Framework/Wire-Cell interface aspects

Kyle Knoepfel

The Second Wire-Cell Reconstruction Summit

11 April 2024

Definitions

Wire-Cell

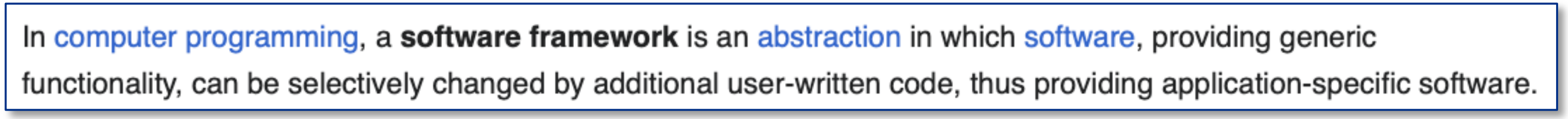
(you already know this one)

Definitions

Wire-Cell

(you already know this one)

Framework



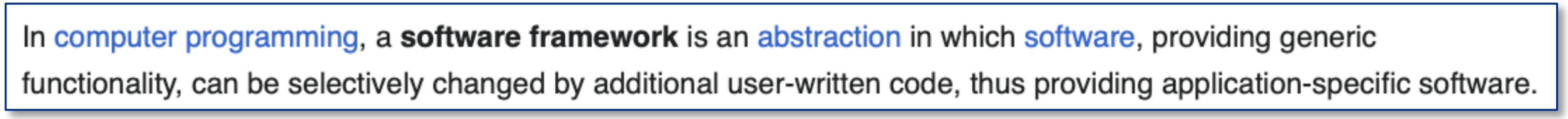
In computer programming, a **software framework** is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.

Definitions

Wire-Cell

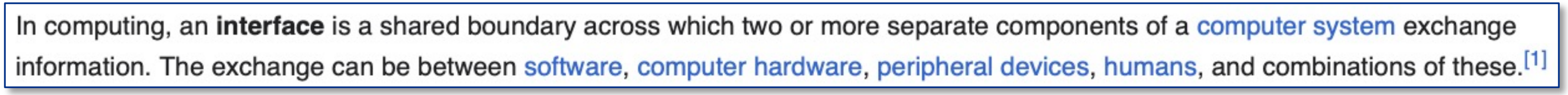
(you already know this one)

Framework



In computer programming, a **software framework** is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.

Interface



In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

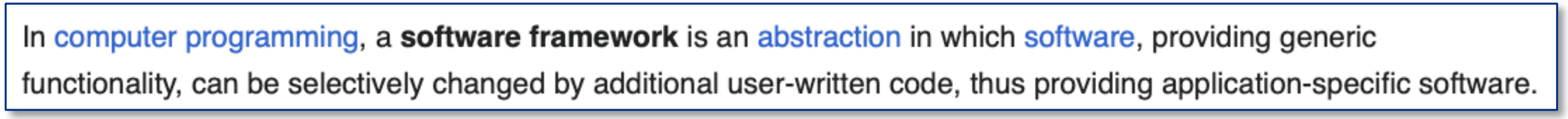
Fermilab

Definitions

Wire-Cell

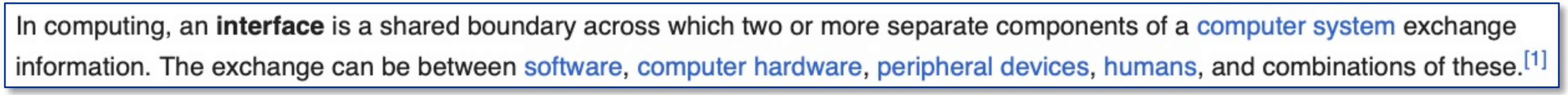
(you already know this one)

Framework



In computer programming, a **software framework** is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.

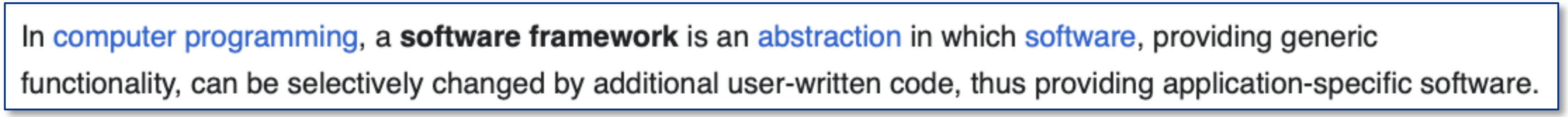
Interface



In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

We'll start with framework concepts and then get to interface aspects.

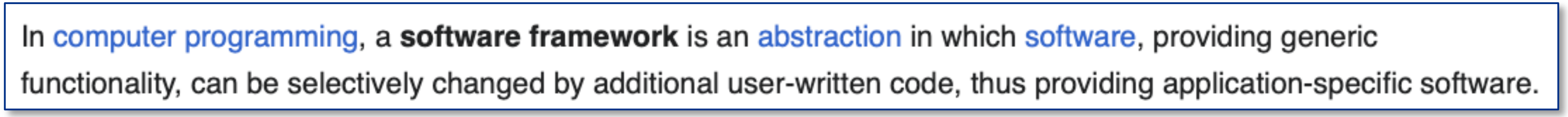
HEP frameworks



In computer programming, a **software framework** is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.

