END POINT

ROS-driven User Applications
in idempotent environments

Matt Vollrath (US) @mdevolving
Wojciech Ziniewicz (PL) @fribulusxax

Liquid Galaxy

What is it?

- content delivery platform
  - presentations (corporate content)
  - virtual tours (museums)
  - engaging environments
- immersive experience - from 0° to virtually 270°
- 3D content support
- interactive digital signage

Liquid Galaxy

END POINT

Input devices

- Logitech Space Navigator
- Generic touchscreen
- LEAP Motion, depth cameras, etc.

A Brief Story of Liquid Galaxy

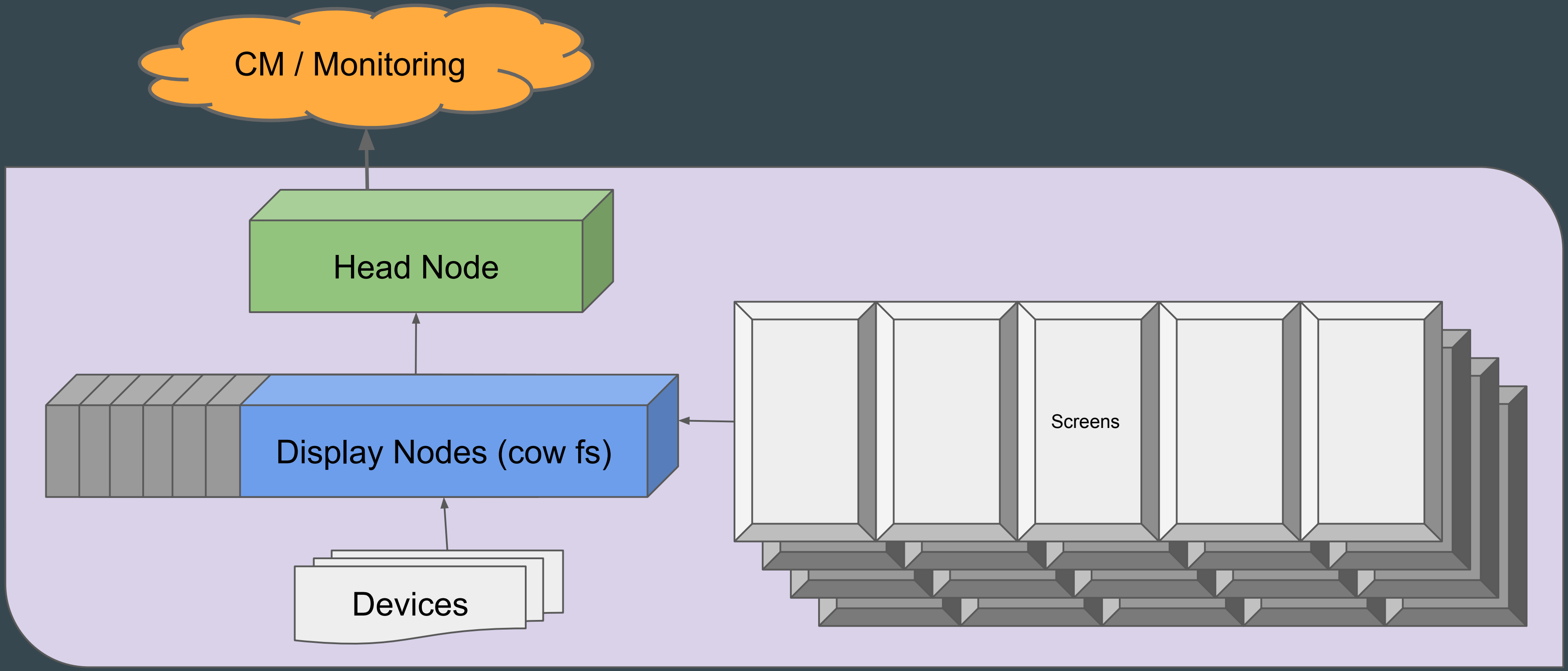
- 20% Project of Google engineer Jason Holt ca. 2010
- Brought to End Point Corp to deploy, scale and support
- From ad-hoc services/protocols to ROS in 2015

```
while :; do
  google-earth
done
```

Deploying Distributed ROS to Bare Metal

- A few hundred computers and growing
- Full automatic provisioning from USB seed
- Standard bombproof shipping unit
  - Fast setup, teardown

