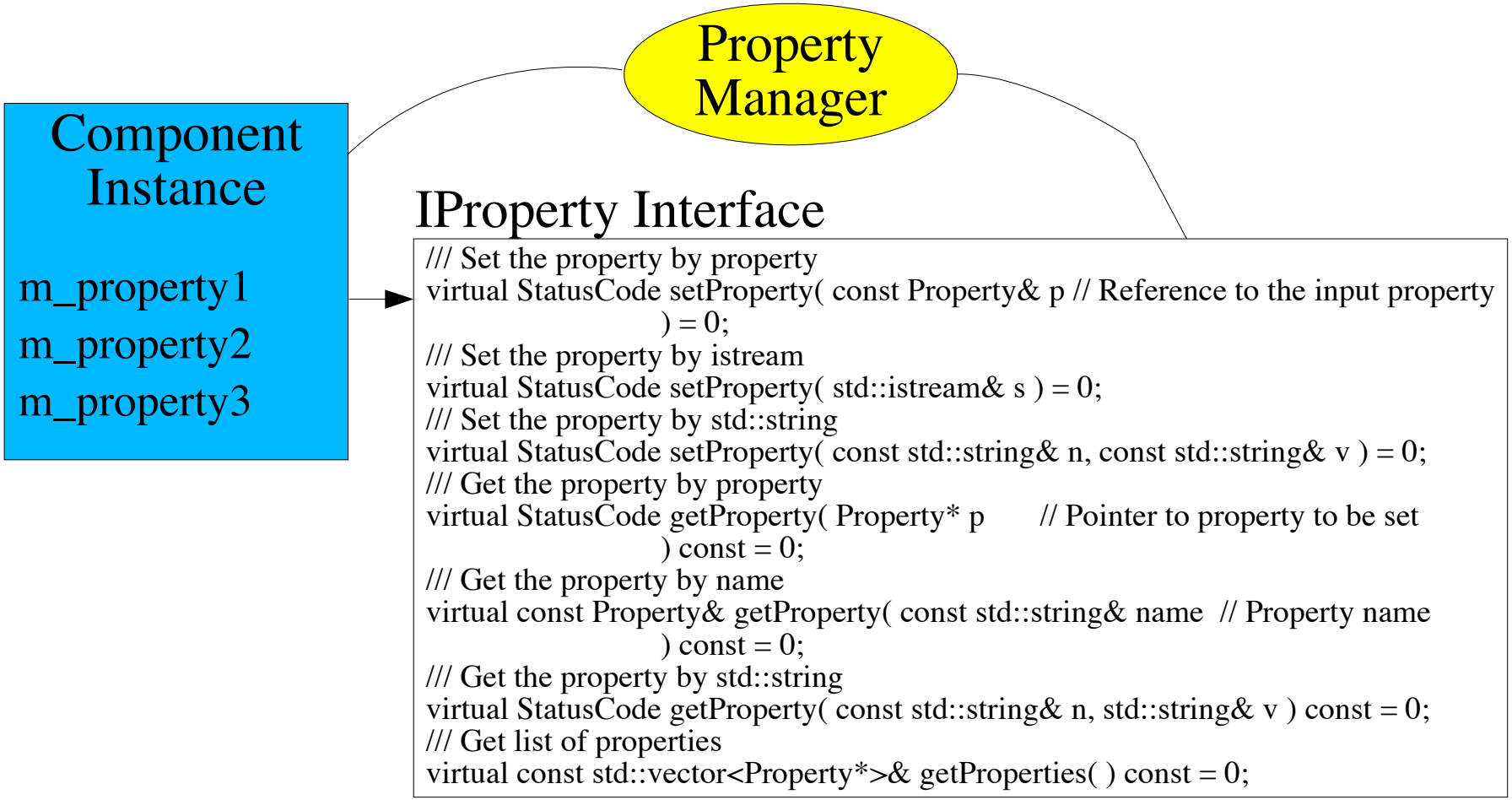
National Energy Research Scientific Computing Center (NERSC)

