NIDays09

WORLDWIDE GRAPHICAL SYSTEM DESIGN

CONFERENCE

NATIONAL INSTRUMENTS

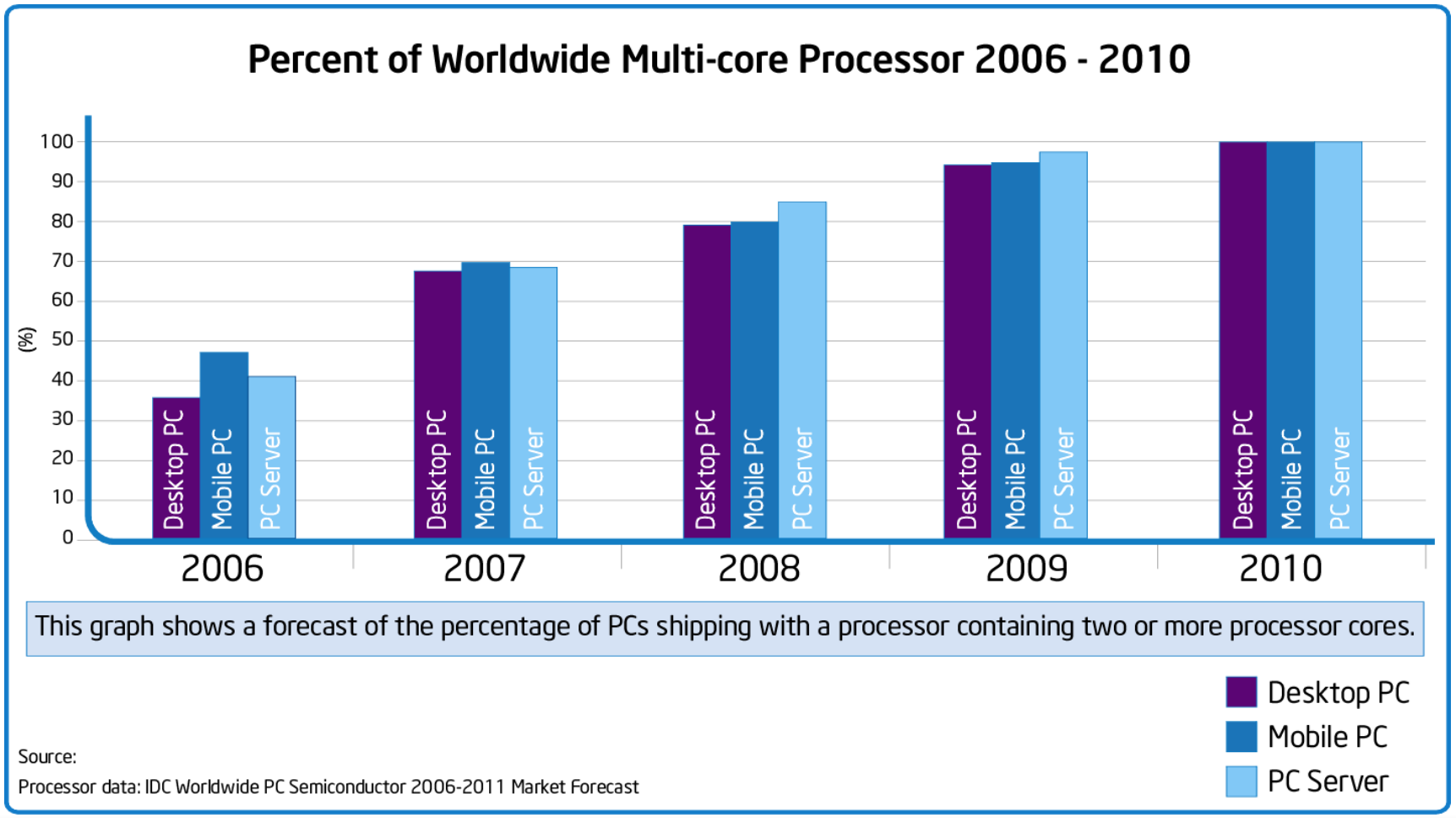
LabVIEW Programming Techniques for Multicore Systems

NIDays 2009

Agenda

- Multithreading Overview
- Parallel Programming Techniques
- Multicore Programming Challenges
- Conclusions

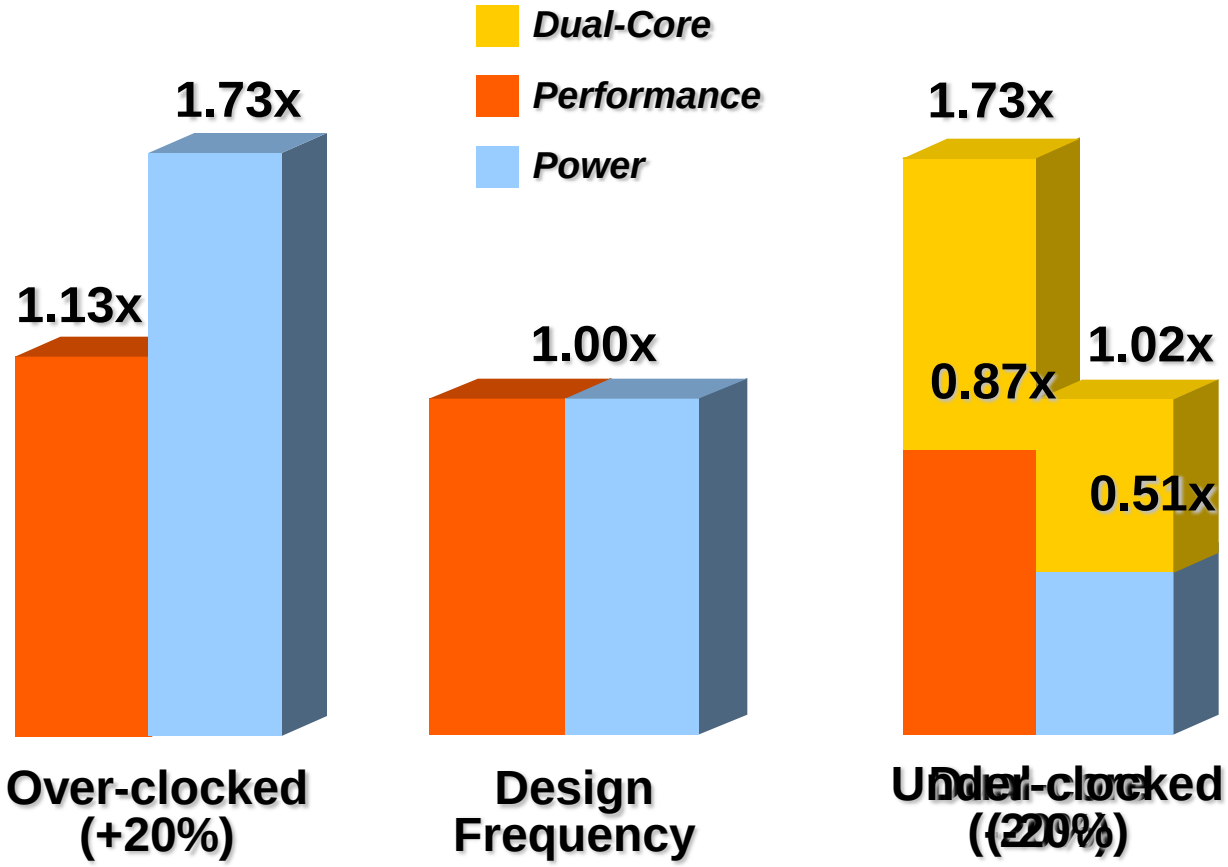
The Computing World is Going Multi-core – Are You Ready?



CPU Performance Now Comes From Multicore Designs

- Why multicore?
  - Faster clocks = more power and heat
  - Answer: slower clocks, but multiple cores
- GHz wars now replaced by N-core wars
- Many-core CPUs are closer than they appear
  - Polaris: experimental 80 core processor
  - Larrabee: 16-32 cores, 4x hyperthreaded

