Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

Meld: Exploring the feasibility of a framework-less framework

Kyle J. Knoepfel

11 May 2023

26th International Conference on Computing in High Energy & Nuclear Physics

What type of framework does DUNE need?

What type of framework does DUNE need?

- **What DUNE has:**

  DUNE's current framework (art) originates from a collider-physics experiment, steeped in event-based concepts.

- **But:**

  The "event" is not always a helpful concept for neutrino experiments.

- **What DUNE needs…**

What type of framework does DUNE need?

- **What DUNE has:**

  DUNE's current framework (art) originates from a collider-physics experiment, steeped in event-based concepts.

- **But:**

  The "event" is not always a helpful concept for neutrino experiments.

- **What DUNE needs…**

DUNE Offline Computing

Conceptual Design Report

https://doi.org/10.48550/arXiv.2210.15665

Fermilab

Some DUNE framework requirements (paraphrased)

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

🙂 **Fine, assuming it's a requirement for those writing algorithms.**

Fermilab

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

🙂 **Fine, assuming it's a requirement for those writing algorithms.**

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

Fermilab

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

🙂 **Fine, assuming it's a requirement for those writing algorithms.**

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

😐 **Okay, tricky but probably doable.**

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

🙂 **Fine, assuming it's a requirement for those writing algorithms.**

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

😐 **Okay, tricky but probably doable.**

The framework should support "sliding event windows" to provide "edge effect" coverage for extended time readouts during supernovae events.

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

🙂 **Fine, assuming it's a requirement for those writing algorithms.**

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

😐 **Okay, tricky but probably doable.**

The framework should support "sliding event windows" to provide "edge effect" coverage for extended time readouts during supernovae events.

☹️ **Cannot take advantage of statistical independence of events, memory issues, etc.**

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

🙂 **Fine, assuming it's a requirement for those writing algorithms.**

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

😐 **Okay, tricky but probably doable.**

The framework should support "sliding event windows" to provide "edge effect" coverage for extended time readouts during supernovae events.

☹️ **Cannot take advantage of statistical independence of events, memory issues, etc.**

The framework should make minimal assumptions about the data model.

Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

🙂 **Fine, assuming it's a requirement for those writing algorithms.**

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

😐 **Okay, tricky but probably doable.**

The framework should support "sliding event windows" to provide "edge effect" coverage for extended time readouts during supernovae events.

☹️ **Cannot take advantage of statistical independence of events, memory issues, etc.**

The framework should make minimal assumptions about the data model.

😧 ***That sounds like a framework-less framework...***

But is it so crazy?

- How many of art’s assumptions can be relaxed/removed to meet DUNE’s needs?

But is it so crazy?

- How many of art's assumptions can be relaxed/removed to meet DUNE's needs?
- Asking this question has resulted in a 2-year project called **Meld**, a laboratory-directed R&D project based at Fermilab.
- The goal is to explore options, not necessarily to provide software.

But is it so crazy?

- How many of art's assumptions can be relaxed/removed to meet DUNE's needs?
- Asking this question has resulted in a 2-year project called **Meld**, a laboratory-directed R&D project based at Fermilab.
- The goal is to explore options, not necessarily to provide software.

- Meld has been heavily influenced by:
  - Regular discussions with DUNE experts
  - Existing framework capabilities and limitations
  - Functional programming (e.g. Haskell)
  - Mathematics (set, graph, and category theory)

But is it so crazy?

- How many of art's assumptions can be relaxed/removed to meet DUNE's needs?
- Asking this question has resulted in a 2-year project called **Meld**, a laboratory-directed R&D project based at Fermilab.
- The goal is to explore options, not necessarily to provide software.

- Meld has been heavily influenced by:
  - Regular discussions with DUNE experts
  - Existing framework capabilities and limitations
  - Functional programming (e.g. Haskell)
  - Mathematics (set, graph, and category theory)

Prerequisites

Support user-provided algorithms written in C++20 or newer

Design for concurrency

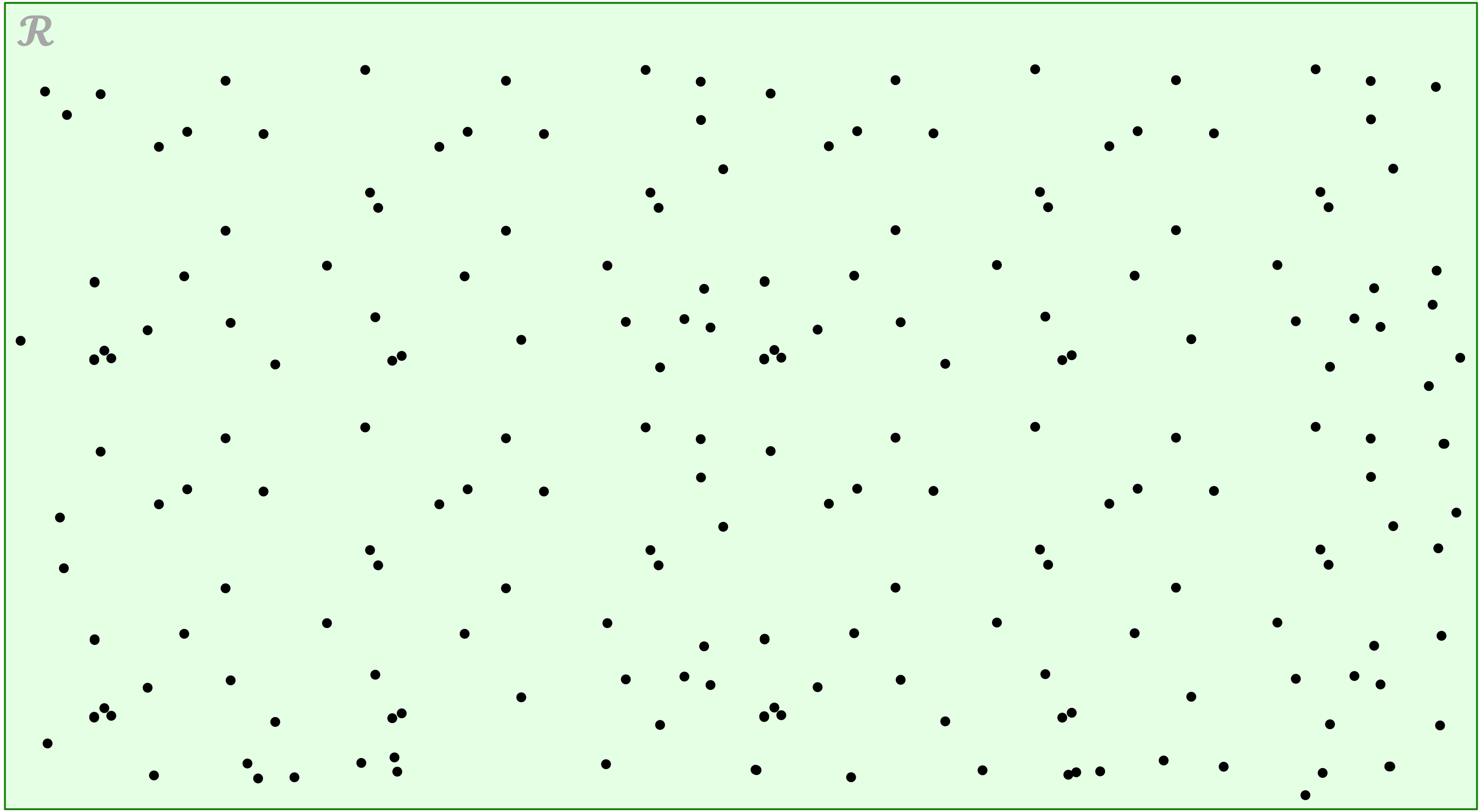
Favor community-provided software

Looking at the data

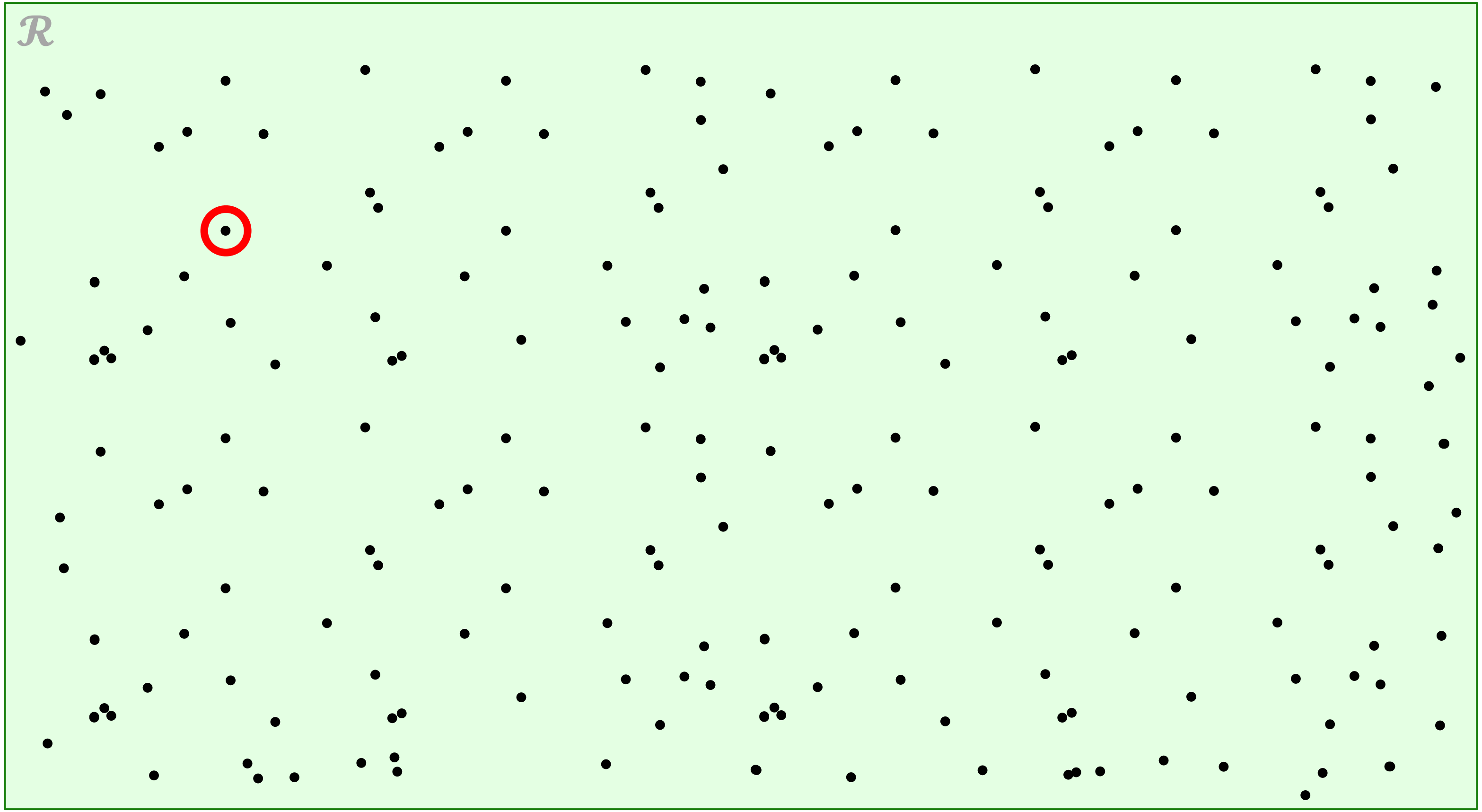
The following discussion describes a logical organization of data.

It does not imply a specific in-memory representation of data.

