Fermilab | U.S. DEPARTMENT OF ENERGY | Office of Science

art outlook

Kyle J. Knoepfel

Mini-workshop: Event processing software systems

8 July 2018

art's approach

- *art* is intended to be used for analysis jobs and large-scale production jobs.

- Much of our approach is forward-looking. We aim to (e.g.):
  - run *art* on HPC machines
  - deliver cutting-edge software tools (e.g. TensorFlow)
  - enable experiments to benefit from modern language features (*art* supports C++17)

- We have users that prefer to develop on macOS systems
  - We support open-source Clang builds on macOS and Linux

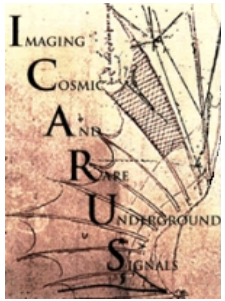
- We actively contribute to the code bases of *art*-using experiments/projects:
  - As of May 2018, the SciSoft team supports and develops code for LArSoft

art's approach

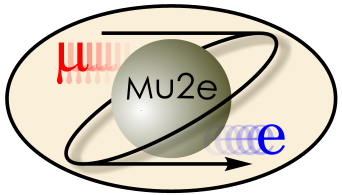
- We support the offline (and some online) processing for 11 projects/experiments.
- Our development efforts are guided by:
  - Current and future needs of *art*-using experiments (represented by stakeholders).
  - Current and future software and hardware technology, in and outside of HEP.
  - Feedback from individual users, and our own estimation of what features would make *art* simpler to use.
- Experiment support:
  - Design guidance and code reviews at the request of experiments
  - Small-scale profiling efforts at the request of experiments
- Dedicated stakeholder meeting each week:
  - Discussion of upcoming changes and issues with stakeholders
  - Sharing among experiments
    - e.g. heist (from g-2) wraps *gallery* for ease of use with Python; usable by all *art* users

Who uses *art*?

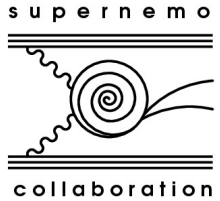
artdaq project



LArIAT experiment



Previous and potential users



art concepts

- Hierarchical data processing ($run \supset subrun \supset event$)
- **Experiments decide** how to define the processing levels (e.g. event)
- All processing elements are plugins, loaded at run-time via user configuration
  - Input source
  - Data-processing modules
  - Output modules
  - Other utilities that facilitate data-processing
- *art* provides various input sources and output modules, but all processing elements can be user-defined
- Workflows are assembled by a configuration file loaded at run-time
  - Adjustments to workflows do not require recompilation of C++ source code

Highlighted features

- Concurrent processing of events within a subrun (as of June 2018)
  - Processing model is inspired by CMSSW's
- Data-product management is thread-, type-, and `const`-safe
- Configuration description and validation suite
- Implicit data-product aggregation for non-event products
- Secondary input (backing) files
- Module time- and memory-tracking facilities
- Graph of data dependencies between modules

Configuration description and validation

- Common problem: how do I configure my program? What if I make a mistake?

Configuration description and validation

- Common problem: how do I configure my program? What if I make a mistake?
- Users can provide C++ documentation/validation structures:

```
class art::Prescaler : public SharedFilter {
public:
  struct Config {
    Atom<size_t> prescaleFactor{
      Name("prescaleFactor"),
      Comment("This module filters one of every n events, where n\n"
              "is the 'prescaleFactor'.")};
    Atom<size_t> prescaleOffset{
      Name("prescaleOffset"),
      Comment("An offset is allowed--i.e. the sequence of events\n"
              "does not have to start with the first event."),
      {}};
  };

  using Parameters = Table<Config>;
  explicit Prescaler(Parameters const&, ProcessingFrame const&);
```

```
Prescaler::Prescaler(Parameters const& config, ProcessingFrame const&)
  : SharedFilter{config}
  , n_{config().prescaleFactor()}
  , offset_{config().prescaleOffset()}
{
  async<InEvent>();
}
```

