Computing and Software Infrastructure

Fermilab-CERN HCP Summer School 2024
25-26 July 2024

Oliver Gutsche - Fermilab

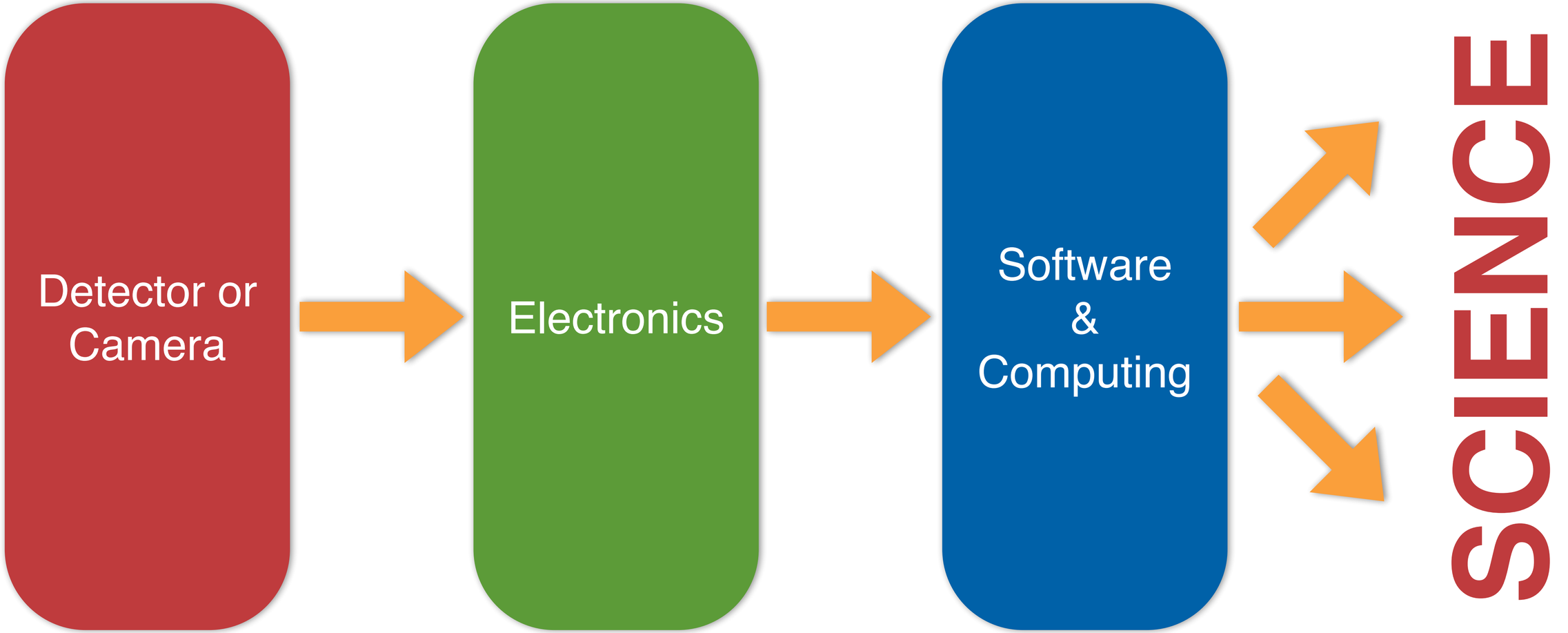
Introduction and Disclaimer

- Oliver “Oli” Gutsche
  - Senior Scientist at Fermilab
  - Member of CMS since 2005
  - Technical specialization: software and computing
  - Currently: U.S. CMS Software & Computing Operations Program manager and active in Fermilab CS/AI management

- Software and Computing Infrastructure every difficult to cover entirely in 2 hours
  - Will try to give an overview and point out important areas
  - Had a lot of help from friends and family
  - I am biased towards LHC computing for CMS, Department of Energy computing infrastructure, NSF OSG and IRIS-HEP computing and software ➜ there is a lot more out there!
  - All mistakes are mine. I appreciate your help in improving the talk.

Motivation

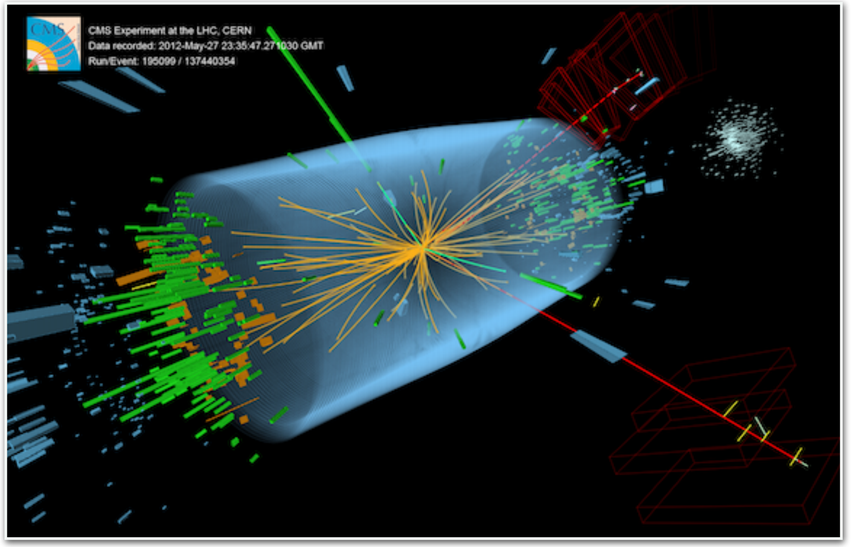
Particle Physics



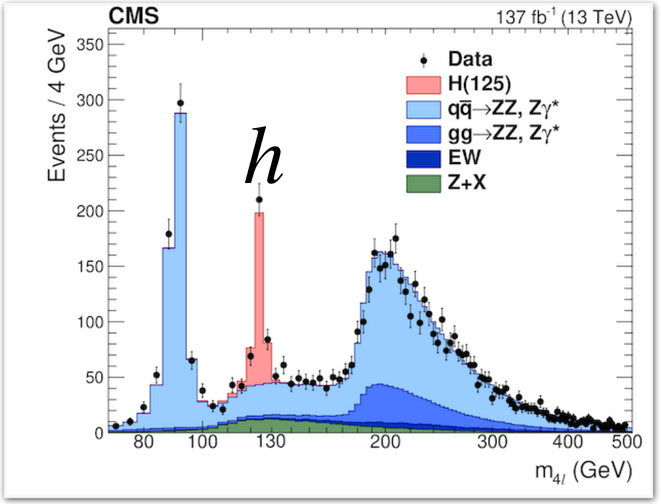
Three steps

1. Collide particles

2. Take pictures



3. Infer parameters

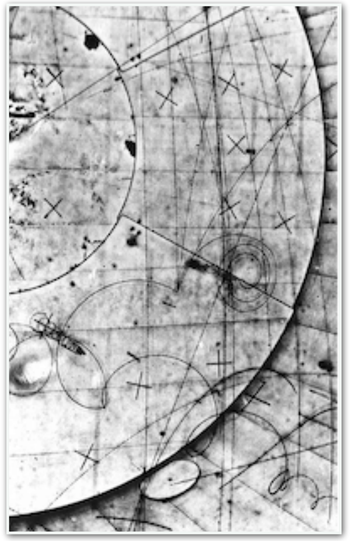


Software and Computing Infrastructure - Why should I care?

AI generated cartoon of a haystack

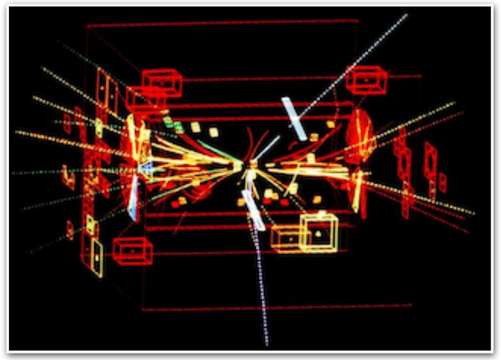
- Particle Physics is a statistical science
- We need many computationally intensive ingredients
  - Reconstructed data
  - Monte Carlo Simulations
  - Machine Learning Training and Inference and others
- Need to store and provide access to a large amount of data
  - O(100) Petabytes

Scale



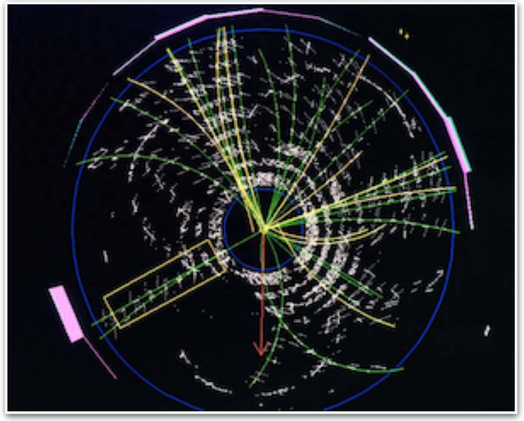
CERN Big European Bubble Chamber (BEBC)
1 photo / event
~6M events

cds:1733654

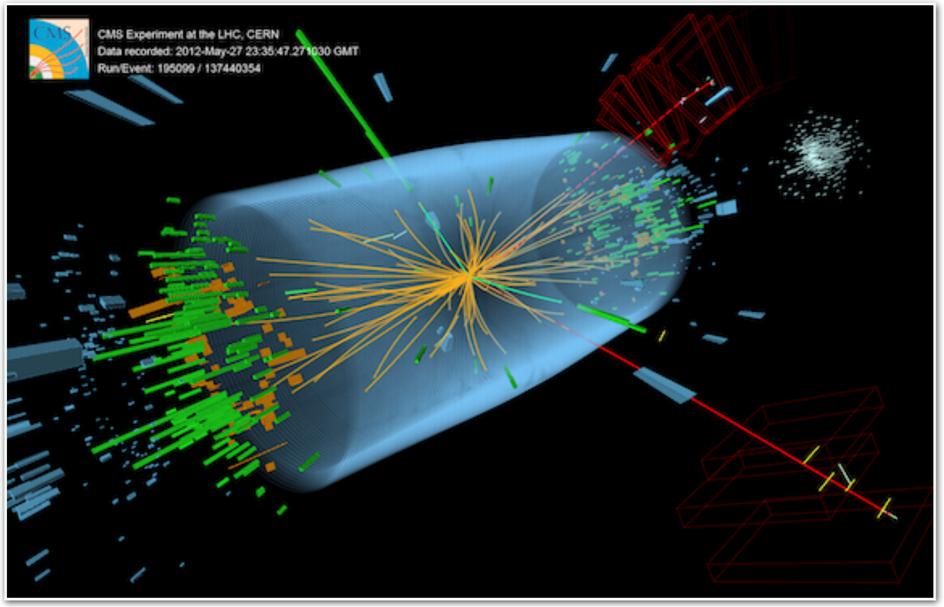


CERN UA1
~100kB / event
~10M events

cds:182190



Fermilab Tevatron
~100kB / event
~1B events 10.1016/j.nima.2017.01.043



CERN LHC (CMS)
~1MB / event
~100B events

So why should I really care?

Climate impacts of particle physics

Kenneth Bloom[1,*], Veronique Boisvert[2,**], Daniel Britzger[3], Micah Buuck[4], Astrid Eichhorn[5], Michael Headley[6], Kristin Lohwasser[7], and Petra Merkel[8]

[1]University of Nebraska-Lincoln, Lincoln, NE, USA
[2]Royal Holloway University London, United Kingdom
[3]Max-Planck-Institute for Physics, Munich, Germany
[4]SLAC National Accelerator Laboratory, Menlo Park, CA, USA
[5]CP3-Origins, University of Southern Denmark, Denmark
[6]Sanford Underground Research Facility (SURF), Lead, SD, USA
[7]University of Sheffield, United Kingdom
[8]Fermi National Accelerator Laboratory (Fermilab), Batavia, IL, USA

Abstract. The pursuit of particle physics requires a stable and prosperous society. Today, our society is increasingly threatened by global climate change. Human-influenced climate change has already impacted weather patterns, and global warming will only increase unless deep reductions in emissions of CO_2 and other greenhouse gases are achieved. Current and future activities in particle physics need to be considered in this context, either on the moral ground that we have a responsibility to leave a habitable planet to future generations, or on the more practical ground that, because of their scale, particle physics projects and activities will be under scrutiny for their impact on the climate. In this white paper for the U.S. Particle Physics Community Planning Exercise ("Snowmass"), we examine several contexts in which the practice of particle physics has impacts on the climate. These include the construction of facilities, the design and operation of particle detectors, the use of large-scale computing, and the research activities of scientists. We offer recommendations on establishing climate-aware practices in particle physics, with the goal of reducing our impact on the climate. We invite members of the community to show their support for a sustainable particle physics field [1].

arxiv:2203.12389

- International Panel on Climate Change:
  - "It is unequivocal that human influence has warmed the atmosphere, ocean and land. Widespread and rapid changes in the atmosphere, ocean, cryosphere and biosphere have occurred."
  - "Global warming of 1.5 C and 2 C will be exceeded during the 21st century unless deep reductions in CO2 and other greenhouse gas emissions occur in the coming decades."
- Limiting warming requires significant reductions in CO2 and other greenhouse gas emissions, in line with Paris Agreement.
  - Every 1000 gigaton of cumulative CO2 emissions leads to 0.27-0.63 C increase in warming → must adhere to a carbon budget.
  - IPCC: Total budget of 300 gigaton CO2e (CO2 equivalent) emissions for 83% chance to limit warming to < 1.5 C →
    - **1.1 tCO2e per capita per year until 2050.**
  - U.S. has a significant role to play in this:
    - Current per capita per year rate: ~14 tCO2e, ~3x global average.

Software and Computing Infrastructure - Carbon Footprint

- Data centers and computing contribute 2-4% of global green house gas emissions, only expected to grow.
- Up-front considerations: where do we place computing facilities and how are they powered?
  - Great variation of electricity emissions across countries and even regions.
- Can we be smarter about how we use existing facilities?
  - Can compute centers expose information on their specific carbon impact, so that experiments can use it in scheduling?
  - Can we schedule jobs to run at times when electricity supplies tend to be cheaper/cleaner (midday/nighttime)?
  - Can we consider carbon impact as an element of computing “performance” in benchmarking?
  - Can we invest in optimization of power consumption for products/libraries in widespread use in the field?
    - (Or at least track progress over release history?)
- Looking ahead: electricity must be de-carbonized, but:
  - Expect higher demand for electricity overall.
  - Concerns about “embodied carbon” in computing facilities.

Ten simple rules to make your computing more environmentally sustainable

- Rule 1: Calculate the carbon footprint of your work
- Rule 2: Include the carbon footprint in your cost–benefit analysis
- Rule 3: Keep, repair, and reuse devices to minimize electronic waste
- Rule 4: Choose your computing facility
- Rule 5: Choose your hardware carefully
- Rule 6: Increase efficiency of the code
- Rule 7: Be a frugal analyst
- Rule 8: Releasing a new software? Make its hardware requirements and carbon footprint clear
- Rule 9: Be aware of unanticipated consequences of improved software efficiency
- Rule 10: Offset your carbon footprint

PLOS COMPUTATIONAL BIOLOGY

EDITORIAL

Ten simple rules to make your computing more environmentally sustainable

Loïc Lannelongue[1,2,3], **Jason Grealey**[4,5], **Alex Bateman**[6]*, **Michael Inouye**[1,2,3,4,7,8]

1 Cambridge Baker Systems Genomics Initiative, Department of Public Health and Primary Care, University of Cambridge, Cambridge, United Kingdom, **2** British Heart Foundation Cardiovascular Epidemiology Unit, Department of Public Health and Primary Care, University of Cambridge, Cambridge, United Kingdom, **3** Health Data Research UK Cambridge, Wellcome Genome Campus and University of Cambridge, Cambridge, United Kingdom, **4** Cambridge Baker Systems Genomics Initiative, Baker Heart and Diabetes Institute, Melbourne, Victoria, Australia, **5** Department of Mathematics and Statistics, La Trobe University, Melbourne, Australia, **6** European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Wellcome Genome Campus, Hinxton, United Kingdom, **7** British Heart Foundation Centre of Research Excellence, University of Cambridge, Cambridge, United Kingdom, **8** The Alan Turing Institute, London, United Kingdom

* agb@ebi.ac.uk

OPEN ACCESS

Citation: Lannelongue L, Grealey J, Bateman A, Inouye M (2021) Ten simple rules to make your computing more environmentally sustainable. PLoS Comput Biol 17(9): e1009324. https://doi.org/10.1371/journal.pcbi.1009324

Editor: Russell Schwartz, Carnegie Mellon University, UNITED STATES

Published: September 20, 2021

Funding: LL was supported by the University of Cambridge MRC DTP (MR/S502443/1). JG was supported by a La Trobe University Postgraduate Research Scholarship jointly funded by the Baker Heart and Diabetes Institute and a La Trobe University Full-Fee Research Scholarship. This work was supported by core funding to MI from the: UK Medical Research Council (MR/L003120/1), British Heart Foundation (RG/13/13/30194;RG/18/13/33946), NIHR Cambridge Biomedical Research Centre (BRC-1215-20014) and Health Data Research UK, which is funded by the UK Medical Research Council, Engineering and Physical Sciences Research Council, Economic and Social Research Council, Department of Health and Social Care (England), Chief Scientist Office of the

Introduction

We are in the midst of a man-made climate emergency, and yet, there is a widespread under-appreciation in our community as to the effects of our computations on carbon emissions and global warming. Global temperatures are rising, largely caused by greenhouse gas (GHG) emissions, which is leading to melting of the ice caps and permafrost. Acidification of the oceans is threatening the entire ocean ecosystem. Global biodiversity is rapidly declining across the globe. Although science can contribute to our understanding of the environment and has great potential to develop strategies and technology to slow down global warming, we do not have a free pass when it comes to the environmental impact of our work. There is little doubt that data science in particular will be a key tool to tackle climate change, but we often forget to consider how our own work also contributes to the problem. The infrastructure we use and the algorithms and code that we write and run all consume large quantities of electricity, whose production is responsible for significant GHG emissions.

