# Node Attributes

Module 6

Autodesk

# Autodesk Maya Python API Training

***Kristine Middlemiss,*** *Senior Developer Consultant*
*Autodesk Developer Network (ADN)*

Autodesk

# Questions Day 5:

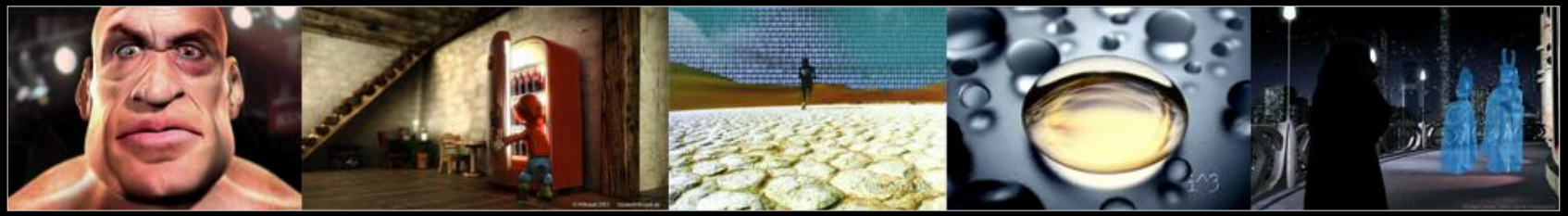
1. Where do the .py files go so they show up in the plugin Manager?
2. Is there a way to query in Maya what MAYA_PLUG_IN_PATH will return?
3. In lay man term Plug is a portal to Connect attribute. Whereas attribute are actual data flowing??
4. In the samples, there is a check in the compute() methods to see if the plugs are input attributes. Should we do this? or only check if the plug corresponds to an output attribute?

# Homework Review

# Agenda

- Exploring MFnAttribute in Depth
- Looking at Static, Dynamic and Extension Attributes
- Setting up Customer User Interfaces for MPxNode

# Types of Attributes







Autodesk®

# API Classes for Attributes

- Base Class: MFnAttribute

- Most Common Used Classes
  - MFnNumericAttribute
  - MFnCompoundAttribute
  - MFnTypedAttribute
  - MFnMatrixAttribute
  - MFnGenericAttribute

# MFnCompoundAttribute

```
nAttr= OpenMaya.MFnNumericAttribute()
transCircleNode.inputTranslateX = nAttr.create( "inputTranslateX", "itX",
OpenMaya.MFnNumericData.kDouble, 0.0 )
nAttr.setStorable(True)

transCircleNode.inputTranslateY = nAttr.create( "inputTranslateY", "itY",
OpenMaya.MFnNumericData.kDouble, 0.0 )
nAttr.setStorable(True)

transCircleNode.inputTranslateZ = nAttr.create( "inputTranslateZ", "itZ",
OpenMaya.MFnNumericData.kDouble, 0.0 )
nAttr.setStorable(True)

comAttr = OpenMaya.MFnCompoundAttribute()
transCircleNode.inputTranslate = comAttr.create("inputTranslate","it")
comAttr.addChild(transCircleNode.inputTranslateX)
comAttr.addChild(transCircleNode.inputTranslateY)
comAttr.addChild(transCircleNode.inputTranslateZ)
comAttr.setStorable(True)
```

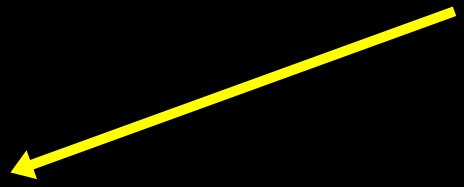
# Attribute Data Types

- Basic
  - Numeric (float, int32,etc.)
  - String
  - Matrix
  - Etc.
- Complex
  - Mesh
  - NurbsSurface
  - Generic (accepts more than one type)
  - Etc.

# MFnTypedAttribute

- Function set for typed attributes, a typed attribute accepts exactly one type of data ( vs. MFnGenericAttribute)

```
MFnTypedAttribute::create ( const MString &  full, const MString &  brief, MFnData::Type type ,
MObject  defaultData, MStatus *  ReturnStatus)
```



- MFnData::Type
  - kNumeric
  - kString
  - kMatrix
  - kIntArry, kDoubleArray, kPointArray…
  - kMesh, kNurbsSurface….
  - Etc.

