Customizing 3D Slicer

Jean-Christophe Fillion-Robin

Free Biomedical Image Analysis and Visualization 3-day Course
March 14th 2019

jfillionr

jcfr

thehiddenbrain

Kitware

Outline

- Writing correct and understandable code
- Slicer Architecture
- Slicer Scripting Interface
- Slicer Scripted Module
- Slicer Jupyter
- Creating Slicer Extensions
- Slicer Plugin Infrastructure
- Slicer Custom Applications
  - Customization Approaches
  - Packaging
  - Installing Additional Python Packages
  - Build System Overview
  - Python Wrapping Infrastructure
- Slicer and the Cloud

Kitware

Acknowledgments

Slides with the text “Source: PerkLab Bootcamp” in the top-right corner have been adapted from slides available at https://github.com/PerkLab/PerkLabBootcamp

Introduction: Writing correct and understandable code

See https://github.com/PerkLab/PerkLabBootcamp/tree/master/Doc
by Andras Lasso, Laboratory for Percutaneous Surgery (Perk Lab)

Kitware

Slicer Architecture

General Principle: Model View Controller pattern

- Model
  - Data handled by program (images, segmentations, annotations)
- View
  - Visual representation of data in the Model, e.g., spreadsheet, chart, 3D rendering
  - Respond to update notifications from Model
- Controller
  - Modifies the Model

Kitware

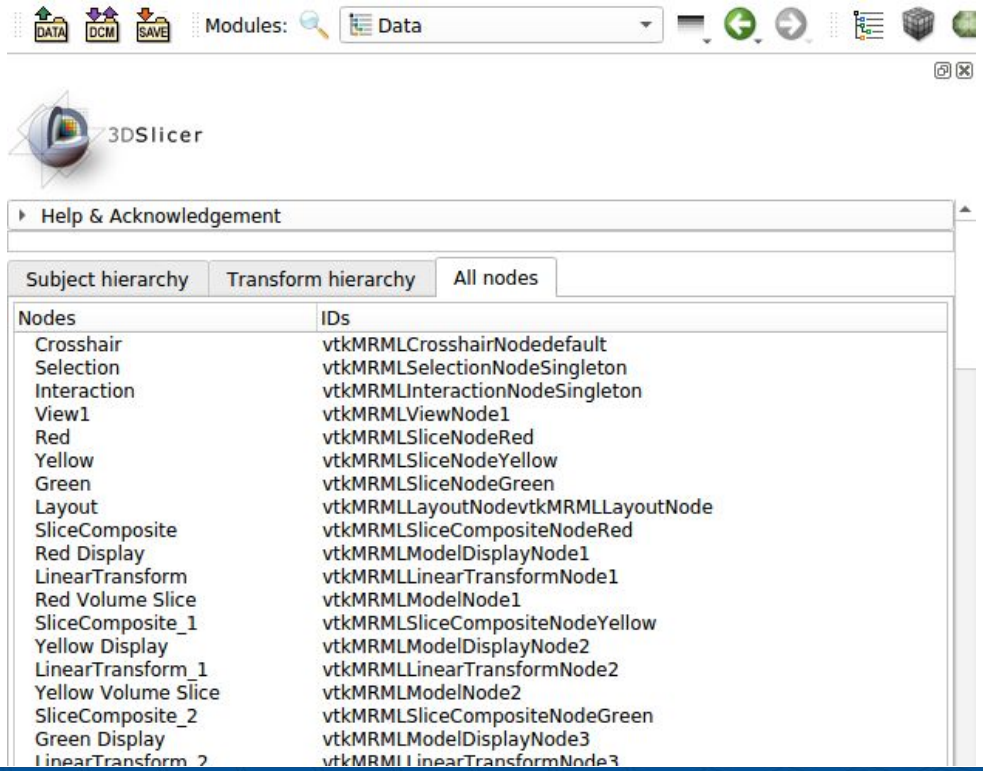
Slide courtesy of Cory Quammen, Kitware Inc.

Slicer Data Model

- Why? Data organization and serialization
- How? Supported by MRML library (pronounced as ‘mermel)
  - API for managing medical image data types, their vizualization and storage
  - Data types: Volumes, Models, Transforms, Fiducials, Cameras, etc

Kitware

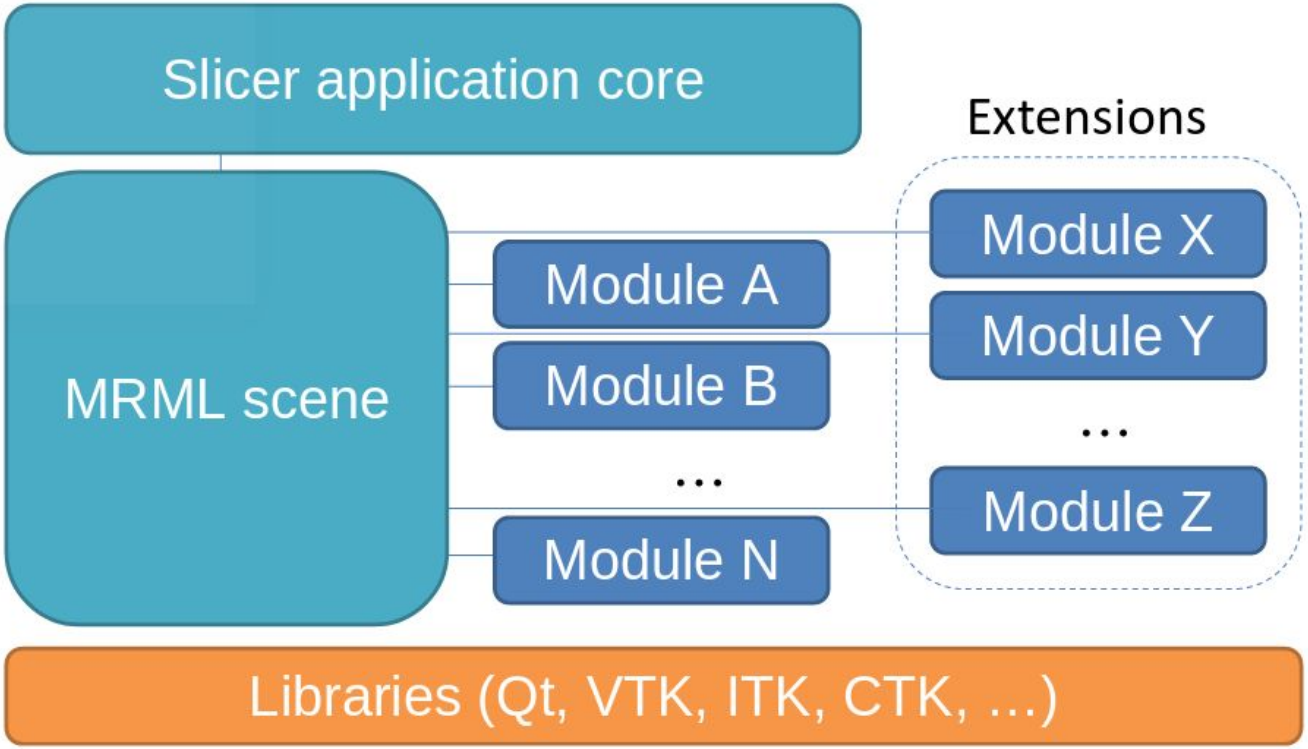
Slicer Data Module



Kitware

Source: PerkLab Bootcamp

Slicer Application Architecture



MRML (pronounced as ‘mermel)

- MRML = Medical Reality Markup Language
  - XML-based format used to serialize MRML scene on disk (.mrml file)
- MRML node
- MRML scene

See https://www.slicer.org/wiki/Documentation/Nightly/Developers/MRML

Kitware

MRML Node (1 / 2)

- Identified by a unique string ID
- References, observations between nodes
- Modules communicate through reading/writing MRML nodes

MRML Node (2 / 2)

- Slicer in-memory data structure managing some data element
- Designed to store
  - state of the Slicer application
  - raw data
  - visualization and storage parameters
- Organized into C++ class hierarchies
  - Derived from vtkMRMLNode

MRML Scene

- Global repository for all data
- Collection of all instances of Slicer MRML nodes
- API to manages nodes : add, delete, find, find by type, etc.
- Provides persistence of MRML nodes (reading/writing to/from XML file)

Kitware

Why different nodes “types” ?

- same data can be
  - visualized in various ways
  - stored in different formats.

MRML Node types

- Data node:
  - store the raw data (such as vtkMRMLScalarVolumeNode stores the voxel of a volume, spacing, position, orientation).
- Display node:
  - describes how the data should be visualized (there can be multiple display nodes for the same raw data, e.g., one for volume rendering and one for displaying as an image slice)
- Storage node:
  - describes how the data should be stored persistently on disk (file format, file name)

Kitware

MRML Node Interfaces

- Displayable
- Storable
- Transformable

MRML: What else ?

- MRML node attributes
  - store custom attributes as (attribute name; attribute value) pairs.
  - See https://www.slicer.org/wiki/Documentation/Nightly/Developers/MRML#MRML_node_attributes

- MRML References
  - MRML nodes can reference and observe other MRML nodes using the node reference API
  - Framework takes care of
    - read/write/copy of node references
    - updating references on scene import
    - adding and deleting nodes
  - See https://www.slicer.org/wiki/Documentation/Nightly/Developers/MRML/NodeReferences

Kitware

MRML Node Responsibilities

- Store data
- Serialization to/from XML for file storage
- No display or processing methods
- Modules do not need to know about each other!

Kitware

Architecture: View

- Maintains state of the visual elements
- Functionality provided by
  - GUI
  - Displayable manager classes
    - maintain consistency between the internal MRML state (Model)
    - and the visual appearance of the Qt GUI data viewers of the application

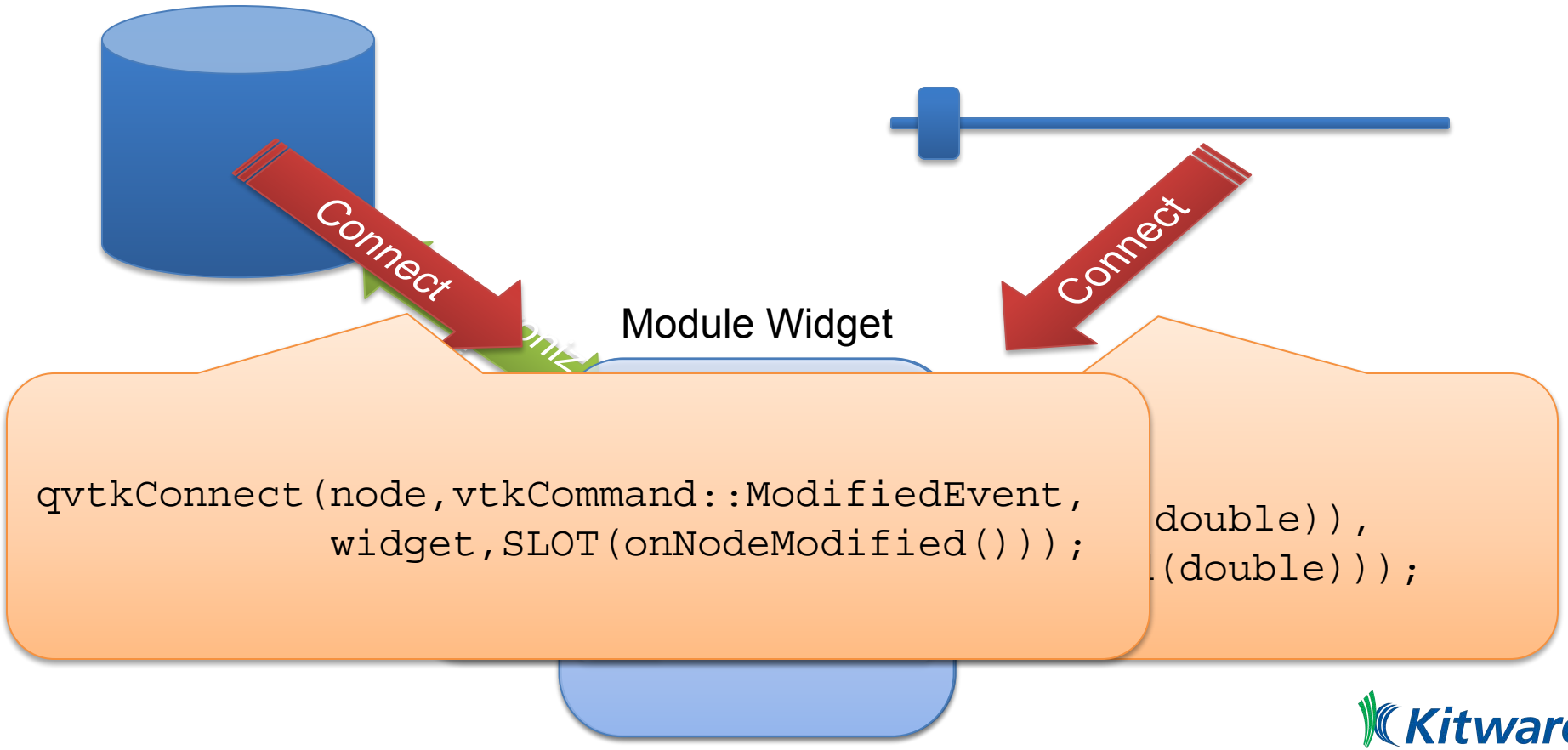
Architecture: View and Interaction

- Updates to GUI and visualization components trigger events that propagate via the corresponding GUI classes to the “stateful” MRML nodes.

- Displayable managers maintain consistent visualization of certain data elements in multiple views/viewers.

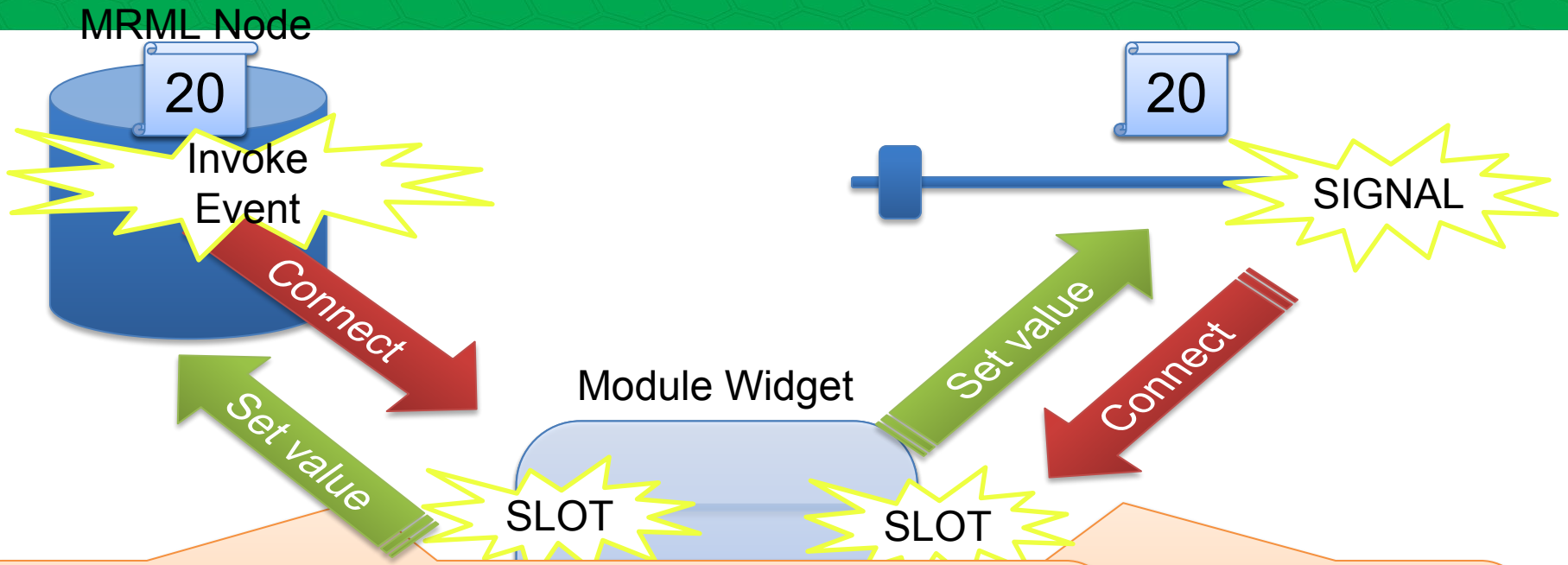
Kitware

MRML Node



Kitware

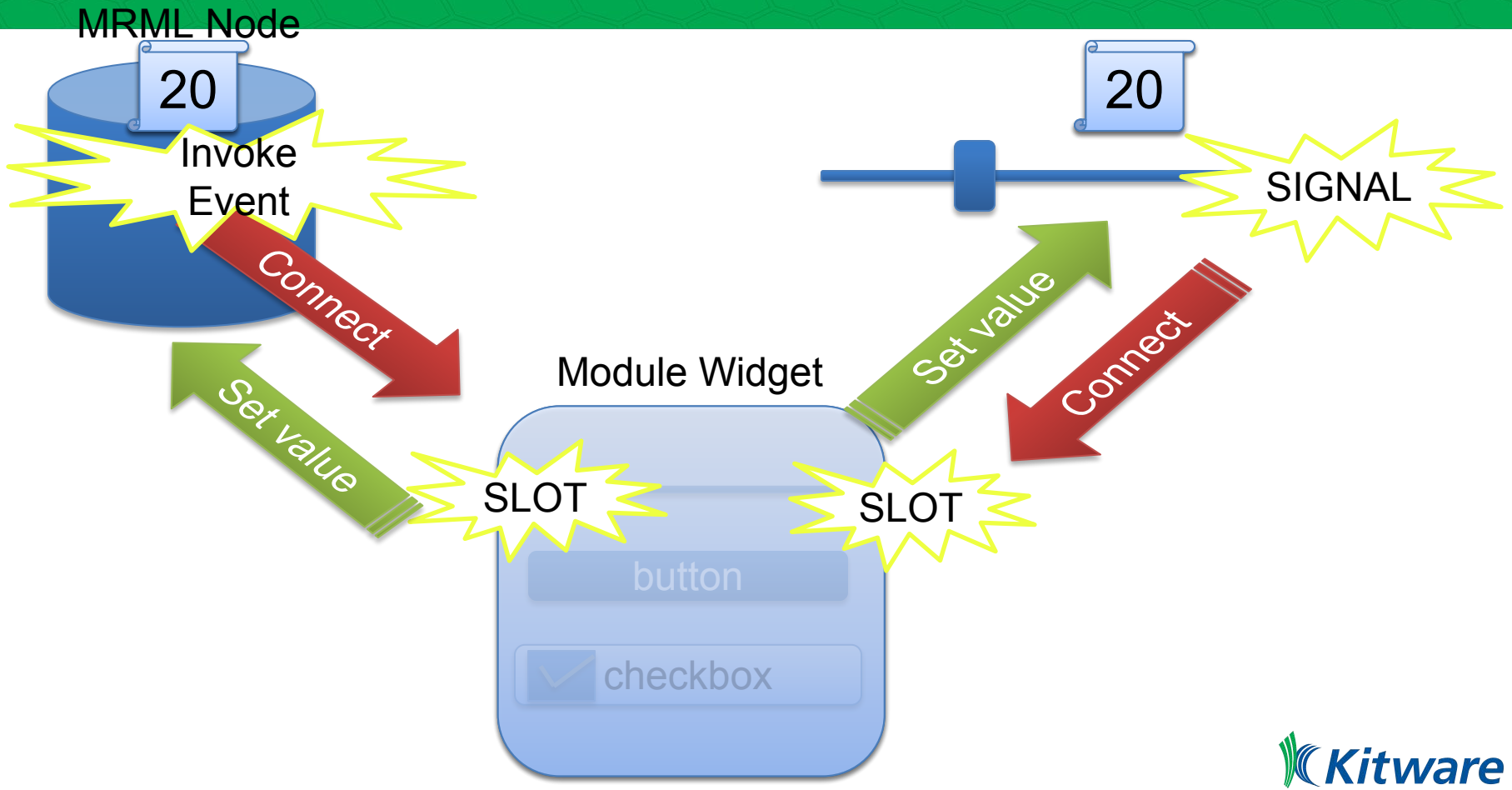
Slide courtesy of Julien Finet, Kitware Inc.




```
void qSlicerModuleWidget::onMRMLNodeModified(
  vtkMRMLNode* node)
{
 slider->setValue(node->GetValue());
}
```

```
anged(double
```


Slide courtesy of Julien Finet, Kitware Inc.



Kitware

Slide courtesy of Julien Finet, Kitware Inc.

