PUSHING THE LIMITS OF LABVIEW

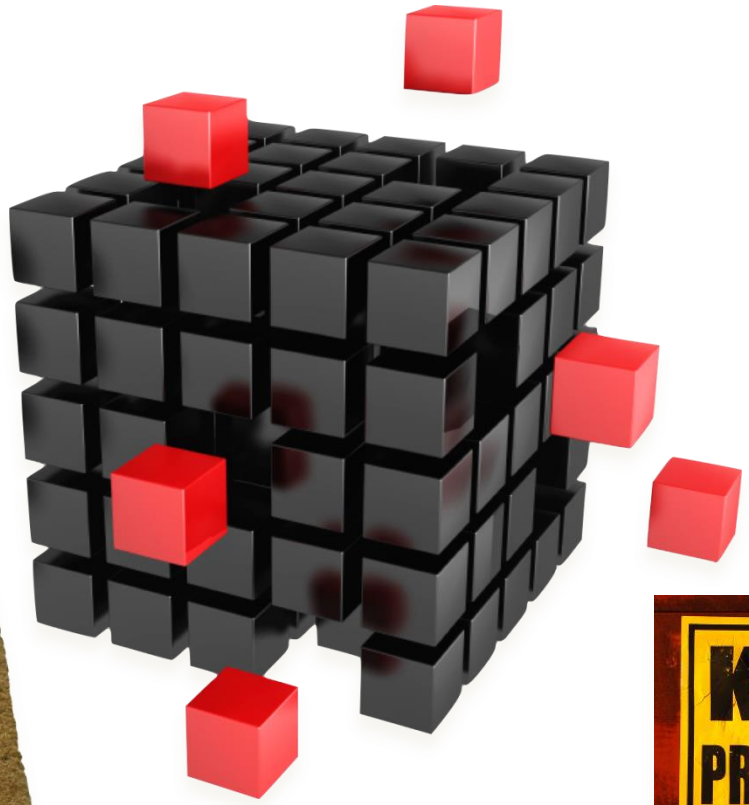
Public Events for “Modules”

A form of Interprocess Communication in LabVIEW

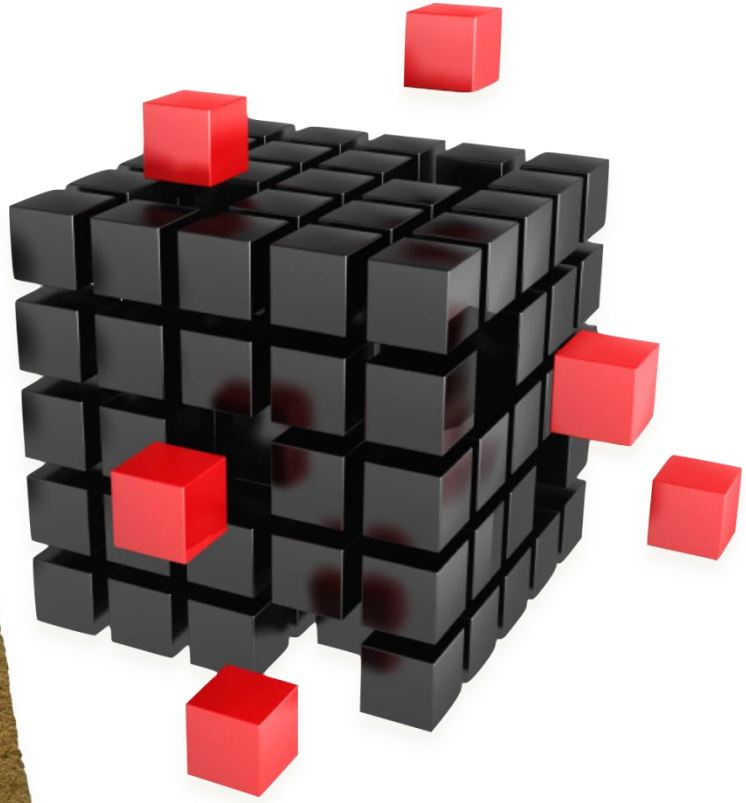
Jim Kring, JKI

at the European CLA Summit 2013

A Public & Private Events Framework



Public Events for Modules



The Scenario (Use Case)

- An Active Object with a process is
  - doing things asynchronously and
  - generating (producing) associated data
- One or more processes needs to
  - make synchronous calls into the active object
  - know when an event happens, and
  - get (consume) the associated event data.

Presentation Focus

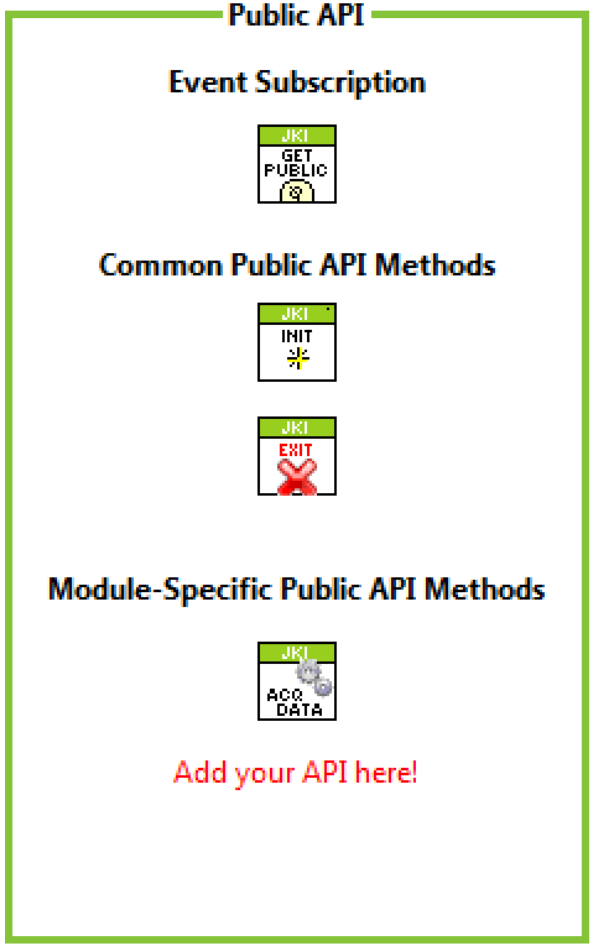
A Solution Based on User Events

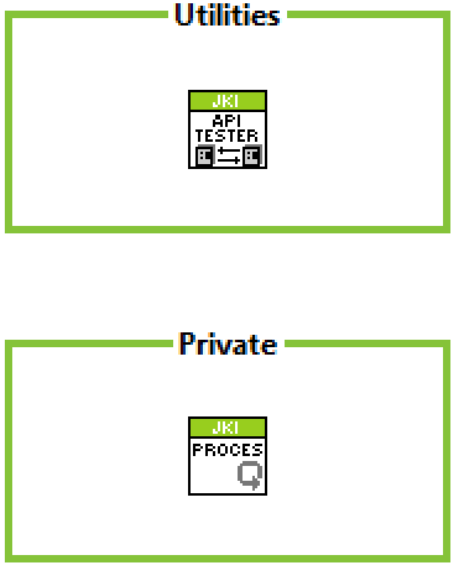
Let’s Get This Straight

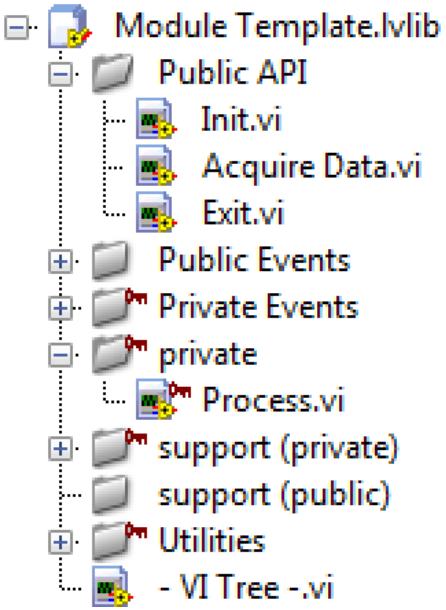
- User Events are a very easy-to-use feature with a lot of cool functionality.
- They form the basis of JKI’s primary application frameworks & templates.
- If we could get a couple things fixed/added to LabVIEW, we could do even better.
- Things keep getting better.

Let’s see some code!

