Advanced LIGO Automation with Guardian

Jameson Graef Rollins
LIGO Caltech

KAGRA seminar
April 14, 2015

LIGO-G1500506

Outline

Introduction

Introduction

Advanced LIGO has developed a novel automation system called **Guardian**.

The goal was to provide a framework for complete automation of the interferometer and all of its subsystems, in a way that is amenable to the unique way that these complicated instruments are commissioned.

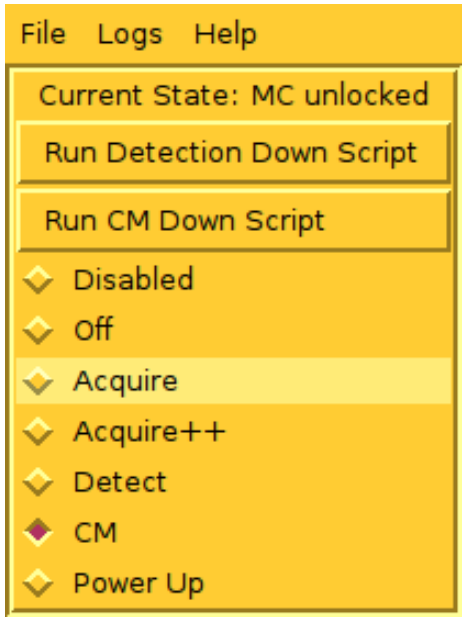
This presentation will describe some of the considerations that went in to the design of Guardian, the Guardian design itself, and the process of commissioning the aLIGO interferometers with Guardian.

Interferometer automation of the past

Interferometer automation has evolved a lot over the years...

Interferometer automation of the past

Initial LIGO automation was handled entirely by a handful of shell scripts that used command-line EPICS IO binaries to inspect and change the state of the real-time front end controllers.



AutoRUN.pl

The scripts described sequences of actions needed to transition the interferometer between various states, e.g.:

UNLOCKED → RESET → LOCK → COMMON MODE → etc.

The scripts were unified/managed by a simple program (*AutoRUN.pl*) that handled lock acquisition and recovery.

Interferometer automation of the past

There were numerous problems with this system:

- **no verification/validation**: Scripts ran "blind", with no verification that the appropriate initial conditions were met.
- **unreliable and slow**: All communication was done through shell EPICS IO binaries without any checks that connections were properly made, or that read/write calls had succeeded.
- **asynchronous**: The state of the system at any point in time was not well defined.
- **monolithic and inflexible**: The system was difficult to extend or reconfigure.

The system was also **unstructured** and **not well managed**.

Moving to Advanced LIGO

Advanced LIGO is *significantly* more complex than Initial/Enhanced LIGO:

- many more subsystems
- increased complexity of all subsystems, all of which now have their own control systems
- increased inter-dependency between subsystems

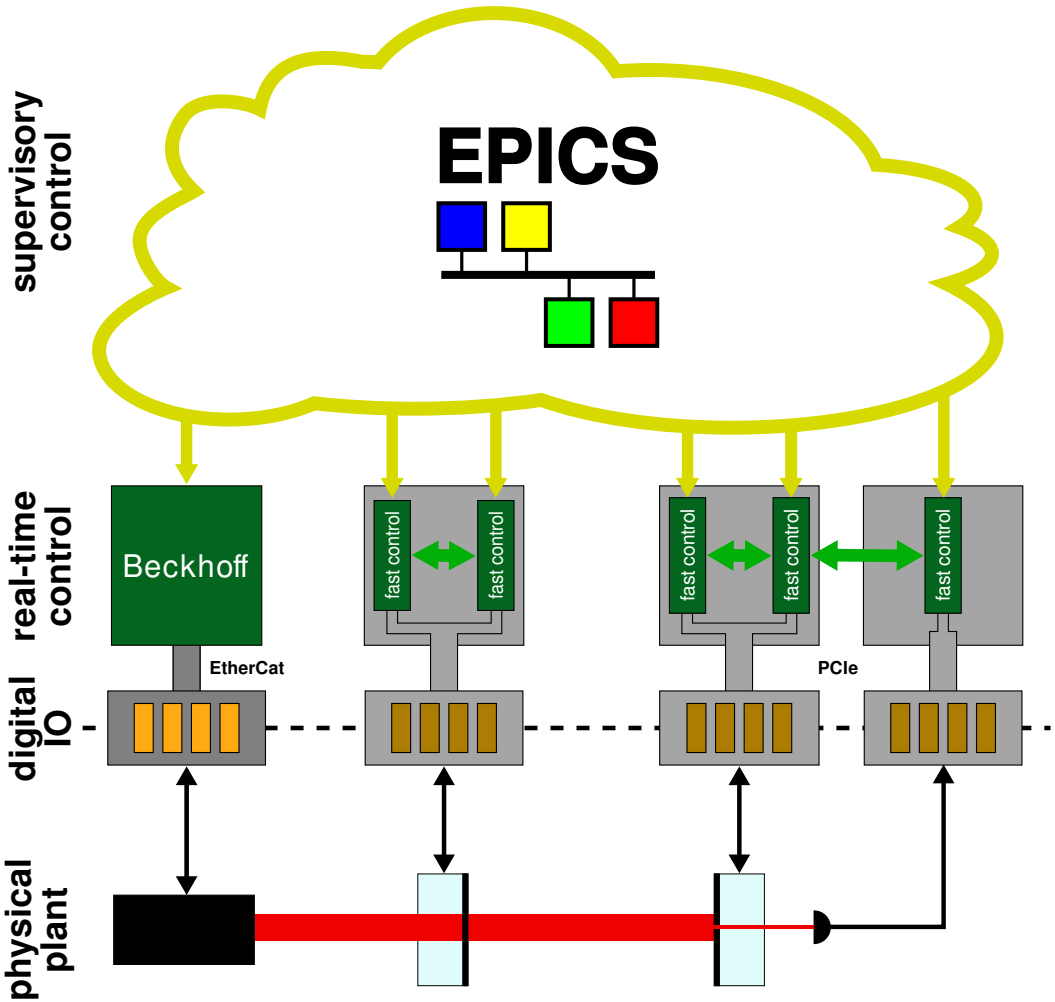
The "loose bundle of scripts" from iLIGO is inadequate.

Something more sophisticated is required.

Interferometer automation design considerations

The aLIGO digital controls

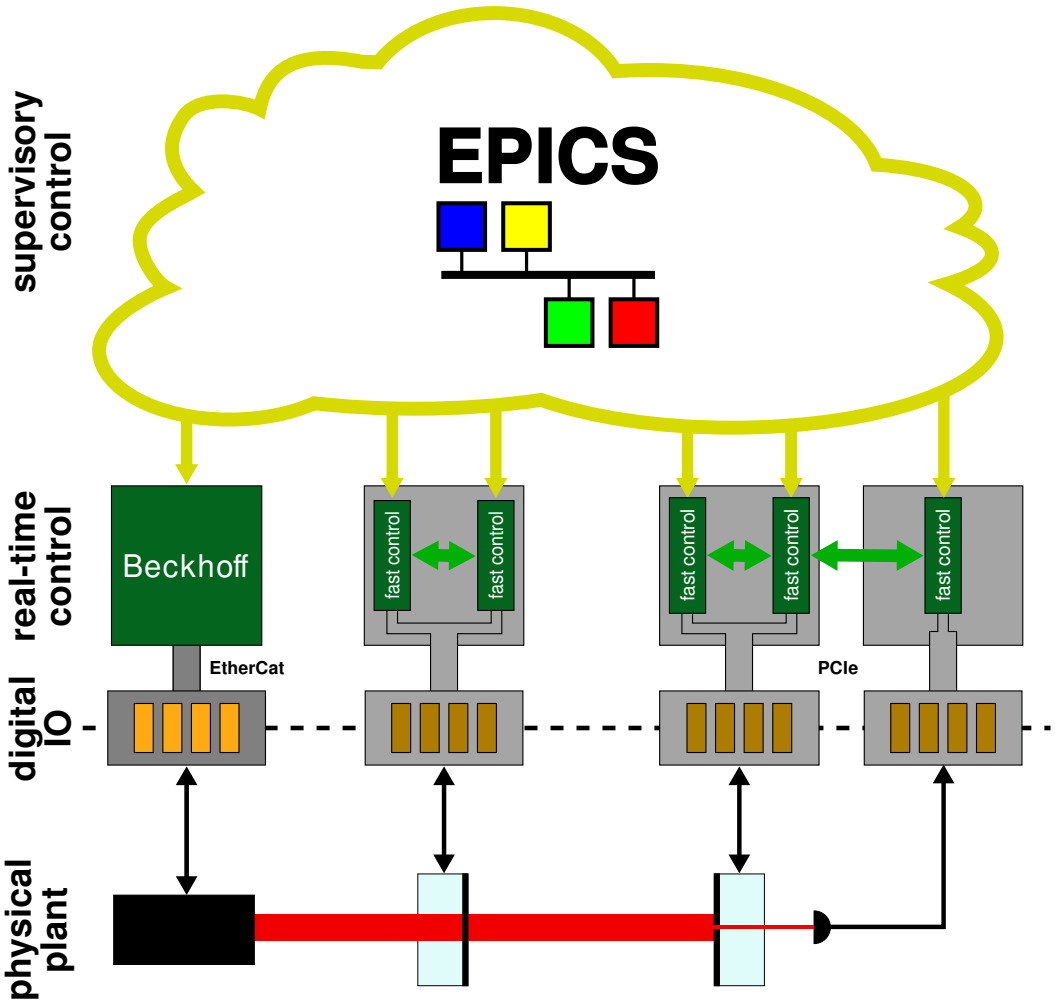
When we started looking at "supervisory controls" solutions, the aLIGO digital control infrastructure (**`advLigoRTS`**) was already in place.



