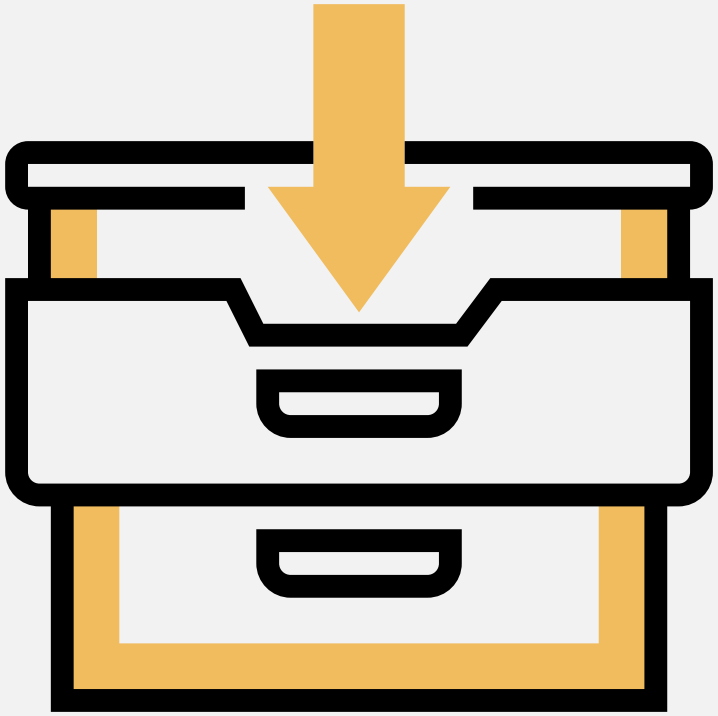
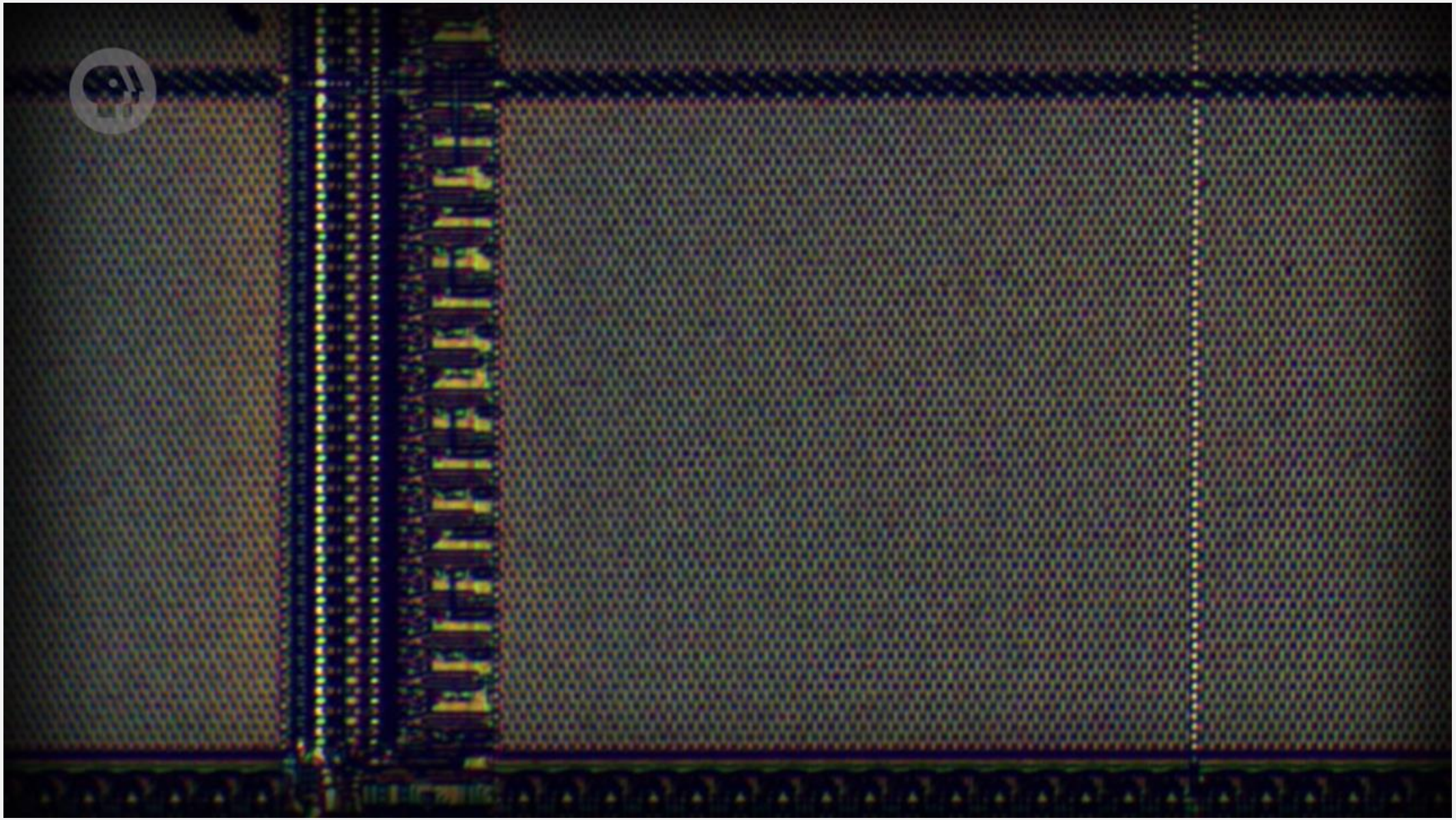
ARRAYS AND SEQUENTIAL SEARCH

