Network Streams

Everything you even wanted to know about…

Jeffrey Habets

Agenda

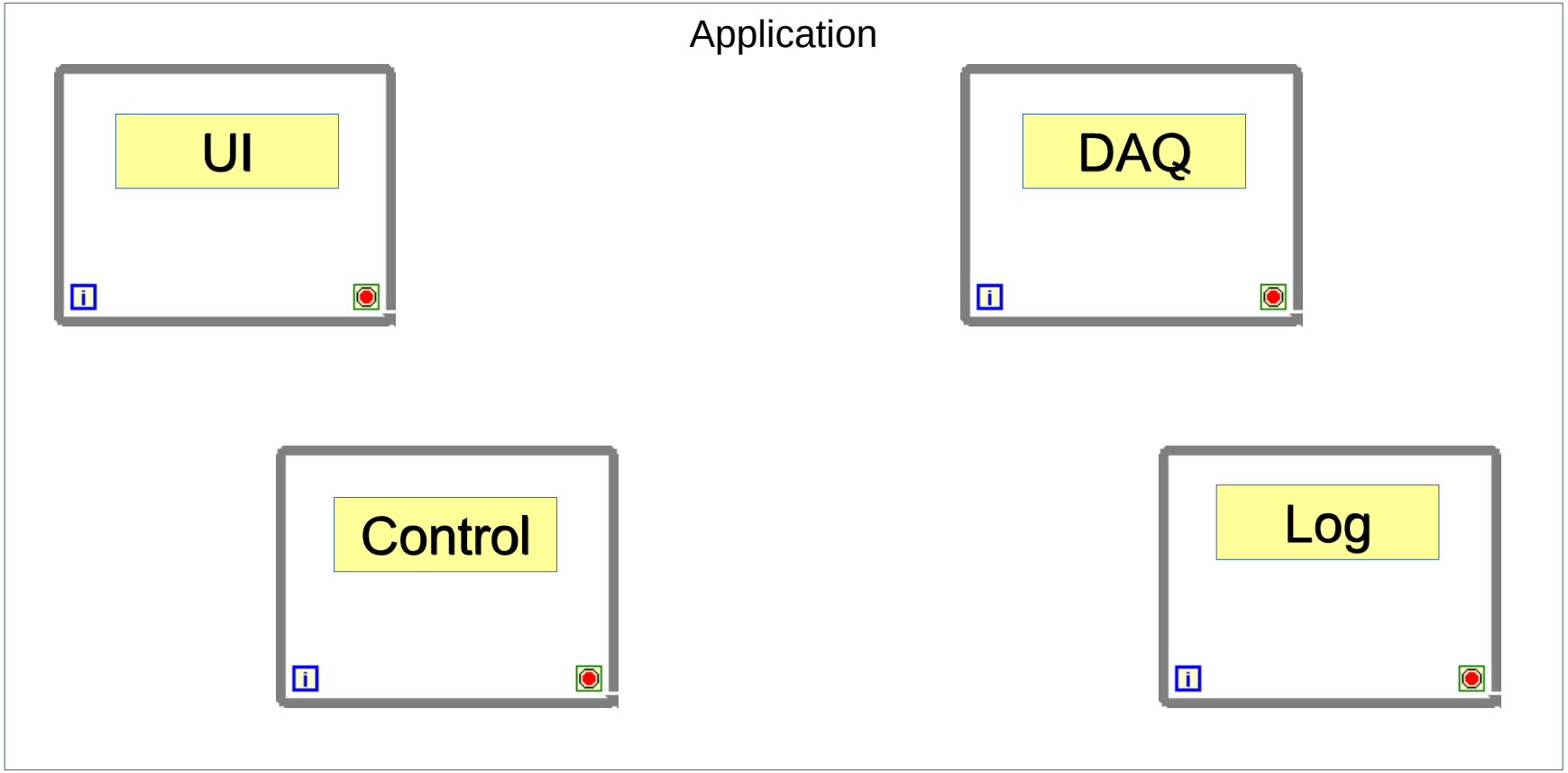
- Why this presentation
- Why use network streams
- What are network streams
- NS basics – Stream Creation and Operation
- Connection Management
- Demo
- Considerations
- Conclusion
- Other Interesting Network Stream stuff

Why This Presentation

- Did you know about network streams?
- When did you first hear about network streams?
- Me around 2012-ish, I think an NIWeek pres.
- They've been around since 2010
- They are not only suitable for streaming data

