Benchmarking WLCG resources using HEP experiment workloads

Andrea Valassi (CERN IT-SC)
On behalf of the HEPiX CPU Benchmarking WG

WOSSL Workshop, 23 July 2020
https://indico.in2p3.fr/event/21698

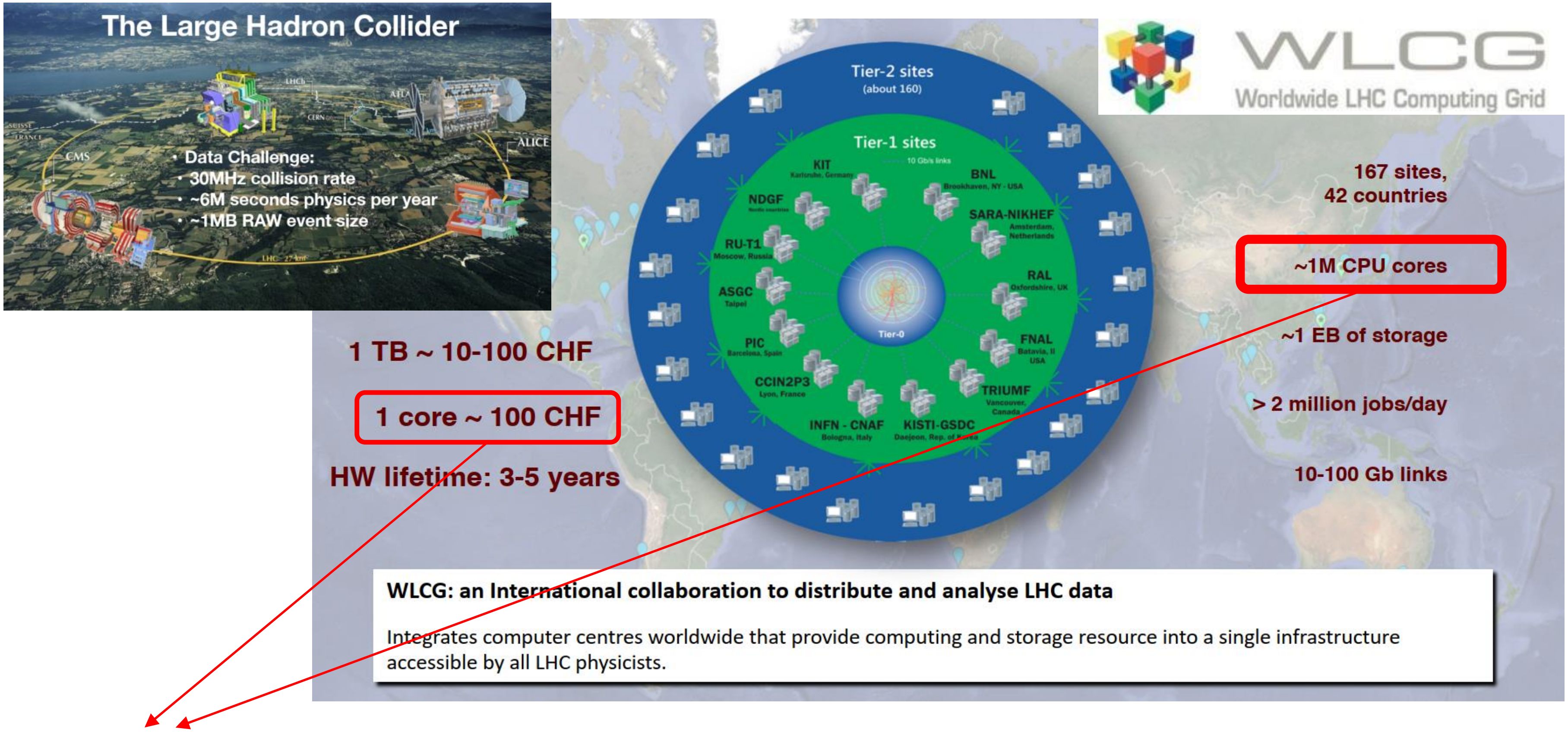
Outline

- Overview: CPU benchmarking in WLCG
  - Why benchmarking?
  - Current approach (HEP-SPEC06) and its limitations

- New approach: benchmarking using HEP experiment workloads
  - Overview, implementation, status
  - Applicability to HPCs and GPUs

- Conclusions

WLCG (Worldwide LHC Computing Grid): a varied computing landscape



S. Campana, ESPP, Grenada, May 2019

- *"CPU cores" are not all equivalent to one another* (sites are managed independently):
  - Some CPU cores are able to do more "work" than others per unit time (throughput)
  - Some CPU cores are more expensive than others

Why benchmarking CPU resources in WLCG?

