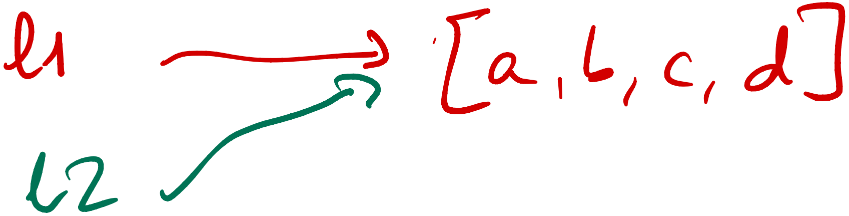
Mutable

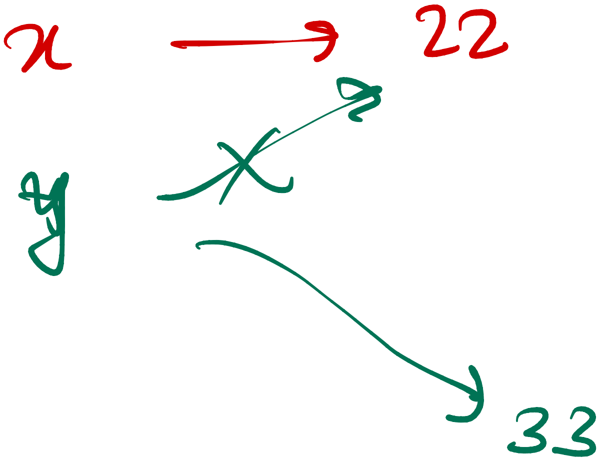
l2=l1



Immutable

y=x

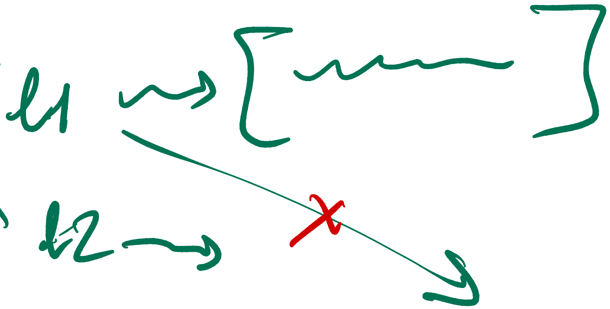
y=33



```
def f(a,b):
    a=m
    b=n
```

f(m,n)

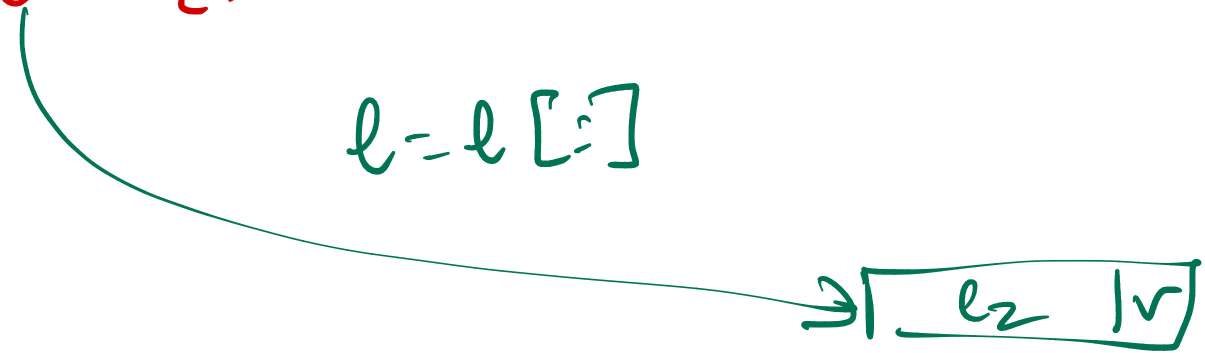
```
def mycopy(m,n)
    m=n
    l2=l1
```



```
def myappend2(l, v)
    l = l + [v]
```

