(PRACTICAL) COMPUTATIONAL PHYSICS

Physics 551
Lecture 13

NOTATION

Extra Reading

Optional Exercise

Recommended

- This lecture slides for this course will attempt to use a uniform notation throughout. A normal paragraph looks like this.

✒ *Italicized paragraphs with pen bullets will indicate definitions, with the defined word or phrase shown in* ***SMALL-CAPS****.*

✏ Pencil bullets will indicate the introduction of **new notation**.

☞ Pointing hand bullets indicate important points that might otherwise be overlooked.

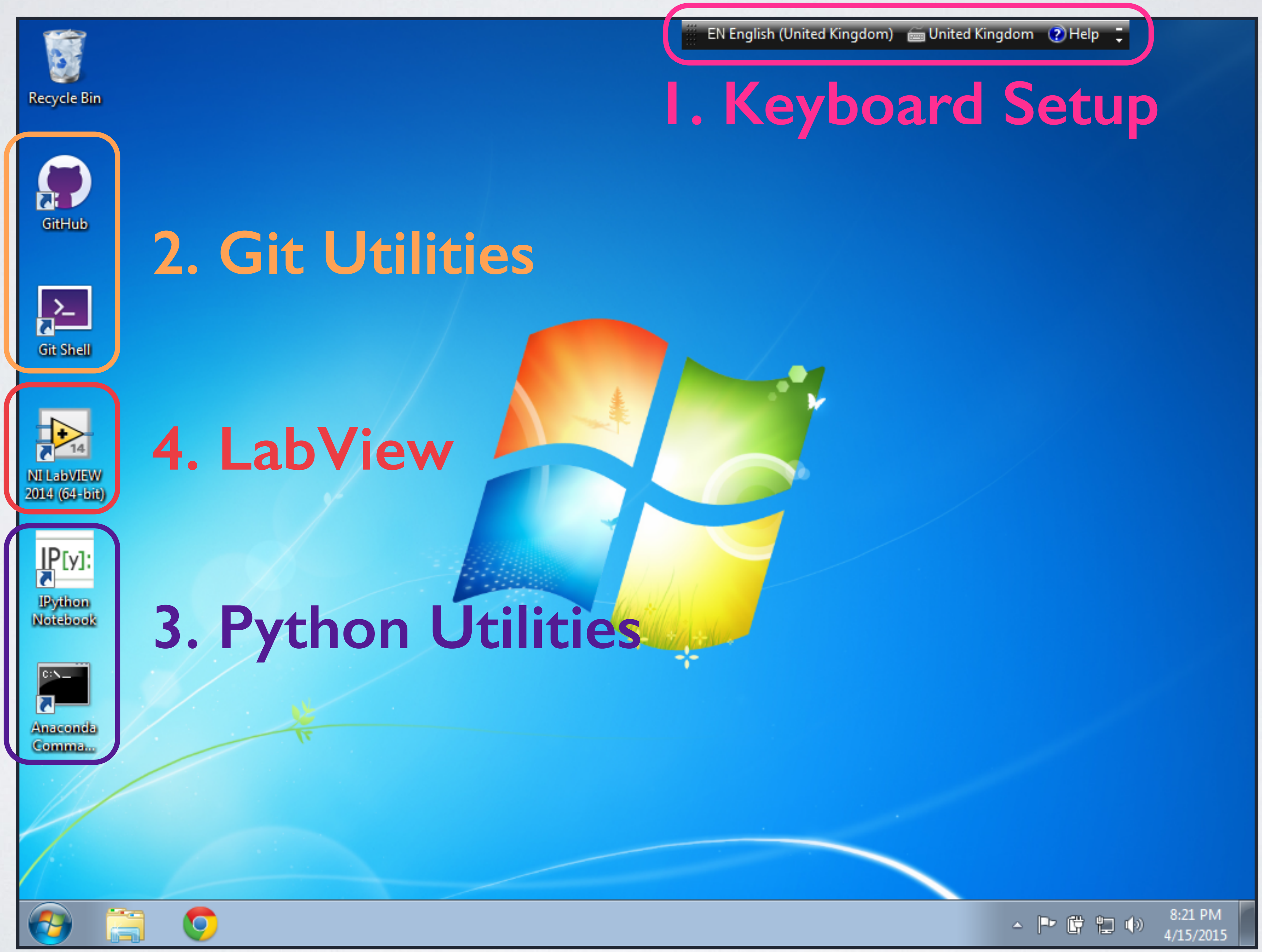
ANNOUNCEMENTS

ANNOUNCEMENTS

- This week a **new** VirtualBox virtual machine that uses **Microsoft Windows 7** as its operating system is required.
- The Virtual Machine image requires approximately **30 GB** of storage space.
- Instructions for obtaining it can be found on the Blackboard Learn website.

MICROSOFT WINDOWS ENVIRONMENT

THE DESKTOP AND SETTING UP



WINDOWS GIT SETUP

- Run the "Git Shell" utility using the Desktop shortcut.
- Invoke the following commands:

```
$ git config --global user.name "<your_github_username>"

$ git config --global user.email "<your_email_address>"

$ git clone https://github.com/hughdickinson/CompPhysL13Labview.git
  C:/Users/ComputationalPhysics/Documents/labview/lecture13
```

- Replace the **red** text with values that are appropriate for you.

DEMONSTRATION

Cloning the LabView Demonstration Material **in Windows**

LABVIEW

LABVIEW OVERVIEW

- ✑ *National Instruments **LabView** is an **INTEGRATED DEVELOPMENT ENVIRONMENT** (IDE) that is designed to simplify the development of hardware control and data acquisition interfaces.*
- ☞ LabView provides a **graphical programming interface** and implements a **data flow** programming paradigm using a programming language called ***G***.
- ☞ Program development entails definition of **connections** that **transfer data** between discrete **computational elements**.

