Introduction to Actor Framework

Anders Rohde, MSc.EE, CLD.
National Instruments Denmark

Agenda

- What Actor Framework Replaces
- How Actor Framework Minimizes Code Replication Using Command Pattern
- Getting started with Actor Framework
  - What Is An Actor
  - What Is A Message

Does this sound familiar?

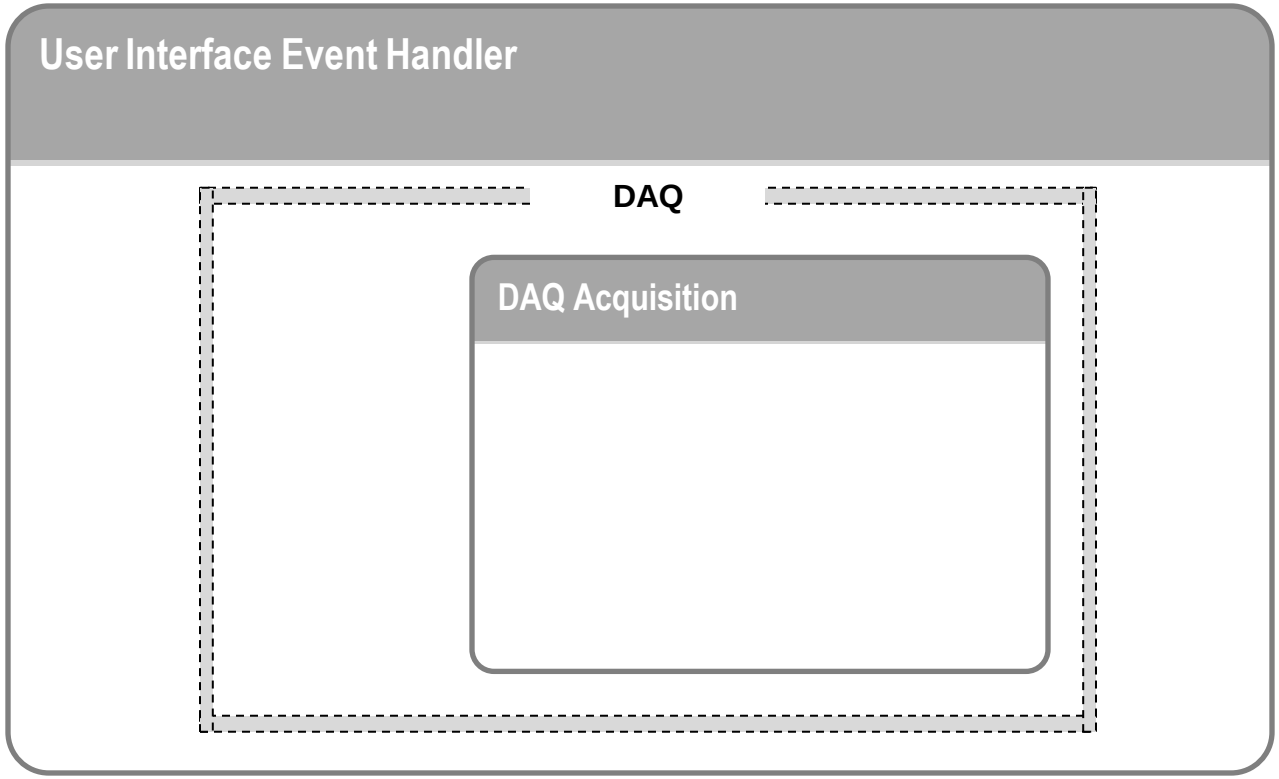
- Adding or changing support for hardware requires significant effort and time.
- You're constantly copying and duplicating code to add similar, but different functionality.
- Adding new functionality breaks code due to brittle design.

Scalable
Modular
Reusable

NATIONAL INSTRUMENTS

What Actor Framework Replaces

Why is Coupling Bad?



Execution: *The user interface cannot respond to other input while acquiring data*
Development: *Modifying one process requires modifying the other, introducing added risk*
Extensibility: *How do you add a data logging process?*

De-Coupling Independent Processes

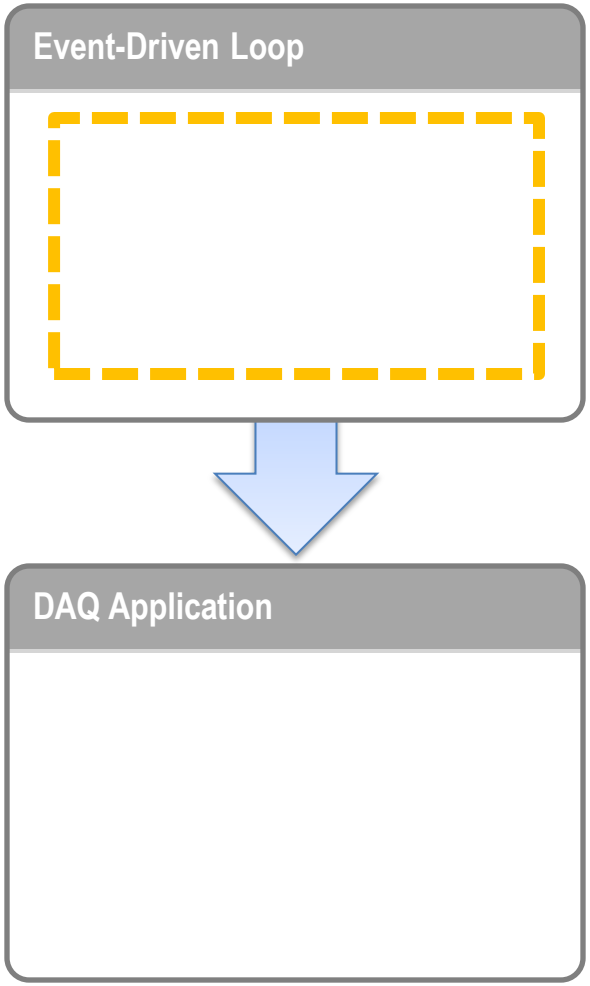
Best Practices

1. Identify data scope
2. Delegate actions to appropriate process
3. Do not poll or use timeouts*

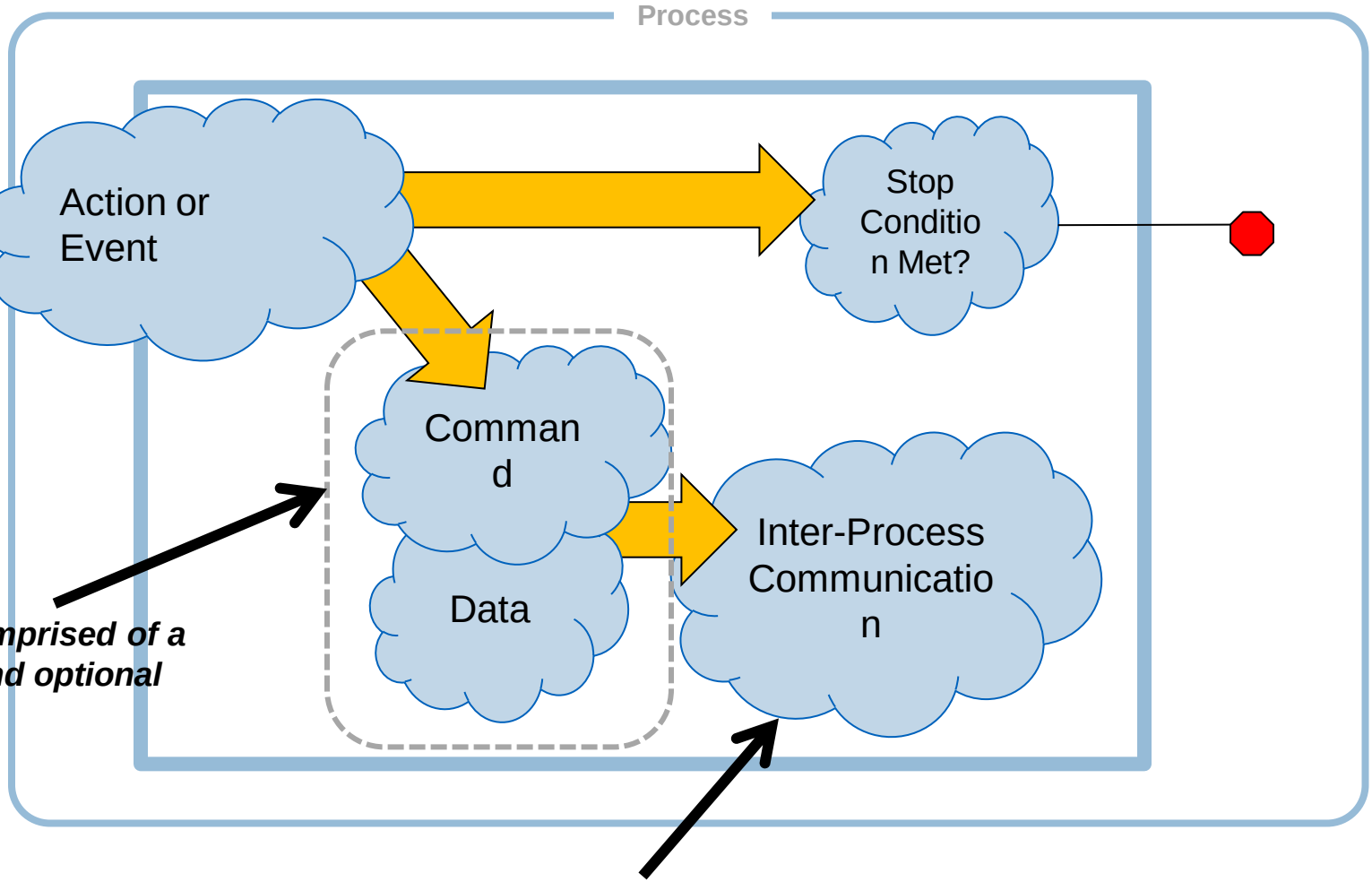
**except for code that communicates with hardware*