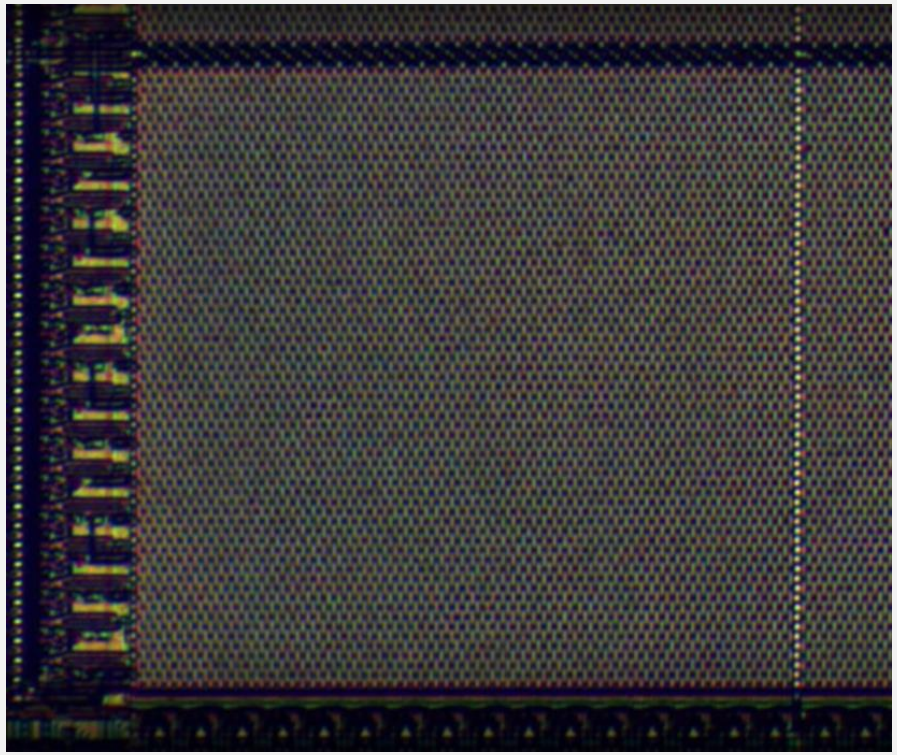
HOW MEMORY WORKS



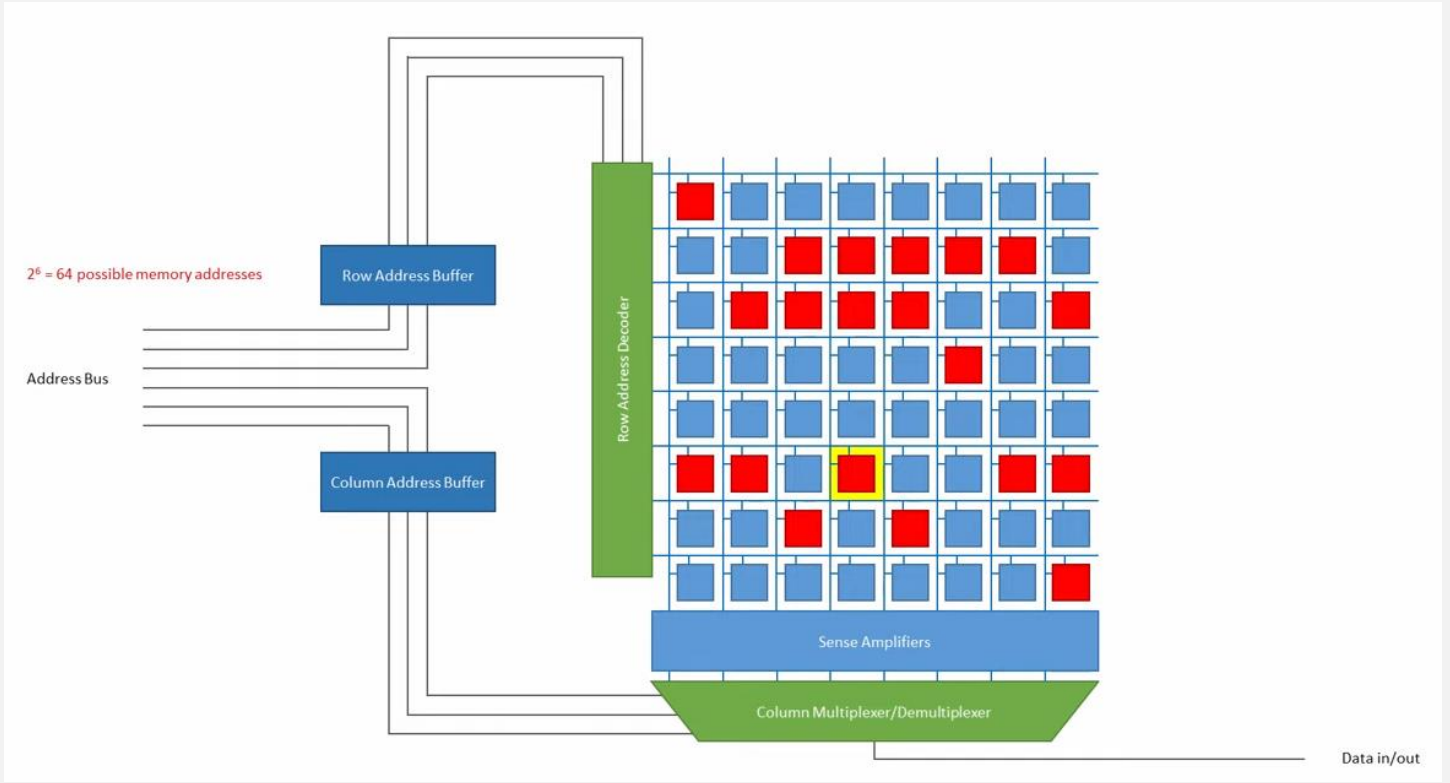


