TRIUMF

DEAP V1720 Frontends and Event Builder

Thomas Lindner | TRIUMF

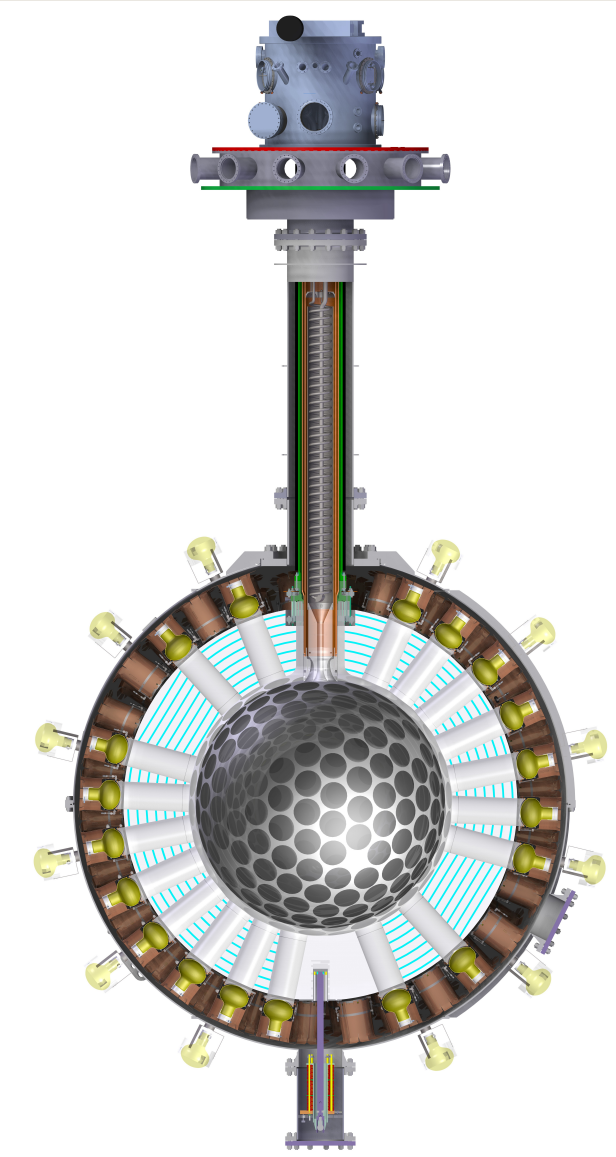
MIDAS Seminar – July 2015

Outline

- DEAP Detector
- DTM
- V1720 Digitizers
- Event Builder

DEAP Introduction

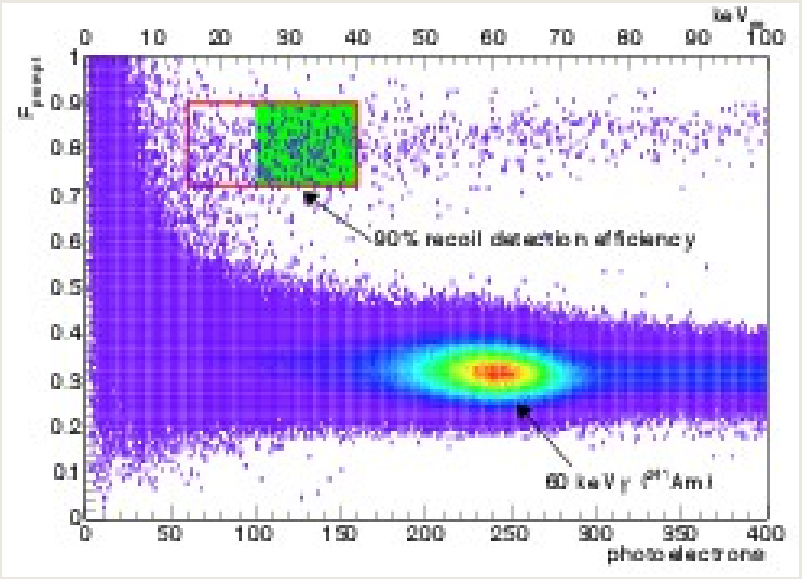
- DEAP-3600 is liquid argon dark matter experiment in SNOLAB, Sudbury, Ontario.
- Liquid argon vessel is imaged by array of 255 PMTs.
- From DAQ point of view, main challenges is ~3.6kHz rate of Ar-39 decays which produce signal similar to WIMP.
  - Can suppress Ar39 background offline using pulse shape discrimination.
  - But how to handle high rate online?



DEAP Trigger Module (DTM)

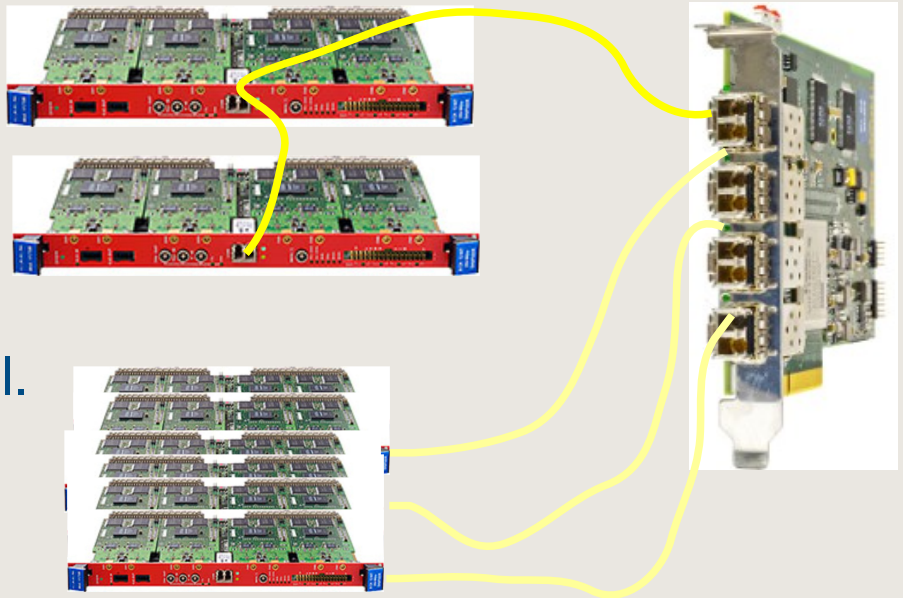
- Summed signals from PMTs fed into DTM digitizer.
- Digitized signal are processed to determine whether events are WIMP-like or background-like.
- DTM has flexible system for deciding, event-to-event, which digitizers to trigger.
- Also, DTM gets busy signals from digitizers; can be used to 'throttle' rate of digitizer triggers.

