Challenges and Opportunities for Tools in a Changing HPC Landscape

Jean-Baptiste BESNARD
ParaTools SAS

Changing HPC Landscape

Challenges and Opportunities for Tools

Is HPC at the Crossroads ?

HPC is changing…

- Generative AI is a worldwide use-case of HPC
- We may soon witness the rise practical of quantum computers
- Increasing pressure from hardware (variety, FP16, GPUs, …)
- Rise of higher level programming languages
- …

Is HPC at the Crossroads ?

HPC is still the same …

- Parallelism concepts are easily transposable (e.g. xCCL)
- MPI is the de-facto standard for distributed parallelism
- Most languages and codes are native C++/Fortran/C:
  - Some codes have inertia (mission-critical &-/ complexity)
  - Often supercomputers are used by super-specialists (knowledge-gap)
- There is a strong coherence in HPC environments:
  - Batch manager (Slurm, PBS, …)
  - MPI + OMP + Y
  - Strong tool incentive (MPI and OMP)

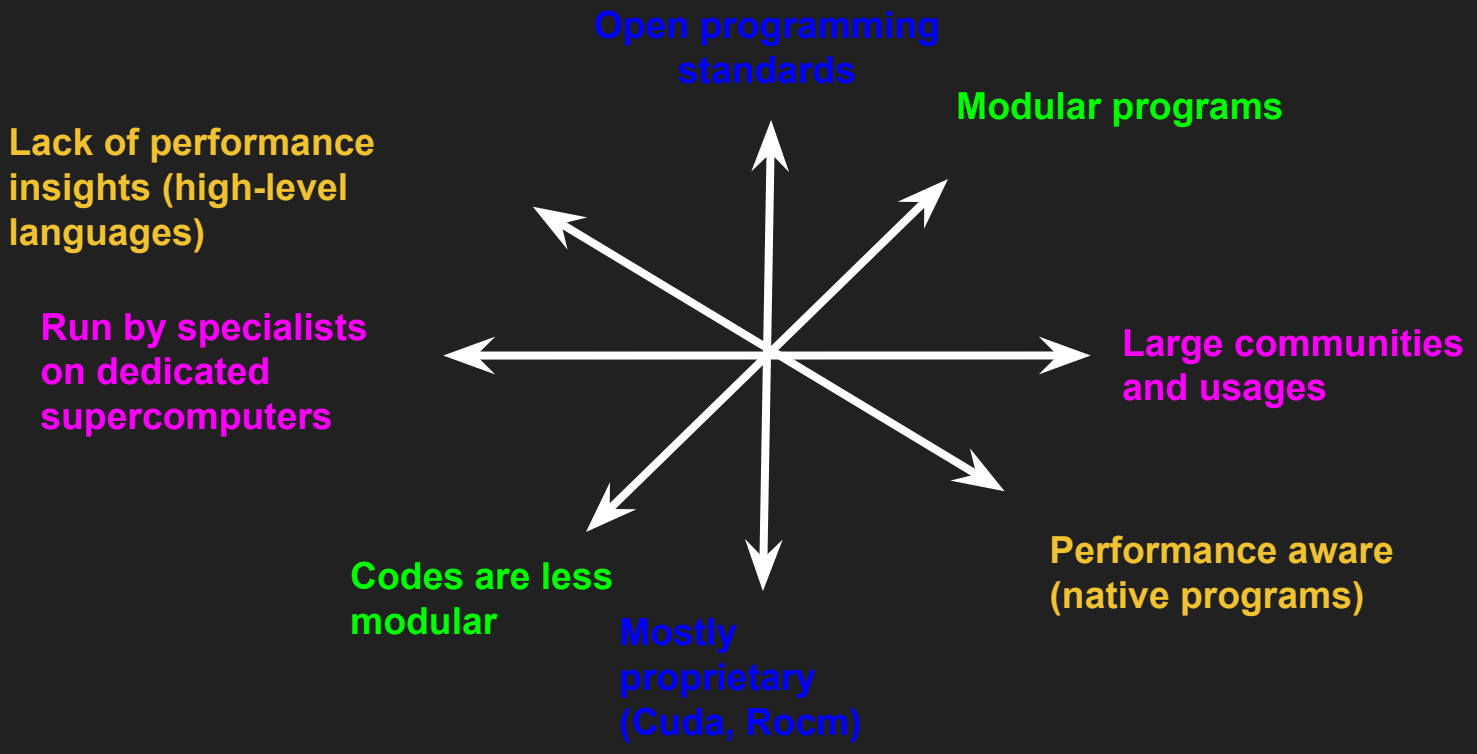
Usage of Compute in AI versus HPC

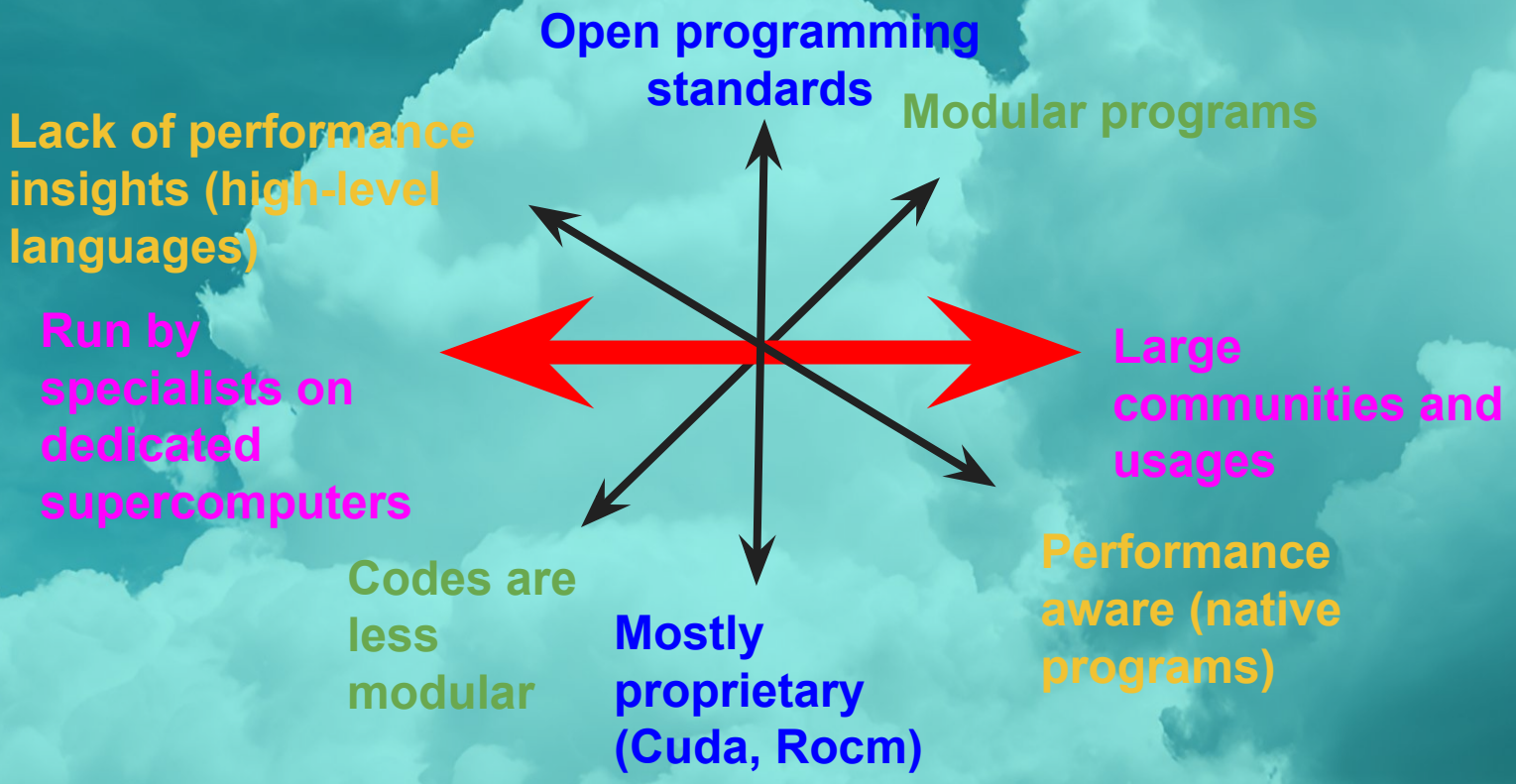
Aspect	Machine Learning	High-Performance Computing (HPC)
Approach	Practical (almost empirical) approach of computing	Now more in the “production era” in feed-forward
Languages	High-Level Languages calling optimized kernels	Uses compiled languages C,C++,Fortran
Backend Optimization	Calls optimized kernels in back-end (BLAS)	Running bare-metal code
Distributed	For training and less for inference	Most of the time
Structure	Structured around software libraries	Structured by standardized interfaces (MPI, OMP)
Resource Allocation	More inclined to leverage “cloud” approaches	On premise (own supercomputer)
Entry Cost	Low entry cost (possible with single GPU)	Reserved to those owning supercomputers
Community	Millions of usages (scripts) and a diverse community	Large codes driven by specialized programmers

Both Weaknesses **and** Strenghts

Aspect	Machine Learning (AI stuff)	High-Performance Computing (HPC)
Strenghts	• Modular programs extensive use of libraries and bindings • Large communities and usages (can be done on a single GPU)	• Performance aware (native programs) • Open programming standards
Weaknesses	• Lack of performance insights/flexibility (high-level languages) • Mostly proprietary (Cuda, Rocm)	• Codes are less modular and tend to reimplement things • Run by specialists on dedicated supercomputers