LABVIEW OVERVIEW

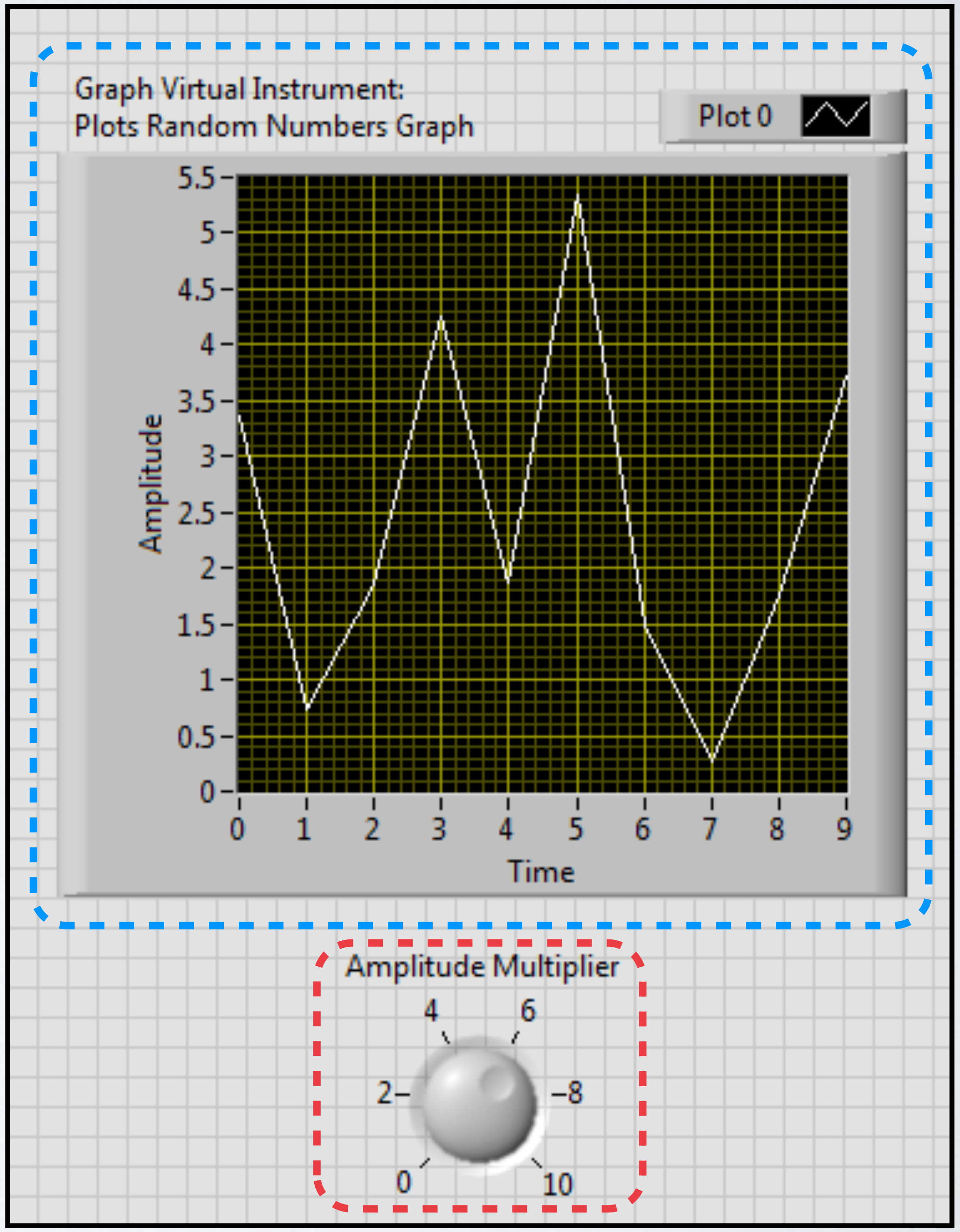
- *LabView programs are called* ***VIRTUAL INSTRUMENTS*** *(or* ***VI****s).*
- *VIs can be used as* ***self-contained functional elements*** *within other VIs. In this case, they are referred to as* ***SUB-VIS****.*
- *Communication of data* ***between*** *sub-VIs is possible if their definition includes appropriate* ***TERMINAL*** *elements.*
- Hardware vendors often provide sets of **ready-to-use** VIs that interface with the instruments they sell.

THE LABVIEW INTERFACE

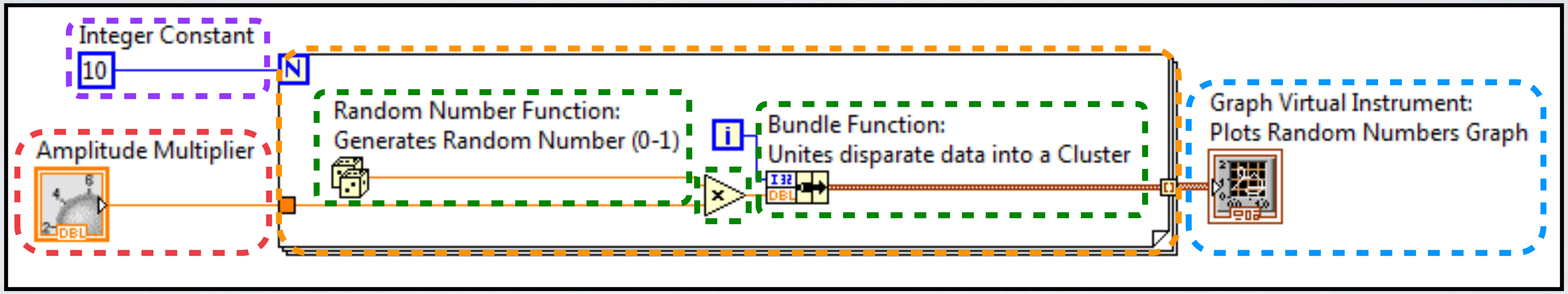
- The **basic** LabView interface comprises **two main windows**.
- ✑ *The **FRONT PANEL** window provides a preview of the Graphical User Interface with which the **end-user** of the software will interact.*
- ✑ *The **BLOCK DIAGRAM** window displays a graphical representation of the **G** program and provides an interactive interface to modify and develop its functionality.*

THE FRONT PANEL

- The front panel is used to **position** and **customize** the UI elements of **your** program.
- Elements in the front panel are classified as **controls** or **indicators.**
- The front panel is **not** used to develop functional ***G*** code.

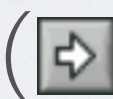


THE BLOCK DIAGRAM

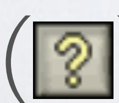


- *The block diagram displays the connections (or* ***Wires****) between the different elements that comprise a LabView program.*
- In addition to **control** and **indicator** elements, the block diagram also displays **functional** elements that perform mathematical computations or data manipulation.
- **Numerical constants** and **flow control structures** like ***for-loops*** can also be defined in the block diagram.

CRUCIAL INTERFACE ELEMENTS

- To test-execute a LabView VI while it is under development use the ***Run*** button ().

☞If there are problems with your VI that prevent it from executing, the Run button will display a broken arrow ().

- To display the values of data flowing along wires in the block diagram use the ***Highlight Execution*** button ().
- Brief **contextual help** for any elements in the block diagram or front panel can be obtained using the ***Context Help Button*** ().

DEMONSTRATION

Very **Basic** Introduction to the **LabView Interface**

DATA FLOW MODEL

- The ***G*** language models computer programs as the ***flow*** of data along ***wires*** that connect functional ***elements***.

