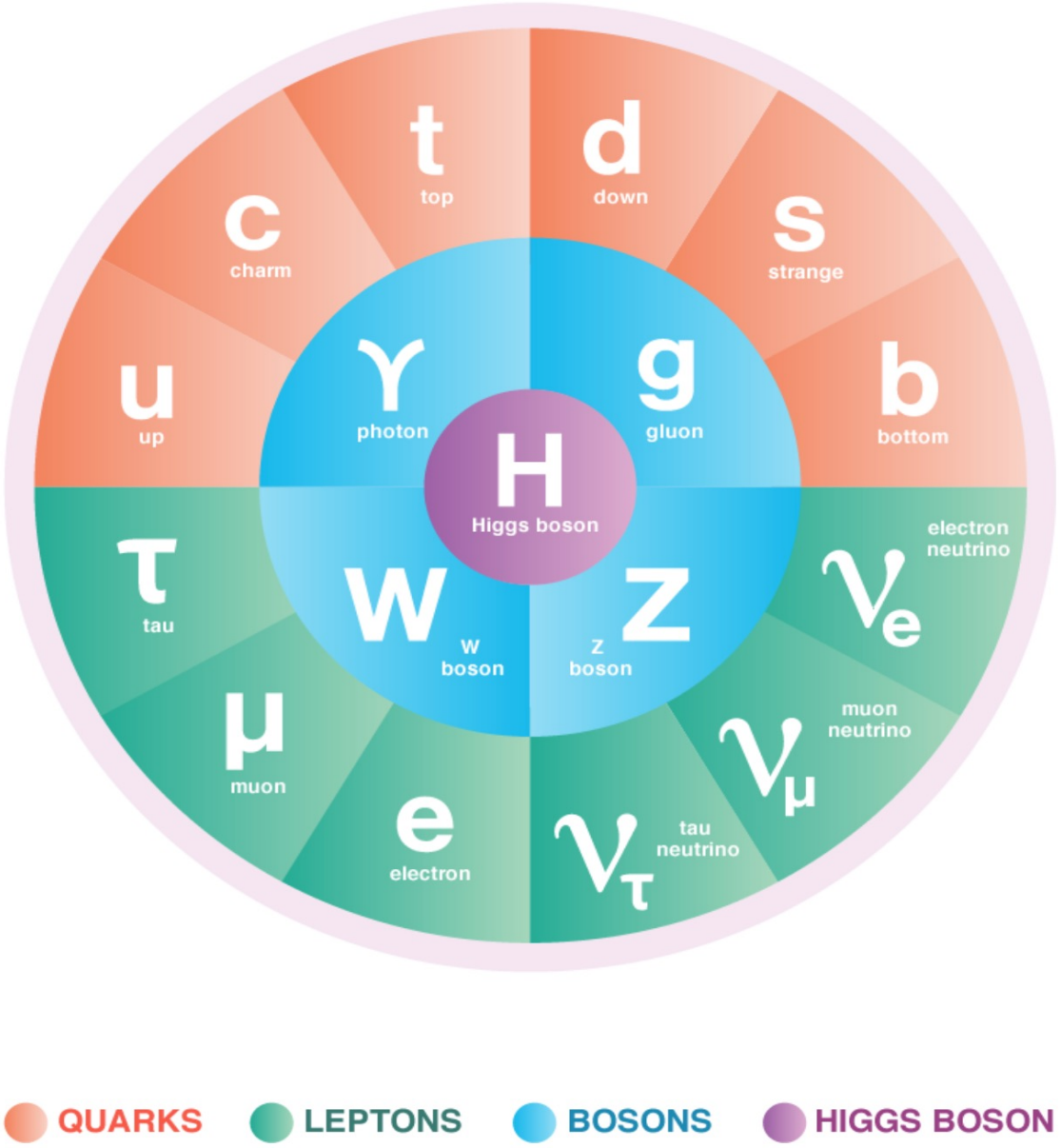
Achieving near native runtime performance and cross-platform performance portability for random number generation through SYCL interoperability

Vincent R. Pascuzzi[1] and Mehdi Goli[2]

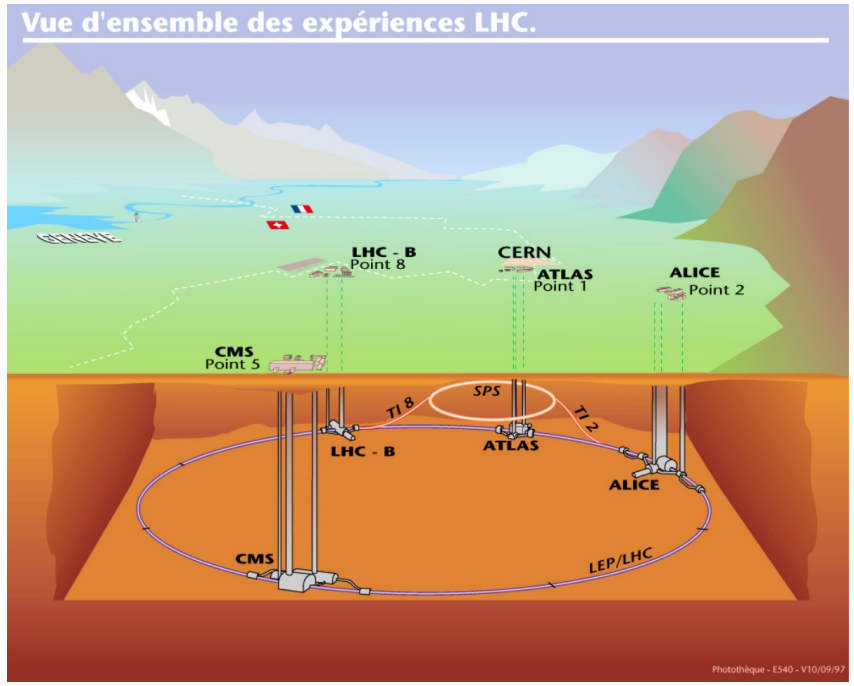
[1] Brookhaven National Laboratory (US)
[2] Codeplay Software Ltd. (UK)

November 14, 2021

Particle physics deals with the fundamental constituents of Nature and the forces through which they interact

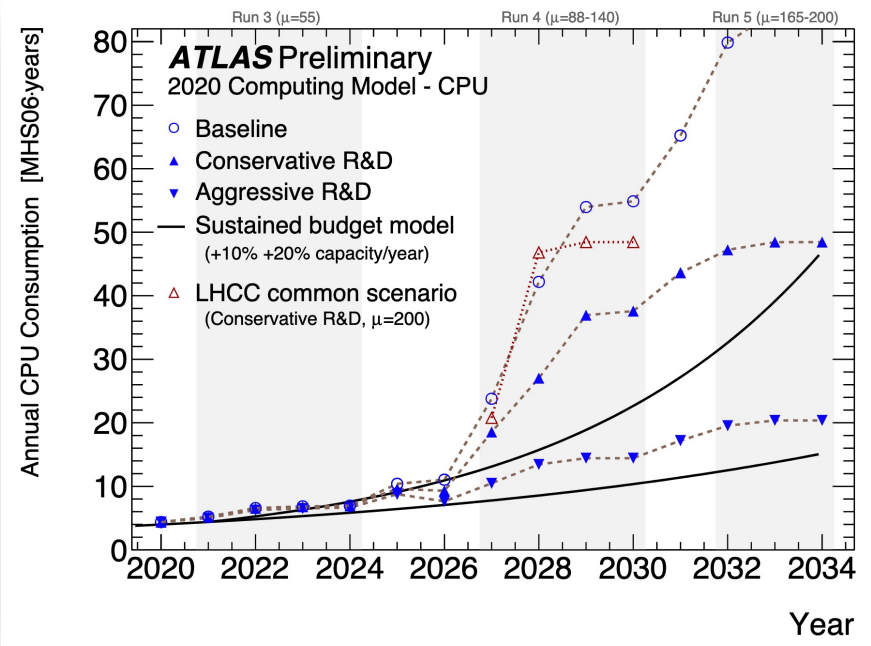


The Large Hadron Collider (LHC) is a superconducting particle accelerator at the European Organization for Nuclear Research, designed to collide protons at a center-of-mass energy of 14 TeV, at a rate of $4x10^6$ times per second



Run-2/3 data rate ~10 PB.