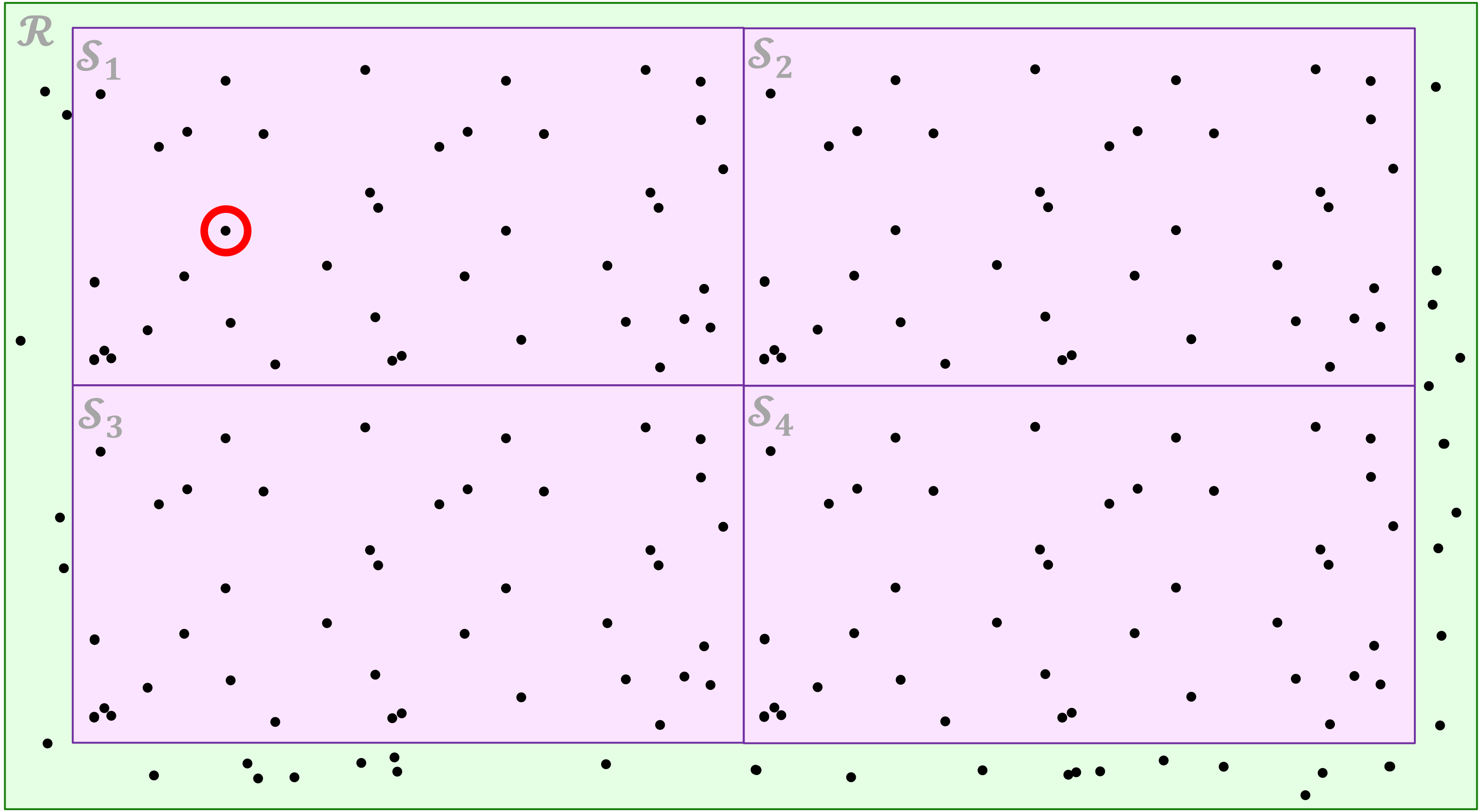
Looking at the data (set)



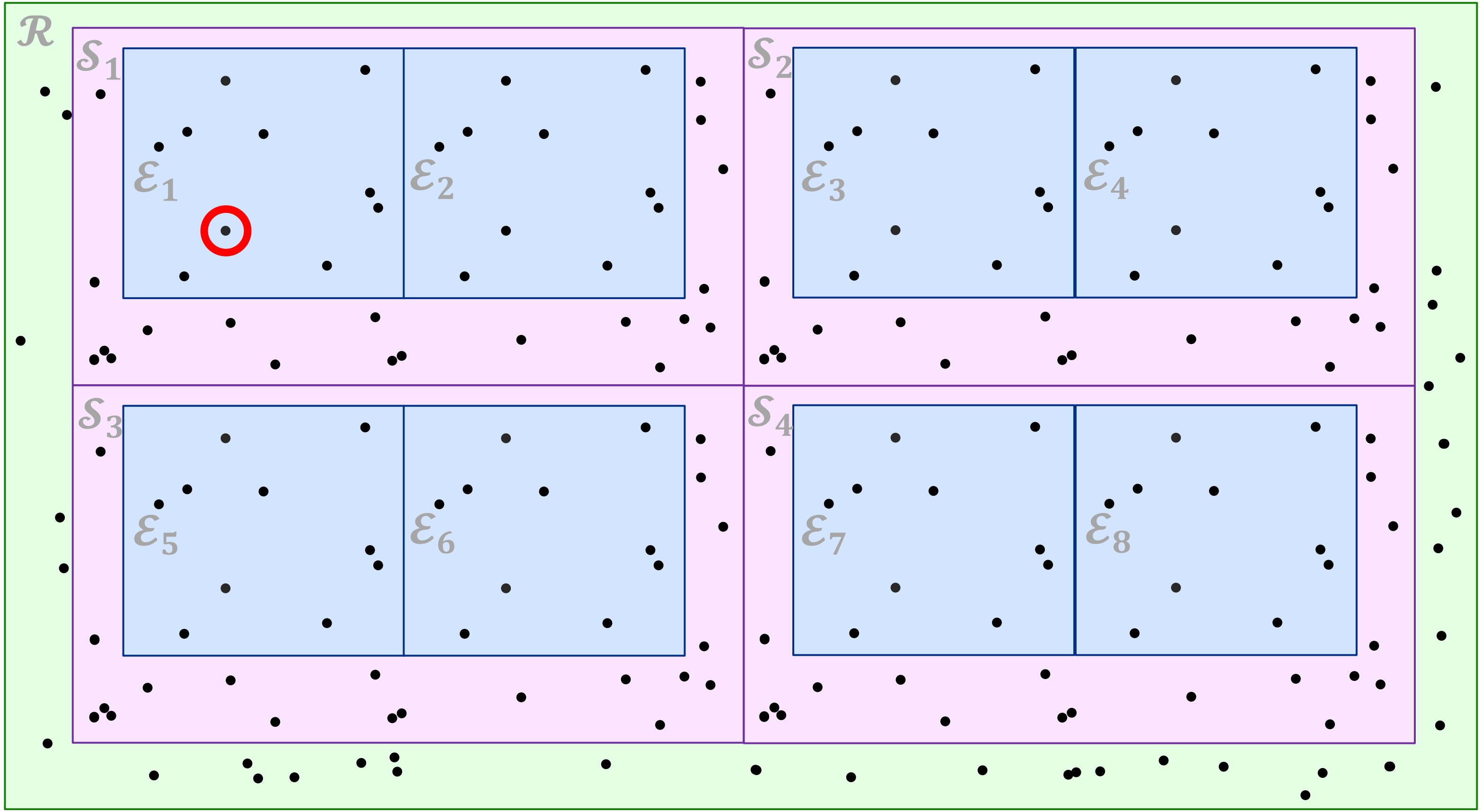
Looking at the data (products)



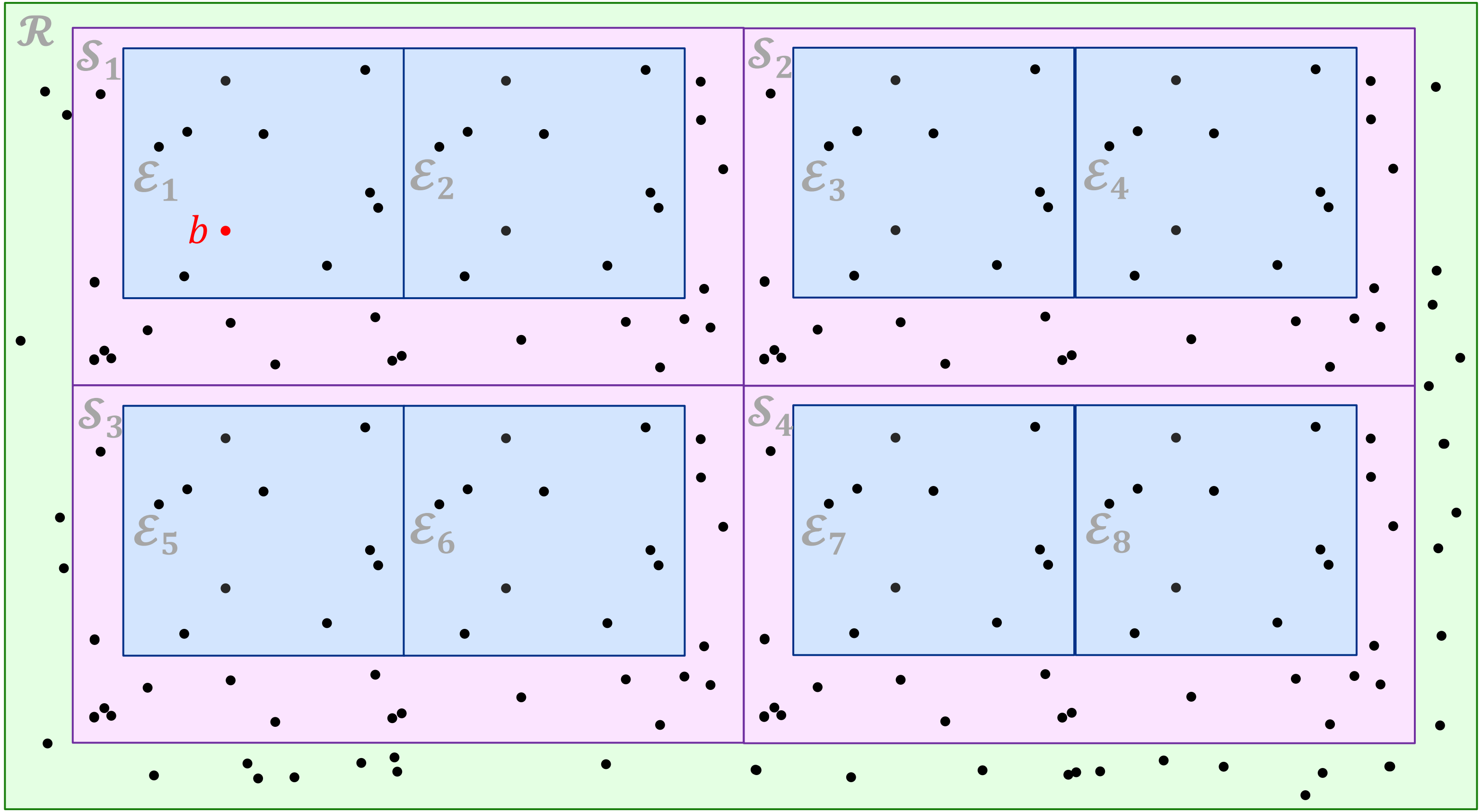
Looking at the data (products)



