Introduction to
LabVIEW Design Patterns

NATIONAL INSTRUMENTS

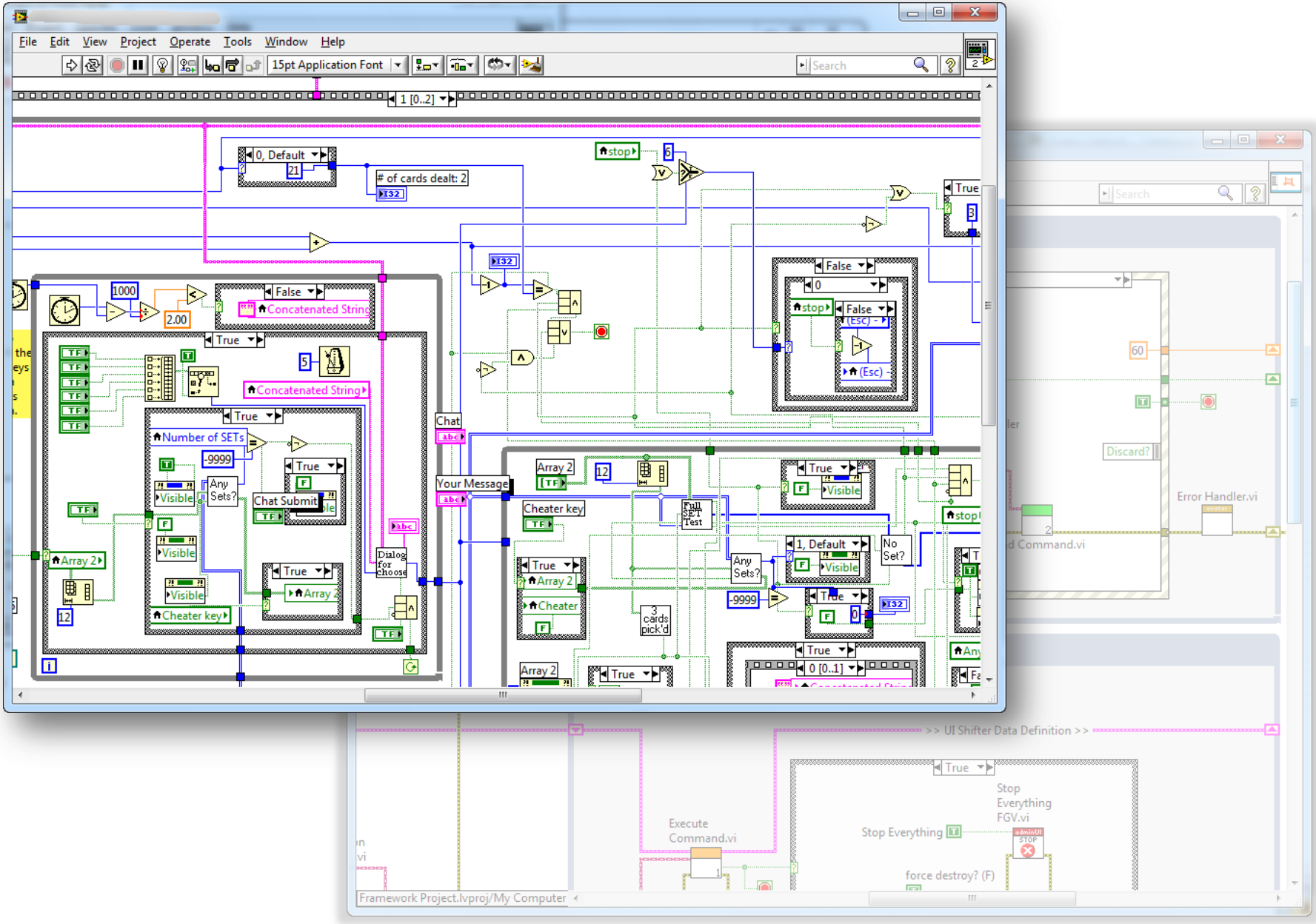
What is a Design Pattern?

Definition: A well-established solution to a common problem.

Why Should I Use One?

Save time and improve the longevity and readability of your code.

… or else…



NATIONAL INSTRUMENTS

Examples of Software Engineering Debt

(just *some* of the most common LabVIEW development mistakes)

- ✓ No source code control (or Project)
- ✓ Flat file hierarchy
- ✓ ‘Stop’ isn’t tested regularly
- ✓ Wait until the ‘end’ of a project to build an application
- ✓ Few specifications / documentation / requirements
- ✓ No ‘buddying’ or code reviews
- ✓ **Poor planning (Lack of consideration for SMoRES)**
- ✓ No test plans
- ✓ Poor error handling
- ✓ No consistent style
- ✓ Tight coupling, poor cohesion

ni.com/largeapps

NATIONAL INSTRUMENTS

Designing for SMoRES

Criteria for a well designed software application:

Scalable: how simple is N + 1?

Modular: is the application broken up into well-defined components that stand on their own?

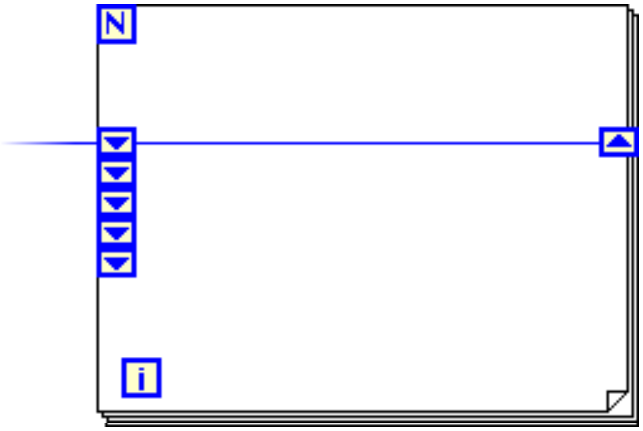
Reusable: is the code de-coupled from the current application well-enough such that it could be reused in a future project?

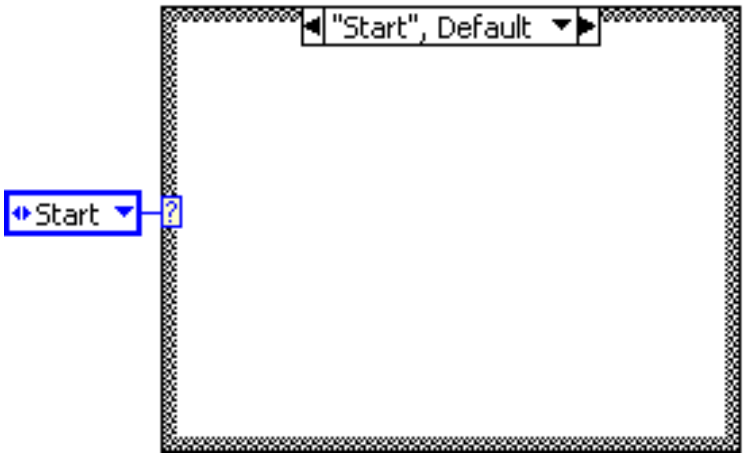
Extensible: how painful is it to add new functionality?

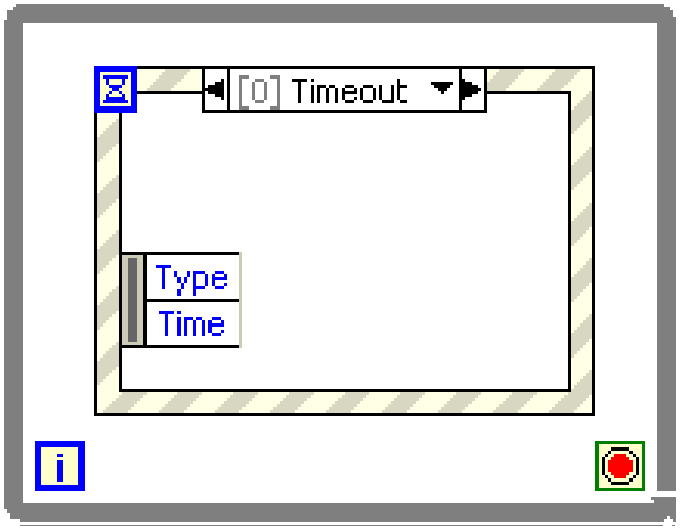
Simple: what is the simplest solution that satisfies all of the listed criteria and the requirements of the application?

You Should Already Be Familiar With..

- Loops
- Shift Registers
- Case Structures
- Enumerated Constants
- Event Structures
- LabVIEW Classes







NATIONAL INSTRUMENTS

Design Patterns

- Functional Global Variable
- State Machine / Statecharts
- Event Driven User Interface
- Producer / Consumer
- Queued State Machine – Producer / Consumer

Functional Global Variables

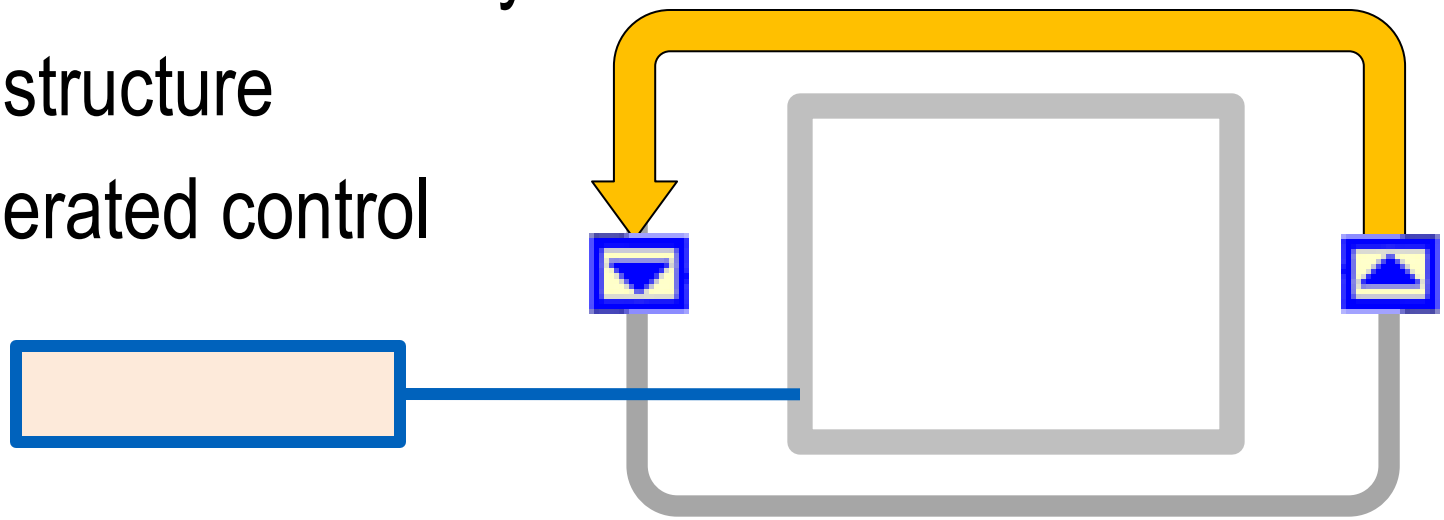
How do I share data across a application without using Global or Local Variables?

Background: Global and Local Variables

- Can cause race conditions
- Create copies of data in memory
- Cannot perform actions on data
- Cannot handle error wires

Breaking Down the Design Pattern

- While loop
- Uninitialized shift registers have memory
- Case structure
- Enumerated control