The aLIGO digital controls

`advLigoRTS` is the "fast" real-time digital control system that maintains the interferometer resonance conditions. All supervisory control of `advLigoRTS` is done via EPICS.

Beckhoff is also used for some "slow" hardware control.



aLIGO automation requirements

When looking for an automation solution, we started by outlining a set of basic requirements. It should be:

- **modular**: Program once for repeated subsystem components.
- **extensible**: Easily expand and extend the system as needed.
- **robust**: IO failures should always produce errors.
- **flexible**: Operators/commissioners should be able to redirect the system as needed.
- **responsive**: Respond to *unintended* changes in the plant, e.g. lockloss, activation of hardware protection system, etc.
- **fast**: Respond to changes promptly, to keep downtime to a minimum.
- Well structured, maintainable, auditable, etc.

aLIGO automation requirements

We also thought about more specific requirements:

- **EPICS**: Must speak to the existing fast digital control system (`advLigoRTS`) through the existing network IPC interface.
- **distributed**: Program and control individual subsystem components independently, then integrate them seamlessly into the global control in a unified way.
- **deterministic**: State should be well defined, and transitions should always behave the same way. No "hidden" variables. (LIGO is a **scientific instrument**, so we must have a complete record of the state of the instrument at all points in time.)

Importance of commissioning process

We also thought a lot about how the interferometers, including the automation logic, are actually commissioned:

The system must integrate with the unique interferometer commissioning process.

The interferometer locking procedure *can not* be fully designed ahead of time; it is *discovered* during commissioning. The development and operator interfaces of the automation system must facilitate this commissioning process, not hamper it.

This is a *very* important consideration, as we will see.

The automation system should be simple and flexible. It should reduce the complexity of the instrument, not add to it.

Possible solutions

So what are the options?

Possible solutions: PLCs

The industry standard for automation is **programmable logic controllers** (PLCs), typically represented by IEC-61131-3. Beckhoff systems (also used in aLIGO) are examples of PLCs.

PLCs are *firm real-time systems* (like `advLigoRTS`): code is run at fixed time intervals, and output must be calculated from input before the next code cycle. This makes them:

- synchronous
- robust
- fast
- responsive

Industry standard also implies potential external support.

Possible solutions: PLCs

However, PLCs would be difficult to integrate into aLIGO for *global supervisory control* for a number of reasons:

- No EPICS. Would need to develop a shim, or build a separate interface into the `advLigoRTS`.
- Typically run on Windows or other special platforms, whereas we prefer Linux.

But most importantly:

- Code requires *compilation*, which means the turn around time for making changes would be unacceptably long
  → **bad for commissioning**

(Our experience with Beckhoff has also been not been great; many usability issues.)

Possible solutions: extend `advLigoRTS`

Another possible solution is to **extend the `advLigoRTS`** to support supervisory control directly. This has some benefits:

- most of the benefits of PLCs (synchronous, robust, etc.)
- fully integrated into the existing infrastructure

However, it would have required a lot of changes and extensions to the `advLigoRTS`:

- support slower model rates
- create means of direct access to all process variables from supervisor models
- limited library support: Linux kernel libraries only

Also, as with PLCs, **would require compilation** and slow turn-around times → **bad for commissioning**

Possible solution: Guardian

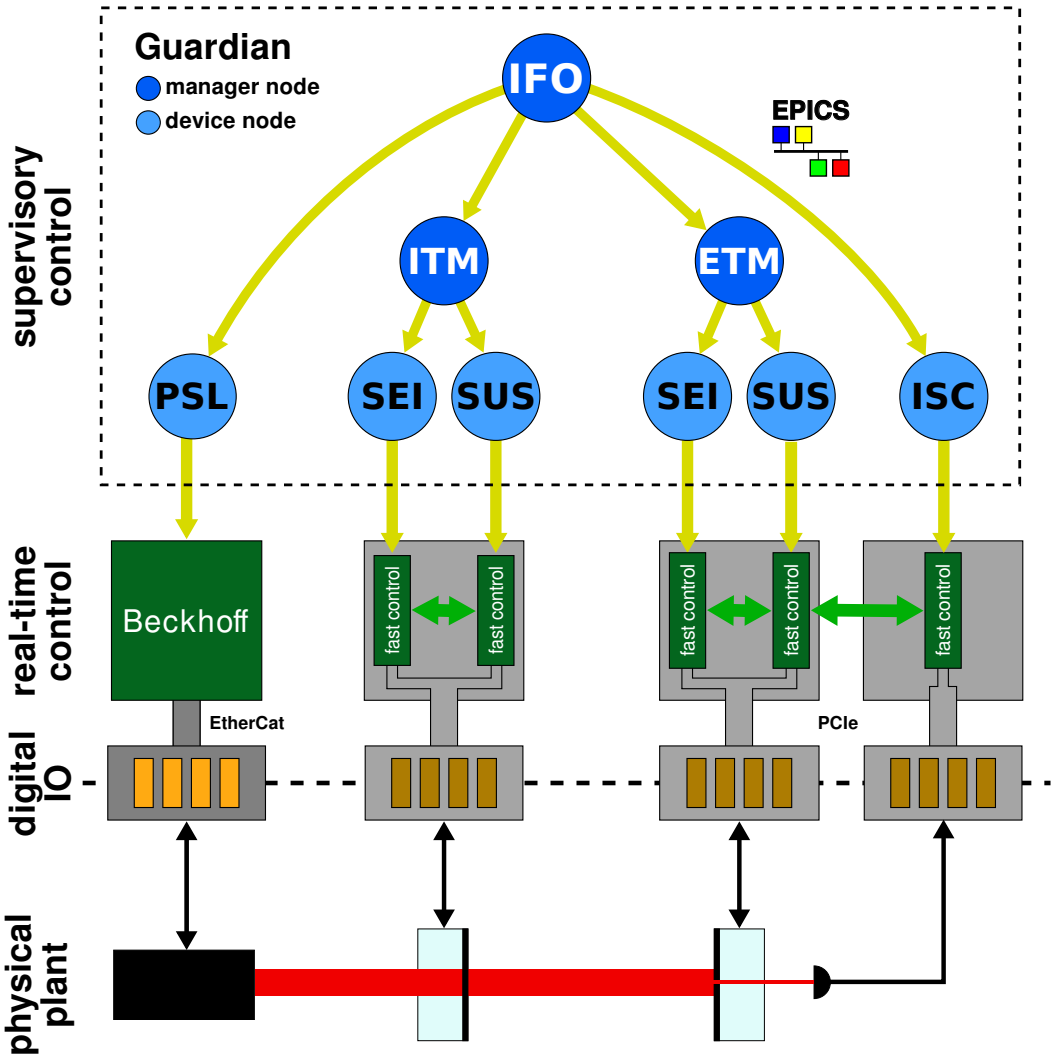
In the end, we decided to build a system tailored to our needs...

aLIGO Guardian

Guardian design concept

Guardian is a **hierarchical, distributed, state machine**.

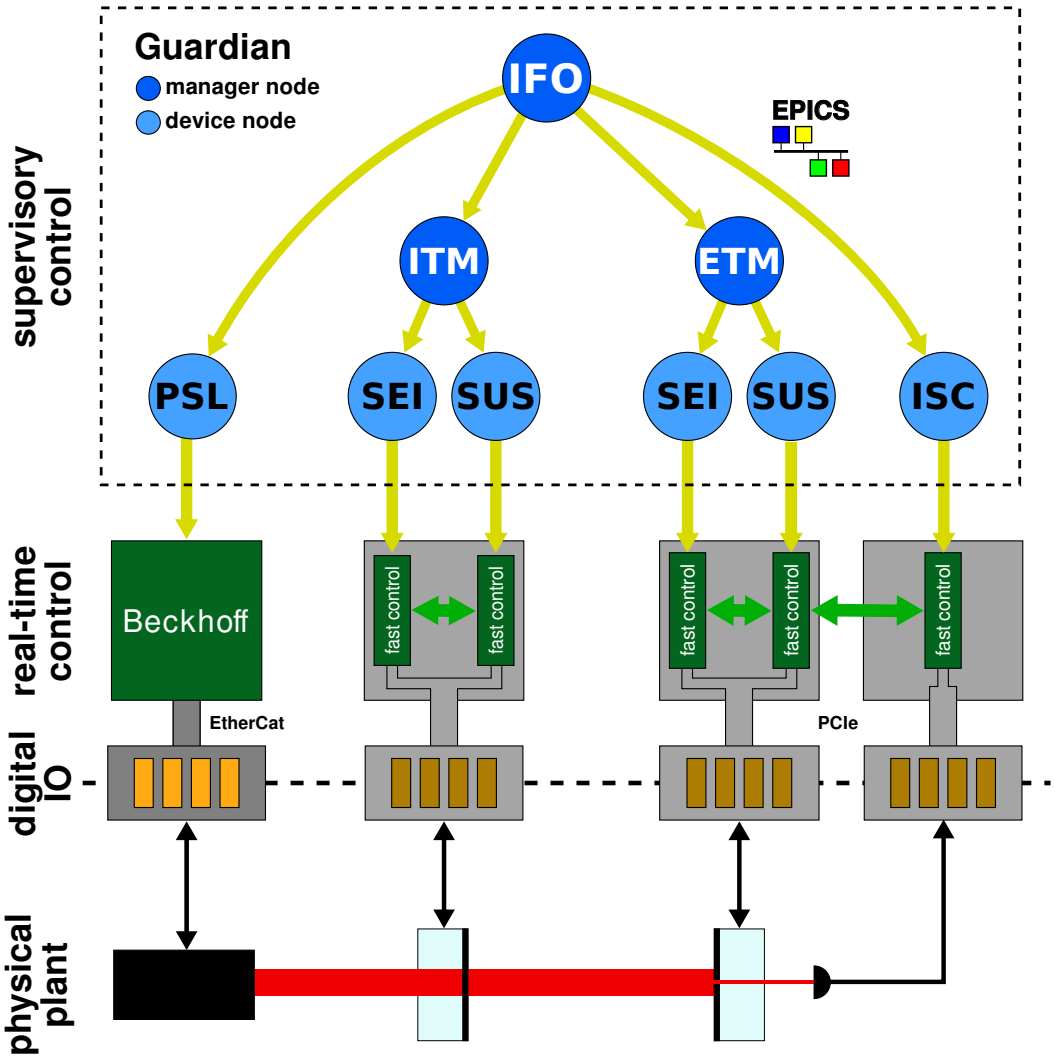
Individual **nodes** oversee well defined sub-domains of the interferometer.



