Multithreading support in LArSoft

Kyle J. Knoepfel

2 March 2023

LArSoft MT workshop

Overview

- Previous talks and documentation
- Setting the stage
- art/LArSoft MT basics
- SHARED services in LArSoft
  - Notions of “current”
  - Persistent data structures
- Work in progress
  - Concurrent caching
- Summary

Overview

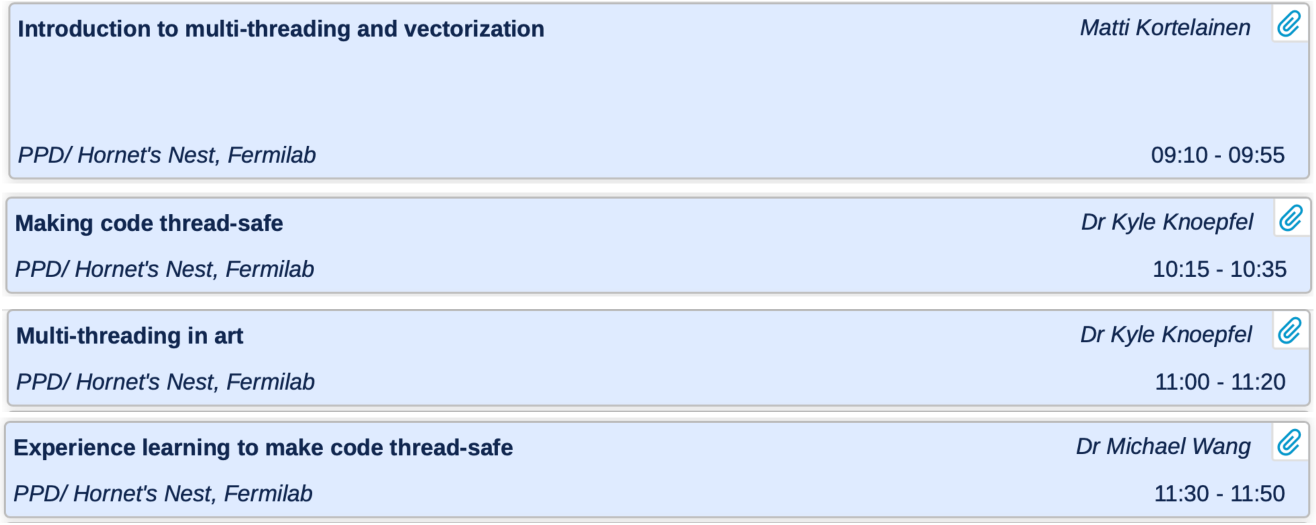
- Previous talks and documentation
- Setting the stage
- art/LArSoft MT basics
- SHARED services in LArSoft
  - Notions of “current”
  - Persistent data structures
- Work in progress
  - Concurrent caching
- Summary

I will not cover everything (sorry)

Interrupt and ask questions

Previous talks and documentation

- **MT basics — https://larsoft.org/larsoft-workshop-june-2019/**



Introduction to multi-threading and vectorization *Matti Kortelainen*
PPD/ Hornet's Nest, Fermilab 09:10 - 09:55

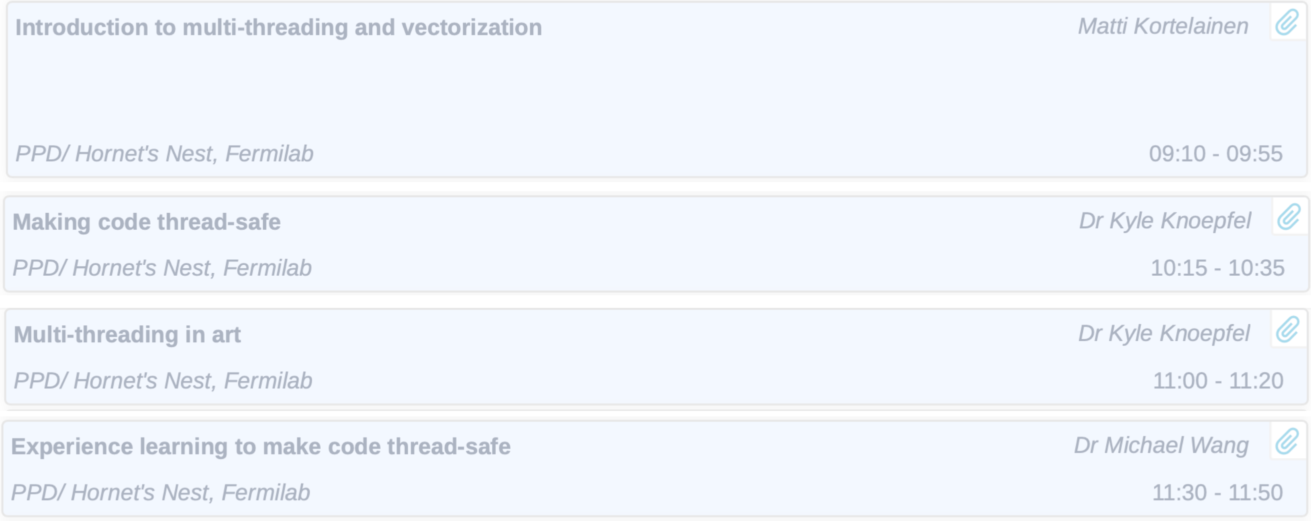
Making code thread-safe *Dr Kyle Knoepfel*
PPD/ Hornet's Nest, Fermilab 10:15 - 10:35

Multi-threading in art *Dr Kyle Knoepfel*
PPD/ Hornet's Nest, Fermilab 11:00 - 11:20

Experience learning to make code thread-safe *Dr Michael Wang*
PPD/ Hornet's Nest, Fermilab 11:30 - 11:50

Previous talks and documentation

- MT basics — https://larsoft.org/larsoft-workshop-june-2019/



Introduction to multi-threading and vectorization — *Matti Kortelainen*
PPD/ Hornet's Nest, Fermilab — 09:10 - 09:55

Making code thread-safe — *Dr Kyle Knoepfel*
PPD/ Hornet's Nest, Fermilab — 10:15 - 10:35

Multi-threading in art — *Dr Kyle Knoepfel*
PPD/ Hornet's Nest, Fermilab — 11:00 - 11:20

Experience learning to make code thread-safe — *Dr Michael Wang*
PPD/ Hornet's Nest, Fermilab — 11:30 - 11:50

- **MT within art**
  - https://cdcvs.fnal.gov/redmine/projects/art/wiki#Multithreaded-processing-as-of-art-3
  - I will cover the details which are necessary for this discussion

Setting the stage

- LArSoft is not intended to be used exclusively within an art context.