$l_2 \rightsquigarrow$

```
l = l[:]
```



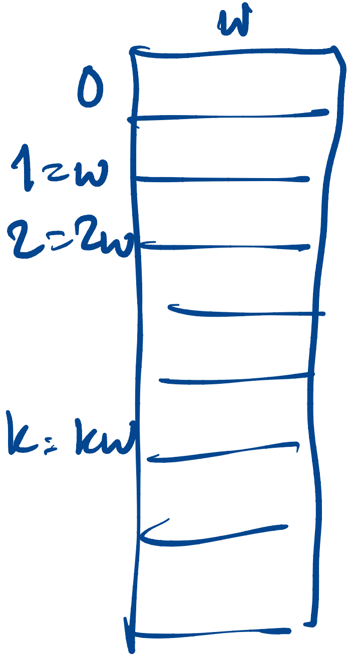
Like m=n in first ex.

Array

Contiguous block of storage

Fixed size

Elements are uniform



$a[k]$ is at $a[0] + k \cdot w$

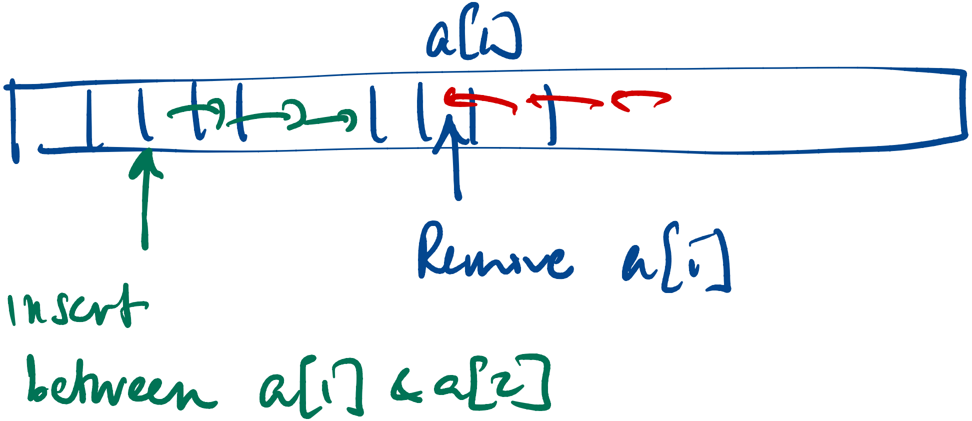
"Random access"

