LabVIEW Developer Days

Build Code. Form Communities. Gain Confidence.

NATIONAL INSTRUMENTS™

Data Communication for Scalable Systems

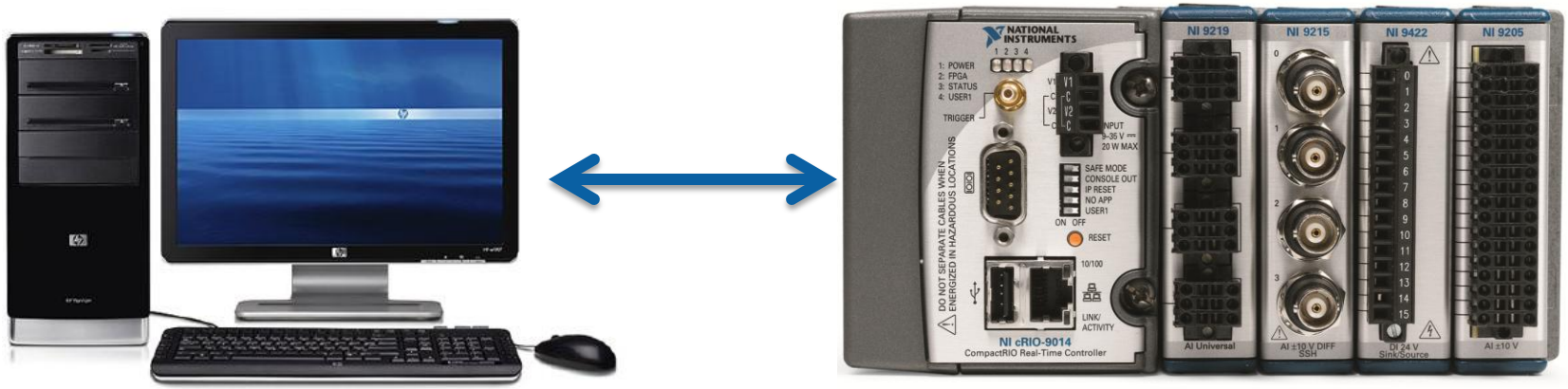
Matt Pollock: Sr Systems Engineer

NATIONAL INSTRUMENTS

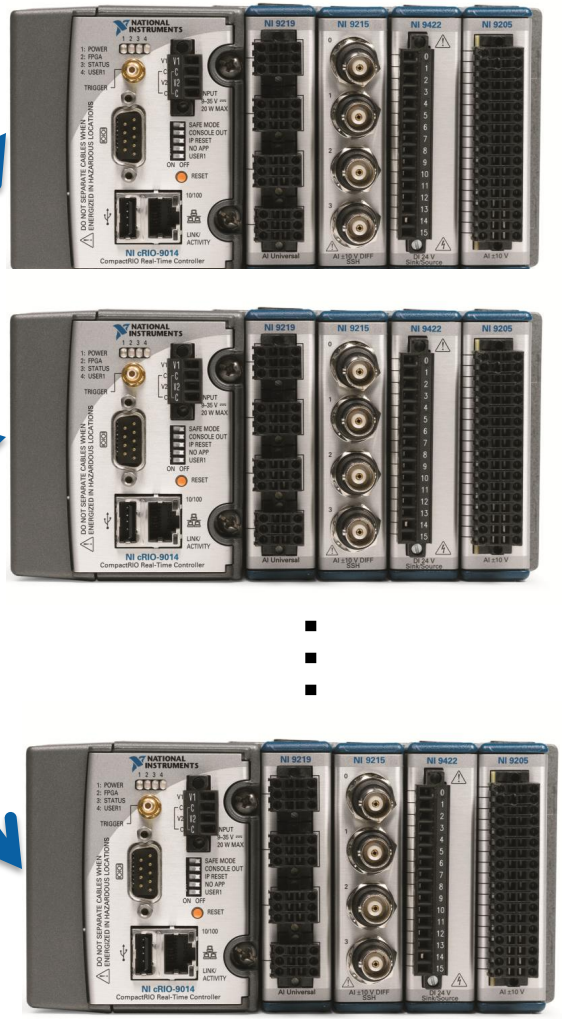
Session Goals

- Discuss common architectures to address scalability in embedded systems
- We will not discuss the basics of each type of data communication mechanism (what is TCP, what is a queue, etc)
  - For more information on the basics, see the LabVIEW for CompactRIO Developer's Guide (ni.com/compactriodevguide)
- We will discuss the tradeoffs of different mechanisms – when to use one over another

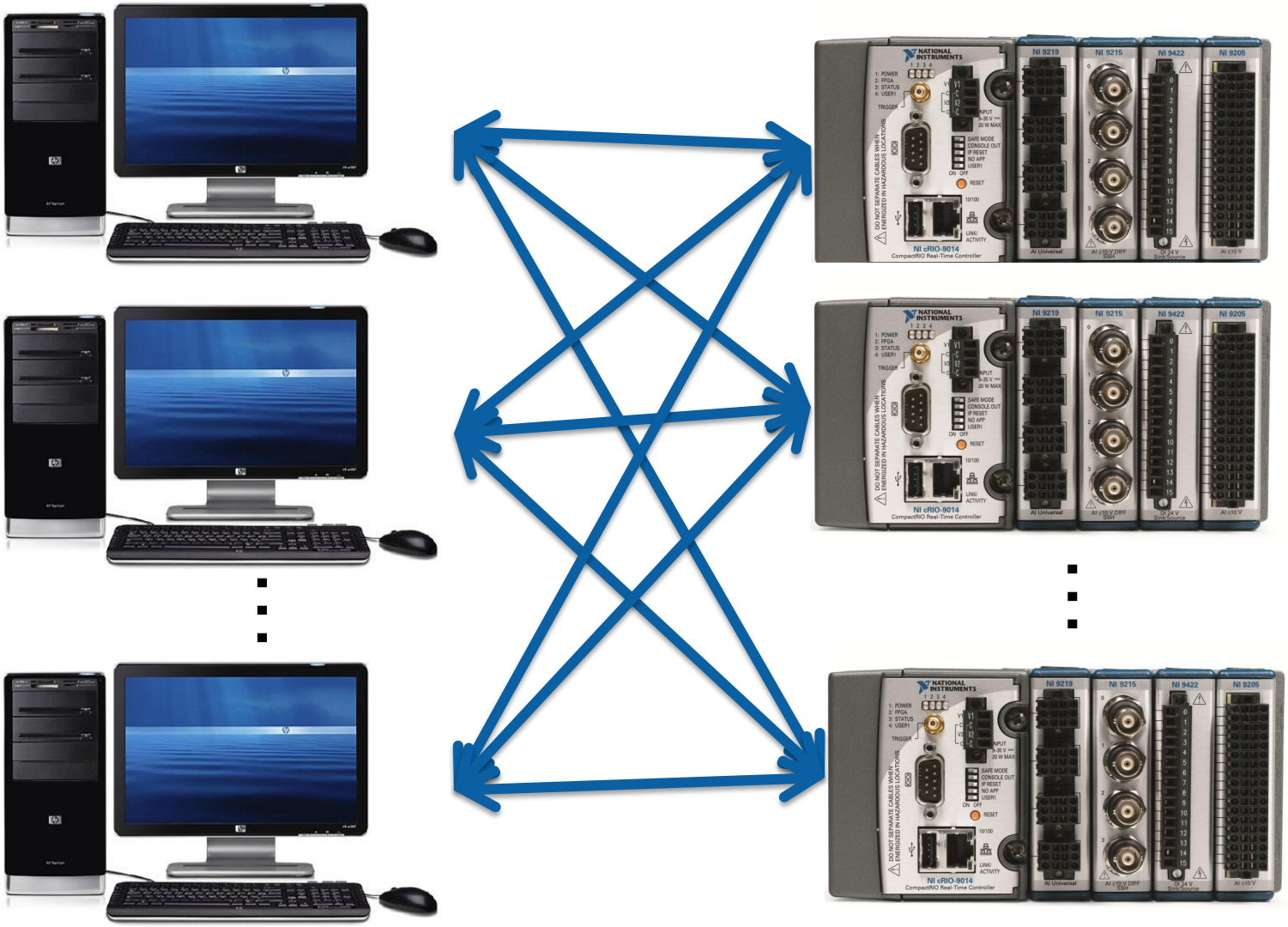
Why is Scalability Important?



