15-213 Recitation 11

Introduction to Computer Systems

Fahim Dalvi

14 November, 2013

Malloc Lab

- Due 20th November, 2013
- Submit on autolab constantly!
- Time is running out very soon

Exam

- 20th November, 2013
- The exam covers from Caches to Dynamic memory allocation, inclusive.
- Exam is on the same day as malloc lab
  - Hopefully you have a few late days

Some helpful tools

- What should you do when you get errors?
  - The driver will try to help you, with messages like:
    - `payload address not aligned to 8 bytes`
    - `block i has 2 garbled bytes`
    - `out of memory`
  - Garbled bytes → It means that you have overwritten something my mistake, like the header of the next block or so
  - But sometimes,
    - Segmentation faults → You know these well, use `gdb`

Some helpful tools

- `gdb`
  - `break expr`
  - `watch expr`
    - eg. `watch *0x11223344`
    - Breaks program execution when `0x11223344` is modified
  - `rwatch expr`
    - Same as watch, but breaks even if the location is read
  - `awatch expr`
    - Breaks if the location is read or modified

Heap corruption

- Again, this is key to avoid many, many problems
- A lot of the time, your heap is corrupted.
  - How can you check what's going wrong in the heap?
    - Print the entire heap
    - But this is going to get very tiring, very soon
    - The amount of data you have to parse through is very high

Heap corruption

- Again, this is key to avoid many, many problems
- A lot of the time, your heap is corrupted.
  - How can you check what's going wrong in the heap?
    - mm_checkheap
    - Call it every time before you modify the heap
    - Your program might actually `seg fault` much after your corrupted your heap, hence its important that you check for the correctness of the heap at every step

mm_checkheap

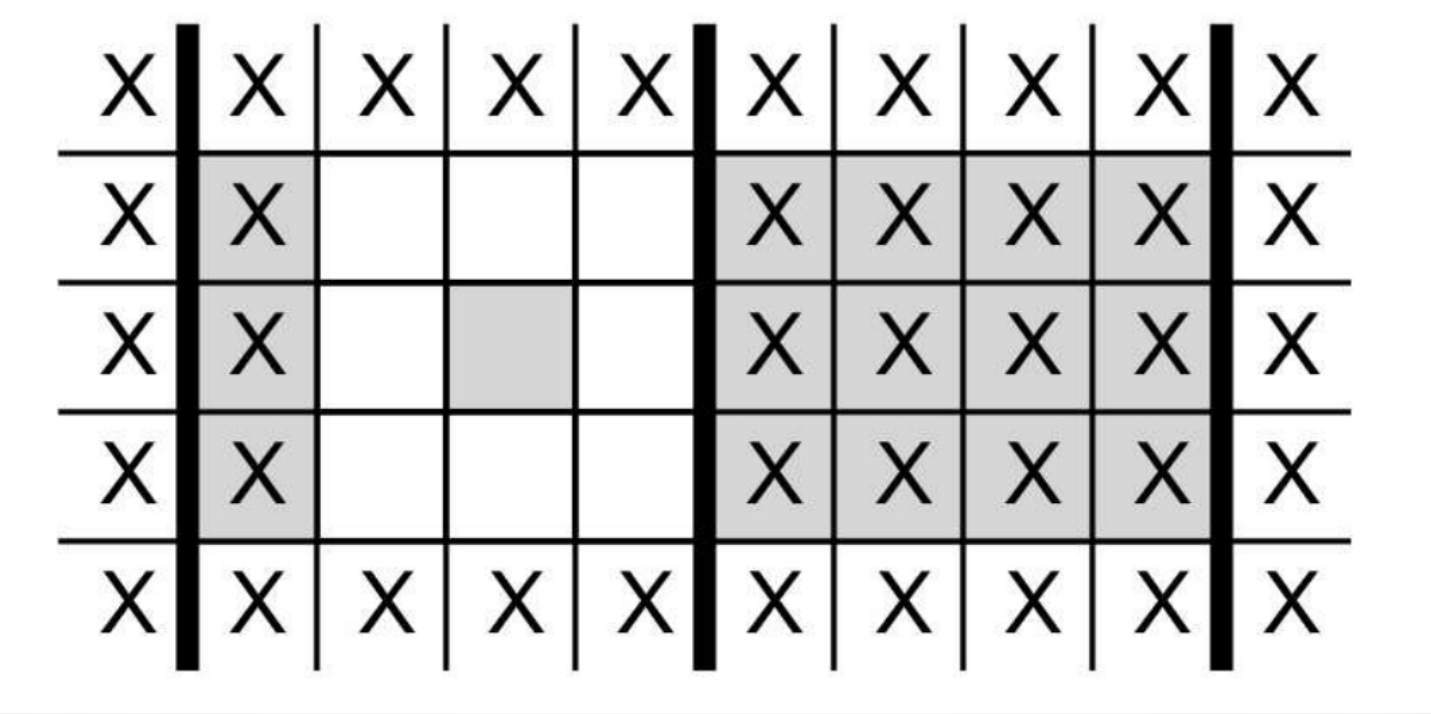
- Block level:
  - Header and footer match
  - Payload data is aligned
- List level:
  - Next/prev pointers in consecutive free blocks are consistent
  - Free list contains no allocated blocks
  - All free blocks are in the free list
  - No contiguous free blocks in memory (unless you defer coalescing)
  - There are no cycles in the list
  - Segregated lists contain only blocks that are of the correct size class

mm_checkheap

- Heap level:
  - Check if your prologue and epilogue blocks are correctly implemented
  - All allocated/free blocks should be between these at any point of time
- Any more suggestions?

Any Questions?

Lets refresh your caches!



http://www.qatar.cmu.edu/~kharras/courses/15213-f13/oldexams/exam2-f03.pdf
Question 3

Lets refresh your caches!

- 16 kB cache
- 16 bytes lines
- N x N array
- $N = 2^k$ for some k
- The entire cache cannot even hold one complete row
- Hence, the total lines in the cache is 1024
- And $N > 1024$

Lets refresh your caches!

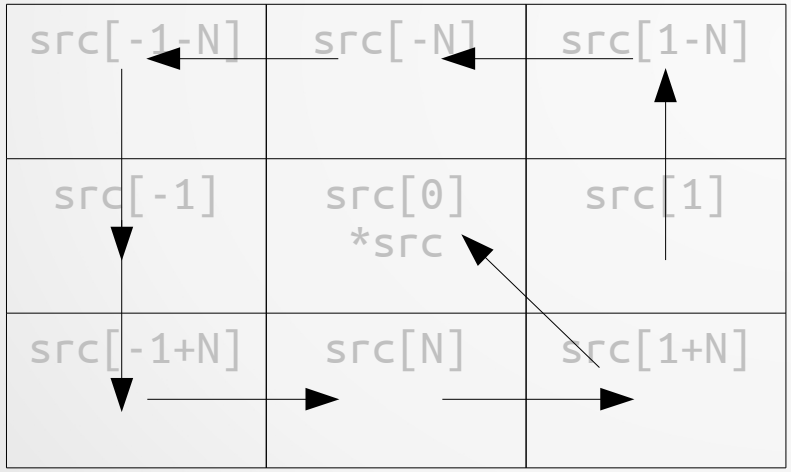
- Notice the memory accesses:

src[-1-N]	src[-N]	src[1-N]
src[-1]	src[0] *src	src[1]
src[-1+N]	src[N]	src[1+N]

```
n = src[ 1 ];
n += src[ 1 - N];
n += src[ - N];
n += src[-1 - N];
n += src[-1 ];
n += src[-1 + N];
n += src[ N];
n += src[ 1 + N];
/* update the next generation */
*dst = (((*src != 0) && (n == 2))
|| (n == 3)) ? 1 : 0;
```

Lets refresh your caches!

- Notice the memory accesses:

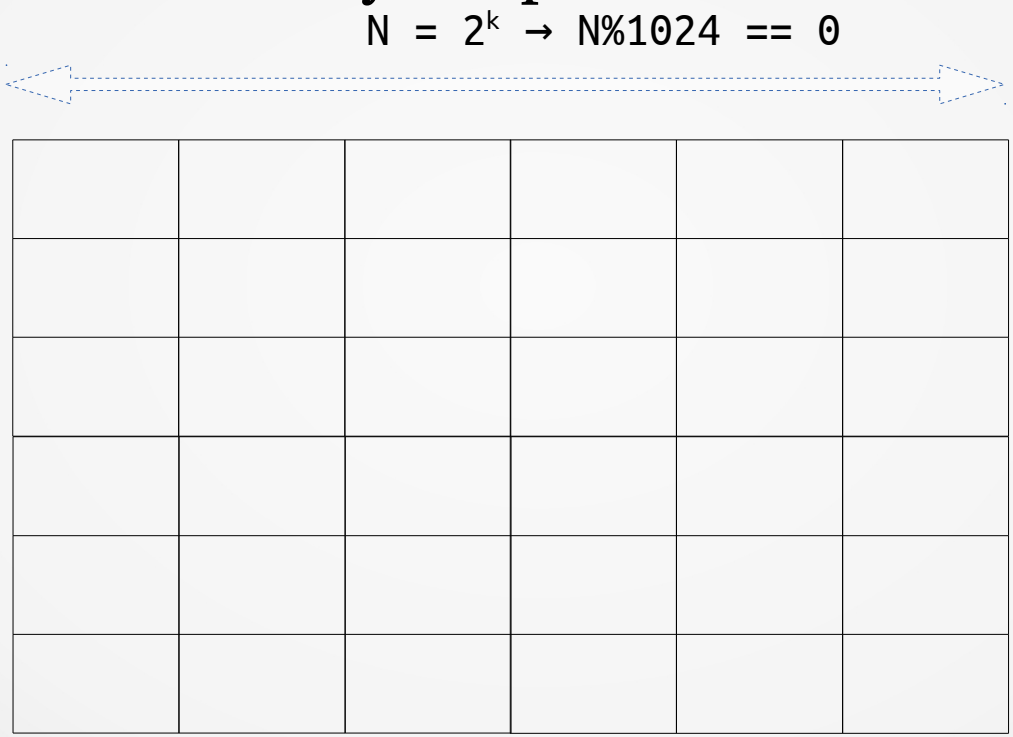



```
n = src[ 1 ];
n += src[ 1 - N];
n += src[ - N];
n += src[-1 - N];
n += src[-1 ];
n += src[-1 + N];
n += src[ N];
n += src[ 1 + N];
/* update the next generation */
*dst = (((*src != 0) && (n == 2))
|| (n == 3)) ? 1 : 0;
```

- Note that the last access in every iteration is *src

Lets refresh your caches!