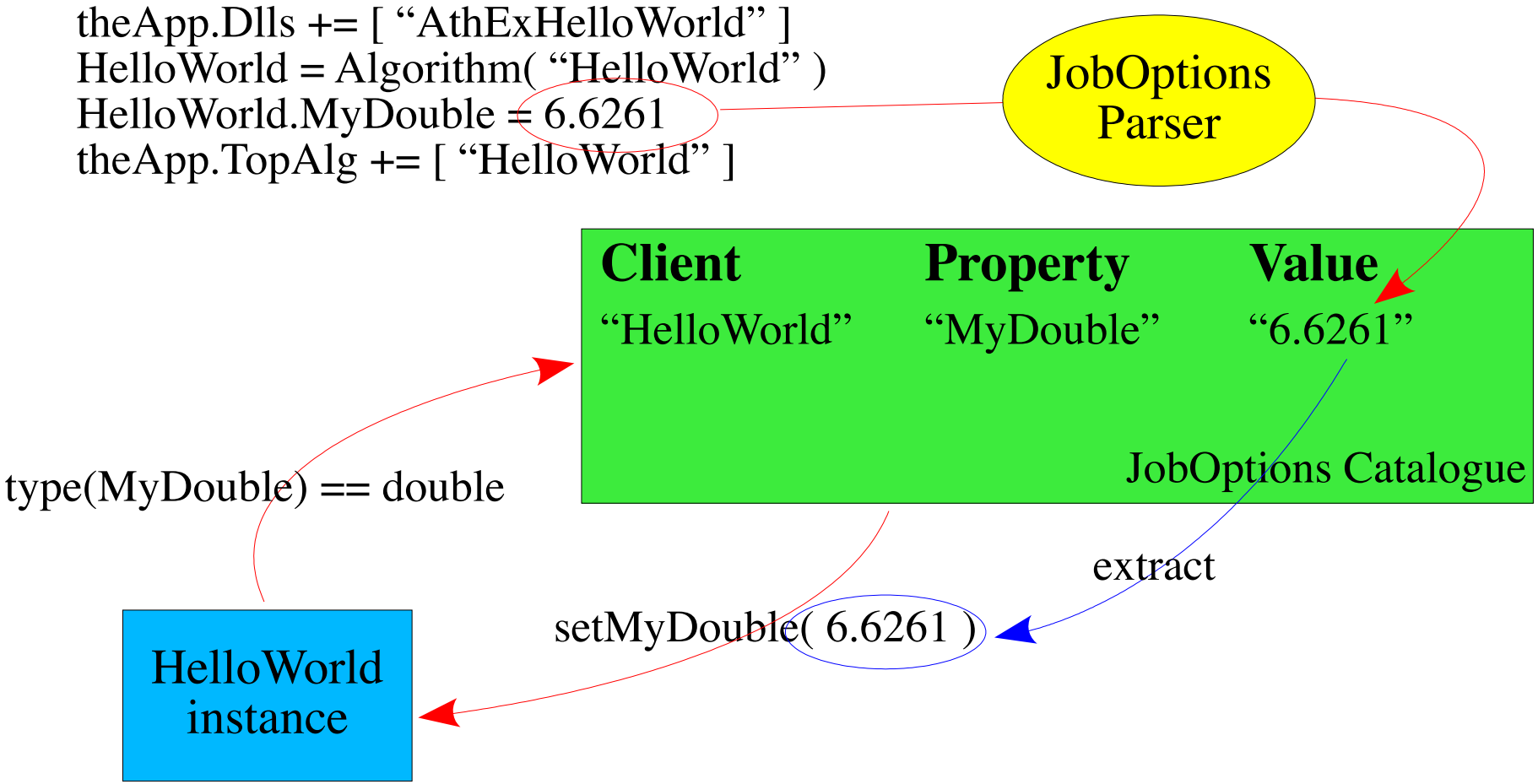
Improving Job Configuration

Wim T.L.P. Lavrijsen
NERSC HENPC, LBNL
HENPC Group Meeting - 10/21/05

Gaudi Configuration Model



Today (Simplified)



Many points of failure.