- The term “framework” is used often in HEP, usually without definition. The assumption is that “you know it when you see it,” or that you recognize the contexts in which it is used (HLT, reconstruction, etc.).
- The user “plugs in” their code to a framework, often through dynamically loaded libraries called “plugins”.

HEP frameworks


In computer programming, a **software framework** is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.

- The term "framework" is used often in HEP, usually without definition. The assumption is that "you know it when you see it," or that you recognize the contexts in which it is used (HLT, reconstruction, etc.).
- The user "plugs in" their code to a framework, often through dynamically loaded libraries called "plugins".
- This often means that the framework owns the '`int main(…)`' function, which calls user code under the covers.
- Frameworks are often used in a high-level trigger environment, for reconstructing physics objects from detector signals, or for simulating physics processes.
  - HEP frameworks are often not used in the context of analysis.

Existing HEP frameworks

There are several existing frameworks (e.g.):

- ***art***: Used by most intensity-frontier experiments at Fermilab. Originated as a fork of CMSSW ca. 2010.
- **CMSSW framework**: Used by the CMS experiment. CMSSW comprises the framework, other core functionality, physics algorithms, etc.
- **Gaudi**: Used by ATLAS, LHCb, Daya Bay, and MINERvA. Gaudi comes in different flavors highly tailored according to the experiments' needs.
- **JANA2**: Used by the Electron-Ion Collider.
- **O2**: Used by ALICE.

Existing HEP frameworks

There are several existing frameworks (e.g.):

- ***art***: Used by most intensity-frontier experiments at Fermilab. Originated as a fork of CMSSW ca. 2010.
- **CMSSW framework**: Used by the CMS experiment. CMSSW comprises the framework, other core functionality, physics algorithms, etc.
- **Gaudi**: Used by ATLAS, LHCb, Daya Bay, and MINERvA. Gaudi comes in different flavors highly tailored according to the experiments' needs.
- **JANA2**: Used by the Electron-Ion Collider.
- **O2**: Used by ALICE.
- **(Wire-Cell**: Contains many framework ingredients modulo a provenance system and an extensive I/O layer.)

What we are not talking about

Examples that are not data-processing frameworks:

- **LArSoft**: LAr toolkit/library that is designed to be framework-agnostic
- **ROOT**: toolkit used ubiquitously in HEP analysis and as underlying library for persisting C++ data structures to disk.
- **Geant4**: library used for detector simulation, response, etc.
- **A particular computing language (C++, Python, etc.)**: Even though a framework is often implemented in a specific language, the language is not a framework.
- **Small programs written by physicists**: often used for individual analyses.

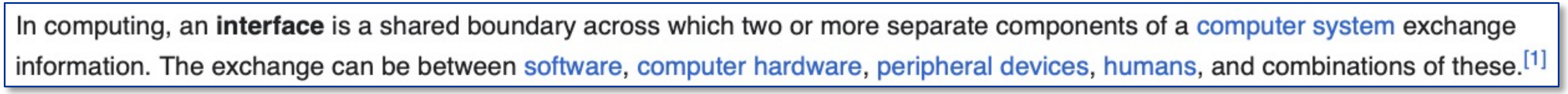
What we are not talking about

Examples that are not data-processing frameworks:

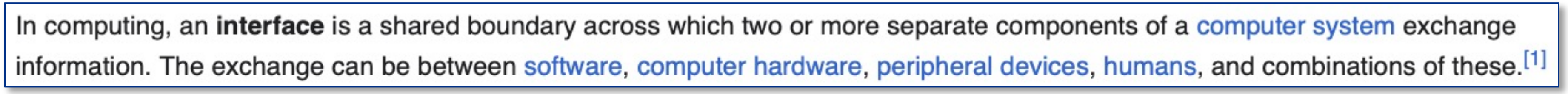
- **LArSoft**: LAr toolkit/library that is designed to be framework-agnostic
- **ROOT**: toolkit used ubiquitously in HEP analysis and as underlying library for persisting C++ data structures to disk.
- **Geant4**: library used for detector simulation, response, etc.
- **A particular computing language (C++, Python, etc.)**: Even though a framework is often implemented in a specific language, the language is not a framework.
- **Small programs written by physicists**: often used for individual analyses.

A framework is intended for decomposing workflows into units.
It is a means of sharing code, data, and workflow patterns.

