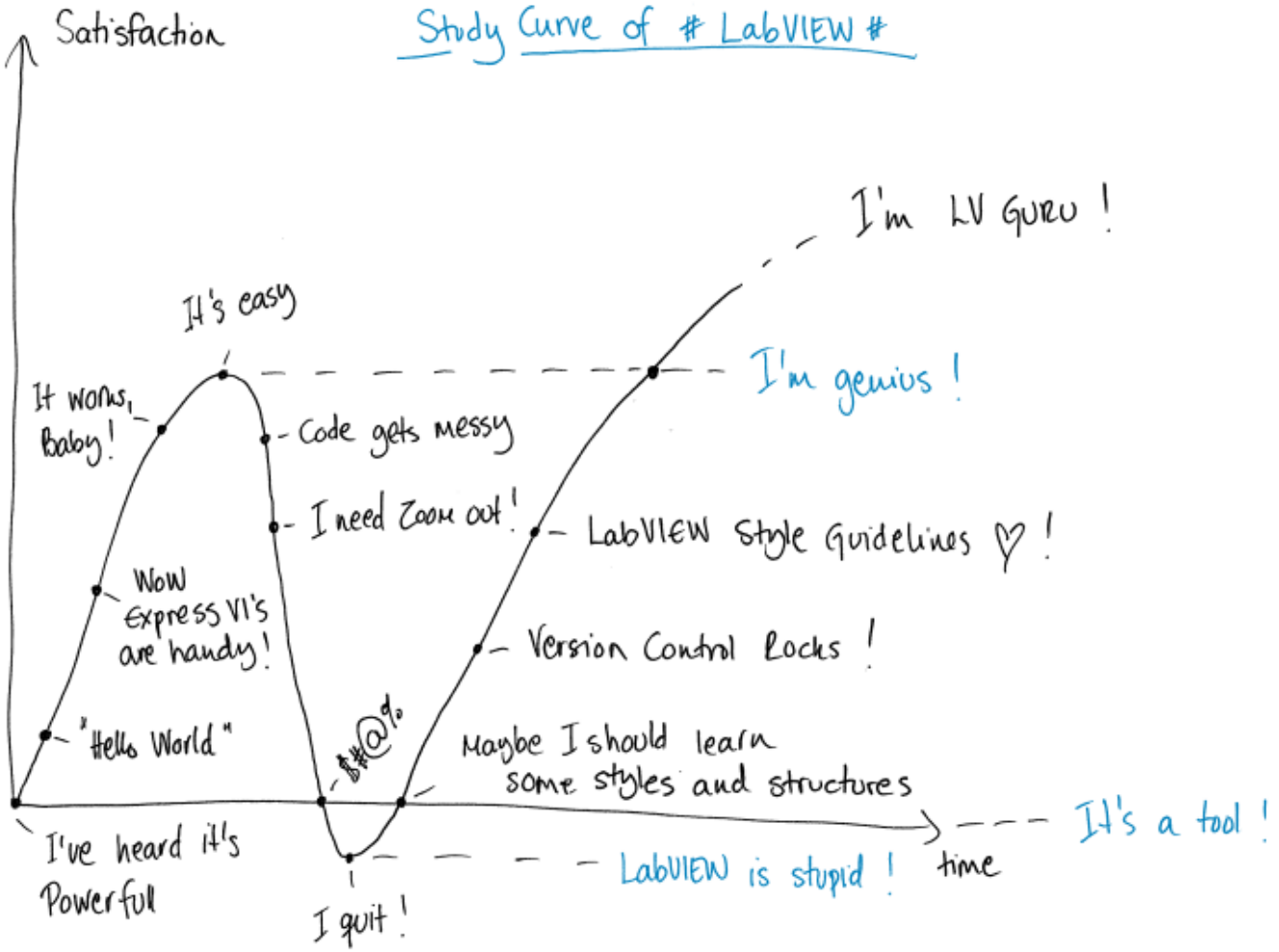
LabVIEW Programmer Days

johan.olsson@ni.com
0709-27 95 02

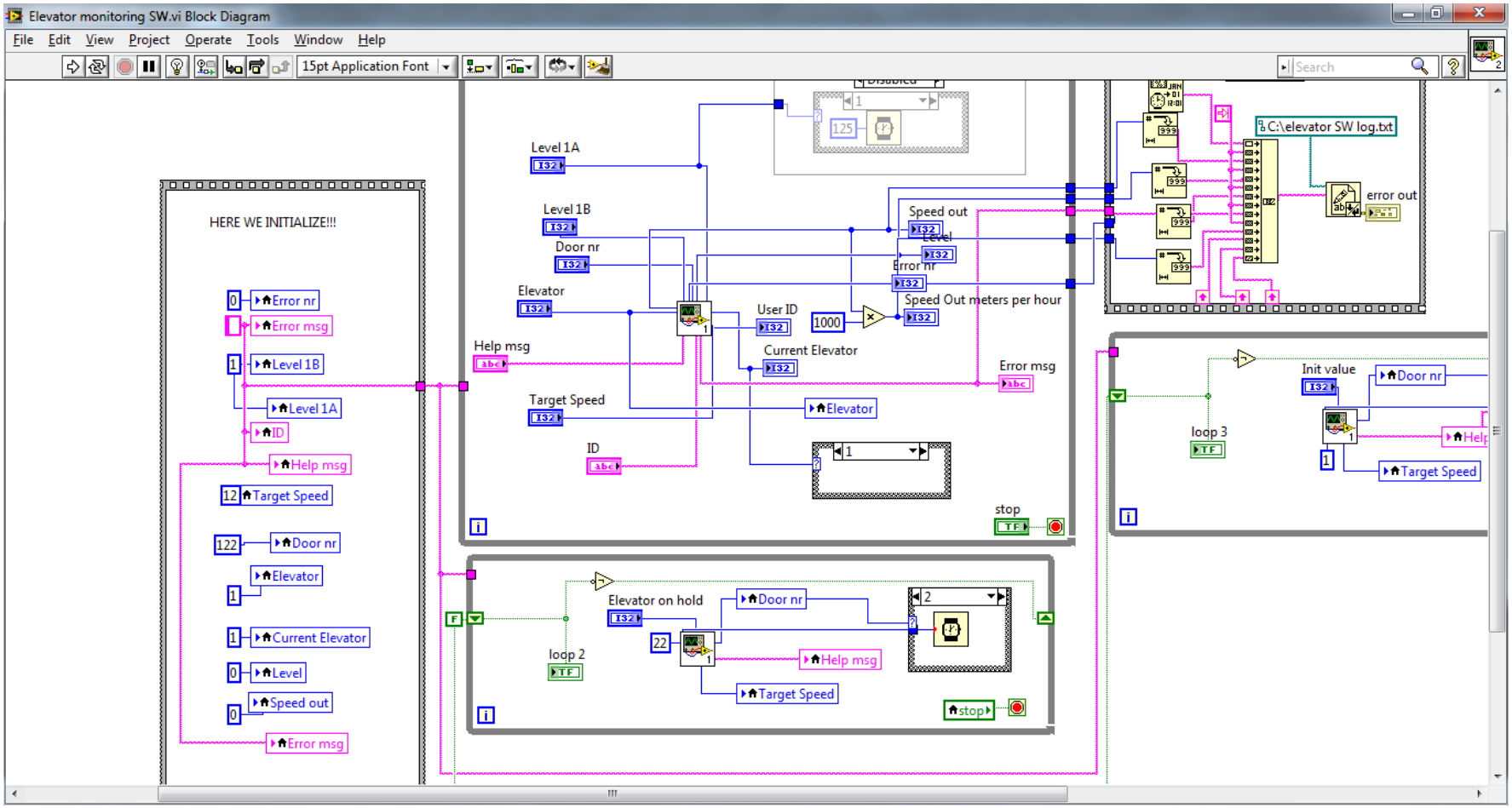
Challenges when developing LabVIEW applications?



Agenda

- How to start programming your LabVIEW Application?
- Common pitfalls
- Project Organization, File Naming and Control
- Structuring your application
- Error Handling
- Race conditions
- What is Next?

Common Pitfalls



LabVIEW Style Guidelines

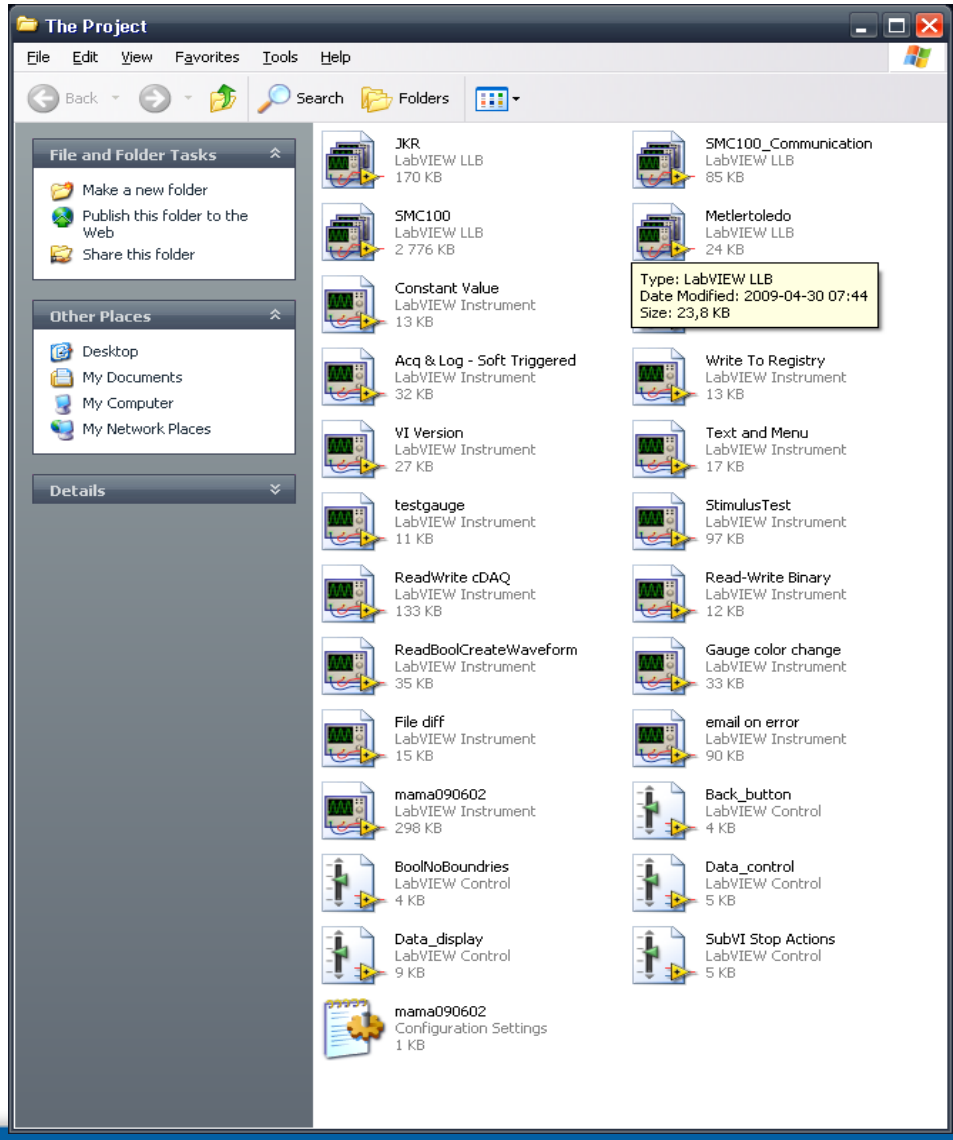
- Each developer has his own style and that can be a problem when multiple developers work on a project
- Inconsistent style makes software difficult to maintain and reuse.
- Select a set of guidelines that works for you and your development team and make sure everyone follows those guidelines.
- **LabVIEW Style Checklist:**
  http://zone.ni.com/reference/en-XX/help/371361G-01/lvdevconcepts/checklist/

Project Organization, File Naming and Control

- Create the Folder Hierarchy (before you start coding)
- Use the LabVIEW Project
- Use proper source filenames:
  - Create unique filenames (never default names)
  - Do not abbreviate filenames
  - Identify the top-level VIs
- Use Source Control (for example: Perforce, Tortoise SVN, MS Visual SourceSafe)

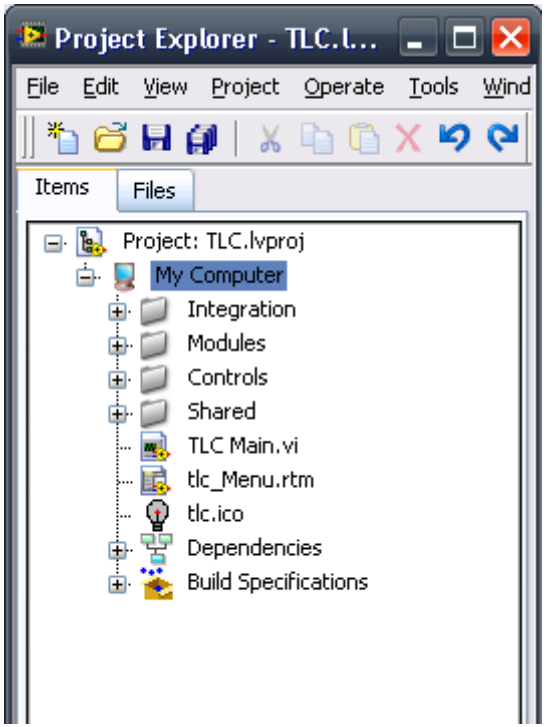
LabVIEW Projects

- Why should I use the Project Explorer?
- Which of these is the start VI?



