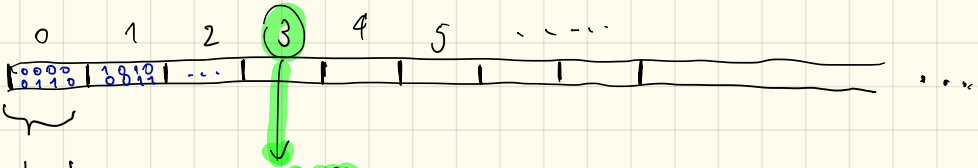
STORING ARRAYS IN MEMORY

The computer needs to write somewhere all the info that needs to be processed/stored

↳ RAM → linear container



1 byte
||
8 bits

how many? ⇒ billions ! GB = 10^9 bytes

convention, all the computers we nowadays organized this way

All the bits always have a state 1/0, depending on past computations, random initialization, ...

What is the most natural way of storing a sequence of numbers in this memory?

E.G.: we want to store [2, 14, 3, 8]

Binary representation

2 → 00000010
14 → 00001110
3 → 00000011
8 → 00001000

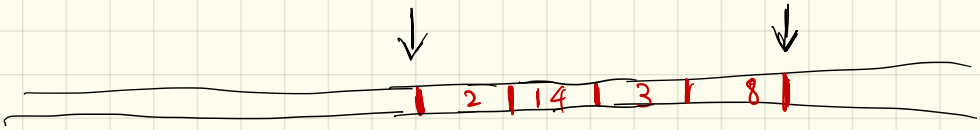
} they all fit in 1 byte!

For example take 14 →

8 bits

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	0	0	1	1	1	0

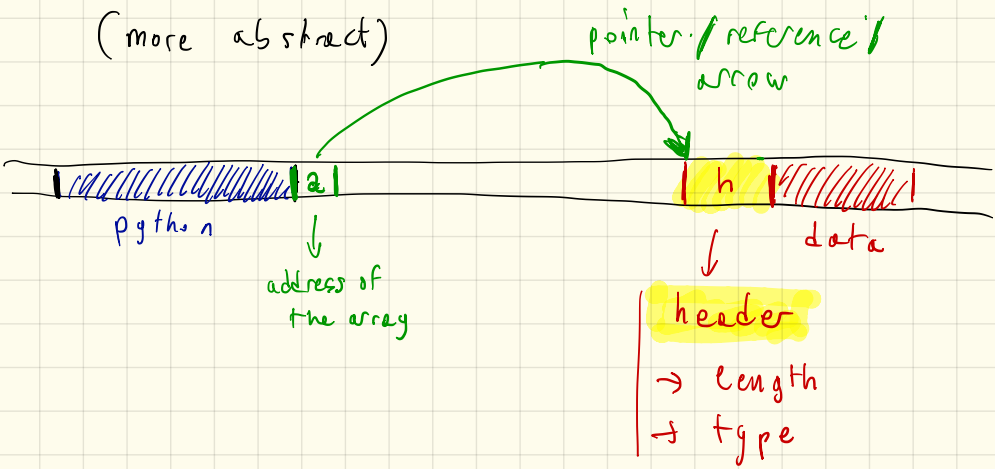
⇒ Use consecutive slots in the RAM:



start? end?

there is nothing special about these bits, unless I have a flag or something that points there!

Storing 1-d ndarrays (#1)



WHY DO I NEED THE TYPE?

→ uint 8 ⇒ each element takes 1 byte

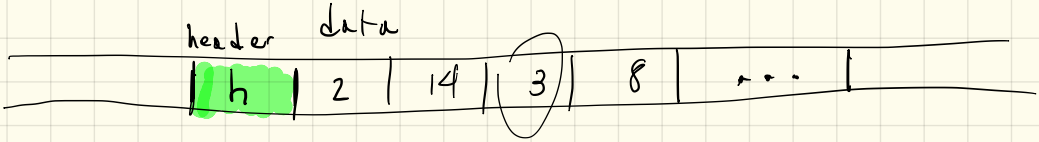
→ uint 32 ⇒ each element takes 4 bytes

⇒ homogeneity of the entries +
length +
type =

= now the CPU knows how to read the content!

Storing python list (#1)

Could I use the same compact representation for lists?



