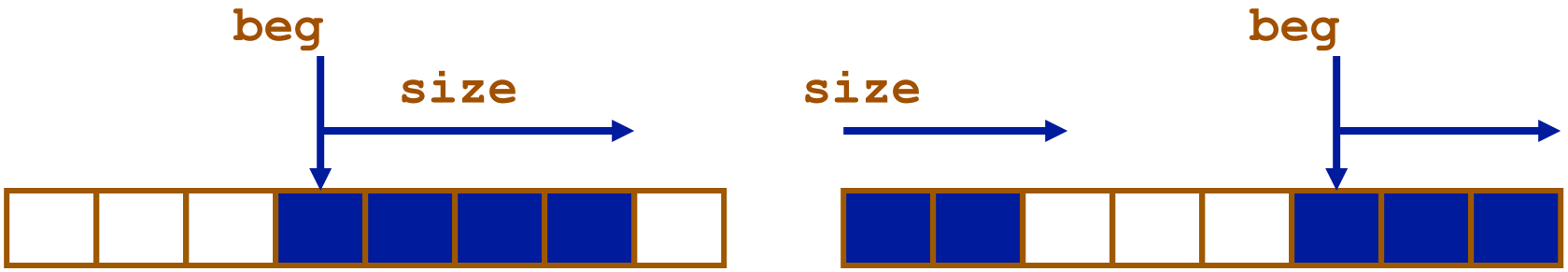
Dynamic Array Deque Implementation

Let the partially filled block "float"

- One solution: Rather than always use index zero as our starting point, allow the starting index to "float"
- Maintain two integer values:
  - Starting or beginning index (**`beg`**)
  - Count of elements in the collection (**`size`**)
- Still need to reallocate when size equal to capacity

Dynamic Array Deque

- First filled element no longer always at index 0
- Filled elements may wrap around back to the front end of array
- Called **`ArrDeque`**



