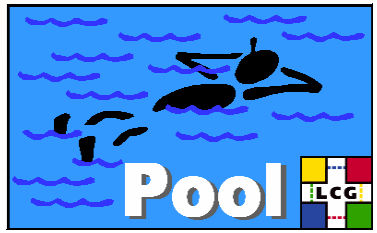
POOL References / Collections in ROOT – an Outlook

- Refs
- Collections
- Solutions and non-solutions
  - The simple
  - the difficult
  - and the hopeless



Goals

- Allow interpreted pool::Ref<T> be interchangeably used with compiled pool::Ref<T>
- Allow access to POOL collections (or a subset) from the interactive root prompt
- Make the good parts of POOL available from ROOT interactively

Current Assumptions

- What the different requirements imposed…

1. There is no checkpoint when the entire object cache can be safely deleted
2. The cache instance objects live in is at user's choice
3. No _*significant*_ ROOT-only developments are done

Fundamentals

- **There are 2 approaches to interactive ROOT**

1. **"Cintify" POOL**
   - Use pool from root prompt like compiled root
   - Need to translate all classes
     - Including all argument types
     - Pool::Ref, pool::DataSvc, …
2. **"Rootify" POOL**
   - Initialize POOL once
   - Then use ROOT classes directly

Root dict's must be generated from LCG dictionaries

“Cintify” POOL

- **Proof was done**
  - Not too elegant, but concept works
  - Interpreted Ref<T> instances are passed to compiled pool code
  - Required separate Ref<T> and cache implementation
- **Some POOL implementation need changes**
- **But this should be feasible**

- **Example:**
  /cvs/POOL/contrib/RootRefs
  /cvs/POOL/pool/config/cmt/RootRefs/cmt/*.C

pool::Ref<T> Interpreted vs.Compiled

- **Currently CINT has problems to deal with pool::Ref<T>**
- **Interpreter (technical, hence no-) problems**
  - Less the references, but rather all the mess one gets with dependent headers are problematic
- **More fundamental problems**
  - pool::Ref<T> uses ***typeid(T)*** operator…
    typeid(T) works in CINT and works in compiled code
    But they are instances of unrelated classes and can only be used in their own environment
  - If to be solved only with workaround!

“Rootify” POOL (1)

- **Basic idea is to allow for analyses like:**

```
gSystem->Load(“pool.dll”)
TClass* c=gROOT->GetClass(“PoolInitialize”);
PoolInitialize initialize();
```

POOL initialization

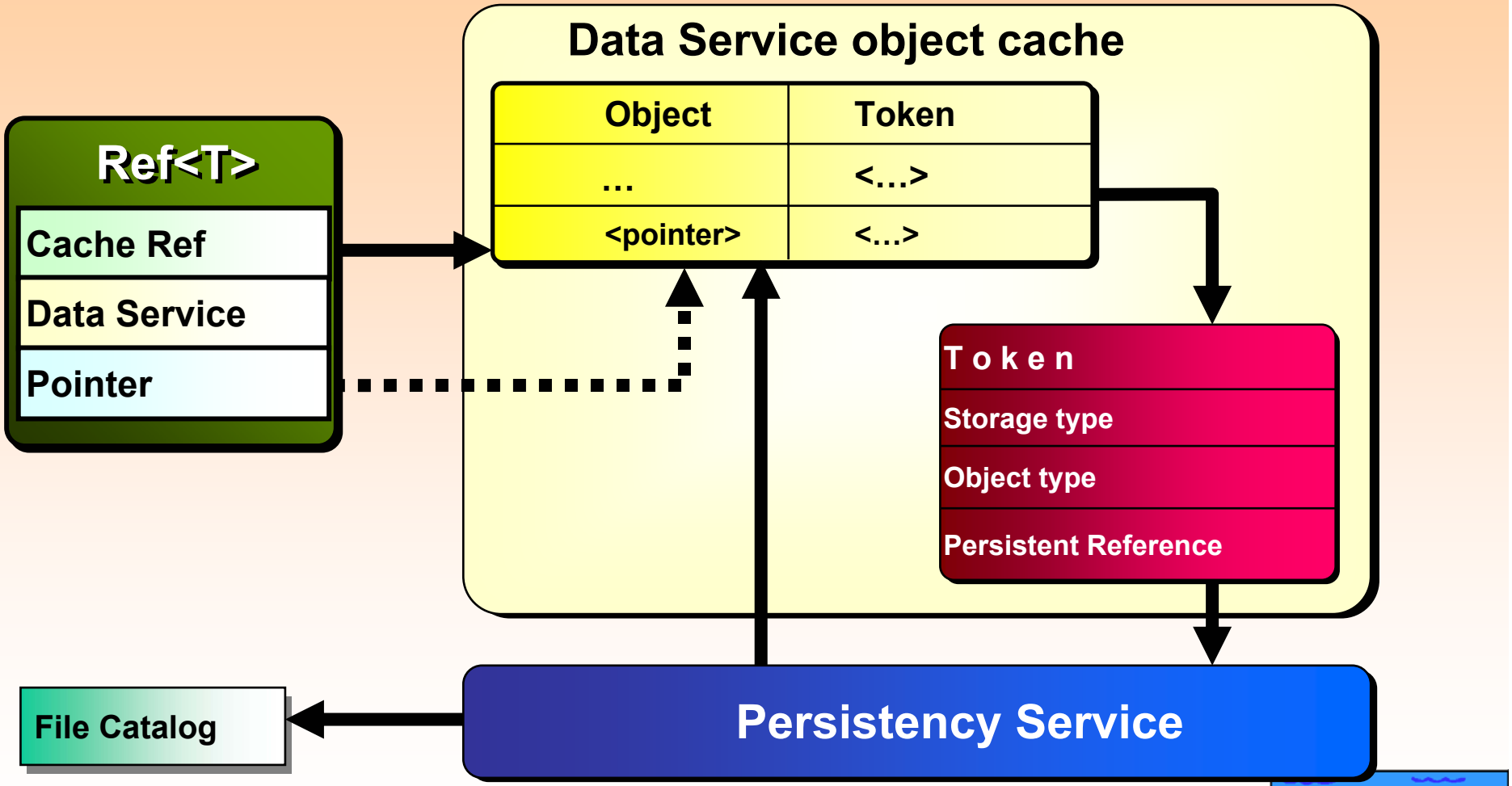
```
TFile* f = new TFile(“my_pool_file.root”);
TTree* t = (TTree*)f->Get(“My_pool_container”);
TBranch* b = f->GetBranch(“My_pool_container”);
MyObj* pObj = new MyObj();
b->SetAddress(&pObj);
For ( int i=0; i<t->GetEvents(); ++i)  {
  int nbytes = b->GetEvent(i);
  if ( nbytes > 0 )  { . . . }
}
```