Architecture: Controller (Logic component)

- Encapsulates the processing/analysis functionality
- Does not depend on the existence of GUI
- Fully aware of MRML data structures

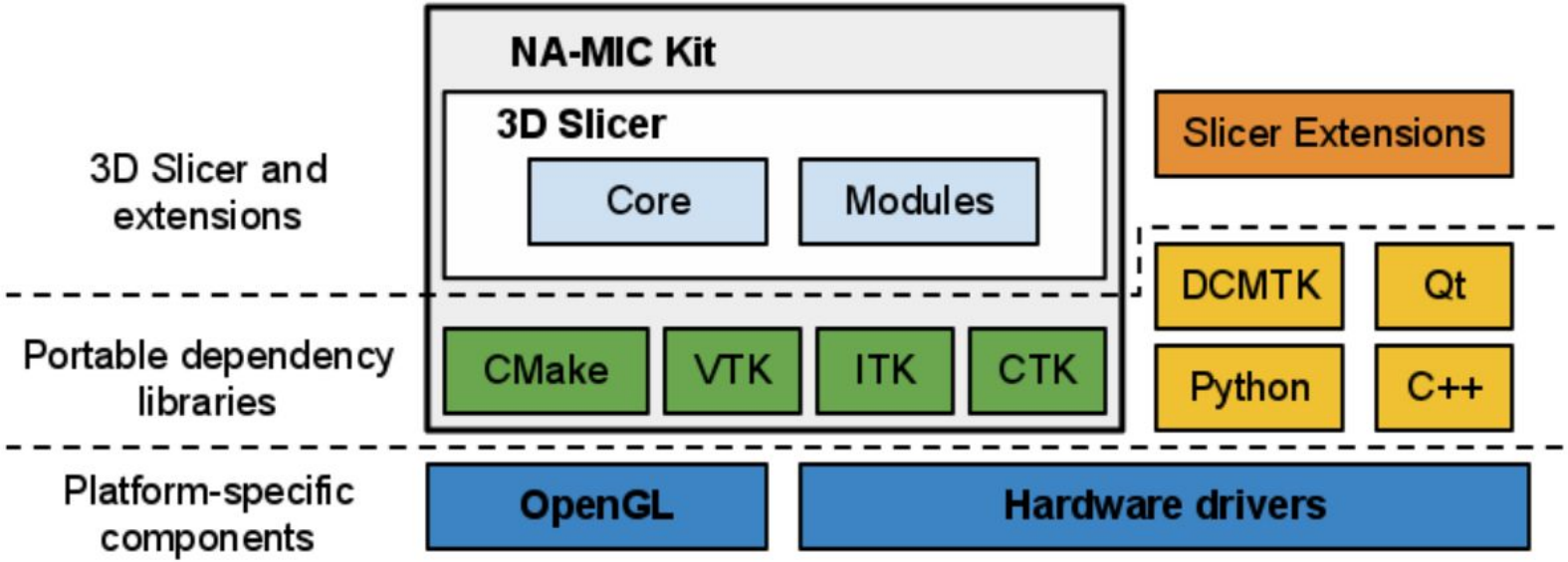
https://www.slicer.org/wiki/Documentation/Nightly/Developers/Logics

Kitware

Architecture:
Communication between view and controller

- Indirect through changes to the MRML data structures
- Logic classes use the MRML nodes for storing the computation results
- View classes can register to receive event updates from the MRML scene / nodes

Modular Infrastructure

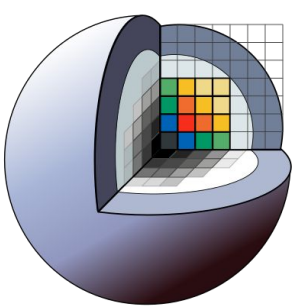


Kitware

Source: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3466397/

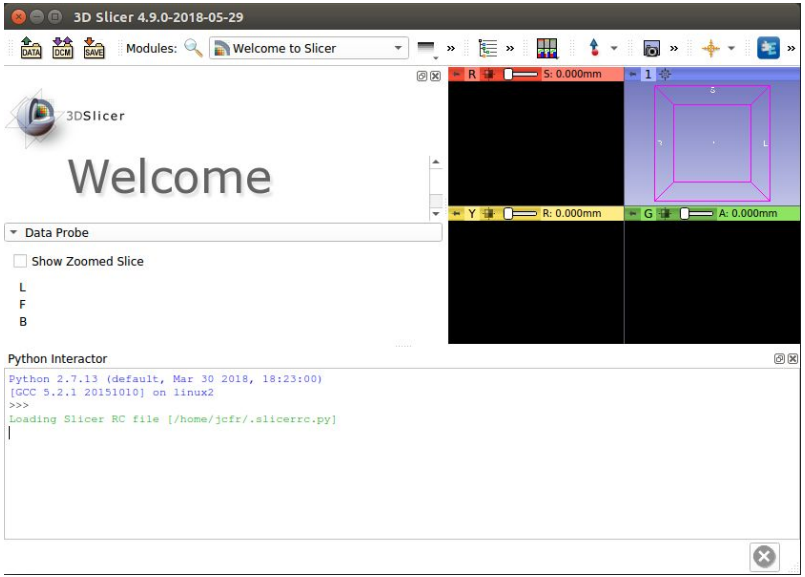
Slicer Scripting Interface

Slicer Python Interactor



+

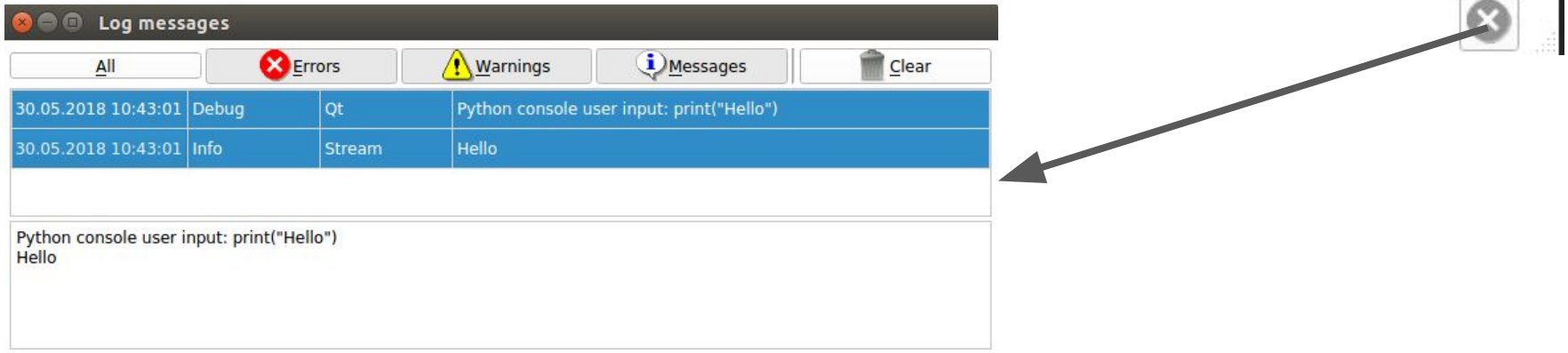
- Menu -> Window->Python Interactor
  - Control-3 on Linux or Windows
  - Command-3 on macOS



Kitware

Hello World

```
>>> print("Hello")
Hello
>>>
```



Slicer Python Interactor: Features and capabilities

- Introspect “live” objects
- Instantiate new objects
- Built-in access to the VTK, Qt, and Slicer wrapped APIs
- Auto-completion
- Namespaces: slicer, ctk, vtk, qt

Kitware

Slicer Python Interactor: Caveats

- Indentation is very important!
- Use spaces, not TABs!
- Slicer Python Interactor is **not** the python interpreter
- Python is embedded
- Integrated with the Qt event loop
- Limit in how sys.stdout, sys.stderr and sys.stdin can be used

Kitware

Source: PerkLab Bootcamp

General Remarks about Python

- **Blocks defined by indentation: 2 spaces (for historical reason)**
- Case sensitive
- Comments

```
# This whole row is a comment
"""This is a potentially multi-line comment"""
```

- An object refers to itself as **self** (in C++: **this**)

Kitware

Source: PerkLab Bootcamp

Text editor / IDE

- Using a proper text editor is essential
  - Replace all, Easy comment/uncomment, indent, …
  - Syntax highlighting
  - Keyboard shortcuts
  - Recommended:
    Windows-only: **Notepad++**, Mac-only: Xcode
    Cross-platform: Atom, Sublime Text,
- Integrated development environment:
  - Text editor + debugger, code browser, …
  - Recommended: **PyCharm**, LiClipse

Kitware

Examples and Exercises

- Goal:
  - Introduce the python interface
  - Improve our understanding of the Slicer data model
  - Grasp the key concepts to create user interfaces

- Convention:
  - Text in `fixed font` can be executed in the Python Interactor

Kitware

Example “Loading Sample Data”

```
import SampleData
sampleDataLogic = SampleData.SampleDataLogic()
sampleDataLogic.downloadMRHead()
```

Kitware

“Loading Sample Data”” Example Analysis

- Search for python module or package named “SampleData” and import it

```
import SampleData
```

- Instantiate the logic class provided by the SampleData module

```
sampleDataLogic = SampleData.SampleDataLogic()
```

- Sychronously download and load the dataset

```
sampleDataLogic.downloadMRHead()
```

Kitware

Example “Get reference to loaded dataset”

```
node = getNode('MRHead')
```

Kitware

“Get reference to loaded dataset” Example Analysis

- Equivalent to `slicer.util.getNode()`
  - Note: This should be used in scripts
- `getNode()` is a convenience method only available in the interpretor namespace
- “MRHead” is the name of the dataset
- **Problem**: How to uniquely reference an object ?

Kitware

Exercise 1: Find the ID of the node

- Method 1: Using the Data module
- Method 2: Using the python interactor
  - Hint: `node.Get` <TAB>

Kitware

Getting help

```
>>> help(getNode)
Help on function getNode in module slicer.util:

getNode(pattern='*', index=0, scene=None)
Return the indexth node where name or id matches ``pattern``.
By default, ``pattern`` is a wildcard and it returns the first node
associated with ``slicer.mrmlScene``.
Throws MRMLNodeNotFoundException exception if no node is found
that matches the specified pattern.
```

Kitware

Source: PerkLab Bootcamp

Information about variables

- Get variable type and pointer: enter the variable name

```
v
(vtkMRMLScalarVolumeNode)0000008FF76243B8
```

Note: It's always good to check the variable after you create it

- Show node content: all members and attributes inheritance tree

```
print(v)
```

- Show node API: description of all methods

```
help(v)
```

Kitware

Slicer Python Package&Module Organization

- ctk
- slicer
  - **app**
  - **modules**
  - **mrmlScene**
  - util
  - ...
- qt
- vtk

Kitware

slicer.app, slicer.mrmlScene

- Associated with corresponding instance of
  - Running Slicer application
  - Instantiated scene

```
>>> slicer.app.mrmlScene() == slicer.mrmlScene
True
```

Kitware

Accessing the MRML scene and nodes

- Using utility functions – in slicer.util

```
v = getNode('MRHead')
```

OR

```
v = getNode('MR*')
```

getNode: somewhat ambiguous, recommended for testing & debugging only

- Accessing MRML scene directly

```
v=slicer.mrmlScene.GetFirstNodeByName('MRHead')
```

OR

```
v=slicer.mrmlScene.GetFirstNodeByClass('vtkMRMLScalarVolumeNode')
```

Kitware

slicer.modules.<modulename>

- Attributes of “modules” correspond to loaded Slicer modules

Exercise 2a: Select a module

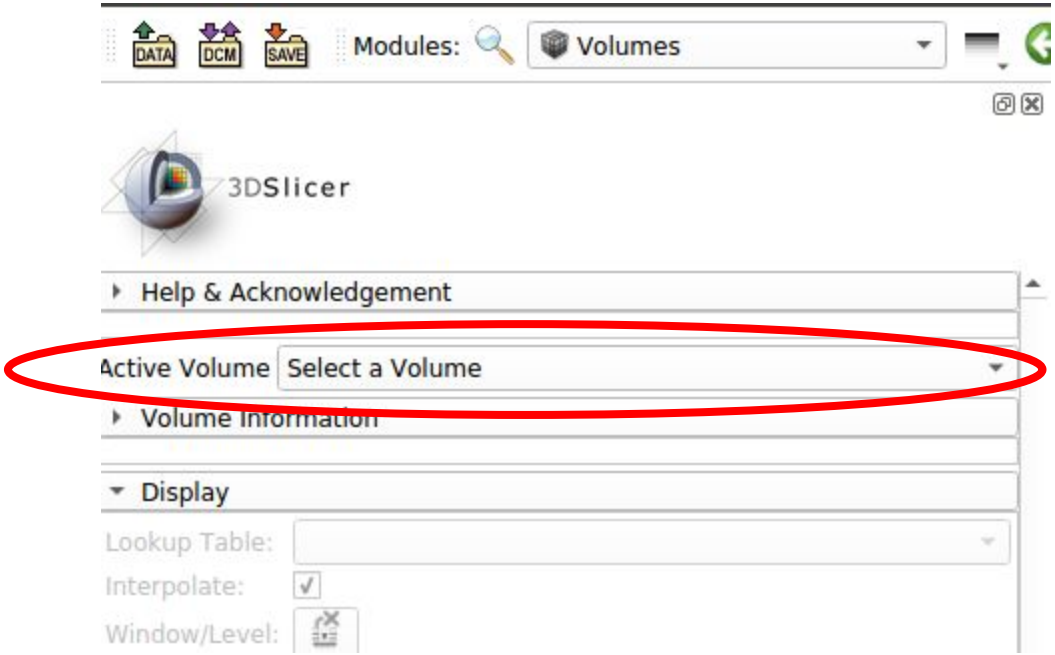
- Hint: use `slicer.util.selectModule`

Exercise 2b: Then select a node

- Hint: use `slicer.util.openNodeModule`

Kitware

Step-by-step: Select module and node using Python



Kitware

Approach 1: Open the best module for a given node

```
>>> slicer.app.openNodeModule(getNode("MRHead"))
```

Kitware

Analysis of “Approach 1”

- Modules are associated with node types.

- Application is in charge of
  - finding the modules supporting the given node
  - identifying the most suitable module
  - calling `setEditedNode()` on the module representation
  - selecting the module

Approach 2: Select module -> set current node

```
>>> selectModule(slicer.modules.volumes)

>>> volumesWidget = getModuleGui(slicer.modules.volumes)

>>> nodeSelector = findChild(volumesWidget, "ActiveVolumeNodeSelector")

>>> nodeSelector.setCurrentNode(getNode("MRHead"))
```

Kitware

What is the “ActiveVolumeNodeSelector” string ?

- Object name
  - Defined in UI file (more on this later)
  - Explicitly set after instantiating object
- In this particular case
  - See qSlicerVolumesModuleWidget.ui

UI File

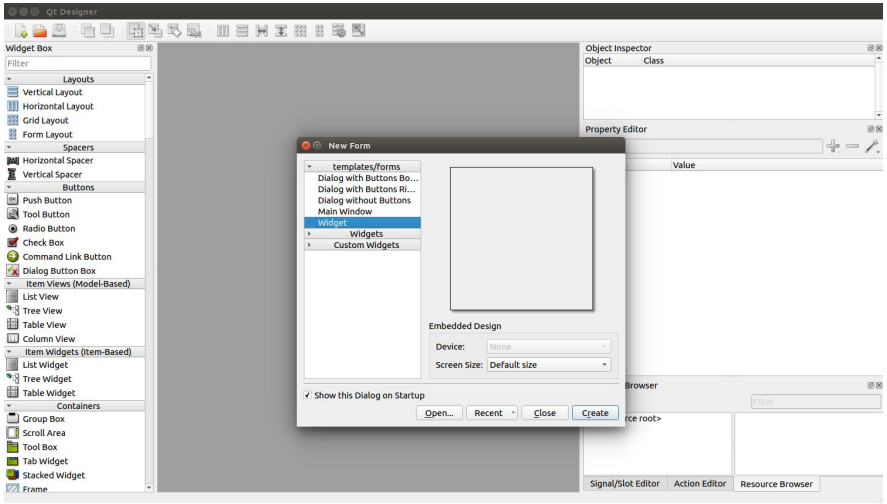
- Represent the widget tree of the “form” in XML format
- Crafted using Qt Designer
- Two integration cases:
  - Compile Time Form Processing
  - Run Time Form Processing

- References
  - https://www.slicer.org/wiki/Documentation/Nightly/Developers/Tutorials/WidgetWriting
  - http://doc.qt.io/qt-5/designer-using-a-ui-file.html

Kitware

Qt Designer

- Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets



Slicer Managers

- Extension manager
- IO manager
- Layout manager
- Module manager

Following slides illustrate how to access these managers.

Extensions Manager: List installed extensions

```
>>> slicer.app.extensionsManagerModel().installedExtensions
(u'MarkupsToModel', u'SegmentEditorExtraEffects')
```

Kitware

IO Manager

```
>>> slicer.app.ioManager().openAddDataDialog()
```

Convenience IO methods provided by `slicer.util`:

```
>>> help(slicer.util.loadModel)
Help on function loadModel in module slicer.util:

loadModel(filename, returnNode=False)

>>> help(slicer.util.openAddModelDialog)
Help on function openAddModelDialog in module slicer.util:

openAddModelDialog()
```

Kitware

Layout Manager: Switch layout

```
>>> slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.SlicerLayoutDual3DView)
```

Kitware

Module Manager: Get module by name

```
>>> slicer.app.moduleManager().module("Volumes")
qSlicerVolumesModule (qSlicerVolumesModule at: 0x4feefd0)
```

Convenience module methods provided by `slicer.util`:

```
>>> slicer.util.getModule("Volumes")
qSlicerVolumesModule (qSlicerVolumesModule at: 0x4feefd0)
```

Kitware

Qt connection

```
>>> def displayHello():
...   print("Hello")
...
>>> button = qt.QPushButton("Hello")
>>> button.show()
>>> button.connect("clicked()", displayHello)
True
>>>
Hello
>>> button.click()
Hello
```

- References:
  - http://doc.qt.io/qt-5/object.html
  - http://doc.qt.io/qt-5/signalsandslots.html
  - http://doc.qt.io/qt-5/qpushbutton.html

Kitware

Exercise 3: Display name of selected module

Hints:

- `mainWindow().moduleSelector()`
- Signal is `"moduleSelected(QString)"`
- http://apidocs.slicer.org/master/classqSlicerModuleSelectorToolBar.html

Kitware

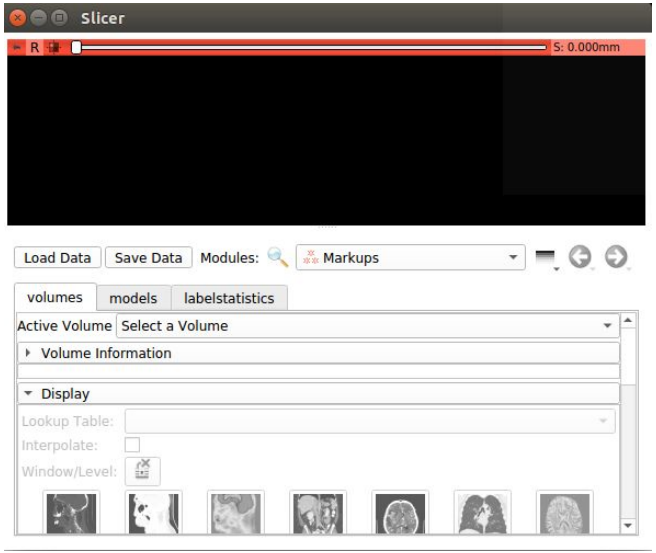
Solution “Exercise 3”

```
>>> def onModuleSelected(moduleName):
...   print("Module '%s' was selected" % moduleName)
...
>>> mainWindow().moduleSelector().connect("moduleSelected(QString)", onModuleSelected)
True
>>> selectModule("Volumes")
Module 'Volumes' was selected
```

Kitware

Exercise 4: Add missing connection

- Download exercise4.py
- Execute
  - Copy/Paste its content in the interactor
  - Or use CTRL-R to run the script
- Simple UI with:
  - 3D view
  - button to load data
  - tab widget
  - module selector allowing to add any module to the tab widget
- **Exercise**: Add the missing connection
  - Ensure that selected module are added to the tab widget



Kitware

Solution “Exercise 4”

```
>>> moduleSelector.connect('moduleSelected(QString)', onModuleSelected)
True
```

Styling Widgets using “CSS” #1

```
editor = qt.QWidget()
layout = qt.QVBoxLayout()
editor.setLayout(layout)

content = qt.QTextEdit()
content.plainText = slicer.app.styleShe
layout.addWidget(content)

apply = qt.QPushButton("Apply")
layout.addWidget(apply)

def applyStyleSheet():
  mainWidget.setStyleSheet(content.plainText)
  editor.setStyleSheet(content.plainText)

apply.clicked.connect(applyStyleSheet)
editor.show()
```

Kitware

Styling Widgets using “CSS” #2

- https://doc.qt.io/archives/qt-5.10/stylesheet-syntax.html

```
QPushButton {
        border: 1px solid #60A7E5;
        border-radius: 3px;
        background-color: qlineargradient(x1: 0, y1: 1, x2: 0, y2: 0, stop: 0 rgba(108,185,252,1), stop: 1 rgba(96,167,229,1));
        min-width: 80px;
        min-height: 24px;
        color: white;
}


QPushButton:hover {
        background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 rgba(118,180,244,1), stop: 1 rgba(144,203,255,1));
}


QPushButton:pressed {
        background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 rgba(108,185,252,1), stop: 1 rgba(96,167,229,1));
}
QPushButton:disabled {
        background-color: #CDCDCD;
        border: 1px solid #999999;
        color: #777777;
}
```

Slicer --no-main-window/--python-* options

```
--python-script     Python script to execute after slicer loads.
--python-code       Python code to execute after slicer loads.
[...]
--no-main-window    Disable display of the main slicer window.
                    Use with --python-script for alternate interface
```

Kitware

Exercise 5: Start Slicer showing only the SimpleUI

- Simple UI corresponds to the example of “exercise 4”

Solution “Exercise 5”

```
Slicer --no-main-window --python-script exercise4.py
```

Exercise 6: Increase MRHead contrast using NumPy

Hints:

- `slicer.util.arrayFromVolume`
- `slicer.util.arrayFromVolumeModified`

Kitware

Solution “Exercise 6”

```
>>> volumeNode = getNode('MRHead')
>>> a = arrayFromVolume(volumeNode)
>>> # Increase image contrast
>>> a[:] = a * 2.0
>>> arrayFromVolumeModified(volumeNode)
```

References:

- Accessing Volume data as numpy array

Kitware

Source: PerkLab Bootcamp

Setting window/level values from python

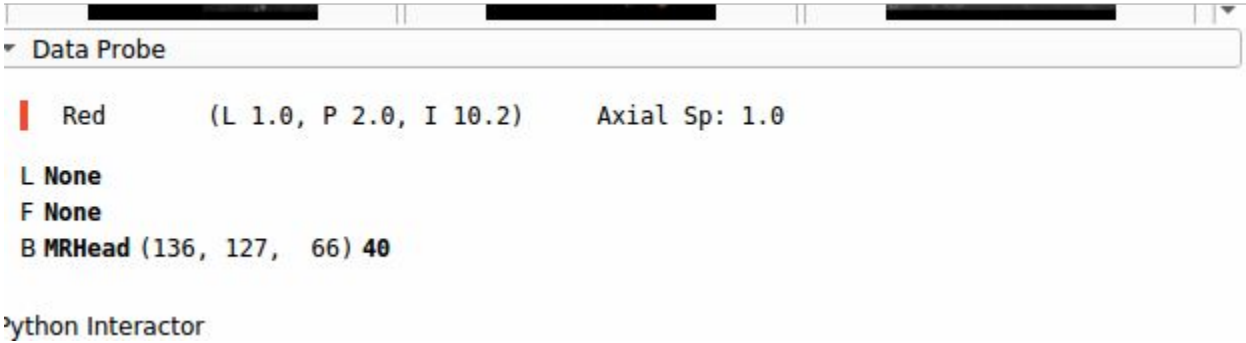
```
>>> v = getNode('MRHead')
>>> vd = v.GetDisplayNode()
>>> vd.SetAutoWindowLevel(0)
>>> vd.SetWindowLevel(350,40)
```

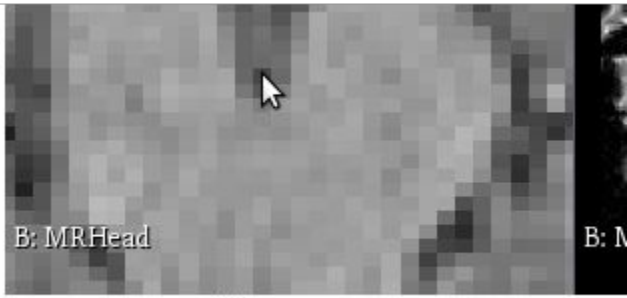
Kitware

Source: PerkLab Bootcamp

Exercise 7: Accessing voxel value of NumPy array

- Load MRHead dataset
- Get voxel value at (136,127,66) IJK position





Kitware

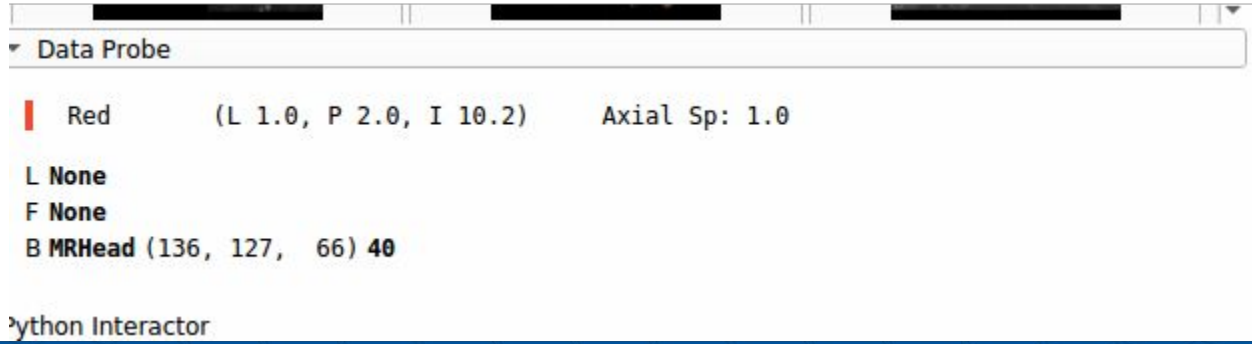
Source: PerkLab Bootcamp

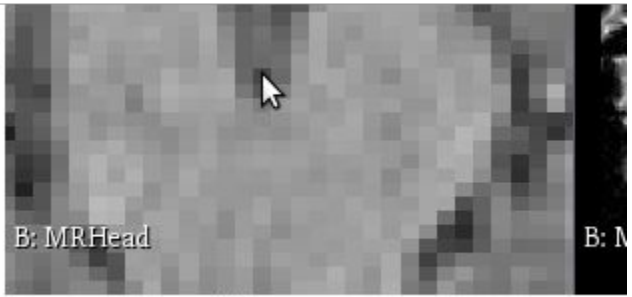
Solution “Exercise 7”

