Services: algorithms in disguise

DUNE framework requirements – joint meeting
26 April 2024

Preface: A note before we begin

Thank you for the opportunity to review your framework requirements.

We are excited to work on something truly new, and we thank you for your participation (and patience!) in helping us understand your needs.

We have many more questions (see responses in the google docs), but we are moving forward.

Although most of the requirements are not yet finalized, a design is forming based on:

- Many years of work on existing frameworks, understanding limitations, etc.
- Work with DUNE as an *art* stakeholder
- Dedicated LDRD work for DUNE's needs (i.e. Meld)
- Many discussions among the developers since January (>500 pages of meeting transcripts)

We want to be proactive with you in discussing ideas…which means more meetings. 😉

Preface: Definitions

"There are only two hard things in Computer Science: cache invalidation and naming things."

—Phil Karlton (by way of Martin Fowler)

We (both DUNE and the developers) have worked hard to introduce terms that we can use to describe framework concepts.

- DUNE provided a set of definitions as part of v0 of its DUNE Software Framework Requirements document.
- The developers have also started a glossary of terms in the "All responses" document at: https://docs.google.com/document/d/1GlnENatRxzk8ewMapL7ewCOS_OPTqWg5QUpR27_K3QE/edit

We should expect these definitions to change as concepts become clearer.

Relevant requirements for today

[Req. 3] The framework MUST be sufficiently modular in design to allow re-use of framework services.

[Req. 47] The framework MUST provide the infrastructure/functionality/mechanism (known as a service) to wrap useful classes.

[Req. 48] The procedure/syntax to wrap these classes MUST NOT prevent these classes from being used outside of the framework context.

[Req. 49] The wrapping procedure SHOULD include a mechanism to declare whether the useful class is thread safe or not, since that may influence execution steps. The framework should have the ability to treat the two cases appropriately. It is not up to the framework to determine or ensure whether the useful class itself is thread safe.

Our responses:

https://docs.google.com/document/d/1GlnENatRxzk8ewMapL7ewCOS_OPTqWg5QUpR27_K3QE/edit

Some of our responses re. services

"We do not assume that a 'service' will be an essential ingredient of the framework."

- First Devs response to requirement 3

"We would like to discuss with you replacing the concept of a service with simply a data-providing algorithm."

- Second Devs response to requirements 47-49

Today's meeting is that discussion.

Warning: Services cannot be discussed in a vacuum—we will also touch on (e.g.) framework-independent algorithms, data provenance, and flexible processing units.

We hope today's discussion will guide how to handle some of the other unsettled requirements.

Framework and external services

DUNE Software Framework Requirements (v0)

Framework Service - An object that is initialized at runtime that can be accessed by algorithms/modules to provide some functionality. They are configured at runtime before initialization. The intent of having these services is to have a standardized configuration and initialization of each service within a given runtime.

- Example of class + services
    - DUNE Software class (header) that accesses a space charge calibration
    - Wrapper that creates a service that is accessible within modules as described above

External Service - An object that can be accessed by algorithms/modules to provide some functionality that is initialized outside the framework (i.e. database utilities).

Framework and external services

Framework Service - An object that is initialized at runtime that can be accessed by algorithms/modules to provide some functionality. They are configured at runtime before initialization. The intent of having these services is to have a standardized configuration and initialization of each service within a given runtime.

- Example of class + services
  - DUNE Software class (header) that accesses a space charge calibration
  - Wrapper that creates a service that is accessible within modules as described above

Good separation between experiment code and framework glue.

External Service - An object that can be accessed by algorithms/modules to provide some functionality that is initialized outside the framework (i.e. database utilities).

What is an *art* service?

An *art* service exhibits the following characteristics:

1. It is an object constructed from a dynamically loaded library specified by user configuration.
2. Its lifetime begins before the first *art* module is constructed and ends after the last *art* module is destroyed.
3. It enables the registration of functions to be called by the framework during transitions that are not otherwise accessible to *art* modules.
4. It can be globally accessed from any code executed during the framework job through the creation of an `art::ServiceHandle`.
5. It can depend on other services through the `art::ServiceHandle` interface.
6. It can be polymorphic, where `art::ServiceHandle` provides access to a concrete service through a service interface.

Why do people use *art* services?

1. Message-logging

2. Basic profiling of user-defined algorithms (e.g. `TimeTracker`, `MemoryTracker`)

3. Simplifying the use of external libraries that require management of global state (e.g. `TFileService`)

4. Enabling the sharing of singleton-like objects (e.g. `Geometry`)

5. Efficiently providing data to algorithms independent of the *run-subrun-event* hierarchy (such as conditions-style access to external databases and calculations of derived data).

Why do people use *art* services?

1. Message-logging
2. Basic profiling of user-defined algorithms (e.g. `TimeTracker`, `MemoryTracker`)

   Expect these facilities to be provided without services.

3. Simplifying the use of external libraries that require management of global state (e.g. `TFileService`)
4. Enabling the sharing of singleton-like objects (e.g. `Geometry`)
5. Efficiently providing data to algorithms independent of the *run-subrun-event* hierarchy (such as conditions-style access to external databases and calculations of derived data).

Why do people use *art* services?

1. Message-logging

2. Basic profiling of user-defined algorithms (e.g. `TimeTracker`, `MemoryTracker`)

   Expect these facilities to be provided without services.

3. Simplifying the use of external libraries that require management of global state (e.g. `TFileService`)

   Managing global data is necessary, but a non-service alternative is possible.

4. Enabling the sharing of singleton-like objects (e.g. `Geometry`)

5. Efficiently providing data to algorithms independent of the *run-subrun-event* hierarchy (such as conditions-style access to external databases and calculations of derived data).

Why do people use *art* services?

1. Message-logging
2. Basic profiling of user-defined algorithms (e.g. `TimeTracker`, `MemoryTracker`)

   Expect these facilities to be provided without services.

3. Simplifying the use of external libraries that require management of global state (e.g. `TFileService`)

   Managing global data is necessary, but a non-service alternative is possible.

4. Enabling the sharing of singleton-like objects (e.g. `Geometry`)
5. Efficiently providing data to algorithms independent of the *run-subrun-event* hierarchy (such as conditions-style access to external databases and calculations of derived data).

   This is what algorithms are for—providing data (today's discussion)**.**

Making tracks with a calibration offset

A common pattern in current algorithms

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


  ...
}
```

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

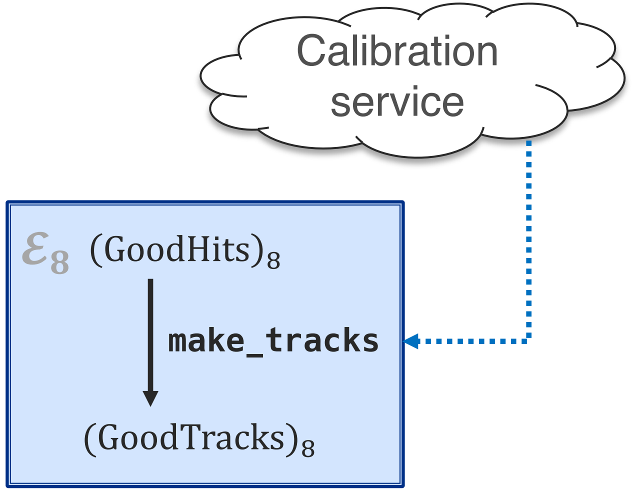
$$\begin{tikzcd} \mathcal{E}_8 & (\text{GoodHits})_8 \arrow[d, "\texttt{make\_tracks}"] \\ & (\text{GoodTracks})_8 \end{tikzcd}$$

Making tracks with a calibration offset

A common pattern in current algorithms

```
Tracks make_tracks(Hits const& hits)
{
  art::ServiceHandle<Calibration> calibration;
  ScalarOffset const& offset = calibration->Offset();
  ...
}
```

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

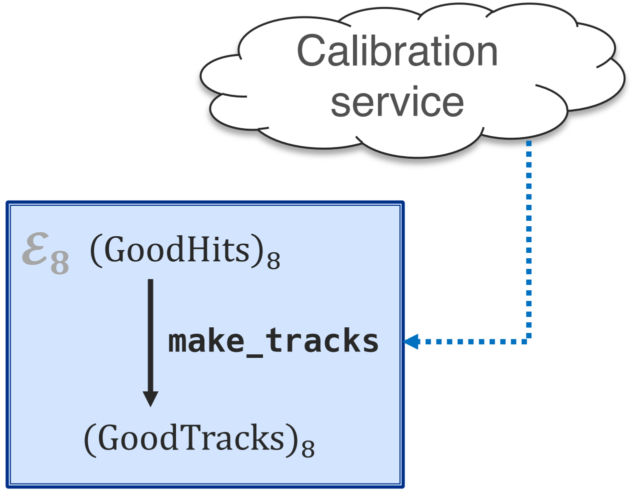


Making tracks with a calibration offset

A common pattern in current algorithms

```
Tracks make_tracks(Hits const& hits)
{
  art::ServiceHandle<Calibration> calibration;
  ScalarOffset const& offset = calibration->Offset();
  ...
}
```

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



With this approach:

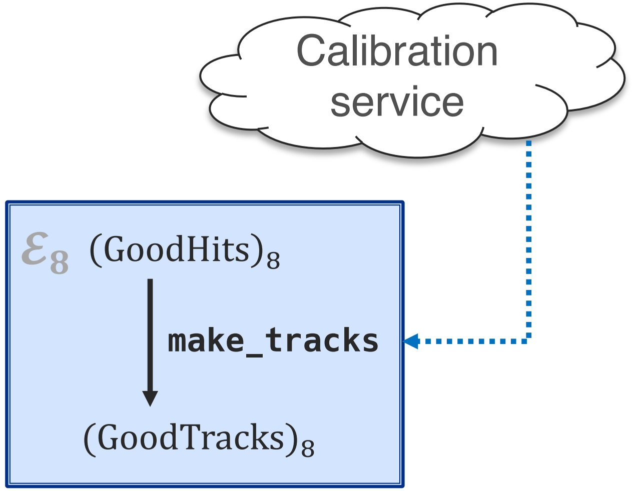
1. The algorithm directly depends on the framework.
2. The algorithm has an implicit data dependency, making it more difficult to understand and use.
3. Incomplete provenance of the data provided to/created by the algorithm.
4. The thread-safety of the algorithm is either compromised or, at best, unclear.

Making tracks with a calibration offset

An improvement: *retrieve the service outside of the algorithm*

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

```
void TrackMaker::produce(art::Event& e)
{
  Hits const& hits = e.getProduct<Hits>("GoodHits");
  art::ServiceHandle<Calibration> calibration;
  Tracks tracks = make_tracks(hits, calibration->Offset());
  e.put(std::make_unique<Tracks>(std::move(tracks)), "GoodTracks")
}
```

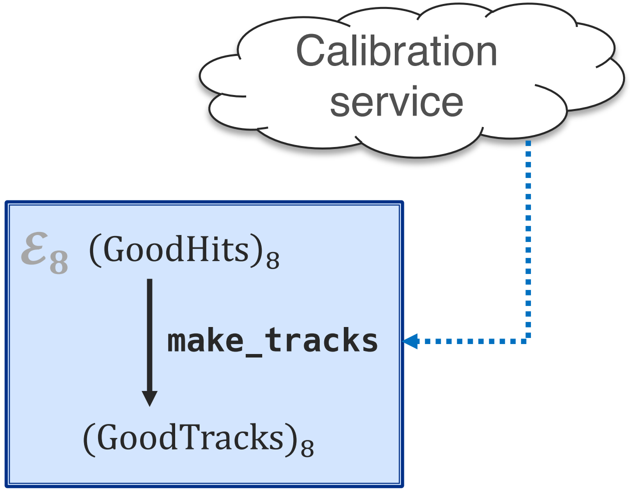


Making tracks with a calibration offset

***An improvement:** retrieve the service outside of the algorithm*

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

```
void TrackMaker::produce(art::Event& e)
{
  Hits const& hits = e.getProduct<Hits>("GoodHits");
  art::ServiceHandle<Calibration> calibration;
  Tracks tracks = make_tracks(hits, calibration->Offset());
  e.put(std::make_unique<Tracks>(std::move(tracks)), "GoodTracks")
}
```



With this approach:

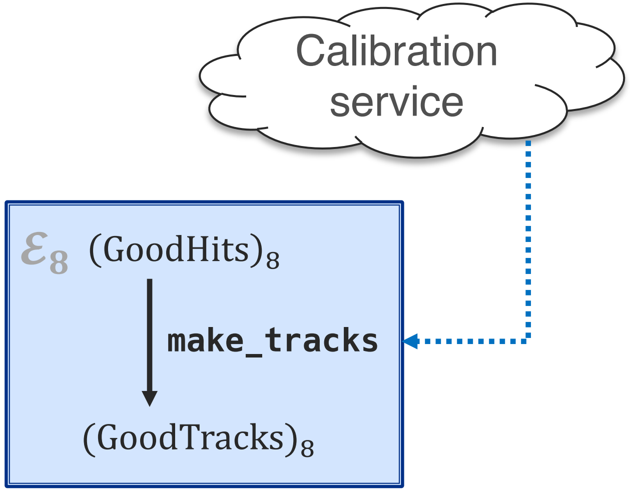
1. ~~The algorithm directly depends on the framework.~~
2. ~~The algorithm has an implicit data dependency, making it more difficult to understand and use.~~
3. Incomplete provenance of the data provided to/created by the algorithm.
4. ~~The thread-safety of the algorithm is either compromised or, at best, unclear.~~

Making tracks with a calibration offset

***An improvement:** retrieve the service outside of the algorithm*

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