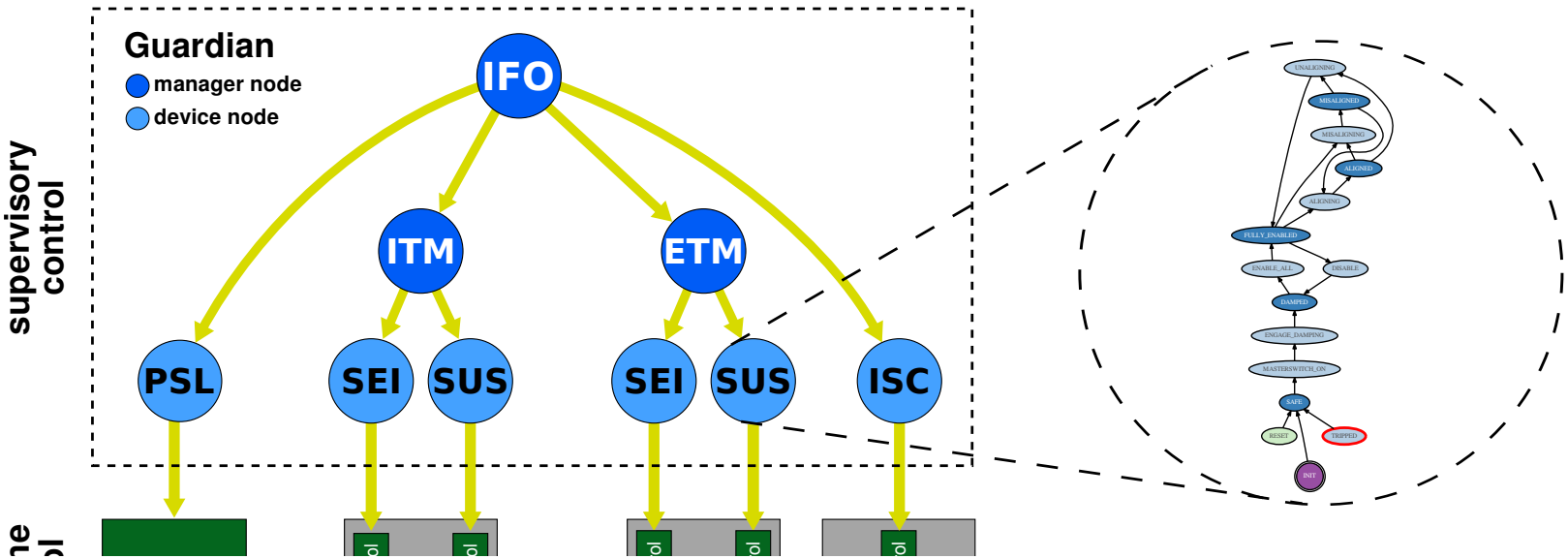
Guardian design concept

A hierarchy of nodes control the full interferometer.

Upper-level **manager** nodes control lower-level **subordinate** nodes, with **device** nodes talking directly to front end hardware devices via EPICS.



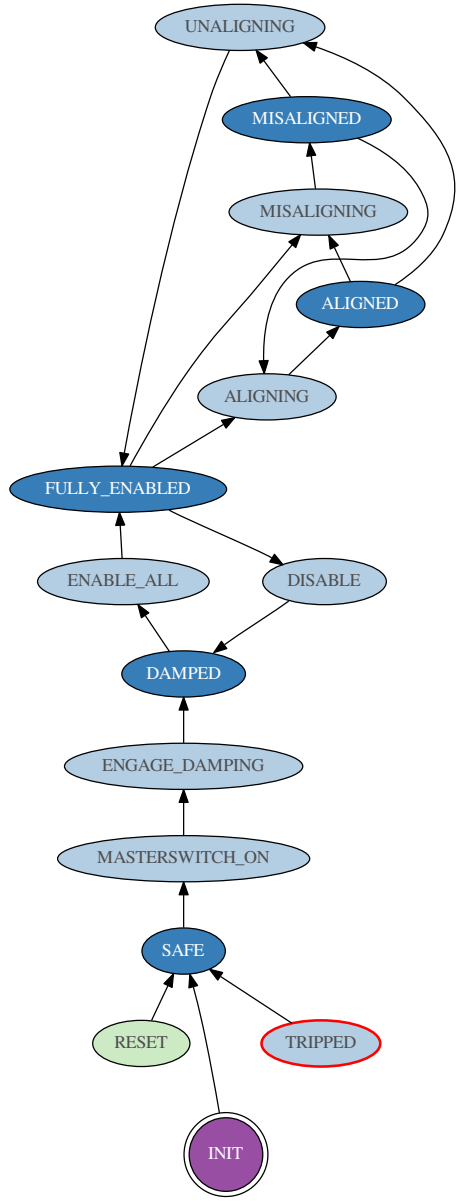
Guardian design concept: nodes



Each node executes a **state graph** for its system.

Guardian design concept: state graphs

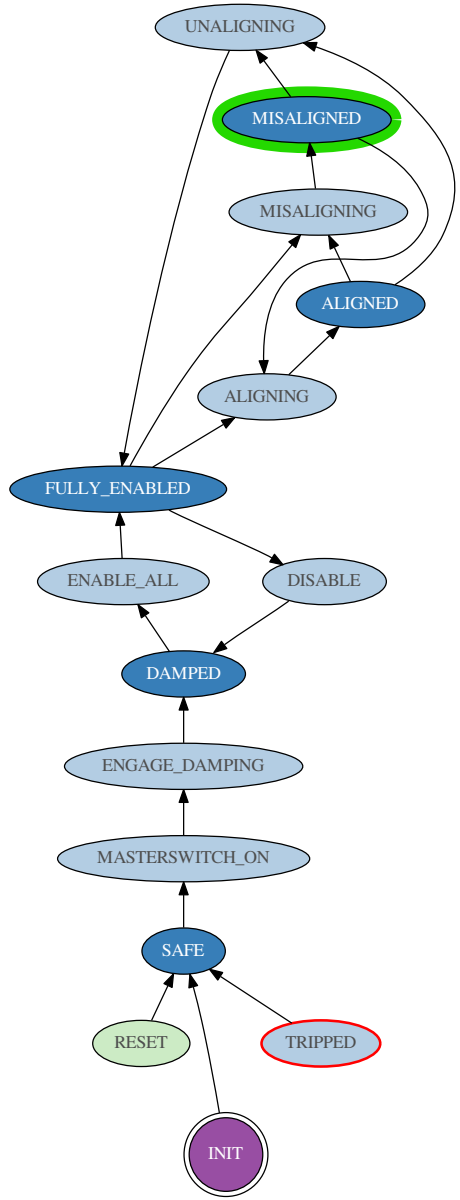
The **state graphs** describe the accessible states of the system, and the allowable transitions between states.



Guardian design concept: state graphs

The **state graphs** describe the accessible states of the system, and the allowable transitions between states.

The node accepts commands in the form of a **state request**.