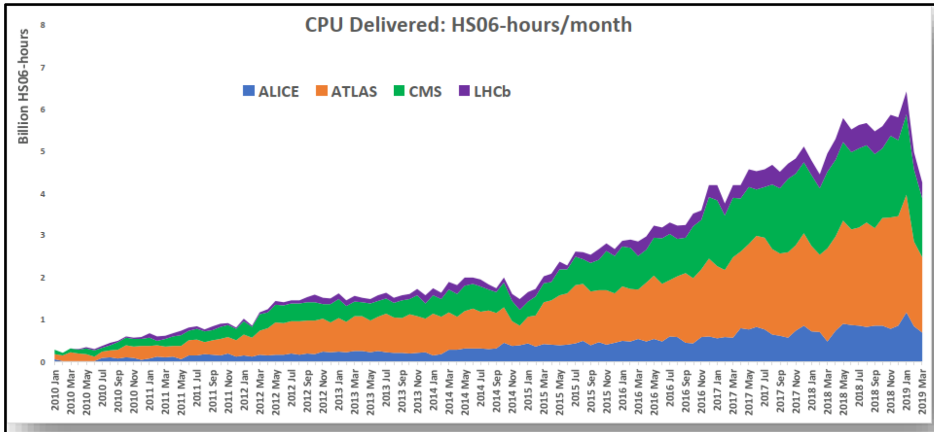
Two main use cases for WLCG:

- ***Accounting***
  - Experiments *request "X" CPU* resources to do their computing for one year
  - Funding agencies and sites *provision "X" CPU* resources to the experiments
  - Resource review boards *compare the "X" used to the "X" requested*
- ***Procurement***
  - Each site buys the CPU resources providing the *best "X" per CHF/EUR/…*

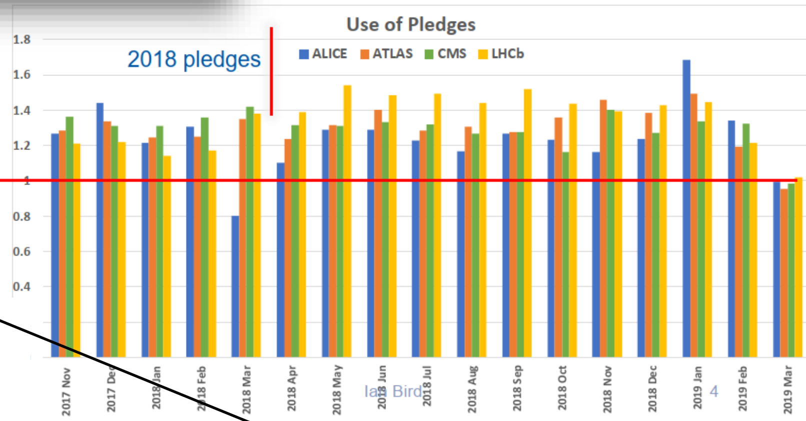
In addition:

- **Scheduling**
- **Software optimizations**
  - *NB: in this talk "benchmarking" refers to computing resources, not to software*

WLCG accounting: current benchmark is *HEP-SPEC06 (HS06)*



CPU Delivered



New peak: ~270 M HS06-days/month
~ 860 k cores continuous

Ian Bird, April 2019, Computing Resources Review Board

Very approximate rule of thumb: 10 HS06 per core (9M HS06 is ~0.9M cores)

HS06 is derived from SPEC CPU2006®

- Standard Performance Evaluation Corporation: industry standard since 1988
  - (Since this is an OSS workshop: note that SPEC CPU2006 is NOT open source software)
- *Real applications (from non-HEP domains), not a synthetic or kernel benchmark*
- After evaluating several subsets of SPEC CPU2006, chose the “all_cpp” subset
  - Seven C++ benchmarks (recompiled for HEP) – HS06 score is their geometric mean
  - Execution time: O(4h)

Bmk	Int vs Float	Description
444.namd	CF	92224 atom simulation of apolipoprotein A-I
447.dealII	CF	Numerical Solution of Partial Differential Equations using the Adaptive Finite Element Method
450.soplex	CF	Solves a linear program using the Simplex algorithm
453.povray	CF	A ray-tracer. Ray-tracing is a rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world
471.omnetpp	CINT	Discrete event simulation of a large Ethernet network.
473.astar	CINT	Derived from a portable 2D path-finding library that is used in game's AI
483.xalancbmk	CINT	XSLT processor for transforming XML documents into HTML, text, or other XML document types

4 Floating Point benchmarks

3 Integer benchmarks

HS06 was chosen because (in 2009) it seemed representative of HEP workloads

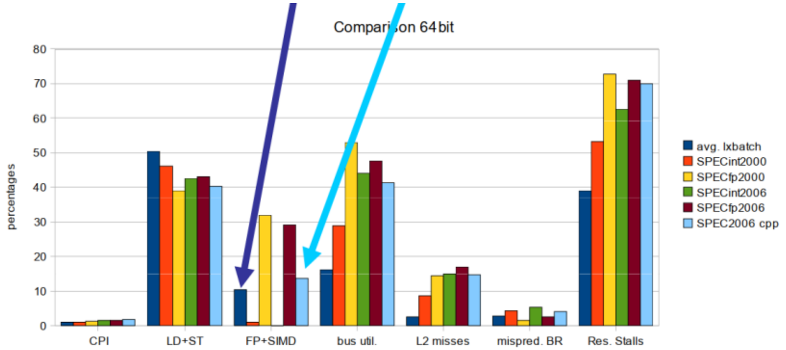
- *HS06 showed good correlation to the **throughputs** of HEP workloads*
  - Throughput (events per second) is the most relevant metric for HEP processing

Correlation	Generation	Simulation	Reconstruction	Total
Atlas	0.9969	0.9963	0.9960	0.9968
Alice pp MinBias	0.9994		0.9832	0.9988
Alice PbPb	0.9984		0.9880	0.9996
LhcB	0.9987			
CMS HiggsZZ	0.9982		0.9987	0.9983
CMS MinBias	0.9982		0.9974	0.9974
CMS QCD 80 120	0.9988		0.9987	0.9988
CMS Single Electron	0.9987		0.9942	0.9981
CMS Single MuMinus	0.9986		0.9926	0.9970
CMS Single PiMinus	0.9955		0.9693	0.9955
CMS TTbar	0.9985		0.9589	0.9987