```
void TrackMaker::produce(art::Event& e)
{
  Hits const& hits = e.getProduct<Hits>("GoodHits");
  art::ServiceHandle<Calibration> calibration;
  Tracks tracks = make_tracks(hits, calibration->Offset());
  e.put(std::make_unique<Tracks>(std::move(tracks)), "GoodTracks")
}
```



With this approach:

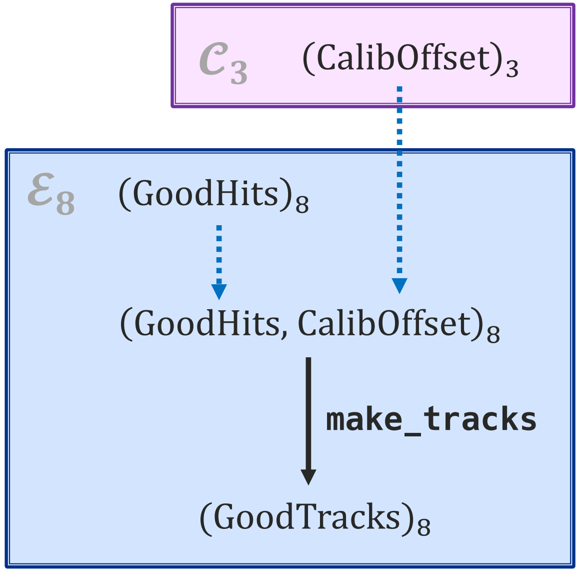
1. The algorithm is independent of the framework.
2. The algorithm has explicit dependencies.
3. Incomplete provenance of the data provided to/created by the algorithm.
4. Thread-safety of algorithm is easier to understand.

Making tracks with a calibration offset

Where we plan to go

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

```
REGISTER(m)
{
  m.with(make_tracks)
   .transform("GoodHits"_in("event"),
              "CalibOffset"_in("calibration"))
   .to("GoodTracks")
   .for_each("event");
}
```

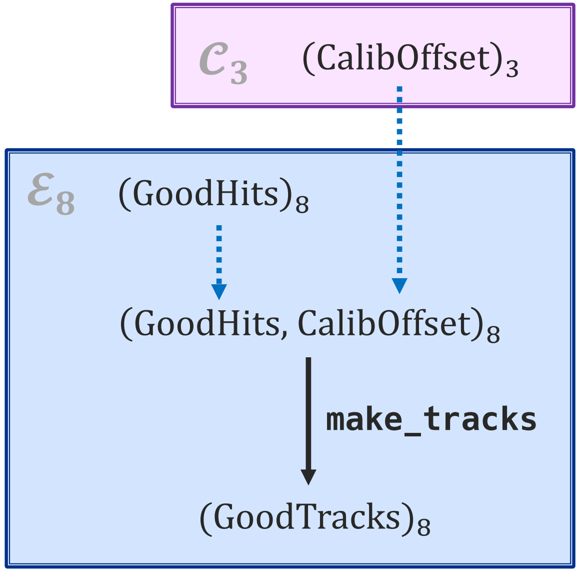


Making tracks with a calibration offset

Where we plan to go

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

```
REGISTER(m)
{
  m.with(make_tracks)
   .transform("GoodHits"_in("event"),
              "CalibOffset"_in("calibration"))
   .to("GoodTracks")
   .for_each("event");
}
```



With this approach:

1. The algorithm is independent of the framework.
2. The algorithm has explicit dependencies.
3. ~~Incomplete provenance of the data provided to/created by the algorithm.~~
4. Thread-safety of algorithm is easier to understand.

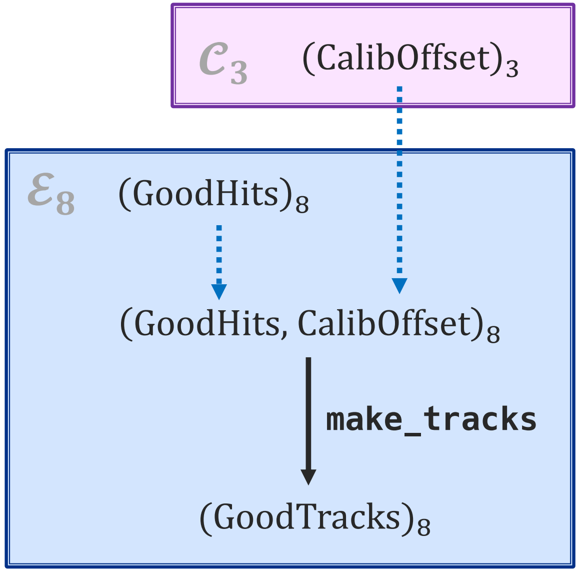
Fermilab

Making tracks with a calibration offset

Where we plan to go

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

