

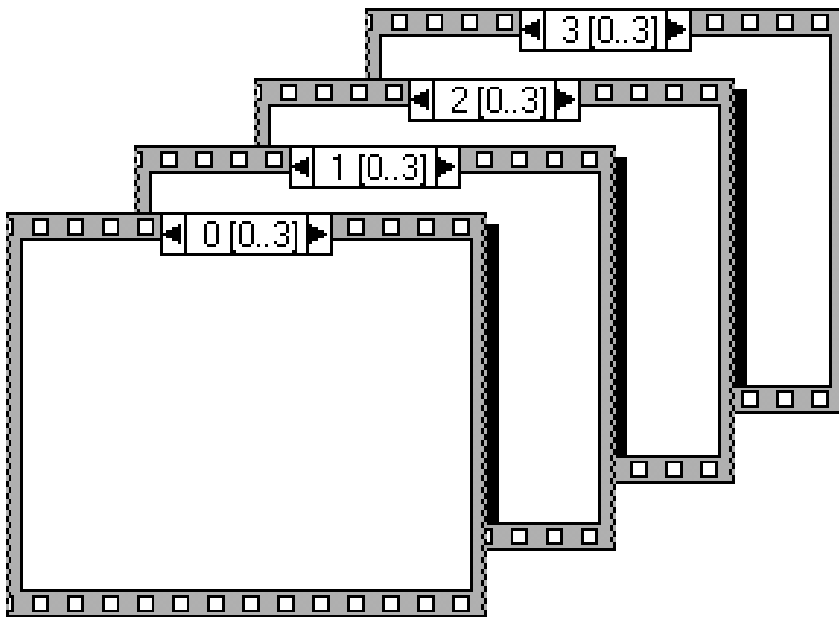
Lecture 3-3

Loops and Charts,
Arrays, Graphs, Clusters,
Case, Sequence Structures,
Local, Global Variables, and
Property Nodes
in Graphical Programming Language

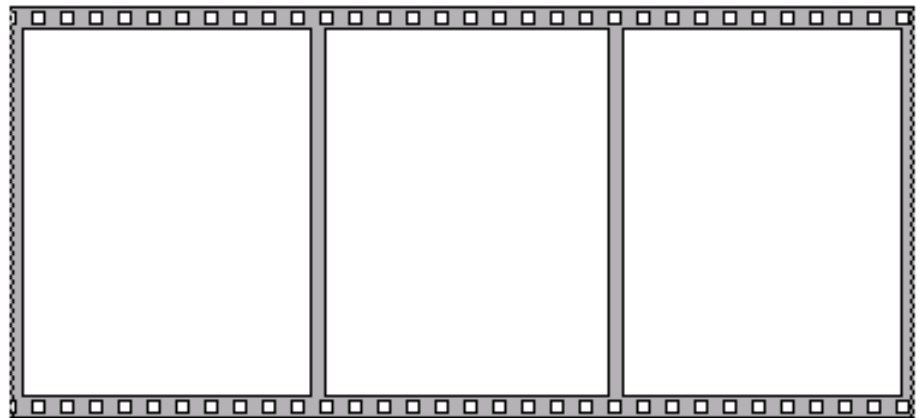
吳文中

Sequence Structures

- In the Structures subpalette of Functions palette
- Executes diagrams sequentially, Frame 0 (0 . . **x**), where **x** is the total number of frames



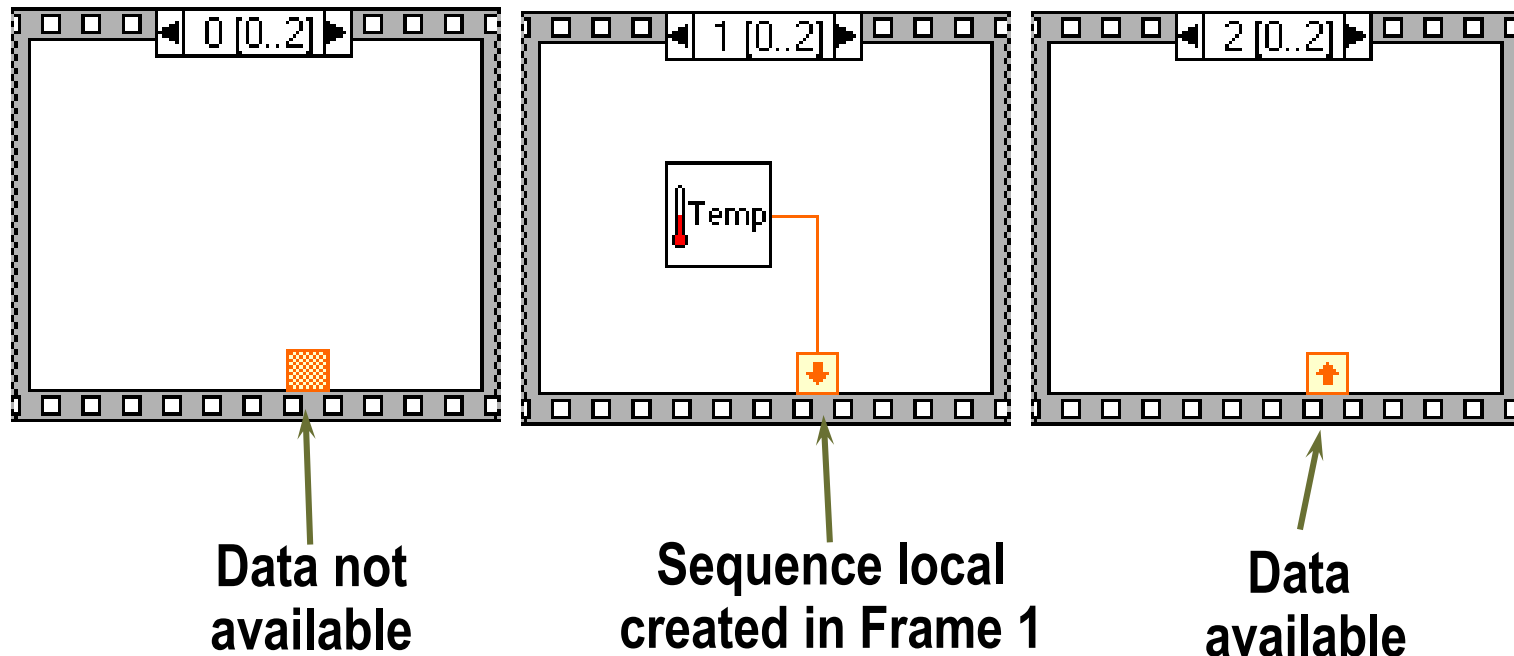
Stacked Sequence Structure



Flat Sequence Structure

Sequence Locals

- Pass data from one frame to future frames in stacked sequence structure
- Created at the border of the Sequence structure

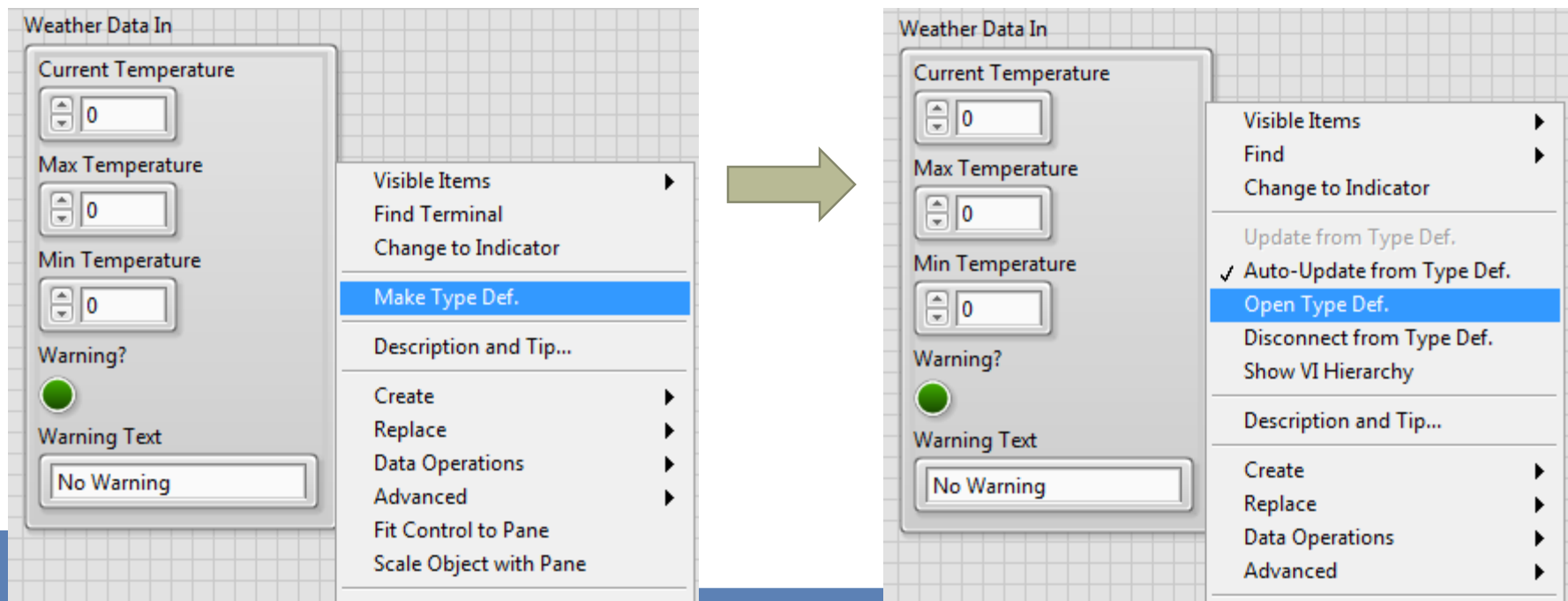


Type Definitions (Type Def)

- A type definition is a master copy of a custom data type (control, indicator, or constant).
 - A custom data type is saved in a .ctl file.
 - Instances of a type def are linked to the .ctl file.
- Instances can be controls, indicators, or constants.
- When the type def changes, instances automatically update.
 - Changes include data type changes, elements added, elements removed, and adding items to an enum.

Creating Type Definitions (Type Def)

1. Right-click a control, indicator or constant and select **Make Typedef.**
1. Right-click the object again and select **Open Type Def.**
2. Edit control, if needed.
3. Save control as a .ctl file.