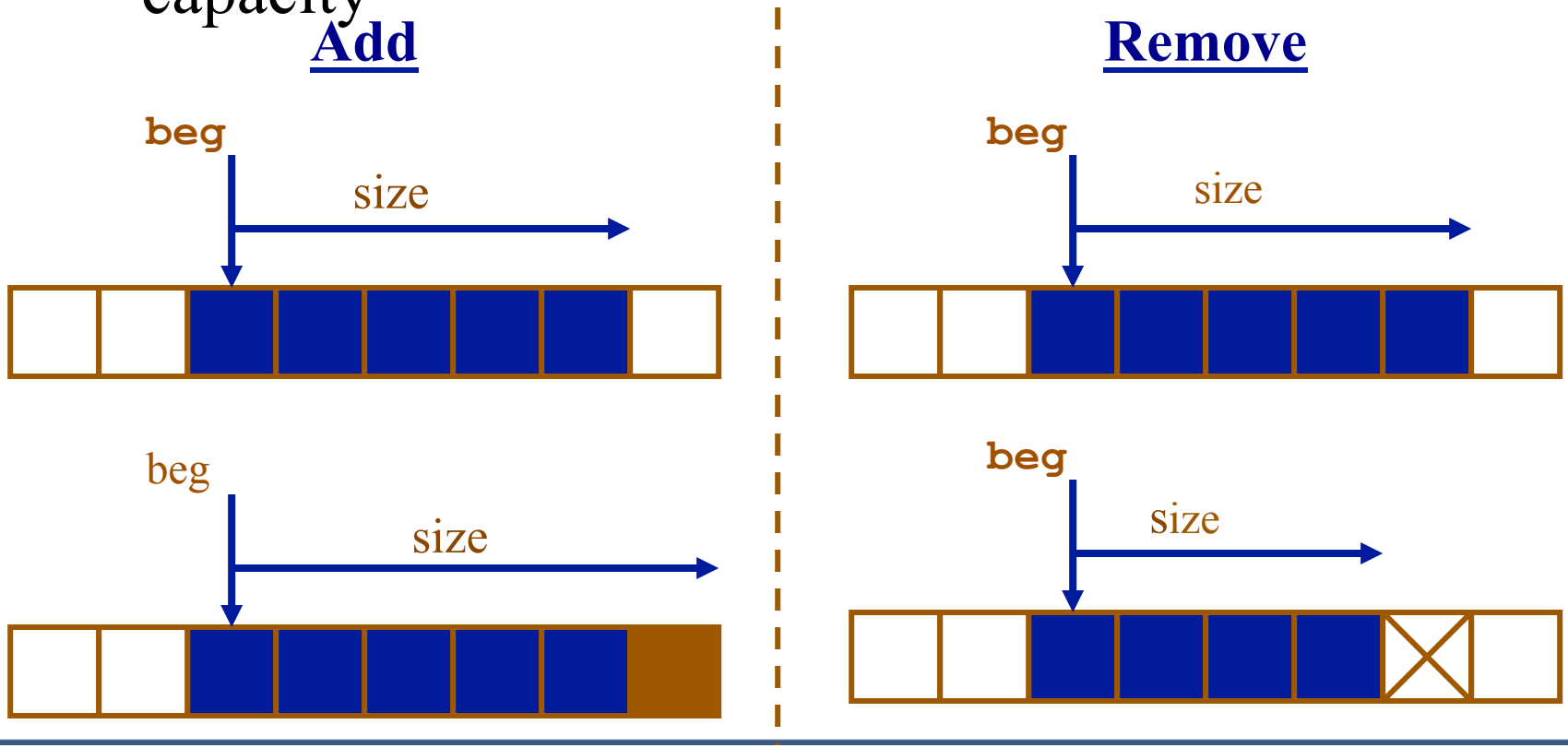
ArrayDeque Structure

```
struct ArrDeque {
  TYPE *data;   /* Pointer to data array. */
  int   size;    /* Number of elements in collection. */
  int   beg;    /* Index of first element. */
  int   cap;    /* Capacity of array. */
};

void initArrDeque(struct ArrDeque *d, int cap) {
  d->data = malloc(cap * sizeof(TYPE)));
  assert(d->data != 0);

  d->size = d->beg = 0;
  d->cap = cap;
}
```