Considerations

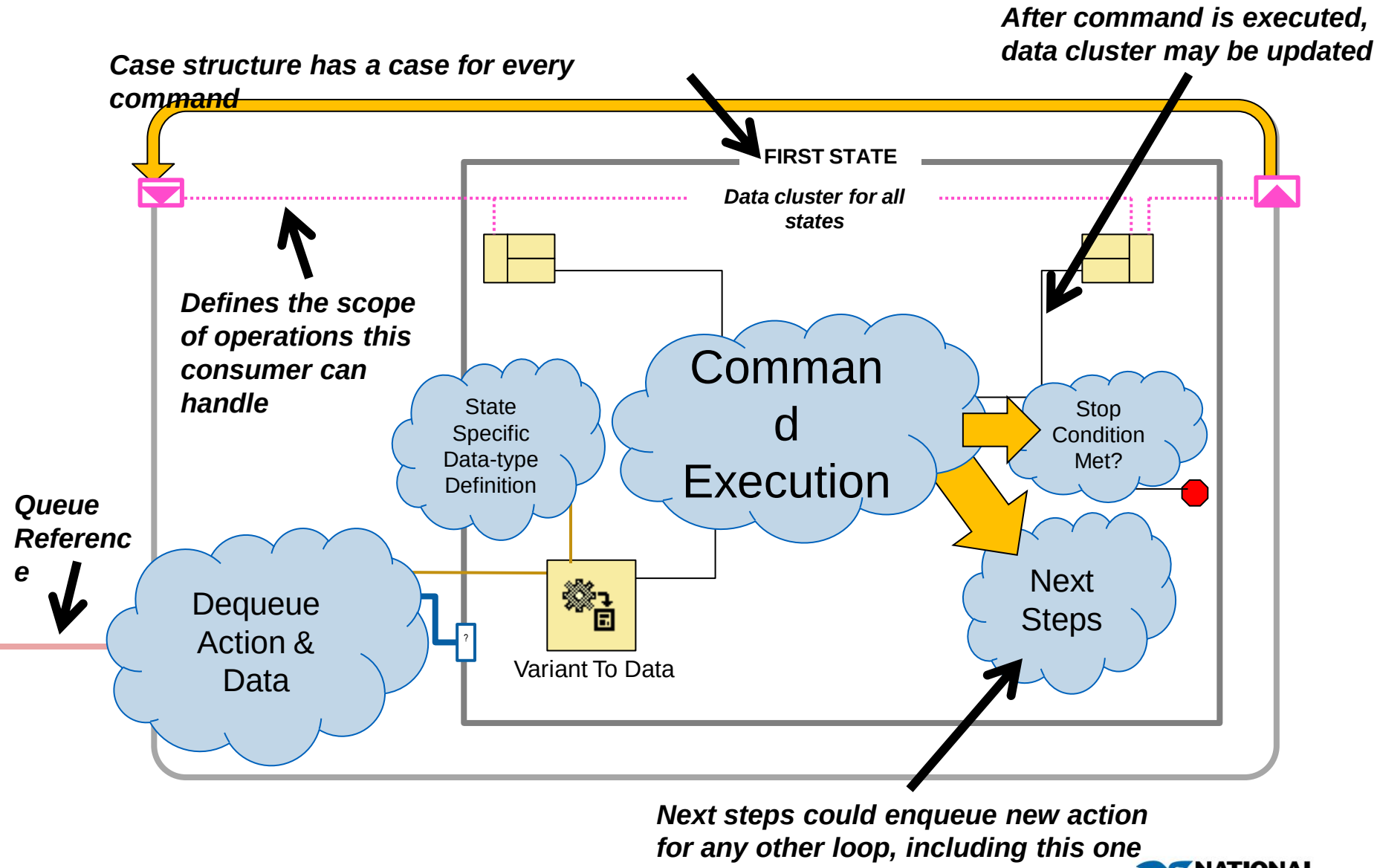
1. How do you send commands?
2. How do you send data?
3. Which processes can communicate with each other?
4. How do you update the UI?



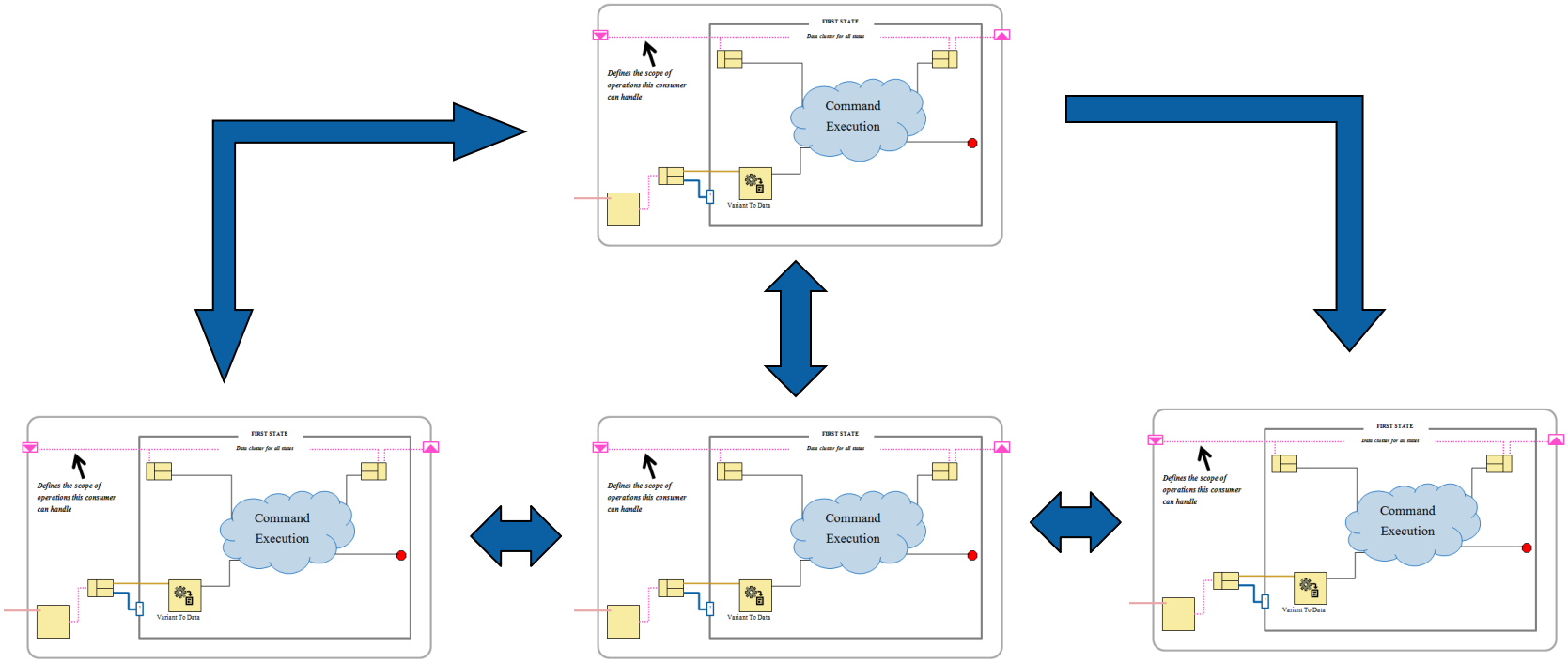
Anatomy of a Message Producer Process



Anatomy of a Message Consumer Process



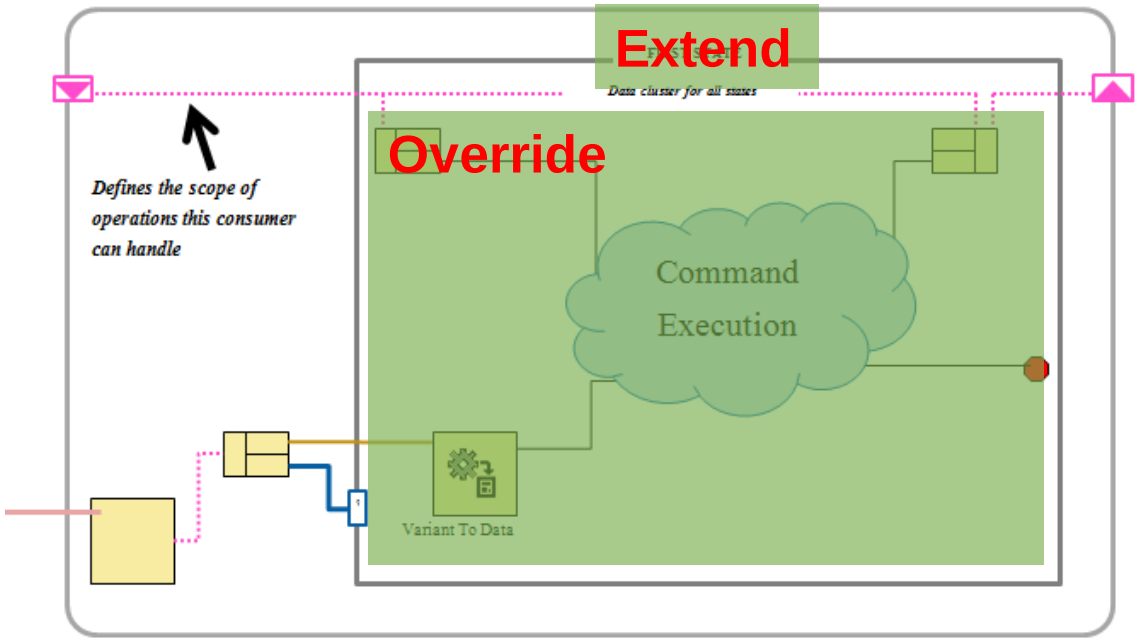
Larger Systems



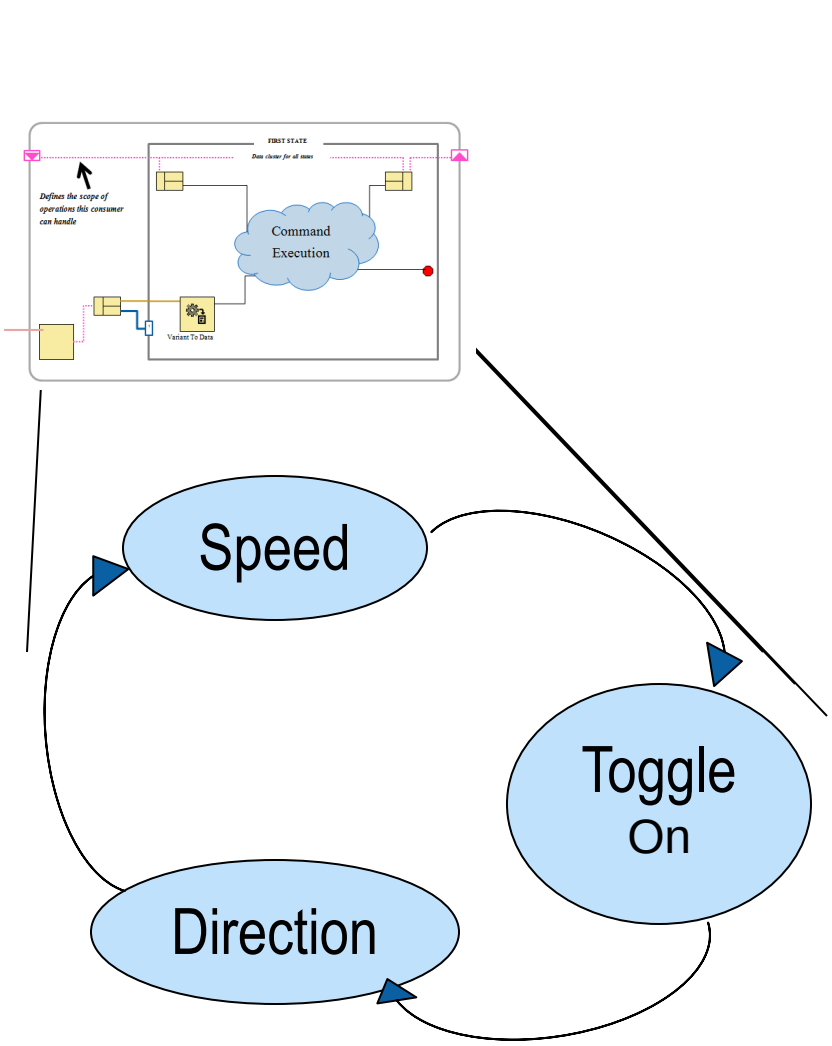
How Actor Framework Minimizes Code Replication Using Command Pattern