Setting the stage

- LArSoft is not intended to be used exclusively within an art context.

- To make LArSoft conducive to MT requires that LArSoft providers, algorithms, and data structures are thread-safe and efficient.
  - Today, though, I will focus primarily on services as they relate closely to LArSoft providers.

Setting the stage

- LArSoft is not intended to be used exclusively within an art context.

- To make LArSoft conducive to MT requires that LArSoft providers, algorithms, and data structures are thread-safe and efficient.
  - Today, though, I will focus primarily on services as they relate closely to LArSoft providers.

- Available SciSoft effort requires us to focus on specific constructs:
  1. We have targeted experiment-specific workflows and worked toward upgrading each component for MT (requires experiment buy-in and good communication)
  2. We have targeted widely used LArSoft providers and adjusted downstream code as necessary (easier to achieve, but harder to get a full end-to-end MT workflow)

art/LArSoft MT basics

- art and LArSoft do not manage their own threads—TBB does this for us.
  - Consequence: relying on `thread_local` statics is fragile

art/LArSoft MT basics

- art and LArSoft do not manage their own threads—TBB does this for us.
  - Consequence: relying on `thread_local` statics is fragile

- art's configuration options allow you to specify:
  - the maximum allowed concurrency (`--nthreads`)
  - the number of schedules (concurrent art events, `--nschedules`)
  - the stack size (default is 10 MiB)

art/LArSoft MT basics

- art and LArSoft do not manage their own threads—TBB does this for us.
  - Consequence: relying on `thread_local` statics is fragile
- art's configuration options allow you to specify:
  - the maximum allowed concurrency (`--nthreads`)
  - the number of schedules (concurrent art events, `--nschedules`)
  - the stack size (default is 10 MiB)
- Trigger/end paths can run in parallel on the same event
- Within a single module, you can invoke parallel algorithms
  - Best performance by using TBB's parallel algorithms

art/LArSoft MT basics

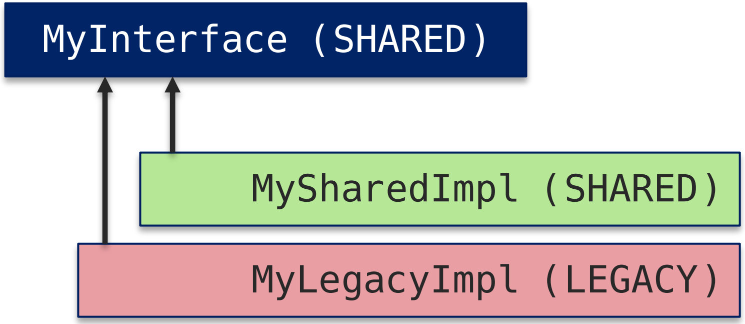
- The most difficult aspect of LArSoft MT-wise is the large number of services.

- If you would like to use a service with an art job configured with more than one thread, the service must have a scope of `SHARED`.

- Service *scope* definitions
  - `LEGACY`: service that can be used with only one schedule and only one thread configured
  - `SHARED`: service that can be used with *n* schedules and *m* threads

art/LArSoft MT basics

- The most difficult aspect of LArSoft MT-wise is the large number of services.
- If you would like to use a service with an art job configured with more than one thread, the service must have a scope of SHARED.
- Service *scope* definitions
  - LEGACY: service that can be used with only one schedule and only one thread configured
  - SHARED: service that can be used with *n* schedules and *m* threads
- As of art 3.05, service implementations are (nearly) decoupled from each other
  - LEGACY service interfaces must have LEGACY implementations
  - SHARED service interfaces may have either SHARED or LEGACY implementations



art/LArSoft MT basics

- **What happens if you try to run an MT job with a `LEGACY` service?**
  - Exception thrown with message:

```
The service 'MyInterface' (provider: 'MyLegacyImpl') is a legacy service,
which can be used with only one schedule and one thread.
This job uses 4 schedules and 4 threads.
Please reconfigure your job to use only one schedule/thread.
```

art/LArSoft MT basics

- **What happens if you try to run an MT job with a `LEGACY` service?**
  - Exception thrown with message:

```
The service 'MyInterface' (provider: 'MyLegacyImpl') is a legacy service,
which can be used with only one schedule and one thread.
This job uses 4 schedules and 4 threads.
Please reconfigure your job to use only one schedule/thread.
```

- **What if you want to process only one event at a time but still want to use multiple threads within that event?**
  - Use a SHARED service that inherits from `lar::EnsureOnlyOneSchedule<MySharedImpl>`
  - If you use more than one schedule for a job with such a service, you get another exception throw:

```
This job uses 4 schedules, but the type 'MySharedImpl' supports
processing only one event at a time. Please reconfigure your job
to use only one schedule.
```

SHARED services in LArSoft

Regular	Interfaces		Implementations
`geo::AuxDetGeometry`	`calib::IPhotonCalibratorService`		
`geo::Geometry`	`detinfo::DetectorClocksService`	→	`detinfo::DetectorClocksServiceStandard`
`sim::LArG4Parameters`	`detinfo::DetectorPropertiesService`	→	`detinfo::DetectorPropertiesServiceStandard`
`sim::LArVoxelCalculator`	`detinfo::LArPropertiesService`	→	`detinfo::LArPropertiesServiceStandard`
	`geo::AuxDetExptGeoHelperInterface`		
	`geo::ExptGeoHelperInterface`	→	`geo::StandardGeometryHelper`
	`lariov::ChannelStatusService`	→	`lariov::SIOVChannelStatusService`
	`lariov::DetPedestalService`	→	`lariov::SIOVDetPedestalService`
	`spacecharge::SpaceChargeService`	→	`spacecharge::SpaceChargeServiceStandard`

