CS 2110, SP24

Discussion 10: Shared Buffers

Bounded Queue & Ring Buffers

Bounded Queue ADT (`BoundedQueue.java`)

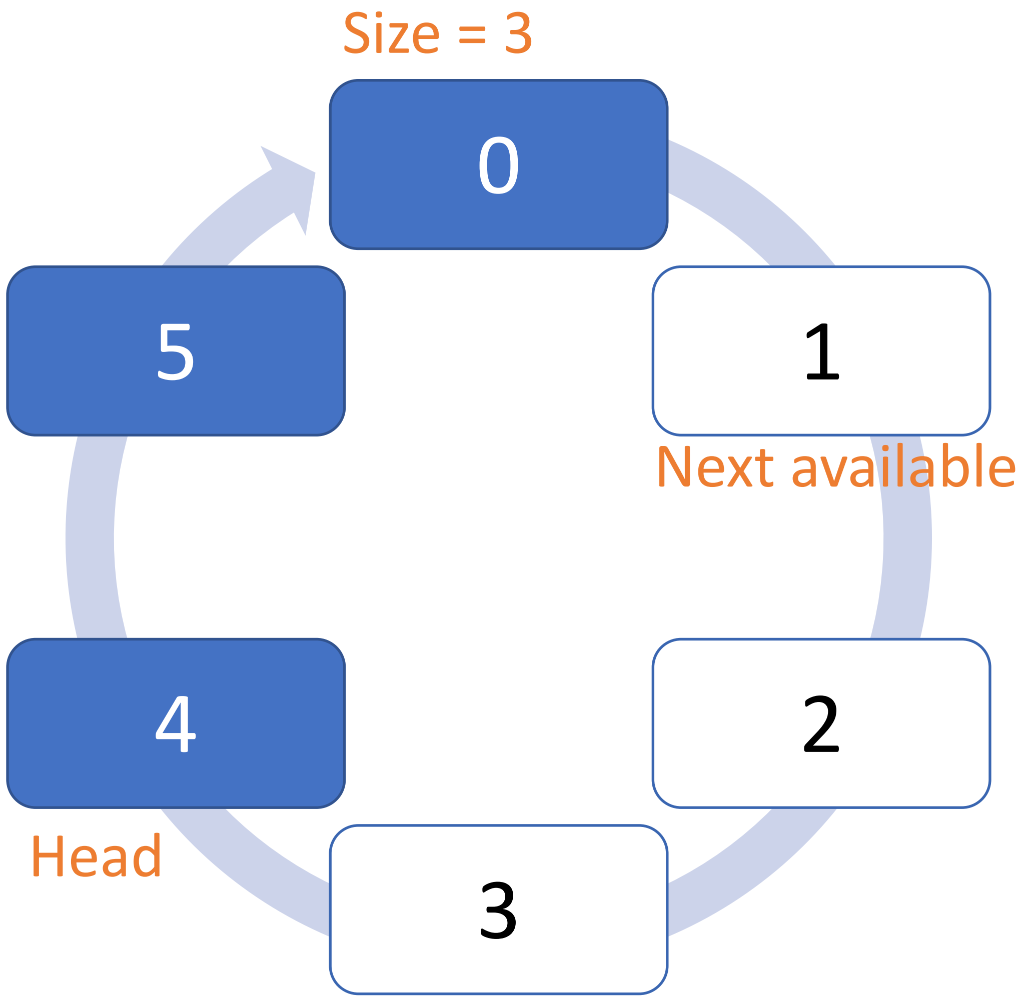
Queue (FIFO) with a fixed capacity.

Operations:

- `put()` – inserts only if capacity is not met.
- `get()` – removes oldest value if the queue is not empty.
- `isFull()`
- `isEmpty()`

