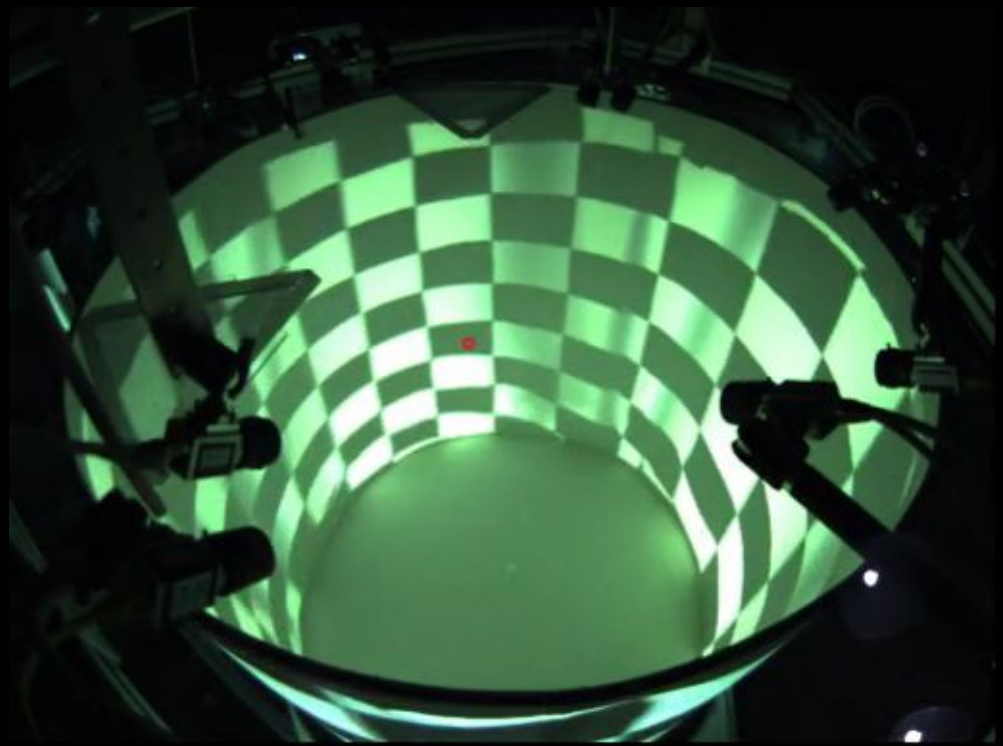
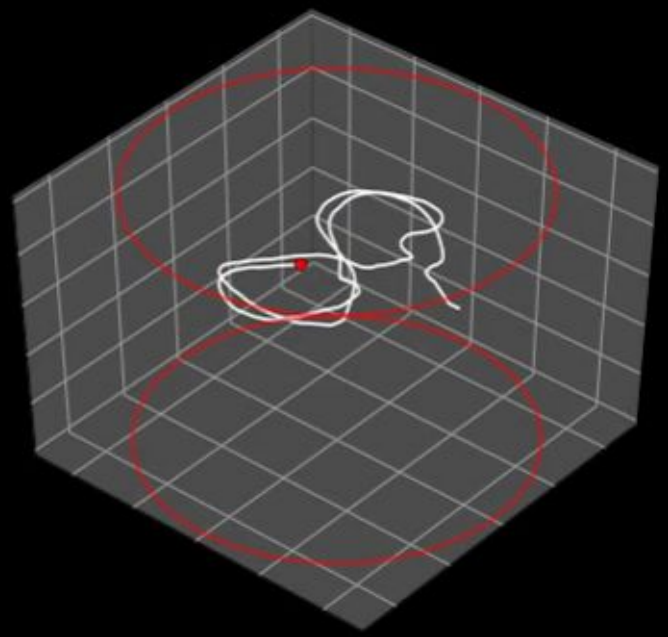
Managing Complex Experiments, Automation, and Analysis using Robot Operating System

