Snakes on a Rocket

Because "Python in an Avionics Testing System" doesn't sound quite as cool.

Lucas Mehl

Disclaimers

- These slides have been approved for release by NASA and/or Jacobs ESSSA.
- BUT...
- Any views or opinions expressed in this talk do not necessarily represent those of NASA, Jacobs ESSSA, Tuskegee University, or anyone other than me.
- AND...
- I apologize in advance for excessive use of memes

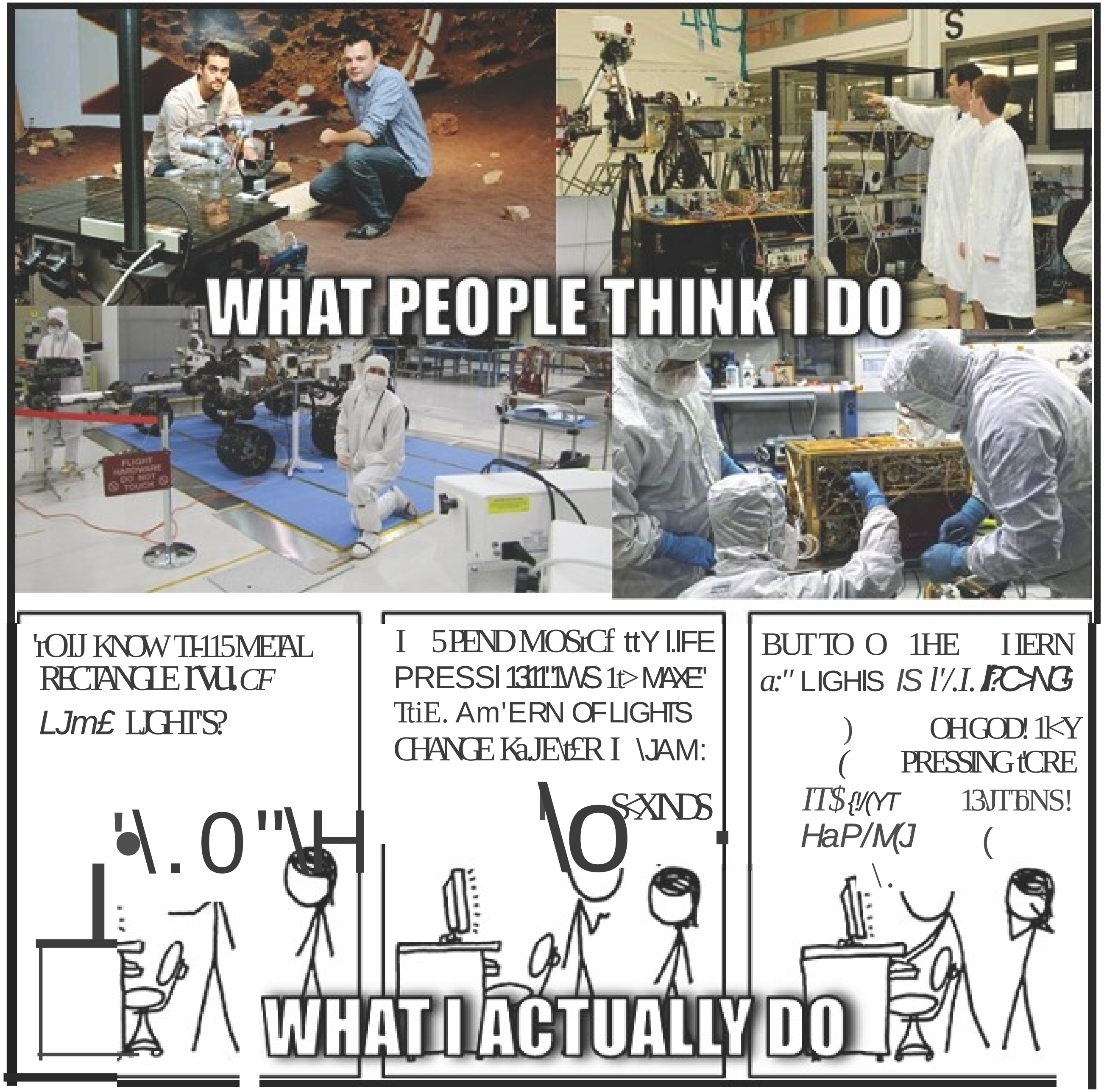
Agenda

- Me
- MAESTRO at a glance
- Context
  - History (Why we do what we do)
  - Space Launch System fun facts
  - What we're up against
- MAESTRO in depth (well, more in depth)
- Other Considerations (time permitting)

Me

- Aerospace engineering background
- Mostly self-taught programmer
- Linux user for 10+ years
- Python user for 2+ years
- NASA/Jacobs ESSSA/Tuskegee University
- github.com/LucasRMehl
- twitter.com/LucasRMehl
- But don't go there...

PyTennessee attendee every year since inception


WHAT PEOPLE THINK I DO
FLIGHT HARDWARE DO NOT TOUCH
WHAT I ACTUALLY DO

MAESTRO

Managed

Automation

Environment for

Simulation,

Test, and

Real-time

Operations

A Python-based automation framework that serves as the communication layer and the user interface for the Space Launch System's hardware-in-the-loop avionics testing.


SO I GUESS YOU COULD SAY THAT MAESTRO...
ORCHESTRATES
THE TESTING.
YEEEAAAHHHH!!!!

Avionics

What are avionics?

Avionics

Avionics => **Avi**ation electr**onics**

What Avionics Are:

1. Triple-redundant flight computers
2. Sensors. Lots of sensors.
3. Power supplies
4. Actuators (sometimes)

What Avionics Are Not:

Fire goes that way

(J/K, propulsion engineers, we <3 you)

But...

Even though we are only testing avionics, we still have to simulate the rest.

History of Rocket Testing

Wernher Von Braun

Mercury-Redstone Booster Development

January 31, 1961

Mercury-Redstone 2

Last test before we put a man into space

Ham The Chimp

Measure	Target	Actual
Apogee	115 miles	157 miles
Distance	292 miles	422 miles
Max Speed	4400 mph	5857 mph
Max g-force	11 g	14.7 g

Sad von Braun

What happened?

- Problem: A servo valve did not properly regulate flow of H_2O_2, making the fuel pumps overpowered, draining the fuel too fast, triggered abort when the engine chamber pressure dropped.
- **AVIONICS!**
- Solution: Replace the thrust regulator and velocity integrator (analog control system)
- Also, harmonic vibrations in topmost section due to aerodynamic stress, so they added stiffeners and whatever (not avionics).

For some reason, von Braun didn't want to put a human on the very next rocket.

March 24, 1961

Mercury-Redstone BD

Went as well as rocket launches could in those days

- No launch holds
- Within 1% of altitude target
- Within 2% of distance target

The next day, March 25, 1961, the USSR successfully launched and recovered another dog, making their record three out of five.

Zvezdochka ("Starlet")

60%? Good enough, comrade!

Yuri Gagarin became first man in space on April 12, 1961.

Three weeks later...

May 5, 1961
Mercury-Redstone 3
Alan Shepard became the first American in space.

Alan Shepard could have been first...

...but we weren't sure about the safety
...and if we're disregarding safety anyway, Yuri Gagarin could also have gone up instead of one of Starlet's predecessors.

Lessons from von Braun and the Space Race

Safety/Reliability is kinda important

- For aircraft and spacecraft, safety/reliability over everything, including schedule
- For other software...probably not (reliability over features?)

Beware problems hiding problems

Spaceflight is hard

30 astronaut fatalities

150+ non-astronaut fatalities

32 non-fatal flight incidents

35 non-fatal training incidents

Hundreds and hundreds of launch failures

How do we deal with that?