```
REGISTER(m)
{
  m.with(make_tracks)
   .transform("GoodHits"_in("event"),
              "CalibOffset"_in("calibration"))
   .to("GoodTracks")
   .for_each("event");
}
```



$$\begin{tikzcd}
\mathcal{C}_3 & & (\text{CalibOffset})_3 \arrow[dd, dotted] \\
\mathcal{E}_8 & (\text{GoodHits})_8 \arrow[d, dotted] & \\
& (\text{GoodHits, CalibOffset})_8 \arrow[d, "\texttt{make\_tracks}"] & \\
& (\text{GoodTracks})_8 &
\end{tikzcd}$$

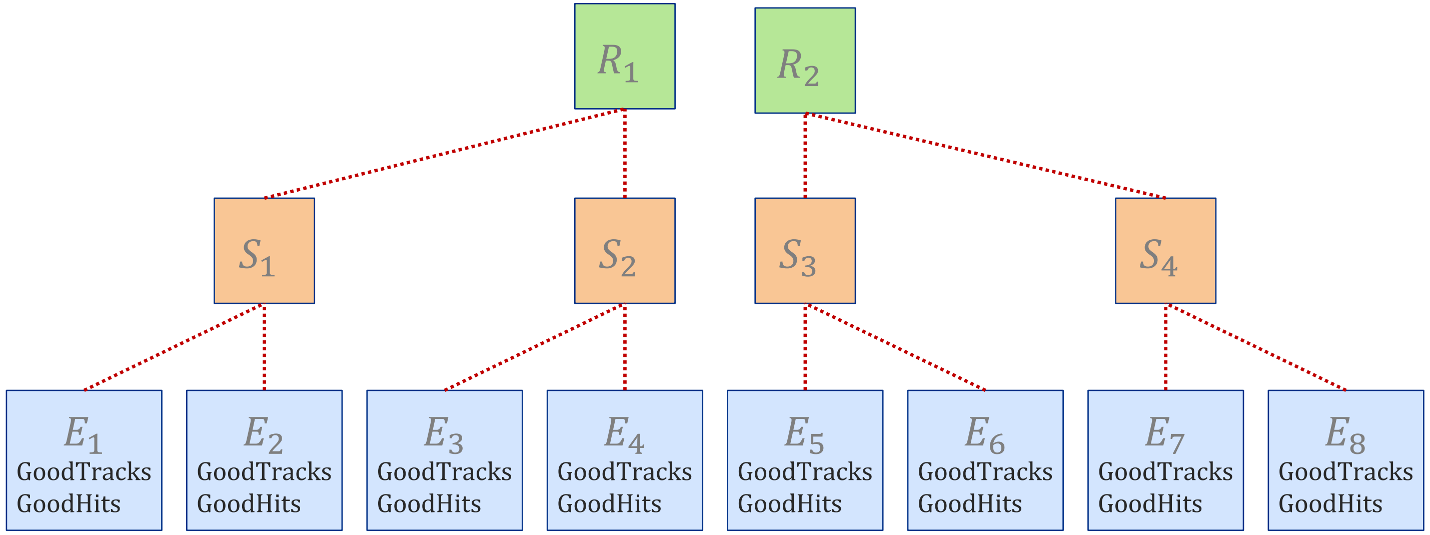
With this approach:

1. The algorithm is independent of the framework.
2. The algorithm has explicit dependencies.
3. Data fully tracked by framework.
4. Thread-safety of algorithm is easier to understand.

Relationship to “flexible processing units”

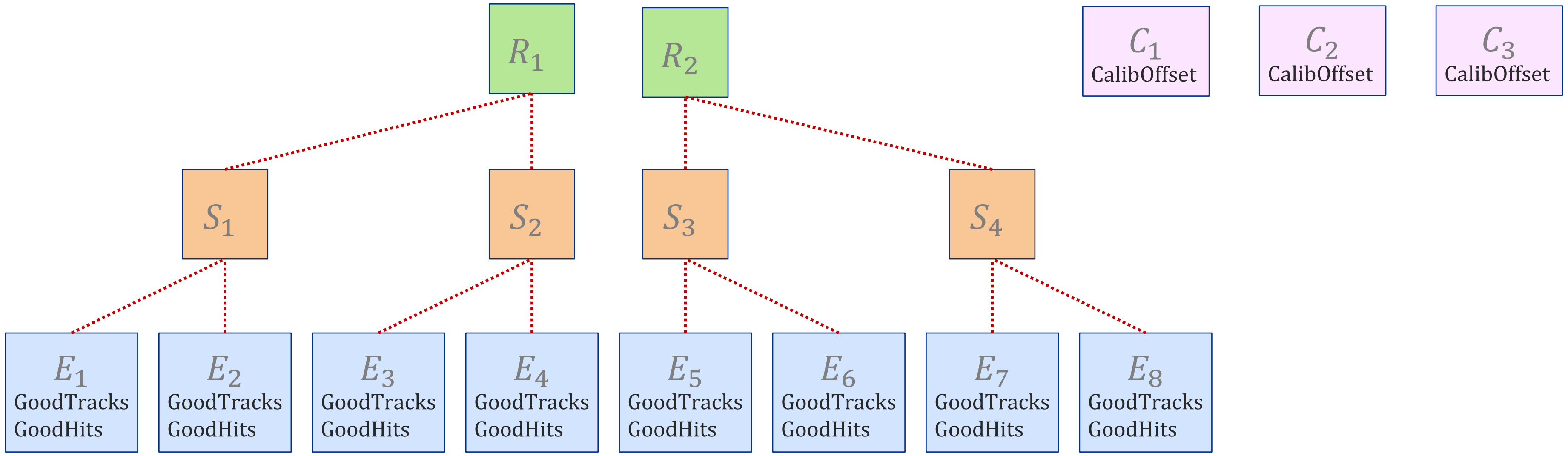
Relationship to “flexible processing units”

DUNE is used to the rigid art hierarchy


R_1
R_2
S_1
S_2
S_3
S_4
E_1 GoodTracks GoodHits
E_2 GoodTracks GoodHits
E_3 GoodTracks GoodHits
E_4 GoodTracks GoodHits
E_5 GoodTracks GoodHits
E_6 GoodTracks GoodHits
E_7 GoodTracks GoodHits
E_8 GoodTracks GoodHits

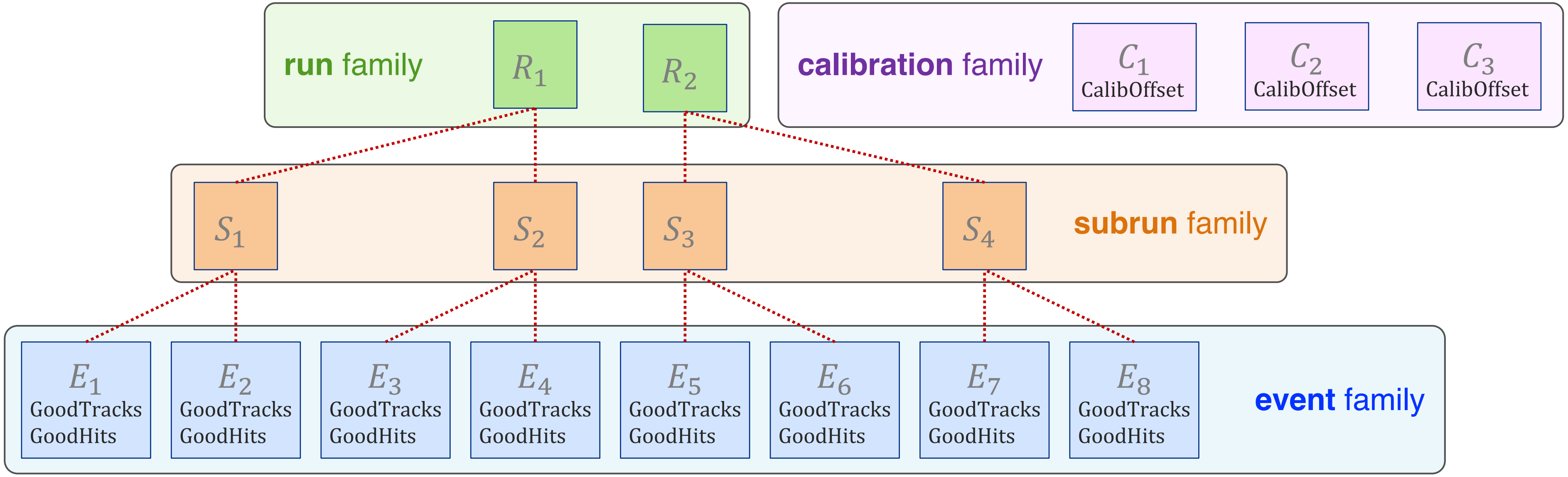
Relationship to “flexible processing units”

DUNE is used to the rigid art hierarchy, but it needs something more flexible



R_1
R_2
C_1 CalibOffset
C_2 CalibOffset
C_3 CalibOffset
S_1
S_2
S_3
S_4
E_1 GoodTracks GoodHits
E_2 GoodTracks GoodHits
E_3 GoodTracks GoodHits
E_4 GoodTracks GoodHits
E_5 GoodTracks GoodHits
E_6 GoodTracks GoodHits
E_7 GoodTracks GoodHits
E_8 GoodTracks GoodHits

Data families

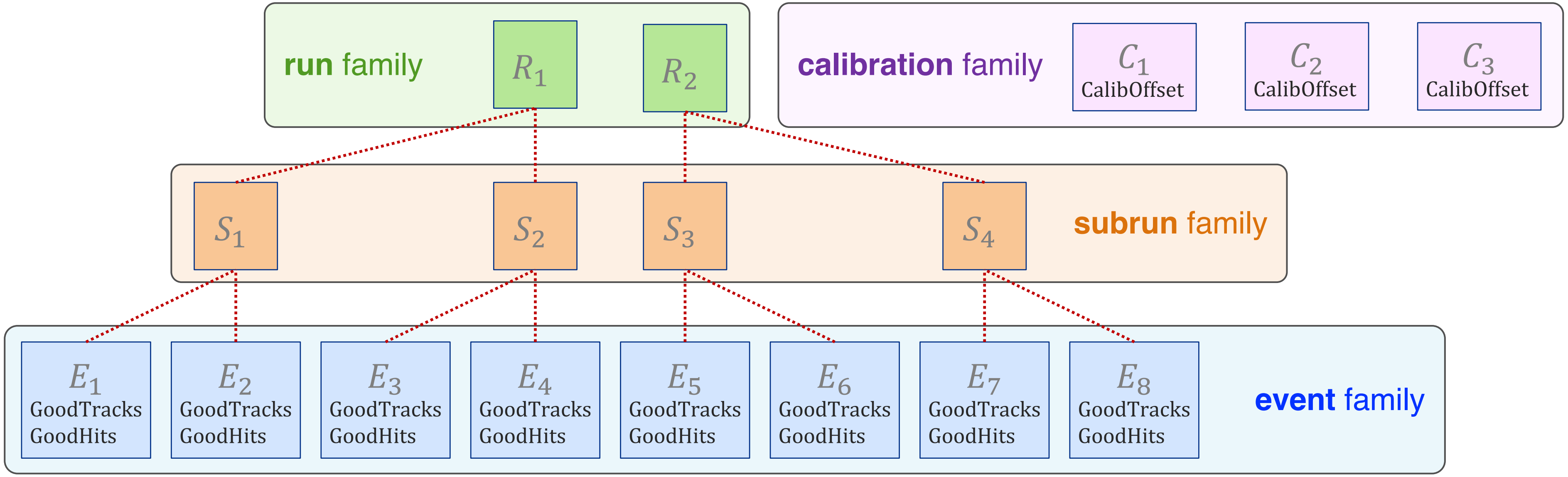

run family
R_1
R_2
calibration family
C_1 CalibOffset
