

# 2006 **NI Technical Symposium**

PROFESSIONAL DEVELOPMENT SERIES FOR ENGINEERS



# Trucos para Mejorar el Desempeño de LabVIEW



[ni.com](http://ni.com)





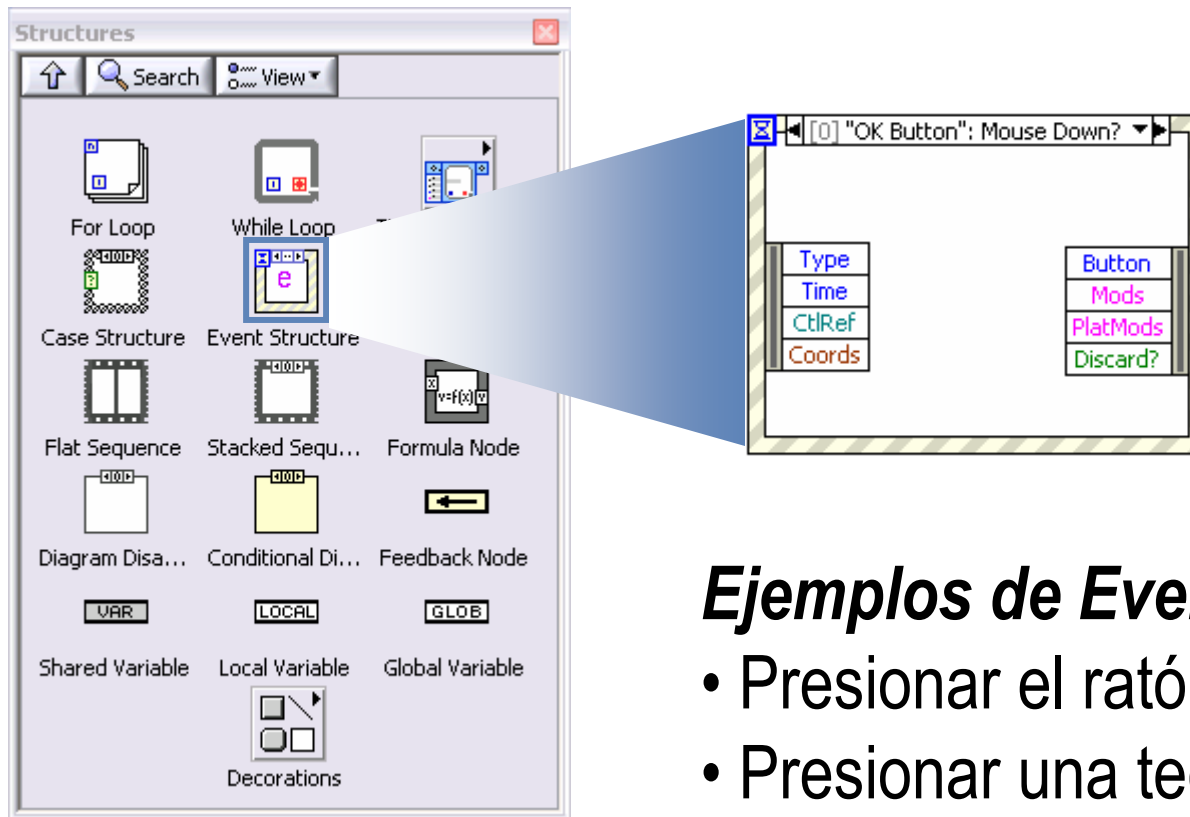
# Contenido

- Programación Manejada por Eventos
- Administración de Memoria
  - Carga Dinámica de VIs
  - Conversiones de Tipo
  - Reubicación de Memoria
- Desempeño de Interfaz del Usuario
- Análisis Automatizado de Código Estático





# Programación con Eventos en LabVIEW



## *Ejemplos de Eventos de LabVIEW*

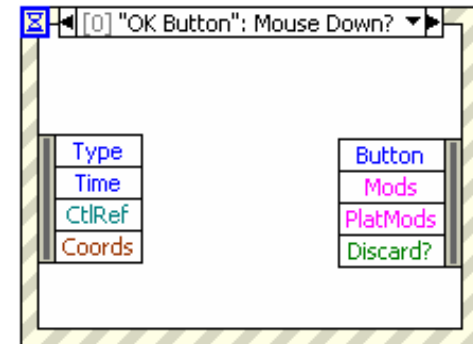
- Presionar el ratón
- Presionar una tecla del teclado
- Interacción del usuario con la pantalla principal





# Estructura de Eventos en LabVIEW

- ¿Cómo funciona?
  - **Duerme hasta ocurrir un evento**
  - Ejecuta el caso del evento apropiado
  - No realiza ciclos implícitamente
- Ventajas
  - **No hace *polling* (no se sobrecarga al procesador)**
  - Garantía de captación de todos los eventos
  - Los casos de eventos se ejecutan en el orden en que ocurren





# Eventos de Notificación versus Eventos de Filtración

- Los eventos se dividen en dos categorías: **Filtración** y **Notificación**
- Los eventos de notificación son las notificaciones después de ocurridos los hechos
- Los eventos de filtración le ayudan a cambiar los datos del evento a medida que ocurren o bien, descartar el evento por completo
  - Eventos de teclas, eventos de ratón, eventos del menú, y el evento para cerrar la ventana principal son todos eventos de filtración.
- Ejemplos de eventos de filtración:
  - Hacer controles tipo string para mapear mayúsculas a minúsculas
  - Crear diálogos de confirmación cuando el usuario intente cerrar la ventana principal





