CS 261: Data Structures

Dynamic Arrays

Introduction

Arrays -- Pros and Cons

```
# define MAX_SIZE 100
struct Bag {
    TYPE data[MAX_SIZE];
    int size;
};
```

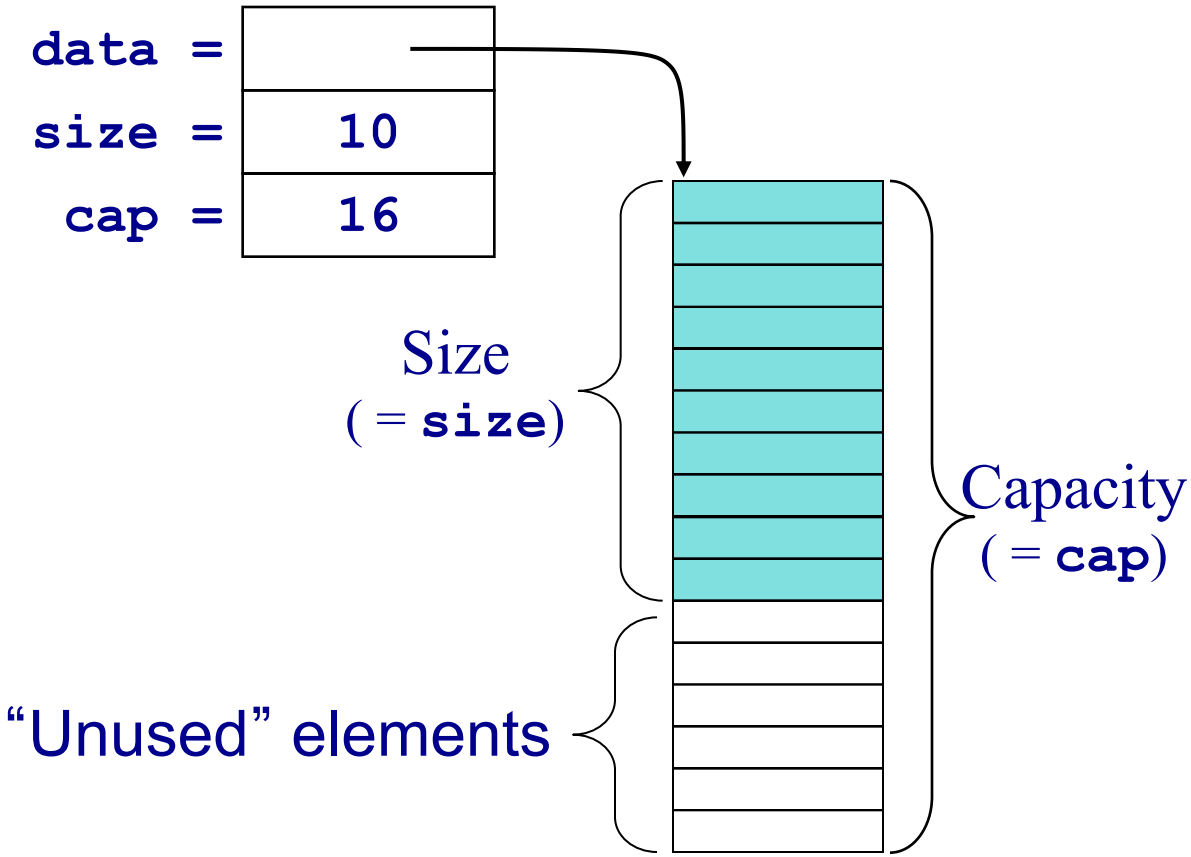
- Positives:
  - Simple
  - Each element accessible in O(1)

- Negatives:
  - Size must be fixed when created
  - What happens when the program later requires more space?