C_2 CalibOffset
C_3 CalibOffset
S_1
S_2
S_3
S_4
subrun family
E_1 GoodTracks GoodHits
E_2 GoodTracks GoodHits
E_3 GoodTracks GoodHits
E_4 GoodTracks GoodHits
E_5 GoodTracks GoodHits
E_6 GoodTracks GoodHits
E_7 GoodTracks GoodHits
E_8 GoodTracks GoodHits
event family

Fermilab

Data families

DUNE must be able to define new families

(to be discussed in the future meeting on FPUs)

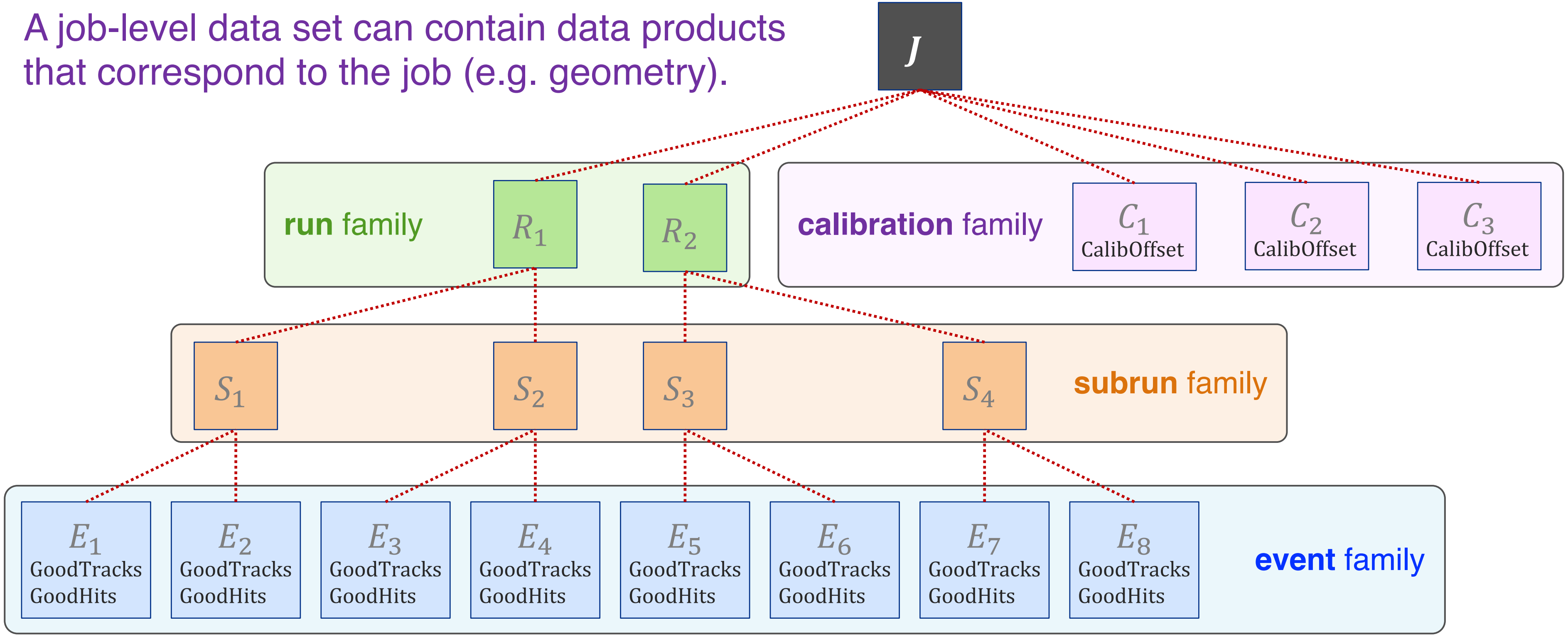


run family
R_1
R_2
calibration family
C_1 CalibOffset
C_2 CalibOffset
C_3 CalibOffset
S_1
S_2
S_3
S_4
subrun family
E_1 GoodTracks GoodHits
E_2 GoodTracks GoodHits
E_3 GoodTracks GoodHits
E_4 GoodTracks GoodHits
E_5 GoodTracks GoodHits
E_6 GoodTracks GoodHits
E_7 GoodTracks GoodHits
E_8 GoodTracks GoodHits
event family

Fermilab

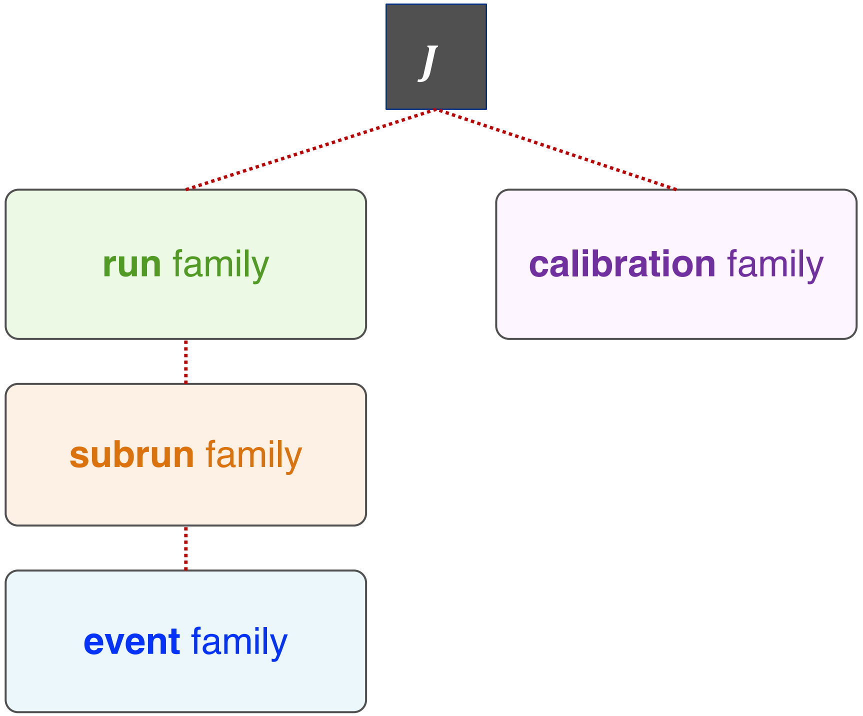
Data families

A job-level data set can contain data products that correspond to the job (e.g. geometry).



J
run family
R_1
R_2
calibration family
C_1 CalibOffset
C_2 CalibOffset
C_3 CalibOffset
S_1
S_2
S_3
S_4
subrun family
E_1 GoodTracks GoodHits
E_2 GoodTracks GoodHits
E_3 GoodTracks GoodHits
E_4 GoodTracks GoodHits
E_5 GoodTracks GoodHits
E_6 GoodTracks GoodHits
E_7 GoodTracks GoodHits
E_8 GoodTracks GoodHits
event family

Data-family hierarchies



Upshot

We believe DUNE’s “services” needs can be met without globally accessible stateful objects.

The framework’s support of experiment-defined FPUs is a crucial ingredient of this.

As DUNE identifies more facilities that must be provided by the framework, the developers will evaluate whether such facilities can be provided without resorting to a “services” approach.