No difference in time to access any $a[j]$

Arrays

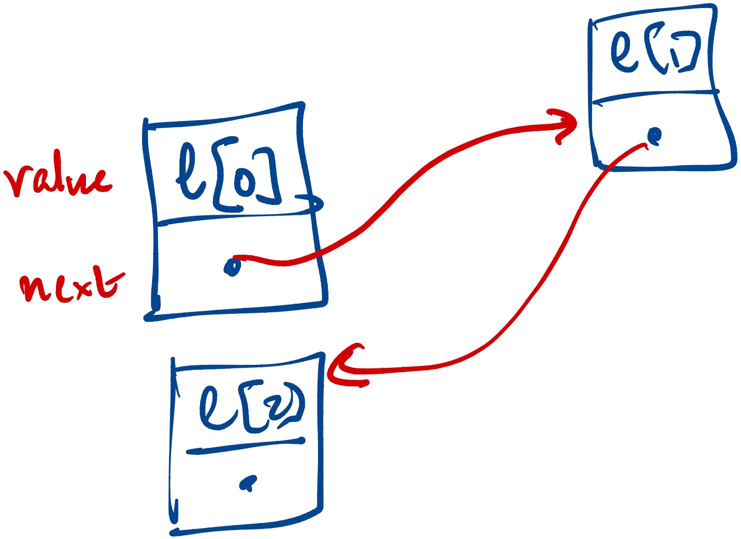
Inflexible in size

Insert/delete are expensive



List – flexible

Made up of "linked" units

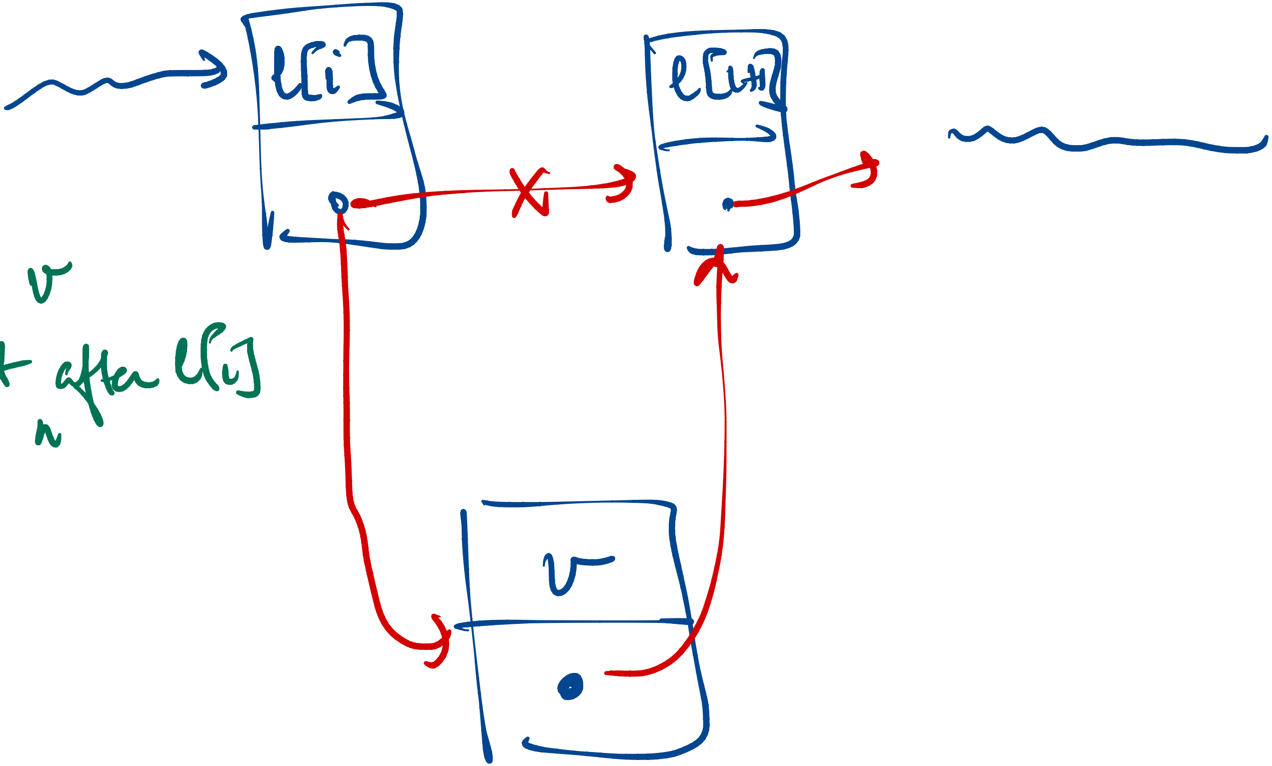


To access l[i]

– start at l[0]

