VisTrails Packages

Making VisTrails work with your own software

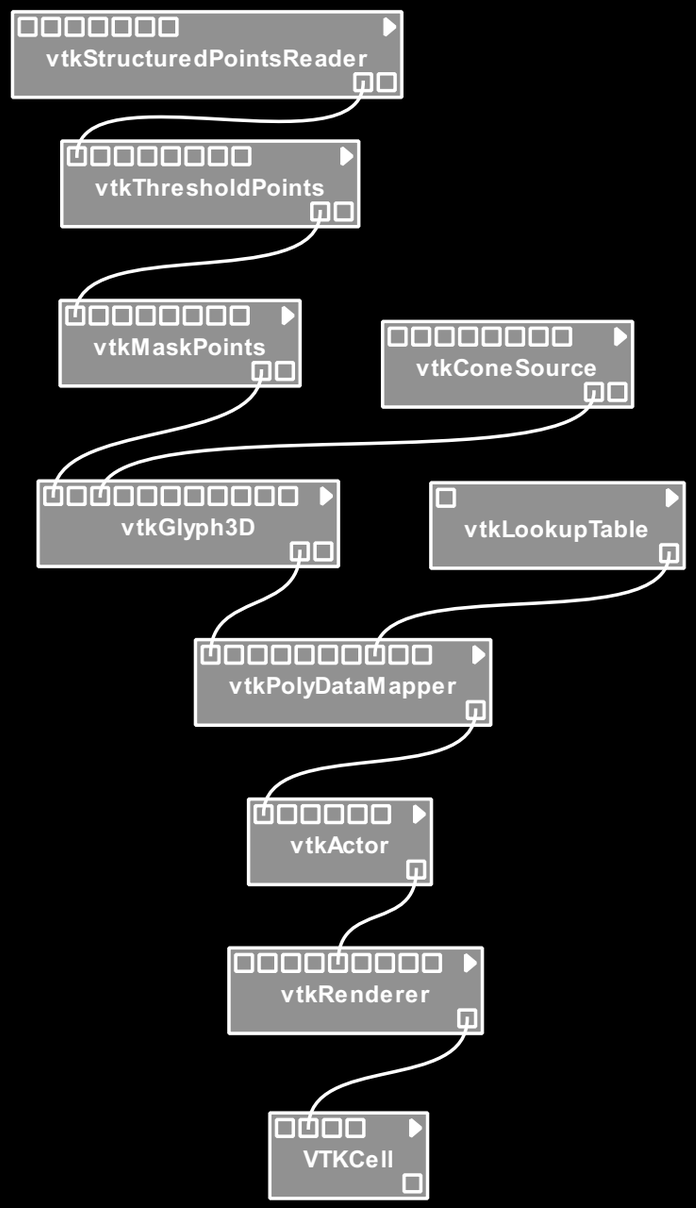
Basic VisTrails module Infrastructure

- Only "special" modules are the **Basic Modules**
  - Module, Integer, File, Variant, PythonSource
- **Everything** else is in packages
- *show packages around*

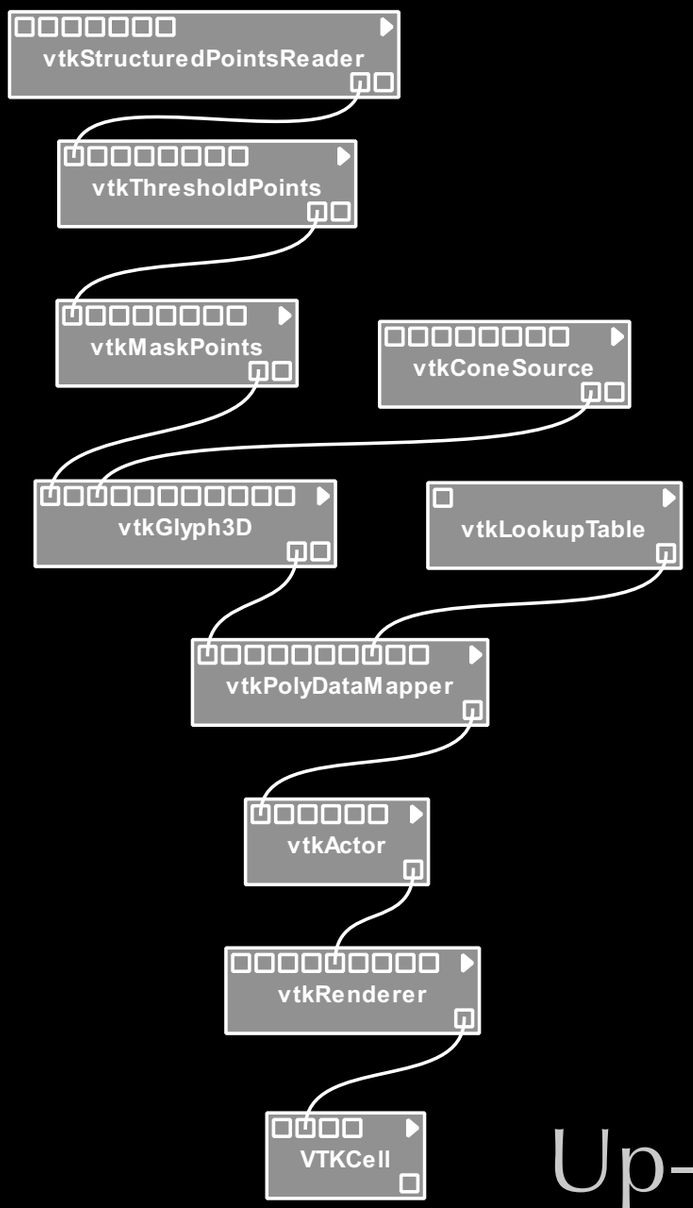
Execution Model

- Modules become **up-to-date** by being executed
  - Upstream modules are updated first
- Bottom-up activation, top-down execution

Execution Model

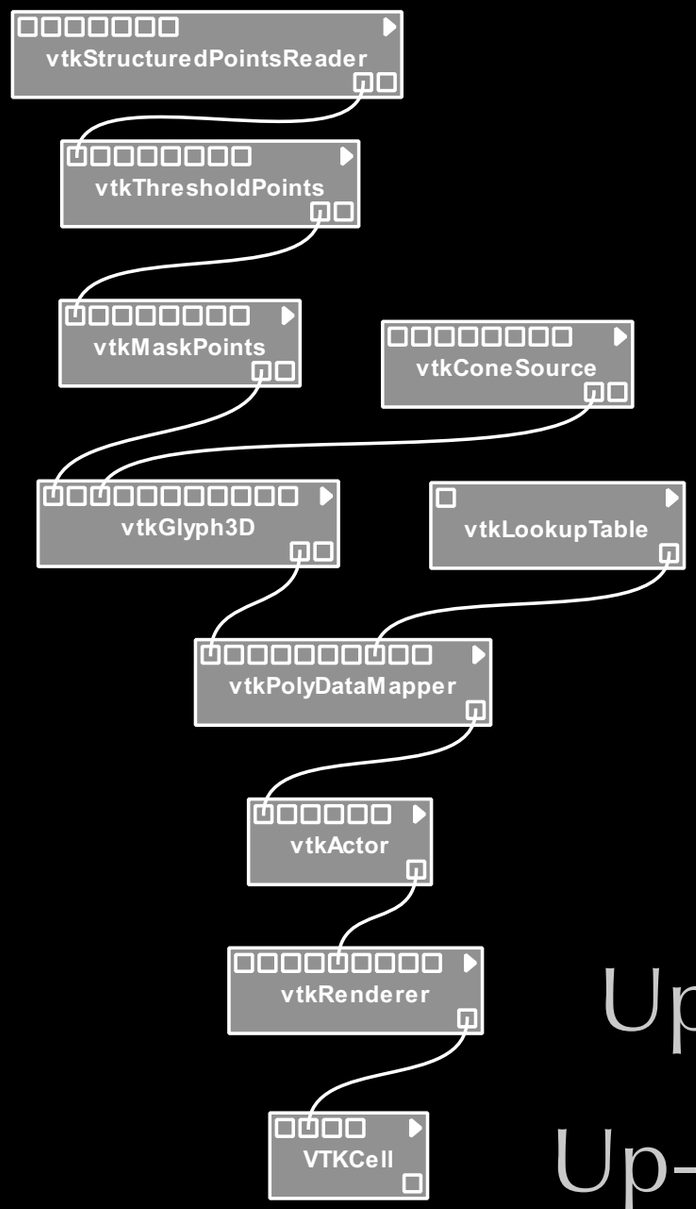


Execution Model



Up-to-date?

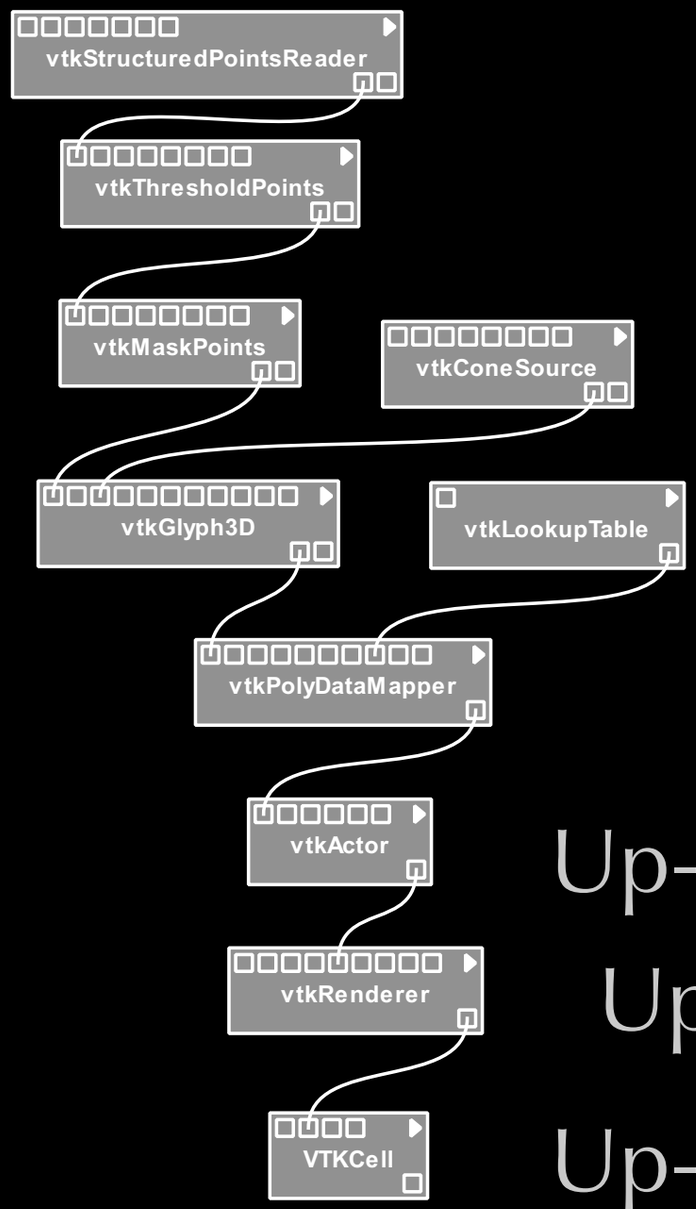
Execution Model



Up-to-date?

Up-to-date?

Execution Model

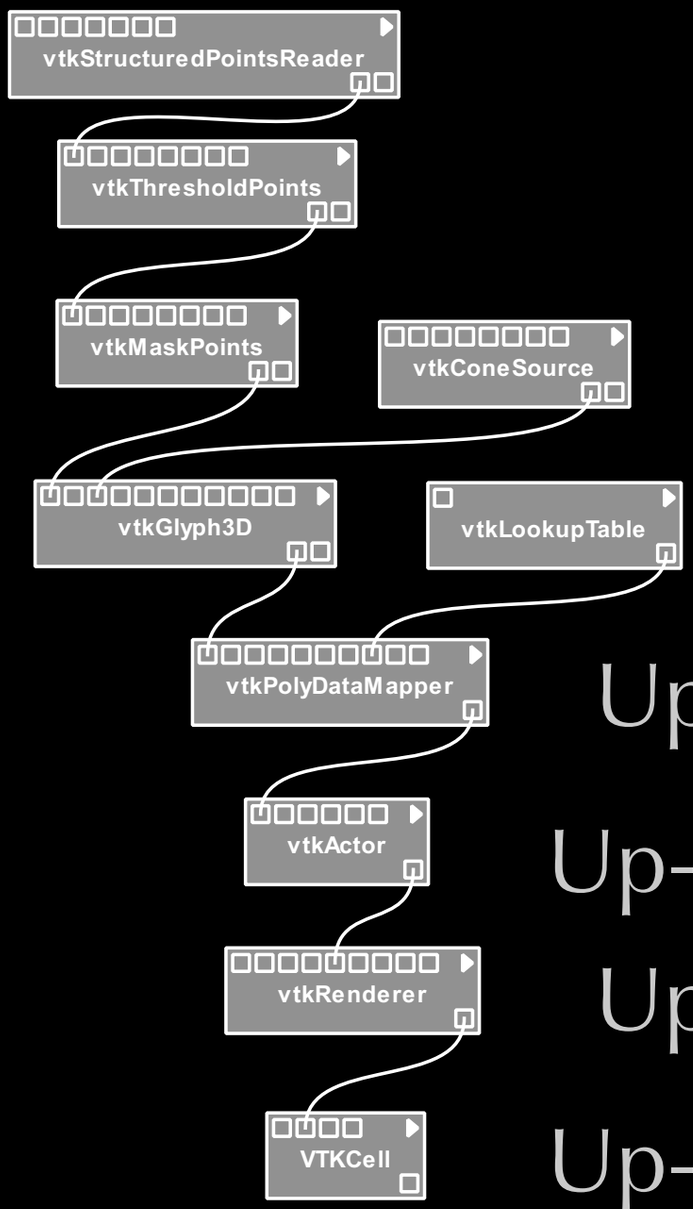


Up-to-date?

