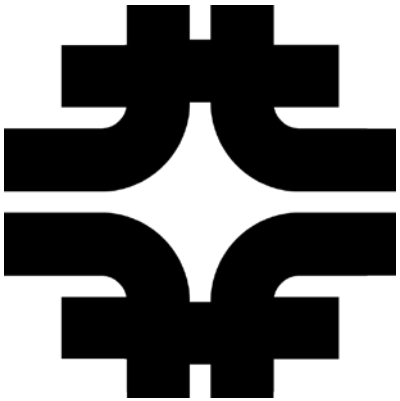
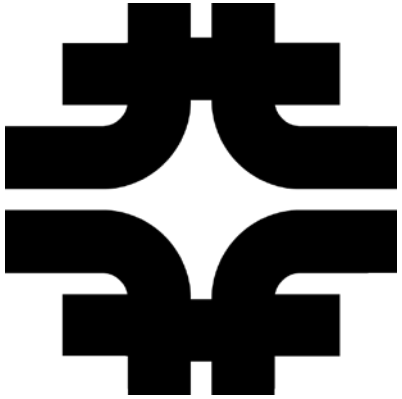
CmsToyGV: Tests of GeantV in CMS Software Framework

Kevin Pedro (FNAL)
on behalf of the CMS Collaboration
& in association with the GeantV R&D Team
March 28, 2018



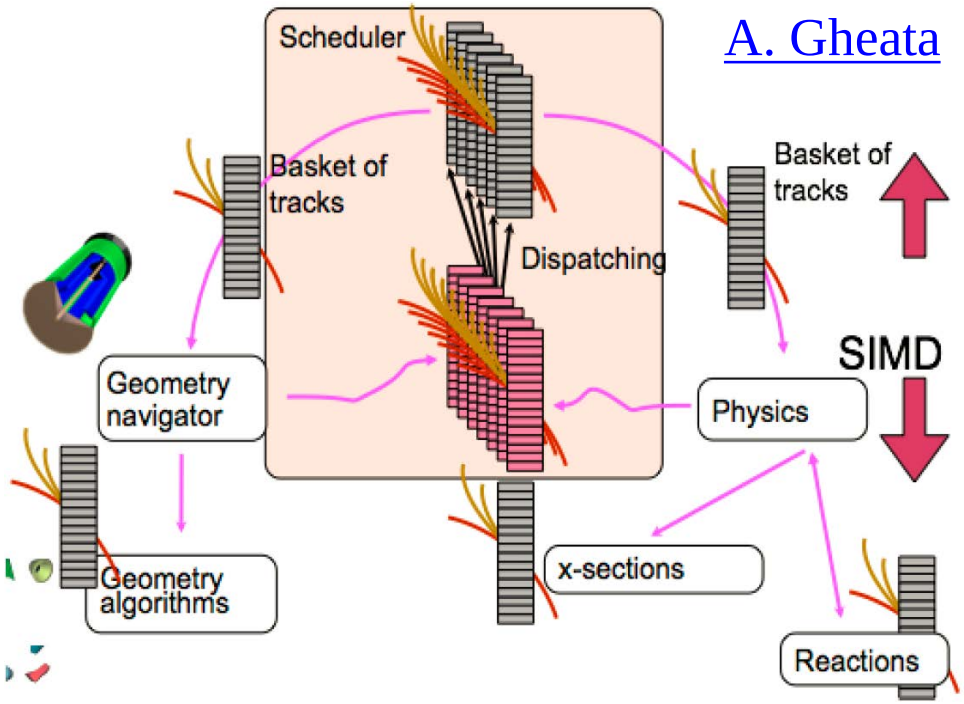
Acknowledgements

- GeantV R&D Team:
  - CERN, Fermilab, BARC (India), CIC (IPN, Mexico)
  - Financial support from Intel
- Development of CmsToyGV example:
  - GeantV: Guilherme Lima, Andrei Gheata, Philippe Canal, Soon Yung Jun
  - CMS: Chris Jones, Daniel Elvira, KJP
- A team effort – thanks to all!