Configuration description and validation

- Common problem: how do I configure my program? What if I make a mistake?
- Users can provide C++ documentation/validation structures:

```
class art::Prescaler : public SharedFilter {
public:
  struct Config {
    Atom<size_t> prescaleFactor{
      Name("prescaleFactor"),
      Comment("This module filters one of every n events, where n\n"
              "is the 'prescaleFactor'.")};
    Atom<size_t> prescaleOffset{
      Name("prescaleOffset"),
      Comment("An offset is allowed--i.e. the sequence of events\n"
              "does not have to start with the first event."),
      {}};
  };

  using Parameters = Table<Config>;
  explicit Prescaler(Parameters const&, ProcessingFrame const&);
```

```
Prescaler::Prescaler(Parameters const& config, ProcessingFrame const&)
  : SharedFilter{config}
  , n_{config().prescaleFactor()}
  , offset_{config().prescaleOffset()}
{
  async<InEvent>();
}
```

```
module_type : Prescaler (or "art/Framework/Modules/Prescaler")

    provider: art
    type    : filter
    source  : /home/knoepfel/art/art/Framework/Modules/Prescaler
    library : /home/knoepfel/scratch/builds/debug/art/lib/libart

Allowed configuration
---------------------

    ## Any parameters prefaced with '#' are optional.

    <module_label>: {

       module_type: Prescaler

       errorOnFailureToPut: true  # default

       ## This module filters one of every n events, where n
       ## is the 'prescaleFactor'.

       prescaleFactor: <unsigned long>

       ## An offset is allowed--i.e. the sequence of events
       ## does not have to start with the first event.

       prescaleOffset: 0  # default
    }
```

Configuration description and validation

- Common problem: how do I configure my program? What if I make a mistake?
- Users can provide C++ documentation/validation structures:

```
class art::Prescaler : public SharedFilter {
public:
  struct Config {
    Atom<size_t> prescaleFactor{
      Name("prescaleFactor"),
      Comment("This module filters o
               "is the 'prescaleFacto
    Atom<size_t> prescaleOffset{
      Name("prescaleOffset"),
      Comment("An offset is allowed-
               "does not have to star
      {}};
  };

  using Parameters = Table<Config>;
  explicit Prescaler(Parameters cons
```

```
Prescaler::Prescaler(Parameters cons
  : SharedFilter{config}
  , n_{config().prescaleFactor()}
  , offset_{config().prescaleOffset(
{
  async<InEvent>();
}
```

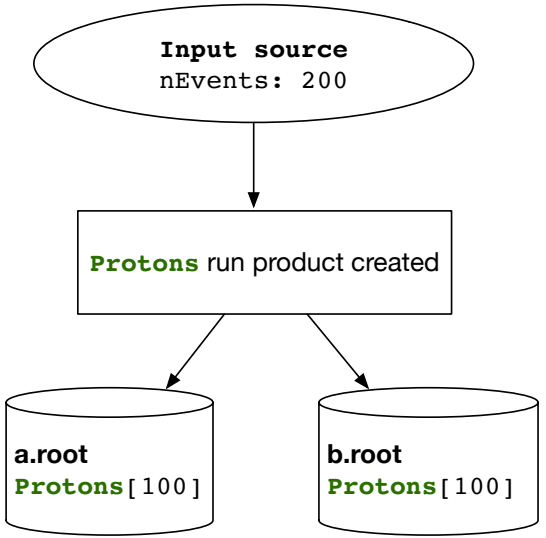
```
module_type : Prescaler (or "art/Framework/Modules/Prescaler")

    provider: art
    type    : filter
    source  : /home/knoepfel/art/art/Framework/Modules/Prescaler
              /knoepfel/scratch/builds/debug/art/lib/libart

                                                          on
                                                          --

                                                          rs prefaced with '#' are optional.

                                                           {

                                                           Prescaler

                                                          reToPut: true  # default

                                                          le filters one of every n events, where n
                                                          rescaleFactor'.

                                                          or: <unsigned long>

                                                          is allowed--i.e. the sequence of events
                                                          have to start with the first event.

        prescaleOffset: 0  # default
    }
```

```
================================================================

!! The following modules have been misconfigured: !!

----------------------------------------------------------------


Module label: prescaleEvents
module_type : Prescaler

Any parameters prefaced with '#' are optional.
Unsupported parameters:

 + prescaleoffset                  [ ./test.fcl:6 ]


================================================================
```


Fermilab

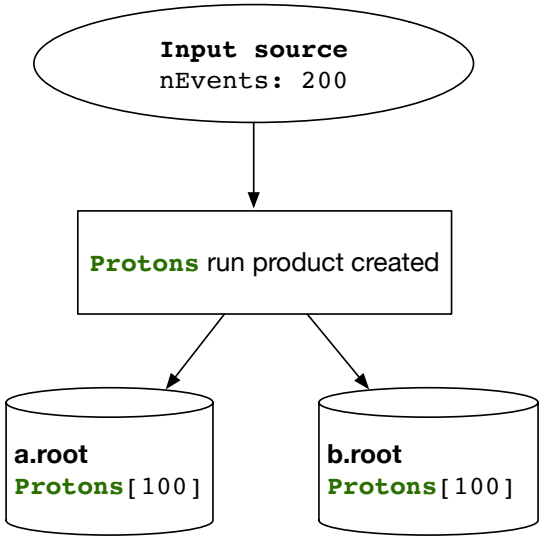
Implicit data-product aggregation

- Users can configure output modules to rollover to a new output file when a condition is met (max. number of events, file size, time open, etc.).
- (Sub)run products can be spread across multiple files
- Whenever the files are concatenated together, *art* can combine the products according to an aggregation behavior (e.g.):
  - Count of protons-on-target are summed
  - Map of particle species are combined via insert
  - Any user-defined aggregation function