Pulse shape discrimination in liquid argon
electrons vs neutrons



V1720 Digitizers

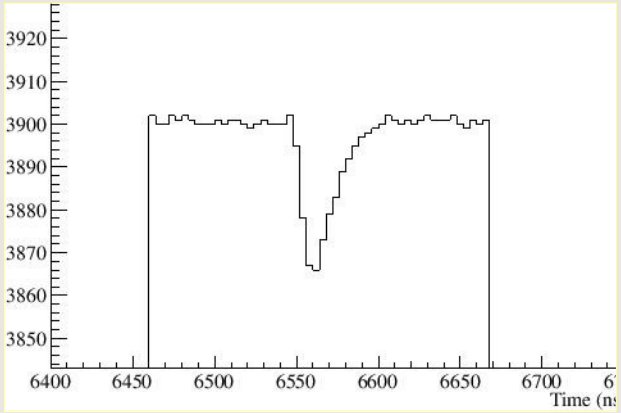
- Use CAEN V1720 250MHz FADC digitizers: one channel per PMT.
- V1720 FADC signals are processed on-board using Zero Length Encoding algorithm; only save sample below certain threshold (and neighbouring samples).
- Connections to V1720 made using fibre link with CAEN A3818 PCI card. 4 fibres per A3818, 2 V1720 per fibre.
- Each A3818 card can handle up to ~80MB/s.
- V1720 buffers handle 1028 events; when half full V1720 sets 'busy' signal.



PMT signal are actually shaped and split by custom boards, as well as being readout by slower V1740 FADCs. In this talk mostly focus on V1720 readout.

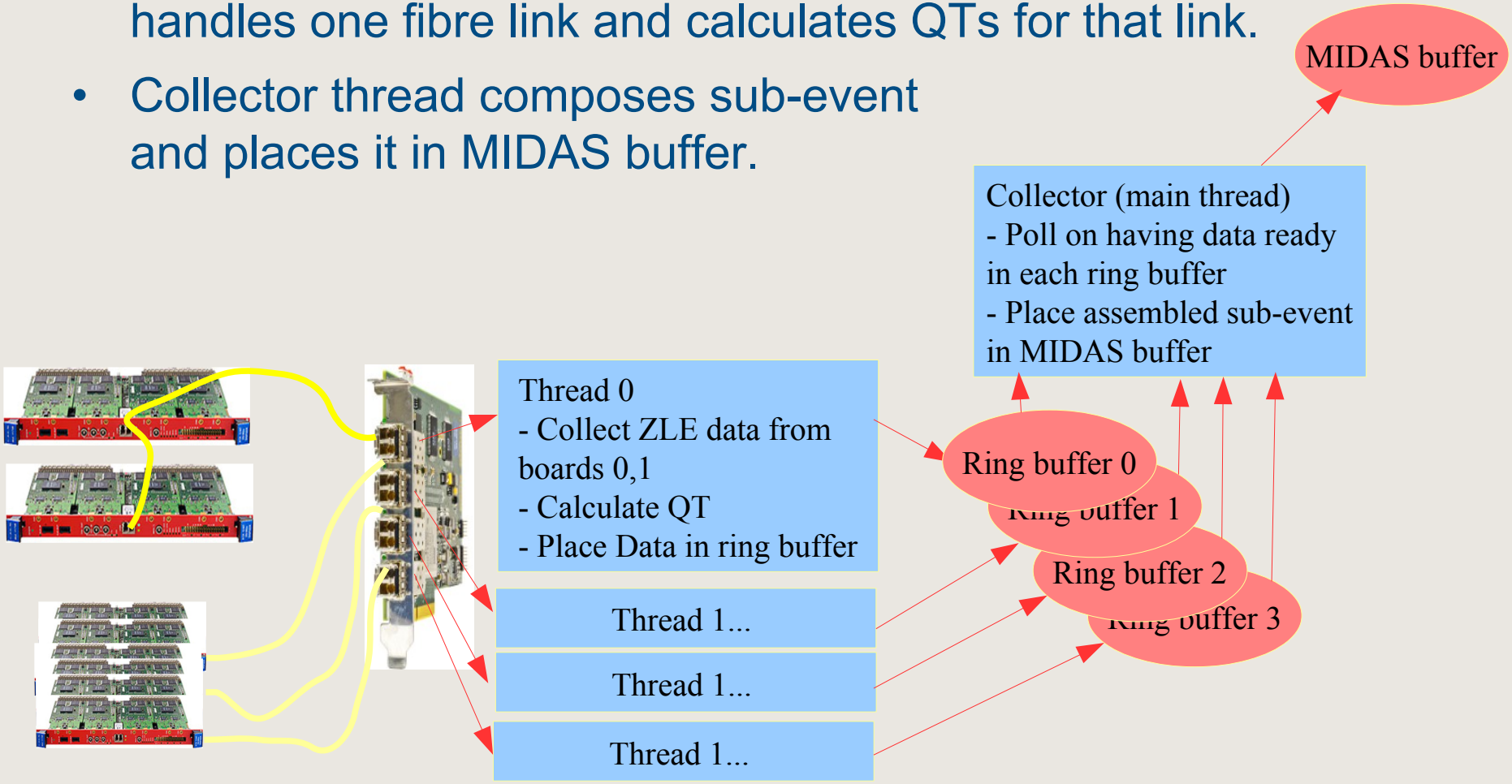
V1720 – Further data suppression

- Saving full ZLE for all triggered events would saturate DAQ.
- So in V1720 front-end code we apply another level of data suppression.
- For each V1720 ZLE pulse we calculate QT (charge/time).
- For certain events (background-like, as determined by DAQ) we only save QT and discard ZLE data.



