

Game Developers Conference®

February 28 - March 4, 2011
Moscone Center, San Francisco
www.GDCConf.com

Introduction to Maya Dependency Graph Programming

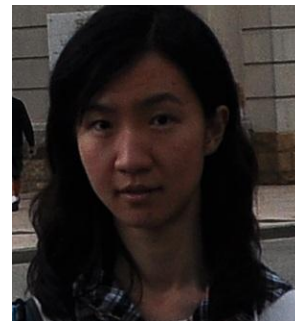
Naiqi Weng
Developer Consultant,
Autodesk Developer Network (ADN)



Biography

Naiqi Weng -- Autodesk Developer Network

- Education
 - Bachelor of Computer Science
 - Master of Computer Science



- Working experience



Autodesk®

Naiqi.weng@autodesk.com

- Supporting software: Maya API, MotionBuilder SDK and 3dsMax SDK

Dependency Graph



Image courtesy of Johan Vikström, Shilo, The Spine, National Film Board of Canada Production ©, Ool Digital, Mikros Image

Agenda

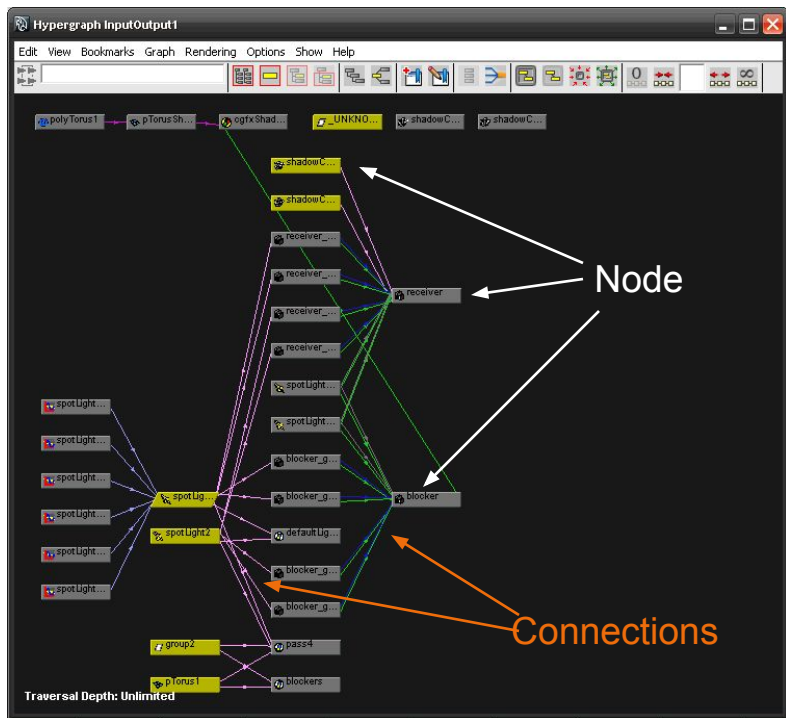
- 🐉 What is Dependency Graph (DG)?
- 🐉 Concepts/Elements of DG Nodes
- 🐉 Custom plug-in DG node with Maya API
- 🐉 DG's push-pull mechanism
- 🐉 Correct coding with DG



Image courtesy of Johan Vikström, Shilo, The Spine, National Film Board of Canada Production ©, Ool Digital, Mikros Image

Maya Hypergraph

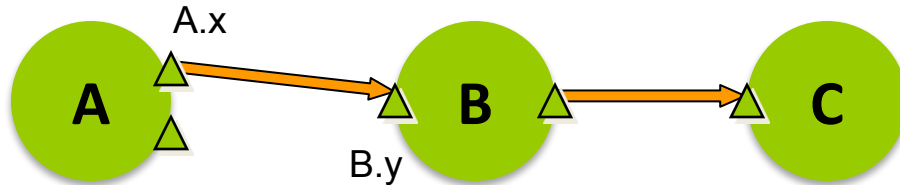
- Menu: “Window”– “Hypergraph:Connections”



- Nodes
- Connections
- Attributes

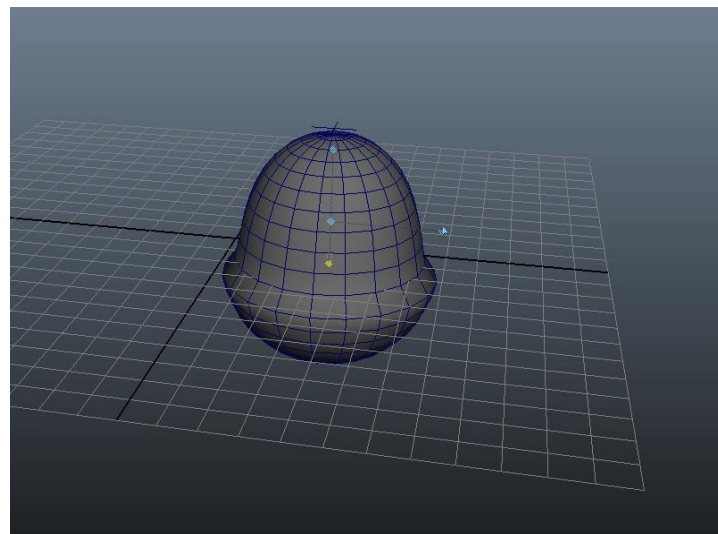
What is the Dependency Graph?

- Control system of Maya
- Patented technology:
 - US Patent #5,808,625
 - US Patent #5,929,864
- A collection of nodes that transmit data through connected attributes

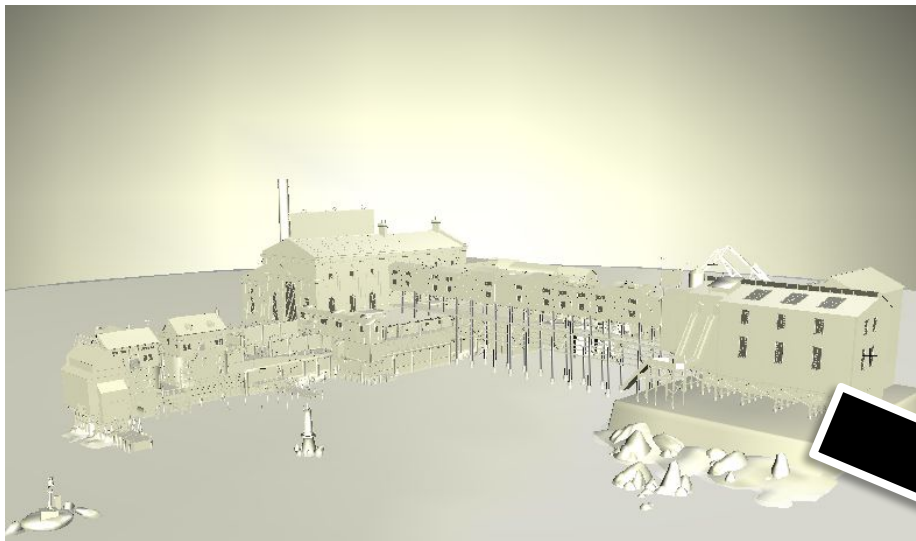


Dependency Graph

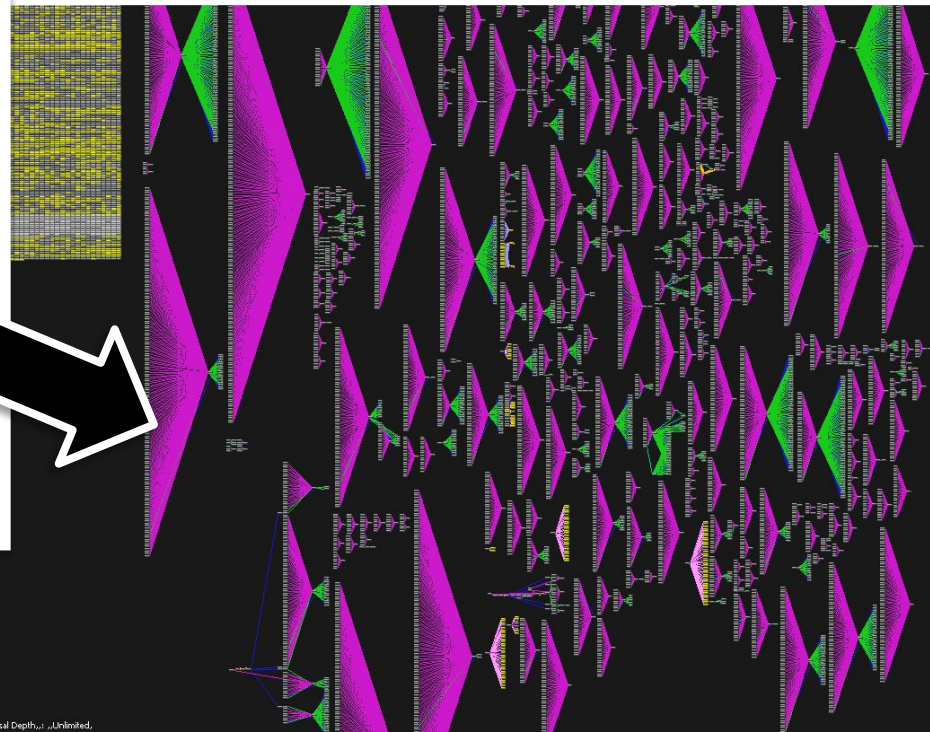
- Everything in Maya 3D space maps to DG Nodes
- Anything you do in the UI will affect DG



