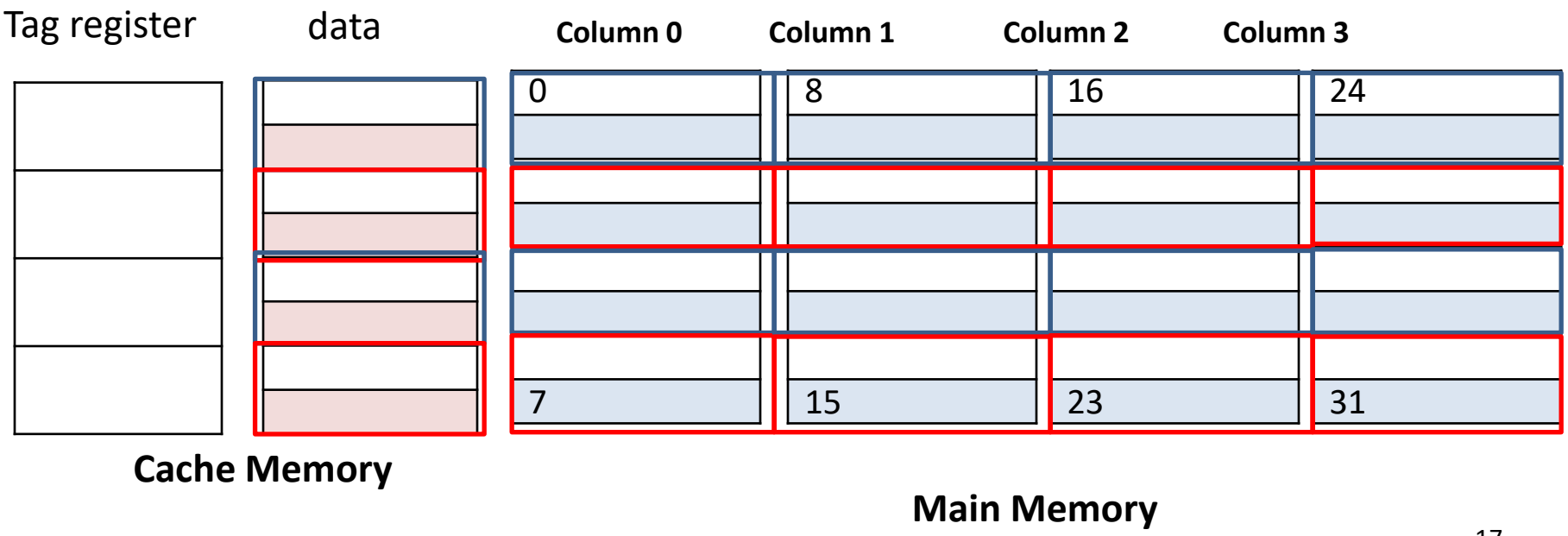
Direct-Mapping

- Rearrange main memory as a number of columns (pages) so that **each column (page) has the same size as the cache**
- How many columns you can have when memory size is 32 bytes, cache is 8 bytes?



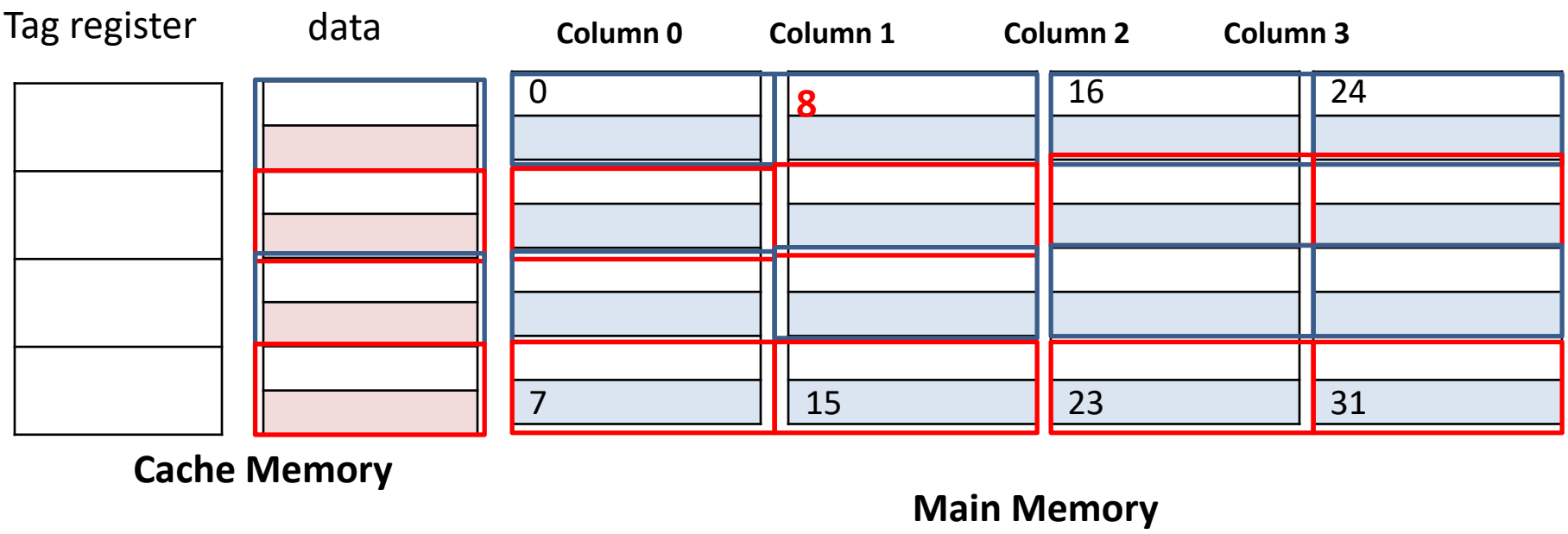
Direct-Mapping

- Rearrange main memory as a number of columns (pages) so that **each column (page) has the same size as the cache**
- Main memory and caches are divided into equally sized blocks (refill lines)
  - Block (Line) size: Typically between 4 and 64 bytes long (**must be power of 2**)



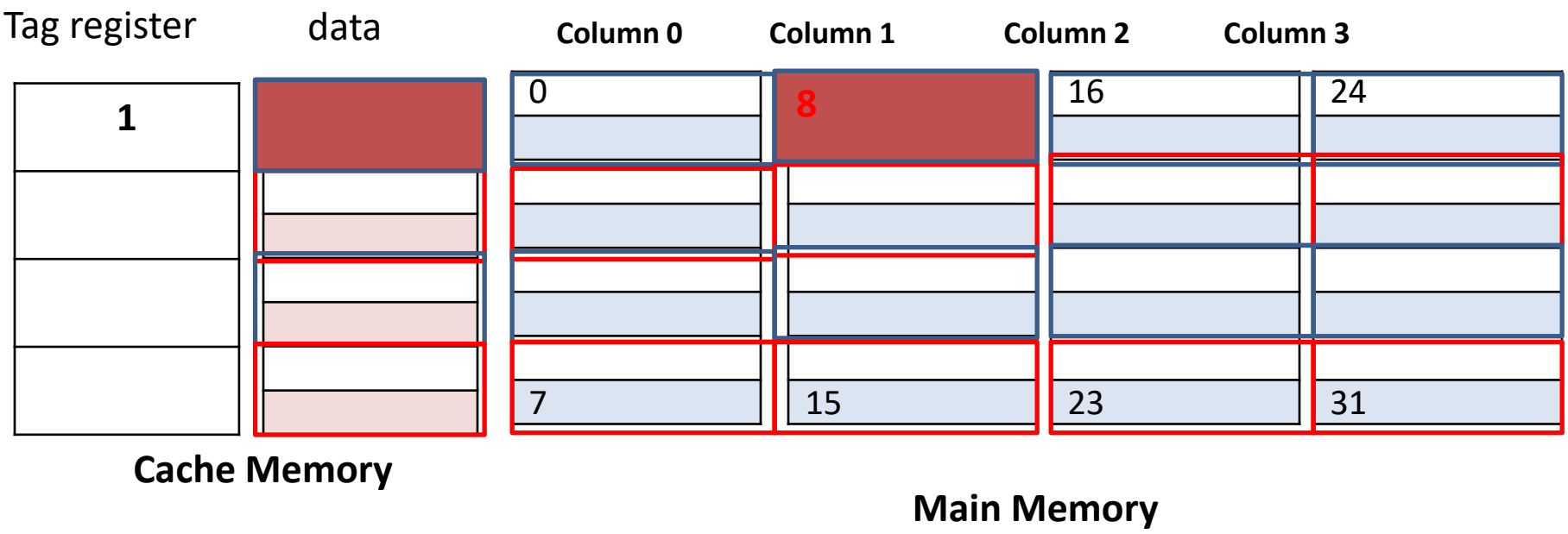
Direct-Mapping

- Want to read data at address **8 (01000_2)** in main memory into cache
- Map the **0th block** (line) in main memory in any column (page) to the 0th block in cache



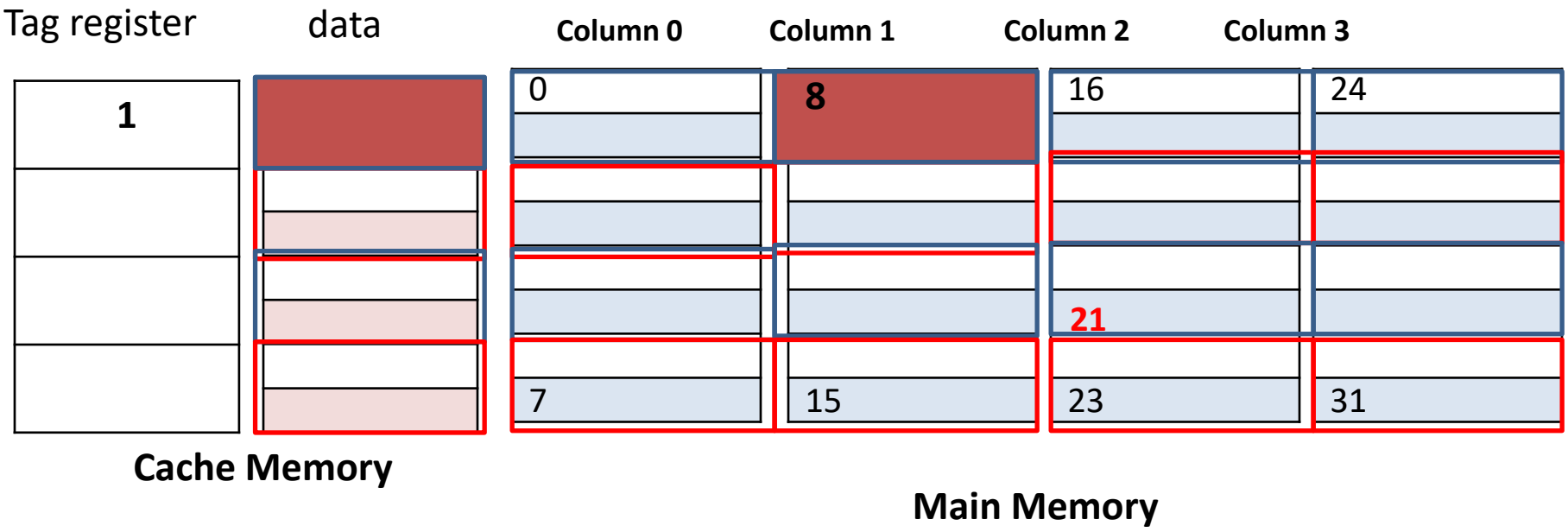
Direct-Mapping

- Want to read data at an address **8 (01000_2)** in main memory into cache
- Map the **0^{th} block** (line) in main memory in any column (page) to the 0^{th} block in cache and update tag