GeantV Transport Engine

- Track-level parallelism
  - Exploit single instruction, multiple data (SIMD) vectorization
  - Group similar† tracks into *basket* (from multiple events)
  - Send entire basket to algorithm: process particles in parallel
- Other features:
  - Use of templates for generic code: promote SIMD for different architectures and libraries
  - Improved code and data structures: memory locality, fewer cache misses
  - Adaptive scheduler: monitor MT performance, adjust parameters at runtime



A. Gheata

† (particle type, geometry/material in step)

Vectorized Components

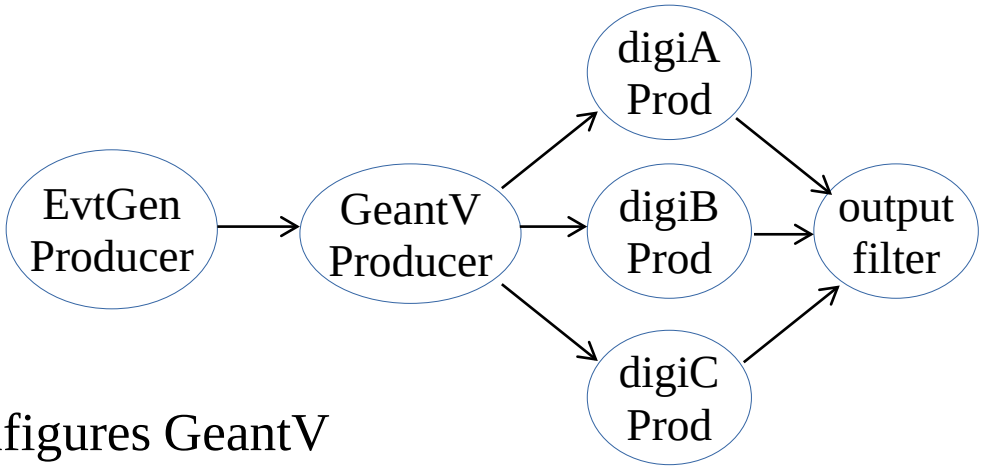
- VecCore (GitHub/root-project)
  - Libraries: Vc, UME::SIMD
  - Utilities: pRNG, math functions & constants
- VecGeom (CERN/GitLab)
  - Vectorized geometry and navigation, multi-particle interface
  - Improved code, compatible w/ Geant4
    → already in use by CMS (scalar mode)
  - First results: sizable gains in unit tests, room for further optimizations
    (still to be tested in integrated environments)
- GeantV (CERN/GitLab)
  - Alpha release now available!
  - Generic magnetic field propagation included
  - Physics: currently only EM processes in scalar mode
    → vectorized physics pursued for beta release (2019)

Examples for Experiments

- Standalone examples:
  - FullCMS: Geant4 and GeantV versions
  - Good place to learn interface differences, etc.
  - FullLHCb also provided for GeantV
- Integrated example: cmsToyGV
  - GeantV simulation controlled by external framework
  - toy-mt-framework developed by Chris Jones (FNAL) as testbed for multithreading CMS software
  - Uses Intel TBB (Thread Building Blocks): task-based event processing
  - Minimal version of toy framework ported to GeantV repository → self-contained example

CmsToyGV

- JSON configuration file:
  - # threads
  - # simultaneous events (streams)
  - modules & their parameters



- **GeantVProducer**:
  - CMS module, instantiates & configures GeantV
  - Calls *produce*() once per event (stream), each call uses one thread
  - Output sent to downstream modules
- **Geant::RunManager**:
  - Configures and controls GeantV components
  - Gets events from framework, processes, returns when finished
  - *RunSimulationTask*() called by *produce*() in **GeantVProducer**
  - Uses multiple threads to process events cooperatively (tracks mixed together in baskets)