Situation Today

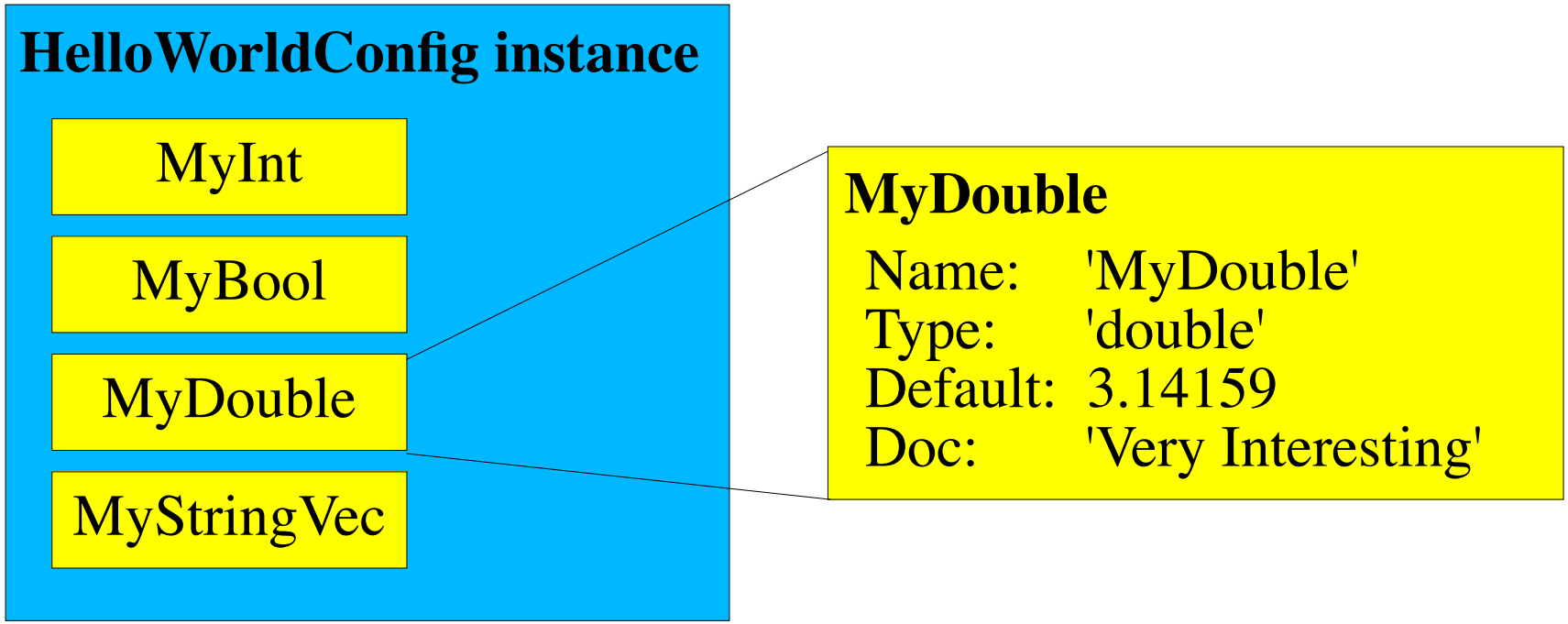
- **Thin layer on top of Gaudi facilities**
  - This is a good thing per se
    - Least restrictive / great for interactivity
    - Automatable: easy maintenance
- **No knowledge availabe during config**
  - String based: *anything* goes
  - Knowledge could be put in by hand
    - Several (often disparate) efforts underway
    - Somewhat hard to maintain
- **Acknowledged we can do better**

Progress - 1

- **Provide more info at configuration time:**
  - List of properties with defaults and doc
    - Allows browsing and understanding before use
    - Solves multiple-instance problem
    - Allowed values, ranges, etc. possible
  - Specification of needed DIIs and Svcs
    - Reduces pbs/overhead due to dependencies
    - No need to touch the ApplicationMgr
  - Conventions for end-user tweaks
    - Reduces number of C&P jobOptions fragments

Tomorrow (simplified)

```
from AthExHelloWorld.AthExHelloWorldConfig import HelloWorldConfig
HelloWorld = HelloWorldConfig( "HelloWorld" )
HelloWorld.MyDouble = 6.6261      # <- can be immediately verified
```



"Single" point of failure!

