CMS IO Overview

Brian Bockelman
Scalable IO Workshop

Topics I Want to Cover

- Goal for today is to do as broad an overview of “CMS IO” as possible:
  - Outline what “CMS Computing” actually does.
  - Characterize our workflows in terms of IO and high-level semantics.
  - Hit on a few file format requirements.
  - Outline pain points and opportunities.

CMS Computing

- The CMS Offline & Computing organization aims to:

  1. Archive data coming off the CMS detector.

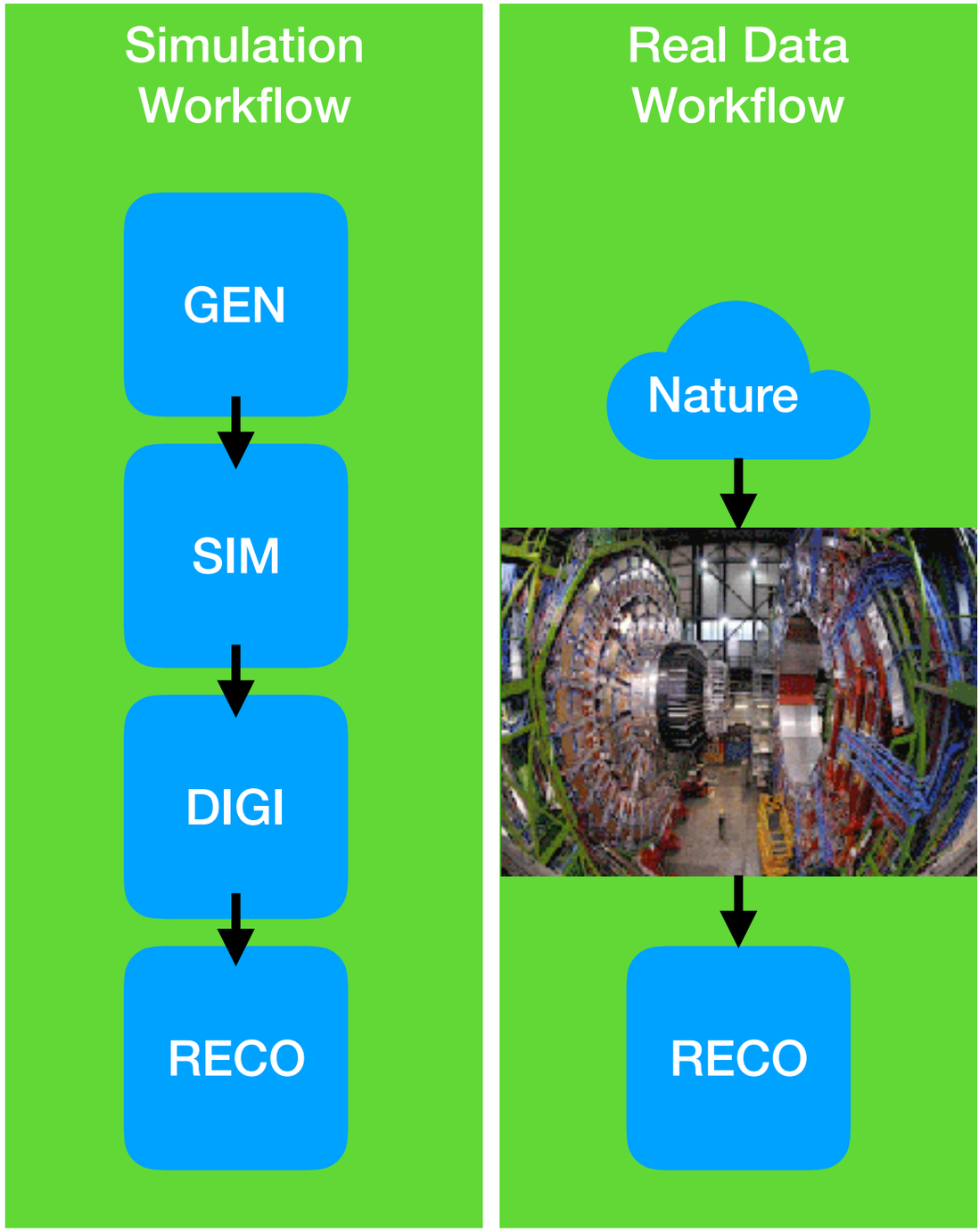
  2. Create the datasets required for CMS physicists to perform their research.

