
LABVIEW: TORTOISE OR HARE?
Nate Moehring
Sept 19th, 2017
SPACEX

Foreword

- This presentation contains a collection of best practices for writing highly performant LabVIEW applications as learned by top LabVIEW developers at SpaceX and presented in pure academic form for the benefit of the greater LabVIEW community.
- This presentation does not contain any information specific to SpaceX in order to remain in compliance with ITAR regulations.
- This presentation is not an official marketing outreach of SpaceX, but has received approval from the Marketing and Communication office for use in the 2017 CLA Summit.
- All pictures in this presentation are publically available on the Internet.

SpaceX regularly challenges LabVIEW in terms of functionality, performance, and scalability. Many of the patches released for LabVIEW 2013 and 2015 have been in direct support of SpaceX. Thank you to LabVIEW R&D for their support.

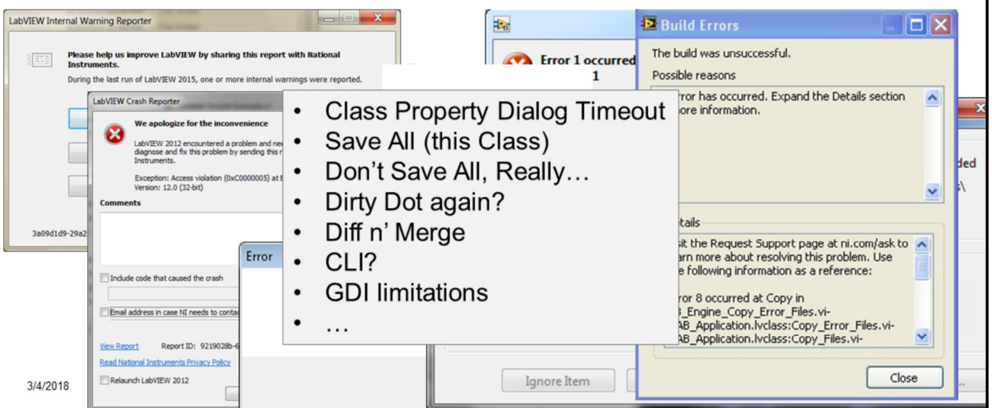
Characterizing Ground Software

- Automation for Test, Launch, and Mission Ops
- Primary User Interface for Operators
- Industrial Control of Ground Support Equipment
- Data Aggregation, Processing, Distribution

- Distributed Architecture
- Heavy OOP

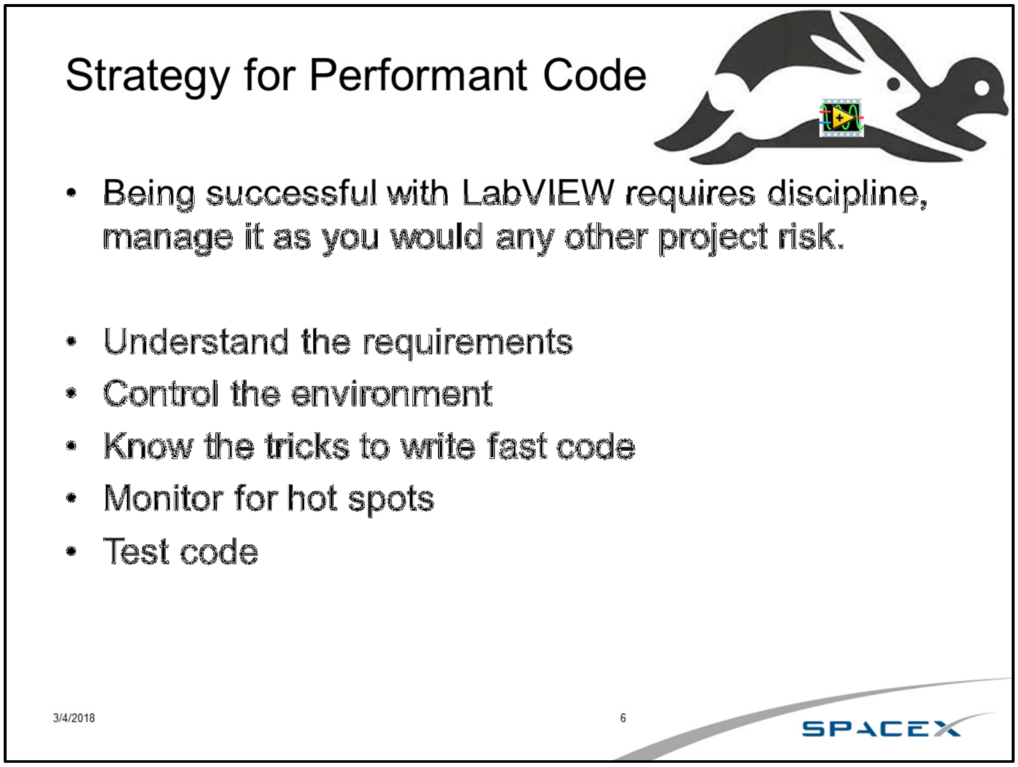
LV: Tortoise or Hare?