We will believe pursuing a functional approach ***to the greatest extent possible*** will be best for meeting DUNE’s computing (and ultimately physics) needs.

Thanks

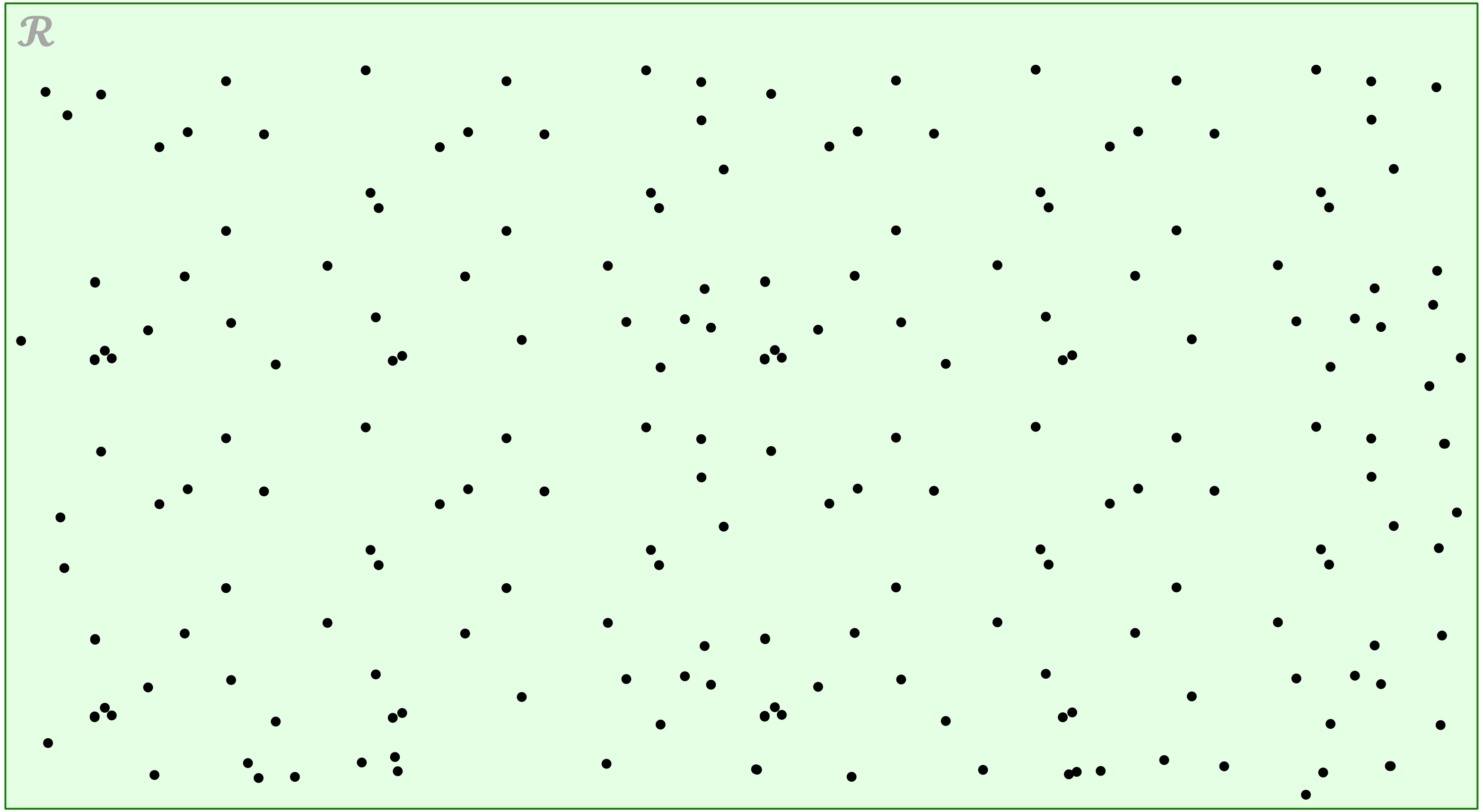
Back-up slides

Sets, partitions, refinements, and families

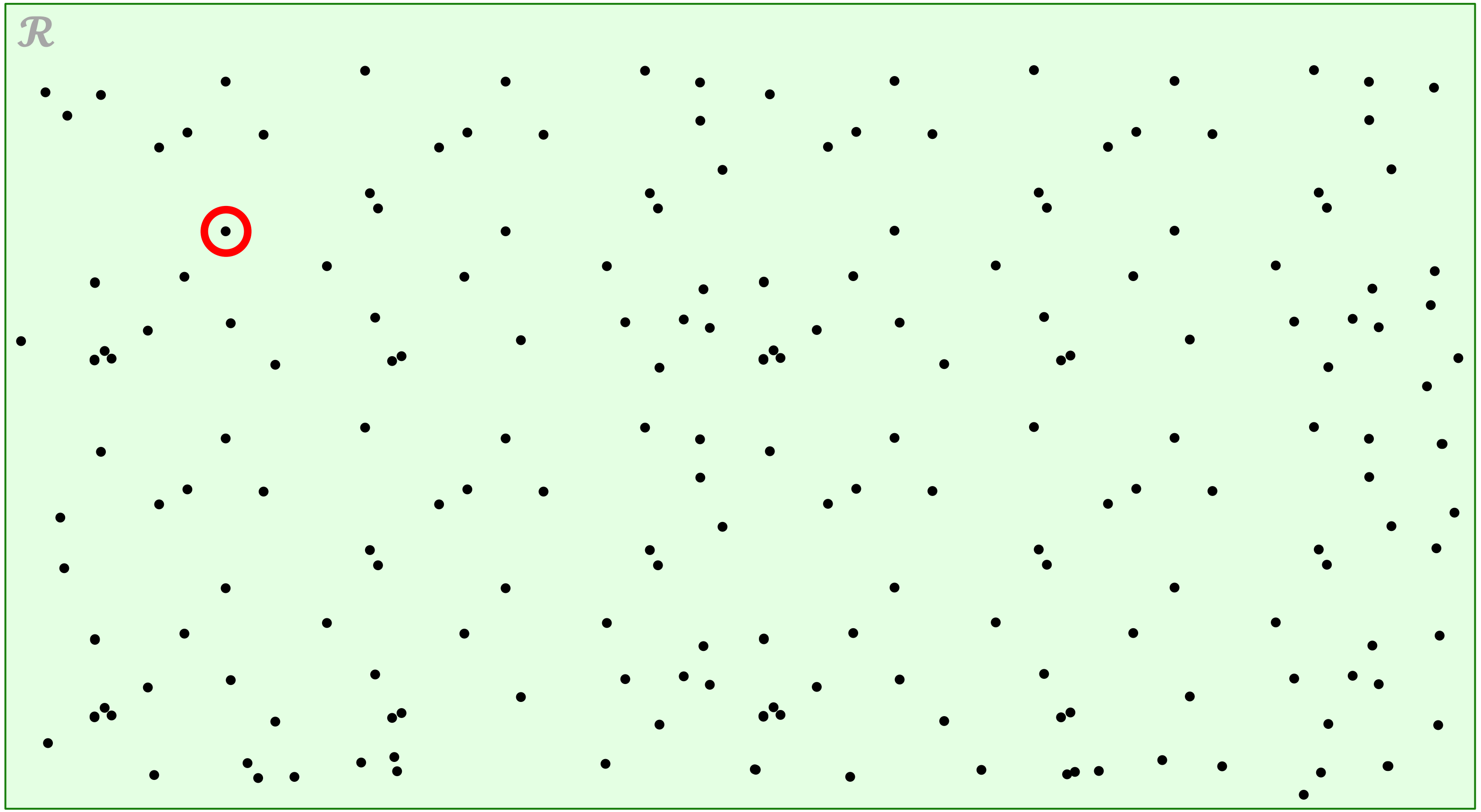
- A hierarchy of data sets corresponds to mathematical set partitions and refinements
  - **Set**: All data in job
  - **Partition**: Data grouped into runs
  - **Refinement**: Runs grouped into subruns, etc.
  - **Further refinement:** …

- A hierarchy of data families is a family tree.

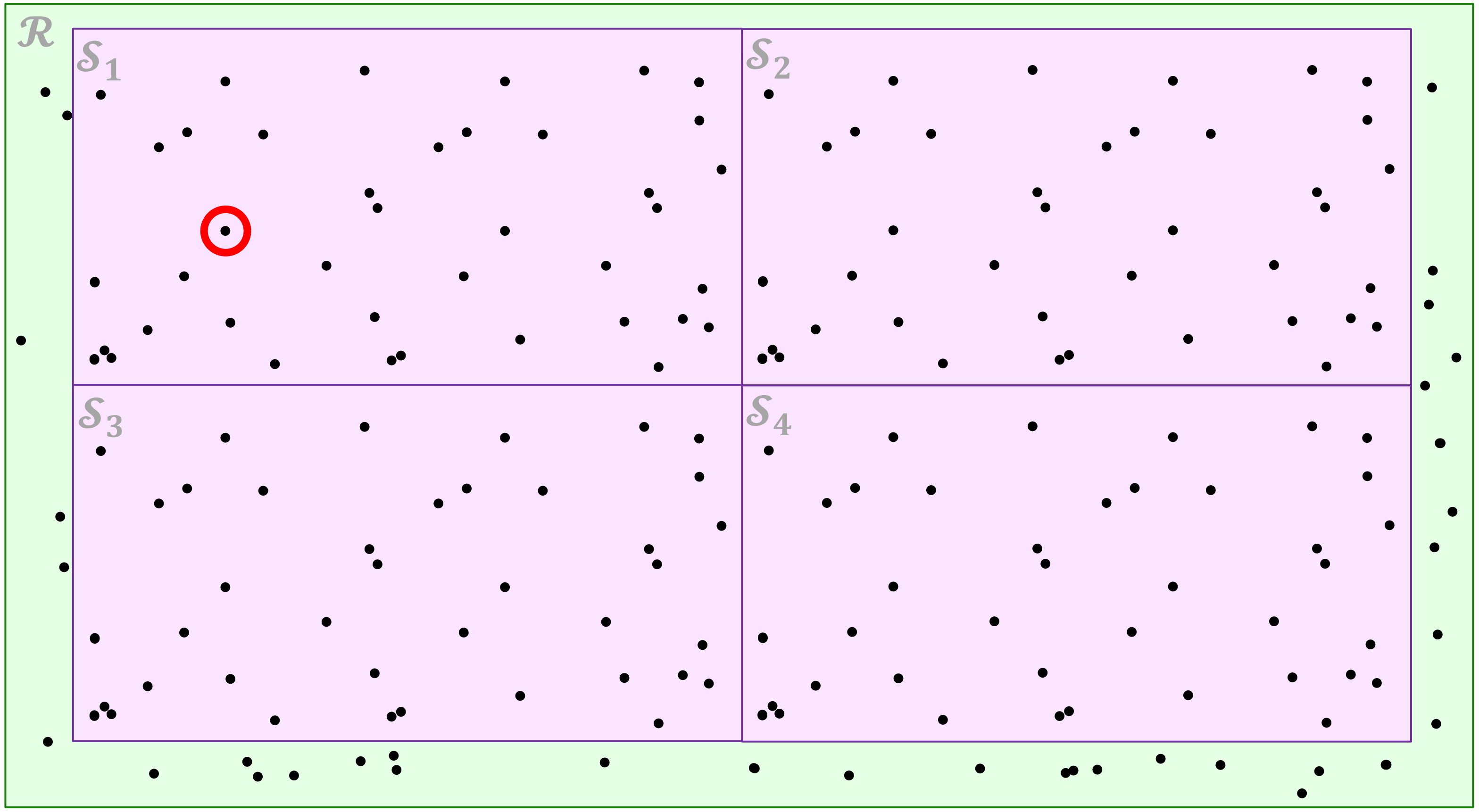
Looking at the data (set)



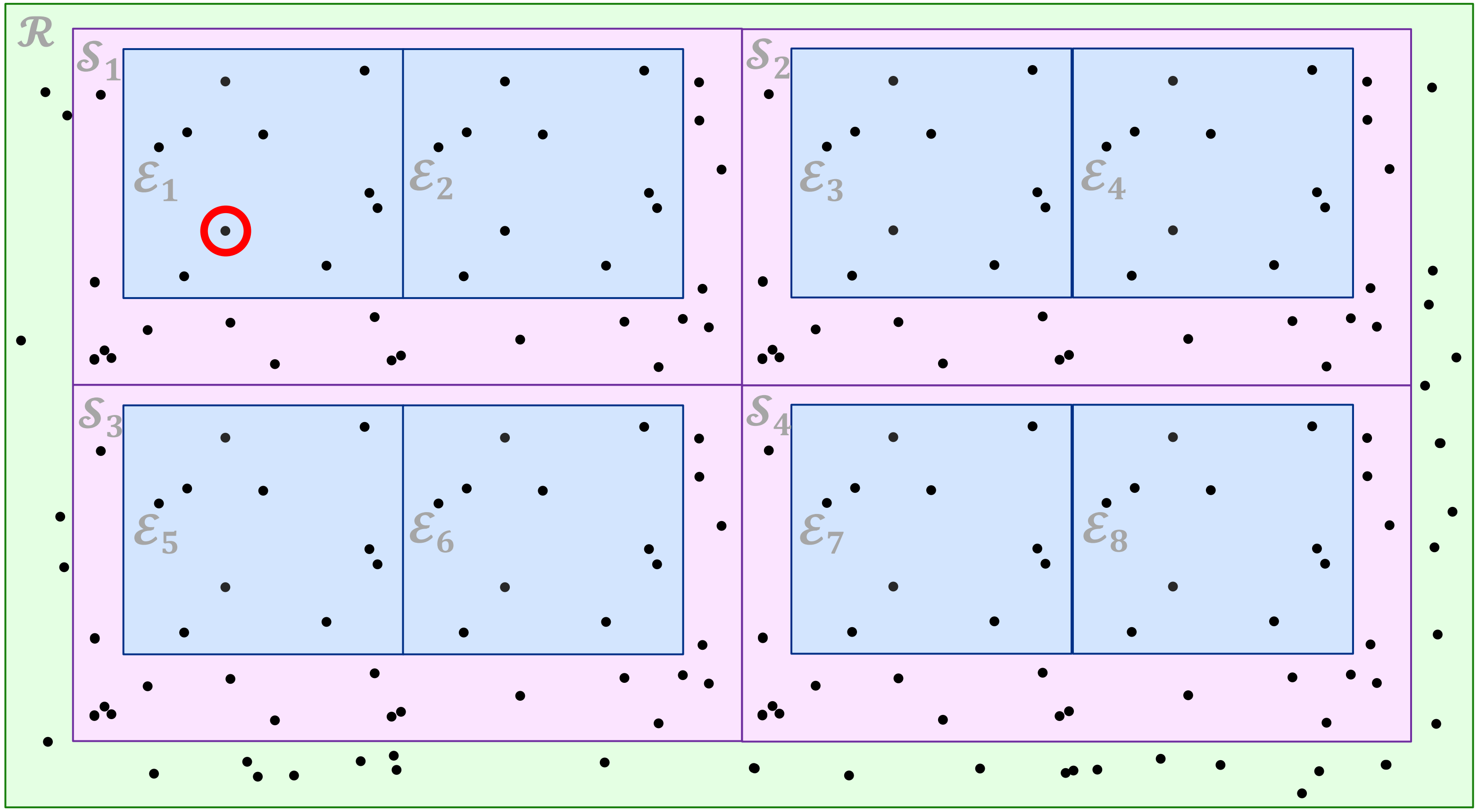
Looking at the data (objects)



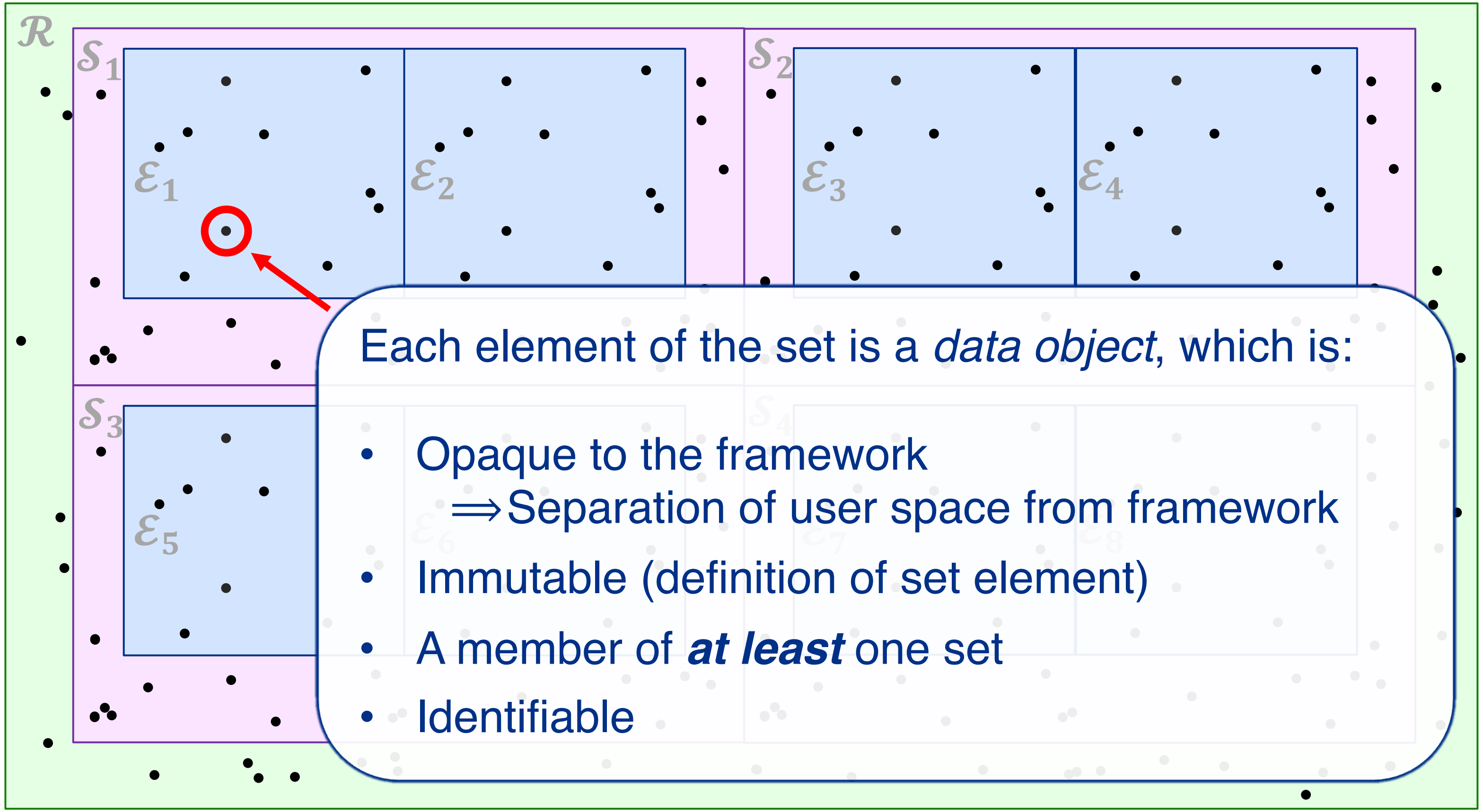
Looking at the data (objects)



Looking at the data (objects)



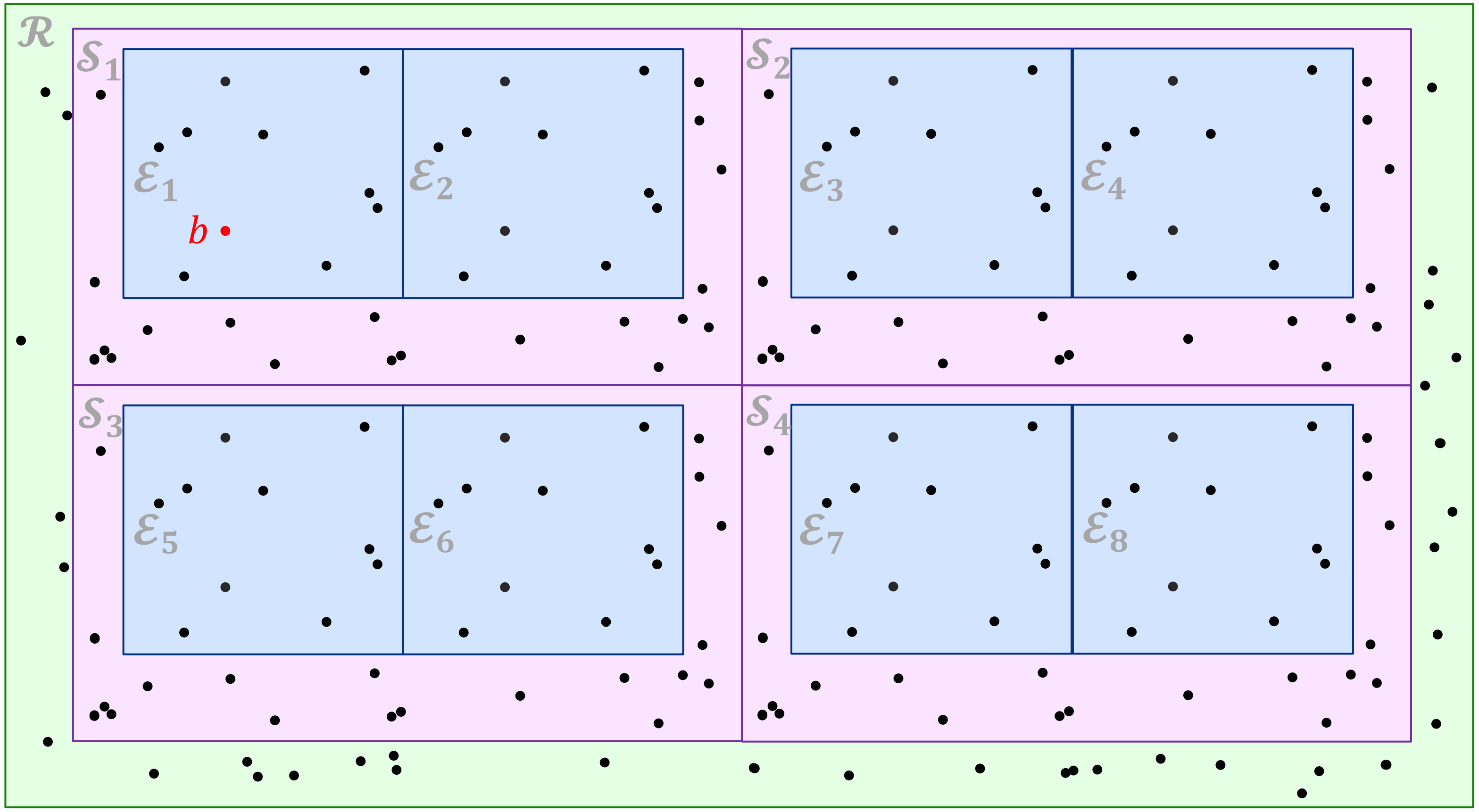
Looking at the data (objects)



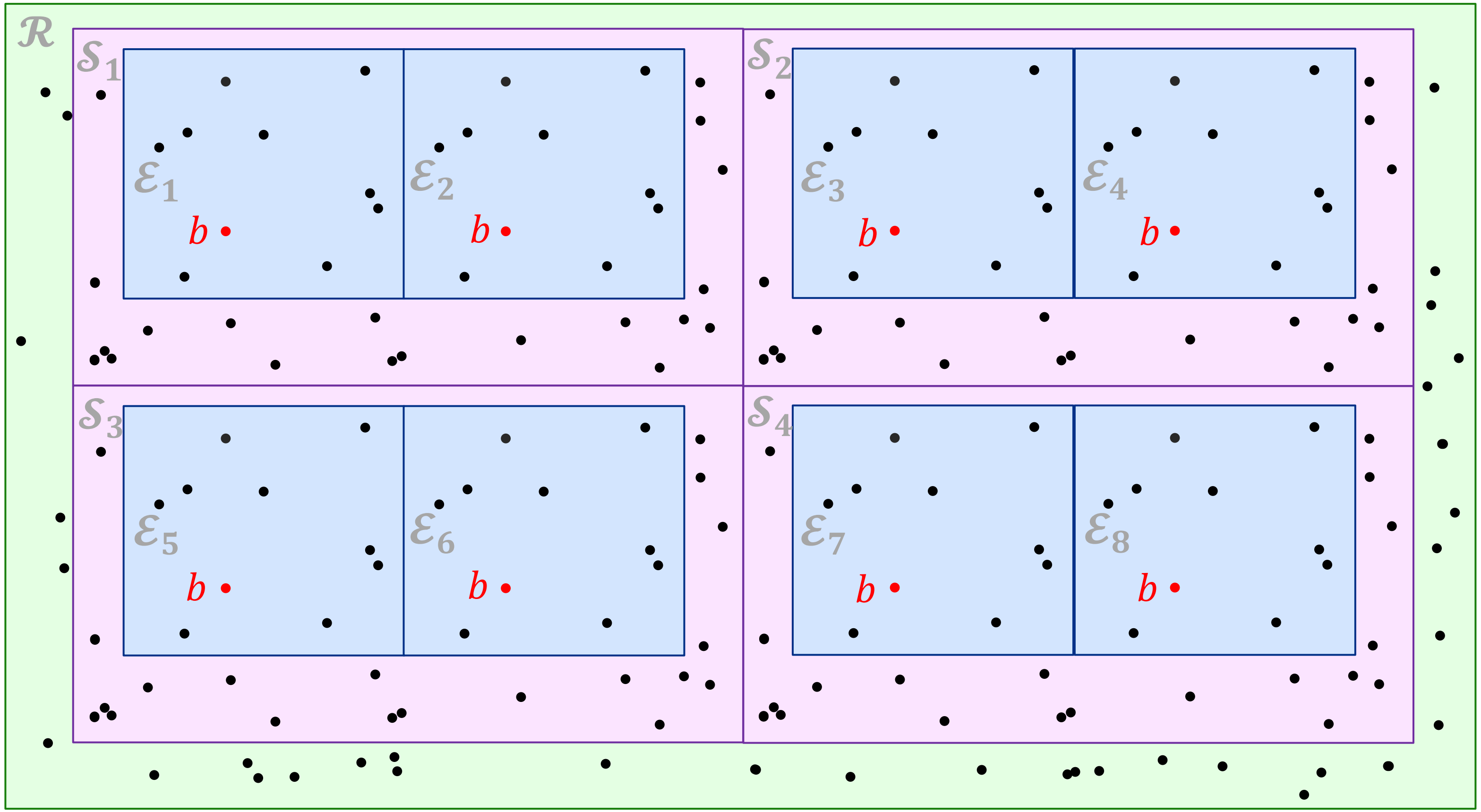
Each element of the set is a *data object*, which is:

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

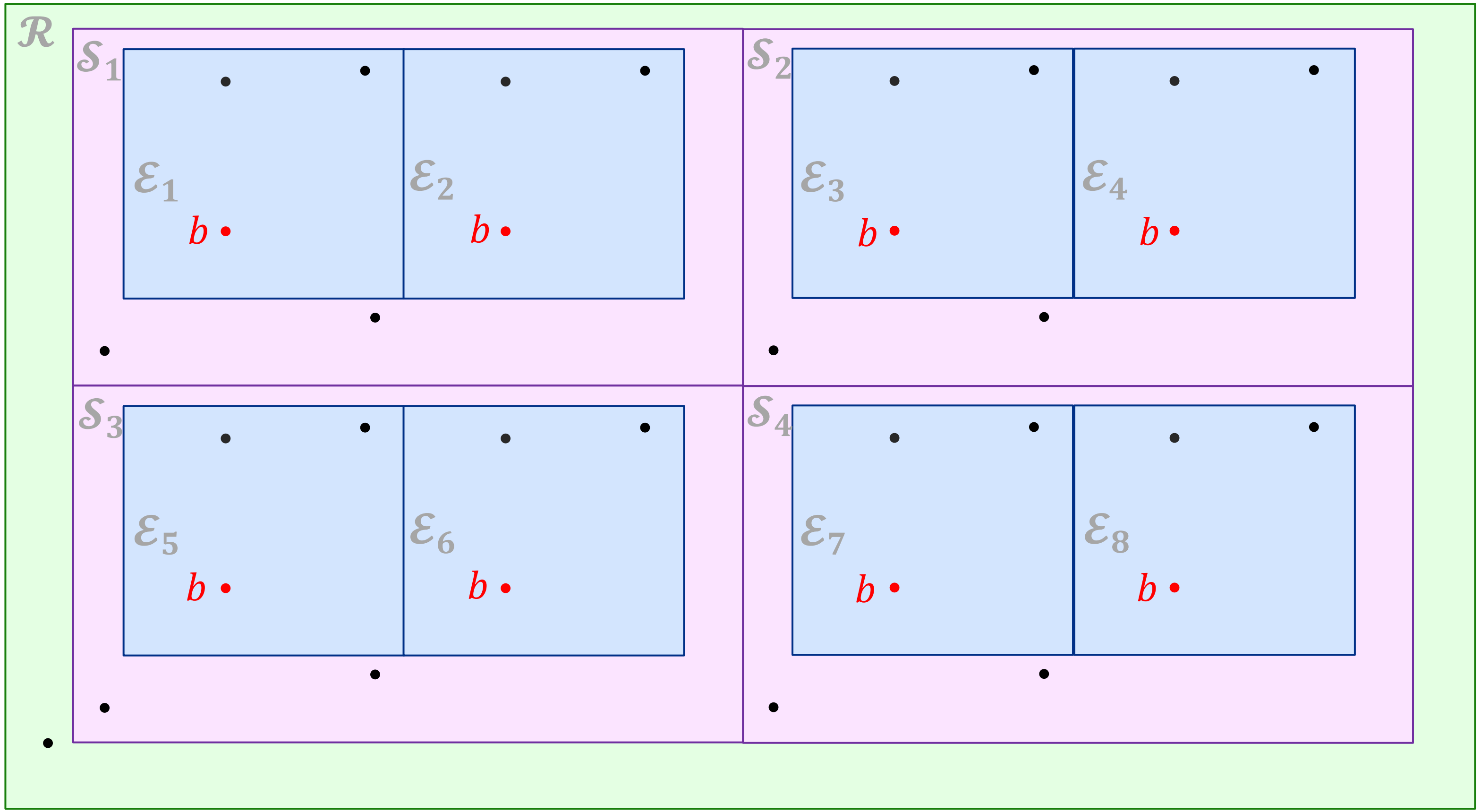
Looking at the data (objects)



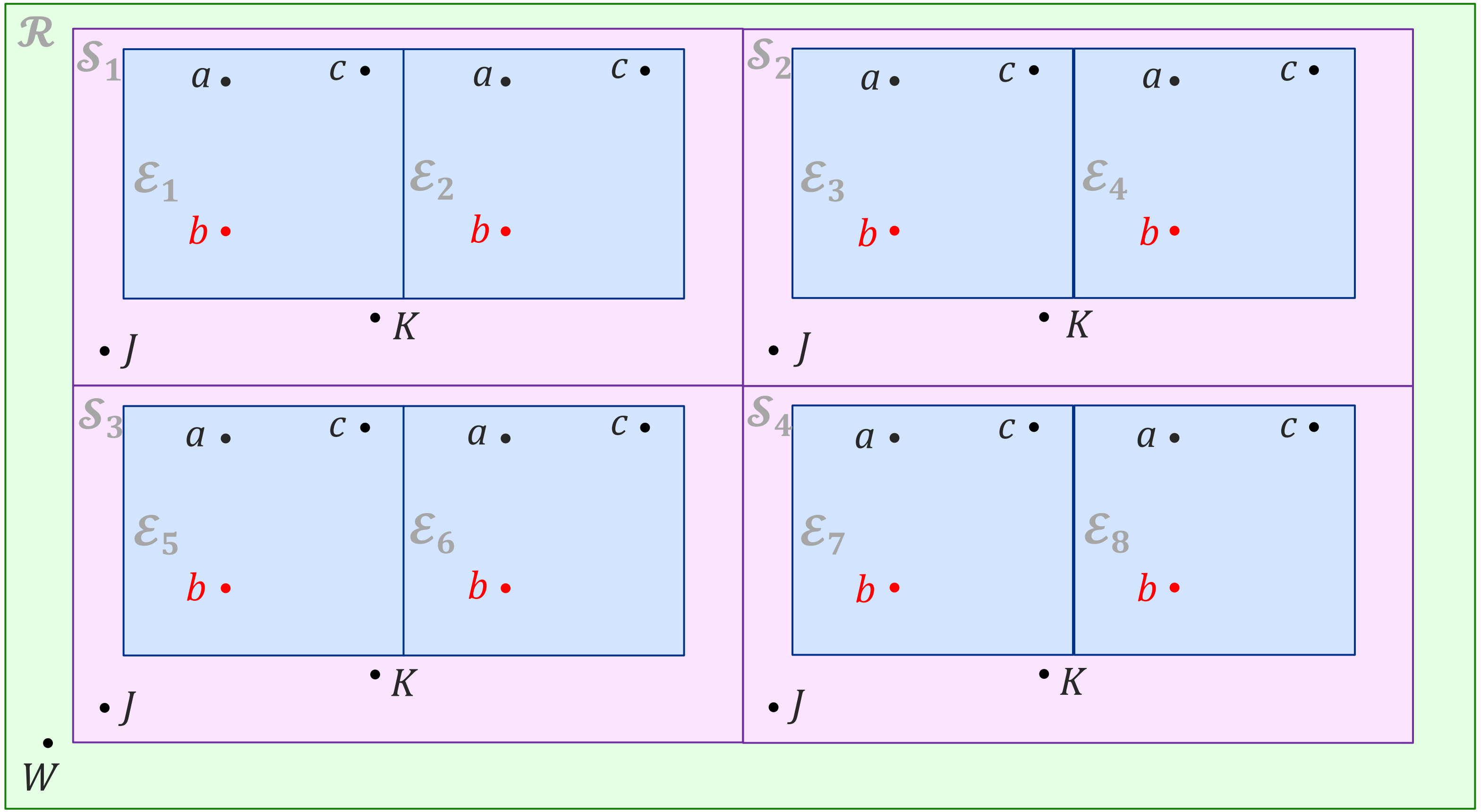
Looking at the data (objects)



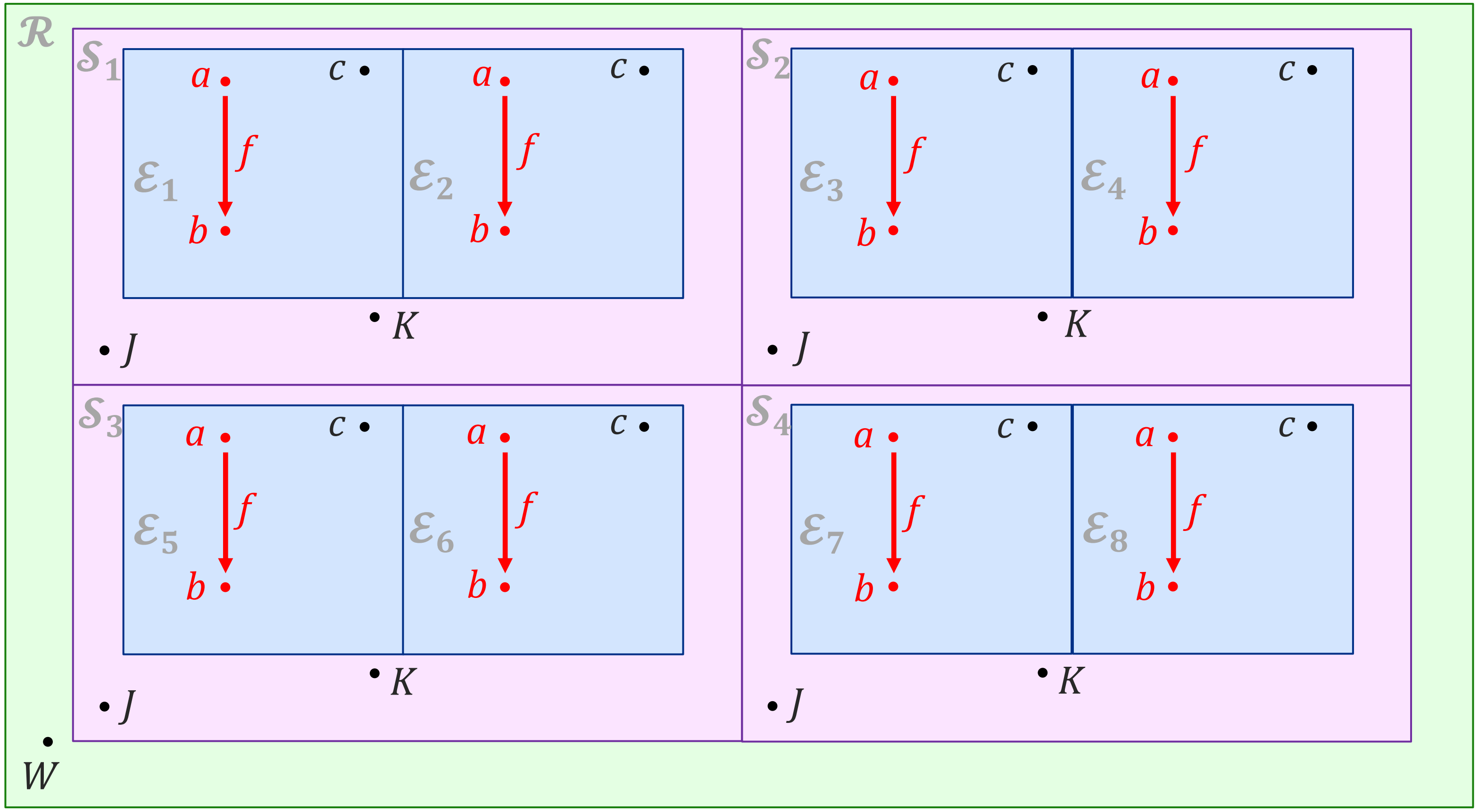
Looking at the data (objects)



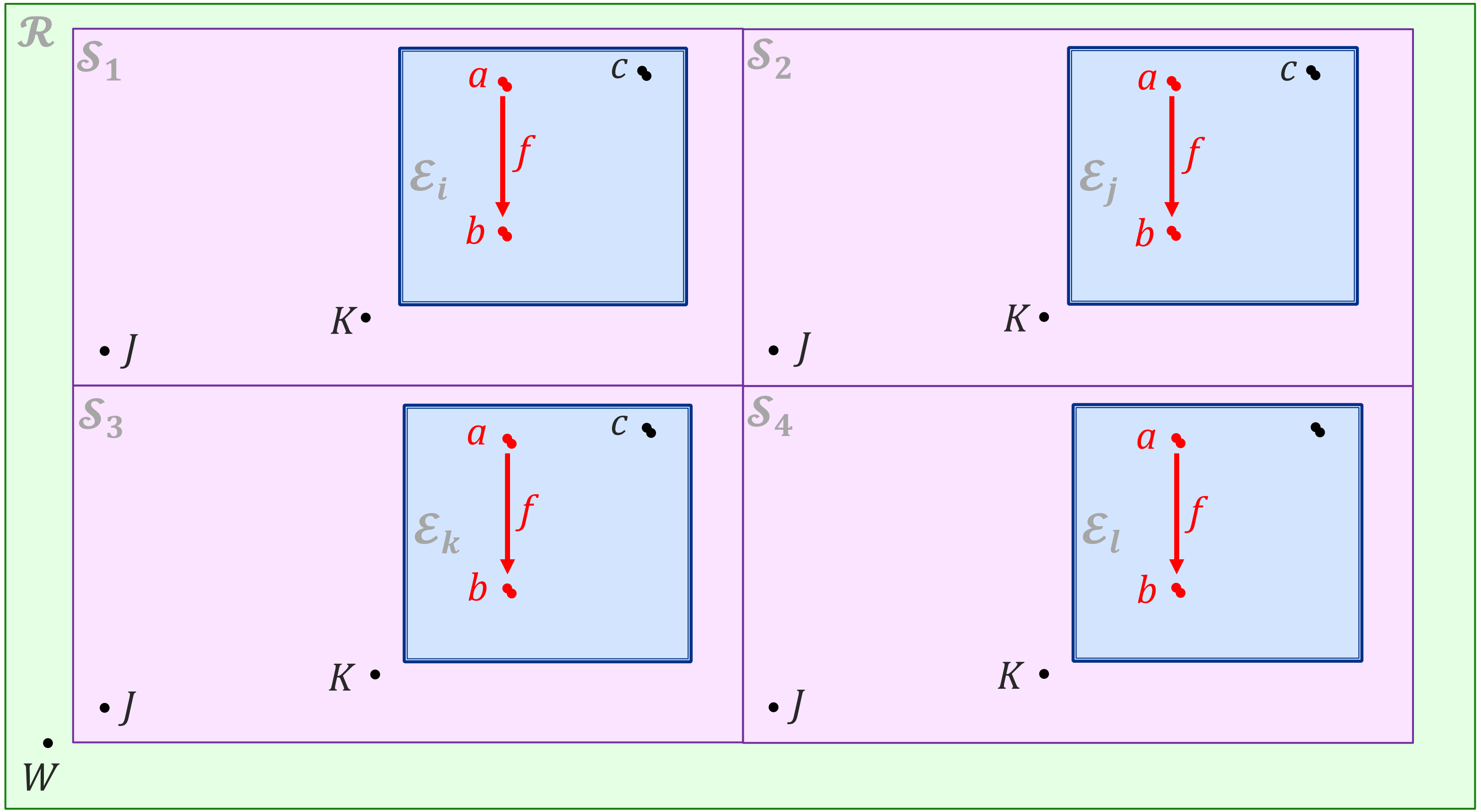
Looking at the data (objects)



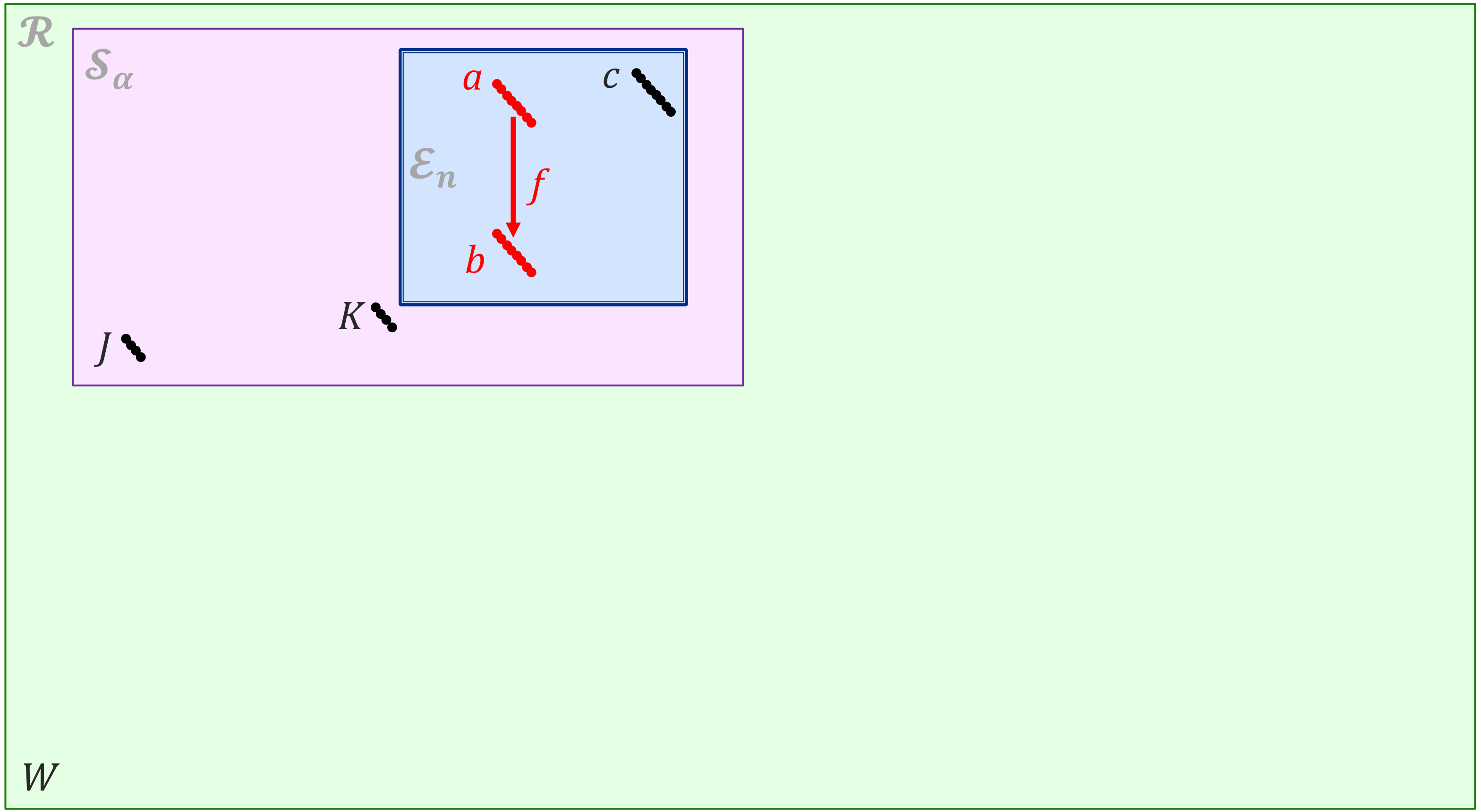
Looking at the data (object mappings)



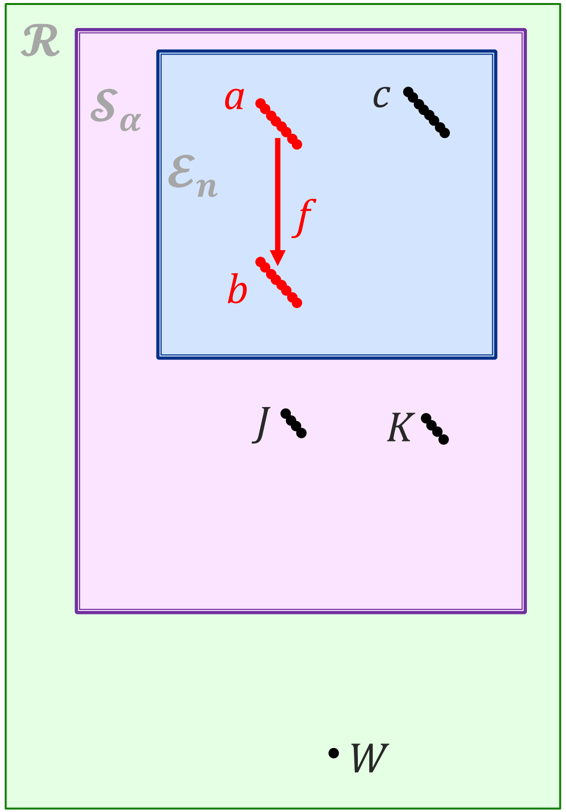
Looking at the data (object sequences)



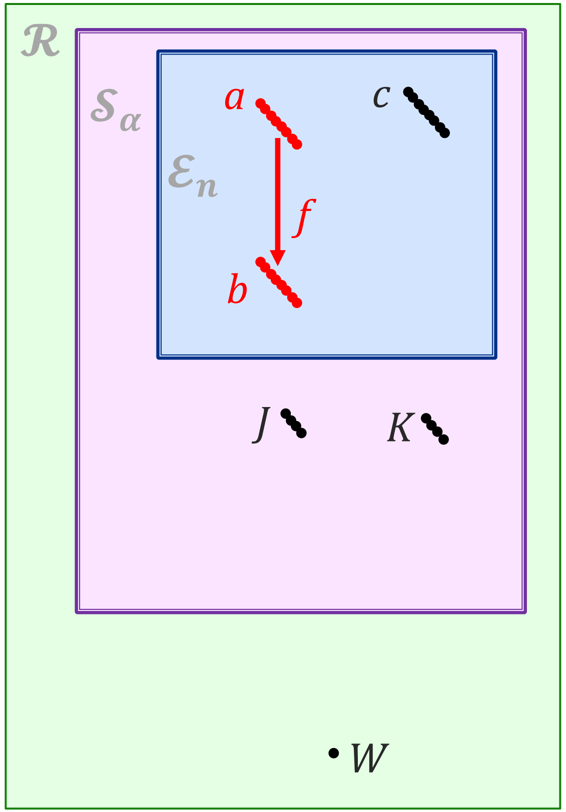