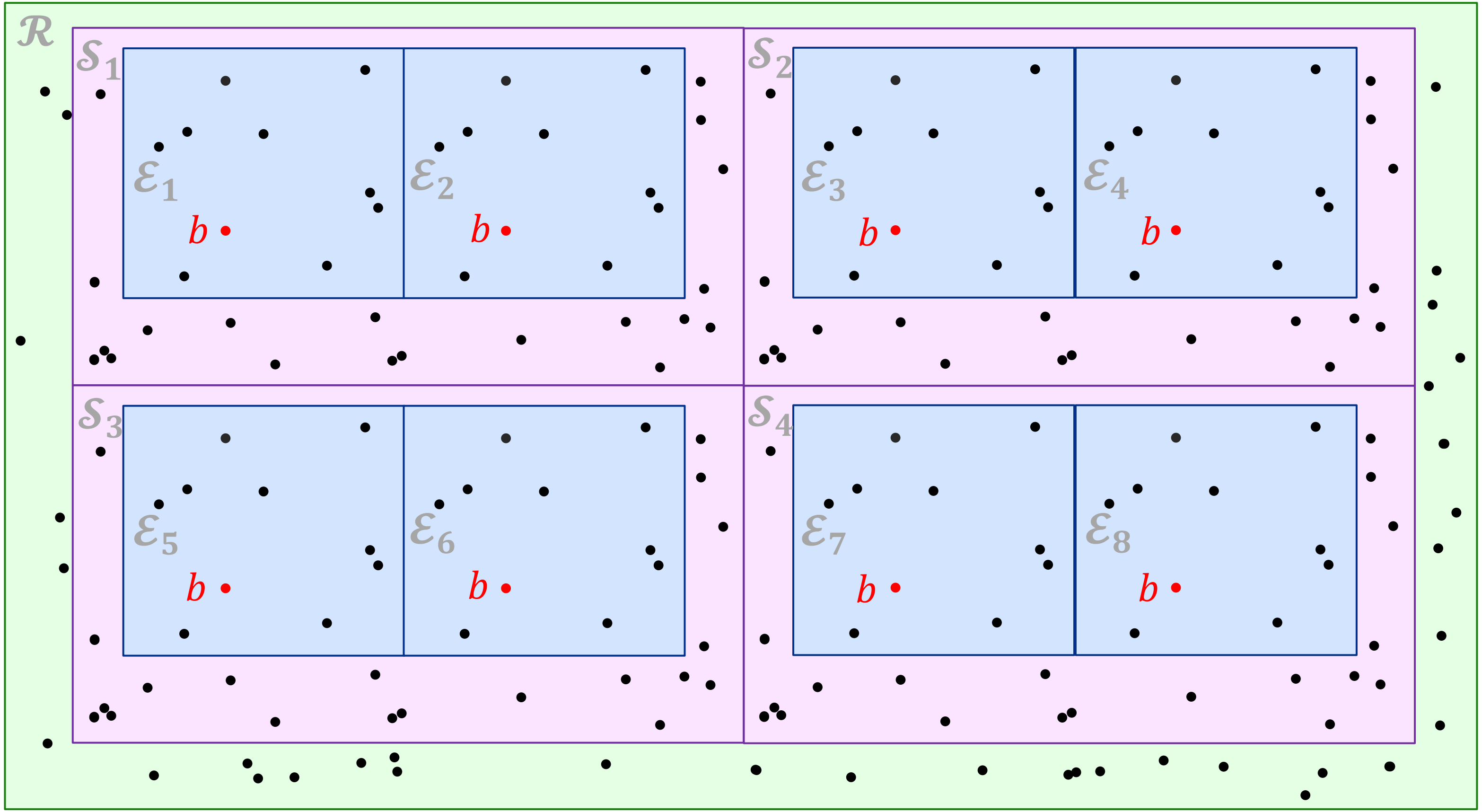
Looking at the data (products)



Looking at the data (products)

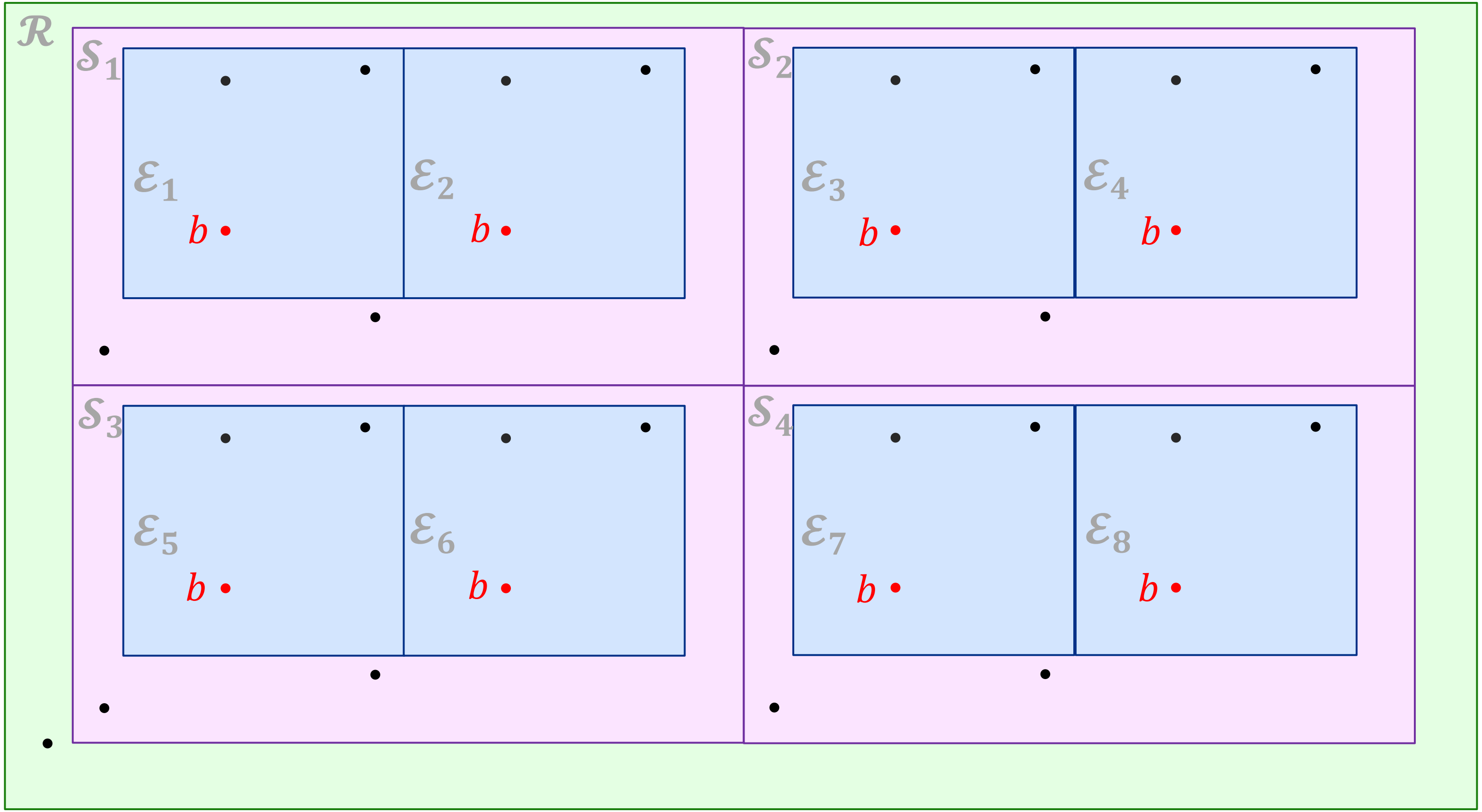


Looking at the data (products)

