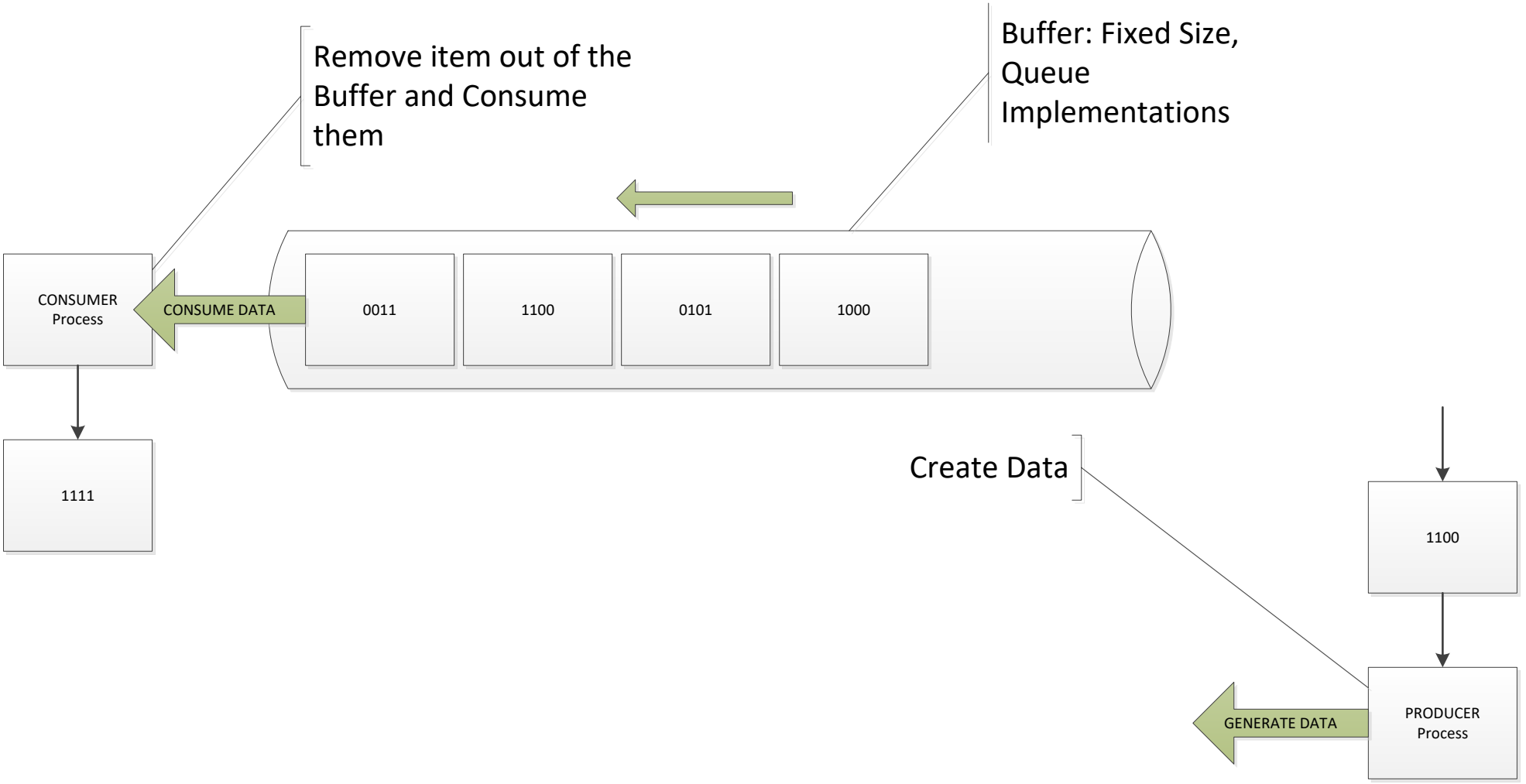