Up-to-date?

Up-to-date?

Execution Model



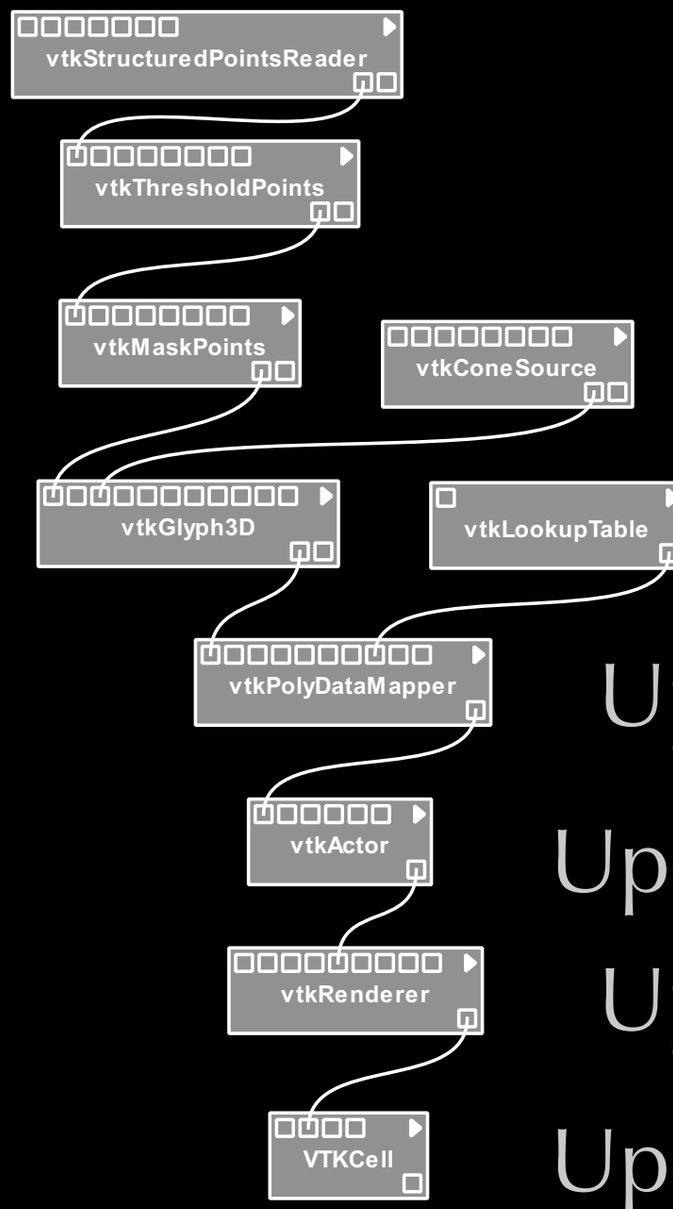
Up-to-date?

Up-to-date?

Up-to-date?

Up-to-date?

Execution Model



Up-to-date?

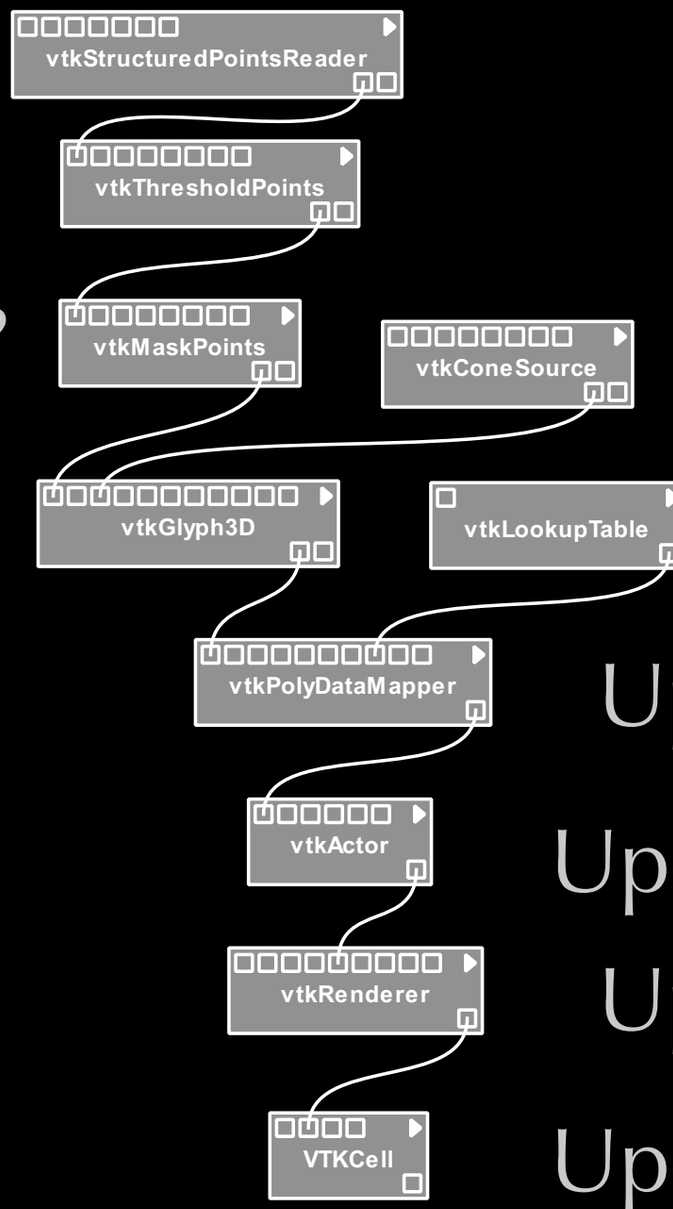
Up-to-date?

Up-to-date?

Up-to-date?

Up-to-date?

Execution Model



Up-to-date?

Up-to-date?

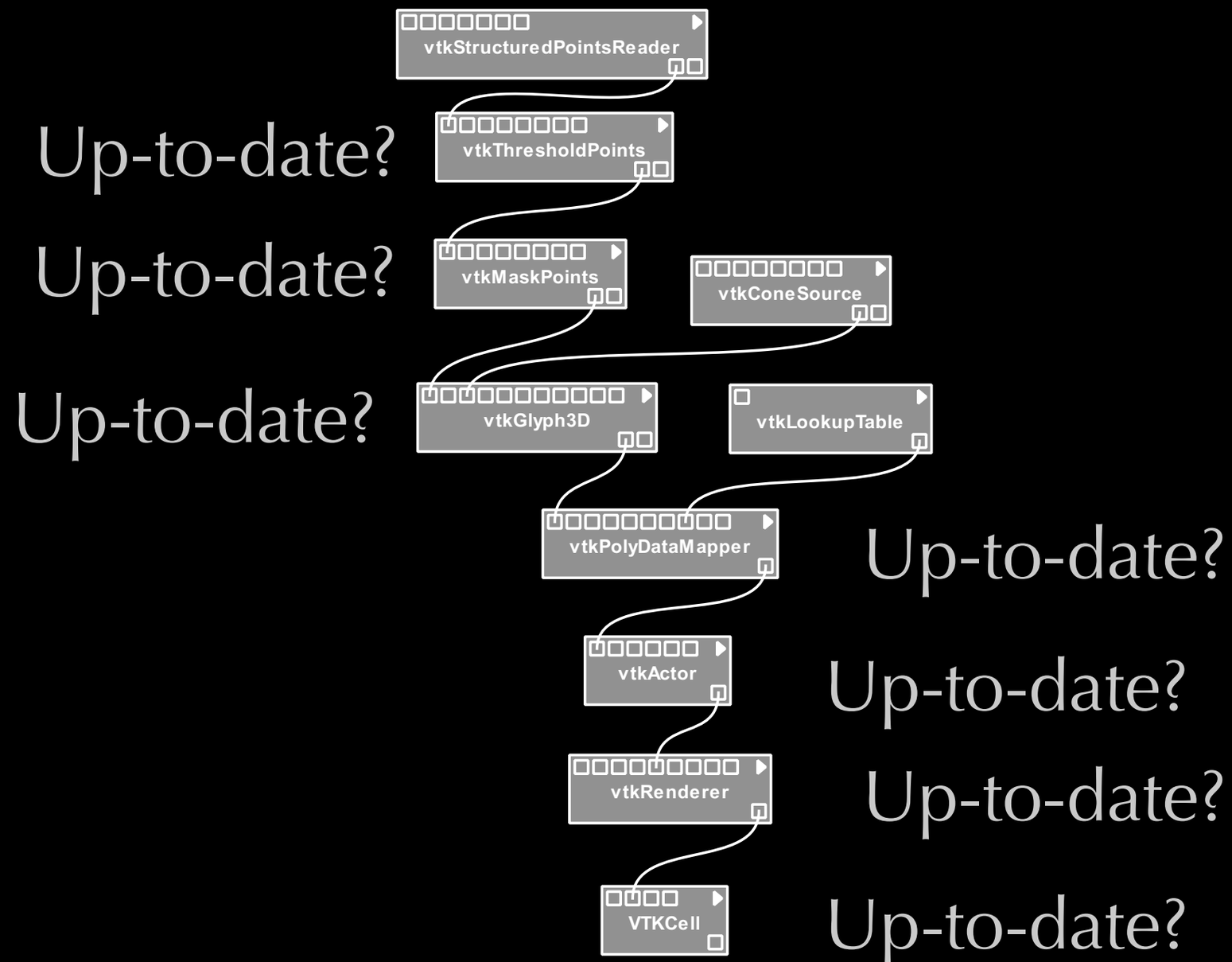
Up-to-date?

Up-to-date?

Up-to-date?

Up-to-date?

Execution Model



Up-to-date?

Up-to-date?

Up-to-date?

Up-to-date?

Up-to-date?

Up-to-date?

Execution Model

Up-to-date?

Up-to-date?

Up-to-date?

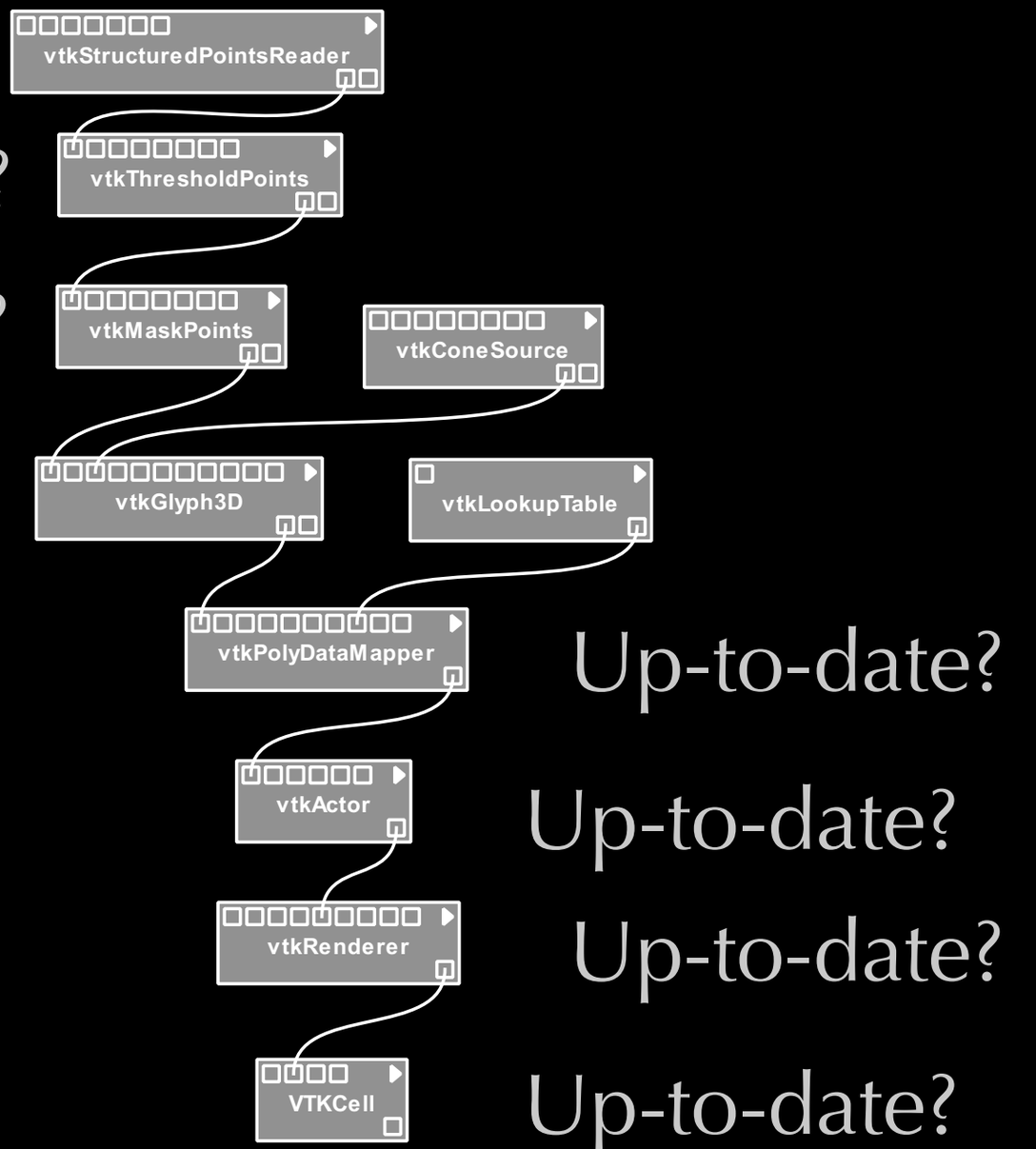
Up-to-date?

Up-to-date?