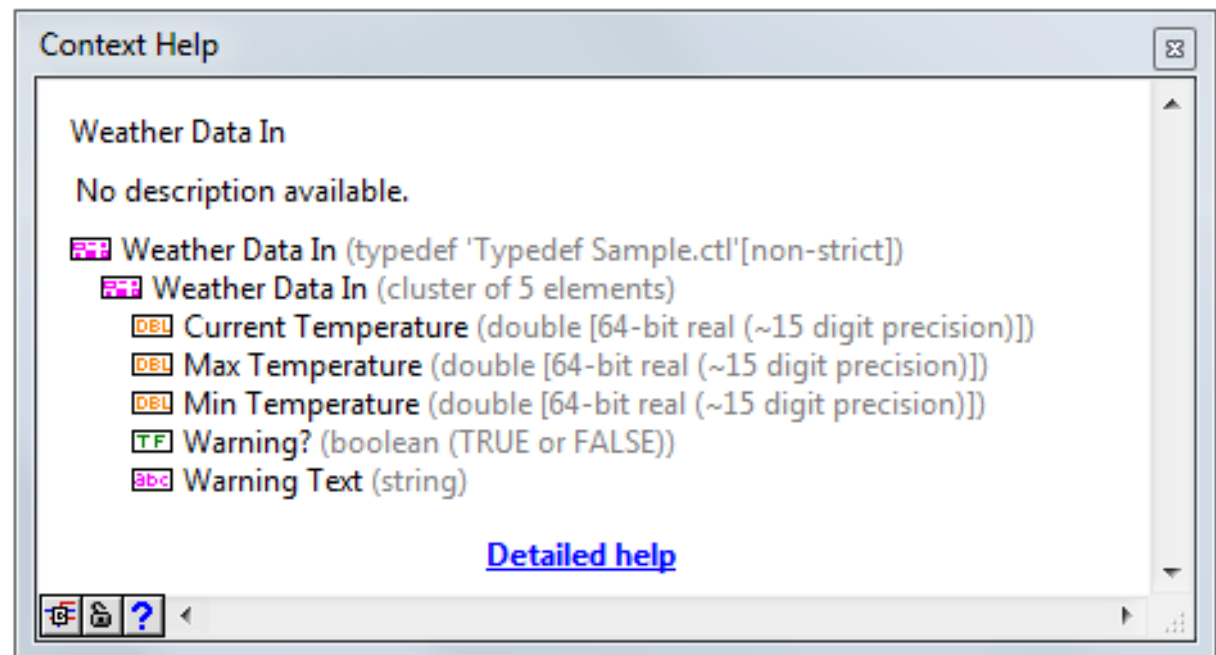
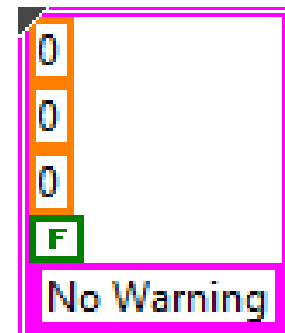
Identifying Type Definitions (Type Def)

- Look for a glyph marking the upper left corner of terminals and constants.
- Hover cursor over glyph to view tip strip.
- View Context Help while hovering cursor over terminal or constant.

Weather Data In



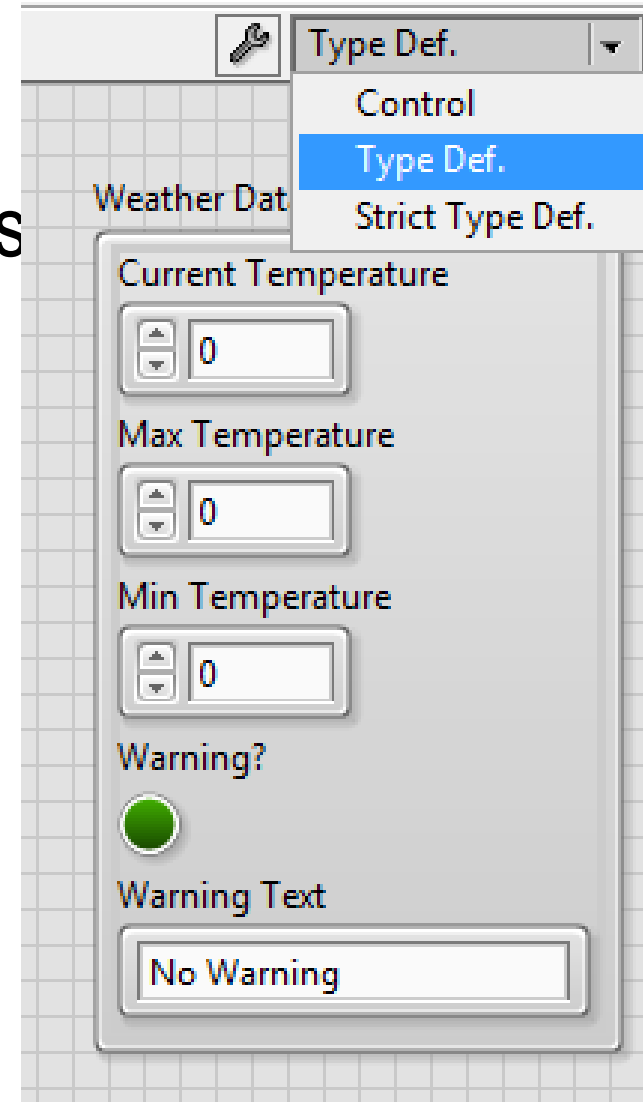
Weather Data In



Other Control Options

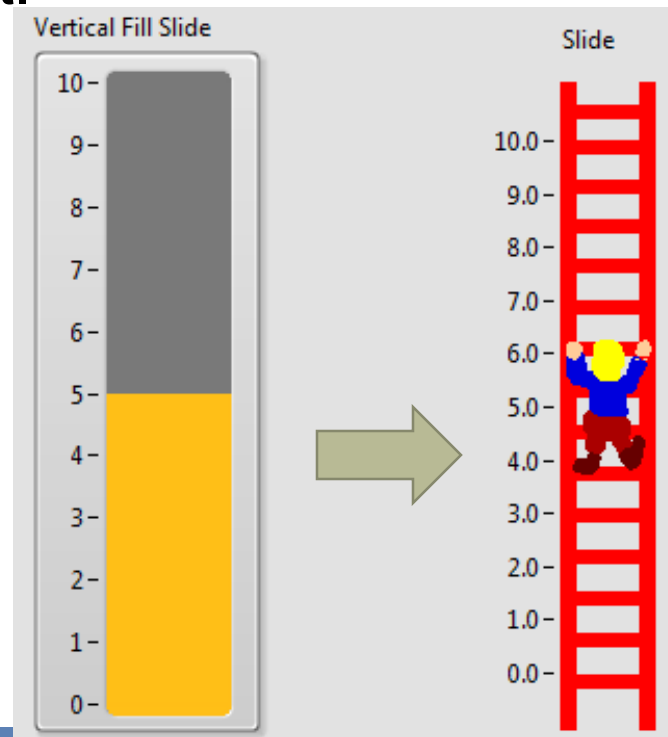
You can save a custom control as

- Control
- Type Definition
- Strict Type Definition



Control

- Instances are not linked to a .ctl file.
- Each instance is an independent copy of the control.
- Used to create controls that behave like existing controls but look different.

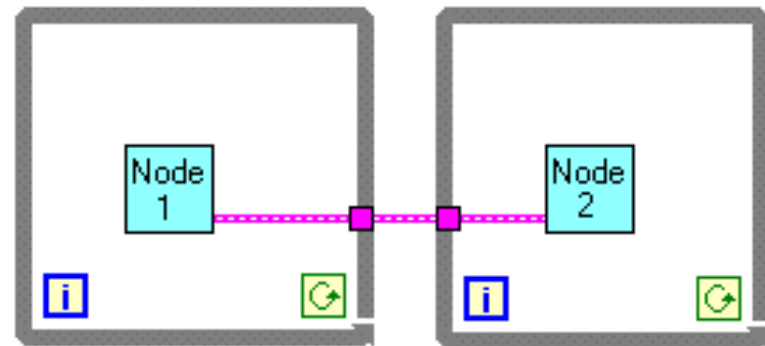


Strict Type Definition

- Strict type definitions are similar to a type definition in that:
 - All instances link to .ctl file.
 - When attributes or data types change, all instances update.
 - Examples: Changing a knob to a dial, a round LED to a square LED, or a double to an integer.
- Strict type definitions enforce every aspect of a instance except label, description, and default value.
- Use strict type definitions to ensure all front panel instances have the same appearance.

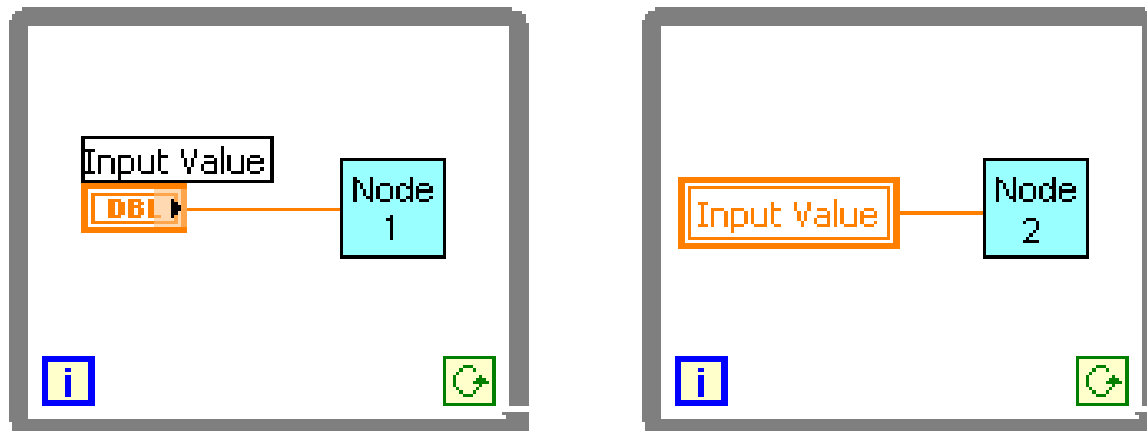
Data Management Techniques in LabVIEW

- Dataflow is the most efficient way to transfer data
 - Nodes execute when data is available to all inputs
 - Nodes supply data to all outputs when finished executing
 - Data passes immediately from source to destination
- Most applications use wires to transfer data
- Cannot use wires to transfer data between parallel tasks

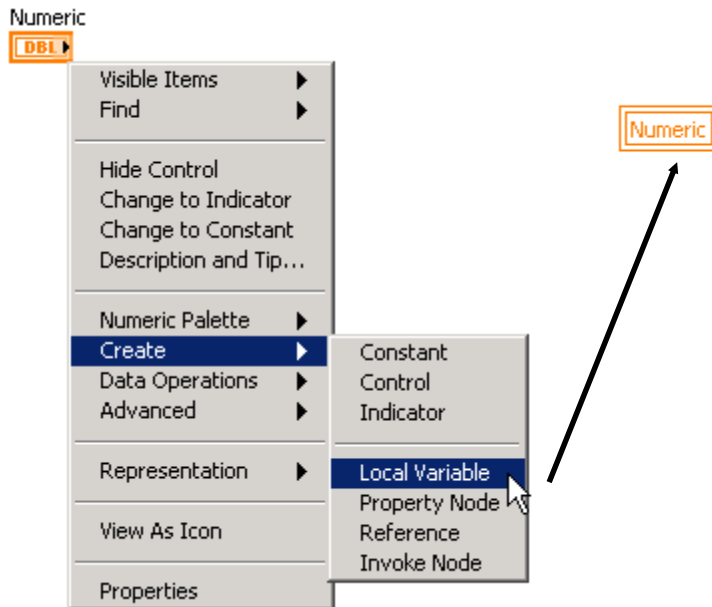


Local Variables

- Local Variables allow data to be passed between parallel loops
- They also break the dataflow programming paradigm



Creating Local Variables

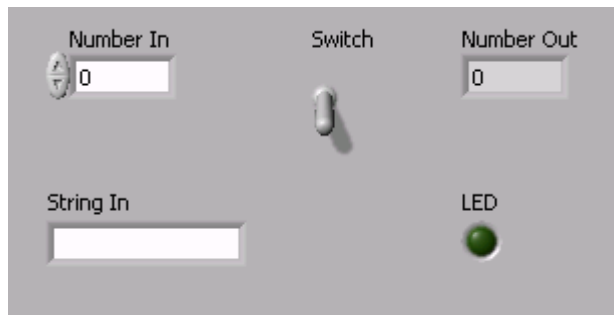


- Access panel objects from several locations on the diagram
- Two ways to create a local variable:
 - Right-click on an object's terminal and select **Create»Local Variable**
 - Select a local variable from the Structures palette

