Plug-and-play algorithms for data selection *à la* `ptmp`

Philip Rodrigues

University of Oxford

August 14, 2020

Outline

- Reminder: `ptmp` data structures
- Reminder: `ptmp` modules (aka `TPAgents`)
- Plug-and-play algorithms with`TPFilter`
- Changes needed for DUNE DAQ

A general comment

- `ptmp` was designed specifically for ProtoDUNE-I, so we wouldn't plan to just "use `ptmp` in DUNE DAQ". But it's useful to see the solutions it provides for DUNE DAQ use cases

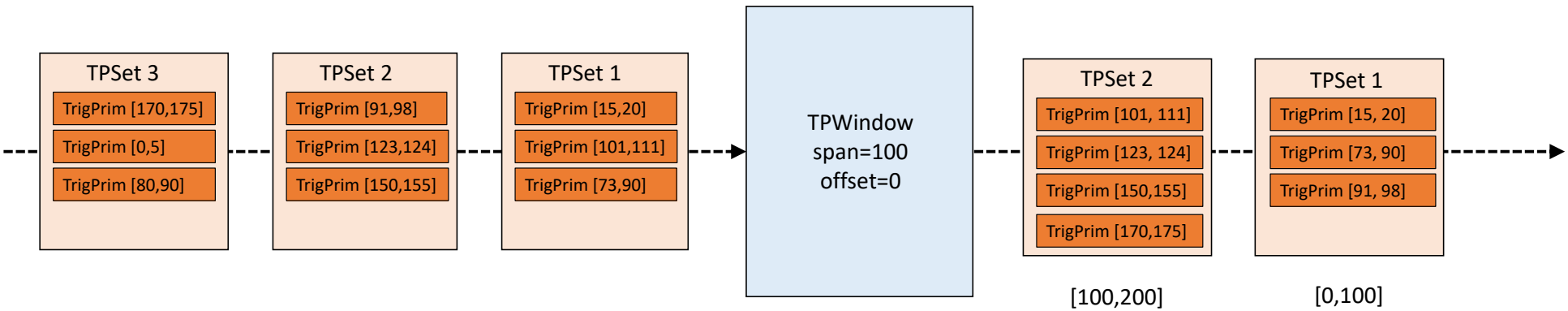
Reminder: ptmp data structures

```
message TrigPrim {
    // The channel of this TP
    required uint32 channel = 1;
    // Start time, clock ticks
    required uint64 tstart = 2;
    // The duration of the primitive
    // measured in hardware data clock ticks.
    optional uint32 tspan = 3;
    // The total ADC of the TP
    optional uint32 adcsum = 4;
    // The peak ADC of the TP above baseline
    optional uint32 adcpeak = 5;
}
```

```
message TPSet {
    // Sequence number
    required uint32 count = 1;
    // Identify the detector portion that this TPSet derives.
    required uint32 detid = 2;
    // Wall-clock time TPSet created
    required int64 created = 3;
    // The smallest tstart of the TPs, ie the tstart
    // of the earliest TP, in HW clock ticks
    required uint64 tstart = 4;
    // The time span of the TPs in the set, measured
    // in units of hardware clock "ticks".
    optional uint32 tspan = 5;
    // The channel providing the lower bound on the set, inclusive.
    optional uint32 chanbeg = 6;
    // The channel providing the upper bound on the set, inclusive.
    optional uint32 chanend = 7;
    // sum of ADC of all TPs in the set.
    optional uint32 totaladc = 8;
    // The TPs
    repeated TrigPrim tps = 9;
}
```

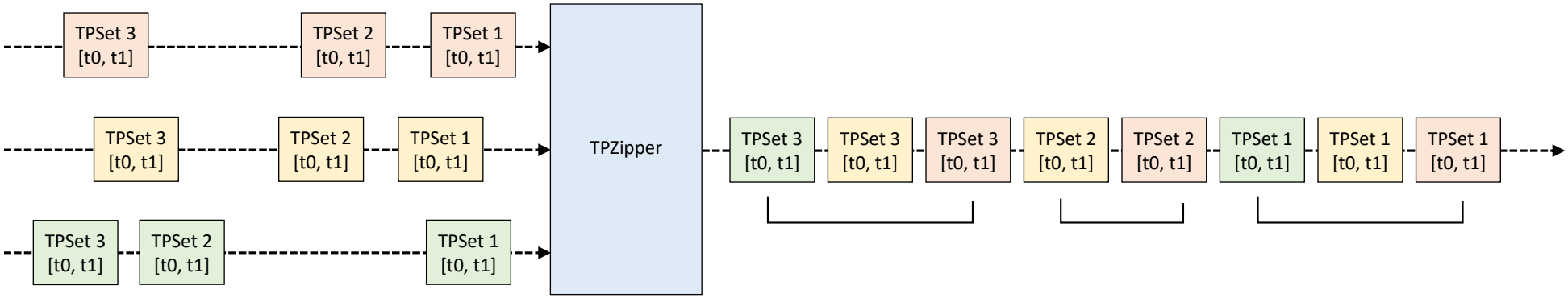
- Only TPSets are sent between modules
- TPSets stand in for trigger candidates and trigger decisions. DUNE DAQ will need separate classes

