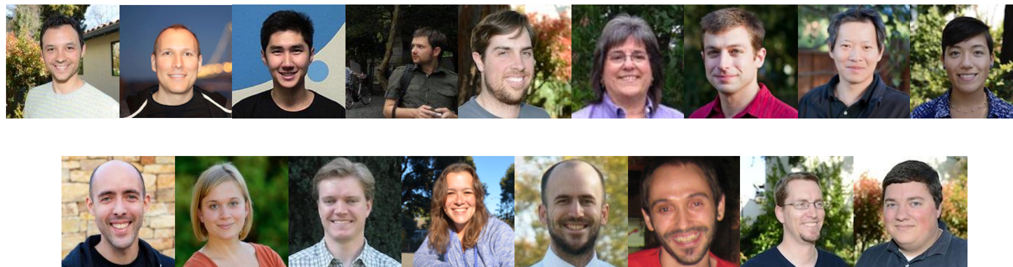
C++ in Open Source Robotics

Jackie Kay
Louise Poubel

OSRF



Jackie

Louise

The mission of OSRF is to support the development, distribution, and adoption of open source software for use in robotics research, education, and product development.

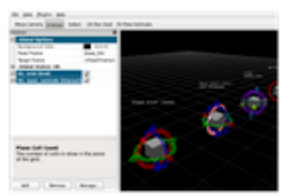
What is Robot Operating System?

What is Robot Operating System?

=

Plumbing

+



Tools

+

Capabilities

+

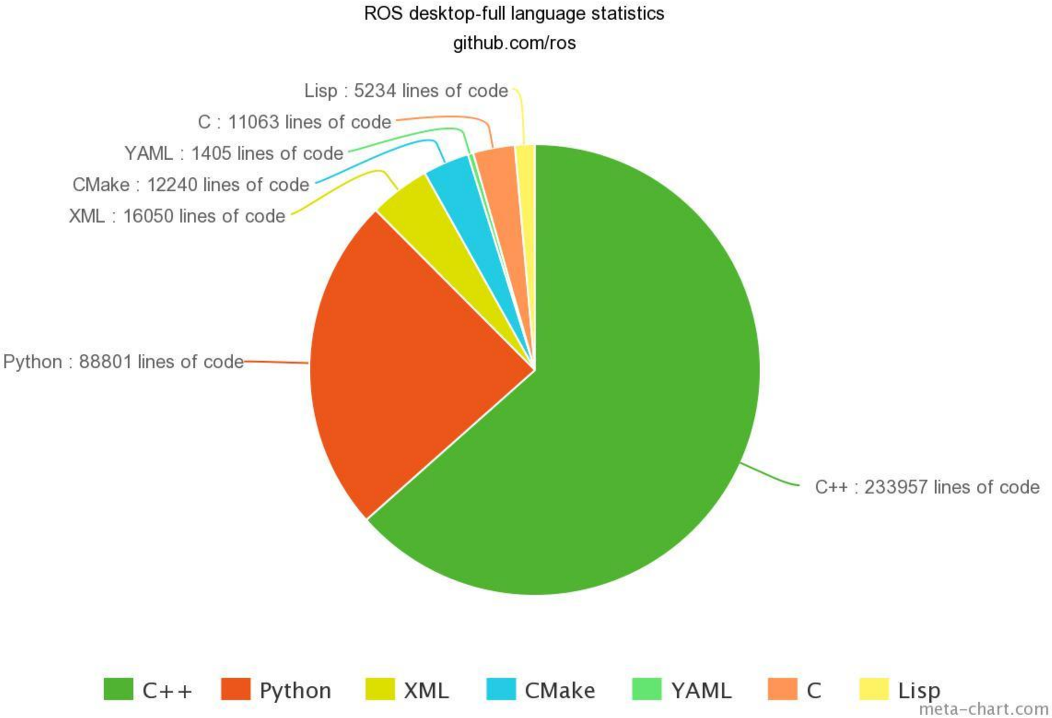
Ecosystem

ROS Statistics

Metrics as of July 2015 (http://wiki.ros.org/Metrics):

- Unique IPs downloading ROS debs: ~45,000/month
- Academic papers citing original paper: 1843
- Robot models officially supported: >101
- wiki.ros.org pageviews: ~37,000/day

Longest distance a ROS robot has traveled from Earth: 270 miles



ROS Communication

ROS Communication

```
geometry_msgs::Twist::Ptr msg =
  boost::make_shared<...>();
...
pub->publish(msg);
```

ROS Communication

Publisher Node

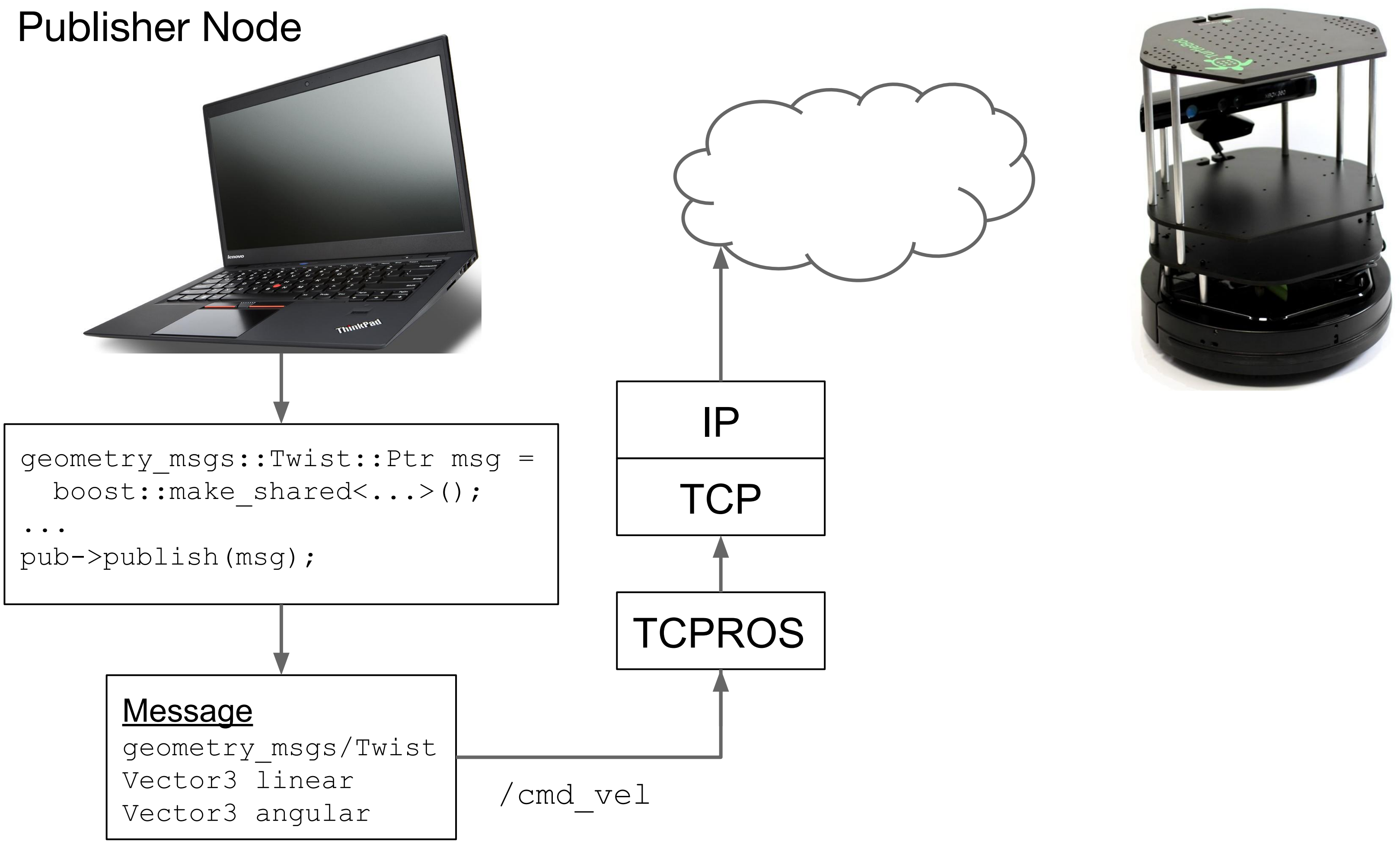
```
geometry_msgs::Twist::Ptr msg =
  boost::make_shared<...>();
...
pub->publish(msg);
```

<u>Message</u>

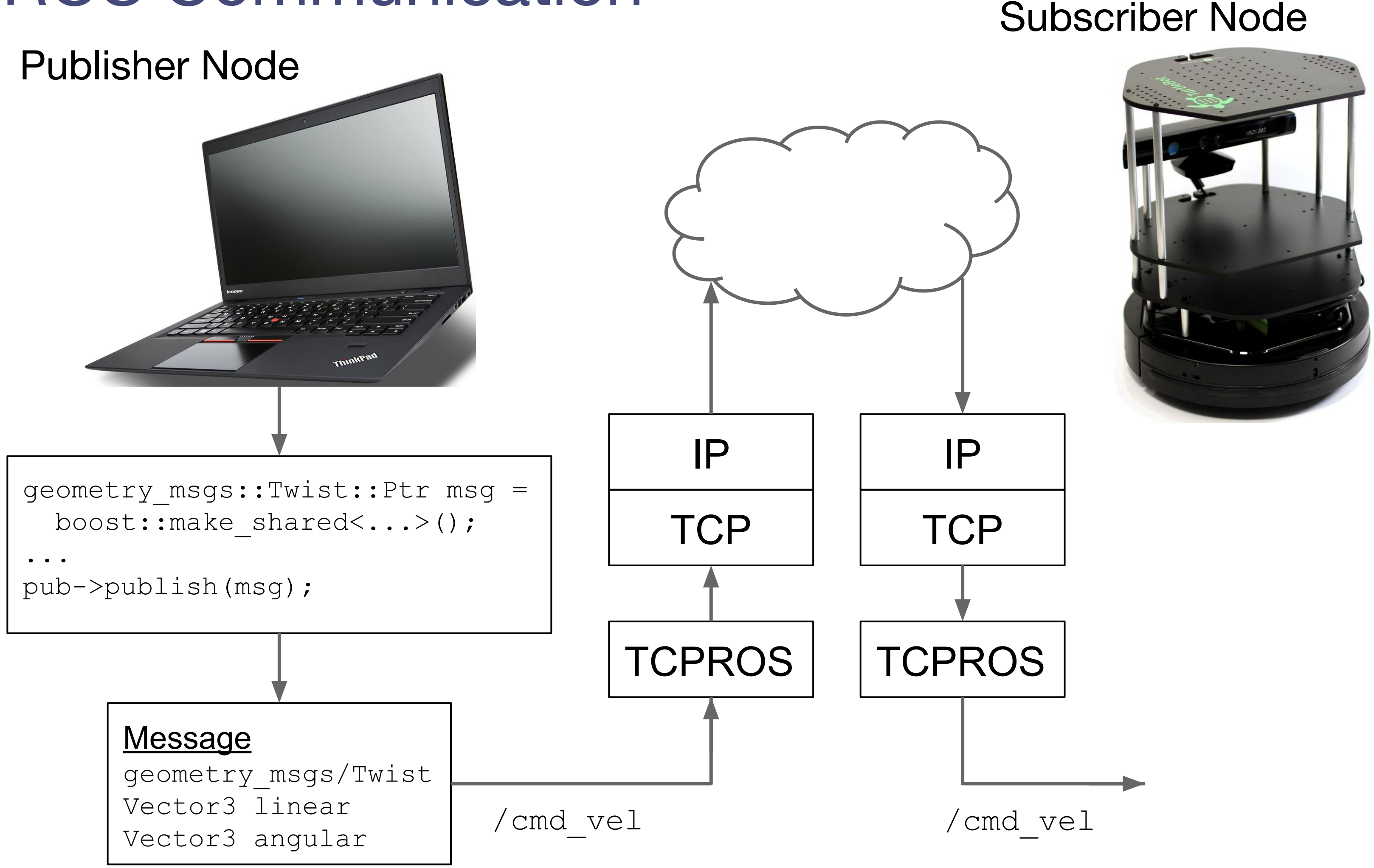
```
geometry_msgs/Twist
Vector3 linear
Vector3 angular
```