# Ejemplo: Estructura del Evento LabVIEW



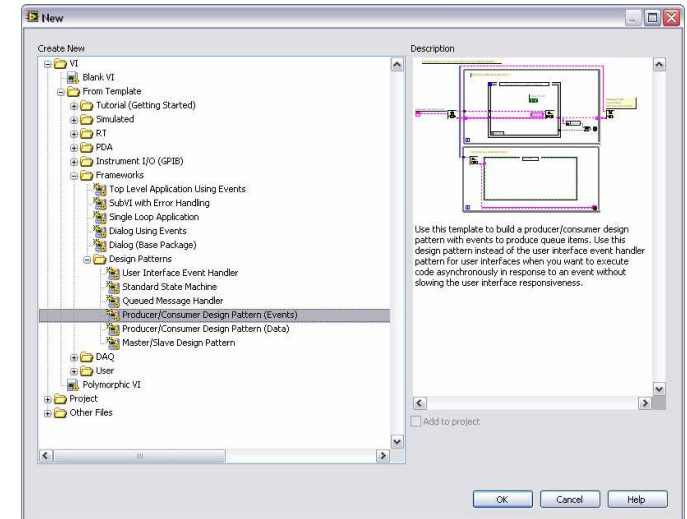
[ni.com](http://ni.com)





# Utilice Plantillas para la Estructura del Evento

- Máquina de Estados
- Arquitectura de Ciclos Paralelos
- Manejador de Mensajes
- Productor/Consumidor (Datos)
- ***Manejador de Eventos de Interfaz del Usuario***
- ***Productor/Consumidor (Eventos)***
- Combine o expanda las arquitecturas como se necesite





# Pasos para Administrar la Memoria

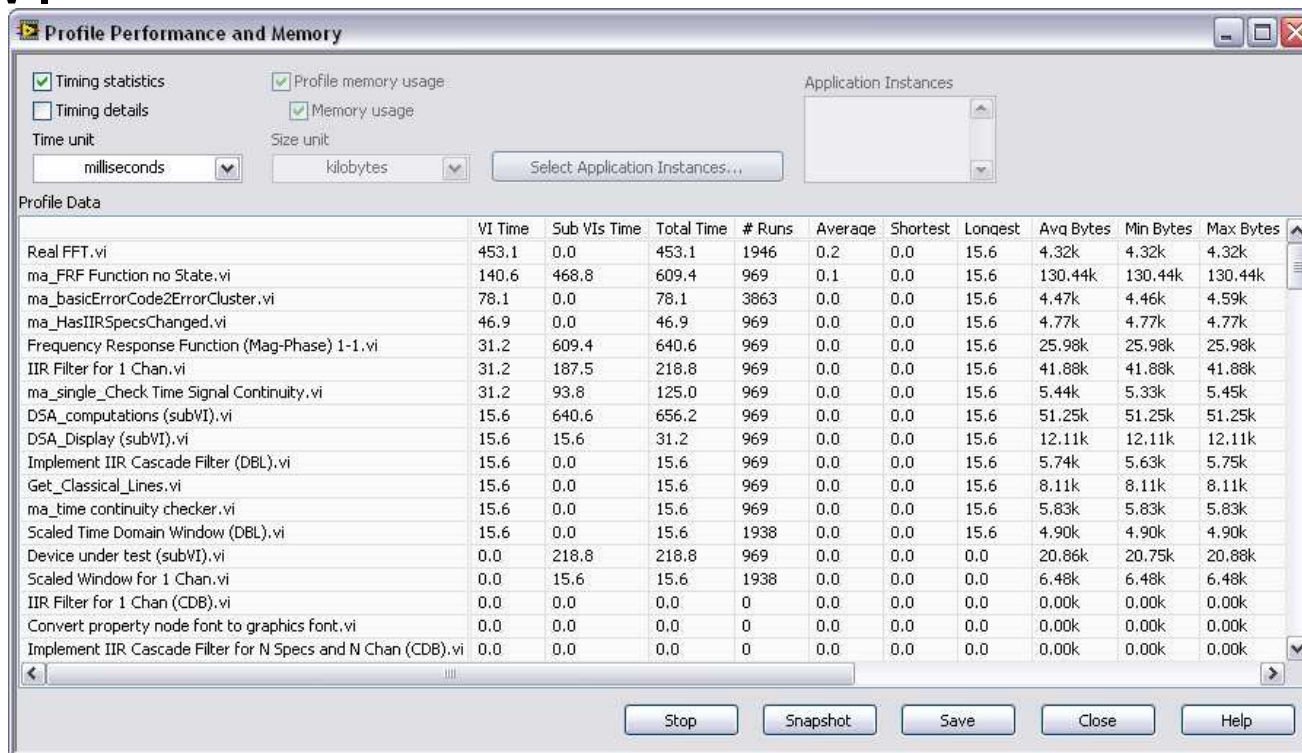
1. Separe los VIs en subVIs
2. Analice los VIs para localizar problemas
3. Cargue dinámicamente los subVIs cuando sea apropiado
4. Reduzca copias de datos y reubicación de memoria
5. Minimice las actualizaciones del panel frontal





