CMS multi-thread support for generators

Congqiao Li

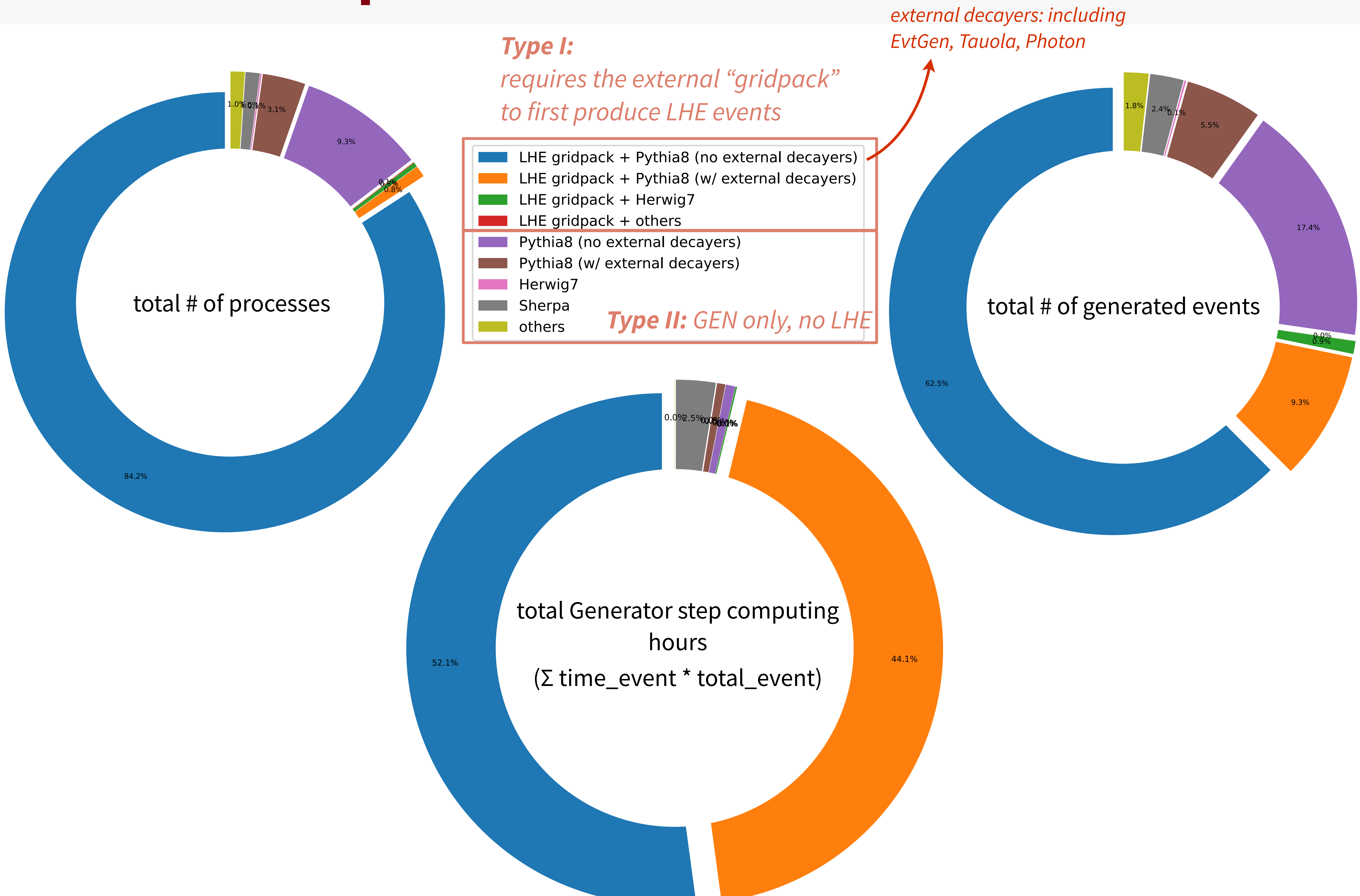
HSF Frameworks WG Meeting

1 December, 2021

Introduction

- ***Event generators***
  - Event generation is at the earliest step (the "GEN step" in CMS) in the MC event processing chain
  - In GEN step, CMSSW interfaces with external generator C++ libraries (Pythia, Herwig, Sherpa…)
  - different physics processes may use different generators
- ***Multithreading × Event generators?***
  - concurrent computing in generators is demanded in CMSSW
  - difficulty might be: concurrent GEN methods may vary depending on the specific generator type (the specific third-party library used)
  - various CMSSW modules for GEN designed by the framework team to enable multithreading
- Aim of the talk:
  - a technical summary of concurrent GEN modules in CMSSW
    - for each module, talk about the mechanism, performance improvement, and future plan
  - an overall picture of multi-thread GEN application in CMS
    - generator configs that do/don't support multithreading
    - validation results

Generator step in CMS

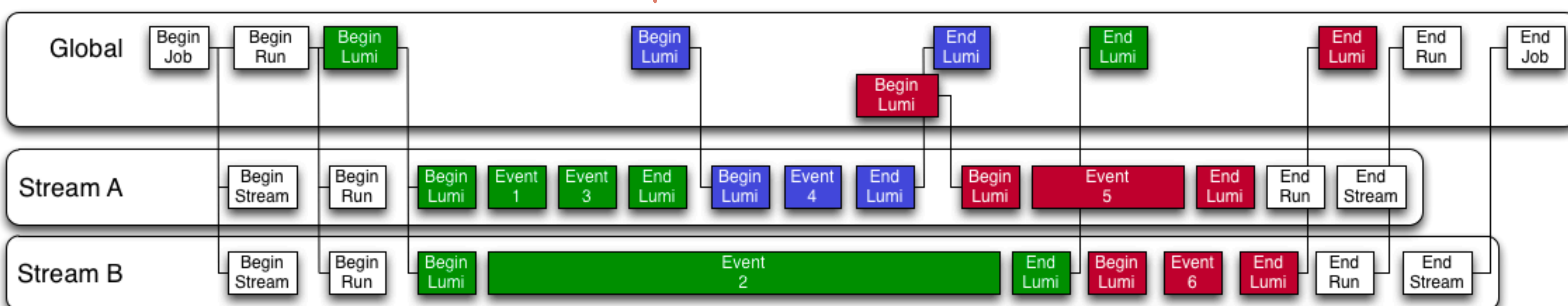


Multi-threaded framework in CMS

- → Concurrent GEN implementation based on concurrent infrastructure in CMSSW
- → Three modules types to inherit from: Global, Stream, and One [twiki]
  - ❖ Global and Stream are supported modules to enable multithreading
  - ❖ Global and Stream module:
    - capable to run concurrent lumi and event processing on multiple threads
    - choice specific to module design: whether a single event on a stream can be conveniently processed without seeing the global event info
  - ❖ One: only run one event at a time, all other threads stuck