- Lets see how the array maps in the cache

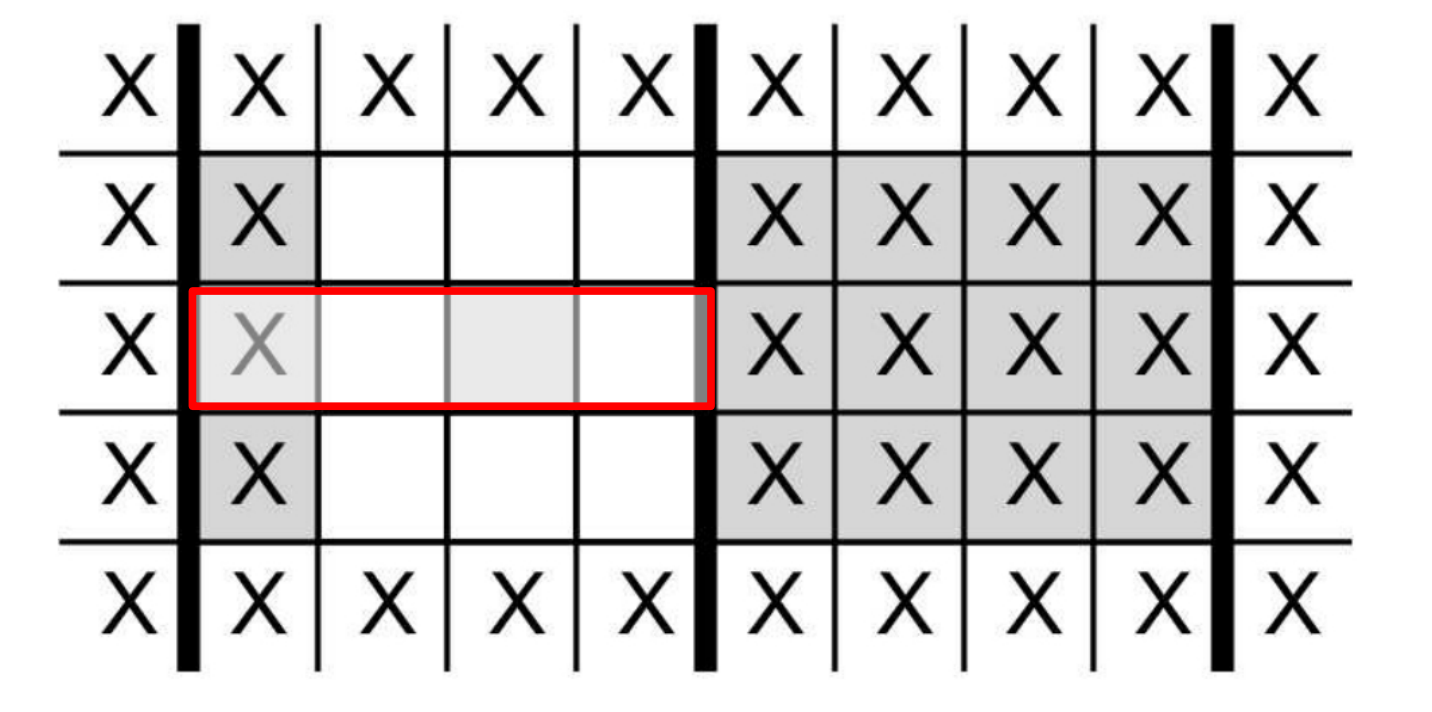


- Every column cell maps to the same set in the cache

Lets refresh your caches!

- Part 1
- From the last iteration, the cache has the middle line, as *src (Now src[-1] was accessed last)

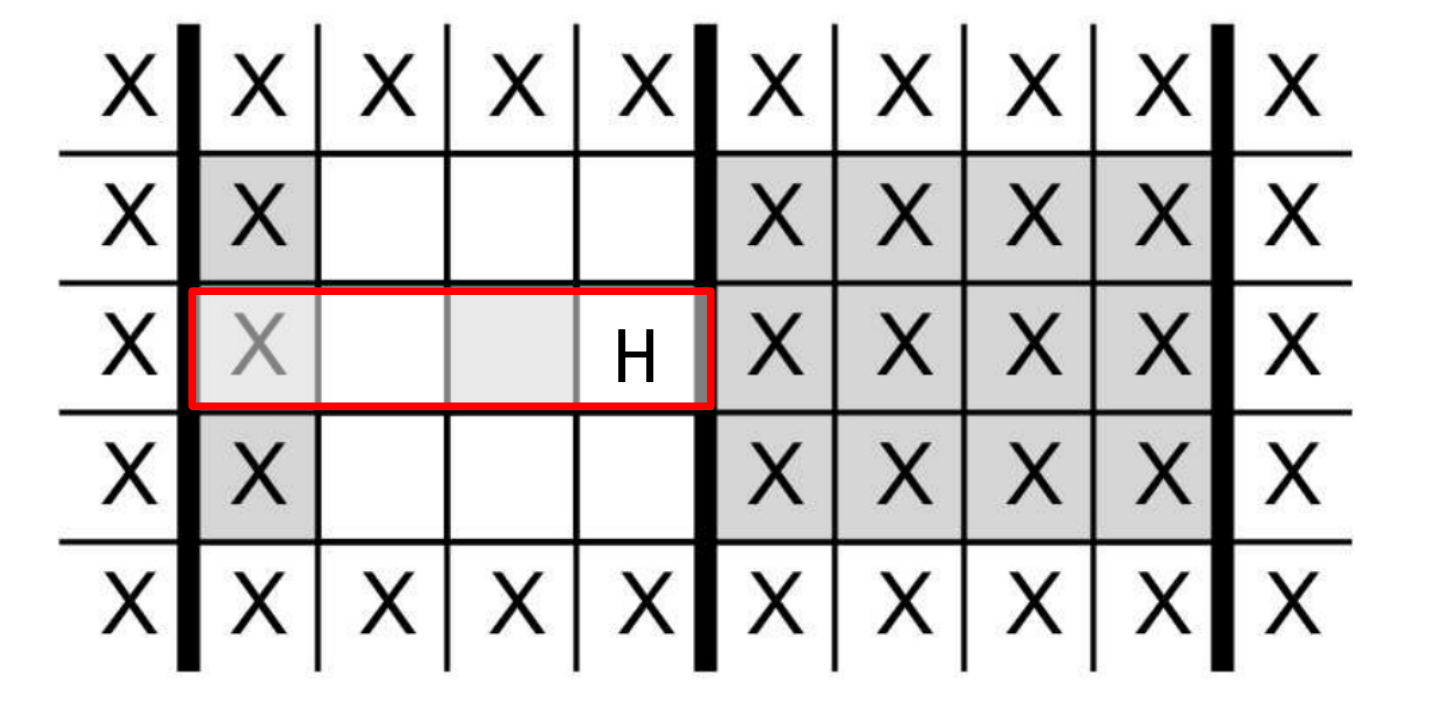
```
Set i's content
```



Lets refresh your caches!

- Part 1

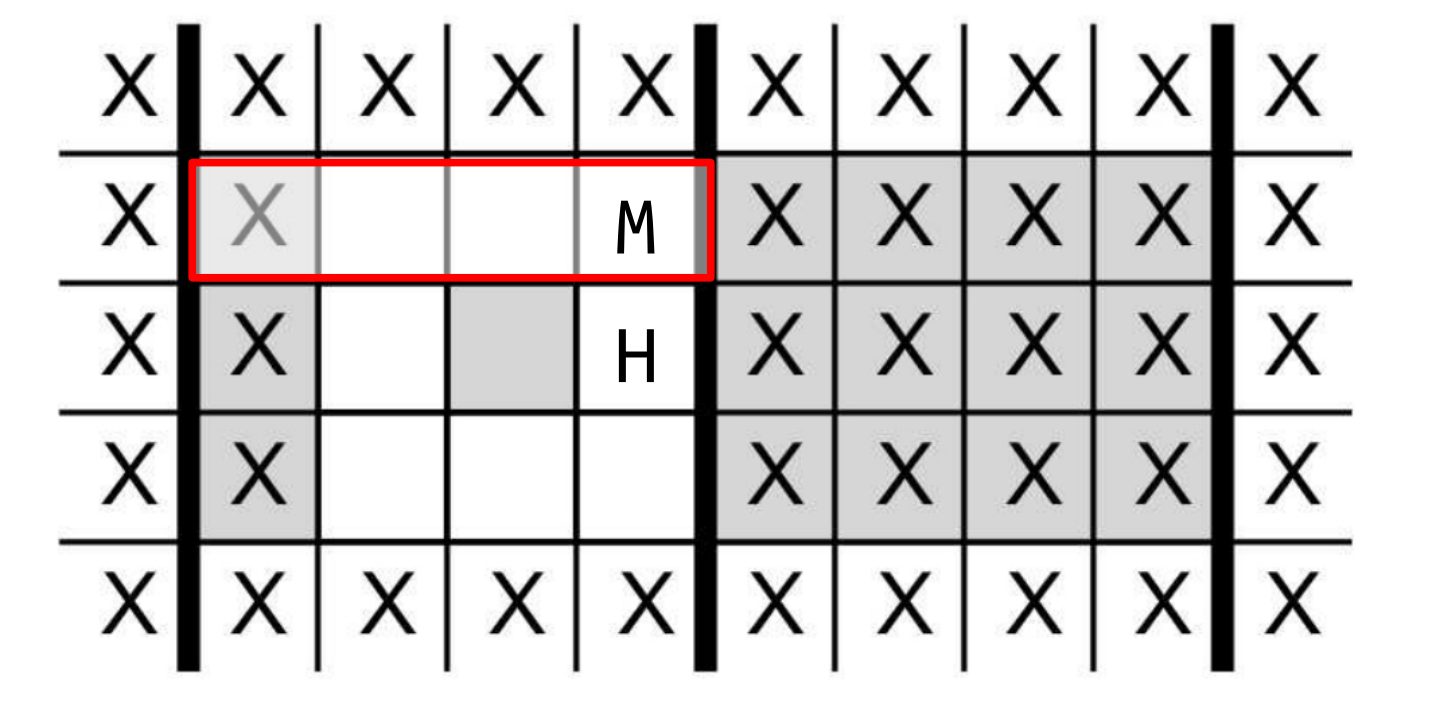
```
Set i's content
```



Lets refresh your caches!