Dynamic Arrays

- Our goal: Hide memory management details behind an Application Program Interface (API)
- Each element is still accessible in O(1)
- But a dynamic array can change **capacity**

Dynamic Array



Size and Capacity

- Size:
  - Current number of elements
  - Managed by an internal data value
- Capacity:
  - Number of elements that a dynamic array can hold before it must resize

Adding an Element

- Increment the size
- Put the new value at the end of the dynamic array

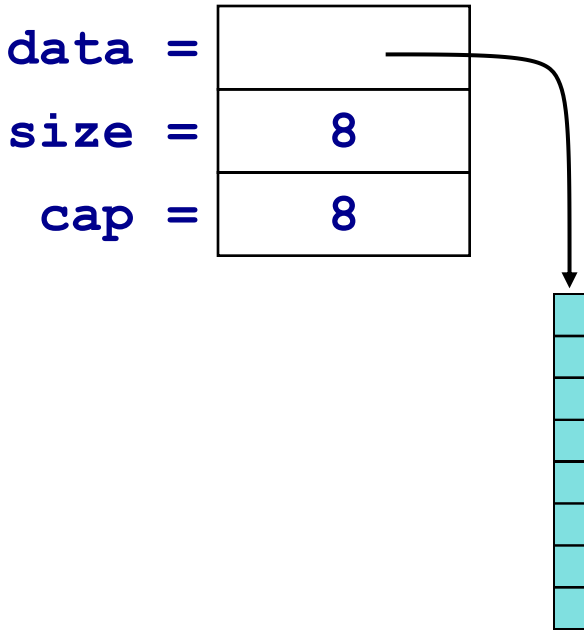
Adding an Element

- What happens when size == capacity?

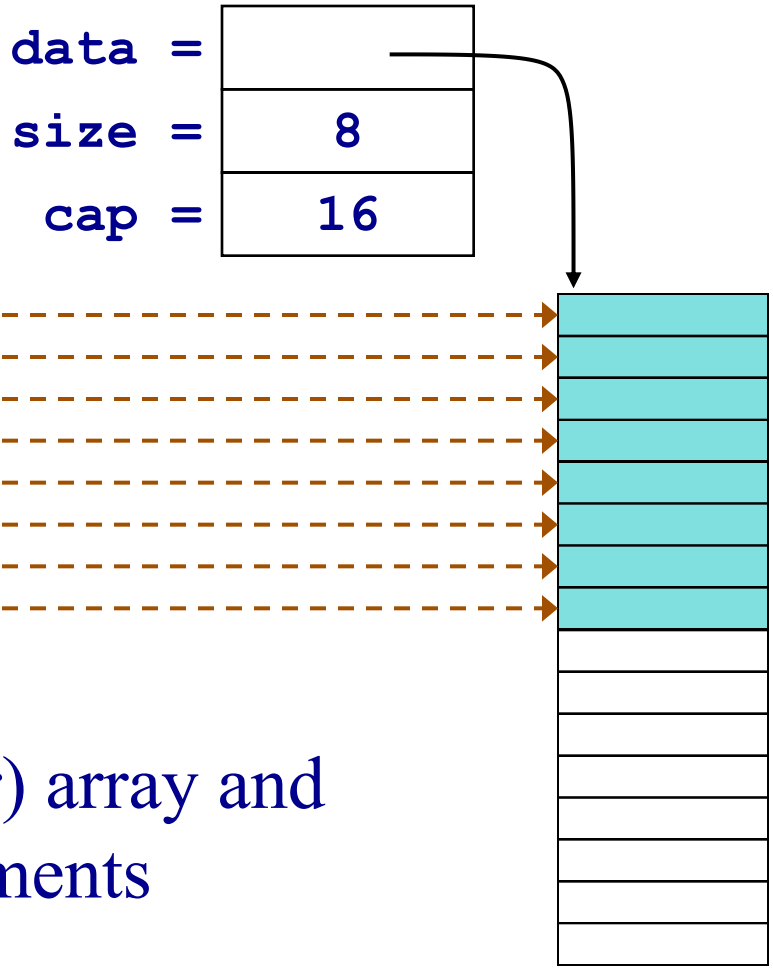
- Must:
  - reallocate new space
  - copy all data values to the new space
  - hide these details from the user