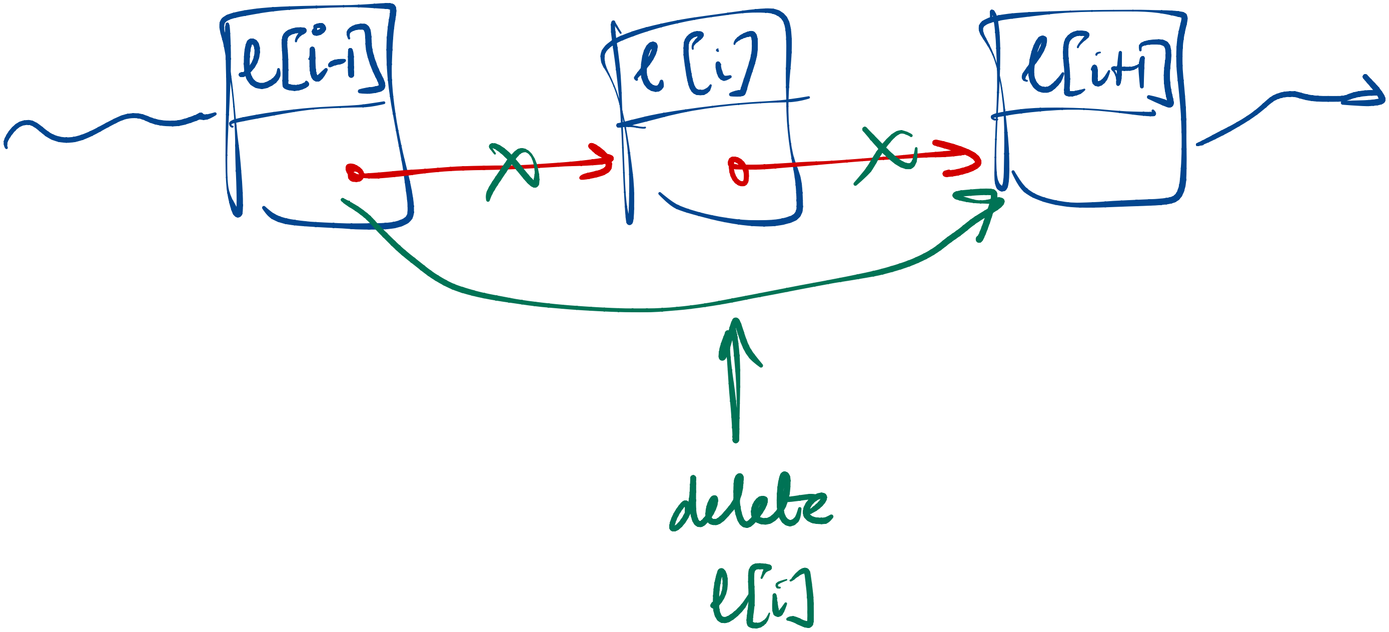
& "walk" i steps

Insert



Insert v after l[i]