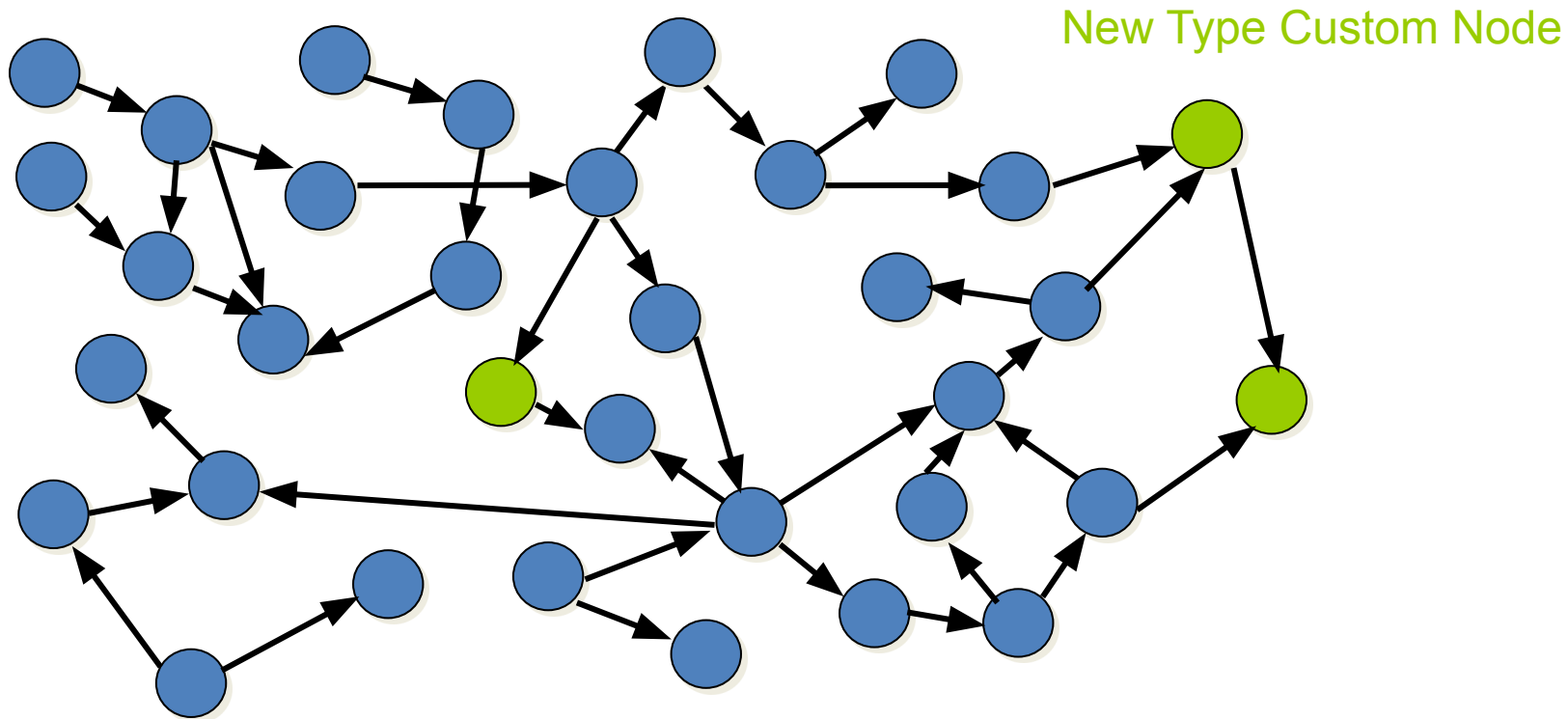
Dependency Graph



© 2009 WET, Artificial Mind and Movement (now Behaviour Interactive)
© 2009 Bethesda Softworks. Image courtesy of Behaviour Interactive



Dependency Graph

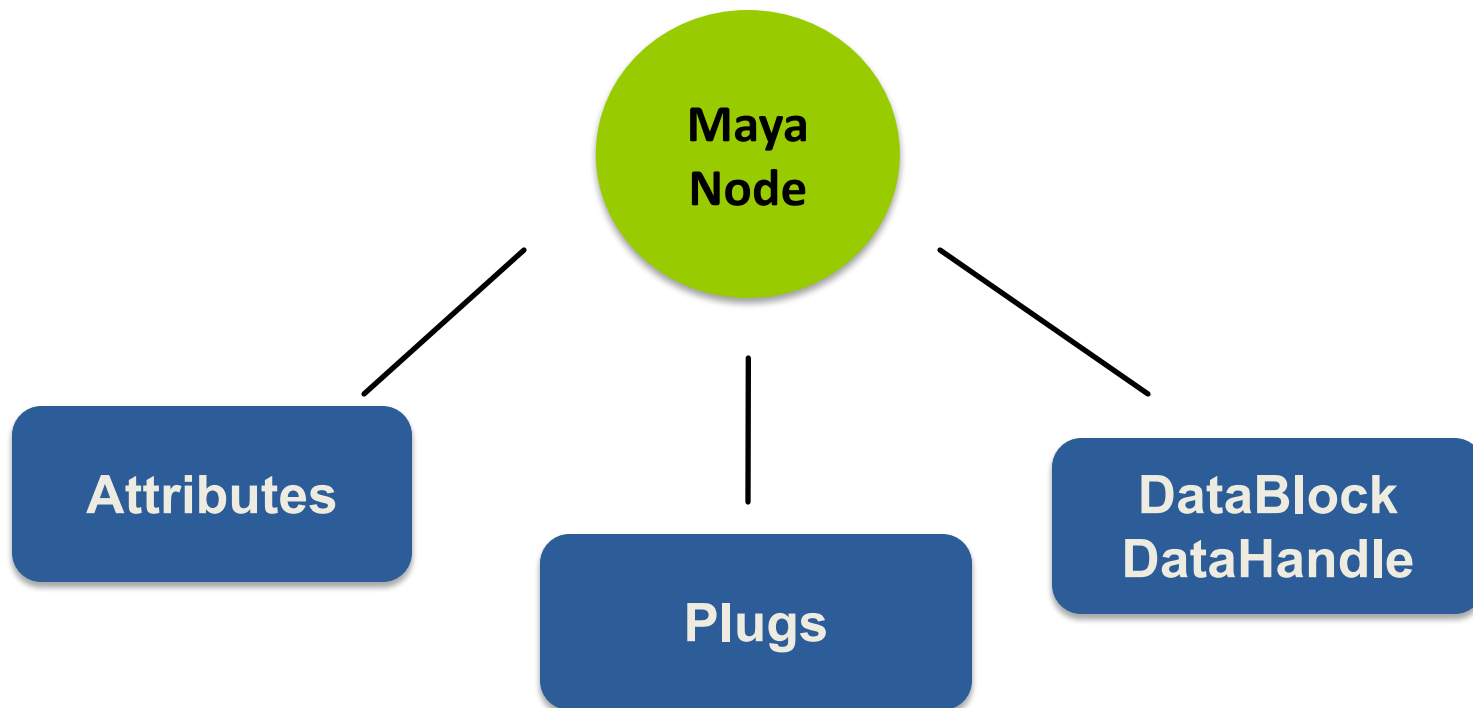


What does a node do?



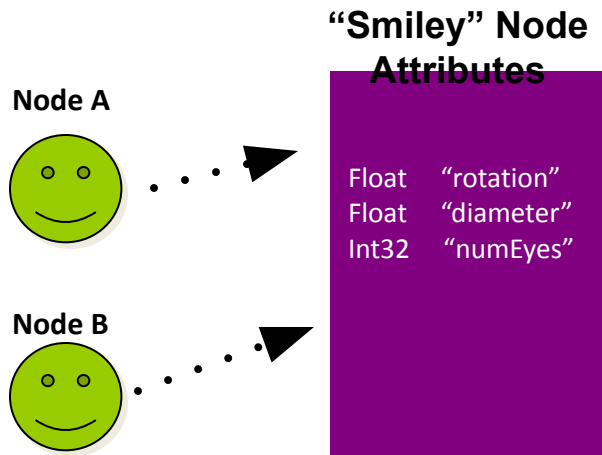
- Know its own attributes
- Store data efficiently in “datablocks”
- Accept input data, compute, generate output data
- Connect with other nodes through connections

Different elements of a Node



Attributes

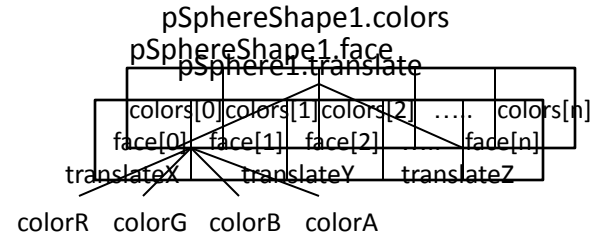
- Describe data that belongs to nodes of a given type