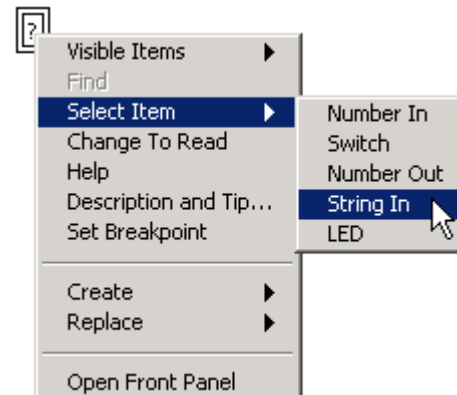
Creating Local Variables

- Right-click on the local variable node and chose Select Item to select the desired object
- Owned label becomes variable name
- Select whether you want to read or write to the local

Front Panel



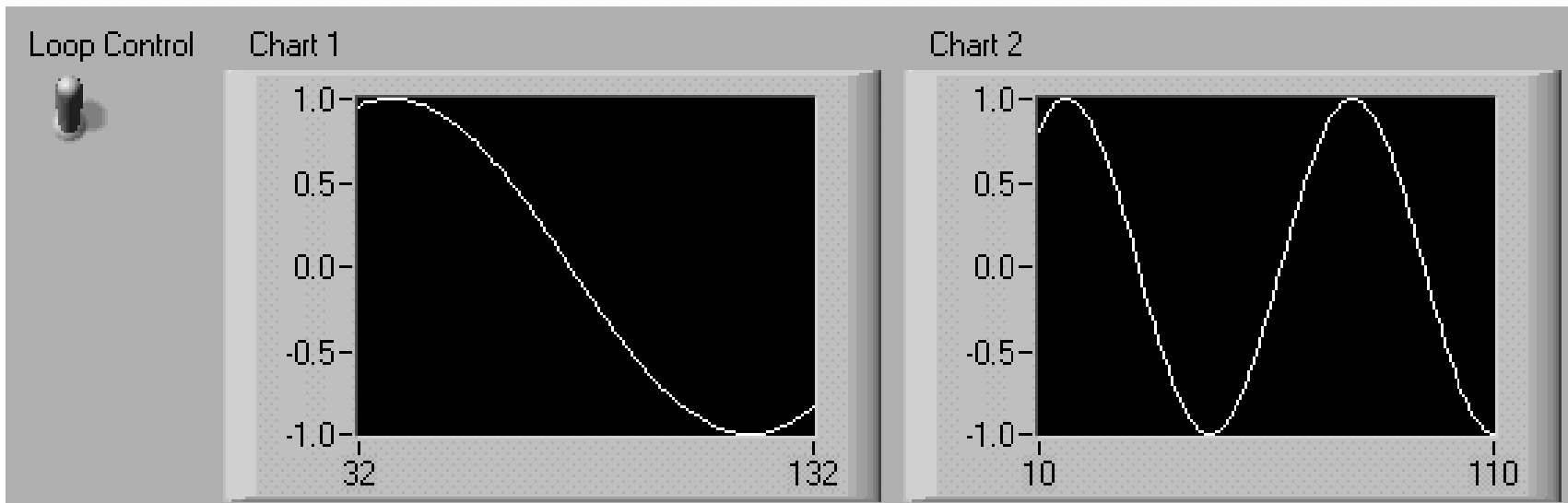
Block Diagram



String In

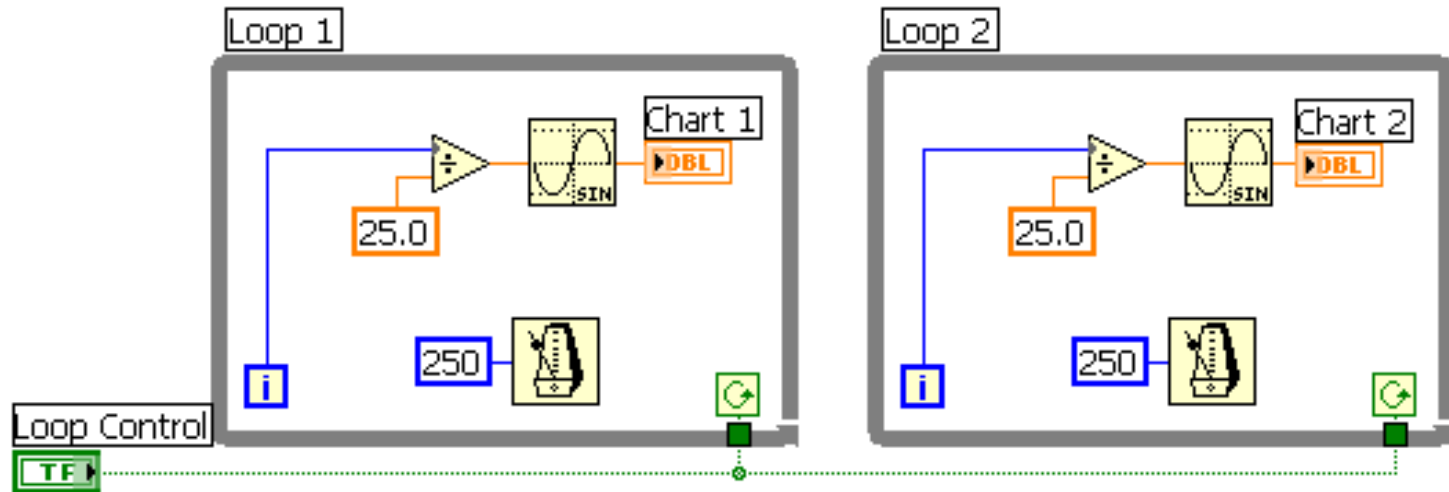
Local Variable Example

- Consider an application where you need to stop two data-independent (or *parallel*) While Loops at the same time
- Each While Loop plots a sine wave on a chart



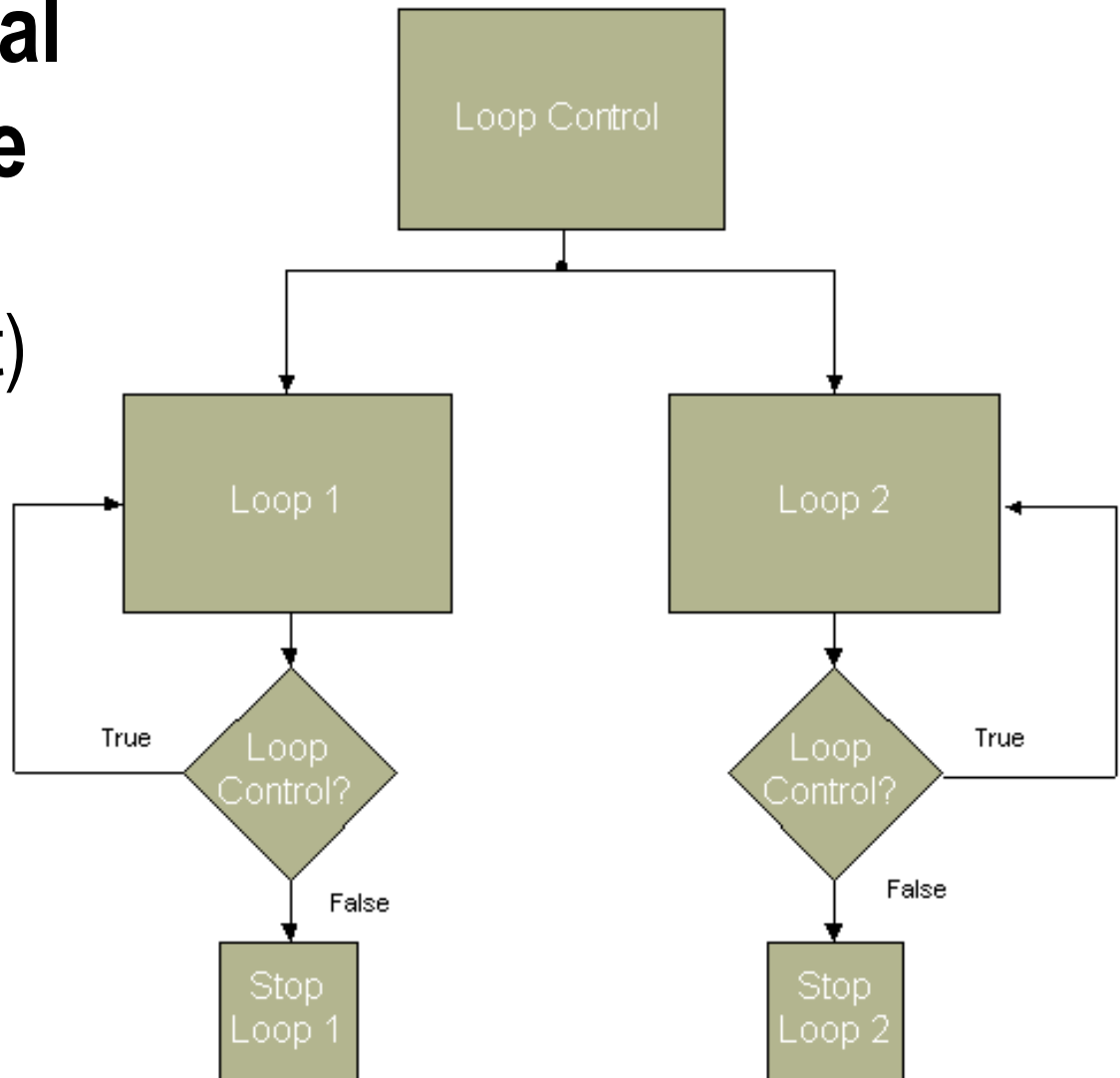
Solutions to Local Variable Example

Method 1



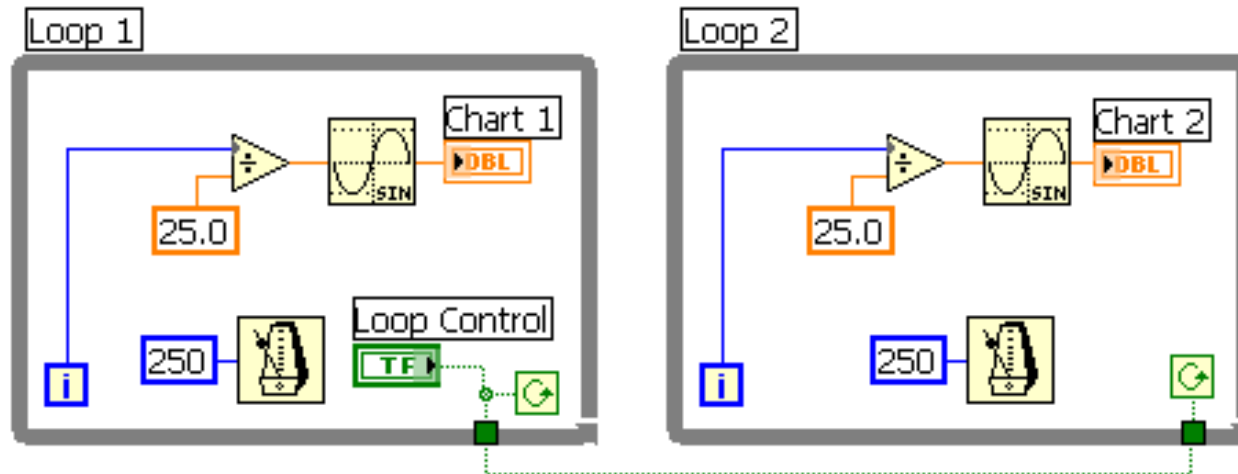
Solutions to Local Variable Example

Method 1 (Incorrect)



