

# Técnicas de Desenvolvimento para Sistemas Real Time com LabVIEW

**André Oliveira**

Engenheiro de Vendas

**Rodrigo Schneiater**

Engenheiro de Aplicações

NIDays 2011

# Agenda

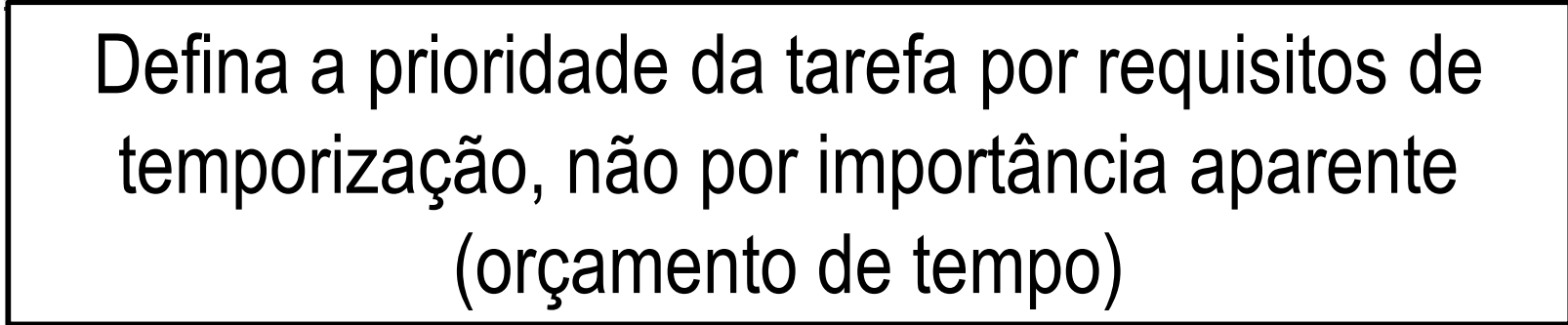
- Projeto
  - Entendendo Modelos de Agendamento de Prioridades
  - Utilize Padrões de Projeto
  - Separe Tarefas
- Desenvolvimento
  - Evitando Jitter
  - Comunicação com a Interface de Usuário
  - Usando “Shared Variables” Efetivamente
- Distribuição
  - Mude as Definições de Distribuição
  - Realize Benchmark Corretamente
  - Minimize o uso de CPU

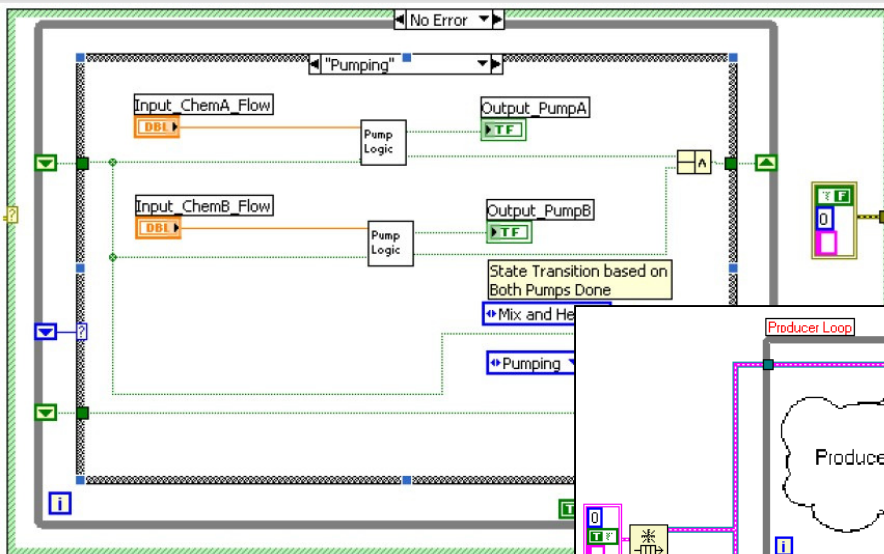
# Terminologia

- **Threads:** Executando concorrentemente tarefas no Sistema Operacional
  - **Thread Priority:** Classificação relativa de uma thread
    - O LabVIEW Real-Time associa prioridades de OS e VI (ou estrutura temporizada) para threads
  - **Sistema de Execução:** Agrupamento de threads específico do LabVIEW
- **Scheduler:** Atribui threads para executar na(s) CPU(s)
  - **Fila de execução:** Fila de threads preparadas para agendamento
  - **Preemptive vs Round-Robin:** Agendamento baseado em classificação ou FIFO