Ring Buffer Data Structure

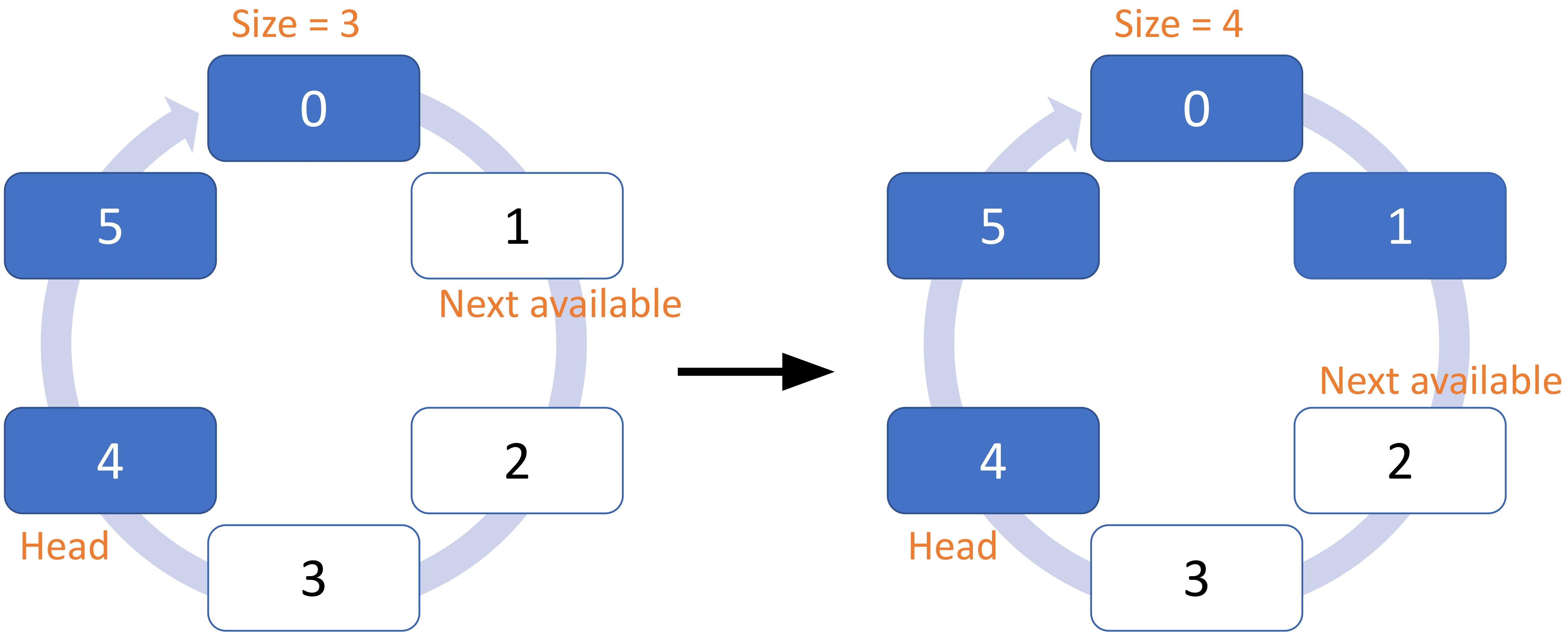
- Implements Bounded Queue
- Elements stored in fixed-capacity array
  - Additional state: head pointer, size