SHARED services in LArSoft

Regular	Interfaces	Implementations
geo::AuxDetGeometry	**calib::IPhotonCalibratorService**	
geo::Geometry	detinfo::DetectorClocksService	detinfo::DetectorClocksServiceStandard
sim::LArG4Parameters	detinfo::DetectorPropertiesService	detinfo::DetectorPropertiesServiceStandard
sim::LArVoxelCalculator	**detinfo::LArPropertiesService**	**detinfo::LArPropertiesServiceStandard**
	geo::AuxDetExptGeoHelperInterface	
	geo::ExptGeoHelperInterface	**geo::StandardGeometryHelper**
	lariov::ChannelStatusService	lariov::SIOVChannelStatusService
	lariov::DetPedestalService	lariov::SIOVDetPedestalService
	spacecharge::SpaceChargeService	**spacecharge::SpaceChargeServiceStandard**

Trivially thread-safe in the contexts used

SHARED services in LArSoft

Regular

- `geo::AuxDetGeometry`
- `geo::Geometry`
- `sim::LArG4Parameters`
- `sim::LArVoxelCalculator`

Interfaces	Implementations
`calib::IPhotonCalibratorService`	
`detinfo::DetectorClocksService` →	`detinfo::DetectorClocksServiceStandard`
`detinfo::DetectorPropertiesService` →	`detinfo::DetectorPropertiesServiceStandard`
`detinfo::LArPropertiesService` →	`detinfo::LArPropertiesServiceStandard`
`geo::AuxDetExptGeoHelperInterface`	
`geo::ExptGeoHelperInterface` →	`geo::StandardGeometryHelper`
`lariov::ChannelStatusService` →	**`lariov::SIOVChannelStatusService`**
`lariov::DetPedestalService` →	**`lariov::SIOVDetPedestalService`**
`spacecharge::SpaceChargeService` →	`spacecharge::SpaceChargeServiceStandard`

Trivially thread-safe in the contexts used

Thread-safe if only one event is processed at a time

SHARED services in LArSoft

Regular

- geo::AuxDetGeometry
- geo::Geometry
- sim::LArG4Parameters
- sim::LArVoxelCalculator

Interfaces	Implementations
calib::IPhotonCalibratorService	
detinfo::DetectorClocksService	**detinfo::DetectorClocksServiceStandard**
detinfo::DetectorPropertiesService	**detinfo::DetectorPropertiesServiceStandard**
detinfo::LArPropertiesService	detinfo::LArPropertiesServiceStandard
geo::AuxDetExptGeoHelperInterface	
geo::ExptGeoHelperInterface	geo::StandardGeometryHelper
lariov::ChannelStatusService	lariov::SIOVChannelStatusService
lariov::DetPedestalService	lariov::SIOVDetPedestalService
spacecharge::SpaceChargeService	spacecharge::SpaceChargeServiceStandard

Trivially thread-safe in the contexts used

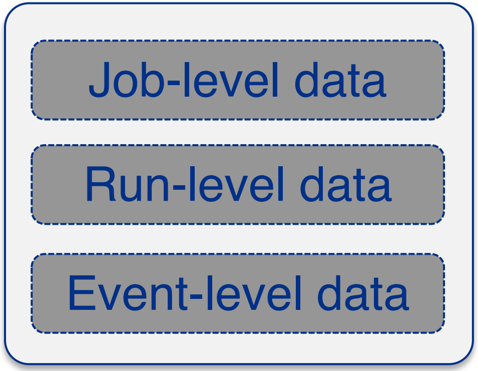
Thread-safe if only one event is processed at a time

Thread-safe by use of persistent data structures

- Breaks up monolithic data structures and avoids notions of “current”

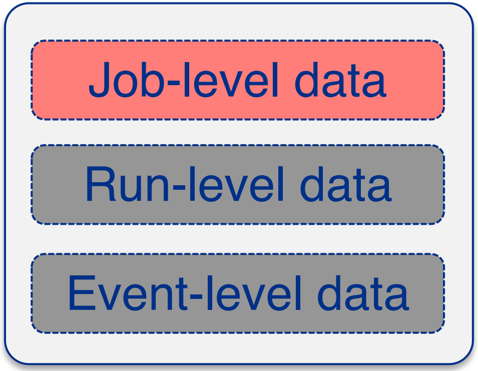
Problems with the idea of "current"

- Monolithic data structures are often chosen for managing *mutable* data corresponding to different processing granularities.



- This was true for various LArSoft facilities (e.g. `DetectorClocks` and `DetectorProperties`).
- It is inherently thread-*unsafe* as it often relies on the notion of "current", which is ill-defined in multi-threaded environments.

Problems with the idea of "current"



Thread 1	Thread 2
Create service	

Problems with the idea of “current”

