Accessing Conditions Data in AthenaMT

Charles Leggett

June 6 2016

Glasgow TIM

Conditions: The Problem

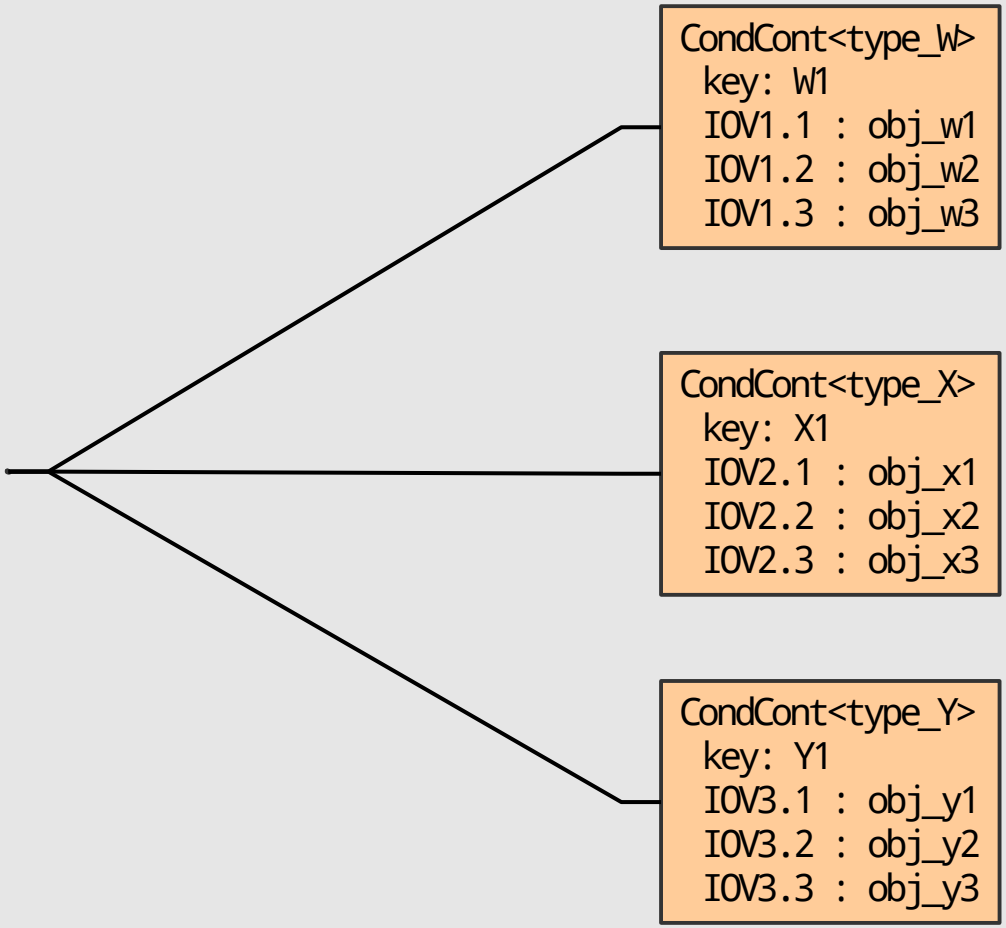
- Conditions objects fall into 2 categories
  - "**raw**" : what's read from the DB
  - "**calibrated**" : what's created after a IOVSvc callback function is executed
    - usually stored in as an AlgTool's data member, and accessed via Tool interface
    - Managed by IOVSvc, which updates/resets Handles as IOV boundaries are reached, and also triggers callback function
- The current conditions access pattern **fails** when several events are executed simultaneously
  - events can straddle IOV boundaries - which one is used?
  - callback functions are not thread safe: data often cached locally
- We need a way to:
  - manage multiple simultaneous copies of conditions data for different IOVs
  - make sure all callback functions are thread safe, and update their information for the correct event