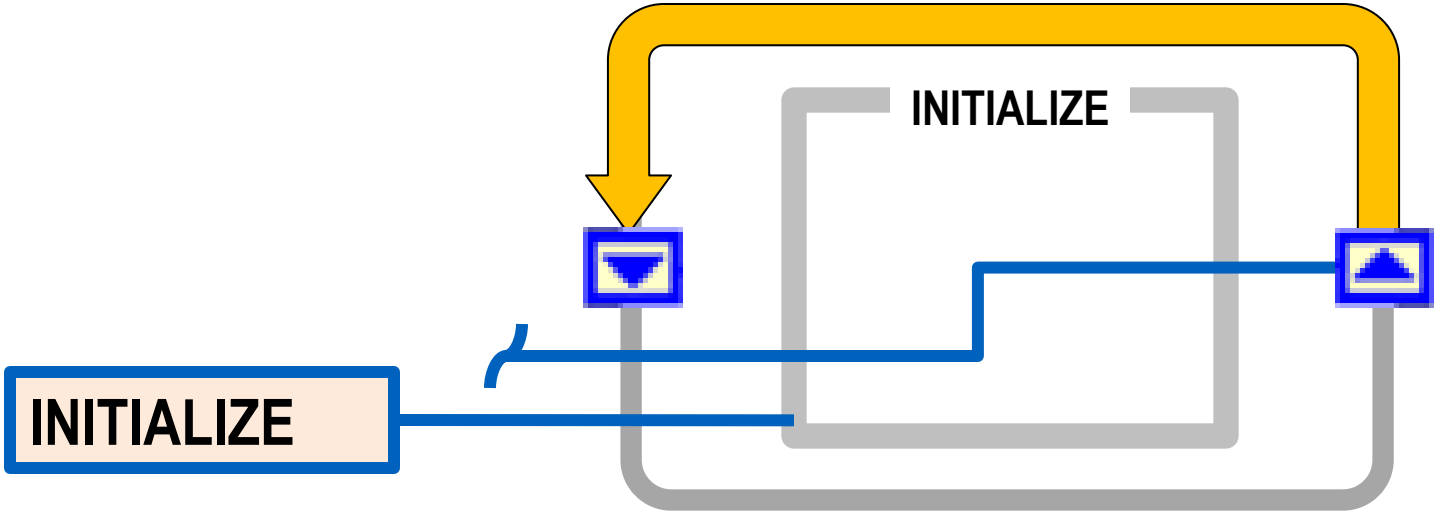


Uninitialized Shift Registers

DEMO

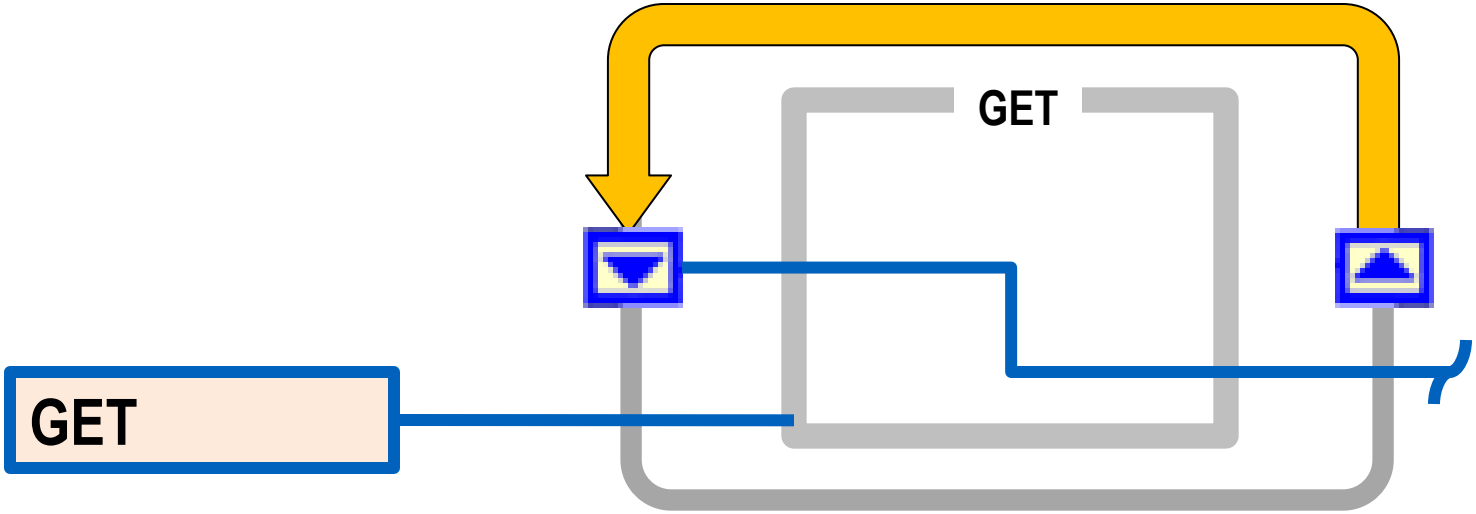
Basic Actions

- Set the value of the shift register



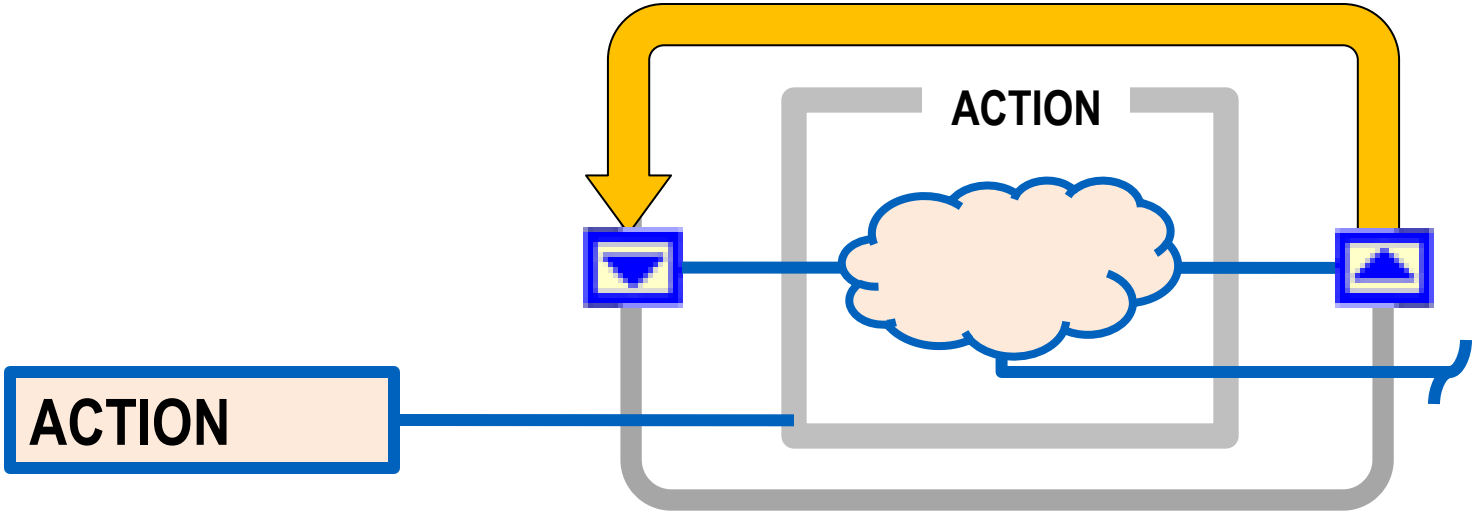
Basic Actions

- Get the value currently stored in the shift register



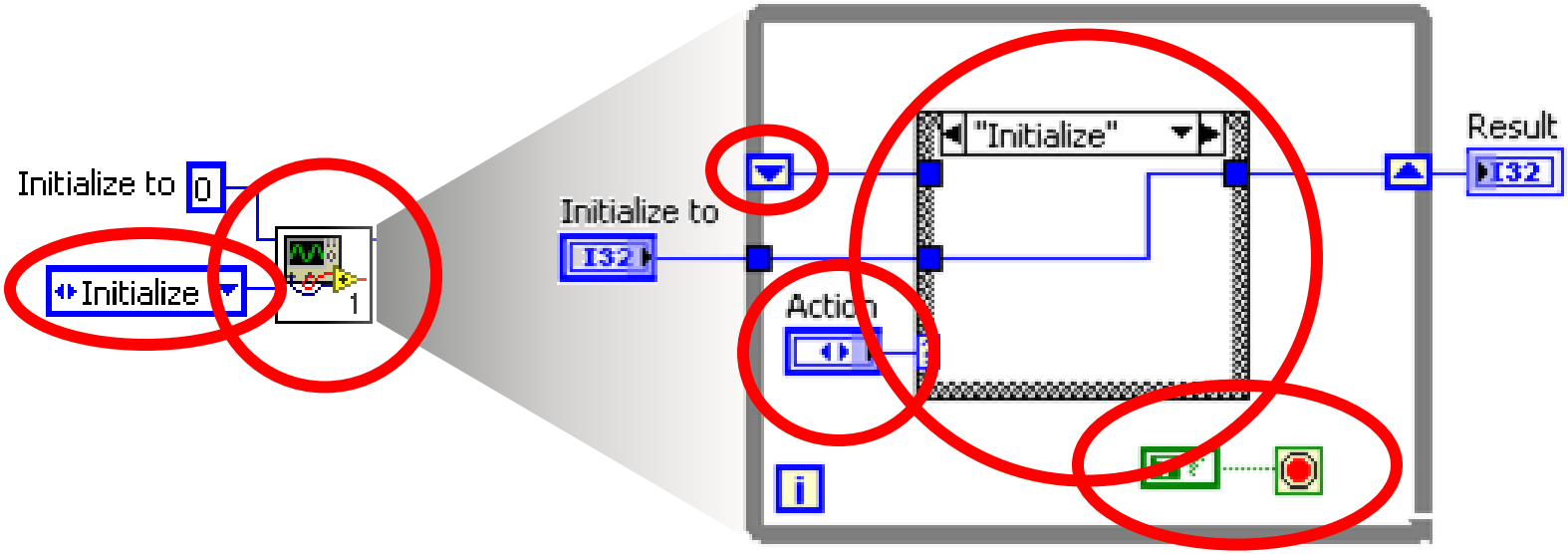
Action Engine

- Perform an operation upon stored value and save result
- You can also output the new value

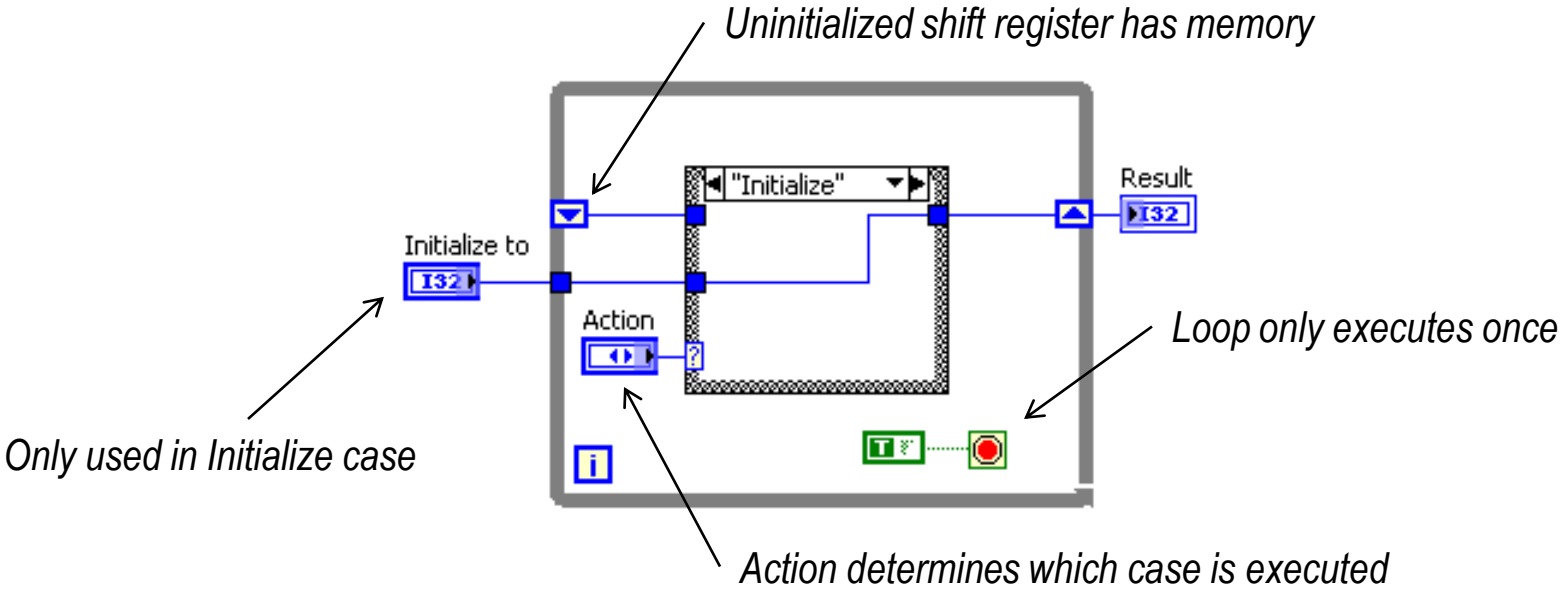


How It Works

1. Functional Global Variable is a **Non-Reentrant** SubVI
2. Actions can be performed upon data
3. Enumerator selects action
4. Stores result in uninitialized shift register
5. Loop only executes once



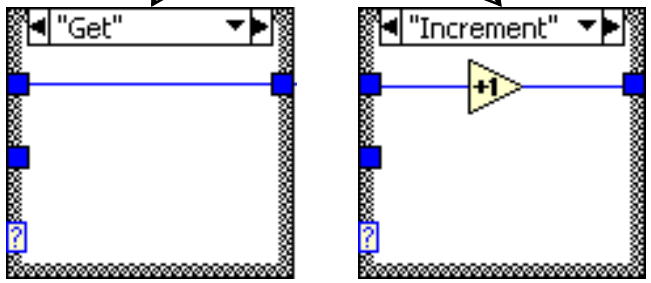
NATIONAL INSTRUMENTS



Functional Global Variables

DEMO

Examples of other 'actions'



NATIONAL INSTRUMENTS

Benefits: Comparison

Functional Global Variables

- Prevent race conditions
- No copies of data
- Can behave like action engines
- Can handle error wires
- Take time to make

Global and Local Variables

- Can cause race conditions
- Create copies of data in memory
- Cannot perform actions on data
- Cannot handle error wires
- Drag and drop

Recommendations

Use Cases

- Communicate data between code without connecting wires
- Perform custom actions upon data while in storage

