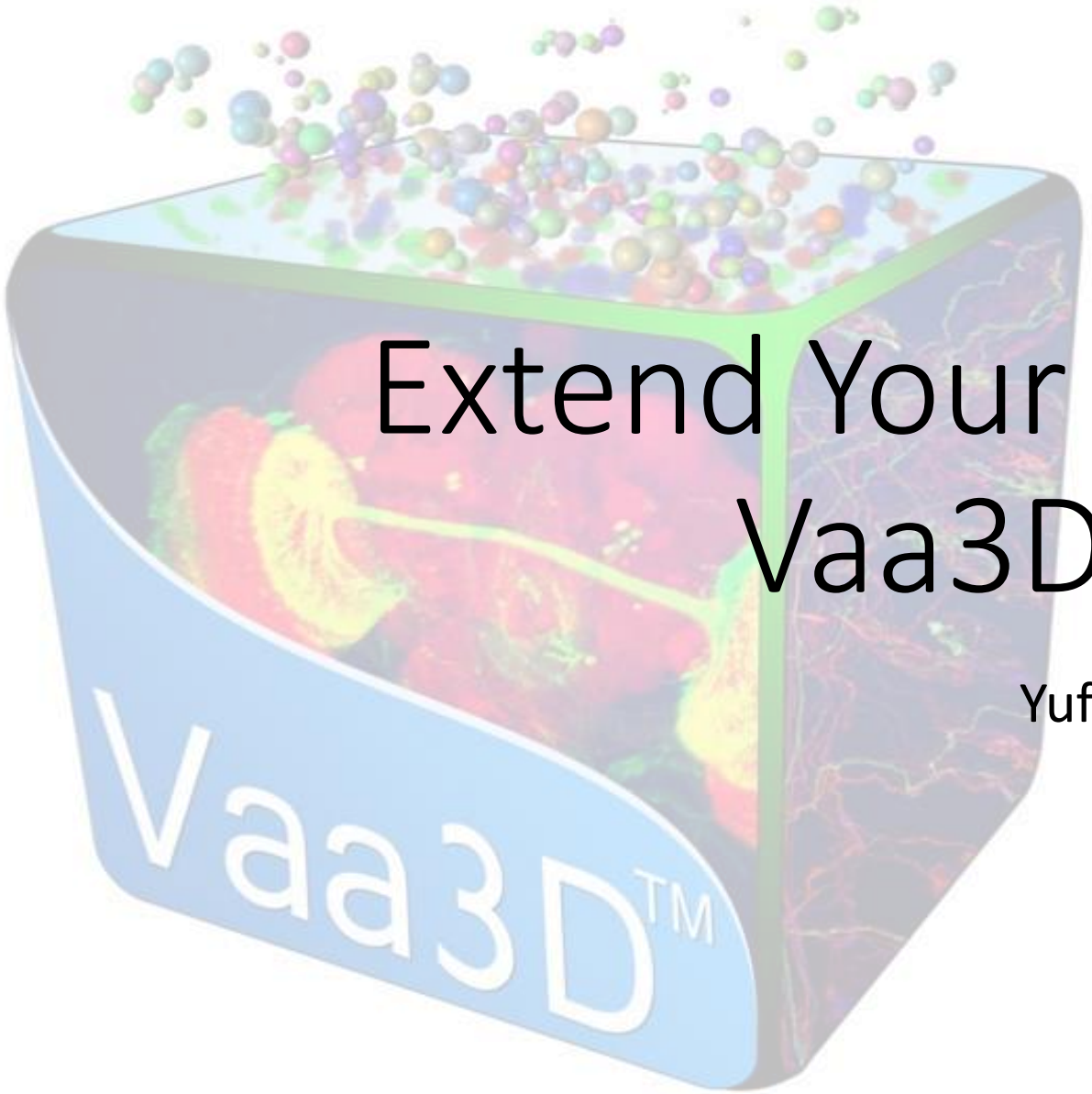
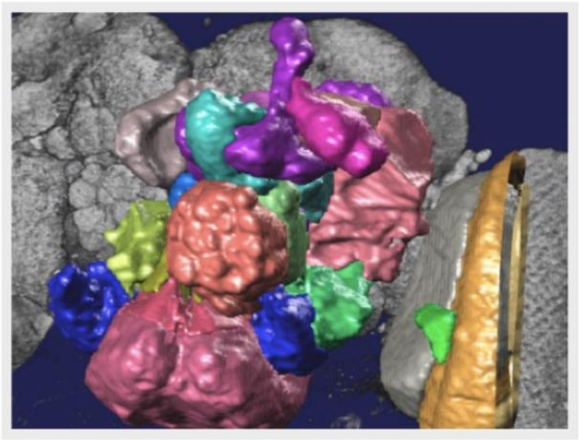
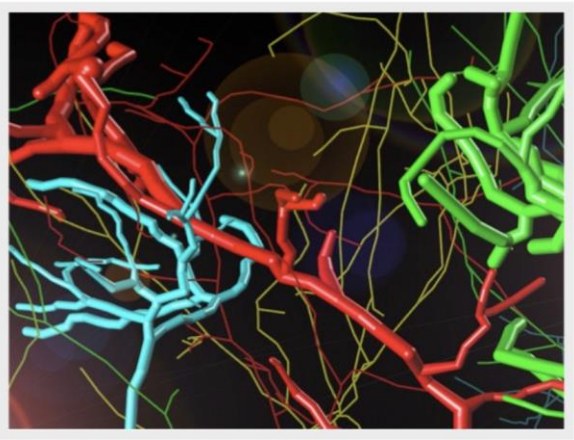