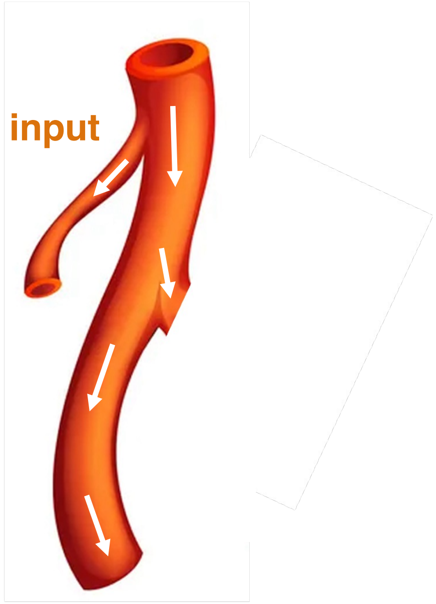
Interface aspects

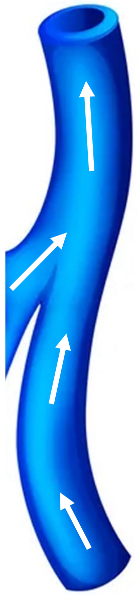

In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

Interface aspects


In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

framework input data flow

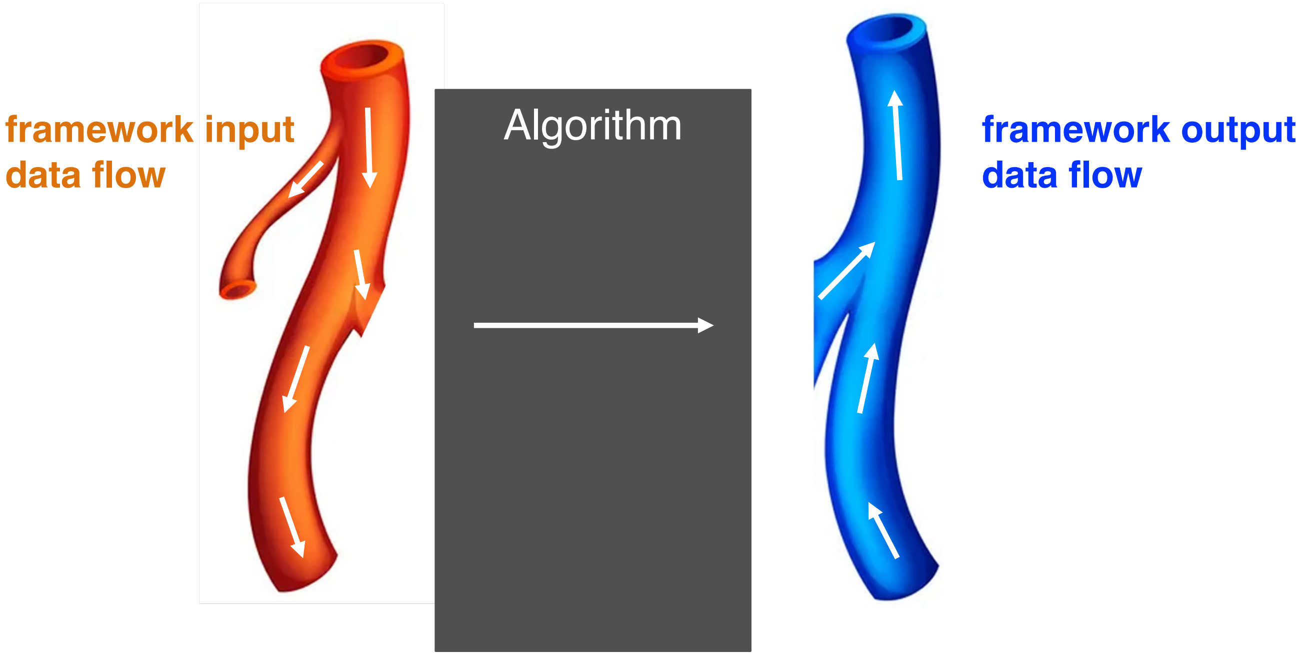




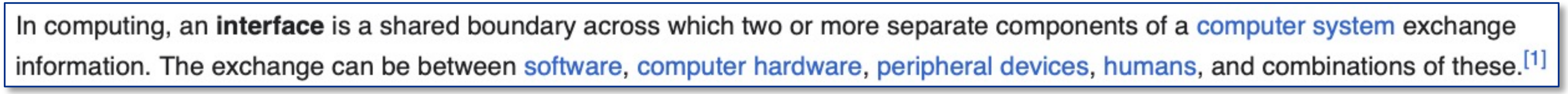
framework output data flow

Interface aspects

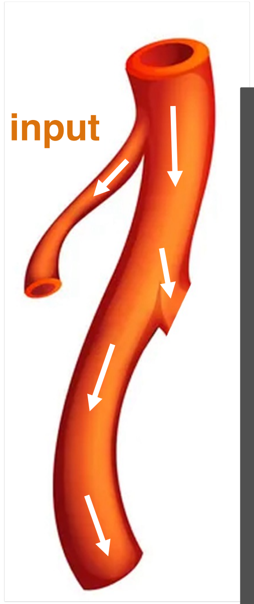
In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

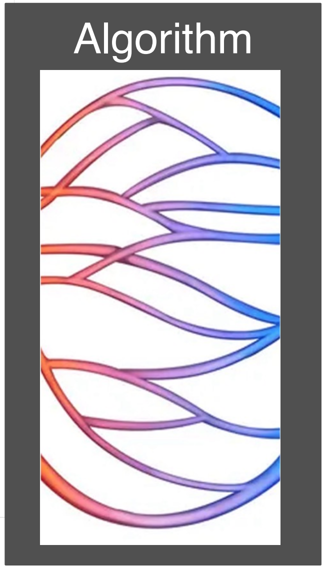


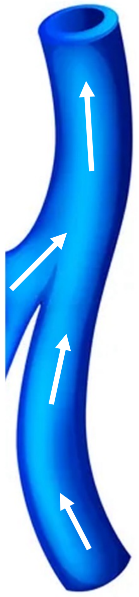
Interface aspects


In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

framework input data flow



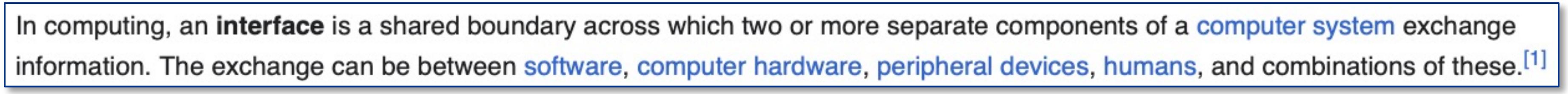




framework output data flow

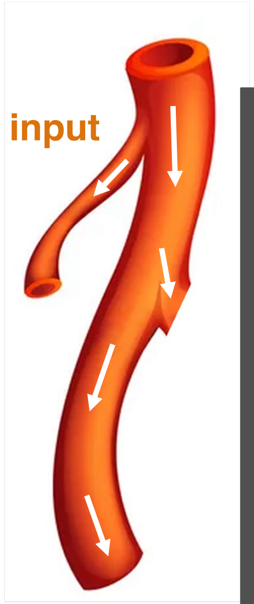
Algorithms have internal structure / data flow