Bit Matrix: 32 x 32 bits = 1024 bits

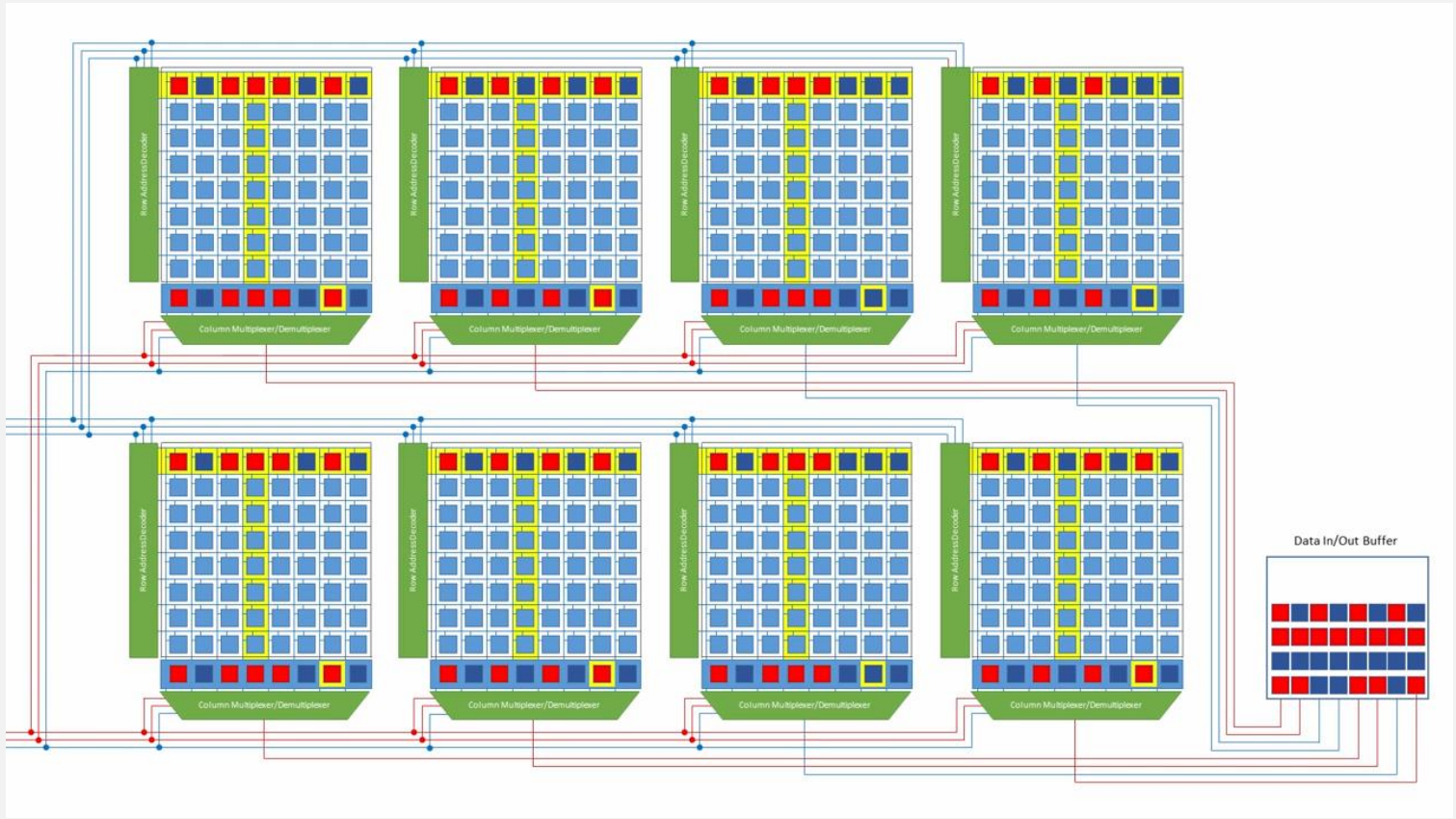
THE MAIN MEMORY



Simplification of the actual bit matrix (8x8):