- Get voxel value at (136,127,66) IJK position:

```
>>> v = getNode('MRHead')
>>> va = slicer.util.arrayFromVolume(v)
>>> va[66,127,136]
40
```

- IJK in Slicer, KJI in NumPy (C-order indexing)

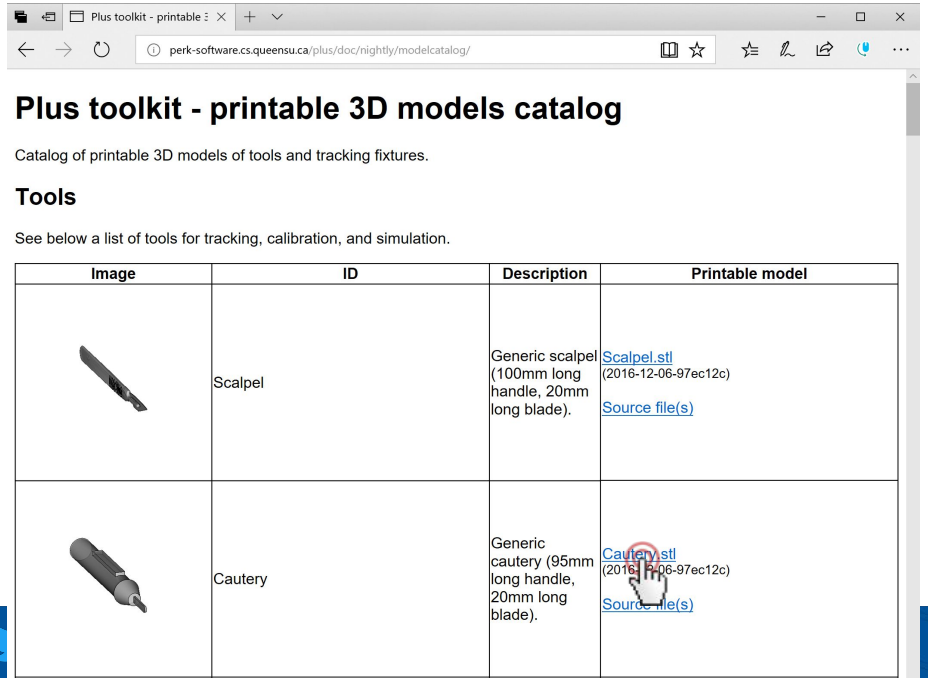




Source: PerkLab Bootcamp

Exercise 8 preparation: Load Model

- Go to http://perk-software.cs.queensu.ca/plus/doc/nightly/modelcatalog/
- Download and load “Cautery.stl”



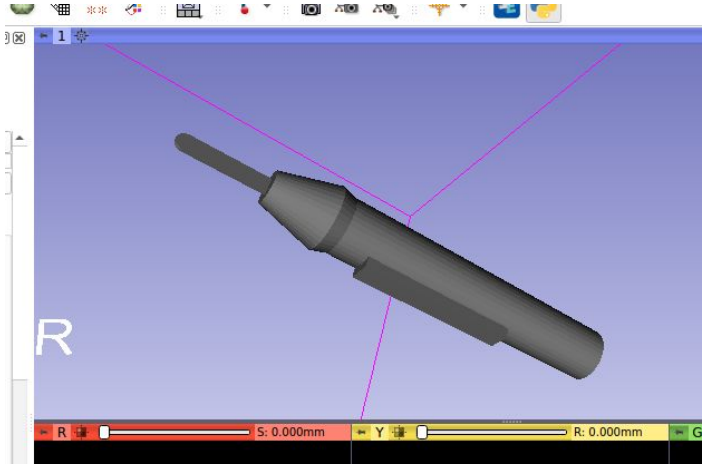
Plus toolkit - printable 3D models catalog

Catalog of printable 3D models of tools and tracking fixtures.

Tools

See below a list of tools for tracking, calibration, and simulation.

Image	ID	Description	Printable model
	Scalpel	Generic scalpel (100mm long handle, 20mm long blade).	Scalpel.stl (2016-12-06-97ec12c) Source file(s)
	Cautery	Generic cautery (95mm long handle, 20mm long blade).	Cautery.stl (2016-12-06-97ec12c) Source file(s)



Kitware

Source: PerkLab Bootcamp

Exercise 8: Setting model color programmatically

Hints:

- `slicer.vtkMRMLModelNode.GetDisplayNode`
- `slicer.vtkMRMLModelDisplayNode.SetColor`

Kitware

Source: PerkLab Bootcamp

Solution “Exercise 8”

```
>>> c = getNode('Cautery')
>>> cd = c.GetDisplayNode()
>>> cd.SetColor(1,0,0)
```

Kitware

Source: PerkLab Bootcamp

Exercise 9: Change model to a sphere

Hints:

- `vtk.vtkSphereSource`
- `vtk.vtkSphereSource.SetRadius`
- `vtk.vtkSphereSource.SetCenter`
- `slicer.vtkMRMLModelNode.SetAndObservePolyData`

Kitware

Source: PerkLab Bootcamp

Solution “Exercise 9”

```
>>> s = vtk.vtkSphereSource()
>>> s.SetRadius(30)
>>> s.SetCenter(30,40,60)
>>> s.Update()
>>> c.SetAndObservePolyData(s.GetOutput())
```

Kitware

VTK connection

```
>>> object = vtk.vtkObject()
>>> def onObjectModified(caller, eventId):
...   print("className '%s' event '%s'" % (caller.GetClassName(), eventId))
...
>>> object.AddObserver(vtk.vtkCommand.ModifiedEvent, onObjectModified)
1
>>> object.Modified()
className 'vtkObject' event 'ModifiedEvent'
```

Problem: We need to keep track of observation tags to unobserve …

```
>>> obsTag = object.AddObserver(vtk.vtkCommand.ModifiedEvent, onObjectModified)
>>> f.RemoveObserver(obsTag)
```

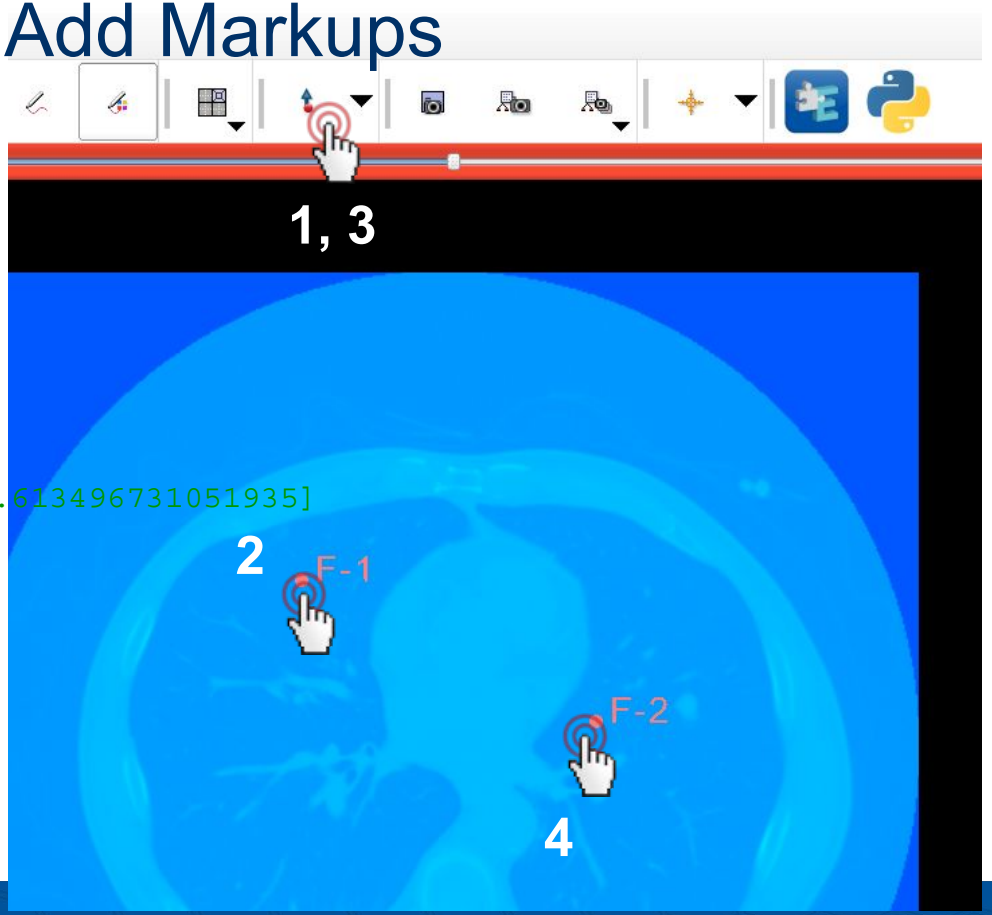
Kitware

Source: PerkLab Bootcamp

Exercise 10 Preparation: Add Markups

- Create 2 markup points
- Get markup position

```
>>> f = getNode('F')
>>> pos=[0,0,0]
>>> f.GetNthFiducialPosition(0,pos)
>>> pos
[-4.118395381158081, 4.581543577439959, -27.613496731051935]
```



Source: PerkLab Bootcamp

Exercise 10: Print first fiducial position when moved

Hints:

- Implement function **`printFiducialPosition`**

```
def printFiducialPosition(caller=None, event=None):
  # Add code displaying position of first fiducial
```

- Use `vtk.vtkCommand.ModifiedEvent`

- `Use slicer.vtkMRMLMarkupsFiducialNode.AddObserver`

Kitware

Source: PerkLab Bootcamp

Solution “Exercise 10”

```
>>> f = getNode('F')

>>> def printFiducialPosition(caller=None, event=None):
... f = getNode('F')
... pos=[0,0,0]
... f.GetNthFiducialPosition(0,pos)
... print(pos)
...

>>> obsTag = f.AddObserver(vtk.vtkCommand.ModifiedEvent, printFiducialPosition)
```

-> Try to move fiducial 1

```
>>> obsTag = f.RemoveObserver(obsTag)
```

-> Try again to move fiducial 1

Kitware

VTK connection: slicer.util.VTKObservationMixin

- Keep track of observation tags
- Manage observations (remove, add, find, …)
  - See How can I access callData argument in a VTK object observer callback function

```
from slicer.util import VTKObservationMixin

class MyClass(VTKObservationMixin):
  def __init__(self):
    VTKObservationMixin.__init__(self)
    self.addObserver(slicer.mrmlScene, slicer.vtkMRMLScene.NodeAddedEvent,
self.nodeAddedCallback)

  @vtk.calldata_type(vtk.VTK_OBJECT)
  def nodeAddedCallback(self, caller, eventId, callData):
    print("Node added")
    print("New node: {0}".format(callData.GetName()))

myObject = MyClass()
```

Kitware

Slicer Python Interactor vs Slicer Python Interpreter

- Slicer Python Interactor
  - “Slicer” sets environment and **launches** “SlicerApp-real”
  - “SlicerApp-real.exe” executable depends and uses the “python” library
- Slicer Python Interpreter
  - “PythonSlicer” sets environment and **launches** “python-real”
  - “SlicerApp-real” executable depends and uses on “python” library

Kitware

The “launcher”

- Lightweight open-source utility
- Allow to set environment before starting a real application.
- Configured using settings file
- Binaries available for macOS, Linux and Windows
- Statically built against Qt

References:

- https://www.slicer.org/wiki/Documentation/Nightly/Developers/Launcher
- https://github.com/commontk/AppLauncher#readme

Kitware

Launcher Configuration

- Manually crafted or automatically generated
- Settings (*.ini files)
  - By convention, <executableName>LauncherSettings.ini
  - Can exist along side, in bin or lib sub-directory
- Slicer Launcher Settings files
  - Slicer
    - SlicerLauncherSettings.ini
    - SlicerLauncherSettingsToInstall.ini
  - SlicerPython
    - SlicerPythonLauncherSettings.ini
    - SlicerPythonLauncherSettingsToInstall.ini

Kitware

More Examples

- Script repository
  https://www.slicer.org/wiki/Documentation/Nightly/ScriptRepository

- Slicer sources
  https://github.com/Slicer/Slicer/tree/master/Modules/Scripted
  https://github.com/Slicer/Slicer/tree/master/Applications/SlicerApp/Testing/Python

- Slicer extension sources
  https://github.com/Slicer/ExtensionsIndex

Kitware

Useful Documentation Links

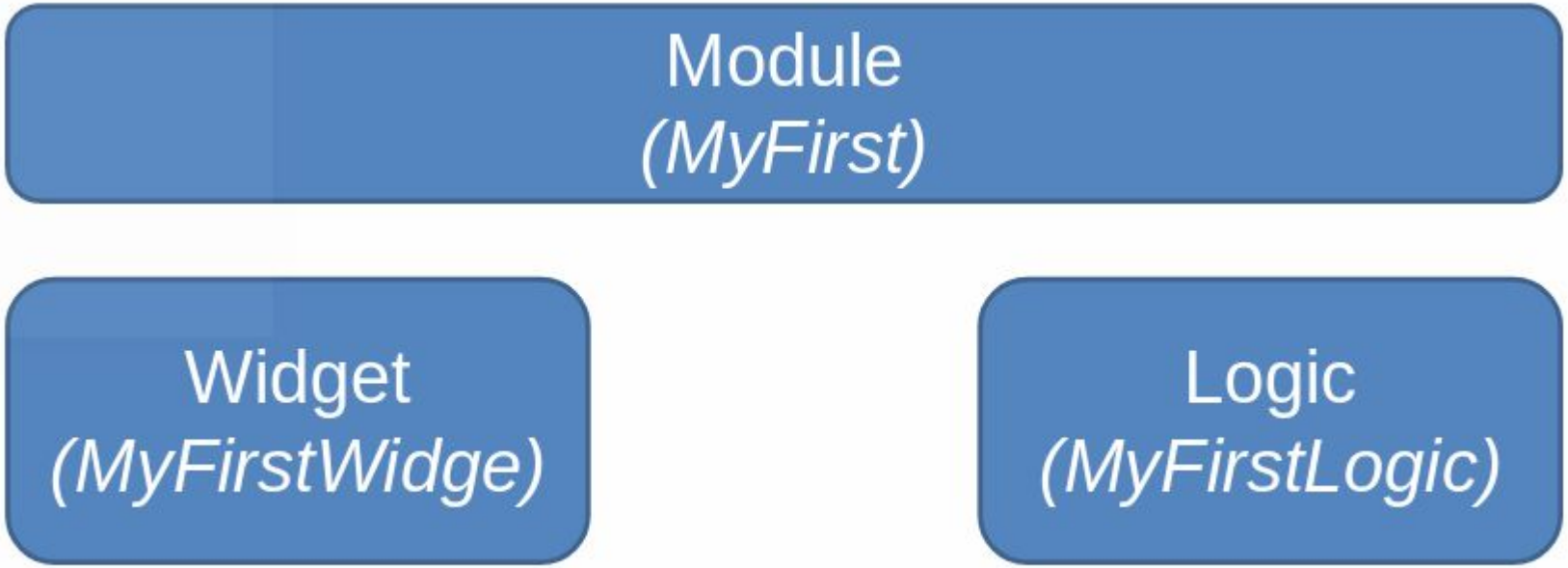
- https://www.slicer.org/wiki/Documentation/Nightly/Developers/Python_scripting
- https://www.slicer.org/wiki/Documentation/Nightly/Developers/FAQ/Python_Scripting
- http://apidocs.slicer.org
- https://doc.qt.io/archives/qt-5.10/classes.html
- And also Python, VTK, CTK, … API documentation

Kitware

Slicer Scripted Module

Source: PerkLab Bootcamp

Scripted module implementation



Kitware

Source: PerkLab Bootcamp

Module class

- Required. Only one global instance exists:

  module = slicer.modules.volumes

- Stores module name, description, icon, etc.
- Creates and holds a reference to logic and widget:
  - Loadable modules:

    ```
    widget = module.widgetRepresentation()
    logic = module.logic()
    ```

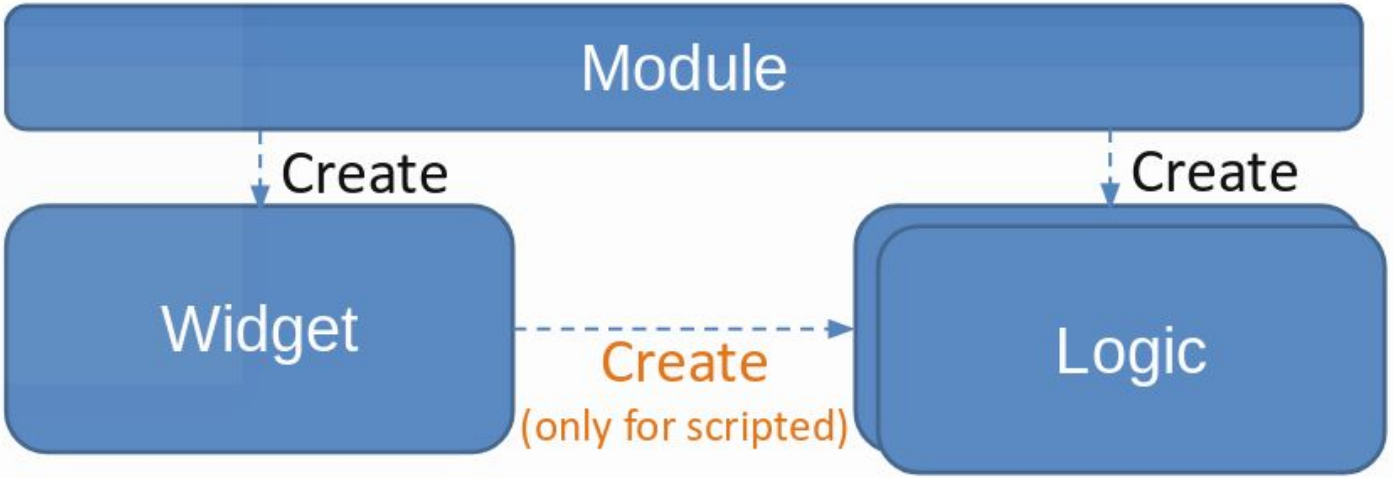
  - Python scripted modules:

    ```
    widget = module.widgetRepresentation().self()
    logic = widget.logic
    ```

Kitware

Source: PerkLab Bootcamp

Scripted module implementation



Scripted module logic is not created automatically, it has to be instantiated in the Widget class.

Kitware

Source: PerkLab Bootcamp

Widget Class (1 / 2)

- Needed if the module has a user interface
- Typically only one global instance exists
- Defines the module's user interface
- Keeps user interface and nodes in sync (observes MRML nodes to get change notifications)
- Launches processing methods implemented in the logic class

Kitware

Source: PerkLab Bootcamp

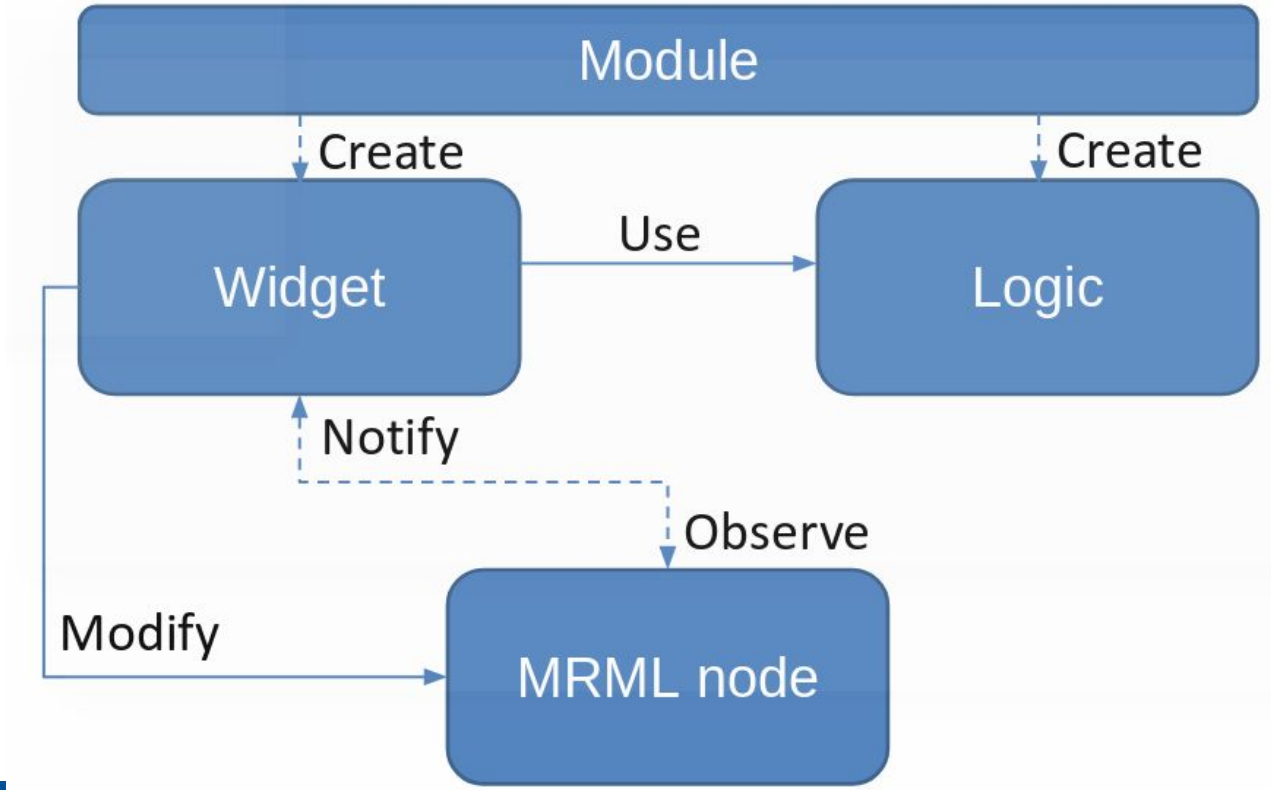
Widget Class (2 / 2)

- Include a parameter node selector at the top (or use a singleton parameter node)
- If a parameter node is selected then add an observer to its modified events; if modified then call widget's `updateGUIFromParameterNode()`
- If a parameter is changed in the GUI then update MRML node