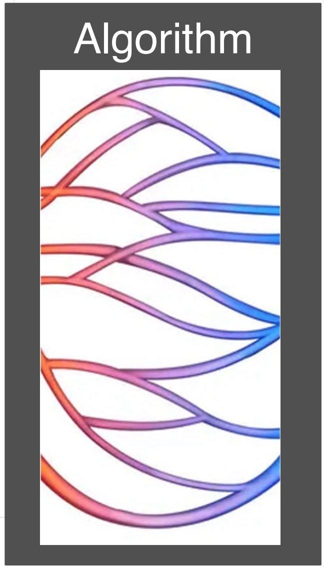
Interface aspects

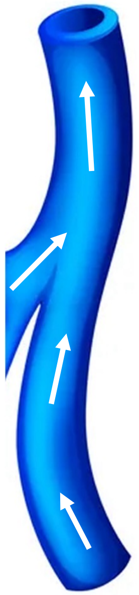


In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

framework input data flow





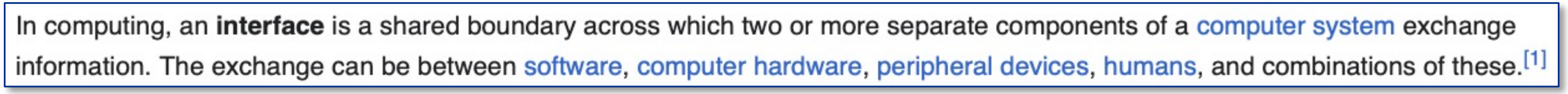


framework output data flow

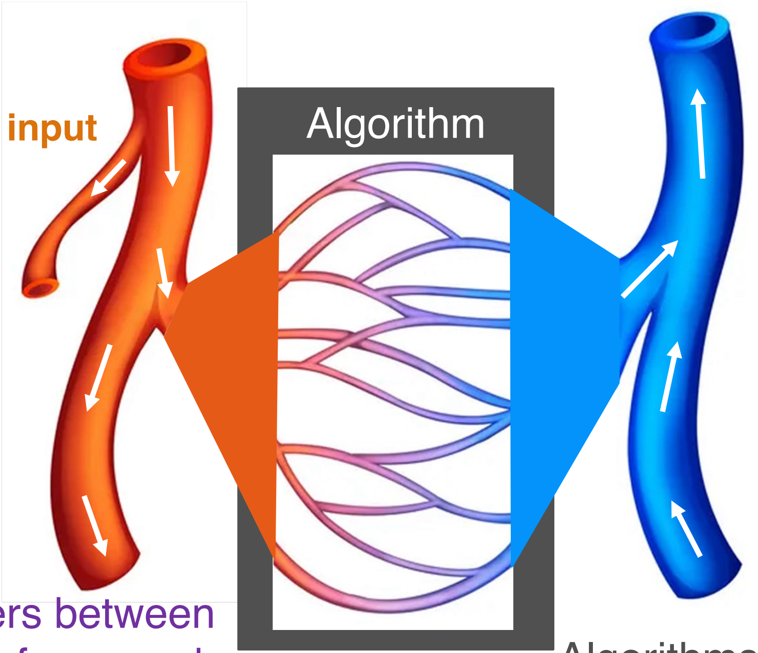
This can lead to mismatches in data flow concepts between the framework and the algorithm(s).

Algorithms have internal structure / data flow

Interface aspects


In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.[1]

framework input data flow



framework output data flow

This can lead to mismatches in data flow concepts between the framework and the algorithm(s).