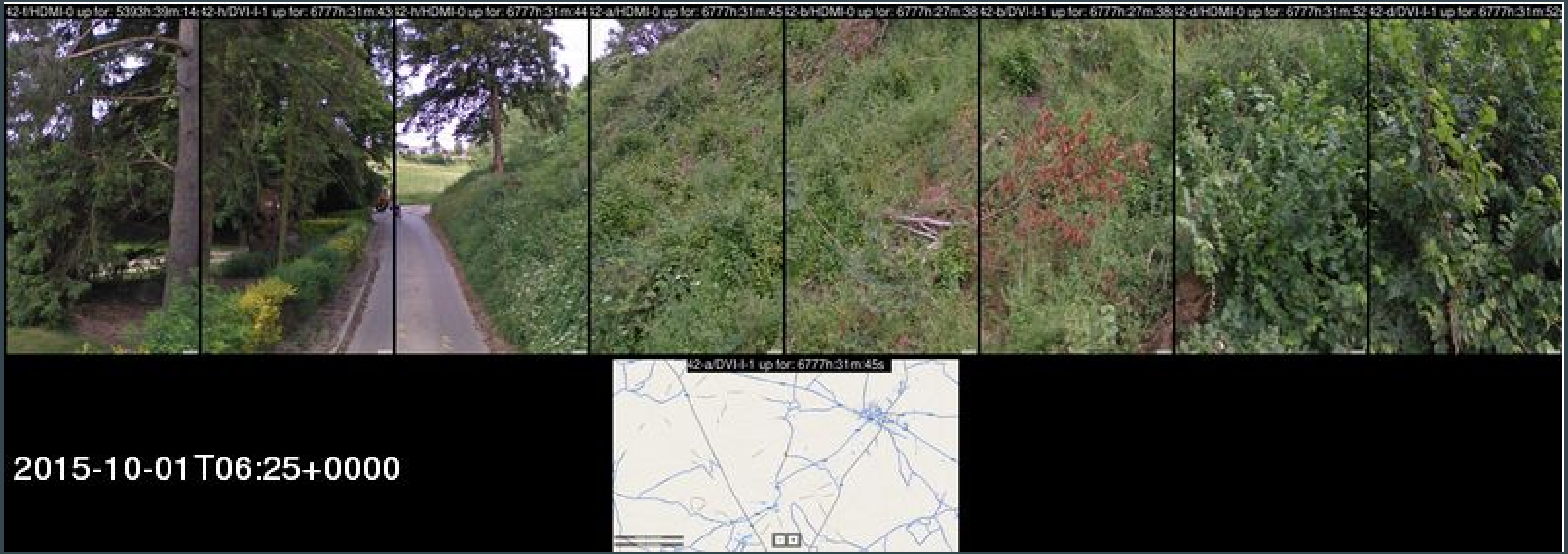
Systems Hierarchy



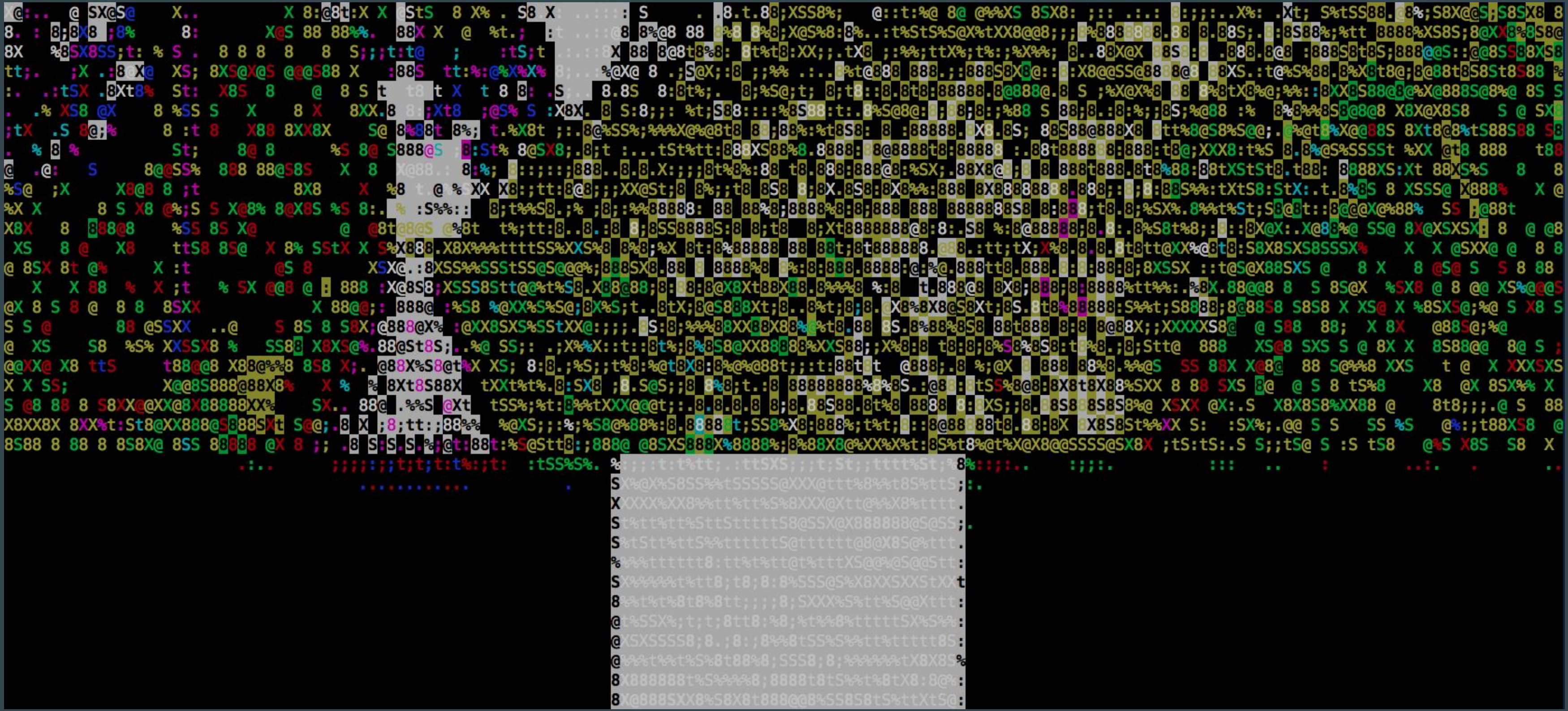
Monitoring and Error Handling

- Distributed Nagios with handlers
- Restart escalation: ROS, Xorg, power
- Live diagnosis via SSH

lg-slingshot


t2-f/HDMI-0 up for: 5393h:39m:14
t2-h/DVI-I-1 up for: 6777h:31m:43
t2-h/HDMI-0 up for: 6777h:31m:44
t2-a/HDMI-0 up for: 6777h:31m:45
t2-b/HDMI-0 up for: 6777h:27m:38
t2-b/DVI-I-1 up for: 6777h:27m:38
t2-d/HDMI-0 up for: 6777h:31m:52
t2-d/DVI-I-1 up for: 6777h:31m:52
t2-a/DVI-I-1 up for: 6777h:31m:45s
2015-10-01T06:25+0000

lg-ascii



Continuous Testing from bottom to the top

- ROS: catkin_make run_tests
- CM: Chef minitests
- ISO: Nested virtualbox to test DHCP/TFTP
- Selenium for roslibjs browser apps

Liquid Galaxy - internals

- https://github.com/EndPointCorp/lg_ros_nodes - it’s OpenSource - Apache License 2.0
- https://github.com/EndPointCorp/appctl - lower level generic library (OpenSource)
- ROS handles 100% of Liquid Galaxy’s logic and state
- Chef i used to manage the configuration and generate roslaunch xml files

Liquid Galaxy ROS nodes

https://github.com/EndPointCorp/appctl

A ROS node that provides support software for controlling processes on an system. The core concept of appctl is that each application configuration can run in any number of "Modes," with only one "Mode" active at any given moment.

It also provides low level library for controlling processes on machines (proc_runner.py).