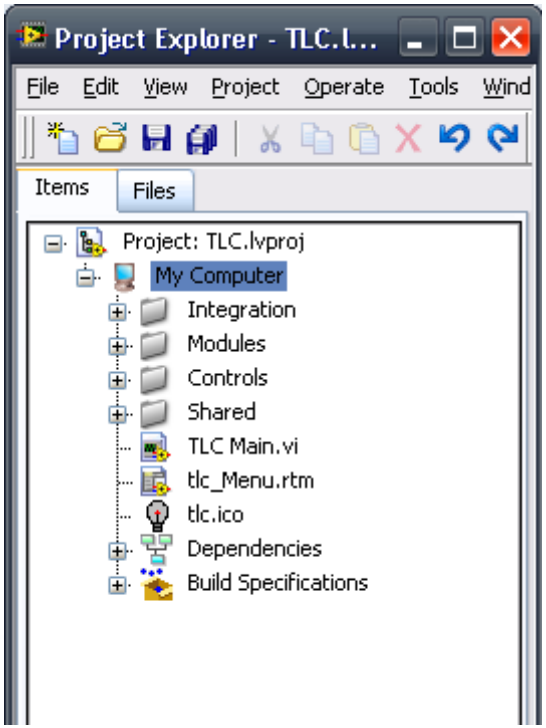
Demo

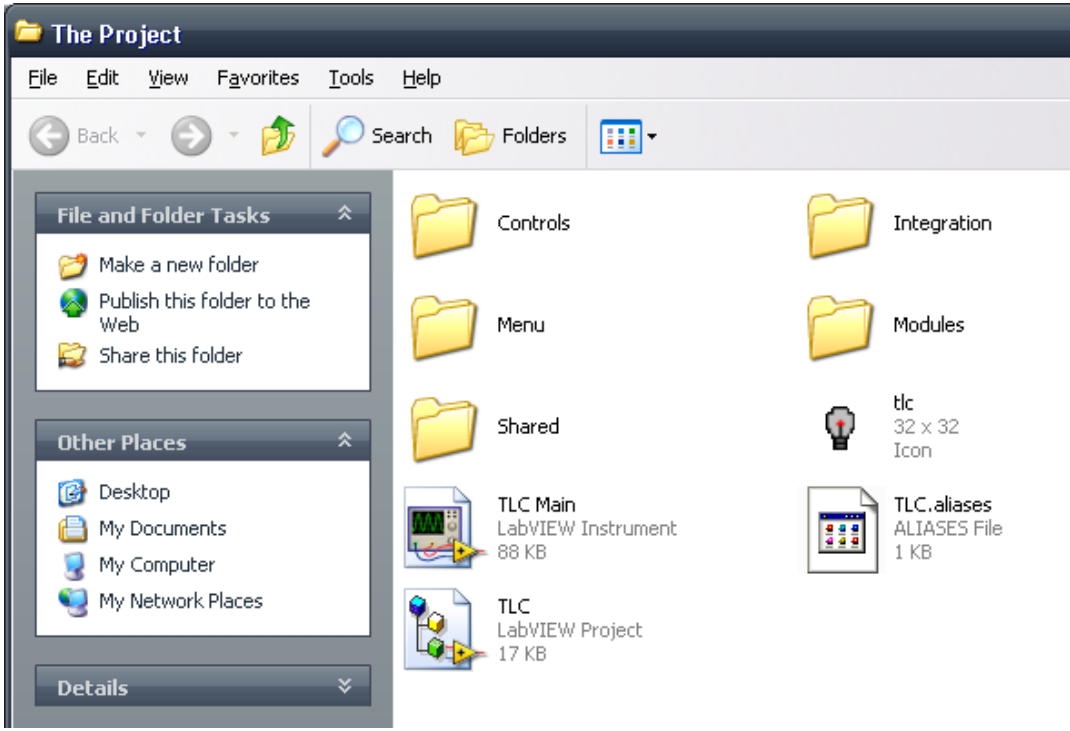
Project Explorer (5)



LabVIEW Projects

- Tip: Try to use the same file structure in your LabVIEW Project and on your hard disk





Why is Structure Important?

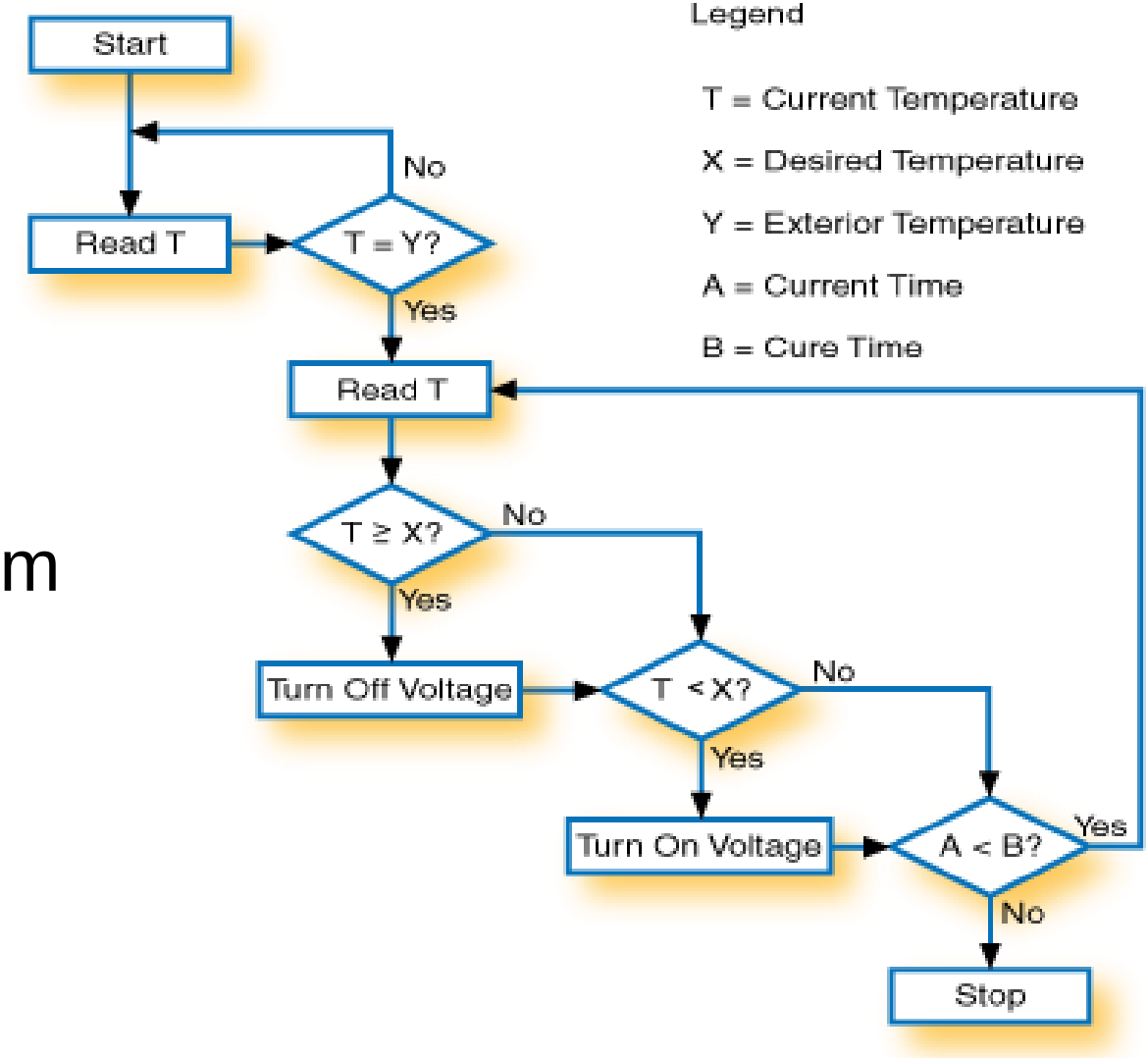
- Think about structure before you start!
  - + Easier to understand, upgrade and maintain
  - + Easier project management
  - + Find problems earlier
  - + Self documenting
- LabVIEW has built in Templates
  - + No need to reinvent the wheel
  - + Get started faster

How to start your main application

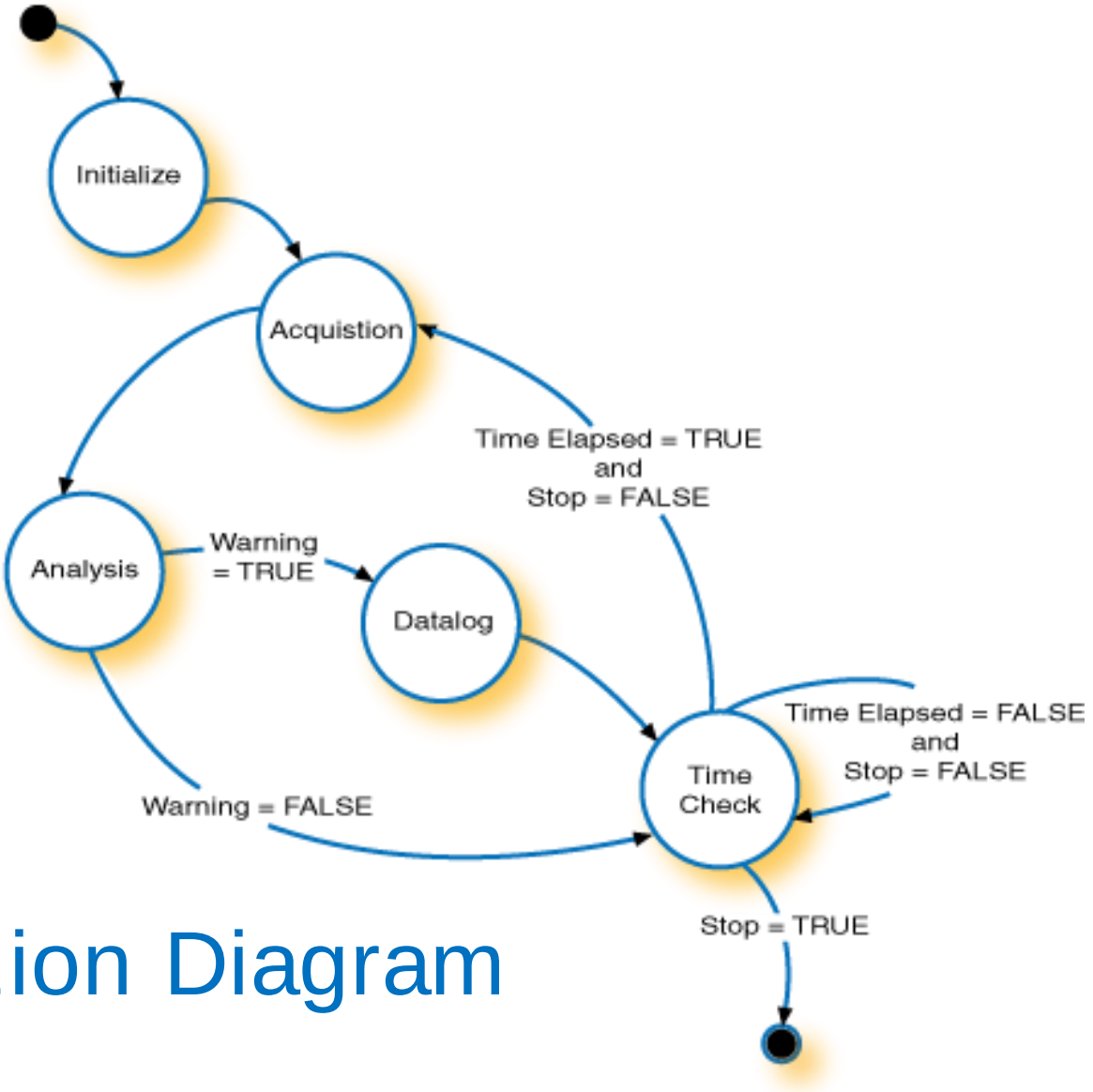
- Start on paper
- Requirements
- Program flow & data flow
  - State transition diagram
  - Flow chart

Flow chart

Visual design
of an algorithm



State Transition Diagram



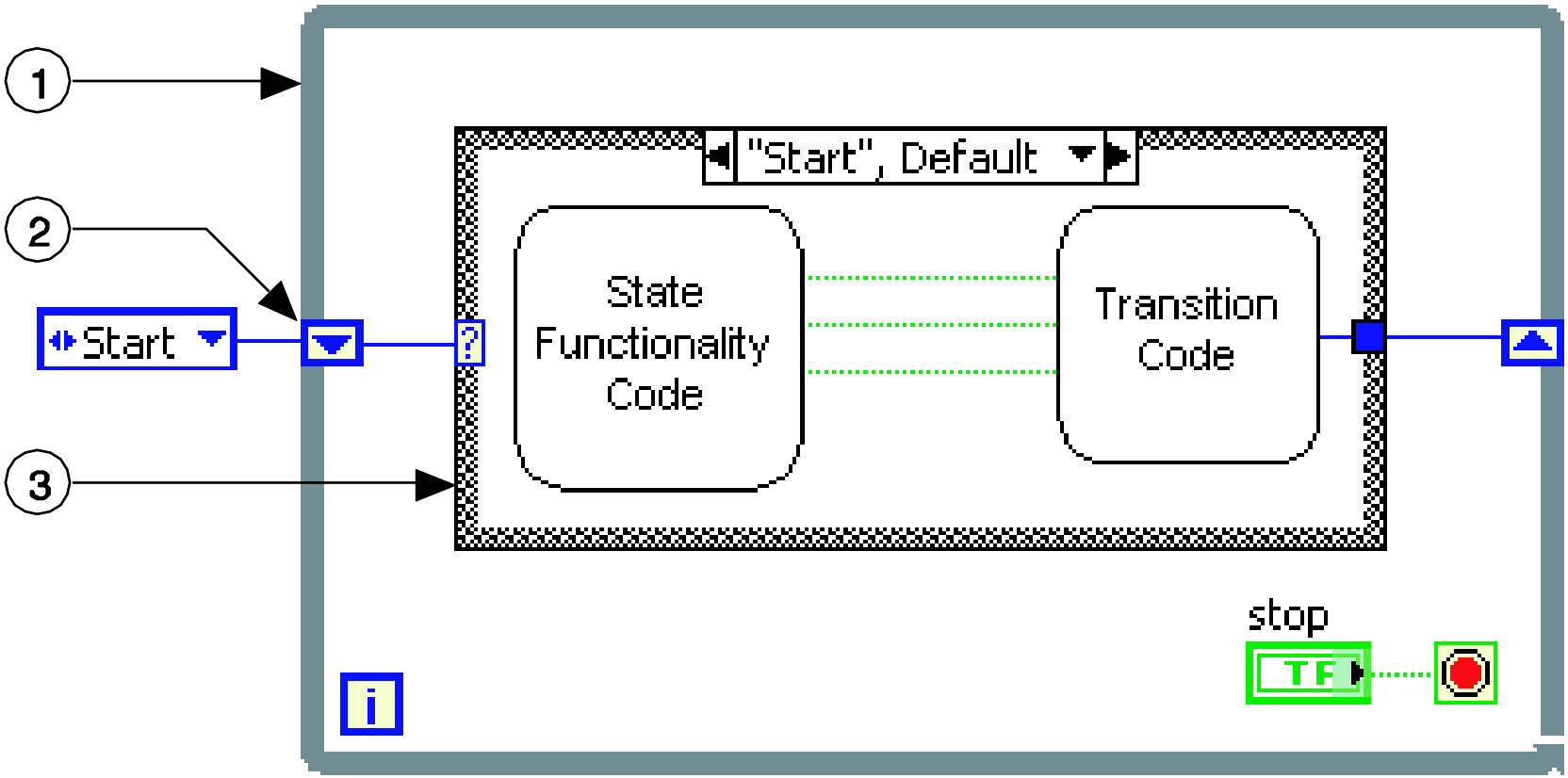
Design Patterns

- State Machine
- Producer/Consumer (Data)
- Producer/Consumer (Events)
- + many more