Simulations.

Flight computers

Redstone: electromechanical autopilot manufactured by Waste King Corp, manufacturer of garbage disposals and waste incinerators

Atlas D: solid-state analog autopilot

Saturn V: 0.0012 MIPS

SLS: triple redundant, hundreds of MIPS*

*Not sure of exact numbers. Orion is 480 MIPS, but reliability is far more important. Orion has triple redundant flight computers, plus a 4th computer that can take over in emergency crew survival and return situations

In comparison

Raspberry Pi 2: 1,186 MIPS

Core i7 5960X: 238,310 MIPS

Tianhe-2 supercomputer: ~30,000,000,000 MIPS

Launch capacities

Redstone: 0 lbs to LEO

Atlas D: 2900 lbs to LEO

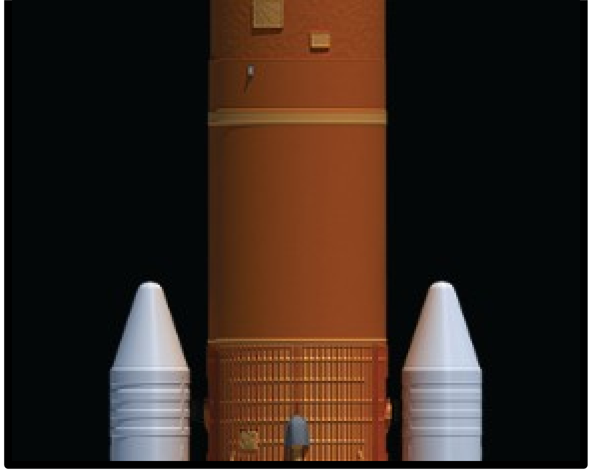
Saturn V: 260,000 lbs to LEO

SLS Block 1: 150,000 lbs to LEO

SLS Block 2: 290,000 lbs to LEO

#ThingsSLSCouldLaunch

3 adult male sperm whales



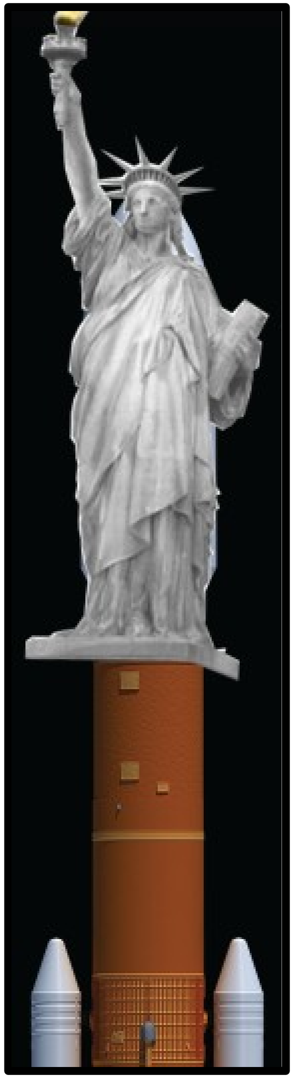
#ThingsSLSCouldLaunch

One year's worth of (legally) consumed marijuana in Colorado

Not allowed to show a picture, but it's a lot.

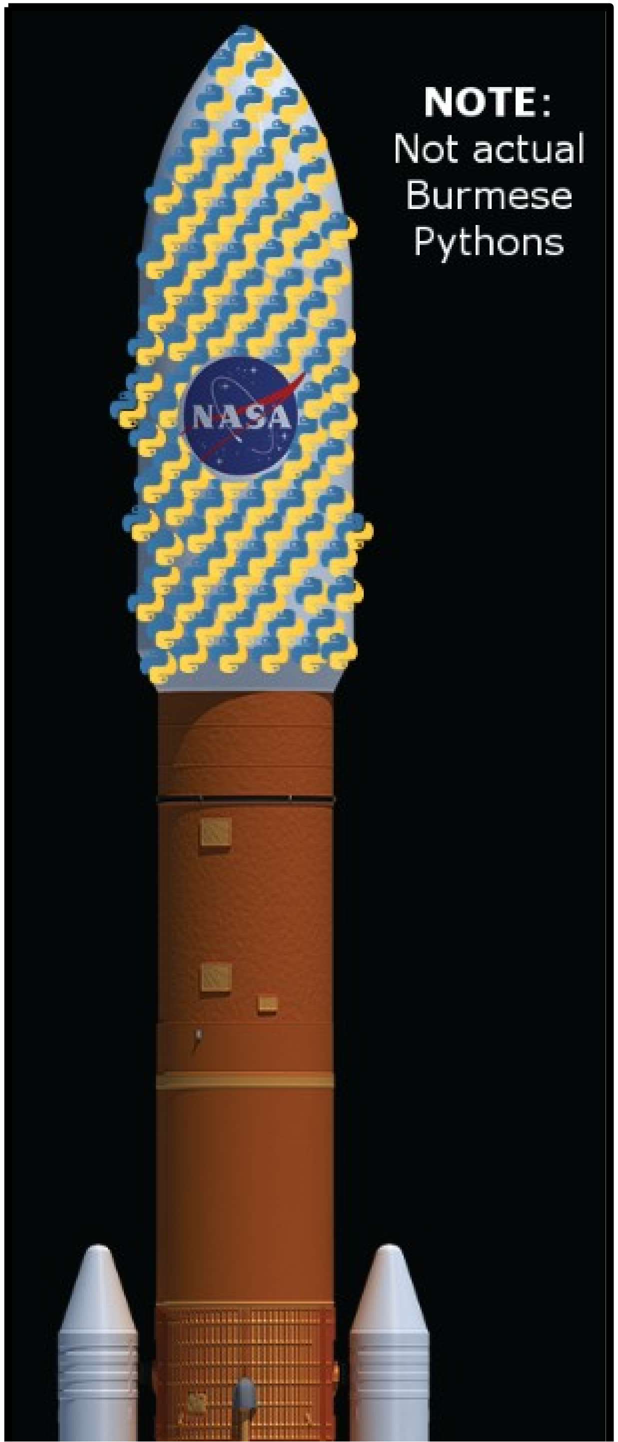
#ThingsSLSCouldLaunch

The steel frame of the Statue of Liberty, plus half of the copper skin



#ThingsSLSCouldLaunch

1,500 very large Burmese Pythons


NASA
NOTE:
Not actual
Burmese
Pythons

SLS: When the Force Awakens

SLS Block I: 8.4 million pounds of thrust

If I could apply that directly to me, I (or what's left of me) would be traveling 800,000 miles per hour after one second.

SLS Block II: 9.2 million pounds of thrust

Inertial dampeners FTW?

So that's what we're up against in terms of physics.

What about in terms of avionics?