- Attributes are shared among nodes of the same type and all derived node types

Attributes

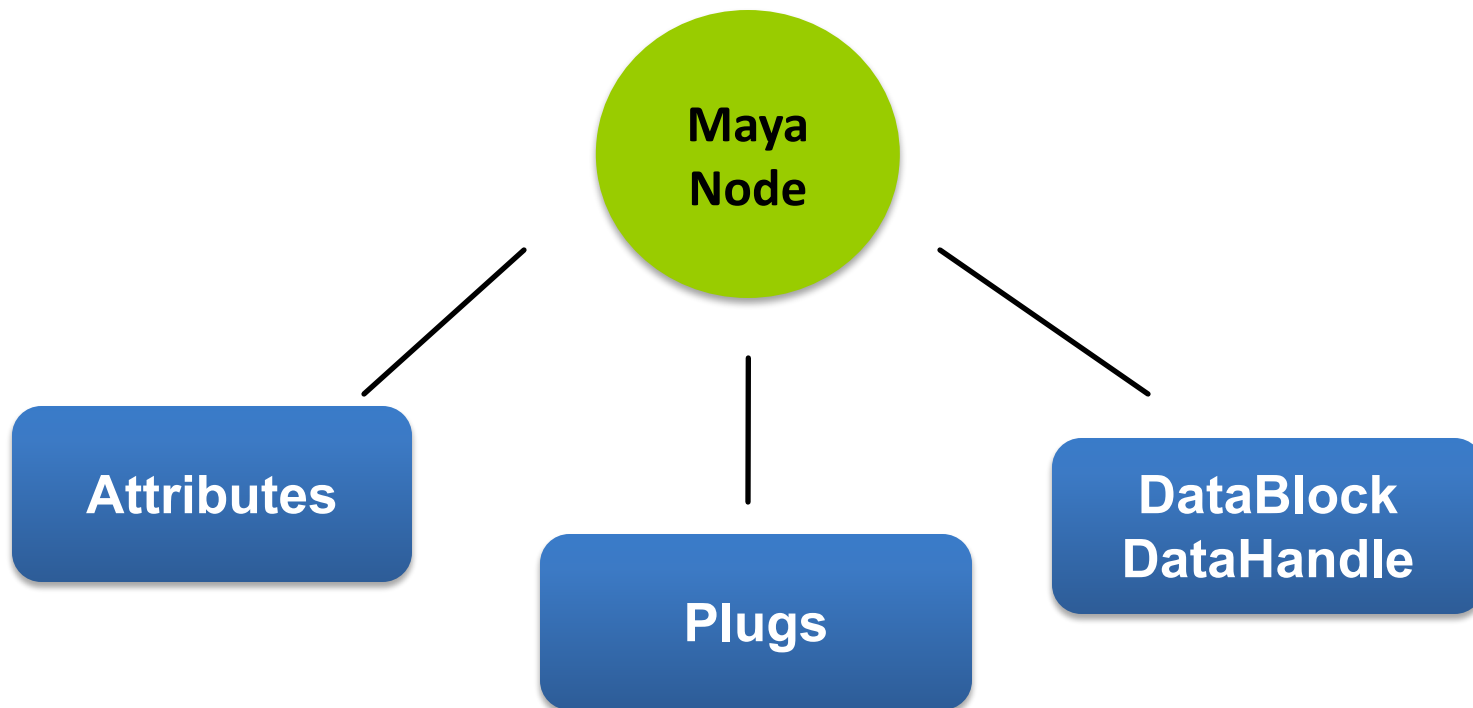
- Define the interface of the node type including
 - Names
 - Basic: numeric (float, int32 etc...), string, matrix, etc..
 - Complex: mesh, nurbsSurface, generic, etc...
 - Custom data
 - Structure
 - Simple, compound, array, compound array
 - Properties
 - readable, writable, storable, keyable, etc...



API Classes for Attributes

- Base Class: MFnAttribute
 - Takes care of all the common aspect of an attribute on node
 - readable/writable, connectable, storable, keyable
- Most Common Used Classes
 - MFnNumericAttribute
 - MFnCompoundAttribute
 - MFnTypedAttribute
 - MFnMatrixAttribute
 - MFnGenericAttribute

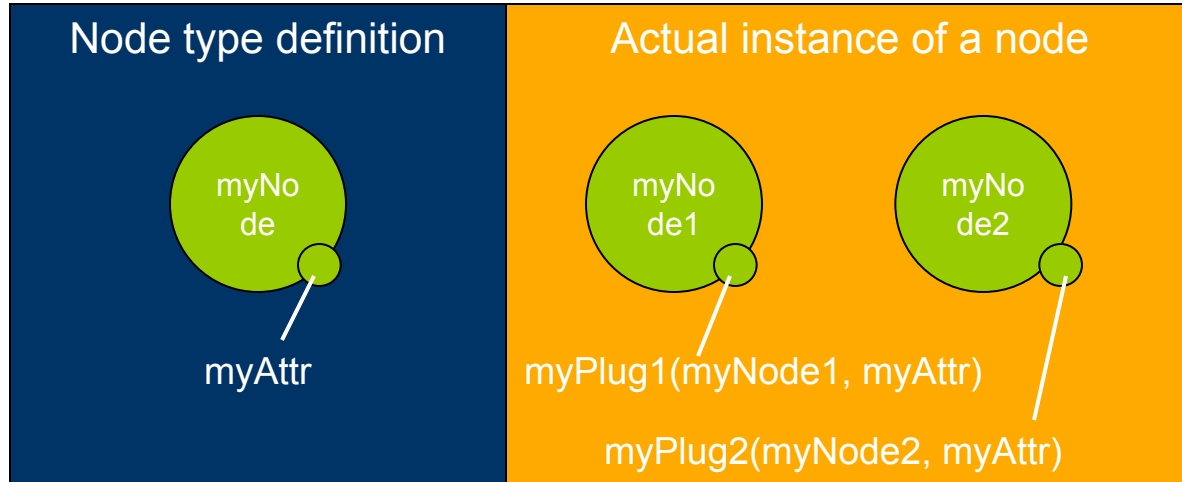
Different elements of a Node



Plugs

- Pointer to an attribute on a specific node (ie. a specific instance of an attribute)

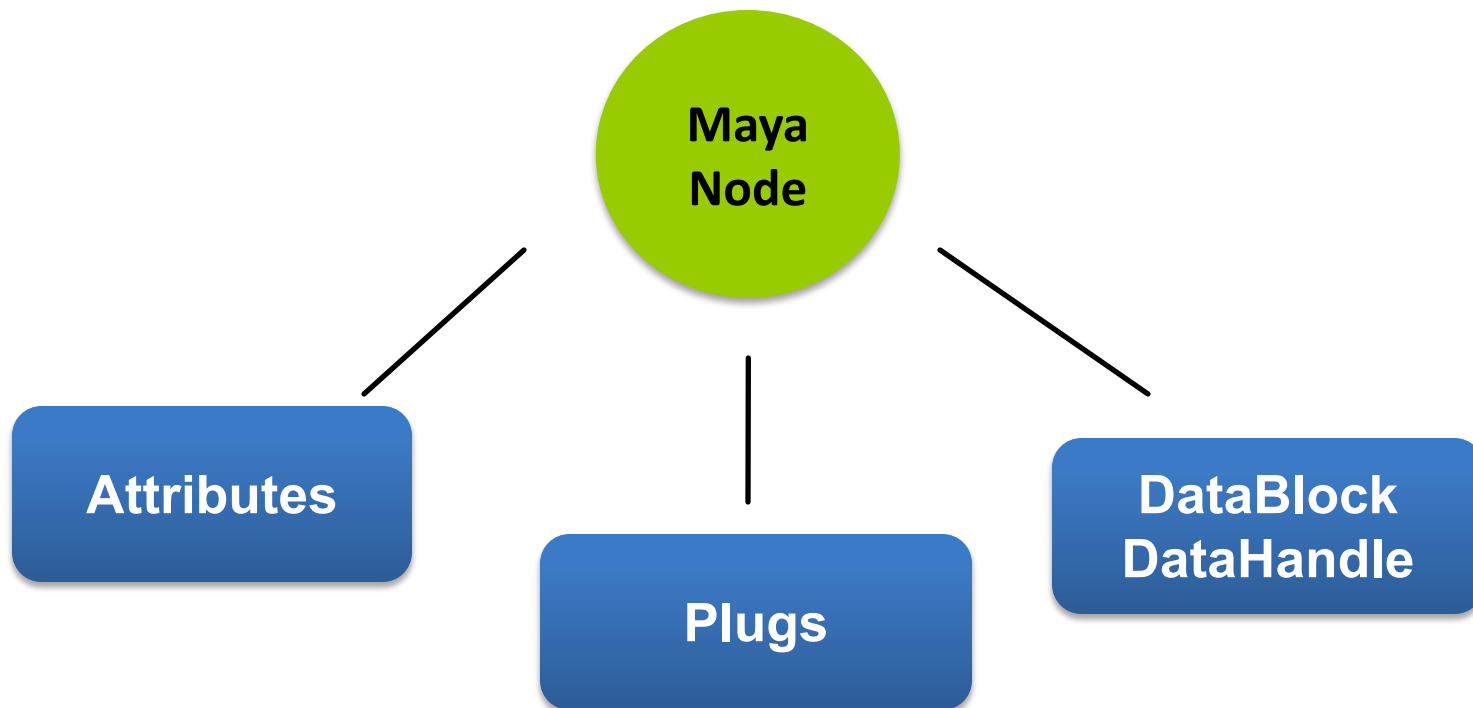
```
setAttr myNode1.myAttr 5;  
setAttr myNode2.myAttr 10;
```