# Analice el Desempeño y Memoria del VI

- Analice el tiempo de ejecución y uso de memoria del VI



Profile Data	VI Time	Sub VIs Time	Total Time	# Runs	Average	Shortest	Longest	Avg Bytes	Min Bytes	Max Bytes
Real FFT.vi	453.1	0.0	453.1	1946	0.2	0.0	15.6	4.32k	4.32k	4.32k
ma_FRF Function no State.vi	140.6	468.8	609.4	969	0.1	0.0	15.6	130.44k	130.44k	130.44k
ma_basicErrorCode2ErrorCluster.vi	78.1	0.0	78.1	3863	0.0	0.0	15.6	4.47k	4.46k	4.59k
ma_HasIIRSpecsChanged.vi	46.9	0.0	46.9	969	0.0	0.0	15.6	4.77k	4.77k	4.77k
Frequency Response Function (Mag-Phase) 1-1.vi	31.2	609.4	640.6	969	0.0	0.0	15.6	25.98k	25.98k	25.98k
IIR Filter for 1 Chan.vi	31.2	187.5	218.8	969	0.0	0.0	15.6	41.88k	41.88k	41.88k
ma_single_Check Time Signal Continuity.vi	31.2	93.8	125.0	969	0.0	0.0	15.6	5.44k	5.33k	5.45k
DSA_computations (subVI).vi	15.6	640.6	656.2	969	0.0	0.0	15.6	51.25k	51.25k	51.25k
DSA_Display (subVI).vi	15.6	15.6	31.2	969	0.0	0.0	15.6	12.11k	12.11k	12.11k
Implement IIR Cascade Filter (DBL).vi	15.6	0.0	15.6	969	0.0	0.0	15.6	5.74k	5.63k	5.75k
Get_Classical_Lines.vi	15.6	0.0	15.6	969	0.0	0.0	15.6	8.11k	8.11k	8.11k
ma_time continuity checker.vi	15.6	0.0	15.6	969	0.0	0.0	15.6	5.83k	5.83k	5.83k
Scaled Time Domain Window (DBL).vi	15.6	0.0	15.6	1938	0.0	0.0	15.6	4.90k	4.90k	4.90k
Device under test (subVI).vi	0.0	218.8	218.8	969	0.0	0.0	0.0	20.86k	20.75k	20.88k
Scaled Window for 1 Chan.vi	0.0	15.6	15.6	1938	0.0	0.0	0.0	6.48k	6.48k	6.48k
IIR Filter for 1 Chan (CDB).vi	0.0	0.0	0.0	0	0.0	0.0	0.0	0.00k	0.00k	0.00k
Convert property node font to graphics font.vi	0.0	0.0	0.0	0	0.0	0.0	0.0	0.00k	0.00k	0.00k
Implement IIR Cascade Filter for N Specs and N Chan (CDB).vi	0.0	0.0	0.0	0	0.0	0.0	0.0	0.00k	0.00k	0.00k