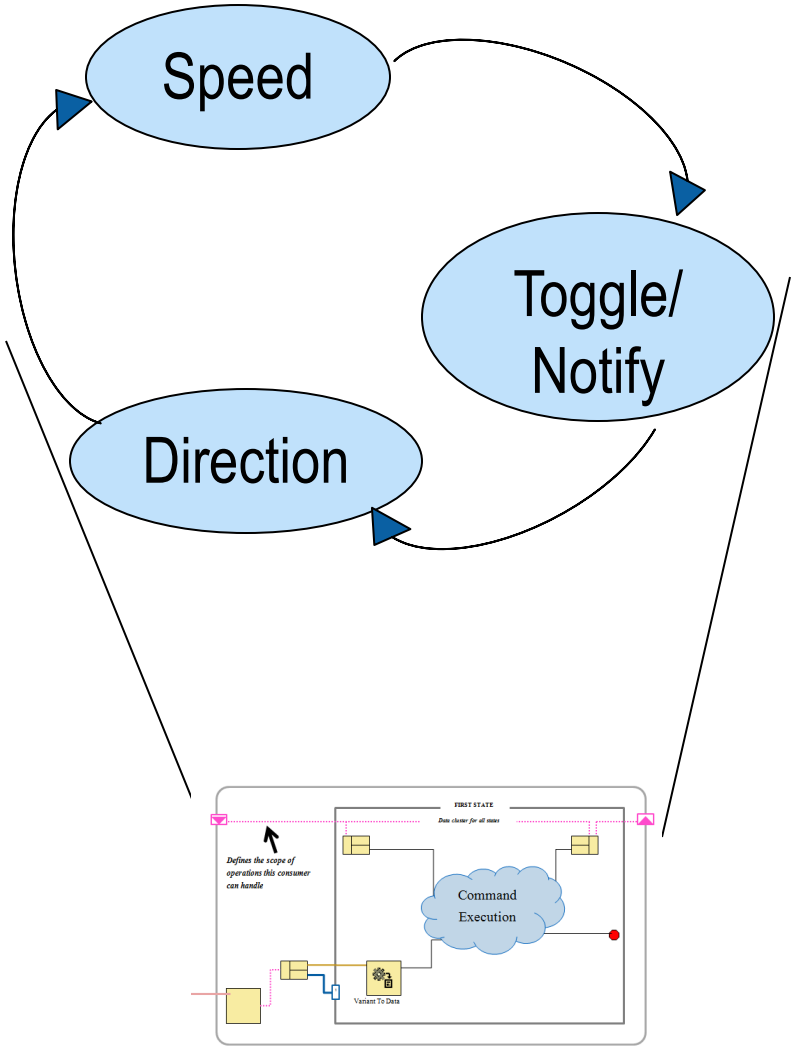
Common Sources of Code Replication

1. **Override** the handling of one message
2. **Extend** the set of handled messages

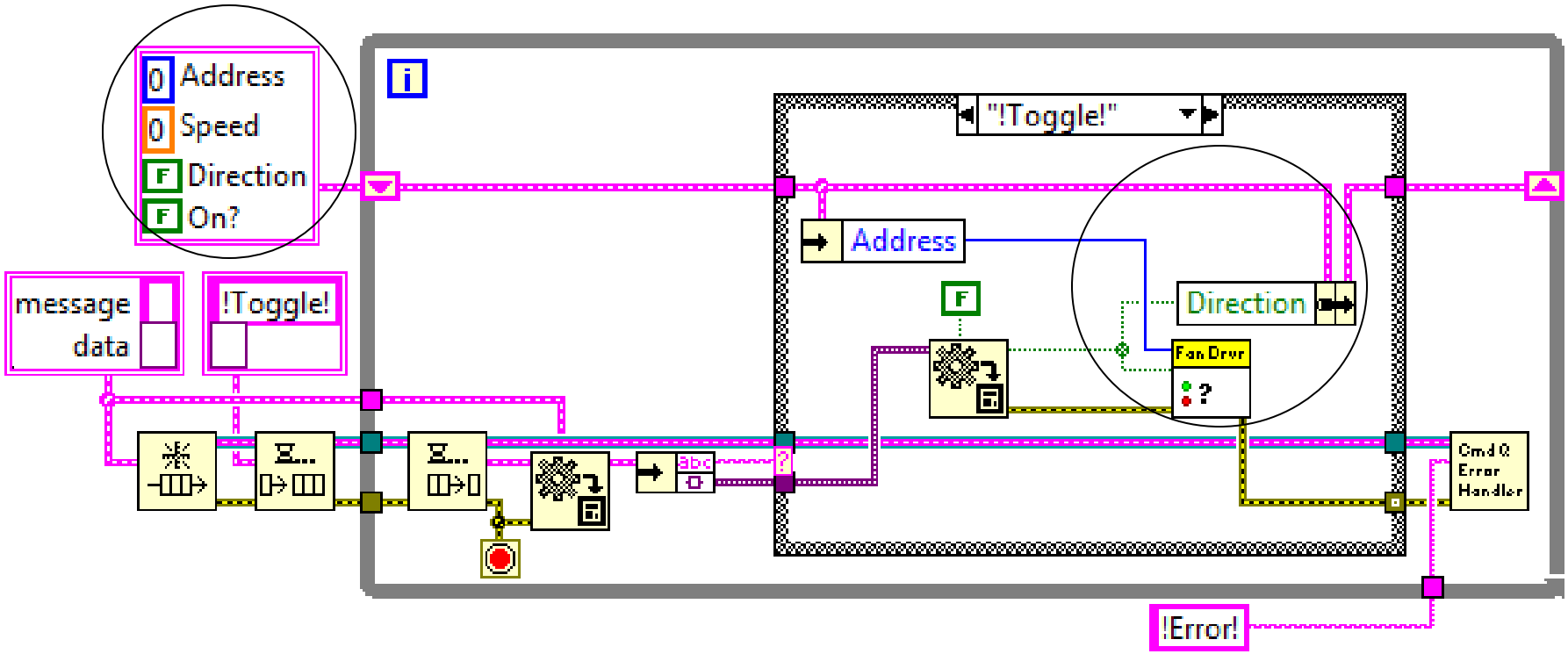


Source 1: Override

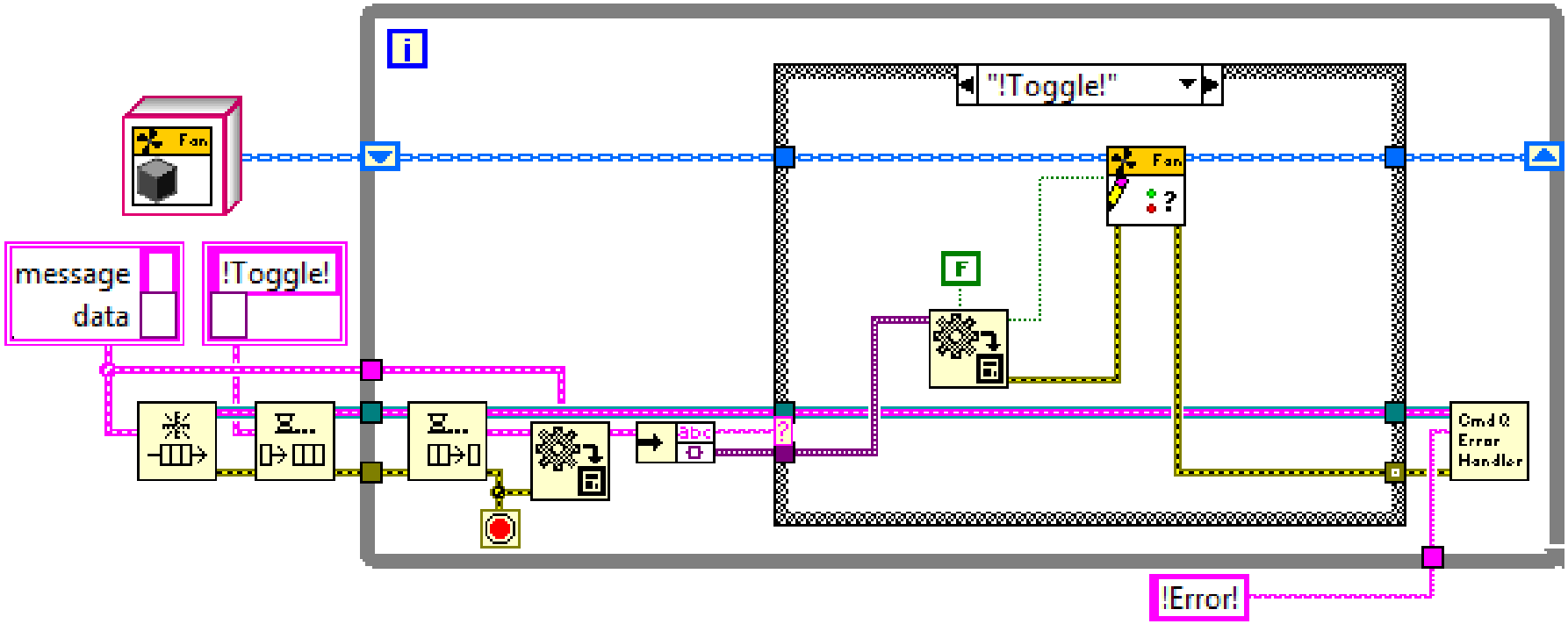




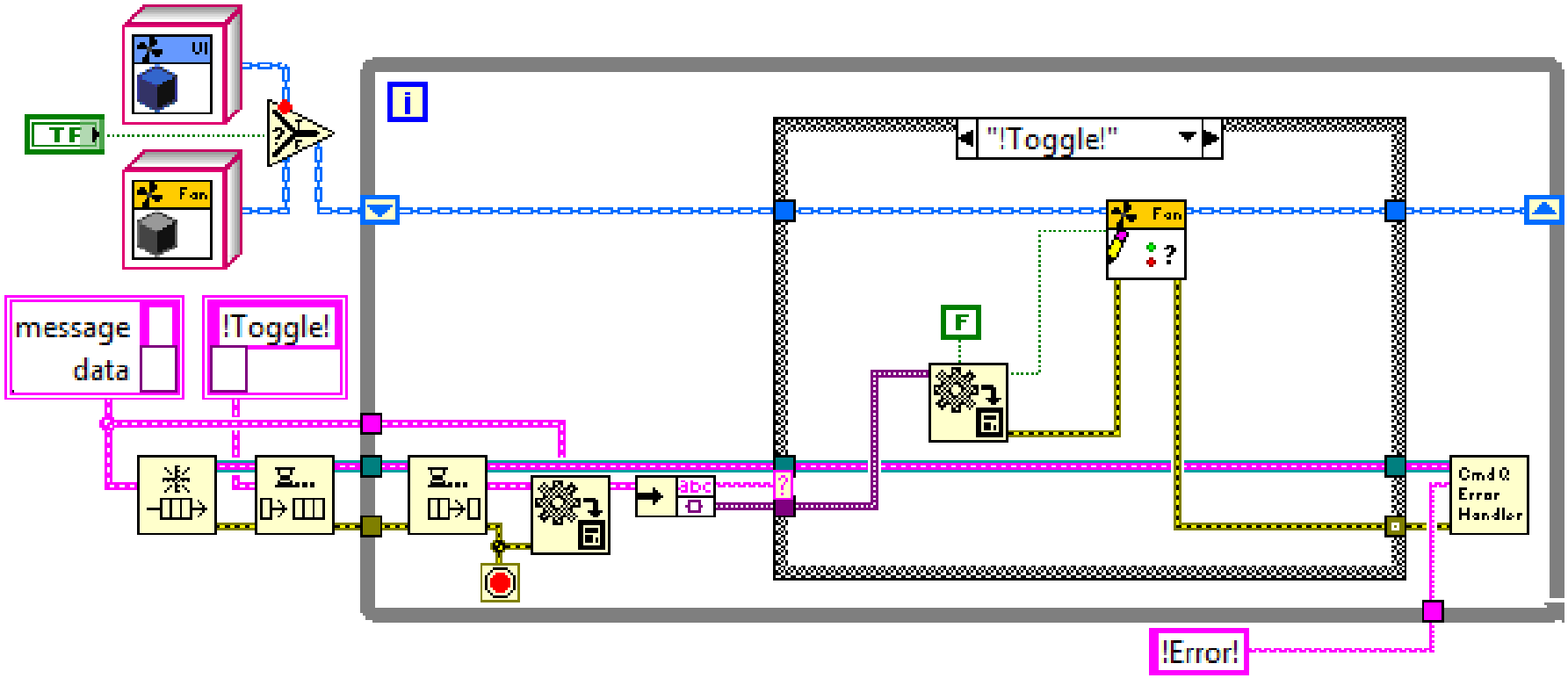
Cluster and Node...



