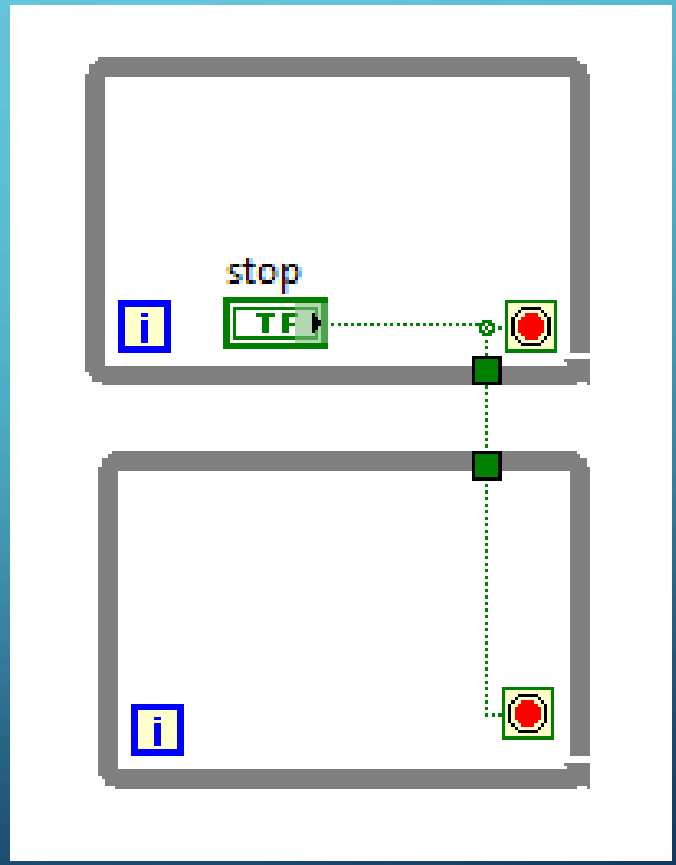
DATA MANIPULATION AND TRANSFER

INCLUDES INFORMATION ABOUT: PASSING DATA, QUEUES, NOTIFIERS, GLOBAL VARIABLES, FUNCTIONAL GLOBAL VARIABLES

TRANSFERRING DATA FROM TWO DIFFERENT WHILE LOOPS

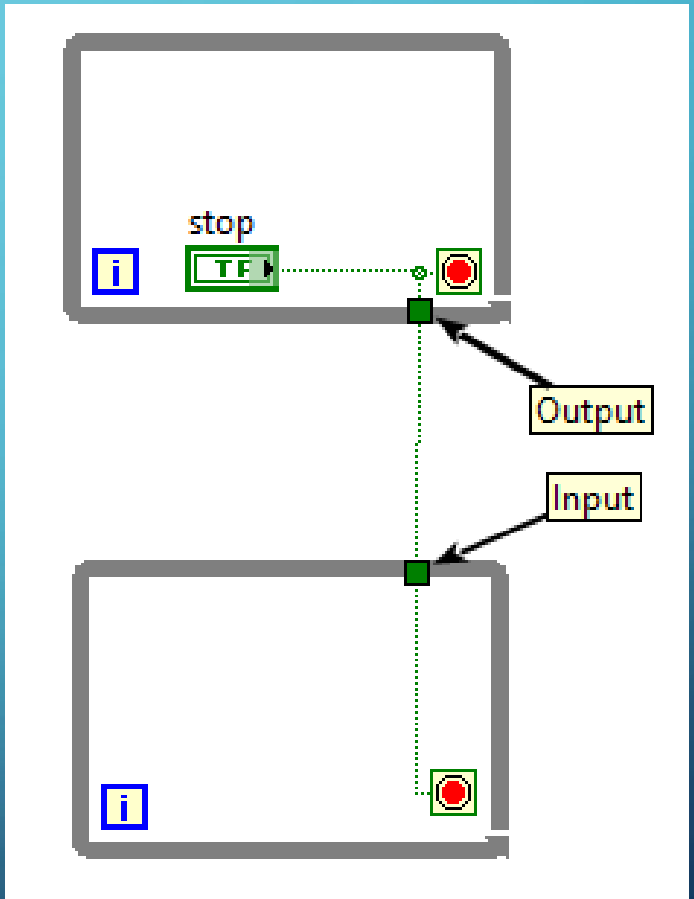
If I wanted to have one stop button stop both while loops, then would this work?



TRANSFERRING DATA FROM TWO DIFFERENT WHILE LOOPS

- The bottom while loop would **not** start running until the top while loop finished.
- The top while loop cannot output until after the loop is finished, and the bottom while loop can't start until all the inputs are available and ready.

The answer is that the stop button would stop both while loops, but the two while loops couldn't run simultaneously.



CLAD QUESTION

Can a wire be used to pass data between loops that are intended to run in parallel?

a. Yes
b. No

CLAD QUESTION

Can a wire be used to pass data between loops that are intended to run in parallel?

a. Yes
b. No

PASSING DATA BETWEEN PARALLEL LOOPS USING VARIABLES

One way that we know of right now to pass data between parallel loops is using a local variable or using property node (value change). This works, but there could be **potential problems** with using them known as race conditions. The problem is that if both loops write to the same variable at almost the same time, the loop that writes to the variable first would get overwritten by the 2nd loop when it tries to write to that same variable causing the first loops information to likely get missed.

Queues are the solution to race conditions.

QUEUES

Queues are a way to store data and to transfer data between different sections of code. Most likely, queues are used to transfer data from one while loop to another while loop. This is known as the producer/consumer scheme (one while loop produces the data while the other while loop reads or consumes the data).

Queues use the first in/first out (FIFO) order for data items, just like waiting in a queue at a bank or somewhere else. The first person (or item) that comes in is the first person (or item) to exit out.

QUEUES (ADVANTAGES & DISADVANTAGES)

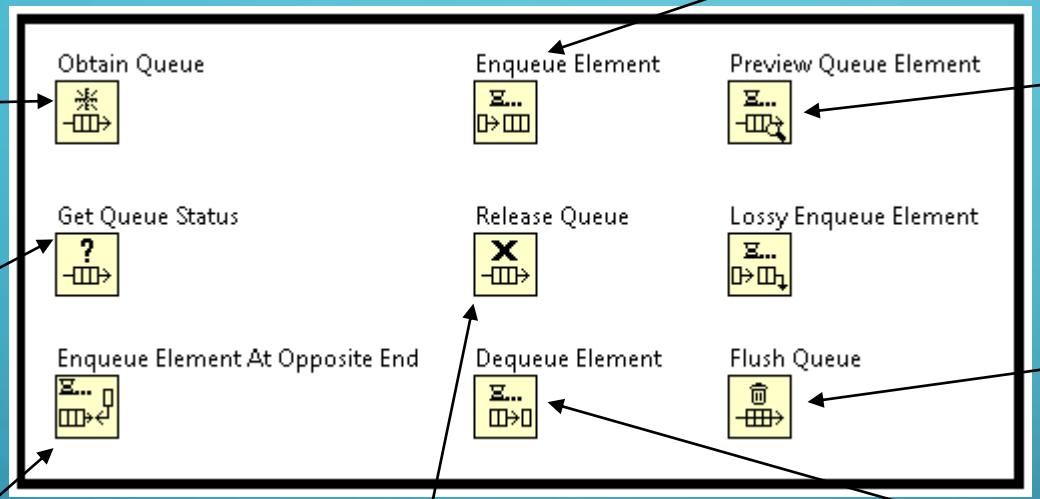
The biggest advantage to using queues is that data remains in some allocation of memory space until the data is read (dequeued). This way, **data is not lost**. Data without using queues (maybe using local variables, etc.) can easily get lost if the producer produces faster than the consumer consumes. Therefore, **always use queues (AKA "FIFO Buffer") when the read and write rates differ**.

One issue is that if the producer is creating lots of data and the consumer is not reading or de-queueing that data fast enough, then your computer's memory can fill up. So you have to be careful that this doesn't happen or your computer could eventually crash.

QUEUES FUNCTIONS

Places an item on the queue

Initializes a queue (creates a reference to a queue)



Reads an item from queue, but doesn't dequeue it.

Tells info. such as elements in the queue (useful)

Flushes out all items from the queue, but different than "Release Queue" because it doesn't close the queue.

Puts an element at the front of a queue

Removes an item from the queue. If no items are on the queue, then it **waits indefinitely** unless a timeout (in ms) is specified.

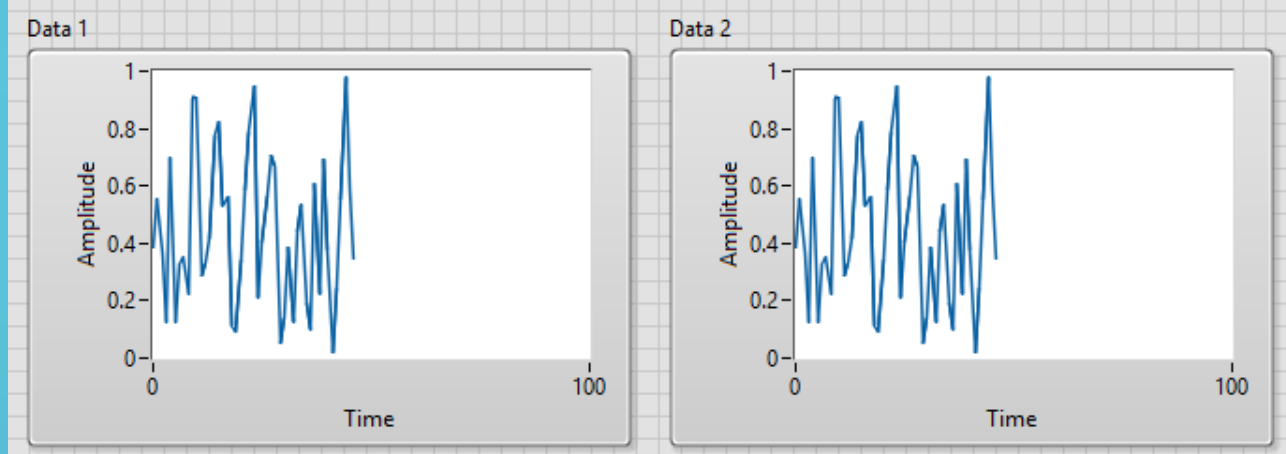
Closes a queue and its references. Outputs all the remaining elements left in the queue.

SIMPLE QUEUE EXAMPLE

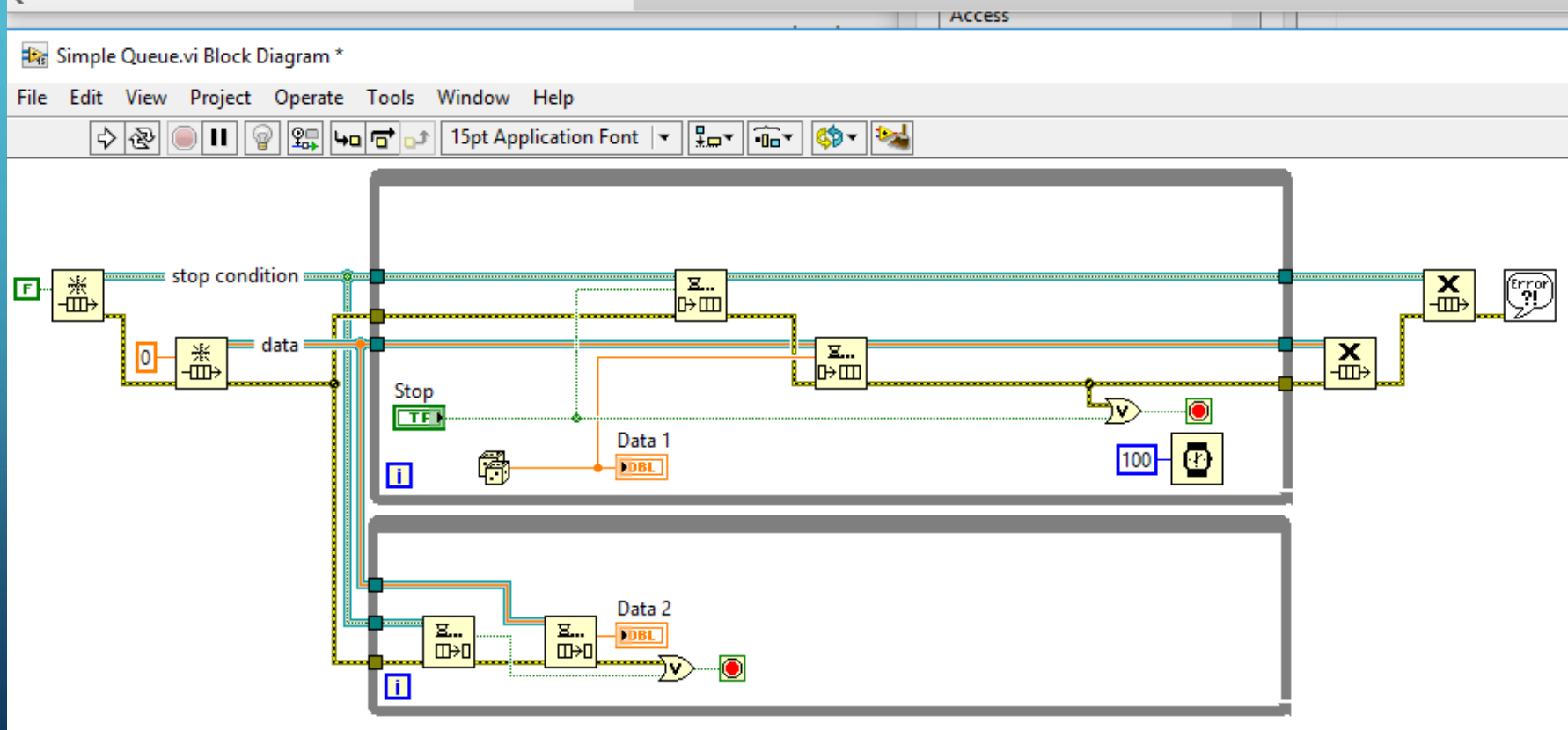
This was taken from LabVIEW examples: “Simple Queue.”

