MT in LArSoft update

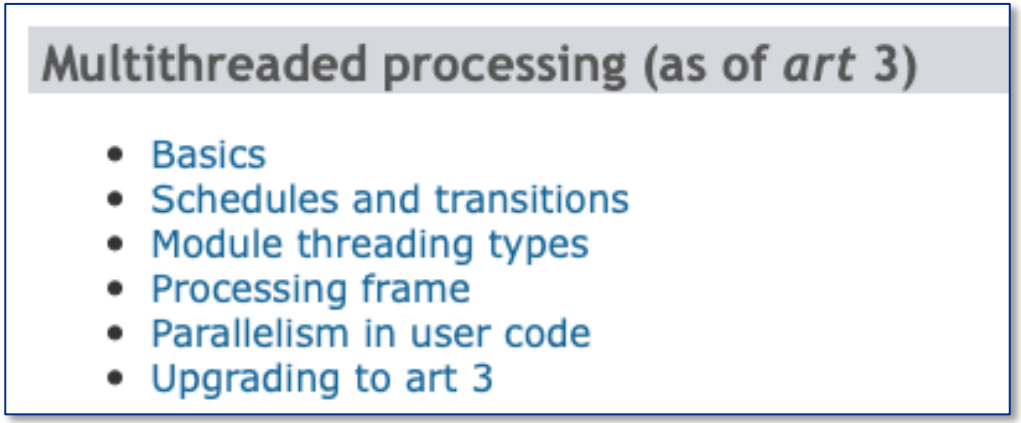
Kyle J. Knoepfel

23 April 2019

LArSoft coordination meeting

MT links

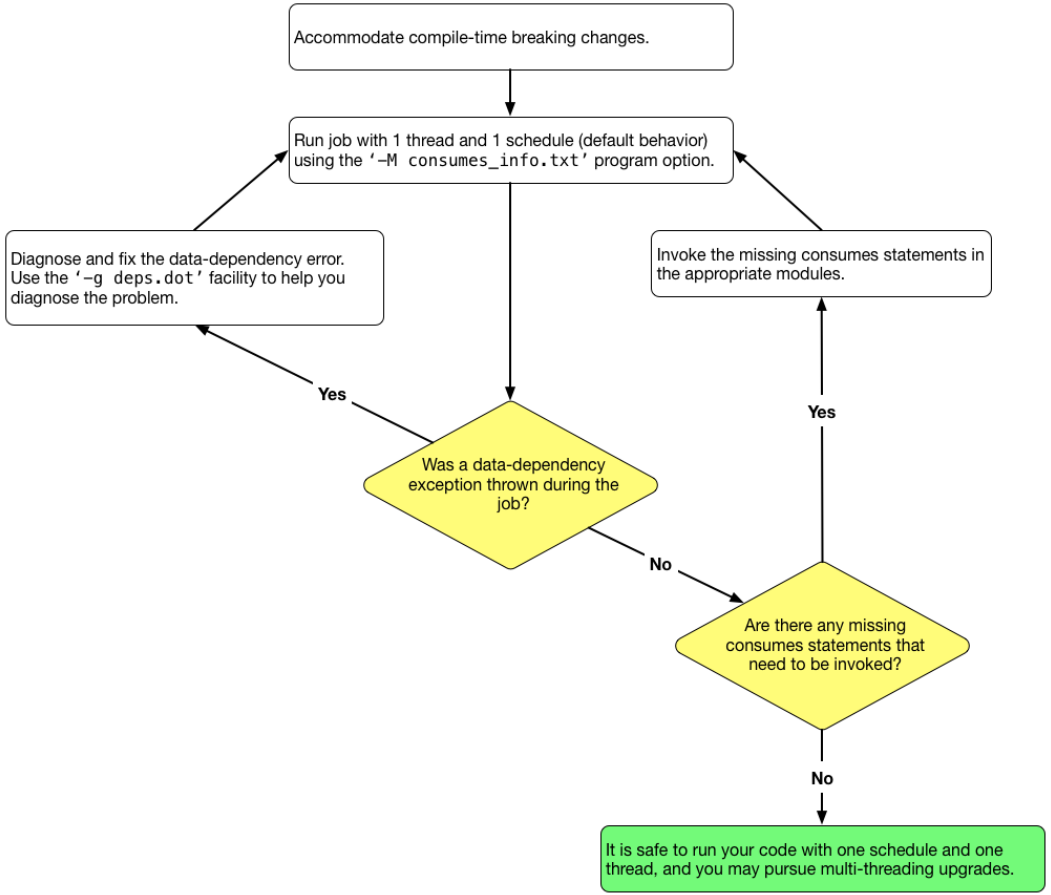
- https://cdcvs.fnal.gov/redmine/projects/art/wiki#Multithreaded-processing-as-of-art-3



Encouraged migration path

- If you look at the “Upgrade to art 3” link, you’ll see a flow chart for how to upgrade to *art* 3.

- Although this is the encouraged path, it can be difficult to adopt in LArSoft, which has ~50 services and ~250 modules.



Fermilab

Approaching upgrades

- I will save the more thorough description of how to write thread-safe code for the LArSoft workshop in June.
- No single approach works for each modules/services.
- The main thing to remember is:

  Data races occur when mutable data is shared among threads.