State Machine Design Pattern

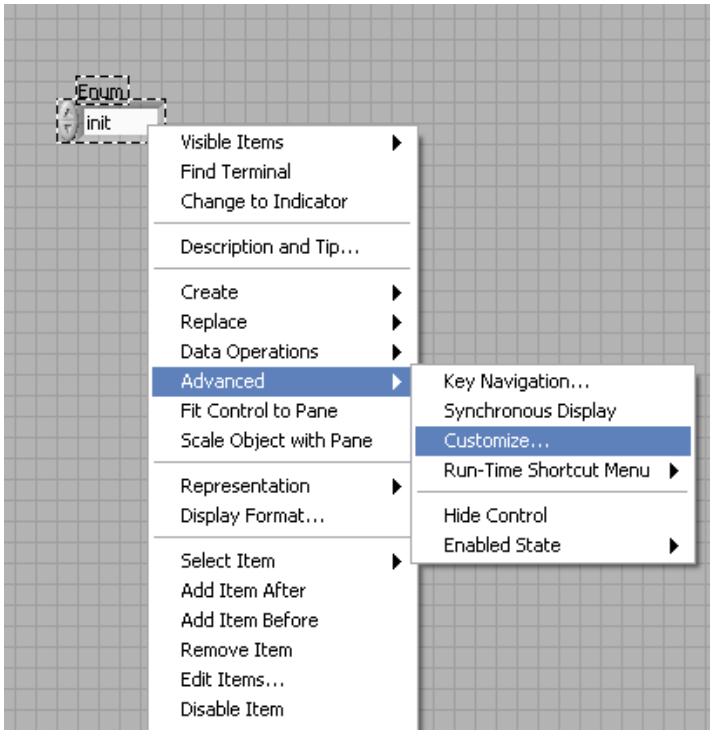
- You are facing these challenges:
  - Set of task to perform in sequence
  - Sequence is not fixed
  - Example: Acquiring data and performing actions based on signal level

State Machine Design Pattern



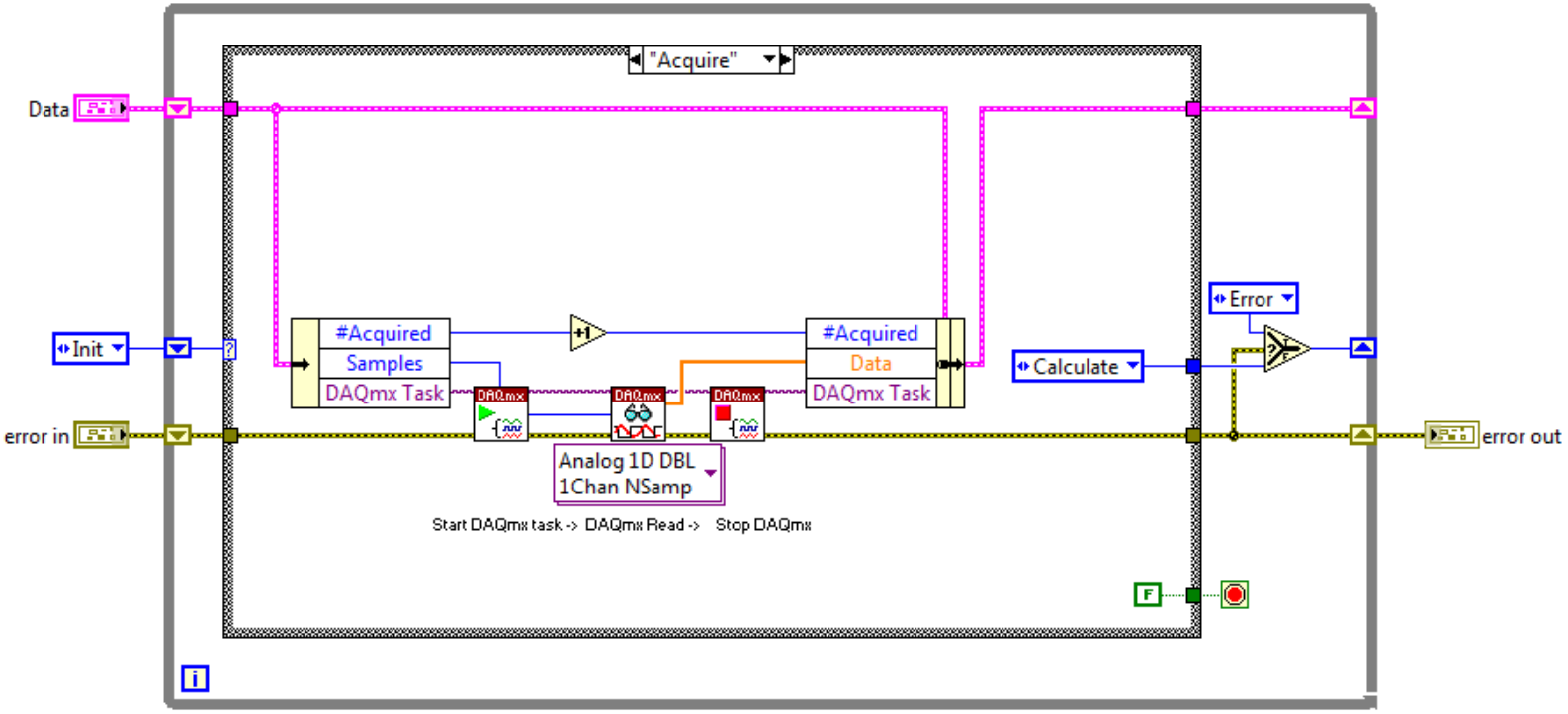
State Machine Design Pattern

- Tip: Type Def the enum used to control the cases in the State Machine.



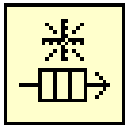
Demo

State machine with Cluster (2)



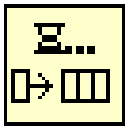
Queues

Obtain Queue



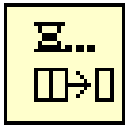
Create a Queue with a datatype and a name (optional)

Enqueue Element



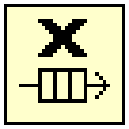
Put a piece of data last in the queue

Dequeue Element



Take the first piece of data from the queue

Release Queue

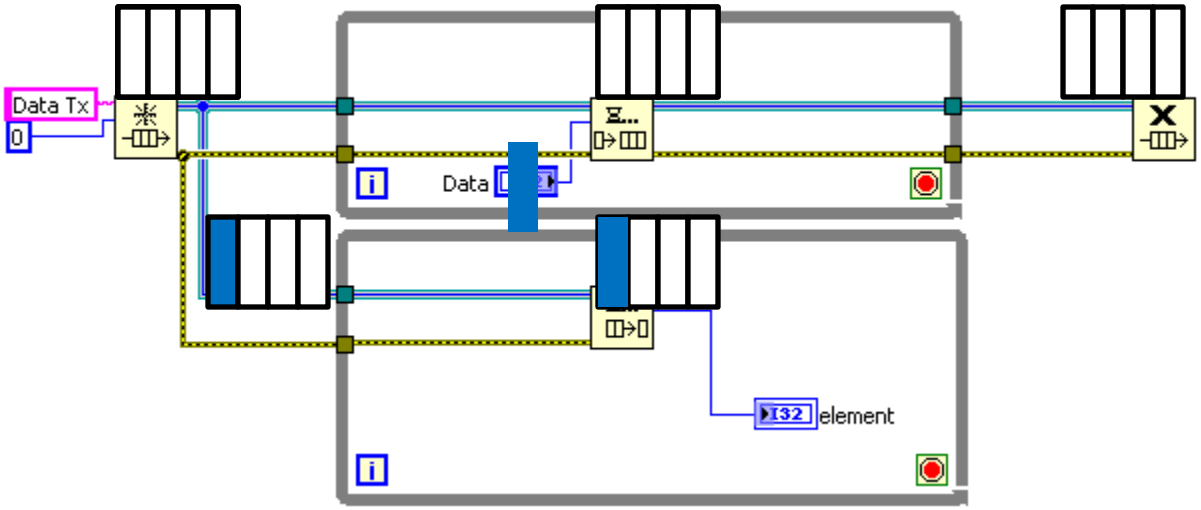


Release the queue to destroy it and free up the memory

- Works just like a queue at the supermarket
  - First in, First out
  - Buffers data, allows different rates of Enqueue and Dequeue
  - Can be used to transfer data between loops in same VI or between VIs

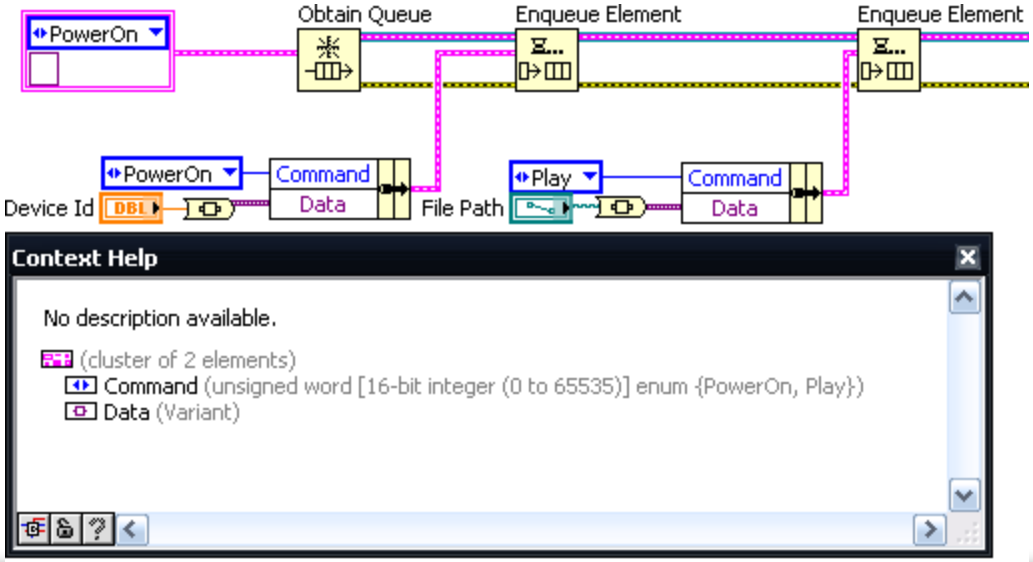
Queues

- Normal Use and Behind the Scenes



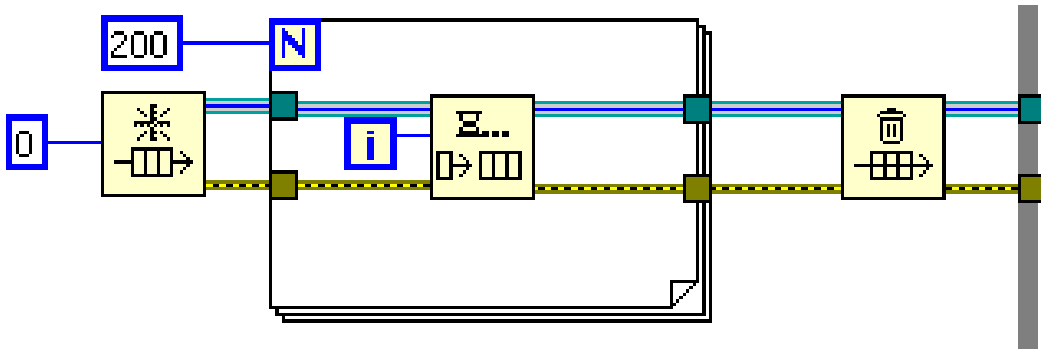
Queues

- Tip: Use cluster with Enum and Variant to get multifunction queue
  - Enum is command that explains datatype/action
  - Variant contains data of any type



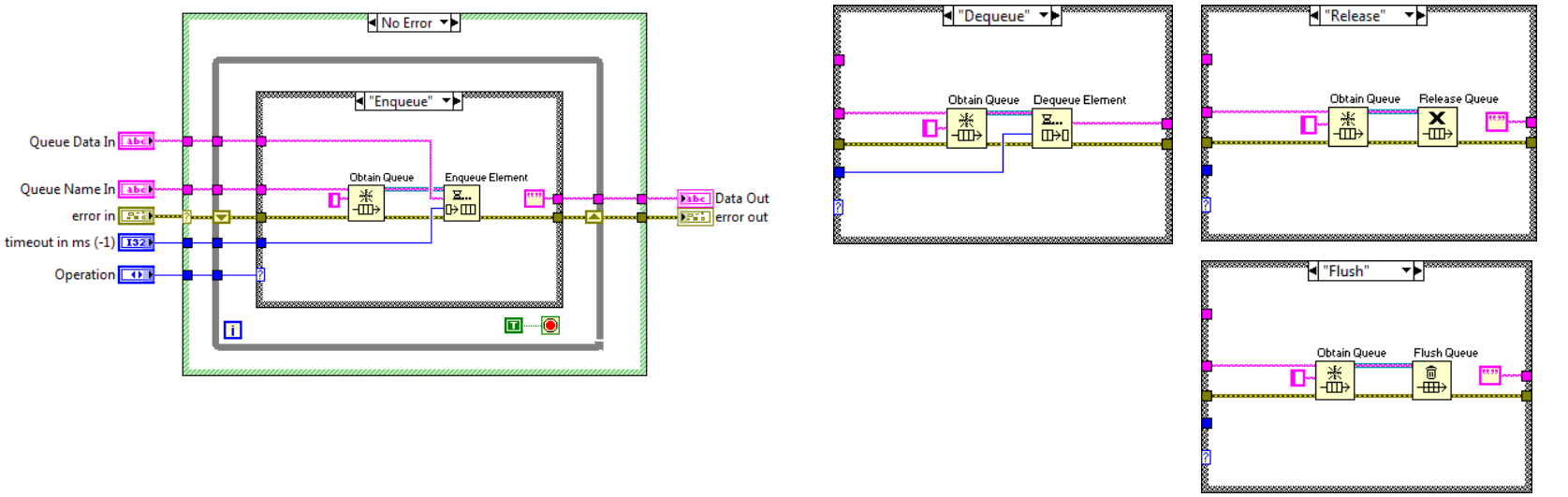
Queues

- Tip: For time critical applications pre-allocate queue size for best performance



Queues

- Tip: You can create a single VI to handle all Queue operations without using reference wires in the main program.



