
# User Interface Programming

13 November 2017

# Who are we?

# Who are we?

## Paul Chana

Lead Software Developer ROLI

# Who are we?

## Paul Chana

Lead Software Developer ROLI

## Vlad Voina

Freelance Software Engineer

# A few things to note…

# A few things to note…

- Slides are numbered

# A few things to note…

- Slides are numbered
- Please ask questions!

# A few things to note…

- Slides are numbered
- Please ask questions!
- Code was tweaked to fit the slides…

# A few things to note…

- Slides are numbered
- Please ask questions!
- Code was tweaked to fit the slides…
- Where can you download the source code and keynote from?

# A few things to note…

- Slides are numbered
- Please ask questions!
- Code was tweaked to fit the slides…
- Where can you download the source code and keynote from?
  - https://www.dropbox.com/s/5qznoknskl2zpzk/ADC2017_UI_Workshop.zip?dl=0

# Topics we will focus on today

- Quick overview of Model-View-Controller (MVC) Architecture

# Topics we will focus on today

- Quick overview of Model-View-Controller (MVC) Architecture
- AudioProcessorValueTreeState

# Topics we will focus on today

- Quick overview of Model-View-Controller (MVC) Architecture
- AudioProcessorValueTreeState
- Responsive layout with Flexbox and Grid

# Topics we will focus on today

- Quick overview of Model-View-Controller (MVC) Architecture
- AudioProcessorValueTreeState
- Responsive layout with Flexbox and Grid
- Graphical assets

# Topics we will focus on today

- Quick overview of Model-View-Controller (MVC) Architecture
- AudioProcessorValueTreeState
- Responsive layout with Flexbox and Grid
- Graphical assets
- Cross thread communication (if we have time!)

# Model-View-Controller Architecture

# What is MVC?

Divides the application in to 3 conceptual parts:

# What is MVC?

Divides the application in to 3 conceptual parts:

- **Model.** Expresses the application behaviour independently of the UI

# What is MVC?

Divides the application in to 3 conceptual parts:

- **Model.** Expresses the application behaviour independently of the UI
- **View.** Any output for the information from the model

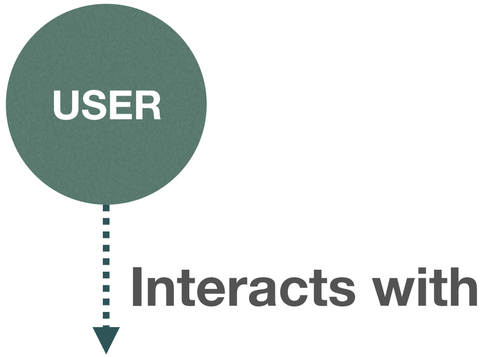
# What is MVC?

Divides the application in to 3 conceptual parts:

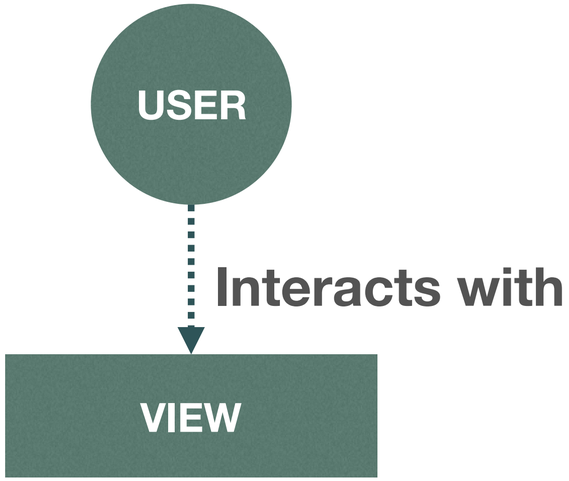
- **Model.** Expresses the application behaviour independently of the UI
- **View.** Any output for the information from the model
- **Controller.** Connects the model and the view, converting inputs and outputs to the correct format

