Efficient Scientific Computing School – 13th Edition

Introduction to Software Portability Among Heterogeneous Architectures

CASUS
CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

www.casus.science

HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

TECHNISCHE UNIVERSITÄT DRESDEN

CBG Max Planck Institute of Molecular Cell Biology and Genetics

UFZ HELMHOLTZ Centre for Environmental Research

Uniwersytet Wrocławski

SPONSORED BY THE

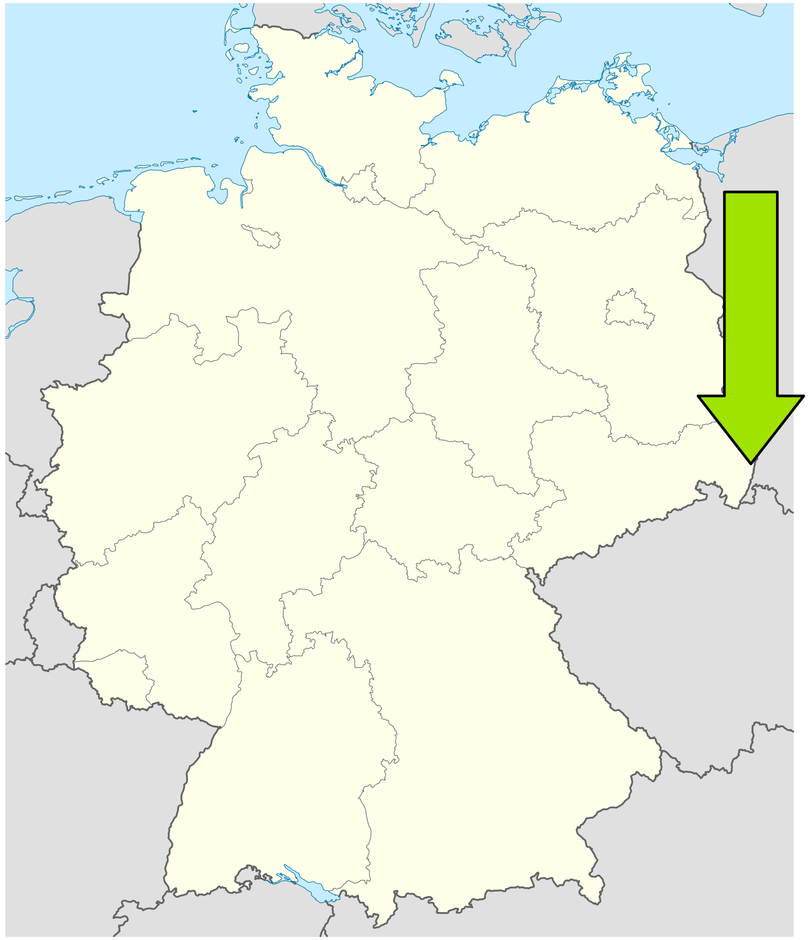
Federal Ministry of Education and Research

STAATSMINISTERIUM FÜR WISSENSCHAFT KULTUR UND TOURISMUS

Freistaat SACHSEN

About Me

- Computer Scientist
  - Email: **j.stephan@hzdr.de**
  - GitHub: **https://www.github.com/j-stephan**
- PhD Student @ CASUS – Center for Advanced Systems Understanding
  - Located in Görlitz, Germany
  - Department of Helmholtz-Zentrum Dresden-Rossendorf (HZDR)
  - Web page: **https://www.casus.science**
- Strong high-performance computing (HPC) focus
  - Programming for heterogeneous hardware
  - Designing and implementing abstraction layers
  - Performance analysis



What is a heterogeneous system?

Life After TBB

So far on ESC22

- Computer architectures
- Modern C++ programming
- Floating-point computations
- Parallel C++ programming with TBB

Life After TBB

So far on ESC22

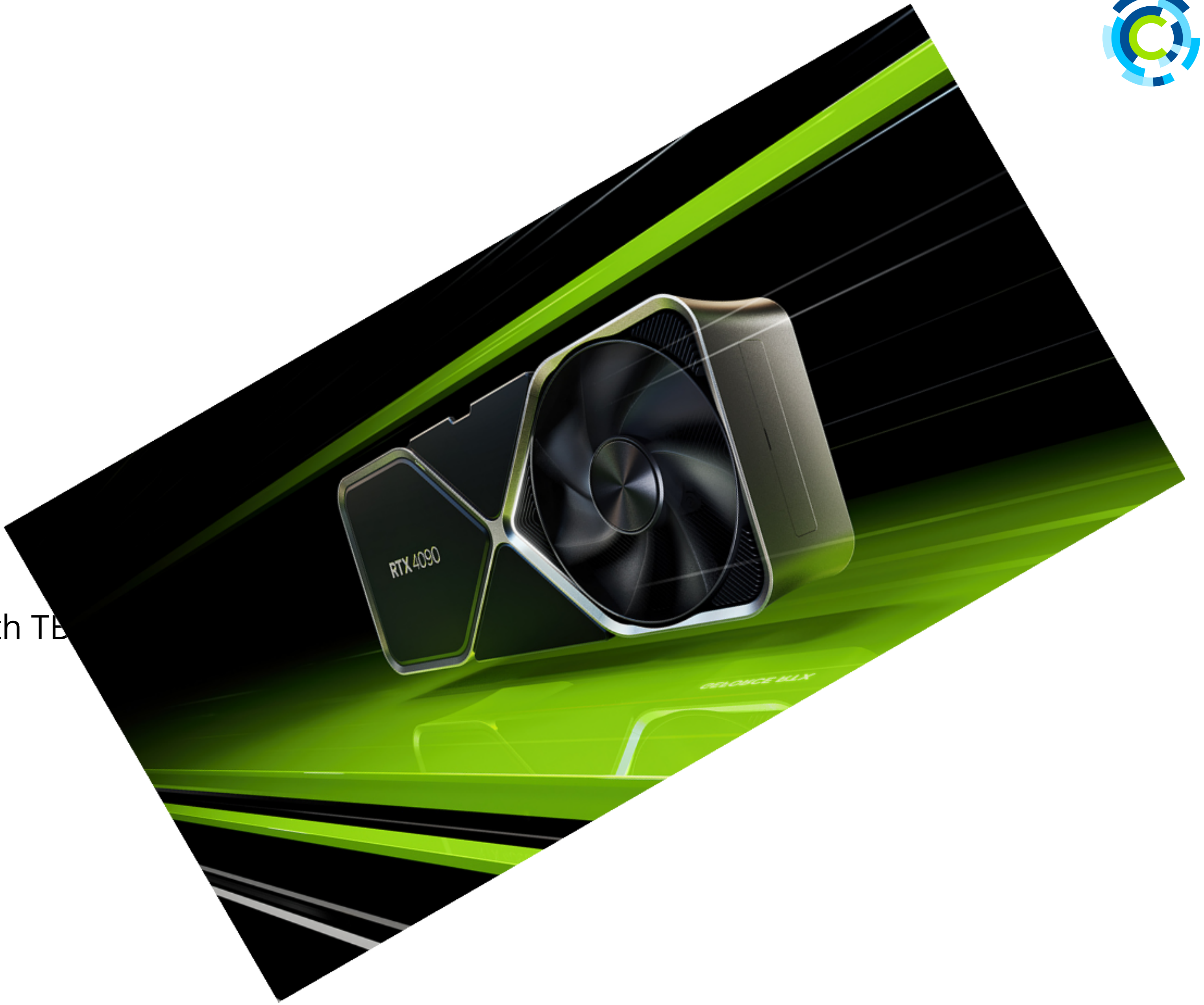
- Computer architectures
- Modern C++ programming
- Floating-point computations
- Parallel C++ programming with TBB