example of phase transition in CMSSW for a Global/Stream module

run luminosity block concurrently

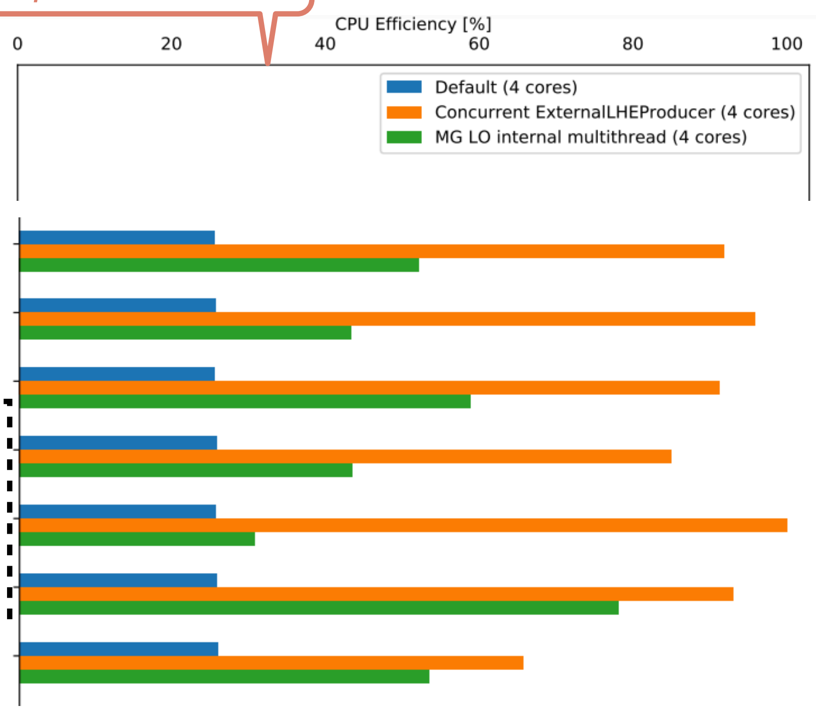


run events concurrently

Concurrent LHE generation

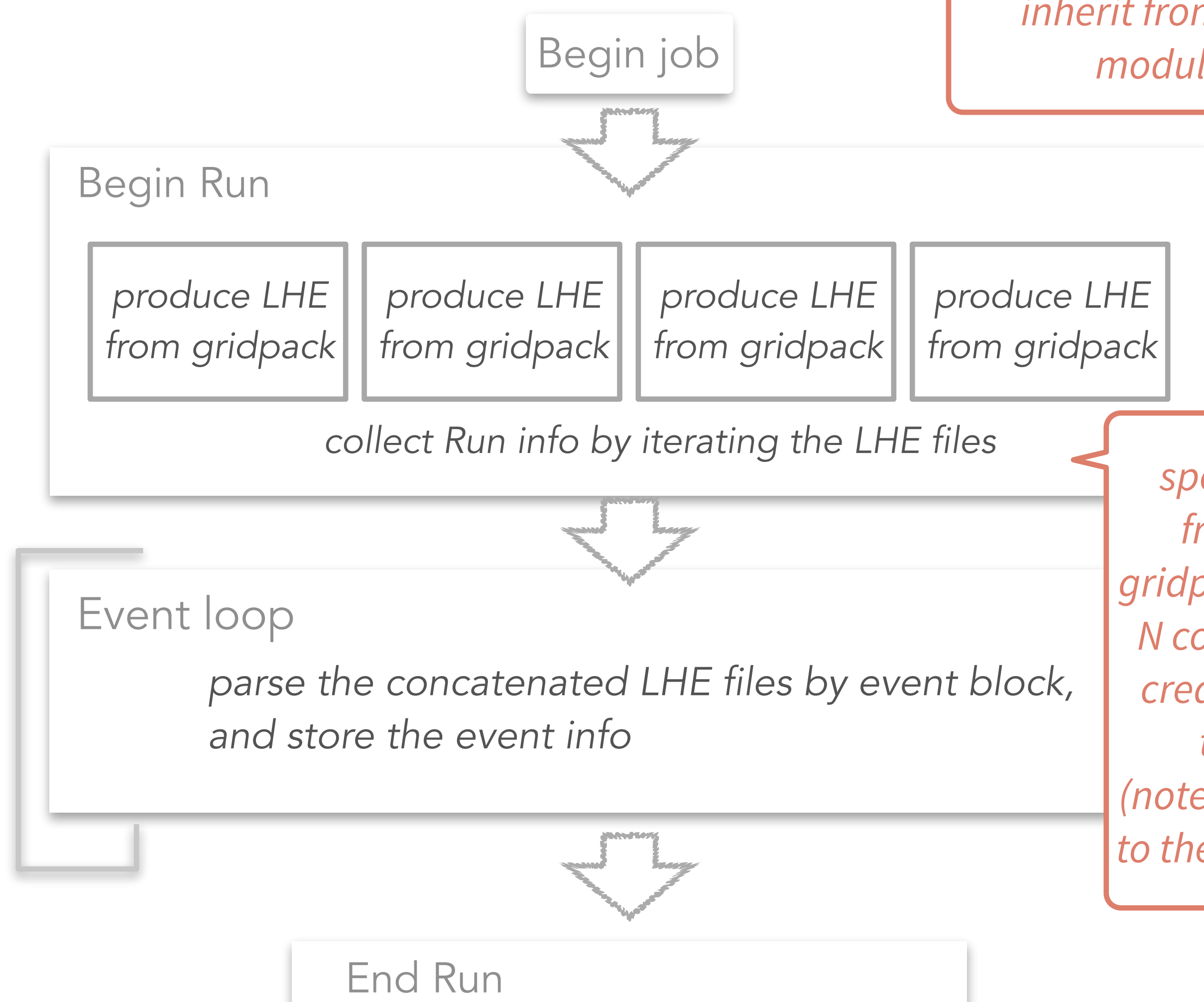
- ➔ Running LHE: the `ExternalLHEProducer` module,
  - ❖ LHEs produced at the start of the event processing chain
  - ❖ CMS uses the gridpack mechanism to produce LHE: an external tarball containing a sealed generator with information of a specific physics process (integration results, etc) ⇒ a black box to output LHE events
  - ❖ LHE production launched by an external script, not belonging to the standard CMS workflow
    - can use standalone method to enable multithreading

orange vs. blue: see CPU efficiency improvement



Generator processes for CMS official workflow

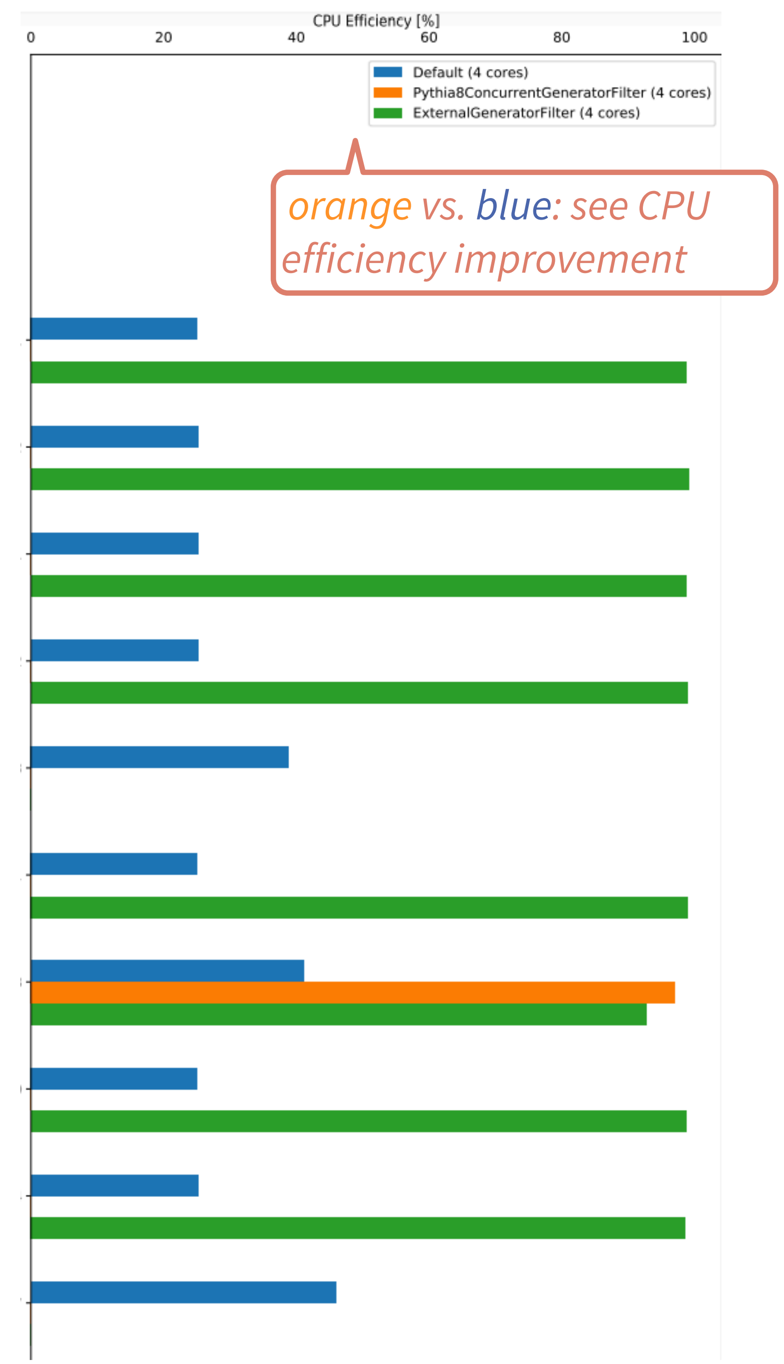
concurrent workflow, inherit from One module



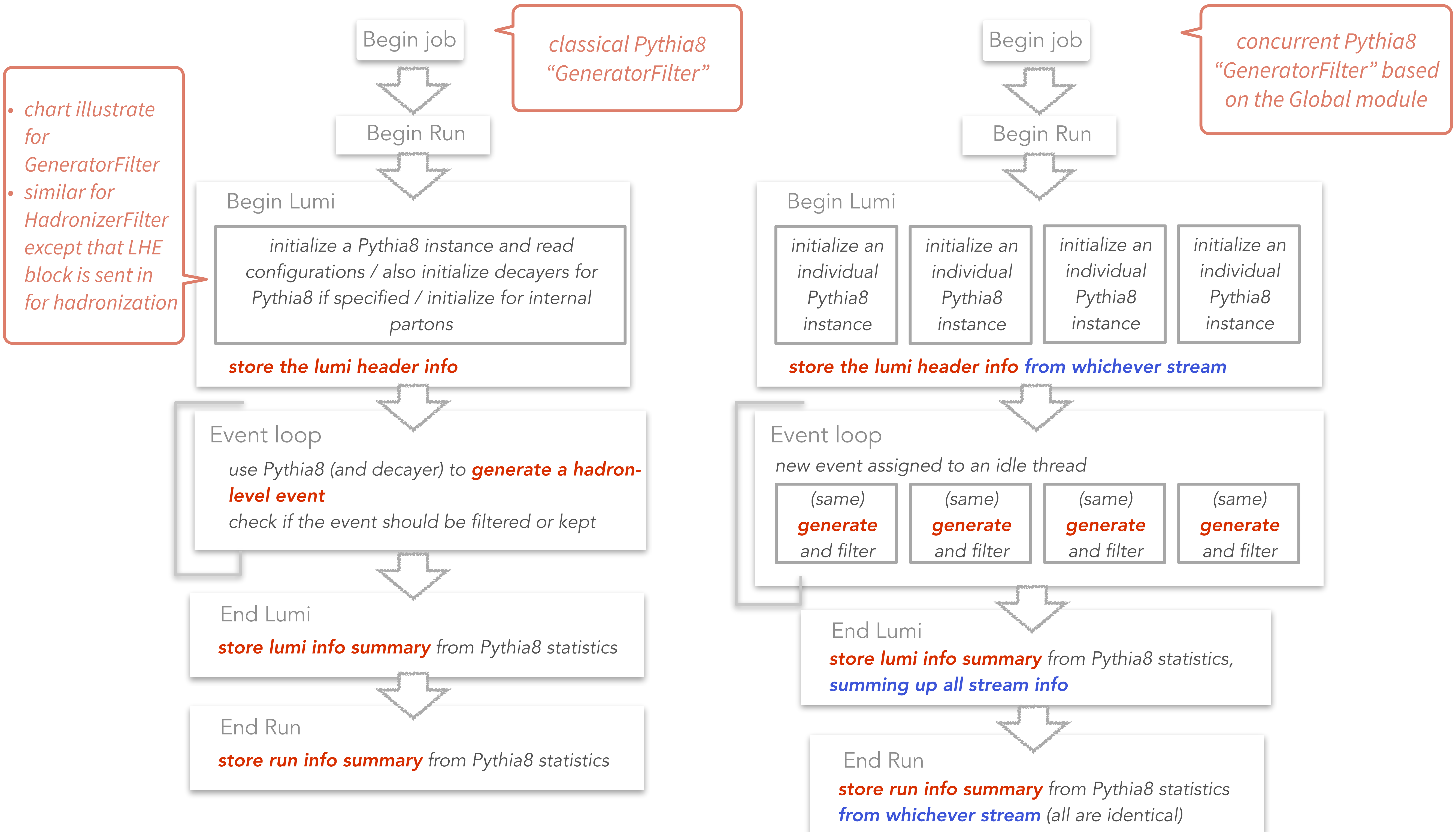
speed up comes from multiple gridpack processing: N concurrent tasks created using TBB task_group (note: this is external to the main instance)