# Diagrams make things easier

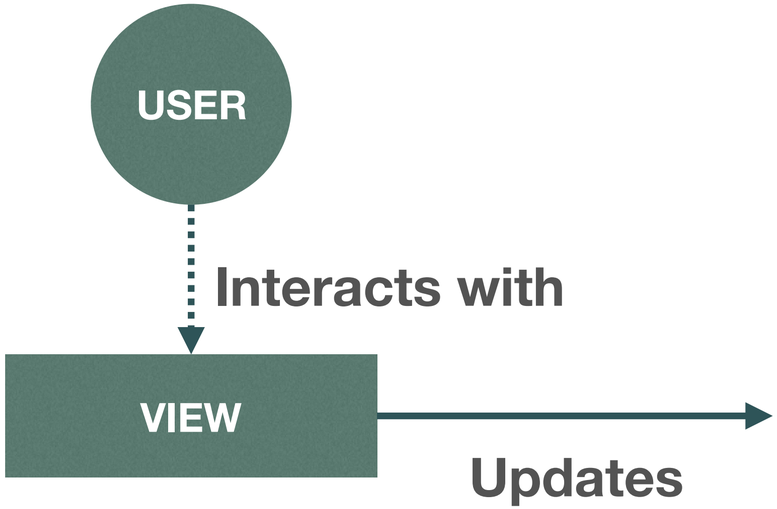
# Diagrams make things easier



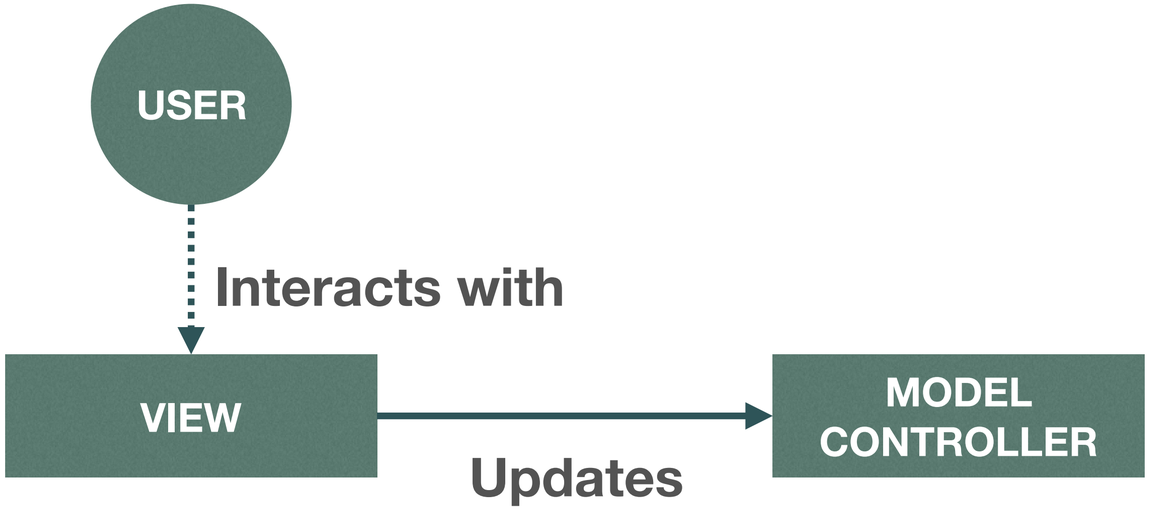
# Diagrams make things easier



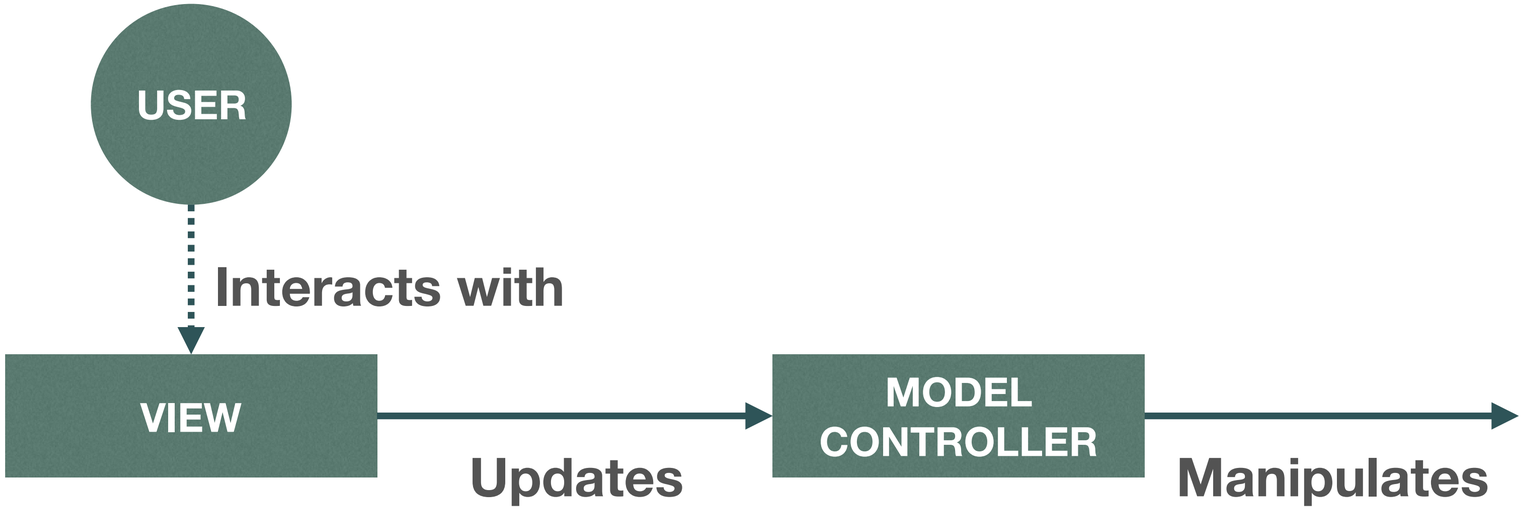
# Diagrams make things easier



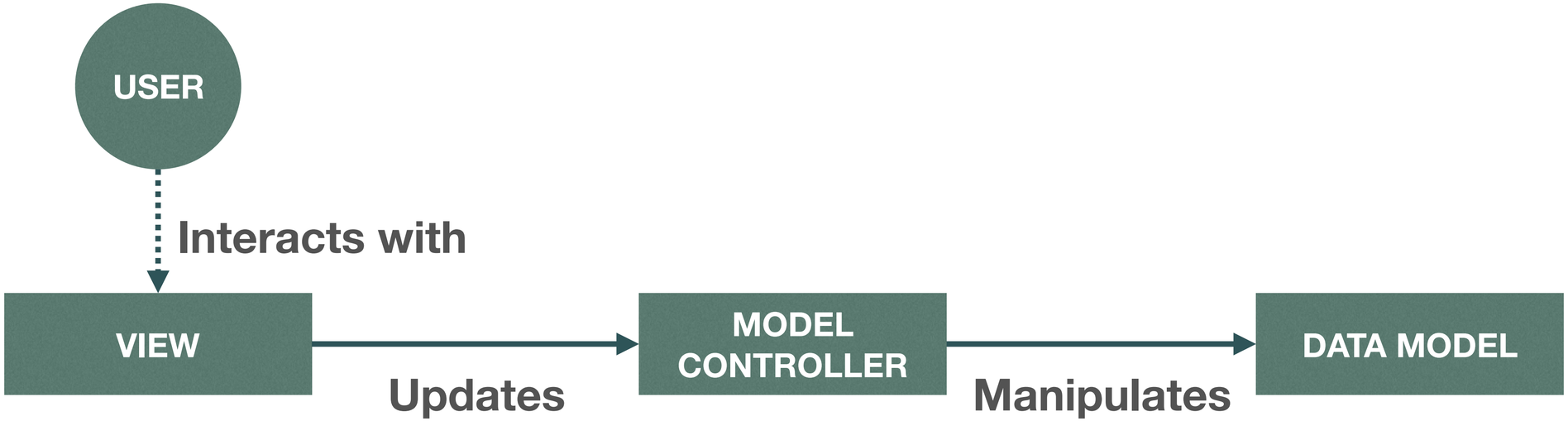
# Diagrams make things easier



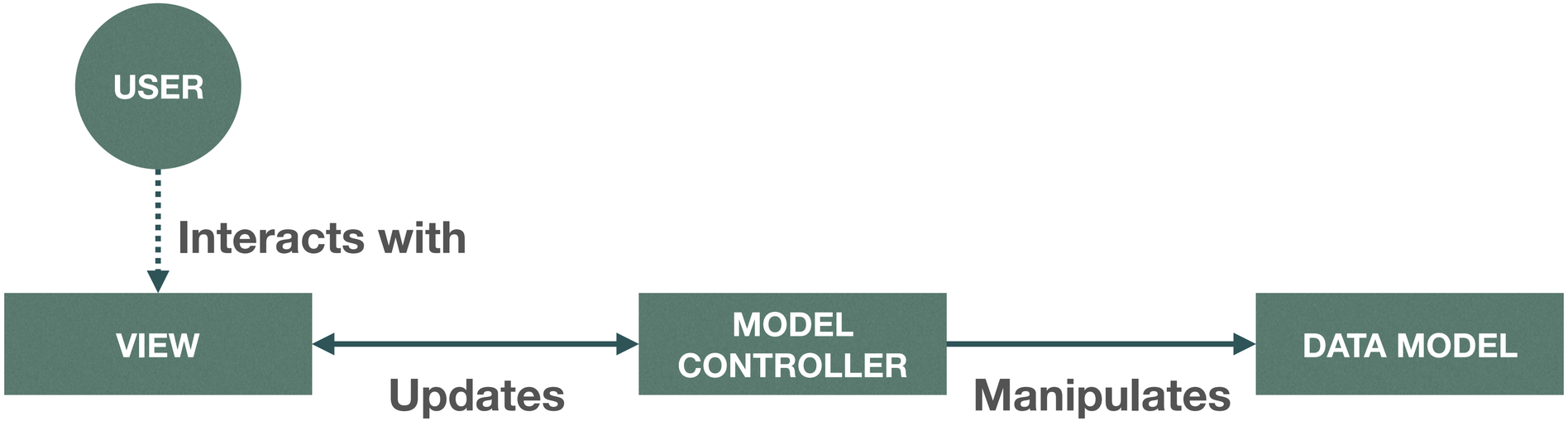
# Diagrams make things easier



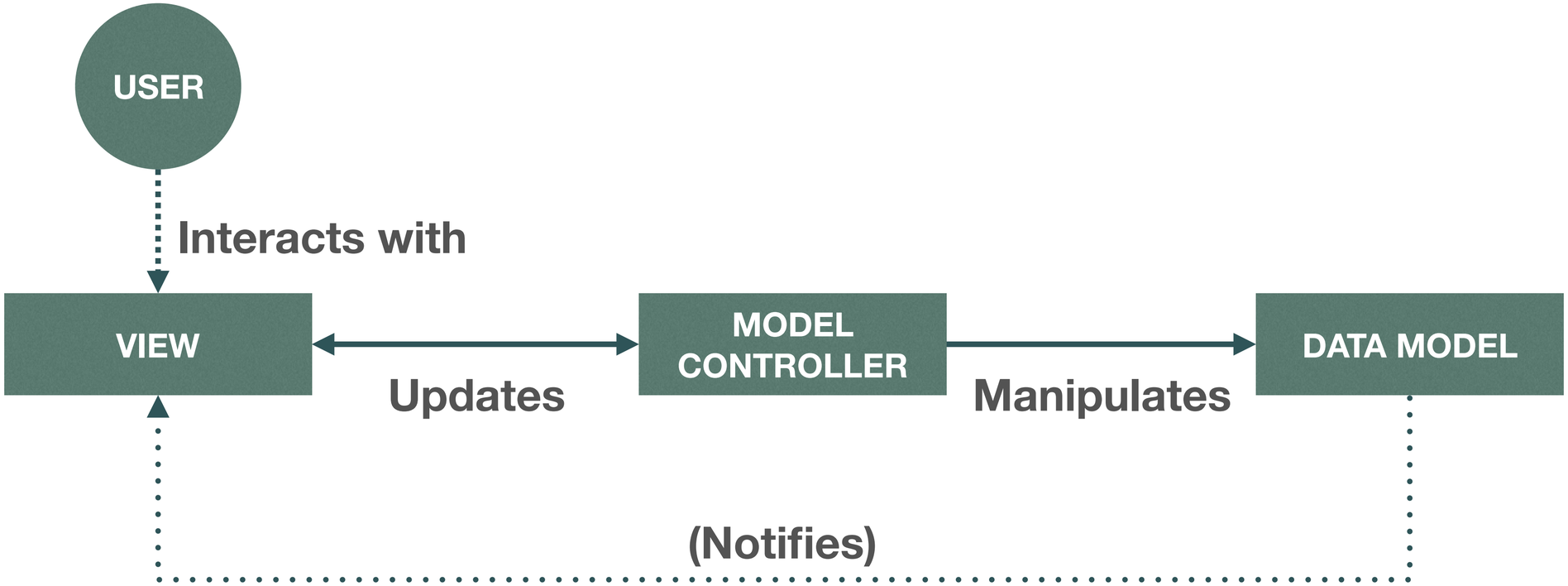
# Diagrams make things easier



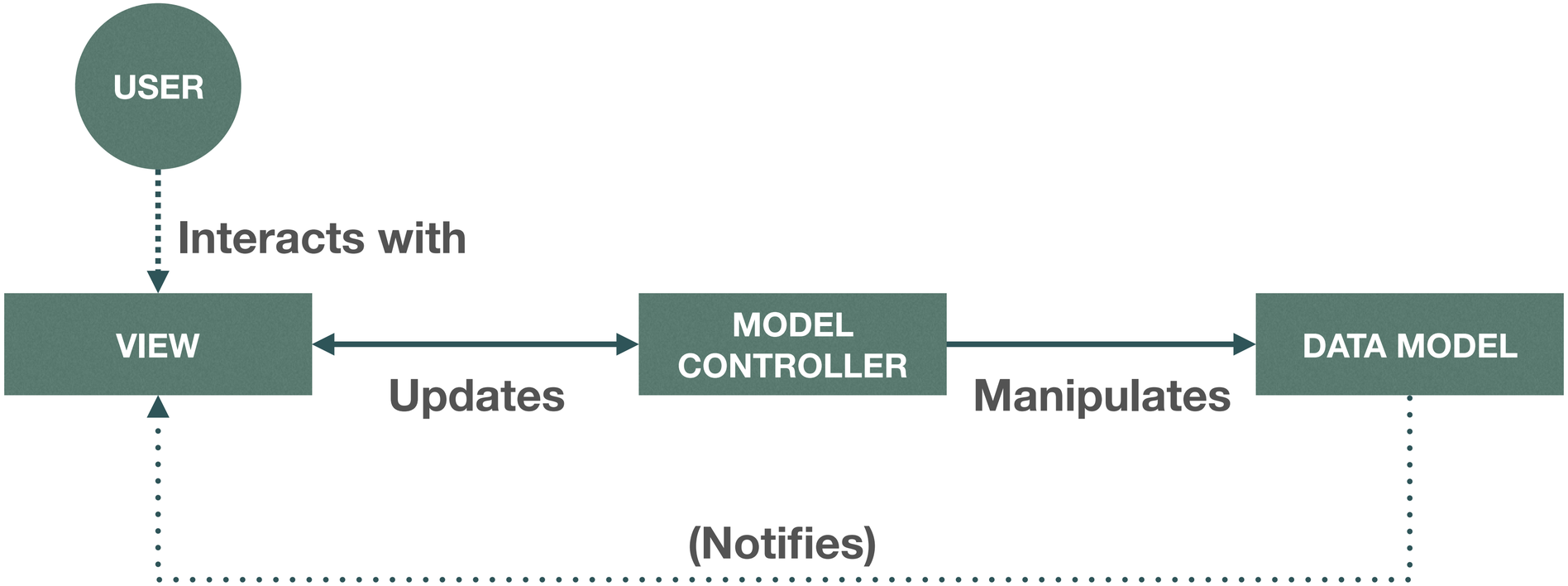
# Diagrams make things easier



# Diagrams make things easier

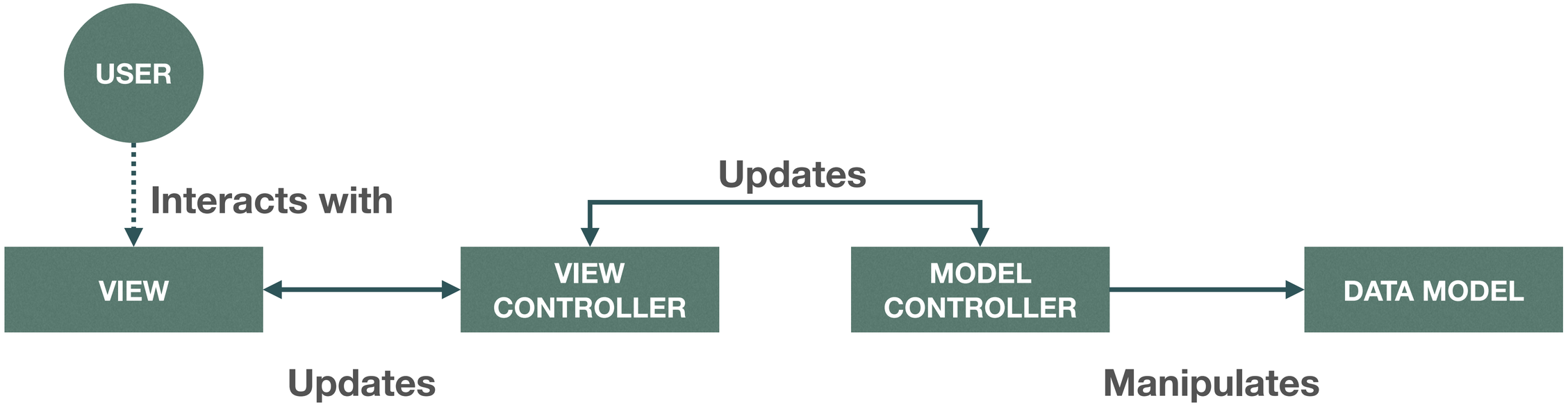


# Diagrams make things easier

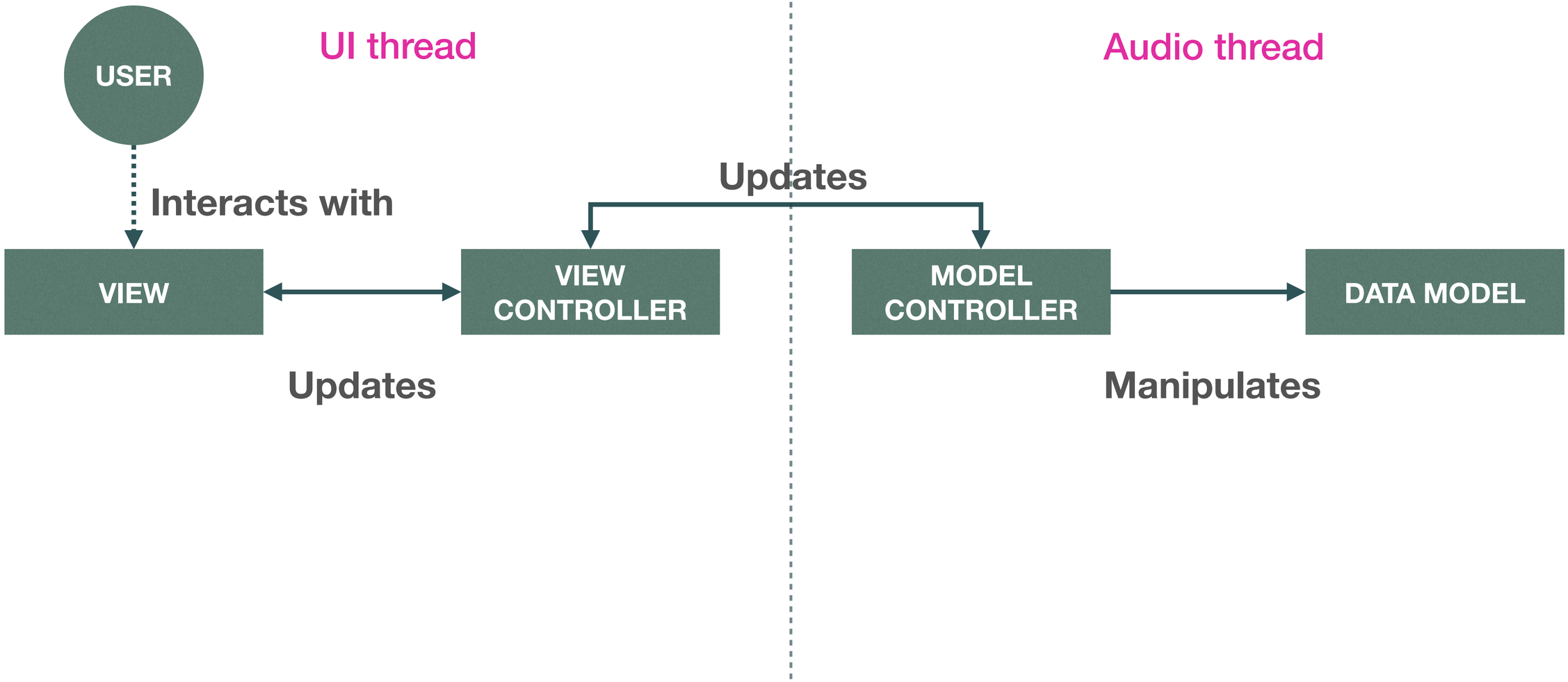



Note that this is *not* a class diagram

# Multiple controllers



# Threading model for audio plugins



# Why use MVC?

- Allows us to separate ***business logic*** from display state

# Why use MVC?

- Allows us to separate ***business logic*** from display state
- Allows us to present multiple views on the same data, without changing that data

# Why use MVC?

- Allows us to separate ***business logic*** from display state
- Allows us to present multiple views on the same data, without changing that data
- The controller can sandbox the model

# Why use MVC?

- Allows us to separate ***business logic*** from display state
- Allows us to present multiple views on the same data, without changing that data
- The controller can sandbox the model
- Allows us to trivially divide the view from the back-end

# AudioProcessorValueTreeState

# Some terminology for those not familiar with Juce…

Some terminology for those not familiar with Juce…

- var is a union of multiple types (int, float, bool, String etc)

Some terminology for those not familiar with Juce…

- var is a union of multiple types (int, float, bool, String etc)
- Component is a UI widget

Some terminology for those not familiar with Juce…

- var is a union of multiple types (int, float, bool, String etc)
- Component is a UI widget
- AudioProcessor is a wrapper around a VST / AU / AAX etc

# MVCFilter example

# MVCFilter example

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterKeys[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    var parameters[numberOfParameters];
};
```

# MVCFilter example

- Our parameters

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterKeys[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    var parameters[numberOfParameters];
};
```

# MVCFilter example

- Our parameters
- Some boilerplate to support conversion to and from parameters

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterKeys[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    var parameters[numberOfParameters];
};
```

# MVCFilter example

- Our parameters
- Some boilerplate to support conversion to and from parameters
- Listener callback functions

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterKeys[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    var parameters[numberOfParameters];
};
```

# MVCFilter example

- Our parameters
- Some boilerplate to support conversion to and from parameters
- Listener callback functions
- Code to save and restore the plugin

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterKeys[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    var parameters[numberOfParameters];
};
```

# MVCFilter example

- Our parameters
- Some boilerplate to support conversion to and from parameters
- Listener callback functions
- Code to save and restore the plugin
- Code to connect editor and controller

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterKeys[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    var parameters[numberOfParameters];
};
```

# How is the editor connected to the controller?

# How is the editor connected to the controller?

```
void Controller::bindEditor (AudioProcessorEditor& theEditor)
{
    if (auto* editor = dynamic_cast<PluginView*>(&theEditor))
        editor->cutoff.addListener (this);
}
```

```
void Controller::sliderValueChanged (Slider* slider)
{
    if (slider->getComponentID() == parameterNames [int (Parameter::cutoff)])
        setParameter (Parameter::cutoff, Filter::getFrequencyRange().convertFrom0to1 (float (slider->getValue())));
}
```

# Let's remove some boilerplate!

## What is a ValueTree?

# Let's remove some boilerplate!

## What is a ValueTree?

- A powerful tree structure that can be used to hold free-form data, and which can handle its own undo and redo behaviour.

# Let's remove some boilerplate!

## What is a ValueTree?

- A powerful tree structure that can be used to hold free-form data, and which can handle its own undo and redo behaviour.
- A ValueTree contains a list of named properties as var objects, and also holds any number of sub-trees.

# Let's remove some boilerplate!

What is a ValueTree?

- A powerful tree structure that can be used to hold free-form data, and which can handle its own undo and redo behaviour.
- A ValueTree contains a list of named properties as var objects, and also holds any number of sub-trees.

For a deeper dive on ValueTrees visit ALT/TAB at 10:40 tomorrow and listen to Dave Rowland!

# Let's remove some boilerplate

What is a AudioProcessorValueTreeState?

# Let's remove some boilerplate

What is a AudioProcessorValueTreeState?

- A class which contains a ValueTree which is used to manage an AudioProcessor's entire state

# Let's remove some boilerplate

What is a AudioProcessorValueTreeState?

- A class which contains a ValueTree which is used to manage an AudioProcessor's entire state
- Has an internal timer for updating Components that you connect to it

# Let's remove some boilerplate

What is a AudioProcessorValueTreeState?

- A class which contains a ValueTree which is used to manage an AudioProcessor's entire state
- Has an internal timer for updating Components that you connect to it
- Provides simple serialisation methods

# Updating the controller

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    var parameters[numberOfParameters];
};
```

# Updating the controller

- Replace the parameters

```
class Controller : private Slider::Listener, private ComboBox::Listener, private Button::Listener
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    void updateControlValues (AudioProcessorEditor& theEditor);
    void sliderValueChanged (Slider* slider) override;
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged) override;
    void buttonClicked (Button*) override;

    Filter filter[2];
    Distortion distortion[2];

    AudioProcessorValueTreeState parameters;
};
```

# Updating the controller

- Replace the parameters
- Remove listener boilerplate

```
class Controller
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    void setParameter (Parameter param, var value);
    void setParameterWithStringValue (Parameter param, String value);
    var getParameter (Parameter param) const;
    String getParameterValue (Parameter param) const;
    void resetParameters();

    Filter filter[2];
    Distortion distortion[2];

    AudioProcessorValueTreeState parameters;
};
```

# Updating the controller

- Replace the parameters
- Remove listener boilerplate
- Remove the parameter boilerplate

```
class Controller
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);
    void unbindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    Filter filter[2];
    Distortion distortion[2];

    AudioProcessorValueTreeState parameters;
};
```

# Updating the controller

- Replace the parameters
- Remove listener boilerplate
- Remove the parameter boilerplate
- Remove editor boilerplate

```
class Controller
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();

    Filter filter[2];
    Distortion distortion[2];

    AudioProcessorValueTreeState parameters;
};
```

# Updating the controller

## Final version!

```
class Controller
{
public:

    enum class Parameter { drive = 0, distortionMode, cutoff, resonance, filterMode, distortionIsPreFilter };

    Controller();

    void bindEditor (AudioProcessorEditor& editor);

    String getParameterInfo (Parameter param) const;

    void setSampleRate (float fs);

    XmlElement saveState() const;
    void restoreState (XmlElement& xml, AudioProcessor& audioProcessor);

    void process (AudioSampleBuffer& buffer);

private:

    static constexpr int numberOfParameters = int (Parameter::distortionIsPreFilter) + 1;
    static constexpr const char* parameterKeys[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterNames[numberOfParameters] = {/* Removed for clarity */};
    static constexpr const char* parameterDescriptions[numberOfParameters] = {/* Removed for clarity */};

    void prepareForProcess();
    void constructParameters();

    Filter filter[2];
    Distortion distortion[2];

    AudioProcessorValueTreeState parameters;
};
```

# How did we define a parameter?

```
void Controller::setParameter (Parameter param, var value)
{
    parameters[int (param)] = value;
}

var Controller::getParameter (Parameter param) const
{
    return parameters[int (param)];
}

void Controller::resetParameters()
{
    for (int param = 0; param < numberOfParameters; ++param)
        setParameter (Parameter (param), defaultParameters[param]);
}

void Controller::setParameterWithStringValue (Parameter param, String value)
{
    if (param == Parameter::drive || param == Parameter::resonance)
        setParameter (param, value.replace (" %", {}).getFloatValue() / 100.0f);

    // <SNIP> Other parameters here...
}

String Controller::getParameterValue (Parameter param) const
{
    if (param == Parameter::drive || param == Parameter::resonance)
        return String (roundToInt (float (parameters[int (param)]) * 100.0f)) + "%";

    // <SNIP> Other parameters here...

    jassertfalse;
    return {};
}
```

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

Key to identify the parameter

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

Key to identify the parameter

Name

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

Key to identify the parameter

Name

Label

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

Key to identify the parameter
Name
Label
Range

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

Key to identify the parameter
Name
Label
Range
Default value

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

- Key to identify the parameter
- Name
- Label
- Range
- Default value
- Value to string mapper

# How to define a parameter?