- Part 1

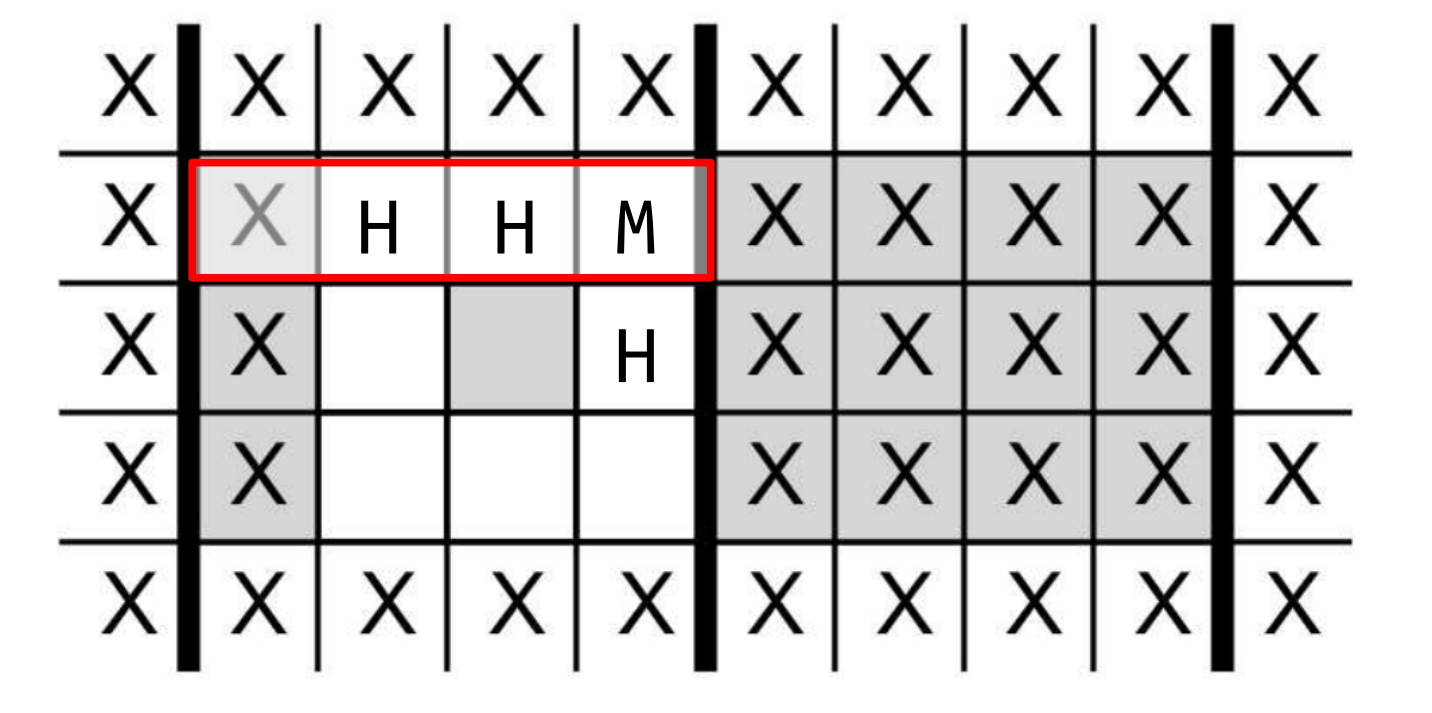
```
Set i's content
```



Lets refresh your caches!

- Part 1

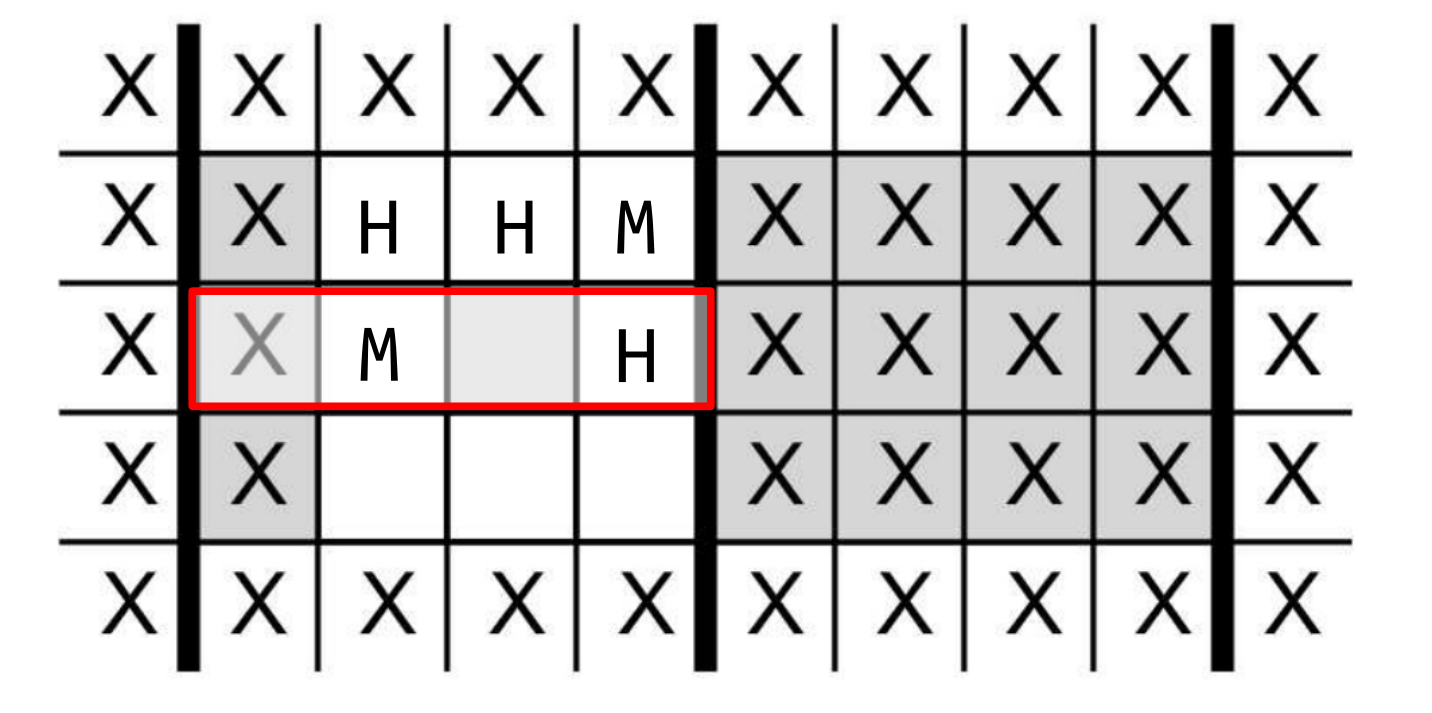
```
Set i's content
```



Lets refresh your caches!

- Part 1

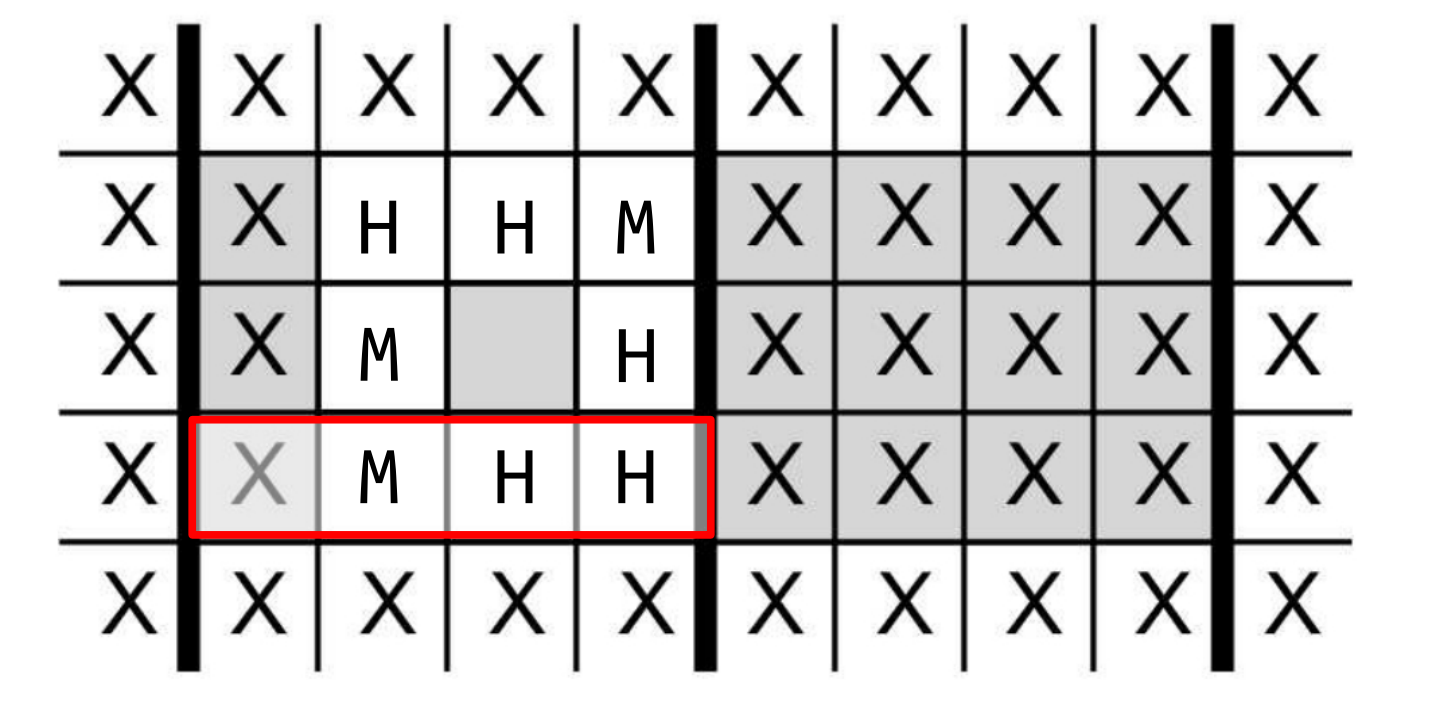
```
Set i's content
```



Lets refresh your caches!

- Part 1

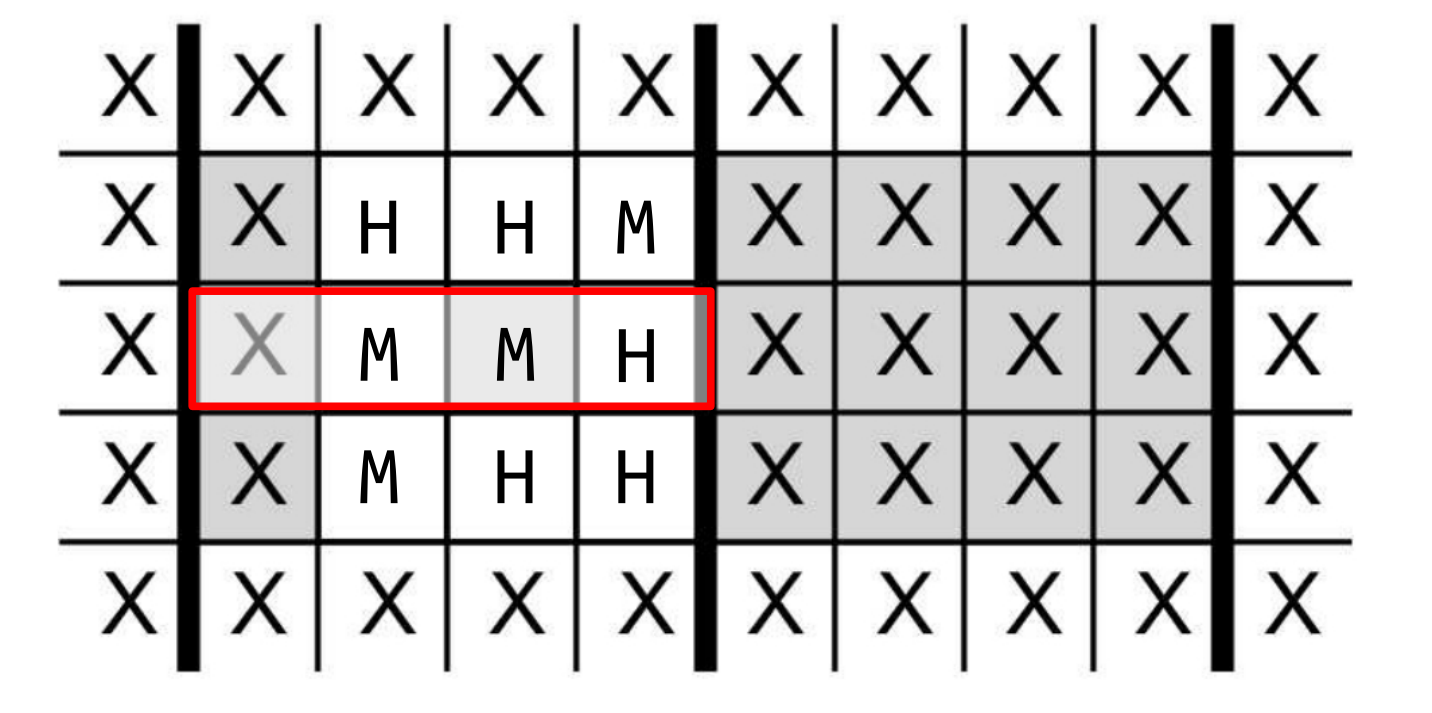
```
Set i's content
```



Lets refresh your caches!