Standard ROOT

“Rootify” POOL (2)

- **This approach would work …if**
  - MyObj has no aggregated pool::Ref<T>
- **There are several reasons…**
  - But the main reason is the same why C-pointers got replaced by pool::Ref<T>
    Load-on-demand and reference counting of objects
  - pool::Ref<T> does not take ownership
  - Ownership is with the object cache - whatever the cache implementation is
  - Tight collaboration between cache and reference

“Rootify” POOL: Basics



“Rootify” POOL (3)

- **For any number of instances of pool::Ref<T> referring to the *same* object**
  - Only one line in the pool cache may be filled
  - Any *aggregated* Ref<T> must at load-time check if it’s cache line is present – if not inject a new cache line
- **The collaboration between pool::Ref<T> and the data cache is broken if objects are not loaded with pool**
  - References will not work
  - Neither will they be written properly
  - Nor can they be de-referenced

POOL Collections

- **Effectively all the arguments made for objects still valid**
- **“Cintified” collections – feasible**
- **“Rootified” collections (???)**

- **Collection “items” can always be read by ROOT if the collection was written by ROOT**
  - It’s an N-tuple !
  - Trees get booked with each primitive as separate branch

POOL Collections

- **But the refs……or how do I get the event ?**
- **Same argument as before:**
  Refs want to collaborate with caches
  they live in and from the POOL framework
  which feeds them and caresses them
- **If there is none – tough luck**

- **However, normally the situation is a bit better**
  - Token typically is stored as a string
  - Object typically can be retrieved, but with workarounds

The Hopeless

- **tree->Draw("pool_ref->dataItem()");**
  - No handles can be installed to prepare pool::Ref<T> (or they are unknown to me)
  - No connection to cache is available

Current Assumptions

1. **There is no checkpoint when the entire object cache can be safely deleted**
   - Now object _*only*_ disappear when all refs are out-of-scope
2. **The cache instance objects live in is at user's choice**
3. **No _*significant*_ ROOT-only developments are done**

? What happened if we give up these ?

Loosening Current Assumptions

1. **There is no checkpoint when the entire object cache can be safely cleared**
   - → hooks in TTreePlayer ?
   - → hooks in TTree/TBranch::Fill() / GetEntry()
2. **The cache instance objects live in is at user's choice**
   - Cache is set atomically at reading time
3. **No _*significant*_ ROOT-only developments are done**
   - Root specific reference validation in streamer function instead of generic handling

Summary

- **It depends what is required…**
- **If "Cintify" POOL is sufficient**
  - In principle straight forward
  - Effort is needed, but the path is straight
- **If "Rootify" POOL & the hopeless is a must**
  - Technology independent pool implementations can only be used at a very limited scale
  - Pool::Ref<T> and an appropriate cache go together
  - Need to install hooks in ROOT in order to ensure reference-cache collaboration
  - Significant development