```
parameters.createAndAddParameter (parameterKeys[int (Parameter::drive)],
                                  parameterNames[int (Parameter::drive)],
                                  "%",
                                  NormalisableRange<float> (0.0f, 1.0f),
                                  0.25f,
                                  [] (float value) -> juce::String
                                  {
                                      return juce::String (roundToInt (value * 100.0f));
                                  },
                                  [] (const juce::String& value) -> float
                                  {
                                      return float (value.getIntValue()) / 100.0f;
                                  });
```

- Key to identify the parameter
- Name
- Label
- Range
- Default value
- Value to string mapper
- String to value mapper

# How to connect controller and view?

```
void Controller::bindEditor (AudioProcessorEditor& theEditor)
{
    if (auto* editor = dynamic_cast<PluginView*>(&theEditor))
    {
        using APVTS = AudioProcessorValueTreeState;

        editor->sliders.add (new APVTS::SliderAttachment (parameters,
                                                          parameterKeys[int (Parameter::cutoff)],
                                                          editor->cutoff));

        editor->combos.add (new APVTS::ComboBoxAttachment (parameters,
                                                           parameterKeys[int (Parameter::distortionMode)],
                                                           editor->distortionMode));

        editor->buttons.add (new APVTS::ButtonAttachment (parameters,
                                                          parameterKeys[int (Parameter::distortionIsPreFilter)],
                                                          editor->distortionIsPreFilter));
    }
}
```

# Updating save...

```
XmlElement Controller::saveState() const
{
    XmlElement xml (JucePlugin_Name);

    for (int param = 0; param < numberOfParameters; ++param)
        xml.setAttribute (parameterKeys[param]), getParameterValue (Parameter (param)));

    return xml;
}
```

# Updating save...

```
XmlElement Controller::saveState() const
{
    XmlElement xml (JucePlugin_Name);

    for (int param = 0; param < numberOfParameters; ++param)
        xml.setAttribute (parameterKeys[param]), getParameterValue (Parameter (param)));

    return xml;
}
```

```
XmlElement Controller::saveState() const
{
    if (ScopedPointer<XmlElement> parametersXml = parameters.state.createXml())
        return *parametersXml;

    jassertfalse;
    return XmlElement (JucePlugin_Name);
}
```

# Updating restore...

```
void Controller::restoreState (juce::XmlElement& xml, juce::AudioProcessor& audioProcessor)
{
    jassert (xml.hasTagName (JucePlugin_Name));
    if (xml.hasTagName (JucePlugin_Name))
    {
        for (int param = 0; param < numberOfParameters; ++param)
            setParameterWithStringValue (Parameter (param),
                                         xml.getStringAttribute (parameterKeys[param]));

        if (auto* editor = audioProcessor.getActiveEditor())
            updateControlValues (*editor);
    }
}
```

# Updating restore...

```
void Controller::restoreState (juce::XmlElement& xml, juce::AudioProcessor& audioProcessor)
{
    jassert (xml.hasTagName (JucePlugin_Name));
    if (xml.hasTagName (JucePlugin_Name))
    {
        for (int param = 0; param < numberOfParameters; ++param)
            setParameterWithStringValue (Parameter (param),
                                         xml.getStringAttribute (parameterKeys[param]));

        if (auto* editor = audioProcessor.getActiveEditor())
            updateControlValues (*editor);
    }
}
```

```
void Controller::restoreState (XmlElement& xml)
{
    jassert (xml.hasTagName (parameters.state.getType()));

    if (xml.hasTagName (parameters.state.getType()))
        parameters.state = ValueTree::fromXml (xml);
}
```

# Updating restore...

```
void Controller::restoreState (juce::XmlElement& xml, juce::AudioProcessor& audioProcessor)
{
    jassert (xml.hasTagName (JucePlugin_Name));
    if (xml.hasTagName (JucePlugin_Name))
    {
        for (int param = 0; param < numberOfParameters; ++param)
            setParameterWithStringValue (Parameter (param),
                                         xml.getStringAttribute (parameterKeys[param]));

        if (auto* editor = audioProcessor.getActiveEditor())
            updateControlValues (*editor);
    }
}
```

```
void Controller::restoreState (XmlElement& xml)
{
    jassert (xml.hasTagName (parameters.state.getType()));

    if (xml.hasTagName (parameters.state.getType()))
        parameters.state = ValueTree::fromXml (xml);
}
```

# Any other benefits?

Key benefits of working with the AudioProcessorValueTreeState

# Any other benefits?

Key benefits of working with the AudioProcessorValueTreeState

- 'Free' implementation of Plugin parameters (e.g. in your DAW)

# Any other benefits?

## Key benefits of working with the AudioProcessorValueTreeState

- 'Free' implementation of Plugin parameters (e.g. in your DAW)
  - Notice that MVCFilter doesn't have any parameters in a host. VTFilter does, but we have added no code to support them!

# Any other benefits?

## Key benefits of working with the AudioProcessorValueTreeState

- 'Free' implementation of Plugin parameters (e.g. in your DAW)
  - Notice that MVCFilter doesn't have any parameters in a host. VTFilter does, but we have added no code to support them!
- Removed boilerplate, which reduces the chance of errors

# Any other benefits?

## Key benefits of working with the AudioProcessorValueTreeState

- 'Free' implementation of Plugin parameters (e.g. in your DAW)
  - Notice that MVCFilter doesn't have any parameters in a host. VTFilter does, but we have added no code to support them!
- Removed boilerplate, which reduces the chance of errors
- Keeps the Application Logic contained within a few small functions in the Controller class

# Responsive layout

# Dynamic layout

- Components

# Dynamic layout

- Components
- Relative positions

# Dynamic layout

- Components
- Relative positions
- Proportional sizes

# Dynamic layout

- Components
- Relative positions
- Proportional sizes
- Main axis and cross axis

# Dynamic layout

- Components
- Relative positions
- Proportional sizes
- Main axis and cross axis
- Declarative

# Dynamic layout

- Components
- Relative positions
- Proportional sizes
- Main axis and cross axis
- Declarative
- Breakpoints

# Components

- Overuse components

# Components

- Overuse components
- Components are your bridge to responsive layout

# Components

- Overuse components
- Components are your bridge to responsive layout
- They modularise your layout logic

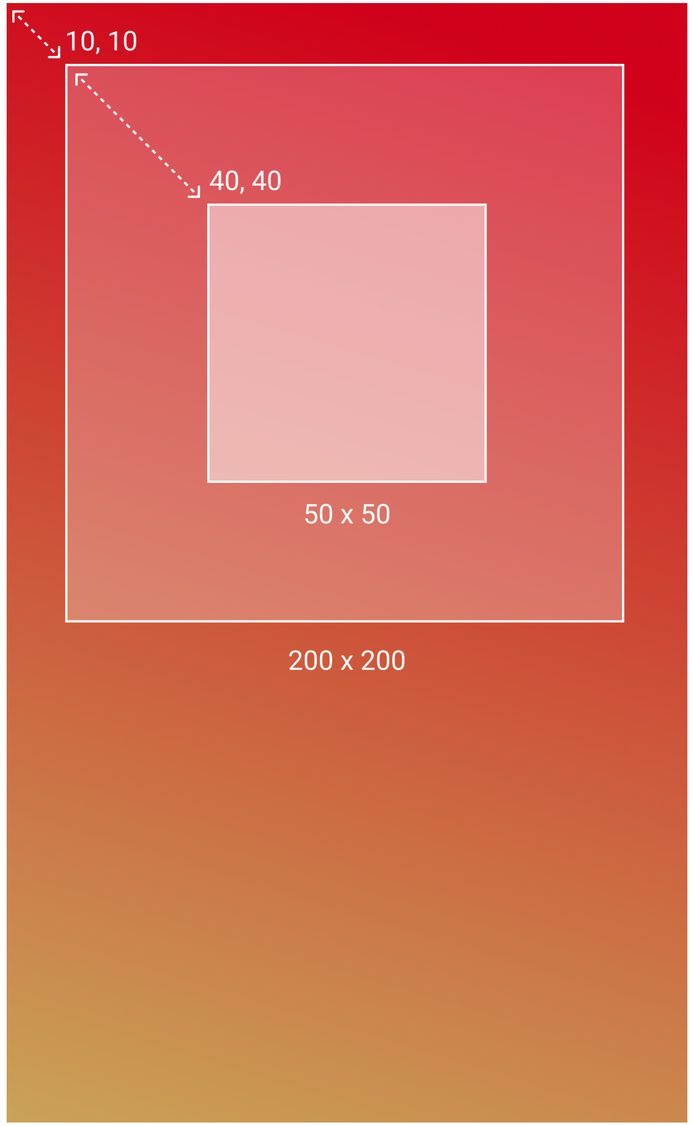
# Components

- Overuse components
- Components are your bridge to responsive layout
- They modularise your layout logic
- They simplify your mouse interaction logic

# Relative positions

Assume that your screen size

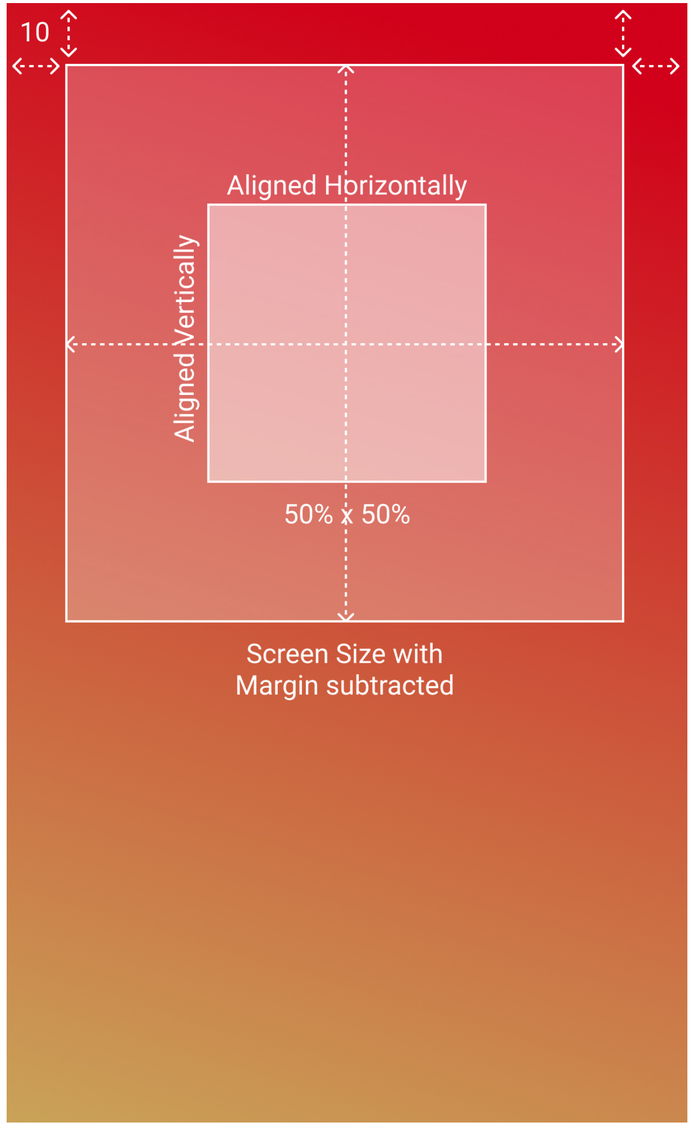
and component size are subject to change



# Relative positions

Assume that your screen size

and component size are subject to change



# Proportional sizes

## Assume your screen DPI changes

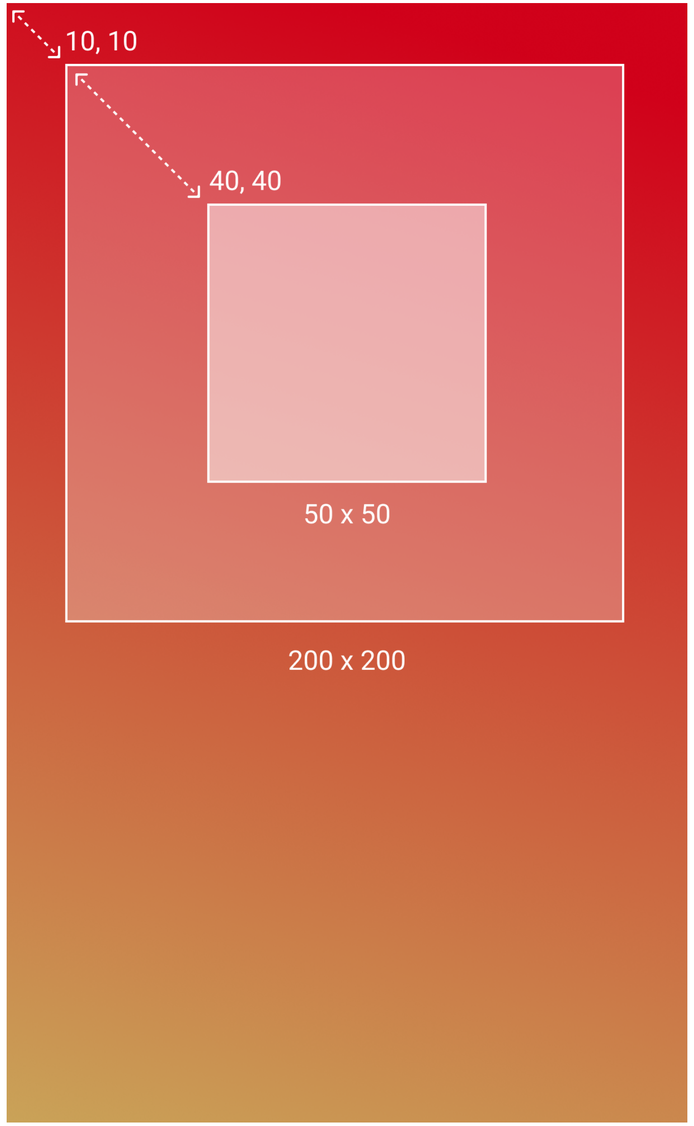
## Flex in FlexBox, Fr in Grid

Negative space can be Flex/Grid Items too

## Note the difference between Margin and Padding

Margins defined outside a component

Padding defined inside a component



# Proportional sizes

## Assume your screen DPI changes

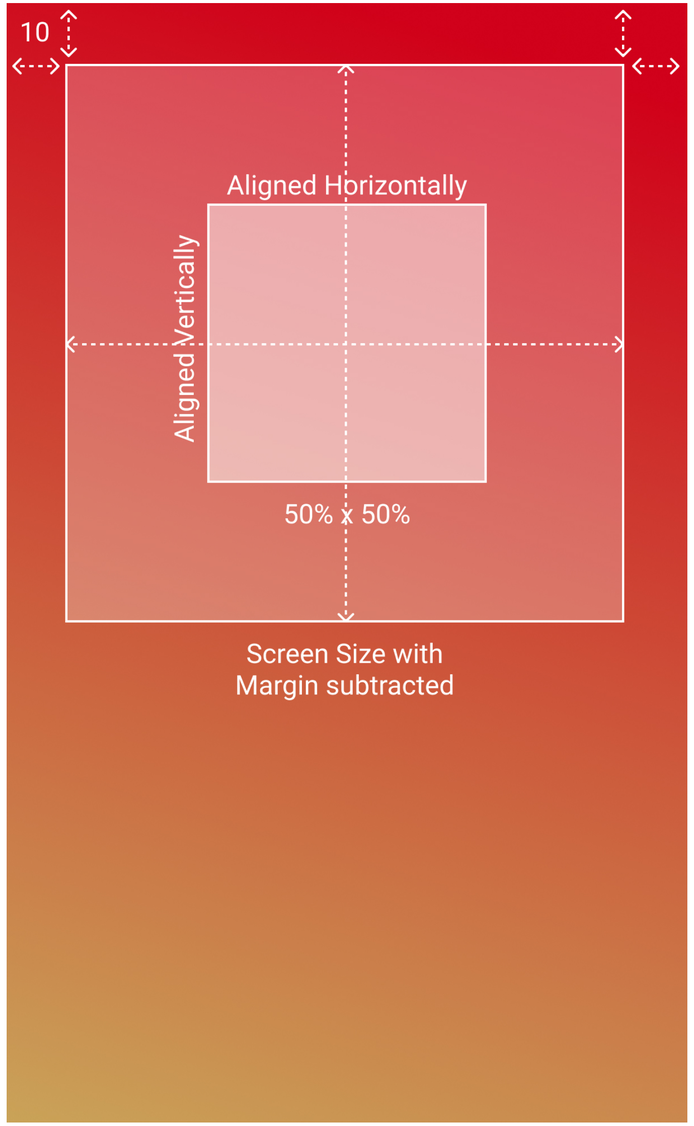
## Flex in FlexBox, Fr in Grid

Negative space can be Flex/Grid Items too

## Note the difference between Margin and Padding

Margins defined outside a component

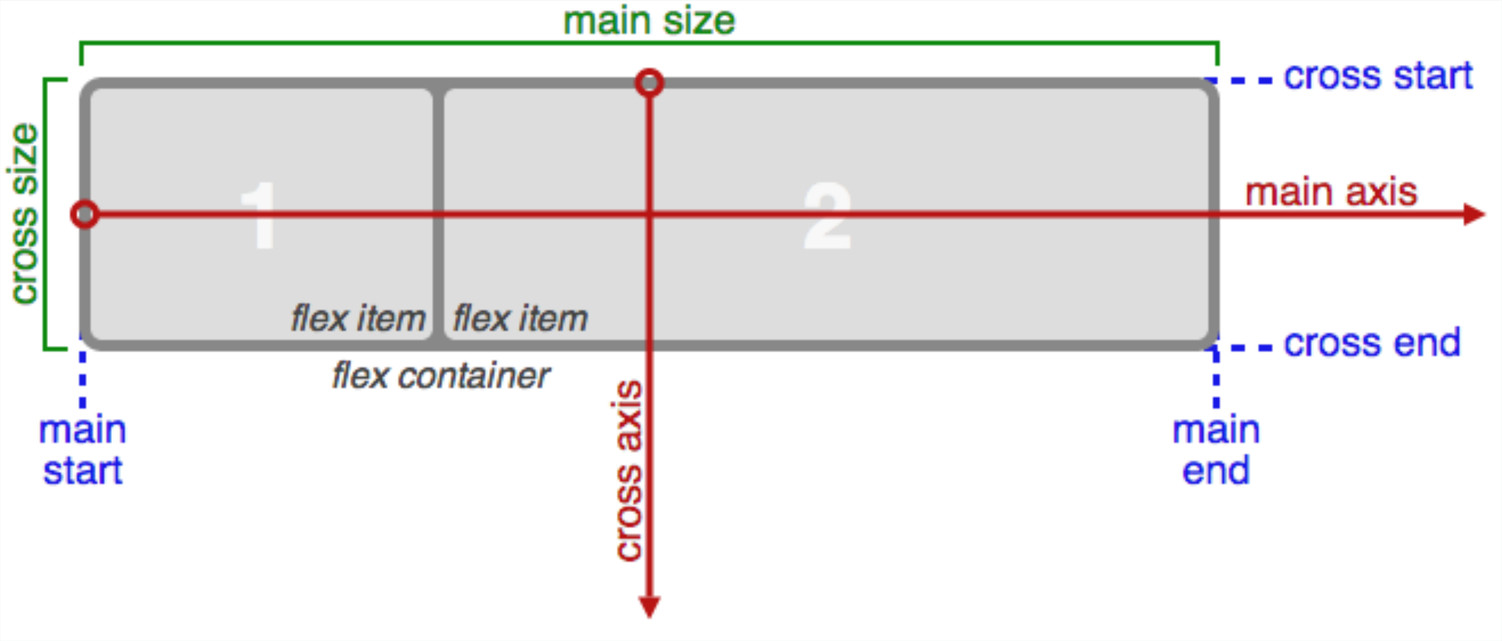
Padding defined inside a component



# Main axis and Cross axis

Justify along the main-axis

Align along the cross-axis



# Declarative layout

# Declarative layout

- Use FlexBox and Grid containers as much as possible

# Declarative layout

- Use FlexBox and Grid containers as much as possible
- Only add imperative logic when your containers can’t handle it

# Declarative layout

- Use FlexBox and Grid containers as much as possible
- Only add imperative logic when your containers can’t handle it
- Most of the times, this layout logic should be Breakpoints

# Breakpoints

Imperative logic based on:

# Breakpoints

Imperative logic based on:

- Aspect Ratio

# Breakpoints

Imperative logic based on:

- Aspect Ratio
- Available Width or Height

# Breakpoints

Imperative logic based on:

- Aspect Ratio
- Available Width or Height
- Screen DPI

# Breakpoints

Imperative logic based on:

- Aspect Ratio
- Available Width or Height
- Screen DPI

NOT based on:

- Platform flags

# Grid Example

# Useful links

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

https://gridbyexample.com

http://cssgridgarden.com

https://aerolab.co/blog/flexbox-grids/

# Graphical assets

# Component artwork tips

What's the best way to integrate graphical assets for a resizable interface?

You can either:

# Component artwork tips

What's the best way to integrate graphical assets for a resizable interface?

You can either:

- Use vector image formats (e.g. SVG or Juce Paths)

# Component artwork tips

What's the best way to integrate graphical assets for a resizable interface?

You can either:

- Use vector image formats (e.g. SVG or Juce Paths)
- Use Raster image formats (e.g. PNG) that are rendered at a higher DPI + size than you need and downscale

# Component artwork tips

What's the best way to integrate graphical assets for a resizable interface?

You can either:

- Use vector image formats (e.g. SVG or Juce Paths)
- Use Raster image formats (e.g. PNG) that are rendered at a higher DPI + size than you need and downscale
- Use vector drawing techniques

# Component artwork tips

What's the best way to integrate graphical assets for a resizable interface?

You can either:

- Use vector image formats (e.g. SVG or Juce Paths)
- Use Raster image formats (e.g. PNG) that are rendered at a higher DPI + size than you need and downscale
- Use vector drawing techniques
- Use font based resources

# Use an SVG