John Stowers (TU Wien) • John.stowers@gmail.com • June 26th 2013

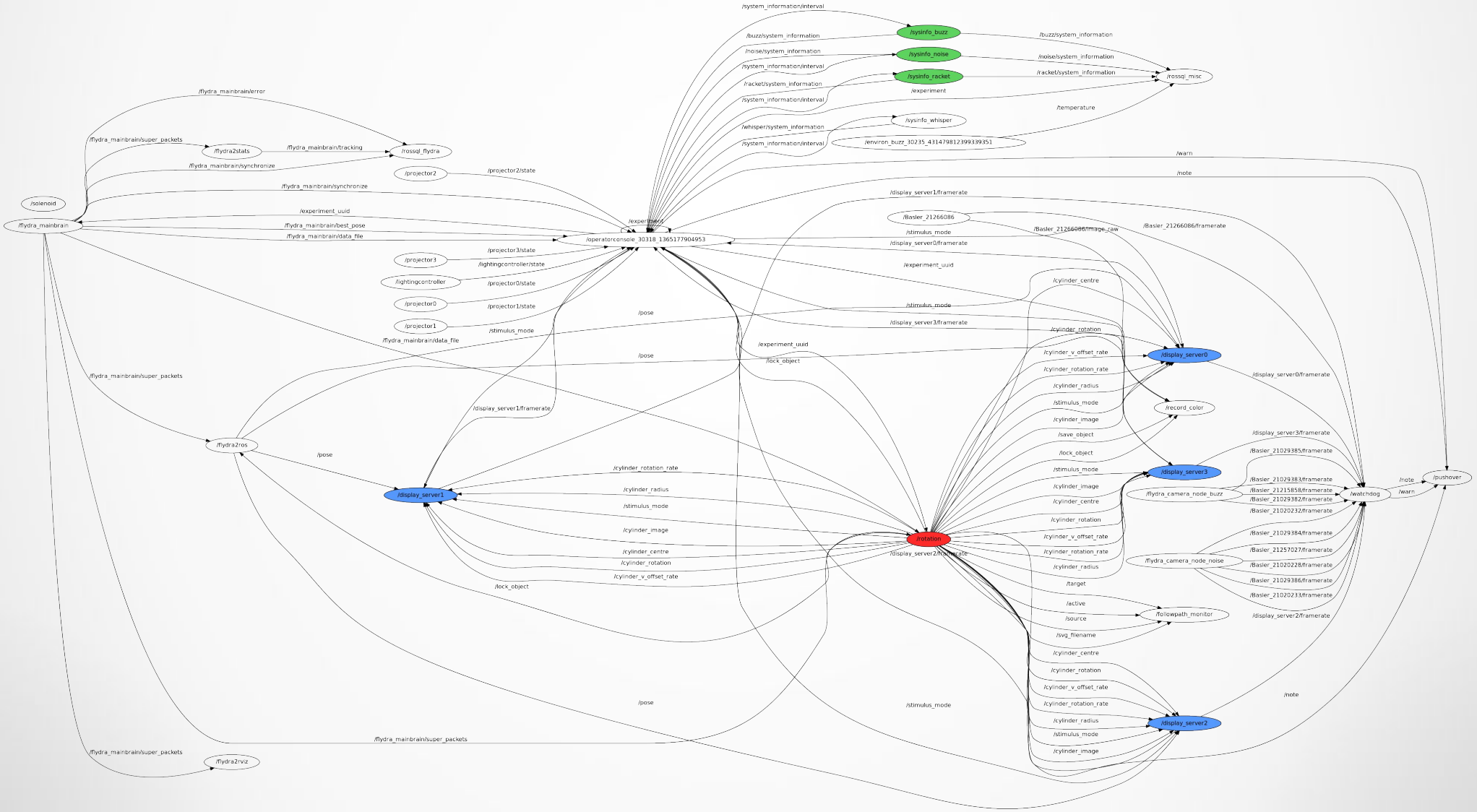
Motivation

- Real-time closed-loop virtual reality control of Drosophila
- Experiment distributed over 4 computers
- A variety of data collected



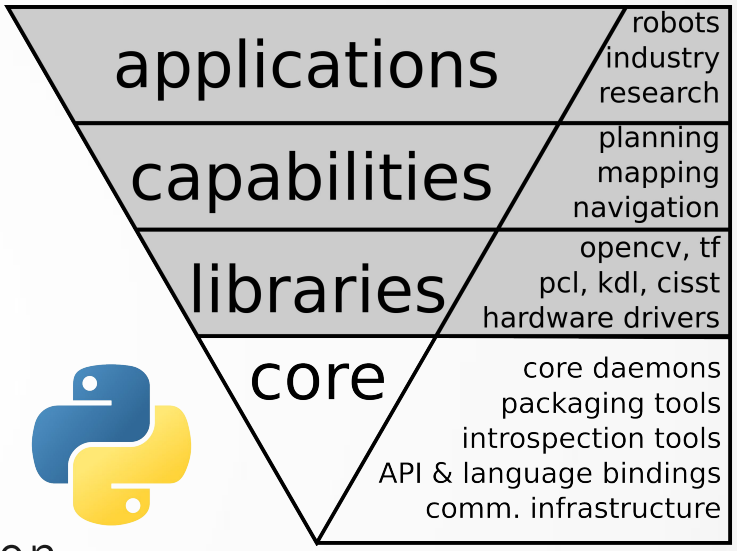


Motivation


/system_information/interval
/buzz/system_information
/noise/system_information
/system_information/interval
/racket/system_information
/system_information/interval
/whisper/system_information
/system_information/interval
/sysinfo_buzz
/sysinfo_noise
/sysinfo_racket
/sysinfo_whisper
/environ_buzz_30235_431479812399339351
/buzz/system_information
/noise/system_information
/racket/system_information
/experiment
/temperature
/rossql_misc
/flydra_mainbrain/error
/flydra_mainbrain/super_packets
/flydra2stats
/flydra_mainbrain/tracking
/rossql_flydra
/flydra_mainbrain/synchronize
/projector2
/projector2/state
/flydra_mainbrain/synchronize
/solenoid
/flydra_mainbrain
/experiment_uuid
/flydra_mainbrain/best_pose
/flydra_mainbrain/data_file
/experiment
/operatorconsole_30318_1365177904953
/warn
/note
/display_server1/framerate
/Basier_21266086
/stimulus_mode
/display_server0/framerate
/experiment_uuid
/projector3
/projector3/state
/lightingcontroller
/lightingcontroller/state
/projector0
/projector0/state
/projector1
/projector1/state
/stimulus_mode
/flydra_mainbrain/data_file
/pose
/display_server3/framerate
/cylinder_centre
/cylinder_rotation
/cylinder_v_offset_rate
/cylinder_rotation_rate
/cylinder_radius
/stimulus_mode
/cylinder_image
/save_object
/lock_object
/experiment_uuid
/lock_object
/pose
/flydra_mainbrain/super_packets
/flydra2ros
/pose
/display_server1/framerate
/cylinder_rotation_rate
/cylinder_radius
/stimulus_mode
/cylinder_image
/cylinder_centre
/cylinder_rotation
/cylinder_v_offset_rate
/lock_object
/display_server1
/rotation
/display_server2/framerate
/display_server0
/record_color
/display_server3
/flydra_camera_node_buzz
/flydra_camera_node_noise
/Basier_21029385/framerate
/Basier_21215858/framerate
/Basier_21029382/framerate
/Basier_21020232/framerate
/Basier_21029384/framerate
/Basier_21257027/framerate
/Basier_21020228/framerate
/Basier_21029386/framerate
/Basier_21020233/framerate
/display_server2/framerate
/watchdog
/note
/warn
/pushover
/target
/active
/source
/svg_filename
/followpath_monitor
/cylinder_centre
/cylinder_rotation
/cylinder_v_offset_rate
/cylinder_rotation_rate
/cylinder_radius
/stimulus_mode
/cylinder_image
/display_server2
/note
/pose
/stimulus_mode
/flydra_mainbrain/super_packets
/flydra_mainbrain/super_packets
/flydra2rviz