- G allows us to be prolific SW developers
- But LabVIEW carries a lot of baggage and getting high performance can be…tricky.



- Class Property Dialog Timeout
- Save All (this Class)
- Don't Save All, Really…
- Dirty Dot again?
- Diff n' Merge
- CLI?
- GDI limitations
- …

3/4/2018



This presentation will focus on managing the risk associated with LabVIEW performance and touch on issues to help reduce risk of failures through real-time metrics analysis. An entirely separate presentation could be written to discuss managing the risks associated with *some* of the problems listed on the previous slide.

Understand the Requirements

- Performance
- Opportunities for Parallelism
  - Think scalability, redundancy?
- Interfaces
- Shared resources

- Don't make the wrong thing…really fast
- Only optimize where necessary
- Prototype to buy down performance risk

3/4/2018 7 SPACEX

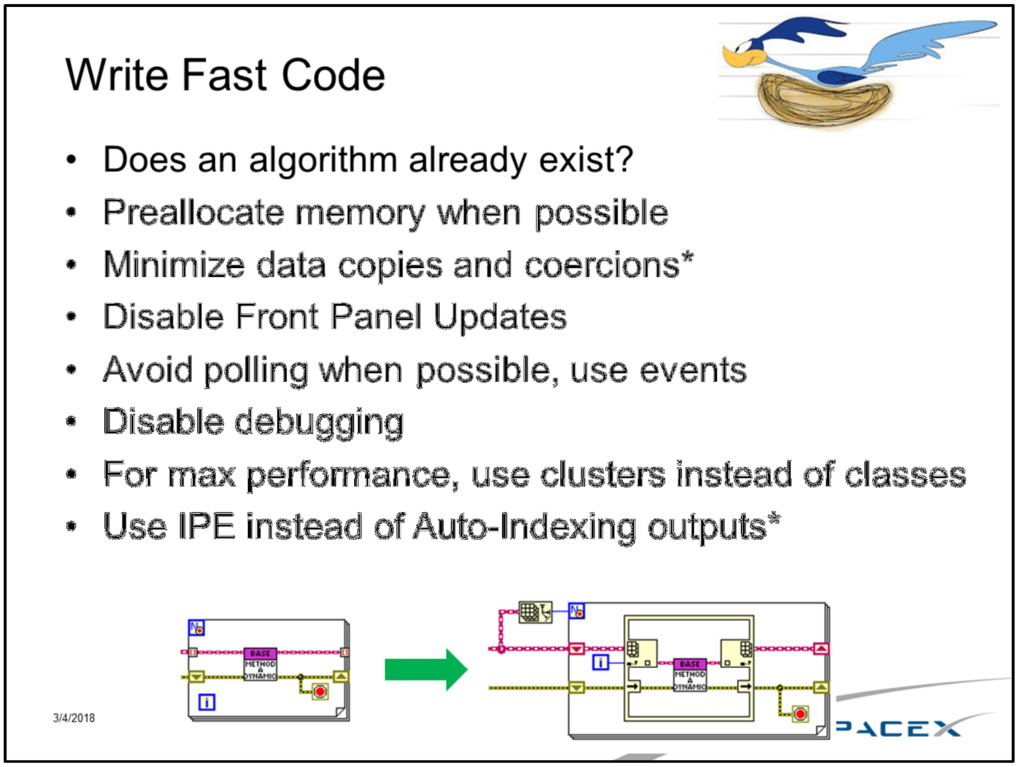
Throughput, latency, jitter

Control the Environment

- Standardize
  - Supported hardware targets
    - Minimum RAM, CPU, NICs, SSDs, RAIDs, etc…
  - OS + driver + app Images
    - ex: Disable SVN Icon Overlay Caching
    - ex: OpenGL driver version
  - BIOS/driver settings
    - Power-save settings
    - NIC teaming
- Isolate the networks if possible
  - Virtual LANs
  - Disable Automatic Updates
  - Talk about Group Policy updates with IT
  - Firewall rules
- Need a strong IT team to manage these well

