A Closer Look at Dependency Graph

Module 7

Autodesk

Autodesk Maya Python API Training

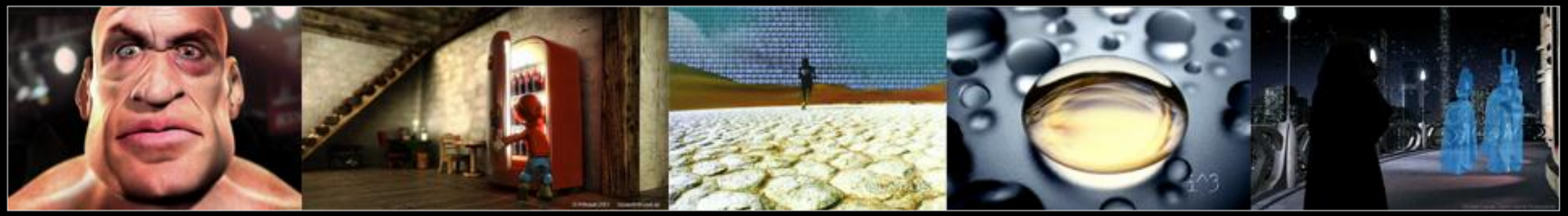
Kristine Middlemiss, *Senior Developer Consultant*
Autodesk Developer Network (ADN)

Autodesk

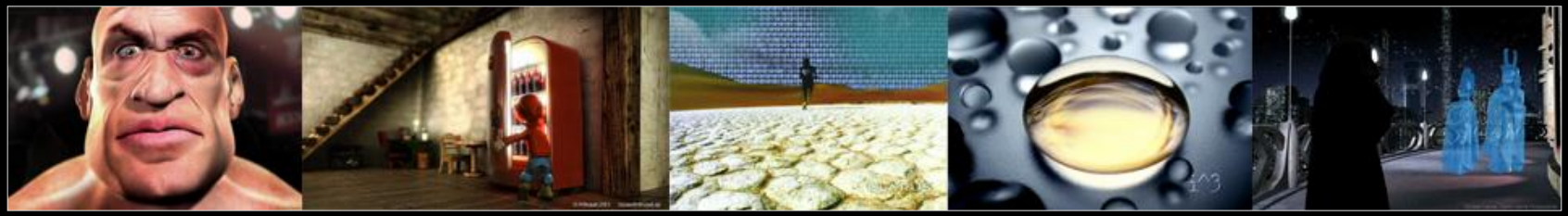
Homework Review

Agenda

- Dependency Graph Push-Pull Mechanism
- Data Storage
- Data Access



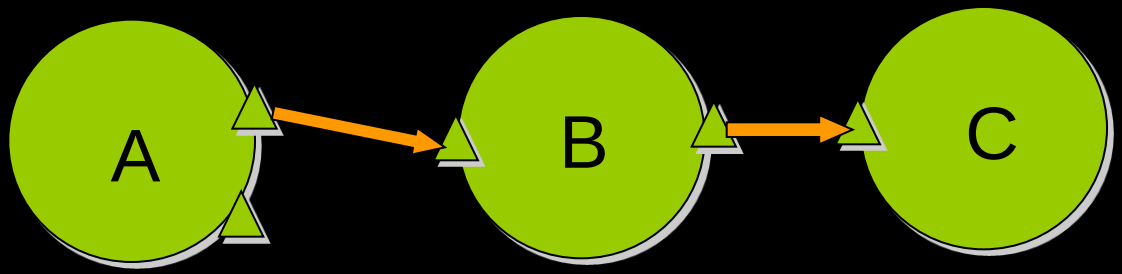
Dependency Graph Push-Pull Mechanism



Autodesk®

Dependency Graph

Dependency Graph and DG Nodes



Autodesk®

How DG works

Two step Push-Pull mechanism:

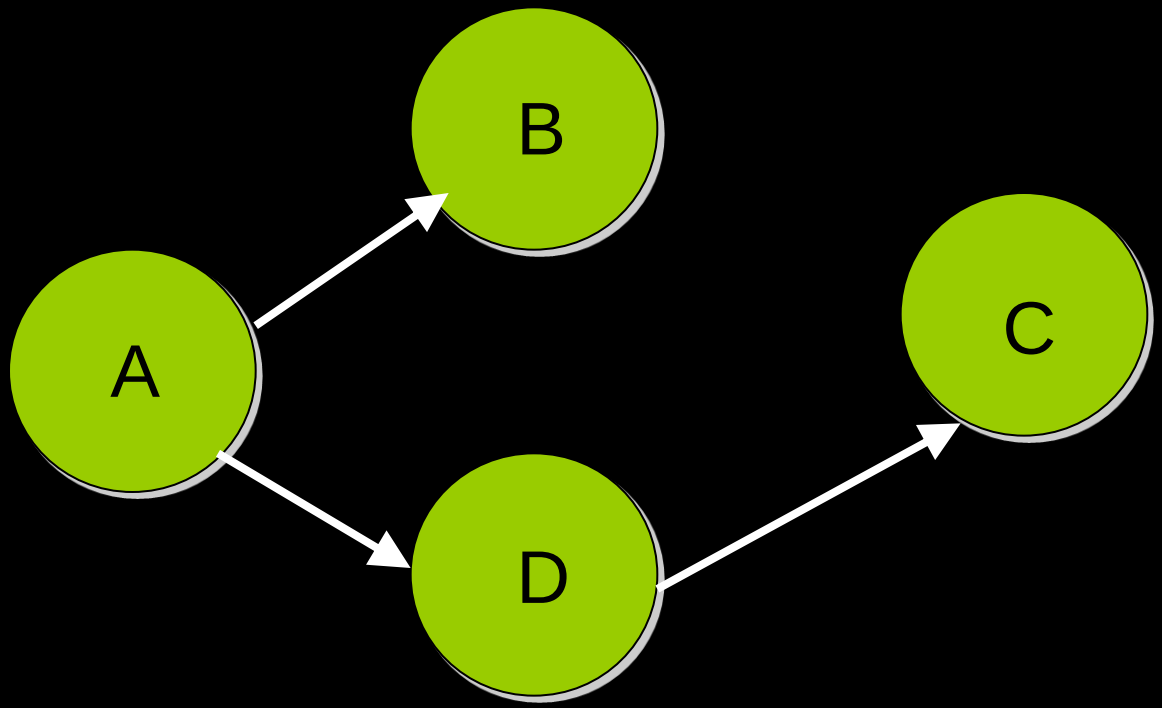
1. Dirty Propagation
2. Evaluation

The Dirty Process

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

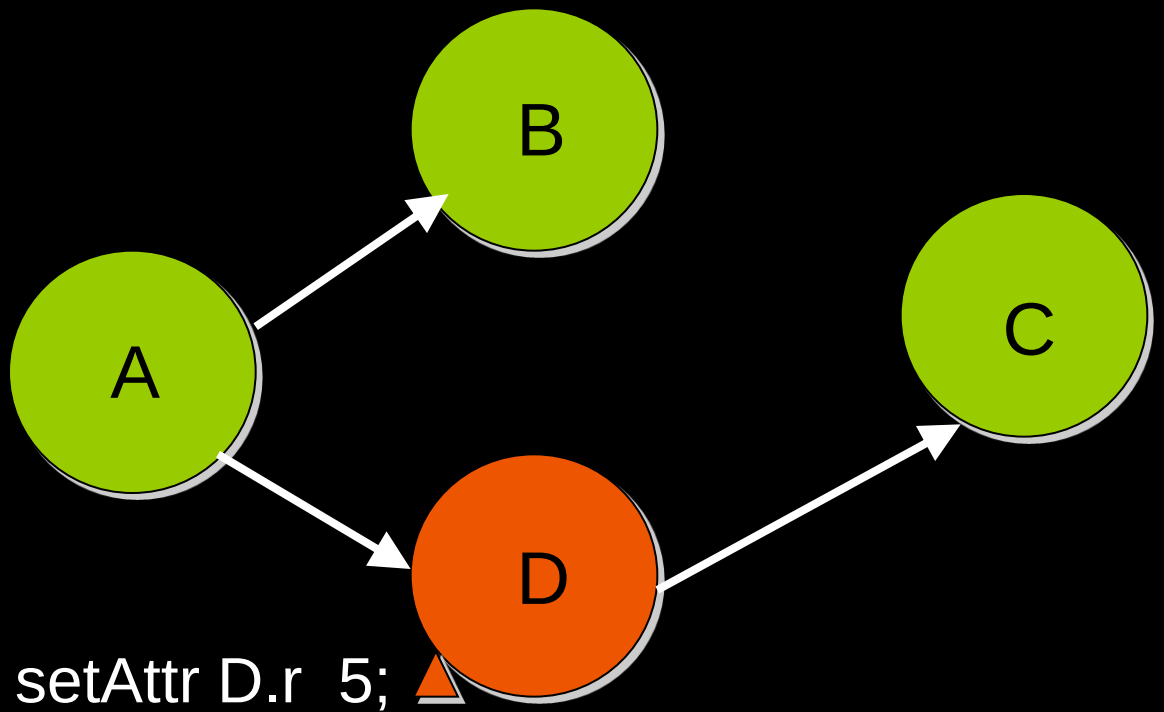
Data Flow Example

- **Key**



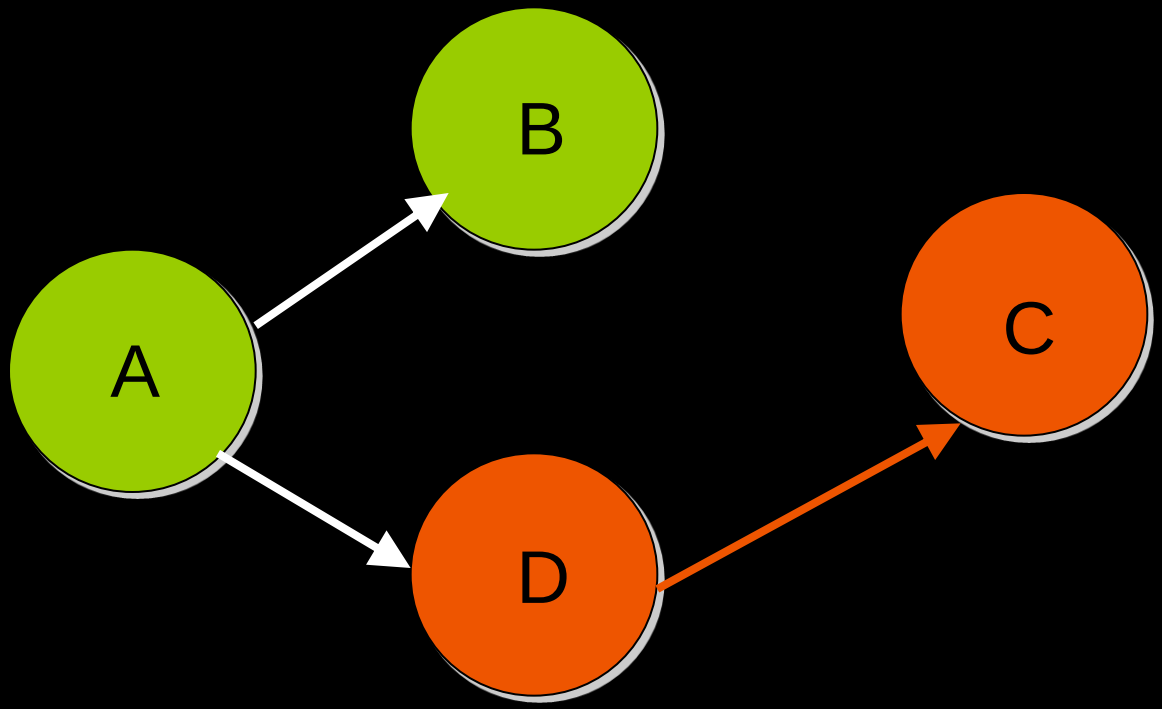
The Dirty Process

Initiated by value changes



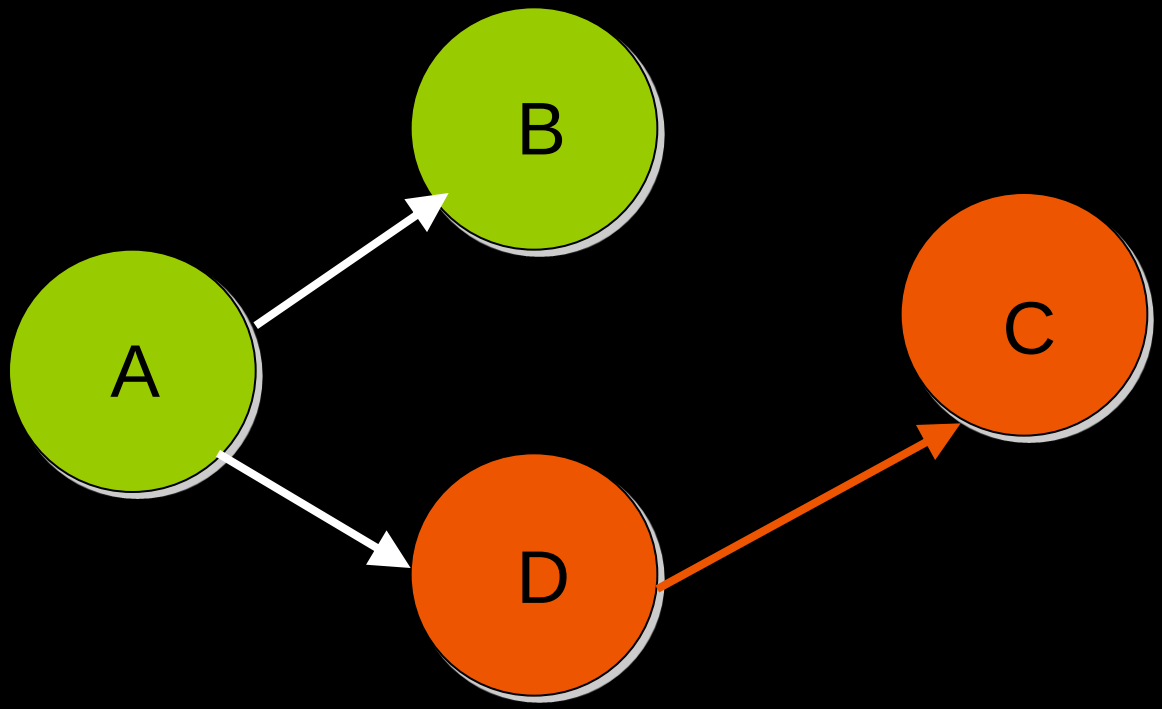
The Dirty Process

Dirty message propagates forward



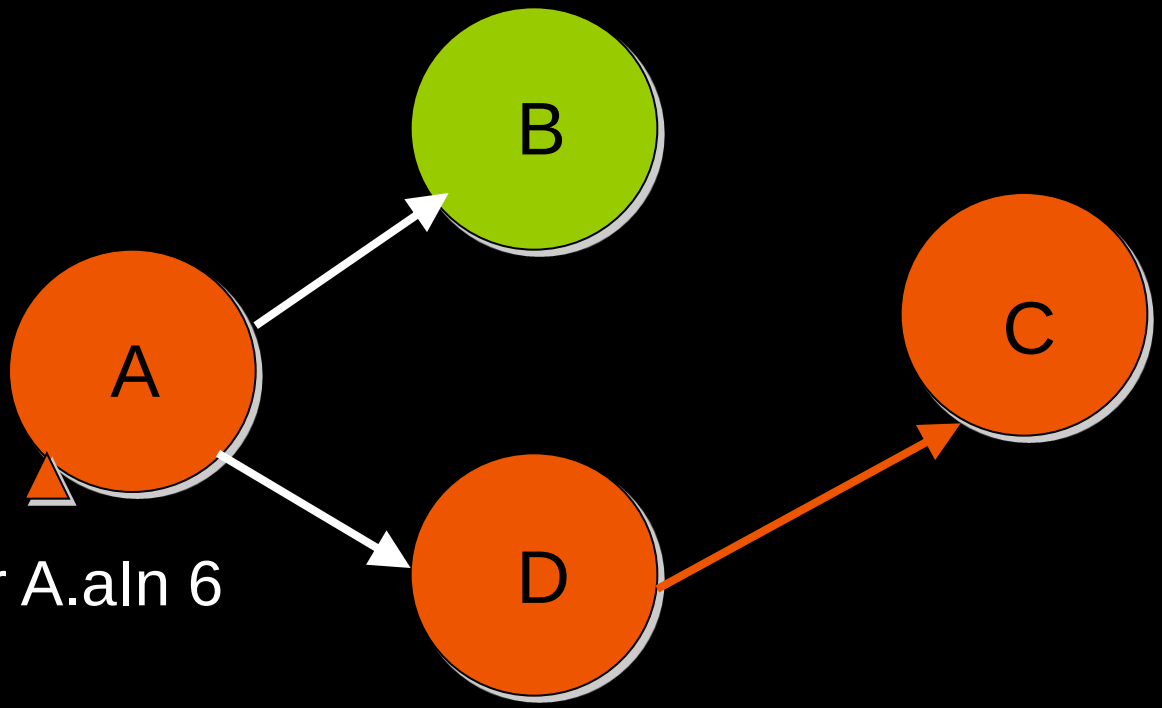
The Dirty Process

No evaluation has been requested. Data remains dirty.