Notice that the top while loop creates data and puts that data in a queue. The bottom while loop dequeues the data and reads it into a chart. Also notice that the producer rate is slower than the consumer rate.

Both charts match. 😊
Data is not lost.



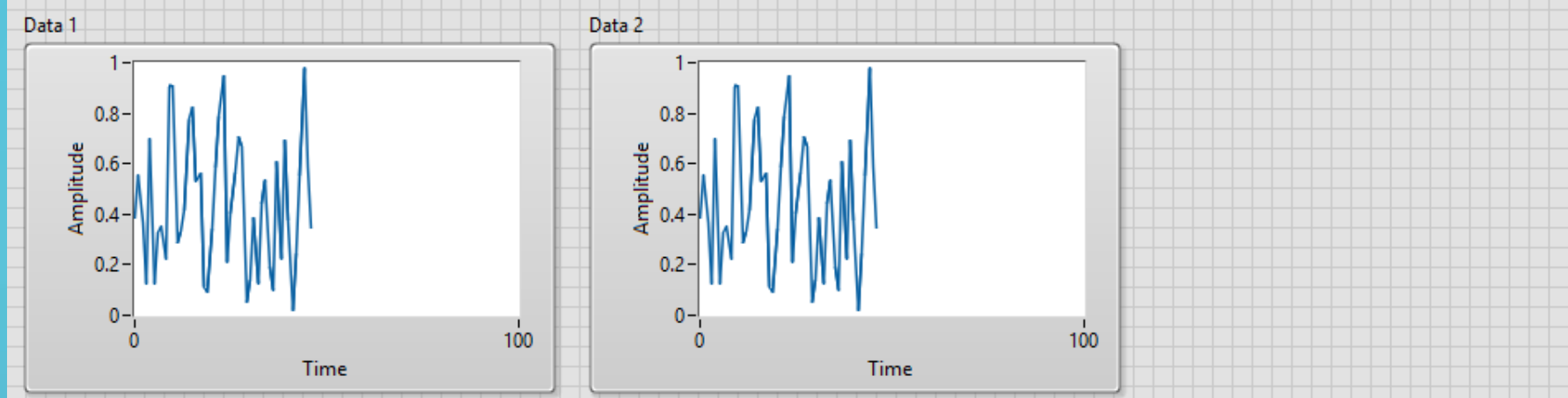
The data generated by Loop 1 on the diagram is passed to Loop 2 through a queue. The stop condition is also monitored in Loop 1 and sent to Loop 2 in a queue.

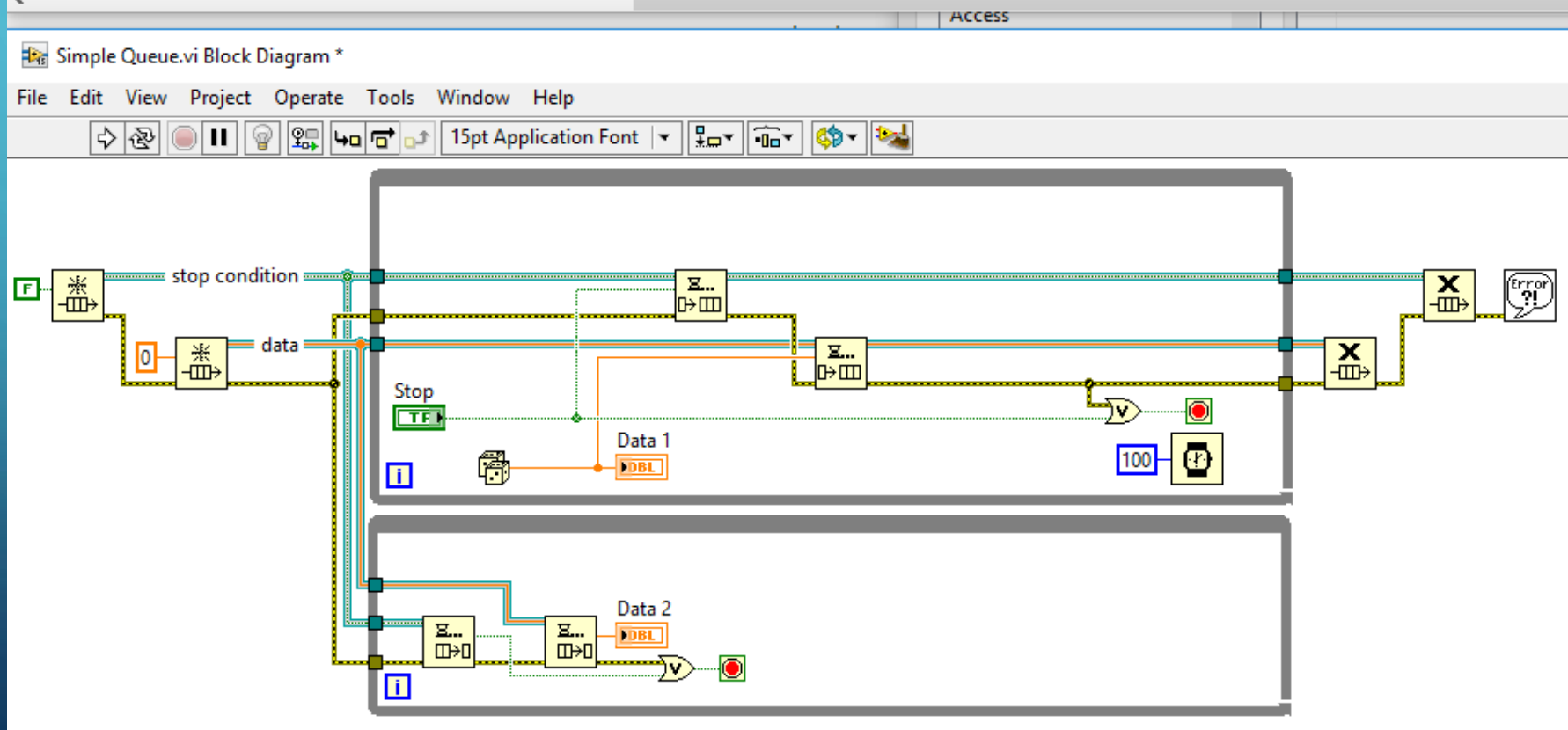


SIMPLE QUEUE EXAMPLE

Also note that even though the bottom while loop doesn't have a Wait (ms) function, it still will run at 100 ms/iteration because it waits for data from the producer to become available. Since the producer produces data every 100 ms, the consumer will consume data at that same rate.

What do you think will happen if I add a 200 ms delay on the 2nd while loop?