Concurrent Pythia8 event processing

- Pythia8 is a widely-used MC generator for parton showering and hadronization
- Module for hadronization to produce HepMC-format output in CMS:
  - `GeneratorFilter` module: run a hadronized event from scratch with no LHE events as input
  - `HadronizerFilter` module: read an LHE event then hadronize it
- Pythia8 in cmssw has both modules
  - since Pythia8 may also interface to non-thread-safe generators e.g. Tauola, EvtGen; the general Pythia8 `GeneratorFilter/HadronizerFilter` module is not multi-threaded supported
  - standalone Pythia8 supports multi-threaded instance
  - new modules designed to only run Pythia8 without interfacing to non-thread-safe modules
    - named `ConcurrentGeneratorFilter/ ConcurrentHadronizerFilter`



Concurrent Pythia8 event processing (II)

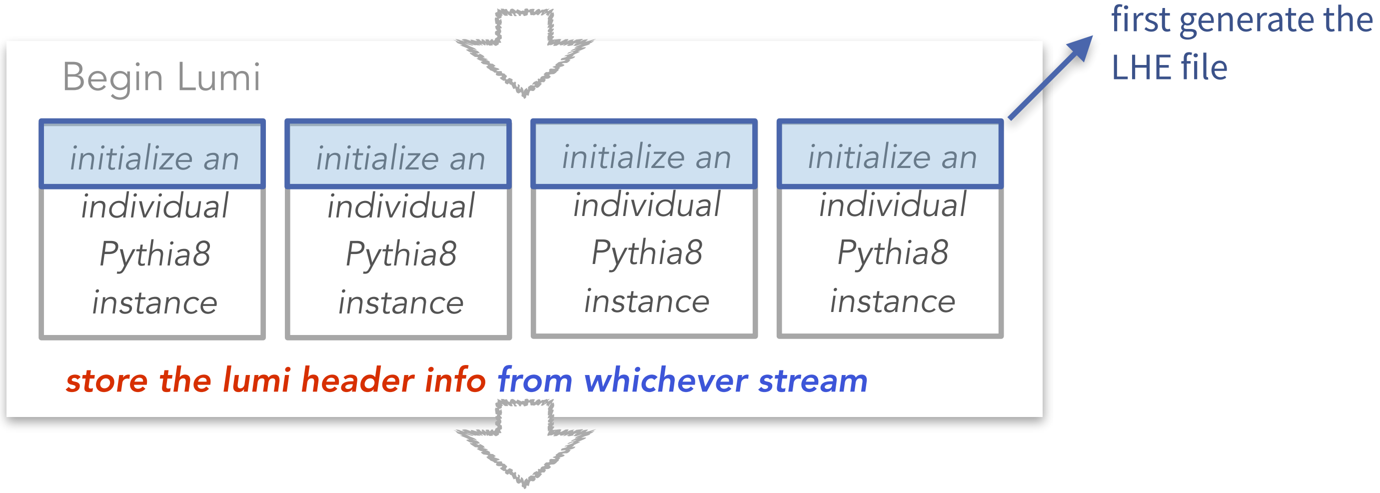


Concurrent Pythia8 event processing (III)

➔ Open questions

- next plan is to enable the "random gridpack" mode of Pythia8 GeneratorFilter
  - Pythia8 GeneratorFilter has a special feature to use customized LHEs produced from a large number of gridpacks for hadronization – used in SUSY search
  - idea is to specify NEvent/Nthread events to each thread to produce LHE in begin Lumi, then demand the event loop for that thread to hadronize that **fixed** number of LHEs – a little contradict to the concurrent design

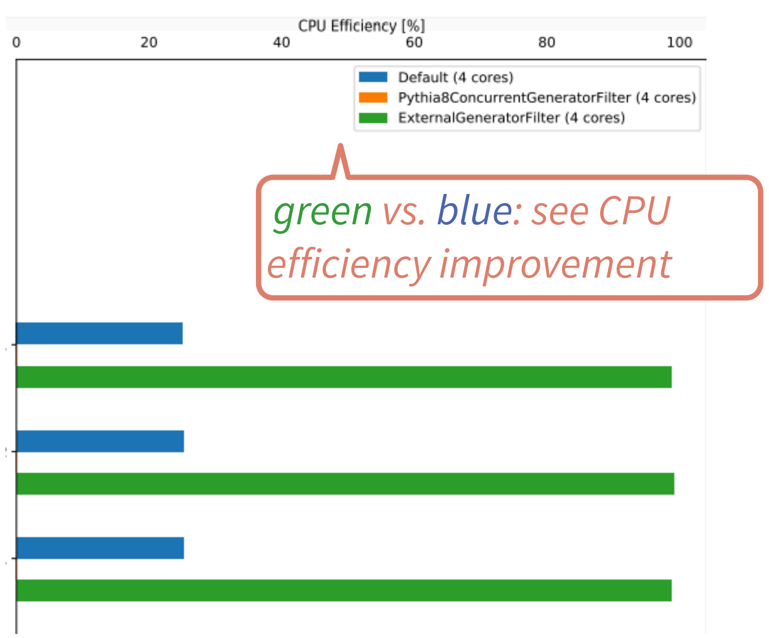
illustration of the special workflow to run LHE+GEN in the GeneratorFilter module

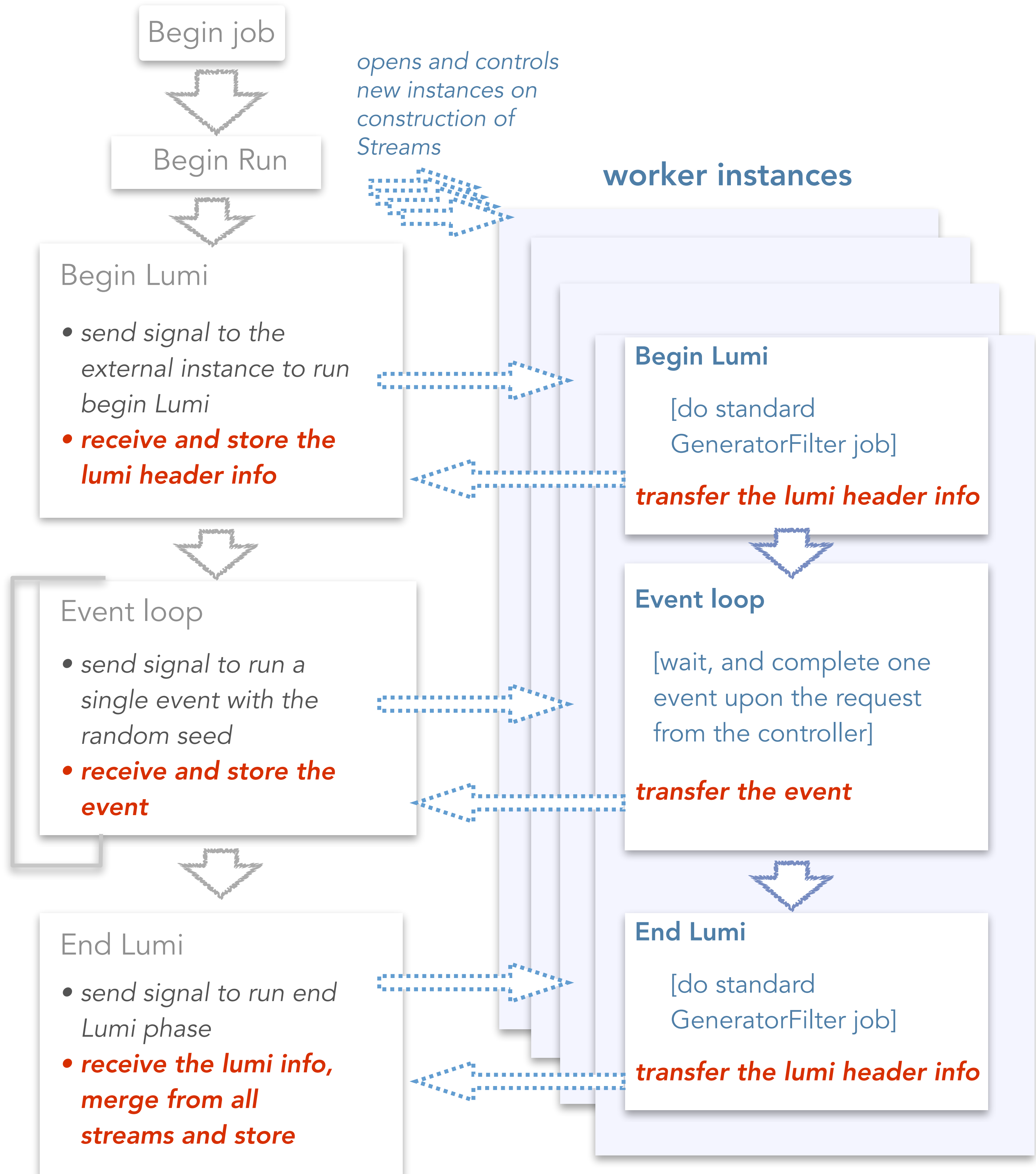


External solution for GeneratorFilter (I)

- The module `ExternalGeneratorFilter` is developed to extend the concurrency ability to the classic `GeneratorFilter`
  - an interprocess solution, further beyond the current multi-threaded infrastructure
    - each stream opens a new external instance (workers) and runs a GeneratorFilter on it
    - produced event info and lumi/run summary info transferred from the external worker to each stream
  - eternally solve the thread-safety issue because different instances do not share the memory with each other

Generator processes for CMS official workflow





External solution for GeneratorFilter (II)

- Open questions
  - will some type of GeneratorFilter modules (specific generator classes) destroy the inter-process communication?
    - a) Sherpa GeneratorFilter enables MPI internally – will multiple Sherpa instances interact within the MPI mechanism which we do not expect?
    - b) previously see that random job failures in the worker (for Sherpa case) could send the unexpected signal and mess up the interprocess communication
  - next plan is to write a similar module for HadronizerFilter to also benefit the LHE+Pythia8 (with decayers) workflow
    - this workflow takes up the majority cases which has not supported multithreading (10% of total GEN events)