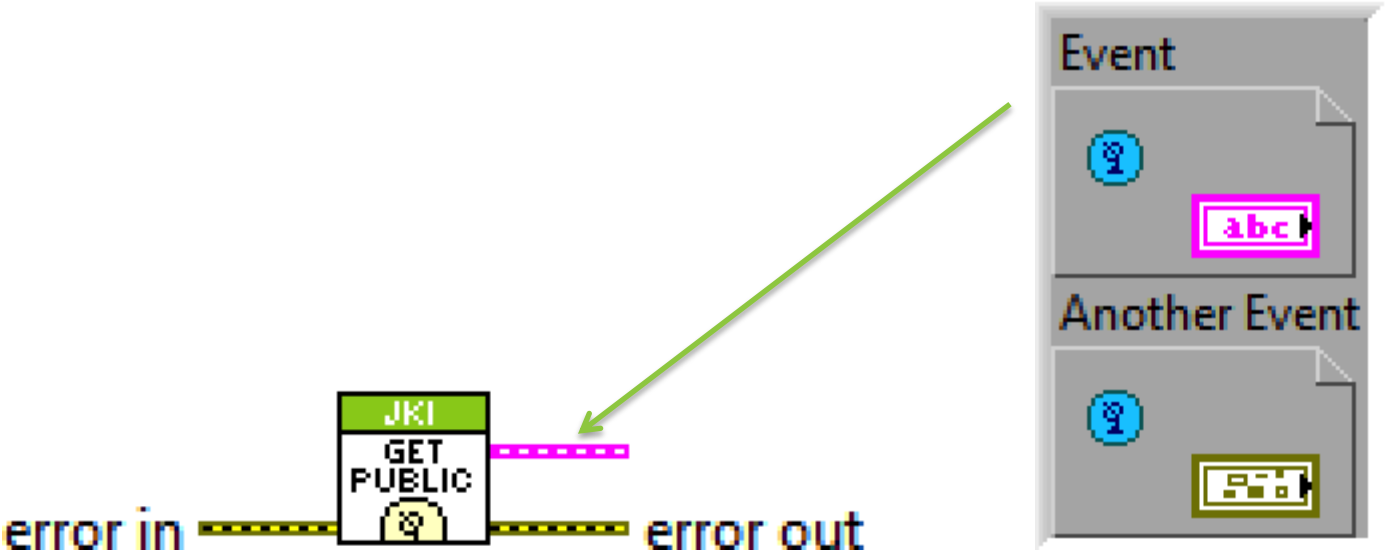
A “Typical” Module VI Tree







Get Public Events

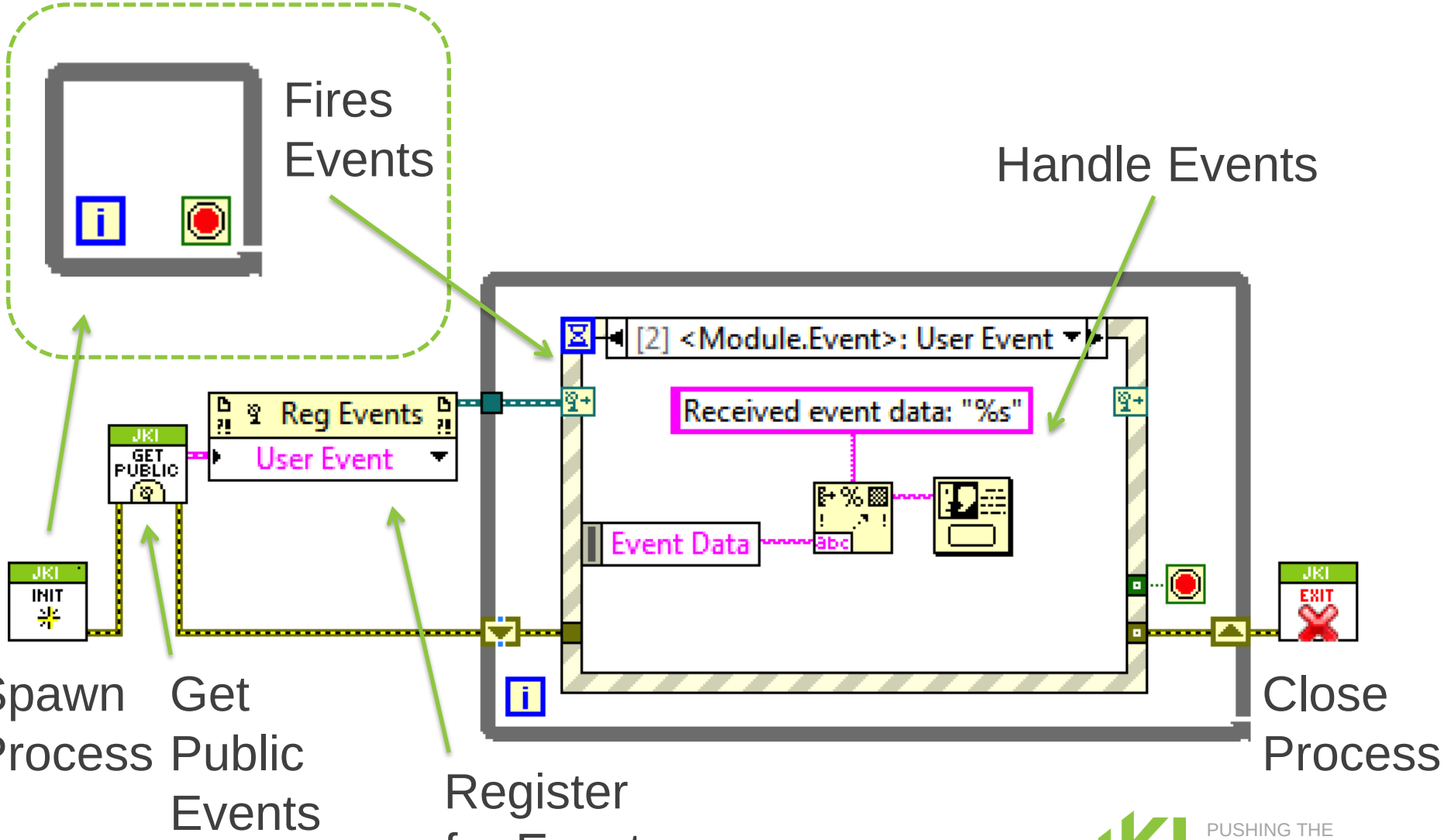


Get Public Events.vi

Public Events.ctl

Using public events is easy!

Using Public Events

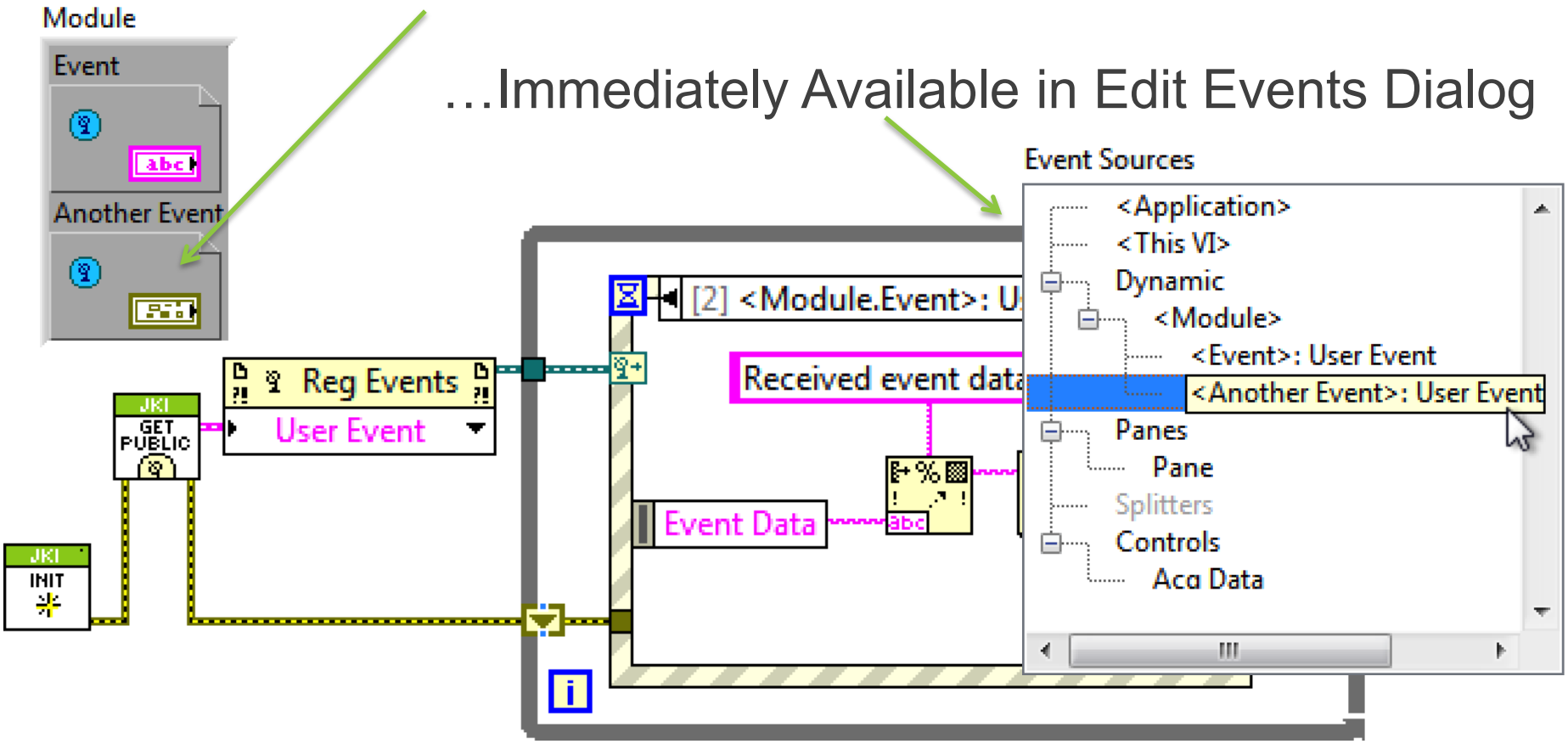


Code maintenance is easy!