Producer/Consumer (Data)

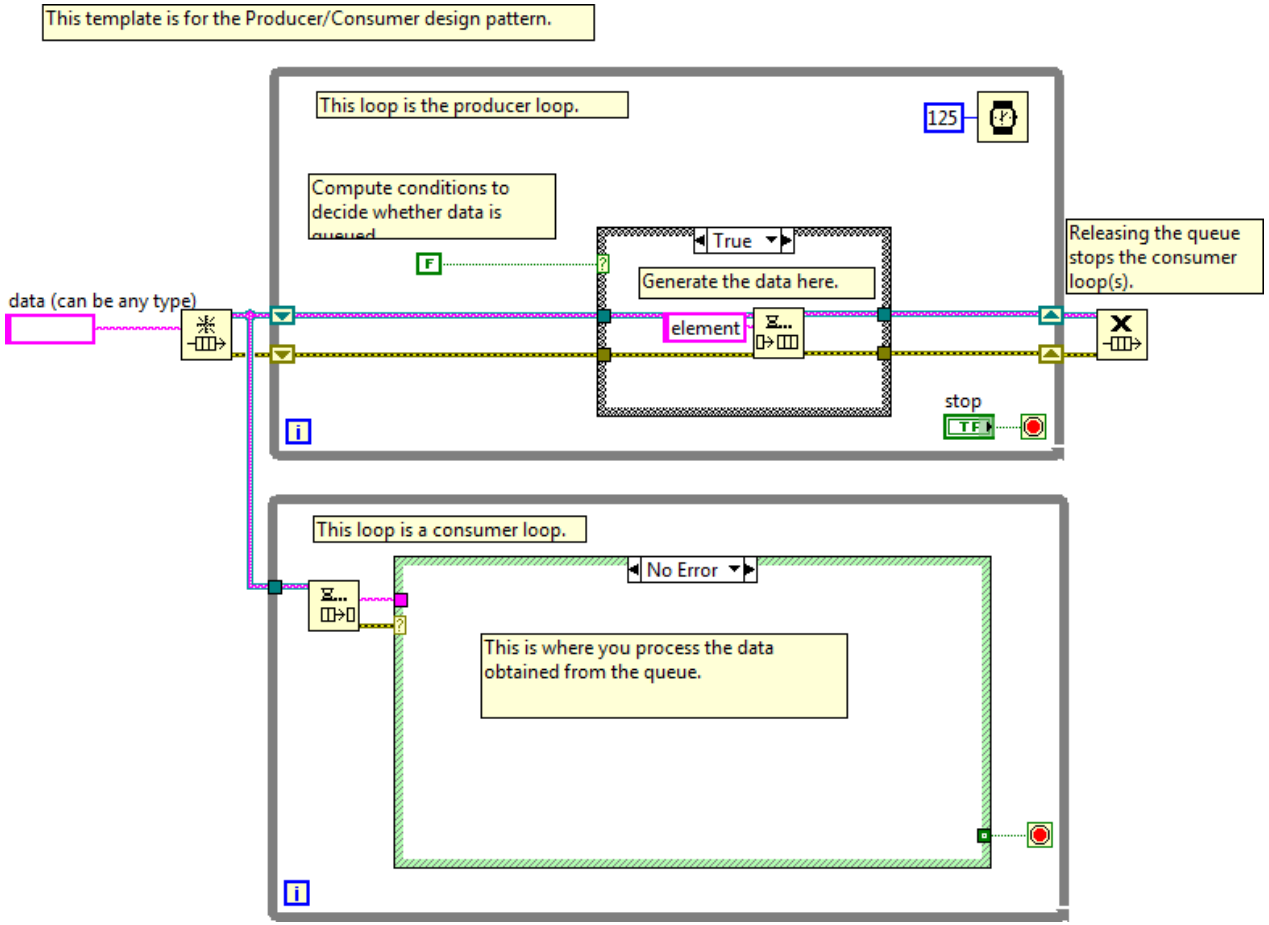
- You are facing these challenges:
  - Need to do intensive processing
  - Amount of code too much for one loop, takes too long to execute
  - Different parts in the program need to execute at different speeds
  - Can't afford to lose data

  - **Example**: Time critical data acqusition with intensive analysis

Producer/Consumer (Data)

- Enhanced data sharing between multiple loops running at different rates
- Two categories of processes; those that produce data and those that consume data
- Use when you need to acquire multiple sets of data that must be processed in order

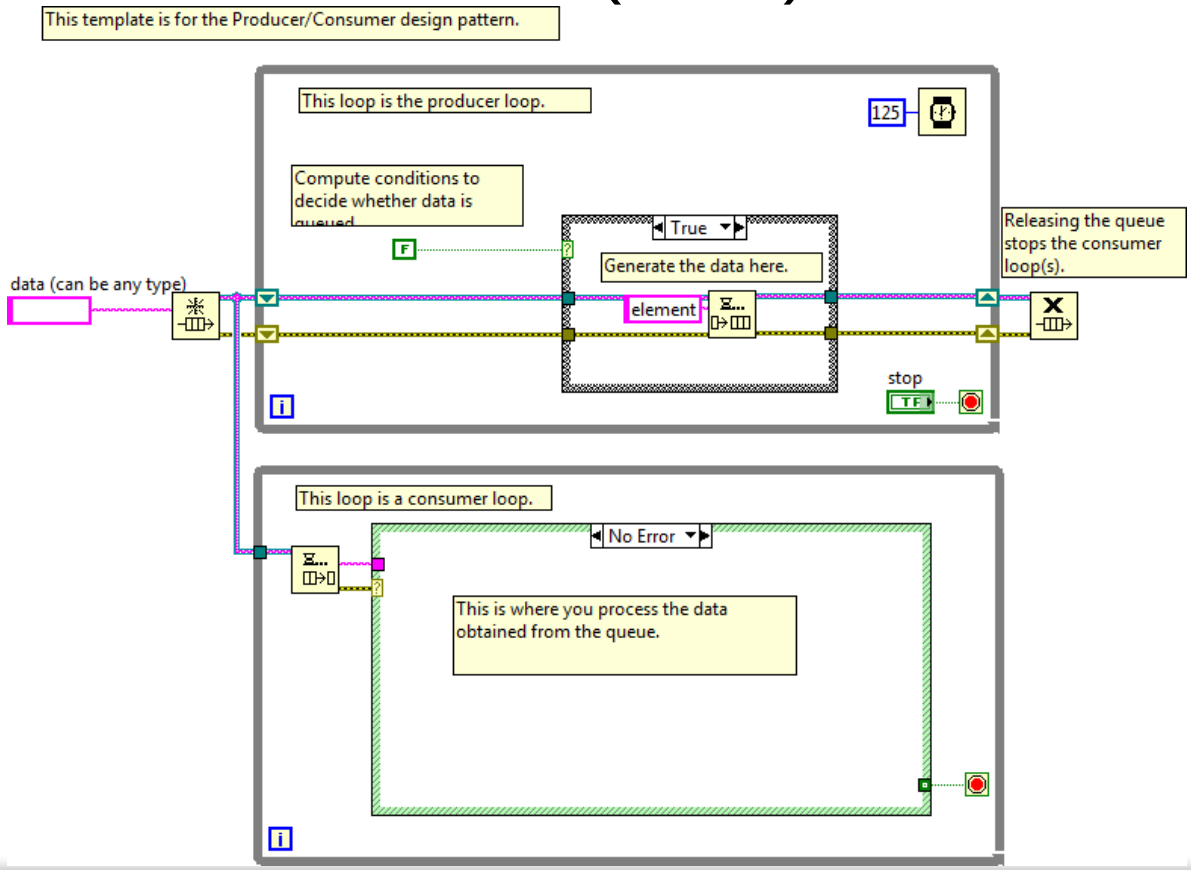
Producer/Consumer (Data) Design Pattern



- Benefits
  - Acquire multiple sets of data that must be processed in order
  - Queues data, no data loss
- Considerations
  - Timing the producer
  - Sending data from the consumer to the producer

Demo (template)

Producer/Consumer (Data)



Demo (Weather Station UI)

Producer/Consumer (Events)

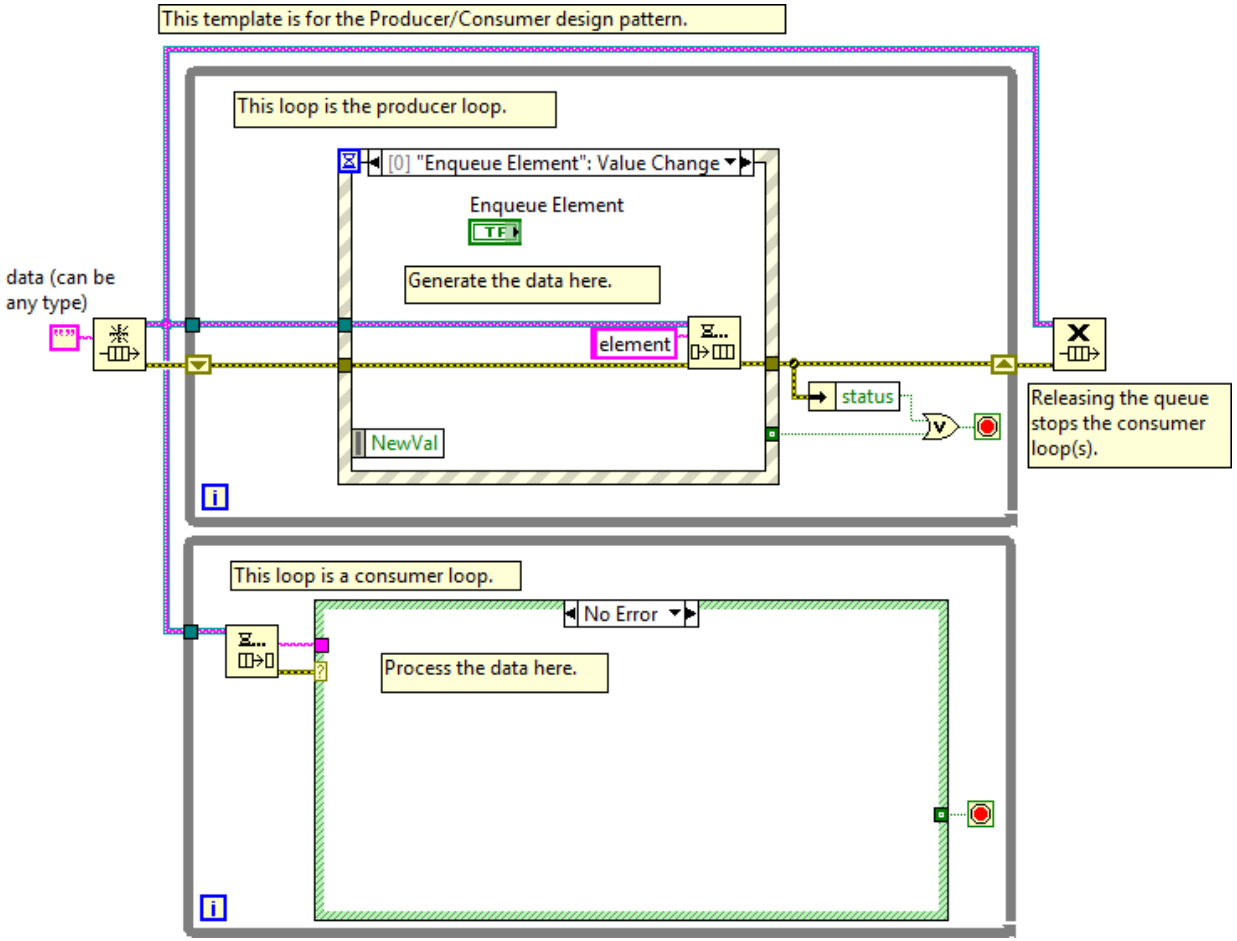
- You are facing these challenges:
  - User controls application via GUI, no set sequence
  - Heavy data processing required
  - GUI needs to respond directly at all times