Automation

- **MyPackage Config.py generated with:**

  $ genconf MyPackage

  – Assumes components in 'libMyPackage.so'
  – Works for Algorithm, Service, & AlgTool

- **Build-time Config.py generation:**

```
$ cat MyPackage/cmt/requirements
[...]
apply_pattern genconf_run
[...]
```

“genconf” Example

```
class HelloWorld( ConfigurableAlgorithm ):
   __slots__ = { 'MyInt' : 0, 'MyBool' : False, 'MyDouble' : 0.,
              'MyStringVec' : [] }

  def getServices( self ):
    return [ 'MessageSvc' ]

  def getDlls( self ):
    return 'AthExHelloWorld'

  def setDefaults( self, hello ):
    hello.MyInt = 666
    hello.MyBool = 1
    hello.MyDouble = 3.14159
    hello.MyStringVec = [ 'Welcome', 'to', 'Athena', 'Framework', 'Tutorial' ]

  def gaudiType ( self ):
    return 'Algorithm'

  def configureMessageSvc( self, msgsvc ):
    msgsvc.OutputLevel = INFO
```

Specialisation

- Derive your own class from Config.py
  - Implement hook for specialization:

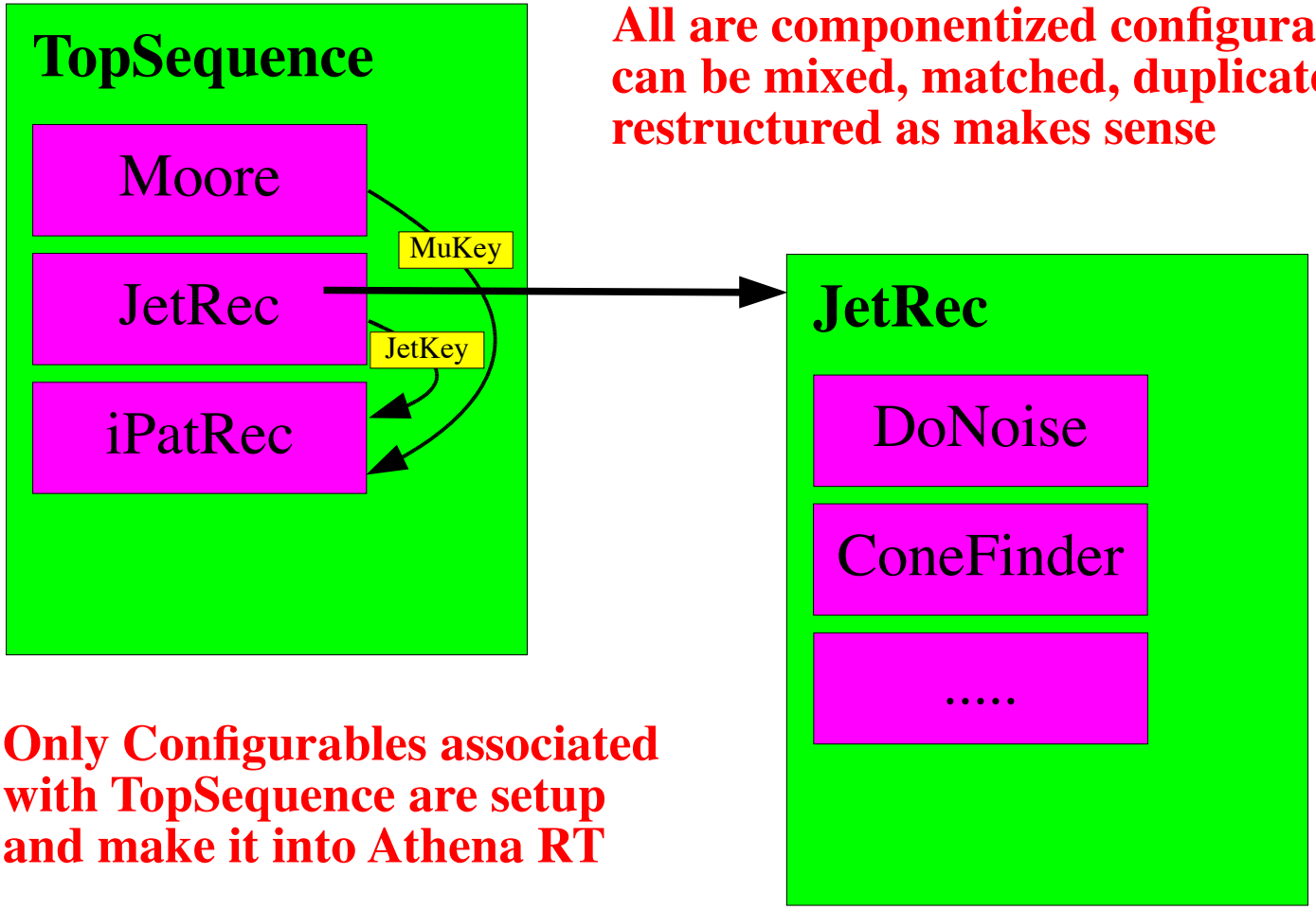
```
class MyClassSpecial( MyClassConfig ):
    def setUserDefaults( self, handle ):
        handle.special = specialValue
```