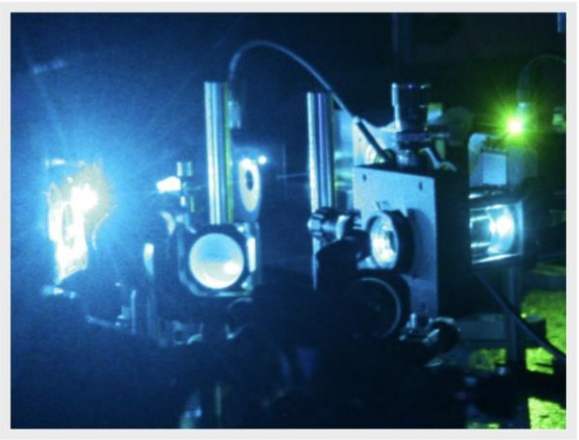
Extend Your Research with Vaa3D Plugins

Yufeng Liu

Vaa3D: A Swiss Army knife for exploring big big image data







FAST

Vaa3D visualizes and explores big 3D/4D/5D images with giga-voxels and even tera-voxels, within seconds or sub-seconds!

COOL

Vaa3D extracts complex surface objects from images, and performs comprehensive analyses such as brain connectome mapping.

EXTENSIBLE

100+ plugins for image acquisition, microsurgery, data management and analysis, and massive-scale pipelining.

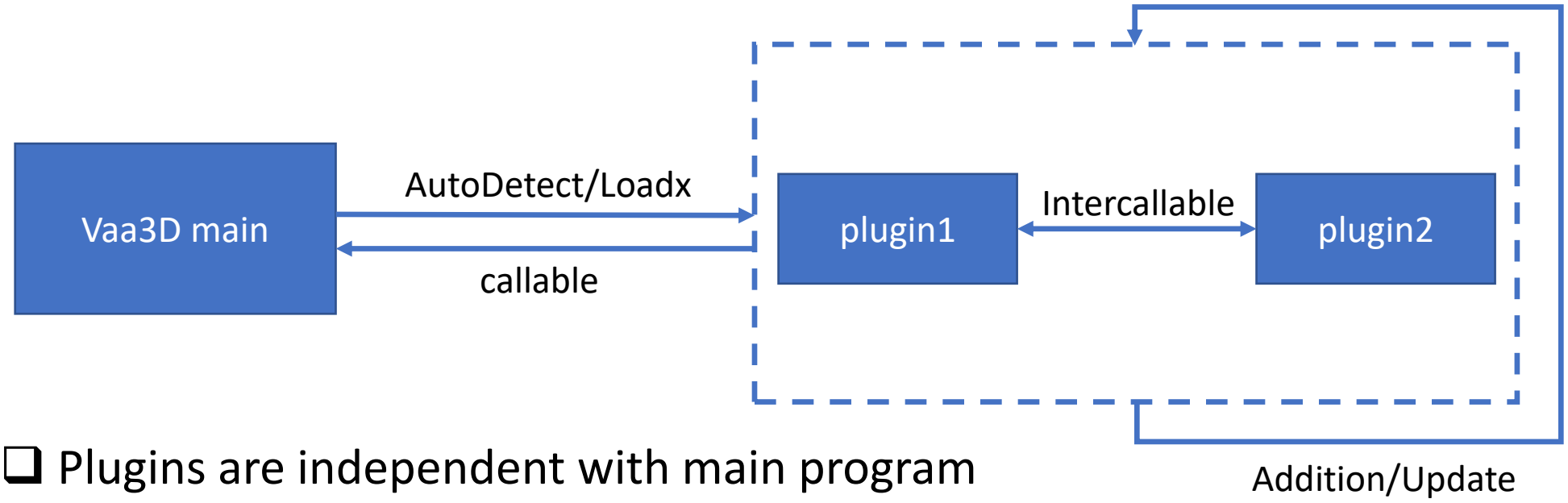
V3D enables real-time 3D visualization and quantitative analysis of large-scale biological image data sets

