# Adaptable framework LDRD update

https://indico.fnal.gov/event/52666/contributions/231769/attachments/153101/198580/SCDProjects_LDRD_Knoepfel.pdf

Kyle J. Knoepfel

Monthly DUNE/LDRD meeting

9 August 2023

# Over the last few months

- Presented at

  CHEP 2023 in Track 5 (wrote draft of CHEP proceedings)

  Fermilab Frameworks Workshop

  DOE Review of US DUNE Offline & Computing

- Upgraded laptop to Mac silicon

  Identified error in one of the hashing algorithms I was using (fixed)

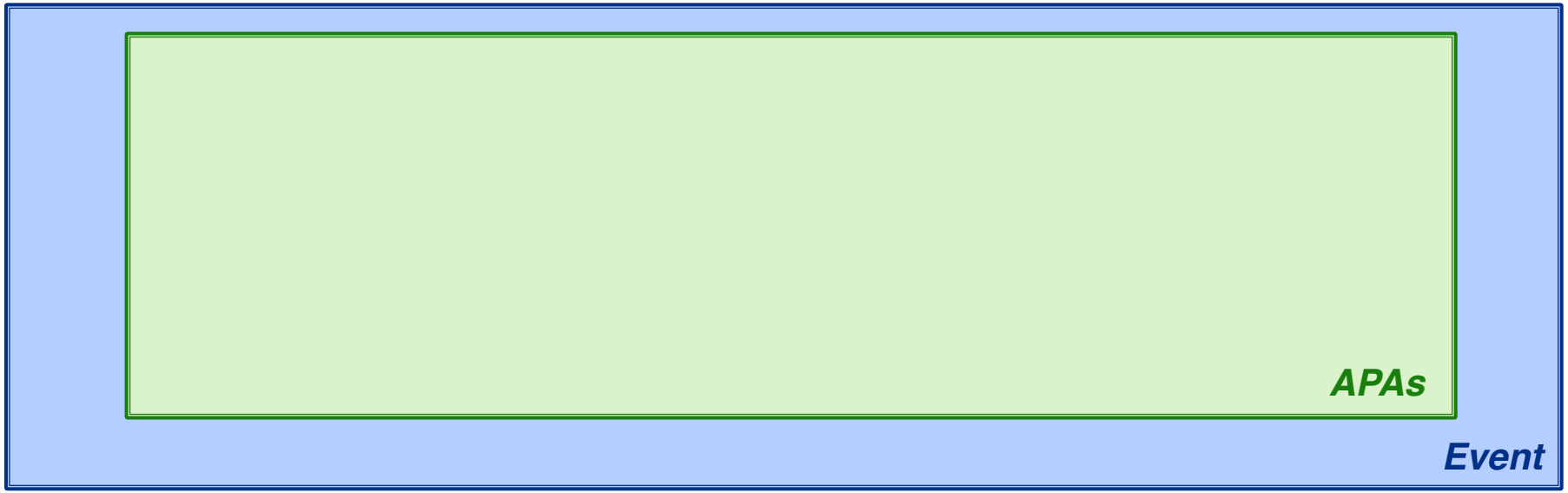
  Improved compatibility between GCC/Linux and Apple Clang

# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

# The prototype design

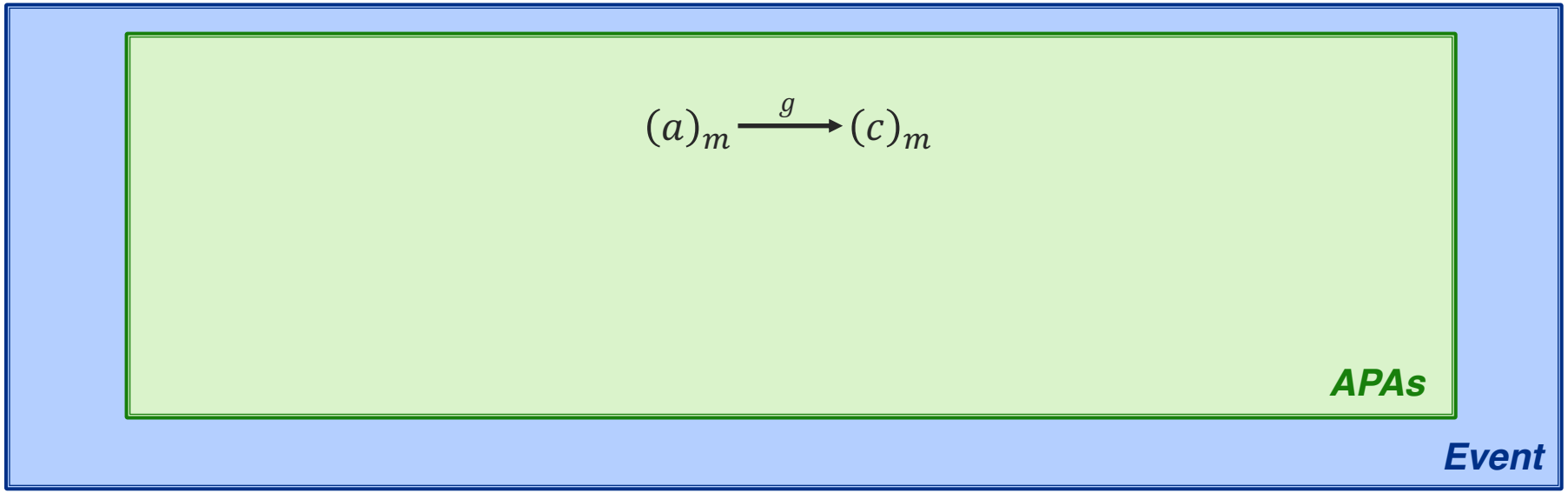
- A framework job is a graph of data products connected by user-provided operations of higher-order functions.



# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

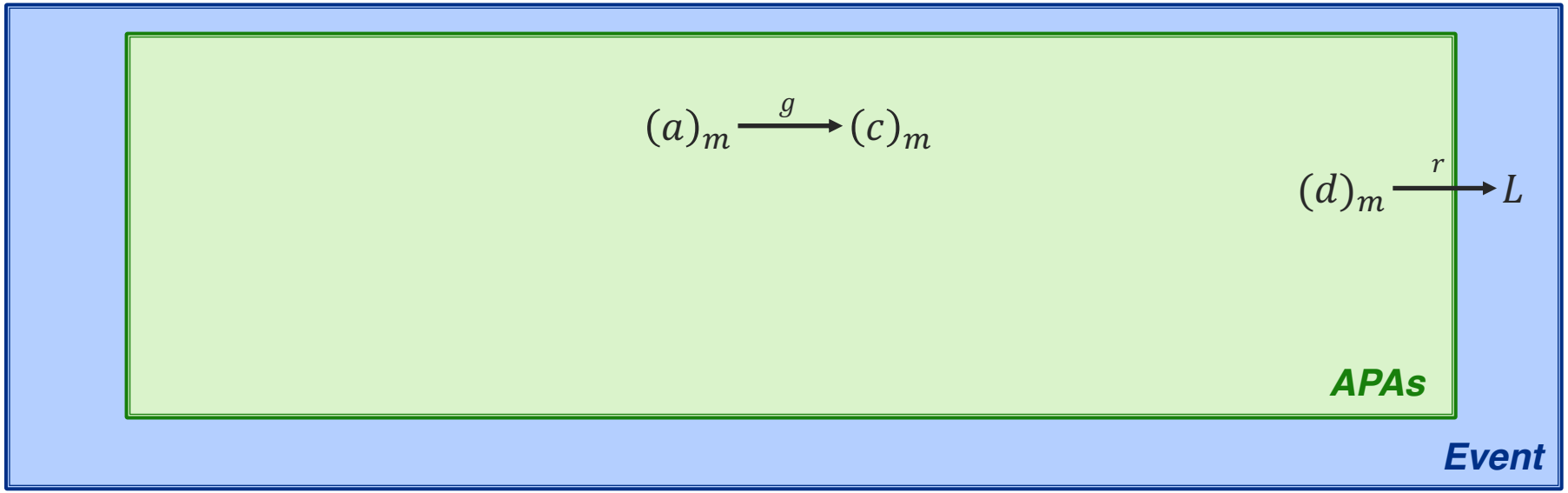
**Transform:** $g * (a)_m = (c)_m$ where $g(a) = c$



# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

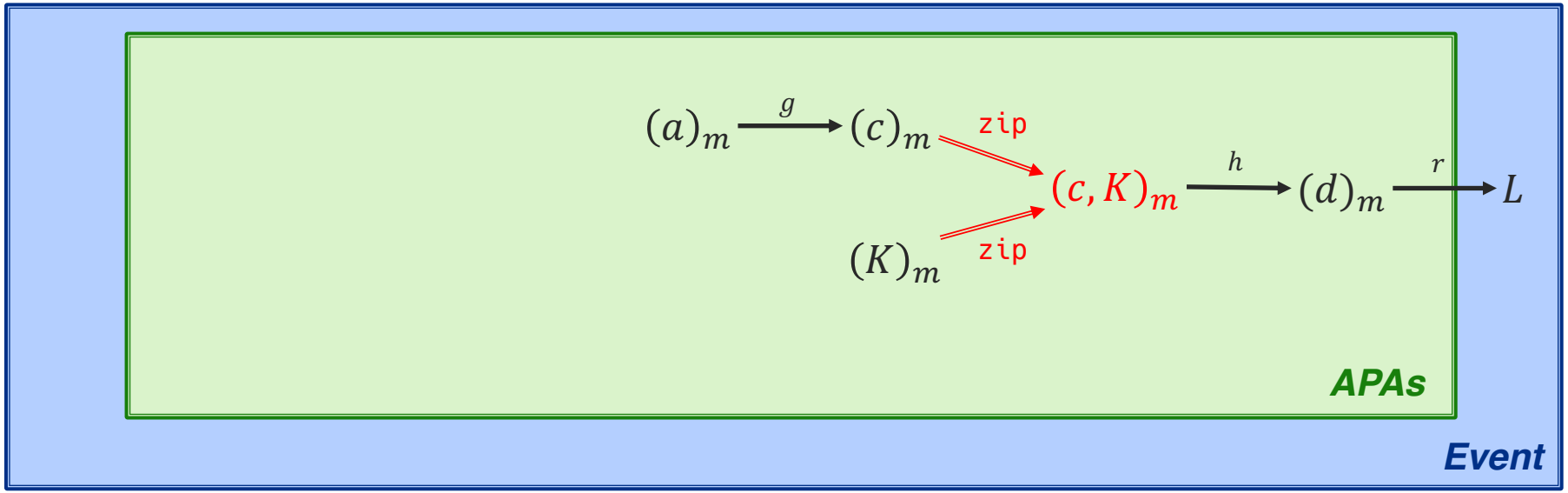
**Reduction:** $r^n/(d)_m = L$ where $r(l, d) = l$



# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

**Transform (two arguments):** $h * (c, K)_m = (d)_m$ where $h(c, K) = d$



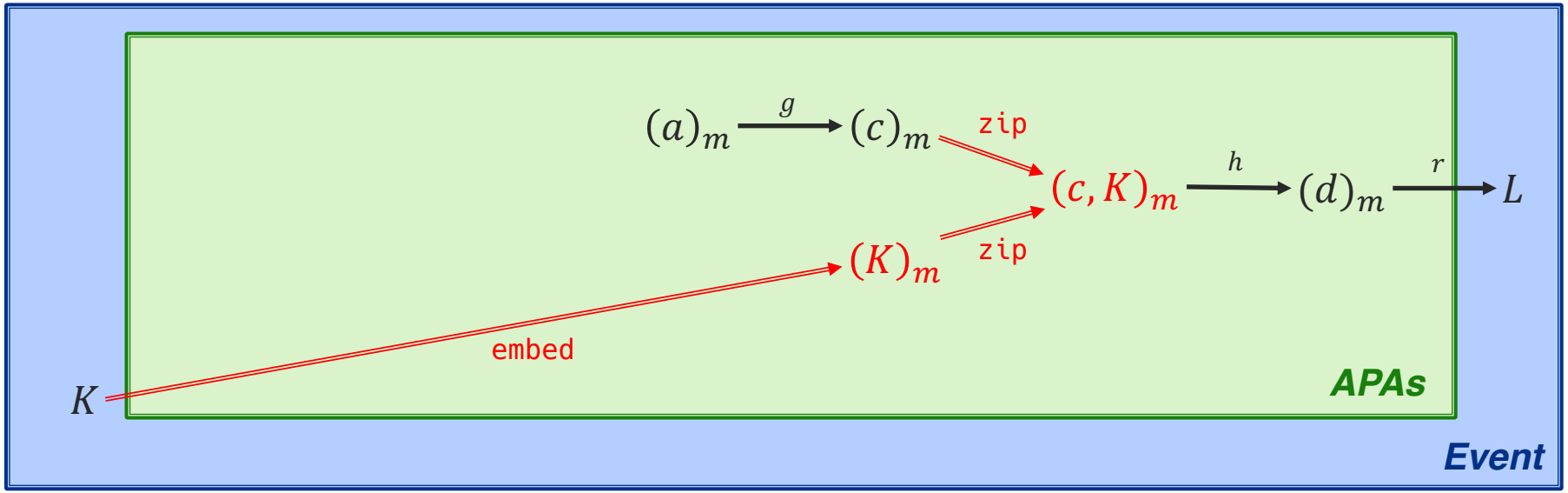



Fermilab

# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

**Transform (two arguments, two domains):** $h * (c, K)_m = (d)_m$ where $h(c, K) = d$



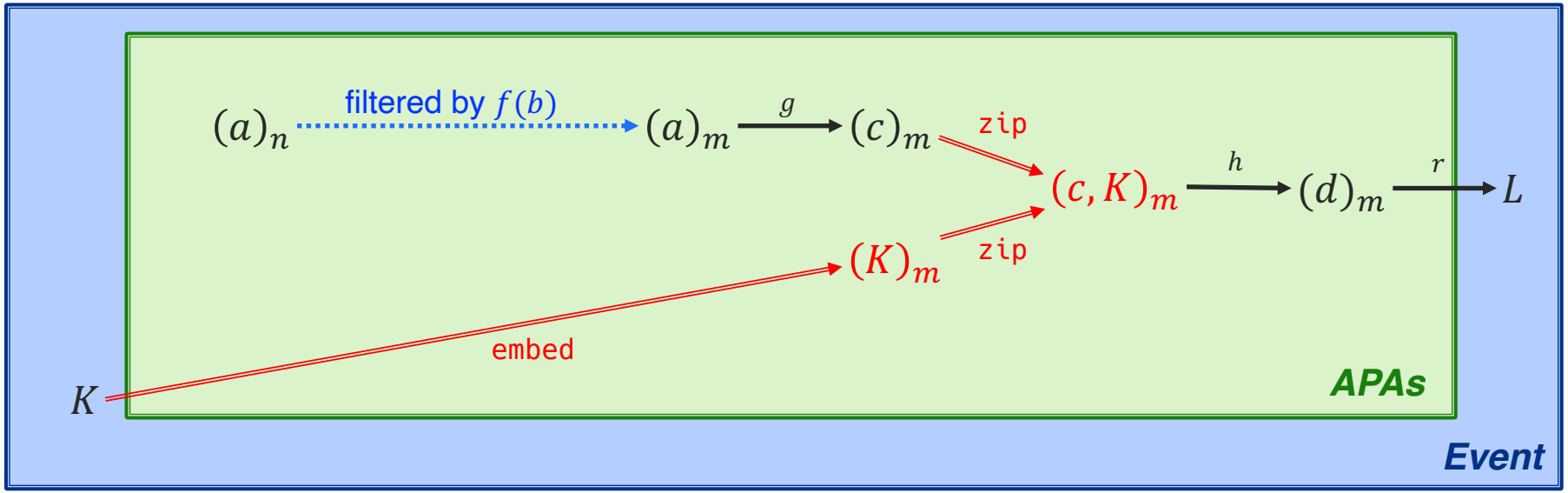



Fermilab

# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

**Filter:** $f \triangleleft (a)_n = (a)_m$ where $m \leq n$



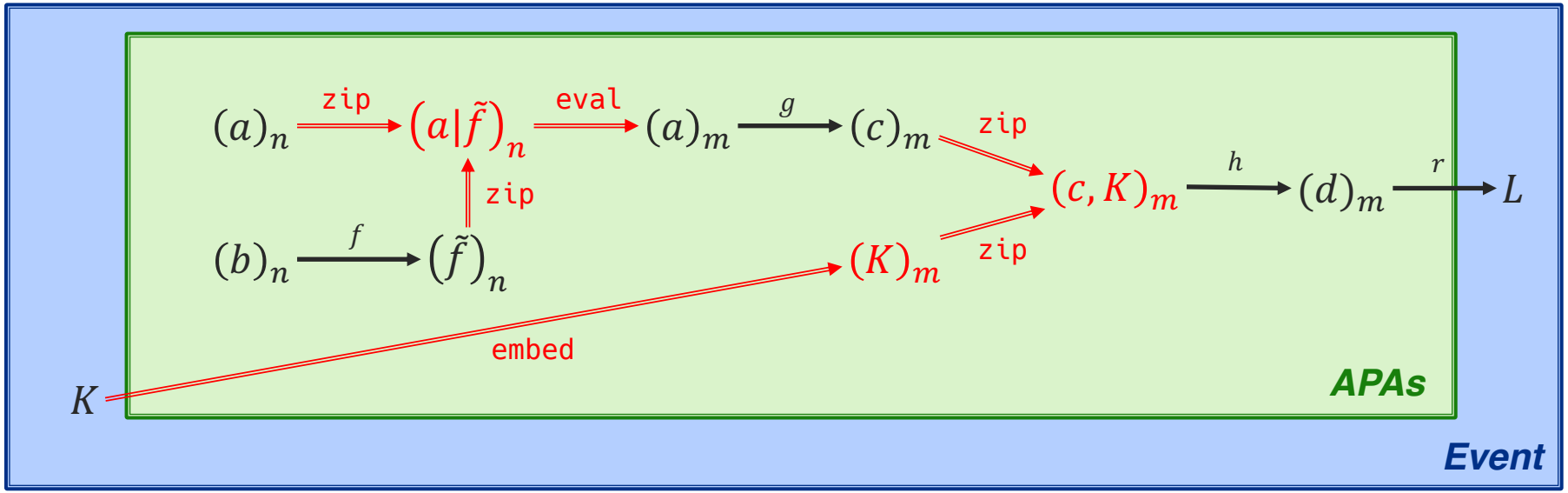



Fermilab

# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

**Filter:** $f \triangleleft (a)_n = (a)_m$ where $m \leq n$



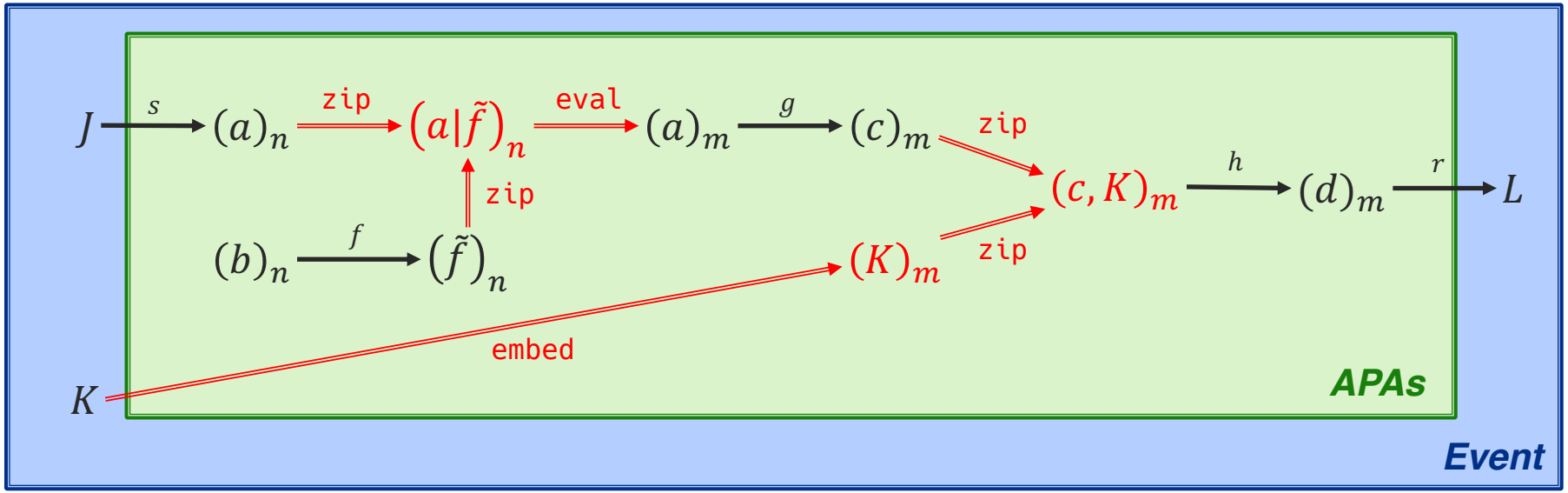



Fermilab

# The prototype design

- A framework job is a graph of data products connected by user-provided operations of higher-order functions.

**Split:** $s_n \setminus J = (a)_n$ where $s(j) = (j, a)$






Fermilab

# Framework-supported algorithm constructs

| Construct | Input data | Output data | Output level |
|---|---|---|---|
| **Transform** (map) | Product sequence | Product sequence | Same as input |
| **Filter** | Product sequence | Boolean sequence | Same as input |
| **Monitor** (absorb) | Product sequence | None | Same as input |
| **Reduction** (fold) | Product sequence | Product | Above input |
| **Splitter** (unfold) | Product | Product sequence | Below input |

# Next steps

- Determine further avenues to explore.  Options include:

  Sliding “event” windows

  Demonstrating support for asynchronous execution

  Compile-time generation of graph components

  *i.e. letting the program split-apply-combine a particular algorithm without requiring the user to specify the individual graph components*