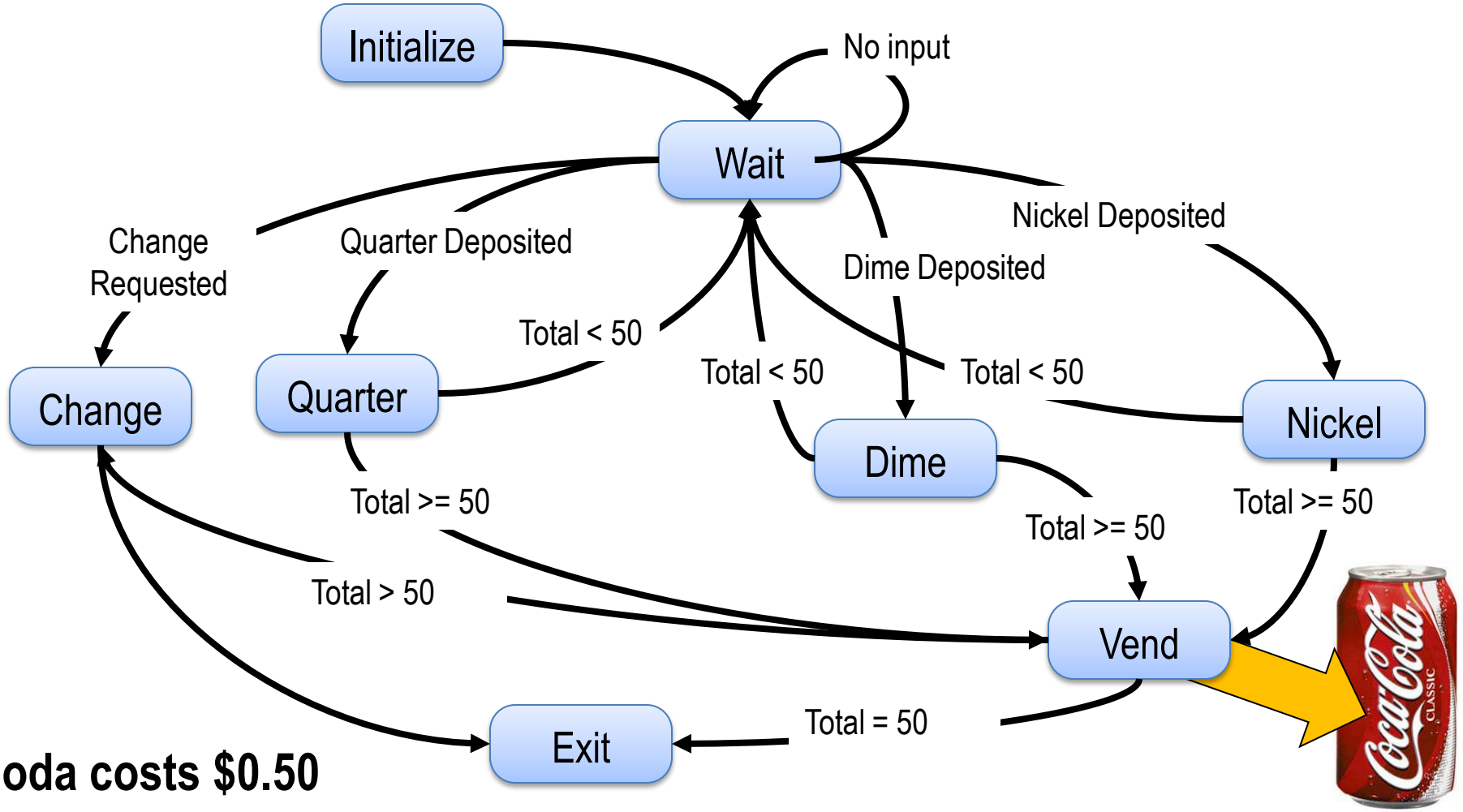
Considerations

- All owning VIs must stay in memory
- Use clusters to reduce connector pane
- Using stacked shift registers will track multiple iterations

State Machine

I need to execute a sequence of events, but the order is determined programmatically

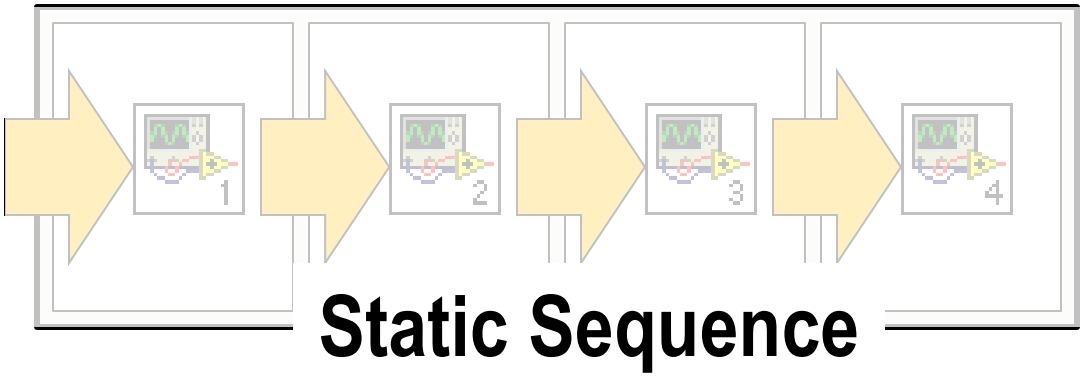
Soda Machine



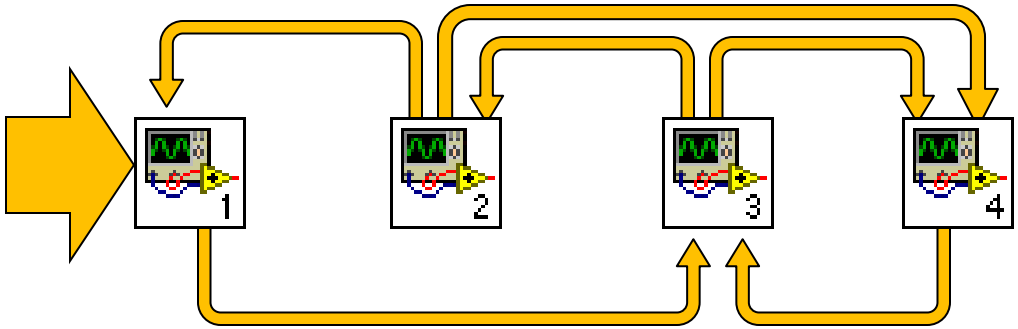
Soda costs $0.50

NATIONAL INSTRUMENTS

Background



Dynamic Sequence: Allows distinct states to operate in a programmatically determined sequence

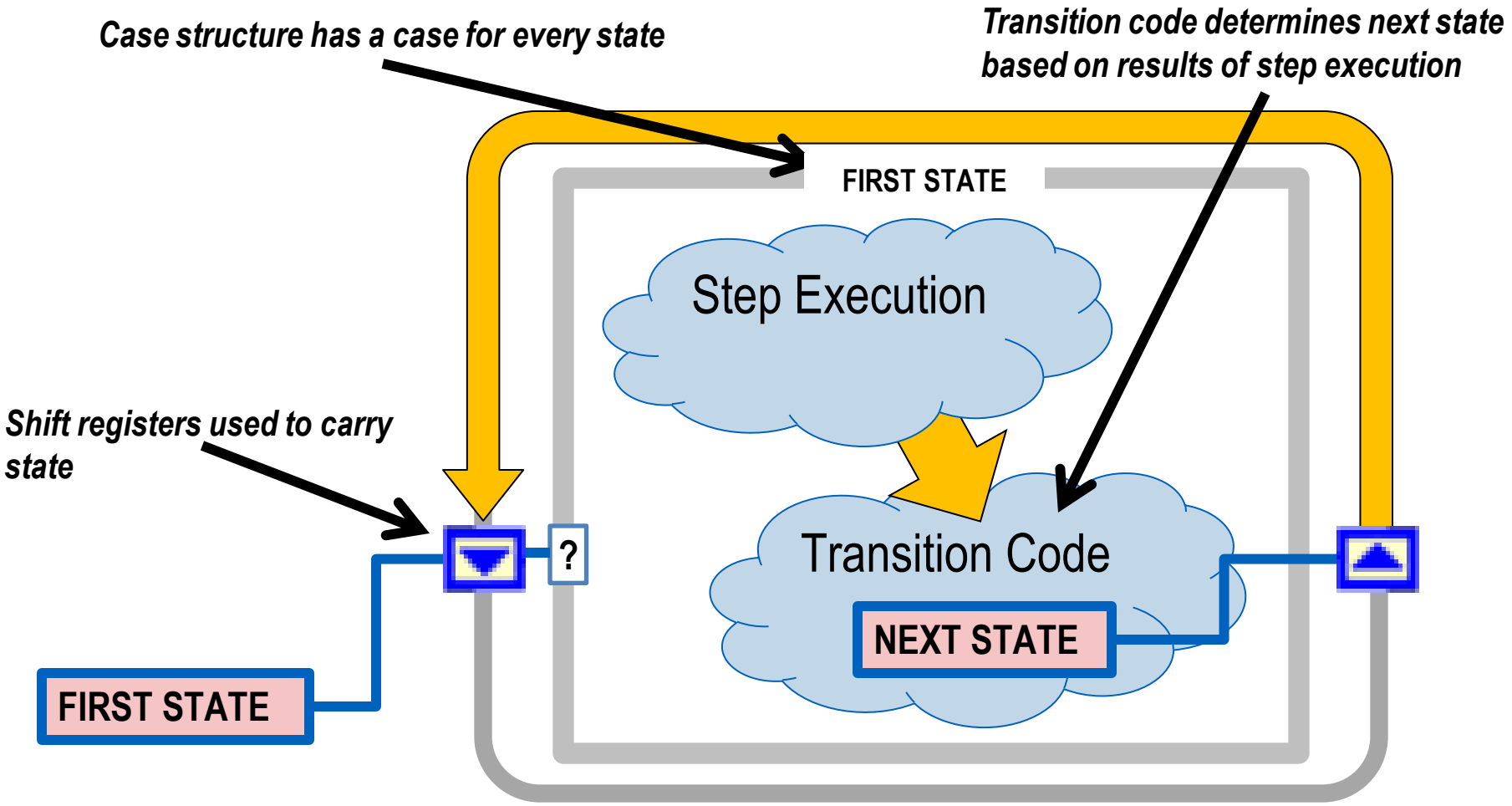


NATIONAL INSTRUMENTS

Breaking Down the Design Pattern

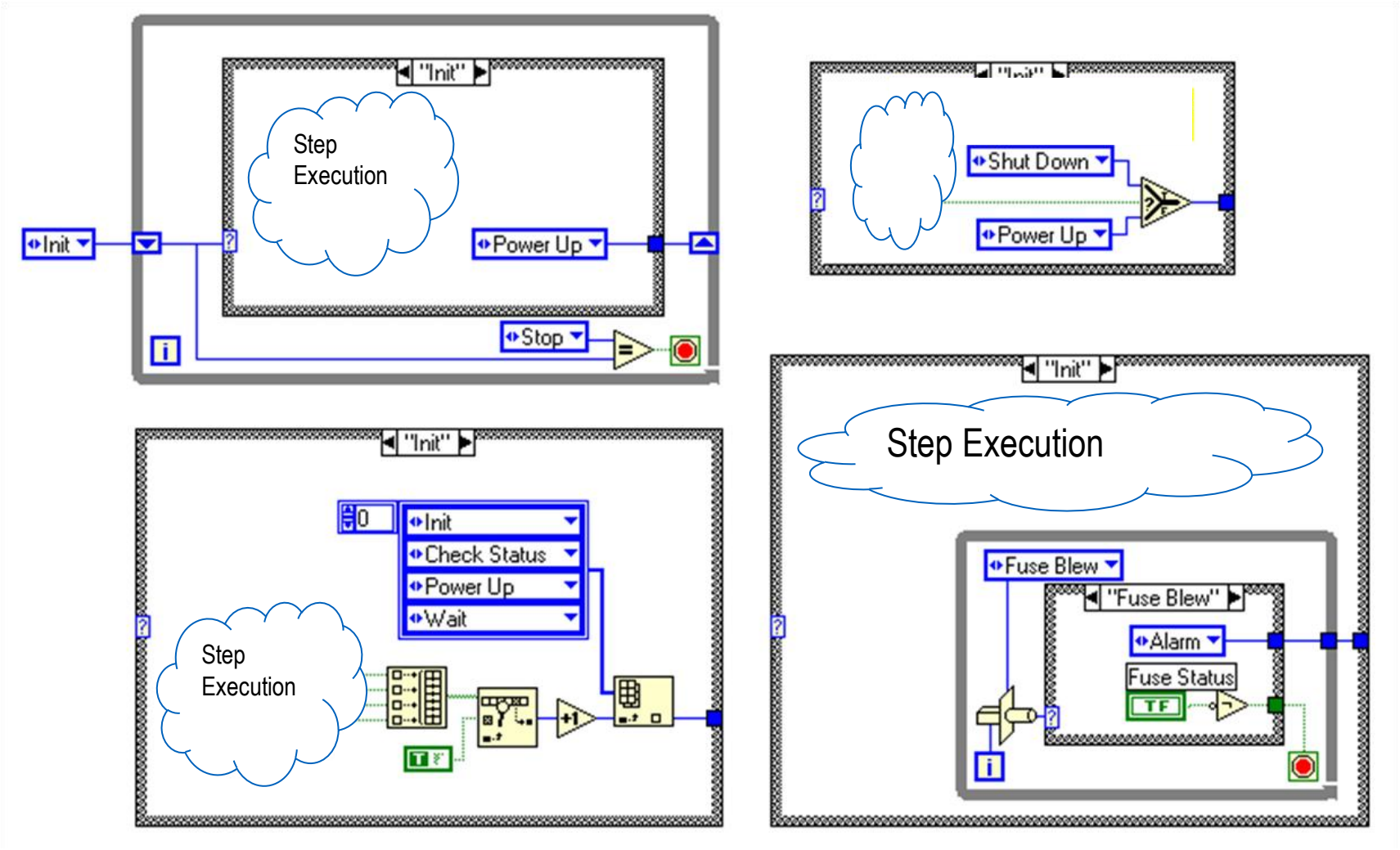
- Case Structure inside of a While Loop
- Each case is a state
- Current state has decision making code that determines next state
- Use enumerators to pass value of next state to shift registers

The Anatomy of a State Machine



NATIONAL INSTRUMENTS

Transition Code Options



NATIONAL INSTRUMENTS

State Machine

DEMO

Recommendations

Use Cases

- User interfaces
- Data determines next routine

Considerations

- Creating an effective State Machine requires the designer to make a table of possible states.