3/4/2018 8 SPACEX

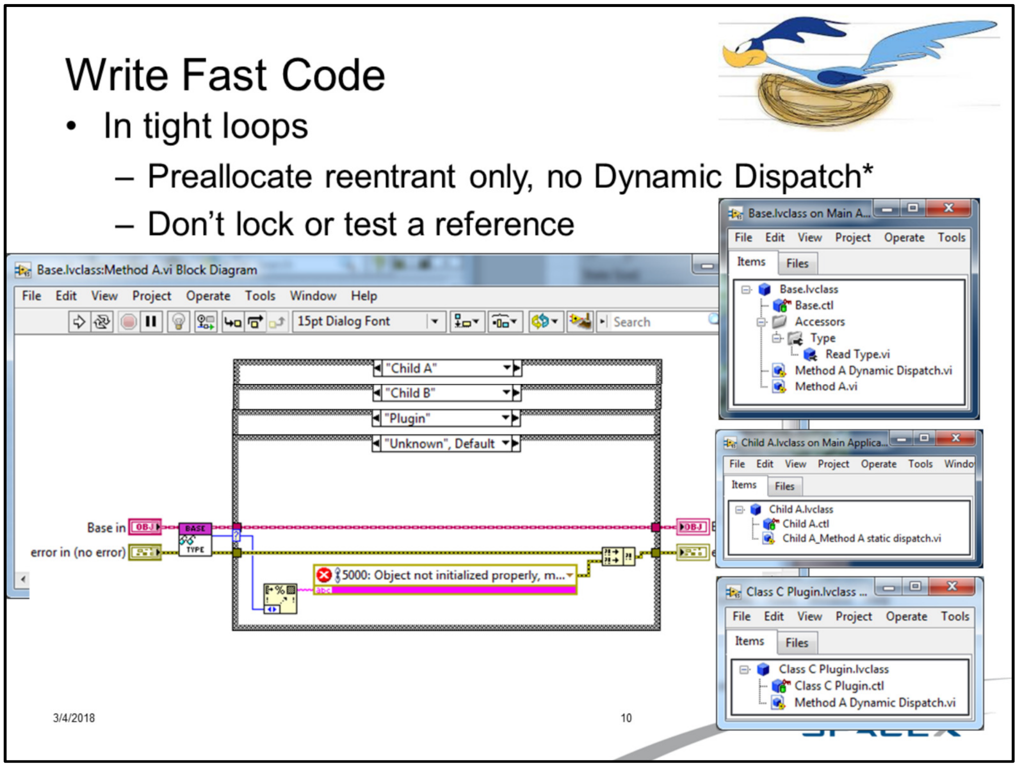
This is probably the most important slide in this presentation! This is where we've had significant struggles and wins.



TagBus is a great way to accommodate preallocation and minimize data copies and coercions

This might be a 2013 bug, or it might be a function of the datatype or dynamic dispatching. The In Place Algorithm will likely recognize that the output can reuse the input array memory space, but semantically this is not expressed or promised by this traditional use of auto indexing inputs and outputs. The diagram on the right, although less concise, does the express the desired behavior semantically.

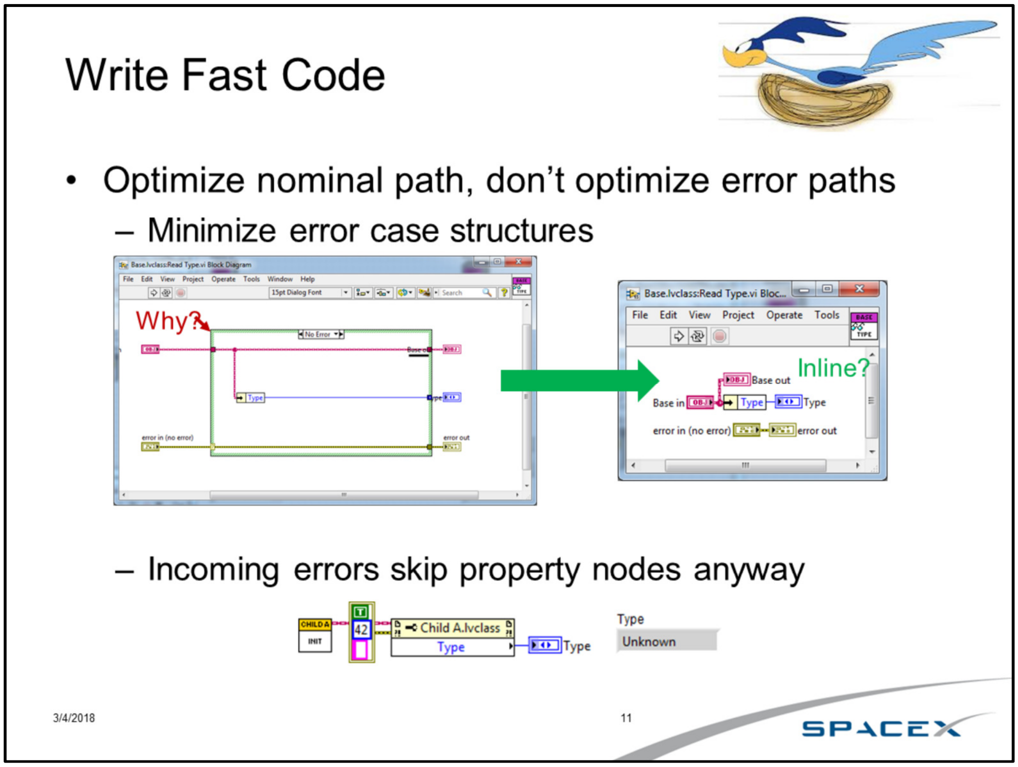
Note that the size of the output arrays will be different with the shown implementation, which basically assumes that if an error occurs the application will be shutting down, so the change in array size is not significant. You would need to remove the conditional terminal from the loop on the left in order to guarantee the inputs and outputs match length.