GSDO

Space Launch System Avionics

- 26 avionics boxes
- 3 flight computers
- Need to be able to test Hardware-in-the-Loop

Software Groups Involved in Avionics Testing

ARTEMIS (simulates all of the hardware)

MAESTRO (hardware/software interface, user interface)

Flight Software* (controls vehicle)

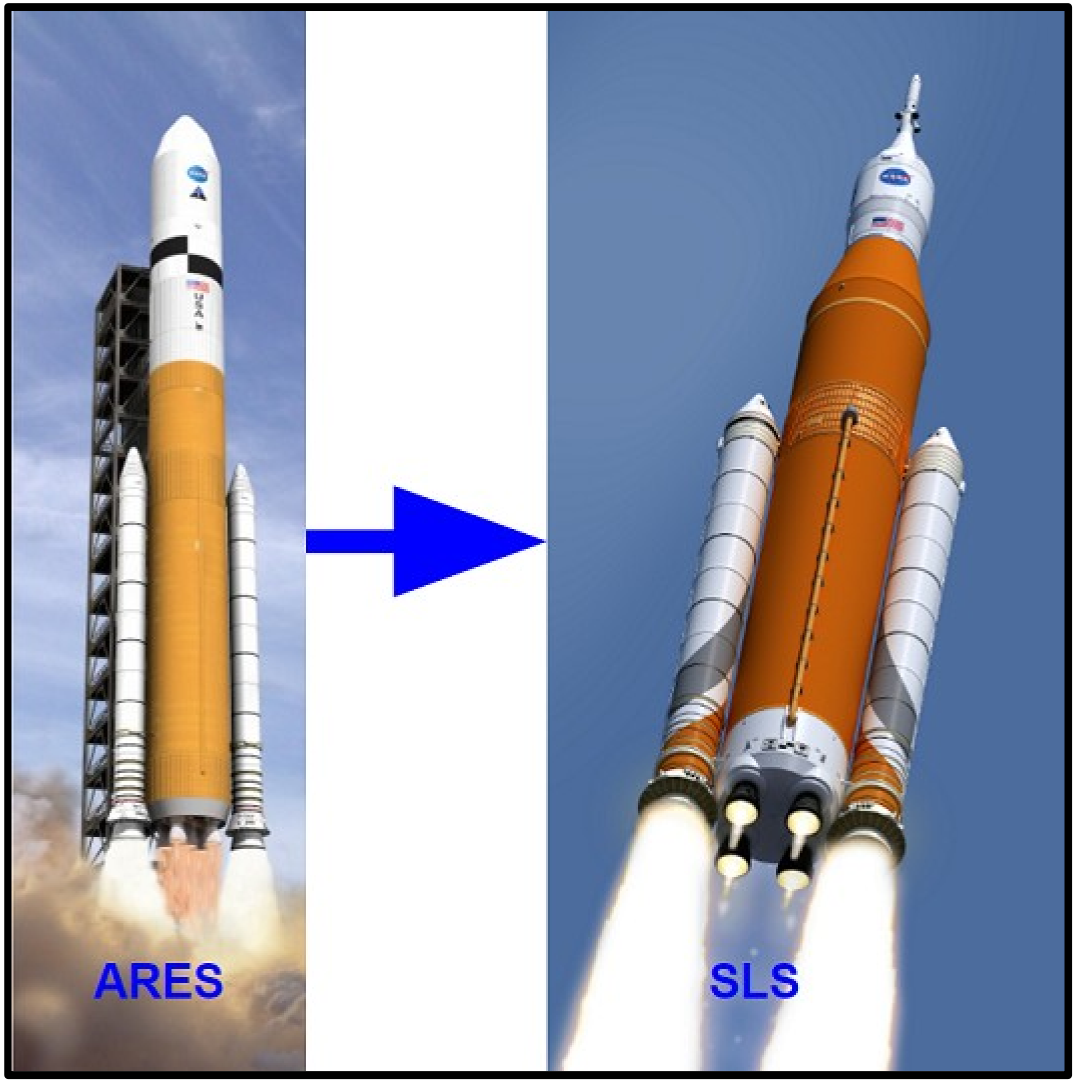
*Not cool enough for punny acronym

Software Test and Software Quality**

**Definitely not cool enough for punny acronyms or even their own bullet

ARTEMIS

Advanced
Real-
Time
Ehhh
Mumblemumble
Information?
Simulation, probably
Let's play: spot the pun*!
*Not actually a pun.



How about now?

Siblings!

Anyway, ARTEMIS simulates everything. Models, models, everywhere.

So we've got many of the pieces:

1. Avionics boxes & flight computers
2. Software models of all of them
3. Software models of the rocket itself
4. Flight software

Now what?

We need computing power to run those models.

20+ facilities, most with:

1-2 Windows VMs (test control & monitoring)

1 High-end Windows or Linux desktop (visualization)

1-2 CentOS Linux VMs (the MAESTRO "Configuration Manager" & facility manager)

6-16 Redhawk Linux 12-20 CPU core rack-mounted PCs (simulation & data recording)

But that's not all!

MAESTRO also needs to act like an SLS emulator, i.e. receive commands from test control software run at other places (e.g. NASA Johnson, Lockheed Martin).

So, why Python?

One language to rule them all

- Core Services (communication, IO, transfer protocol)
- Test Control (scripting, command implementation)
- Test Monitoring (real-time data collection and monitoring, distribution)
- Data Analysis
- GUIs

Well-supported on Windows and Linux

Ease of development

Maintainability

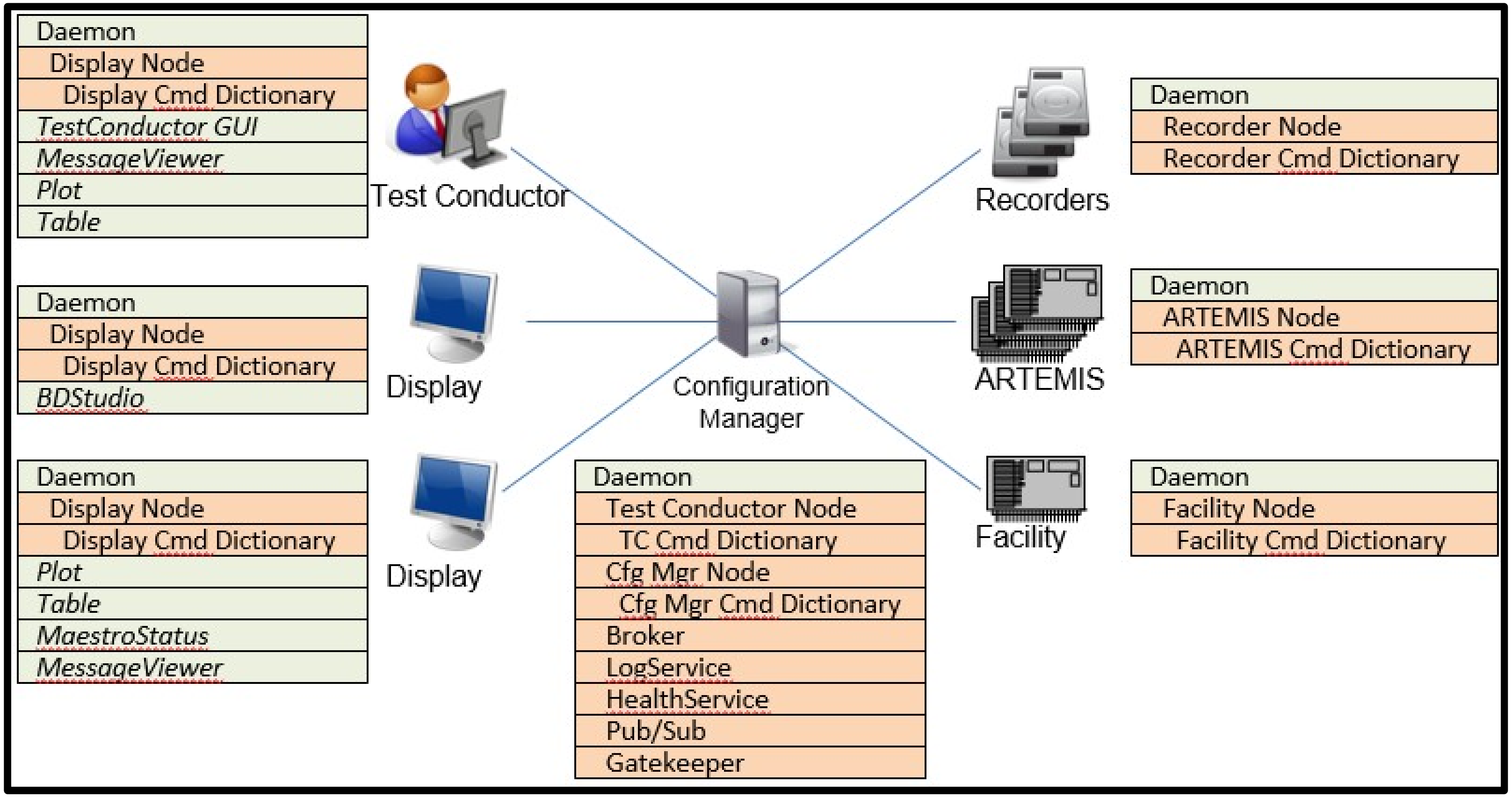
Extensibility

PyPI

Community/philosophy

Now, MAESTRO ...orchestrate!