Hanchuan Peng*, **Zongcai Ruan**, **Fuhui Long**, **Julie H. Simpson**, and **Eugene W. Myers**

Extensibility: Plugins

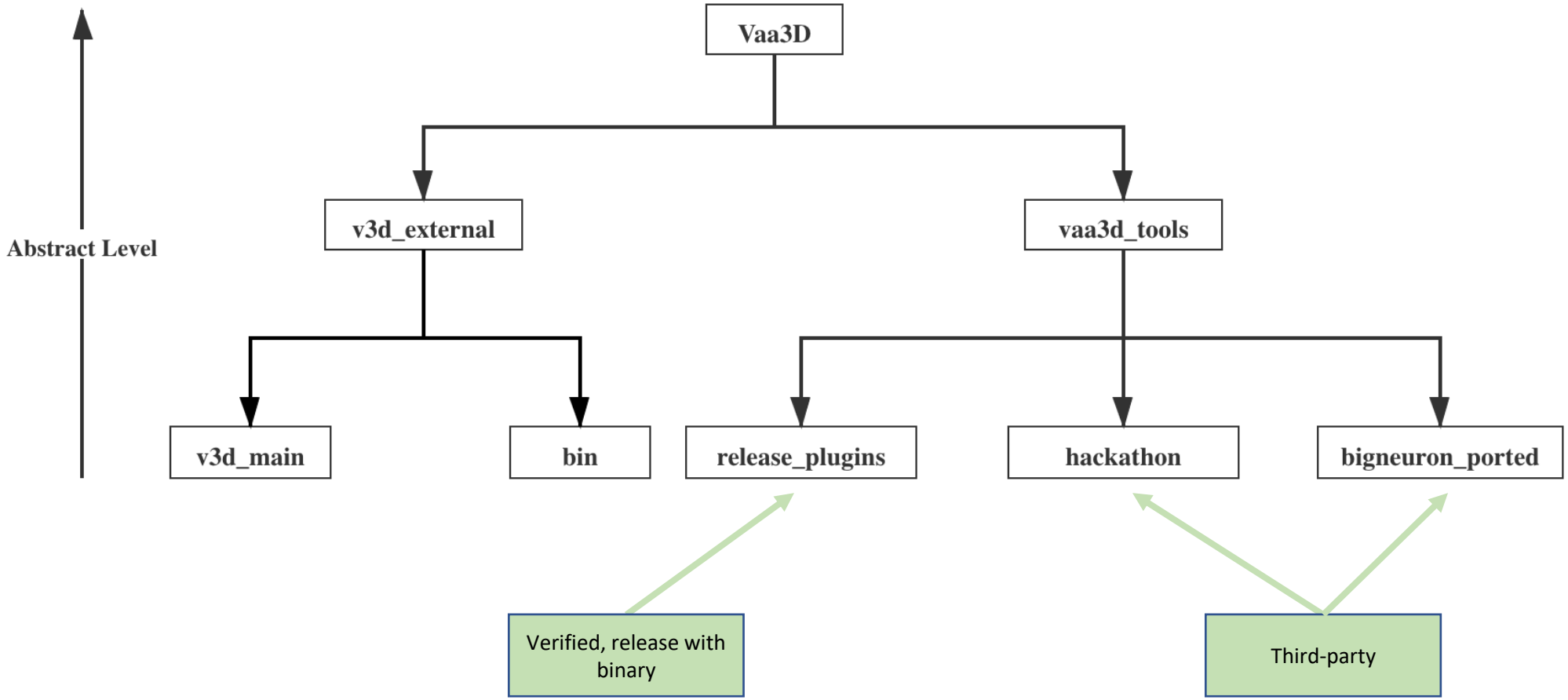
- Dynamic libraries on top of Vaa3D that empower the main program with specific functions
  - make use of Vaa3D core functions
  - call other plugin functions: super-plugin
- Two-folds
  - Large number of available plugins
  - Plugin implementation required only minimal effort