  - Example: Any heavy processing, user interactive application

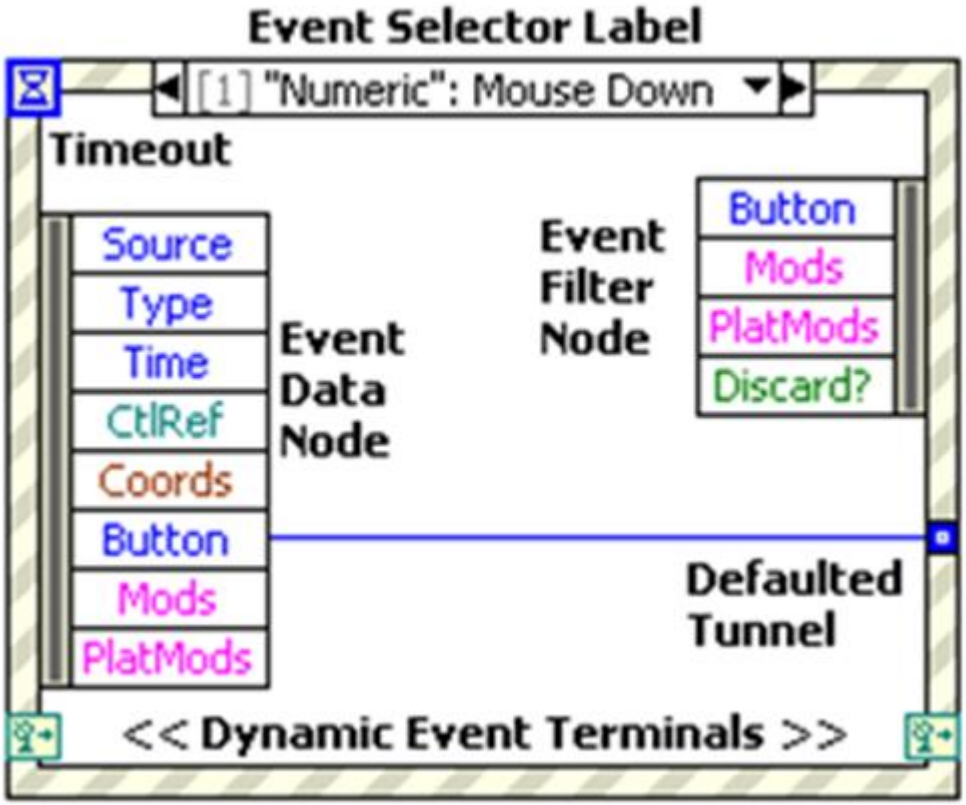
Producer/Consumer (Events)

Benefits

- Efficiently responds asynchronously to the user interface
- Queues can transfer any data type

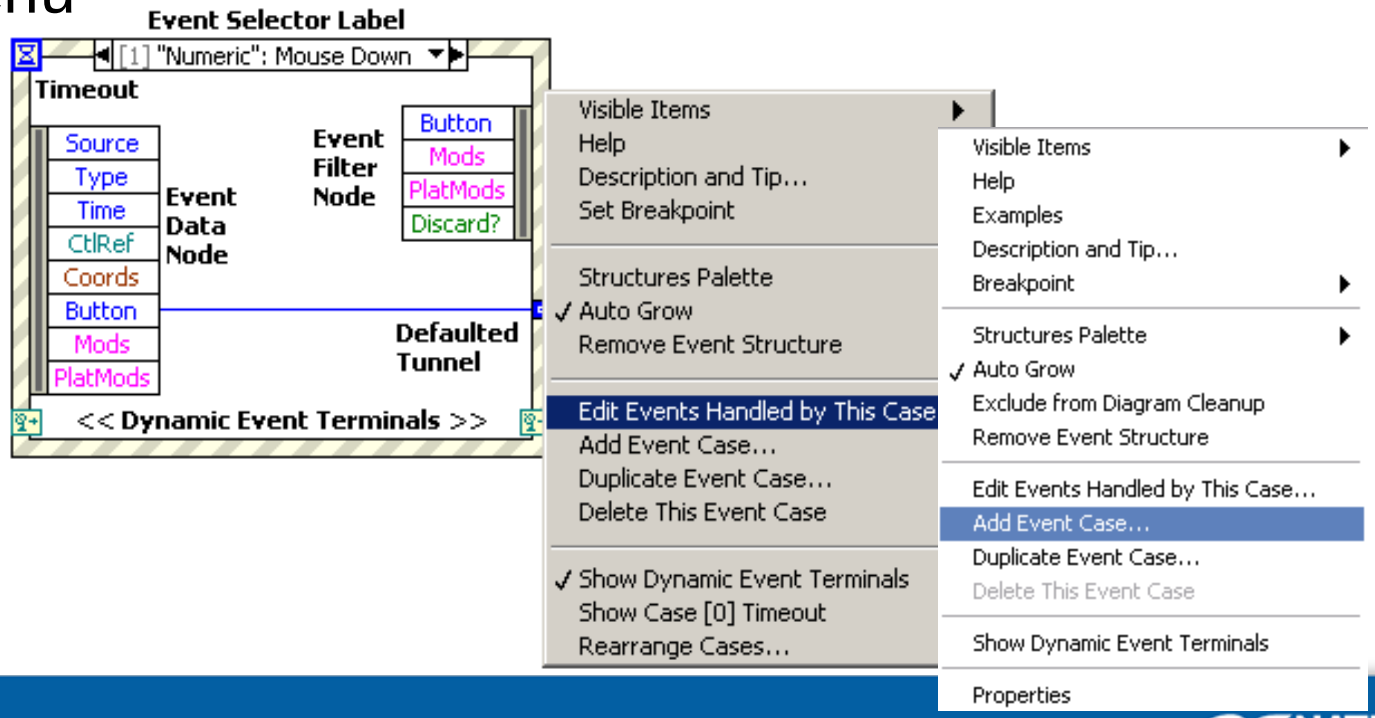


Events



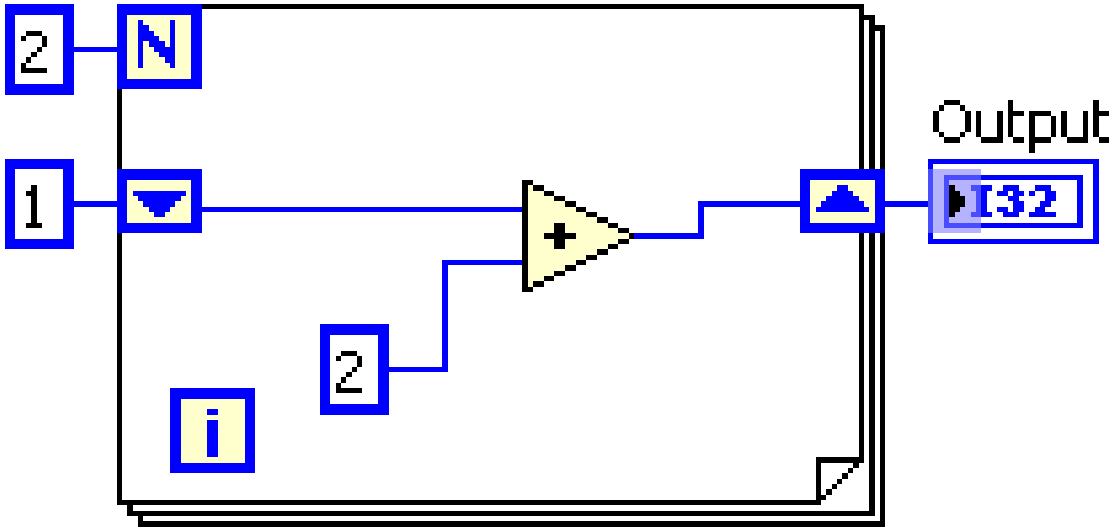
Events - Configuration

- Use a dialog box to configure events by right-clicking the Event structure border and selecting **Add Event** or **Edit Events Handled by This Case** from the shortcut menu



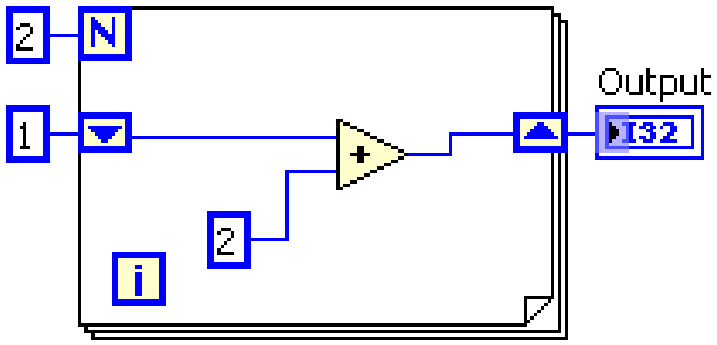
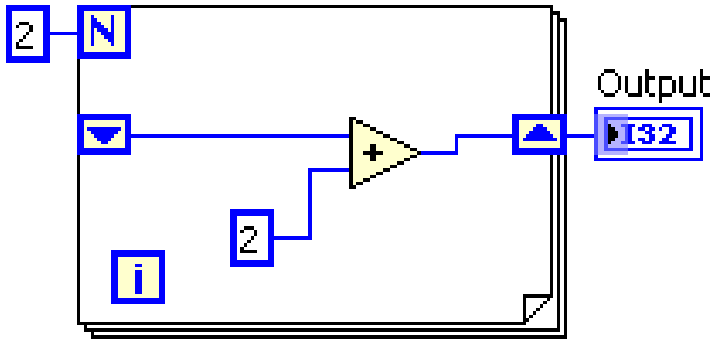
Shift Registers

- When using loops, you often need to remember data from previous iterations
- Shift registers transfer values from one loop iteration to the next



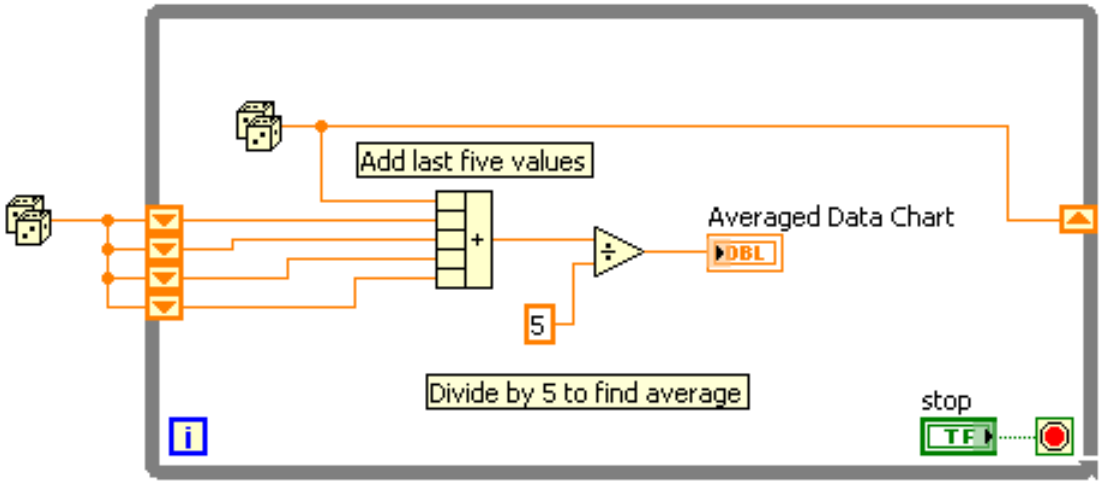
Shift Registers - Initializing

Run once → VI finishes → Run again

Block Diagram	1st run	2nd run
Initialized Shift Register 	Output = 5	Output = 5
Not Initialized Shift Register 	Output = 4	Output = 8

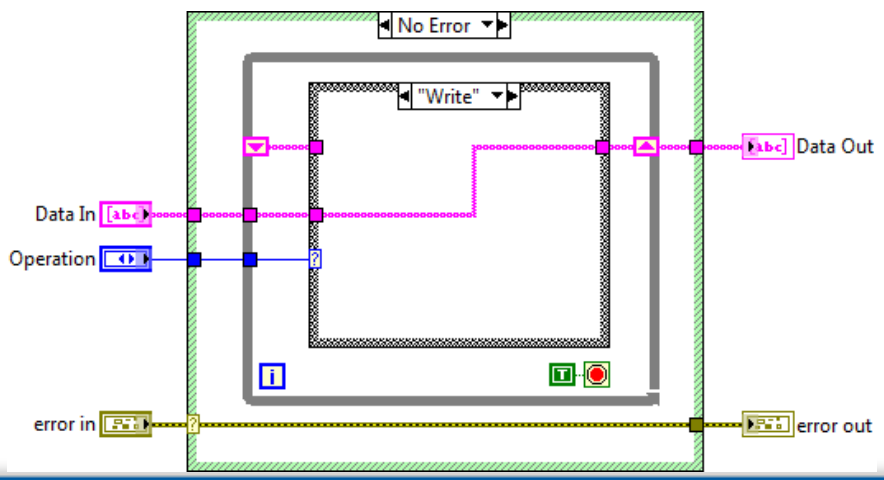
Stacked Shift Registers

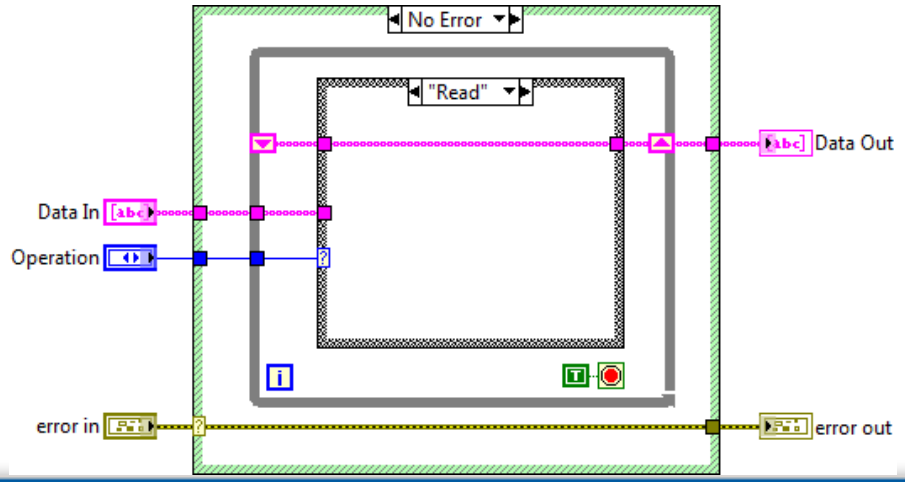
- Stacked shift registers remember values from multiple previous iterations
- Right-click the left shift register and select **Add Element**



Functional Global Variable (FGV)

- Fast
- More memory efficient and reliable than Local or Global Variables
- Expanded Functionality possible