MAESTRO



MAESTRO Architecture

- Point-to-Point Communication with Broker
- Telemetry Service
- Health Service
- Log Service

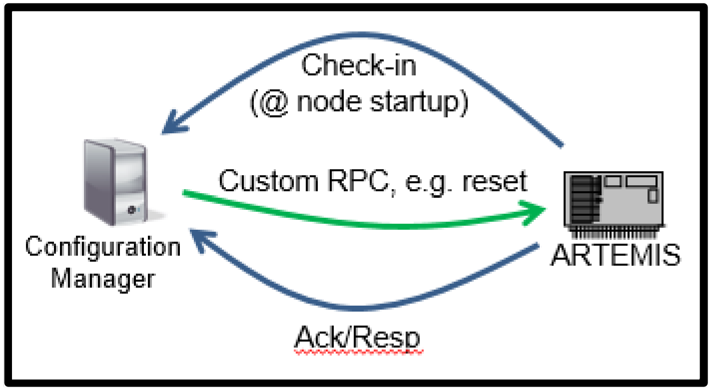
Point-to-Point:

Custom asynchronous RPC mechanism

- Developed by several SLS stakeholders
- Important for acting as an SLS emulator

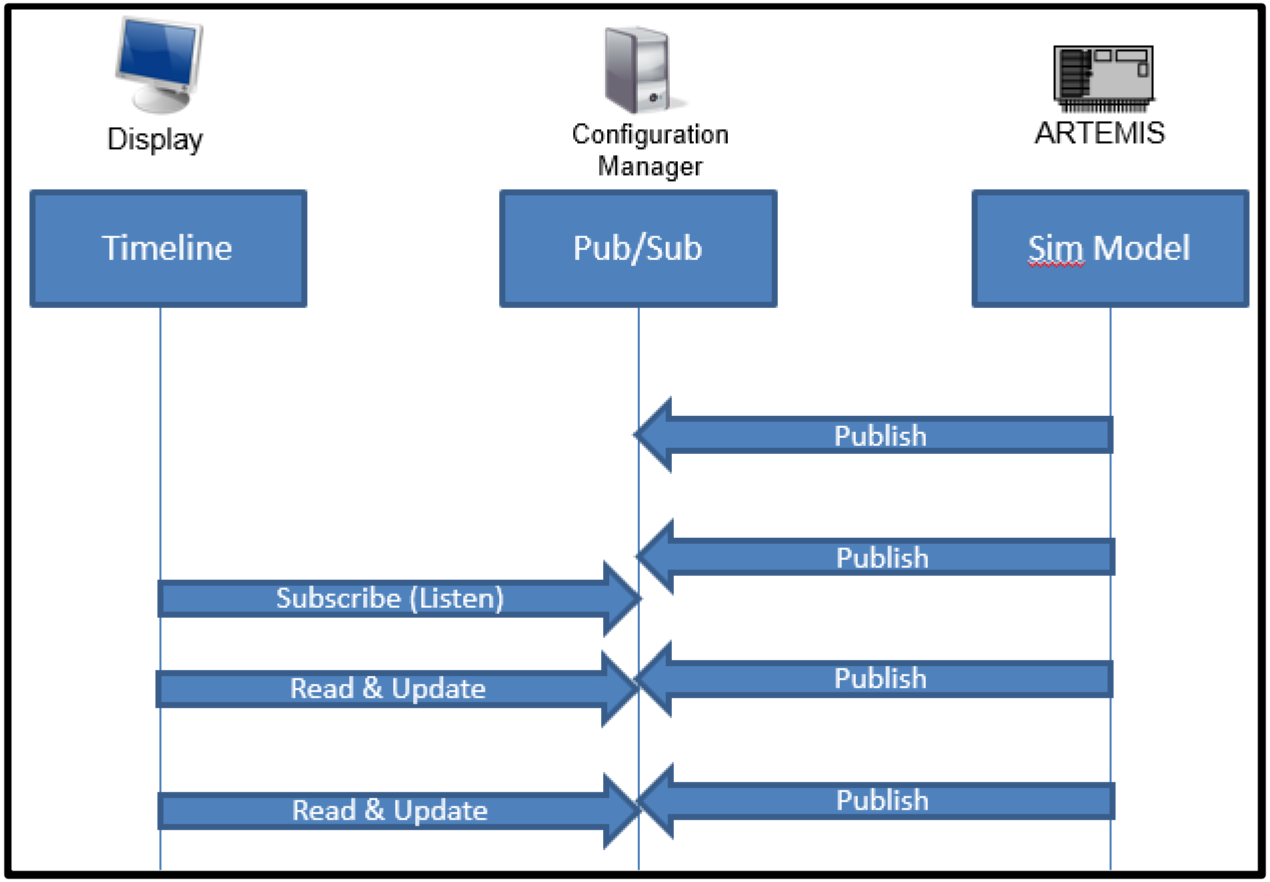
Defined in high-level node configuration

- Defines Broker IP, port
- Defines own IP, port
- Defines data archiving location



Telemetry Service:

Twisted
Simple Text-Oriented Messaging Protocol (STOMP)
Publish/subscribe mechanism



Health Service and Log Service:

Twisted

Same custom RPC mechanism as before

MAESTRO can act upon health events (e.g. stop test on FATAL)