Easy for Users: New Events

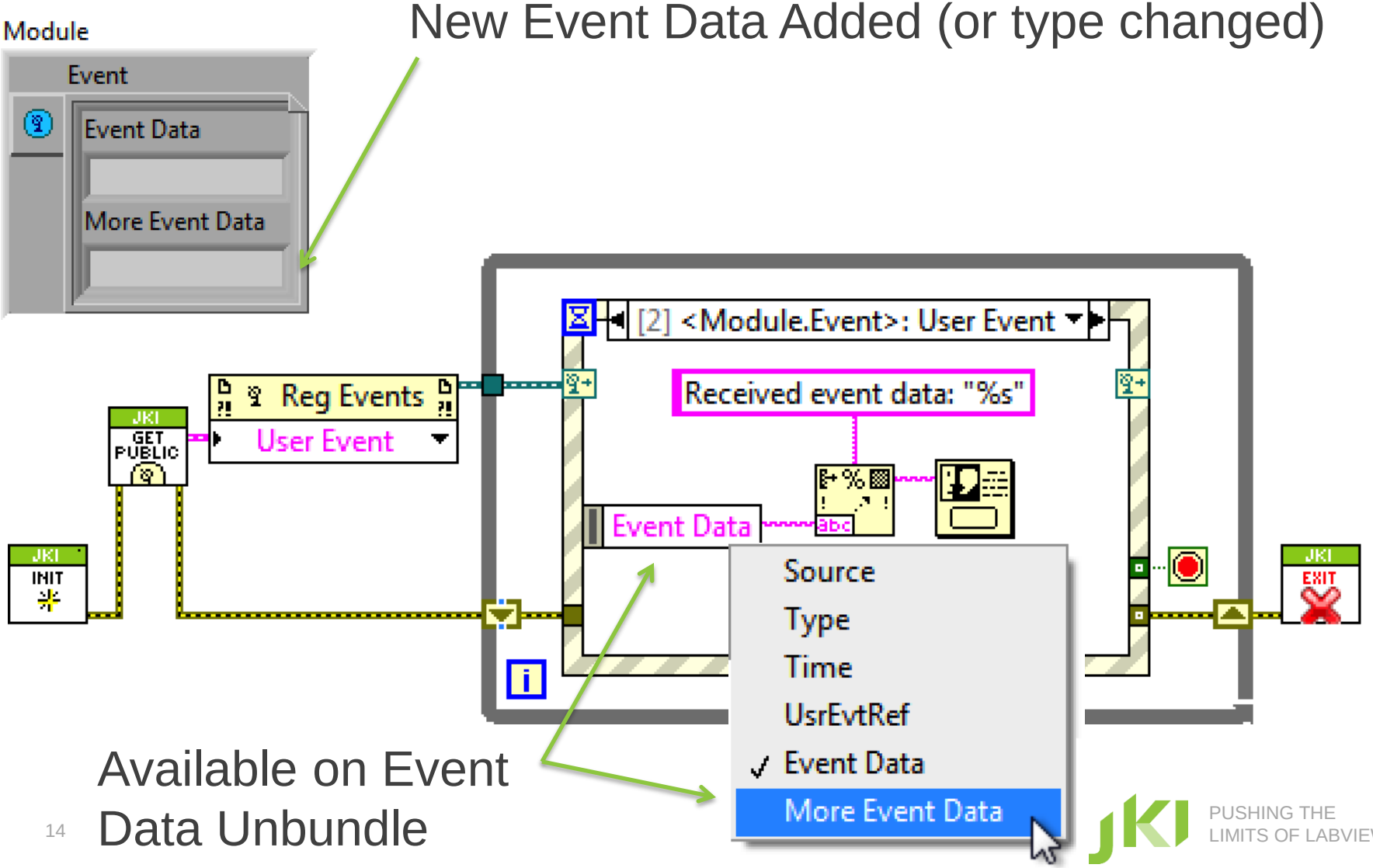
New Events Added to the Event Cluster…

…Immediately Available in Edit Events Dialog



Easy for Users: Event Data Changes

New Event Data Added (or type changed)



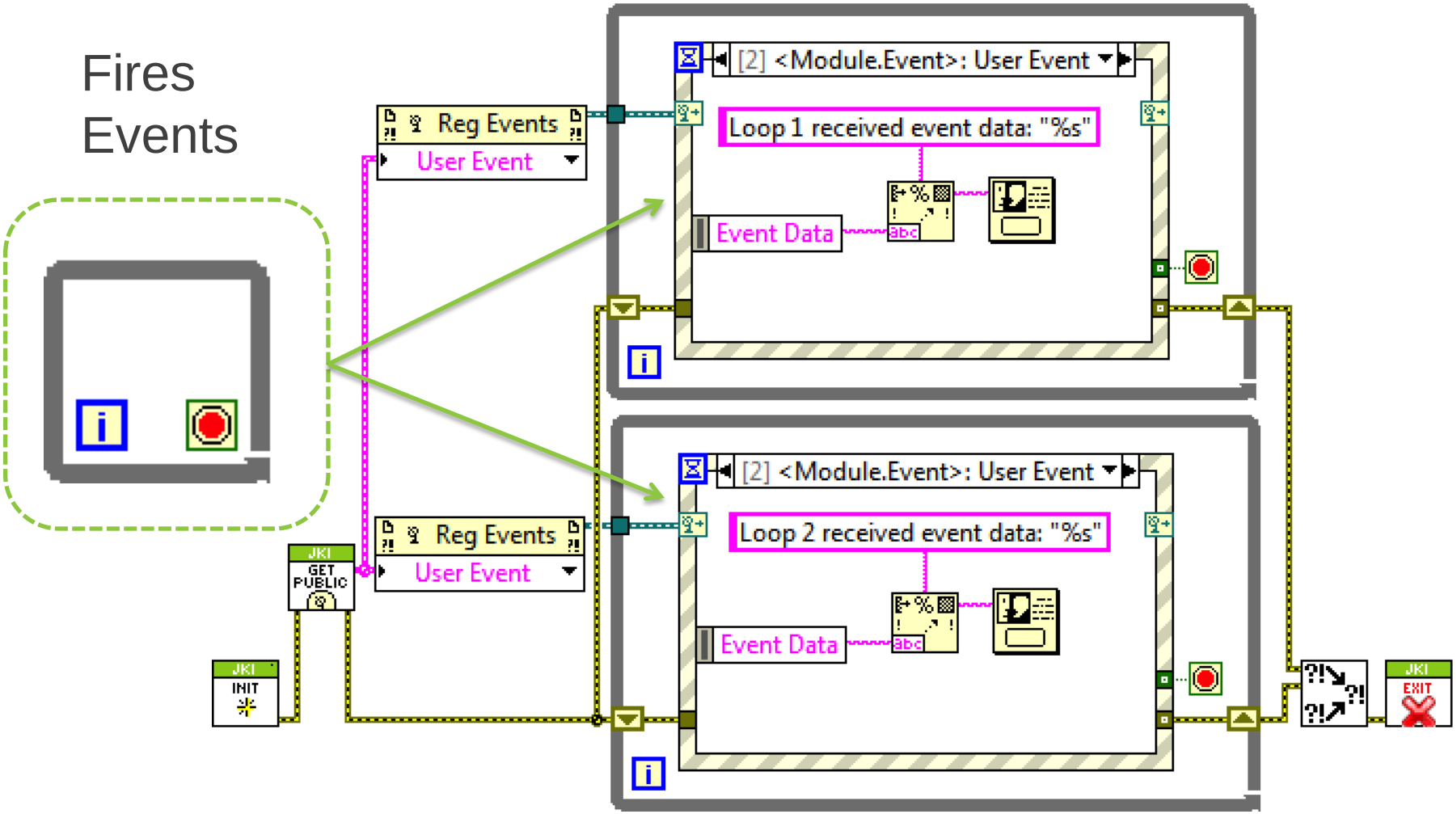
Available on Event Data Unbundle

One to Many

(multiple consumers)

Multiple Consumers

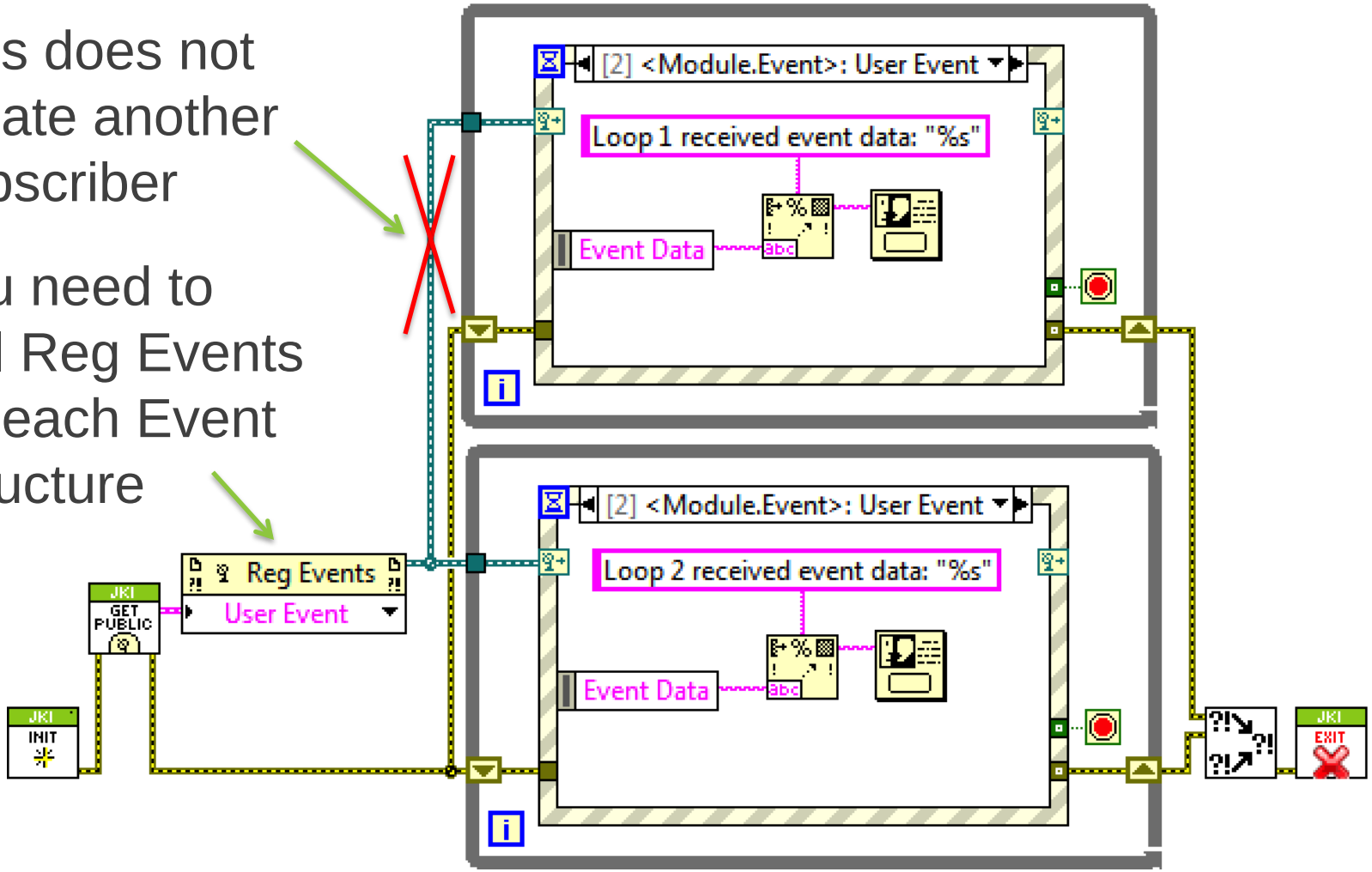
Fires Events



Don’t Fork the Event Registration!