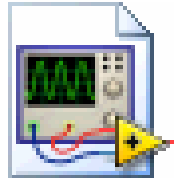


**Tools»Profile»Performance and Memory**

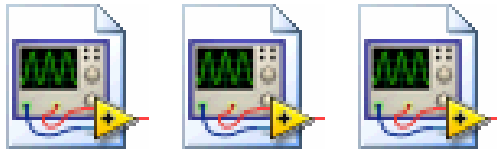


# Jerarquía Típica de un VI

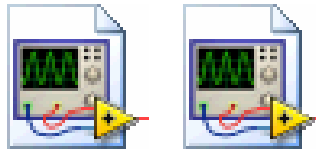
*VI de Alto Nivel*



*VIs de Adquisición de Datos*



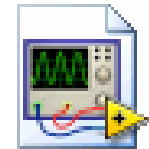
*VIs de E/S a Archivos*



*VI de Calibración*



*VI para Guardar Reportes*



**Tareas Comunes. Cargar con VI.**

**Tareas Proco Frecuentes.  
Cargar según se requiera.**



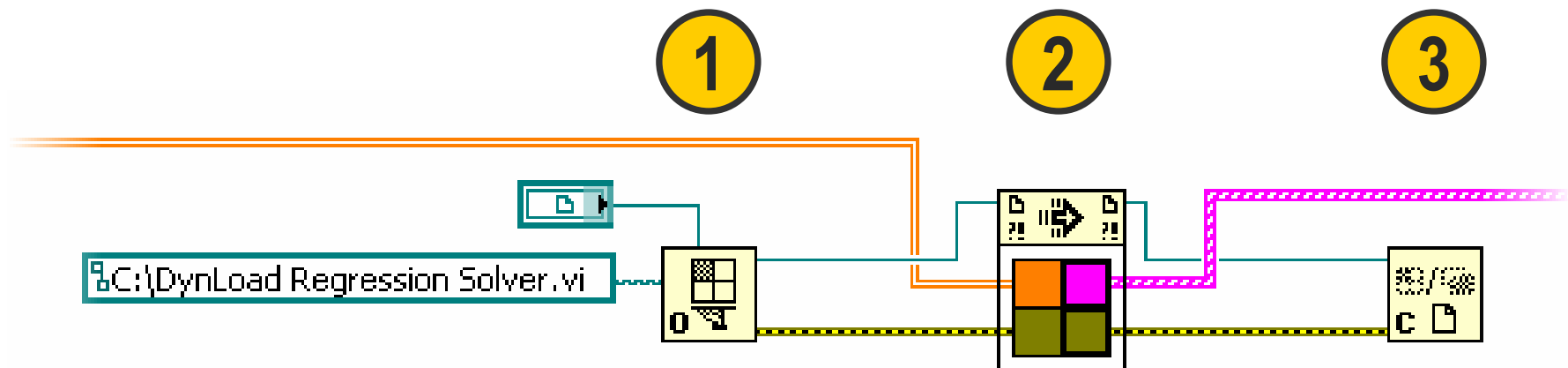
ni.com





# Carga Dinámica Programática de VI

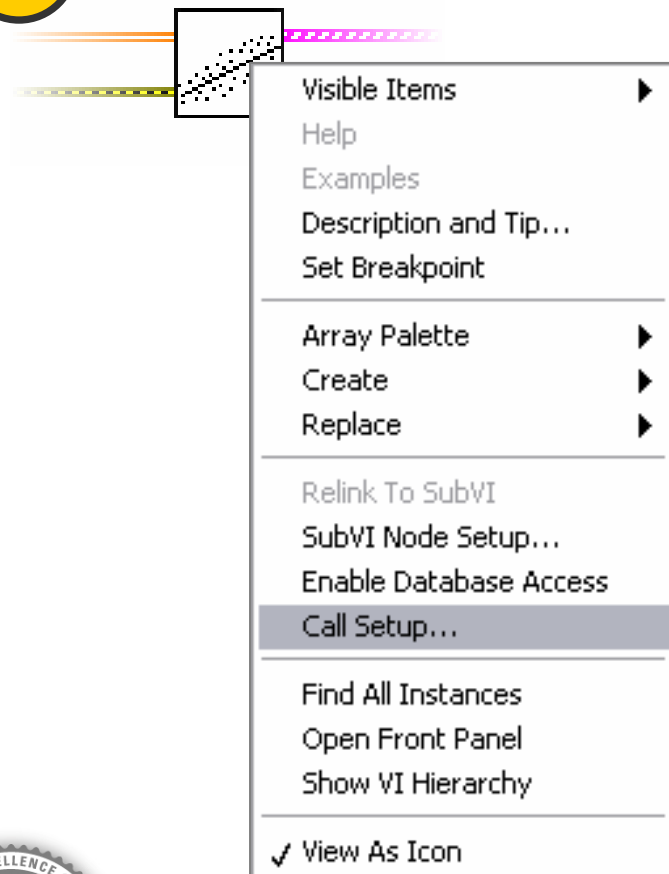
- 1 Abrir Referencia al VI
- 2 *Call by Reference Node*
- 3 Cerrar Referencia al VI



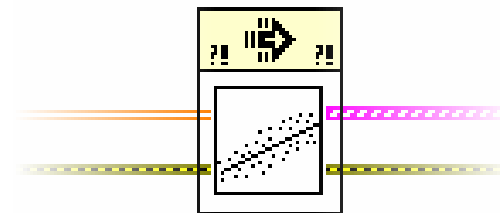
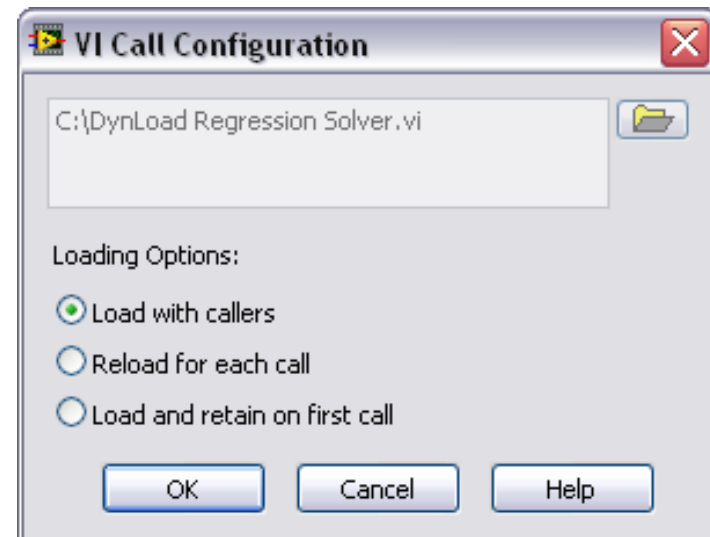


# Cargar Configurable – LabVIEW 8.20

1



2





# Ejemplo: Cargar Dinámicamente los VIs



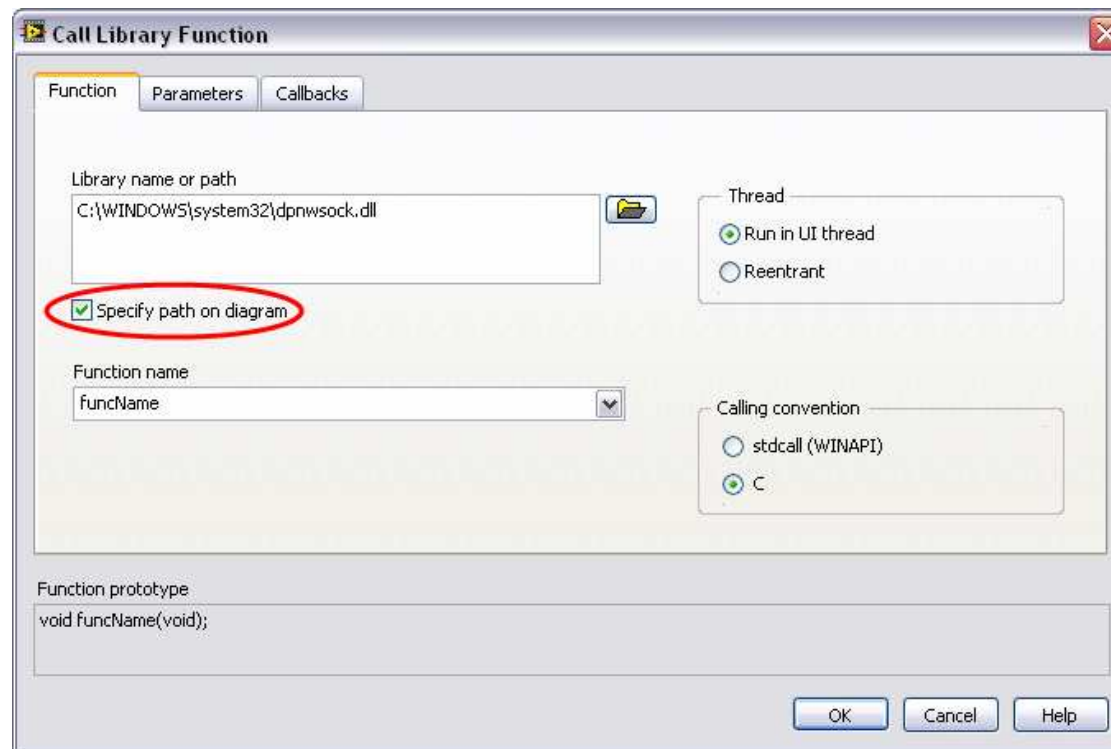
[ni.com](http://ni.com)