- Part 1

```
Set i's content
```

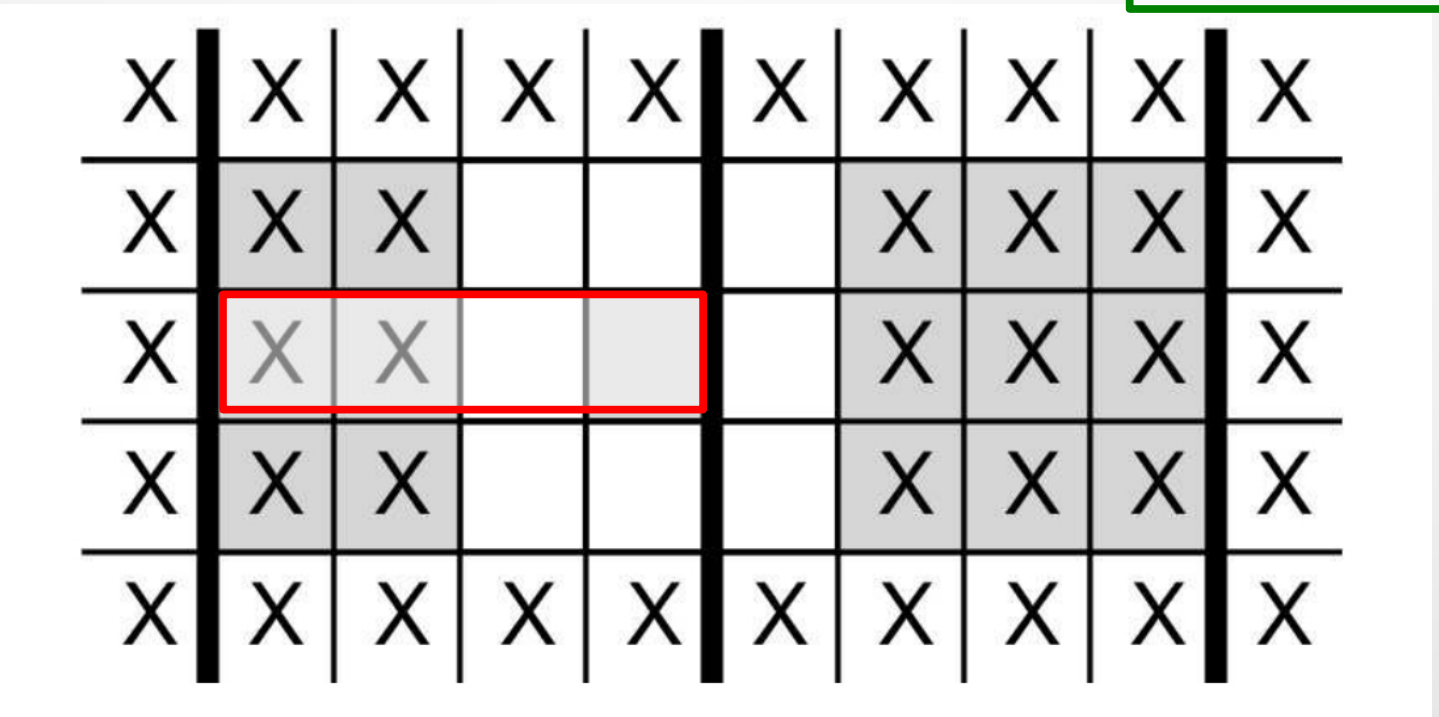


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

Set (i+1)'s content

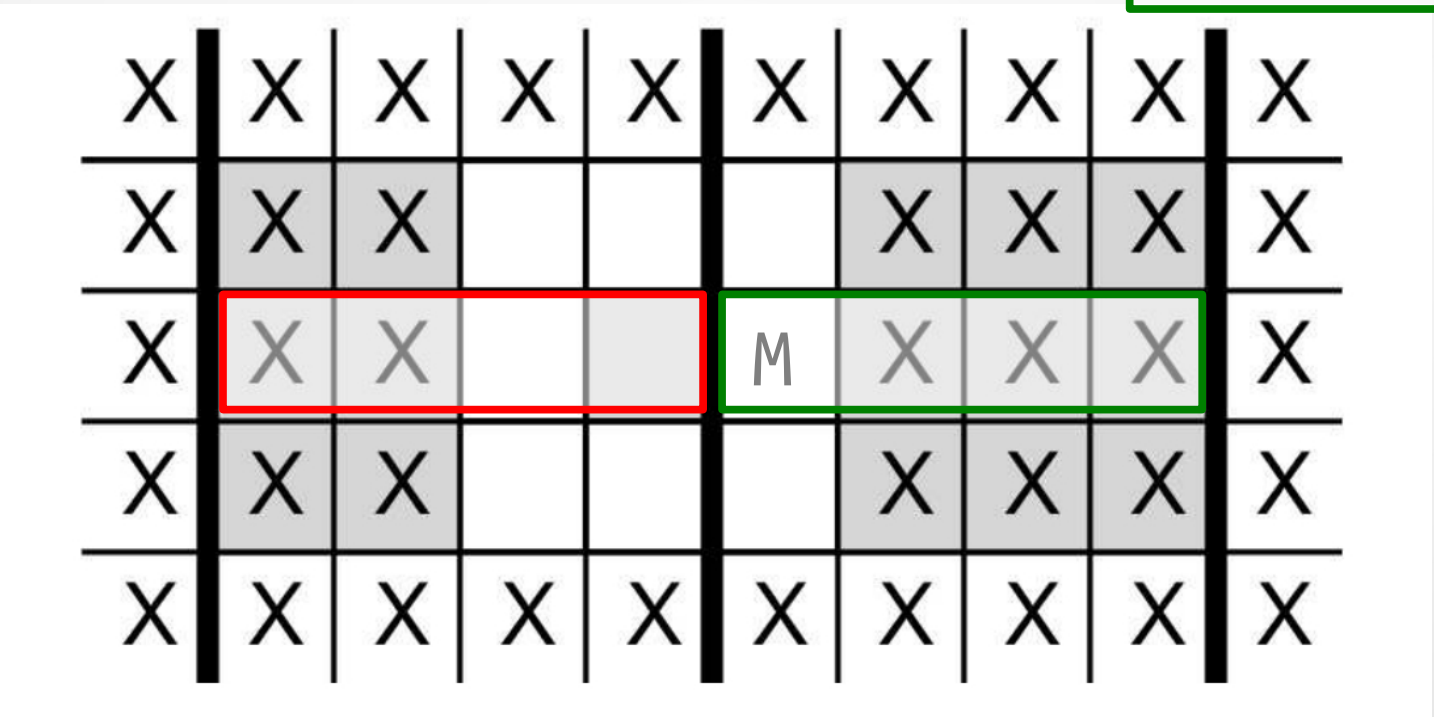


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

Set (i+1)'s content

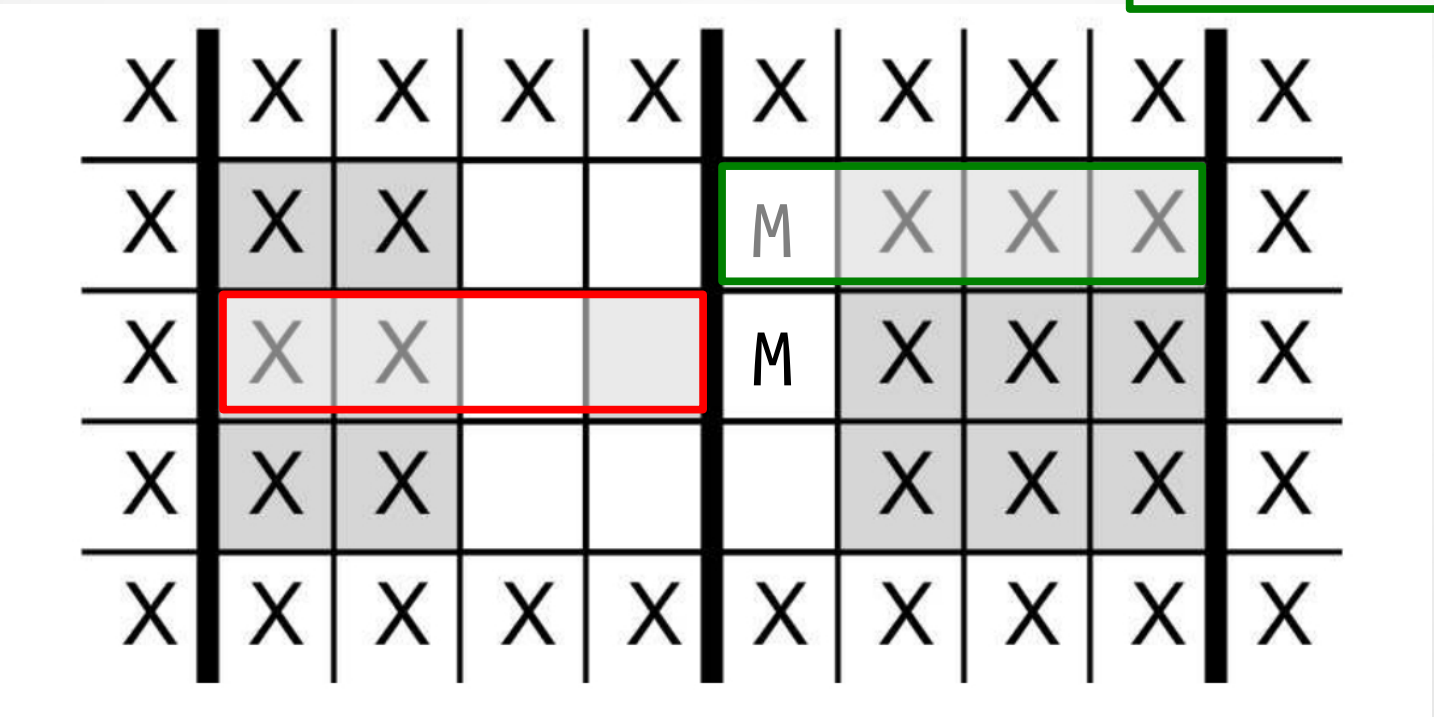


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

Set (i+1)'s content

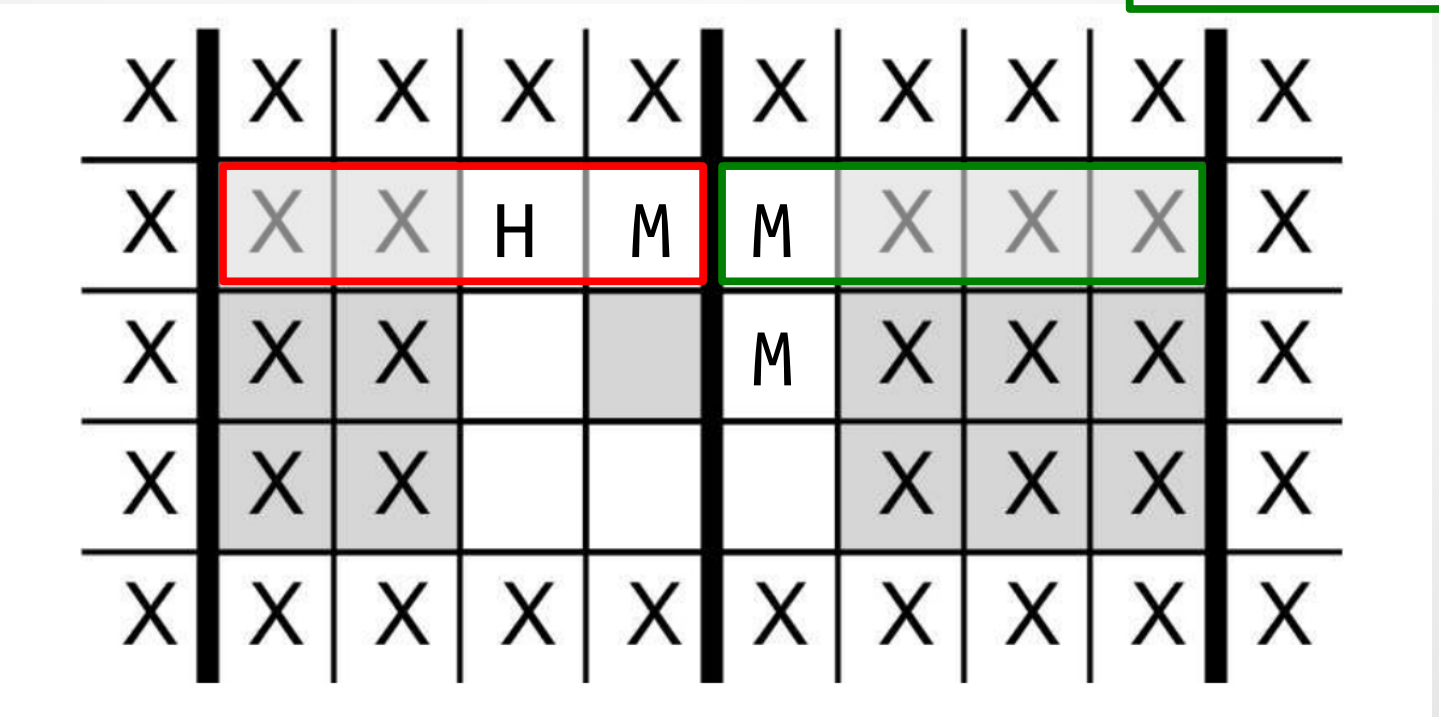