Looking at the data (object sequences)



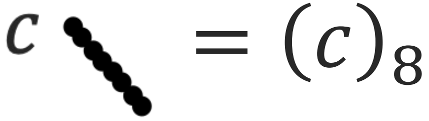
Looking at the data (object sequences)



Looking at the data (object sequences)



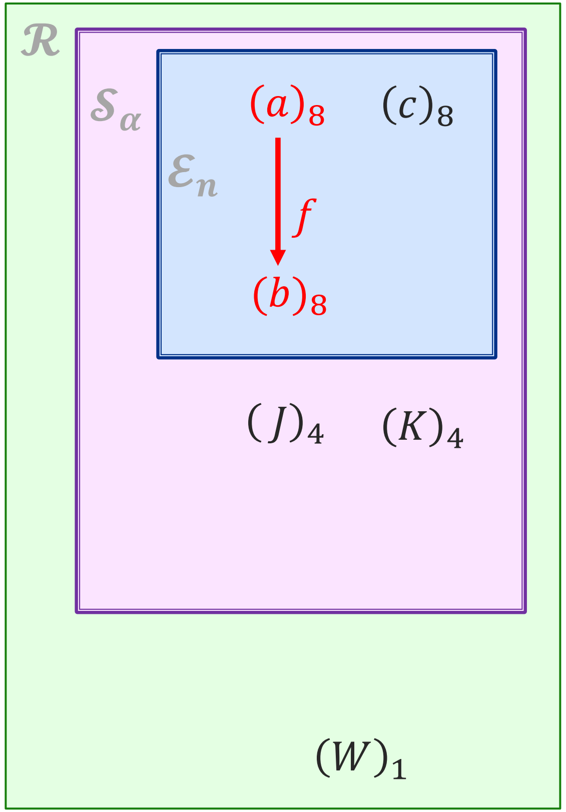
We can make the following replacement (e.g.):



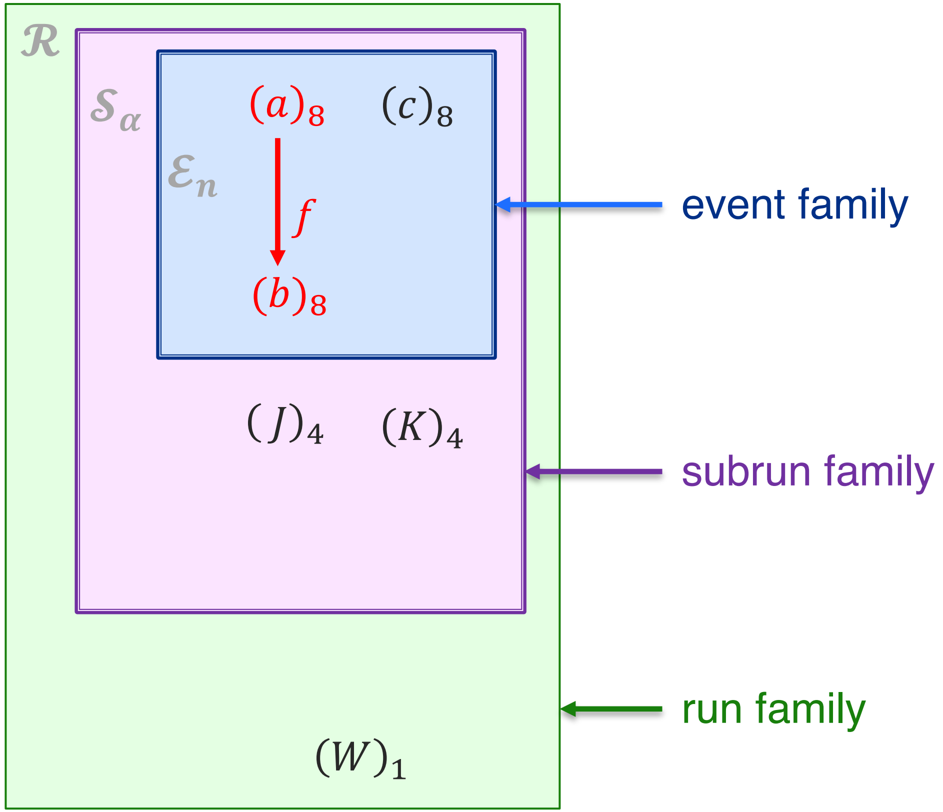
$$c \;= (c)_8$$

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

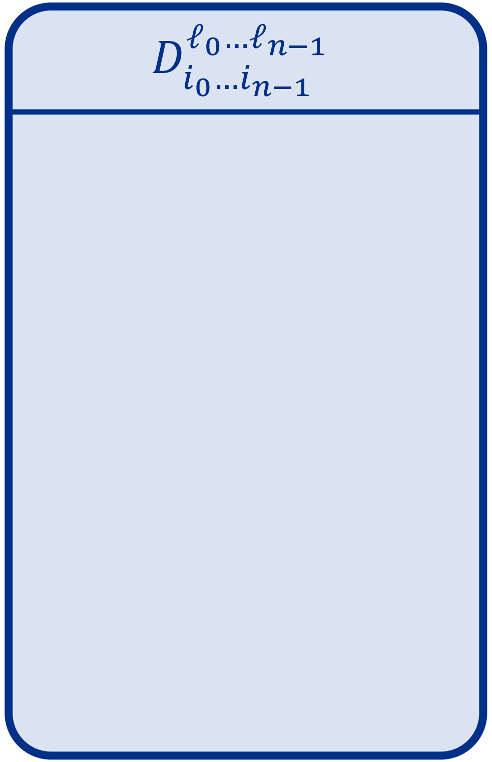
Looking at the data (object sequences)



Looking at the data (object sequences)



Data set



A data set $D^{\ell_0 \ldots \ell_{n-1}}_{i_0 \ldots i_{n-1}}$ is an indexed set where $\ell \in \mathcal{L}$, the label set, and $i \in \mathbb{N}$.

Example: For art, the following labels always hold:

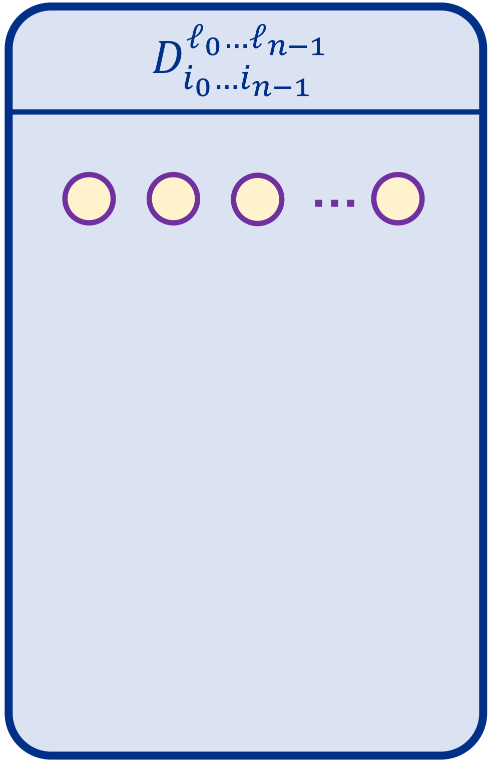
$$\ell_0 = \text{run},\ \ \ell_1 = \text{subrun},\ \ell_2 = \text{event}$$

The event identified by (3,5,9) can be written as $D^{\text{run,subrun,event}}_{3,5,9}$.

Abbreviations allowed when the context is unambiguous (e.g.):

$$D^{\text{run,subrun,event}}_{3,5,9} = E_{3,5,9}\ (\subset S_{3,5} \subset R_3)$$

Data set

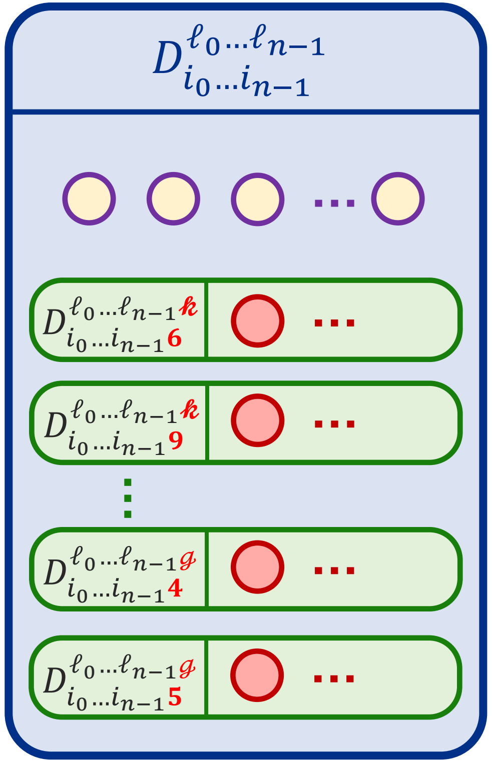


A data set contains the following elements:

- Zero or more data objects, and

Fermilab

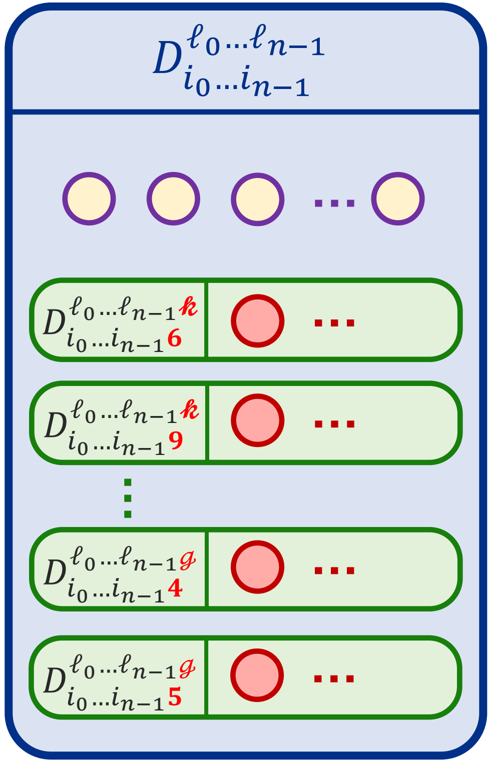
Data set


$D^{\ell_0 \dots \ell_{n-1}}_{i_0 \dots i_{n-1}}$
$D^{\ell_0 \dots \ell_{n-1} \mathscr{k}}_{i_0 \dots i_{n-1} 6}$
$D^{\ell_0 \dots \ell_{n-1} \mathscr{k}}_{i_0 \dots i_{n-1} 9}$
$D^{\ell_0 \dots \ell_{n-1} \mathscr{g}}_{i_0 \dots i_{n-1} 4}$
$D^{\ell_0 \dots \ell_{n-1} \mathscr{g}}_{i_0 \dots i_{n-1} 5}$

A data set contains the following elements:

- Zero or more data objects, and
- Zero or more nested data sets $\{D^{\ell_0 \dots \ell_{n-1} \mathscr{k}}_{i_0 \dots i_{n-1} j}\}_{j \in \mathbb{N}}$ where $\mathscr{k} \in \mathcal{L}$.

Data set



$D^{\ell_0 \dots \ell_{n-1}}_{i_0 \dots i_{n-1}}$
$D^{\ell_0 \dots \ell_{n-1} \mathcal{k}}_{i_0 \dots i_{n-1} 6}$
$D^{\ell_0 \dots \ell_{n-1} \mathcal{k}}_{i_0 \dots i_{n-1} 9}$
$D^{\ell_0 \dots \ell_{n-1} \mathcal{g}}_{i_0 \dots i_{n-1} 4}$
$D^{\ell_0 \dots \ell_{n-1} \mathcal{g}}_{i_0 \dots i_{n-1} 5}$

A data set contains the following elements:

- Zero or more data objects, and
- Zero or more nested data sets $\{D^{\ell_0 \dots \ell_{n-1} \mathcal{k}}_{i_0 \dots i_{n-1} j}\}_{j \in \mathbb{N}}$ where $\mathcal{k} \in \mathcal{L}$.

Two observations:

1. For a given family of data sets, the value of $\mathcal{k}$ is constant, whereas j varies over the set of natural numbers.
2. The values of j do not need to be dense

Higher-order functions

- Higher-order functions either take a function as an argument or return a function.
- We are interested in higher-order functions that take one user-provided function f that is applied to each data object a in sequence of n objects $(a)_n$.

$$(a)_n \xrightarrow{f} (b)_m$$

- Each object a corresponds to (a tuple of) arguments passed to f.
- The signature of f and the value $f(a)$ depend on the higher-order function.
- The sequence $(a)_n$ is formed from objects within a data family.

Higher-order function examples

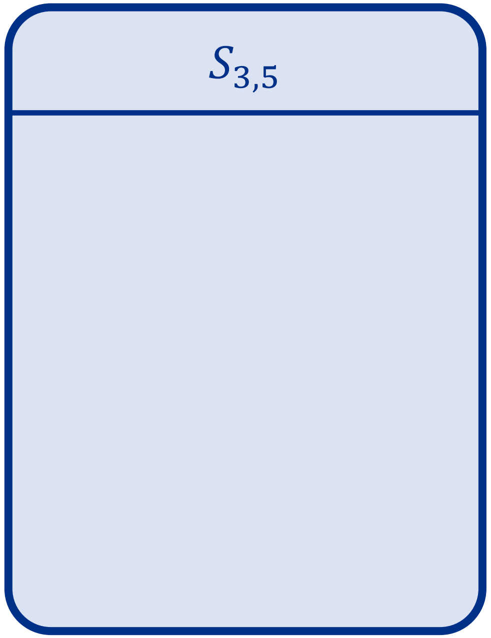
Meld term	CS term	Mathematical description	
Transform	Map	$(a)_n \xrightarrow{f} (b)_n$	where $f(a) \rightarrow b$
