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

14 December 2022

Framework-supported algorithm constructs

Constructs that can be framework-agnostic:

- **Transform** (producer) – creates data products from existing data of the same processing level
- **Reduction** (producer) – creates data products based on accumulations of data at a more granular processing level (e.g. endSubRun)
- **Monitor** (analyzer) – consumes data products and does not produce any new data
- **Filter** – supports processing a subset of data based on satisfying Boolean criteria

Framework-supported algorithm constructs

Constructs that can be framework-agnostic:

- **Transform** (producer) – creates data products from existing data of the same processing level
- **Reduction** (producer) – creates data products based on accumulations of data at a more granular processing level (e.g. endSubRun)
- **Monitor** (analyzer) – consumes data products and does not produce any new data
- **Filter** – supports processing a subset of data based on satisfying Boolean criteria

Constructs that must be framework-aware:

- **Source** – creates product stores that provide data products
- **Splitter** – splits existing product stores into smaller ones for downstream processing
- **Output** – writes product stores to an output file, stream, etc.

Framework glue code

- **Naturally separates user code from framework assumptions**
  - The input arguments for each registered function are data products produced by upstream functions or provided by the input source.
  - The return value(s) of each registered function are registered as a data product.

Framework glue code

- **Naturally separates user code from framework assumptions**
  - The input arguments for each registered function are data products produced by upstream functions or provided by the input source.
  - The return value(s) of each registered function are registered as a data product.

```
constexpr int add(int i, int j) { return i + j; }
```

Framework glue code

- **Naturally separates user code from framework assumptions**
  - The input arguments for each registered function are data products produced by upstream functions or provided by the input source.
  - The return value(s) of each registered function are registered as a data product.

```
constexpr int add(int i, int j) { return i + j; }


#include "meld/module.hpp"

DEFINE_MODULE(m) // pset can also be passed in
{
  m.declare_transform("add", add)
   .concurrency(unlimited)
   .input(”num", ”neg_num")
   .output("sum");
}
```

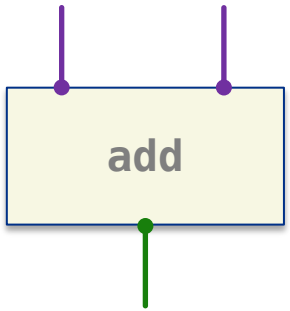
Framework glue code

- **Naturally separates user code from framework assumptions**
  - The input arguments for each registered function are data products produced by upstream functions or provided by the input source.
  - The return value(s) of each registered function are registered as a data product.

```
constexpr int add(int i, int j) { return i + j; }
```

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

DEFINE_MODULE(m) // pset can also be passed in
{
  m.declare_transform("add", add)
   .concurrency(unlimited)
   .input("num", "neg_num")
   .output("sum");
}
```

Generates graph node



Fermilab

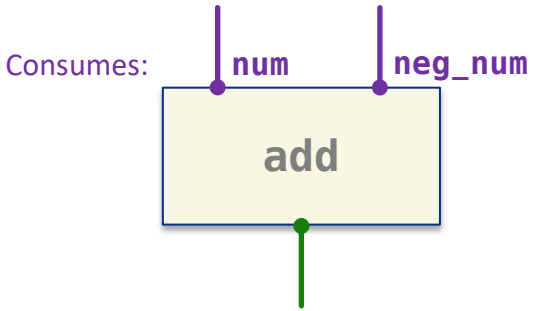
Framework glue code

- **Naturally separates user code from framework assumptions**
  - The input arguments for each registered function are data products produced by upstream functions or provided by the input source.
  - The return value(s) of each registered function are registered as a data product.

