Event Source Actor

Subscriber – Publisher in Actor Oriented System

Piotr Kruczkowski CLA

Software Solutions Developer

Agenda

- Introduction
- Reusability
- Zero coupling messaging
- Event Source Actor
- Use cases
- Tutorial
  - Creating Event Source with Events
  - Creating Implementations for Events and using Event Source
  - Integration
- Details
- Q&A

Introduction

- Many actor systems need to break the tree hierarchy for communication
- This can introduce difficult and strange practices, connections
- Needs to be tracked and debugged
- Should be standardized
- Should be extensible and decoupled

- Event based approach is proven to work

Reusablity

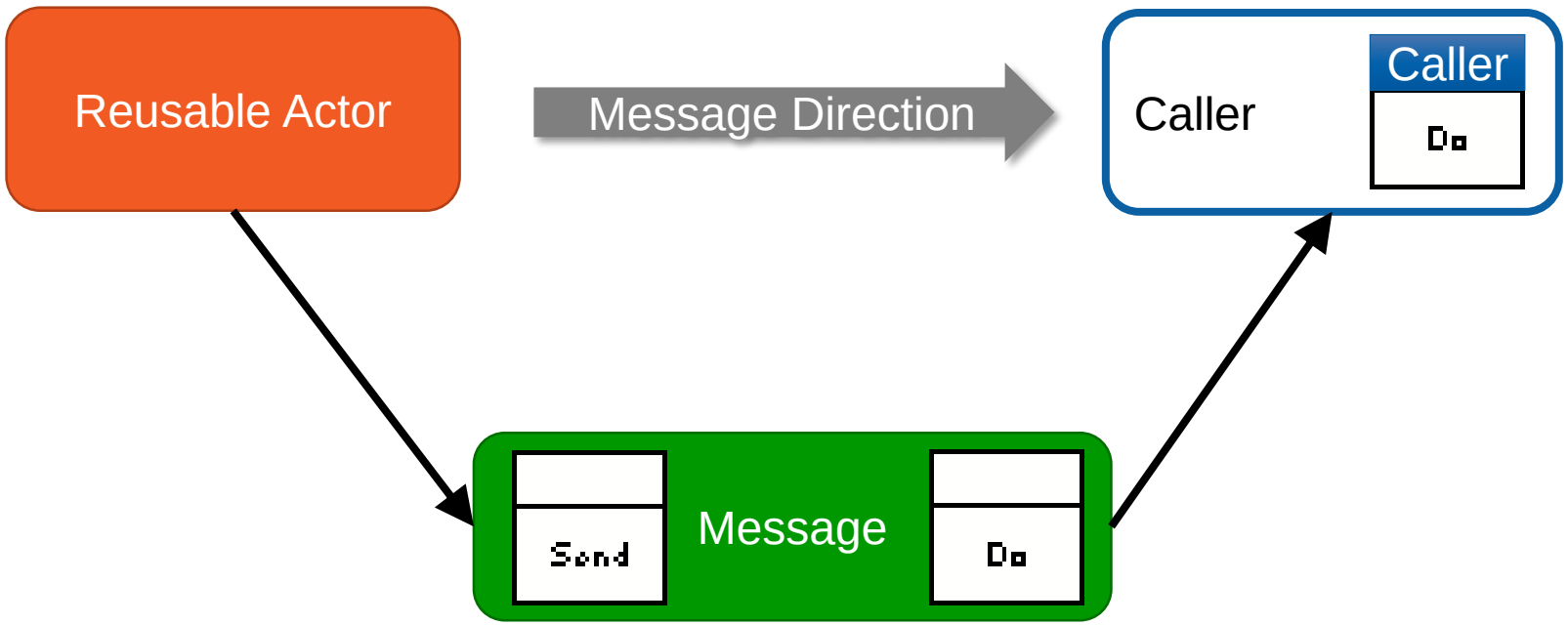
- Limited with coupling
- Only loading what is needed
- Reusable system needs to have all interfaces defined and understood
- Interactions between modules should be limited to usage of interfaces
  - Calling methods
  - Messages
  - Inheritance and override
  - Plugins

Zero coupling messaging

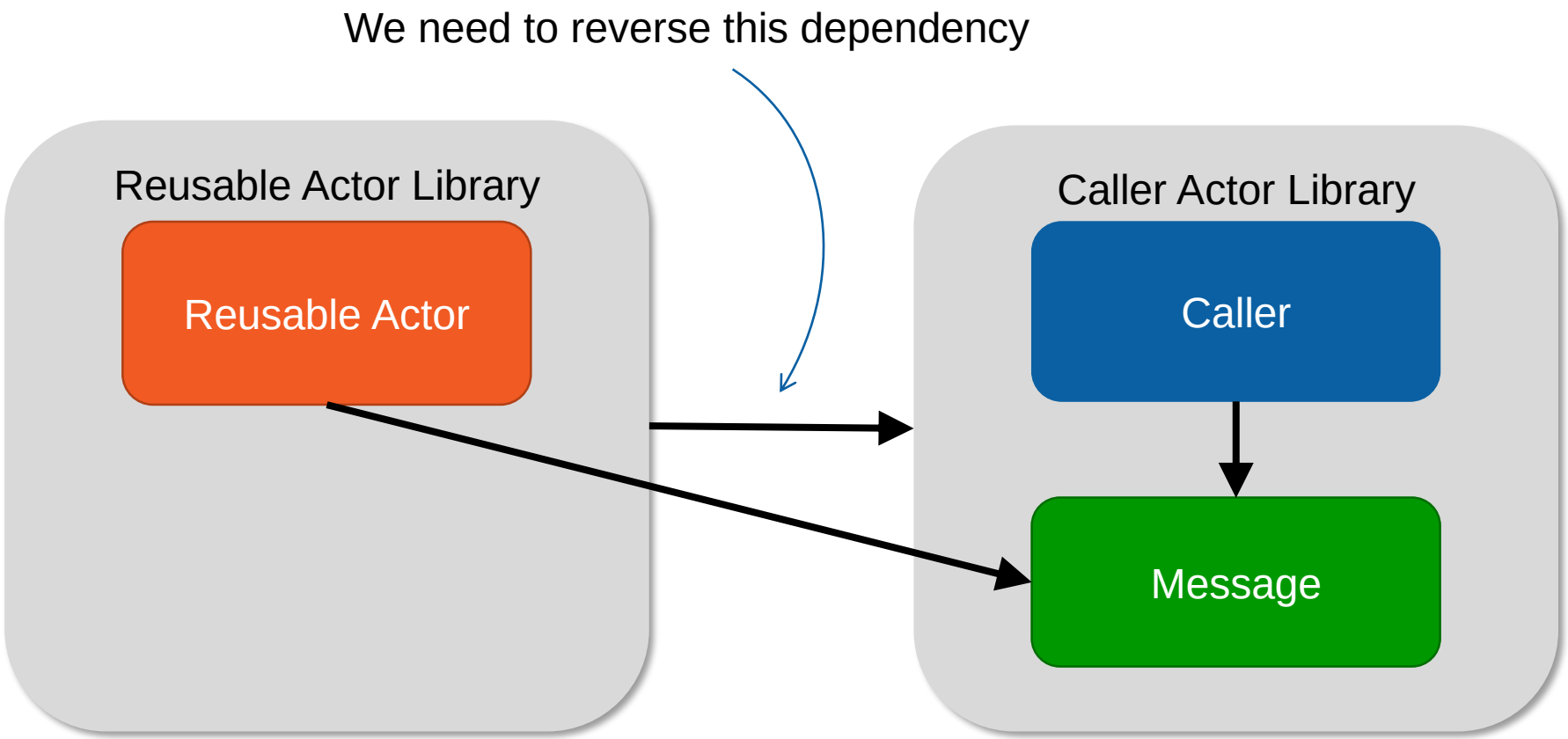
- The message defines the functionality
  - Messaging actor B from actor A loads actor B as actor A dependency
  - This causes tight coupling
  - To solve it you need an interface that will be loaded instead of actor B
  - You then implement concrete methods using the interface