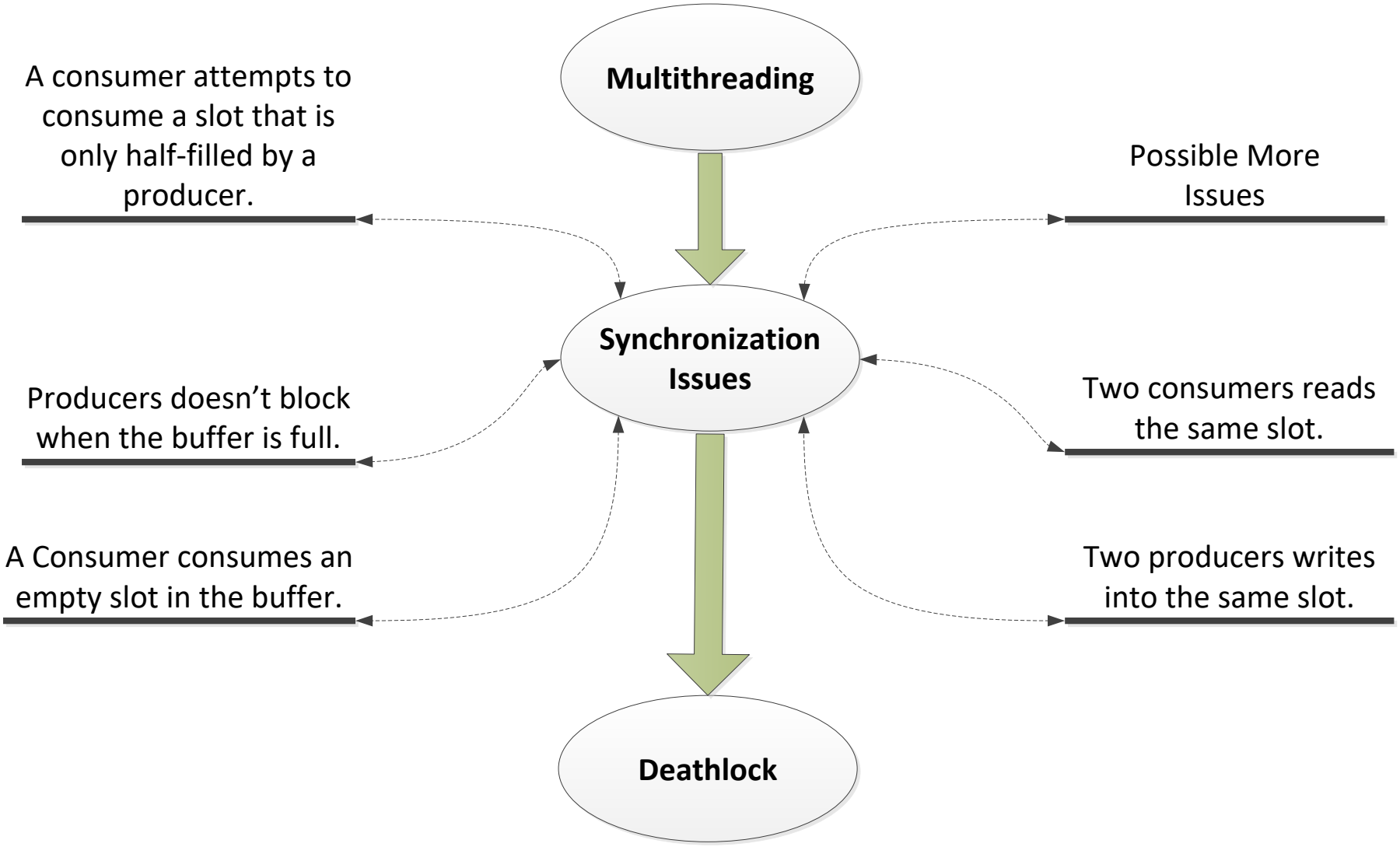
Producer Consumer Problem