High Extensibility with Minimal Effort



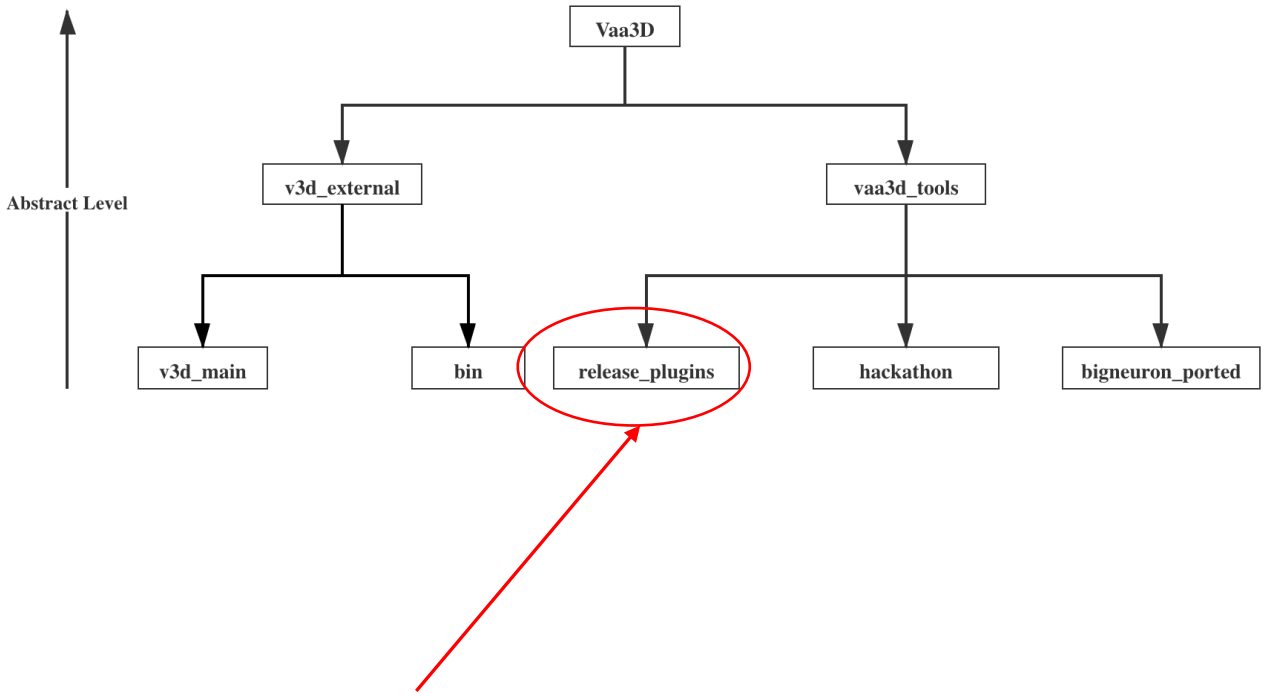
- Plugins are independent with main program
  - Interact at dynamic library level (*.so/*.dll/*.dylib)
  - Addition/update of plugin does not require re-compiling of main program
- Vaa3D provide facilities that
  - automatically detect, load, and call plugins
  - reserves extensible interface for new plugins

