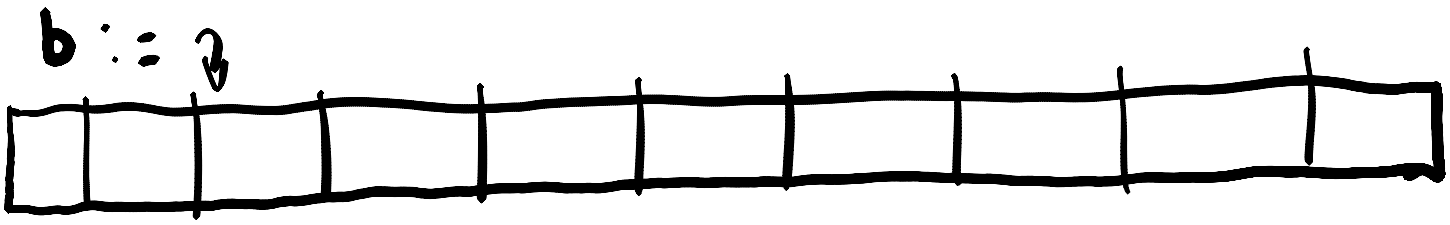
Ring Buffer



b[i] ≃ CBOR Buffer of size CBOR_BUFFERSize
(check ringbuffer.h)

len(b) ≃ BUFFER_SIZE

Algorithm:

- For the point of view of the algoritham the buffer is infinite.

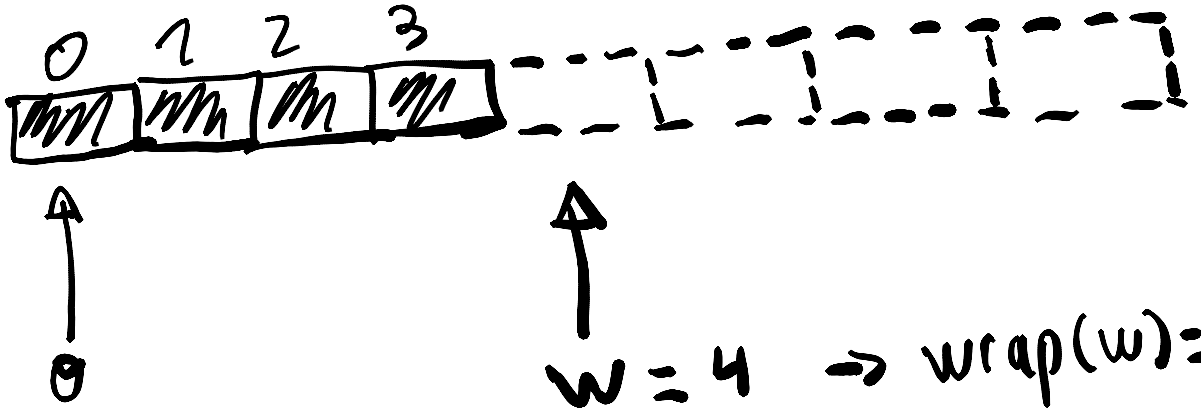
- We just wrap the index once we write.