TRUC HUYNH

- A well-known multi-process synchronization problem (proprosed by Edsger W. Dijkstra). It is also call Bound-Buffer problems.
- The problem describes two processes,[4] the producer and the consumer:
  - The producer generate data, push it into the buffer
  - Consumer is consuming the data one piece at a time

NORMAL OPERATION

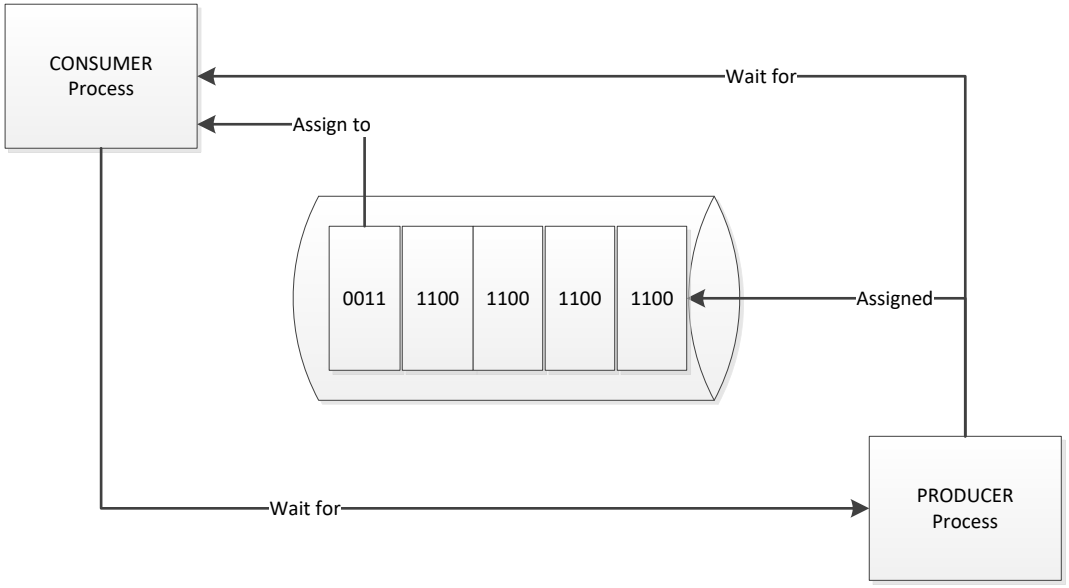




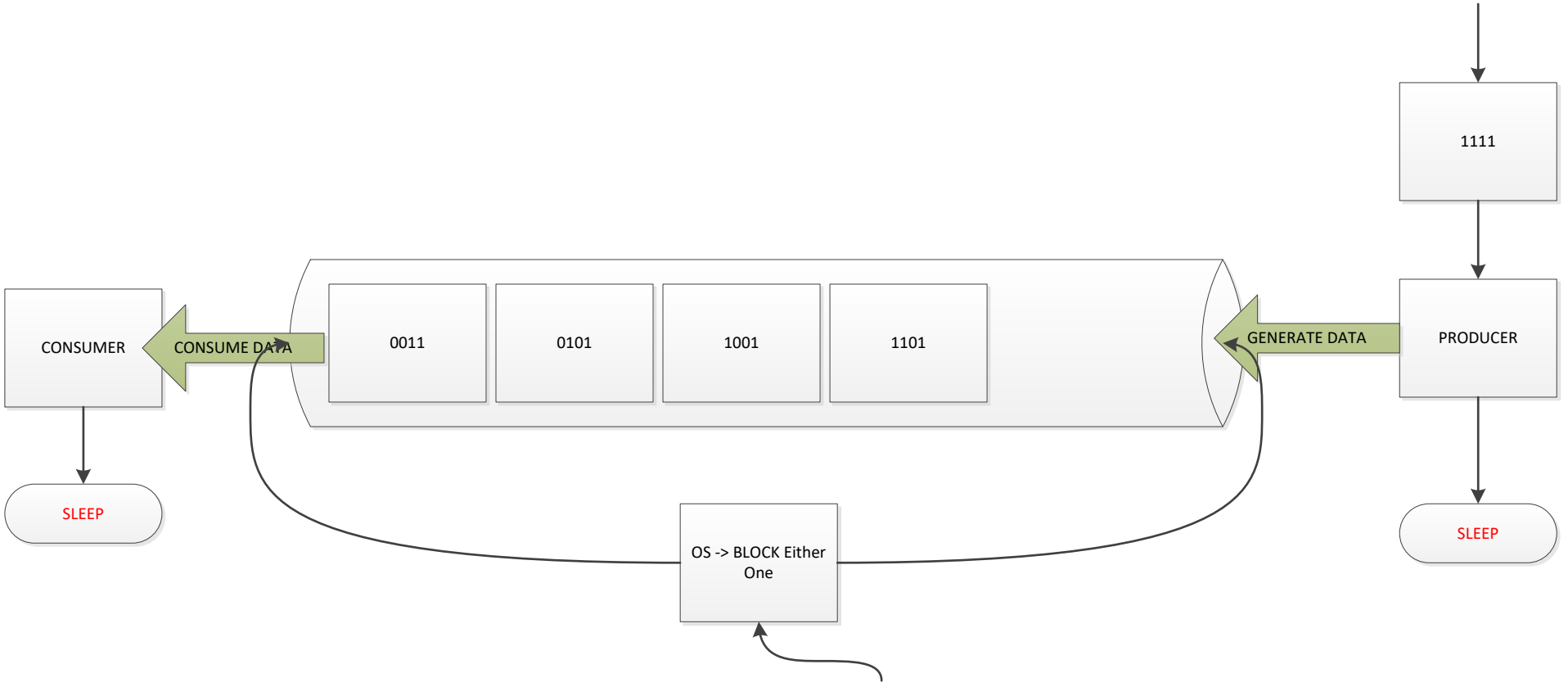
Retrieved from [4]