  3. Make resources (software, computing, data, storage) available necessary for CMS physicists to perform their analyses.

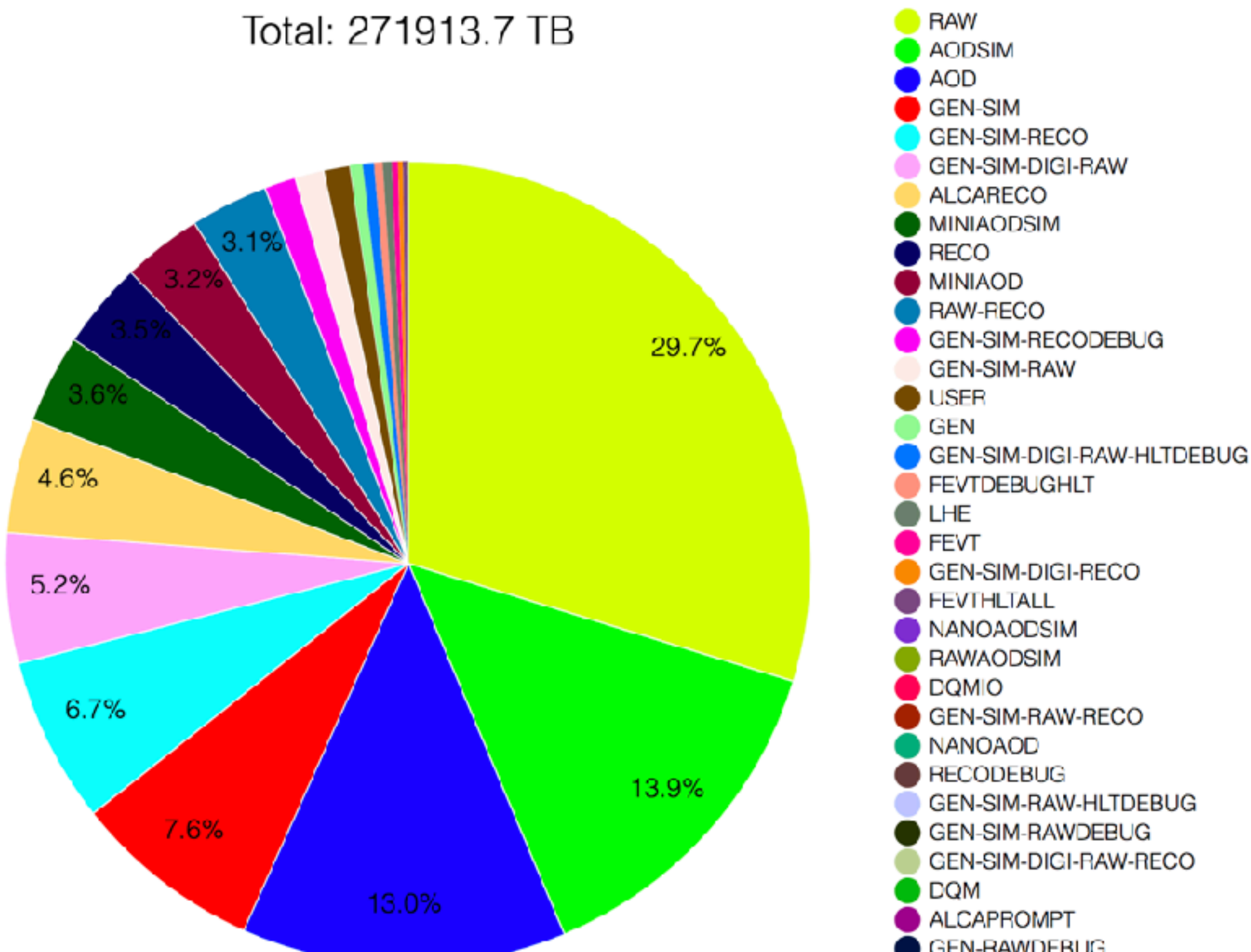
- As we're at a *scalable IO* workshop, I'll focus mostly on item (2) above.

Dataset Production

- What do we do with datasets?
  - **Process data** recorded by the detector — convert raw detector readouts to physics objects.
  - **Simulate data** from simulated particle decay to corresponding physics objects.