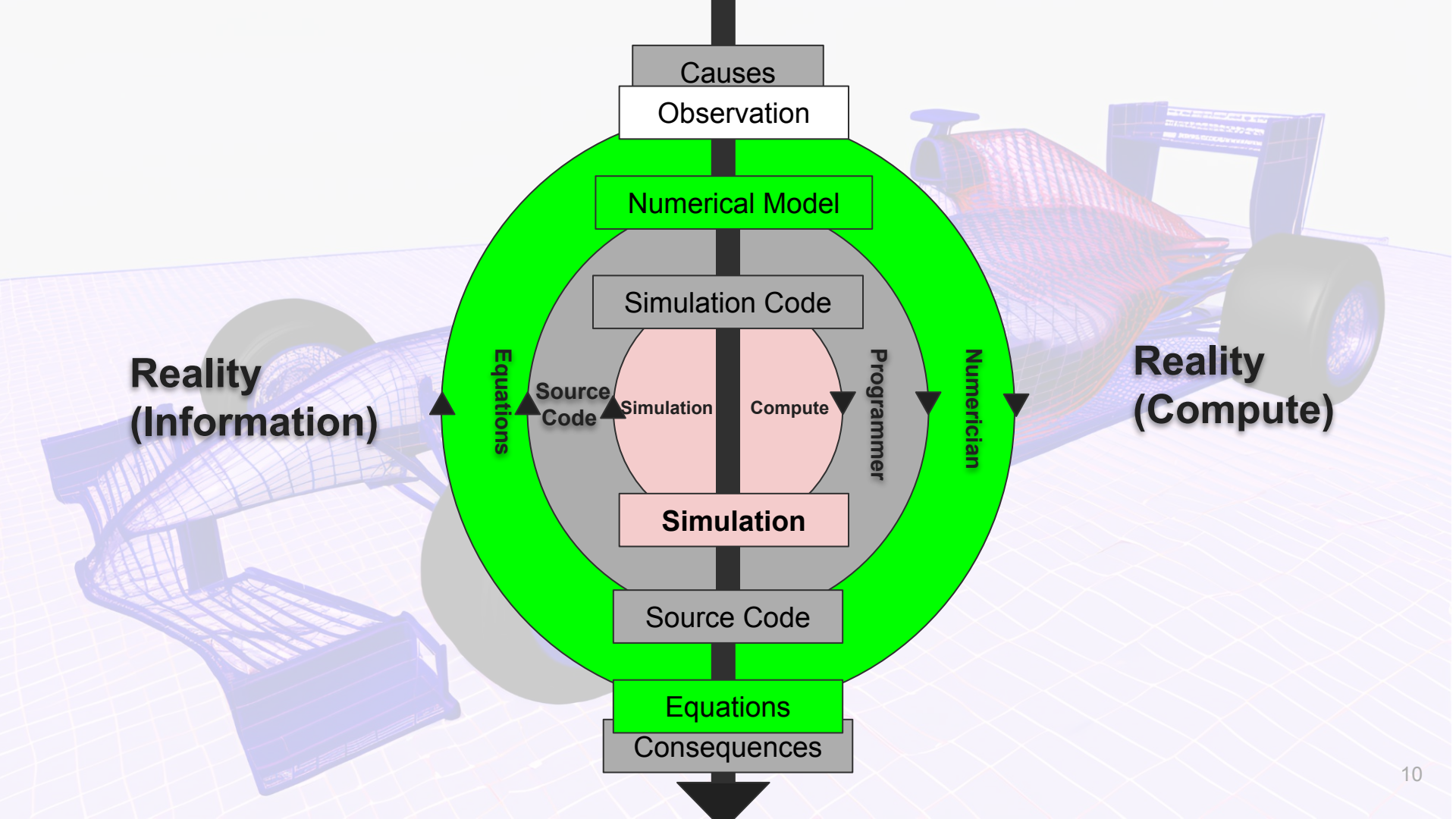
Future is somewhere inbetween these trade-offs

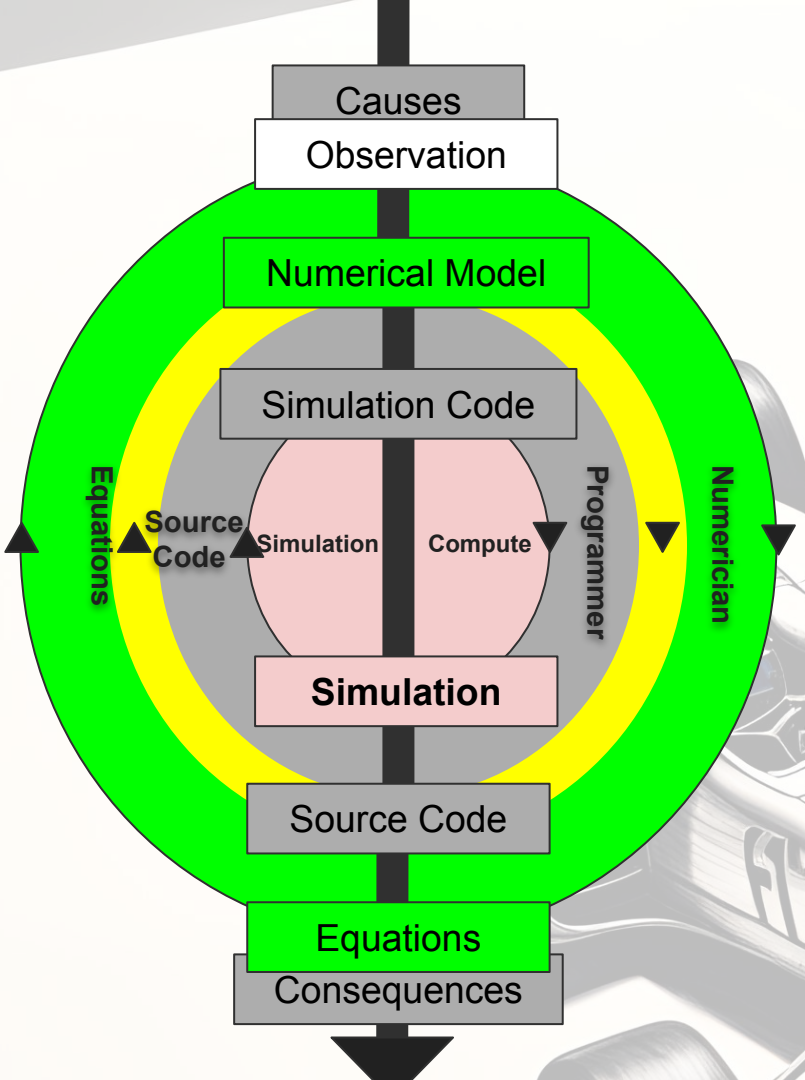



Open programming standards
Modular programs
Lack of performance insights (high-level languages)
Run by specialists on dedicated supercomputers
Large communities and usages
Codes are less modular
Mostly proprietary (Cuda, Rocm)
Performance aware (native programs)

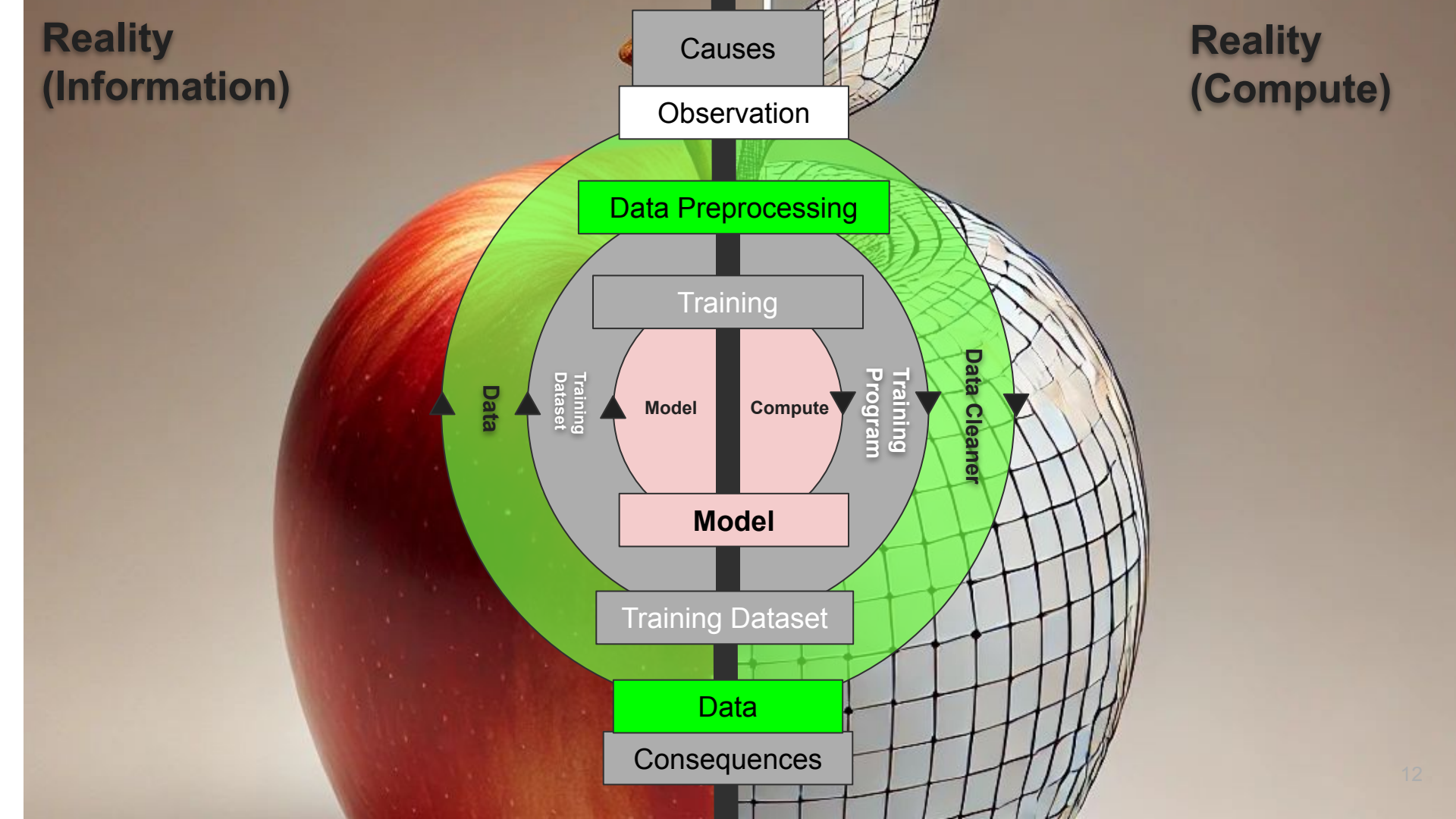
Specialists on dedicated supercomputers ⟷ **Large communities and usages**

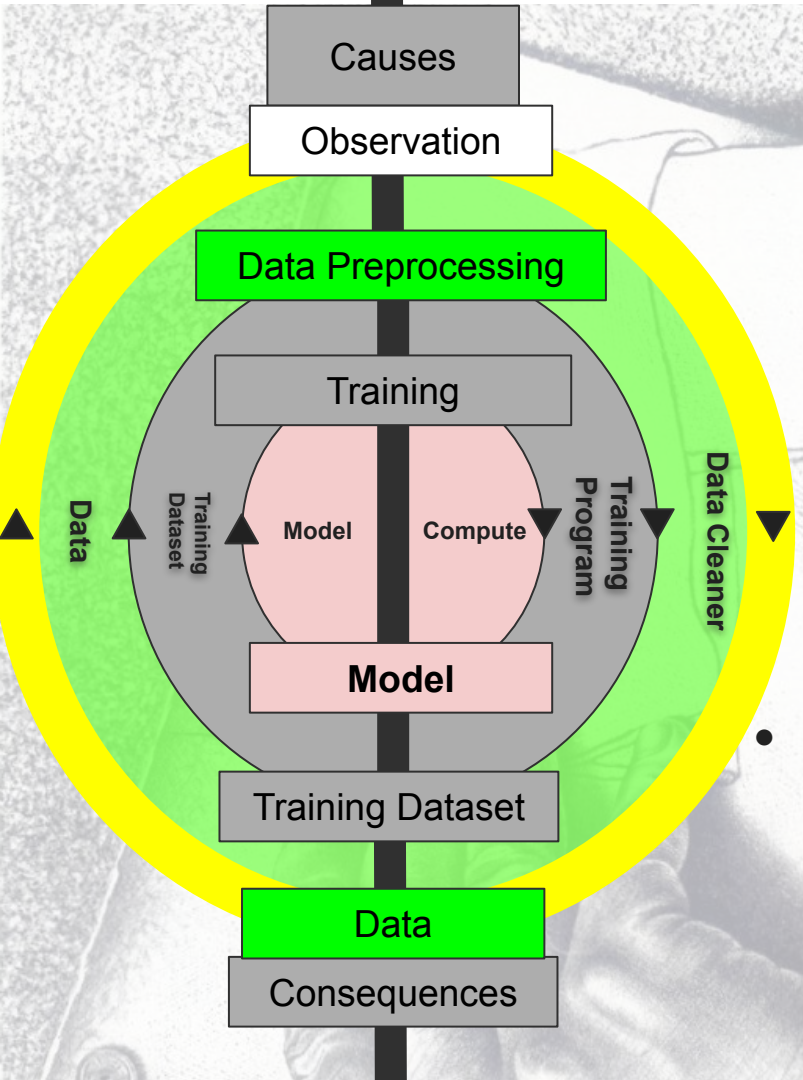
AI Supercomputers are used differently leading to **wider compute and data spectrums**.