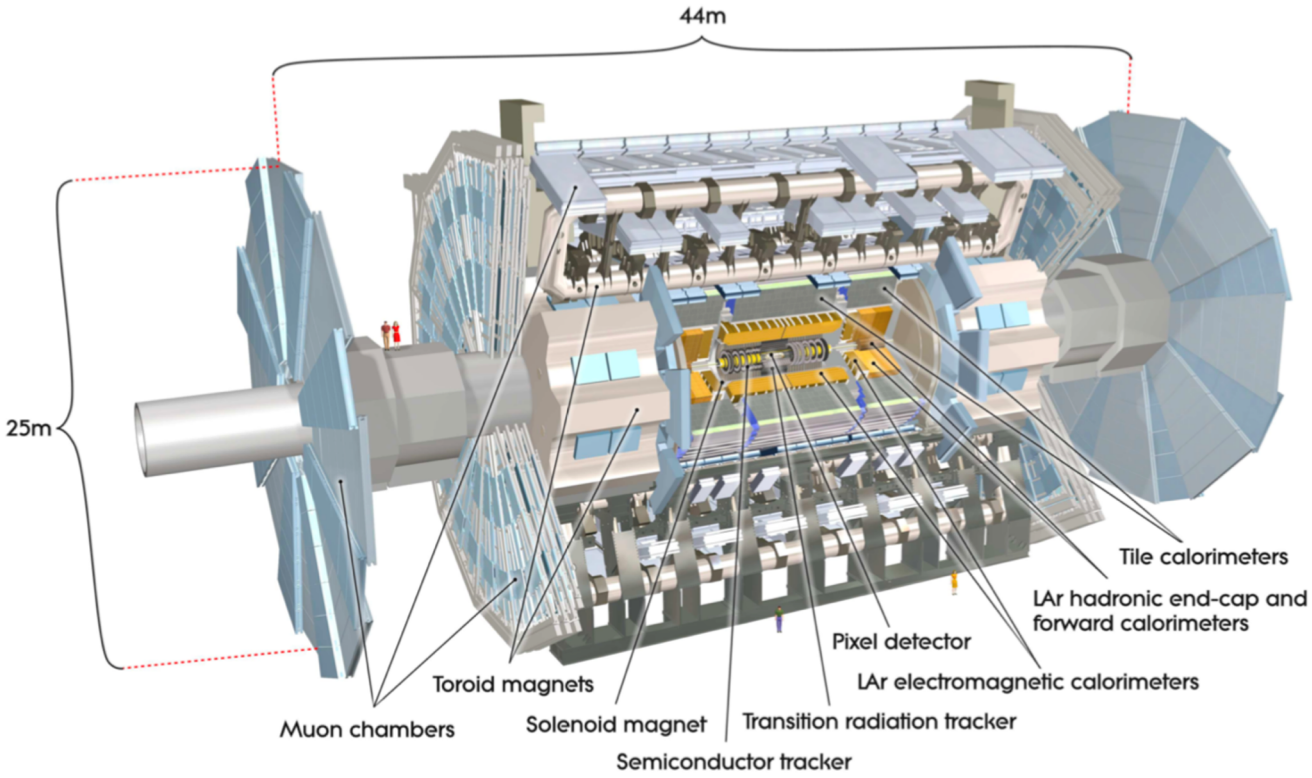
Run-3/4 can expect ~1 EB.



Brookhaven National Laboratory

The ATLAS detector is one of the general-purpose experiments stationed along the LHC ring, designed for particle physics research

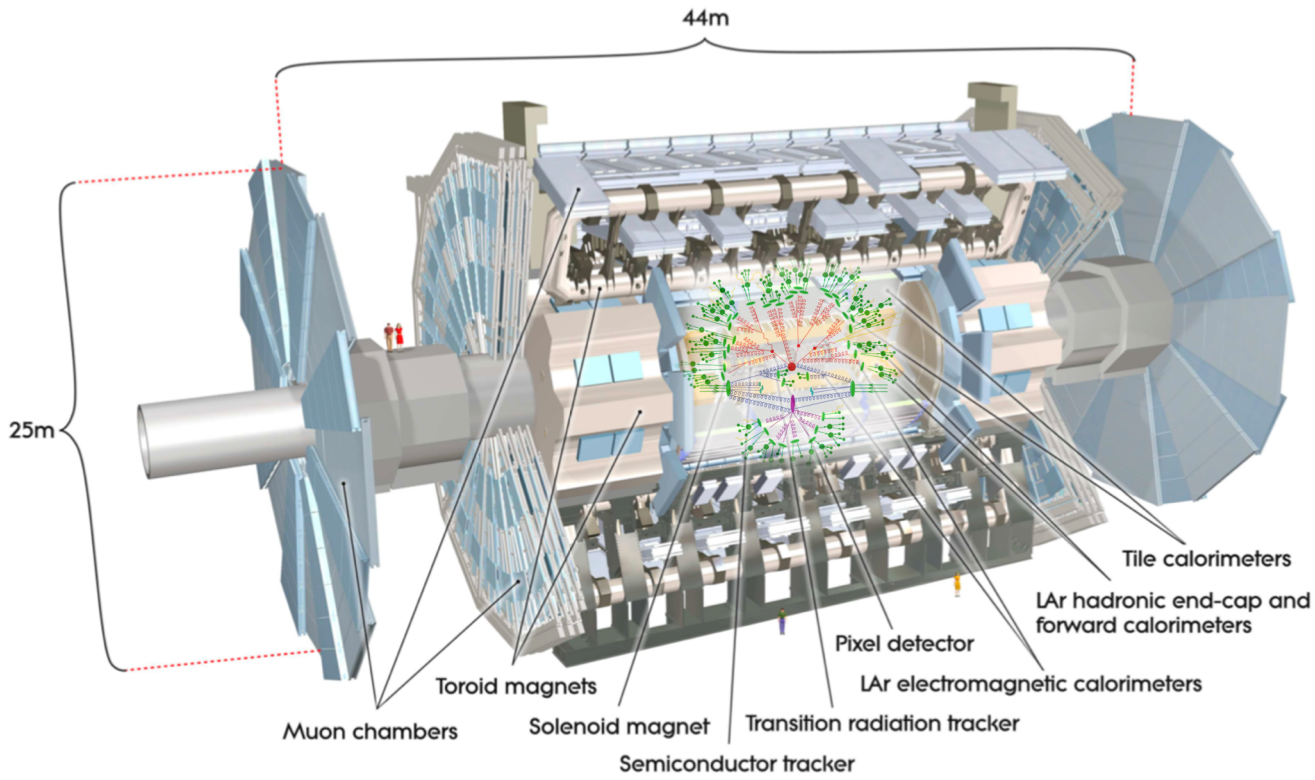
- Further test the Standard Model (SM)
- Discover the Higgs boson (or something like it)
- Search for Supersymmetry and other beyond-the-SM physics