Overall picture of concurrent GEN in CMS

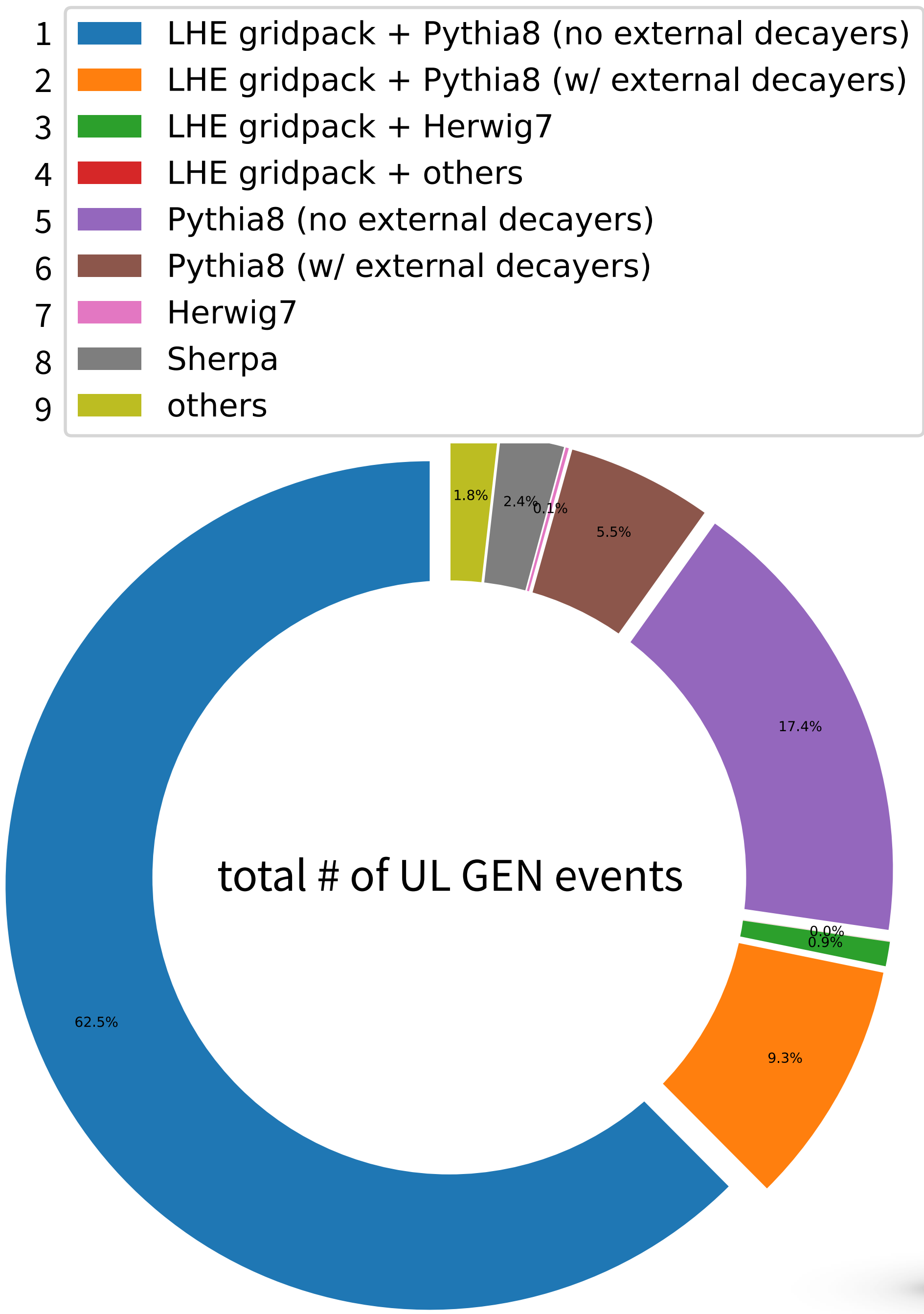
Summary of concurrent GEN

#1,3,4,5 CENTRALLY DEPLOYED

		campaign	concurrent mechanism	application scope
1	*concurrent ExternalLHE-Producer*	LHE step	a general scheme in LHE production that launches the external LHE production script in multiple separate instances	for all `ExternalLHE-Producer` module
2	*internal MG5 LO multi-thread*	LHE step	use MG5's internal "read-only" gridpack mode to enable multithreading	for `ExternalLHEProducer` module using MG5 LO gridpacks (≥ V2.6.1)
3	*Pythia8-Concurrent-HadronizerFilter*	GEN step after LHE	enable to run P8 concurrently, each thread using an individual P8 instance (module based on `edm::global`); set a dummy external decayer interface	for `Pythia8HadronizerFilter` module without `ExternalDecays` set
4	*Pythia8-Concurrent-GeneratorFilter*	GEN-only step	same as above	for `Pythia8GeneratorFilter` module without `ExternalDecays` set
5	*External-GeneratorFilter*	GEN-only step	run `GeneratorFilter` concurrently by controlling an external cmsRun instance for each thread (module based on `edm::global`)	for all modules inherit from `GeneratorFilter` & belong to GEN-only campaign (i.e. except `Herwig7GenertorFilter` usage after LHE)

Requests do/do not support concurrent GEN

- ➔ Summary from the perspective of the generator type
  - ❖ which GEN requests do or do not support the multi-thread methods?

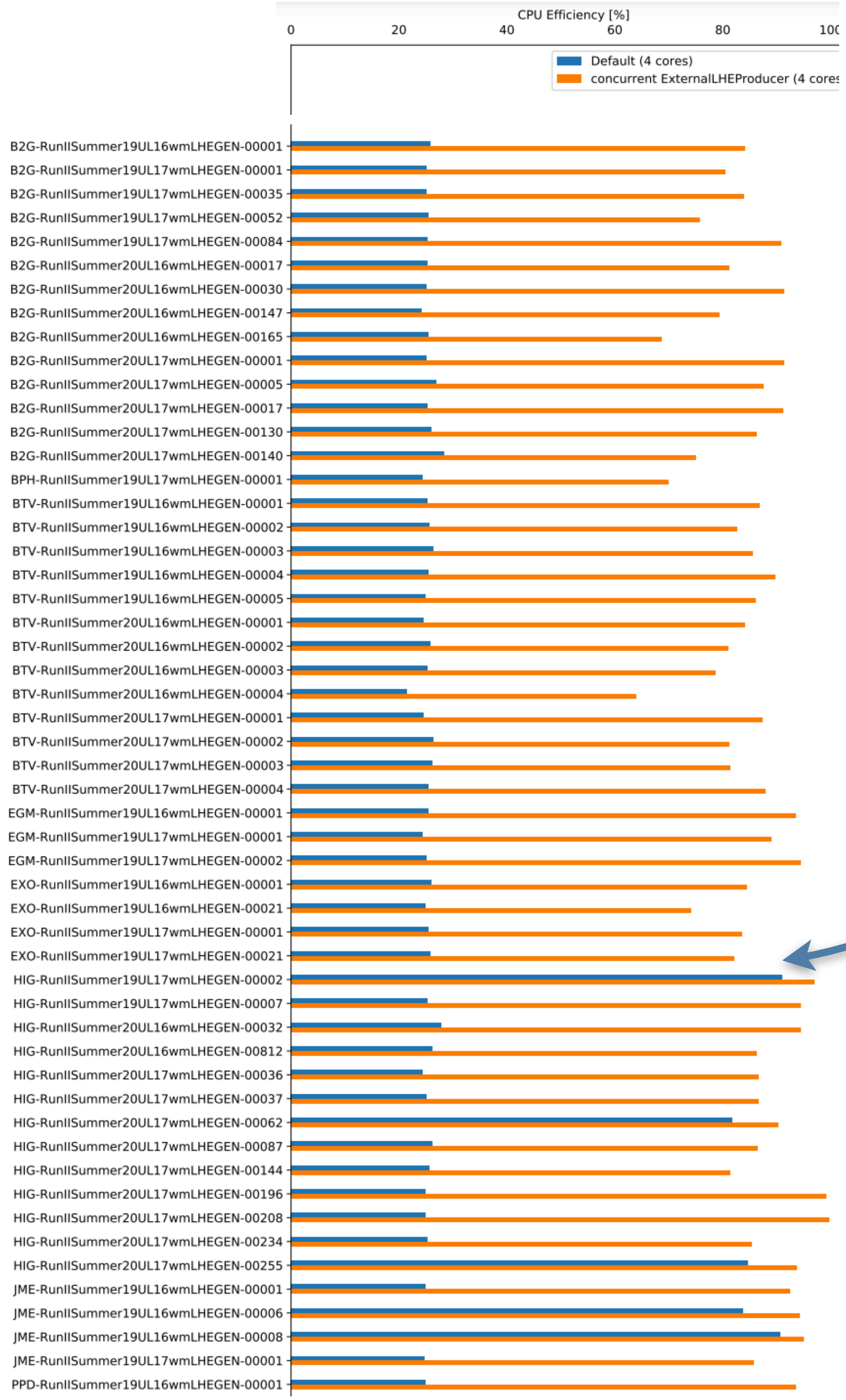


1. MT supported: #1 + #3
2. MT partially supported: #1 only
   - reason: no concurrent method available for Pythia8HadronizerFilter using a thread-hostile external decayers (EvtGen, Tauola, Photon)
3. MT partially supported: #1 only
   - reason: Herwig7GeneratorFilter does not support to use ExternalGeneratorFilter for MT if it directly reads the LHE file (Herwig7HadronizerFilter with MT support is under development to adapt closer to this case)
4. (no such cases so far)
5. MT supported: #4
6. MT supported: #5
   - for 5, 6: need to rule out the rare case when LHE gridpacks are used inside a Pythia8GeneratorFilter - in such cases, the parameter RandomizedParameters for Pythia8GeneratorFilter is set
7. MT supported: #5
8. MT supported: #5
9. MT supported for particular GeneratorFilter-based cases (match items in table): #5
   - for cases other than using a GeneratorFilter-based module: no MT support developed (may include FlatRandomPtGunProducer, FlatRandomEGunProducer, Pythia8PtGun, Pythia8PtAndDxyGun…)