Vaa3D architecture



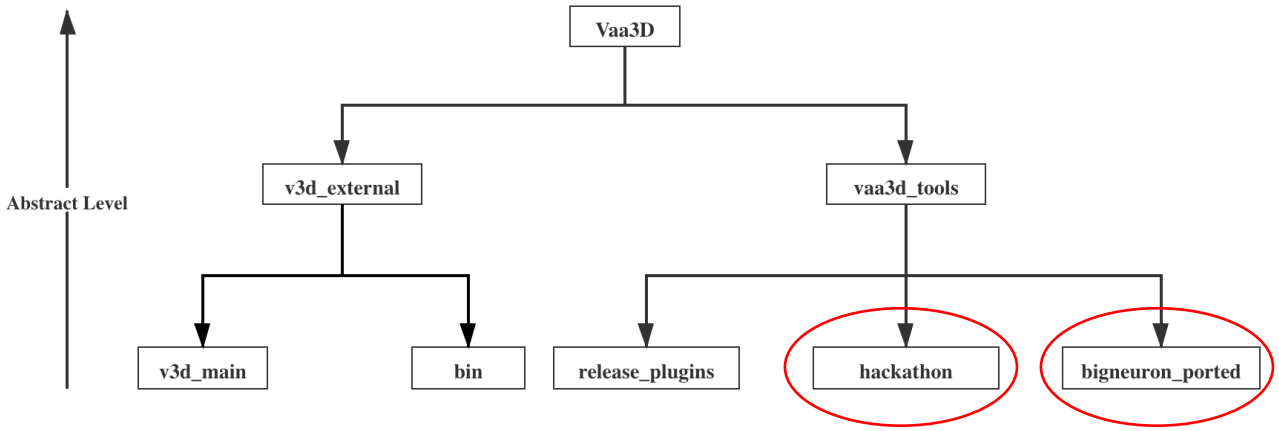
released_plugins: Built-in plugins

- ❑ 140+
- ❑ pre-built with binary, automatic built while compiling
- ❑ binaries located at `*bin*`
- ❑ Diverse functionalities:
  - ❑ 27 automatic tracing algorithms
  - ❑ Neuron analysis, resampling
  - ❑ Image analysis, transformation, filtering, visualization
  - ❑ Registration, stitching
  - ❑ File IO and conversion
  - ❑ …

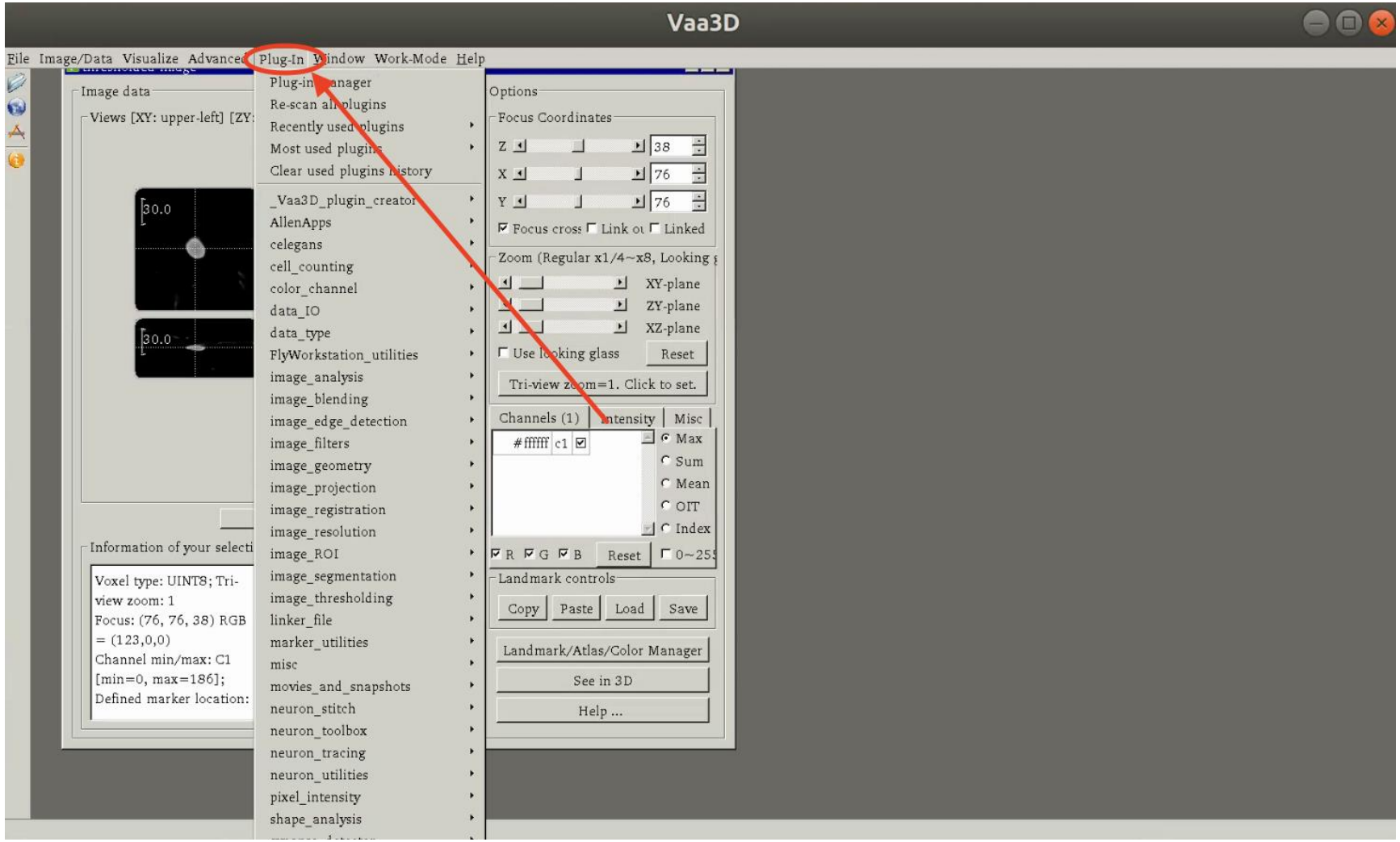


Third-party plugins

- Hackathon & BigNeuron_ported
  - ~400 plugins!!!
  - Compile manually
  - Users are encouraged to implement their plugins here.



