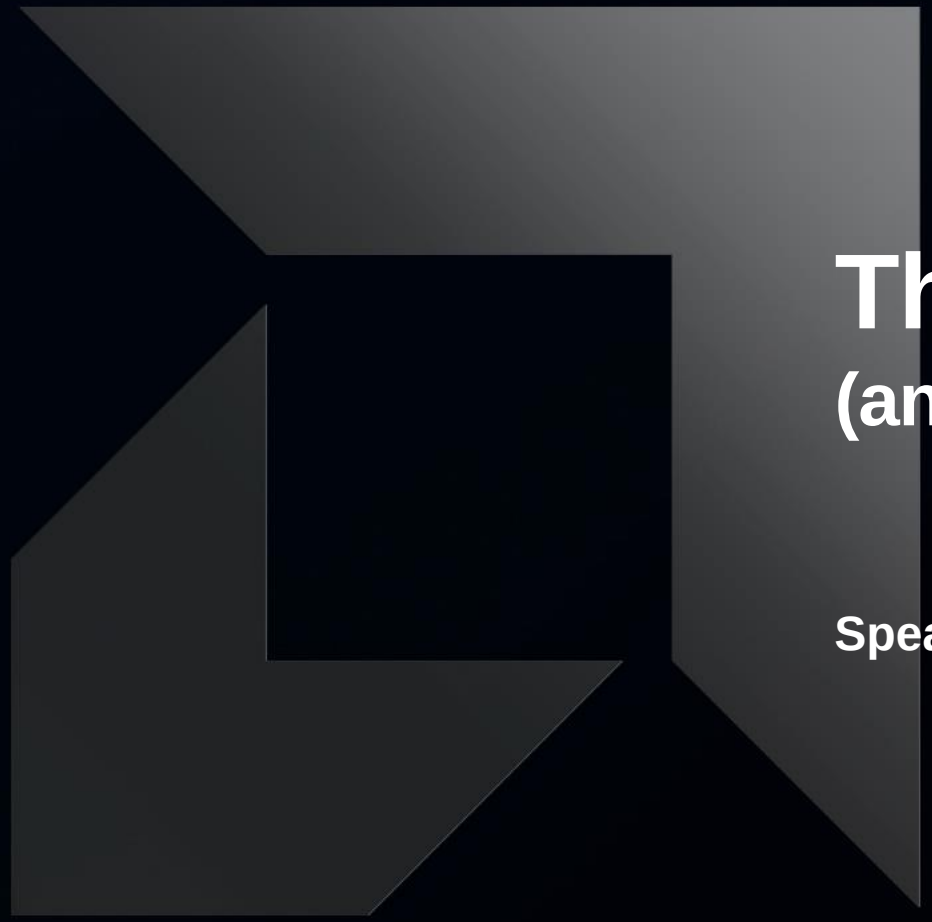


The AMD ROCm™ Platform
(and its GPU programming models)

Speaker: Michael Klemm
Principal Member of Technical Staff
Compilers, Languages, Runtimes & Tools
Machine Learning & Software Engineering

AMD
together we advance_

Developing for AMD Hardware

AMD EPYC

AMD Optimized CPU Compiler (AOCC)
AMD Optimized CPU Libraries (AOCL)
AMD ZenDNN
AMD µProf

AMD ROCm

Heterogenous-computing Interface for Portability (HIP)
OpenMP API
Machine Learning Frameworks
Acceleration Libraries
ROCm™ Communication Libraries (RCCL)

AMD EPYC AMD INSTINCT

In addition to numerous options with open source, community tools

AMD Compilers

AMD EPYC

- **AMD Optimizing C/C++ Compiler (AOCC)**
- **Targets x86 AMD CPUs (no offloading)**

- C, C++ and Fortran compilers based on LLVM with extensive optimizations for AMD EPYC™ processors

AMD INSTINCT

- **ROCm™ Compiler Collection**
- **Supports offloading to AMD GPUs**

- C, C++ and Fortran compilers based on LLVM with additional open-source features and optimizations