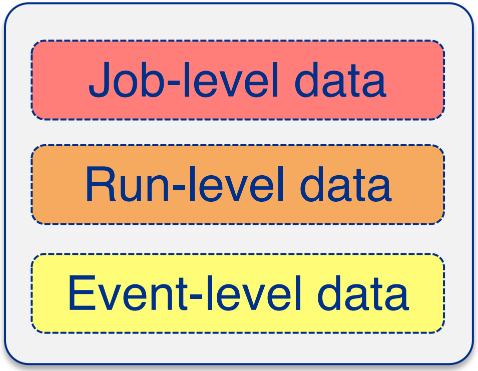
Job-level data

Run-level data

Event-level data

Thread 1	Thread 2
Begin Run 1	

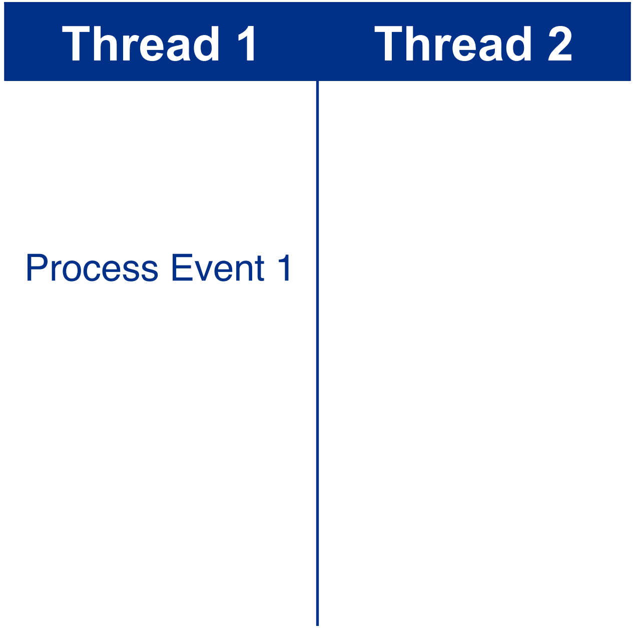
Problems with the idea of “current”



Job-level data

Run-level data

Event-level data

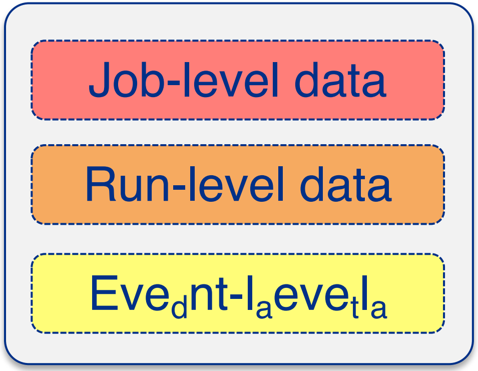


Thread 1

Thread 2

Process Event 1

Problems with the idea of “current”

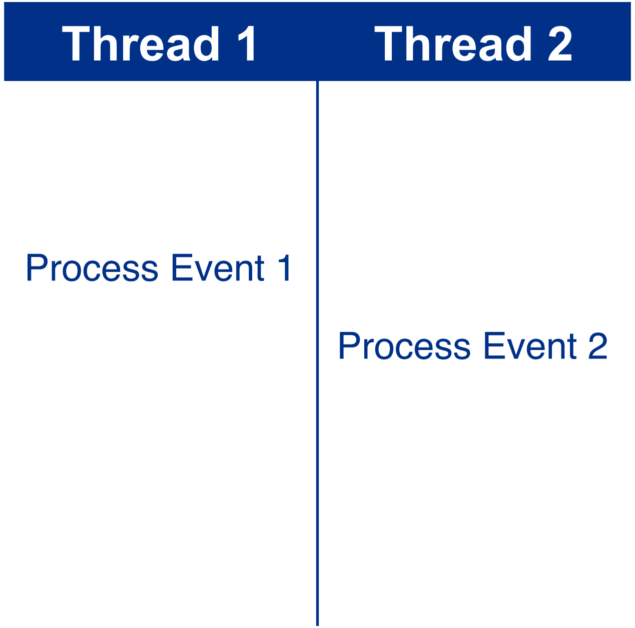


Job-level data

Run-level data

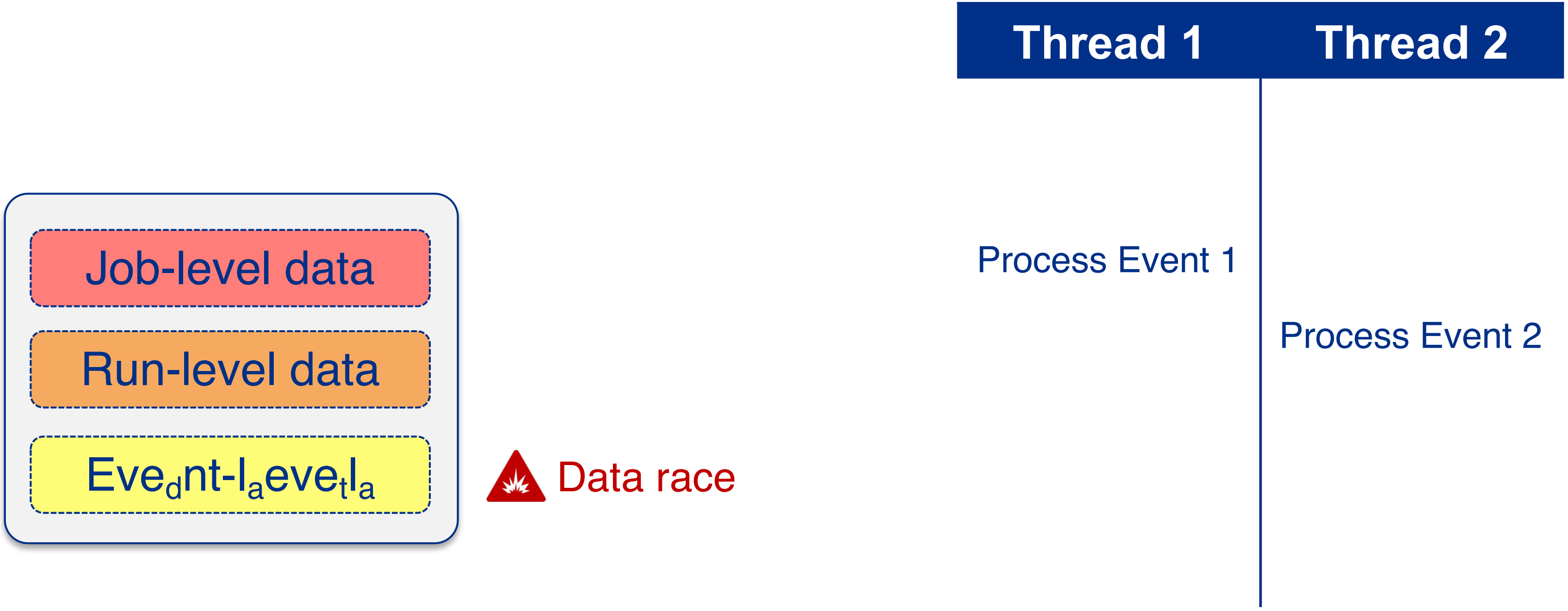
$\text{Eve}_{\text{d}}\text{nt-l}_{\text{a}}\text{eve}_{\text{t}}\text{l}_{\text{a}}$