- Requirement for real in AOD
- Prevents dependency linking
- Provides a clean interface
- Removes coupling

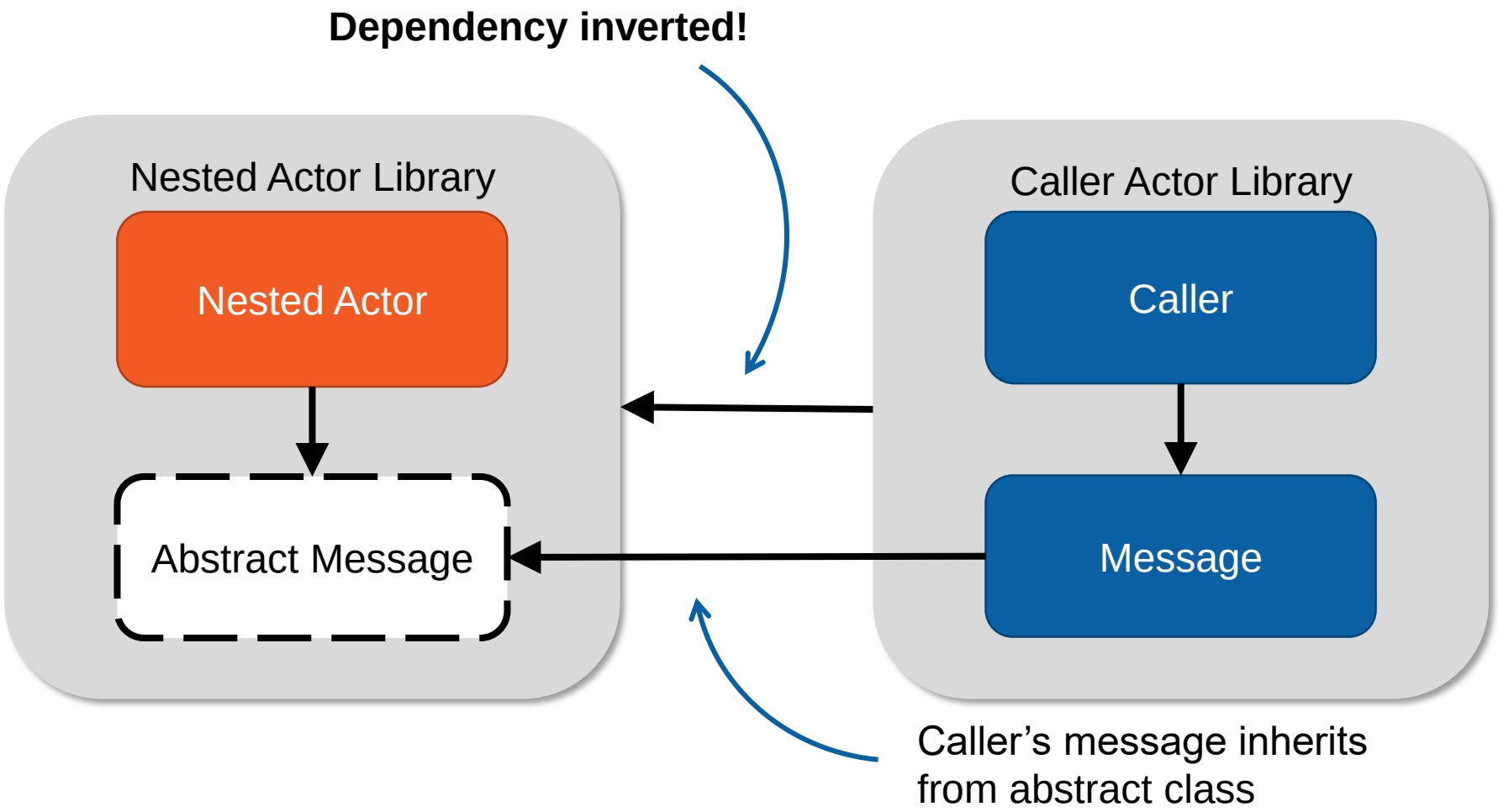
The Problem Is…



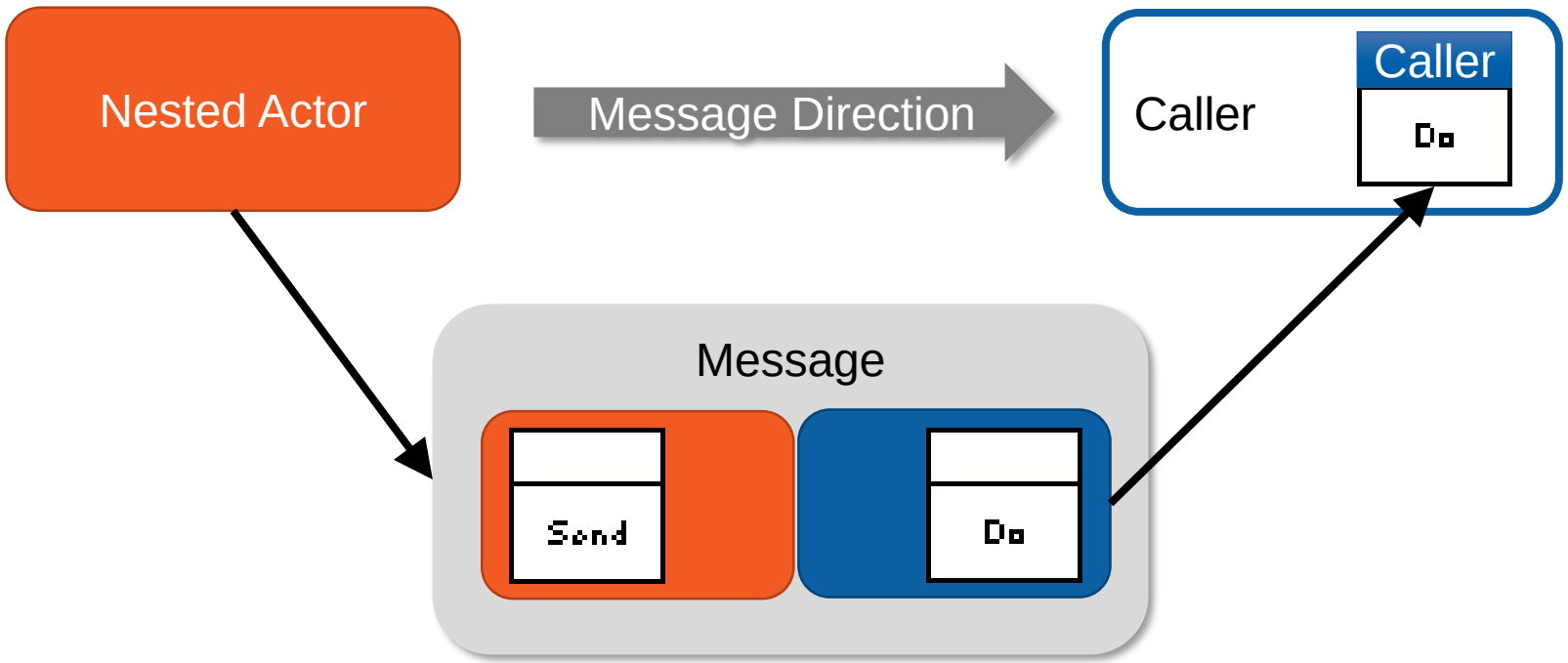
Dependency Inversion



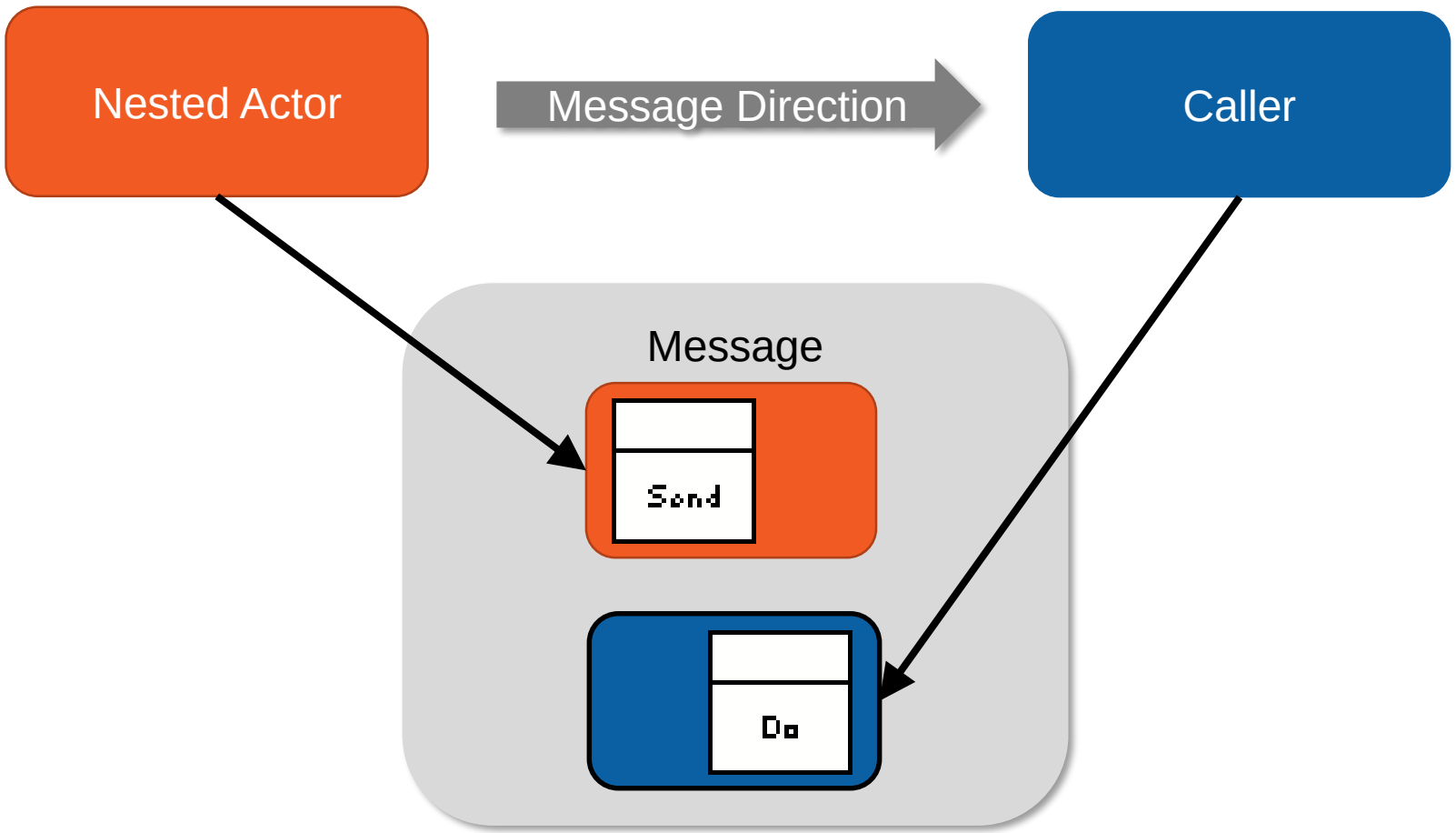
Dependency Inversion



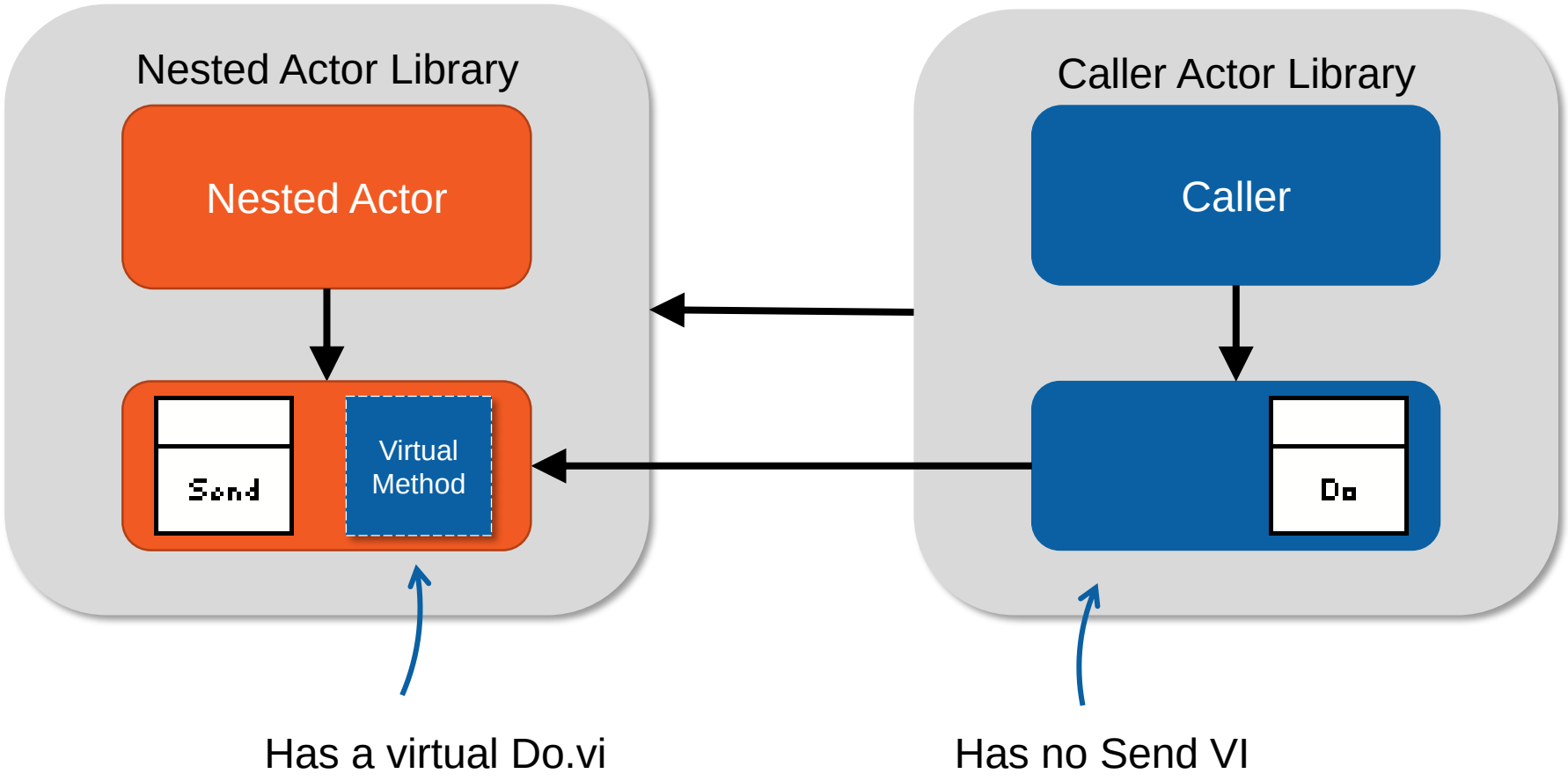
Zero Coupling



Zero Coupling



Dependency Inversion



Zero coupling messaging