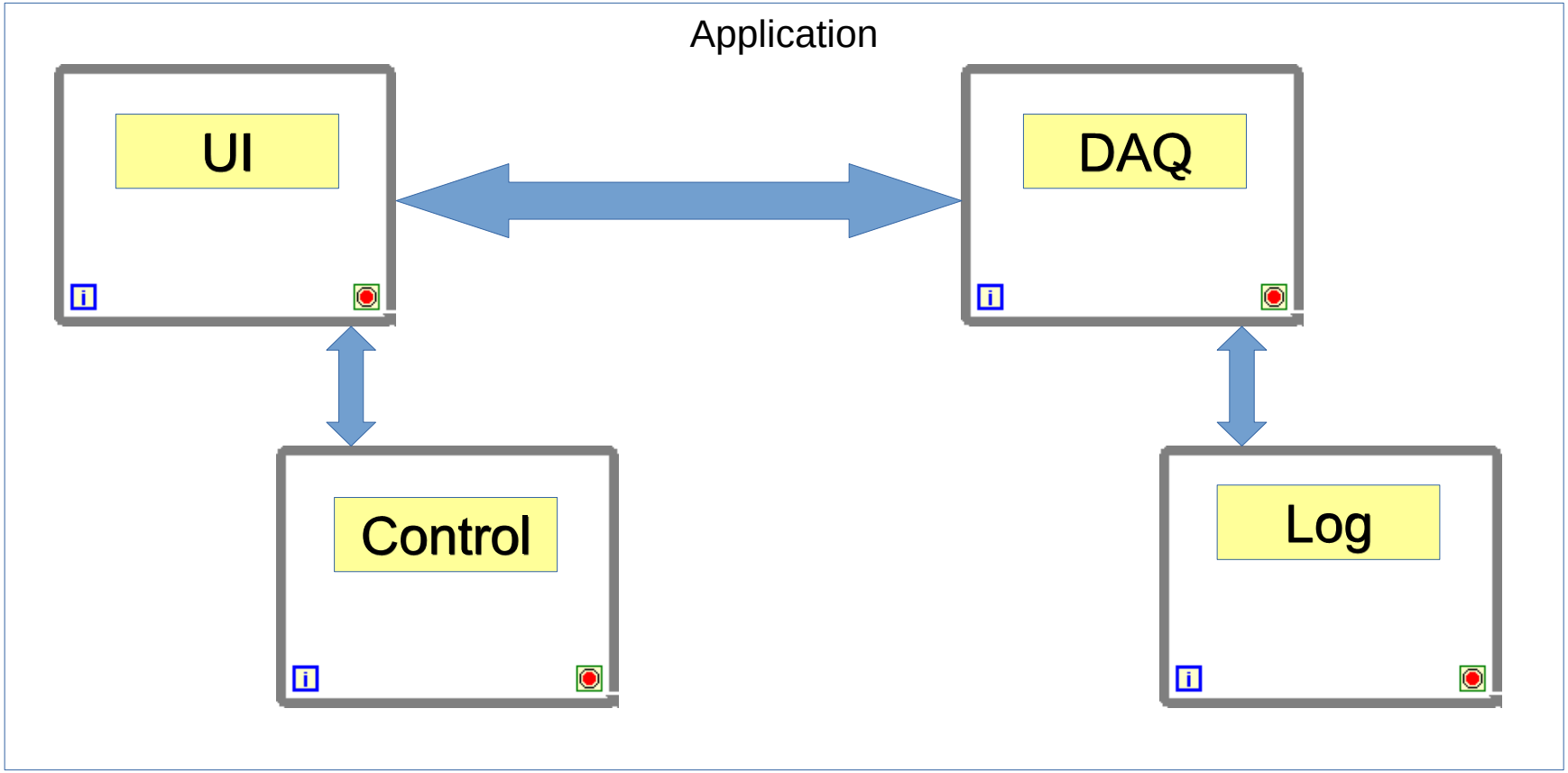
Why use Network Streams

Larger applications usually have more then one process executing concurrently



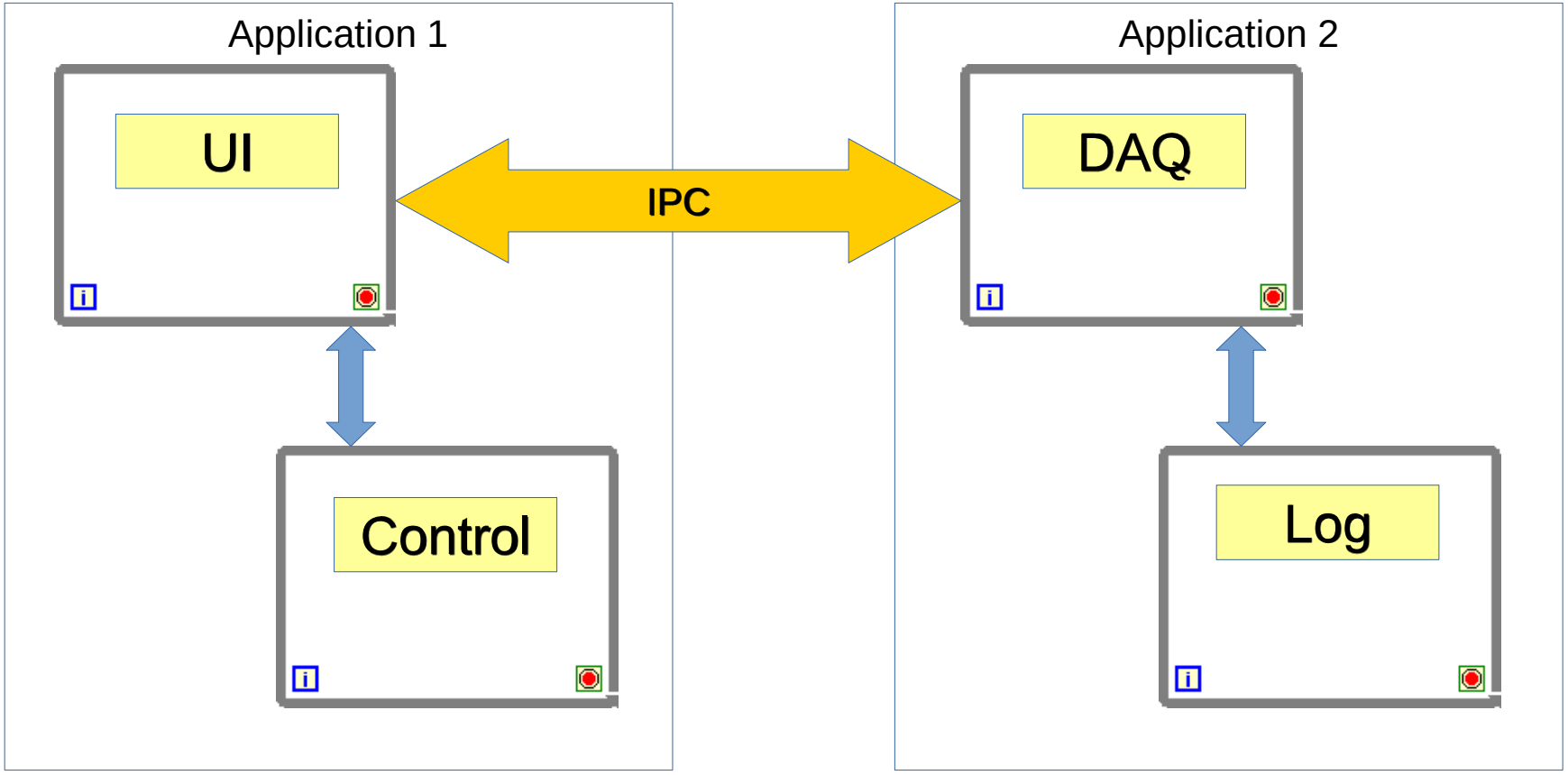
Why use Network Streams

Several ways of communicating between the various processes: Queues, User events, ~~Globals~~, ~~Locals~~, ...



Why use Network Streams

Inter-Process Communication



Why use Network Streams

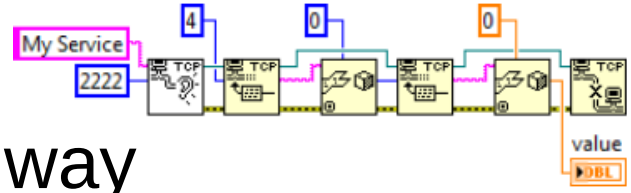
How Communicate Across App Instances or the Network?

- TCP & UDP
- Shared variables (3 flavours)