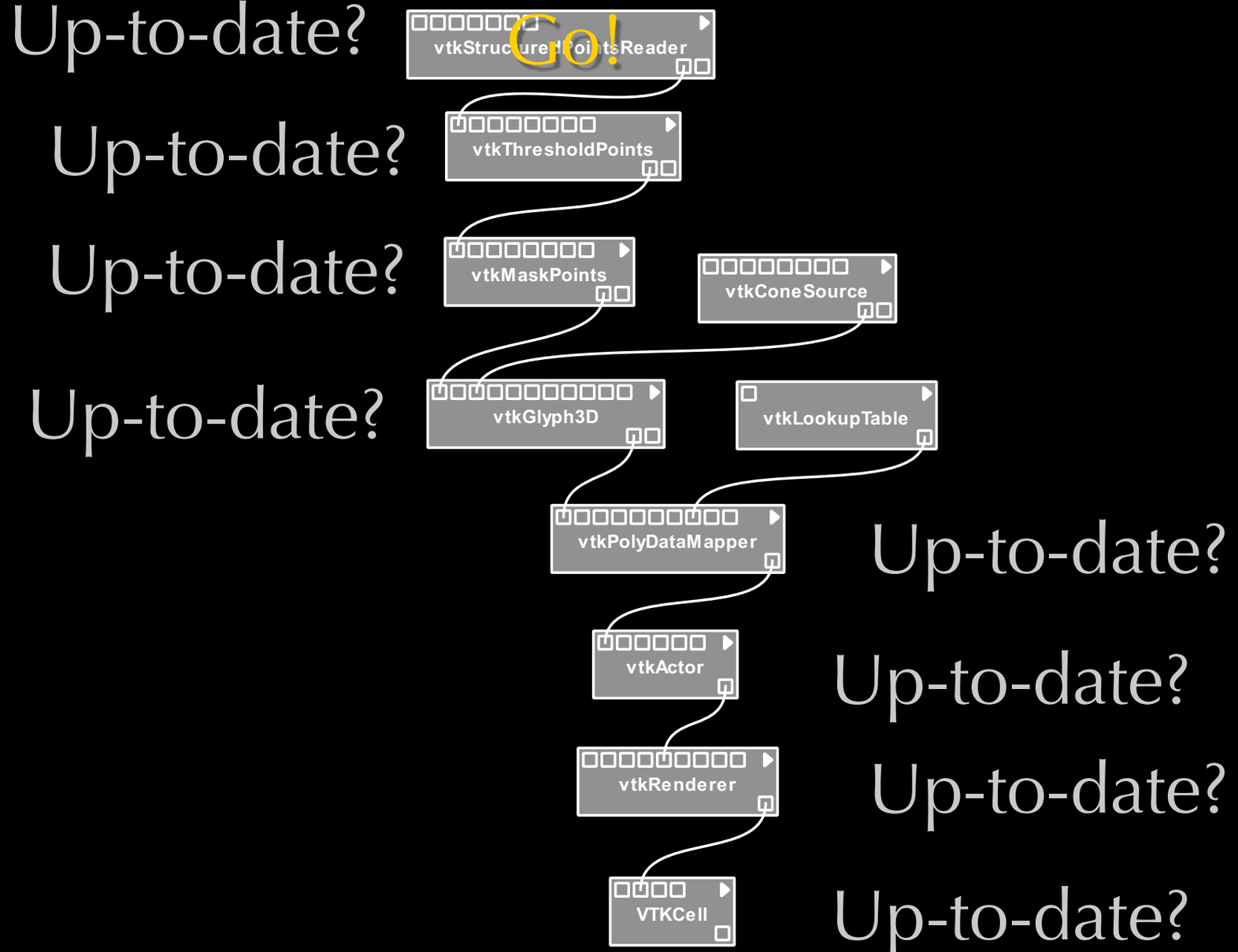
Up-to-date?

Up-to-date?

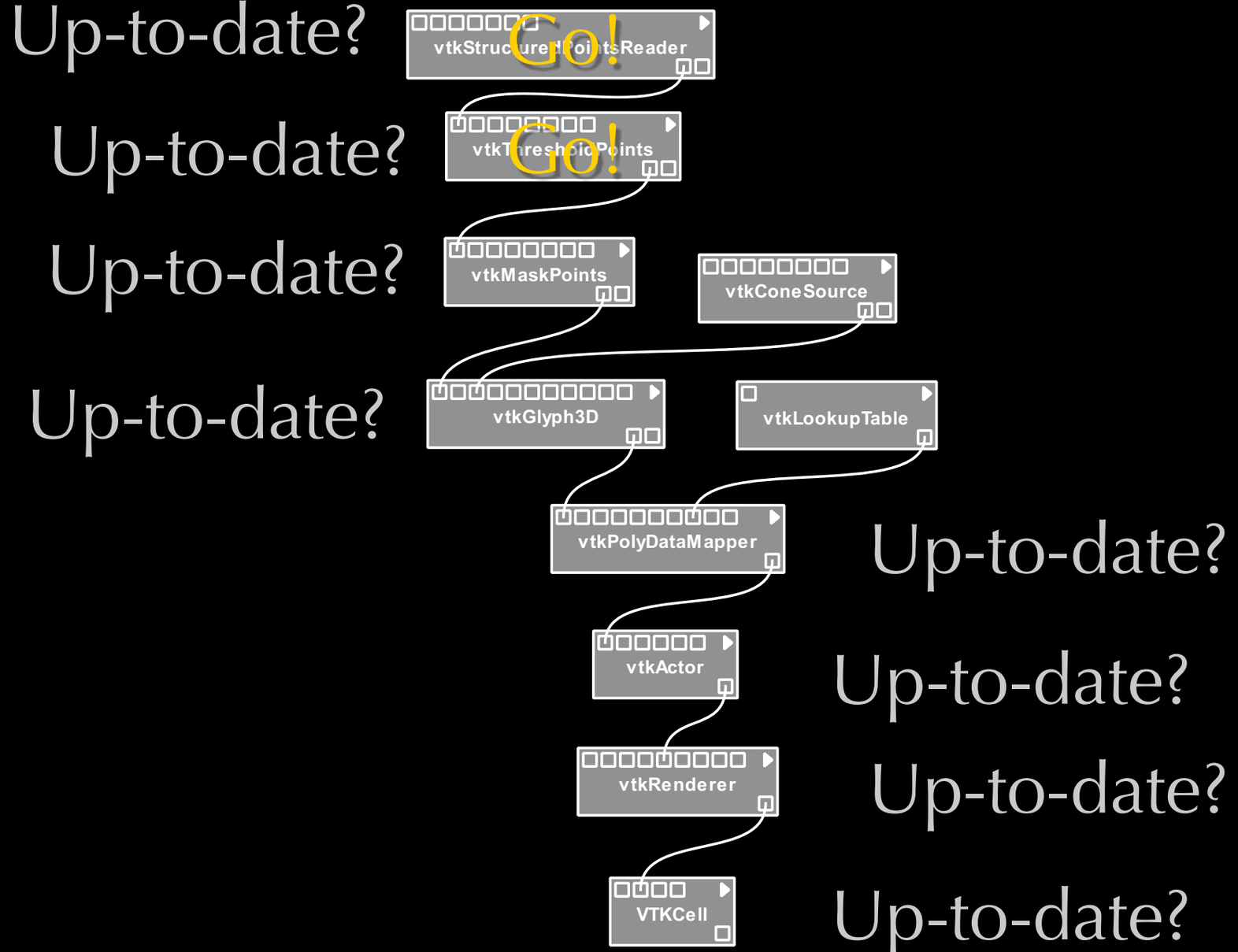
Up-to-date?