Event Driven User Interface

I’m polling for user actions, which is slowing my application down, and sometimes I don’t detect them!

Background

Procedural-driven programming

- Set of instructions are performed in sequence
- Requires polling to capture events
- Cannot determine order of multiple events

Event-driven programming

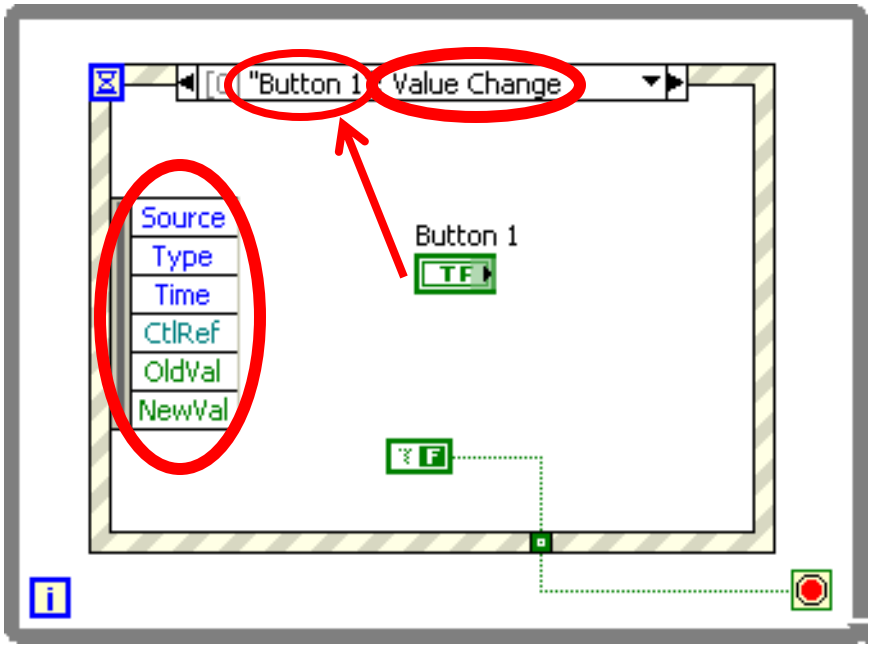
- Execution determined at run-time
- Waits for events to occur without consuming CPU
- Remembers order of multiple events

Breaking Down the Design Pattern

- Event structure nested within loop
- Blocking function until event registered or timeout
- Events that can be registered:
  - Notify events are only for interactions with the front panel
  - Dynamic events allows programmatic registration
  - Filter events allow you to screen events before they're processed

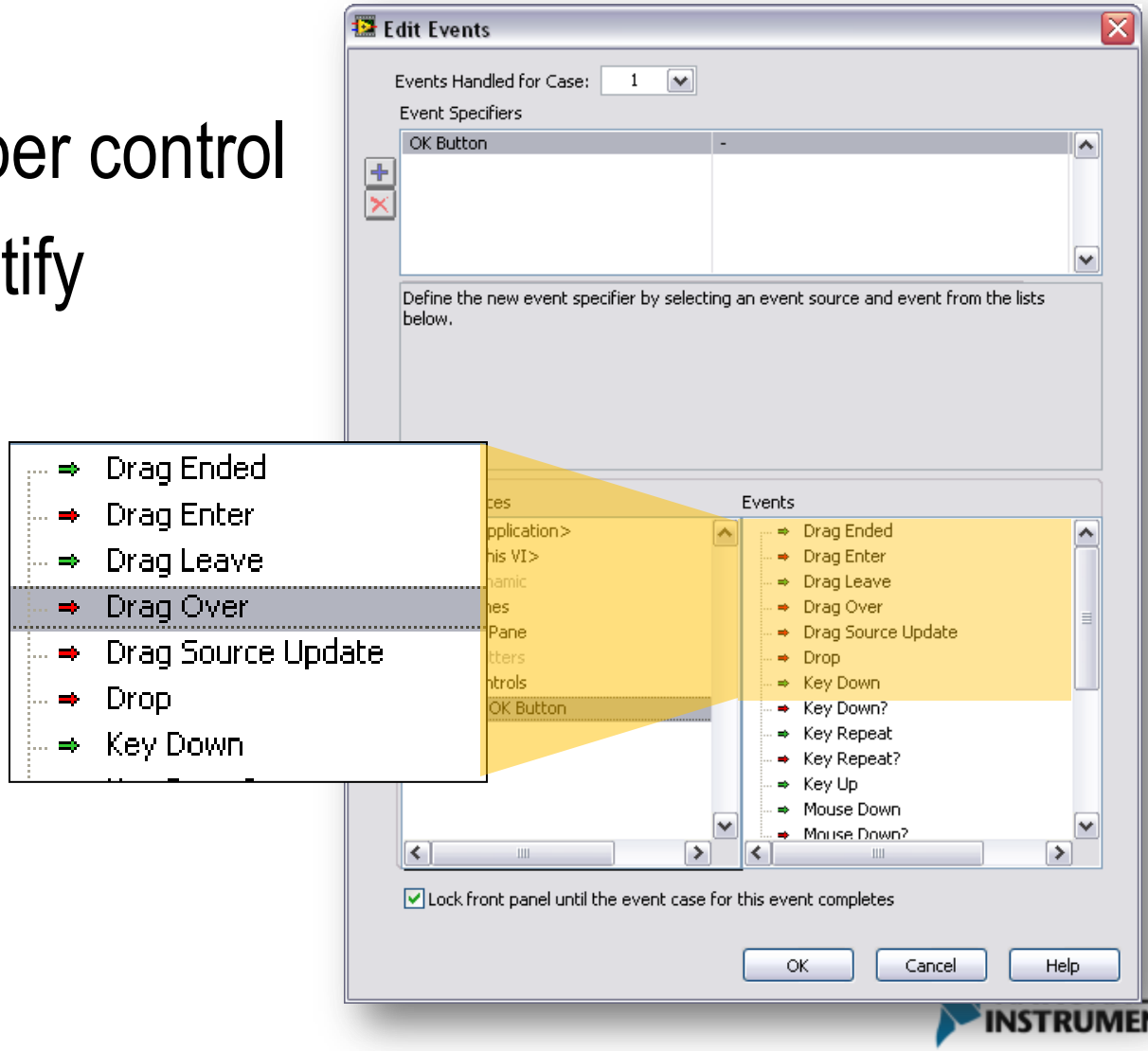
How It Works

1. Operating system broadcasts system events (mouse click, keyboard, etc..) to applications
2. Registered events are captured by event structure and executes appropriate case
3. Event structure returns information about event to case
4. Event structure enqueues events that occur while it's busy



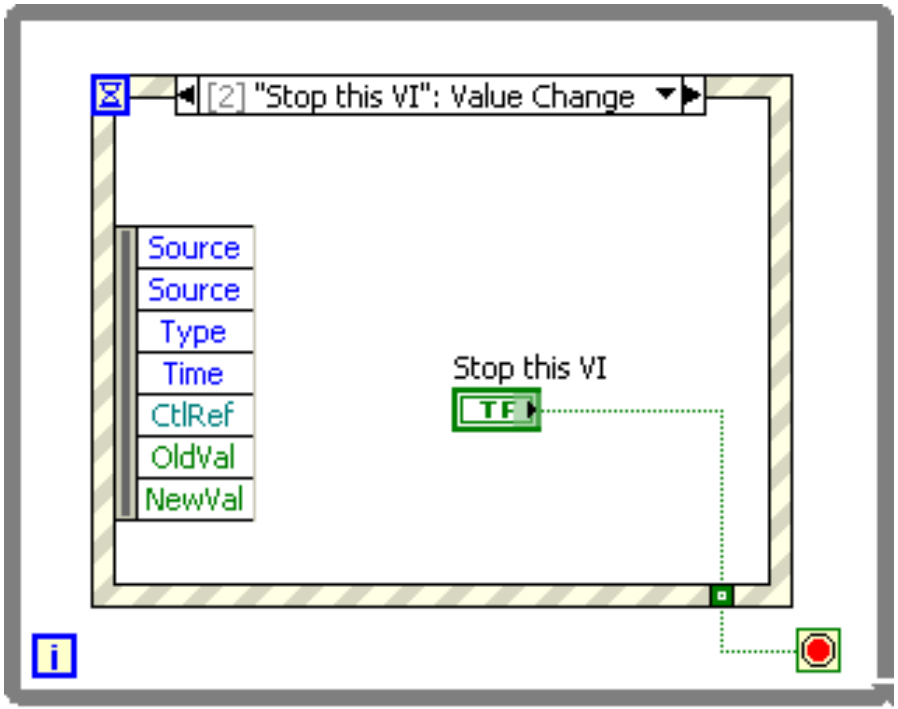
How It Works: Static Binding

- Browse controls
- Browse events per control
- Green arrow: notify
- Red arrow: filter



Event Driven User Interface

DEMO



Recommendations

Use Cases

- UI: Conserve CPU usage
- UI: Ensure you never miss an event
- Drive slave processes

Considerations

- Avoid placing two Event structures in one loop
- Remember to read the terminal of a latched Boolean control in its Value Change event case
- When using subpanel controls, the top-level VI containing the subpanel control handles the event

Producer / Consumer

I have two processes that need to execute at the same time, and I need to make sure one can't slow the other down

Background

I want to execute code in parallel and at asynchronous rates, but I need to communicate between them!

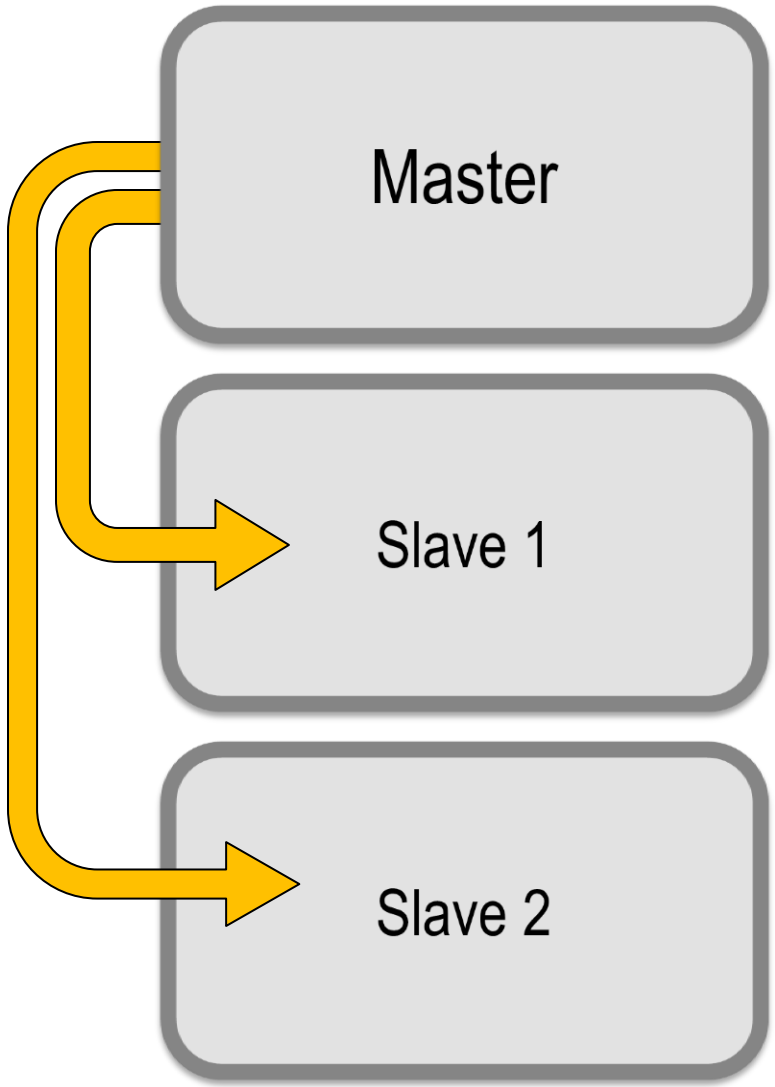
I have two processes that need to execute at the same time, but I want them to be independent of one another, and I need to make sure one can't slow the other down

Breaking Down the Design Pattern

- Data independent loops
- Master / slave relationship
- Communication and synchronization between loops