/cmd_vel

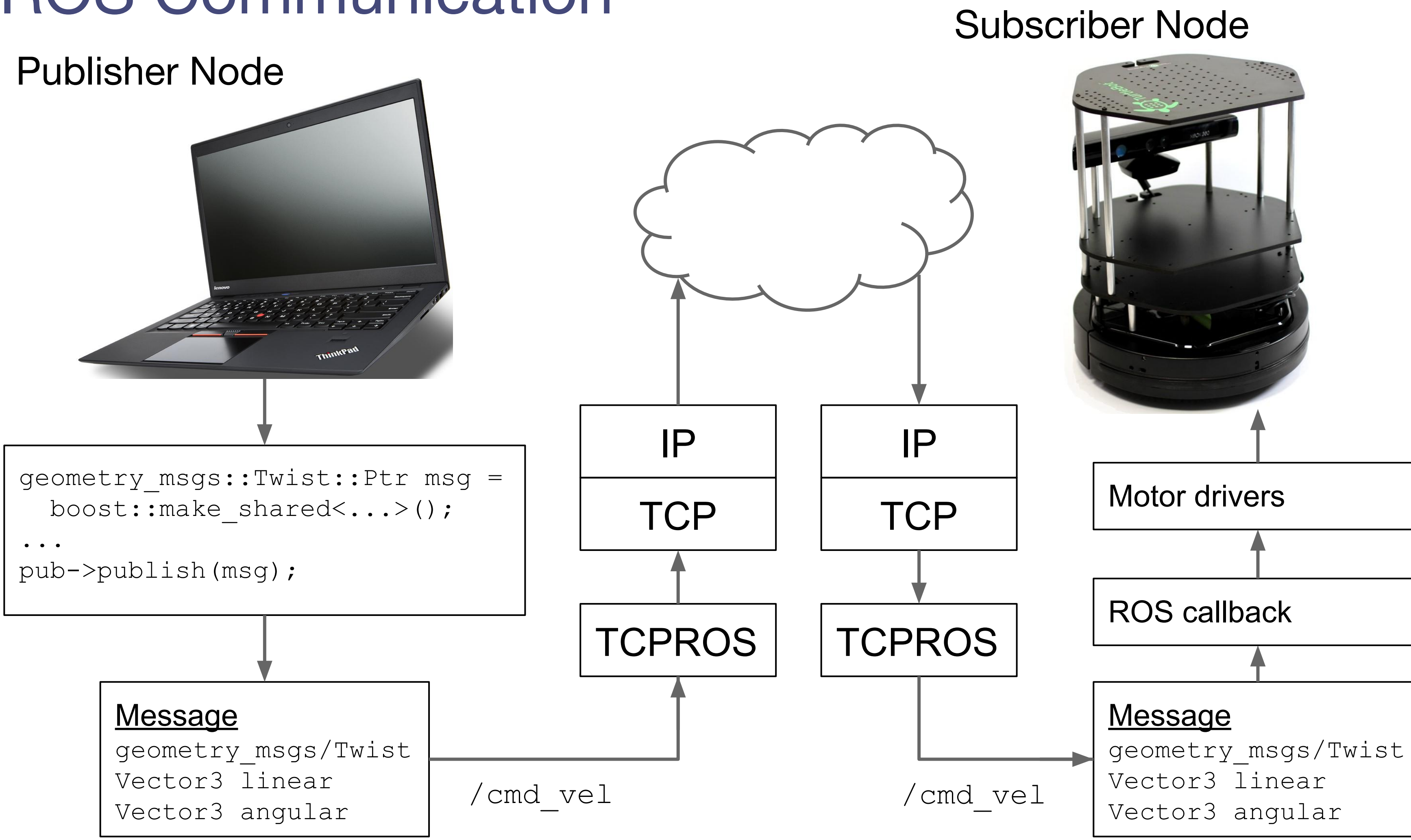
ROS Communication



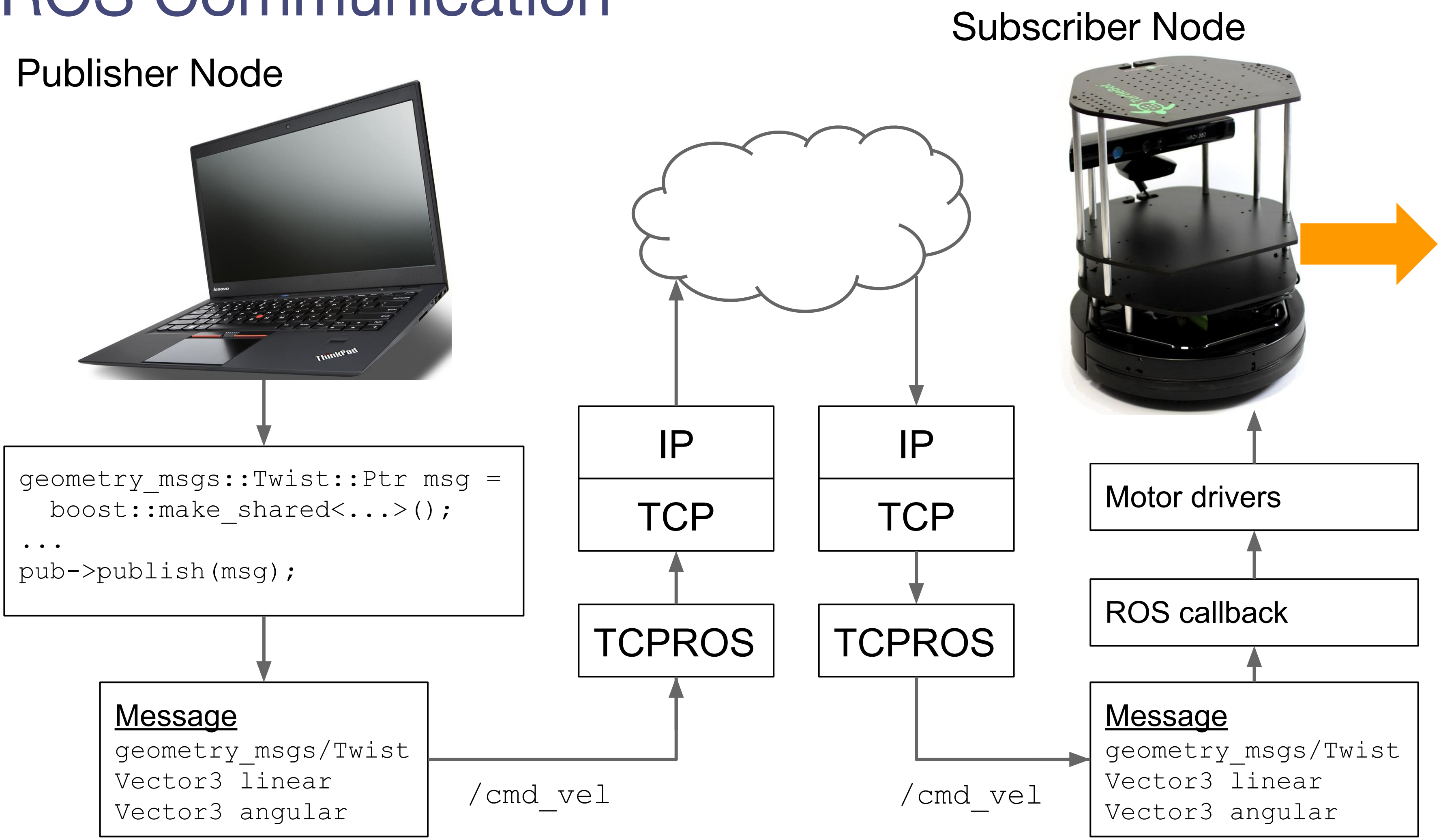
ROS Communication



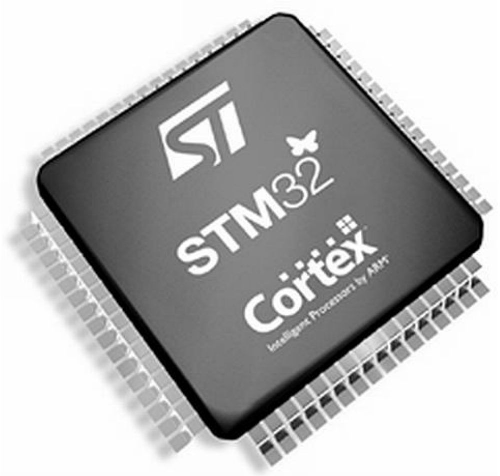
ROS Communication



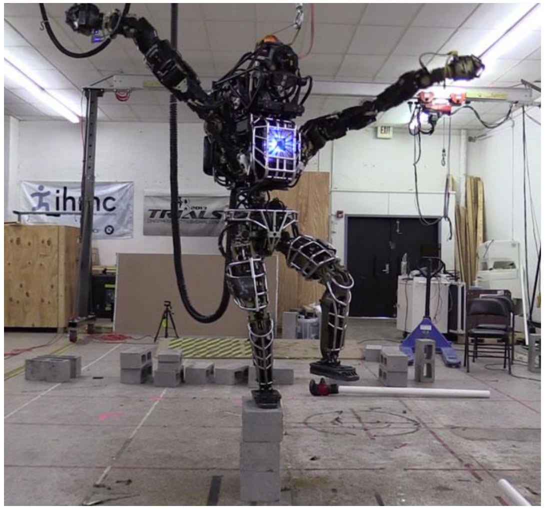
ROS Communication



ROS 2: Motivation



Bare-metal embedded



Real-time performance

High-traffic robot fleets

Open Source Robotics Foundation

Not pictured: major API makeover after 8 years development!

How is ROS 2 different?

DDS (Data Distribution Service) middleware
Cross-platform support
C++11
Python 3 (irrelevant for this conference)

ROS 1 vs. ROS 2 example

```cpp
void chatterCallback(const std_msgs::
String::ConstPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  ros::init(argc, argv, “listener”);
  ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe(
      "chatter", 7, chatterCallback);

  ros::spin();

  return 0;
}
```

```cpp
void chatterCallback(const
  std_msgs::String::ConstSharedPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  rclcpp::init(argc, argv);
  auto node =
    rclcpp::Node::make_shared("listener");

  auto sub =
    node->create_subscription
    <std_msgs::String>("chatter", 7,
    chatterCallback);

  rclcpp::SingleThreadedExecutor executor;
  executor.add_node(node);
  executor.spin();

  return 0;
}
```

ROS 1 vs. ROS 2 example

```cpp
void chatterCallback(const
std_msgs::String::ConstPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe(
      "chatter", 7, chatterCallback);

  ros::spin();

  return 0;
}
```

```cpp
void chatterCallback(const
  std_msgs::String::ConstSharedPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  rclcpp::init(argc, argv);
  auto node =
    rclcpp::Node::make_shared("listener");

  auto sub =
    node->create_subscription
    <std_msgs::String>("chatter", 7,
    chatterCallback);

  rclcpp::SingleThreadedExecutor executor;
  executor.add_node(node);
  executor.spin();

  return 0;
}
```

ROS 1 vs. ROS 2 example

```
void chatterCallback(const std_msgs::
String::ConstPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe(
      "chatter", 7, chatterCallback);

  ros::spin();

  return 0;
}
```

```
void chatterCallback(const
  std_msgs::String::ConstSharedPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  rclcpp::init(argc, argv);
  auto node =
    rclcpp::Node::make_shared("listener");

  auto sub =
    node->create_subscription
    <std_msgs::String>("chatter", 7,
    chatterCallback);

  rclcpp::SingleThreadedExecutor executor;
  executor.add_node(node);
  executor.spin();

  return 0;
}
```

ROS 1 vs. ROS 2 example

```
void chatterCallback(const std_msgs::
String::ConstPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  ros::init(argc, argv, “listener”);
  ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe(
      "chatter", 7, chatterCallback);

  ros::spin();

  return 0;
}
```

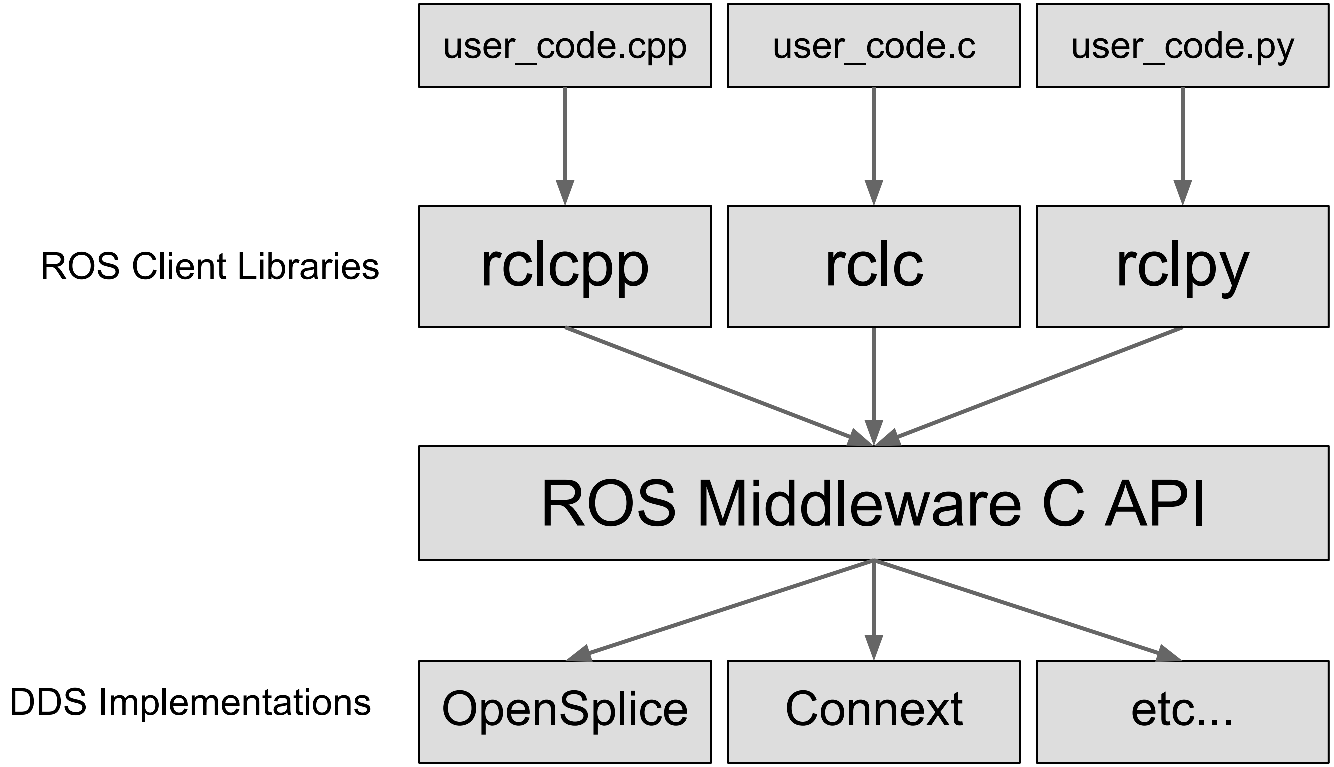
```
void chatterCallback(const
  std_msgs::String::ConstSharedPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  rclcpp::init(argc, argv);
  auto node =
    rclcpp::Node::make_shared("listener");

  auto sub =
    node->create_subscription
    <std_msgs::String>("chatter", 7,
    chatterCallback);

  rclcpp::SingleThreadedExecutor executor;
  executor.add_node(node);
  executor.spin();

  return 0;
}
```