Conditions Data Structure

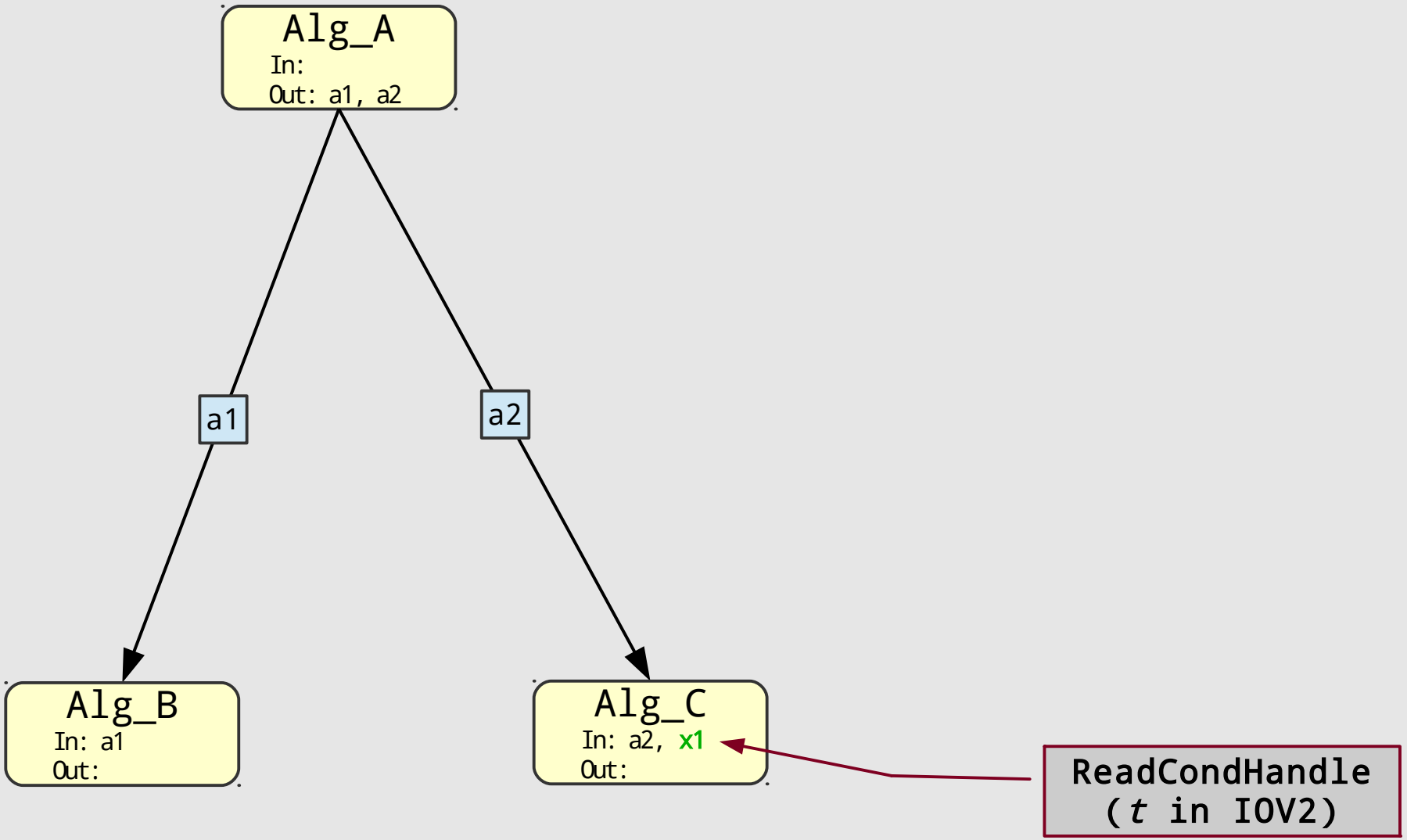
- Conditions objects moved to a ConditionStore
  - instance of StoreGate
  - accessed via ConditionsHandle
- Several instances of any Conditions data object will be kept in the ConditionStore, indexed by IOV
  - ConditionStore objects are actually **containers**
  - will need a garbage collection facility, to ensure that the ConditionStore does not grow too large
- Callback functions replaced with Algorithms, scheduled by framework
  - Conditions Algs declare their data dependencies like any other Alg
    - Inputs, if they depend on data in the EventStore, or other CS data
    - Outputs, for the data they put into the CS
    - done automatically via Handle usage