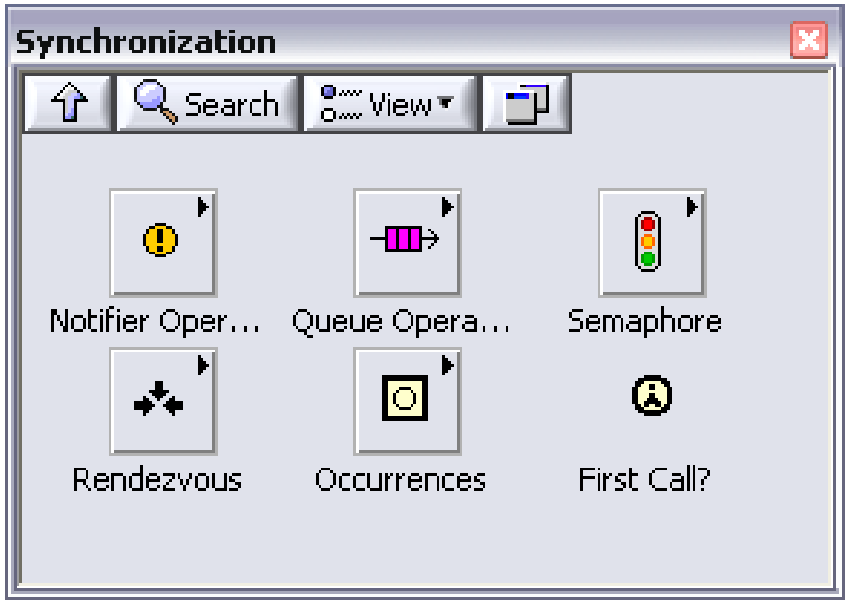
How It Works

- One or more slave loops are told by a master loop when they can run
- Allows for a-synchronous execution of loops
- Data-independence breaks dataflow and allows multi-threading
- De-couples processes



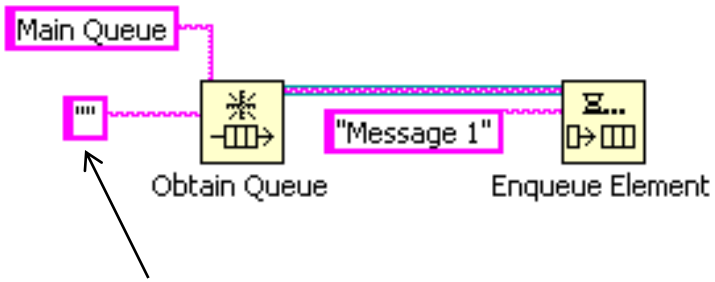
Master / Slave: Loop Communication

- Variables
- Occurrences
- Notifier
- Queues
- Semaphores
- Rendezvous



Queues

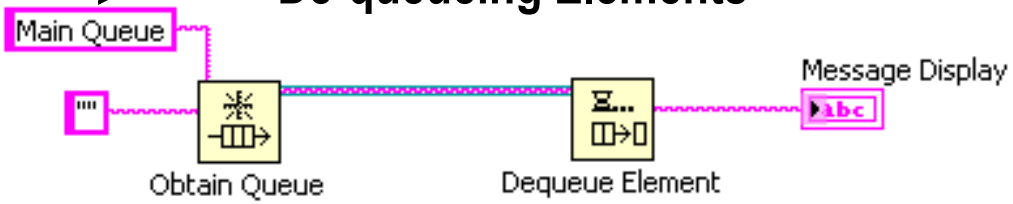
Adding Elements to the Queue



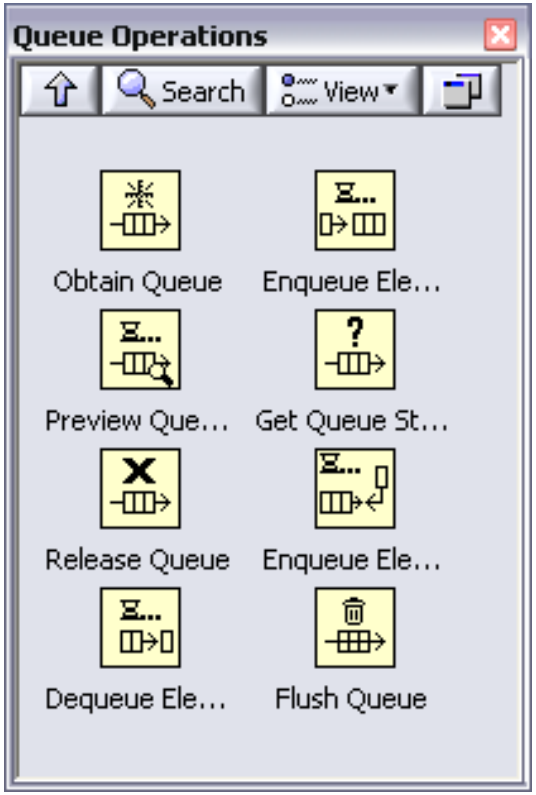
Select the data-type the queue will hold

Reference to existing queue in memory

De-queueing Elements

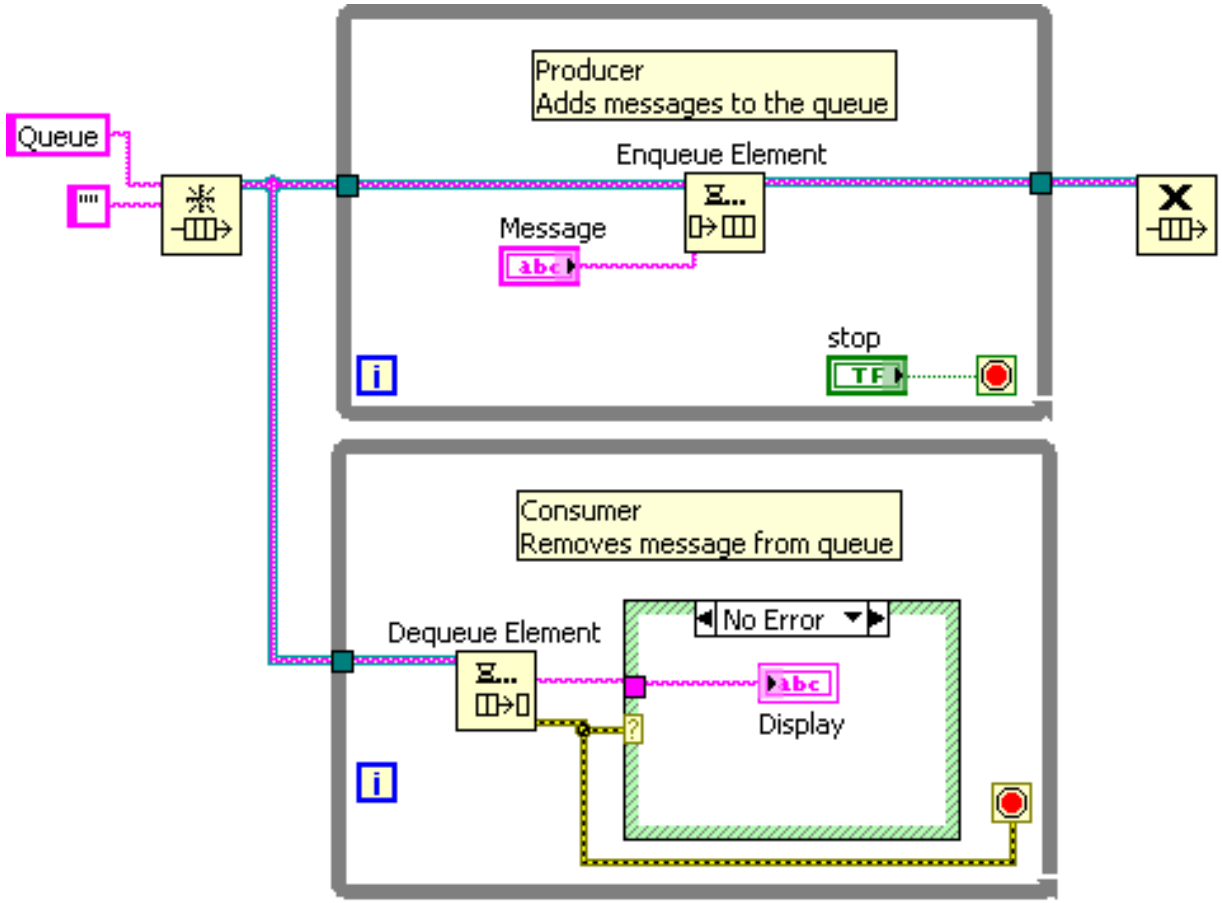


Dequeue will wait for data or timeout (defaults to -1)



NATIONAL INSTRUMENTS

Producer / Consumer



Producer / Consumer

DEMO

Recommendations

Use cases

- Handling multiple processes simultaneously
- Asynchronous operation of loops

Considerations

- Multiple producers → One consumer
- One queue per consumer
- If order of execution of parallel loop is critical, use occurrences

Queued State Machine & Event-Driven Producer / Consumer

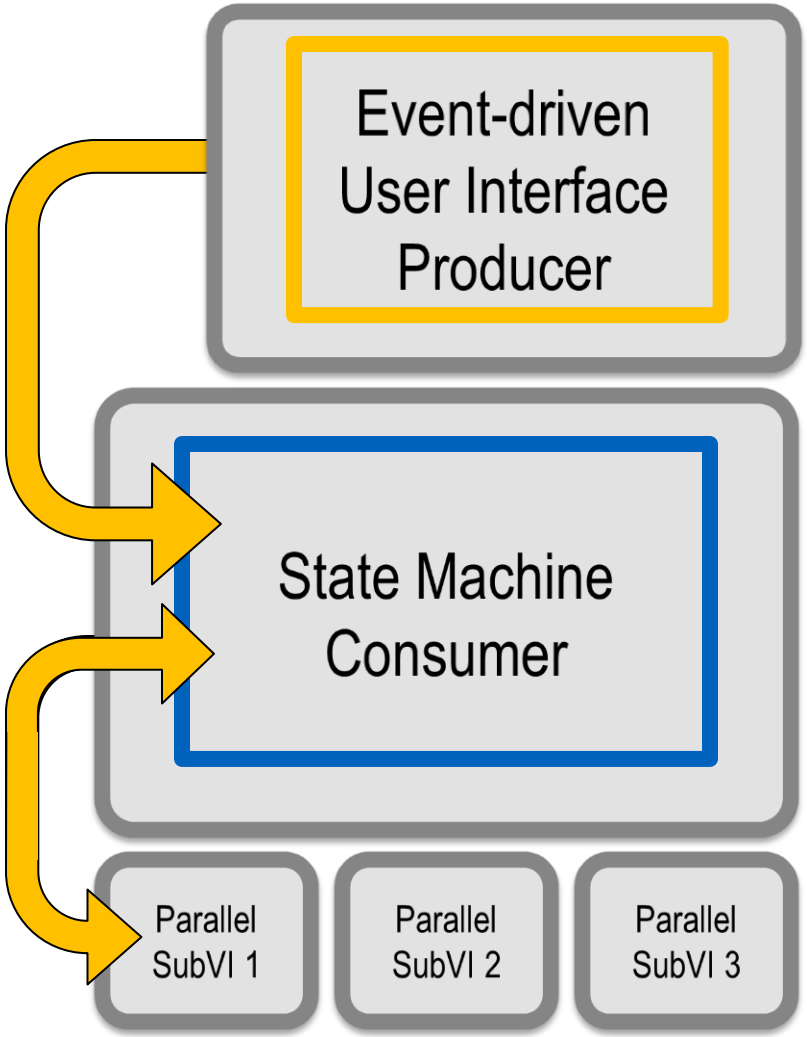
I need to enqueue events from a user that control the sequence of events in a consumer loop

Breaking Down the Design Pattern

- Event-driven user interface design pattern
- State machine design pattern
- Producer consumer design pattern
- Queued communication between loops