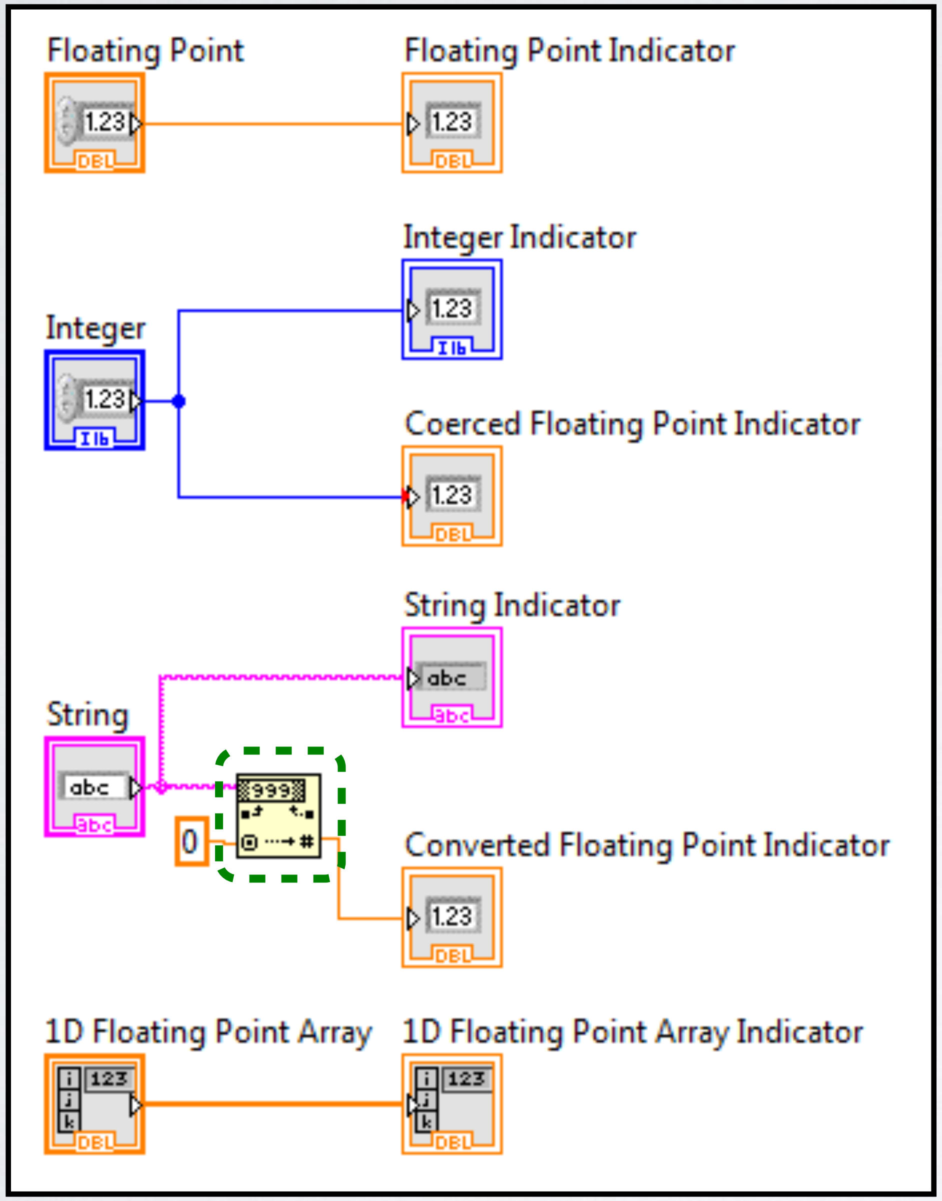
☞ LabView is a **strongly typed** language.

☞ The **colour**, **width** and **patterning** of a wire identifies the **type** of data it carries.

- ***If no information is lost***, *the LabView interface will convert between numeric types. This is called* ***COERCION***.

☞ All other type conversions **must** be **explicitly** specified in the block diagram.

EXAMPLES OF DATA TYPES



Floating Point Data

Integer Data

Red Connection Indicates ***coercion***.

String Data

Explicit **conversion** between **string** and **floating point** data.

Thicker lines indicate array data.

FUNDAMENTAL NUMERIC TYPES

- LabView provides two **fundamental** numeric types
  - **Integral** types.
  - **Floating-point** types.
- Like C++, LabView can **represent** numerical data using different, **fixed** numbers of bytes.
- The type and representation associated with numerical **controls**, **indicators**, and **constants** can be selected from the front panel **and** the block diagram.

DEMONSTRATION

Working with **typed data** in LabView

ARRAYS

- ☞In ***G***, arrays are collections of data that have the **same type**.
- ☞Array **elements** can be defined ***statically*** (i.e. at compile time), or generated ***dynamically*** using **functions**, **VIs** or **looping structures**.
- ☞***G*** supports the creation of **multidimensional** arrays.
- ☞The **dimensionality** of array data that are transmitted along wires is indicated by their **width** in the block diagram.