# Mejoras al *Call Library Node*

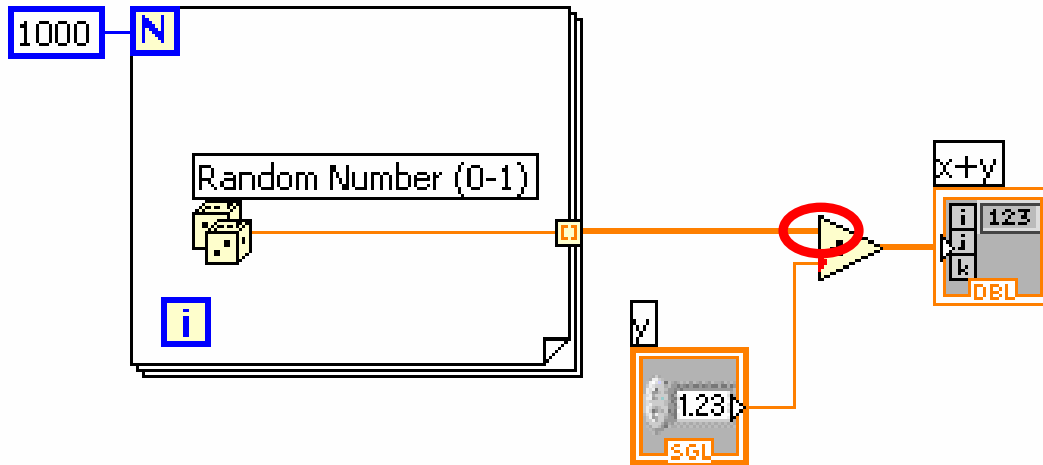
***Carga y descarga*** dinámicamente los DLLs en LabVIEW 8.20.





# Conversiones de Tipo

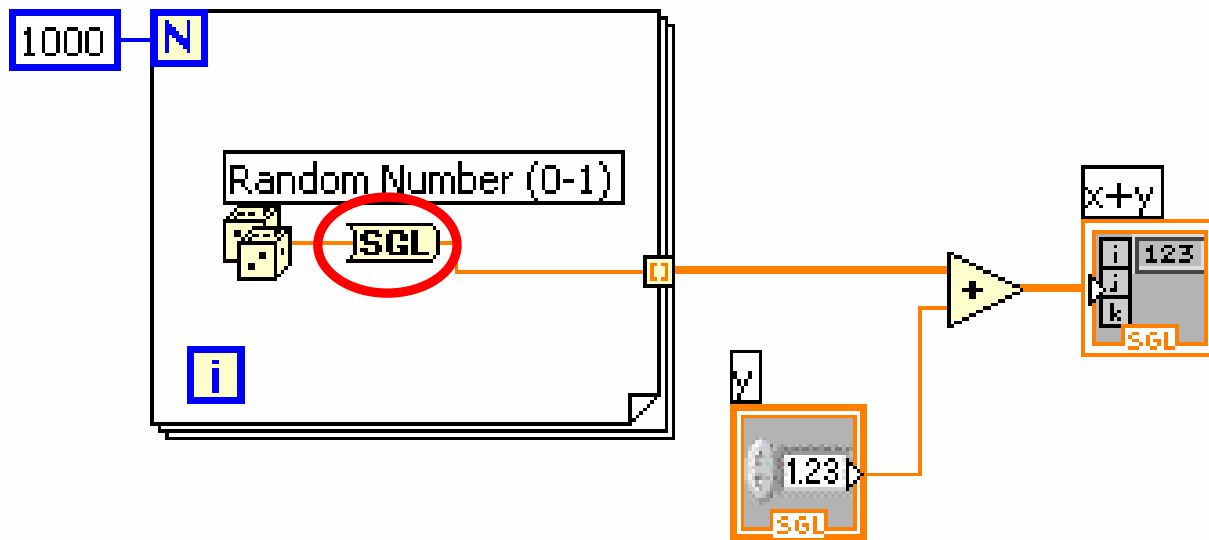
- Los puntos de coerción significan dos tipos de datos numéricos diferentes unidos al mismo punto.
  - Coerciones en LabVIEW requieren copia de datos
  - Coerciones que involucran grandes arreglos requieren de mucha memoria