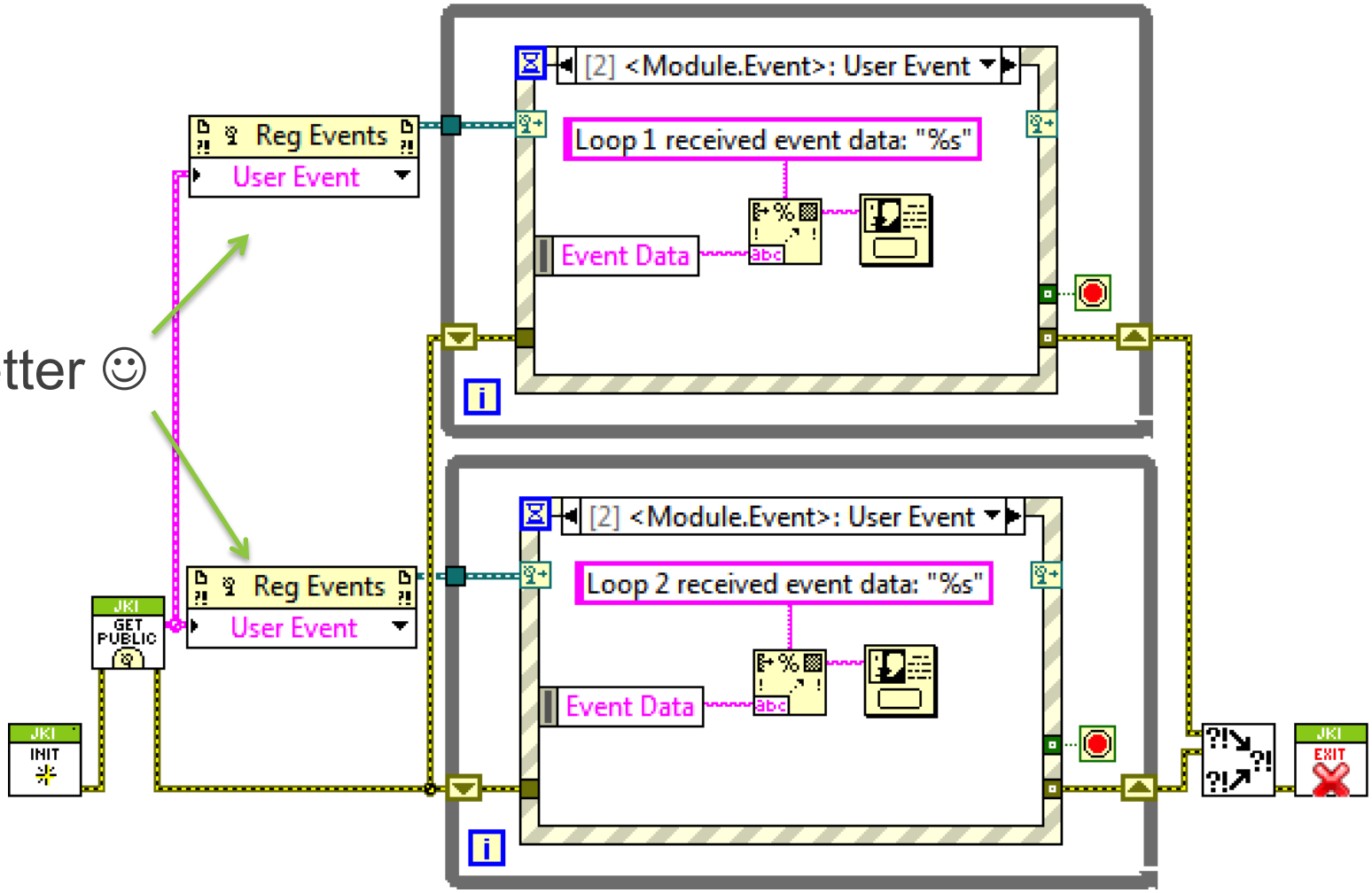
This does not create another subscriber

You need to call Reg Events for each Event Structure



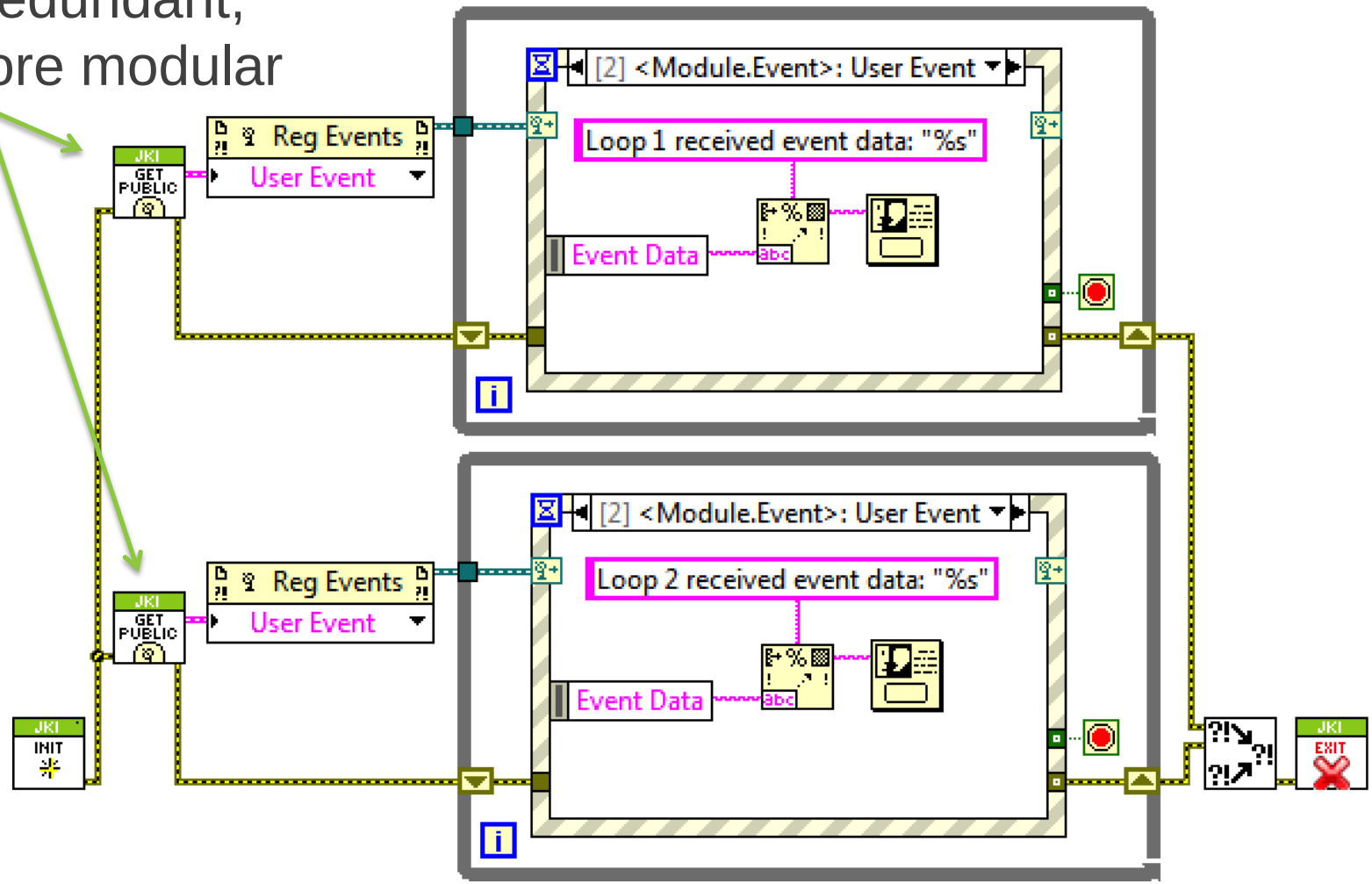
Multiple Subscribers

That's better ☺



Can Get Public Events Where Registering

Seems redundant,
but is more modular



Use in a state machine

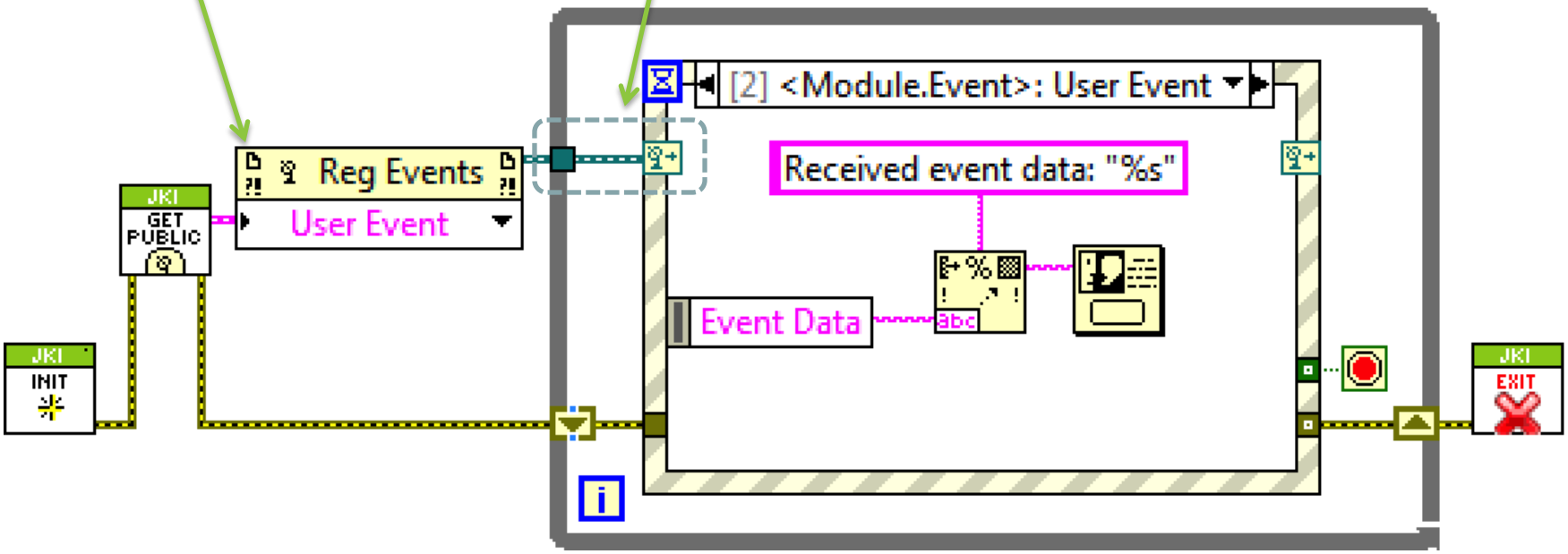
First, some gotchas

(les pièges)