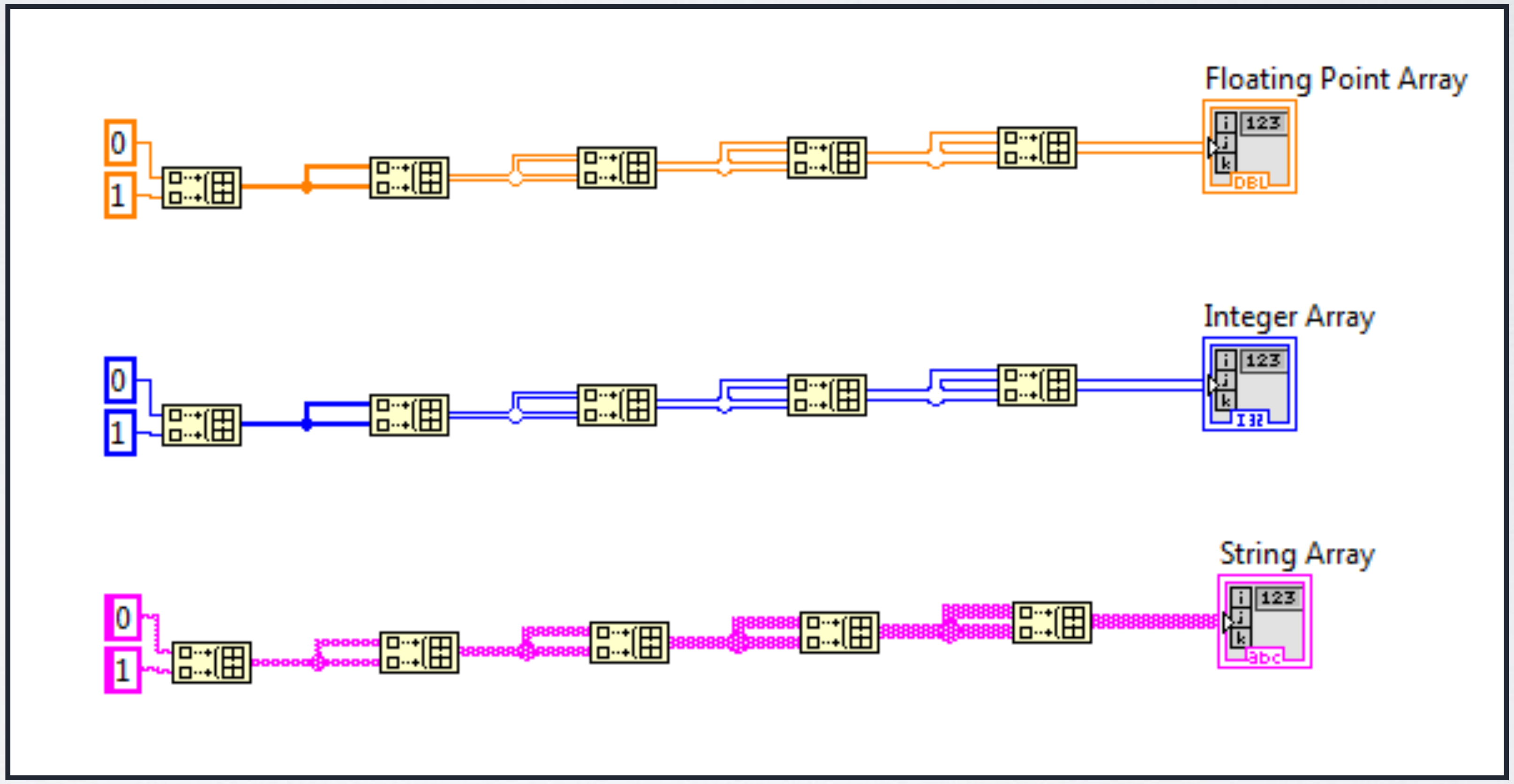
ARRAYS

0D 1D 2D 3D 4D 5D

Floating Point

Integer

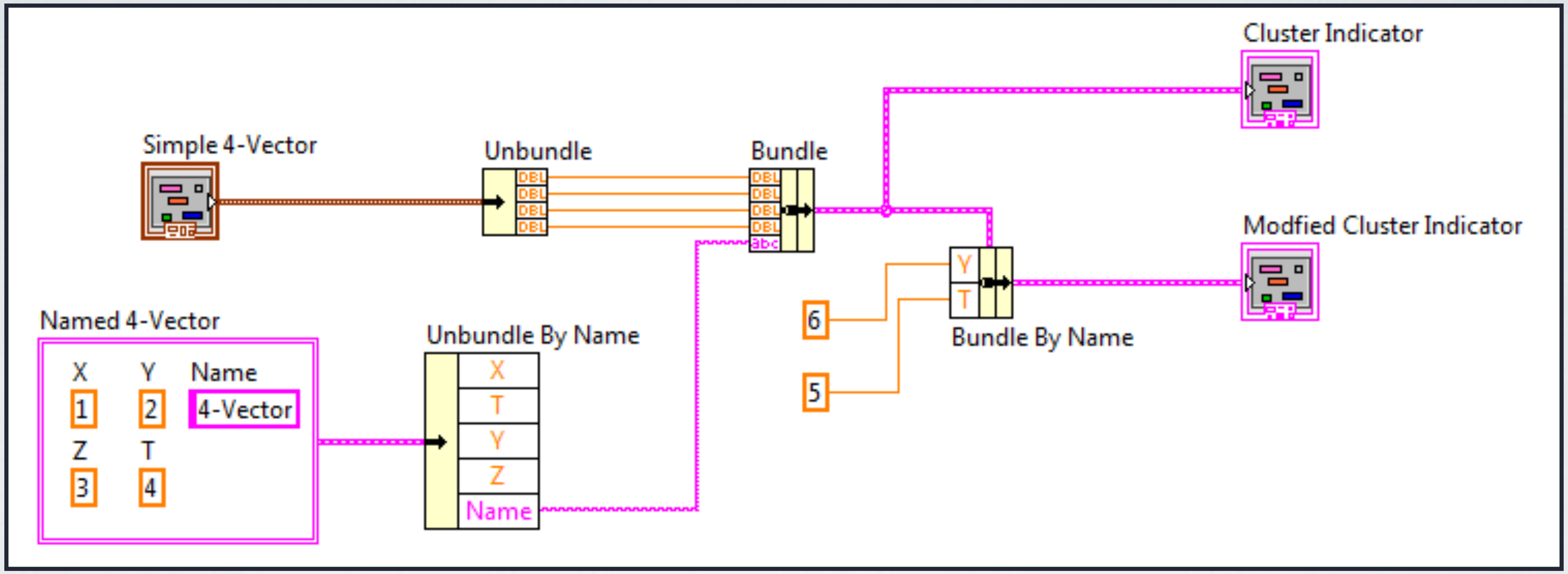
String



CLUSTERS

- ✑ *In **G**, **CLUSTERS** are collections of data that have **different types**.*
- ☞ Like arrays, the **elements** of clusters can be defined ***statically*** (i.e. at compile time), or generated ***dynamically*** using **functions** or **VIs**.
- ☞ Clusters elements are stored **in order** and can therefore be retrieved according to their **positions** within the cluster.
- ☞ Cluster elements may also have a **unique** associated **identifiers**, which can also be used to retrieve their values.

CLUSTERS



- The ***Unbundle*** function retrieves cluster elements based upon their positions
- The ***Unbundle By Name*** function retrieves elements based on their identifiers.
- The ***Bundle By Name*** function can be used to modify cluster element values.
- The ***Bundle*** function can be used to **add new** elements to a cluster.

DEMONSTRATION

Working with **Arrays and Clusters** in LabView

SUB-DIAGRAMS

- Implementation of **conditional** or **looping** structures in a graphical programming language like ***G*** requires **isolation of subsets** of the block diagram.

✑ *The isolated subsets are referred to as **SUB-DIAGRAMS**.*

☞ Sub-diagrams are defined by **enclosing** elements of the block diagram within a **graphical boundary**.

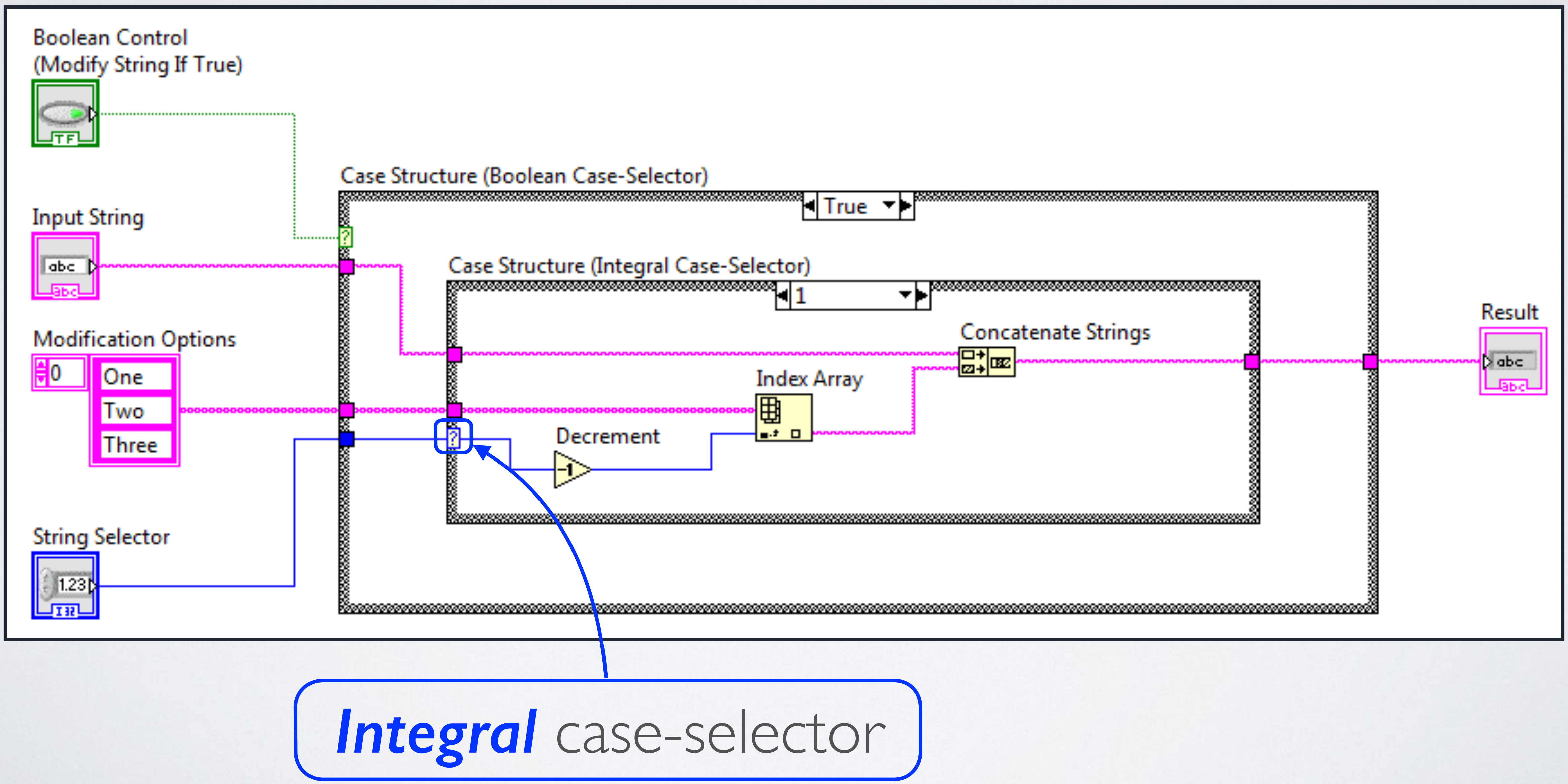
☞ Wires that **cross** the boundary of a sub-diagram must pass through a **terminal** as they do so.