Coarse translation layers between Wire-Cell and the framework

Algorithms have internal structure / data flow

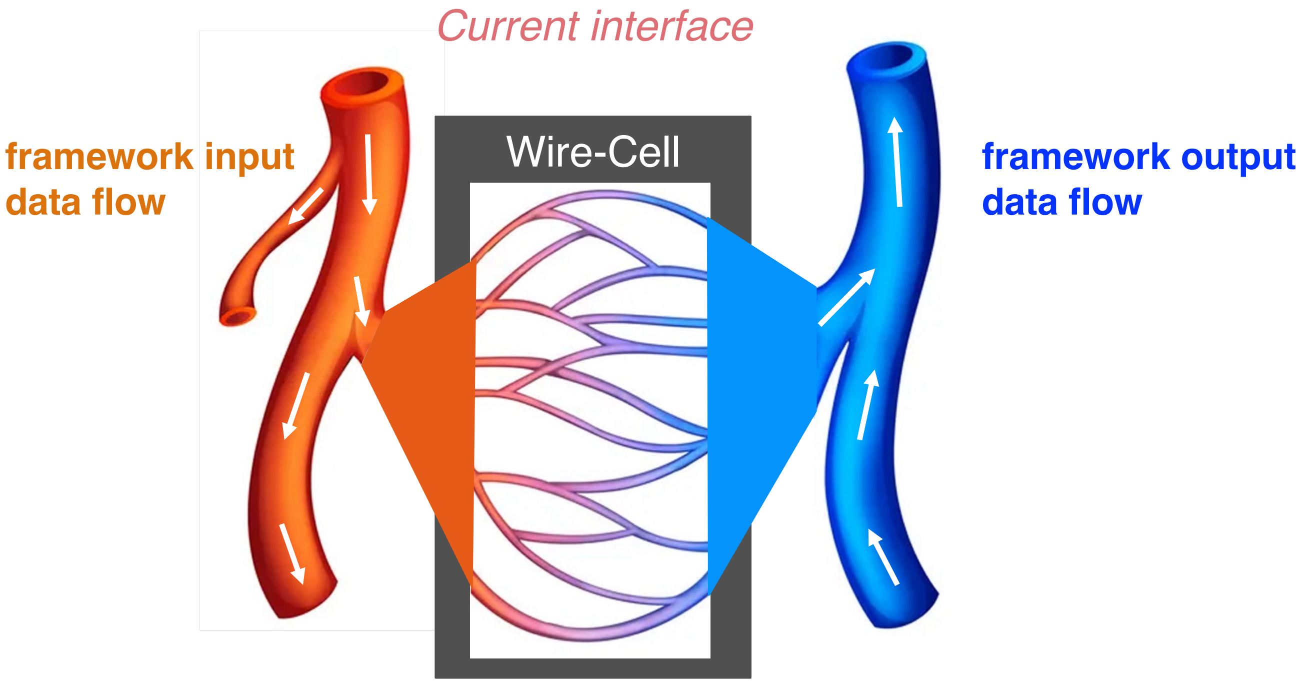
Translation layers between Wire-Cell and the framework

- See Brett's earlier talk

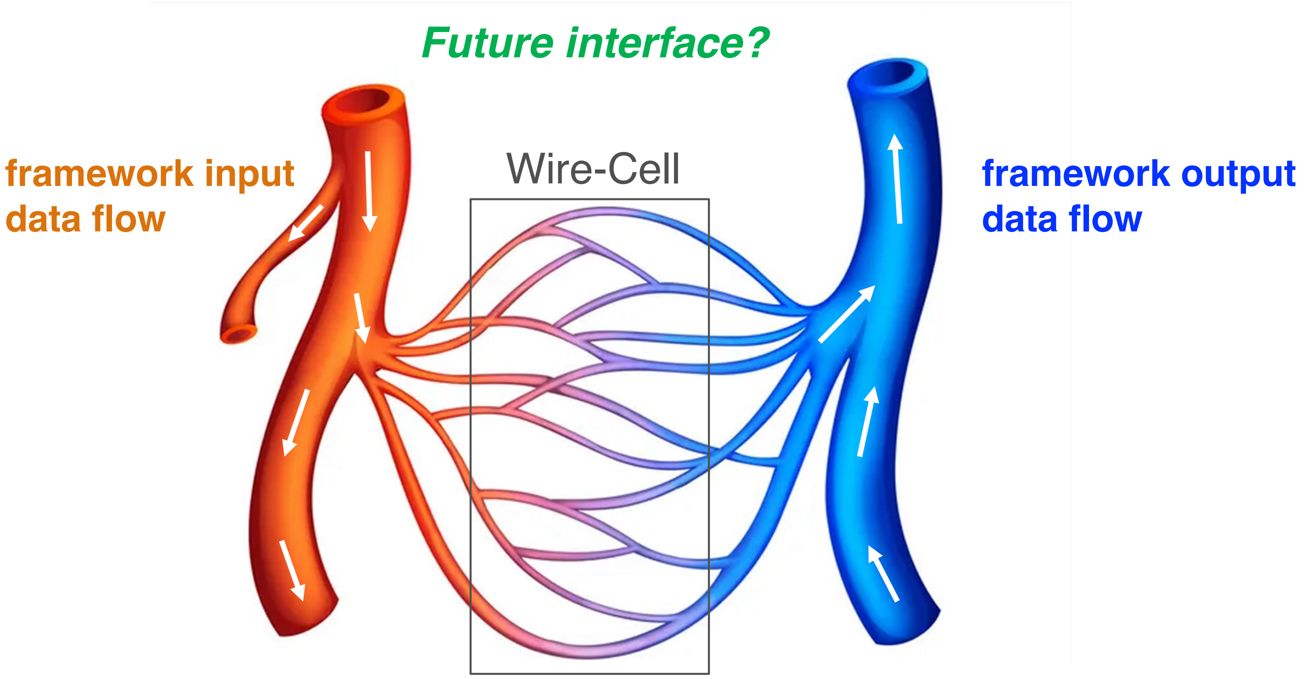
- Some complications create an "impedance mismatch" that (a) requires extra effort to overcome, and (b) results in execution inefficiencies:
    1. **Differences in configuration patterns** used by different libraries (not just the languages)
    2. **Mismatch in data layouts** expected by different libraries (LArSoft vs. Wire-Cell)
    3. **Suboptimal granularity of data** as presented to Wire-Cell (event-level vs. frame-level vs. …)

- Items 1 and 2 resolved by coordination among libraries
- Solving item 3 requires a better framework solution (today's focus)

A better framework / Wire-Cell interface



A better framework / Wire-Cell interface



Can the interface boundary between the framework and Wire-Cell be thinner?
- Ideally, the framework presents the data in a way that is best suited for Wire-Cell.