… **What is missing?**

Life After TBB

So far on ESC22

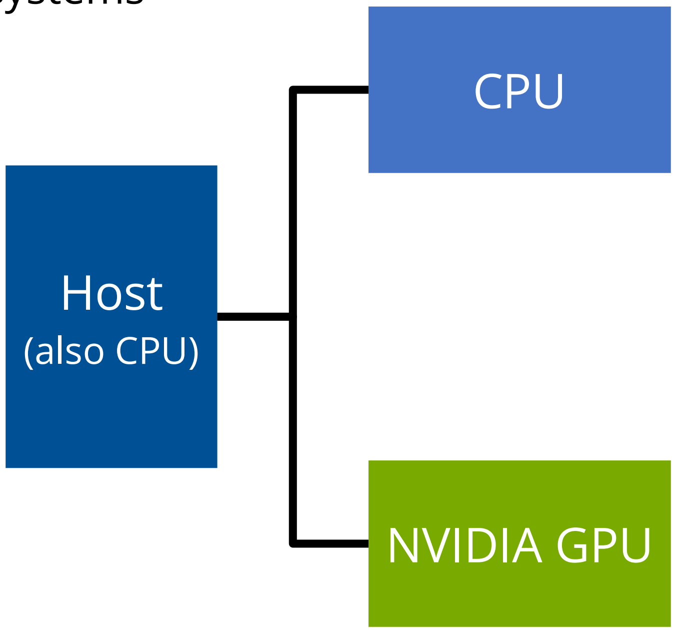
- Computer architectures
- Modern C++ programming
- Floating-point computations
- Parallel C++ programming with TB



… **What is missing?**

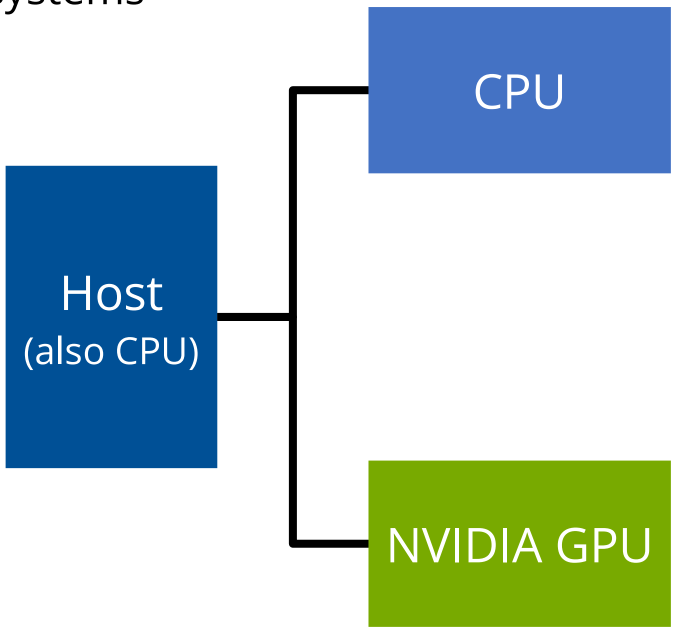
Modern Heterogeneous Systems

- (Virtually) all modern desktops and laptops are heterogeneous systems
  - Smartphones, too (probably)
- In science, we want to use all available computing power
- Problem: How to exploit the hardware?



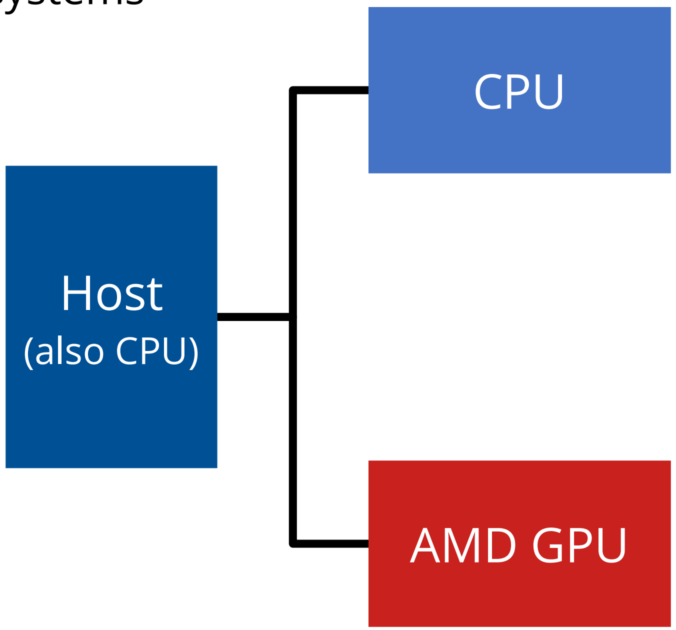
Modern Heterogeneous Systems

- (Virtually) all modern desktops and laptops are heterogeneous systems
  - Smartphones, too (probably)
- In science, we want to use all available computing power
- Problem: How to exploit the hardware?
- Solution: Use NVIDIA CUDA! (Tomorrow on ESC22)



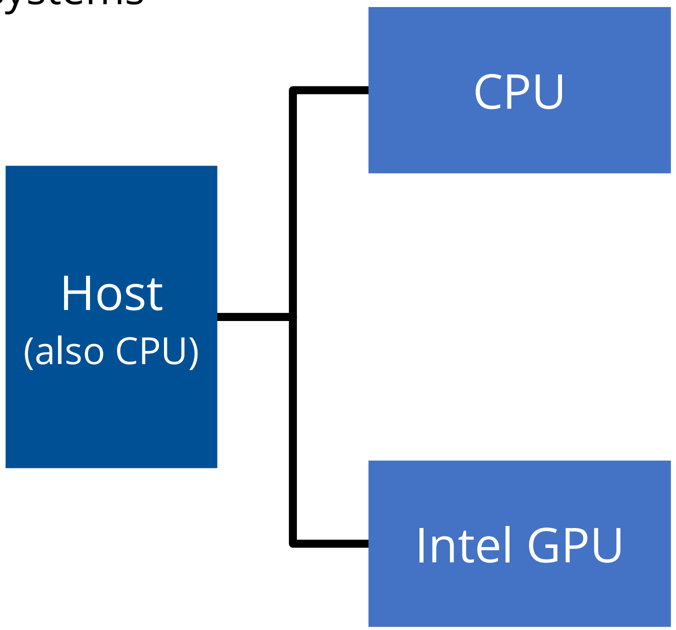
Modern Heterogeneous Systems

- (Virtually) all modern desktops and laptops are heterogeneous systems
  - Smartphones, too (probably)
- In science, we want to use all available computing power
- Problem: How to exploit the hardware?
- Solution: Use NVIDIA CUDA?