Correlation of HEP-SPEC06 with several kinds of applications and different experiments

M. Michelotto et al. (2010)
J. Phys. Conf. Ser. 219 052009

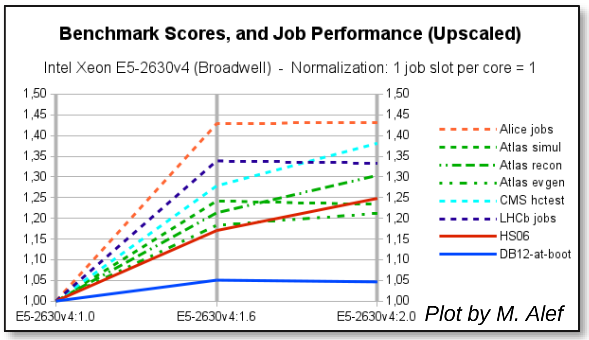
Similar FLOATING POINT fraction (~10%) for ***HEP workloads on lxbatch*** *and* ***SPEC2006 "all_cpp"***
(lower for SPEC INT, higher for SPEC FP)



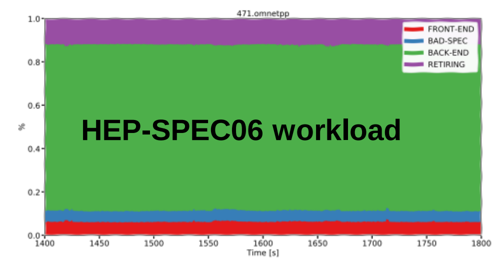
- *HS06 showed similar **CPU usage patterns** to those of HEP workloads*
  - Hardware performance counters (FP+SIMD, Load+Store, Mispredicted Branch)
    - Analysis using perfmon on lxbatch (compute nodes of LHC experiments at CERN)

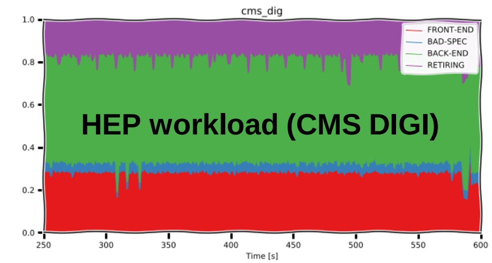
After 10 years, HS06 does not describe HEP workloads well enough any longer

- *HS06 score shows poor correlation to the **throughputs** of HEP workloads*
  - Issue reported by ALICE, LHCb – somewhat better agreement for ATLAS, CMS
    - Use of 32-bit benchmark for 64-bit applications explains part of the discrepancy



D. Giordano, CHEP2018, Sofia, July 2018





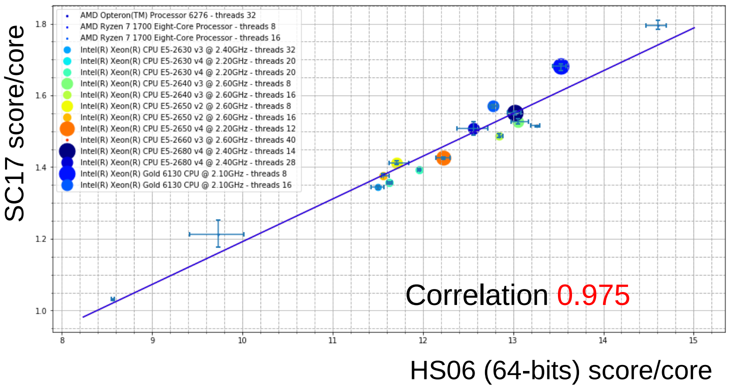
D. Giordano, WLCG GDB, May 2019

- *HS06 shows different **CPU usage patterns** from those of HEP workloads*
  - Hardware performance counters (front-end, back-end, retiring, bad speculation)
    - Analysis using the Trident toolkit, similar to that done with perfmon in the past

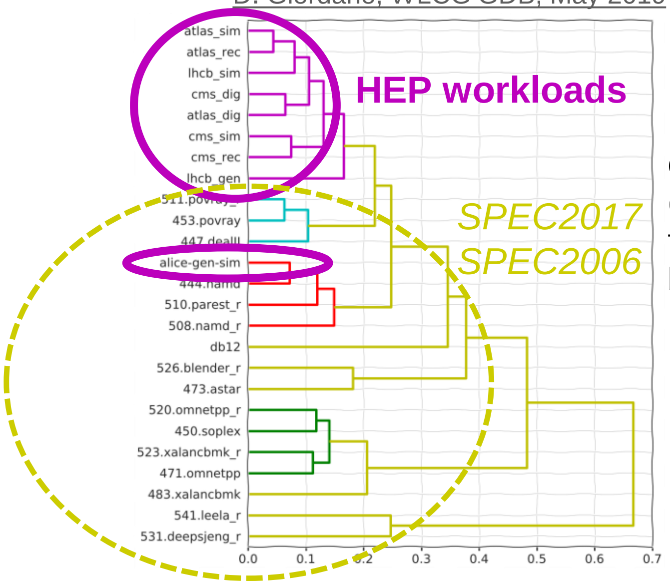
SPEC CPU 2017 has been evaluated, but it has the same issues as HS06