Plugs

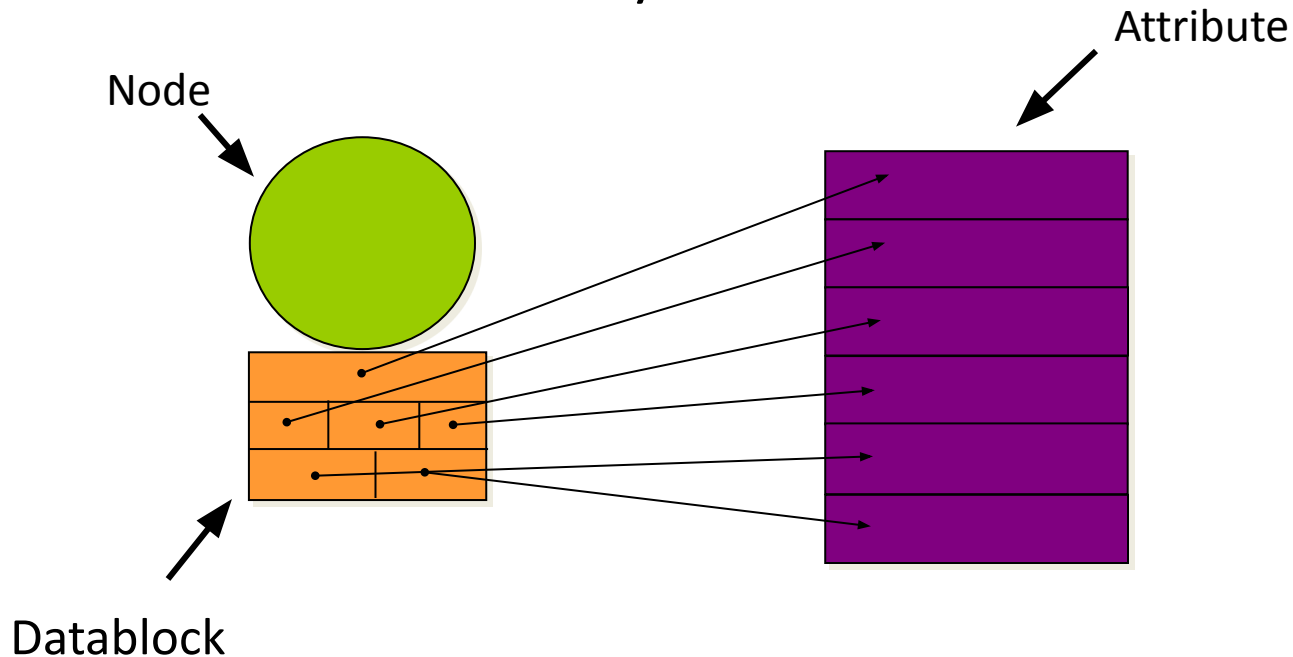
- Plugs can be used to:
 - query / set a value
 - create / remove /query a connection
- Does not store attribute data
- API class: MPlug
 - *MPlug::MPlug (const MObject &node, const MObject &attribute)*
 - *MObject MPlug::attribute (MStatus * ReturnStatus)*
 - *MObject MPlug::node (MStatus * ReturnStatus)*
 - *MStatus MPlug::getValue (double & val, MDGContext & ctx)*
 - *MStatus MPlug::setValue (double val)*

Different elements of a Node



Datablock

- Node stores data for every attribute



Datablocks

- Datablock is the actual storage for the input and output data of a node
- For every non-array attribute, datablock stores:
 - Data
 - Dirty/clean status
- Data handles are lightweight pointers into the data in the datablock

API Classes for datablock

- MDataBlock

- Only valid during compute()
- Pointers to data block should not be retained after compute()

MStatus MPxNode::compute(const MPlug& plug, MDataBlock& dataBlock)

*MDataHandle MDataBlock::inputValue(const MObject& attribute, MStatus * ReturnStatus)*

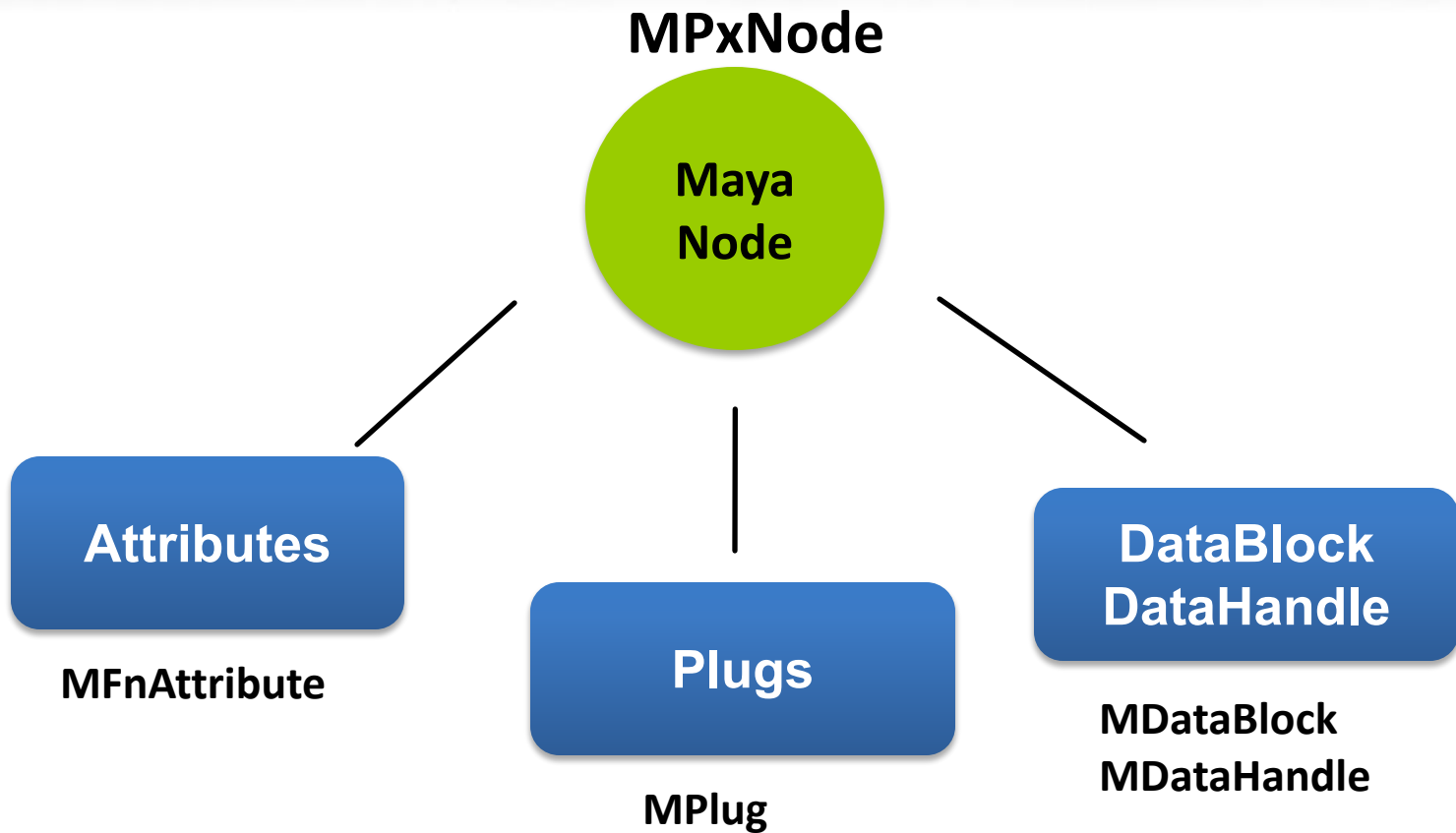
*MDataHandle MDataBlock::outputValue (const MObject & attribute, MStatus * ReturnStatus)*

- MDataHandle

- a smart pointer for information contained in data block

double & MDataHandle::asDouble ()

void MDataHandle::set(double val)



Custom Node Plug-in Implementation



Image courtesy of Johan Vikström, Shilo, The Spine, National Film Board of Canada Production ©, Ool Digital, Mikros Image

Custom DG Node in Maya

- Entirely new operations
 - MPxNode: base class for custom node implementation
- Extend existing Maya nodes
 - MPxDeformerNode
 - MPxFieldNode
 - MPxEmitterNode
 - MPxSpringNode
 - MPxIkSolverNode
 - MPxHwShaderNode

Custom Node Code Skeleton

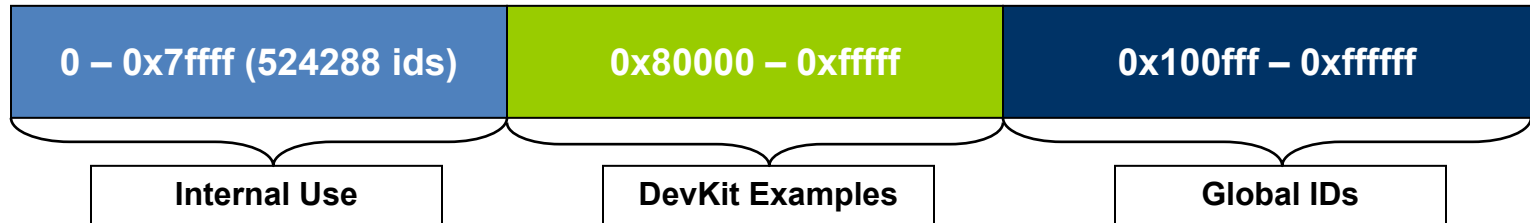
```
class myNode : public MPxNode
{
    public:
        myNode();
        virtual ~myNode();
        static void* creator();
        static MStatus initialize();
        virtual MStatus compute( const MPlug& plug, MDataBlock& data );


    private:
        static MTypeId id;
        static MObject myInputAttr;           //input attribute
        static MObject myOutputAttr;         //output attribute
        static MObject myOutputAttrTwo;      //second output attribute
};
```

Custom Node Registration

- Every node type requires a unique identifier

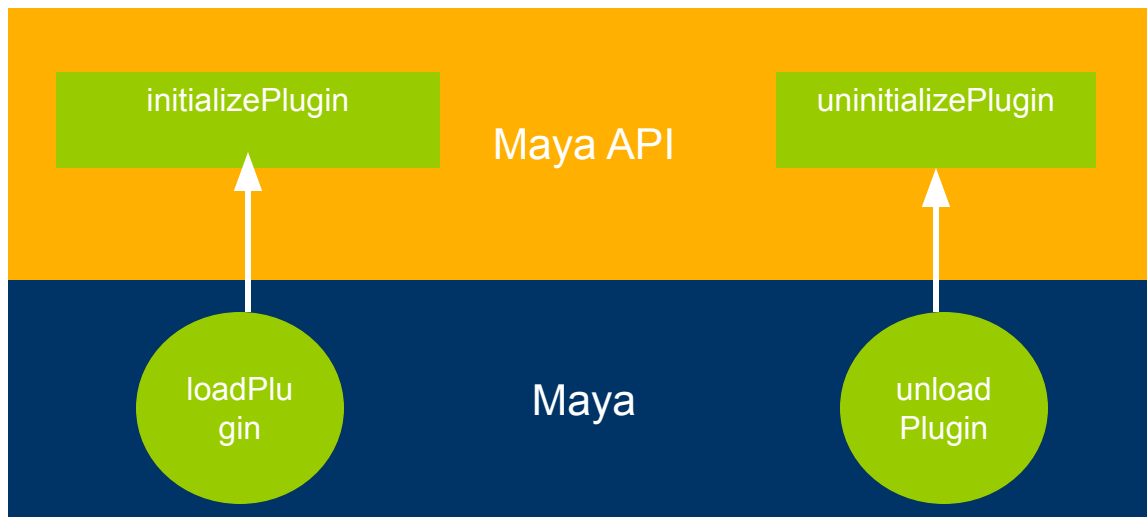
```
MTypeId myNode::id( 0x80000 );
```



- For plug-ins that you intend to share between sites 
 - Will require a globally unique ID issued to you by Autodesk.
 - IDs are allocated in blocks of 64/128/256/512.
 - Contact ADN M&E for unique global IDs.