# MFnTypedAttribute

## Create a string Attribute:

```
typedAttr = OpenMaya.MFnTypedAttribute()
defaultString = "description string for current node"
fnStringData = OpenMaya.MFnStringData()
defaultStringObj = fnStringData.create(defaultString)
descString = typedAttr.create("descString", "dStr", OpenMaya.MFnData.kString, defaultStringObj)
typedAttr.setStorable(1)
```

# Data Creation: MFnData

- MFnData: parent class for all DG data function sets

MFnData::Type

kNumeric

kString

kMatrix

kIntArry, kDoubleArray, kPointArray…

kMesh, kNurbsSurface….

Etc.

MFnNumericData,

MFnStringData,

MFnMatrixData,

MFnIntArrayData, MFnDoubleArrayData MFnPointArrayData,….

MFnMeshData, MFnNurbsSurfaceData.

Etc.

# MFnTypedAttribute

- MFnTypeAttribute with MFnData::kNumeric

```
typedAttr = OpenMaya.MFnTypedAttribute()
fnNumericData = OpenMaya.MFnNumericData();
defaultDataObj = fnNumericData.create(OpenMaya.MFnNumericData.k3Float)
fnNumericData.setData3Float(1.5, 2.5, 3.5)
typedNumeric = typedAttr.create(“typedNumeric", “tNum", OpenMaya.MFnData.kNumeric, defaultDataObj )
```

MEL: addAttr -longName numAttr –dataType float3;

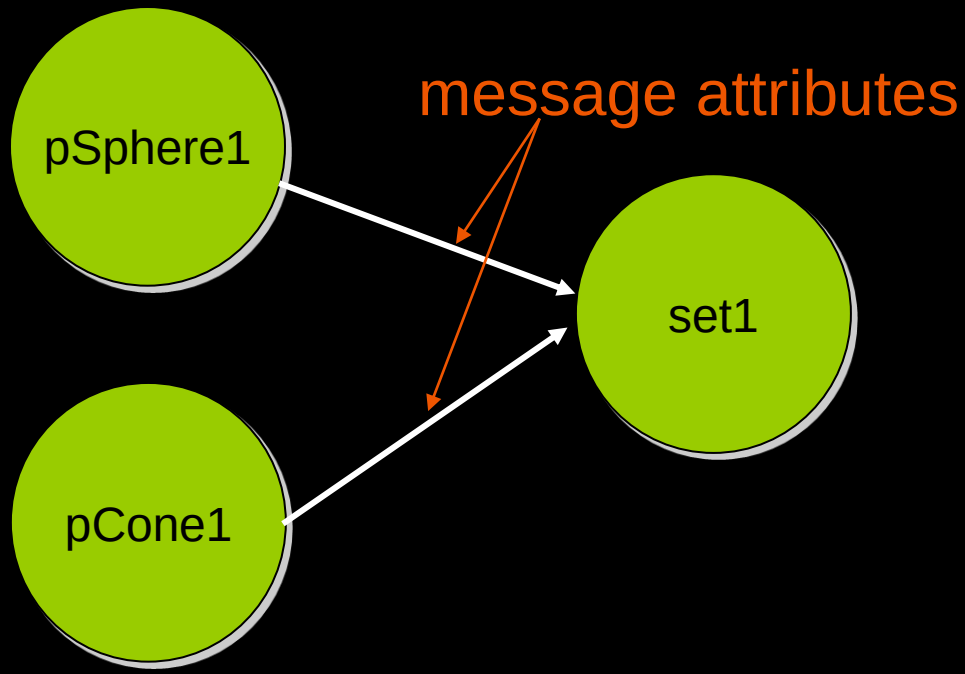
- MFnNumericAttribute

```
nAttr = OpenMaya.MFnNumericAttribute();
input = nAttr.create( "input", "in", OpenMaya.MFnNumericData.kFloat, 0.0 );
```

MEL: addAttr –longName singleNum –attributeType float;