Modern Heterogeneous Systems

- (Virtually) all modern desktops and laptops are heterogeneous systems
  - Smartphones, too (probably)
- In science, we want to use all available computing power
- Problem: How to exploit the hardware?
- Solution: Use NVIDIA CUDA?



Illustrating the problem

Your boss walks into your office…

"I need a simulation for our next paper!"

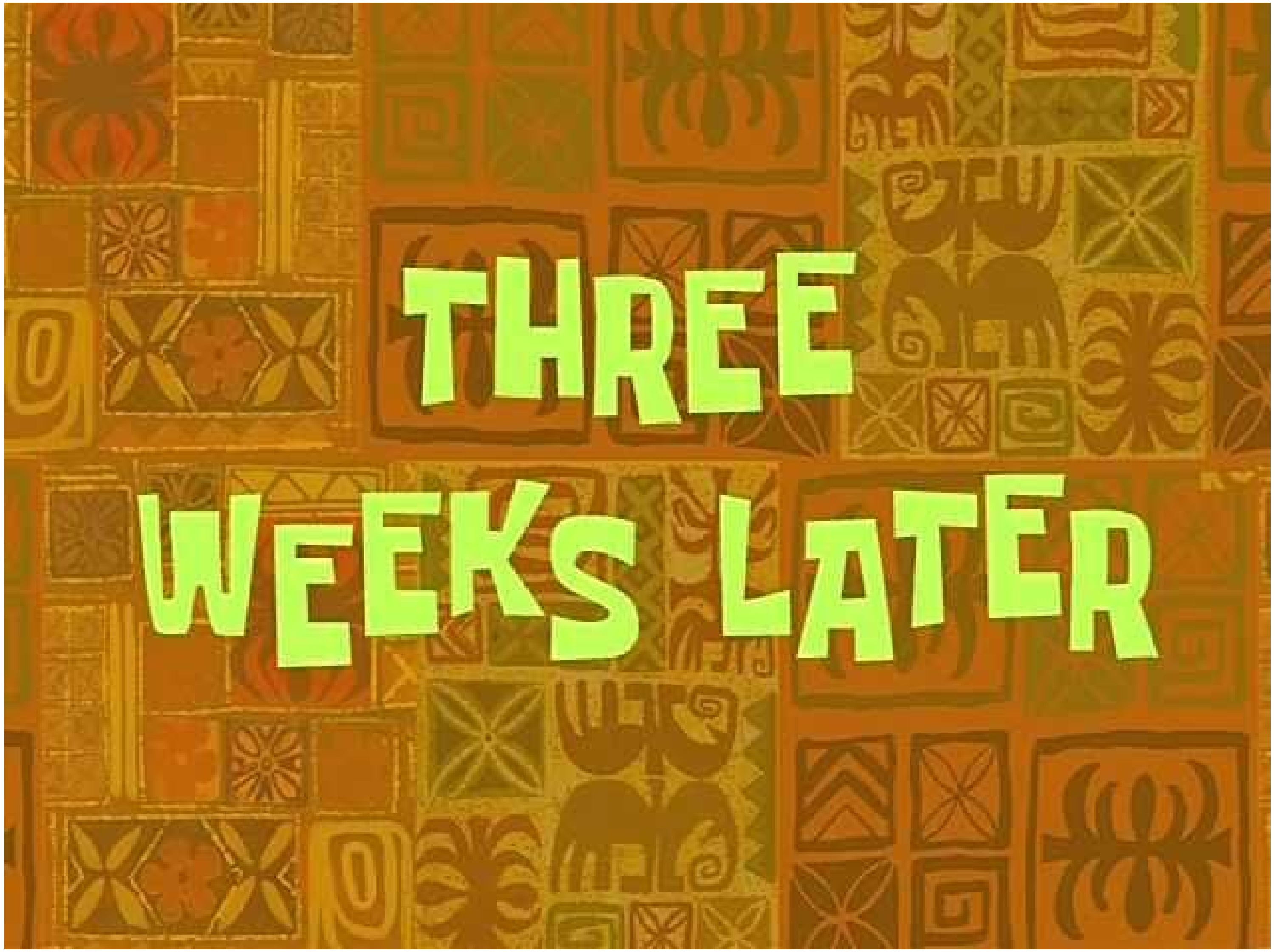
- Easily parallelizable algorithm
- Large amount of data
- Your workstation has a good CPU
- What do you do?


SOME TIME LATER...

Some time later…

“Hey, your workstation has a NVIDIA GPU, right?”

- CUDA and TBB require different API calls, memory management, etc.
- You want to run the TBB-accelerated code, too.
- What do you do?


THREE
WEEKS LATER

Three weeks later…

“Our IT department just installed new compute nodes!”