Custom Node Registration

- initializePlugin() and uninitializePlugin() are entry point and exit point of custom plug-in node



Custom Node Registration

- To register your node with Maya:

```
MStatus initializePlugin(MObject obj)
{
    MFnPlugin plugin(obj, "Autodesk", "1.0", "any");

    MStatus status = plugin.registerNode("myNode", myNode::id, myNode::creator, myNode::initialize),

    return status;
}
```

- To deregister your node

```
MStatus uninitializePlugin(MObject obj)
{
    MFnPlugin plugin(obj);

    MStatus status = plugin.deregisterNode(myNode::id);

    return status;
}
```


Custom Node Code Skeleton

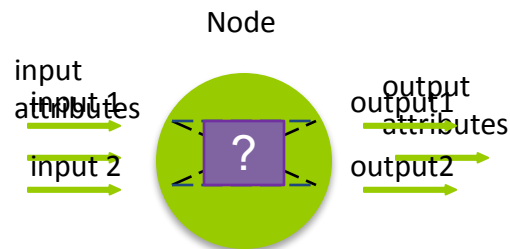
- `MPxNode::creator()`
 - The creator method is called to return a new instance of the node

```
void* myNode::creator()  
{  
    return new myNode;  
}
```

- In the Maya UI
 - MEL: `createNode myNode;`

Custom Node Code Skeleton

- `MPxNode::initialize()`
 - Override this method to define the attribute interface for your node.
 - create attributes
 - set the attribute's properties
 - add the attribute to the node
 - Inherit attributes if necessary
 - define attribute relationships



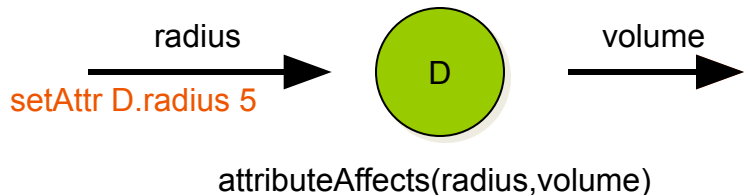
Attribute Dependency

- Attributes can affect other attributes
- MEL command: affects

sphere -n sphere;

affects tx sphere;

- MPxNode::attributeAffects()
 - Once created on a node, an “attributeAffects” relationship can be setup to denote a dependency



Custom Node Code Skeleton

```
MStatus myNode::initialize()
{
    MFnNumericAttribute nAttr;

    myInputAttr = nAttr.create("myInput", "mi", MFnNumericData::kFloat, 1.0);
    nAttr.setStorable(true);

    myOutputAttr = nAttr.create("myOutput", "mo", MFnNumericData::kFloat, 1.0);
    nAttr.setStorable(true);

    myOutputAttrTwo = nAttr.create("myOutputTwo", "motwo", MFnNumericData::kFloat, 1.0);
    nAttr.setStorable(true);

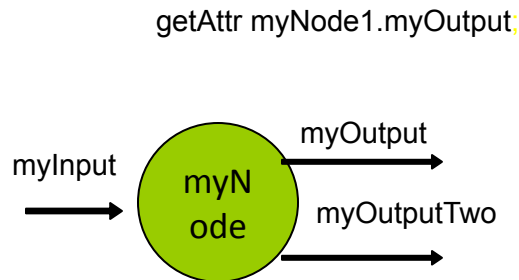
    addAttribute(myInputAttr);
    addAttribute(myOutputAttr);
    addAttribute(myOutputAttrTwo);

    attributeAffects(myInputAttr, myOutputAttr);
    return MS::kSuccess;
}
```

Custom Node Code Skeleton

- MPxNode::compute()
 - called when the node is asked to evaluate an output

```
MStatus myNode::compute(const MPlug& plug, MDataBlock& dataBlock)
{
    if (plug == myOutputAttr )
    {
        //your compute algorithm for output1
    }
    if (plug == myOutputAttrTwo)
    {
        //your compute algorithm for output2
    }
    return MStatus::kUnknownParameter
}
```



Examples

- Devkit Plug-in Examples:
 - C:\Program Files\Autodesk\Maya2011\devkit\plug-ins
- The Maya API Documentation contains a wealth of information on all aspects of the API.

Autodesk®
Maya® 2011

Contents Index Search Favorites

Developer Resources
API Guide
Maya API Introduction
Selecting with the API
Command plug-ins
DAG Hierarchy
Writing a Shading Node
Dependency graph plug-ins
Manipulators
Shapes
Writing a Hardware Shading Node
Writing a Custom Transform Node
Writing a Deformer Node
Writing File Translators
Multithreading plug-ins
Polygon API
Working with Qt
Setting up your build environment
Maya Python API
Distributing Maya Plug-ins
Technical Notes
Translated resource file for Japanese
Example Plug-ins
Appendices
Environment Variables
File Formats

API Docs

Developer Resources > API Guide > Maya API introduction >

Introduction

Autodesk® Maya® is an open product. This means that anyone outside of Autodesk can change Maya's existing features or add entirely new features. There are several ways you can modify Maya:

- MEL™—(Maya Embedded Language) is a powerful and easy to learn scripting language. Most common operations can be done using MEL.
- Python™—is a powerful and easy to learn scripting language, which provides an interface to the Maya commands.
- C++ API—(Application Programmer Interface) provides better performance than MEL or Python. You can add new objects to Maya using the API, and code executes approximately ten times faster than when you perform the same task using MEL. Also, you are able to execute MEL commands from the API.
- Maya Python API—Based on the API and allows the API to be used through the Python scripting language.

Please see the "MEL and Expressions" book for an introduction to MEL and the "Python" book for its equivalent interface. This book provides a technical introduction to the Maya API and the Maya Python API.

Overview

The Maya API is a C++ API that provides internal access to Maya and is available on the following platforms: Microsoft® Windows®, Linux®, and Apple® Mac OS® X. You can use the API to implement two types of code resources: plug-ins which extend the functionality of Maya, or stand-alones such as console applications which can access and manipulate a Maya model.

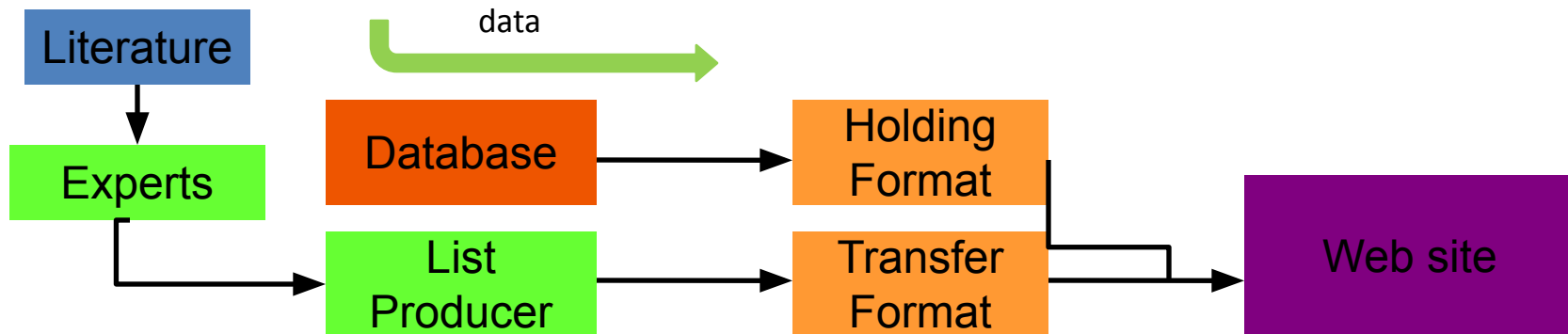
Plug-ins can be built in two ways:

- As dynamic or relocatable libraries which are loaded into Maya using standard operating system functionality. Plug-ins work by accessing the symbol space of the host application Maya. Access to the symbol space of other loaded plug-ins is not available.
- As scripts that use the Maya Python API.

Many of the examples in the API development kit are provided with both C++ and Python source codes. To allow both versions to be loaded into Maya at the same we have adopted the convention of prefixing the commands and nodes from Python plug-ins with "sp" (e.g. spHelix). You are not required

How does Dependency Graph work?

- Control system for Maya
- Glue that holds together disparate operations and lets them work together seamlessly
- DG is not a usual dataflow system....



How does Dependency Graph Work?