Reallocate and Copy

Before reallocation:



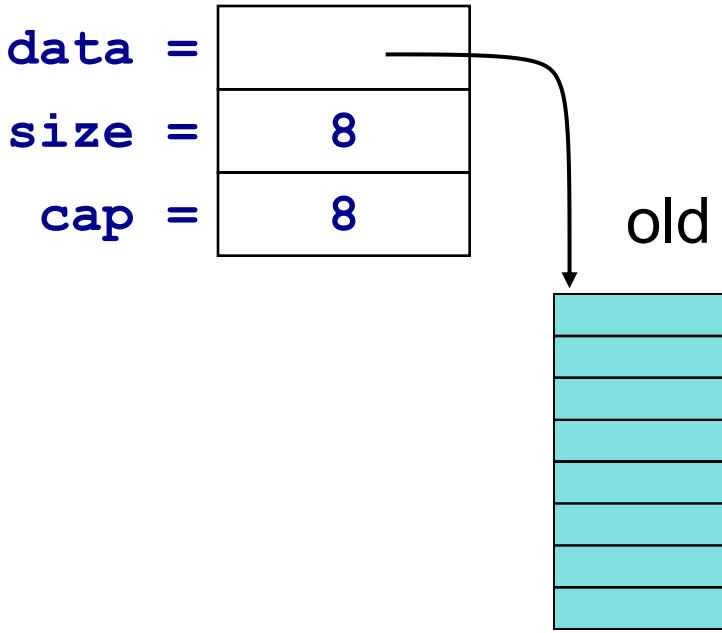
After reallocation:



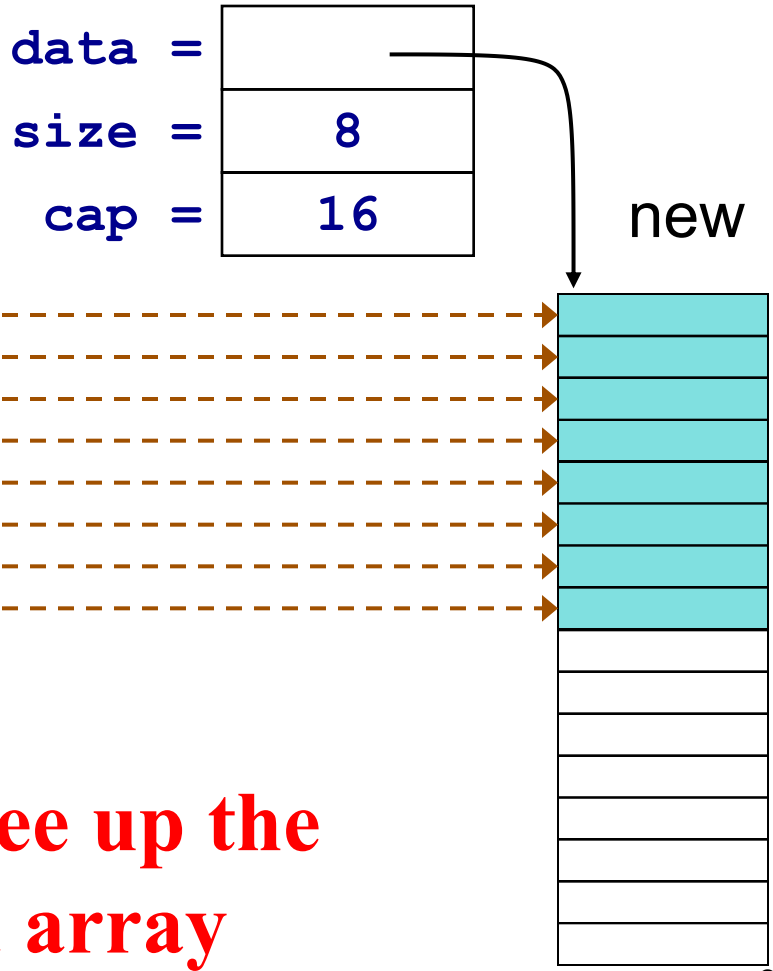
Must allocate new (larger) array and copy valid data elements