CONDITIONAL STRUCTURES

- ***Conditional branching*** *in LabView is accomplished using* ***CASE STRUCTURES****.*
- Case structures comprise **several** sub-diagrams containing **different** assemblages of programatic elements.
- *The sub-diagram that is executed is controlled by a* ***value*** *supplied to a* ***CASE-SELECTOR*** *terminal provided by the case structure.*
- The ***value*** that is supplied **must** be ***boolean*** or ***integral***.

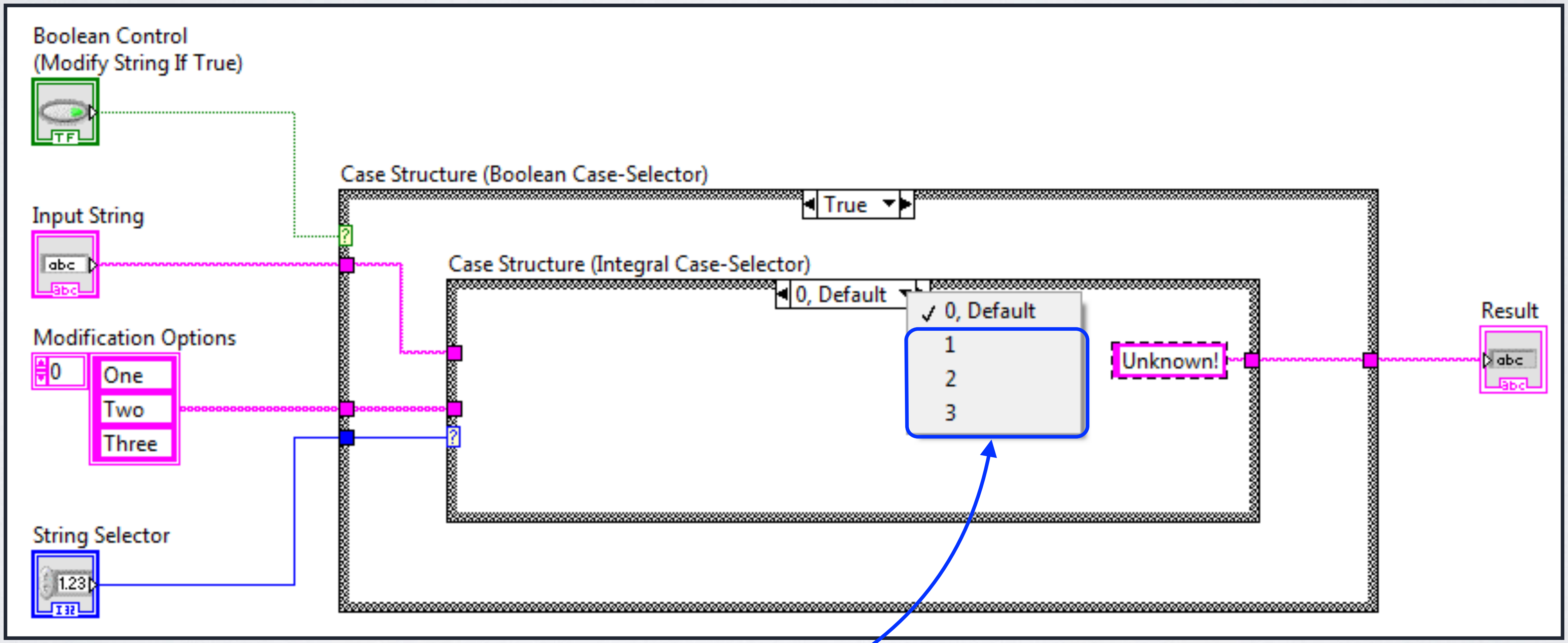
CASE STRUCTURE EXAMPLE

Integral case-selector is valid **and**, ***boolean*** is **True**.



CASE STRUCTURE EXAMPLE

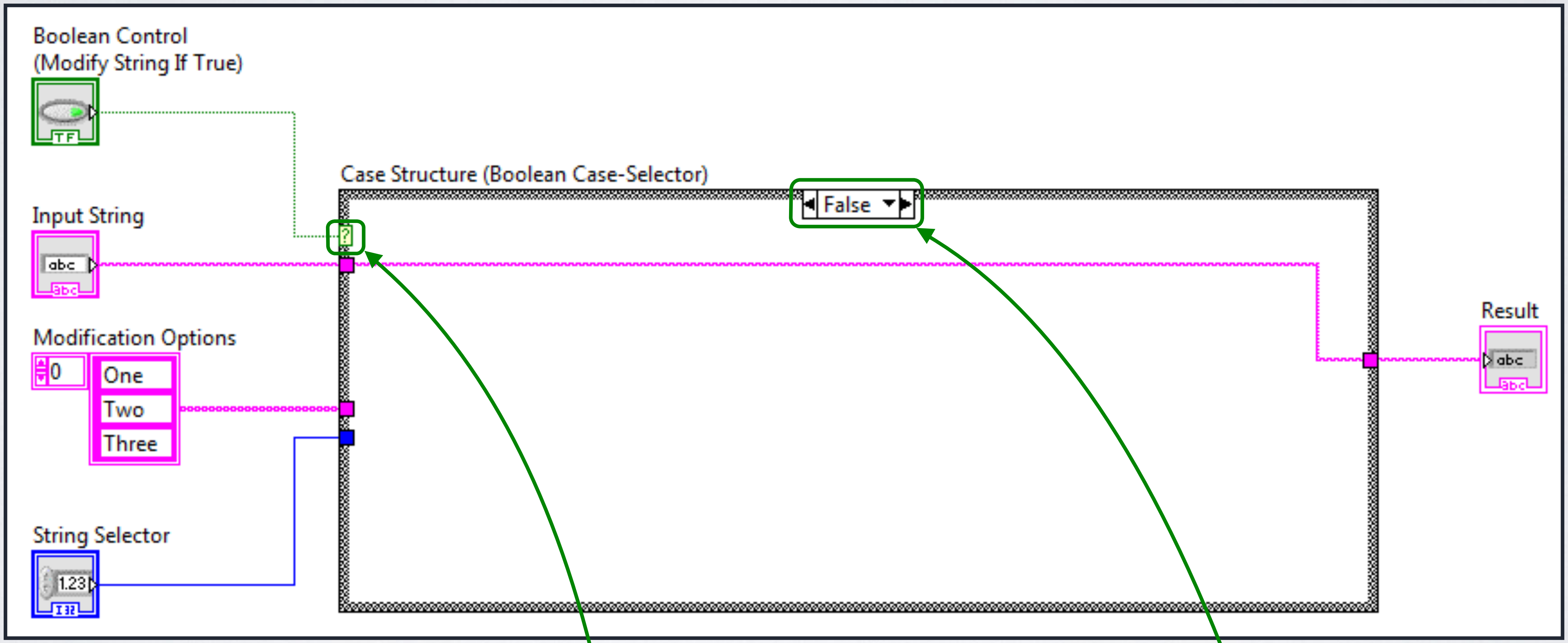
Integral case-selector is **invalid**, so **default** case executes.



Valid *integral* values

CASE STRUCTURE EXAMPLE

Boolean case-selector is **False**. String **not** modified.