# Message Attribute

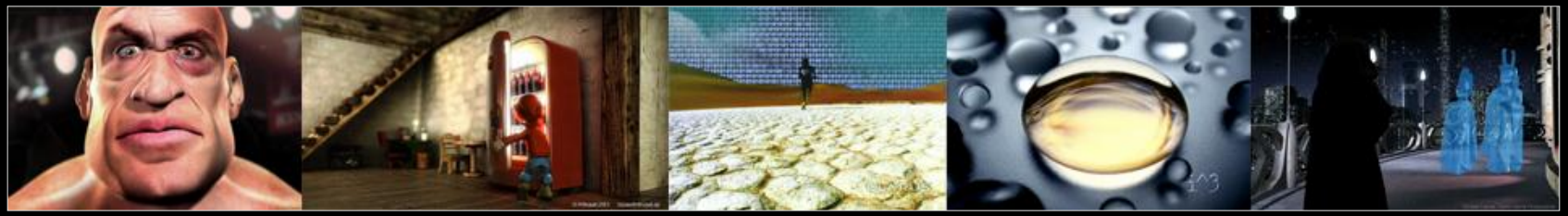
- Some connections indicate relationships rather than propagate data flow (eg. sets)



# Message Attribute

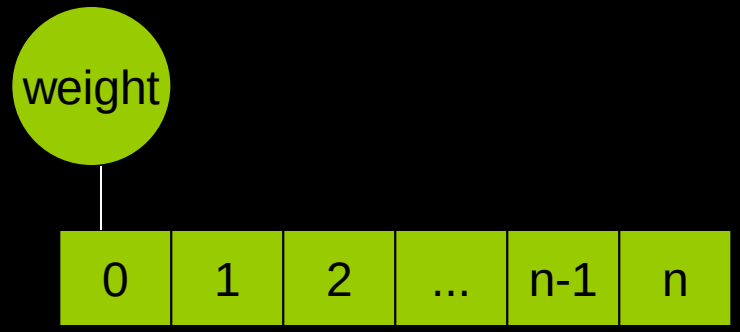
- Some connections indicate relationships rather than propagate data flow (eg. sets)
  - Indicate membership in a grouping
  - No data is actually stored

- **Array Attribute (Multi)**



# Array Attribute (Multi)

- An array of simple data
- The data type of each element is defined to be the type specified by the attribute
- Each element plug can contains its own value
- Each element plug can have its own connection




getAttr blendShape1.weight[0]

# Array Attribute

- Array can be sparse
- Logical index v.s. Physical index
  - Logical indexes are sparse and used by MEL

    MPlug:: elementByLogicalIndex()

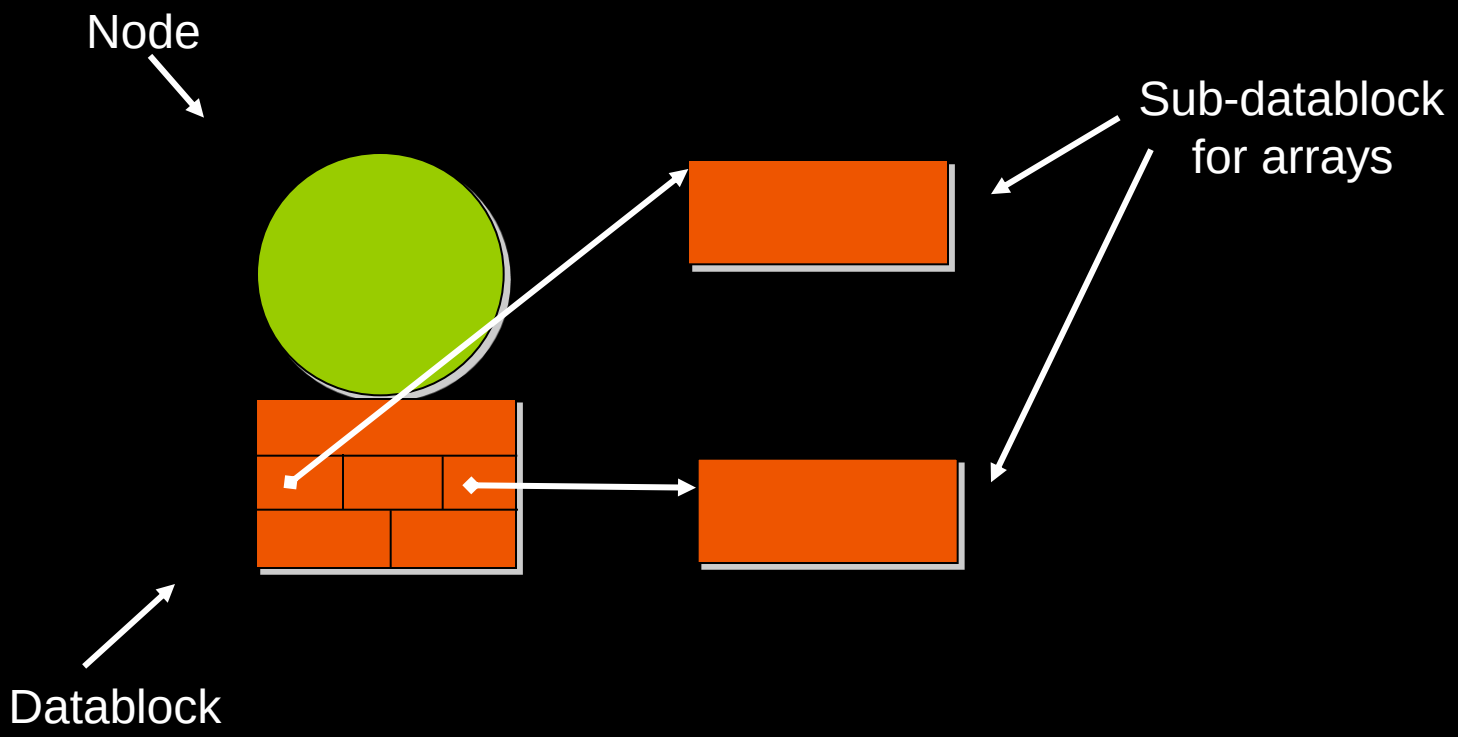
    When try to retrieve element plug value, element plug will be created if does not exist already