- The message defines the functionality
  - Messaging actor B from actor A loads actor B as actor A dependency
  - This causes tight coupling
  - To solve it you need an interface that will be loaded instead of actor B
  - You then implement concrete methods using the interface

- Requirement for real in AOD
- Prevents dependency linking
- Provides a clean interface
- Removes coupling

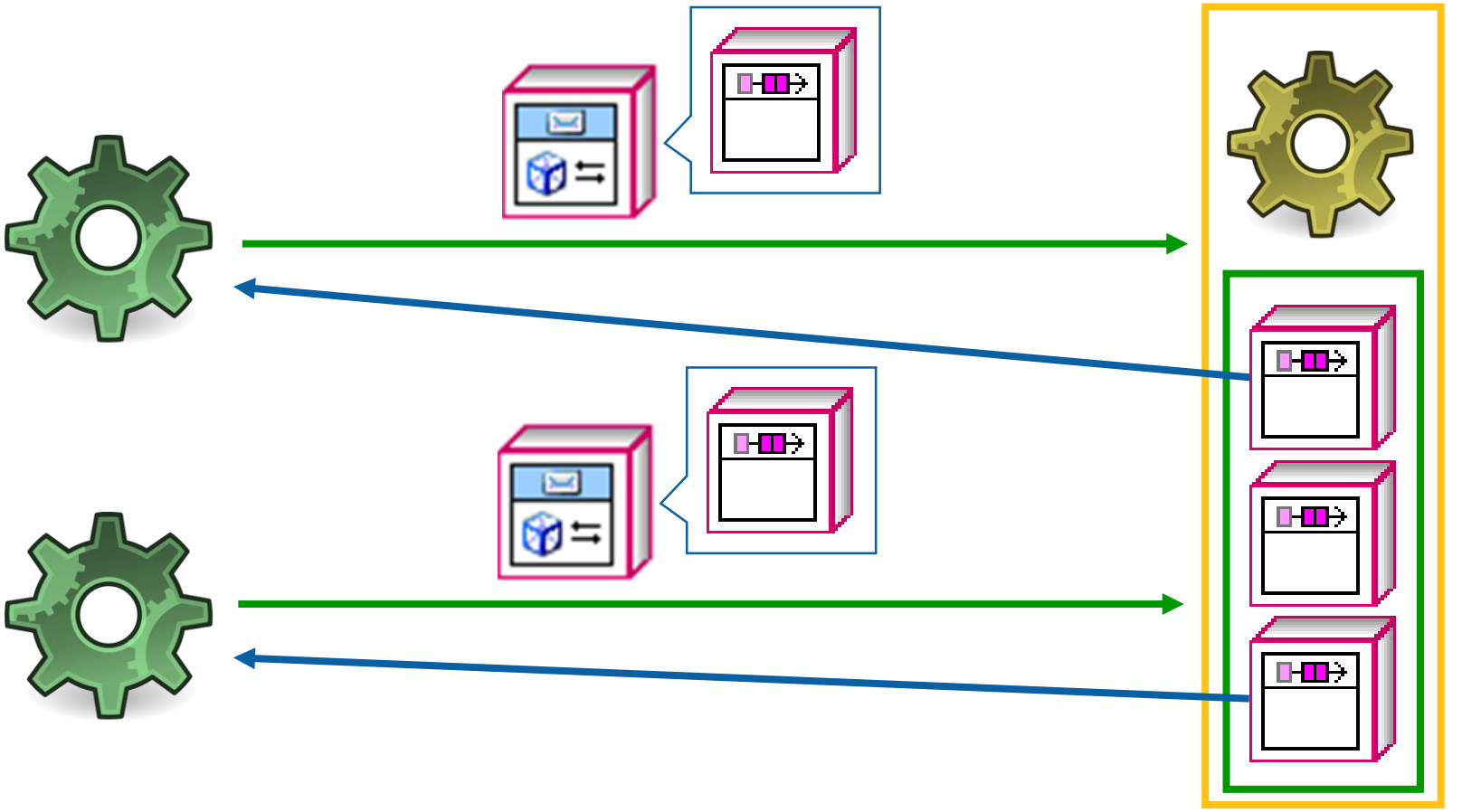
Subscriber Publisher

- Actors subscribe to receive data
- Event Source Actor only works with subscribed Actors
- Event Source Actor launched before Actors that Subscribe
- Event driven behavior is the AOD guiding principle

- Reacting gracefully to Actors being stopped - Unsubscribing
- Sharing the address by Subscribing
- Benefits of direct communication
- Plays nice with zero coupling

- Tree hierarchy is broken
- Debugging techniques become more important i.e. DETT