The Dirty Process

Now an input on A changes

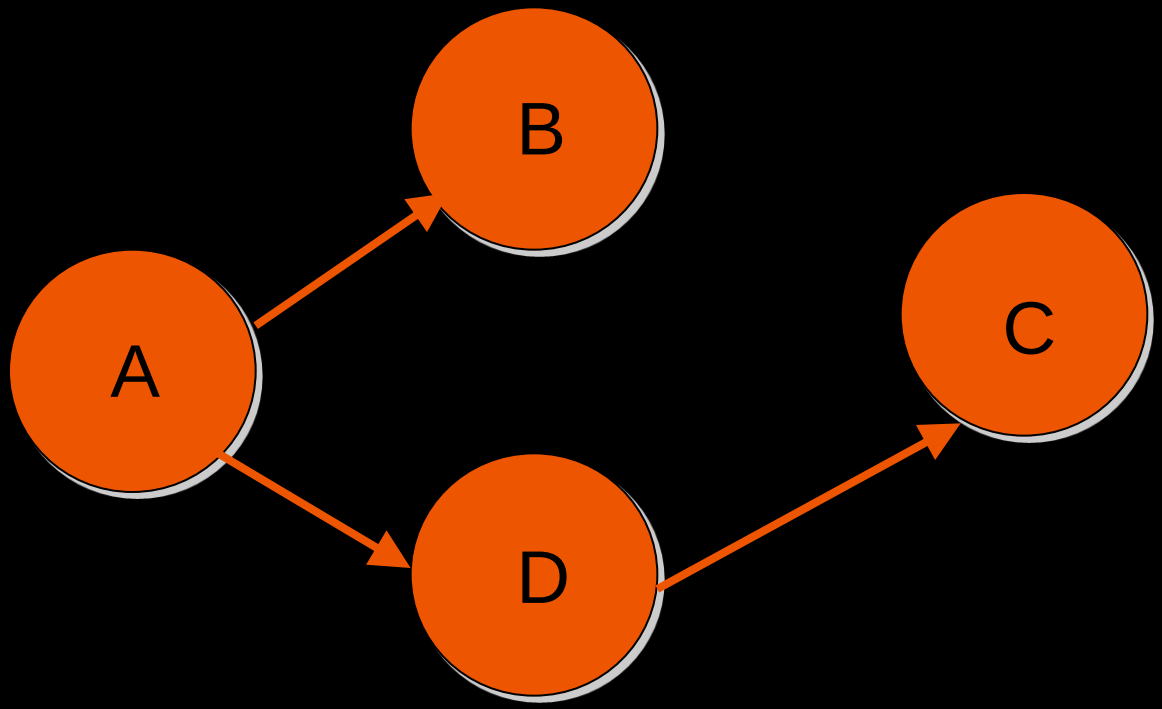


setAttr A.aIn 6

Autodesk®

The Dirty Process

Dirty propagates out all outgoing connections

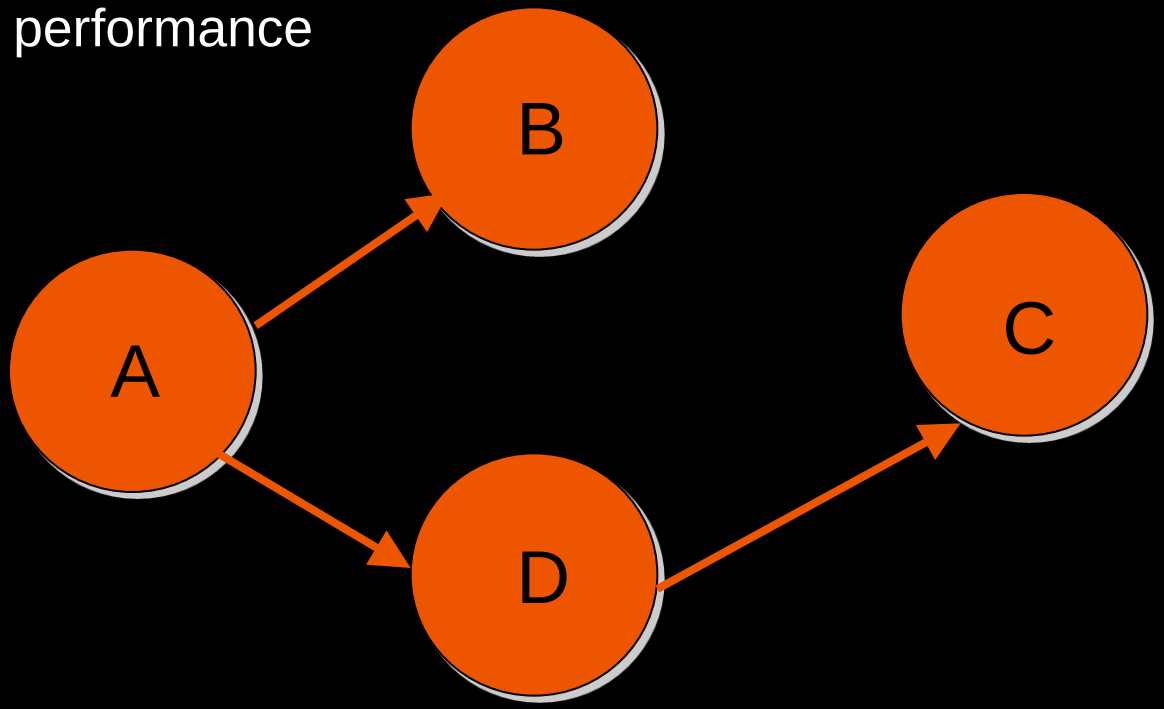


The Dirty Process

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

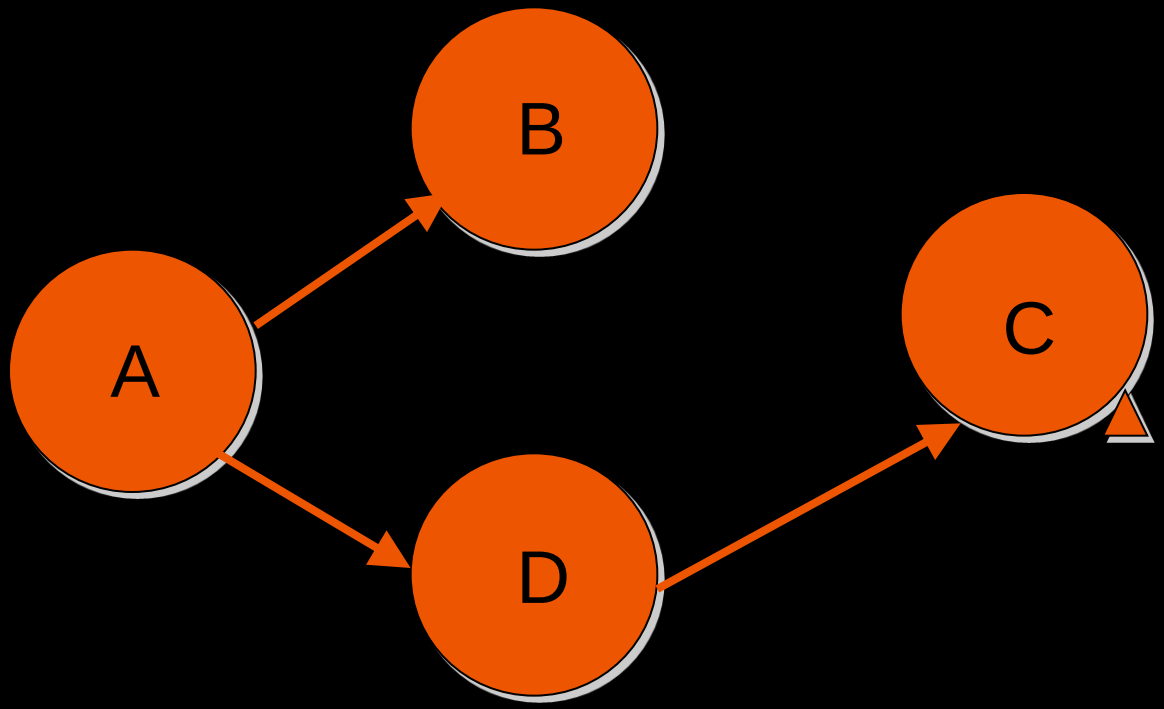
- Lazy Evaluation: On demand

- Evaluation is trigged when values are requested:
  - Refresh
  - Attribute editor
  - Channel box
  - Rendering
  - getAttr command
  - Etc…

- Evaluation is minimal
  - Only requested values are computed
  - Non-requested values are left dirty