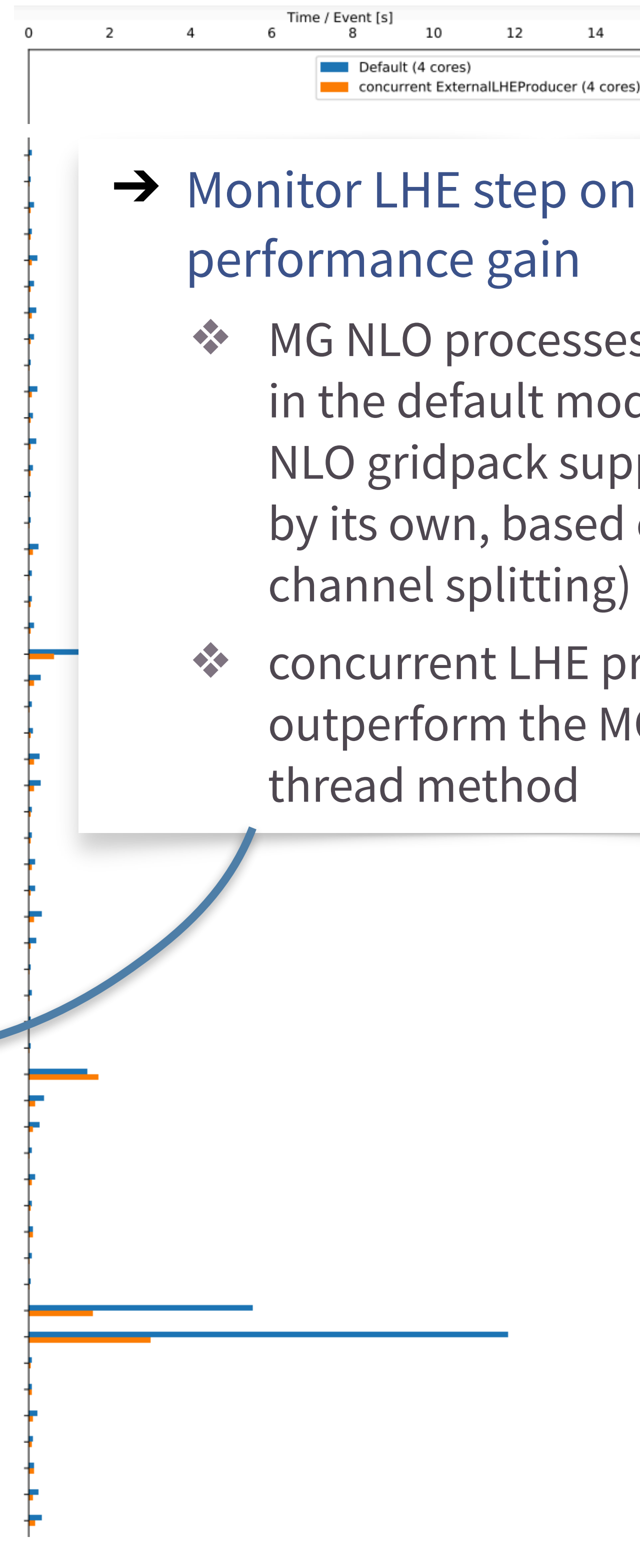
Validation - performance of concurrent LHE

compare default (blue) vs. #1 (orange), both nThreads=4

CPU efficiency plot [full PDF link]



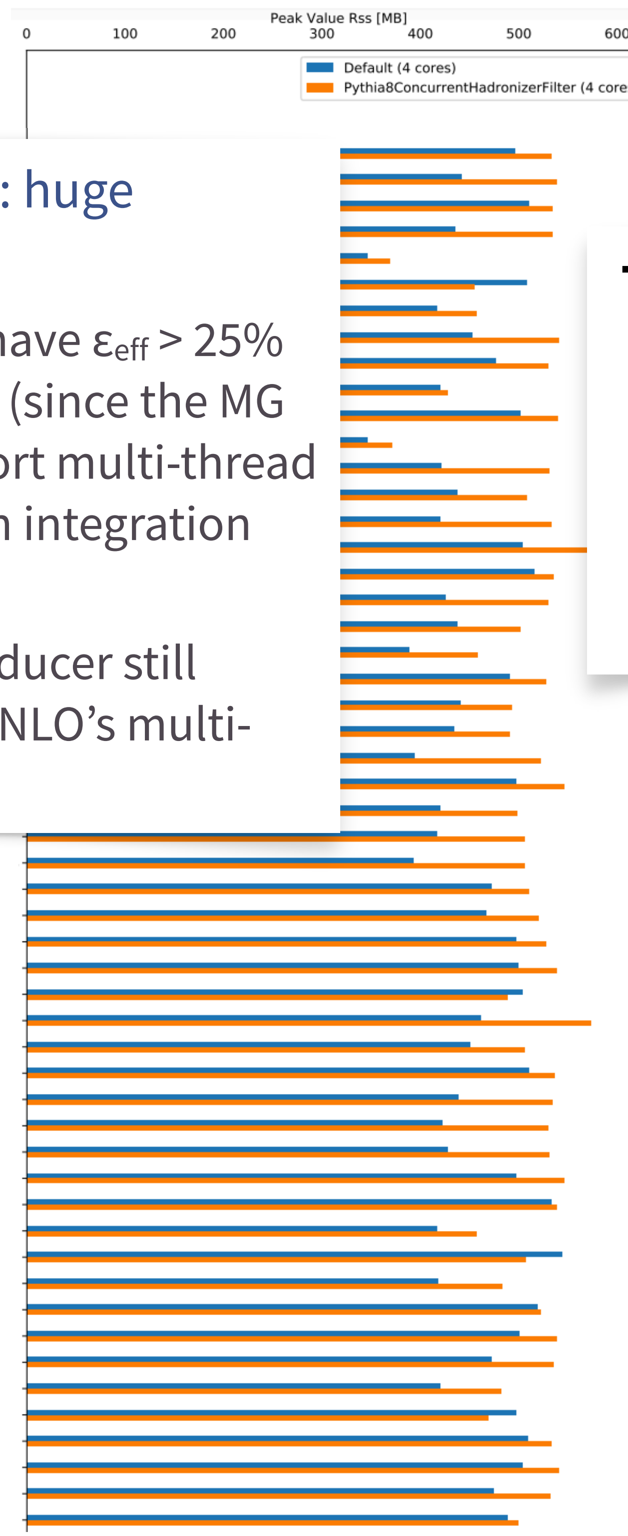
time / event plot [full PDF link]



➔ Monitor LHE step only: huge performance gain

- MG NLO processes have $\varepsilon_{eff} > 25\%$ in the default mode (since the MG NLO gridpack support multi-thread by its own, based on integration channel splitting)
- concurrent LHE producer still outperform the MG NLO's multi-thread method

RSS plot [full PDF link]