```
constexpr int add(int i, int j) { return i + j; }
```

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

DEFINE_MODULE(m) // pset can also be passed in
{
  m.declare_transform("add", add)
   .concurrency(unlimited)
   .input("num", "neg_num")
   .output("sum");
}
```

Generates graph node



Fermilab

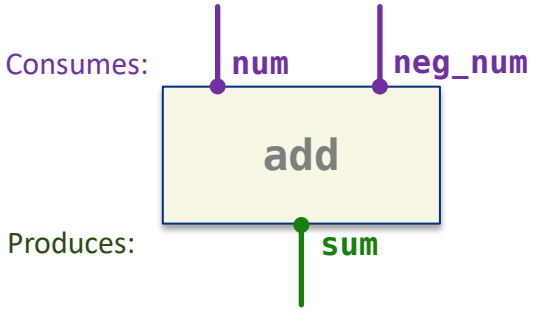
Framework glue code

- **Naturally separates user code from framework assumptions**
  - The input arguments for each registered function are data products produced by upstream functions or provided by the input source.
  - The return value(s) of each registered function are registered as a data product.

```
constexpr int add(int i, int j) { return i + j; }
```

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

DEFINE_MODULE(m) // pset can also be passed in
{
  m.declare_transform("add", add)
   .concurrency(unlimited)
   .input("num", "neg_num")
   .output("sum");
}
```

Generates graph node



Fermilab

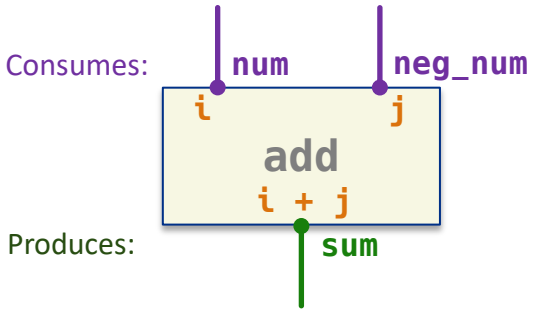
Framework glue code

- **Naturally separates user code from framework assumptions**
  - The input arguments for each registered function are data products produced by upstream functions or provided by the input source.
  - The return value(s) of each registered function are registered as a data product.

```
constexpr int add(int i, int j) { return i + j; }
```

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

DEFINE_MODULE(m) // pset can also be passed in
{
  m.declare_transform("add", add)
   .concurrency(unlimited)
   .input("num", "neg_num")
   .output("sum");
}
```

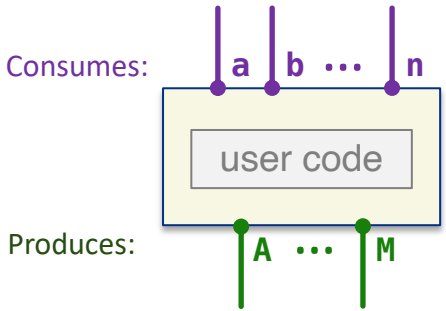
Generates graph node



Fermilab

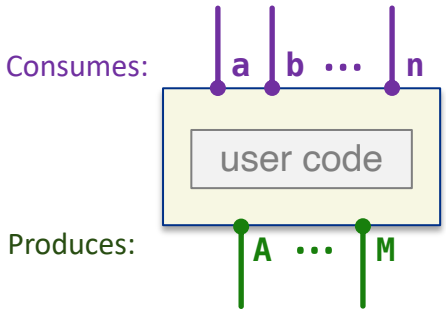
Graph nodes

- For *framework-agnostic constructs*, framework details **are not** accessed by the user within the node.

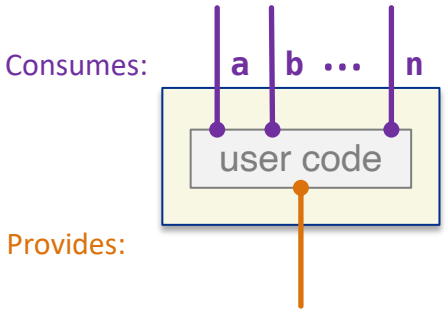


Graph nodes

- For *framework-agnostic constructs*, framework details **are not** accessed by the user within the node.