V1720 Frontend Program

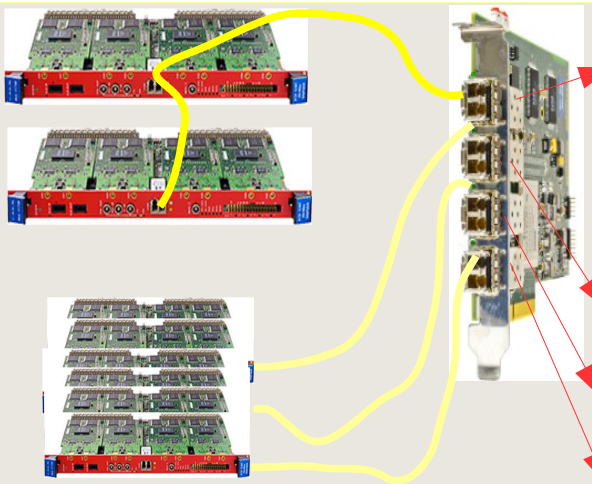
- Wrote multi-threaded polling V1720 frontend; each thread handles one fibre link and calculates QTs for that link.
- Collector thread composes sub-event and places it in MIDAS buffer.



V1720 Frontend Program

tend; each thread
's for that link.

Benefits of Multi-threading:
- Better throughput without lots of front-ends.
- QT calculation can be more complicated, since there is 4X more CPU available for calculation.



Thread 0
- Collect ZLE data from boards 0,1
- Calculate QT
- Place Data in ring buffer

Thread 1...

Thread 1...

Thread 1...

Ring buffer 0

Ring buffer 1

Ring buffer 2

Ring buffer 3

Collector (main thread)
- Poll on having data ready in each ring buffer
- Place assembled sub-event in MIDAS buffer

MIDAS buffer

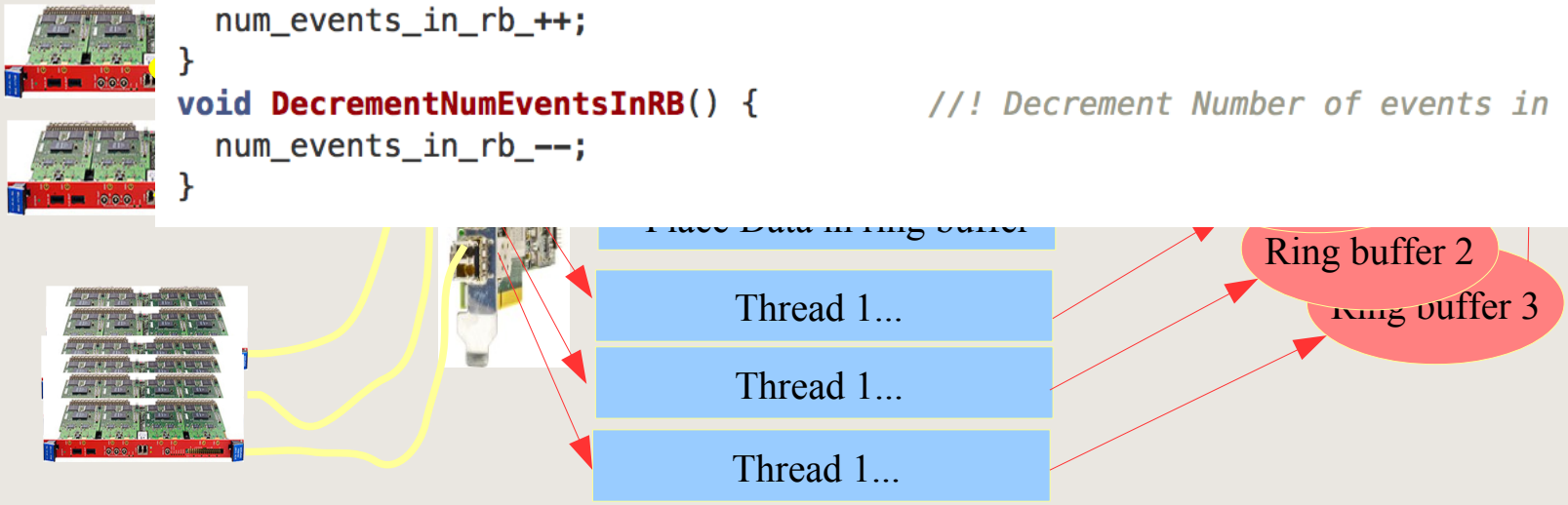
V1720 Frontend Program

- 1720 frontend; each thread
  ulates QTs for that link.
- -event

Technical point:
Thread-safe communication is handed by making counter atomic (C++11 feature)

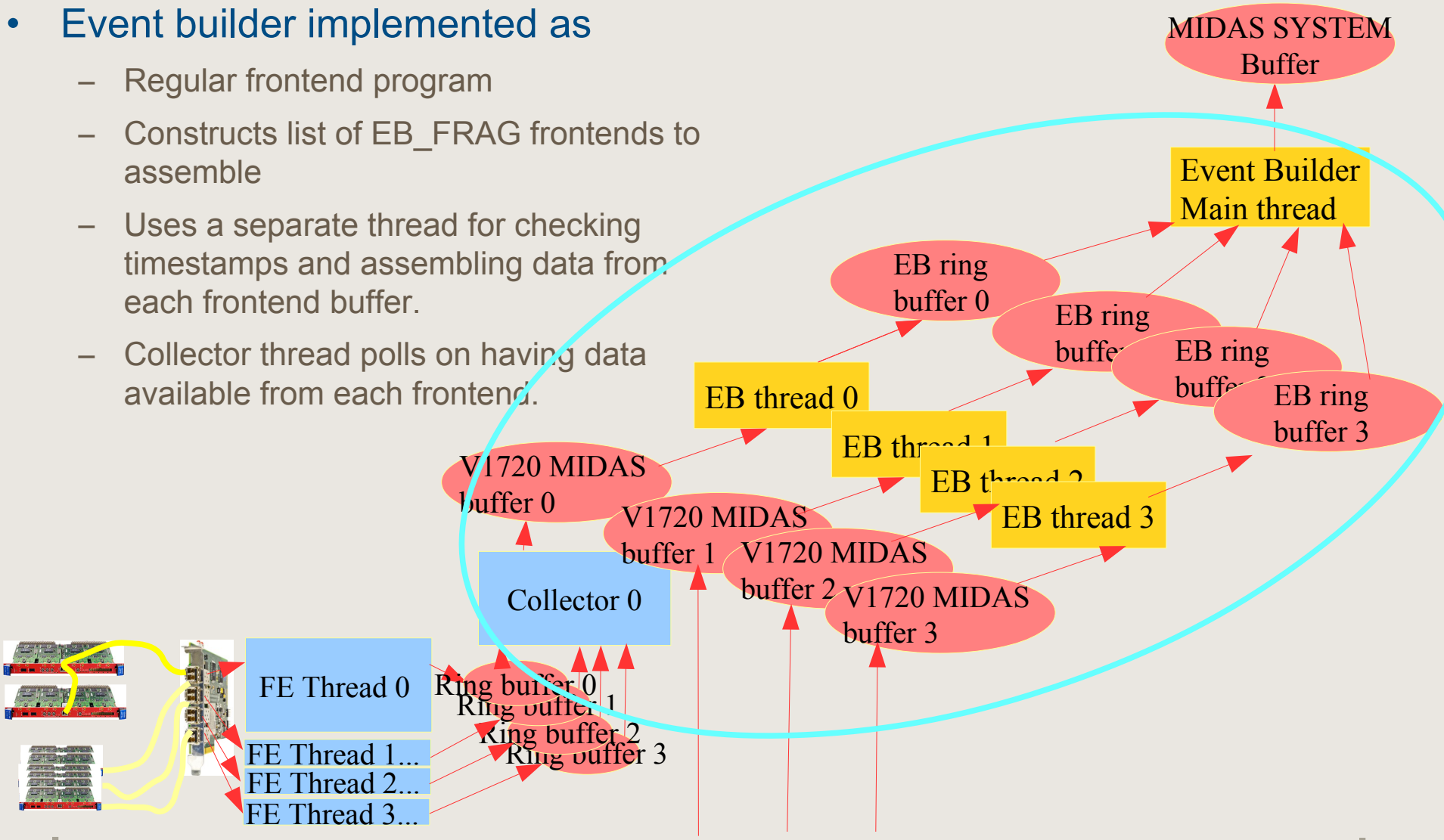
```
std::atomic<int> num_events_in_rb_;  //!< Number of events stored in ring buffer

/* These are atomic with sequential memory ordering. See below */
void IncrementNumEventsInRB() {        //! Increment Number of events in ring buffer
  num_events_in_rb_++;
}
void DecrementNumEventsInRB() {        //! Decrement Number of events in ring buffer
  num_events_in_rb_--;
}
```