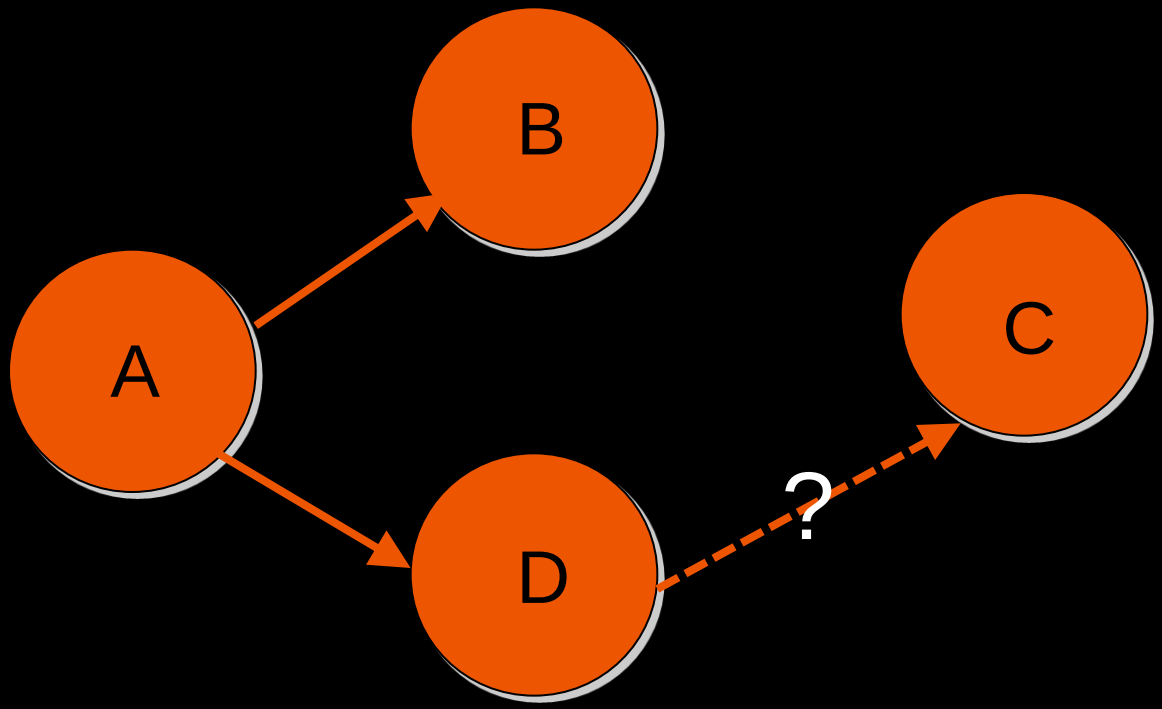
The Evaluation Process

Example: getAttr C.output



The Evaluation Process

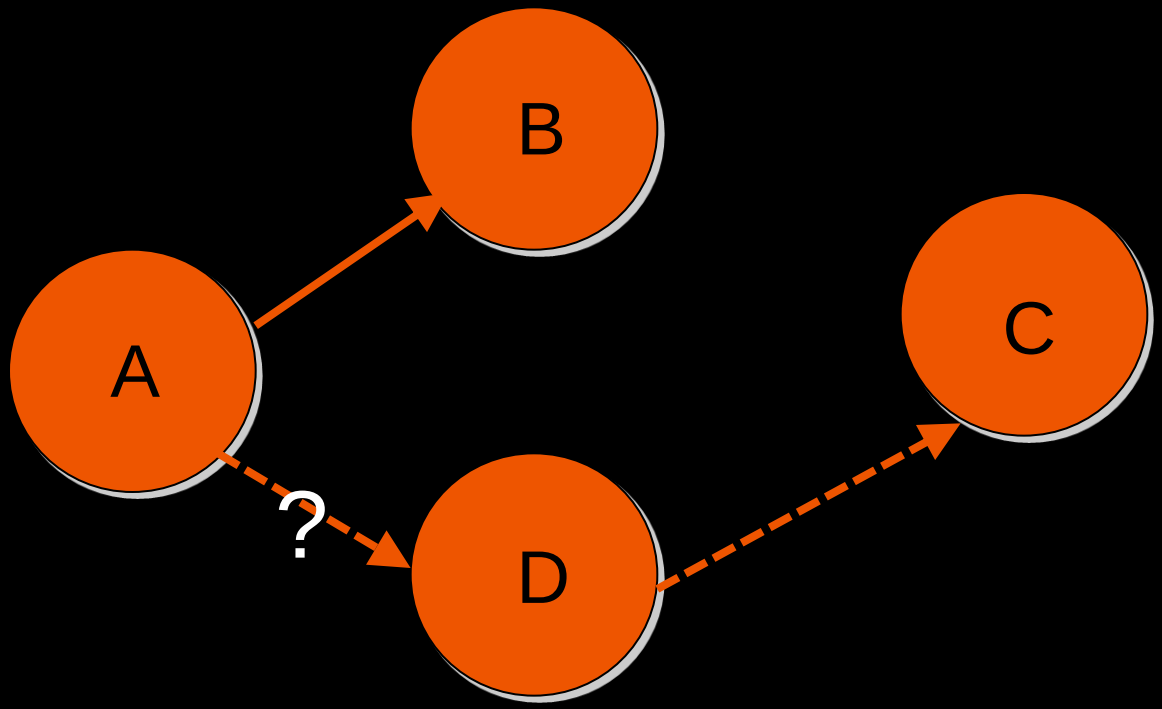
C computes: requests input value from connection



The Evaluation Process

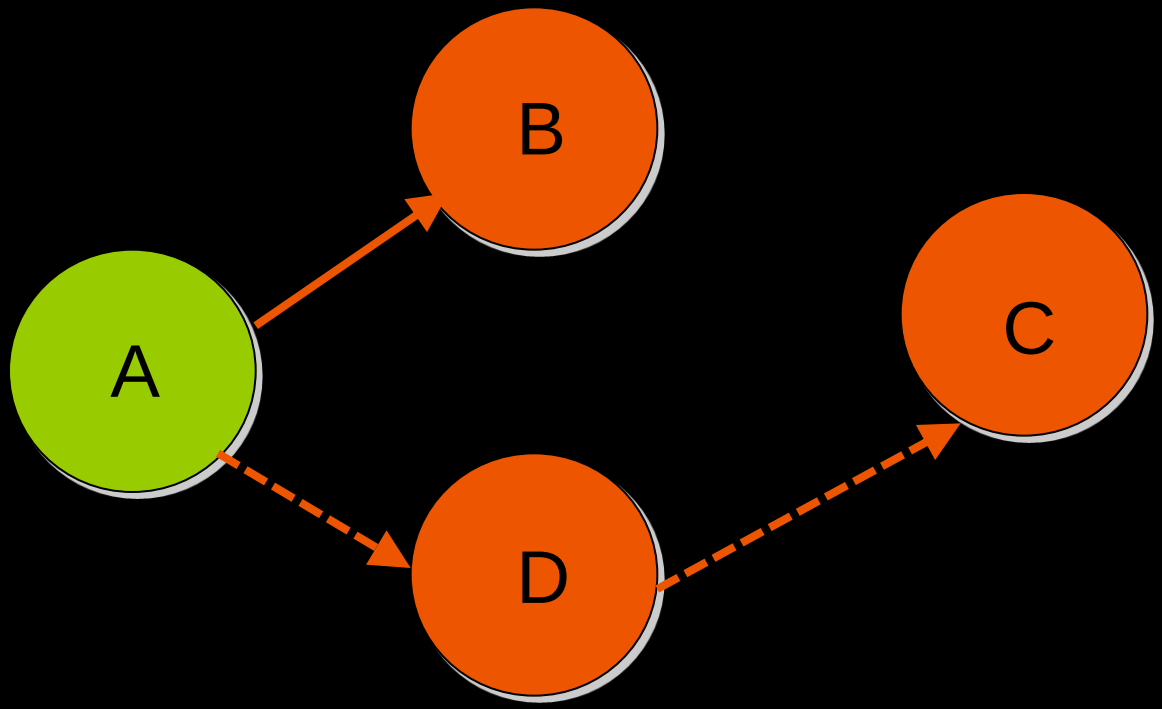
D computes.