- *SC17 score shows <u>poor correlation to the **throughputs** of HEP workloads</u>*
  - Because it is highly correlated, i.e. essentially equivalent, to the HS06 score

D. Giordano, CHEP2018, Sofia, July 2018



D. Giordano, WLCG GDB, May 2019



Dendrogram of workload similarity (in 3-D space of front-end, back-end, bad-speculation %)

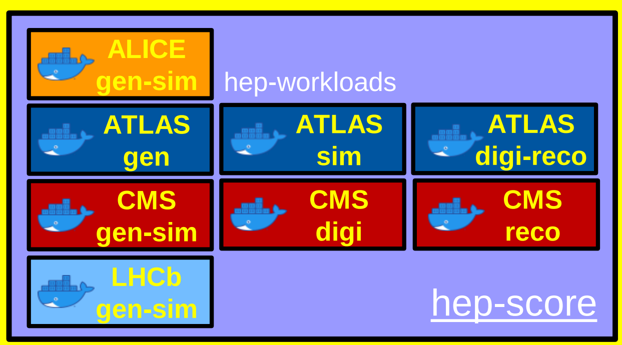
- *SC17 shows <u>different **CPU usage patterns** from those of HEP workloads</u>*
  - Whereas it has very similar CPU usage patterns to those of HS06

We have developed an alternative solution: benchmarking CPUs using HEP workloads

- Why did we choose HS06? Because it seemed representative of HEP WLs!
  - Score correlated to HEP WL throughput, CPU usage similar to HEP WLs

- *By construction,* using HEP workloads directly is guaranteed to give
  - *A score with <u>high correlation to the throughputs of HEP workloads</u>*
  - *A <u>similar CPU usage pattern to that of HEP workloads</u>*

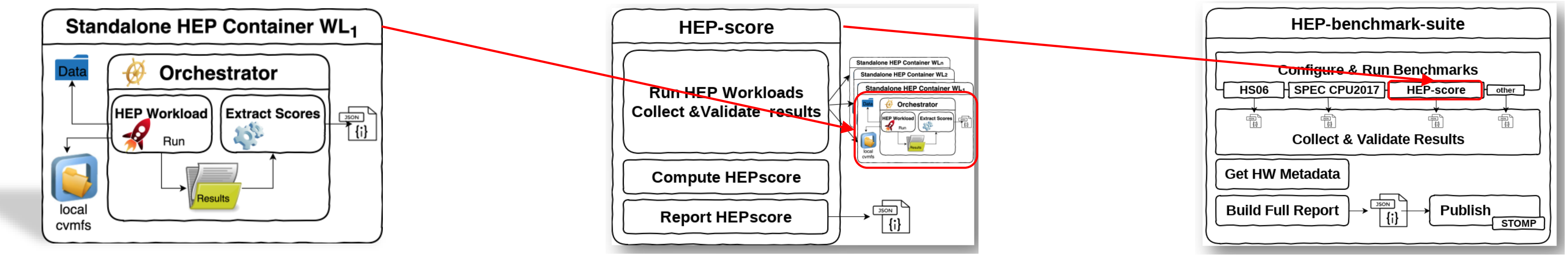
It seems obvious… why did we not do this before?



What allows us to do this now is the availability of *container technology!*

"HEP benchmarks" project overview

- Three* main repositories under https://gitlab.cern.ch/hep-benchmarks:



hep-workloads
- *runs a single HEP workload*
- includes common and WL-specific infrastructure to build WL containers
- most active package so far

hep-score
- *runs several HEP workloads*
- average/combine individual scores to give *a single benchmark number*

hep-benchmark-suite
- *runs several benchmarks* (hep-score, HS06 and others)
- collects results in a database

*Plus more recent additions (one for HS06, one for GPU WLs, etc.)

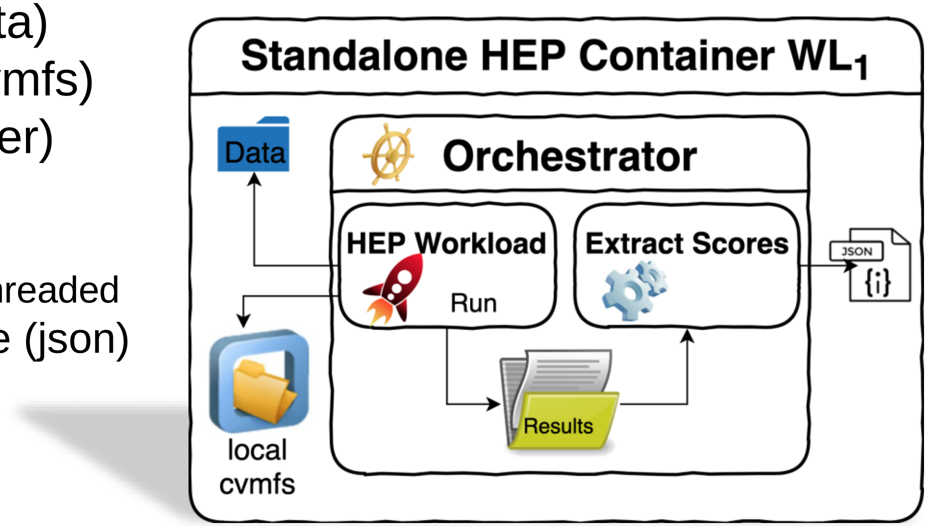
- Project organization (this is an activity of the HEPIX benchmarking WG)
  - Team: core development and infrastructure, testing, experiment experts
  - Track work progress via Jira Project and Twiki