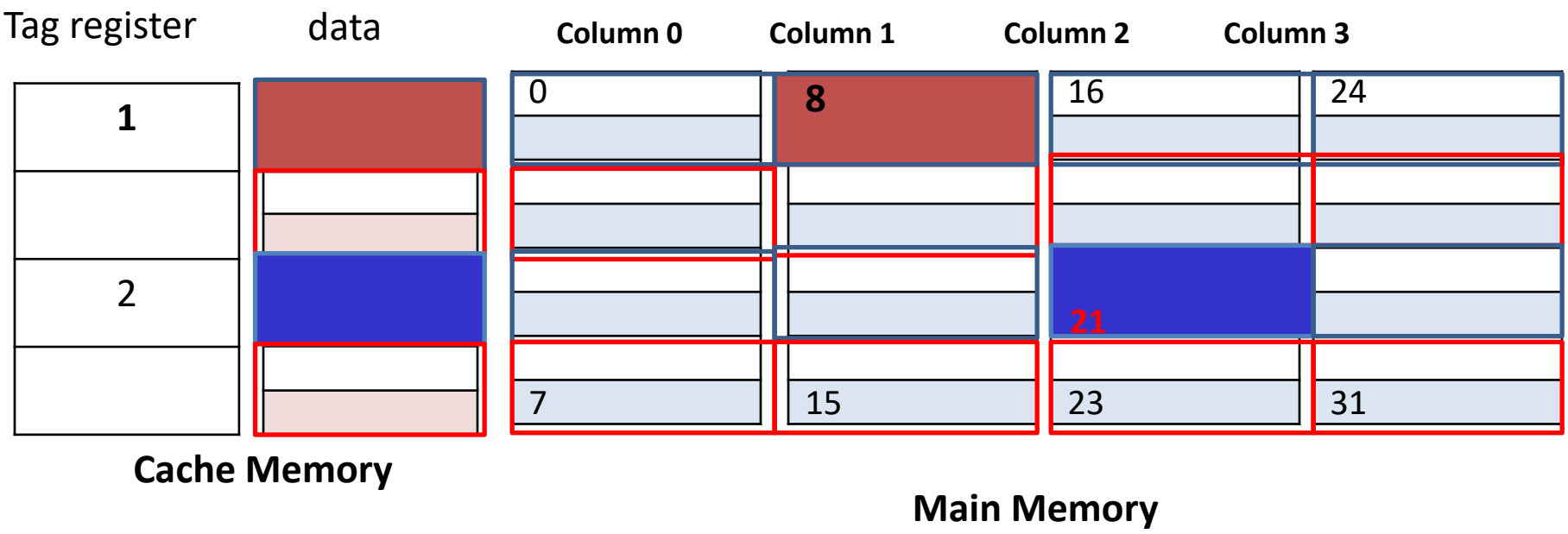
Direct-Mapping

- Want to read data at an address **21 (10101_2)** in main memory into cache
- Map the **2nd block** (line) in main memory in any column (page) to the 2nd block in cache



Direct-Mapping

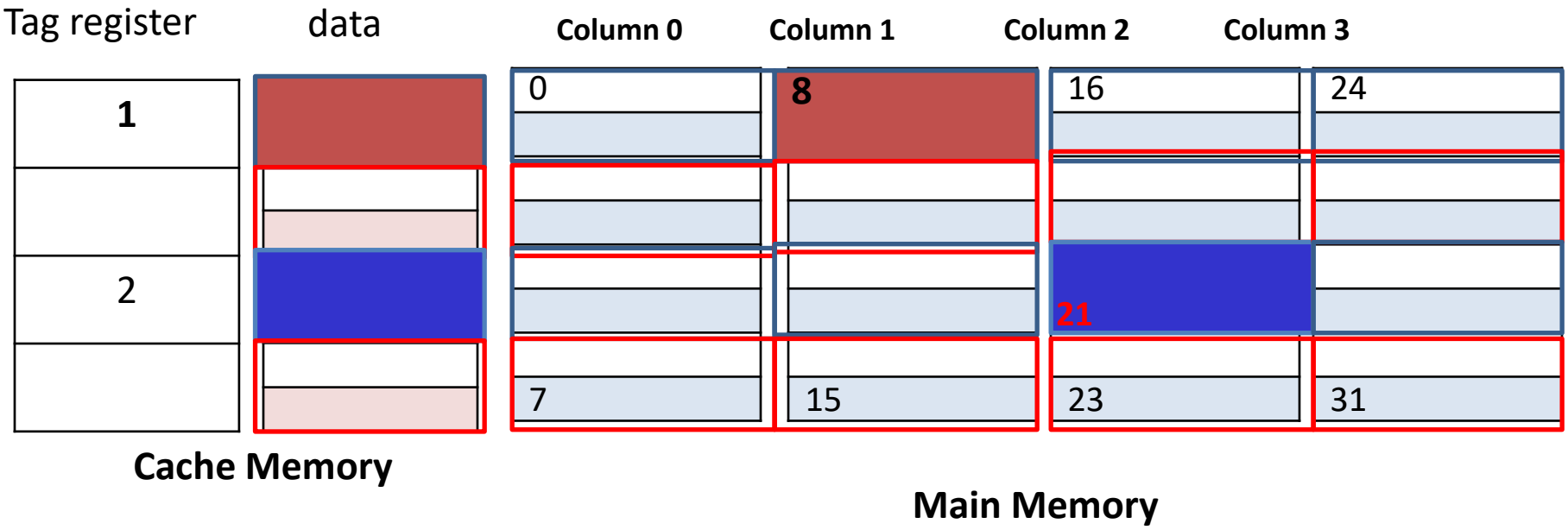
- Want to read data at an address **21 (10101_2)** in main memory into cache
- Map the **2nd block** (line) in main memory in any column (page) to the 2nd block in cache and update tag



Direct-Mapping

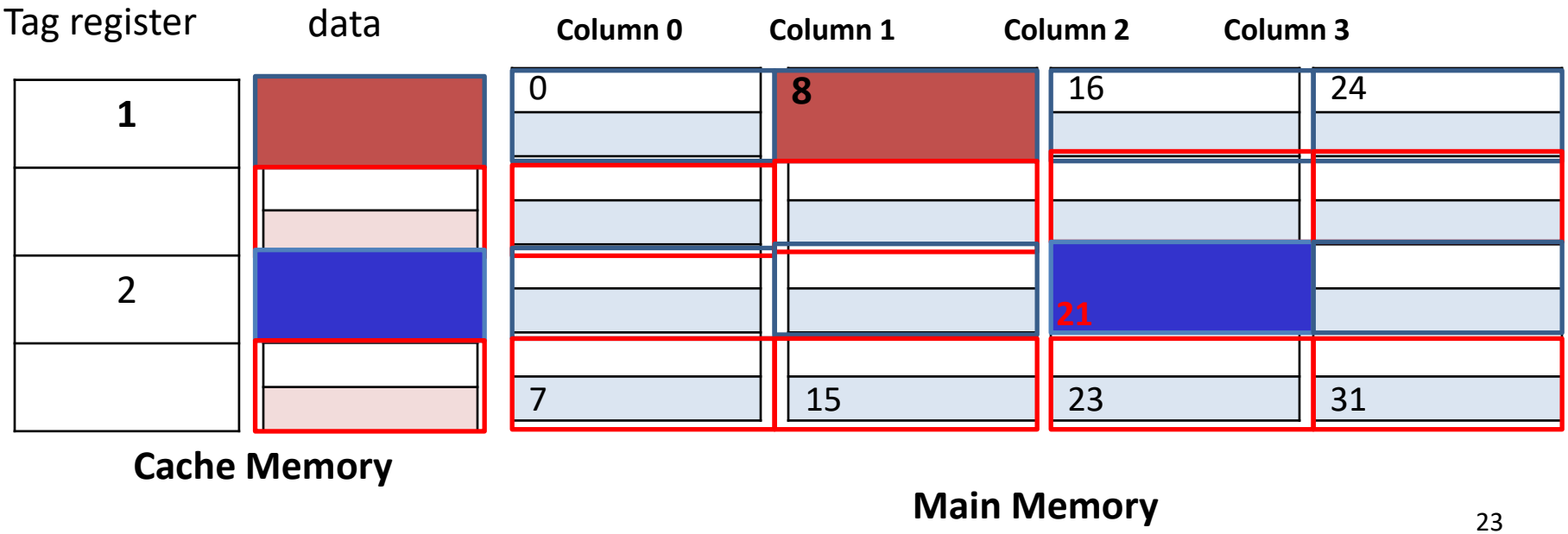
- Mapping → map to the same block location
- **What value is in the tag register?**

  The column (page) number
- How to calculate the tag value from an address?
- Address 8 → tag 1 and Address 21 → tag 2



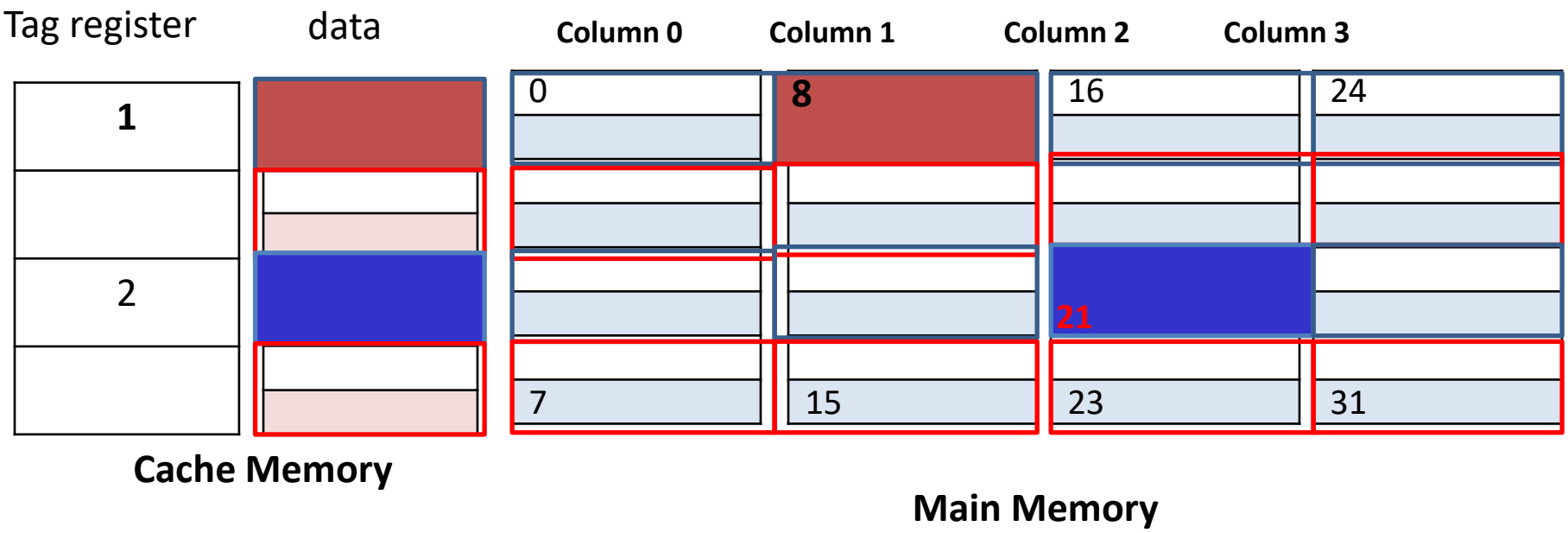
Direct-Mapping

- Out of a 5-bit address, how many bits are for column number?
- The first two bits (since there are 4 pages)
  - 8 = **01**000 → tag is 01_2
  - 21 = **10**101$_2$ → tag is 10_2
- What are the other bits?