Implicit data-product aggregation

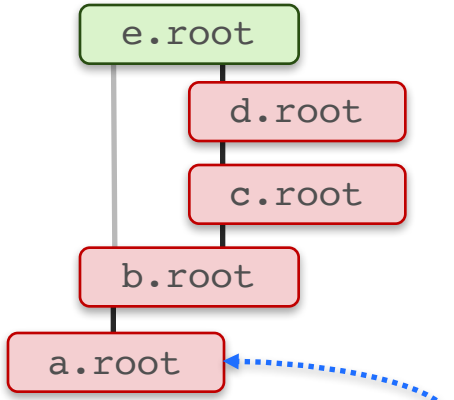
- Users can configure output modules to rollover to a new output file when a condition is met (max. number of events, file size, time open, etc.).
- (Sub)run products can be spread across multiple files
- Whenever the files are concatenated together, *art* can combine the products according to an aggregation behavior (e.g.):
  - Count of protons-on-target are summed
  - Map of particle species are combined via insert
  - Any user-defined aggregation function



- *art* infrastructure necessary for this make some multi-threading issues easier:
  - Distinction between a (e.g.) full run vs. a run fragment
  - Set of events corresponding to a given product (to avoid double counting)

Secondary input files

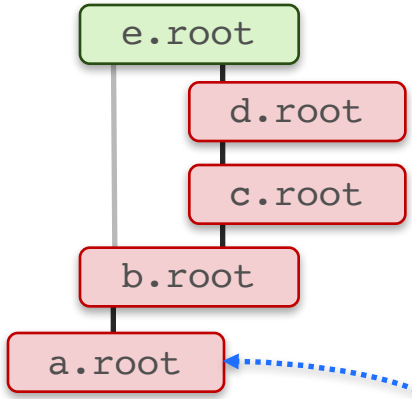
- *art* has a backing-store system, where a hierarchy of input files can be traversed to find the requested product.




```
auto const p = event.getValidHandle<string>(tag);
```

Secondary input files

- *art* has a backing-store system, where a hierarchy of input files can be traversed to find the requested product.




```
auto const p = event.getValidHandle<string>(tag);
```

- I do not know if this feature will survive.
  - Some uncomfortable multi-threading problems to handle.
  - Fermilab's file-delivery system for large-scale production is not filename-based.