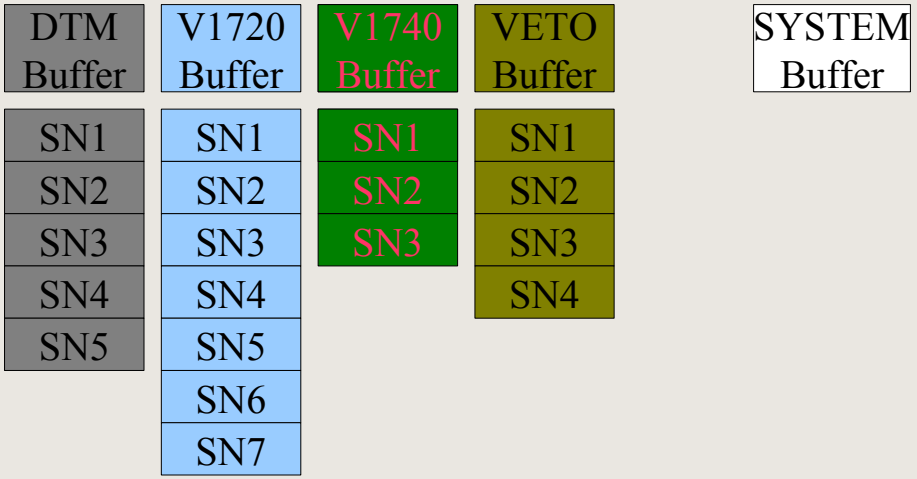
Event Builder

- Event builder implemented as
  - Regular frontend program
  - Constructs list of EB_FRAG frontends to assemble
  - Uses a separate thread for checking timestamps and assembling data from each frontend buffer.
  - Collector thread polls on having data available from each frontend.



DEAP Event Builder Processing Flow

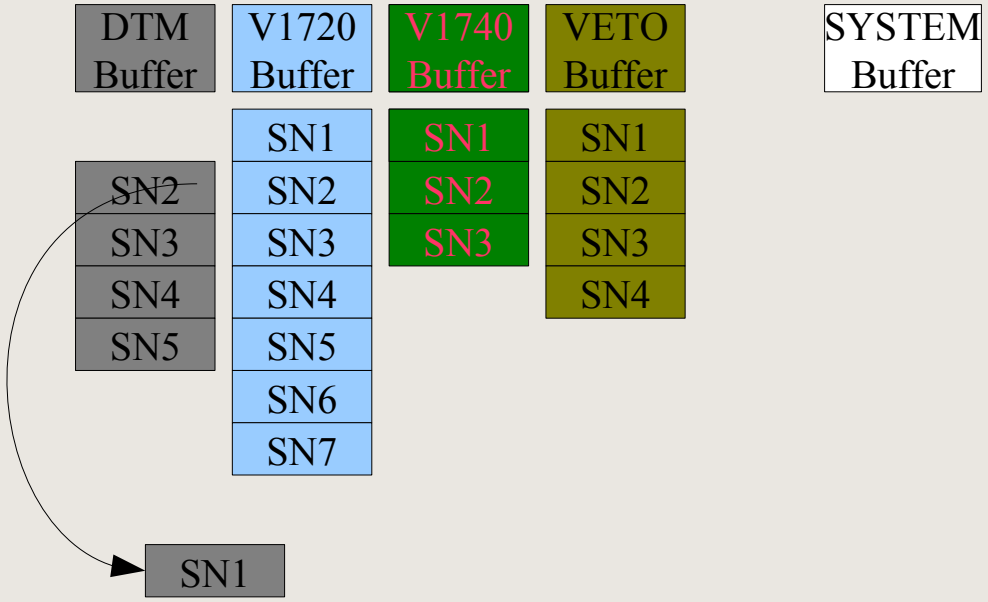
Example :
DTM running with all V1720 events, prescaling V1740/VETO by factor of 3.



Notes:
•SN = Serial number. Each event fragment is assigned a sequential serial number by front-end.
•Show a single V1720 buffer above, for simplicity. But in reality there are four V1720 buffers, one per front-end.
•Also, I'm omitting other front-ends like CALIB, though the same concept will apply.
•Buffers are being filled asynchronously and at different rates for each front-end. So don't expect same number of event fragments in each buffer.