Liquid Galaxy ROS nodes

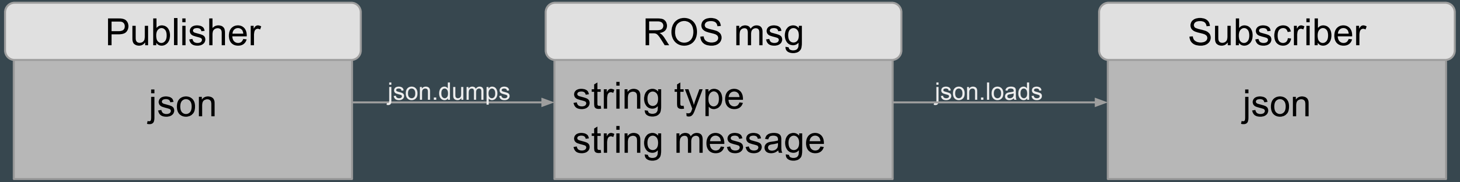
Open source: https://github.com/EndPointCorp/lg_ros_nodes

- **lg_common**: library with helpers, webserver for browsers, browsers management
- **lg_activity**: event driven state tracking from input devices
- **lg_attract_loop**: user-less content playback
- **lg_cms_director**: relay for content messages - “master of puppets”
- **lg_earth**: Google Earth(™)
- **lg_nav_to_device**: spacenavnode to /dev/input/v_spacenav
- **lg_replay**: send virtually any input device data to ROS
- **lg_sv**: street view support
- **lg_media:** synced video content support (Popcorn.js or good old mplayer)
- **liquidgalaxy** - Debian metapackage (ros-indigo-liquidgalaxy)

Sending complex data as json

Complex messages may be transported as json serialized to string:

interactivespaces_msgs.msg.GenericMessage:



Interactivespaces framework https://github.com/interactivespaces/interactivespaces

Lessons learned - DI and ROSless logic

How:

- delegate your logic to separate classes and make them as ROS agnostic as possible
- pass Subscribers and Publishers to them
- scripts must be thin - they should only contain initialization

Advantages:

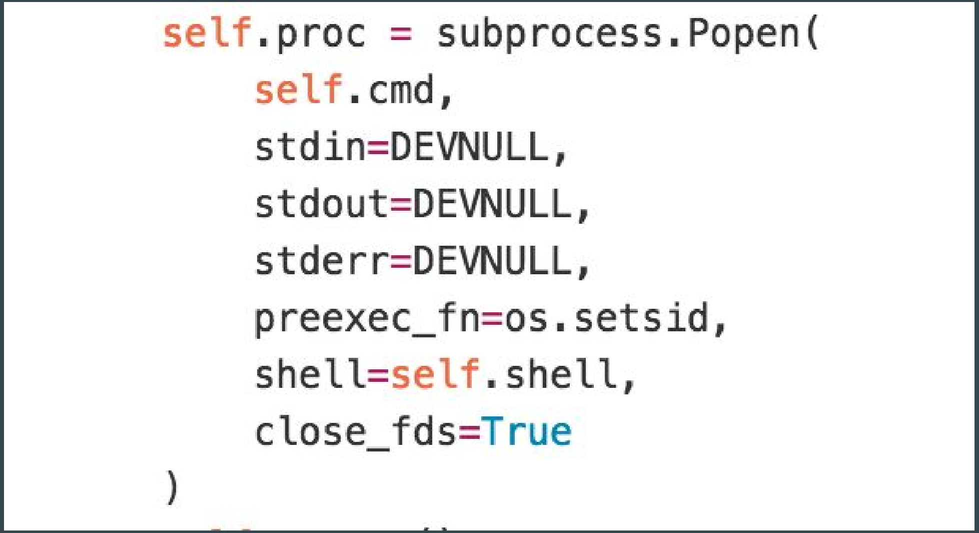
- writing unittests is simpler (offline tests are faster)
- your logic is easy to observe using mocks:
  lg_attract_loop/test/online/test_lg_attract_loop.py
- code is cleaner and less monolithic

Lessons learned

- use debian packages (don’t deploy catkin builds)
- keep previous versions of debs for rollback
- decouple everything
    - possibly blocking operations should be executed as separate tracked processes (appctl)
    - if it’s not possible - decouple them from the rest of your stuff
- automate everything: less entropy = higher quality
- tests are hard but they save your time
- TDD is your friend
- parametrize everything
    - you don’t need to deploy each time you need to change someting
    - data driven world is just better

