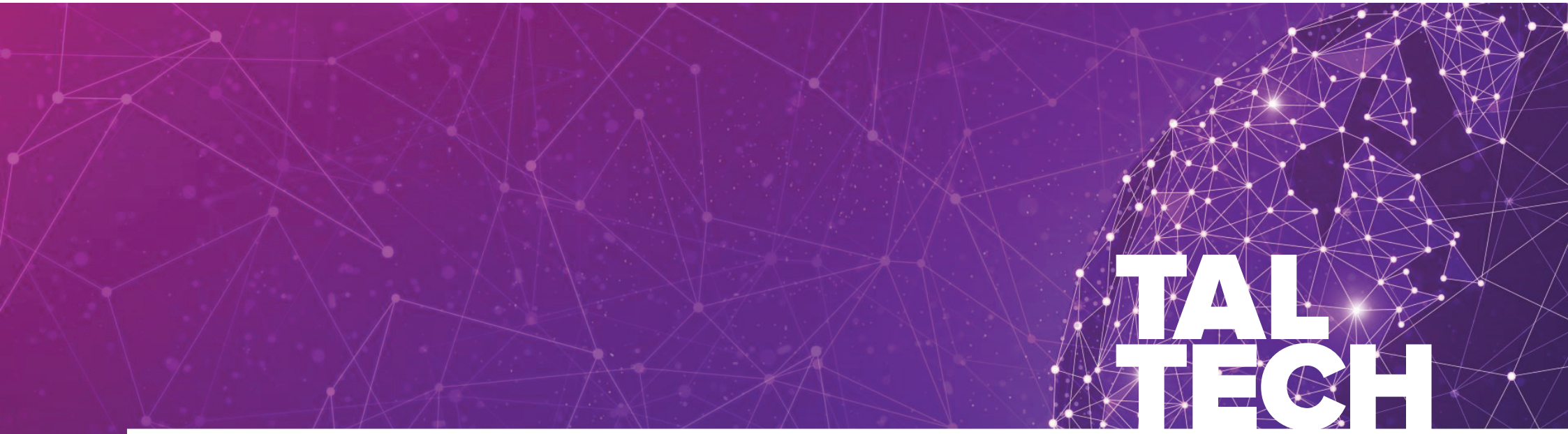


MICROPROCESSOR SYSTEMS (IAS0430)

Department of Computer Systems
Tallinn University of Technology

19.11.2021

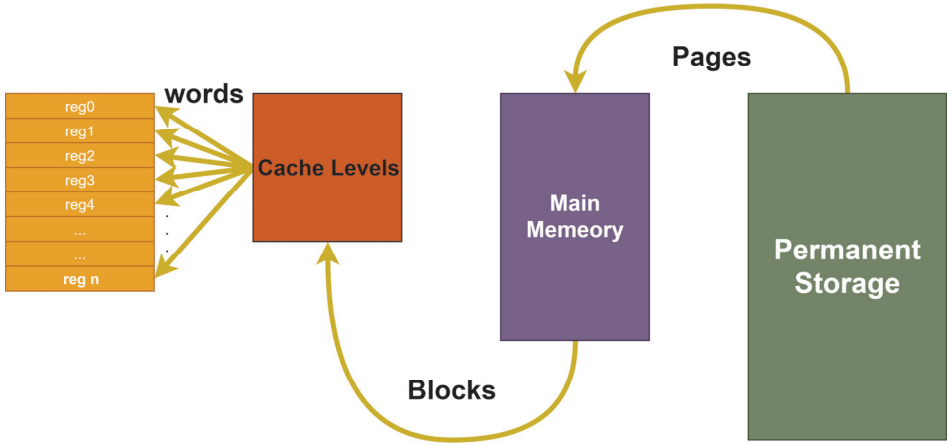
MEMORY MANAGEMENT

- Last few weeks we talk about memory hierarchy and what is memory made of
  - We talked about technologies (SRAM, DRAM, …)
  - We talked about memory structures (SRAM arrays, Registers, …)
  - We talked about memory data structures (words, blocks, …)
  - We talked about memory devices (reg files, cache, RAM, …)

- Information is propagated from one level of memory to lower level of memory.
  - Once the CPU requests information, there are few places where it can find the data.
    - In the registers, in the form of **words.** Words are data or instructions that are either **32-bit or 64-bit** long depending on how the CPU is designed.
    - In the cache, words are stored as part of a **cache block/line**. A cache block is a collection of words (data) stored in a memory location in the cache.
    - In the RAM, where cache blocks are stored as part of **pages**. Pages are a collection of consecutive addresses that make up a program. A program can be composed of different pages. Pages are divided into blocks when moved into RAM.

MEMORY MANAGEMENT

- **Data Propagation**
  - As we saw, programs are stored in the permanent storage.
  - When moved to the RAM, they are divide into Pages.
  - Those pages are then divided into blocks which are then copied to the cache
  - In the cache, the blocks are divided into words
  - Words are then sent to the register file, and the register file delivers those words to the CPU.
  - We will take this propagation as step by step, starting with the RAM.



MEMORY MANAGEMENT

- **Data Propagation**
  - There are four important points to remember:
    - **Programs (Jobs) are always in permanent storage**. Before they are moved to the RAM, they are assigned an identifier (a number).
      - This number is maintained by the **MMU**.
      - They are divided into pages only when they are required to execute.
    - **The RAM only contains a copy of the pages of a program**. Programs are never moved from where they are on permanent storage.
      - This allows multiple copies of programs to be executed at the same time, while avoiding corrupting or destroying program data.
    - **RAM always have the page we require.** Even if we try to access a page that is still in permanent storage, it must be loaded into RAM, before the CPU can request access to it.

    - **From now on, our RAM works using the demand paging scheme. This is the default scheme unless otherwise stated.**

MEMORY MANAGEMENT

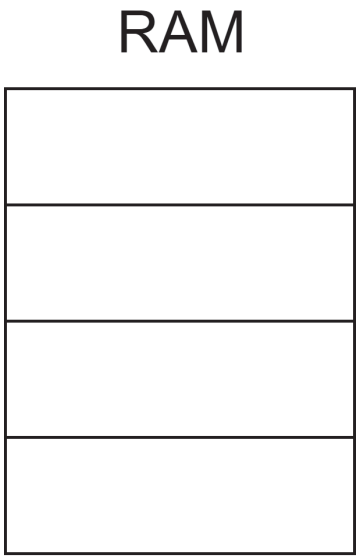
- **RAM Management**
- **First**, we need to learn about techniques called the **Replacement Policies**:
  - **Are algorithms or optimization programs that a hardware-managed structure can use to manage a computer memory device.**
  - These policies are largely used for **managing blocks in the cache and pages in the RAM**.
- These are two replacement policies of interest to this course:
  - **First In First Out (FIFO)**
    - Resembles a queue of data entered one by one and removed by the oldest element in the queue
  - **Least Recently Used (LRU)**
    - Resembles a queue where the least used element is pushed to the back after each access, then removed when it is at the end of the queue.

MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail.

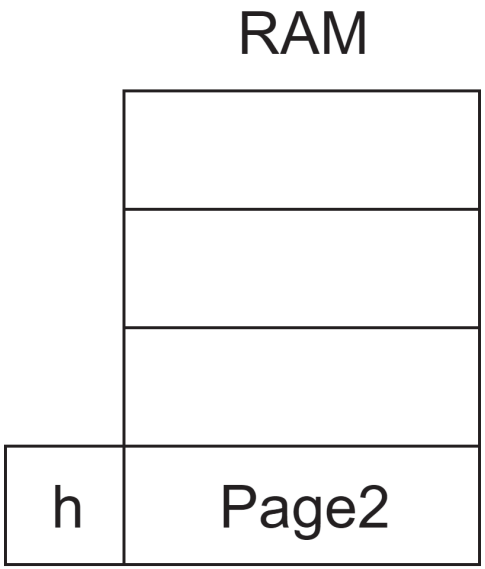
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page3, Page1, Page3, Page6, Page1, Page4, page7



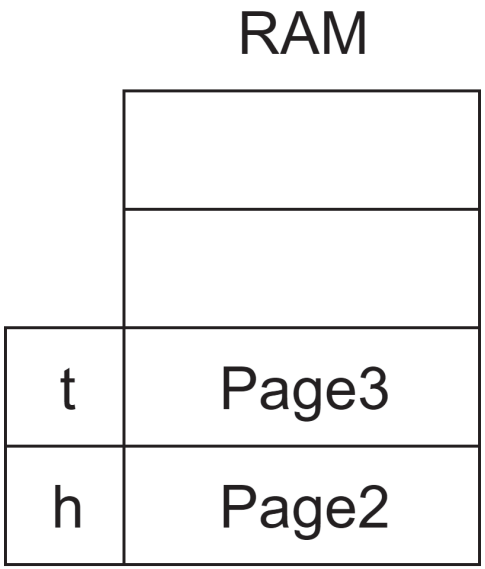
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - **Page2**, Page3, Page1, Page3, Page6, Page1, Page4, page7
    - Page2 is entered first



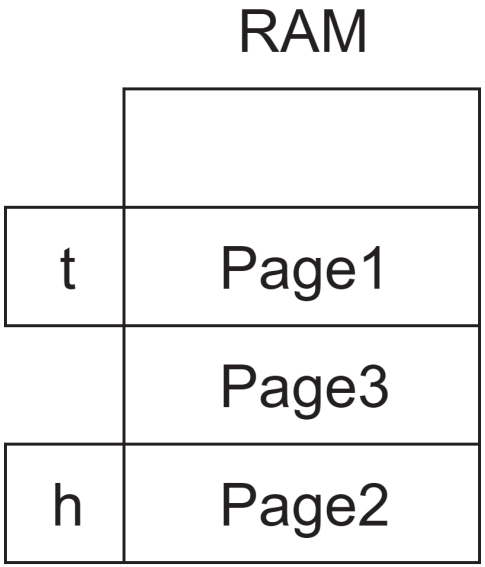
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, **Page3**, Page1, Page3, Page6, Page1, Page4, page7
    - Page3 is entered next



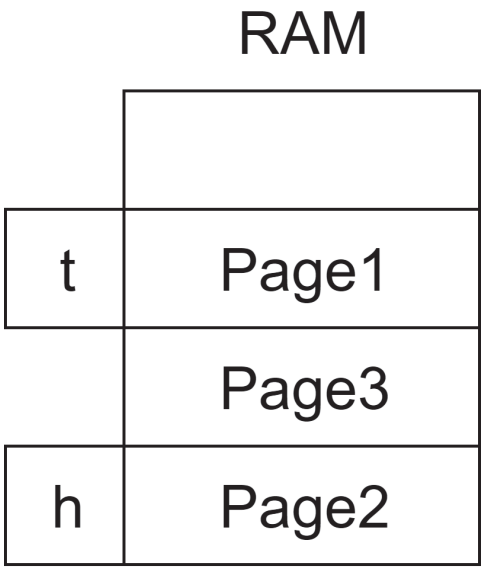
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page3, **Page1**, Page3, Page6, Page1, Page4, page7
    - Page1 is entered next



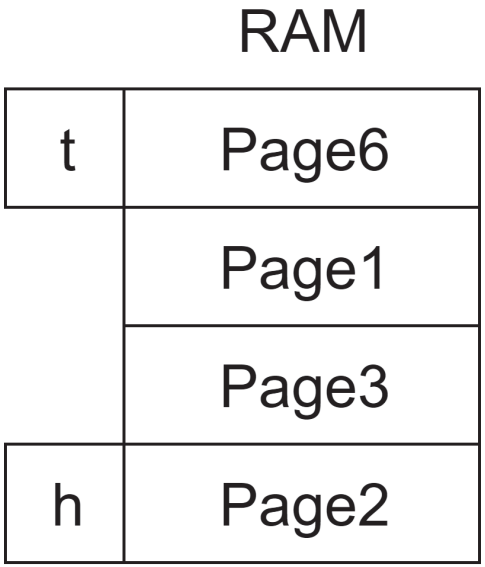
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page3, Page1, **Page3**, Page6, Page1, Page4, page7
    - Page3 is already in the RAM, so nothing needs to be done



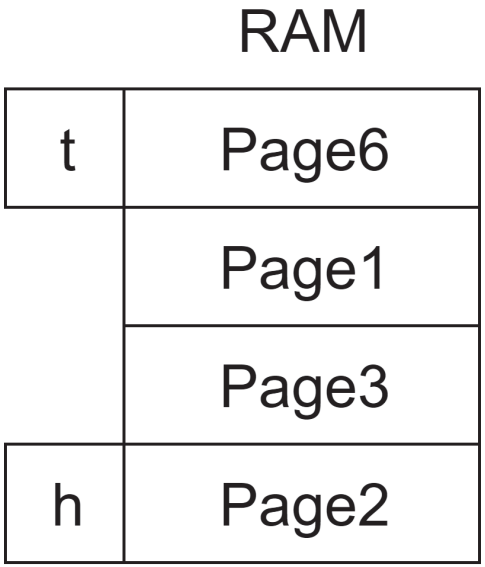
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page3, Page1, Page3, **Page6**, Page1, Page4, page7
    - Page6 is entered