D requests input value from connection



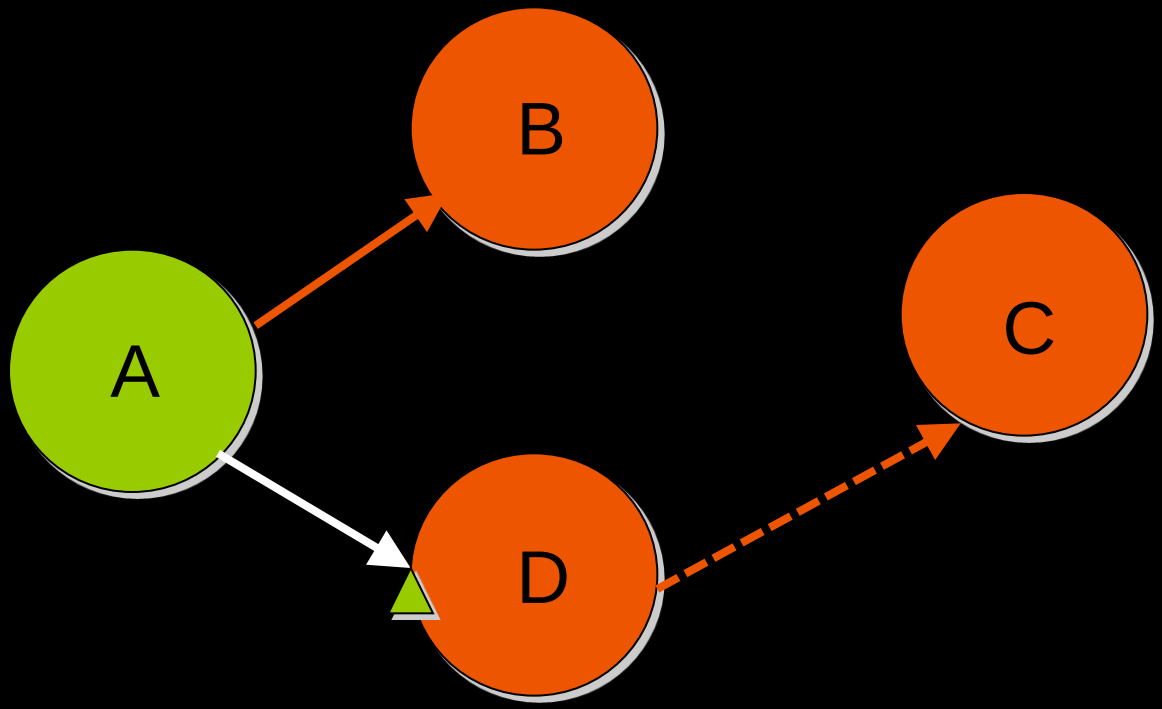
The Evaluation Process

A computes requested output



The Evaluation Process

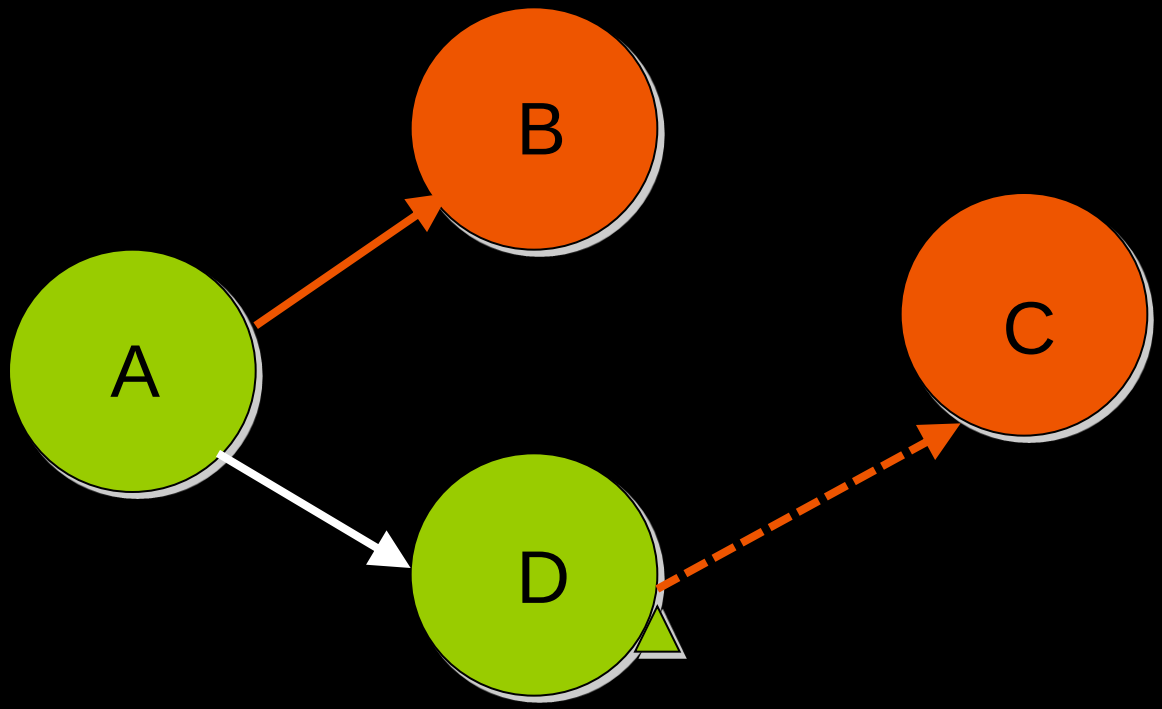
Value copied forward to D’s input



The Evaluation Process

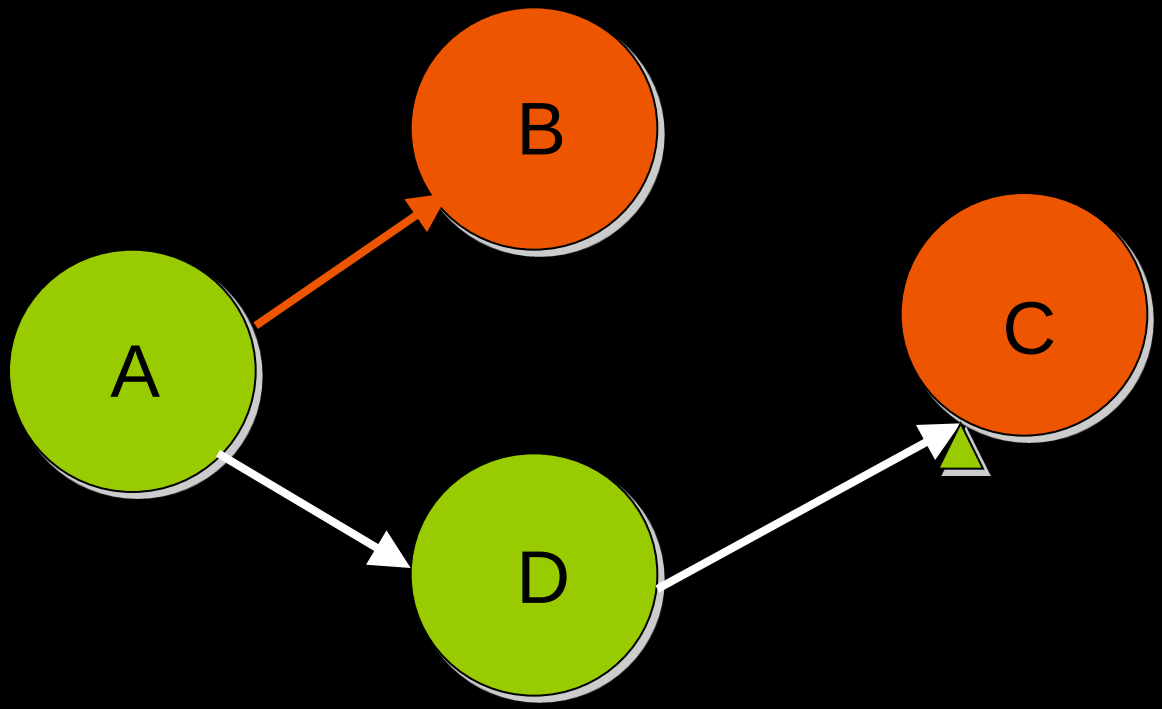
D computes requested result.

D sets value in output.