Usage of plugins: through main menu



Shortcut configuring

- Append the most frequently used plugins to the main menu

Usage of plugins: through command line

- ❑ large scale data
- ❑ Additional configurable parameters for some plugins
- ❑ Better exception control
- ❑ Speed up via parallelization

Usage of plugins: through command line

Full list of plugins:

```
vaa3d -h (for Mac OS and Linux)
vaa3d_msvc.exe /h (for Windows)
```

Arguments form is different for Windows

Help information of a specific plugin:

```
vaa3d -h -x <plugin_name> (for Mac OS and Linux)
vaa3d_msvc.exe /h /x <plugin_name> (for Linux)
```

Usage of plugins: through command line

In Linux shell:

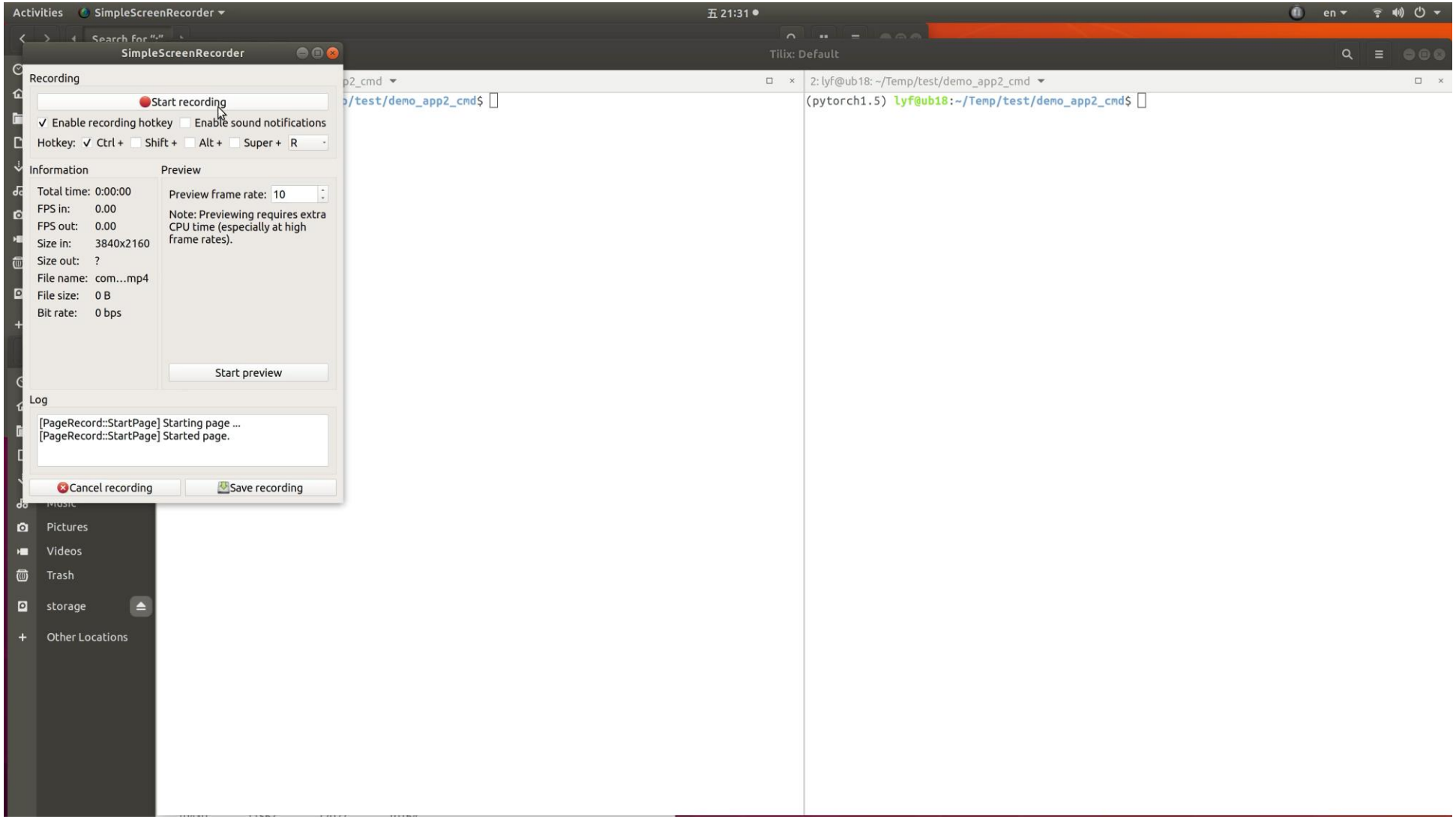
```
vaa3d -h -x <plugin_name> #find out the usage
vaa3d -x <app2_so_path> -f app2 -i <input_image> \
    -o <output_image> -p <marker_file> 0 AUTO 0 \
    # execute APP2 in auto mode, with pre-defined soma location
```