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

Set (i+1)'s content

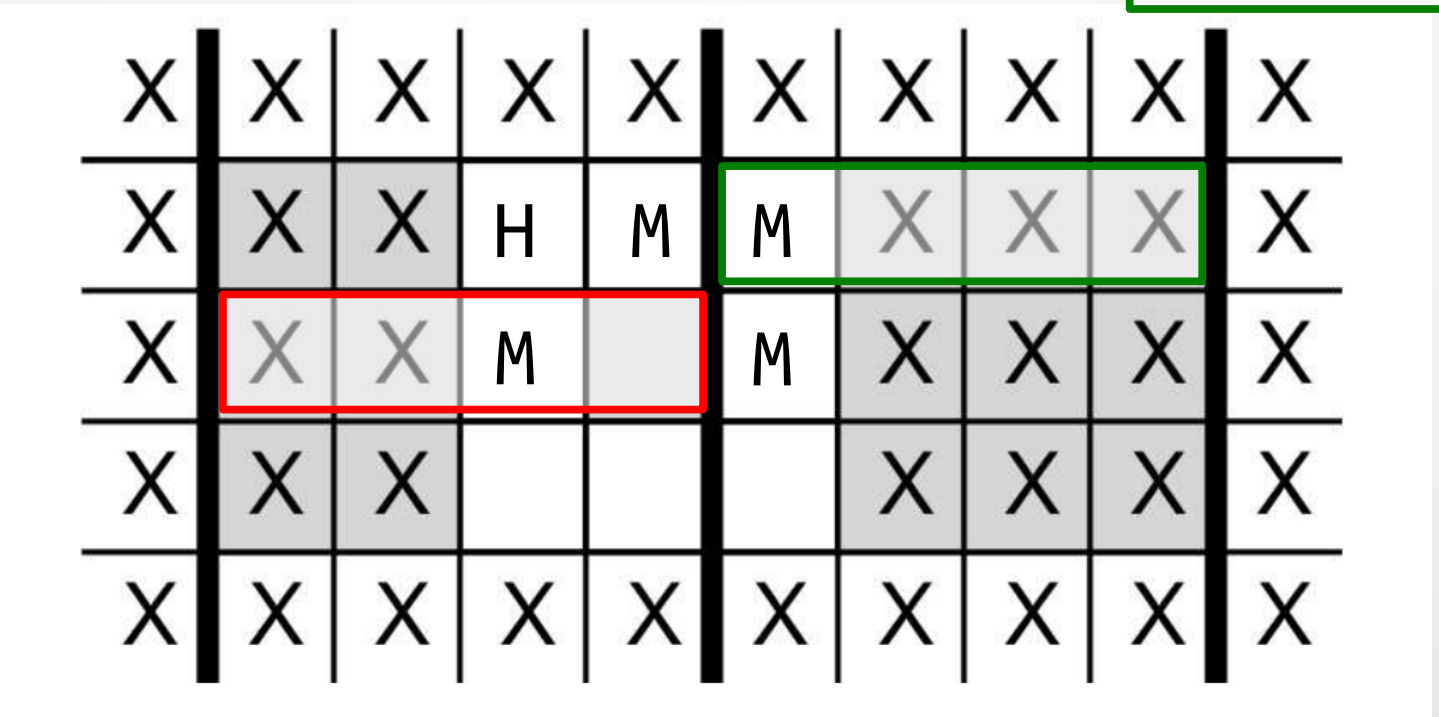


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

Set (i+1)'s content

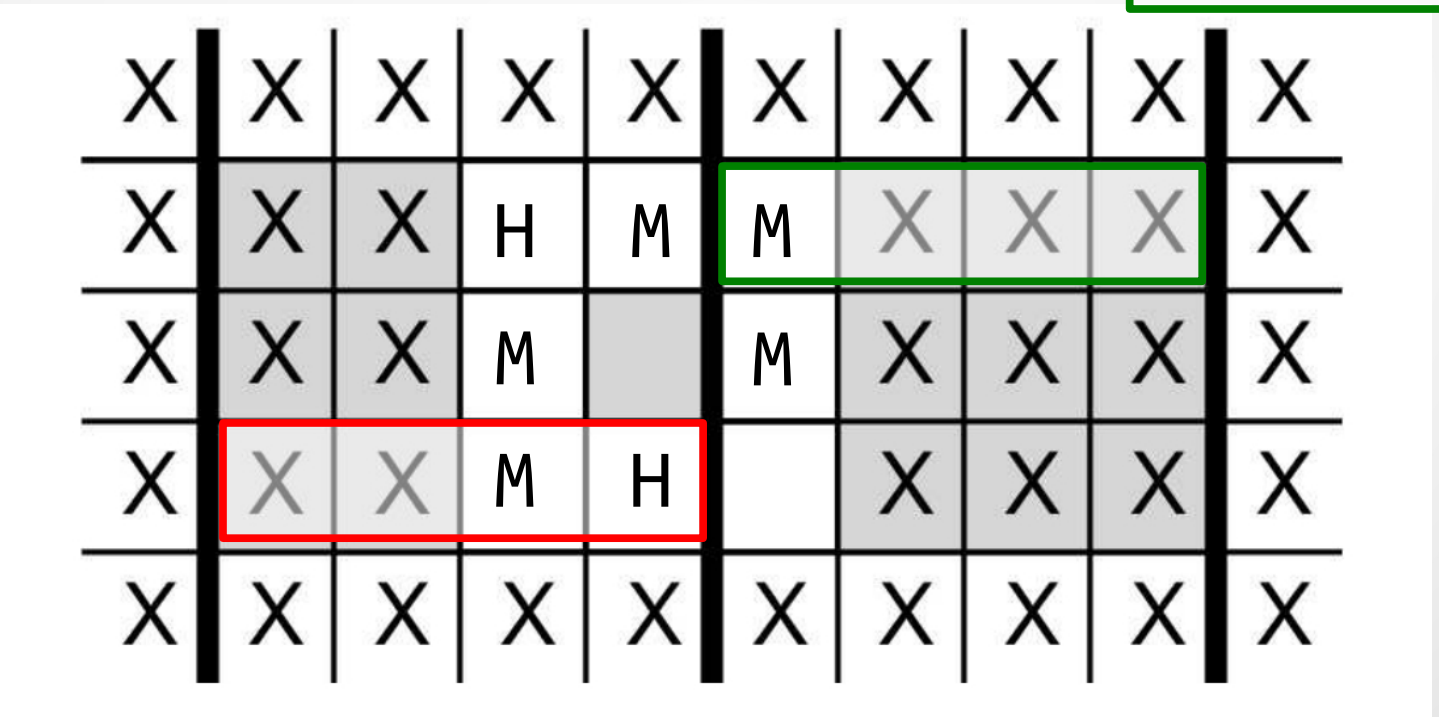


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

Set (i+1)'s content

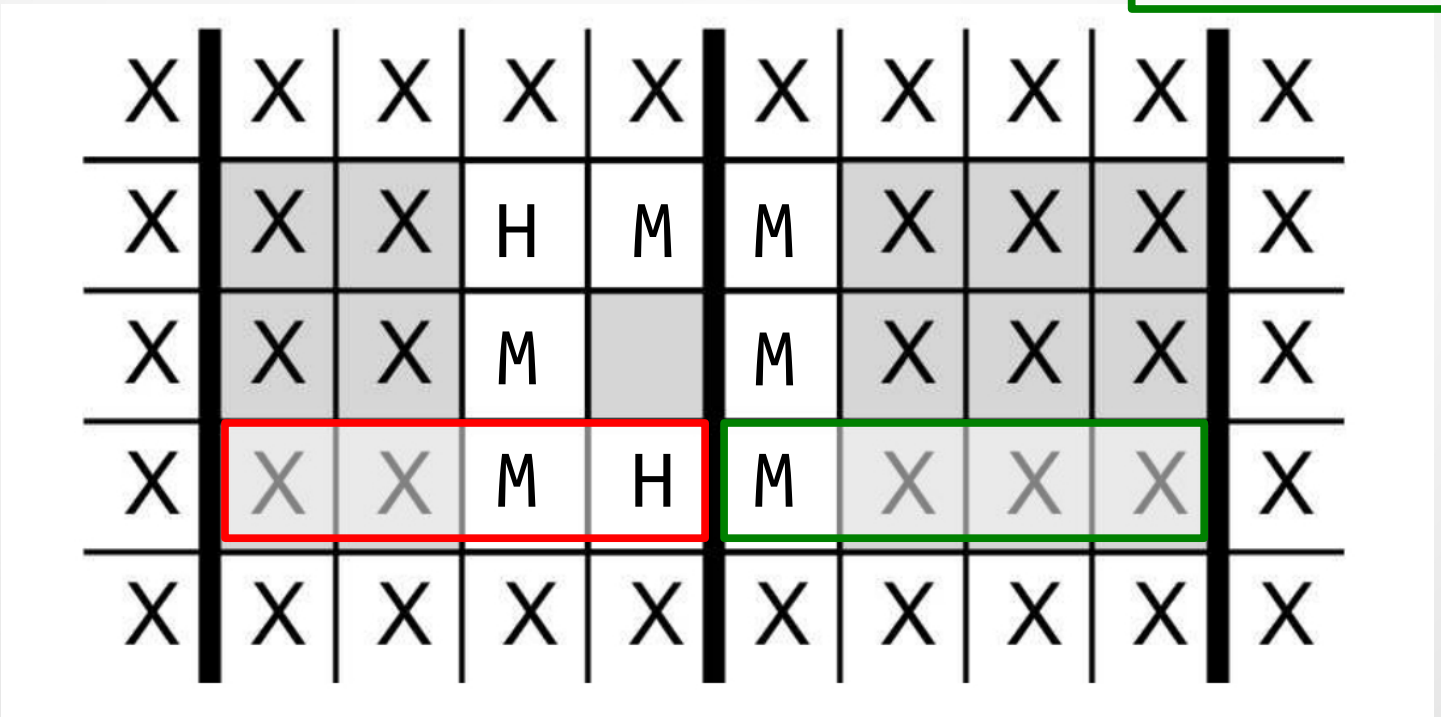


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