Boolean case-selector

False *boolean* value

DEMONSTRATION

Working with **Case Structures** in LabView

FLOW CONTROL STRUCTURES

- The ***G*** language provides control structures including **for-loops** and **while-loops** that permit repeated execution of a subset of the elements that constitute a VI.

☞ In order to interact with **real-world** hardware systems LabView implements an **asynchronous** execution model.

- To handle asynchronicity, ***G*** also defines a number of **precisely timed**, **strictly sequential** or **event driven** control structures.

☞ **Independent** components of a VI execute in **parallel** if possible.

FOR-LOOPS

- For-loop structures repeatedly execute the sub-diagram elements that they enclose a **fixed** number of times.

☞ For-loop structures define one **external *count*** terminal that **may receive** the fixed number of loop iterations to perform.

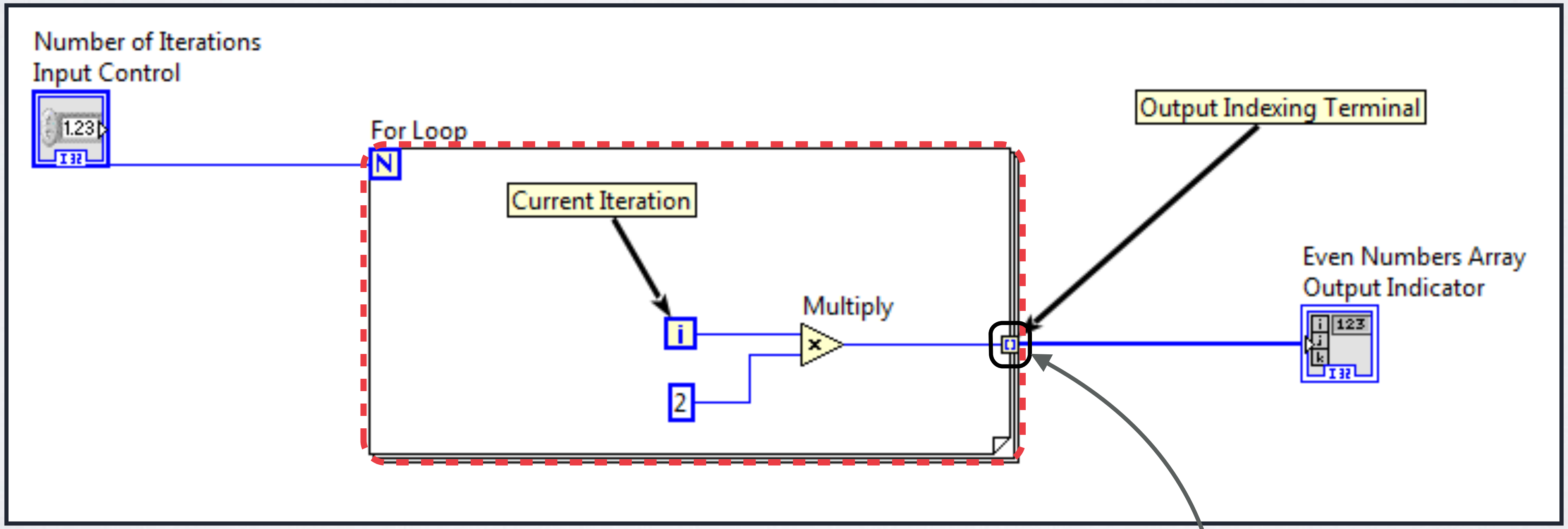
☞ For-loop structures define one **internal *iteration*** terminal that **provides** the current iteration to the logic **within** the loop structure.

INDEXING TERMINALS

- ✑ *A for-loop structure will **automatically** iterate over **all** elements of an array if a wire carrying array-type data is connected to an **INDEXING TERMINAL** on the structure.*
- ☞ An indexing **input** terminal is created **by default** whenever a wire carrying array-type data **enters** a **for-loop** structure.
- ☞ An indexing **output** terminal is created **by default** whenever a wire **exits** a **for-loop** structure.

FOR LOOP EXAMPLE

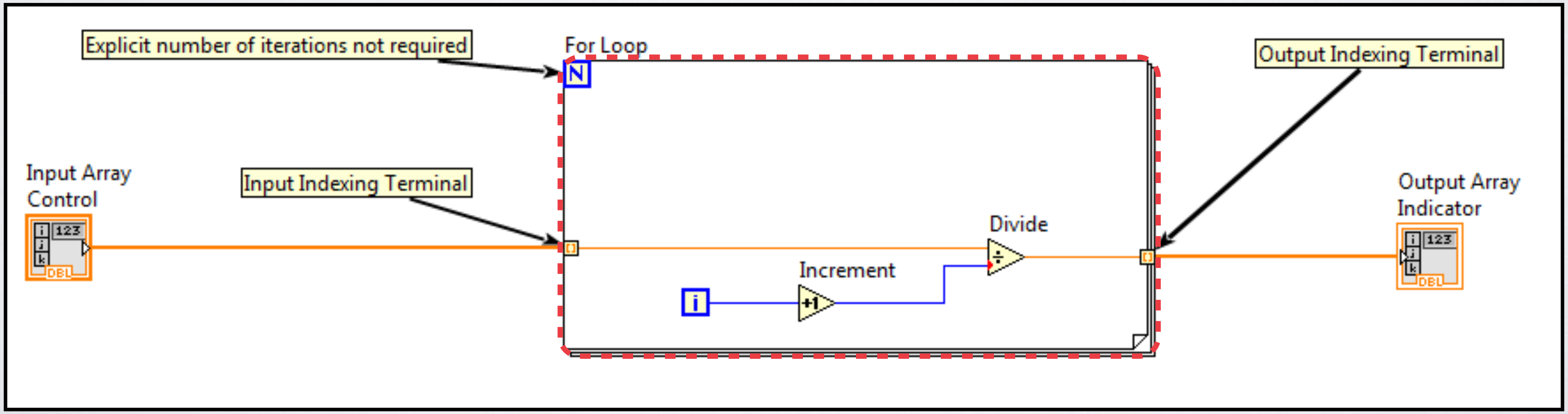
- **Explicit** specification of required **iteration multiplicity** entails connection of a wire carrying ***integral*** data to the **count** (**N**) terminal on the boundary of the **for-loop sub-diagram**.



☞ Indexing terminals appear **hollow**

FOR LOOP EXAMPLE

- Wires carrying **array-type** data **automatically** create indexing input terminals when they transit the sub-diagram **boundary of a for-loop**. In such cases, **explicit** specification of the iteration multiplicity is **not** required.