AMD Next-Gen Fortran Compiler – Public Downloads

Introducing AMD's Next-Gen Fortran Compiler

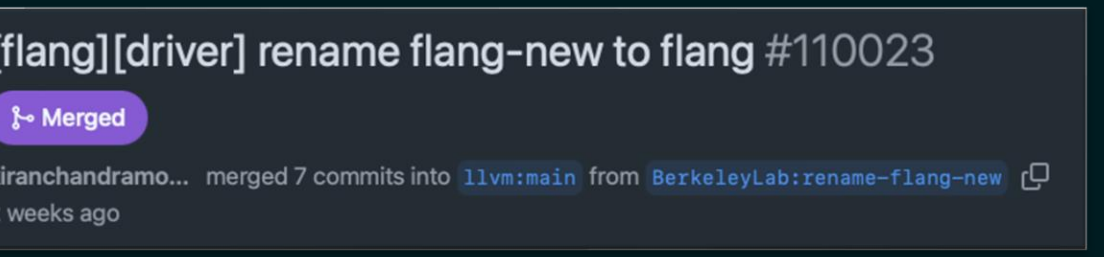
We are excited to share a brief preview of AMD's Next-Gen Fortran Compiler, our new open source Fortran complier supporting OpenMP offloading. AMD's Next-Gen Fortran Compiler is a downstream flavor of LLVM Flang, optimized for AMD GPUs. Our Next-Gen Fortran Compiler enables OpenMP offloading and offers a direct interface to ROCm and HIP. In this blog post you will:

1. Learn how to use AMD's Next-Gen Fortran Compiler to deploy and accelerate your Fortran codes on AMD GPUs using OpenMP offloading.
2. Learn how to use AMD's Next-Gen Fortran Compiler to interface and invoke HIP and ROCm kernels.
3. See how AMD's Next-Gen Fortran Compiler OpenMP offloading exhibits competitive performance against native HIP/C++ codes, benchmarking on AMD GPUs.
4. Learn how to access a pre-production build of the new AMD's Next-Gen Fortran Compiler.

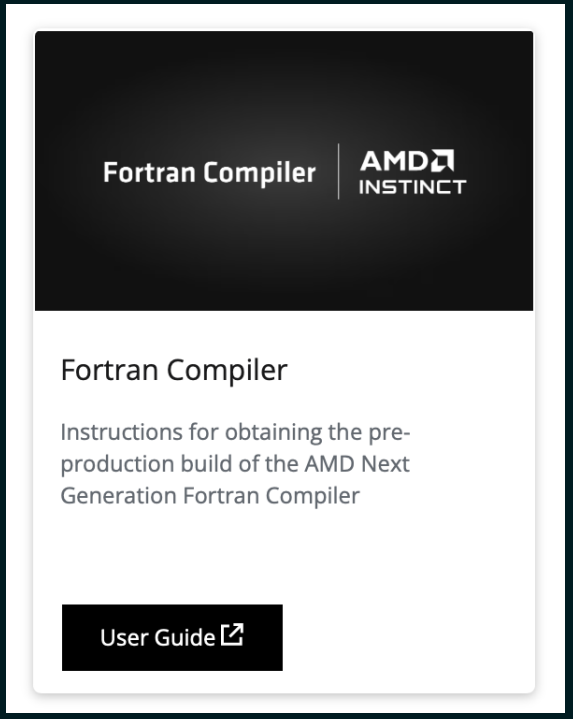
Our commitment to Fortran

Fortran is a powerful programming language for scientific and engineering high performance computing applications and is core to many, some very crucial, HPC codebases. Fortran remains under active development as a standard, supporting both legacy and modern codebases. The need for a more modern Fortran compiler motivated the creation of the LLVM Flang project and AMD fully supports that path. In following with community trends, AMD's Next-Gen Fortran Compiler will be a downstream flavor of LLVM Flang and will in time supplant the current AMD Flang compiler, a downstream flavor of "Classic Flang".