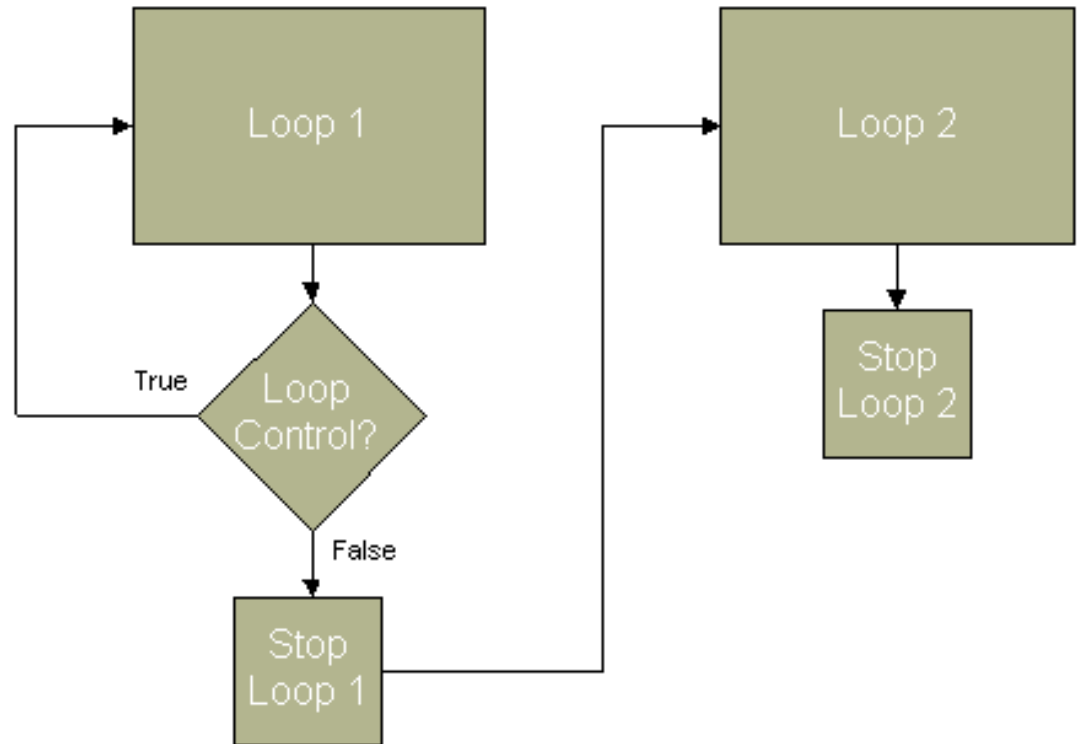
Solutions to Local Variable Example

Method 2



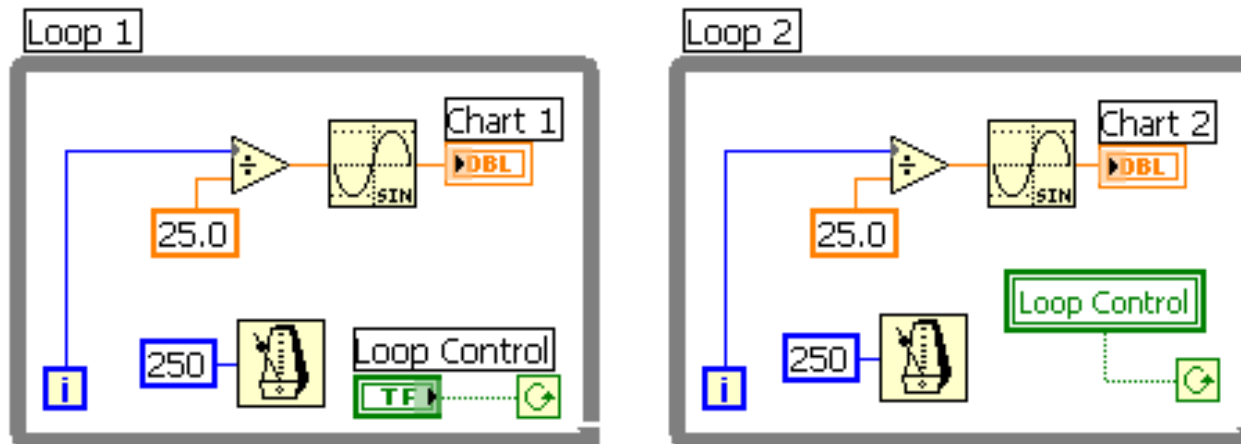
Solutions to Local Variable Example

Method 2 (Incorrect)



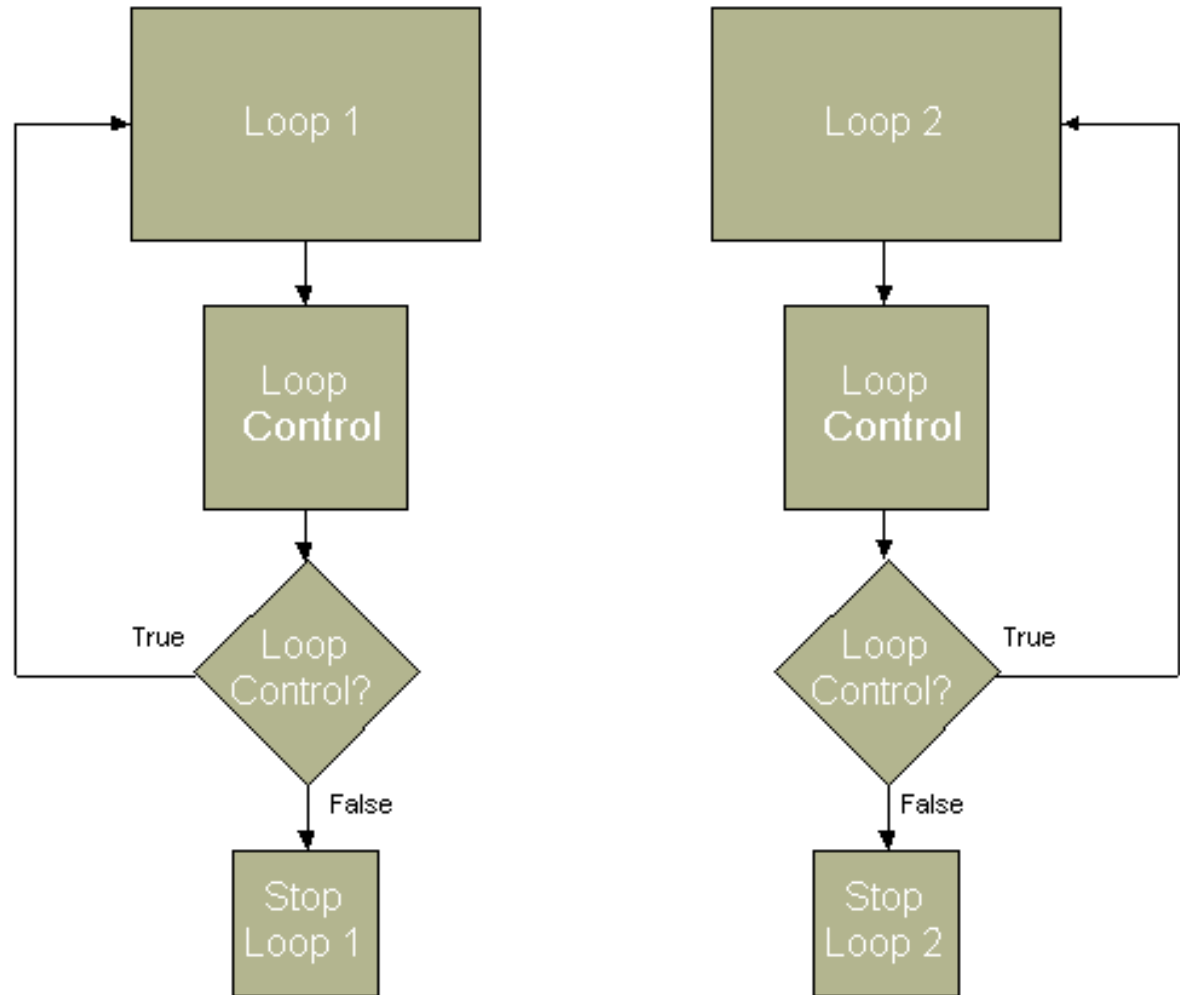
Solution to Local Variable Example

Method 3



Solution to Local Variable Example

Method 3 (Correct)

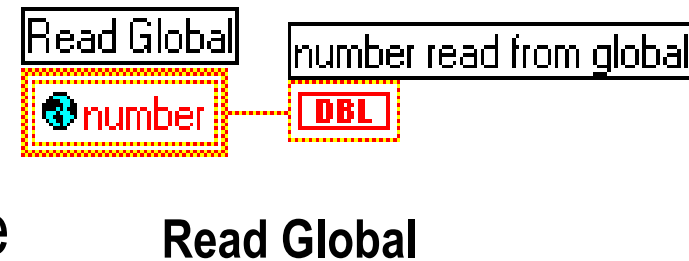


Local Variable Reminders

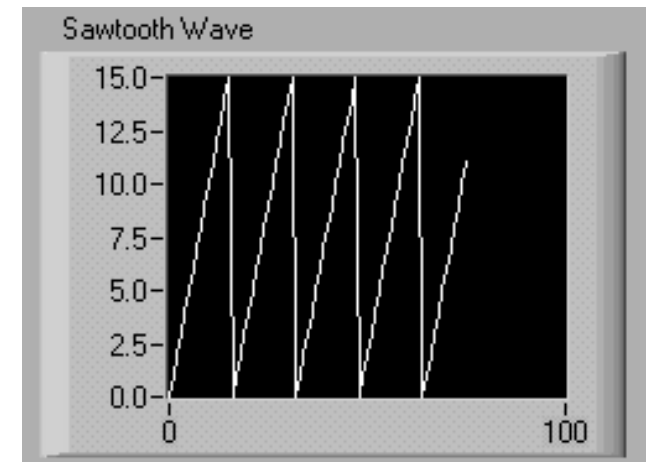
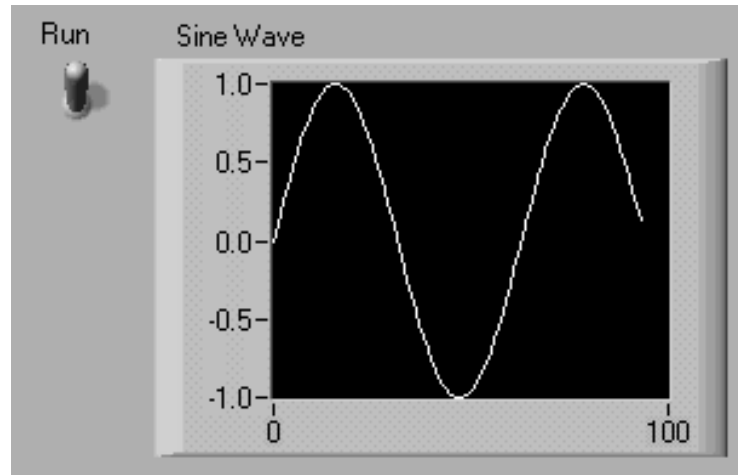
- If a front panel object will have a local variable associated with it, it must have an owned label
- When you write to a local variable, you update its corresponding front panel object
- When you read from a local variable, you read the current value of its corresponding front panel object