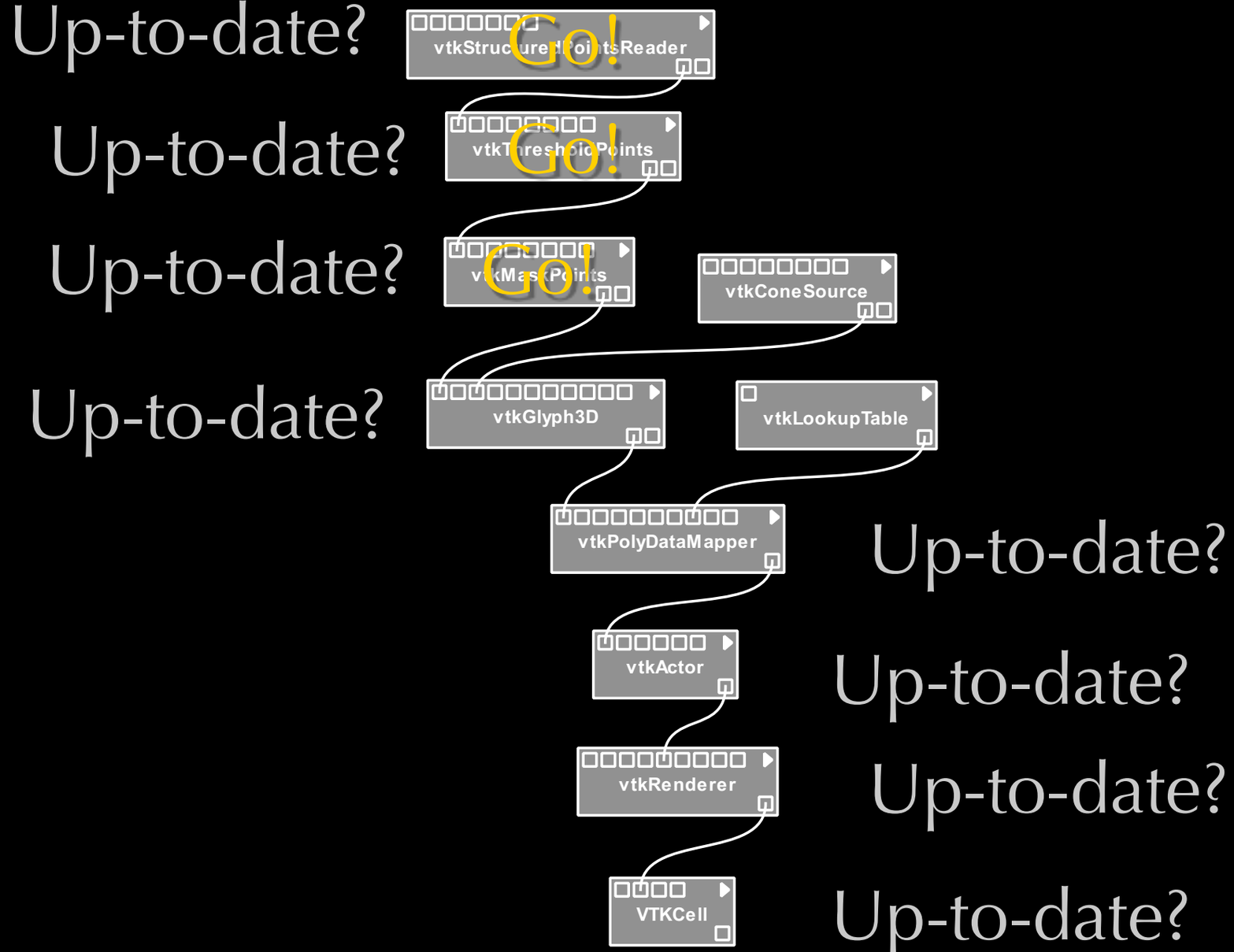
Execution Model



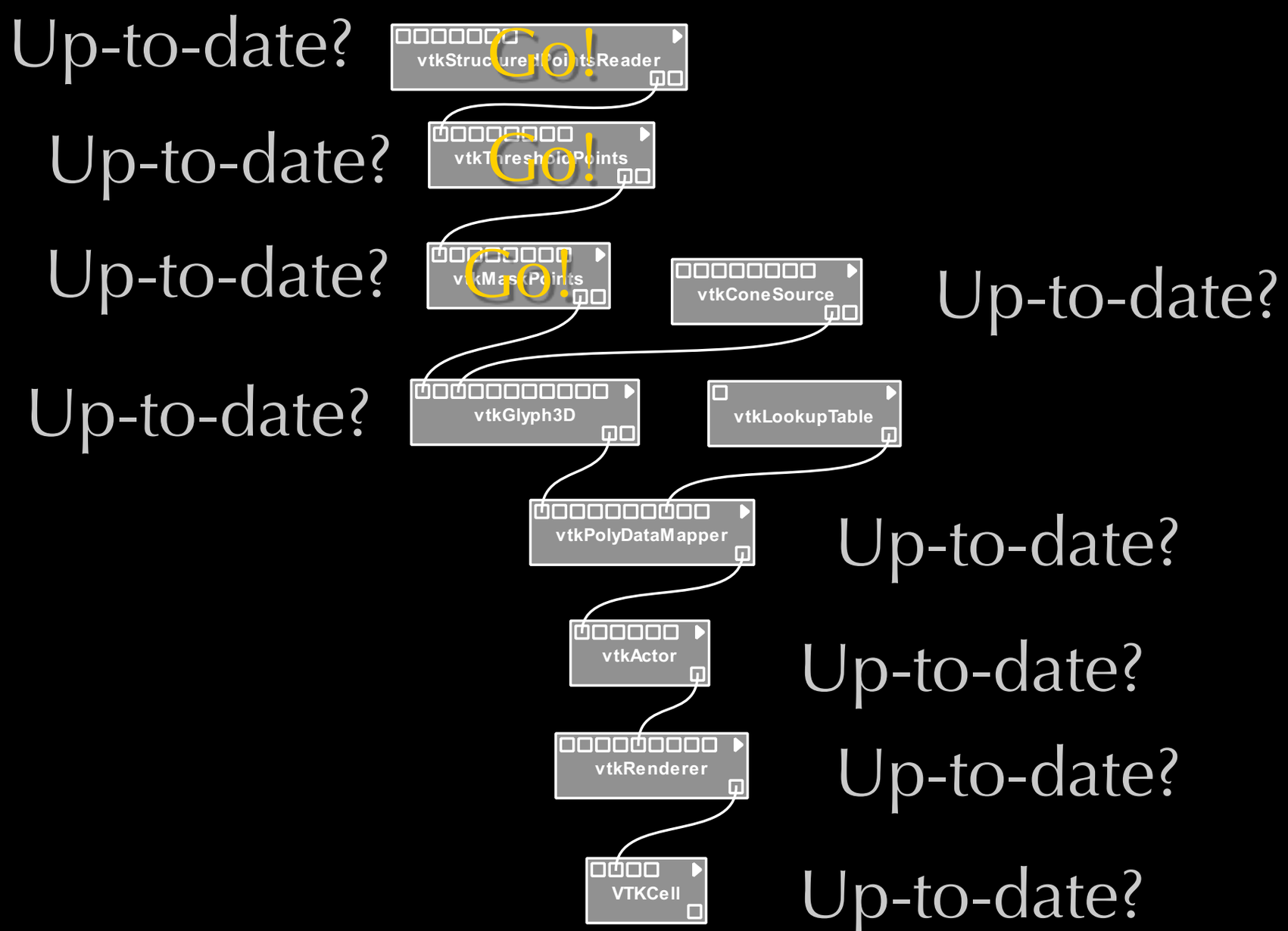
Execution Model



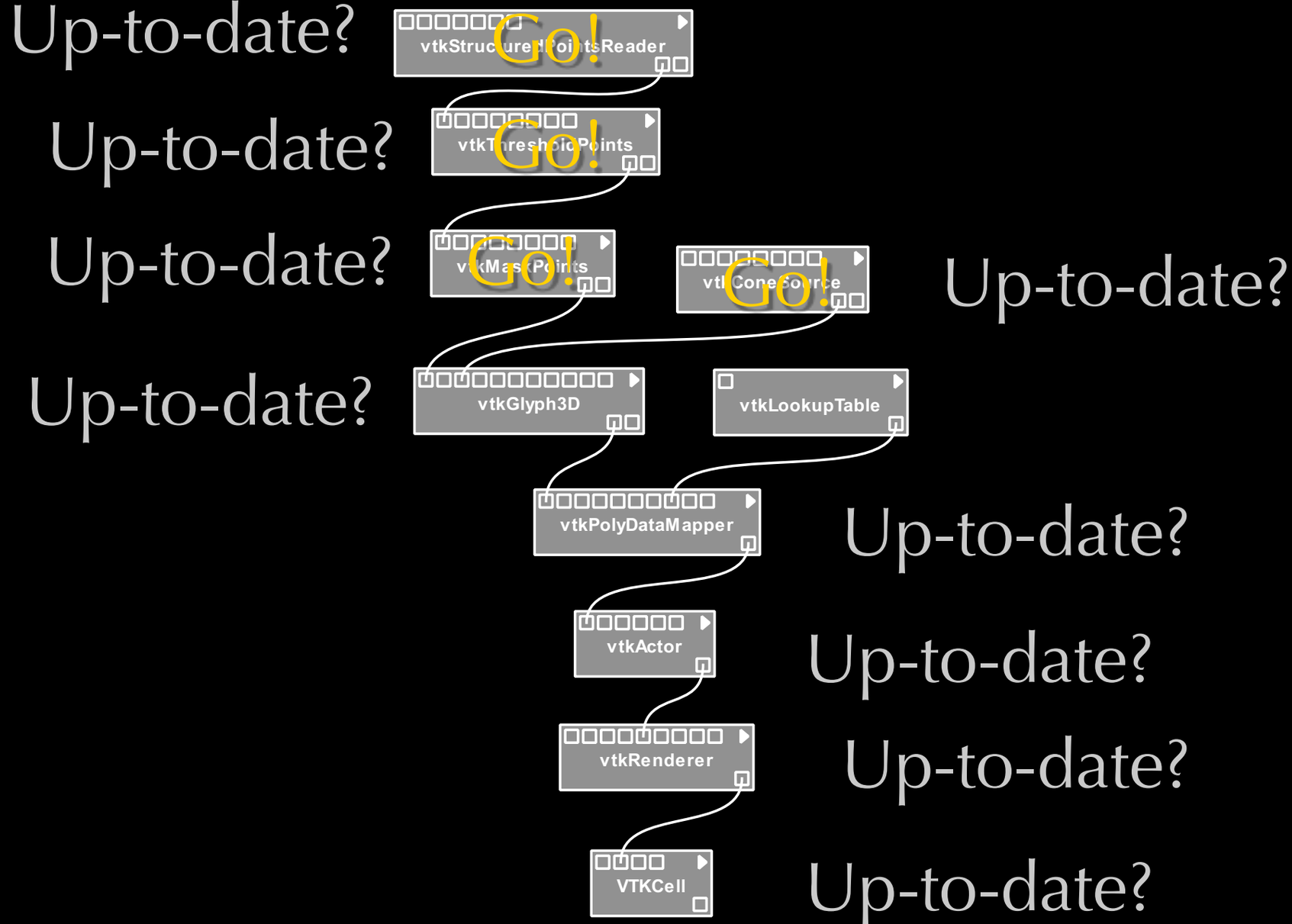
Execution Model



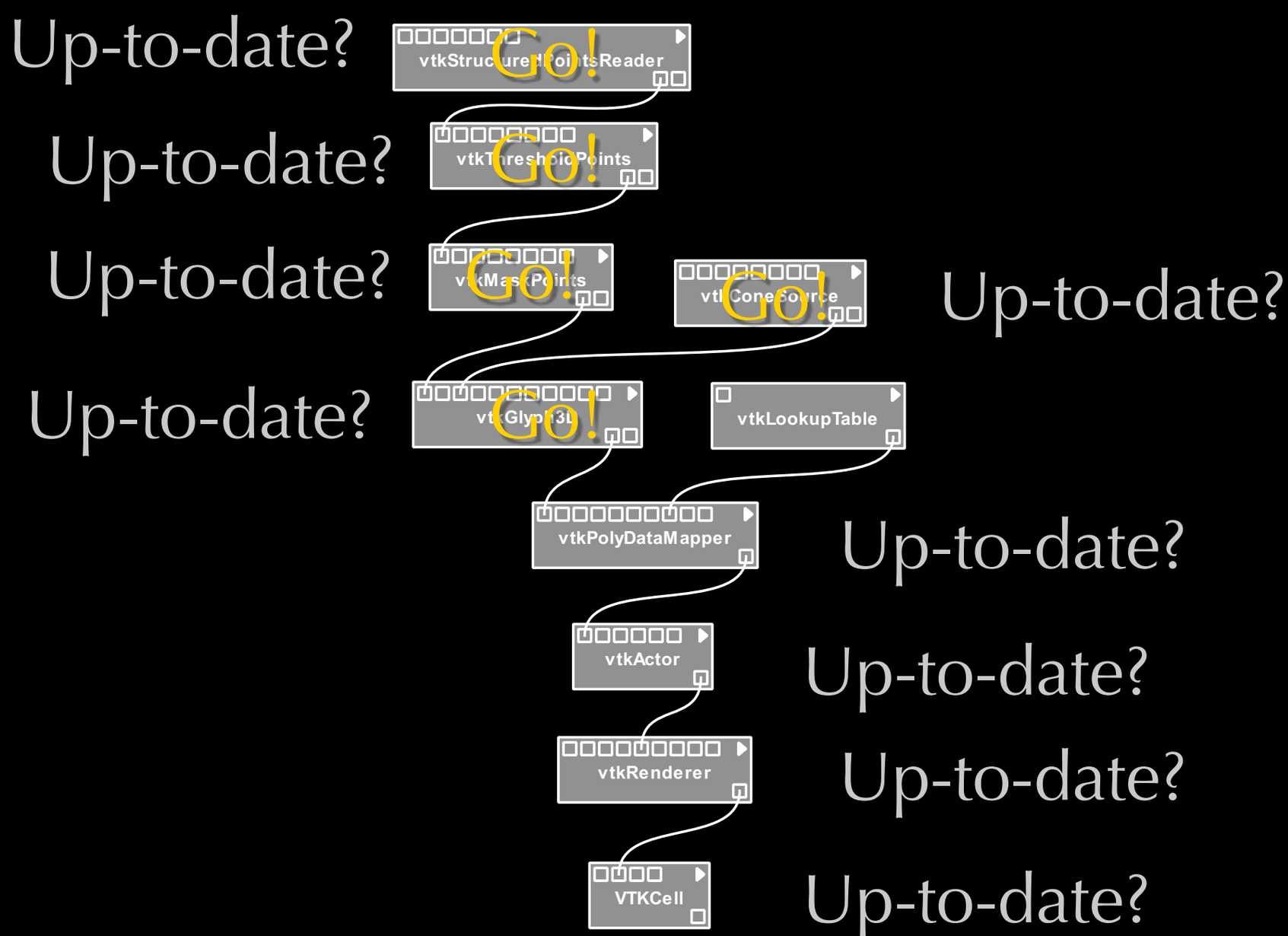
Execution Model



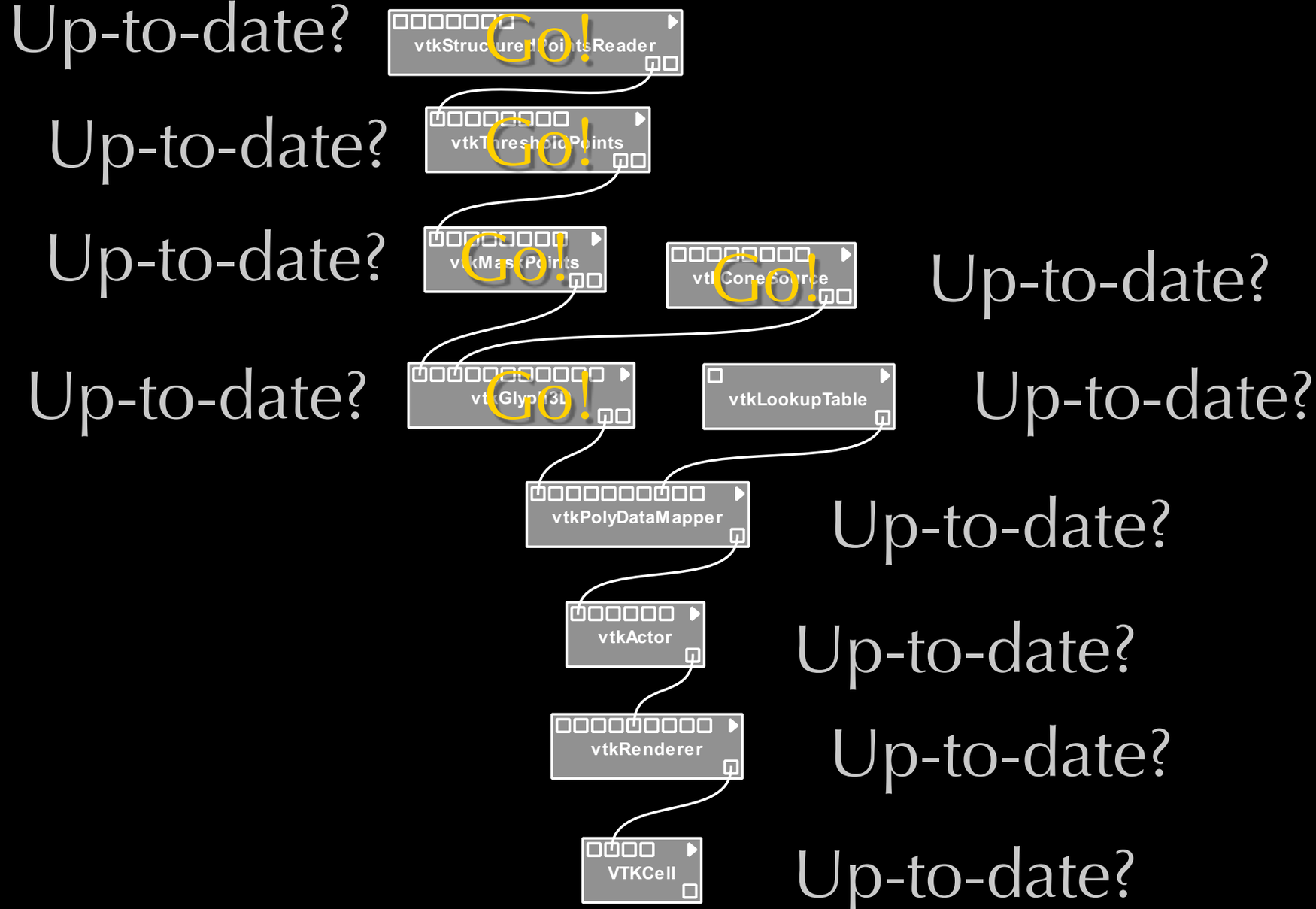
Execution Model



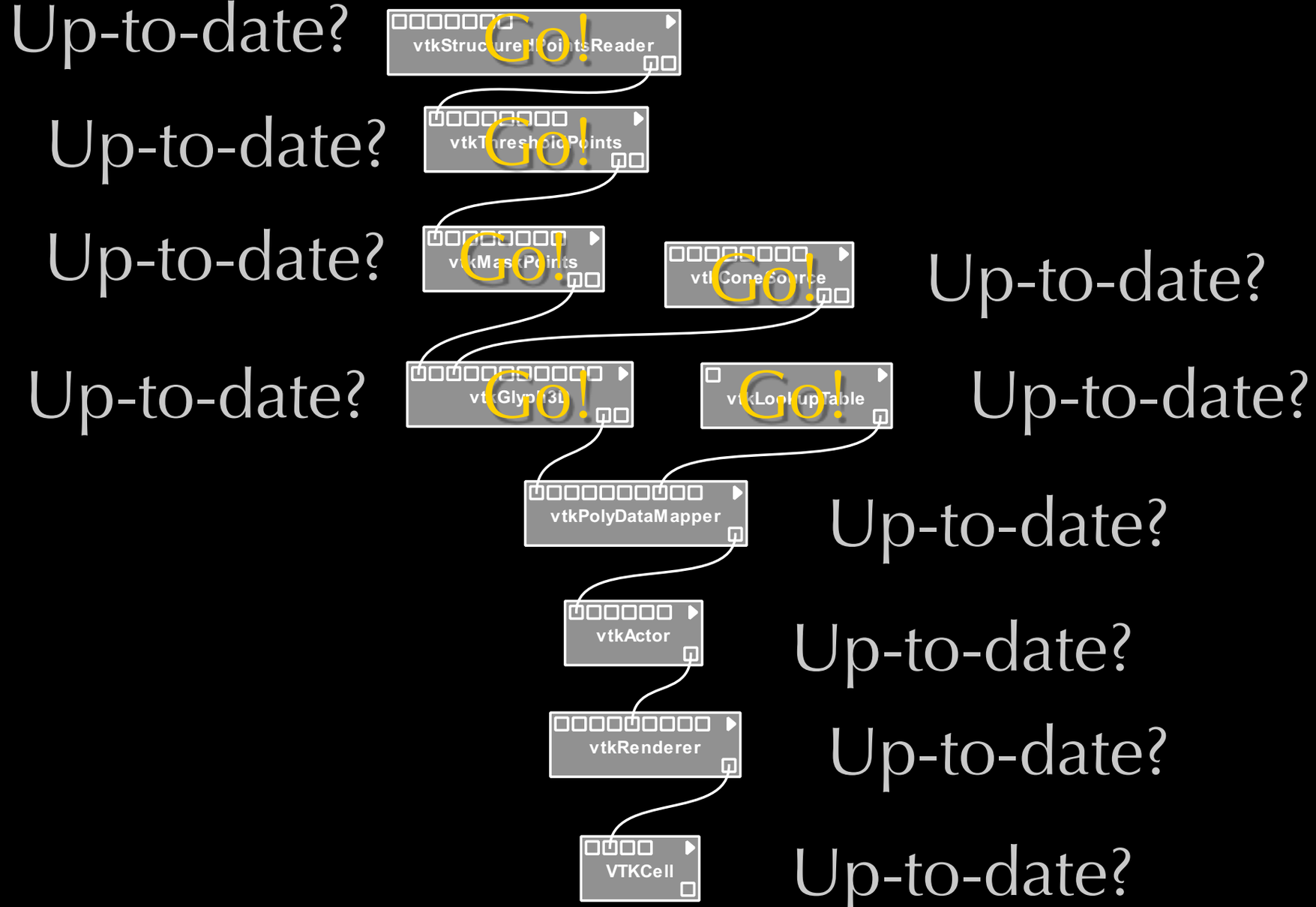
Execution Model



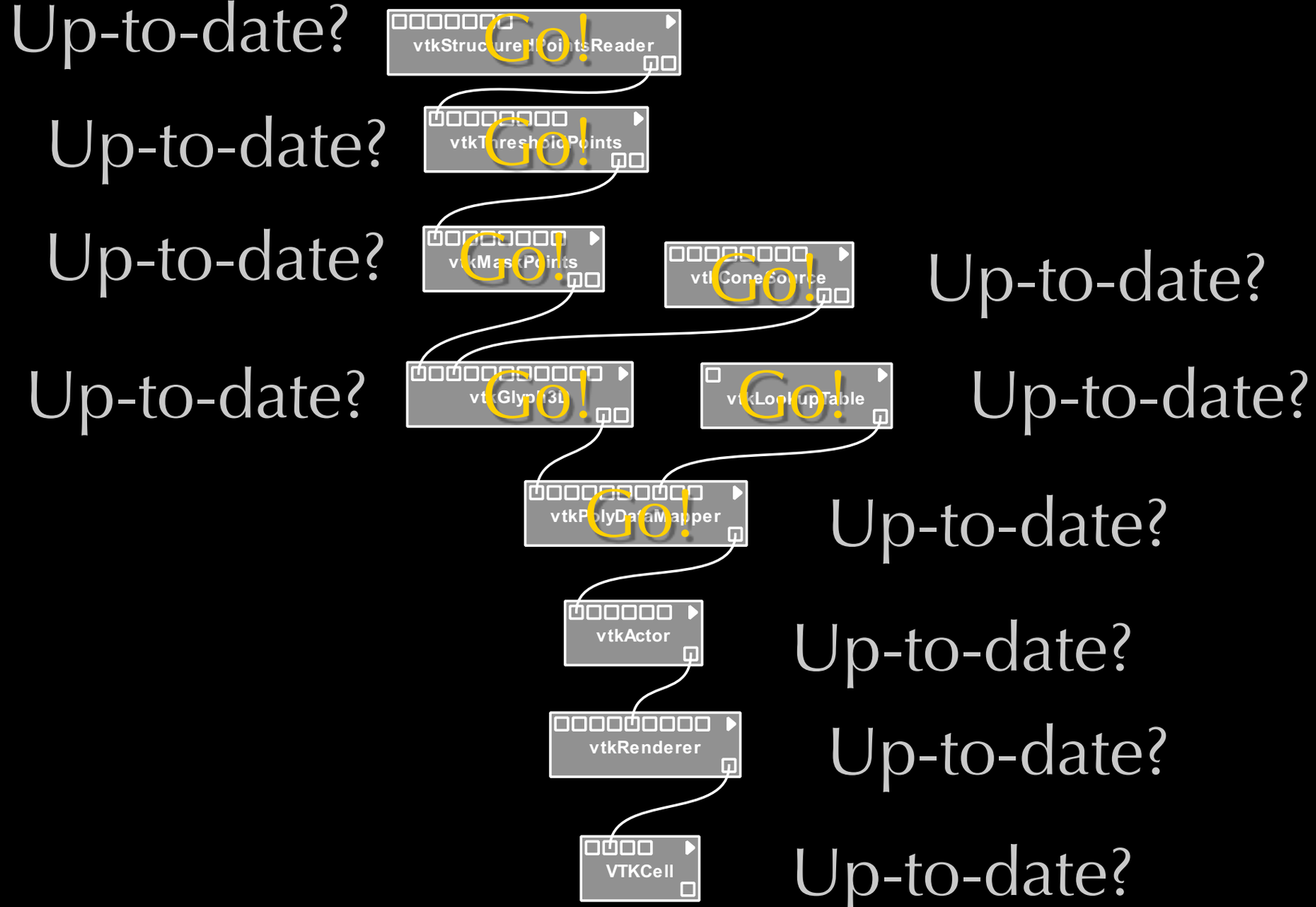
Execution Model



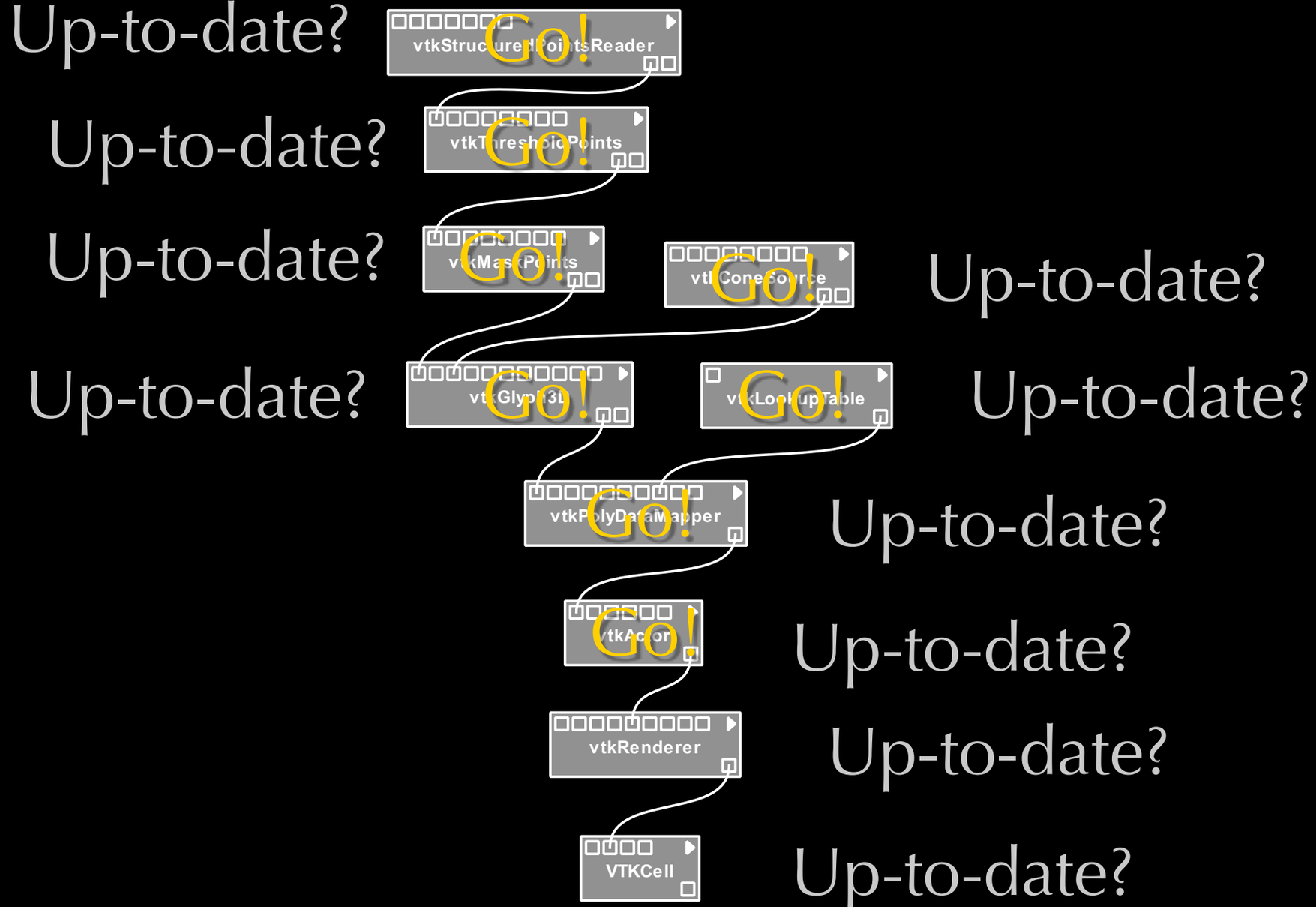
Execution Model



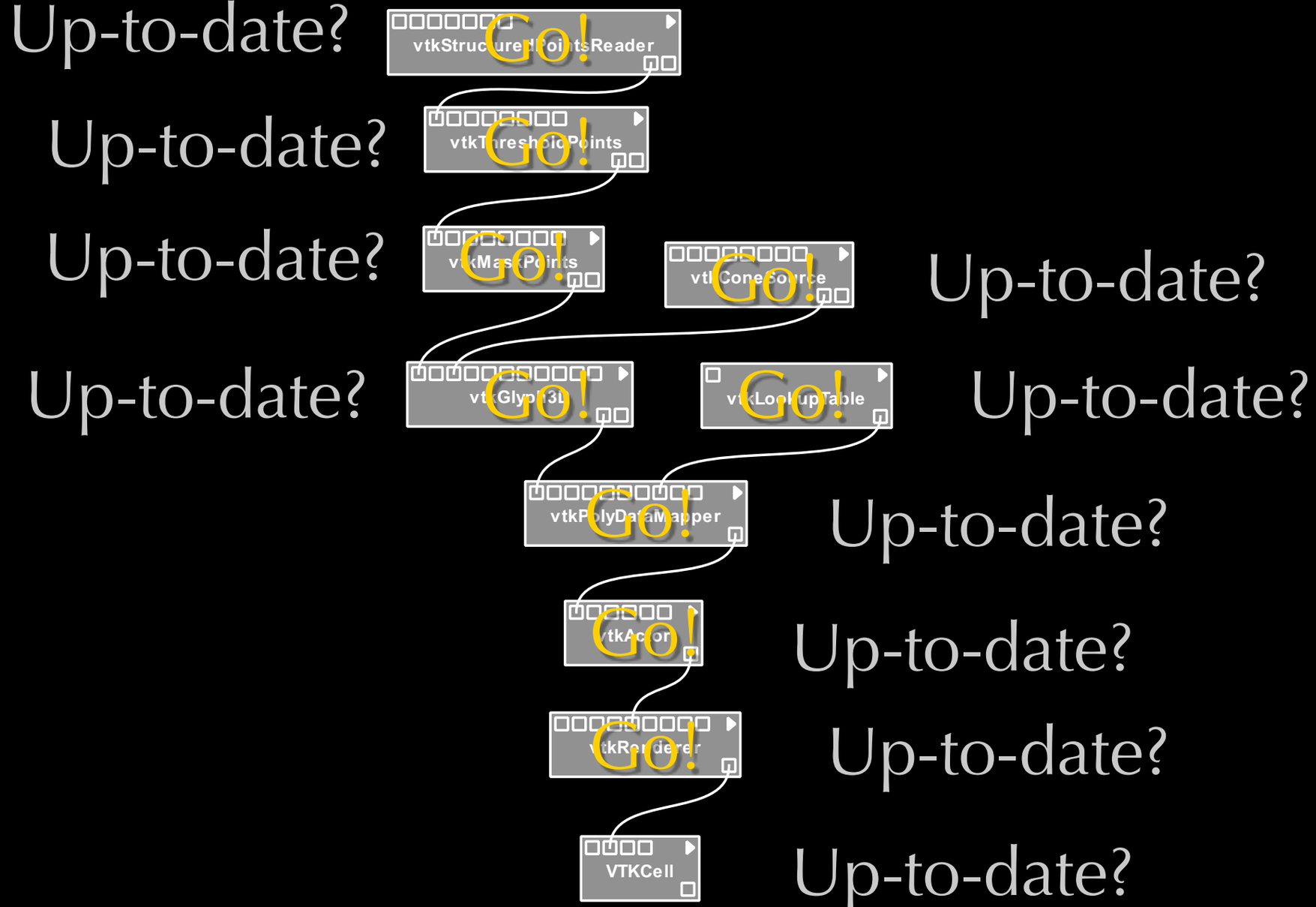
Execution Model



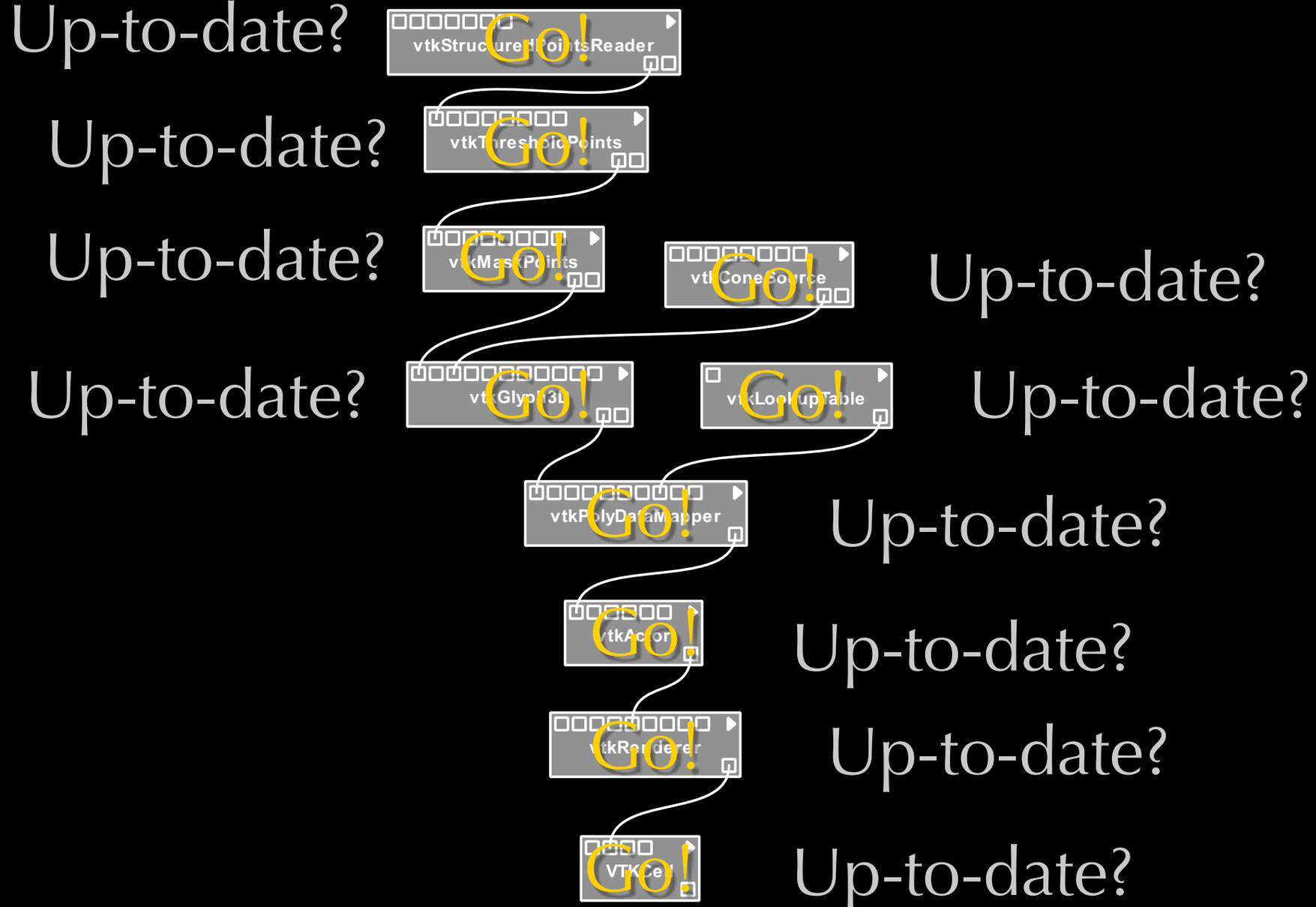
Execution Model



Execution Model



Execution Model



Writing Modules

```
class MeshFilter(Module):

    def guess_input_format(self, input_file_name):
        i = input_file_name.rfind('.')
        if i == -1:
            return None
        else:
            return input_file_name[i:]

    def compute(self):
        self.checkInputPort('input_file')
        input_file = self.getInputFromPort('input_file')
        if self.hasInputFromPort('output_format'):
            output_suffix = self.getInputFromPort('output_format')
        else:
            output_suffix = self.guess_input_format(input_file.name)
        if not output_suffix:
            output_suffix = '.off'
        output_file = self.interpreter.filePool.create_file(suffix=output_suffix)
        values = [_mesh_filter_path,
                  input_file.name,
                  output_file.name]
        cmdline = list2cmdline(values)
        print cmdline
        result = os.system(cmdline)
        if result != 0:
            raise ModuleError(self, 'Execution failed')
        self.setResult('output_file', output_file)
```

- a VisTrails module is a Python class

Writing Modules