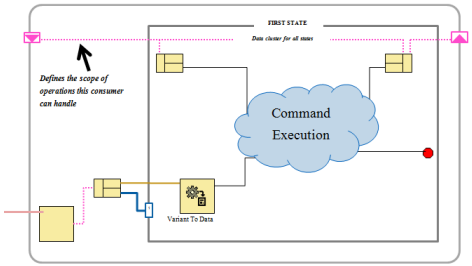
… become Class and Method

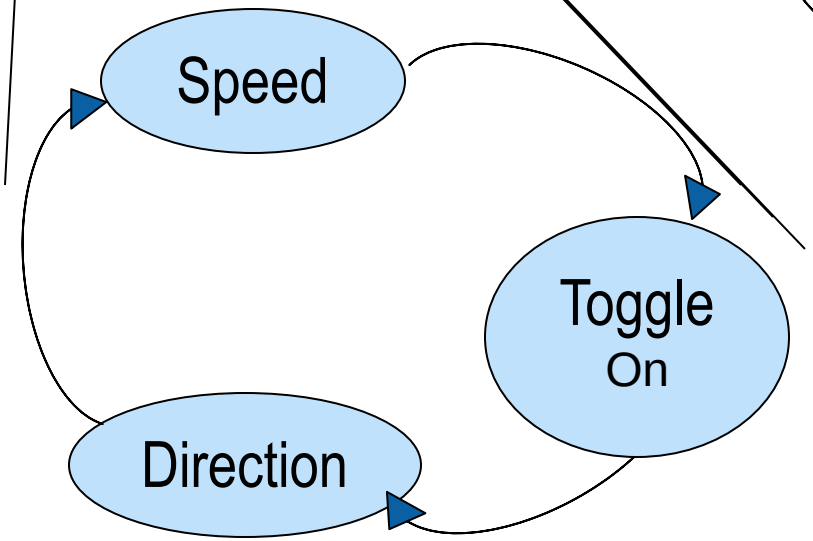


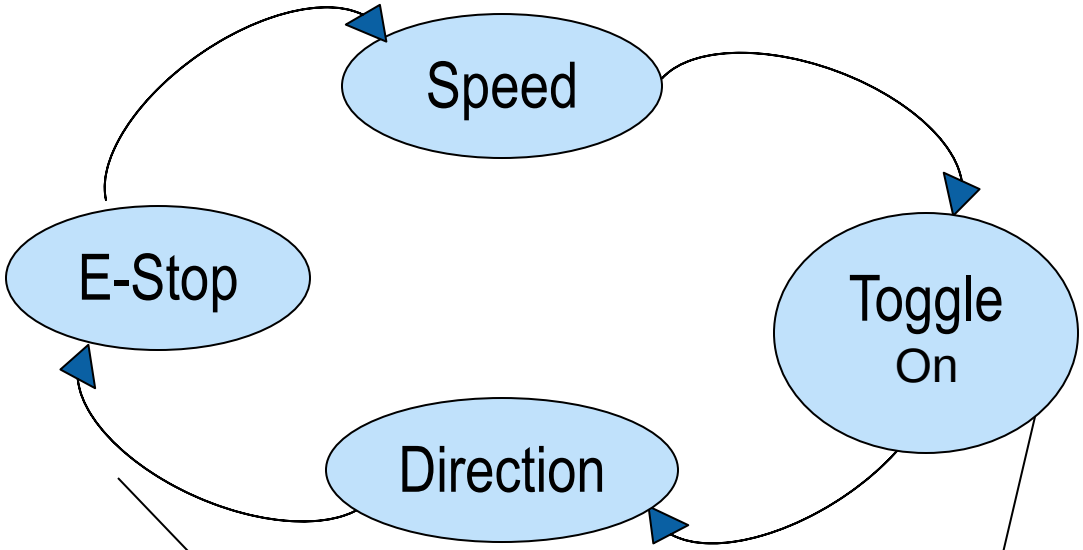
… become Class and Method

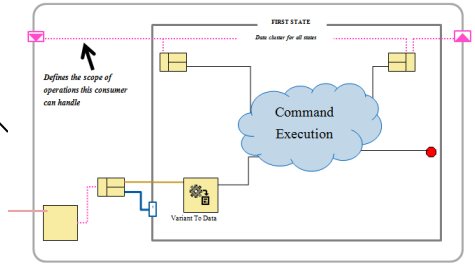


Source 2: Extend

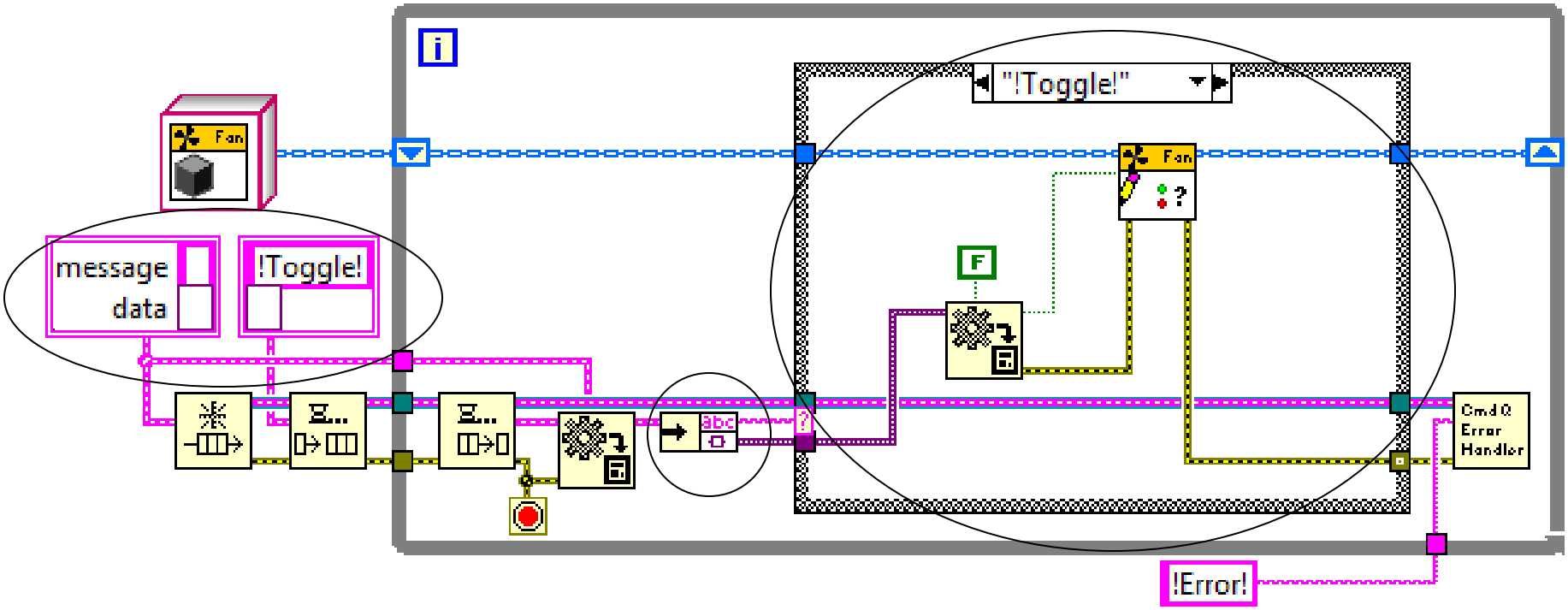




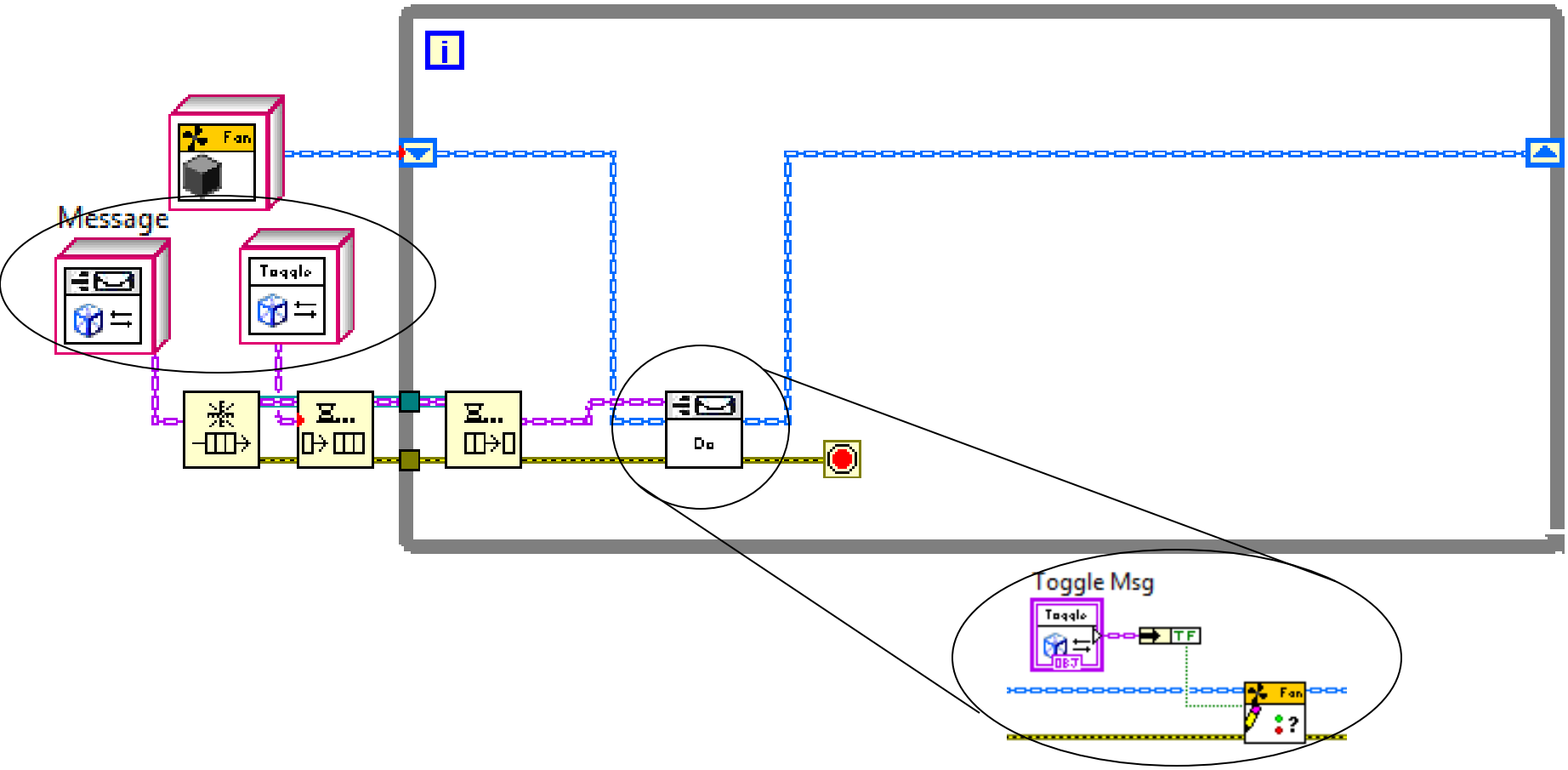




Message and Case Structure…

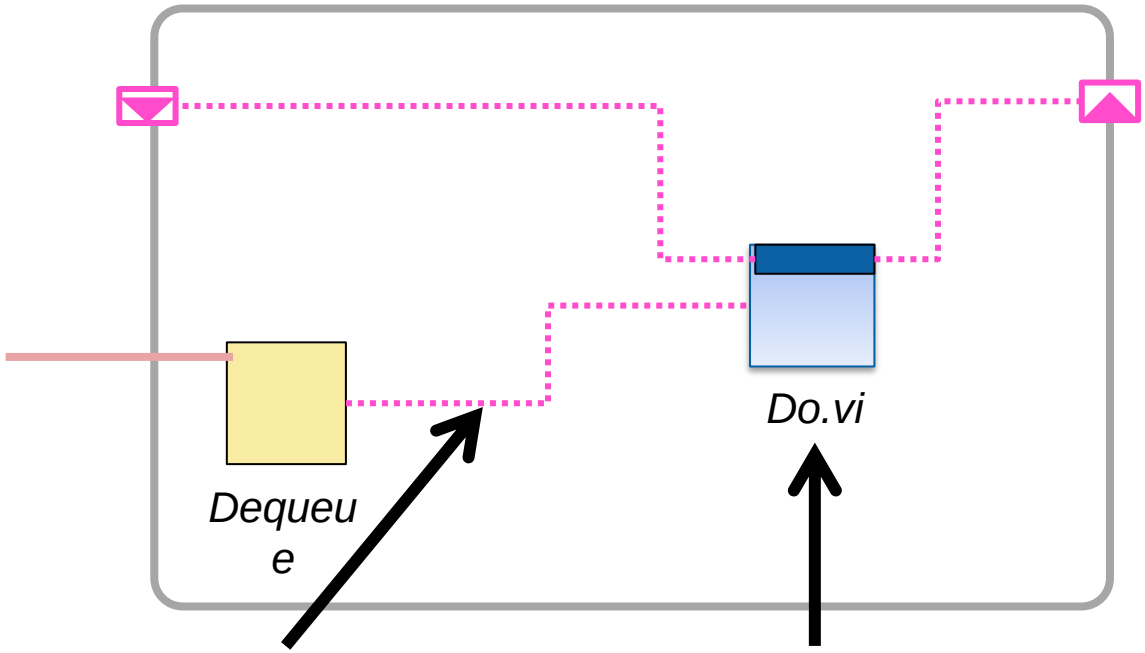


… become Class and Dynamic Dispatch



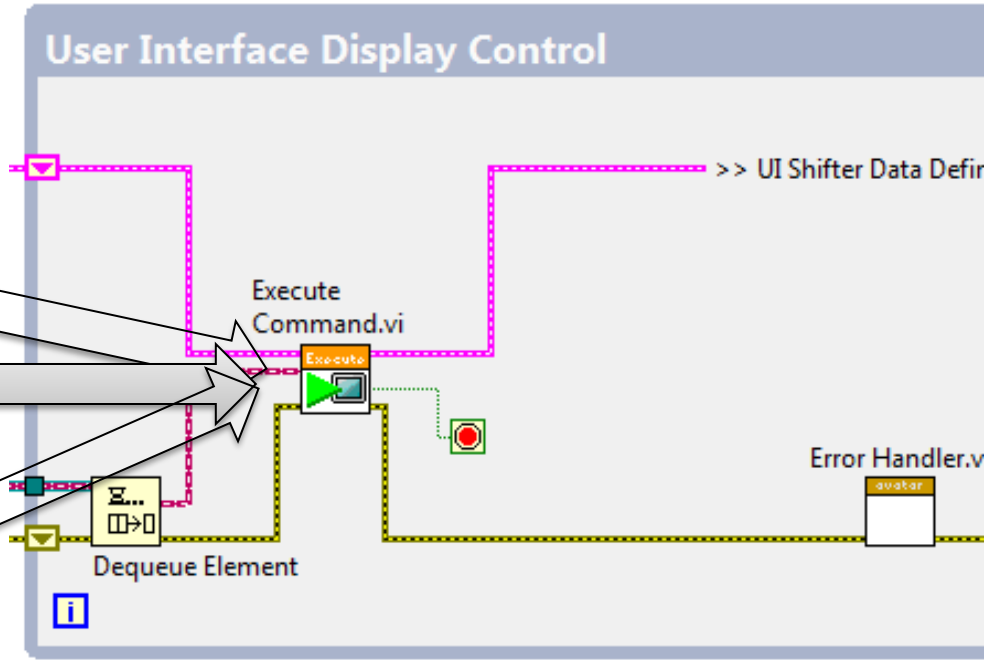
The Command Pattern

The object that is de-queued dictates which copy of Do.vi gets run at run-time, because Do.vi is ***Dynamically Dispatched***



Dynamic Dispatching Commands

- User Interface
  - UI Command Manager
    - UI Command Manager.lvclass
      - UI Command Manager.ctl
      - Send Command.vi