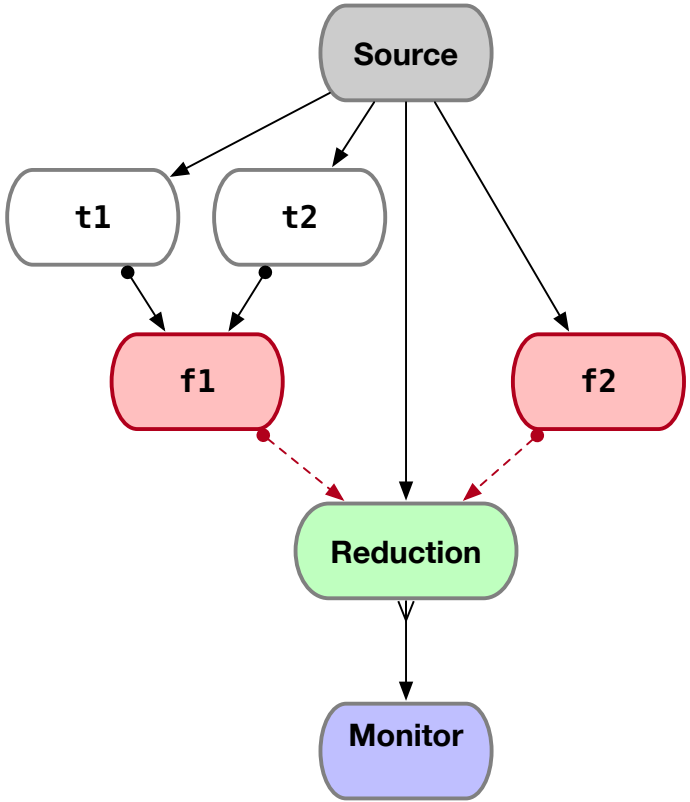
- For *framework-aware constructs*, framework details **must be** accessed by the user within the node.



Fermilab

Data flow

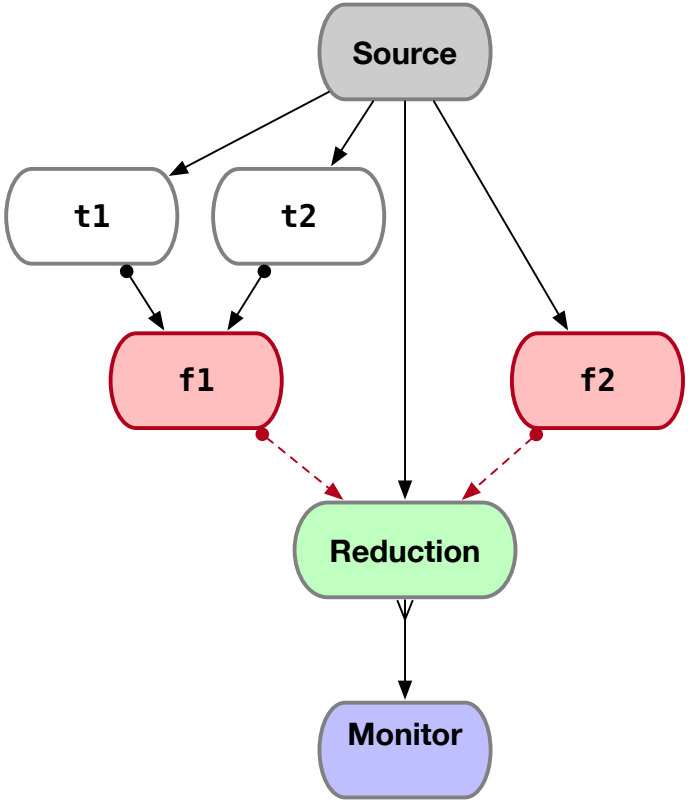
- Communication between nodes occurs via messages, which contain either:
  - (a) a product store (black lines), or
  - (b) a Boolean filter result (red, dashed lines)



Data flow

- Communication between nodes occurs via messages, which contain either:
  - (a) a product store (black lines), or
  - (b) a Boolean filter result (red, dashed lines)
- Product stores are shallow—the products from one store are not propagated to downstream stores.