Meld and DUNE’s future framework

- Meld is a prototype framework that was developed as part of an LDRD project, aimed at exploring what would meet DUNE’s data-processing needs.
  - https://github.com/knoepfel/meld

- Its goals:
  1. Enable users to specify the organization of data that makes the most sense for the job
  2. Optimize workflow execution of user algorithms through graph-based data-flow processing
  3. Adopt programming patterns that are naturally thread-safe using higher-order functions
  4. Minimize boilerplate code for registering user algorithms with the framework

- Fermilab’s CSAI directorate has stated its commitment to develop a DUNE framework in a manner motivated by Meld’s model.

Different data organizations

- In the most widely used HEP frameworks, the data are organized according to a rigid hierarchy of data-processing levels:

$$run \supset subrun \supset event$$

- DUNE needs something more flexible than this:
  - Different processing levels must be specifiable by the user
  - The relationships among the processing levels might be more complicated

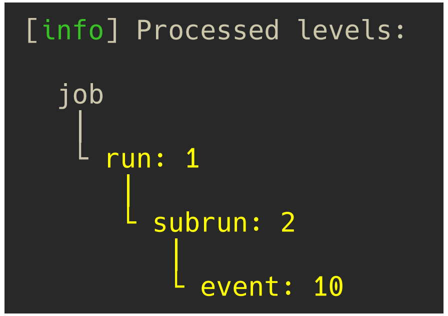
Different data organizations

- In the most widely used HEP frameworks, the data are organized according to a rigid hierarchy of data-processing levels:

$$run \supset subrun \supset event$$

- DUNE needs something more flexible than this:
  - Different processing levels must be specifiable by the user
  - The relationships among the processing levels might be more complicated

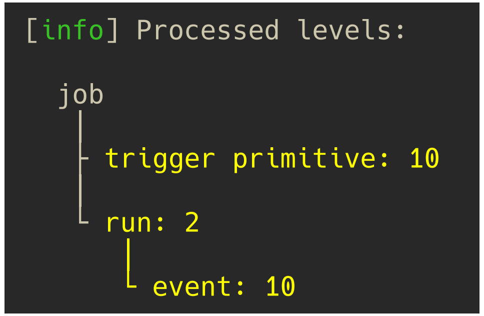
Sample hierarchies:



```
[info] Processed levels:

  job
   │
   └ run: 1
       │
       └ subrun: 2
           │
           └ event: 10
```

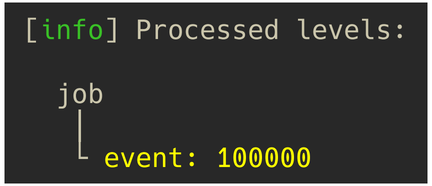
art-based hierarchy



```
[info] Processed levels:

  job
   │
   ├ trigger primitive: 10
   │
   └ run: 2
       │
       └ event: 10
```

Non-trivial hierarchy



```
[info] Processed levels:

  job
   │
   └ event: 100000
```

Flat hierarchy

Framework boilerplate

- Create tracks from hits for each event.

```
Tracks make_tracks(Hits const& hits) { ... }
```

Framework boilerplate

- Create tracks from hits for each event.

```
Tracks make_tracks(Hits const& hits) { ... }
```

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

Tracks make_tracks(Hits const& hits) { ... }

namespace expt {
  class TrackMaker : public art::SharedProducer {
  public:
    TrackMaker(fhicl::ParameterSet const&)
    {
      consumes<Hits, art::InEvent>("GoodHits");
      produces<Tracks, art::InEvent>("GoodTracks");

      async<art::InEvent>();
    }

    void produce(art::Event& e,
                 art::ProcessingFrame const&) override
    {
      auto const& hits = e.getProduct<Hits>("GoodHits");
      auto tracks = make_tracks(hits);
      e.put(std::make_unique<Tracks>(std::move(tracks)),
            "GoodTracks");
    }
  };
}

DEFINE_ART_MODULE(expt::TrackMaker)
```

Framework boilerplate

- Create tracks from hits for each event.

```
Tracks make_tracks(Hits const& hits) { ... }
```

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

Tracks make_tracks(Hits const& hits) { ... }

namespace expt {
  class TrackMaker : public art::SharedProducer {
  public:
    TrackMaker(fhicl::ParameterSet const&)
    {
      consumes<Hits, art::InEvent>("GoodHits");
      produces<Tracks, art::InEvent>("GoodTracks");

      async<art::InEvent>();
    }

    void produce(art::Event& e,
                 art::ProcessingFrame const&) override
    {
      auto const& hits = e.getProduct<Hits>("GoodHits");
      auto tracks = make_tracks(hits);
      e.put(std::make_unique<Tracks>(std::move(tracks)),
            "GoodTracks");
    }
  };
}

DEFINE_ART_MODULE(expt::TrackMaker)
```

Please don't read this.

It has what you need to use `make_tracks` in art.

But using `make_tracks` in a framework should be easier.