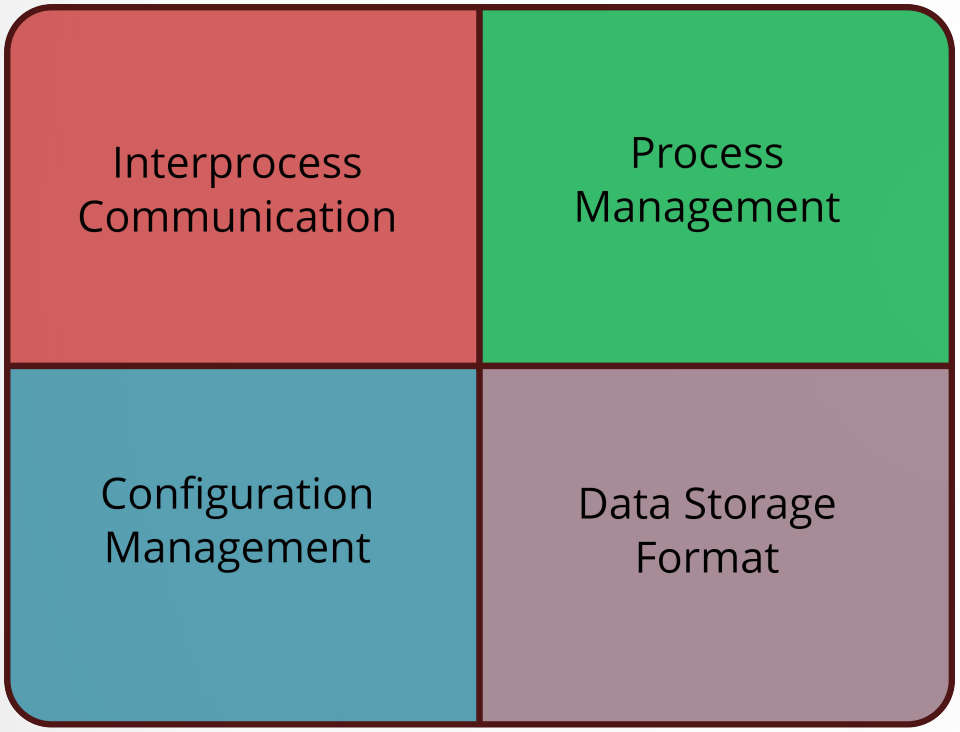
What Is Robot Operating System (ROS)

- A meta-operating system
- A collection of robotics related libraries
- A (bad) package management system
- An architecture for distributed inter-process inter-machine communication and configuration
- Development tools for system runtime and data analysis



The ROS core has nothing to do with robotics and has everything needed for reproducible Science

The Major Concepts of ROS



- ROS Python API
  - Exposes the ROS core
  - Used to interface with the ROS network
- All strongly typed

ROS IPC

- Nodes
  - Correspond to processes (ros mainloop)
- Topics
  - Asynchronous “stream-like” communication
  - Can have one or more publishers, and one or more subscribers
- Services
  - Synchronous, “function-call-like”
  - Can have only one server, and multiple clients

Event.msg

```
Header header
int32 reward
float32 speed

int32 REWARD_RED=1
int32 REWARD_GREEN=2
```

Command.srv

```
string command
---
string response
```

ROS Process and Configuration Management

- Launch files
  - XML files for launching multiple nodes across multiple machines
  - associate a set of parameters and nodes with a single file
  - hierarchically compose collections of other launch files
  - automatically re-spawn nodes if they crash
- Parameter Server
  - Provides parameter values to nodes
- Command line tools
  - for interacting with nodes
  - great for debugging
  - rosbag for recording messages on the network
- Gui tools
  - simple plotting
  - visualization of the ros graph
  - many vision/camera related tools

The ROS API(s)

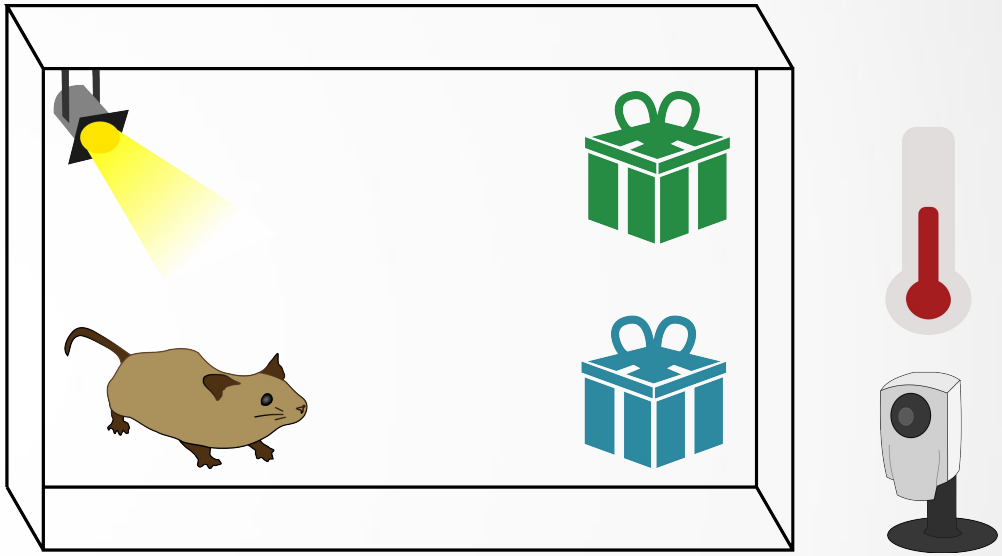
- The IPC is one API
  - Strongly typed, described by paths and msg / srv files
- Use the rospy module to
  - Communicate using this IPC
  - Get/Set parameters
  - Save data to bag files
  - Also provides timing primitives