The Evaluation Process

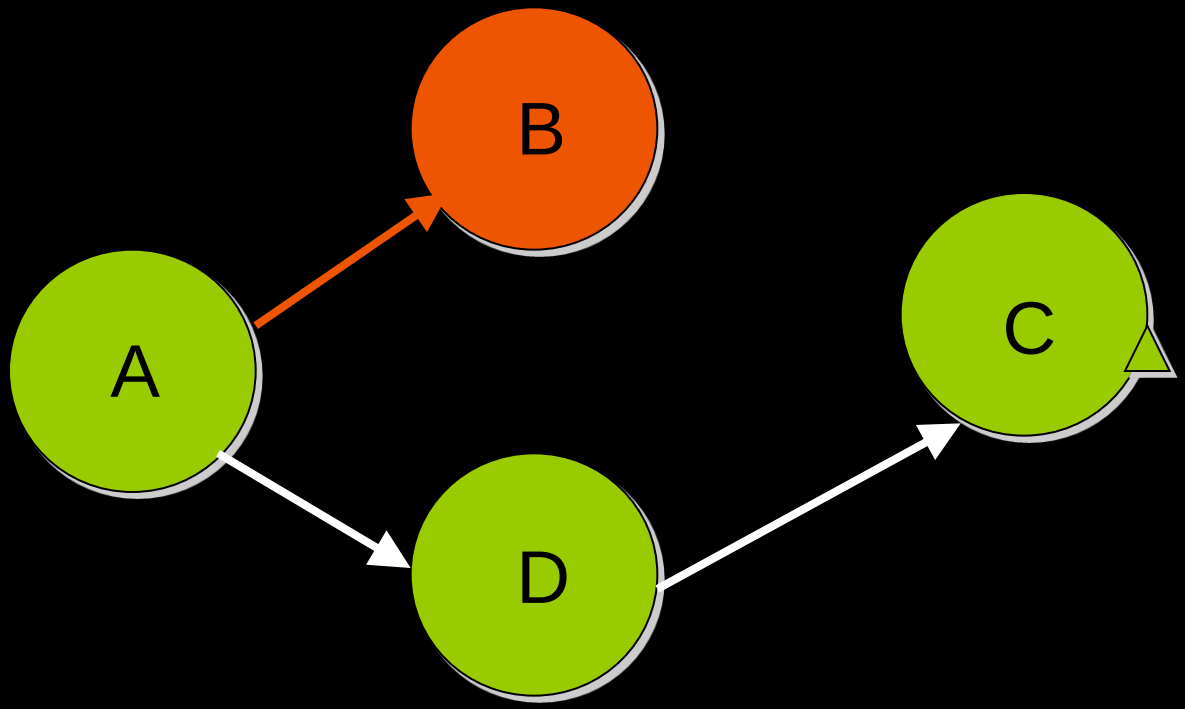
Value is copied forward to C



The Evaluation Process

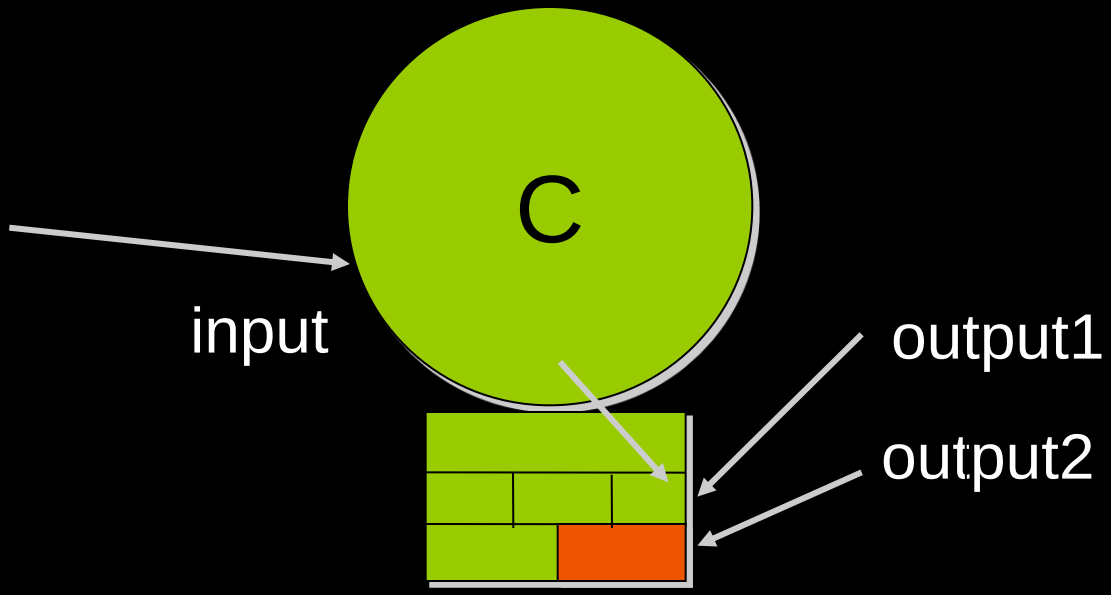
C computes requested output.

B remains dirty.



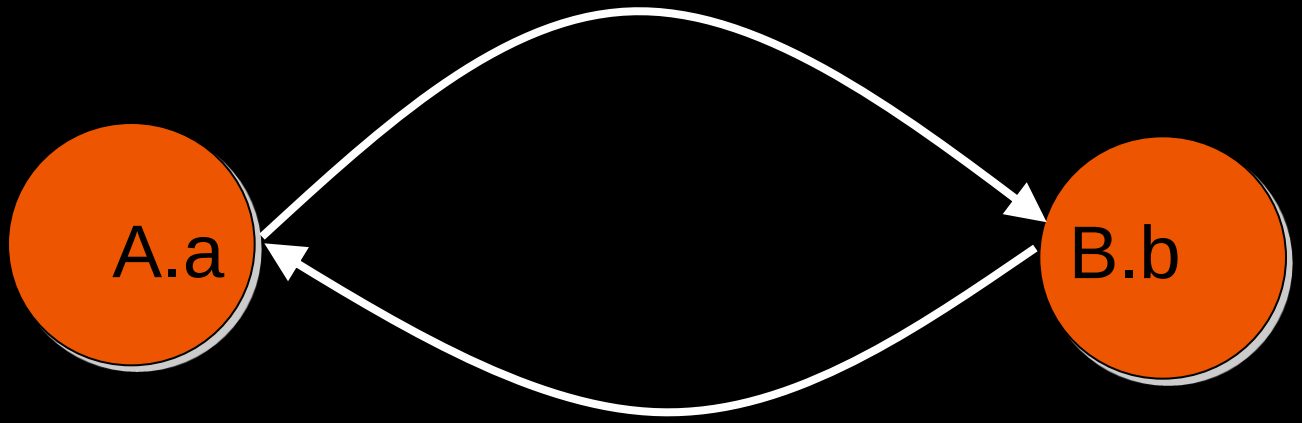
The Evaluation Process

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



Data Flow: Cycles

- Cycles are detected and avoided
- A.a may follow B.b, or vice versa, but not usually both; result is not guaranteed to be consistent



File Operations

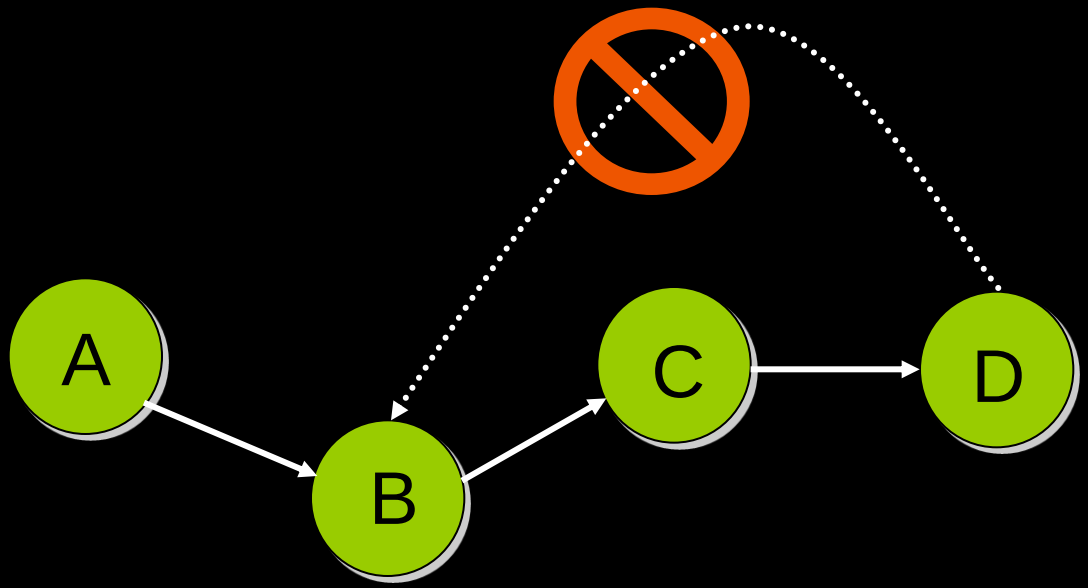
- File Open changes data flow
  - No dirty message propagation
  - No node evaluation until refresh
  - All connections assumed dirty
  - All setAttr values assumed clean
  - Unmentioned inputs are default (and clean)
  - Computed outputs are default (and dirty)
- File Save forces evaluation when required

Black Box Rule

Autodesk®

MPxNode::compute() Tips

- Get/set data only through datablock using data handles
- Avoid sending dirty messages, e.g.
  - Don’t set data via plugs (i.e. MPlug::setValue)
  - Don’t execute commands!
  - Don’t get or set values on other nodes



MPxNode::compute()

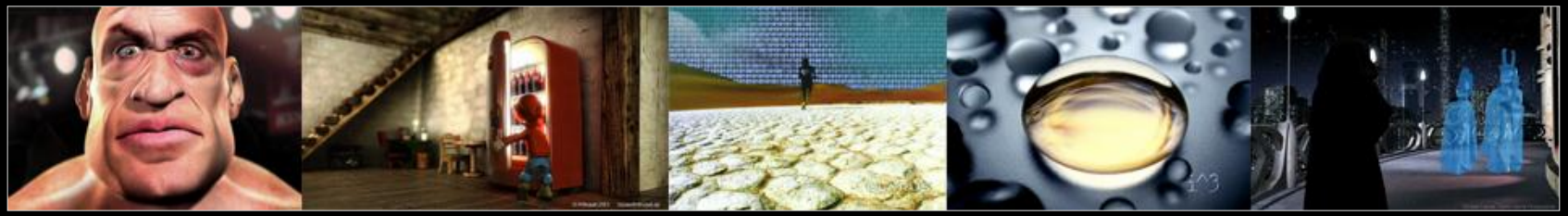
- MDataBlock::inputValue(MPlug& plug) will:
  - Return a smart pointer to read a value from the data block
  - Cause an evaluation, if dirty. (and subsequently mark it clean)

- MDataBlock::outputValue(MPlug& plug) will:
  - Retrieve a smart pointer to write a value to the data block
  - The value of that pointer is not guaranteed to be valid.

- Call MDataBlock::inputValue for all inputs affecting the requested output to ensure they are cleaned, even if the value itself is not necessary to compute the output.