      - Execute Command.vi
    - Initialize Command.lvclass
      - Initialize Command.ctl
      - Execute Command.vi
      - Read UI Configuration Information.vi
      - Write UI Configuration Information.vi
    - Load SubPanel Command.lvclass
      - Load SubPanel Command.ctl
      - Execute Command.vi
      - Read VI reference.vi
      - Write VI reference.vi
      - Display Plugin UI in SubPanel.vi
    - Menu Toggle Command.lvclass
    - Populate Menu Command.lvclass
    - Resize Menu Command.lvclass
    - Set Header Text Command.lvclass

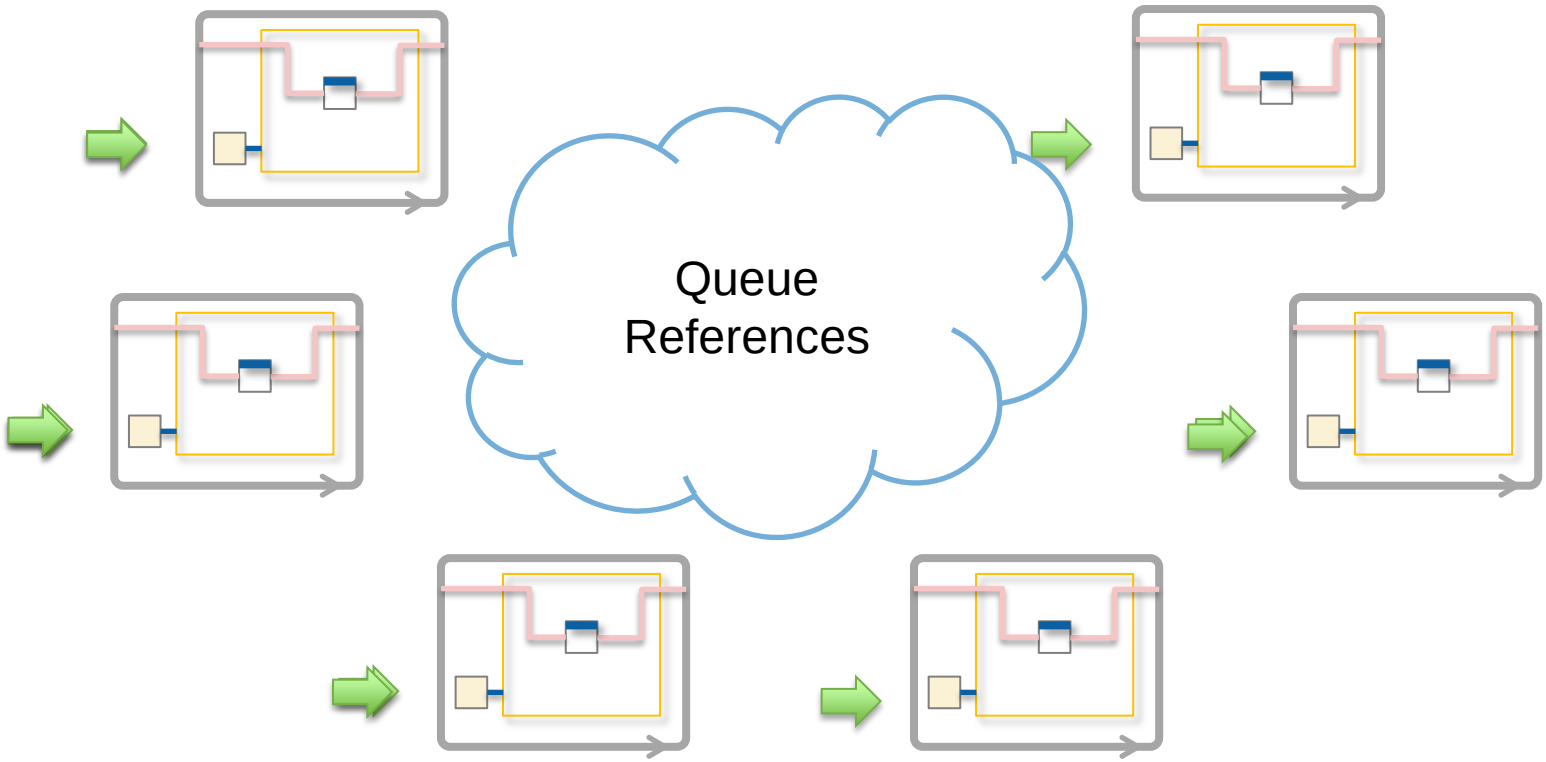


Small Demo

Command Pattern

Getting started with Actor Framework

Use Case for the Actor Framework



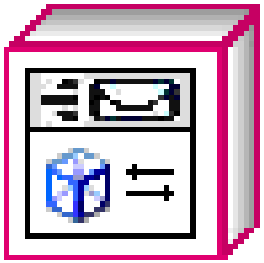
Use the Actor Framework to scale numerous different, but similar consumer loops