N.B. **Products are immutable once committed to the framework.**



Fermilab

Data flow

- Communication between nodes occurs via messages, which contain either:

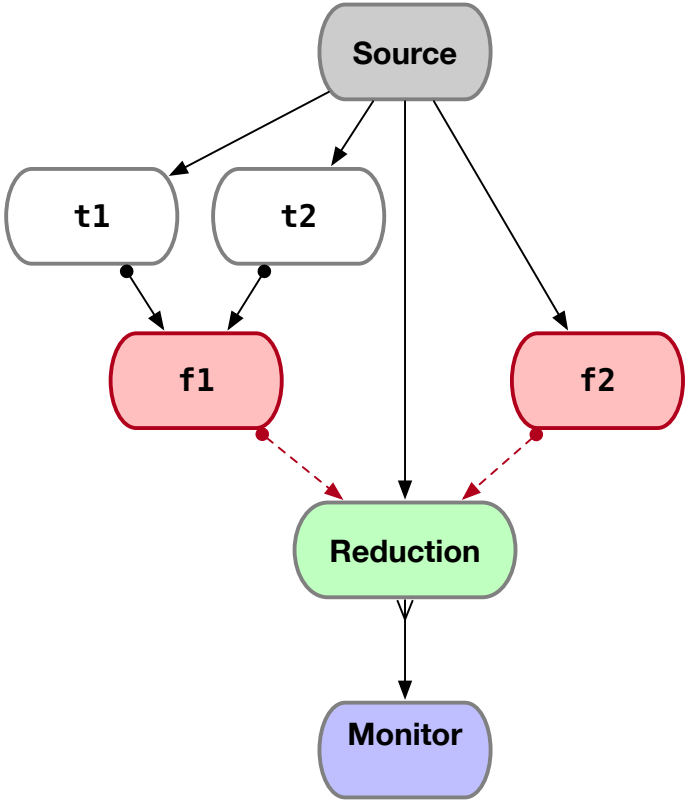
  (a) a product store (black lines), or

  (b) a Boolean filter result (red, dashed lines)

- Product stores are shallow—the products from one store are not propagated to downstream stores.

  N.B. Products are immutable once committed to the framework.

- Each non-filter generates its own product store with the products produced by that node.

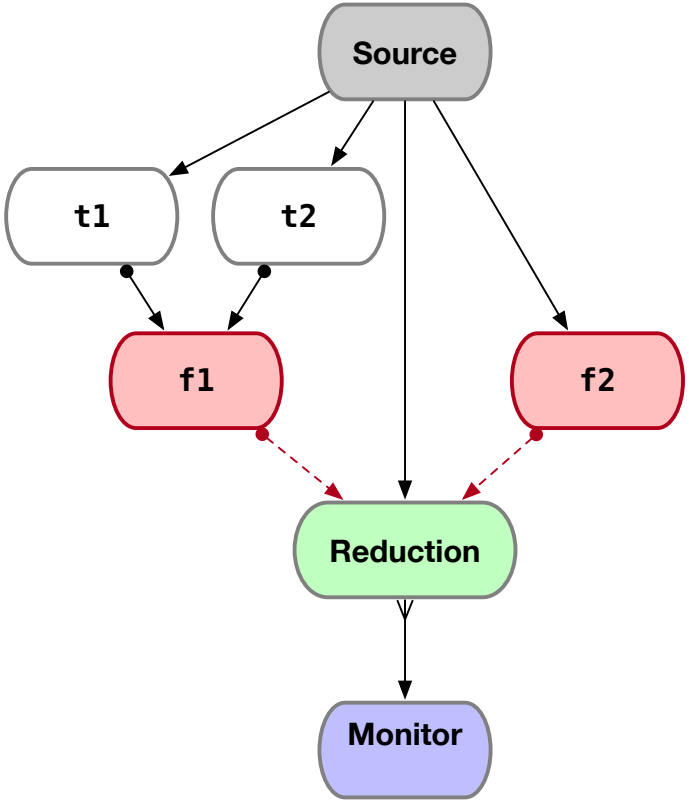


Fermilab

Data flow

- Communication between nodes occurs via messages, which contain either:
  - (a) a product store (black lines), or
  - (b) a Boolean filter result (red, dashed lines)
- Product stores are shallow—the products from one store are not propagated to downstream stores.