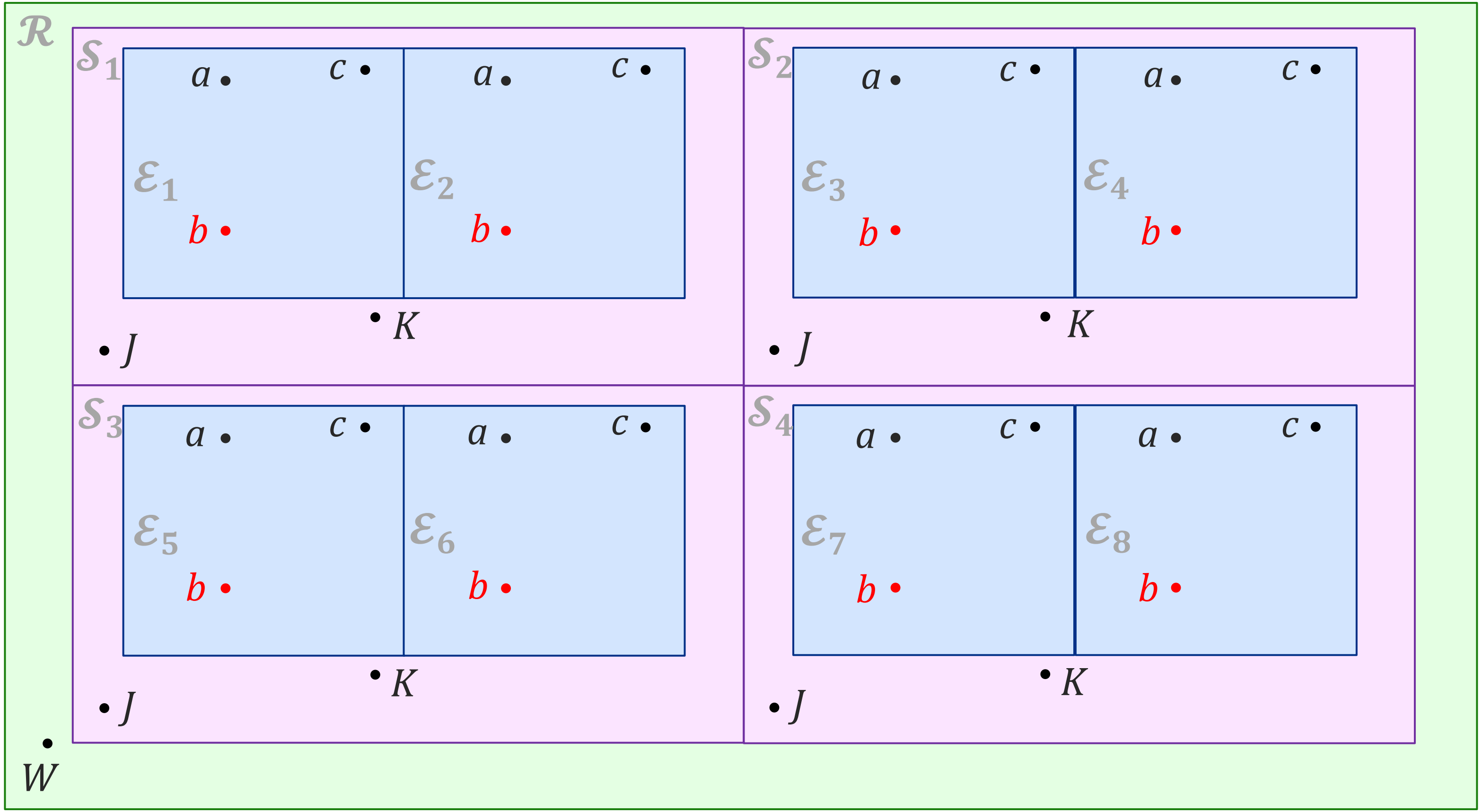


Fermilab

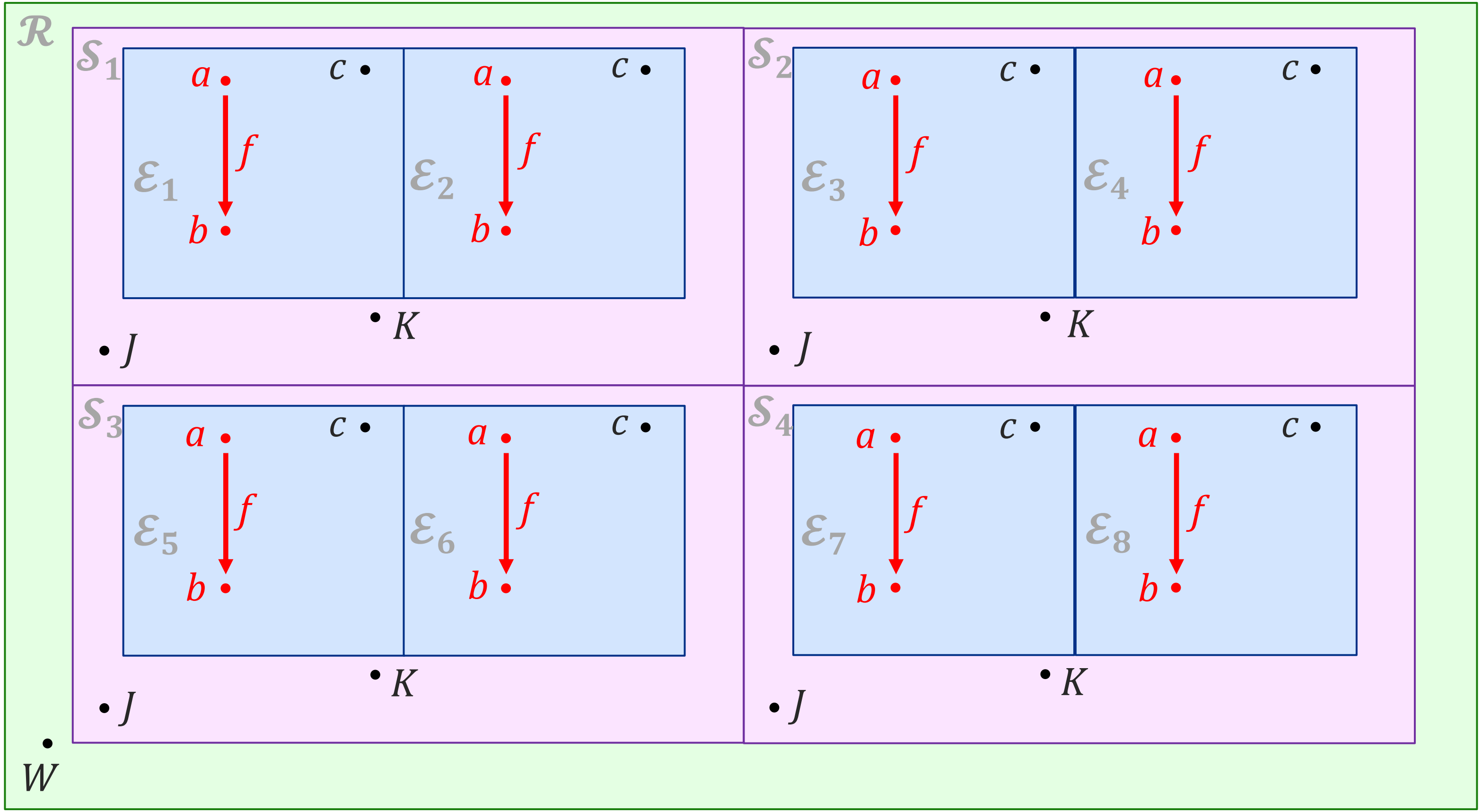
Looking at the data (products)



Looking at the data (products)

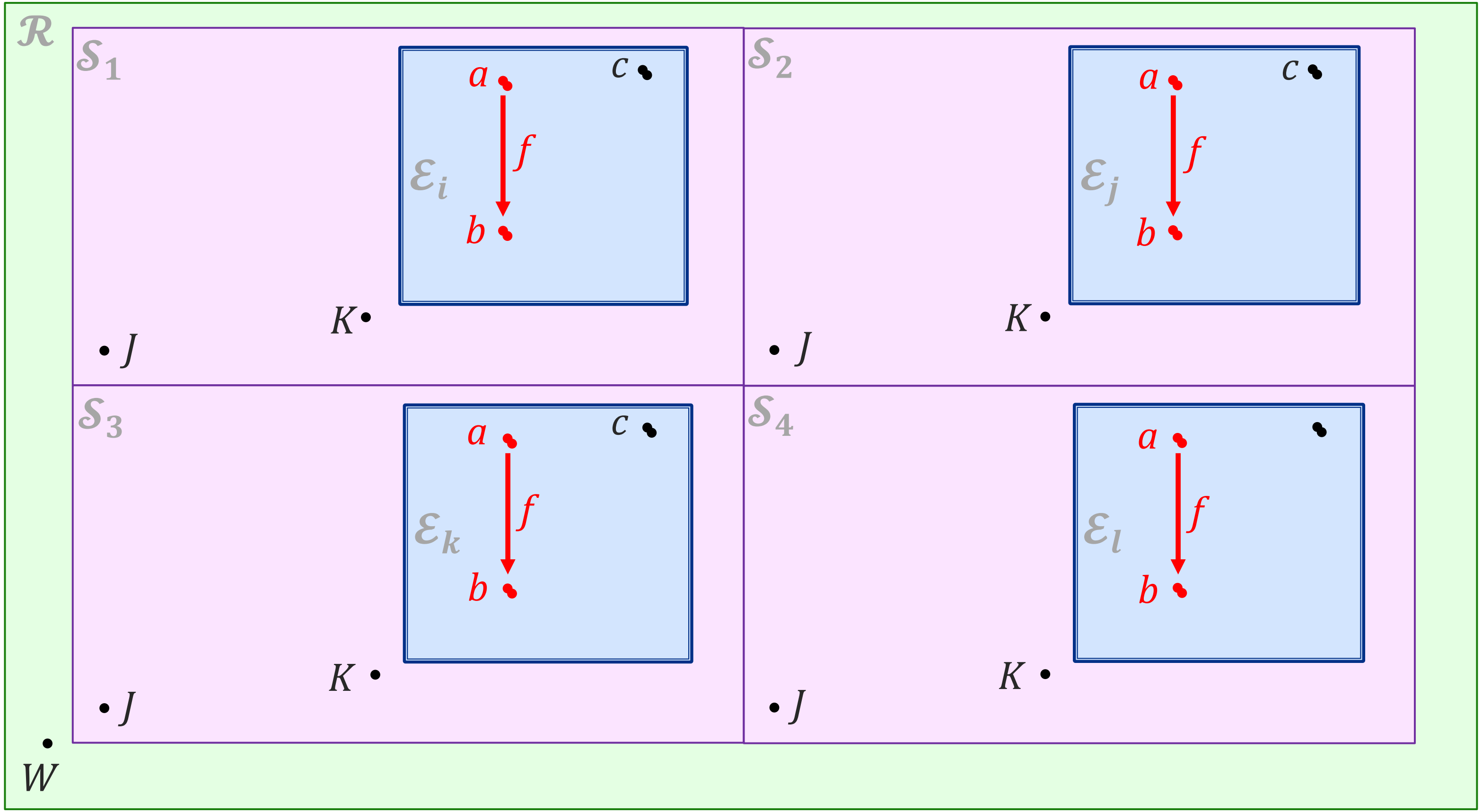


Looking at the data (product mappings)

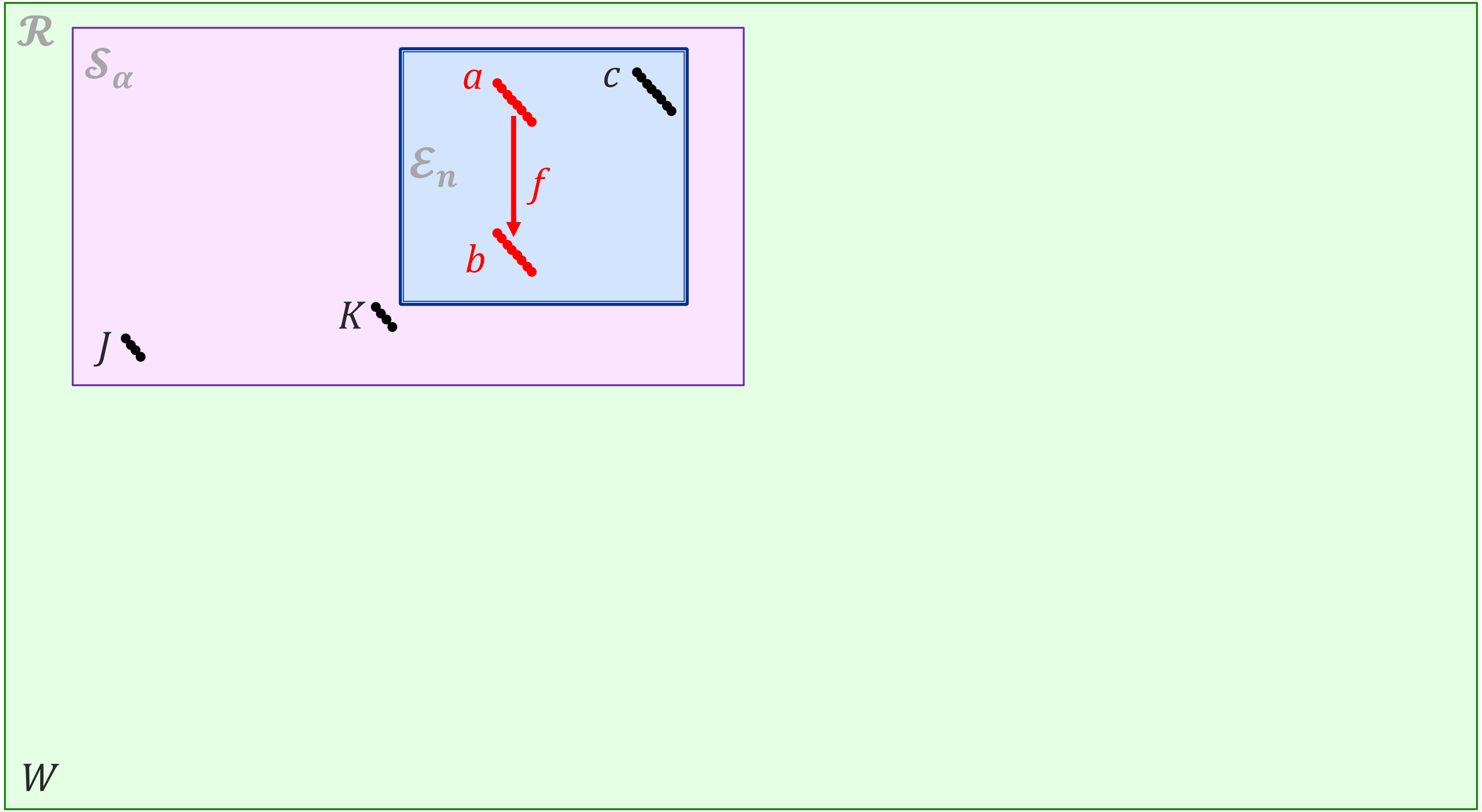


Fermilab

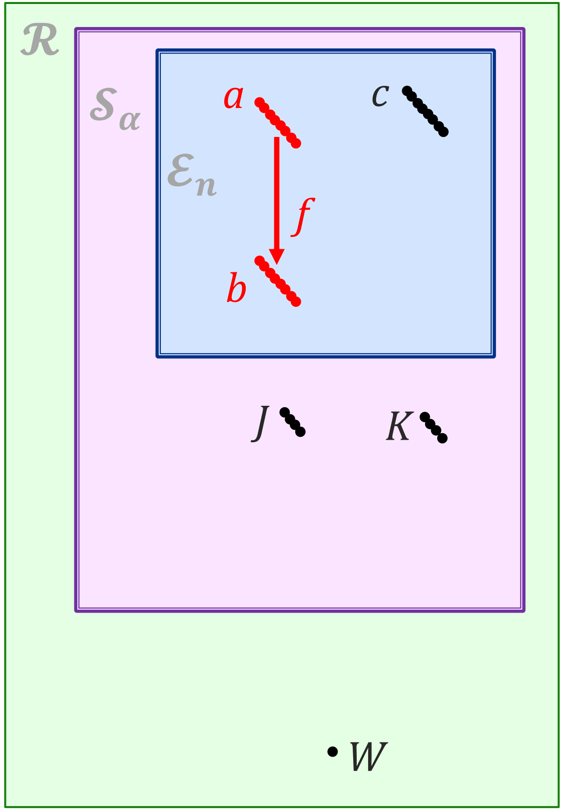
Looking at the data (product sequences)