CMS Datasets



Distributed Computing in CMS

- CMS is a large, international collaboration - computing resources are provided by several nations.
    - Aggregate US contribution (DOE + NSF) is about 30-40% of total.
- The “atomic” unit of processing in HEP is the event.
- Multiple events can be processed independently, leading to a pleasantly parallel system.
    - E.g., given 1B events to simulate and 10 sites, one could request 100M events from each site.
- In practice, dataset processing (or creation) is done as a **workflow**. Entire activity is split into distinct tasks, which are mapped to independent batch jobs that contain dependency requirements.

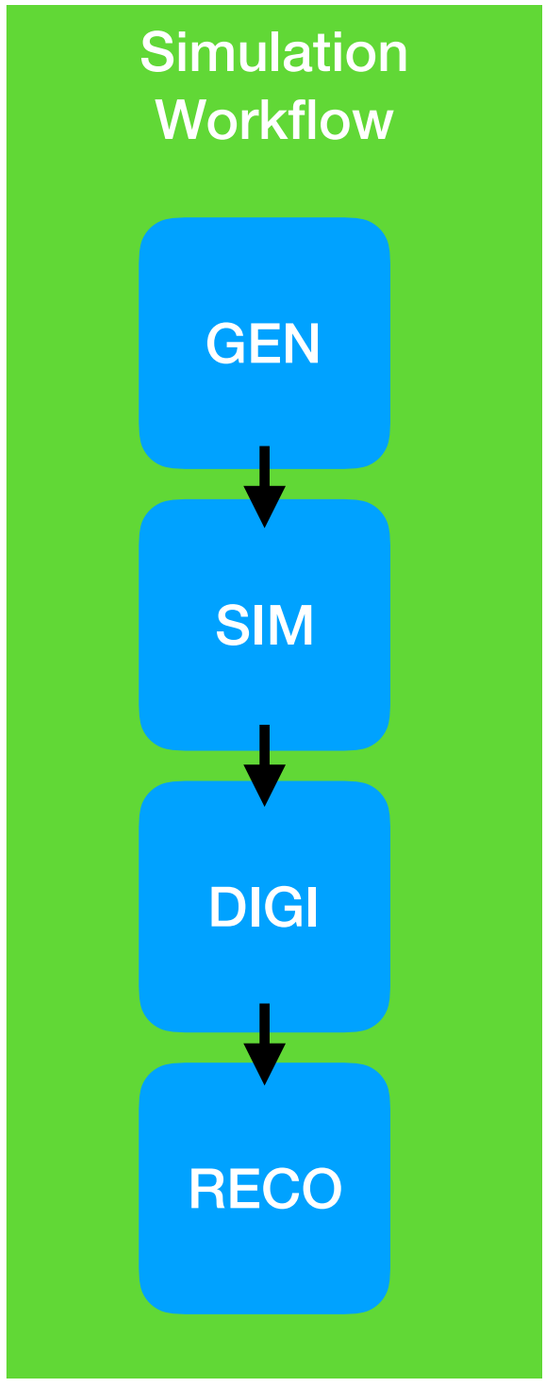
In the last 24 Hours	
497,000	Jobs
4,058,000	CPU Hours
7,765,000	Transfers
241	TB Transfers
In the last 30 Days	
12,855,000	Jobs
130,265,000	CPU Hours
151,736,000	Transfers
13,554	TB Transfers
In the last 12 Months	
143,895,000	Jobs
1,577,979,000	CPU Hours
2,185,186,000	Transfers
188,000	TB Transfers

OSG delivered across 125 sites

CMS Scale

- We store data for analysis and processing at about **50 sites**.
- Total of **272PB data volume**.
- **300,000 distinct datasets, 100M files**.
- **330PB / year inter-site** data transfers.
- Typically have 500 workflows running at a time utilizing ~150k (Xeon) cores.

Simulation Workflow



- **GEN** (generation): Given a desired particle decay (config file), determine its decay chain.
- **SIM** (simulation): Given output of GEN (particles traveling through space), simulate their interaction with the matter & magnetic field of the CMS detector.
- **DIGI** (digitization): Given particles' interaction with CMS detector, simulate the corresponding electronics readout.
- **RECO** (reconstruction): Given the (simulated) detector readouts, determine the corresponding particles in the collision.
    - NOTE: in a perfect world, the output of **RECO** would be the same particle decays that came from **GEN**.