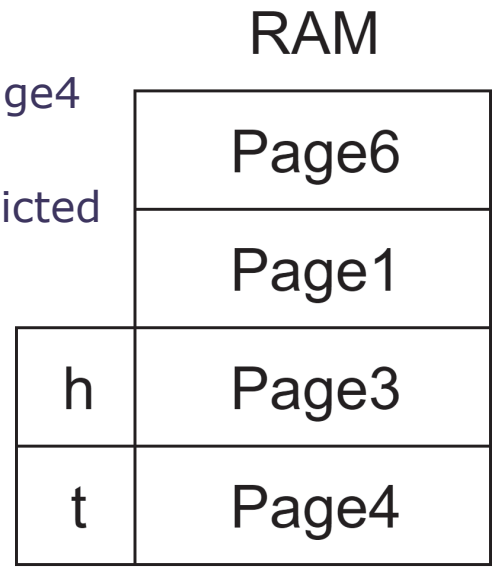
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page3, Page1, Page3, Page6, **Page1**, Page4, page7
    - Page1 is already in the RAM, so nothing needs to be done



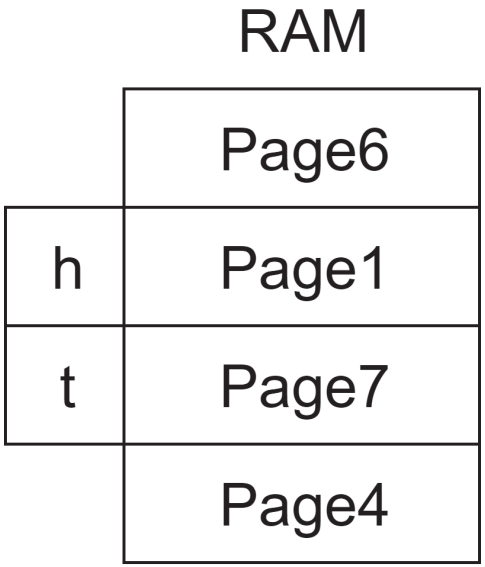
MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page3, Page1, Page3, Page6, Page1, **Page4**, page7
    - The RAM is now full, so there is no free page frame to enter page4
    - This requires one page to be removed.
    - Since it is FIFO replacement policy, the first page entered is evicted
      - In this case, page2 is the first to enter
      - Page3 is removed from the RAM
      - Then replaced by page4



MEMORY MANAGEMENT

- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using FIFO:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page3, Page1, Page3, Page6, Page1, Page4, **page7**
    - Page7 also requires a page to removed
      - Since Page3 is now the oldest in the RAM
      - Page3 is removed and replaced by page7



MEMORY MANAGEMENT

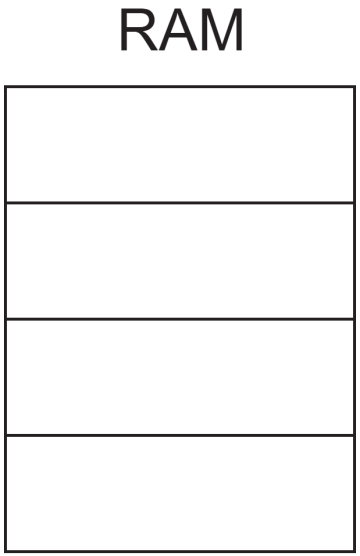
- **First In First Out (FIFO)**
  - This replacement policy resembles a queue. The first element to enter the queue is the first element to be removed when the queue is full.
  - The **MMU** keeps track of which of the blocks or pages entered the memory first. Assigns the first element to the queue as the head and last as the tail in a **linked list**.
  - FIFO is generally slow and does not take into account the frequency of use of each of the data element.
    - Regardless of an element being used frequently, if it is at the head of the queue, it will be evicted if the queue become full.

MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full

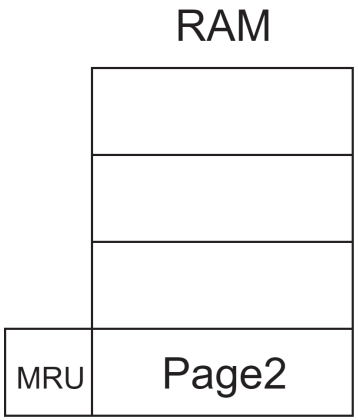
MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page1, Page2, Page3, Page5, Page1, Page4, page7



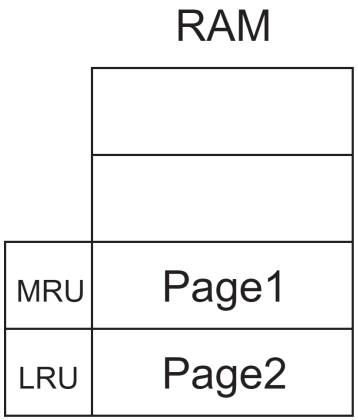
MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - **Page2**, Page1, Page2, Page3, Page5, Page1, Page4, page7
    - Page2 is entered first and is marked as the most recently used



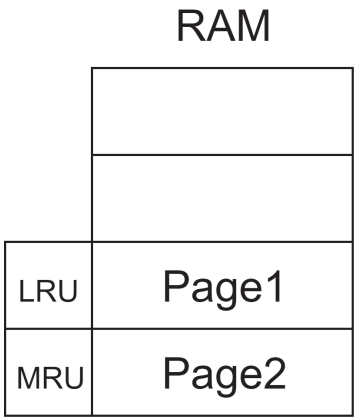
MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, **Page1**, Page2, Page3, Page5, Page1, Page4, page7
    - Page1 is entered first and is marked as the most recently used
    - Since page2 was used the least, we mark it as least recently used



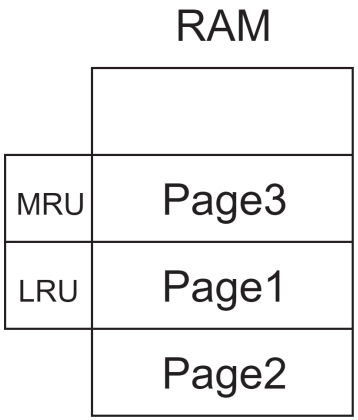
MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page1, **Page2**, Page3, Page5, Page1, Page4, page7
    - Page2 is already in the RAM, we mark it as the MRU
    - We mark the least recently used Page1 as LRU



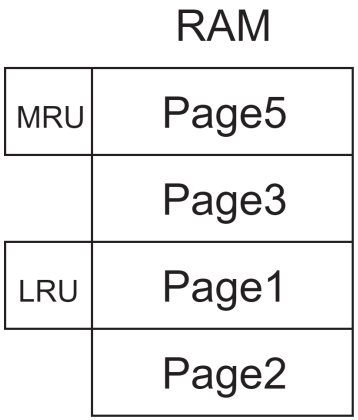
MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page1, Page2, **Page3,** Page5, Page1, Page4, page7
    - Page3 is entered and marked as MRU
    - Page1 stays LRU since we did not used it recently



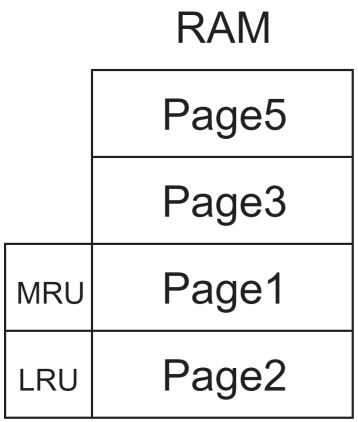
MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page1, Page2, Page3**, Page5**, Page1, Page4, page7
    - Page 5 is entered and marked as MRU



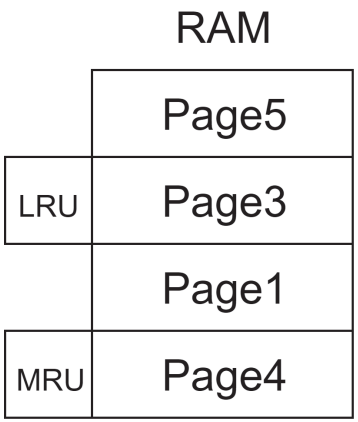
MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page1, Page2, Page3, Page5, **Page1**, Page4, page7
    - Page1 is already in the cache, we change it back to MRU
    - Now, since Page2 was the least page used, it becomes LRU



MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page1, Page2, Page3, Page5, Page1, **Page4**, page7
    - RAM is now full. We replace the LRU element with the new element
      - This case it is Page2
    - Then mark the least recently used element with LRU
      - This case it is Page3
    - Page4 is marked LRU



MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - Lets say we want to enter the following pages into a **RAM** with only four page frames using LRU:
    - Page1, Page2, Page3, Page4, Page5, Page6, Page7
  - Those pages are requested by the RAM in the following order:
    - Page2, Page1, Page2, Page3, Page5, Page1, Page4, **page7**
    - We replace the LRU element with the new element
      - This case it is Page3
    - Then mark the least recently used element with LRU
      - This case it is Page5
    - Page7 is marked LRU

RAM

LRU	Page5
MRU	Page7
	Page1
	Page4

MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - LRU is an excellent replacement policy because it favors data that is used most during execution.
  - This allows better memory performance.. Why?

MEMORY MANAGEMENT

- **Least Recently Used (LRU)**
  - This replacement policy resembles a queue where the most recently used element is put at the head of the queue. Once the queue is full, the tail of the queue is replaced.
  - The **MMU** keeps track of which of the blocks or pages is being used most recently. This allows the **most recently used (MRU)** to stay inside the memory for the longest time possible. It also keeps track of which element was the **least recently used (LRU)** in order to evict it once the memory is full
  - LRU is an excellent replacement policy because it favors data that is used most during execution.
  - This allows better memory performance.
    - Having the most recently used element in memory helps avoiding wasting time bringing data from higher memory hierarchy
      - This reduces the over time execution of a program

MEMORY MANAGEMENT

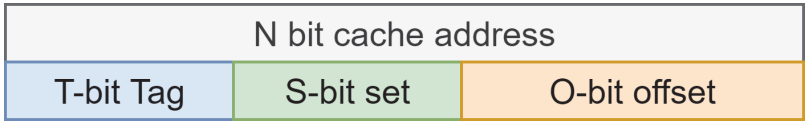
- **Cache Management**
  - The cache has a little more complicated scheme of replacement policies that RAM does.
  - This is because caches require a much more efficient form of data management since they are smaller and faster than RAM.
  - **Mapping**
    - Mapping is **the technique used to copy contents from the main memory to the cache.**
      - Mapping **allows the content of the main memory to be copied to the cache** in an organized fashion making it easier to track what data is being used where.
      - Mapping also **allows increasing performance by using more flexible techniques of organizing the cache** allowing taking advantage of the full capacity of the cache.
        - It is important to note that the content of the main memory is being **copied** not moved.

MEMORY MANAGEMENT

- **Cache Management**
- The cache takes the address in order to locate the cache block requested by the register file.
  - Once the block is located, the word is then loaded to the register file to be processed. This called a **cache hit**.
  - If the block is not in the cache. This is called a **cache miss**. The block has to be in the Main memory. This is because a block is always located in a page.
    - There are three types of cache misses that we are concerned with:
      - **Compulsory misses (Read and Write)**
        - Happen when we try to write access a cache location that is **empty or was not used before (invalid)**.
      - **Conflict misses (Write)**
        - Happen when a cache block is loaded where another cache block is already stored
      - **Capacity misses (Write)**
        - Happen when a cache block is needed, but the cache is full. A block must be evicted before the new block is stored.

MEMORY MANAGEMENT

- **Cache Management**
  - As shown before, locations in the cache have an address.