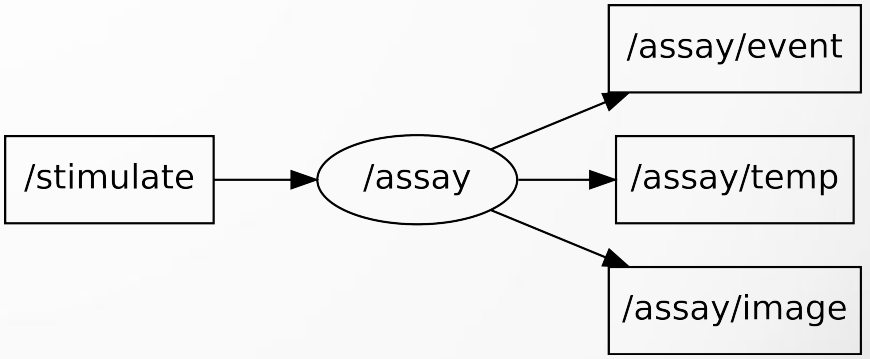
```
import roslib; roslib.load_manifest('scipy')
import rospy
from std_msgs.msg import Float32

rospy.init_node("answer")
pub = rospy.Publisher("info", Float32,
                      latch=True)
pub.publish(42)
rospy.spin()
```

```
import roslib; roslib.load_manifest('scipy')
import rospy
from std_msgs.msg import Float32

def on_msg(msg):
    print "the meaning of life is", msg.data

sub = rospy.Subscriber("info", Float32,
                       on_msg)

rospy.init_node("question")
rospy.spin()
```

A Real Life Example

- A conditioning assay
  - In response to stimulus (light) the mouse makes a choice
  - Record the choice
  - Record experiment data
    - Video
    - Temperature
- Use ROS to implement and parallelize





Live Coding
https://github.com/nzjrs/scipy2013-presentation

Best Practices for ROS Applications

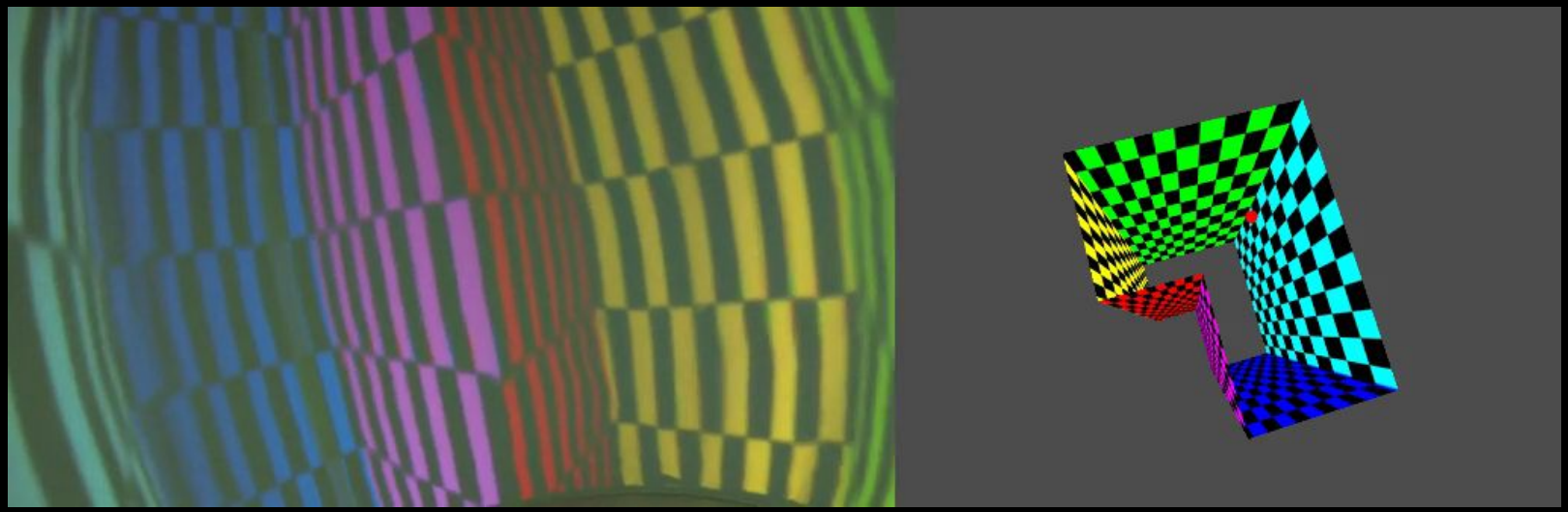
- Prefer command line arguments to ROS remapping
- Use standard message types where possible
- Don't feel compelled to use ROS for everything
- Take advantage of subscriber callbacks and Latched messages
- Synchronizing Time
  - To correlation data collected on multiple machines
  - ROS does have a special notion of time, /clock
- But managing your own timebase is often better
  - PTPd maintains us synchronization between computer clocks
- Pandas indexing / interpolation = win