Set (i+1)'s content

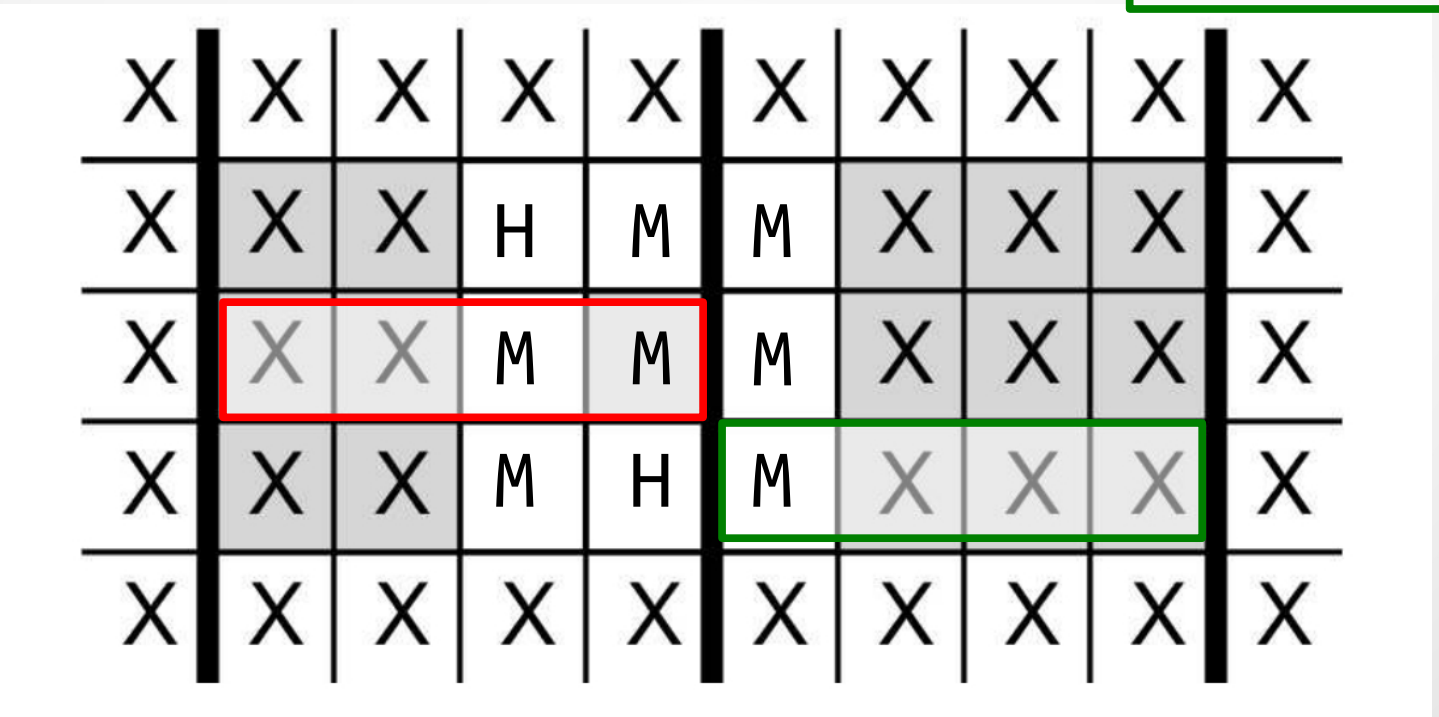


Lets refresh your caches!

- Part 1: Iteration 2

Set i's content

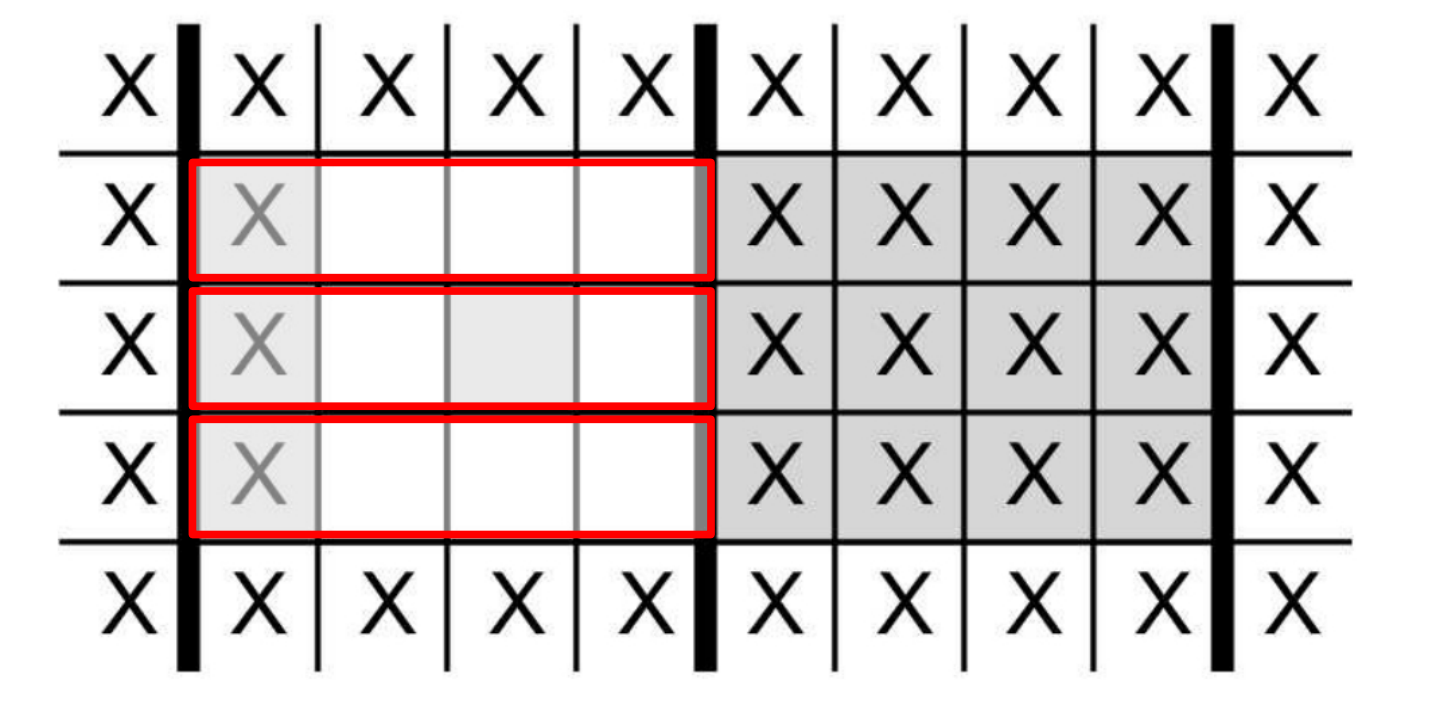
Set (i+1)'s content



Lets refresh your caches!

- Part 2
- 3-Way associative, so we have all the lines already in our cache from last iteration

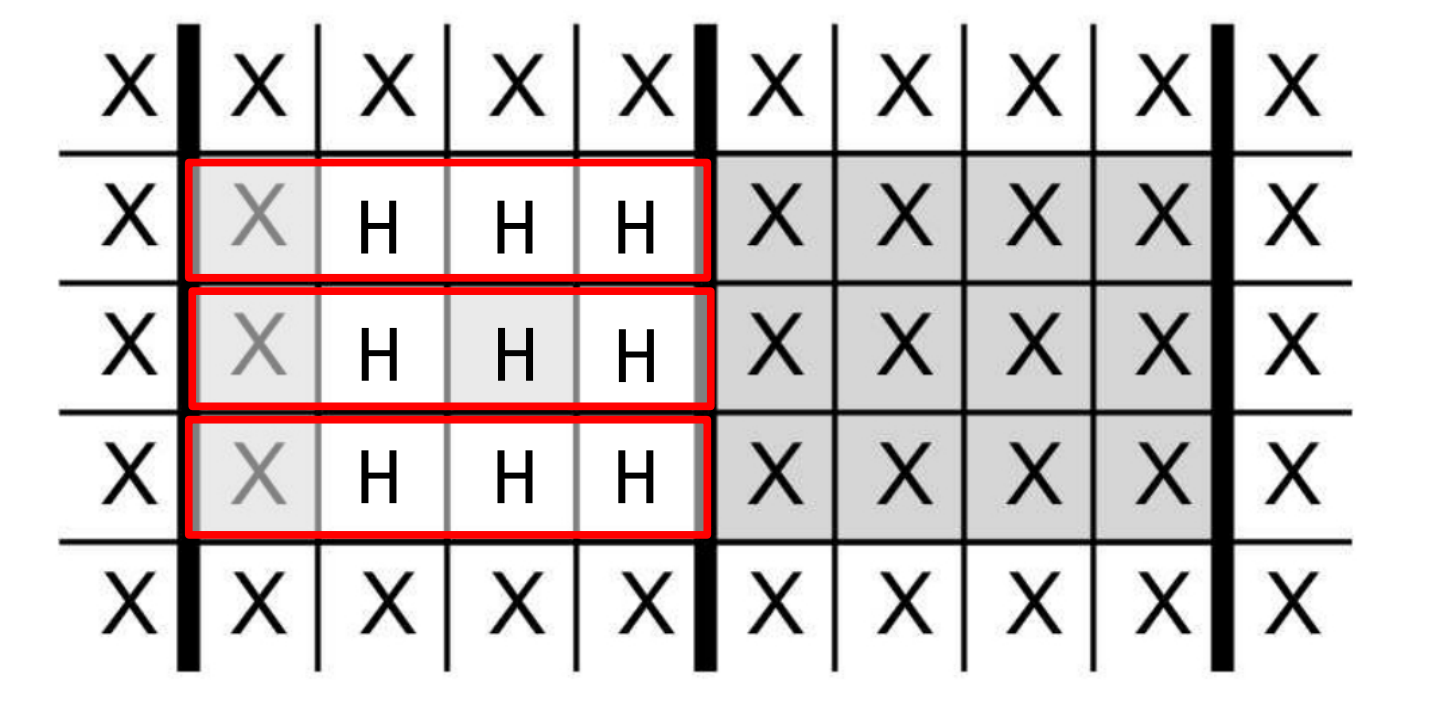
```
Set i's content
```



Lets refresh your caches!