Kitware

Source: PerkLab Bootcamp

Scripted module implementation



Kitware

Source: PerkLab Bootcamp

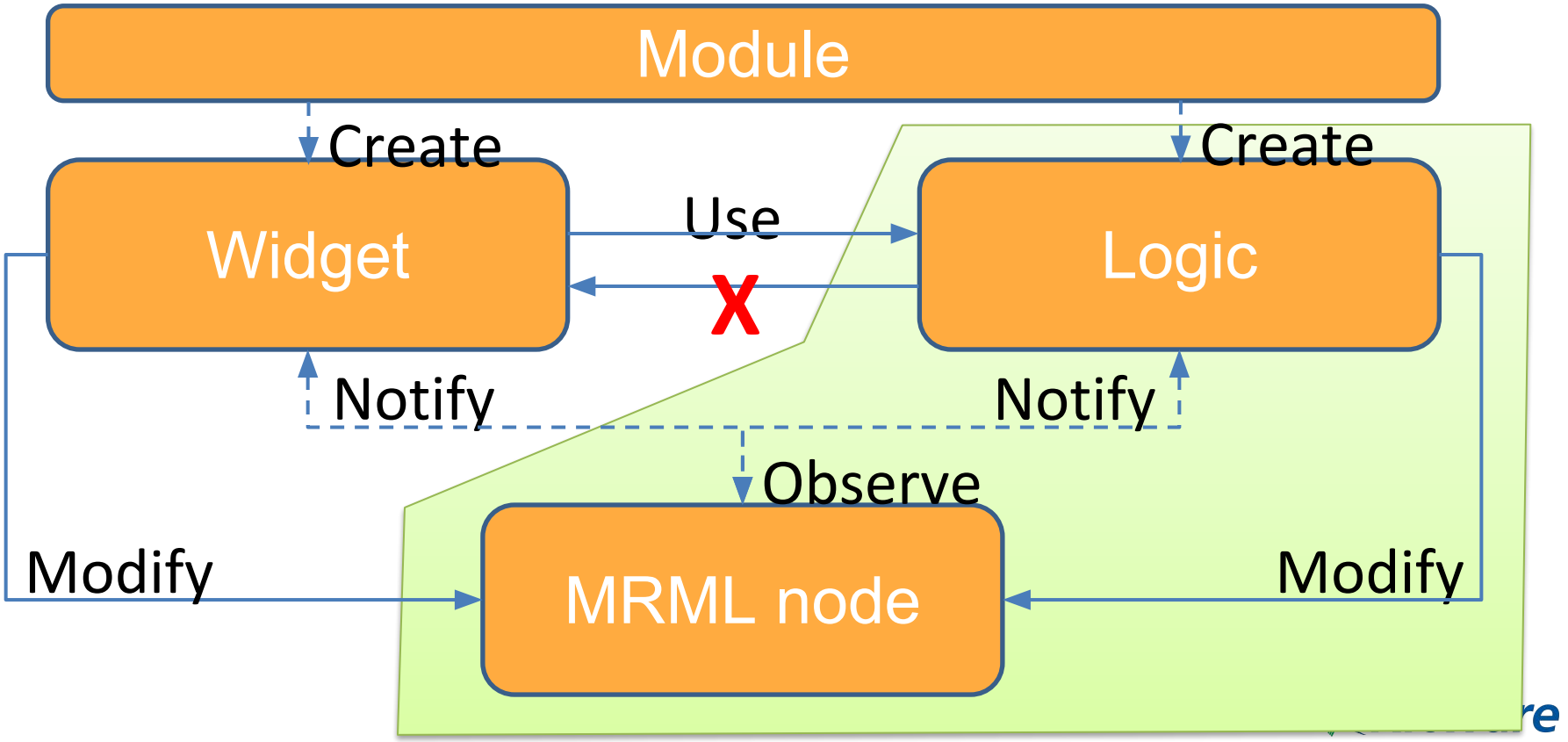
Logic Class

- Needed if the module does any processing (always?)
- The module must be usable from another module, just by calling logic methods
- Must not rely on the Widget class: the module must be usable without even having a widget class
- Logic may be instantiated many times (to access utility functions inside)
- Logic may observe nodes: only if real-time background processing is needed (e.g., we observe some input nodes and update other nodes if input nodes are changed)

Kitware

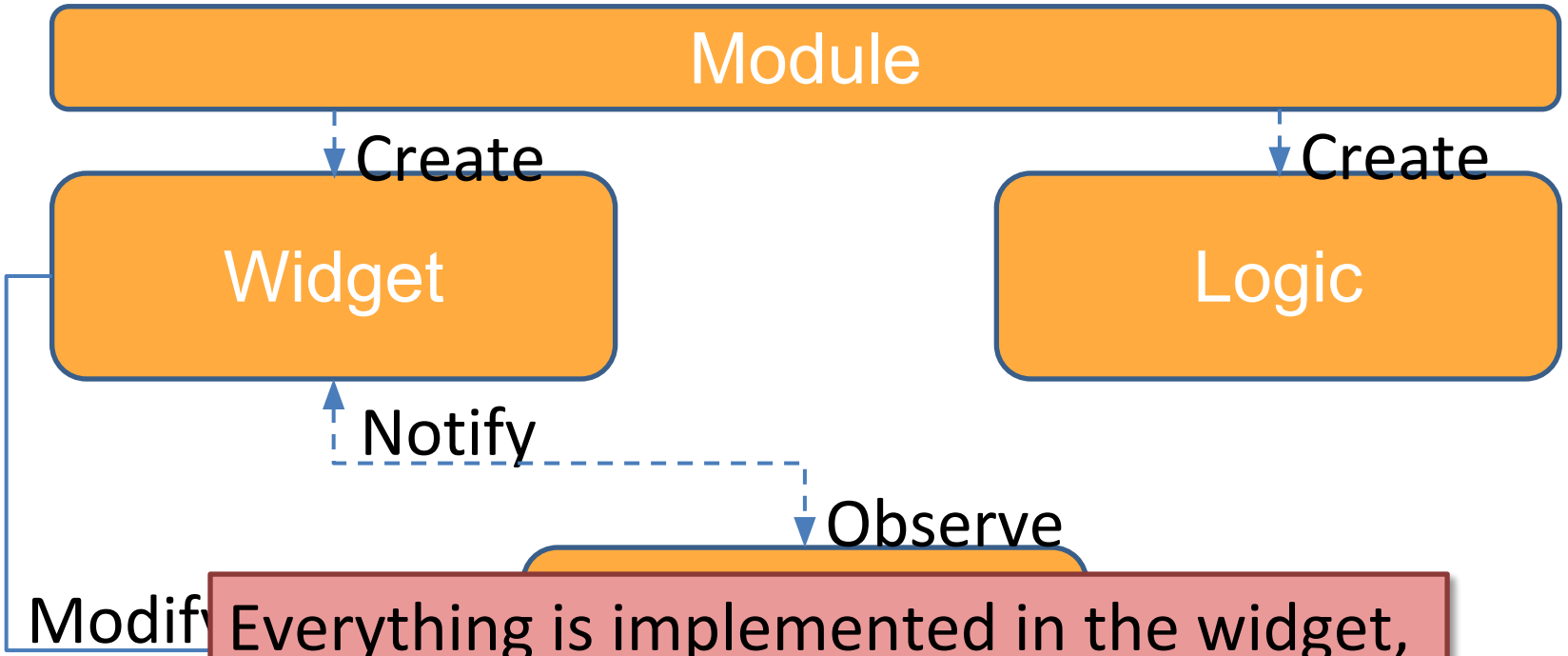
Scripted module implementation

Source: PerkLab Bootcamp



Common Mistake 1

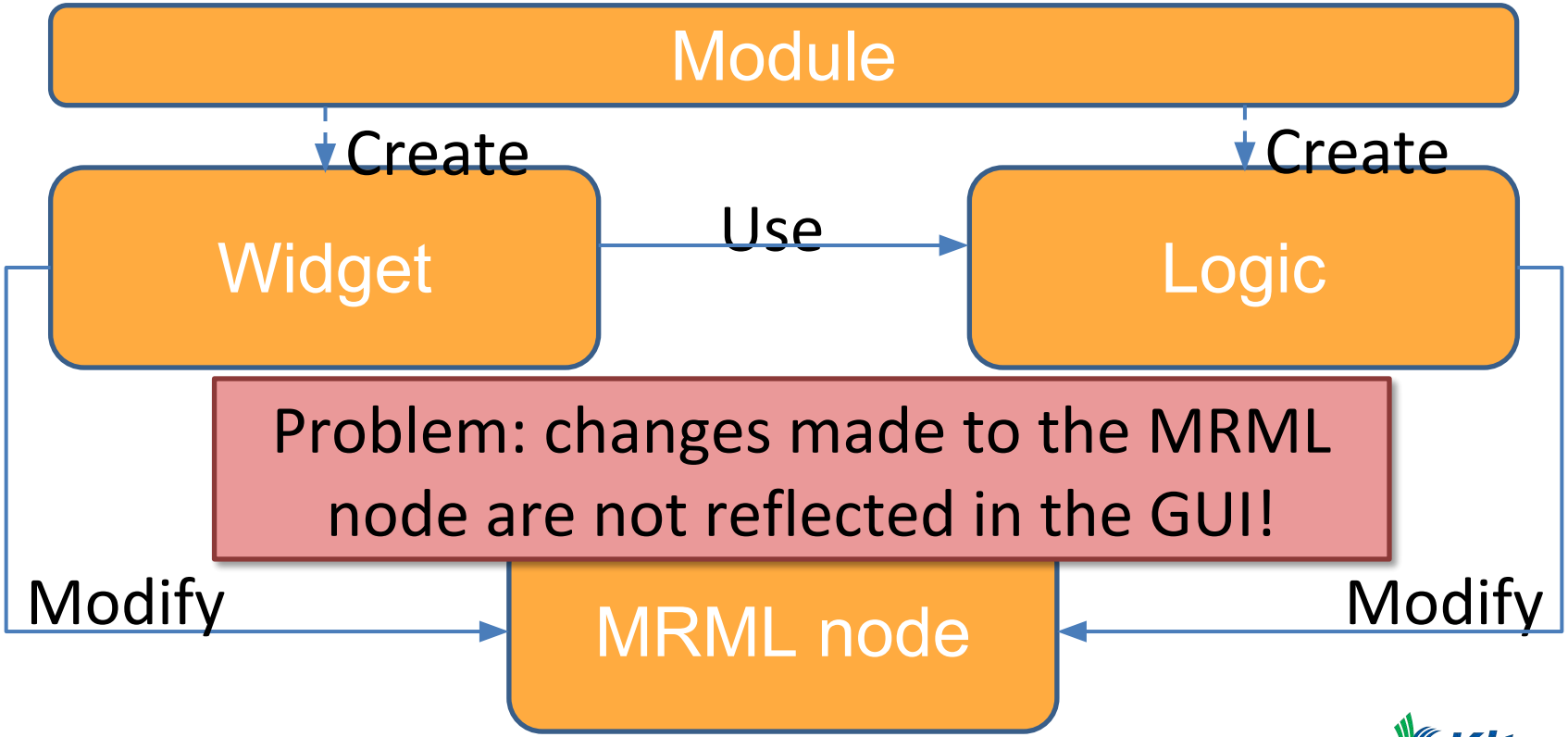
Source: PerkLab Bootcamp



Everything is implemented in the widget, therefore the module is not usable from another module or with a custom GUI!

Common Mistake 2

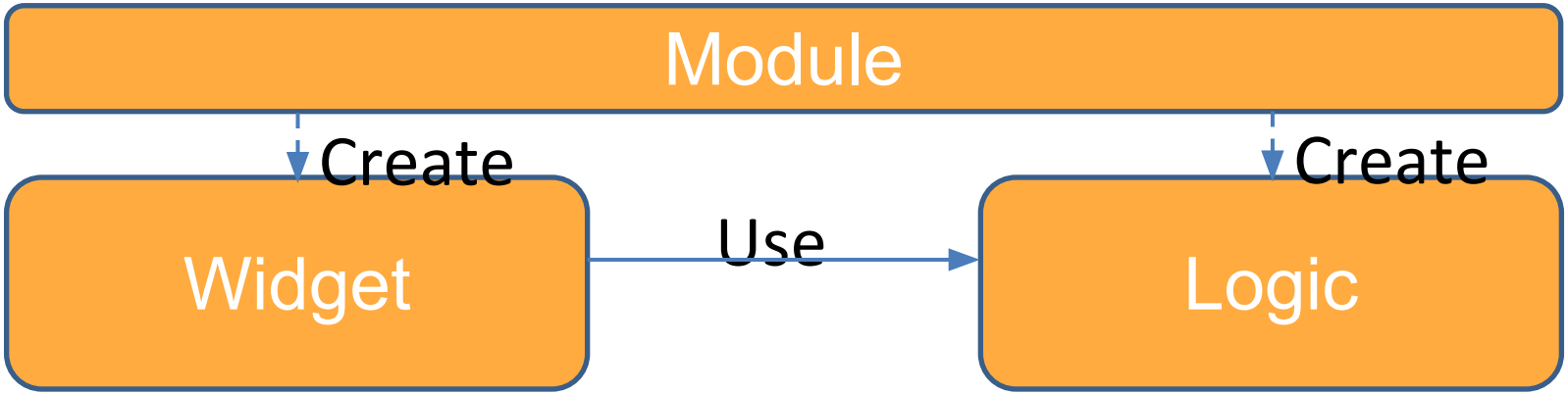
Source: PerkLab Bootcamp



Kitware

Common Mistake 3

Source: PerkLab Bootcamp

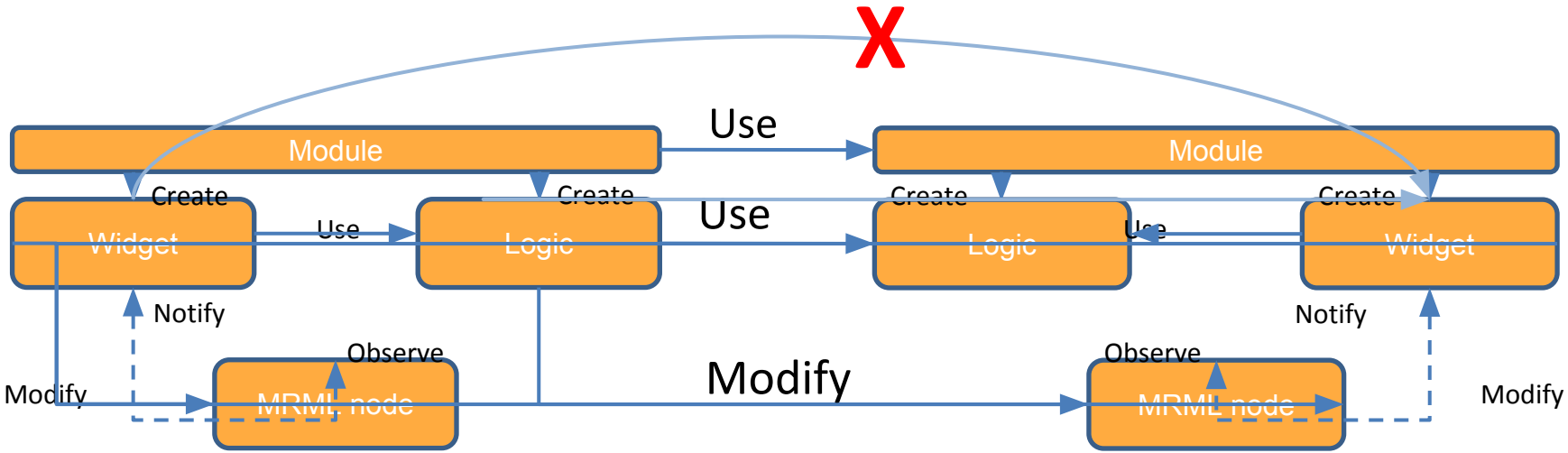


No parameter node is used. When the scene is saved and reloaded all the settings in the module user interface are lost!

Kitware

Communication between modules

Source: PerkLab Bootcamp



- Module logic may modify any MRML nodes – most common form of communication
- Module logic class may use another module's logic class
- Module class may use another module class (e.g., to access module logic and pass it to its own logic)

Creating Slicer Extensions

Extension vs Module

Extension: “package” for delivering new functionality to the user via Extension Manager.

Extension can contain one or more **modules**. After installing an extension, the associated modules will be presented to the user as built-in ones.

http://wiki.slicer.org/slicerWiki/index.php/Documentation/Nightly/Developers/FAQ#What_is_an_extension_.3F

Kitware

Example: SlicerRT

SlicerRT extension packaging Slicer modules for radiotherapy applications

See http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/SlicerRT

Extension Description

- SlicerRT is one of the themes of the SparKit (Software Platform and Adaptive Radiotherapy Kit) project with the goal of making 3D Slicer a powerful radiotherapy research platform.
  SparKit is a project funded by Cancer Care Ontarioand the Ontario Consortium for Adaptive Interventions in Radiation Oncology (OCAIRO) to provide free, open-source toolset for radiotherapy and related image-guided interventions.
- The SlicerRT extension incorporates Plastimatch modules and algorithms.
- Additional information for users can be found on the User's guide page

Modules

- DICOM-RT import
- DICOM-RT export
- Contours
- Dose volume histogram
- Dose accumulation
- Dose comparison
- Isodose line and surface display
- Contour comparison
- Contour morphology

Kitware

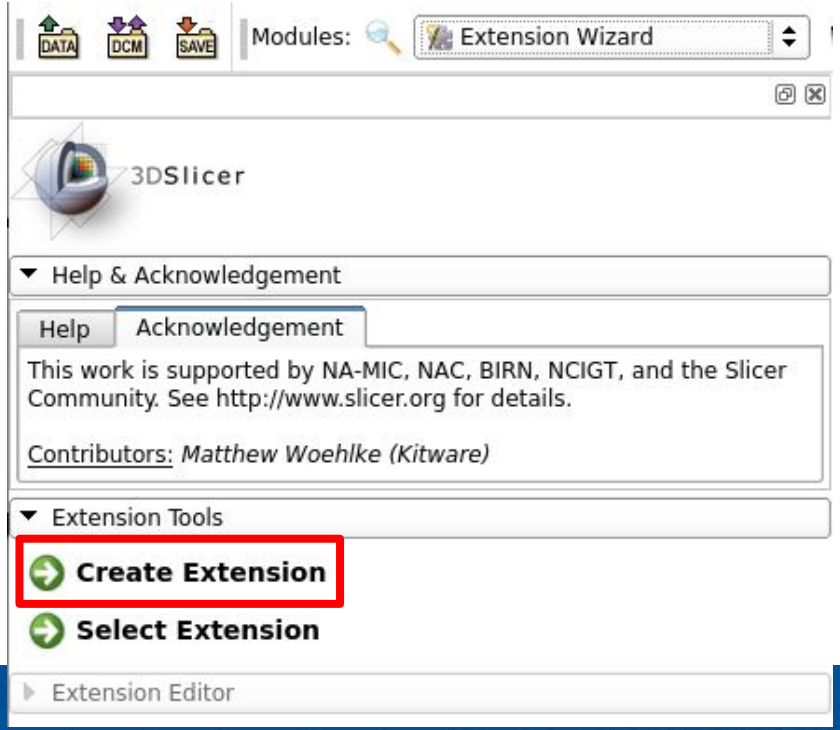
Module Type Recommendations

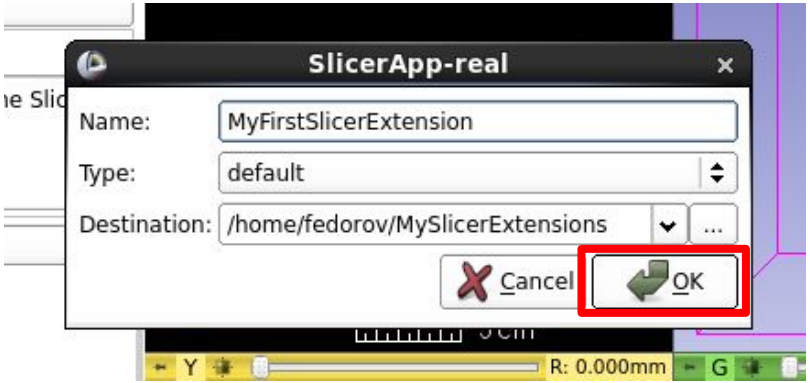
- Command Line Interface (CLI, C++):
  - standalone executable with limited input/output arguments
- Command Line Interface (CLI, Python):
  - Support added in April 2018. (See PyCLIModule4Test.py and PyCLIModule4Test.xml)
- Scripted Loadable Modules (Python):
  - recommended for fast prototyping
- Loadable Modules (C++)
  - optimized for heaving computation
-

Kitware

Slicer Extension Wizard: Create boilerplate

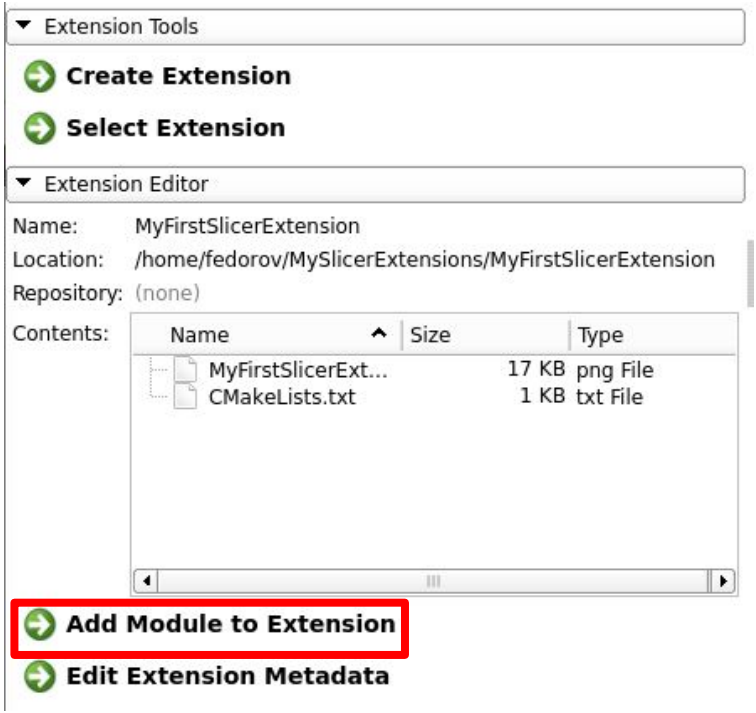
- Modules -> Extension Wizard

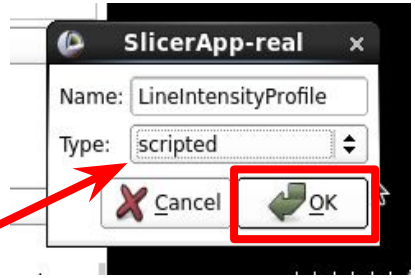


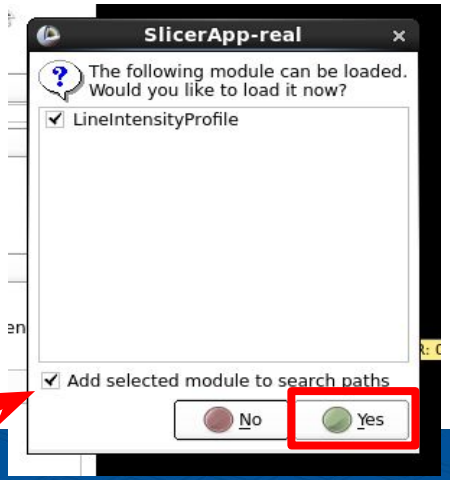


Kitware

Slicer Extension Wizard: Add module







Kitware

Reload and Test

- Python script can be edited in place and reloaded
- Built-in testing infrastructure
  - See https://www.slicer.org/wiki/Documentation/Nightly/Developers/Tutorials/SelfTestModule