Event Type Propagation Gotchas

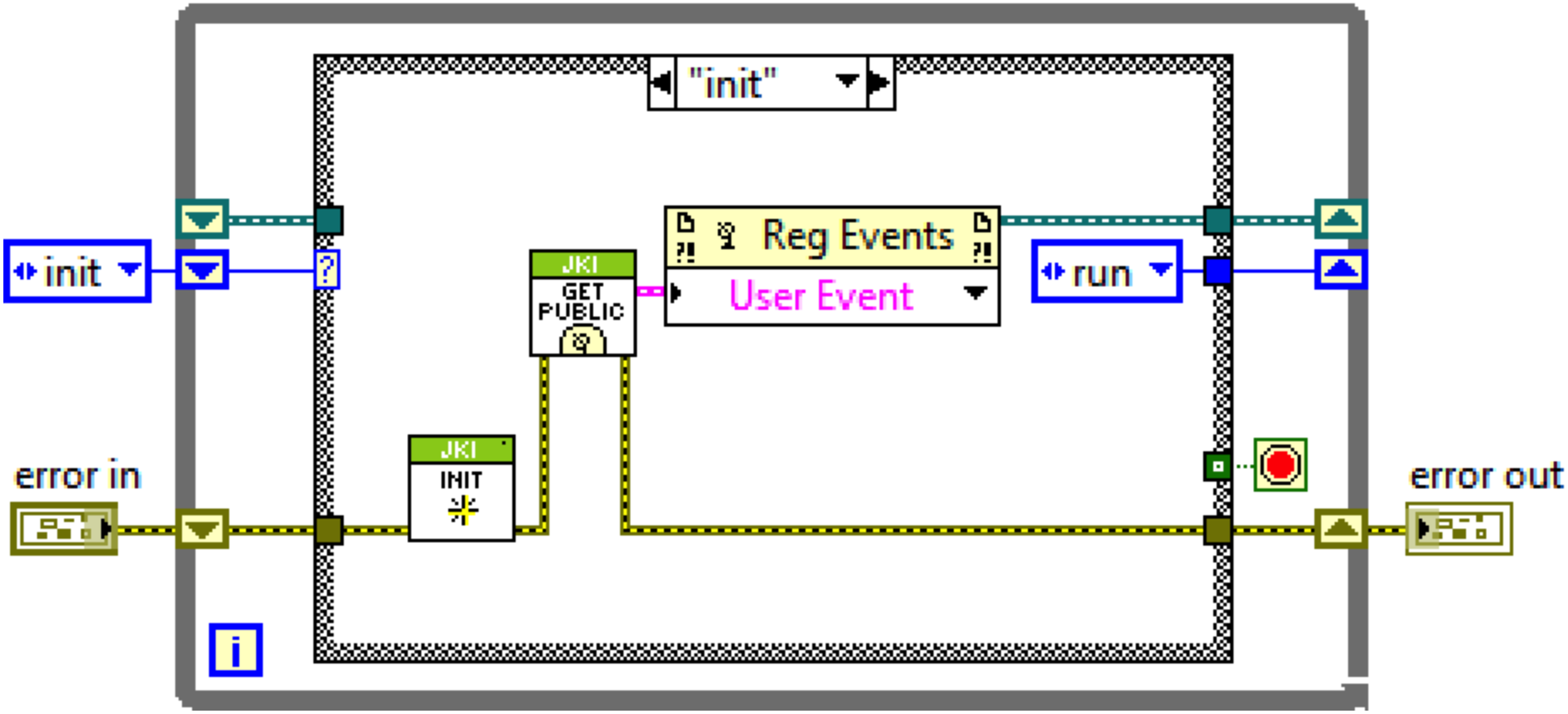
Don't wire input to Register for Events

Don't typedef (or in other ways "constrain") type propagation on this wire*

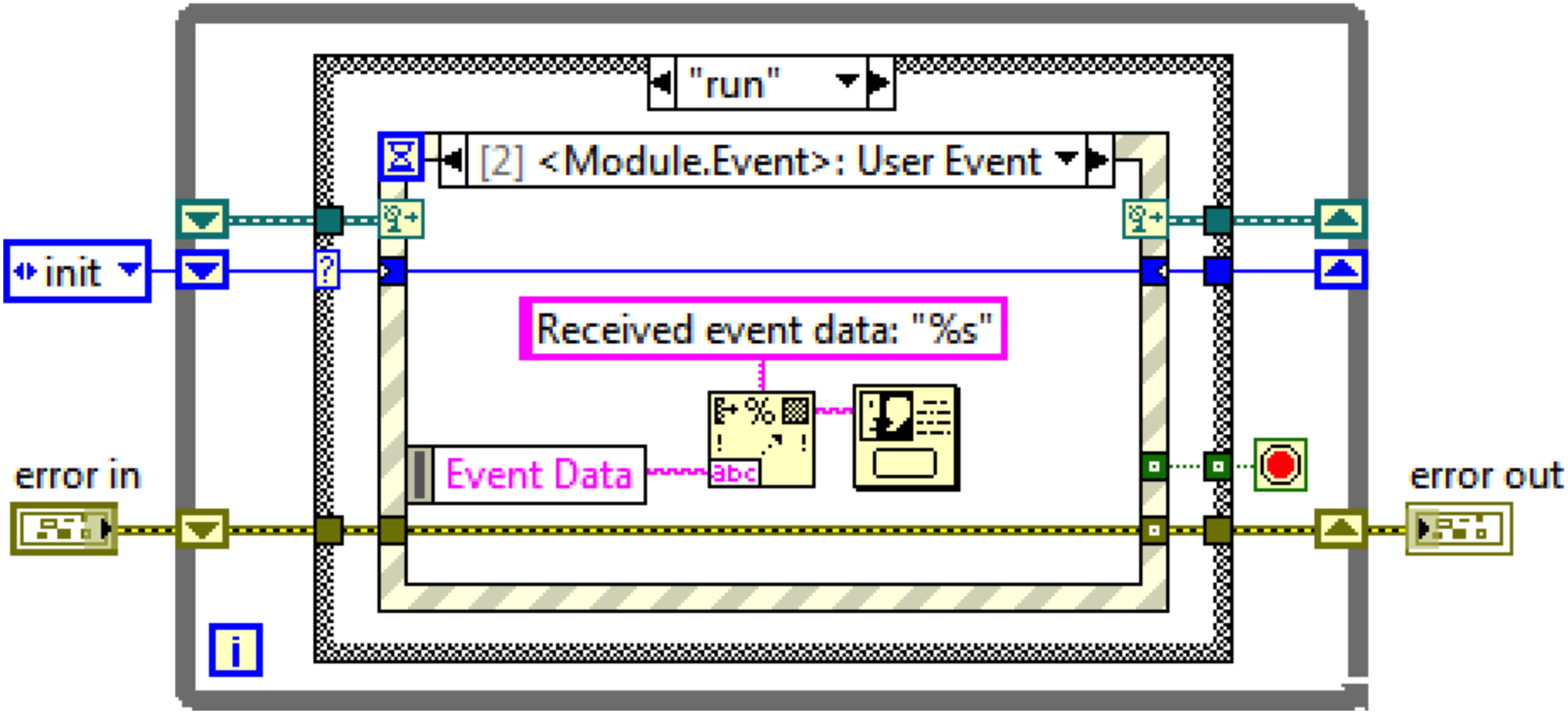


*Type must flow freely from Reg Events node to Event Structure's Dynamic Events terminal.