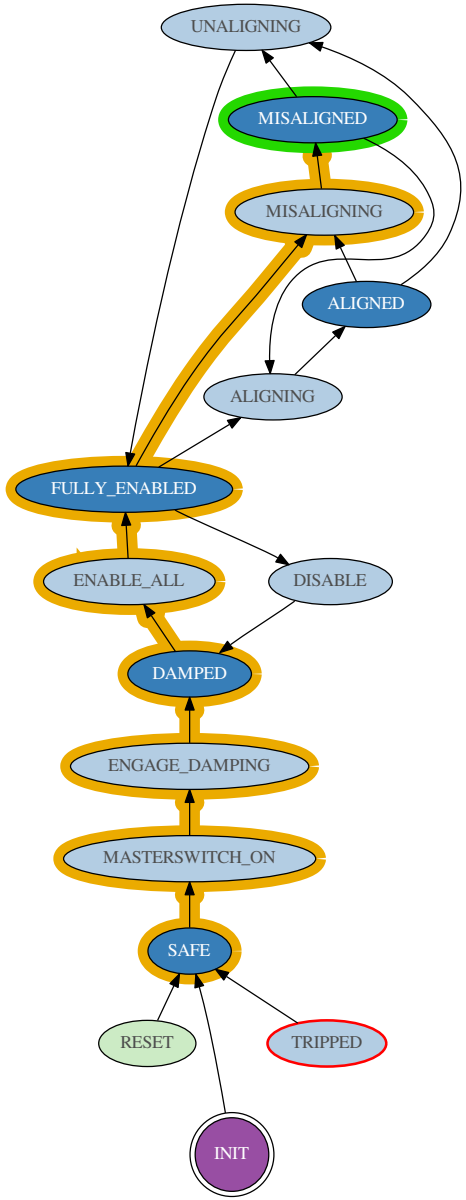


Guardian design concept: state graphs

The **state graphs** describe the accessible states of the system, and the allowable transitions between states.

The node accepts commands in the form of a **state request**.

Guardian then calculates the path from the current state to the requested state, and executes all states in the path in sequence.

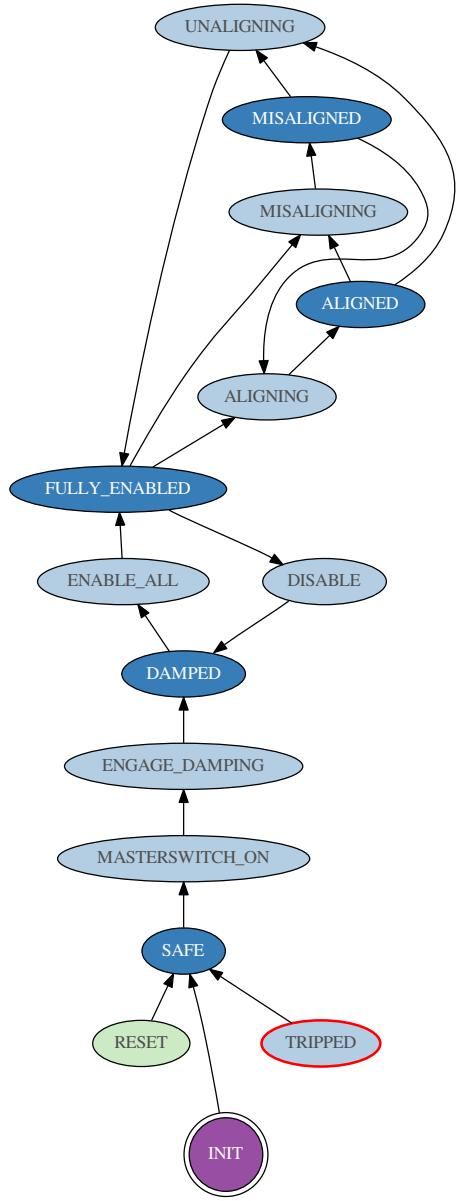


Guardian design concept: state graphs

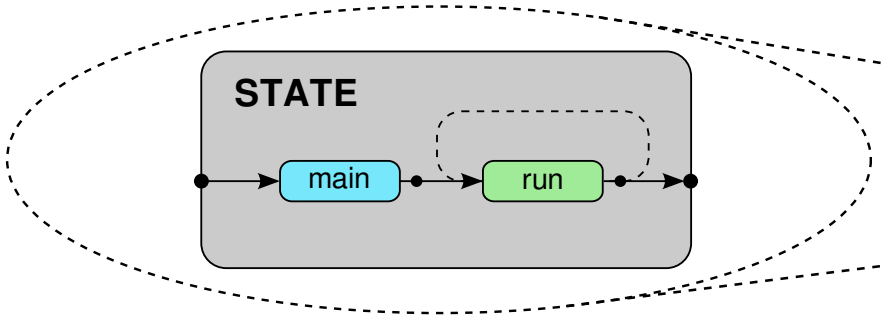
Guardian is a **finite state machine** (FSM): each state is a logically distinct block of code.

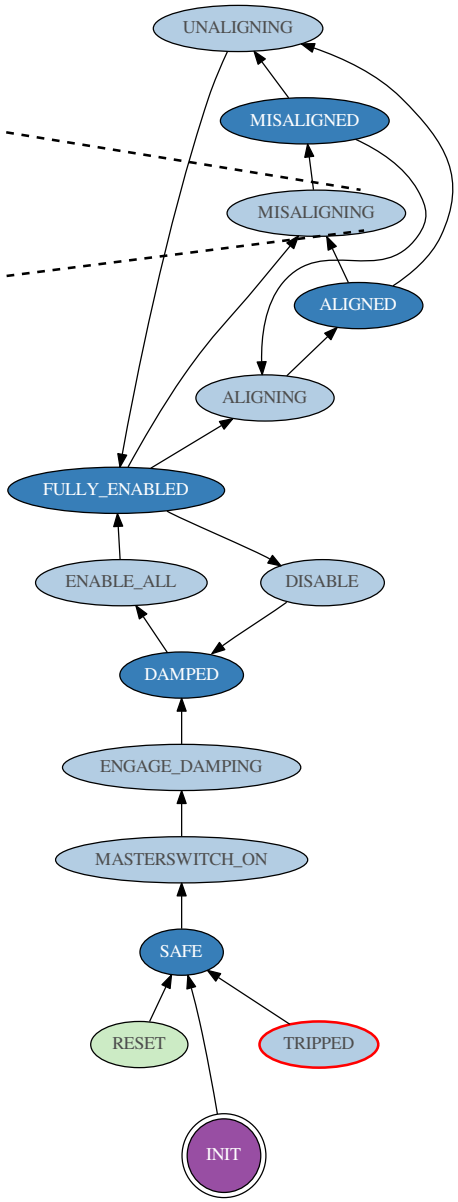
The FSM design forces all persistent process variables to be *external* to the code, in this case stored in EPICS records that are fully recorded by the data acquisition system.

The full state of the automation system at any point in time can then be **completely reconstructed** from data on disk.



Guardian design concept: state graphs



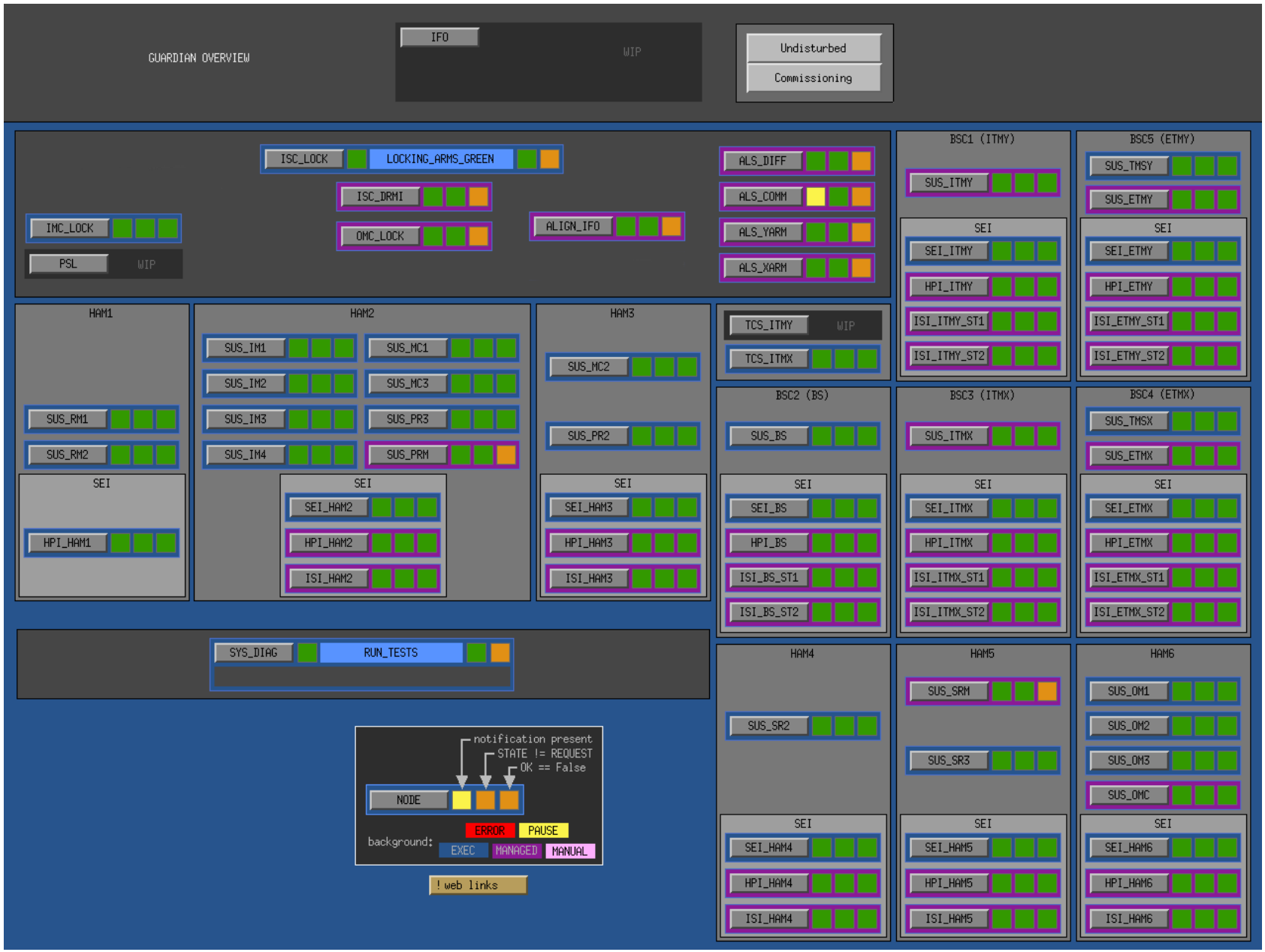


Guardian nodes are **soft real-time systems** that employ a timed *run loop* to execute the primary state code (*main*), and code that continually checks for state exit conditions (*run*).