Moving To Multicore

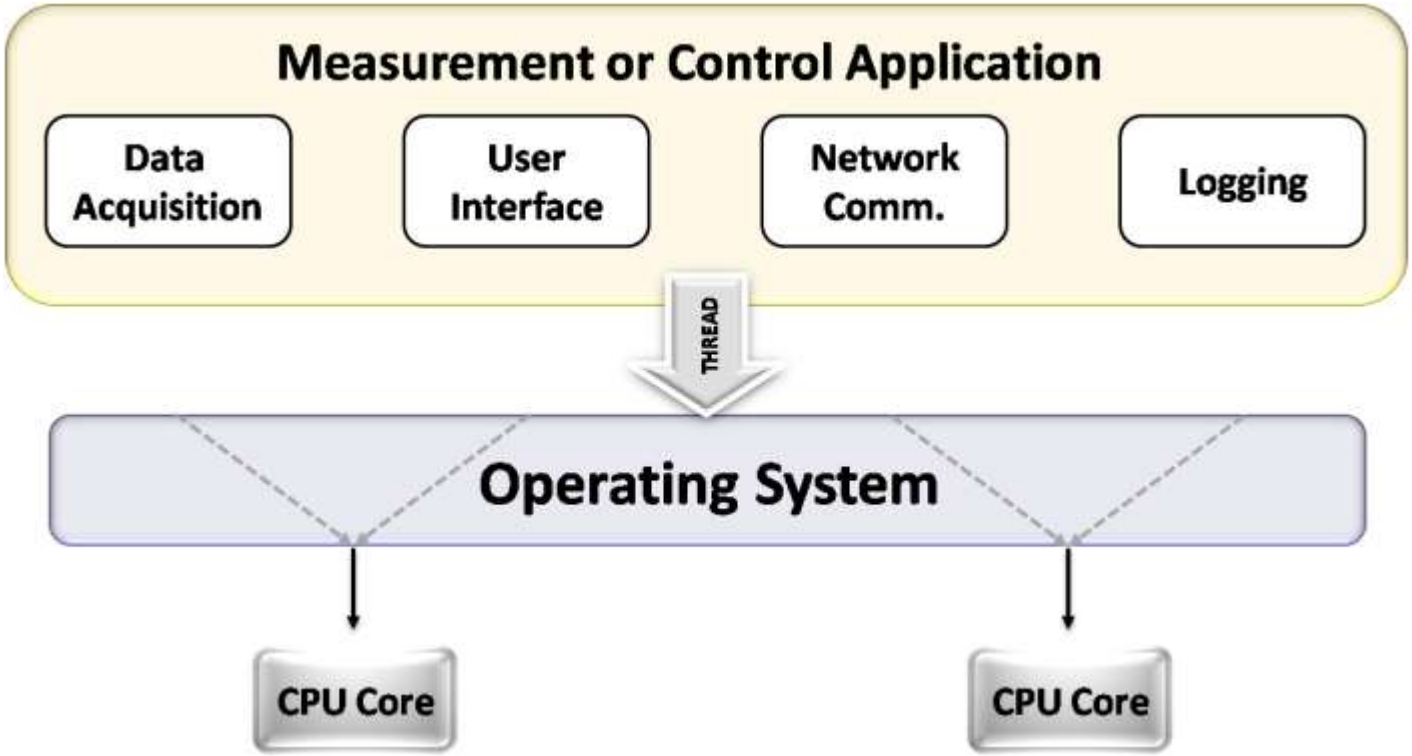


Relative single-core frequency and Vcc

MULTITHREADING OVERVIEW

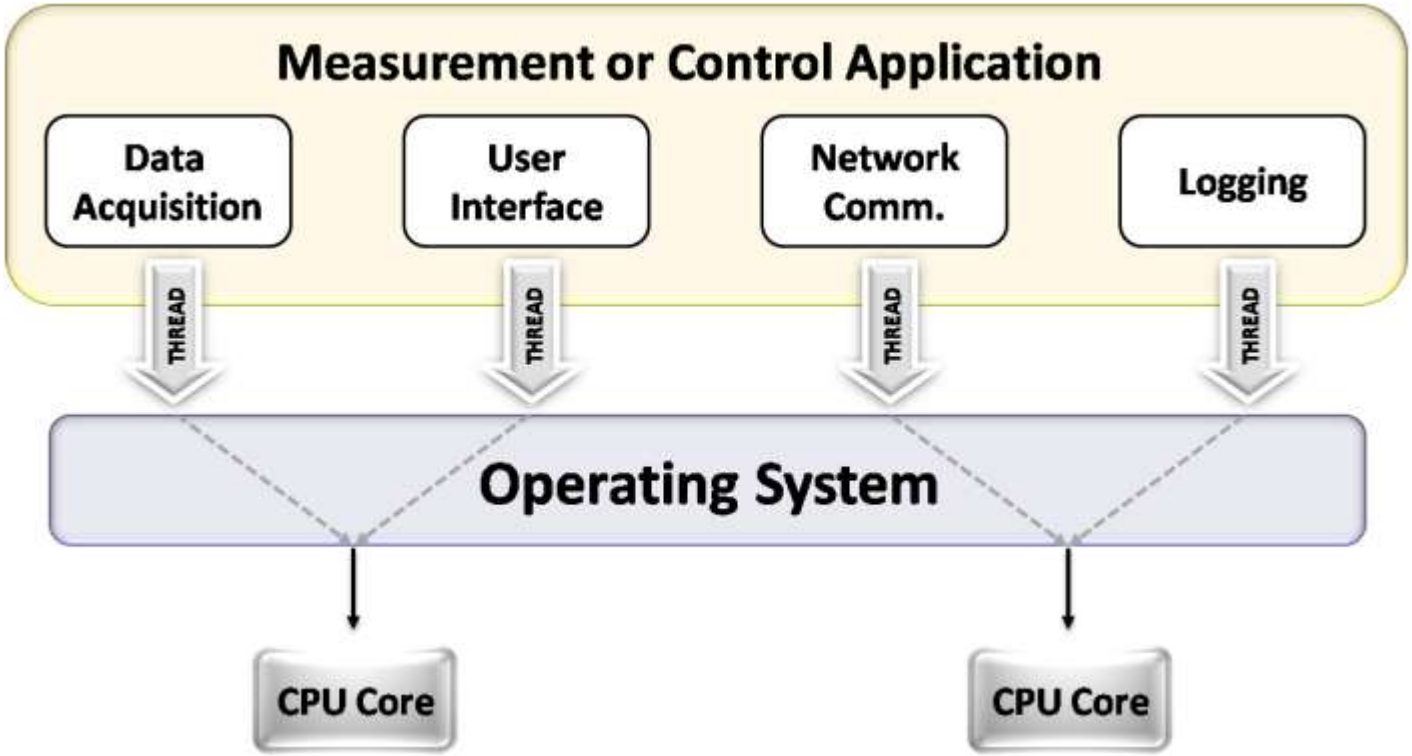
Impact on Engineers and Scientists

Engineering and scientific applications are typically on dedicated systems (i.e. little multitasking).



Creating Multithreaded Applications

Engineers and scientists **must** use threads to benefit from multicore processors.

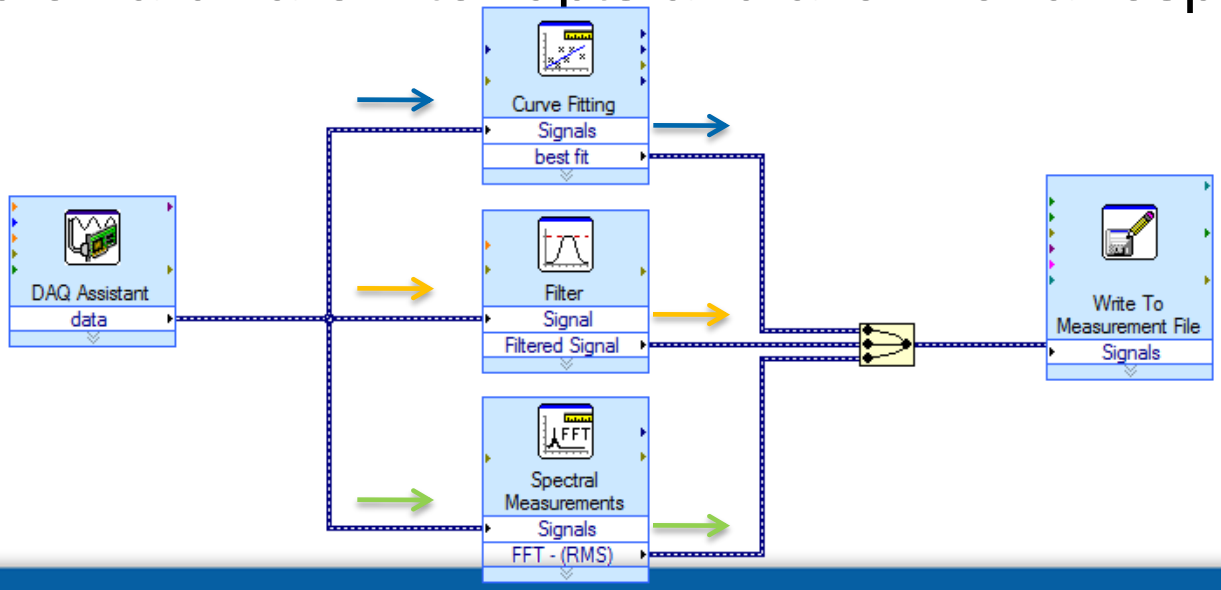


How LabVIEW Implements Multithreading

- Implicit Parallelism / Threading
  - Automatic Multithreading using LabVIEW Execution System
  - Parallel code paths on block diagram can execute in unique threads
- Explicit Parallelism / Threading
  - Timed Structures spawn a single thread

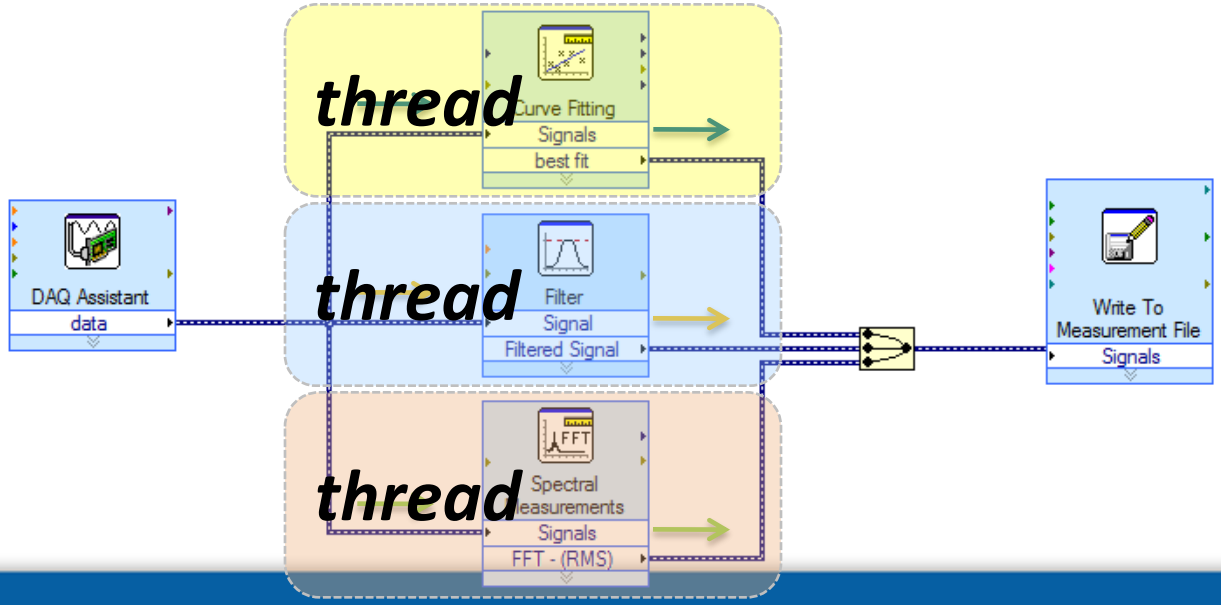
Automatic Multithreading in LabVIEW

- LabVIEW automatically divides each application into multiple execution threads (originally introduced in 1998 with LabVIEW 5.0)
- Original goal of multithreading was to more elegantly handle hardware interrupts and allow for a responsive UI