- VI Server
- Network Streams
- Web Services
- Peer-to-peer streaming
- Database
- Files
- ActiveX
- DataSocket
- EPICS
- STM
- AMC
- HTTP
- FTP
- …

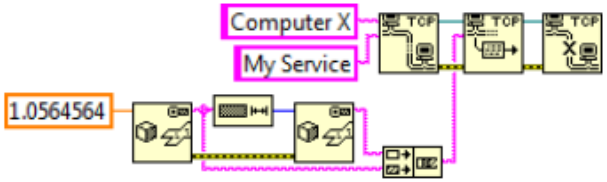
Why use Network Streams

…instead of e.g. TCP

- Connections Management
  - Ordering requirements
  - Disconnections due to unreliable networks
    - Handle possible in-flight data loss

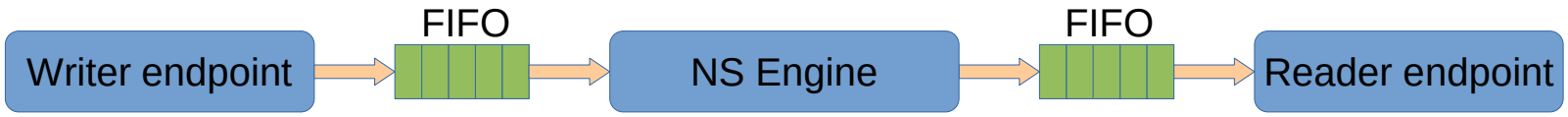


- Transferring data in a generic way
  - Serialize to and from binary stream
  - Develop own protocol for identifying data