It was estimated that the IT sector was responsible for 2% to 6% of global CO_2 emissions in 2020, a share that could grow to 20% by 2030 [1]. Data centres themselves have a substantial carbon footprint of around 100 megatons of CO_2e (comparable to American commercial aviation) emitted just from the yearly generation of 200 TWh of electricity [2]. Projections estimate that this footprint will increase by 2- to 9-fold in the next decade [3], with an electricity usage potentially as high as 974 TWh in 2030 [4]. This doesn't even include the environmental impact of producing and disposing of the hardware needed for computation. Cryptocurrencies are another source of concerns, and, although they rely on dedicated mining farms and hardware, their energy usage (estimated at 70 TWh/year in July 2021) is growing at a worrying pace [5]. Coupling these substantial carbon footprints with the growing global demand for computation warrants that we must both individually and collectively do our utmost to make our computations more environmentally friendly. Here, we describe 10 simple rules to help achieve this.

Rule 1: Calculate the carbon footprint of your work

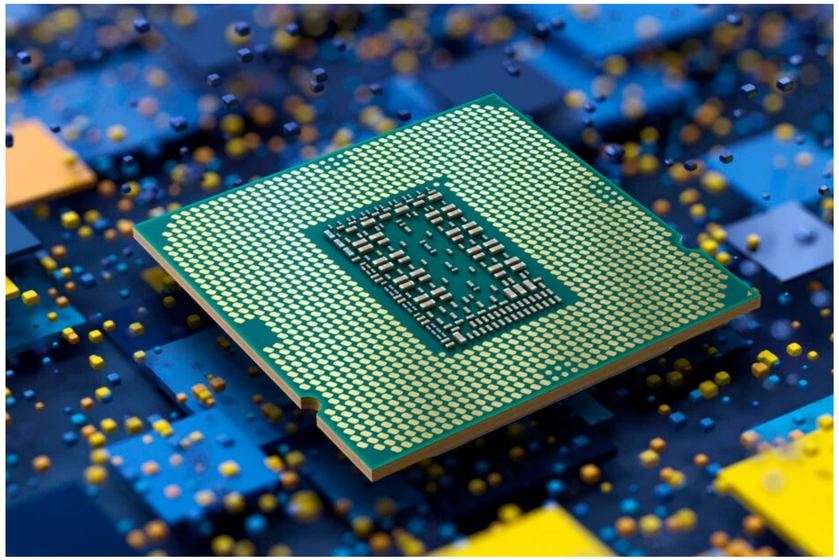
We live in a world ruled by data, where a problem doesn't exist until it has been measured. There is still very limited information available about the carbon footprint of computational

https://doi.org/10.1371/journal.pcbi.1009324

Computing Hardware in the Datacenter

Rule 4: Choose your computing facility

Computing Hardware - Generic View



Compute

Storage

HDD

SSD

Networking

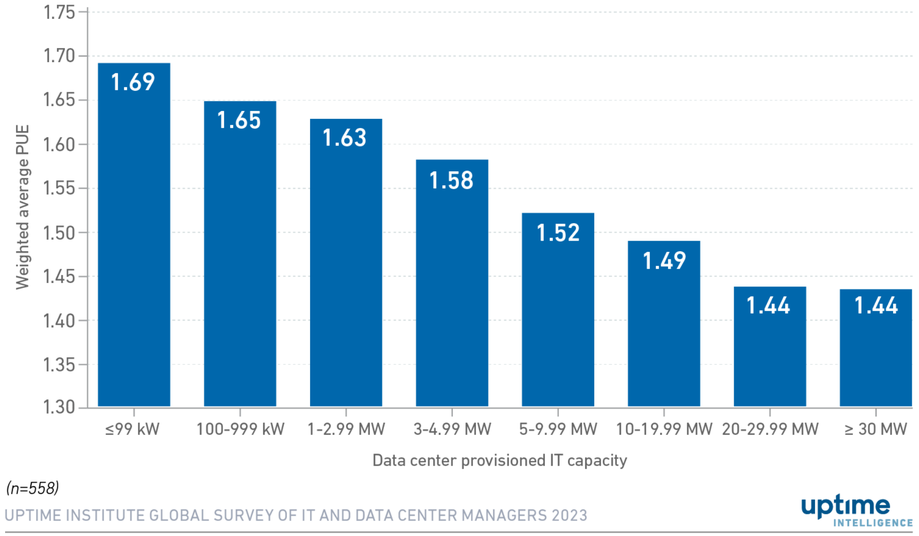
New: GPUs

Hardware in the data center

- CPU box (1U)
  - Dual-socket unit
  - Current generation: 1 kW for 256 hyper threaded cores
- Harddrive storage box
  - 3-4 U for up to 60 hdds
  - Current generation: 500 W for 1 PB
- SSD storage box
  - 1 U for up to X NVME drives
  - Current generation: Y W for Z PB
- Networking equipment
  - Routers and switches
- Tape libraries
  - 10k tape cartridges and O(50) drives
- GPUs
  - Specialized 3-4U box with up to 8 GPUs
  - Current generation: 3-4 kW for 8 NVidia A100

Datacenter

- Climate controlled building with enough electrical and cooling power for all the hardware
- Large rooms with racks
  - Rack: 40-50U
- Cooling:
  - Forced Air or Water
  - Newest developments: immersion cooling
- Power Usage Efficiency (PuE)
  - https://en.wikipedia.org/wiki/Power_usage_effectiveness
  - “Power usage effectiveness (PUE) or power unit efficiency is a ratio that describes how efficiently a computer data center uses energy; specifically, how much energy is used by the computing equipment (in contrast to cooling and other overhead that supports the equipment).”
  - Average PuE: 1.4-1.7 (between 40% and 70% is “wasted!”)



(n=558)

UPTIME INSTITUTE GLOBAL SURVEY OF IT AND DATA CENTER MANAGERS 2023

uptime INTELLIGENCE

https://journal.uptimeinstitute.com/large-data-centers-are-mostly-more-efficient-analysis-confirms/

Exercise: Analysis Carbon Footprint

- Assumption: Chicago data center
  - PuE 1.6
  - Electricity mix
    - (https://app.electricitymaps.com/zone/US-MIDA-PJM):
      - 35% low-carbon, 1% renewable: **433 gCO2 / kWh**
- Modern CPU system (Dual-CPU 32 core AMD EPYC): **1.17 W/HS23**
  - 1.8 KW, 128 HT cores
- Modern Disk server (24 disk JBOD): **0.61 W/TB**
  - 250W, 410 TB usable
- CMS workload:
  - Produce MC: **1098 HS23s**
  - Reconstruct data: **333 HS23s**
  - Analysis data: **1.5 kB / event**
- Naive Exercise:
  - 1B MC events ➜ **248 tCO2**
  - 1B data events ➜ **75 tCO2**
  - Provide access to 1B MC and 1B data events for 1 year ➜ **11 tCO2**
- **Compare to yearly CO2 budget per capita of 1.1 tCO2 / year**
- **2 min exercise: estimate carbon footprint of your analysis needs**

Rule 4: Choose your computing facility

- Perlmutter @ NERSC
  - Supercomputer at the National Energy Research Scientific Computing Center (NERSC) at Berkeley Lab
  - NVIDIA A100 GPUs, AMD "Milan" EPYC CPUs, a novel HPE Slingshot high-speed network, and a 35 petabyte – all FLASH – scratch file system
  - Direct water cooling
  - **PuE: 1.05 - 1.08**
  - https://perlmutter.carrd.co/

- Green IT Cube: supercomputing center for GSI and FAIR
  - 768 racks, 12 MW cooling capacity
  - water cooling in doors of computer cabinets
  - **PuE: < 1.07**
  - https://www.gsi.de/en/researchaccelerators/research_an_overview/green-it-cube

What computing hardware to use?

Rule 5: Choose your hardware carefully

Rack density

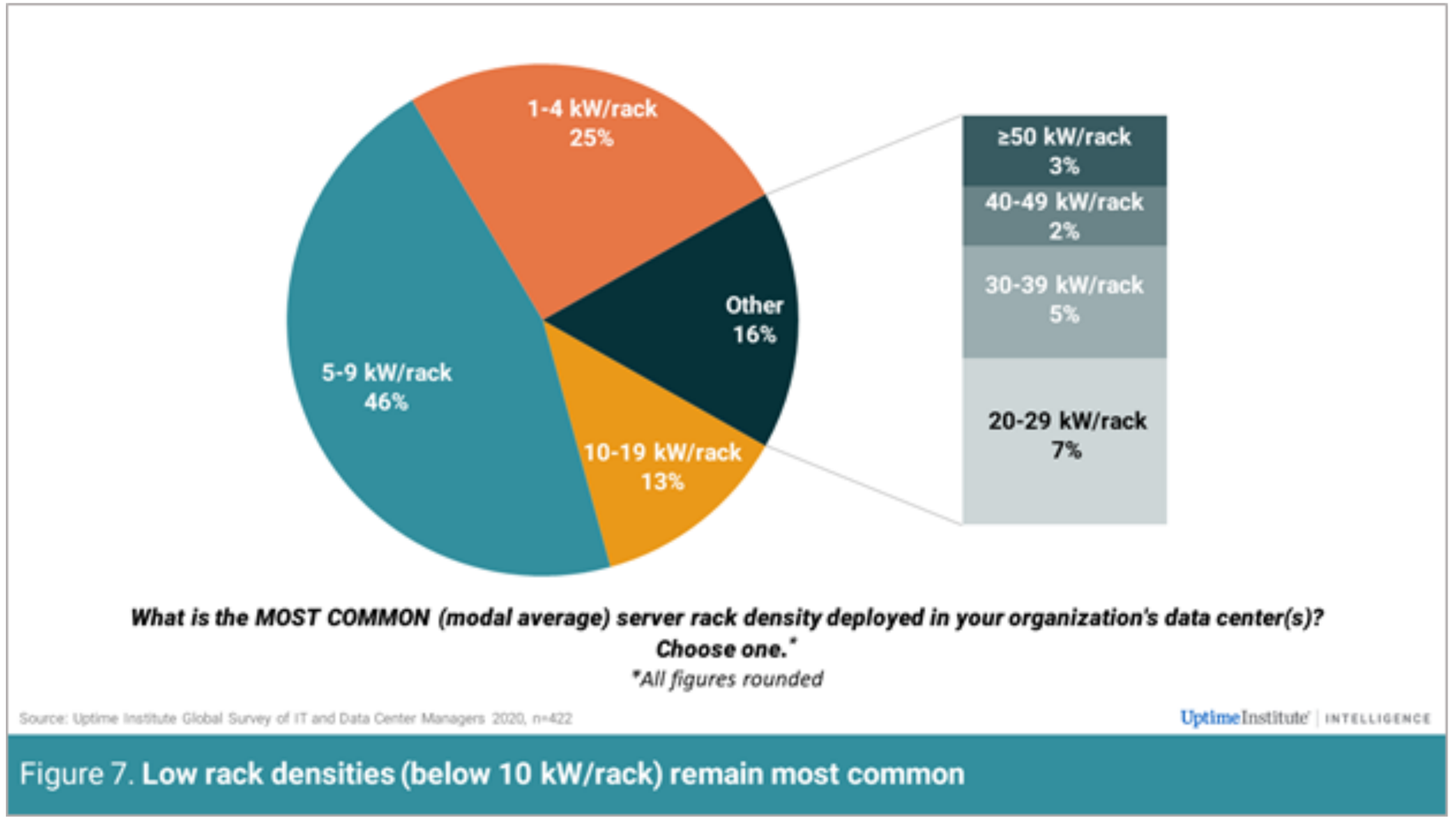
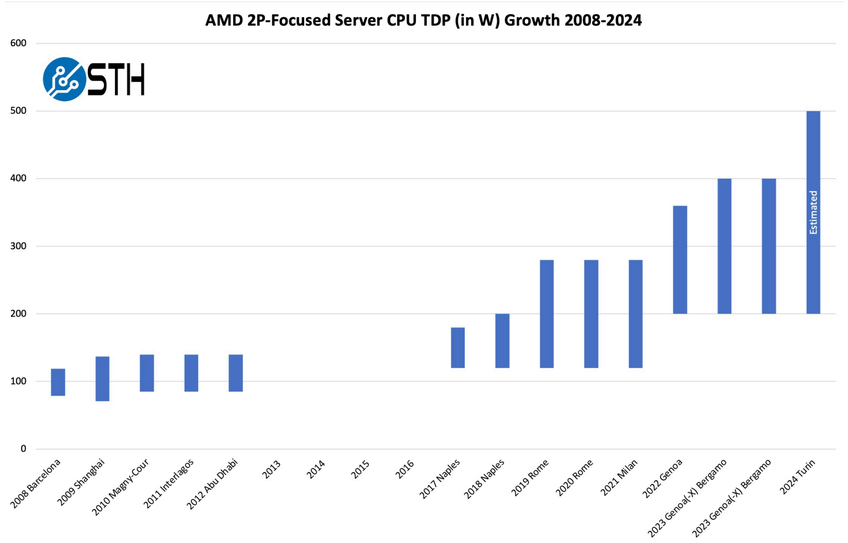


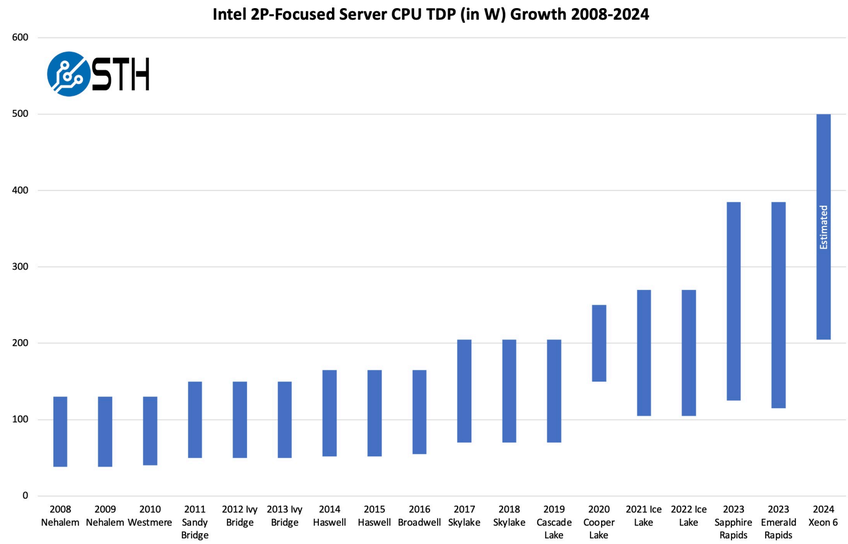
Figure 7. **Low rack densities (below 10 kW/rack) remain most common**

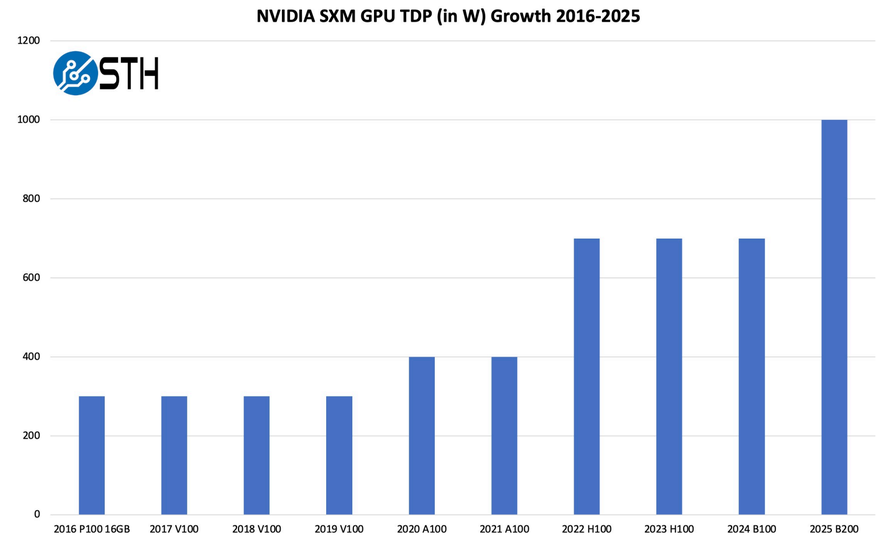
https://journal.uptimeinstitute.com/rack-density-is-rising/

- Rack capabilities (electricity and cooling) defines how we can deploy hardware in the data center
  - Current average rack density: 12 kW ➜ 6 example CPU boxes???