Global Variables

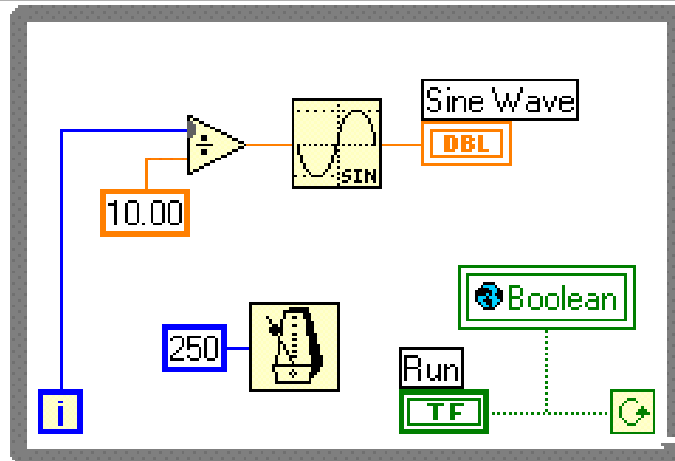
- Special kind of VI
 - Front panel and front panel objects only
 - No block diagram
- Panel objects are storage places to write and read data
- Used to pass data between VIs that are executing or between VIs that cannot be connected by a wire
- Read and write global variables
 - Write global used to write a value to the global
 - Read global used to read the current value of the global
 - Right-click on the node to toggle between read and write



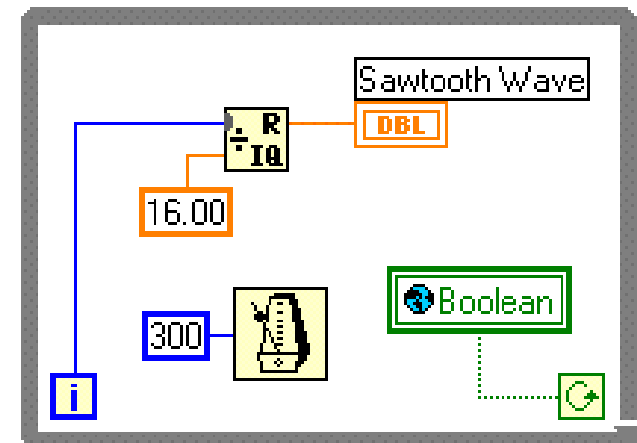
Passing Data Between VIs



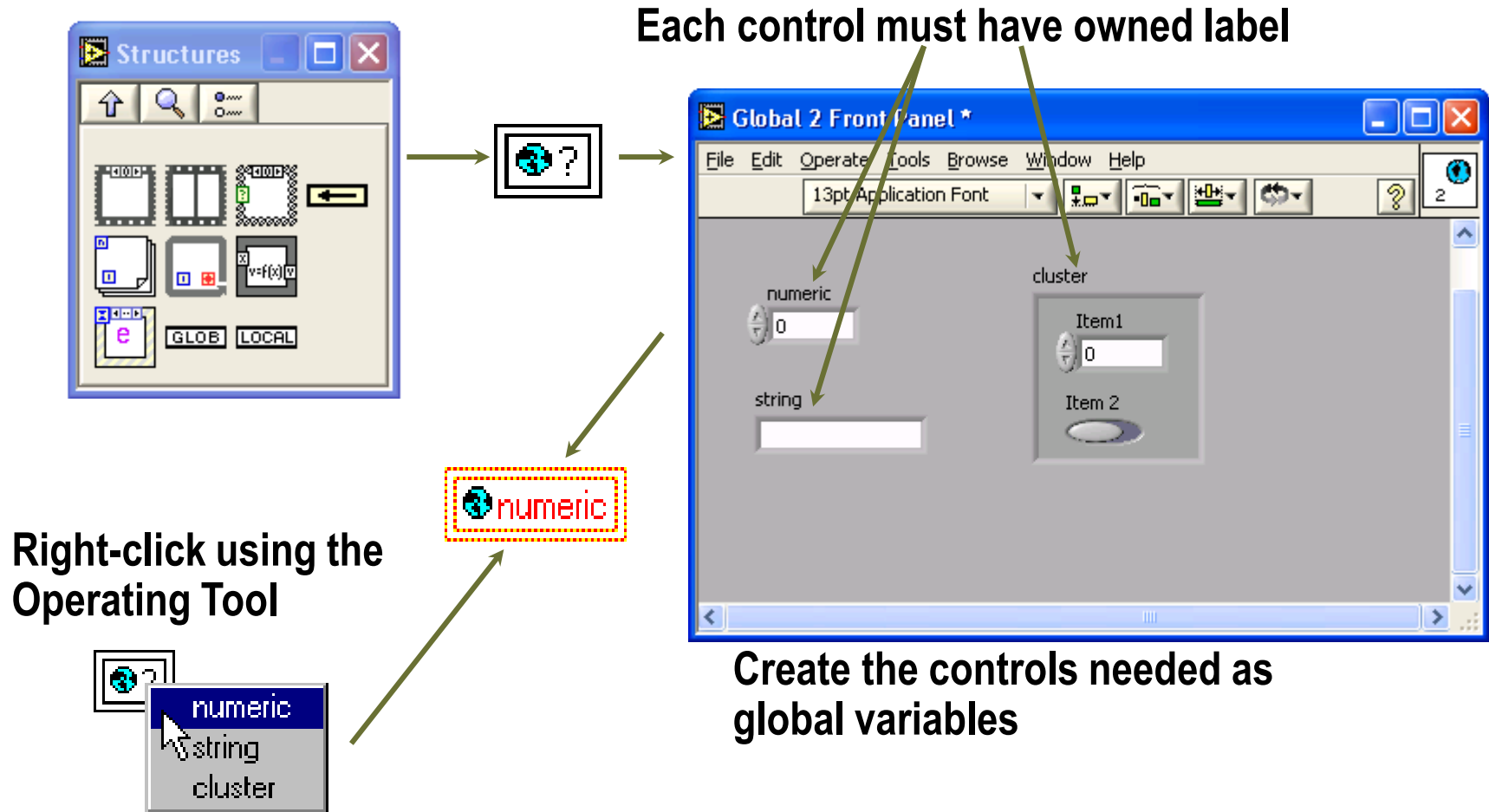
VI Number 1



VI Number 2



Creating Global Variables



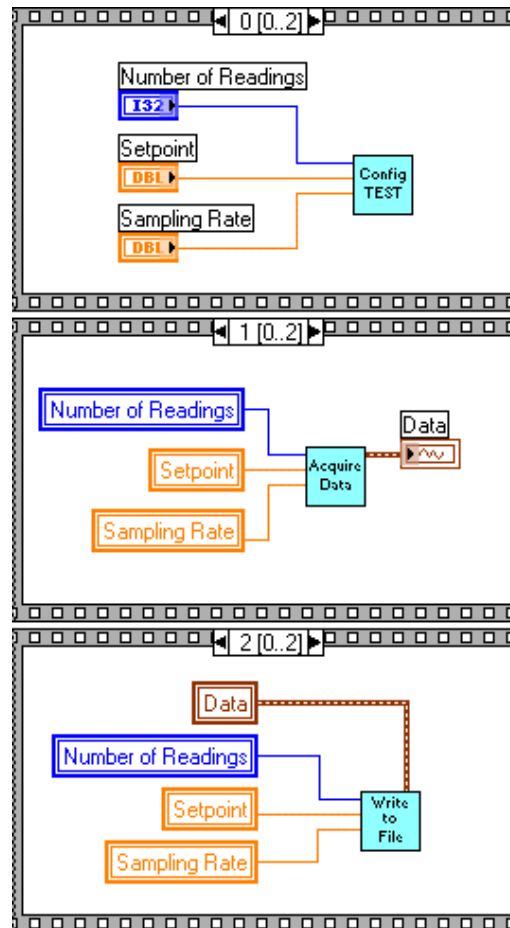
Global Variable Reminders

- Give each global variable an owned label
- Write to (initialize) a global variable before reading from it
- If you do not initialize a global variable, the default value for that object is returned
- You write to the global variable at a separate location from where you read it, usually in a different VI

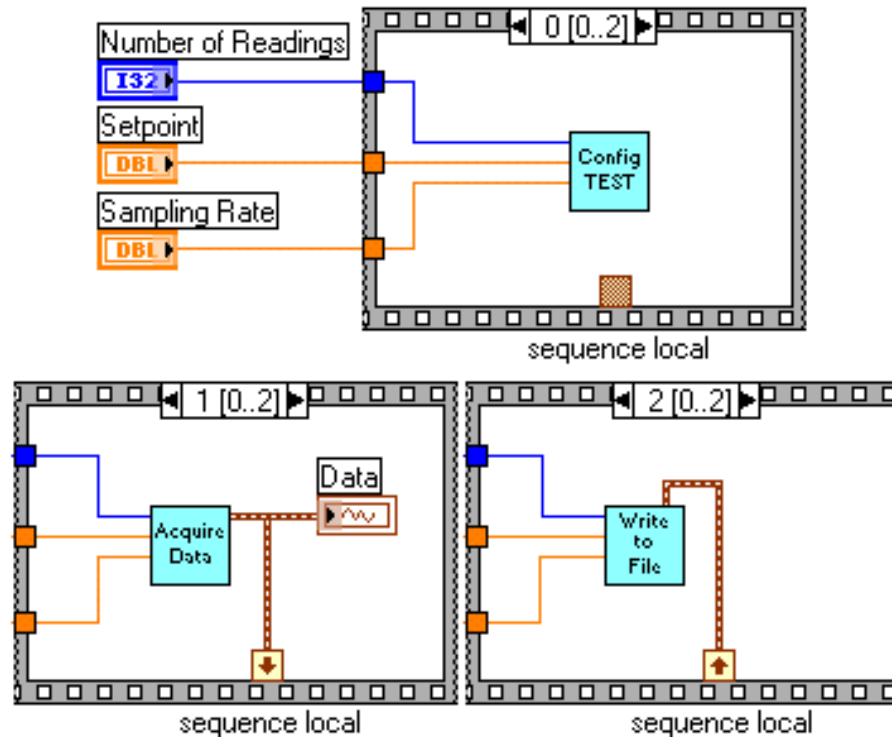
Using Local and Global Variables Carefully

- Local and Global Variables are inherently not part of the LabVIEW dataflow execution model
- Make block diagrams difficult to read and slow performance

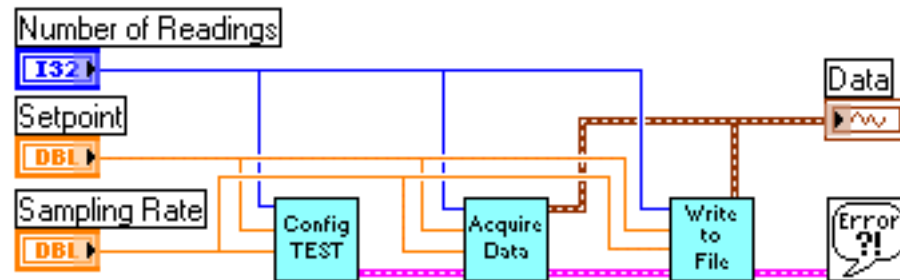
Using Local and Global Variables Carefully



Using Local and Global Variables Carefully



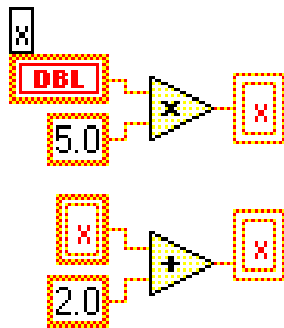
Using Local and Global Variables Carefully



Using Local and Global Variables Carefully

- Initialize local and global variables before reading them
- You must carefully avoid “race conditions”

LabVIEW Code



No clear data dependency exists, so the order of execution is not precisely known



Sequential Code

$x = x * 5$

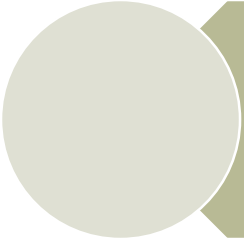
$x = x + 2$

OR

$x = x + 2$

$x = x * 5$

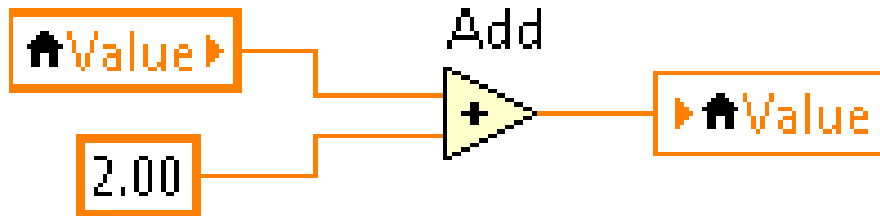
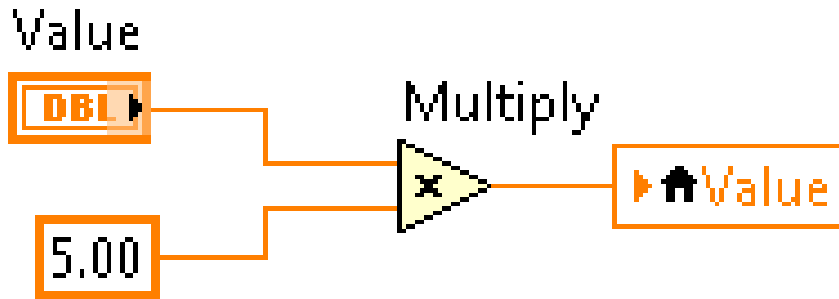
Race Conditions



Race Condition – 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.