Brookhaven National Laboratory

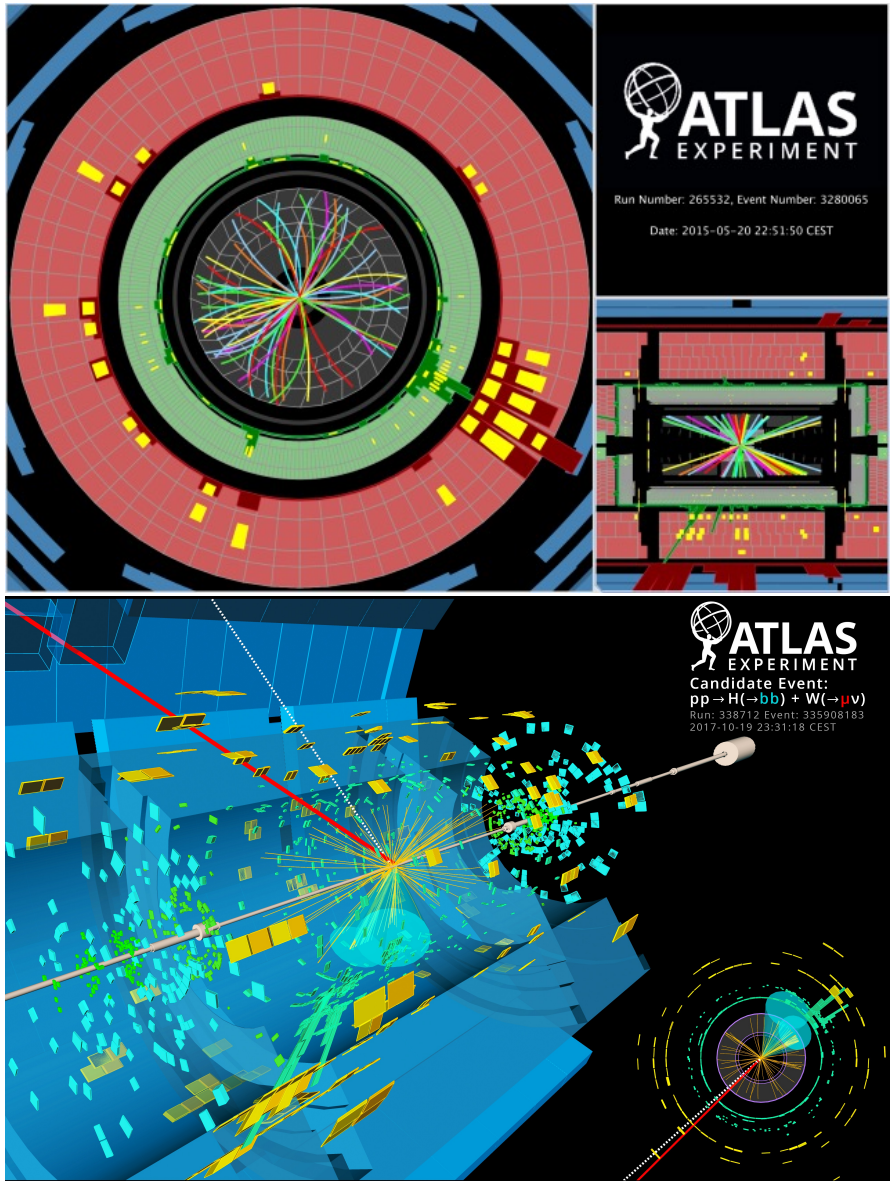
Due to the LHC's high instantaneous luminosity (number of collisions per second per unit area), tens of collisions occur every 25ns

- Filter out the 'uninteresting' collisions (events) from typically one interesting event
- Transfer raw data off-detector
- Reconstruct events offline



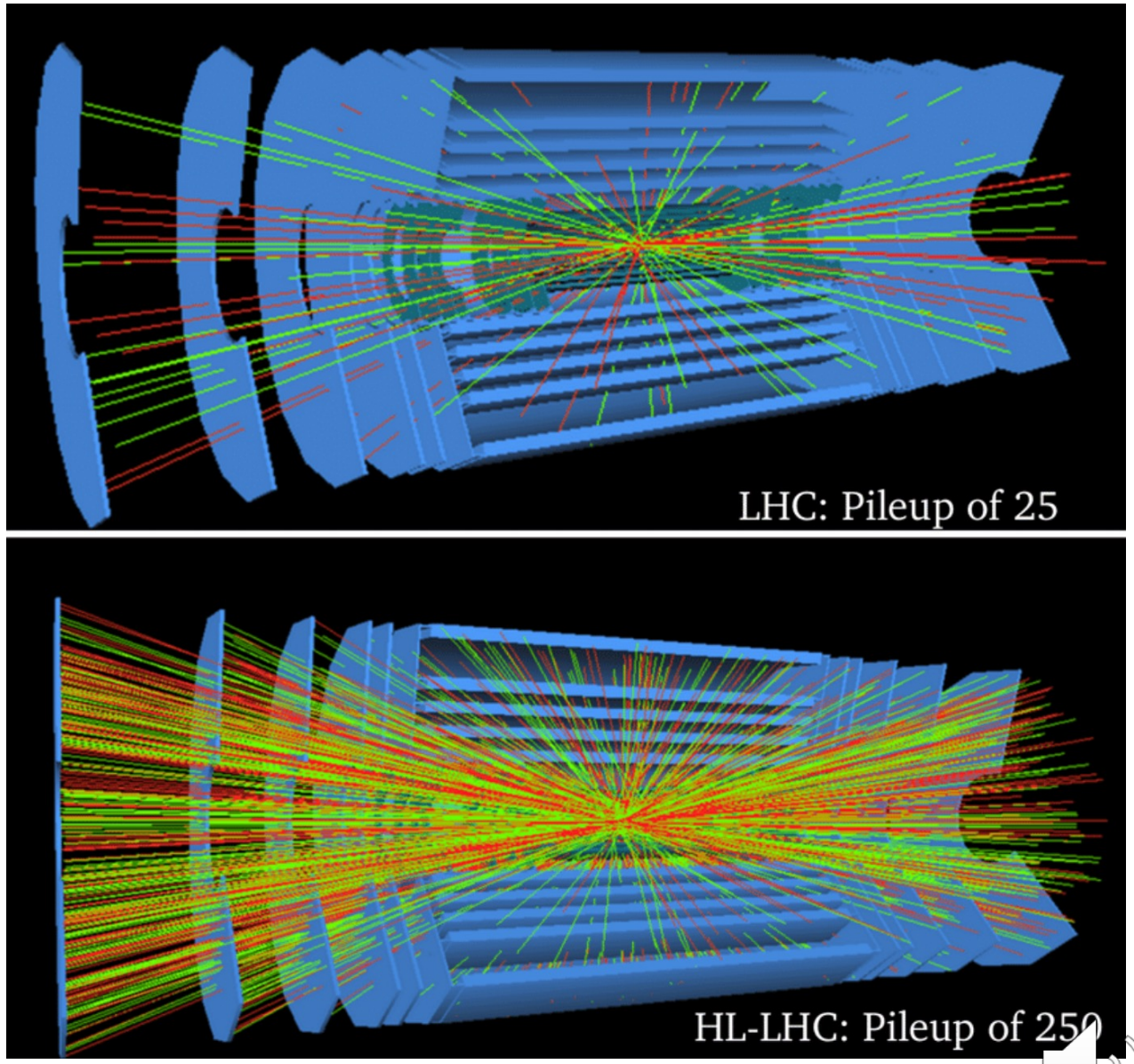
To validate our measurements, and to ensure the detector is functioning nominally, we need *a lot* of simulated Monte Carlo events

- Full-scale detector description contains ~10^6 volumes
- Geant4-based simulations of particles traversing the detector (particle material interactions, kinematics, *etc.*) can take ~minutes for a _single_ simulated event
- Will become less manageable after high-luminosity LHC (HL-LHC) upgrades are complete (~100s simultaneous collisions)



Brookhaven National Laboratory

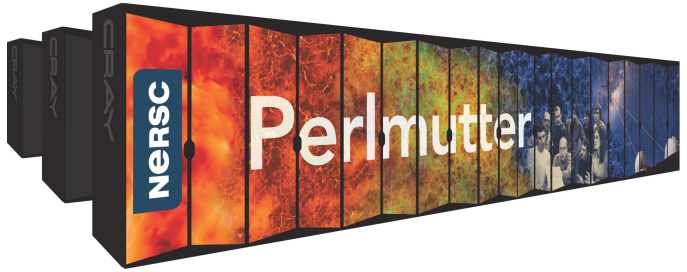
To continue successful physics programs, it is crucial ATLAS and other high-energy physics experiments utilize heterogeneous resources!