Systems consume more and more energy







- x86 CPUs are approaching 500W
- GPUs are approaching 1 kW
- Question: Is ARM a possibility?
  - Gains more traction in the data center after taking over the cell phone and laptop market
  - But introduces a non-x86 architecture for software

High end example

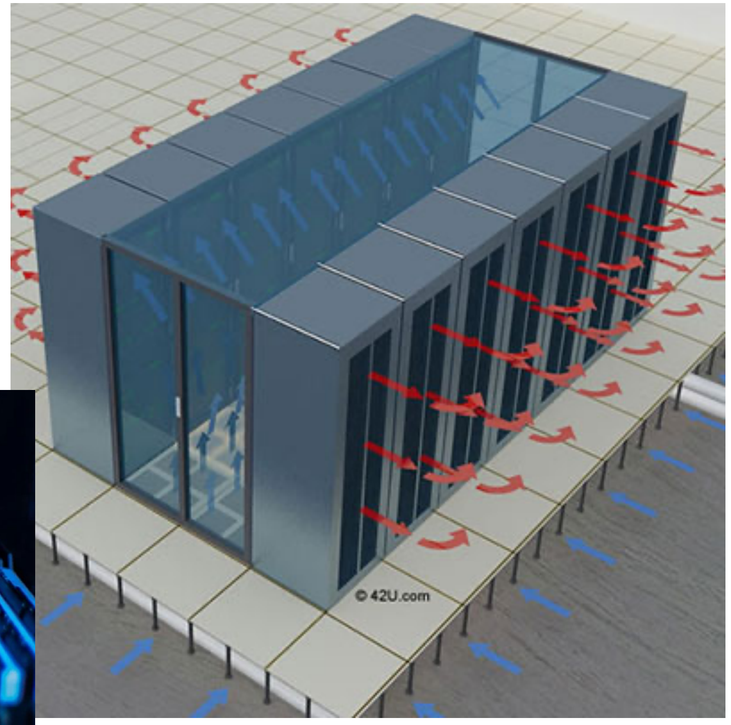
https://www.nvidia.com/en-us/data-center/gb200-nvl72/

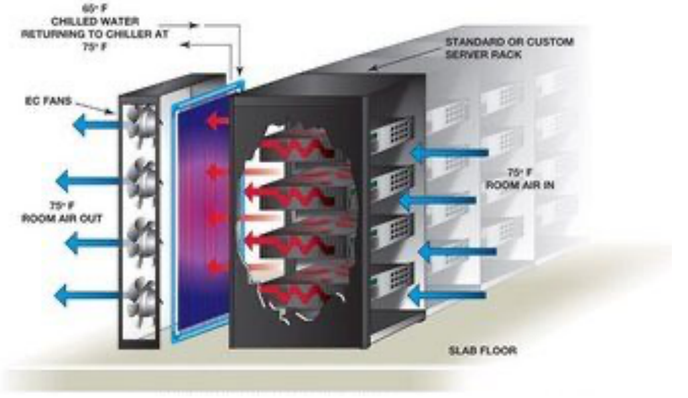
- NVIDIA GB200 NVL72
  - Connects 36 Grace CPUs and 72 Blackwell GPUs in a rack-scale design.
  - Liquid-cooled
  - 72-GPU NVLink domain that acts as a single massive GPU
  - 120 kW!

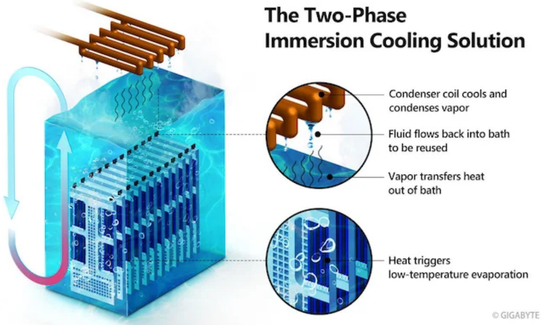
- Tesla Cybertruck: 123 kW battery
- NVidia GPU rack uses 1 Cybertruck EVERY HOUR!

About cooling

- 2 main types of cooling computing hardware in data centers
  - Forced Air cooling
  - Liquid cooling
    - Direct-to-Chip (D2C) Cooling
    - Rear Door Heat Exchanger (RDHx)
    - Immersion Cooling
  - Differ in effectiveness and how dense computing hardware can be deployed in the data center





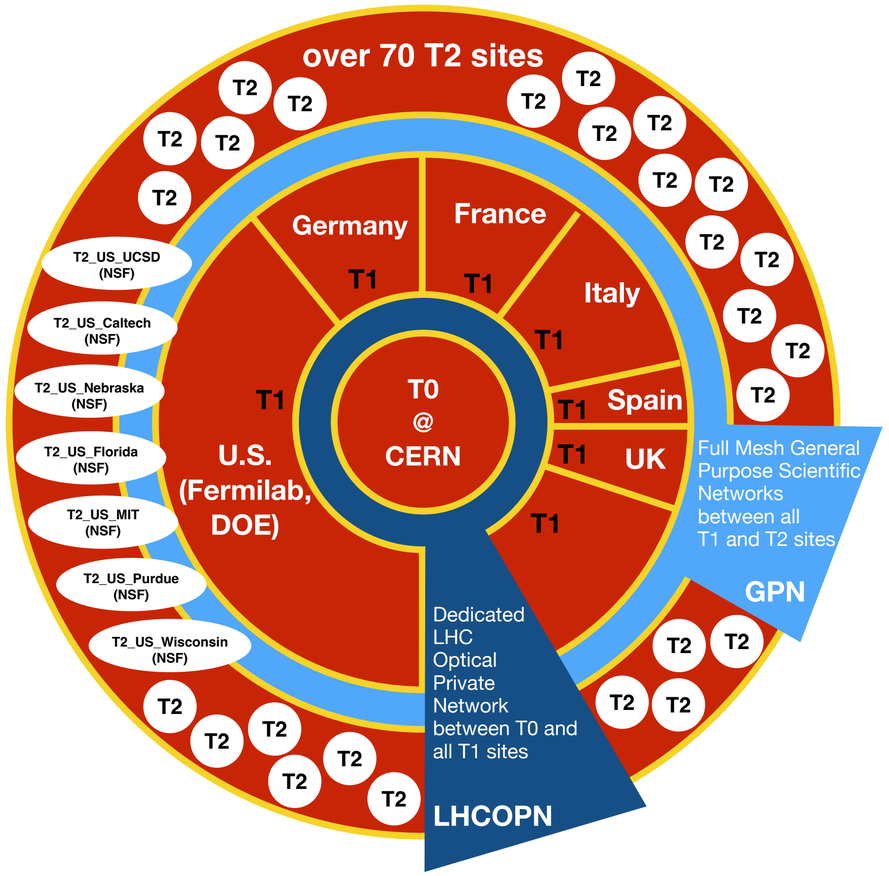


What did we learn so far?

- Software and Computing is integral part of the science process
- This does not come for free, carbon footprint is not negligible
- Computing hardware performance increases a lot over time, but at the cost of higher energy density
  - Data centers will have to accommodate this
- Energy efficient data centers are coming, but not everywhere and not fast enough!
- **Conclusion:**
  - **Computing infrastructure will be distributed and specialized**
  - **Need to be able to orchestrate our work**
    - **Centrally on large scales and down to the individual researcher**

Say goodbye to the old days! The Grid!

- HEP computing has a long tradition
  - Some of the first linux clusters were used by HEP
  - Some of the first wide area network connections were used by HEP
- Since the start of the LHC, HEP computing is firmly based on distributed computing
  - “Seamless” access to computing all over the world
- The Grid
  - Authentication & Authorization!
- All based on resources that are funded, built and operated by the HEP community
  - We get funding for our own data centers or co-locate in large scientific data centers
- In terms of computing resources, this means
  - **We get access to a certain amount of computing capacity for a whole year, renewed every year, 24/7/365**
  - **We can use unused capacity opportunistically on a fair-share basis**
- **Everything is x86-based!**



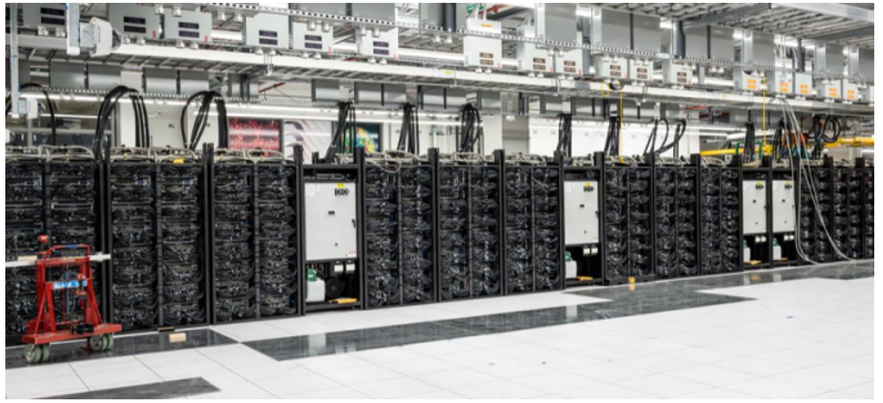
The new frontier - commercial clouds

- Industry can do grid computing now much better than us
  - Massive data centers with $B investment provide access to vast amounts of resources
  - HEP resources are sizable but tiny compared to industry
  - Industry selling compute in small slices for profit
  - Allows for both large scale (if you can pay for it) and fine granularity
  - Some hyperscalers (Google) offer subscription models that allow to boost into unused capacity
  - In general higher prices to buy elasticity

This is not a warehouse!

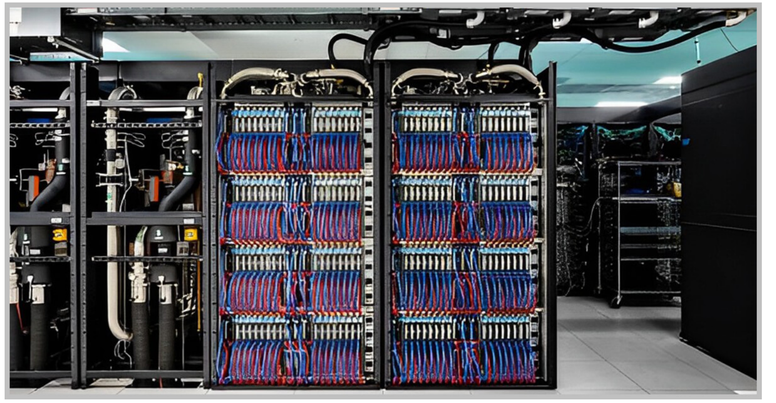
The new frontier - Supercomputers

- High Performance Computing (HPC) is designed for single large applications using significant resources
  - Think climate models or lattice QCD
  - Requires specialized hardware with very fast interconnects (makes a lot of processors act like one big CPU)
  - Recently they are opening up to HEP workflows (HEP = high throughput computing (HTC))
    - Fast interconnects don’t play a too big role



Frontier at Oak Ridge National Laboratory

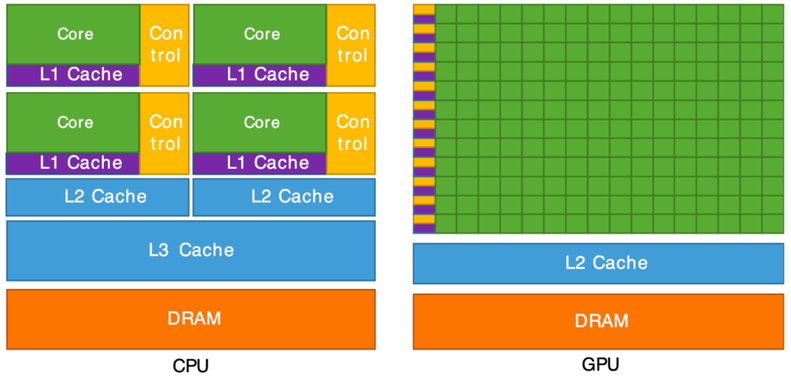
- HPC Installations accessible to HEP are publicly funded and processing time is allocated through approval processes based on science use case
  - Difficult for HEP because we tend to run workflows for large number of science use cases
- Clash also with the planning process
  - resources are not necessarily available 24/7/365,
  - work can be queued for a HPC and has to wait till there is room in the queue of the HPC ➜ predictability of resource planning is different



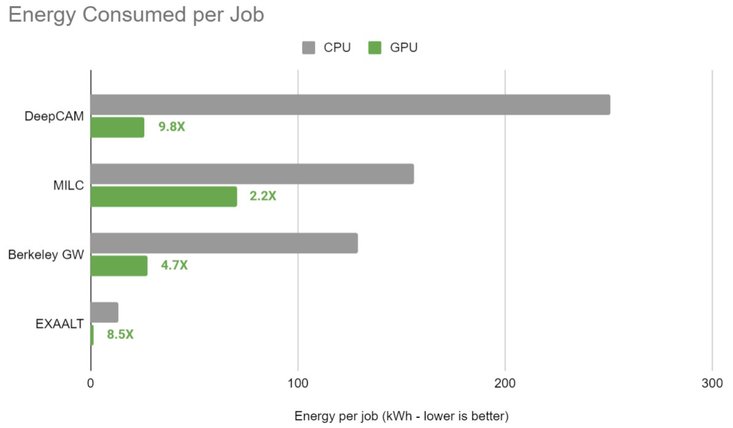
Aurora at Argonne National Laboratory

GPUs improve energy efficiency

- There is a large difference between CPUs and GPUs
  - CPUs don't have as many arithmetic logic units (ALUs) or floating point units (FPUs) as GPUs (the small green boxes, roughly speaking), but the ALUs and FPUs in a CPU core are individually more capable.
  - CPUs have more cache memory than GPUs. GPU cores are less powerful than CPU cores and have less memory.
  - While CPUs can switch between different instruction sets rapidly, a GPU simply takes a high volume of the same instructions and pushes them through at high speed.
- A GPU has a much better energy efficiency to do the same computation
  - **On Perlmutter @ NERSC, energy efficiency rose 5x on average. An application for weather forecasting logged gains of 9.8x.**
- Without GPUs, large HPCs would consume so much electricity making them impossible to build and operate
- GPUs are (currently) the future of energy efficient large scale computing
  - But you need to know how to used them, how write software for them!