DEATH LOCK

Occurs when a process or thread enters a **waiting state** because a requested system resource is held by another **waiting process**, which in turn is waiting for another resource held by another waiting process [4].



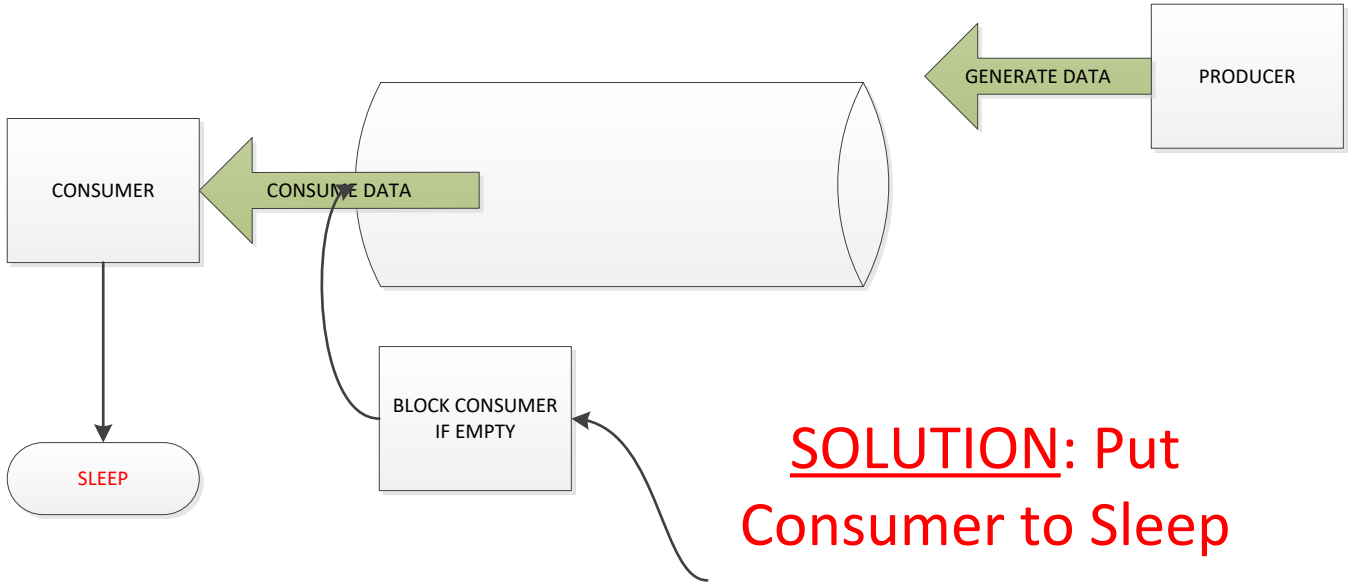
2 PROCESS OPERATE AT THE SAME TIME



SOLUTION: Put either Producer or Consumer to sleep

CONSUME EMPTY BUFFER

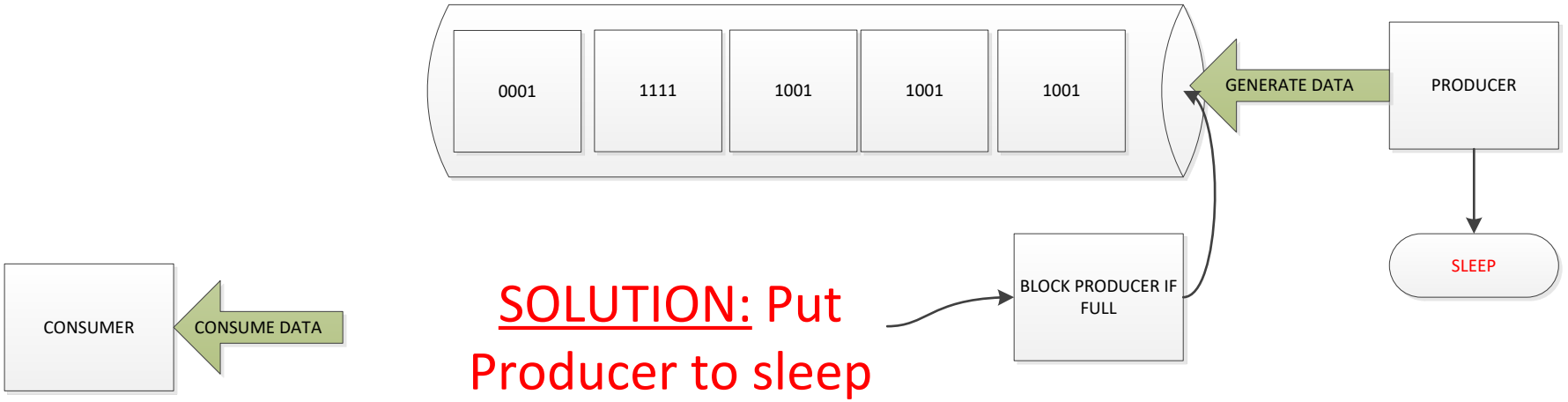
- In normal operation, the consumer can go to sleep if it finds the buffer to be empty [4].
- The next time the producer puts data into the buffer, it wakes up the sleeping consumer [4].
- could result in a deadlock when both processes are waiting to be awakened.