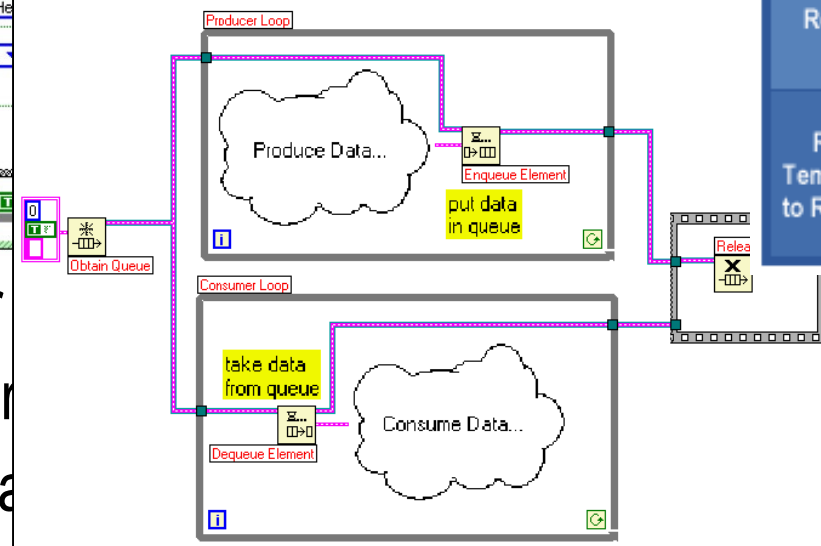
# Uma Nota Sobre Agendamento

- Regras Básicas de Agendamento RT
  1. Realizar agendamento preemptivo entre threads de prioridades diferentes
  2. Realizar agendamento round-robin entre threads de prioridades iguais
  3. Threads de VI's críticas sempre executam até a conclusão
- Uma thread “bloqueada” está impedida de executar por
  - recurso compartilhado
  - função de espera
  - operação de E/S