```
const String svgData (CharPointer_UTF8 (BinaryData::juceLogo_svg), BinaryData::juceLogo_svgSize);

ScopedPointer<Drawable> svg;

if (ScopedPointer<XmlElement> root = XmlDocument::parse (svgData))
    svg = Drawable::createFromSVG (*root);
```

# Use an SVG

```
const String svgData (CharPointer_UTF8 (BinaryData::juceLogo_svg), BinaryData::juceLogo_svgSize);

ScopedPointer<Drawable> svg;

if (ScopedPointer<XmlElement> root = XmlDocument::parse (svgData))
    svg = Drawable::createFromSVG (*root);
```

- This is fine if you have a fairly simple UI

# Use an SVG

```
const String svgData (CharPointer_UTF8 (BinaryData::juceLogo_svg), BinaryData::juceLogo_svgSize);

ScopedPointer<Drawable> svg;

if (ScopedPointer<XmlElement> root = XmlDocument::parse (svgData))
    svg = Drawable::createFromSVG (*root);
```

- This is fine if you have a fairly simple UI
- With a complex UI you might find that the rendering speed drops, especially when under load from animations

# Use an SVG

```
const String svgData (CharPointer_UTF8 (BinaryData::juceLogo_svg), BinaryData::juceLogo_svgSize);

ScopedPointer<Drawable> svg;

if (ScopedPointer<XmlElement> root = XmlDocument::parse (svgData))
    svg = Drawable::createFromSVG (*root);
```

- This is fine if you have a fairly simple UI
- With a complex UI you might find that the rendering speed drops, especially when under load from animations
- SVG renders never look *quite* the same as your designers application

# Use a Path

- Open the SVG and copy the part beginning d=" up to the last "

```
<svg width="18px" height="18px" viewBox="0 0 18 18">
    <g id="Page-1" stroke="none" stroke-width="1" fill="none"
fill-rule="evenodd">
        <g id="00" transform="translate(-334.000000,
-28.000000)" stroke="#FFFFFF" stroke-width="1.2">
            <g id="ic_settings"
transform="translate(327.000000, 21.000000)">
                <path d="M21.7113844,16.8 C21.7113844,16.56
21.7913844,16.32 21.7913844,16 C21.7913844,15.68
21.7913844,15.44 21.7113844,15.2 L23.3913844,13.84
C23.5513844,10.72 18.0313844,10.48 L17.7113844,8.4
C17.7913844,14.48 18.5913844,16 C18.5913844,17.52
17.3113844,18.8 15.7913844,18.8 L15.7913844,18.8 Z"
id="Shape"></path>
            </g>
        </g>
    </g>
</svg>
```

# Use a Path

- Open the SVG and copy the part beginning d=" up to the last "
- Paste it in to the Projucer "SVG Path Converter", which will generate your path data

```
<svg width="18px" height="18px" viewBox="0 0 18 18">
    <g id="Page-1" stroke="none" stroke-width="1" fill="none"
fill-rule="evenodd">
        <g id="00" transform="translate(-334.000000,
-28.000000)" stroke="#FFFFFF" stroke-width="1.2">
            <g id="ic_settings"
transform="translate(327.000000, 21.000000)">
                <path d="M21.7113844,16.8 C21.7113844,16.56
21.7913844,16.32 21.7913844,16 C21.7913844,15.68
21.7913844,15.44 21.7113844,15.2 L23.3913844,13.84
C23.5513844,10.72 18.0313844,10.48 L17.7113844,8.4
C17.7913844,14.48 18.5913844,16 C18.5913844,17.52
17.3113844,18.8 15.7913844,18.8 L15.7913844,18.8 Z"
id="Shape"></path>
            </g>
        </g>
    </g>
</svg>
```

```
static const unsigned char pathData[] =
{
    110,109,0,0,176,65,0,0,0,0,108,0,0,128,63,0,0,
    0,0,98,45,178,219, 62,0,0,0,0,0,0,0,0,45,178,
    219,62,0,0,0,0,0,0,128,63,108,0,0,0,0,0, 0,176,65,
    98,0,0,0,0,55,145,180,65,45,178,219,62,0,0,184,65
};

Path path;
path.loadPathFromData (pathData, sizeof (pathData));
```

# Use a Path

- Open the SVG and copy the part beginning d=" up to the last "
- Paste it in to the Projucer "SVG Path Converter", which will generate your path data
- Paste the output code in to your application for use

```
<svg width="18px" height="18px" viewBox="0 0 18 18">
    <g id="Page-1" stroke="none" stroke-width="1" fill="none"
fill-rule="evenodd">
        <g id="00" transform="translate(-334.000000,
-28.000000)" stroke="#FFFFFF" stroke-width="1.2">
            <g id="ic_settings"
transform="translate(327.000000, 21.000000)">
                <path d="M21.7113844,16.8 C21.7113844,16.56
21.7913844,16.32 21.7913844,16 C21.7913844,15.68
21.7913844,15.44 21.7113844,15.2 L23.3913844,13.84
C23.5513844,10.72 18.0313844,10.48 L17.7113844,8.4
C17.7913844,14.48 18.5913844,16 C18.5913844,17.52
17.3113844,18.8 15.7913844,18.8 L15.7913844,18.8 Z"
id="Shape"></path>
            </g>
        </g>
    </g>
</svg>
```

```
static const unsigned char pathData[] =
{
    110,109,0,0,176,65,0,0,0,0,108,0,0,128,63,0,0,
    0,0,98,45,178,219, 62,0,0,0,0,0,0,0,0,45,178,
    219,62,0,0,0,0,0,0,128,63,108,0,0,0,0,0, 0,176,65,
    98,0,0,0,0,55,145,180,65,45,178,219,62,0,0,184,65
};

Path path;
path.loadPathFromData (pathData, sizeof (pathData));
```

# Use a Path

- That's okay when it works well

# Use a Path

- That's okay when it works well
- But… expect to see "No path generated.. Not a valid SVG path string?" appearing quite a lot…

# Use a Path

- That's okay when it works well
- But… expect to see "No path generated.. Not a valid SVG path string?" appearing quite a lot…
  - Ask your designers nicely to export the SVG using *1 layer and 1 singular path*

# Use a Path

- That's okay when it works well
- But… expect to see "No path generated.. Not a valid SVG path string?" appearing quite a lot…
  - Ask your designers nicely to export the SVG using *1 layer and 1 singular path*
- Be aware that path rendering depends on the fill type (either stroke or fill)

# Use a Path

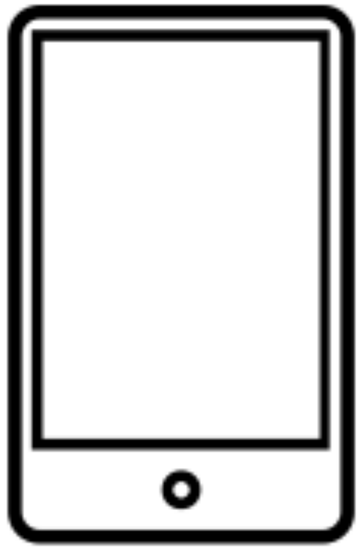
- That's okay when it works well
- But… expect to see "No path generated.. Not a valid SVG path string?" appearing quite a lot…
  - Ask your designers nicely to export the SVG using *1 layer and 1 singular path*
- Be aware that path rendering depends on the fill type (either stroke or fill)
- Has an in memory overhead vs loading resources on demand

# Use a Path

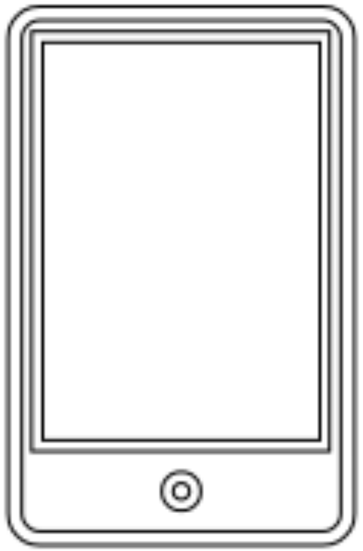
Be aware that path rendering depends on the how you render the path to look correct, using either Fill or stroke:

```
g.strokePath (path, PathStrokeType (1.0f));