SOLUTION: Put Consumer to Sleep

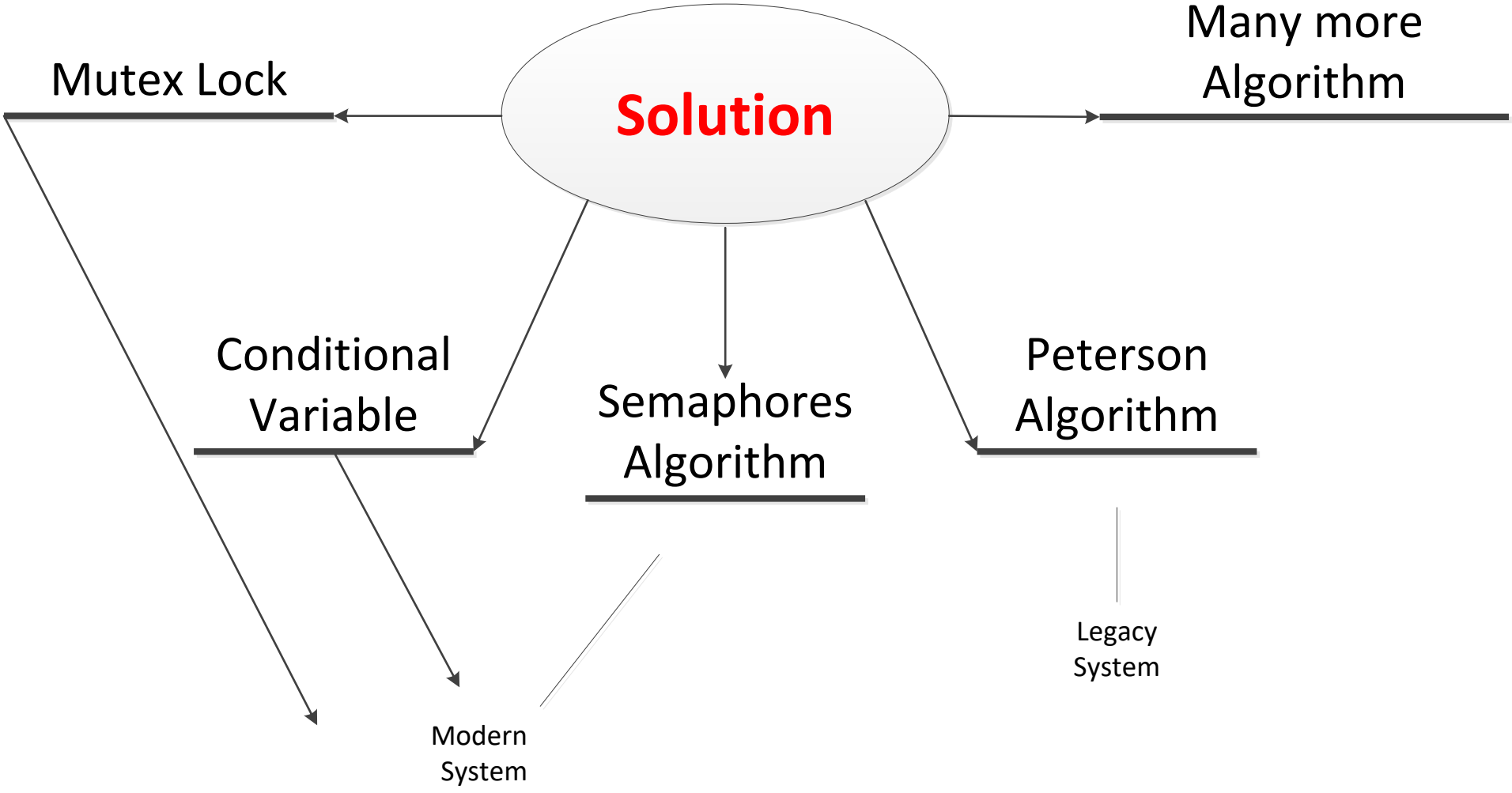
ADD TO FULL BUFFER

- In normal operation, the producer is to either go to sleep or discard data if the buffer is full [4].
- The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again [4].
- Could result in a deadlock when both processes are waiting to be awakened [4].