Delete

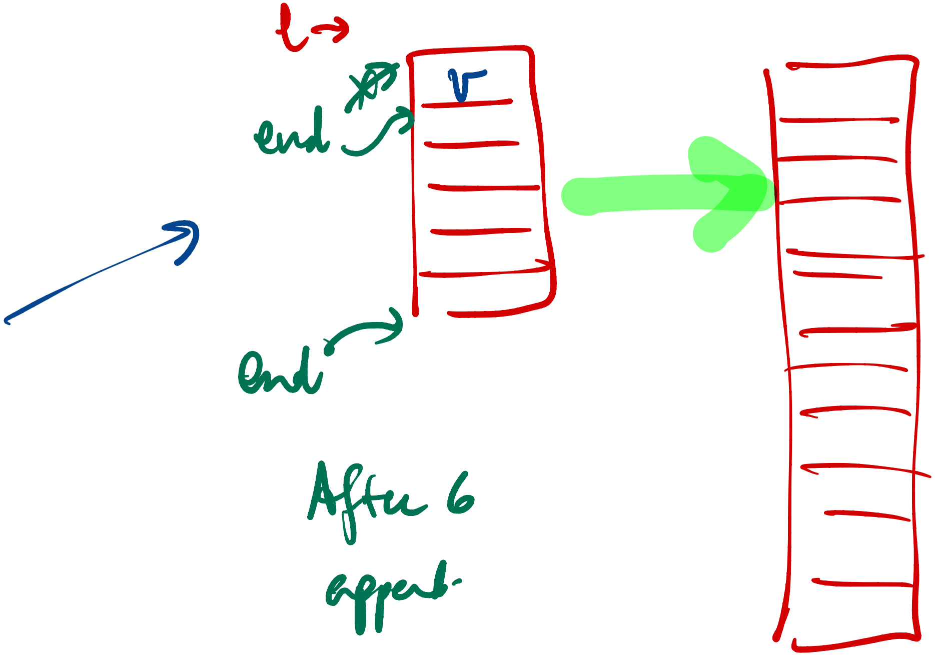


What is a Python list?

Growing array

$l = [\,]$

$l.\text{append}(v)$



After 6 append.

fresh array, double size

l.append(v) – cheap, modulo doubling

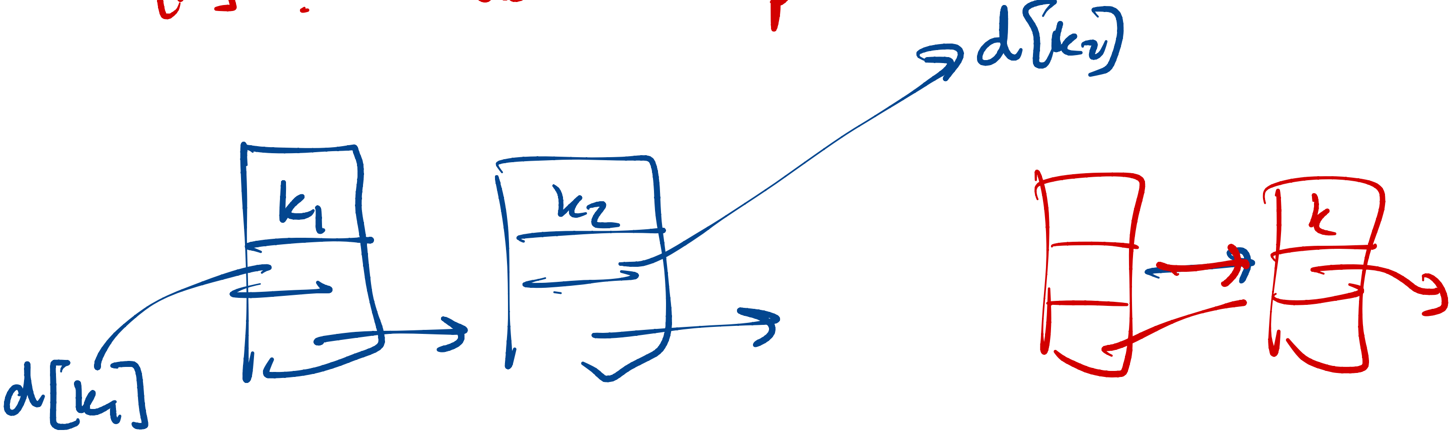
l.insert(v, i) – should be expensive in Python

Dictionary ?

key → value

Map keys to memory location

d[k] ? Where does k point to

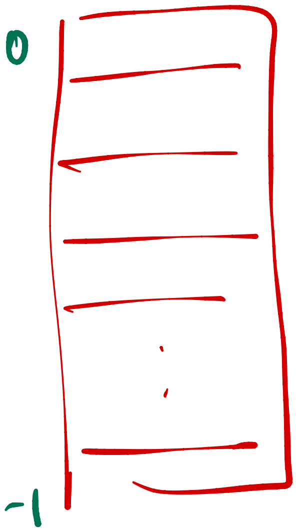


Instead

$d = \{\}$ ⟶ allocate an array

Function that maps

keys to $\{0, 1, \ldots, m-1\}$



k → binary string → binary number

↓

mod m

"hashing"

Collision

$$h(k_1) = h(k_2)$$

Minimize collision

- Design good hash function