What are Network Streams

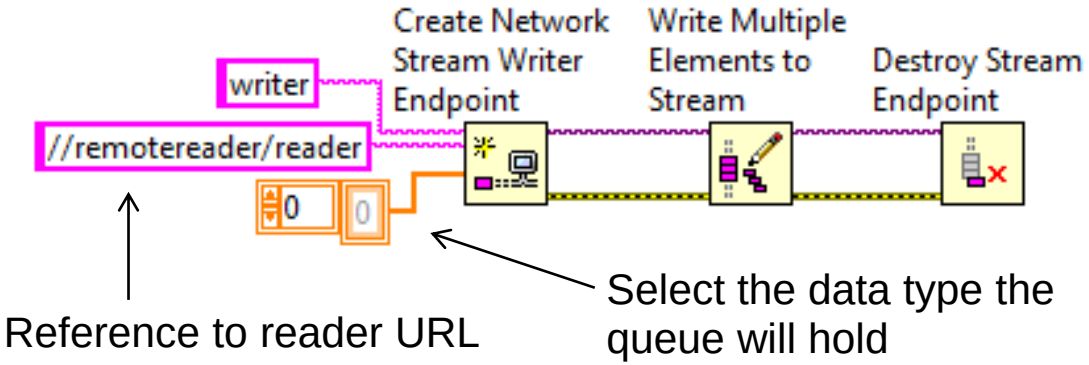
- A one-way, point-to-point lossless buffered communication → Or a networked queue



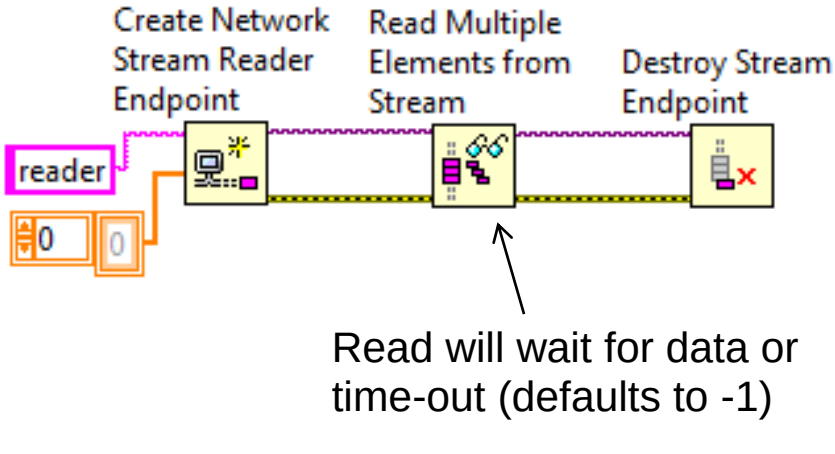
- Ideal for high throughput data streaming with through-puts comparable to TCP/IP
- Also usable for low latency command sending
- Takes care of some of the low-level TCP/IP complexity
- Direct support for most LabVIEW datatypes
  Not datatypes that include references or classes (exception for Vision Image ref.!)