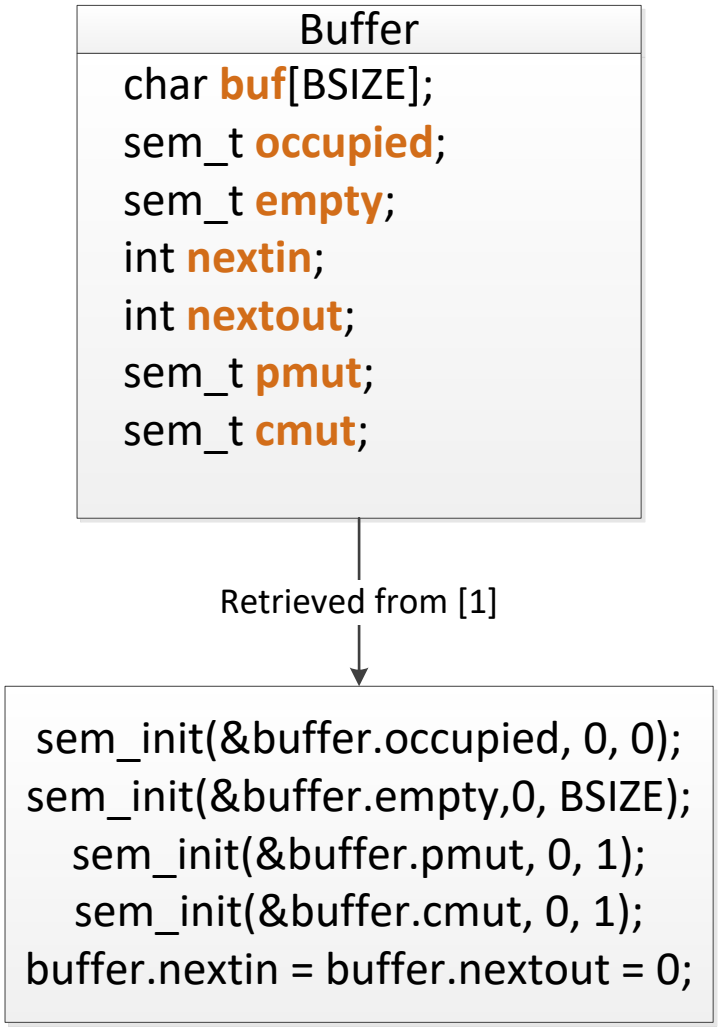


SOLUTION: Put Producer to sleep

JAVA IMPLEMENTATION



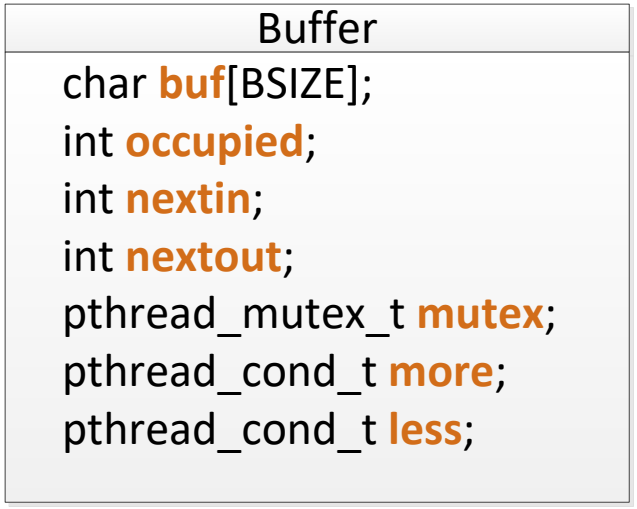
Producer and Consumer Problem With Semaphores



To use a **semaphore** (designed by E. W. Dijkstra in the late 1960), the thread that wants access to the shared resource tries to acquire a permit [2].

- If the **semaphore's count** (sem_t) is greater than zero, then the thread acquires a permit, which causes the **semaphore's count** to be decremented.
- Otherwise, the thread will be blocked until a permit can be acquired.
- When the thread no longer needs an access to the shared resource, it releases the permit, which causes the semaphore's count to be incremented.
- If there is another thread waiting for a permit, then that thread will acquire a permit at that time.

Producer and Consumer Problem With Condition Variables



Retrieved from [1]

Example The Producer/Consumer Problem--the Producer

```
void producer(buffer_t *b, char item)
{
   pthread_mutex_lock(&b->mutex);
      while (b->occupied >= BSIZE)
       pthread_cond_wait(&b->less, &b->mutex);
   assert(b->occupied < BSIZE);
   b->buf[b->nextin++] = item;
   b->nextin %= BSIZE;
   b->occupied++;

   /* now: either b->occupied < BSIZE and b->nextin is the index
      of the next empty slot in the buffer, or
      b->occupied == BSIZE and b->nextin is the index of the