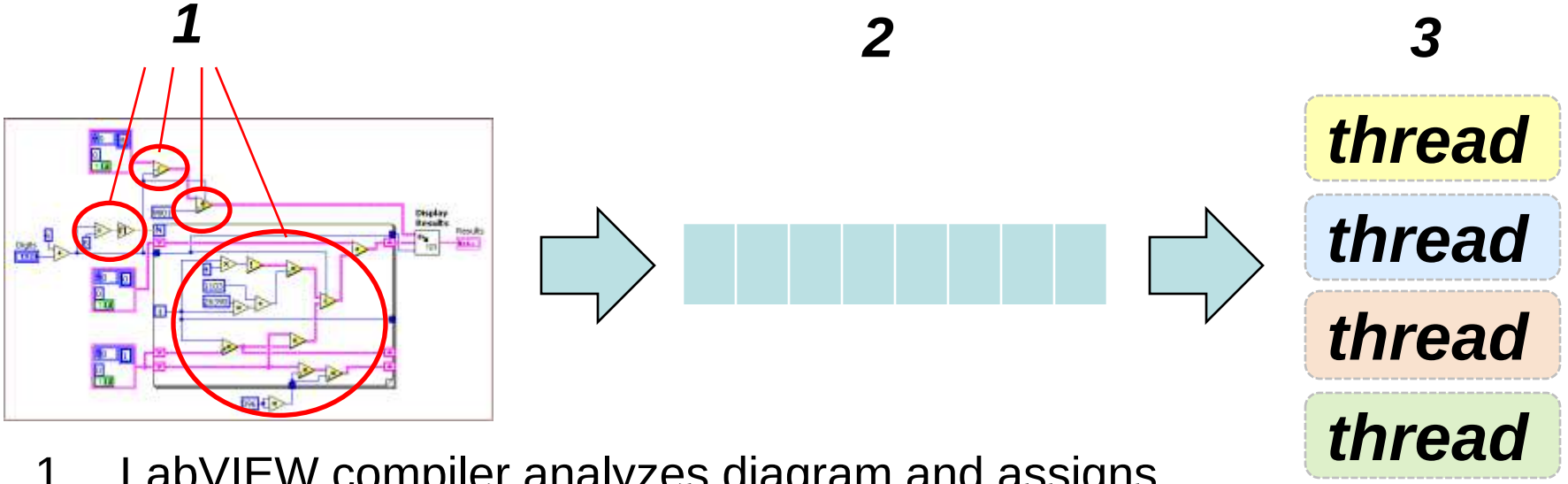


Automatic Multithreading in LabVIEW

- An oversimplification of threading in LabVIEW is shown below
- Main idea is parallel code paths will execute in unique threads



How the LabVIEW Execution System Works



1. LabVIEW compiler analyzes diagram and assigns code pieces to “clumps”
2. Information about which pieces of code can run together are stored in a run queue
3. If block diagram contains enough parallelism, it will simultaneously execute in all system threads

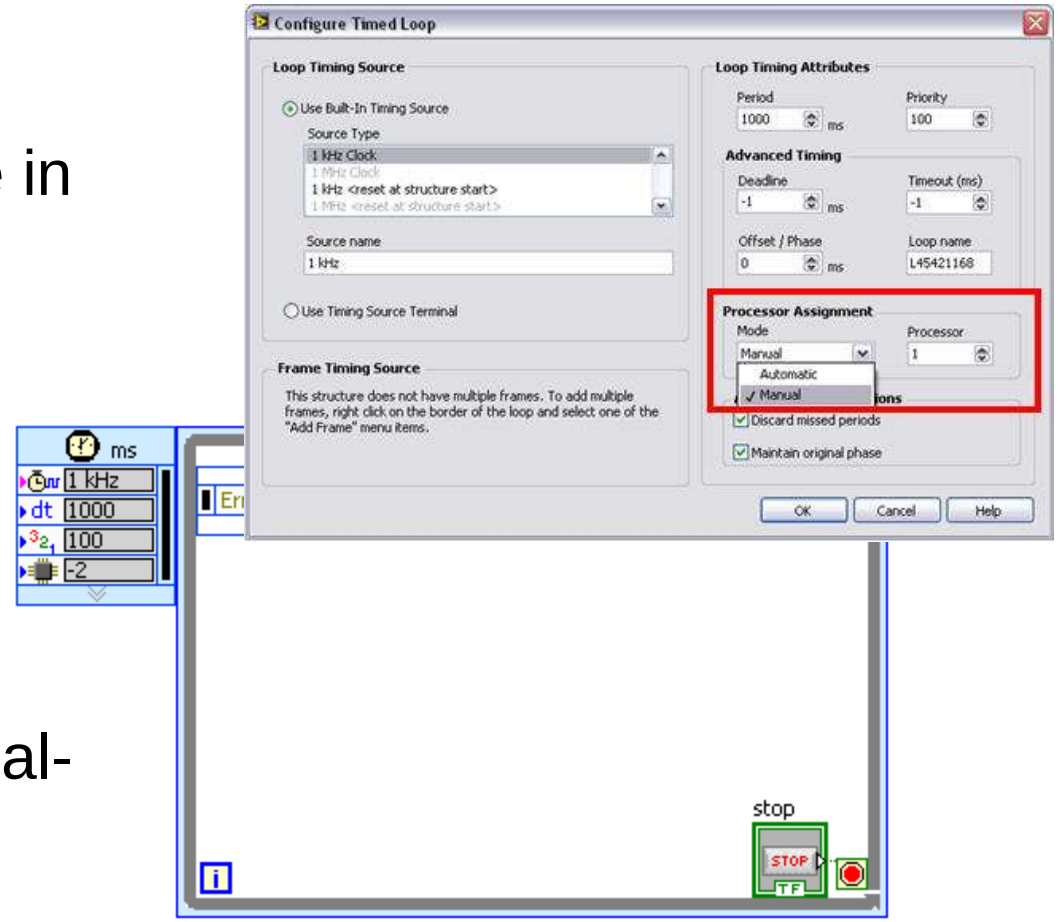
# of threads scales based on # of CPUs

Multithreading Features in LabVIEW 8.5/8.6

- Scale # of execution system threads based on available cores
- Improved thread scheduling for LabVIEW timed loops (to allow multicore support)
- Processor Affinity capability with Timed Structures
- Real-Time Features:
  - Support for Real-Time targets with Symmetric Multiprocessing
  - Real-Time Execution Trace Toolkit 2.0

Explicit Threading with Timed Structures

- Code within timed structures will execute in precisely 1 thread (no more)
- Can be assigned a relative priority
- Can be used to set processor affinity
- Recommended for Real-Time development

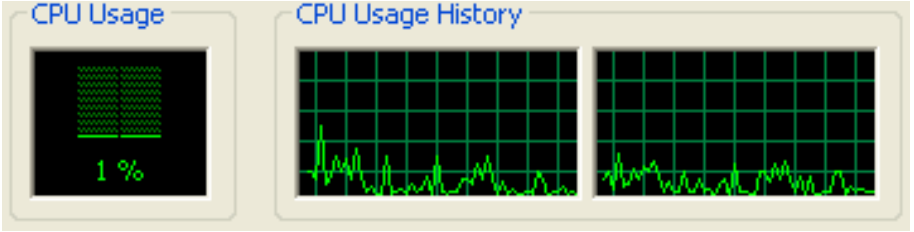


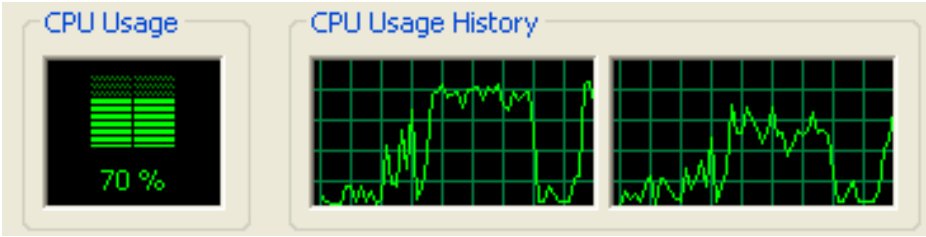
Demo Multithreading in LabVIEW

- Explicit Parallelism (Control of threads with Timed Loops)

PARALLEL PROGRAMMING TECHNIQUES

Some code is not worth making parallel…





- Don't parallelise code
  - Just because it's clever
  - With low CPU utilisation
  - I/O bound
- Do parallelise code that
  - Eats significant CPU cycles
- You need to get visibility of the runtime behaviour

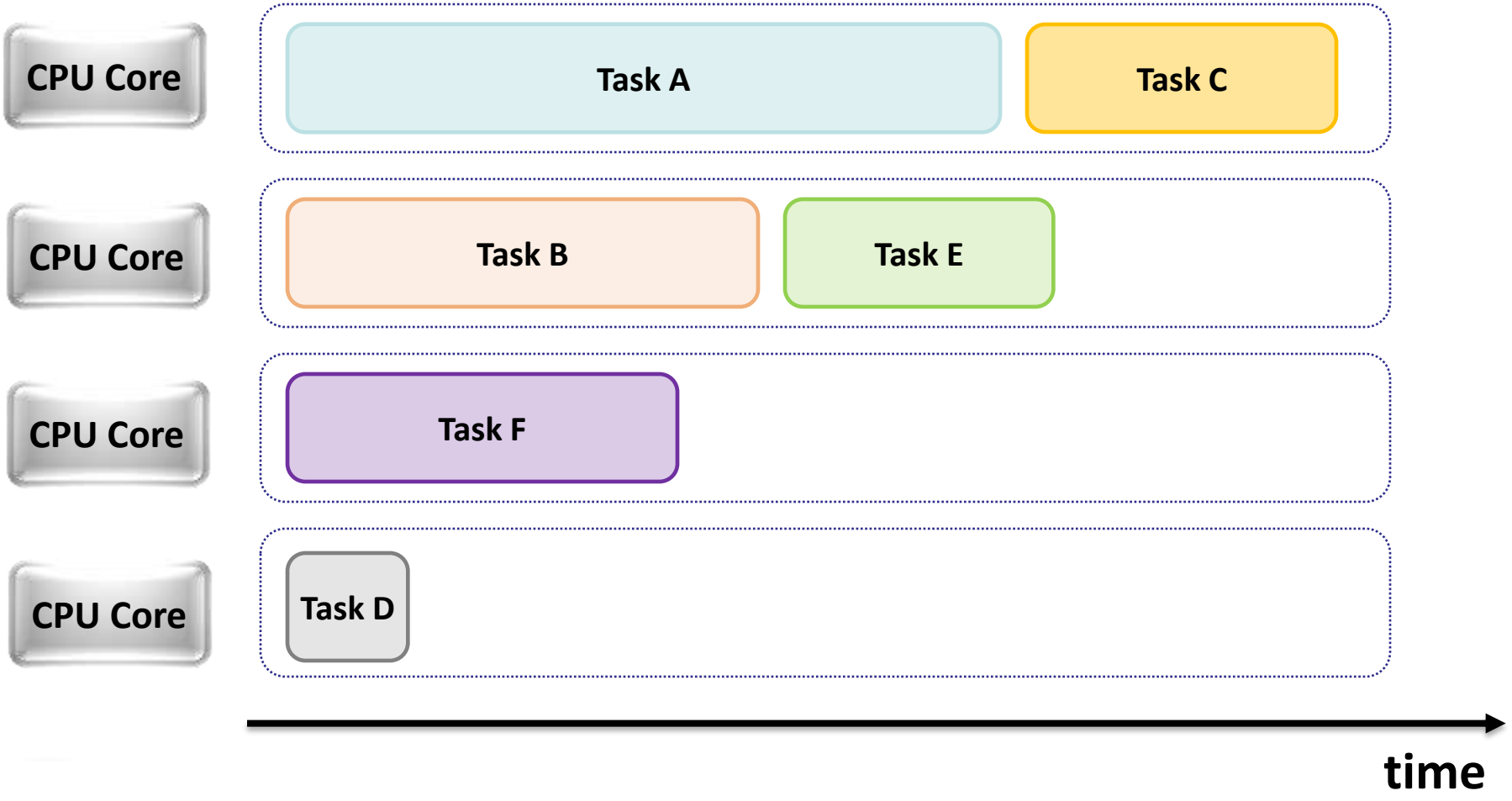
Multicore Programming Goals

- Increase code execution speed (# of FLOPS)
- Maintain rate of execution but increase data throughput
- Evenly balance tasks across available CPUs (fair distribution of processing load)
- Dedicate time-critical tasks to a single CPU (Real-time use-case)