  - Addresses help manage the cache by pointing to the exact locations of cache that require writing or reading.
  - The address also helps avoiding the eviction of the wrong cache blocks.
  - All the mapping techniques rely on the cache block addresses
  - There are three mapping techniques that we will focus on in the course.
    - Direct Mapping
    - Fully Associative Mapping
    - K-way Associative Mapping

MEMORY MANAGEMENT

- **Direct Mapping**
  - Direct mapping technique is based on the principle **that every block of data in the main memory has only one location in the cache where it can be mapped/copied**.
  - The implementation of the cache **is very simple** as there is not need for a complex design.
  - **Replacement policies are not required** since each block can only be put in one location, so if that location is occupied, it will be emptied before the new block can be copied to its location.
  - Let us assume the following cache requests executed twice

MEMORY MANAGEMENT

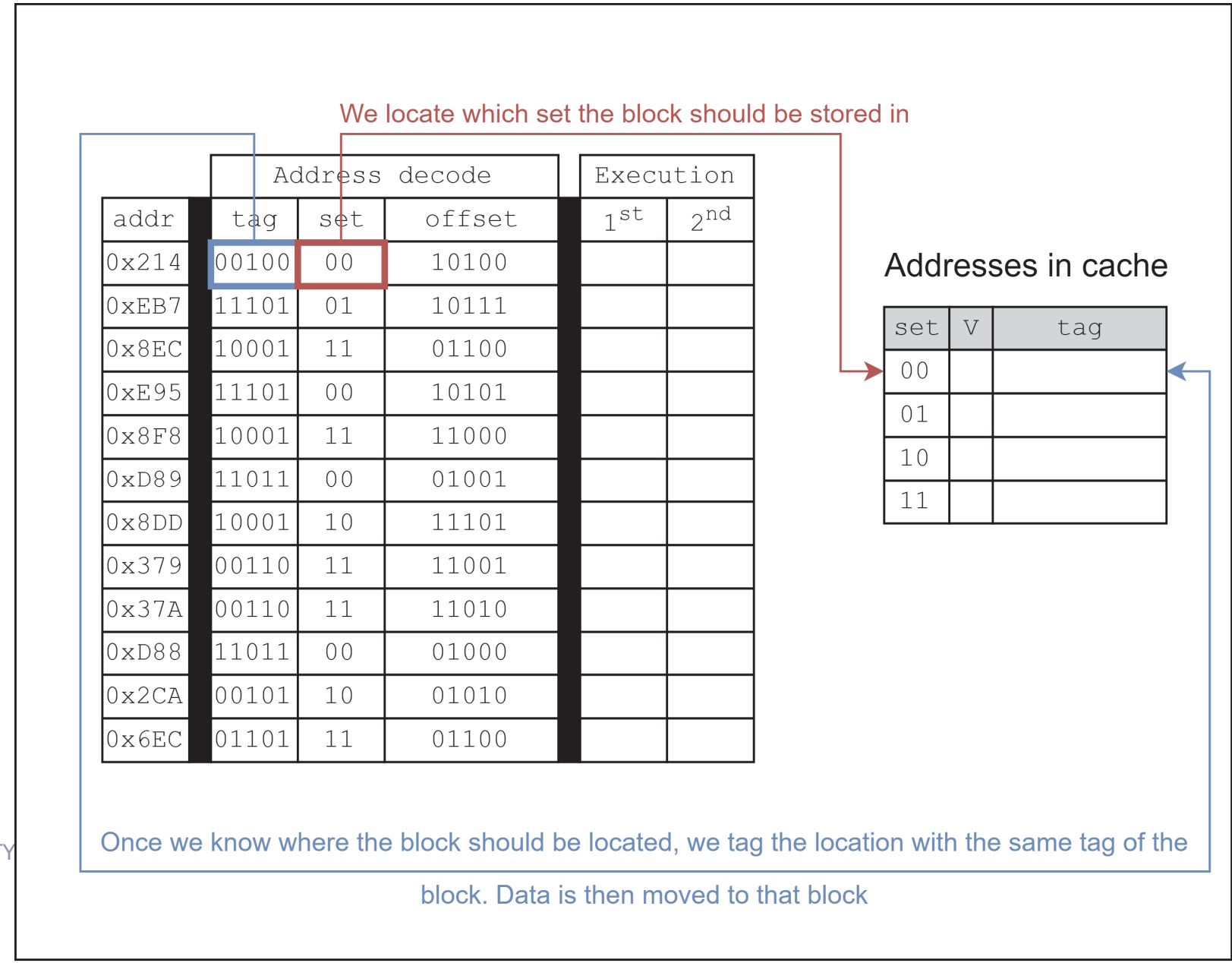
Addresses in main memory

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100		
0xEB7	11101	01	10111		
0x8EC	10001	11	01100		
0xE95	11101	00	10101		
0x8F8	10001	11	11000		
0xD89	11011	00	01001		
0x8DD	10001	10	11101		
0x379	00110	11	11001		
0x37A	00110	11	11010		
0xD88	11011	00	01000		
0x2CA	00101	10	01010		
0x6EC	01101	11	01100		

Addresses in cache

set	V	tag
00		
01		
10		
11		

MEMORY MANAGEMENT



MEMORY MANAGEMENT

since the set was origionally empty, this is called a compulsory miss. It is a miss that is unavoidable and must always happen when a set is used for the first time

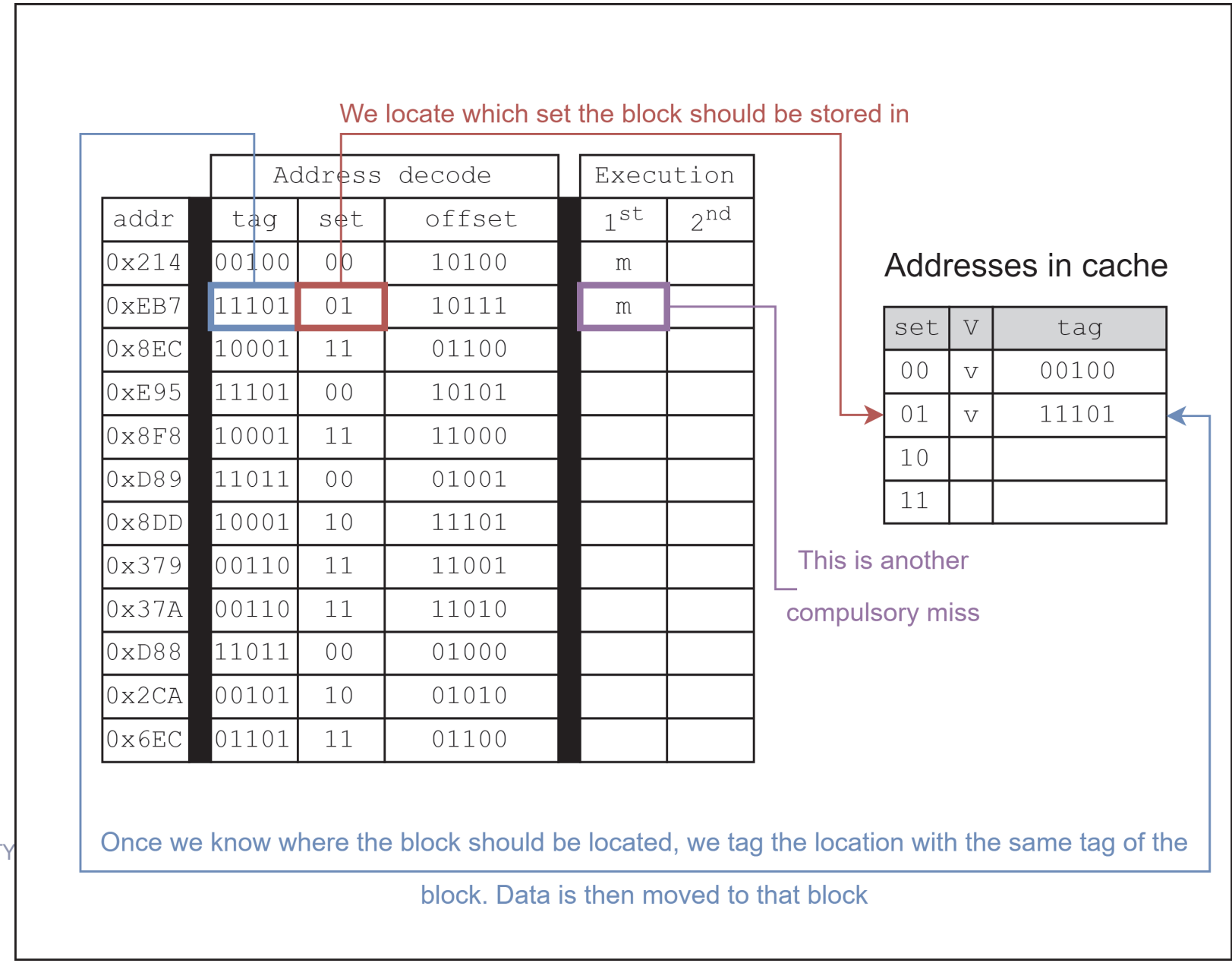
	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111		
0x8EC	10001	11	01100		
0xE95	11101	00	10101		
0x8F8	10001	11	11000		
0xD89	11011	00	01001		
0x8DD	10001	10	11101		
0x379	00110	11	11001		
0x37A	00110	11	11010		
0xD88	11011	00	01000		
0x2CA	00101	10	01010		
0x6EC	01101	11	01100		

Addresses in cache

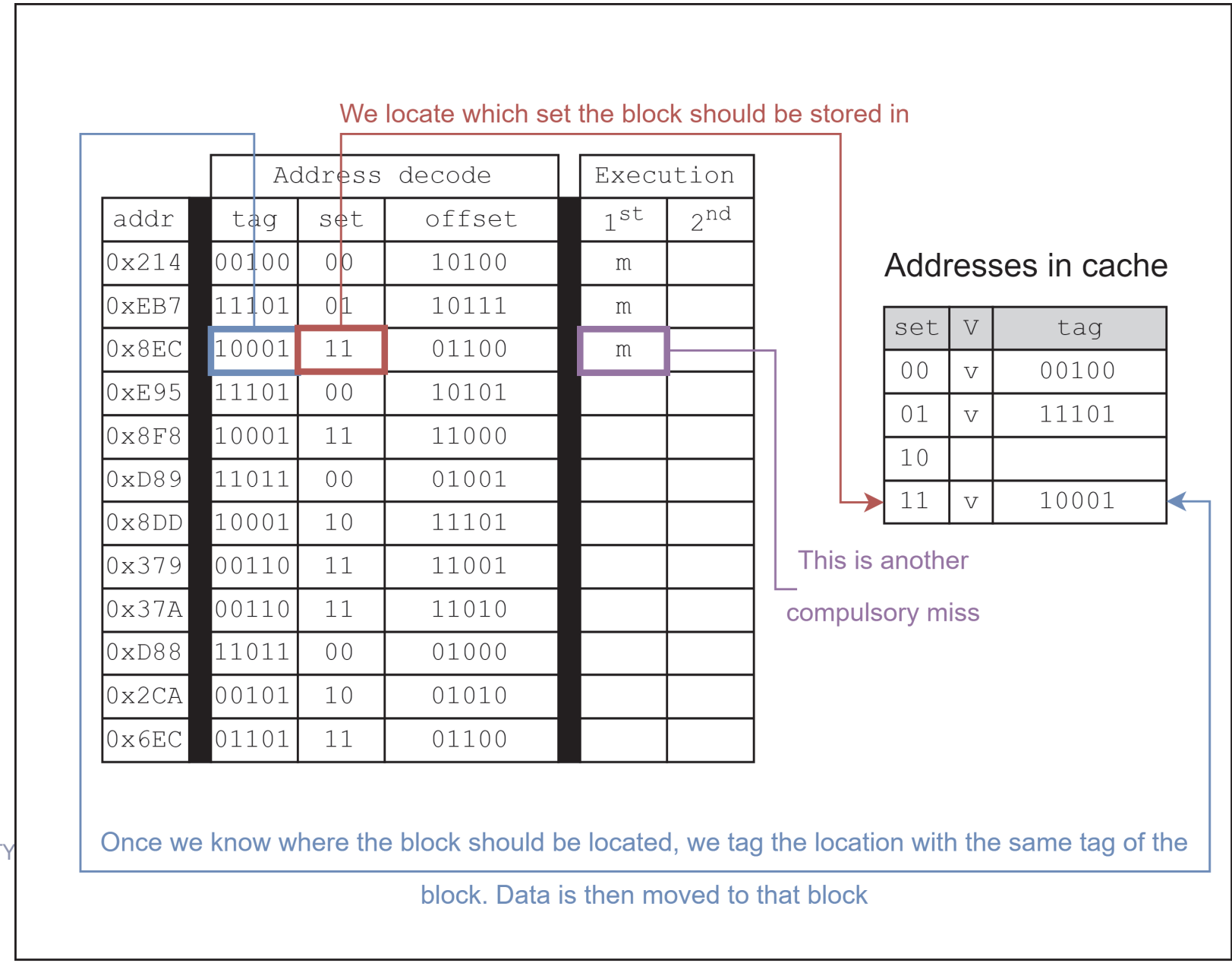
set	V	tag
00	v	00100
01		
10		
11		

The valid bit is then updated to signal that the set is in use.