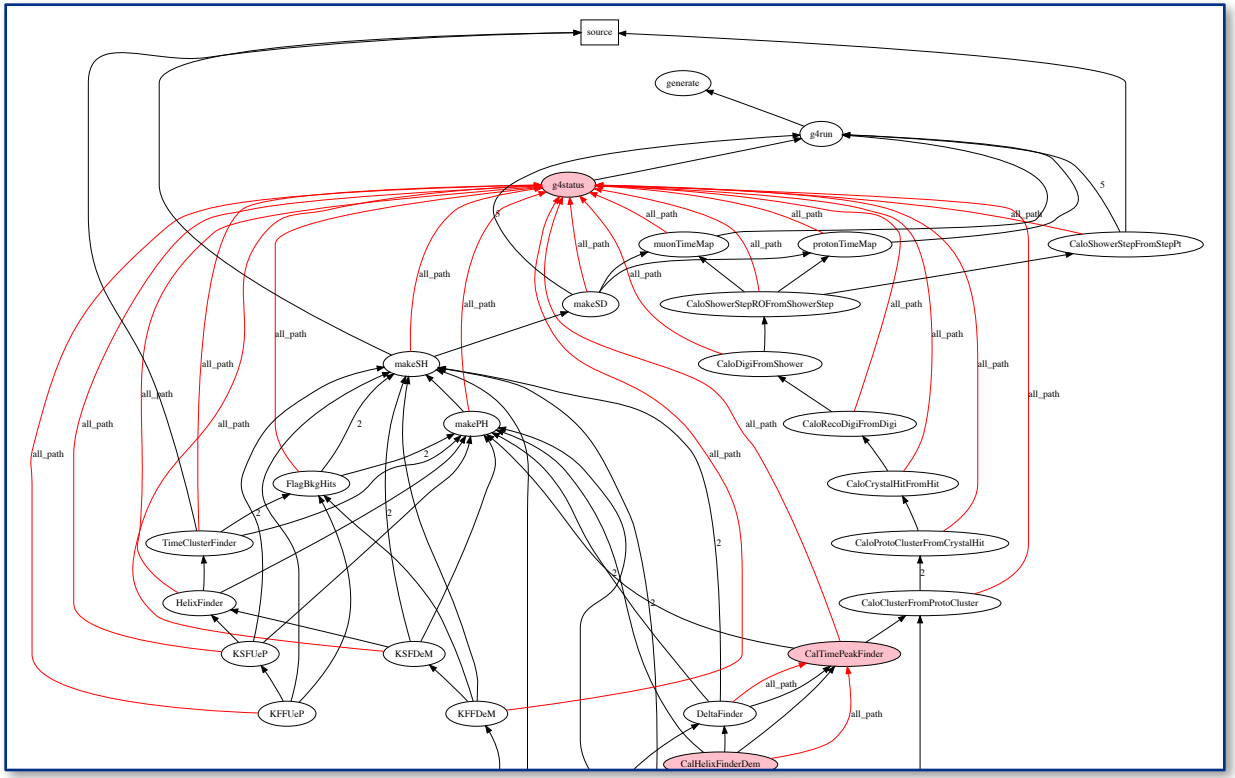
Time- and memory-tracking utilities

- *art* provides simple utilities for profiling user code
  - `TimeTracker`
  - `MemoryTracker`
- These are run-time enabled services used to isolate problematic modules
  - They do not replace full-fledged profiling tools, but they are often sufficient to point users in the right direction without the overhead of profiling tools.
- Output can be printed to terminal or stored in SQLite database
  - We have scripts/utilities to interpret output

```
TimeTracker printout (sec)                          Min          Avg          Max          Median       RMS          nEvts
Full event                                          0.000662971  0.000767011  0.00983449   0.000708468  0.000314318  1000
source:EmptyEvent(read)                             0.000307314  0.000360045  0.00705727   0.00033096   0.000216672  1000
tp:prescaleEvents:Prescaler                         1.4132e-05   1.64184e-05  6.7205e-05   1.5068e-05   2.99402e-06  1000
[art]:TriggerResults:TriggerResultInserter          0.000154377  0.000176913  0.00158631   0.000165199  4.91682e-05  1000
```

Data-dependency graph

- *art* assembles the graph of data dependencies between modules.
- Data-dependency errors are caught at job start-up time, just after module constructors have been called.
- We do not yet use the graph to optimize event processing.
  - We intend to do so.



Fermilab

Near-term/imminent plans (2018-2019)

- Help experiments upgrade to *art* 3 to realize multi-threading benefits.
- Move toward non-event level parallelism.
- Continue to prune library dependencies from core framework infrastructure:
  - ROOT will soon be unnecessary for core framework usage
  - Various Boost dependencies can be removed when C++17 is adopted
- Continue research with HDF5 file format:
  - HDF is the primary data-storage formats on HPC machines
  - Work done in collaboration with the HDF group over the last couple years
  - HDF5 analysis-format alternative for ROOT TTree (Handing over to NOvA functioning software they want to use now — July 9)
    - 42 TB data read/decompressed in under 20 seconds on Cori (1200 KNL nodes) in Python.

Middle-term R&D (2020-2022)

- Working toward the ability to use HDF as an alternative I/O system in *art*

- Extend *art* concepts to better support intra-module parallelism
  - Experiments like DUNE may not be able to afford multiple events in flight at the same time
  - Concurrently process independent portions of an event

- Is the event the right abstraction?
  - We are considering how to support user-defined levels of data aggregation.

- Work done during this time informs long-term R&D

Long-term R&D (2023-)

- Exploring in what ways our event model is compatible for single-node, grid, and HPC jobs.
- The intent is that the module authors' views of data would not change; the workflow orchestration, however, may be largely different.
  - Think solely in terms of datasets instead of files
  - Workflow details are subsumed by the framework
- SciDAC work is directly looking at an implementation for HPC jobs:
  - http://computing.fnal.gov/hep-on-hpc/
  - Addresses data-store aspect of distributed, hierarchical data

A few more remarks

- Many of the *art*'s concepts have been refined over the last few years
- In developing the framework, our *modus operandi* has been:

  No framework code is so precious that it must be saved; nothing is untouchable.

  Our users' code is precious, and we should break as little of it as possible.
- Much of the code internal to the framework has been replaced, without imposing many breaking changes on users.
- Perhaps surprisingly, though, many users *are* willing to move to different ways of thinking.
- Let's not be afraid of user perceptions.