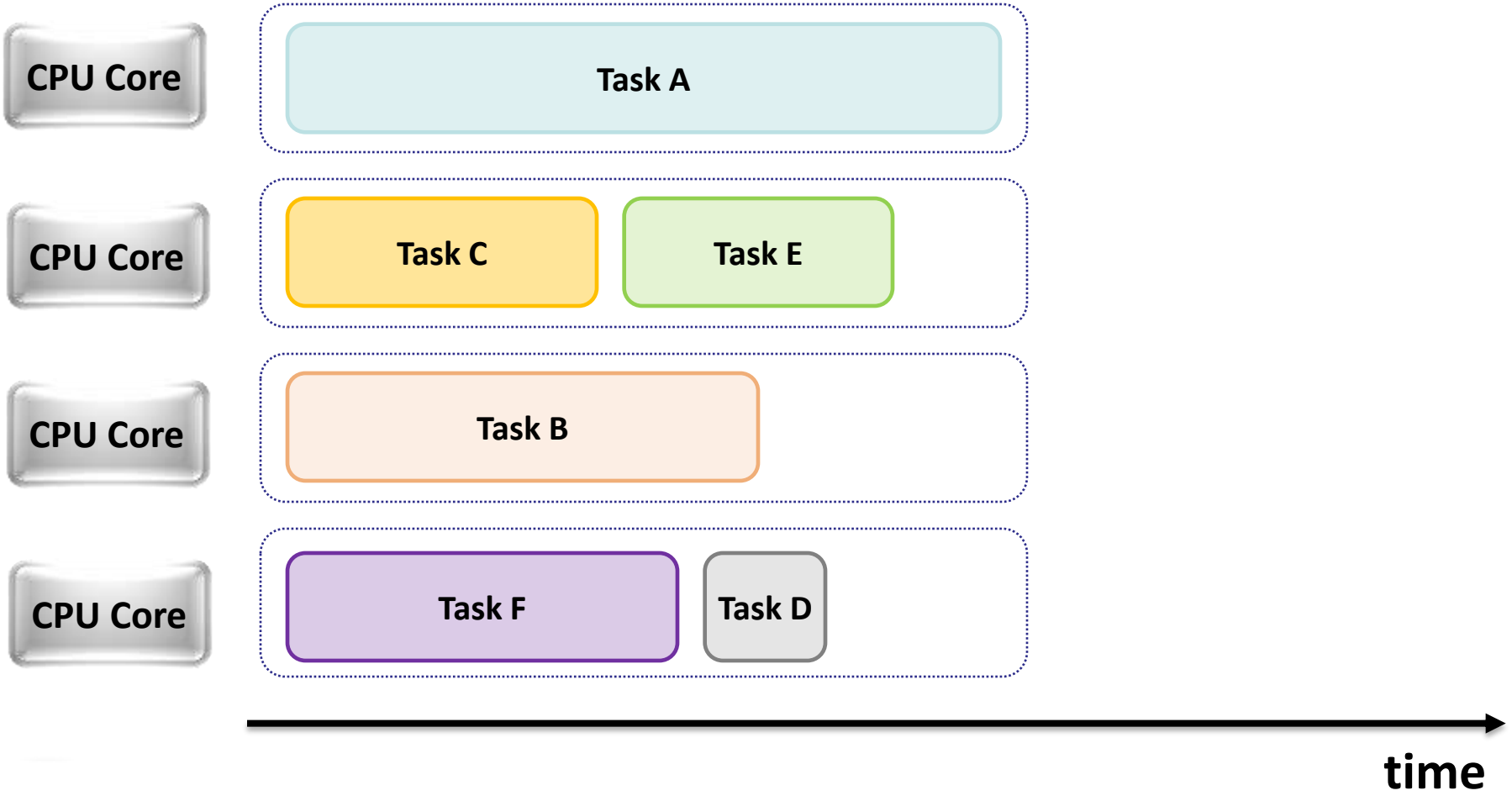
What are the Trade-offs?

- Parallel programming overhead is greater than sequential
- Optimizing for speed can come at a cost of more memory utilization

Example: Unbalanced Parallel Tasks



Goal: Balanced Parallel Tasks



Choosing the Right Parallel Strategy

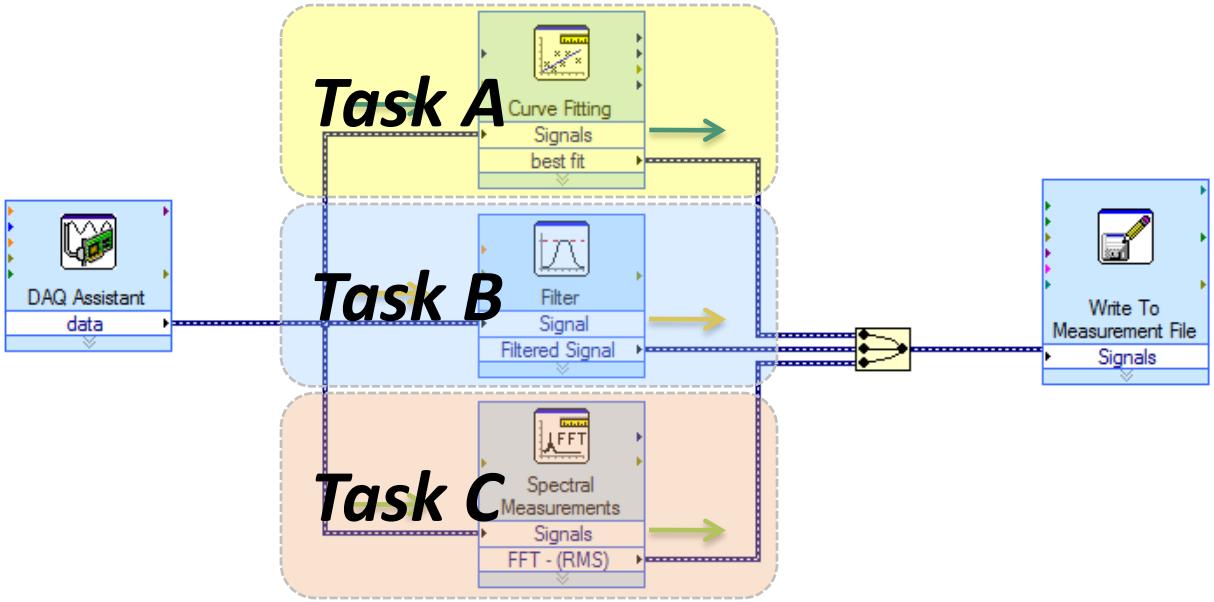
- Task Paradigm
  - Best suited for programs with independent functionalities that have minimal data dependencies
- Data Paradigm
  - Best suited for data operations that are completely independent
- Data Flow Paradigm
  - Best suited for data that have dependencies and require prior computation

Task Parallelism

- Tasks
  - Code is comprised of logically independent blocks of functionality
- Divide & Conquer
  - A section of code that can be decomposed into parallelized subsections, once completed results are merged together

Task Parallelism

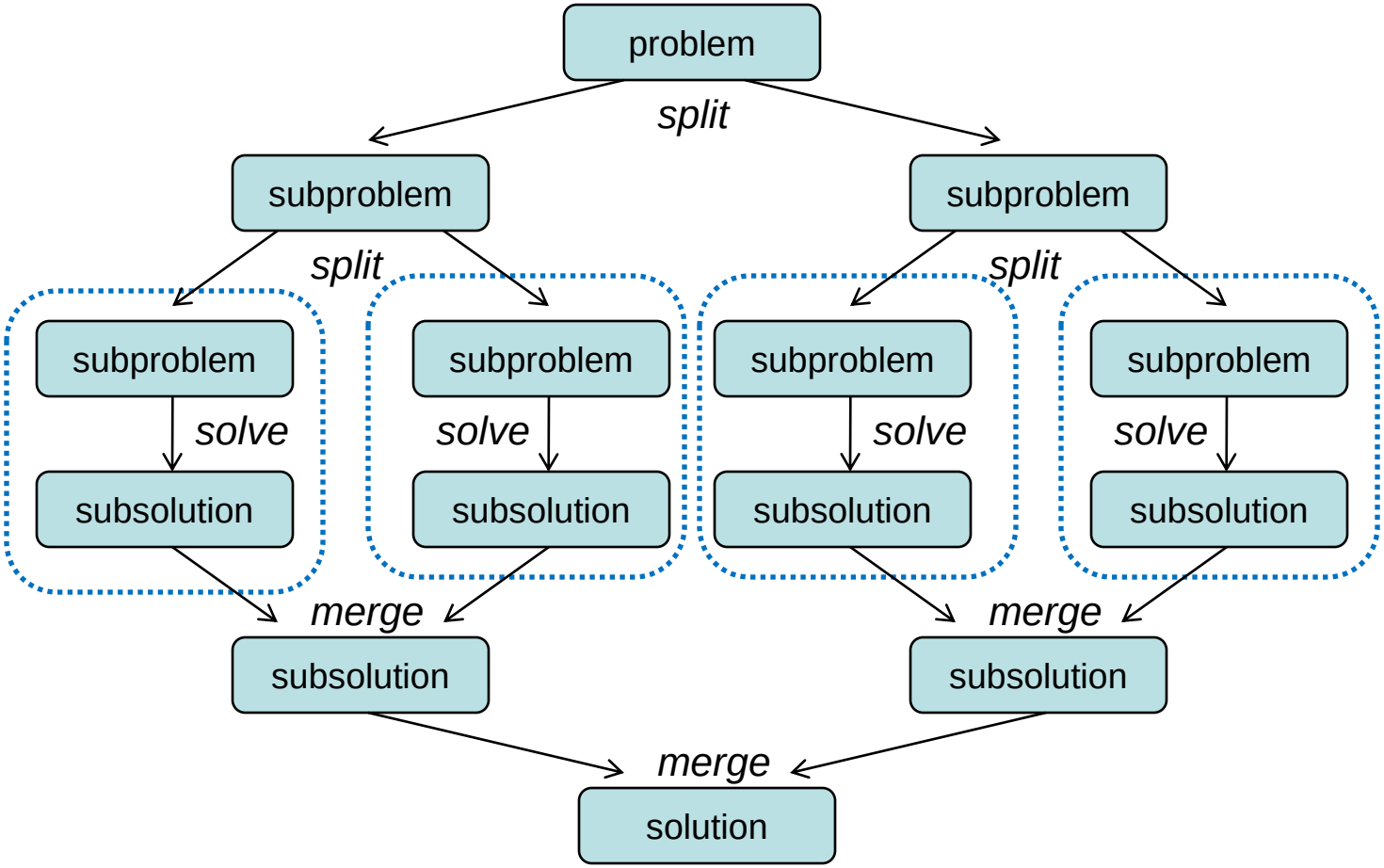
- Not all code requires sequential execution
- Isolate independent chunks of code and mark them as tasks