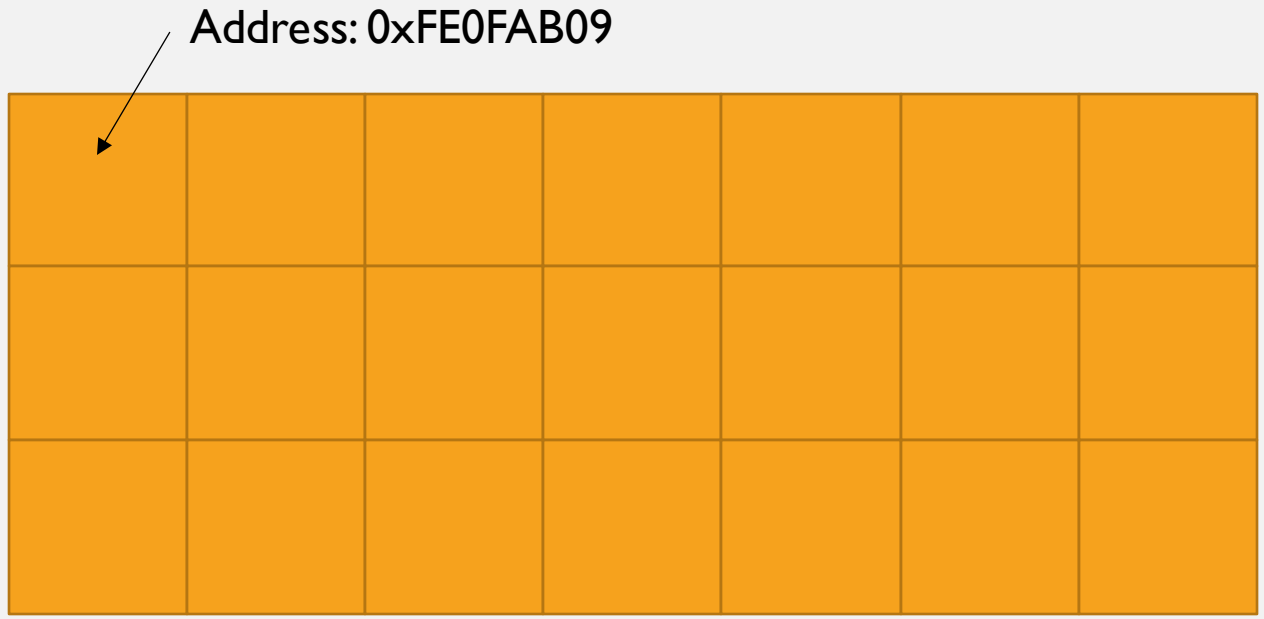


THE MAIN MEMORY: ADDRESS SPACE

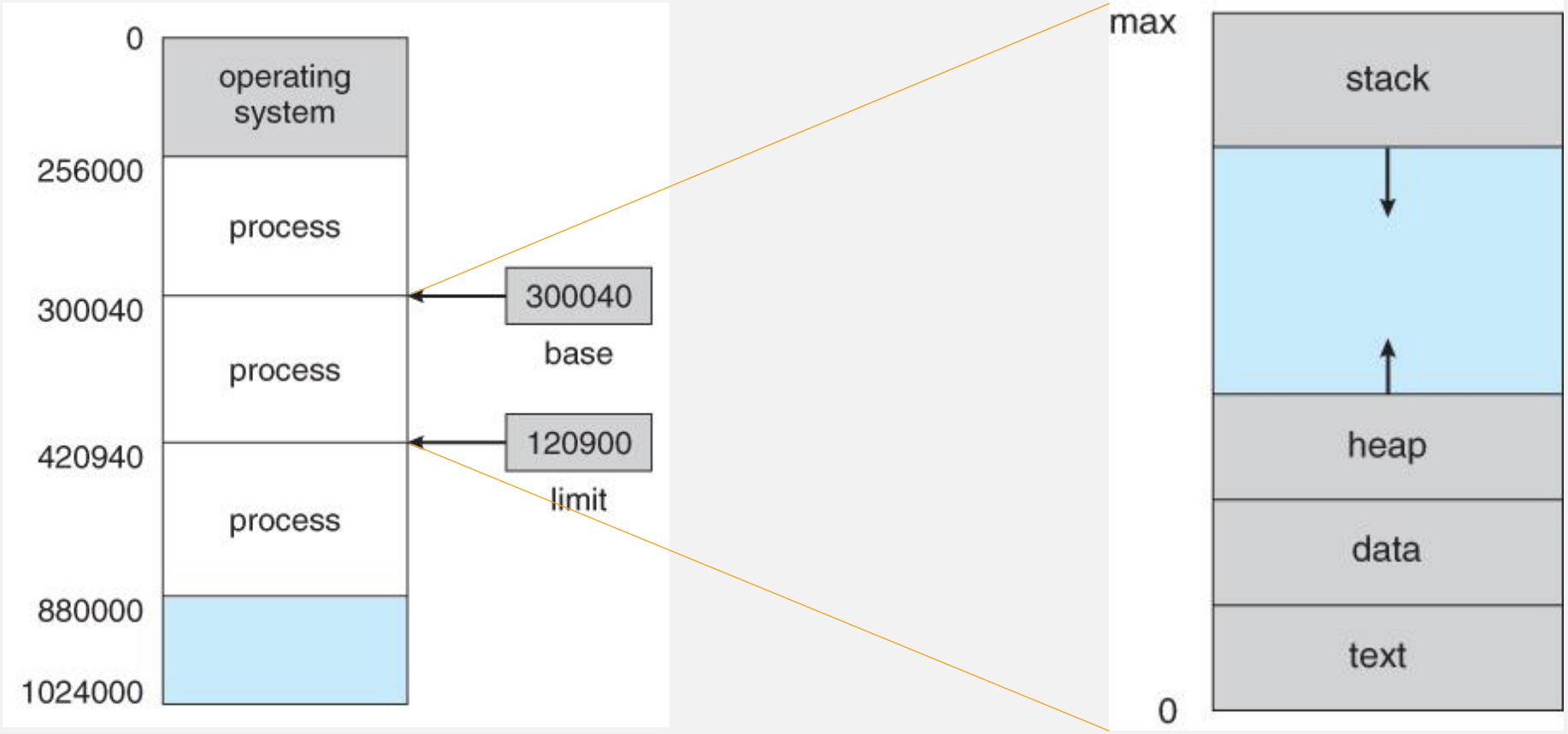


Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

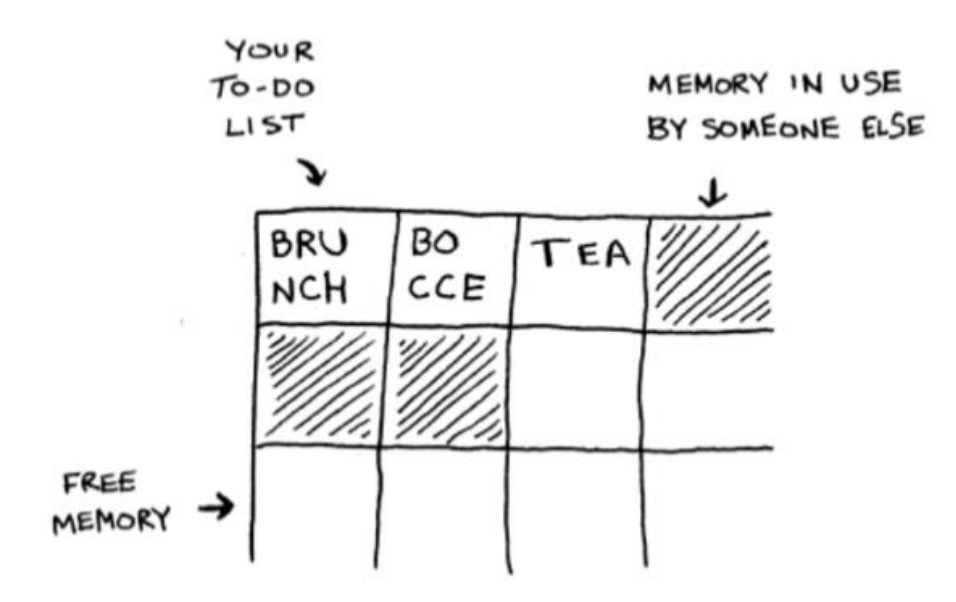