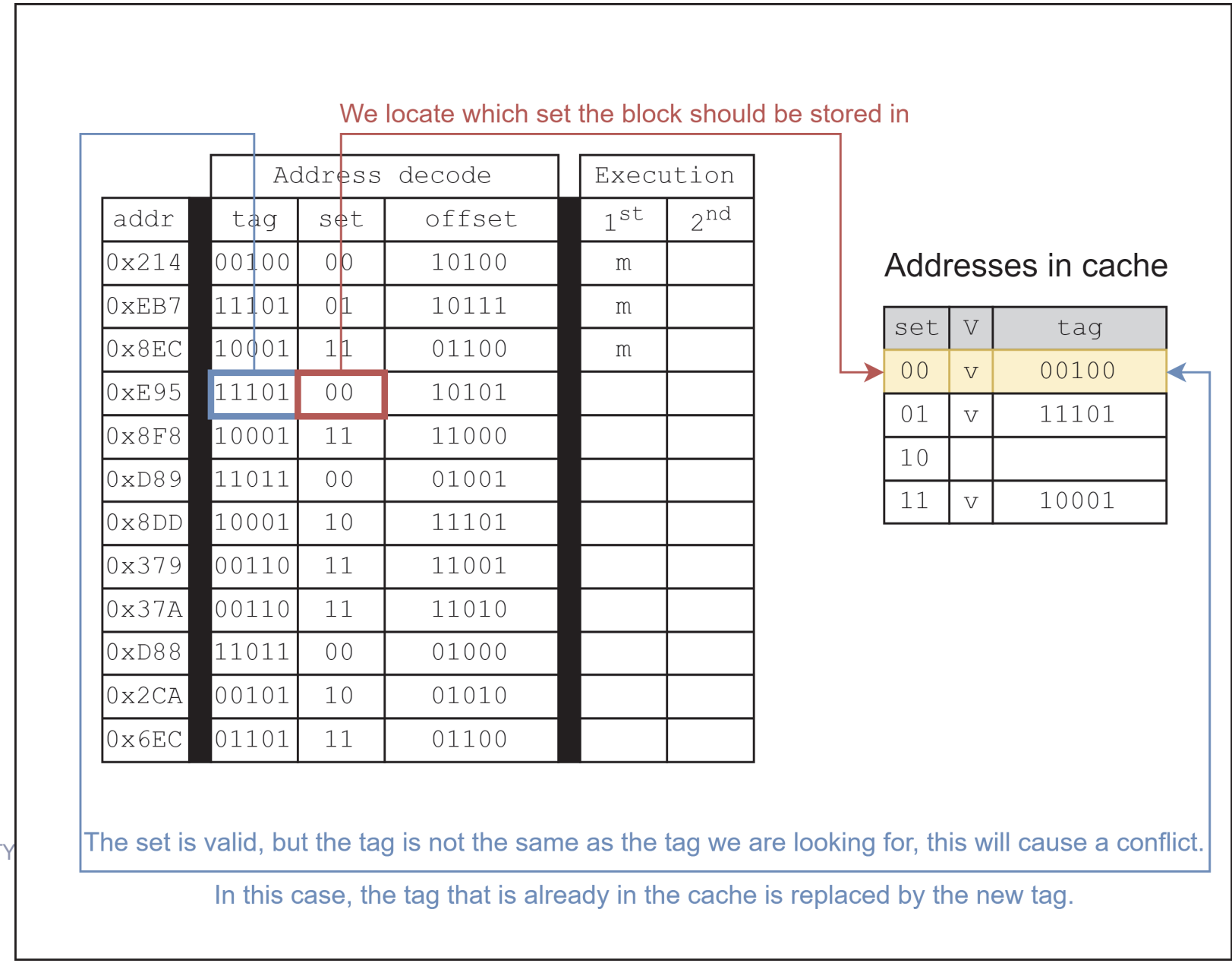
MEMORY MANAGEMENT



MEMORY MANAGEMENT



MEMORY MANAGEMENT



MEMORY MANAGEMENT

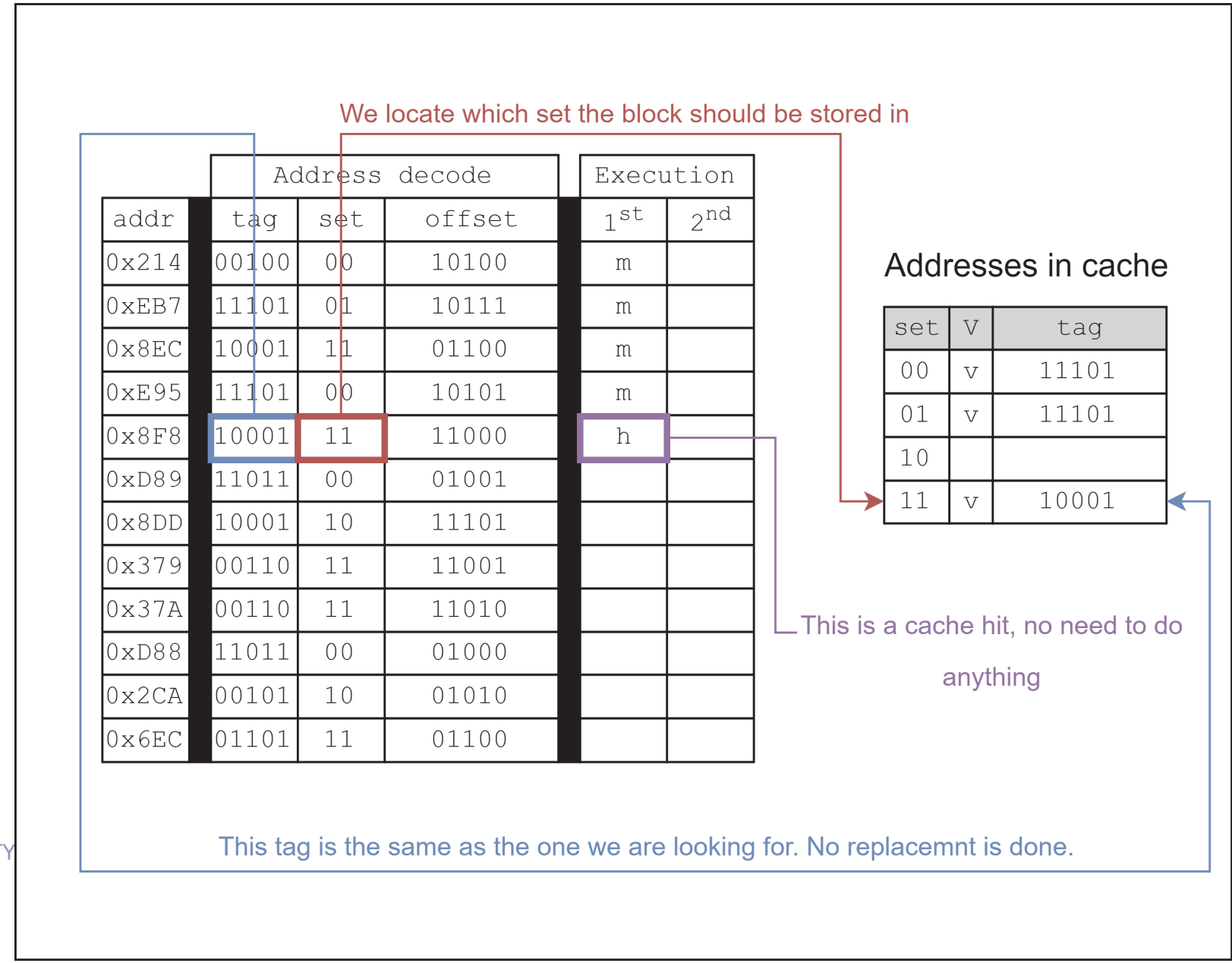
	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000		
0xD89	11011	00	01001		
0x8DD	10001	10	11101		
0x379	00110	11	11001		
0x37A	00110	11	11010		
0xD88	11011	00	01000		
0x2CA	00101	10	01010		
0x6EC	01101	11	01100		

Addresses in cache

set	V	tag
00	v	11101
01	v	11101
10		
11	v	10001

This is a conflict miss because the set is valid, but has a different tag that the one we are looking for

MEMORY MANAGEMENT



MEMORY MANAGEMENT

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101		
0x379	00110	11	11001		
0x37A	00110	11	11010		
0xD88	11011	00	01000		
0x2CA	00101	10	01010		
0x6EC	01101	11	01100		

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10		
11	v	10001

MEMORY MANAGEMENT

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001		
0x37A	00110	11	11010		
0xD88	11011	00	01000		
0x2CA	00101	10	01010		
0x6EC	01101	11	01100		

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	10001
11	v	10001

MEMORY MANAGEMENT

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001	m	
0x37A	00110	11	11010		
0xD88	11011	00	01000		
0x2CA	00101	10	01010		
0x6EC	01101	11	01100		

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	10001
11	v	00110

MEMORY MANAGEMENT

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001	m	
0x37A	00110	11	11010	h	
0xD88	11011	00	01000	h	
0x2CA	00101	10	01010		
0x6EC	01101	11	01100		

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	10001
11	v	00110

MEMORY MANAGEMENT

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001	m	
0x37A	00110	11	11010	h	
0xD88	11011	00	01000	h	
0x2CA	00101	10	01010	m	
0x6EC	01101	11	01100		

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	00101
11	v	00110

MEMORY MANAGEMENT

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001	m	
0x37A	00110	11	11010	h	
0xD88	11011	00	01000	h	
0x2CA	00101	10	01010	m	
0x6EC	01101	11	01100	m	

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	00101
11	v	01101

MEMORY MANAGEMENT

- **For all cache replacement policies, there exists two rates:**
  - **The miss rate:** the miss rate indicates the percentage of the misses compared to the overall accesses to the cache. It is described as follows:
    - $\boldsymbol{miss\ rate = \frac{Total\ number\ of\ misses}{Total\ number\ of\ accesses} * 100\%}$
  - **The hit rate:** the hit rate indicates the percentage of the hits compared to the overall accesses to the cache. It is described as follows:
    - $\boldsymbol{hit\ rate = \frac{Total\ number\ of\ hits}{Total\ number\ of\ accesses} * 100\%}$
    - **From those equations, both the miss rate and hit rate can be found if either of them are known:**
      - $\boldsymbol{miss\ rate = \left(1 - \frac{Total\ number\ of\ hits}{Total\ number\ of\ accesses}\right) * 100\%}$
        - **or** $\boldsymbol{Miss\ rate = (100\% - hit\ rate)}$
      - $\boldsymbol{hit\ rate = \left(1 - \frac{Total\ number\ of\ misess}{Total\ number\ of\ accesses}\right) * 100\%}$
        - **or** $\boldsymbol{hit\ rate = (100\% - miss\ rate)}$

MEMORY MANAGEMENT

- ***miss rate***

- ***hit rate***

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001	m	
0x37A	00110	11	11010	h	
0xD88	11011	00	01000	h	
0x2CA	00101	10	01010	m	
0x6EC	01101	11	01100	m	

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	00101
11	v	01101

MEMORY MANAGEMENT

- $miss\ rate = \frac{Total\ number\ of\ misses}{Total\ number\ of\ accesses} * 100\%$
  - $= \frac{9}{12} * 100\%$
  - $= 75\ \%$

- $hit\ rate$

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001	m	
0x37A	00110	11	11010	h	
0xD88	11011	00	01000	h	
0x2CA	00101	10	01010	m	
0x6EC	01101	11	01100	m	

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	00101
11	v	01101

MEMORY MANAGEMENT

- $miss\ rate = \frac{Total\ number\ of\ misses}{Total\ number\ of\ accesses} * 100\%$
  - $= \frac{9}{12} * 100\%$
  - $= 75\ \%$

- $hit\ rate = (100\% - miss\ rate)$
  - $= 100\% - 75\ \%$
  - $= 25\ \%$

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	00100	00	10100	m	
0xEB7	11101	01	10111	m	
0x8EC	10001	11	01100	m	
0xE95	11101	00	10101	m	
0x8F8	10001	11	11000	h	
0xD89	11011	00	01001	m	
0x8DD	10001	10	11101	m	
0x379	00110	11	11001	m	
0x37A	00110	11	11010	h	
0xD88	11011	00	01000	h	
0x2CA	00101	10	01010	m	
0x6EC	01101	11	01100	m	

Addresses in cache

set	V	tag
00	v	11011
01	v	11101
10	v	00101
11	v	01101

MEMORY MANAGEMENT

- **Fully Associative Mapping**
  - Fully Associative Mapping is **based on the principle that a block can be mapped to any free location in the cache.**
  - In fully associative, **the cache is more flexible as it will cause less data to be evicted.** At the same time it has a complex cache design.
  - Since we do not evict any blocks until the cache is full, **we need a way to decide which block will be evicted.**
    - In order to do this, we employ replacement policies
  - **Replacement policies used with Fully Associative mapping**:
    - First In First Out (FIFO): First block to occupy the cache, is the first block to be evicted.
    - **Least Recently Used (LRU):** The first block to be evicted, is the oldest unaccessed block in the cache.

MEMORY MANAGEMENT

Addresses in main memory

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100		
0xEB7	1110101	10111		
0x8EC	1000111	01100		
0xE95	1110100	10101		
0x8F8	1000111	11000		
0xD89	1101100	01001		
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag

MEMORY MANAGEMENT

since the set was origionally empty, this is called a compulsory miss. It is a miss that is unavoidable and must always happen when a set is used for the first time

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111		
0x8EC	1000111	01100		
0xE95	1110100	10101		
0x8F8	1000111	11000		
0xD89	1101100	01001		
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
	v	0010000

The valid bit is then updated to signal that the set is in use.

The first tag is moved to the first free set in the cache.

This is done for the tags that can fit inside the cache.