Details

- CMS 2018 geometry, uniform 4T magnetic field
  (to be updated to realistic field)
- Real physics only available for limited EM processes, no HAD
  (if EM and HAD both used, simple cross section tables applied for both)
- Convert generated events (particle gun or HepMC) to Geant::**EventSet** format
- GeantV basketizes and processes tracks as it receives events
- Secondary tracks (produced w/in GeantV) also basketized and queued
- Event is reassembled once all tracks processed → *RunSimulationTask*() then returns control of thread to the framework
- Downstream modules are placeholders that use specified CPU time

➢ Development of this example led to modifications of **RunManager** and related GeantV classes to allow external control of event loop and task assignment → crucial feedback between experiments and simulation library

Next Steps

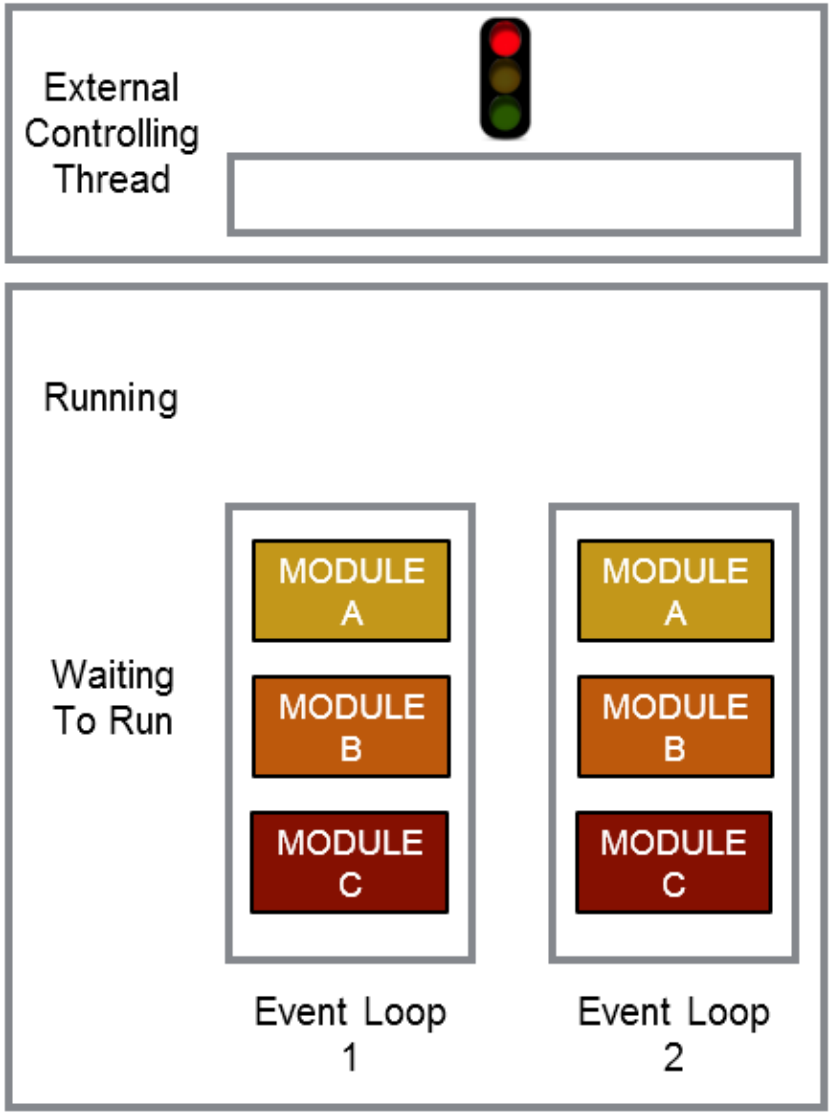
- Integration in full CMSSW framework

1. GeantV alpha release can be installed as CMSSW external package ✓
   (dependencies understood)
2. Port **GeantVProducer** to full framework → first version now running! ✓
   (still need further integration with CMSSW & simulation sequence)
3. Adapt to use new **ExternalWork** feature in full framework:
   - More efficient path: *acquire* () → GeantV processing → *produce*()