- Subclass of `basic_modules.Module`
- All work happens on `compute`
  - only method you really **have** to write
  - should get input from ports, do stuff, then write results to ports
- Declare interface to VisTrails

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

Hello, world!

Module subclass

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

`compute` method

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

getting inputs

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
_input_ports = [('value1', [(basic.Float, '')]),
                ('value2', [(basic.Float, '')]),
                ('op', [(basic.String, '')])]
_output_ports = [('value', [(basic.Float, '')])]
```

setting outputs

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
_input_ports = [('value1', [(basic.Float, '')]),
                ('value2', [(basic.Float, '')]),
                ('op', [(basic.String, '')])]
_output_ports = [('value', [(basic.Float, '')])]
```

declaring interface

Hello, world!

```
class PythonCalc(Module):
    def compute(self):
        def op(v1, v2):
            op = self.getInputFromPort("op")
            if op == '+': return v1 + v2
            elif op == '-': return v1 - v2
            elif op == '*': return v1 * v2
            elif op == '/': return v1 / v2
        v1 = self.getInputFromPort("value1")
        v2 = self.getInputFromPort("value2")
        self.setResult("value", op(v1, v2))
    _input_ports = [('value1', [(basic.Float, '')]),
                    ('value2', [(basic.Float, '')]),
                    ('op', [(basic.String, '')])]
    _output_ports = [('value', [(basic.Float, '')])]
```

Input/Output

- `getInputFromPort`