# Conversión de Datos Recomendada

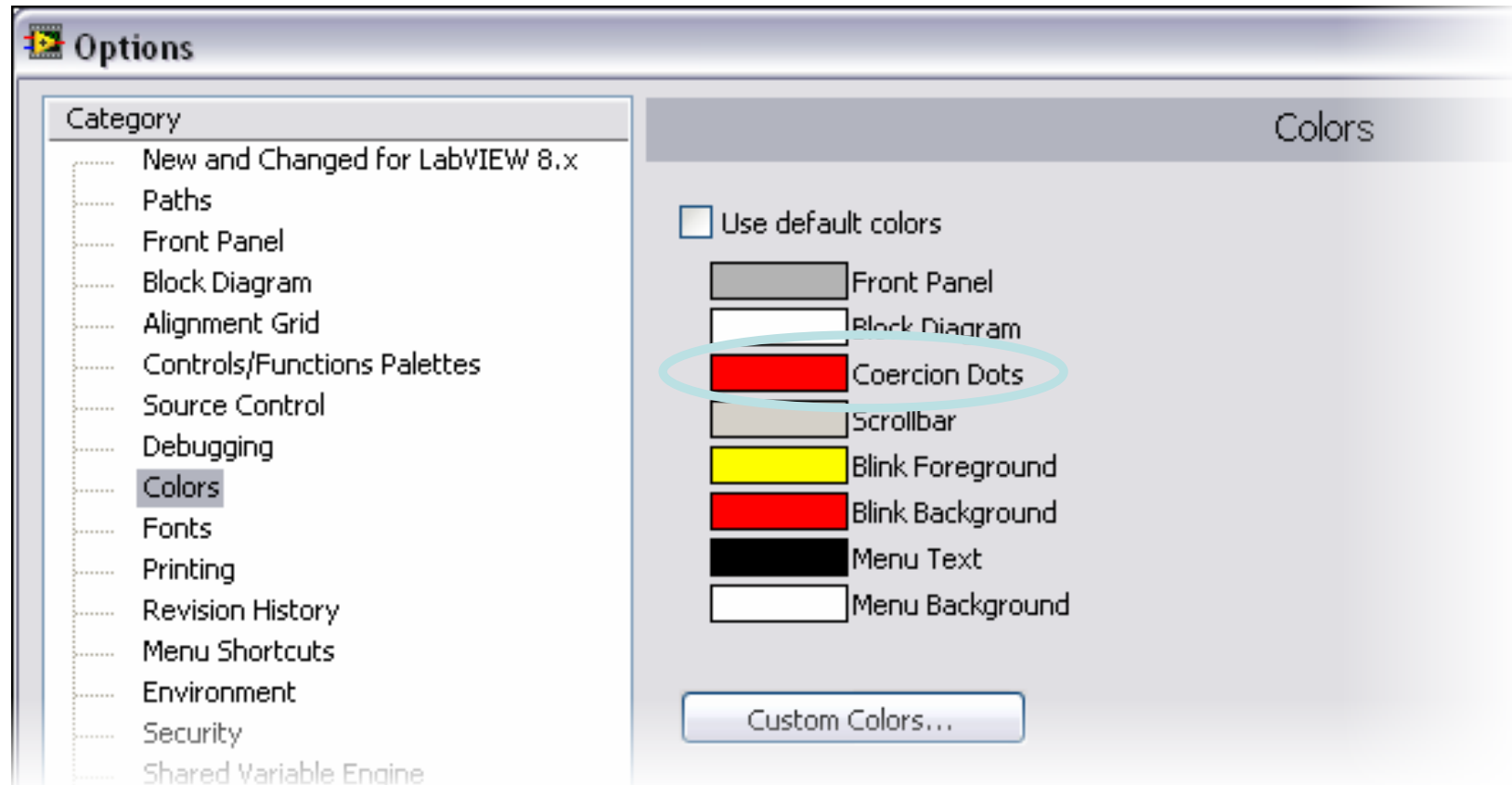
- La mejor solución es convertir un número aleatorio a medida que es creado
  - Evita la conversión de un gran búfer de datos





# Configurando Puntos de Coerción

*Tools>>Options>>Colors to change coercion dot color*





# Construyendo Arreglos y *Strings*

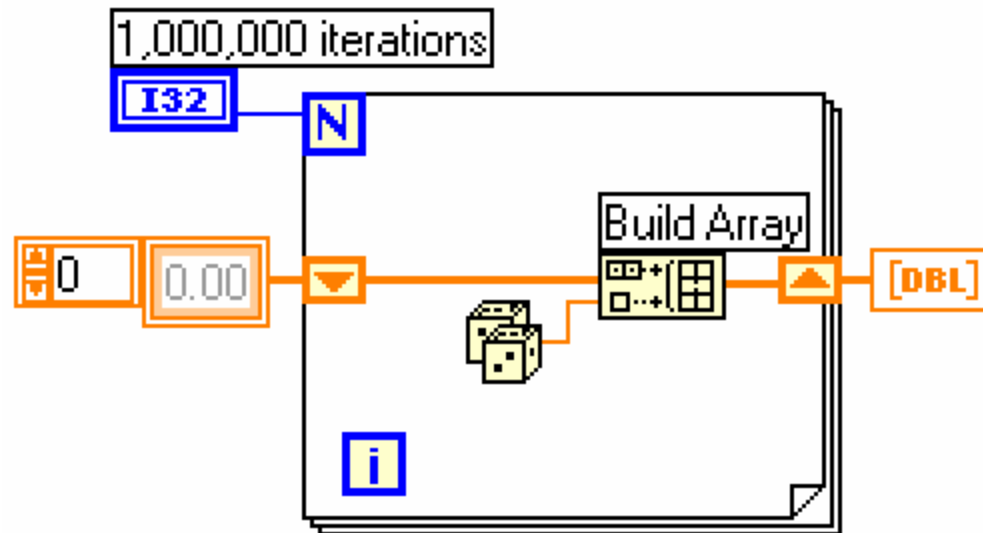
- Reubicar memoria es costoso cuando se realiza muy seguido
- Deben limitarse las funciones que tienden a causar un reacomodo de memoria
  - Build array
  - Concatenate strings





# Construya un Arreglo en 18.7 Segundos

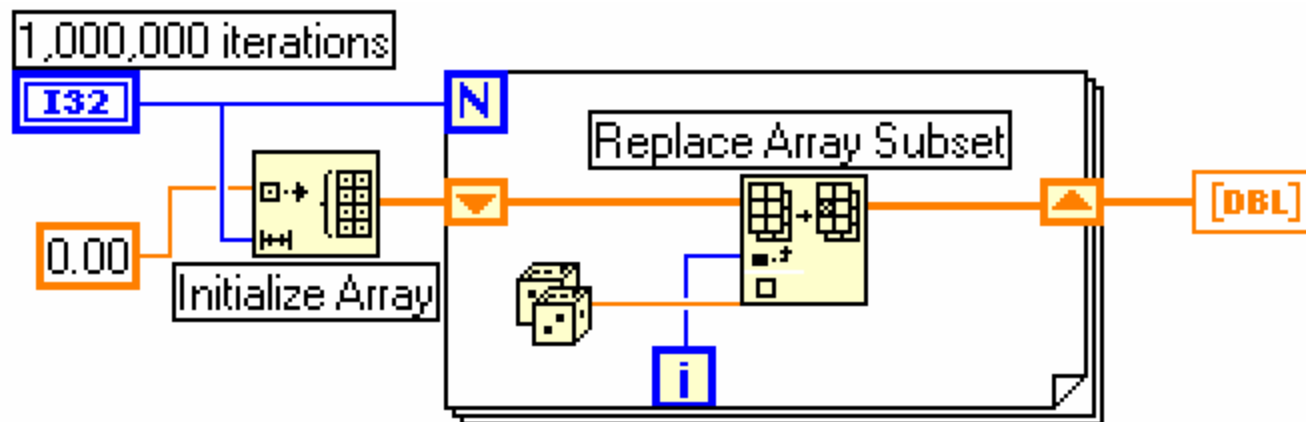
- Muy lento debido a que cada iteración involucra reubicar memoria





