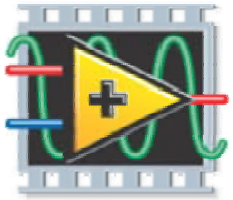
Embargoed until
August 6, 2007



LabVIEW 8.5

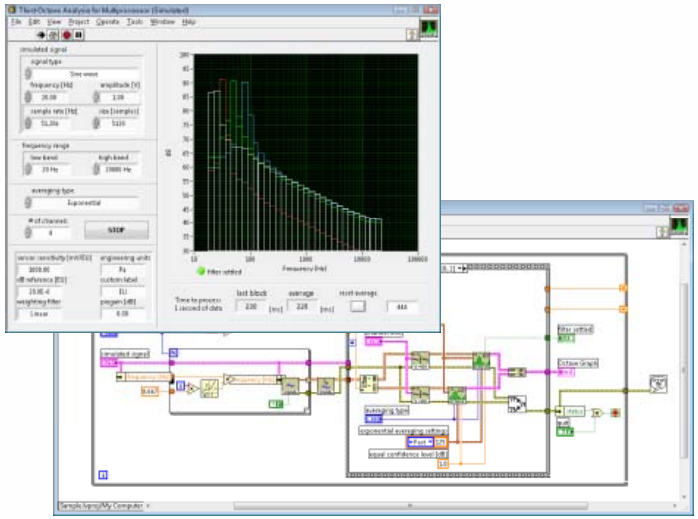
Delivering Multicore Performance to Scientists and Engineers

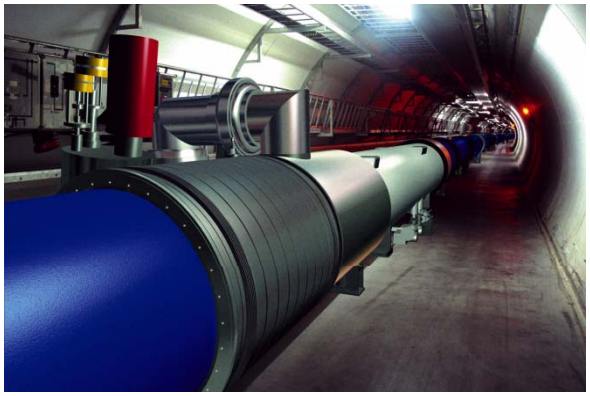
Graphical System Design

LEGO MINDSTORMS NXT
"the smartest, coolest toy of the year"

POWERED BY

NATIONAL INSTRUMENTS LabVIEW™

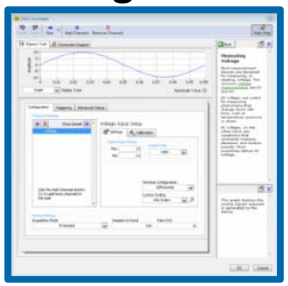




CERN Large Hadron Collider
"the most powerful instrument on earth"

High-Level Design Tools

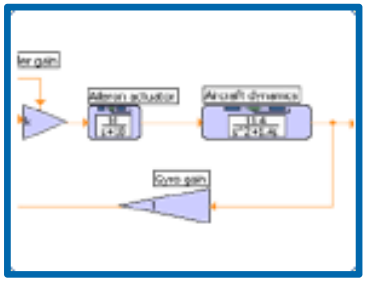
Configuration



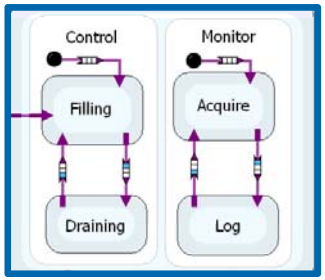
Textual Math

```
1 c = 0.285 + 0.013i;
2 [X Y] = meshgrid(x, y);
3 z = X + i*Y;
4 for k=1:30
5     z = z.^2 + c;
6 end
```

Simulation



Statechart



LabVIEW

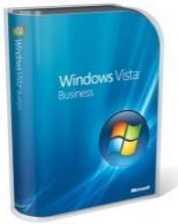
Graphical Programming

Desktop Platform

Linux®

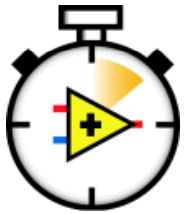
Macintosh

Windows

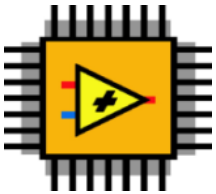


Embedded Platform

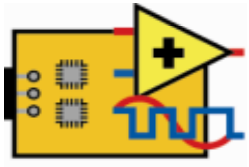
Real-Time



FPGA



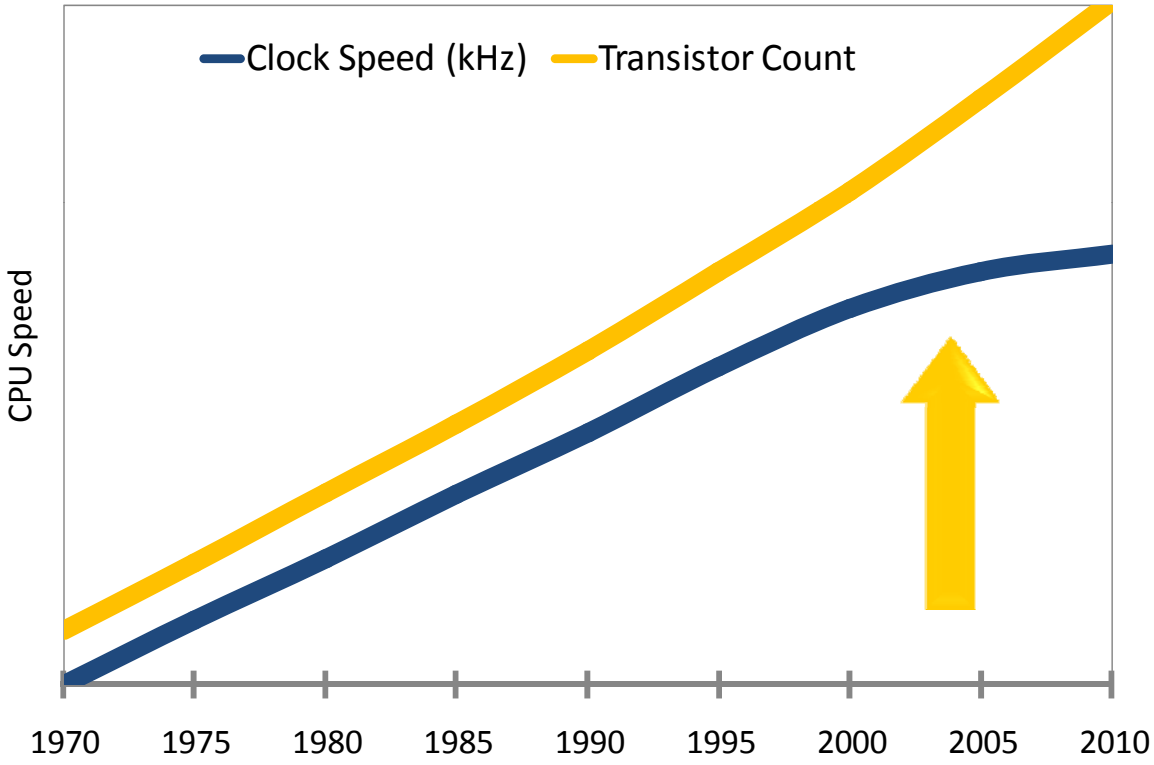
MPU



NATIONAL INSTRUMENTS

Parallel Architectures Drive Performance

Faster processors → Multicore processors



Intel QX6700 Quad Core Processor
- 4 processors (pair of Core 2 dies)
- 2.66 GHz clock speed

Multicore Systems Require Different Programming Strategies

"The good news is that **processors are going to continue to become more powerful**. The bad news is that, at least in the short term, the growth will come mostly in directions that **do not take most current applications along for their customary free ride.**"

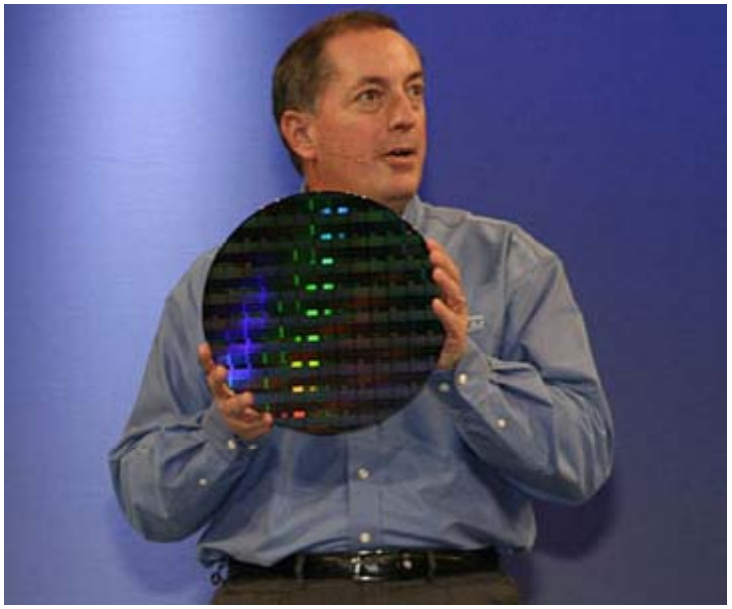
- Herb Sutter, Microsoft Software Architect

Source: "The Free Lunch Is Over"
Dr Dobb's Journal, March 2005

The Future of Parallel Architectures

"Intel pledges 80 cores in five years"