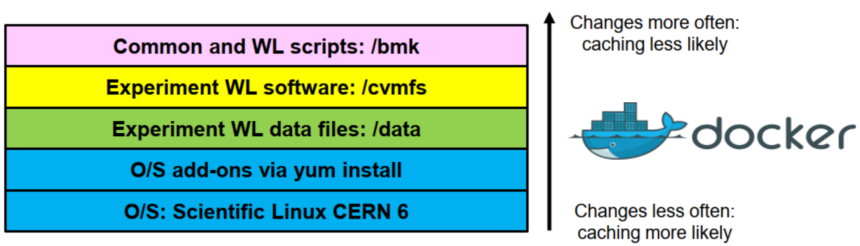
(Note: the infrastructure we developed is in the process of being licensed as GPLv3)

From HEP reference workloads to containers: the hep-workloads project

- Main requirements:
  - Self-contained (no network), easy to use, fast/small, stable/reproducible…

- *One workload ↔ One standalone Docker container (with all dependencies)*
  - Operating system
  - Input data (event and conditions data)
  - Experiment-specific software (on cvmfs)
  - Orchestrator script (benchmark driver)
    - Sets environment
    - Runs application (many copies)
      - Each copy may be multi-process/threaded
    - Parses output to generate WL score (json)

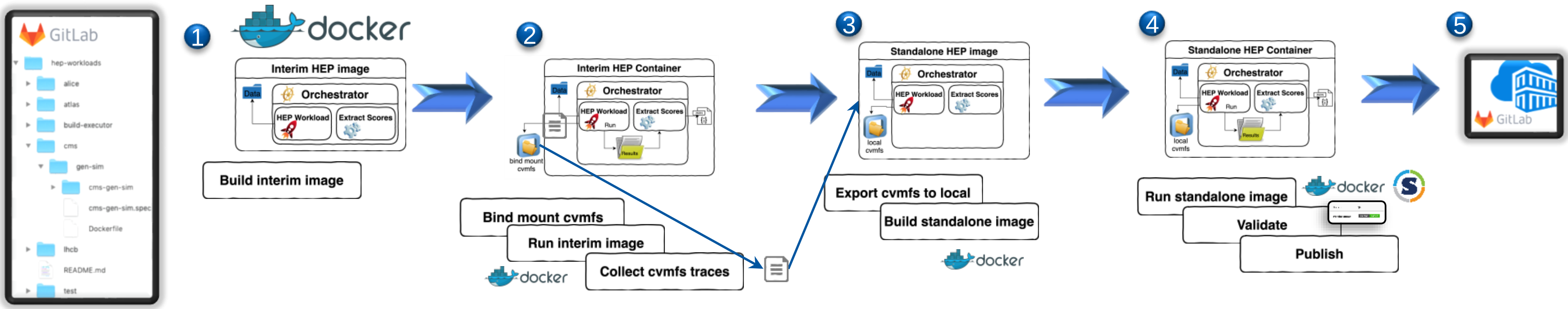




Docker WL images are made up of layers

Workload containers are built in the hep-workloads gitlab CI

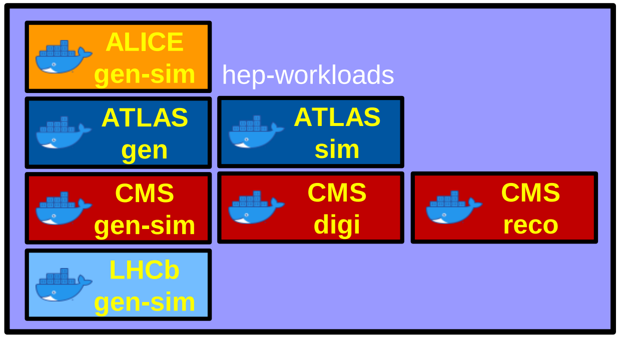
- *Main idea: experiment software is on /cvmfs, discover what is needed in a dry run*
- *Enabling technology: cvmfs tracing mechanism*



- Starting from gitlab repo containing only CI and WL orchestrator scripts:
  1. Build interim Docker image: /cvmfs is the standard network-connected service
  2. Run WL in interim Docker image: generate cvmfs traces listing which files were accessed
  3. Build standalone Docker image: /cvmfs is a local folder, copy all relevant files
  4. Test standalone Docker image (both in Docker and Singularity)
  5. Push standalone Docker to gitlab registry

The hep-workloads container registry: available images

- The following images can currently be downloaded and tested:
  - *GEN and SIM workloads are available for all four LHC experiments*
  - *DIGI and RECO workloads are available for CMS* (work in progress for ATLAS)
  - Available from https://gitlab.cern.ch/hep-benchmarks/hep-workloads/container_registry



- *Executing one specific workload benchmark is a one-liner:*
  - Example for CMS DIGI, both via Docker and Singularity:

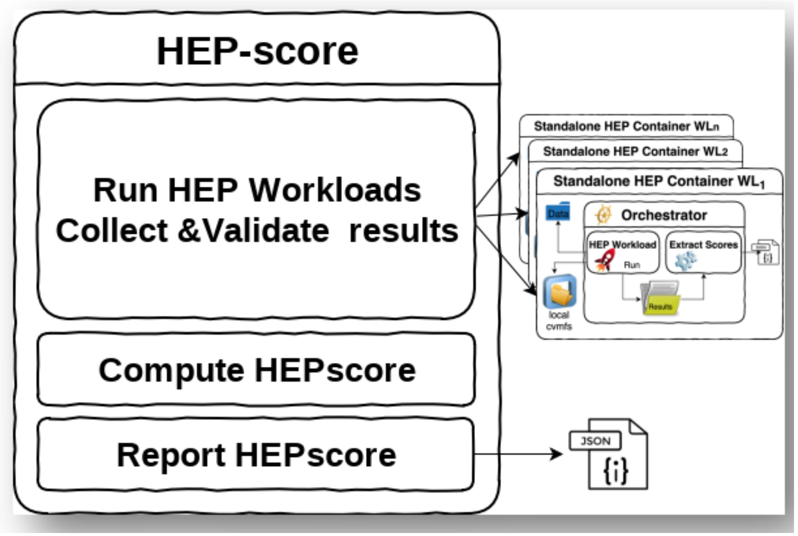
```
IMAGE=gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/cms-digi-bmk:latest
docker run -v /tmp/results:/results $IMAGE
singularity run -B /tmp/results:/results docker://$IMAGE
```

  - A json summary and detailed logs are then found in /tmp/results on the host system

The hep-score benchmark: many degrees of freedom, one number

- Each HEP workload stresses different components of a computer system
  - Some are I/O intensive, others not; some are vectorized, others not…

- Using a single metric to characterize performance is difficult (and dangerous)
  - But this is what we often need for accounting and/or procurement
  - *Presently, HEP score is the geometric mean of a small subset of HEP workloads*
    - *But the json output also keeps a record of each individual WL score independently!*

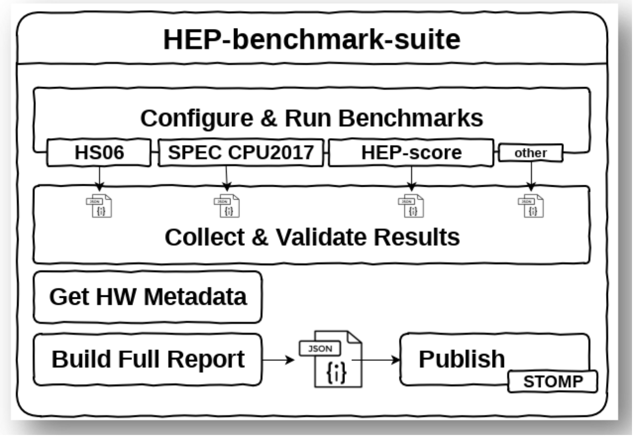


HEPscore19 prototype
(use the most stable and best understood workloads):

atlas-gen-bmk v1.1
atlas-sim-bmk v1.0
cms-gen-sim-bmk v1.0
cms-digi-bmk v1.0
cms-reco-bmk v1.0
lhcb-gen-sim-bmk v0.12

The hep-benchmark-suite toolkit

- A single toolkit to coordinate execution and result collection for several benchmarks
  - Example: execute HS06, SPEC2017 and HEP-SCORE on a set of reference machines
  - Collect results of all benchmarks in a global JSON document and upload it to a database



- This has been used for a systematic analysis of the new benchmark suite:
  - Collected a set of reference machines (like the “lxbench” cluster used for HS06 studies)
  - Systematically studied the correlation of individual HEP WL’s to one another and to HS06

Comparison between HEPscore and HS06

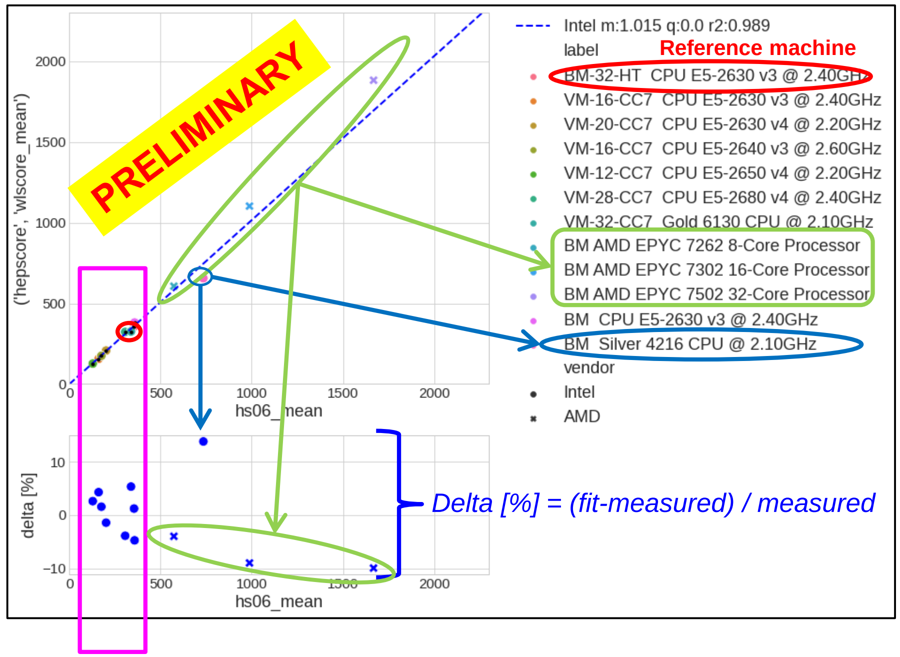
- Use a reference machine to set the normalization, i.e. the absolute scale: HEPscore (ref) = HS06 (ref) = 355