Filter	Filter	$(a)_n \xrightarrow{f} (a)_m$ where $m \leq n$	where $f(a) \rightarrow \mathrm{Boolean}$
Monitor	—	$(a)_n \xrightarrow{f} (\ )_0$	where $f(a) \rightarrow \mathrm{Void}$
Reduction	Fold	$(a)_n \xrightarrow{f} c$	where $f(a, c) \rightarrow c$
Splitter	Unfold	$a \xrightarrow{f} (d)_m$	where $f(a) \rightarrow (d)_\mathrm{n}$

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 $f(h) = t$.

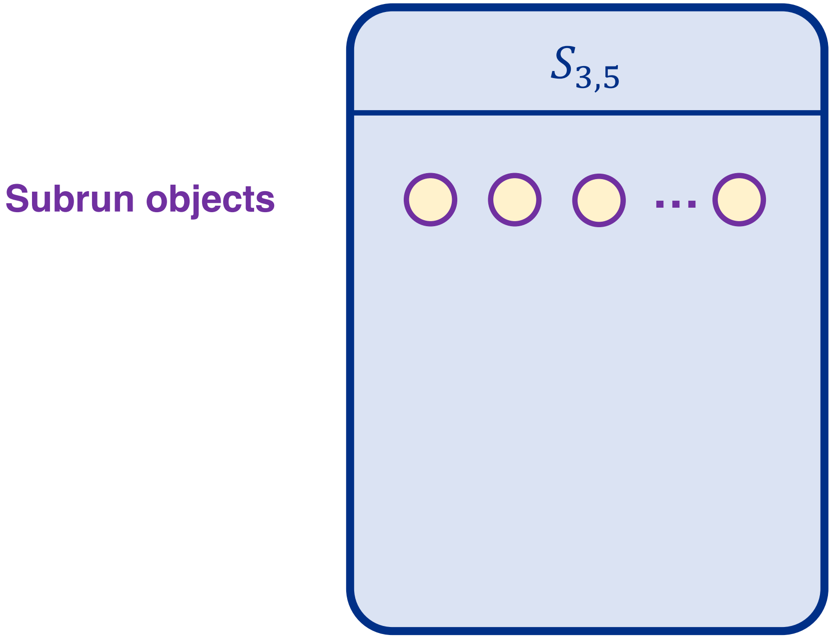
Transform example

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



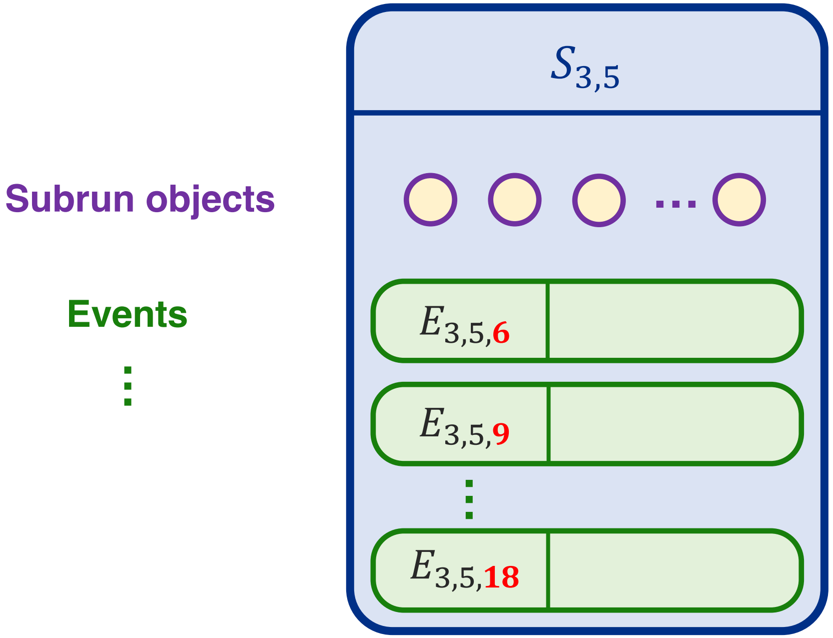
Transform example

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



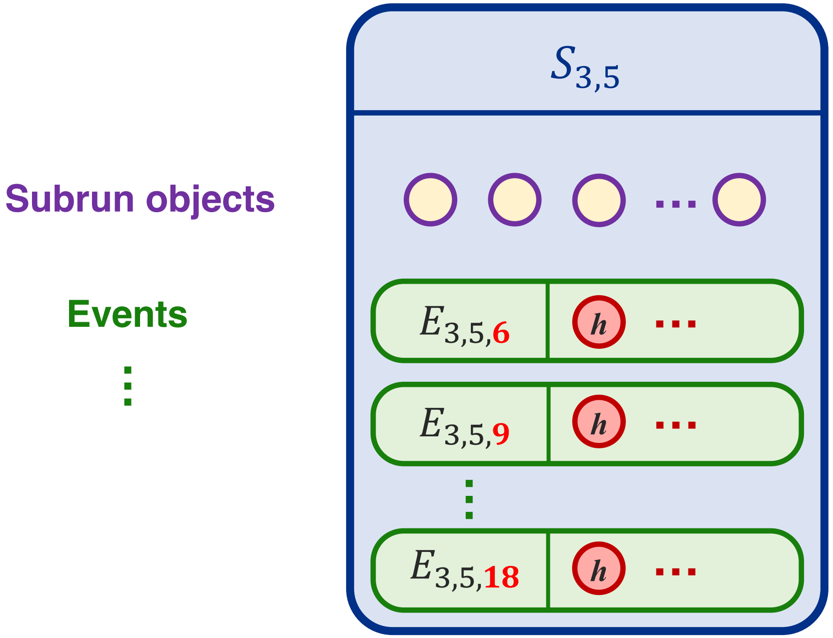
Transform example

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



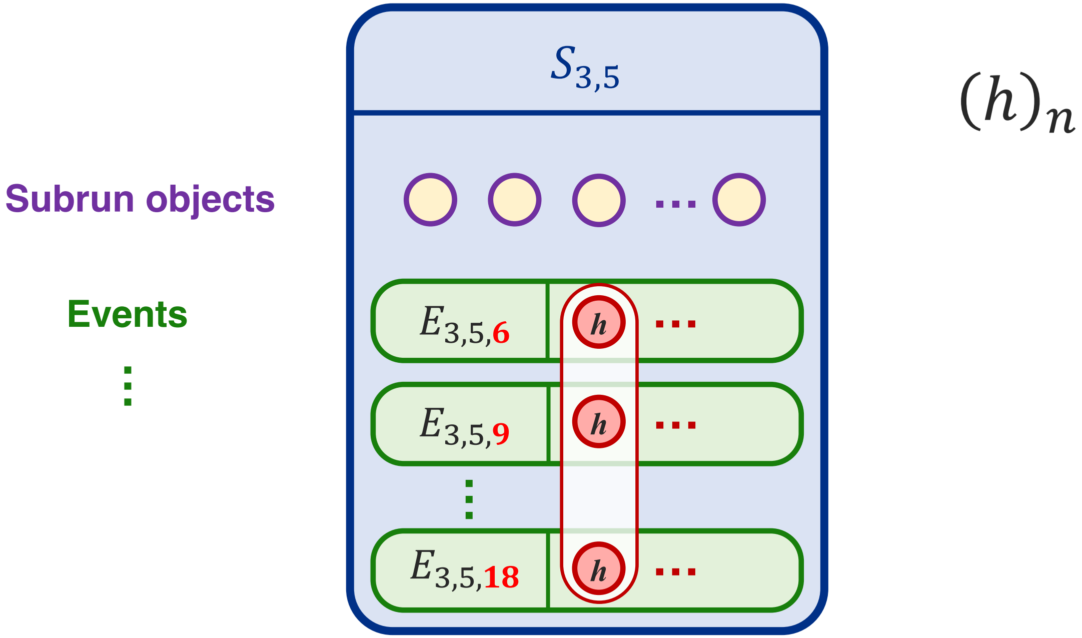
Transform example

- 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 $f(h) = t$.



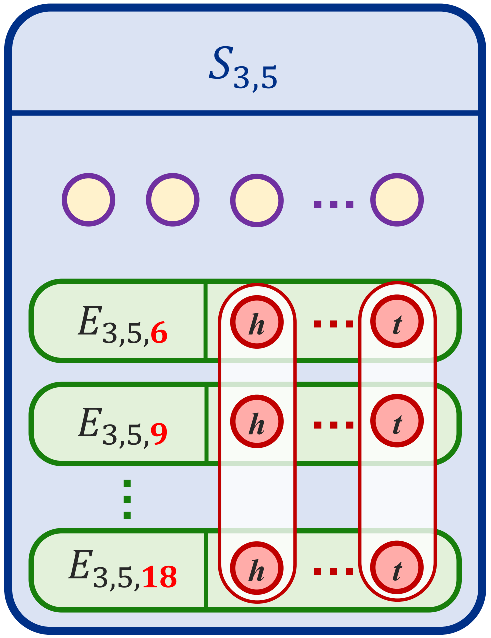
Transform example

- 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 objects

Events

⋮



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

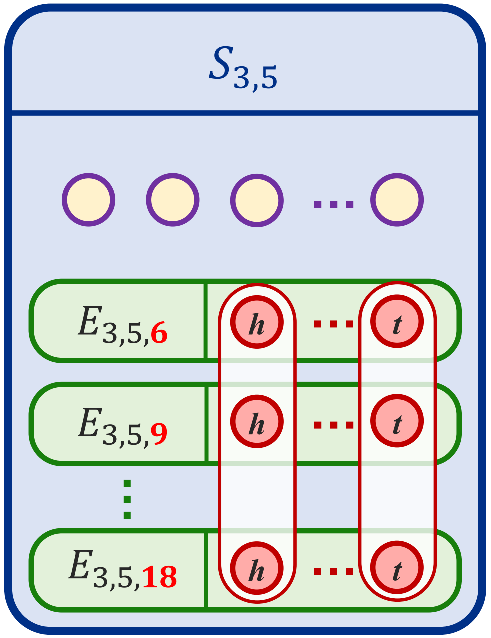
Transform example

- 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 objects

Events

⋮



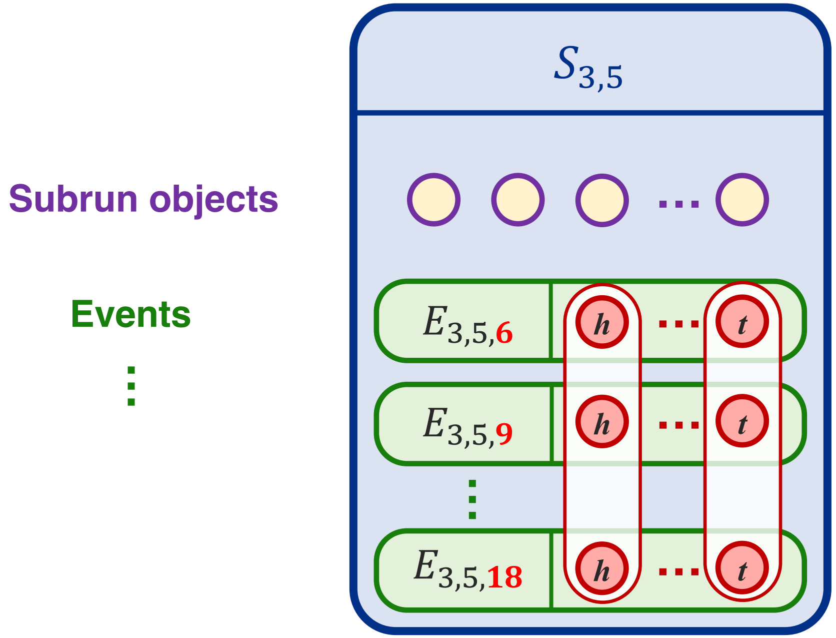
$$(h)_n \xrightarrow{f} (t)_n$$

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

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

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 $f(h) = t$.



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

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.