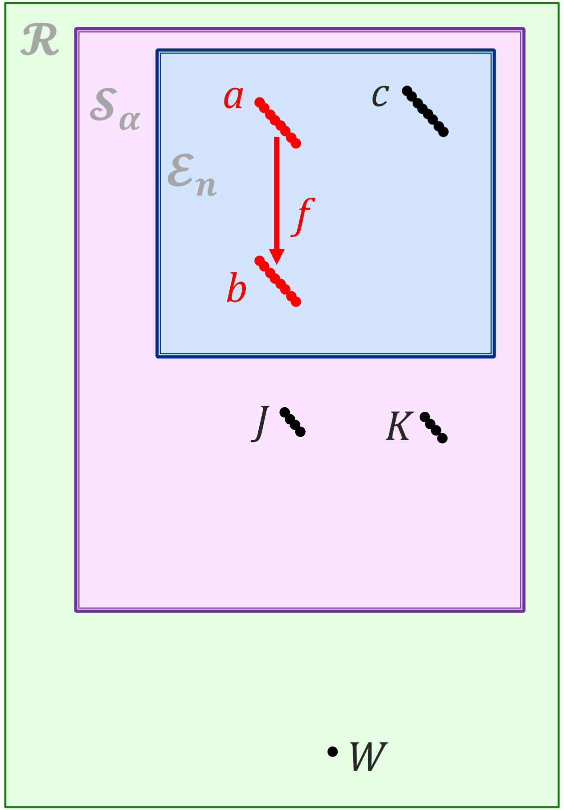
Looking at the data (product sequences)



Looking at the data (product sequences)



Looking at the data (product sequences)

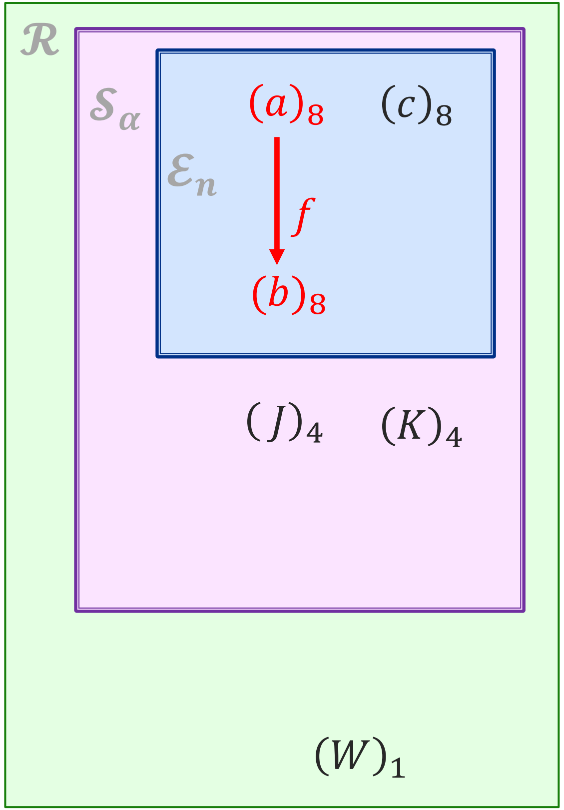


We can make the following replacement (e.g.):

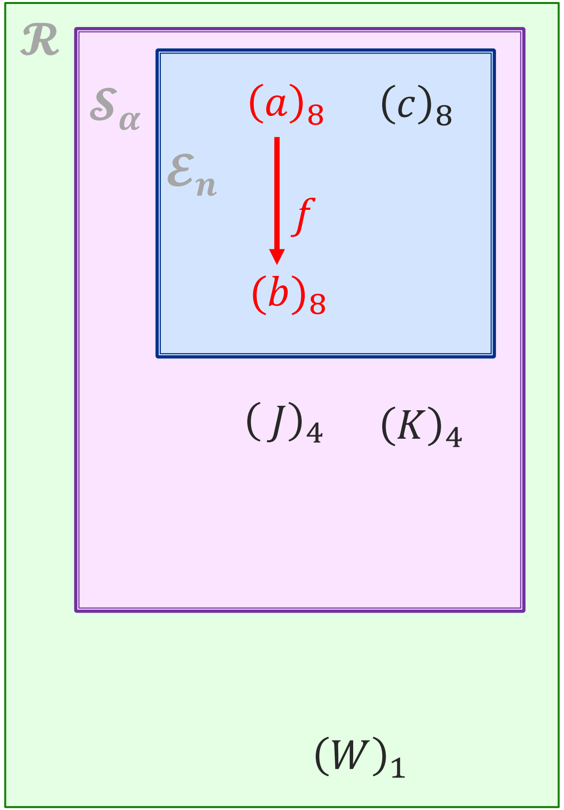
$$= (c)_8$$

depicting the data products labeled c from 8 events as a sequence.

Looking at the data (product sequences)

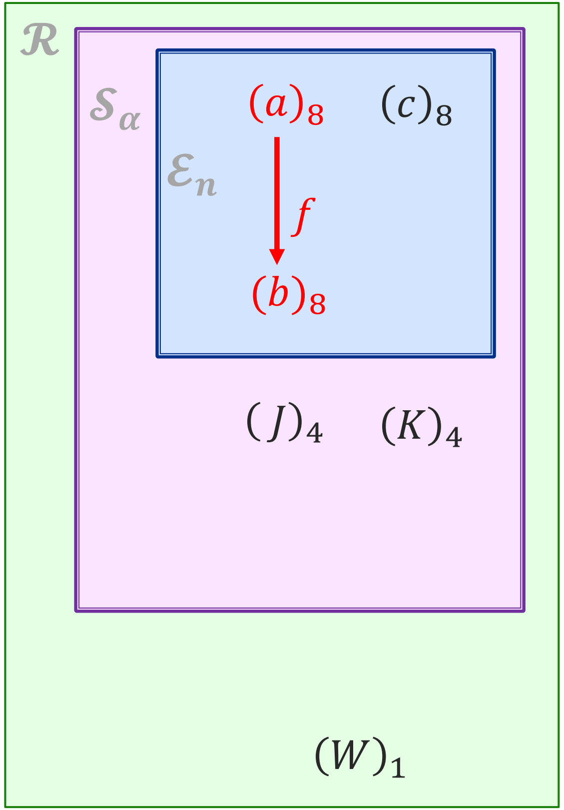


What type of things are we dealing with?



- An operation that converts a sequence of elements $(a)_8$ to a sequence of elements $(b)_8$ *of the same length* using a function f:

What type of things are we dealing with?

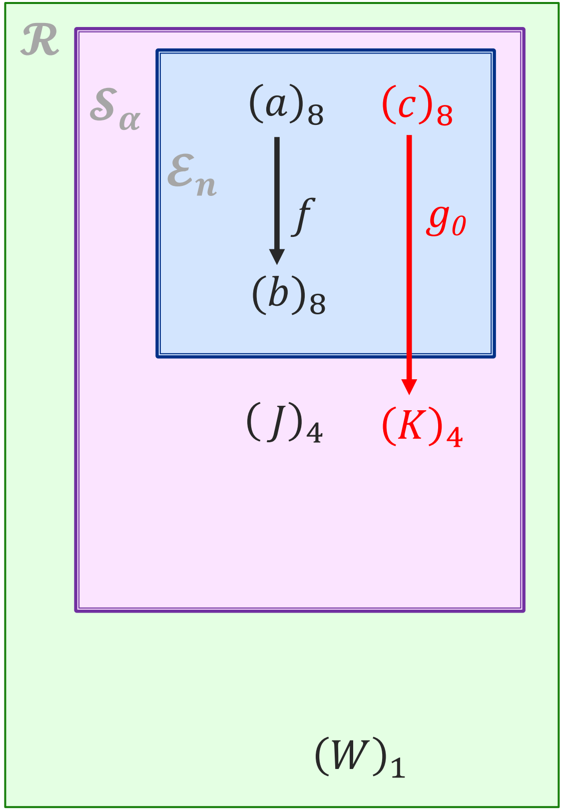


- An operation that converts a sequence of elements $(a)_8$ to a sequence of elements $(b)_8$ *of the same length* using a function f:

This is a map or transform.

Fermilab

What type of things are we dealing with?

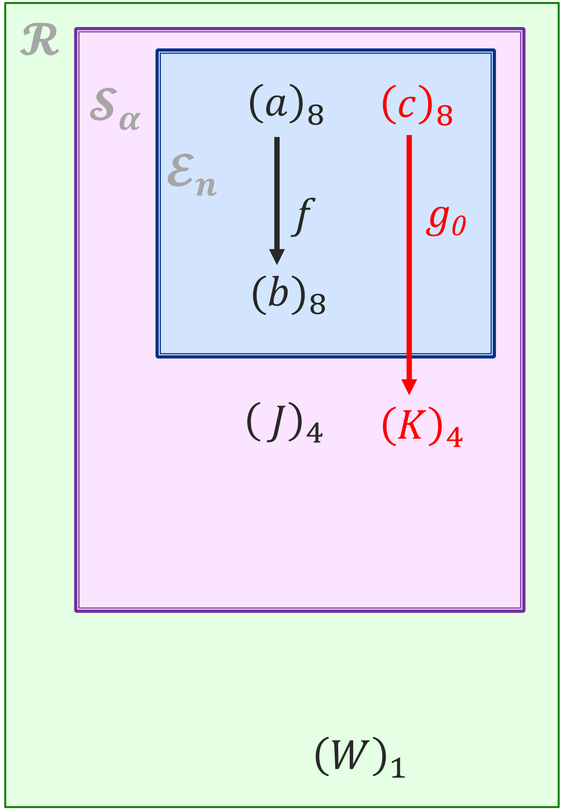


- An operation that converts a sequence of elements $(a)_8$ to a sequence of elements $(b)_8$ *of the same length* using a function f:

  This is a map or transform.

- An operation that converts a sequence of elements $(c)_8$ to a shorter sequence of elements $(K)_4$ at a higher level of nesting, using a function g_0:

What type of things are we dealing with?