Static Dispatch is faster than Dynamic Dispatch by single digit microseconds. If you're writing code to avoid dynamic dispatch, consider the overhead of that code adds to the Static Dispatch time. It's possible that dynamic dispatch is now fast enough that this trick is no longer required, but in LV2015 I do not believe that to be the case.

Dynamic Dispatch defaults to Shared Clone Reentrancy. Non-reentrant is an option for Dynamic Dispatch (excludes recursion), but Preallocate Reentrancy is not. Shared Reentrancy can introduce jitter in an application.

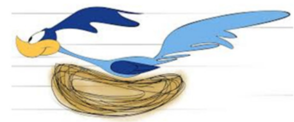
To More Specific can be a potentially expensive operation but in this case it is not because it is known thanks to the case structure that the object *is* of the requested type, and the types are only 1 generation apart from each other. No search down the ancestry required to find the matching type.



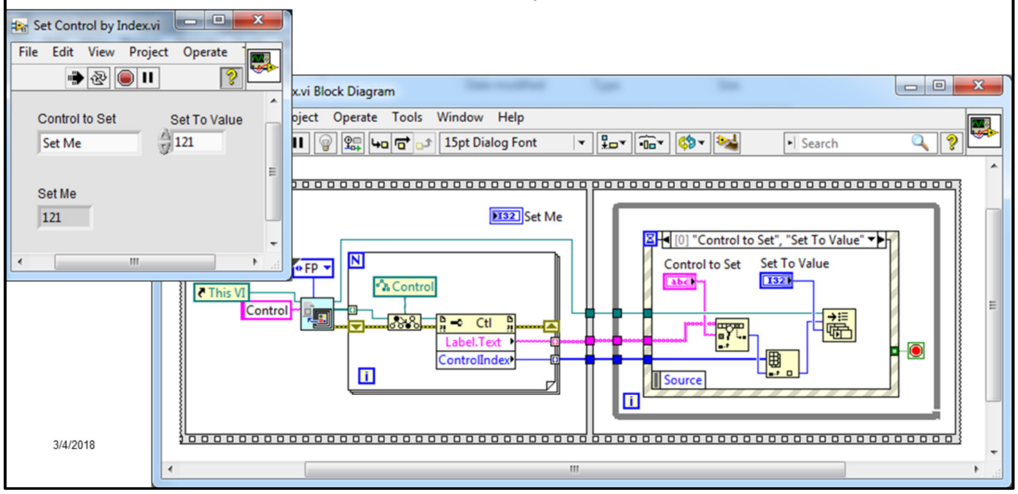
Although unused, error terminals are required on this Read Accessor in order to make the accessor callable in a property node.

Consider modifying the Read Accessor template VI in the resource folder in order to change the default construction of this VI.