Kitware

Exercise 11: Using the Extension Wizard

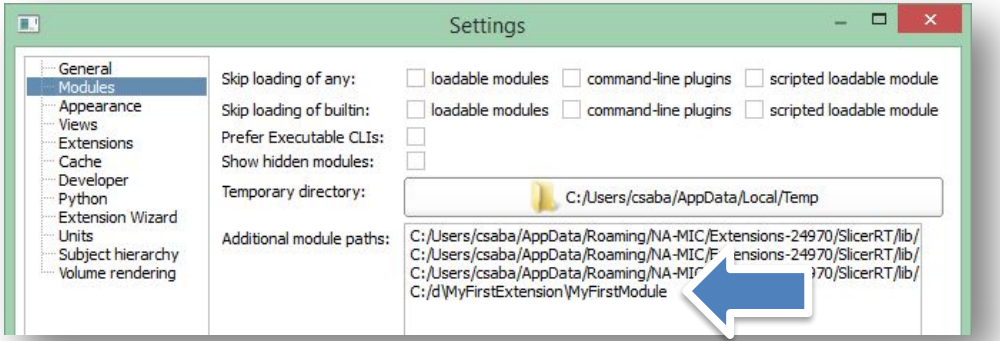
- Create an extension
- Add a scripted module
- Edit file
- Reload

Kitware

Source: PerkLab Bootcamp

Module Paths

- In Application Settings

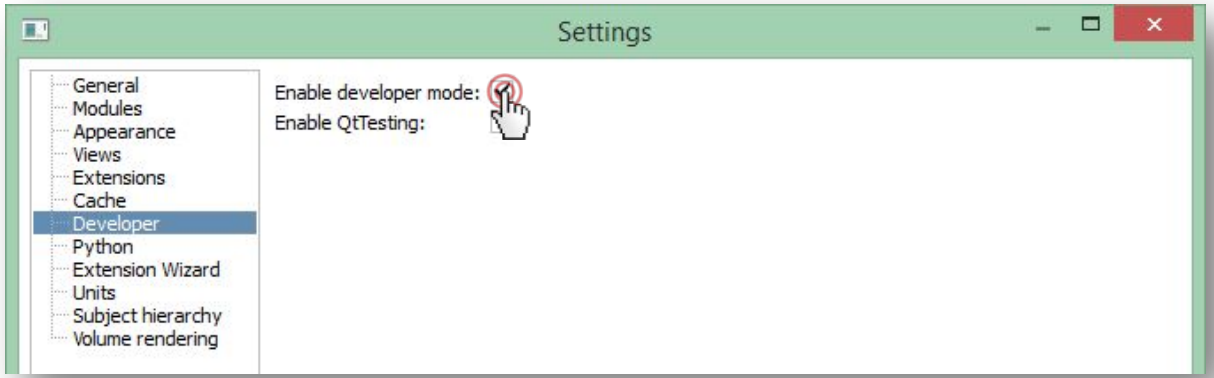


- You can add by drag&drop
  - But you don't need to because checking the "Add selected module to search paths" checkbox did it for you

Source: PerkLab Bootcamp

Enabling Developer Mode

- Enable features useful for development
  - Allow dynamic reload of source code etc.
- In Application Settings

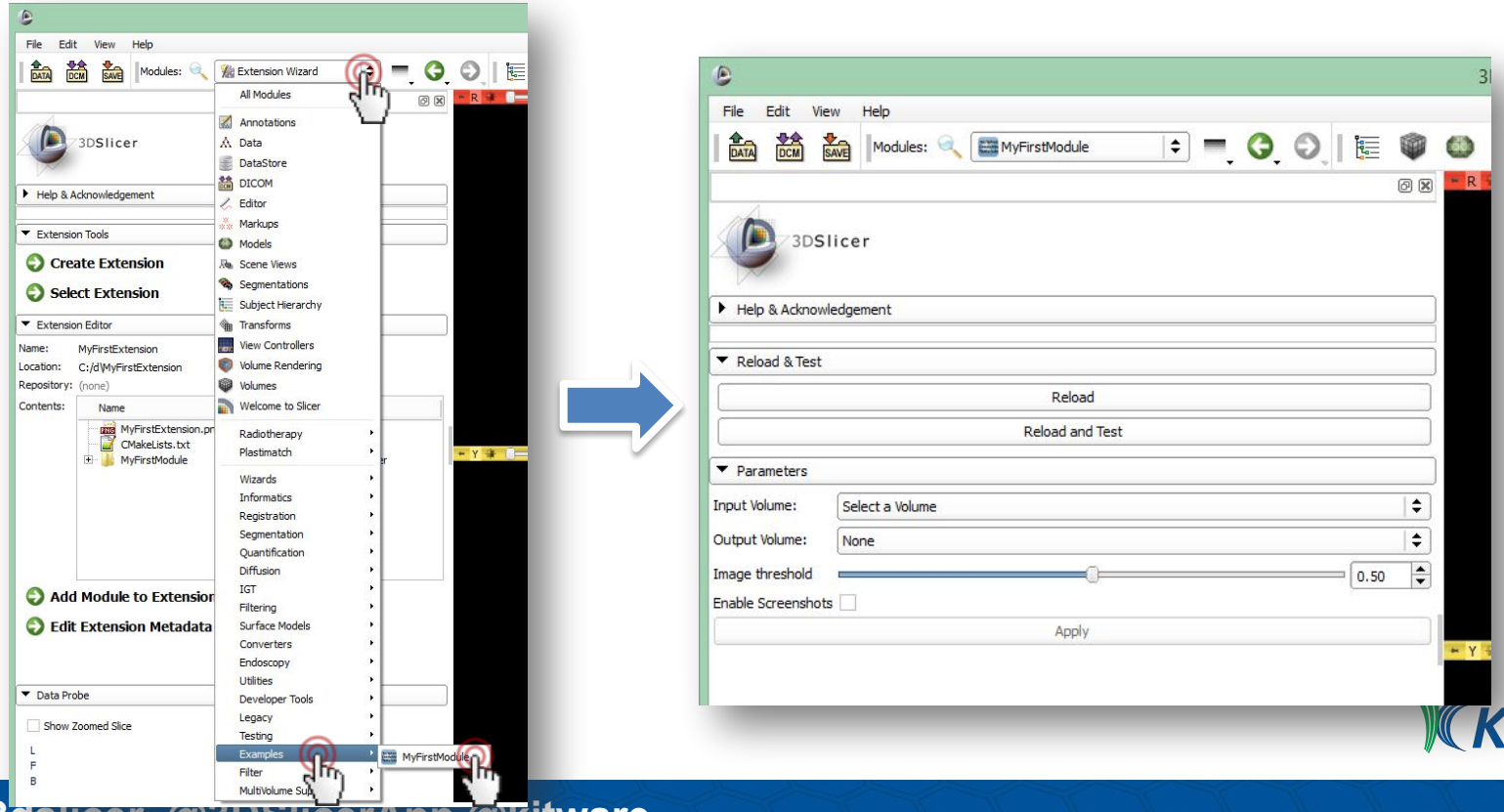


- Restart Slicer

Kitware

Source: PerkLab Bootcamp

Finding the new module in Slicer



Kitware

Source: PerkLab Bootcamp

Writing our scripted module #1

- Open *MyFirstExtension\MyFirstModule\MyFirstModule.py*
- Rename the module

```
def __init__(self, parent):
  ScriptedLoadableModule.__init__(self, parent)
  self.parent.title = "Center of Mass"
```

- Create widgets to specify inputs:
  Under **def setup(self)** look for **input volume selector**, change

```
self.inputSelector.nodeTypes = ["vtkMRMLMarkupsFiducialNode"]
self.inputSelector.selectNodeUponCreation = False
```

- and

```
parametersFormLayout.addRow("Input Markups: ", self.inputSelector)
```

Kitware

Source: PerkLab Bootcamp

Writing our scripted module #2

- Look for **class MyFirstModuleLogic** towards the bottom
- Insert this code above the **Run** function

```
def getCenterOfMass(self, markupsNode):
    centerOfMass = [0,0,0]

    import numpy as np
    sumPos = np.zeros(3)
    for i in range(markupsNode.GetNumberOfMarkups()):
      pos = np.zeros(3)
      markupsNode.GetNthFiducialPosition(i, pos)
      sumPos += pos

    centerOfMass = sumPos / markupsNode.GetNumberOfMarkups()

    logging.info('Center of mass for \'' + markupsNode.GetName() + '\': ' + repr(centerOfMass))

    return centerOfMass
```

Kitware

Writing our scripted module #3

- Change the **Run** function to look like this:

```
def run(self, inputMarkups, outputVolume, imageThreshold, enableScreenshots=0):
    """
    Run the actual algorithm
    """
    self.centerOfMass = self.getCenterOfMass(inputMarkups)

    return True
```

Writing our scripted module #4

- Create the text field in the **setup** function under the **Apply** button, above **connections**

```
self.centerOfMassValueLabel = qt.QLabel()
parametersFormLayout.addRow("Center of mass", self.centerOfMassValueLabel)
```

Source: PerkLab Bootcamp

Writing our scripted module #5

- Validating button state and displaying the output into **`MyFirstModuleWidget` – replace these functions:**

```
  def onSelect(self):
    self.applyButton.enabled = self.inputSelector.currentNode()

  def onApplyButton(self):
    logic = MyFirstModuleLogic()
    enableScreenshotsFlag = self.enableScreenshotsFlagCheckBox.checked
    imageThreshold = self.imageThresholdSliderWidget.value
    logic.run(self.inputSelector.currentNode(),
self.outputSelector.currentNode(), imageThreshold, enableScreenshotsFlag)
    self.centerOfMassValueLabel.text = str(logic.centerOfMass)
```

Kitware

Source: PerkLab Bootcamp

Trying out scripted module

- Go to our module in *Examples / Center of Mass*
- Add a few markups
- Press *Apply*
  - 1. Display displacement center of mass position → works!
  - 2. Displays nothing → error can be seen in the Python interactor

Kitware

Source: PerkLab Bootcamp

Add auto-update #1

- Repurpose the checkbox:

```
# check box to trigger taking screen shots for later use in tutorials
self.enableScreenshotsFlagCheckBox = qt.QCheckBox()
self.enableScreenshotsFlagCheckBox.checked = 0
self.enableScreenshotsFlagCheckBox.setToolTip(“Enable auto-update")
parametersFormLayout.addRow("Auto-update", self.enableScreenshotsFlagCheckBox)
self.observedMarkupNode = None
self.markupsObserverTag = None
self.enableScreenshotsFlagCheckBox.connect("toggled(bool)", self.onEnableAutoUpdate)
```

Kitware

Source: PerkLab Bootcamp

Add auto-update #2

- Respond to modification events: add these above onApplyButton()

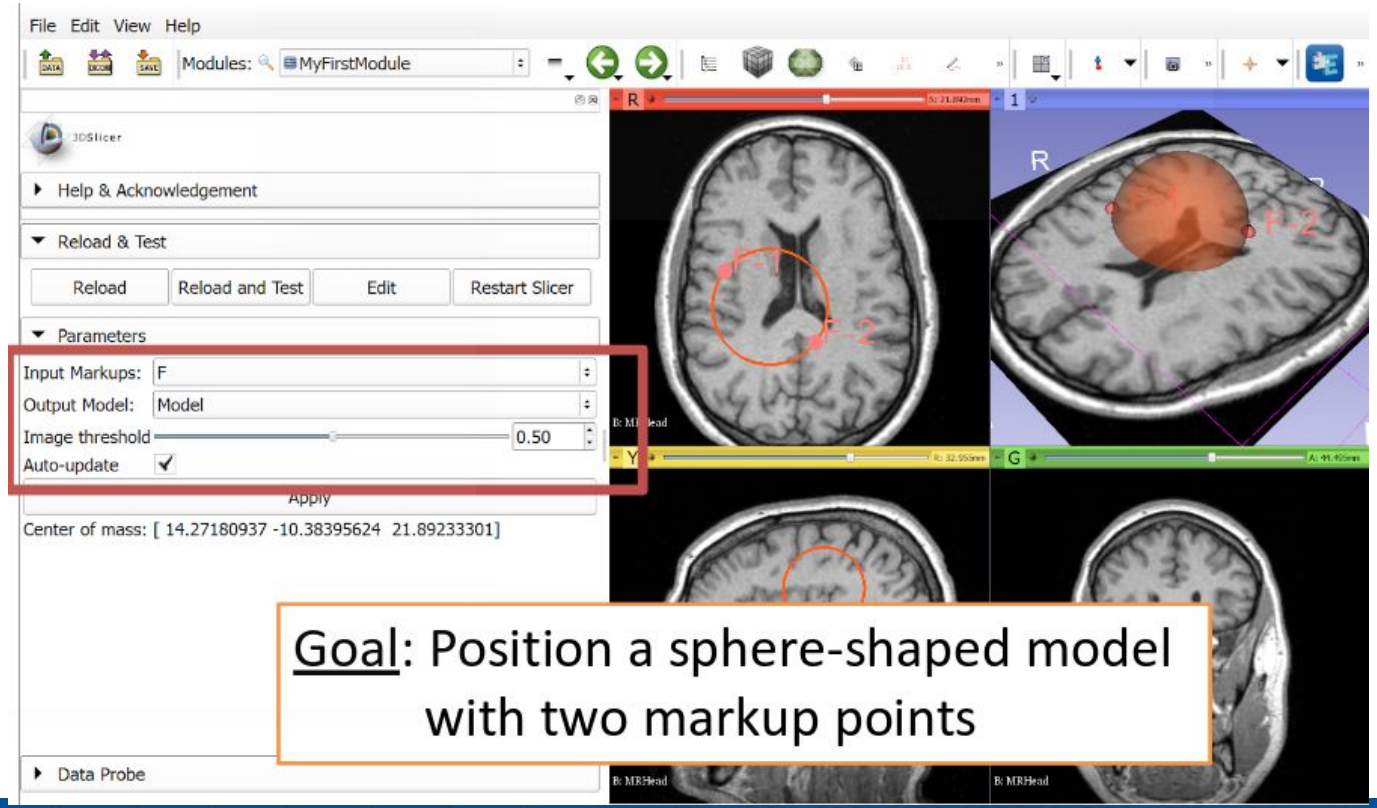
```
def onEnableAutoUpdate(self, autoUpdate):
  if self.markupsObserverTag:
    self.observedMarkupNode.RemoveObserver(self.markupsObserverTag)
    self.observedMarkupNode = None
    self.markupsObserverTag = None
  if autoUpdate and self.inputSelector.currentNode:
    self.observedMarkupNode = self.inputSelector.currentNode()
    self.markupsObserverTag = self.observedMarkupNode.AddObserver(
      vtk.vtkCommand.ModifiedEvent, self.onMarkupsUpdated)

def onMarkupsUpdated(self, caller=None, event=None):
  self.onApplyButton()
```

Kitware

Source: PerkLab Bootcamp

Homework #1

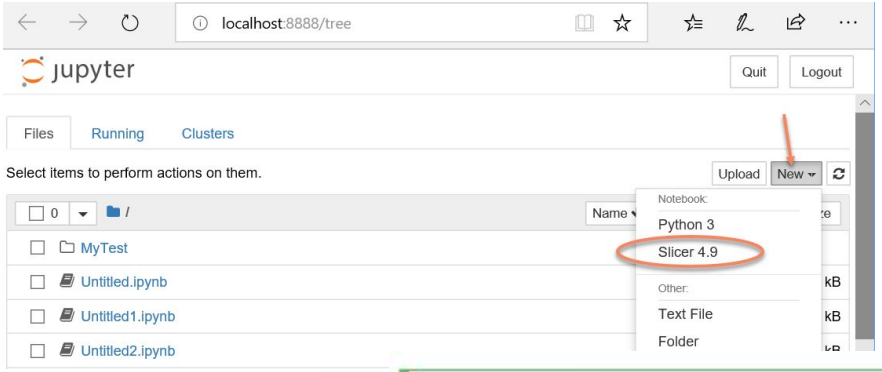


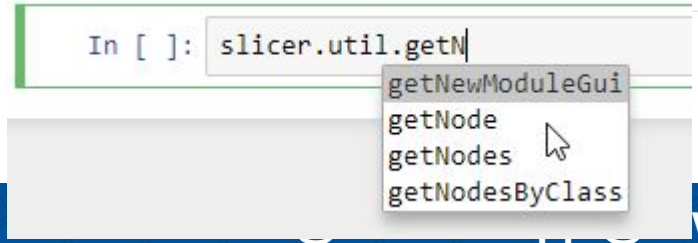
Homework #2

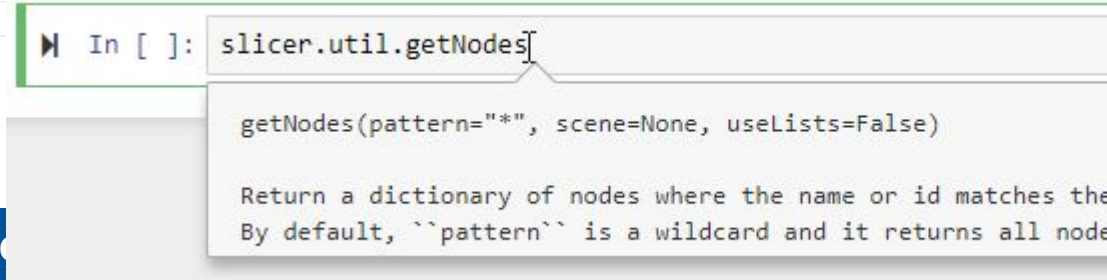
Follow Developing and contributing extensions for 3D Slicer tutorial

Slicer Jupyter

See https://github.com/Slicer/SlicerJupyter#readme







Slicer Plugin Infrastructure

Approaches

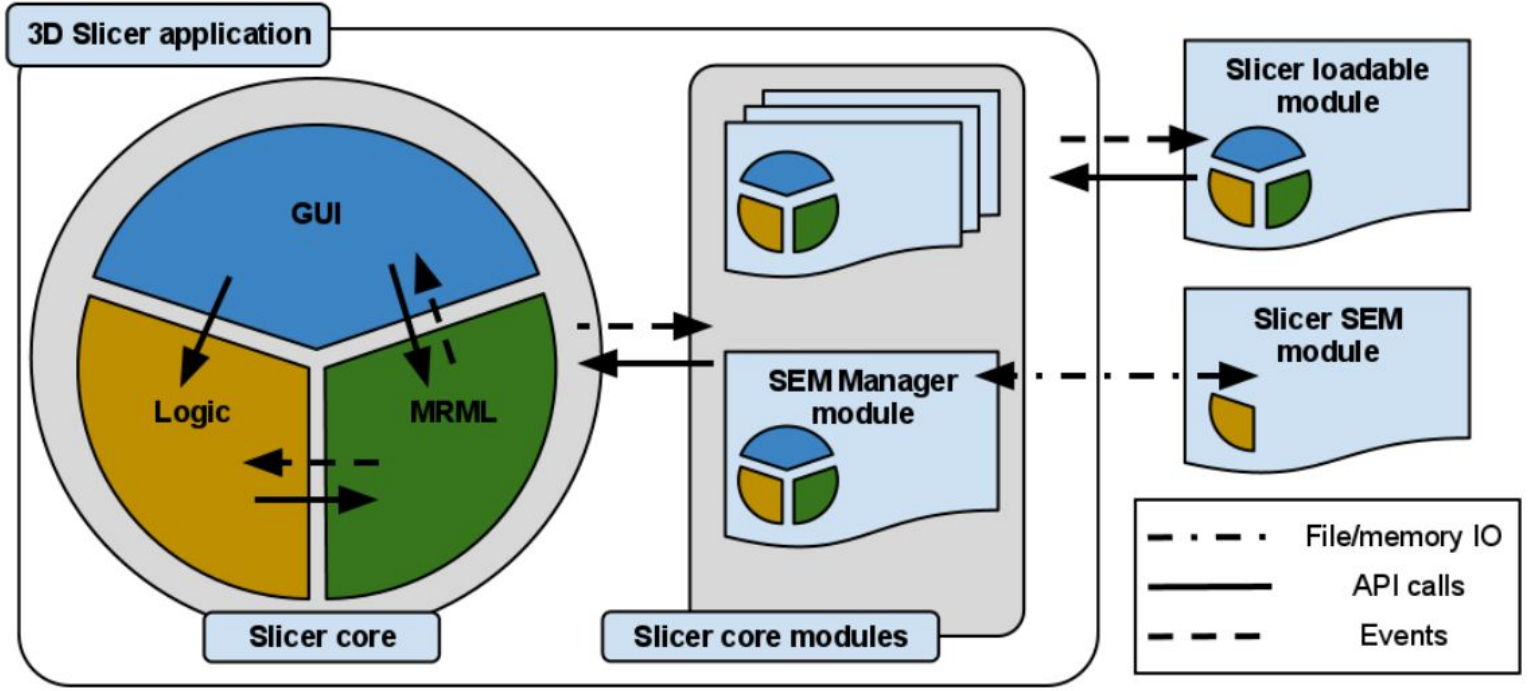
- Two type of approaches
  - Slicer Execution Model (SEM) plugins
  - Slicer loadable modules (C++ or Python)

- Approaches require different level of
  - integration with the Slicer core application
  - expertise in Slicer internals required from the developer.

- Slicer loadable module written in python are called **Scripted Loadable module**

Kitware

Overview



Source: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3466397/

Overview: Source Code (1 / 3)

- Slicer
  - Modules
    - Core
    - CLI
      - AddScalarVolumes
      - ...
    - Loadable
      - Colors
      - Data
      - Models
      - Transforms
      - Volumes
      - …
    - Scripted
      - DICOM
      -

- SlicerExecutionModel: CLI Module
- Slicer Loadable module
- Bridging C++ and Python
- Module lifecycle
- SubjectHierarchy Plugins
- DICOM Plugin
- SegmentEditor Plugin
- IO Plugin
- Custom Layout
- Qt Designer plugin

Kitware

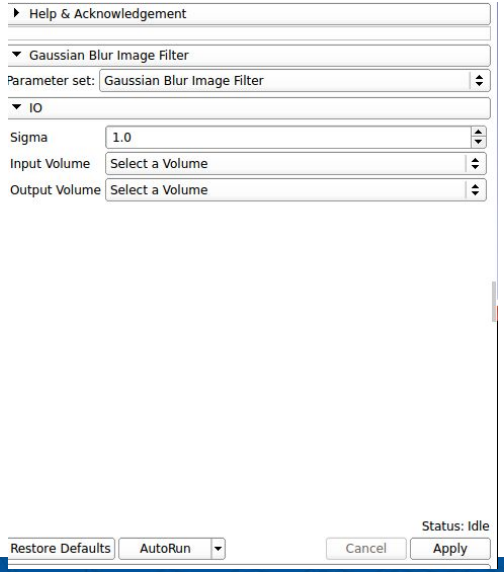
Slicer Execution Model (SEM)

- No knowledge of 3D Slicer architecture
- No dependencies on the Slicer libraries or source code
- Implement the task-specific Controller component of MVC design
- View and Model elements maintained by application via SEM Module Manager (SEM-MM)

Kitware

Slicer Execution Model (SEM)

- Interface specified using XML
  - input/output data (e.g., images, transformations and points)
  - processing parameters
- GUI auto-generated based on XML
- Written in C++
  - executable files
  - shared libraries
- Written as scripts
  - Python
  - MATLAB




```
<?xml version="1.0" encoding="utf-8"?>
<executable>
  <category>Filtering.Denoising</category>
  <index>3</index>
  <title>Gaussian Blur Image Filter</title>
  <description><![CDATA[Apply a gaussian blurr to an image]]></description>
  <version>0.1.0.$Revision: 1.1 $(alpha)</version>
  <documentation-url>http://wiki.slicer.org/slicerWiki/index.php/Documentation/N
  <license/>
  <contributor>Julien Jomier (Kitware), Stephen Aylward (Kitware)</contributor>
  <acknowledgements><![CDATA[This work is part of the National Alliance for Medi
  <parameters>
    <double>
      <name>sigma</name>
      <longflag>sigma</longflag>
      <flag>s</flag>
      <label>Sigma</label>
      <description><![CDATA[Sigma value in physical units (e.g., mm) of the Gaus
      <default>1.0</default>
    </double>
    <label>IO</label>
    <description><![CDATA[Input/output parameters]]></description>
    <image>
      <name>inputVolume</name>
      <label>Input Volume</label>
      <channel>input</channel>
      <index>0</index>
      <description><![CDATA[Input volume]]></description>
    </image>
    <image reference="inputVolume">
      <name>outputVolume</name>
      <label>Output Volume</label>
      <channel>output</channel>
      <index>1</index>
      <description><![CDATA[Blurred Volume]]></description>
    </image>
```

Source Files Organization

- CLI
  - AddScalarVolumes
    - CMakeLists.txt
    - AddScalarVolumes.cxx
    - AddScalarVolumes.xml
  - ...