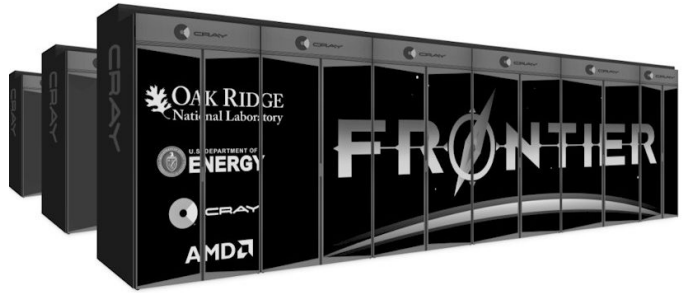
Caveat We are limited in developers (we are physicists) and there are numerous architectures and platforms.

- We cannot afford to support and maintain multiple codebases.
- We need to utilize leadership computing facilities.
- We need portability and achieve a fair level of performance.

National Energy Research Scientific Computing Center (NERSC), 2021
AMD CPU, NVIDIA GPU

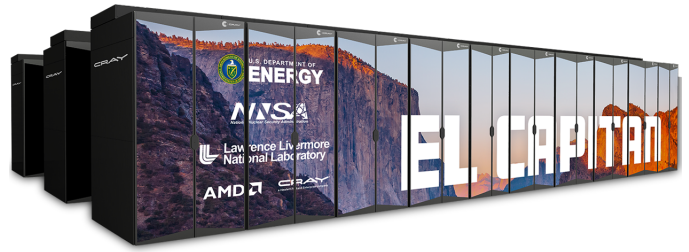


Oak Ridge National Laboratory (ORNL), 2021
AMD CPU, AMD GPU



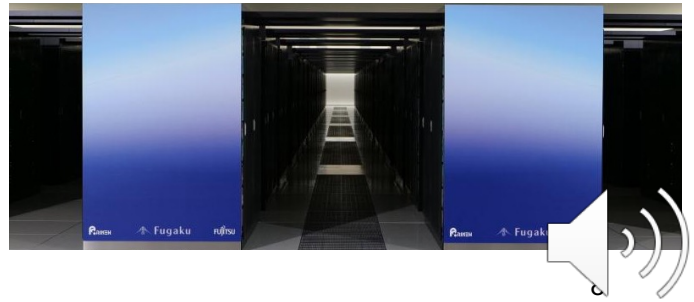
Argonne National Laboratory (ANL), 2022
Intel CPU, Intel GPU

Lawrence Livermore National Laboratory (LLNL), 2023
AMD CPU, AMD GPU



Swiss National Supercomputing Center (CSCS), 2023
NVIDIA/ARM CPU, NVIDIA GPU

Riken Center for Computational Science, 2021
ARM CPU



Brookhaven National Laboratory

HEP Center for Computational Excellence (CCE)

A Department of Energy High-Energy Physics program investigating:

- Performance portability
- I/O
- Complex workflows
- Event generators*

Portable Parallelization Strategies (PPS) effort focuses on performance and portability solutions for current and future HEP software

- Select among the participating experiments a number of x86-based ‘testbeds’ and rewrite the codes in various programming models

HEP-CCE Experiments

* Event generator software is written and maintained by theorists.

Brookhaven National Laboratory

Performance portability

"An application is performance portable if it achieves a consistent ratio of the actual time to solution to either the best-known or the theoretical best time to solution on each platform with minimal platform specific code required." [1]

Performance

- It runs: {Yes, No}
- It runs efficiently with respect to some baseline

Portability

- Can execute on multiple systems
- Adaptable to varying architectures and platforms

Productivity

- SLoC, maintainability, sustainability
- Port/migration/translation

Reproducibility

- For another day...

[1] (definition of) Performance Portability, 2016 Department of Energy Center of Excellence Meeting.

SYCL (pronounced ‘sickle’)

A C++-based open standard developed by Khronos Group

- Cross-platform abstraction layer

Provides a single-source programming model for development of heterogeneous systems

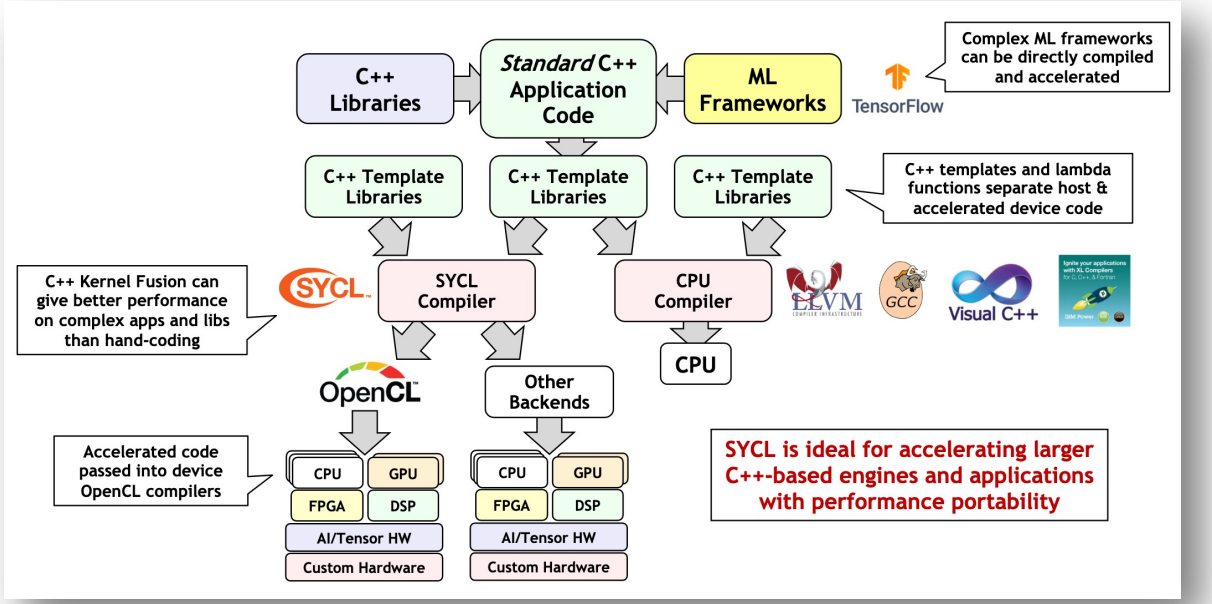
- Both low- and high-level codes

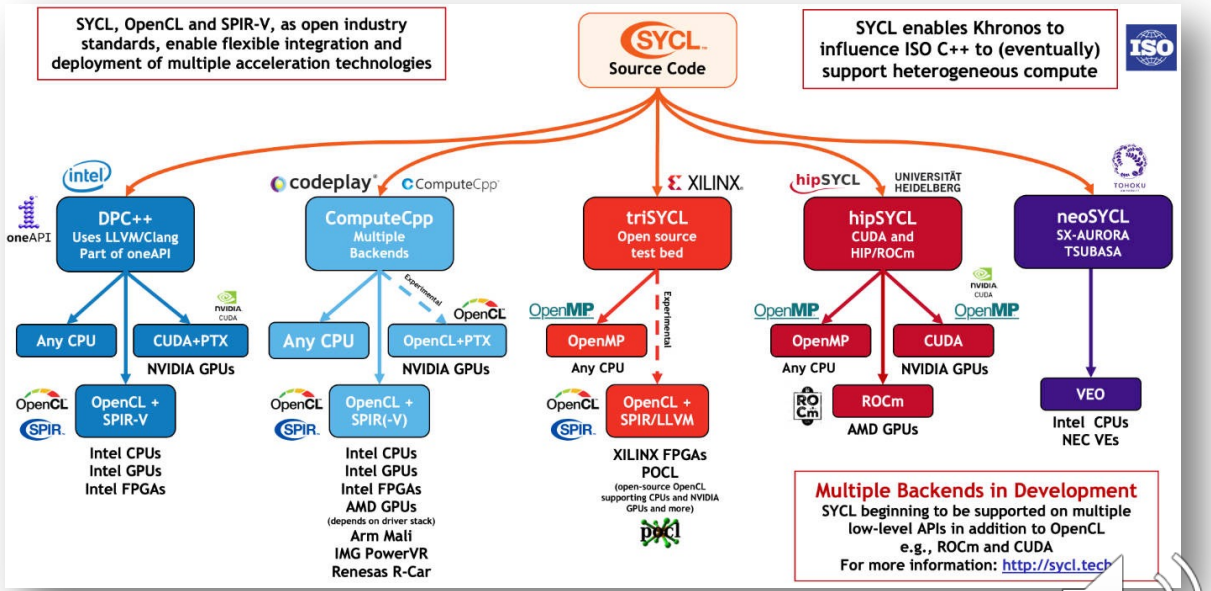
Vast ecosystem

- Numerous implementations, targeting different platforms

Notable features

- Unified Shared Memory (USM)
- C++-like atomic operations
- Interoperability





Khronos Group, SYCL 2020 (Web).