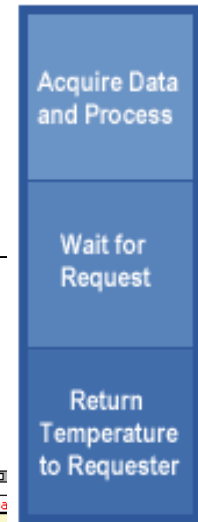




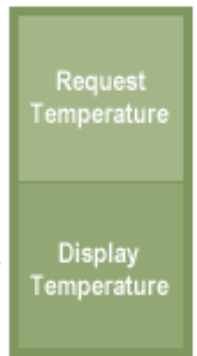
# Projeto



Server Program



Client Program

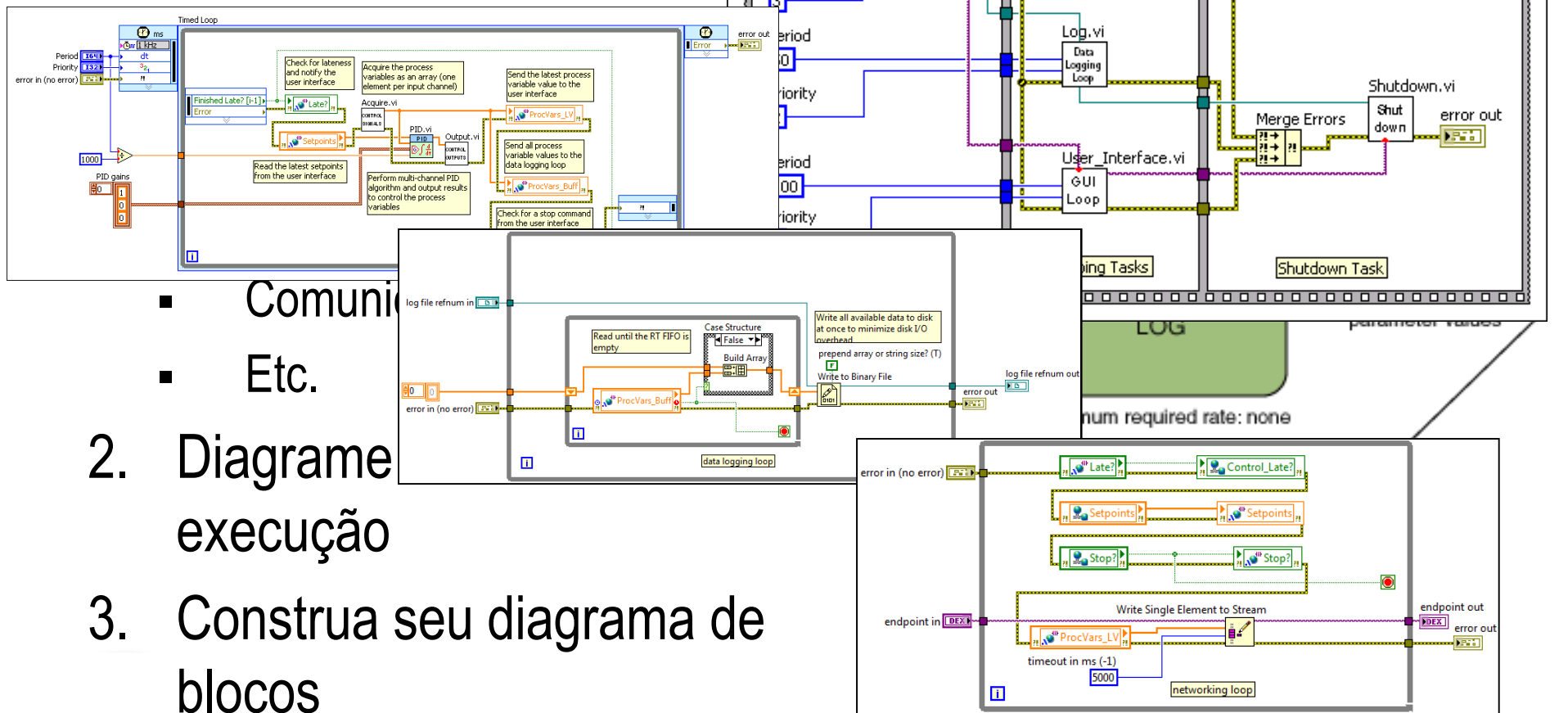


- Aumentar
- Simplificar
- Melhorar a
- Padrões de projeto recomendados NI (veja [ni.com/info](http://ni.com/info))
  - Máquina de Estados (statemachines)
  - Produtor/Consumidor (producerconsumer)
  - Cliente/Servidor (clientserver)

# Mais Terminologia

- **Determinismo:** Execução previsível de uma tarefa
- **Jitter:** Desvio entre o tempo de execução real e esperado
  - **Limitado vs Ilimitado:** Qualificação do jitter máximo como finito ou infinito, respectivamente
- **Tempo Crítico:** A tarefa de usuário de maior prioridade
  - Uma tarefa de tempo crítico tem alto determinismo
  - Uma tarefa de tempo crítico tem jitter baixo, limitado