  - Physical indexes are not sparse, it is guaranteed that the physical indexes will range from 0 to numElements() – 1

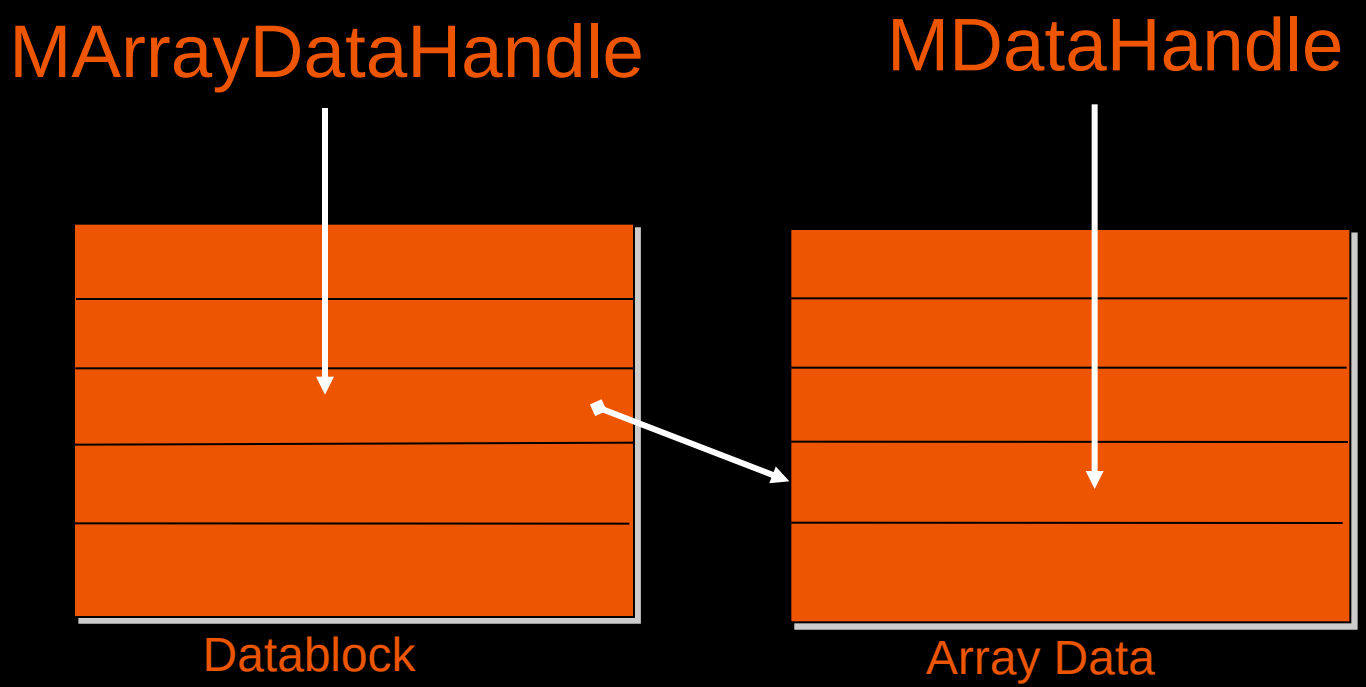
    MPlug:: elementByPhysicalIndex()