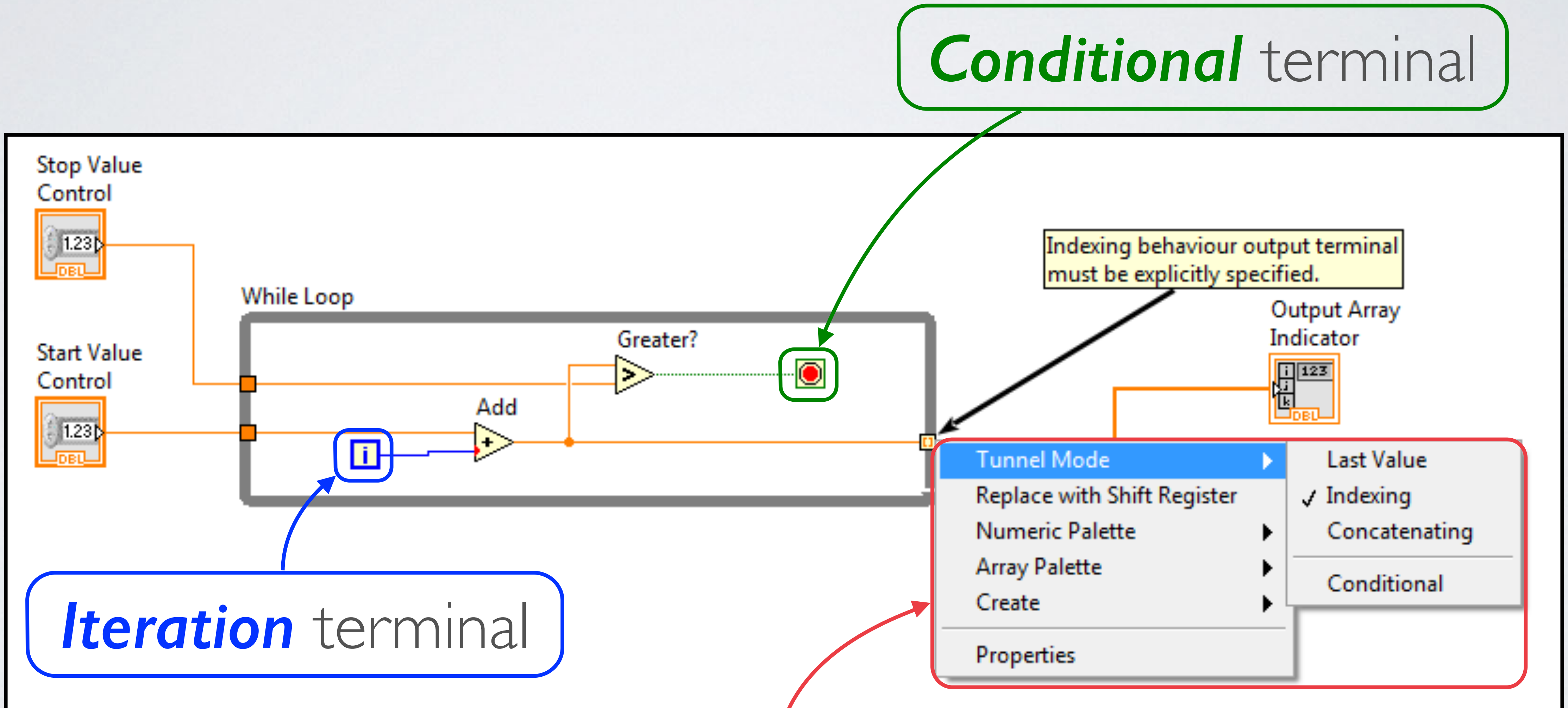
DEMONSTRATION

Working with **For-Loops** in LabView

WHILE-LOOPS

- While-loops repeatedly execute the elements of their associated sub-diagram **until** a ***boolean*** termination criterion is fulfilled.
- While loops provide two internal terminals:
  - The ***conditional*** terminal that **requires** a ***boolean*** input corresponding to the evaluated termination criterion.
  - The ***iteration*** terminal that **provides** the current iteration to the logic **within** the loop structure.

WHILE-LOOP EXAMPLE



☞ While loops do **not** automatically generate indexing terminals on their sub-diagram boundaries.

DEMONSTRATION

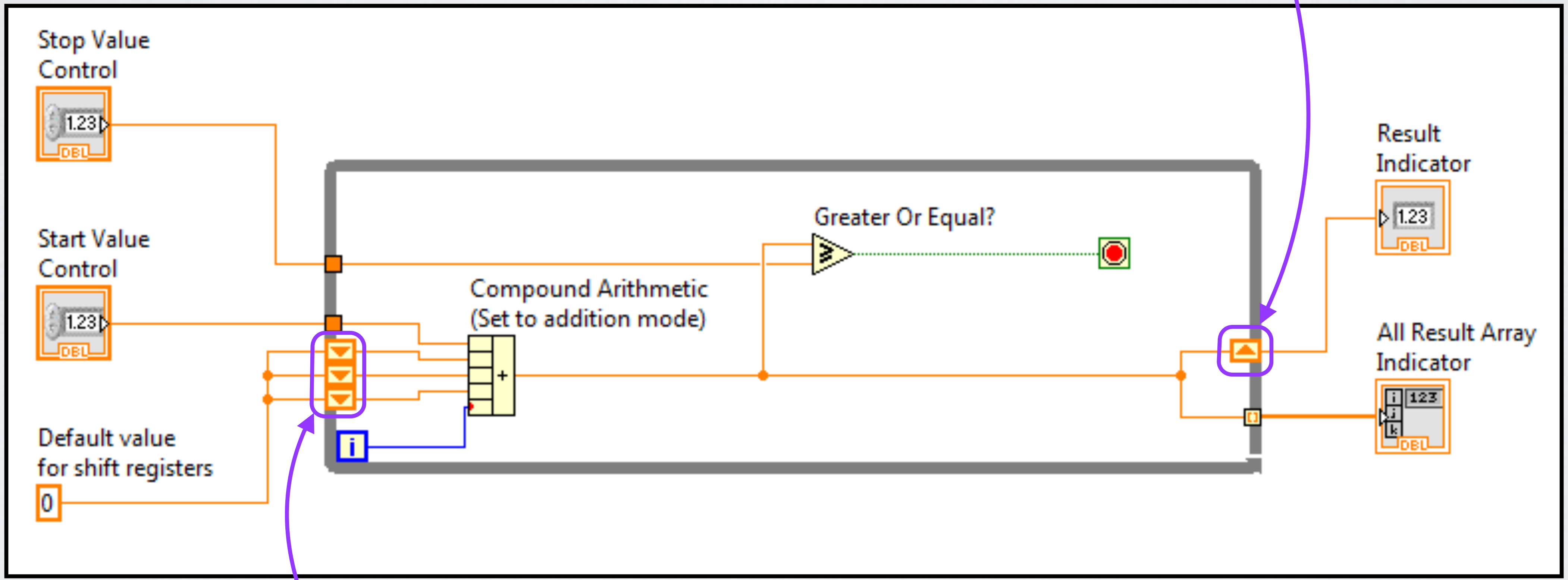
Working with **While-Loops** in LabView

SHIFT REGISTERS

- ***Shift Registers*** *are used to pass data from one loop iteration to the next.*
- **Terminals** on the boundaries of loop-structure sub-diagrams can be **converted** into shift registers using the LabView GUI.
- **Multiple** shift registers can be used to obtain data from **several** previous iterations.
- If a shift register could refer to a **non-existent** iteration, a **default** input value **must** be specified.

SHIFT REGISTER EXAMPLE

Output shift register passes data to subsequent iterations.



Input shift registers receive data from previous iterations.