g.fillPath (path);
```



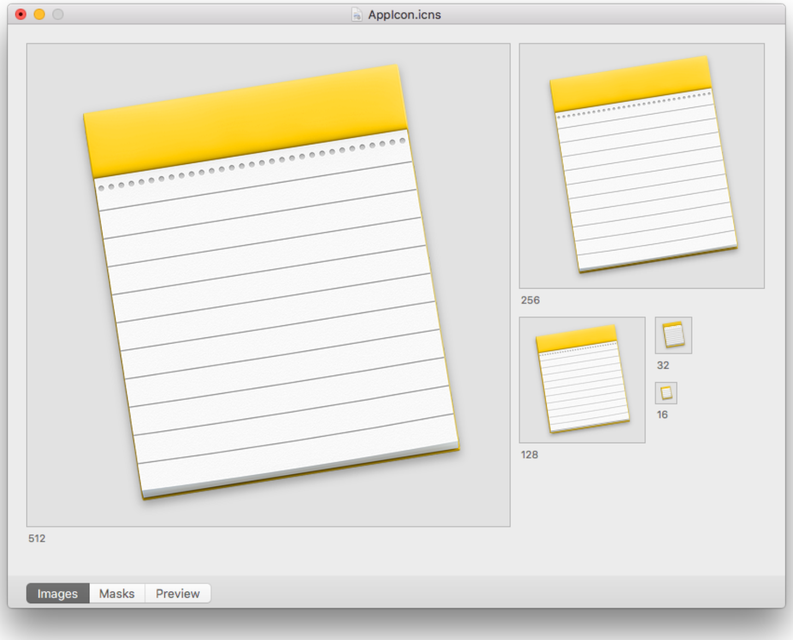
Fill



Stroke

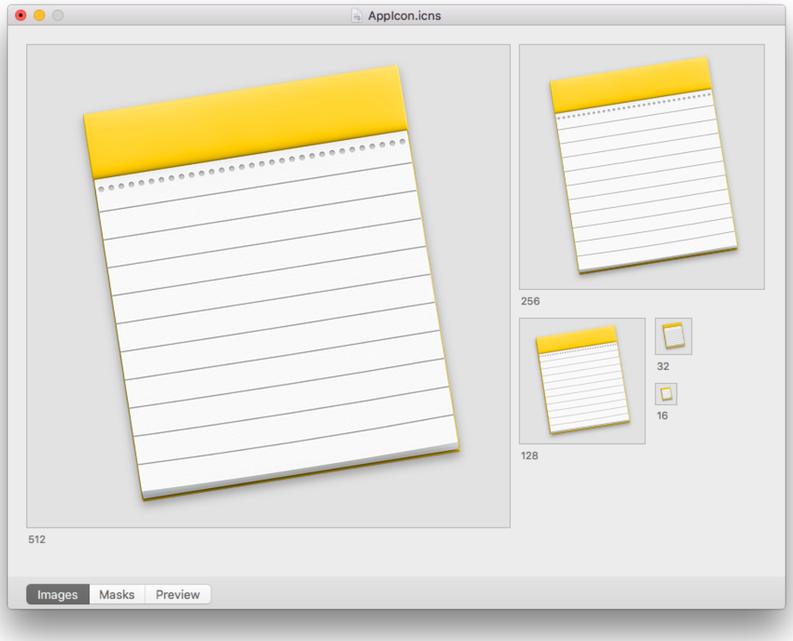
# Use rasters and downscale

- This is how Apple icon files work



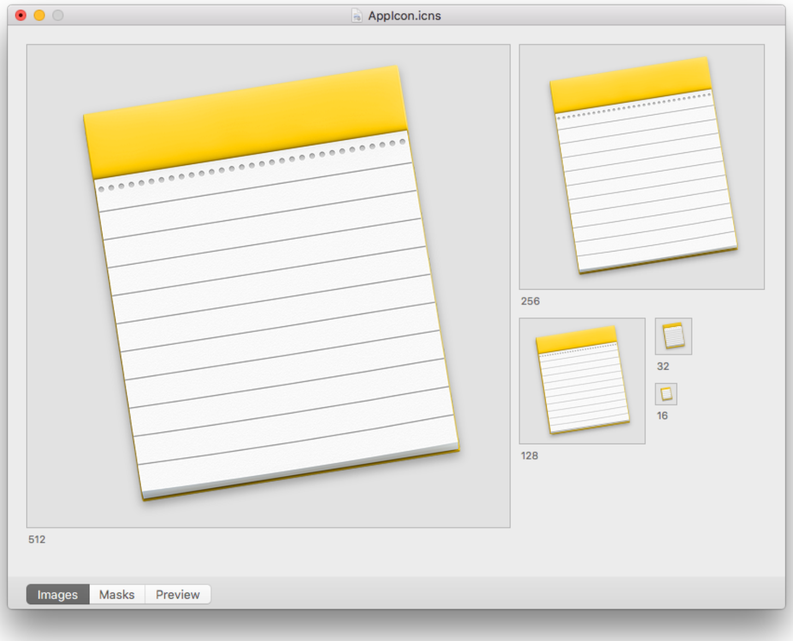
# Use rasters and downscale

- This is how Apple icon files work
- You can control the look at different sizes and resolutions



# Use rasters and downscale

- This is how Apple icon files work
- You can control the look at different sizes and resolutions
- But… you can end up having to maintain lots of resources




icon_16x16.png
icon_16x16@2x.png
icon_32x32.png
icon_32x32@2x.png
icon_128x128.png
icon_128x128@2x.png
icon_256x256.png
icon_256x256@2x.png
icon_512x512.png
icon_512x512@2x.png

# Always use the biggest image possible

- When working with Raster images, always round to the *nearest image with a resolution greater than you require*

# Always use the biggest image possible

- When working with Raster images, always round to the *nearest image with a resolution greater than you require*
- Always scale images down, don't scale them up!

Scaled up

Scaled down

# Use vector drawing techniques

Use LookAndFeel and draw using the built in Path or Graphics functions

# Use vector drawing techniques

Use LookAndFeel and draw using the built in Path or Graphics functions

- With complex GUIs rendering speed might drop

# Use vector drawing techniques

Use LookAndFeel and draw using the built in Path or Graphics functions

- With complex GUIs rendering speed might drop
- Long term maintenance of this kind of drawing can become *hard*

# Use vector drawing techniques

Use LookAndFeel and draw using the built in Path or Graphics functions

- With complex GUIs rendering speed might drop
- Long term maintenance of this kind of drawing can become *hard*
- Because of the way LookAndFeel works, it can be hard if you want 2 similar components to look different

# Use vector drawing techniques

Use LookAndFeel and draw using the built in Path or Graphics functions

- With complex GUIs rendering speed might drop
- Long term maintenance of this kind of drawing can become *hard*
- Because of the way LookAndFeel works, it can be hard if you want 2 similar components to look different
- Depending on the language you are using you can use tools like https://www.paintcodeapp.com

# Use font based resources

- Use a tool such as https://icomoon.io to generate a TTF

# Use font based resources

- Use a tool such as https://icomoon.io to generate a TTF
- Add your font file to your JUCE project via the Projucers "Add Existing Files.." option
  This will add it to the binary resources file

# Use font based resources

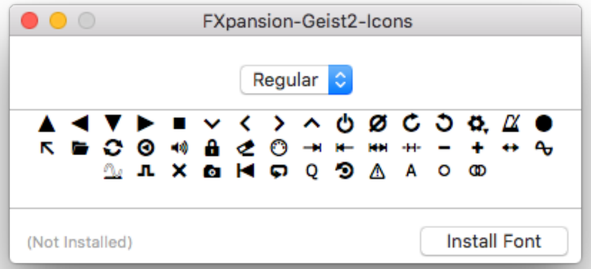
- Create a Font directly from the binary data

```
Font drawFont =
Typeface::createSystemTypefaceFor
                (BinaryData::icons_ttf,
                 BinaryData::icons_ttfSize);
```

# Use font based resources

- Create a Font directly from the binary data
- Open the font file in "Font book" and find the font you want to use.
  Copy it using ⌘+C or Edit → Copy

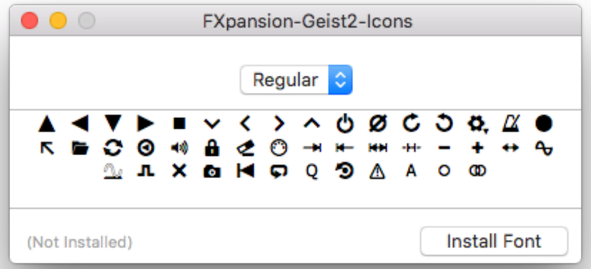
```
Font drawFont =
Typeface::createSystemTypefaceFor
                (BinaryData::icons_ttf,
                 BinaryData::icons_ttfSize);
```

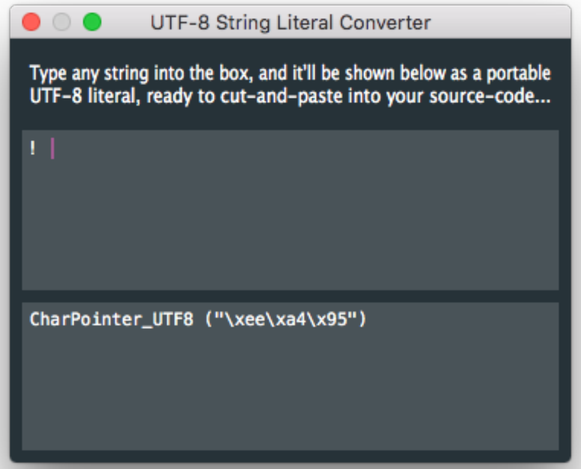


# Use font based resources

- Create a Font directly from the binary data
- Open the font file in "Font book" and find the font you want to use.
  Copy it using ⌘+C or Edit → Copy
- Open the Projucer "UTF-8 String Literal Converter" and paste the copied font resource

```
Font drawFont =
Typeface::createSystemTypefaceFor
                    (BinaryData::icons_ttf,
                     BinaryData::icons_ttfSize);
```



# Use font based resources

- Copy the generated CharPointer_UTF8 in to your source code

# Use font based resources

- Copy the generated CharPointer_UTF8 in to your source code
  - Note that you cannot just paste directly from Font Book to the source code, it will generate Unicode that the compiler will not thank you for…

# Use font based resources

- Copy the generated CharPointer_UTF8 in to your source code
  - Note that you cannot just paste directly from Font Book to the source code, it will generate Unicode that the compiler will not thank you for…
- Draw the font using normal Juce text rendering routines

```
void drawIcon (Graphics& g, Rectangle<float> area, CharPointer_UTF8 character)
{
    g.setFont (drawFont.withHeight (std::min<float> (area.getWidth(), area.getHeight())));
    g.drawText (character, area, Justification::centred);
}
```

# Use font based resources

- Juce will figure out all the kerning issues, rescaling and layout for you

# Use font based resources

- Juce will figure out all the kerning issues, rescaling and layout for you
- But you are limited on the kinds of fills and highlights that you can apply

# Use font based resources

- Juce will figure out all the kerning issues, rescaling and layout for you
- But you are limited on the kinds of fills and highlights that you can apply
- Won't work for an entire component

# Use font based resources

- Juce will figure out all the kerning issues, rescaling and layout for you
- But you are limited on the kinds of fills and highlights that you can apply
- Won't work for an entire component
  - You can't render a button this way, just the icon on a button

# What are performance bottlenecks?

- Drawing text is always expensive

# What are performance bottlenecks?

- Drawing text is always expensive
  - So beware of using the font icon rendering trick…

# What are performance bottlenecks?

- Drawing text is always expensive
  - So beware of using the font icon rendering trick…
- Animation can have a heavy cost if it is over a large amount of the UI and if it is continuous

# What are performance bottlenecks?

- Drawing text is always expensive
  - So beware of using the font icon rendering trick…
- Animation can have a heavy cost if it is over a large amount of the UI and if it is continuous
- Lots of calls to repaint, especially for big components

# Whats best ???

All these solutions have drawbacks, so use a combination:

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts
- Assess the performance of your application

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts
- Assess the performance of your application
  - If SVG is causing performance problems drop back to using scaled rasters

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts
- Assess the performance of your application
  - If SVG is causing performance problems drop back to using scaled rasters
  - Offscreen renders of SVGs

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts
- Assess the performance of your application
  - If SVG is causing performance problems drop back to using scaled rasters
  - Offscreen renders of SVGs
  - Reduce amount of animation

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts
- Assess the performance of your application
  - If SVG is causing performance problems drop back to using scaled rasters
  - Offscreen renders of SVGs
  - Reduce amount of animation
  - Reduce calls to repaint();

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts
- Assess the performance of your application
  - If SVG is causing performance problems drop back to using scaled rasters
  - Offscreen renders of SVGs
  - Reduce amount of animation
  - Reduce calls to repaint();
- Assess the graphical fidelity of your application

# Whats best ???

All these solutions have drawbacks, so use a combination:

- Start by using vector resources, such as SVGs, Paths or Fonts
- Assess the performance of your application
  - If SVG is causing performance problems drop back to using scaled rasters
  - Offscreen renders of SVGs
  - Reduce amount of animation
  - Reduce calls to repaint();
- Assess the graphical fidelity of your application
  - If fidelity is lower than you would like, try using path based drawing techniques

# What's in Juce to help you?

## Buffering a component that is heavy to draw can help:

```
void setBufferedToImage (bool shouldBeBuffered);
void setCachedComponentImage (CachedComponentImage* newCachedImage);
```

# What's in Juce to help you?

Buffering a component that is heavy to draw can help:

```
void setBufferedToImage (bool shouldBeBuffered);
void setCachedComponentImage (CachedComponentImage* newCachedImage);
```

But this wont always help (and in some cases will make performance worse, depending on where your bottleneck is)

# What's in Juce to help you?

Buffering a component that is heavy to draw can help:

```
void setBufferedToImage (bool shouldBeBuffered);
void setCachedComponentImage (CachedComponentImage* newCachedImage);
```

But this wont always help (and in some cases will make performance worse, depending on where your bottleneck is)

You all know this, but remember: *PROFILE, PROFILE, PROFILE*

# Cross thread communication

# Audio Plugin Thread Structure

## Audio thread

# Audio Plugin Thread Structure

## Audio thread

- High priority thread

# Audio Plugin Thread Structure

## Audio thread

- High priority thread
- Don't do memory allocation or deallocation here*

# Audio Plugin Thread Structure

## Audio thread

- High priority thread
- Don't do memory allocation or deallocation here*

*Well you can, but not from the global heap as uses a global lock

# Audio Plugin Thread Structure

## Audio thread

- High priority thread
- Don't do memory allocation or deallocation here*
- Operations are heavily speed bound

*Well you can, but not from the global heap as uses a global lock

# Audio Plugin Thread Structure

UI (Juce Message) thread

# Audio Plugin Thread Structure

## UI (Juce Message) thread

- Low priority thread

# Audio Plugin Thread Structure

## UI (Juce Message) thread

- Low priority thread
- Can (and should) do memory allocation or deallocation here

# Audio Plugin Thread Structure

## UI (Juce Message) thread

- Low priority thread
- Can (and should) do memory allocation or deallocation here
- Any actions must not block / lock up the audio thread

# Plugin threading problems

For all plugins you have the problem of thread communications

# Plugin threading problems

For all plugins you have the problem of thread communications

- How to avoid race conditions

# Plugin threading problems

For all plugins you have the problem of thread communications

- How to avoid race conditions
- How to maintain data integrity

# Plugin threading problems

For all plugins you have the problem of thread communications

- How to avoid race conditions
- How to maintain data integrity
- How to do this in a lock-free way?

# Thread comms option 1

- For simple plugins, using only primitive types (int, float, bool) is a possible solution.

# Thread comms option 1

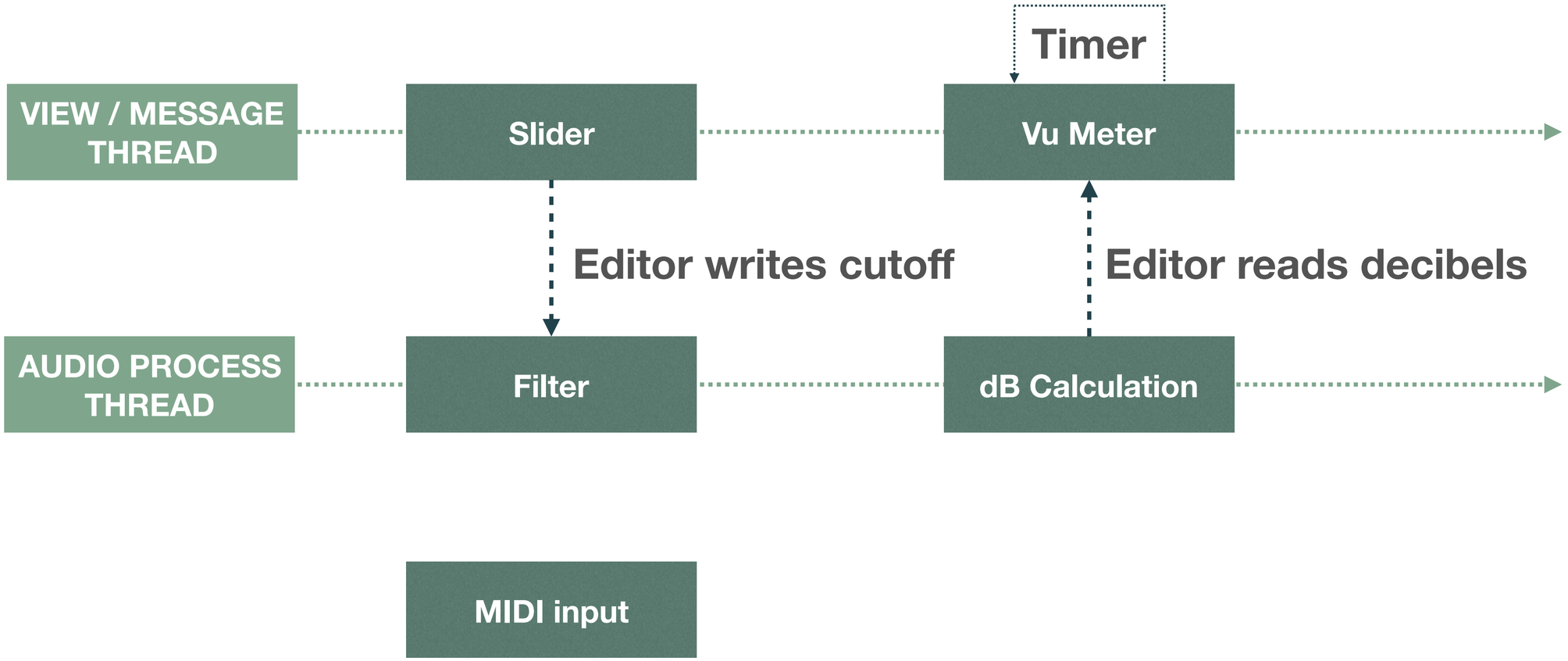
- For simple plugins, using only primitive types (int, float, bool) is a possible solution.