# Seppure Tarefas

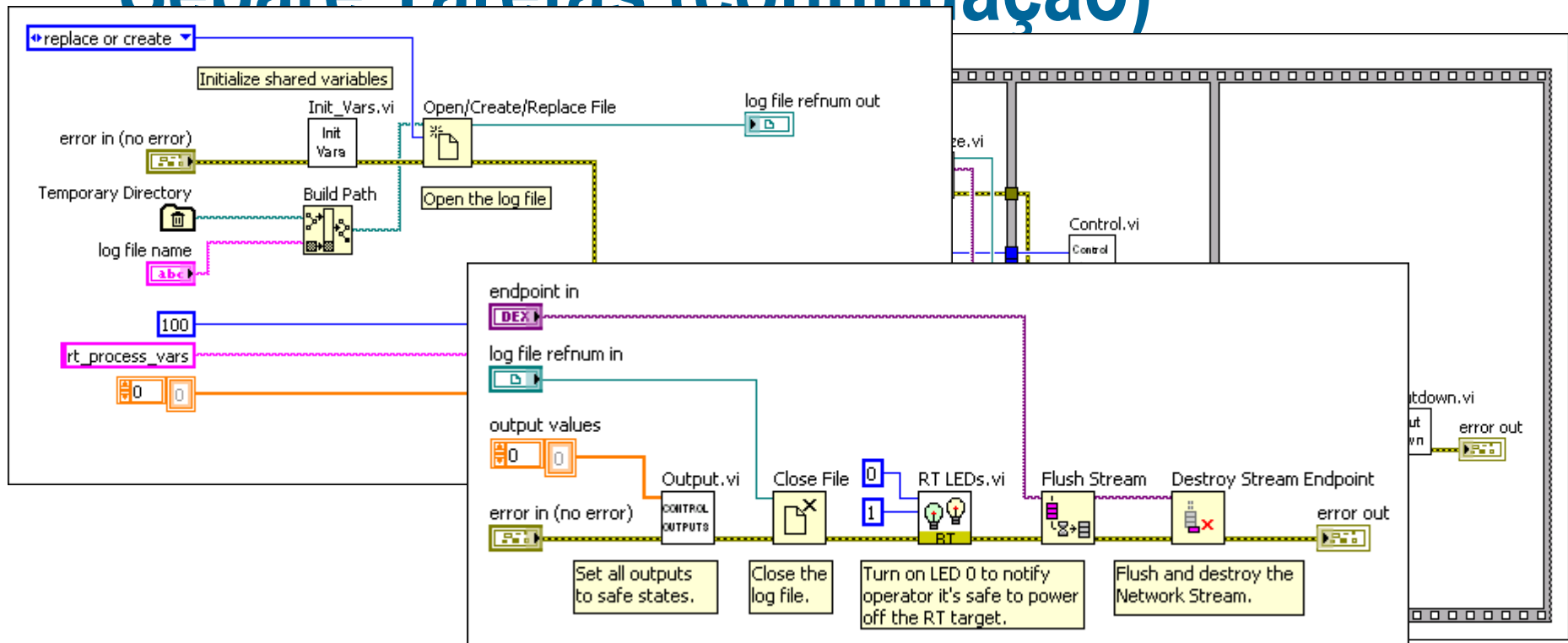


- Comuni
- Etc.

2. Diagrama  
execução
3. Construa seu diagrama de  
blocos



# Separe Tarefas (continuação)



4. Crie uma Rotina de Inicialização
5. Crie uma Rotina de Finalização Segura

# Evitando Jitter

Fontes de Jitter Comuns	Solução
Alocação de Memória	Pre-alocação
Tratamento de Referência	Durante Inicialização e Finalização
Comunicação Inter-thread	RT FIFOs
Recursos Compartilhados	Evitar recursos de HW, canais de E/S e VIs não-reentrantes em tarefas de tempo crítico
Escrita/Leitura de Arquivo	Evitar em tarefas de tempo crítico
Comunicação pela Rede	Evitar em tarefas de tempo crítico
Algoritmos Ilimitados	Fazer benchmark do código de tempo crítico!

# Crie um Esquema de Temporização

## While Loop

- Não possui habilidades de temporização incorporadas
- Estrutura de loop básica
- Multi-thread
- Pequeno overhead de CPU
- Pode ser usado com VI Priorities e Timed Loops

## Timed Loop

- Possui incorporados
  - Controle de temporização
  - Dados de temporização
  - Seleção de CPU
  - Controle de prioridade
- Projetado para aplicações com múltiplas taxas
- Single-thread
- Comparando, overhead de CPU maior que while loop [poucos  $\mu$ s]

Nota: While loops são recomendados para código ilimitado (E/S de arquivo, comunicações de rede, etc.)

# Selecione um Mecanismo de Temporização

- E/S Temporizada por Hardware
  - Frequency
  - Digital Edge Counter
  - Digital Change Detection
  - Signal from Task
- NI Scan Engine
- Funções de Espera
  - Wait
  - Wait until next multiple
- Clocks Internos – somente para estruturas temporizadas
  - Software-Triggered
  - kilohertz
  - Megahertz

Nota: O uso de uma única fonte de temporização para todas as estruturas temporizadas reduz o jitter introduzido pelo scheduler

# Criando um Orçamento de Tempo

Uso de CPU (%) =  $100 * \text{Soma}[\text{Loop 1, Loop 2,..., Loop } n]$ ,  
onde  $\text{Loop } n = (\text{Duração} / \text{Período})$

Tarefa/Loop	Duração (µs)	Período (µs)
Controle	400	1000
Monitoramento	3,000	10,000
Log	16,000	30,000

Tarefa/Loop	Duração (µs)	Período (µs)
Controle	400	1000
Monitoramento	3,000	25,000
Log	16,000	80,000

$$100 * (400/1,000 + 3,000/10,000 + 16,000/30,000) = 123\%$$

$$100 * (400/1,000 + 3,000/25,000 + 16,000/80,000) = 72\%$$

Nota: Conhecido como agendamento de taxa monotônica: Tarefas com taxas altas recebem prioridades maiores