Causes
Observation
Numerical Model
Simulation Code
Reality (Information)
Equations
Source Code
Simulation
Compute
Programmer
Numerician
Reality (Compute)
Simulation
Source Code
Equations
Consequences



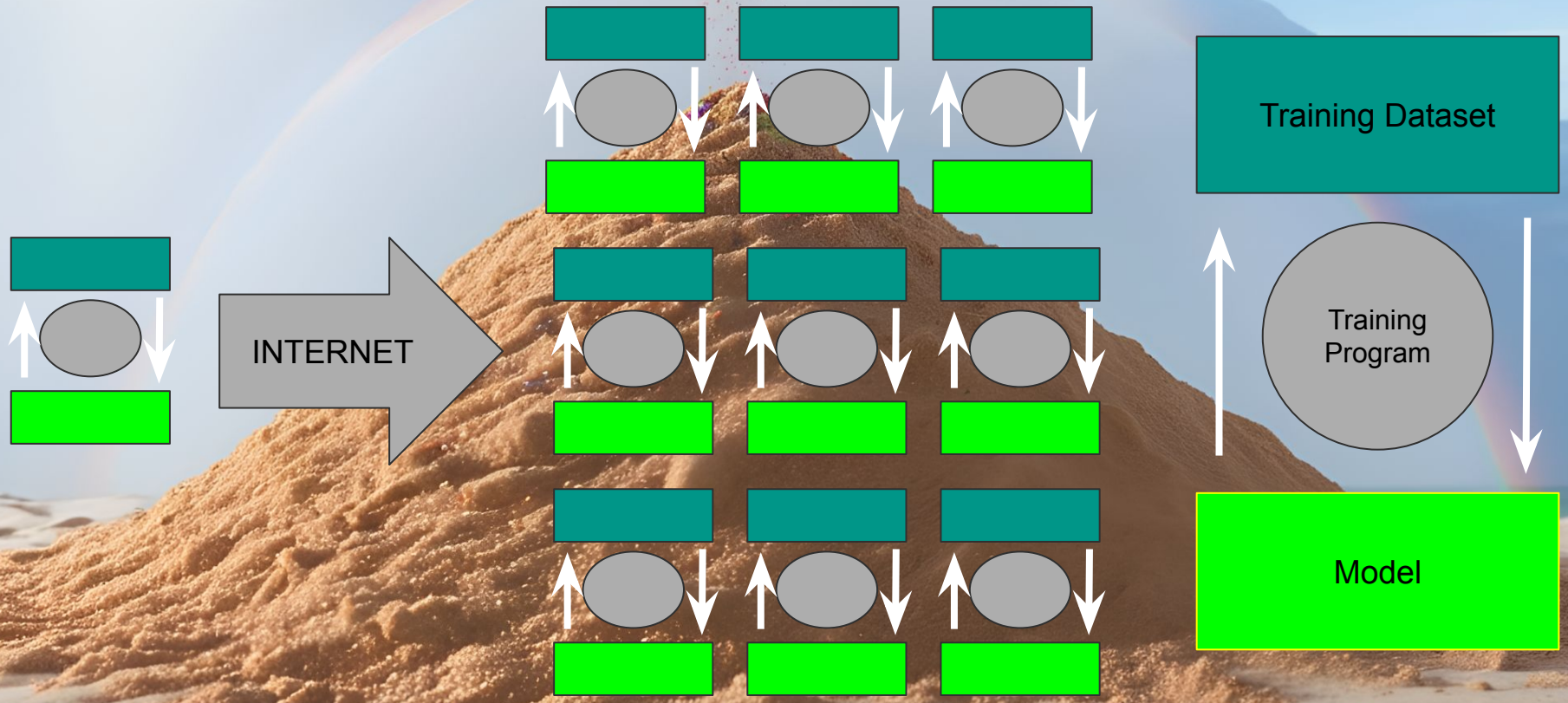
- **Tools are bridges** between abstractions – making abstract states visible to other levels of representation.
- They provide an alternative way of looking at the simulation code to support several values:
  - Correct usage of HW resources
  - Correctness of the code
  - Expressivity of the program
  - Improved performance


Reality
(Information)
Reality
(Compute)
Causes
Observation
Data Preprocessing
Training
Data
Training
Dataset
Model
Compute
Training
Program
Data Cleaner
Model
Training Dataset
Data
Consequences



- In the ML simulation chain, the **tools' frontier is larger**, indeed more automated processes are involved in the chain
  - Optimizing inference
  - Optimizing training
  - Part of current and future LLM progress lies in their ability to **call other software tools** (Python, HTTP, …)

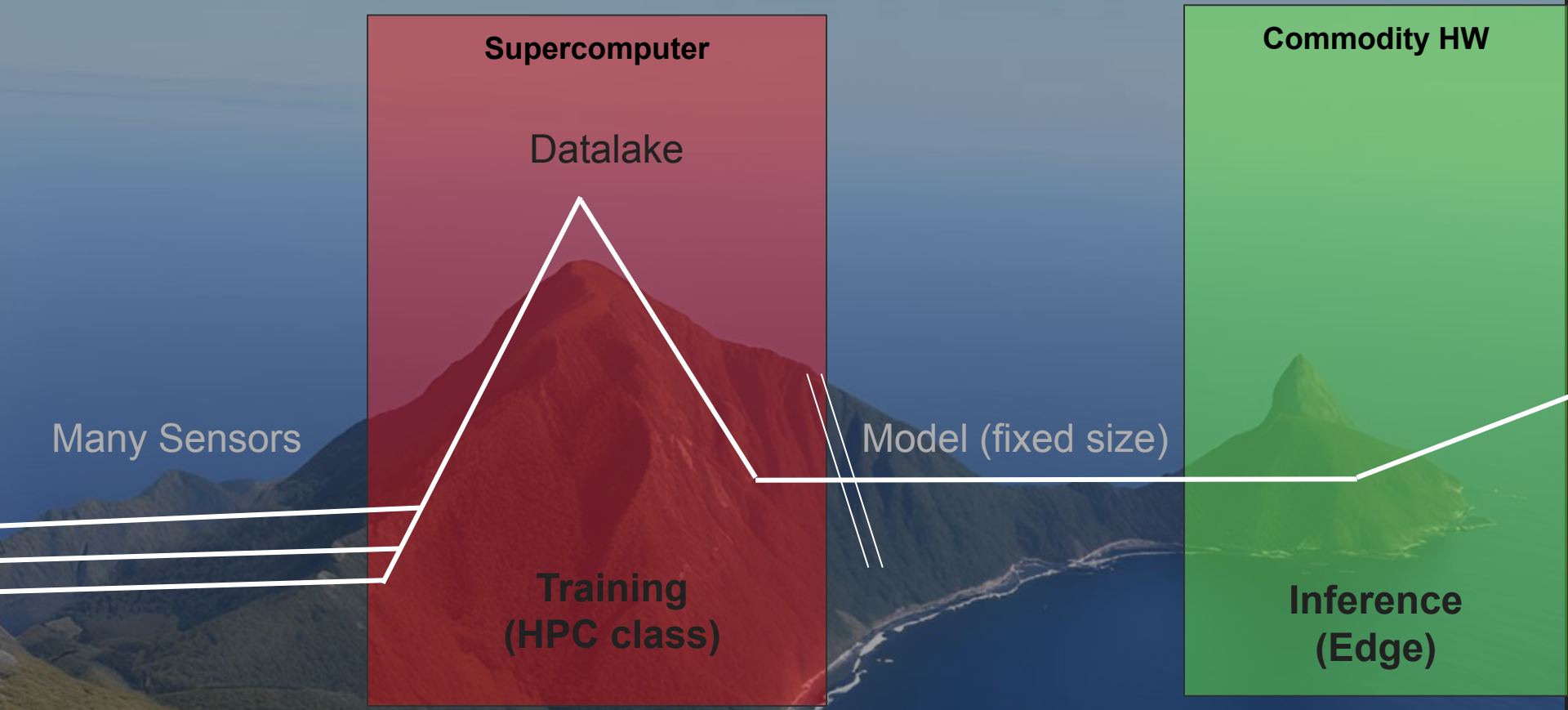
Training Phase → Accumulating Data



Datalake in Supercomputer

Sampling Phase → Inferring Data

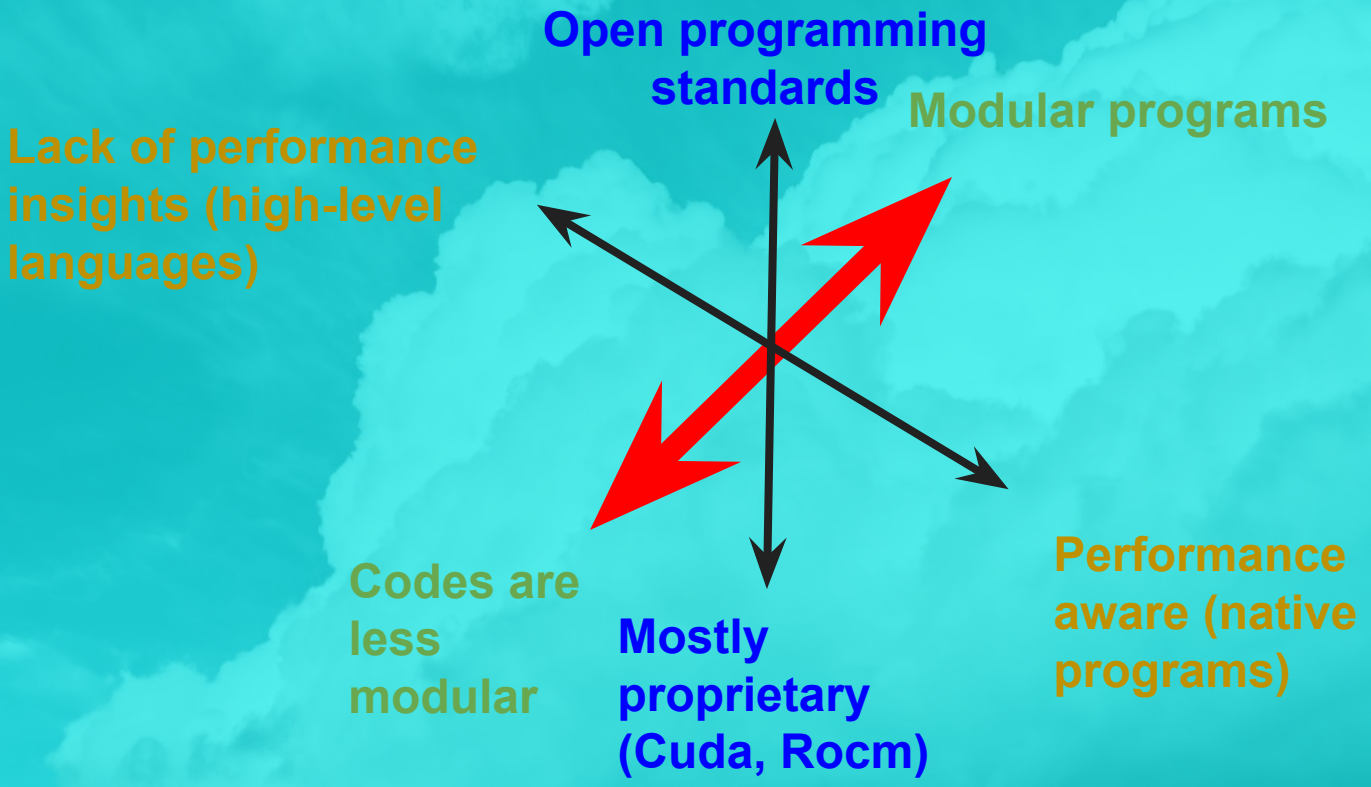
Evolution of Data-Size and Computing Disparity