Traditional Solution: Dynamically load VI or duplicate code

Understanding the Actor Framework

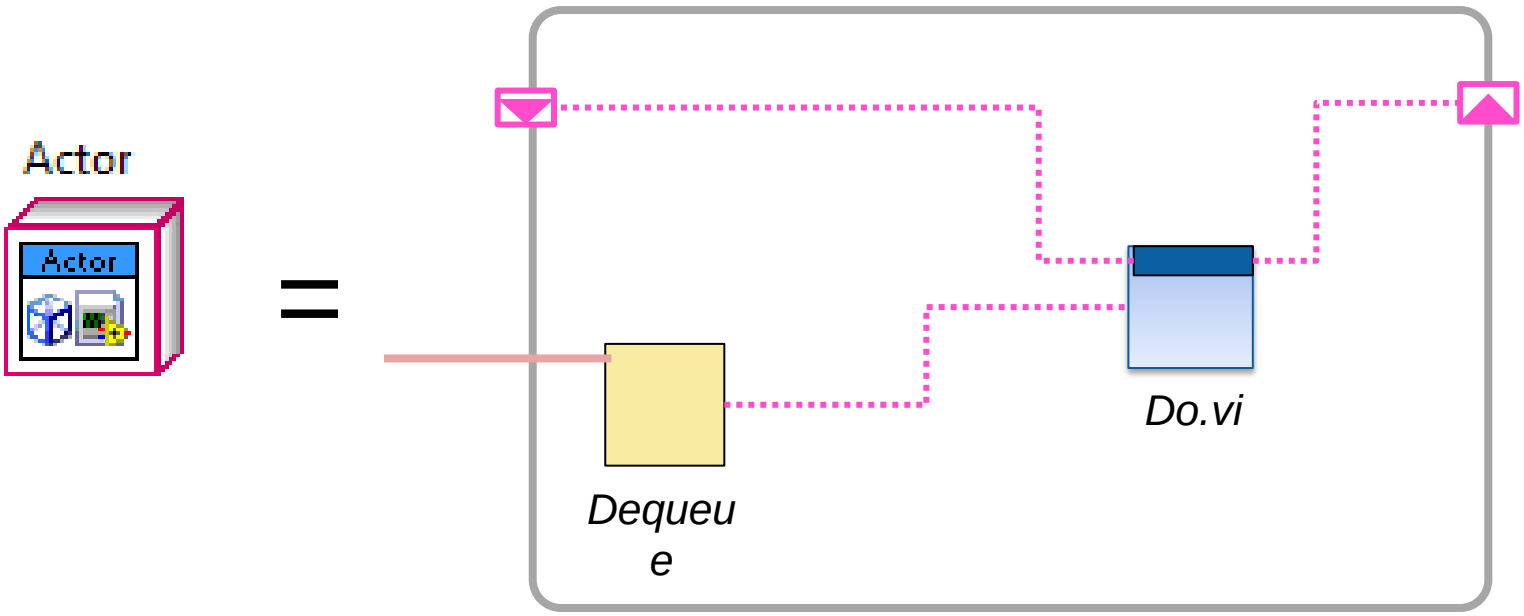
Two key components:

Actor



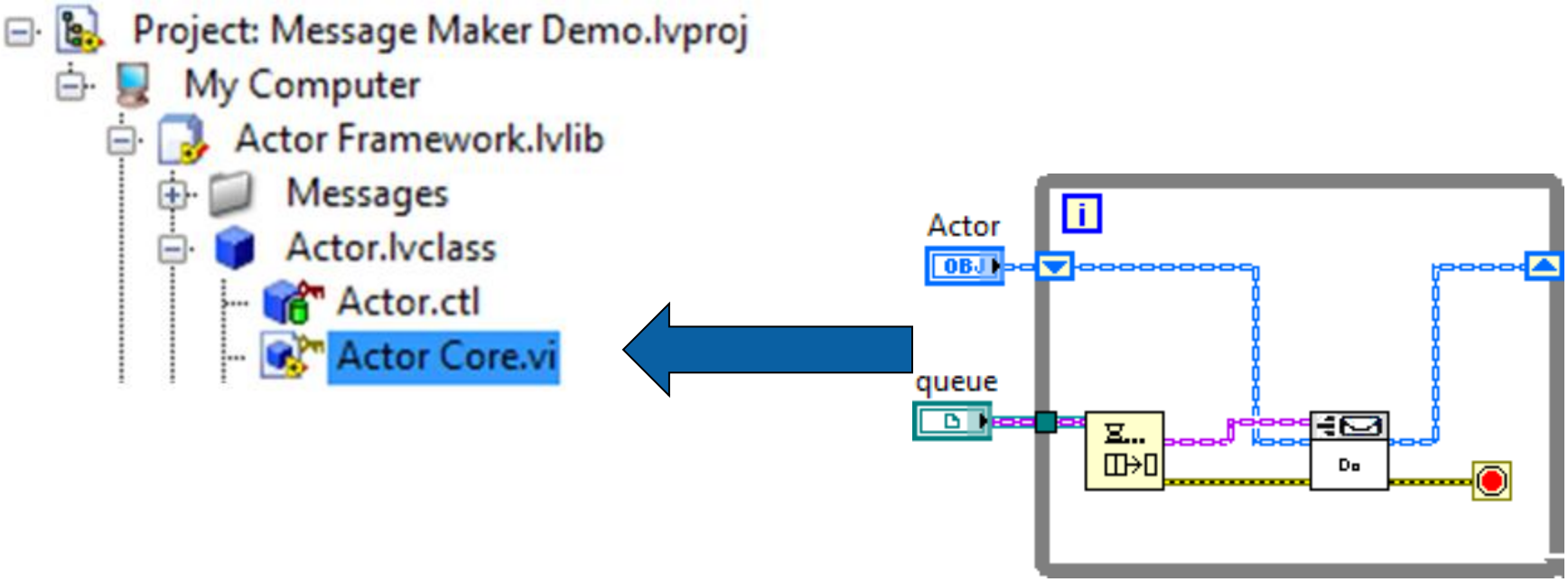
Message

What is an Actor?

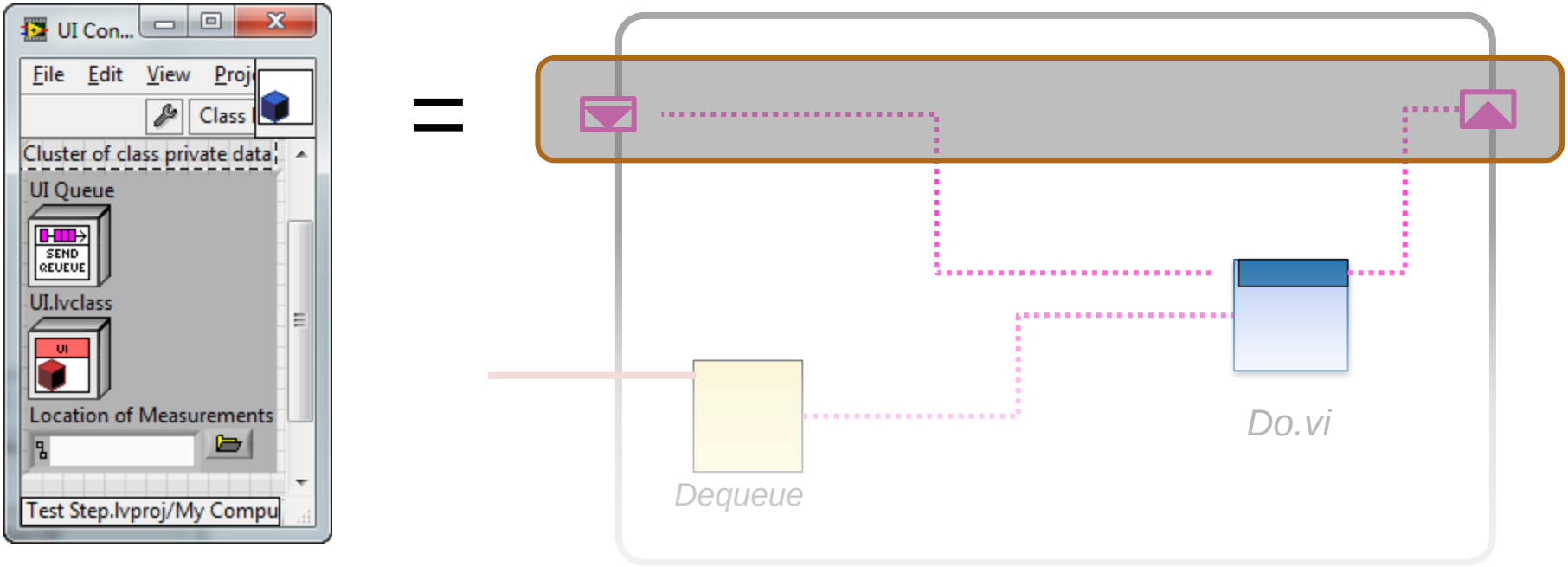


An actor is conceptually the same as a queued message handler (QMH) or queued state machine (QSM). This functionality actually resides within **actor core.vi**

New Class: Actor.lvclass

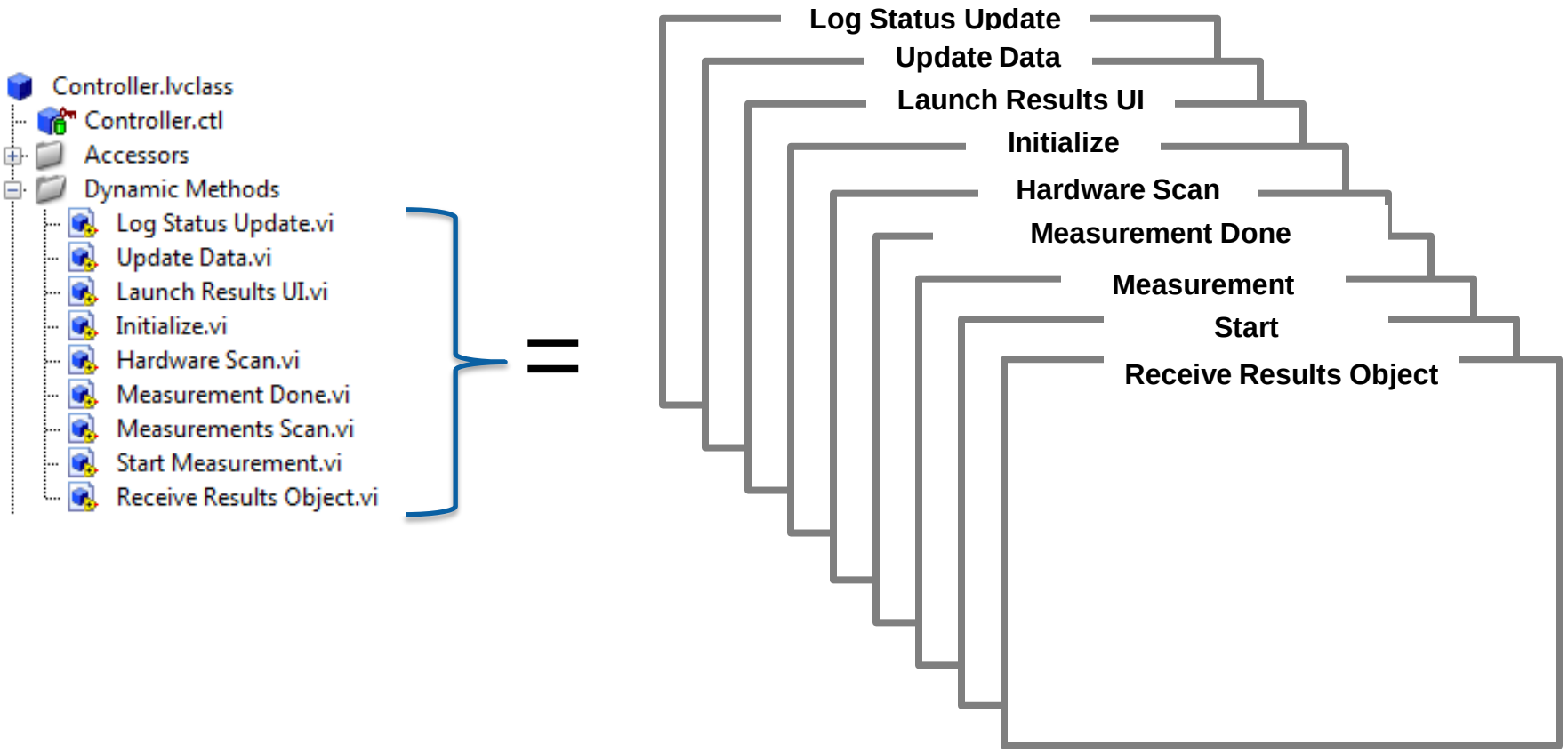


What is the Data Scope of an Actor?