- An operation that converts a sequence of elements $(a)_8$ to a sequence of elements $(b)_8$ *of the same length* using a function f:

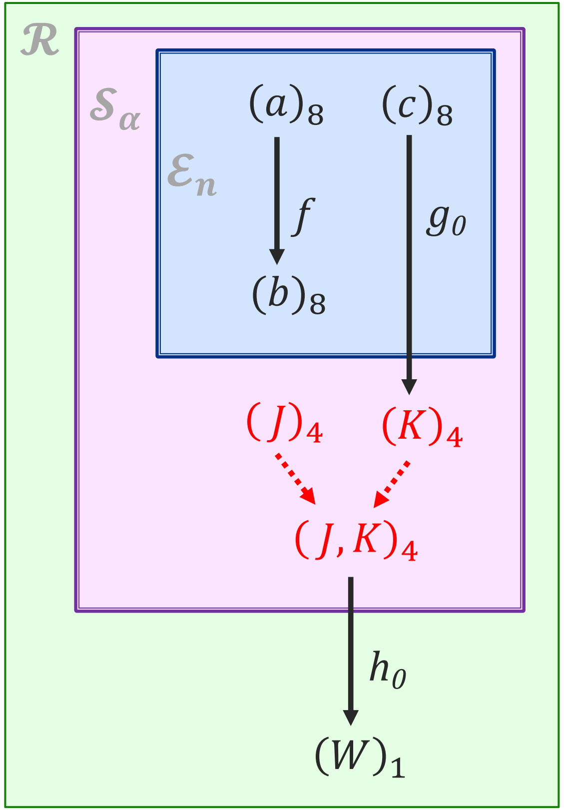
  This is a map or transform.

- An operation that converts a sequence of elements $(c)_8$ to a shorter sequence of elements $(K)_4$ at a higher level of nesting, using a function g_0:

  This is a fold or reduction.

Fermilab

What type of things are we dealing with?



- An operation that converts a sequence of elements $(a)_8$ to a sequence of elements $(b)_8$ *of the same length* using a function f:

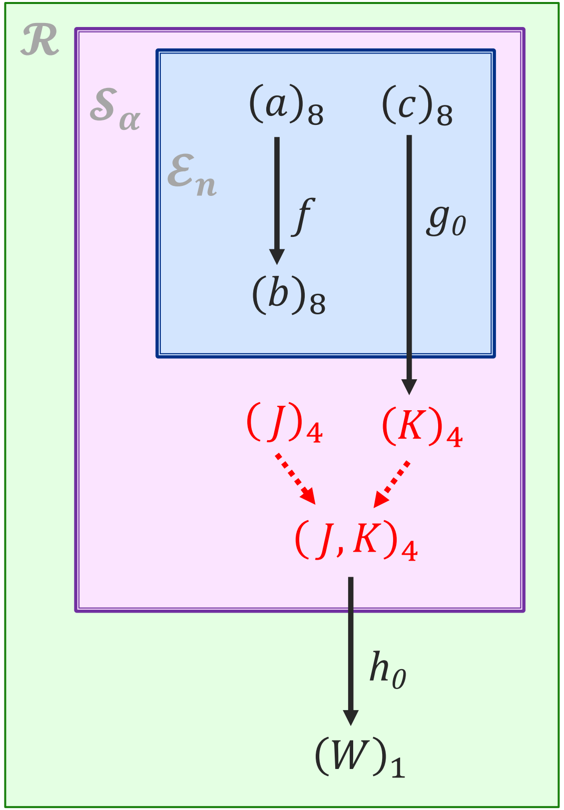
  This is a map or transform.

- An operation that converts a sequence of elements $(c)_8$ to a shorter sequence of elements $(K)_4$ at a higher level of nesting, using a function g_0:

  This is a fold or reduction.

- An operation that pairs element of two sequences $(J)_4$ and $(K)_4$ into one sequence $(J, K)_4$:

What type of things are we dealing with?



- An operation that converts a sequence of elements $(a)_8$ to a sequence of elements $(b)_8$ *of the same length* using a function f:

  This is a map or transform.

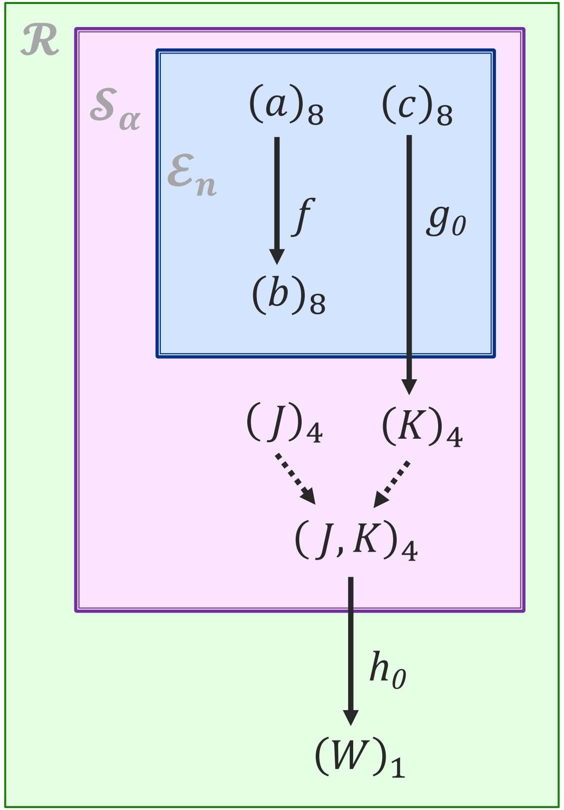
- An operation that converts a sequence of elements $(c)_8$ to a shorter sequence of elements $(K)_4$ at a higher level of nesting, using a function g_0:

  This is a fold or reduction.

- An operation that pairs element of two sequences $(J)_4$ and $(K)_4$ into one sequence $(J,K)_4$:

  This is a zip.

What type of things are we dealing with?



- An operation that converts a sequence of elements $(a)_8$ to a sequence of elements $(b)_8$ *of the same length* using a function f:

  This is a map or transform.

- An o … elements $(c)_8$ … $)_4$ at a highe …

  This is a fold or reduction.

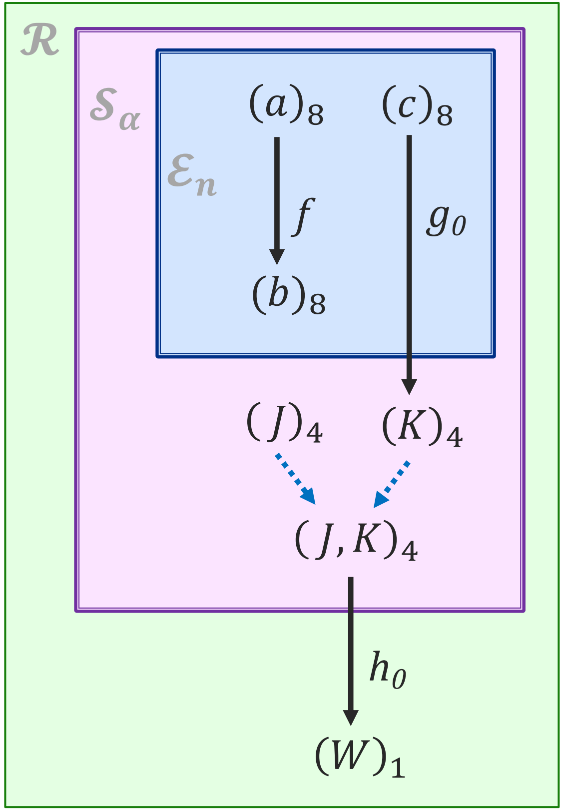
- An operation that pairs element of two sequences $(J)_4$ and $(K)_4$ into one sequence $(J,K)_4$:

  This is a zip.

These have to do with higher-order functions.

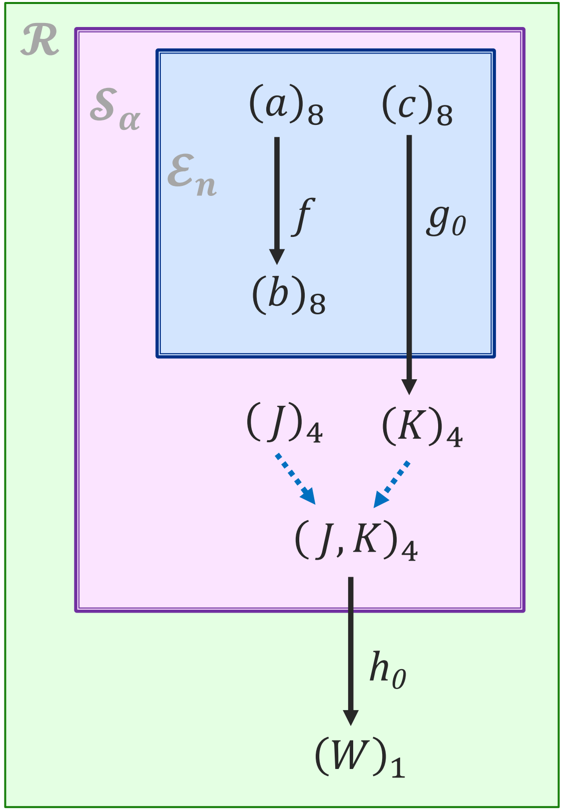
Fermilab

Graph of data-product sequences



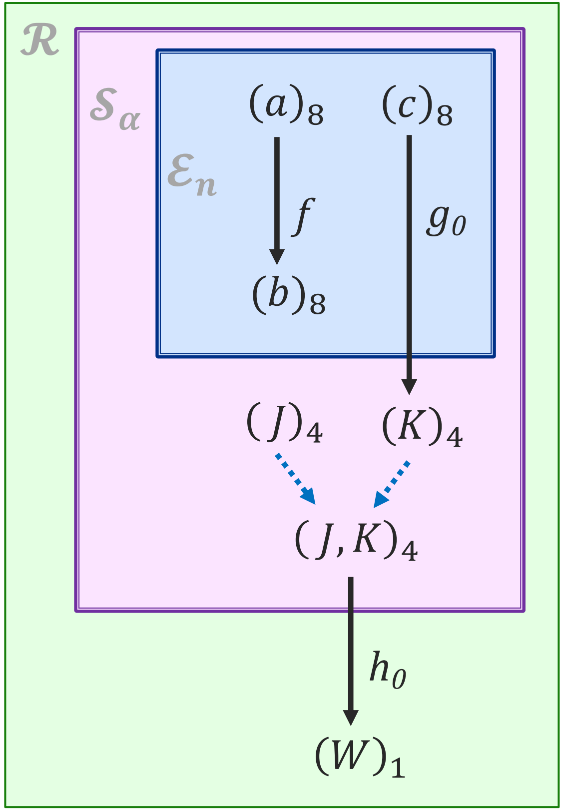
View	Nodes	Edges	
Data-centric	Data products	Mappings	*This work*

Graph of data-product sequences



View	Nodes	Edges	
Data-centric	**Data products**	**Mappings**	*This work*
Map-centric	Mappings	Data products	*More common*

Graph of data-product sequences



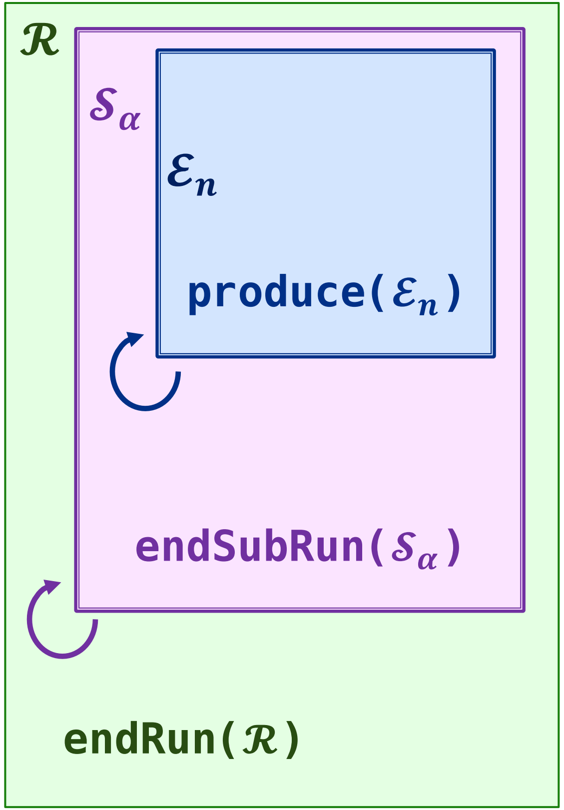
View	Nodes	Edges	
Data-centric	**Data products**	**Mappings**	*This work*
Map-centric	Mappings	Data products	*More common*

The user specifications are the same with either view:

- Which data products to process
- The data set(s) that contain those products (event, etc.)
- Which higher-order function to use (transform, etc.)
- Which user-defined function to serve as the operation to the higher-order function.
- Allowed concurrency of each function.