Specialists on dedicated supercomputers ⟷ Large communities and usages

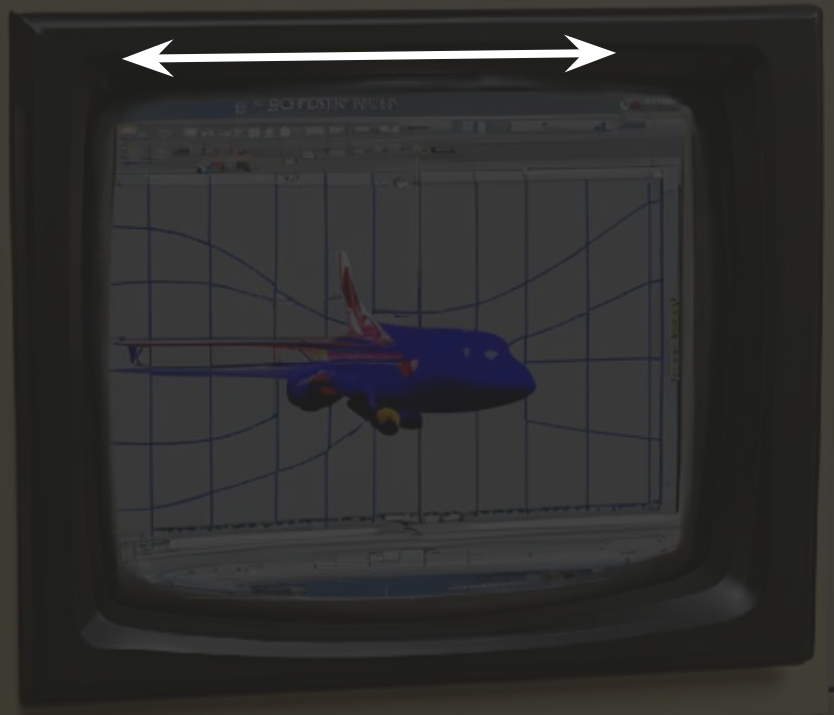
Consequences for tools:

- Broader number of users and applications
- **Varying substrate**
  - From HPC class to workstations and even embedded
- Increasing importance of **workflows** and **systolic** architectures:
  - Massive IOs
  - Data-Streaming
- Need to work on the **computing continuum**
- Supercomputer is part of a service oriented architecture:
  - Reliability, Monitoring, Performance


Open programming standards
Modular programs
Lack of performance insights (high-level languages)
Codes are less modular
Mostly proprietary (Cuda, Rocm)
Performance aware (native programs)

Modular programs

Codes are less modular



From Fortran 77 to Python and the rise of higher-level languages.

HPC Programming Models versus ML

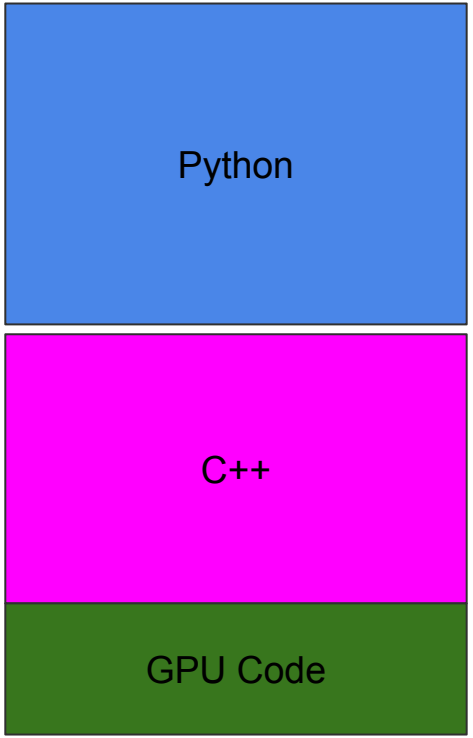
Machine Learning

- Productive (often Python)
  - Expressive syntax
- Composable
  - Build system
  - Package Manager
- Calling in to Low-Level
  - C++ & GPU

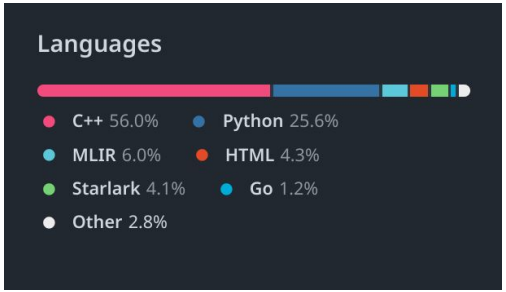
HPC

- Low-Level (C,C++, Fortran)
- Limited Composition
  - Ad-hoc Build
  - Up to recently no PM
- Built for bare-metal performance

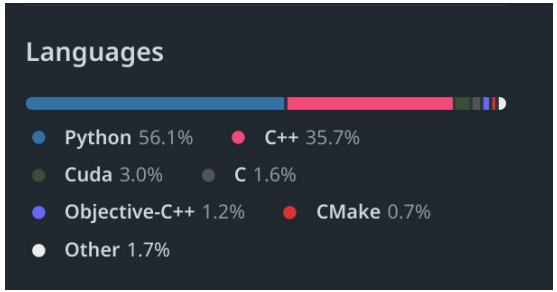
Machine Learning Frameworks and Languages