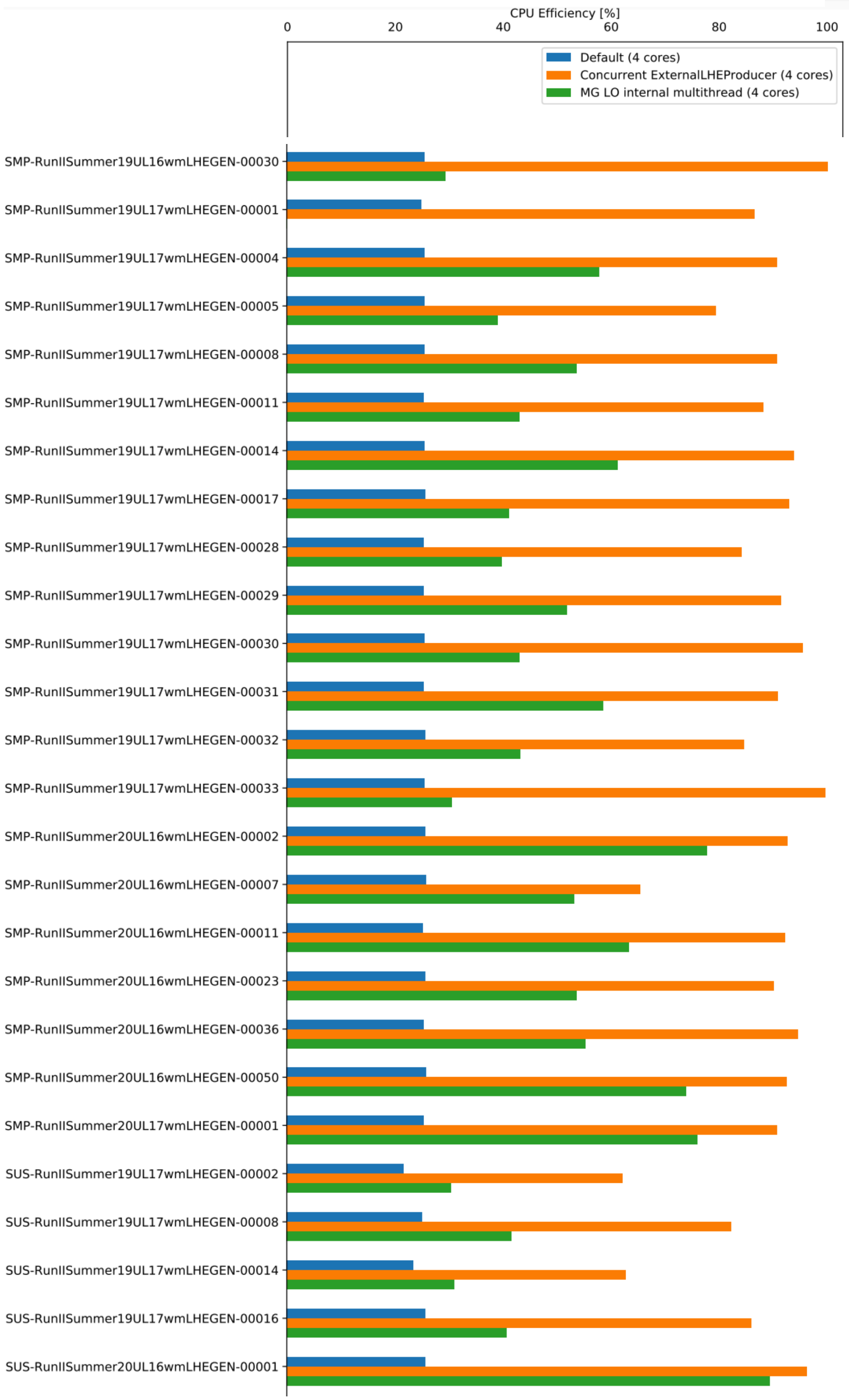


➔ A bit more RAM occupied using concurrent LHE Producer

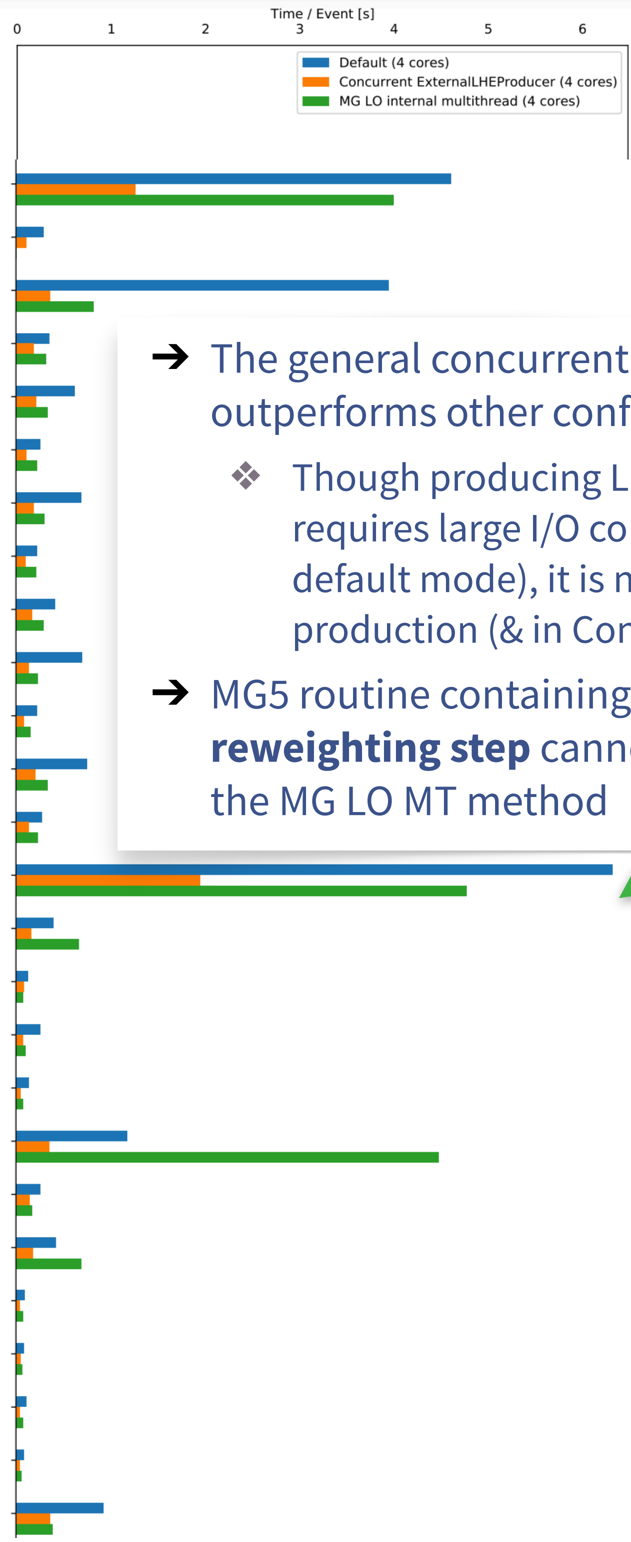
Validation - performance of concurrent LHE

compare default (blue) vs. #1 (orange) vs. #2 (green), both nThreads=4

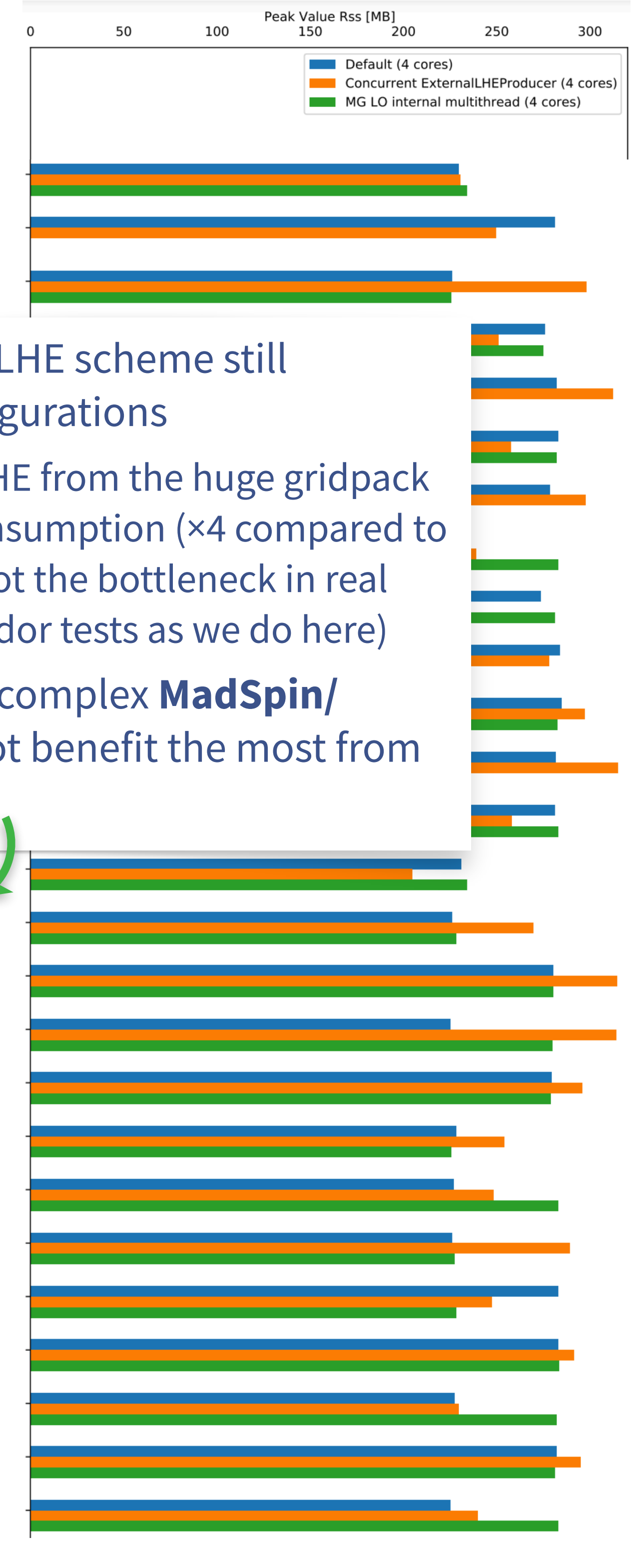
CPU efficiency plot [full PDF link]



time / event plot [full PDF link]



RSS plot [full PDF link]



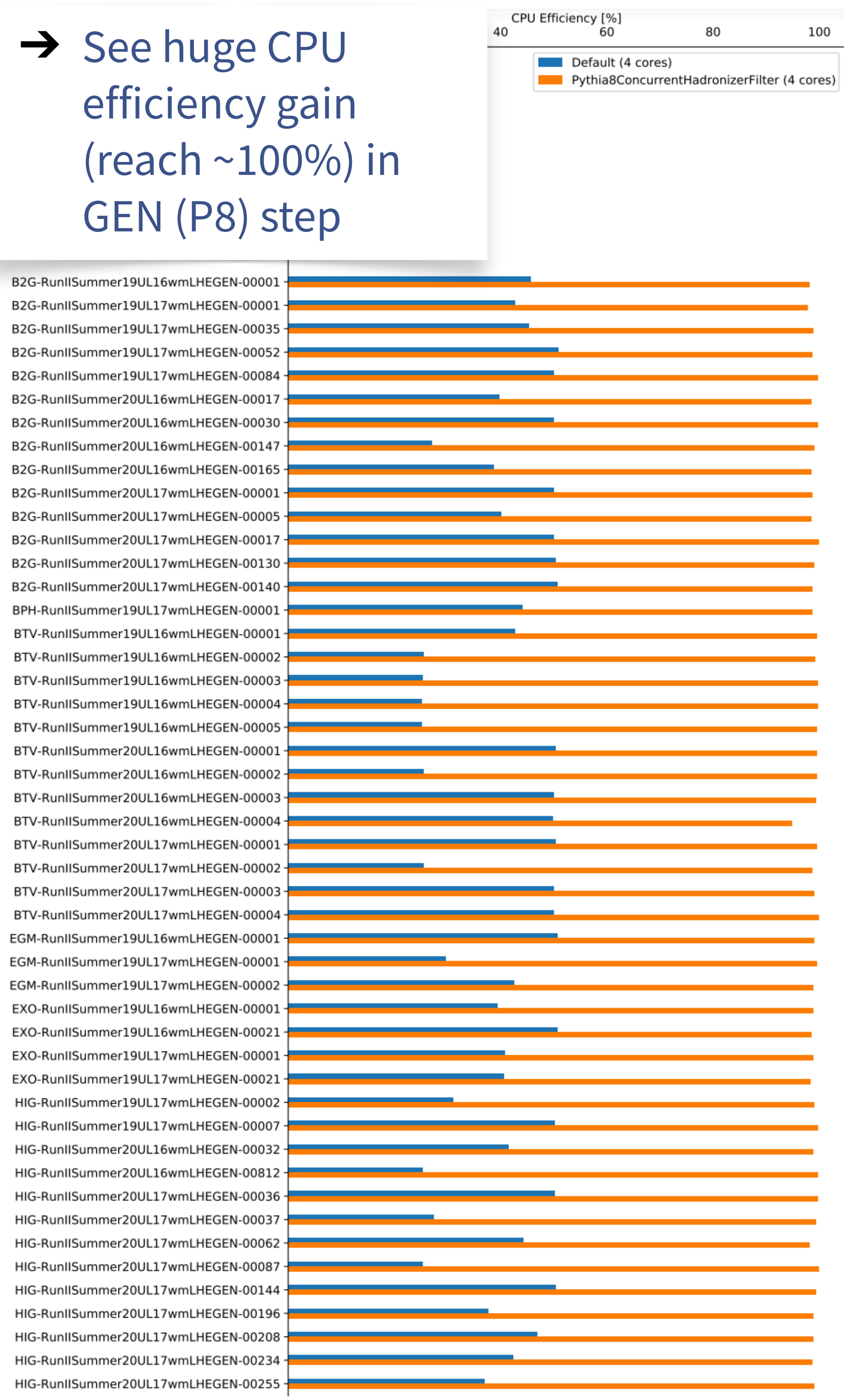
- ➔ The general concurrent LHE scheme still outperforms other configurations
  - ❖ Though producing LHE from the huge gridpack requires large I/O consumption (×4 compared to default mode), it is not the bottleneck in real production (& in Condor tests as we do here)
- ➔ MG5 routine containing complex **MadSpin/ reweighting step** cannot benefit the most from the MG LO MT method