ROS 1 vs. ROS 2 example

```
void chatterCallback(const std_msgs::
String::ConstPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  ros::init(argc, argv, “listener”);
  ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe(
      "chatter", 7, chatterCallback);

  ros::spin();

  return 0;
}
```

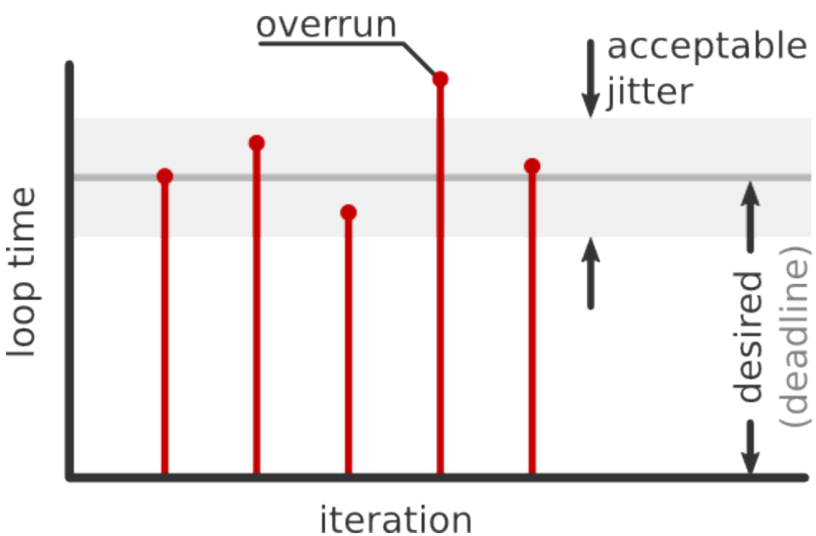
```
void chatterCallback(const
  std_msgs::String::ConstSharedPtr msg)
{
  printf("I heard: [%s]", msg->data);
}

int main(int argc, char *argv[])
{
  rclcpp::init(argc, argv);
  auto node =
    rclcpp::Node::make_shared("listener");

  auto sub =
    node->create_subscription
    <std_msgs::String>("chatter", 7,
    chatterCallback);

  rclcpp::SingleThreadedExecutor executor;
  executor.add_node(node);
  executor.spin();

  return 0;
}
```

ROS 2 Architecture

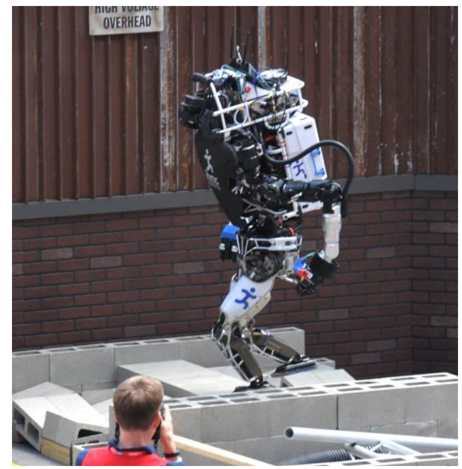


ROS 2 and Real-time Support

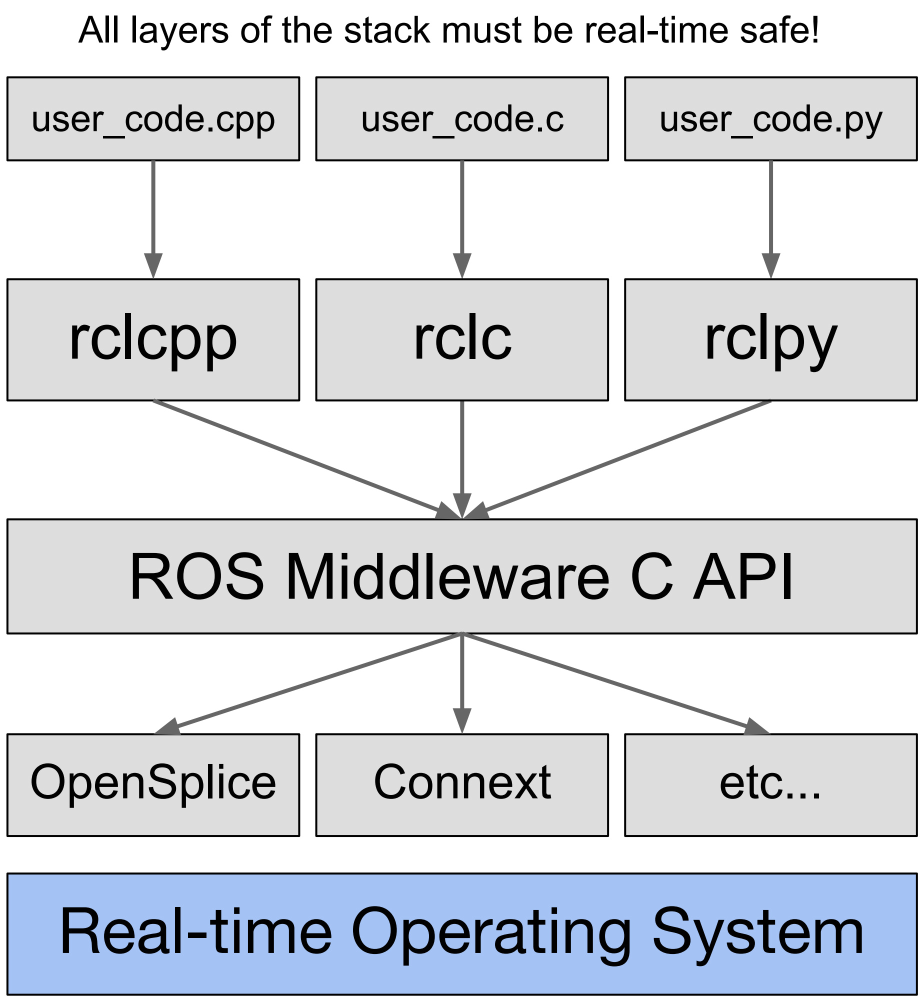
What is real-time computing?



Why do we care?



ROS 2 and Real-time Support



Real-time Requirements

`Executor`

Initialization: Preallocate memory

`spin()`

`rmw_wait(timeout)`

Pass wait conditions to waitset

Wait

Wake up if timed out

Do work if it came in

Cleanup: Deallocate memory

Loop until interrupted

`Executor::spin` is RT-critical

Initialization and cleanup phases can be non-RT safe

ROS 2 Real-time: Memory Management

Dependency injection for memory allocation strategy of Executor

Executor

```
void spin(...) {
  // New subscription arrives
  void * subscription_handle =
  memory_strategy-
>borrow_handle(
  sizeof(Subscription));

  ...
  // Don't need handle anymore
  memory_strategy-
>return_handle(
  subscription_handle);
}
```

Select at runtime

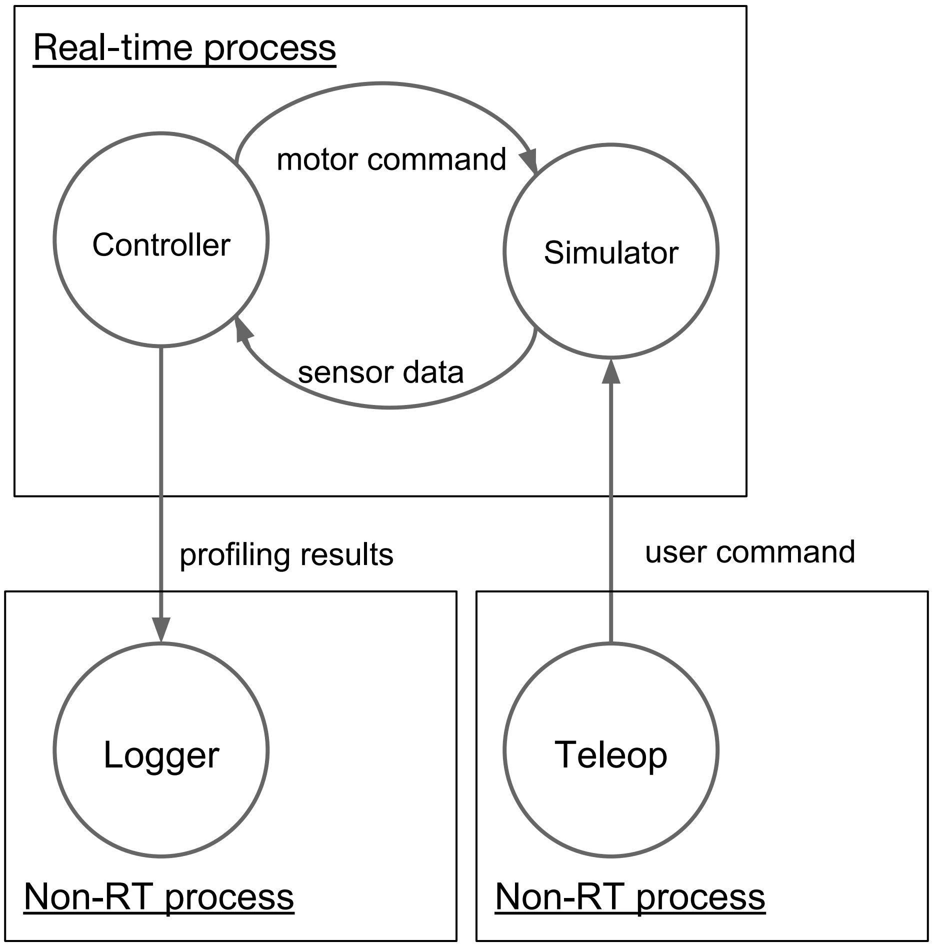
DynamicMemoryStrategy

```
void * borrow_handle
(size_t s) {
  return malloc(s);
}
void return_handle(void *
    ptr) {
  free(ptr);
}
```

HeapPoolMemoryStrategy

```
void * borrow_handle
(size_t s) {
  // reserve s bytes
  return mem_pool[seq];
}
void return_handle(
    void * ptr) {
  // get space reserved
by ptr and set to null
}
```

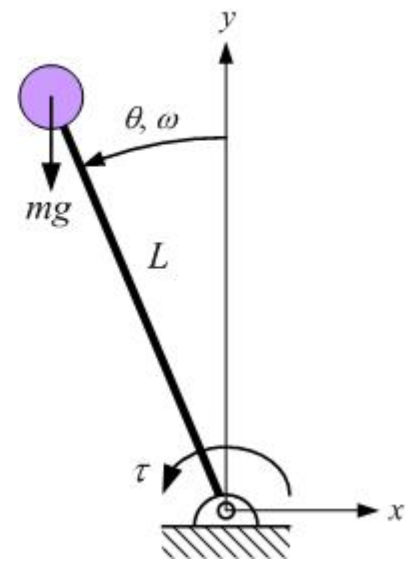
ROS 2 Real-time Benchmarking: Setup



RT_PREEMPT kernel (“soft real-time”)
POSIX threads with `SCHED_RR`

`malloc_hook`: count malloc calls
`getrusage`: count pagefaults

Goal:
1 kHz update loop (1 ms period)
Less than 3% jitter (30 microseconds)



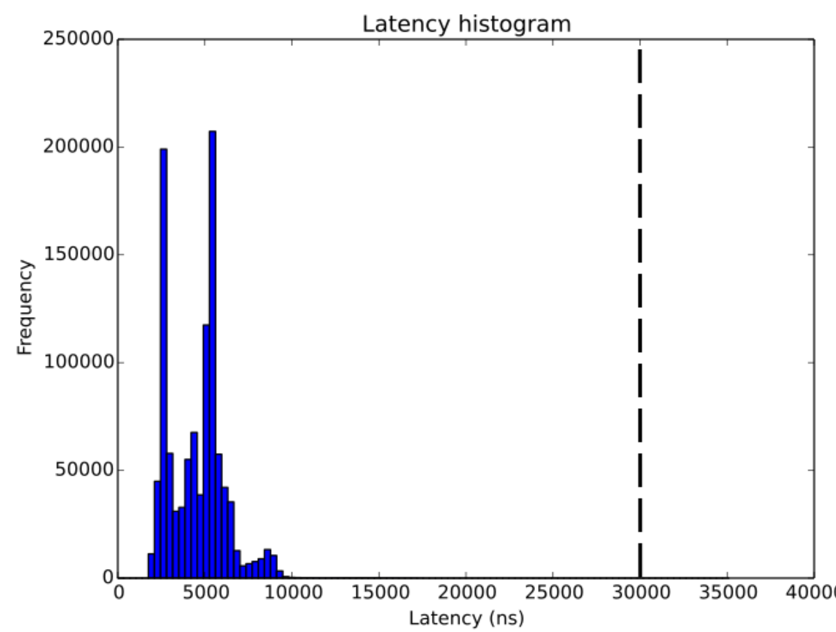
Open Source Robotics Foundation

ROS 2 Real-time Benchmarking: Results

Ran for 165 minutes

	Nanoseconds	% of update rate
Minimum	1620	0.162%
Maximum	35094	3.51%
Mean	4567.1	0.457%

Conclusion:
Off to a decent start



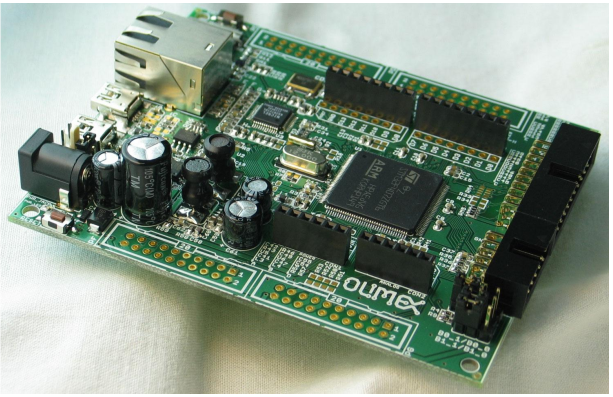
ROS 2 Real-time: Future work

Future work:

- Client/server communication
- Use a real-time safe allocator for common STL classes:
  - `std::vector`
  - `std::map`
  - `std::basic_string`
  - etc.
- Lock-free multithreaded executor

ROS 2 and Multi-platform Support

- Stable continuous integration for Ubuntu Linux, OSX, Windows 8
- Active development on STM32, SAMV71 microcontrollers



ROS 2 Windows Woes: std::function templates

Wanted to do this:

```
template<typename ServiceT>
create_service(
  const std::string & service_name,
  std::function<void(ServiceT::Request, ServiceT::
Response)> callback);

template<typename ServiceT>
create_service(
  const std::string & service_name,
  std::function<void(rmw_request_id_t, ServiceT::
Request, ServiceT::Response)> callback);
```

Calling the function compiles on OSX and Linux but not VS2015:
call is ambiguous.

```
node->create_service("foo", [](AddIntsService::
Request, AddIntsService::Response) { ... });
```

???