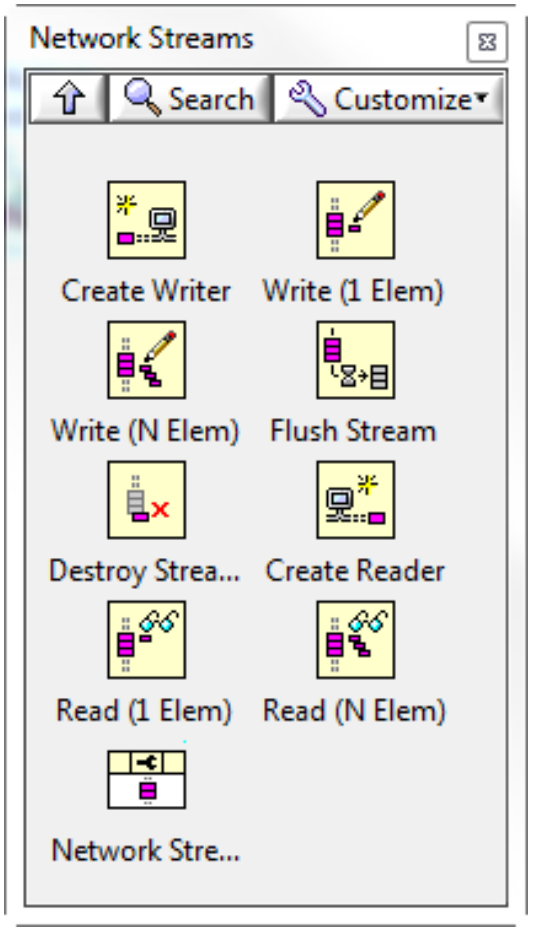
Crash course

Writing Elements to Stream

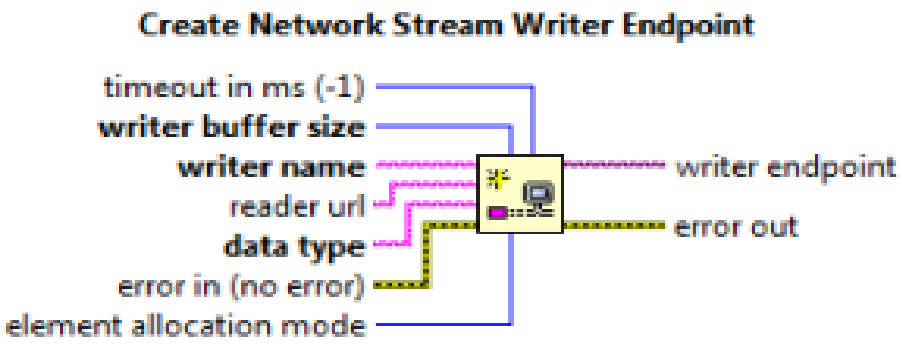


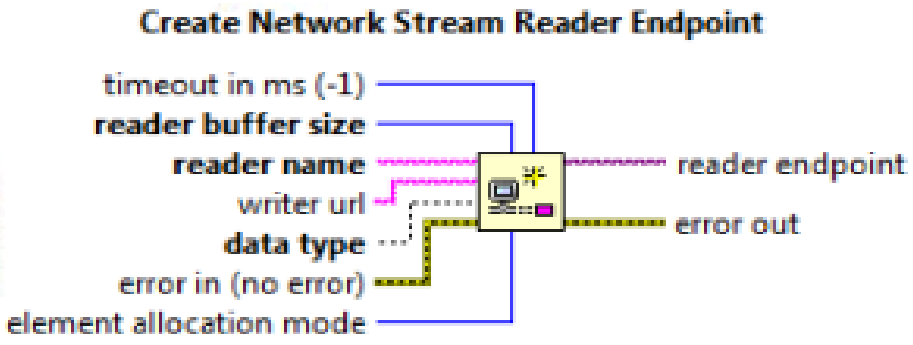
Reading Elements from Stream





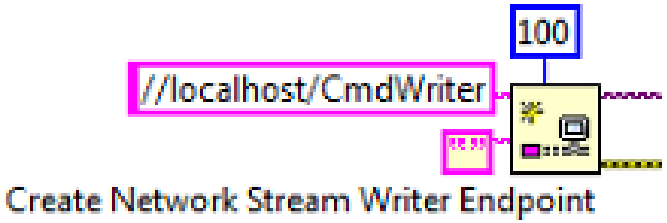
NS Basics – Stream creation



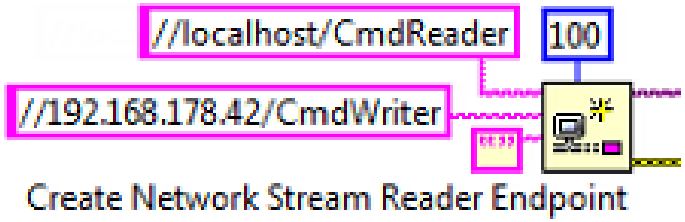


Mandatory inputs

- Writer / reader name
- Writer / reader buffer size
- Data type
- Reader / writer url on one side of the stream !