Divide & Conquer

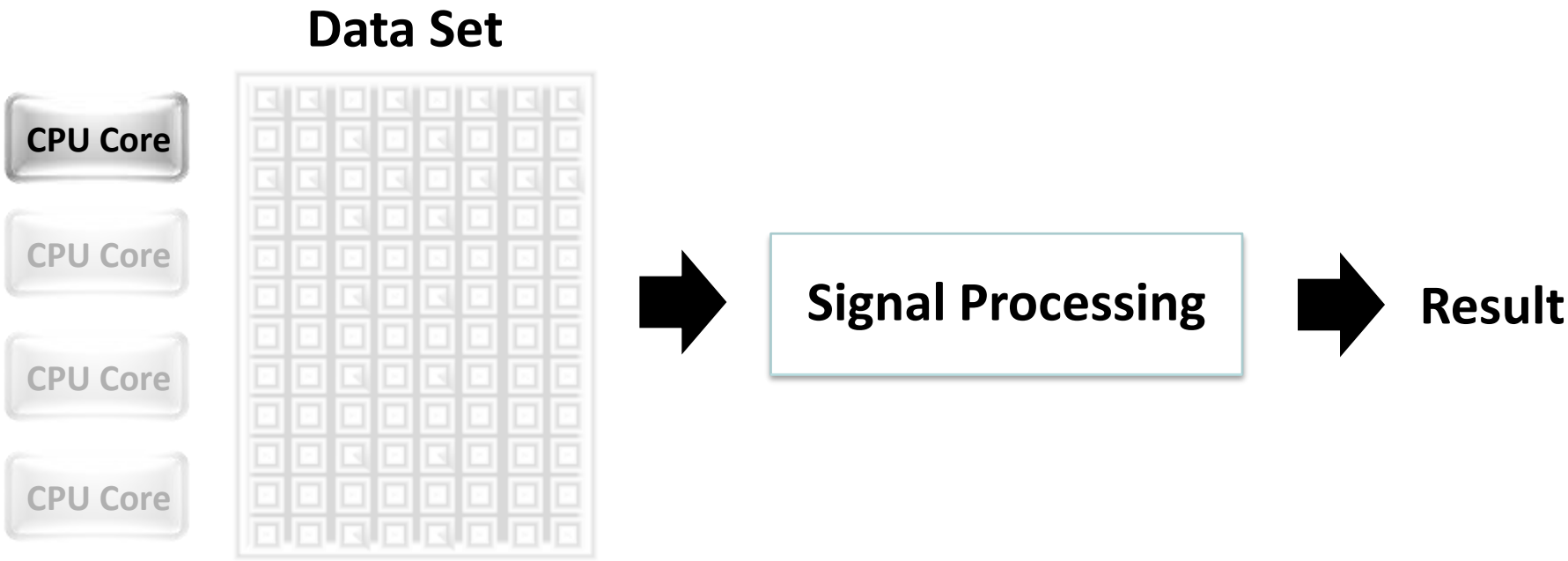
- Useful for problems that can be broken into subsections
- Recursive algorithms such as quick sort and merge sort
- Break the problem into manageable segments according to your resources

Divide & Conquer



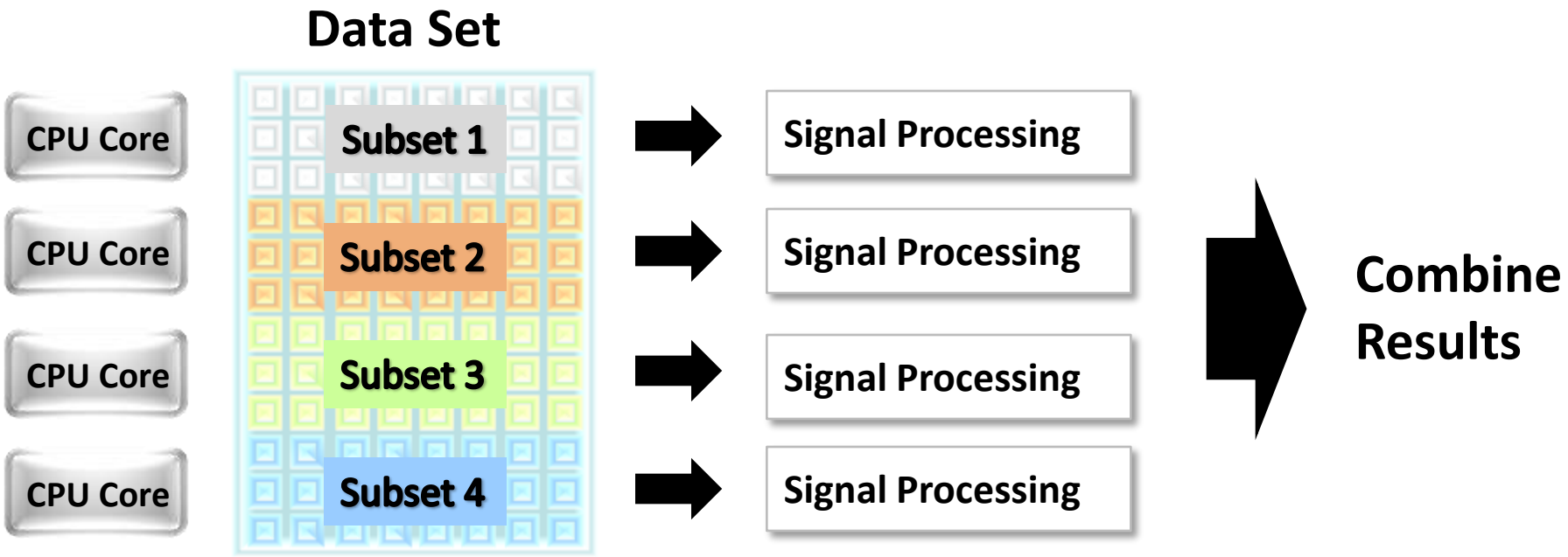
Data Parallelism

You can speed up processor-intensive operations on large data sets by segmenting the data.



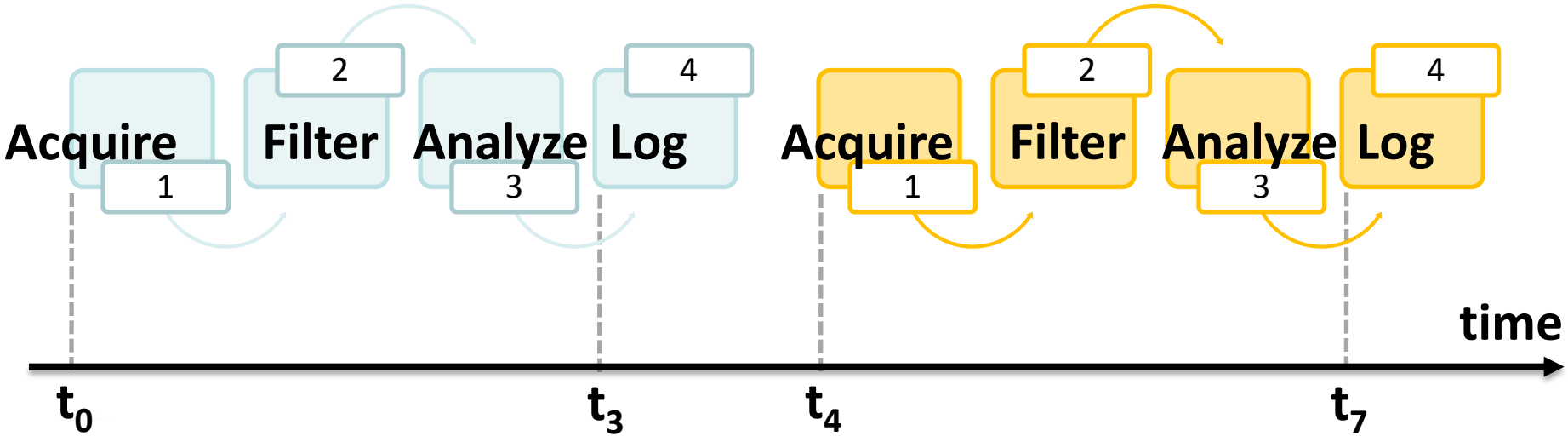
Data Parallelism

You can speed up processor-intensive operations on large data sets by segmenting the data.