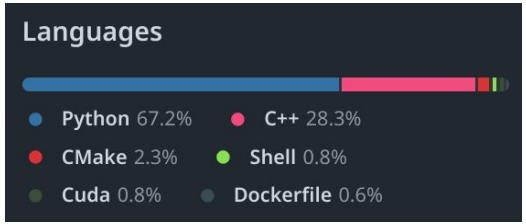
Tensorflow



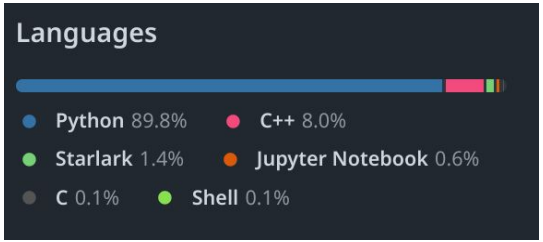
Pytorch



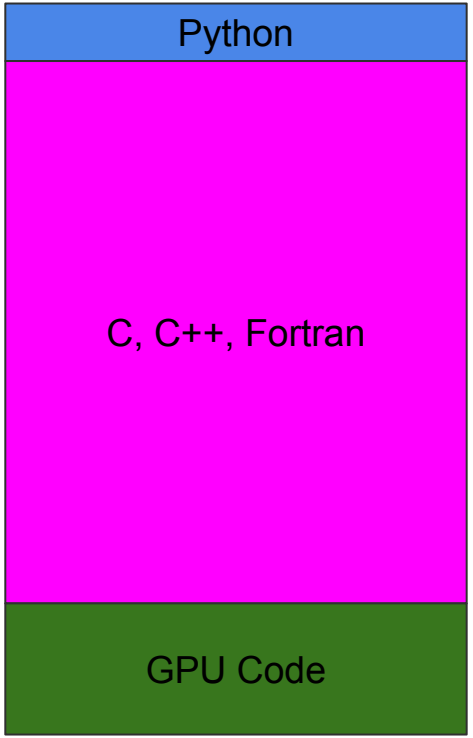
Horovod



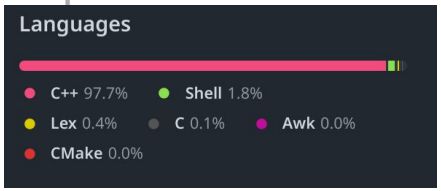
JAX



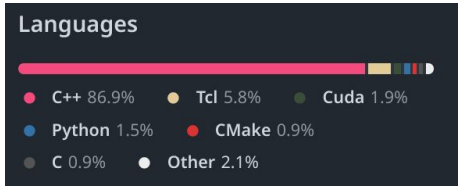
HPC Simulation Codes



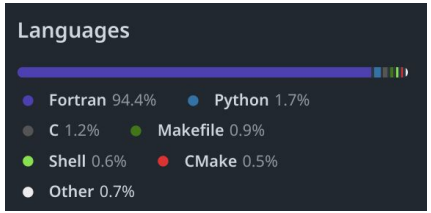
OpenFOAM



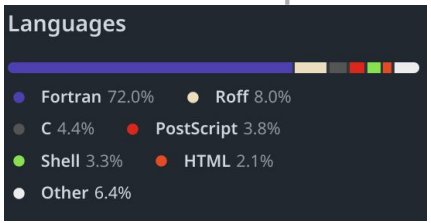
LAMMPS



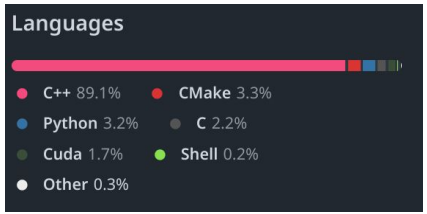
WRF-Hydro



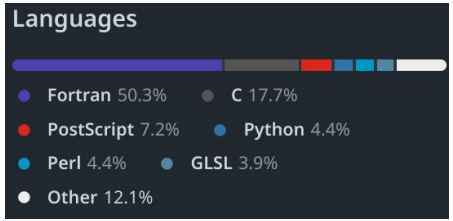
Quantum Espresso



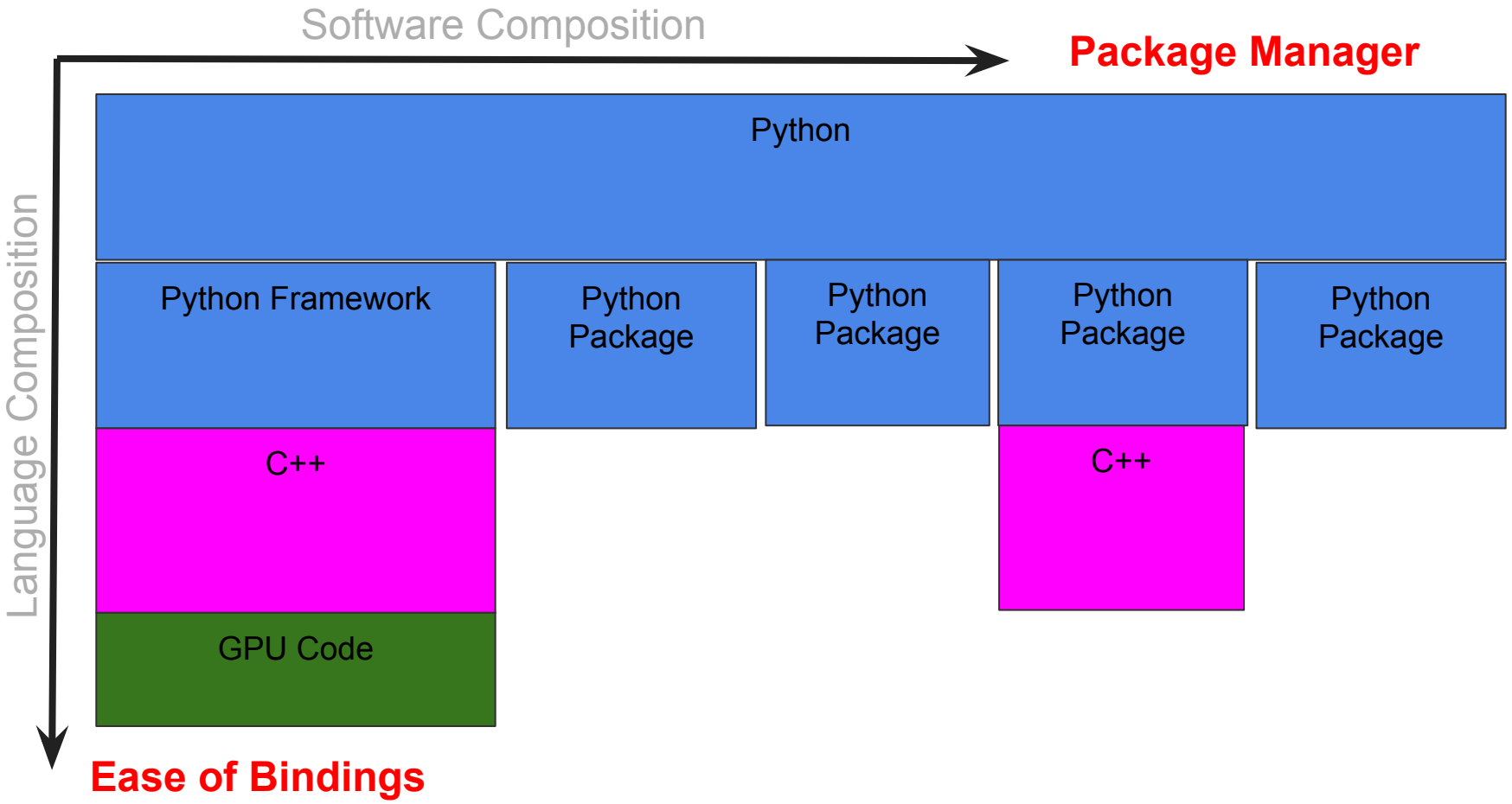
GROMACS



SPECFEM3D



Modularity is a Strength



MPI 4.2 will have a Application Binary Interface (ABI)

MPI Application Binary Interface Standardization

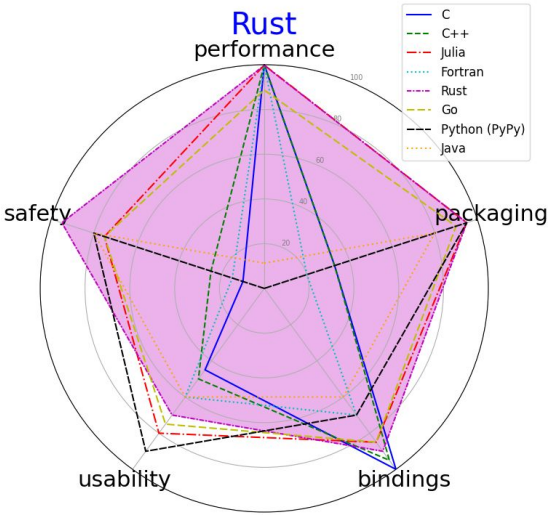
Jeff Hammond, NVIDIA, Finland, jeff_hammond@acm.org
Lisandro Dalcin, KAUST, Saudi Arabia, dalcinl@gmail.com
Erik Schnetter, Perimeter Institute for Theoretical Physics', Canada, schnetter@gmail.com
Marc PéRache, CEA DAM, France, marc.perache@cea.fr
Jean-Baptiste Besnard, ParaTools, France, jbbesnard@paratools.fr
Jed Brown, University of Colorado, USA, jed@jedbrown.org
Gonzalo Brito Gadeschi, NVIDIA, Germany, gonzalob@nvidia.com
Simon Byrne, CalTech, USA, simonbyrne@caltech.edu
Joseph Schuchart, University of Tennessee, Knoxville, USA, schuchart@utk.edu
Hui Zhou, Argonne National Laboratory, USA, zhouh@anl.gov

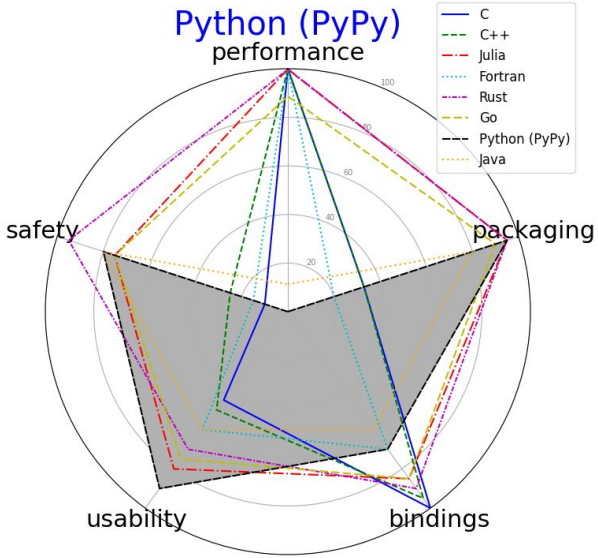
DOI: https://doi.org/10.1145/3615318.3615319

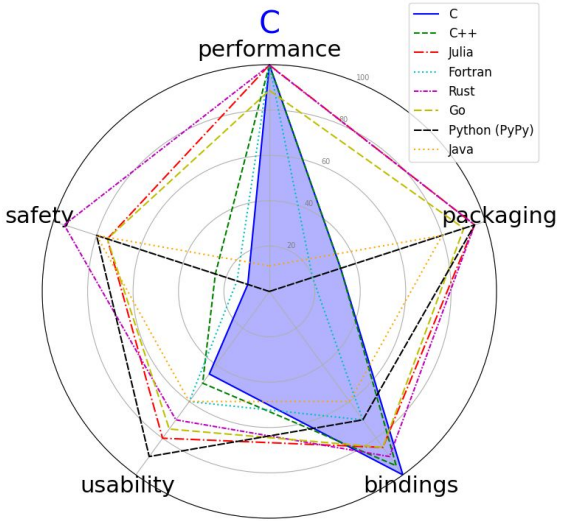
Consequence of the MPI 4.2 ABI for Tools

- No need to recompile your tool to instrument MPI interface (PMPI) in C:
  - Ability to install tools once
  - Improved handler introspection for debuggers
  - Easier for tools distributed in Binary
- In **Fortran** the issue remains:
  - Due to incompatible **Fortran** ABI issues
  - Hopefully Fortran will eventually be externalized (forwarding all to C)
    - **A tool in C should be able to capture all calls**
- **QMPI** wich is the successor of the PMPI aims at implementing plugins:
  - ABI is also a way to enable portable **QMPI** plugins
  - Hopefully Will be the end of the difficult “Fortran wrapper”

Comparing Programming Languages **UNWEIGHTED**

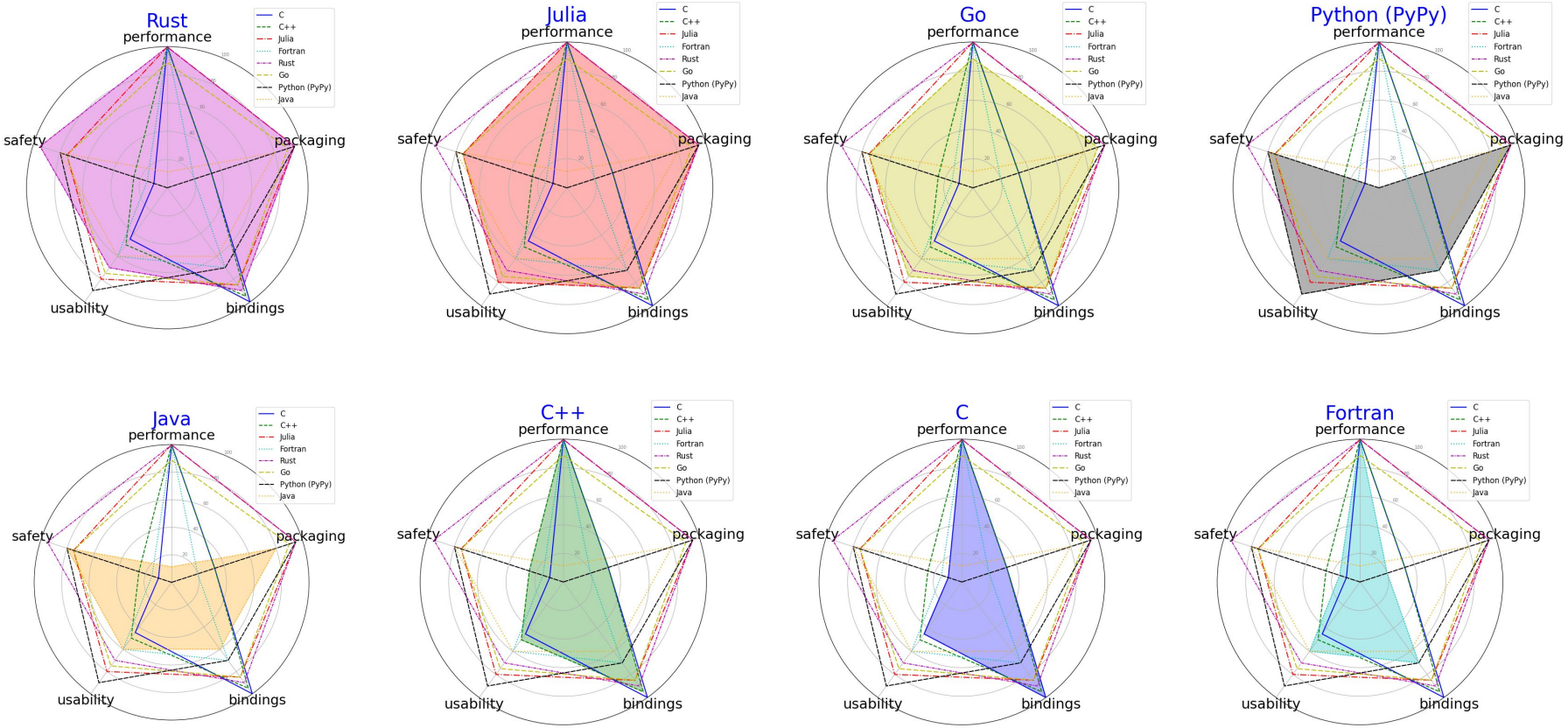






* Performance metrics from: https://niklas-heer.github.io/speed-comparison/ (Computing Pi) – Slowest (Python) as Reference