- Headline following Intel Developer Forum
September 2006



LabVIEW 8.5

Delivering Multicore Performance to Scientists and Engineers

- Automatic multithreading on desktop PCs
- Real-time control on multicore systems
- Statechart module for system-level design

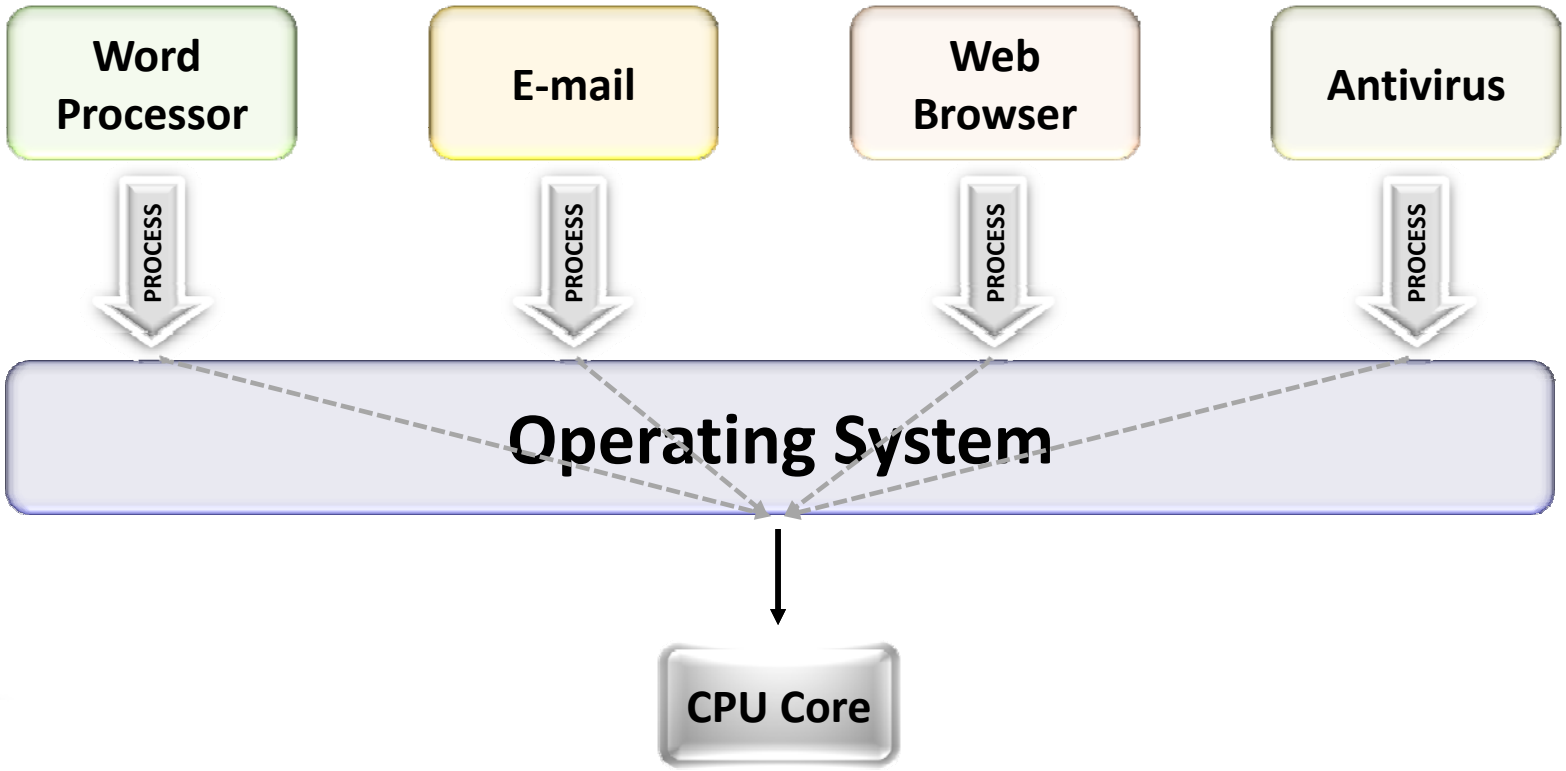
LabVIEW 8.5

Delivering Multicore Performance to Scientists and Engineers

- **Automatic multithreading on desktop PCs**
- Real-time control on multicore systems
- Statechart module for system-level design

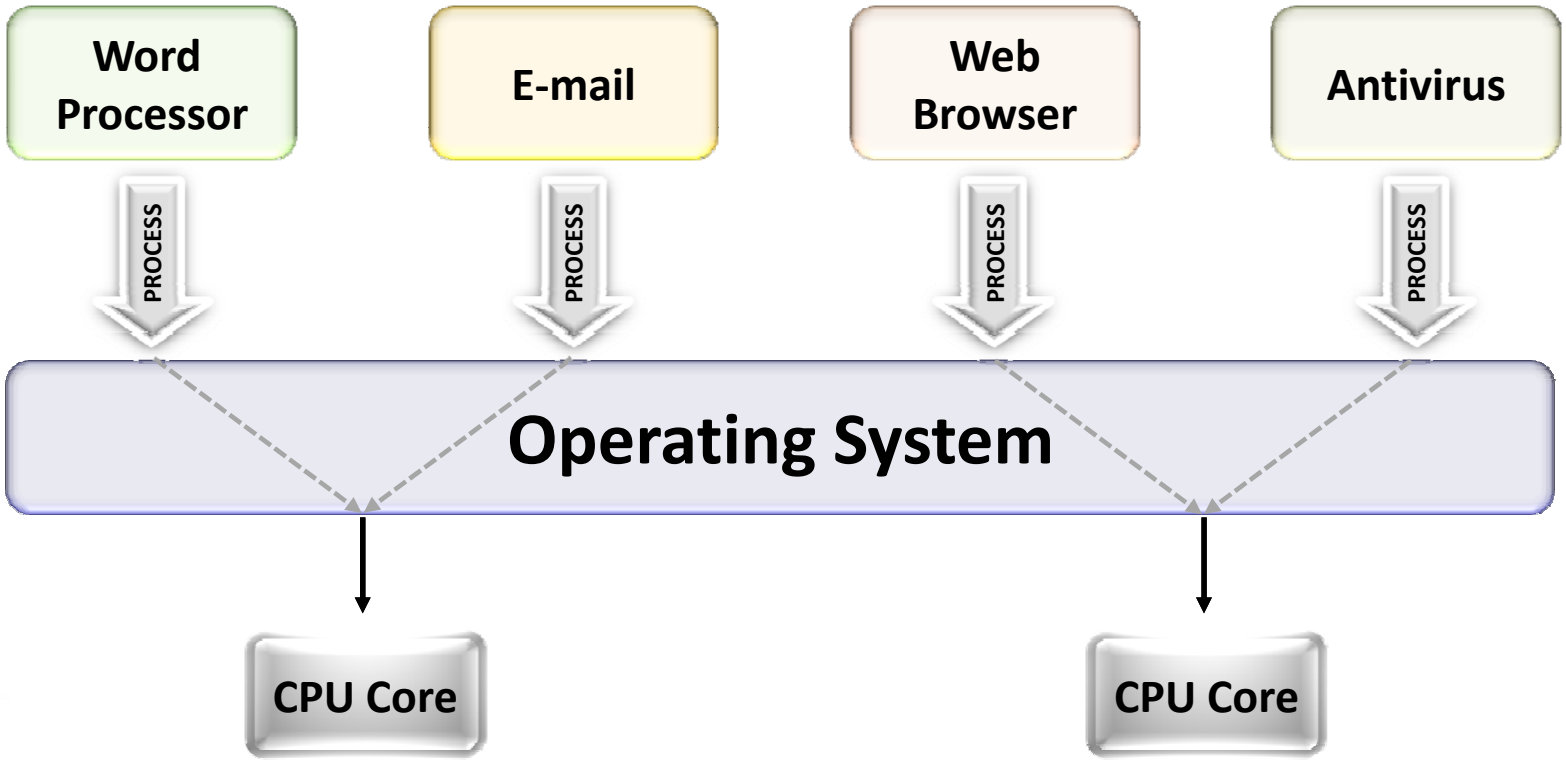
Multitasking on a Single Core

Operating systems schedule tasks on a single core.