# Construya un Arreglo en 0.42 Segundos

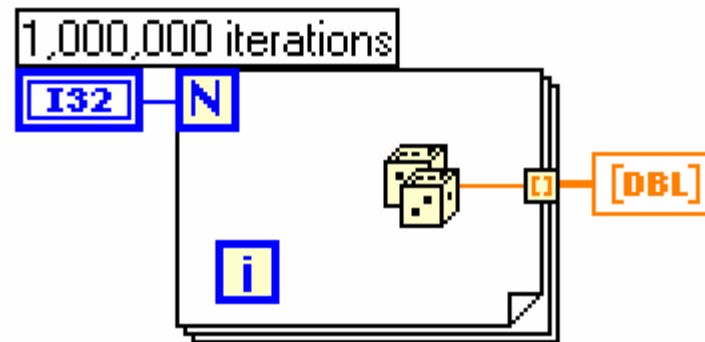
- Mucho más rápido ya que solo ubica memoria una vez





# Construya un Arreglo en 0.40 Segundos

- El método más rápido y también el más limpio





# Ejemplo: Asignación de la Memoria en LabVIEW



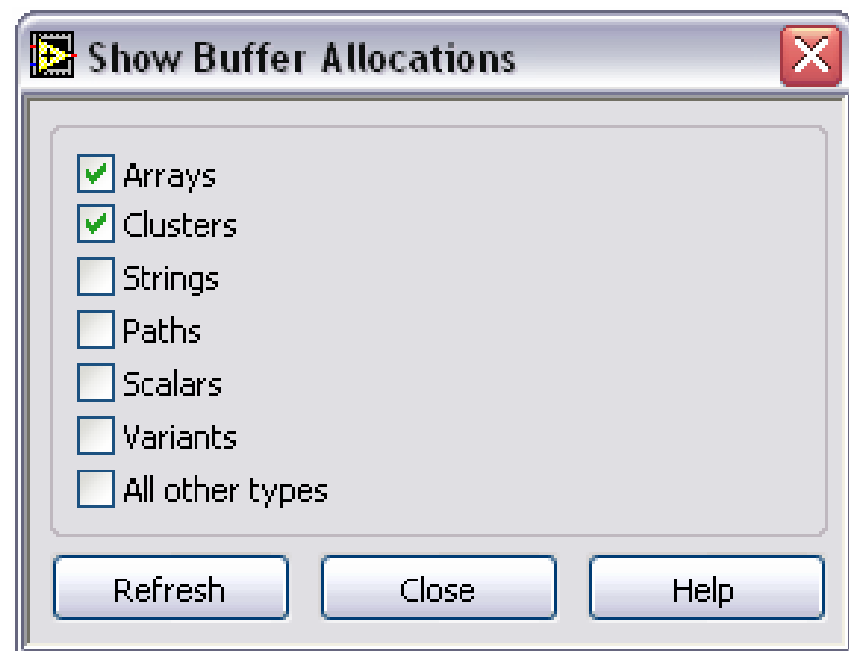
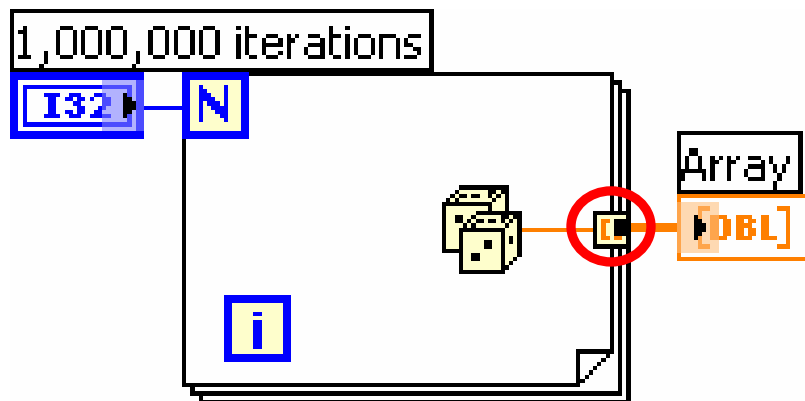
[ni.com](http://ni.com)





# Mostrar la Asignación del Buffer

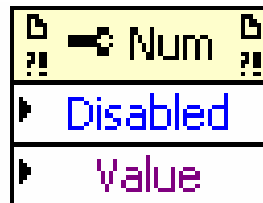
**Show Buffer Allocations** despliega los lugares donde ocurren las ubicaciones de la memoria





# Desempeño de la Interfaz del Usuario

- Los nodos de propiedades e invocación de métodos para controles e indicadores pueden alentar una aplicación
  1. Forzan un cambio de *threads* al *thread* de UI
  2. Puede forzar la actualización de UI al completar el proceso
  3. Cambio al *thread* original