ROS 2 Windows Woes continued

Tried to do this instead:

```
template<typename ServiceT, FunctorT,  typename
std::enable_if<
  std::is_same<
    typename function_traits<FunctorT>:: template
      argument_type<2>,
    typename std::shared_ptr< typename ServiceT::
Response>
  >::value
>::type * = nullptr>
typename Service<ServiceT>::SharedPtr
  create_service(..., FunctorT callback){ ... }
```

Translation: Only overload the function template if the third argument to the callback is of type `ServiceT::Response`

???

ROS 2 Windows Woes: Expression SFINAE

Still didn't compile!

Expression SFINAE is not fully implemented in Visual Studios 2015
See here: http://blogs.msdn.com/b/vcblog/archive/2015/06/19/c-11-14-17-features-in-vs-2015-rtm.aspx

Used this in the body of the function instead to ensure that the function has the right signature:

```
static_assert(
  std::is_same<
   typename function_traits<FunctorT>:: template
argument_type<2>,
    typename std::shared_ptr<typename ServiceT::
Response>
  >::value, "Third argument must be of type
             std::shared_ptr<ServiceT::
Response>");
```

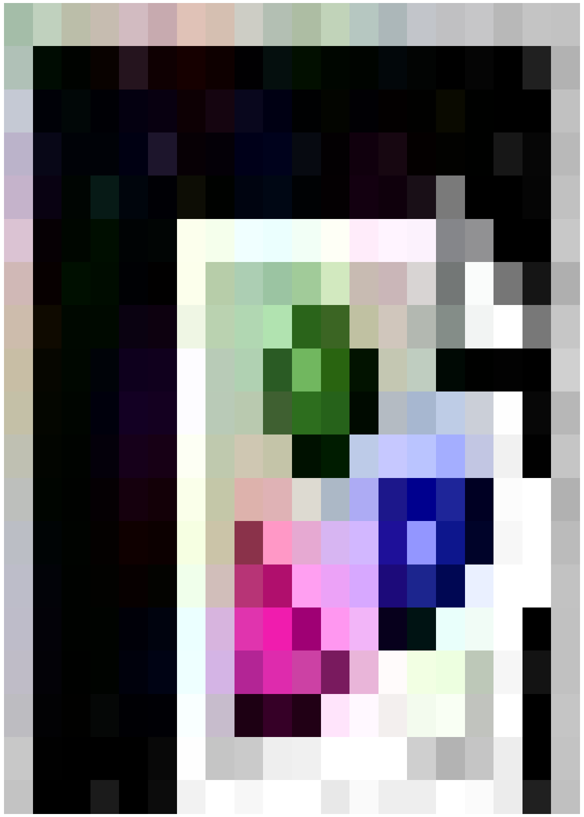
???

ROS/ROS 2: Wrap-up

Now you have everything you need to program robots, right?

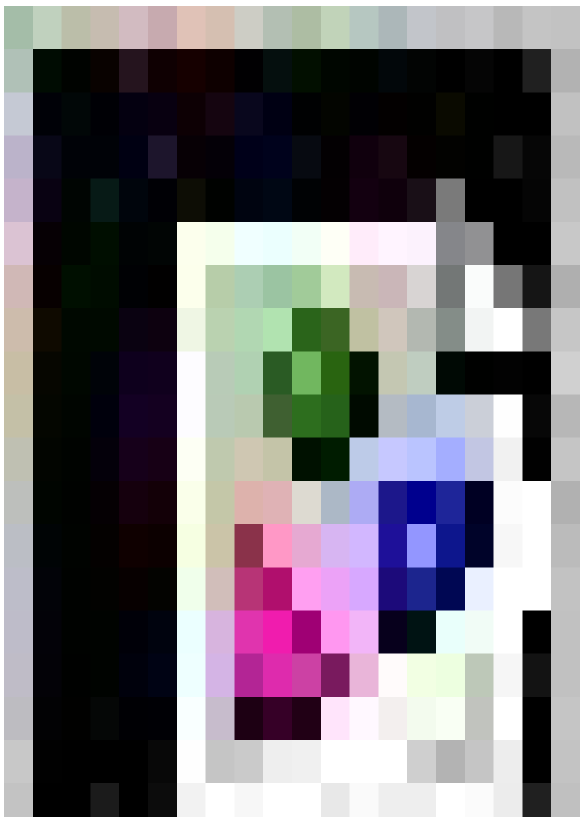
ROS/ROS 2: Wrap-up

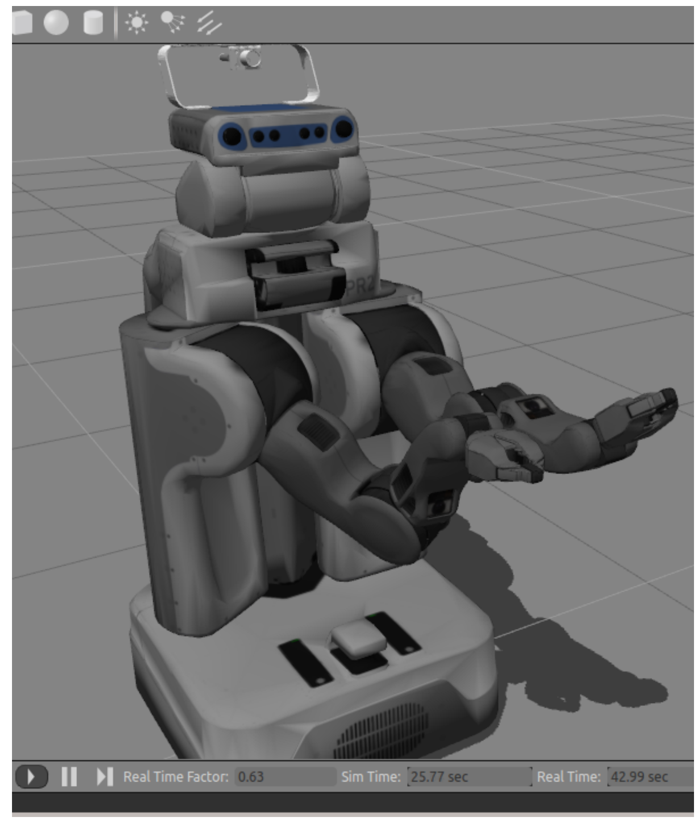
What about a several thousand dollar robot?



ROS/ROS 2: Wrap-up

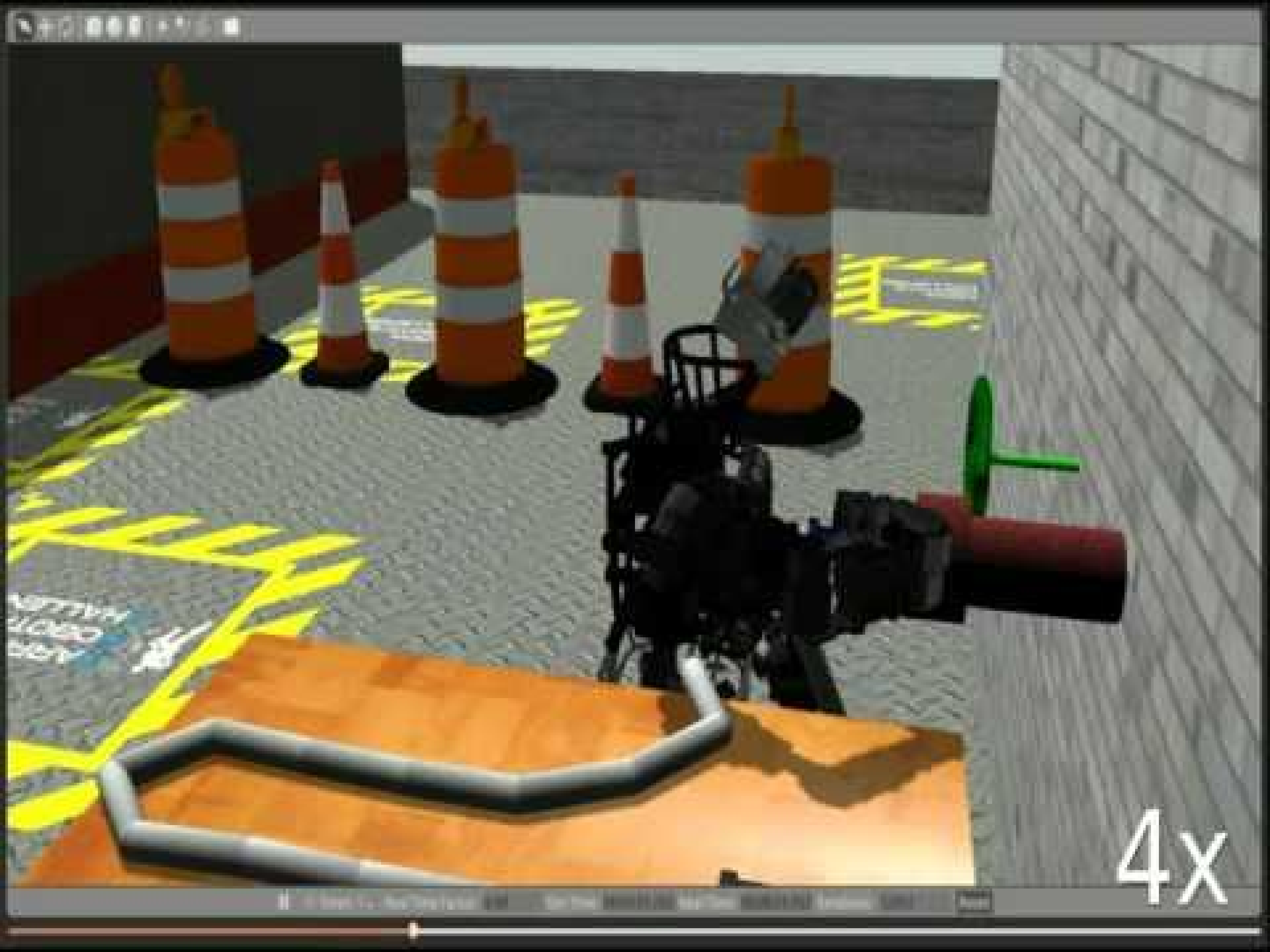
On second thought... how about a free simulation?