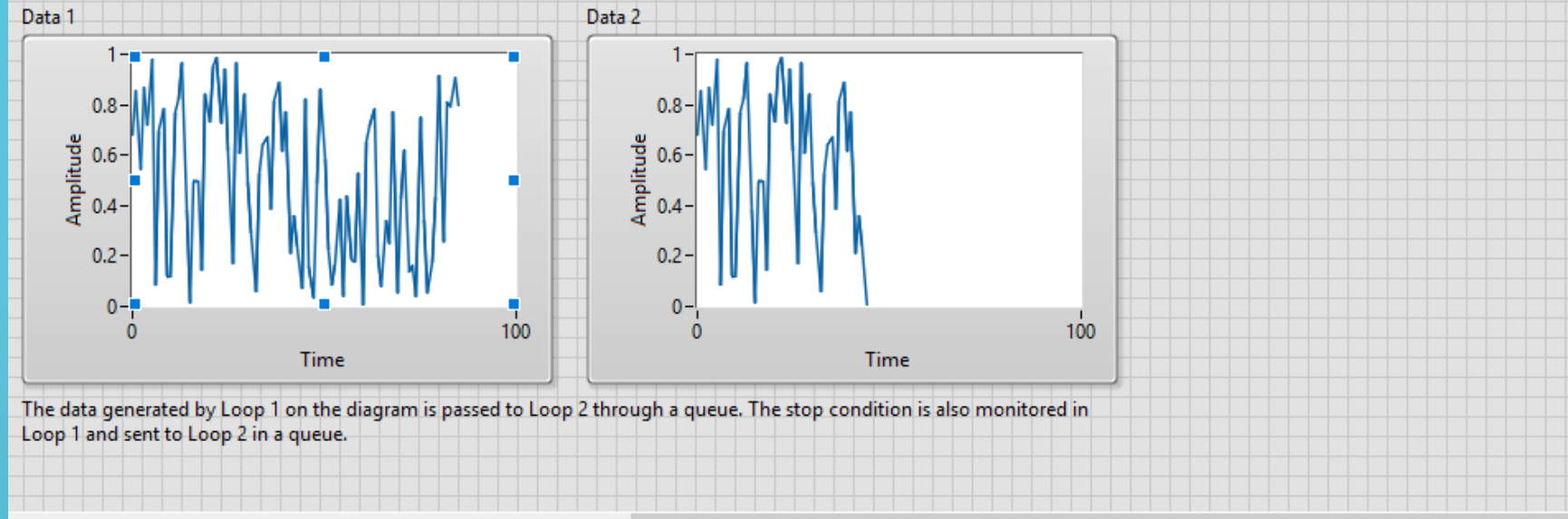
SIMPLE QUEUE EXAMPLE

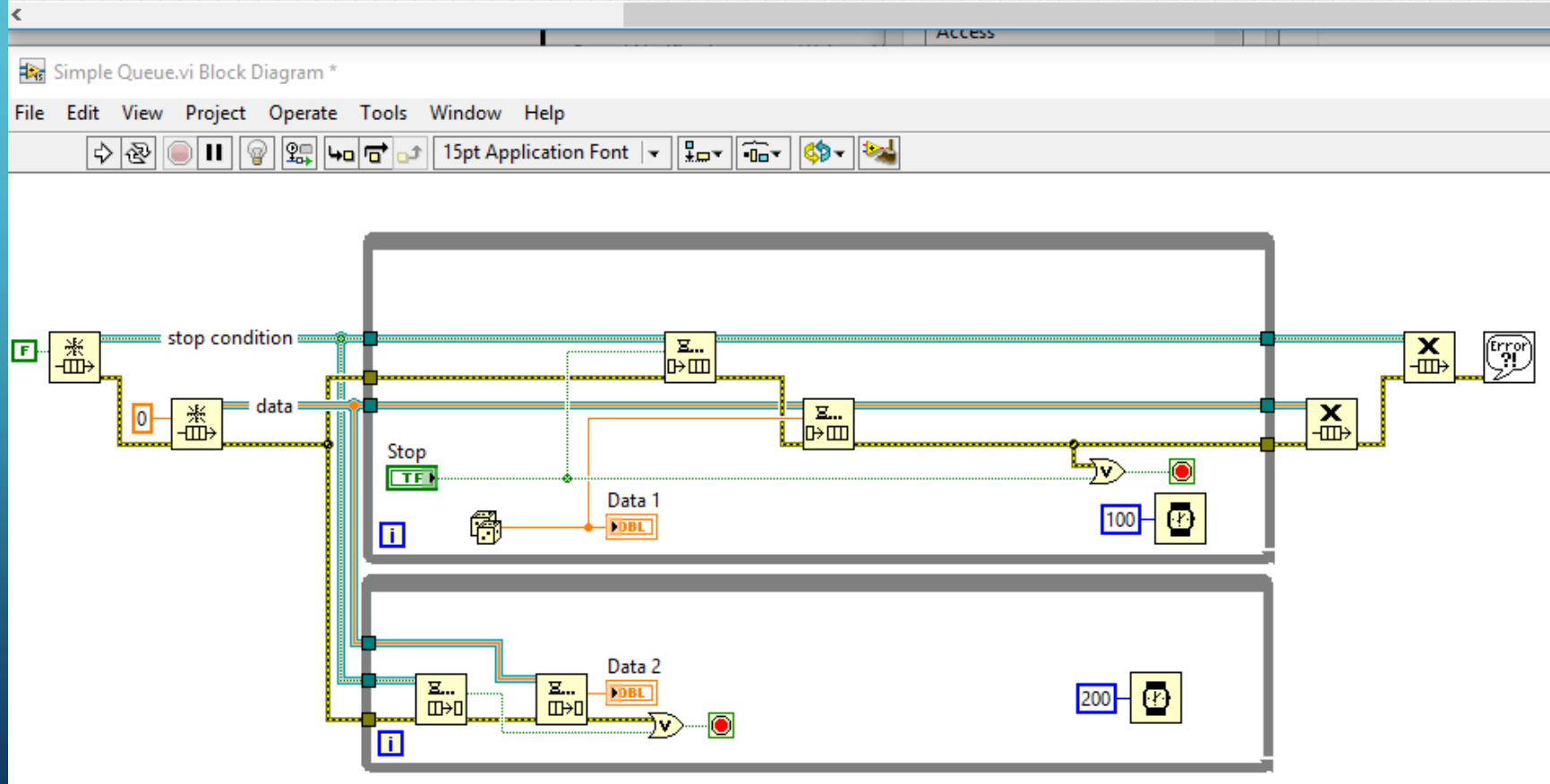
What do you think will happen if I add a 200 ms delay on the 2nd while loop?

Clearly, Data 2's chart is running slower than Data 1's chart. However, the data from the consumer (Data 2) still matches the data from the producer (Data 1).

Data is not lost! 😊

However, the computer's memory will eventually fill up and cause the computer to crash *(unless the consumer rate speeds up)*. ☹





CLAD QUESTION

Q10: How does a producer consumer design pattern process excessive amounts of data?

A Using a single element queue the Producer Consumer processes only current data.
B Using a queue for storage the Consumer loop processes all data when it has time.
C The Producer loop will slow its cycle time to allow the consumer loop to catch up.
D The consumer loop will increase its processing time to match the producer loop.

CLAD QUESTION

Q10: How does a producer consumer design pattern process excessive amounts of data?

A	Using a single element queue the Producer Consumer processes only current data.
B	Using a queue for storage the Consumer loop processes all data when it has time.
C	The Producer loop will slow its cycle time to allow the consumer loop to catch up.
D	The consumer loop will increase its processing time to match the producer loop.

CLAD QUESTION

Q21: What does this function do?

Release Queue

A Transfers the queue reference to other callers

B Sorts all queue elements in order and returns them as an array

C Clears all elements from the queue

D Removes one reference to the queue and returns any remaining elements

CLAD QUESTION

Q21: What does this function do?

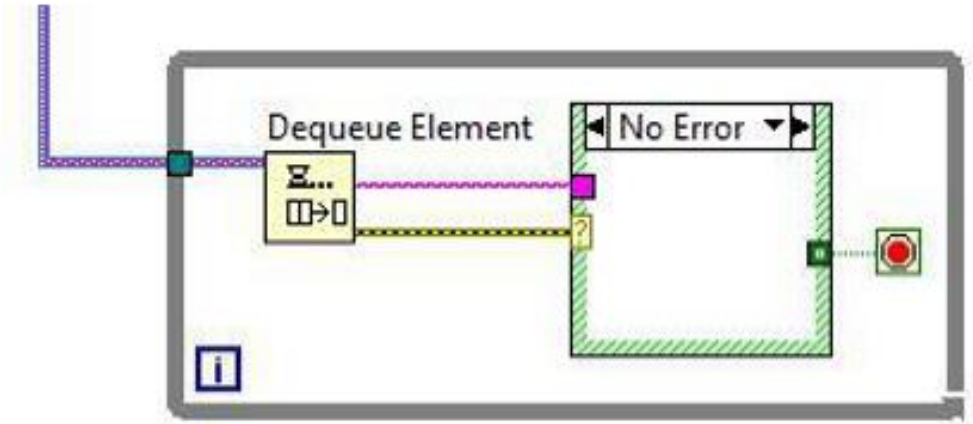
Release Queue

A	Transfers the queue reference to other callers
B	Sorts all queue elements in order and returns them as an array
C	Clears all elements from the queue
D	Removes one reference to the queue and returns any remaining elements

Returned elements are returned in the form of an array.

CLAD QUESTION

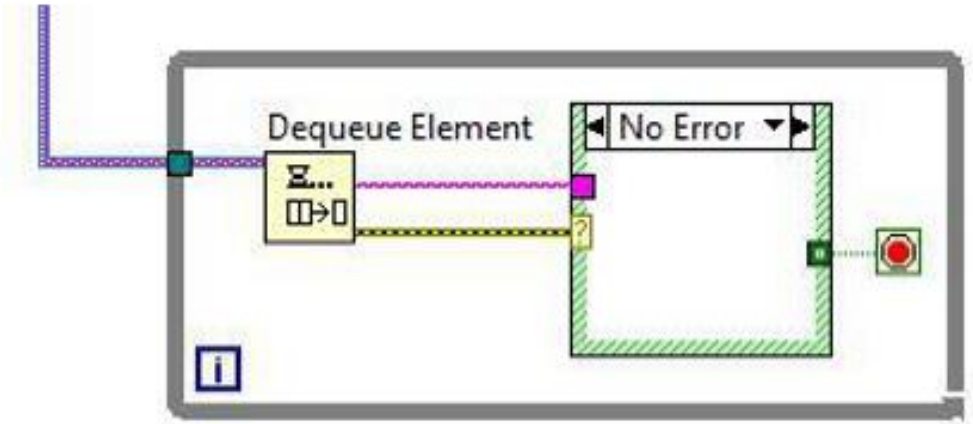
How long does this Dequeue Element function wait to receive data?



a. 1 millisecond (default since unwired)
b. 1 second (default since unwired)
c. Indefinitely
d. It does not wait, it returns immediately

CLAD QUESTION

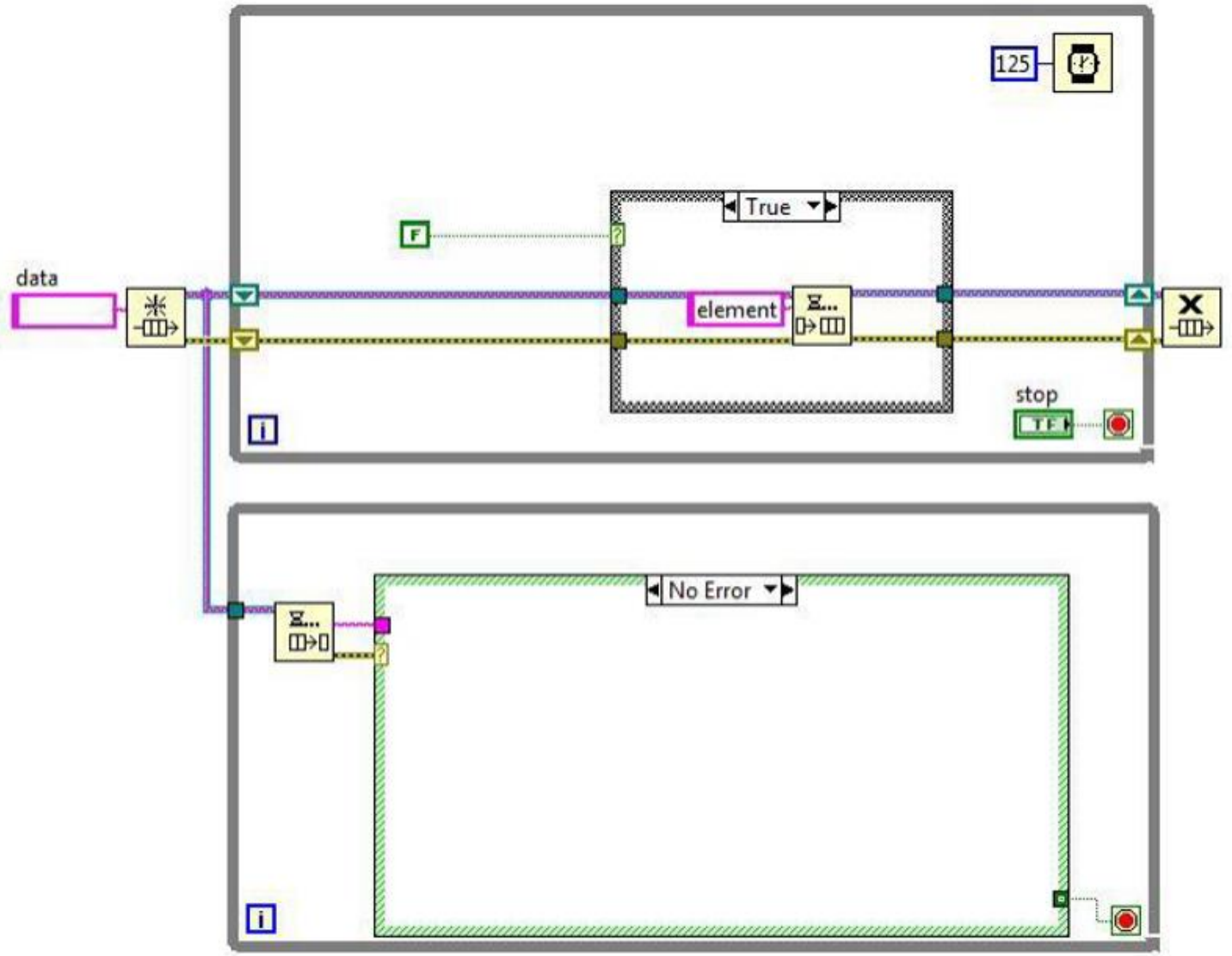
How long does this Dequeue Element function wait to receive data?



a. 1 millisecond (default since unwired)
b. 1 second (default since unwired)
c. Indefinitely
d. It does not wait, it returns immediately