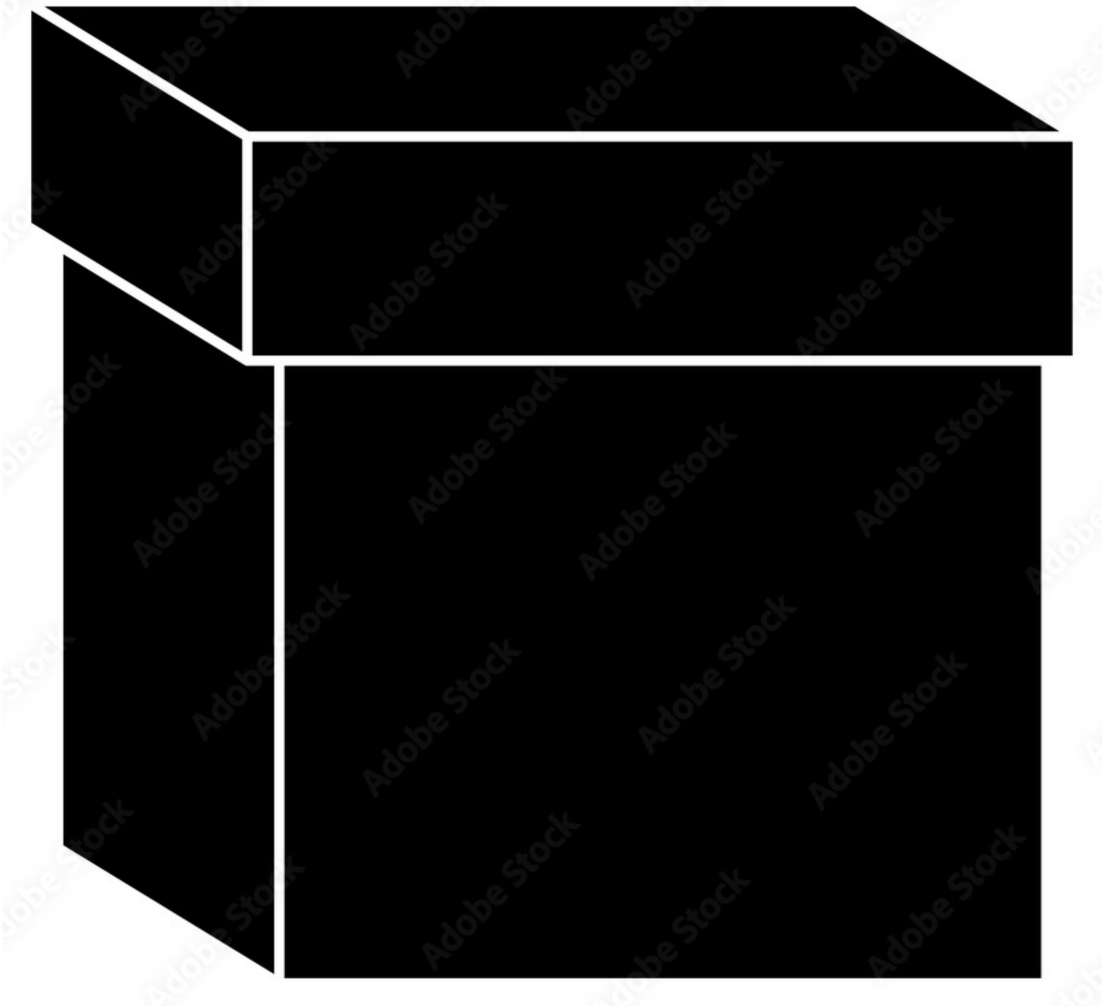


https://cvw.cac.cornell.edu/gpu-architecture/gpu-characteristics/design



https://blogs.nvidia.com/blog/gpu-energy-efficiency-nersc/

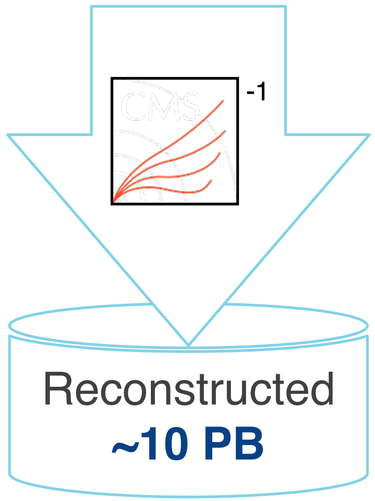
Quantum Computing

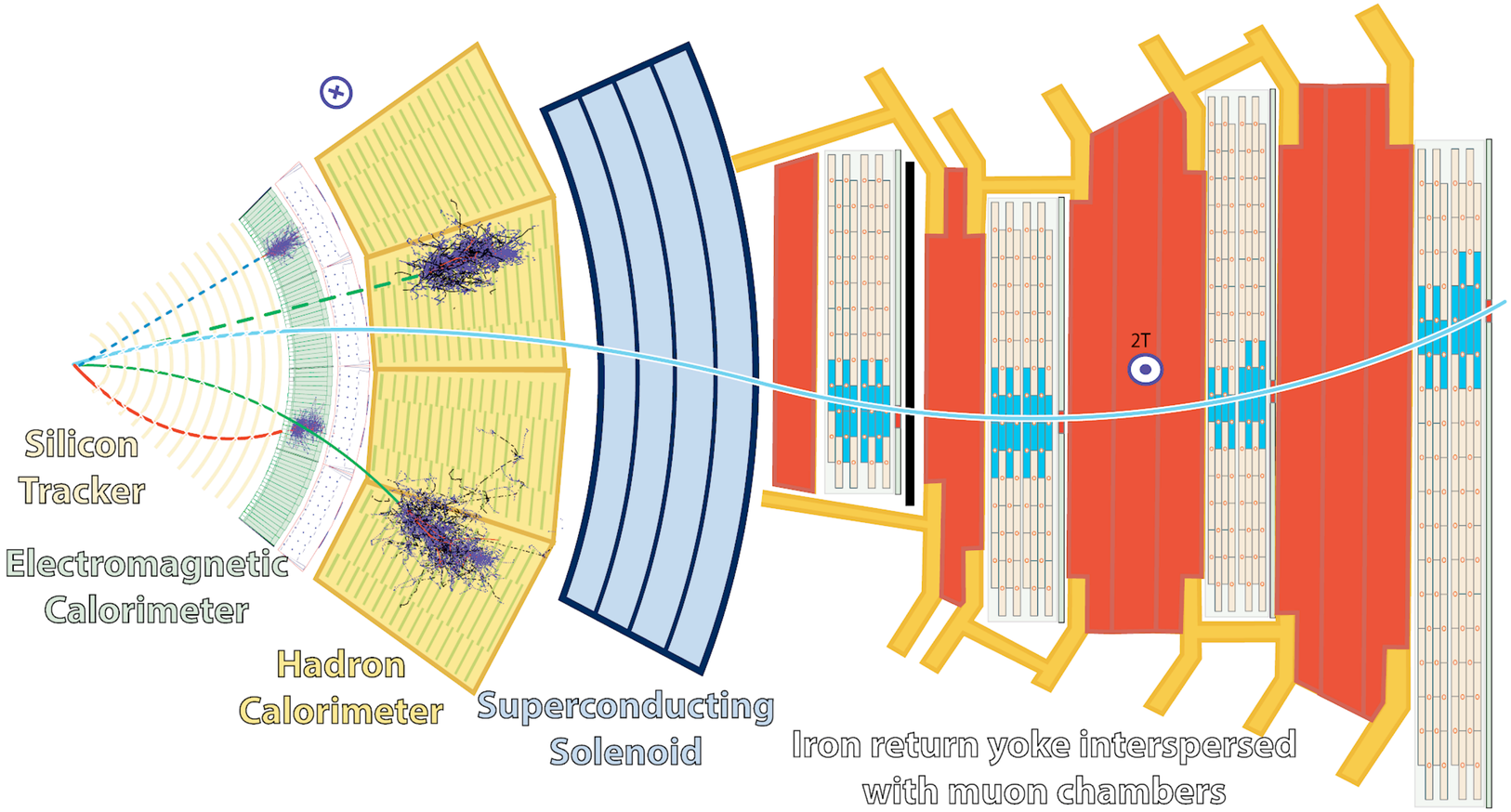


Software is the key to science results

Rule 6: Increase efficiency of the code

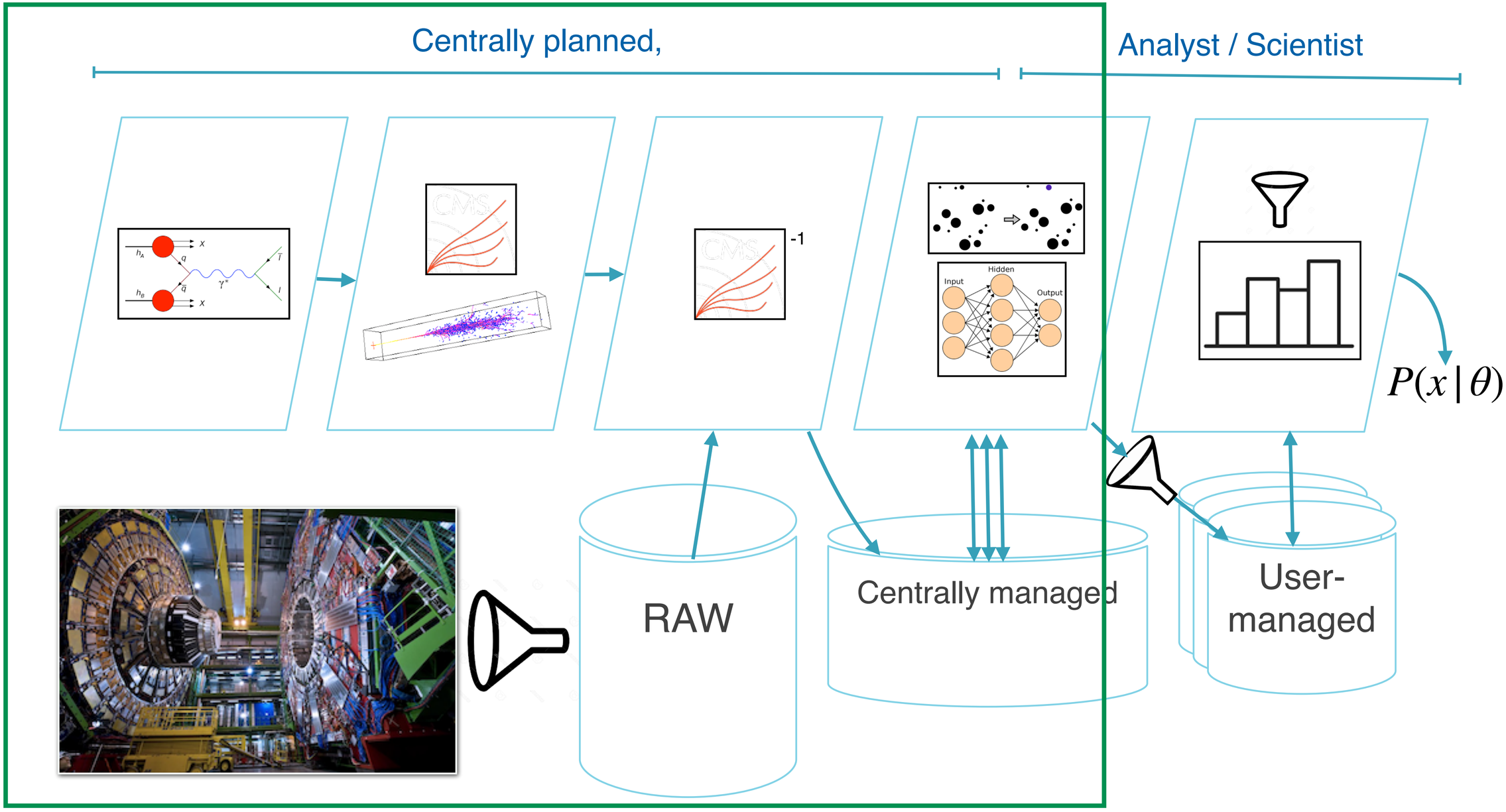
Software is needed to reduce the data space





Muon
Electron
Charged hadron (e.g. pion)
Neutral hadron (e.g. neutron)
Photon

Software is everywhere



In the old times, there was Fortran

Fortran

Article Talk

From Wikipedia, the free encyclopedia

Fortran (/ˈfɔːrtræn/; formerly **FORTRAN**) is a third generation, compiled, imperative programming language that is especially suited to numeric computation and scientific computing.

Fortran was originally developed by IBM.[3] It first compiled correctly in 1958.[4] Fortran computer programs have been written to support scientific and engineering applications, such as numerical weather prediction, finite element analysis, computational fluid dynamics, geophysics, computational physics, crystallography and computational chemistry. It is a popular language for high-performance computing[5] and is used for programs that benchmark and rank the world's fastest supercomputers.[6][7]

Fortran has evolved through numerous versions and dialects. In 1966, the American National Standards Institute (ANSI) developed a standard for Fortran to limit proliferation of compilers using slightly different syntax.[8] Successive versions have added support for a character data type (Fortran 77), structured programming, array programming, modular programming, generic programming (Fortran 90), parallel computing (Fortran 95), object-oriented programming (Fortran 2003), and concurrent programming (Fortran 2008).

Since April 2024, Fortran has ranked among the top ten languages in the TIOBE index, a measure of the popularity of programming languages.[9]

https://en.wikipedia.org/wiki/Fortran

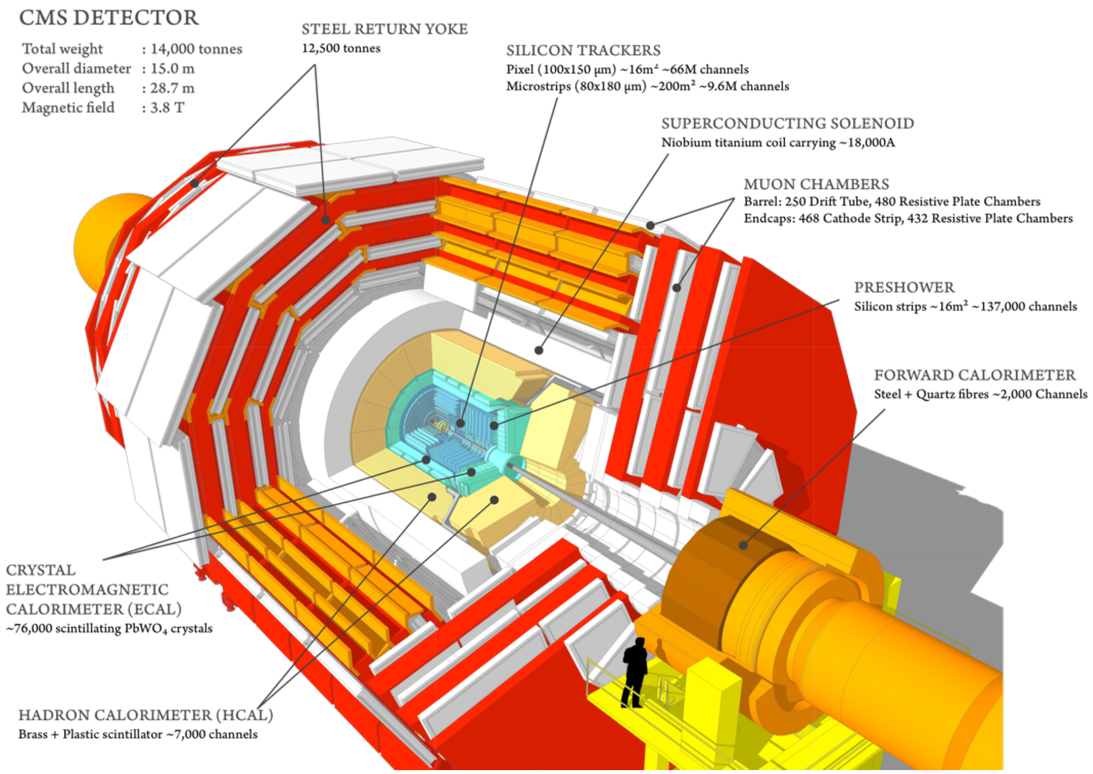
Fortran

Paradigm	Multi-paradigm: structured, imperative (procedural, object-oriented), generic, array
Designed by	John Backus
Developer	John Backus and IBM
First appeared	1957; 67 years ago
Stable release	Fortran 2023 (ISO/IEC 1539:2023) / November 17, 2023; 7 months ago
Typing discipline	strong, static, manifest
Filename extensions	`.f90`, `.f`, `.for`
Website	fortran-lang.org
Major implementations	
Absoft, Cray, GFortran, G95, IBM XL Fortran, Intel, Hitachi, Lahey/Fujitsu, Numerical Algorithms Group, Open Watcom, PathScale, PGI, Silverfrost, Oracle Solaris Studio, others	
Influenced by	
Speedcoding	
Influenced	
ALGOL 58, BASIC, C, Chapel,[1] CMS-2, DOPE, Fortress, MATLAB, PL/I, PACT I, MUMPS, IDL, Ratfor, SAKO (programming language)[2]	

```fortran
RECURSIVE FUNCTION gcd_recursive(a, b) RESULT (gcd)

IMPLICIT NONE

INTEGER :: a, b, gcd

IF (b == 0) THEN

   gcd = a

ELSE

   gcd = gcd_recursive(b, MOD(a, b))

END IF

RETURN

END FUNCTION gcd_recursive
```

AI generated with codellama

Why are we not writing everything in Fortran?



The Details

The CMS collaboration has around:

6288

ACTIVE PEOPLE
(PHYSICISTS, ENGINEERS, TECHNICAL, ADMINISTRATIVE, STUDENTS, ETC.)

Of these members there are about:

2166	1228	1102	1388
PHD PHYSICISTS (1769 MEN, 397 WOMEN)	PHYSICS DOCTORAL STUDENTS (919 MEN, 309 WOMEN)	ENGINEERS (951 MEN, 151 WOMEN)	UNDERGRADUATES (995 MEN, 393 WOMEN)

A typical CMS physics paper will be signed by the PhD physicists and a significant fraction of the doctoral students meaning it will typically have about 2100 signatures.

- HEP uses large detector systems with a variety of different technologies
- Physicists and engineers are developing the technologies **and in the end write the software for it**
- Need to support a large domain-expert developer base (not programming experts) who work together on a single application to reconstruct the detector signals

C++

- C++ provided a solution to the HEP challenge
  - Object-oriented approach to divide up the development into small pieces that individual experts can work on encapsulating algorithms and data structures
  - Software frameworks
    - schedule algorithms and build logical execution trees
    - provide data persistency functionalities
  - CMS code: 4 Million lines of C+ (with 1M lines of python to configure it) with hundreds of modules

C++ (/ˈsiː plʌs plʌs/, pronounced "**C plus plus**" and sometimes abbreviated as **CPP**) is a high-level, general-purpose programming language created by Danish computer scientist Bjarne Stroustrup. First released in 1985 as an extension of the C programming language, it has since expanded significantly over time; as of 1997, C++ has object-oriented, generic, and functional features, in addition to facilities for low-level memory manipulation for systems like microcomputers or to make operating systems like Linux or Windows. It is usually implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Embarcadero, Oracle, and IBM.[14]

https://en.wikipedia.org/wiki/C++

C++

Logo endorsed by the C++ standards committee

Paradigms	Multi-paradigm: procedural, imperative, functional, object-oriented, generic, modular
Family	C
Designed by	Bjarne Stroustrup
Developer	ISO/IEC JTC 1 (Joint Technical Committee 1) / SC 22 (Subcommittee 22) / WG 21 (Working Group 21)
First appeared	1985; 39 years ago
Stable release	C++20 (ISO/IEC 14882:2020) / 15 December 2020; 3 years ago
Preview release	C++23 / 19 March 2023; 15 months ago
Typing discipline	Static, strong, nominative, partially inferred
OS	Cross-platform
Filename extensions	.C, .cc, .cpp, .cxx, .c++, .h, .H, .hh, .hpp, .hxx, .h++ .cppm, .ixx[1]
Website	isocpp.org

Major implementations

GCC, LLVM Clang, Microsoft Visual C++, Embarcadero C++Builder, Intel C++ Compiler, IBM XL C++, EDG

Influenced by

Ada, ALGOL 68,[2] BCPL,[3] C, CLU,[2] F#,[4] [note 1] ML, Mesa,[2] Modula-2,[2] Simula, Smalltalk[2]

