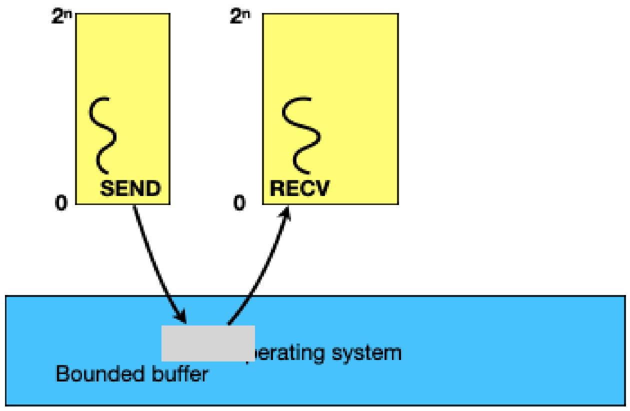
Virtualization Wrap-up

1. Goal: give each module the illusion of its own computer:
    a. It's own interpreter: thread
    b. It's own memory: Virtual memory
    c. It's own communication link: bounded buffer
2. Use multiplexing, aggregation, and emulation to provide these illusions.
3. Two types of multiplexing:
    a. Space multiplexing: split storage into chunks and divy out chunks
    b. Time multiplexing: split time into chunks and divy out chunks.
4. A key task of an **Operating System**
    a. Privileged software (they can do things that normal modules cannot)
    b. Provide system calls that perform operations on a user's behalf
    c. There's LOTS of other things that an OS does (but, sadly, we won't talk in CSE130).
5. Final abstraction to discuss, the bounded buffer...

Bounded Buffer :

1. Provide abstraction of having a communication channel within a system



    a. We'll consider them unidirectional (if you want bidirectional, use two!)
    b. Bounded...
        i. System can't buffer infinite data
        ii. Gives producer/consumer better programming model
    c. Send()
        i. Send bytes to the other side
        ii. Block if there is not room to send any bytes
    d. Recv()
        i. Receive "oldest" bytes sent by other side
        ii. Block if there are no bytes to send

2. Examples: Pipes!
    a. printf “PUT /hello HTTP/1.1\r\nContent-Length:6\r\n\r\nHello\n” | nc localhost 1234
    b. Echo “hello there.” | ./split h -

Synchronization and Concurrency

1. What scary things seem possible in this picture?
    a. Sender and Receiver rely on “empty” and “full” signals
    b. Send and Receive called by **separate threads**!
    c. Can they be **certain** that they know how much data is in the buffer?
    d. What happens if send and receive get out of sync??
2. Bounded Buffers are what we call “the producer/consumer problem”:
    a. Sender is a producer–it creates data
    b. Receiver is a consumer–it consumes data
    c. Originally posed by Djikstra in the 60s
3. Let’s dive into actual pseudocode (next page)
    a. A bounded buffer using a **circular array**.
    b. Example execution of using a bounded buffer:
    c. Source code of Bounded Buffer:
4. What problems does this have???
    a. Is it good to have a while(){} loop? No!
    b. is counter += 1/counter -=1 atomic? NO! What can happen?

```
// producer-consumer—we want to store up to N ints
Globals:
 int buffer[N];   // the buffer itself
 int in = 0;      // the position of the producer
 int out = 0,     // the position of the consumer
 int counter = 0; // number of items in buffer
```

```
//Producer:
void produce(int newi) {

  //wait if full:
  while (counter == N) {}

  //place in buffer:
  buffer[in] = newi;
  in = (in + 1) % N;
  counter += 1;
}
```

```
//Consumer:
int consumer(void) {
  int rtn;

  //wait if empty:
  while (counter == 0) {}

  //place in buffer:
  rtn = buffer[out];
  out = (out + 1) % N;
  counter -= 1;

  return rtn;
}
```