- On x86 / x64 architectures primitive types (up to 64 bits) have non-divisible load and store operations - other architectures may not be!

# Thread comms option 1

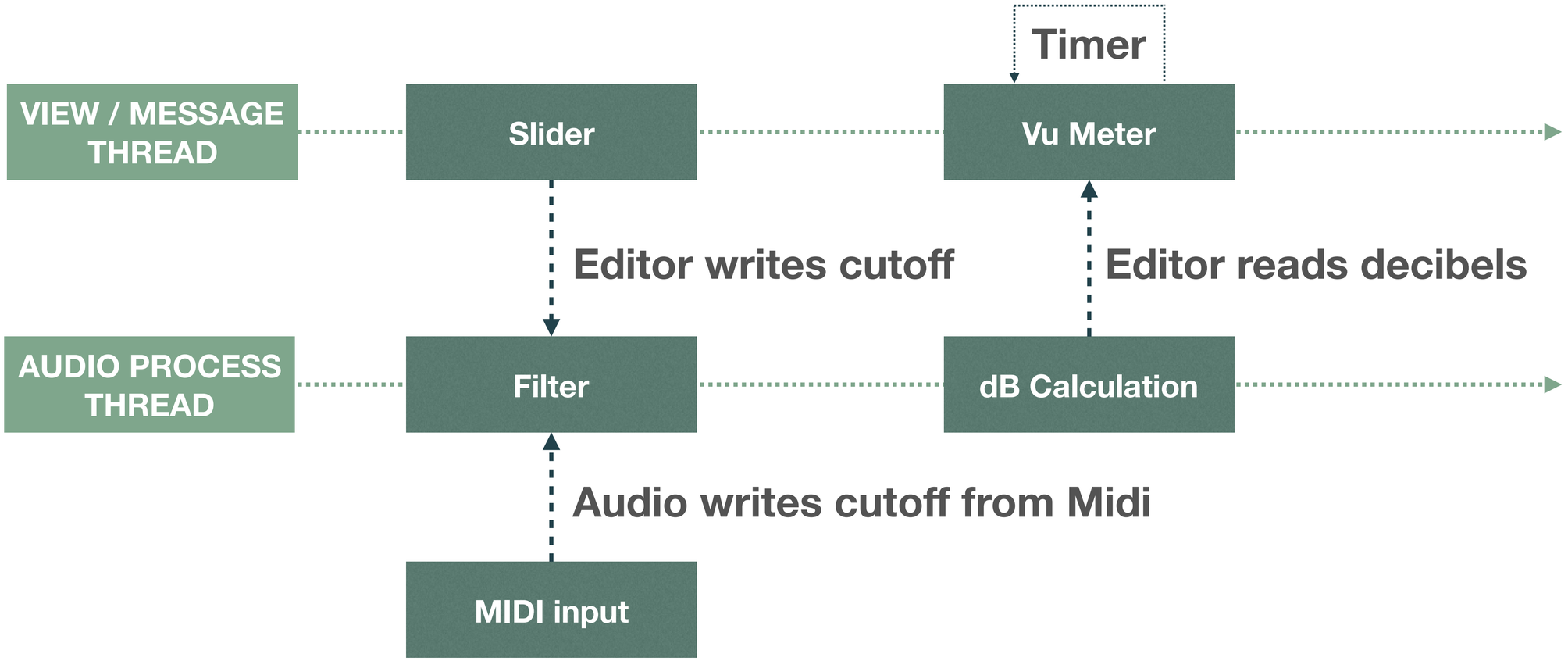
- For simple plugins, using only primitive types (int, float, bool) is a possible solution.
- On x86 / x64 architectures primitive types (up to 64 bits) have non-divisible load and store operations - other architectures may not be!
- You still have a potential for ordering errors, the question is does this matter?

# Thread comms option 1

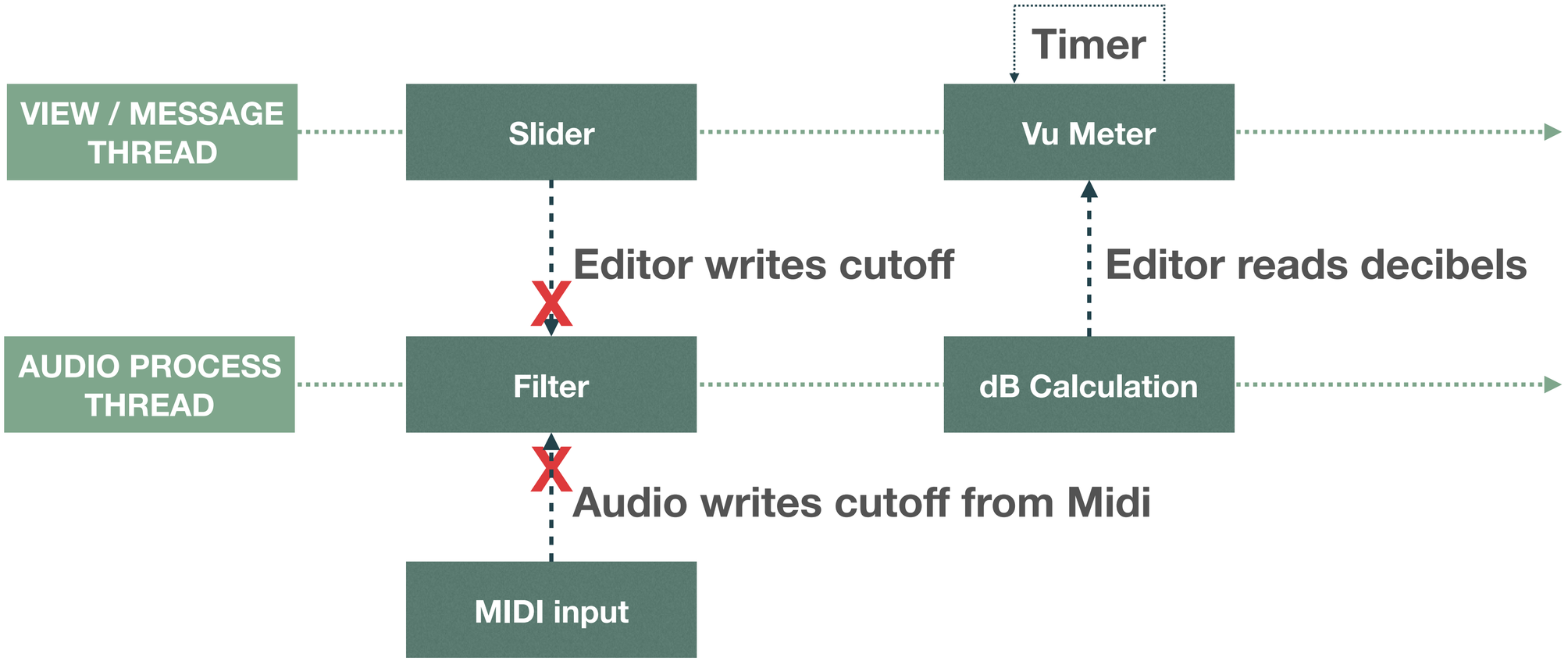
# Thread comms option 1



# Thread comms option 1



# Thread comms option 1



# Thread comms option 2

- For simple plugins you can make the Audio thread the master of all variables - it reads from all the other sources

# Thread comms option 2

- For simple plugins you can make the Audio thread the master of all variables - it reads from all the other sources
  - All reads must be non divisible

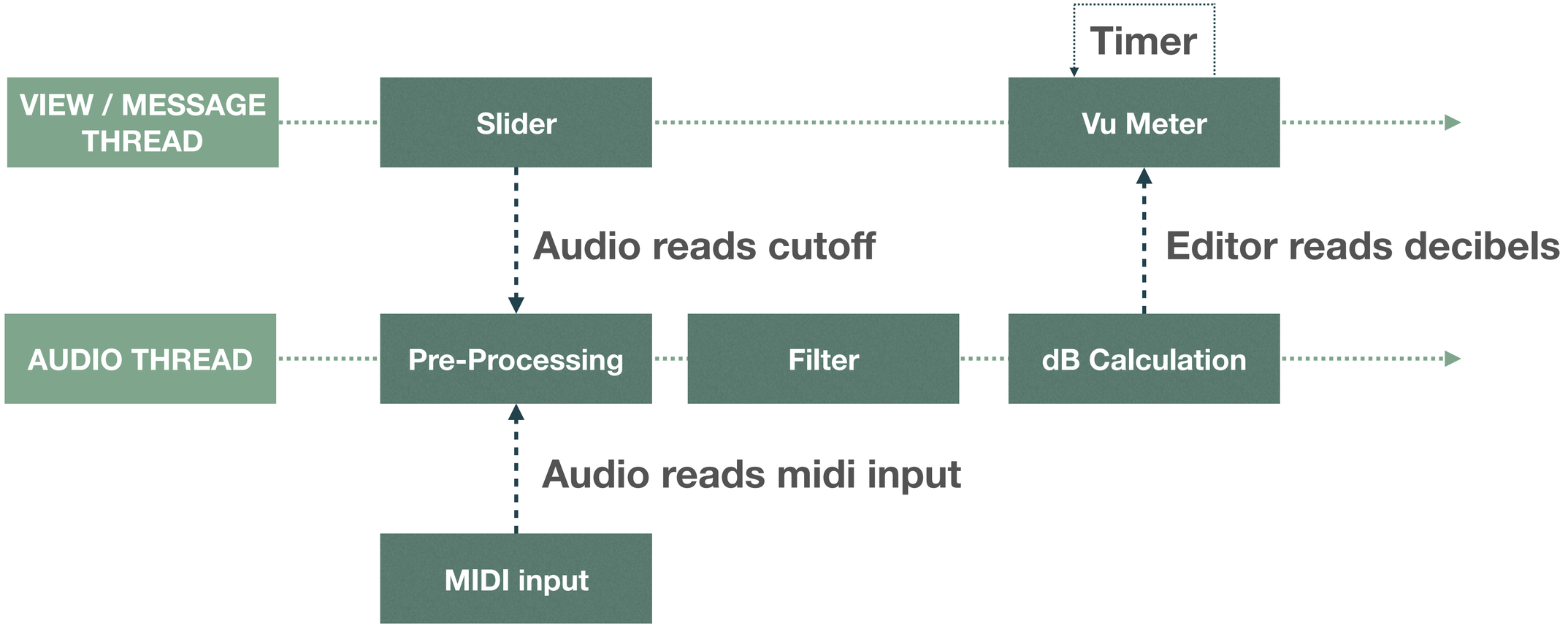
# Thread comms option 2

- For simple plugins you can make the Audio thread the master of all variables - it reads from all the other sources
  - All reads must be non divisible
  - All reads become time bound, so you must be confident about the way that they are being calculated…

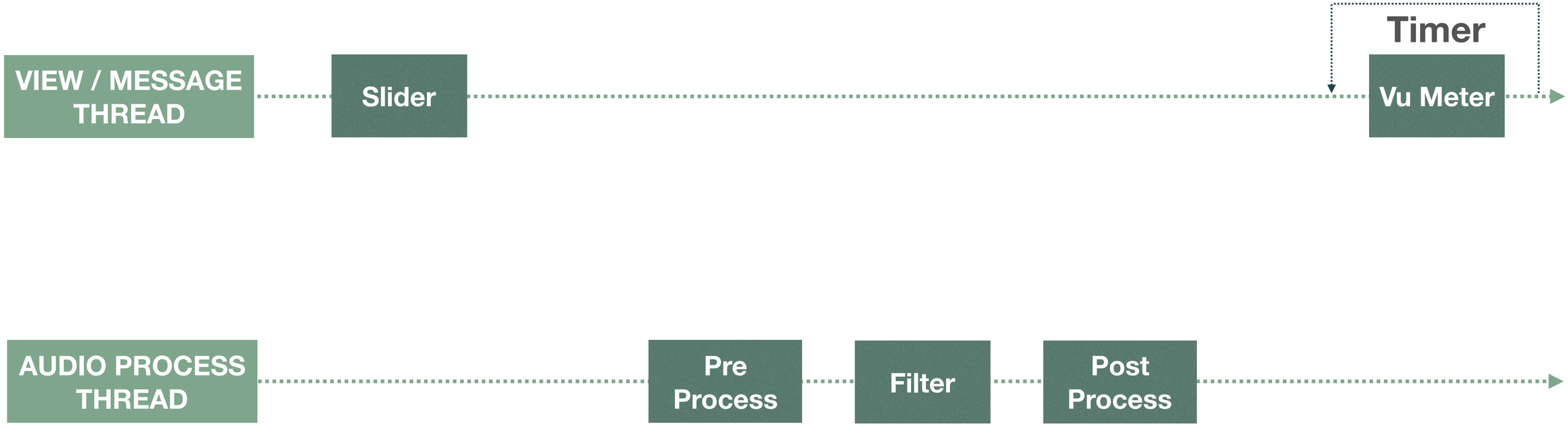
# Thread comms option 2

- For simple plugins you can make the Audio thread the master of all variables - it reads from all the other sources
  - All reads must be non divisible
  - All reads become time bound, so you must be confident about the way that they are being calculated…
- Audio thread must be aware of all inputs and must marshall the values in some way

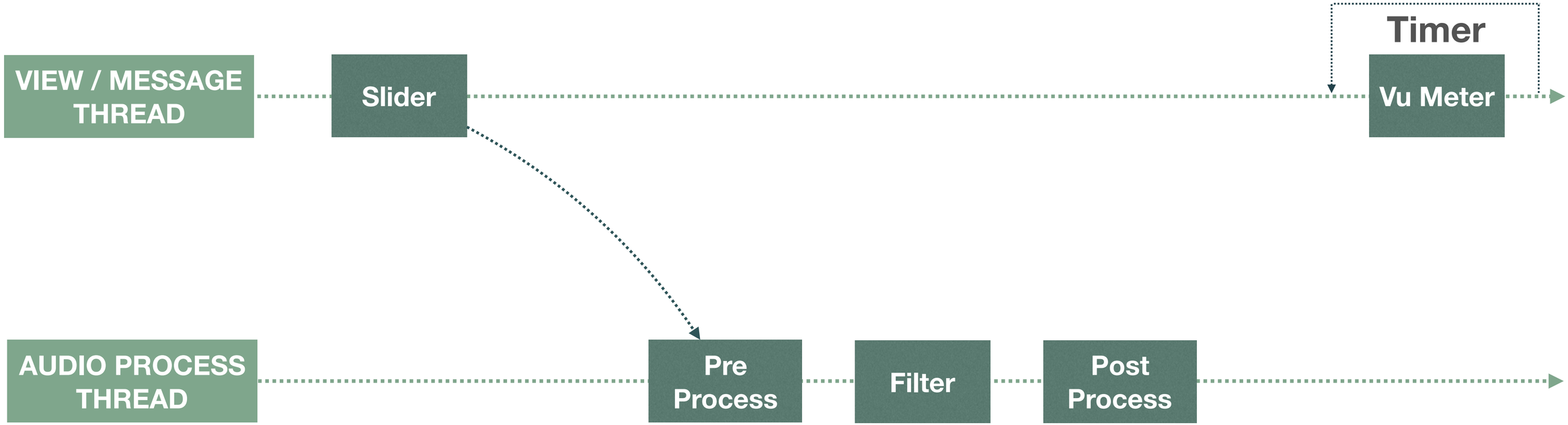
# Thread comms option 2



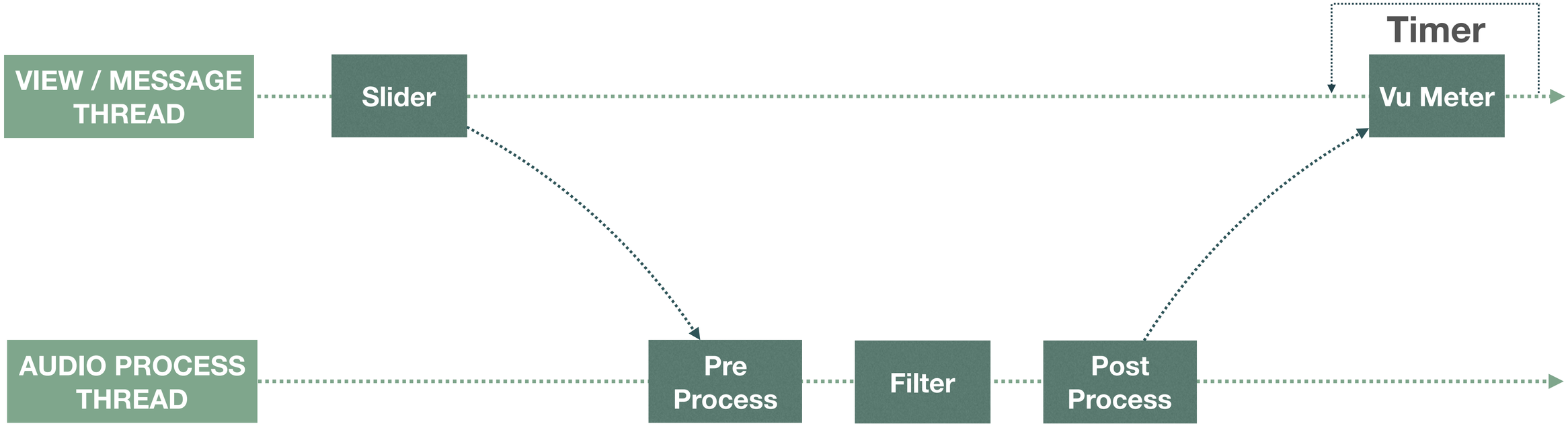
# Thread comms option 3



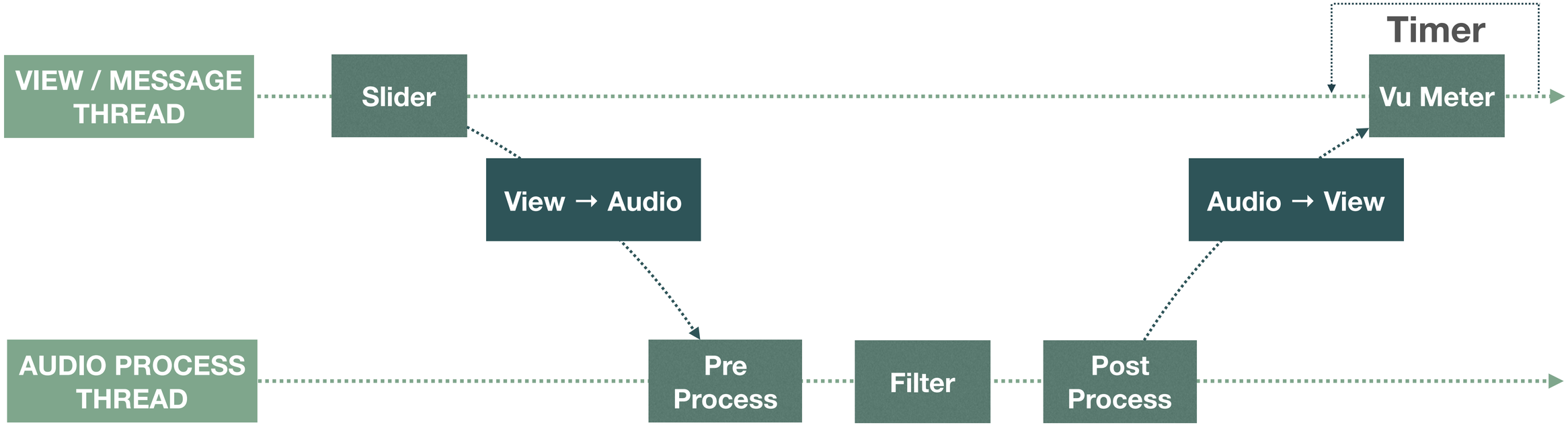
# Thread comms option 3



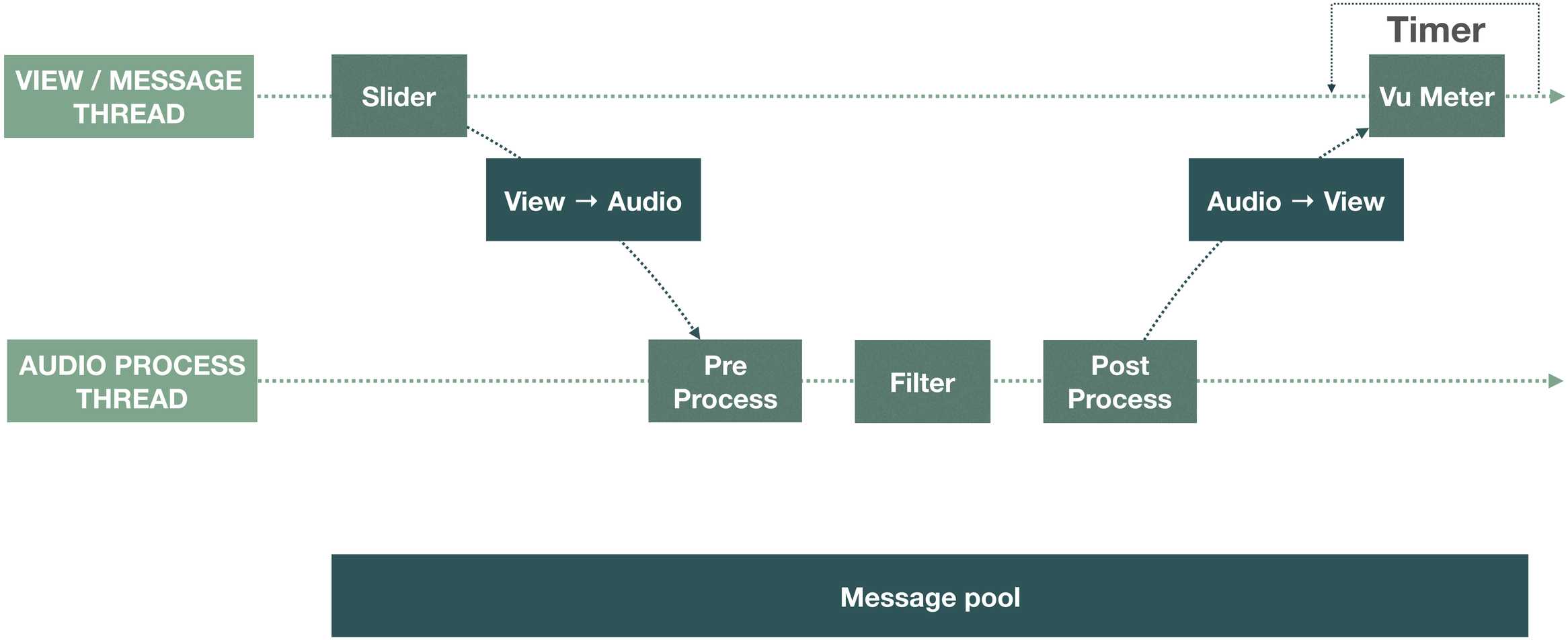
# Thread comms option 3



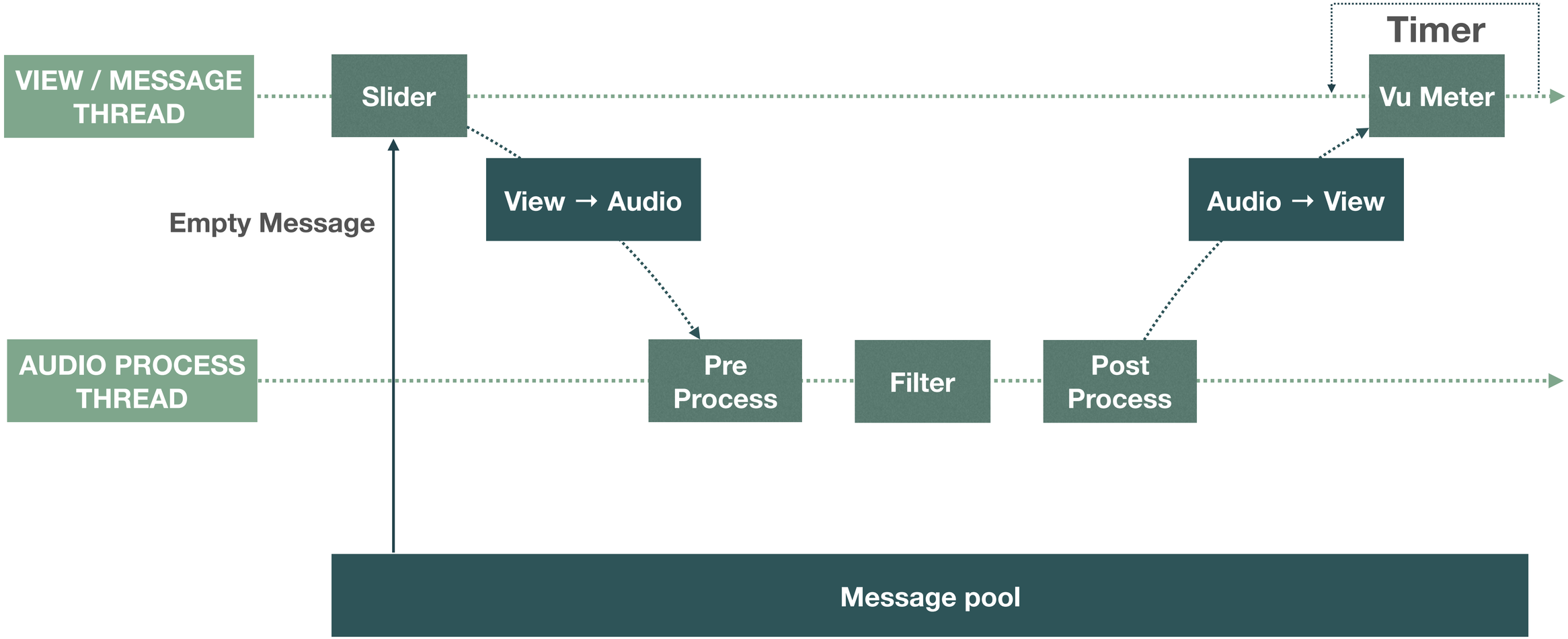
# Thread comms option 3



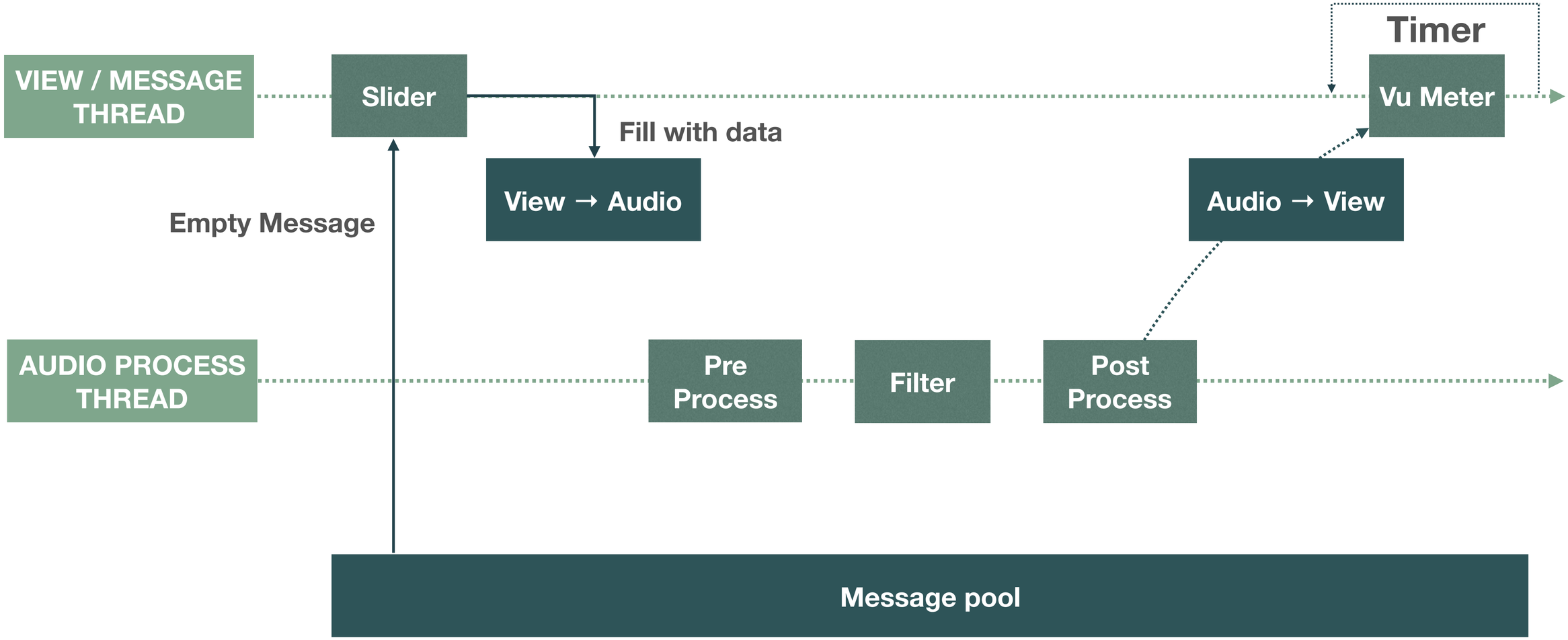
# Thread comms option 3



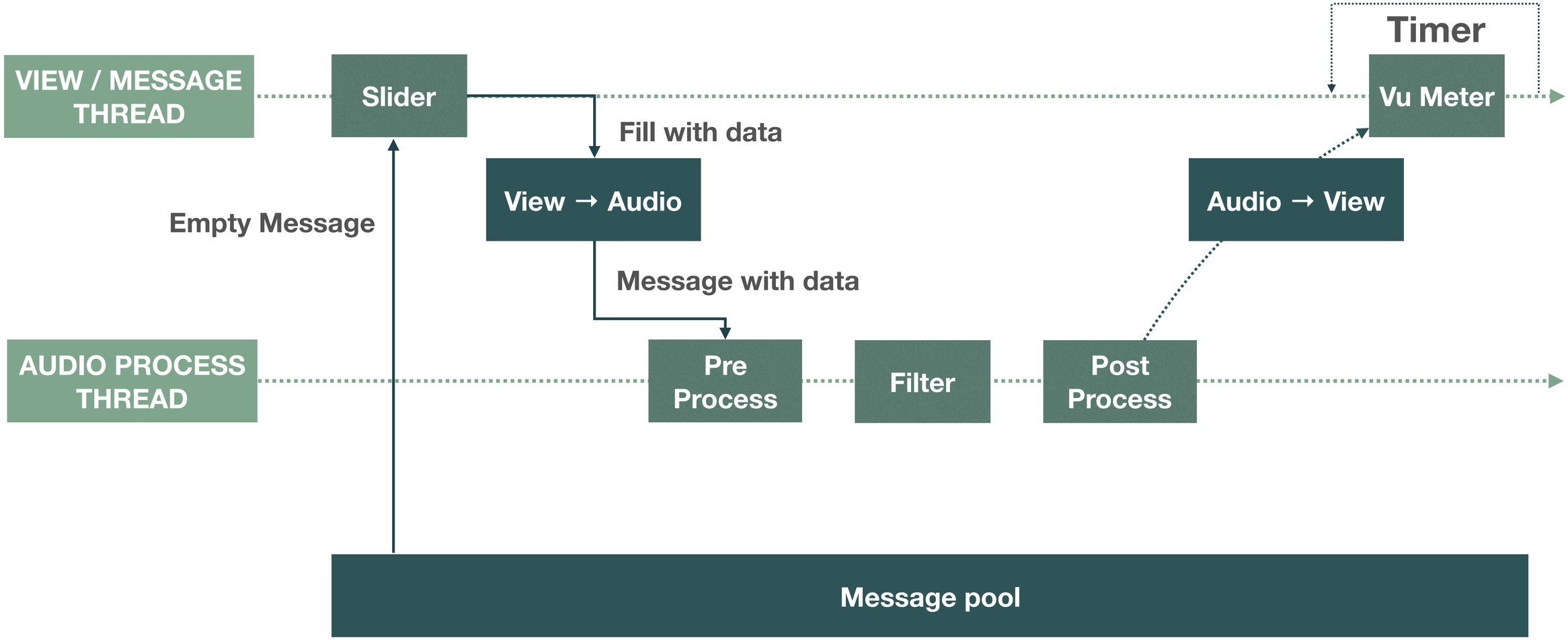
# Thread comms option 3



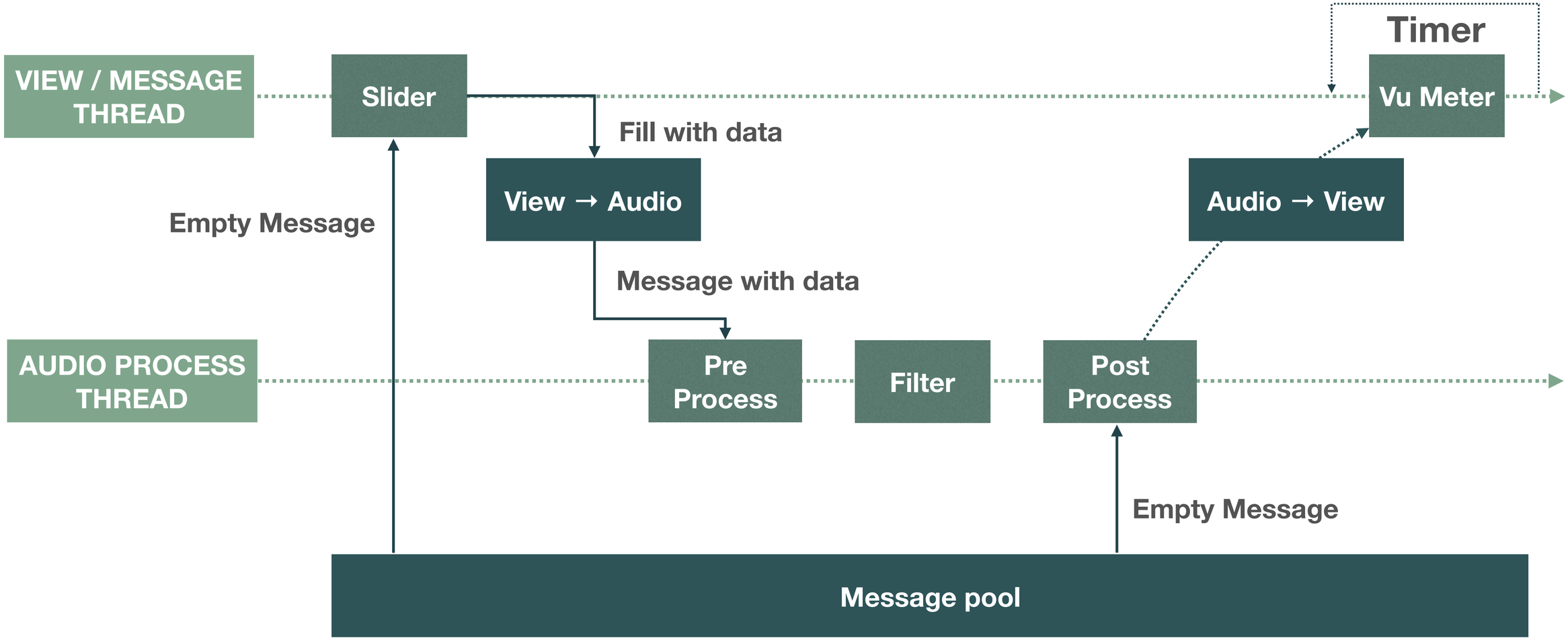
# Thread comms option 3



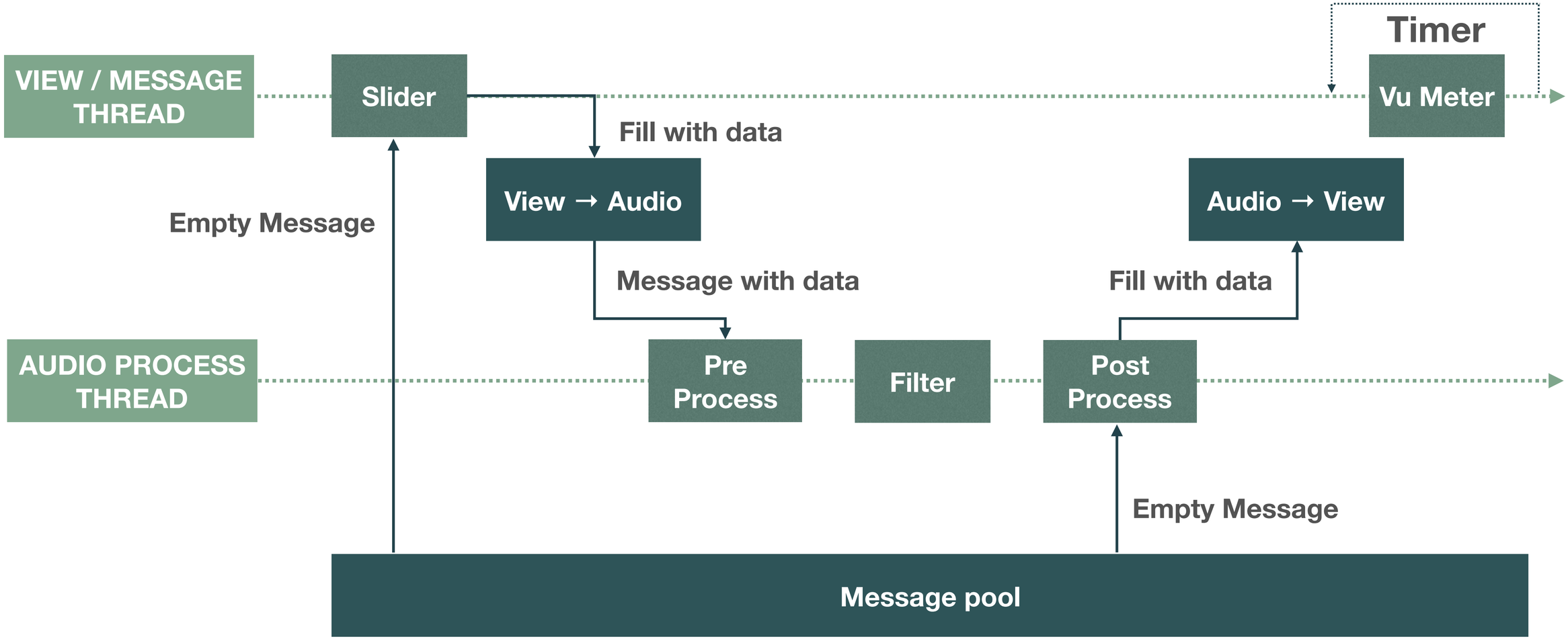
# Thread comms option 3



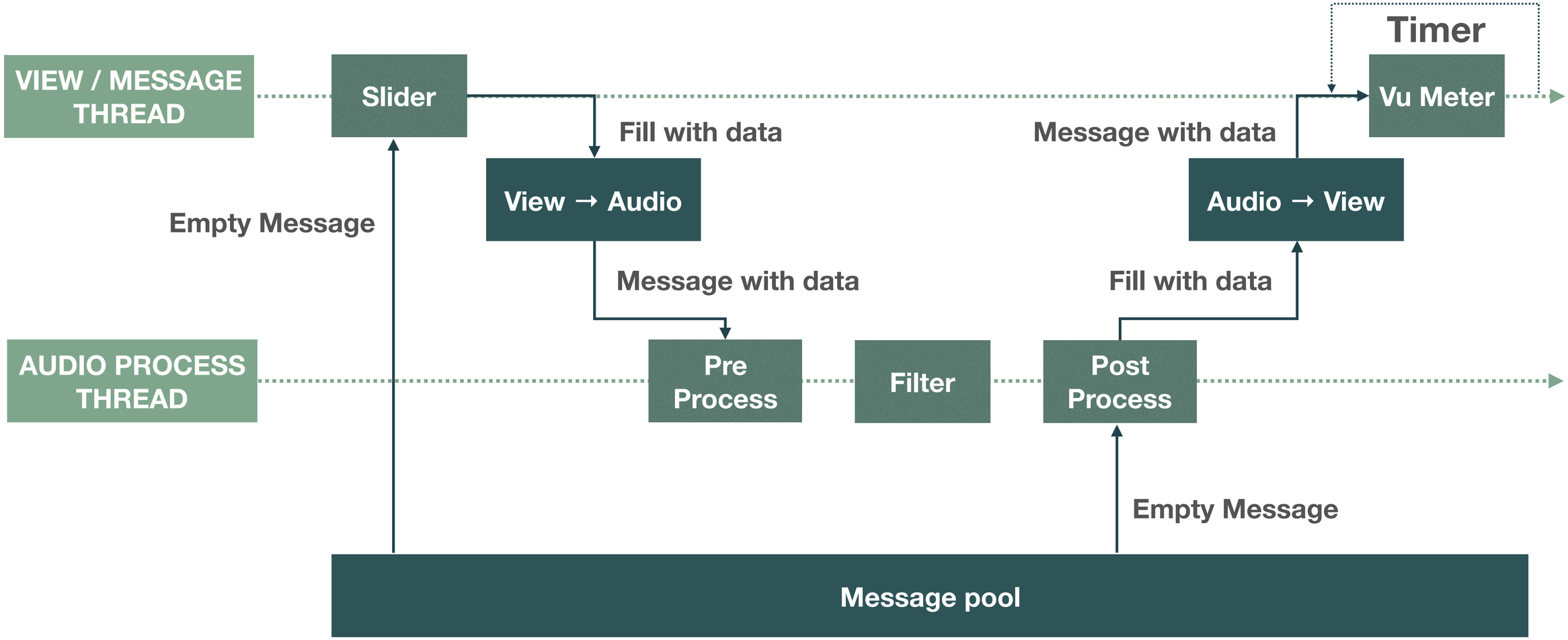
# Thread comms option 3



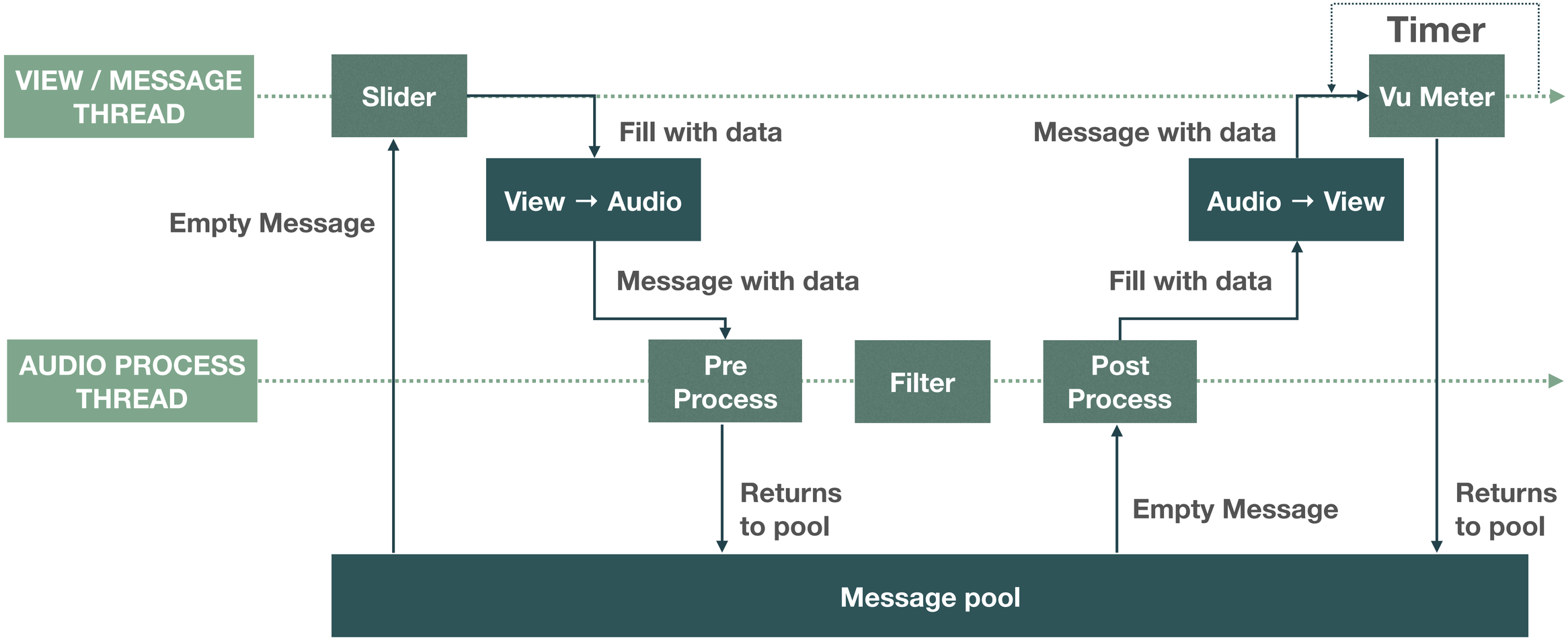