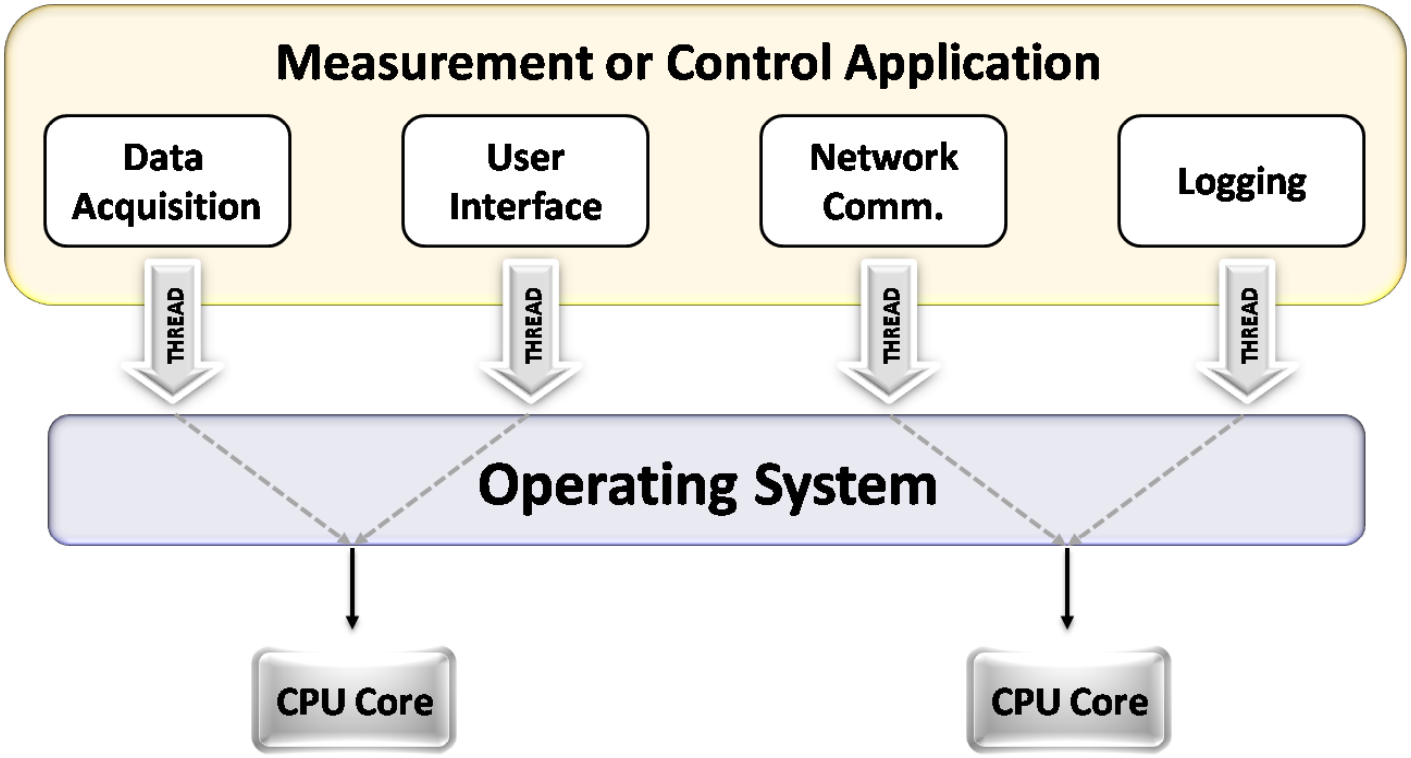
Multitasking with Multiple Cores

Operating systems schedule and automatically load-balance tasks across multiple processors.



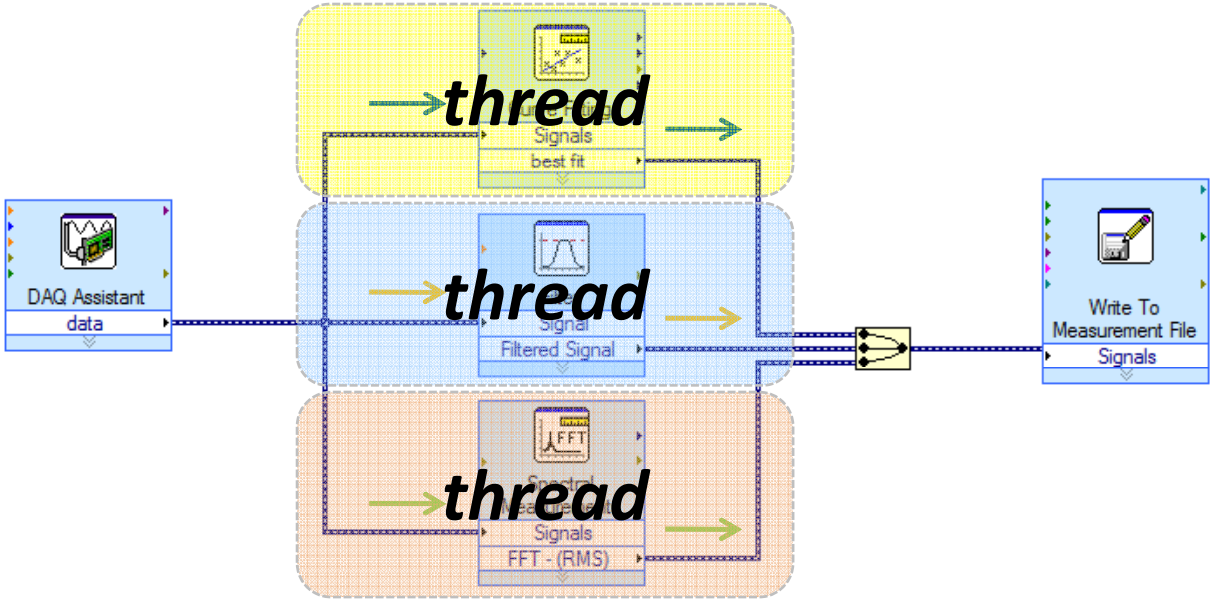
Impact on Engineers and Scientists

Engineers and scientists **must** create multithreaded applications to benefit from multicore processors.



Automatic Multithreading in LabVIEW

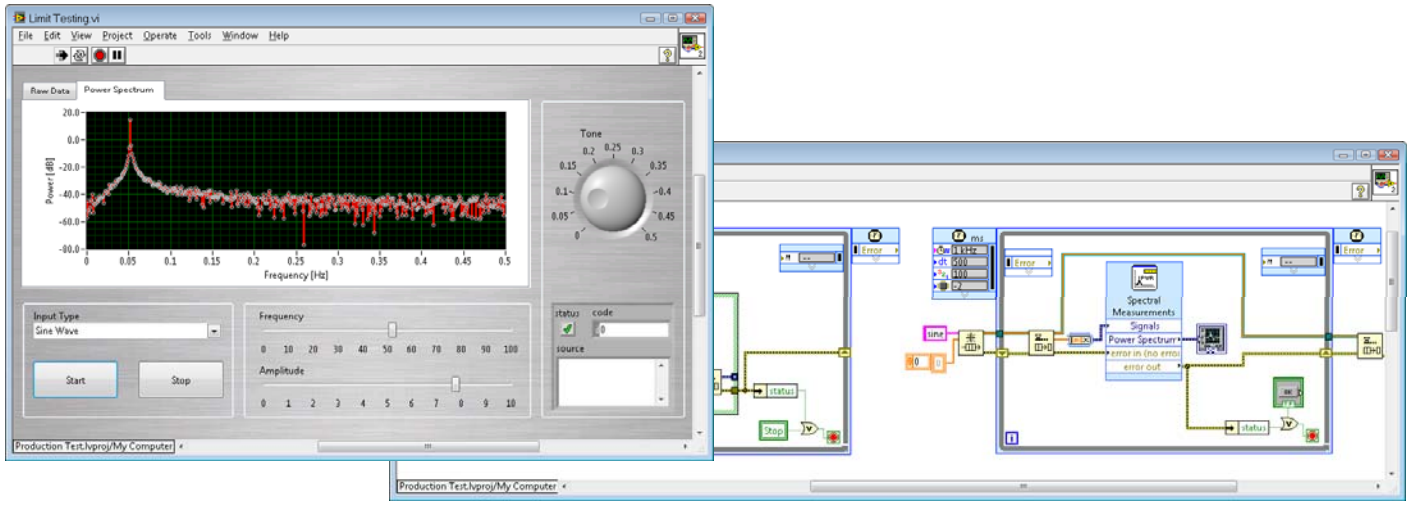
- LabVIEW automatically divides each application into multiple execution threads
- LabVIEW introduced multithreading in 1998



Demo:
Multithreading on a Multicore System

NEW Multithreading Features in LabVIEW 8.5

- Scale execution threads based on available cores
- Improved thread scheduling for LabVIEW timed loops
- Thread-safe NI device drivers for multicore systems

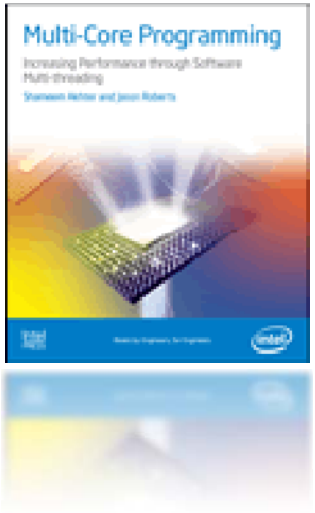


NATIONAL INSTRUMENTS

Programming Threads Is Difficult

Developers using conventional tools must learn new functions to:

- Create and destroy threads
- Communicate between threads
- Synchronize threads
- Debug across threads



Example: Simple Thread Synchronization

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

        // CriticalSection.cpp
#include <windows.h>
        #include <stdio.h>
        #include <process.h>
CRITICAL_SECTION g_criticalSection;
        bool g_bThreadOneFinished = false;
        bool g_bThreadTwoFinished = false;
        int g_iResult = 0;
void ThreadOne( void *)
        {
          for ( int i = 0; i < 10000; i++)
          {
            // Request ownership of the critical section
            EnterCriticalSection(&g_criticalSection);
   // Access the shared resource
            g_iResult += 1;
   // Release ownership of the critical section
            LeaveCriticalSection(&g_criticalSection);
          }
  // Finished
          g_bThreadOneFinished = true;
          _endthread();
        }
        }
void ThreadTwo( void *)
        {
          for ( int i = 0; i < 10000; i++)
          {
            // Request ownership of the critical section
            EnterCriticalSection(&g_criticalSection);
   // Access the shared resource
            g_iResult += 1;
   // Release ownership of the critical section
            LeaveCriticalSection(&g_criticalSection);
          }
  // Finished
          g_bThreadTwoFinished = true;
          _endthread();
        }
int main()
        {
          // Initialize the critical section
          InitializeCriticalSection(&g_criticalSection);
  // Start the threads
          _beginthread(ThreadOne, 0, NULL);
          _beginthread(ThreadTwo, 0, NULL);
  // Wait for the threads to finish
          while (( false == g_bThreadOneFinished)
          || ( false == g_bThreadTwoFinished))
          {
            Sleep(1);
          }
  // Release resources used by the critical section
          DeleteCriticalSection(&g_criticalSection);
  // Print the result
          printf("Result: %i\n", g_iResult);
```

Simple Thread Synchronization

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

// CriticalSection.cpp
#include <windows.h>
#include <stdio.h>
#include <process.h>
CRITICAL_SECTION g_criticalSection;
bool g_bThreadOneFinished = false;
bool g_bThreadTwoFinished = false;
int
```