Flexible ASSIGNMENT → swap 3 for 3^100

↳ I need more bits to store it → I have to move all the rest of the entries

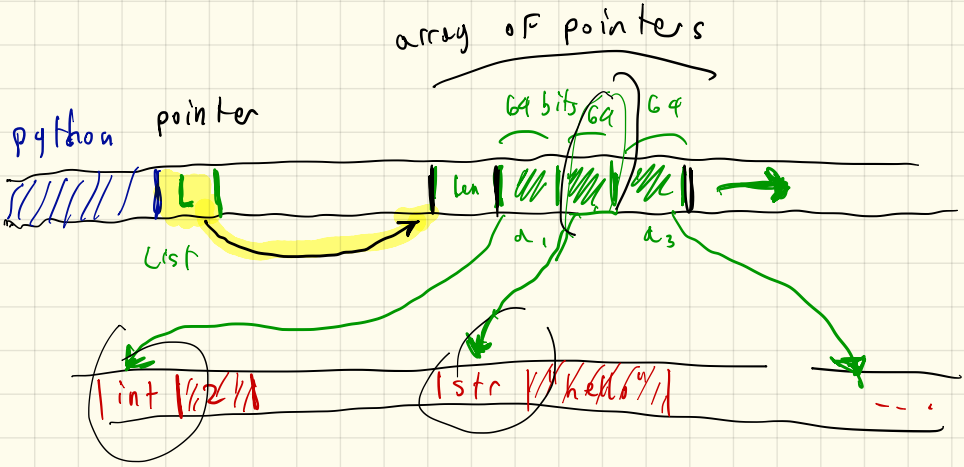
the operation would be O(N), where N is the length of the vector

→ we need O(1)
⇒ constant cost!

Flexible TYPE → what if I swap 3 for "a", how does the program know how to interpret the bits?

IT DOES NOT WORK

Storing python list (#2)



NOW I NEED 2 JUMPS TO FETCH THE CONTENT OF THE LIST L, BUT

→ assignment becomes O(1)

↳ write new content somewhere else

↳ update the pointer

Why is it efficient to do operations within some type?

int64

1 + 2.5 = 3.5

same operation? NO

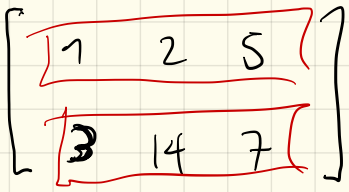
same result

1.0 + 2.5 = 3.5

float64

→ First row requires a conversion (int → float)

Storing 2-d ndarrays (#1)

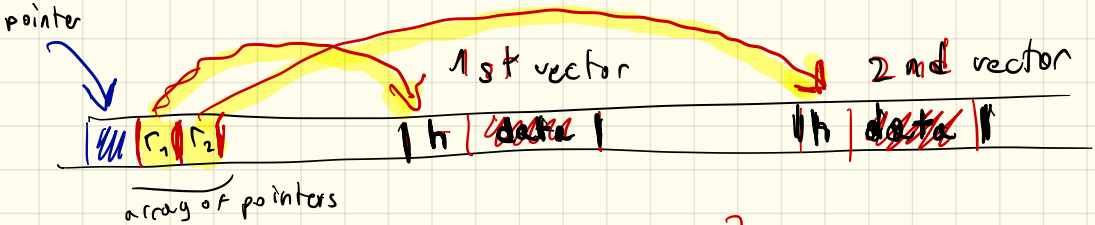


→ 2D object: how to store it?

(RAM is 1D)

(NOT HOW NUMPY DOES IT!)

1st possible way → store each row as a separate vector, and then link them



[[1 2 5],[3 14 7]]

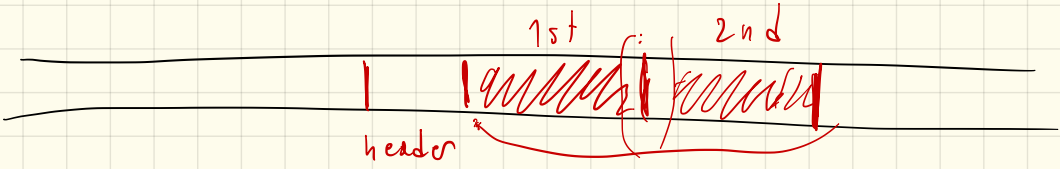
this is indeed similar to how python would store a list of lists

↓

add another layer of pointers

Storing 2-d ndarrays (#1)

Numpy groups all the data in contiguous slots ⇒ SINGLE HEADER



I NEED MORE INFO NOW

- type
- size = tot number of entries
- dim = 2
- shape = (n rows, n cols) = (2, 3)

In general d-dimensional tuple

In what order am I storing the table?

ROW-MAJOR ORDER → C, C++, python, ...

transpose

COLUMN-MAJOR ORDER → Fortran, Matlab, Julia, ...

Row-major ⇒ go row by row!