Framework boilerplate

- Create tracks from hits for each event.

```
Tracks make_tracks(Hits const& hits) { ... }
```

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

Tracks make_tracks(Hits const& hits) { ... }

namespace expt {
  class TrackMaker : public art::SharedProducer {
  public:
    TrackMaker(fhicl::ParameterSet const&)
    {
      consumes<Hits, art::InEvent>("GoodHits");
      produces<Tracks, art::InEvent>("GoodTracks");

      async<art::InEvent>();
    }

    void produce(art::Event& e,
                 art::ProcessingFrame const&) override
    {
      auto const& hits = e.getProduct<Hits>("GoodHits");
      auto tracks = make_tracks(hits);
      e.put(std::make_unique<Tracks>(std::move(tracks)),
            "GoodTracks");
    }
  };
}

DEFINE_ART_MODULE(expt::TrackMaker)
```

An easier way

```
#include "meld/module.hpp"

Tracks make_tracks(Hits const& hits) { ... }

DEFINE_MODULE(m, config) {
  m.with(make_tracks, concurrency::unlimited)
   .transform("GoodHits").in_each("Event")
   .to("GoodTracks");
}
```

Framework boilerplate

- Create tracks from hits for each event.

```
Tracks make_tracks(Hits const& hits) { ... }
```

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

Tracks make_tracks(Hits const& hits) { ... }

namespace expt {
  class TrackMaker : public art::SharedProducer {
  public:
    TrackMaker(fhicl::ParameterSet const&)
    {
      consumes<Hits, art::InEvent>("GoodHits");
      produces<Tracks, art::InEvent>("GoodTracks");

      async<art::InEvent>();
    }

    void produce(art::Event& e,
                 art::ProcessingFrame const&) override
    {
      auto const& hits = e.getProduct<Hits>("GoodHits");
      auto tracks = make_tracks(hits);
      e.put(std::make_unique<Tracks>(std::move(tracks)),
            "GoodTracks");
    }
  };
}

DEFINE_ART_MODULE(expt::TrackMaker)
```

An easier way

```
#include "meld/module.hpp"

Tracks make_tracks(Hits const& hits) { ... }

DEFINE_MODULE(m, config) {
  m.with(make_tracks, concurrency::unlimited)
   .transform("GoodHits").in_each("Event")
   .to("GoodTracks");
}
```

- Minimal boilerplate.
- `Event` is now a label.

Framework boilerplate

- Create tracks from hits for each event.

```
Tracks make_tracks(Hits const& hits) { ... }
```

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

Tracks make_tracks(Hits const& hits) { ... }

namespace expt {
  class TrackMaker : public art::SharedProducer {
  public:
    TrackMaker(fhicl::ParameterSet const&)
    {
      consumes<Hits, art::InEvent>("GoodHits");
      produces<Tracks, art::InEvent>("GoodTracks");

      async<art::InEvent>();
    }

    void produce(art::Event& e,
                 art::ProcessingFrame const&) override
    {
      auto const& hits = e.getProduct<Hits>("GoodHits");
      auto tracks = make_tracks(hits);
      e.put(std::make_unique<Tracks>(std::move(tracks)),
            "GoodTracks");
    }
  };
}

DEFINE_ART_MODULE(expt::TrackMaker)
```

An easier way

```
#include "meld/module.hpp"