Thread 1

```
void ThreadOne( void *)
{
 for ( int i = 0; i < 10000; i++)
 {
  // Request ownership of the critical section
  EnterCriticalSection(&g_criticalSection);
// Access the shared resource
  g_iResult += 1;
// Release ownership of the critical section
  LeaveCriticalSection(&g_criticalSection);
 }
// Finished
 g_bThreadOneFinished = true;
 _endthread();
}
}
```

Thread 2

```
void ThreadTwo( void *)
{
 for ( int i = 0; i < 10000; i++)
 {
  // Request ownership of the critical section
  EnterCriticalSection(&g_criticalSection);
// Access the shared resource
  g_iResult += 1;
// Release ownership of the critical section
  LeaveCriticalSection(&g_criticalSection);
 }
// Finished
 g_bThreadTwoFinished = true;
 _endthread();
}
int main()
{
 // Initialize the critical section
 InitializeCriticalSection(&g_criticalSection);
// Start the threads
 _beginthread(ThreadOne, 0, NULL);
 _beginthread(ThreadTwo, 0, NULL);
// Wait for the threads to finish
 while (( false == g_bThreadOneFinished)
 || ( false == g_bThreadTwoFinished))
 {
  Sleep(1);
 }
// Release resources used by the critical section
 DeleteCriticalSection(&g_criticalSection);
// Print the result
 printf("Result: %i\n", g_iResult);
```

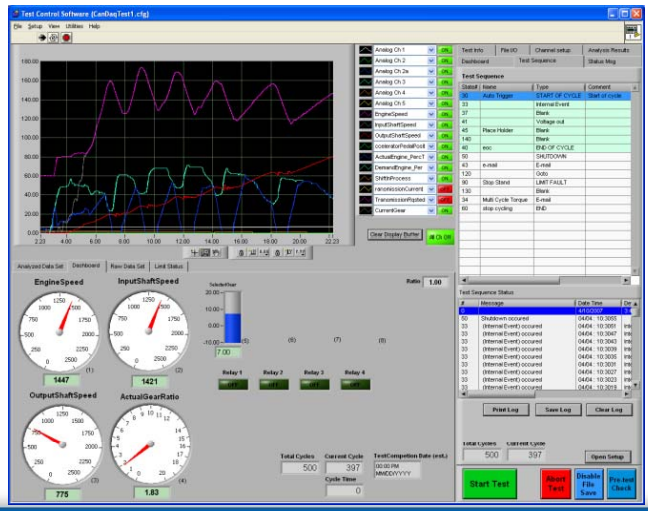
Task Parallelism Applied: Automotive Test

Eaton created a portable in-vehicle test system for truck transmissions using LabVIEW.

- Acquired and analyzed 16 channels on single core
- Now acquire and analyze 80+ channels on multicore

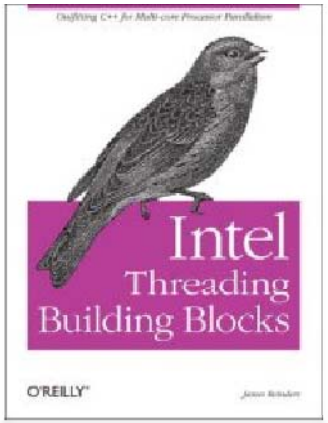
*"There was **no need to rewrite our application** for the new multicore processing platforms."*

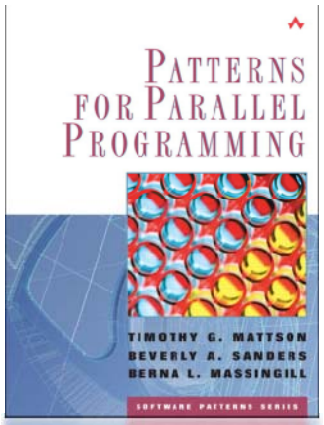
Scott Sirine
Lead Design Engineer
Eaton Truck Division



Techniques to Improve Performance on Multicore Systems

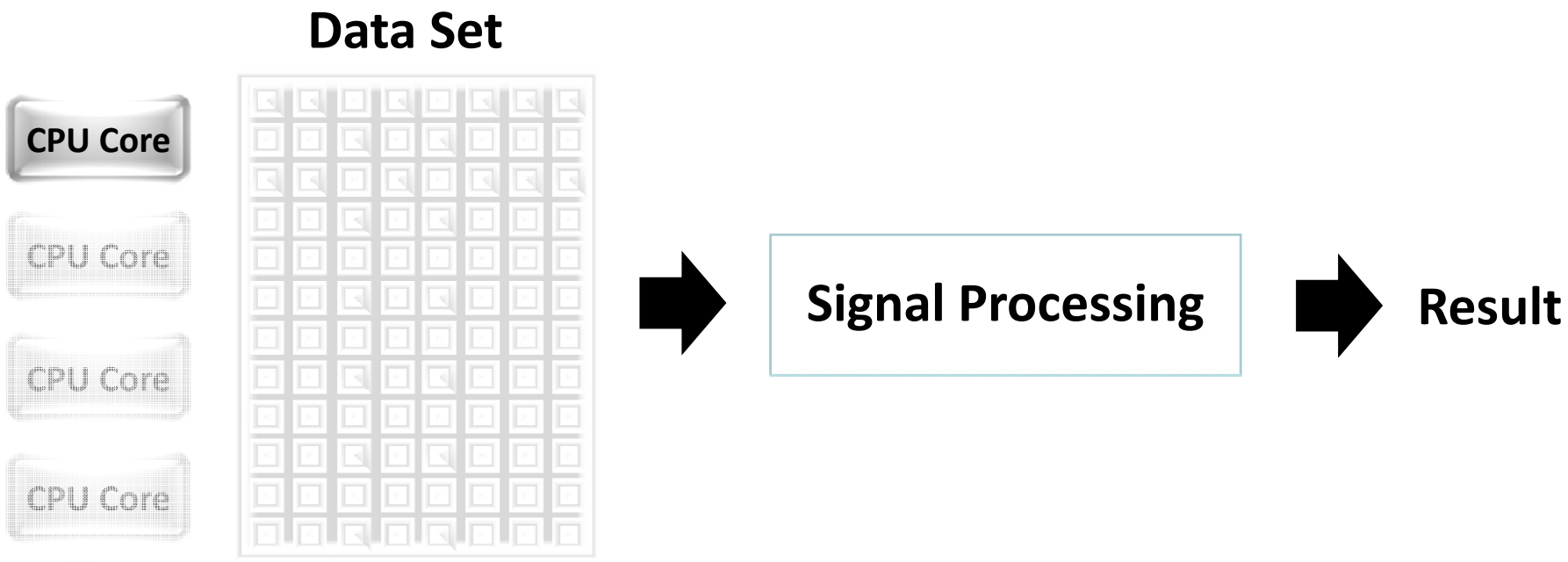
- Task Parallelism
- Data Parallelism
- Pipelining





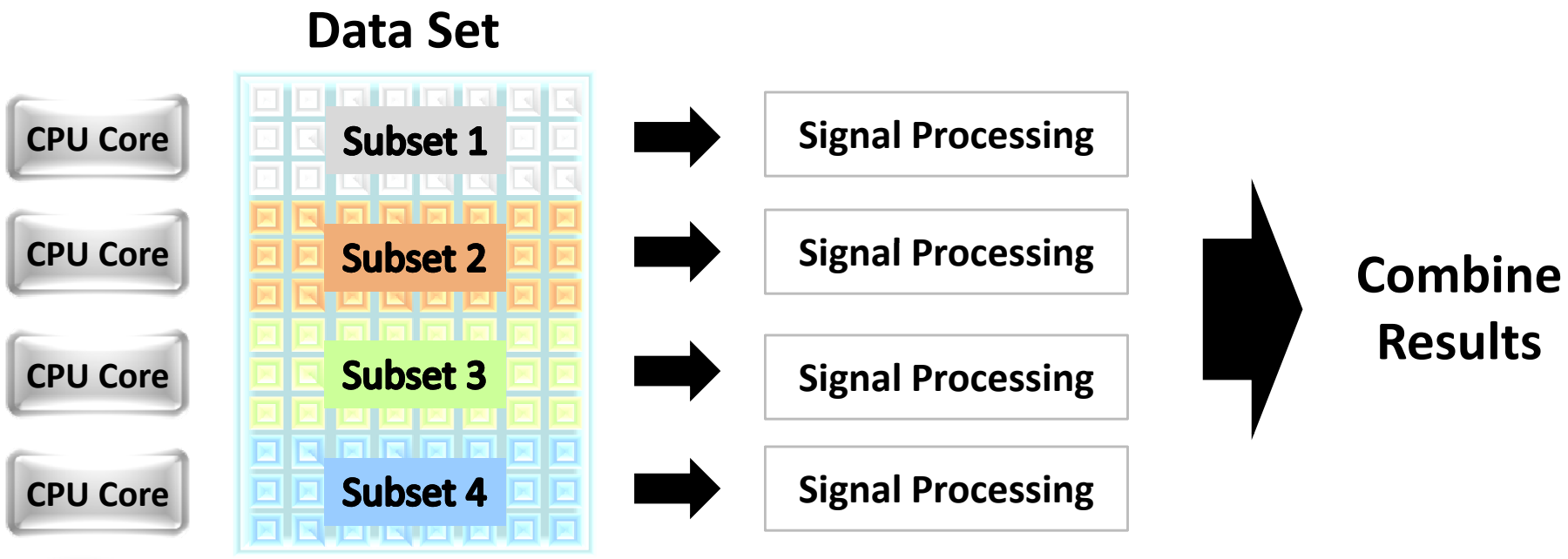
Data Parallelism

You can speed up processor-intensive operations on large data sets on multicore systems.