ptmp module: TPWindow



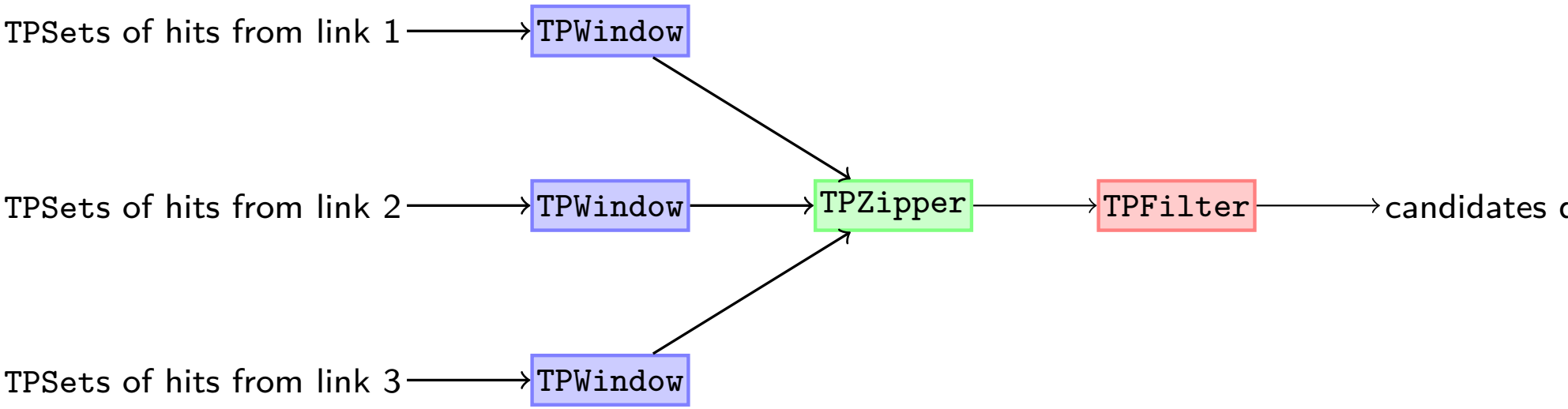
- Repackage `TrigPrims` into new `TPSets` with start (data) times in windows starting on a fixed boundary
- Input hits not required to be sorted by start time. Hits are buffered for a configurable time before being sent out. Late hits are dropped
- Full details: https://github.com/brettviren/ptmp/blob/master/docs/tpwindow.org

ptmp module: TPZipper



- Aggregate multiple `TPSet` message streams, outputting unchanged `TPSets` in time order with bounded latency
- `TPSets` arriving after "their" time has been sent out are considered "tardy" and dropped (eg green "`TPSet 2`" in my example above)

Putting it together



- `TPFilter` is where the data selection algorithm lives
- With this setup, algorithm is guaranteed to receive `TPSets` with the same time boundaries[1], in strict time order

[1] provided you set up the `TPWindows` the same

Implementing algorithms for use in `TPFilter`

- Create filter function by inheriting from `ptmp::filter::engine_t` and overriding the function

```
virtual void operator()(const ptmp::data::TPSet& input_tpset,
                        std::vector<ptmp::data::TPSet>& output_tpsets)
```

- `TPFilter` calls this function for every input `TPSet`
- If the algorithm wants to send out a trigger candidate/decision, it appends an appropriate `TPSet` to `output_tpsets`
- The choice of algorithm is made at runtime based on a configuration object (`ptmp` uses JSON configs). Configuration object also contains settings for the algorithm itself
- `TPFilter` handles the message sending/receiving

Example: simple coincidence engine

- Output a trigger decision if at least `nway` input links have `TPSets` in the same time window

```
void Coincidence_engine::operator()(const ptmp::data::TPSet& in_set,
                                    std::vector<ptmp::data::TPSet>& output_tpsets)
{
    ++m_n_sets_total;

    if(in_set.tstart() > m_last_tstart){
        // This is the first item from a new time window
        m_last_tstart=in_set.tstart();
        m_n_sources=0;
    } // end if(new time window)

    ++m_n_sources;

    if(m_n_sources==m_nway){
        // Trigger! Create the output TPSet
        ptmp::data::TPSet trigger_tpset;
        trigger_tpset.set_tstart(in_set.tstart());

        // more TPSet setting omitted

        output_tpsets.push_back(trigger_tpset);
    }
}
```

Modified from https://github.com/philiprodrigues/ptmp-tcs/blob/master/src/Coincidence_engine.cc

Changes needed/possible for DUNE DAQ

- TC and TD objects. Versions of `ptmp::filter::engine_t` for TP->TC, TC->TD. Say:

```
virtual void TCEngine::operator()(const TPs& tps_in, std::vector<TC>& tcs_out) = 0;
virtual void TDEngine::operator()(const TCs& tcs_in, std::vector<TD>& tds_out) = 0;
```

  - Pierre is working on this, including interface with `appfwk`

A possible issue

- `TPFilter` calls algorithm's `operator()` once for each `TPSet` that arrives. `ptmp` drops empty `TPSets` (to reduce traffic from TPs)
- Implies that algorithm code doesn't know it's received all the available `TPSets` for a given time window until the first `TPSet` from a later time window
- If the input `TPSets` are low-rate (more likely for TCs), this could induce a long latency
- Possible responses/solutions:
  1. It's not a problem: the algorithm was given all the available `TPSets` as soon as they were available. If it couldn't decide to trigger with what it had, it wasn't going to trigger anyway
  2. Could instead buffer `TPSets` in `TPFilter` and call `operator()` with a list of all the available `TPSets` for a given window
  3. Could call `operator()` with individual `TPSets` and also have a "current time window done" function to override
  4. Could send empty `TPSets`. Would need to think about network bandwidth/CPU cost in upstream parts of system
- Not clear to me what is right