# Comunique com a Interface de Usuário

Casos de Uso	Protocolo
Monitoramento dos valores atuais	Network-Published Shared Variable
Transmitir dados em alta velocidade do target RT para um computador host, enviando comandos para um target RT	Network Streams
Protocolos centrais, ponto de partida para protocolos personalizados como STM	TCP ou UDP
Comunicação determinística entre targets RT	Time-Triggered Shared Variable
Controle dinâmico de VIs RT	VI Server
Interfaces de usuário baseadas na Web, integração com tecnologias padrão da Web	Web Services

# Use os Recursos Certos

Casos de Uso	Protocolos Recomendados
Comando (um para um)	Funções Network Streams
Comando (um para vários)	Network-published shared variables
Comando (vários para um)	VIs de Web services ou network-published shared variables
Monitoramento (um para um)	Network-published shared variables
Monitoramento (um para vários)	VIs de Web services ou network-published shared variables
Monitoramento (vários para um)	Network-published shared variables
Transmissão contínua de dados (um para um)	Funções Network Streams
Transmissão contínua de dados (um para vários)	TCP ou UDP
Transmissão contínua de dados (vários para um)	TCP ou UDP



Supervisório

Comunicação  
não Crítica

Rotinas de Controle

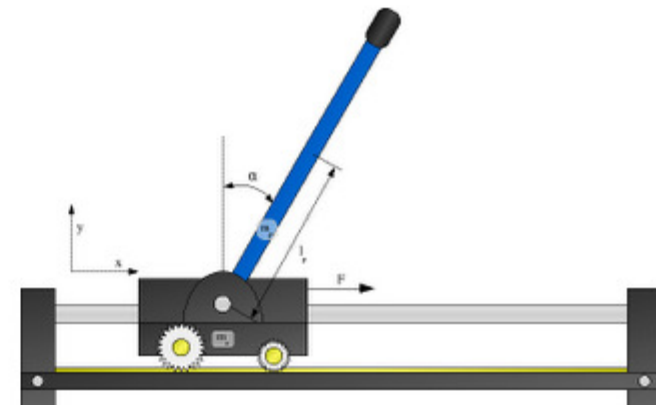
Real-Time  
Controller

Reconfigurable  
Chassis

I/O  
Modules



Temporização  
Crítica



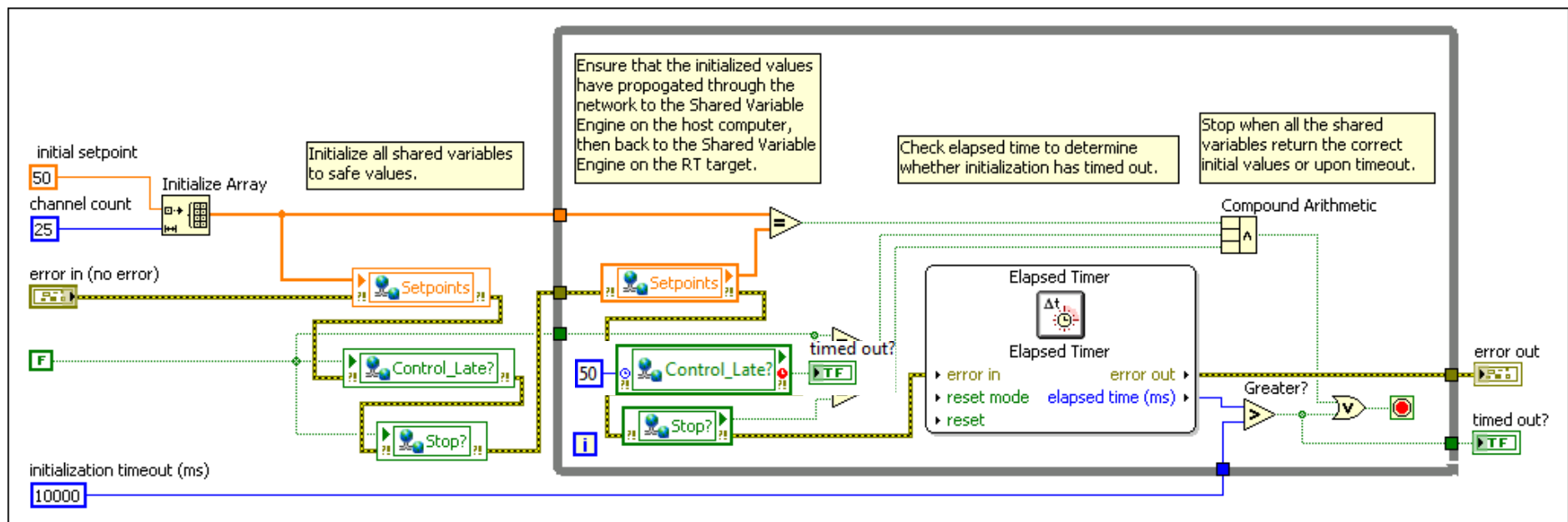


# Usando Shared Variables Efetivamente

- Limite o Uso de Shared Variable