How It Works

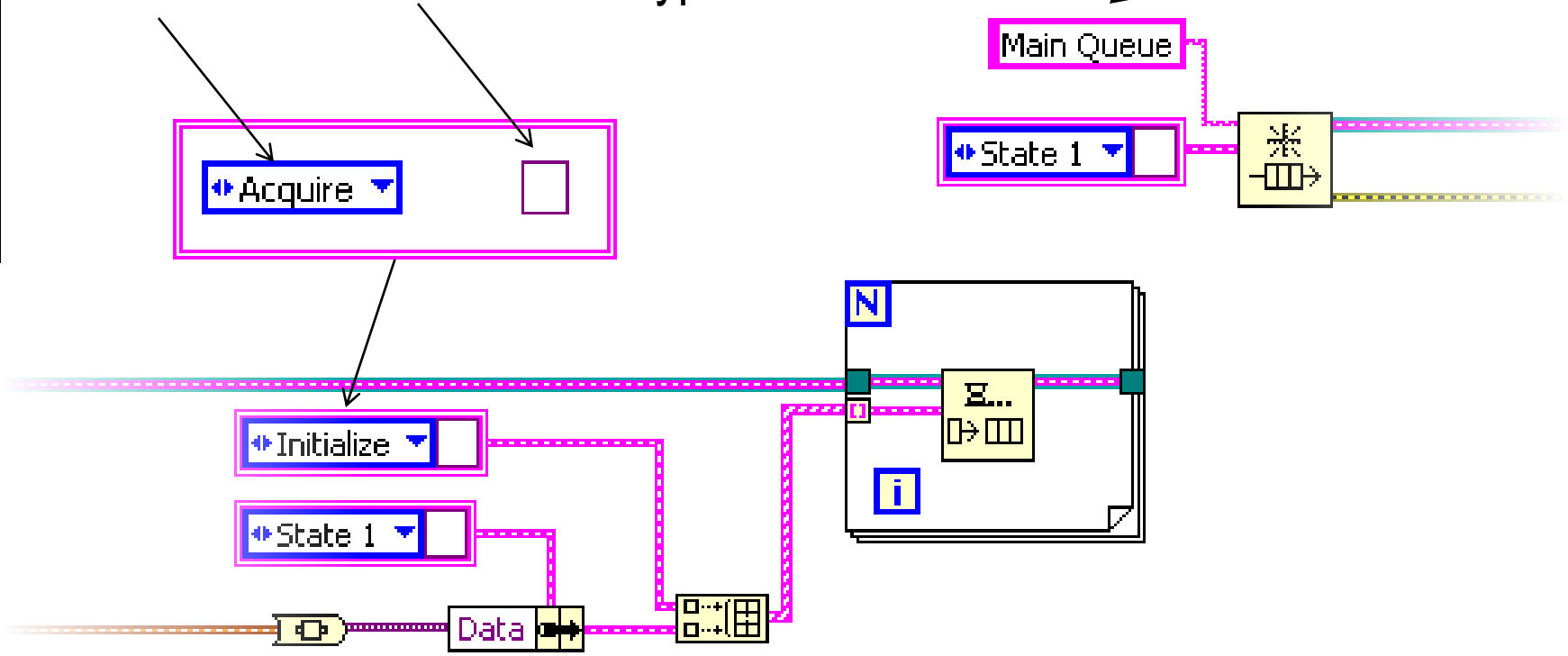
1. Events are captured by producer
2. Producer places data on the queue
3. State machine in consumer executes on dequeued data
4. Parallel SubVIs communicate using queue references



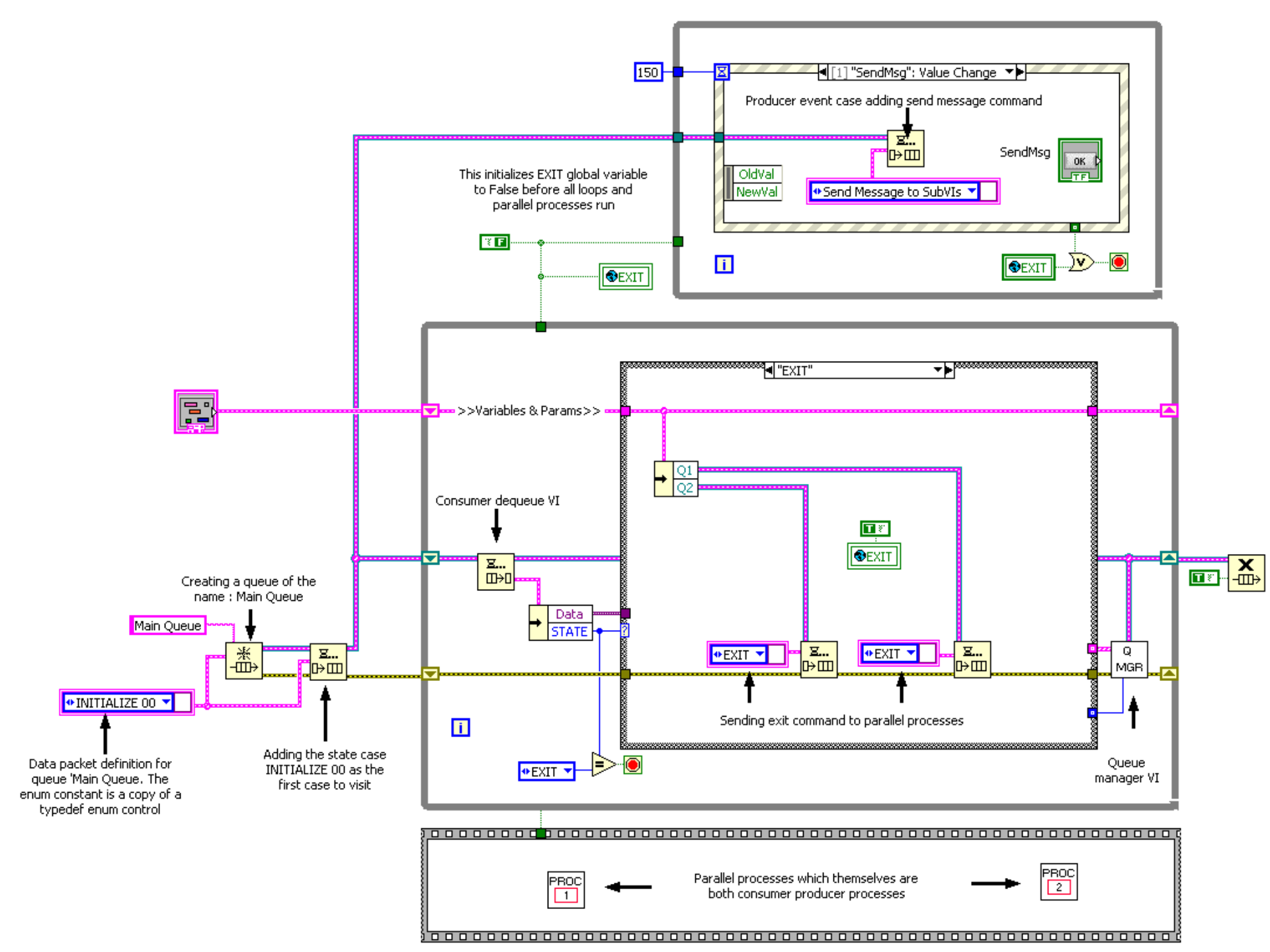
Queues Recommendations

Refer to queues by name for communication across VIs

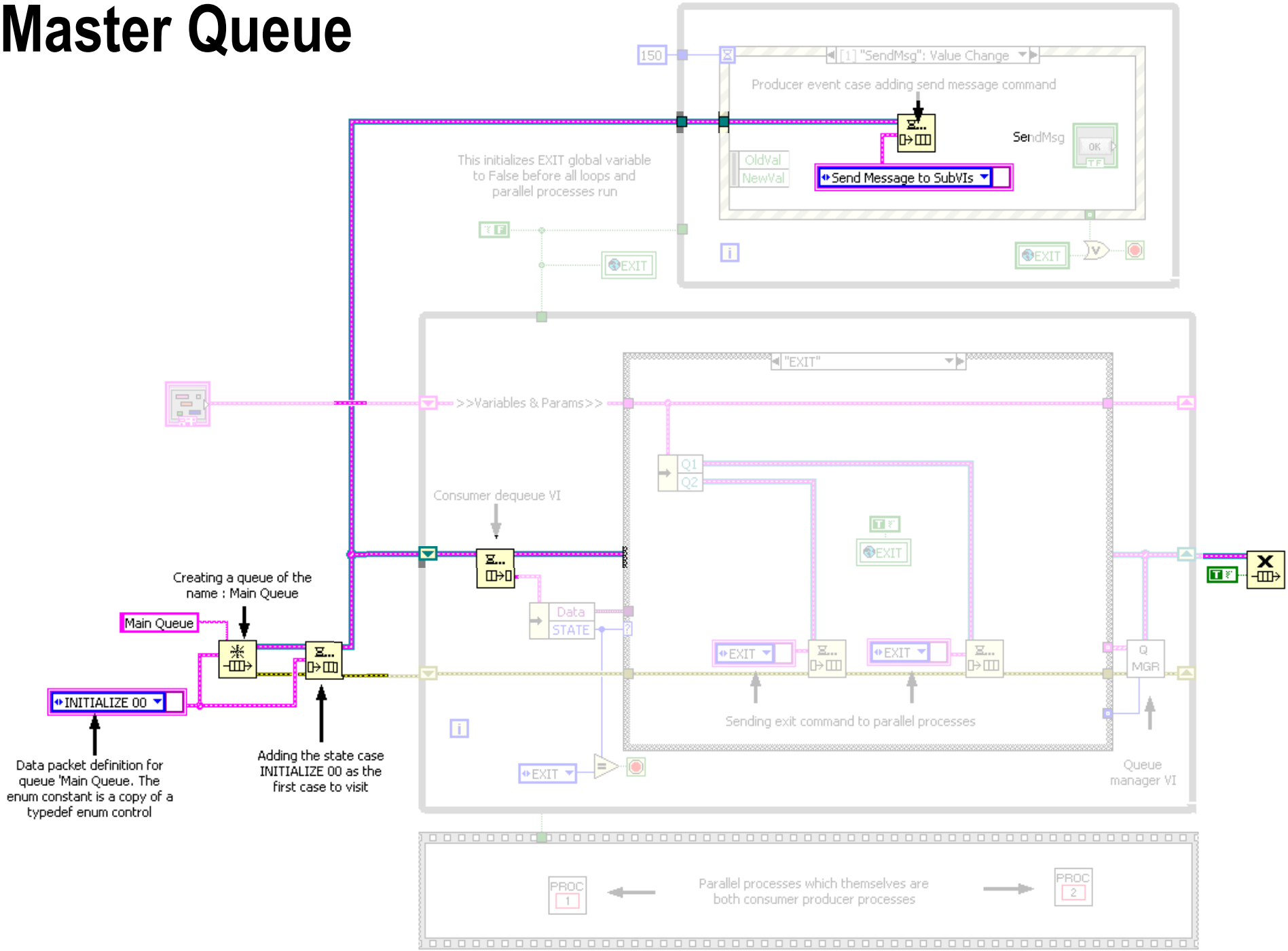
Use a cluster containing an enum and variant as data-type



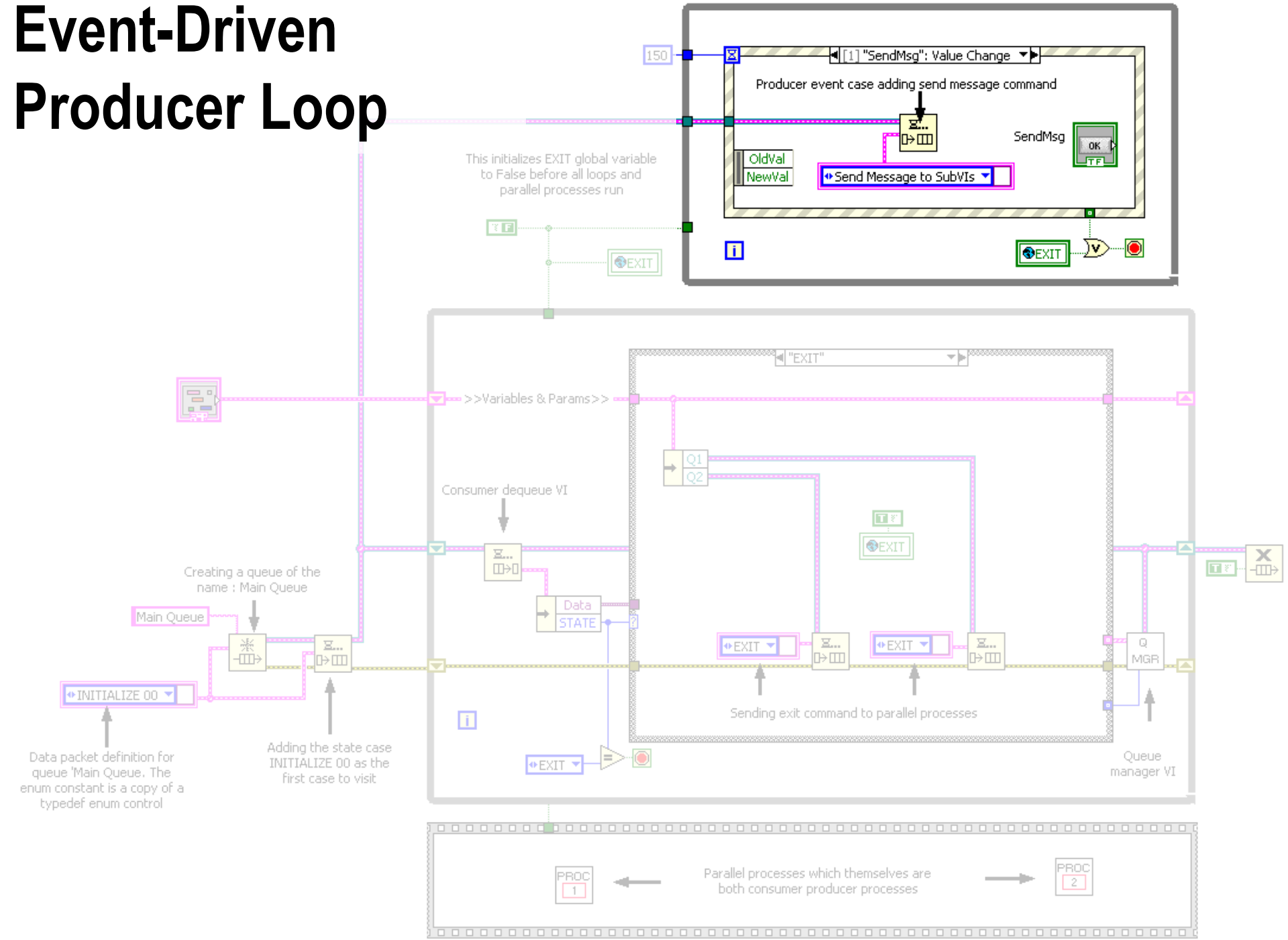
NATIONAL INSTRUMENTS


150
[1] "SendMsg": Value Change
Producer event case adding send message command
SendMsg
OK
TF
OldVal
NewVal
Send Message to SubVIs
This initializes EXIT global variable to False before all loops and parallel processes run
EXIT
"EXIT"
>>Variables & Params>>
Q1
Q2
Consumer dequeue VI
Creating a queue of the name : Main Queue
Main Queue
Data
STATE
INITIALIZE 00
Sending exit command to parallel processes
Q MGR
Queue manager VI
Data packet definition for queue 'Main Queue. The enum constant is a copy of a typedef enum control
Adding the state case INITIALIZE 00 as the first case to visit
PROC 1
Parallel processes which themselves are both consumer producer processes
PROC 2