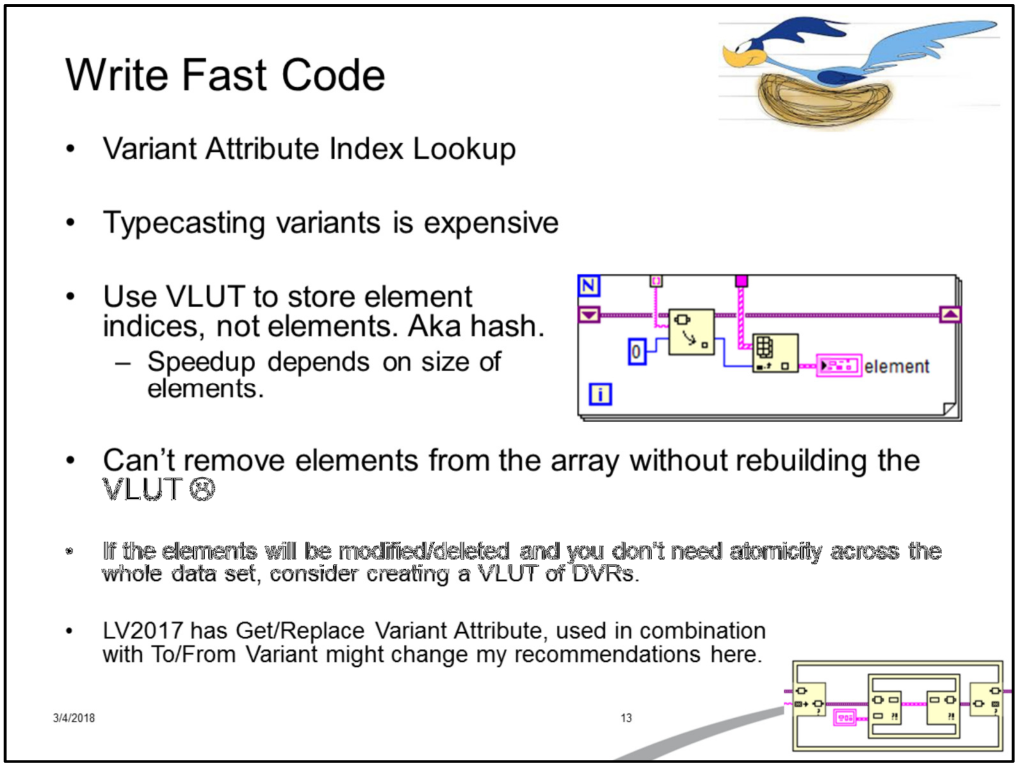
Write Fast Code



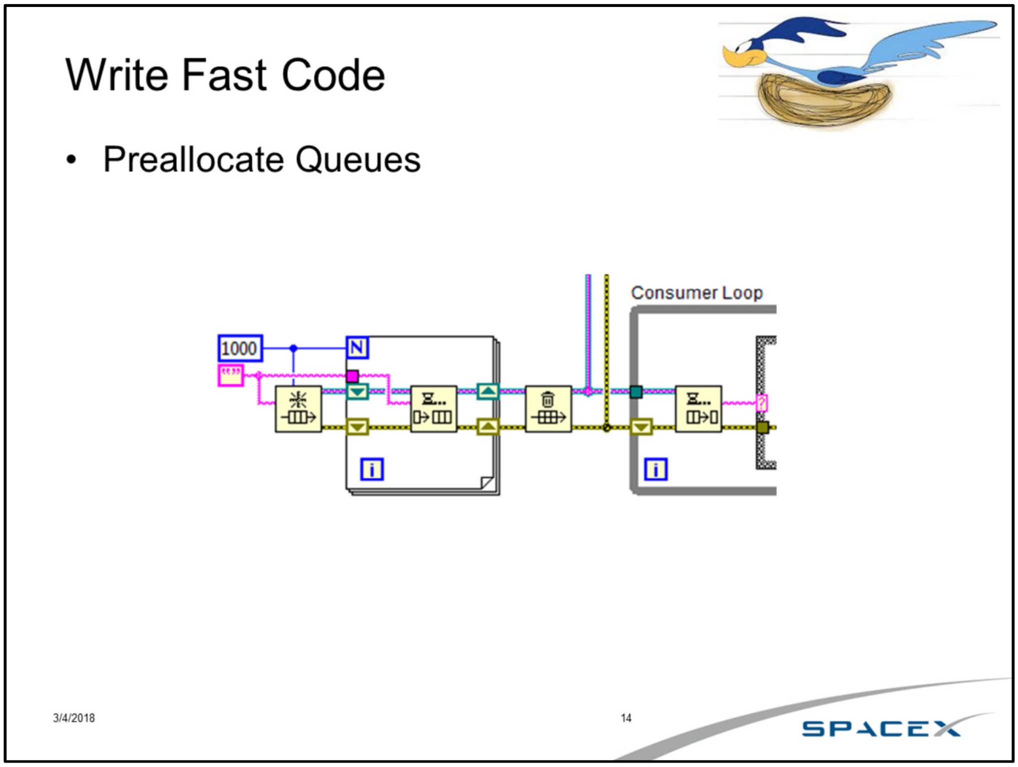
- **Set Control By Index**
  - Set values by reference without using a reference
  - Runs in Execution Thread, not UI Thread



Available since 2013

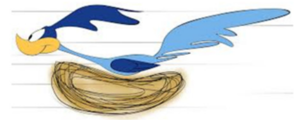


We haven't upgraded to LV2016 or 2017 yet, so I haven't experimented with the new IPE nodes yet. Allen Smith suggests this will likely be more beneficial if you're using nested attributes.

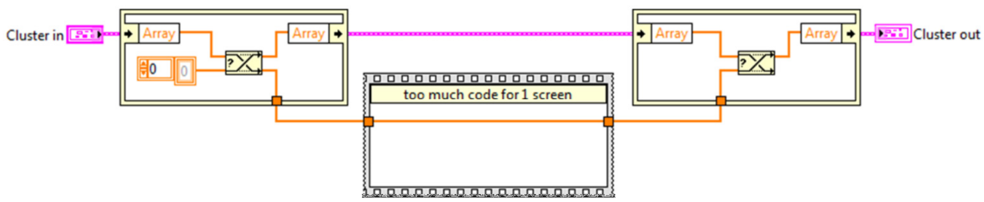


Queues amortize their growth by allocating memory in increasing-sized blocks. Therefore the next enqueue operation might require a memory allocation to make room for the new element. Preallocating the queue prevents jitter because no memory allocation will occur during execution.