Subscriber - Publisher

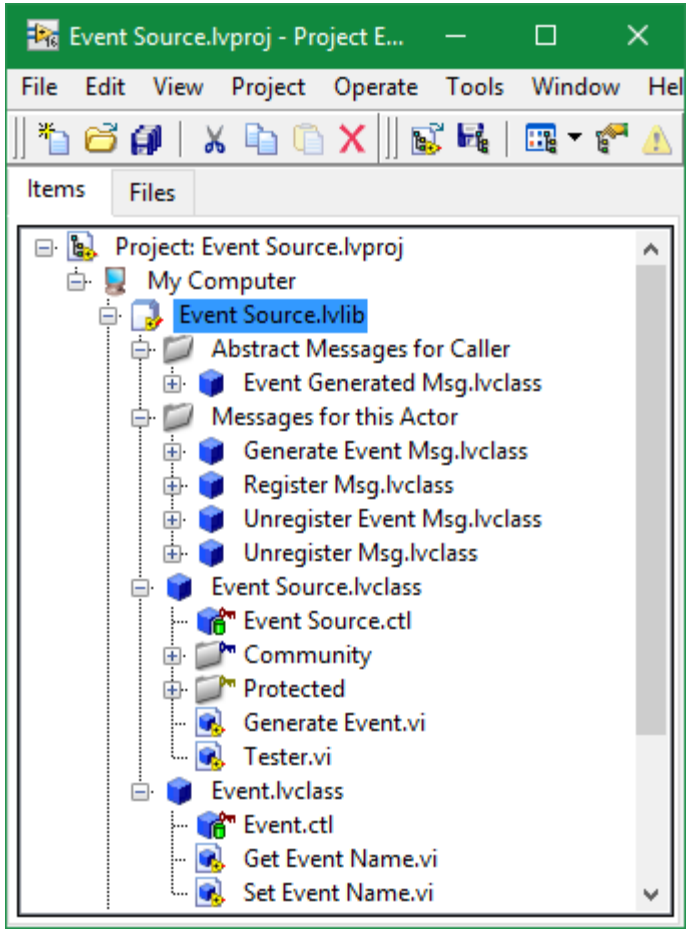


Event Source Actor

- Subscriber – Publisher pattern
- Actors register to receive event notifications
- Actors unregister to stop receiveing the updates

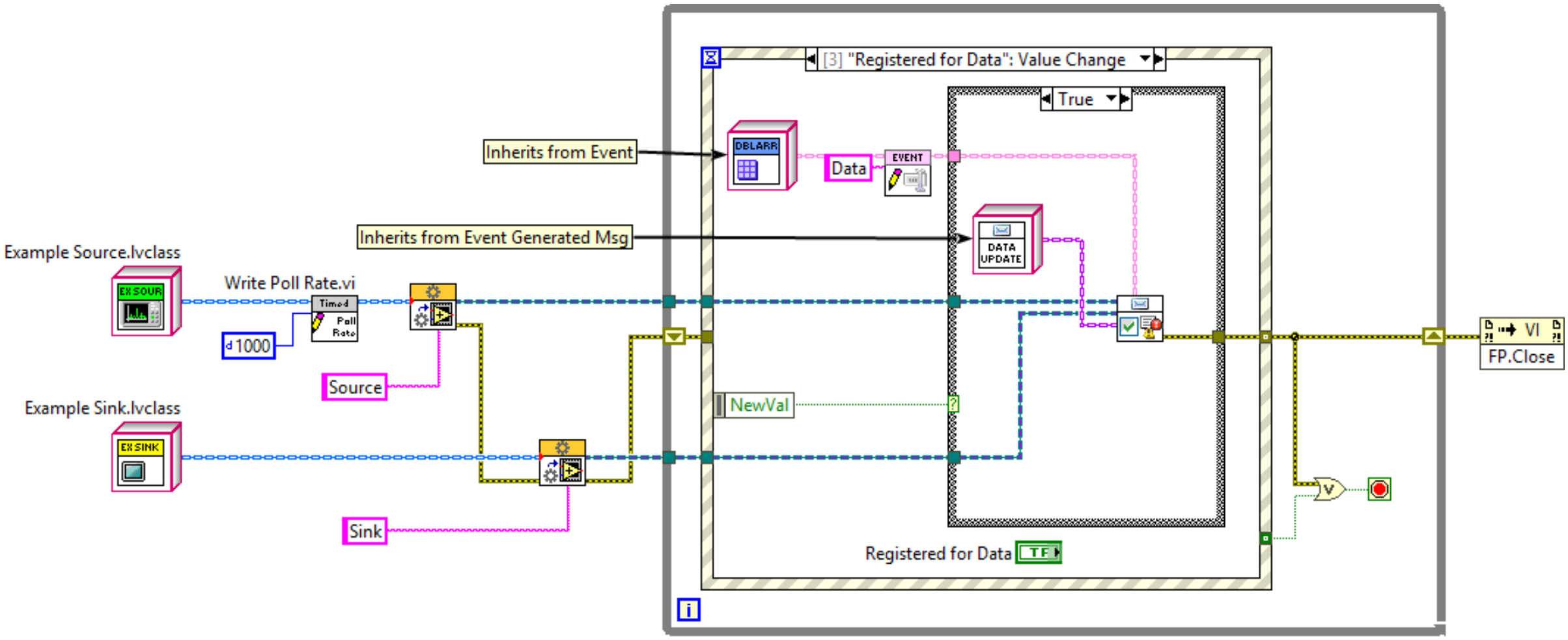
- Standardized mechanism available through inheritance
- Any actor can become an Event Source Actor and gain all the abilities of Event Source

- Event Source Actor generates events on his own schedule
- All subscribed actors get notified to handle the event with their own implementations



NATIONAL INSTRUMENTS

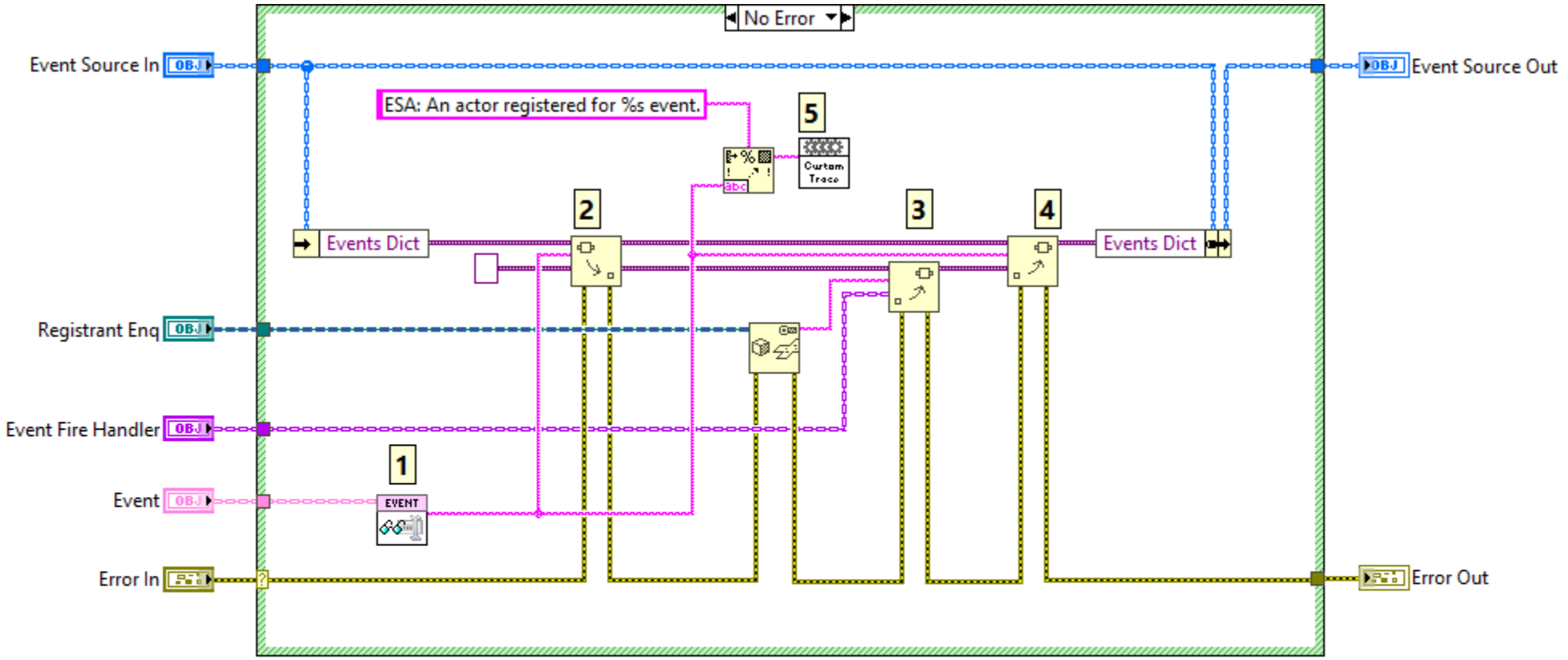
Custom Application - Send Register Message