Direct-Mapping

- Out of a 5-bit address, the first two bits (since there are 4 pages)
- What are the other bits?
- 8 = **01 00 0_2** → **column 1**, **0^{th} block**, **0^{th} in the block**
- 21 = **10 10 1_2** → **column 2**, **2^{nd} block**, **1^{st} in the block**



Direct Mapping

Divide the address into **tag**, **row** and **offset bits**

- Memory size: **L** bytes
- Direct-mapped Cache size: **M** bytes
- The Cache block size: **K** bytes

➔ # Columns in a main memory: **L/M** → **# tag bit** is $\log_2(L/M)$

➔ # Blocks in a cache memory: **M/K** → **# row number bit** is $\log_2(M/K)$

➔ block size in a main memory: **K** → **offset bit** is $\log_2 K$

Tag	Row/line	Offset

Direct Mapping - Example

Divide the address into **tag**, **row** and **offset** bits

- Memory size: **32** bytes
- Direct-mapped Cache size: **8** bytes
- The Cache block size: **2** bytes

➔ # Columns in a main memory: 32/8 = 4 → # tag bit is ?

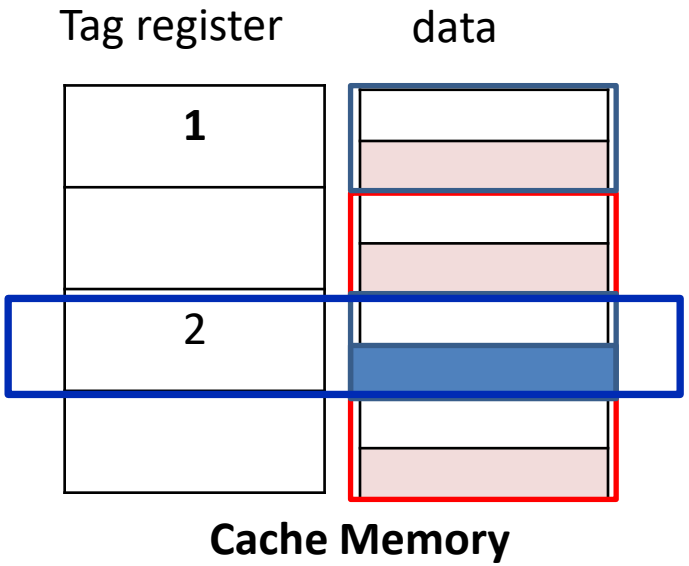
➔ # Blocks in a main memory: 8/2 = 4 → # row number bit is ?

➔ block size in a main memory: 2 → offset bit is ?

Tag (2)	Row/line (2)	Offset (1)

Find Hit/Miss in a Direct-Mapped Cache

1. Given an address (21 = 10101_2)
   1. get tag/block (10_2), row (10_2) and offset (1_2)
2. Go to the same row (block: 10_2) in a cache, check the valid tag value in the tag register.
3. If the valid tag matches, then hit; otherwise miss.
4. When hit, then find the content in that offset location (1_2) in that block



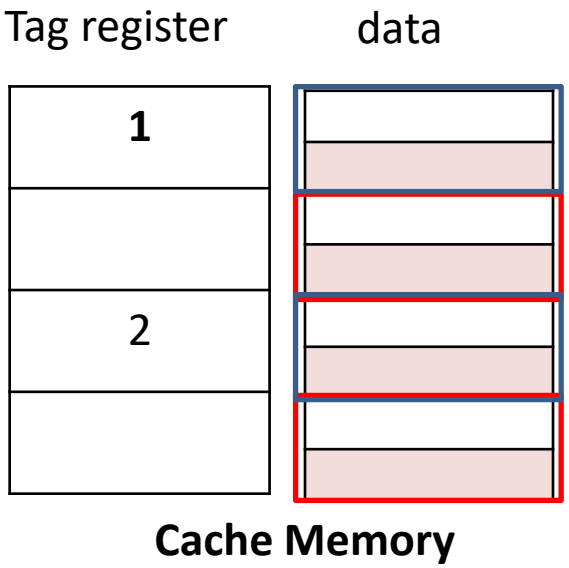
Find Hit/Miss in a Direct-Mapped Cache

Suppose that the length of address is 5 bits:

2-bit tag, 2-bit row, and 1-bit offset

From the following cache image, see if the followings are hit or miss

1) Address 9
2) Address 17
3) Address 29



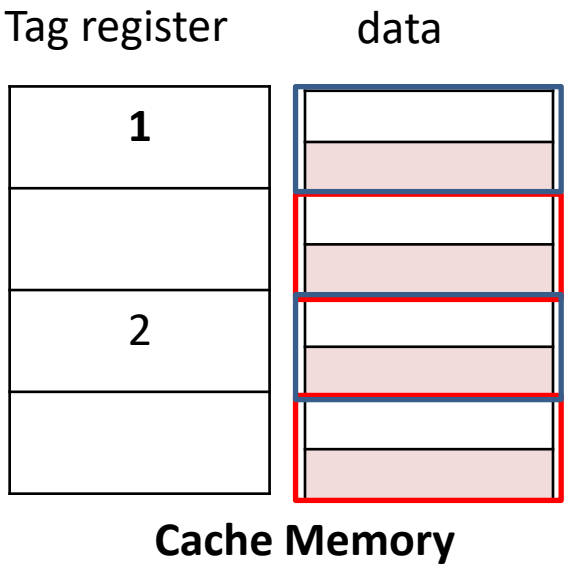
Find Hit/Miss in a Direct-Mapped Cache

Suppose that the length of address is 5 bits:

2-bit tag, 2-bit row, and 1-bit offset

From the following cache image, see if the followings are hit or miss

1) Address 9 = 01 00 1: block is 0, tag is 1 (hit)
2) Address 17 = 10 00 1: block is 0, tag is 2 (miss: not matches 1)
3) Address 29 = 11 10 1: block is 2, tag is 3 (miss: not matches 2)



Class Exercise: Direct-Mapped Cache

- Main memory: 2^{14} bytes
- Cache: direct-mapped
  - 16 blocks (refill lines)
  - Each block has 8 bytes

Q1) What is the size of cache?

Now we will divide the address bits:

Q2) How many bits for the offset (where is the data within a block)?

Q3) How many bits for line number (what is the block number)?

Q4) How many bits for tag (What is the number of pages in a main memory)?

Class Exercise: Direct-Mapped Cache

- Main memory: 2^{14} bytes
- Cache: direct-mapped
  - 16 blocks (refill lines)
  - Each block has 8 bytes

Q1) What is the size of cache?

16 blocks X 8 bytes = 2^7 bytes

Q2) How many bits for the offset?

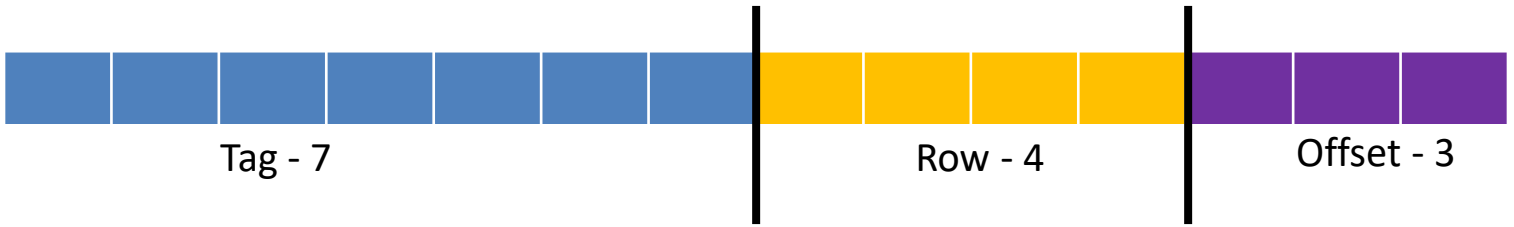
Each block has 8 bytes. So, 3 bits for the offset.

Q3) How many bits for block (row/line) number?

Each page has 16 blocks (same as the cache). So, 4 bits for the block number. (Or the row/line number)

Q4) How many bits for tag (What is the number of pages in a main memory)?

Each page should be the same size with cache. 14 – 3 – 4 = 7 bits for tag



Limitation of Direct-Mapped Cache

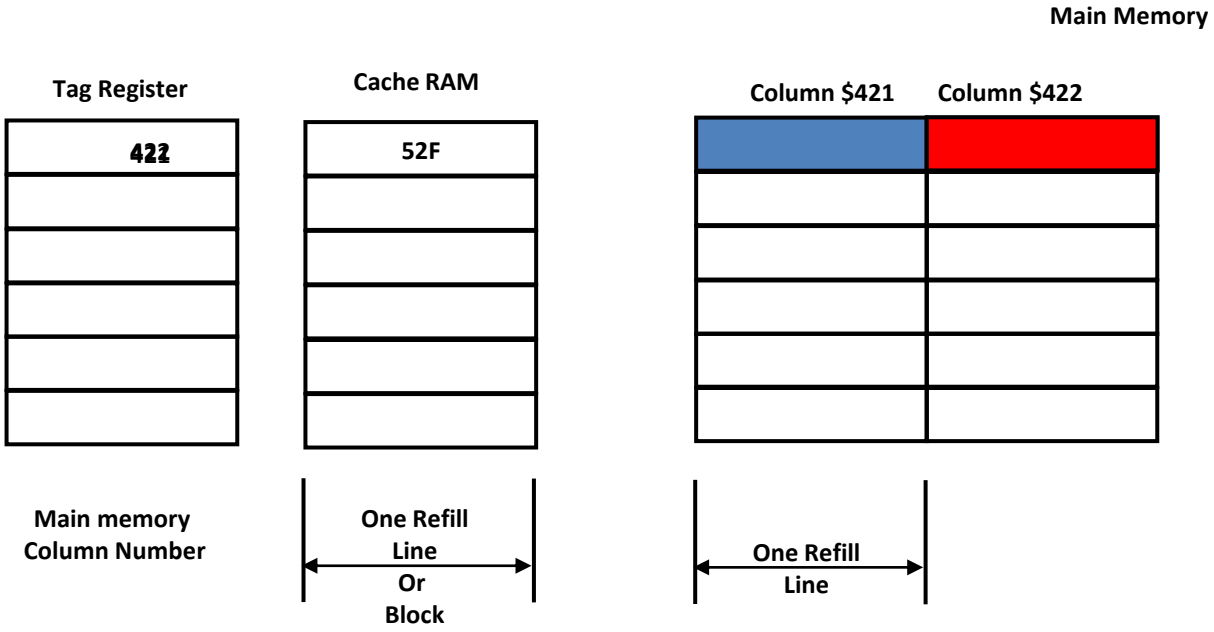
- What if you repeatedly access the data of the same block but from different pages? (swapping the blocks causes lots of miss penalty)
- Example

```
Loop:       JSR subroutine ($10854BCA)
            {…}
            BNE loop
Subroutine: {…}     ($10894BC0)
            RTS
```

  - $10854BCA
    - Offset: 0xx0A, **Row = 0x52F**, Column (page) = 0x0421
  - $ 10894BC0
    - Offset: 0xx00, **Row = 0x52F**, Column (page) = 0x0422

Limitation of Direct-Mapped Cache

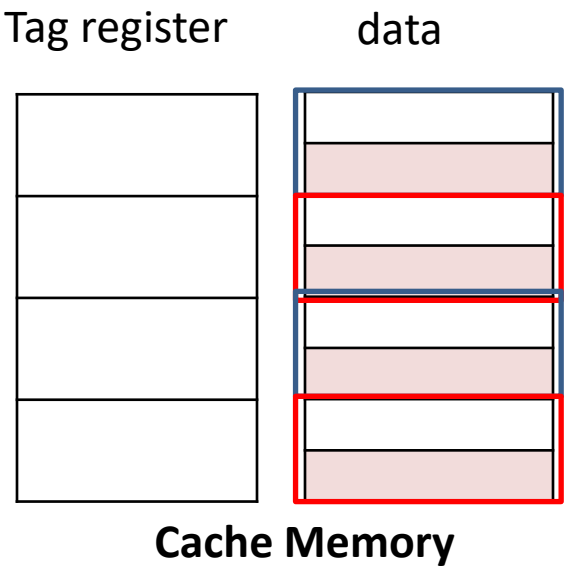
- $10854BCA
  - Offset: 0xx0A, **Row = 0x52F**, Column (page) = 0x0421
- $ 10894BC0
  - Offset: 0xx00, **Row = 0x52F**, Column (page) = 0x0422