MEMORY MANAGEMENT

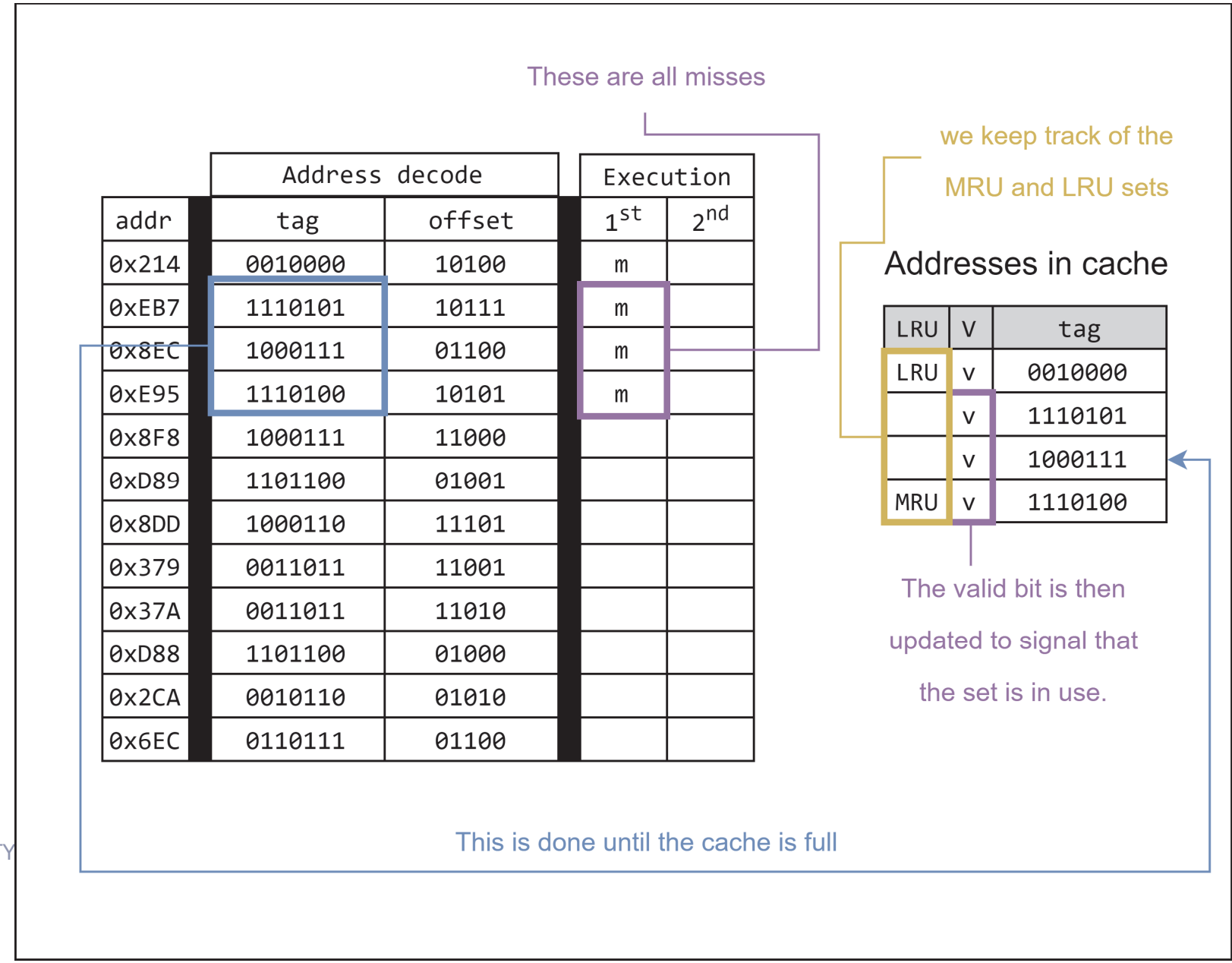
	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111		
0x8EC	1000111	01100		
0xE95	1110100	10101		
0x8F8	1000111	11000		
0xD89	1101100	01001		
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
MRU	v	0010000

We mark this set as the most recently used set

MEMORY MANAGEMENT



	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000		
0xD89	1101100	01001		
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

LRU	V	tag
LRU	v	0010000
	v	1110101
	v	1000111
MRU	v	1110100

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000		
0xD89	1101100	01001		
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
LRU	v	0010000
	v	1110101
	v	1000111
MRU	v	1110100

This is a hit, since this cache block is in the cache

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
0xD89	1101100	01001		
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
LRU	v	0010000
	v	1110101
MRU	v	1000111
	v	1110100

we change it to the MRU since it was a hit access

This is a hit, since this cache block is in the cache

There is no need to evict any blocks, LRU stays the same.

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
~~0xD89~~	1101100	01001		
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
LRU	v	0010000
	v	1110101
MRU	v	1000111
	v	1110100

This block is not in the cache, which means it must replace the LRU block.

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
~~0xD89~~	1101100	01001	m	
0x8DD	1000110	11101		
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
MRU	v	1101100
LRU	v	1110101
	v	1000111
	v	1110100

The new block is MRU, while the oldest block in the cache becomes LRU

This block is not in the cache, which means it must replace the LRU block.

The oldest block becomes the least recently used block in the cache.

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
0xD89	1101100	01001	m	
0x8DD	1000110	11101	m	
0x379	0011011	11001		
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
	v	1101100
MRU	v	1000110
	v	1000111
LRU	v	1110100

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
0xD89	1101100	01001	m	
0x8DD	1000110	11101	m	
0x379	0011011	11001	m	
0x37A	0011011	11010		
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
	v	1101100
	v	1000110
LRU	v	1000111
MRU	v	0011011

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
0xD89	1101100	01001	m	
0x8DD	1000110	11101	m	
0x379	0011011	11001	m	
0x37A	0011011	11010	h	
0xD88	1101100	01000		
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
	v	1101100
	v	1000110
LRU	v	1000111
MRU	v	0011011

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
0xD89	1101100	01001	m	
0x8DD	1000110	11101	m	
0x379	0011011	11001	m	
0x37A	0011011	11010	h	
0xD88	1101100	01000	h	
0x2CA	0010110	01010		
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
MRU	v	1101100
	v	1000110
LRU	v	1000111
	v	0011011

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
0xD89	1101100	01001	m	
0x8DD	1000110	11101	m	
0x379	0011011	11001	m	
0x37A	0011011	11010	h	
0xD88	1101100	01000	h	
0x2CA	0010110	01010	m	
0x6EC	0110111	01100		

Addresses in cache

LRU	V	tag
	v	1101100
LRU	v	1000110
MRU	v	0010110
	v	0011011

MEMORY MANAGEMENT

	Address decode		Execution	
addr	tag	offset	1st	2nd
0x214	0010000	10100	m	
0xEB7	1110101	10111	m	
0x8EC	1000111	01100	m	
0xE95	1110100	10101	m	
0x8F8	1000111	11000	h	
0xD89	1101100	01001	m	
0x8DD	1000110	11101	m	
0x379	0011011	11001	m	
0x37A	0011011	11010	h	
0xD88	1101100	01000	h	
0x2CA	0010110	01010	m	
0x6EC	0110111	01100	m	

Addresses in cache

LRU	V	tag
	v	1101100
MRU	v	0110111
	v	0010110
LRU	v	0011011

MEMORY MANAGEMENT

- **K-way Associative Mapping**
  - K-way Associative Mapping techniques is based on the principle that a cache is divided into sets and each set holds a certain **group** of cache blocks.
  - This is a combination of Similar to Direct Mapping, but with Fully associative flexibility.
  - K-way associative uses replacement policies for each of the sets on its own. This allows a relatively flexible cache, but at high cost of high complexity.
  - Since we have four sets needed to map the blocks to the cache:
    - 2-way associative is used in our example..

MEMORY MANAGEMENT

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100		
0xEB7	111010	1	10111		
0x8EC	100011	1	01100		
0xE95	111010	0	10101		
0x8F8	100011	1	11000		
0xD89	110110	0	01001		
0x8DD	100011	0	11101		
0x379	001101	1	11001		
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0					
1					

MEMORY MANAGEMENT

We locate which set the block should be stored in

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100		
0xEB7	111010	1	10111		
0x8EC	100011	1	01100		
0xE95	111010	0	10101		
0x8F8	100011	1	11000		
0xD89	110110	0	01001		
0x8DD	100011	0	11101		
0x379	001101	1	11001		
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0					
1					

Once we know where the block should be located. Once that is determined, the tag is moved to the first available way in the set. In this case, it is way 0

MEMORY MANAGEMENT

This is a compulsory miss

	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111		
0x8EC	100011	1	01100		
0xE95	111010	0	10101		
0x8F8	100011	1	11000		
0xD89	110110	0	01001		
0x8DD	100011	0	11101		
0x379	001101	1	11001		
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

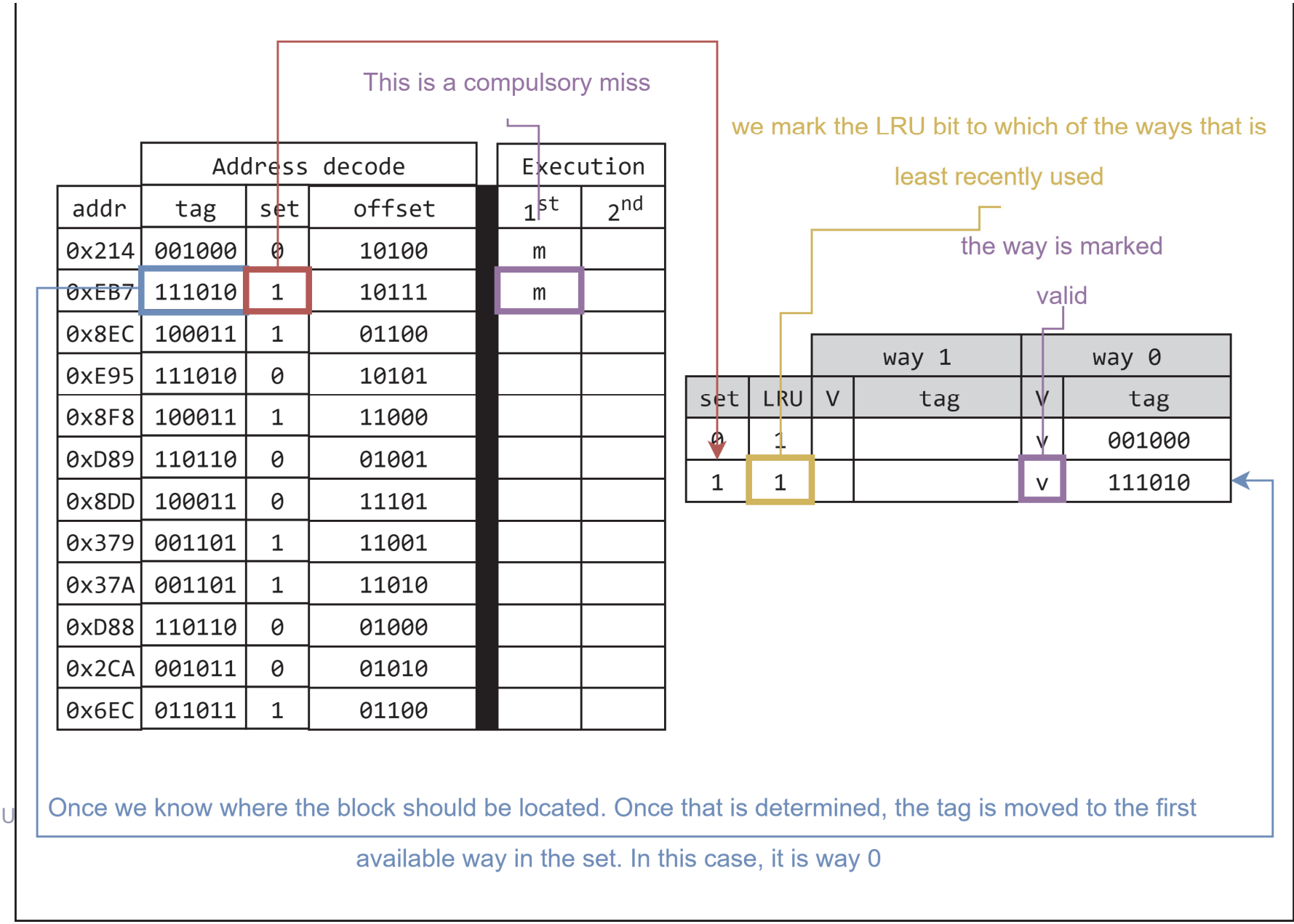
we mark the LRU bit to which of the ways that is least recently used