CLAD QUESTION

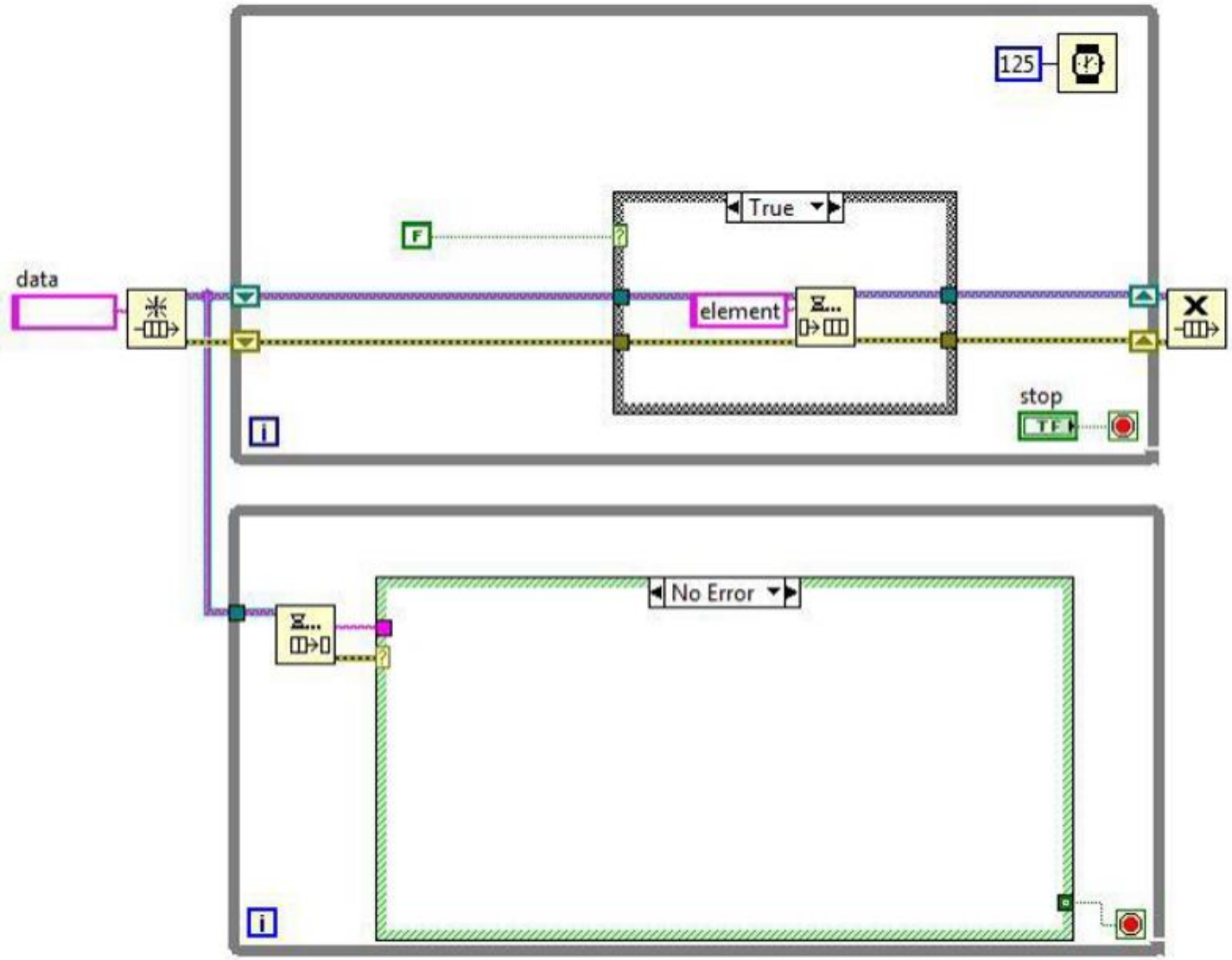
Q19: What is the name of this common design pattern?



A State Machine
B Producer Consumer (data)
C Producer Consumer (events)
D Queued Message Handler

CLAD QUESTION

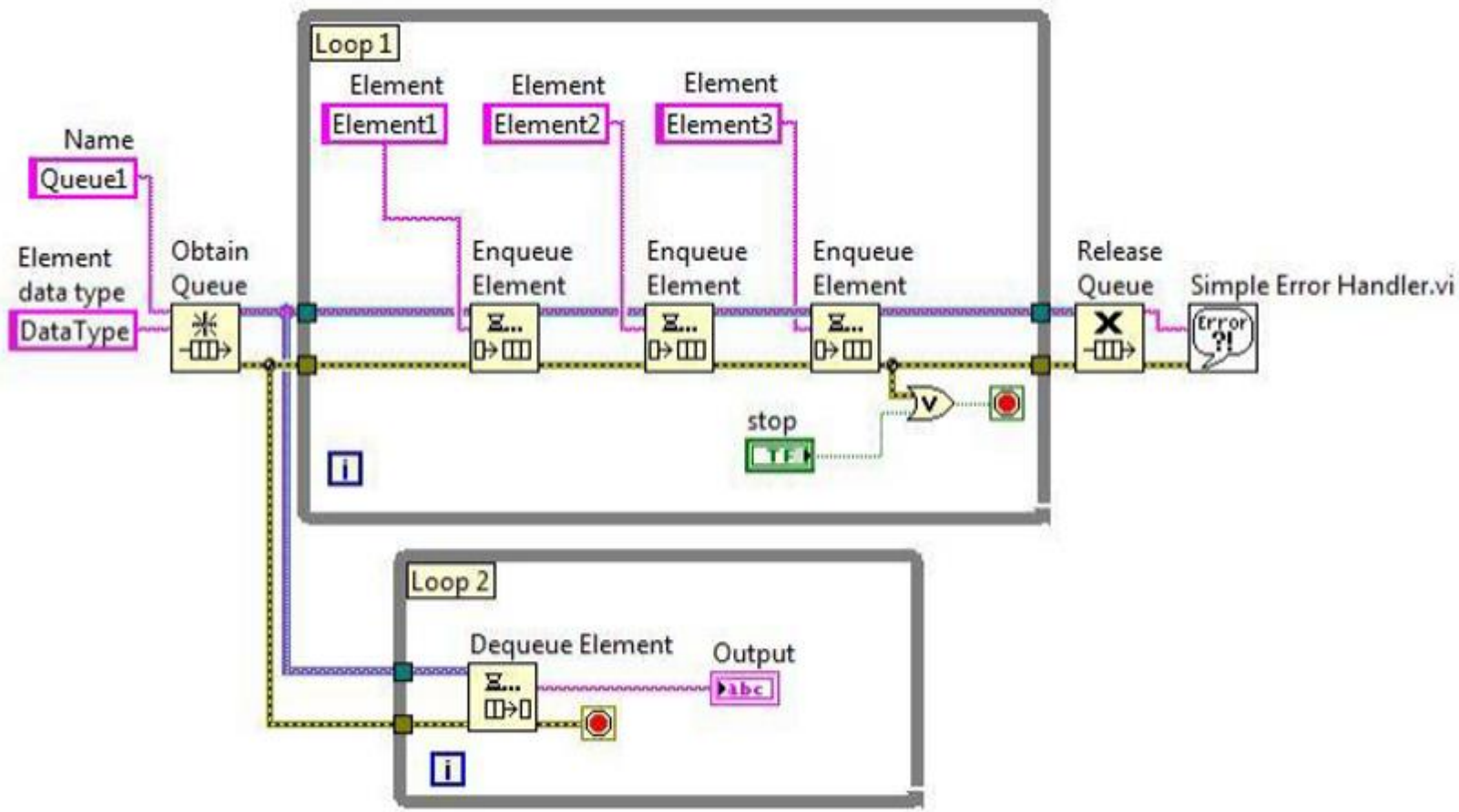
Q19: What is the name of this common design pattern?



A	State Machine
B	Producer Consumer (data)
C	Producer Consumer (events)
D	Queued Message Handler

CLAD QUESTION

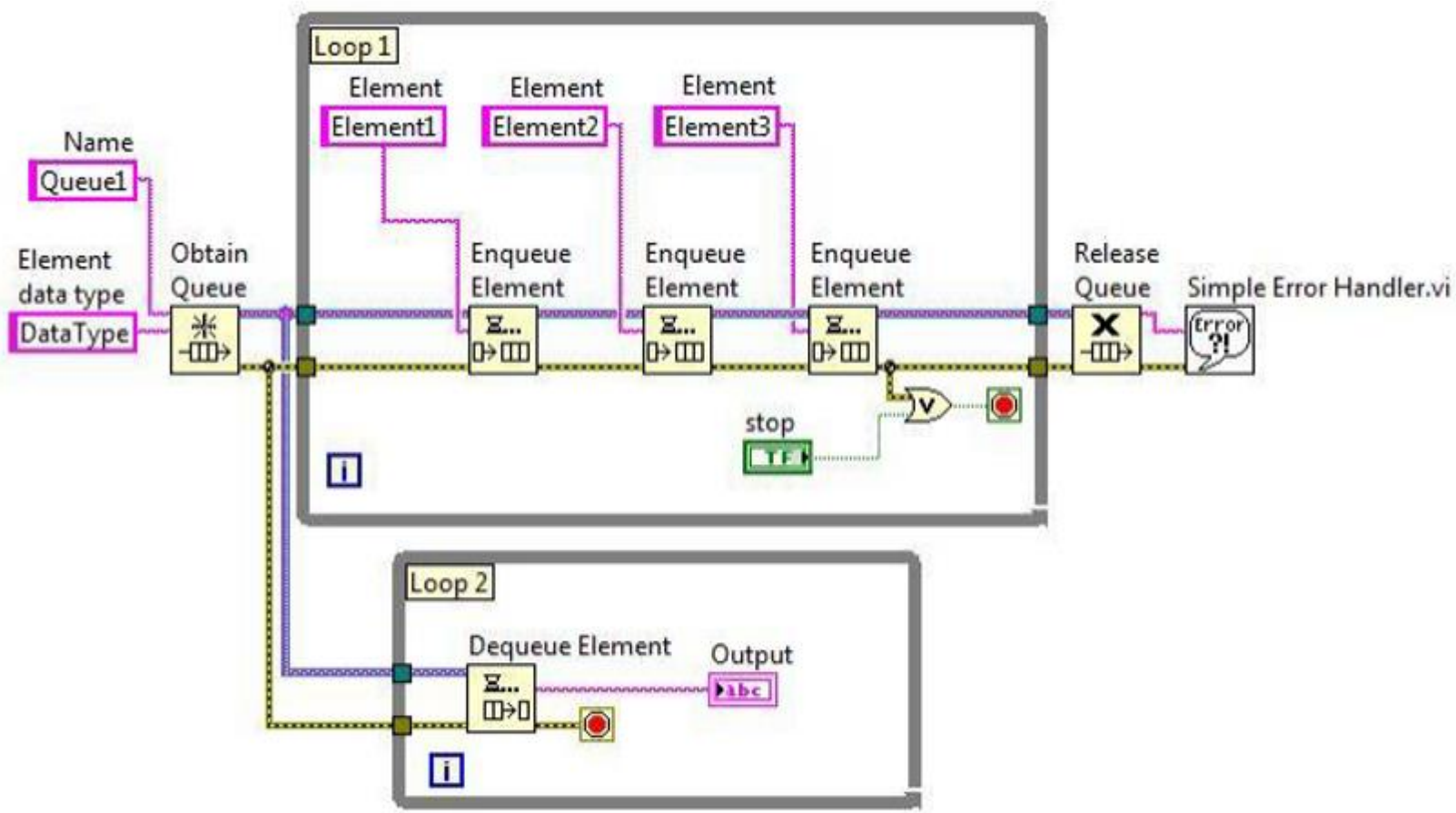
Q21: What value is displayed in the **Output** indicator when the fifth iteration (i=4) of Loop 2 completes?



A	Element1
B	Element2
C	Element3
D	Element1Element2Element3

CLAD QUESTION

Q21: What value is displayed in the **Output** indicator when the fifth iteration (i=4) of Loop 2 completes?



A	Element1
B	Element2
C	Element3
D	Element1Element2Element3

Top while loop enqueues element 1,2,3,1,2,3, and so on.

Bottom while loop dequeues 1,2,3,1,2

NOTIFIERS

Notifiers are another way to transfer data from one While loop to another While loop or transfer data between two VIs running on the same machine (doesn't work through a network).

Notifiers are similar to Queues, but instead of data being added into memory in the queue, data is overwritten. This leads to a problem that some data could get missed, especially if the reading rate is slower than the writing rate. However, this prevents the problem of the computer's memory filling up since the same memory location is written over each time.

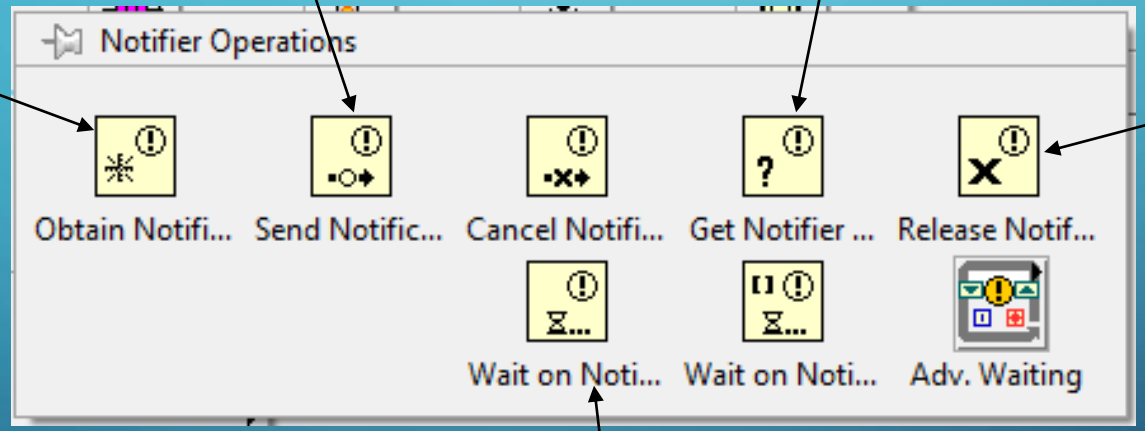
NOTIFIER FUNCTIONS

Creates a Notifier and returns a reference to that Notifier.