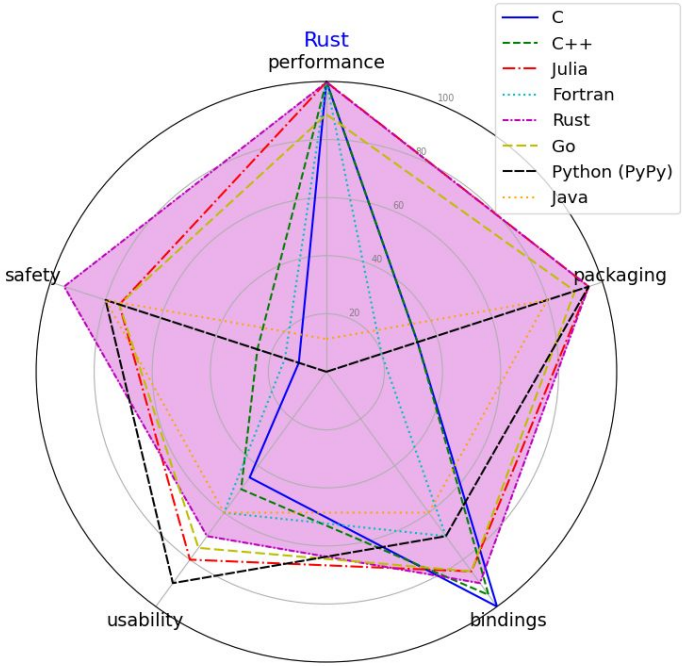
Comparing Programming Languages **UNWEIGHTED**



* Performance metrics from: https://niklas-heer.github.io/speed-comparison/ (Computing Pi) – Slowest (Python) as Reference

Main Constraints for Languages

- Important language features:
  - **Ease of bindings**: connect to other languages
  - **Ease of composition**: compose software libraries/packages
  - **Performance**: run efficiently on cutting-edge hardware
- The rise of **safety** in **Rust**:
  - **Rust** has set a new standard on writing safe code
  - Much time is spent on memory problems …
    - This makes Rust **VERY** productive
  - Ownership for memory management



Snapped Debugger in Rust

https://github.com/paratoolsinc/snapped/

Snapped: a Snapshot debugger implemented in Rust (3.7k loc)

- Features:
  - Relies on a TBON (à la STAT)
  - Wraps the GDB-MI interface
  - Self-wiring
- Implementation:
  - Took 3 days to make
  - Scalled 2000 cores fast
  - Relies on many crates →

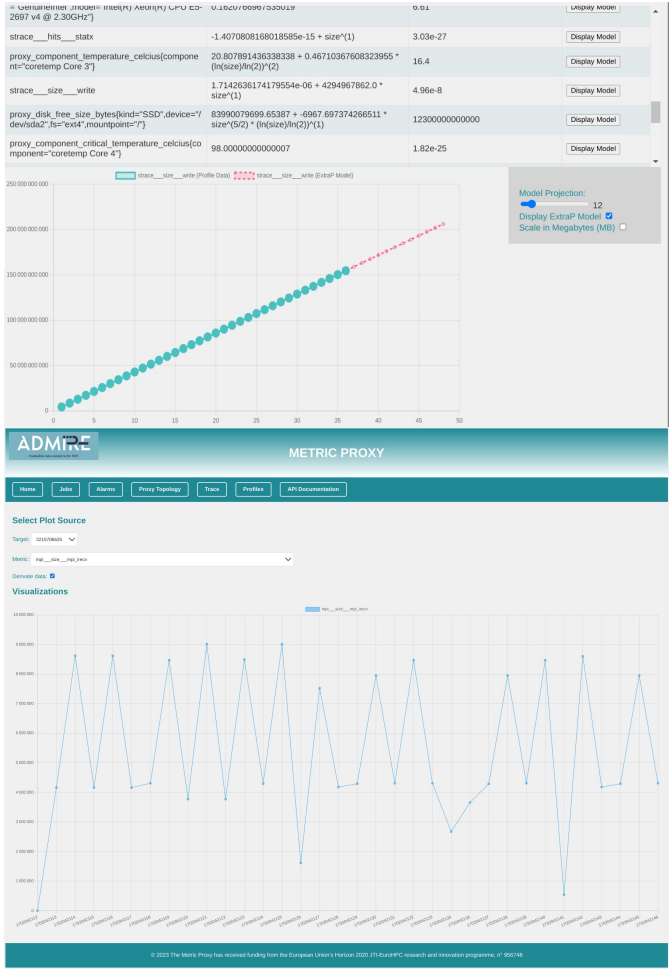
```
snapped v0.1.0 (/home/jbbesnard/repo/snapped_pt)
├── anyhow v1.0.86
├── ascii_tree v0.1.1
├── clap v4.5.16
│   ├── clap_builder v4.5.15
│   │   ├── anstream v0.6.15
│   │   │   ├── anstyle v1.0.8
│   │   │   ├── anstyle-parse v0.2.5
│   │   │   │   └── utf8parse v0.2.2
│   │   │   ├── anstyle-query v1.1.1
│   │   │   ├── colorchoice v1.0.2
│   │   │   ├── is_terminal_polyfill v1.70.1
│   │   │   └── utf8parse v0.2.2
│   │   ├── anstyle v1.0.8
│   │   ├── clap_lex v0.7.2
│   │   └── strsim v0.11.1
│   └── clap_derive v4.5.13 (proc-macro)
│       ├── heck v0.5.0
│       ├── proc-macro2 v1.0.86
│       │   └── unicode-ident v1.0.12
│       ├── quote v1.0.36
│       │   └── proc-macro2 v1.0.86 (*)
│       └── syn v2.0.74
│           ├── proc-macro2 v1.0.86 (*)
│           ├── quote v1.0.36 (*)
│           └── unicode-ident v1.0.12
├── colored v2.1.0
│   └── lazy_static v1.5.0
├── ctrlc v3.4.5
│   └── nix v0.29.0
│       ├── bitflags v2.6.0
│       ├── cfg-if v1.0.0
│       └── libc v0.2.158
│           [build-dependencies]
│           └── cfg_aliases v0.2.1
├── env_logger v0.11.5
│   ├── anstream v0.6.15 (*)
│   ├── anstyle v1.0.8
│   ├── env_filter v0.1.2
│   │   ├── log v0.4.22
│   │   └── regex v1.10.6
│   │       ├── aho-corasick v1.1.3
│   │       │   └── memchr v2.7.4
│   │       ├── memchr v2.7.4
│   │       ├── regex-automata v0.4.7
│   │       │   ├── aho-corasick v1.1.3 (*)
│   │       │   ├── memchr v2.7.4
│   │       │   └── regex-syntax v0.8.4
│   │       └── regex-syntax v0.8.4
│   ├── humantime v2.1.0
│   └── log v0.4.22
├── gdb-machine v0.1.0 (/home/jbbesnard/repo/snapped_pt/gdb-machine)
│   ├── anyhow v1.0.86
│   ├── env_logger v0.11.5 (*)
│   ├── gethostname v0.5.0
│   │   └── rustix v0.38.34
│   │       ├── bitflags v2.6.0
│   │       └── linux-raw-sys v0.4.14
│   ├── log v0.4.22
│   ├── rayon v1.10.0
│   │   ├── either v1.13.0
│   │   └── rayon-core v1.12.1
│   │       ├── crossbeam-deque v0.8.5
│   │       │   ├── crossbeam-epoch v0.9.18
│   │       │   │   └── crossbeam-utils v0.8.20
│   │       │   └── crossbeam-utils v0.8.20
│   │       └── crossbeam-utils v0.8.20
│   ├── regex v1.10.6 (*)
│   ├── serde v1.0.207
│   │   └── serde_derive v1.0.207 (proc-macro)
│   │       ├── proc-macro2 v1.0.86 (*)
│   │       ├── quote v1.0.36 (*)
│   │       └── syn v2.0.74 (*)
│   └── serde_json v1.0.124
│       ├── itoa v1.0.11
│       ├── memchr v2.7.4
│       ├── ryu v1.0.10
│       └── serde v1.0.207 (*)
└── log v0.4.22
```

```
=SNAPPED= root server is running on aldebaran:42359
=SNAPPED= Built a tree of 8 processes in 1.502 seconds
=SNAPPED= All processed joined root server
=SNAPPED= Started processes in 0.272 seconds
a.out: t.c:6: bar: Assertion `0' failed.
=SNAPPED= Stopped processes in 0.085 seconds
=SNAPPED= Collected backtraces in 0.136 seconds
8 .
├─ 3 main t.c:38 -> foo();
│   ├─ 1 foo t.c:20 -> bar();
│   │   └─ 1 bar t.c:6 -> assert(0 /* Crash in bar */);
│   │       └─ 1 __GI___assert_fail assert.c:101
│   │           └─ 1 __assert_fail_base assert.c:92
│   │               └─ 1 __GI_abort abort.c:79
│   │                   └─ 1 __GI_raise raise.c:50
│   │                       └─ 1 Received a Signal
│   │                            SIGABRT
│   └─ 2 foo t.c:24 -> baz();
│       └─ 2 baz t.c:11 -> ((void (*)())0x0)();
│           └─ 2 ??
│               └─ 2 Received a Signal
│                    SIGSEGV
├─ 4 Process has exited (no stack)
│   ├─ 1 Exited Badly
│   │    Exit Code 1
│   └─ 3 Exited Normally
└─ 1 main t.c:53 -> abort();
    └─ 1 __GI_abort abort.c:79
        └─ 1 __GI_raise raise.c:50
            └─ 1 Received a Signal
                 SIGABRT
```

ADMIRE Metric Proxy in Rust

A Machine-Wide monitoring framework
(7.6 k loc)

- Real time tracking of per-job metrics (using TBON)
- HTTP GUI and REST iface
- Traces (resampled)
- Profiles
- Automatic performance modelling:
  - FTIO (frequency)
  - Extra-P (scalability)
- Prometheus compatibility



ADMIRE
malleable data solutions for HPC

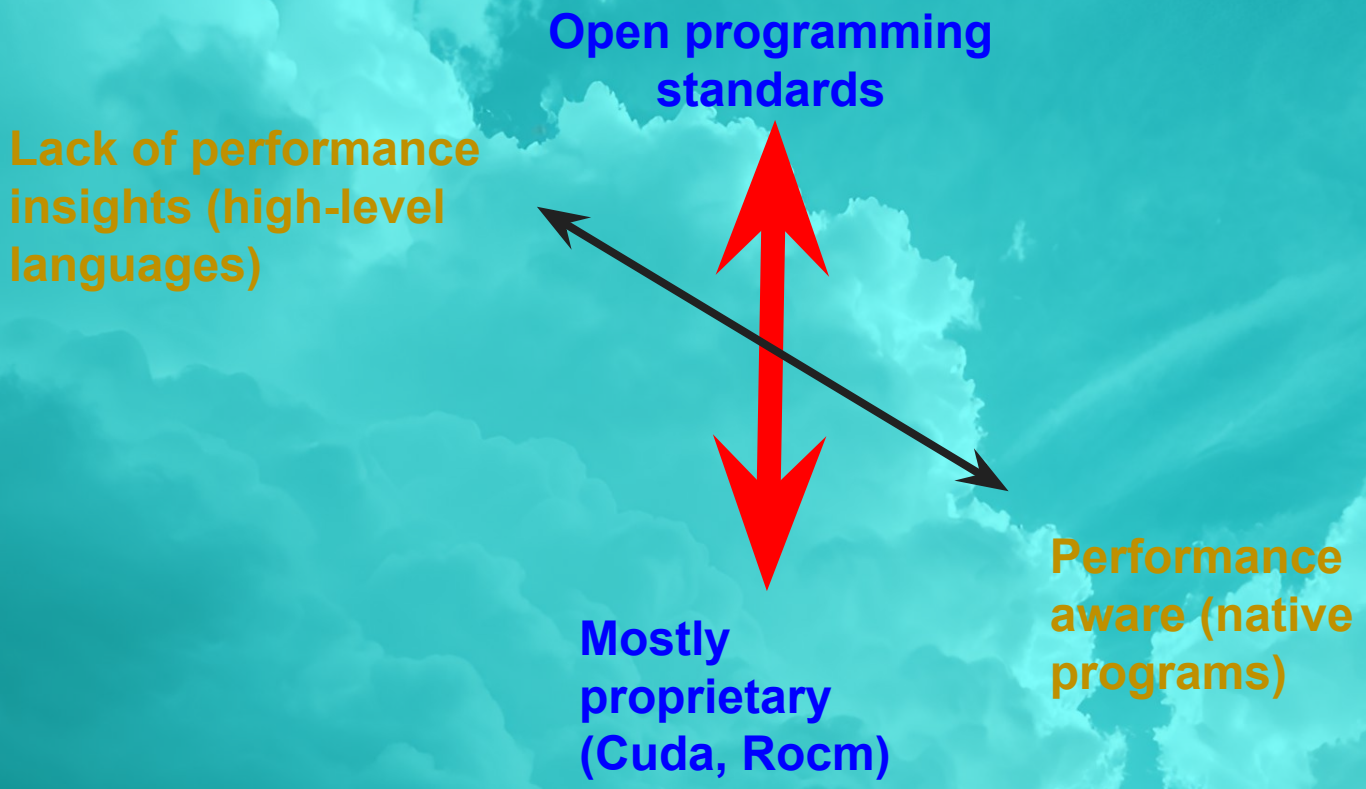
EuroHPC
Joint Undertaking

This project has received funding from the European Union's Horizon 2020 JTI-EuroHPC research and innovation programme, n° 956748