ConditionStore

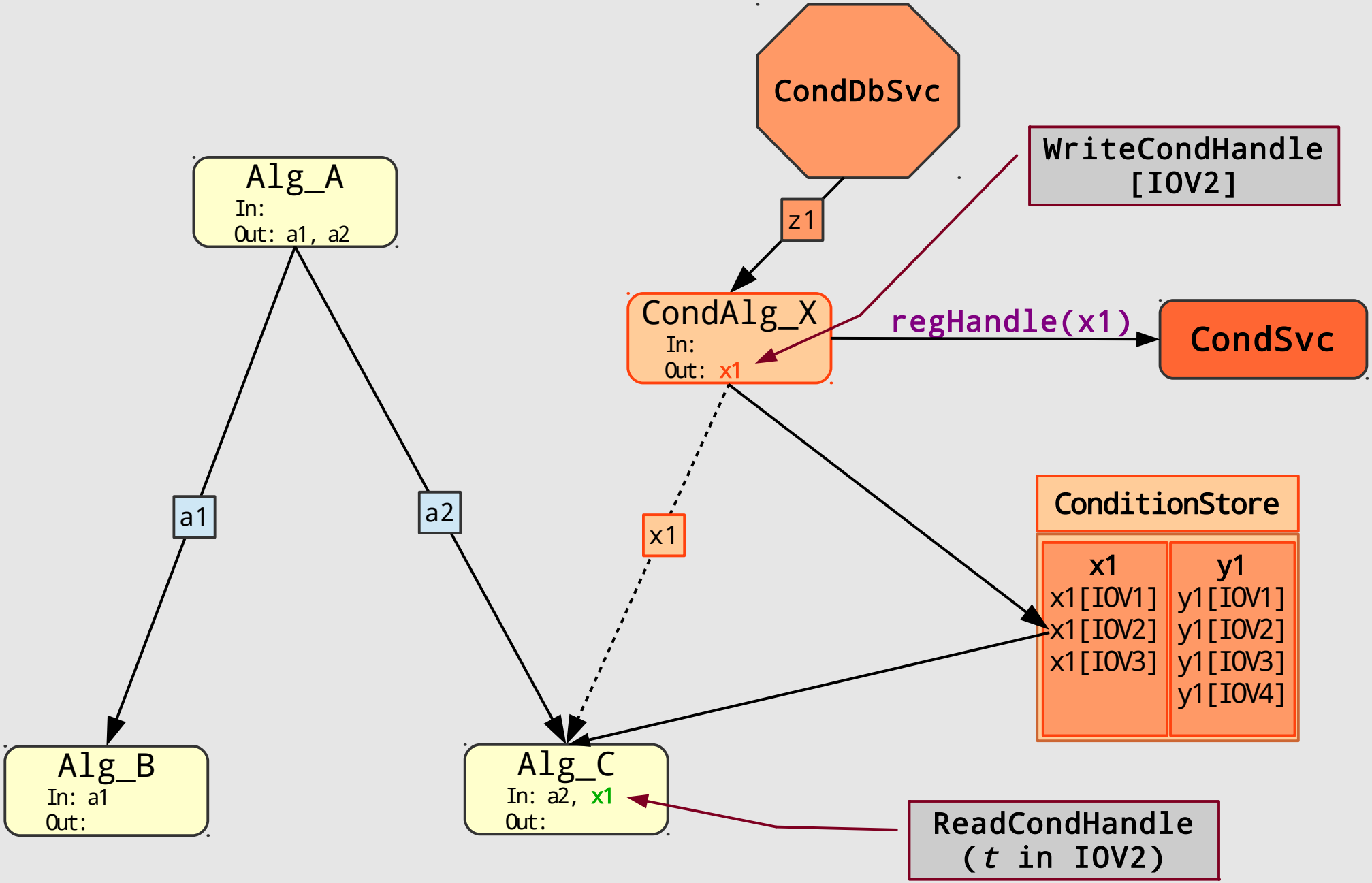
- Store of Containers of Conditions objects
  - one container element per range of validity (IOV)
    - IOV can be time based, or Run/Event or lumi/Event
  - keyed like any object in event store



ConditionHandles



ConditionHandles



Client Use of ReadCondHandle

```
class CaloLocalHadCoeff : public AthAlgorithm {
 private:
   SG::ReadCondHandleKey<CaloLocalHadCoeff> m_rchk
}

CaloHadCoefTestAlg::CaloHadCoefTestAlg(const std::string& name, ISvcLocator* pSL) :
   ::AthAlgorithm( name, pL ) , m_rchk("EMFracClassify") {
    declareProperty("KEY_RCH",m_rchk);
}

StatusCode CaloHadCoefTestAlg::initialize() {
   if(m_rchk.initialize().isFailure()) {
     ATH_MSG_ERROR("unable to initialize ReadCondHandleKey " << m_rchk.fullKey() );
     return StatusCode::FAILURE;
   }
   return  StatusCode::SUCCESS;
}

StatusCode CaloHadCoefTestAlg::execute() {
   SG::ReadCondHandle<CaloLocalHadCoeff>  rch(m_rchk);
   const CaloLocalHadCoeff* cdo{nullptr};
   cdo = *rch;
   if (cdo != nullptr) {
     ATH_MSG_INFO("had coeff val: " << cdo->val() );
   }
}
```

created using current time from Gaudi::Hive::currentContext(), or explicitly with extra parameter

can also do explicit retrieve(EventIDBase) to get value at different time

Functionality

- During initialize, CondAlgs register their WriteCondHandles with the CondSvc

```
StatusCode ICondSvc::regHandle(IAlgorithm* alg,
                               const Gaudi::DataHandle& id,
                               const std::string& dBkey);
```

- At the start of each event, the ForwardScheduler will:
  - query CondSvc to determine which CondObjIDs are valid/invalid
  - query ExecutionFlowGraph to find producer CondAlg of these objects
    - we could build this locally once since it's fixed, but the EFG is pretty efficient
  - if any objects produced by a CondAlg is invalid, schedule the Alg to execute, otherwise mark it as already executed
  - update data catalog with all valid CondObjIDs