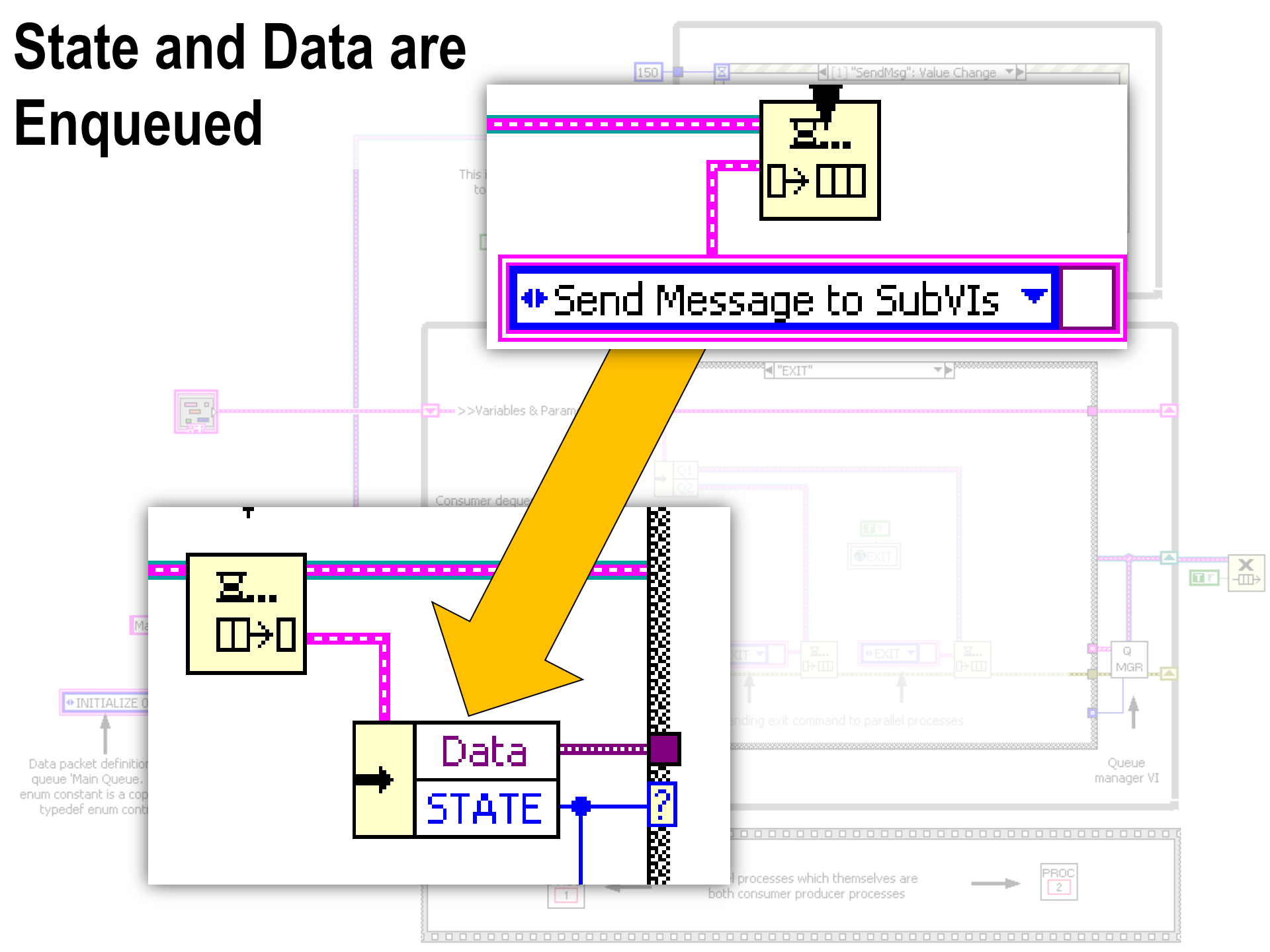
Master Queue



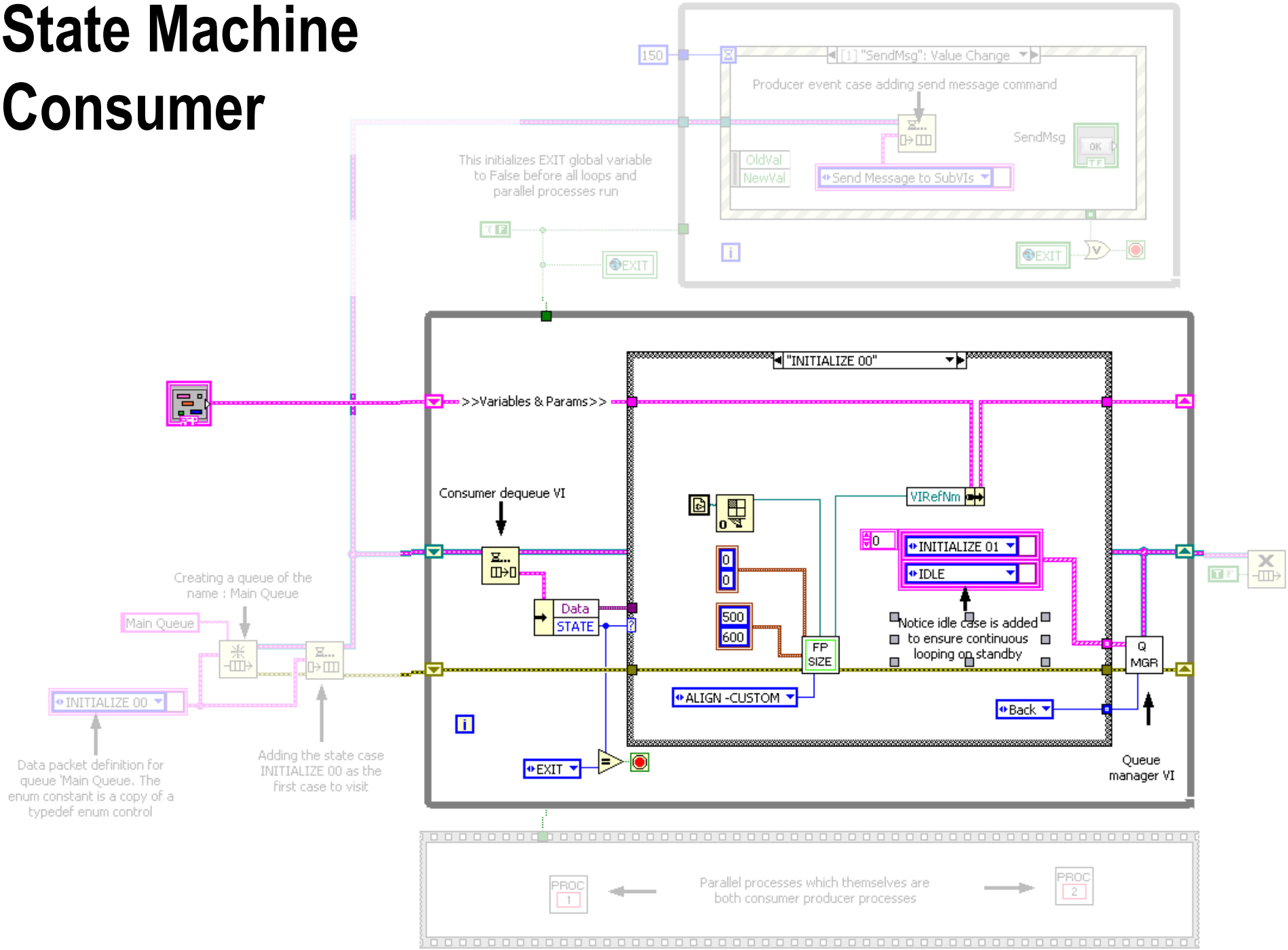
Event-Driven Producer Loop



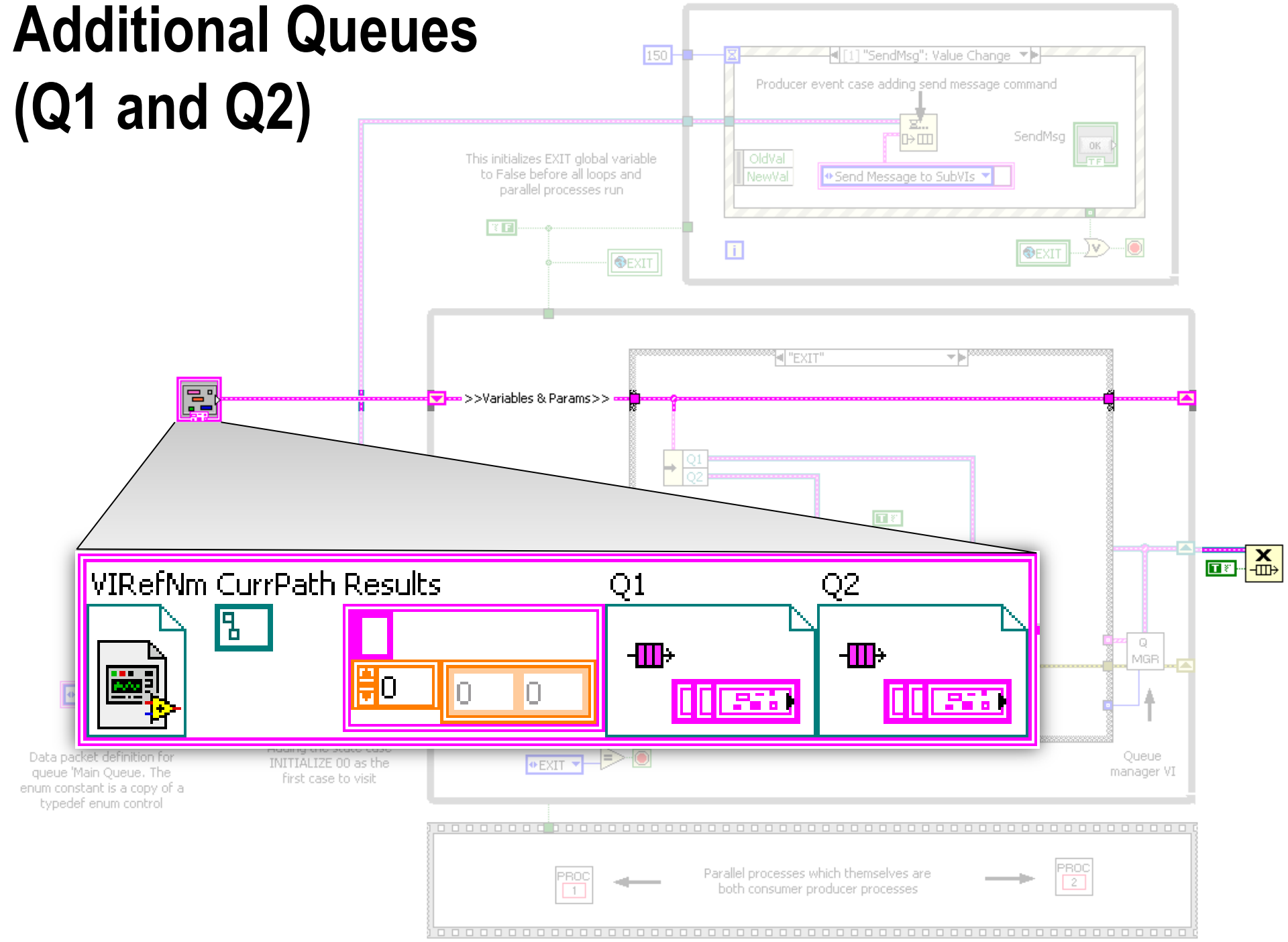
State and Data are Enqueued



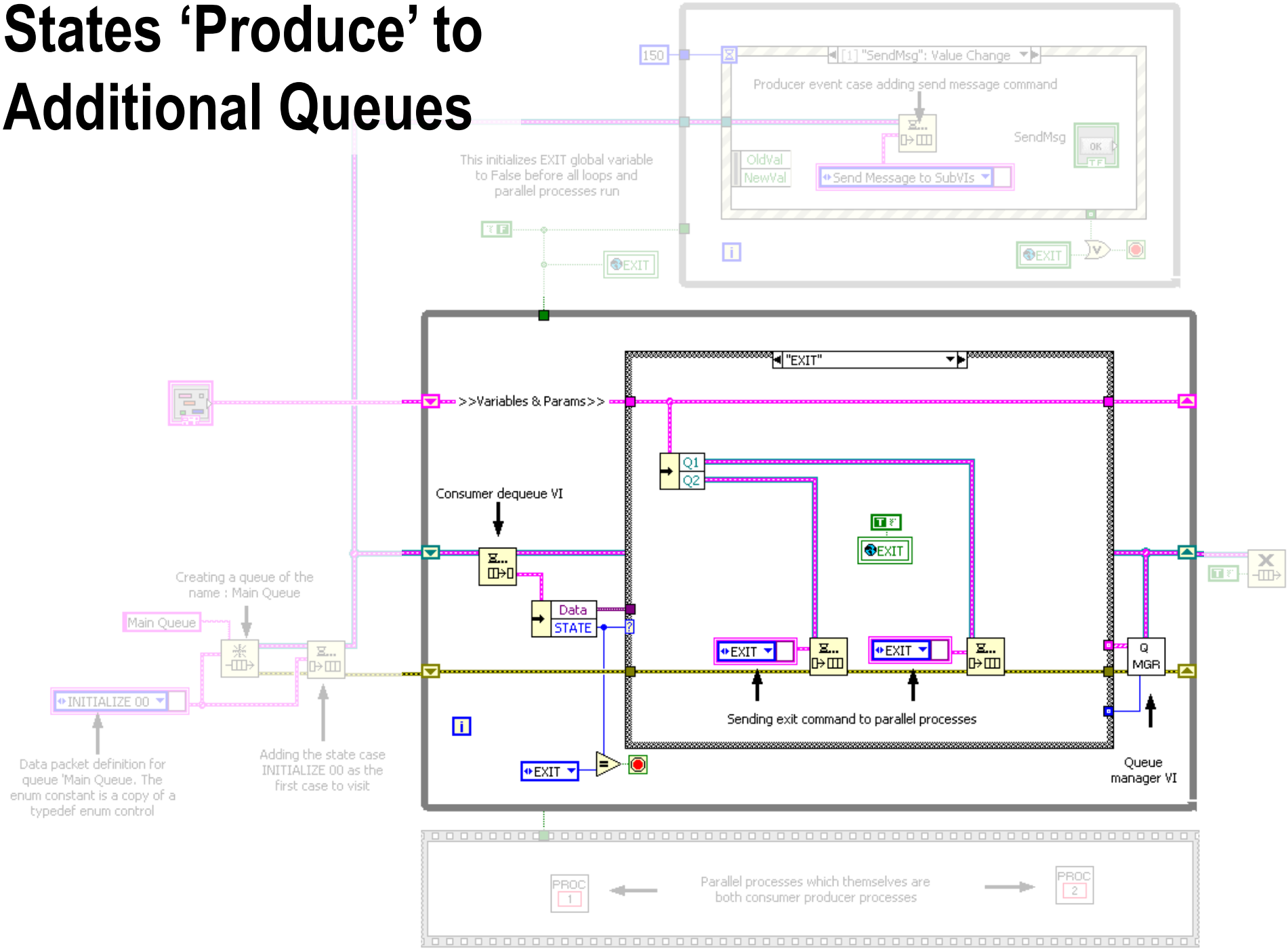
State Machine Consumer



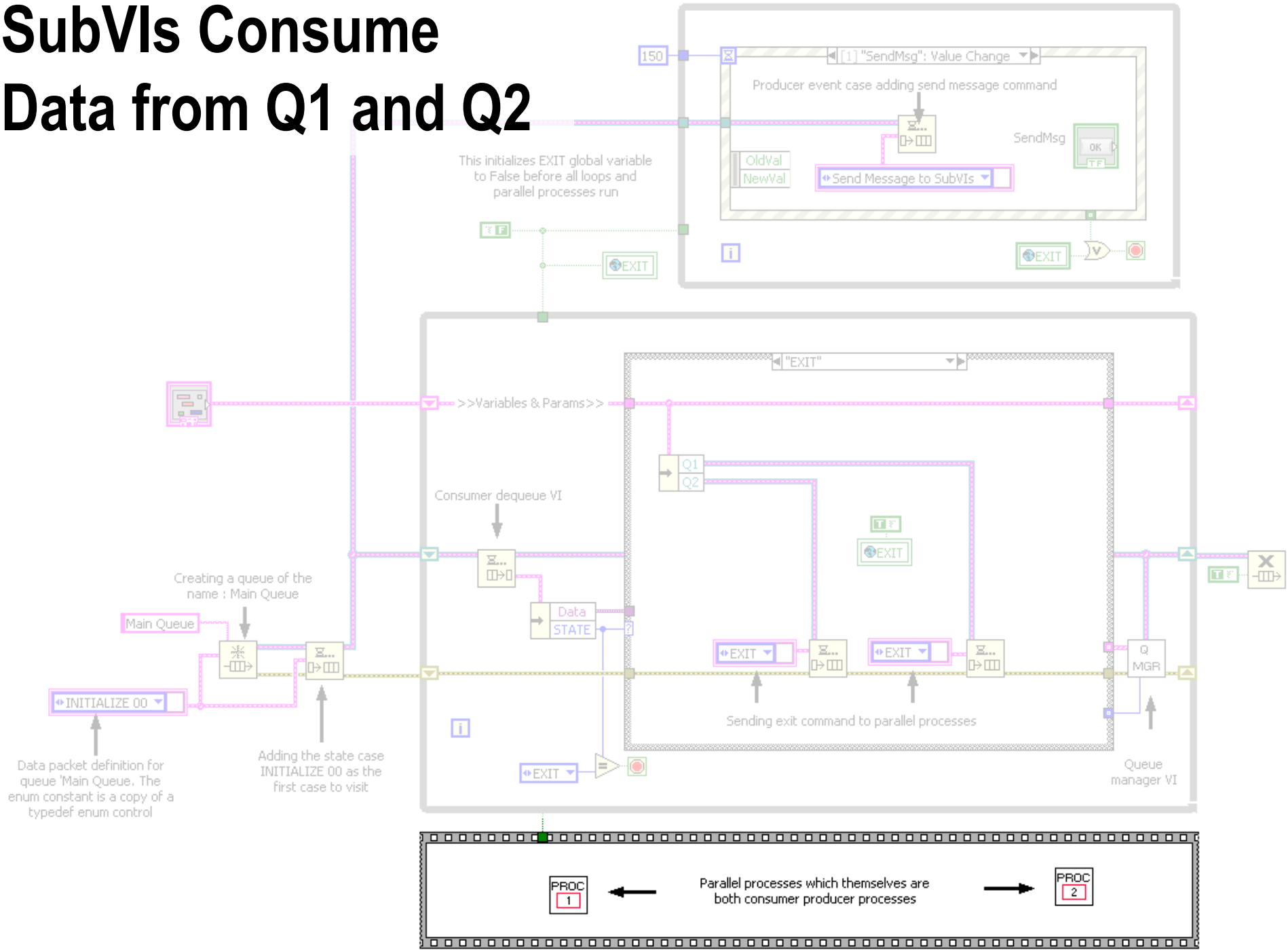
Additional Queues (Q1 and Q2)



States ‘Produce’ to Additional Queues



SubVIs Consume Data from Q1 and Q2



Queued State Machine – Producer/Consumer

DEMO

Recommendations

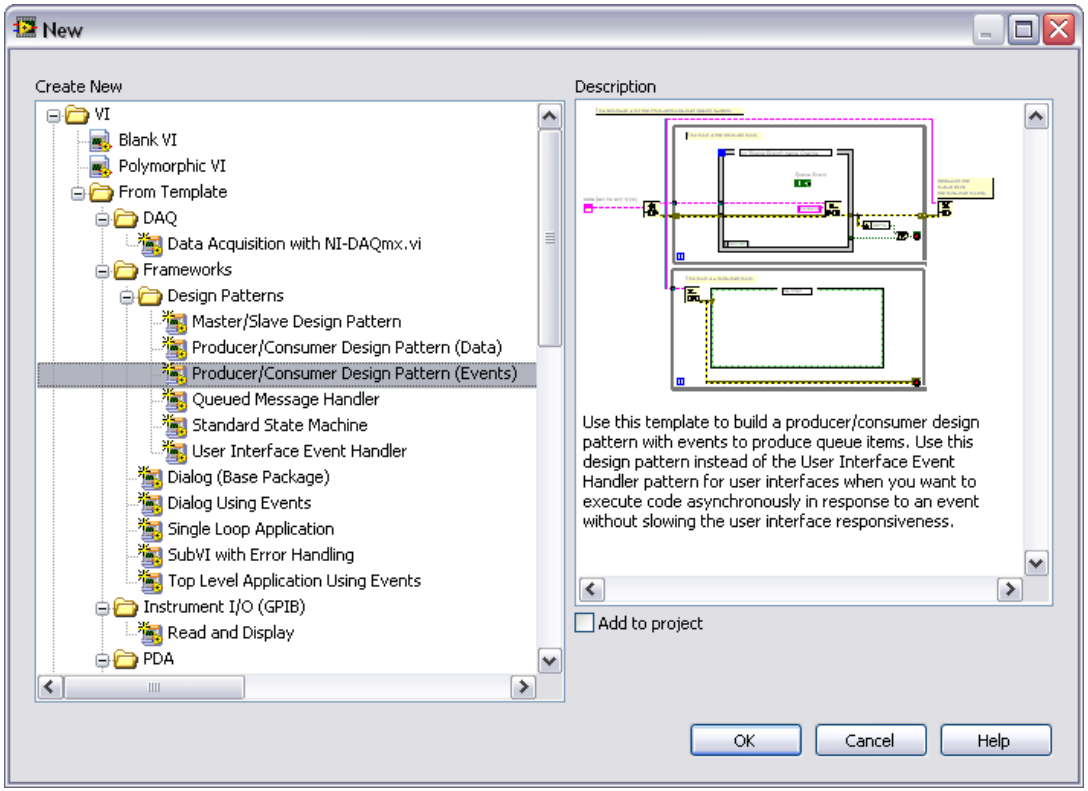
Use Cases

- Popular design pattern for mid to large size applications
- Highly responsive user interfaces
- Multithreaded applications
- De-coupling of processes

Considerations

- Complex design

Adding Your Own Design Patterns