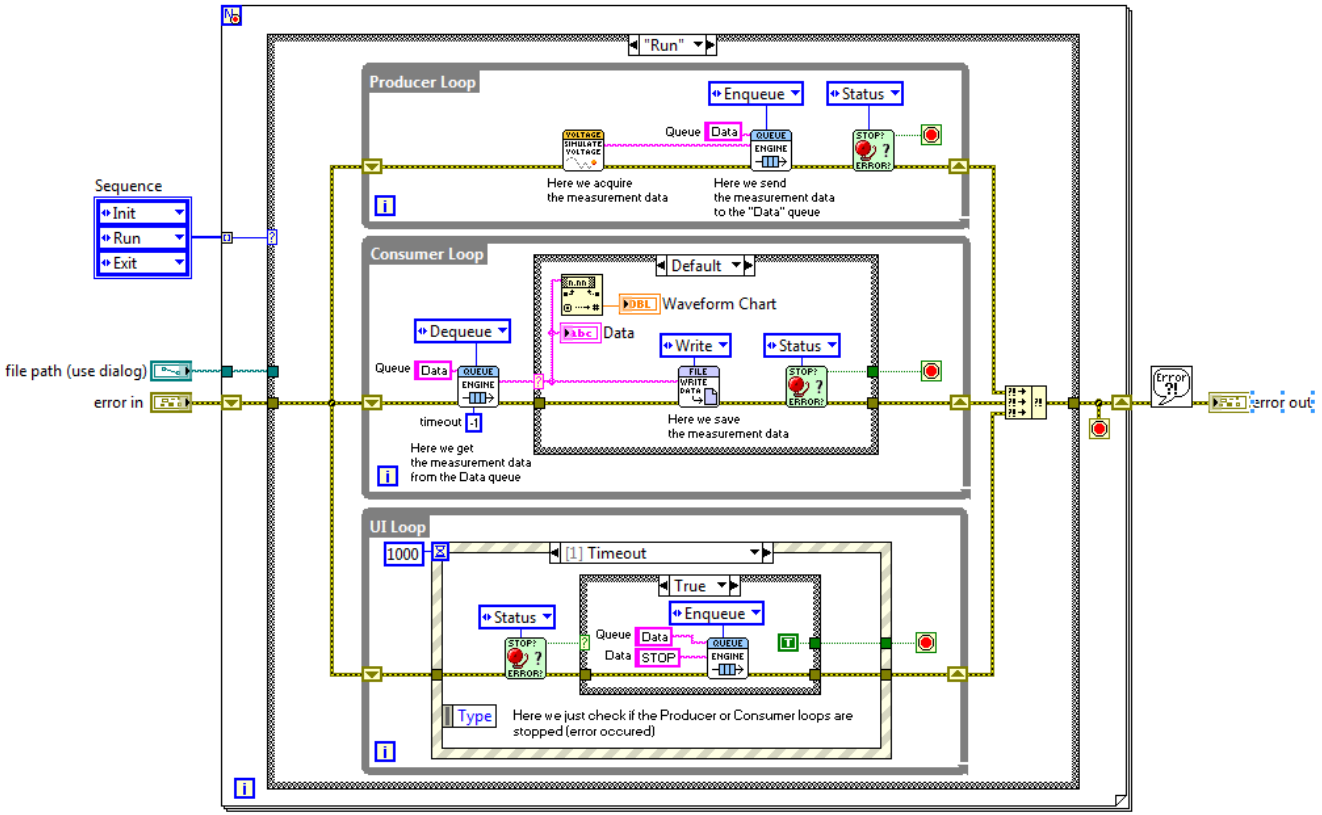


Demo - Functional Global Variable

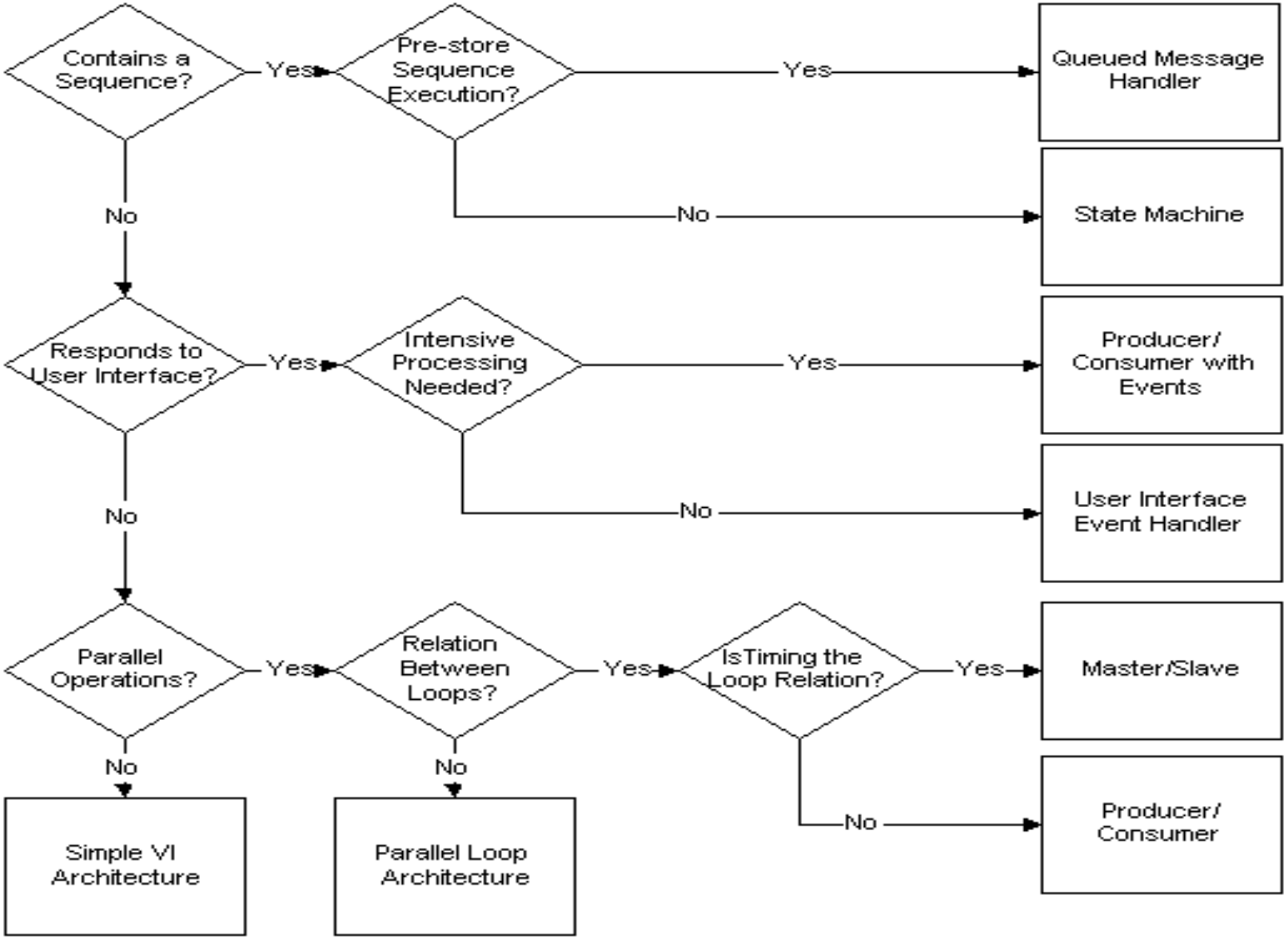
Timer

Demo

Producer/Consumer/User Interface (6)



Choosing the best Design Pattern

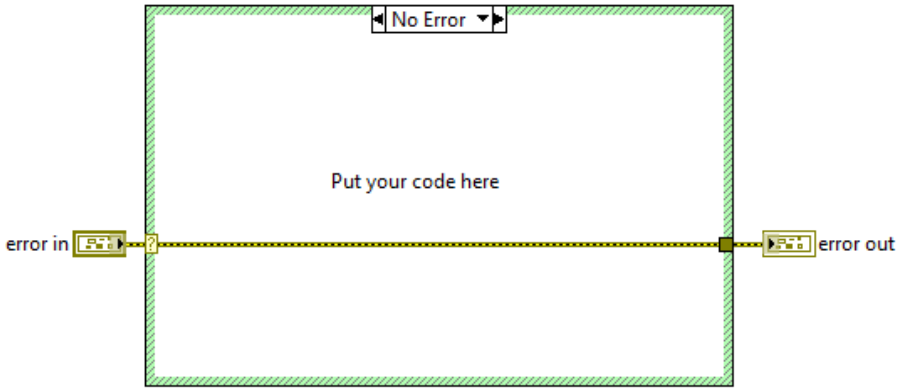


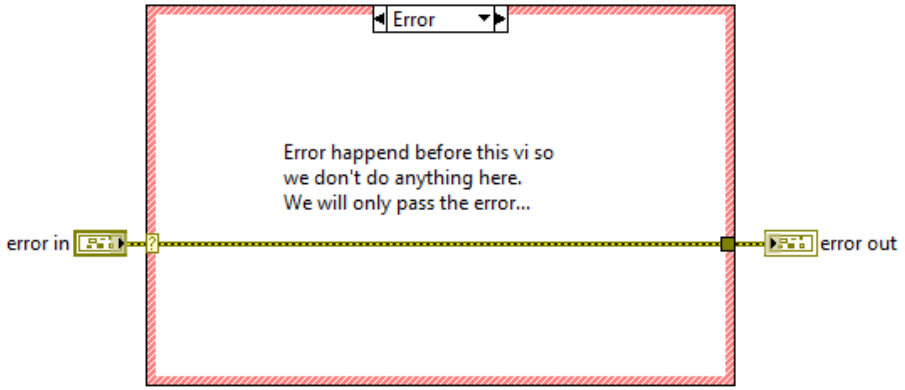
Error Handling 

- LabVIEW applications execute as programmed and therefor are not immune to bugs.
- Error handling is essential part of any application.
- Debugging LabVIEW application without proper error handling is same as programming blindfolded.
- So how to handle errors?

Error Handling

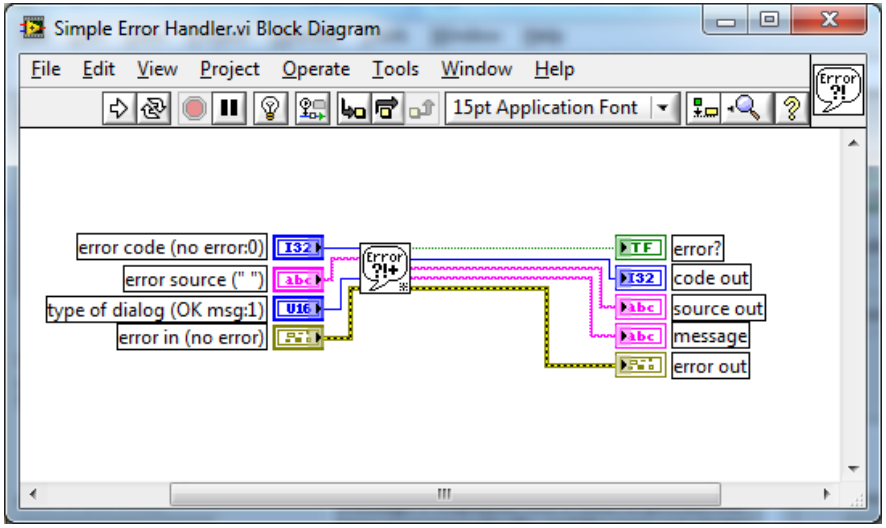
- Use the ”No Error / Error” case with most of your subVIs!





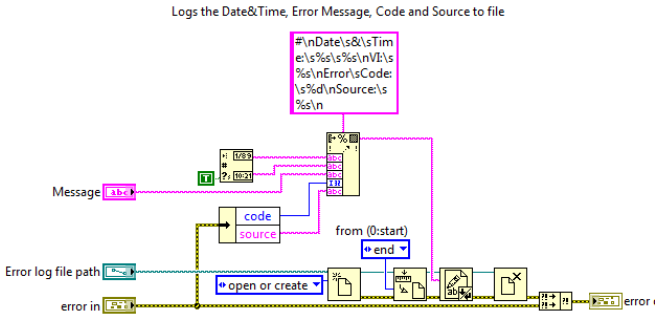
Error Handling

- Use General Error Handler VI instead of Simple Error Handler VI
  - Simple Error Handler VI is just caling General Error Handler VI and hiding some of it's inputs.



Error Handling

- Log all Errors
  - Users may not report every error to developers
  - Even the minor errors may indicate bugs in the code. For example ini-file may be in the wrong place and user has to always seek the right path.
  - Errors may happen within specific time so there might be a memory leakage or some other problem that log file could help to discover.

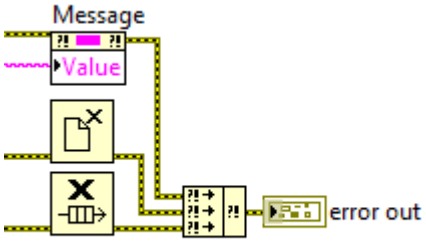


Error Handling

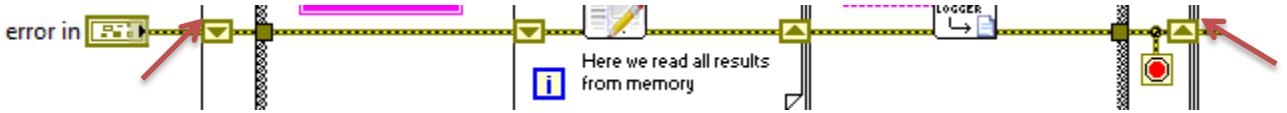
- Can I create custom Error Codes? Yes you can! And you should!
- Error Codes in the range of -8999 to -8000 and 5000 to 9999 are reserved to user-defined errors
- Maintain user-defined error codes within an XML file
- Use Error Code File Editor:
  - **Tools -> Advanced -> Edit Error Codes**

Error Handling

- Use Merge Errors function for parallel execution



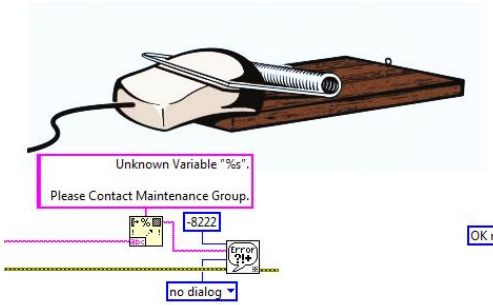
- Use shift registers in loops with Error cluster
  - For loop with 0 iterations will clear the Error (if shift register is not used)!