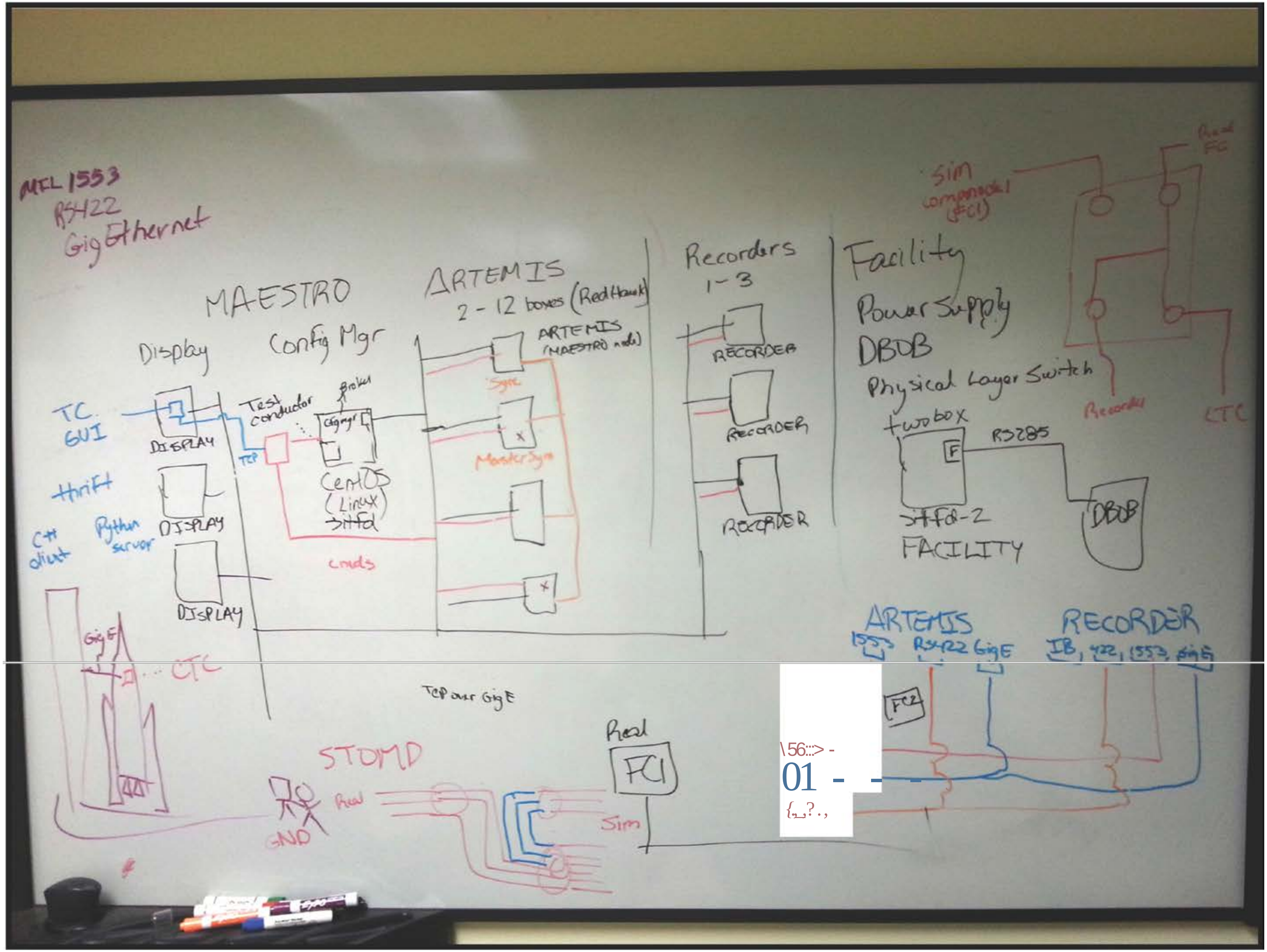
Asynchronous logging from multiple machines to one log file on one machine

We also need to (optionally) talk to all that hardware.

- Physical Layer Switches
- Custom break-out boxes
- Power Supplies
- These things allow MAESTRO to switch between real and simulated hardware without moving cables around

We can test individual pieces of hardware from different vendors without having issues from other hardware affecting the test

Our team also develops the facility data acquisition and monitoring system


MIL 1553
RS422
Gig Ethernet
MAESTRO
ARTEMIS
2 - 12 boxes (RedHawk)
Recorders
1 - 3
Facility
Power Supply
DBDB
Physical Layer Switch
Display
Config Mgr
TC
GUI
DISPLAY
DISPLAY
DISPLAY
Thrift
C#
client
Python
server
Test
conductor
TCP
CentOS
(Linux)
cmds
Sync
RECORDER
RECORDER
RECORDER
two box
RS285
F
DBDB
FACILITY
Sim
Recorder
CTC
ARTEMIS
RECORDER
1553
RS422
GigE
IB, 422, 1553, GigE
TCP over GigE
Real
FC1
FC2
Sim
STDMD
GND
Gig E
CTC

Configuration

XML

- Class generation
- Validation
- Used for all test-related config files

INI

- ConfigParser
- Used for GUIs

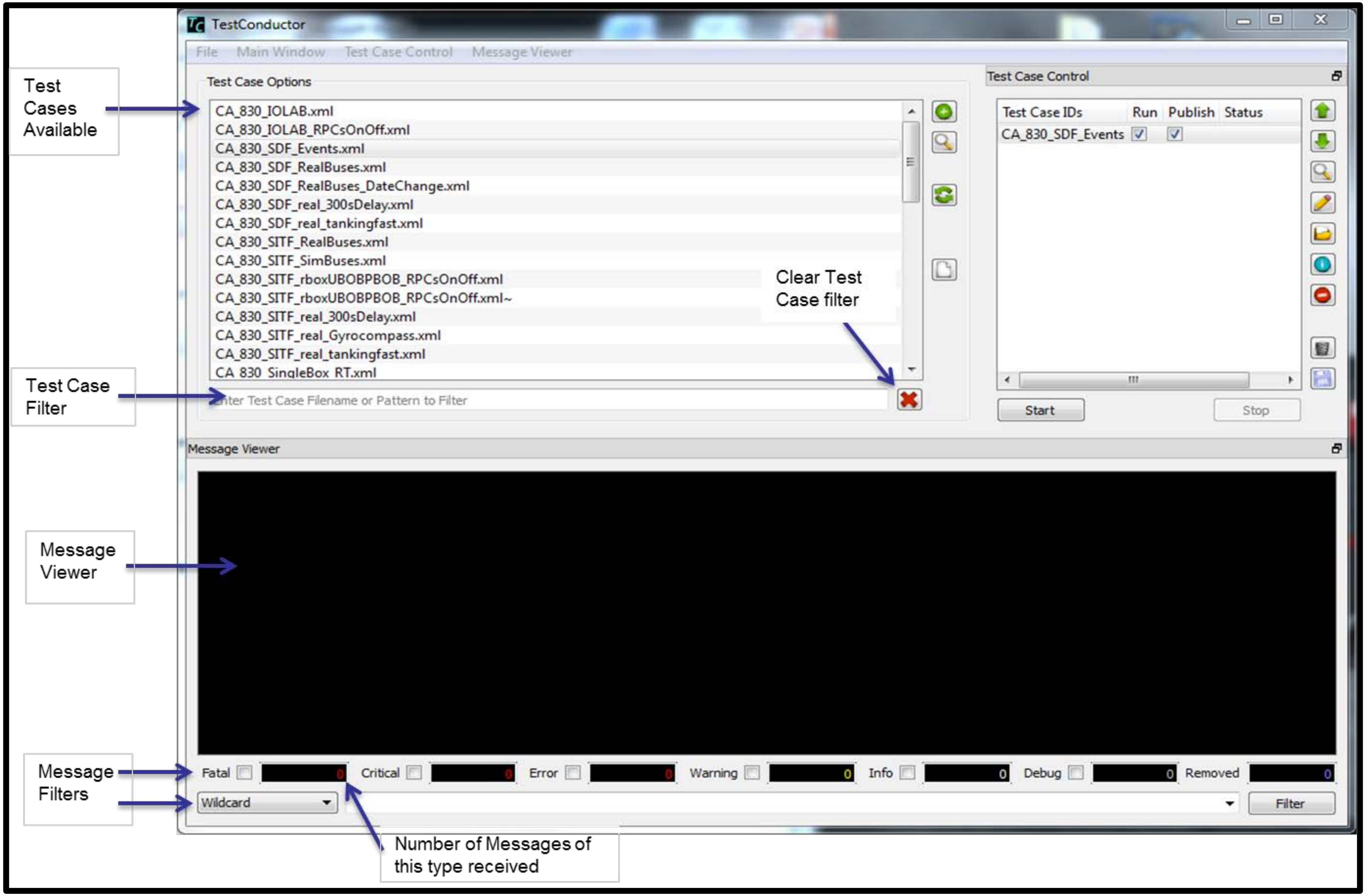
JSON

- Command Dictionaries
- Used for self-test (dictionary serialization)