The source will periodically receive UpdateMsg. Handling that msg fires two events to all registered actors.
The only registered actor is the sink. He provides his mechanism for handling the fired events durin registration.

The mechanism for handling is based on abstract Event Generated Msg defined by the Event Source. The only thin user has to do is create a specific msg based on that generic one.

Clockwork - Register Method

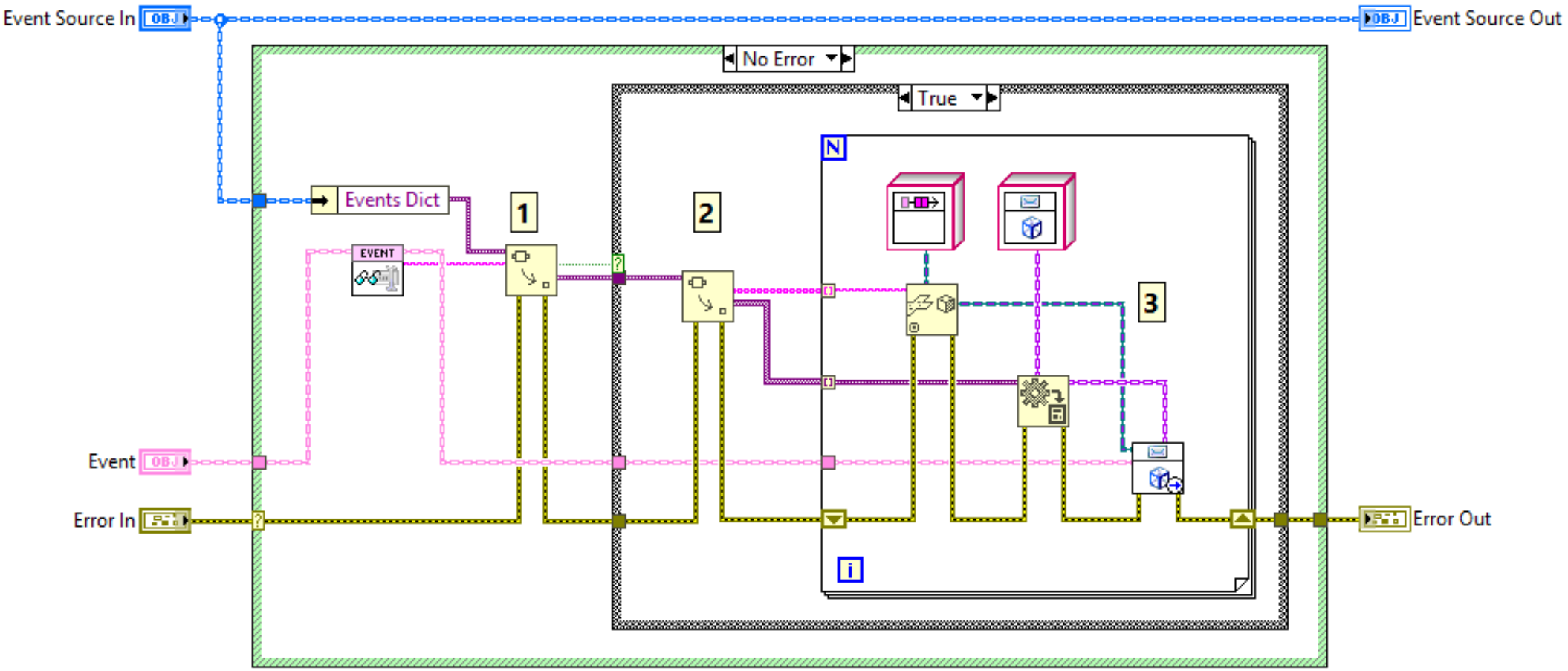


1. Get name of event for registration. The event name will be used as an entry in the dictionary.
2. Retrieve the dictionary entry for the event name. If no entry existed before an empty dict will be used.
3. Write the flattened enqueuer as unique dictionary entry and Zero Coupling msg as data.
4. Write that new dictionary into events dictionary as an entry.

5. Generate the registration trace for DETT if **AF_Debug_Trace** = **TRUE**

NATIONAL INSTRUMENTS

Clockwork - Generate Event



1. Read an events dictionary.
2. Read all registrants for this event.
3. Convert the keys and values to enqueuers and Zero Coupling based Event Fired Msgs and send the event data to registrants.

NATIONAL INSTRUMENTS

Event.lvclass

- Defines the format of data of the event
- Base class has only a name
- Subscriptions are based on the name of the event