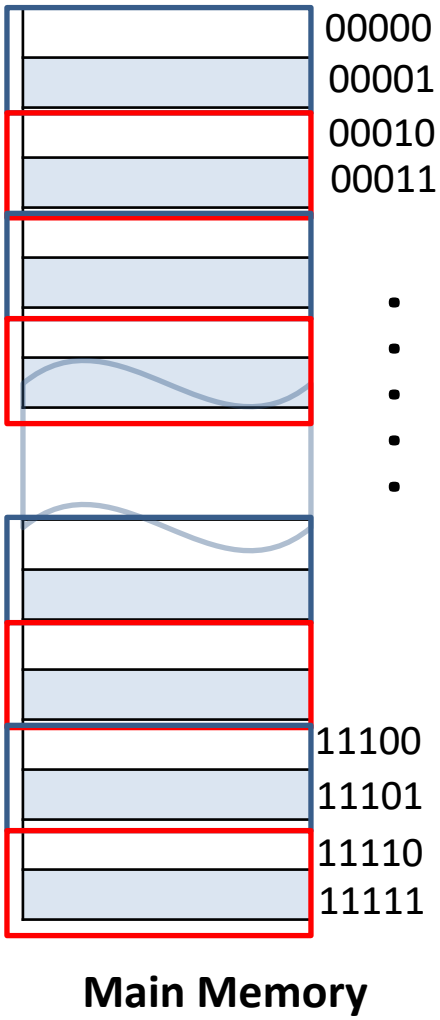


Associative Cache

Associative Cache

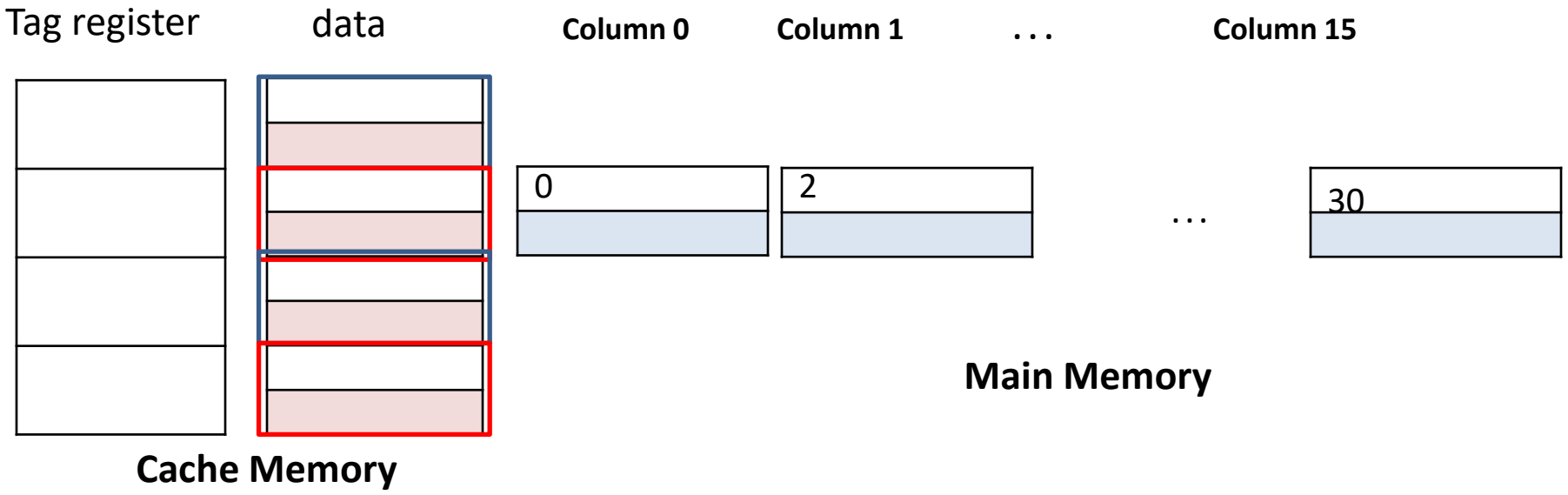
- Rearrange the main memory based on the size of a block (refill line size)
- **number of columns = number of blocks**





Associative Cache

- Rearrange the main memory based on the size of a block (refill line size)
- Each column is one block
  - **number of columns = number of blocks**
- Each block can be mapped to any available block

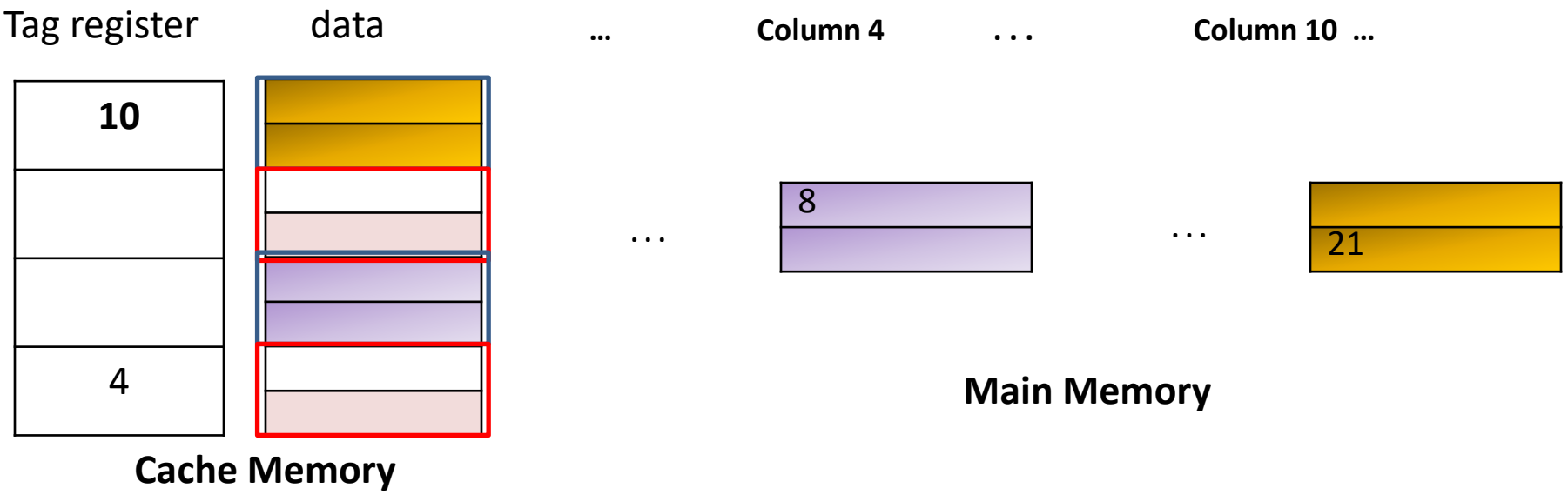


Associative Cache

- Data at address 8 will be at the column 4: 8 = 0100 0_2
- Data at address 21 will be at the column 10: 21 = 1010 1_2

Tag (4)	Offset (1)

- Map any main memory block to any available cache block



Associative Cache

Divide the address into tag and offset (**No ROW number!!**)

- Memory size: **L** bytes
- ***Associative*** Cache size: **M** bytes
- Cache block size: **K** bytes

➔ # Columns in a main memory: **L/K** → # tag bit is $\log_2(L/K)$

➔ Block size in a main memory: **K** → offset bit is $\log_2(K)$

Associative Cache - Example

Divide the address into tag and offset (**No ROW number!!**)

- Memory size: 32 bytes
- ***Associative*** Cache size: 8 bytes
- Cache block size: 2 byte

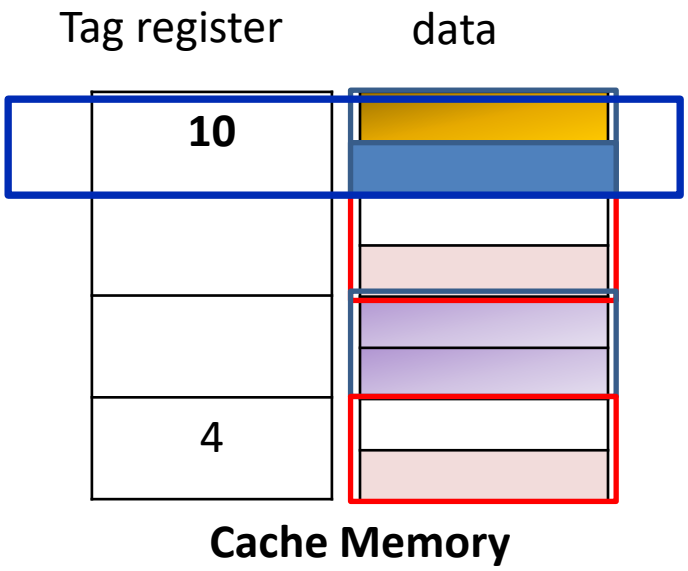
➔ # Columns in a main memory: 32/2 = 16 ➔ # tag bit is ?

➔ block size in a main memory: 2 ➔ offset bit is?

Tag (4)	Offset (1)

Find Hit/Miss in an Associative Cache

1. Given a memory address (21), get tag (1010_2=10) and offset (1)
2. *Search* the Tag register to find the block whose tag matches (1010_2=10)
3. If found the matching tag, then hit; otherwise miss.
4. When hit, then find the content in the cache block which is at the offset location



Class Exercise: Associative Cache

- Main memory: 2^{14} bytes
- Cache: associative cache
  - 16 blocks (refill lines)
  - Each block has 8 bytes

Q1) What is the size of cache?

Q2) How many bits for the offset? (where the data within in a block)

Q3) How many bits for block (line) number?

Q4) How many bits for tag (What is the number of pages in a main memory)?

Class Exercise: Associative Cache

- Main memory: 2^{14} bytes
- Cache: associative cache
  - 16 blocks (refill lines)
  - Each block has 8 bytes

Q1) What is the size of cache?

16 blocks X 8 bytes = 2^7 bytes

Q2) How many bits for the offset? (where the data within in a block)

Each block has 8 bytes. So, 3 bits for the offset

Q3) How many bits for block (line) number?

No block number. Any main memory block can be mapped to any cache block.

Q4) How many bits for tag (What is the number of pages in a main memory)?

bits for tag = 14 - 3 = 11 bits.

Conceptually, each block can be a page.