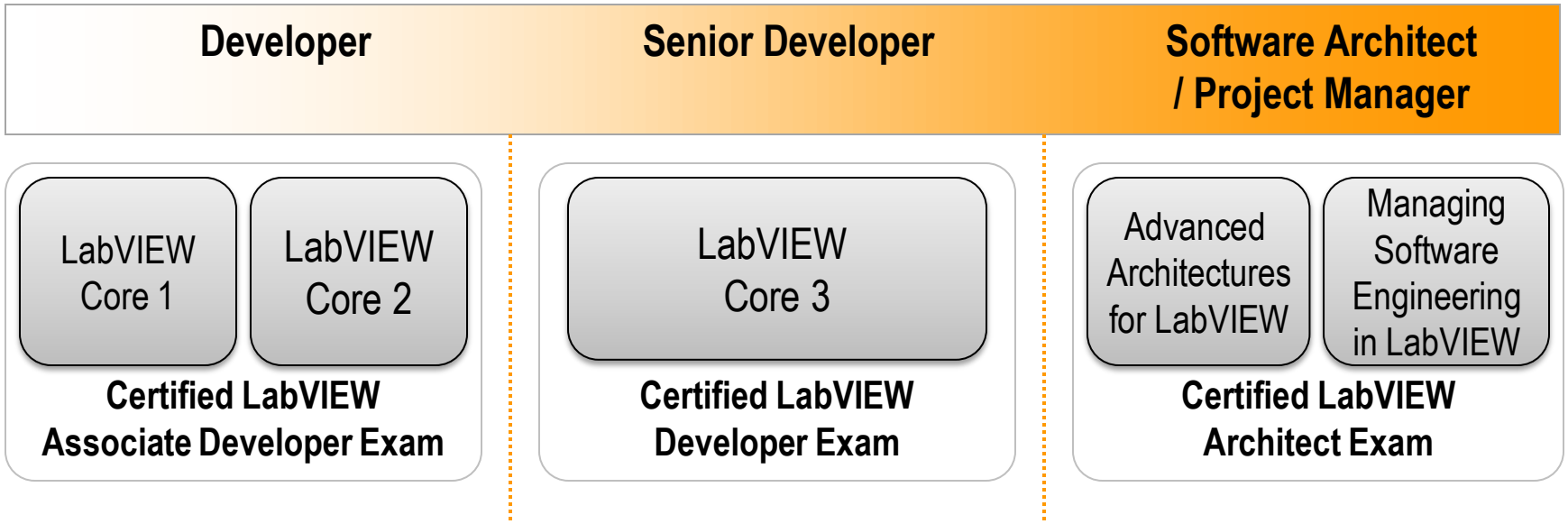
C:\Program Files\National Instruments\LabVIEW 8.5\templates\Frameworks\DesignPatterns

NATIONAL INSTRUMENTS

Resources

- Example Finder
- New >> Frameworks
- Ni.com/labview/power
- Training
  - LabVIEW Intermediate I & II
- White Paper on LabVIEW Queued State Machine Architecture
  - Expressionflow.com

NI Certifications Align with Training



"Certification is an absolute must for anyone serious about calling himself a LabVIEW expert... At our organization, we require that every LabVIEW developer be on a professional path to become a Certified LabVIEW Architect."

- President, JKI Software, Inc.

Download Examples and Slides

ni.com/largeapps

Software Engineering Tools

Development Practices

LargeApp Community