Automation logic is programmed in Python, with full access to all python functionality. **New user code can be loaded *on-the-fly*, without requiring re-compilation.**

Commissioning with Guardian

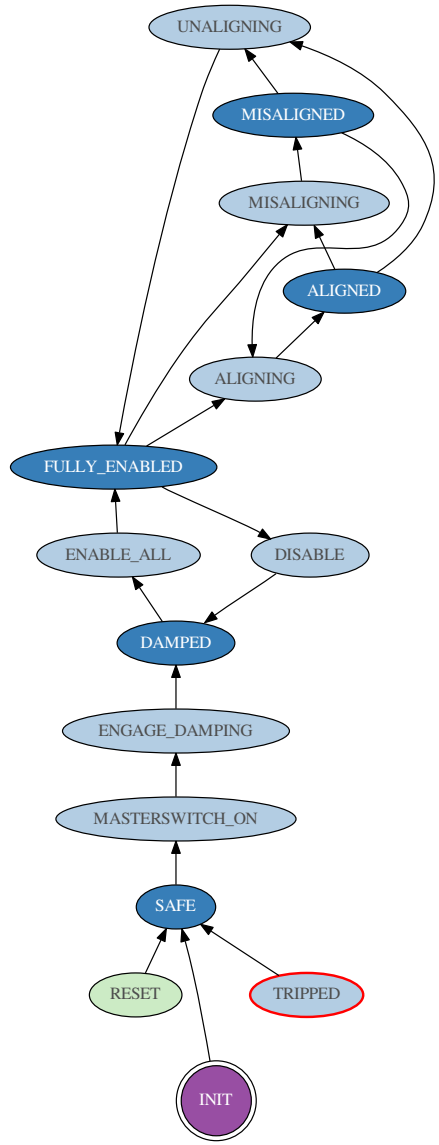
H1 Guardian overview



Subsystems: suspensions (SUS)

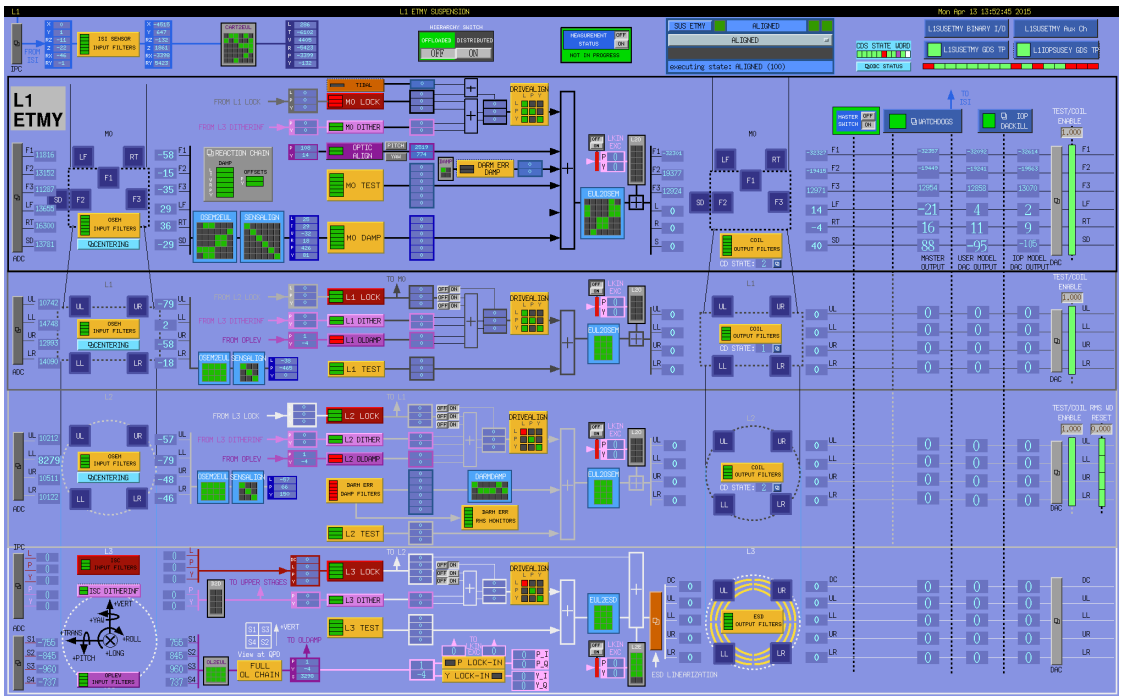
The **suspension** subsystem (SUS) was the simplest subsystem to automate. All suspensions have similar controller topology, which allows us to use the **same Guardian code for all suspensions**.

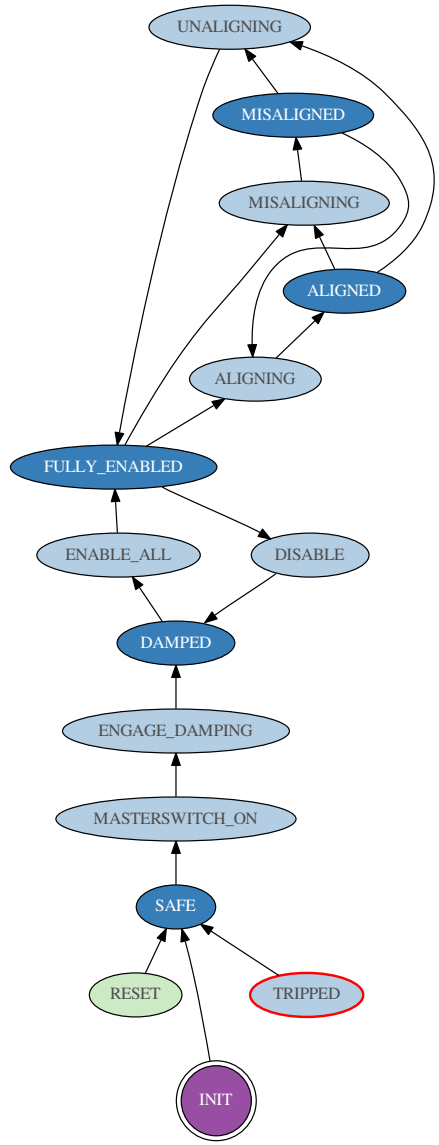
Provides automated recovery to the requested state after activation of the hardware protection system ("watchdog"), i.e. *one-click* reset.



Subsystems: suspensions (SUS)

The simplified interface provided by Guardian reduces the complexity of the system significantly:



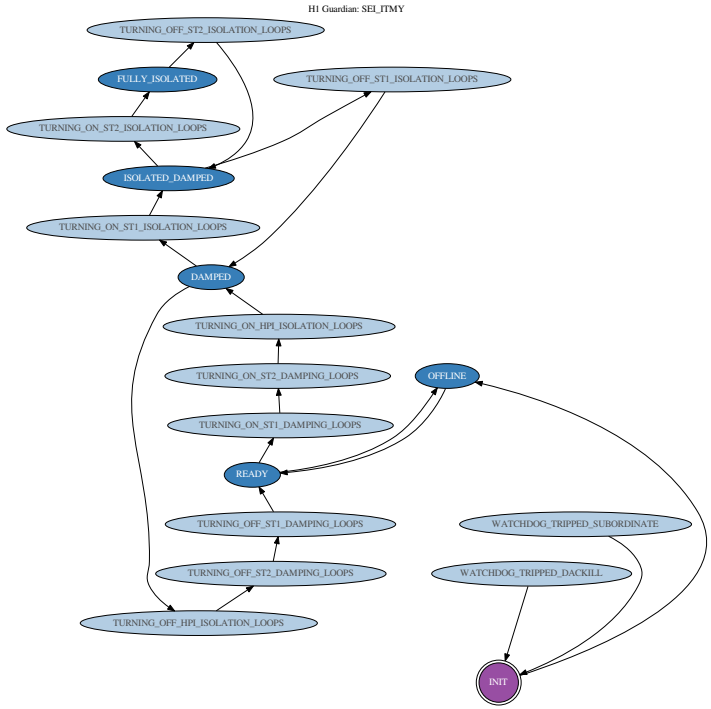


Guardian provides straightforward access to the most common and useful states of the system.