The focus is just different.

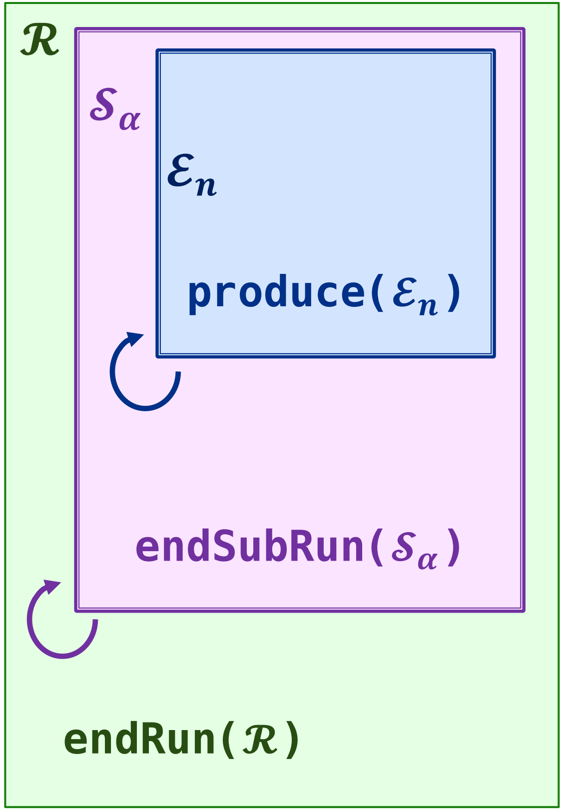
How are data products and their mappings supported now?



With art, users do not transparently interact with data products. They instead:

- Implement functions based on datasets (e.g. event)
- "Open" the dataset to retrieve and insert products

How are data products and their mappings supported now?



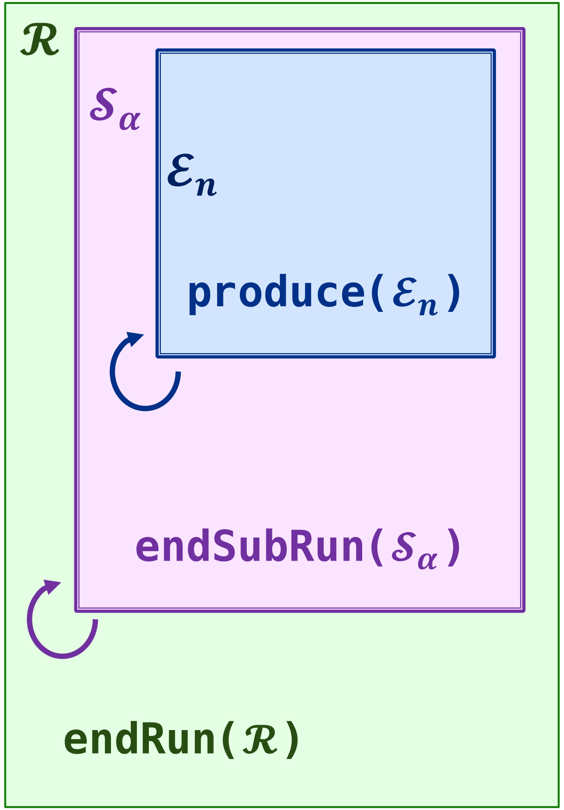
With art, users do not transparently interact with data products. They instead:

- Implement functions based on datasets (e.g. event)
- "Open" the dataset to retrieve and insert products

Some of this is historical and due to:

- The object-oriented nature of the framework.
- Technical limitations of C++ whenever the framework was designed.

How are data products and their mappings supported now?



With art, users do not transparently interact with data products. They instead:

- Implement functions based on datasets (e.g. event)
- "Open" the dataset to retrieve and insert products

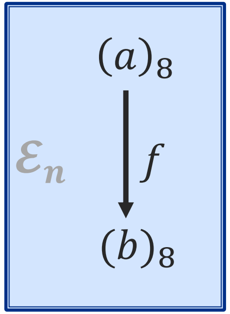
Some of this is historical and due to:

- The object-oriented nature of the framework.
- Technical limitations of C++ whenever the framework was designed.

Results in a lot of software mechanics...

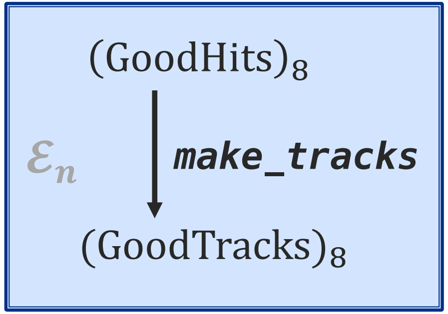
Example

- Create tracks from hits for each event.



Example

- Create tracks from hits for each event.



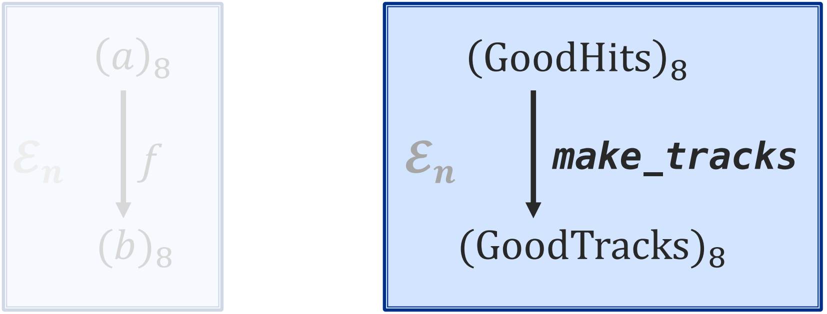

```
Tracks make_tracks(Hits const& hits) { ... }
```


Fermilab

Example

art

- Create tracks from hits for each event.

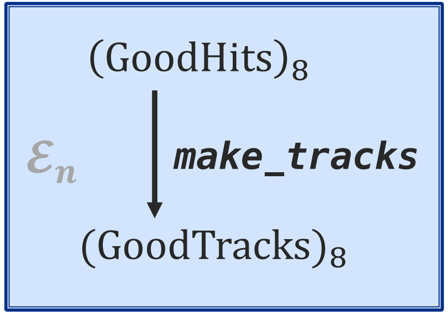



```
Tracks make_tracks(Hits const& hits) { ... }
```

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

namespace {
  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)
```

Example

- Create tracks from hits for each event.




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

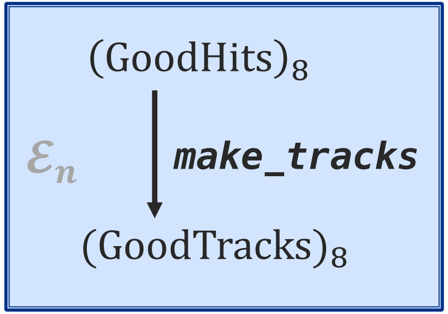
This is just a transform? 😬

art

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

namespace {
  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)
```

Example

- Create tracks from hits for each event.



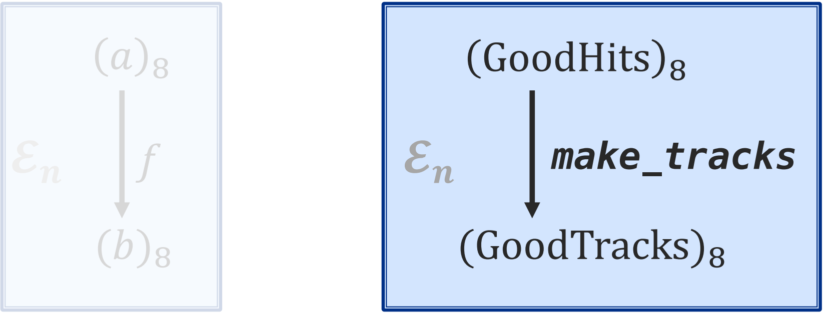

```
Tracks make_tracks(Hits const& hits) { ... }
```

This is just a transform? 😬

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

namespace {
  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)
```

Example

- Create tracks from hits for each event.




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

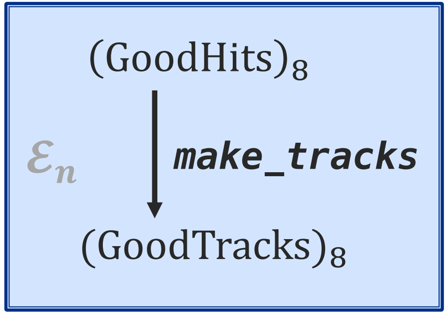
This is just a transform? 😬
Nobody wants this.

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

namespace {
  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)
```

Example

- Create tracks from hits for each event.

Meld




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

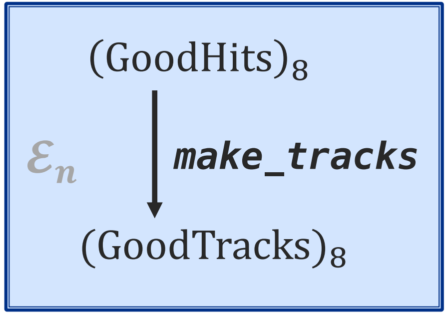
```
#include "meld/module.hpp"

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

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

A better way…

Example

- Create tracks from hits for each event.

Meld




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

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

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

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

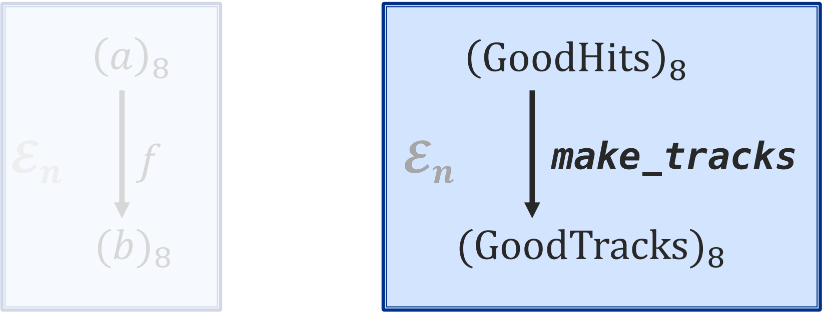
A better way…

- Minimal boilerplate.

Example

- Create tracks from hits for each event.

Meld




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

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

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

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

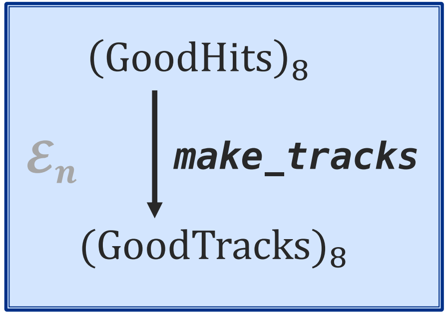
A better way…

- Minimal boilerplate.
- Event is now a label.

Example

- Create tracks from hits for each event.

Meld




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

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

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

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

A better way…

- Minimal boilerplate.
- Event is now a label.
- Higher-order function is now explicit.

Meld implementation

- https://github.com/knoepfel/meld (not even alpha release)
- Implemented using oneTBB's flow graph

Supported construct	User function	
Transform (Map)	$f(\boldsymbol{a}) \rightarrow \boldsymbol{b}$	*Standard data-processing idioms*
Filter	$f(\boldsymbol{a}) \rightarrow \text{Boolean}$	
Monitor	$f(\boldsymbol{a}) \rightarrow \text{Void}$	
Reduction (Fold)	$f_c(\boldsymbol{a}) \rightarrow \boldsymbol{c}$	***For splitting and then combining events***
Splitter (Unfold)	$f_n(\boldsymbol{a}) \rightarrow (\boldsymbol{d})_n$	
Zip	—	***For combining arguments to user functions***
Sliding window	—	*To do:* ***For sliding over adjacent events***

Sample hierarchies tested by Meld