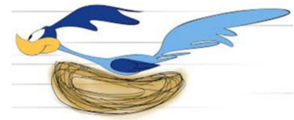
Writing Fast Code



- Swap Primitive
  - More efficient to dissect the data out of the cluster, operate on it standalone, then inject back into cluster.
  - Think of this is as a multi-diagram IPE

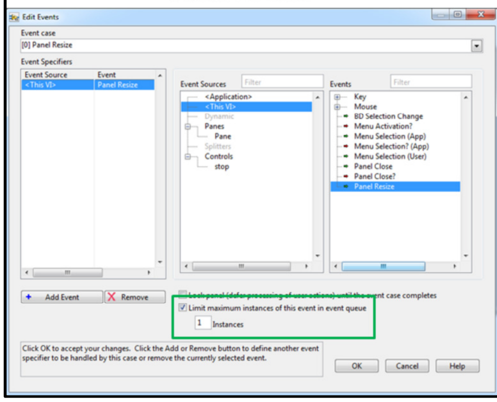


Write Fast Code

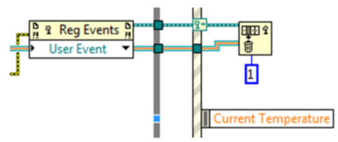


- Limit static events
- Filter excess dynamic events
- View > Event Inspector Window

Static Events

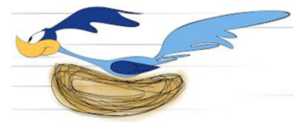


Dynamics Events



16

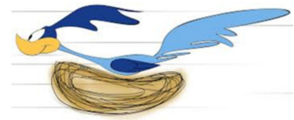
Write Fast Code



- Inline (beware: stale links, increased load time)
- De-normalize data structures and tasks
  - Line delimited string instead of array of strings, or track integer pointers into a single string
  - Too many LabVIEW tasks causes thrashing
- Minimize references, use data flow
- Down-sample when possible
- Buffer where appropriate
- Allocate modules to different Execution Systems*
- As a last resort, consider using different VI priorities

3/4/2018 17 SPACEX

Make sure all dependencies are reentrant to ensure execution does not synchronize across Execution Systems.



Write Fast Code

- Segment and prioritize data
  - Control parameters
  - Engineering review data
  - High speed dynamics data

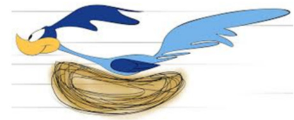
- Choose fast data storage
  - Spool large amounts of data to disk in **binary** format
  - Insert into SQL for fast random access

- Save readable formats like json for external interfaces

3/4/2018 18 SPACEX

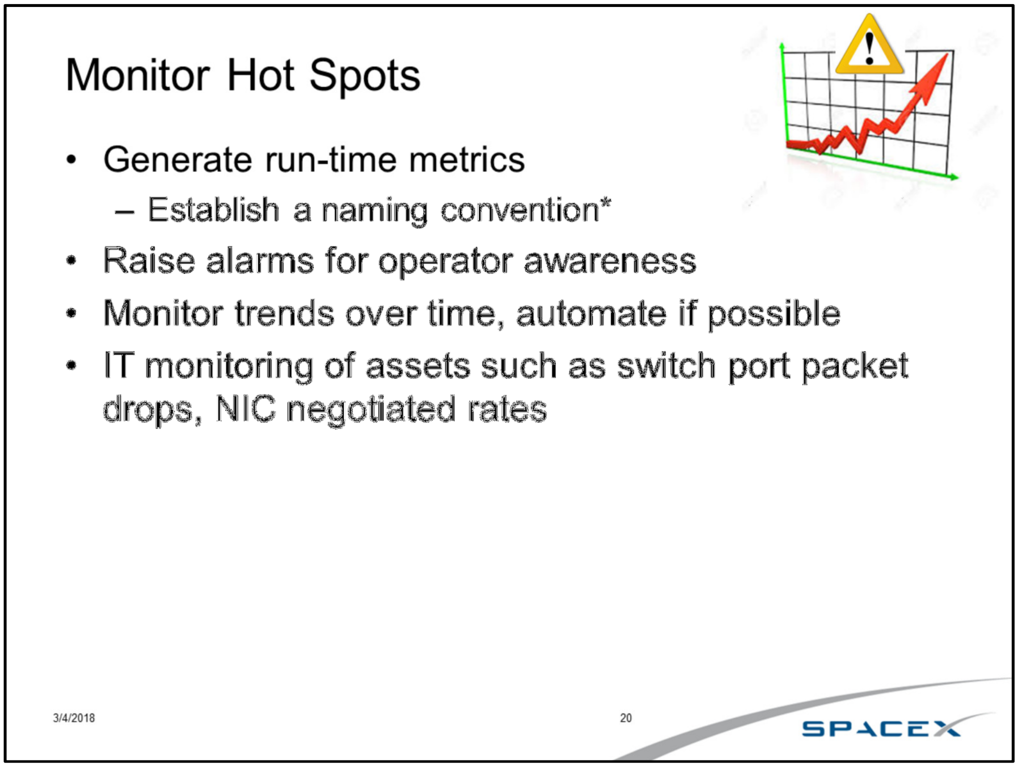
Not all data is created equal. Prioritize your data by keeping it separated from less important data.