  N.B. **Products are immutable once committed to the framework.**
- Each non-filter generates its own product store with the products produced by that node.
- Product stores and filter results can be cached to support hierarchical data processing (dots).

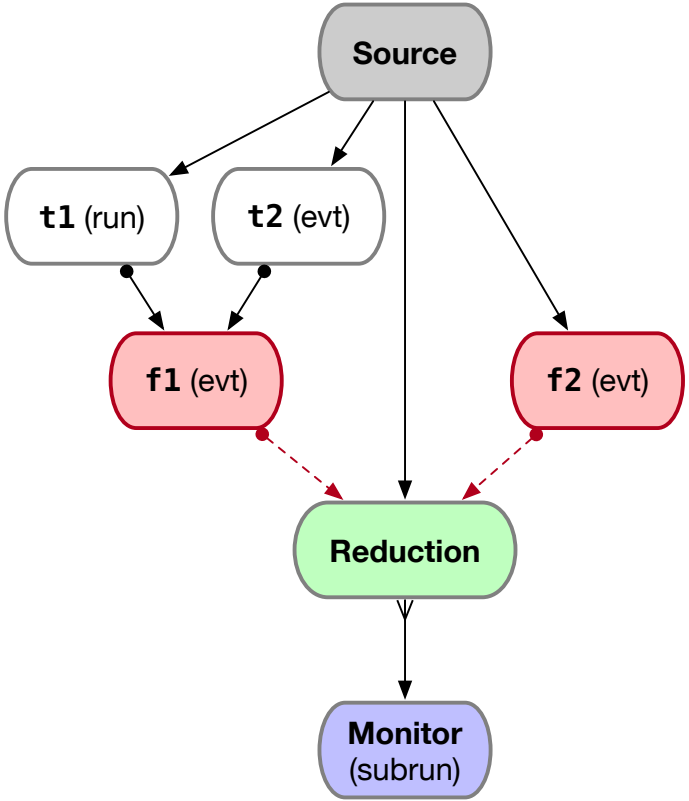


Fermilab

Data flow

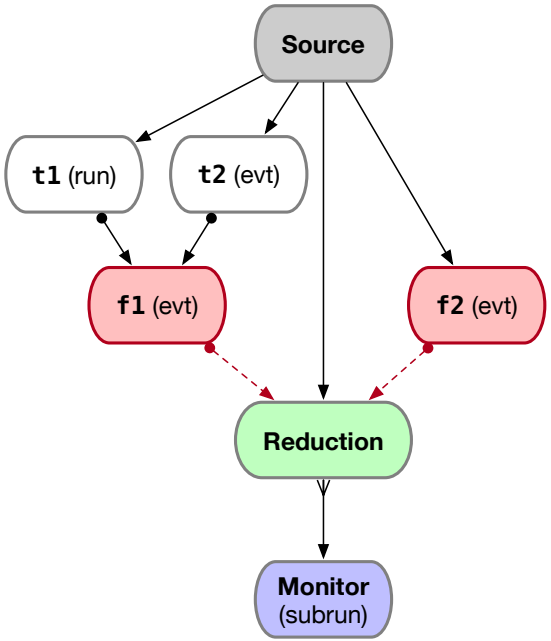
- Communication between nodes occurs via messages, which contain either:
  - (a) a product store (black lines), or
  - (b) a Boolean filter result (red, dashed lines)
- Product stores are shallow—the products from one store are not propagated to downstream stores.

  N.B. Products are immutable once committed to the framework.
- Each non-filter generates its own product store with the products produced by that node.
- Product stores and filter results can be cached to support hierarchical data processing (dots).