   - *acquire*() sends event data from CMSSW to GeantV, then returns
     → asynchronous processing, threads aren't blocked
   - *produce*() only called once GeantV is done with an event
     → introduce a callback function to handle this notification
4. Conduct program of tests to understand (and improve!) computing
   performance of GeantV in experiment's framework
5. Complete integration of GeantV with CMS simulation:
   adapt SensitiveDetector classes that handle additional step actions

External Work in CMSSW

Setup:

- TBB controls running modules
- Concurrent processing of multiple events
- Separate helper thread to control external
- Can wait until enough work is buffered before running external process



- See backup for remaining steps

Program of Tests

4. Conduct program of tests to understand (and improve!) computing performance of GeantV in experiment's framework

   - ➢ Comparison baseline: Geant4 event-level parallelism

   - • Conduct tests in full CMSSW framework for most accurate comparison

   a) Different particles (electrons, photons) and multiplicities

   b) Different number of threads (scaling)

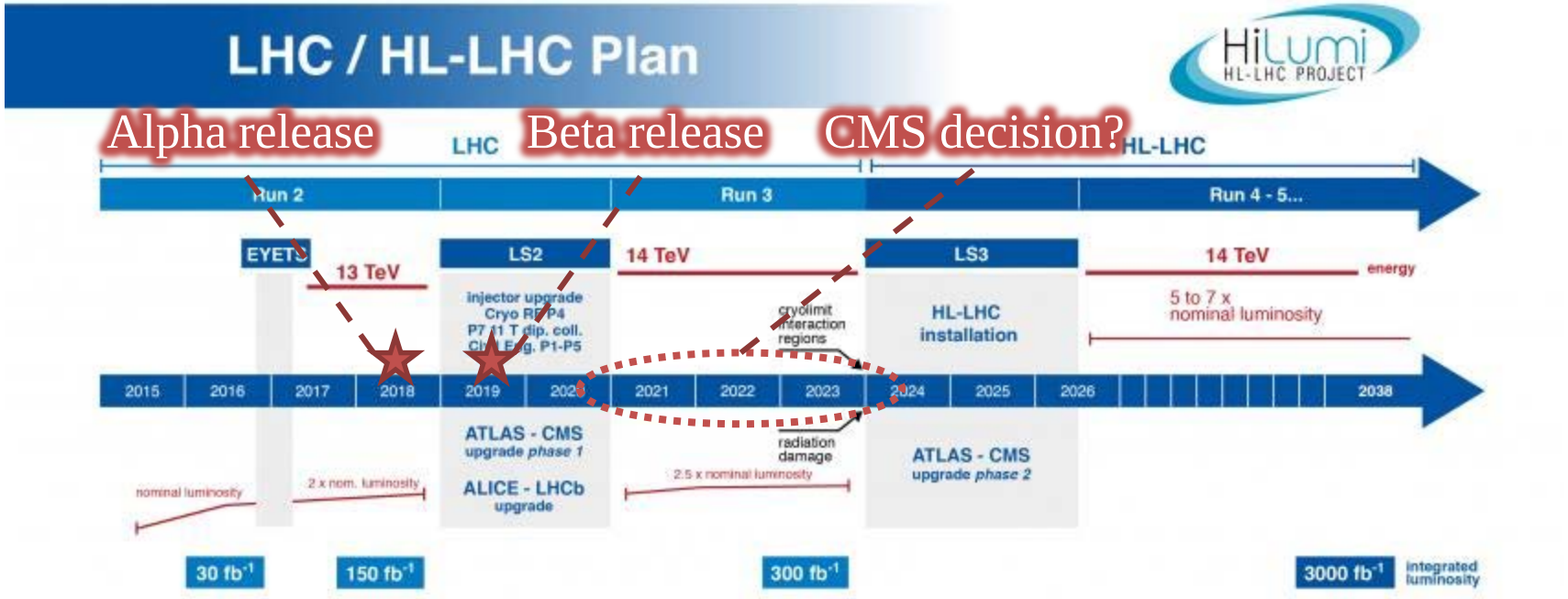
   c) Different CMS geometries (e.g. current (2018), Phase2 (more complex))