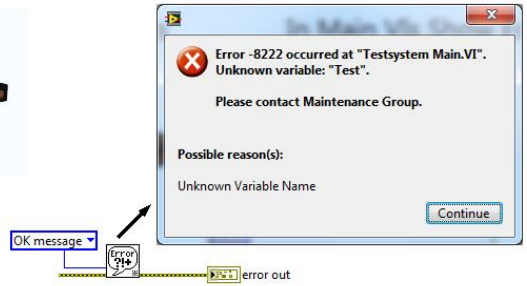
Error Handling

- Do not popup Error Window in lower level VIs. Only acknowledge errors there and pass that data forward with error cluster to the Main VI level.
- Use Error Popups in the Main VIs to show user where the error happend and why.

In subVIs Only Trap Errors



In Main VIs Show Error Dialog

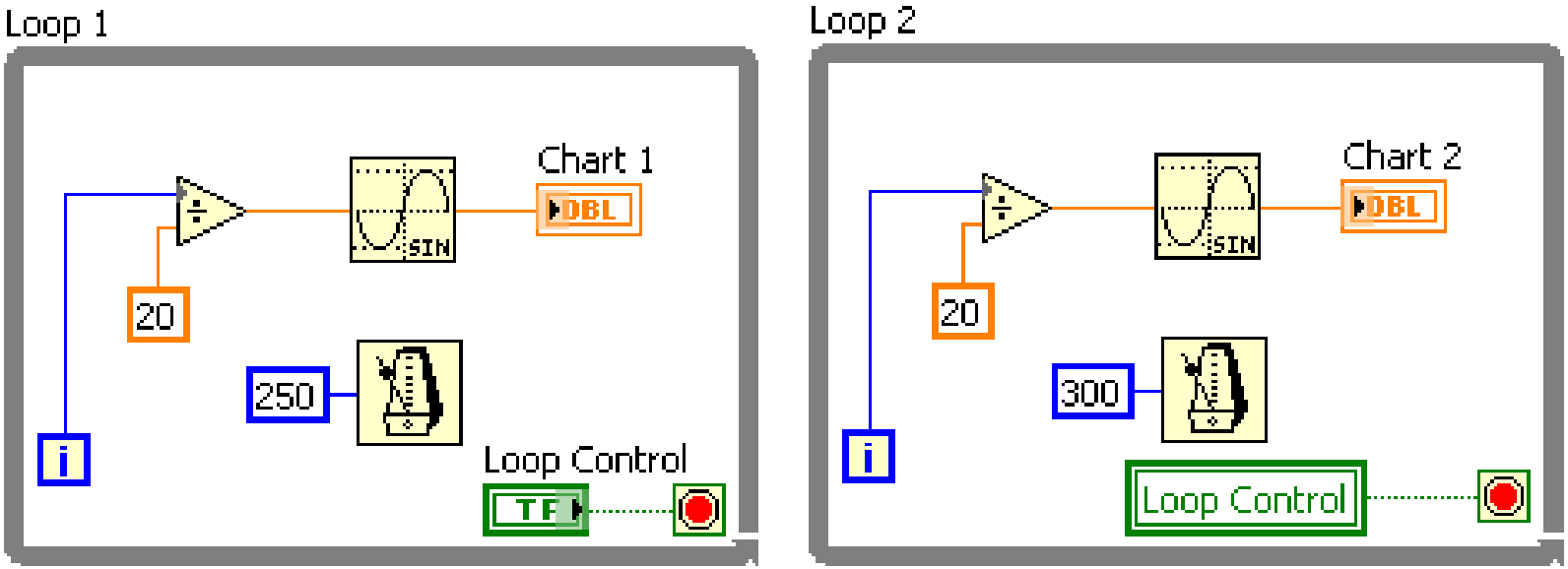


Local Variables

- Local Variables make no use of the standard Data Flow principle
- Ideal for situation when wires cannot be used, for example between separate loops
- Do not overuse! Shift Registers are more effective
- More advanced cases use Queues or Notifiers instead

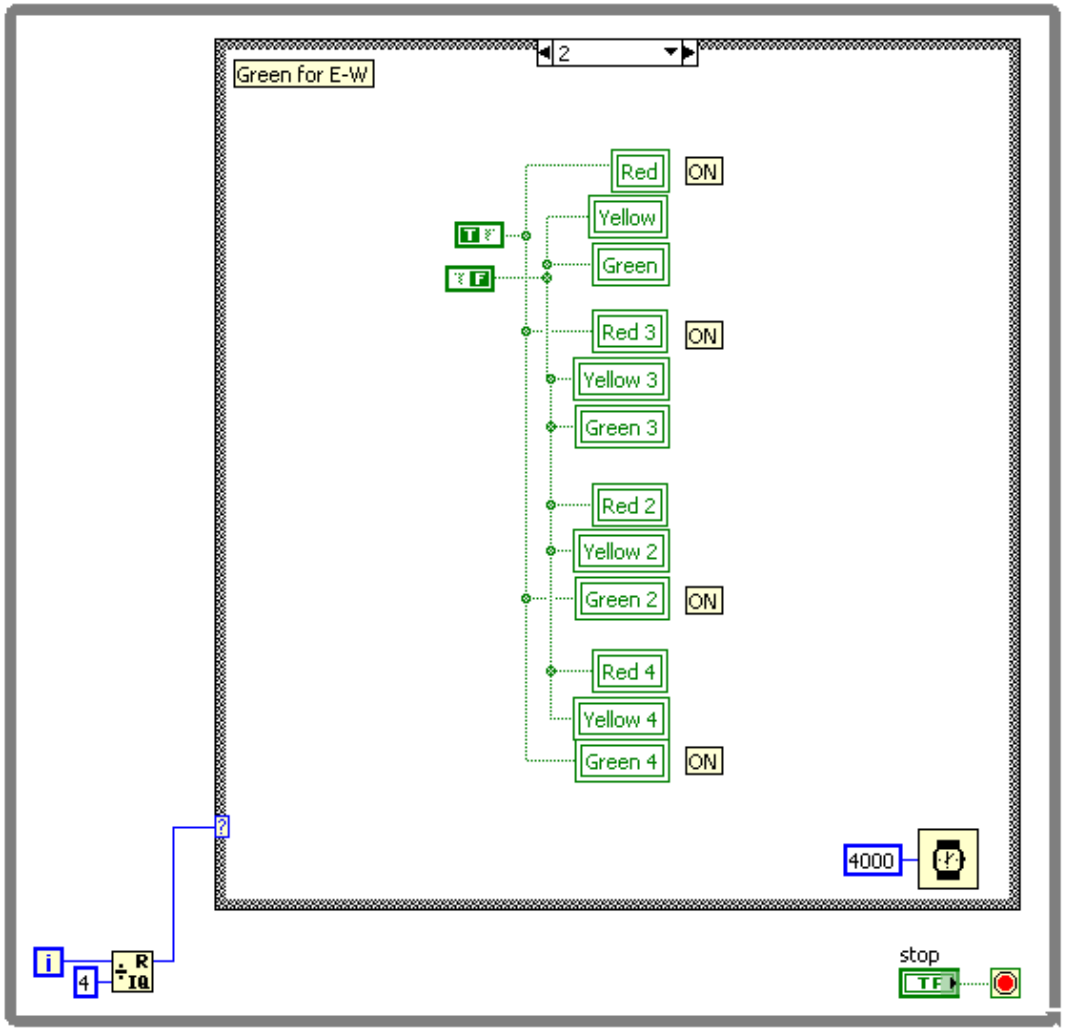
Local Variables

- Good use case: Use local variables to pass stop command between parallel loops



Local Variables

- Bad use case: Main Architecture
- Use Cluster and Shift Registers instead!
- Do not forget to initialize



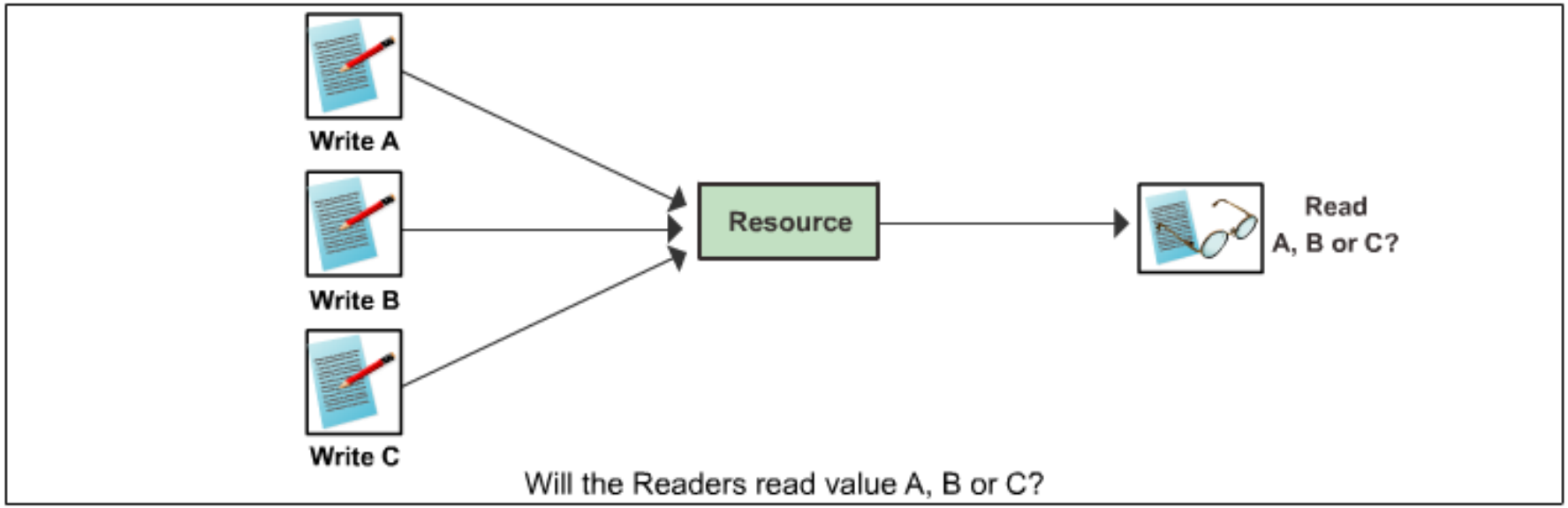
Race Conditions

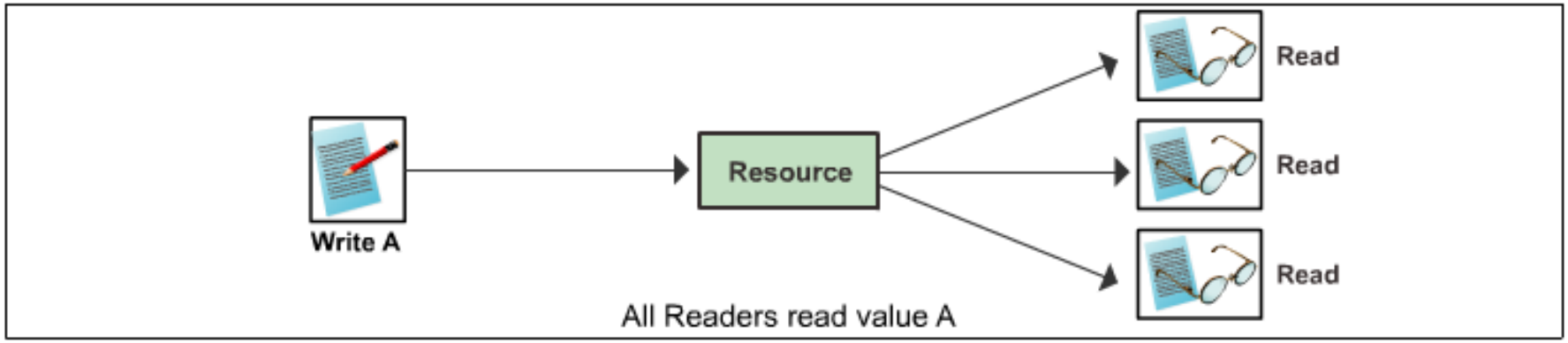
- A race condition is a situation where the timing of events or the scheduling of tasks may unintentionally affect an output or data value
- Race conditions are a common problem for programs that execute multiple tasks in parallel and share data between the tasks

Race Conditions

- Race conditions are very difficult to identify and debug
- Often, code with a race condition can return the same result thousands of times in testing, but still be capable of returning a different result
- Avoid race conditions by:
  - Controlling shared resources
  - Properly sequencing instructions
  - Identifying and protecting critical sections within your code
  - Reducing use of variables