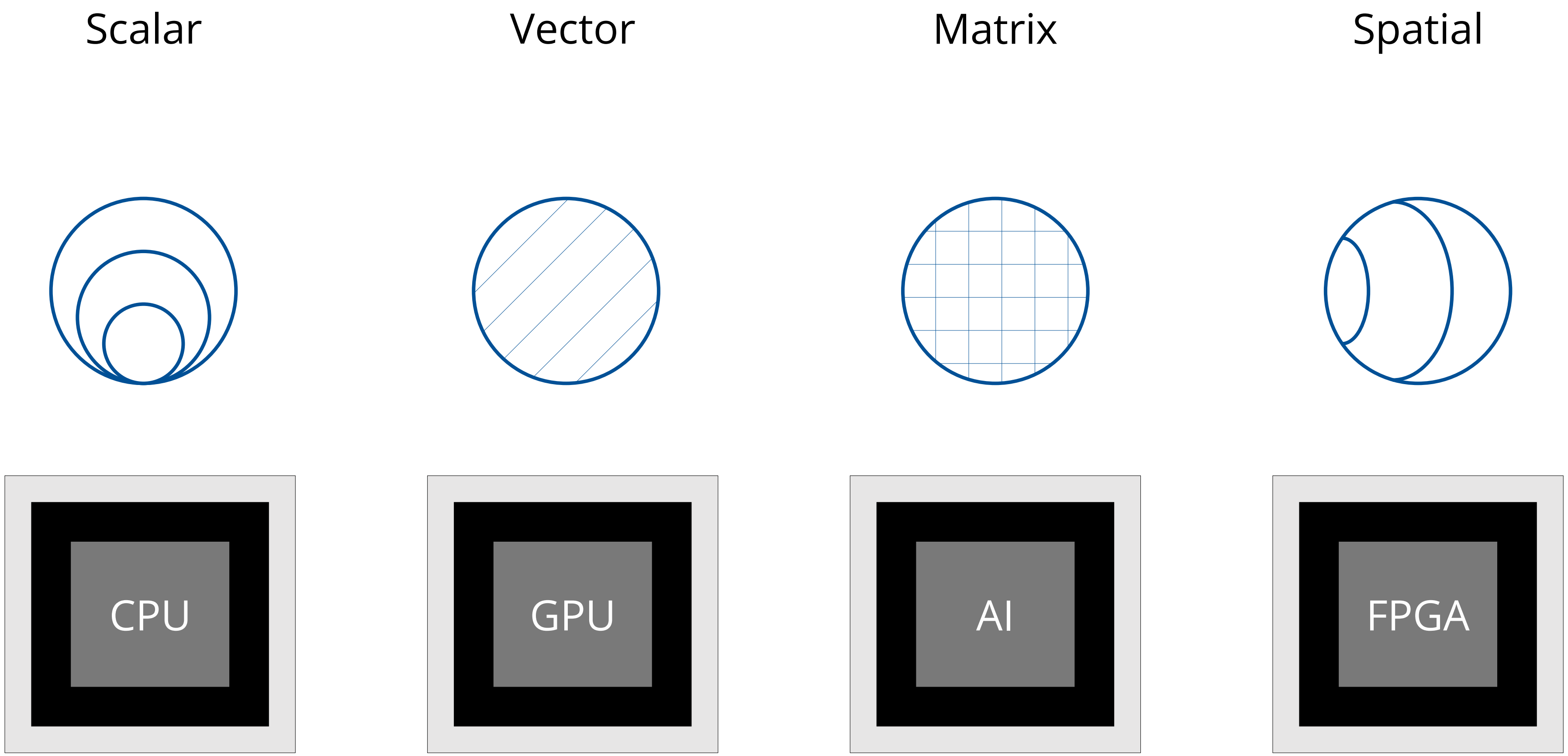
- Unfortunately (for you), they bought Intel GPUs.
- Intel GPUs are programmed using oneAPI DPC++ (a.k.a SYCL).
- Do you know oneAPI DPC++?
- What do you do?

The Portability Problem

- Vendors supply their own toolchains for their special hardware.
- It is hard to program all of them efficiently.
  - Different APIs
  - Different performance characteristics
  - …
- Maintenance & portability become time-consuming tasks.

Choosing the right tool for the job

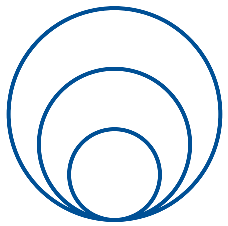
Workload Patterns

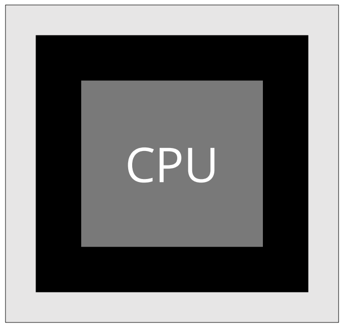


CPUs

- General purpose tool
- Starting point and *Host* of any program
- Good for task parallelism and data parallelism with little data loads
- Bad for data parallelism with massive data loads
  - Mitigated by (non-portable) SIMD instructions

Scalar

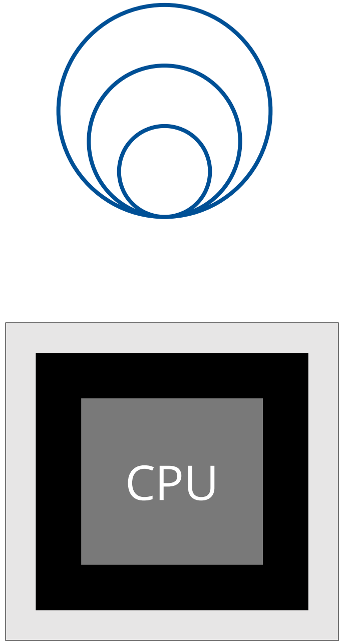




CPUs

- General purpose tool
- Starting point and *Hos*
- Good for task parallel
- Bad for data parallelis
    - Mitigated by (non-portal

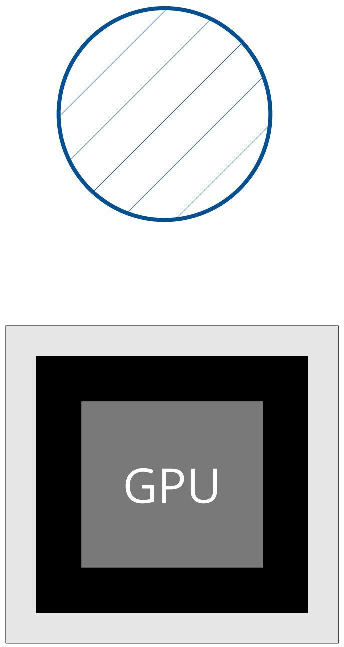
Scalar



GPUs

- Optimized for independent pixel processing
- Ideal for massively parallel workloads
- Bad for algorithms with much divergence (`if` … `else`)
- Good half-precision and single-precision floating-point performance
- (Historically) bad double-precision floating-point performance
- Okay integer performance

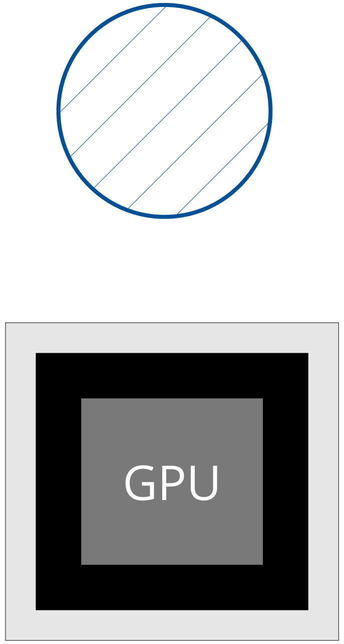
Vector



GPUs

- Optimizec
- Ideal for n
- Bad for al
- Good half
- (Historical
- Okay integ

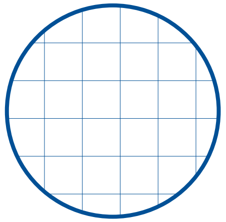
Vector

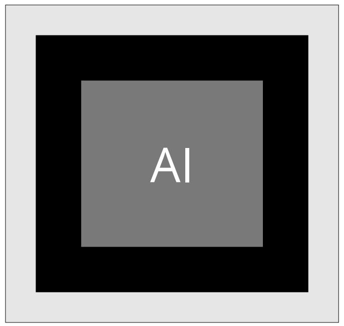


AI Accelerators

- Optimized for AI training workloads
- Ideal for matrix-matrix or matrix-vector operations
- Good mixed-precision performance
- May support integer / float precisions not found on GPUs (Example: 4-bit integers)
  - The supported precisions for floating point and integer vary between vendors and/or hardware generations
  - Some common precisions may not be supported (Example: single-precision floating point on NVIDIA tensor cores)