Limitation of Associative Cache

1. Should search all the blocks (rows) in a cache to find the matching tag (no block number information embedded in the address)
   - Complex hardware is needed for address comparisons if main memory is large

2. When cache is full, what should be removed (victim block) when a new block should be loaded into cache?
   - Need replacement algorithm (will cover later)

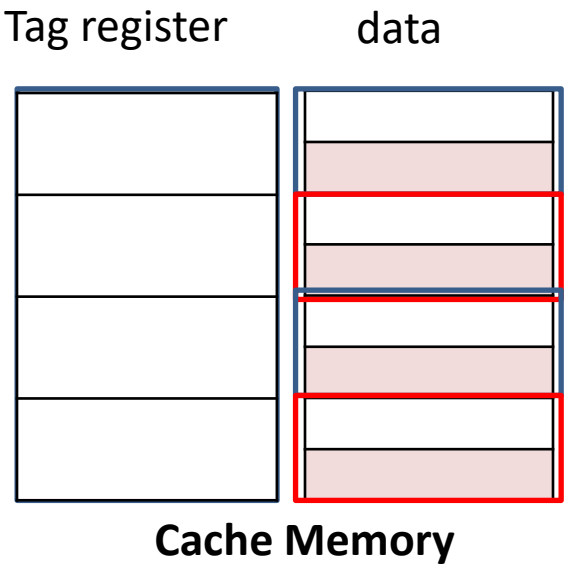
Set-Associative Cache

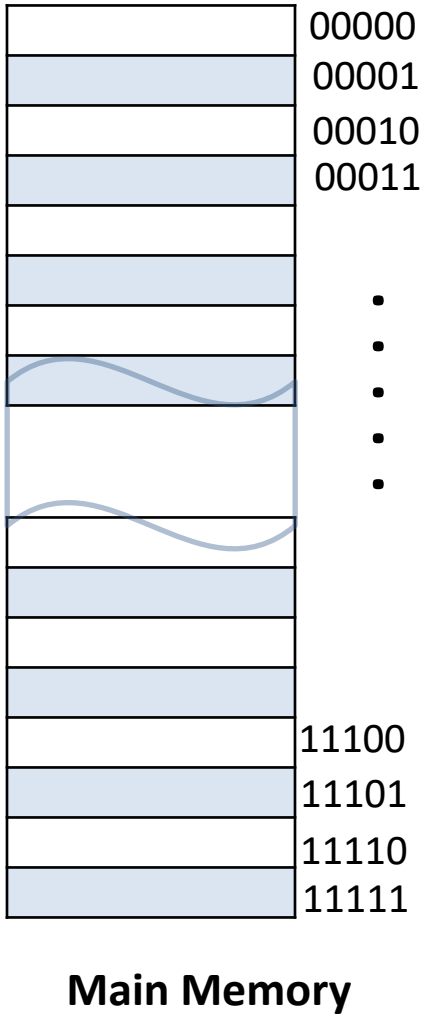
Set-Associative Cache

- Also called **N-way set-associative cache**
- **Combines the properties of the direct-mapped and associative cache** into one system
  - Direct-mapped Cache is very restrictive
  - Associative Cache is very expensive in terms of search
  - So, **combine those two → Set-Associative Cache**
  - Most commonly used in modern processors (4-way set-associative cache)
- **N-way set-associative**
  - Divide the cache into **multiple sets**
  - **N is the number of blocks in a set**
  - **!!! N is not the number of sets !!!**

2-way Set-Associative Cache

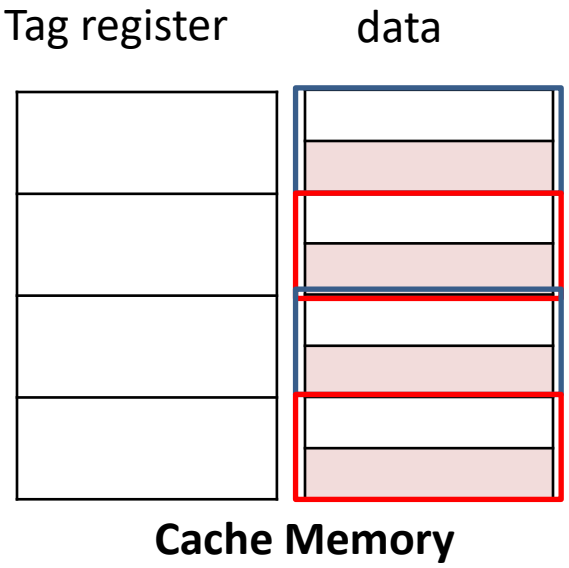
- Suppose a byte-addressable main memory system is 32-byte capacity
- A 2-way set-associative cache has 8-byte capacity
- A block has 2 bytes

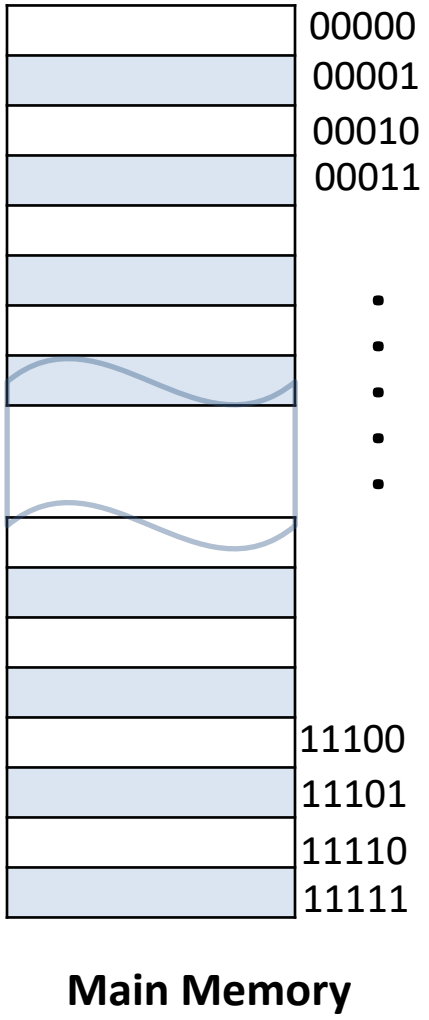




2-way Set-Associative Cache

- Rearrange main memory as a number of columns (pages) so that the **number of blocks in each column (page)** is **the same as** the **number of cache sets**

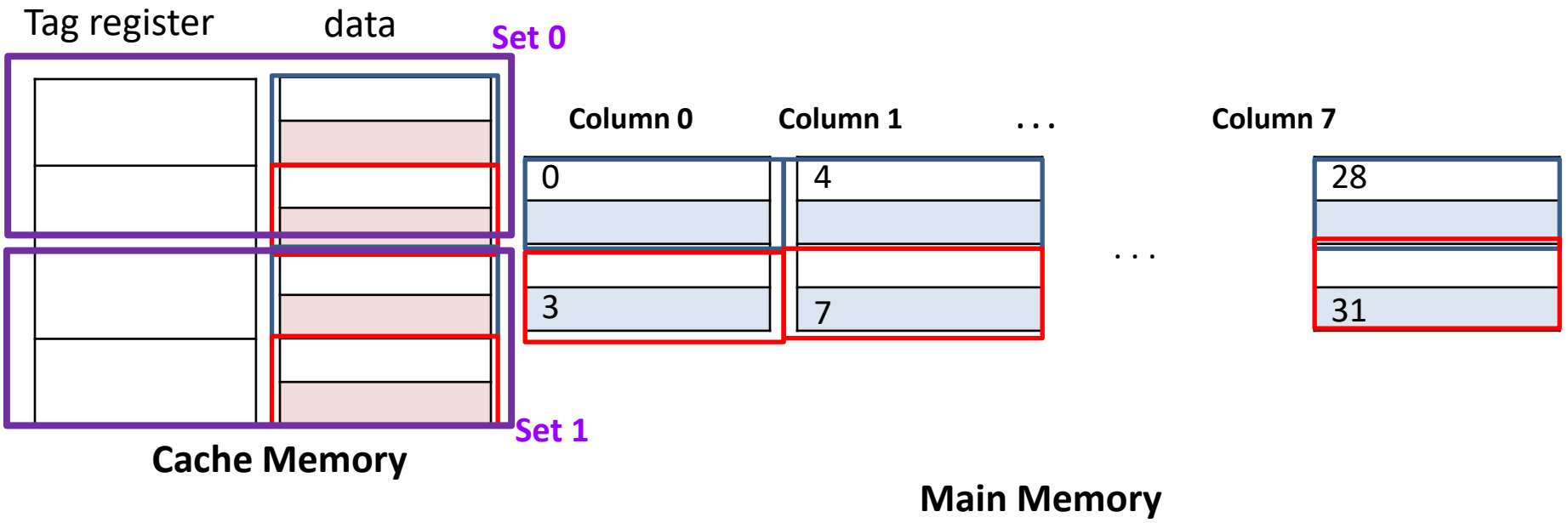




2-way Set-Associative Cache

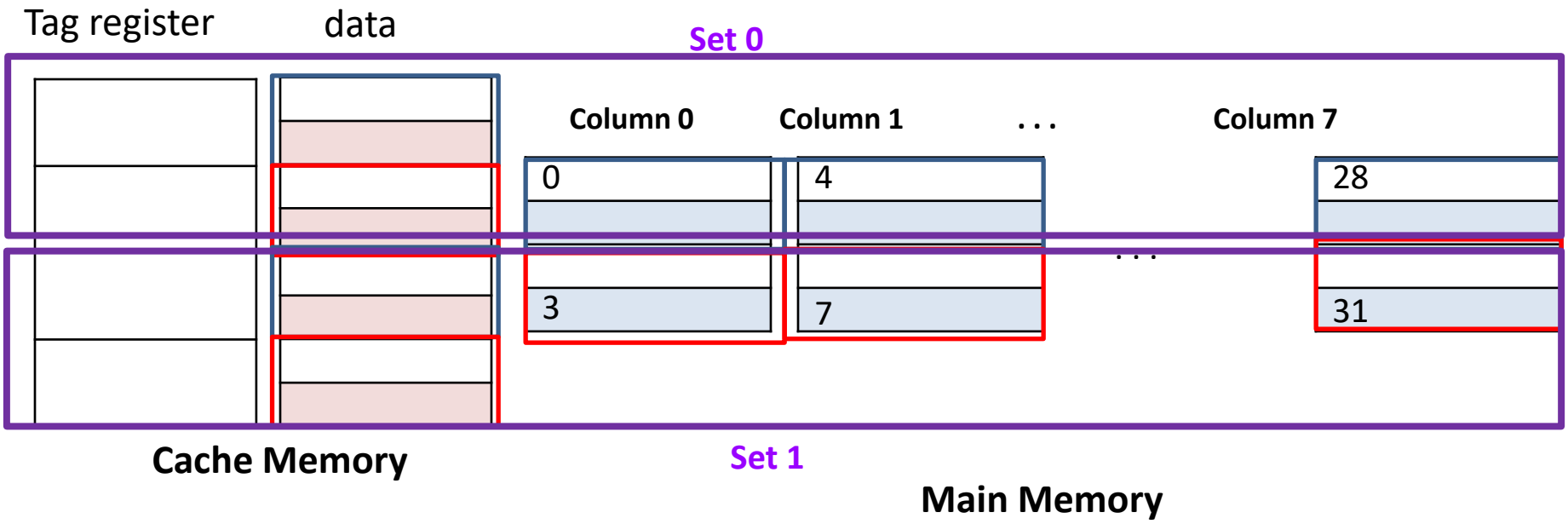
- Rearrange main memory so that 2 blocks in a cache map to each block in a column