⚠ Data race



Thread 1	Thread 2
Process Event 1	
	Process Event 2

Problems with the idea of “current”



Thread 1

Thread 2

Job-level data

Run-level data

Eve$_{d}$nt-l$_{a}$eve$_{t}$l$_{a}$

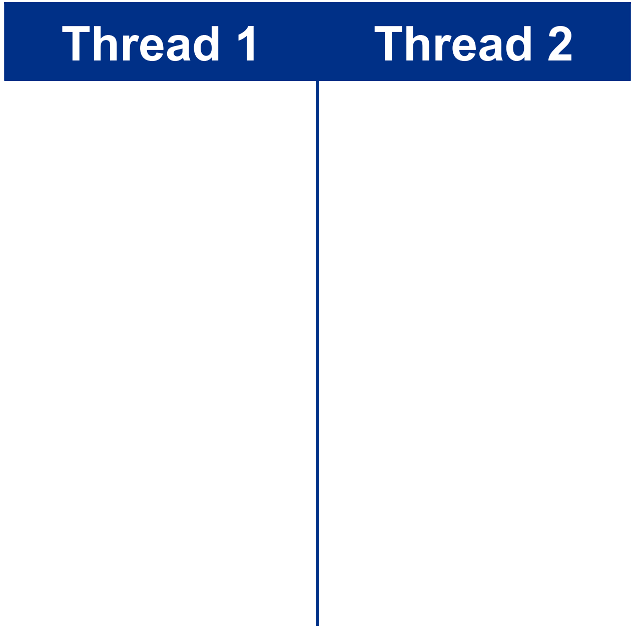
Data race

Process Event 1

Process Event 2

- To solve this problem for the `DetectorClocks` and `DetectorProperties` providers and services, we adopted the “persistent data structure” approach.
  - Data structures broken up according to the processing steps required.
  - In what follows, all boxes represent immutable objects.