Configuration

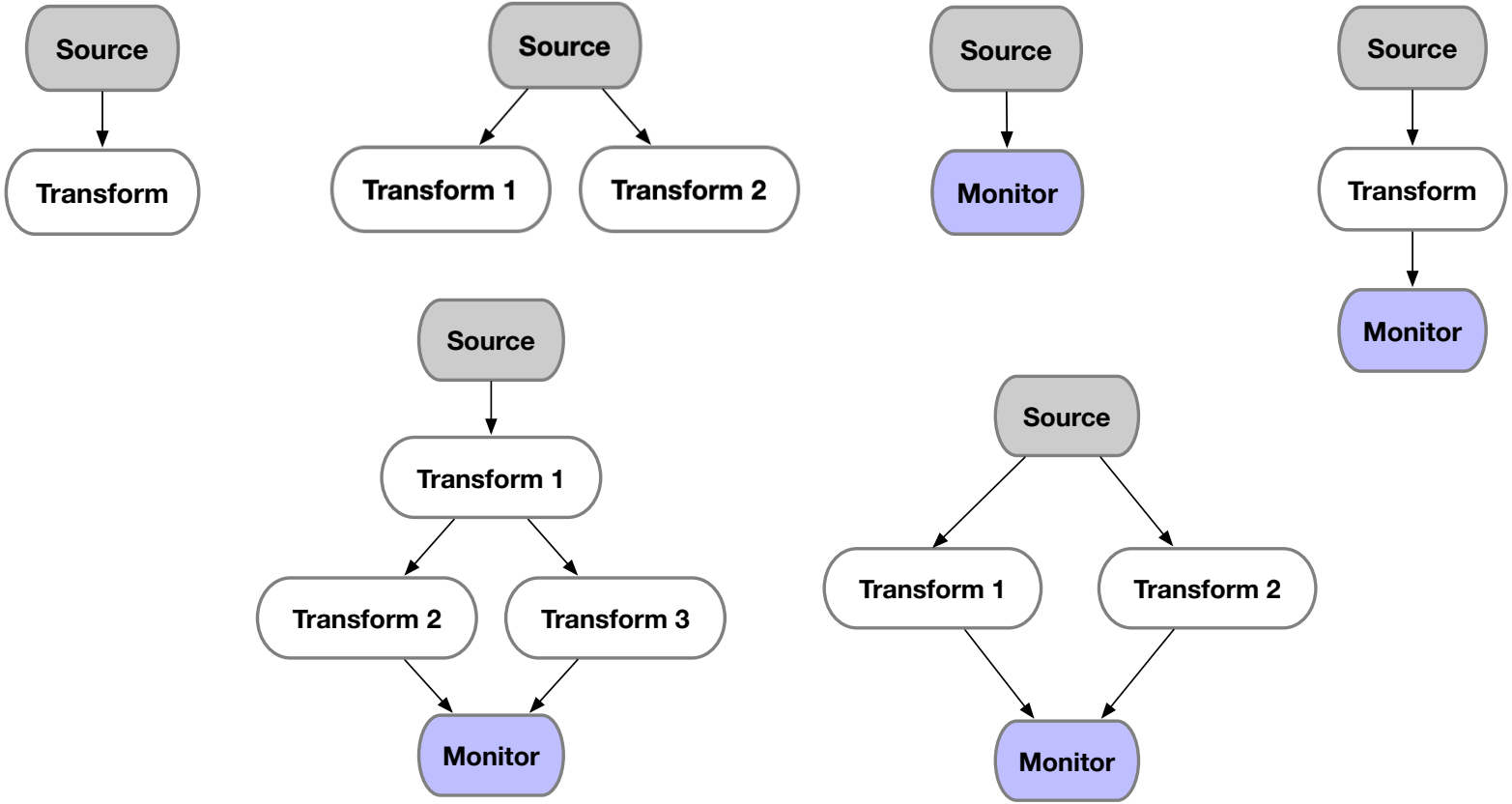
All modules (except sources) may specify preceding filters via configuration:




```
{
  source: { plugin: 'source_t' },
  modules: {
    t1: { plugin: 'transform_1' },
    t2: { plugin: 'transform_2' },
    f1: { plugin: 'filter_1' },
    f2: { plugin: 'filter_2' },
    reduction: {
      plugin: 'reduction',
      filtered_by: ['f1', 'f2'], # Logical AND of f1 and f2
    },
    monitor: { plugin: 'monitor' },
  },
}
```