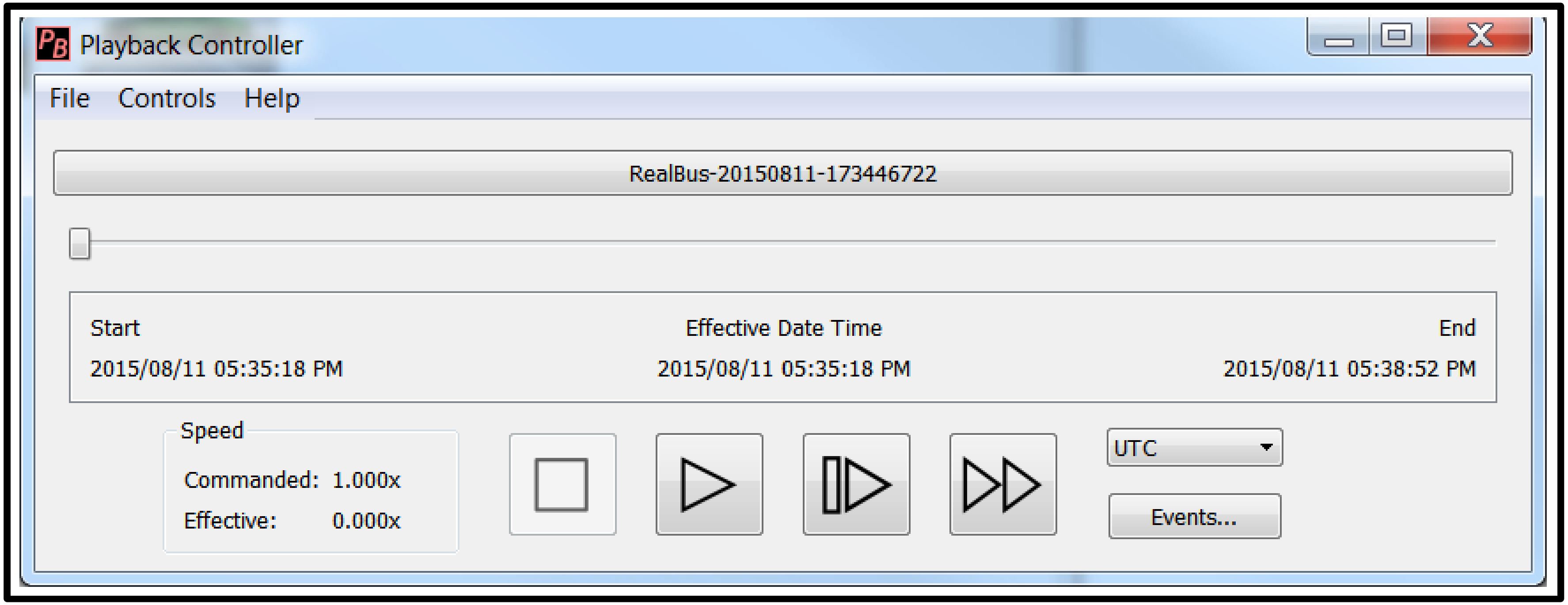
GUIs

Mostly PyQt
PyQwt
matplotlib
PyOpenGL

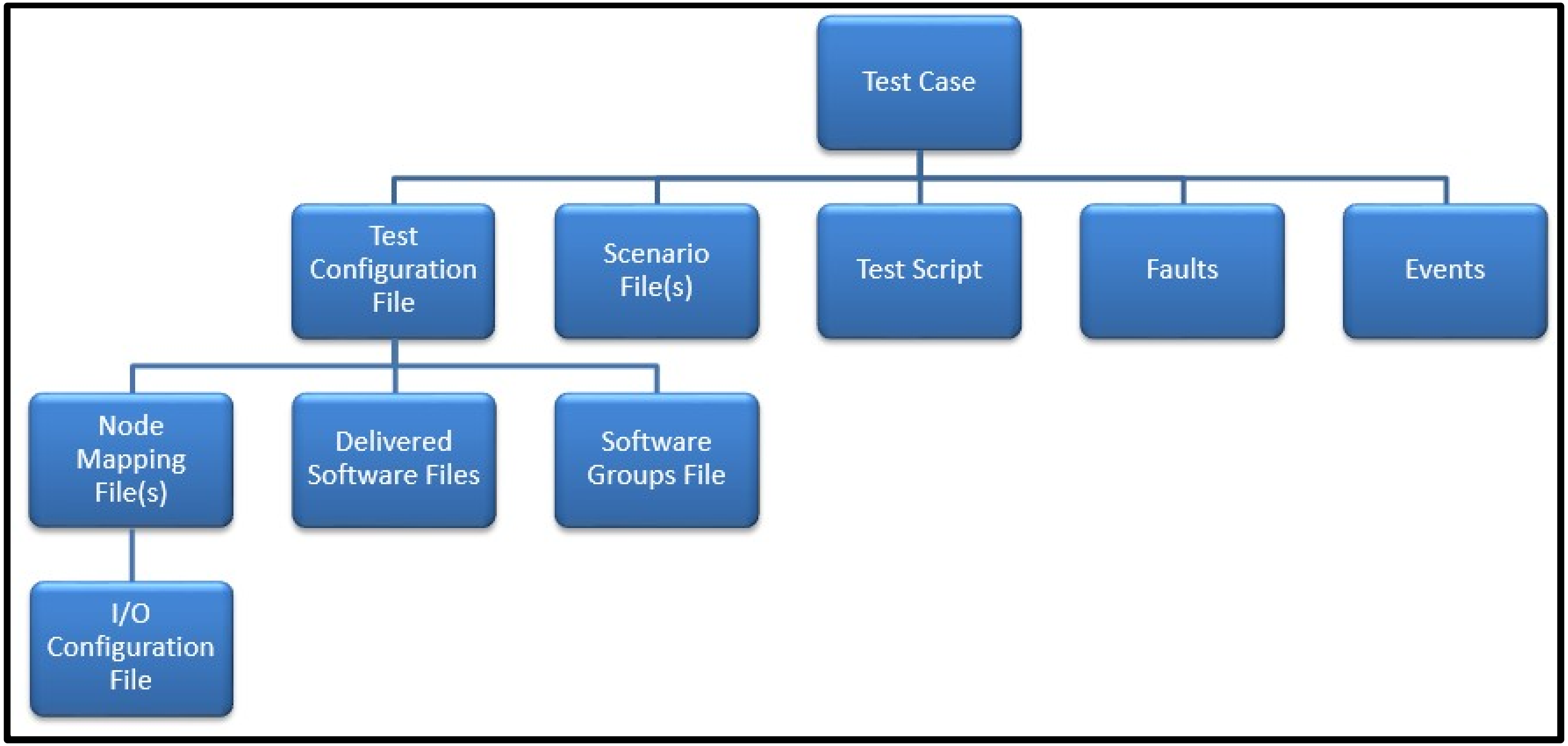
Test Conductor



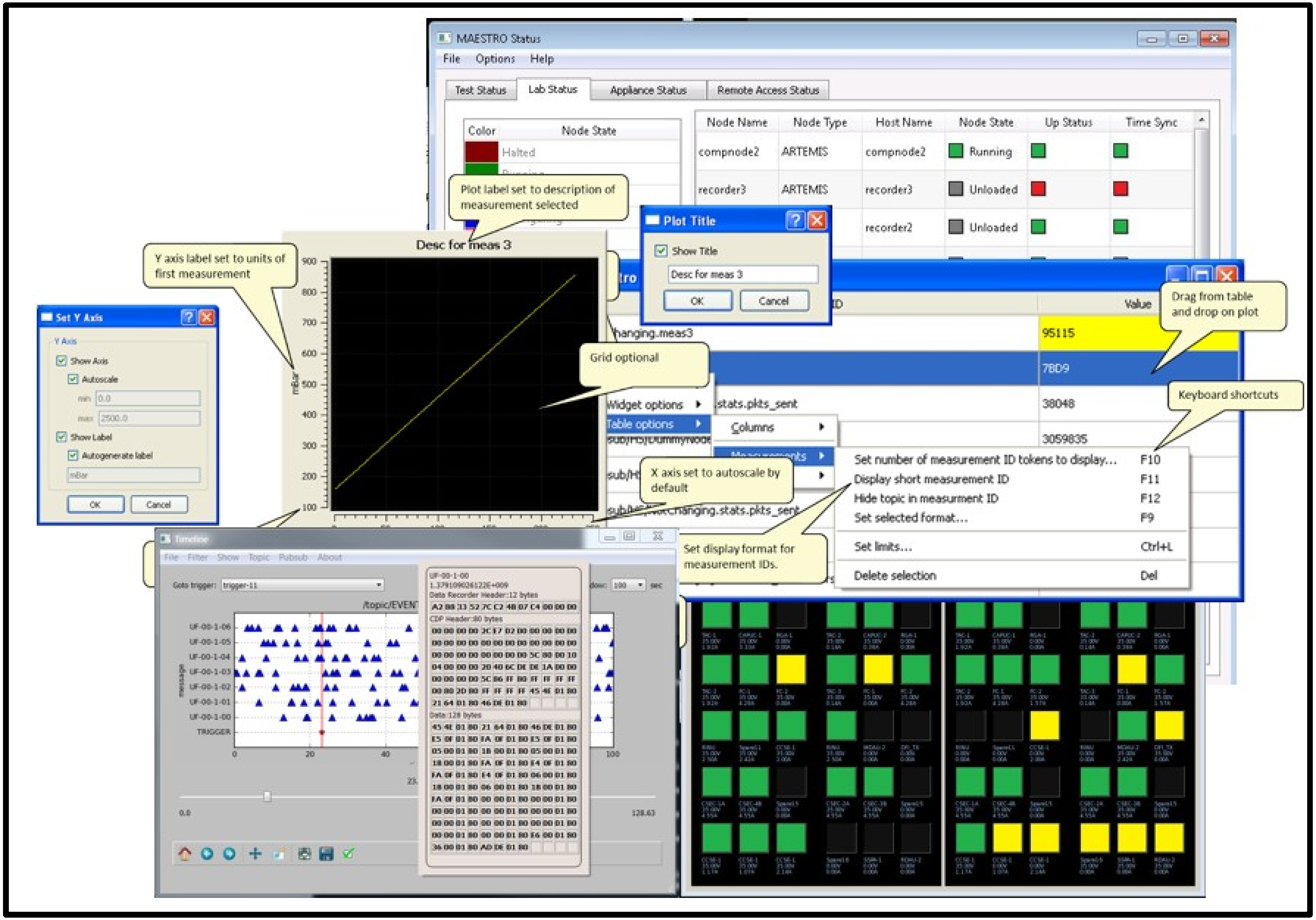
MAESTRO also supports data playback



Configuring Tests



Test Monitoring



Several GUIs ported from Java

- GUIs are more consistent
- Maintainance is easier
- Installation is easier
- Easier on developers, easier on operations

Speaking of operations...

MAESTRO
OPS

Remember those 20+ labs?

Troubleshooting

- Logs
- Bash
- Ansible
- Self-test (Ansible API)

Software Maintenance

- Ansible Playbooks
- Pip
- Wheels
- Virtualenv

Ansible

Got to see in action PyTN 2015!