# Multi Attribute & DataBlock

- Arrays are stored separately in sub-datablocks, and accessed through array data handles



# MultiAttribute & DataBlock



# Initialization of Multi Attribute

```
def nodeInitializer():
        nAttr = OpenMaya.MFnNumericAttribute()
        myNode. arrayInputAttr = nAttr.create("arrayInput", "ai", OpenMaya.MFnNumericData.kFloat, 1.0)
        nAttr.setStorable(1)
        nAttr.setArray(1)
        myNode.addAttribute(myNode.arrayInputAttr)

        myNode.arrayOutputAttr = nAttr.create("arrayOutput", "ao", OpenMaya.MFnNumericData.kFloat, 1.0)
        nAttr.setStorable(1)
        nAttr.setArray(1)
        nAttr.setWritable(0)
        myNode.addAttribute(myNode.arrayOutputAttr)

        return OpenMaya.MStatus.kSuccess
```

# Attribute Relationship

```
def setDependentsDirty(self, inPlug, affectedPlugs ):
        if inPlug == myNode.arrayInputAttr :
            outArrayPlug = OpenMaya.MPlug (self.thisMObject(), myNode.arrayOutputAttr)
        if inPlug.isElement() == true:
            logicalIndex = inPlug.LogicalIndex()
            outputElem = OpenMaya.Mplug(outArrayPlug.elementByLogicalIndex(logicalIndex))
            affectedPlugs.append(outputElem)
          else
           affectedPlugs.append(outArrayPlug);

        return OpenMaya.Mstatus.kSuccess
```

# Compute Array Attribute

```
def compute(self, plug, dataBlock):
    if plug == myNode.arrayOutputAttr :
        if  plug.isElement() == True:
            indexToCompute = plug.logicalIndex()
            inputArrayHandle = dataBlock.inputArrayValue(myNode.arrayInputAttr)
            inputArrayHandle.jumpToElement(indexToCompute)

            inputElementHandle = inputArrayHandle.inputValue()
            inputElementData = inputElementHandle.asFloat()
```

# Compute Array Attribute

- outputArrayHandle = dataBlock.outputArrayValue(arrayOutputAttr)
- outputArrayHandle.jumpToElement(indexToCompute)
- outputElementHandle = outputArrayHandle.outputValue()
- 
- outputElementHandle.setFloat( inputElementData )
- 
- return OpenMaya.Mstatus.kSuccess
- 
- 
- return OpenMaya.Mstatus.kUnknownParameter

# Array Attributes vs. Array Data

- Array Attributes
  - using MFnAttribute::setArray(true)
    - the elements of the array are accessible through MEL by using:

      ```
      getAttr node.attribute[element];
      ```

      (also available in the attribute editor)
    - not very effective for large arrays in terms of memory usage and speed
    - no array elements defined at creation time
    - allows access to individual element

# Array Attributes vs. Array Data

- Array Data
  - using MFnTypedAttribute to create a kDoubleArray
    - the array elements are not accessible through MEL
    - effective for large arrays
    - can be constructed with a default value
    - Easier to handle as data “chunk”

# Ways to add Attributes

1. Static Attributes
   - Built into the Maya code or are members of plug-in nodes, and cannot be modified or removed.
2. Dynamic Attributes
   - Allow you to add or delete attributes on a particular node. Dynamic attributes are different from static attributes in that they are only married to a specific instance of a node type.
3. Extension Attributes
   - Allow you to add or delete attributes, at run time, on all nodes of a given type.
   - For example, you can add an attribute to a transform type instead of an instance of a transform.