Updates the value of the Notifier

Reads the value of the Notifier, but doesn't wait for the Notifier to have a new value. It also provides the status of the Notifier.

Closes the Notifier and releases the reference.



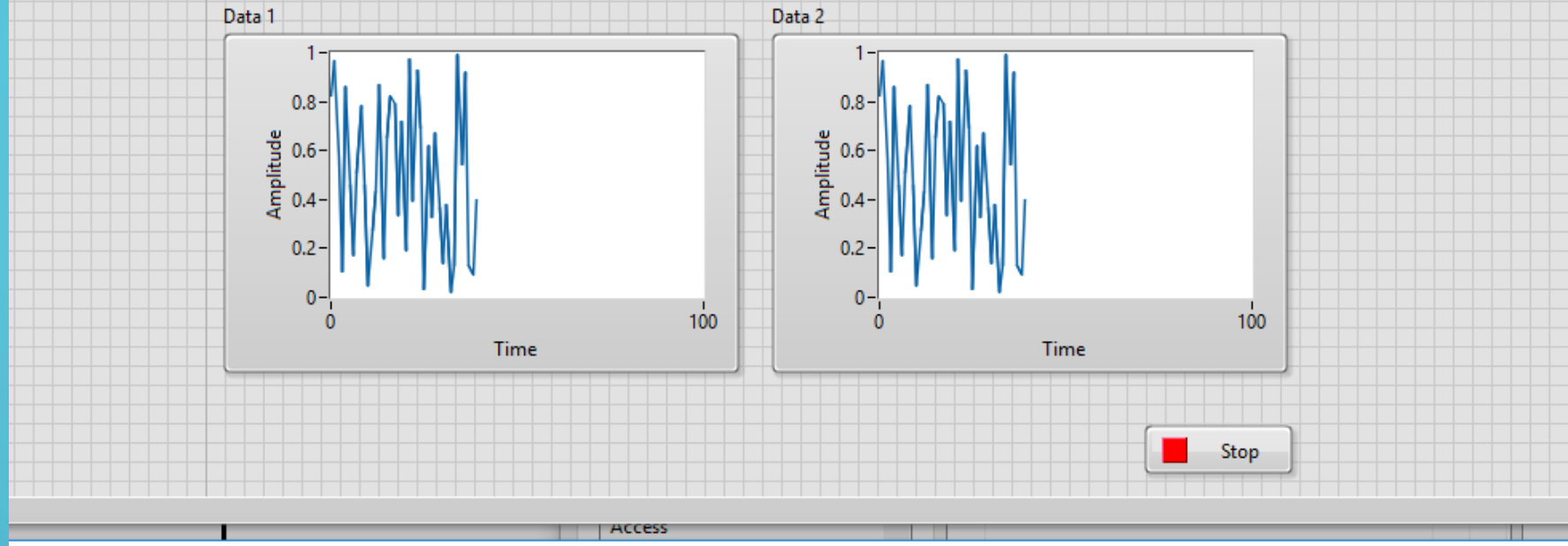
Waits for a Notifier indefinitely unless a timeout (in ms) is specified.

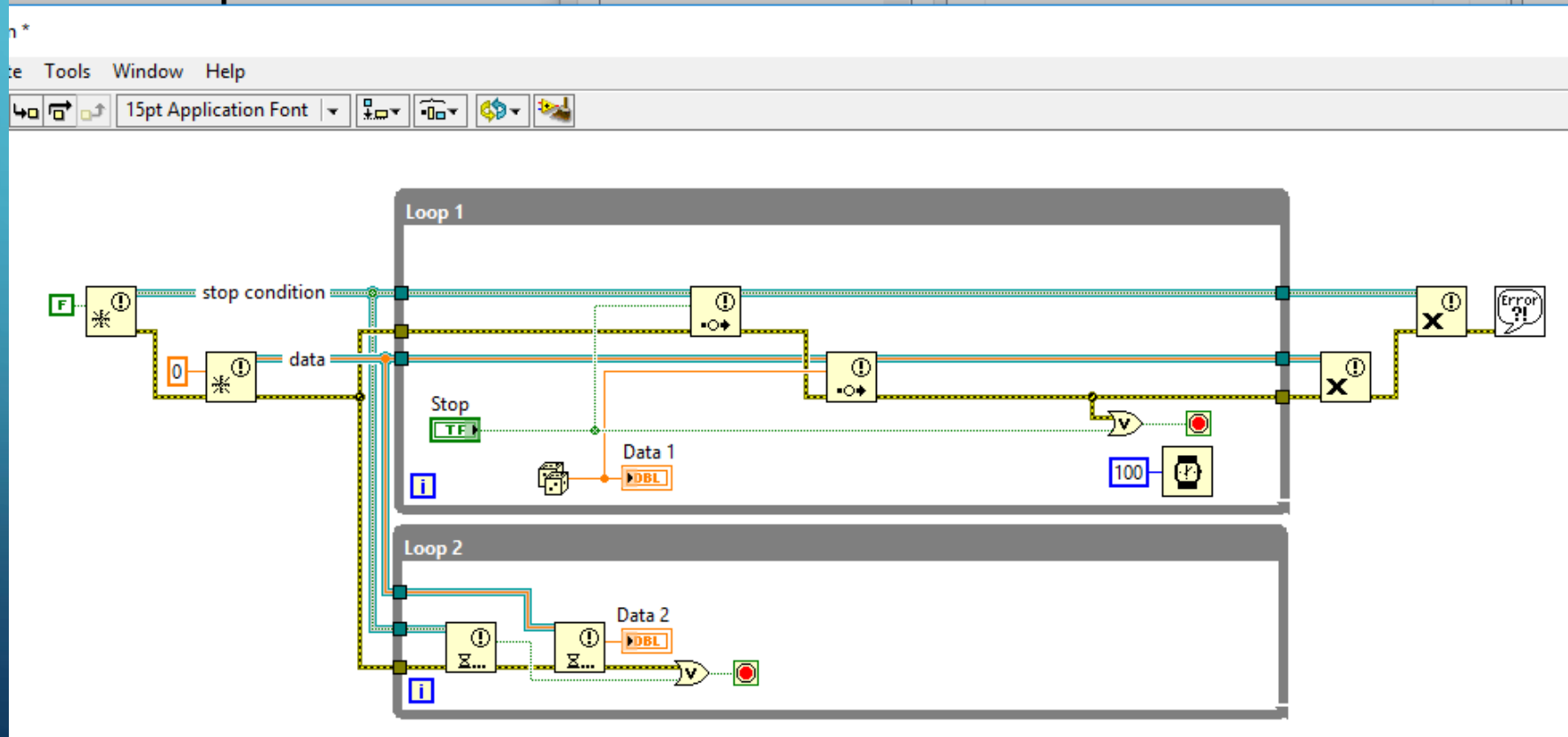
SIMPLE NOTIFIER LABVIEW EXAMPLE

This program works almost identical to the Simple Queue.VI previous example.

Notice that the data is the same on both charts.

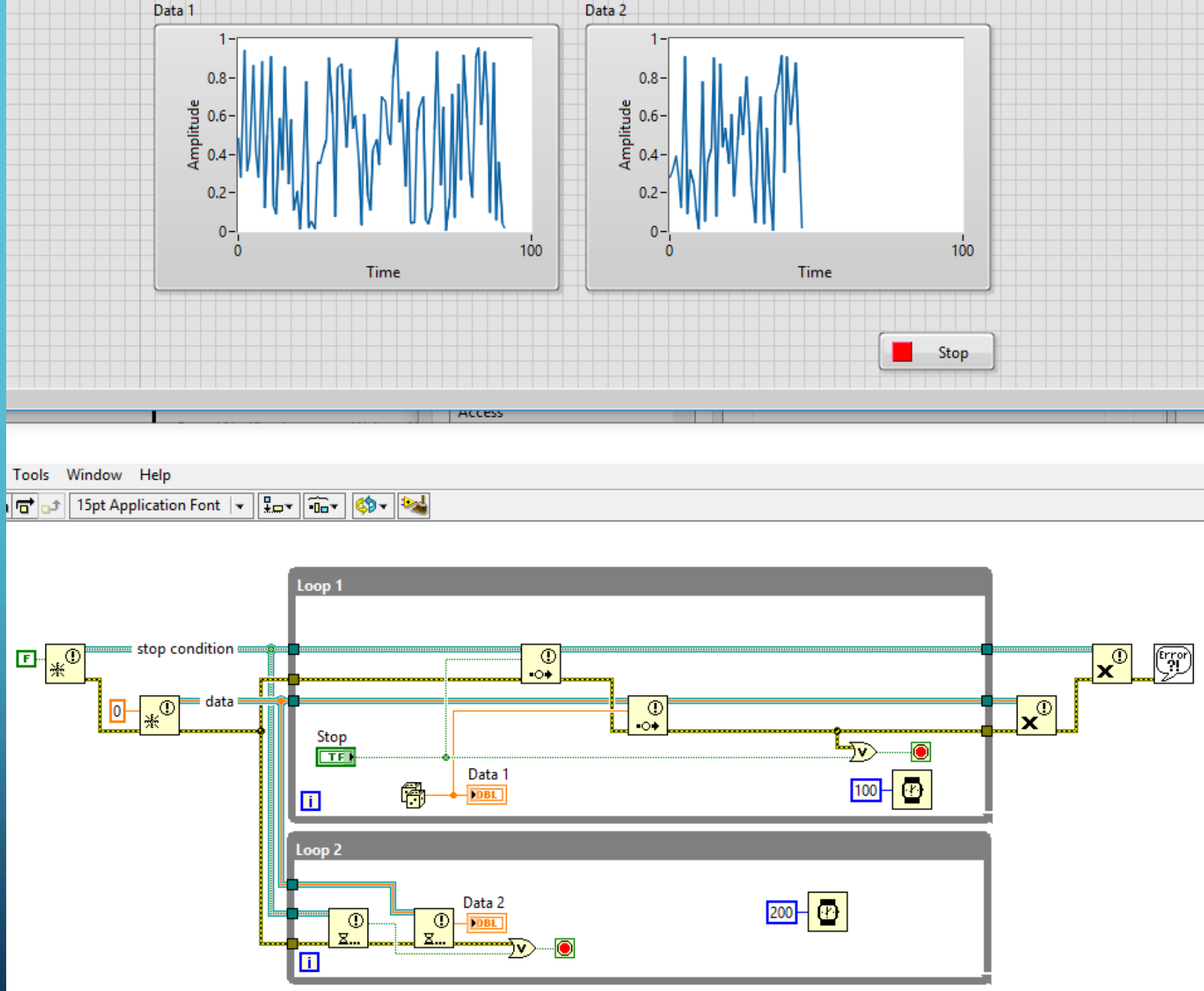
What do you think will happen if I make the bottom While loop run slower than the top While loop?





SIMPLE NOTIFIER LABVIEW EXAMPLE

With a 200 ms delay added to the 2nd while loop, the **data from both charts are not equal**. ☹



QUEUES VS. NOTIFIERS

Queue Advantage: Queues do not lose data, while notifiers could.

Notifier Advantage: Notifiers do not consume much of the computer's memory, while queues could.

