Linked Lists vs. Arrays

Arrays:

- fixed size
- O(1) random access

Linked lists:

- O(1) addition
- O(*n*) random access

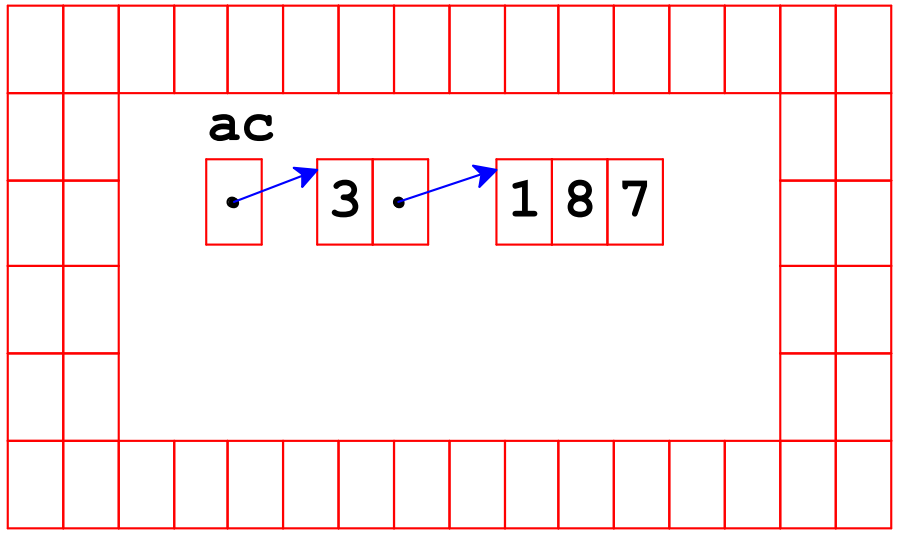
Array Containers

An array has a fixed size, but an ***array container*** can grow by swapping in a larger array

```
struct container {
  int count;
  int *a;
};
```

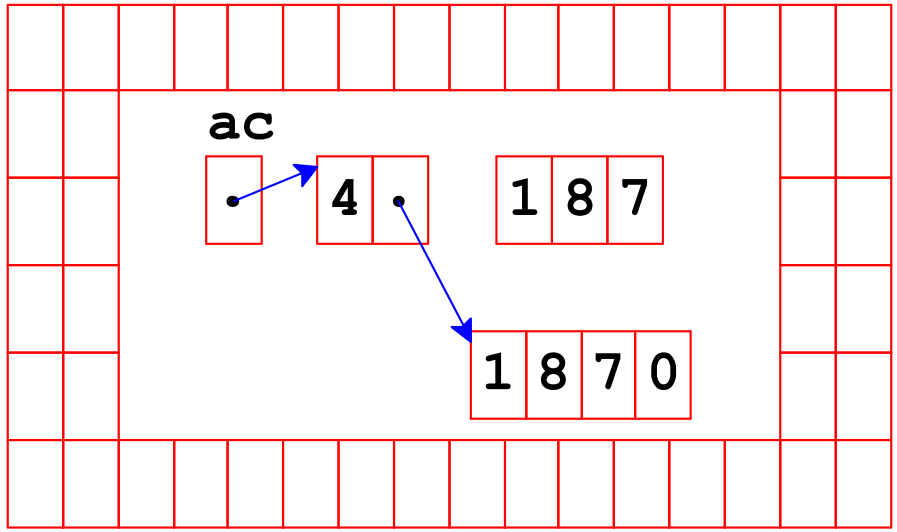
Array Container

Before:




```
add_to_back(ac, 0);
```

After:



Array Container Performance

Adding to the back:

- Copy existing n items: $O(n)$
- n items: $O(n^2)$

Adding to the front:

- Copy existing n items: $O(n)$
- n items: $O(n^2)$

Anticipating Further Additions

We could lower the cost of adding an item if we keep some extra space in the array

How much extra space?