Reallocate and Copy

Before reallocation:



After reallocation:



DO NOT forget to free up the memory of the old array

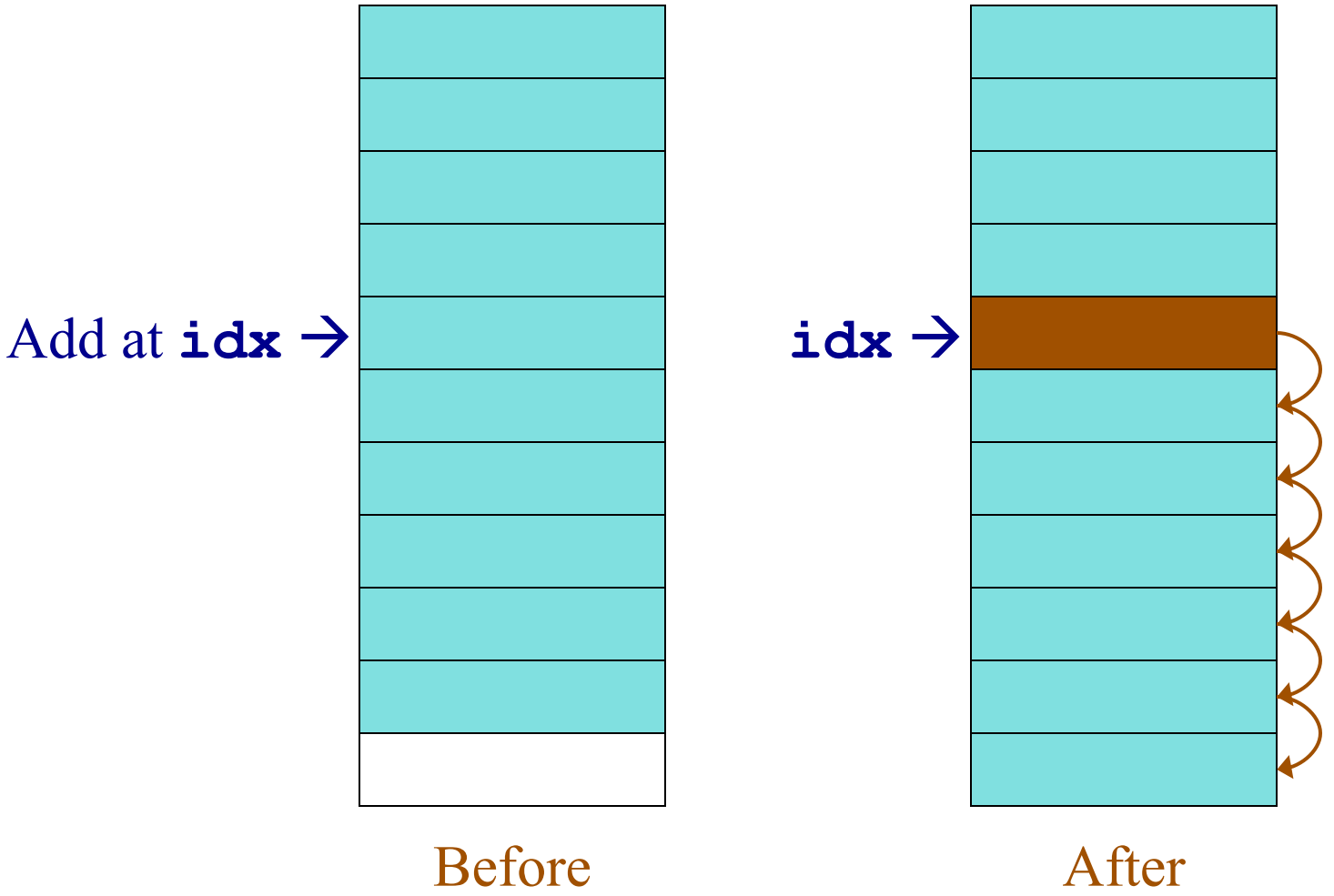
Inserting an Element in the Middle

- May also require reallocation
  - When?