HOW MEMORY WORKS



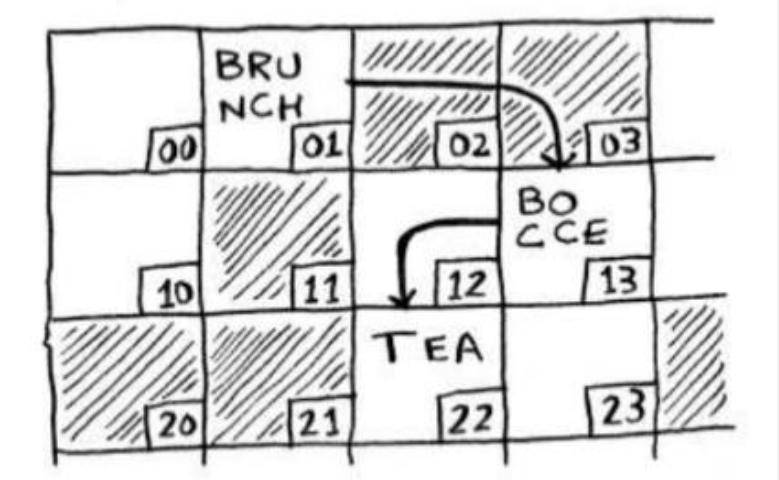
THE PROCESS MEMORY



COLLECTIONS: ARRAYS AND LINKED LISTS



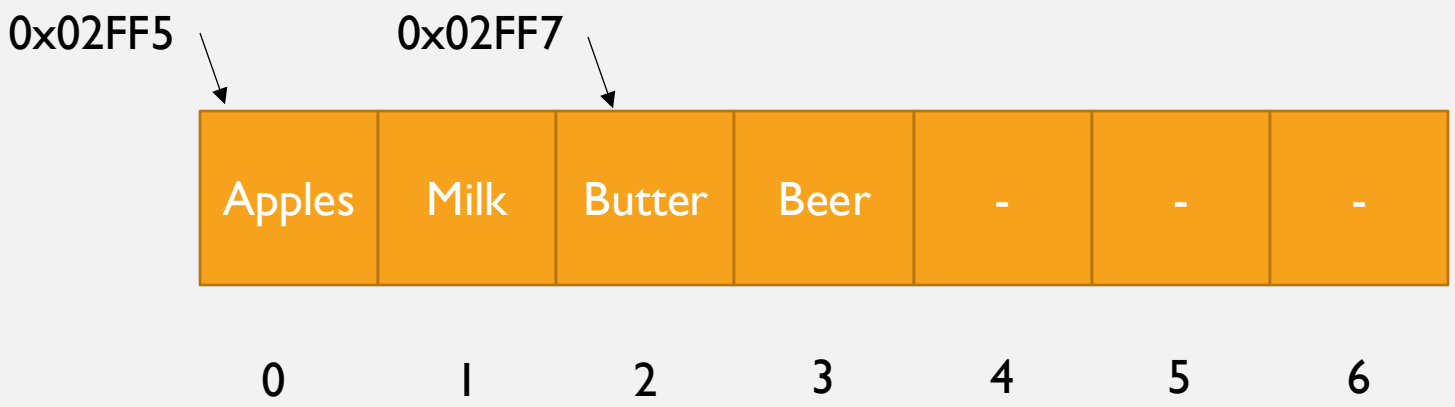
ARRAY



LINKED LIST

SEARCHING AND SORTING

ARRAYS



If the array is stored at Mem Address 0x02FF5, each element can be access by offsetting this address as many times as its index times the size of each cell.