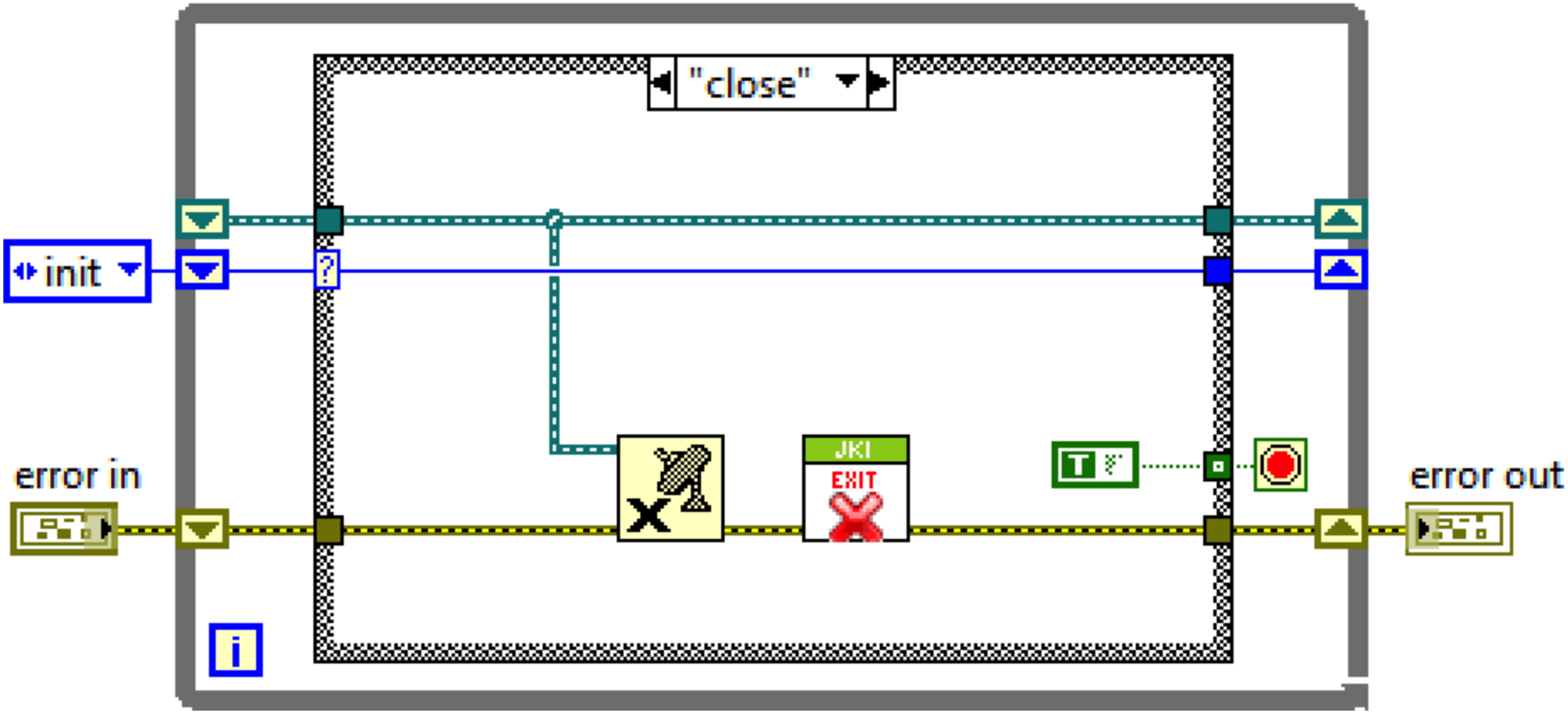
Initialize and Register for Events



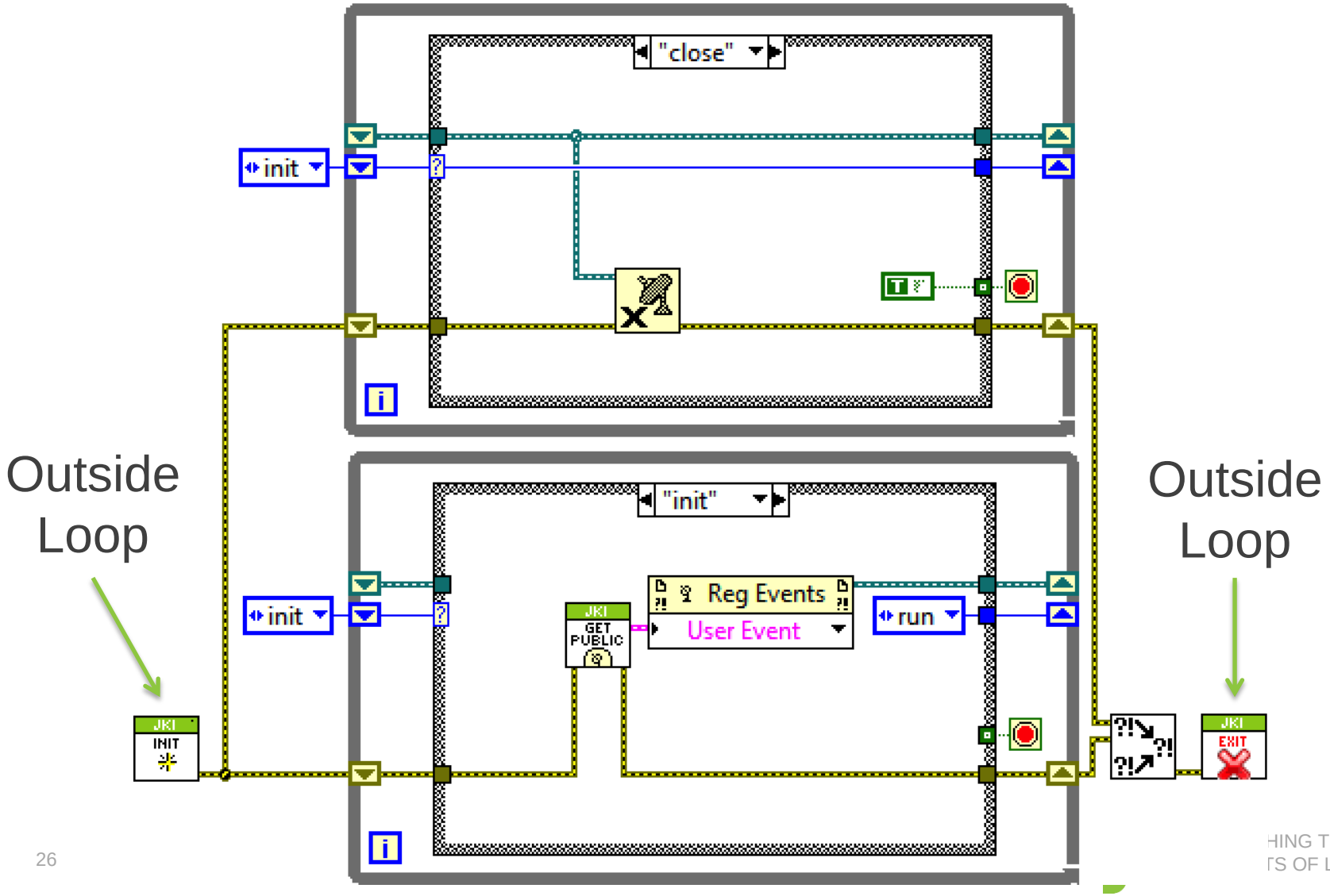
Run, Wait for Events, and Handle Them



Destroy User Event Registration and Close

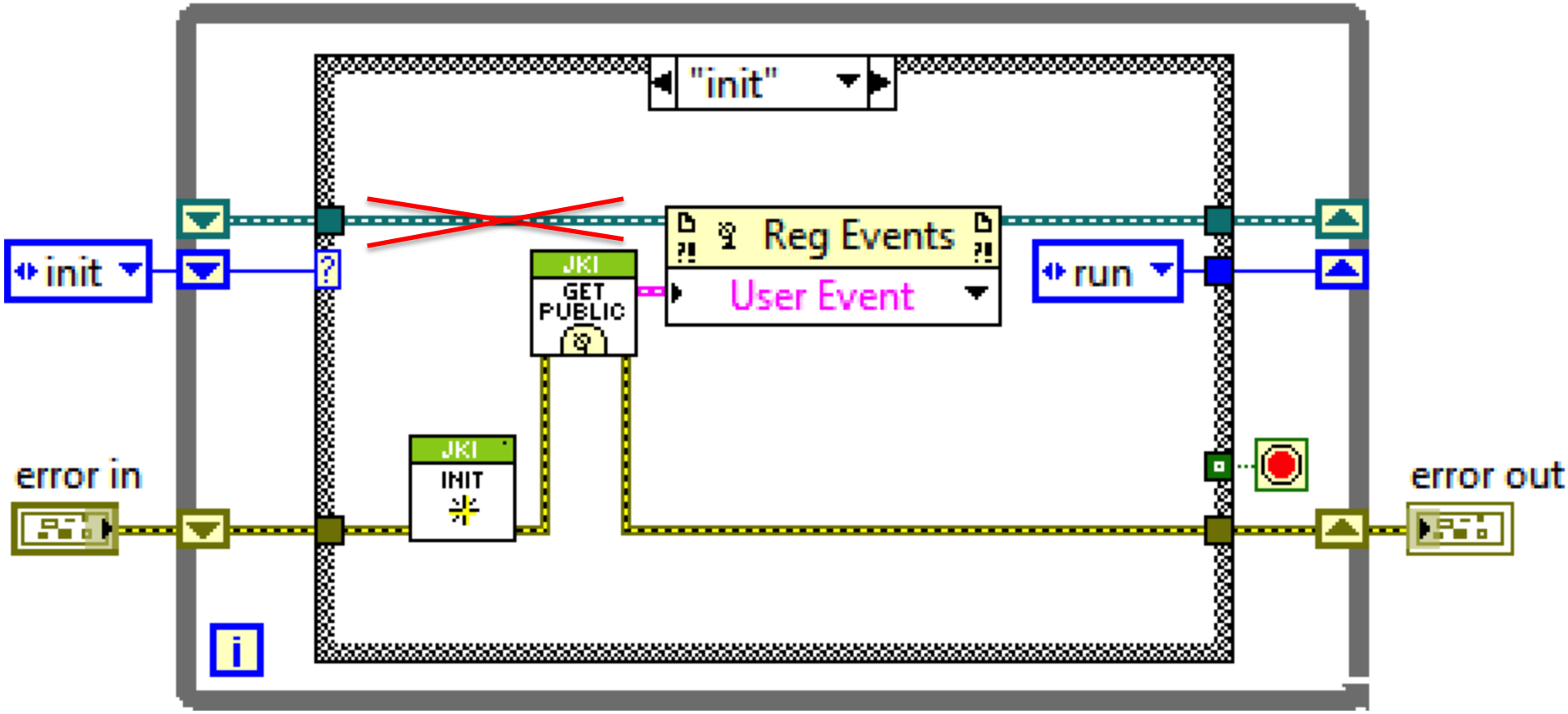


Two State Machine Subscribers