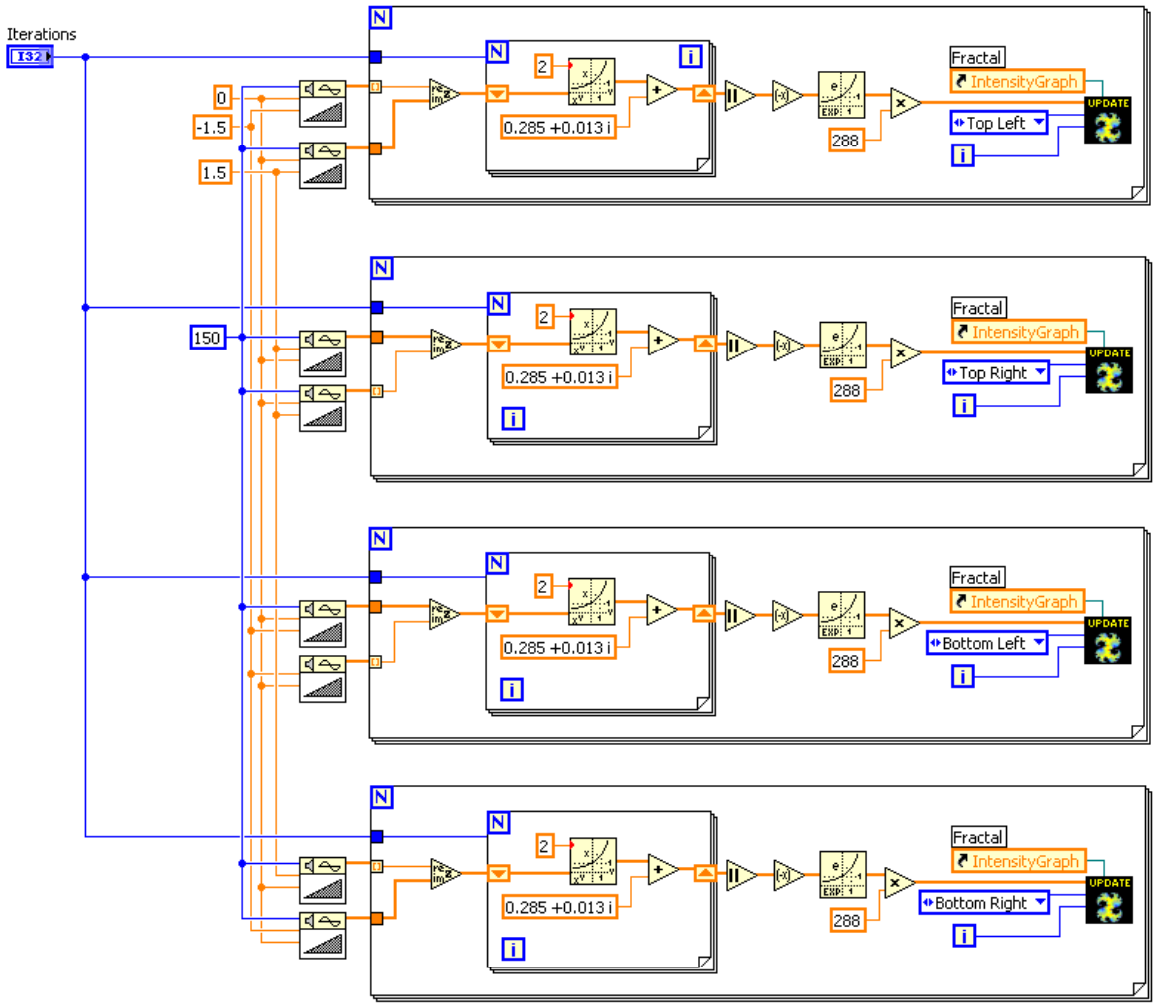


Data Parallelism

You can speed up processor-intensive operations on large data sets on multicore systems.



Data Parallelism in LabVIEW



Data Parallelism Applied: Process Control

- Max Planck Institute (Munich, Germany)
- Plasma control in nuclear fusion tokamak with LabVIEW on 8 core system

*"…with LabVIEW, we obtained a **20X processing speed-up** on an octal-core processor machine over a single-core processor…"*

Louis Giannone

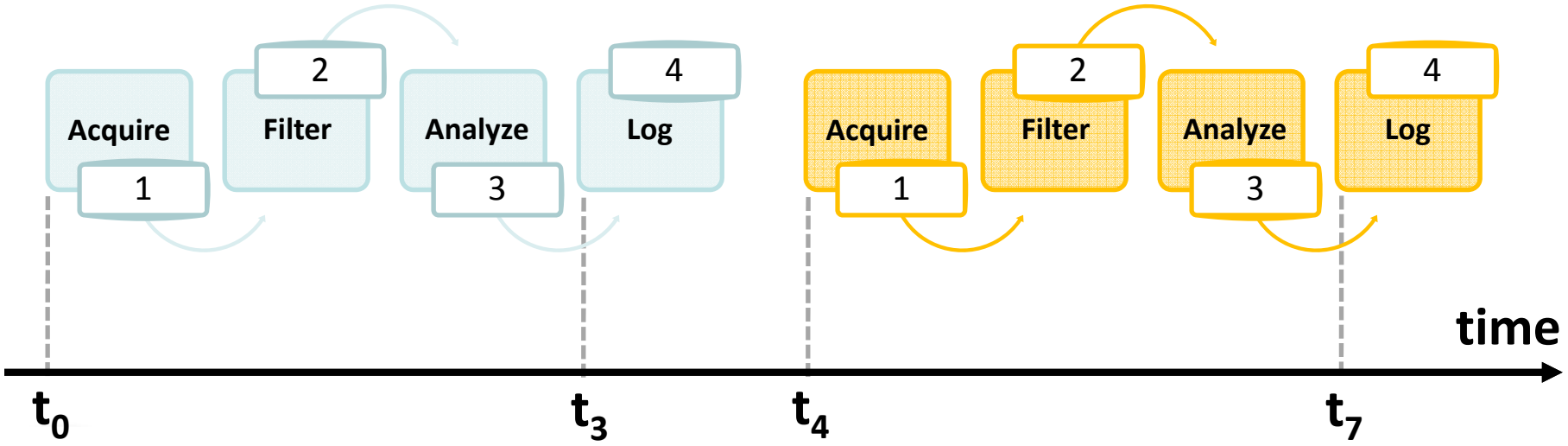
Lead Project Researcher

Max Planck Institute

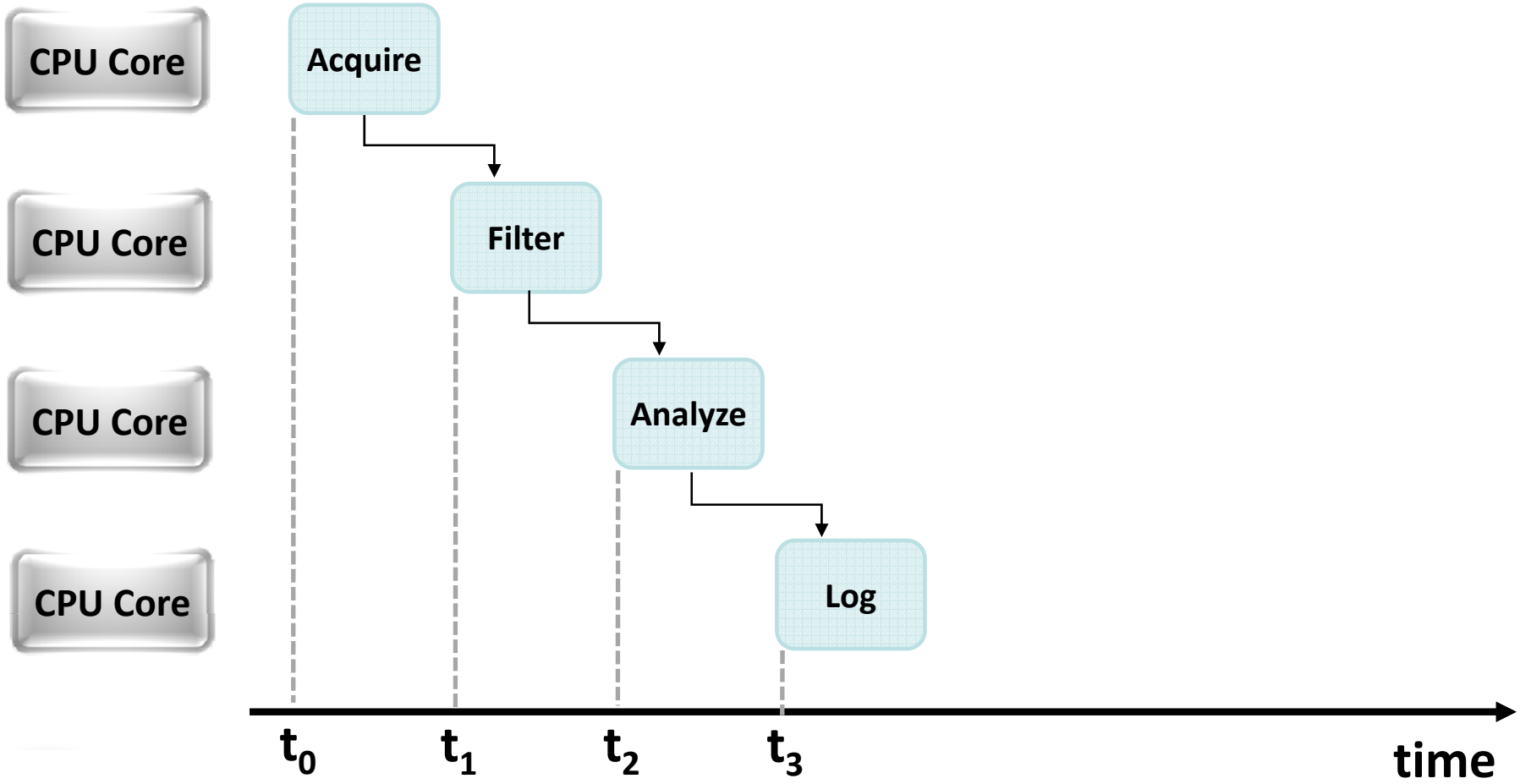
IPP Max-Planck-Institut für Plasmaphysik