DEAP Event Builder Processing Flow

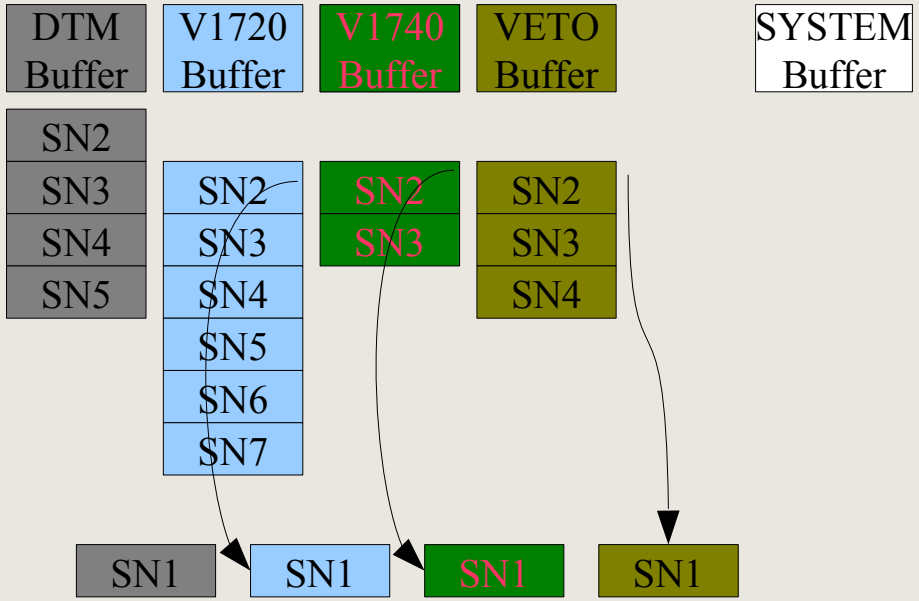
Example :
DTM running with all V1720 events,
prescaling V1740/VETO by factor of 3.



Step 1: Check DTM event. In this case, find that DTM triggered all the equipments.

DEAP Event Builder Processing Flow

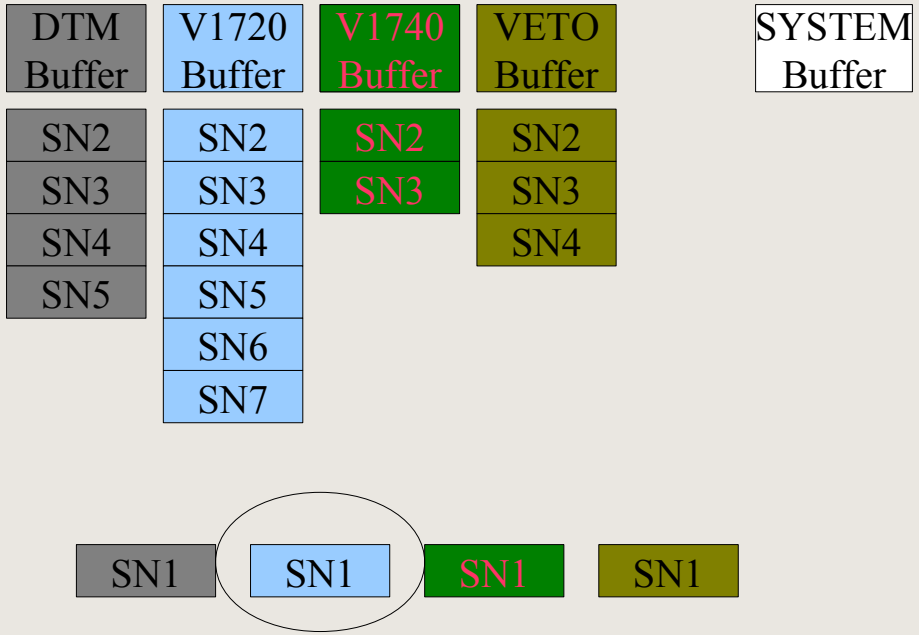
Example :
DTM running with all V1720 events,
prescaling V1740/VETO by factor of 3.



Step 2: Get each of the front-end fragments from buffers. Compare timestamps; assume they match.

DEAP Event Builder Processing Flow

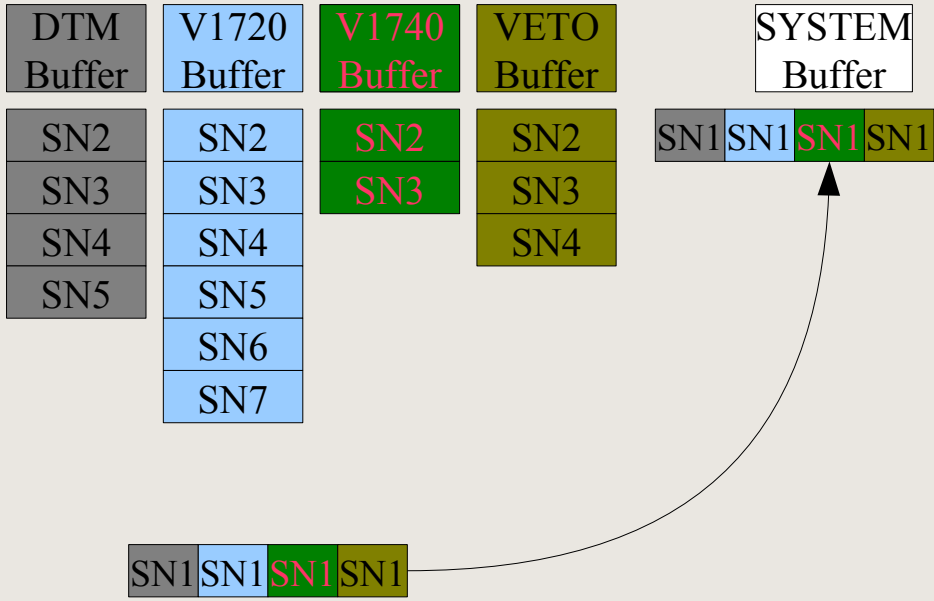
Example :
DTM running with all V1720 events, prescaling V1740/VETO by factor of 3.



Step 3: Use QT information from V1720s to create summed histograms for filtering decision.

DEAP Event Builder Processing Flow

Example :
DTM running with all V1720 events,
prescaling V1740/VETO by factor of 3.

