Discussion input for future ROOT functionality/features.

Peter van Gemmeren, Marcin Nowak
for the ATLAS I/O team

- Interface separation of ROOT I/O
- Dynamic structure creation to minimize ROOT branch count

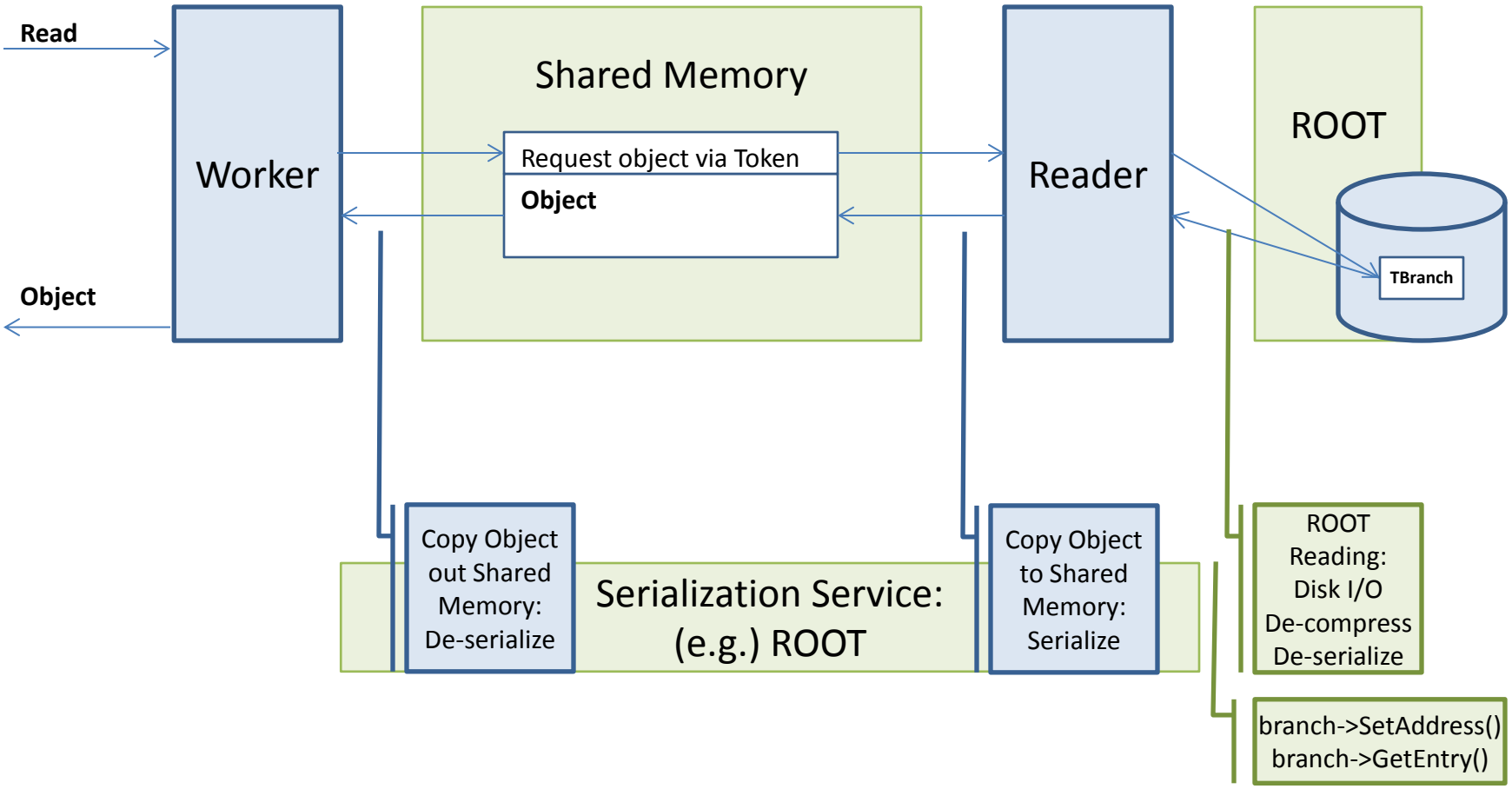
Event Data Streaming

- ATLAS is developing Shared Reader/Writer framework for AthenaMP (and others):
  - Worker nodes will not access ROOT file directly.
    - Avoid having multiple worker read (and decompress) same data (because their events are in the same ROOT cluster).
    - Save memory (e.g. TTreeCache).
  - Has to work with fine granularity dispatching (EventServer).
    - Single events for Simulation