Why is Scalability Important?



Why is Scalability Important?



Goal of Scalability

Minimize added work and risk of adding ‘one more’

Scalability Strategies

- Use flexible communication schemes that minimize foreknowledge on the systems
  - No hard coded addresses if possible
- Use communication pipes that are optimized for the type of communication necessary
  - You probably need more than one.
  - If all you have is a hammer…
- Architect the system such that communications performance has minimal impact on critical operations

Available Data Communication Mechanisms

TCP and UDP
Network Streams
Shared Variables x 3
DMAs
Actor Framework
Peer-to-Peer Streaming
Queues
Dynamic Events
Functional Global Variables
RT FIFOs
Datasocket
Local Variables
VI Server

Target-scoped FIFOs
Notifier
Simple TCP/IP Messaging (STM)
AMC
HTTP
FTP
DVR
Web Services
ZMQ
AMQP
DDS
….
Many more

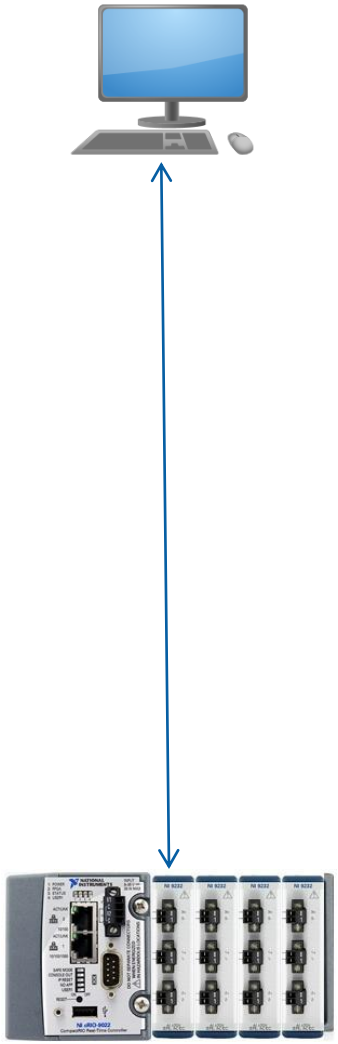
Data Communication Types

Tag *Current value data*

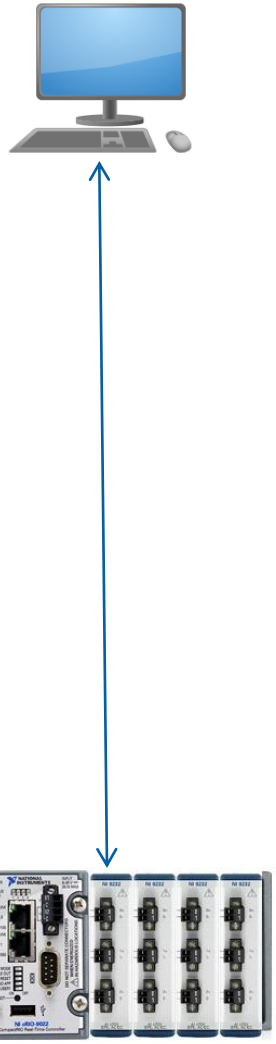
Messages/Commands *Intermittent data, reliable delivery*

Stream *Continuous acquisition, high throughput*

Single-Node Architecture



Single-Node Architecture



Great for 1:1
Focus on scalability: low to none
Functionality is usually first concern

Applications:
Prototypes
Simple control and monitoring

Potential concerns:
Easy replacement of either client or node

Headless Control System

Arbitrary computer
Zero-install requirement
Any operating system
Need to quickly attach to a maintenance interface