# Dynamic Attributes

How to create a dynamic attribute in Maya:

1. Attribute Editor

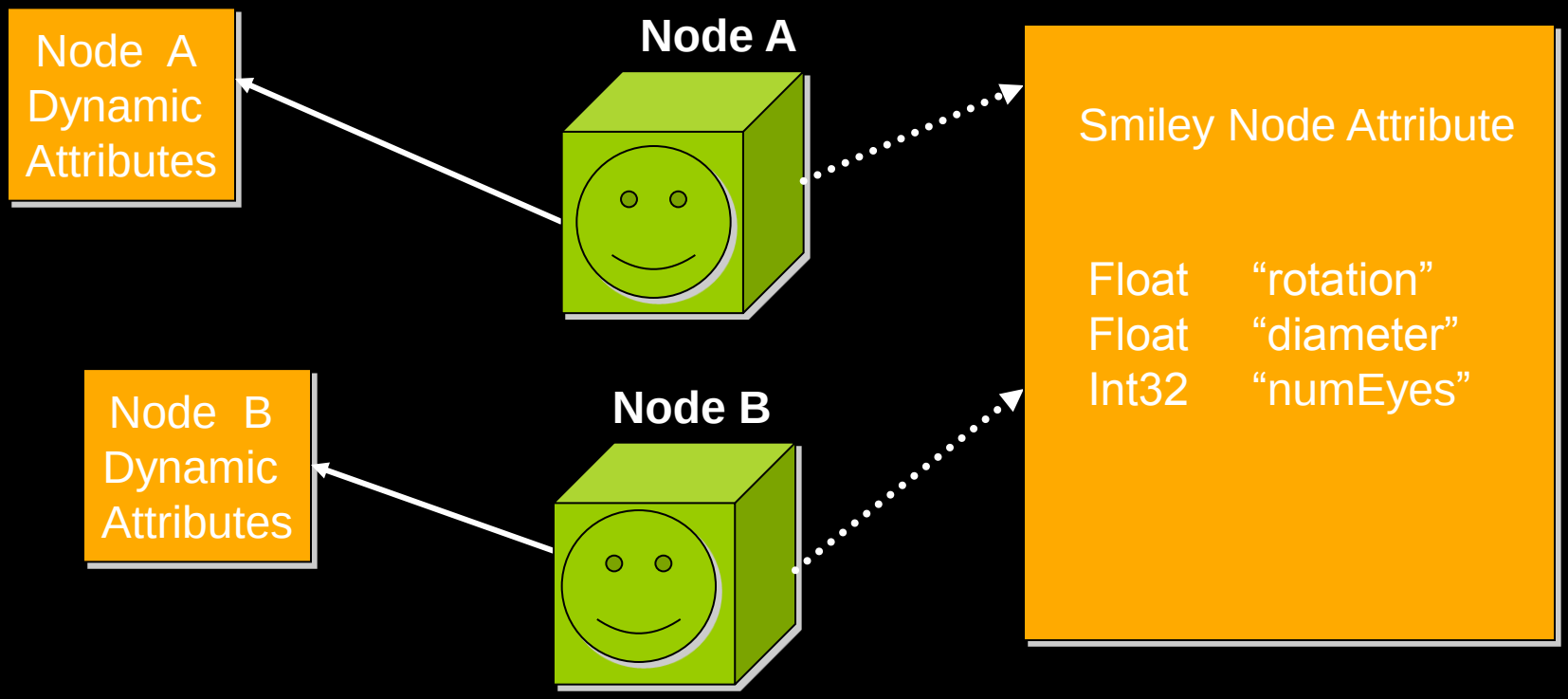
2. MEL Command: addAttr

   addAttr -longName oneAttr -at double;

3. Any code outside of MPxNode::initialize() to create an attribute

# Dynamic Attributes

- Dynamic attributes belong to the node



# Dynamic Attribute

- MPxNode::postConstructor()

```
def postConstructor(self):

  nAttr = OpenMaya.MFnNumericAttribute()
  dynAttr = nAttr.create( "dynAttr", "da", OpenMaya.MFnNumericData.kFloat, 0.0 )
  nAttr.setStorable(1)
  nAttr.setKeyable(1)

  thisNode = self.thisMObject()
  depNode = OpenMaya.MFnDependencyNode(thisNode)
  depNode.addAttribute(dynAttr)
```

# Attribute Relationship

MPxNode::setDependentsDirty()

- Handle dynamic attribute as well as non-dynamic attribute
- More flexible relationship
- Do not perform any DG computation

# MPxNode::setDependentsDirty()

```
dynNode.aOutput  # Non-dynamic attribute “aOutput"

def setDependentsDirty(self, dirtyPlug, plugArray):
        if dirtyPlug == self.dynAttr:
                thisNode = self.thisMObject()
                try:
                        plug = OpenMaya.MPlug(thisNode, dynNode.aOutput)
                        plugArray.append(plug)
                except:
                        pass
        return OpenMayaMPx.MPxNode.setDependentsDirty(self, dirtyPlug, plugArray)
```