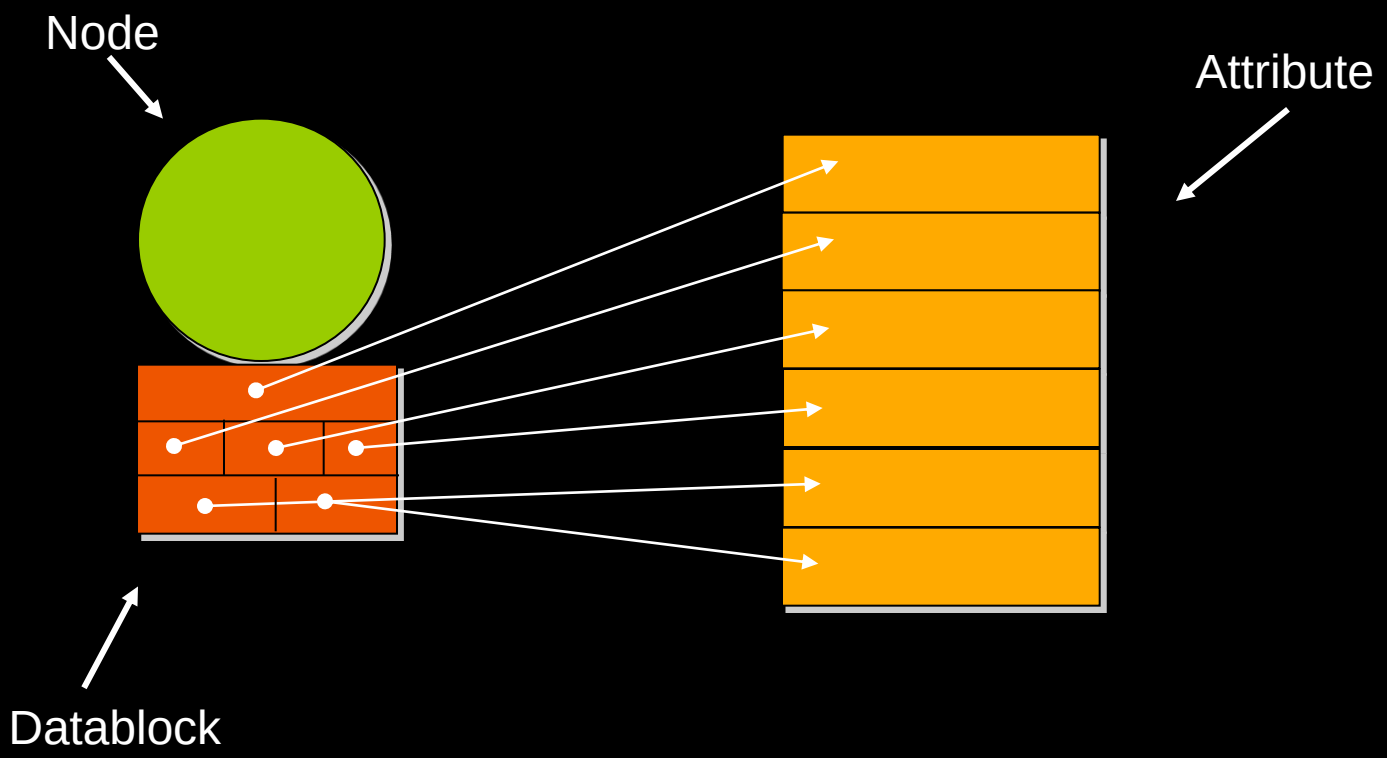
  attributeAffects() OR setDependentsDirty()

Data Caching in Dependency Graph



Autodesk®

Data Caching

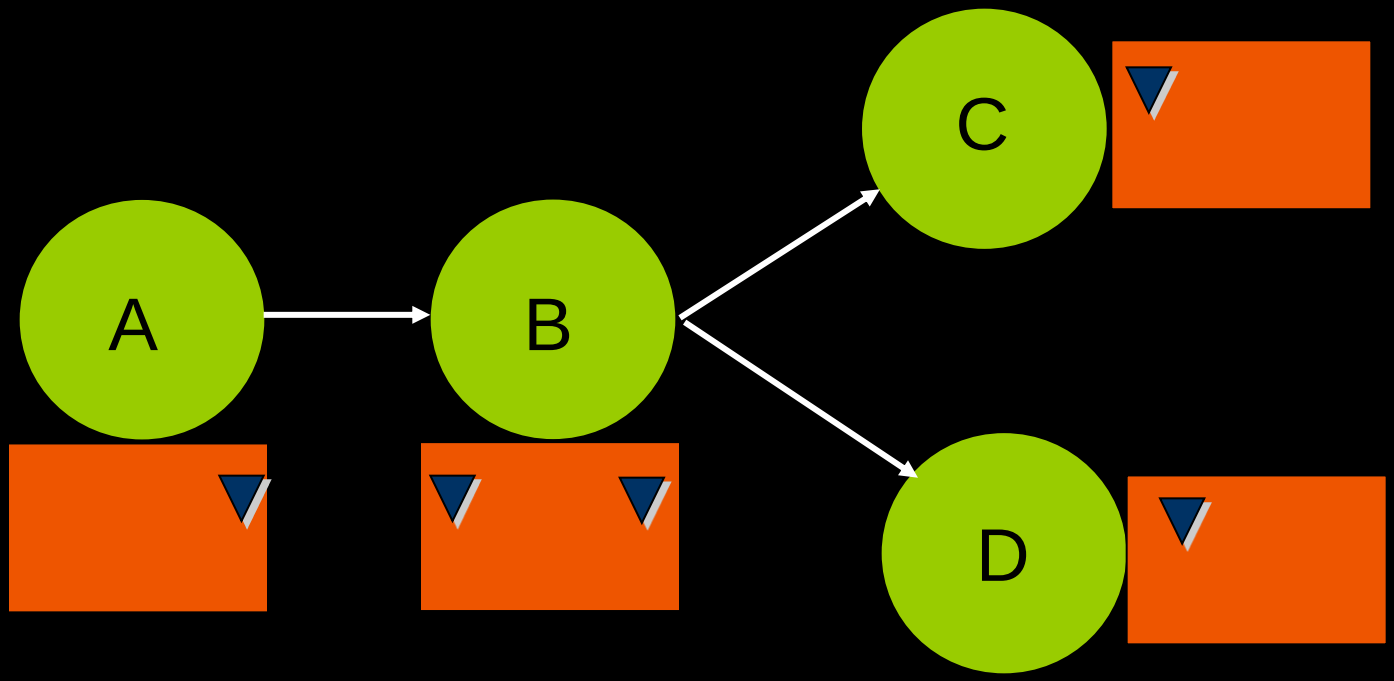


Data Caching

- Light data (numerical data)
  - duplicated in each datablock
  - converted as required (e.g. Int32 to Float)
- Heavy data (e.g. surfaces, curves, matrices)
  - reference counted
  - only duplicated when two nodes both want to modify it
- Message attributes contain no data, but evaluate the same as other attributes