Optimize Fast Code



- Tools >
  - Profile > Show Buffer Allocations, “Hide the Dots”
  - Profile > Performance and Memory
  - Profile > Buffer Allocations

- Desktop Execution Trace Toolkit
  - Shows memory allocations, collected references, and much more
  - Tends to be information overload for large apps
    - Use User-Defined Trace Events to bookend areas of interest



Naming convention might include things like
<source>.<measurement>_<value>_<valueunit>_<rateunit>

Generating Metrics

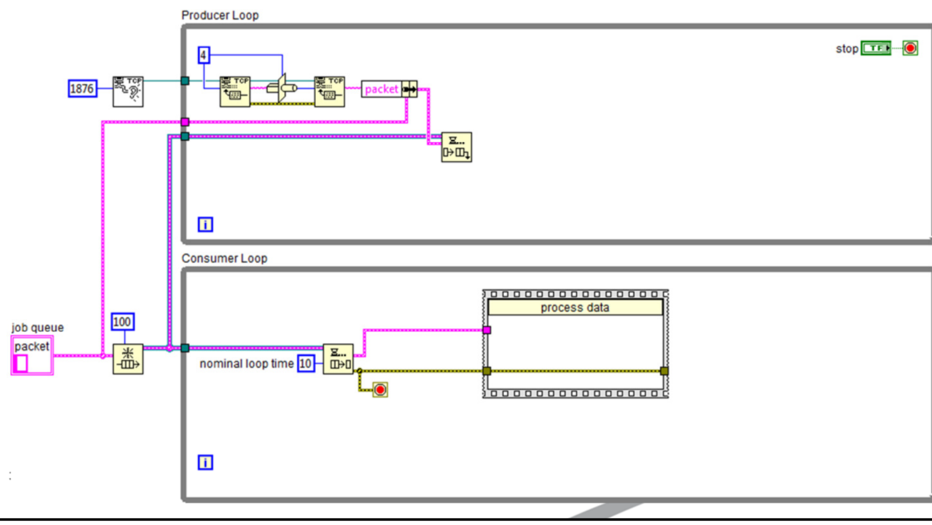
- Execution times
  - Use High Precision Timer
  - Report metrics at a slow rate using avg/max, per sec
  - Use absolute counts to report notable events
- Queue sizes
  - Max queue size, per sec
  - Overflows, total count