Interacting With External Partners

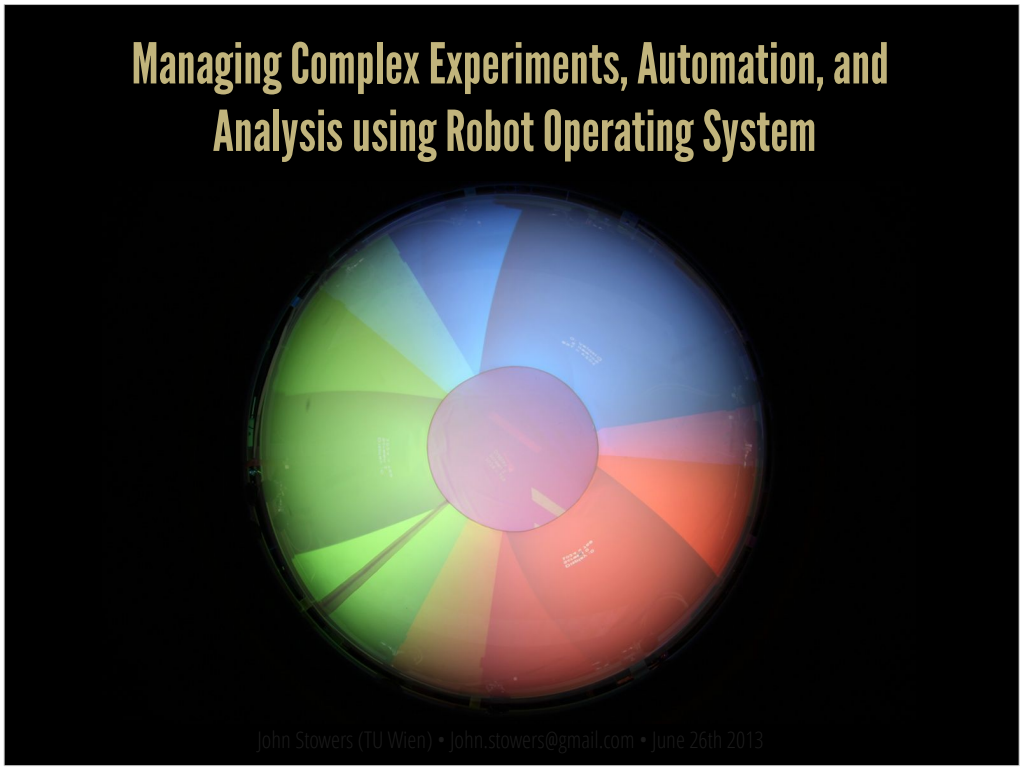
- Default data storage format is .bag files
  - Scientists should user inter-operable file formats
    - bag2hdf5 (export bag files to clean hdf5)
- ros_sql (persist bag messages an a sql database)
- ros_freeze
  - Repackage ROS nodes as standard python packages (including dependencies)
- http://www.github.com/strawlab/

Conclusion

- ROS is a healthy vibrant project with a strong future
- It's performance is sufficient for demanding real-time tasks
- Its graph/IPC model maps well to managing experiments
- The ROS core and API is pure python and easy to get to know

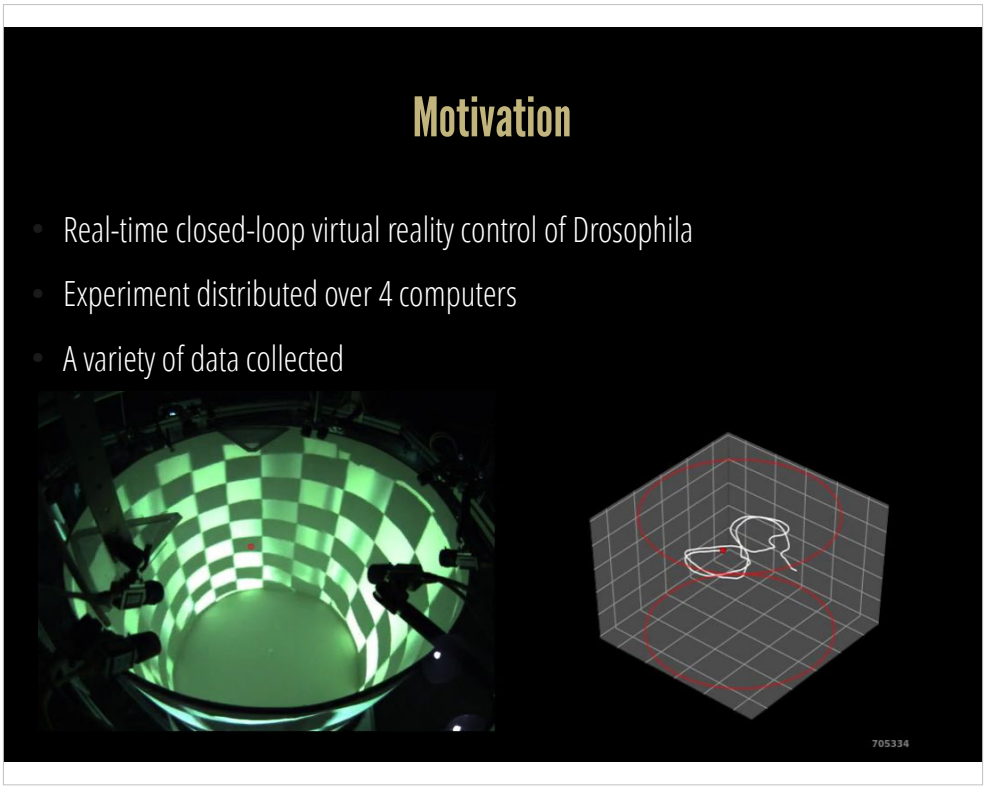


We are hiring • http://www.strawlab.org



WRITE GITHUB ADDRESS ON BOARD

Hi Everyone, I'm John Stowers, a postdoc at TU Vienna, and I'm going to talk to you about how we use ROS to mange and automate our experiments, and how you should too.