Two step Push-Pull mechanism:

- Dirty Propagation
- Evaluation



Image courtesy of Johan Vikström, Shilo, The Spine, National Film Board of Canada Production ©, Ool Digital, Mikros Image

Dirty Propagation

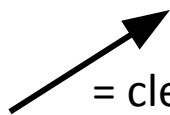
- Maya DG caches values
- Uses dirty system to denote elements that require updating:
 - Attributes
 - Connections
- MEL Commands:
 - dgdirty
 - isDirty



Image courtesy of Johan Vikström, Shilo, The Spine, National Film Board of Canada Production ©, Ool Digital, Mikros Image

Data Flow Example

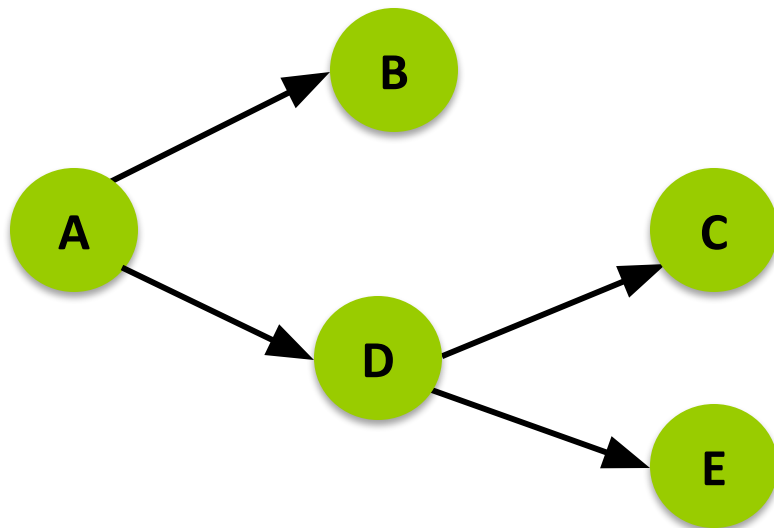
- Key



= clean connection

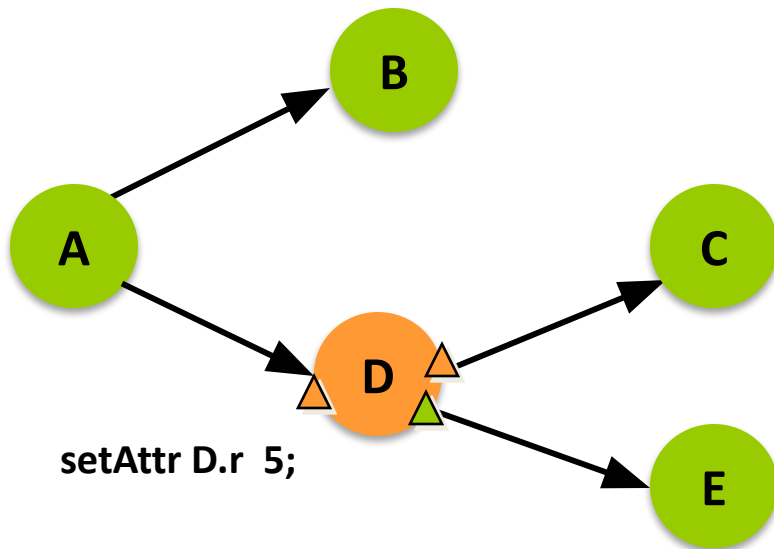


= dirty connection



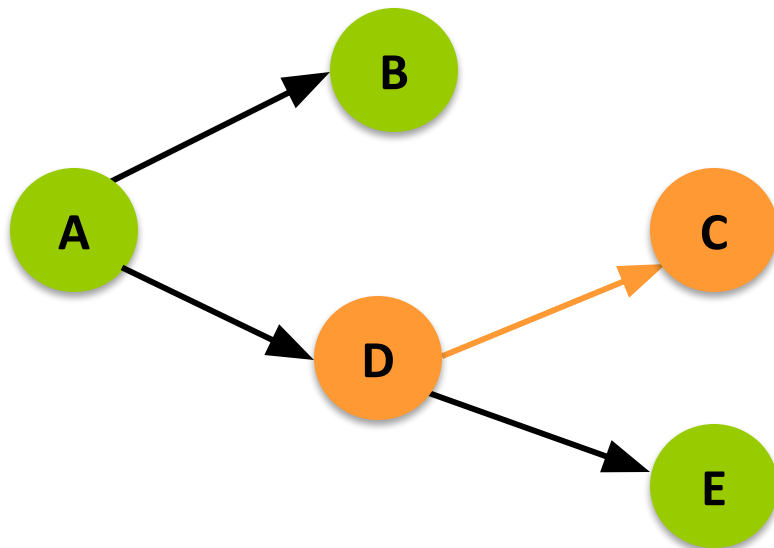
The Dirty Process

Initiated by value changes



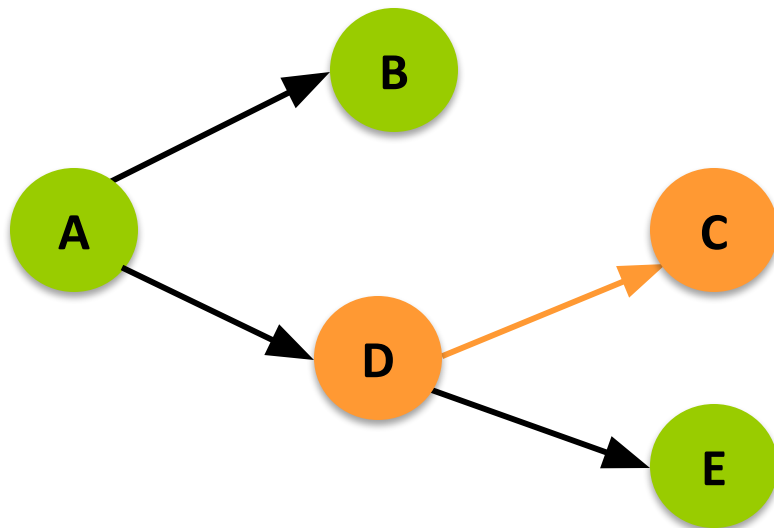
The Dirty Process

Dirty message propagates forward



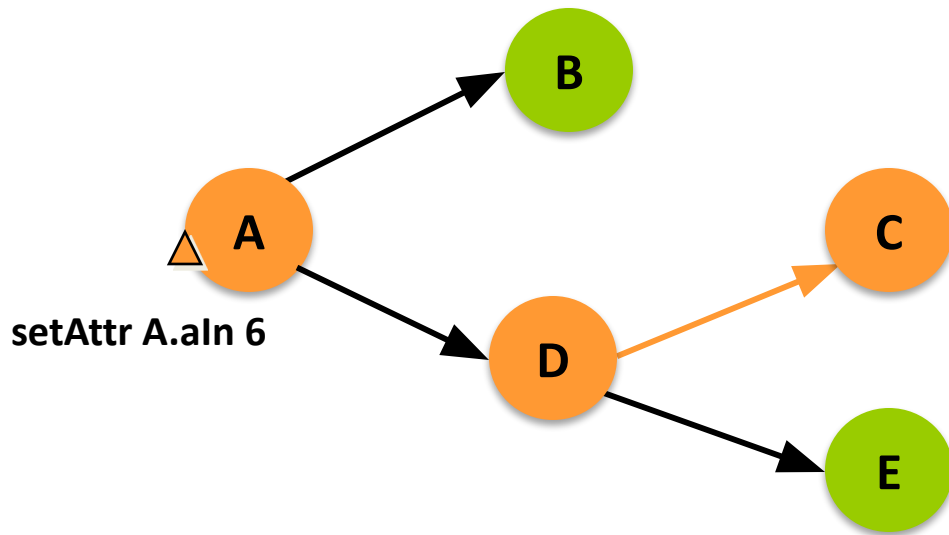
The Dirty Process

No evaluation has been requested. Data remains dirty.