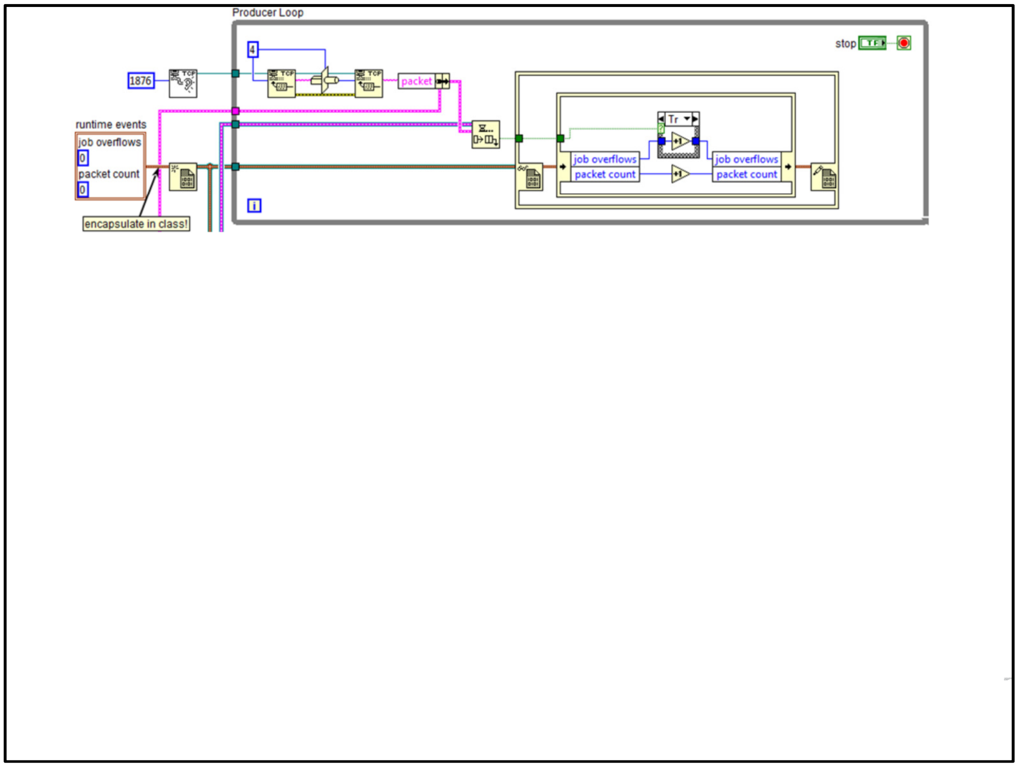
Generating Metrics

- Network Traffic
  - packets/bytes reads/writes per sec
  - NIC utilization
- CPU utilization for host and process
- Disk
  - Space remaining
  - Bytes read/write per sec
- Number of clients/references
  - Avoid RDP
- .NET Performance Counters are great

Generating Metrics

- Typical Producer/Consumer
- What metrics might we want to track?





(Animation is covering diagrams here)

Slide 24

NM1 Nate Moehring, 9/17/2017

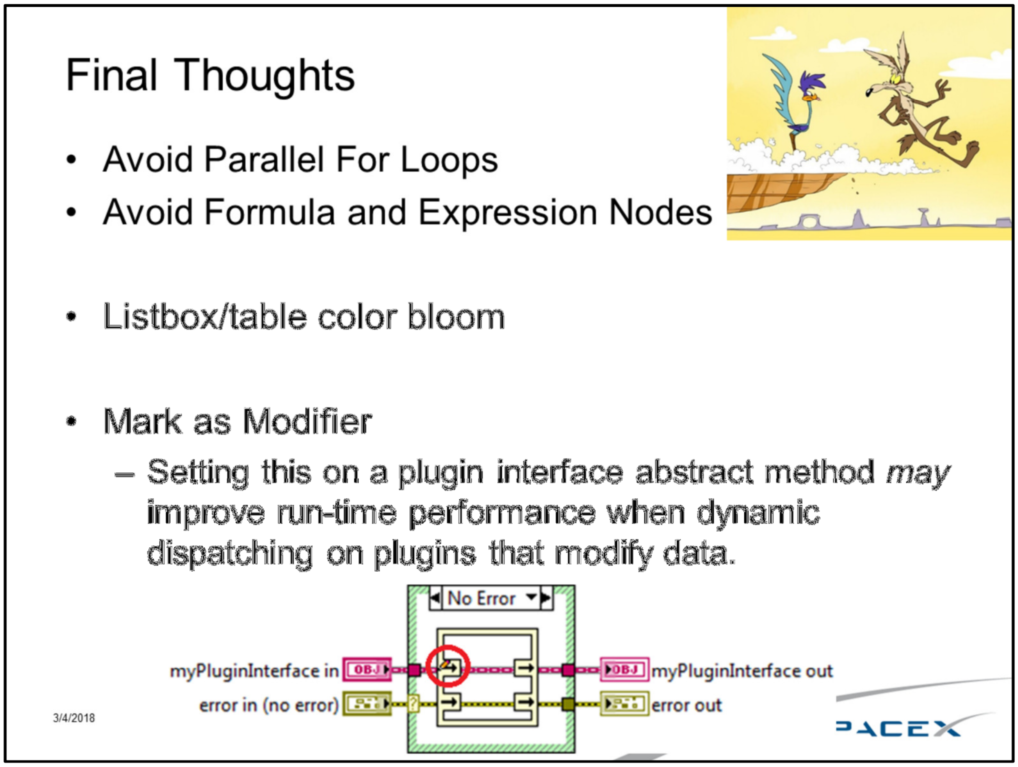
Test code

- “Nobody can predict the LabVIEW compiler” - NI
- If performance is a requirement, always verify it.

- VI Analyzer
- Smart Benchmarks (D. Nattinger, slides 14-17)
- Unit tests (performance tests?)
- End-to-End stress tests (automated?)

3/4/2018 25

SPACEX



I've experienced too many problems with run-time crashes using Parallel For Loops to make them desirable to you. I've also seen Parallel For Loops run slower than just sequentially running through all of the elements, even if it didn't crash

Color Bloom technique is the idea that when a user is scrolling through a table or a listbox, his eye is probably on ~ the middle of the table. Therefore, when coloring the rows or text of that table/listbox, color bloom is a technique coined by TurboPhil that describes coloring the middle row first, then alternating above and below rows, moving out towards the top and bottom.

So is LabVIEW a Tortoise or a Hare?