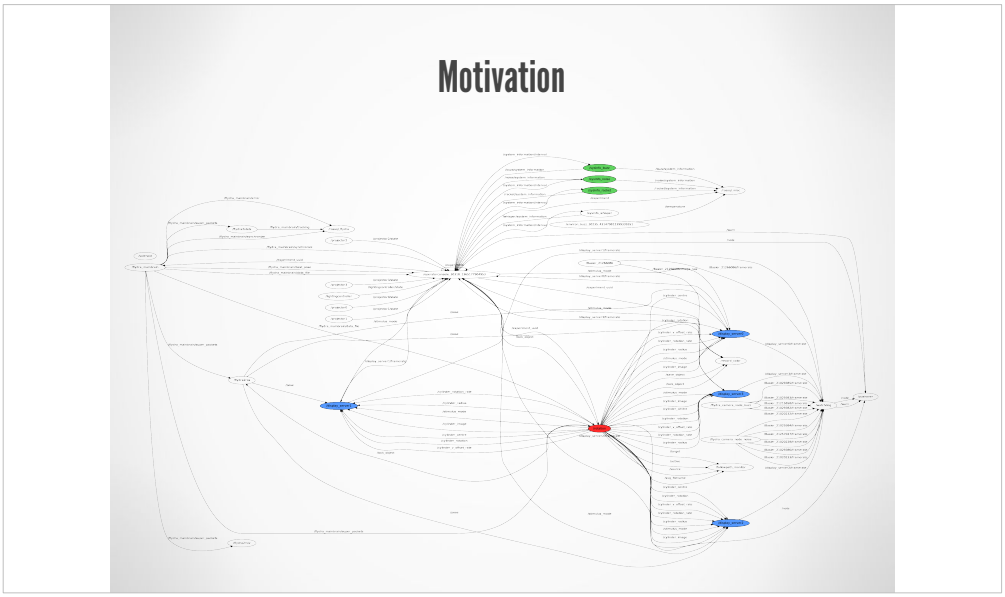


In the straw lab we perform virtual reality experiments on flying Drosophila (fruit fly) for the purposes of understanding the how the visual system works

What you see here is...

Real time close-loop behavioural experiments require signifigant computer power

Distributing processing over multiple systems is often necessary



This is a graph showing a visual depiction of that same experiment.

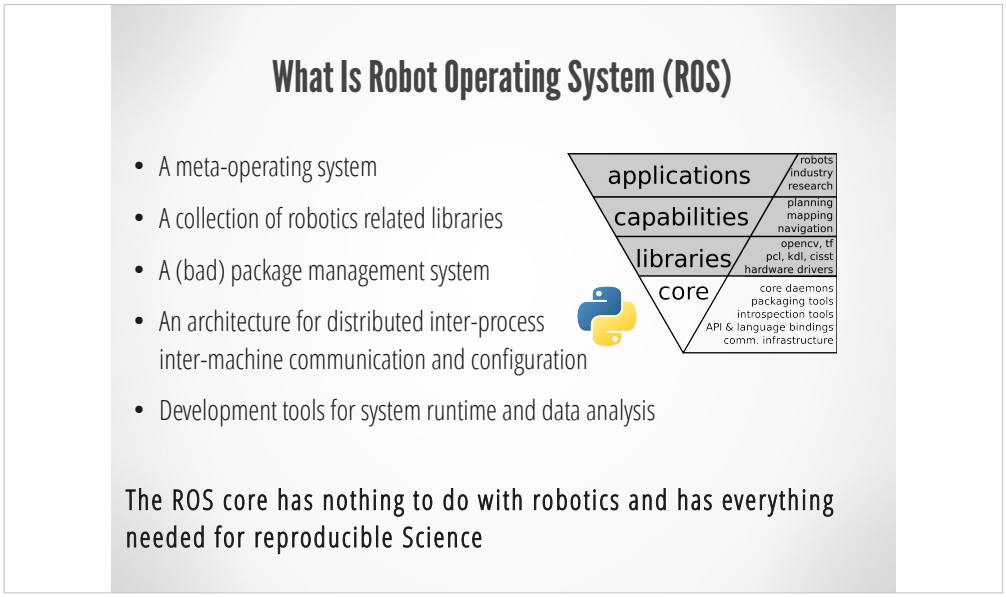
ROS is a graph, where nodes are processes, and edges are topics or services.

Don't be too concerned with the details here, red=experiment

The motivation is,

How to start and manage such and experiment reliably.

How to shield complexity from less experienced members of the lab



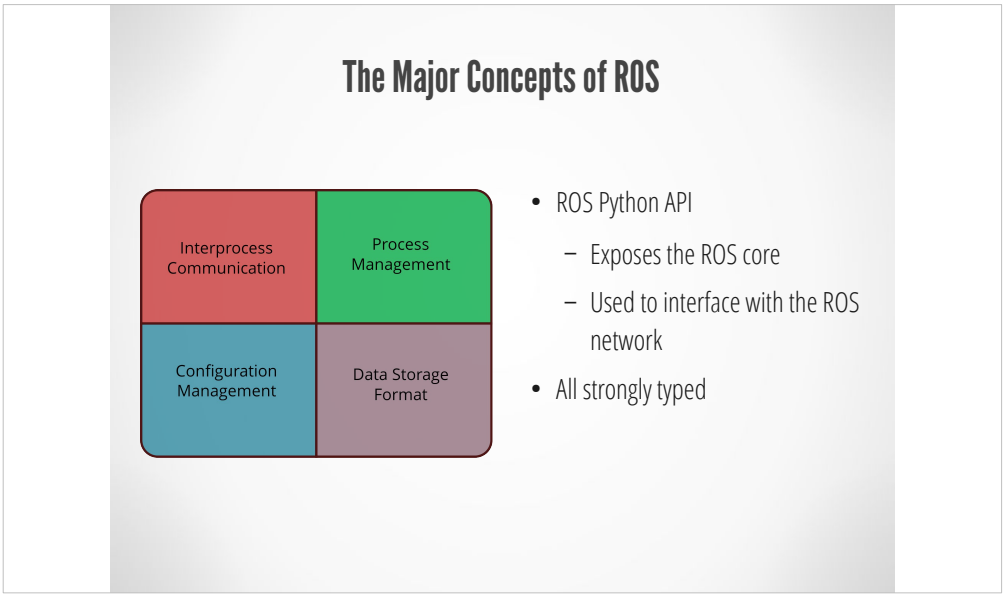
Some of you might have heard of ROS, especially if you work in engineering or robotics, where it is more well known.