  - Returns instance of (subclass of) Module in most cases, but actual simple python value if value is a “constant” (Integer, String, Float, Boolean, etc) ***more later***
- optional ports?
  - `hasInputFromPort`

Input/Output

- `setResult`
  - Use type you advertised
- `setResult("foo", self)` is ok
  - and sometimes useful

Modules go in packages

- Register package modules with VisTrails

```
_modules = [PythonCalc]
```

- Meta-data: package name, version, identifier

```
version = '0.1.0'
identifier = 'org.vistrails.pythoncalc'
name = 'PythonCalc'
```

Temporary Files

- Many modules communicate through files
- VisTrails has a basic facility for temporary file management
  - "GC" for files

Temporary Files

```
def compute(self):
    o = self.interpreter.filePool.create_file(suffix='.m')
```

- Creates a VisTrails File object
  - `o.name` is the name of a file in the filesystem that can be written to
  - use `o` itself as the output:

```
self.setResult("output", o)
```

  - VisTrails will manage file's lifetime

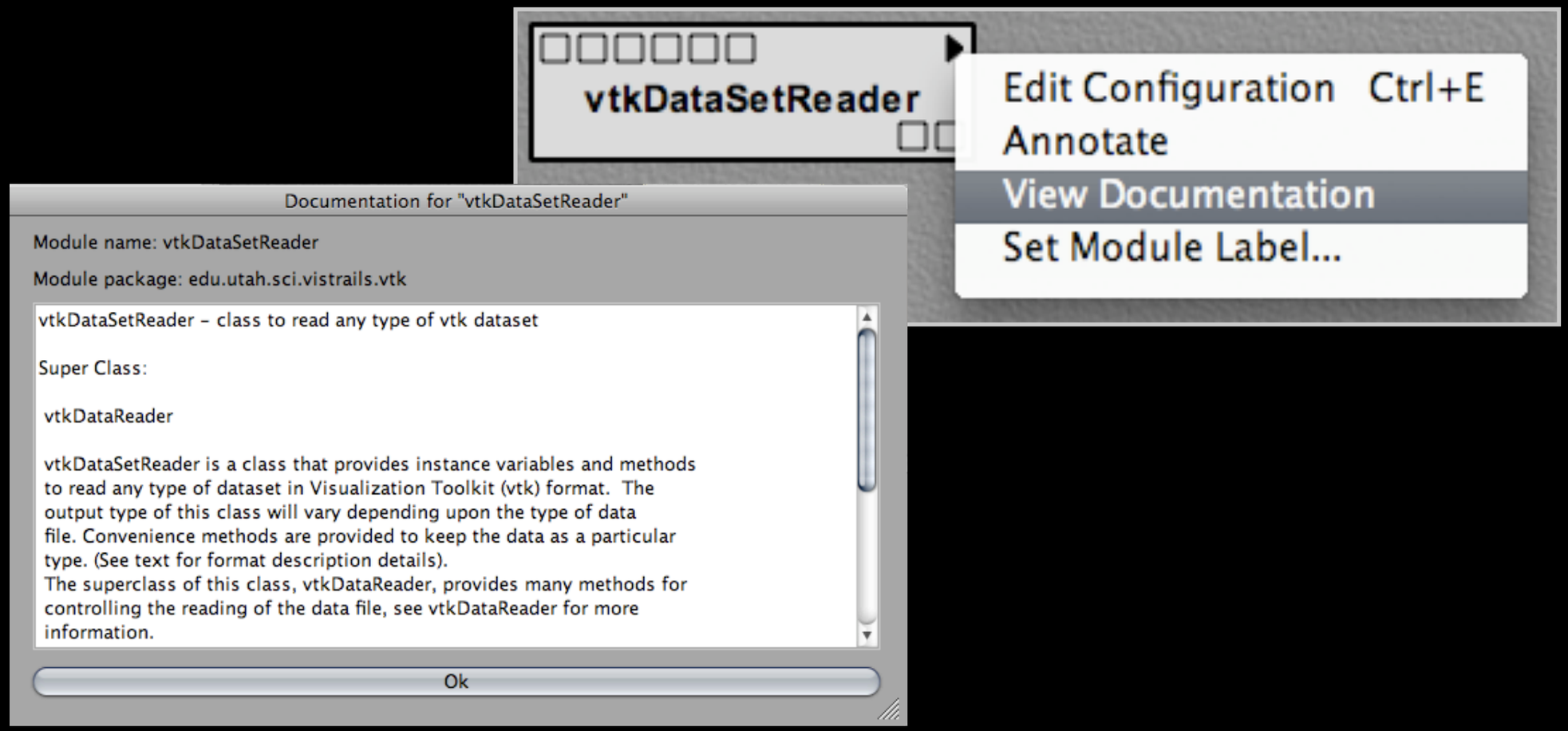
Error Handling

- `compute` can signal errors:

```
if not self.hasInputFromPort("file"):
    raise ModuleError(self, "Needs input file")
```

- Caveat: if library likes to segfault, we suggest wrapping it in a different binary and calling a new process

Documentation



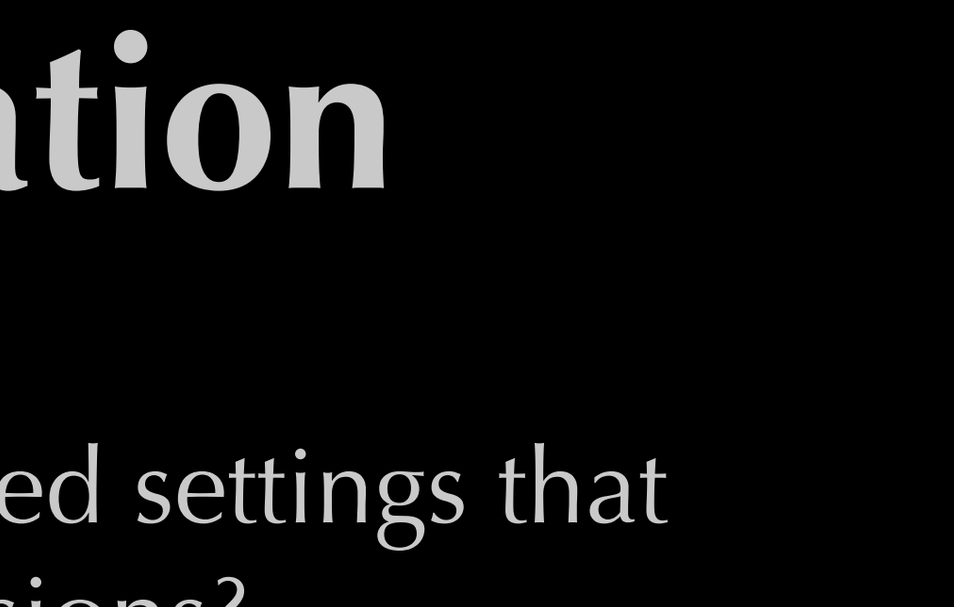
- We just inspect class docstrings

Configuration

- Need to store user-defined settings that are persistent across sessions?

```
configuration = ConfigurationObject(path=(None, str),
                                    debug=False)
```

- user interacts from GUI



Simple Complete Example

- afront: Command-line utility

- *walk through package code*

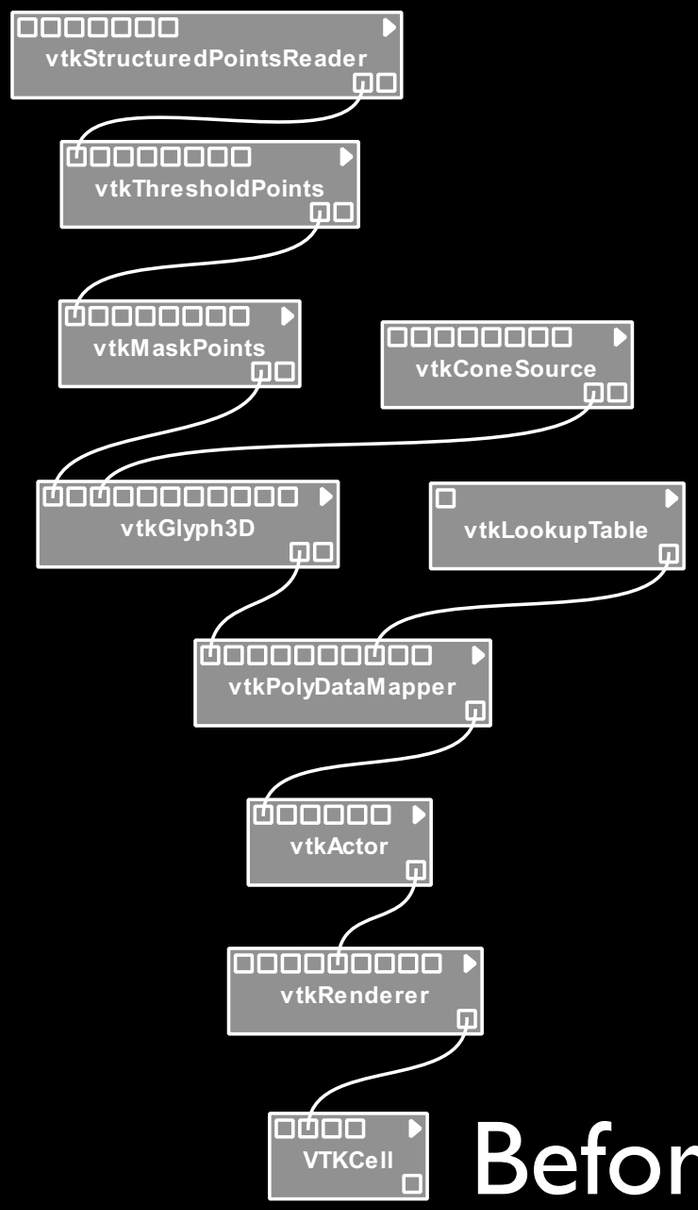
Where does it go?