Pipelining Strategy

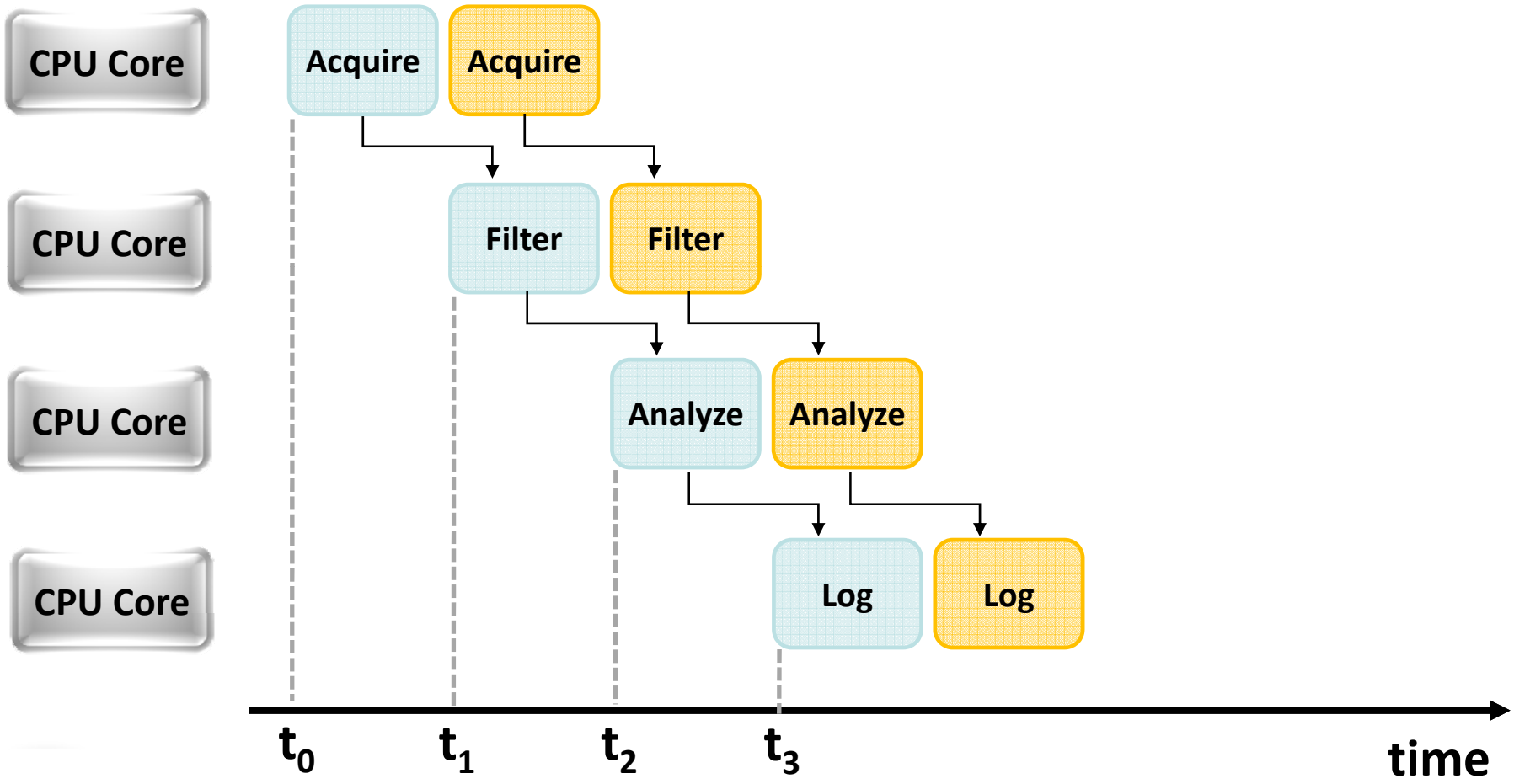
- Many applications involve sequential, multistep algorithms
- Applying pipelining can increase performance