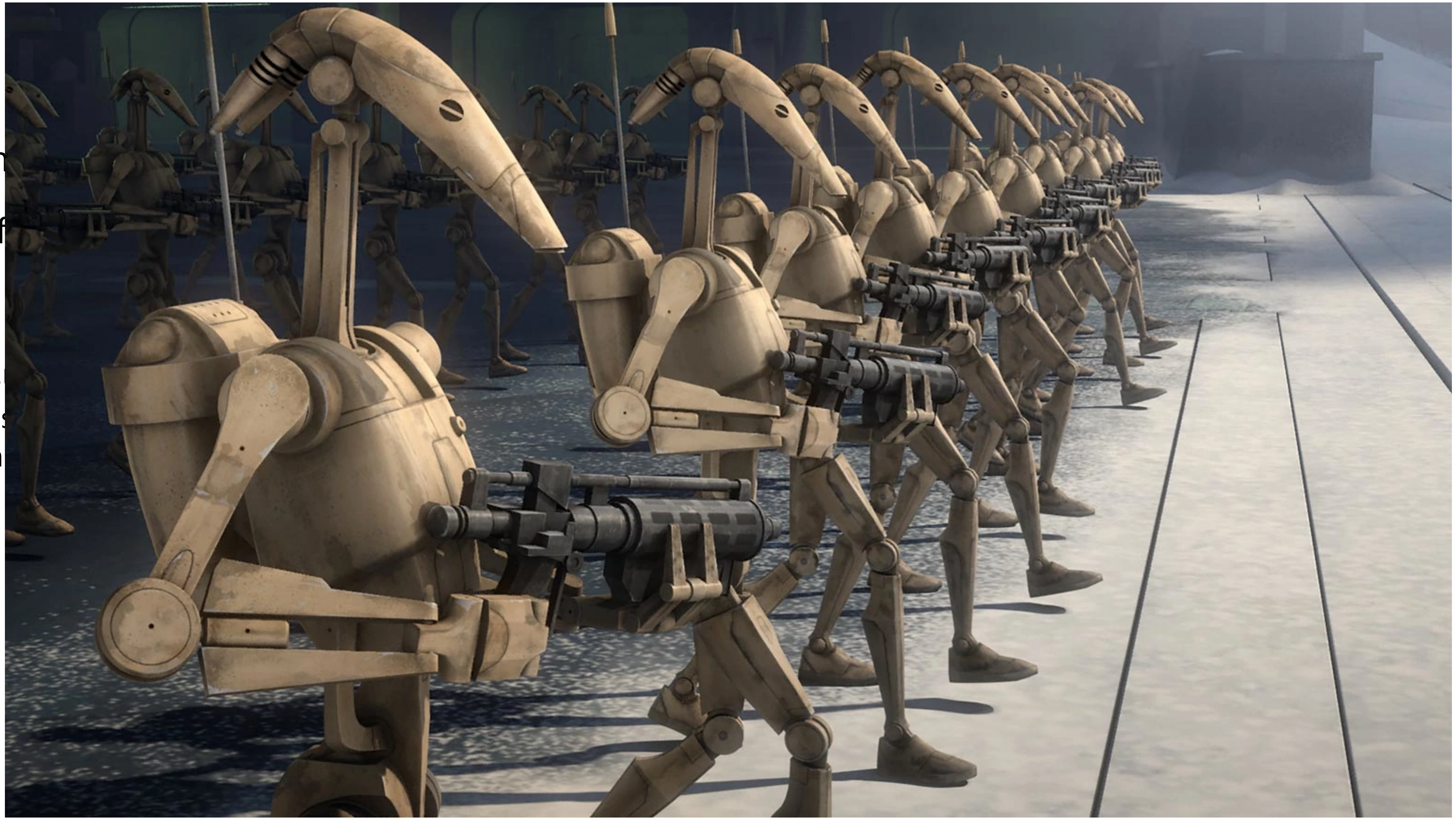
Matrix



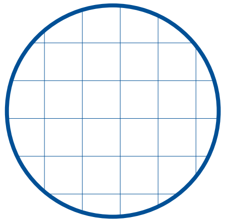


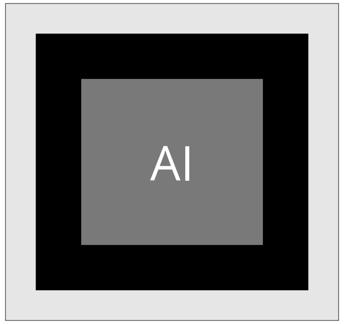
AI Accelerators

- Optim
- Ideal f
- Good
- May s
  - The
  - Som



Matrix

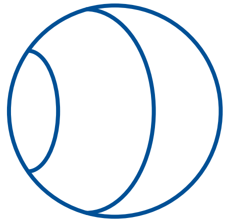




Field-Programmable Gate Arrays (FPGAs)

- Allows to use hardware specialized for use case
- Can support (almost) any-length integers
- Data parallelism achieved by multiplying hardware layout
- Task parallelism achieved by placing hardware layouts next to each other
- (User-)deterministic time behaviour
  - Example: Image processing with exactly 300 MHz
- Constrained by chip limits
  - Low frequency
  - Limited disk space
- Hardware synthesis takes a long time

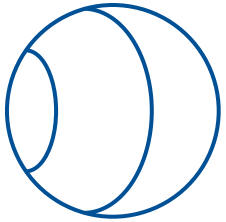
Spatial



Field-Programmable Gate Arrays (FPGAs)