   d) Different instruction sets for VecGeom

   e) Use Intel VTune profiler to assess bottlenecks/stalls

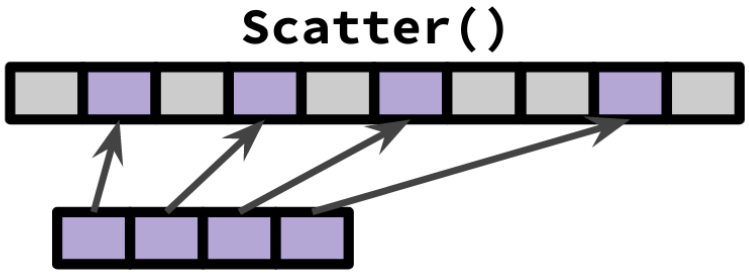
CMS Plans

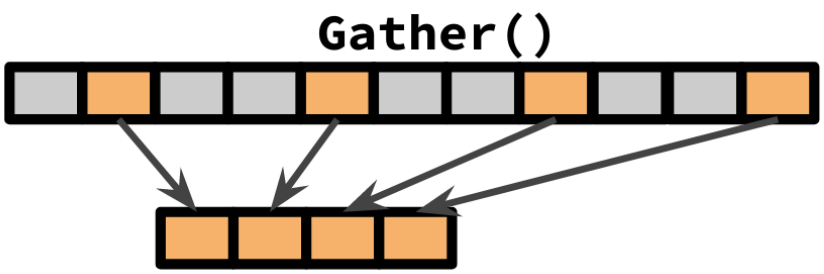
- Co-development model: test consistency of threading models and interfaces
- Provide feedback to prevent divergence between CMS and GeantV
- Post-beta release:
  - Assess GeantV performance and CMS needs
  - Decide about migrating to GeantV on timescale of HL-LHC
  - If CMS migrates, GeantV transport engine will be integrated into Geant4 simulation toolkit & supported by Geant4 collaboration



Conclusions

- GeantV alpha release is available
- Other vectorized components (VecCore, VecGeom) also available
- Development continues toward beta release in 2019
  (including vectorized EM physics)
- CmsToyGV example exists to demonstrate integration with experiment's software framework
- Next step: integration & testing in full CMSSW framework
  → in progress, stay tuned!
- Eventually: aim for 2–5× speedup in final version of GeantV