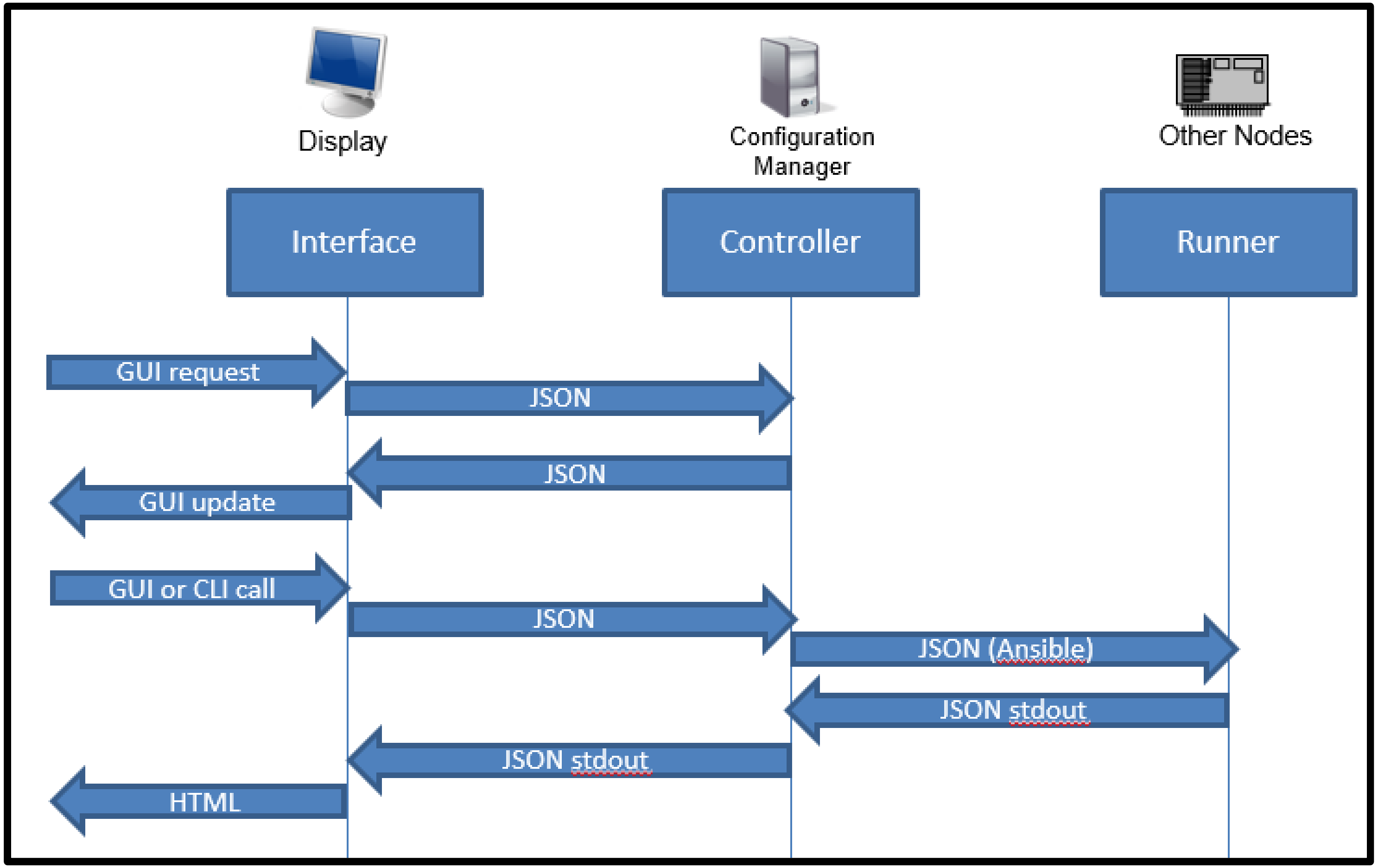
Super awesome for command line usage

Super awesome for installation procedures

Windows support is iffy

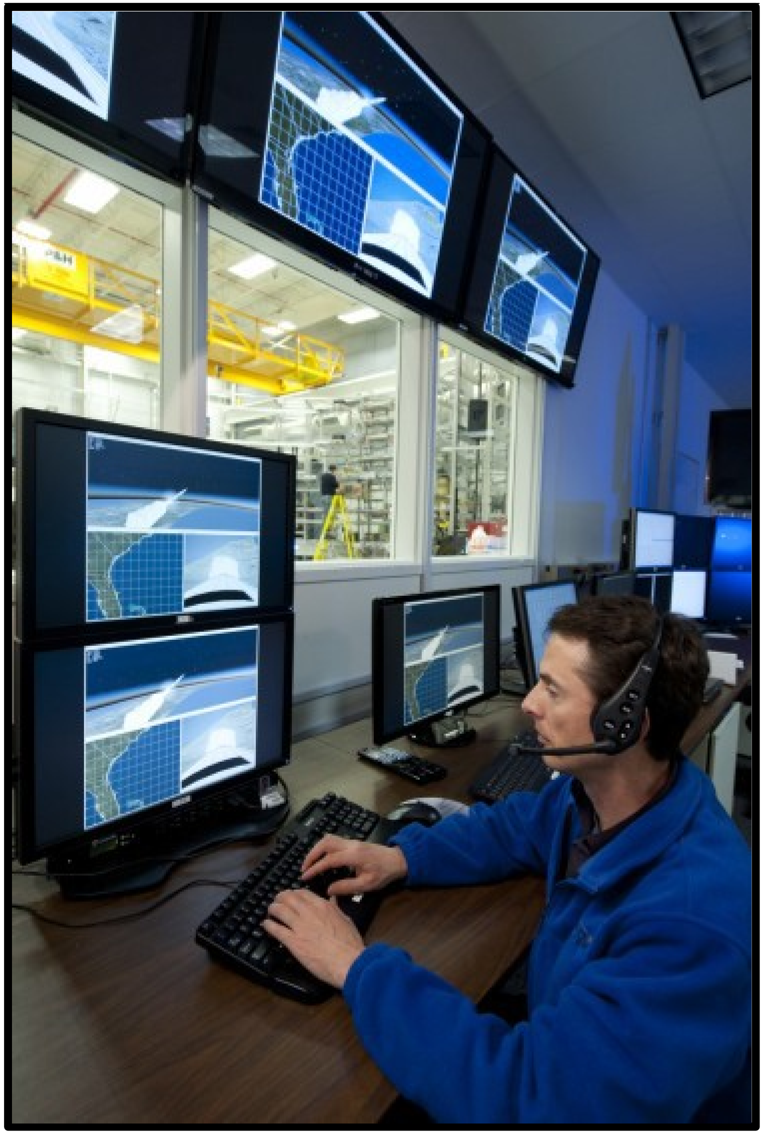
- Requires Powershell upgrade
- Requires service to be enabled
- Requires additional Python modules

MAESTRO Self-Test



Only problem: passing JSON as a command line argument

Bringing it all together:



Other Considerations

NASA <3's Open Source

https://github.com/nasa/

https://open.nasa.gov/

https://code.nasa.gov/

https://data.nasa.gov/

Development Process: It's Pretty Scrummy

Requirements filtered through product lead