# Ejemplo: Cambios de *Threads*



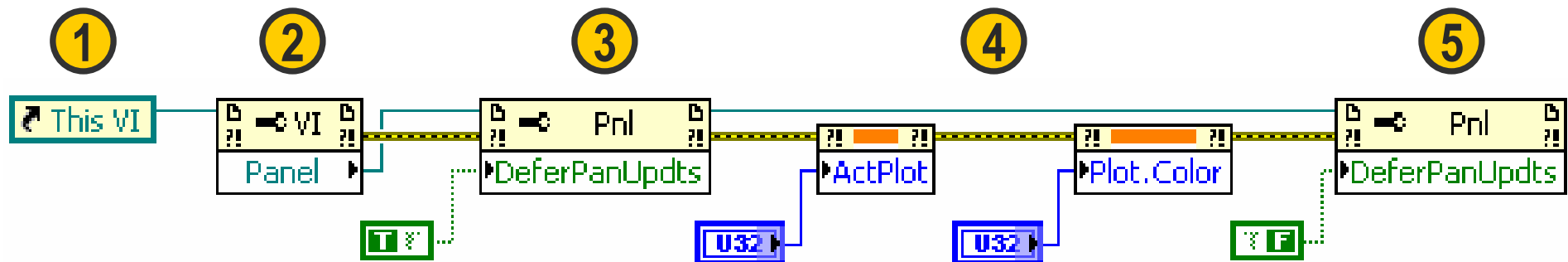
[ni.com](http://ni.com)





# Método Preferido de Actualización de la IU

Use ***Defer Panel Updates*** para deshabilitar el panel frontal durante múltiples actualizaciones



**1** Referencia al IV

**2** Referencia del Panel

**3** Deshabilitar el Panel

**4** Hacer Cambios

**5** Habilitar el Panel

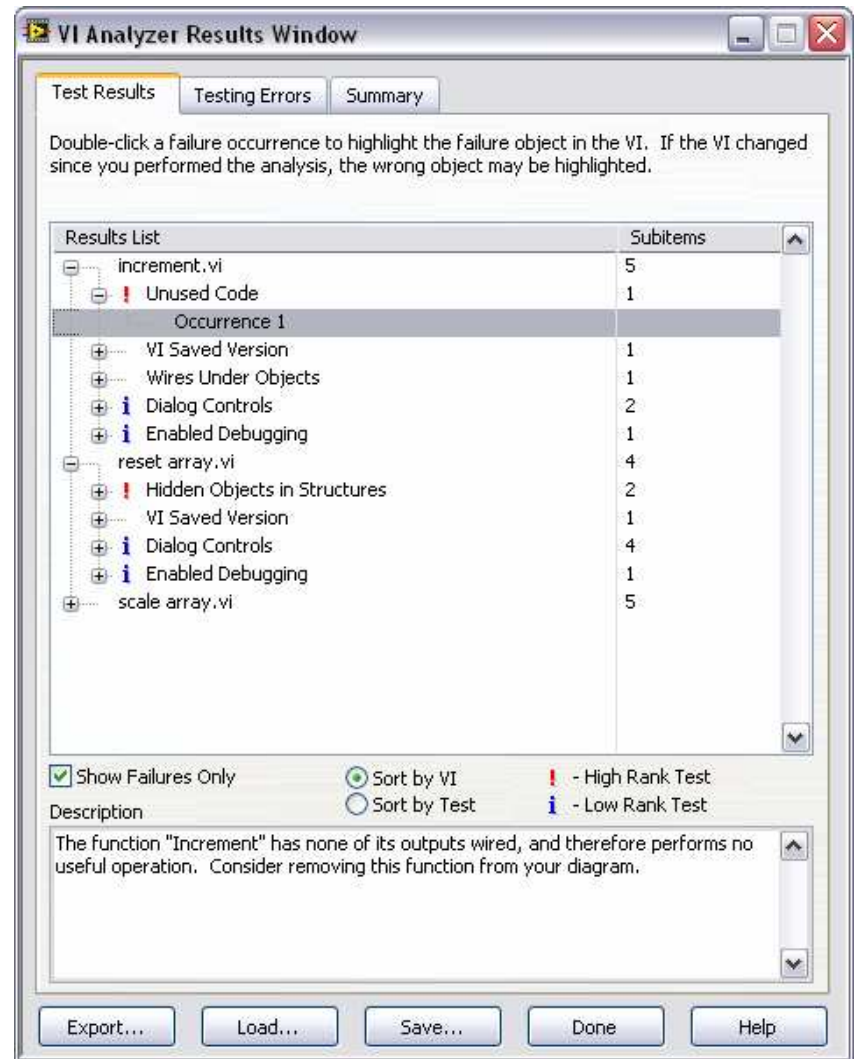




# LabVIEW VI Analyzer

- Herramienta adicional para LabVIEW\*
- Automatice el análisis de código con más de 60 pruebas configurables
  - Desempeño
  - Estilo
- Inspeccione fallas de forma interactiva
- Genere reportes personalizados

*\* Incluido en el NI Developer Suite Core*





# Pruebas de Desempeño con VI Analyzer

- Arreglos y *Strings* en Ciclos
- Puntos de Coerción
- Depuración habilitada
- Espera en Ciclos While
- Tamaño del VI
- Valores Predefinidos del Arreglo
- Objetos Escondidos en Estructuras
- Variables Globales y Locales
- Código en desuso
- Terminales Conectadas en Subdiagramas
- Uso de la Estructura de Secuencia





# Ejemplo: VI Analyzer Toolkit



[ni.com](http://ni.com)





# Resumen

- Utilice la programación basada en eventos para minimizar el uso del procesador
- Cargue dinámicamente los VIs para mejorar el desempeño de la memoria y tiempo inicial de carga
- Reduzca la reasignación de memoria e intercambio de *threads* para mejorar la velocidad de ejecución
- Localice conflictos de desempeño con la inspección automatizada del código