What is the final value of the Value variable?



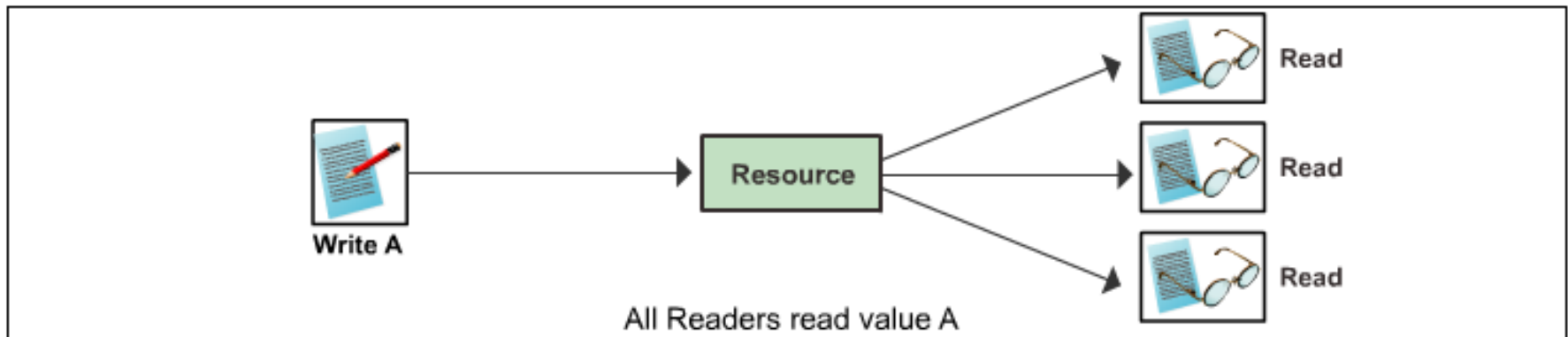
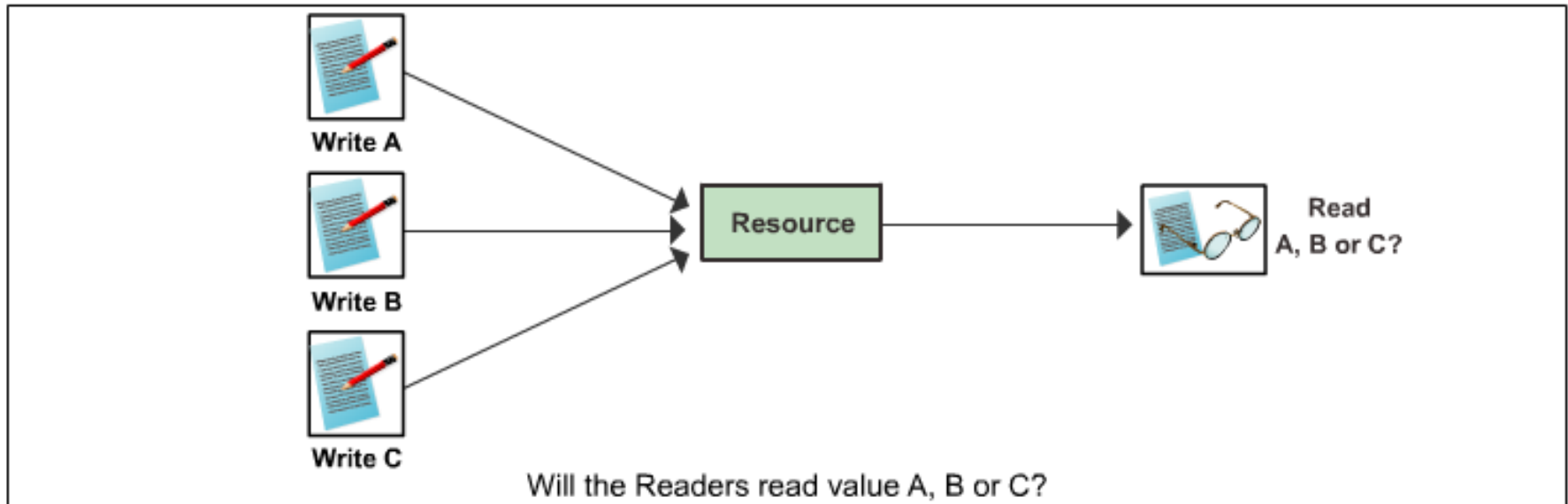
• Four possible outcomes:

- $\text{Value} = (\text{Value} * 5) + 2$
- $\text{Value} = (\text{Value} + 2) * 5$
- $\text{Value} = \text{Value} * 5$
- $\text{Value} = \text{Value} + 2$

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.
 - Use one writer, multiple readers.
 - Properly sequencing instructions.
 - Reducing use of variables.

Controlling Shared Resources



Summary

- You can use global and local variables to access a given set of values throughout your LabVIEW application
- Local variables access front panel objects of the VI in which you placed the local variable
- When you write to a local variable, you update its corresponding front panel control or indicator
- When you read from a local variable, you read the current value of its corresponding front panel control or indicator
- Global variables are built-in LabVIEW objects that pass data between VIs

Summary, cont.

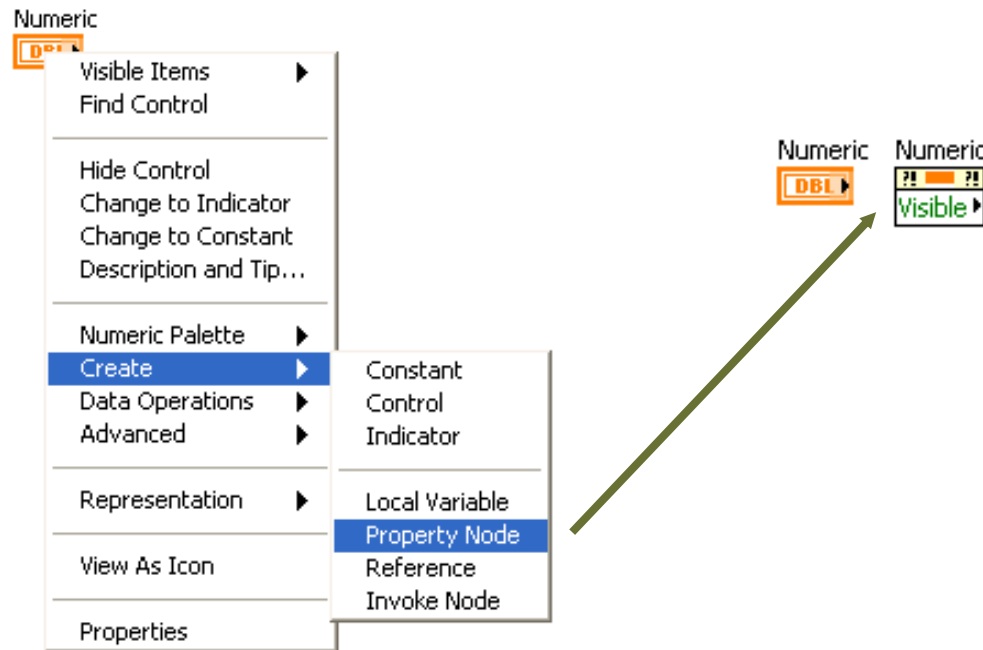
- Always write a value to a global variable before reading from it
- Write to local and global variables at locations separate from where you read them to avoid race conditions
- Use local and global variables only when necessary
- Local and global variables do not use data flow

Property Nodes

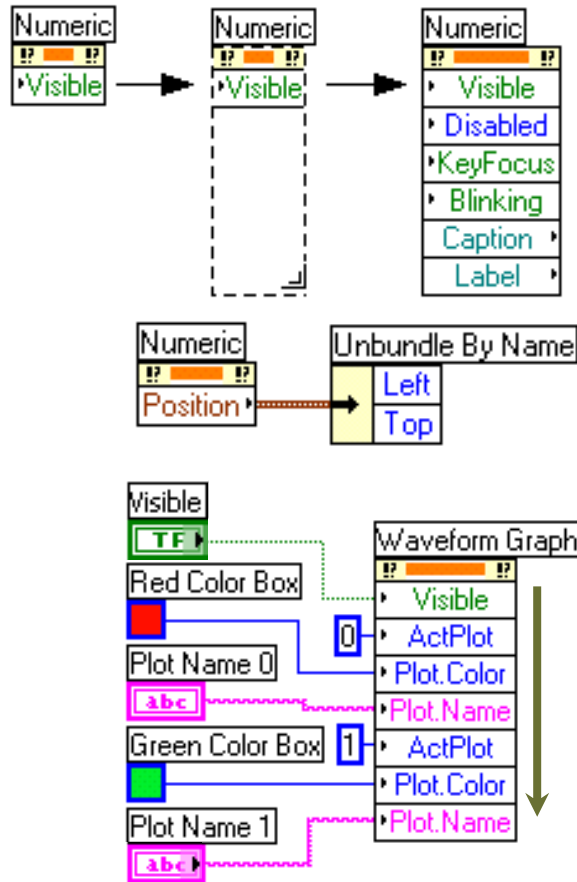
- Properties are characteristics or qualities about an object
- Property nodes allow you to programmatically set and read the properties of front panel objects, including an object's data value
- Examples of properties:
 - Display colors of front panel objects
 - Visibility of front panel objects
 - Menu selections for a ring control
 - Scales and cursors on graphs
 - Size of front panel objects
 - Location of front panel objects

Creating Property Nodes

- Right-click on a panel object or its diagram terminal and choose **Create»Property Node**
- Use the Operating tool to select which property to read or write