- **Two blocks** in the cache is in the same set



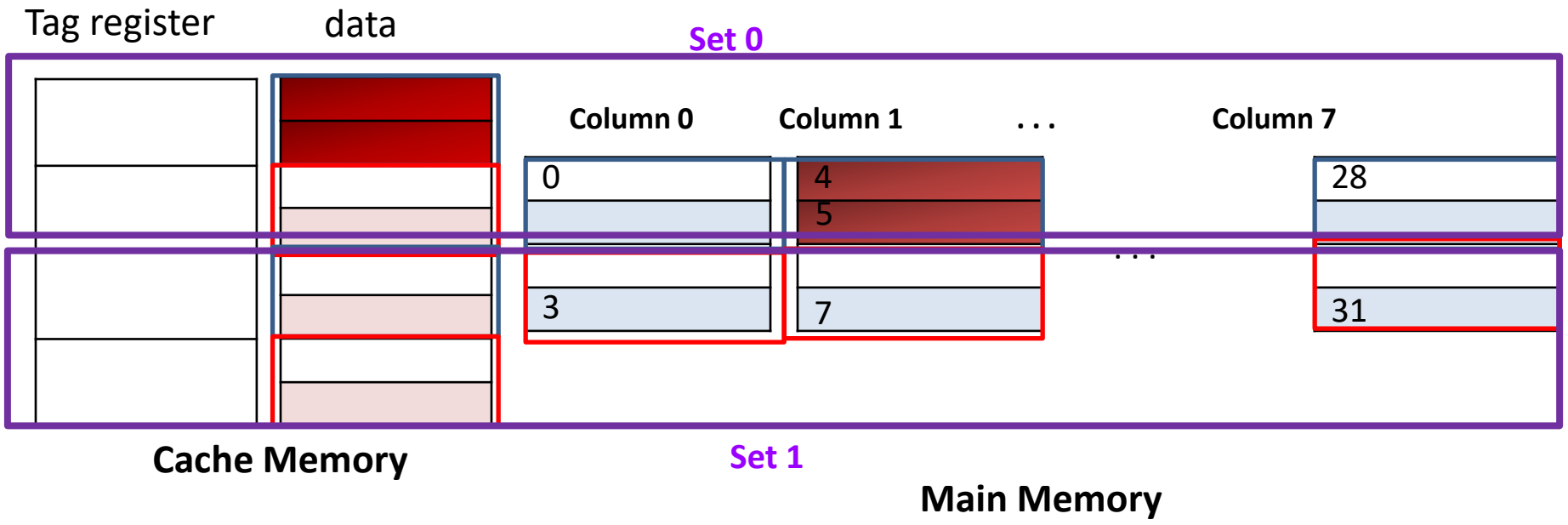
2-way Set-Associative Cache

- Want to read data at address **5(00101_2)** in main memory into a cache



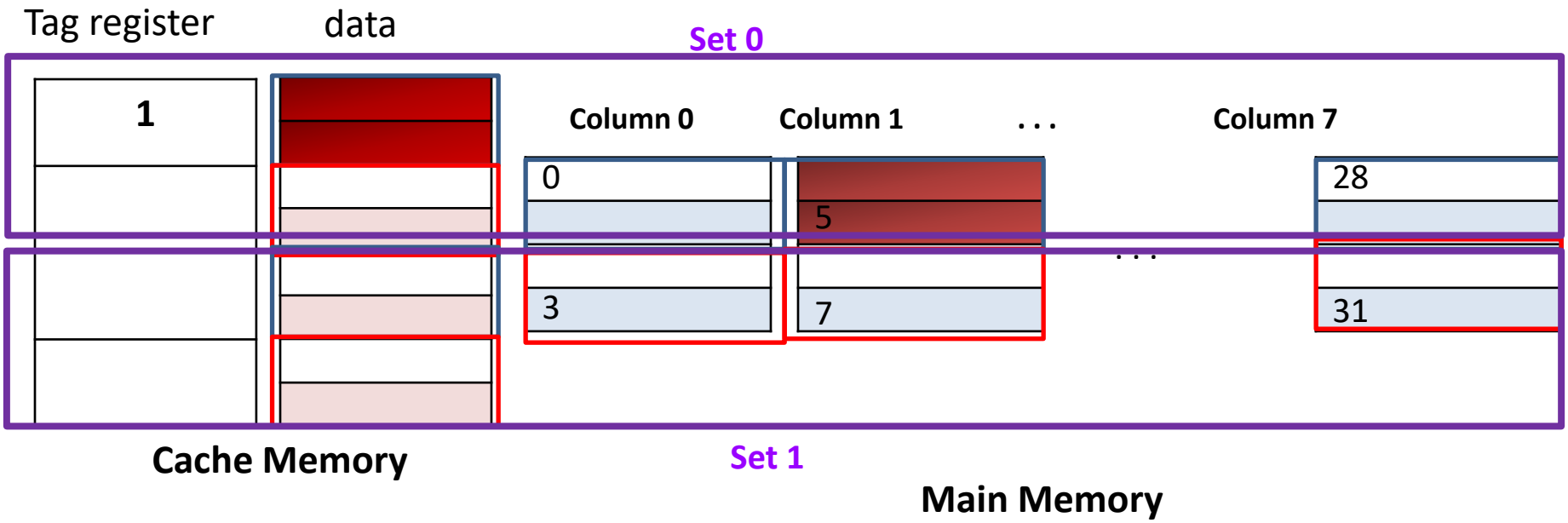
2-way Set-Associative Cache

- Want to read data at address **5 (00101_2)** in main memory into a cache
- Map the **0^{th} block** in main memory to any available block in the **0^{th} set of** cache



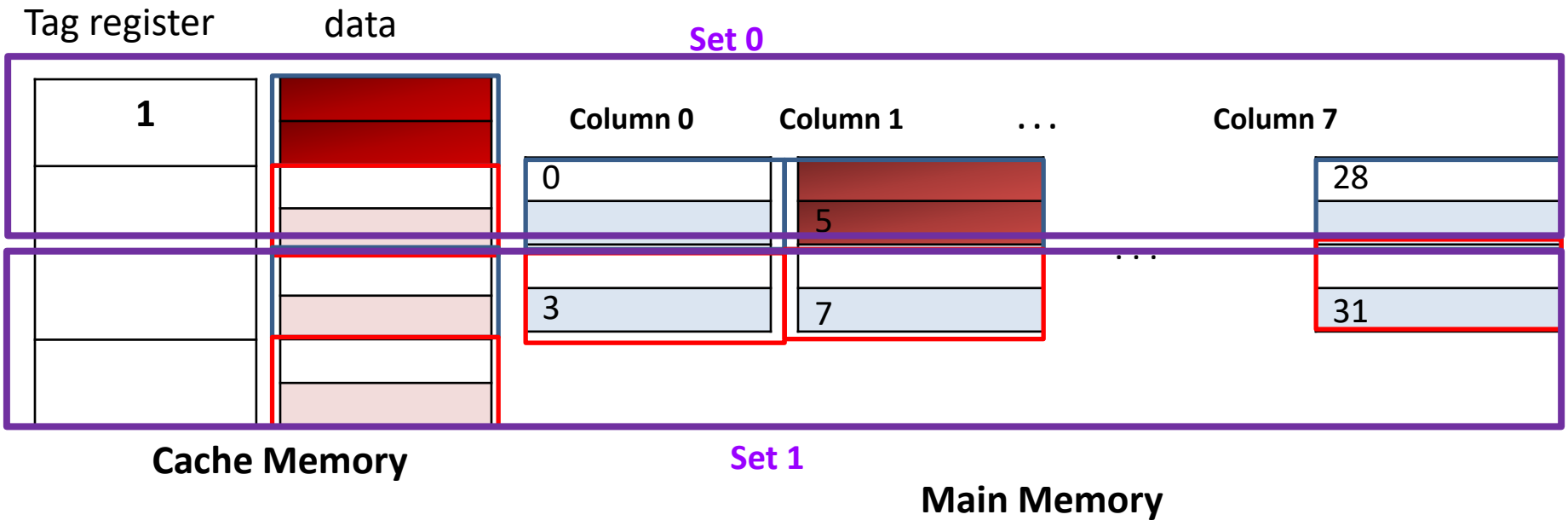
2-way Set-Associative Cache

- Want to read data at address **5(00101_2)** in main memory into a cache
- Map the **0^{th} block** in main memory to any available block in the **0^{th} set of** cache **and update tag register**



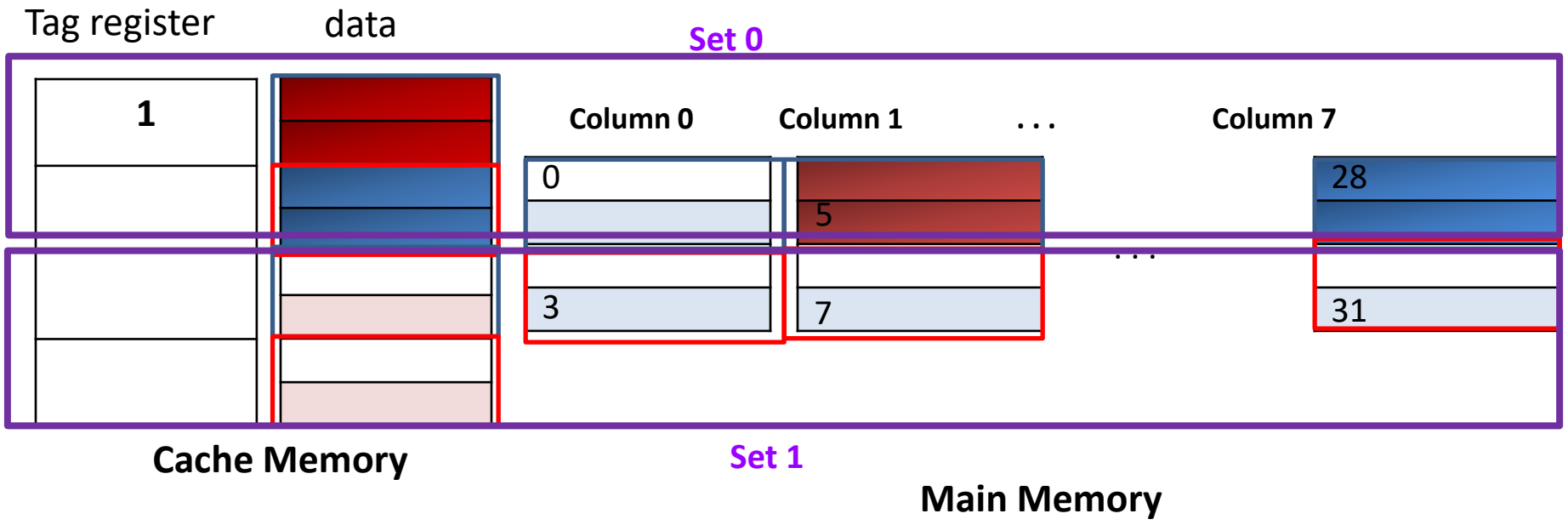
2-way Set-Associative Cache

- Want to read data at address **28 (11100_2)** in main memory into a cache



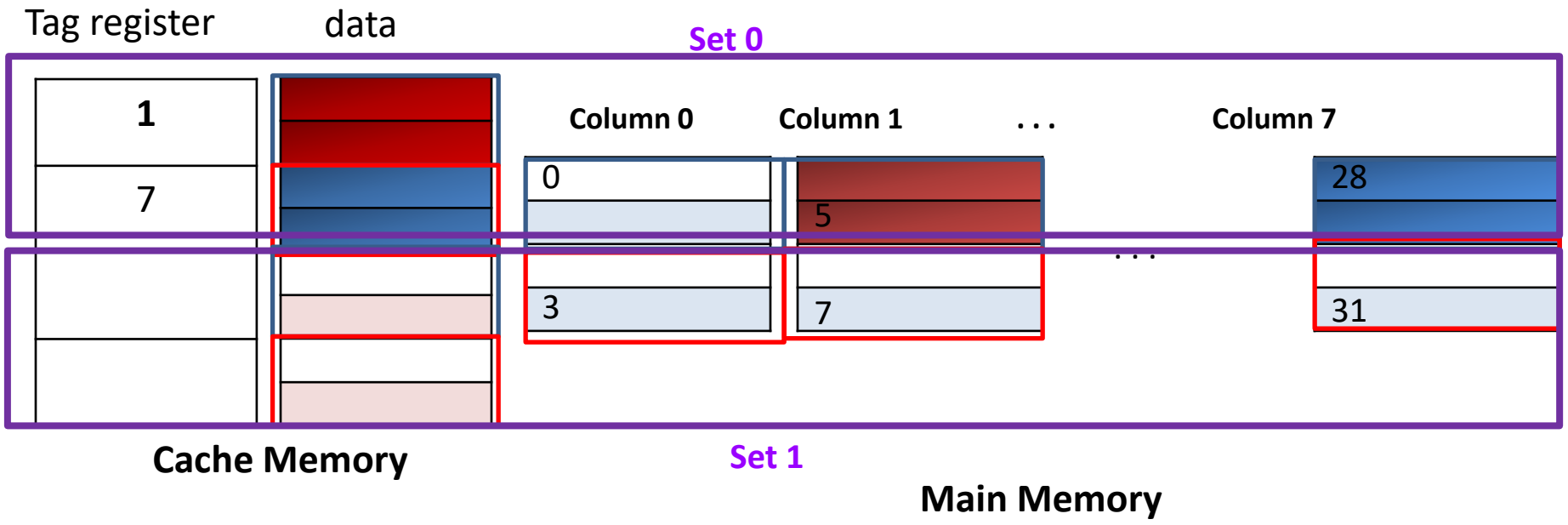
2-way Set-Associative Cache

- Want to read data at address **28 (11100_2)** in main memory into a cache
- Map the **0^{th} block** in main memory to any available block in the **0^{th} set of** cache