oneMKL open-source interfaces library (OSI)

Developed using SYCL programming model

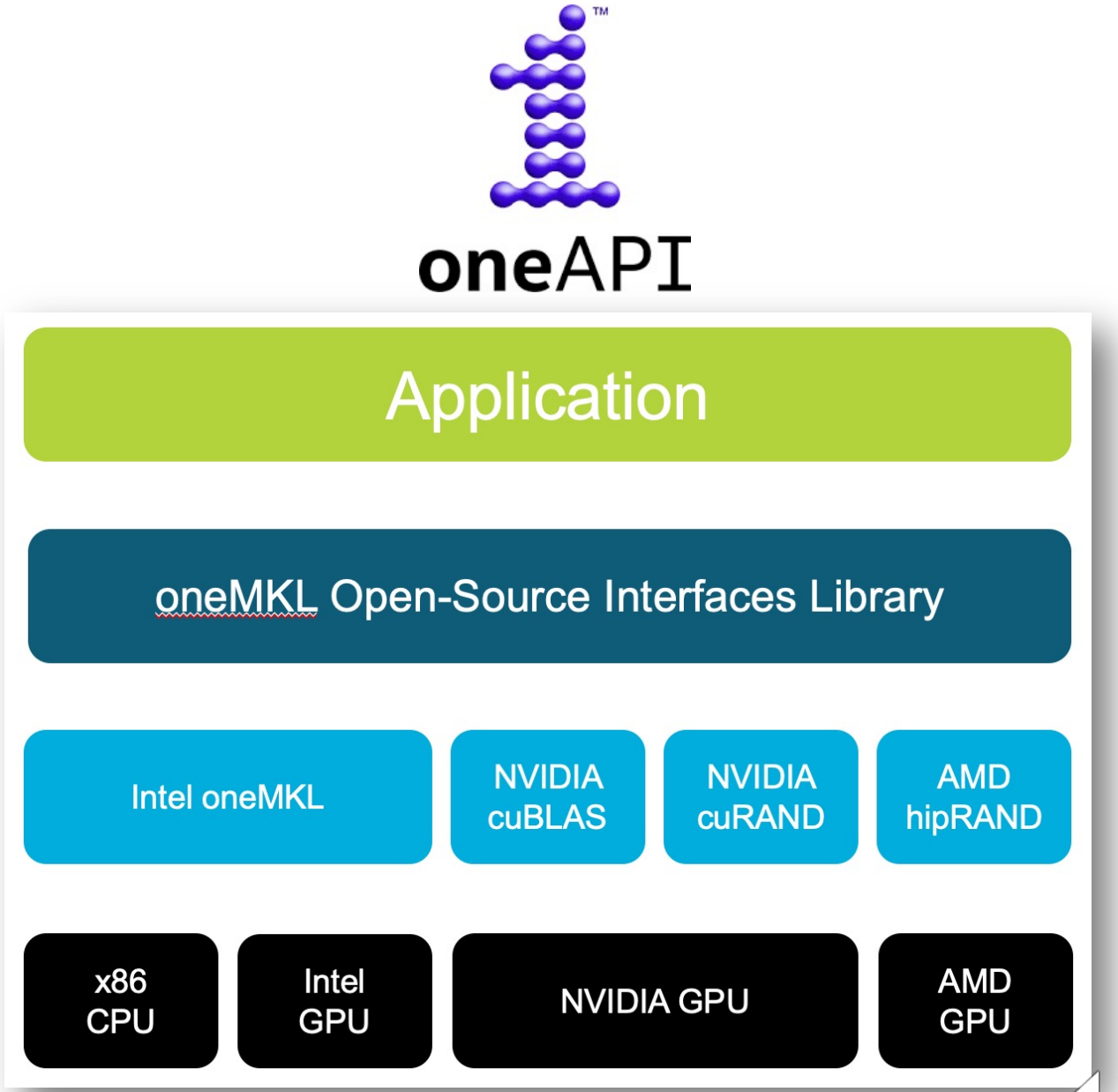
- Part of the oneAPI initiative

Linear algebra and random number generation (RNG) functionality

- NETLIB LAPACK
- Intel oneMKL*
- cuBLAS

Community-driven

- Technical Advisory Board members provide feedback to the overall oneAPI specification



oneapi-src/oneMKL (Github).

* Note the difference between oneMKL OSI and Intel oneMKL.

Integrating support for {cu,hip}RAND

Did not have resources to develop a new RNG

- Instead, utilize existing highly-optimized libraries

Required SYCL 2020 features, *e.g.*, `std::atomic_ref`, interoperability, ...