- Only CondAlgs that produce new data (ie, the CondObj has entered a new validity range) will execute

Conditions Data With No Callback

- Significant fraction of Conditions data has no associated callback function ("raw")
  - big overhead if we have to create a new CondAlg for each one!
  - want to just read them in, provide WriteCondHandles for them (to satisfy downstream data dependencies), update the handle when it gets into a new validity range

- generic alg **IOVSvc/CondInputLoader**
  - supply with list of db items (folders) to be loaded, just like with the IOVDbSvc

```
from IOVSvc.IOVSvcConf import CondInputLoader
topSequence += CondInputLoader( "CondInputLoader" )

topSequence.CondInputLoader.load += [
    ('AthenaAttributeList', '/path/to/DB/folder1'),
    ('AthenaAttributeList', '/path/to/DB/folder2'),
    ('CaloLocalHadCoeff', '/CALO/HadCalib/CaloEmFrac') ]
```

Database Issues

- IOVDb folder name and StoreGate key can be different. Which to specify in CondHandle configuration?
  - once the data has been read in from the db, the folder name is no longer needed, but clients need the SGkey to access the data
  - SGkey is actually encoded into the db
  - specify the folder name in the configuration, set the appropriate the SGKey once the db is read

CondInputLoader Functionality

```
CondInputLoader::initialize() {
  // do translation of FolderName to SGKey
  …
  // set the Write dependencies via linking Property "Load" to ExtraOutputDeps()
  …

  // register Write DataHandles with CondSvc
  for (auto &e : m_load) {
    if (e.key() == "") {
      sc = StatusCode::FAILURE;
      ATH_MSG_ERROR("   ERROR: empty key is not allowed!);
    } else {
      Gaudi::DataHandle dh(e, Gaudi::DataHandle::Writer, this);
      if (m_condSvc->regHandle(this, dh, e.key()).isFailure()) {
        ATH_MSG_ERROR("Unable to register WriteCondHandle " << dh.fullKey());
        sc = StatusCode::FAILURE;
      }
      // remove proxy reset control from old IOVSvc
      m_IOVSvc->ignoreProxy(dh.fullKey().clid(), e.key());
    }
  }
}
```

CondInputLoader Functionality

```
CondInputLoader::execute() {
    for (auto &obj: m_load) {

    CondContBase* ccb(0);
    if (! m_condStore->retrieve(ccb, obj.key()).isSuccess()) {
      ATH_MSG_ERROR( "unable to get CondContBase* for " << obj
                     << " from ConditionStore" );
      continue;
    }

    if (! ccb->valid(now)) {
       if (m_IOVSvc->createCondObj( ccb, obj, now ).isFailure()) {
          std::string dbKey = m_folderKeyMap[obj.key()];
          ATH_MSG_ERROR("unable to create Cond object for " << obj << " dbKey: "
                        << dbKey);
          return StatusCode::FAILURE;
    } else {
       ATH_MSG_INFO( "  CondObj " << obj << " is still valid at " << now );
    }
    evtStore()->addedNewTransObject(obj.clid(), obj.key());
}
```

IOVSvc::createCondObj

```
IOVSvc::createCondObj(CondContBase* ccb, const DataObjID& id,
                      const EventIDBase& now) {
   if (getRangeFromDB(id.clid(), id.key(), t_now, range, tag, ioa).isFailure()) {
    ATH_MSG_ERROR( "unable to get range from db for "
                   << id.clid() << " " << id.key() );
    return StatusCode::FAILURE;
  }

  DataProxy *dp = ccb->proxy();
  DataObject* dobj(0);
  void* v(0);
  if (dp->loader()->createObj(ioa, dobj).isFailure()) {
    ATH_MSG_ERROR(" could not create a new DataObject ");
    return StatusCode::FAILURE;
  } else {
    v = SG::Storable_cast(dobj, id.clid());
  }
  EventIDRange r2(EventIDBase(range.start().run(), range.start().event()),
                  EventIDBase(range.stop().run(),  range.stop().event()));

  if (!ccb->insert( r2, v)) {
    ATH_MSG_ERROR("unable to insert Object at " << v << " into CondCont "
                  << ccb->id() << " for range " << r2 );
    return StatusCode::FAILURE;
  }
  return StatusCode::SUCCESS;
}
```

What's Left To Do

- Clean up and merge CondSvc into IOVSvc
  - CondSvc has a bunch of unnecessary routines to manage an ASCII test database
  - IOVSvc will also need to inherit from ICondSvc, as that lives in GaudiKernel
- Make new scheme with CondAlgs work in serial Athena
- Develop migration plan for clients
- Improve validity checking in Scheduler
  - all the condition information can be attached to the Data and Control nodes in the Event Flow Graph.
  - validity checking and CondAlg scheduling is then done on demand