- Examples of shared data in framework job:
  - Services (shared among threads and events)
  - Modules (possibly shared among threads and events)

Approaching upgrades

- I will save the more thorough description of how to write thread-safe code for the LArSoft workshop in June.
- No single approach works for each modules/services.
- The main thing to remember is:

  Data races occur when mutable data is shared among threads.

- Examples of shared data in framework job:
  - Services (shared among threads and events)
  - Modules (possibly shared among threads and events)

Best way to make code thread-safe:
stop sharing data, or make the data immutable.

Approaching upgrades

- I will save the more thorough description of how to write thread-safe code for the LArSoft workshop in June.
- No single approach works for each modules/services.
- The main thing to remember is:

  Data races occur when mutable data is shared among threads.

- Examples of shared data in framework job:
  - Services (shared among threads and events)
  - Modules (possibly shared among threads and events)

Best way to make code thread-safe:
stop sharing data, or make the data immutable.

Making data immutable

- Many times, it is not necessary to have mutable data.

Making data immutable

- Many times, it is not necessary to have mutable data.

```
class FilterNoMCParticles : public art::SharedFilter {
public:
  explicit FilterNoMCParticles(fhicl::ParameterSet const& pset,
                               art::ProcessingFrame const&);
private:
  bool filter(art::Event&, art::ProcessingFrame const&) override;
  std::string const fLArG4ModuleLabel;
};
```

Making data immutable

- Many times, it is not necessary to have mutable data.

```
class FilterNoMCParticles : public art::SharedFilter {
public:
  explicit FilterNoMCParticles(fhicl::ParameterSet const& pset,
                               art::ProcessingFrame const&);
private:
  bool filter(art::Event&, art::ProcessingFrame const&) override;
  std::string const fLArG4ModuleLabel;
};
```

```
FilterNoMCParticles::FilterNoMCParticles(fhicl::ParameterSet const& pset,
                                         art::ProcessingFrame const&)
  : SharedFilter{pset}
  , fLArG4ModuleLabel{pset.get<std::string>("LArG4ModuleLabel", "NoLabel")}
{
  async<art::InEvent>();
}
```

Making data immutable

- Many times, it is not necessary to have mutable data.

```
class FilterNoMCParticles : public art::SharedFilter {
public:
  explicit FilterNoMCParticles(fhicl::ParameterSet const& pset,
                               art::ProcessingFrame const&);
private:
  bool filter(art::Event&, art::ProcessingFrame const&) override;
  std::string const fLArG4ModuleLabel;
};
```

```
FilterNoMCParticles::FilterNoMCParticles(fhicl::ParameterSet const& pset,
                                         art::ProcessingFrame const&)
  : SharedFilter{pset}
  , fLArG4ModuleLabel{pset.get<std::string>("LArG4ModuleLabel", "NoLabel")}
{
  async<art::InEvent>();
}
```

```
bool FilterNoMCParticles::filter(art::Event& evt, art::ProcessingFrame const&) {
  auto const& mcps = *evt.getValidHandle<std::vector<simb::MCParticle>>(fLArG4ModuleLabel);
  return not mcps.empty();
}
```

Making data immutable

- Many times, it is not necessary to have mutable data.

- If it is necessary, at least restrict the places where data are allowed to be mutable.
  - **To the extent possible, remove interface that modifies data**
  - Since the *art* framework does not support program reconfiguration (with the exception of classes that inherit from `evdb::Reconfigurable`), all `reconfigure` functions should be removed.

- I am working on some branches where the reconfigure functions have been removed.

What to expect in the future

- Fewer services, perhaps much fewer
  - LArFFT service replaced by dedicated class (Mike Wang)
  - More interactions through the Event, SubRun, and Run, which are guaranteed to have a thread-safe interface

What to expect in the future

- Fewer services, perhaps much fewer
  - LArFFT service replaced by dedicated class (Mike Wang)
  - More interactions through the Event, SubRun, and Run, which are guaranteed to have a thread-safe interface
- Constrained usage of `ServiceHandles`
  - *art* will likely make it difficult to create `ServiceHandles` outside of a module, service, or source.
  - If necessary, create `ServiceHandles` in modules, services, and sources. If a function needs the functionality provided by a service, create a handle in the module, and pass the dereferenced handle to the function.

What to expect in the future

- Fewer services, perhaps much fewer
  - LArFFT service replaced by dedicated class (Mike Wang)
  - More interactions through the Event, SubRun, and Run, which are guaranteed to have a thread-safe interface

- Constrained usage of `ServiceHandles`
  - *art* will likely make it difficult to create `ServiceHandles` outside of a module, service, or source.
  - If necessary, create `ServiceHandles` in modules, services, and sources. If a function needs the functionality provided by a service, create a handle in the module, and pass the dereferenced handle to the function.

Discouraged

```
double get_offset() {
  return ServiceHandle<Utility>{}->offset();
}
```

Encouraged

```
double get_offset(Utility const& util) {
  return util.offset();
}
```

A request

- Making code thread-safe is not easy.
- **We don't want to upgrade code that nobody uses!**
- Please take a look at the following lists:
  - https://cdcvs.fnal.gov/redmine/projects/knoepfel/wiki/MT_status_of_LArSoft_modules
  - https://cdcvs.fnal.gov/redmine/projects/knoepfel/wiki/Migration_path_for_LArSoft_services_in_art_3/
- If you are aware of any modules/services that are not in use, please let us know at scisoft-team@fnal.gov.
- If you are unsure of the thread-safety implications of your code, ask us!