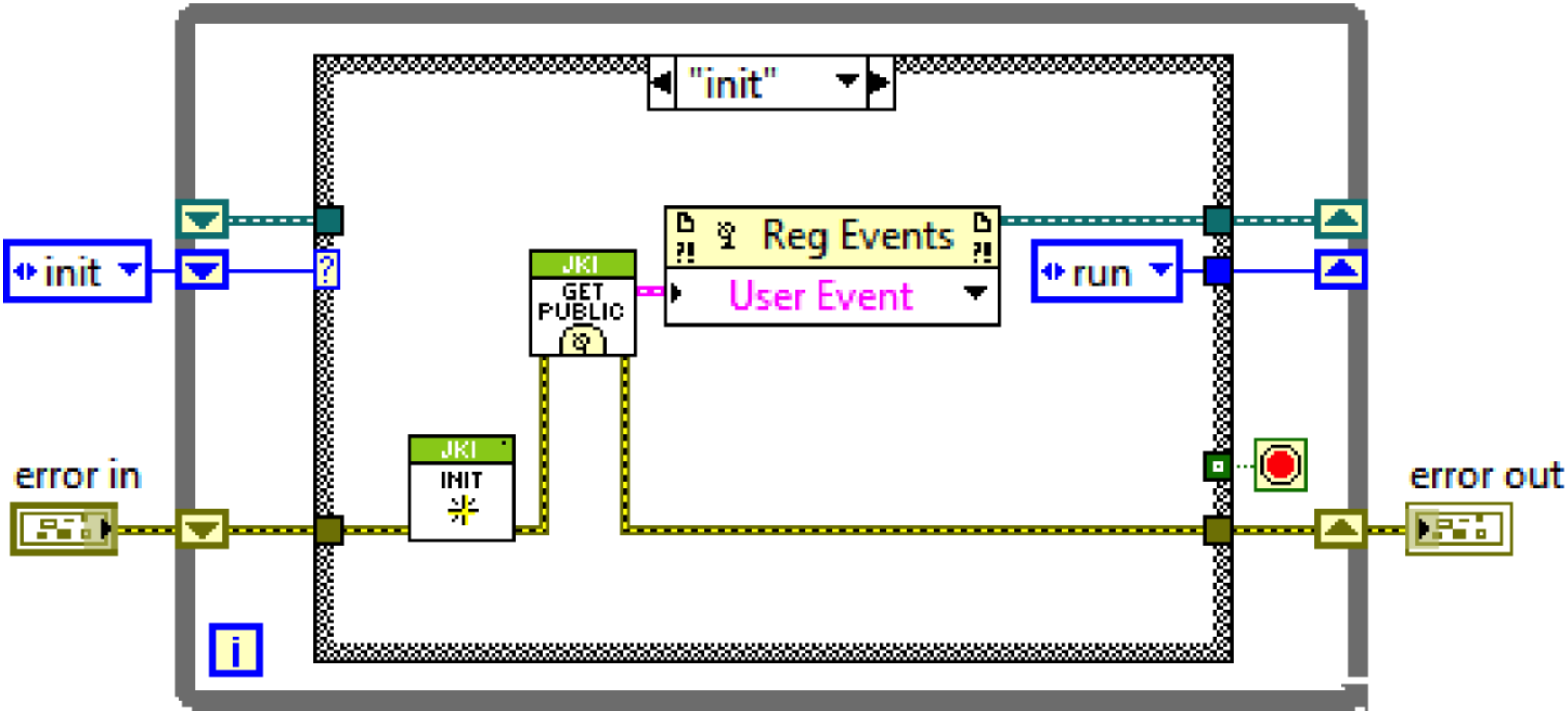


Remember the gotchas?

Don’t Wire the Event Registration Input

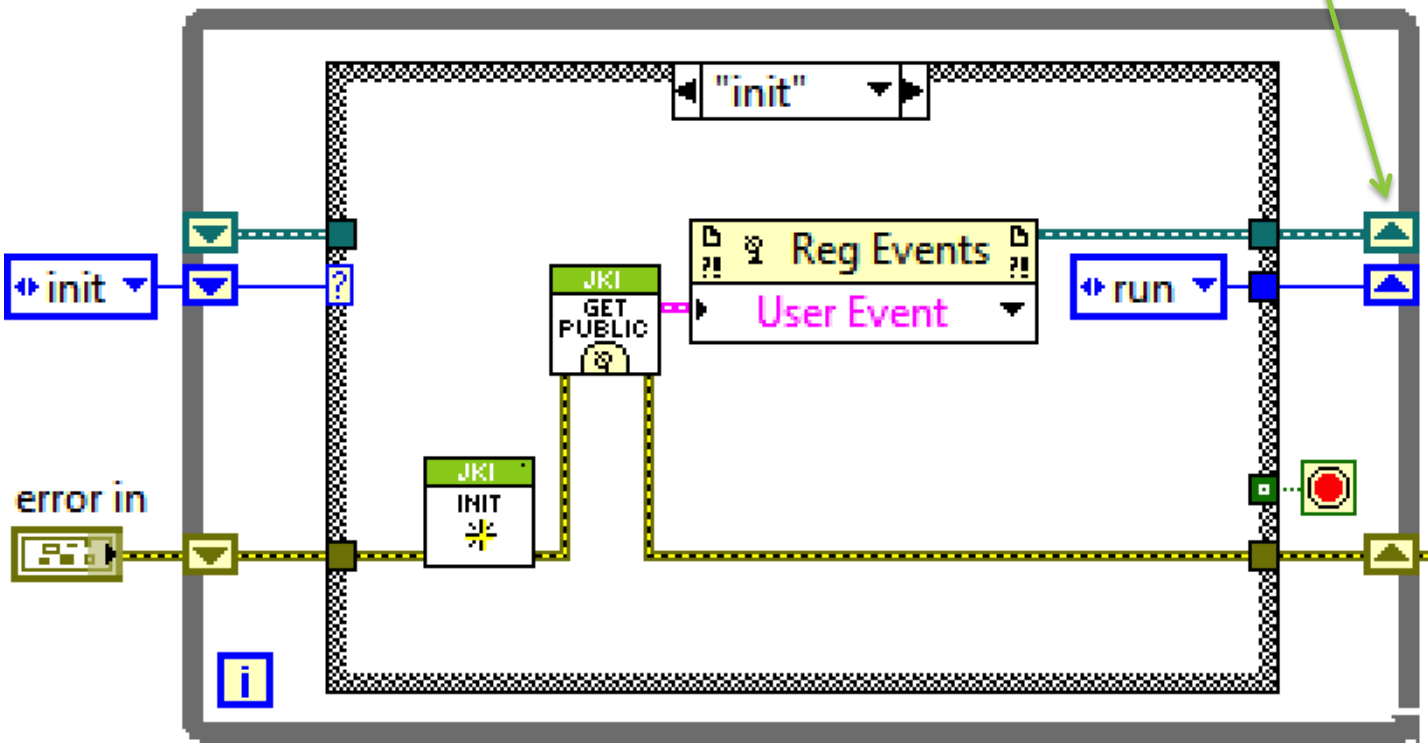


Don’t Wire the Event Registration Input



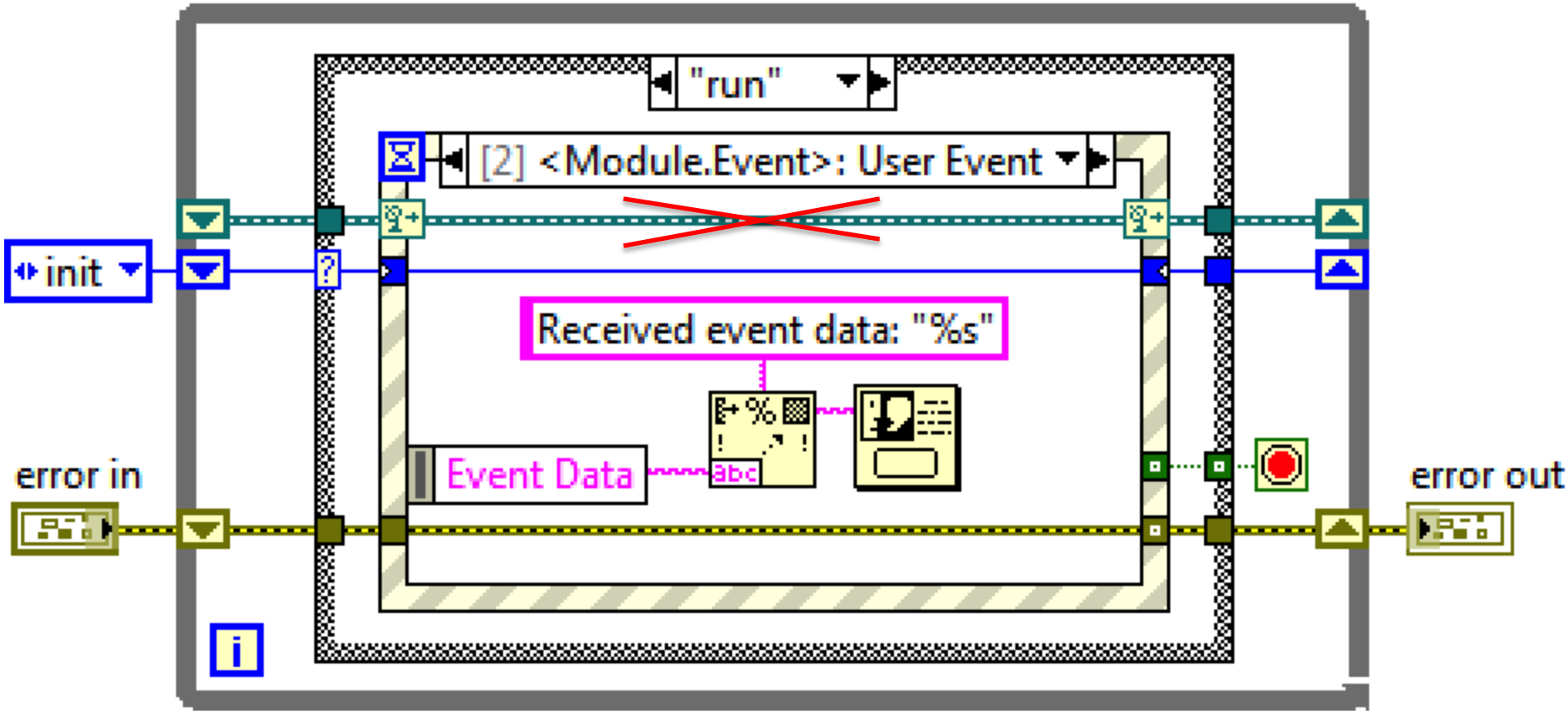
Put Event Reg in Separate Shift Register