GAZEBO

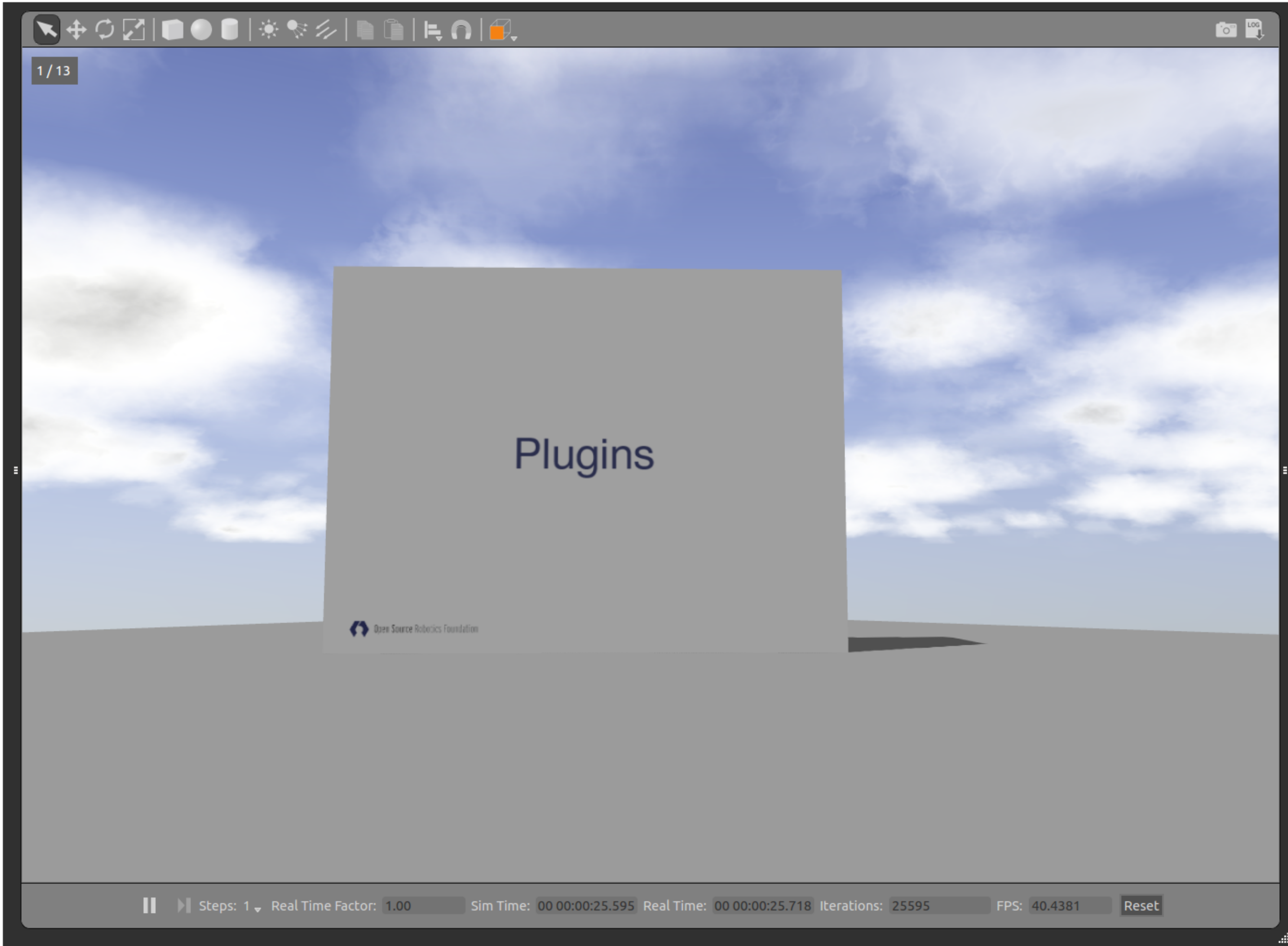
http://gazebosim.org/

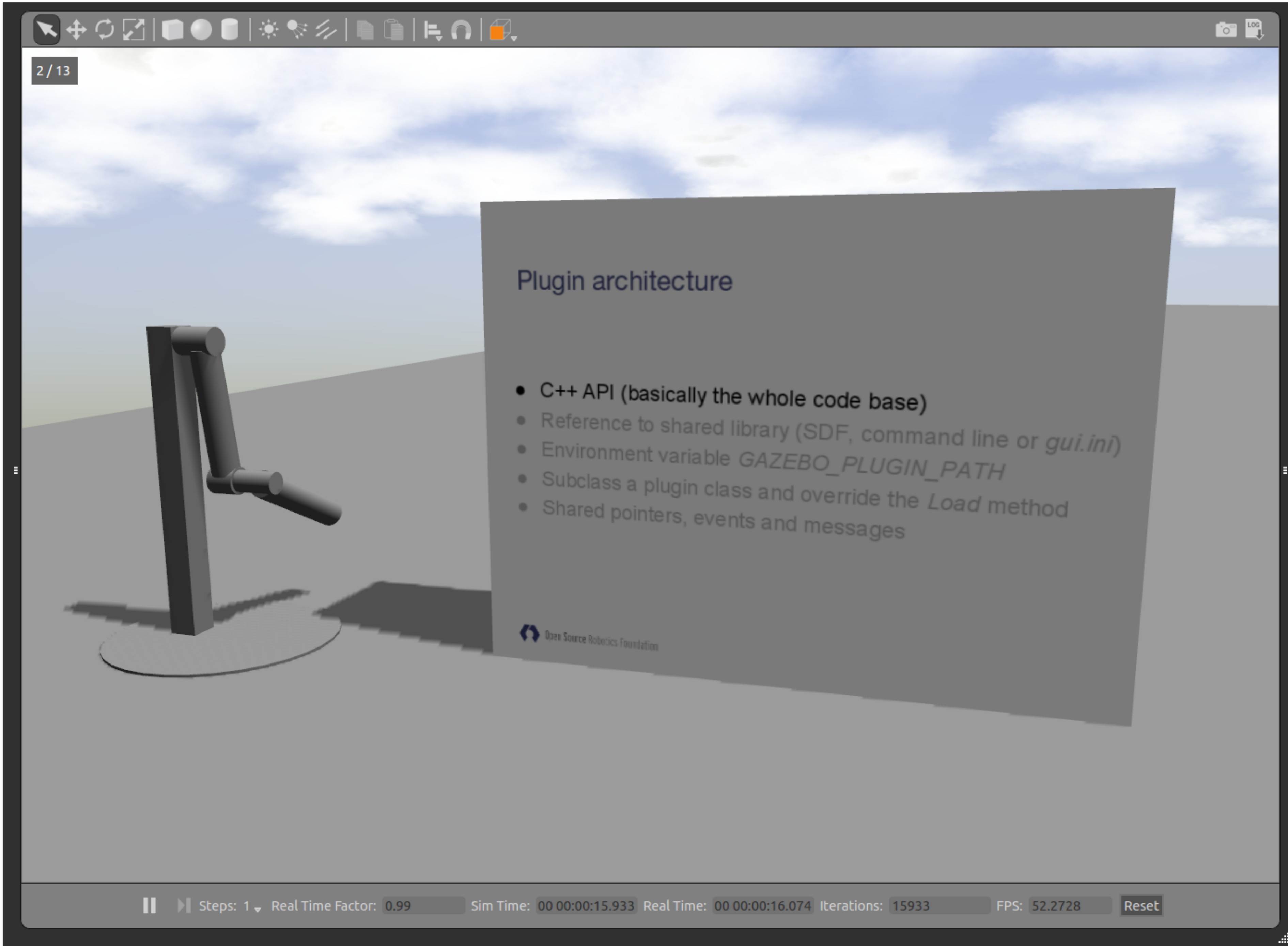

4x

Gazebo 6

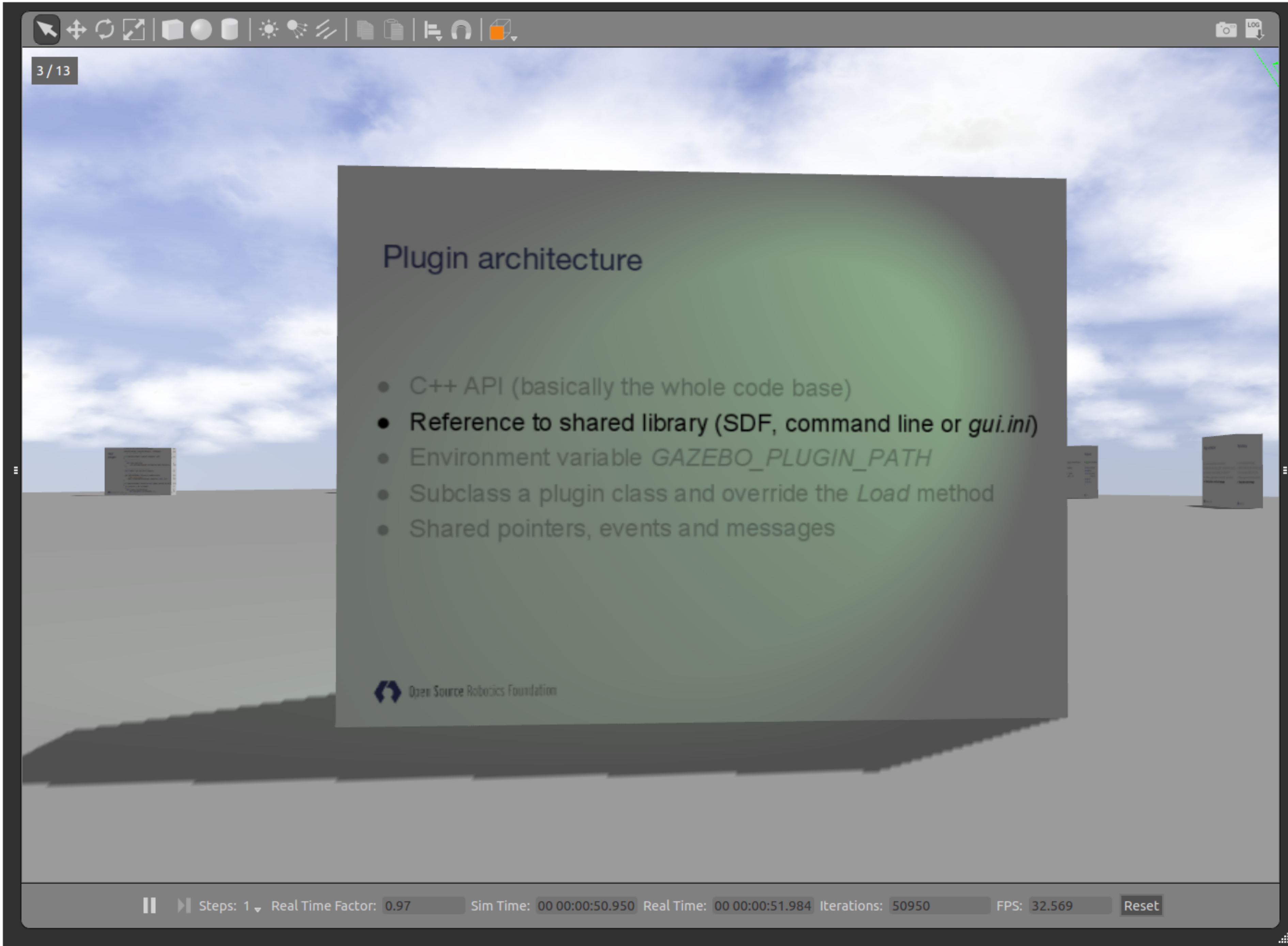
Birth	Fall 2002
Downloads	1k/month
Lines of code	266k
Lines of comments	89k
Test function coverage	47.9%
Test branch coverage	39.1%
Tests	901
Contributors	60+

