Conditions strategy 1: fully-threaded

Allow each event to potentially have its own view of conditions.

- Most natural way to proceed seems to have a separate detector store for each event slot, along with the event store.
- Need to be able to share conditions objects that are actually the same.
  - Probably not a fundamental problems, since objects in the store are already reference-counted.
  - But new code needed to manage the bookkeeping.
- Currently tools/algorithms register callbacks that read conditions information, process it, and store it in the tool/algorithm instances.
  - Clearly won't work in this case. Store processed conditions back in the detector store instead?
  - Then callbacks start looking like algorithms working on the detector store. Could they be scheduled that way?
- Does conditions backend/database code need to be thread-safe?
  - Could probably serialize it if needed.
- Beware of increasing memory requirements! Remember that saving memory is why we're doing this in the first place.

Conditions strategy 2: LB barrier

Only change conditions at a LB barrier. Always finish all events from one LB before starting another.

- Most conditions code probably doesn't have to change; conditions backend remains serialized.
- Present callback code probably still also works.
- No need for event sorting to maintain LB groupings.
- CMS takes this approach.
- Some loss of effieciency at LB transitions. For normal reco, this is probably not a large effect.
- But implies that multithreaded processing may not be useful for certain kinds of derived data. Probably only a small fraction of total workload, so maybe not a big deal.
  - Use a barrier only at LB transitions where conditions actually change.
- Time-based (DCS) conditions get rounded to nearest LB boundary.
  - Should be OK for most conditions data, but are there exceptions?
  - Ex: conditions data for calo noise bursts need to be accessed at a granularity of individual events. HV ramping?

Conditions strategy 3: LB barrier with sub-LB exceptions

Adopt the barrier strategy. For the (few?) conditions objects for which a granularity smaller than a LB is needed, replace the conditions object with one that holds information for the entire LB.

- Mostly same pros and cons as previous slide.
- Need to design new type of conditions objects and migrate code.
- Maybe not that many classes need to change, though.

Way forward?

Tentatively adopt strategy 3.

- Core changes: IOVSvc needs to interact with scheduler/event loop. Event loop calls IOVSvc when it's time for a transition; IOVSvc tells event loop when the next transition will occur.
- Need prototype for conditions object holding LB-worth of data.
  - Migration issues?
- Audit conditions objects for thread-safety.
- Catalog conditions information required for reconstruction/derivation/analysis jobs?
  - Type, volume, frequency of change.
- Poll detector groups to find which conditions require sub-LB granularity.