Through other languages, *e.g.* python:

```
def exec_app2(input_image, output_image,
              vaa3d='<vaa3d_external>/bin/vaa3d',
              plugin='<vaa3d_external>/bin/plugins/neuron_tracing/Vaa3D_Neuron2/libvn2.so>',
              marker_file=<your_marker_file>):

    cmd_str = f'{vaa3d} -x {plugin} -f app2 -i
                {input_image} -o {output_image}
                -p {marker_file} 0 AUTO 0'
    try:
        p = subprocess.check_output(cmd_str, timeout=6000, shell=True)
    except subprocess.TimeoutExpired:
        print(f'Execution of image: {input_image} is too time-consuming. Skip!')
        p = ''
    return p
```

Usage of plugins: through command line



How to find out specific plugin you want?

- For built-in plugins:
  - GUI: click Plug-ins, drop down and find the specific plugins
  - Command line: vaa3d –h for information, or find plugin with specific name, try:
    - vaa3d –h | grep “keyword”
    - vaa3d –x “plugin_path” –f help
- For third-party plugins:
  - Go to the directory:
    - vaa3d_tools/hackathon & vaa3d_tools/bigneuron_ported
  - Search by keyword:
    - Find vaa3d_tools/ -name “*keyword*” –type f

Write your own plugin: pre-requisite

Environmental pre-requisite:

- Proper Qt version installed
- C++ compiler (e.g. g++)
- Vaa3D source code (not binary!) downloaded (http://vaa3d.org)

More informations:

- Supported versions refer to: https://github.com/Vaa3D/Vaa3D_Wiki/wiki

Write your own plugin: structure

Minimal file sets:

- plugin.h
- plugin.cpp
- plugin.pro

- Plugin creator:

- More informations:
  - Guidelines: https://github.com/Vaa3D/Vaa3D_Wiki/wiki/PluginDesignGuide.wiki

A Vaa3D Plugin contains essentially only FOUR simple interfacing functions

```
#ifndef __EXAMPLE_PLUGIN_H__
#define __EXAMPLE_PLUGIN_H__

#include <QtGui>
#include <v3d_interface.h>

class ExamplePlugin : public QObject, public V3DPluginInterface2_1
{
        Q_OBJECT
        Q_INTERFACES(V3DPluginInterface2_1);

public:
        float getPluginVersion() const {return 1.1f;}

        QStringList menulist() const;
        void domenu(const QString &menu_name,
                    V3DPluginCallback2 &callback,
                    QWidget *parent);

        QStringList funclist() const ;
        bool dofunc(const QString &func_name,
                    const V3DPluginArgList &input,
                    V3DPluginArgList &output,
                    V3DPluginCallback2 &callback,
                    QWidget *parent);
};

#endif
```