- Part 2

```
Set i's content
```

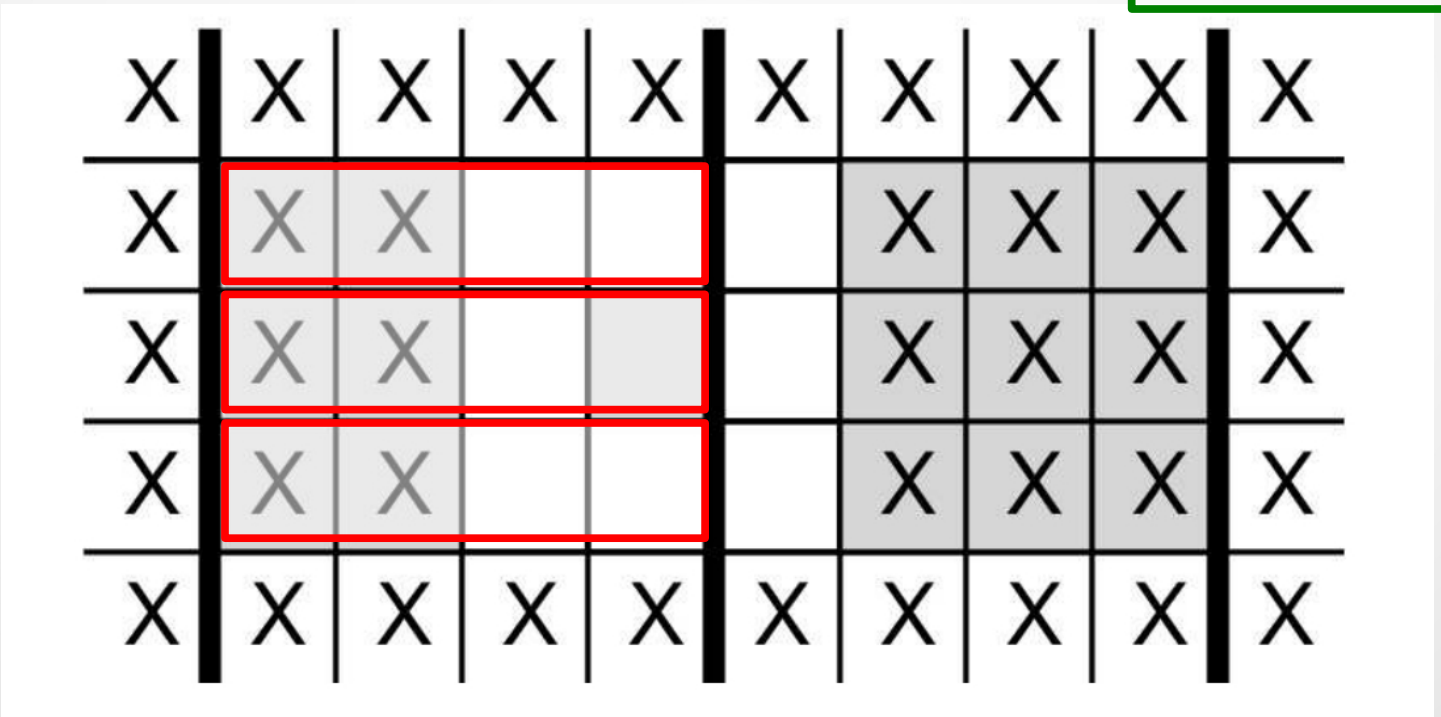


Lets refresh your caches!

- Part 2: Iteration 2

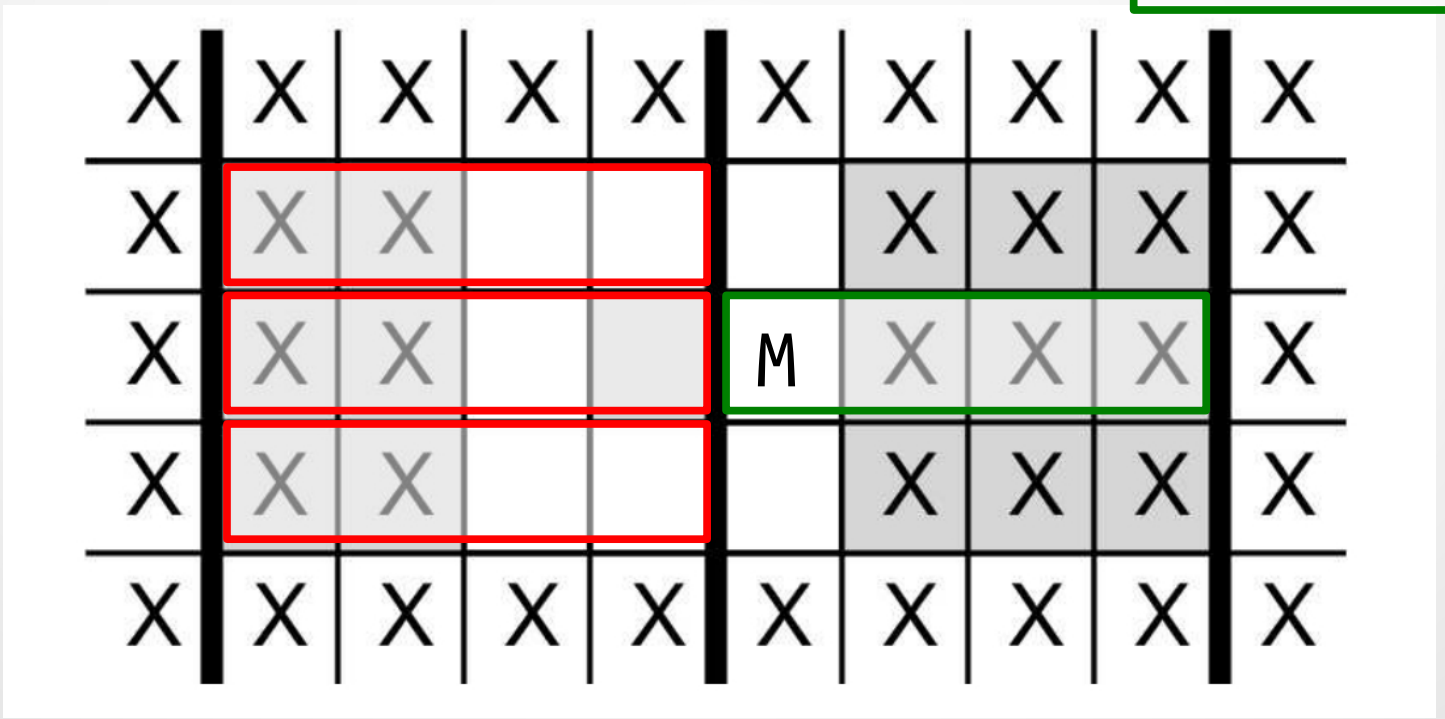
Set i's content

Set (i+1)'s content



Lets refresh your caches!

- Part 2: Iteration 2

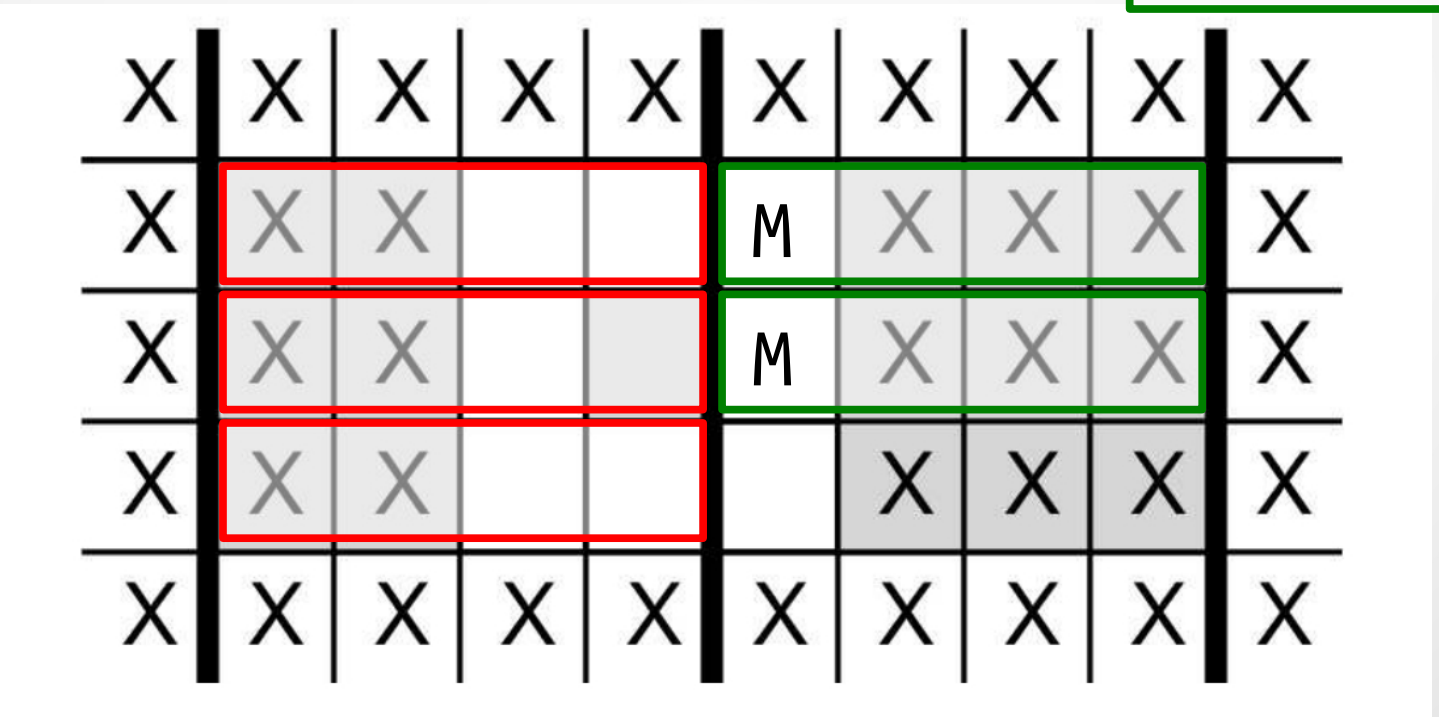


Lets refresh your caches!