—— = Real Space

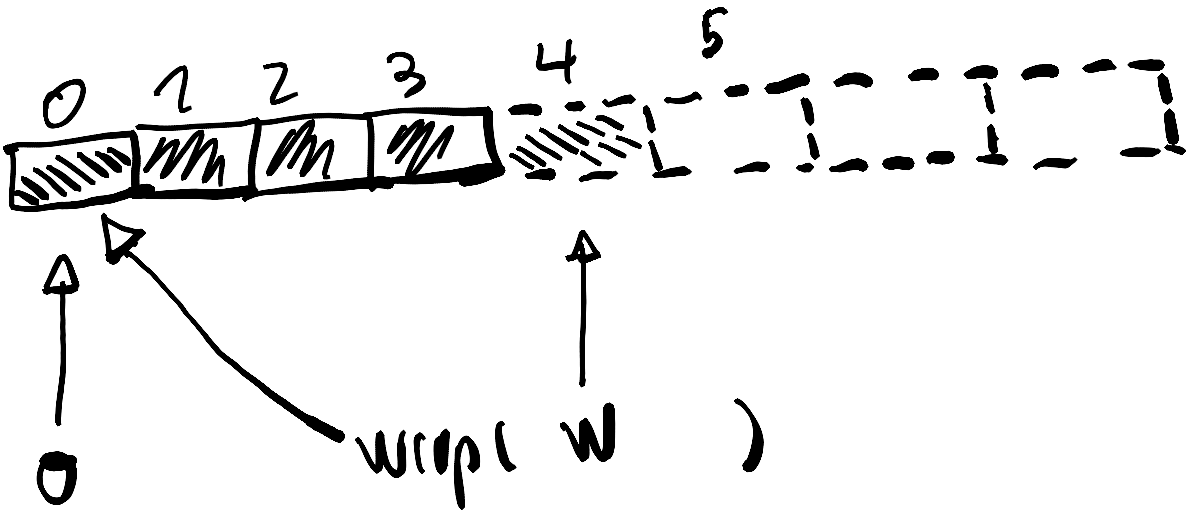
---- = Fake Space

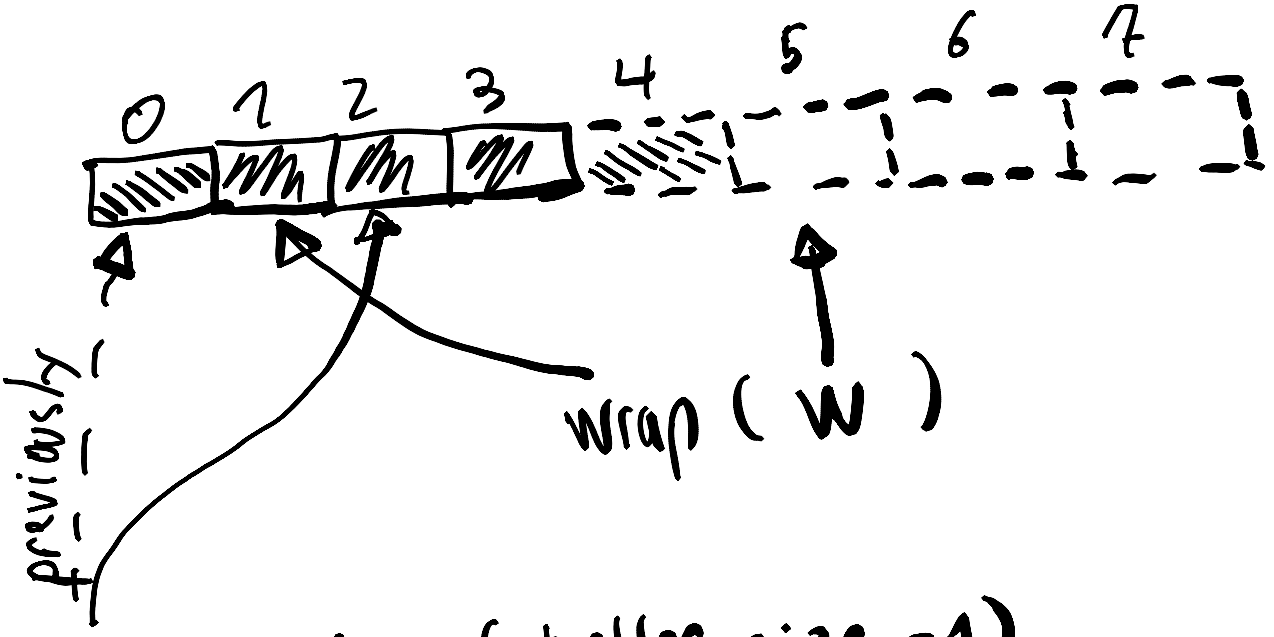
mmmm = Occupied

BUFFER_SIZE = 4



if wrap(w) == 0:
    full ← True





$0 \leftarrow W - (\text{buffer\_size} - 1)$

$= 5 - (3) = 2$

- So after the buffer becomes full for the first time, the distance between W and 0 is $W - 0 + 1 = \text{BUFFER-SIZE}$