- Requires that some elements be moved up to make space for the new one

Inserting an Element

Loop from **THE END backward** while copying

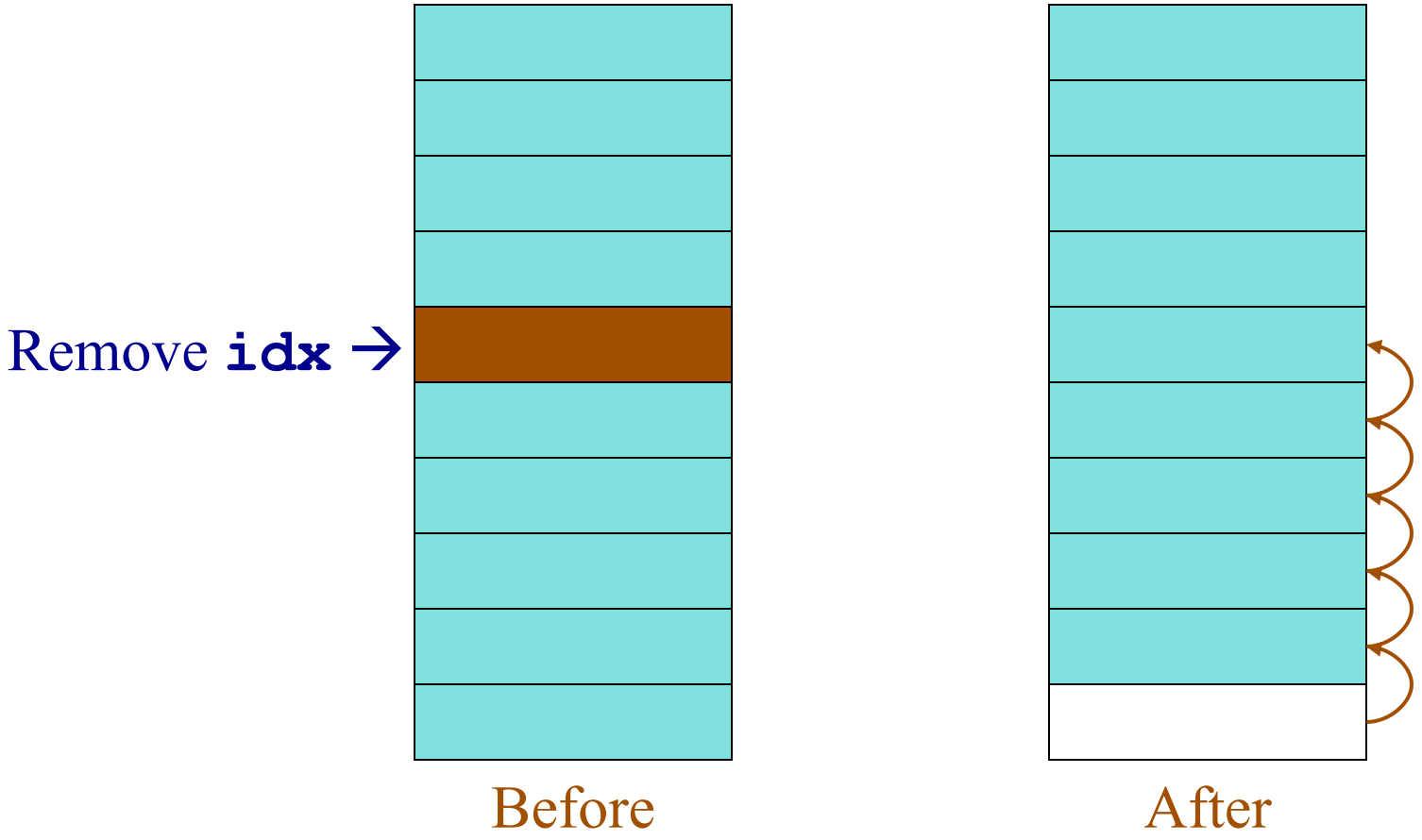


Inserting an Element -- Complexity

O(*n*) in the worst case

Removing an Element

- Remove also requires looping.
- Loop from **`idx`** **forward** while copying



Removing an Element -- Complexity

O(n) worst case

Interface View of Dynamic Arrays

Interface file: **dynArr.h**

```
struct dyArr {
    TYPE * data;   /* Pointer to data array */
    int size;      /* Number of elements */
    int capacity; /* Capacity of array */
};

/* Rest of dynarr.h on next slide */
```

Interface (continued)

```
/* function prototypes */

void initDynArr (struct dyArr *da, int cap);

void freeDynArr (struct dyArr *da);

void addDynArr (struct dyArr *da, TYPE val);

TYPE getDynArr (struct dyArr *da, int idx);

void putDynArr (struct dyArr *da, int idx, TYPE val);

int sizeDynArr (struct dyArr *da);

void _dyArrDoubleCapacity (struct dyArray * da);
```

Implementation View of Dynamic Arrays

initDynArr -- Initialization

```
/* Allocate memory to data array */

