**1. Data Structures**

* **DataPacket**: A struct that holds a buffer (data) to store the fetched data, along with a size field.
* **SharedQueue**: A struct for the shared queue that holds multiple DataPacket structures. The queue uses a circular buffer. This ensures that both the front and rear pointers are always within the bounds of the queue size and that space is efficiently reused.
  + Array (queue) to store the data packets.
  + Front pointer to track the next available packet for reading.
  + Rear pointer to track where to insert the next packet.
  + A count variable to track the number of items in the queue.
  + Mutex (mutex) and condition variables (not\_empty, not\_full) to manage synchronization and avoid data corruption.

**2. Thread Synchronization**

* **Mutex**: Used to protect the shared queue. Only one thread (either a writer or reader) can access the queue at a time.
* **Thread blocking:** Writers should be blocked when the queue is full, and readers should be blocked when the queue is empty. This is managed by the condition variables not\_empty and not\_full.
* **Condition Variables**:
  + **not\_empty**: Signaled by writers when new data is added to the queue.
  + **not\_full**: Signaled by readers when space in the queue becomes available, allowing the writers to continue writing data.

**3. Writer Threads**

* Writer threads continuously fetch data from the external source using the get\_external\_data() function. This data is stored in a DataPacket struct.
* Writer locks the queue.
* Writers then attempt to insert the data packet into the shared queue. If the queue is full, the writer thread waits on the not\_full condition variable.
* Once data is successfully added to the queue, the writer signals the not\_empty condition variable, allowing a reader to begin processing.
* Writer unlocks the queue.

**4. Reader Threads**

* Reader locks the queue.
* Reader threads wait for data to become available in the shared queue. If the queue is empty, the reader thread waits on the not\_empty condition variable.
* When data is available, reader processes the data packet using the process\_data() function, and then signals the not\_full condition variable to notify the writers that space is available in the queue.
* Reader now unlocks the queue.