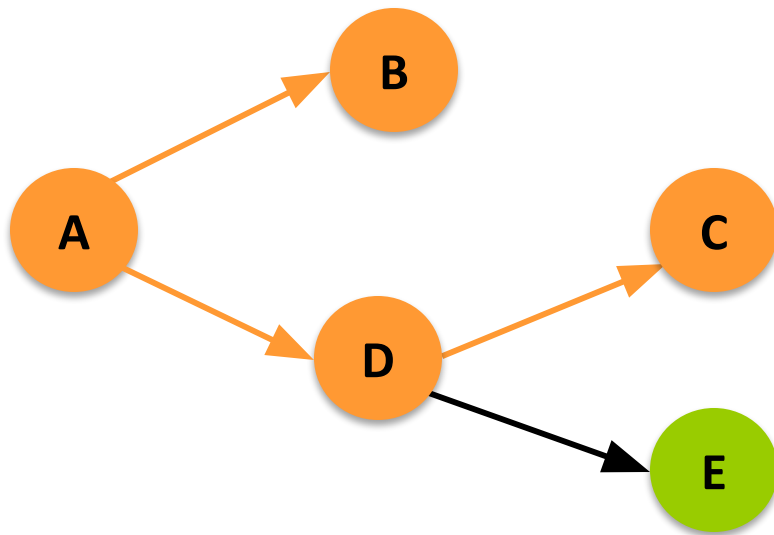
The Dirty Process

Now an input on A changes



The Dirty Process

Dirty propagates out all outgoing connections

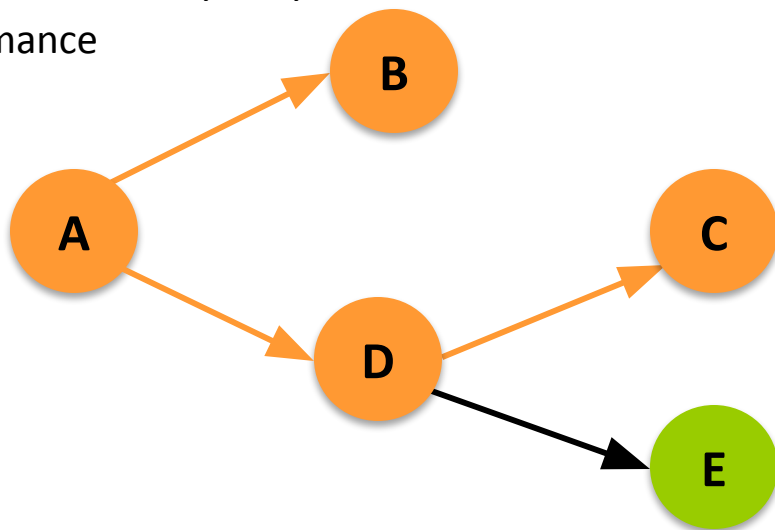


The Dirty Process

B and D propagate dirty to affected attributes

C will not receive dirty message

- Connection to C is already dirty
- Helps performance

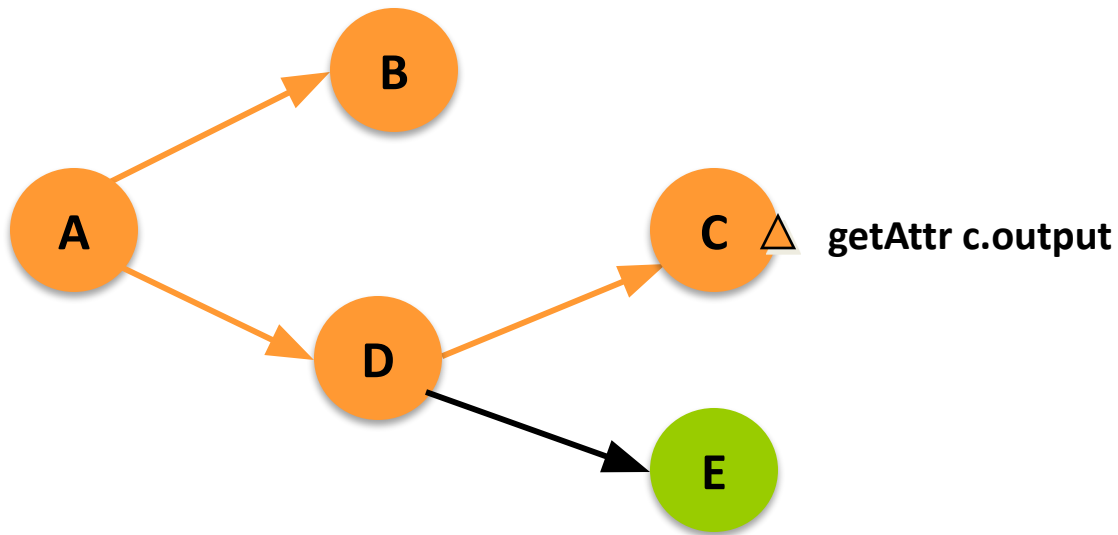


The Evaluation Process

- Lazy Evaluation: On demand
- Evaluation is triggered when values are requested:
 - Viewport refresh
 - Attribute editor
 - Channel box
 - Rendering
 - getAttr command
 - Etc...
- Evaluation is minimal
 - Only requested values are computed
 - Non-requested values are left dirty

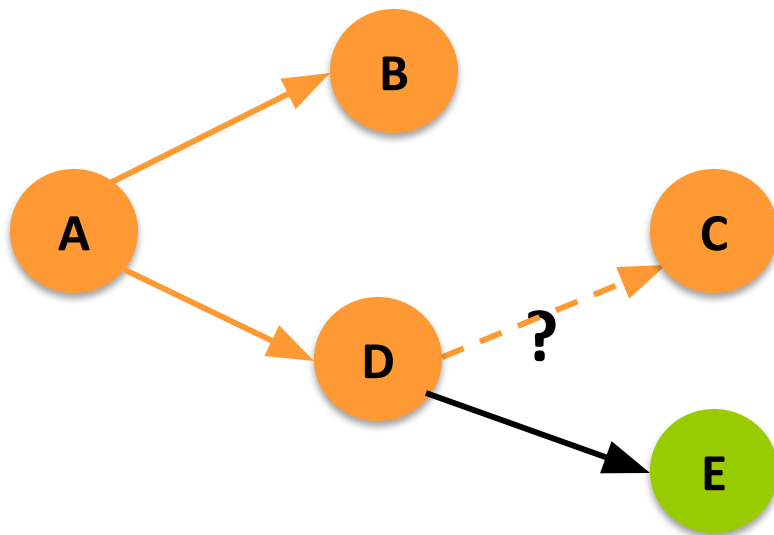
The Evaluation Process

Example: `getAttr C.output`



The Evaluation Process

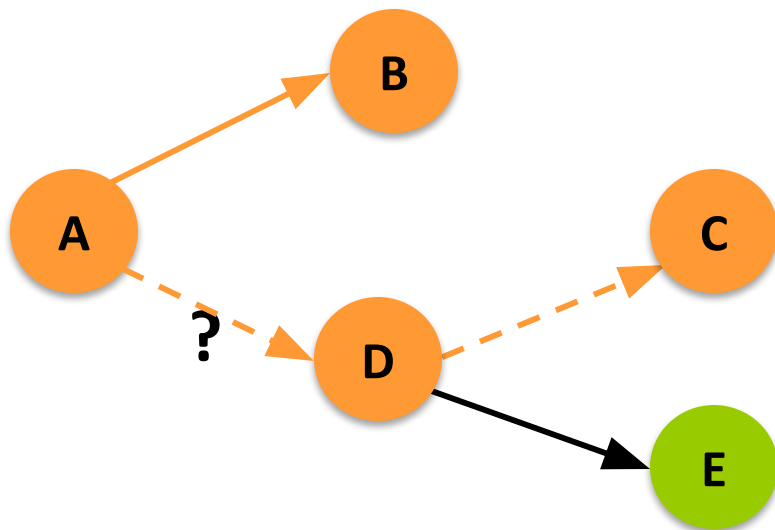
C computes: requests input value from connection



The Evaluation Process

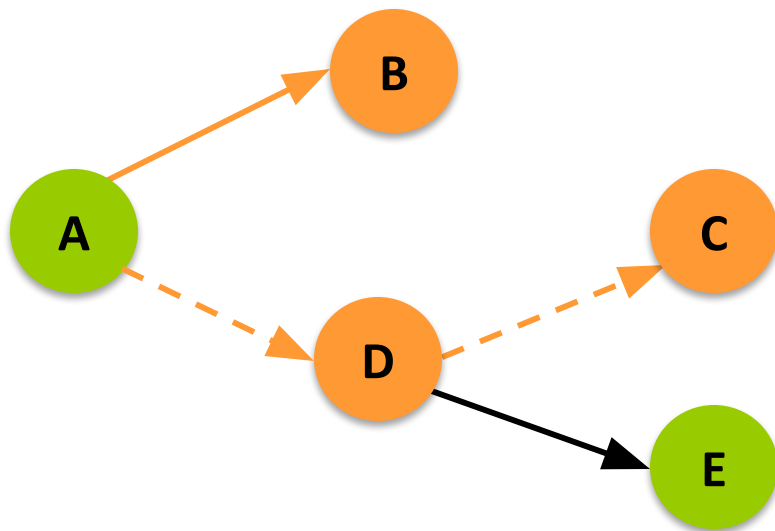
D computes.