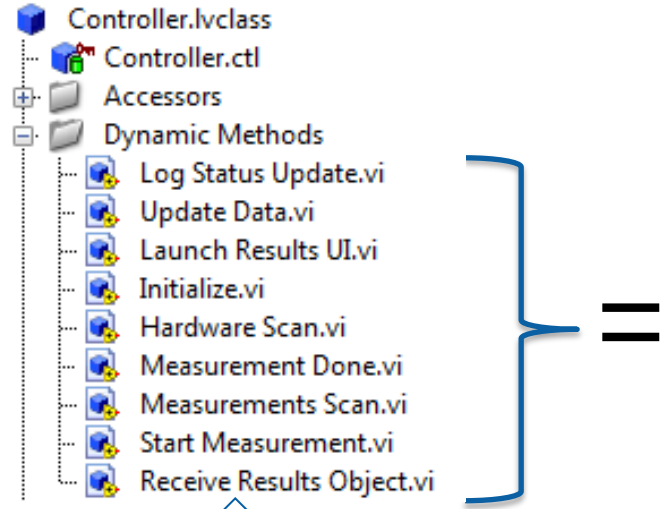
Private data control of an Actor defines the data scope of the queued message handler

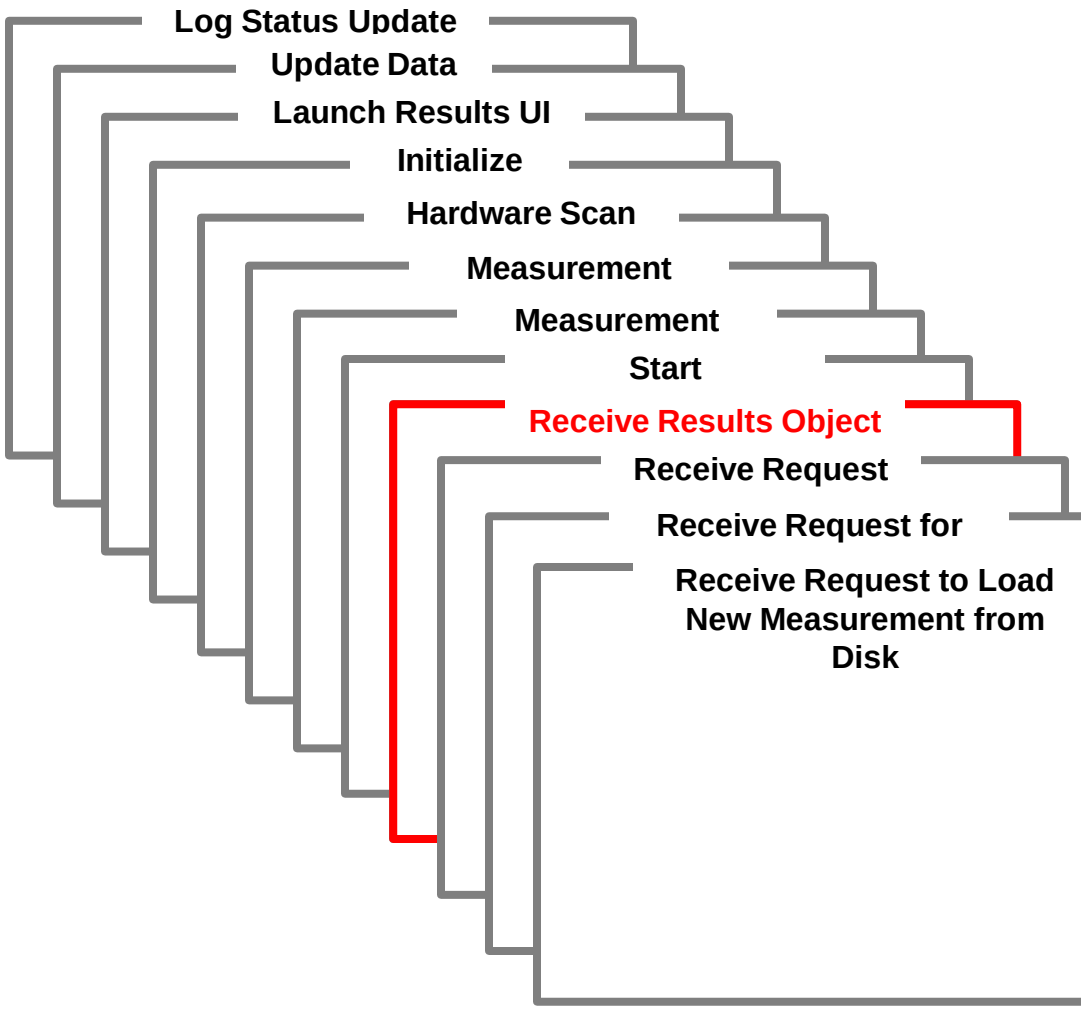
What are the States of the Actor?



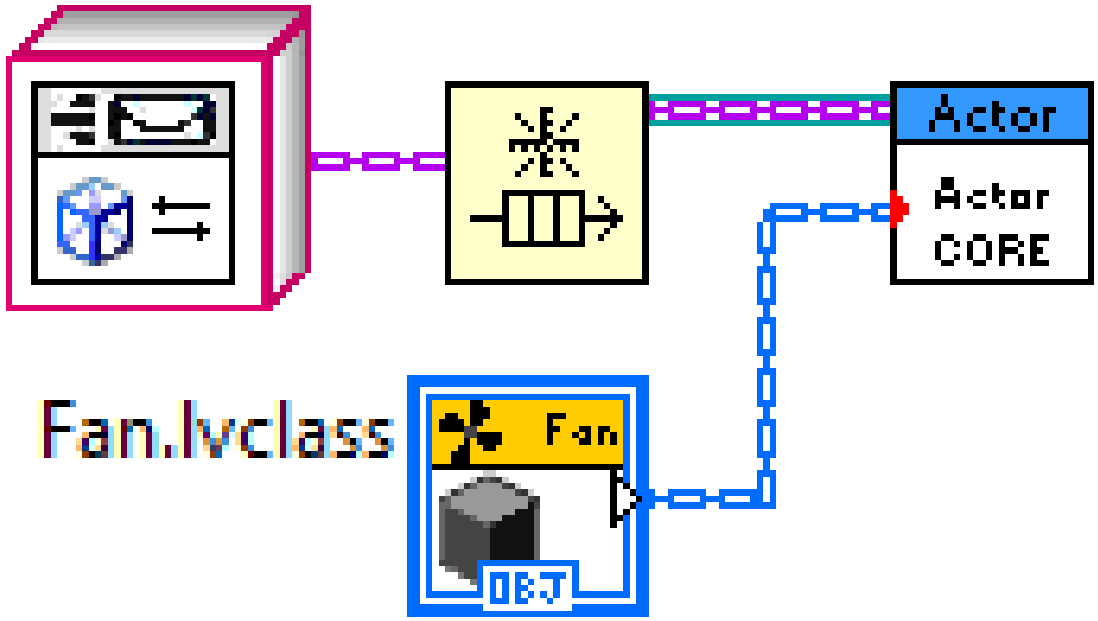
Methods of an Actor are used to represent the various states of a queued state machine
(or messages of a queued message handler)

What are the States of the Actor?