  Mocking up some portion of a DUNE workflow

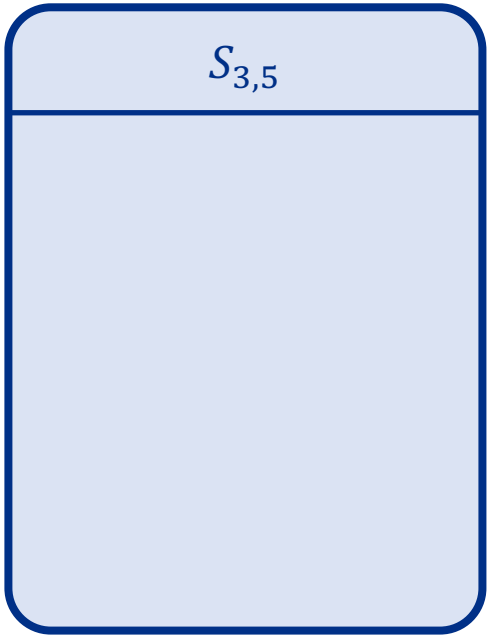
  Configuration

  Your thoughts?

# Backup slides

# Transform example

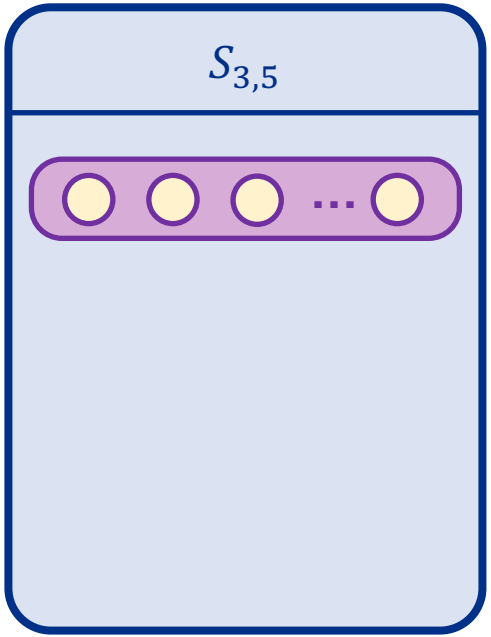
- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.



# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.

**Subrun products**

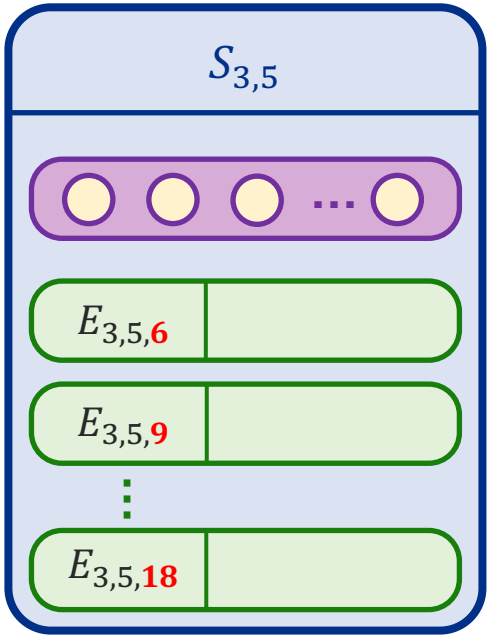





Fermilab

# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.

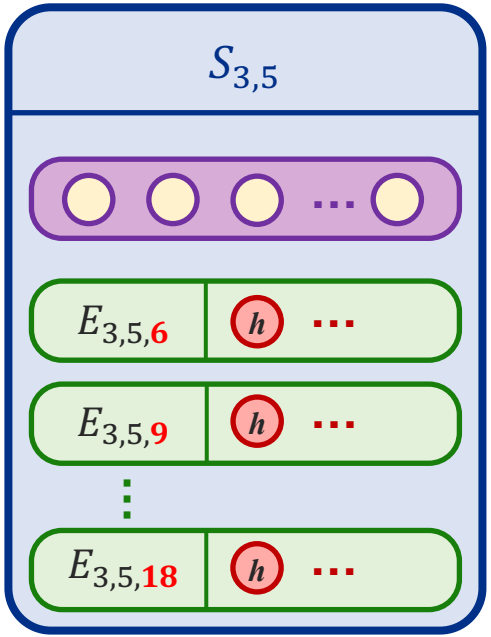


# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.