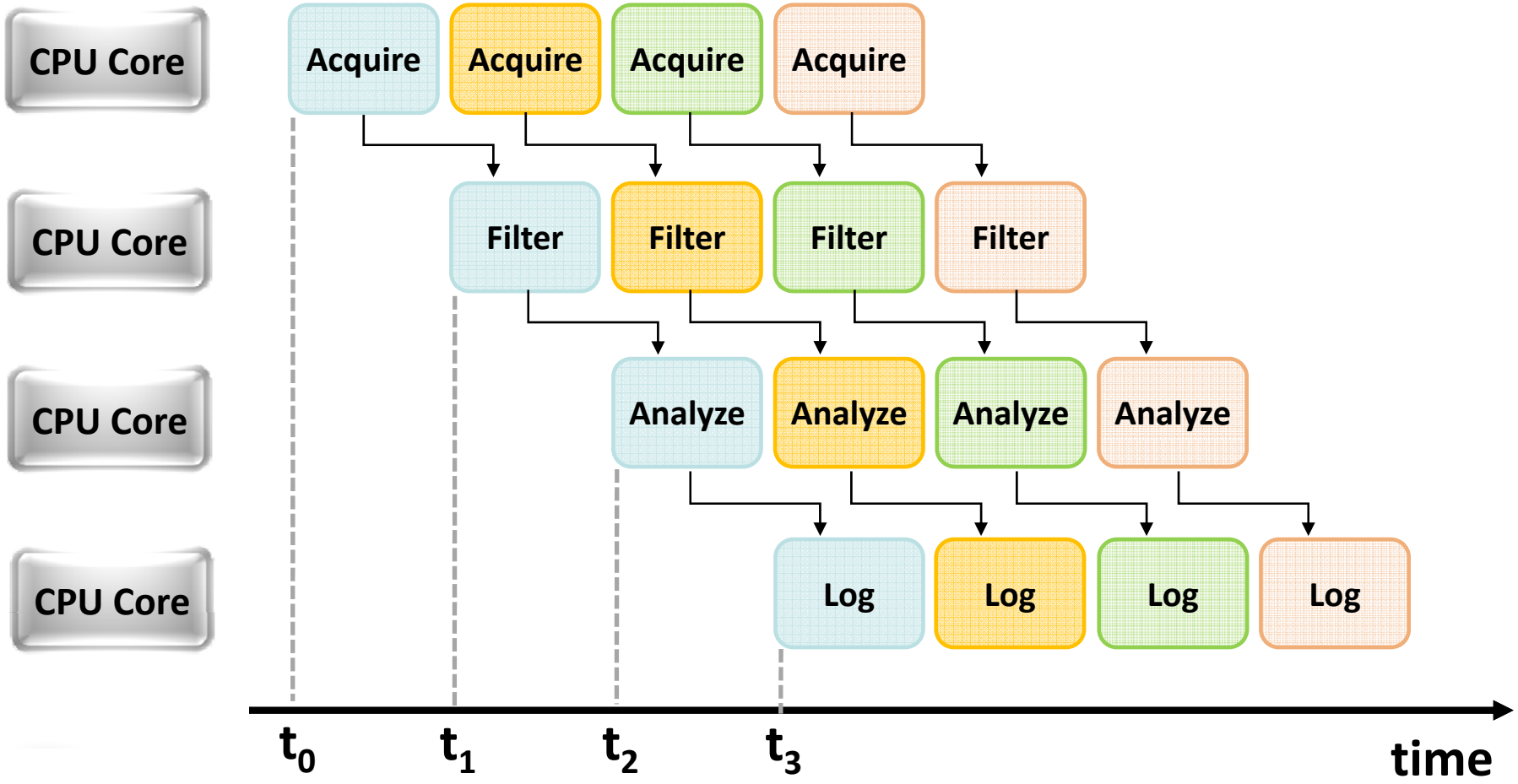
Pipelining Strategy



Pipelining Strategy

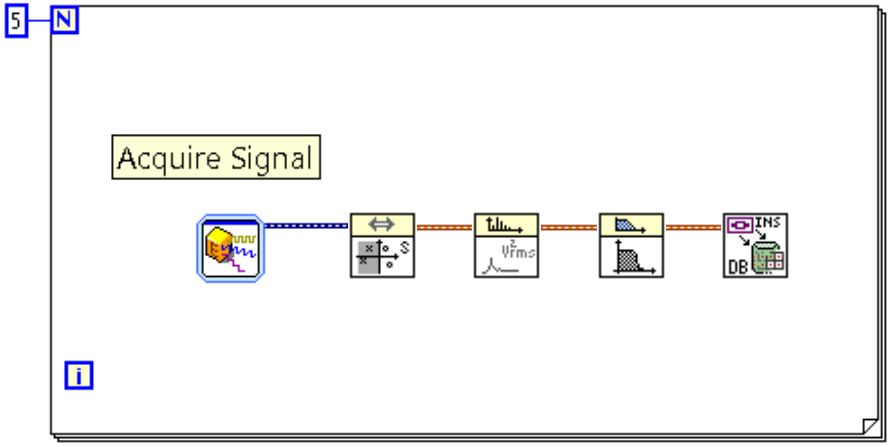


Pipelining Strategy

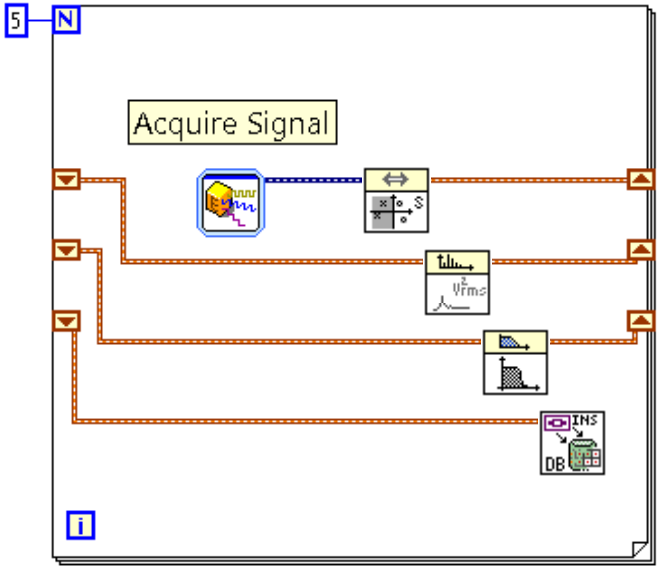


Data Pipelining in LabVIEW

Sequential



Pipelined



Pipelining Applied: Communications Test

- AmFax Ltd. (United Kingdom)
- Create wireless test systems for next-generation phones using pipelined LabVIEW architecture

"With LabVIEW and the dualcore embedded controller we are ***achieving up to 5x speed savings****…."*

Mark Jewell

BDM – Wireless

AmFax Ltd.

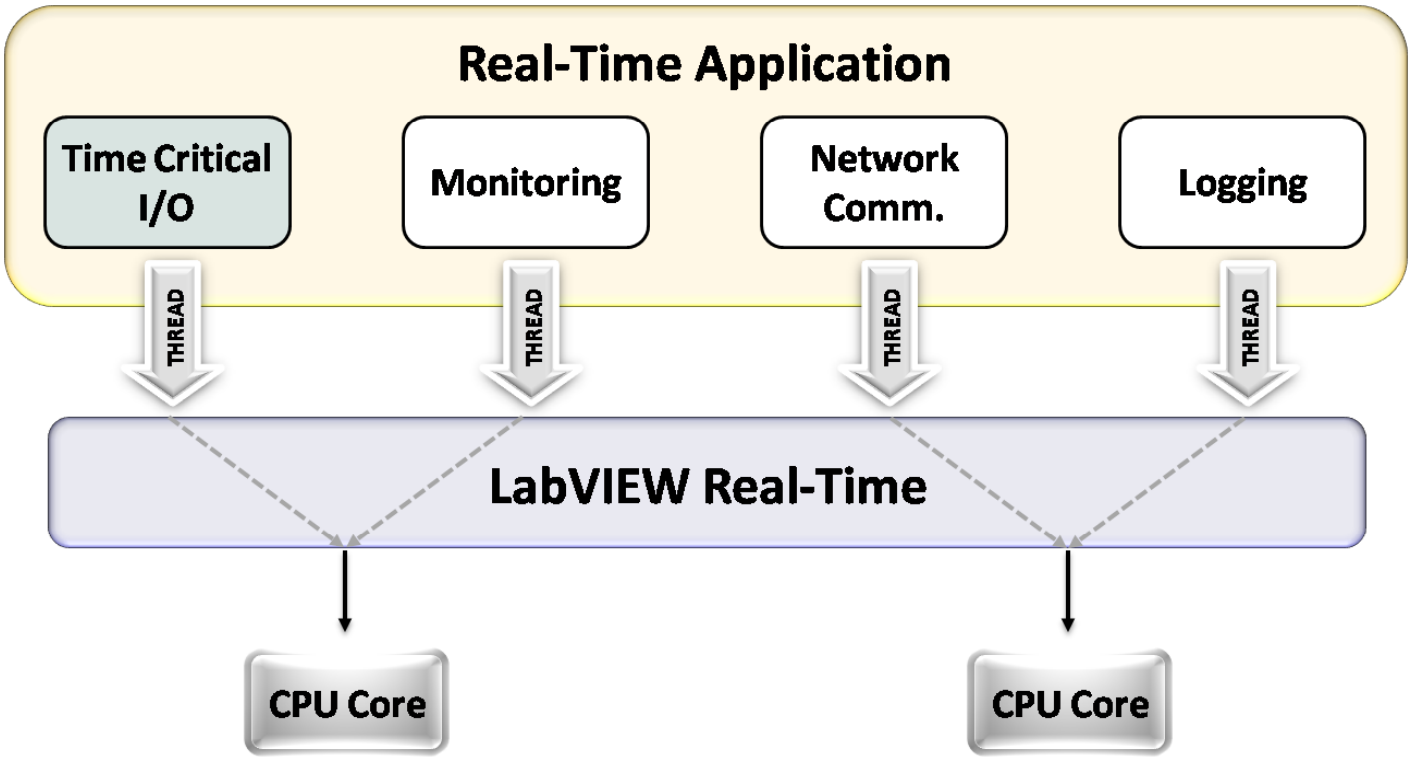
LabVIEW 8.5

Delivering Multicore Performance to Scientists and Engineers

- Automatic multithreading on desktop PCs
- **Real-time control on multicore systems**
- Statechart module for system-level design

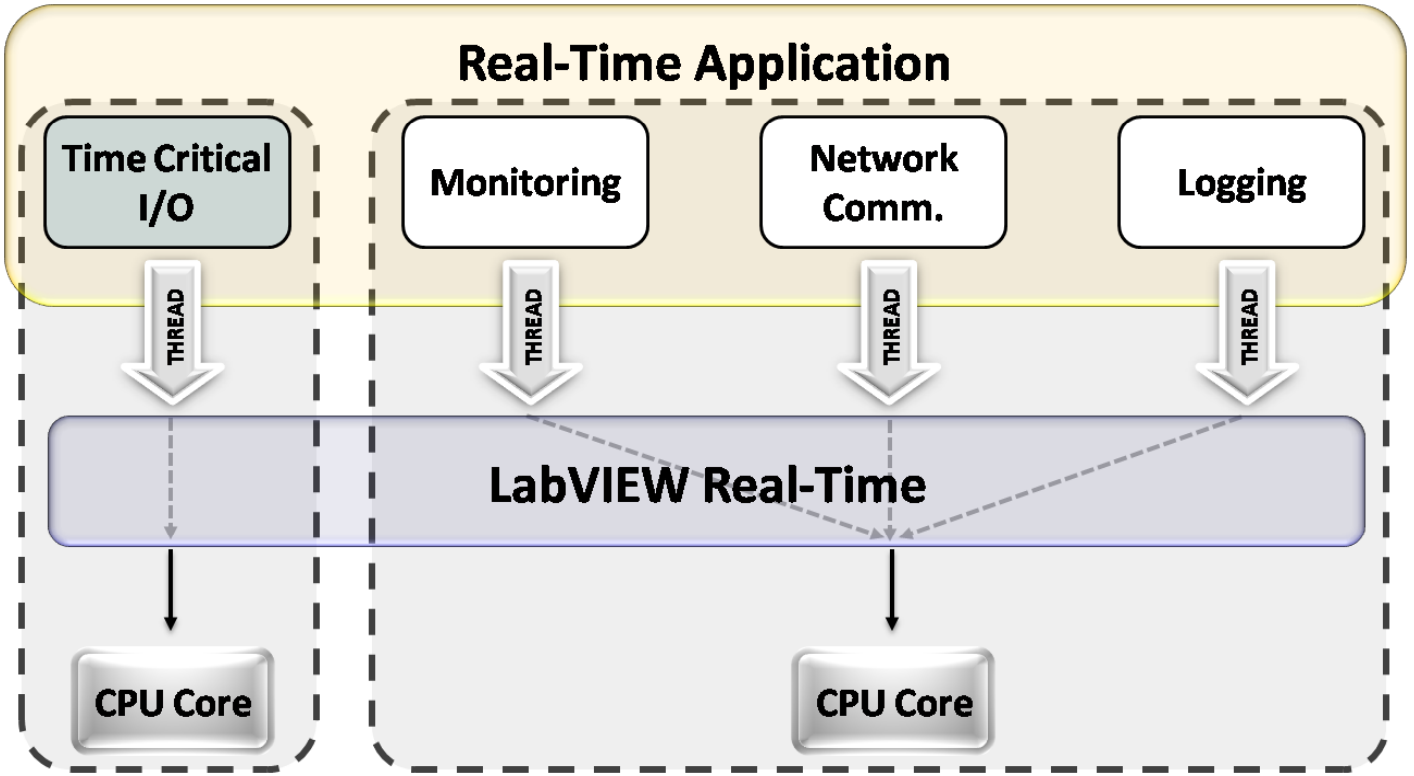
Deterministic Real-Time Systems

LabVIEW 8.5 adds Symmetric Multiprocessing (SMP) for real-time systems.



Assigning Tasks to Specific Cores

In LabVIEW 8.5, users can assign code to specific processor cores using the LabVIEW Timed Loop.