the way is marked

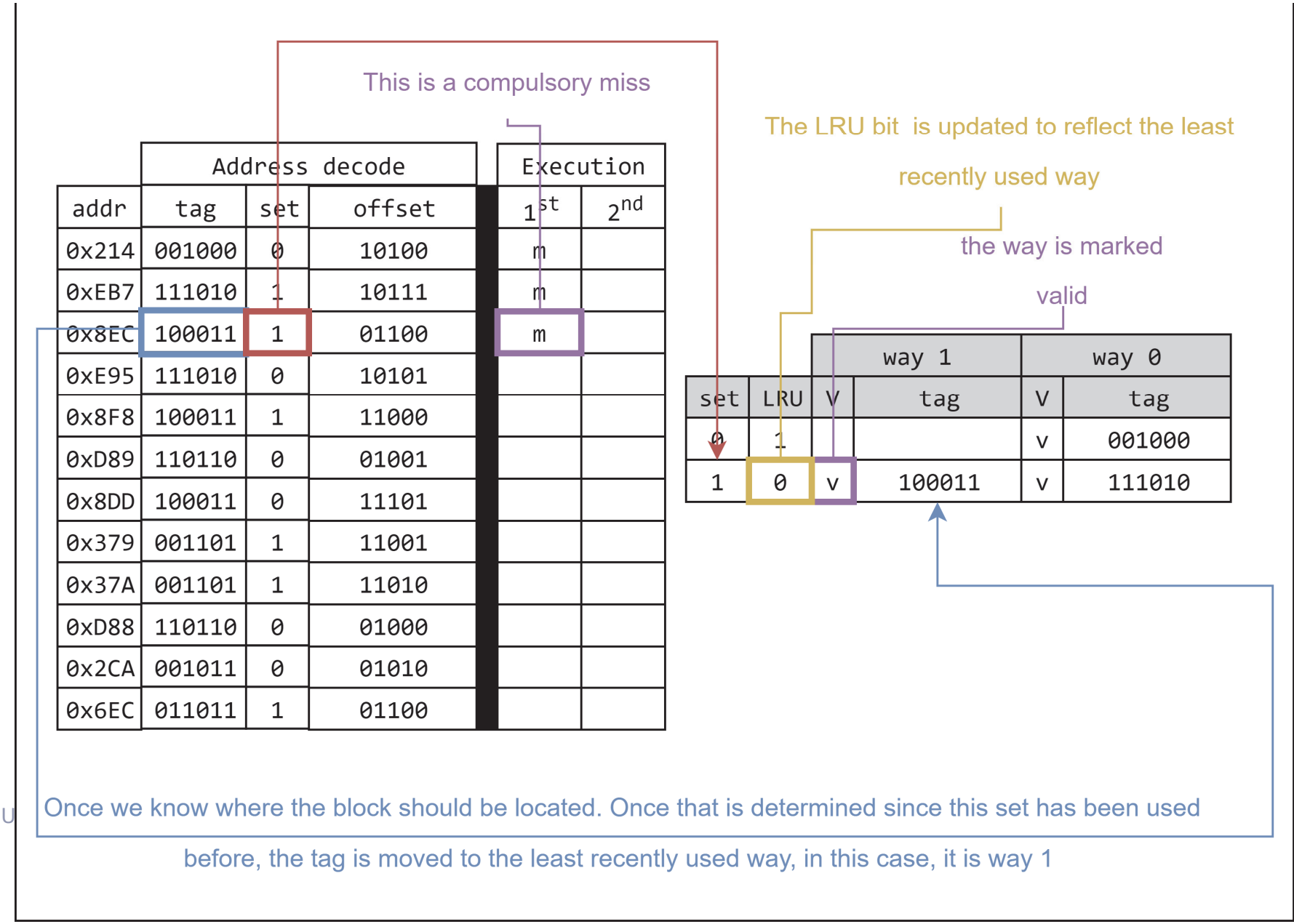
valid

		way 1		way 0	
set	LRU	V	tag	V	tag
0	1			v	001000
1					

MEMORY MANAGEMENT



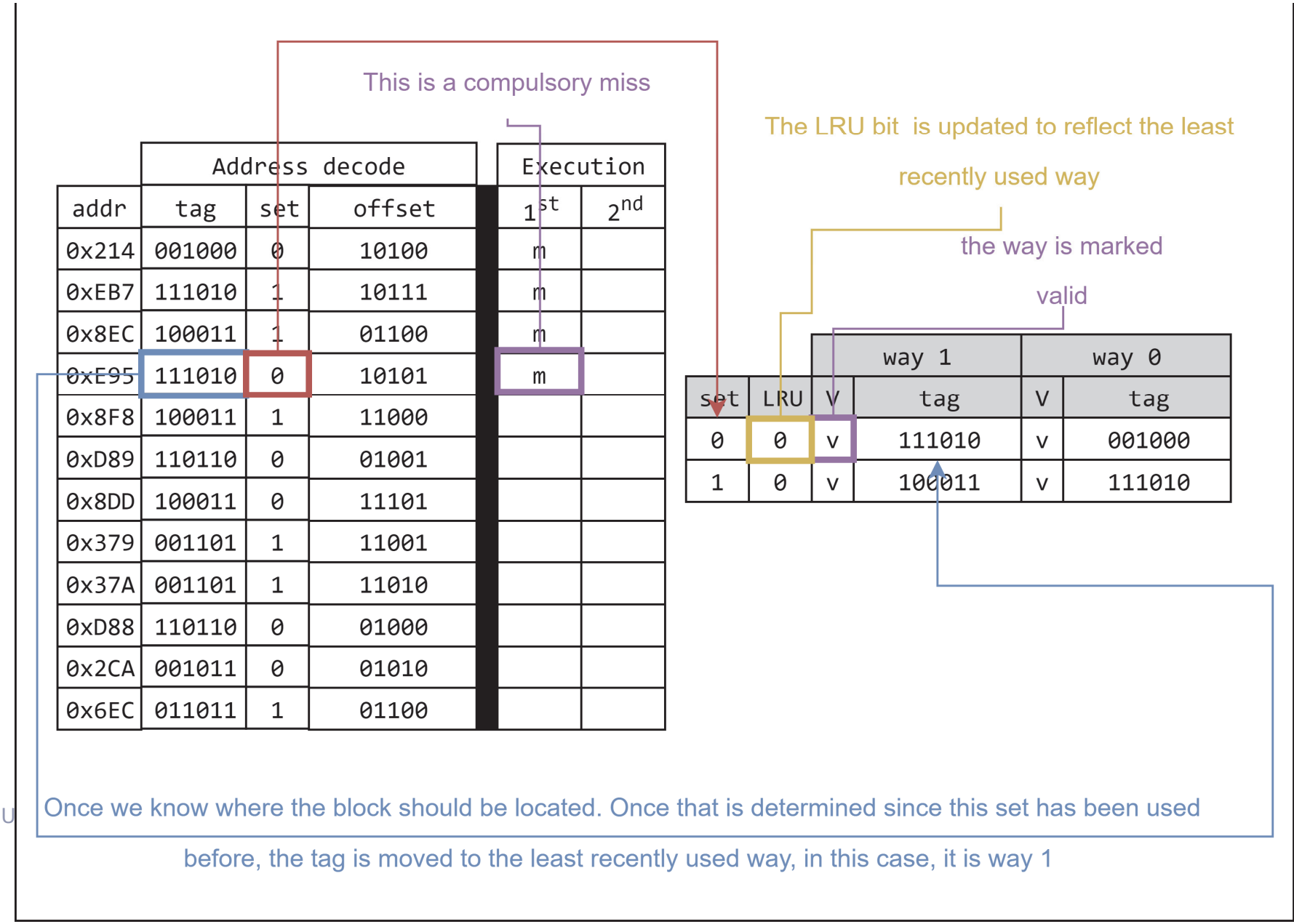
MEMORY MANAGEMENT



	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101		
0x8F8	100011	1	11000		
0xD89	110110	0	01001		
0x8DD	100011	0	11101		
0x379	001101	1	11001		
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0	1			v	001000
1	0	v	100011	v	111010

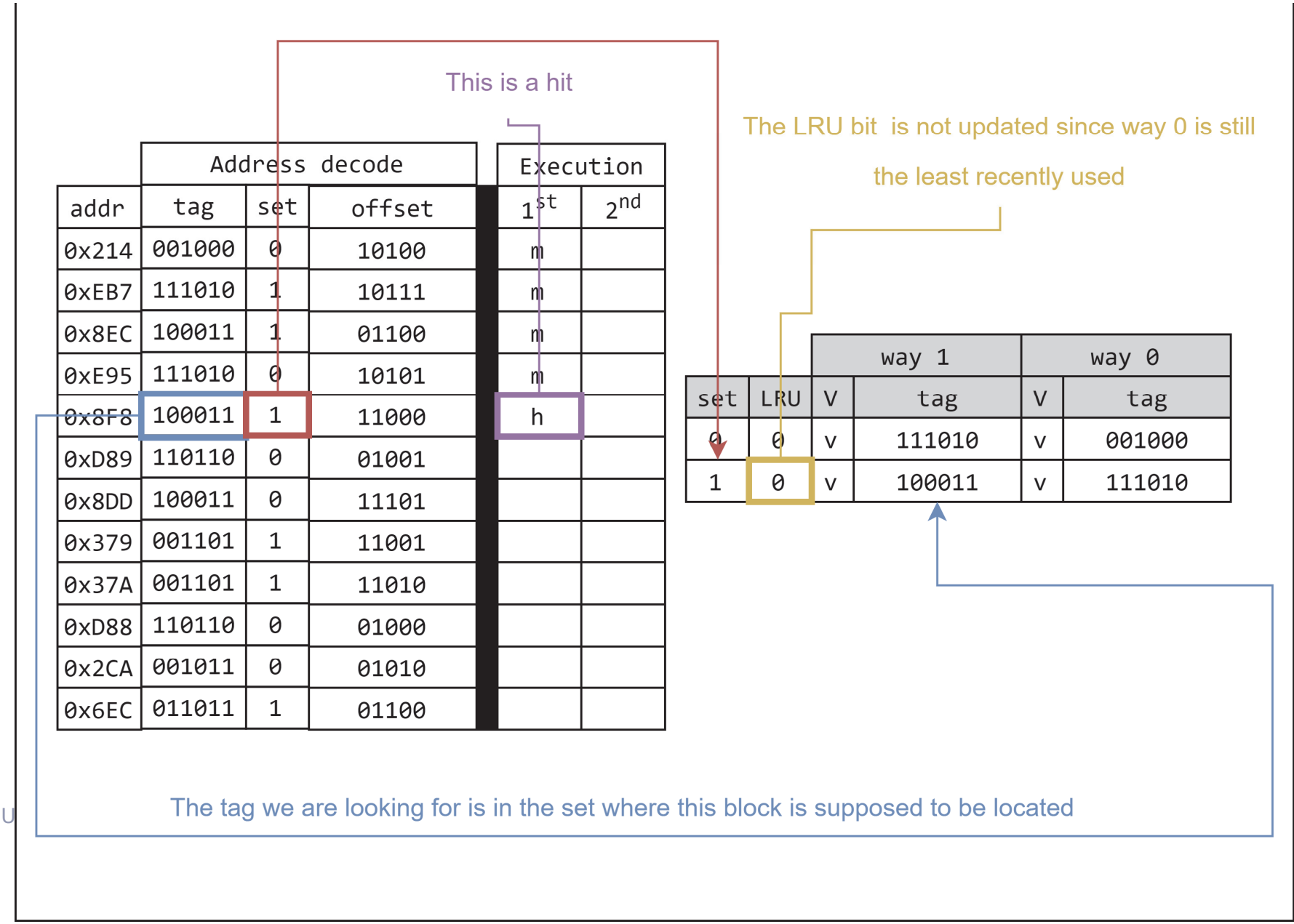
MEMORY MANAGEMENT



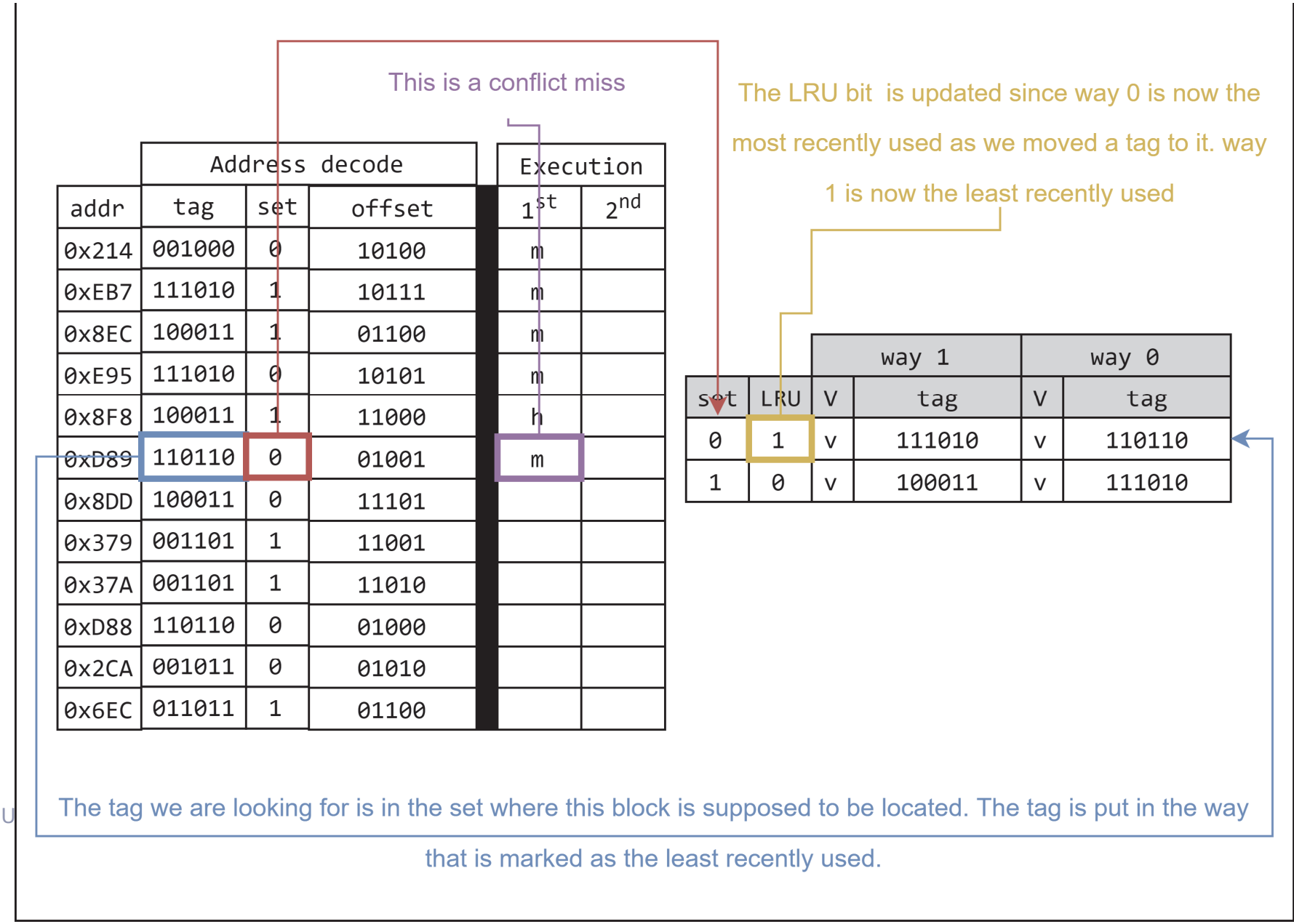
	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101	m	
0x8F8	100011	1	11000		
0xD89	110110	0	01001		
0x8DD	100011	0	11101		
0x379	001101	1	11001		
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0	0	v	111010	v	001000
1	0	v	100011	v	111010