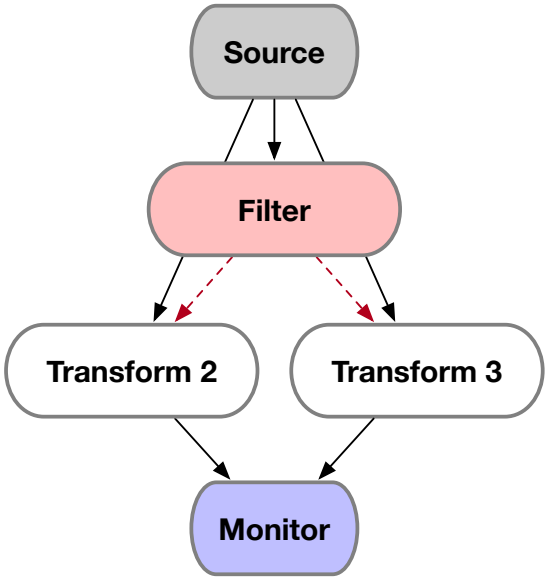
Data product dependencies specified by glue code.

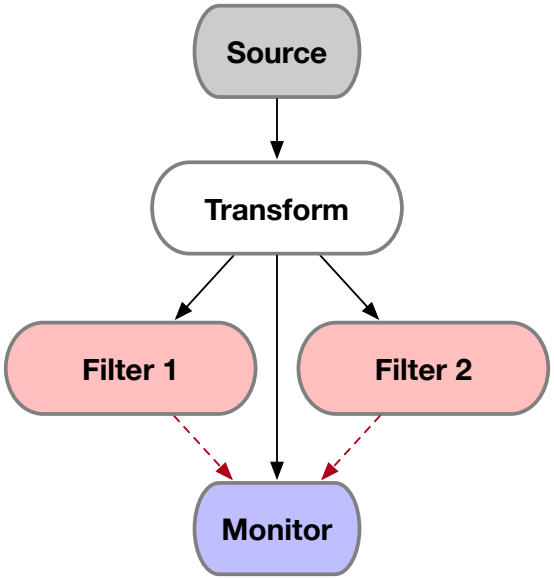
Benchmarks to test (transforms and monitors)

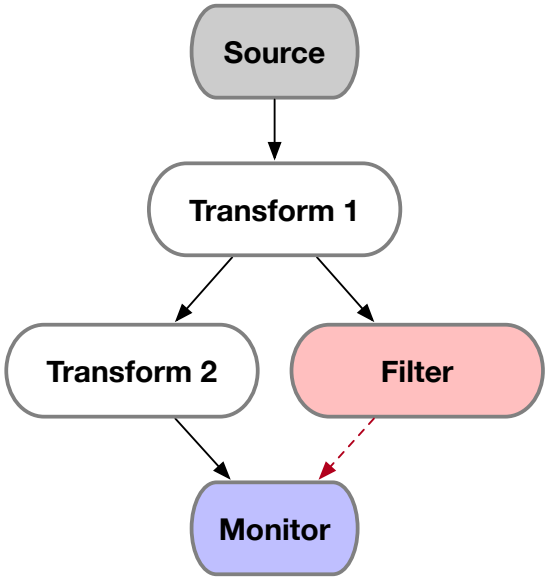


Fermilab

Benchmarks to test (filters)