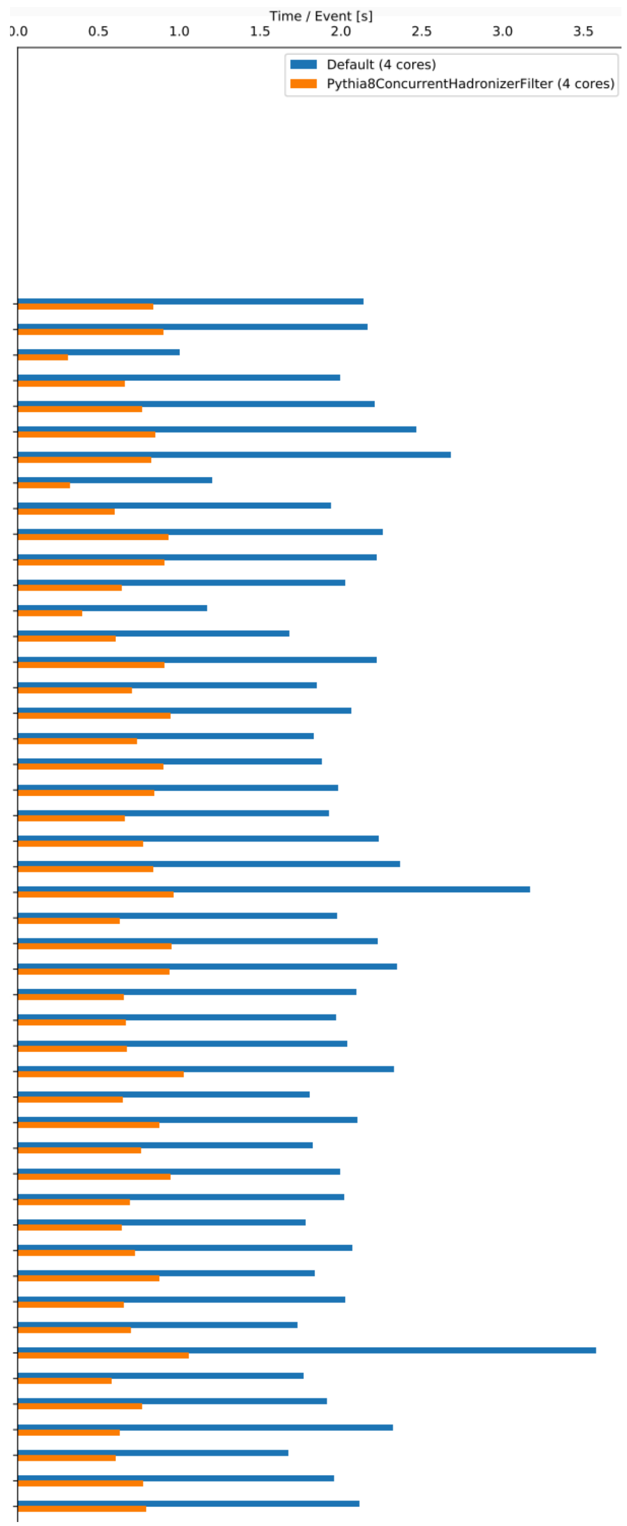
Validation - performance of concurrent GEN (following LHE)

compare default (blue) vs. #3 (orange), both nThreads=4

CPU efficiency plot [full PDF link] | *time / event plot [full PDF link]* | *RSS plot [full PDF link]*

→ See huge CPU efficiency gain (reach ~100%) in GEN (P8) step



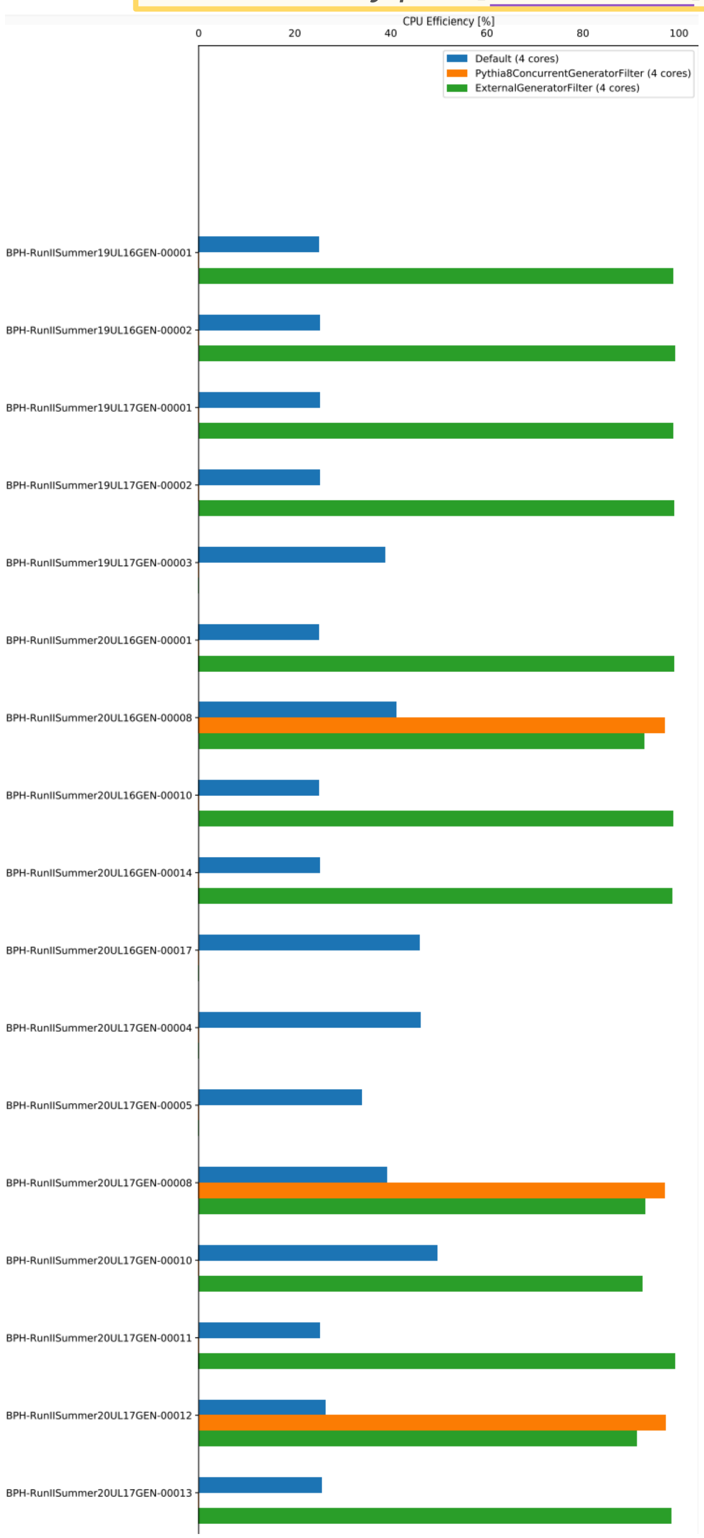


→ A bit more RAM occupied using concurrent P8 hadronizer

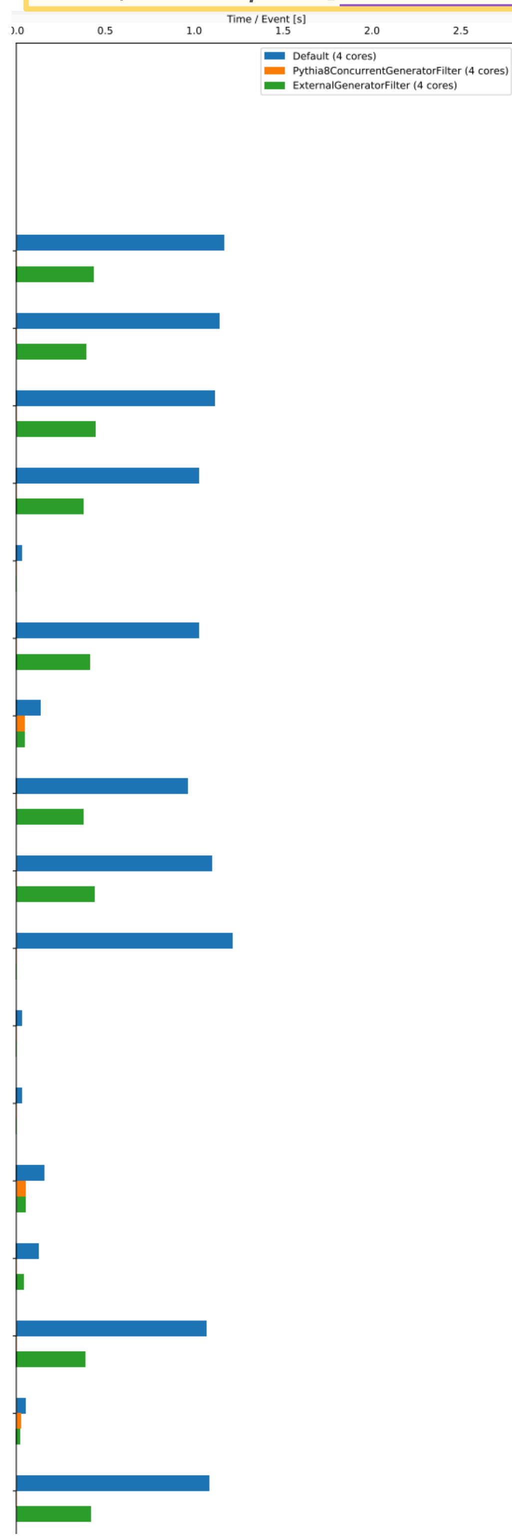
Validation - performance of concurrent GEN-only