MEMORY MANAGEMENT



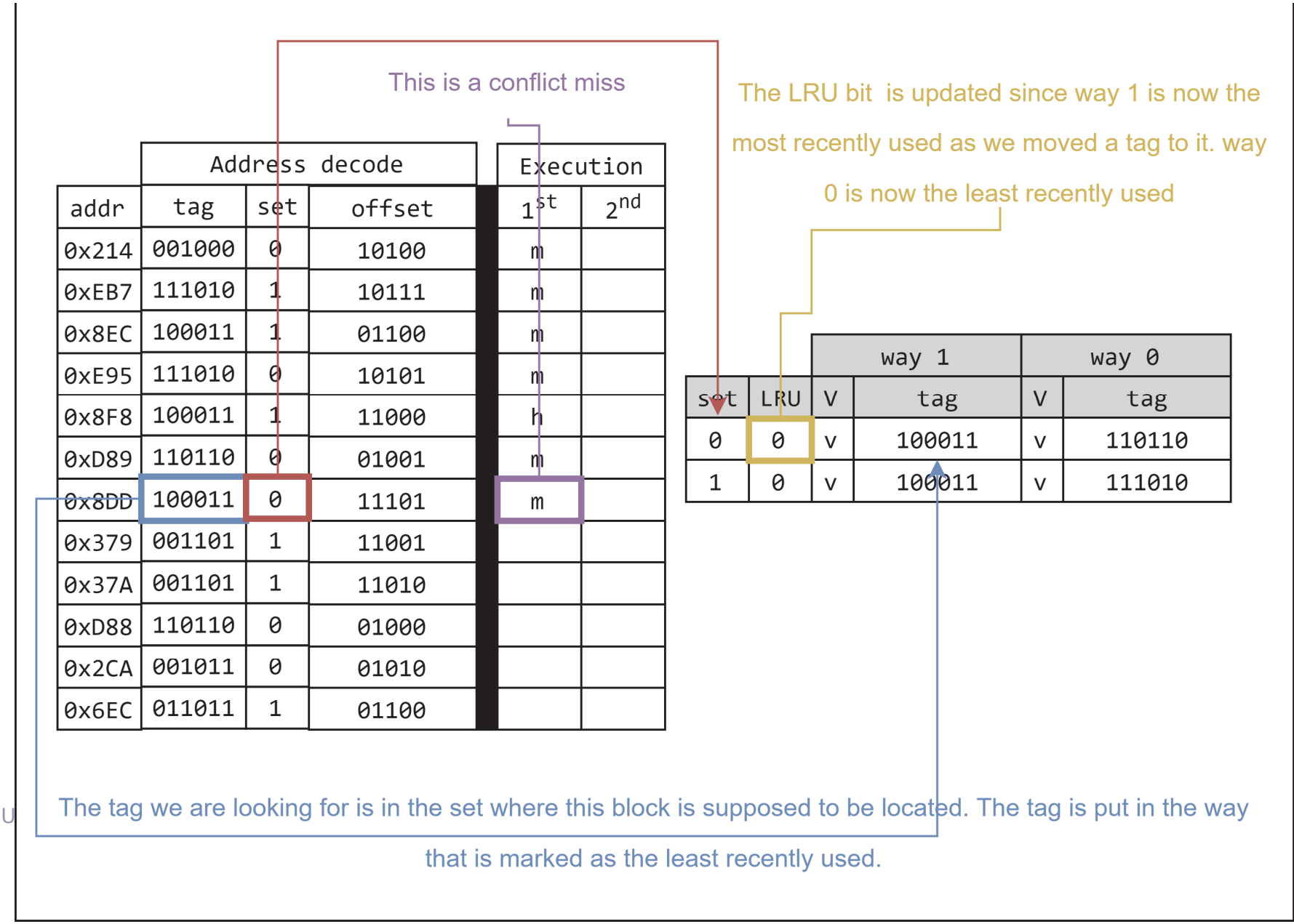
MEMORY MANAGEMENT



	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101	m	
0x8F8	100011	1	11000	h	
0xD89	110110	0	01001	m	
0x8DD	100011	0	11101		
0x379	001101	1	11001		
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0	1	v	111010	v	110110
1	0	v	100011	v	111010

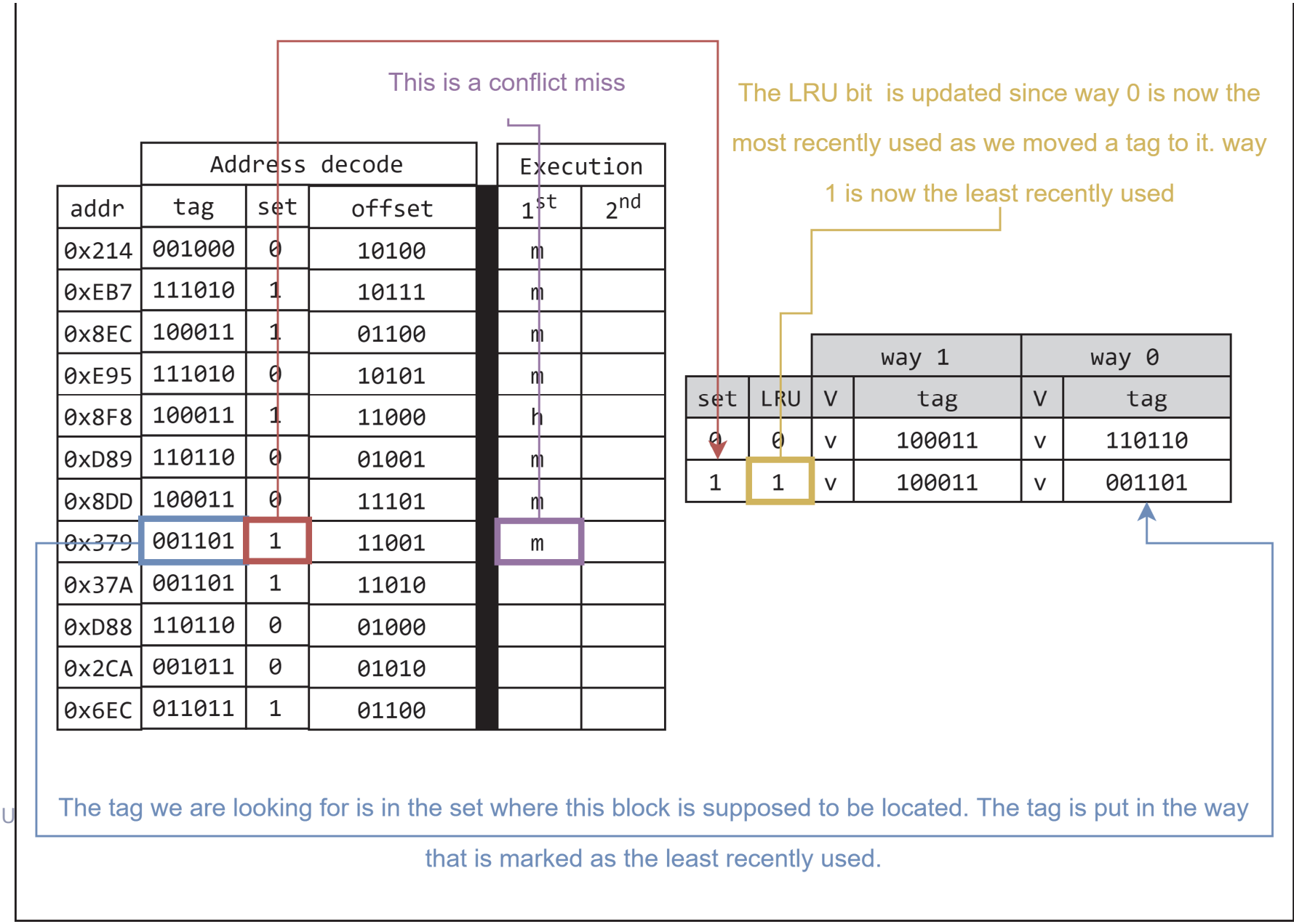
MEMORY MANAGEMENT



	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101	m	
0x8F8	100011	1	11000	h	
0xD89	110110	0	01001	m	
0x8DD	100011	0	11101	m	
0x379	001101	1	11001		
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0	0	v	100011	v	110110
1	0	v	100011	v	111010