Race Conditions – Shared Resources

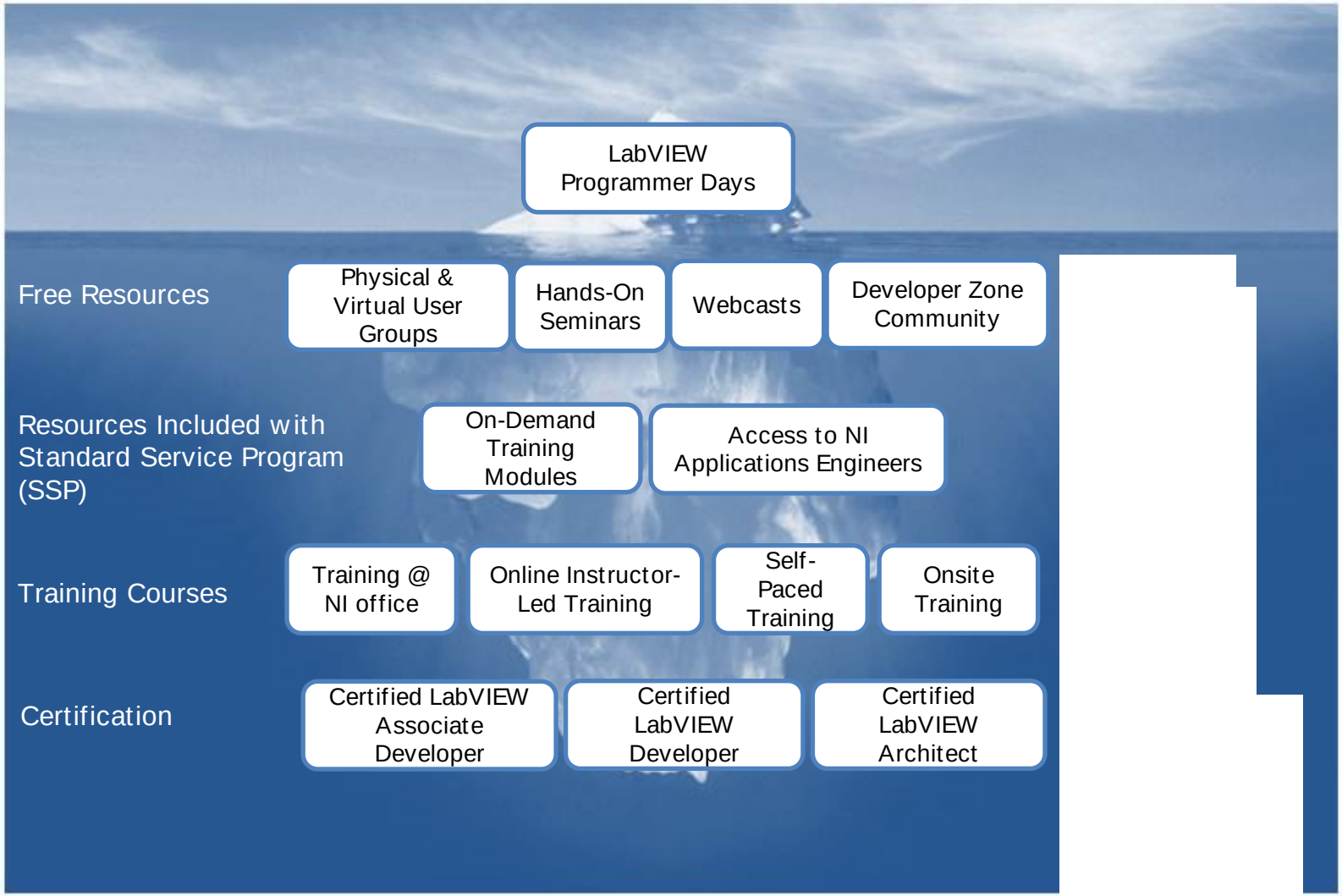




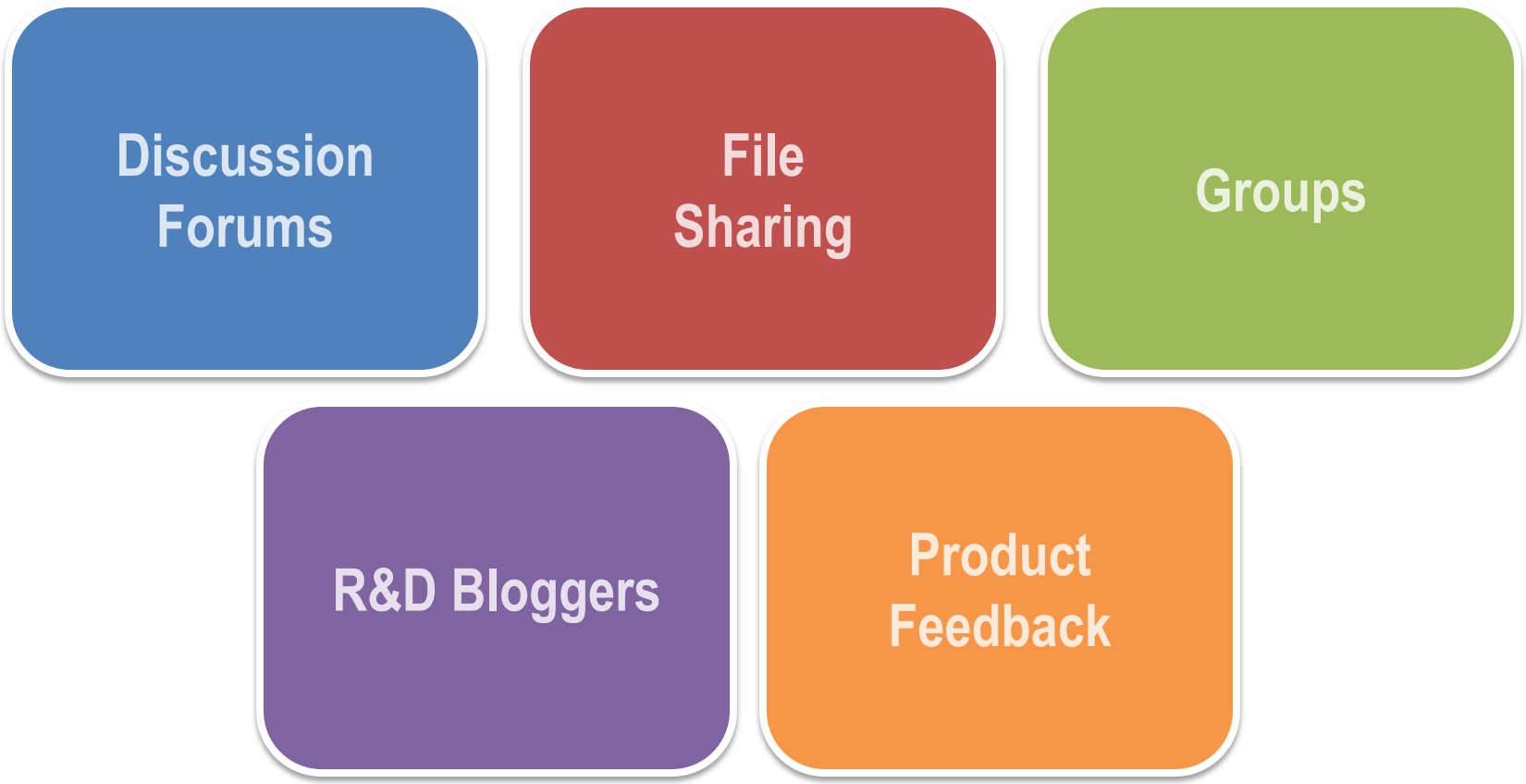
Demo – Race Conditions

What is Next?

LabVIEW Programmer Days: Just the Tip of the Learning Iceberg



Online LabVIEW Community



ni.com/community

The Power of the Platform

Community

110,000+ online members
350,000+ support posts nine languages
300+ user groups worldwide
2,000+ robotics experts through FIRST
400,000+ children through LEGO
1,000+ job postings online

Collaboration

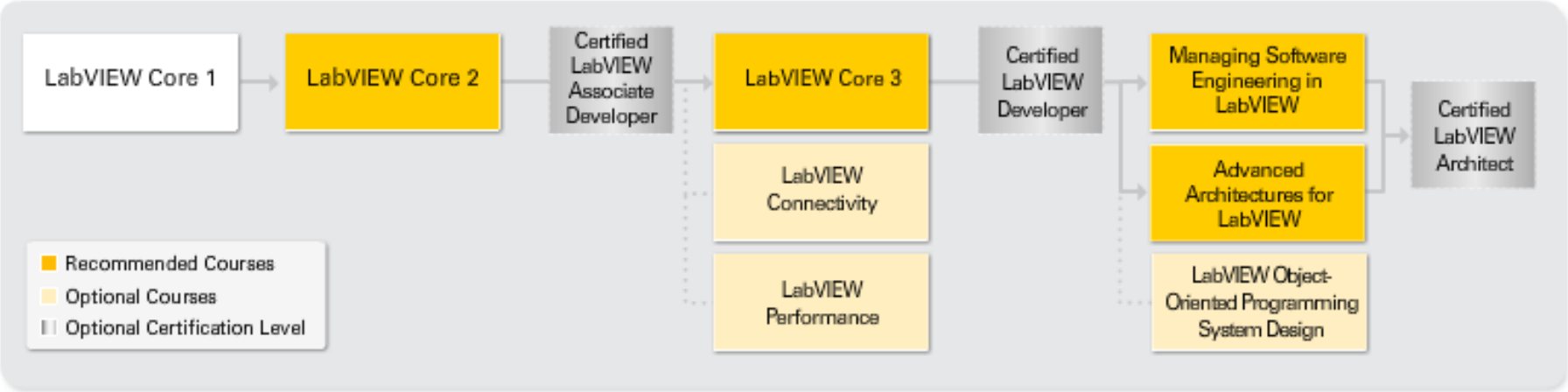
400+ third-party add-ons
400+ solution partners
1,000+ value added resellers
35+ classroom training courses
15+ online training courses

NATIONAL INSTRUMENTS LabVIEW™

Connectivity

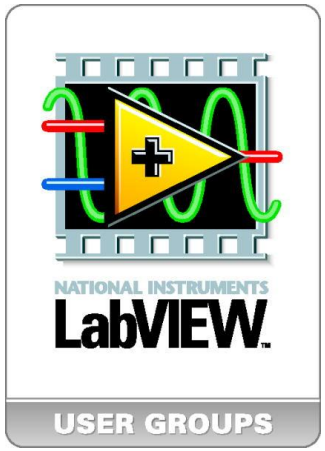
6,000+ example files
7,000+ instrument drivers
1,000+ motion drives
1,000+ smart sensors
500+ industrial cameras
1,000+ Third-party PAC devices

LabVIEW Courses



User Groups

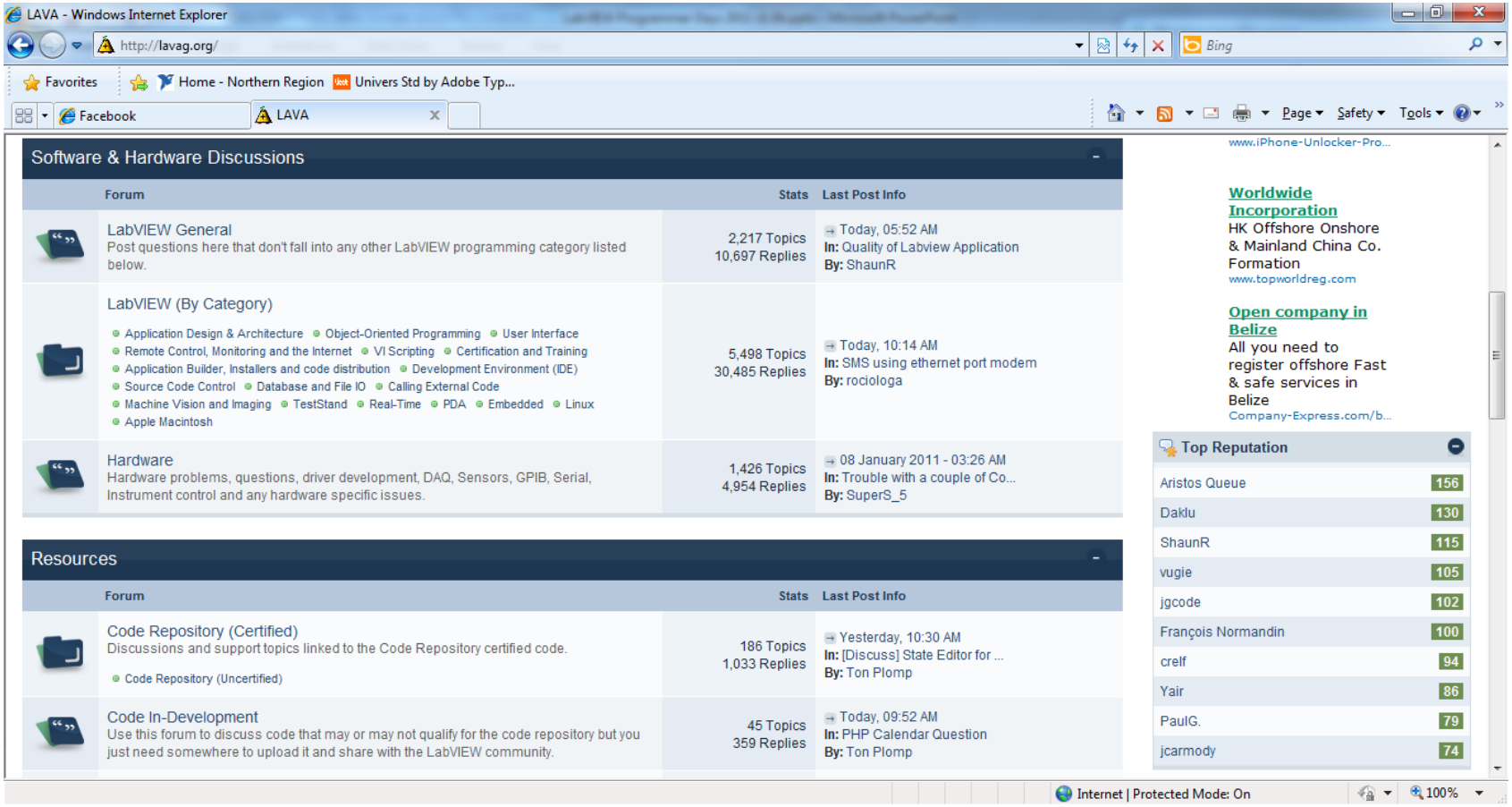
- Browse existing user groups
- Explore user group content
- Host your user group on ni.com
  - Share presentations
  - Display a group calendar
  - Communicate with other members
  - Read meeting notes



ni.com/usergroups

LAVA
LabVIEW Advanced Virtual Architects

http://lavag.org/



Links of interested:

- Developers Zone:
  http://zone.ni.com/
- NI Support
  http://www.ni.com/Support
- Alliance Program
  http://www.ni.com/alliance/
- Training
  http://www.ni.com/training
- LabVIEW Fundamentals Exam
  http://www.ni.com/training/labview_exam.htm