compare default (blue) vs. #4 (orange) vs. #5 (green), with nThreads=4

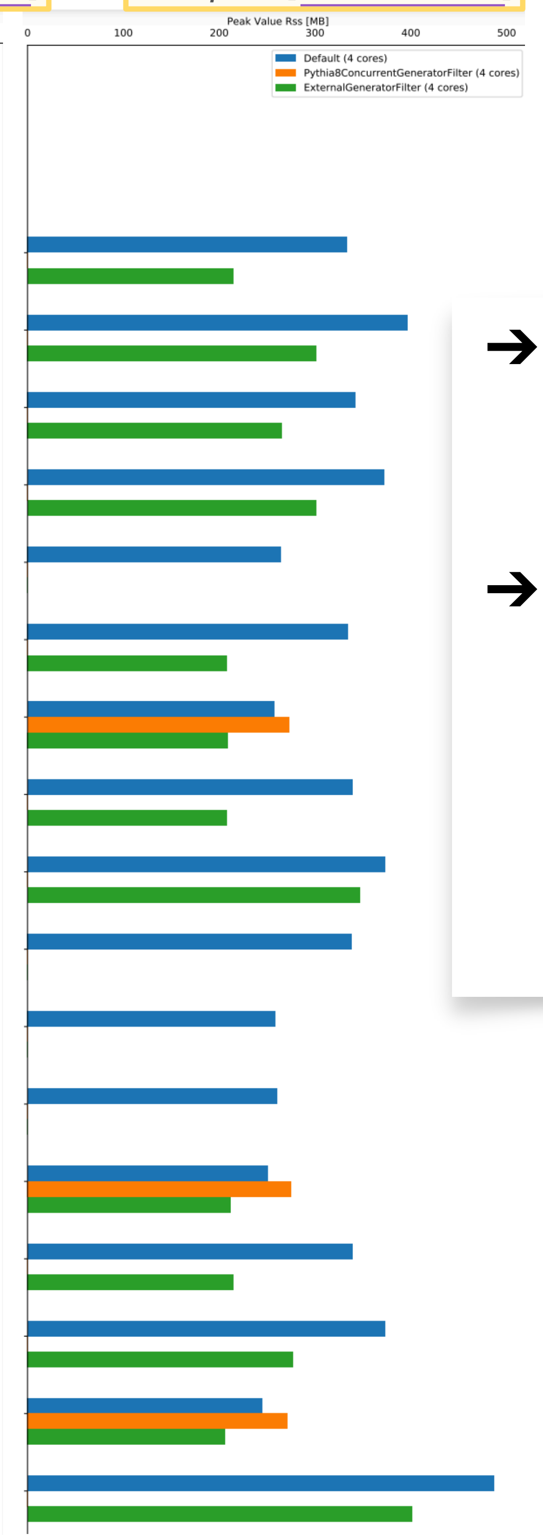
CPU efficiency plot [full PDF link]



time / event plot [full PDF link]



RSS plot [full PDF link]

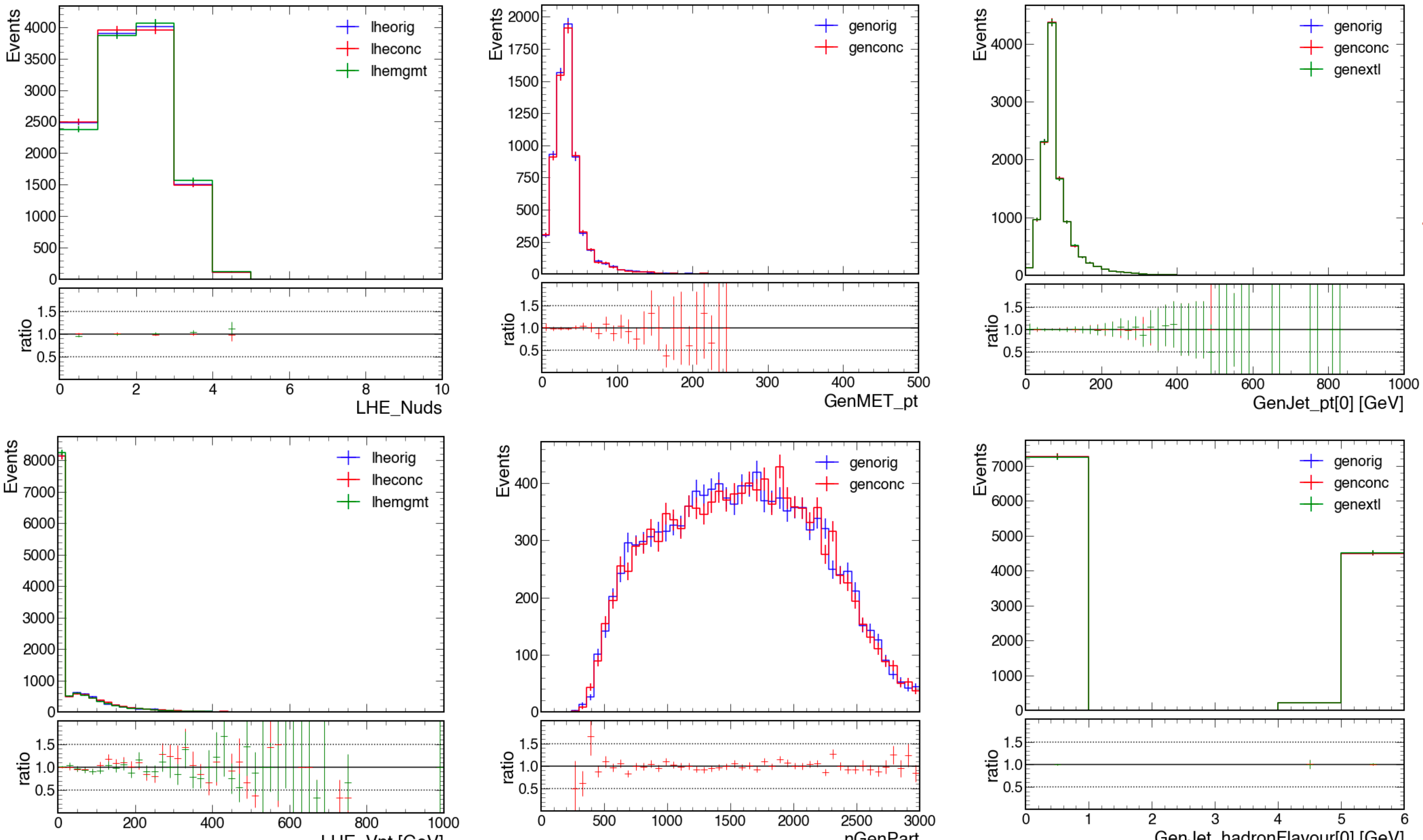


- ➔ See huge CPU ε_{eff} gain switching to either two cases (4 or 5)
- ➔ Less RSS from default → ExternalGeneratorFilter (5)
  - ❖ the RSS cost does not trace the memory consumption in the external process

Physics validation

➔ Validate if the single-threaded and multi-threaded mode give the same physics result

- examine the generator-level physics kinematics
- ~350 existing CMS GEN processes with different generator types are validated



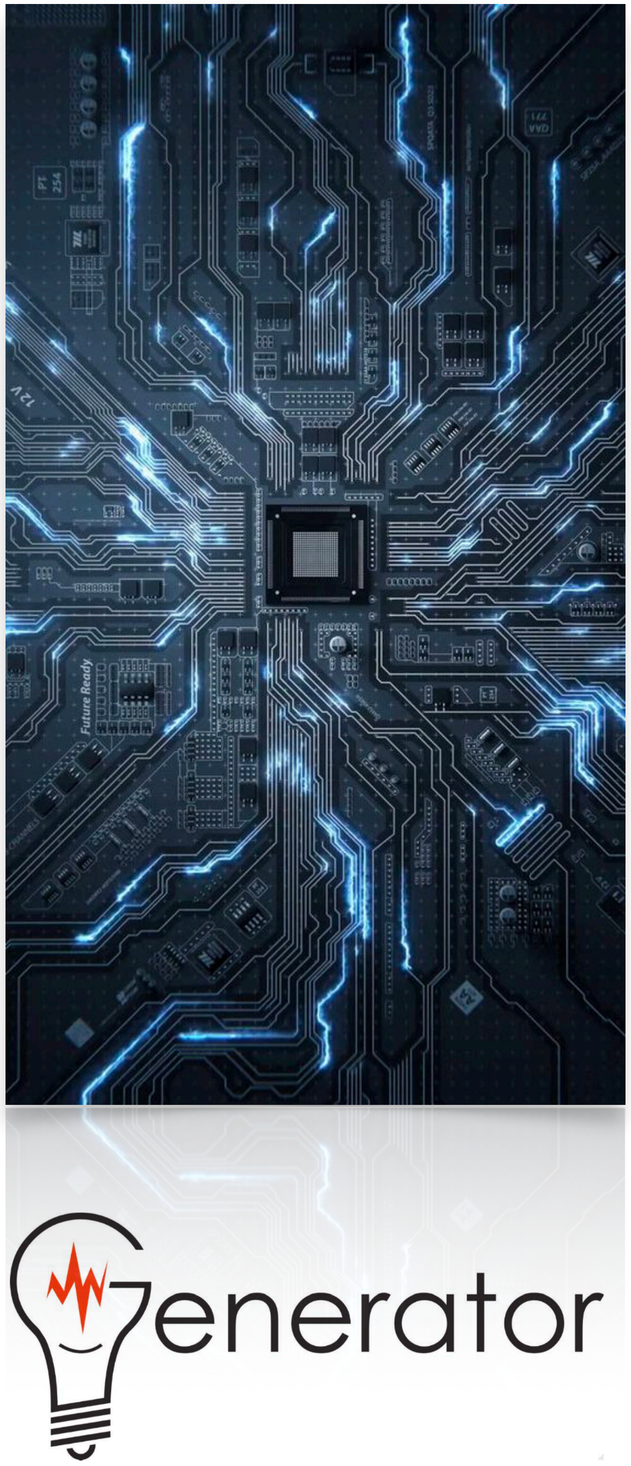
see good agreement in overall

Use in CMS grid computing

- ➔ Concurrent GEN are set as default configuration since September this year
- ➔ Enables to run the whole event processing chain in StepChain
  - ❖ StepChain vs. TaskChain: StepChain runs all steps (GEN, SIM, …) in one job; TaskChain contains a set of grid jobs for different procedures. StepChain recommended over TaskChain
  - ❖ GEN step without multi-core supports generally have low CPU efficiency → will be assigned to run GEN in TaskChain (with 1 thread) and others in StepChain (with multi threads)
  - ❖ now can run all steps in StepChain with multiple threads, without worrying the CPU eff

Conclusion

- ➔ In CMS, various generator multithreading utilities developed thanks to the work of many contributors
  - ❖ many modules developed thanks to Chris et al
  - ❖ twiki WorkBookGenMultithread setup to provide recipes for CMS users and Generator contacts
  - ❖ all current methods well-validated in physics and see good computing performance
- ➔ Some next-ups in the module development
  - ❖ hope to cover all generator configurations currently in use



Backup