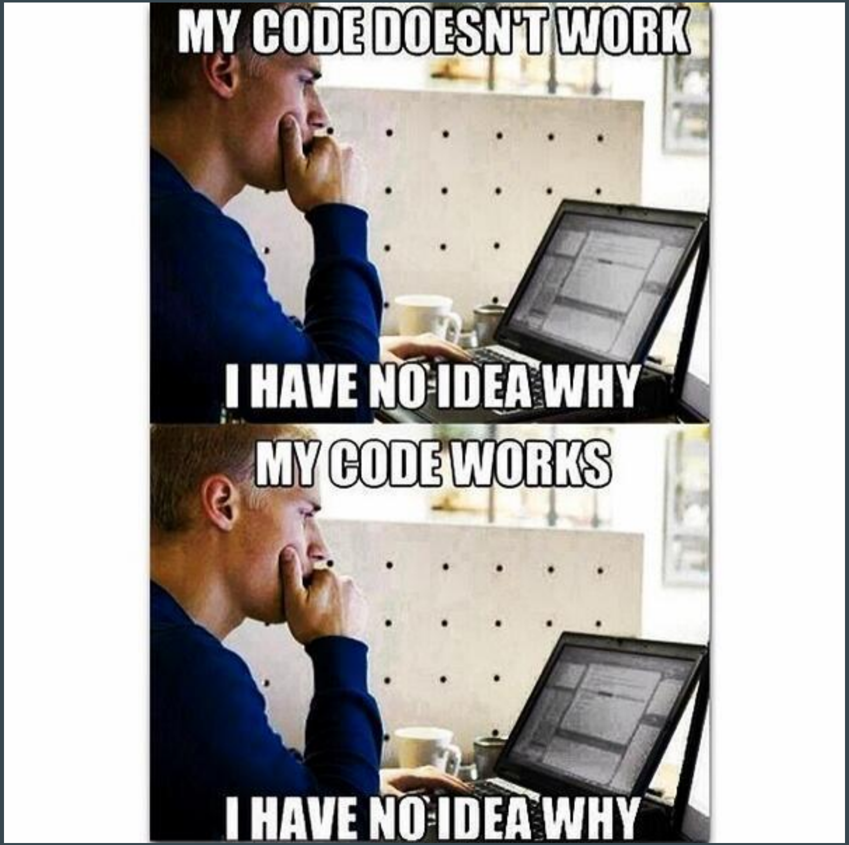
Lessons learned - the hard way

subprocess.Popen should have close_fds set to True

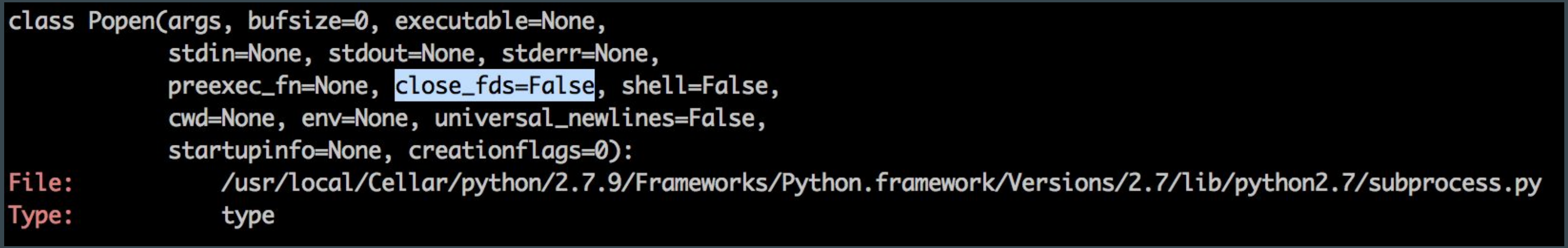


```
        self.proc = subprocess.Popen(
            self.cmd,
            stdin=DEVNULL,
            stdout=DEVNULL,
            stderr=DEVNULL,
            preexec_fn=os.setsid,
            shell=self.shell,
            close_fds=True
        )
```

Popen ‘close_fds’ - Bug or feature?



Lessons learned - the hard way



```
class Popen(args, bufsize=0, executable=None,
            stdin=None, stdout=None, stderr=None,
            preexec_fn=None, close_fds=False, shell=False,
            cwd=None, env=None, universal_newlines=False,
            startupinfo=None, creationflags=0):
File:           /usr/local/Cellar/python/2.7.9/Frameworks/Python.framework/Versions/2.7/lib/python2.7/subprocess.py
Type:           type
```

locking on IO in ROS is tricky

- http://bugs.python.org/issue7213 (2009)
- http://legacy.python.org/dev/peps/pep-0446/ (2013)
- http://legacy.python.org/dev/peps/pep-0433/ (2013)
- http://trac.edgewall.org/ticket/10991 (2012)
- https://github.com/denyhosts/denyhosts/issues/33 (2015)

Thank you