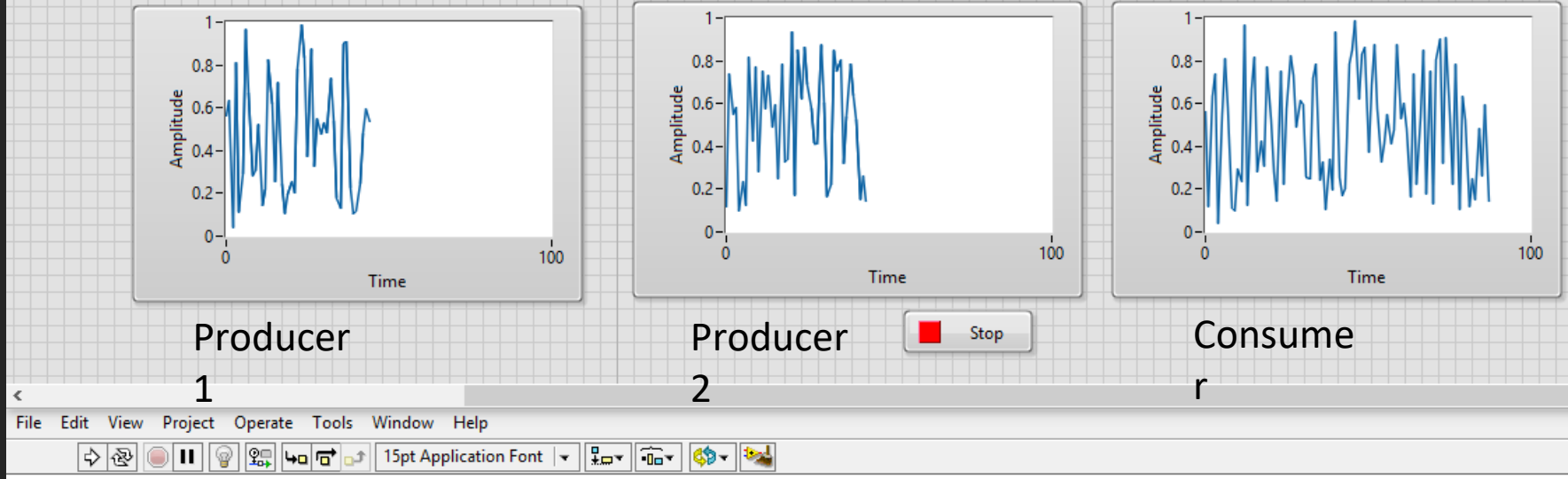
One other difference between the two is that **notifiers are typically used to have one-to-many communication (one writer, many readers) and queues are typically used to have many-to-one communication (many writers, one reader).**

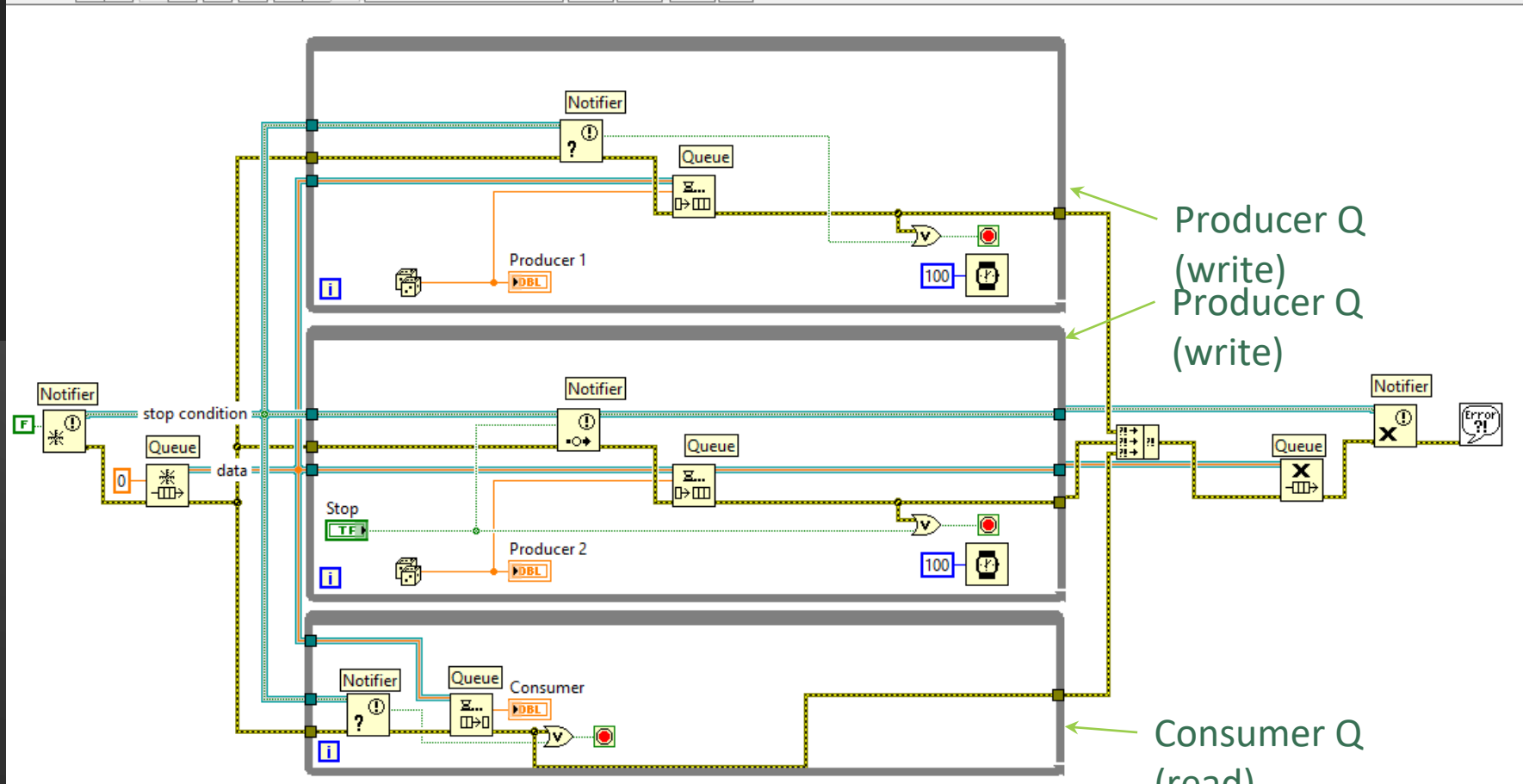
In this example, I use both Queues and Notifiers.

Concerning queues . . .

For writing the data, the top two while loops are producers and both writing random numbers. If you interweave the data from Producer 1 with the data from Producer 2, then you get the 3rd chart (consumer chart).

Therefore, you can have the many-to-one design pattern.

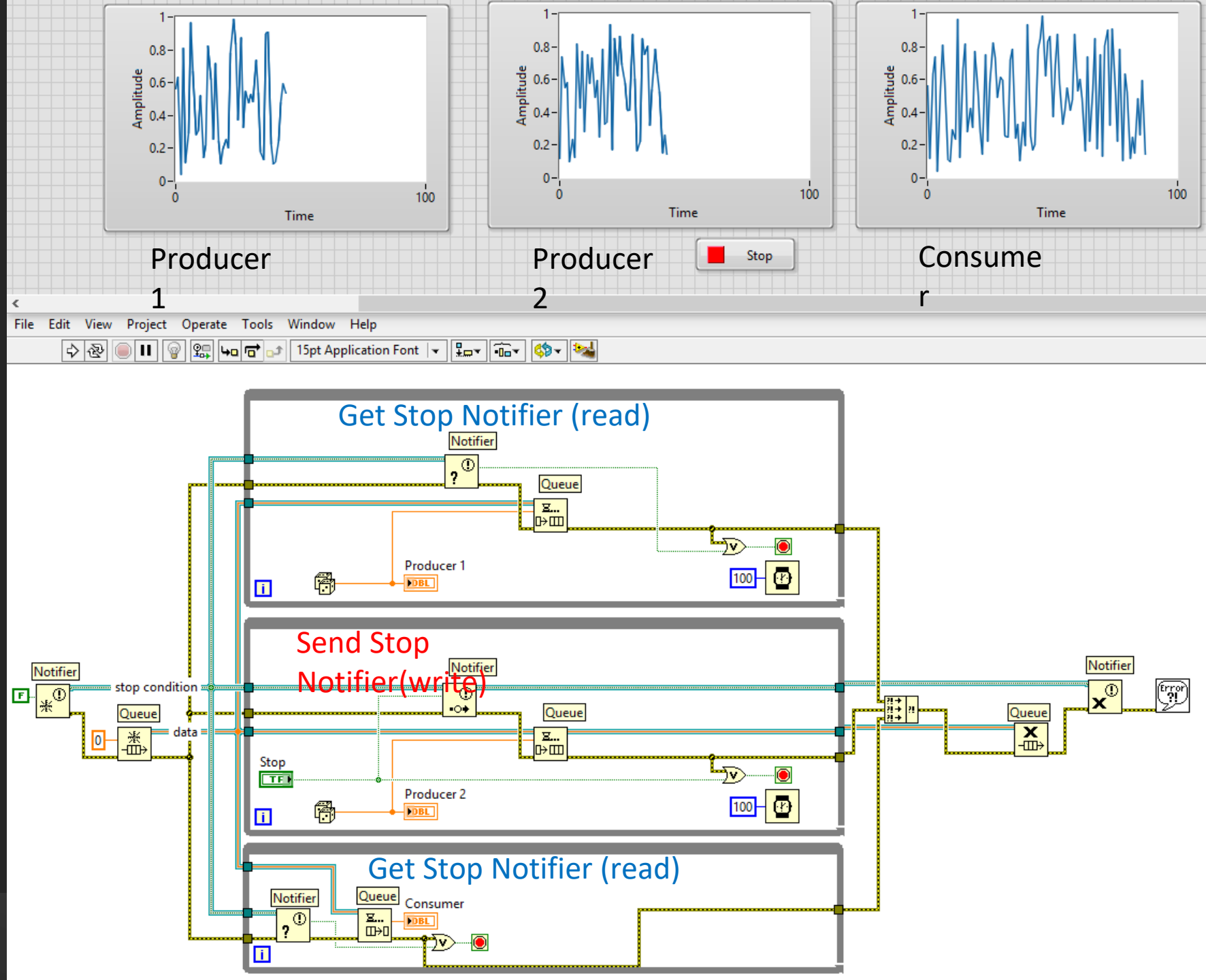




Concerning Notifiers . . .

I also wanted one stop button to stop all 3 while loops. I didn't want to use a queue here because when reading the queue, the data gets taken off of the queue so one of the while loops would miss the Stop condition. I chose to use **notifiers** instead since this data doesn't get taken out of memory once it is read.

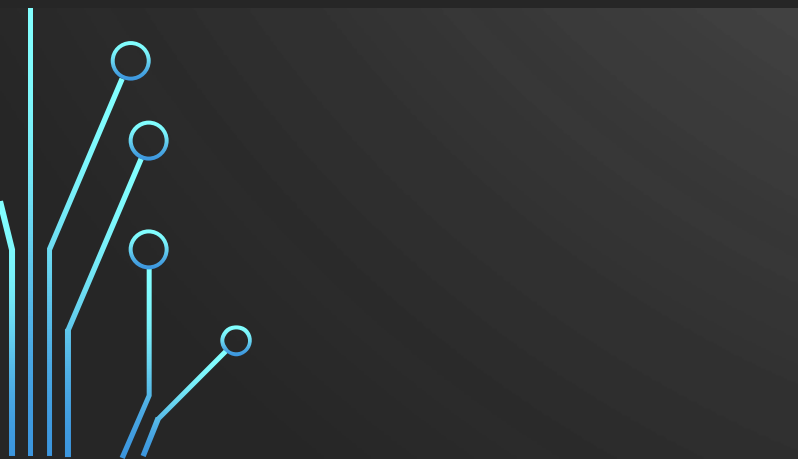
This program works because once the stop button is pressed, the middle while loop stops and since nothing else is sending notifiers, the notifier stays set to True. The other while loops can read the notifier when they get around to it to stop their individual while loops.