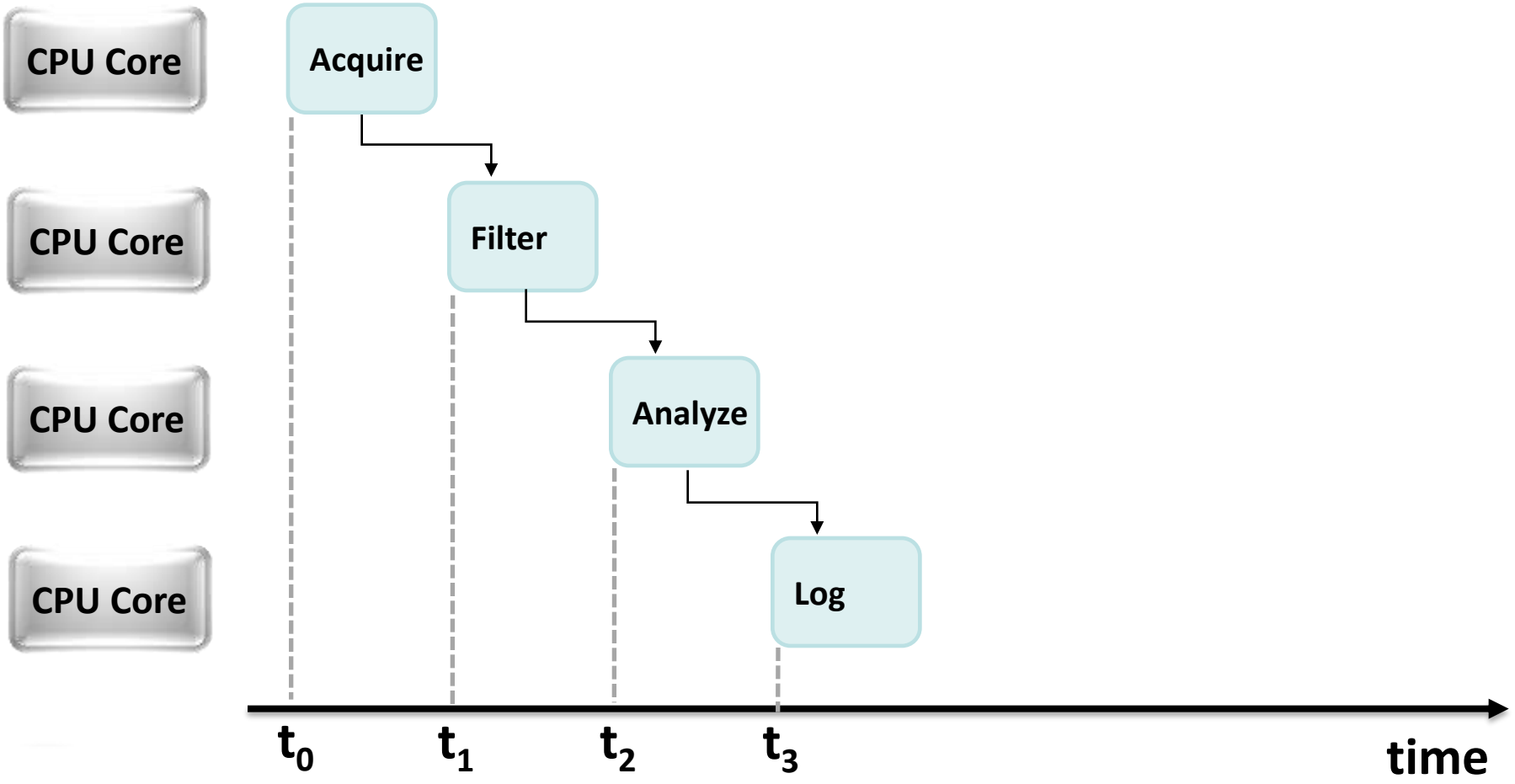


Data Flow Parallelism - Pipelining

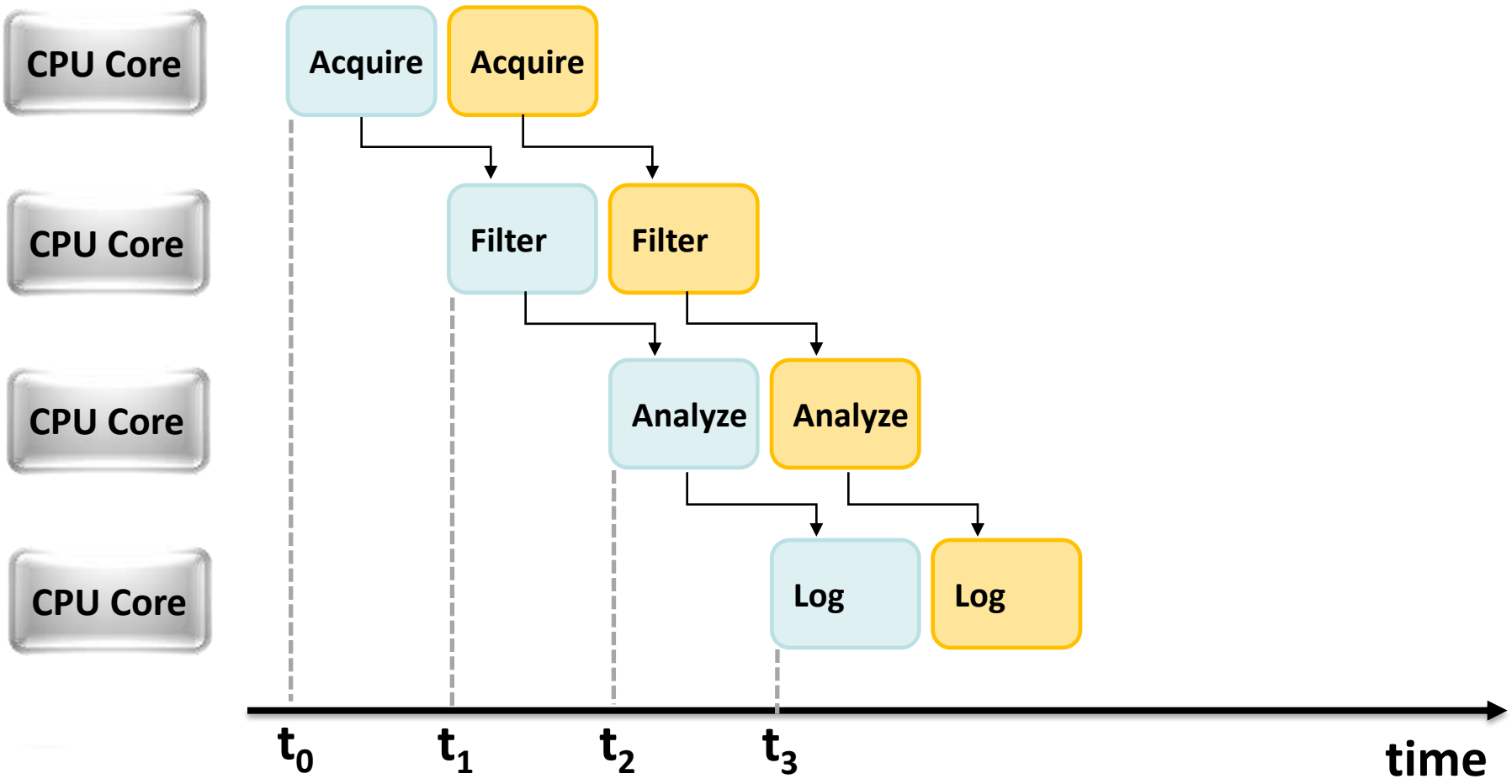
- Many applications involve sequential, multistep algorithms
- Applying pipelining can increase performance