void initDynArr (struct dyArr *da, int cap){
      assert (cap >= 0);
      da->capacity = cap;
      da->size = 0;
      da->data = (TYPE *)
                  malloc(da->capacity * sizeof(TYPE));
      assert (da->data != 0); /* check the status */
}
```

freeDynArr -- Clean-up

```
void freeDynArr (struct dyArr * da)
{
        assert (da != 0);
        free (da->data); /*free entire array*/
        da->capacity = 0;
        da->size = 0;
}
```

Size

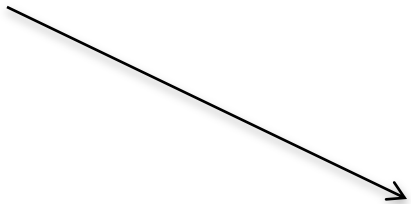
```
int sizeDynArr (struct dyArr * da){
    return da->size;
}
```

Get the Value at a Given Position

```
TYPE getDynArr (struct dyArr *da, int idx);
{     /*always make sure the input is meaningful*/
     assert((sizeDynArr(da) > idx) && (idx >= 0));
     return da->data[idx];
}
```

why?

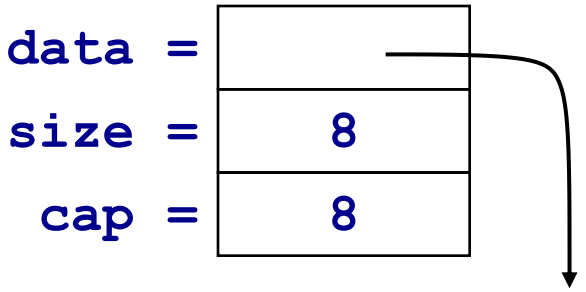
Add a New Element



```
void addDynArr (struct dyArr * da, TYPE val){

        /*make sure there is enough capacity*/

        if (da->size >= da->capacity)

                    _dyArrDoubleCapacity(da);

        da->data[da->size] = val;

        da->size++; /*must increase the size*/

}
```