- On older CPUs: fit HEPscore vs HS06 (Intel Haswell, Broadwell, Skylake): good correlation, agreement within 5%

- On newer CPUs (AMD EPYC Rome and Intel Cascade Lake Silver):
  - With respect to HEPscore (i.e. real HEP WLs!):
  - HS06 underestimates AMD EPYC by ~10%
  - HS06 overestimates Intel Silver by ~13%

  (NB: these numbers only reflect how well/badly our applications exploit these specific CPUs)

HEPscore vs HS06 for some Intel and AMD CPU models @ CERN DC

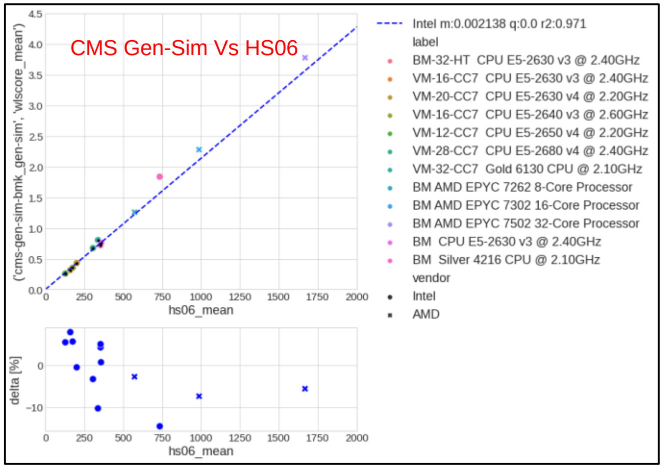


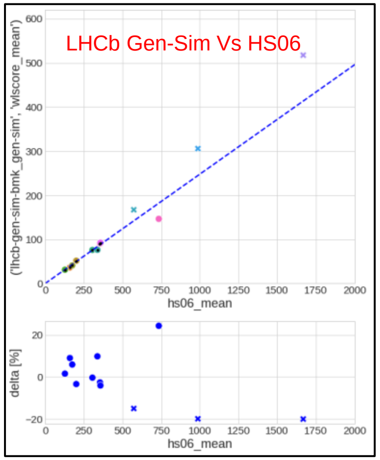
Fit region (only includes Intel Haswell, Broadwell, Skylake)

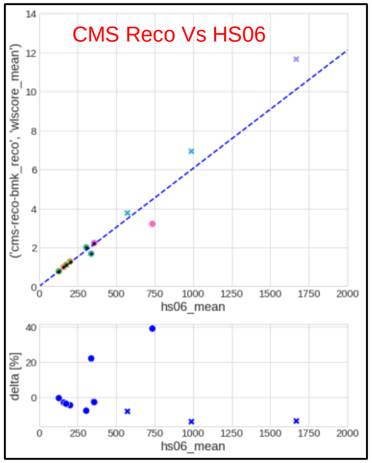
Comparison of individual HEP WLs to HS06

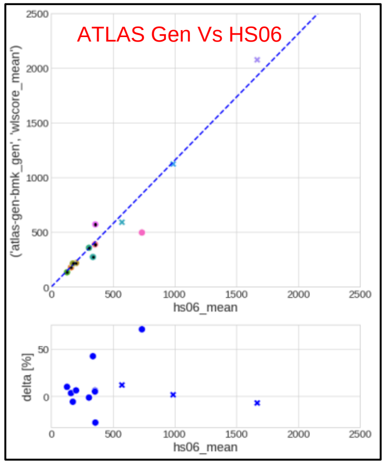
PRELIMINARY

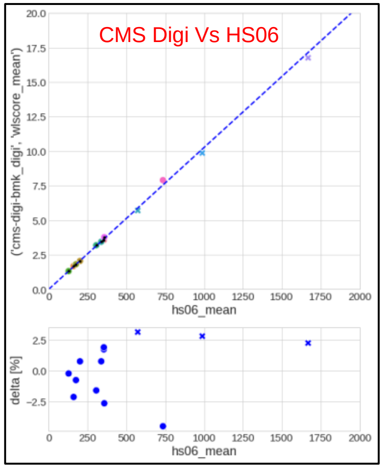
- By construction, HEPscore allows the study of performance for individual HEP WLs
  - –This is not possible with HS06
- Larger discrepancies are observed than for the average of all WLs (this is consistent with previous reports from the experiments)

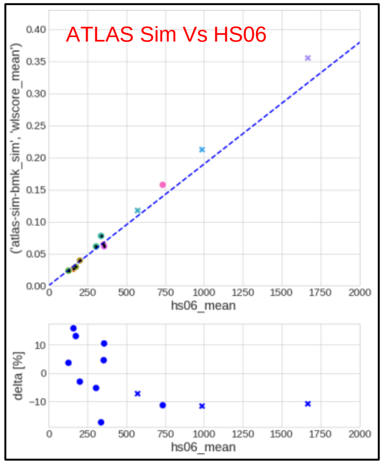












GPUs, HPCs and heterogeneous resources

- All of the work on hep-workloads described so far refers to x86 architectures
- WLCG computing is expected to go well beyond x86 in the medium term future
  - Non-x86 HPC supercomputers (ARM, Power9, GPUs…) will probably play a large role
- By and large, the software of the experiments is not yet production-ready for this
  - Porting and validating it (and having the people to do that) is one of the first priorities
  - But our new benchmarks must be ready in time to do the accounting for these resources!
- *Specifically: a HEP workload container involving GPUs is ready and is being tested*
  - CMS event reconstruction, with optional GPU offload of pixel tracking *(Patatrack)*
  - The container build approach described earlier applies also in this case
  - But defining a benchmark for heterogeneous systems (CPU+GPU) may be more tricky