Kitware

Slicer Loadable Modules

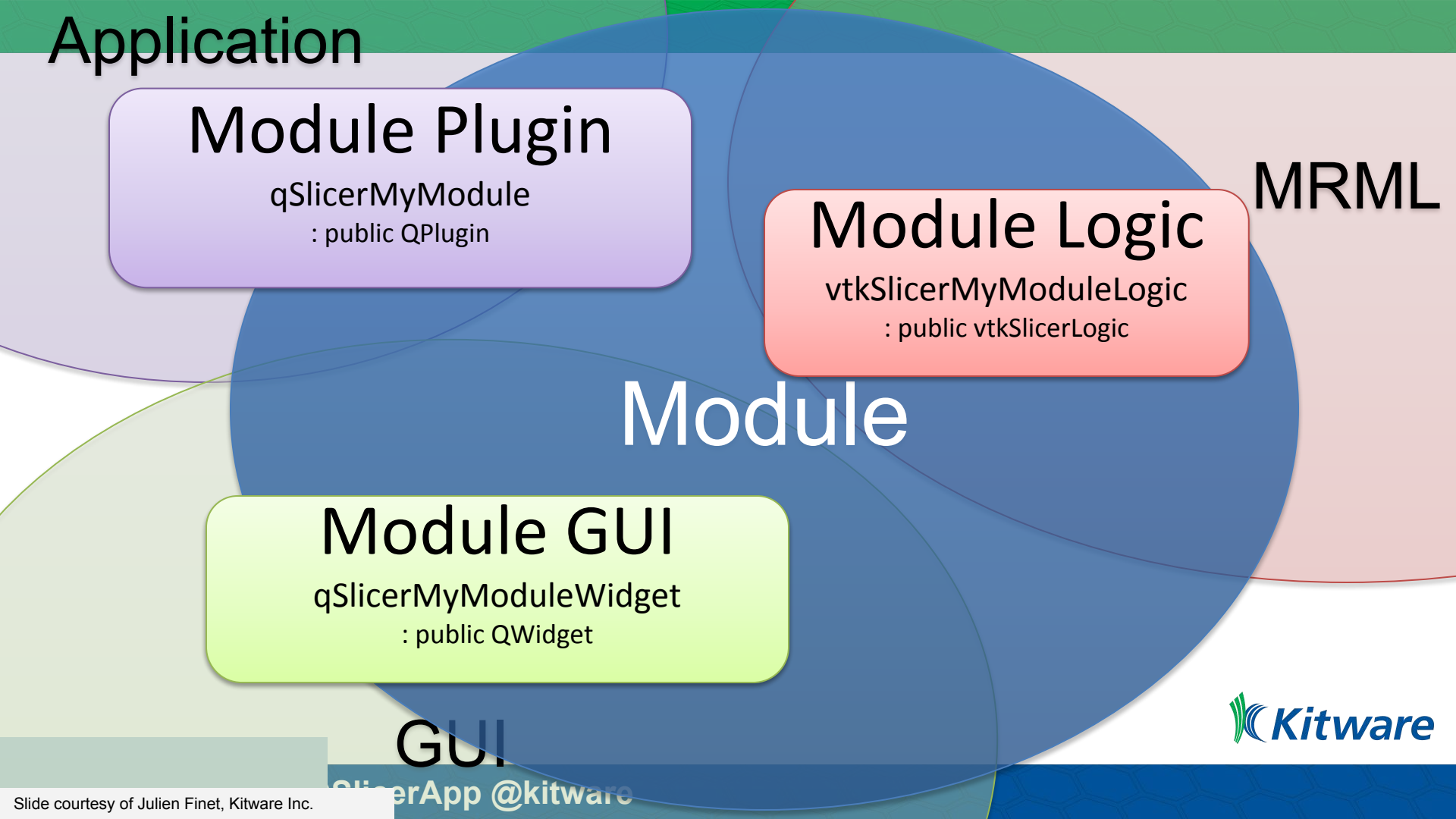
- Two flavors: C++ and Python
- Use cases:
  - for interactive tools
  - for new MRML data types & event handling
  - for customization of the main Slicer GUI
- Same MVC design pattern as the application:
  - module-specific Logic, GUI and MRML classes

Kitware

Slicer Loadable Modules

- Logic classes
  - core computation / analysis algorithms
  - ITK and/or VTK pipelines
- View elements - Allow user to:
  - interact with the module
  - initialize the inputs and processing parameters
  - interact with the viewers (e.g seed points)

Kitware



Slide courtesy of Julien Finet, Kitware Inc.

The Real Class Hierarchy

- qSlicerMyModule
  - qSlicerLoadableModule
    - qSlicerAbstractModule
      - qSlicerAbstractCoreModule
        - QObject
- qSlicerMyModuleWidget
  - qSlicerAbstractModuleWidget
    - qSlicerWidget
      - qSlicerWidget
        - QWidget
        - qSlicerObject
    - qSlicerAbstractModuleRepresentation

Source files organization (1 / 2)

- Slicer/Modules/Loadable
  - MyModule
    - Documentation
    - Logic
    - MRMLDM
    - MRML
    - Resources
      - UI
    - Testing
    - Widgets
      - DesignerPlugins

Kitware

Source Files Organization (2 / 2)

- Slicer/Modules/Scripted
  - MyModule
    - MyModule.py

Bridging C++ and Python

- Application Core
  - written in c++
  - embed python
  - manage plugin based on “interfaces”:
    - qSlicerLoadableModule
    - qSlicerAbstractModuleWidget

- Solution: Create “bridge” classes

Kitware

Bridge Classes (1 / 2)

- Use Python C/API
- Load python script (or look up imported module)
  - Get reference to python class
  - Instantiate python object from c++
  - Get reference to object methods

Kitware

Bridge Classes (2 / 2)

- qSlicerScriptedLoadableModule
- qSlicerScriptedLoadableModuleWidget

```
205
206 //-----------------------------------------------------------------------------
207 void qSlicerScriptedLoadableModuleWidget::exit()
208 {
209   Q_D(qSlicerScriptedLoadableModuleWidget);
210   this->Superclass::exit();
211   d->PythonCppAPI.callMethod(Pimpl::ExitMethod);
212 }
213
214 //-----------------------------------------------------------------------------
215 bool qSlicerScriptedLoadableModuleWidget::setEditedNode(vtkMRMLNode* node,
216                                                         QString role /* = QString()*/,
217                                                         QString context /* = QString()*/)
218 {
219   Q_D(qSlicerScriptedLoadableModuleWidget);
220   PyObject* arguments = PyTuple_New(3);
221   PyTuple_SET_ITEM(arguments, 0, vtkPythonUtil::GetObjectFromPointer(node));
222   PyTuple_SET_ITEM(arguments, 1, PyString_FromString(role.toLatin1()));
223   PyTuple_SET_ITEM(arguments, 2, PyString_FromString(context.toLatin1()));
224   PyObject* result = d->PythonCppAPI.callMethod(d->SetEditedNodeMethod, arguments);
225   Py_DECREF(arguments);
226   if (!result)
227     {
228     // Method call failed (probably an omitted function), call default implementation
229     return this->Superclass::setEditedNode(node);
230     }
231
232   // Parse result
233   if (!PyBool_Check(result))
234     {
235     qWarning() << d->PythonSource << ": qSlicerScriptedLoadableModuleWidget: Function 'setE
236     return false;
237     }
238
239   return (result == Py_True);
240 }
```

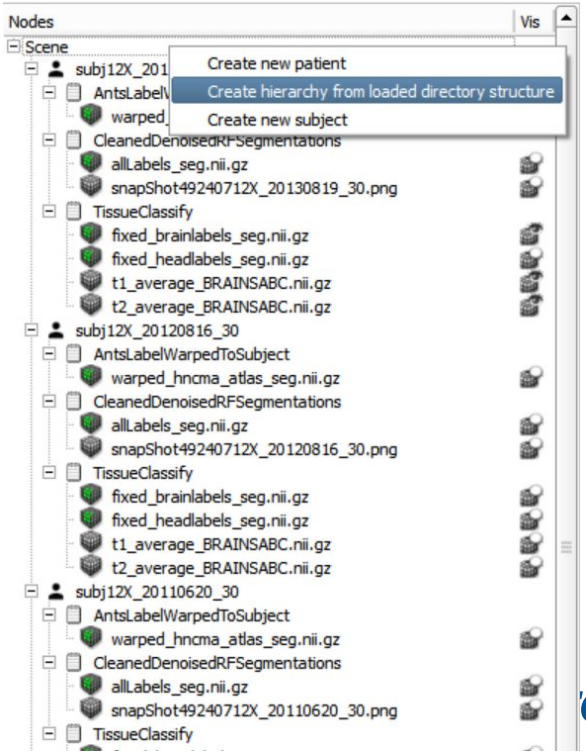
Life of a module…

1. Slicer start-up
2. Slicer discovers MyModule
   a. Resolve dependencies
   b. Add module title in menu
   c. Create module logic
      - Optionally register new nodes
   d. Setup module
      - Optionally registerIO, displayableManager, ...
3. Slicer loads a file (Add Data)
   a. Check that registered IO can read
   b. Display IO options within Add Data dialog
   c. Load file using IO
4. Create module widget on first show

Kitware

SubjectHierarchy Plugins

- support for data node types
- context menu items.
- Flavors:
  - C++: Derive qSlicerSubjectHierarchyAbstractPlugin
  - Python: Derive AbstractScriptedSubjectHierarchyPlugin

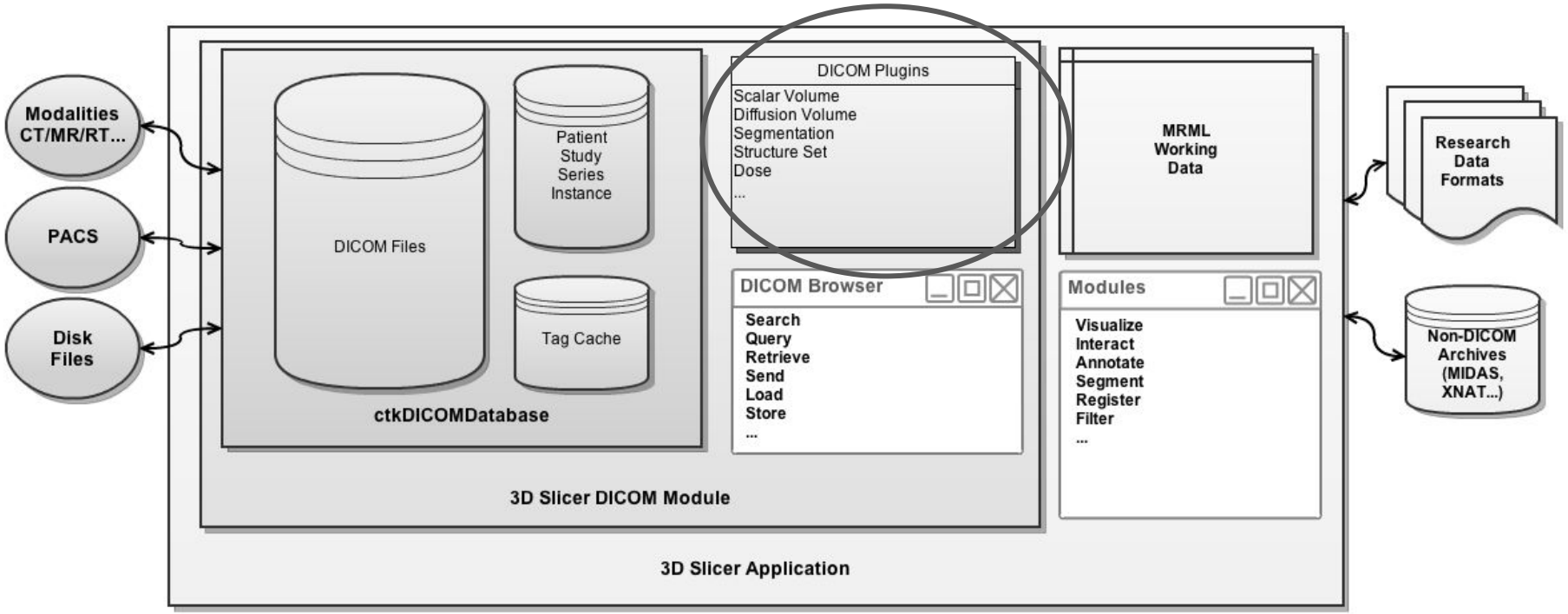


Module Specific Plugin

- Domain specific
  - DICOM plugin (python)
  - Segmentation plugin (python or c++)

Kitware

DICOM Plugin: Architecture Overview



Source: https://www.slicer.org/wiki/Documentation/Nightly/Modules/DICOM

DICOM Plugin

- Allows acquisition-specific and modality-specific interpretation of DICOM data
- Written in Python

```
153 class DICOMDiffusionVolumePlugin:
154   """
155   This class is the 'hook' for slicer to detect and recognize the plugin
156   as a loadable scripted module
157   """
158   def __init__(self, parent):
159     parent.title = "DICOM Diffusion Volume Plugin"
160     parent.categories = ["Developer Tools.DICOM Plugins"]
161     parent.contributors = ["Steve Pieper (Isomics Inc.)"]
162     parent.helpText = """
163     Plugin to the DICOM Module to parse and load diffusion volumes
164     from DICOM files.
165     No module interface here, only in the DICOM module
166     """
167     parent.acknowledgementText = """
168     This DICOM Plugin was developed by
169     Steve Pieper, Isomics, Inc.
170     and was partially funded by NIH grant 3P41RR013218.
171     """
172
173     # Add this extension to the DICOM module's list for discovery when the module
174     # is created.  Since this module may be discovered before DICOM itself,
175     # create the list if it doesn't already exist.
176     try:
177       slicer.modules.dicomPlugins
178     except AttributeError:
179       slicer.modules.dicomPlugins = {}
180     slicer.modules.dicomPlugins['DICOMDiffusionVolumePlugin'] = DICOMDiffusionVolumePluginClass
```

```
16   def __init__(self):
17     super(DICOMDiffusionVolumePluginClass,self).__init__()
18     self.loadType = "Diffusion Volume"
19     # these are the required tags for each vendor
20     # TODO: it doesn't seem that DicomToNrrd supports
21     # standard DICOM Diffusion, but when it does we should
22     # add a set of required tags based on supplement 49
23     self.diffusionTags = {
24         'GE' : [
25             '0043,1039', # B Value of diffusion weighting
26             '0019,10bb', # X component of gradient direction
27             '0019,10bc', # Y component of gradient direction
28             '0019,10bd', # Z component of gradient direction
29           ],
30         'Siemens' : [
31             '0051,100b', # "Mosiac Matrix Size"
32             '0019,100a', # "Number of Images In Mosaic"
33             '0019,100c', # "B Value of diffusion weighting"
34             '0019,100e', # "Diffusion Gradient Direction"
35             '0019,1027', # "Diffusion Matrix"
36             '0029,1010', # "Siemens DWI Info"
37           ],
38         'Philips' : [
39             '2001,1003', # "B Value of diffusion weighting"
40             '2001,1004', # "Diffusion Gradient Direction"
41             '2005,10b0', # "Diffusion Direction R/L"
42             '2005,10b1', # "Diffusion Direction A/P"
43             '2005,10b2', # "Diffusion Direction F/H"
44           ],
45         }
```

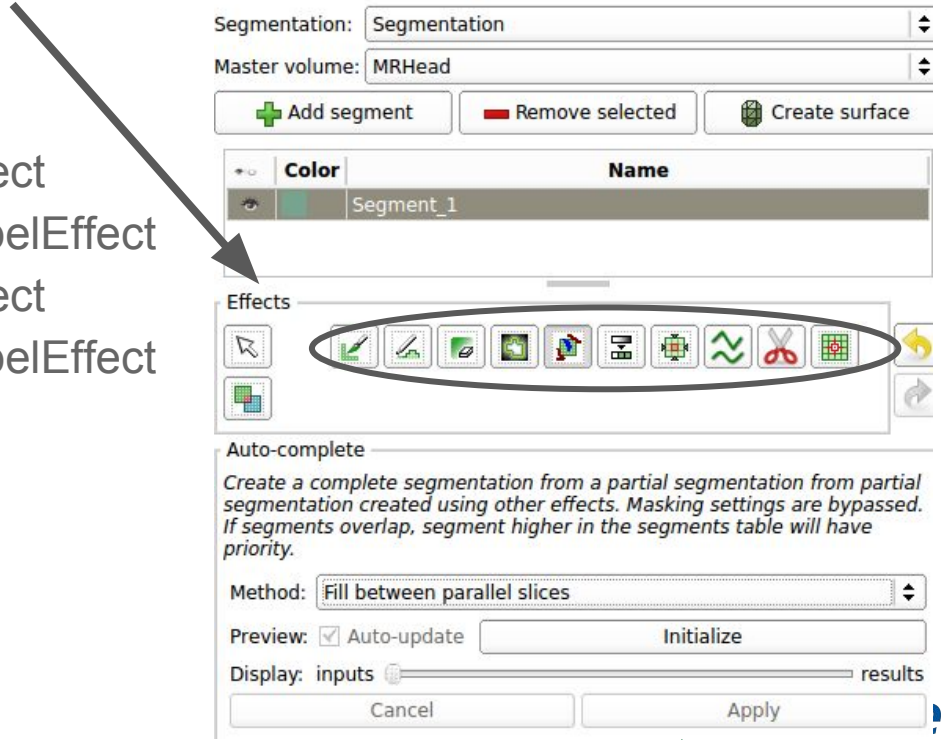
Source: Modules/Scripted/DICOMPlugins/DICOMDiffusionVolumePlugin.py

SegmentEditor

- Delineate structures of interest
- Features:
  - Overlapping segments
  - Conversion to model representation is automatic and real-time
  - Segments table allows robust per-structure handling, and advanced visualization settings for segments
  - 3D brushes allow the user to paint in 3D rather than slice by slice
  - Segment Editor widget can be embedded into any module or slicelet
  - Master Volume Selection allows users to change the master volume during segmentation
  - Oblique slices can be edited

SegmentEditor: Plugins (aka Effects)

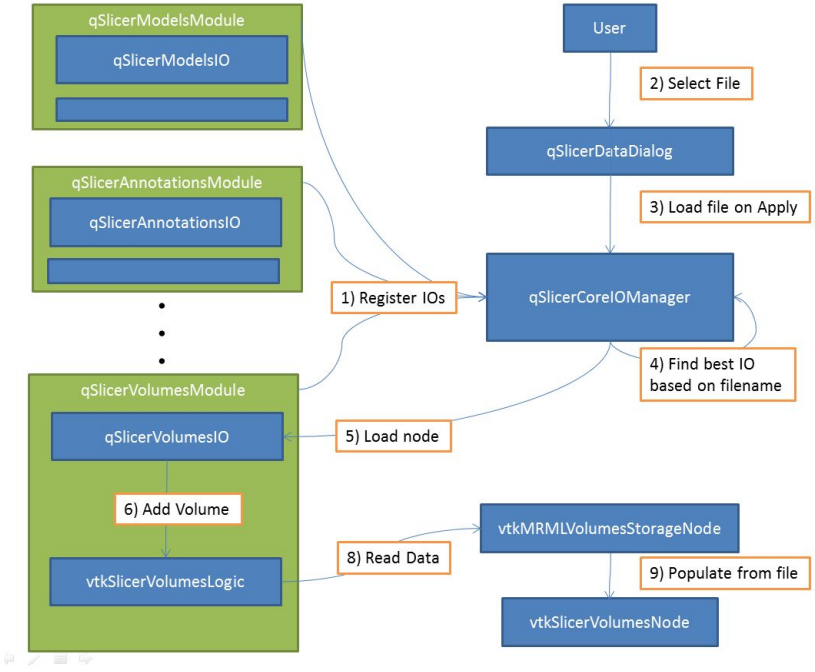
- Adding a new EditorEffect
- Type of plugins:
  - qSlicerSegmentEditorAbstractEffect
  - qSlicerSegmentEditorAbstractLabelEffect
  - qSlicerSegmentEditorScriptedEffect
  - qSlicerSegmentEditorScriptedLabelEffect



IO Plugin (1 / 3)

- Registered from application or loadable module

- vtkSlicerXYZNode:
  - Storable node
  - Represents the data from file (e.g. vtkMRMLModelNode)
- vtkSlicerXYZStorageNode:
  - File reader
  - Populates storable node using vtkMRMLXYZStorageNode::ReadDa



Source: wiki.slicer.org

IO Plugin (2 / 3)

- vtkSlicerXYZsLogic:
  - MRML logic of the XYZ module
  - Exposes method AddXYZ(const char* fileName, const char* nodeName=0)
    - Creates “vtkMRMLXYZNode” and its associated ”vtkMRMLXYZStorageNode”
    - Add nodes into scene
    - Call vtkMRMLXYZStorageNode::ReadData(vtkMRMLStorableNode*,bool)
- qSlicerXYZsIO
  - plugin registered by modules into qSlicerCoreIOManager
  - interface between Qt and MRML logics
  - internally calls vtkSlicerXYZsLogic::AddXYZ().
- qSlicerXYZsIOOptionsWidget
  - widget that sets loading options that gets passed to the logic.