Ring Buffer Data Structure

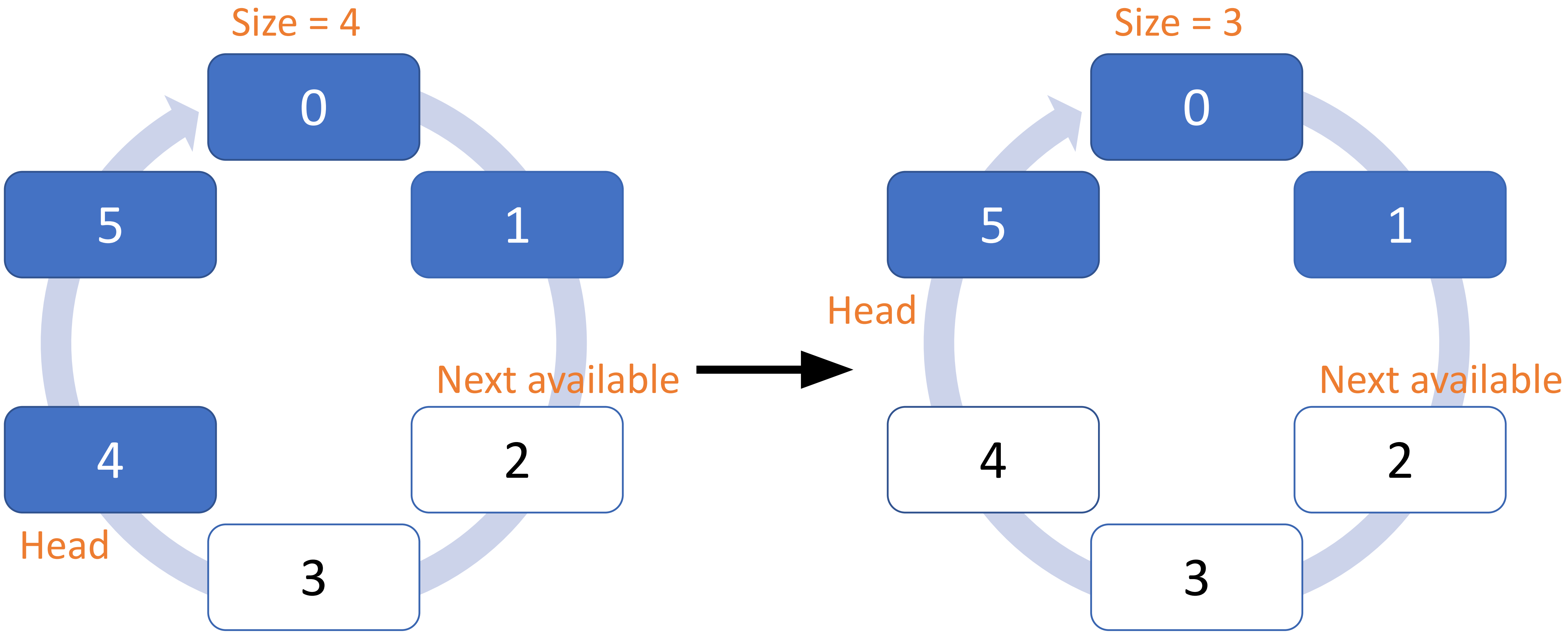
Put: store in next available index (requires size < capacity)

- `(head + size) % capacity`



Ring Buffer Data Structure

Get: advance head, return previous value (requires size > 0)



Review: Iterators

Java Iterator

- Generic interface expressing Iterator ADT
- Methods:
  - `boolean hasNext();`
  - `T next();`

Usage:

```
Iterator<String> it = ...;
while (it.hasNext()) {
  String s = it.next();
  // Do something with s
}
```

Enhanced for-loops

```
List<String> names = …;
for (int i=0; i<names.size(); ++i)
{
    String name = names.get(i);
    …
}
```

```
List<String> names = …;
for (String name : names) {
    …
}
```

… are translated into while loops ("syntactic sugar")

```
List<String> names = …;
for (String name : names) {
    …
}
```

```
List<String> names = …;
Iterator<String> it =
  names.iterator();
while (it.hasNext()) {
    String name = it.next();
    …
}
```

Iteration interfaces

`Iterable<T>` - `RingBufferBQ`

- "Something that can be iterated over"
- Can use in an enhanced for-loop
- Yields Iterators

- `Iterator<T> iterator();`

`Iterator<T>` - `RingBufferBQIterator`

- Helper class for actually doing the iteration
- Mutable (one-time use) - need a new one for each loop
- Yields values