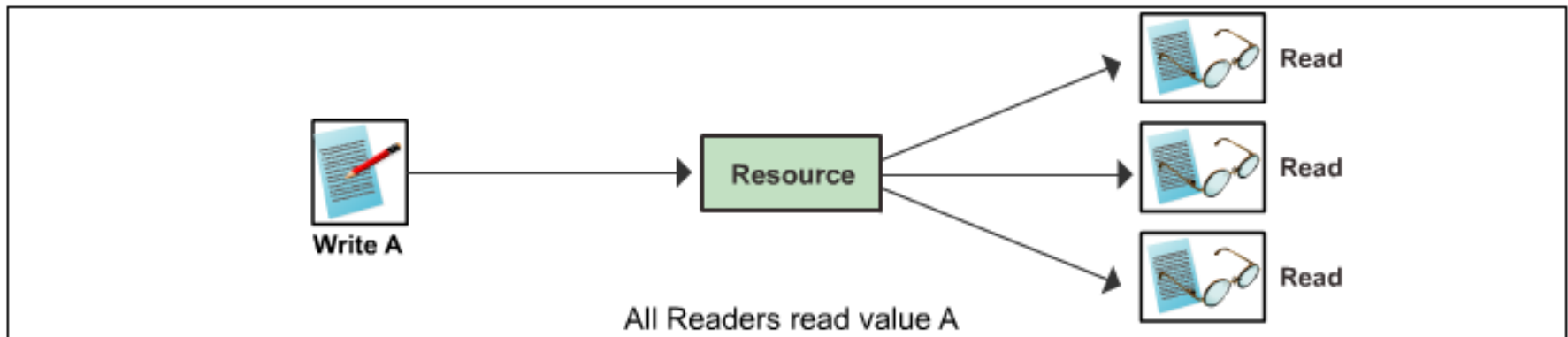
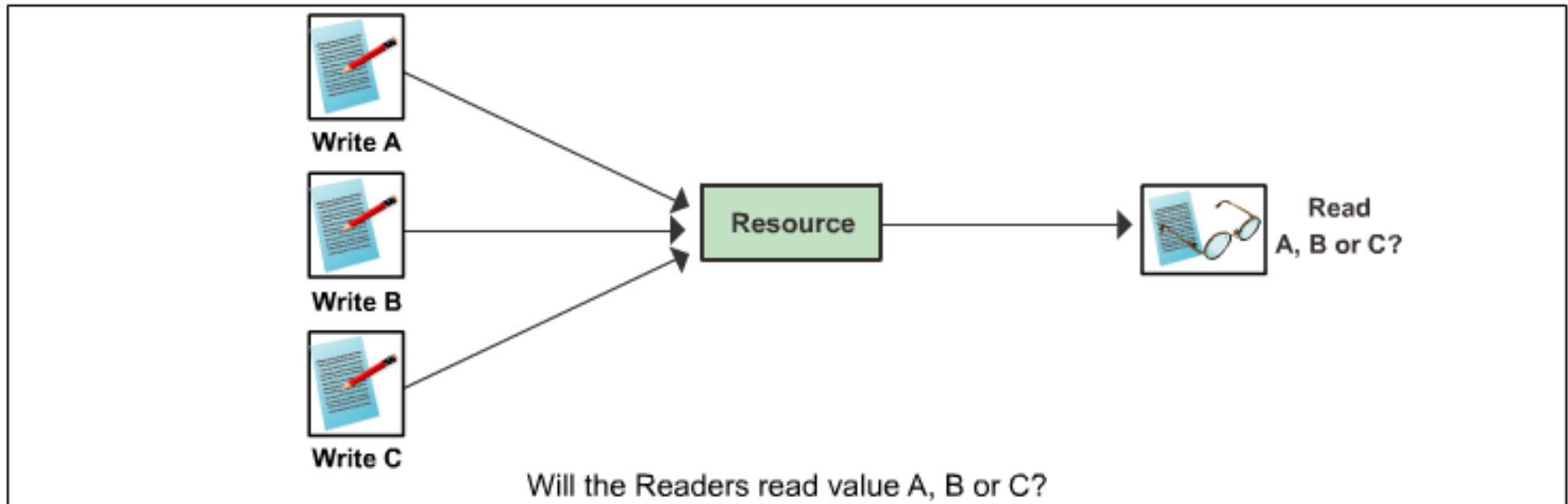


Implementing Property Nodes



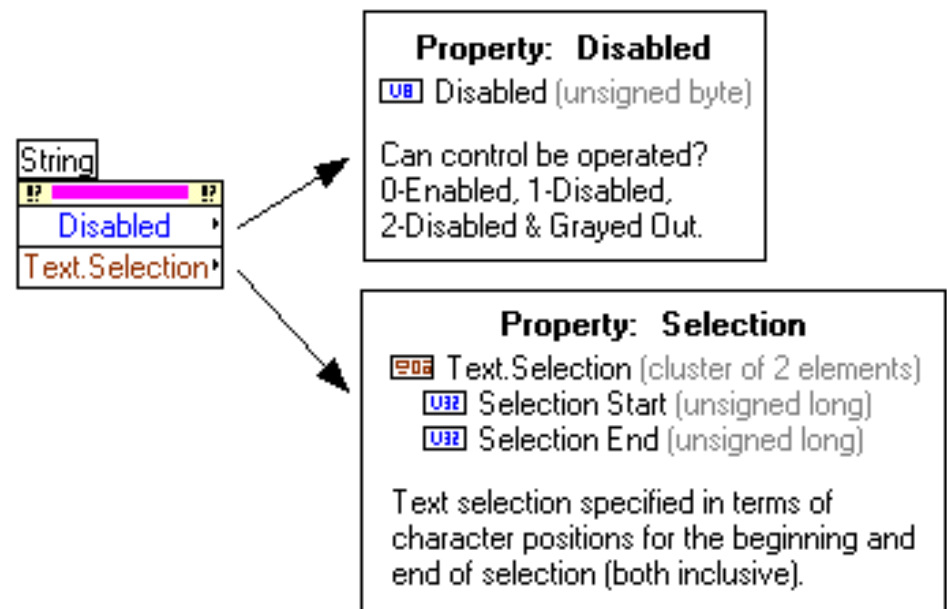
- Right-click on property node to choose Read or Write
- You can add terminals by resizing the property node
- Some properties are clusters—use Bundle and Unbundle functions
- Properties in a single property node execute top-to-bottom

Controlling Shared Resources



Use the Context Help Window

- The Context Help window and Online Reference are invaluable tools for using property nodes
- With the Context Help window active, pass the cursor over the properties to see descriptions

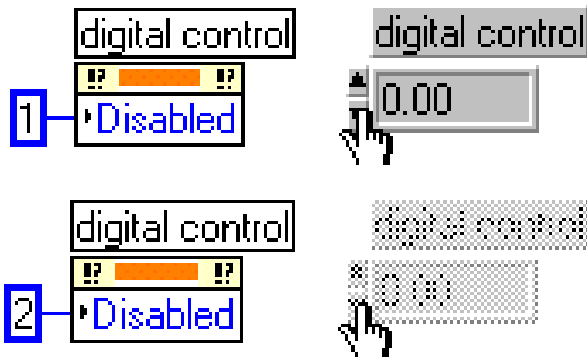


Common Properties

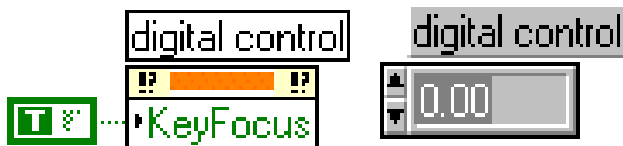
Visible Property



Disabled Property



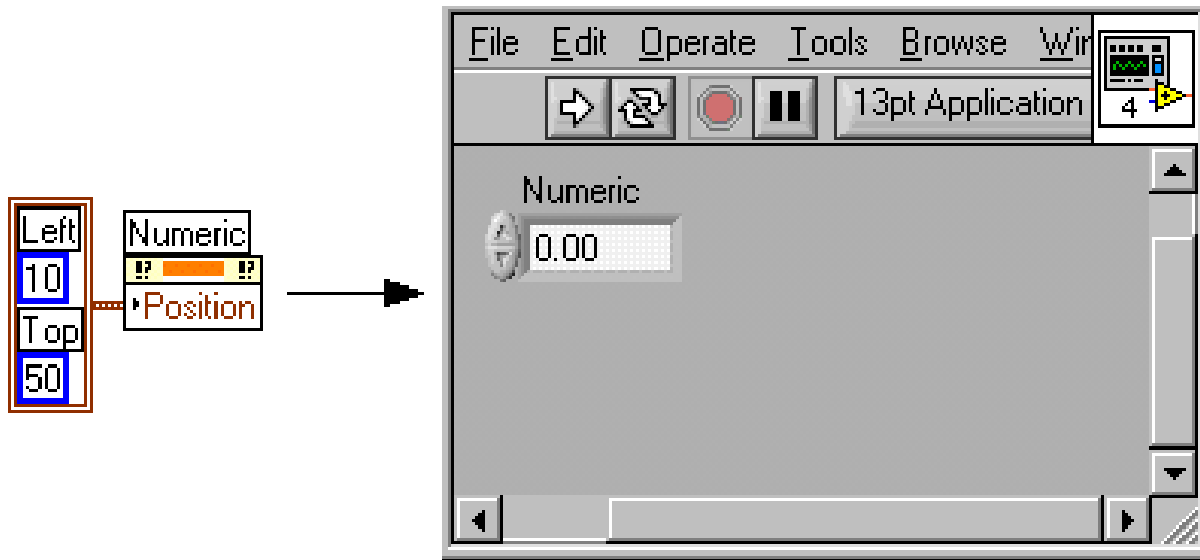
Key Focus Property



- Makes the selected front panel object visible or invisible
- Disables user access to the control without altering its appearance
- Disables user access to an object and grays it out
- Sets or reads the key focus—if TRUE, the cursor is active in the object's numeric or text fields

Common Properties

Position Property



- Sets or reads the location of a front panel object
- Causes a control to flash if set TRUE

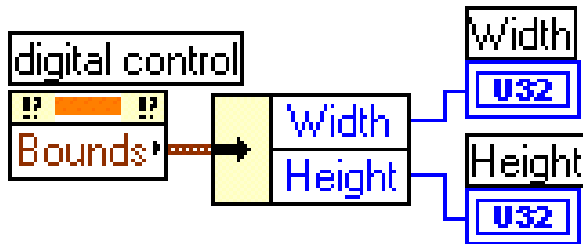
Blinking Property



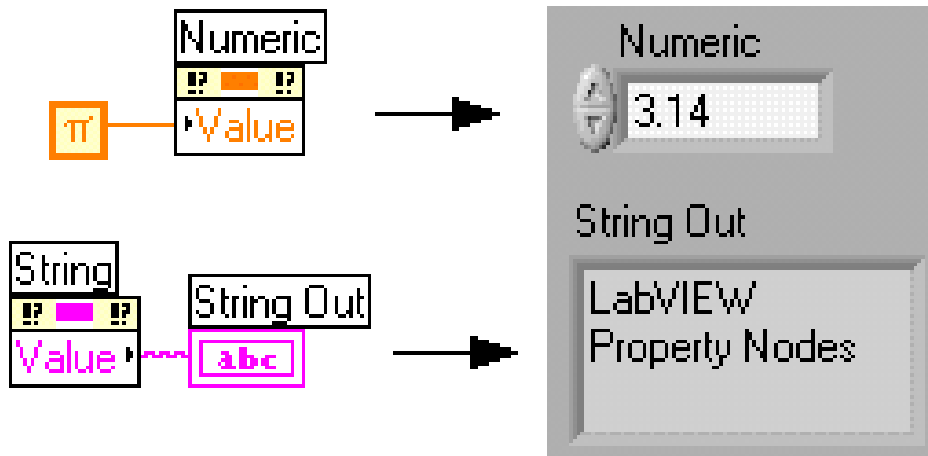
- Blink rate and colors configured in the LabVIEW Preferences