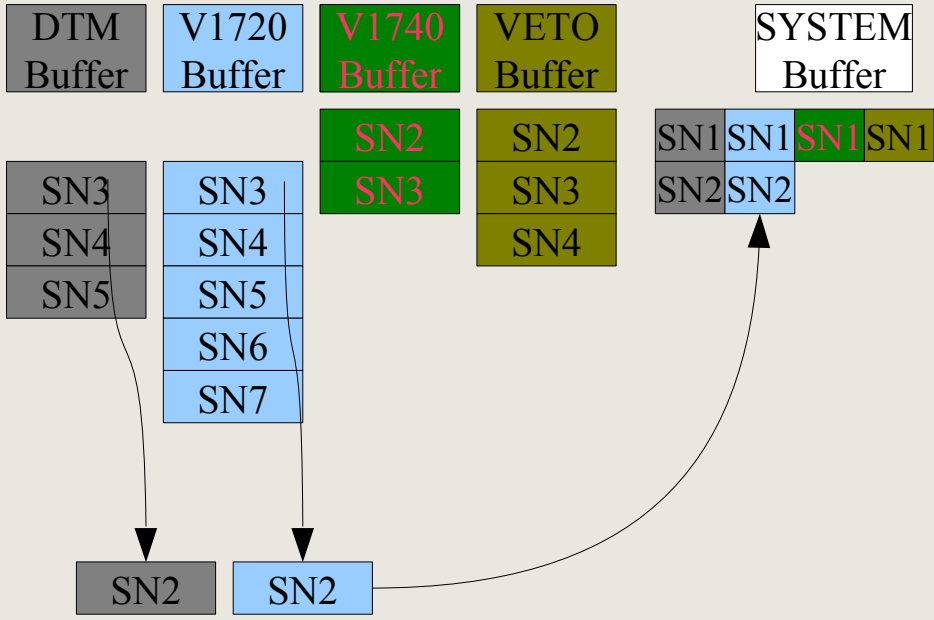


Step 4: Make filtering decision; in this case, assume we keep all information. Compose final event.

Note: we don't actual store the 'serial number' for each fragment in the final event. There is just a single serial number for the final event. I only show the individual serial numbers here so it is clear how the fragments are getting assembled

DEAP Event Builder Processing Flow

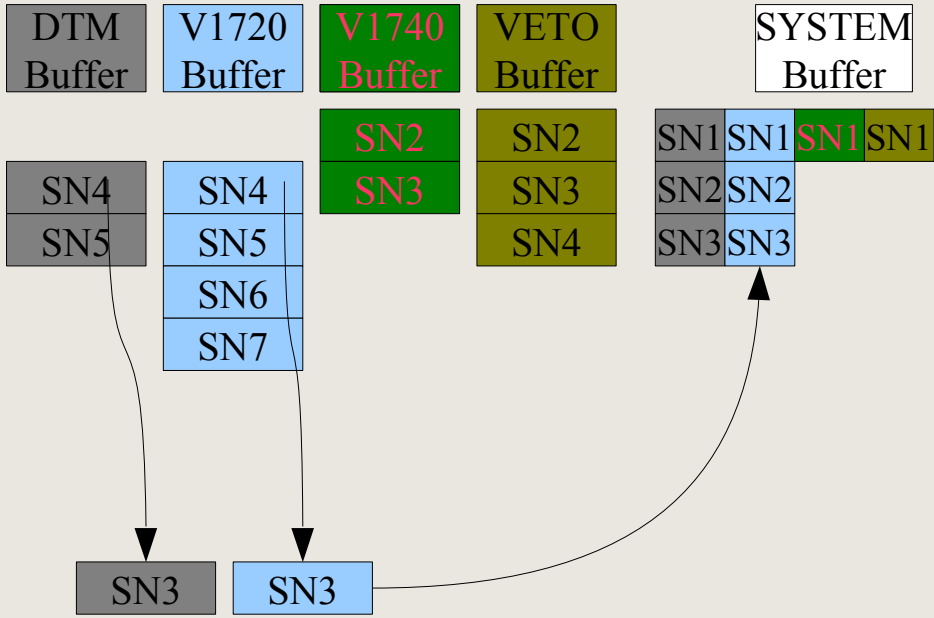
Example :
DTM running with all V1720 events, prescaling V1740/VETO by factor of 3.



Next event: same sequence as last time, except that in this case the DTM only triggered the V1720s. So we only take the fragments from the DTM and V1720 buffers.

DEAP Event Builder Processing Flow

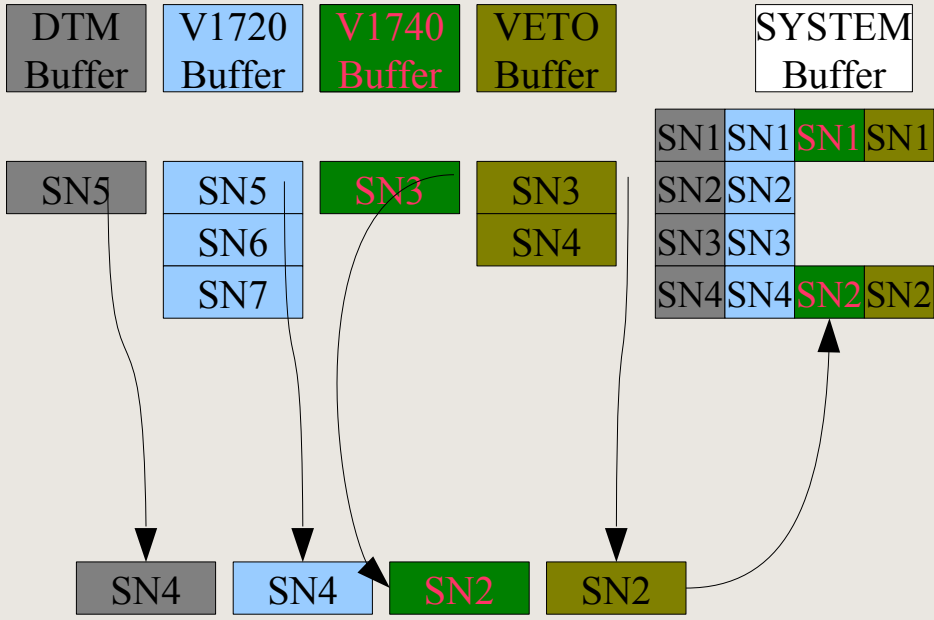
Example :
DTM running with all V1720 events,
prescaling V1740/VETO by factor of 3.



Third event: same again...