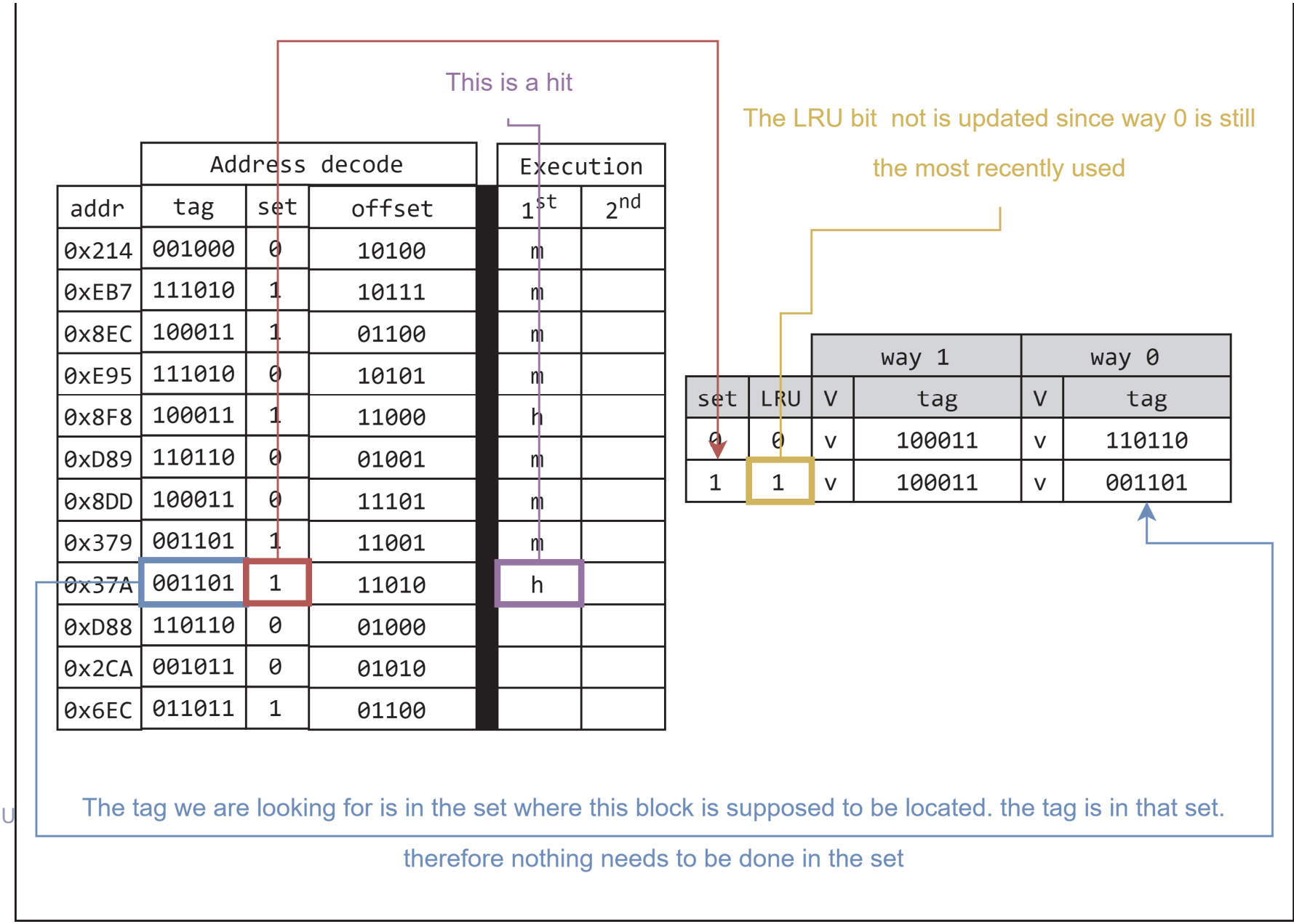
MEMORY MANAGEMENT



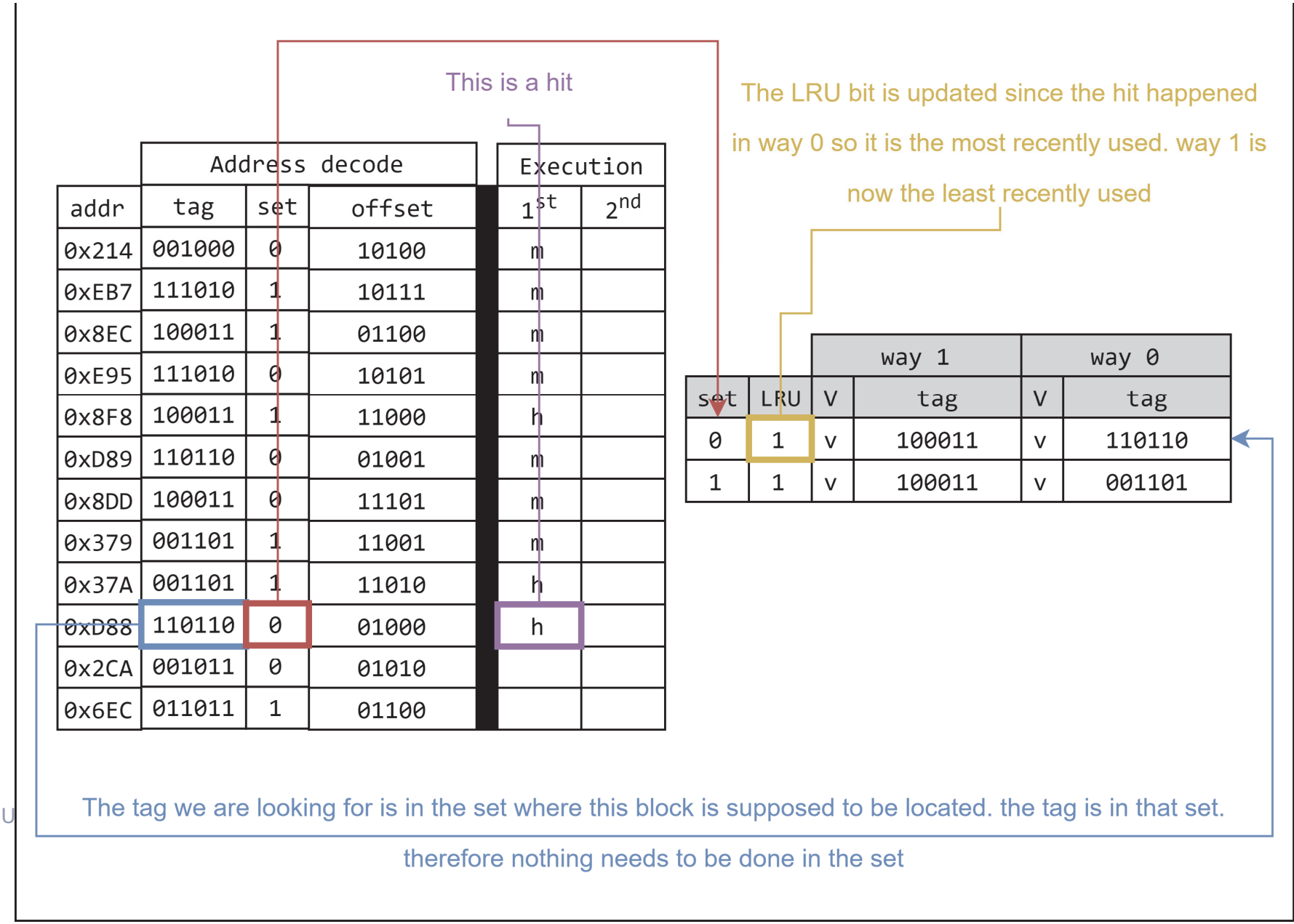
	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101	m	
0x8F8	100011	1	11000	h	
0xD89	110110	0	01001	m	
0x8DD	100011	0	11101	m	
0x379	001101	1	11001	m	
0x37A	001101	1	11010		
0xD88	110110	0	01000		
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0	0	v	100011	v	110110
1	1	v	100011	v	001101

MEMORY MANAGEMENT



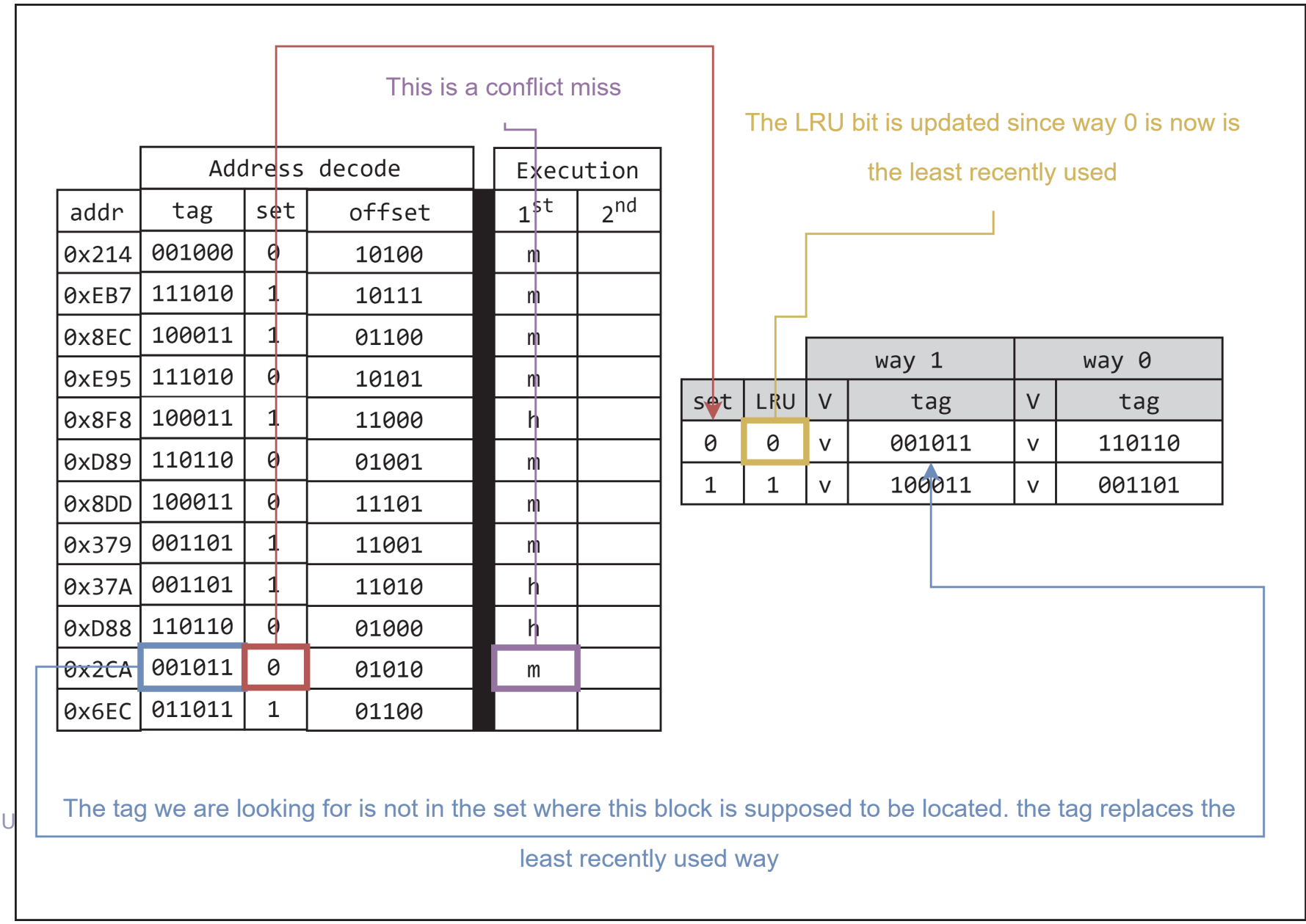
MEMORY MANAGEMENT



	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101	m	
0x8F8	100011	1	11000	h	
0xD89	110110	0	01001	m	
0x8DD	100011	0	11101	m	
0x379	001101	1	11001	m	
0x37A	001101	1	11010	h	
0xD88	110110	0	01000	h	
0x2CA	001011	0	01010		
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0	1	v	100011	v	110110
1	1	v	100011	v	001101

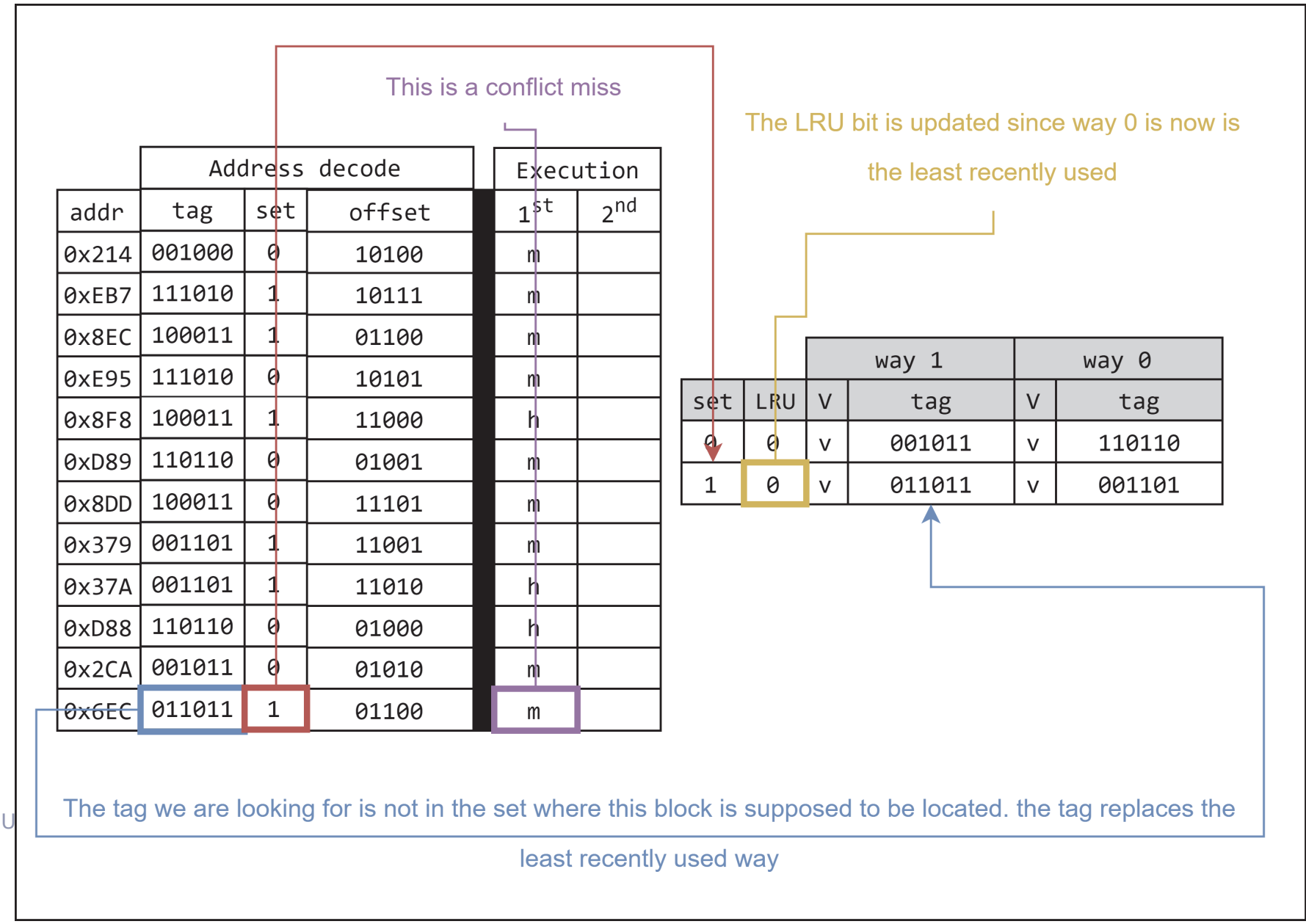
MEMORY MANAGEMENT



	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101	m	
0x8F8	100011	1	11000	h	
0xD89	110110	0	01001	m	
0x8DD	100011	0	11101	m	
0x379	001101	1	11001	m	
0x37A	001101	1	11010	h	
0xD88	110110	0	01000	h	
0x2CA	001011	0	01010	m	
0x6EC	011011	1	01100		

		way 1		way 0	
set	LRU	V	tag	V	tag
0	0	v	001011	v	110110
1	1	v	100011	v	001101

MEMORY MANAGEMENT



	Address decode			Execution	
addr	tag	set	offset	1st	2nd
0x214	001000	0	10100	m	
0xEB7	111010	1	10111	m	
0x8EC	100011	1	01100	m	
0xE95	111010	0	10101	m	
0x8F8	100011	1	11000	h	
0xD89	110110	0	01001	m	
0x8DD	100011	0	11101	m	
0x379	001101	1	11001	m	
0x37A	001101	1	11010	h	
0xD88	110110	0	01000	h	
0x2CA	001011	0	01010	m	
0x6EC	011011	1	01100	m	

		way 1		way 0	
set	LRU	V	tag	V	tag
0	0	v	001011	v	110110
1	0	v	011011	v	001101