Persistent data structure approach



Thread 1

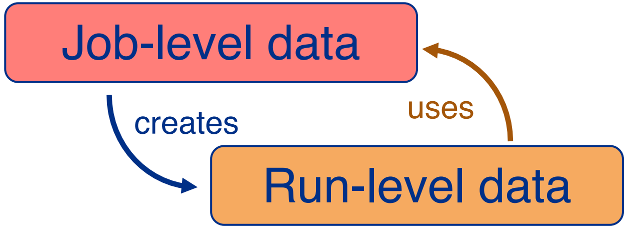
Thread 2

Persistent data structure approach

Job-level data

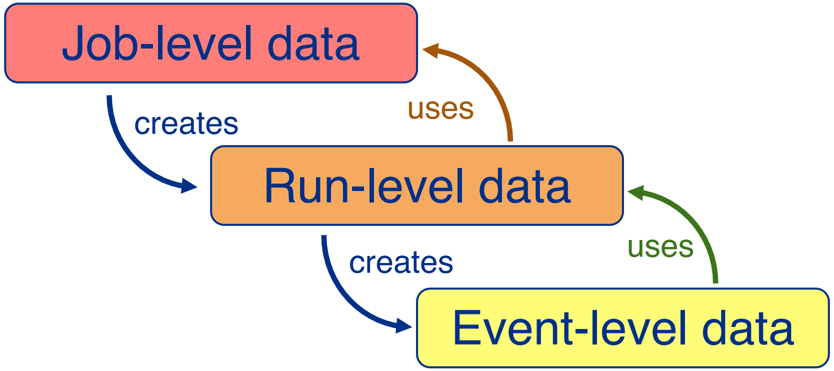
Thread 1	Thread 2
Create service	

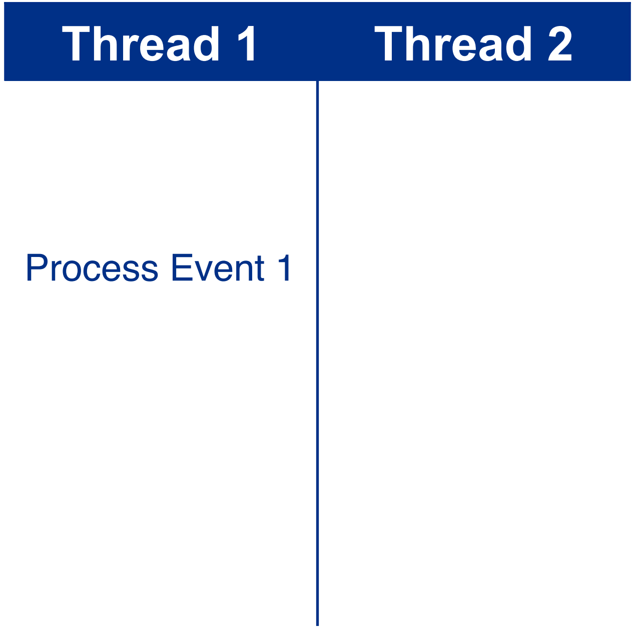
Persistent data structure approach



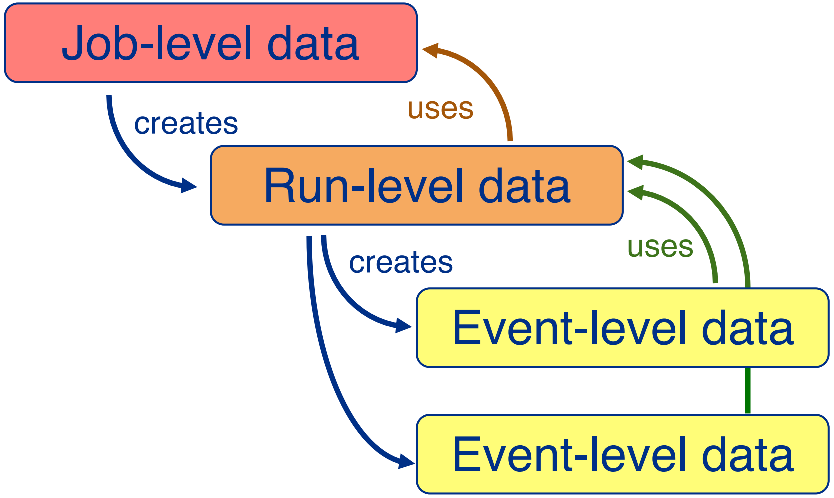
Thread 1	Thread 2
Begin Run 1	

Persistent data structure approach



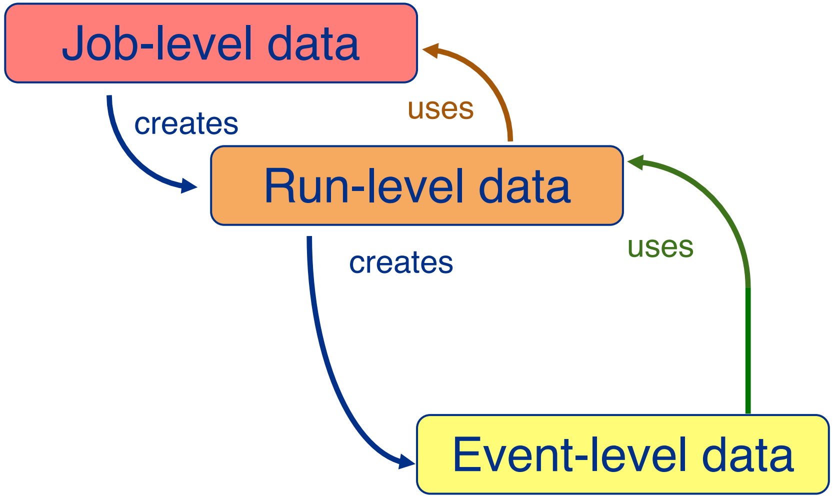


Persistent data structure approach



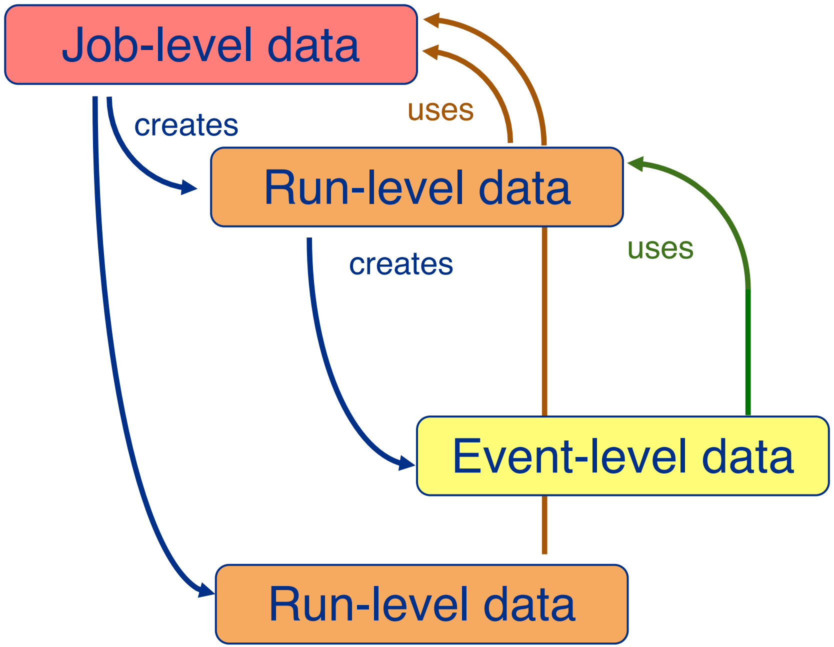
Thread 1	Thread 2
Process Event 1	
	Process Event 2