  - Bom para aplicações pequenas com poucos canais
    - Single-process (local) e Networked
  - Combine canais de tipos idênticos em um array
- Utilize funções RT FIFO
  - Arquitetura escalável para grandes aplicações (somente local)
  - Modos Polling e blocking (Alto desempenho ou baixo uso de CPU, respectivamente)

# Usando Shared Variables Efetivamente

- Inicialize as Shared Variables
- Serialize a execução das Shared Variables
- Evite ler dados velhos das Shared Variables



# Usando Shared Variables Efetivamente

- Evite Bufferização Desnecessária
  - Previne perda de dados causada por atrasos temporários na rede
  - Não garante transferência de dados sem perdas
  - Consome mais CPU e memória
  - **Vem habilitada por padrão** em Network-Published Shared Variables
- Selecione um Host Adequado para as Shared Variables
  - Host RT simplifica acesso de múltiplos hosts e aumenta a estabilidade
  - Host RT também aumenta overhead de CPU e de memória

# Mude as Definições de Distribuição

Modifique as configurações de distribuição para otimizar desempenho e determinismo

- **Configurações da BIOS**

- Desabilite Legacy USB Support
- Desabilite Hyperthreading

- **Configurações do MAX**

- Configure Ethernet Packet Detection para Polling

- **Configurações do Target no Projeto LabVIEW**

- Desabilite “CPU Load Monitoring”
- Desabilite “Debugging”

- **NI Web-Based Monitoring and Configuration**

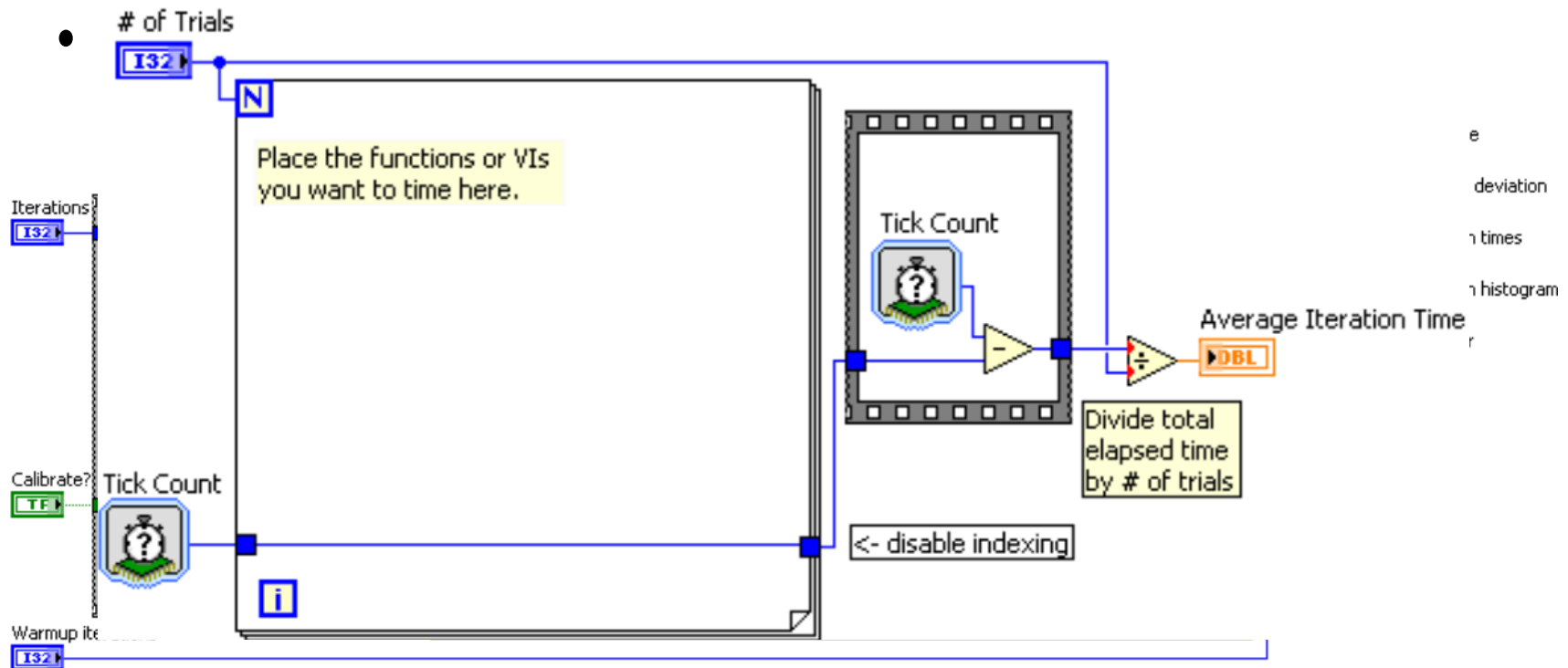
- Desabilite “Automatically Refreshing the Console Output”