  - Or add e.g. children (AlgTools) in ctor
  - Use like any other Config.py class
  - C++ style type checks help you get it right
  - Override other methods such as getDlls()

Progress - 2

- **Better organisation of jo-fragments**
  - Config classes derive from *Configurable*
    - Sequential and hierarchical structuring
    - StoreGate key propagation
    - JOT-style tracing of property setting
  - Automatic mapping of Config classes?
    - And converters, tools, etc.
  - Rethinking of current structure?
    - CVS structure (and hence InstallArea) is determined by “history.” Python packaging (e.g. with PyBus) gives new flexibility

Example: Configurables



All are componentized configurables and can be mixed, matched, duplicated, and restructured as makes sense

Only Configurables associated with TopSequence are setup and make it into Athena RT

Example: AthExHelloWorld

```
def buildHello( tag ):
    job = AlgSequence( tag )
    job += HelloWorld()
    return job

# private configuration
myJob = buildHello( "my" )
myJob.HelloWorld = HelloWorld( "myHello" )

# additional, private options
myJob.myHello.MyInt = 42   # note: alg, not type name

# normal processing ... theApp.run(); theApp.exit()
```

Example: JetRec (still under development)

```
   Cone4Jets = JetAlgorithm(“Cone4Jets”)
# configure Cone4Jets private tools
   FinalEtCut = JetSignalSelectorTool("FinalEtCut")
   FinalEtCut.UseTransverseEnergy = True
   FinalEtCut.MinimumSignal       = 10.*GeV
# add them to Cone4Jets
   Cone4Jets += [ DoNoise, InitialEtCut, ConeFinder,
                  SplitMerge, CellCalibrator, FinalEtCut ]
   Cone4Jets.AlgTools = Cone4Jets.sequence()
# and finally...
   topSeq += Cone4Jets
```

- replaces Cone4TowerJet_jobOptions.py
- 10 such jo files in JetRec
  - capture common features, avoid c&p, provide structure

Resources

- **Ongoing documentation:**
  - https://uimon.cern.ch/twiki/bin/view/Atlas/HighLevelJobConfiguration
  - https://uimon.cern.ch/twiki/bin/view/Atlas/PrototypeDataDrivenConfig#Prototype_with_Configurable