Next steps: moving HEPscore to production

- General agreement in our community that HEPscore should replace HS06
- From a *technical* point of view, the infrastructure is essentially ready
- What we need now are *policy decisions* (on non-technical issues)
  - How to define the one benchmark (or use many benchmarks?)
    - E.g. weighted geometric mean of WLs: which WLs, which weights?
    - Negotiations needed with all relevant sites and experiments involved
  - Keep the benchmark stable or allow evolutions over time?
    - For comparison, HS06 was used unchanged for 10 years
  - A task force for this has been created by the WLCG Management Board

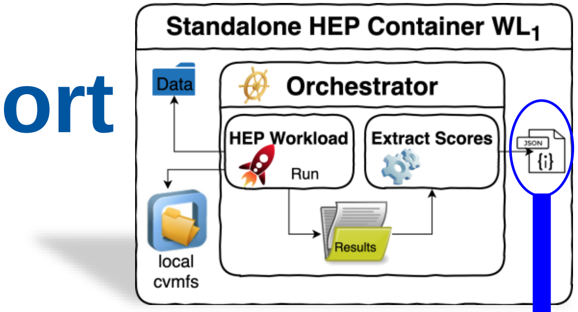
Conclusions

- After 10 years, HEP-SPEC06 no longer describes well enough HEP workloads
- Our solution: build a new benchmark directly from HEP workload throughputs
  - Enabling technologies: Docker containers and cvmfs tracing mechanism
- Technically, the new infrastructure for CPU benchmarking is essentially ready
  - Individual containers exist for the main workloads of all four LHC experiments
  - A HEPscore prototype exists to compute a benchmark as an average of specific WLs
  - The correlation (or lack thereof) with HS06 has been extensively studied for many CPUs
- To adopt HEPscore as a production CPU benchmark, policy decisions are needed
  - WLCG has created a task force to work towards this goal
- Work is in progress to extend this to GPUs and non-x86 CPU resources
  - Other specificities of running this tool at HPC centers are also being addressed
- The approach is reusable elsewhere and our solution is open source software
  - Non-LHC and non-HEP communities have already expressed their interest

For more information: *https://arxiv.org/abs/2004.01609*

Backup slides

The hep-workloads output report



JSON document with the essential information

- Configuration parameters
  - #copies, #threads, #events, status
- Benchmark score: **total node throughput**
  - **Events per wall second** (sum over all copies)
    - Or events per CPU second in some cases
  - Details for each application copy
    - Statistics: mean, median, max, min…
- Additional metrics for performance studies:
  - Memory and CPU utilization
- Workload metadata
  - Description, version, checksum


```
"report": {
  "wl-scores": {
    "gen-sim": 0.4438
  },
  "wl-stats": {
    "CPU_score": {
      "max": 0.0226,
      "score": 0.1123,
      "median": 0.0225,
      "avg": 0.0225,
      "min": 0.0222
    },
    "throughput_score": {
      "max": 0.0892,
      "score": 0.4438,
      "median": 0.089,
      "avg": 0.0888,
      "min": 0.088
    }
  },
  "log": "ok",
  "app": {
    "bmkdata_checksum": "e57b3ad19144b7e9574b97056fb35d11",
    "cvmfs_checksum": "b2ab0e3bd4ba1333ebfc7dc49a024536",
    "bmk_checksum": "fc73ae9f18c4ef90791f097cd31b45dc",
    "version": "v1.0",
    "description": "CMS GEN-SIM of ttbar events, based on CMSSW_10_2_9"
  },
  "threads_per_copy": 4,
  "copies": 5,
  "events_per_thread": 100
},
```

HEP software and computing evolves… so do HEP CPU benchmarks!

1980's
MIPS (M Instr Per Sec)
VUPS (VAX units)
CERN units

1990's – 2000's
SI2k (SPEC INT 2000)
INTEGER benchmarks
200 MB footprint

2009
HS06 (SPEC CPU 2006 all_cpp)
INTEGER + FP benchmarks
1 GB footprint
32-bit
x86 servers
single-threaded/process on multi-core

2019
2 GB footprint (or more)
64-bit
multi-threaded, multi-process
multi-core, many-core
vectorization (SSE, … AVX512)
x86 servers, HPCs
ARM, Power9, GPUs…?

- As time goes by, *WLCG computing is becoming more and more heterogeneous*
- One of the challenges is how to summarize performance using a single number
  - Unfortunately, this is needed at least for accounting purposes

Docker layers in hep-workloads images

- Docker container images are always made up of *layers*
  - – Translating Docker images to Singularity also keeps this layer structure unchanged
  - – From the bottom up, these layers can be *cached* until the first difference is found
- The hep-workloads CI builds these layers to make them as cacheable as possible
  - – The bottom layers contain what is expected to change least often
  - – The top layers may change more frequently (across different workloads or versions)
  - – Advantage in the CI: faster builds/tests, save storage space (both Docker and Singularity)
    - Advantage for users: faster tests, save storage space (if Docker and Singularity caches are set up)