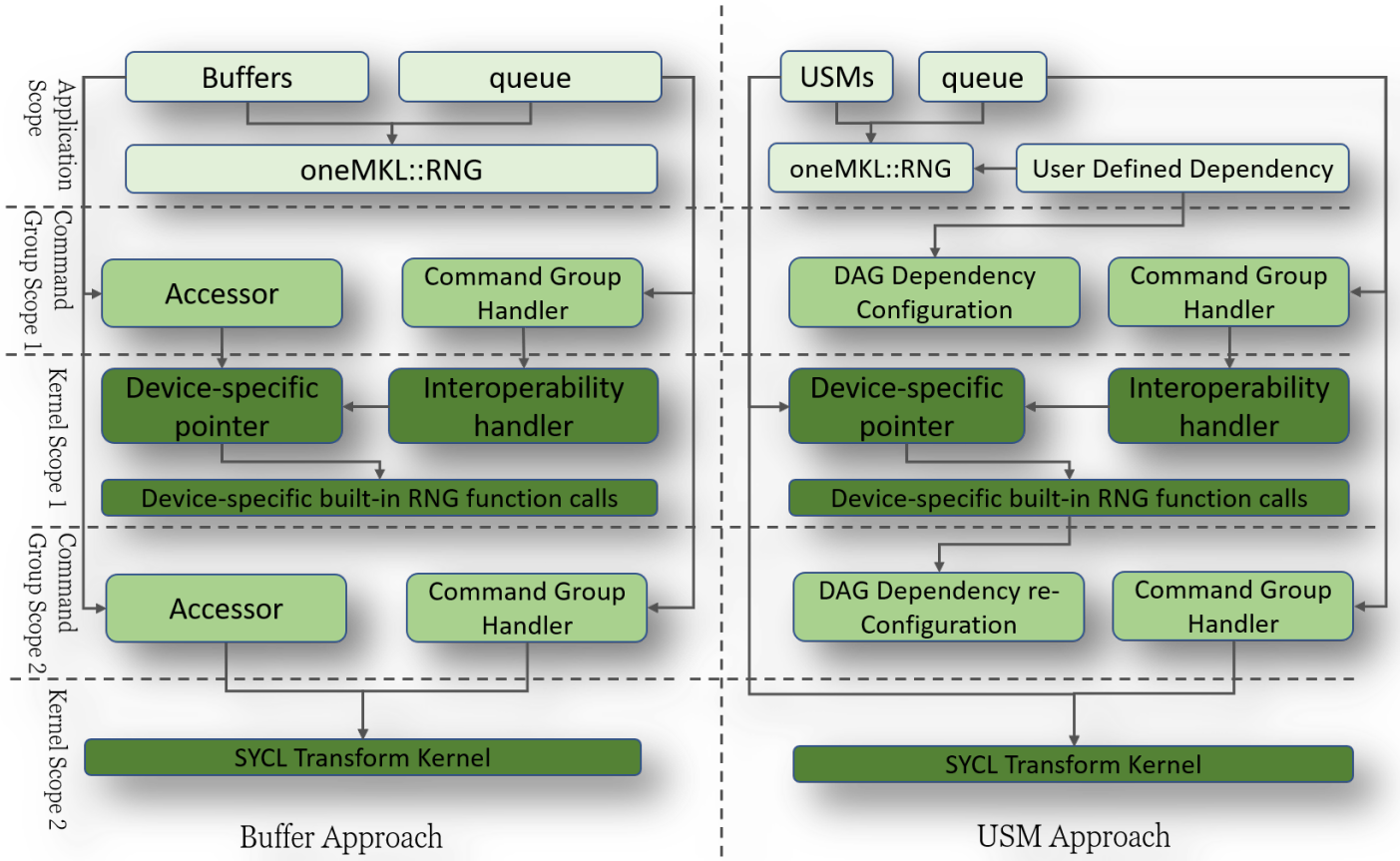
- intel/llvm
- illuhad/hipSYCL

oneMKL OSI does not provide handle to support resource allocation or kernel ordering

- Explicit synchronization between streams/queues to ensure order
- Global vs. per-`queue` contexts

Host and device APIs

- oneMKL support for host (`curand.h`)



RNG algorithms and kernels

oneMKL OSI implements Philox- and MRG-based algorithms

- 36 common high-level generate function templates (PImpl), 18 buffer and 18 USM
- Specify distribution and properties, and output types

`{cu,hip}RAND` have no concept of range, and distributions are coded into specific functions

- SYCL kernels written to address range transformations
- Distribution template parameter used to call correction native generate function

ICDF not supported by {cu}RAND pseudorandom generators

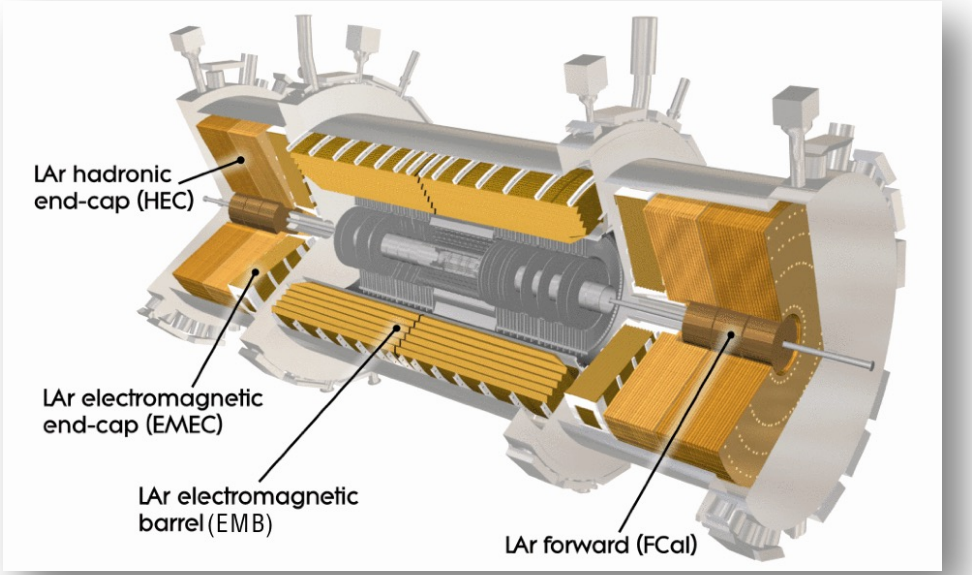
- 20/36 `generate` functions supported in our work

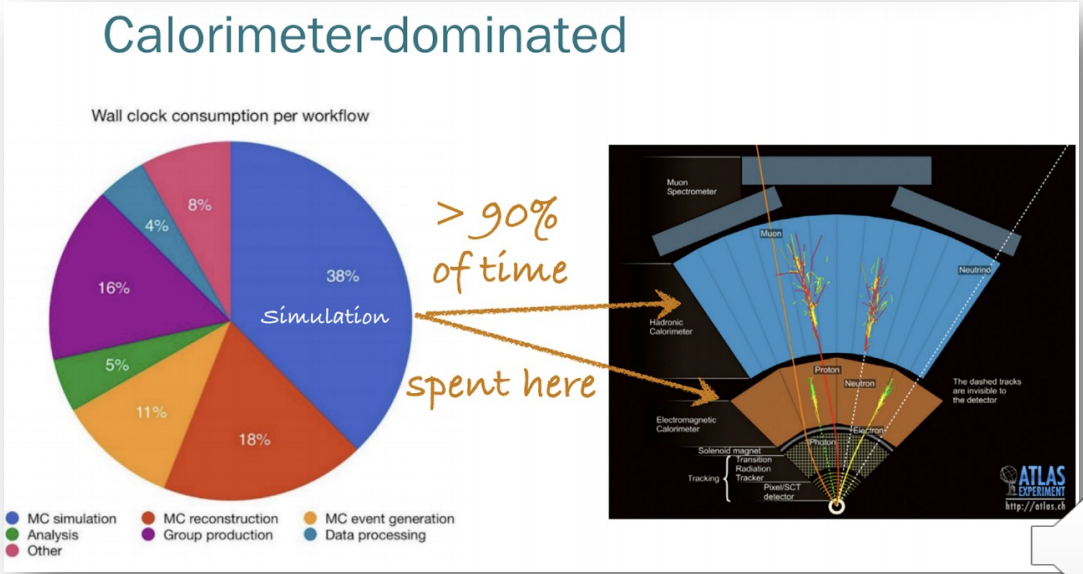
```
virtual inline void generate(
  const oneapi::mkl::rng::uniform<float, uniform_method::standard>& distr,
  std::int64_t n, cl::sycl::buffer<float, 1>& r) override {
    queue_.submit([&](cl::sycl::handler& cgh) {
      auto acc = r.get_access<cl::sycl::access::mode::read_write>(cgh);
      cgh.codeplay_host_task([=](cl::sycl::interop_handle ih) {
      auto r_ptr = reinterpret_cast<float*>(
        ih.get_native_mem<cl::sycl::backend::cuda>(acc));
      curandStatus_t status;
      CURAND_CALL(curandGenerateUniform, status, engine_, r_ptr, n);
      cudaError_t err;
      CUDA_CALL(cudaDeviceSynchronize, err);
    });
  });
  range_transform_fp<float>(queue_, distr.a(), distr.b(), n, r);
}
```

```
template <typename T>
static inline void range_transform_fp(cl::sycl::queue& queue, T a, T b,
                                      std::int64_t n,
                                      cl::sycl::buffer<T, 1>& r) {
  queue.submit([&](cl::sycl::handler& cgh) {
    auto acc =
      r.template get_access<cl::sycl::access::mode::read_write>(cgh);
    cgh.parallel_for(cl::sycl::range<1>(n), [=](cl::sycl::id<1> id) {
      acc[id] = acc[id] * (b - a) + a;
    });
  });
}
```

Benchmark applications

1. Single artificial benchmark used to stress hardware for different backends
    - Generates 1-10^8 random numbers
    - Common code to ensure consistent runtime behavior among backends
2. Parameterized calorimeter simulation software
    - 190k ‘sensors’, ~10 MB geometry
    - Inputs total ~GB, loaded at runtime
    - Single-particle simulations require 10^2-10^7 random numbers per event