Kitware

IO Plugin (3 / 3)

- qSlicerCoreIOManager:
  - central class where any IO operation must go through
  - qSlicerIOs can be registered using qSlicerCoreIOManager::registerIO(qSlicerIO*)
  - nodes can be loaded using qSlicerCoreIOManager::loadNodes(...)
- qSlicerDataDialog:
  - dialog that allows the user to select the files to load.
- qSlicerScriptedFileReader
- qSlicerScriptedFileWriter

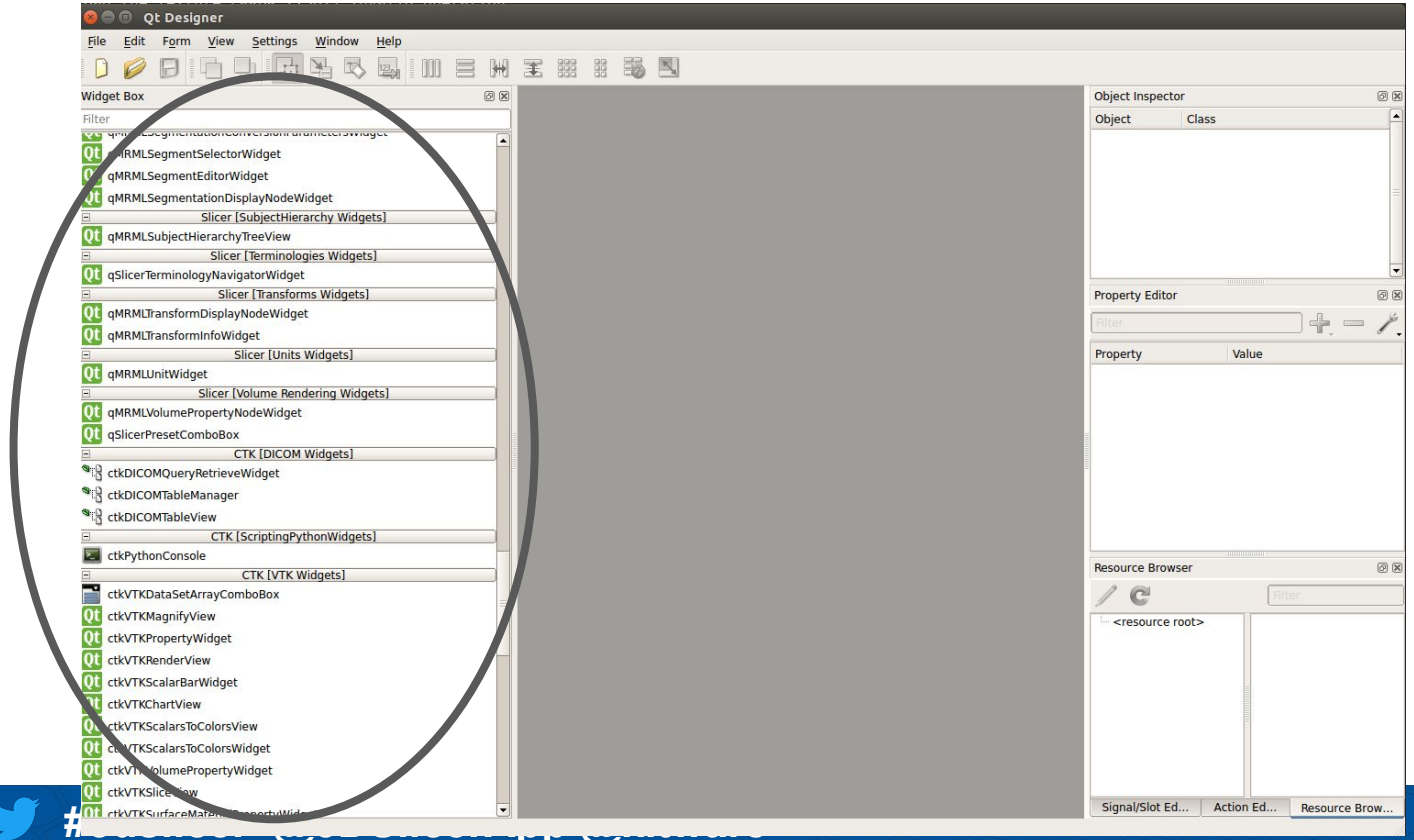
Kitware

Custom Layout

- Custom layout can be registered using python or C++.
  - For example, see here
- Slice/3D/Chart/Table view factory can be replaced.
  - For example, see here
- References:
  - https://www.slicer.org/wiki/Documentation/Nightly/ScriptRepository#Customize_viewer_layout
  - https://www.slicer.org/wiki/Documentation/Nightly/Developers/Layouts

Kitware

Qt Designer plugin (1 / 2)



Kitware

Qt Designer plugin (2 / 2)

```
138     qMRMLTransformSlidersPlugin.h
139     qMRMLTreeViewPlugin.h
140     qMRMLVolumeInfoWidgetPlugin.h
141     qMRMLVolumeThresholdWidgetPlugin.h
142     qMRMLWidgetPlugin.h
143     qMRMLWindowLevelWidgetPlugin.h
144     )
145
146   set(${KIT}_TARGET_LIBRARIES
147     qMRMLWidgets
148     )
149
150   ctkMacroBuildQtDesignerPlugin(
151     NAME ${PROJECT_NAME}
152     EXPORT_DIRECTIVE ${${KIT}_EXPORT}
153     FOLDER "${${KIT}_FOLDER}"
154     SRCS ${${KIT}_SRCS}
155     MOC_SRCS ${${KIT}_MOC_SRCS}
156     TARGET_LIBRARIES ${${KIT}_TARGET_LIBRARIES}
157     )
```

Libs/MRML/Widgets/DesignerPlugins/CMakeLists.txt

```
1   #ifndef __qMRMLLinearTransformSliderPlugin_h
2   #define __qMRMLLinearTransformSliderPlugin_h
3
4   #include "qMRMLWidgetsAbstractPlugin.h"
5
6   class QMRML_WIDGETS_PLUGINS_EXPORT qMRMLLinearTransformSliderPlugin : public QObject,
7                                       public qMRMLWidgetsAbstractPlugin
8   {
9     Q_OBJECT
10
11  public:
12    qMRMLLinearTransformSliderPlugin(QObject *_parent = 0);
13
14    QWidget *createWidget(QWidget *_parent);
15    QString domXml() const;
16    QIcon   icon() const;
17    QString includeFile() const;
18    bool isContainer() const;
19    QString name() const;
20
21  };
22
23  #endif
```

- SlicerExecutionModel: CLI Module
- Slicer Loadable module
- Bridging C++ and Python
- Module lifecycle
- SubjectHierarchy Plugins
- DICOM Plugin
- SegmentEditor Plugin
- IO Plugin
- Custom Layout
- Qt Designer plugin

Kitware

Slicer Custom Application

Slide: Credits Andras Lasso, The Perk Lab, Queens University

Translational Research: Right tool for the right job

Technological prototype

Can it be done?

Innovative,
not robust,
usually single developer
supported

Research tool

Should it be done?

Robust and usable
enough for clinical
evaluation, flexible,
open, portable,
community supported

Clinical tool

Patient ready
FDA approved, company
supported, closed source

Kitware

What are the building blocks ?

Adapted fom Andras Lasso slides, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

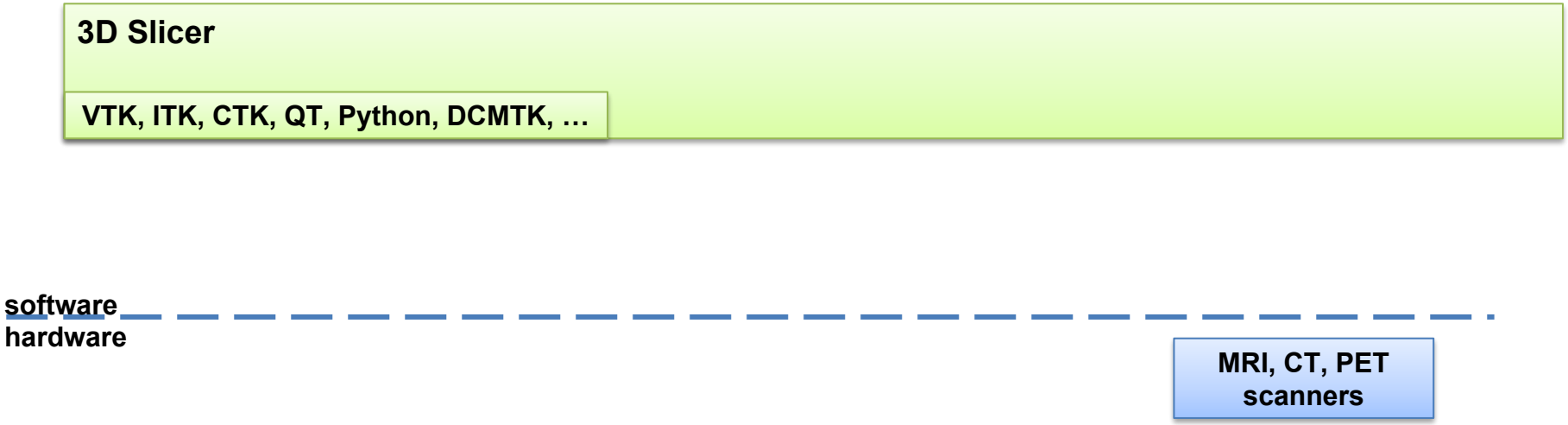
Medical Scanners

software
hardware

MRI, CT, PET scanners

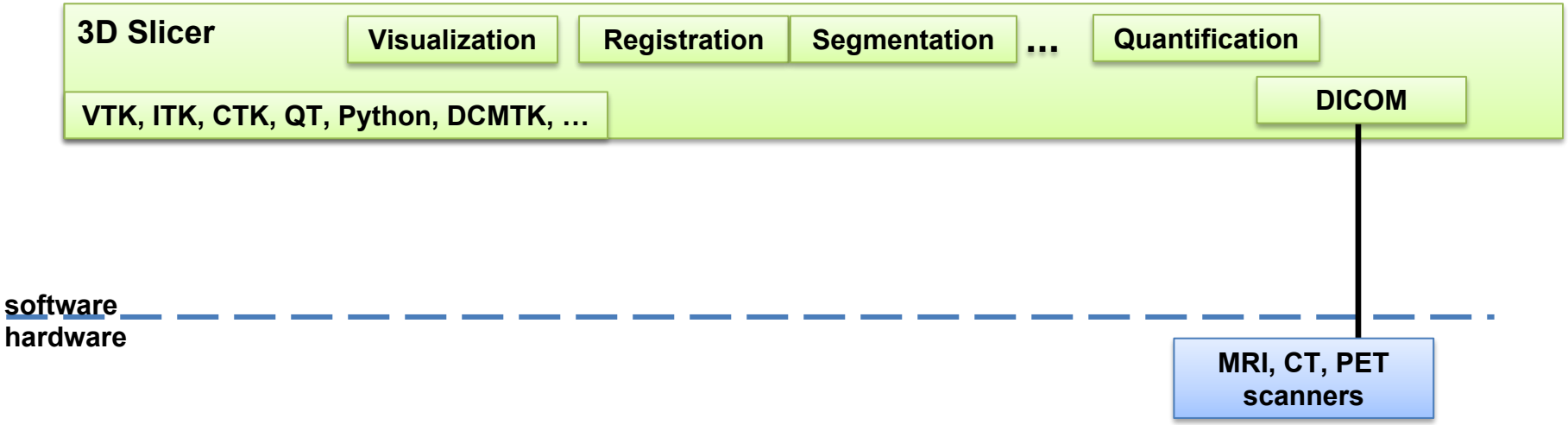
Adapted fom Andras Lasso slides, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

3D Slicer application and libraries



Adapted fom Andras Lasso slides, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

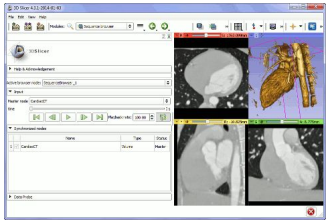
3D Slicer modules



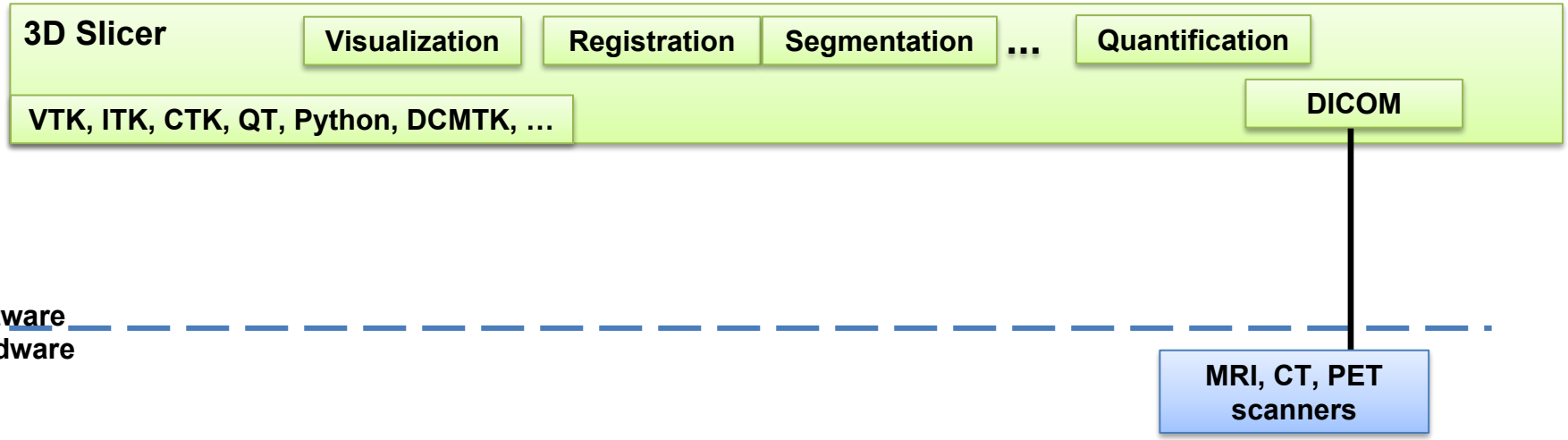
Adapted fom Andras Lasso slides, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

3D Slicer extensions

… more than 100 extensions



Slicer extensions



Customization Approaches

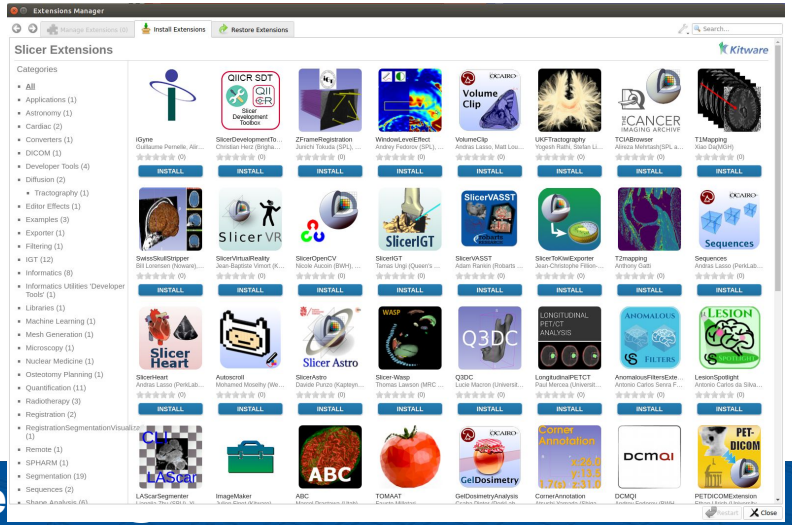
- Level 1: Installing Extensions
- Level 2: Slicelet and Guidelet
- Level 3: Slicer Custom Application: support packaging and distribution

Customization Level 1

Customization Options: Level 1

- Adding new module: adds new features to 3D Slicer, may contain new data types (MRML nodes), algorithms (logic class), user interface (widget class, displayable manager class)

- Adding new extension: package of related modules that users can install



Customization Level 2

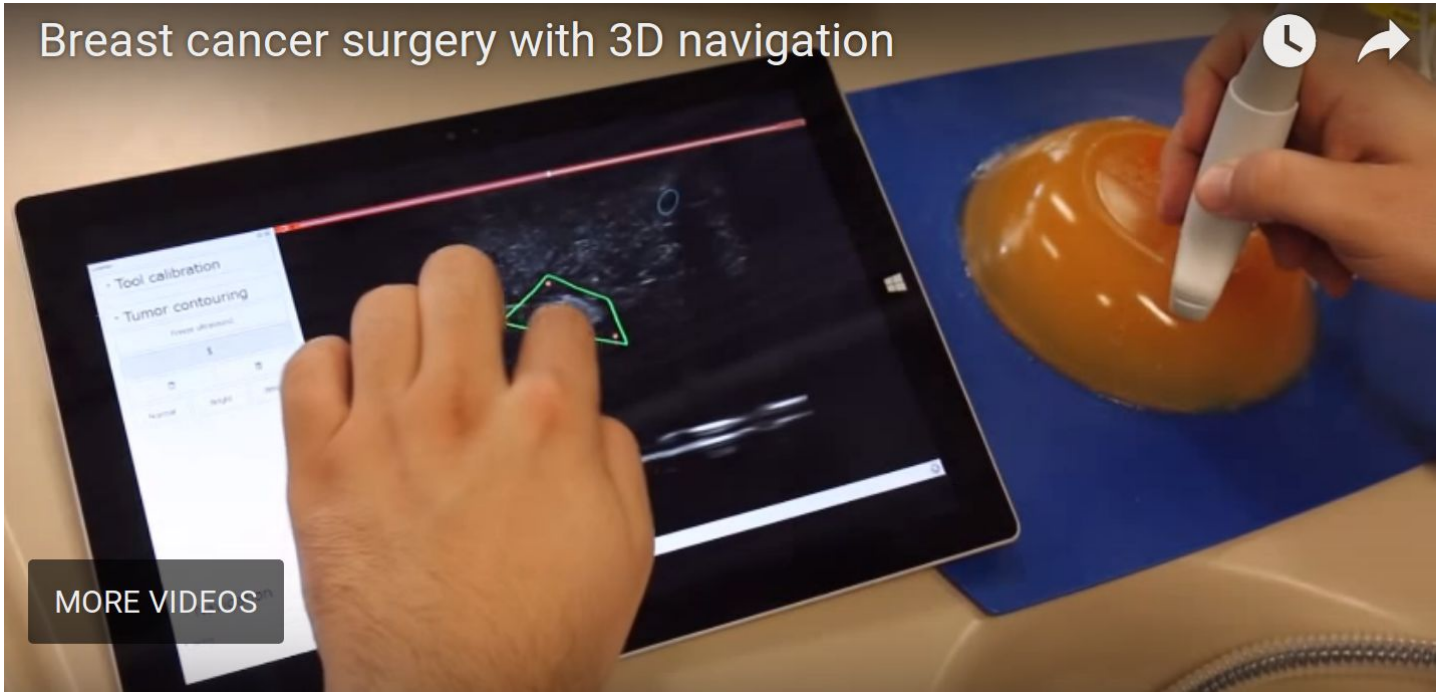
Customization Options: Level 2 (Slicelet and Guidelet)

- Slicelet: module that draws its entire user interface – it can run without creating a 3D Slicer main application window or by hiding the main application window, can toggle between full Slicer GUI/simple GUI
  See www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Developers/Slicelets

- Guidelet: slicelet for interventional guidance applications – has built-in support for tool navigation, real-time imaging control (ultrasound), touch optimized, workflow based user interface (www.slicerigt.org)

Kitware

Level 2: Example #1: Slicelet/Guidelet for Suregry



See http://www.slicerigt.org/wp/breast-cancer-surgery/

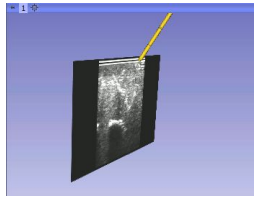
Kitware

Adapted fom Andras Lasso slides, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

Level 2: Example #1: Real time guidance - slicerigt.org

Custom application and interface

LumpNav ... ProstateNav



Slicer extensions

SlicerIGT

Pivot calibration | Model registration | ... | Breach warning | Tool watchdog | ...

3D Slicer SDK

Visualization | Registration | Segmentation | ... | Quantification

VTK, ITK, CTK, QT, Python, DCMTK, …

OpenIGTLink | DICOM

PLUS

Calibration, synchronization, pre-processing, simulation, record&replay | OpenIGTLink

Device interface

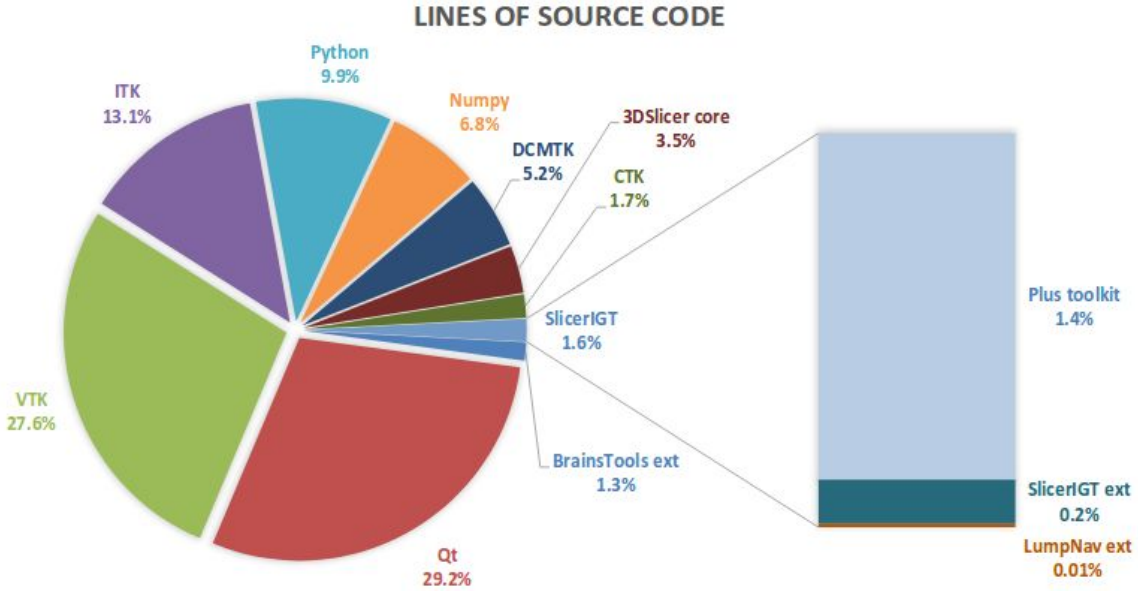
software

hardware

Surgical microscopes, endoscopes | Ultrasound scanners | Navigation systems | Robotic devices, manipulators | MRI, CT, PET scanners

hundreds of devices (imaging, position tracking, various sensors, and manipulators)

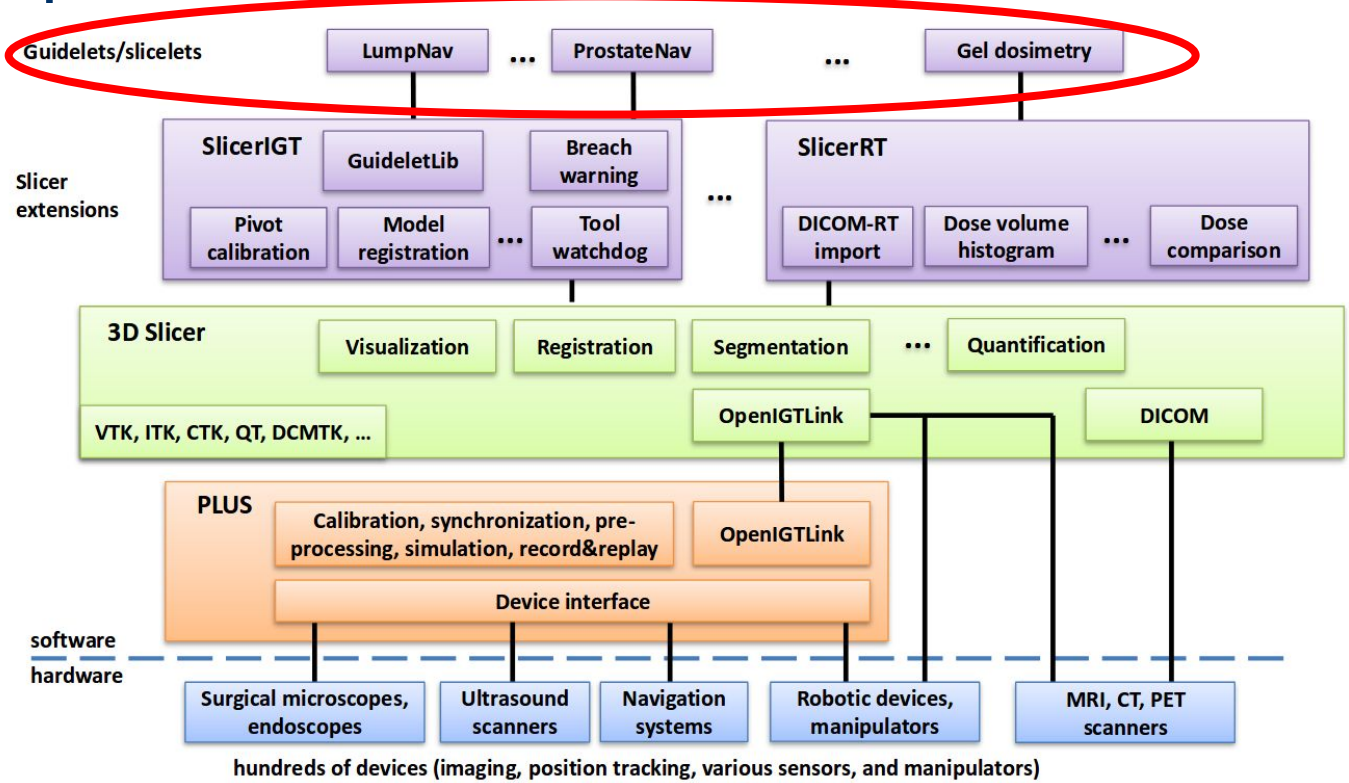
Level 2: Example #1: Software Size Statistics