# Extension Attributes

How to create a extension attribute in Maya:

1. Attribute Editor

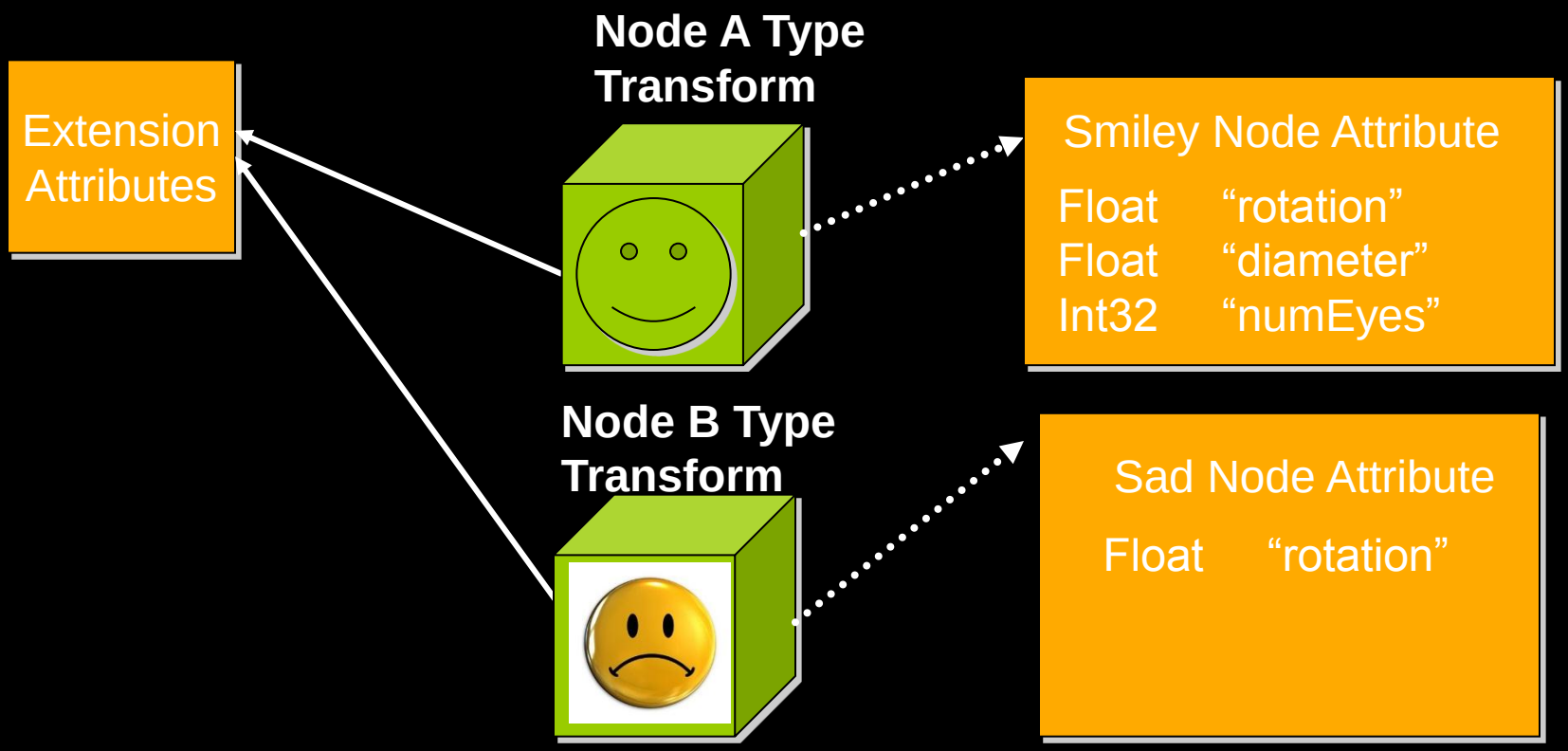
2. MEL Command: addExtension

   addExtension -nt "mesh" -shortName ms -longName mass -defaultValue 1.0 -minValue 0.001 -maxValue 10000;