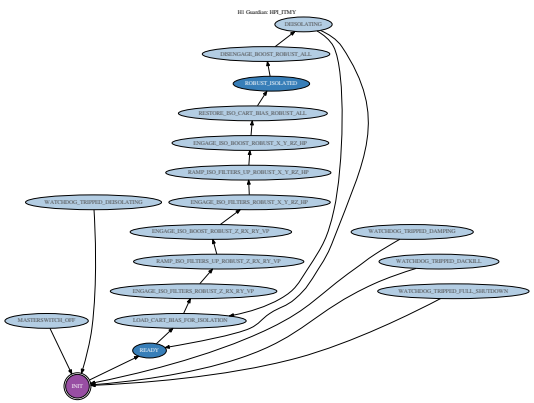
Subsystems: seismic isolation (SEI)

The **seismic isolation** subsystem (SEI) is one of the more complicated subsystems.

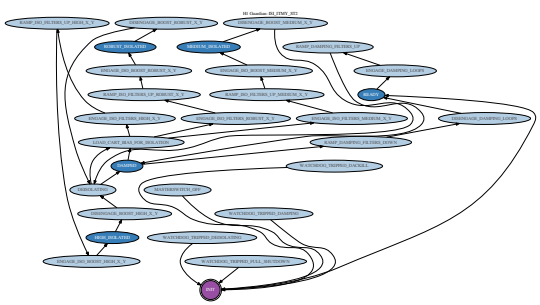
SEI in the core optic "BSC" chambers utilizes four Guardian nodes, including one *chamber manager*.



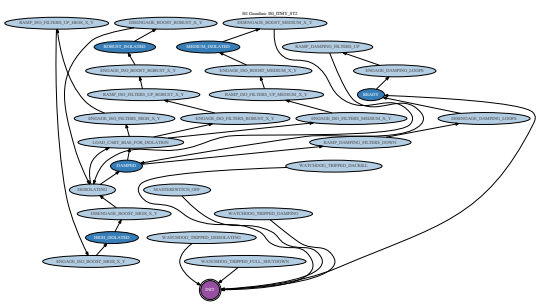
SEI manager



HPI



ISI ST1



ISI ST2

Subsystems: seismic isolation (SEI)

The SEI chamber manager simplifies the isolation procedure by coordinating state transitions between the three active stages of the isolation system, as well as by providing a simple interface for the operator/commissioner and the rest of the automation system.

SEI manager

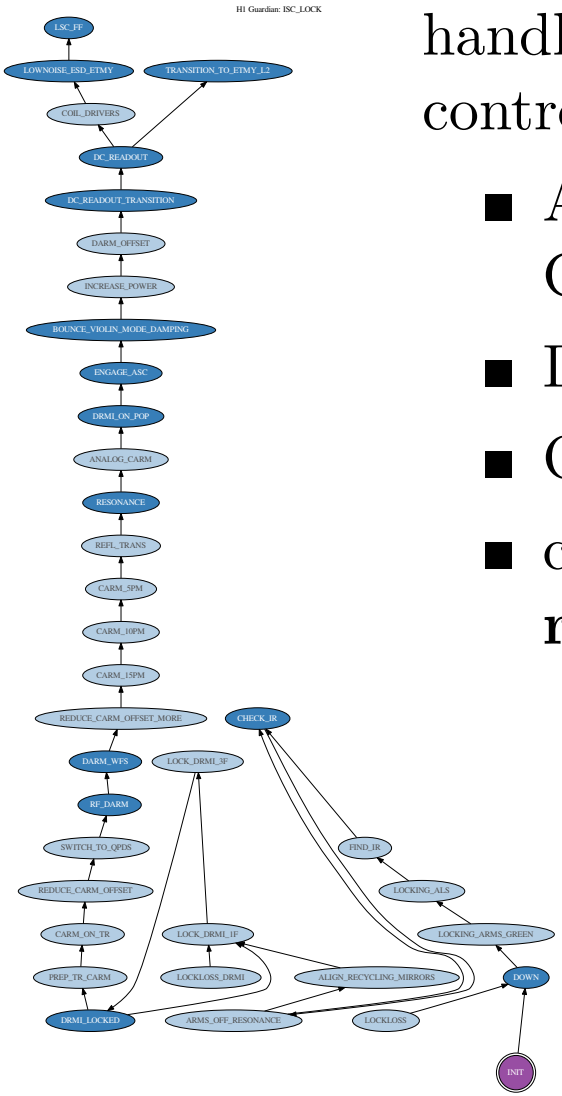
Subsystems: seismic isolation (SEI)

As with SUS, the modularity of Guardian allows us to use the same automation logic for all seismic platforms.

The SEI Guardians also provide full recovery after watchdog trips.

SEI manager

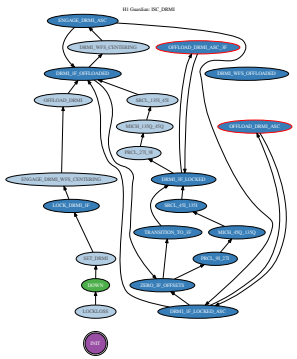
Subsystems: sensing and control (ISC)



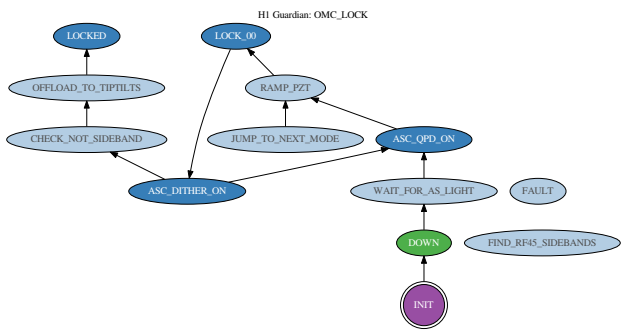
ISC_LOCK

Interferometer sensing and control (ISC) is handled by multiple nodes that independently control:

- ALS locking of the arms (XARM, YARM, COMM, DIFF)
- DRMI degrees of freedom
- OMC lock
- coordinating all subsystems for **full DC readout lock**



ISC_DRMI



OMC_LOCK

Guardian MEDM control interface

The nodes themselves use EPICS for the control interfaces.

The interface allow commissioners and operators to:

- view full status of the node
- REQUEST a state from the system
- PAUSE the node
- LOAD new user code
- view node logs, graphs, code, etc.

Tracking Guardian state

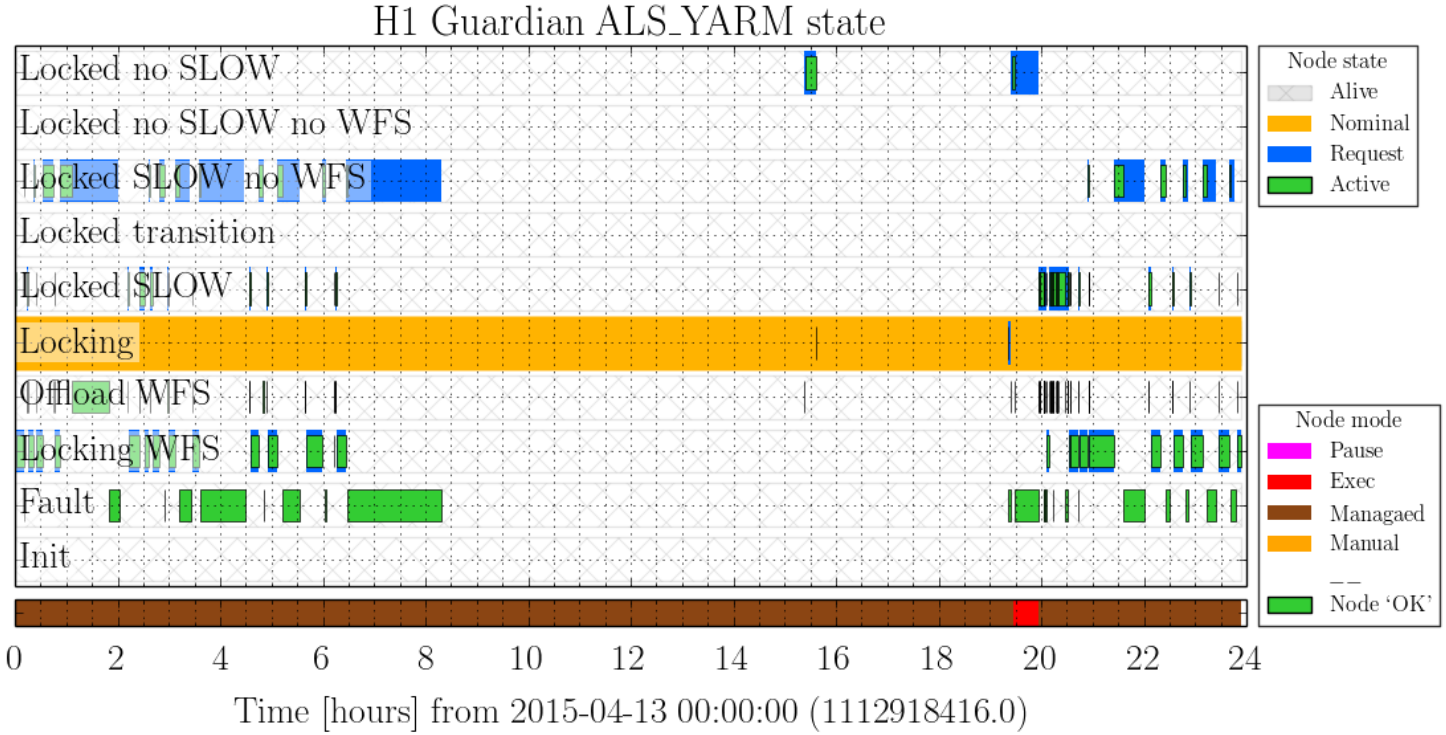
The interface provides feedback that the system has arrived at the requested state, as well as if the system as reached the ultimate “NOMINAL” desired state.