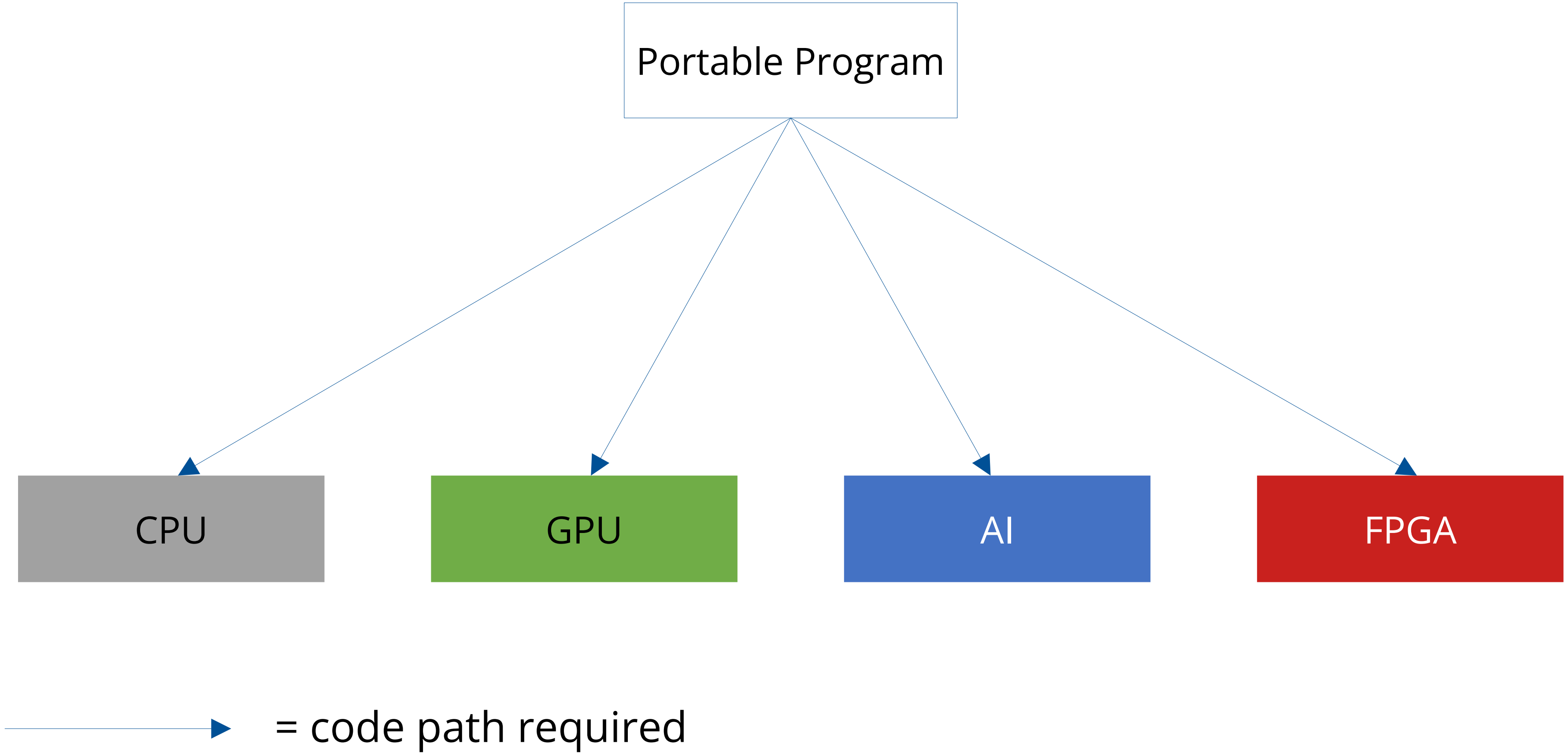
- Allows to
- Can supp
- Data para
- Task para
- (User-)de
  - Example
- Constrain
  - Low freq
  - Limited c
- Hardware

Spatial

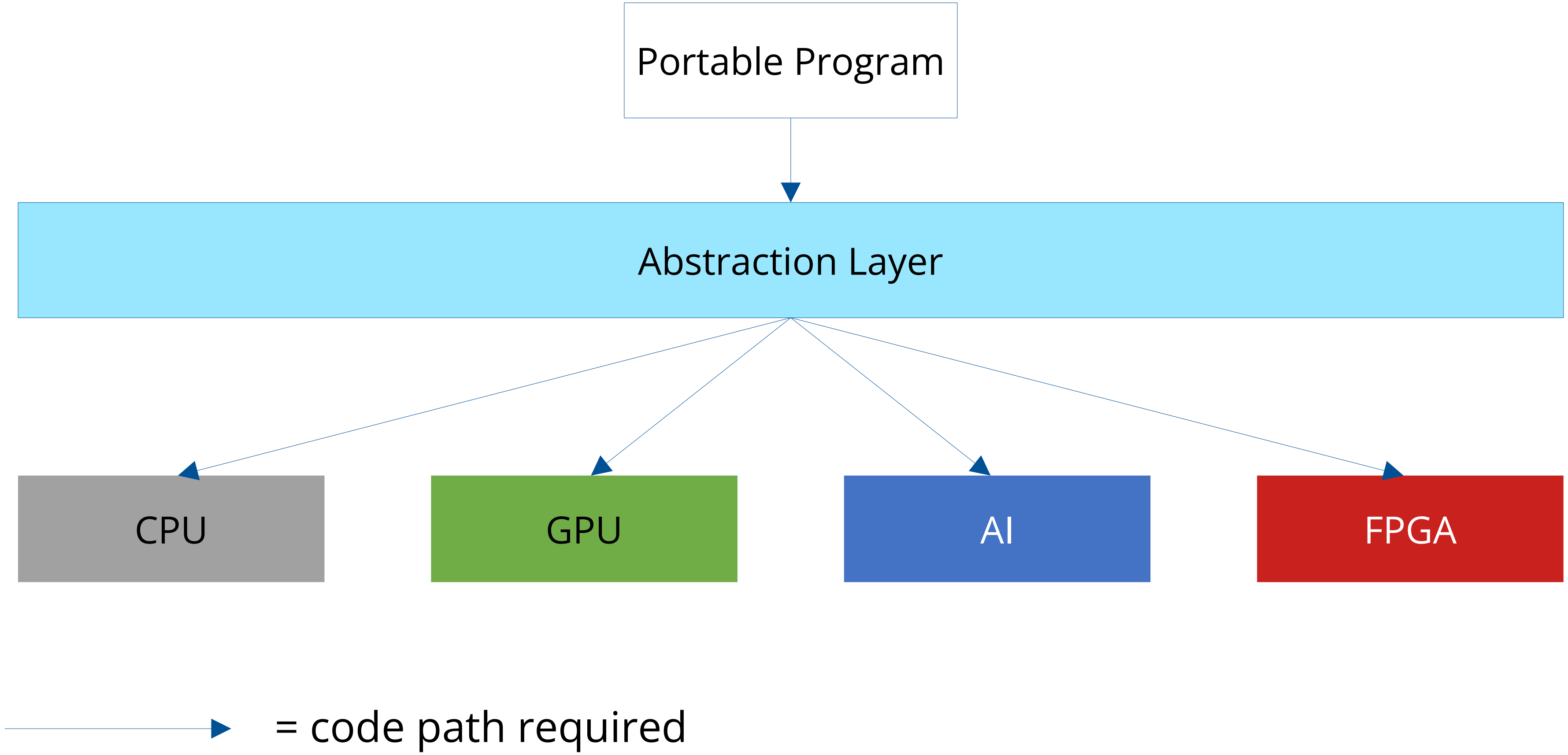


Controlling the heterogeneous landscape

Original Situation



Improved Situation



Available Libraries

Developed by Sandia National Laboratories (USA)

Developed by Lawrence Livermore National Laboratory (USA)

Designed by the Khronos industry consortium (USA)
Implemented by hardware vendors

Developed by Helmholtz-Zentrum Dresden-Rossendorf (Germany)

Available Libraries

kokkos — Developed by Sandia National Laboratories (USA)

RAJA — Developed by Lawrence Livermore National Laboratory (USA)