Modular programs ⟷ Codes are less malleable

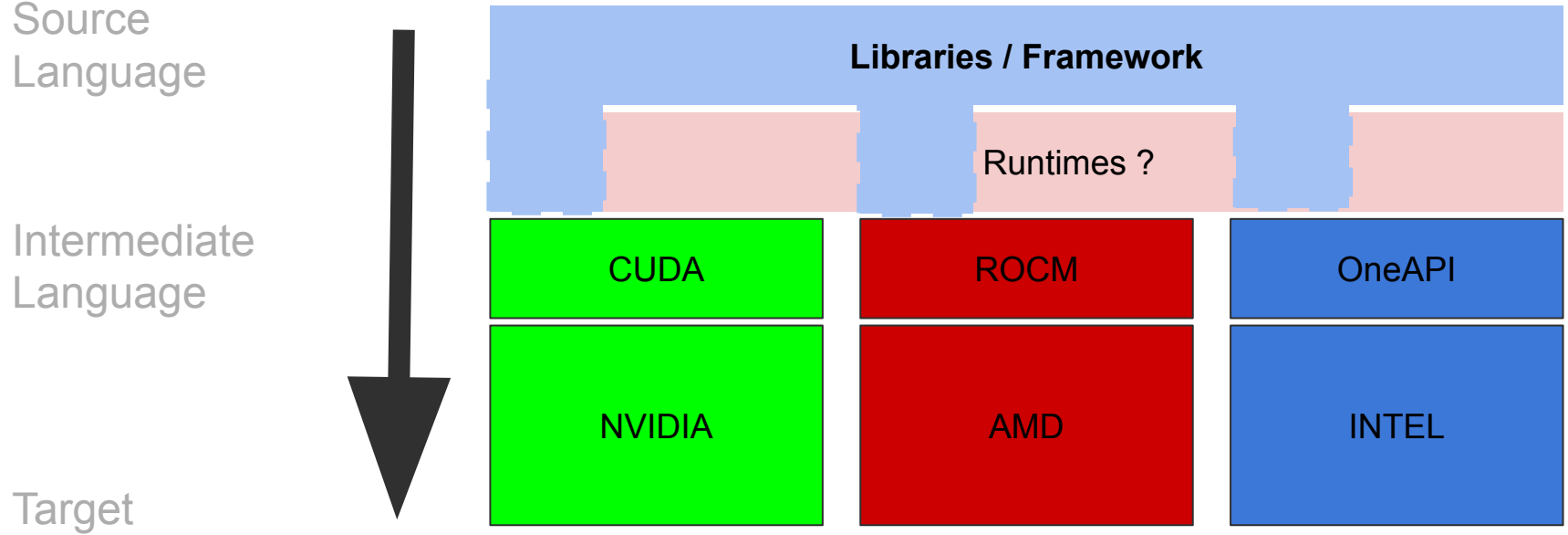
Consequences for tools:

- Need support for larger number of languages (including interpreted ones)
  - Python, Rust, …
- New programs are bound to be **multi-language**
  - Including mixing interpreted and compiled ones
- Programs are **composite** over several libraries
  - Monitoring needs to account for components


Open programming standards
Lack of performance insights (high-level languages)
Performance aware (native programs)
Mostly proprietary (Cuda, Rocm)

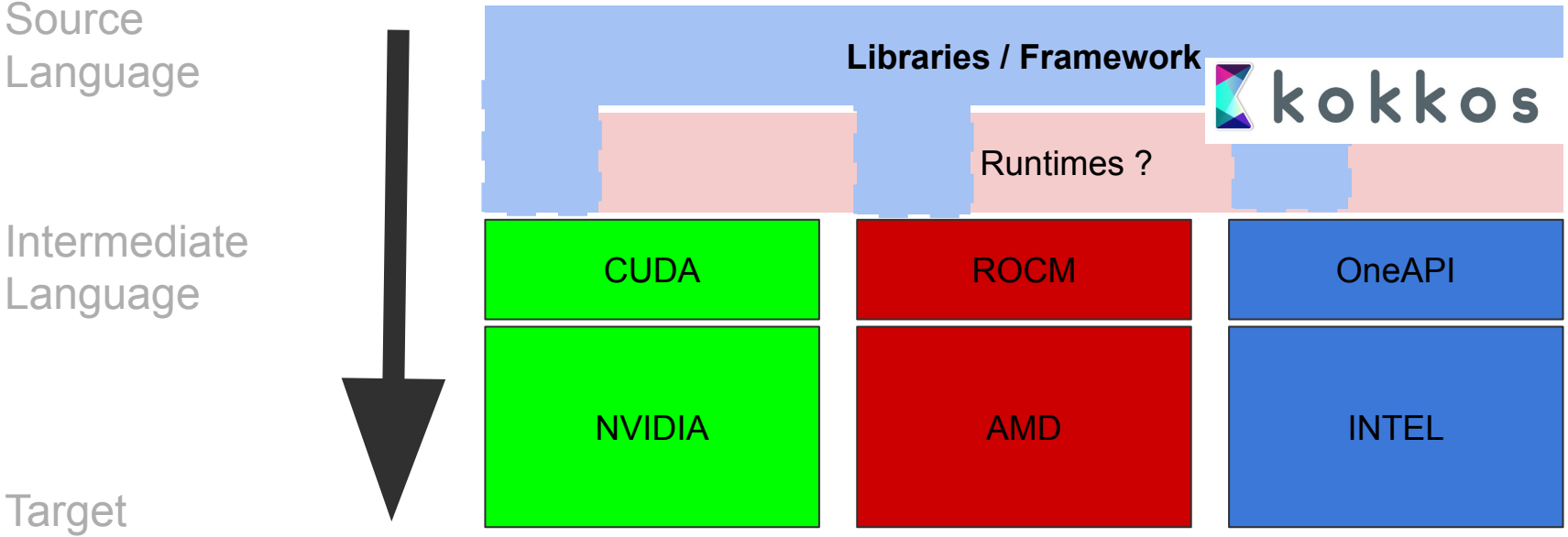
Navigating Differenciated HW : Runtimes VS Libraries

- Hardware discrepancies has led to software discrepancies:
  - GPUs are complex devices
- Runtimes were not able to fully abstract the SIMD model
- Specialized/Libraries became alternative runtimes taking advantage of higher-level languages' expressivity (the true DSL?)



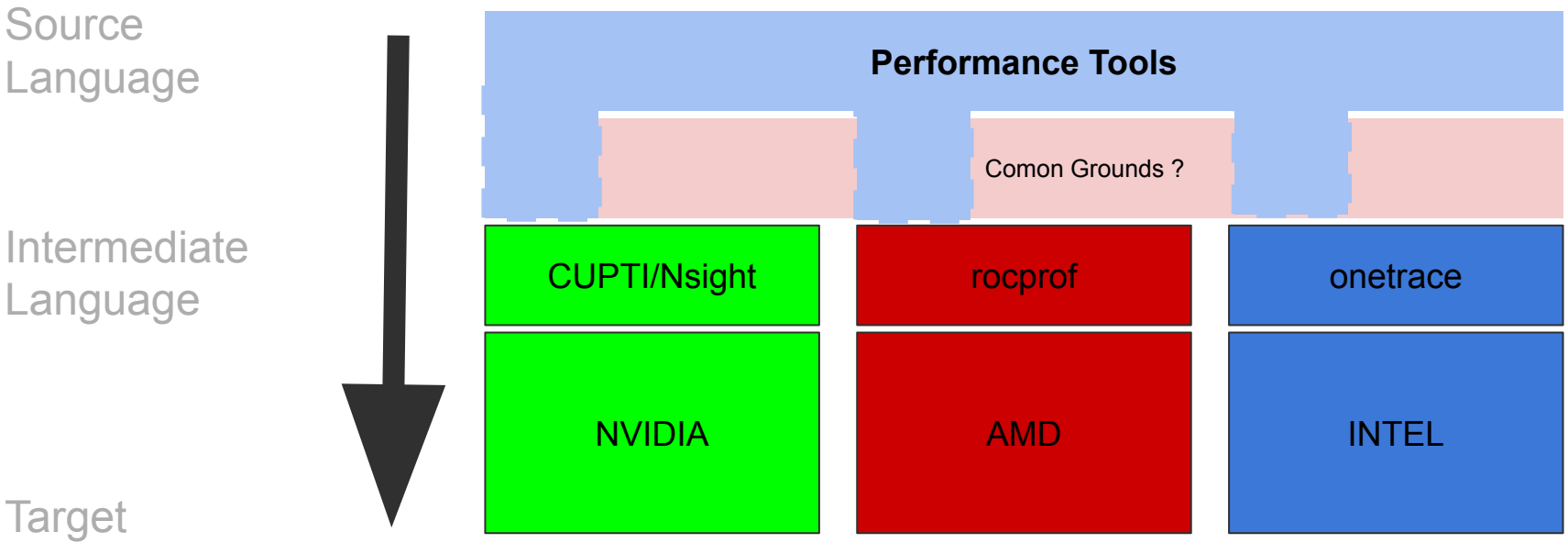
Navigating Differenciated HW : Runtimes VS Libraries

- Hardware discrepancies has led to software discrepancies:
  - GPUs are complex devices
- Runtimes were not able to fully abstract the SIMD model
- Specialized/Libraries became alternative runtimes taking advantage of higher-level languages' expressivity (the true DSL?)