Adding/Removing from Back

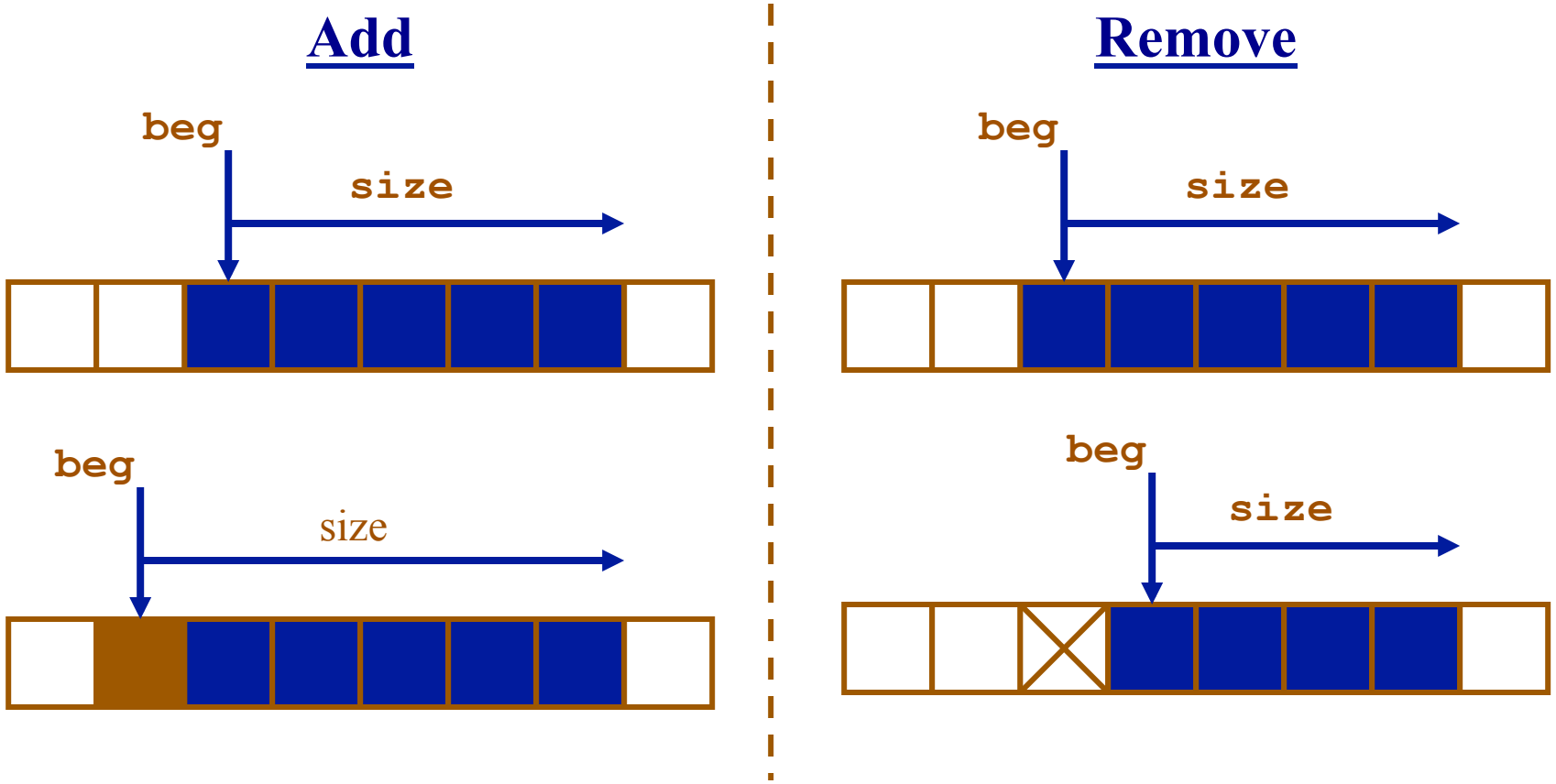
Adding to back is easy, just adjust count of the number of elements

- Still need to reorganize if adding and size = capacity



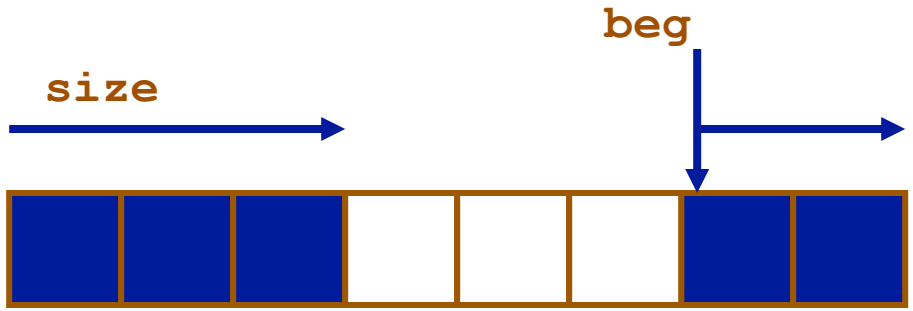
Adding/Removing from Front

Changes to front are easy, just adjust count and starting location



Wrapping Around

Problem: Elements can wrap around from beg to end



Wrapping Around

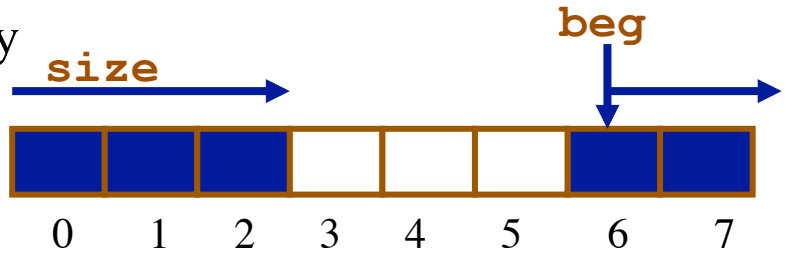
- Calculate offset: add logical (element) index to start (**beg**)

```
offset = beg + logicalIndex;  /* logIndex = 3, offset = 9 */
```