SYCL™ — Designed by the Khronos industry consortium (USA)
Implemented by hardware vendors

alpaka — Developed by Helmholtz-Zentrum Dresden-Rossendorf (Germany)

Introduction to alpaka

CASUS
CENTER FOR ADVANCED
SYSTEMS UNDERSTANDING

alpaka – Abstraction Library for Parallel Kernel Acceleration

alpaka

alpaka is…

- A parallel programming library: Accelerate your code by exploiting your hardware‘s parallelism!
- An abstraction library: Create portable code that runs on CPUs and GPUs!
- Free & open-source software

Introduction to alpaka

Programming with alpaka

- C++ only!
- Header-only library: No additional runtime dependency introduced
- Modern library: alpaka is written entirely in C++17
- Supports a wide range of modern C++ compilers (g++, clang++, Apple LLVM, MS Visual Studio)
- Portable across operating systems: Linux, macOS, Windows are supported

Introduction to alpaka

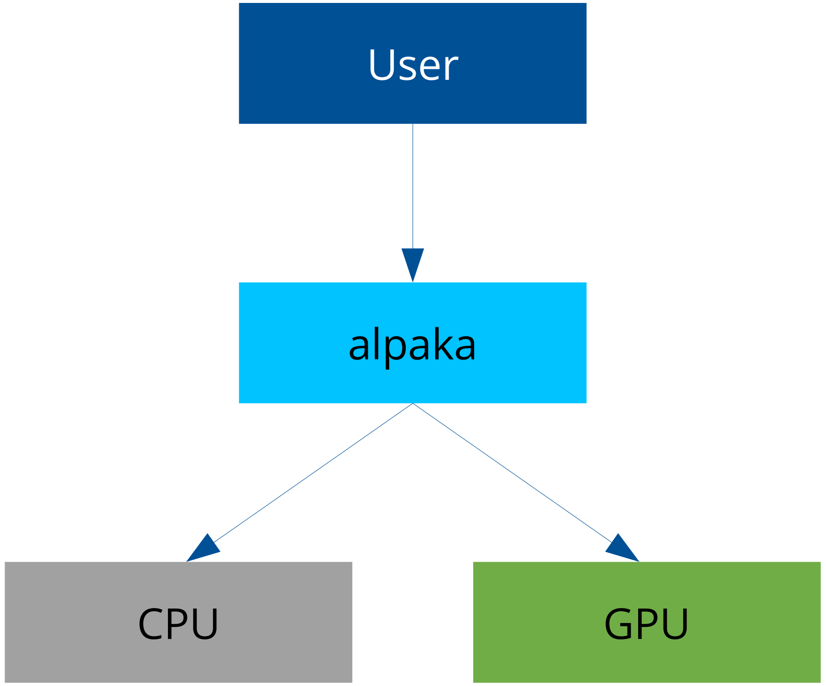
alpaka's purpose

Without alpaka

- Multiple hardware types commonly used (CPUs, GPUs, ...)
- Increasingly heterogeneous hardware configurations available
- Platforms not inter-operable → parallel programs not easily portable

alpaka: one API to rule them all

- Abstraction (not hiding!) of the underlying hardware & software platforms
- Code needs only minor adjustments to support different accelerators

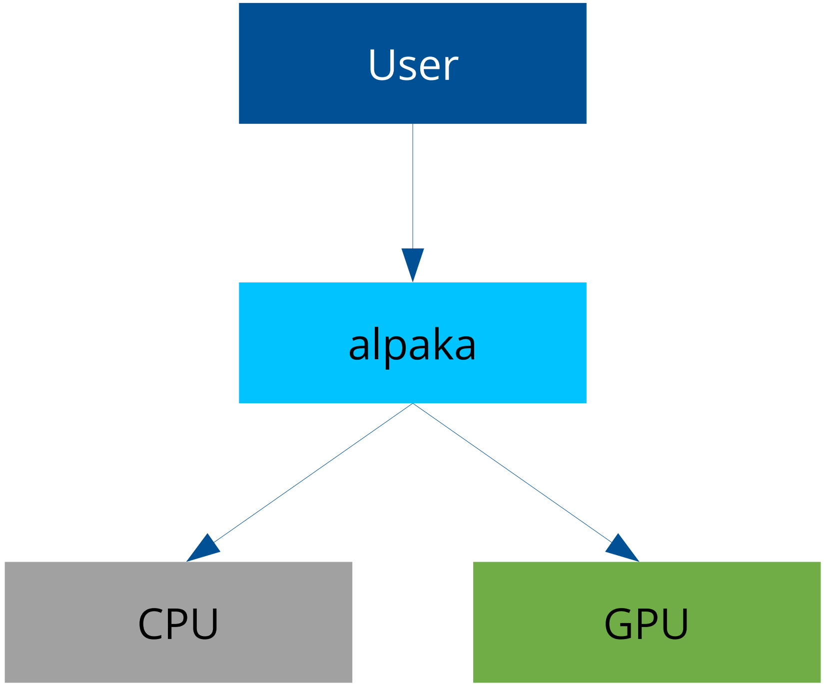


Portable Heterogeneous Parallel Programming

CASUS
CENTER FOR ADVANCED
SYSTEMS UNDERSTANDING

alpaka enables portability!

- Idea: Write algorithms once…
  - … independently of target architecture
  - … independently of available programming models
- Decision on target platform made during compilation
  - Choosing another platform just requires another compilation pass
- alpaka defines an abstract programming model
- alpaka utilizes C++17 to support many architectures
  - CUDA, HIP, OpenMP, TBB, …



The alpaka Library

alpaka's design

<table>
<tr><td>Domain</td><td colspan="6">Host</td><td colspan="2">Device</td></tr>
<tr><td rowspan="2">Origin</td><td colspan="6">alpaka</td><td rowspan="2">User</td><td>alpaka</td></tr>
<tr><td colspan="6">User</td><td>User</td></tr>
<tr><td>Concept</td><td>Device Manager</td><td>Device</td><td>Queue</td><td>Event</td><td>Buffer</td><td>Task</td><td>Kernel</td><td>Accelerator</td></tr>
</table>

Portable Heterogeneous Parallel Programming

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

alpaka enables full utilization of heterogeneous systems!