DEAP Event Builder Processing Flow

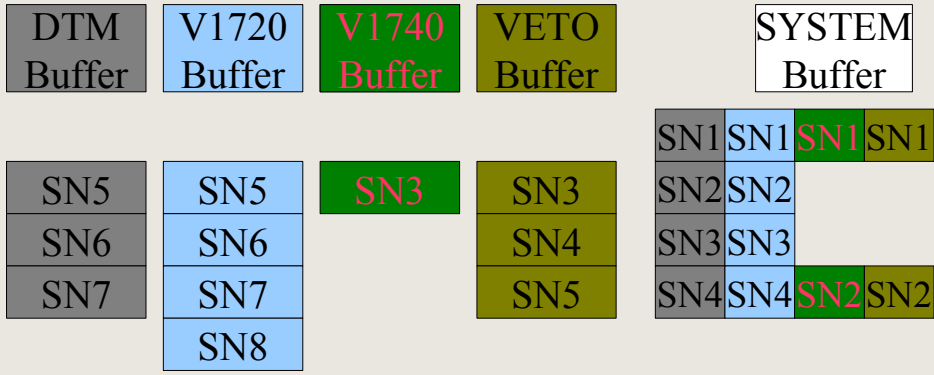
Example :
DTM running with all V1720 events, prescaling V1740/VETO by factor of 3.



Fourth DTM event: now we need all the equipment again.

DEAP Event Builder Processing Flow

Example :
DTM running with all V1720 events,
prescaling V1740/VETO by factor of 3.

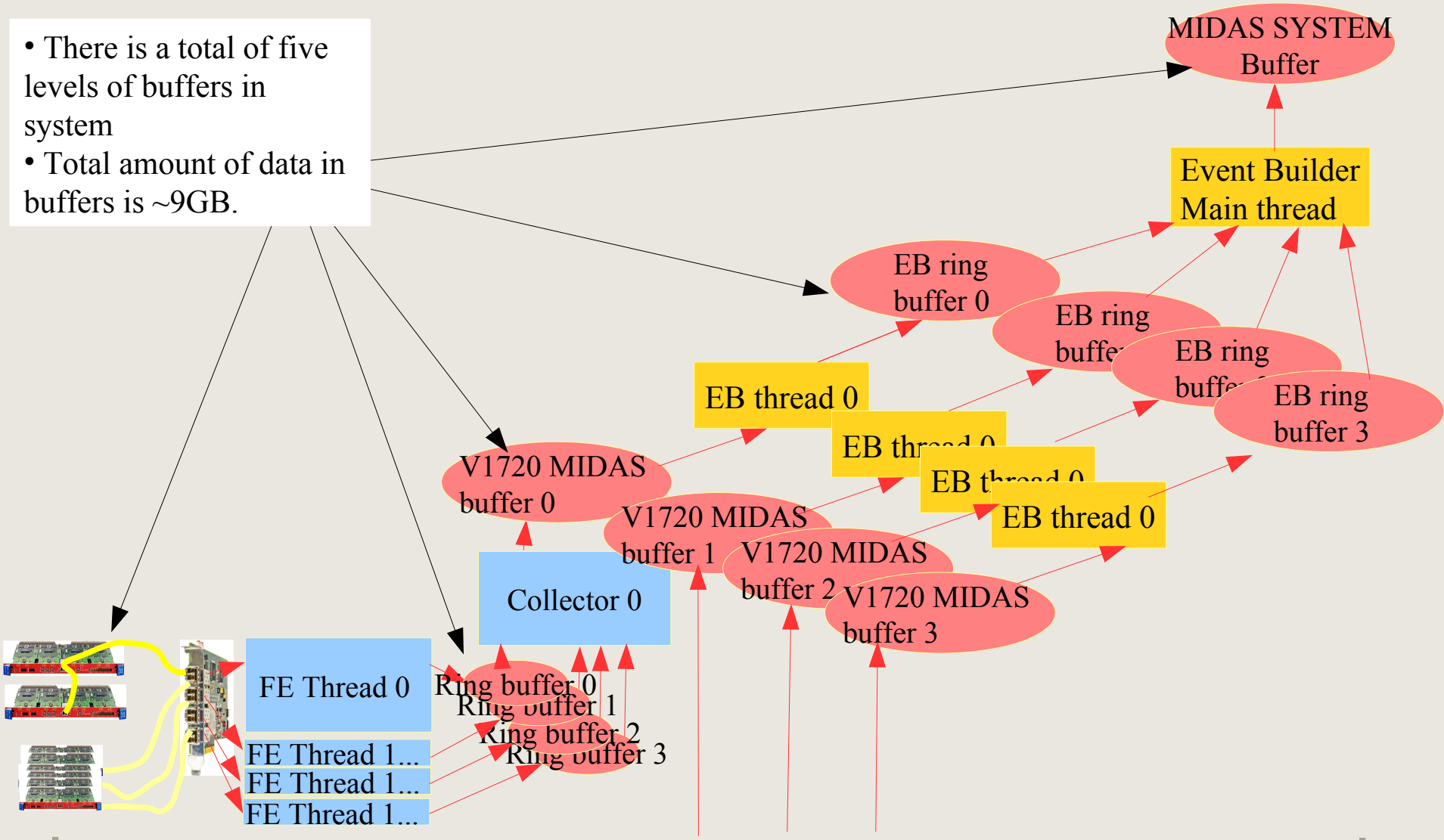


And so on...

Of course we are also adding event fragments to the front-end buffers asynchronously as the event handlers removes event fragments from those buffers...