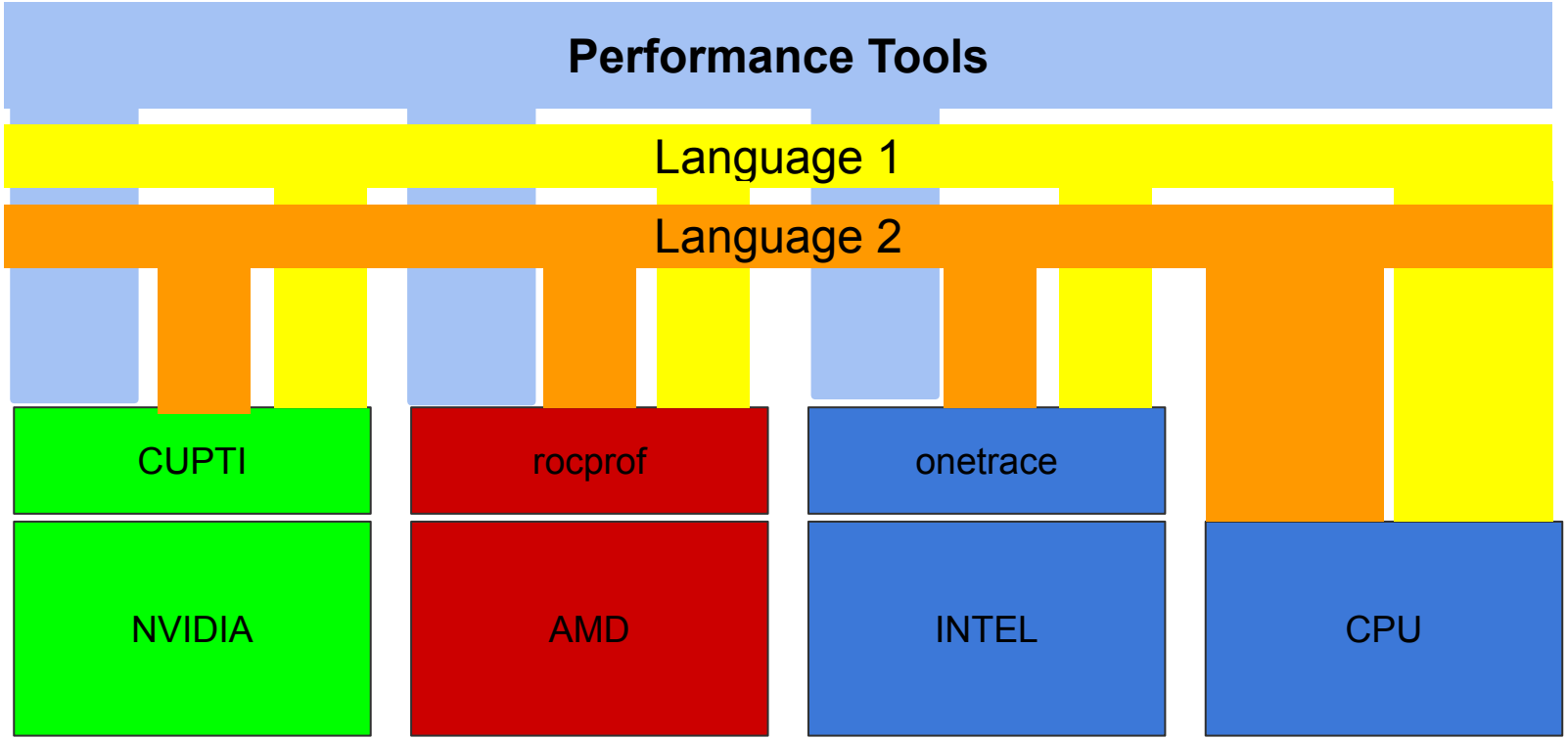


Monitoring is also Facing HW Discrepancies

- Multiple proprietary hardware (rapidly evolving)
- No common monitoring interfaces (previously exposed by runtimes)
- Each vendor provides its tool



Program State is More and More Intertwined



Mostly proprietary
(Cuda, Rocm, …)

Open programming
standards

Consequences for tools:

- Tools have to represent a more complex state for programs:
  - GPU state (no unified representation)
  - CPU state with layered languages
- Need for unified cross-language profiling
- Need to work on unified instrumentation layers for GPUs
- More openness from HW vendors:
  - Software is selling hardware (not the opposite)
  - Efficient tools are an asset for a HW platform
- Tools need to work at slightly higher level with variety in runtimes:
  - No more OMPT, MPI-T, PMPI
  - But now CUPTI, ROCPROF, ONETRACE, KOKKOS, TORCH, …

Lack of performance insights (high-level languages)

Performance aware (native programs)

Performance has always been at the center of HPC

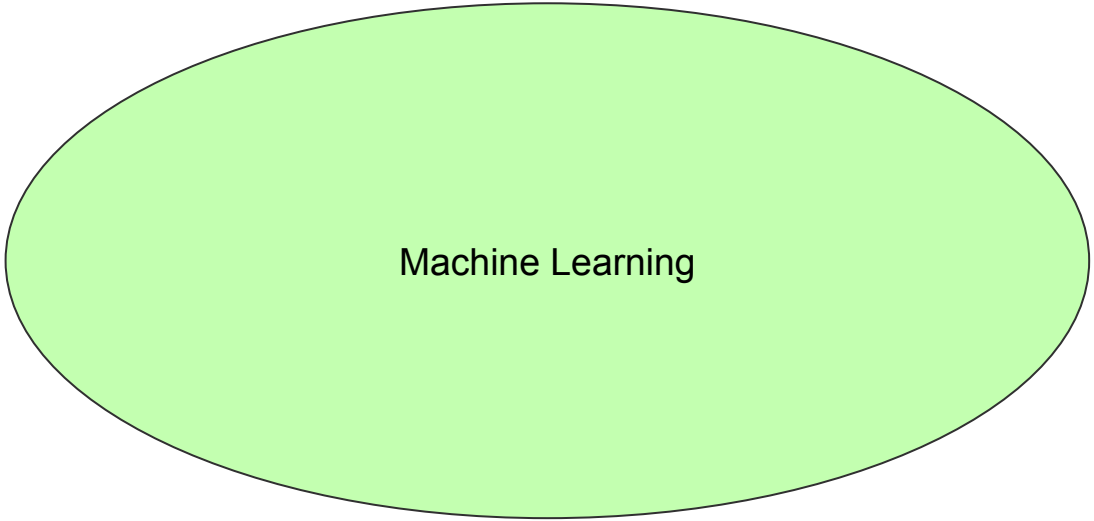
HIGH

PERFORMANCE

COMPUTING

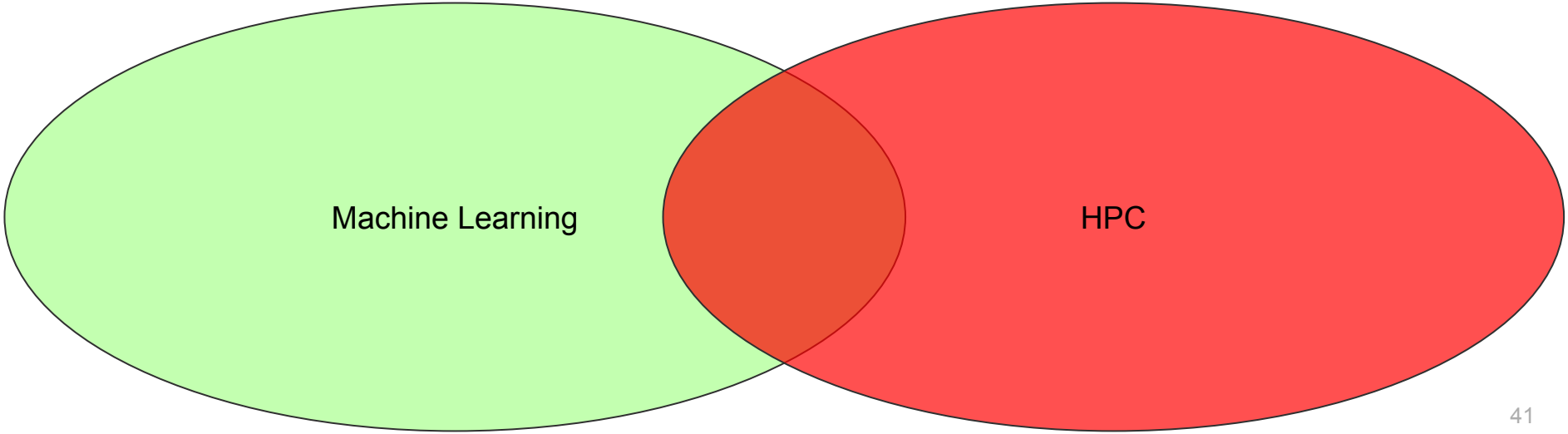
More Users → More Risk of Wasting Resources

- ML is widespread, and thus needs even more optimization:
  - More risk of burning CPU/GPU hours
  - Codes are spanning a larger space (harder optimization):
    - Language mixing with Python
    - Multiplication of layers
  - Low-level optimizations are less trivial
  - Future approaches might be less regular (Tool calling, Chaining, RAG)



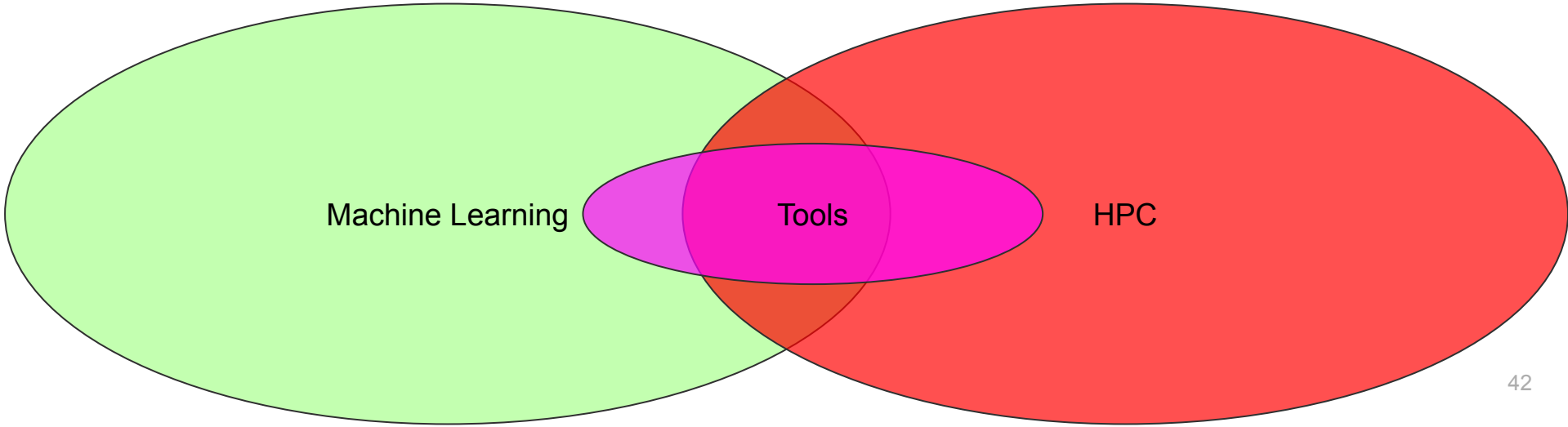
HPC is the Foundation of Parallel Computing

- HPC has provided core technologies and abstractions to ML:
  - GPU and Runtimes
  - Tools, distributed computing abstractions
  - Theory of parallelism and performance modelling
    - Payload invariant
  - No reason it continues to do so



HPC Tools Shall Conquer the ML Space !

- Tools are strongly needed in this space, and should open to:
  - More languages (Particularly Python and Rust)
  - Unified views of programs on multiple devices
  - Horizontal payloads (workflows) and client-server models (continuum?)
  - Uses may span an wider variety of devices:
    - HPC, Desktop and embedded (edge sensors, inference)
  - No reason to give-up on unified profiling layers (aggregation ?)



Challenges and Opportunities for Tools in a Changing HPC Landscape

Jean-Baptiste BESNARD
ParaTools SAS

ParaTools