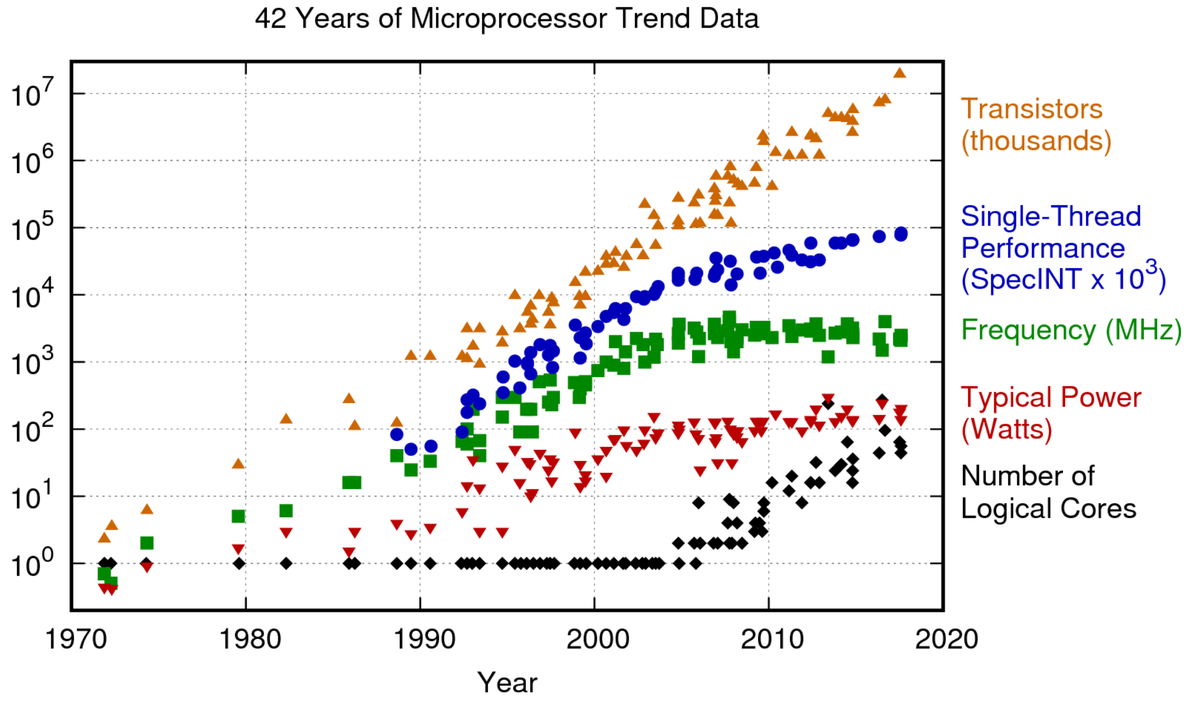
Influenced

Ada 95, C#,[5] C99, Carbon, Chapel,[6] Clojure,[7] D, Java,[8] JS++,[9] Lua,[10] Nim,[11] Objective-C++, Perl, PHP, Python,[12] Rust,[13] Seed7

C++ Programming at Wikibooks

Denard's scaling and multi-threading

- Allowed to go beyond single-core execution with reasonable effort
  - Multi-threading allows to schedule single modules on multiple cores and individual modules on one core in parallel
  - Following the trend that Moore's law is dead
    - "speed and capability of computers can be expected to double every two years, as a result of increases in the number of transistors a microchip can contain"
- Transition for the community was not easy
  - In a nutshell, don't use global variables
  - Transition was equally difficult as the transition from Fortran to C++



https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Dennard scaling

5 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

In semiconductor electronics, **Dennard scaling**, also known as **MOSFET scaling**, is a scaling law which states roughly that, as transistors get smaller, their power density stays constant, so that the power use stays in proportion with area; both voltage and current scale (downward) with length.[1][2] The law, originally formulated for MOSFETs, is based on a 1974 paper co-authored by Robert H. Dennard, after whom it is named.[3]

https://en.wikipedia.org/wiki/Dennard_scaling

GPU programming and portability

- GPU programming is different
  - Need to use special code constructs to exploit the benefits of GPUs
  - Essentially, if-statements have to be rethought
- 3 main GPU vendors: NVidia, AMD, Intel
  - All of them have proprietary toolkits for C++ (and other languages) to program for GPUs
    - NVidia: CUDA
    - AMD: ROKm
    - Intel: oneAPI - SYCL

```
#include <cuda_runtime.h>

__global__ void sqrt(int x, int* result) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    if (i == 0) {
        *result = sqrt(x);
    }
}

int main() {
    int x = 16;
    int result;

    // Set up CUDA
    cudaDeviceReset();
    cudaSetDevice(0);

    // Allocate memory on the GPU for the input and output
    int* d_x;
    cudaMalloc((void**)&d_x, sizeof(int));
    int* d_result;
    cudaMalloc((void**)&d_result, sizeof(int));

    // Copy the input to the GPU
    cudaMemcpy(d_x, &x, sizeof(int), cudaMemcpyHostToDevice);

    // Set up the grid and block dimensions for the kernel call
    dim3 gridDim(1, 1, 1);
    dim3 blockDim(1, 1, 1);

    // Call the kernel to calculate the square root on the GPU
    sqrt<<<gridDim, blockDim>>>(*d_x, d_result);

    // Synchronize the GPU and ensure the kernel has finished executing
    cudaDeviceSynchronize();

    // Copy the result from the GPU to the CPU
    cudaMemcpy(&result, d_result, sizeof(int), cudaMemcpyDeviceToHost);

    // Print the result
    printf("The square root of %d is %d\n", x, result);

    // Free the memory on the GPU and CPU
    cudaFree(d_x);
    cudaFree(d_result);

    return 0;
}
```

AI generated with codellama “Square root of number”

The way out: portability libraries

- Portability libraries
  - Allows to write algorithm once and then compile/ execute on GPUs of different vendors and CPUs
  - Promise to extend to future hardware platforms
  - CMS is using Alpaka
- HEP-CCE studied portability solutions for HEP

	CUDA	HIP	OpenMP Offload	Kokkos	dpc++ / SYCL	alpaka	std::par
NVidia GPU					codeplay and intel/llvm		nvc++
AMD GPU				feature complete for select GPUs	via openSYCL and intel/llvm	hip 4.0.1 / clang	
Intel GPU		CHIP-SPV early prototype		native and via OpenMP target offload		prototype	oneAPI::dpl
multicore CPU							g++ & tbb
FPGA						via SYCL	

Figure 1. Hardware support of portability layers. Dark green indicates full support, light green indicates partial support or that the project is still under development, and red indicates no support.

arxiv:2306.15869

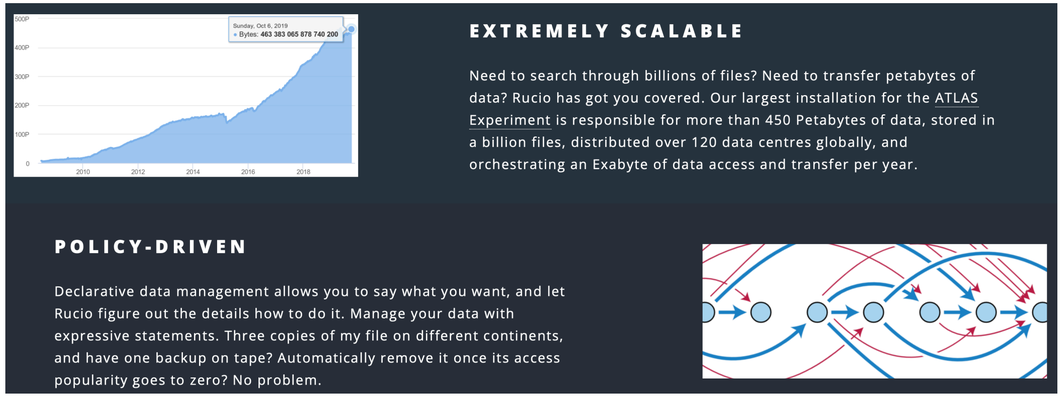
What else do you need to do your science?

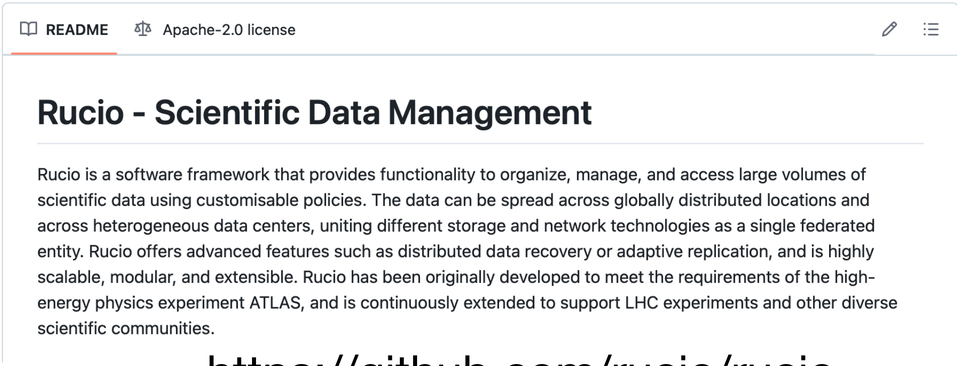
- Application software is not the only thing that makes HEP work
- You need infrastructure software
  - Data Management
  - Workflow Management
  - Large databases and distributed access
  - etc.
- We usually forget that also the infrastructure runs on software 😀

Data Management

- We are dealing with a lot of data (exabytes!) and a complex distributed computing infrastructure
  - We store data on disk (for immediate access) and tape (access takes longer but it is much cheaper, so we can store much more data)
- We use data management systems to catalog data and track the location of data (site A, B, C …)
  - Rucio is a prominent example of a data management system, developed by ATLAS and now used by many more experiments, for example CMS.
- Data management systems can also initiate transfers between sites

https://rucio.cern.ch/



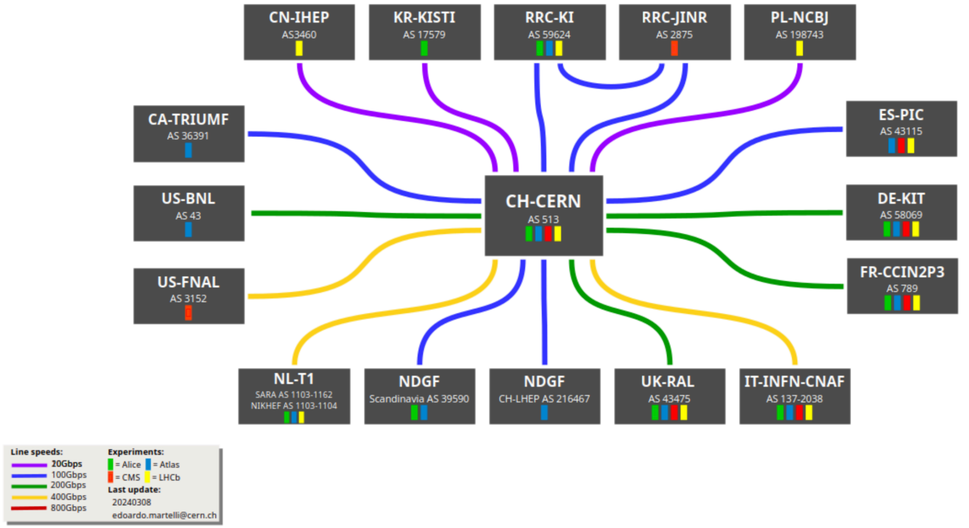


https://github.com/rucio/rucio

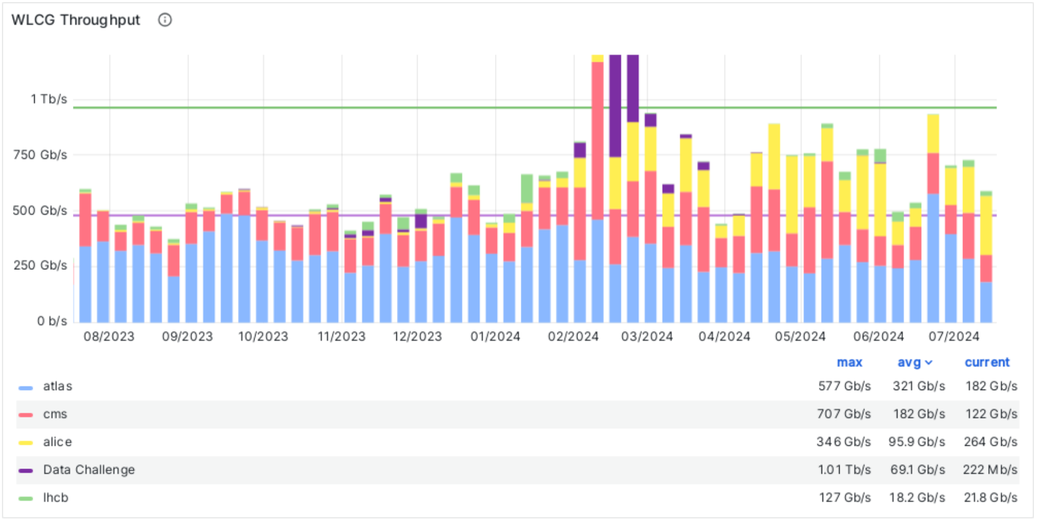
Network

- Strong network connections are the basis of a distributed data-intensive computing infrastructure
  - At the start of LHC: network was assumed to be the most unreliable part of the infrastructure, **it turned out to be the most reliable**
- LHC network as an example
  - LHC Optical Private Network (OPN)
    - Dedicated links between CERN and the Tier-1 centers
    - Quality of service and bandwidth guarantees
  - LHC Open Network Environment (ONE)
    - Virtual overlay network between all sites of WLCG (>100)
    - Allows for special security privileges (easy to get through firewalls)