- User package or system package?

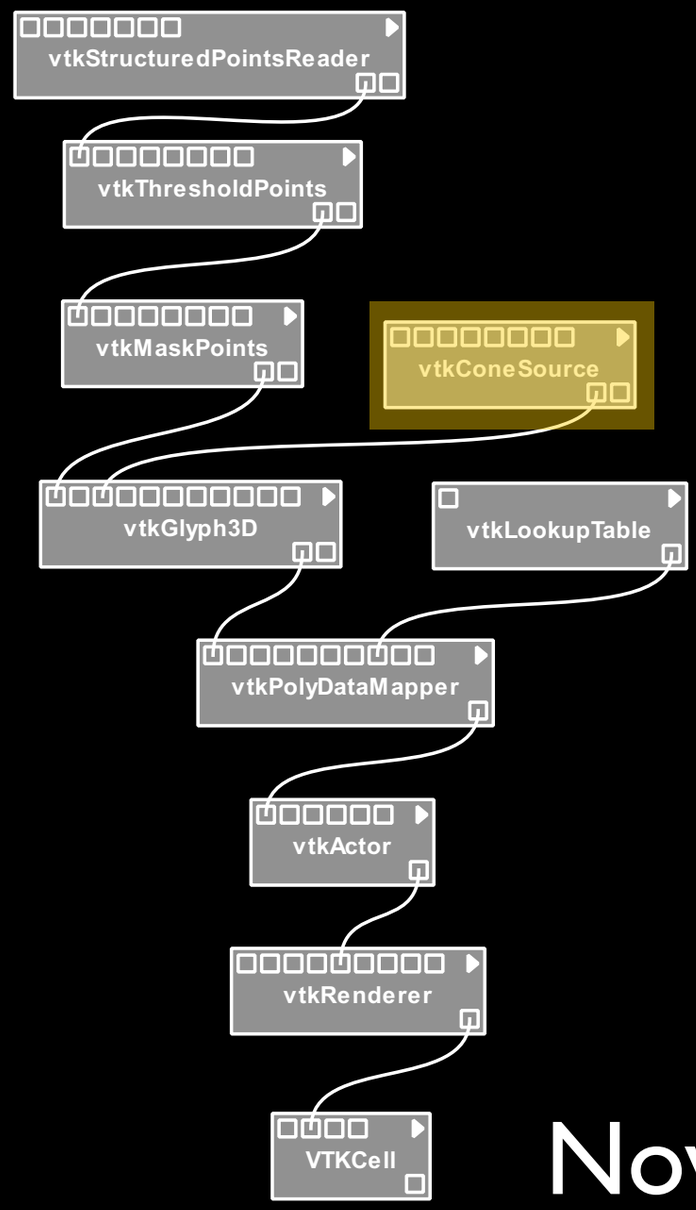
Caching

- VisTrails does not recompute results if possible

Caching



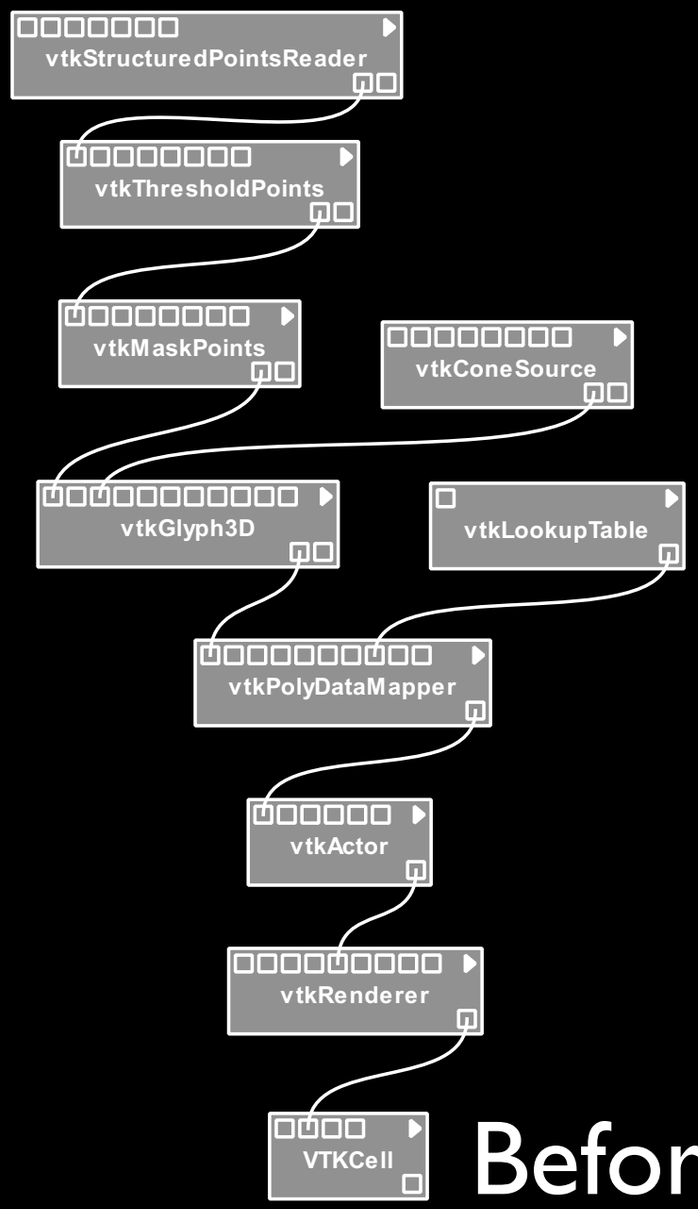
Before



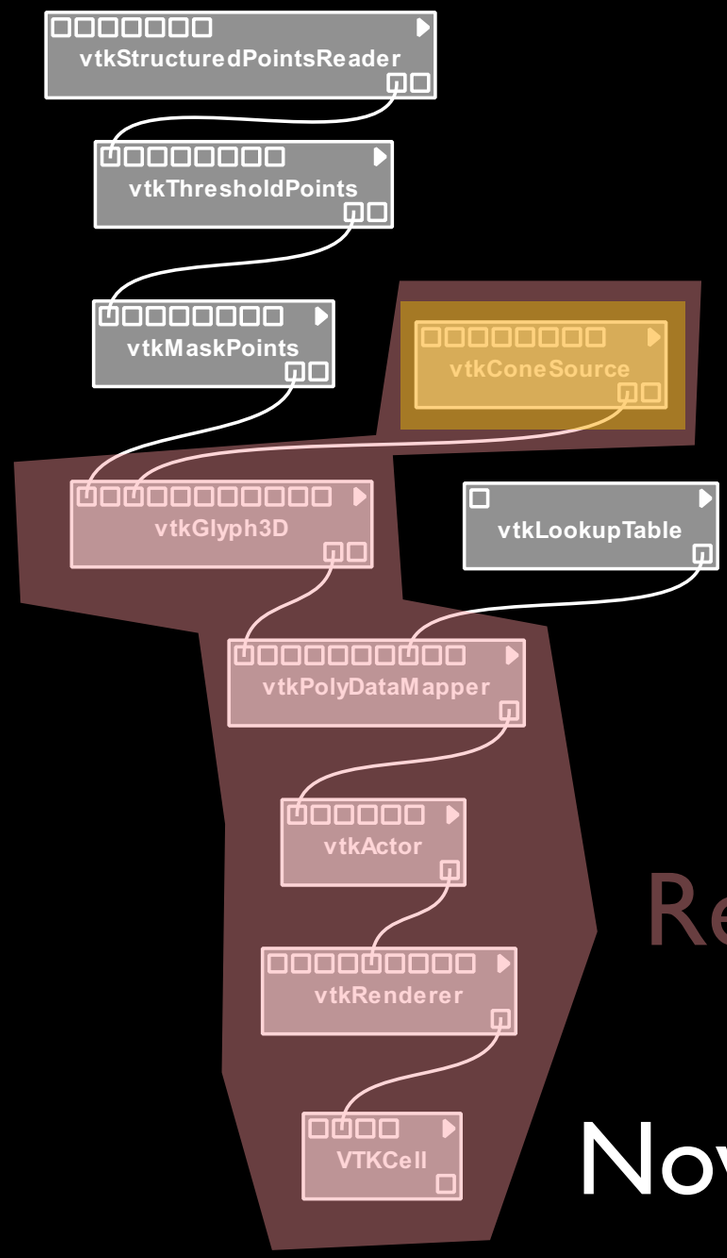
Changed

Now

Caching



Before

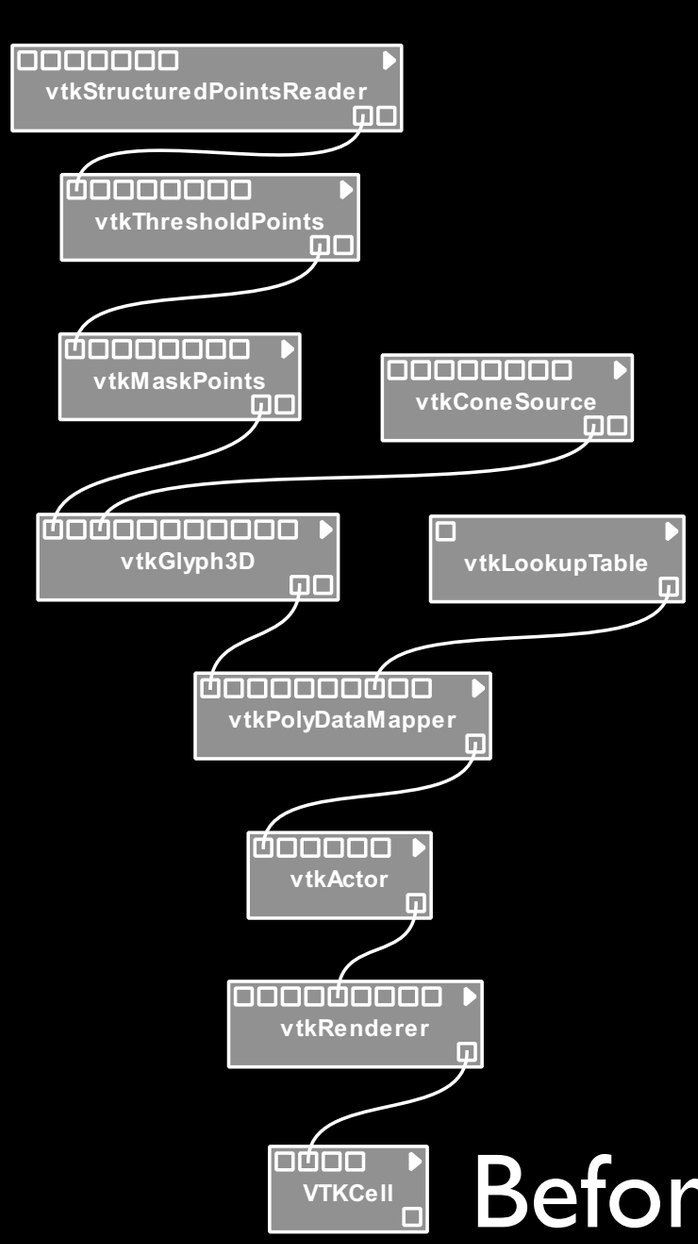


Changed

Re-executed

Now

Caching



Before

Cached

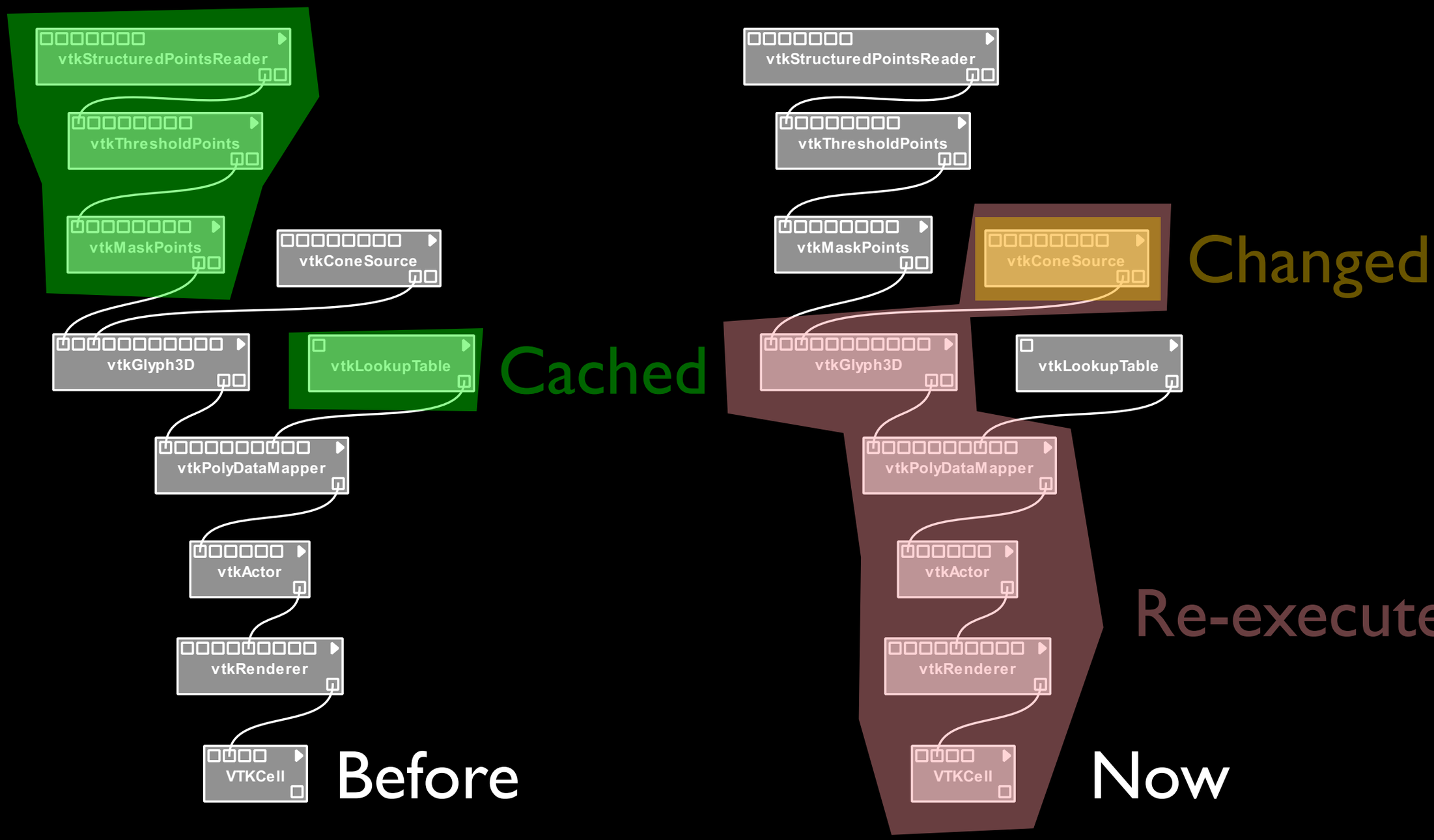
Changed

Cached

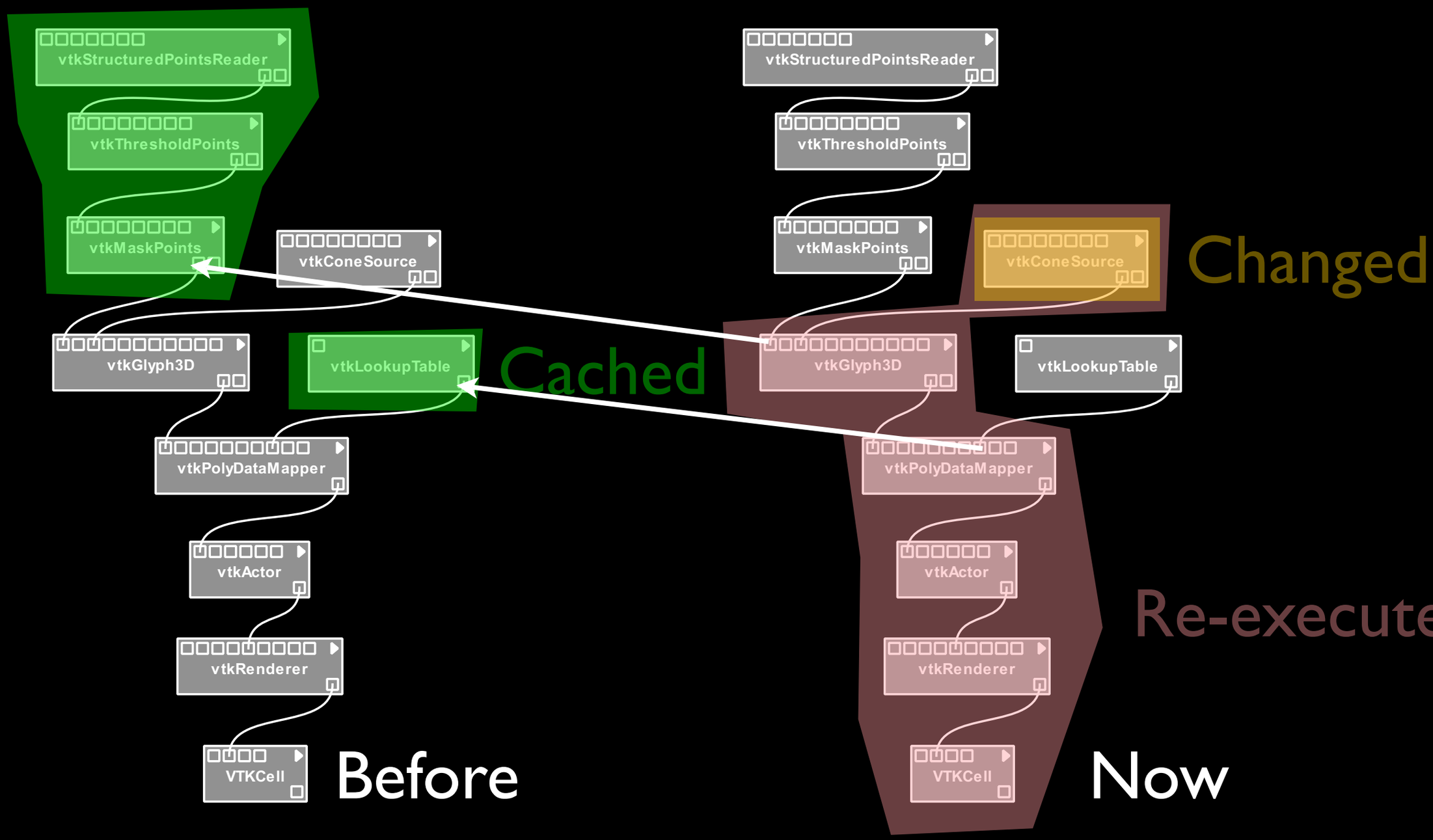
Re-executed

Now

Caching


ed
vtkStructuredPointsReader
vtkThresholdPoints
vtkMaskPoints
vtkConeSource
vtkGlyph3D
vtkLookupTable
Cached
vtkPolyDataMapper
vtkActor
vtkRenderer
VTKCell
Before
vtkStructuredPointsReader
vtkThresholdPoints
vtkMaskPoints
vtkConeSource
Changed
vtkGlyph3D
vtkLookupTable
vtkPolyDataMapper
vtkActor
vtkRenderer
VTKCell
Re-executed
Now

Caching


vtkStructuredPointsReader
vtkThresholdPoints
vtkMaskPoints
vtkConeSource
vtkGlyph3D
vtkLookupTable
vtkPolyDataMapper
vtkActor
vtkRenderer
VTKCell
Cached
Changed
Re-executed
Before
Now

Caching

- This means your module's output should only depend on inputs and parameters
  - Don't mutate input either
  - What if it doesn't? (random numbers, etc.)

```
def is_cacheable(self):
    return not self.getInputFromPort('op') in ['rand', 'nrand']
```

- Or just mix `NotCacheable` in

GUI Interaction

(Qt)

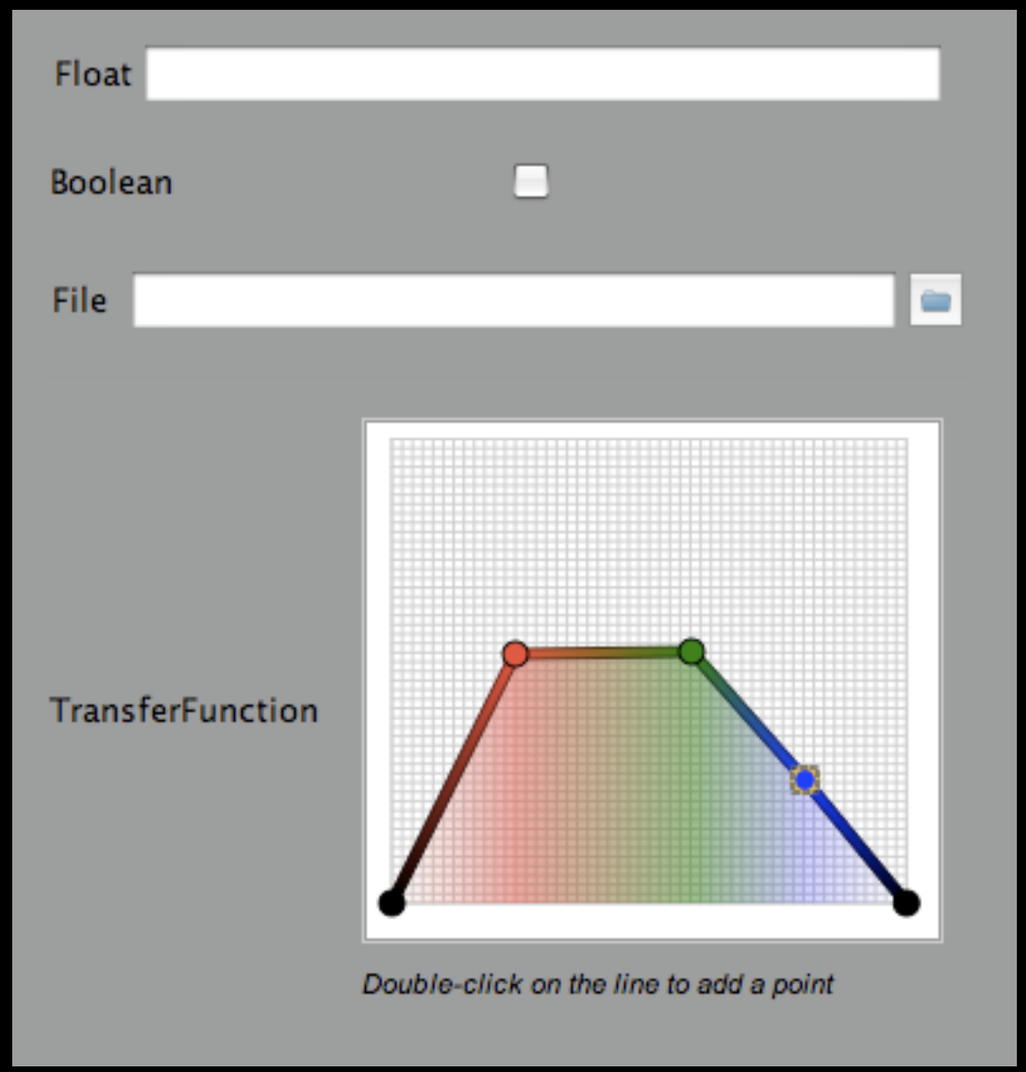
User-defined Constants

- "Constant" is somewhat of a misnomer, since data in VisTrails should all be immutable
- defined mostly for user convenience
  - Ports whose types are all constants are automatically promoted to "methods"

User-defined Constants

- Methods are set directly by users, without ports

- Each constant type has a widget associated with it



User-defined Constants

- Define a Python class that will hold a value
  - values appear in `compute` calls (remember constants were different?)
- `new_constant`
  - *VTK package tf_widget demo*

Spreadsheet Cells

- The simple way to show your own results is to create images and use `ImageViewerCell`