- Usually just a getter and setter are needed in children

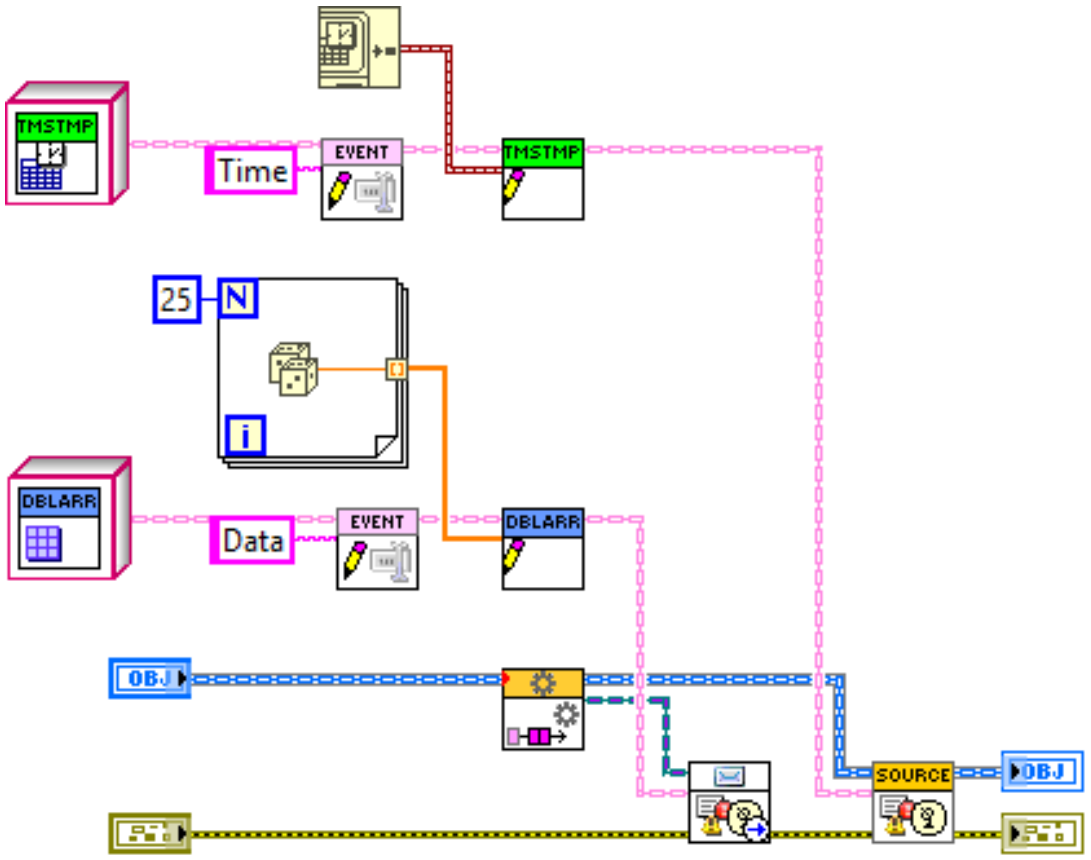
Q: Why not directly use ZC messages? Why add another layer of events?

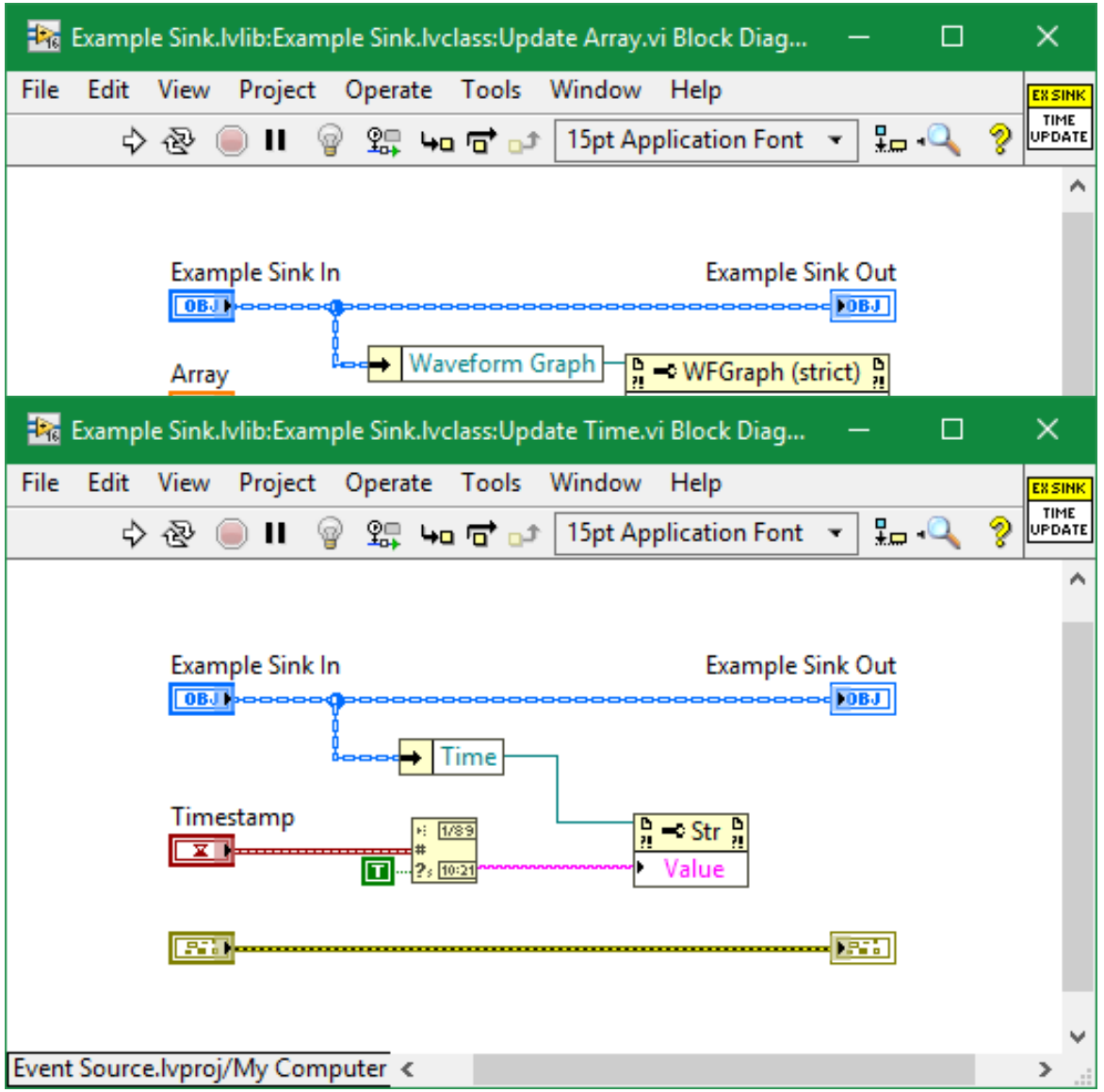
A: Because the publishing mechanism needs to have an ability to pass messages to all registrants based on event name, therefore the data type has to be unified and one interface needs to be used.

Q: How are events different from messages?

A: Messages include sending and doing functionality besides defining the data type. Events only define the data type. **Sending** is done by event actor, and **doing** is done by registered actor. The abstract message is the Event Generated.lvclass