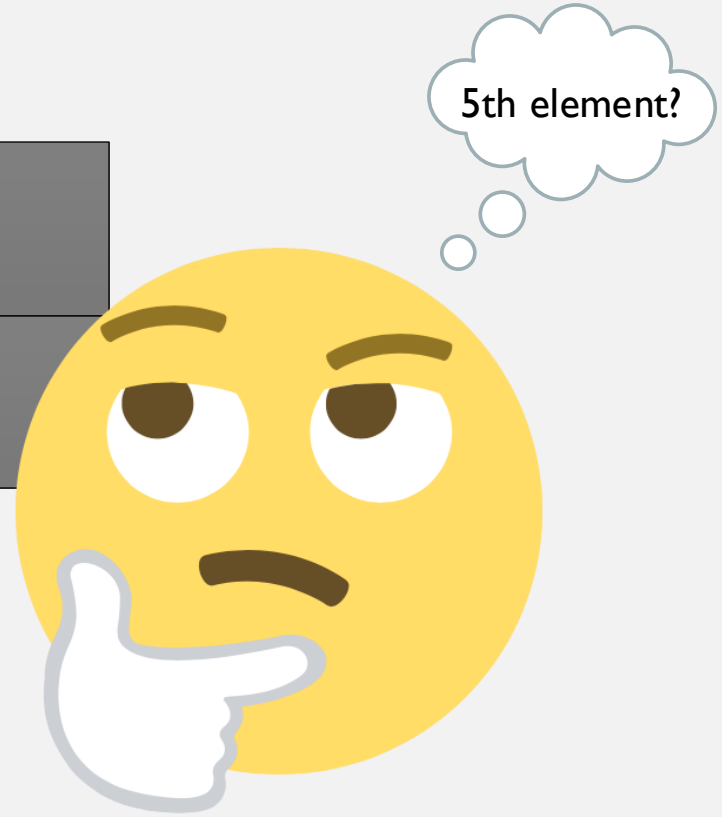

ARRAYS START AT 0
ARRAYS START AT 1
Perl
Fortran
ARRAYS START AT 0 AND 1
ARRAYS START AT 1.000000002

USAGE

```
>>> shopping_list = ['Apples', 'Milk', 'Butter', 'Beer']
>>> shopping_list
['Apples', 'Milk', 'Butter', 'Beer']
>>> shopping_list[0]
'Apples'
>>> shopping_list[0] = 'Oranges'
>>> shopping_list
['Oranges', 'Milk', 'Butter', 'Beer']
```

READ OPERATIONS

Address: 00



O(1) RUNTIME READS

LINEAR (O(N)) WRITES

```
>>> shopping_list[0] = 'Oranges'
```

```
>>> shopping_list.append('Onions')
```

SEQUENTIAL SEARCH

Do I need to buy apples?

Apples	Milk	Butter	Beer	-	-	-

Code it…. If you dare

```
def sequential_search(what, where):
    for i in range(len(where)):
        if where[i] == what:
            return True
    return False
```


Cool! But…
How efficient is this?

LINEAR (O(N)) RUNTIME COMPLEXITY

REVIEW

GIVE SOME EXAMPLES WHERE SEARCHING IN A DATA STRUCTURE IS REQUIRED

WHAT IS THE RUNTIME COMPLEXITY OF COPYING AN ARRAY?

- a) O(1)
- b) O(N)
- c) Other

WHAT IS THE RUNTIME COMPLEXITY OF INSERTING AN ELEMENT IN THE MIDDLE OF AN ARRAY?

- a) O(1)
- b) O(N)
- c) Other

IS THERE A WAY TO IMPROVE OUR SEQUENTIAL SEARCH IMPLEMENTATION TO MAKE IT FASTER?

EXERCISES

- Code a function ***is_palindrome*** that receives an array as input and returns *True* if its content forms a palindrome.
- Code a function ***max_repeated*** that receives an array and returns the largest occurrence of a sequence of repeated elements. For instance, given A = [1, 2, 2, 2, 3, 3, 4, 2, 1], it should return the array [2, 2, 2].
  - What if we modify it to find the element with the most occurrences?
- Extra: create a function ***find_largest_palindrome*** that receives an array as input and returns the largest palindrome found in the array.