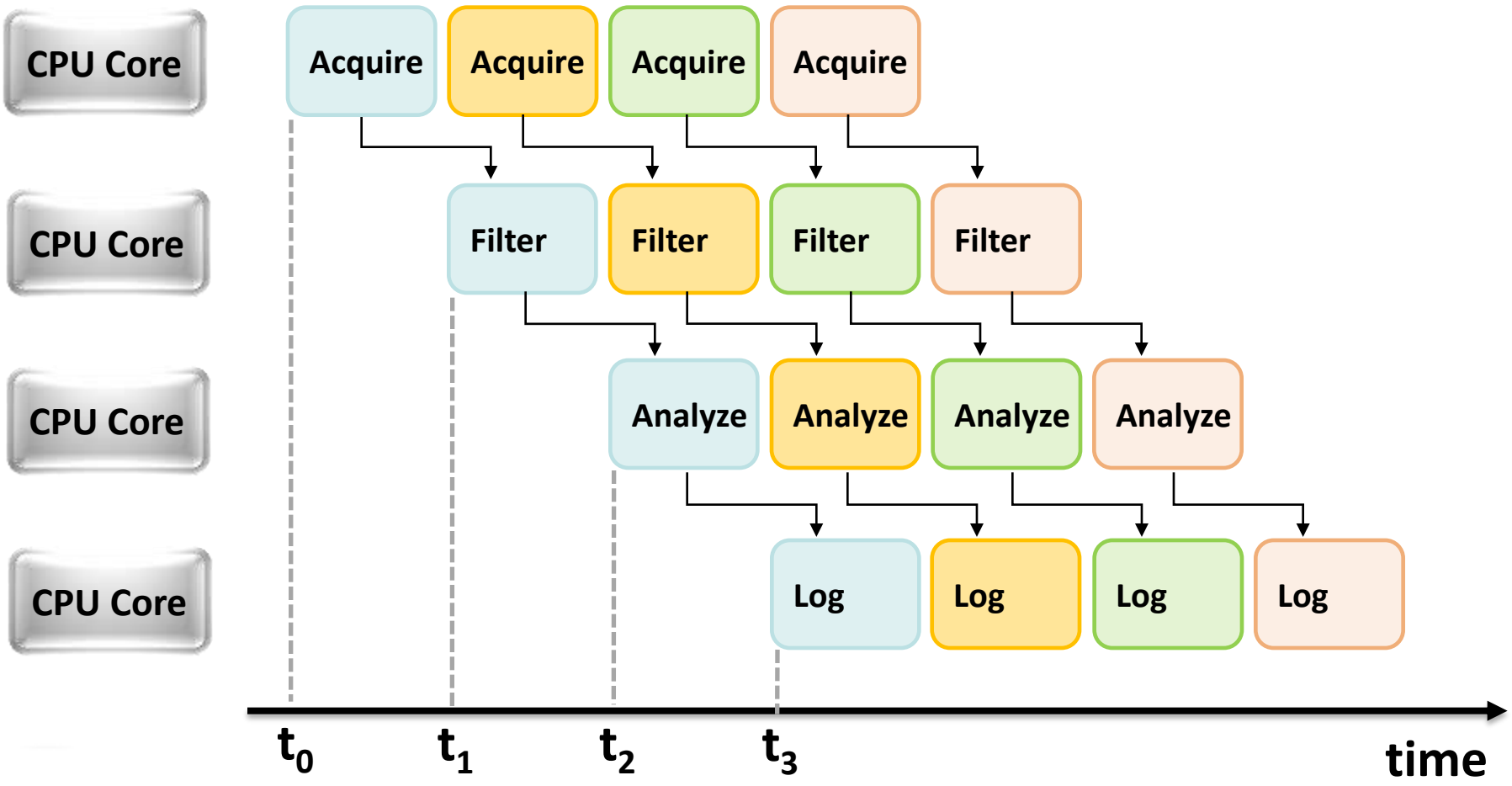
Pipelining Strategy



Pipelining Strategy

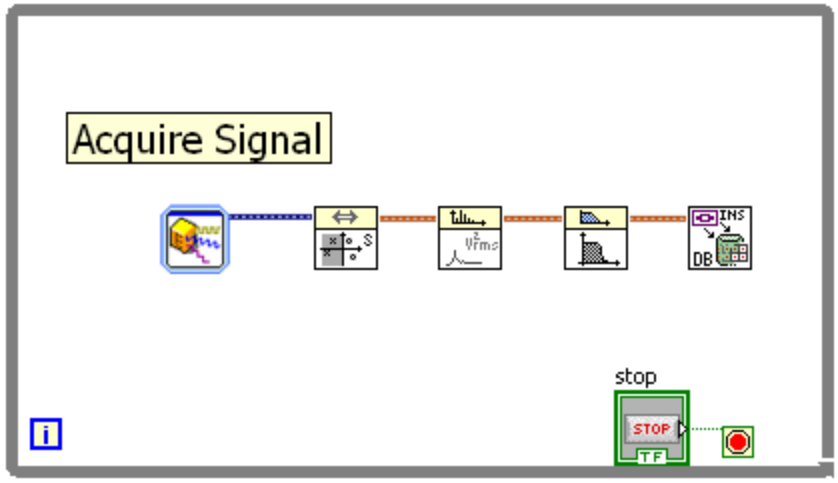


Pipelining Strategy

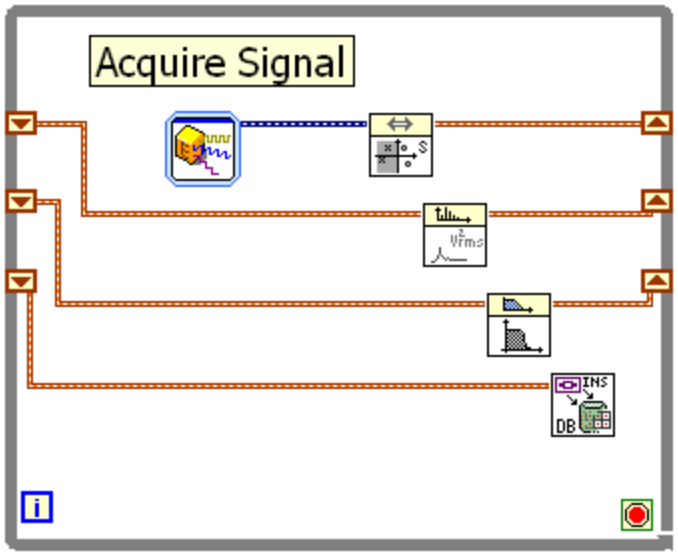


Pipelining in LabVIEW

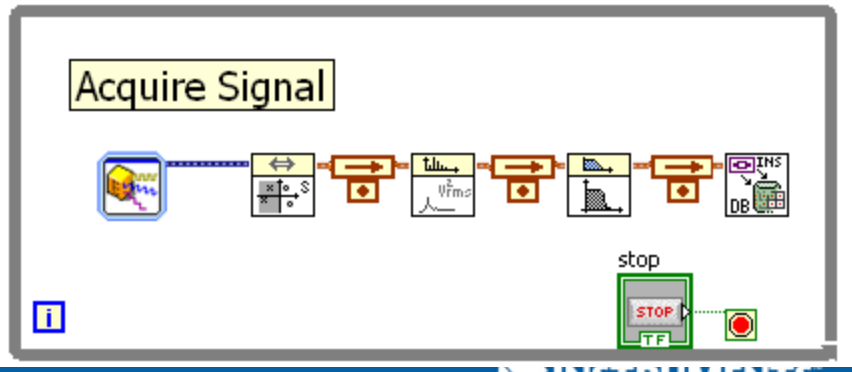
Sequential



Pipelined



or



Note: Queues may also be used to pipeline data between different loops

Demos Parallel Programming Techniques

- Data Parallelism
- Pipelining

MULTICORE PROGRAMMING CHALLENGES

Multicore Programming Challenges

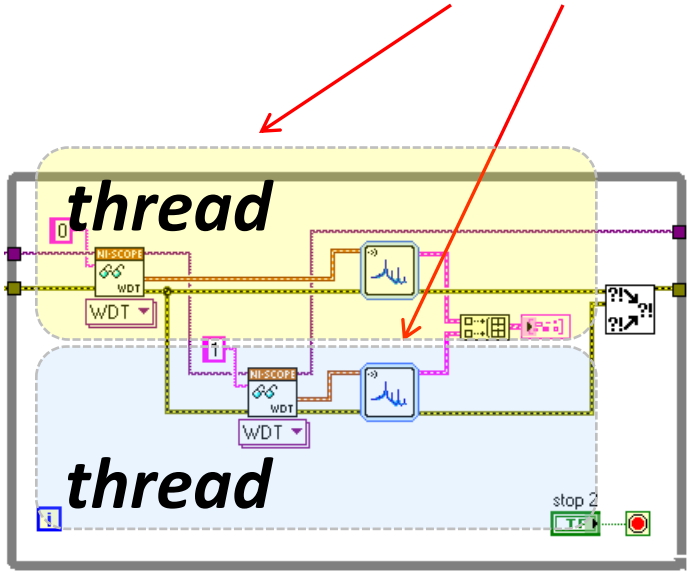
- Programming Caveats
  - Thread Synchronization
  - Race Conditions
  - Deadlocks
  - Shared resources

- Memory
  - Data transfer between processor cores and cache considerations
- Debugging

Addressing Multicore Challenges with LabVIEW

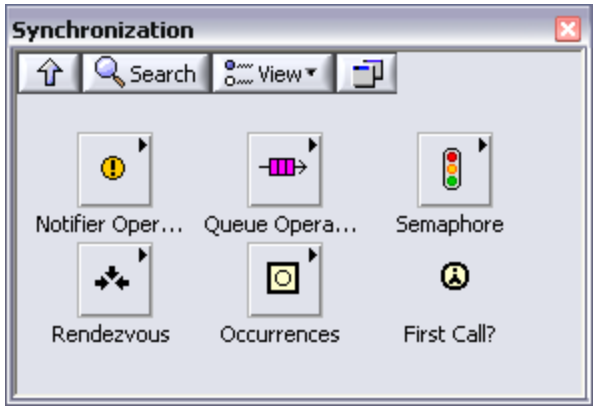
- Dataflow paradigm is a key benefit for multicore programming
  - Helps synchronize threads and prevent race conditions/deadlocks

Parallel code paths are synchronized and order of execution is determined by LabVIEW wires



Synchronization in LabVIEW

- When more synchronization is required, use synchronization mechanisms:
  - Notifiers
  - Queues
  - Semaphores
  - Rendezvous
  - Occurrences

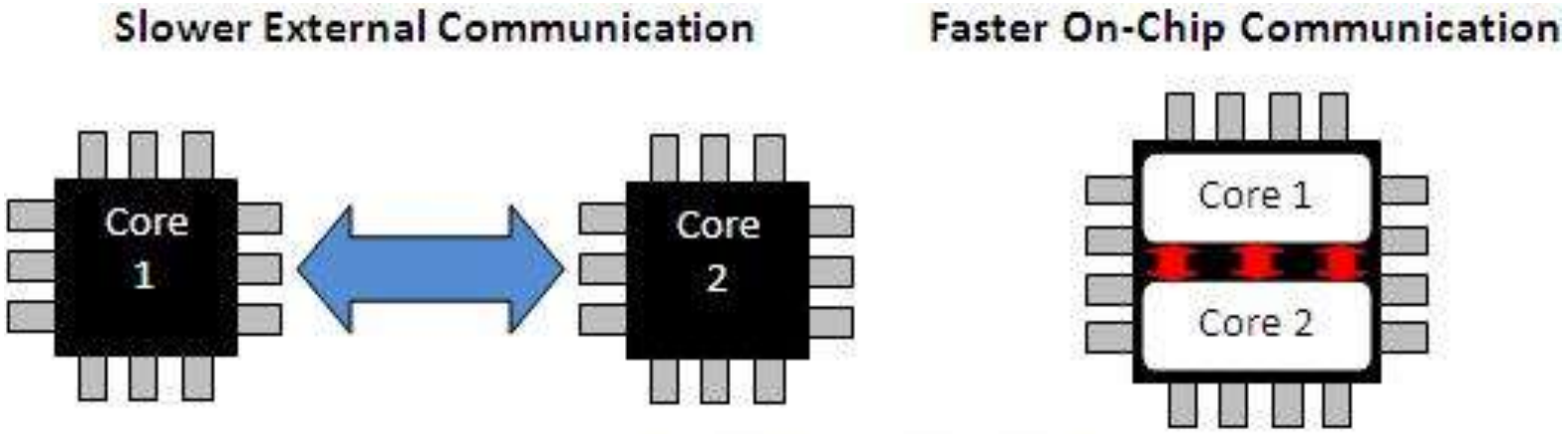


Memory Considerations

- Data transfer between cores
- Cache considerations

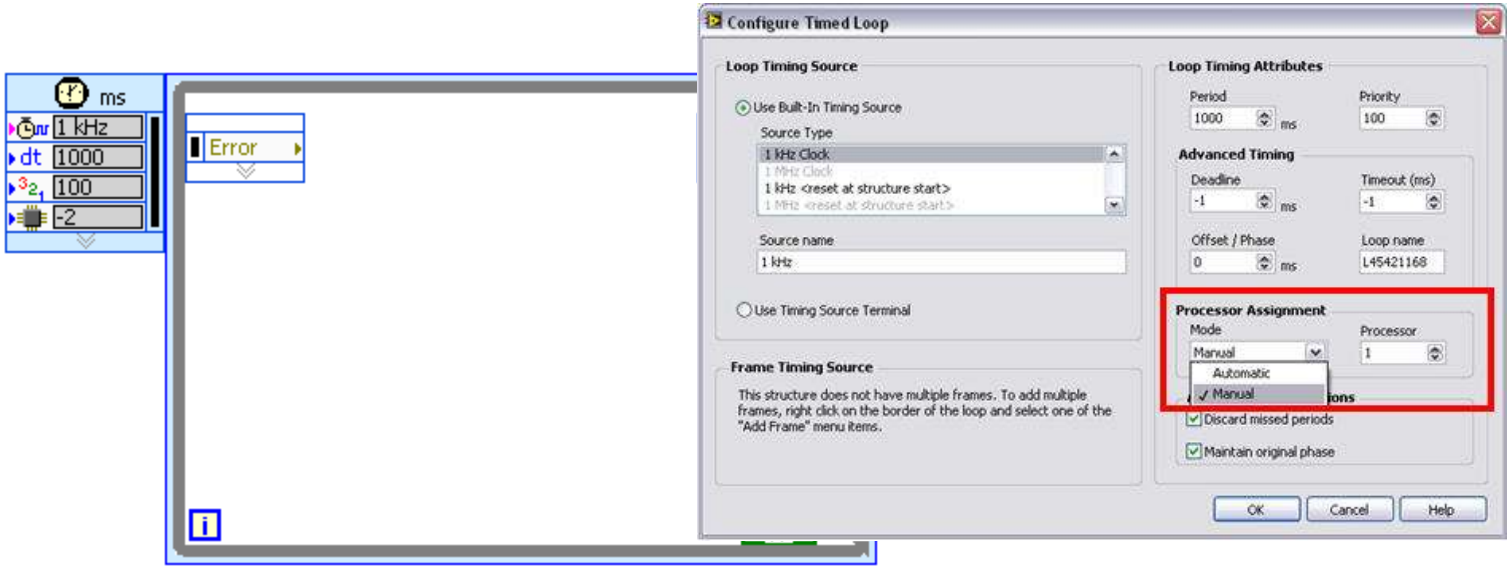
Data Transfer Between Cores

- Physical distance between processor and the quality of the processor connections can have large effect on execution speed