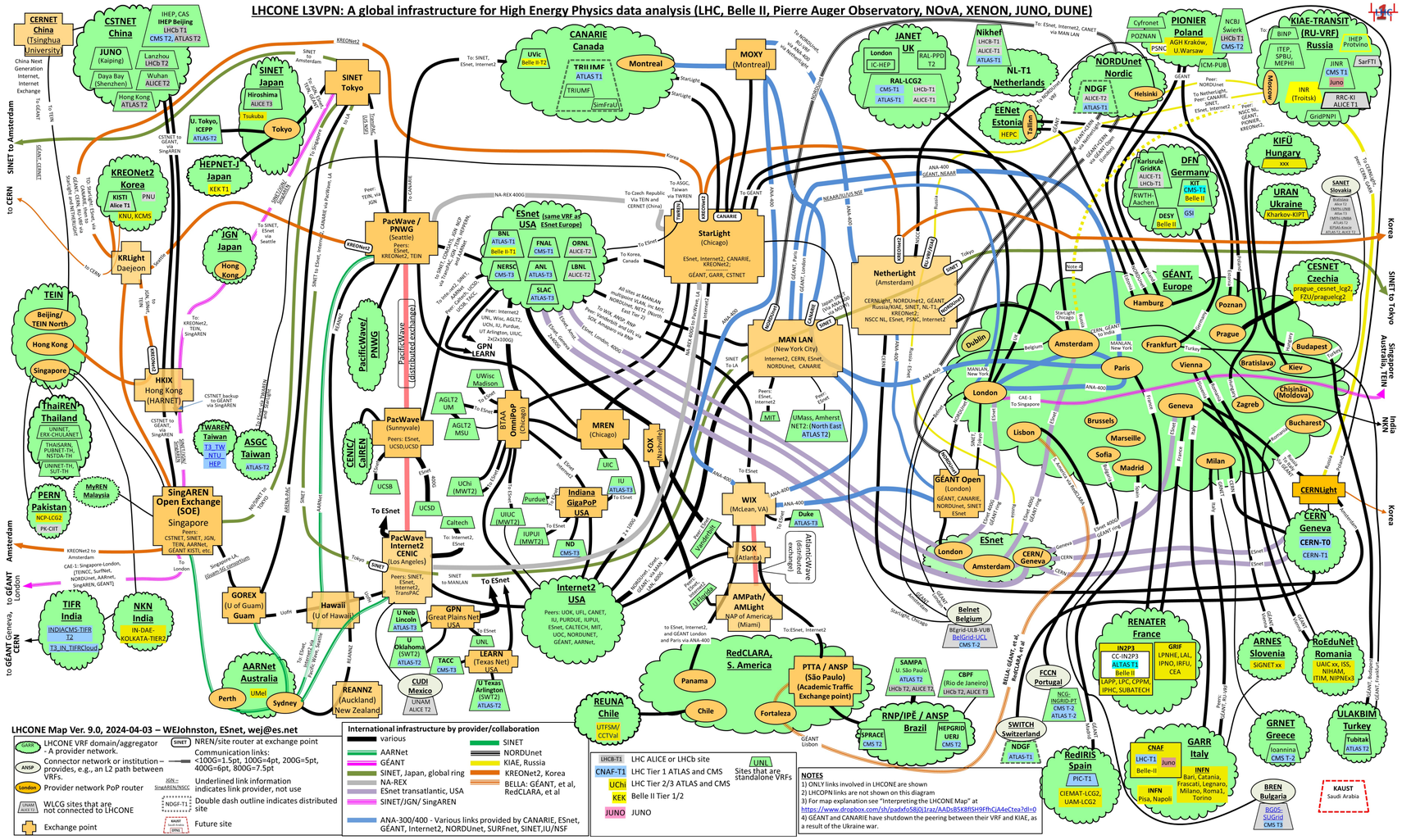
LHCOPN



https://lhcopn.web.cern.ch/



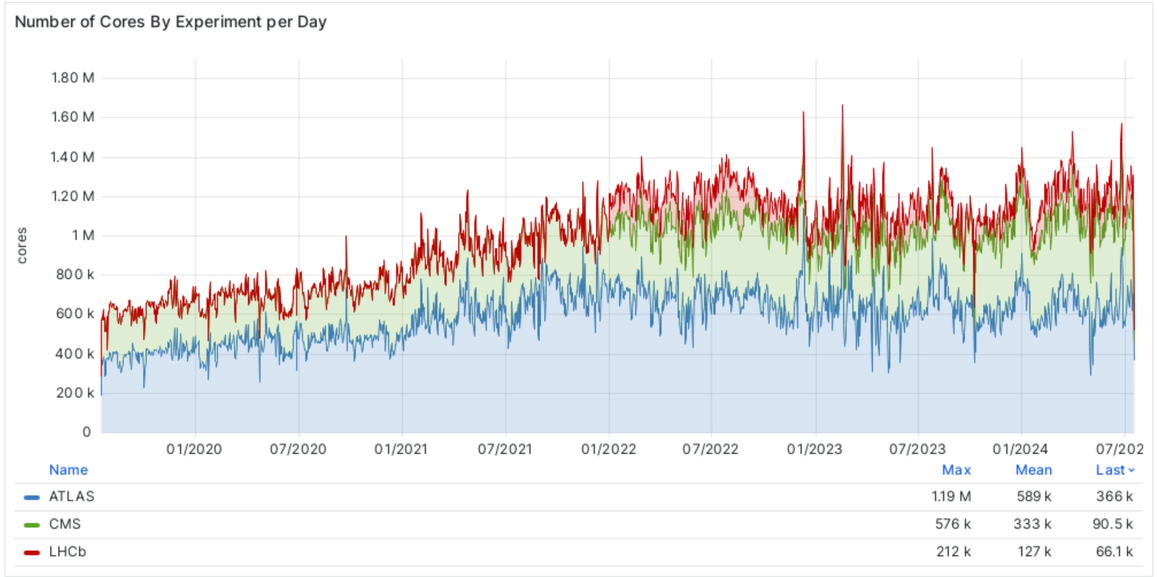
LHCONE



- Scientific networks make distributed scientific computing possible

Workflow Management

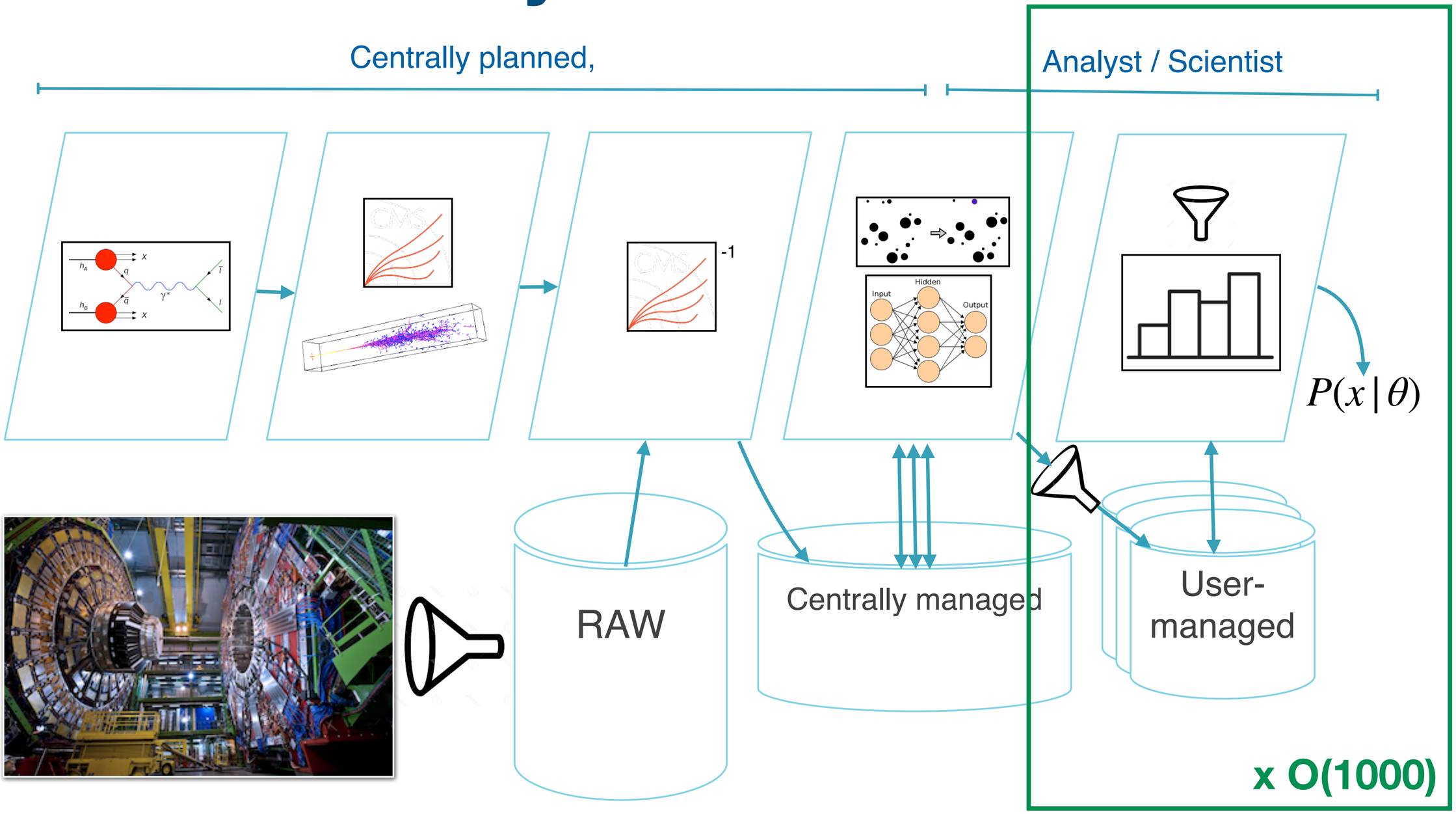
- Multi-dimensional optimization problem : orchestrating work accessing data and producing derived data
- Submission infrastructure software (like HTCondor) and workflow management software (Like Panda, Dirac, WMAgent) automate
  - Job creation
  - Job execution
  - Job monitoring and failure recovery
- WLCG currently uses up to 1.4M cores a day



Data analysis: the last step of the science process

Rule 7: Be a frugal analyst

Software is everywhere



Analysis - the Present

Responsible for:
- Data access
- Query planning
- Workflow scale-out

Analyst

ROOT Data Analysis Framework

HTCondor

slurm workload manager

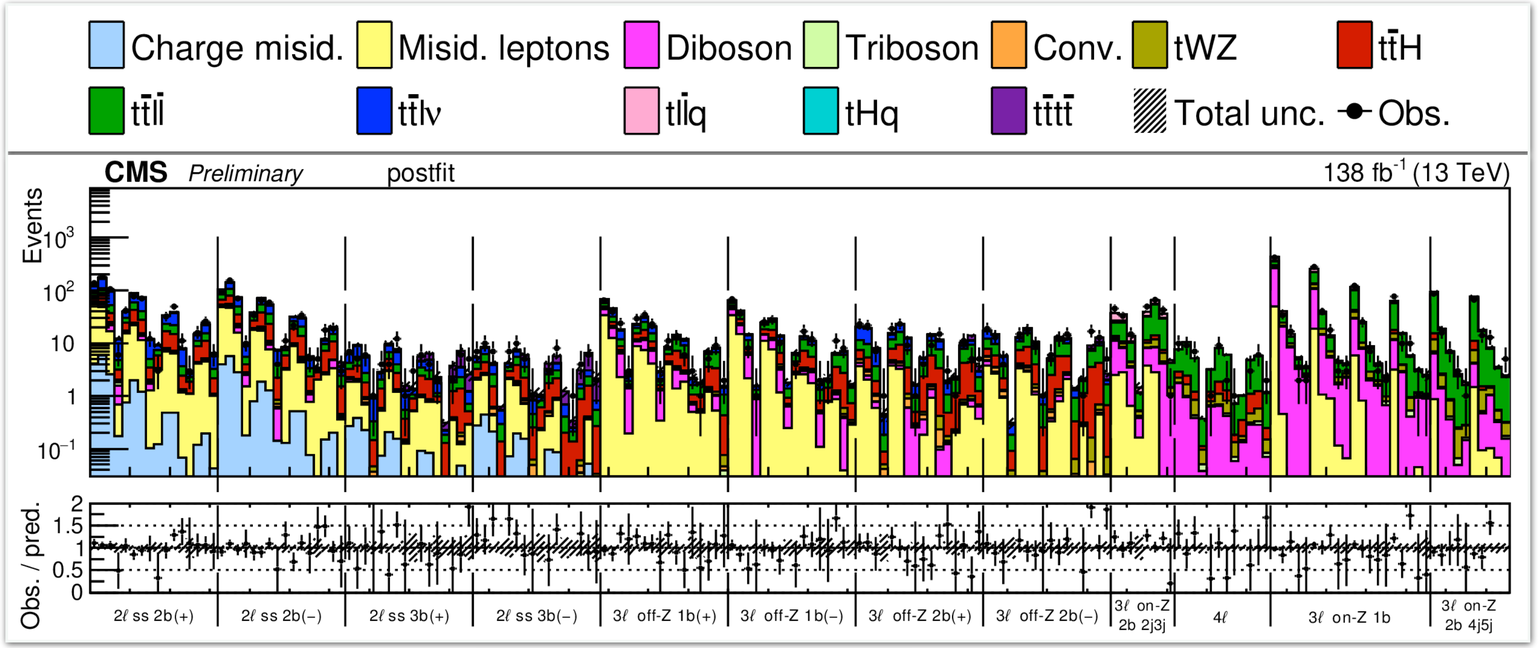
Analysis Frameworks
Software algorithms and tools

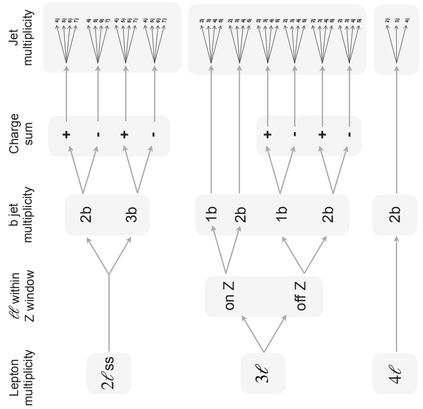
Datasets

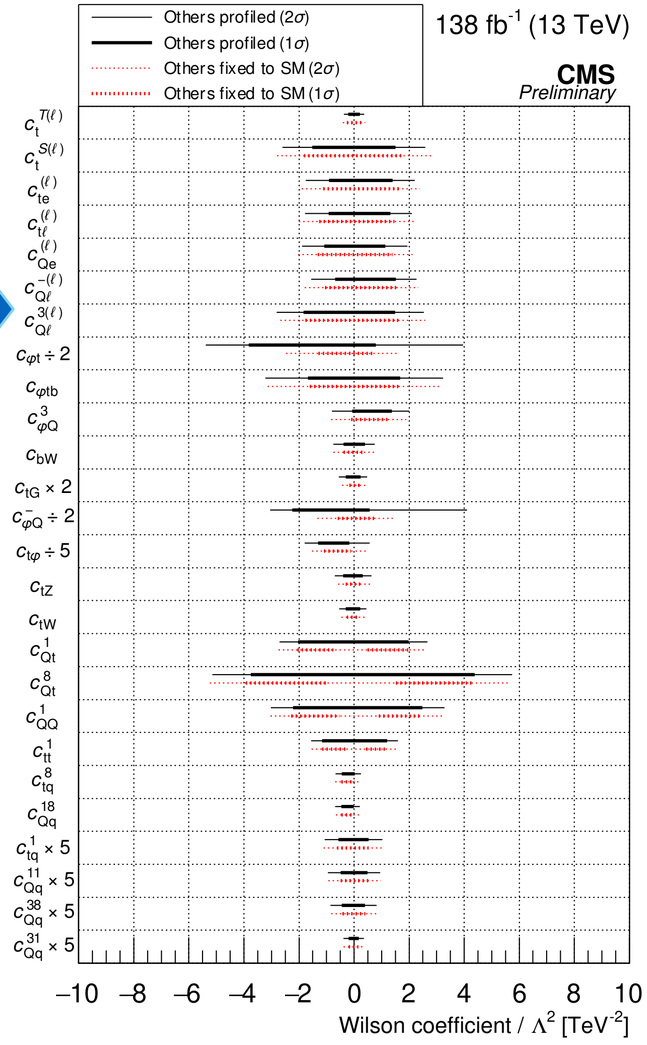
Analysis Resources
Hardware to scale out

Scaling Challenges

- Example: EFT search: many observables to constrain high-dim. parameter space



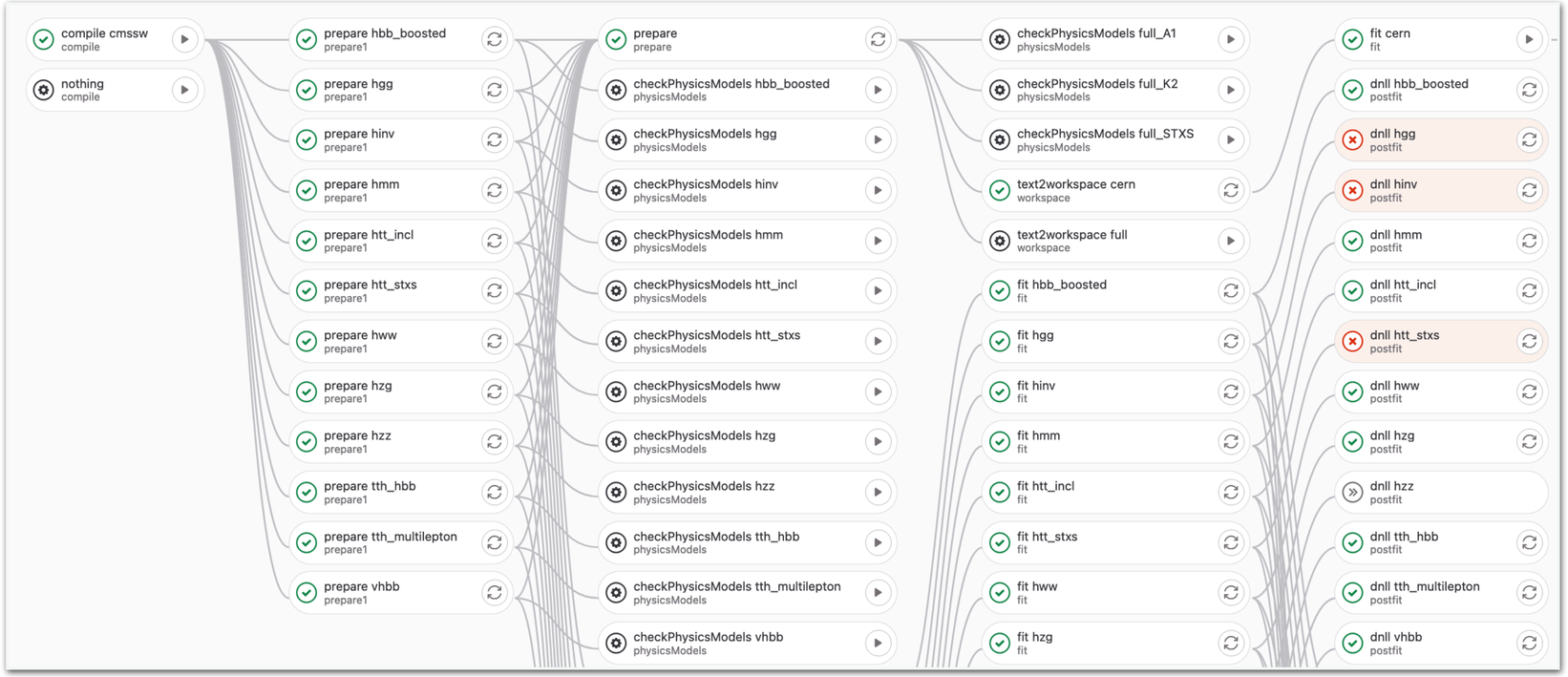


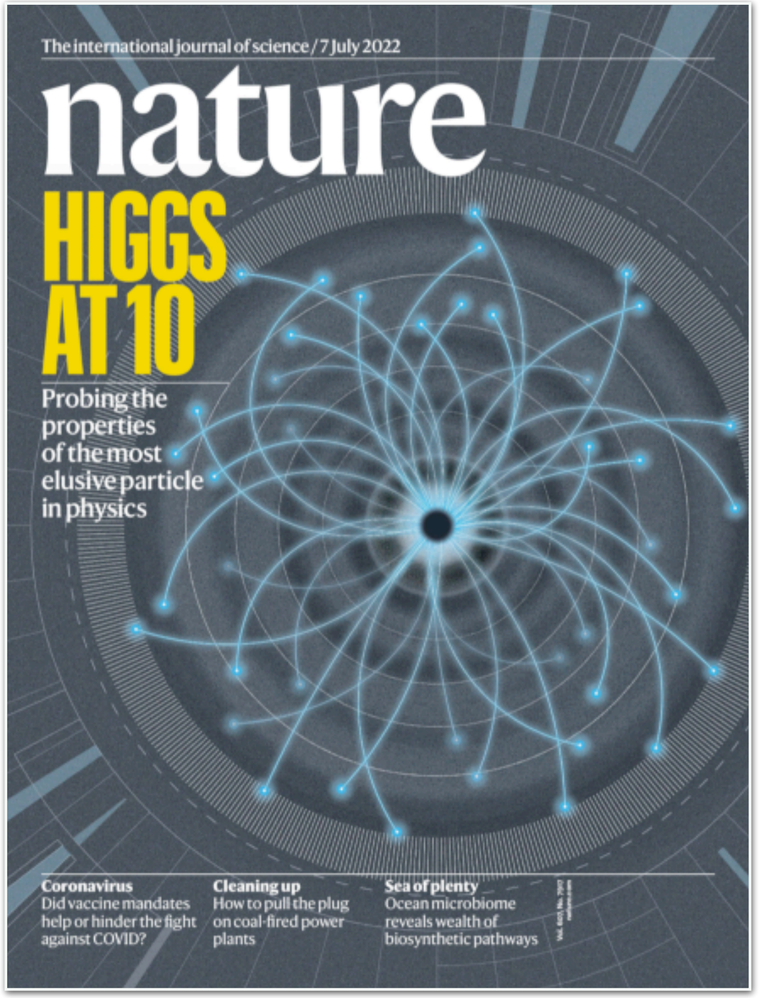


TOP-22-006

Scaling Challenges

- Higgs combination: bringing a dozen complex analysis descriptions together
  - Maximum likelihood fit: 10k parameters, 30h minimizations
  - Workflow management tools become a requirement