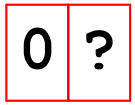
Double the array's size each time it needs to be bigger

- Wasted space for n items is $O(n)$
- Time to add n items: $O(n \log n)$
- On average, adding to n items takes $O(\log n)$

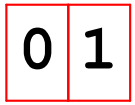
Array Doubling

Array Doubling

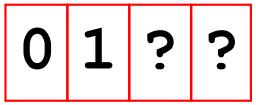
Array Doubling



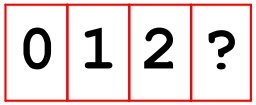
Array Doubling



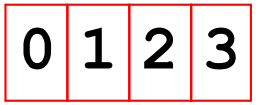
Array Doubling



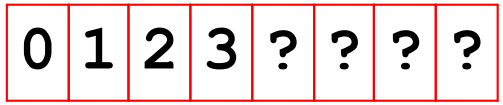
Array Doubling



Array Doubling



Array Doubling



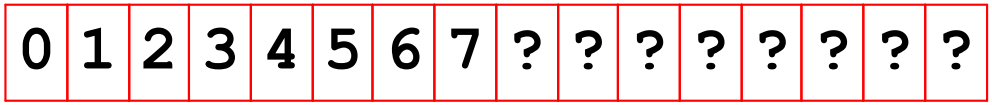
Array Doubling

Array Doubling

Array Doubling

Array Doubling

Array Doubling



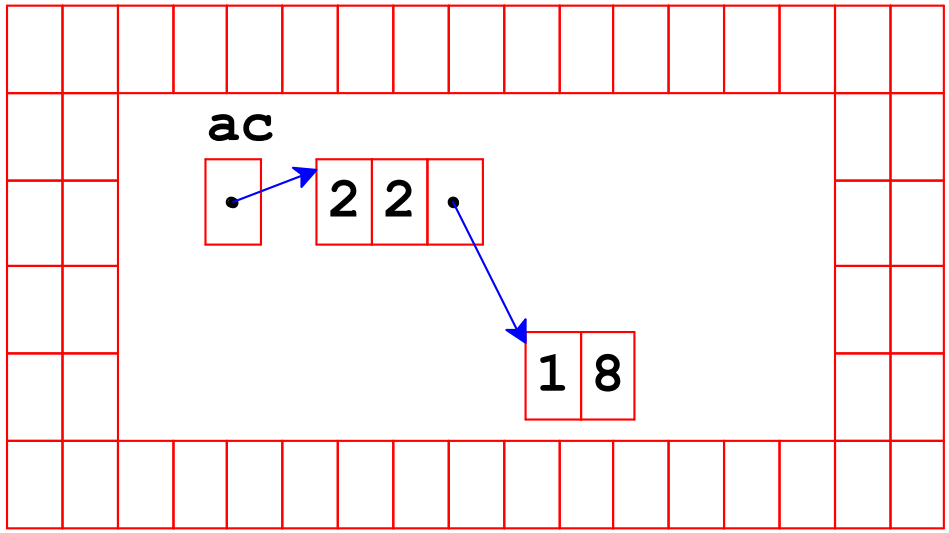
Array Doubling

If you need to double the size to add the *n*th item, then you've doubled only log *n* times so far

- Each doubling copied less than O(*n*) items
- Actually adding the item takes O(1)