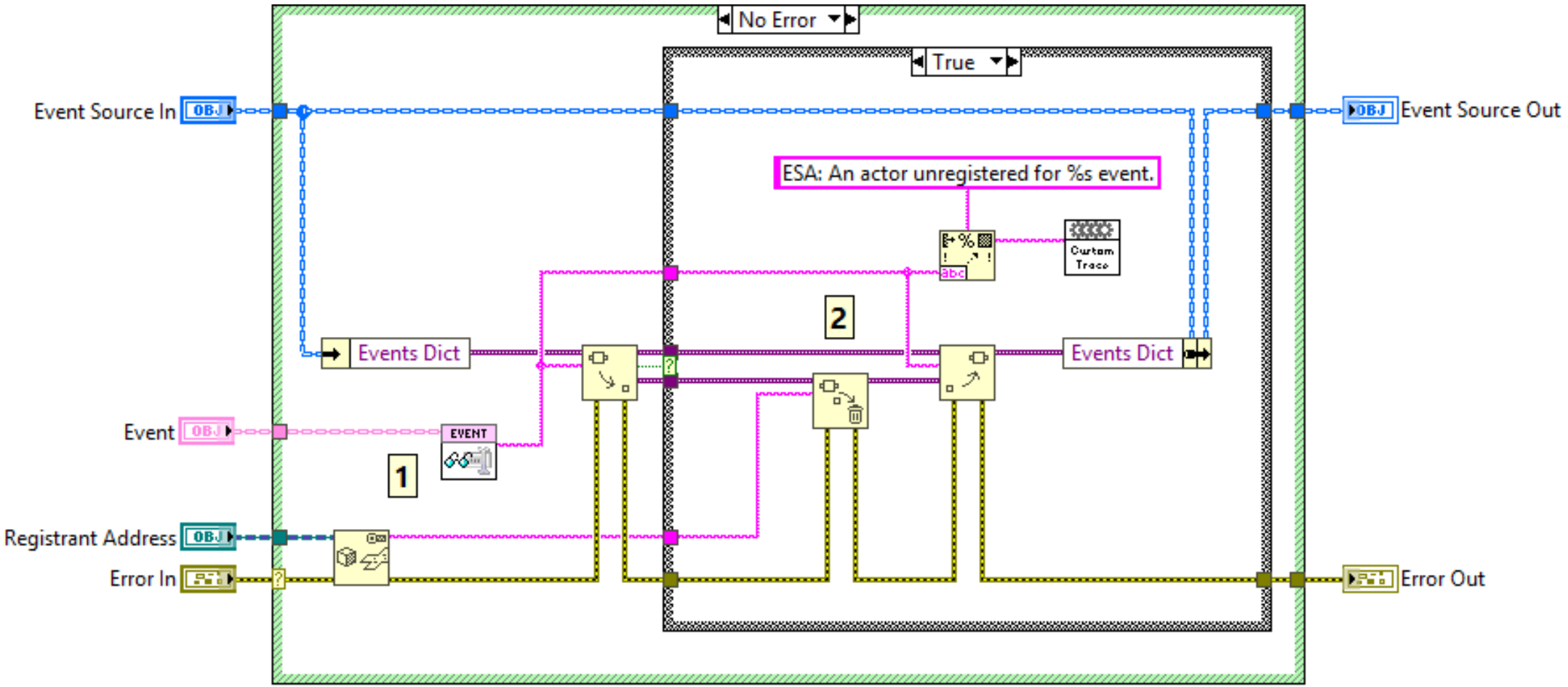
Custom Application – Generating and Handling Events





NATIONAL INSTRUMENTS

Clockwork - Unregister Method



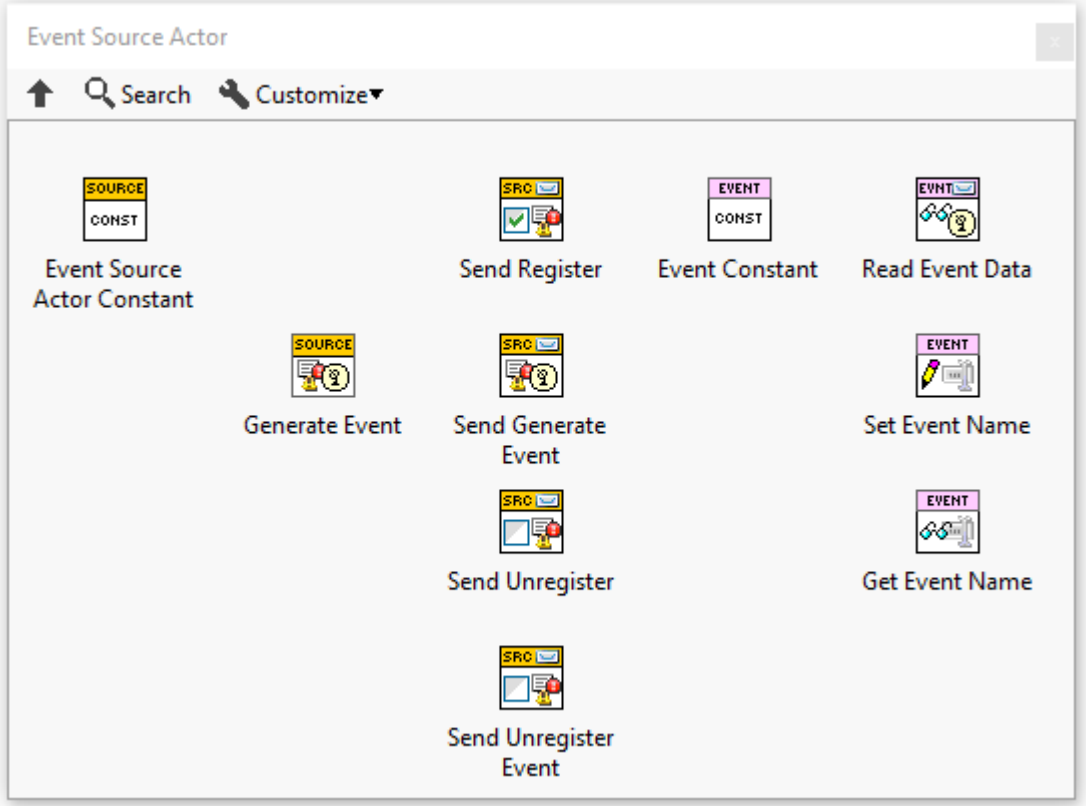
1. Use event name to look up in the dictionary the dictionary for this event.
2. Remove a specific enqueuer in the spcific events dictionary.

NATIONAL INSTRUMENTS

Event Source Actor Package

- VIPM package
- Includes the example shown
- Includes the palette to simplify working with ESA

- Available @ni.com



Integrated Solution

Use cases

- Measurement complete events in DAQ tasks
  - Data ready event
- User clicked event in UI tasks
  - Button clicked event
- Long task completed in Processing tasks
  - Processed data event

- Any actor oriented communication outside the tree comms hierarchy

Creating Event Source with Events

1. Create or select the class to become **Event Source**
2. Add **ESA** to data of your class or make the class inherit from **Event Source Actor**
3. Create the event classes that your **Event Source** will generate
   a) Event classes define the event type and its data
   b) Create a **constructor/destructor** or **getter/setter** for the data in your events
   c) Make them inherit from **Event.lvclass**
4. Add calls to **Generate Event Or Send Generate Event Msg** method inside the functions of **Event Source**, whenever an event is to be generated
   a) Write the data using a setter into your generated event object, if you want registrants to receive it
5. If an event is generated in a helper loop, send **yourself (or your composed ESA)** the **Generate Event Msg** instead (so that the event is generated from inside the Actor Core, and the registrants can be notified)

- Any actor that registers to this event will be notified whenever its generated

Creating Event Source with Events

Creating Implementations for Events and using Event Source

1. Create messages from **abstract message** based on handler methods
2. Select the **Event Generated Msg** as the abstract message you want to inherit from
3. Use **Read Event Data** method and provide a translation of the data in the **Do.vi**, from the event to the input of the called method
    a) If the event has a destructor returning data or getter method, you can use it here

Creating Implementations for Events and using Event Source

Integration

1. Instantiate and run both event source and event registrant actors
2. Register for events from registrant actor
3. Generate events in the source
4. See that your registrant handles events correctly
5. Unregister when finished or when stopping

Integration

Details

- Event Source Actor uses dictionaries to keep track of registrants
- Class names are used as the keys in the dictionary