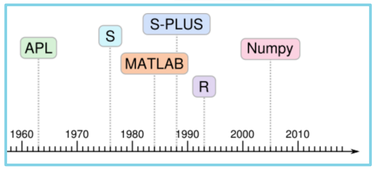


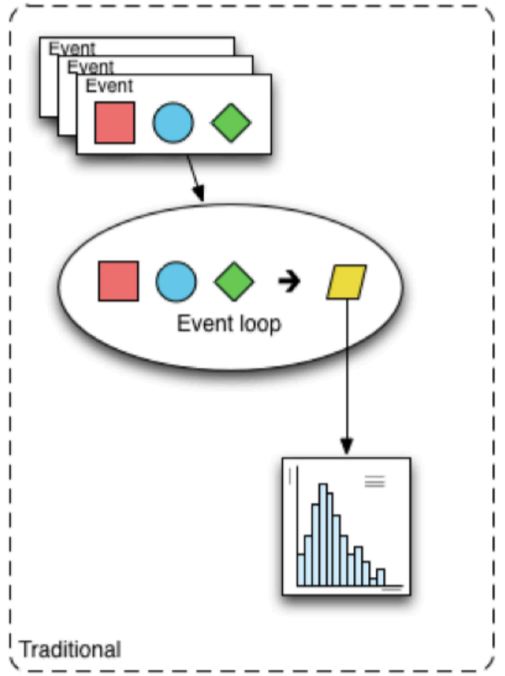


The paradigm shift

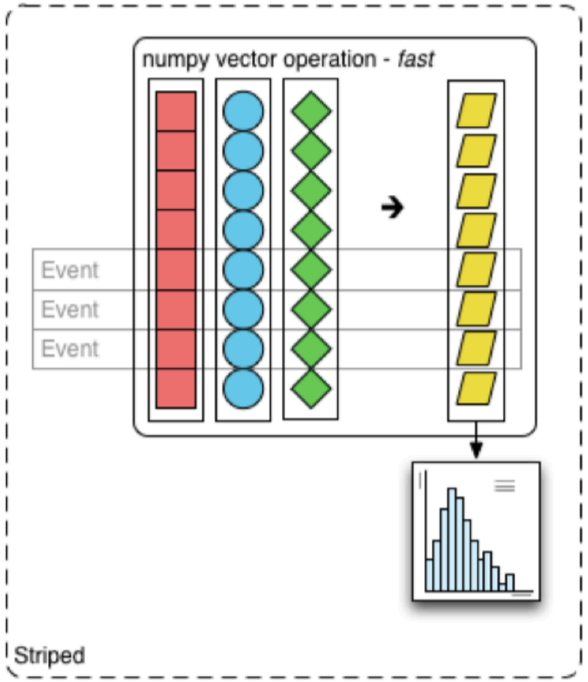
- Event loop analysis:
  - Load relevant values for a specific event into local variables
  - Evaluate several expressions
  - Store derived values
  - Repeat (explicit outer loop)

Array programming is not new!
APL demo on YouTube





- Columnar analysis:
  - Load relevant values for many events into contiguous arrays
  - Evaluate several array programming expressions
  - Implicit inner loops
  - Store derived values



Awkward Array: JSON-like data, NumPy-like idioms

Jim Pivarski, Ianna Osborne, Peter Elmer

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

```
output = []
for sublist in python_objects:
    tmp1 = []
    for record in sublist:
        tmp2 = []
        for number in record["y"][1:]:
            tmp2.append(np.square(number))
        tmp1.append(tmp2)
    output.append(tmp1)
```