D requests input value from connection



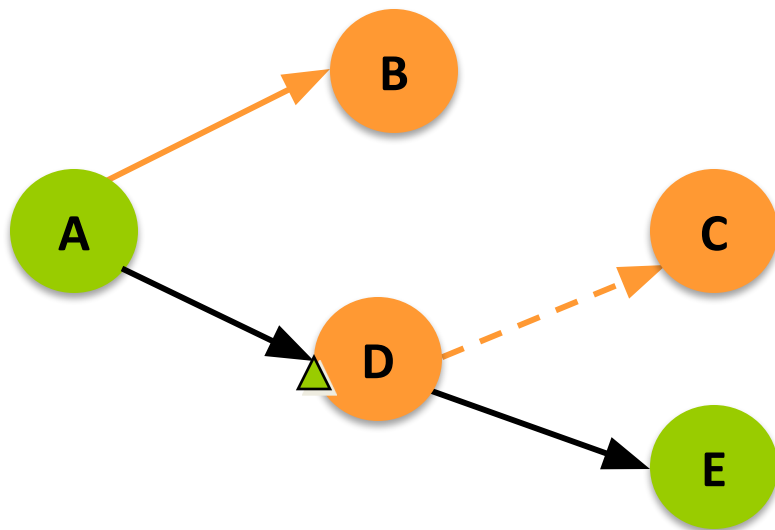
The Evaluation Process

A computes requested output



The Evaluation Process

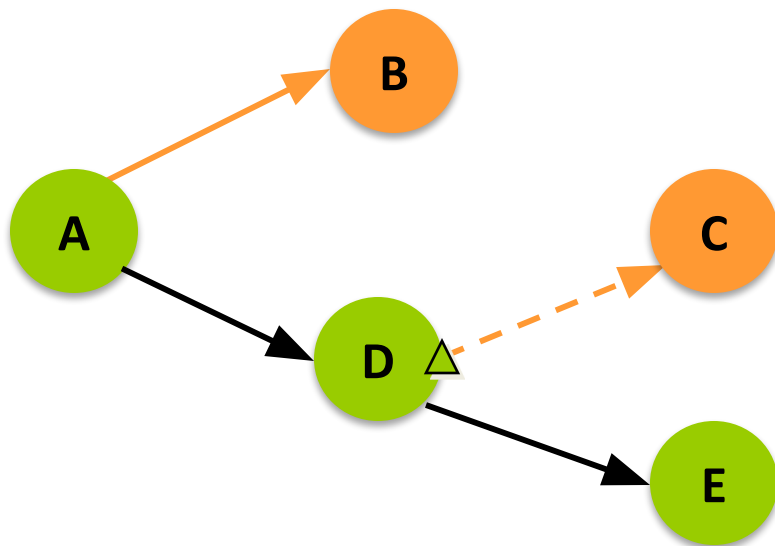
Value copied forward to D's input



The Evaluation Process

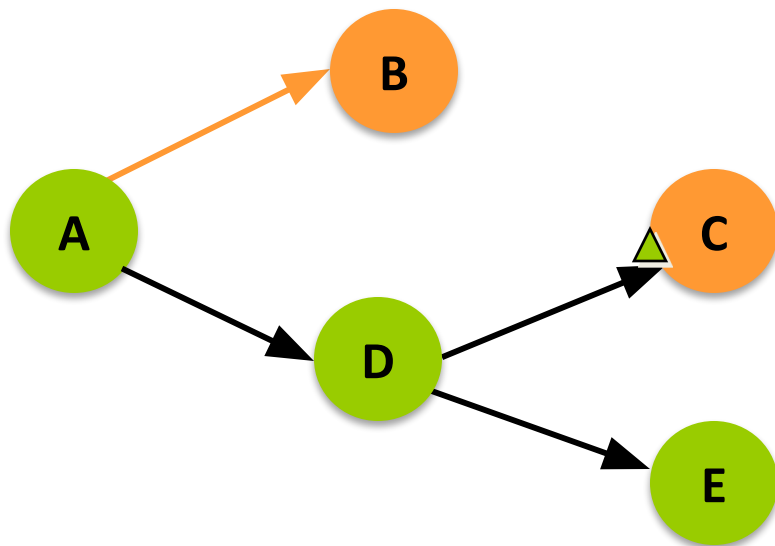
D computes requested result.

D sets value in output.



The Evaluation Process

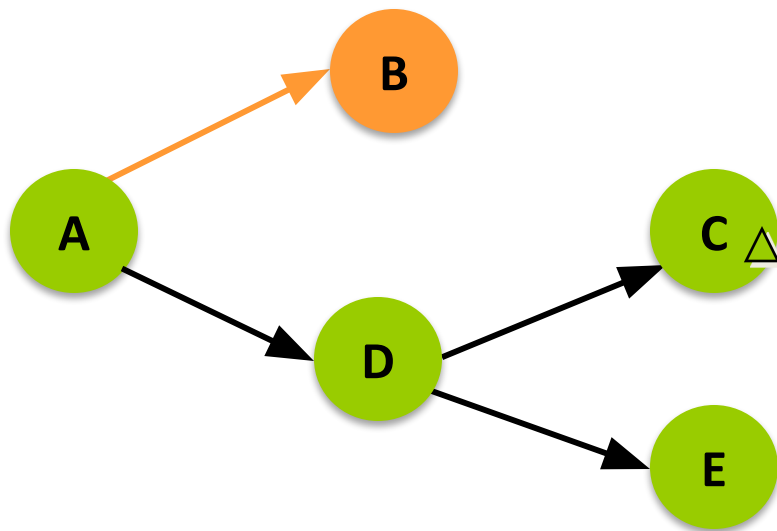
Value is copied forward to C



The Evaluation Process

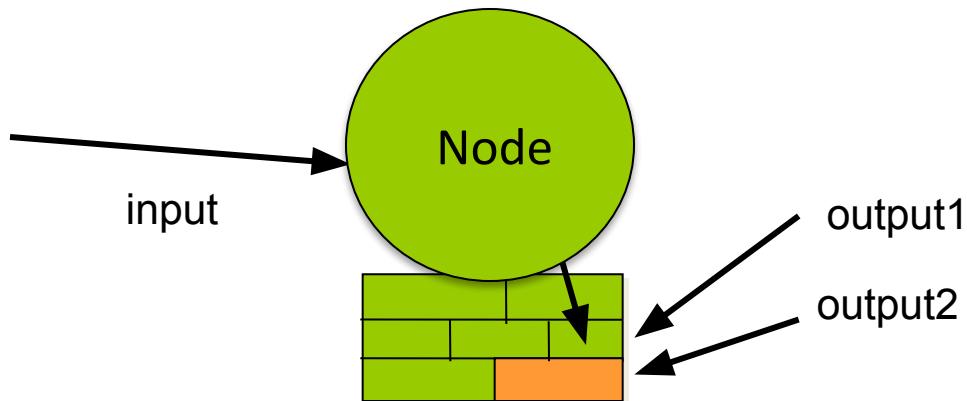
C computes requested output.

B remains dirty.



The Evaluation Process

Only requested outputs are computed, unless node's compute method does more than requested



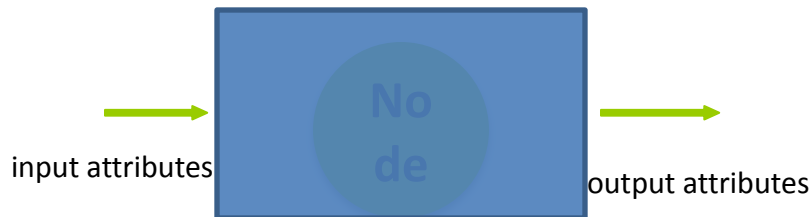
Correct Coding with DG



Image courtesy of Johan Vikström, Shilo, The Spine, National Film Board of Canada Production ©, Ool Digital, Mikros Image

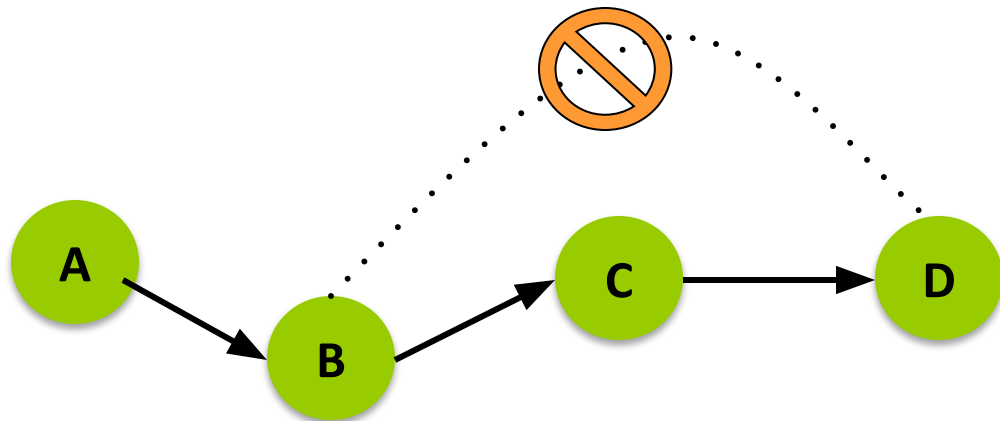
Correct Coding with DG

- Common misuse:
 - put command-type tasks into a custom node implementation
 - execute commands to change the status of current DG
 - get or set values on other nodes
- Black Box Rule



Black Box Rule

- Black-box operation of node is what makes it all work.



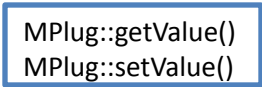
A Closer Look at MPxNode::compute()

- You can not control when compute() is getting called, Maya control when it gets called
- Compute() is called in Evaluation phase
- Inside compute(): avoid sending dirty messages
 - Don't execute commands
 - Don't get or set values on other nodes

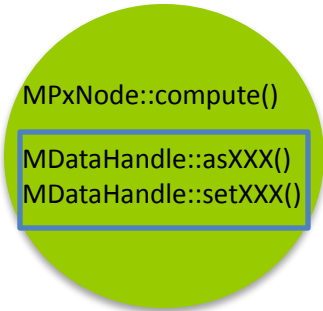


A Closer Look at MPxNode::compute()

- Inside compute(): avoid sending dirty messages
 - Get/set data only through datablock using data handles, don't set data via plugs (i.e. MPlug::setValue)
 - setting data via plug propagates dirty, datahandle does not
 - datahandle set/get methods are more efficient



```
MPlug::getValue()  
MPlug::setValue()
```



```
MPxNode::compute()  
MDataHandle::asXXX()  
MDataHandle::setXXX()
```

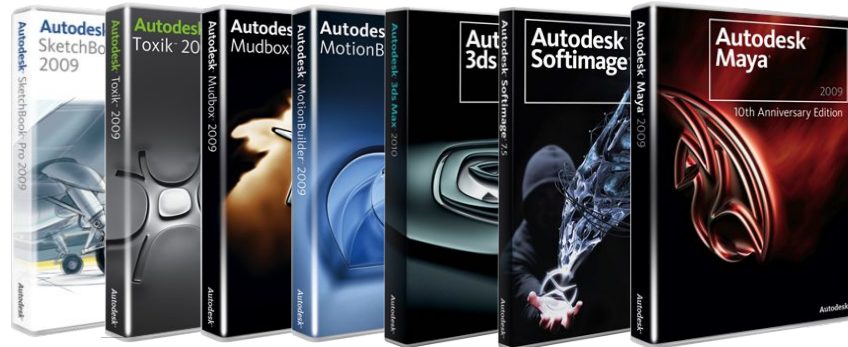

Learning Resources

- Maya Developer Center:
<http://www.autodesk.com/developmaya>
- Questions and Problems: ADN
<http://www.autodesk.com/adn>
Maya API White Paper, DevTV, Webcast training
- Discussion Forum: The AREA
<http://area.autodesk.com/forum/autodesk-maya/sdk/>



Image courtesy of Johan Vikström, Shilo, The Spine, National Film Board of Canada Production ©, Ool Digital, Mikros Image

Q & A



Thank you!