Ros is a meta-os spanning, complex applications, algorithms, libraries, down to the ROS core and below.

And the reason I call it that is because they have reinvented large part of the OS stack

- package management
- dependency resolution
- build system

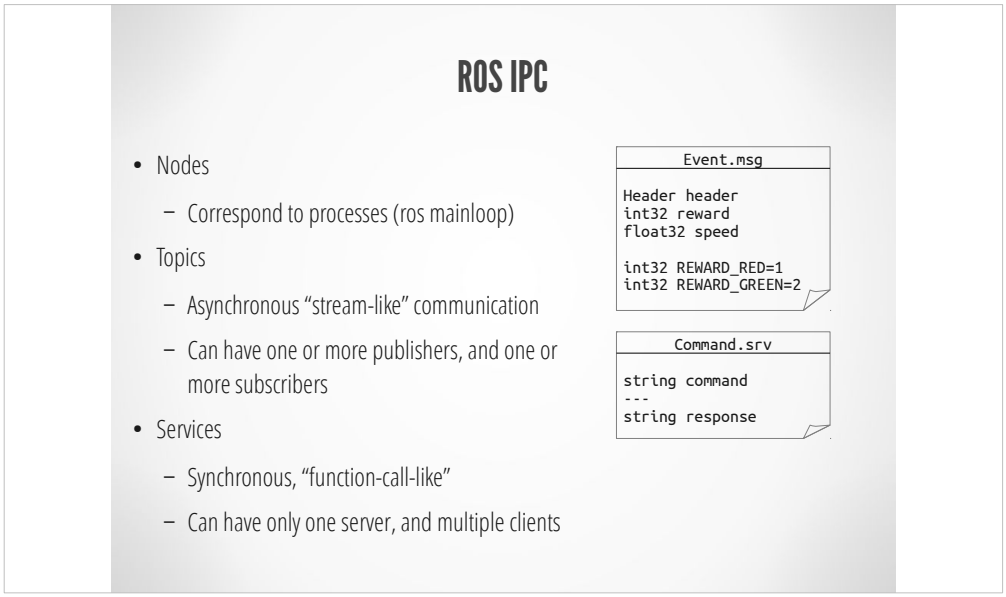
Frequent users of open source might balk at this, but I'd like to reassure you that the ROS is a very healthy ecosystem (commercially supported). The NIH was necessary for ROS goals, and my point of this talk is to exlain the ROS core has nothing specifically to do with robotics, and is perfect to use for reproducable science.



Functionally ROS provides 4 services you should use to create reproducible science.

SAY THEM

I will talk about a the first 3 of these, but the rest are more clearly demonstrated in the live demo later.



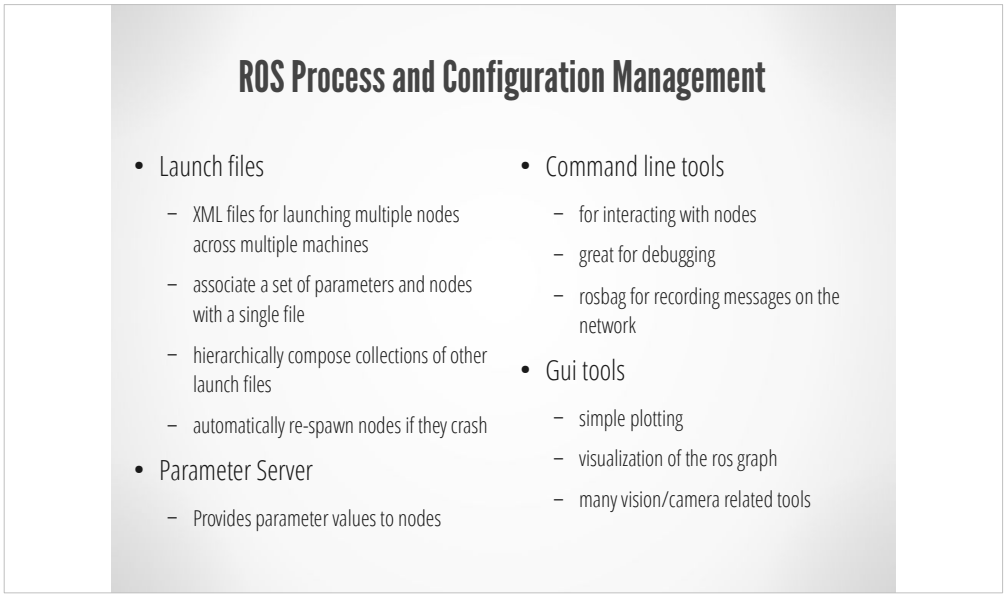
ROS allows you to distribute a soft-realtime system across a strongly typed message bus.

If you recall the graph from the first slide, the edges of that graph repreesent IPC between the nodes. This comes in two forms, services and topics.

TALK

Both of these are described using an IDL seen here.