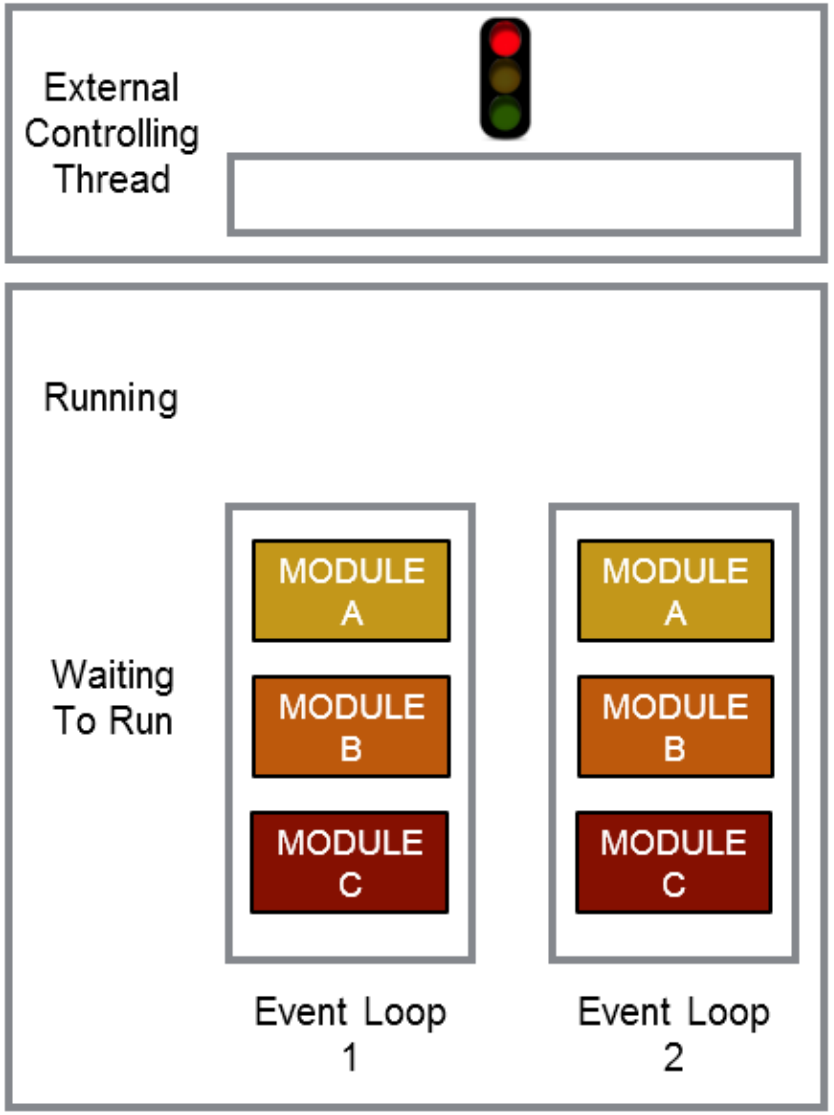


Backup

External Work in CMSSW (1)

Setup:

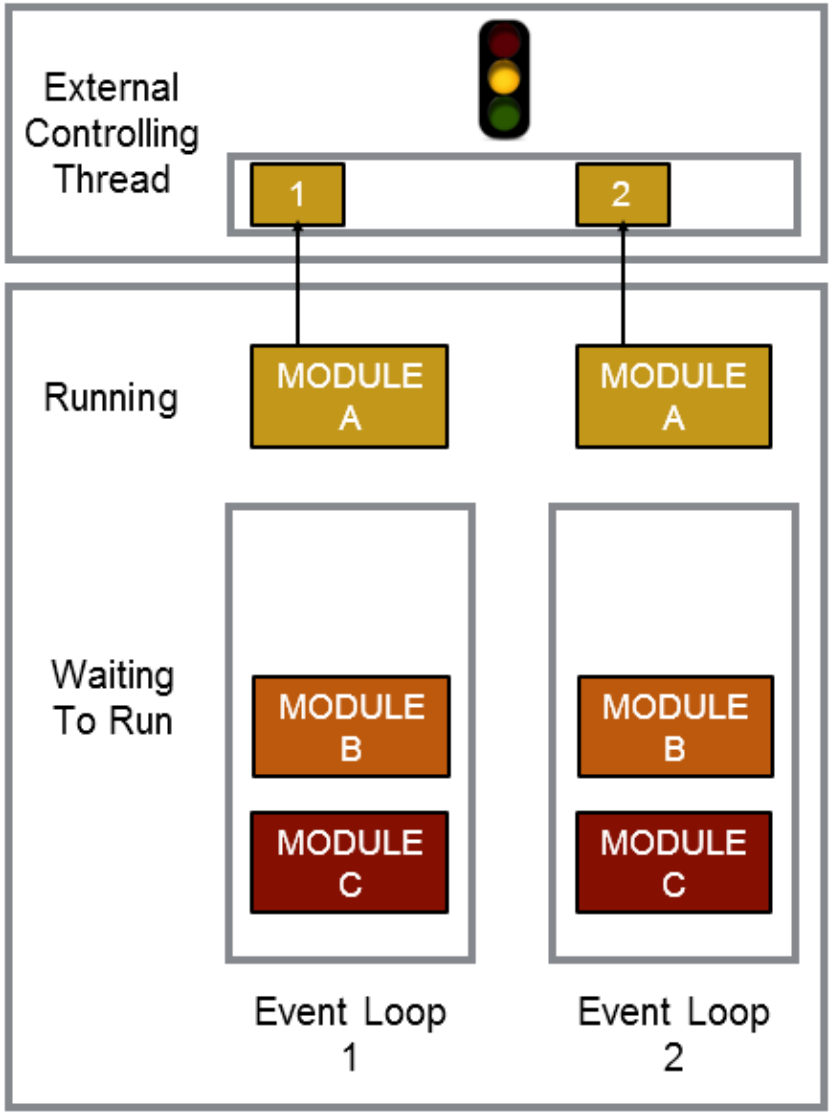
- TBB controls running modules
- Concurrent processing of multiple events
- Separate helper thread to control external
- Can wait until enough work is buffered before running external process