In this example, the custom Slicelet/Guidelet size is ~0.01%

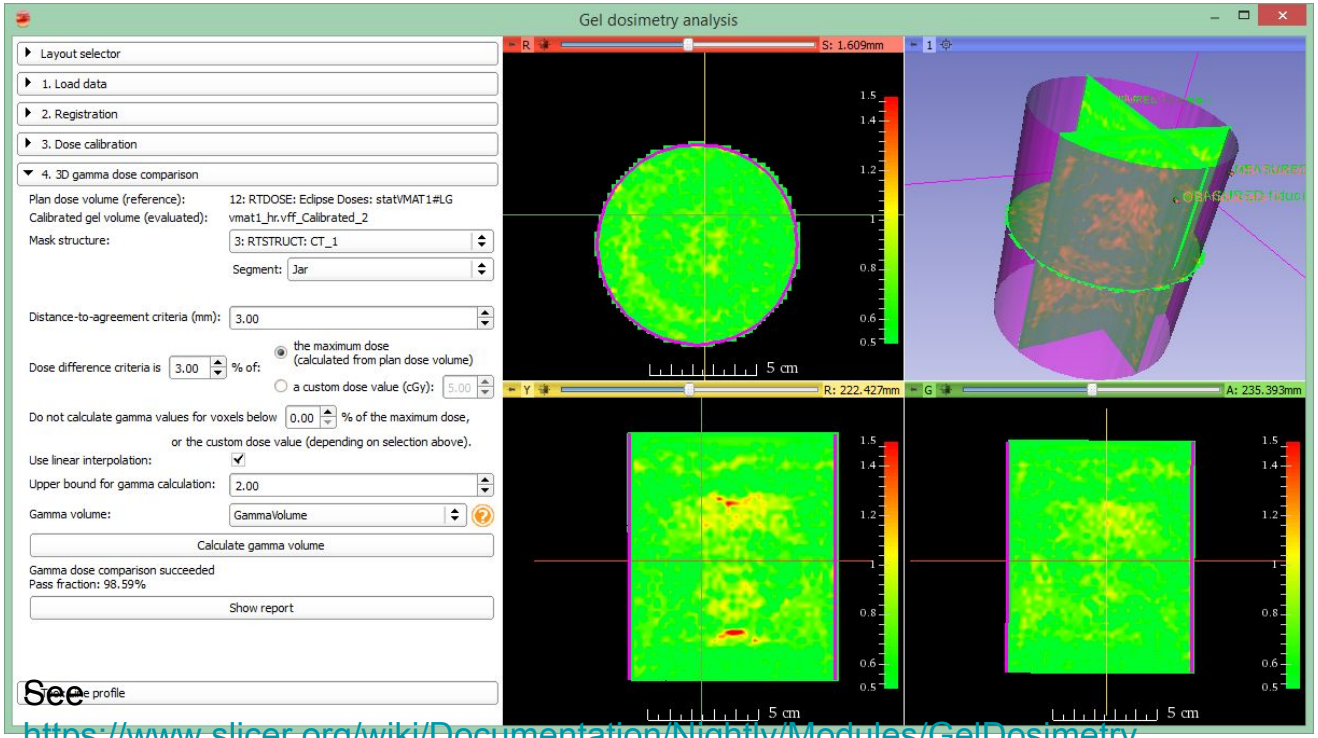
Slide courtesy of Andras Lasso, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

Example #1: System Imlementation Overview



Slide courtesy of Andras Lasso, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

Level 2: Example #2: Slicelet/Guidelet for Gel Dosimetry



See https://www.slicer.org/wiki/Documentation/Nightly/Modules/GelDosimetry

Level 2: Example #1: Hardware Components Overv

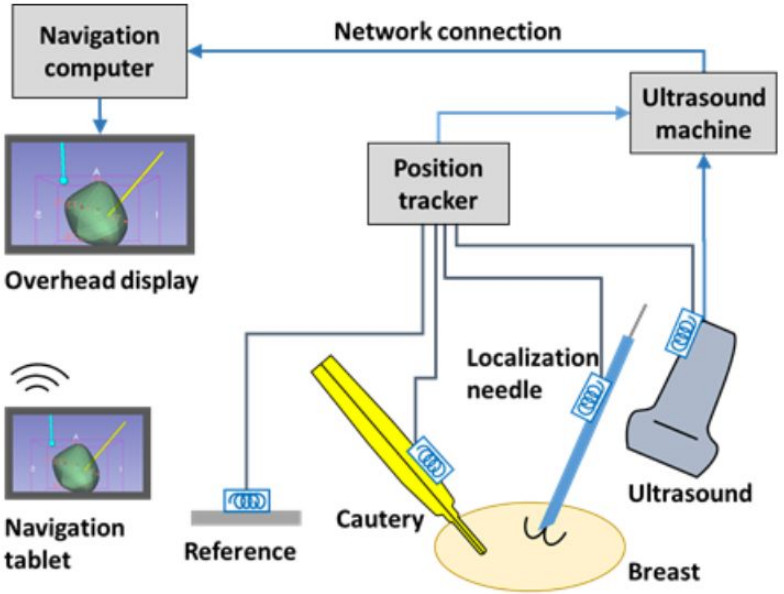


Fig. 1. Overview of the hardware components in the navigation system. Blue coils represent electromagnetic position sensors. The navigation tablet is wirelessly connected to the navigation computer to allow table-side viewing and control.

Video: https://www.youtube.com/watch?v=90l0T1ADe_Y
(Warning - Graphic Content: Real surgery)

Kitware

Ungi, Tamas, et al. "Navigated breast tumor excision using electromagnetically tracked ultrasound and surgical instruments." IEEE Transactions on Biomedical Engineering 63.3 (2016): 600-606.
See http://perk.cs.queensu.ca/sites/perkd7.cs.queensu.ca/files/Ungi2015b.pdf

Adapted fom Andras Lasso slides, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

Level 2: Example #2: Radiation Therapy - slicerrt.org

Custom application and interface

Slicer extensions

...

Gel dosimetry

...

SlicerRT

DICOM-RT import

Dose volume histogram

...

Dose comparison

3D Slicer SDK

Visualization

Registration

Segmentation

...

Quantification

VTK, ITK, CTK, QT, Python, DCMTK, …

DICOM

software

hardware

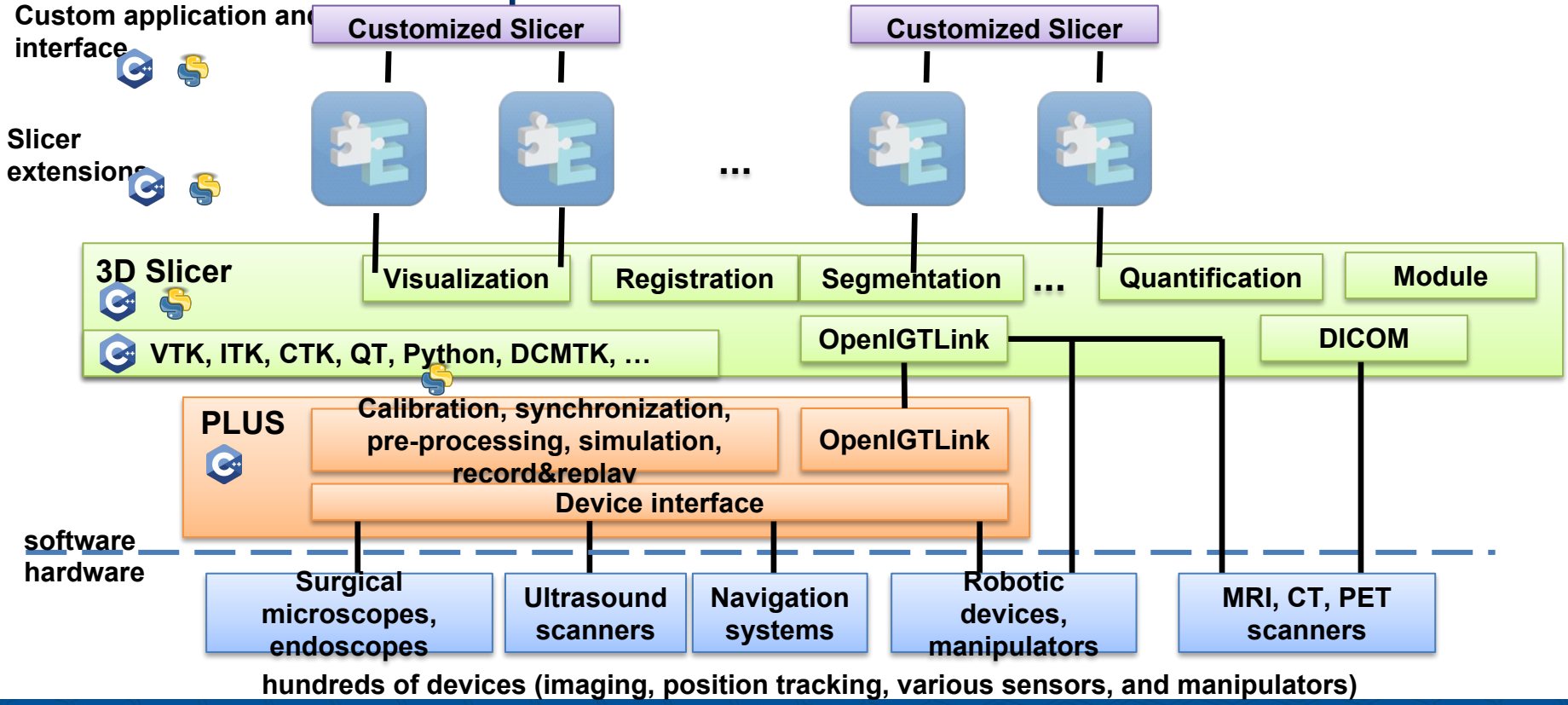
MRI, CT, PET scanners

#3dslicer @3DSlicerApp @kitware

Customization Level 3

Adapted fom Andras Lasso slides, The Perk Lab, Queens University
See https://na-mic.org/w/images/b/b0/Slicelets2016.pdf

Customization Options: Level 3



Customization Options: Level 3

- Custom application: 3D Slicer installation packages can be built with customized branding and feature set – custom modules can be included, some default built-in modules excluded, custom application name, startup message, splash screen, startup module, user preferences, etc.

  See https://github.com/KitwareMedical/SlicerCustomAppTemplate

Kitware

Level 3: SlicerCustomAppTemplate

```
$ pip install cookiecutter

$ cookiecutter \
    gh:KitwareMedical/SlicerCustomAppTemplate
```

Exercise 12: Create a custom application

Level 3: Packaging a Slicer Custom Application

- Similar to Slicer -> build the “package” target

  See https://www.slicer.org/wiki/Documentation/Nightly/Developers/Build_Instructions#PACKAGE_Slicer

Kitware

Installing Additional Python Packages

Installing Additional Python Packages

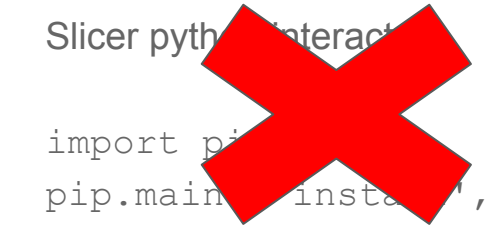
- Pip is available in
  - Slicer python interactor

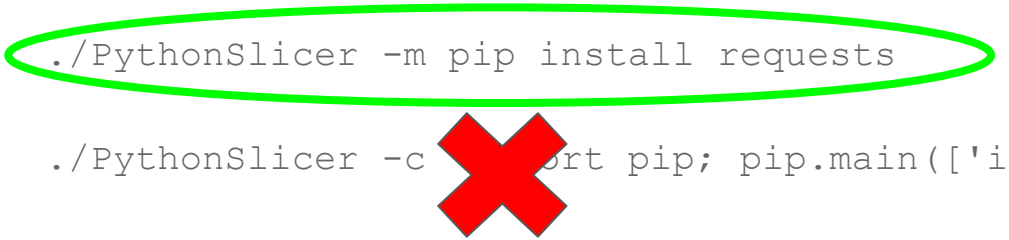
```
import pip
pip.main(['install', 'pygments'])
```

  - Slicer python interpreter

```
./PythonSlicer -m pip install requests
```

```
./PythonSlicer -c "import pip; pip.main(['install', 'requests'])"
```





Kitware

Installing Additional Python Packages: Caveats

- Pure python wheels: All platforms
- Binary python wheels:
  - The one including compiled modules
  - Linux: supported
  - macOS: most likely
  - Windows: not supported.
    - Because the official python 2.7 package are built with a compiler different from the one used for Slicer.
    - After switching Slicer Python to python >= 3.5, it will be supported.
    - Indeed, we already standardized on Visual Studio 2015 (the official compiler for Python 3.5)

Kitware

Slicer Build System

Overview of the Slicer Build System

- Based on CMake
- Modular and extensible
- Cross-platform, cross-generator

Kitware

Python Wrapping Infrastructure

- Two cases:
  - VTK based objects: Leverage VTK infrastructure
  - Qt based objects: Leverage PythonQt, CTK and Qt Meta-Object System
- PythonQt != PyQt != PySide
- References:
  - http://pythonqt.sourceforge.net/
  - https://github.com/Kitware/VTK/tree/master/Wrapping/Python
  - https://www.vtk.org/Wiki/VTK/Python_Wrapper_Enhancement
  - https://www.vtk.org/Wiki/VTK/Python_Wrapping_FAQ

Kitware

Slicer and the Cloud

Slicer Web Services Integration

- Support integration with web services
  - Using commontk/qRestAPI: A Simple Qt library allowing to synchronously or asynchronously query a REST server.
  - Using QtNetwork API
  - Using python packages like urllib3, requests

Kitware

Slicer Hybrid Application

- Slicer embeds a web browser (based on Chromium in Qt5)
  - Extension Manager is a web page

Kitware

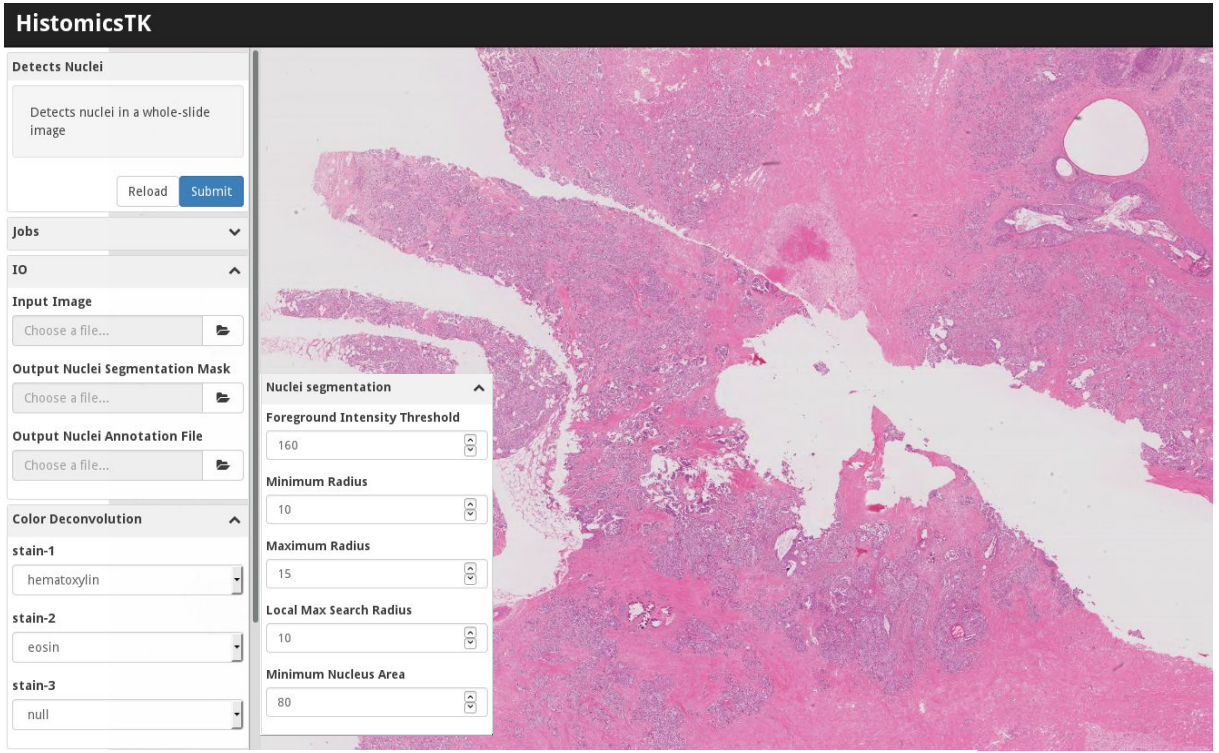
Slicer Docker Container: Use cases

- Use Cases:
  - Running Slicer from a web browser
    - https://github.com/thewtex/docker-opengl/
    - https://github.com/dit4c/dockerfile-dit4c-container-slicer , https://dit4c.github.io/
    - https://discourse.slicer.org/t/cloud-version-for-3d-slicer/860
  - Web application with distributed processing
    - HistomicsTK + Slicer CLI: https://digitalslidearchive.github.io/HistomicsTK/
  - Continuous Integration:
    - https://github.com/thewtex/SlicerDocker
    - https://github.com/Slicer/SlicerBuildEnvironment
    - https://blog.kitware.com/3d-slicer-improves-testing-for-pull-requests-using-docker-and-circleci/
  - Deep Learning
    - http://www.deepinfer.org/

Kitware

SEM + Cloud: Use case: HistomicsTK

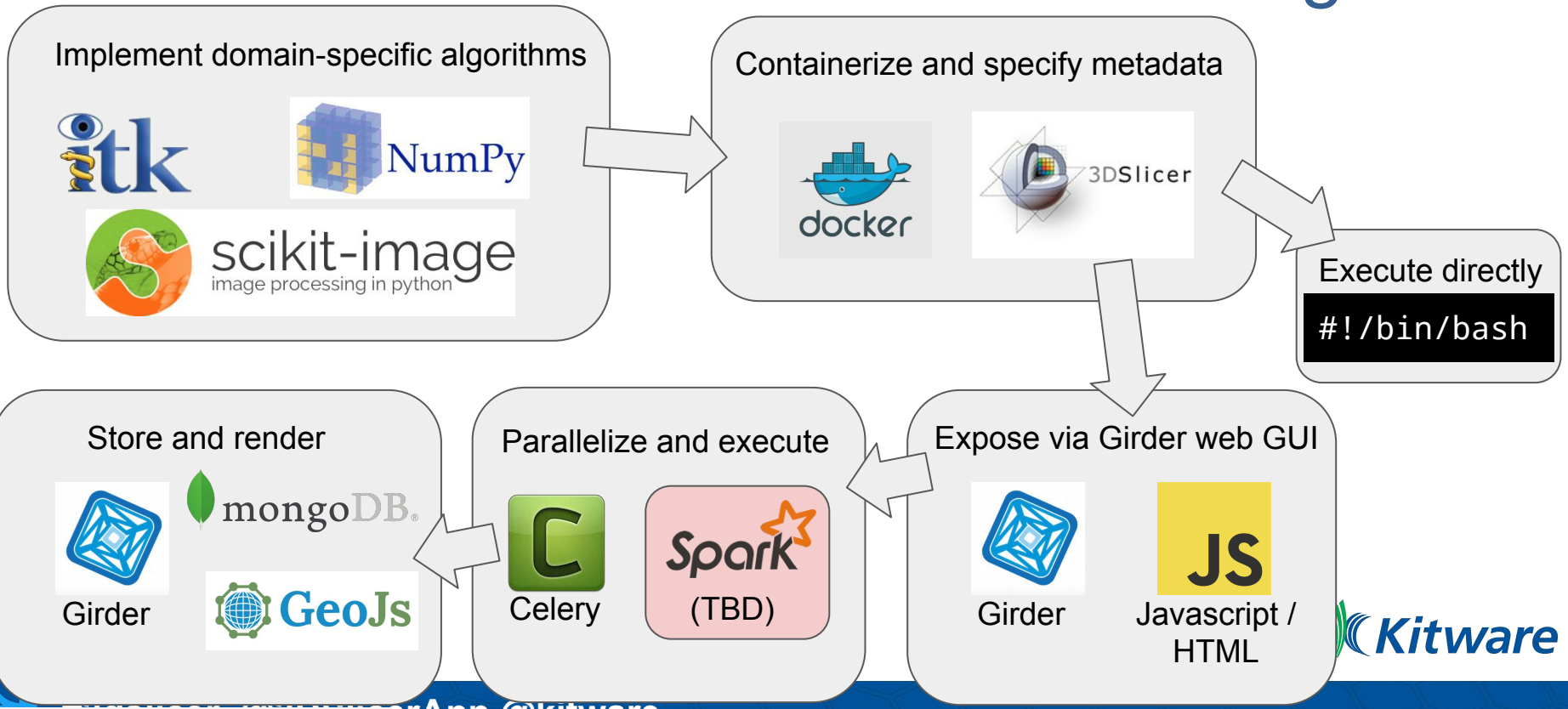
```
    <element>hematoxylin</element>
    <element>eosin</element>
    <element>dab</element>
    <default>eosin</default>
  </string-enumeration>
  <string-enumeration>
    <name>stain_3</name>
    <label>stain-3</label>
    <description>Name of stain-3</description>
    <channel>input</channel>
    <longflag>stain_3</longflag>
    <element>hematoxylin</element>
    <element>eosin</element>
    <element>dab</element>
    <element>null</element>
    <default>null</default>
  </string-enumeration>
</parameters>
<parameters>
  <label>Nuclei segmentation</label>
  <description>Nuclei segmentation parameters</description>
  <double>
    <name>foreground_threshold</name>
    <label>Foreground Intensity Threshold</label>
    <description>Intensity value to use as threshold to segm
    <longflag>foreground_threshold</longflag>
    <default>160</default>
  </double>
  <double>
    <name>min_radius</name>
    <label>Minimum Radius</label>
```



https://www.slicer.org/wiki/Documentation/Nightly/Developers/SlicerExecutionModel
https://digitalslidearchive.github.io/HistomicsTK/

Slide courtesy of Deepak Chittajallu, Kitware Inc.

HistomicsTK: Workflow and Technologies



Slide courtesy of Deepak Chittajallu, Kitware Inc.

Thanks

jfillionr

jcfr

thehiddenbrain

Kitware

Other Topics

- Application Settings
- Plots
- Debugging
- Slicer Orientation Presets