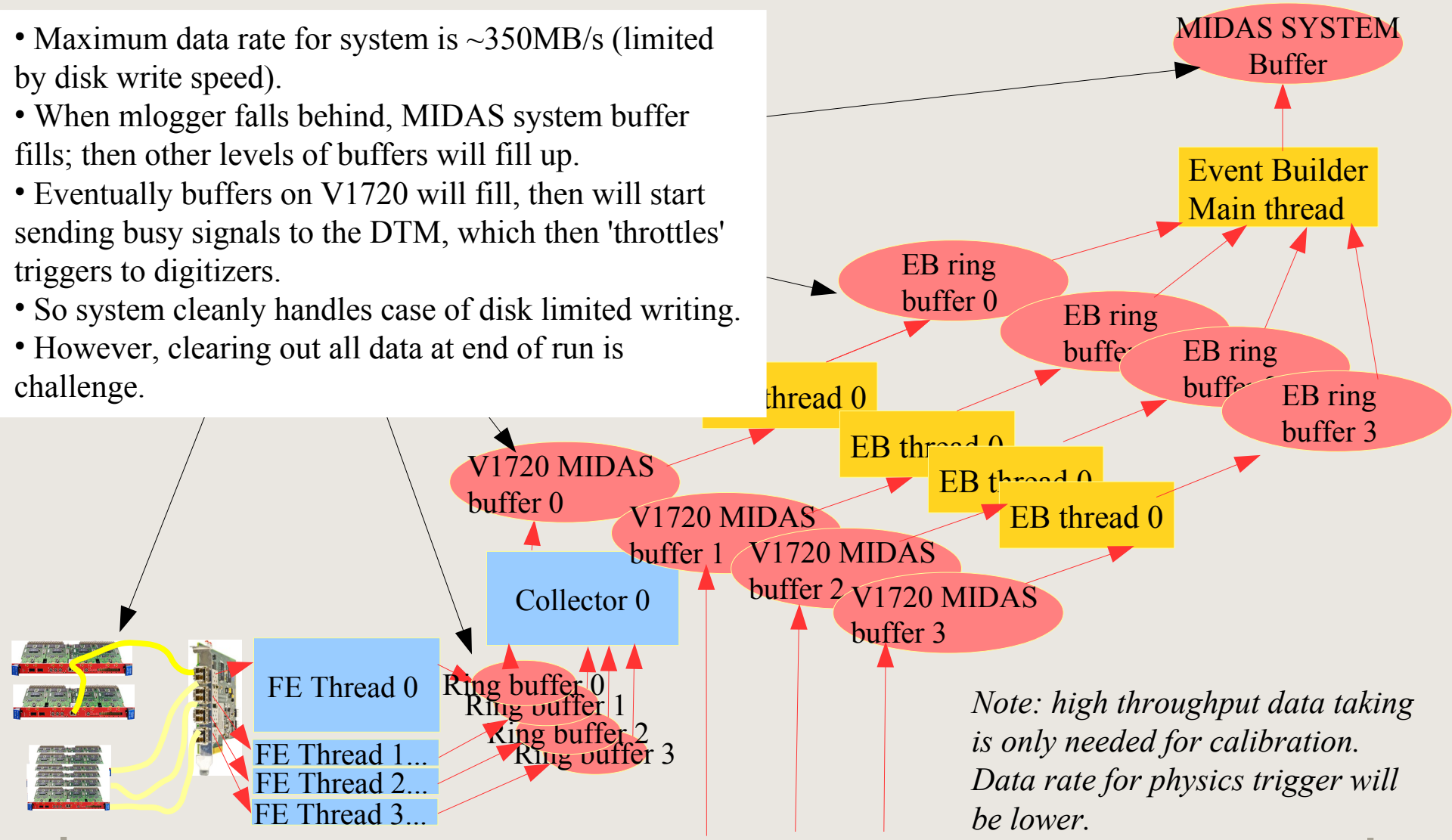
Event Builder Performance

- There is a total of five levels of buffers in system
- Total amount of data in buffers is ~9GB.



Event Builder Performance

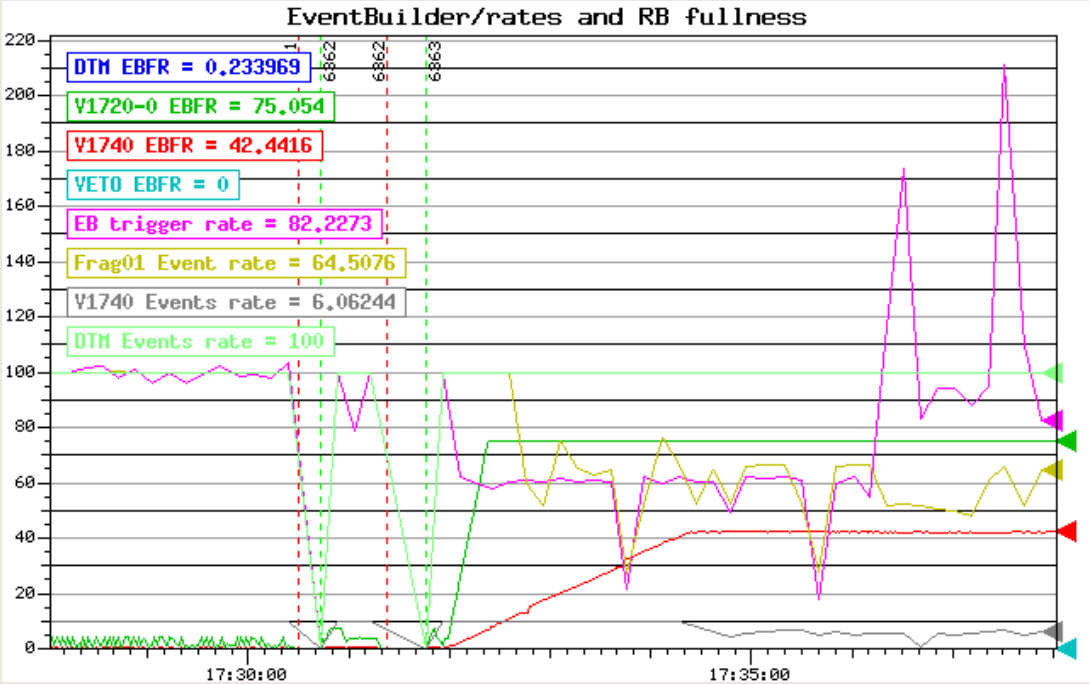
- Maximum data rate for system is ~350MB/s (limited by disk write speed).
- When mlogger falls behind, MIDAS system buffer fills; then other levels of buffers will fill up.
- Eventually buffers on V1720 will fill, then will start sending busy signals to the DTM, which then 'throttles' triggers to digitizers.
- So system cleanly handles case of disk limited writing.
- However, clearing out all data at end of run is challenge.



Note: high throughput data taking is only needed for calibration. Data rate for physics trigger will be lower.

Event Builder: Data Throughput

- Results can be difficult to interpret, as system evolves over first couple minutes of data taking.
- But stable operation possible in the maximize data rate mode.



Conclusion

- Use multi-threading for DEAP frontends and event builder in order to maximize throughput and available CPU for data processing.
- DAQ is working and shown to be (reasonably) stable with high data throughput and variable trigger conditions.
- All code available for inspection here: https://bitbucket.org/ttriumfdaq/deap/src/