External Work in CMSSW (2)

Acquire:

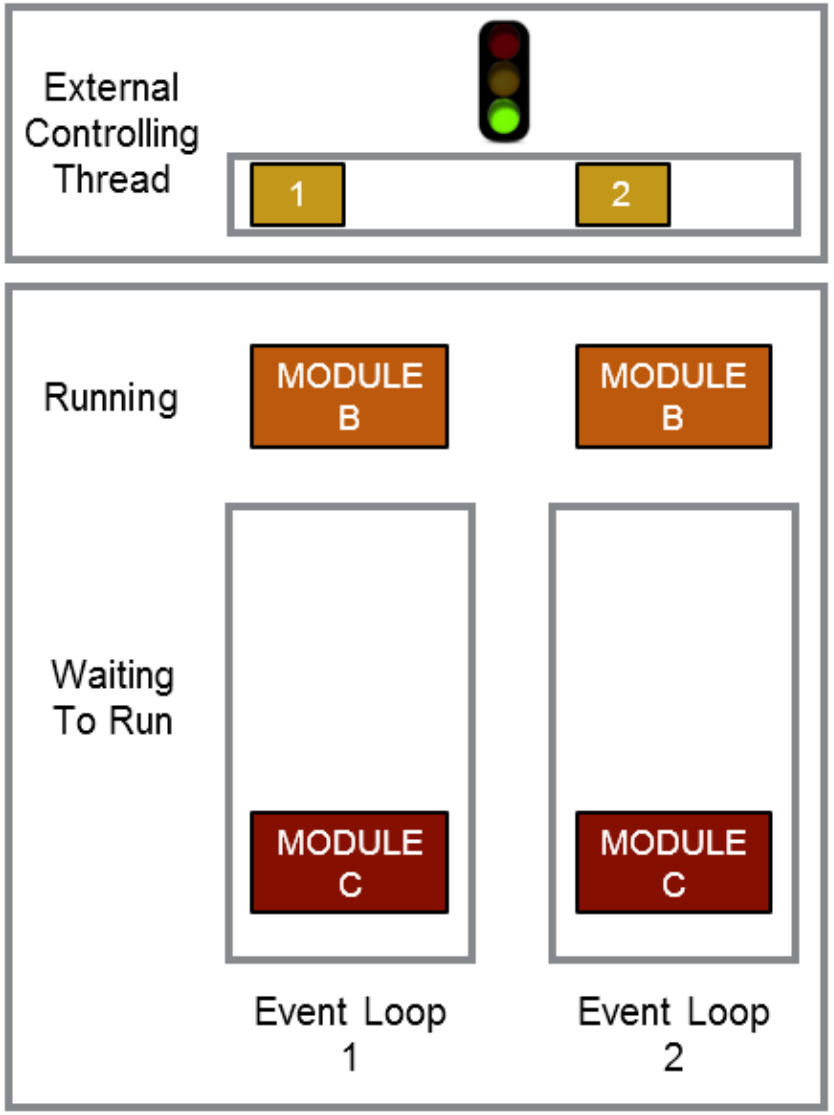
- Module *acquire*() method called
- Pulls data from event
- Copies data to buffer
- Buffer includes callback to start next phase of module running