Enhancement/bugfix requests from users & developers

- Much more common than in open source
- Less pressure to move to latest and greatest

Three week sprints

Less separation between Scrum Master and Developers

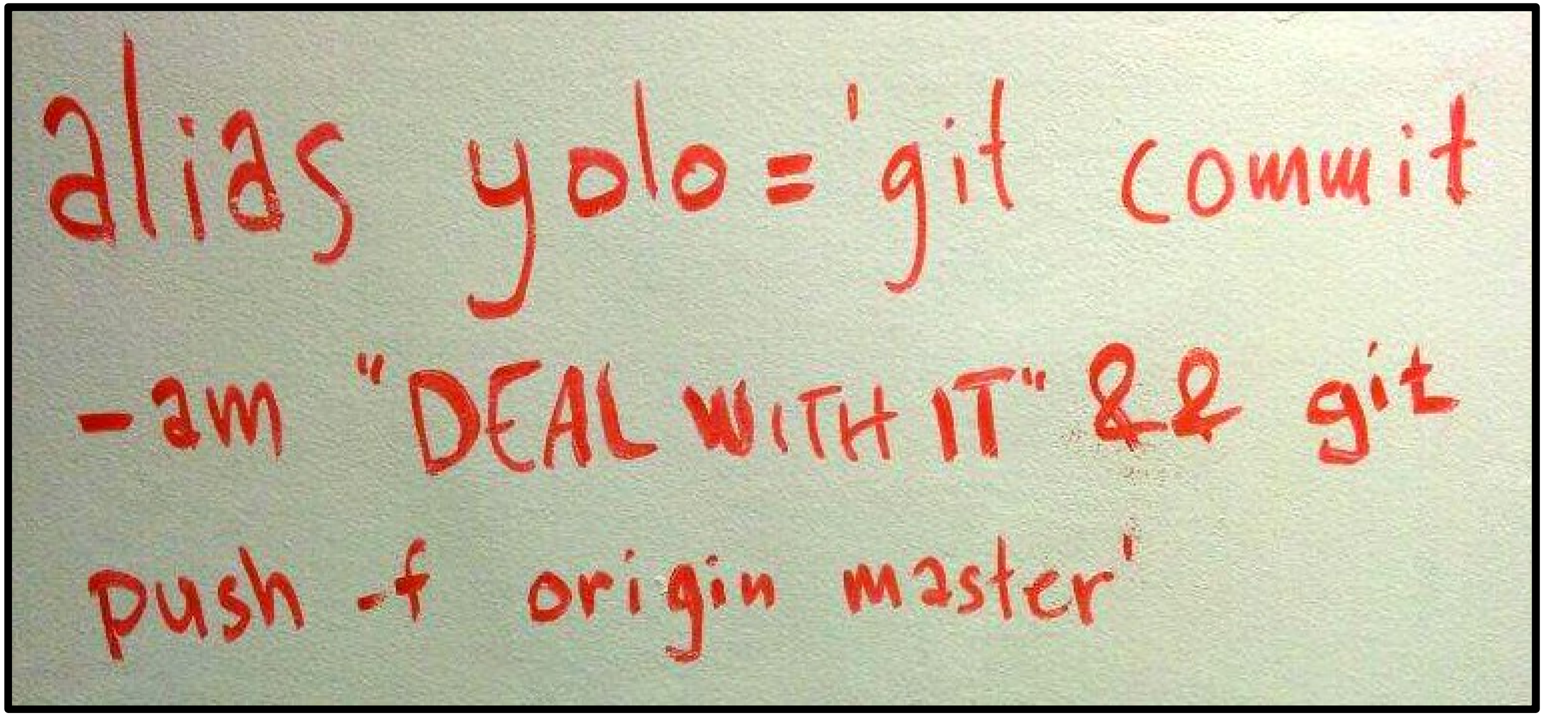
Less separation between Product Lead and Scrum Master

Releases are separate from sprints

Source Control

Subversion for binary, docs, releases

Git for code



Where do we go from here?

In terms of rockets…

In terms of Python…


I DON'T ALWAYS USE PYTHON
BUT WHEN I DO, I USE IT TO
BUILD THE MOST POWERFUL
ROCKET IN HISTORY

Thank you!

Questions?