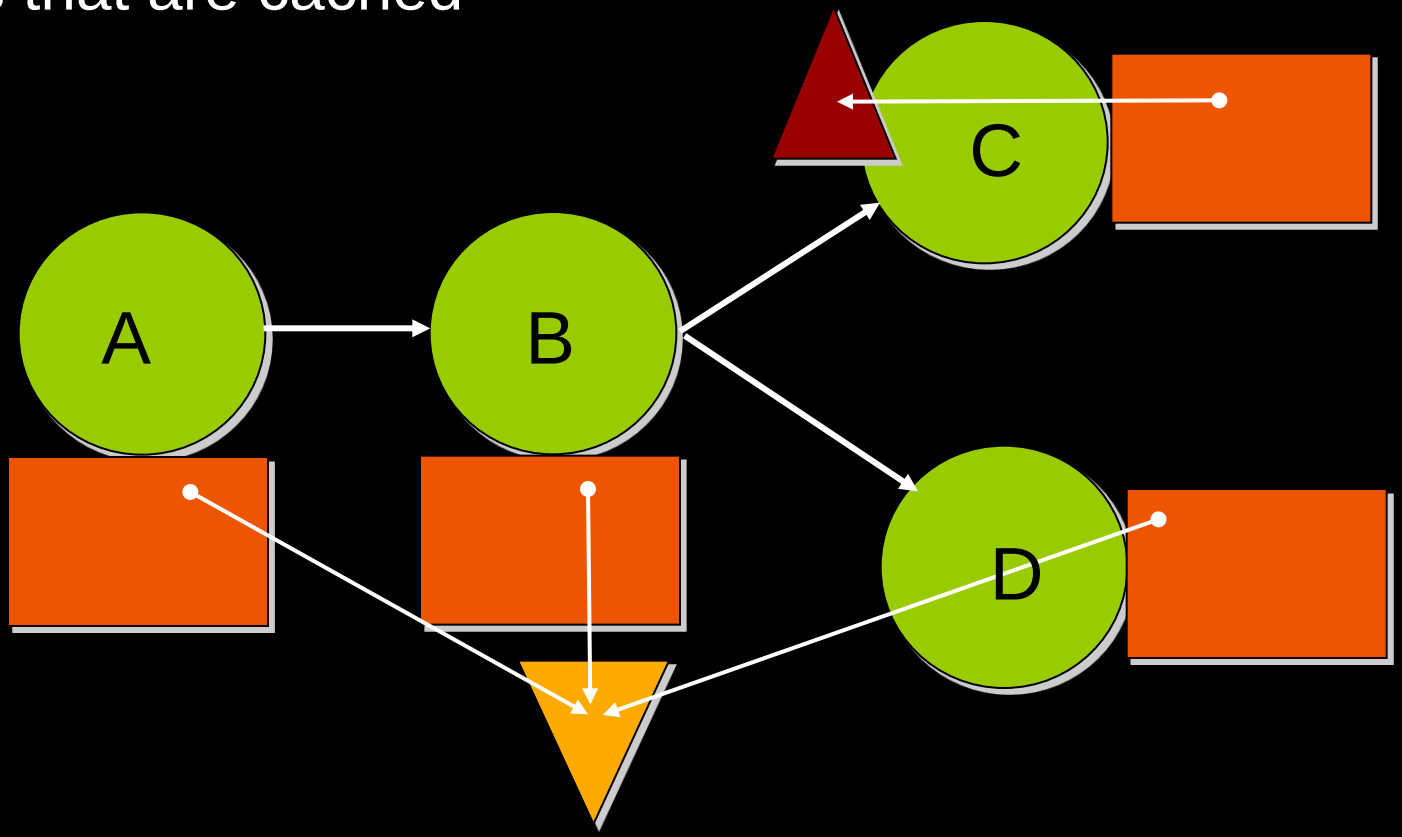
Light Data

During evaluation, numeric data is copied everywhere

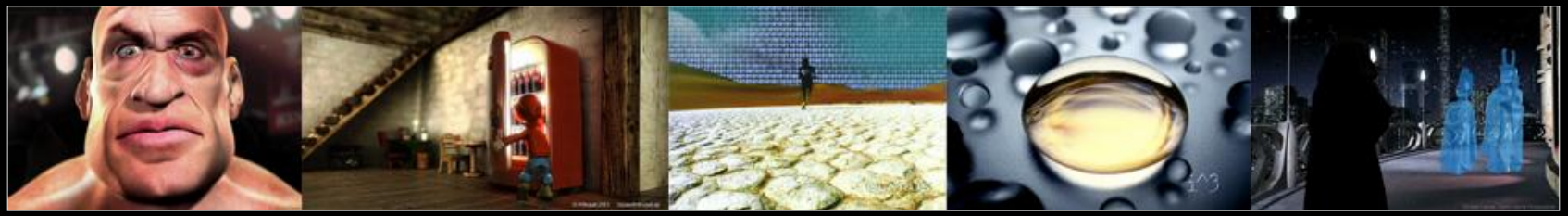


Heavy Data

During evaluation, copies of heavy data exist only at attributes that are cached



Data Access



Autodesk®

Data Access

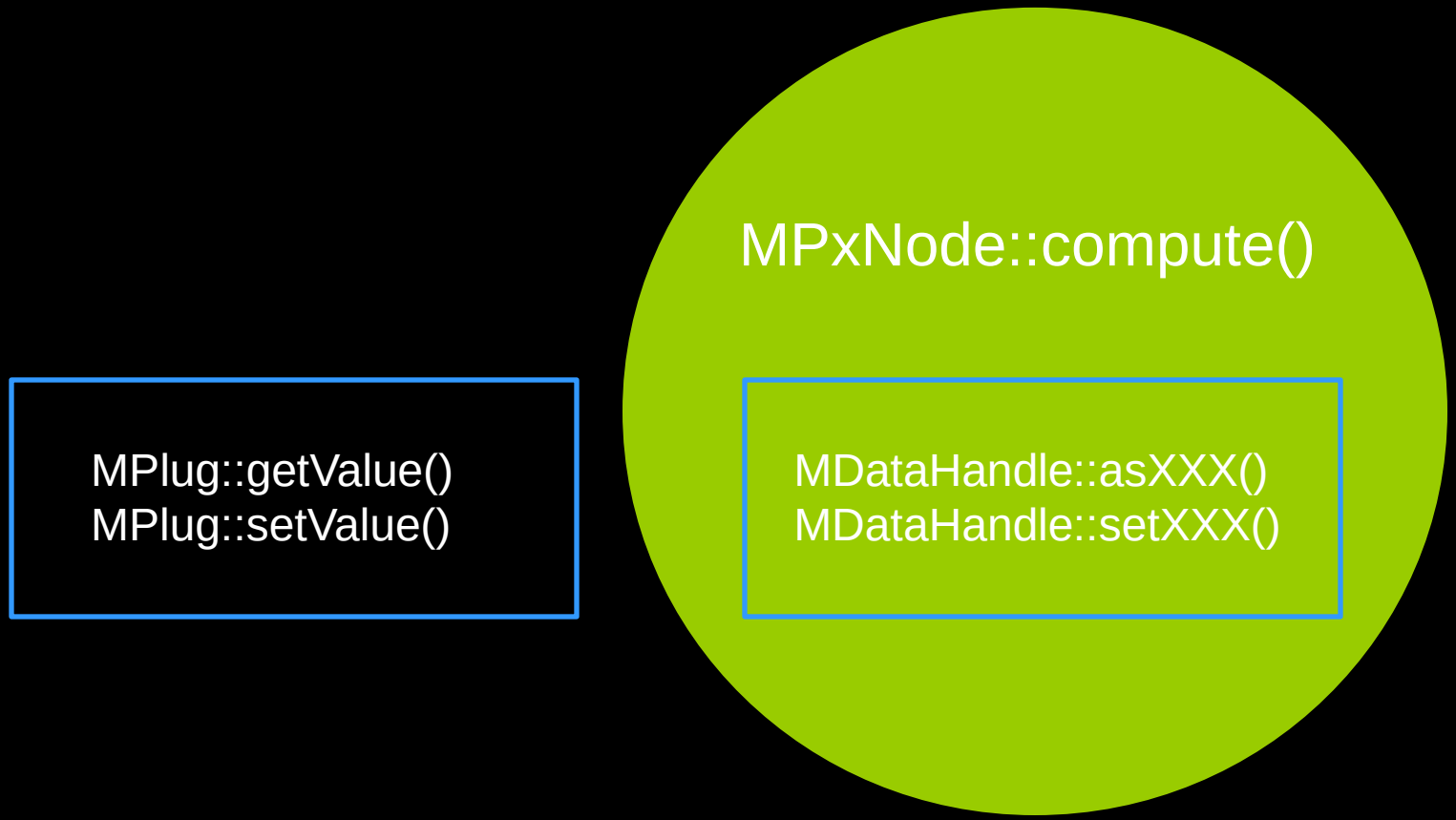
Plug

- query a value
- set a value
- …

DataBlock & DataHandle

main functionality is to query/set a value on a node

Data Access



Data Access

Plugs vs. Datahandles

General rule:

- dataHandles: set/get data during compute
- plugs: set/get data outside of compute

Difference:

- datahandle set/get methods are more efficient
- setting data via plug propagates dirty, datahandle does not

Summary of API DG Classes

Nodes : MPxNode and derived classes

Plugs : MPlug

Attributes : MFnAttribute

Datablocks : MDataBlock

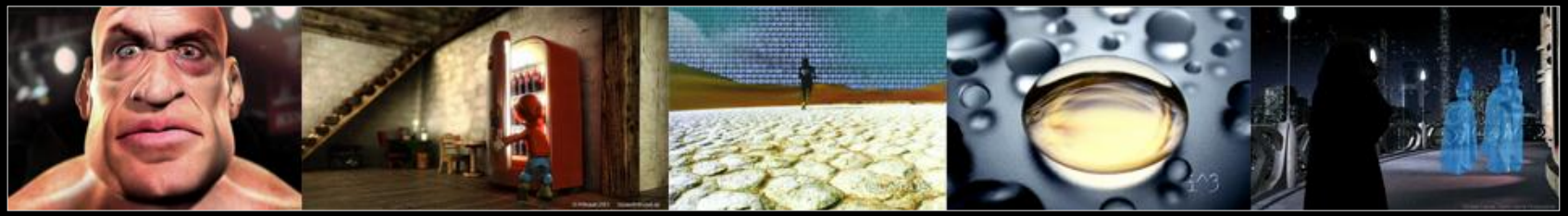
Eval Contexts : MDGContext

Data handles : MDataHandle, MArrayDataHandle, MArrayDataBuilder

Data : MFnData

Connections : no API access, use MPlug methods

Workshop Session



Autodesk®

Example: retrieveWeight

retrieveWeight : In this example, we will create a custom command “retrieveWeight”, it searches attribute “weight” on blendShape node and since it is a multi attribute, it prints out the number of elements in this array attribute and traverse the array to print out plug data on every element.

Autodesk