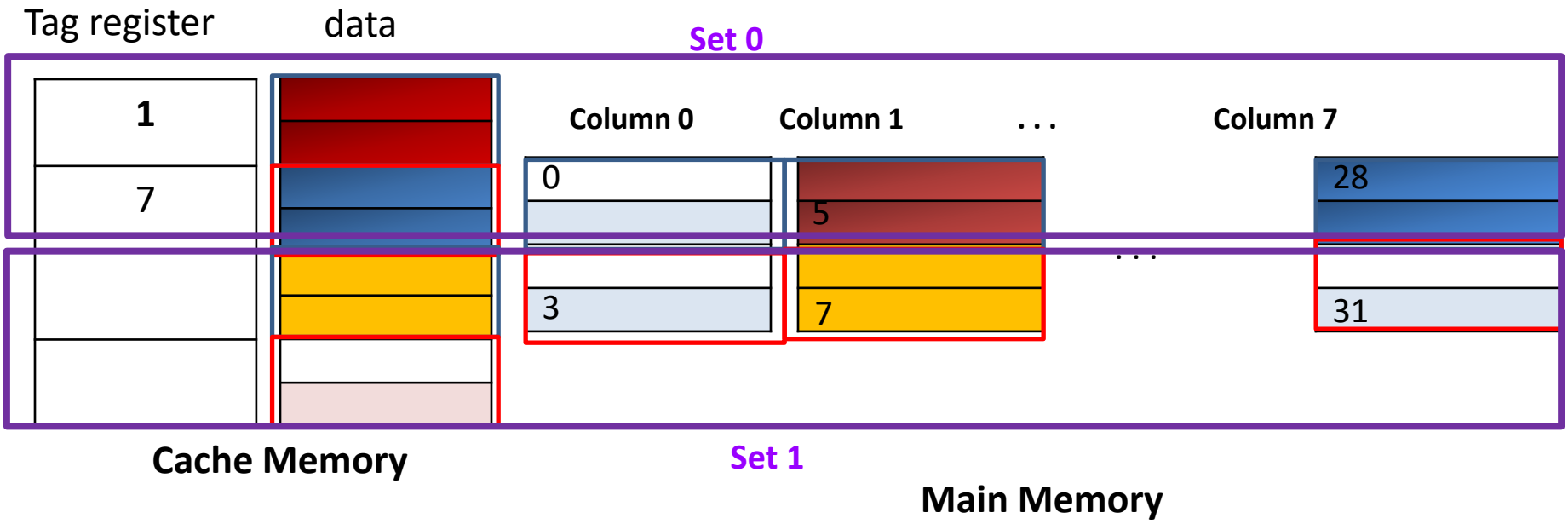
2-way Set-Associative Cache

- Want to read data at address **7 (00111_2)** in main memory into a cache



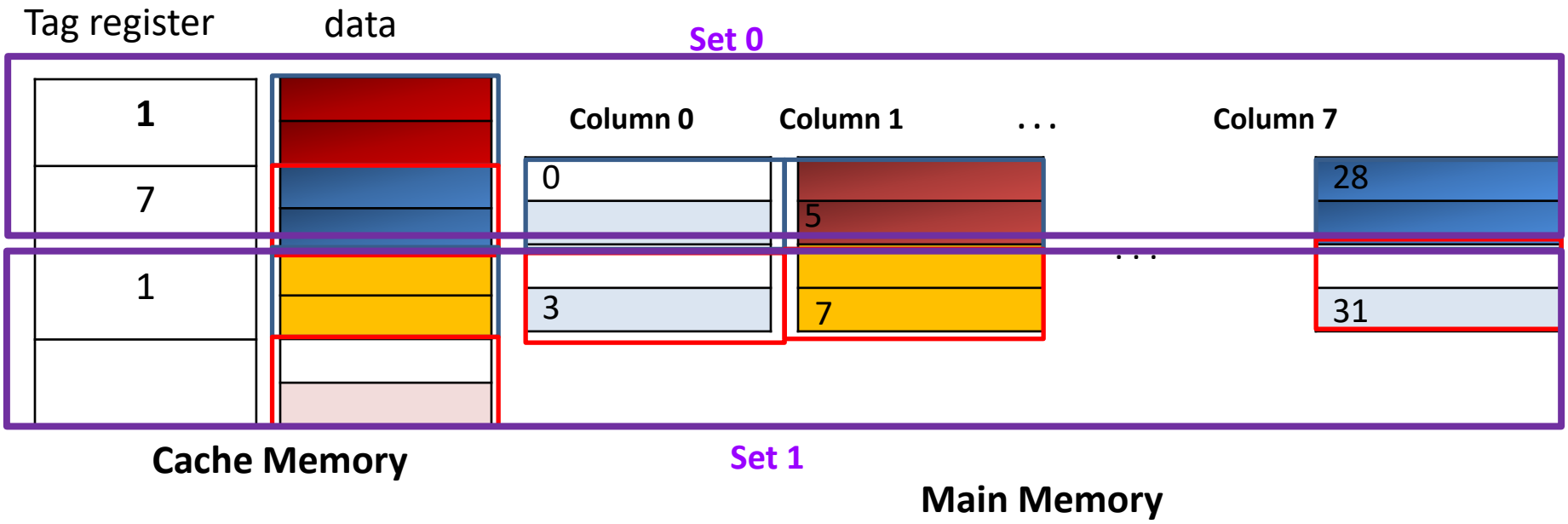
2-way Set-Associative Cache

- Want to read data at address **7 (00111_2)** in main memory into a cache
- Map the **1st block** in main memory to any available block in the **1st set of** cache



2-way Set-Associative Cache

- Want to read data at address **7 (00111_2)** in main memory into a cache
- Map the **1st block** in main memory in to any available block in the **1st set of** cache **and update tag register**



N-way Set-Associative Cache

Divide the address into **tag**, **set** and **offset** bits

- Memory size: **L** bytes
- ***N-Way Set-Associative*** Cache size: **M** bytes
- The Cache block size: **K** bytes

➔ Block size in a main memory: **K** → offset bit is $\log_2(K)$

➔ # of sets in a cache: **# blocks/N** = **(M/K)/N** → # set bit is $\log_2((M/K)/N)$

➔ # Columns (Tag) in a main memory: **L/((M/K)/N * K) = L/(M/N)**

→ # tag bit is $\log_2(L/(M/N))$

Tag	Set	Offset

N-way Set-Associative Cache - Example

Divide the address into tag, set and offset bits

- Memory size: 32 bytes (total address 5 bits)
- ***2-way set-associative*** Cache size: 8 bytes
- The Cache block size: 2 bytes

➔ Block size in a main memory: 2 → offset bit is?

➔ # of sets in a cache = # blocks/N = 4/2 → # set bit is ?

➔ # Columns in a main memory: 32/ (8/2)→ # tag bit is ?

Tag (3)	Set (1)	Offset (1)

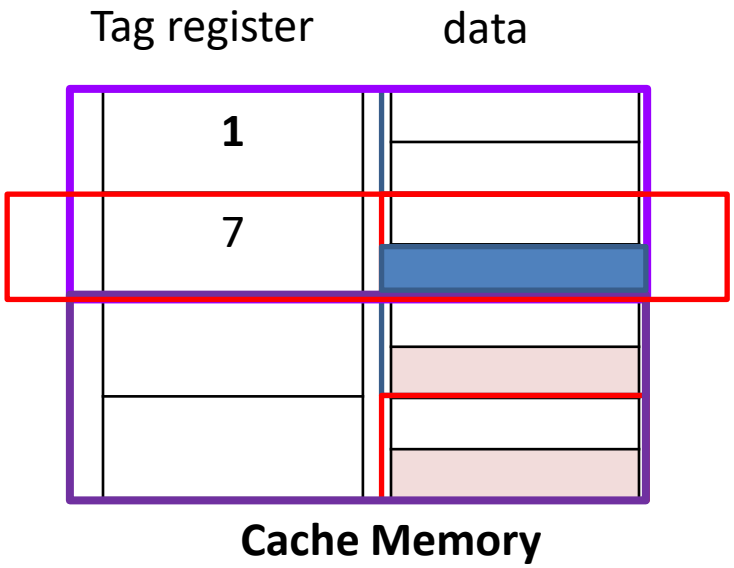
Set-Associative Cache

- Equivalent to a multiple column direct mapping
  - Example: 2-way set-associative cache has two blocks in a set

- The main memory address is partitioned into three pieces
  - Tag, ***Set***, Offset

Find Hit/Miss in a Set-Associative Cache

1. Given a memory address (29 = 11101_2), get tag (111_2), set (0) and offset (1)
2. Go to the set (set 0) in a cache, search for a block in a set whose tag is matched (111_2 = 7)
3. If the tag matches, then hit; otherwise miss.
4. When hit, then find the content in the cache block which is at the offset location



Class Exercise: Set-Associative Cache

- Main memory: 2^{14}bytes
- Cache: 2-way set-associative cache
  - 16 blocks (refill lines)
  - Each block has 8 bytes

Q1) What is the size of cache?

Q2) How many bits for the offset? (where the data within in a block)

Q3) How many bits for set?

Q4) How many bits for the tag (What is the number of pages in a main memory)?

Class Exercise: Set-Associative Cache

- Main memory: 2^{14}bytes
- Cache: 2-way set-associative cache
  - 16 blocks (refill lines)
  - Each block has 8 bytes

Q1) What is the size of cache?

16 blocks X 8 bytes = 2^7 bytes

Q2) How many bits for the offset?

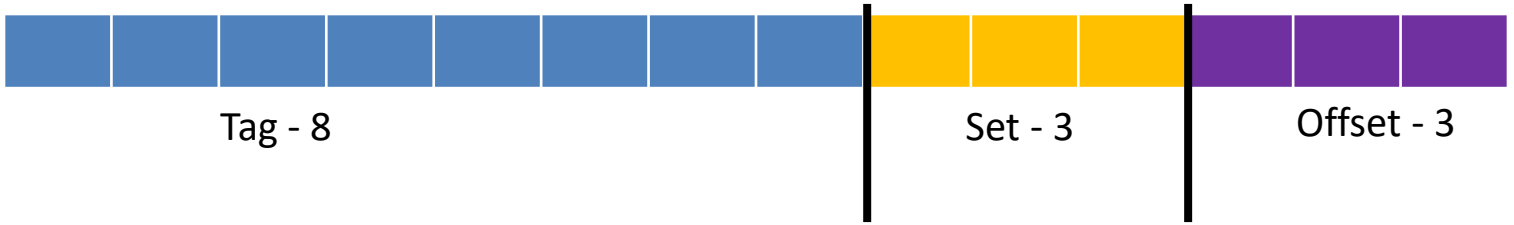
Each block has 8 bytes. So, 3 bits for the offset.

Q3) How many bits for set?

There are total 16 blocks, and each set has 2 blocks. So there are total 8 sets. So, 3-bits.

Q4) How many bits for the tag (What is the number of pages in a main memory)?

bits for tag = 14 – 3 - 3 = 8 bits.