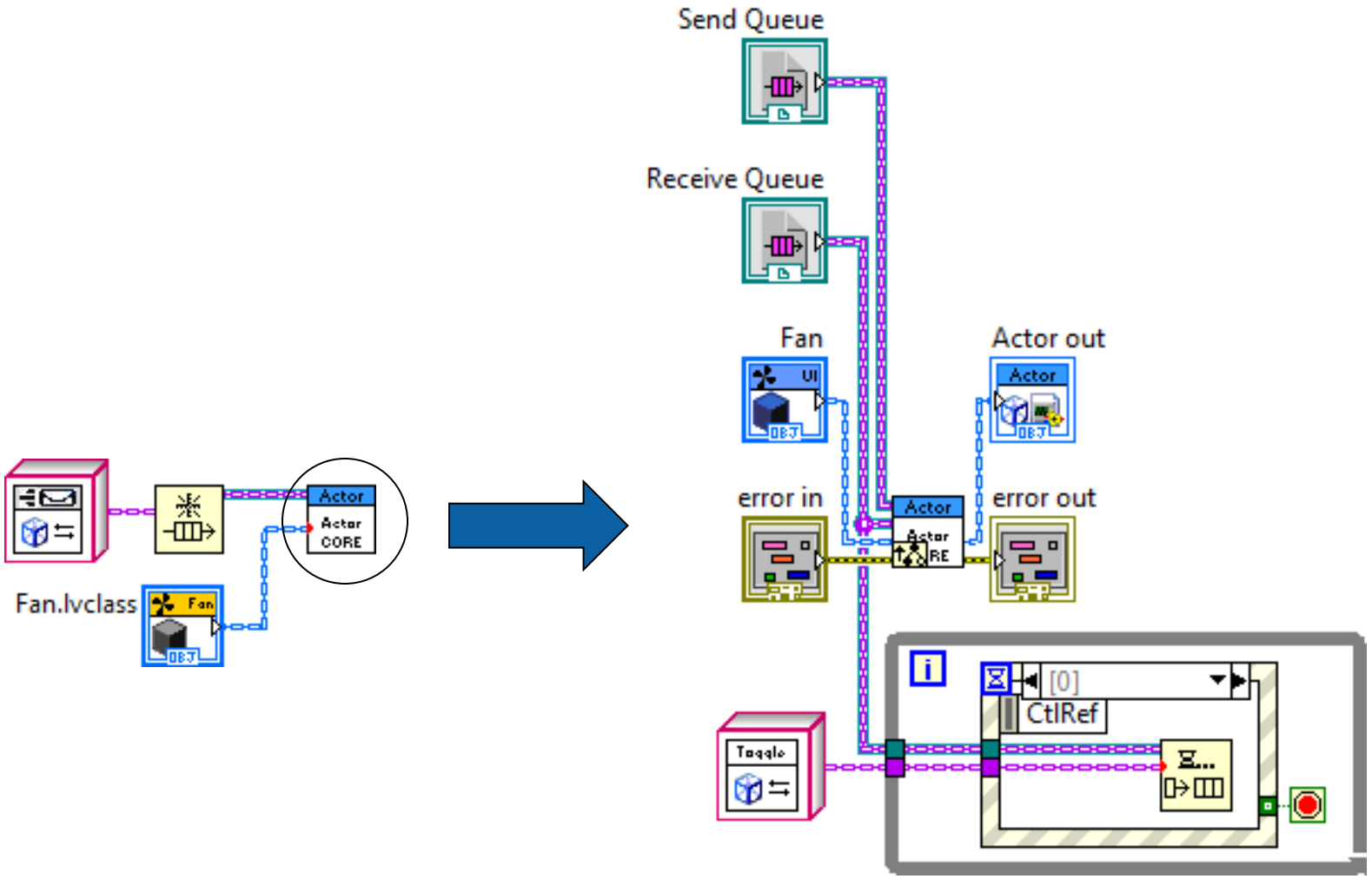




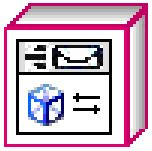
Using the Parent



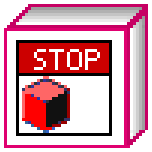
Using the Child



What is a message



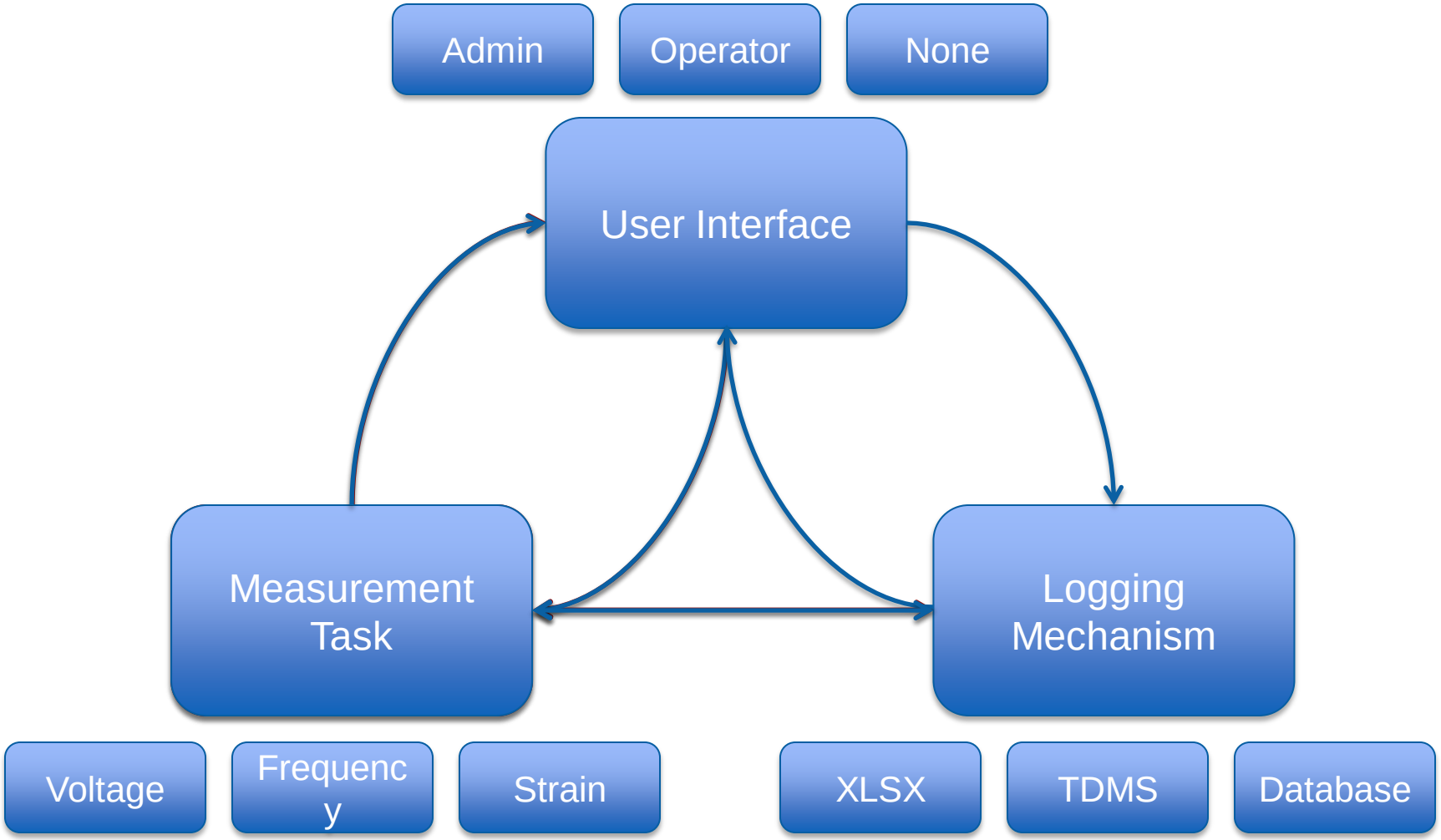
Message.lvclass is the first ancestor class



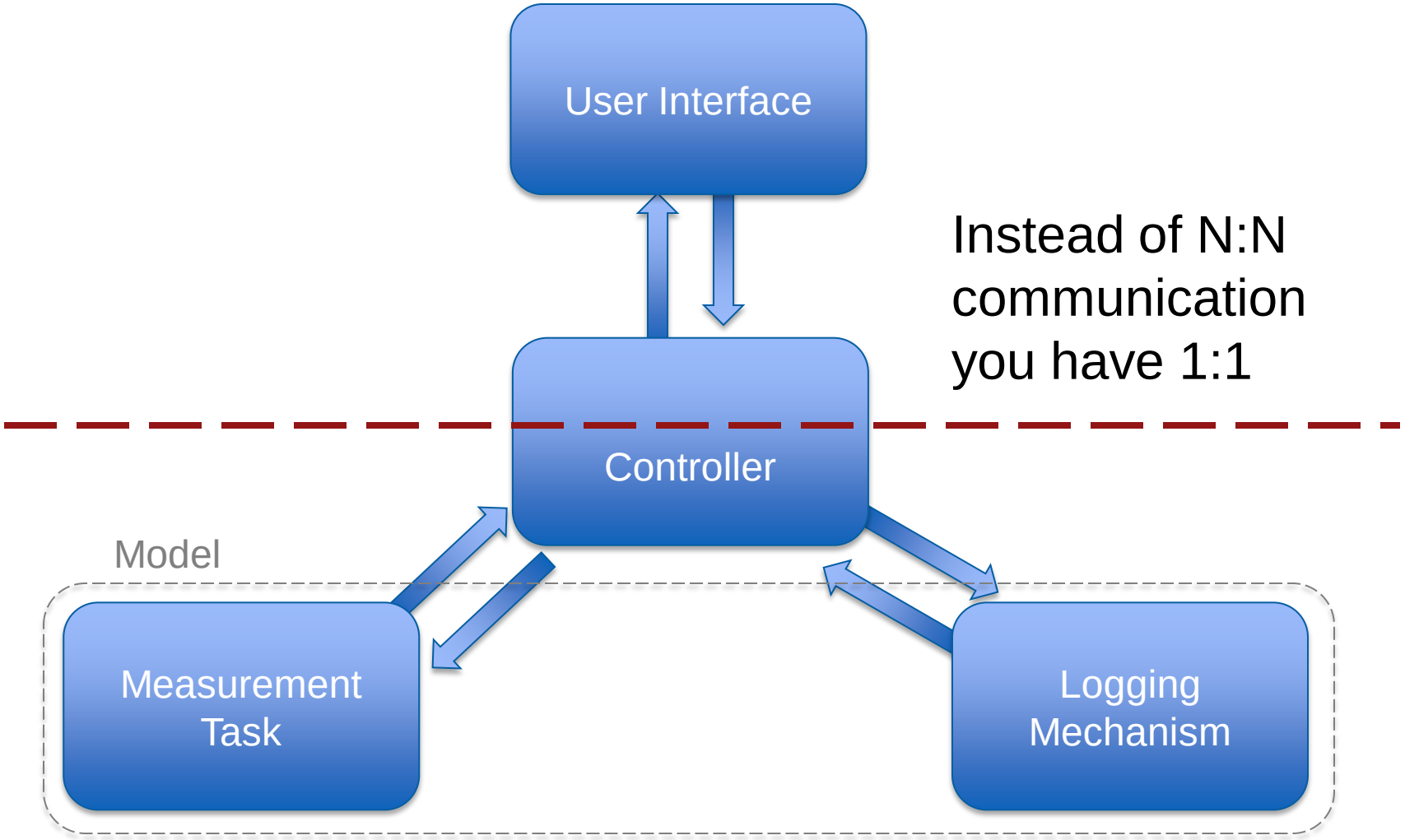
Stop.lvclass stops the receiving Actor

Last Ack.lvclass is the last message an actor sends to its caller

Typical Measurement Application



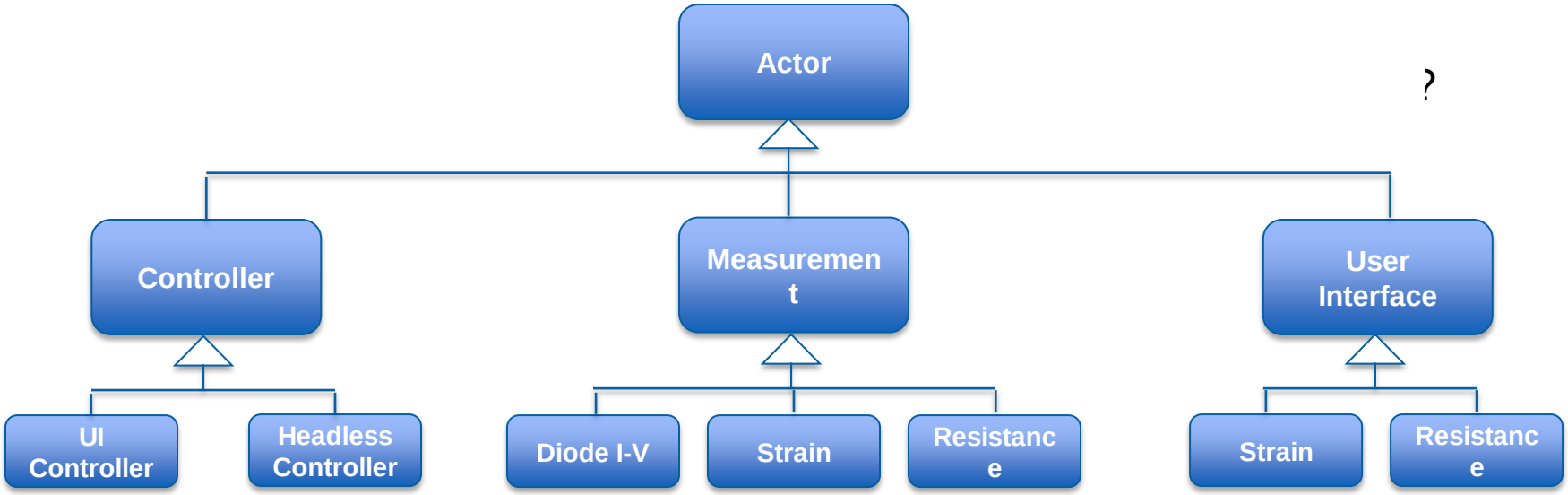
Model, View, Controller Approach



Instead of N:N communication you have 1:1

Model, View, Controller Approach

Class hierarchy view



- Changing the functionality of our MVC is simplified thanks to the use of OOP
- Children easily extend basic functionality for specific functions

Demo

Creating an Actor

Creating a Message

For Best Results

- Announcement messages, not request messages
- Embrace peer-to-peer messaging
- Synchronous messages can be created at your risk