# Considerações Adicionais sobre Distribuição

- Instale o conjunto mínimo de software
  - Aumenta a RAM disponível
  - Minimiza interrupções relacionadas a driver
- Compile e execute código como aplicação autônoma
  - Compilação de código otimizada
  - Remove o overhead da Interface de Usuário do painel frontal

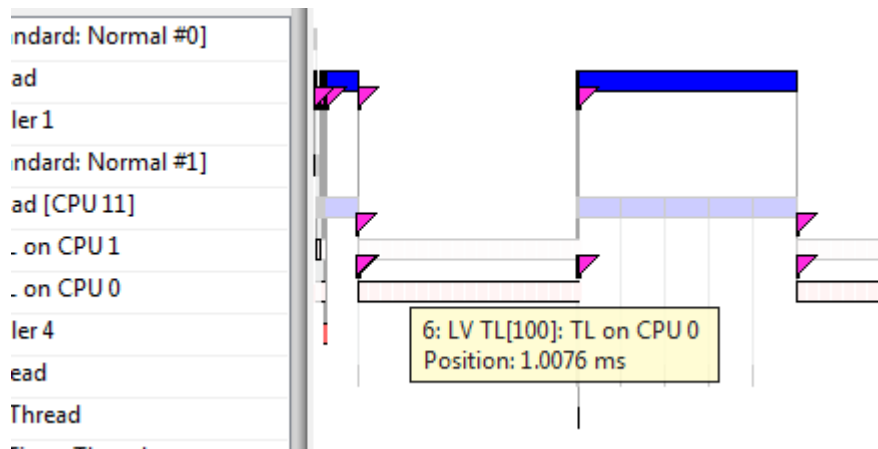
# Realize Benchmark Corretamente

- Faça Benchmark com as Definições de Distribuição



# Realize Benchmark Corretamente

- Capture e Analise usando o Toolkit Real-Time Execution Trace
- Monitore métricas usando o ***NI Distributed System Manager***
  - CPU
  - Memória
  - Alertas
  - Estados de VIs
  - Valores de Shared Variable



**Demo**

# **Monitorando Métricas em Sistemas Real Time**



# Minimize o Uso de CPU

- **Crie um Orçamento de Tempo**
  - Priorize threads por requisitos de tempo
- **Evite Paralelismo Excessivo**
  - Reuse padrões de projeto comuns
  - Faça Benchmark para determinar a arquitetura de código ótima
- **Limite o Uso de Shared Variable**
- **Descarregue Tarefas (quando possível)**
  - Use FPGA para algoritmos de aquisição e controle
  - Use PC Host para log, monitoramento e ser host de shared variable

# Resumo

- **Projeto**
  - Entendendo Modelos de Agendamento de Prioridades
  - Utilize Padrões de Projeto
  - Separe Tarefas
- **Desenvolvimento**
  - Evitando Jitter
  - Comunique com a Interface de Usuário
  - Usando Shared Variables Efetivamente
- **Distribuição**
  - Mude as Definições de Distribuição
  - Realize Benchmark Corretamente
  - Minimize o uso de CPU

# Dúvidas?