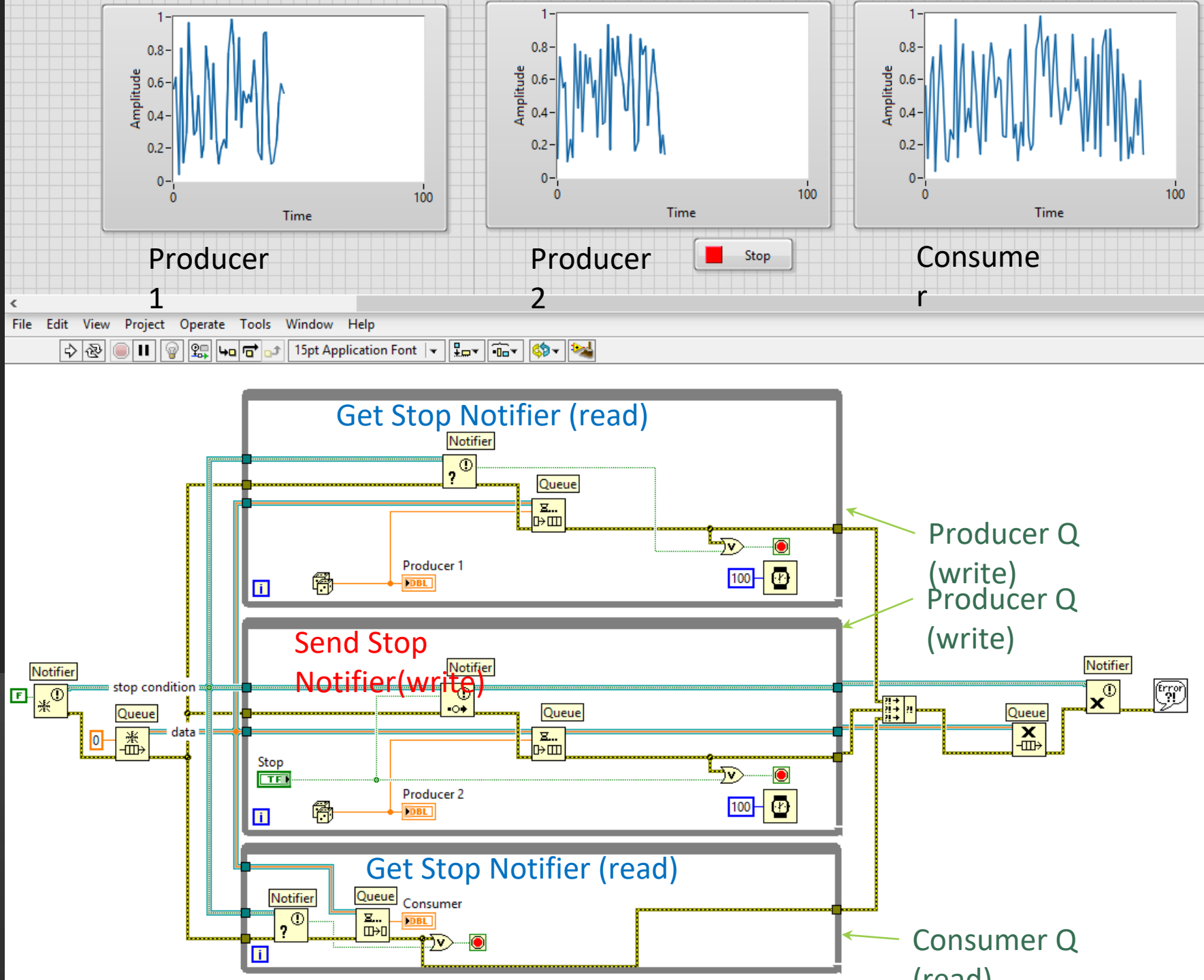


To recap . . .

Queues are used for many-to-one situations (many talkers, one listener) because the program doesn't lose data since data is added onto the computer's memory until it's read. Once it's read, it's taken out of memory.

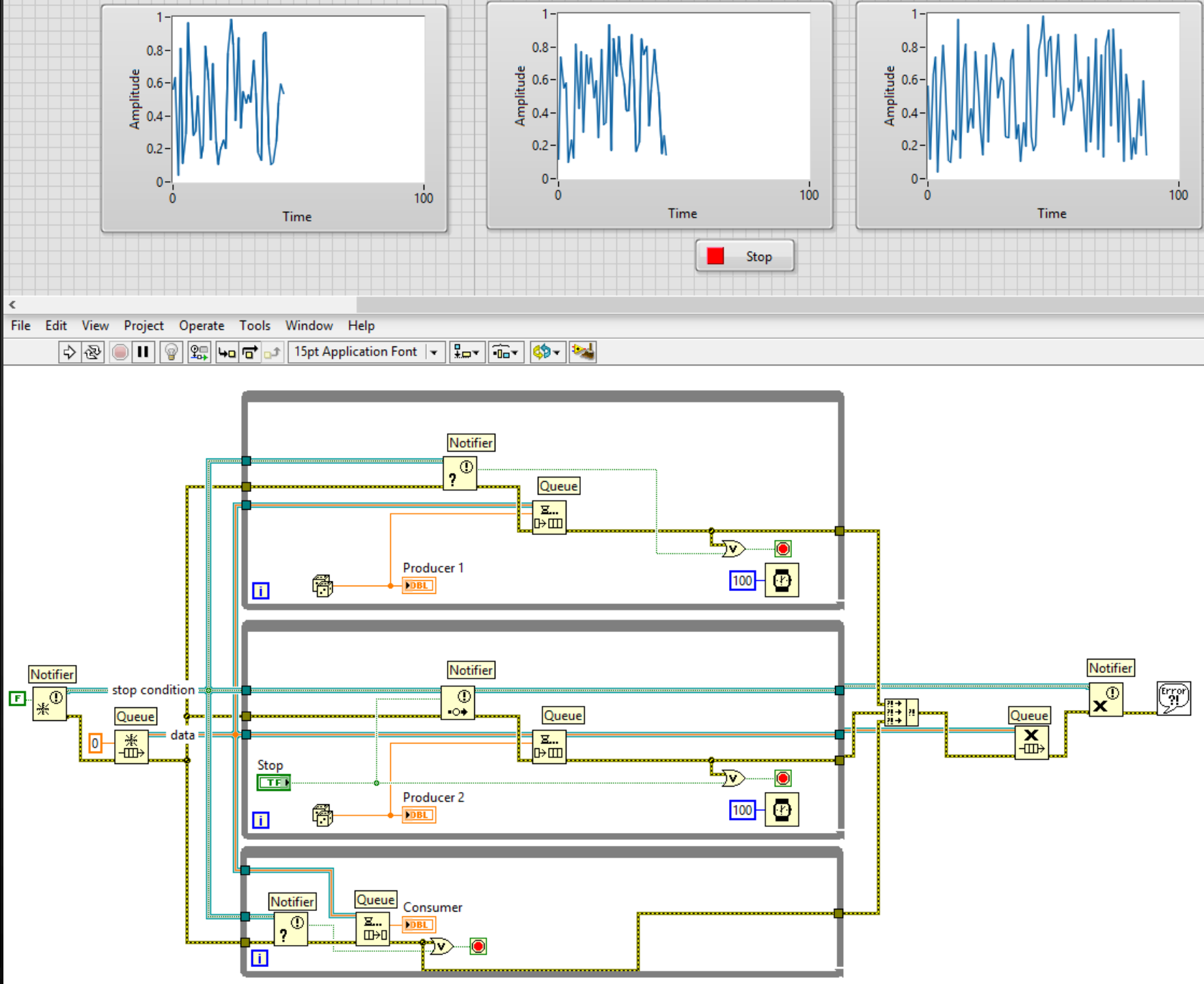
Notifiers are used with one-to-many situations (one talker, many listeners) because the data is left available in the computer's memory until it is written over.





Side Note: If I wanted to only use Queues and no Notifiers, then I could try to use the *Preview Queue Element* function (not shown here) because this reads the queue but doesn't take the element off of the queue. **This would work if you knew the order** at which the 2 while loops are going to read the queue. Ideally, you would make the 1st while loop *Preview Queue Element* and the 2nd while loop *Dequeue* the element. If this order happened to be opposite with *Dequeue* first, then the *Preview Queue Element* being second wouldn't work since the item would have been taken off the queue. And there's no way of knowing which while loop is going to read first.

Therefore, the **best method is to use Notifiers** for one-to-many communication.



CLAD QUESTION

Q19: Which of the following cannot be used to transfer data between two parallel loops?

A	Wires
B	Queues
C	Notifiers
D	Local variables

CLAD QUESTION

Q19: Which of the following cannot be used to transfer data between two parallel loops?

A	Wires
B	Queues
C	Notifiers
D	Local variables

CLAD QUESTION

Which data synchronization mechanism ensures that no data is lost when an application temporarily provides data faster than it is able to process it?

a. Notifier
b. Queue
c. Semaphore
d. Local Variable

CLAD QUESTION

Which data synchronization mechanism ensures that no data is lost when an application temporarily provides data faster than it is able to process it?

a. Notifier
b. Queue
c. Semaphore
d. Local Variable

GLOBAL VARIABLES

Global variables are used when you want to transfer data between two different VIs running on the same machine. One popular use is to have one stop button that stops two different VIs running on one machine at the same time.

An issue with Global Variables is again race conditions. If one VI is writing to the variable at the same time that another VI is writing to that same variable, then you likely will not know what the value of the variable is set to and this could cause problems.

CLAD QUESTION

Which of the following illustrates an advantage of a global variable over a local variable?

a. A global variable can pass data between two independent VIs running simultaneously
b. Only the global variable can pass array data, local variables cannot
c. Global variables follow the dataflow model, and therefore cannot cause race conditions
d. Global variables do not require owned labels to operate

CLAD QUESTION

Which of the following illustrates an advantage of a global variable over a local variable?

a. A global variable can pass data between two independent VIs running simultaneously
b. Only the global variable can pass array data, local variables cannot
c. Global variables follow the dataflow model, and therefore cannot cause race conditions
d. Global variables do not require owned labels to operate

FUNCTIONAL GLOBAL VARIABLE

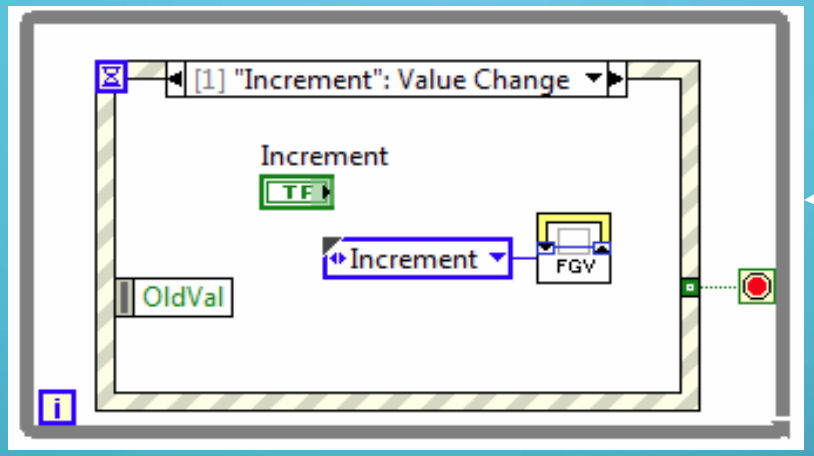
One method that avoids race conditions is to use what is known as a Functional Global Variable (FGV). This works the same as a Global Variable, but prevents race conditions.

Functional Global Variables are variables that use a separate VI (similar to a Type Definition). It is its own subVI.

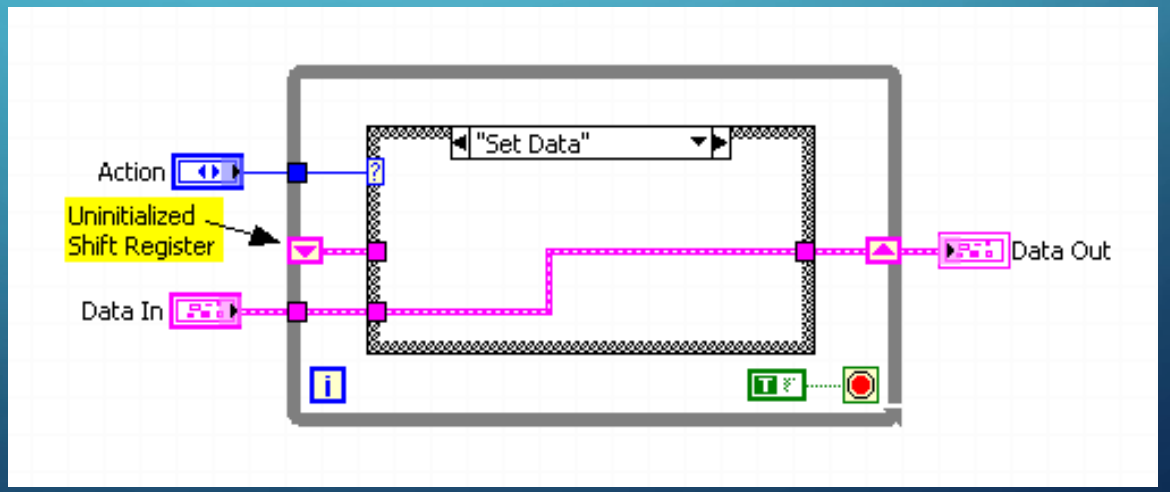
Two VIs cannot call the FGV at the same time, so this prevents race conditions.