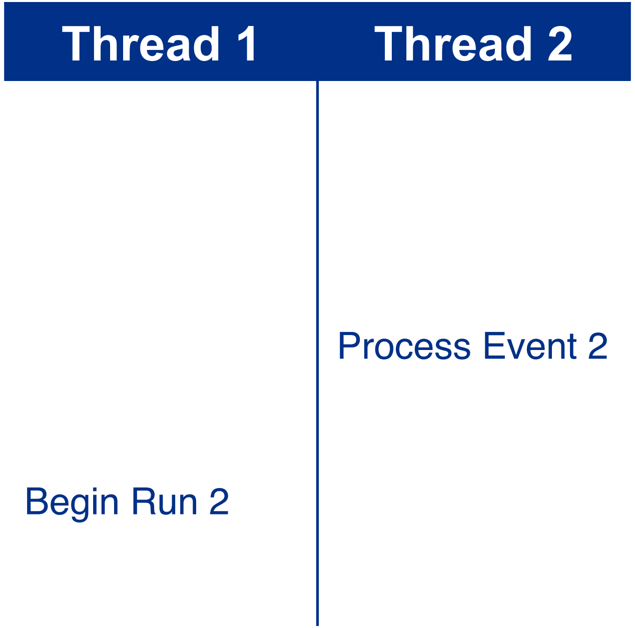
Persistent data structure approach



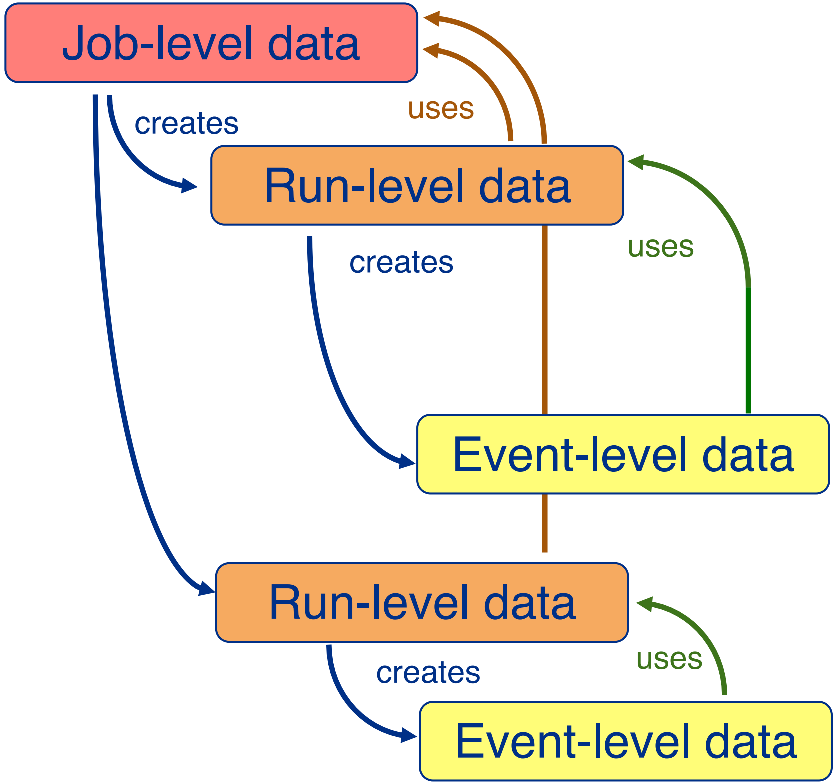
Thread 1	Thread 2
	Process Event 2
Finish Event 1	