Simulation Jobs

- Not currently possible to run the entire pipeline as a single process. We have 5 steps :
  - **GEN-SIM**: Depending on the generator used, GEN may run as a sub-process. Output is “GEN” or “GEN-SIM” format. Typically temporary intermediate files.
  - **DIGI**: Input is GEN-SIM; output is RAWSIM. Always temporary output (deleted once processed).
  - **RECO**: Input is RAWSIM; output formats are:
    - RECOSIM: Physics objects plus “debug data” about detector performance. Rarely written out!
    - AODSIM: Physics objects; strict subset of RECO. *Archived on tape.*
  - **MiniAOD**: Input AODSIM; output **MINIAODSIM**. Highly reduced - but generic - physics objects (10x smaller than AODSIM). Usable by 95% of analyses.
  - **NanoAOD**: Input MINIAODSIM; output **NANOAODSIM**. Highly reduced MINIAOD (10x).
    - NEW for 2018. Goal is usable for 50% of analyses — yet to be proven!
- Work is ongoing to have 5 processes be all in a single job.

SORRY: We refer to the data format and the processing step by the same jargon!

Simulation Details

- We run a workflow for each distinct physics process we want to simulate - this may be 10k distinct configurations resulting in 40-50k datasets.
  - Specialized physics processes may require only 10k events.
  - Common samples may be 100M events.
- Right now, each simulation step may be a separate batch job.
  - Significant effort over the last two years to combine all steps to a single batch job, eliminating intermediate outputs.

GEN is *hard*

- Our most commonly used generator (madgraph) sees *significant* CPU savings per job if we pre-sample the phase-space.
  - Output of this pre-sample is a “grid pack”.
  - Worst case, the gridpack is a 1GB tarball with thousands of files.
  - Each job in a workflow must re-download a tarball and unpack the tarball.
  - Worst case jobs are 2-3 minutes and single-core (generator crashes if run longer in these configs).
- Tough case to solve as generators are maintained by an external community, not CMS.
- **Don’t worry about this case**. Works poorly everywhere and we need to fix this regardless.

DIGI is *hard*

- DIGItization is simply simulating the electronics readout given the simulated particle interactions. **That's not CPU-intensive. Why is it hard?**
  - Particle bunch crossings occur at 40MHz.
  - Multiple collisions occur during bunch crossings.
  - Electronics "reset" slower than the interaction rate.
- A single readout of an interesting ("**signal event**") will contain the remnants of ~200 boring ("**minbias**" or "**background**" events).
  - **So digitization of a single simulation event requires reading 201 raw events from storage.**

Cracking DIGI

- The readouts are additive: we can precompute (“premix”) the background noise and reuse these over and over.
    - Precompute this is our highest-IO workflow: 10-20MB/s per core.
- Options:
    - 40TB library of background noise (1 collision per event); reading 200 events from this library per 1 simulated event. We call this “**classic pileup (PU)**”
    - 600TB library of premixed background noise (200 collisions per event); read 1 event from this library per 1 simulated event. This is “**premixed PU**”.
        - To boot, reduces DIGI-RECO CPU time by ~2x.

Job Types, Parallelism, IO

- **GEN-SIM**: Input is “minimal” (modulo the gridpack issue). Output is in range of 10’s KB/s / core.
  - Generator is (often) single threaded now. Simulation scales linearly to 8+ cores.
  - Amdahl’s law says per-job speedup is limited by the ratio of GEN versus SIM time.
- **DIGI**: Input is signal GEN-SIM (100’s KB/s / core) output is 10s KB/s / core.
  - Classic PU case: background data 5-10MB/s / core. 2-4 cores.
  - Premixed PU case: background data is 100’s KB/s. 8 cores.
- **RECO**. Reconstruction of actual detector data. Input is 100’s KB/s / core; output is 10’s KB/s / core. 8 cores.

Other Workflows

- Processing detector data is “simple” compared to simulation — the RECO step must be run (and creation of the corresponding analysis datasets).
  - Detector data is organized into about 20 different “streams”, depending on physics content. Far simpler than the simulation case.
- Several specialty workflows - such as studies of alignment and calibration (ALCA) - that do not drive CPU budget.
- Purposely not touching user analysis in this presentation.