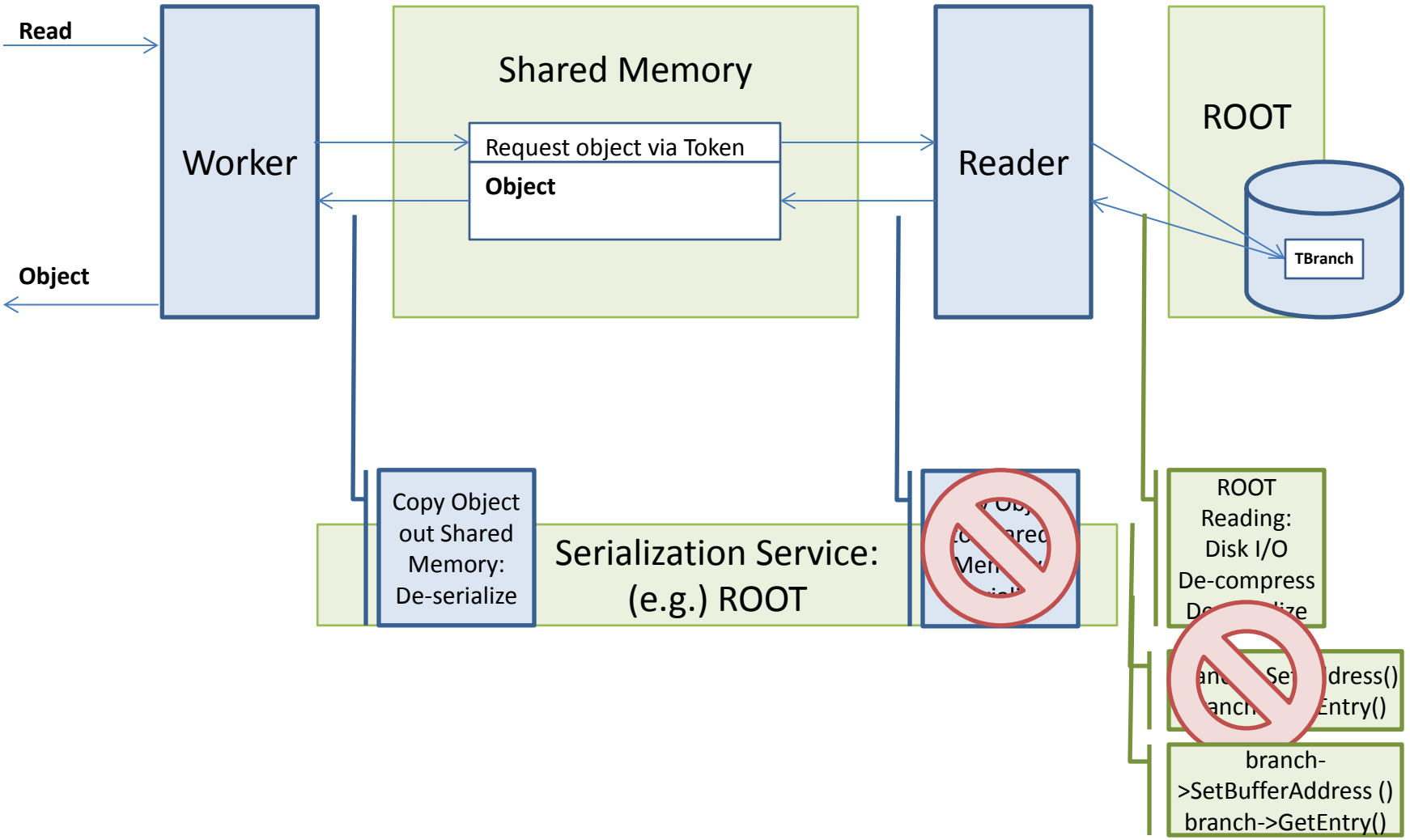
Prototype Implementation

- The (Gaudi-style/ROOT independent) ConversionSvc is configured so that when object is requested from worker's store, it sends a message to a Shared Reader.
- The Shared Reader uses its ConversionSvc to read the object (via POOL/APR and ROOT) and sends it back via shared memory to the worker.

Okay


Read
Object
Worker
Shared Memory
Request object via Token
Object
Reader
ROOT
TBranch
Copy Object out Shared Memory: De-serialize
Serialization Service: (e.g.) ROOT
Copy Object to Shared Memory: Serialize
ROOT Reading: Disk I/O De-compress De-serialize
branch->SetAddress()
branch->GetEntry()