HTTP

Hosted web page
Web Services

NATIONAL INSTRUMENTS

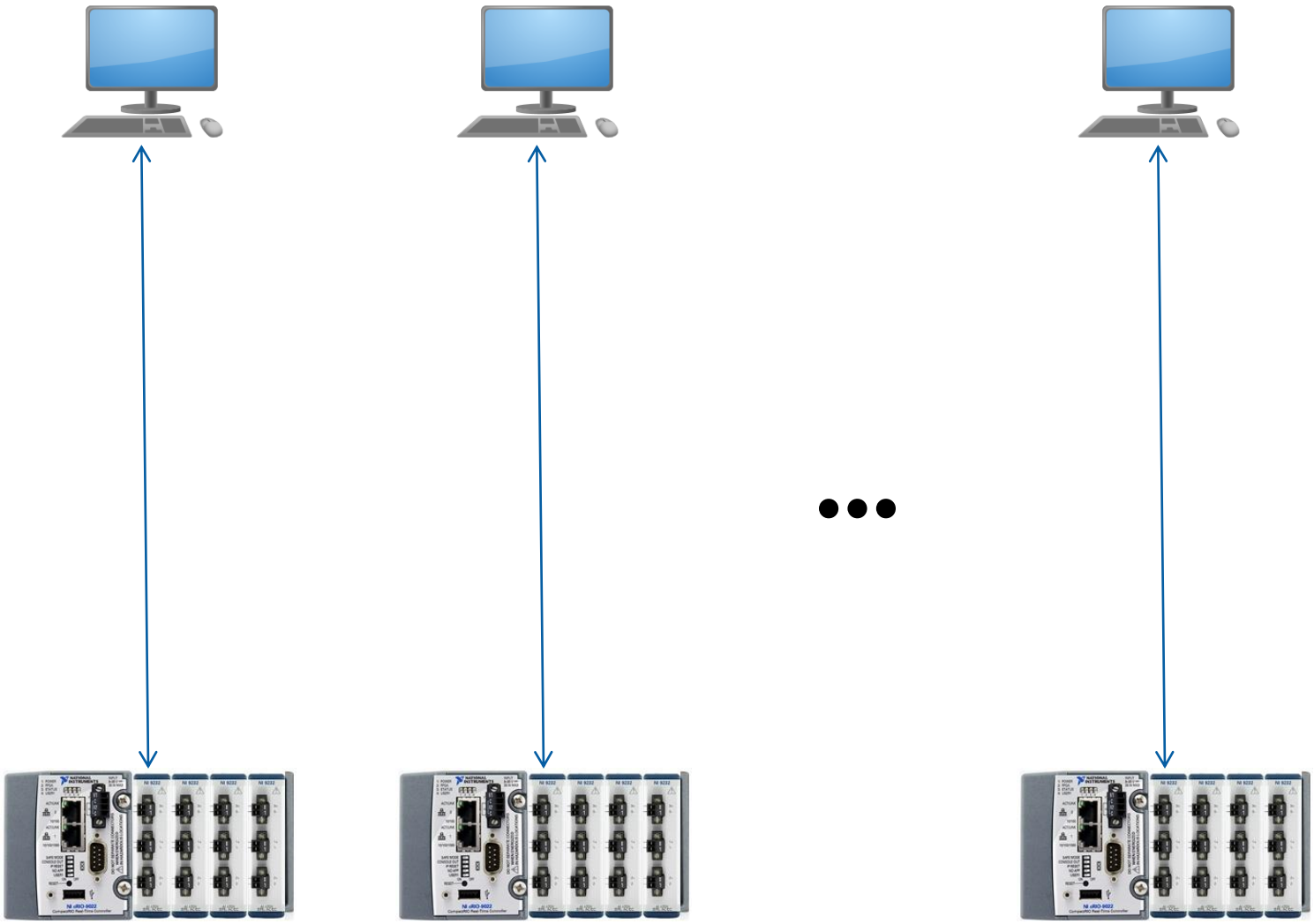
Embedded Datalogger

LV Client
Stream endpoint known in advance
Connects to one node at a time

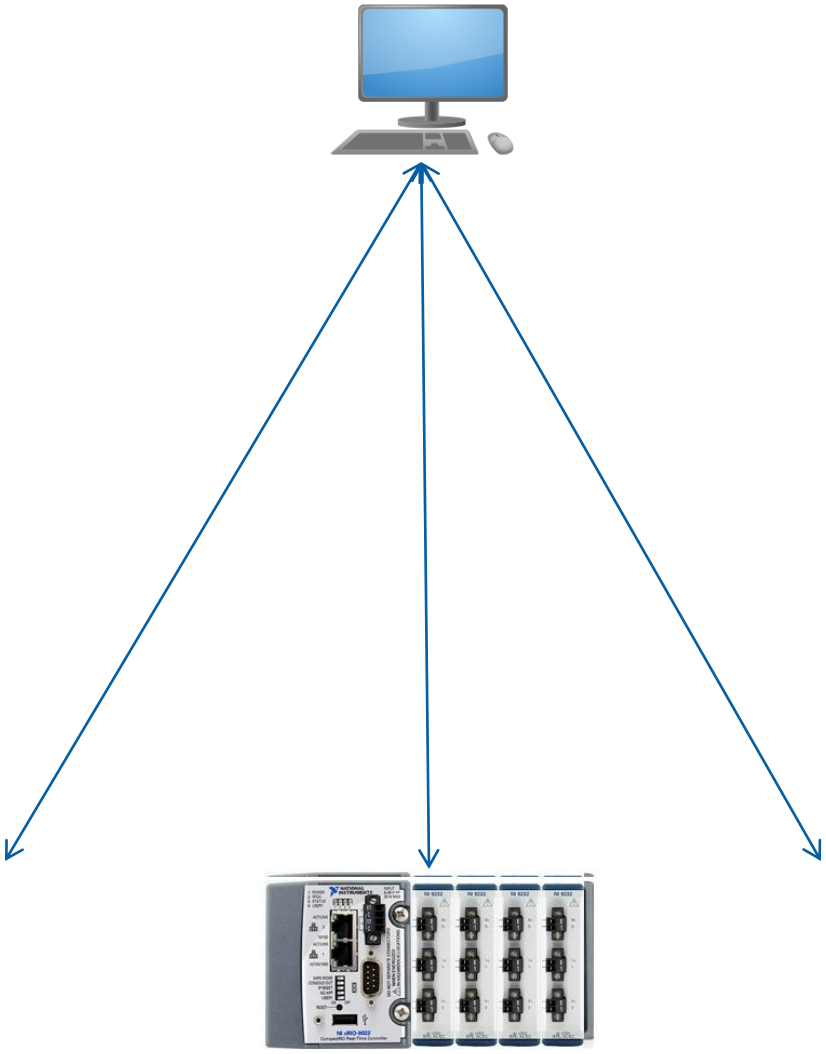
Shared variables for status
Network stream to node for commands
Network stream to client for waveform monitoring
FTP/WebDAV for file transfer

Shared Variable Engine hosts status tags
Streams connect to one HMI at a time

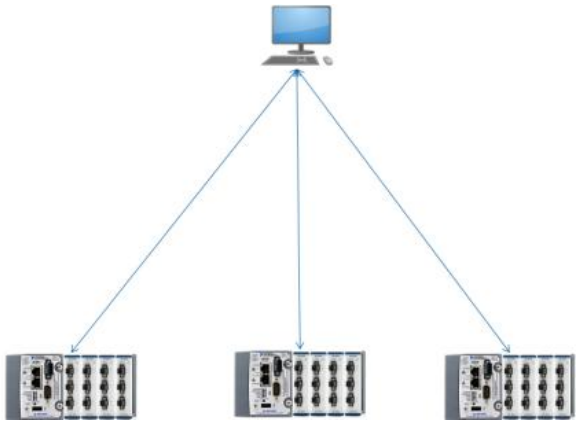
Replication



Multi-Node Architecture



Multi-Node Architecture



Scalability focus: Client code

Applications:
Multiple instances of identical nodes to one client
Low (often fixed) number of nodes

How hard is it to add one more?
- Node code: Little to no effort
- Client code: Medium effort

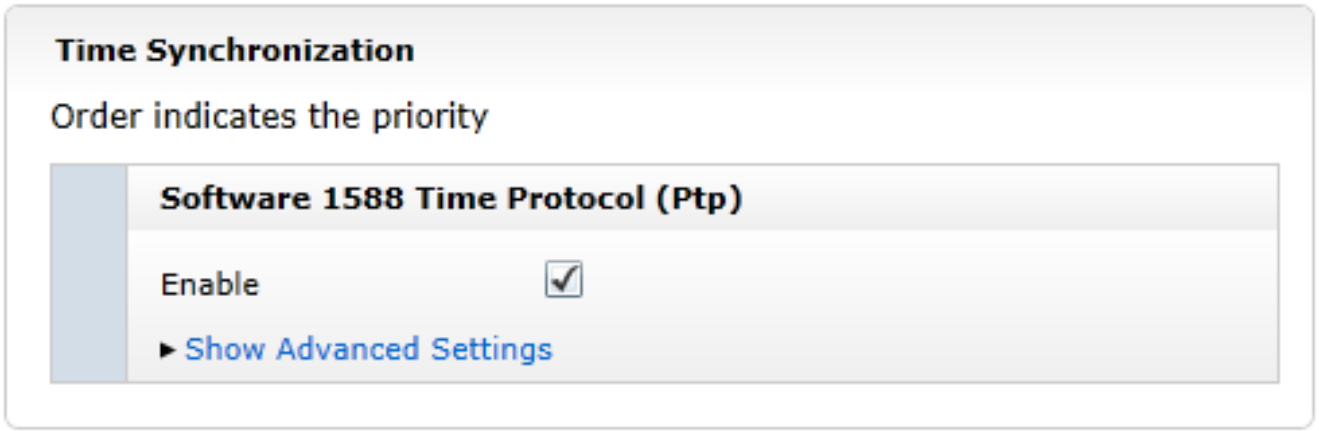
Potential concerns:
- Synchronization of timestamps
- Repurposing HMI screens
- Unique Network Stream endpoint names