```
[info] Number of worker threads: 12
[info] Processed levels:

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

[info] CPU efficiency: 259.55%
[info] Max. RSS: 6.205 MB
```

art-based hierarchy

Performance numbers are preliminary

Fermilab

Sample hierarchies tested by Meld

```
[info] Number of worker threads: 12
[info] Processed levels:

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

[info] CPU efficiency: 259
[info] Max. RSS: 6.205 MB
```

art-based hierarc

Performance numbers are preliminary

```
[info] Number of worker threads: 12
[info] Processed levels:

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

[info] CPU efficiency: 230.81%
[info] Max. RSS: 6.136 MB
```

Non-trivial hierarchy

Sample hierarchies tested by Meld

Performance numbers are preliminary

```
[info] Number of worker threads: 12
[info] Processed levels:

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

[info] CPU efficiency: 259
[info] Max. RSS: 6.205 MB
```

art-based hierarc

```
[info] Number of worker threads: 12
[info] Processed levels:

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

[info] CPU efficiency: 230.81%
[info] Max. RSS: 6.136 MB
```

Non-trivial hierarchy

```
[info] Number of worker threads: 12
[info] Processed levels:

  job
   │
   └ event: 100000

[info] CPU efficiency: 882.50%
[info] Max. RSS: 16.527 MB
```

Flat hierarchy

Summary

> ***"Ways change, Stil."*** —Paul from *Dune* by Frank Herbert

- Supporting DUNE's framework needs suggests rethinking framework concepts.

Summary

> ***"Ways change, Stil."*** —Paul from *Dune* by Frank Herbert

- Supporting DUNE's framework needs suggests rethinking framework concepts.
- Meld seeks to address these needs by considering a framework job as a

    (1) **graph of data products** *connected by*
    (2) **user-provided operations** *of*
    (3) **higher-order functions**.

- It is not a framework-less framework, but it *is* less framework coupling.
- Preliminary work indicates this is a productive avenue to pursue.

Summary

"Ways change, Stil." —Paul from *Dune* by Frank Herbert

- Supporting DUNE's framework needs suggests rethinking framework concepts.
- Meld seeks to address these needs by considering a framework job as a

  (1) graph of data products *connected by*
  (2) user-provided operations *of*
  (3) higher-order functions.

- It is not a framework-less framework, but it *is* less framework coupling.
- Preliminary work indicates this is a productive avenue to pursue.

Thank you for your time and attention.

Backup slides

Accessing provenance information

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

namespace {
-  Tracks make_tracks(Hits const& hits) { ... }
+  Tracks make_tracks(meld::handle<Hits> hits) { ... }
}

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


Fermilab

Class example using lambda expression

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

DEFINE_MODULE(m, config)
{
  auto threshold = config.get<unsigned int>("threshold");
  m.with([threshold](Hits const& hits) { return hits.size() > threshold; })
   .filter("GoodHits").in_each("Event")
   .using_concurrency(unlimited);
}
```

Class example registering two member functions

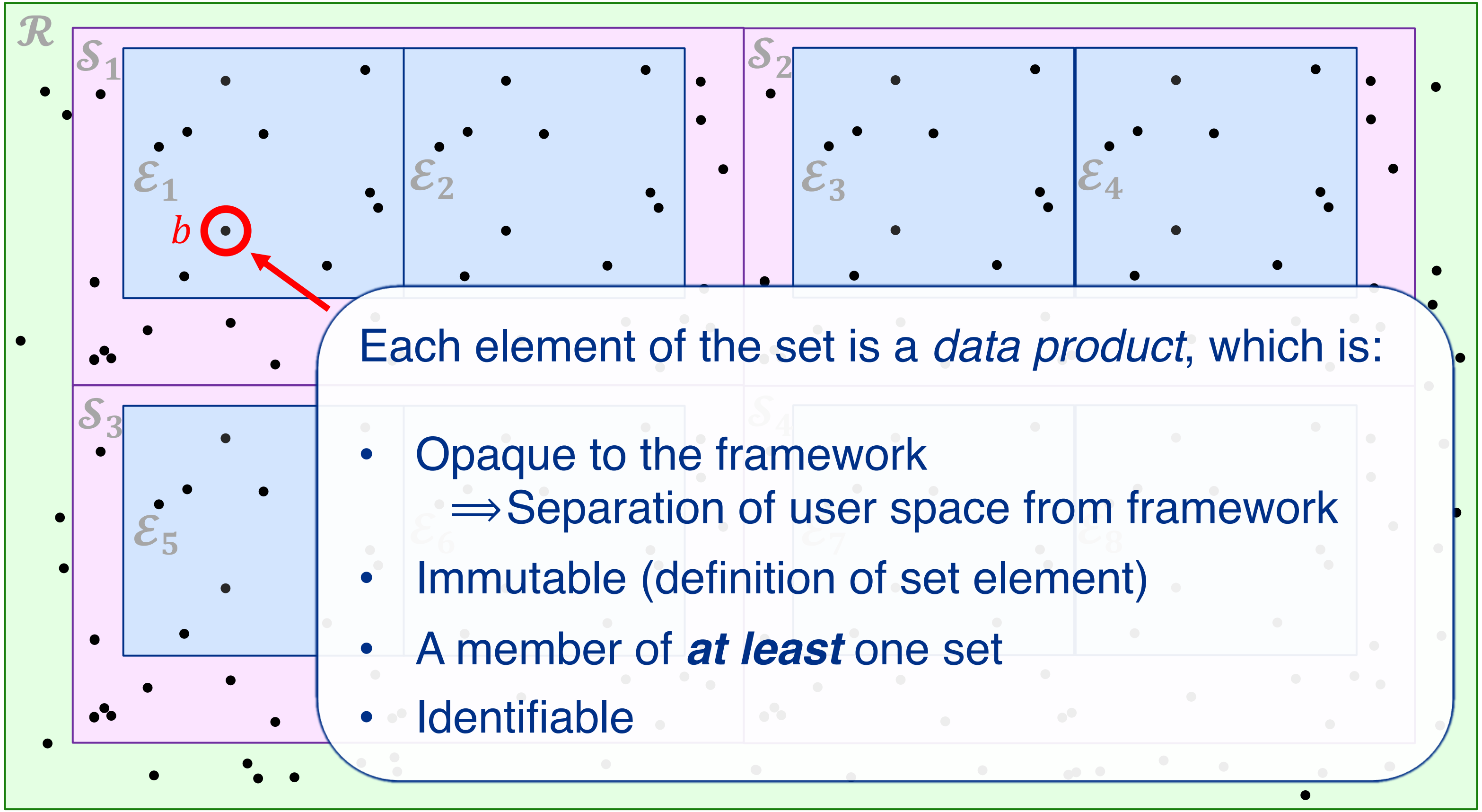
```cpp
#include "meld/module.hpp"

class Selector {
public:
  Selector(unsigned int n) : threshold{n} {}
  bool gt(Hits const& hits) const { return hits.size() > threshold; }
  bool le(Hits const& hits) const { return !gt(hits); }

private:
  unsigned int threshold;
};

DEFINE_MODULE(m, config)
{
  auto threshold = config.get<unsigned int>("threshold");
  auto bound_m = m.make<Selector>(threshold);
  bound_m.with(&Selector::gt).filter("GoodHits").in_each("Event");
  bound_m.with(&Selector::le).filter("GoodHits").in_each("Event");
}
```

Reduction example

```
class MyAccumulator : public art::EDProducer {
public:
  MyAccumulator(ParameterSet const&)
  {
    produces<int, art::InSubRun>("sum");
  }

  void produce(art::Event&) override
  {
    ++counter_;
  }

  void endSubRun(art::SubRun& sr) override
  {
    sr.put(std::make_unique<int>(counter_), "sum");
    counter_ = 0;
  }

private:
  int counter_ = 0;
};

DEFINE_ART_MODULE(MyAccumulator)
```

```
void accumulate(int& counter,
                meld::level_id const&)
{
  ++counter;
}


DEFINE_MODULE(m) {
  m.with(accumulate, 0).for_each("SubRun")
   .reduce("id").in_each("Event")
   .to("sum");
}
```

Looking at the data (products)



Each element of the set is a *data product*, which is:

- Opaque to the framework
  ⟹Separation of user space from framework
- Immutable (definition of set element)
- A member of ***at least*** one set
- Identifiable

Higher-order functions

- We are interested in the mappings of the form:

$$\left\{(\boldsymbol{a})_n \xrightarrow{f} (\boldsymbol{b})_m\right\} \in \mathcal{D}$$

- Each object $\boldsymbol{a}$ corresponds to a tuple of arguments passed to f.
- The signature of f and the value $f(\boldsymbol{a})$, depends on the higher-order function.
- The above mapping happens within a domain $\mathcal{D}$ (e.g. job, run, event).
- Each object $\boldsymbol{a}$ is an element of a subset of the domain $\mathcal{D}$.

Supported higher-order functions

Meld term	CS term	Mathematical description		Domain
Transform	Map	$(\boldsymbol{a})_n \xrightarrow{f} (\boldsymbol{b})_n$	where $f(\boldsymbol{a}) \to \boldsymbol{b}$	Same as $(\boldsymbol{a})_n$
Filter	Filter	$(\boldsymbol{a})_n \xrightarrow{f} (\boldsymbol{a})_m$ where $m \leq n$	where $f(\boldsymbol{a}) \to \text{Boolean}$	Same as $(\boldsymbol{a})_n$
Monitor	—	$(\boldsymbol{a})_n \xrightarrow{f} (\ )_0$	where $f(\boldsymbol{a}) \to \text{Void}$	Same as $(\boldsymbol{a})_n$
Reduction	Fold	$(\boldsymbol{a})_n \xrightarrow{f_c} (\boldsymbol{c})_1$	where $f_c(\boldsymbol{a}) \to \boldsymbol{c}$	**Above** $(\boldsymbol{a})_n$
Splitter	Unfold	$(\boldsymbol{a})_1 \xrightarrow{f_n} (\boldsymbol{d})_m$	where $f_n(\boldsymbol{a}) \to (\boldsymbol{d})_n$	**Below** $(\boldsymbol{a})_n$
Zip	Zip	$\big((a)_n\, , (b)_n\big) \to (a, b)_n$		More nested domain

Fermilab