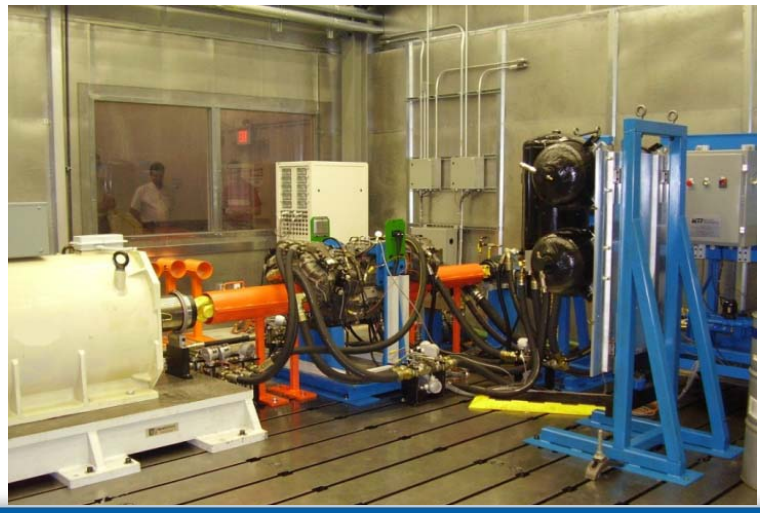
Demo:
Real-Time Multicore System

Real-Time SMP Applied: Closed-Loop Test

Wineman Technology creates closed-loop test solutions

- Hardware-in-the-loop simulation
- Dynamometer test

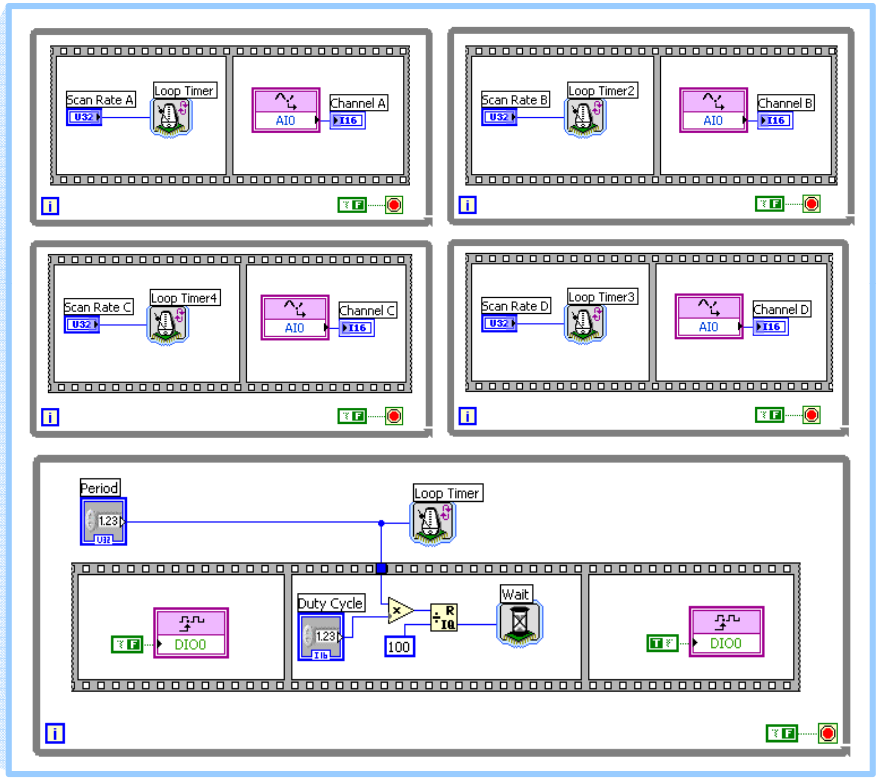
*"With LabVIEW Real-Time 8.5, we **[increased] our maximum loop rate by 40%** by utilizing both processor cores…"*



WINEMAN TECHNOLOGY INCORPORATED

FPGAs Are Parallel Architectures

- High performance
- Truly parallel
- Flexible
- Reliable
- Software configurable



LabVIEW 8.5

Delivering Multicore Performance to Scientists and Engineers

- Automatic multithreading on desktop PCs
- Real-time control on multicore systems
- **Statechart module for system-level design**

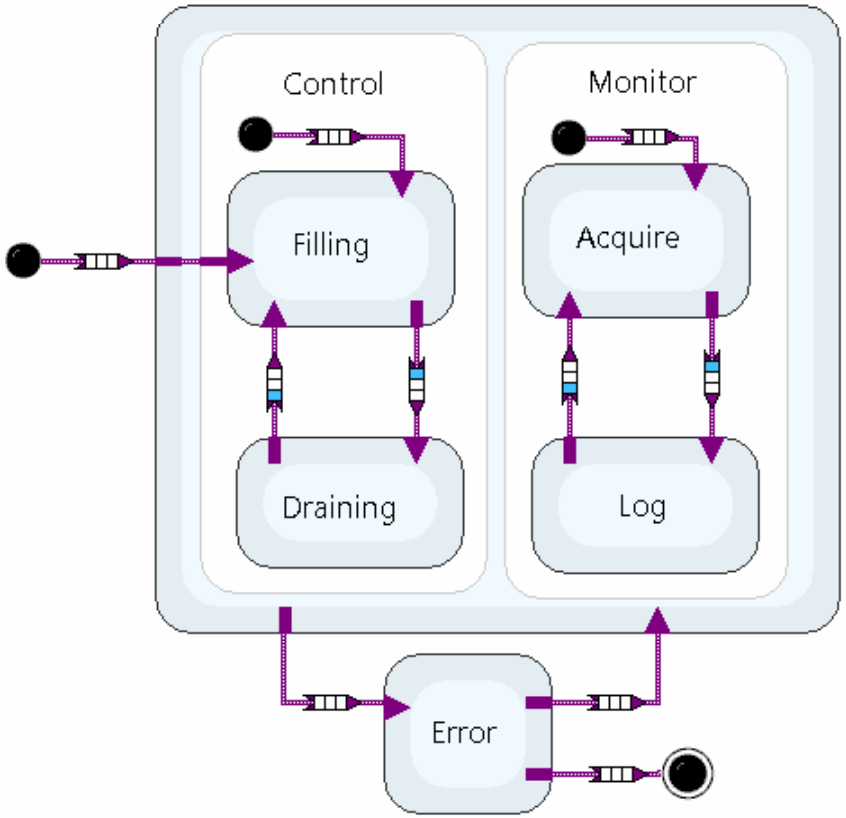
LabVIEW Statechart Module NEW

Design statecharts in LabVIEW

- Complex state machines
- State-based control
- User interfaces
- Communication protocols

Deploy statecharts to

- Desktop PCs
- Real-time systems
- FPGAs
- Microprocessors
- Industrial touch panels



Demo: Statechart Programming

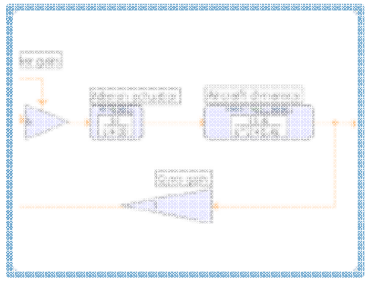
High-Level Design Tools

Configuration

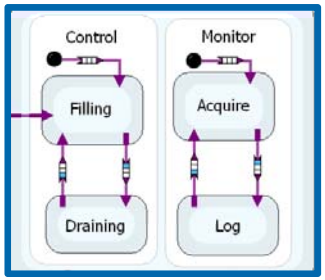
Textual Math

```
c = 0.285 + 0.013i;
[X Y] = meshgrid(x, y);
z = X + i*Y;
for k=1:30
    z = z.^2 + c;
end
```

Simulation



Statechart

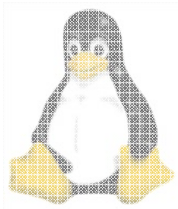


LabVIEW

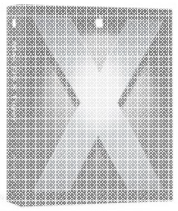
Graphical Programming

Desktop Platform

Linux®



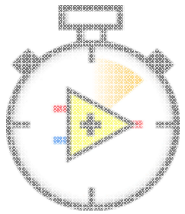
Macintosh



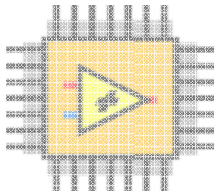
Windows

Embedded Platform

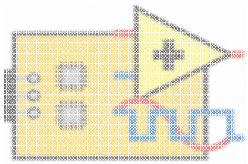
Real-Time



FPGA



MPU



NATIONAL INSTRUMENTS

More Features in LabVIEW 8.5

Embargoed until August 6, 2007

Automated Test

- Vision Express VI
- Vision Assistant
- **Improved vision edge detection algorithms**
- Merge for graphical LabVIEW code
- Integration with additional source code control software vendors
- **GPS reference architecture for RF**
- BLAS standard math library
- 70 additional LabVIEW MathScript functions
- Sound and Vibration Assistant
- LabVIEW PDA Module support for Inline C Node

Industrial Control

- Industrial pipe indicators in LabVIEW DSC
- **NI OPC Servers for PLC communication**
- **OPC debugging tools**
- Statistical process control in LabVIEW DSC
- Shared variable binding to front panels on touch panels
- Code reuse with Inline C Node on touch panels
- Ethernet deployment of touch panel applications

Embedded Design

- FPGA Project Wizard
- Improved simulation module performance
- ColdFire MPU example target support
- QNX OEM bundle for evaluation
- **Multichannel PID on FPGAs**
- New filters for FPGA-based embedded systems
- DMA support in FPGA I/O Wizard
- Memory management with In Place Element Structure on Real-Time systems