Timestamp Synchronization

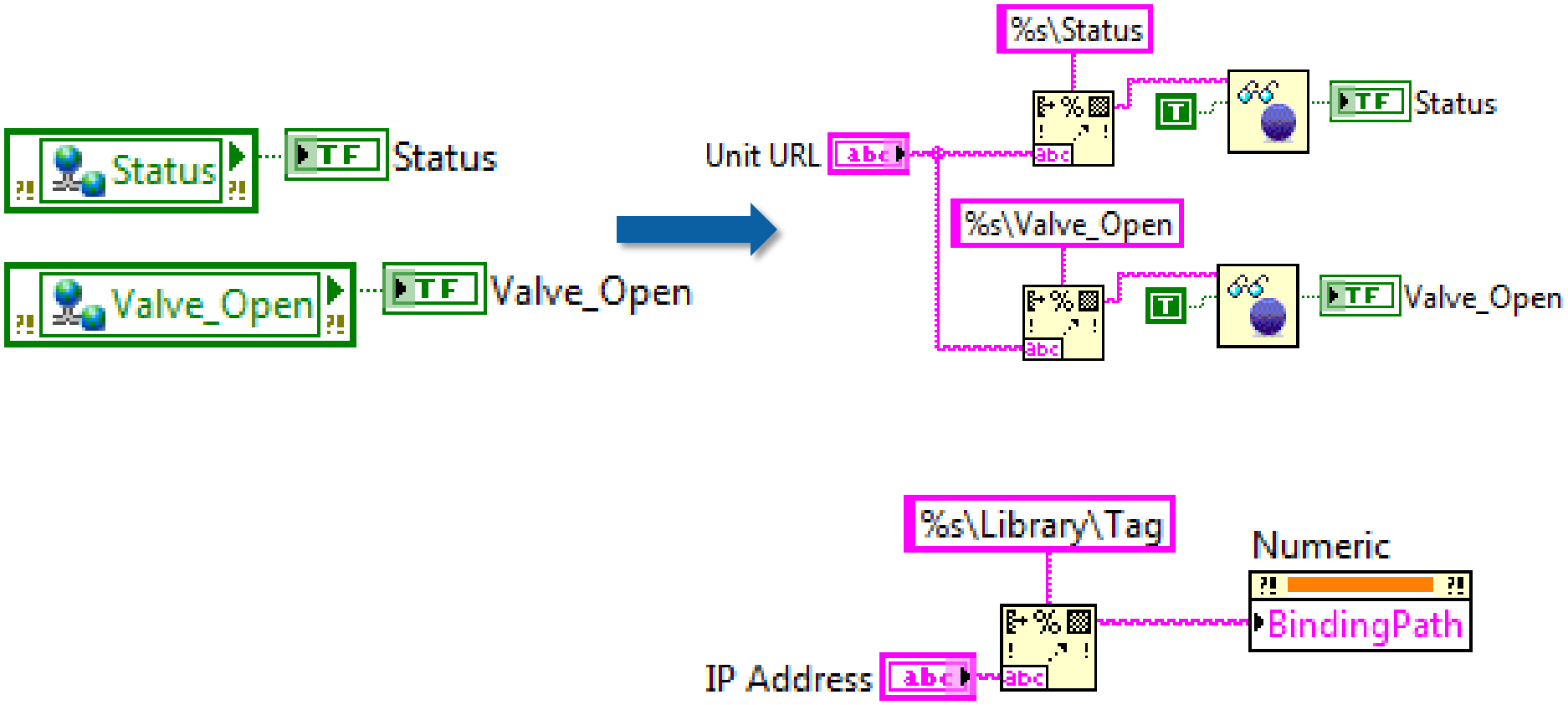


Timestamp Synchronization

- Software: IEEE-1588 Precision Time Protocol
- Installable component for NI RT targets via MAX
- Automatic, no fixed server need be configured
- Nodes on network auto-negotiate, set system time
- For best results, have one node on network synched to an accurate time source (e.g., GPS, hardware 1588, etc)



Reuse HMI Screens

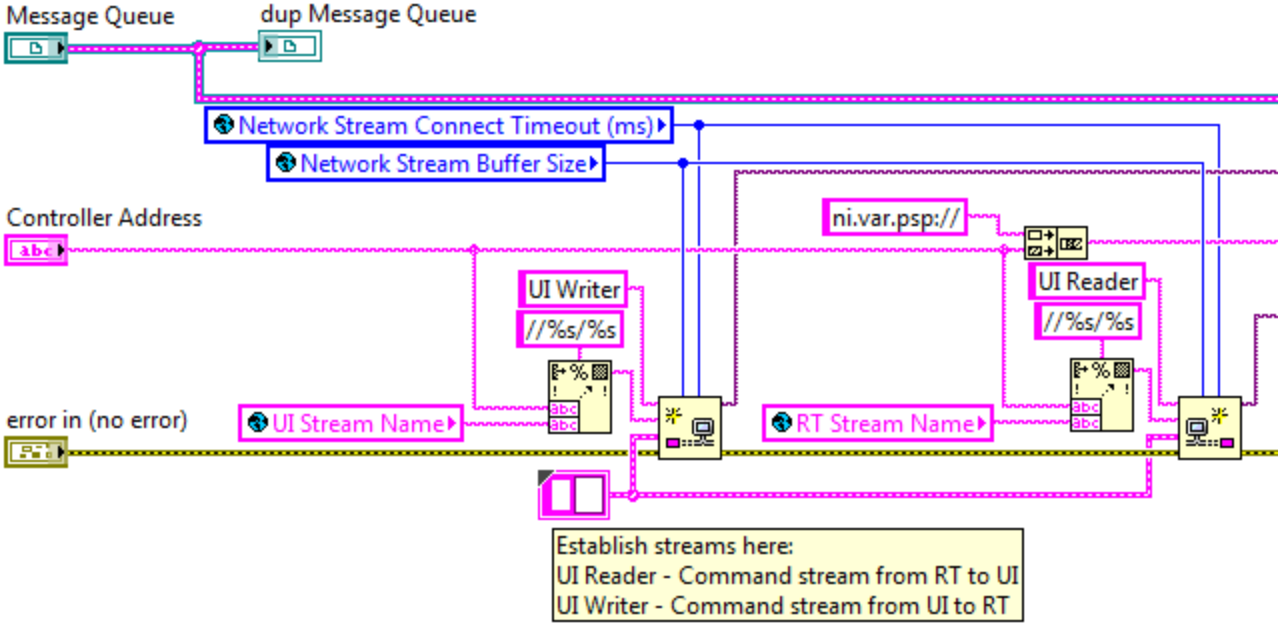


Network Stream Connection Bootstrapping

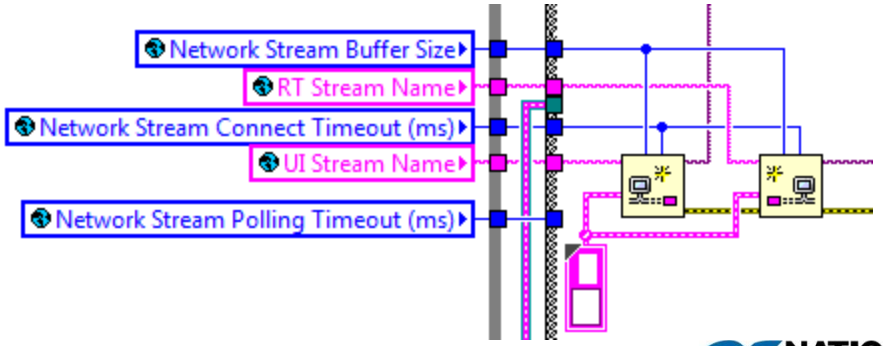
- Network streams are (by design) a 1:1 communication pipe
  - Exactly 1 writer and 1 reader endpoint per stream
  - Endpoint URLs cannot be reused amongst several streams
  - Easiest solution is to hard-code a URL for each node
- To scale, another layer is necessary

Single-Node Network Stream Connection

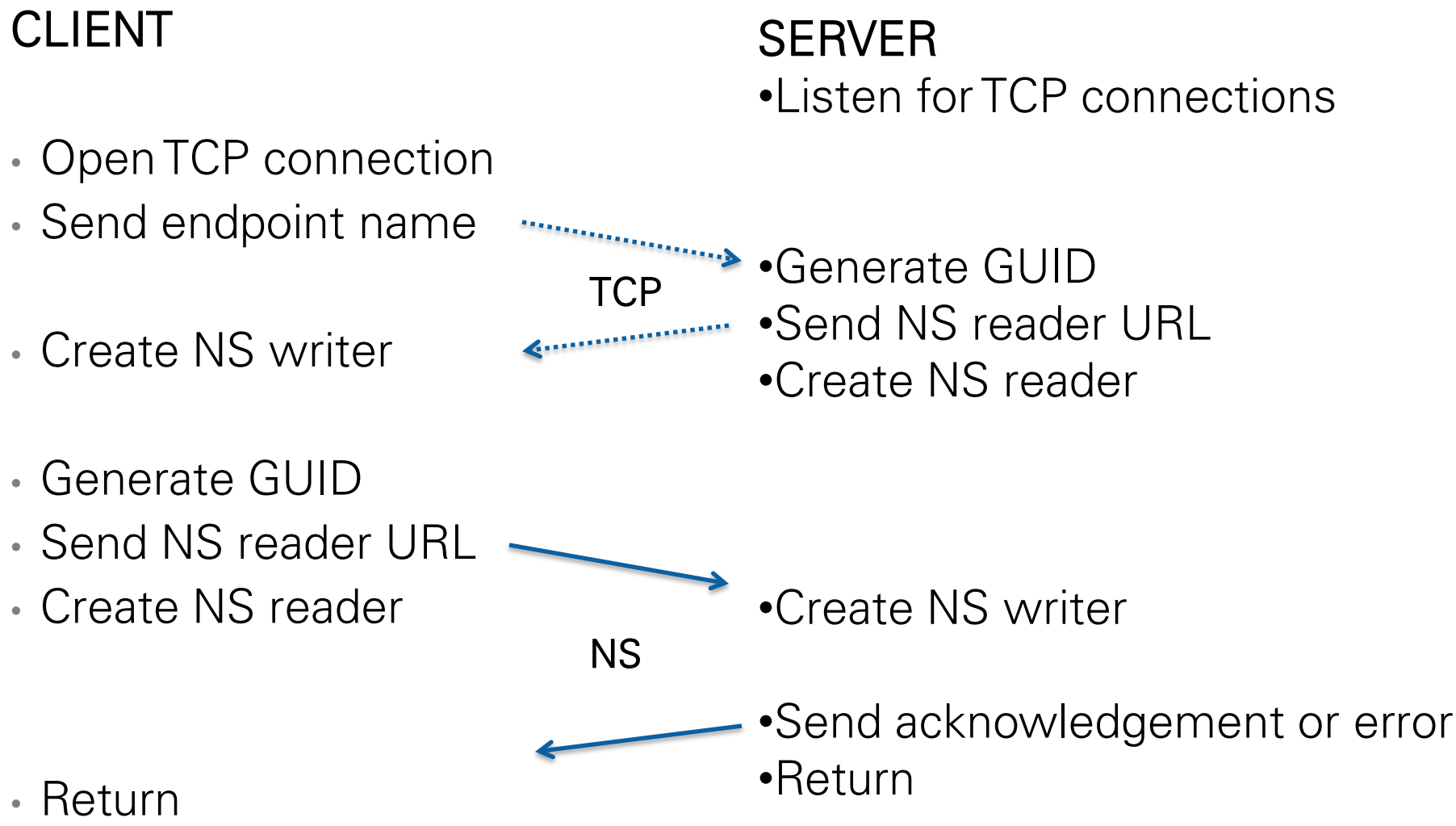
UI:



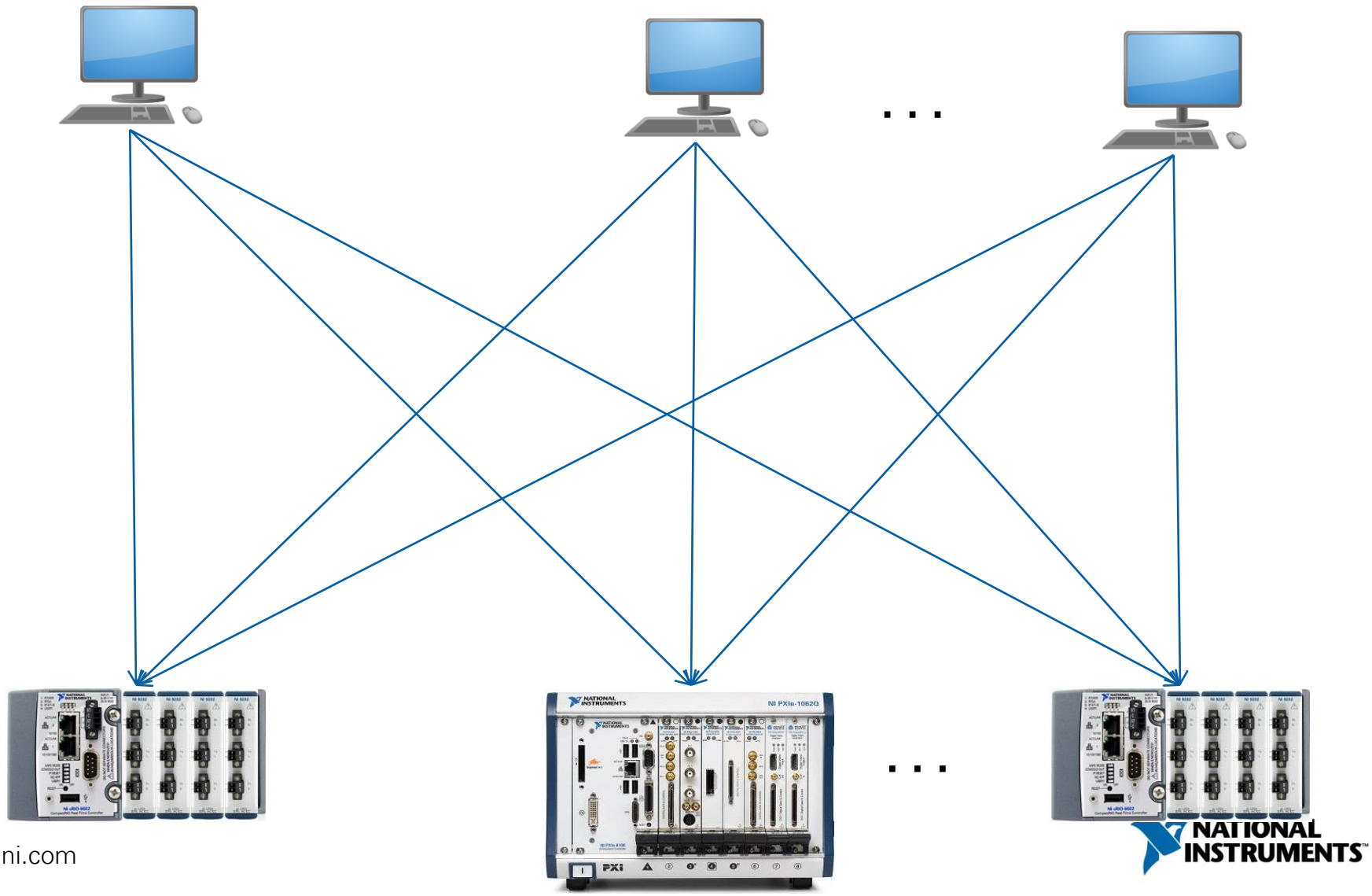
RT:



Network Stream Connection Bootstrapping

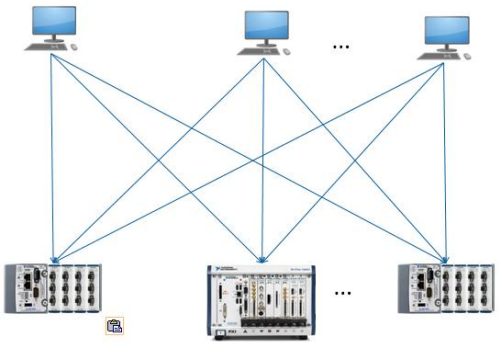


Serverless Architecture



NATIONAL INSTRUMENTS

Serverless Architecture



Scalability focus: Node code and client code

Applications:
Multiple monitoring stations for complex system
Single station in control of any given node

How hard is it to add one more?
- Node code: Medium effort
- Client code: Medium effort