Tracking Guardian state

The interface provides feedback that the system has arrived at the requested state, as well as if the system as reached the ultimate “NOMINAL” desired state.

Tracking Guardian state

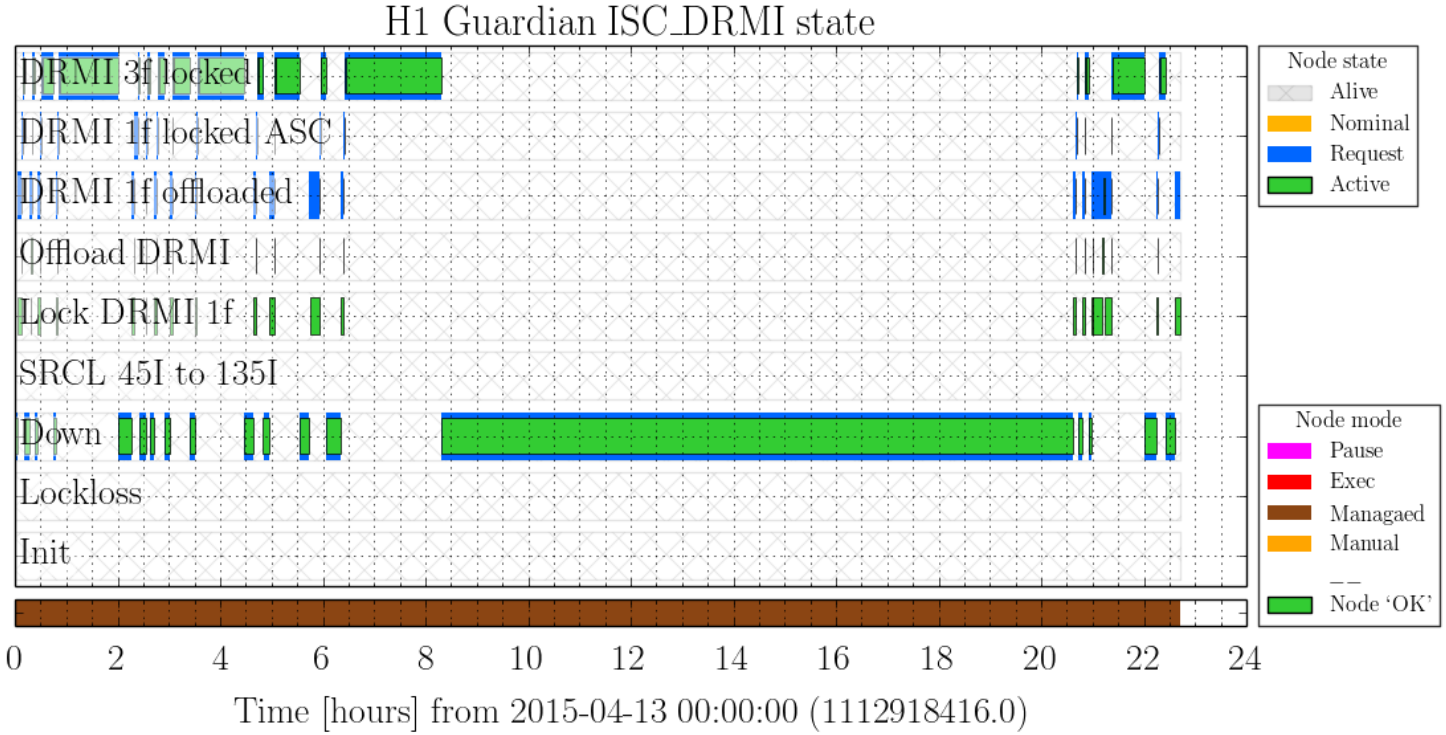
Since the control channels and state readbacks are all in EPICS, the **full state of all Guardian nodes is recorded in the frames**.



This enables **detector characterization** to accurately track the states of all subsystems over time.

Tracking Guardian state

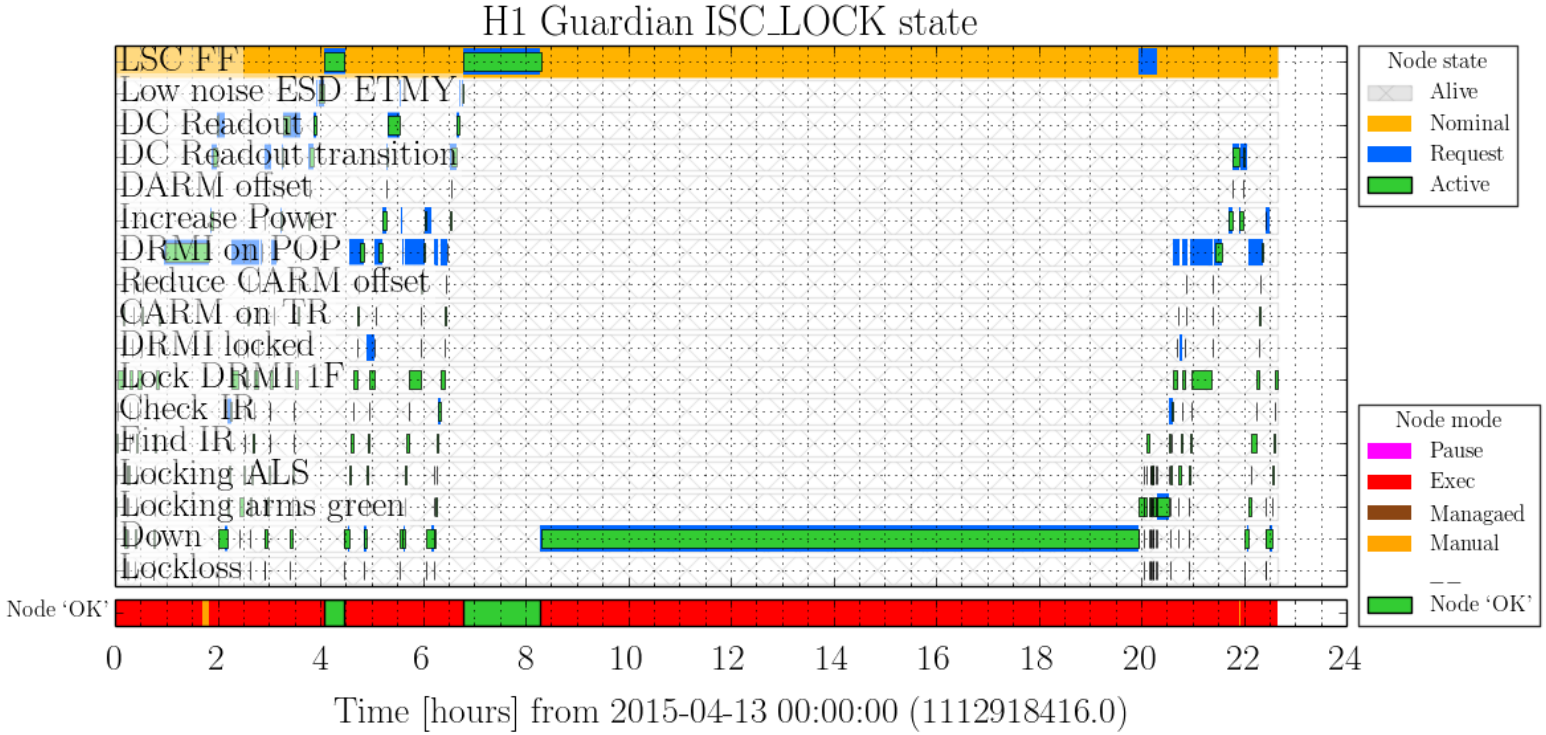
Since the control channels and state readbacks are all in EPICS, the **full state of all Guardian nodes is recorded in the frames**.



This enables **detector characterization** to accurately track the states of all subsystems over time.