      next (occupied) slot that will be emptied by a consumer
      (such as b->nextin == b->nextout) */

   pthread_cond_signal(&b->more);
   pthread_mutex_unlock(&b->mutex);
}
```

Example The Producer/Consumer Problem--the Consumer

```
char consumer(buffer_t *b)
{
   char item;
   pthread_mutex_lock(&b->mutex);
   while(b->occupied <= 0)
      pthread_cond_wait(&b->more, &b->mutex);
   assert(b->occupied > 0);
   item = b->buf[b->nextout++];
   b->nextout %= BSIZE;
   b->occupied--;

   /* now: either b->occupied > 0 and b->nextout is the index
     of the next occupied slot in the buffer, or
     b->occupied == 0 and b->nextout is the index of the next
     (empty) slot that will be filled by a producer (such as
     b->nextout == b->nextin) */

   pthread_cond_signal(&b->less);
   pthread_mutex_unlock(&b->mutex);

   return(item);
}
```

References

Oracle(n.d.) *The Producer/Consumer Problem*. Retrieved from https://docs.oracle.com/cd/E19455-01/806-5257/sync-31/index.html **[1]**

Miglani., G.(2018) *Semaphore in Java.* Retrieved from https://www.geeksforgeeks.org/semaphore-in-java/ **[2]**

Shah., K.; Rithesh (2020) *Introduction of Deadlock in Operating System.* Retrieved from: https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/ **[3]**

Yadav, G.,(2019) *Producer-Consumer solution using threads in Java.* Retrieved from https://www.geeksforgeeks.org/producer-consumer-solution-using-threads-java/ **[4]**