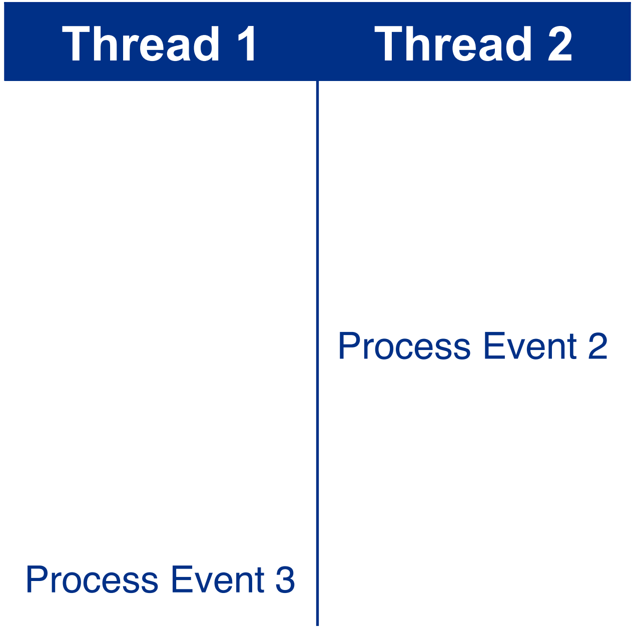
Persistent data structure approach



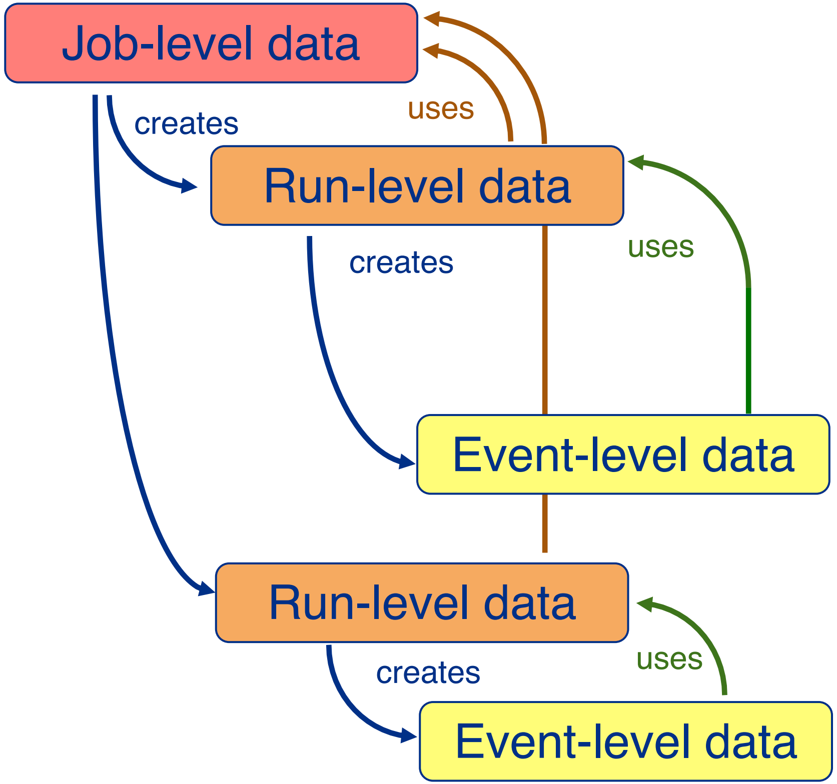


Persistent data structure approach



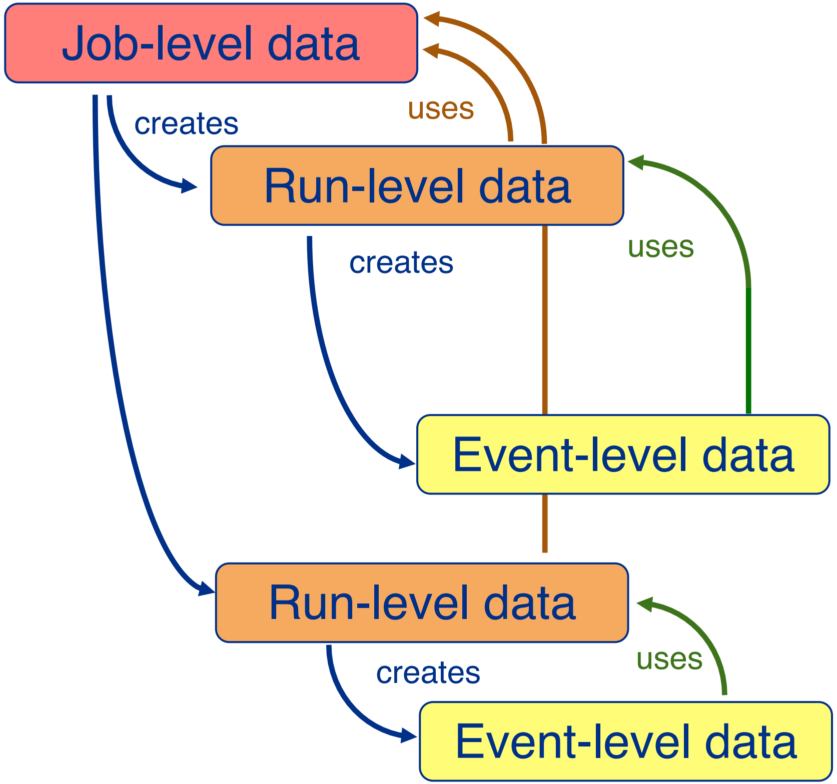


Persistent data structure approach



- **Why does this work?**
  - All objects are immutable.
  - Object construction/destruction happens on one thread.
  - Object of one processing level refers to the object directly above it (via pointer or reference).
  - Assuming data corresponding to each processing levels is small, extra overhead is minimal wrt. thread-unsafe option.

Persistent data structure approach



- **Why does this work?**
  - All objects are immutable.
  - Object construction/destruction happens on one thread.
  - Object of one processing level refers to the object directly above it (via pointer or reference).
  - Assuming data corresponding to each processing levels is small, extra overhead is minimal wrt. thread-unsafe option.
- **Downsides to this approach**
  - May require caching of data across threads. Not so much an issue for `DetectorClocks/Properties`.

What does DetectorClocks look like?

- As only events within a subrun can be processed concurrently at the moment, only event-level data must be thread-safe.

Old interface

```
using detinfo::DetectorClocksService;

MyProducer::MyProducer(ParameterSet const& pset)
{
  ServiceHandle<DetectorClocksService const> clocks;
  double beam_time = clocks->BeamGateTime();
}


void MyProducer::produce(art::Event& e)
{
  ServiceHandle<DetectorClocksService const> clocks;
  double beam_time = clocks->BeamGateTime();
}
```

New interface

```
using detinfo::DetectorClocksService;

MyProducer::MyProducer(ParameterSet const& pset)
{
  ServiceHandle<DetectorClocksService const> clocks;
  auto const clockData = clocks->DataForJob();
  double beam_time = clockData.BeamGateTime();
}

void MyProducer::produce(art::Event& e)
{
  ServiceHandle<DetectorClocksService const> clocks;
  auto const clockData = clocks->DataFor(e);
  double beam_time = clockData.BeamGateTime();
}
```

What does DetectorProperties look like?

- As only events within a subrun can be processed concurrently at the moment, only event-level data must be thread-safe.

Old interface

```
using detinfo::DetectorPropertiesService;

MyProducer::MyProducer(ParameterSet const& pset)
{
  ServiceHandle<DetectorPropertiesService> detProp;
  double dv = detProp->DriftVelocity(…);
}


void MyProducer::produce(art::Event& e)
{
  ServiceHandle<DetectorPropertiesService> detProp;
  double dv = detProp->DriftVelocity(…);
}
```

New interface

```
using detinfo::DetectorPropertiesService;

MyProducer::MyProducer(ParameterSet const& pset)
{
  ServiceHandle<DetectorPropertiesService> detProp;
  auto const dropData = detProp->DataForJob();
  double dv = dropData.DriftVelocity(…);
}

void MyProducer::produce(art::Event& e)
{
  ServiceHandle<DetectorPropertiesService> detProp;
  auto const dropData = detProp->DataFor(e);
  double dv = dropData.DriftVelocity(…);
}
```

Work in progress: services that access databases

- Some services lazily load data from a database and then cache the data across events (e.g.):
  - `lariov::SIOVChannelStatusService`
  - `lariov::SIOVDetPedestalService`

- They rely on the concept of "current" event and cannot be used when more than one schedule is configured.
  - They inherit from `lar::EnsureOnlyOneSchedule`.

- We have started adjusting them to use concurrent caching so that more than one event can be processed at a time
  - For caching details, see https://indico.fnal.gov/event/46562/

Concurrent cache basics

- The cache template is in *hep_concurrency.*

- It is a key-value map (e.g.):

```
hep::concurrency::cache<range_of_validity, calibration_offsets> offsets;
```

- The `range_of_validity` type is user-defined and represents a {start, stop}.

- For a `range_of_validity` that represents {0, 10}, you can type:

```
if (auto handle = offsets.entry_for(7)) {
  calibration_offsets const& offset = *handle; // entry for {0, 10}
  handle->some_member_function_of_calibration_offsets();
}
```

- Methods for thread-safe insertion and deletion of cache entries

LArSoft MT Miscellany

- Not everything has to be a service:
  - **This is preferred!**
  - LArFFTW (regular class owned by a module) replaces LArFFT (global service)

- There are a few modules that use TBB parallel algorithms:
  - `larreco/HitFinder/GausHitFinder_module.cc`
  - `larrecodnn/ImagePatternAlgs/Keras/keras_model.cc`

LArSoft MT summary

- SciSoft team efforts have primarily focused on making LArSoft providers and services thread-safe/efficient.

- The thread-safety approach depends on the context. Approaches include:
  - Make everything immutable
  - Restrict execution to one schedule
  - Use persistent data structures
  - Use concurrent caching
  - Replicate data across schedules

LArSoft MT summary

- SciSoft team efforts have primarily focused on making LArSoft providers and services thread-safe/efficient.

- The thread-safety approach depends on the context. Approaches include:
  - Make everything immutable
  - Restrict execution to one schedule
  - Use persistent data structures
  - Use concurrent caching
  - Replicate data across schedules

- We have targeted specific experiment workflows and providers that appear to be heavily used. But there's more to do…thanks for your patience.

We would like to hear from you about what we should target.