External Work in CMSSW (3)

Work starts:

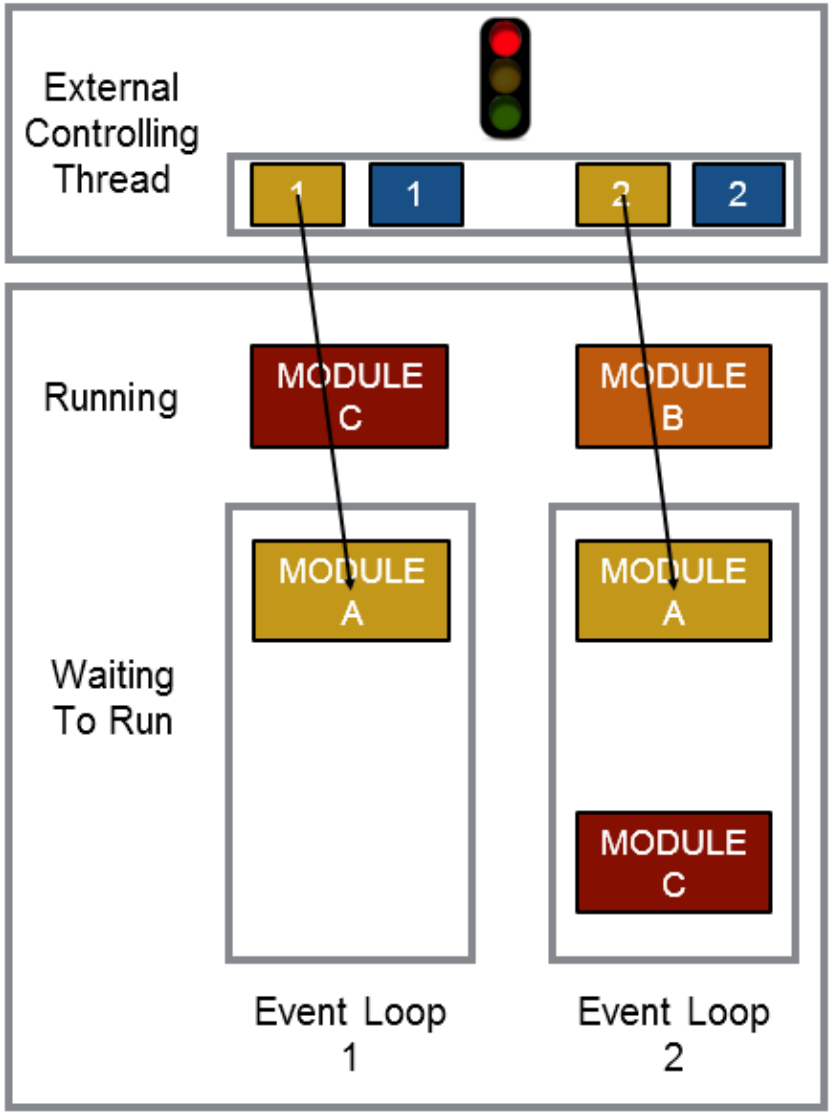
- External process runs
- Data pulled from buffer
- Next waiting modules can run (concurrently)



External Work in CMSSW (4)

Work finishes:

- Results copied to buffer
- Callback puts module back into queue



External Work in CMSSW (5)

Produce:

- Module *produce*() method is called
- Pulls results from buffer
- Data used to create objects to put into event