Cache Optimization with Processor Affinity

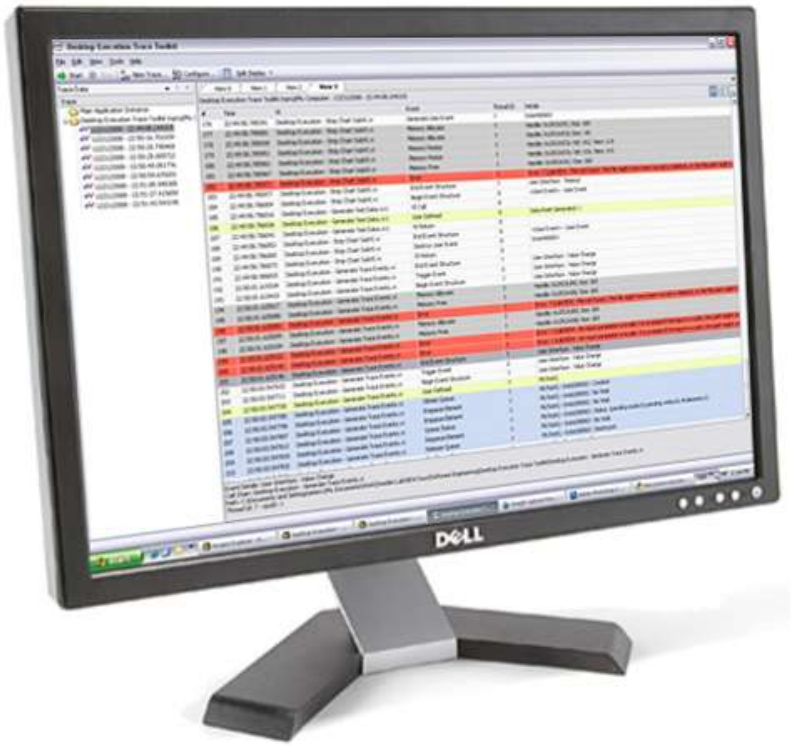
- Setting processor affinity tells the OS which processor to execute the code on
- Processor affinity can prevent OS from scheduling threads in a configuration that hurts cache usage



Demo

- Optimal Data Transfer

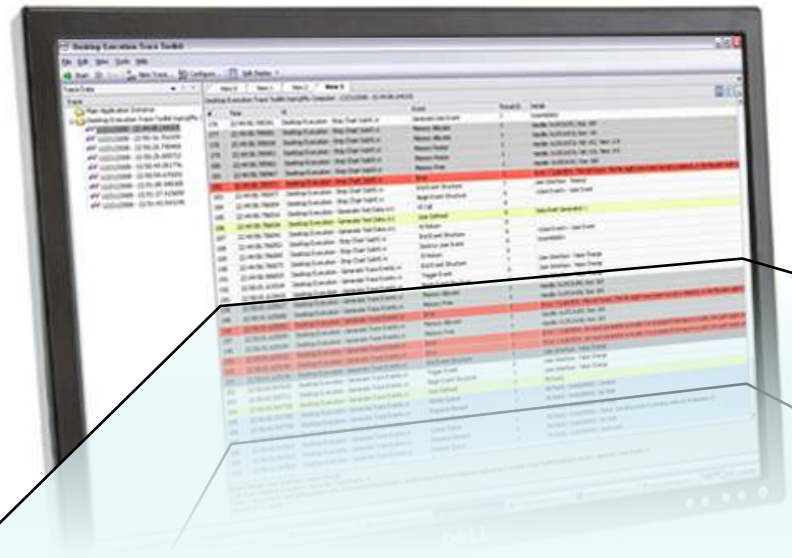
Debugging:
Desktop Execution Trace Toolkit



Trace During Run-Time:

- Event Structures
- Memory Allocation
- Queues / Notifiers
- Reference Leaks
- Thread ID
- Unhandled Errors
- Dynamic / Static SubVIs
- Custom User Strings

Debugging:
Desktop Execution Trace Toolkit



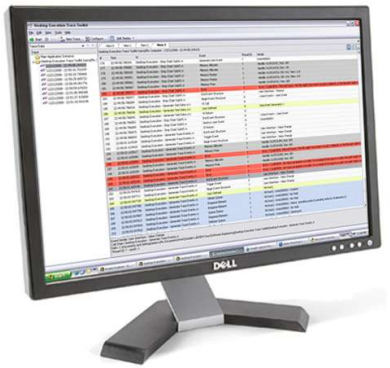
Trace During Run-Time:

- Event Structures
- Memory Allocation
- Queues / Notifiers
- Reference Leaks
- Thread ID
- Unhandled Errors

Strip Chart SubVI.vi	Memory Resize	7	Handle: 0x25CA3C8; Old: 142; New: 118
Strip Chart SubVI.vi	Memory Allocate	7	Handle: 0x25CA3C8; Size: 142
Strip Chart SubVI.vi	Memory Resize	7	Handle: 0x25CA3C8; Old: 142; New: 118
Strip Chart SubVI.vi	Error	7	Error: 7 (LabVIEW: File not found. The file might have
Generate Trace Events.vi	User Defined	7	MyTestQ
Generate Trace Events.vi	Obtain Queue	7	MyTestQ - 0x66200002 : Created
Generate Trace Events.vi	Enqueue Element	7	MyTestQ - 0x66200002 : No Wait

Trace Production Systems Remotely

LabVIEW Desktop Execution Trace Toolkit



Run-Time Execution Information

Network

VIs and Debuggable Executables

LabVIEW Real-Time Execution Trace Toolkit

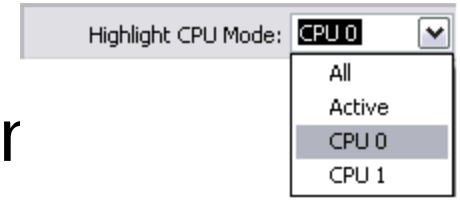
Run-Time Execution Information

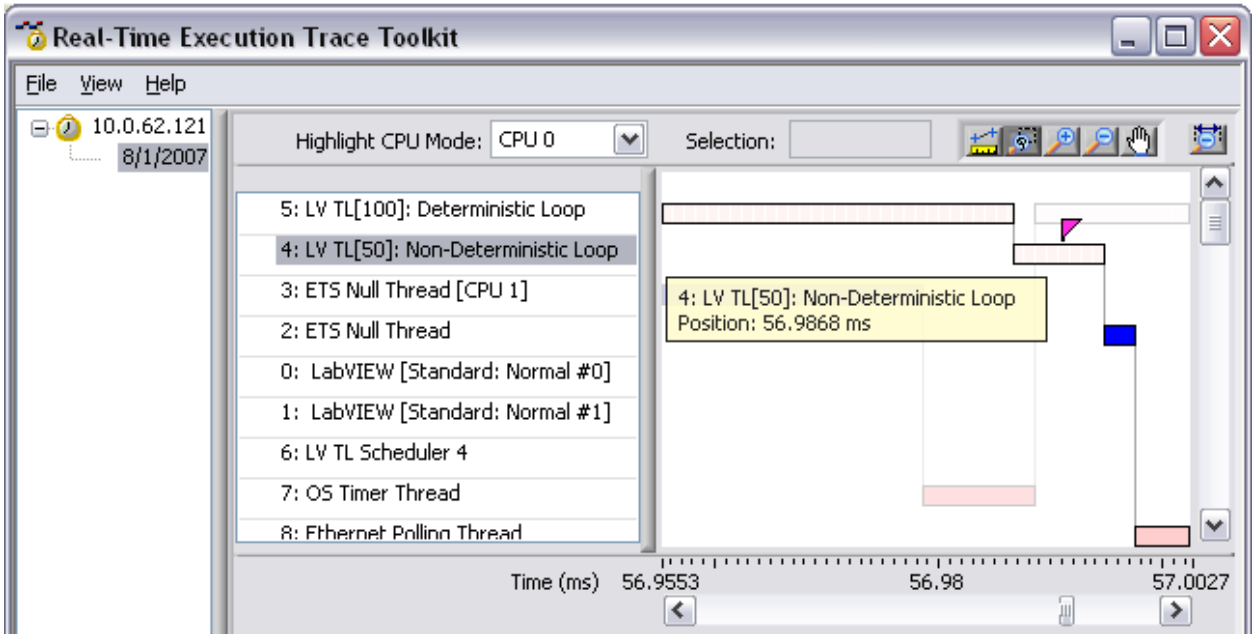
Network

Deployed Real-Time Applications

Debugging Multicore Applications on Real-Time Systems

- Highlight CPU Mode
- Simultaneously view traces runnir on separate cores





Conclusions

- **Multithreading** - LabVIEW offers implicit parallelism (automatic multithreading) and explicit parallelism (timed structures)

- **Parallel Programming Techniques** - There is no silver bullet, the advantage of LabVIEW is that parallelism is much easier expressed by the language

Conclusions (continued)

- **Multicore Programming Challenges** - Debugging and memory considerations have evolved with multicore and play an important role

- With minor modifications, typical LabVIEW applications can be optimized for multicore

Resources

www.ni.com/multicore