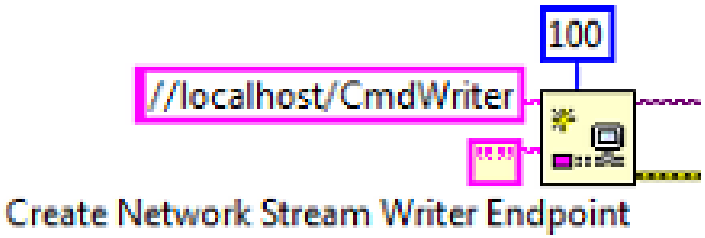


Code on writer side

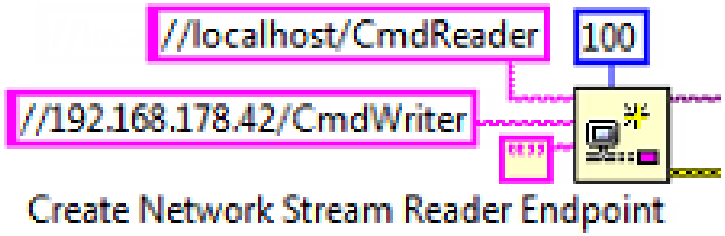


Code on reader side

NS Basics – Stream creation



Code on writer side



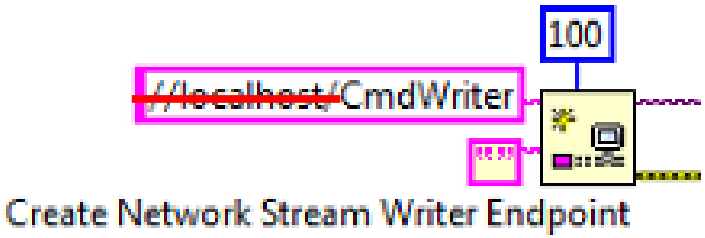
Code on reader side

ni.dex://host_name:context_name/endpoint_name

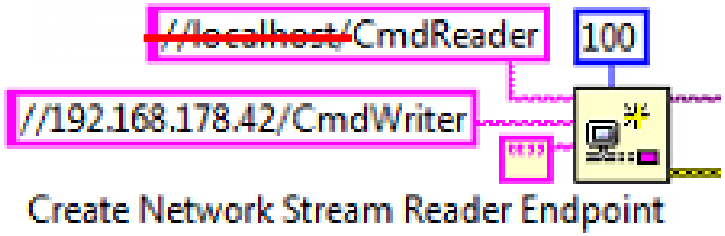
- Where **ni.dex** is the protocol, which is inferred by LabVIEW so you don't have to actually specify it
- **hostname** is the DNS name or IP address of the computer on which the endpoint you refer to resides
- The optional **context_name** identifies which application context the endpoint resides in
- The **endpoint_name** is the actual name of the endpoint and can also be build up as a hierarchical path of strings using forward slashes

//localhost/SubSystem 1/MeasDataStream
//localhost/SubSystem 1/Commands
//localhost/SubSystem 2/MeasDataStream
//localhost/SubSystem 2/Commands
etc..

NS Basics – Stream creation



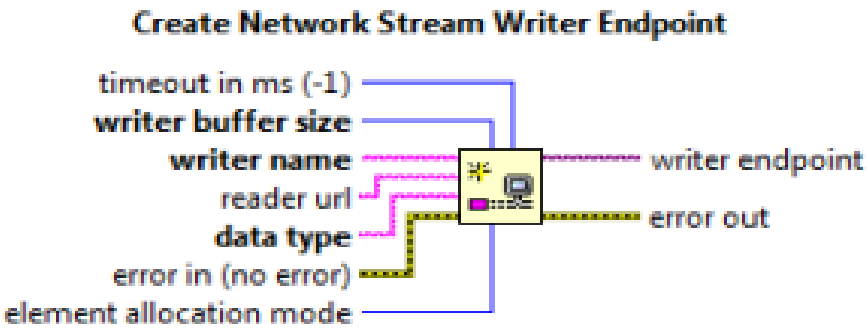
Code on writer side

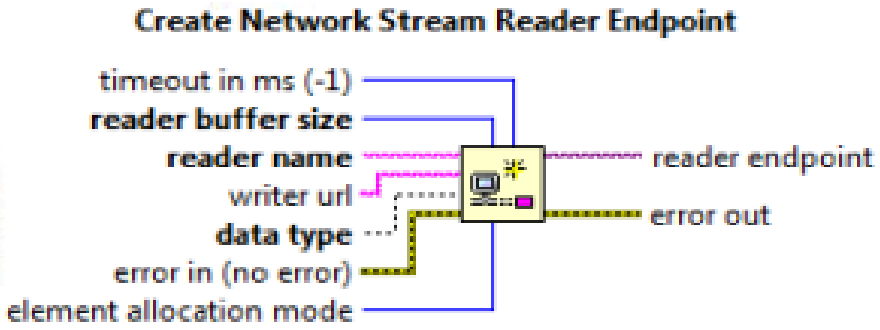


Code on reader side

- For specifying the writer name as well as the reader name inputs, it's sufficient to just specify the endpoint_name part. LabVIEW infers localhost, which is the only viable option. *(Exception: when using a context_name, you'll have to specify localhost as well)*
- At least one side of the stream should specify the reader/writer url parameter for the remote. This should always include a hostname or IP-address
- The endpoint that specifies the remote URL is called the active end-point and is in charge of handling connection loss
- I recommend not entering the remote on both sides because then it will be undetermined which endpoint is the active one and it can lead to unpredictable behaviour

NS Basics – Stream creation





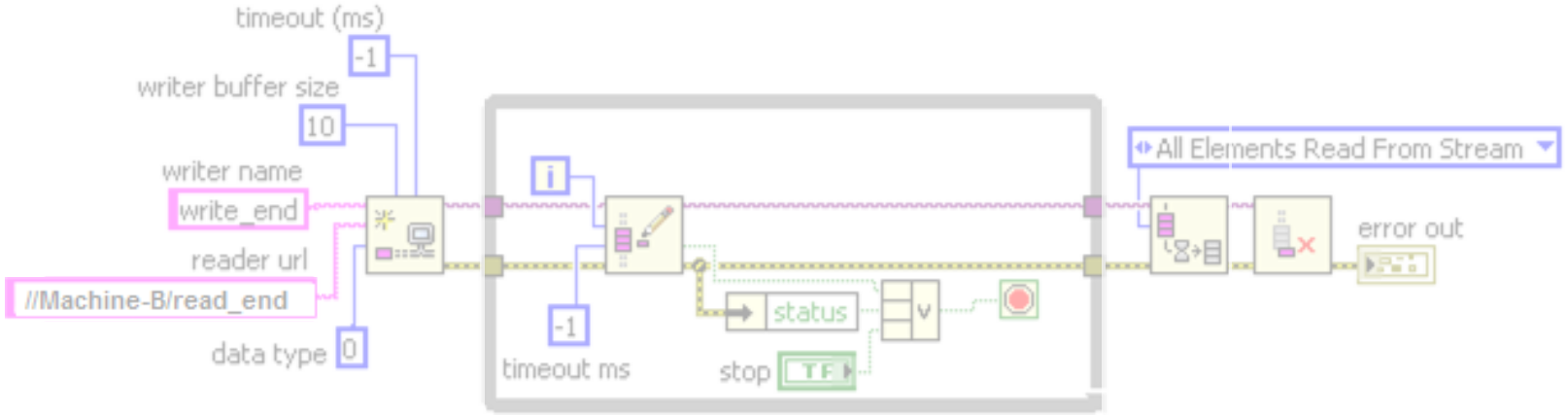
- Determine optimal **buffer size** through benchmarking
- The not mandatory **element allocation mode** determines if the buffer will be preallocated (as much as possible) on stream creation
- **Data type** to be used can be almost any LabVIEW data type except:
  - References or data types that contain them
  - LabVIEW classes or data types that contain them