  - or HTML with `RichTextViewer`

- However, you can create your own cell class

Spreadsheet Cells

- subclass from `SpreadsheetCell`, and call `displayAndWait` on `compute`

- *show pylab example*

Odds and Ends

Package Dependencies

- Easy method:

```
def package_dependencies():
    return ['edu.utah.sci.vistrails.vtk']
```

- Powerful method:

```
def package_dependencies():
    import core.packagemanager
    manager = core.packagemanager.get_package_manager()
    if manager.has_package('edu.utah.sci.vistrails.vtk'):
        return ['edu.utah.sci.vistrails.vtk']
    else:
        return []
```

- **conditionally** add modules

Package Dependencies, ctd.

- To conditionally add modules
  - do not use `_modules`, `_input_ports` and `_output_ports`
- inside `initialize`, use `add_module`, `add_input_port`, `add_output_port`

Package Requirements

- `core.requirements` offers some utilities
  - `python_module_exists`, `executable_file_exists`
- you can use it to check for installed software

Package Requirements

- If your software is available through `yum` or `apt`, VisTrails can install it for you
  - it makes distribution that much easier
  - *if you have some other favorite system package manager, let us know*
- *show VTK package example*

Menu Items

- You can provide menu options for your package

```
def menu_items():
    def show_spreadsheet():
        spreadsheetWindow.show()
    lst = []
    lst.append(("Show Spreadsheet", show_spreadsheet))
    return tuple(lst)
```

Big API

- If you have more than a handful of classes, chances are you don't want to generate them by hand
  - Python allows dynamically generated classes
  - we provide `new_module`
  - *show VTK* `new_module` *example*

Question?

But I really want loops and branching!

- Does not play nicely with caching, so we have tried to avoid it
  - Make each module output a parse tree, and make an interpreter module
    - Done by HE physicists in Cornell
  - Change the interpreter
    - Done by researchers in Rio