FUNCTIONAL GLOBAL VARIABLE EXAMPLE

Notice that the shift register is uninitialized. We originally learned that you should almost always initialize shift registers because shift registers keep the data in memory and if you run a program twice, then it would use the old value from the previous run.



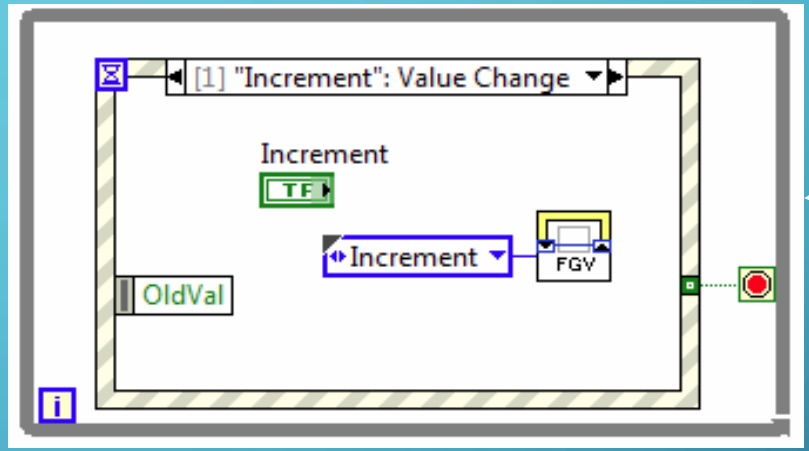
How a FGV is called.



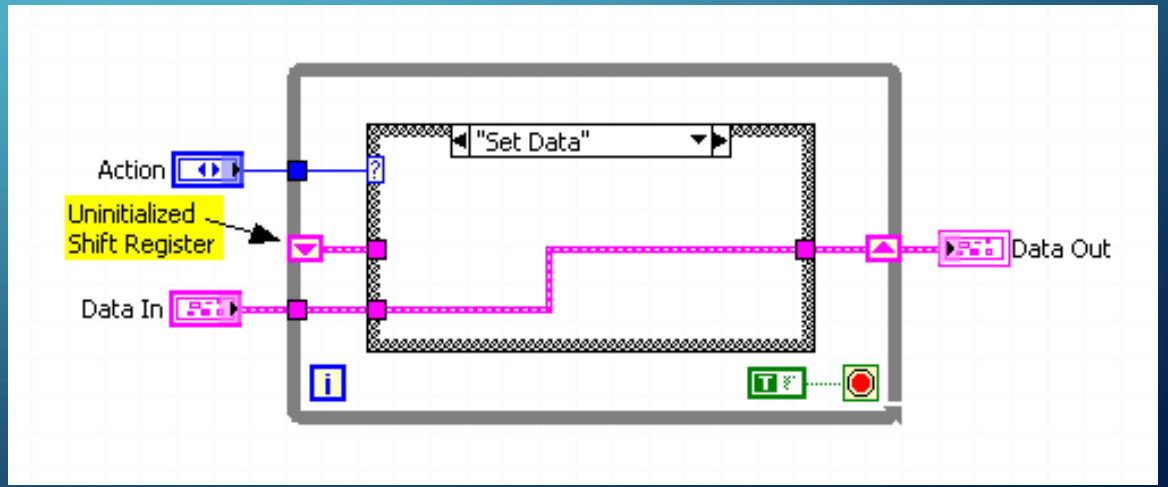
FUNCTIONAL GLOBAL VARIABLE EXAMPLE

However, we want the old value to remain in memory when using a variable, so we use this uninitialized shift register to our advantage.

Notice that the FGV also has an enumerated constant associated with it to tell the variable what action to perform. TRUE is always wired to the stop conditional so the while loop only runs one time. The only reason to have the while loop is so that we can have a shift register.

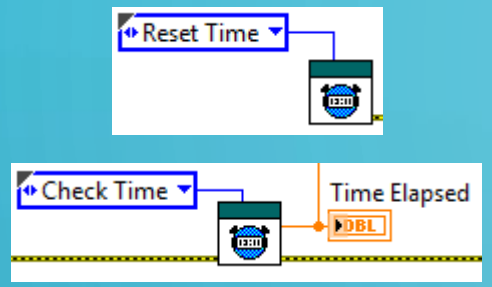


How a FGV is called.

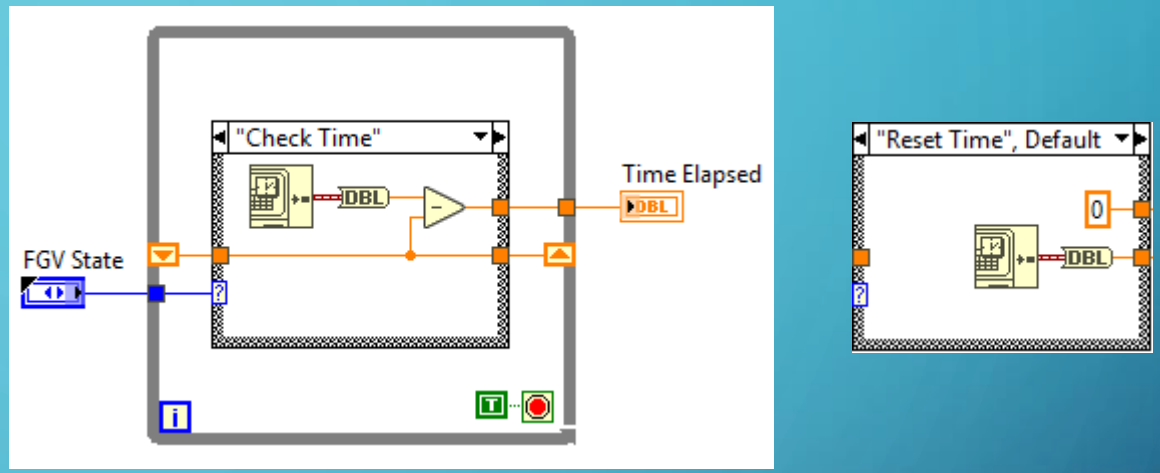


FUNCTIONAL GLOBAL VARIABLE (FGV) EXAMPLE

Calling the FGV



Code for the FGV



This is a great example of using a Functional Global Variable (FGV). Suppose you had multiple applications running on one machine: a main application and two secondary applications. If the secondary applications wanted to know how long the main application was running, then they could tap into this FGV to determine this length of time.

Notice that within the FGV, you can either reset or check the time. If you reset the time, then it just initializes the shift register with the current time. If you check the time, then it subtracts the initialized time from the current time.

CLAD QUESTION

You need to programmatically update the value in a numeric control. Which is the most appropriate strategy?

a. Use a Functional Global Variable
b. Use a Local variable
c. Set the desired value as the default value
d. Use a Data Value Reference

CLAD QUESTION

You need to programmatically update the value in a numeric control. Which is the most appropriate strategy?

a. Use a Functional Global Variable
b. Use a Local variable
c. Set the desired value as the default value
d. Use a Data Value Reference

CLAD QUESTION

Which variable is commonly used to eliminate race conditions by preventing simultaneous access to code or data?

a. Functional global variable
b. Local variable
c. Global variable
d. Shared variable

CLAD QUESTION

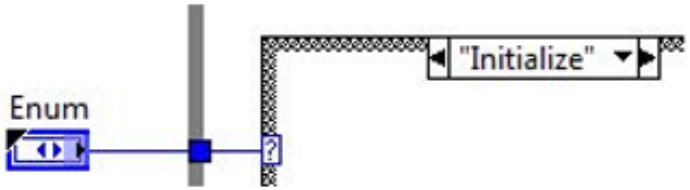
Which variable is commonly used to eliminate race conditions by preventing simultaneous access to code or data?

a. Functional global variable
b. Local variable
c. Global variable
d. Shared variable

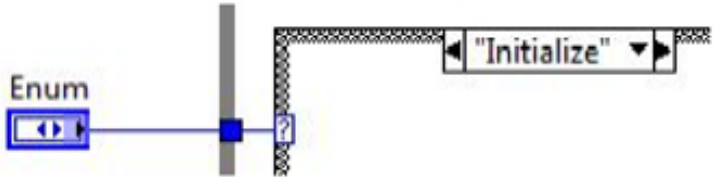
CLAD QUESTION

Q17: What is the best choice for the case selector input for a Functional Global Variable design pattern?

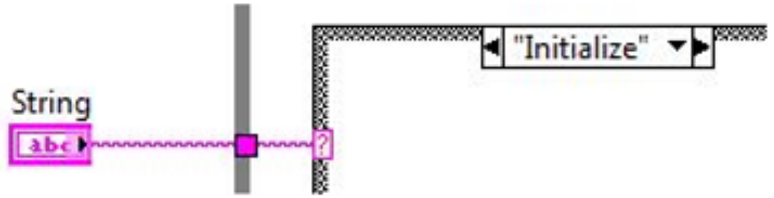
A



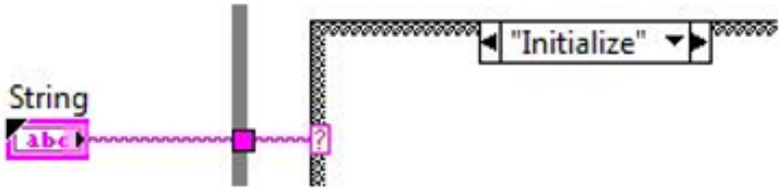
B



C

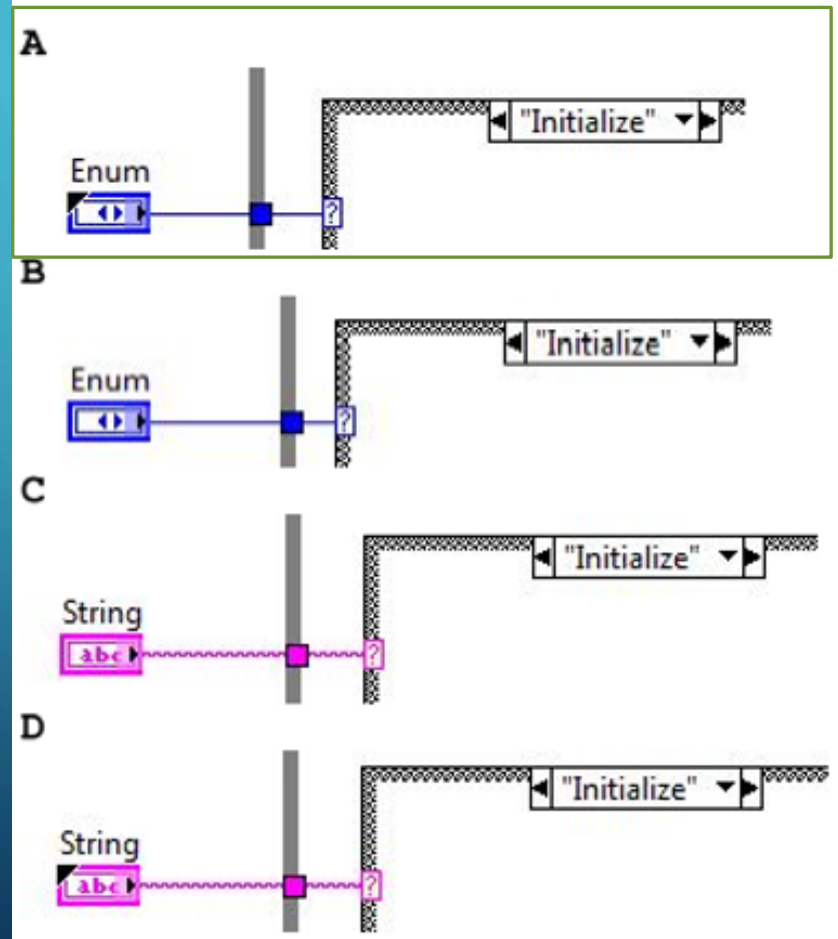


D



CLAD QUESTION

Q17: What is the best choice for the case selector input for a Functional Global Variable design pattern?



Just like with a State Machine, a enumerated constant set to Type Def is the best option. This is so that when we update the enumerated constant, it updates on all of the VIs that use it.