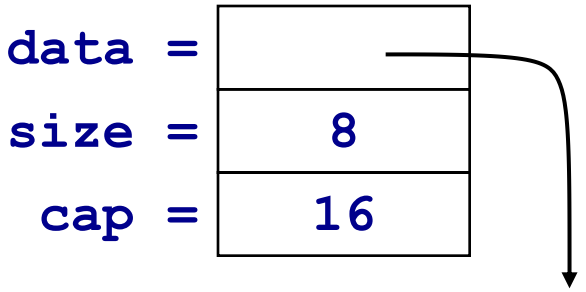
Double the Capacity

Before reallocation:

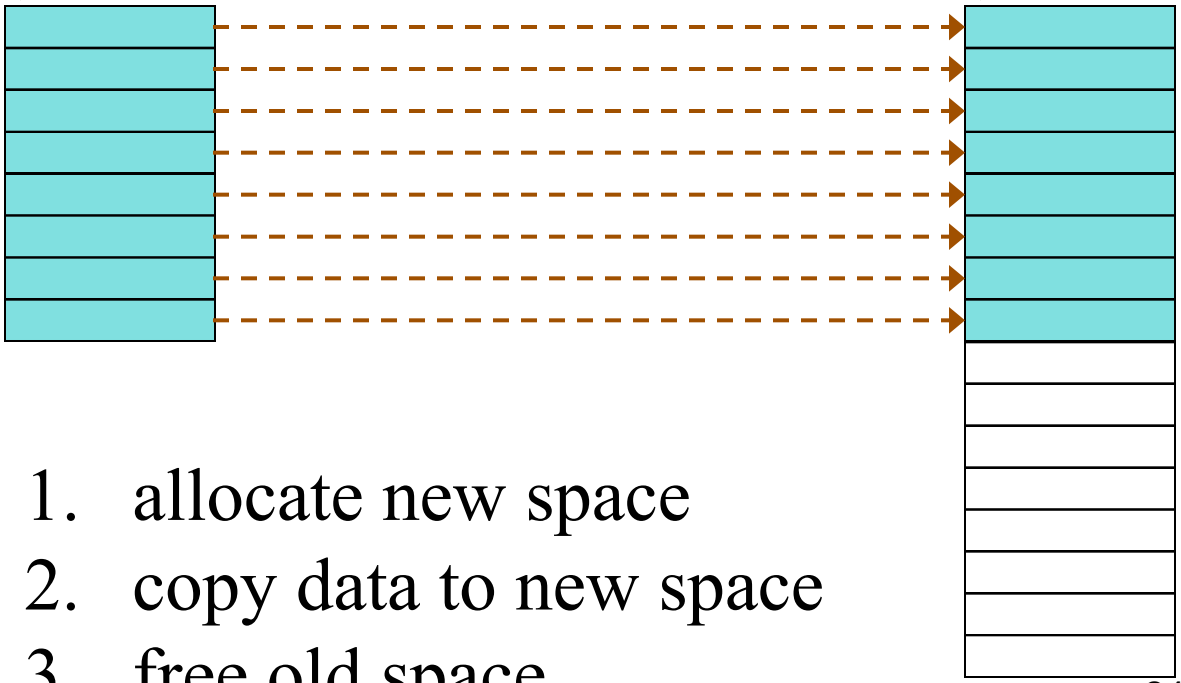


old

After reallocation:



new



MUST:

1. allocate new space
2. copy data to new space
3. free old space

Double the Capacity

```
void _dyArrDoubleCapacity (struct dyArray * da) {
        TYPE * oldbuffer = da->data; /*memorize old*/
        int oldsize = da->size;
        /*allocate new memory*/
        initDynArr (da, 2 * da->capacity);
        for (int i = 0; i < oldsize; i++) /*copy old*/
                da->data[i] = oldbuffer[i];
        da->size = oldsize;
        free(oldbuffer); /*free old memory*/
}
```

Next Class

How to implement

- Stack
- Bag

by using Dynamic Array