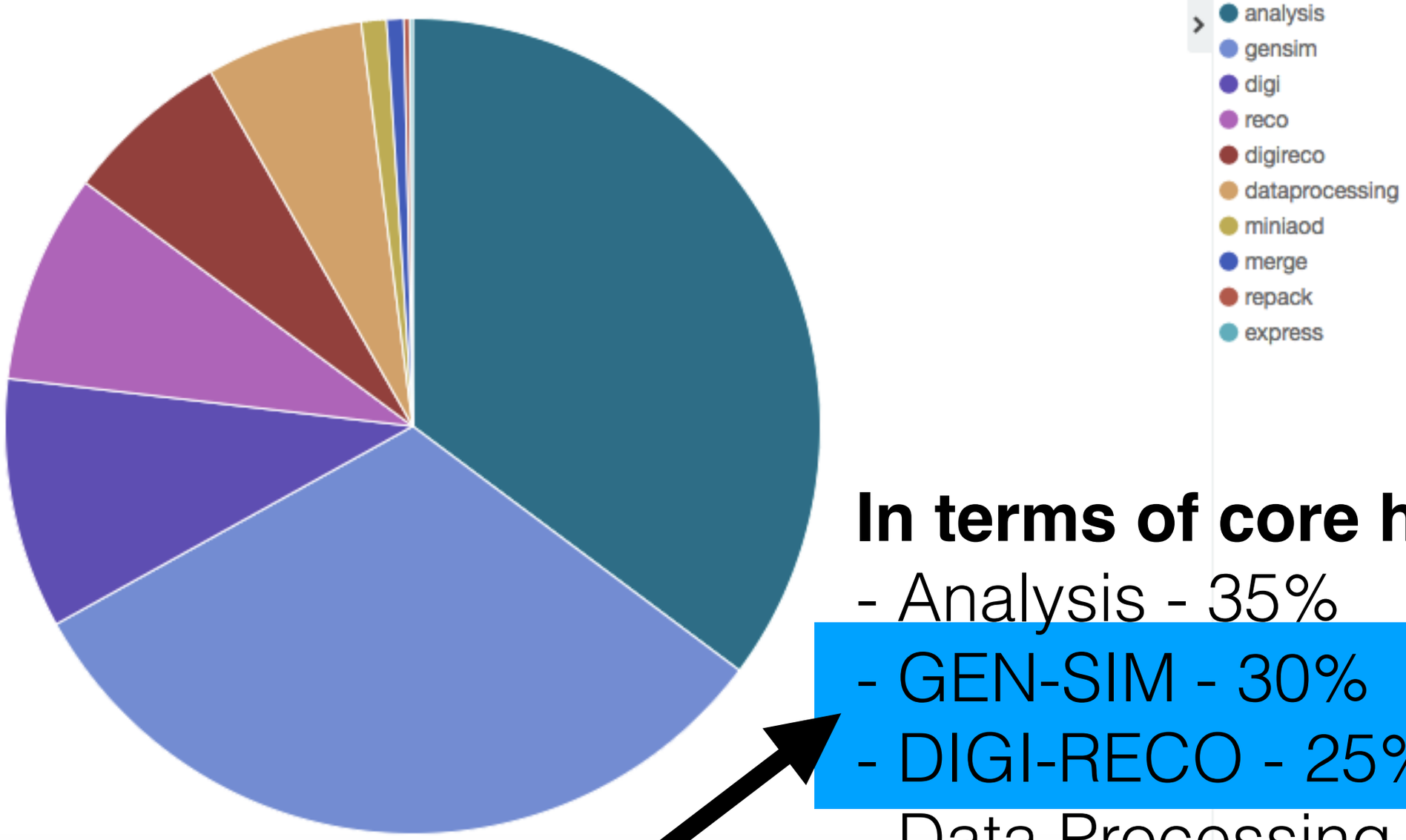
Other Job I/O

- Let’s set aside the worst cases from GEN - causes problem everywhere.
- What does the job I/O look like?
  - Each running job has:
    - 0 (GEN-SIM), 1 (RECO), or (#cores)+1 (DIGI) input files.
    - One or more output files (typically no more than 4).
  - Each job has a working directory consisting of O(100) config + Python files, stdout, err, etc.

Job Output

- Overall, output is typically modest enough we don't discuss it - O(100MB) per core hour [O(30KB/s) per core].
    - Output file goes to local disk and transferred to shared storage at the end of the job.
    - If the job's output file is non-transient and below 2GB, we will run a separate "merge job" to combine it with other output files in the same dataset.
        - Most jobs run for 8 hours on 8 (Xeon) cores; in 2-3 year timeframe, we expect this to double.
        - Around 2020, I hope we hit an era where most jobs output >2GB files and merge jobs become less frequent.