- Algorithms are generally independent of chosen target architecture

```
auto const taskCpu = alpaka::createTaskKernel<AccCpu>(workDivCpu, kernel, …);
auto const taskGpu = alpaka::createTaskKernel<AccGpu>(workDivGpu, kernel, …);
```

- Optimization for specific architecture is still possible

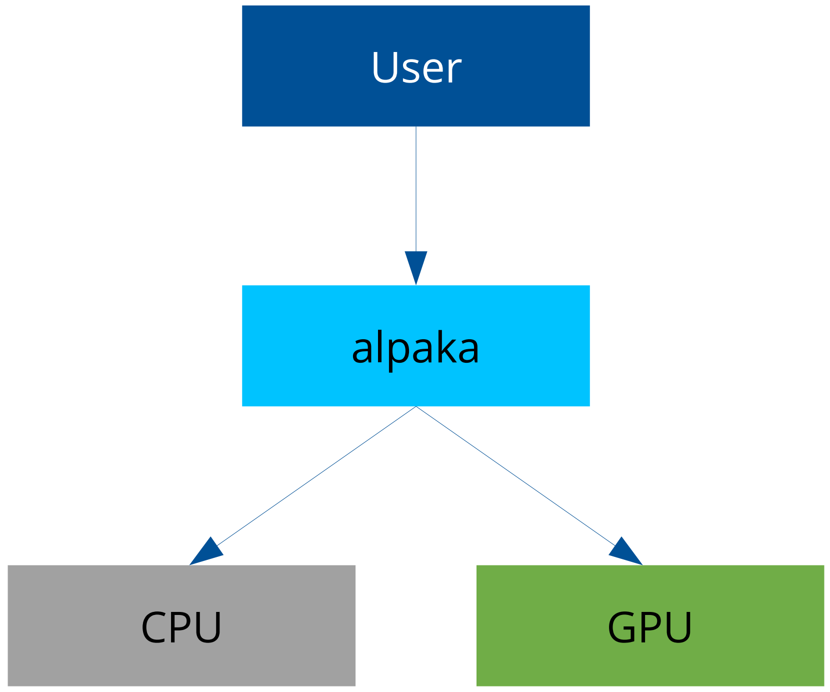
```
// general case
template <typename TAcc>
void computationallyIntensiveFunction(TAcc const & acc) { … };

// specialization for AccGpu
template <>
void computationallyIntensiveFunction<AccGpu>(AccGpu const & acc) { … };
```

Changing the Back-end

Moving from CPU to GPU

alpaka allows for easy …

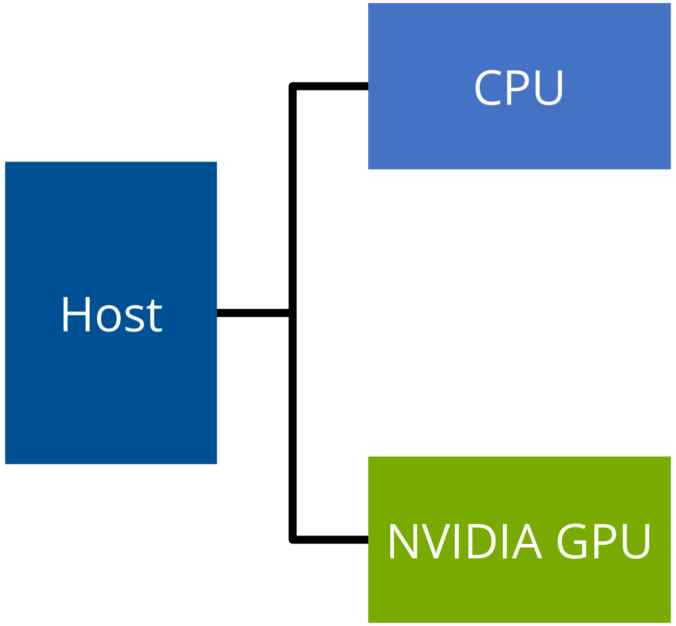
- … exchange of the accelerator
- … porting of programs across accelerators
- … experimentation with different devices
- … mixing of accelerator types



Programming Heterogeneous Systems

Heterogeneous Systems

- Real-world scenario: Use all available compute power
- Also real-world scenario: Multiple different hardware types available
- Requirement: Usage of one back-end per hardware platform
- Requirement: Back-ends need to be interoperable



Programming Heterogeneous Systems

CASUS
CENTER FOR ADVANCED
SYSTEMS UNDERSTANDING

Using multiple Platforms

- alpaka enables easy heterogeneous programming!
- Create one Accelerator per back-end
- Acquire at least one Device per Accelerator
- Create one Queue per Device

```
// Define Accelerators
using AccCpu = AccCpuOmp2Blocks<Dim, Idx>;
using AccGpu = AccGpuCudaRt<Dim, Idx>;

// Acquire Devices
auto devCpu = getDevByIdx<AccCpu>(0u);
auto devGpu = getDevByIdx<AccGpu>(0u);

// Create Queues
using QueueProperty = property::NonBlocking;
using QueueCpu = Queue<AccCpu, QueueProperty>;
using QueueGpu = Queue<AccGpu, QueueProperty>;

auto queueCpu = QueueCpu{devCpu};
auto queueGpu = QueueGpu{devGpu};
```

Programming Heterogeneous Systems

CASUS
CENTER FOR ADVANCED
SYSTEMS UNDERSTANDING

Communication

- Buffers are defined and created per Device
- Buffers can be copied between different Devices / Queues
- Not restricted to a single platform!
- **Restriction**: CPU to GPU copies (and vice versa) require GPU queue

```
// Allocate buffers
auto bufCpu = allocBuf<float, Idx>(devCpu, extent);
auto bufGpu = allocBuf<float, Idx>(devGpu, extent);

/* Initialization … */

// Copy buffer from CPU to GPU - destination comes first
memcpy(gpuQueue, bufGpu, bufCpu, extent);

// Execute GPU kernel
enqueue(gpuQueue, someKernelTask);

// Copy results back to CPU and wait for completion
memcpy(gpuQueue, bufCpu, bufGpu, extent);

// Wait for GPU, then execute CPU kernel
wait(cpuQueue, gpuQueue);
enqueue(cpuQueue, anotherKernelTask);
```

Programming Heterogeneous Systems

Heterogeneous programming with alpaka

- alpaka gives you access to all of your system’s computation resources
- alpaka eases programming for different device types
- alpaka enables simple data transfers between different devices
- alpaka makes your code reusable
- alpaka makes your code portable

Write once, scale everywhere!

The alpaka Library

CASUS
CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

alpaka is free software (MPL 2.0). Find us on GitHub!

Our GitHub organization: **https://www.github.com/alpaka-group**

- Contains all alpaka-related projects, documentation, samples, …
- New contributors welcome!

The library: **https://www.github.com/alpaka-group/alpaka**

- Full source code
- Issue tracker
- Installation instructions
- Small examples

Heterogeneous Programming With the Caravan Ecosystem

I already have a CUDA program. Do I really need to port everything?

- No. Try our CUDA portability layer *cupla*.
- Kernels need to be ported to alpaka-style kernels
- `cudaApiCall()` becomes `cuplaApiCall()`
- **https://github.com/alpaka-group/cupla**

Heterogeneous Programming With the Caravan Ecosystem

How can I easily switch between different memory layouts?

- Example: From array-of-struct to struct-of-array and back
- Problem: Changing memory layout requires changing of algorithm
- Solution: LLAMA
- **https://github.com/alpaka-group/llama**

Heterogeneous Programming With the Caravan Ecosystem

CASUS
CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

But I just want to do transform & reduce!

- Solution: vikunja
- More standard algorithms planned soon
- **https://github.com/alpaka-group/vikunja**

vikunja

CASUS

CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

www.casus.science