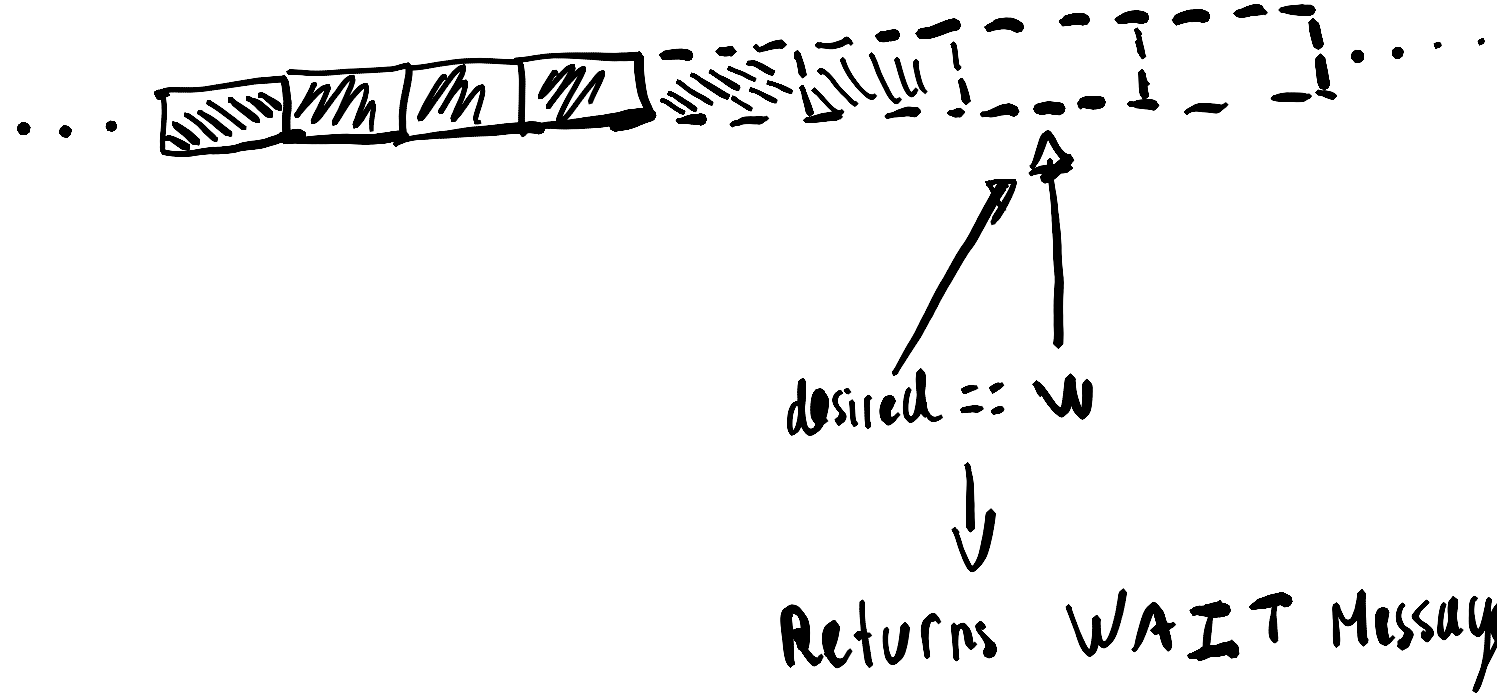
- This ensures that reader never overtakes the writer, but the writer can overwrite the oldest_idx which might be used by a Read Token.

Read Token:

- If not initialized starts with:

  token.idx ← oldest_idx
  (to read)

- Otherwise: desired ← token.idx + 1



- Gap between w and desired should be smaller than buffer size.
- If not, then reader has been slow so

we set its token.idx ← oldest_idx.

- Finally we read from token.idx pos in the ring buffer.

To do:

- Check possible concurrency issues.
  - Reader grabs data from ring buffer struct, performs operations based on it, and then uses that. But what if publisher has already modified that data.

- Implement unit tests from