Potential new concerns:
- Load on nodes due to client access

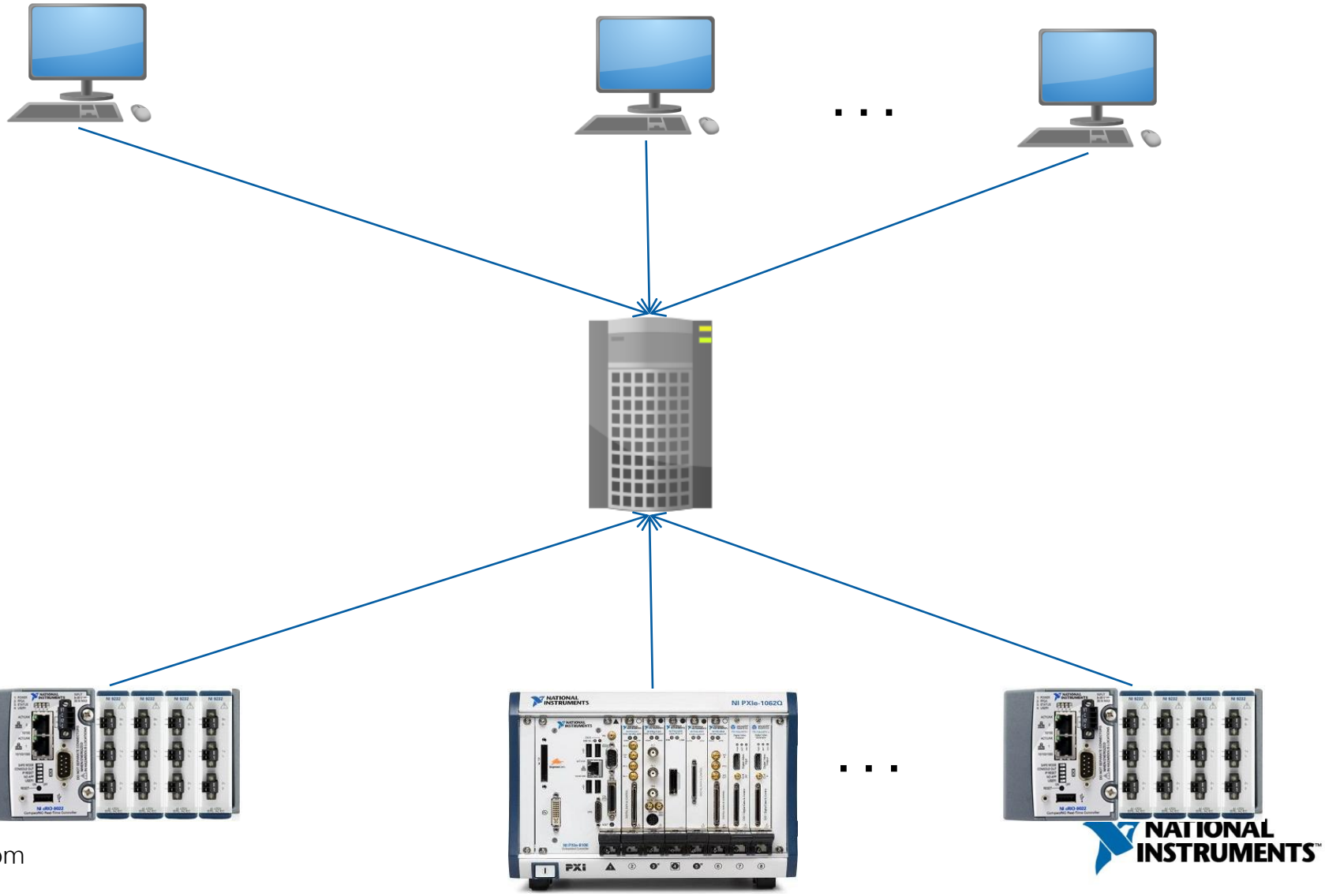
The Sweet Spot for Shared Variables

- Great for current value (tag) monitoring
- Originally intended for the SCADA use case
- Latest value (no buffering), scalar data
- Libraries of 500 variables or less
  - Break up larger libraries into multiple smaller ones
- SCADA data rates (<100 Hz)
- Sparse updates
  - Not everything updated at once every time
- Sparse subscriptions
  - Not every client looking at every variable every time

Scalability Red Flags with Shared Variables

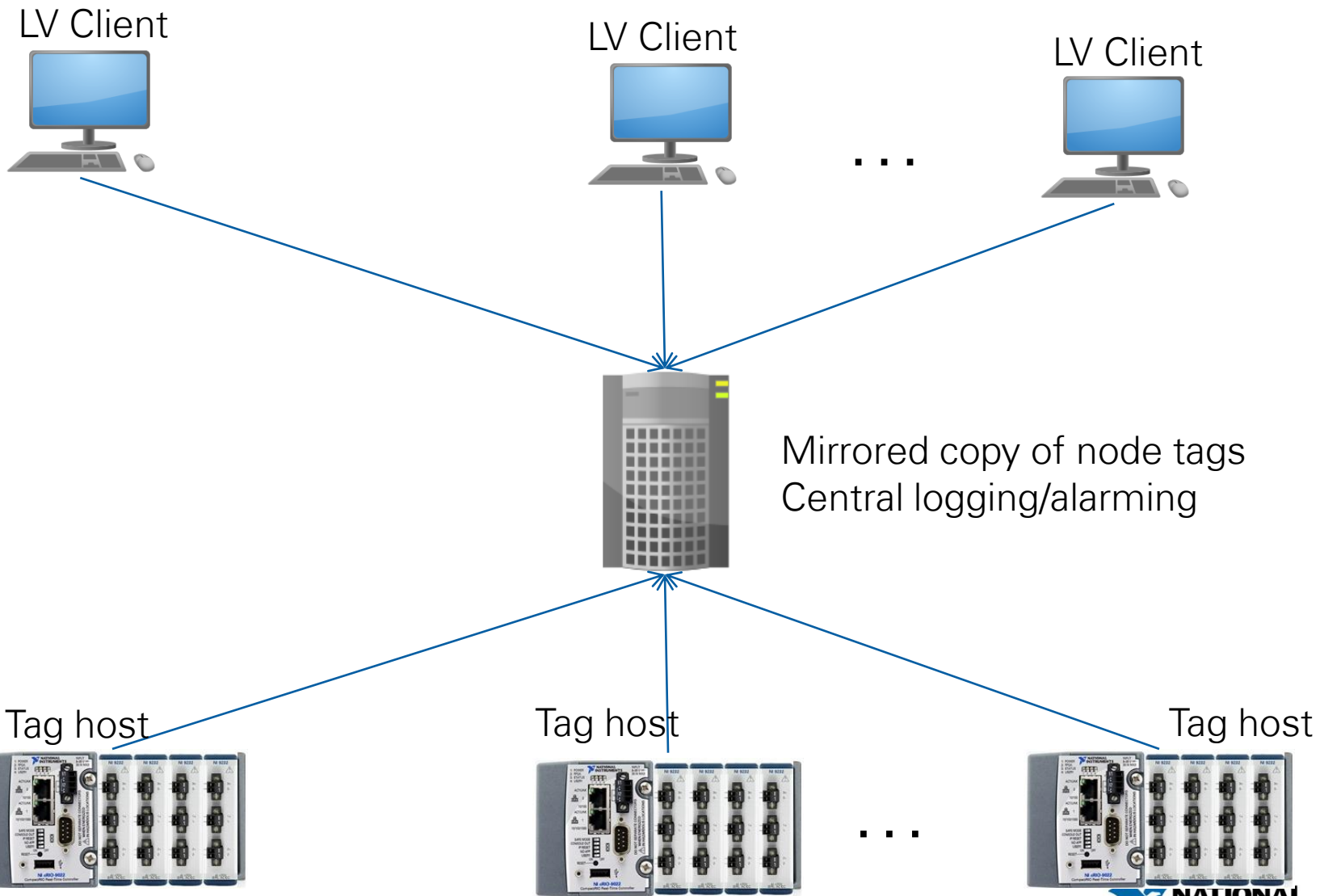
- Shared Variables are not designed for commands or streams.
  - They are a lossy communication mechanism
    - Sending an emergency stop message via Shared Variable is unwise
- Shared Variables work best at SCADA rates (.1-10s of Hz)
  - Pushing streaming data at 1 MB/Sec through a Shared Variable is unwise
  - Use a Network Stream instead
- Load on Shared Variable Engine scales linearly with number of subscribers and frequency of updates
- Tool exists to benchmark performance on your system
  - Search ni.com for 'Shared Variable Benchmark Utility'