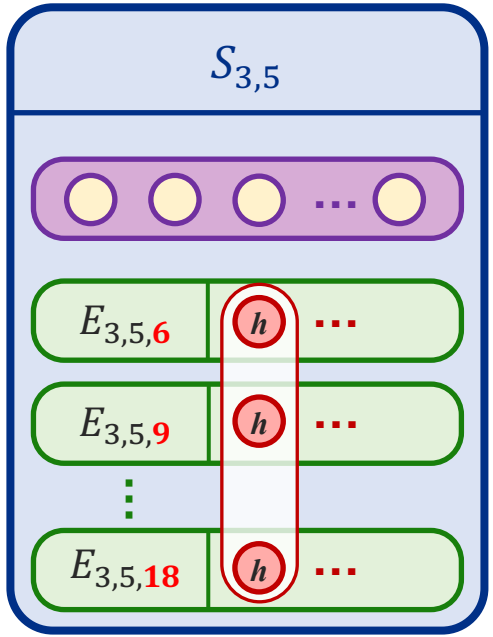
Subrun products

Events



# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.



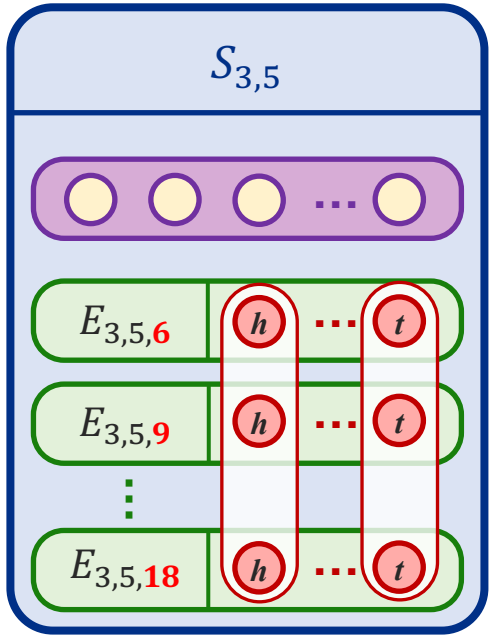

$(h)_n$



Fermilab

# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.



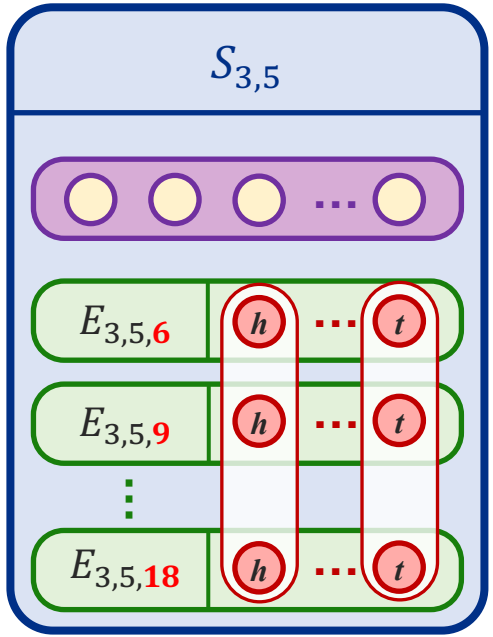

$$(h)_n \xrightarrow{f} (t)_n$$



Fermilab

# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.



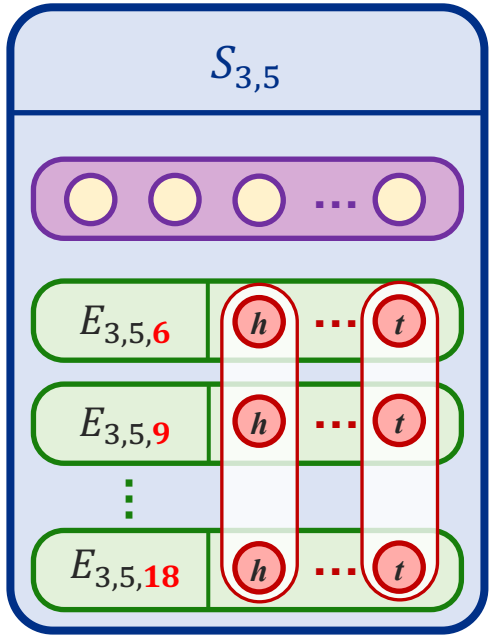

$$\left\{(h)_n \xrightarrow{f} (t)_n\right\} \in S_{3,5}$$



Fermilab

# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.