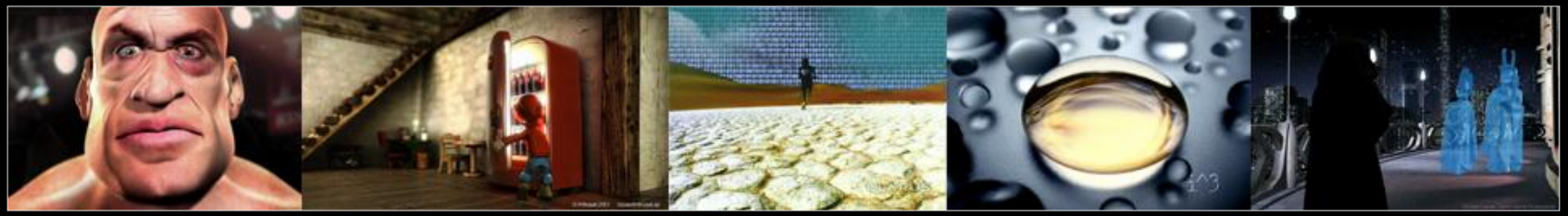
3. Using MDGModifier::addExtensionAttribute

# Extension Attributes

- Extension attributes belong to the Type of Node



# Creating Custom Attribute Editor UI







Autodesk®

# MPxNode -- UI

- Once you've created a new node type, you can create your node via:

**MEL: `createNode myNode`**

**Python:**

```
import maya.cmds as cmds
cmds.createNode("myNode")
```

- More typically, create a command to setup the node (connections, attribute values) and insert it into the scene.

# MPxNode -- UI

- MPxNode in Attribute Editor and Channel Box
  - Attributes in Attribute editor
    - AENodeNameTemplate.mel defines how the attributes from the plugin MPxNode, with the name NodeName, are to display in the attribute editor.

      Node Name: transCircle

      AE template name: AEtransCircleTemplate.mel
  - Attributes in Channel Box
    - The channel box only shows certain simple data types and they need to be either setKeyable(true) or setChannelBox(true).

# Attribute Editor

- In AENodeNameTemplate.mel:
  - The global procedure name should be the same as file name

    global proc AEtransCircleTemplate( string $nodeName ) { …. }

    - editorTemplate
      - editorTemplate -addControl “attributeName”;
      - editorTemplate -suppress “attributeName”;
      - editorTemplate -beginLayout;
      - editorTemplate -endLayout;

- editorTemplate -callCustom “proc_for_new_created_node“ “proc_for_replace_node” “attributeName";

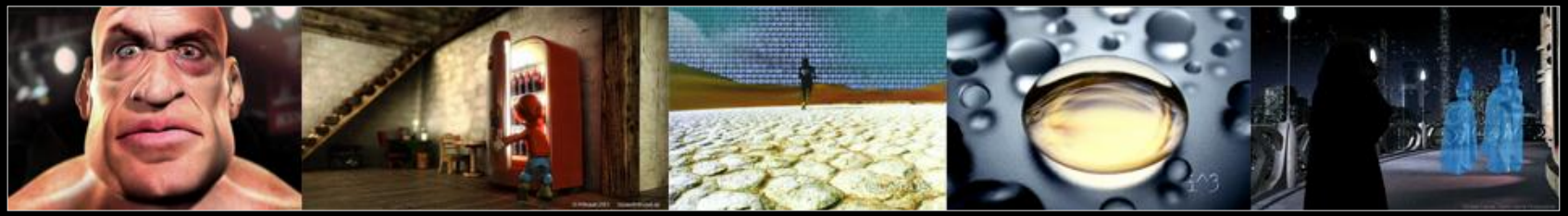
# Attribute Editor

```
editorTemplate -callCustom "transCircleScaleNew“ "transCircleScaleReplace“ "scale";


global proc transCircleScaleNew( string $attrName )
{
        // build the "quick set" control for the scale attribute
        radioButtonGrp  -label "Quick Scale“  -numberOfRadioButtons 3
                -label1 "Five“  -data1 5
                -label2 "Ten“   -data2 10
                -label3 "Fifteen“  -data3 15
                scaleGrp;
        connectControl scaleGrp $attrName;
}


global proc transCircleScaleReplace( string $attrName )
{
   // Install the connection between the radioButtonGrp and the actual scale attribute
   connectControl scaleGrp $attrName;
}
```

# Workshop Session







Autodesk®

# Example: simpleNode - with Typed Attr

In this exercise, we add a special type of attribute: string attribute onto simpleNode from the previous workshop.




Autodesk®

# Autodesk