2.3 minutes to run (22 GB footprint)

```
output = np.square(array["y", ..., 1:])
```

```
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

4.6 seconds to run (2.1 GB footprint)

(single-threaded on a 2.2 GHz processor with a dataset 10 million times larger than the one shown)

SciPy2020 awkward presentation

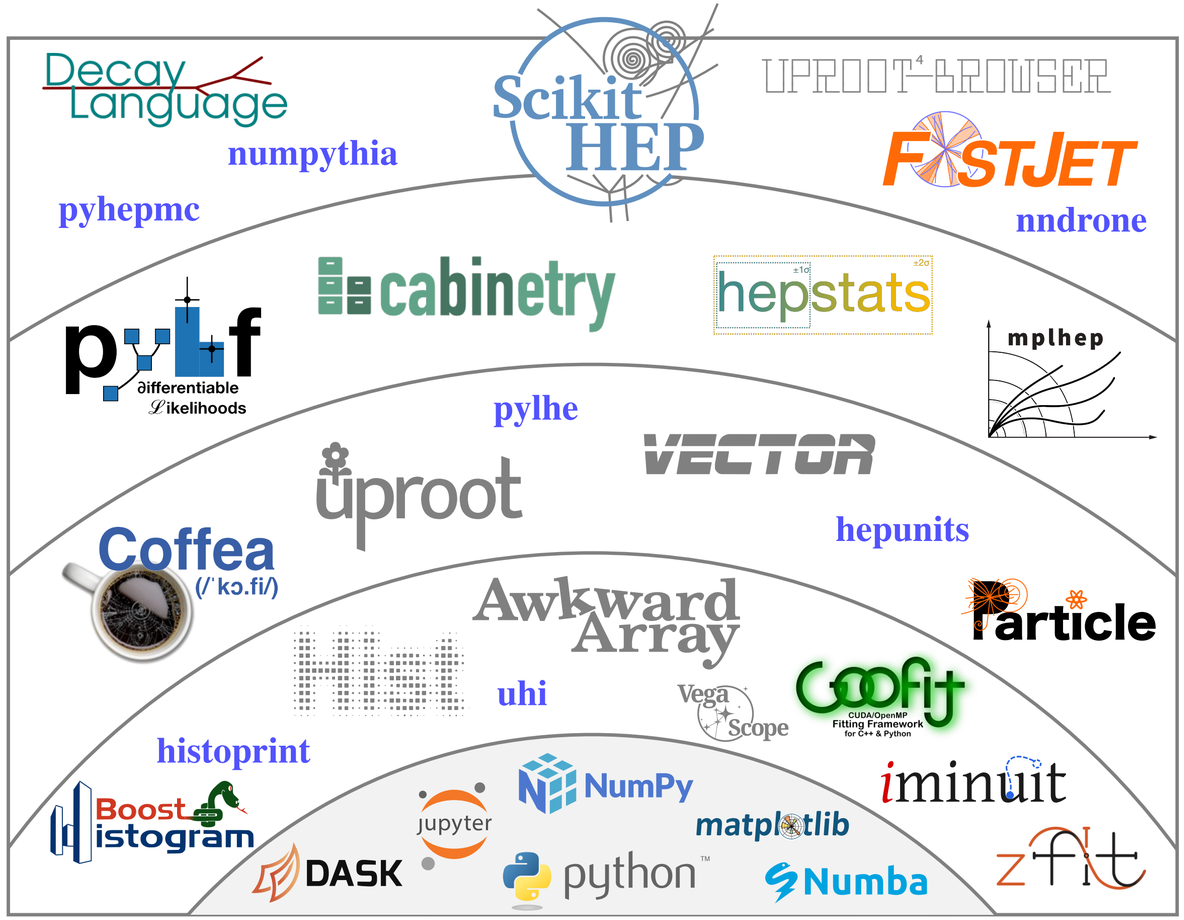
The other paradigm shift

- From vertically-integrated solution to ecosystem

Experiment-specific and analysis frameworks

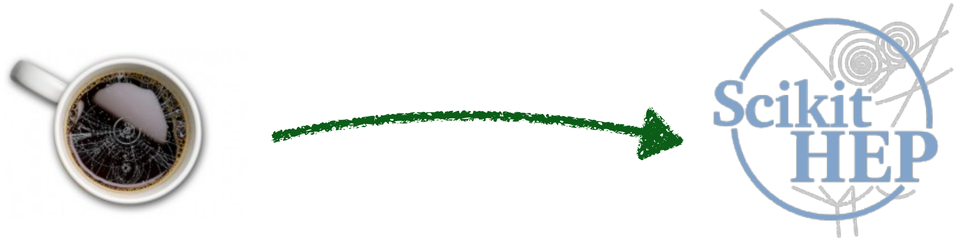
ROOT Data Analysis Framework

https://root.cern.ch/

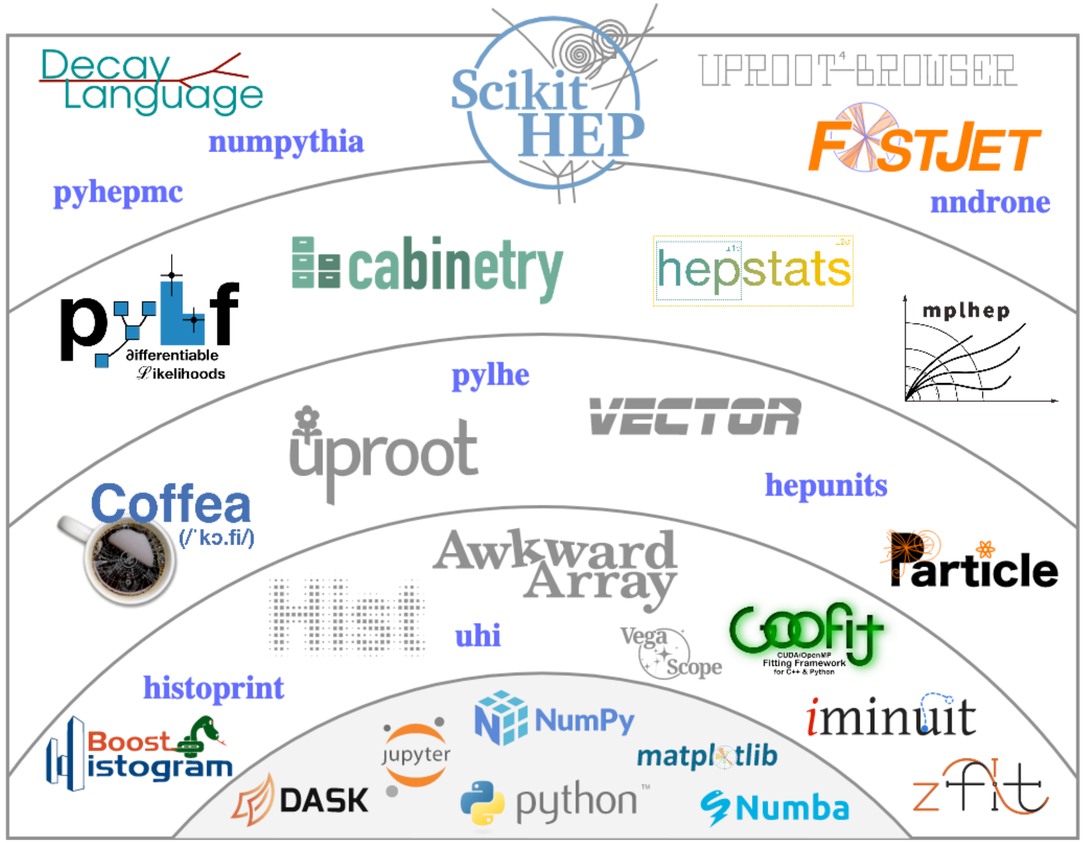


Coffea project

- A user interface to columnar analysis
  - Optimized array programming kernels build an expressive and performant language
  - Seamless integration with ML tools due to shared interface
- An incubator for rapid prototyping
  - Fill in missing pieces of ecosystem
  - Good abstractions are factored out
- A minimum viable product
  - Already used in several CMS publications
  - In use by ATLAS, ProtoDUNE collaborators
  - Early feedback builds ecosystem roadmap
    - Vibrant contributor community



https://coffeateam.github.io/coffea/

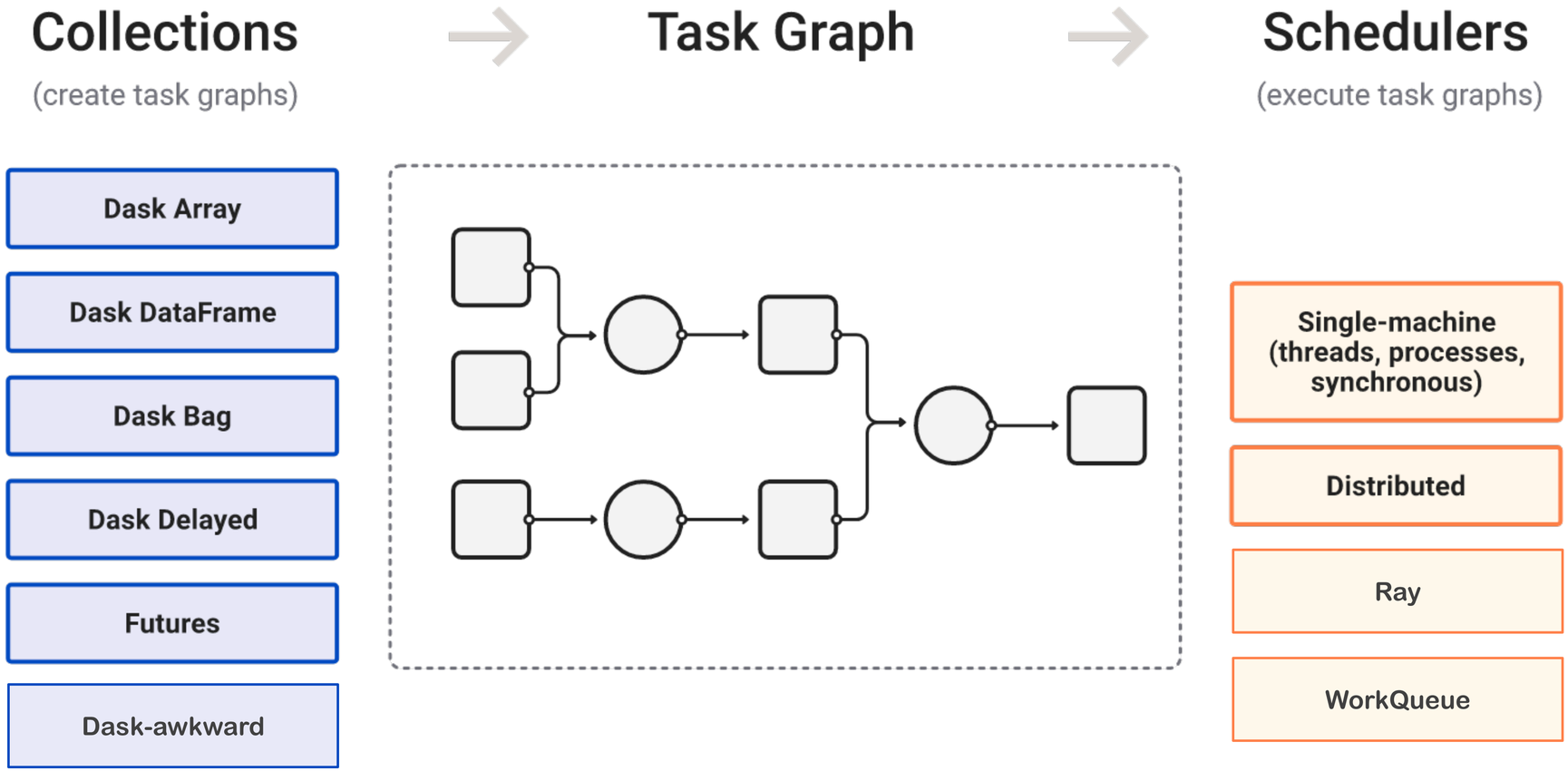


Scaling analyses

- Easy transition from local to distributed execution
  - Ideally no user code change

dask

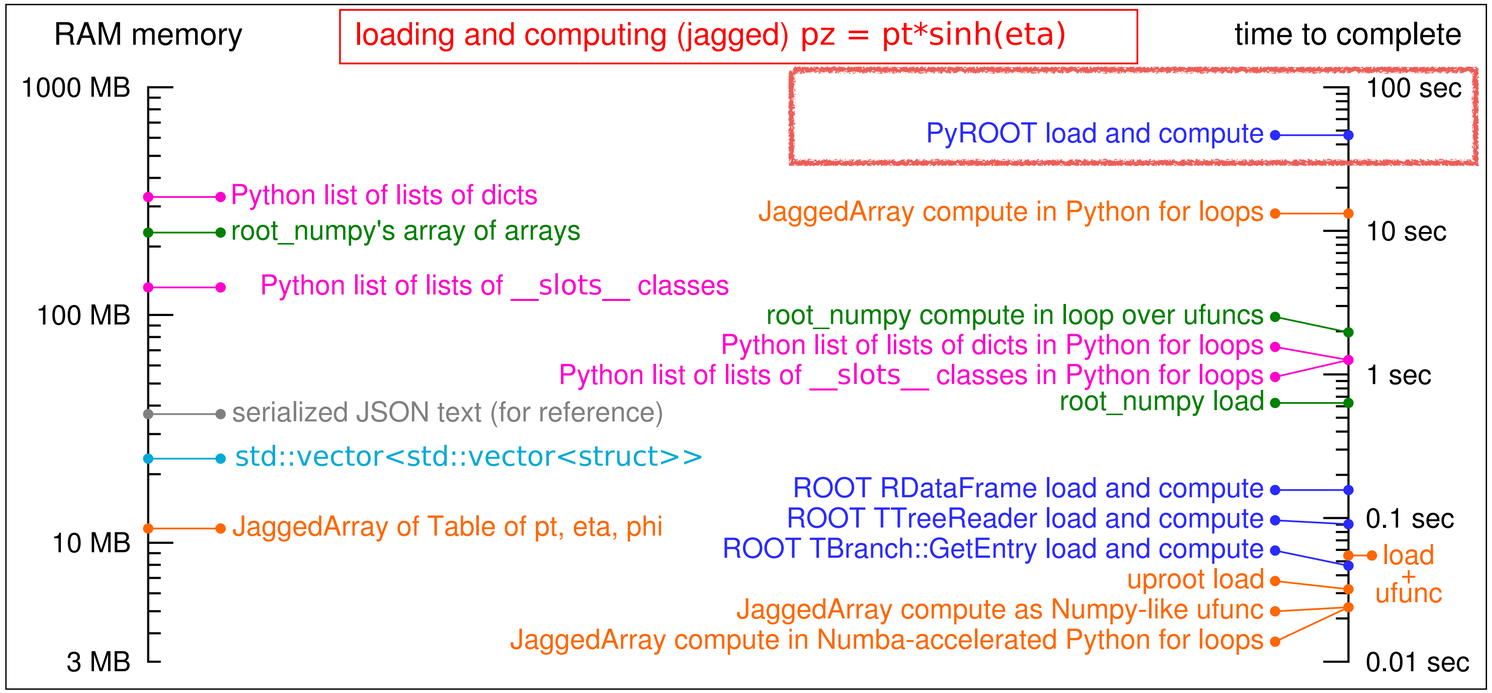
https://www.dask.org/



Batch queues (HTCondor, Slurm) are now resource provisioning

Performance

- For library designers, important to know when we are fast enough
  - µs to ms per event



N. Manganelli

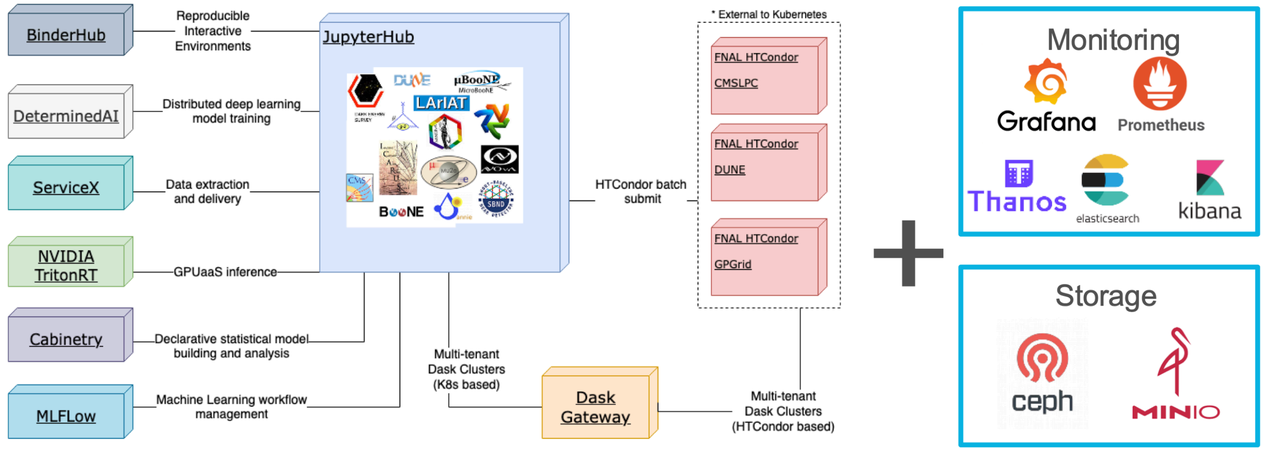
J. Pivarski

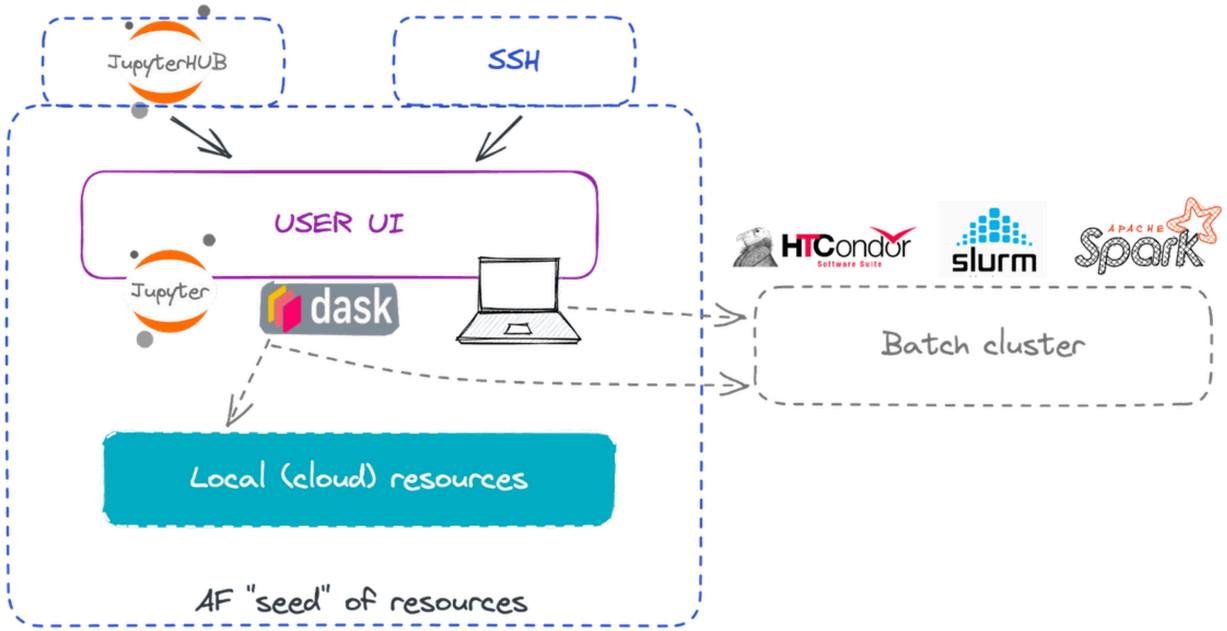
Benchmarking the code and coming out fastest is fantastic

- Factor 3x* is small compared to the O(1000)-O(10000) improvement RDF/coffea have against TTree::Draw-based frameworks (I know of several)

Facility integration

- Data delivery is the bottleneck for columnar analysis at scale
  - True also for AI/ML workloads
- Analysis facility integration would:
  - Reduce manual user data curation
  - Offload expensive algorithms to accelerators
  - Save compute and storage resources
- …to maintain fast time to insight physics
- We have the playgrounds to realize this vision
  - Snowmass contrib: arxiv:2203.10161
  - HSF Analysis Facilities white paper



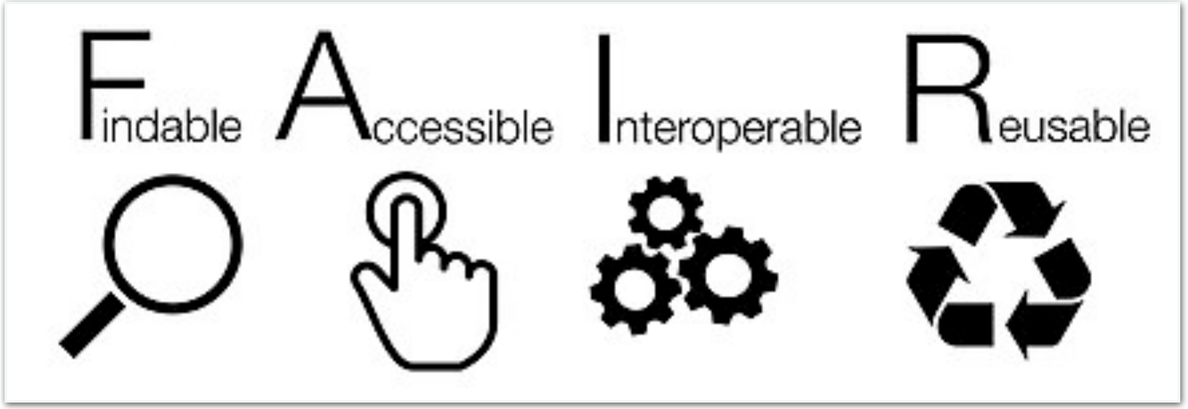


Workforce Development

Culture and Training

Particle Physics Software Culture

- Analysis software is critical for HEP
  - But new collaborators can struggle
    - Chase down requirements / “recipes”
    - Join group with mature framework / toolset
  - Is our data and metadata FAIR?
    - Software sometimes viewed as competitive advantage



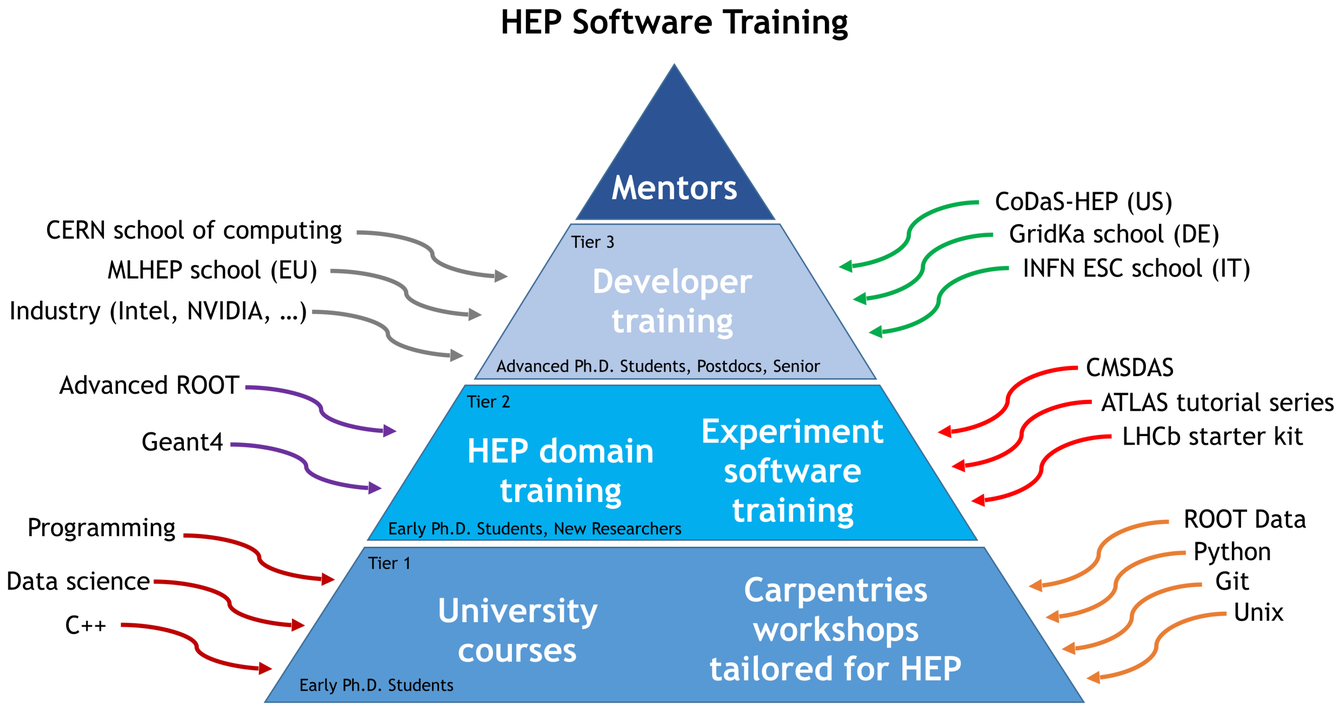
- Need training & mentorship pipeline
  - In same sense as pixels, calorimeters, …software detector?
  - Career paths: Traditional track, Research Software Engineer (US-RSE), ?

- In the right direction: IRIS-HEP, HEP Software Foundation
  - Soon: DPF Coordination Panel for Software and Computing

Training

https://hepsoftwarefoundation.org/workinggroups/training.html

HSF | Working Groups | Activities | Meetings | Communication | Projects & Support | About



Computational and Data Science for High Energy Physics (CoDaS-HEP) Summer School

IRIS-HEP team members help organize events, develop training material and participate as lecturers at all levels of this training vision. We organize and run the annual Computational and Data Science for High Energy Physics (CoDaS-HEP) summer school at Princeton and provide additional professional development for the most advanced students and postdocs by supporting connections with mentors through the IRIS-HEP Fellows Program. Integrated with this is a robust outreach program to engage local communities. This is essential to train STEM students in software at K-12 level by partnering with their teachers. These steps create a workforce capable of solving software challenges in general and those of HEP in particular.

Group photo of participants of the 3rd summer school on tools, techniques and methods for Computational and Data Science for High Energy Physics (CoDaS-HEP) at Princeton University on July 22-26, 2019.

Best Practices

IRIS-HEP continues to document, disseminate, and work towards community adoption of the best practices (from HEP and beyond) in the areas of software sustainability, including topics in software engineering, data/software preservation, and reproducibility. Of particular importance are best practices surrounding the modernization of the software development process for scientists.

https://iris-hep.org/ssc.html

HSF Training

The HSF Training Working Group aims to help the research community to provide training in the computing skills needed for researchers to produce high quality and sustainable software. The group works with experiment training groups, HEP initiatives (such as IRIS-HEP and FIRST-HEP) and organisations like Software Carpentry to coordinate training activities.

The group aims to develop a training program that can be pursued by researchers to achieve the level of required knowledge. This ranges from basic core software skills needed by everyone to the advanced training required by specialists in software and computing.

Join an event!
Discover new topics together with mentors and peers!

Self study!
Learn at your own pace. No matter if you want to get a quick overview or dive in the details, this is for you!

Our mission

The long term sustainability of the research software ecosystem is important for HEP as HL-LHC and other facilities of the 2020s will be relevant through at least the 2030s. Meeting this challenge requires a workforce with a combination of HEP domain knowledge and advanced software skills.

The software skills required fall into two groups. Nearly all researchers need basic programming skills (Python, C++), basic software engineering skills (Unix, git/GitHub/GitLab, continuous integration, performance evaluation), and skills in the core data tools in HEP (e.g., the ROOT data format and analysis framework).

More advanced training is then needed (with domain examples!) in parallel programming, efficient software implementations, performance optimization, and machine learning and data science tools. A workforce trained in this range of software skills is the critical ingredient from which the solutions to the computing challenges can grow.

The end

Final Remarks

- Software and Computing Infrastructure is an integral part of the scientific process
- HEP drove scientific computing for many years, nowadays we are one of many (computational biology, earth and climate sciences, …)
- Distributed scientific computing is evolving from an x86-only architecture to a heterogeneous landscape encompassing scientific data centers, commercial clouds, HPC centers using CPUs, GPUs, and whatever the future will bring
- The carbon footprint of scientific computing is non-negligible! We need to:
  - Rule 4: Choose your computing facility
  - Rule 5: Choose your hardware carefully
  - Rule 6: Increase efficiency of the code
  - Rule 7: Be a frugal analyst
- Learn how to write GPU code! Learn how to do data science efficiently using the pythonic eco system!
  - Being a domain scientist also means being a data scientist and being knowledgable in software and computing!

Acknowledgments

- Thanks to all my colleagues, collaborators, community members, friends for inspiration and materials
- Thanks to the funding agencies who don’t forget that software & computing is an integral part of the science process
- Special thanks to Nick Smith of Fermilab for many slides and discussions.

Backups

Details for carbon footprint calculation

- Details of the calculation: https://docs.google.com/spreadsheets/d/1o3hgAg2-veJ5T7vzBa9JUj5tGRvY2ZW3AsX0cNGw1wE/edit?usp=sharing
- CPU
  - AMD EPYC 7452 32 physical core 64 thread dual cpu system
  - 12 HS23 per hyper threaded core (HTCore) (averaged two entries from https://w3.hepix.org/benchmarking/scores_HS23.html)
  - Energy assumption from FNAL data center experience: 1.8 kW
  - 1.8 kW / 128 HTCores / 12 HS23 *1000 = 1.17 W/HS23
- Disk system
  - 24 disk JBOB system, 20 TB disks, RaidZ2, 410 TB usable
  - Energy consumption from FNAL data center experience: 250 W
  - 250 W / 410 TB = 0.61 W/TB
- Chicago data center
  - PuE 1.6
  - 433 gCO2/kWh (https://app.electricitymaps.com/zone/US-MIDA-PJM) = 0.433 tCO2 / MWh
- Produce 1B MC events at 1098 HS23s per event:
  - 1E9 events * 1098 HS23s/event * 1.17 W/HS23 * 1.6 PuE / 3600 / 1E6 * 0.433 tCO2 / MWh = 248 tCO2
- Reconstruct 1B data events at 333 HS23s per event:
  - 1E9 events * 333 HS23s/event * 1.17 W/HS23 * 1.6 PuE / 3600 / 1E6 * 0.433 tCO2 / MWh = 75 tCO2
- Store 1 B MC events and 1 B data events at 1.5 kB/event for 1 year
  - 2E9 events * 1.5 kB/event * 365 * 24* 3600s * 1.6 PuE * 0.61 W/TB /1E12 / 3600 * 0.433 tCO2 / MWh = 11 tCO2