Tracking Guardian state

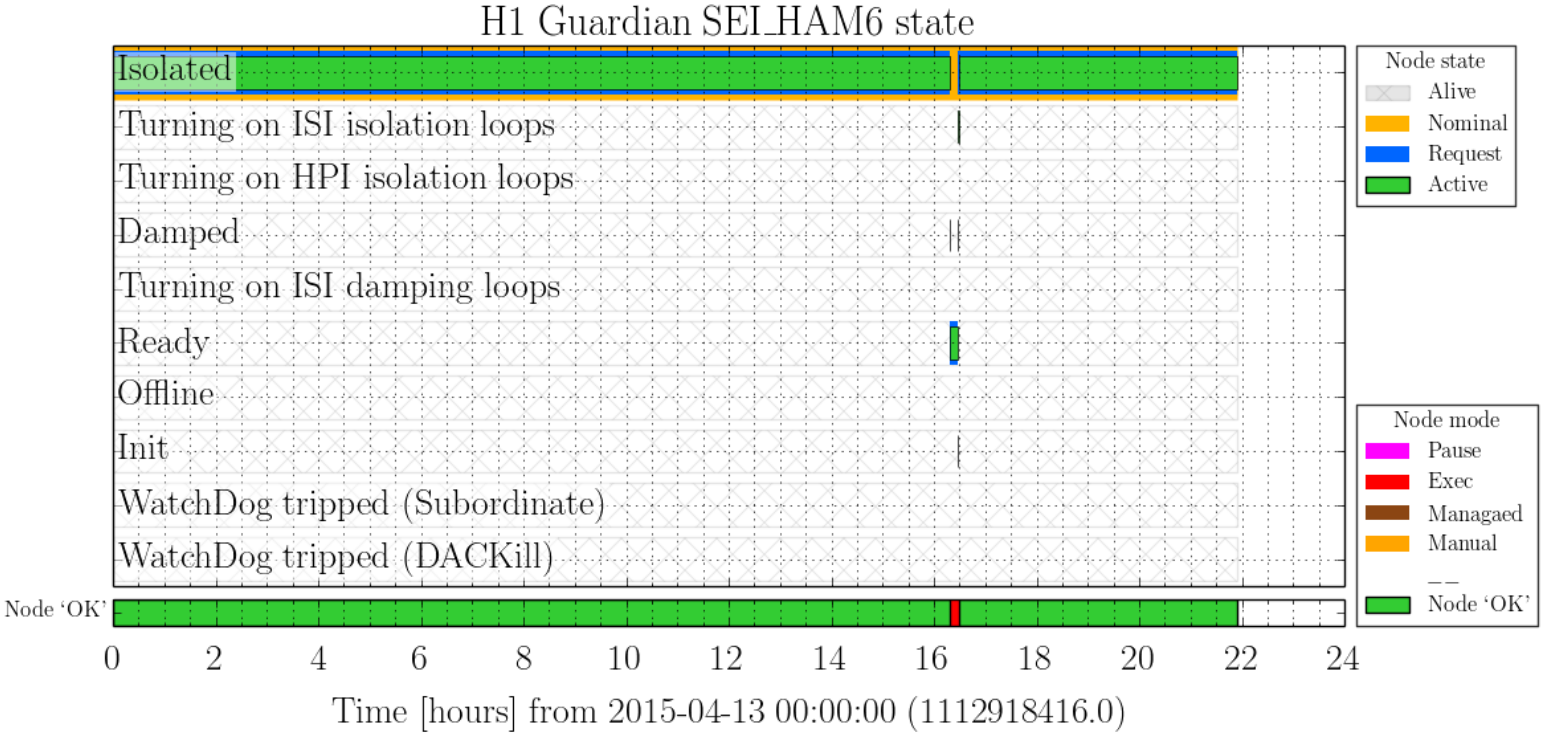
Since the control channels and state readbacks are all in EPICS, the **full state of all Guardian nodes is recorded in the frames**.



This enables **detector characterization** to accurately track the states of all subsystems over time.

Tracking Guardian state

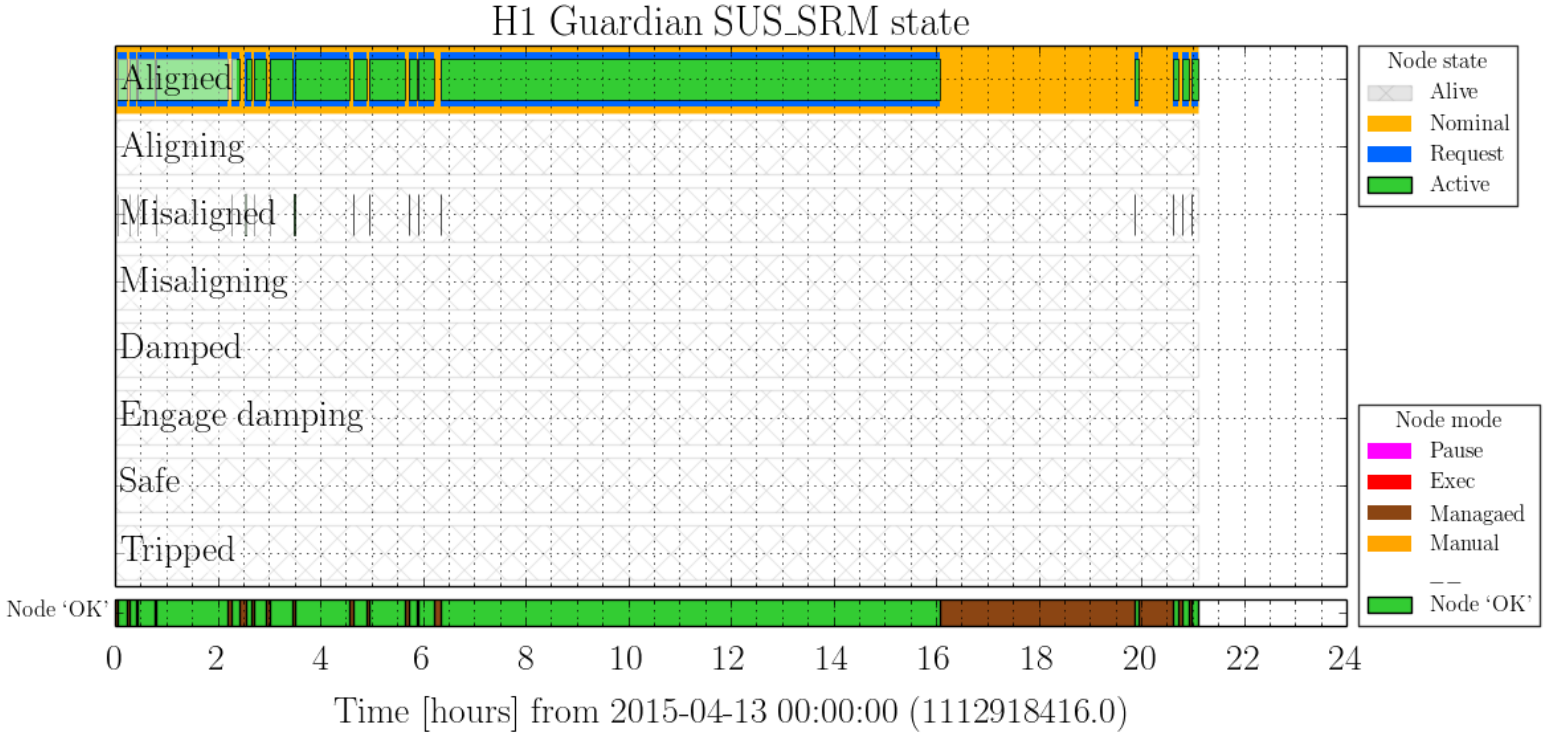
Since the control channels and state readbacks are all in EPICS, the **full state of all Guardian nodes is recorded in the frames**.



This enables **detector characterization** to accurately track the states of all subsystems over time.

Tracking Guardian state

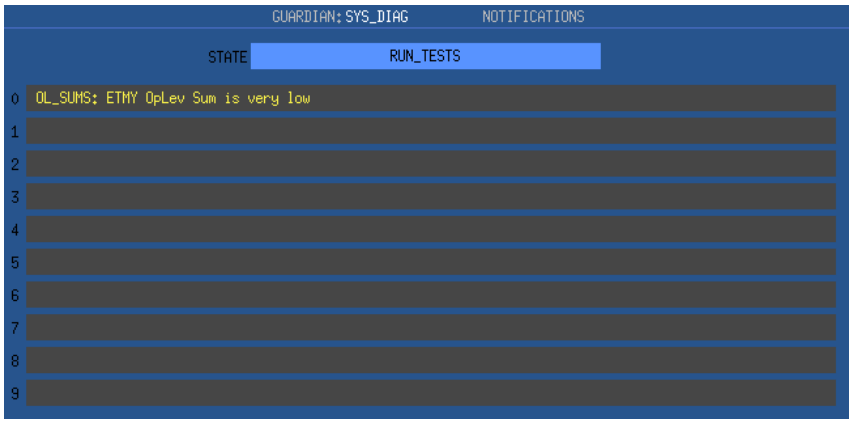
Since the control channels and state readbacks are all in EPICS, the **full state of all Guardian nodes is recorded in the frames**.



This enables **detector characterization** to accurately track the states of all subsystems over time.

Guardian diagnostics

Guardian is also providing useful **diagnostic** about the instrument, from each individual node, and from special *diagnostic nodes*.



Additional useful Guardian interfaces

Guardian also comes with many useful command line and graphical tools for:

- automatically drawing system graphs
- inspecting system code
- directly executing code from individual states or arbitrary graph paths
- interactive *guard shell* environment

Graphical visualization of system states in the form of **state graphs** has proven to be **very valuable**.

Conclusion

Conclusion

Automation of Advanced LIGO with the new Guardian automation platform is working quite well.

Full lock has been achieved at both observatories (H1, L1) with Guardian.

Guardian is also providing the primary readback for the full state of the instruments and all subsystems. This is being actively used for **detector characterization** and **analysis**.

The platform is mostly "feature complete", and we are busy integrating the remaining subsystems into the full automation framework.

Thank you