Tracks make_tracks(Hits const& hits) { ... }

DEFINE_MODULE(m, config) {
  m.with(make_tracks, concurrency::unlimited)
   .transform("GoodHits").in_each("Event")
   .to("GoodTracks");
}
```

- Minimal boilerplate.
- `Event` is now a label.

The system developed for DUNE will support various data-processing patterns, conditions information, etc.

Relevance to Wire-Cell

- Both Meld and Wire-Cell use oneTBB’s flow-graph concurrency library (shared thread pool)

- Incorporating Wire-Cell components to run within a framework context could be simpler (e.g.):
  - Break apart an “event” **at the framework level** into (e.g.) “frames”
  - Register Wire-Cell components to operate on data contained in frames
  - If necessary, collect (or *reduce*) Wire-Cell generated data into “event”-level data

Relevance to Wire-Cell

- Both Meld and Wire-Cell use oneTBB's flow-graph concurrency library (shared thread pool)

- Incorporating Wire-Cell components to run within a framework context could be simpler (e.g.):
  - Break apart an "event" **at the framework level** into (e.g.) "frames"
  - Register Wire-Cell components to operate on data contained in frames
  - If necessary, collect (or *reduce*) Wire-Cell generated data into "event"-level data

- Might remove (or at least reduce) the need for event visitors

- It's conceivable that the DUNE framework and Wire-Cell could use the same data-flow graph.
  - Could result in efficiency improvements (to be demonstrated)
  - Requires consistency in configuration patterns

Conclusion

- The framework/Wire-Cell interface is non-trivial, but it works.
- It could be improved by addressing the impedance mismatch induced by differences in configuration patterns, data design, and data-level granularities.
- The granularity problem should be alleviated by improvements to the framework (a la Meld).
- We should work toward resolving the other issues.

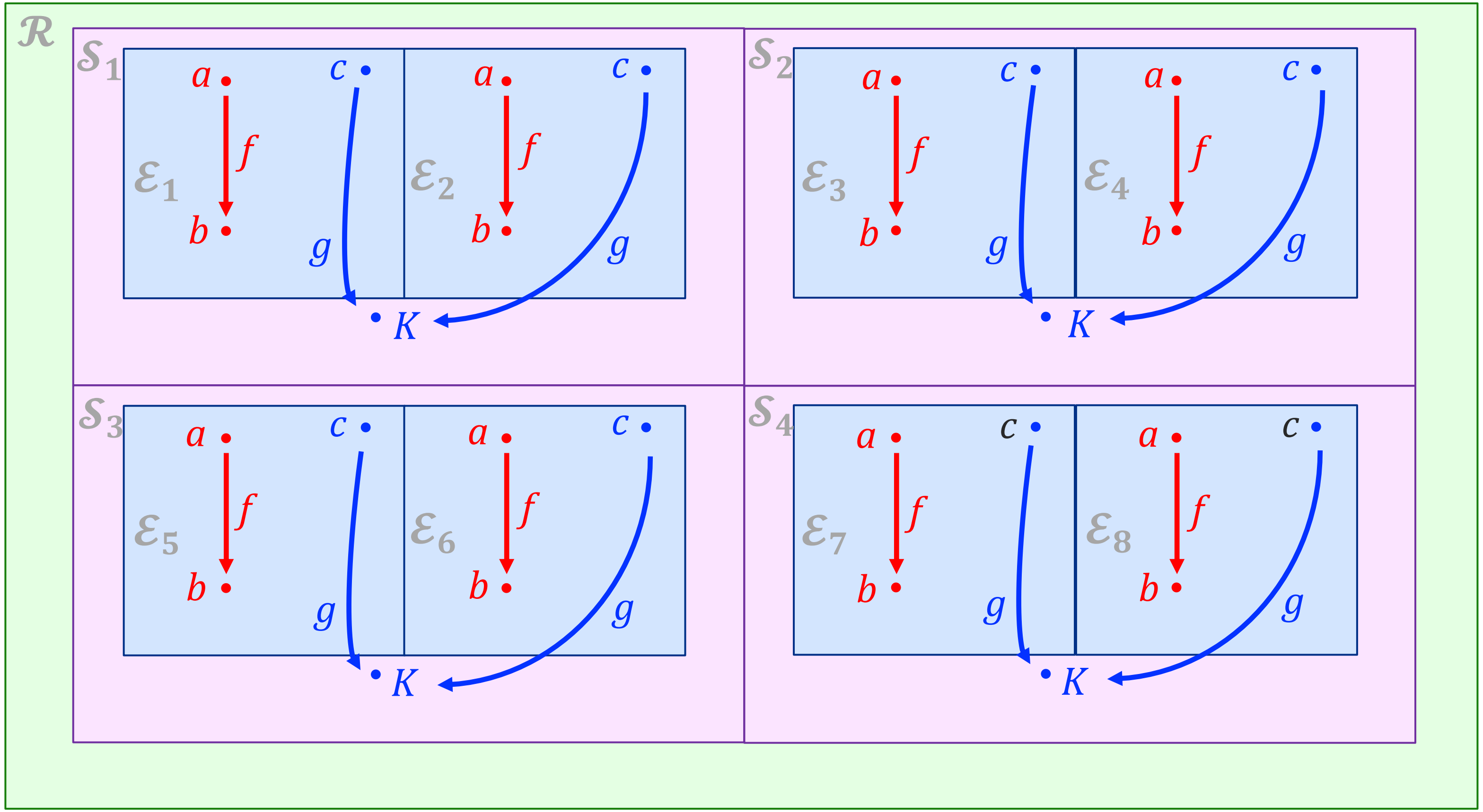
References

- https://en.wikipedia.org/wiki/Software_framework
- https://en.wikipedia.org/wiki/Interface_(computing)
- https://byjus.com/question-answer/what-is-the-function-of-capillary-blood-vessels/
- https://github.com/knoepfel/meld
- https://ffipractitioner.org/wp-content/uploads/2014/11/ffi-working-together.jpg

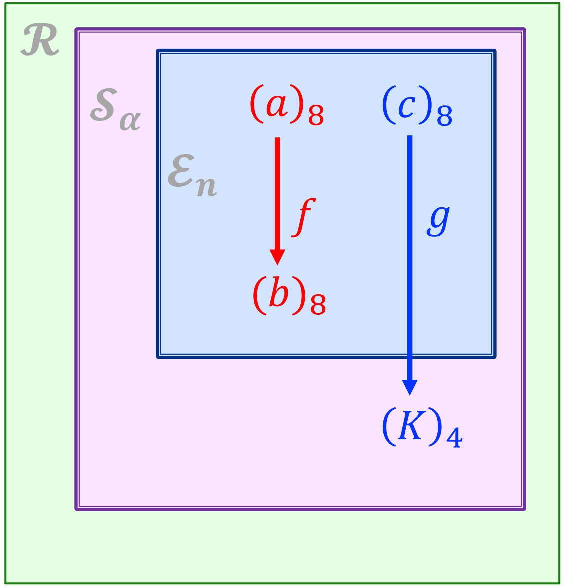
Fermilab

Backup slides

We want to recast this type of procedural processing…



… as a graph of higher-order functions.



Higher-order functions operate on sequences—e.g. $(a)_8$.

Higher-order function	Signature	User function
Transform (Map)	$f * (a)_n \rightarrow (b)_n$	$f(a) \rightarrow b$
Filter	$f \lhd (a)_n \rightarrow (a)_m$	$f(a) \rightarrow \text{Boolean}$
Monitor	$f \lhd (a)_n \rightarrow (\,)$	$f(a) \rightarrow \text{Void}$
Reduction (Fold)	$g\,^n/(c)_n \rightarrow K$	$g(K, c) \rightarrow K$
Splitter (Unfold)	$f\,_n \backslash a \rightarrow (d)_n$	$f_n(a) \rightarrow (d)_n$

The above does not specify any implementation.

- For memory-bound applications, the graph could be processed one event at a time.
- For some algorithms, data could be batched across many events and offloaded to a GPU.
- More exotic combinations.