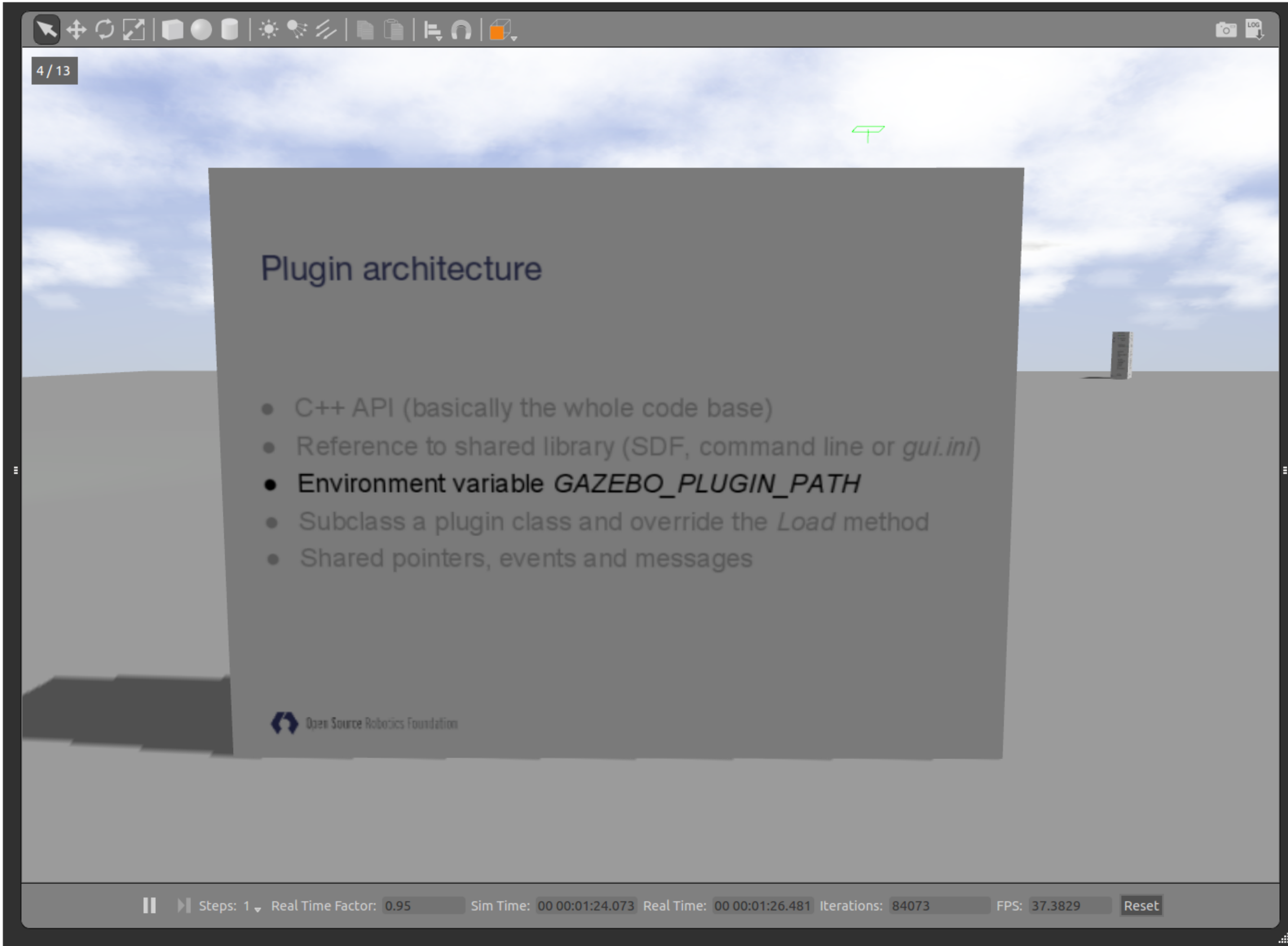
Gazebo 6 dependencies

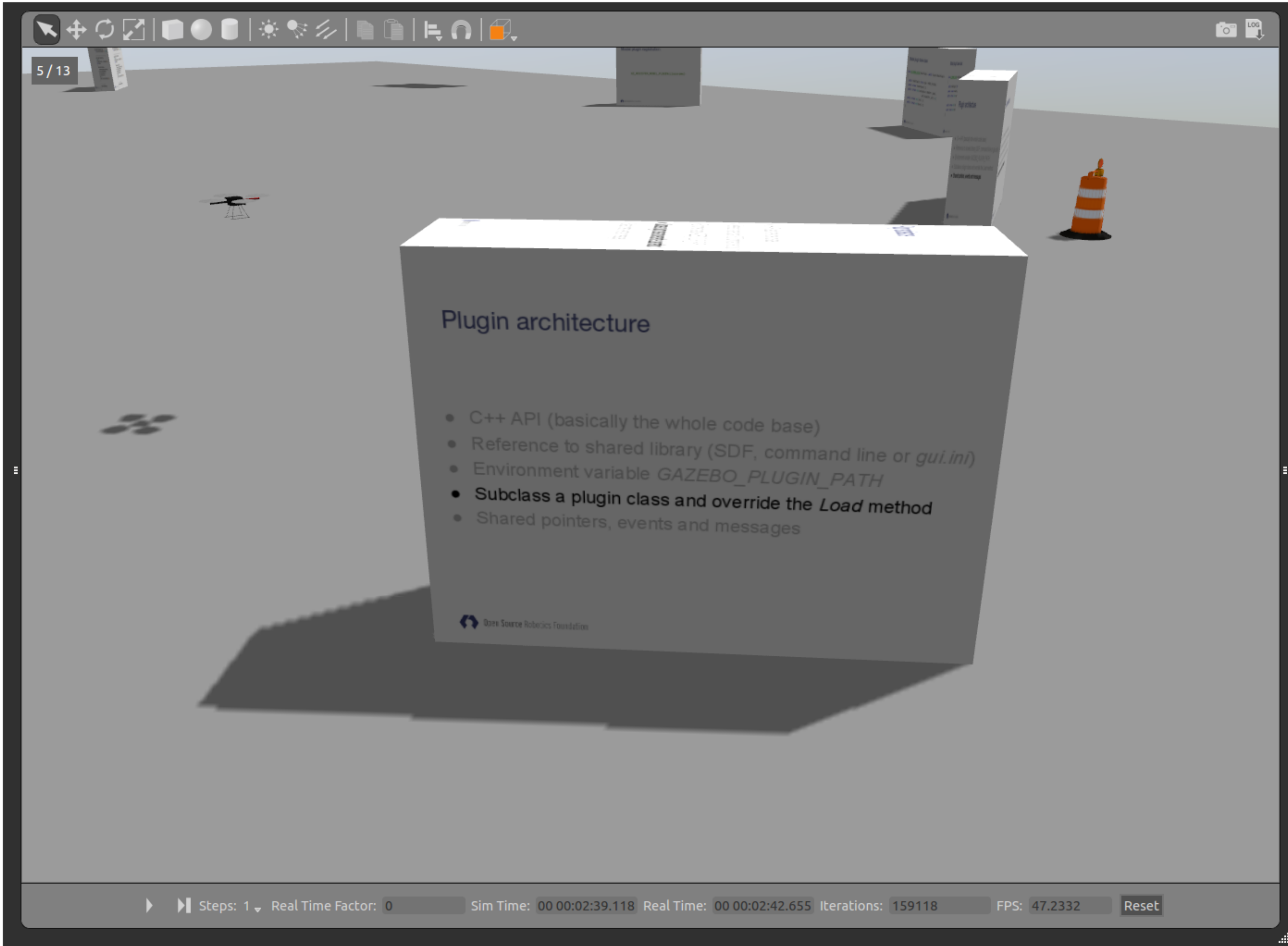

1 / 13
Plugins
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 1.00
Sim Time: 00 00:00:25.595
Real Time: 00 00:00:25.718
Iterations: 25595
FPS: 40.4381
Reset

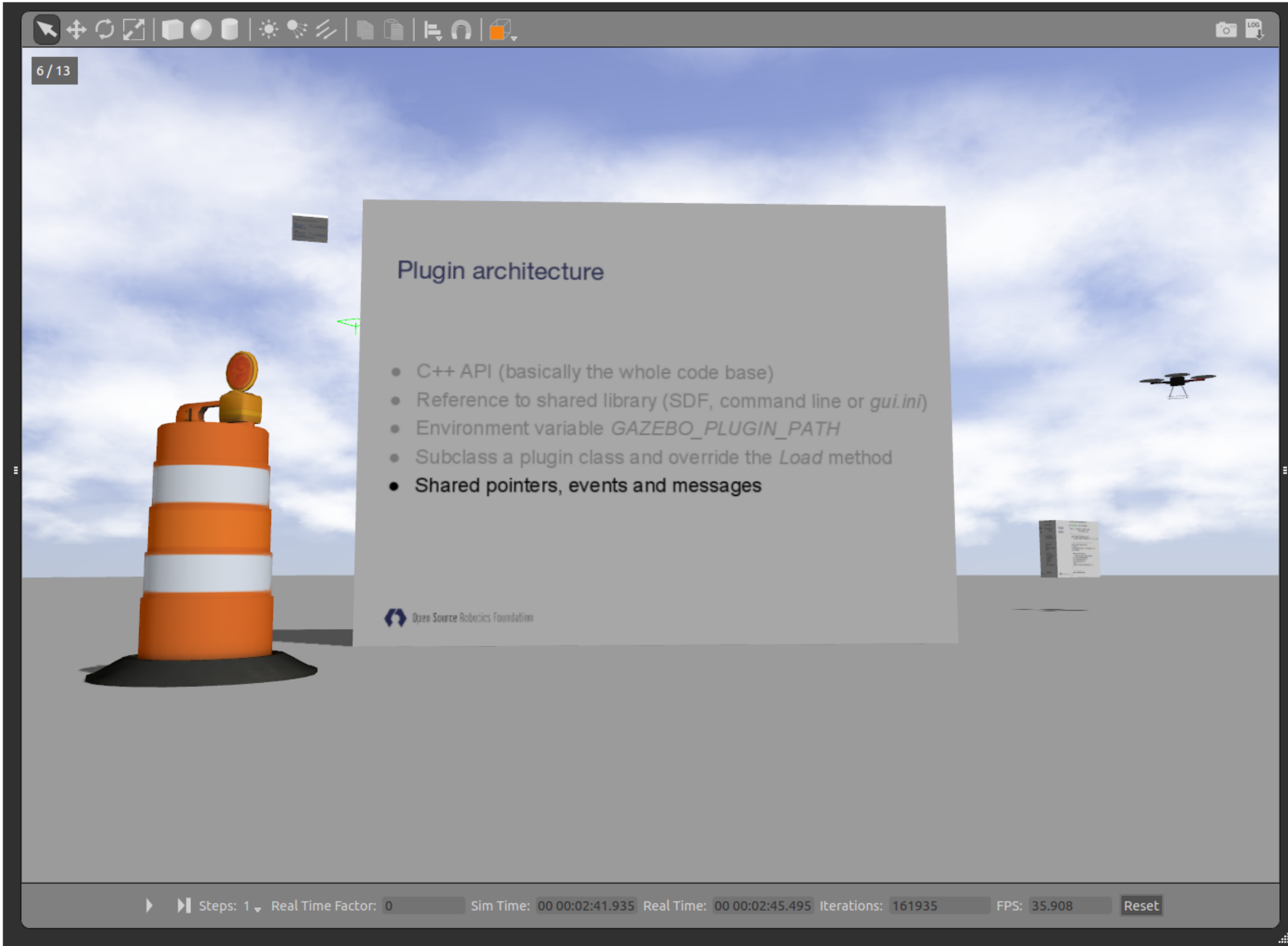


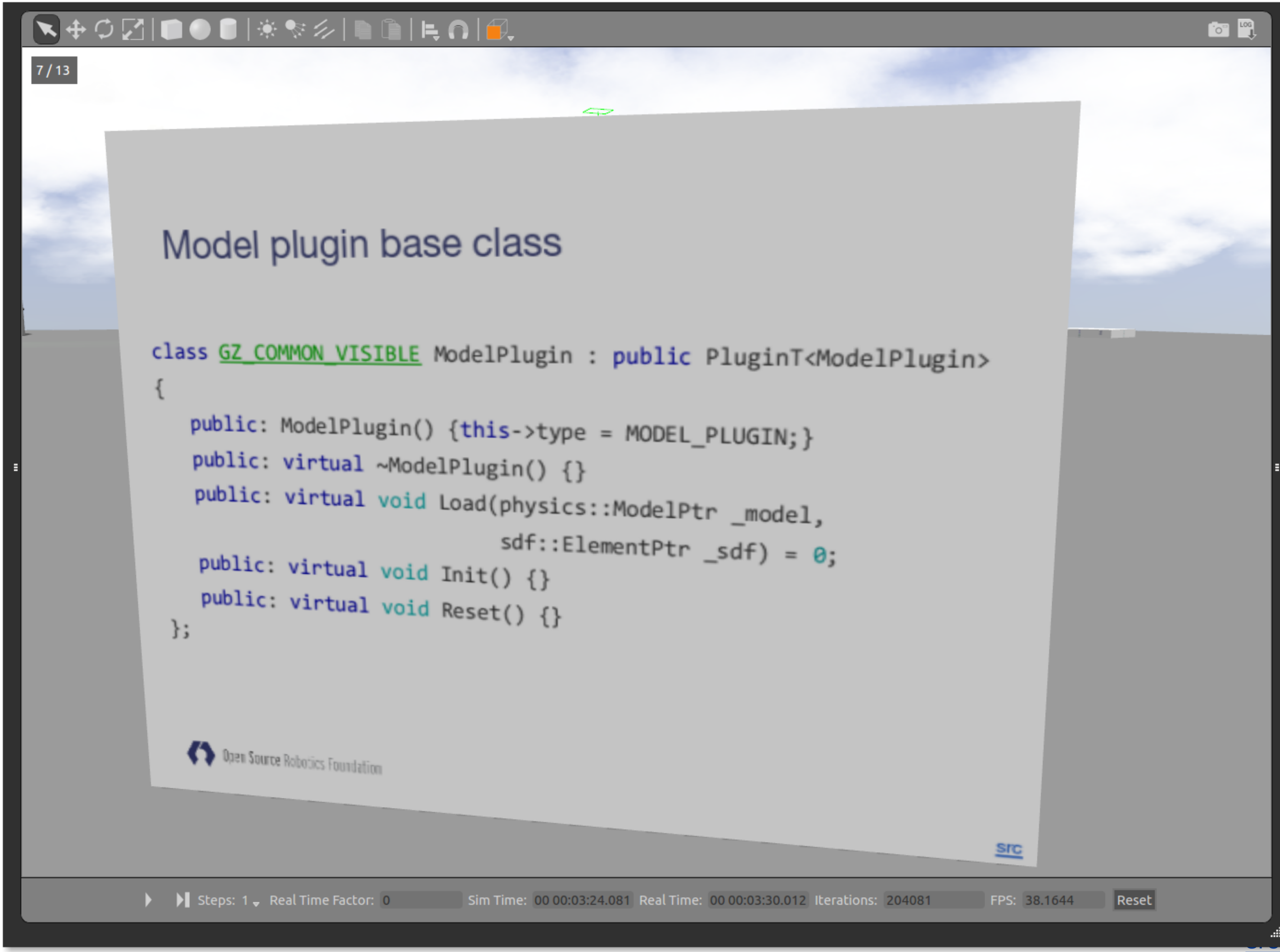
2 / 13
Plugin architecture
C++ API (basically the whole code base)
Reference to shared library (SDF, command line or gui.ini)
Environment variable GAZEBO_PLUGIN_PATH
Subclass a plugin class and override the Load method
Shared pointers, events and messages
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0.99
Sim Time: 00 00:00:15.933
Real Time: 00 00:00:16.074
Iterations: 15933
FPS: 52.2728
Reset