Better

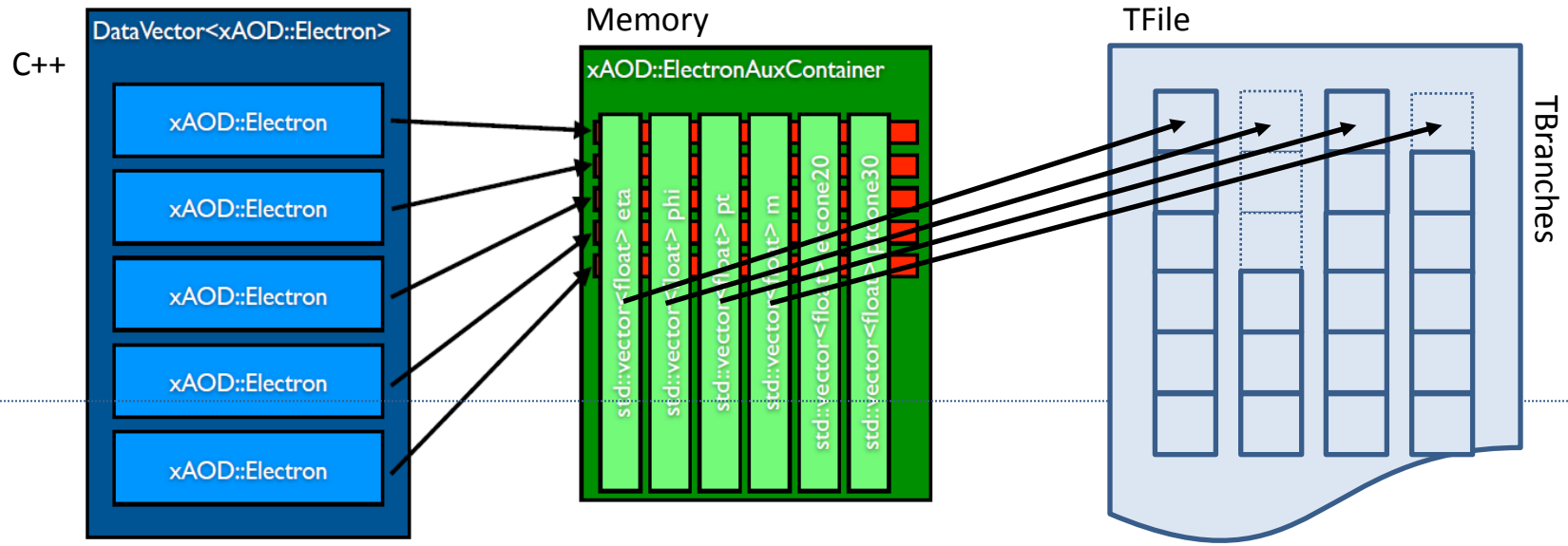

Read
Worker
Object
Shared Memory
Request object via Token
Object
Reader
ROOT
TBranch
Copy Object out Shared Memory: De-serialize
Serialization Service: (e.g.) ROOT
ROOT Reading: Disk I/O De-compress
branch->SetBufferAddress ()
branch->GetEntry()

Please Consider

- We would like the ROOT team to consider:
  - Provide an interface (similar to SetBufferAddress()) set allows client to read Branch entry w/o de-serialization.
    - This should work for all ROOT data
  - Similarly on write, SetBufferAddress() should allow giving serialized data object to ROOT to collect, compress and write.

Run 2 ATLAS EDM - xAOD

- For Run2 ATLAS is using a new EDM based on the concept of dynamic data structure – xAOD
  - An xAOD type can have a fixed set of static attributes and a random set of dynamic attributes
    - Dynamic attributes can be added at any stage of program execution and removed when writing data to file
    - Data payload changes between event processing stages while the class type remains the same



xAOD: TBranches and Memory

- Only dynamic attributes can be dropped between processing stages
  - All attributes that we suspect could be dropped need to be dynamic
  - Every dynamic attribute adds one TBranch
  - ATLAS ESD files can have ~10K branches
  - An open TFile allocates a buffer for each branch
    - Buffer size in memory is multiplied by the compression ration
- We use a lot of memory

Packing Dynamic Attributes

- Combining dynamic attributes into a single structure would significantly reduce the number of branches
  - The exact shape of the structure is known only at runtime, when writing data
    - Cannot prepare a dictionary beforehand
  - ROOT offers 2 ways to define a new class „on the fly”:
    - gSystem->CompileMacro(„filename”)
      - Builds and loads a shared library with the class definition
      - Needs to explicitly include all necessary header files
        - » Attributes can be of any type!
      - (somewhat slow and less elegant solution)
    - gInterpreter->Declare(„C++ class definition”)
      - Fast, knows all types internally
      - But does not produce a TClass with complete reflection information – necessary to use StreamerInfo to query about data members

Using Dynamic Structures

- Declaration
  - Inspecting xAOD dynamic attributes and preparing a string declaration

```
struct xAODtypename_dynamic {
          // xAOD (Aux) storage vector type    attribute name
          std::vector<int>      *int_attr_name = 0;
          std::vector<AnyType>  *anytype_attr_name = 0;  };
```

- Creation
  - With TClass::New()
- Filling
  - A little of void* pointer acrobatics to set all structure members to point to the original xAOD storage vectors
  - Data members' offsets seem to follow standard C structure rules (offset 0, 8, 16, ..)

Dynamic Structures – I/O

- Writing
  - Regular TBranch::SetAddress() and Fill()
  - Works as long as all members of the structure have dictionaries
    - Explicit ROOT error if not (good!)
- Reading
  - ROOT creates the object (GetEntry())
  - Attributes can be inspected using StreamerInfo
    - Data members' offsets are completely different than when writing!
  - No idea how to ask ROOT to read directly into xAOD storage (which we do when working with single branches)
    - Currently using a hack to swap std::vector content

Topics for Discussion

- We are aware we are doing I/O for interpreted types
- Simple tests are working – more testing needed
  - Reading/merging files with different „versions” of the dynamic structures
  - How to best copy/move the vector payload when reading
  - How to handle new dynamic attributes added after the structure was defined