# Thread comms option 3



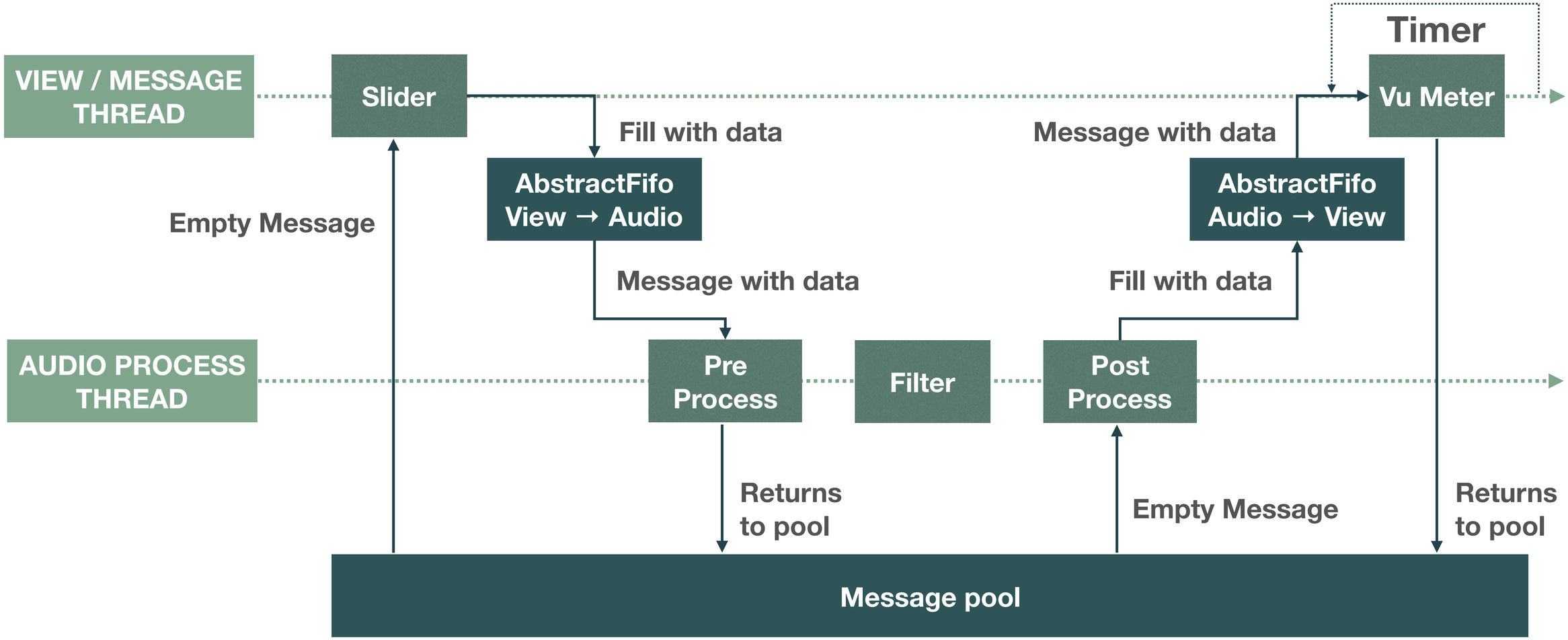
# Thread comms option 3



# Thread comms option 3



# Thread comms option 3



# Thread comms option 3

- Pre-allocate at construction a fixed number of messages (remember we don't want to allocate or deallocate in the process thread due to the global memory lock)

# Thread comms option 3

- Pre-allocate at construction a fixed number of messages (remember we don't want to allocate or deallocate in the process thread due to the global memory lock)
- Allows the audio thread to do high speed pointer switches, avoiding any change of a UI lock

# Q & A

# Questions we were asked over email…

How to make Components accessible to users with disabilities?

# Questions we were asked over email…

How to make Components accessible to users with disabilities?

- The JUCE team sends their apology for this one…

# Questions we were asked over email…

How to make Components accessible to users with disabilities?

- The JUCE team sends their apology for this one…
- But, there is lots you can do to help users that JUCE already supports

# Questions we were asked over email…

How to make Components accessible to users with disabilities?

- The JUCE team sends their apology for this one…
- But, there is lots you can do to help users that JUCE already supports
- Definitely recommend reading the platform accessibility guides:

# Questions we were asked over email…

How to make Components accessible to users with disabilities?

- The JUCE team sends their apology for this one…
- But, there is lots you can do to help users that JUCE already supports
- Definitely recommend reading the platform accessibility guides:
  - Apple https://developer.apple.com/library/content/documentation/Accessibility/Conceptual/AccessibilityMacOSX/
  - Microsoft https://developer.microsoft.com/en-us/windows/accessible-apps

# Questions we were asked over email…

Can i have JUCE 4 and 5 (and version X) next to each other and specify which to use?

# Questions we were asked over email…

Can i have JUCE 4 and 5 (and version X) next to each other and specify which to use?

- *YES!*

# Questions we were asked over email…

Can i have JUCE 4 and 5 (and version X) next to each other and specify which to use?

- *YES!*
- Keep each in its own folder
- Use the corresponding Projucer

# Thank you for your time and attention

# Any questions?


**Thank you**

**Paul Chana / Vlad Voina**