The exception to the exception here is the Vision Image data type, which is available if you have the Vision Development Module installed

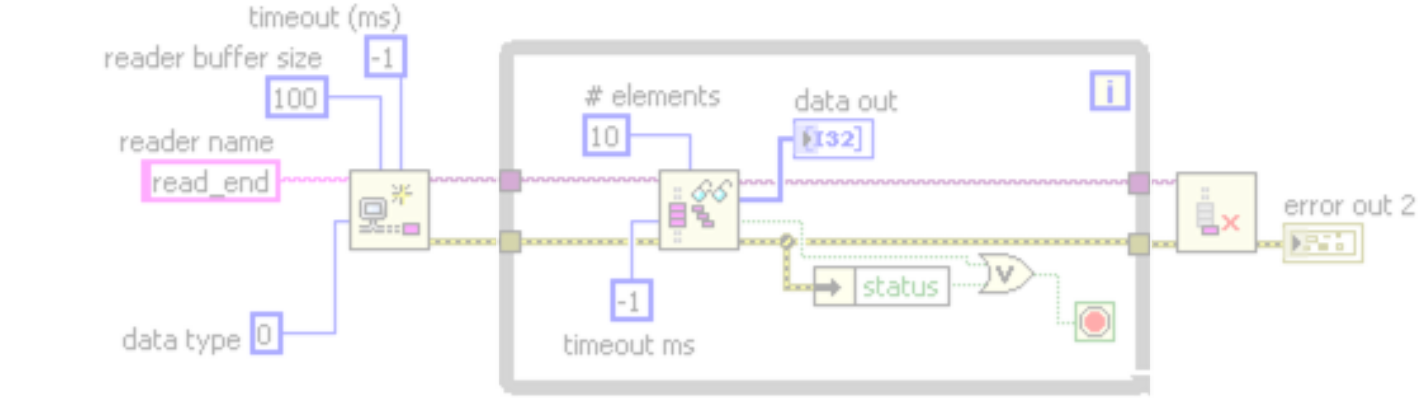
LabVIEW classes can still be send by flattening them to string

NS Basics – Stream creation

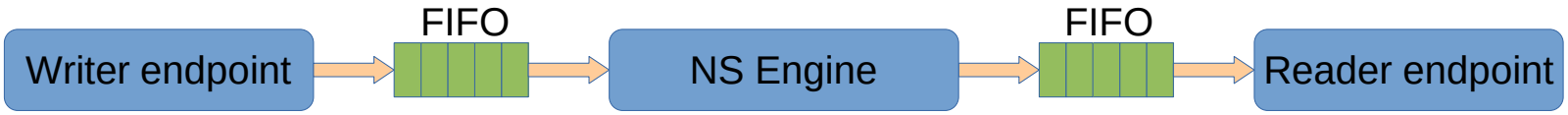
Machine A



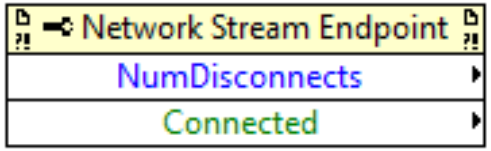
Machine B



Connection Management



- Performed automatically by the protocol in the background (will retry forever)
- Preserve lossless nature of data stream
- Will error if endpoints can’t resynchronize
- Active endpoint initiates reconnection upon a disconnect