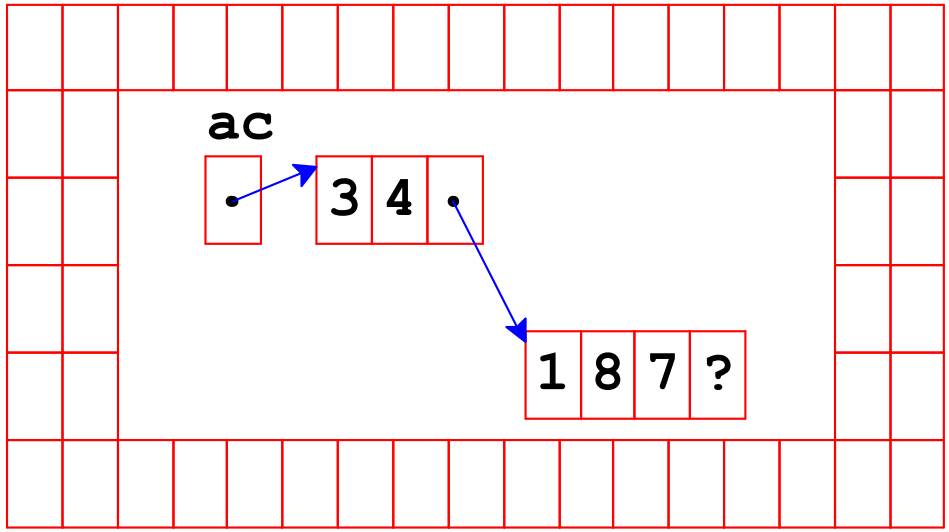
Array-Doubling Container

Before:



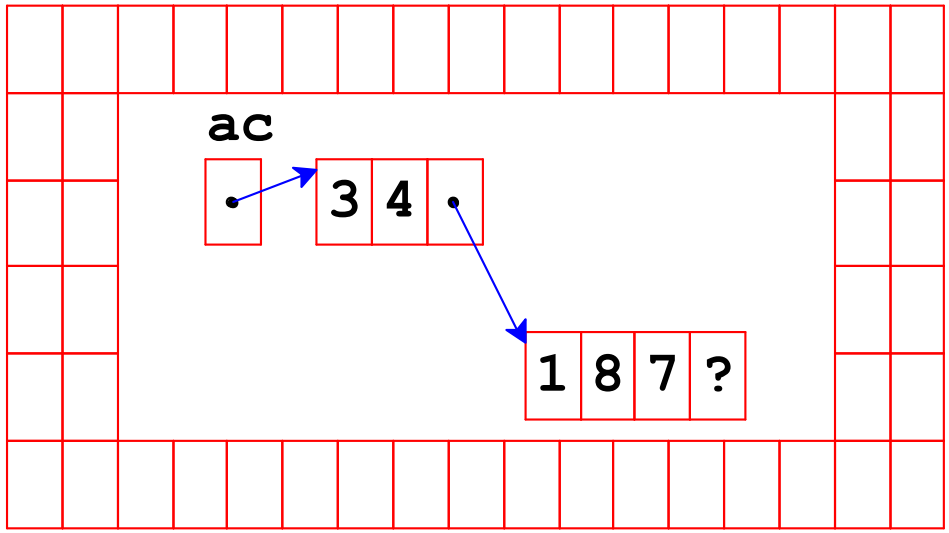

```
add_to_back(ac, 0);
```

After:



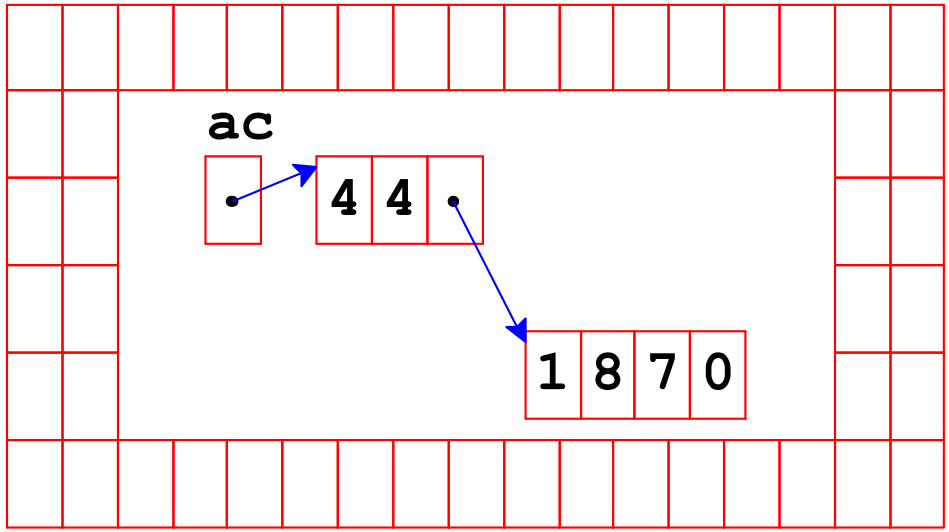
Array-Doubling Container

Before:




```
add_to_back(ac, 0);
```

After:



Circular Buffer

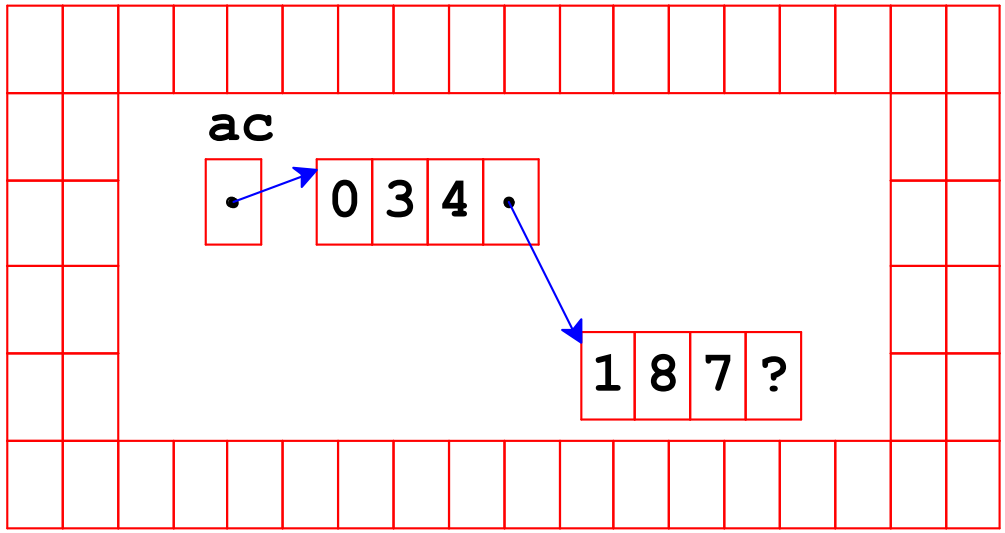
What if we want to add or remove at the front?

... without shifting all data?

- In addition to count, keep a starting point
- Array content can "wrap around" if items are added to the end

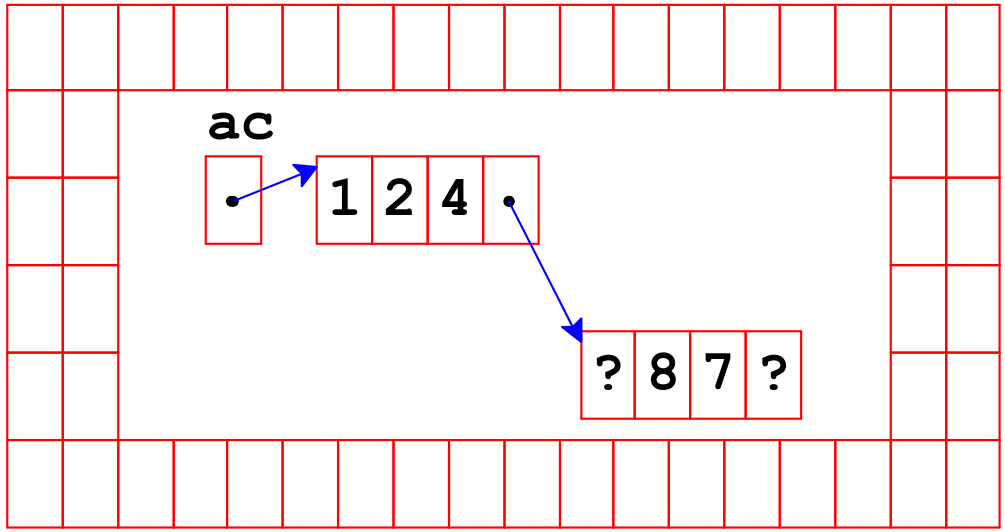
Circular Buffer

Before:



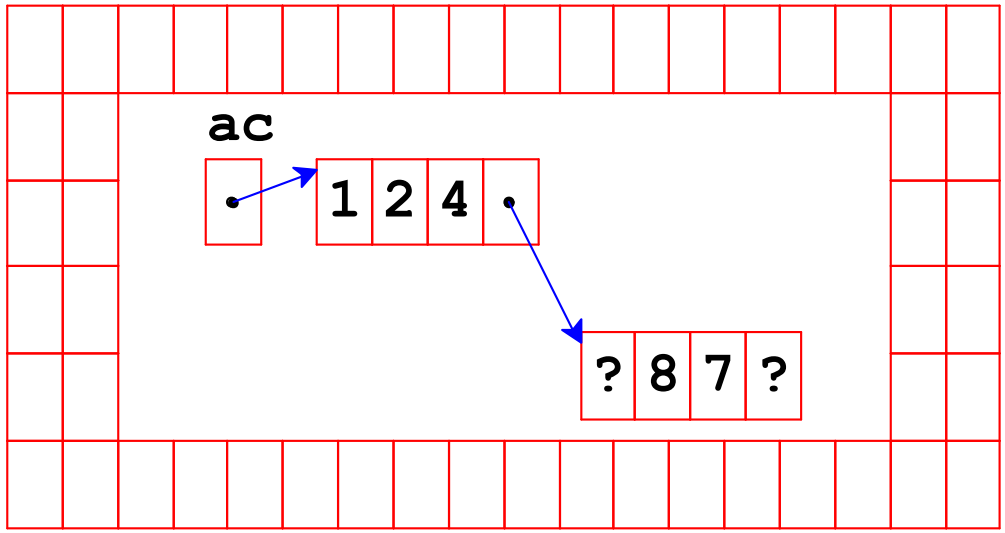

```
remove_from_front(ac);
```

After:



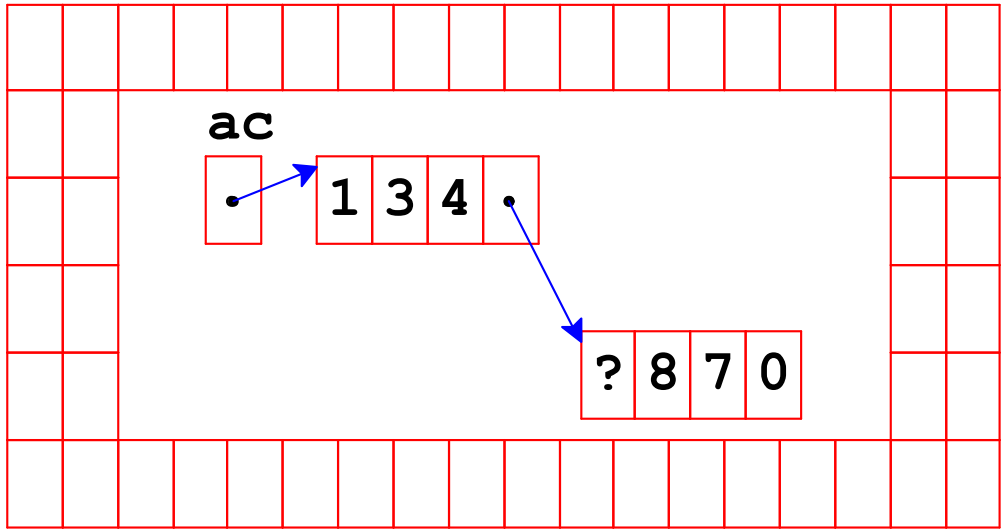
Circular Buffer

Before:



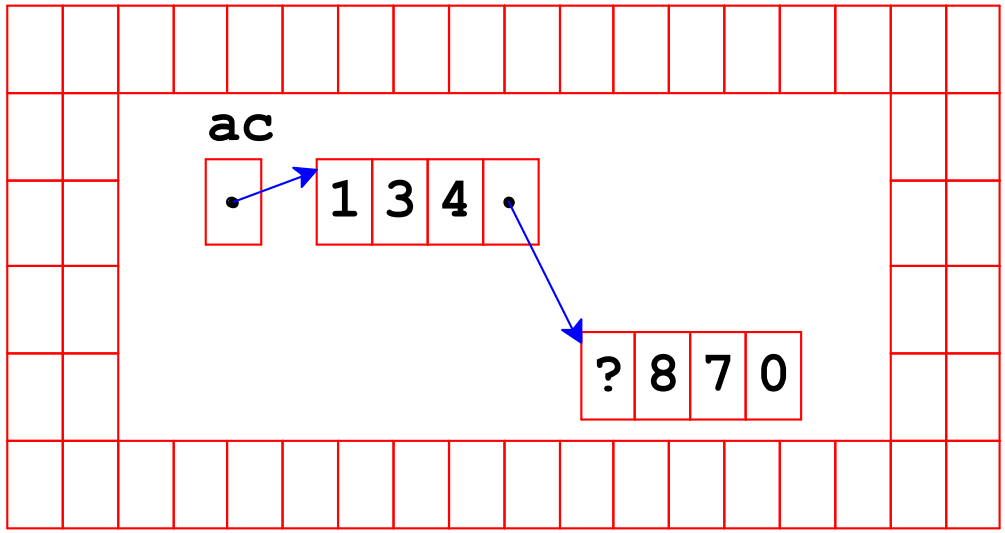

```
add_to_back(ac, 0);
```

After:



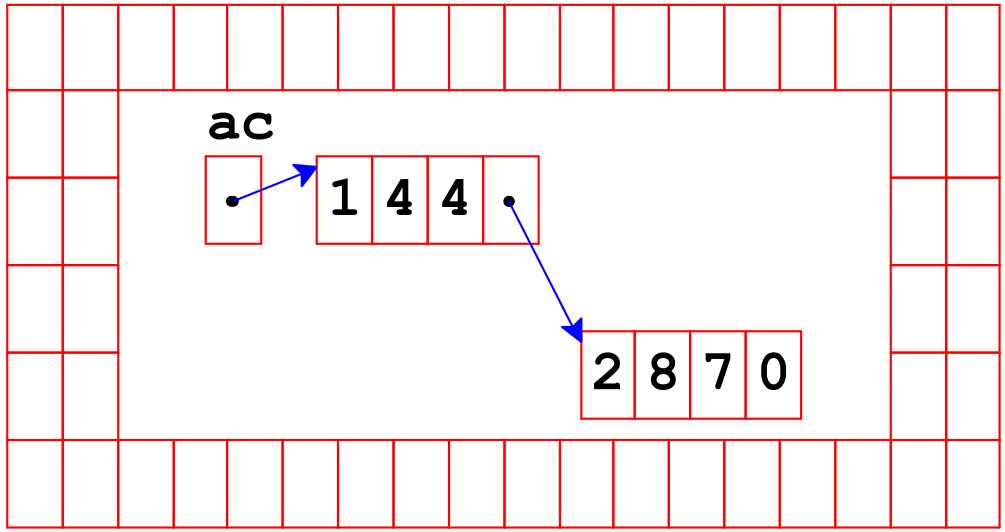
Circular Buffer

Before:



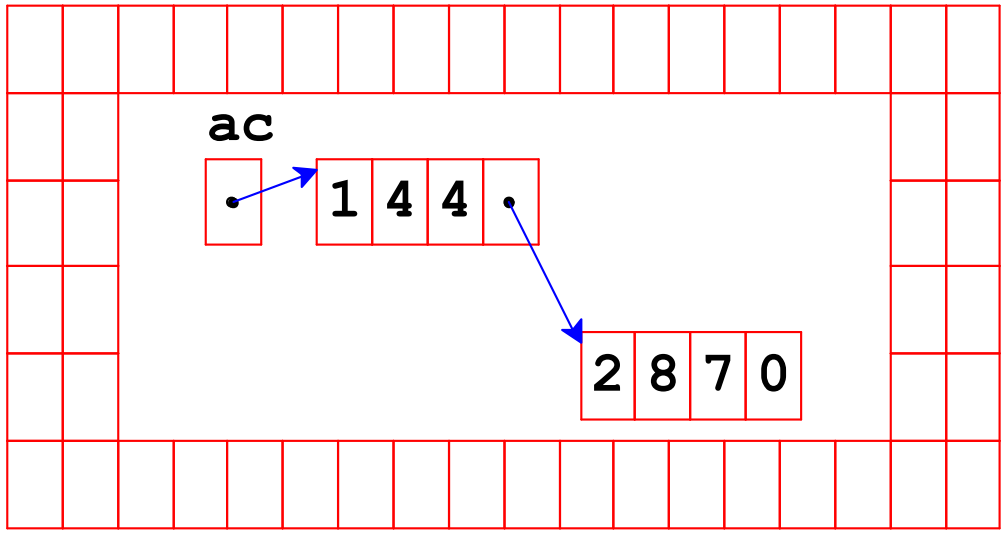

```
add_to_back(ac, 2);
```

After:



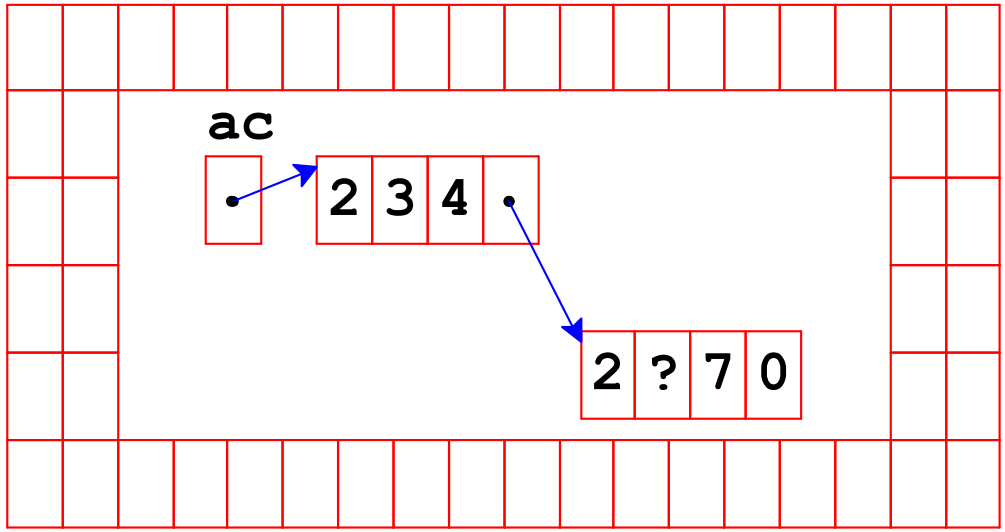
Circular Buffer

Before:




```
remove_from_front(ac);
```

After:



Stacks and Queues

Best kind of container depends on what you need

Two common patterns:

- ***stack***: add to one end, remove from same end

  array doubling (or even fixed size!) is common

- ***queue***: add to one end, remove from other end

  circular buffer is common

Linked Lists versus Arrays

Linked lists can implement stacks and queues

- ***stack***: plain linked list
- ***queue***: linked list tracking head and tail

Compared to array containers:

- advatage: closer to O(1) actual — not just average
- disadvatage: less compact and slower overall

Random access in middle: arrays are better

Inserting and deleting in middle: doubly linked list is better