Global CPU Usage Breakdown - 2017



In terms of core hours:

- Analysis - 35%
- GEN-SIM - 30%
- DIGI-RECO - 25%
- Data Processing - 7%

End-to-end simulation is GEN-SIM + DIGI-RECO

Looking to the HL-LHC

- The LHC community is preparing for a major upgrade (“High Luminosity LHC”, or “HL-LHC”) of the accelerator and detectors, producing data in 2026.
- Seen as a significant computing challenge:
  - Higher luminosity causes increased event size.
  - 5-10X more events recorded by detector.
  - The RECO step CPU usage **increases quadratically** with luminosity.
  - SIM & DIGI CPU usage **increases linearly**.
  - GEN needs for CMS are completely not understood.
- In the HL-LHC era, we foresee RECO CPU costs dominating the work we would have for HPC machines.
  - Currently modeling suggests the overall IO profile would remain roughly the same.

What’s in a file?

- Each file contains an collection of events (stored in a ROOT TTree):
  - Each event has a number of associated objects as part of its description. These objects are in named collections and represent some data relevant to the physics description.
  - An event is typically described by a few hundred objects, containing a few thousand attributes.
- Each file contains “run” and “luminosity” information, also in ROOT TTrees.
  - The luminosity data describes statistics / info aggregated over a number of the events in the tree. Smallest unit of processing & tracking in the CMS computing model.
  - Run data describes statistics / info aggregated across a number of lumis.
  - “Run” and “lumi” terminology and semantics are descriptions from the LHC machine conditions. For simulation, these groupings are more arbitrary.
- Modest amount of additional metadata describing full processing history and provenance of the data in the file.
  - Given any CMS-produced file used for an analysis, we can determine the relevant software configurations of all the steps that produced that file.

What’s in an object?

- Physics objects serialized in the file can be arbitrary C++ objects (caveat - we do have some internal rules on this).
    - A powerful concept that is quite accessible to developers of all skill levels. Enables physicists to think about the physics objects.
    - Senior developers (or professional software engineers) are often more circumspect - *quite aware logical C++ objects may not be performant*.
        - Particularly problematic / expensive objects often go to experts for cleansing.
- Data rarely looks tabular:
    - E.g., each event may contain a varying number of particles. Each particle may contain a number of tasks.
    - Object and event size varies — can be by 20% or 200%, depending on the data tier.

What are our file format “needs”?

- Flexible compression options / efficient use disk space.
- Allows events to have variable amounts of data for each attribute / object.
- Includes (or allows us to build) interfaces for:
  - Primitives, lists, union, records.
  - (Nullable) references between objects.
  - Schema evolution.
  - Data formats produced by non-experts.
- Self-describing schema and files.
- Ability to efficiently read a sparse subset of the events or objects.
- Efficient reads over high-latency / remote connections.

Open I/O Challenges

- Reduce on-disk data sizes:
  - We manage 270PB of data. A 10% decrease in data size is a significant cost savings.
- Thorny nest of tradeoffs between CPU, memory, and disk budgets.
  - **Example**: data size could be decreased by giving more memory to compression algorithm. However, we are often at the limits of our memory-per-core budget.
  - **Example**: we use LZMA for the majority of our data. Despite being very CPU-expensive per byte, its CPU costs are acceptable in comparison to the cost of our physics algorithms.
  - We have perhaps gotten all the mileage possible out of “simple tunables” here.
- Looking at better ways to serialize our more complex data formats and improved compression techniques for structured data.

Open I/O Challenges

- Increase write parallelism:
  - Only one thread can read or write to a given ROOT file at a time.
  - As compression tends to be 10x more expensive than decompression, our current application parallelism is limited on the write side.
  - R&D is ongoing to have multiple in-memory output files and merge them in memory when the physics application has an idle thread.
    - All happens internal to the process - just more memory usage.
    - The scratch filesystem still sees one output file from a single thread.

Preferred HPC Vision

- Here's what I'd like to see:
  - CMS continues the Good Fight against Amdahl's law, reducing serial portions / lock contention until a single host can be effectively utilized by a single executable.
  - CMS can steer large workflows to large HPC allocations.
  - Jobs for a single workflow are bundled to a large batch job; each node sequentially executes N "payload jobs" (N=2? 3?).
  - Input data is read in bulk from a shared filesystem, processed by the payload job, writing to a non-global filesystem. After successful execution, output files are copied to the global file system.
  - After site batch job finishes, data on global file system copied out.

Can we make this work?

Questions?