Performance evaluation

Numerous definitions of *performance portability*

- Adopt from Pennycook *et. al.*[1]

Introduce *application efficiency* metric, *VAVS*

- Ratio between the time-to-solution (*TTS*) of portable implementation to the native
- Useful for identifying runtime overheads introduced by portability layers

Execute codes on a variety of machines with various software stacks

- GNU compiler for ISO C++
- hipSYCL targeting AMD GPU
- intel/llvm (DPC++) targeting SYCL on x86 and CUDA

$$\mathcal{P}(a,p;H) = \begin{cases} \dfrac{|H|}{\sum_{i \in H} \frac{1}{e_i(a,p)}} & \text{if } i \text{ is supported } \forall_i \in H \\ 0 & \text{otherwise} \end{cases}$$

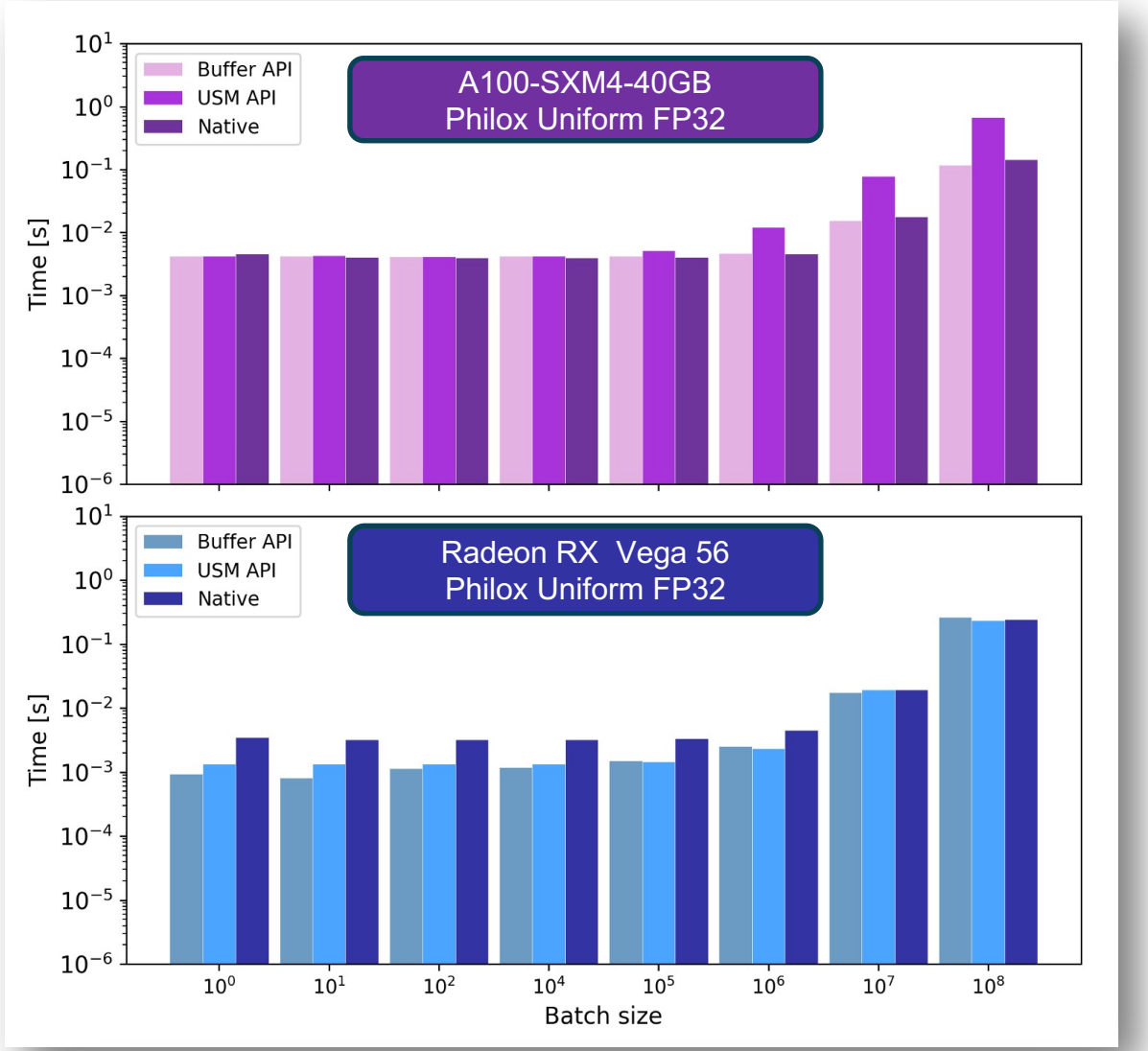
$$\text{VAVS} \equiv \frac{TTS_{\text{portable}}}{TTS_{\text{native}}}$$

Platform	Driver Version	OS and Kernel	Compiler	RNG Library
AMD Rome 7742	-	OpenSUSE 15.0 4.12	GNU 8.2.0 DPC++	CLHEP 2.3.4.6 oneMKL
Intel Core i7-1080H	-	Ubuntu 20.04 5.8.18	GNU 8.4.0 DPC++	CLHEP 2.3.4.6 oneMKL
Intel UHD Graphics	21.11.19310	Ubuntu 20.04 5.8.18	DPC++	oneMKL
Radeon RX Vega 56	20.50	CentOS 7 3.10.0	HIP 4.0.0 hipSYCL 0.9.0	hipRAND 4.0.0 oneMKL
NVIDIA A100	450.102.04	OpenSUSE 15.0 4.12	CUDA 10.2.89 DPC++	cuRAND 10.2.89 oneMKL

[1] Pennycook *et. al.* (2019) doi:10.1016/j.Future.2017.08.007.

Results
RNG burner

- Time-to-solution (TTS) using clocks shown for three kernels: `seed`, `generate` and `transform`
  - SYCL oneMKL OSI buffer and USM APIs
  - Native `{cu,hip}RAND`
- Benchmark ran 100 iterations (none discarded) for each batch size
- Increased TTS of USM on A100 due to explicit synchronization
- Reduced TTS of SYCL on AMD platform
  - Optimizations within hipRAND runtime system for ROCm backend
  - Callbacks introduce notable latencies in small kernels
  - Nearly callback-free hipSYCL runtime visible for batch sizes < 10^7



H	$\mathcal{P}$ buffer	$\mathcal{P}$ USM	$\mathcal{P}$ Mean (buffer+USM)
{Vega 56, A100}	1.070	0.393	0.575
{Vega 56}	0.974	1.076	1.022
{A100}	1.186	0.240	0.400