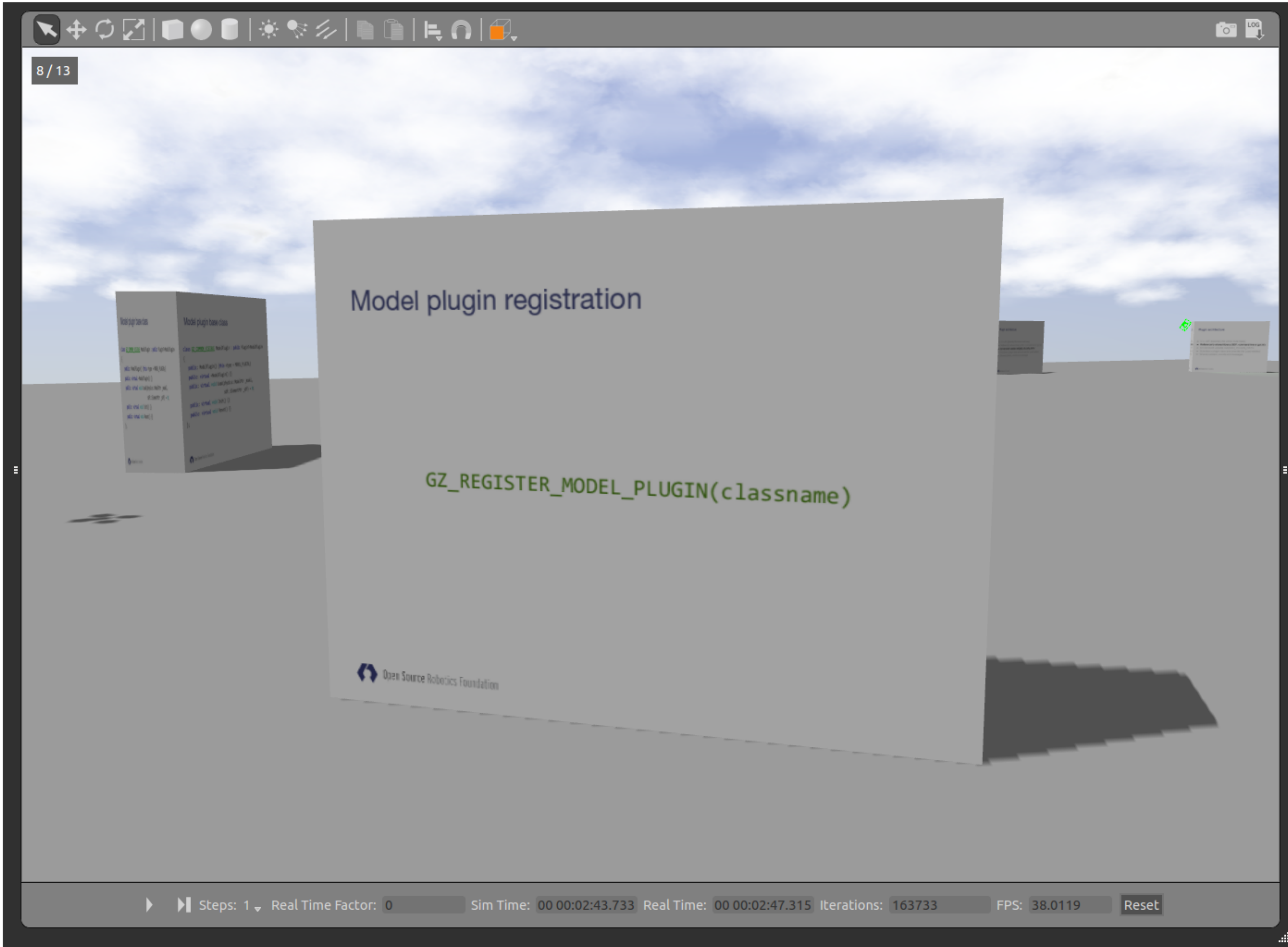

3 / 13
Plugin architecture
C++ API (basically the whole code base)
Reference to shared library (SDF, command line or gui.ini)
Environment variable GAZEBO_PLUGIN_PATH
Subclass a plugin class and override the Load method
Shared pointers, events and messages
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0.97
Sim Time: 00 00:00:50.950
Real Time: 00 00:00:51.984
Iterations: 50950
FPS: 32.569
Reset


4 / 13
Plugin architecture
C++ API (basically the whole code base)
Reference to shared library (SDF, command line or gui.ini)
Environment variable GAZEBO_PLUGIN_PATH
Subclass a plugin class and override the Load method
Shared pointers, events and messages
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0.95
Sim Time: 00 00:01:24.073
Real Time: 00 00:01:26.481
Iterations: 84073
FPS: 37.3829
Reset

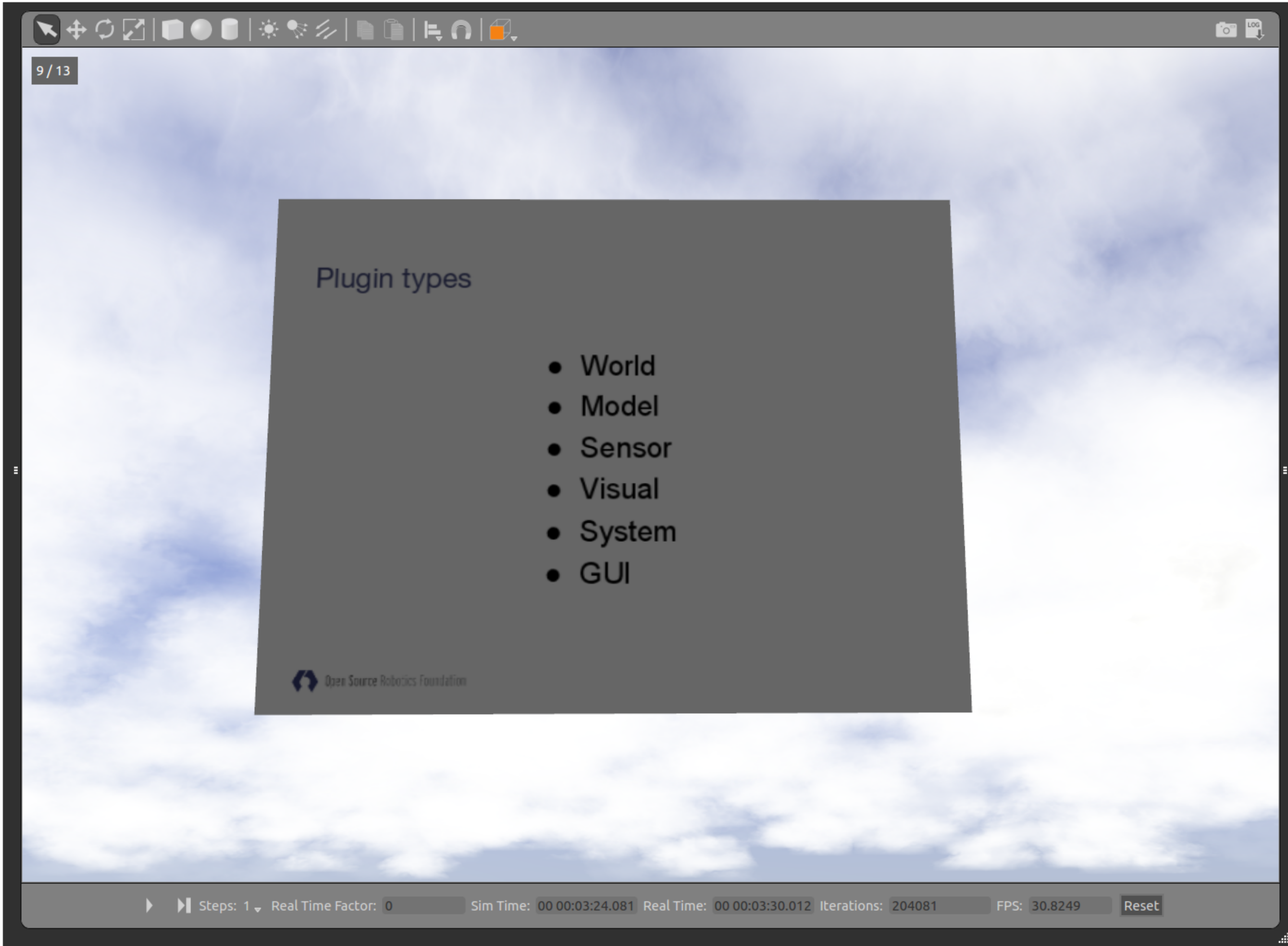

5 / 13
Plugin architecture
C++ API (basically the whole code base)
Reference to shared library (SDF, command line or gui.ini)
Environment variable GAZEBO_PLUGIN_PATH
Subclass a plugin class and override the Load method
Shared pointers, events and messages
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:02:39.118
Real Time: 00 00:02:42.655
Iterations: 159118
FPS: 47.2332
Reset

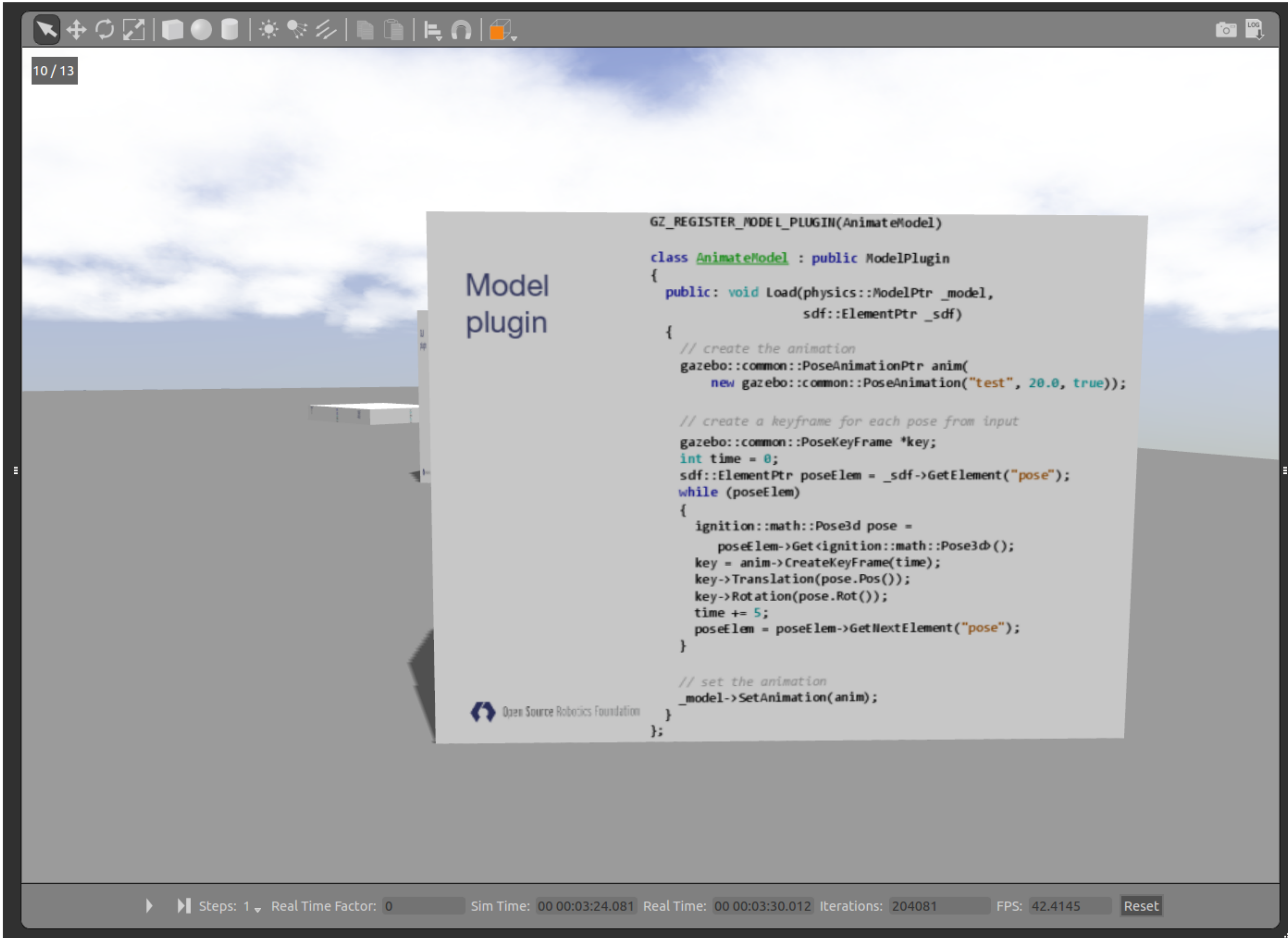

6 / 13
Plugin architecture
C++ API (basically the whole code base)
Reference to shared library (SDF, command line or gui.ini)
Environment variable GAZEBO_PLUGIN_PATH
Subclass a plugin class and override the Load method
Shared pointers, events and messages
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:02:41.935
Real Time: 00 00:02:45.495
Iterations: 161935
FPS: 35.908
Reset


7 / 13
Model plugin base class
class GZ_COMMON_VISIBLE ModelPlugin : public PluginT<ModelPlugin>
{
public: ModelPlugin() {this->type = MODEL_PLUGIN;}
public: virtual ~ModelPlugin() {}
public: virtual void Load(physics::ModelPtr _model,
sdf::ElementPtr _sdf) = 0;
public: virtual void Init() {}
public: virtual void Reset() {}
};
Open Source Robotics Foundation
src
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:03:24.081
Real Time: 00 00:03:30.012
Iterations: 204081
FPS: 38.1644
Reset



8 / 13
Model plugin registration
GZ_REGISTER_MODEL_PLUGIN(classname)
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:02:43.733
Real Time: 00 00:02:47.315
Iterations: 163733
FPS: 38.0119
Reset