Common Properties

Bounds Property



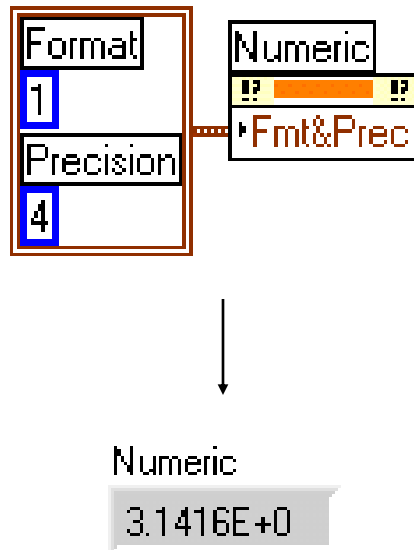
Value Property



- Reads the boundary of an object on the panel in units of Pixels (Read-only property)
- Includes bounds for control and all its parts
- Cluster of two values: Width and Height
- Sets or reads the value of a panel object

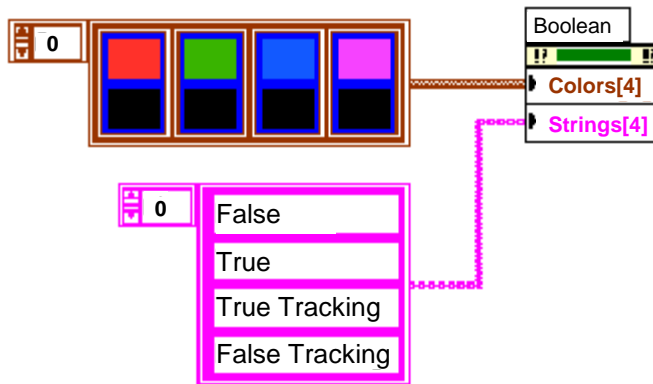
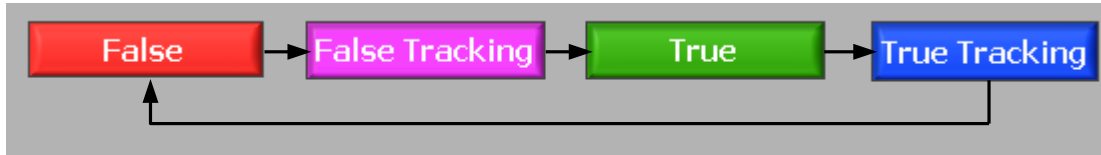
Numeric Properties

Format and Precision Property



- Set or read the format and precision
- Both values set as a cluster, or individually using unsigned byte integers
- Format can be Decimal, Scientific, Engineering, Binary, Octal, Hex, or Relative Time notation

Boolean Properties



Strings Property

- Set strings for True, False, True Tracking, and False Tracking

Colors Property

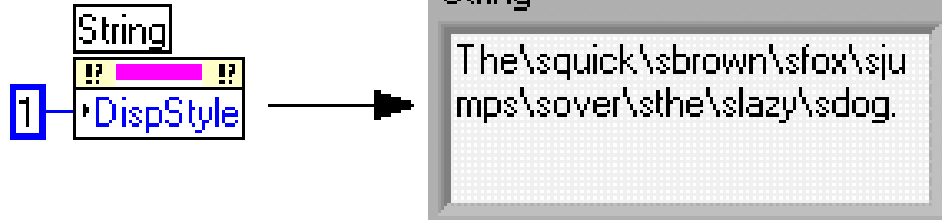
- Set colors for True, False, True Tracking, and False Tracking
- Second colors in clusters are non-functional

Note: Check the “Multiple Strings” option in Customize Control to enable Strings[4] property

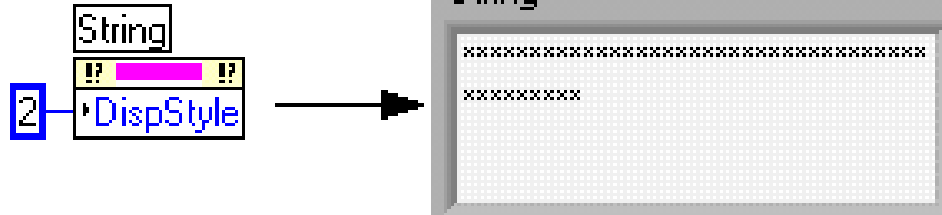
String Properties

Display Style Property

Codes Display



Password Display

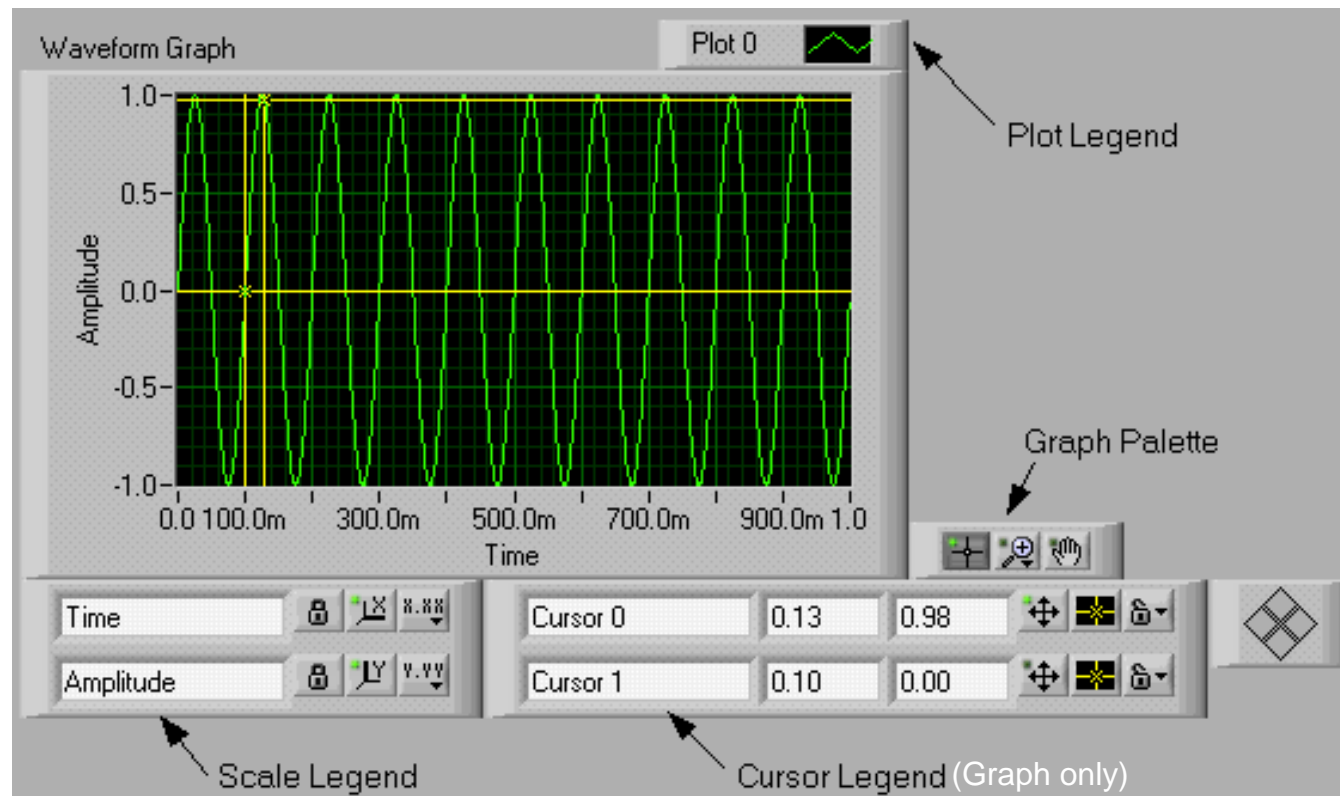


Hex Display



- Reads or sets how the characters in a string are shown— Normal, '\', Codes, Password, or Hex Display

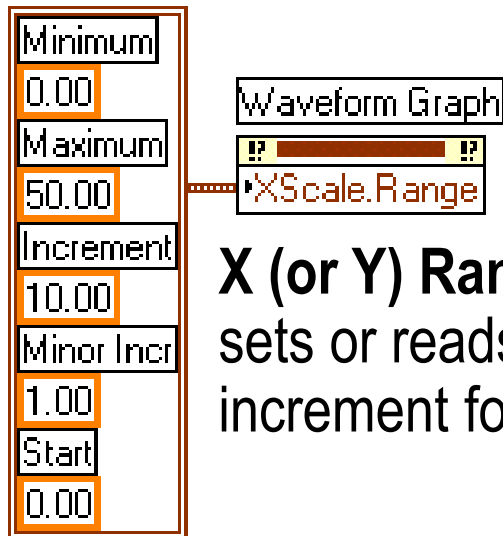
Graph and Chart Properties



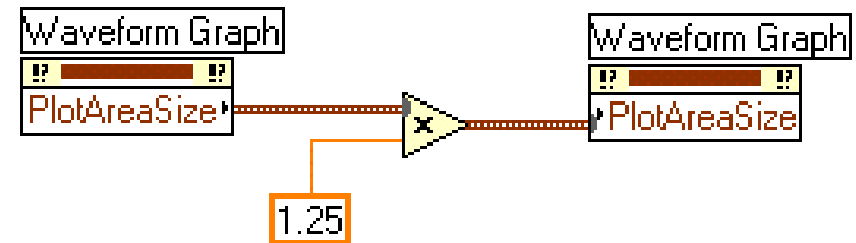
- Functionality of graphs and charts is enhanced with property nodes
- All parts of the graphs and charts can be controlled through properties

Graph and Chart Properties

- Plot color
- X and Y scale information
- Visibility of legend and palette
- Cursors



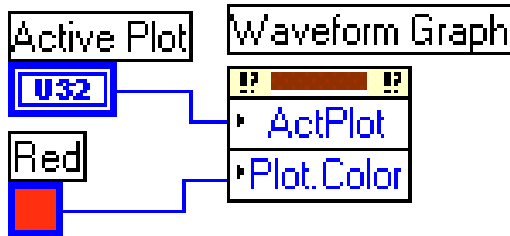
X (or Y) Range Properties
sets or reads range and
increment for the specified axis



Plot Area Size Property
sets or reads the size of the
plotting area of the graph or
chart in pixels

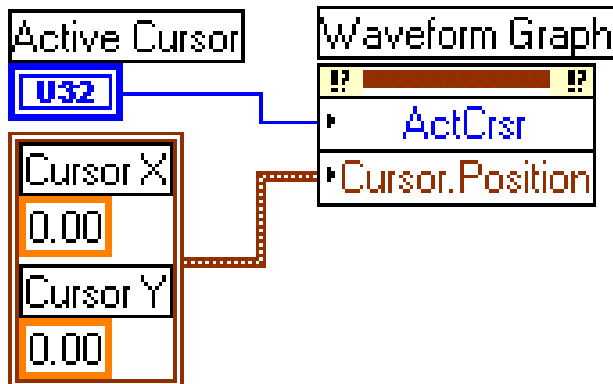
Graph and Chart Properties

Active Plot and Plot Color Properties



Set or read the color or the active plot

Cursor Properties



Properties for cursor position or index of the active cursor—
cursor 1 is set to X=0, Y=0

Summary

- Property Nodes are powerful tools for expanding user interface capabilities.
- You create Property Nodes from the Create shortcut menu of a control or indicator.
- You can create several Property Node terminals for a single front panel object
- You can set or read properties such as user access (enable, disable, dim), colors, visibility, location, and blinking.

Summary, cont.

- Property Nodes greatly expand graph and chart functionality by changing plot size, adjusting the scale values, and operating or reading cursors.
- Use the Context Help window to learn more about individual properties.
- You can use Control References and refnums to access Property Nodes inside subVIs.