DEMONSTRATION

Working with **Shift Registers** in LabView

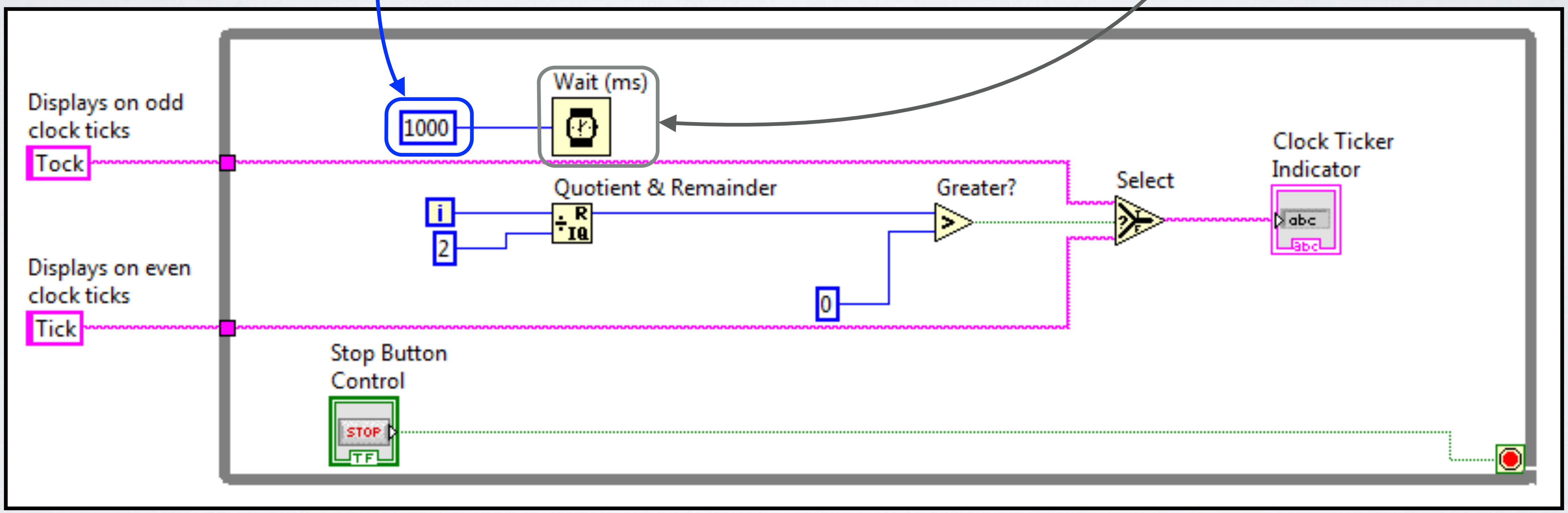
BASIC LOOP TIMING

- LabView provides a ***Timed Loop*** structure that facilitates fine-grained **control** and **monitoring** of the execution duration of the associated sub-diagram.
- Coarse **limitation** of the iteration **rate** can be achieved by including the the LabView ***Wait*** function in the loop sub-diagram.
- The ***Wait*** function pauses execution of the loop sub-diagram for a specified number of **milliseconds** and subsequent iterations will not proceed unless that interval has expired.

BASIC LOOP TIMING EXAMPLE

Integral input to the ***Wait*** function specifies a 1000ms pause in execution.

Wait function pauses execution for a specified number of milliseconds



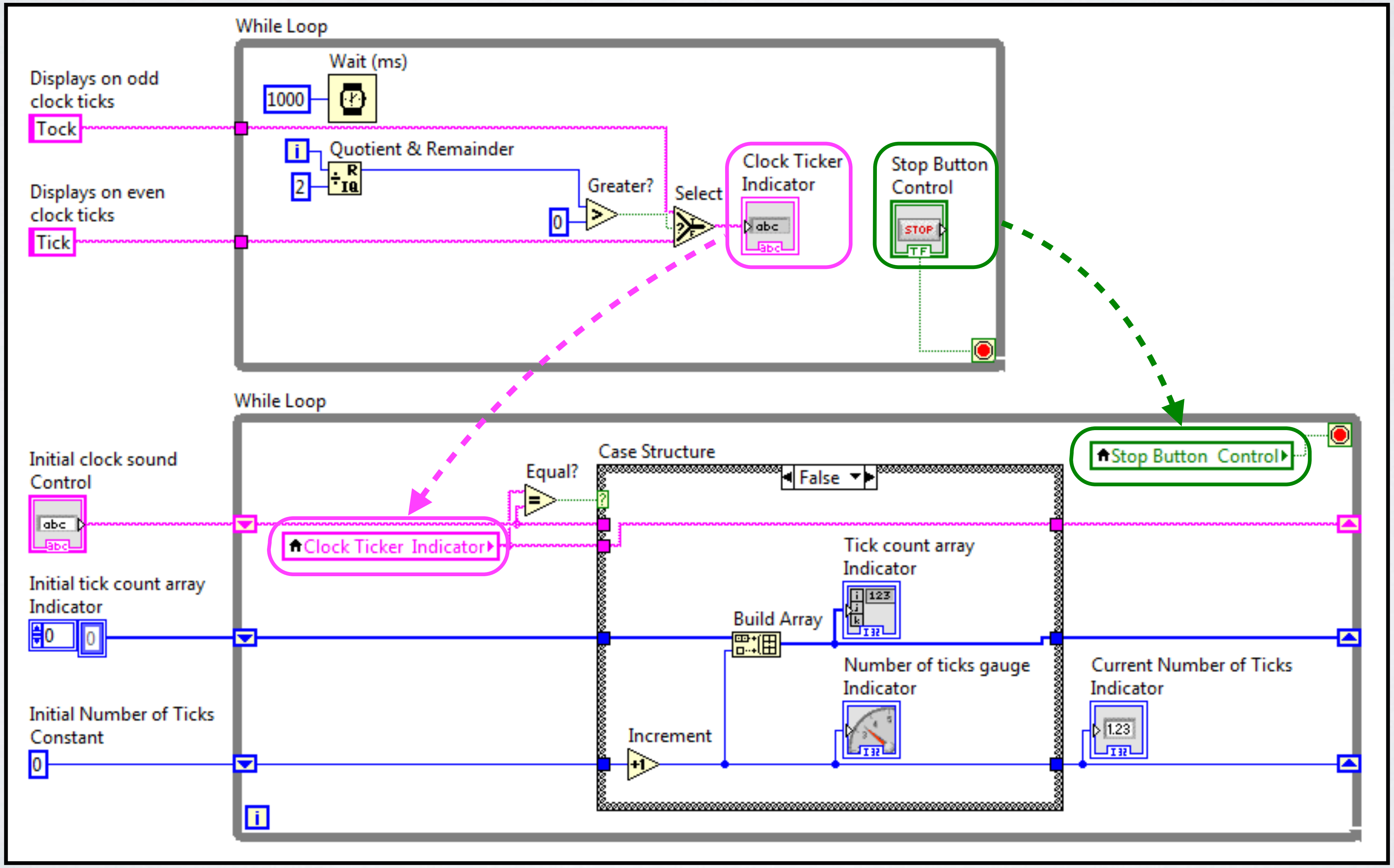
☞Each iteration of the while-loop takes **at least** 1 second. Operations may require additional time to execute.

VARIABLES

- ✑ ***VARIABLES in LabView** are used to transmit data between programatic elements "wirelessly".*
- Variables can transmit data between **elements** and **sub-diagrams** in the **same VI**, between **different VIs** and *even* between **different computers** across a network.
- ☞ Variables avoid the **explicit serial execution** of connected elements associated with the "wired" data flow model.
- ☞ Variables are used to enable transmission of data between sub-diagrams that should execute in **parallel**.

VARIABLES EXAMPLE

- The **values** displayed by **indicators** can be **defined** as variables.



DEMONSTRATION

Basic Loop Timing and **Working with Variables**

RECOMMENDED READING

Learn LabVIEW Online Video Tutorials

(http://www.ni.com/academic/students/learn-labview/)

LabVIEW Basics Online Reference

(http://www.ni.com/getting-started/labview-basics/)

LECTURE 13 HOMEWORK

- Review the ***Recommended Reading*** items listed on the previous slide.

Continue to refine your **final project proposal** and begin working on your **final project** once it is approved.