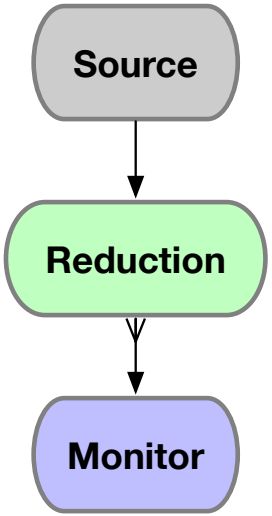


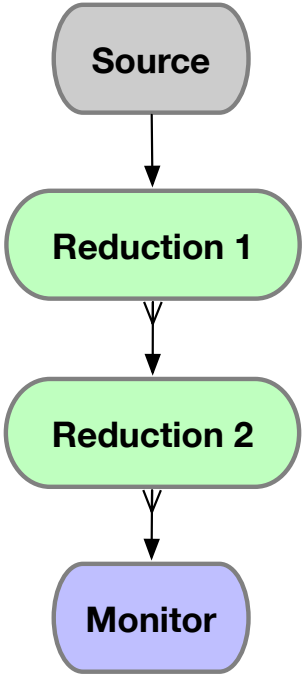




Fermilab

Benchmarks to test (reductions)

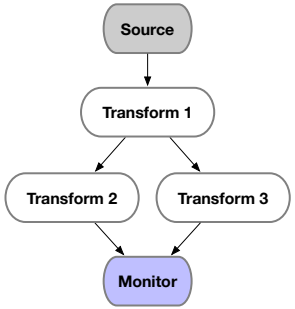




Next steps

- Run performance tests against *art* (started)

1M events



Next steps

- Run performance tests against *art* (started)

art

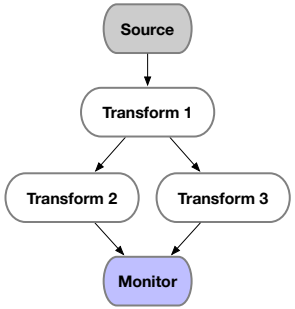
```
knoepfel@scisoftbuild01 benchmark-06.d $ art -c benchmark-06.fcl -j16 -n1000000

TrigReport ---------- Event summary ------------
TrigReport Events total = 1000000 passed = 1000000 failed = 0

TimeReport ---------- Time summary [sec] -------
TimeReport CPU = 342.198614 Real = 85.970735

MemReport  ---------- Memory summary [base-10 MB] ------
MemReport  VmPeak = 359.584 VmHWM = 59.6541
```

Meld

```
knoepfel@scisoftbuild01 build-meld $ ./test/benchmarks/bin/diamond_t
[2022-12-14 10:44:17.039] [info] Number of worker threads: 16
[2022-12-14 10:45:07.664] [info] CPU time: 726.58506s  Real time: 50.62818s  CPU efficiency: 1435.14%
[2022-12-14 10:45:07.664] [info] Max. RSS: 7.032 MB
```

Preliminary

1M events



Fermilab

Next steps

- Run performance tests against *art* (started)

art

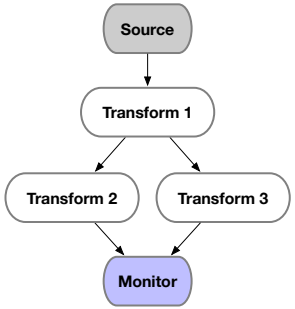
```
knoepfel@scisoftbuild01 benchmark-06.d $ art -c benchmark-06.fcl -j16 -n1000000

TrigReport ---------- Event summary -------------
TrigReport Events total = 1000000 passed = 1000000 failed = 0

TimeReport ---------- Time summary [sec] -------
TimeReport CPU = 342.198614 Real = 85.970735

MemReport  ---------- Memory summary [base-10 MB] ------
MemReport  VmPeak = 359.584 VmHWM = 59.6541
```

Preliminary

Meld

```
knoepfel@scisoftbuild01 build-meld $ ./test/benchmarks/bin/diamond_t
[2022-12-14 10:44:17.039] [info] Number of worker threads: 16
[2022-12-14 10:45:07.664] [info] CPU time: 726.58506s  Real time: 50.62818s  CPU efficiency: 1435.14%
[2022-12-14 10:45:07.664] [info] Max. RSS: 7.032 MB
```

1M events



- Look at I/O
- Explore paths and backwards compatibility
- Thoughts?