- Part 2: Iteration 2

Set i's content

Set (i+1)'s content

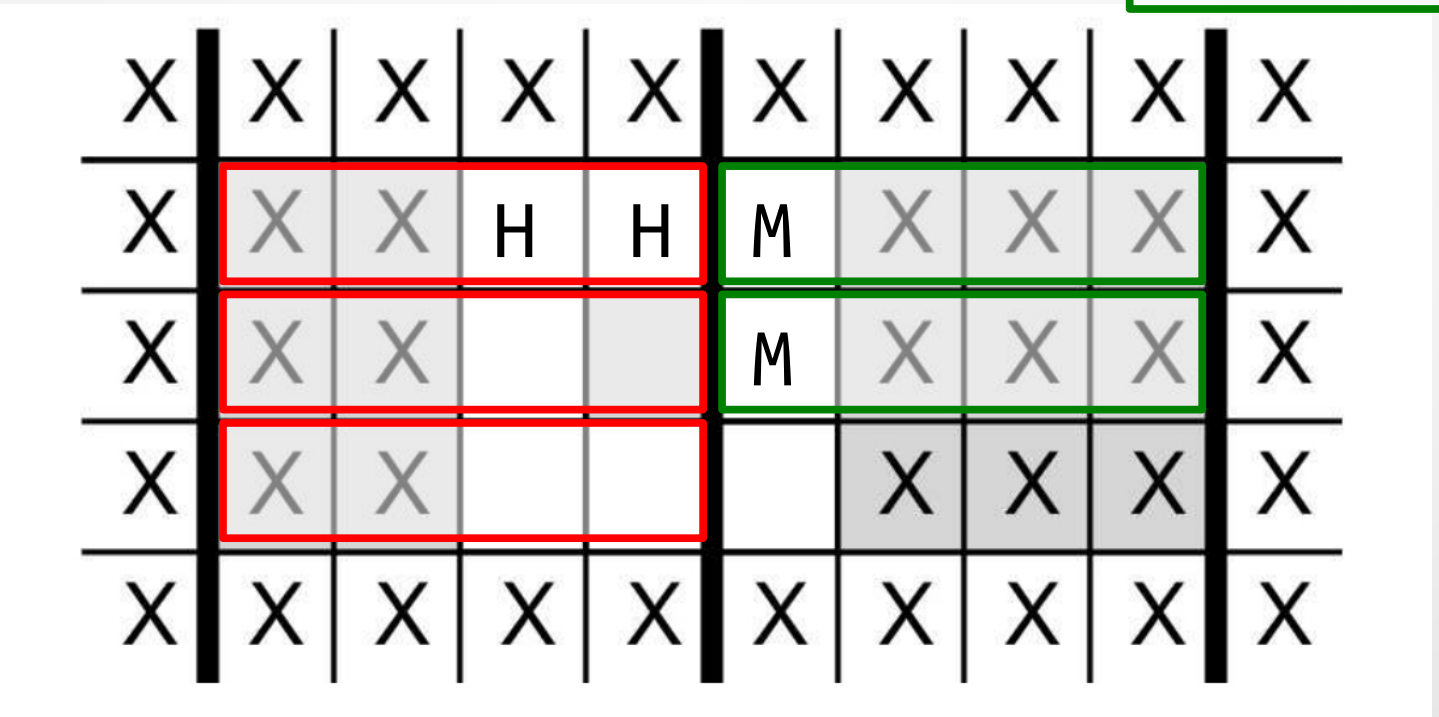


Lets refresh your caches!

- Part 2: Iteration 2

Set i's content

Set (i+1)'s content

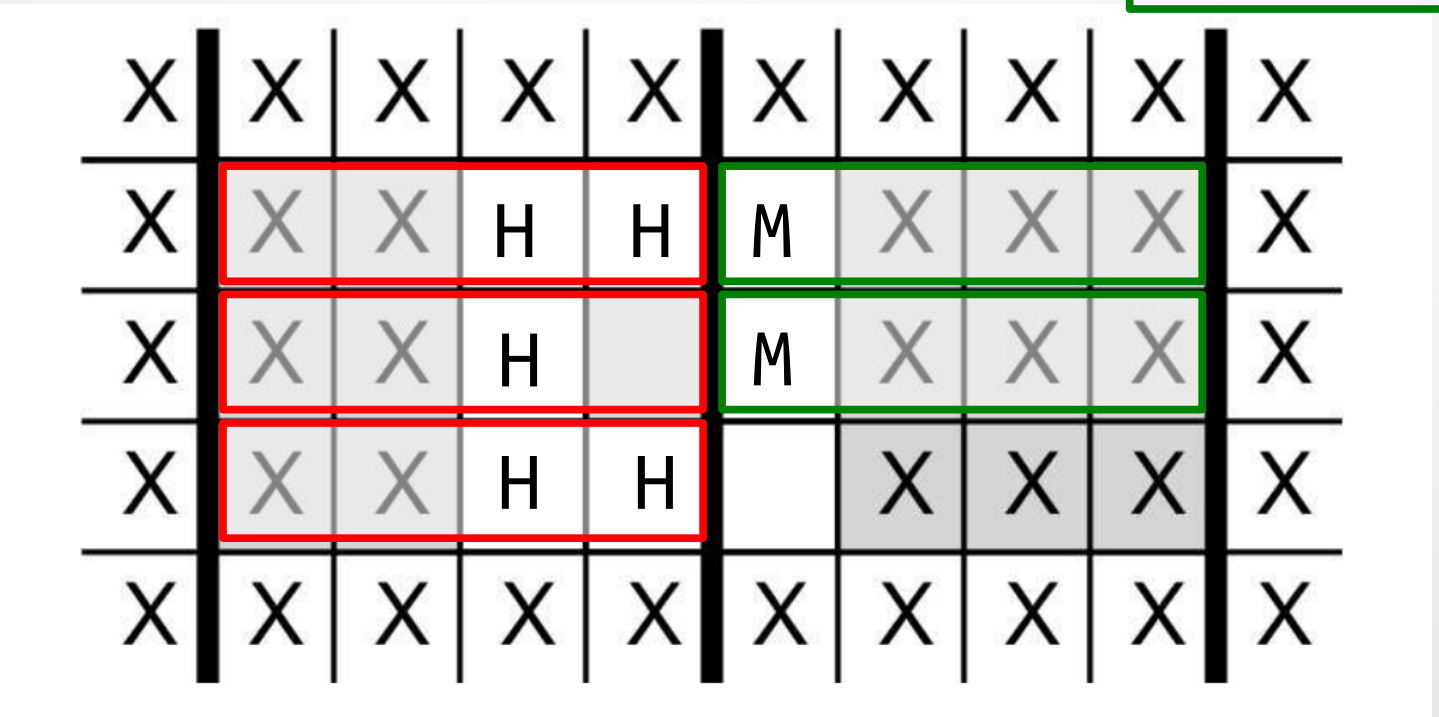


Lets refresh your caches!

- Part 2: Iteration 2

Set i's content

Set (i+1)'s content

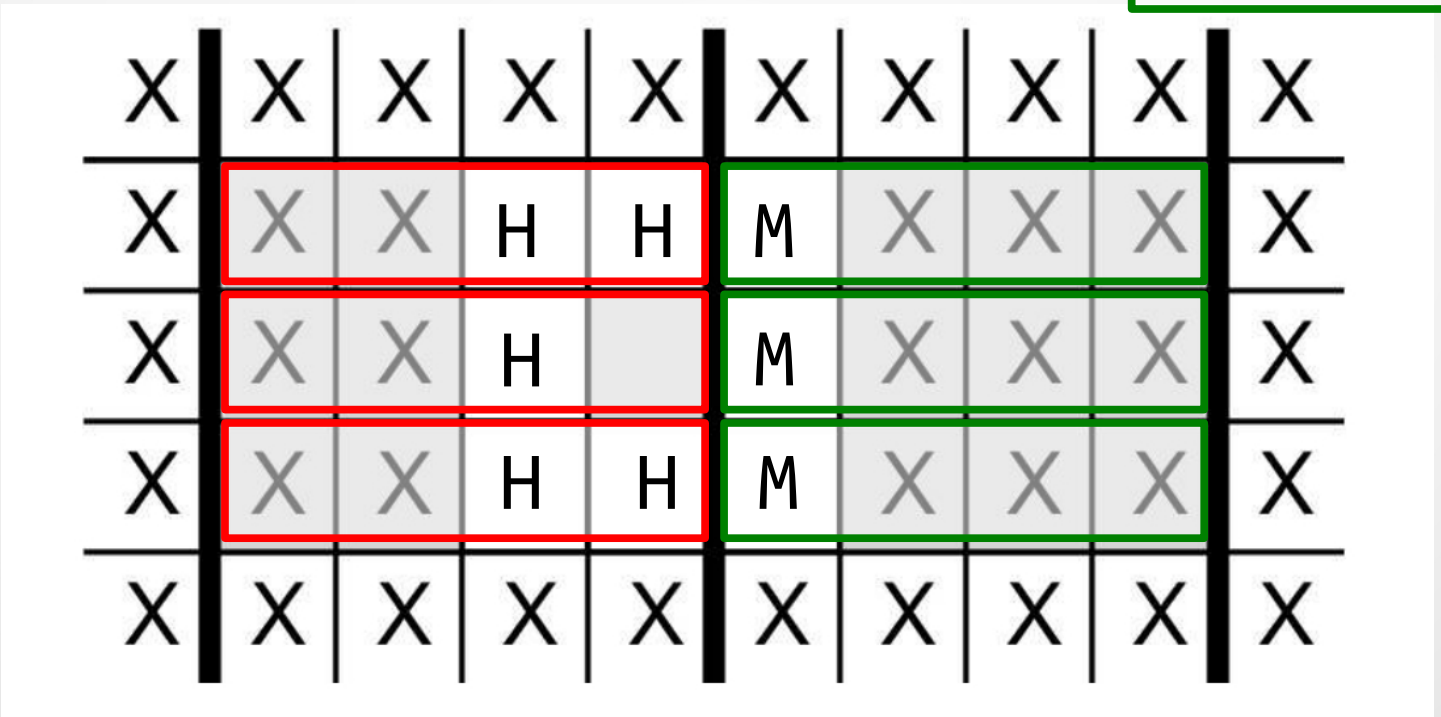


Lets refresh your caches!

- Part 2: Iteration 2

Set i's content

Set (i+1)'s content



Lets refresh your caches!

- Part 2: Iteration 2

Set i's content

Set (i+1)'s content