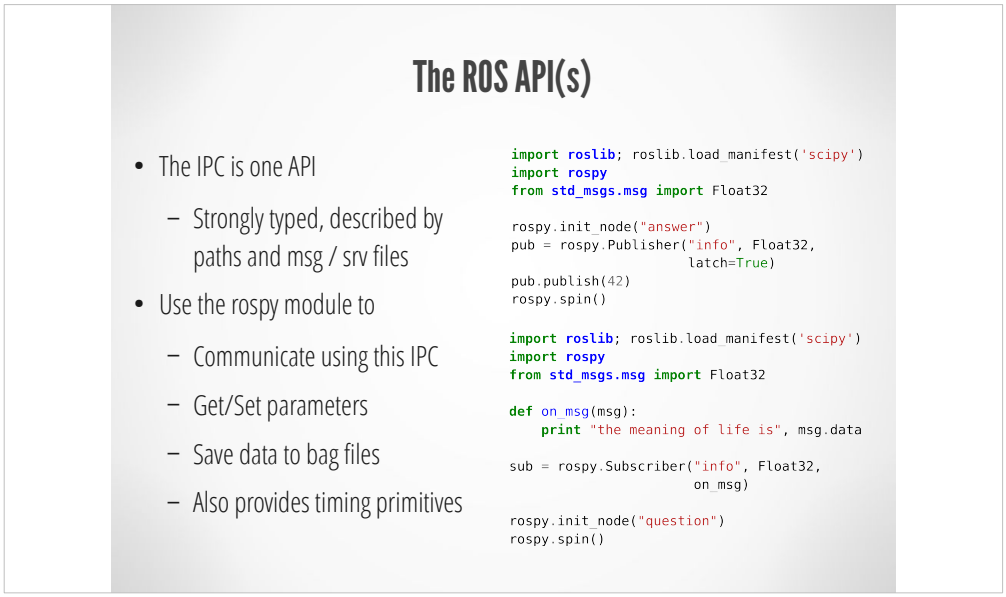
Note the standard types and the special Header type, reminicent of C struct definitions.



The second and third key pillars of ROS are process and configuration management.

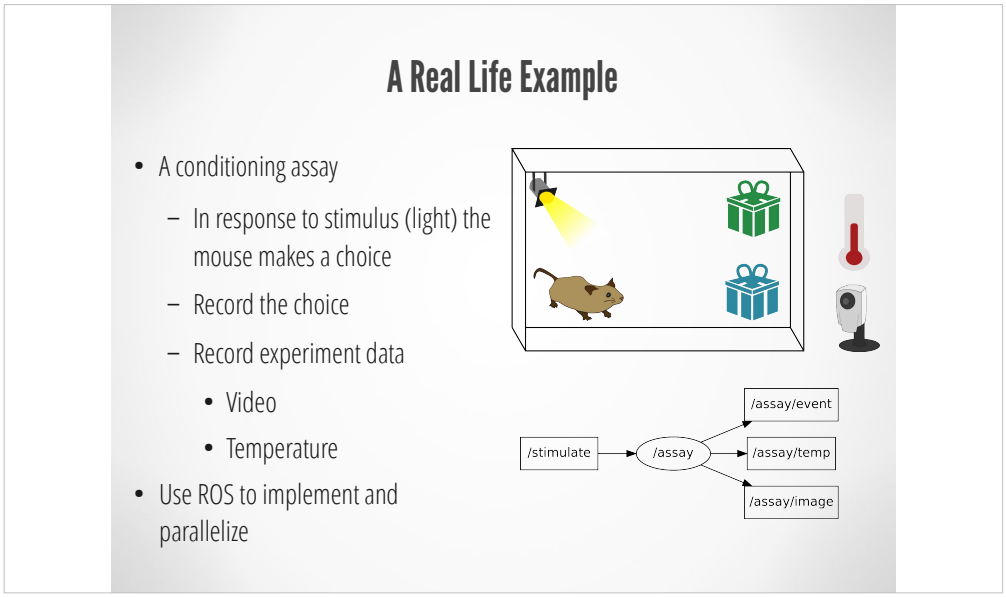
Process management encompasses launching ros nodes, and interacting with nodes on the ROS network using the provided command line tools.

Configuration management is managed by the parameter server. A service hosted by the master ros process by which nodes and get and set heirarchical parameters



Now lets switch gears here, and introduce some concepts that I will use to code up a medium complexity example shortly.

Talk about the two APIs



Because I'm sure many of you could breeze through the ros tutorial in short time, I will instead focus on a medium complexity hypothetical experiment.

The experiment varies the duty cycle of the stimulus and measures which choice the animal makes

Think of this assay as a node

(split screen, kat assay)
Roscd
Rosrun
Rosnode info
Rostopic echo /assay/event
(comment on the image tools)
Rosrun image_view image_view
Rosrun rxplot rxplot /assay/temp

(now lets code up an experiment to control this assay)
(switch to other screen, kat the other on, switch back)

Start the experiment node
Rosgraph

(now lets kill all those and start to parallelize)

Roslaunch
Rosrun rxgraph rxgraph
rosparam

(kill)

Rosrun rxbag rxbag bla
(explore)

Best Practices for ROS Applications

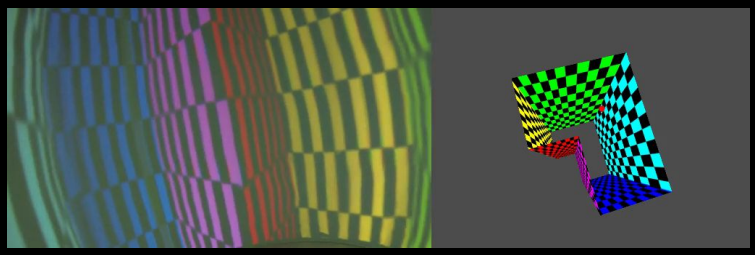
- Prefer command line arguments to ROS remapping
- Use standard message types where possible
- Don't feel compelled to use ROS for everything
- Take advantage of subscriber callbacks and Latched messages
- Synchronizing Time
  - To correlation data collected on multiple machines
  - ROS does have a special notion of time, /clock
- But managing your own timebase is often better
  - PTPd maintains us synchronization between computer clocks
- Pandas indexing / interpolation = win

Interacting With External Partners

- Default data storage format is .bag files
  - Scientists should user inter-operable file formats
    - bag2hdf5 (export bag files to clean hdf5)
- ros_sql (persist bag messages an a sql database)
- ros_freeze
  - Repackage ROS nodes as standard python packages (including dependencies)
- http://www.github.com/strawlab/

Conclusion

- ROS is a healthy vibrant project with a strong future
- It's performance is sufficient for demanding real-time tasks
- Its graph/IPC model maps well to managing experiments
- The ROS core and API is pure python and easy to get to know



We are hiring • http://www.strawlab.org