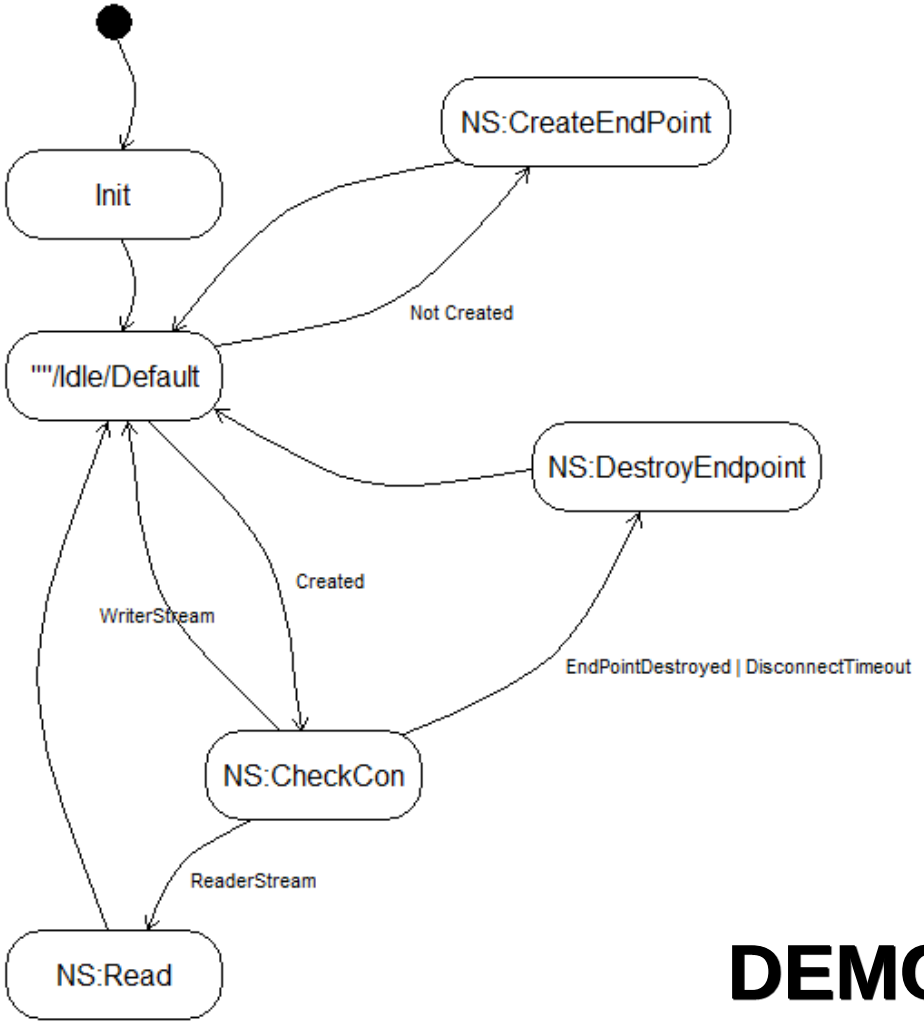


But there's a use-case not covered by this: unforeseen endpoint destruction...

Connection Management

- System that has one of the endpoints hangs or is reset (e.g. a cRIO where WD kicks in)
- The other endpoint has no way of knowing that the remote is gone
- To recover we need a connection watchdog
- Automatic recovery after deliberate destroy would also be nice

Connection Management



SuperNS
NSName : STRING RemoteEP_URL : STRING BufferSize : INT32 Element Allocation Mode : Stream Element Allocation Mode ConnLost Timeout : INT32 Writer : BOOL ReadDataEvent : STRING FlushAfterWrite : BOOL Thread : Cluster Endpoint : RefNum R/W Error : BOOL Flush Wait Condition : Flush And Wait Empty Condition DataReadEvent : StreamDataType NS Created : BOOL BG ConnCheck/Read period : INT32
+Create +Destroy +WriteToStream +GetReadDataEventReg +GetStreamStatus +OpenCloseThreadWin -StartThread -StopThread -ConnRecoveryThread

DEMO !

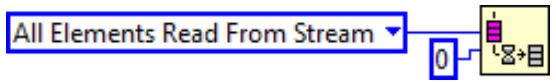
Network streams

Considerations

- For RT target, ensure Network Streams Engine is installed
- Numerics, booleans and arrays thereof transfer fastest
- Benchmark to ensure proper buffer size
- Design an API and architecture around Network Streams that is tailored to your application rather than just copy-pasting from examples.

Conclusion

- Network streams is an easy to use 1-to-1 communication, basically if you know queues, you know network streams.
- Can be optimized for high throughput (measurement data) or low latency (commands)
- The only connection maintenance you'll need to do is cover the case where one of the endpoints is unexpectedly destroyed (e.g. system crash)
- Guaranteed no data loss



- Good practice: encapsulate them in your own API tailored to your application

Read more

- Lossless Communication with Network Streams: Components, Architecture, and Performance

  http://www.ni.com/white-paper/12267/en/
- Actor Framework - Linked Network Actor

  https://decibel.ni.com/content/docs/DOC-24051
- LabVIEW help → Search for 'Network Streams‘
- LabVIEW 2013 and higher RT example projects use NS
- Recommended Firewall Settings When Using Network Streams

  http://digital.ni.com/public.nsf/allkb/BB3A4E062CE21A098625775F00710497
- Using the Right Networking Protocol

  http://www.ni.com/white-paper/12079/en
- VI Threads blog

  http://www.vi-tech.nl/en/blog
- NI GOOP Development Suite

  http://sine.ni.com/nips/cds/view/p/lang/nl/nid/209038

http://nl.linkedin.com/in/vitech @JeffreyHabets