Menu items in GUI

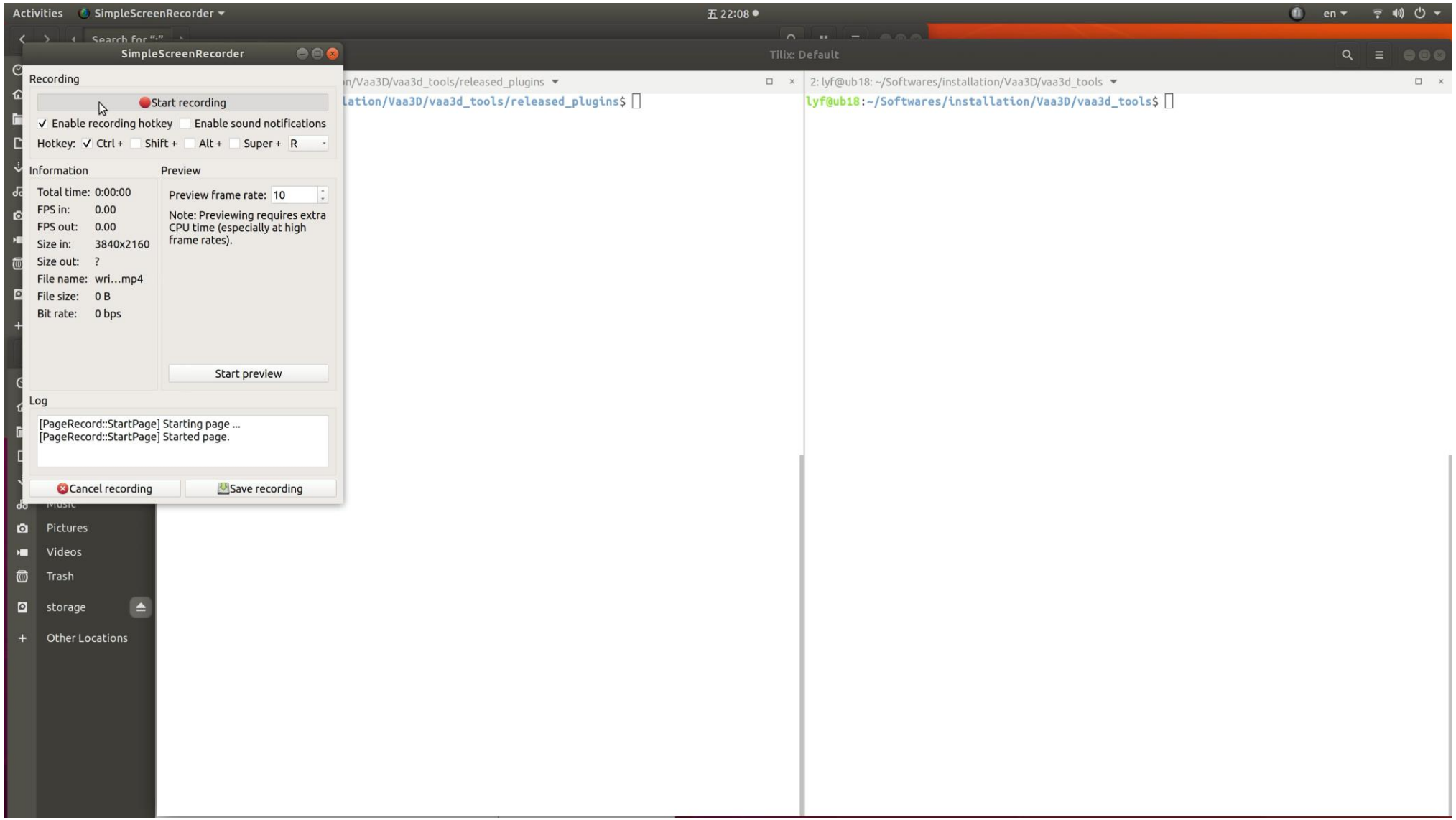
The actual action(s) of each menu item

Function items for any other purposes

The actual action(s) of each function

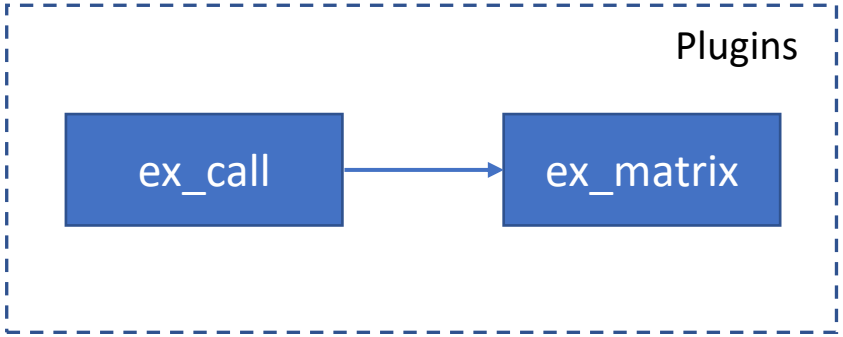
Slide from Dr. Peng

Write your own plugin: an example



Super-plugin: how to call other plugins

- Example: call externel plugin ex_matrix in current plugin (ex_call).



- Usage:
  - v3d.callPluginFunc(ex_matrix_path, func_name)

- Reference code: v3d_plugins/v3dplugin_call_each_other_example

attendees practice

- APP2 tracing:
  - Through command line
- Writing example plugin: “Hello world”
  - Writing & Compiling
  - Validation