- `boolean hasNext();`
- `T next();`

Nested classes

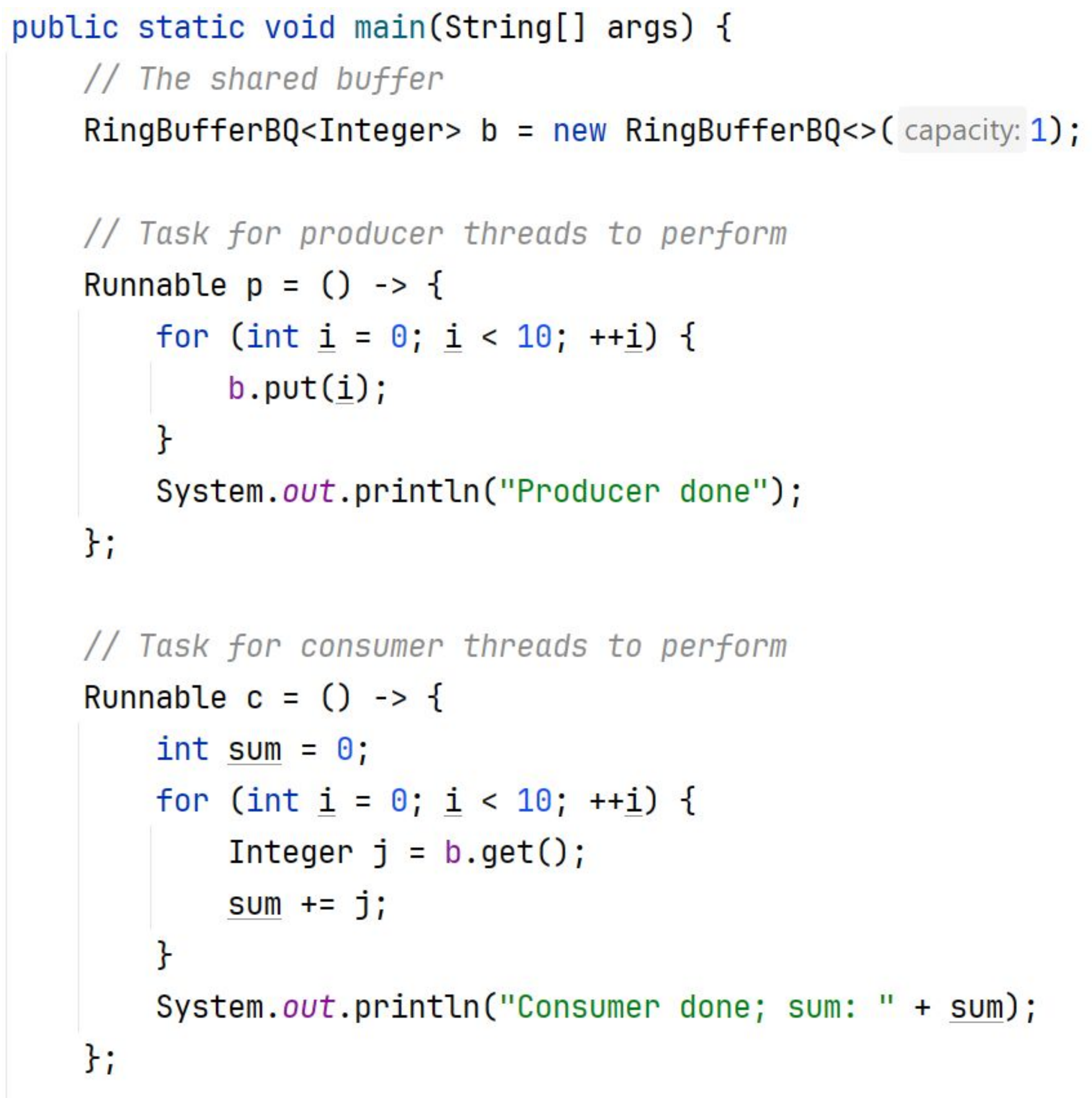
- Classes declared inside other classes (usually a "helper" of some kind)
- Static: Outer class acts as a namespace, can hide class from other potential clients
- Non-static ("inner classes"): Inner class objects are attached to an outer class *instance*
  - Can only be created from an instance of the outer class
  - Can access outer object's fields and methods
  - Common choice for Iterators
    - Enables more encapsulation (private fields)

Shared Buffers

Producer/consumer pattern (example)

- One or more fry cooks slides new fries onto the “ready” shelf
    - Producer
- One or more cashiers take fries from the “ready” shelf to complete orders
    - Consumer
- Shelf can only hold so many fries
    - Bounded queue

RingBufferBQ.main()



```java
public static void main(String[] args) {
    // The shared buffer
    RingBufferBQ<Integer> b = new RingBufferBQ<>( capacity: 1);

    // Task for producer threads to perform
    Runnable p = () -> {
        for (int i = 0; i < 10; ++i) {
            b.put(i);
        }
        System.out.println("Producer done");
    };

    // Task for consumer threads to perform
    Runnable c = () -> {
        int sum = 0;
        for (int i = 0; i < 10; ++i) {
            Integer j = b.get();
            sum += j;
        }
        System.out.println("Consumer done; sum: " + sum);
    };
```

A single shared buffer

Producer Threads:
Put numbers 0..9 into buffer

Consumer Threads:
Sum 10 values from buffer

Spin loop

```
while (COND) { /* spin */ }
```

where COND is true if the resource **shouldn’t** be accessed.

Note: Do **NOT** do this!!!!! (outside of this discussion section)

- We will see why this is a bad idea very soon.