$$\left\{(h)_n \xrightarrow{f} (t)_n\right\} \in S_{3,5}$$

where $h, t \in E_{3,5,i}$ and

where $E_{3,5,i} \subset S_{3,5}$



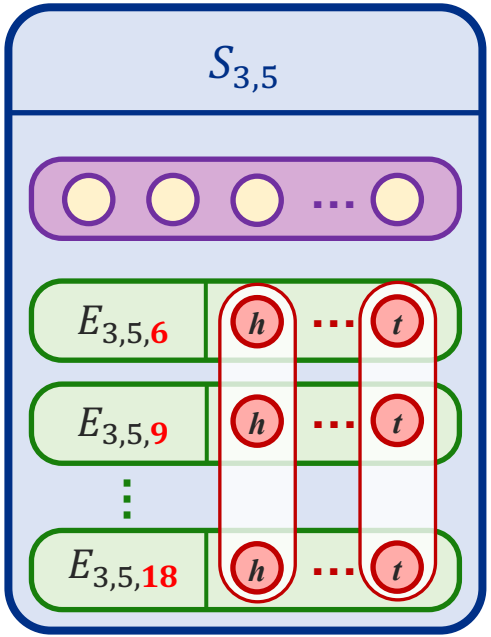
Fermilab

# Transform example

- Within subrun (3,5) use function $f$ to transform a hit collection $h$ in each event into a track collection $t$, such that $f(h) = t$.

Subrun products

Events




$$\left\{(h)_n \xrightarrow{f} (t)_n\right\} \in S_{3,5}$$

where $h, t \in E_{3,5,i}$ and

where $E_{3,5,i} \subset S_{3,5}$

The domain for transforms in framework jobs usually corresponds to the entire job, not a specific subset.



Fermilab

# What's the point?

- With art, specifying the domain of the user's function is done in C++ code (e.g.):

  `art::Event::getValidHandle<T>`, `art::Run::getProduct<T>`, etc.

- With Meld, the domain of the user function is specified via configuration-based string.

```
adder: meld.module('add') {
  input: [
    { element: 'odd', in_each: 'event' },
    { element: 'even', in_each: 'event' },
  ],
}
```

- In addition, when using a **reduction**, not only does one need to specify the domain of the user function, but one also must specify the domain over which the reduction applies.

```
adder: meld.module('adder') {
  domain: 'run'
  input: [
    { element: 'odd', in_each: 'event' },
  ],
}
```

# Consequences (1)

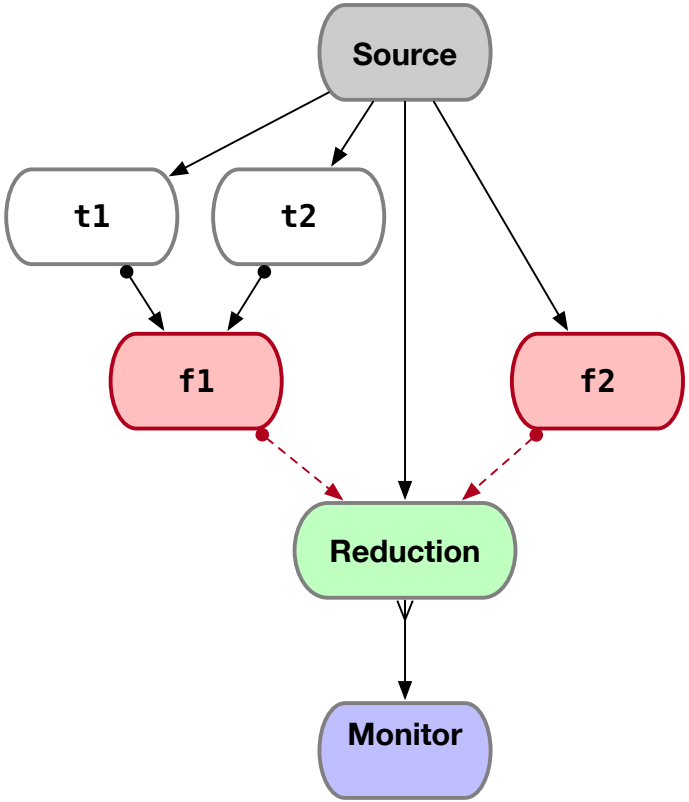
- I have pursued a graph-based approach under the assumption that it naturally expresses dependencies among user-provided algorithms.

  **Nodes**: User-provided algorithm to run on data.

  **Edges**: Expresses dependencies among nodes; data are passed along the edges.

- But graphs seem to work most naturally when data are passed along edges unconditionally.

- And we need to specify “domains” for inputs.



# Consequences (2)

- When considering configuration:

  Where does the domain of the node belong?  In the node configuration or edge configuration?

  If it belongs in the node, then notions of data flow get coupled with the node configuration.

  If it belongs with the edge, then the edge configuration becomes complicated.