Do give Event Registration it's own shift register

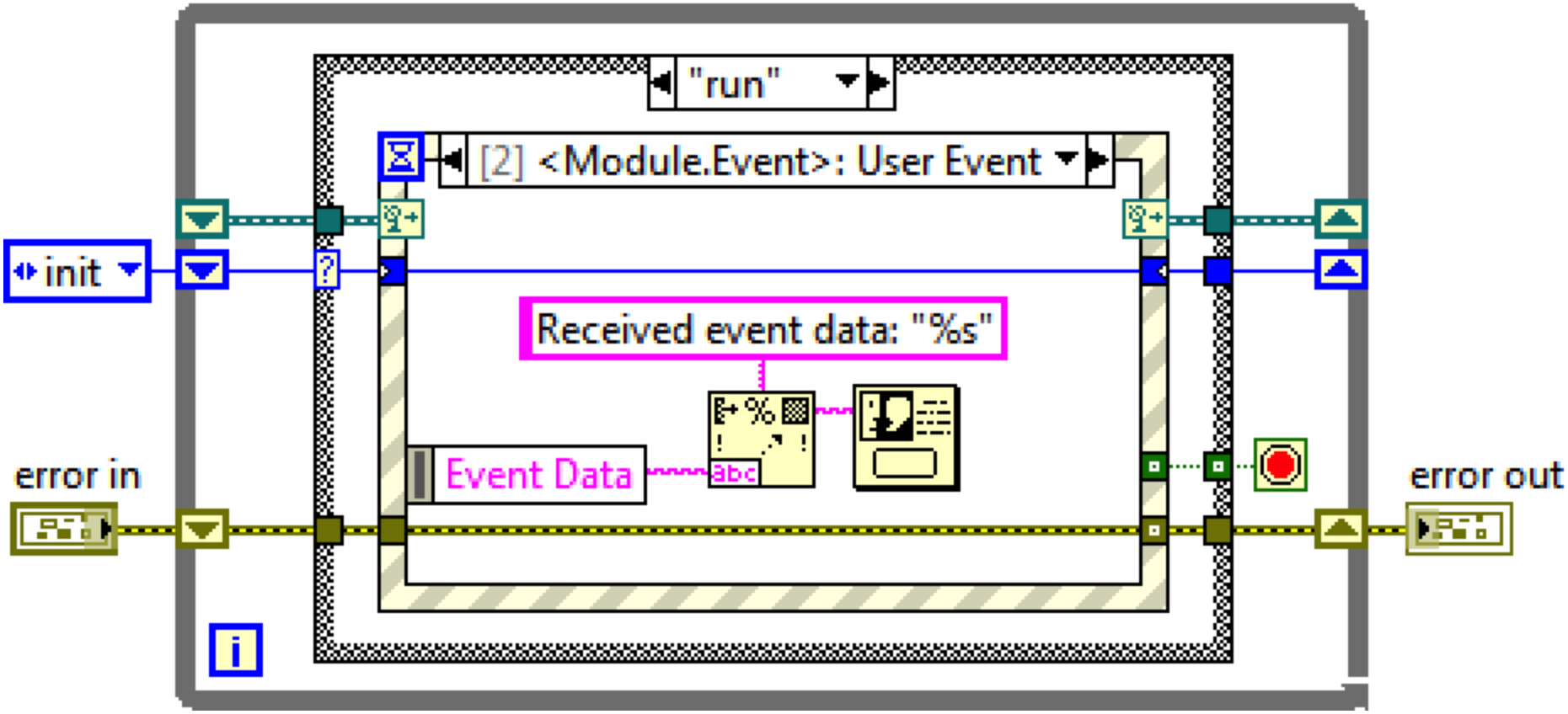


Don't bundle it with state data

Don’t Wire the Event Reg Through Event Struct

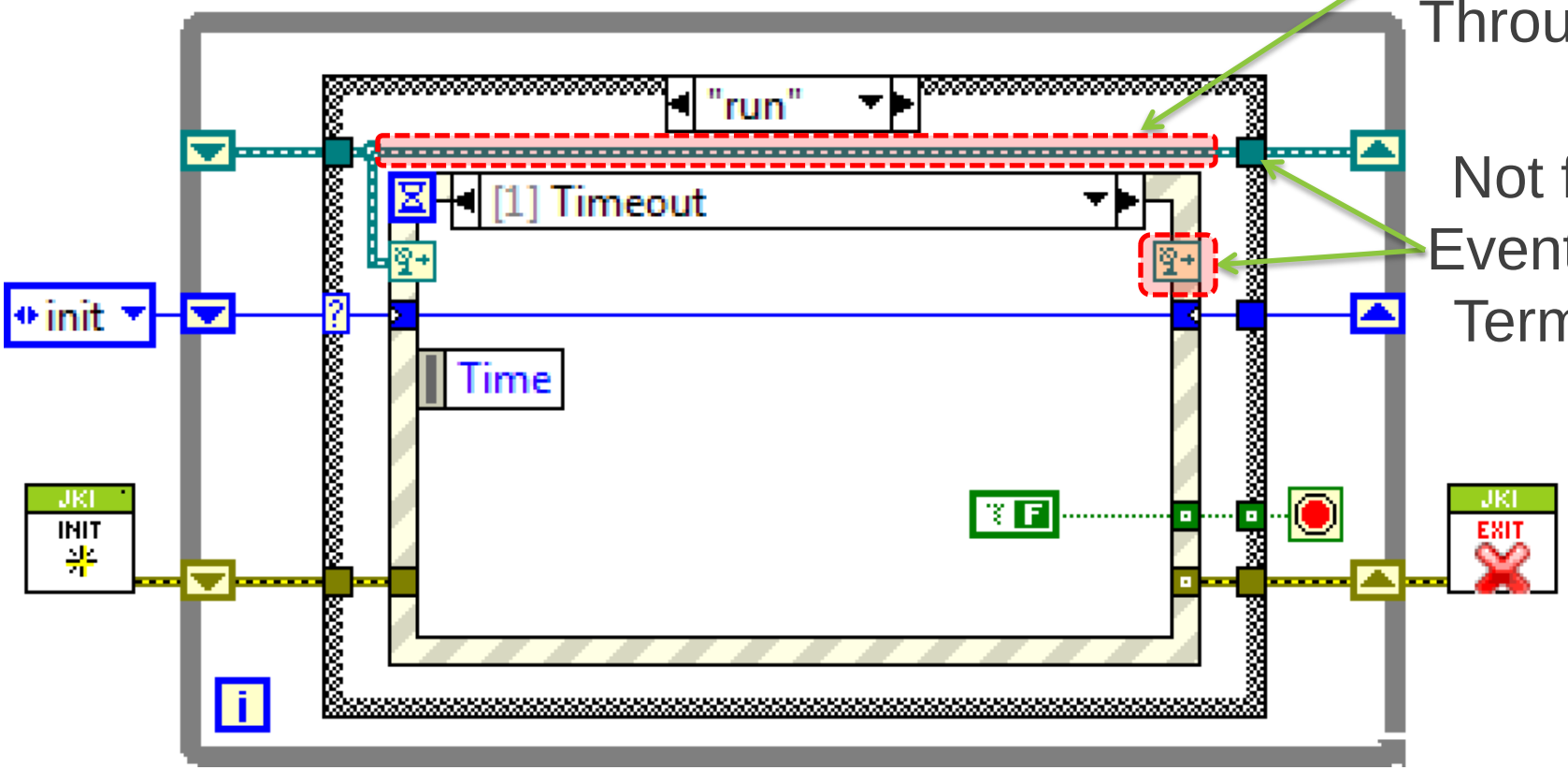


Don’t Wire the Event Reg Through Event Struct



Oh, and one more gotcha.
(That just got me!)

Wire Event Reg Straight Through



Wire Straight Through

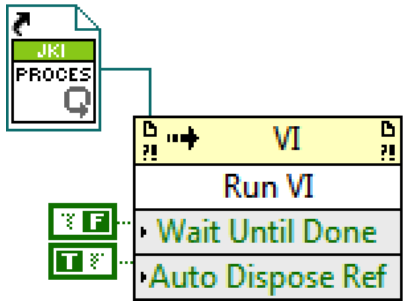
Not from Event Reg Terminal

Benefits of this Approach

- Saves time (lightweight & easy to use)
  - for module developer
  - for module users
- Improves performance
  - by avoiding polling for data
- Improves design
  - by loosely coupling producers from consumers
- Extensible
  - by being compatible with by-ref and by-value architectures

Extensions and Tweaks

- Incorporate into your module template (by-val, by-ref, actors)
- Starting and stopping asynchronous process
- Private/Protected/Community Events
- Sending message <u>into</u> the asynchronous process and getting a synchronous response



Things That Suck(ed)

- Most annoying bug ever (Event Struct. Frame Remapping).
  Fixed in LV2011!
- Event Structure timeout gets reset whenever unhandled events fired.
  Fixed in LV2012!
- No Event queue management or introspection.
  Maybe fixed in LV2013?
- No Notifier-like behavior (“ignore previous”).
  Maybe fixed in LV2013?

Take Home Point

- User Events are a very easy-to-use feature with a lot of cool functionality.
- They form the basis of JKI's primary application frameworks & templates.
- If we could get a couple things fixed/added to LabVIEW, we could do even better.
- Things keep getting better, and we're very thankful!

One More Thing

Join Our Team

Help us build the next generation of instrumentation.

Talk to a JKI engineer or visit **jki.net/careers**.