Single-Server Architecture



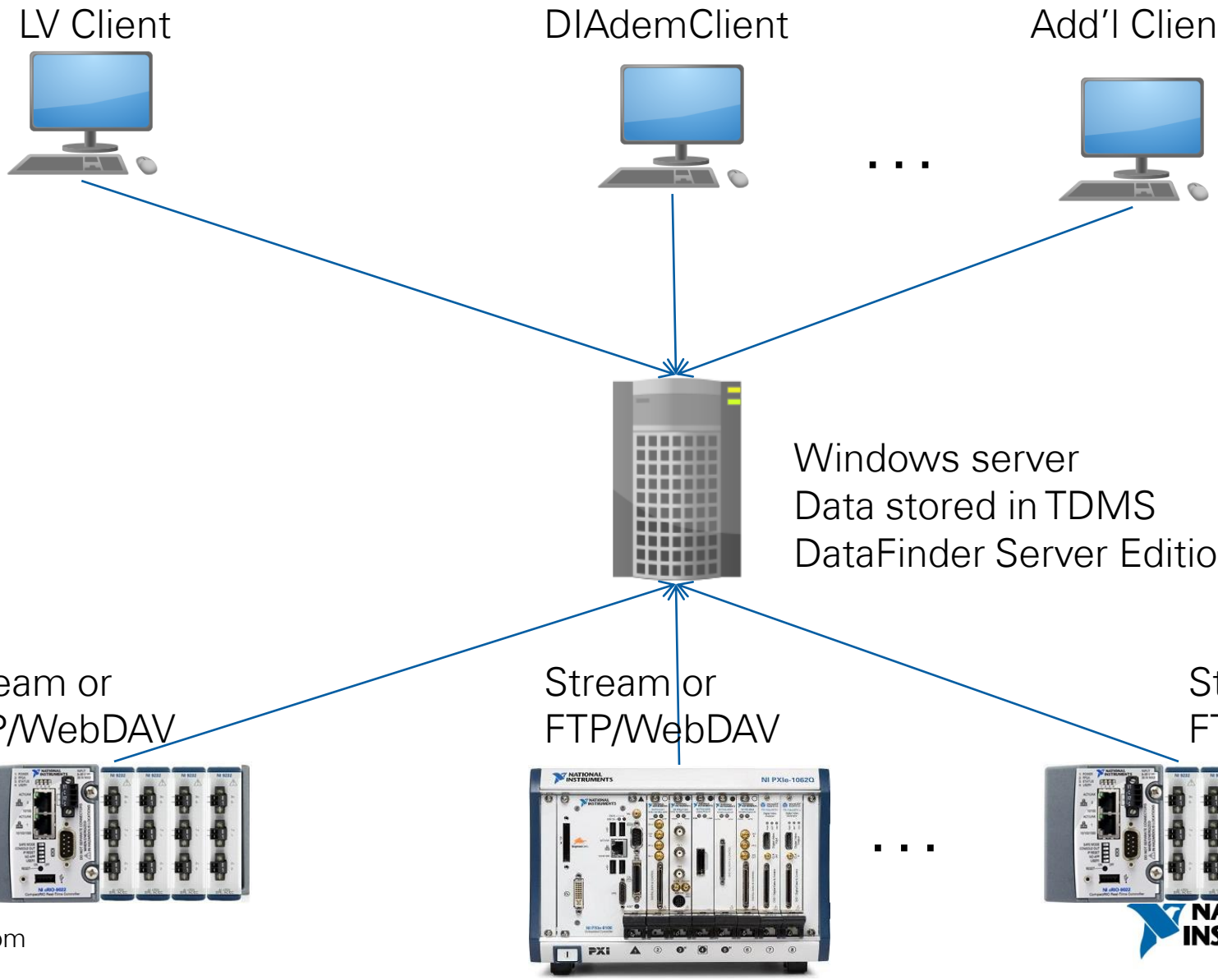
NATIONAL INSTRUMENTS

SCADA Use Case

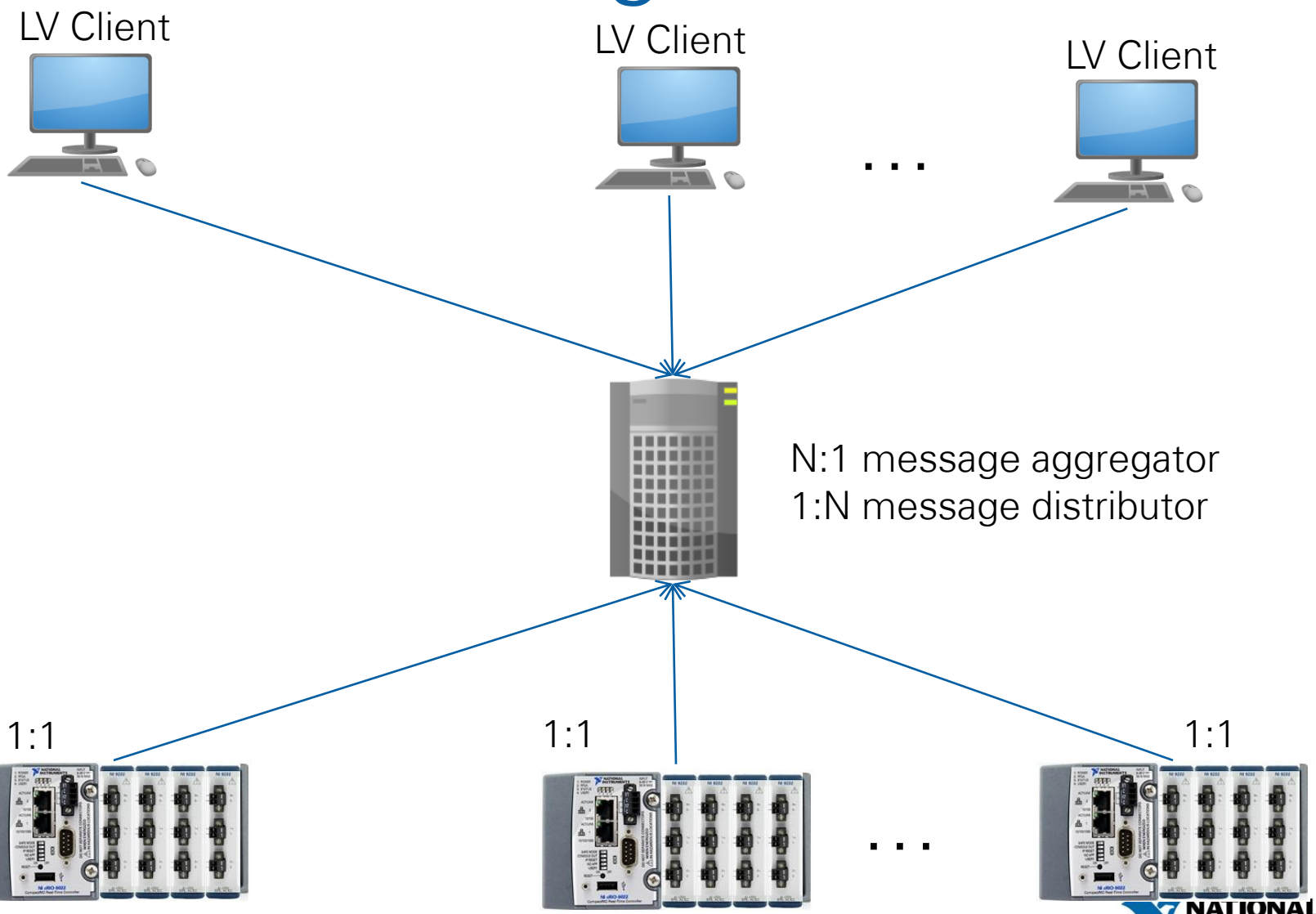


NATIONAL INSTRUMENTS

Offline Waveform Processing

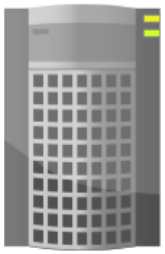


Message Broker

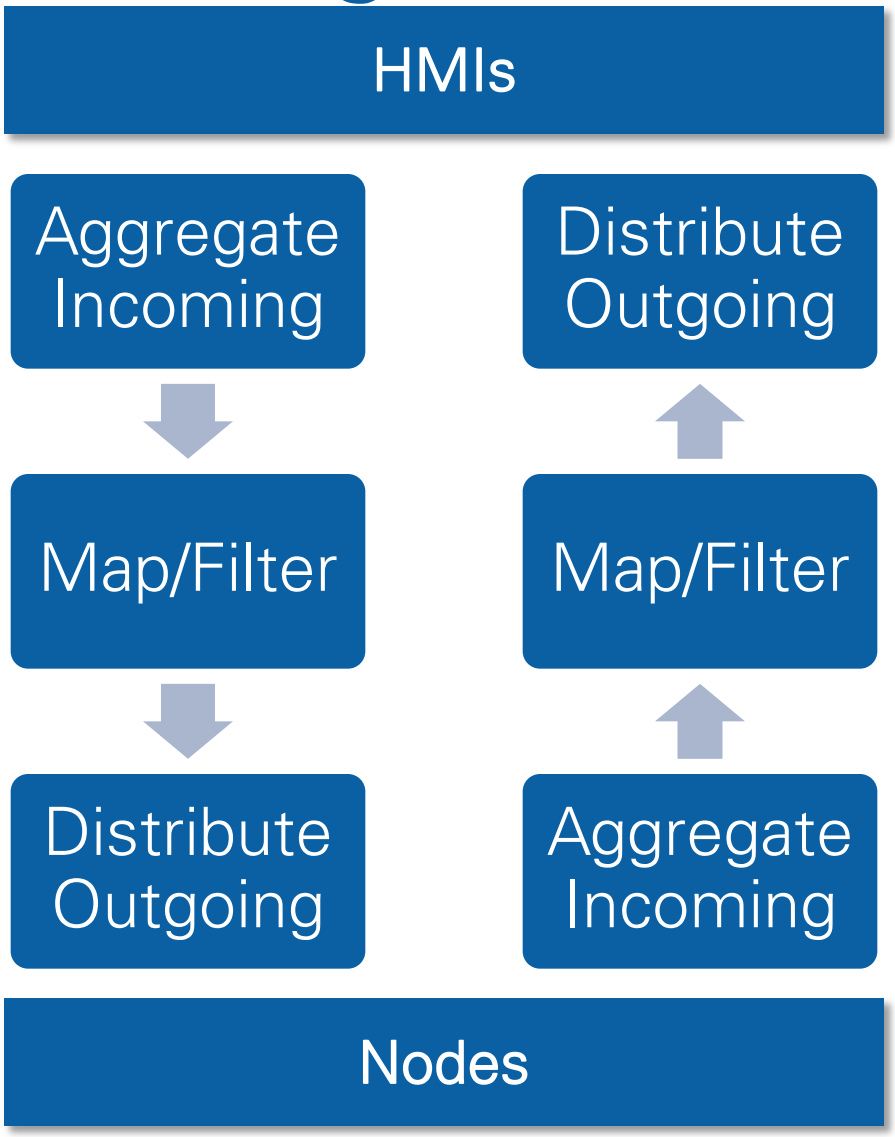


NATIONAL INSTRUMENTS

Message Broker

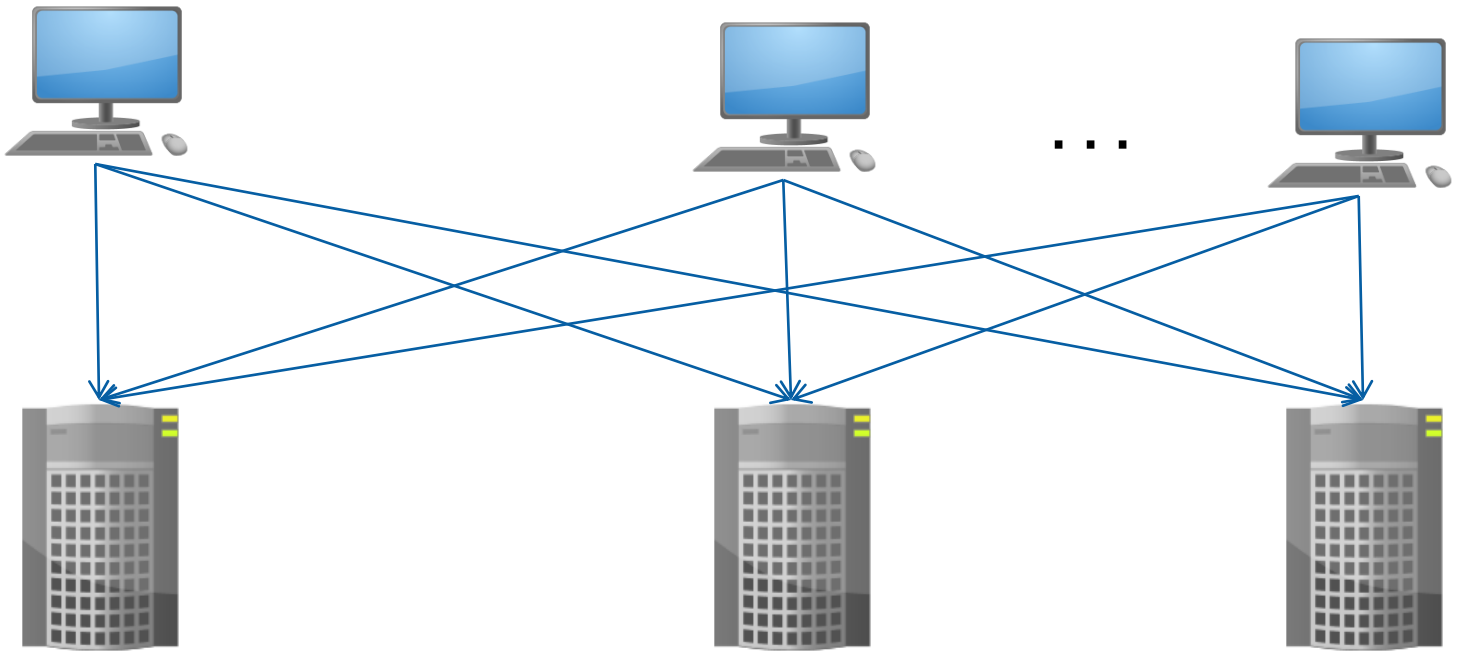


=

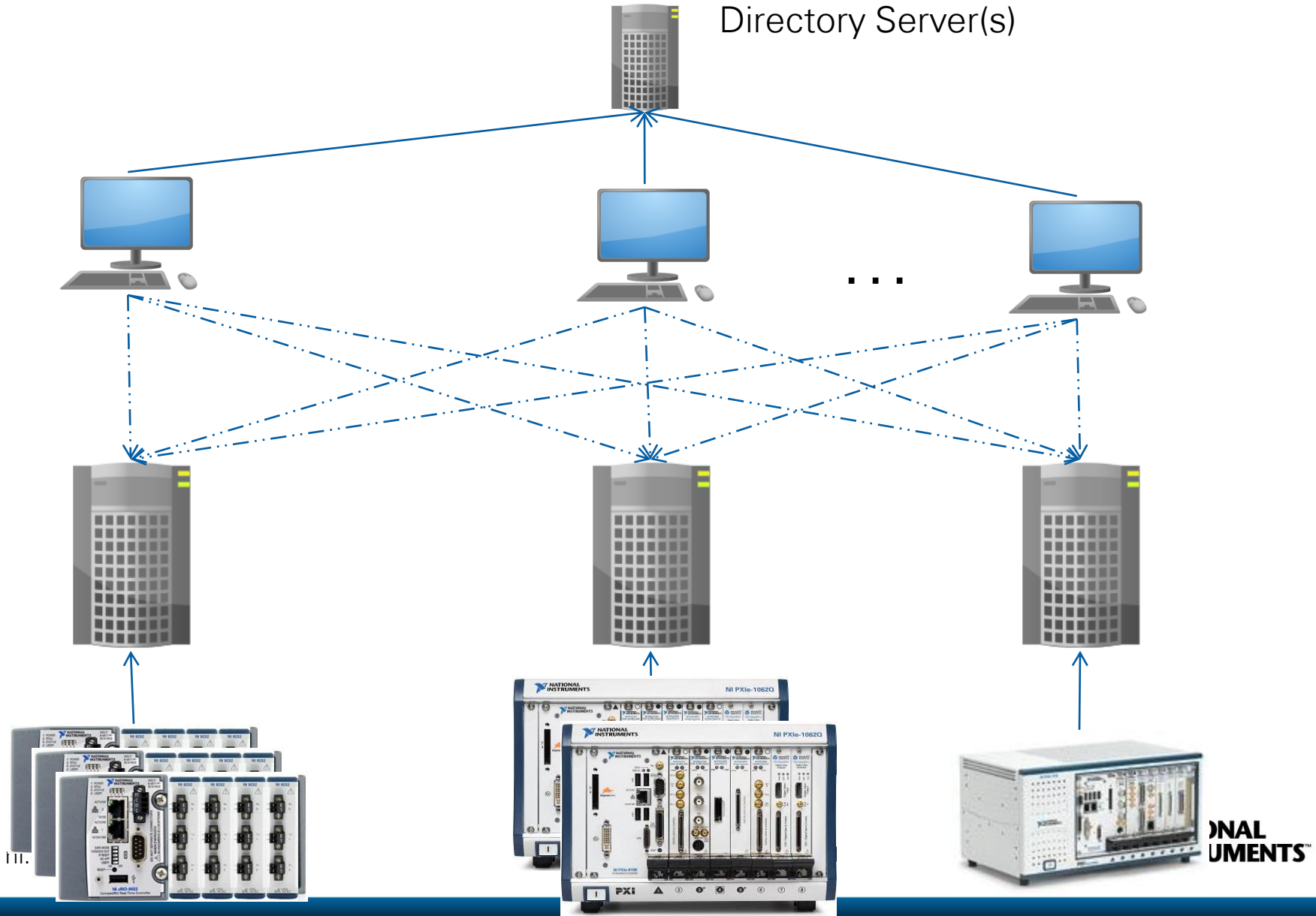


NATIONAL INSTRUMENTS

Multi-Server Architecture



Multi-Server with Lookup Service



Scalability Best Practices

- Proper communication channel selection
  - You probably need more than one!
- Minimize coupling between components (whether loops or nodes)
- Minimize configuration effort
  - Either remove the need, or automate
  - Human-readable config files are your friend
- Minimize traffic
  - Only send deltas, only send to those who need it
- Timestamp synchronization between targets