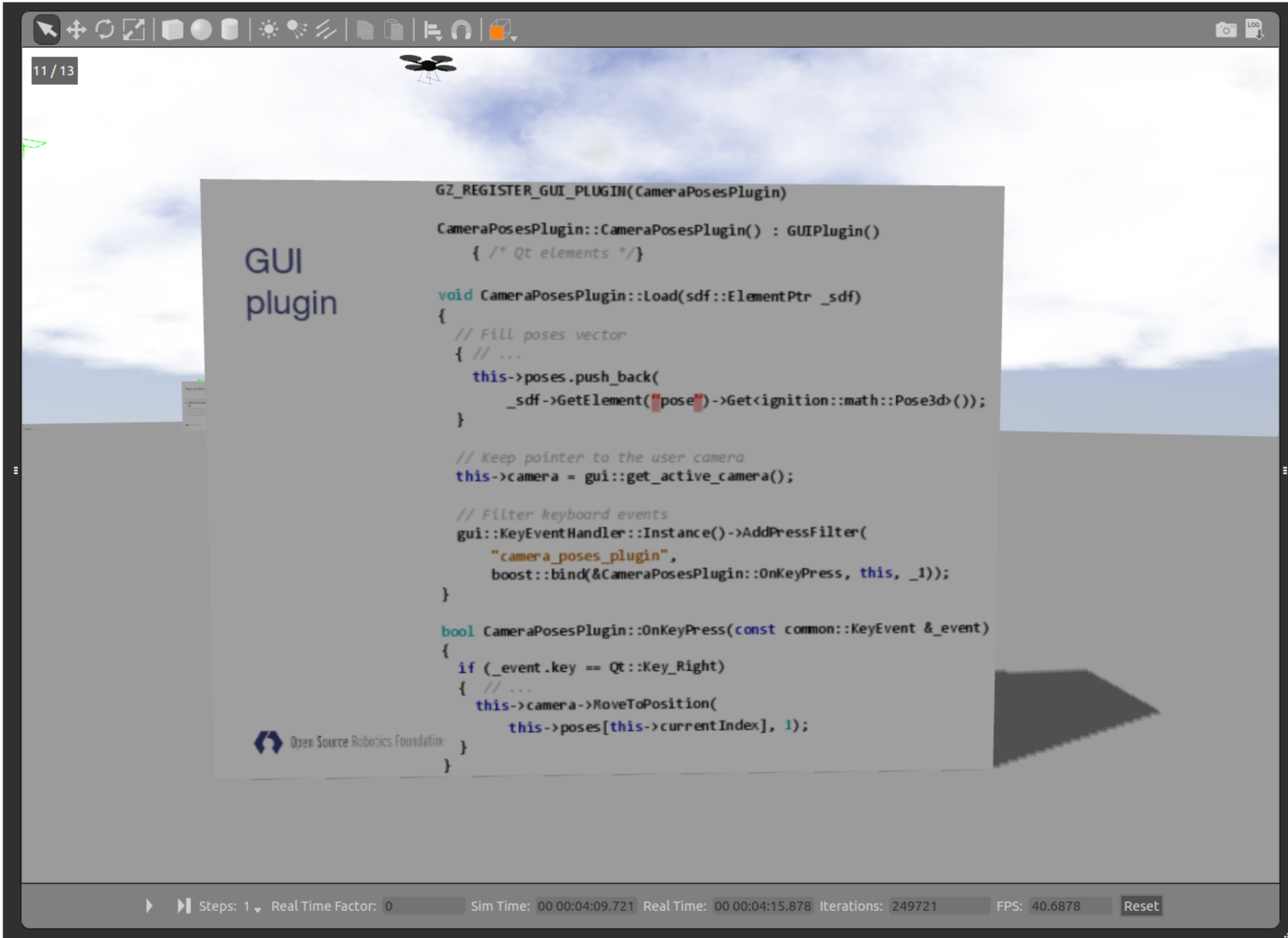

9 / 13
Plugin types
• World
• Model
• Sensor
• Visual
• System
• GUI
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:03:24.081
Real Time: 00 00:03:30.012
Iterations: 204081
FPS: 30.8249
Reset

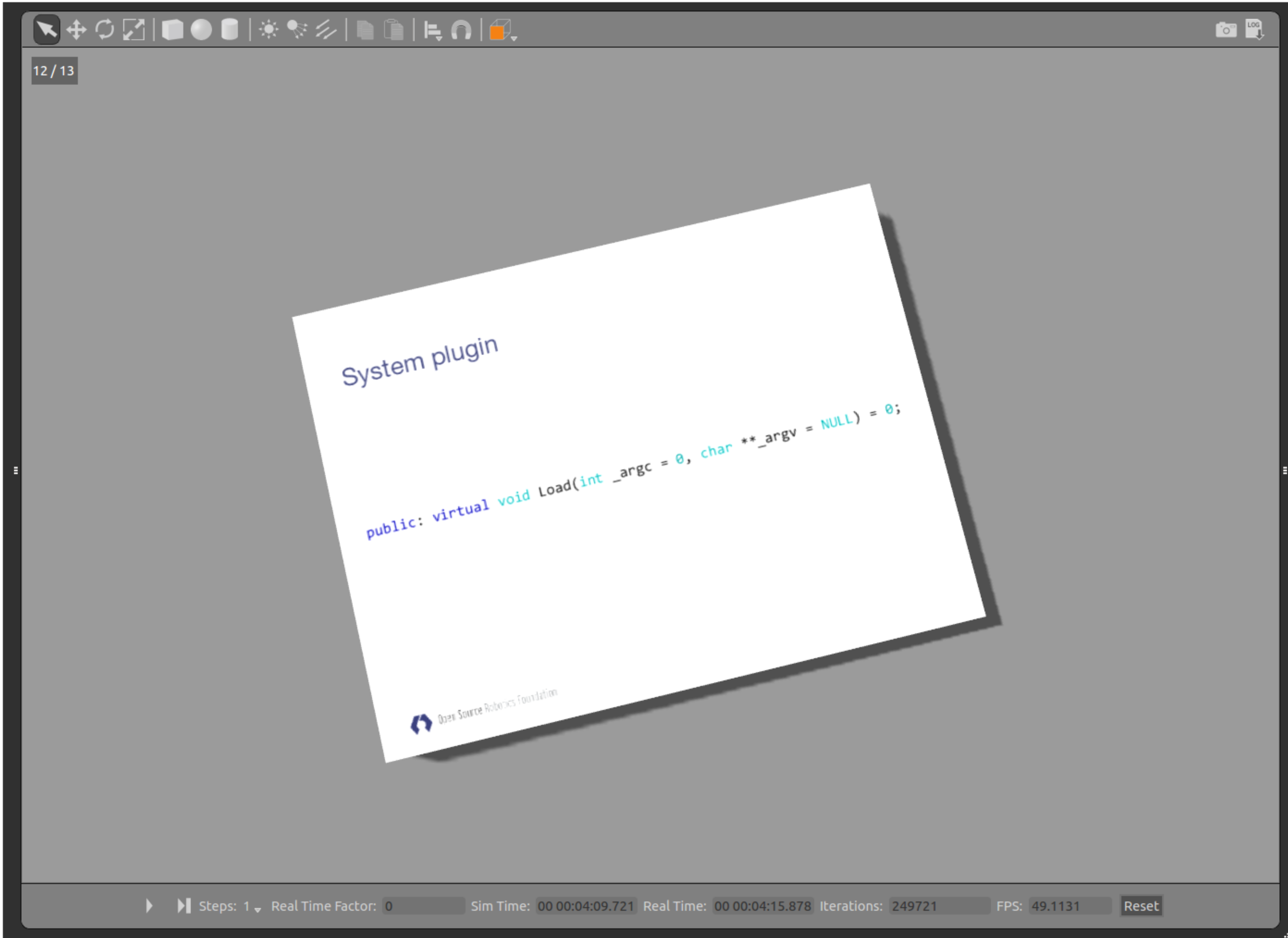

10 / 13
Model plugin
GZ_REGISTER_MODEL_PLUGIN(AnimateModel)

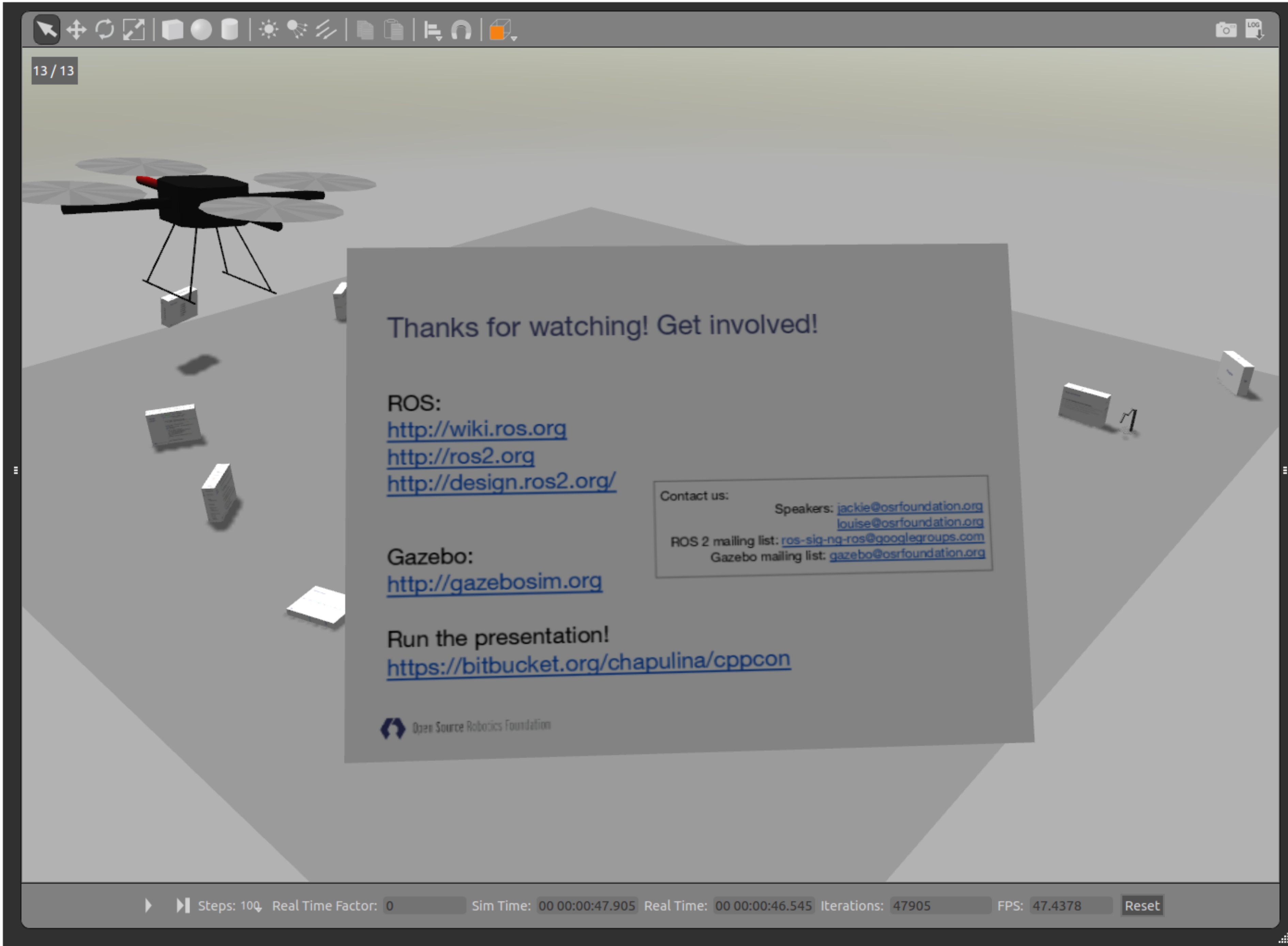
class AnimateModel : public ModelPlugin
{
  public: void Load(physics::ModelPtr _model,
                    sdf::ElementPtr _sdf)
  {
    // create the animation
    gazebo::common::PoseAnimationPtr anim(
        new gazebo::common::PoseAnimation("test", 20.0, true));

    // create a keyframe for each pose from input
    gazebo::common::PoseKeyFrame *key;
    int time = 0;
    sdf::ElementPtr poseElem = _sdf->GetElement("pose");
    while (poseElem)
    {
      ignition::math::Pose3d pose =
         poseElem->Get<ignition::math::Pose3d>();
      key = anim->CreateKeyFrame(time);
      key->Translation(pose.Pos());
      key->Rotation(pose.Rot());
      time += 5;
      poseElem = poseElem->GetNextElement("pose");
    }

    // set the animation
    _model->SetAnimation(anim);
  }
};
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:03:24.081
Real Time: 00 00:03:30.012
Iterations: 204081
FPS: 42.4145
Reset


11 / 13
GUI
plugin
GZ_REGISTER_GUI_PLUGIN(CameraPosesPlugin)
CameraPosesPlugin::CameraPosesPlugin() : GUIPlugin()
{ /* Qt elements */}
void CameraPosesPlugin::Load(sdf::ElementPtr _sdf)
{
// Fill poses vector
{ // ...
this->poses.push_back(
_sdf->GetElement("pose")->Get<ignition::math::Pose3d>());
}
// Keep pointer to the user camera
this->camera = gui::get_active_camera();
// Filter keyboard events
gui::KeyEventHandler::Instance()->AddPressFilter(
"camera_poses_plugin",
boost::bind(&CameraPosesPlugin::OnKeyPress, this, _1));
}
bool CameraPosesPlugin::OnKeyPress(const common::KeyEvent &_event)
{
if (_event.key == Qt::Key_Right)
{ // ...
this->camera->MoveToPosition(
this->poses[this->currentIndex], 1);
}
}
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:04:09.721
Real Time: 00 00:04:15.878
Iterations: 249721
FPS: 40.6878
Reset


12 / 13
System plugin
public: virtual void Load(int _argc = 0, char **_argv = NULL) = 0;
Open Source Robotics Foundation
Steps: 1
Real Time Factor: 0
Sim Time: 00 00:04:09.721
Real Time: 00 00:04:15.878
Iterations: 249721
FPS: 49.1131
Reset


13 / 13
Thanks for watching! Get involved!
ROS:
http://wiki.ros.org
http://ros2.org
http://design.ros2.org/
Gazebo:
http://gazebosim.org
Run the presentation!
https://bitbucket.org/chapulina/cppcon
Contact us:
Speakers: jackie@osrfoundation.org
louise@osrfoundation.org
ROS 2 mailing list: ros-sig-ng-ros@googlegroups.com
Gazebo mailing list: gazebo@osrfoundation.org
Open Source Robotics Foundation
Steps: 100
Real Time Factor: 0
Sim Time: 00 00:00:47.905
Real Time: 00 00:00:46.545
Iterations: 47905
FPS: 47.4378
Reset

Thanks for watching! Get involved!

ROS:
http://wiki.ros.org
http://ros2.org
http://design.ros2.org/

Gazebo:
http://gazebosim.org

Run the presentation!
https://bitbucket.org/chapulina/cppcon

Contact us:
Speakers: jackie@osrfoundation.org
louise@osrfoundation.org
ROS 2 mailing list: ros-sig-ng-ros@googlegroups.com
Gazebo mailing list: gazebo@osrfoundation.org