Cache Mapping Function – Summary

- **Directed mapping**
  - The main memory address is divided into **Tag**, **Row** and **Offset** bits
  - Tag bit matches the log of the number of columns in a main memory
  - Each column has the same size with the cache
  - One to one mapping between blocks in memory and cache - restrictive

- (Fully) **Associative mapping**
  - The main memory address is divided into **Tag** and **Offset** bits
  - Conceptually each block is one column
  - Any block from main memory can be mapped any available cache block
  - Expensive as search is needed to find matching tag

- **N-way set-associative mapping**
  - The main memory address is divided into **Tag**, **Set** and **Offset** bits
  - Blocks are mapped within a set

Class Exercise

A certain processor has an on-chip cache with the following specifications:

- 32-bit wide address and data busses
- On-chip instruction cache is 64K bytes, organized as a 4-way set-associative
- Cache block size (refill line) = 64 bytes
- Average cache hit rate = 95%
- Instructions located in cache execute in 1 clock cycle

- For this memory design burst accesses from main memory requires
  - an address set-up time of 5 clock cycles
  - then each subsequent burst fetch from main memory requires 2 clock cycles per memory fetch

Q) **What is the effective execution time (EET) in nanoseconds** for this processor if the clock frequency is 100 MHz?

Class Exercise

From the lecture notes we recall that:

- 1 clock cycle = 1/100MHz = 10 ns
- Effective execution time = *hit rate * hit time + miss rate * miss penalty*
- *Hit rate: 0.95: Miss rate: 0.05*
- *Hit time = 1 clock cycle * 10 ns*
- *Miss time = (burst memory access + read all 64 bytes (in a block))*10ns*

  = (5 + 2 (64/4))*10 ns = 370 ns*

Note: If we have a cache miss, then the processor must refill one block in the cache. Since this is a 32-bit processor, we can read a maximum of 4 bytes at once. Thus, we must read main memory 64/4 or 16 times.

Since each external memory fetch requires 2 cycles, the memory read requires 16x2 = 32 clock cycles. However, we also have the overhead of setting up the address. This takes 5 clock cycles. Each clock cycle is 10 ns, so the miss penalty causes a total time delay of 370 ns:

Effective Execution Time = 0.95*10 + 0.05*370 = 9.5 + 18.5 **= 28 ns**

Cache Write Strategies

- Cache behavior is relatively straight-forward ***until new data must be stored***
  - If data is updated and stored in the cache:
    - Cache image and memory image are **no longer the same** (coherent)
- Cache activity with respect to data writes can be of two types:
  - ***Write-through*** *cache:*
    - Data is written to the **cache AND main memory** **at the same time**
  - ***Write-back*** *(****post-write****)cache:*
    - Data is **held until** bus activity allows the data to be written without interrupting other operations
    - May also wait until it has an **entire block of data** to be written

- A memory cell that has an ***updated value still in cache*** is called a ***dirty cell***

Cache Write Strategies (2)

- If the data image is not in cache
  - ***Write-around** cache:* Non-cached data is **immediately written to memory** (**do not** go through cache)
  - Or, if there is a cache block available with no corresponding dirty memory cells, it may store the data in that cache first,
    - Then do a write-through, or post-write, depending upon the design

Hits vs. Misses

- Read hits
  - this is what we want
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart
- Write hits
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)
- Write misses
  - write-around cache, or
  - read the entire block into the cache, then write the data

Performance

- Simplified Model

execution time = (execution cycles + stall cycles) × cycle time

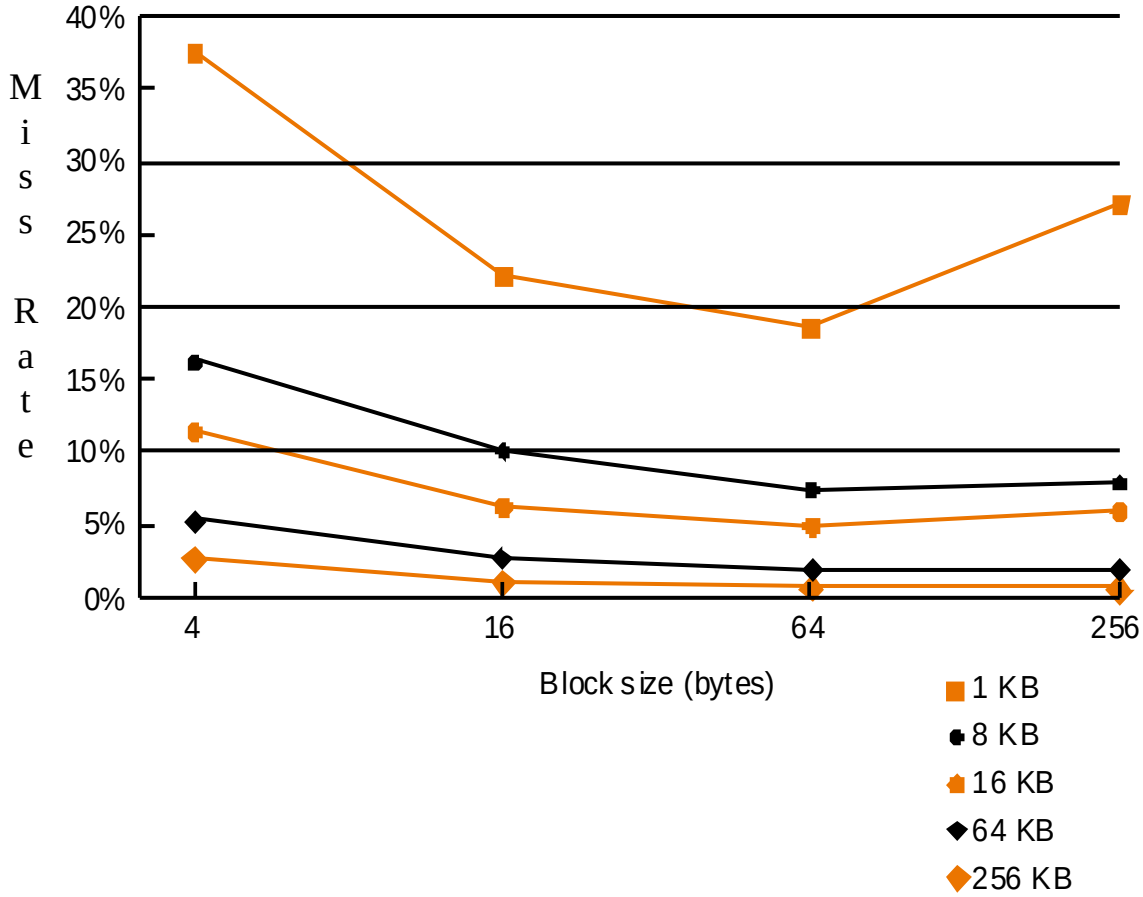
stall cycles = # of instructions × miss ratio × miss penalty

- Two ways of improving performance
  - decreasing the miss ratio
    - Increase the block size
    - What will happen if the block size is too large?

    Larger miss penalty!
  - decreasing the miss penalty

Performance

- Optimal block size?
- Increasing the block size (refill line) tends to decrease miss rate:



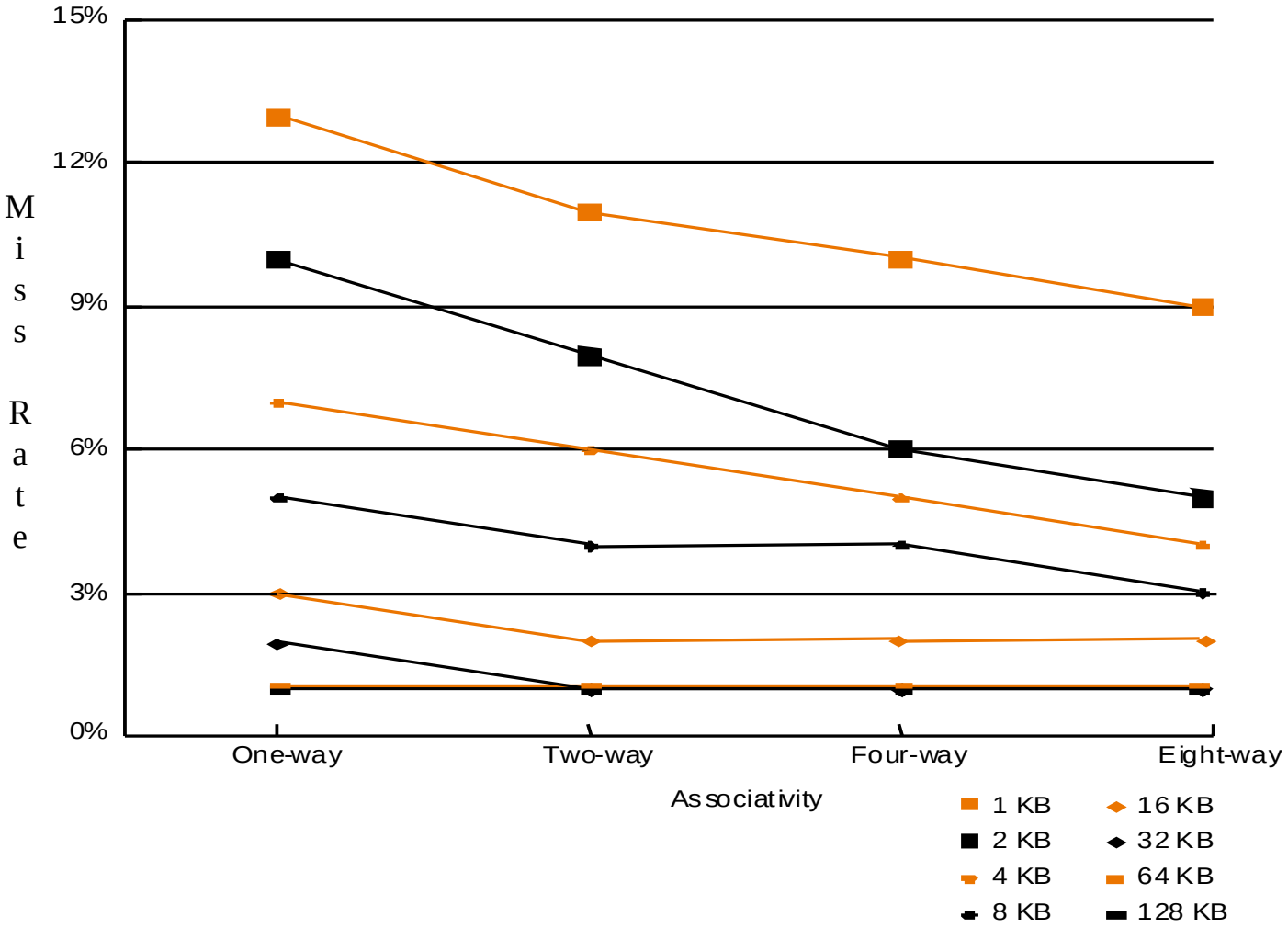
Performance

- How to improve the performance further?
- Use split caches because there is more spatial locality in code
  - Data cache (D-cache)
  - Instruction cache (I-cache)
- Cache performance can be impacted by the nature of the software running on the computer
  - By association, the performance of the cache may be influenced by the compiler used to create the code image
- Cache behavior can dramatically improve or diminish software performance

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Performance

Optimal set number?



Decreasing Miss Penalty with Multilevel Caches

- Add a second level (level two) cache:
  - Often primary cache is on the same chip as the processor
  - Use SRAMs to add another cache above primary memory (DRAM)
  - Miss penalty goes down if data is in 2nd level cache

- Using multilevel caches:
  - Try and optimize the hit time on the 1st level cache
  - Try and optimize the miss rate on the 2nd level cache

Elements of Cache Design

- Mapping function
  - Direct
  - Associative
  - Set-Associative
- Write Policy
  - Write through
  - Write back
- Replacement Algorithm
  - First in first out (FIFP)
  - Random
  - *Least frequently used (LFU)*
  - *Least recently used (LRU)*
- Cache Address
  - Logical
  - Physical
- Cache Size
- Line Size
- Number of Caches
  - Single or two level
  - Unified or split