Results
RNG burner

Per-kernel TTS and relative occupancy for A100

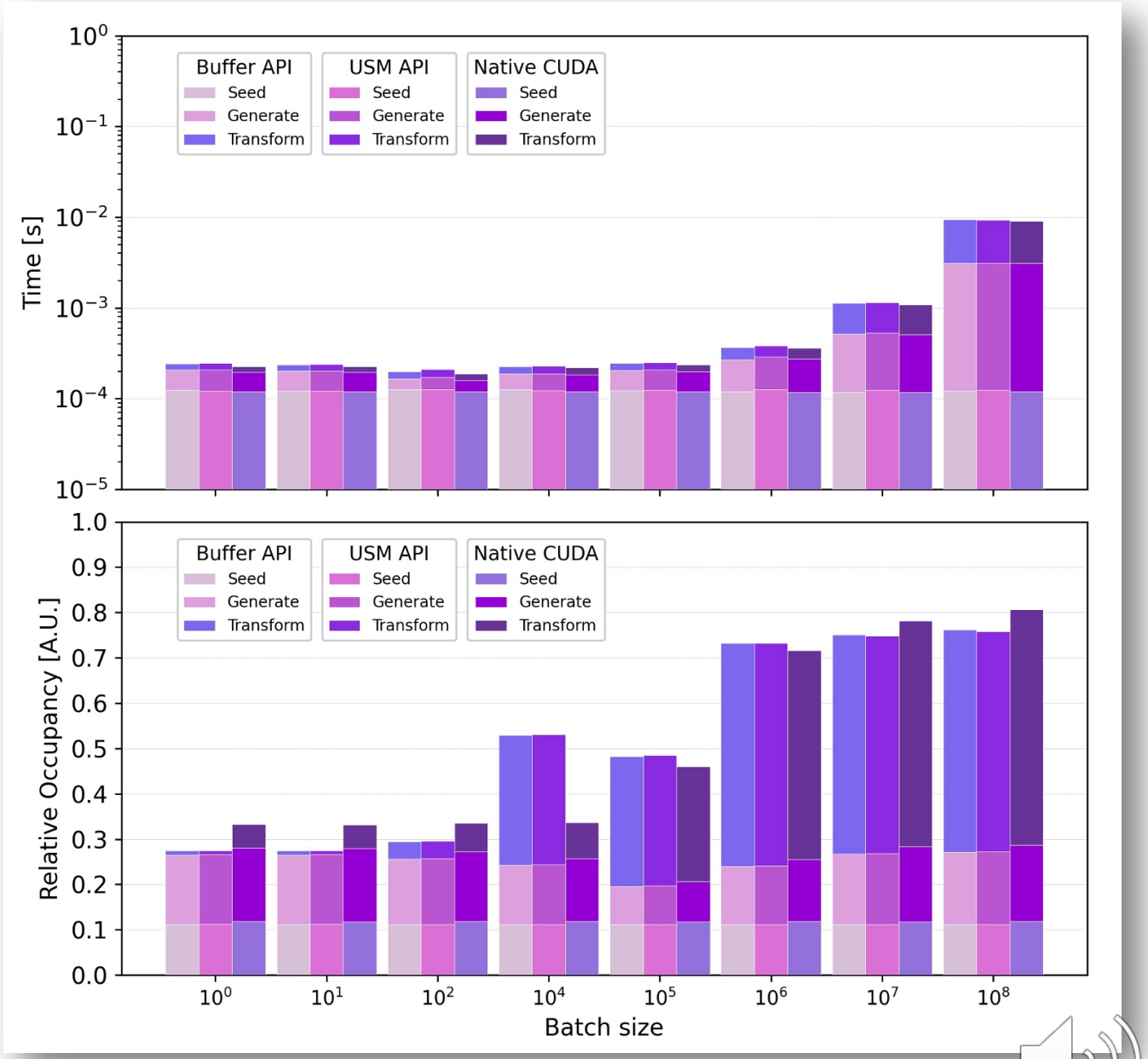
- Data collected using NVIDIA Nsight Compute 2020.2.1

Ten iterations for each backend/API

Both cuRAND kernels (seed and generate) are identical between oneMKL OSI and native

Large increase in relative occupancy between 10^2 and 10^4 for cuRAND kernels

- SYCL runtime system optimizes required block size and threads-per-block when not specified
- Native application fixed block size at 256 and SYCL runtime chose 1024 (no performance gains)



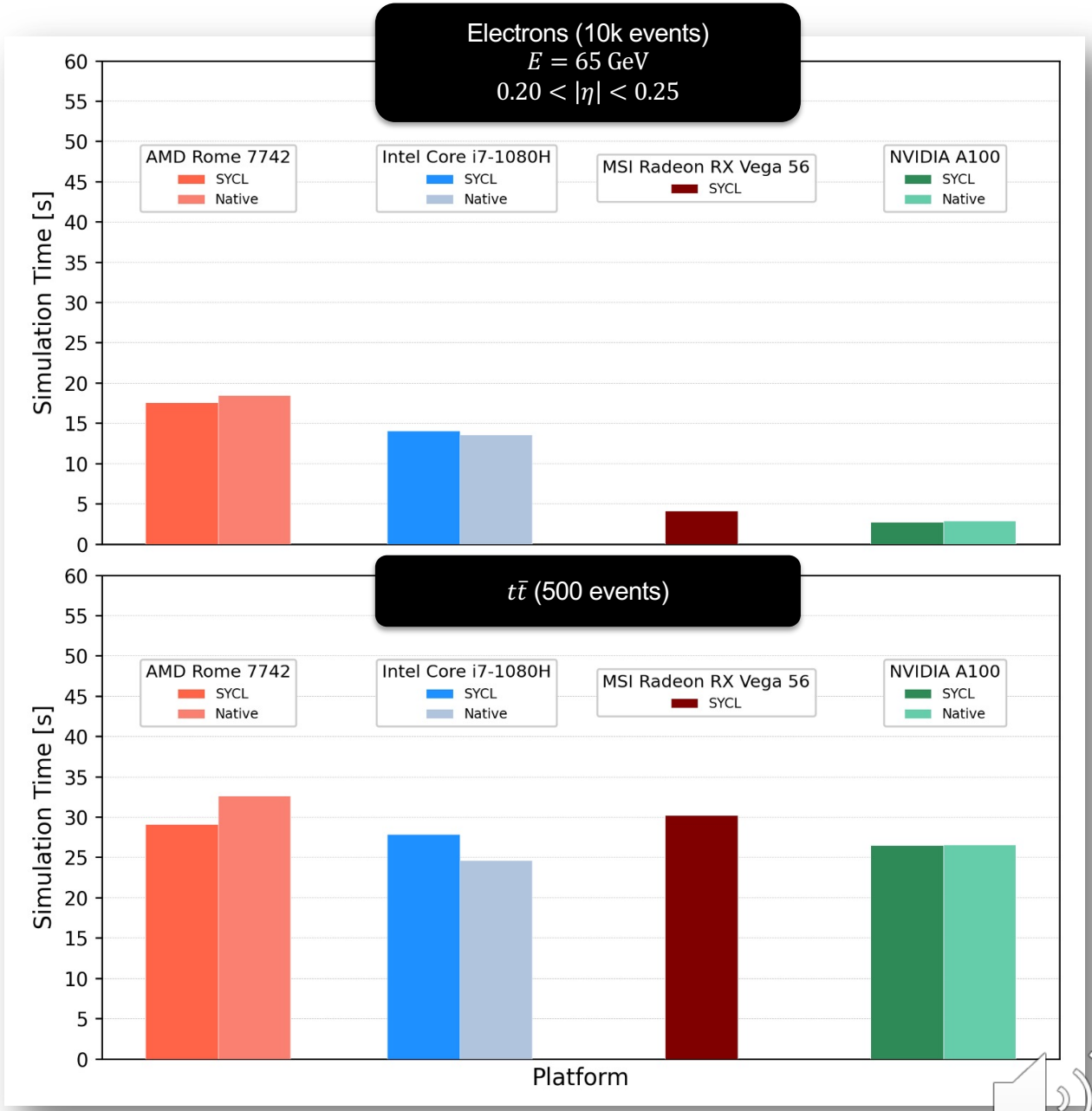
Results FastCaloSim

Demonstrate performance portability using real-world application (re-)written using SYCL programming model

Ten ‘runs’ of single-electron and top quark pair production simulations

- AMD CPU targeted using `host_device` (TBB, no OpenCL backend)
- Intel CPU targeted using `cpu_device` (OpenCL backend)
- ~80% TTS reduction for single electrons when executed using GPU offload
- Top quark simulations achieve no gains on GPUs due to lack of inter-event parallelism and runtime data movement host → device loading parameterizations

The same source runs across four different platforms with fair performance



Summary and Conclusions

High-Energy Physics, and big data science in general, can no longer rely on CPU alone

- Long had single architecture, similar cluster-based resources

Pressure to demonstrate HPC utilization

- Cannot maintain multiple large codebases
- Need a performance portable solution

We investigate SYCL and interoperability with existing highly-optimized vendor libraries

- Achieve considerable performance across four major vendors, two of which are now supported by this work

Plenty of ideas for future work

- Heuristic methods for choosing optimal backend
- Purely SYCL-based math libraries (reproducibility)
- New applications and opportunities in quantum simulations