beg = 6
cap = 8

- If larger than or eq to capacity, subtract capacity

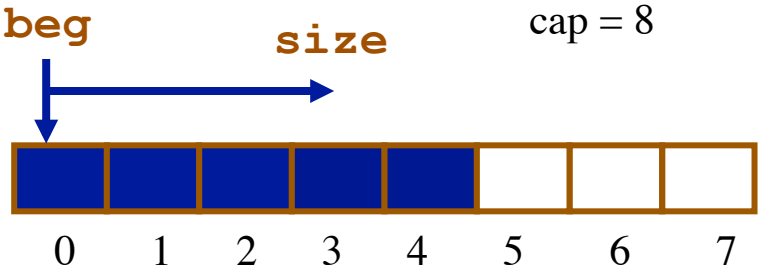
```
if (offset  >= cap)
    absoluteIndex = offset – cap;
```



- If smaller than zero, add cap

```
if (offset  < 0)
    absoluteIndex = offset + cap;
```

beg = 0
cap = 8

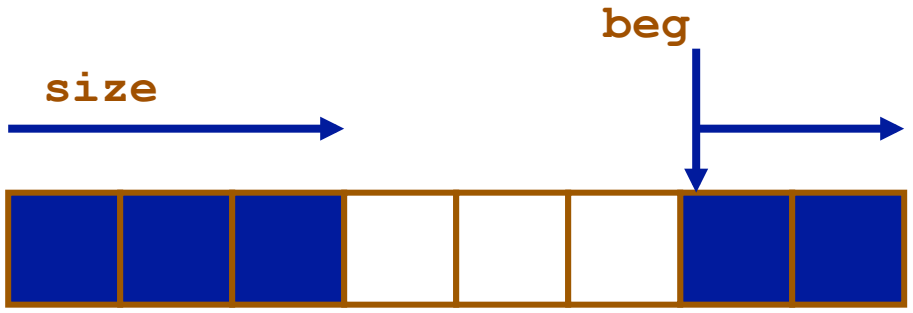


- That way sizes are always in range
- Or..combine into single statement with mod:

```
/* Convert logical index to absolute index. */
absIdx = (logicalIdx + beg) % cap;
```

Example

```
TYPE dequeBack (struct deque * d) {
    int index = d->beg + d->size - 1;
    if (index > d->capacity)
        index -= d->capacity
    return d->data[index];
}
```



beg = 6
Size = 5
Capacity = 8

Index = 10
Index = 10 – 8

Example: AddBack and RemoveBack

```
Void dequeAddBack (struct deque * d, TYPE newValue) {
    int index;
    if (d->size >= d->capacity)
        _dequeSetCapacity(d, 2*d->capacity);

    index = d->beg + d->size; //compute index
    if (index >= d->capacity) //wrap
        index -= d->capacity;
    d->data[index] = newValue;
    d->size++;
}


void dequeRemoveBack (struct deque *d) {
    assert(d->size >  0);
    d->size--;
}
```

Resizing?

- Can we simply copy the elements to a larger array?
- Have to be careful because the wrapping is dependent on the ‘capacity’

Operations Analysis

Operation	Best	Worst	Ave
AddBack	1	n	1+
RemoveBack	1	1	1
AddFront	1	n	1+
RemoveFront	1	1	1

Your Turn…

- Complete Worksheet 20