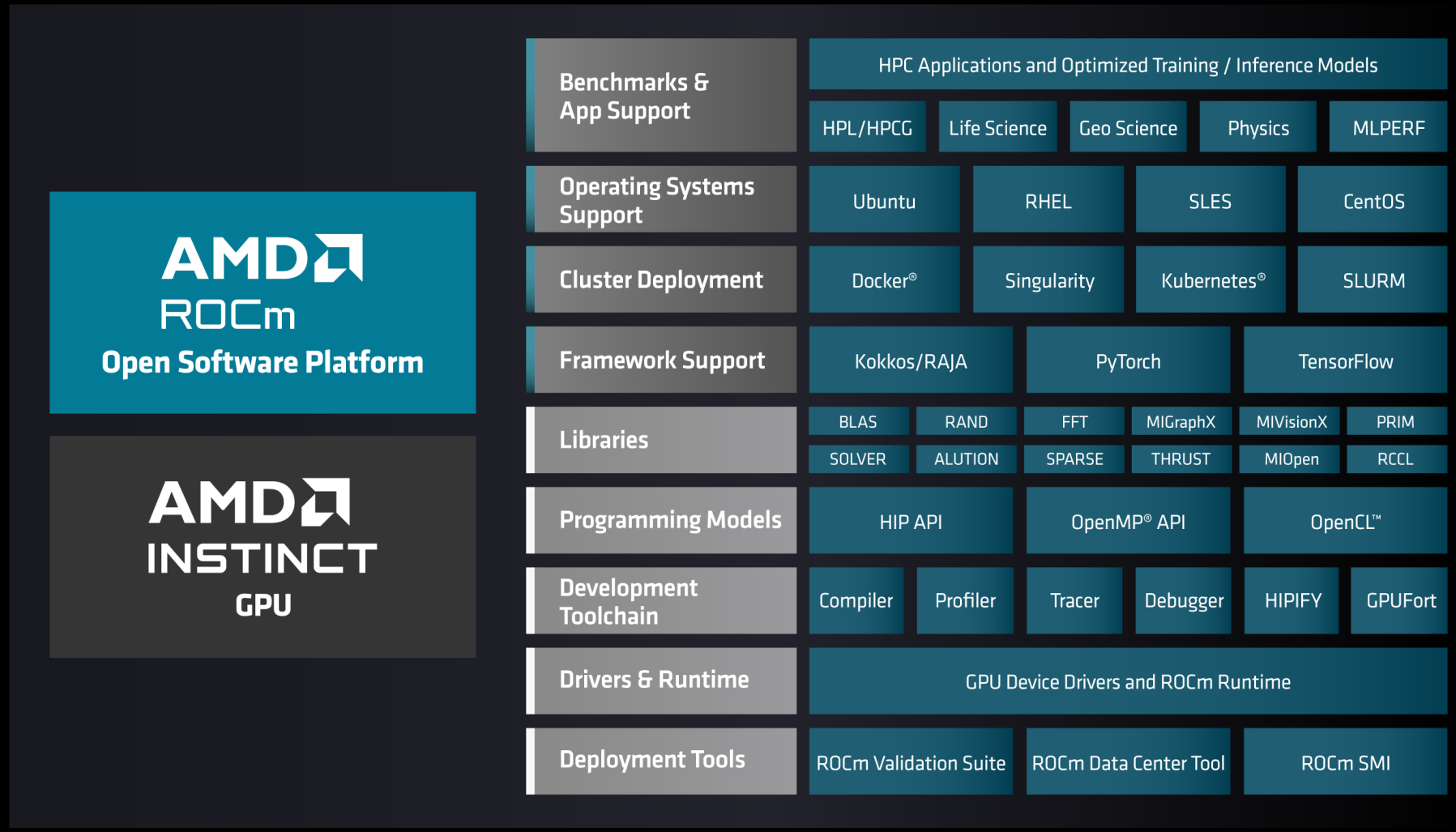
https://rocm.blogs.amd.com

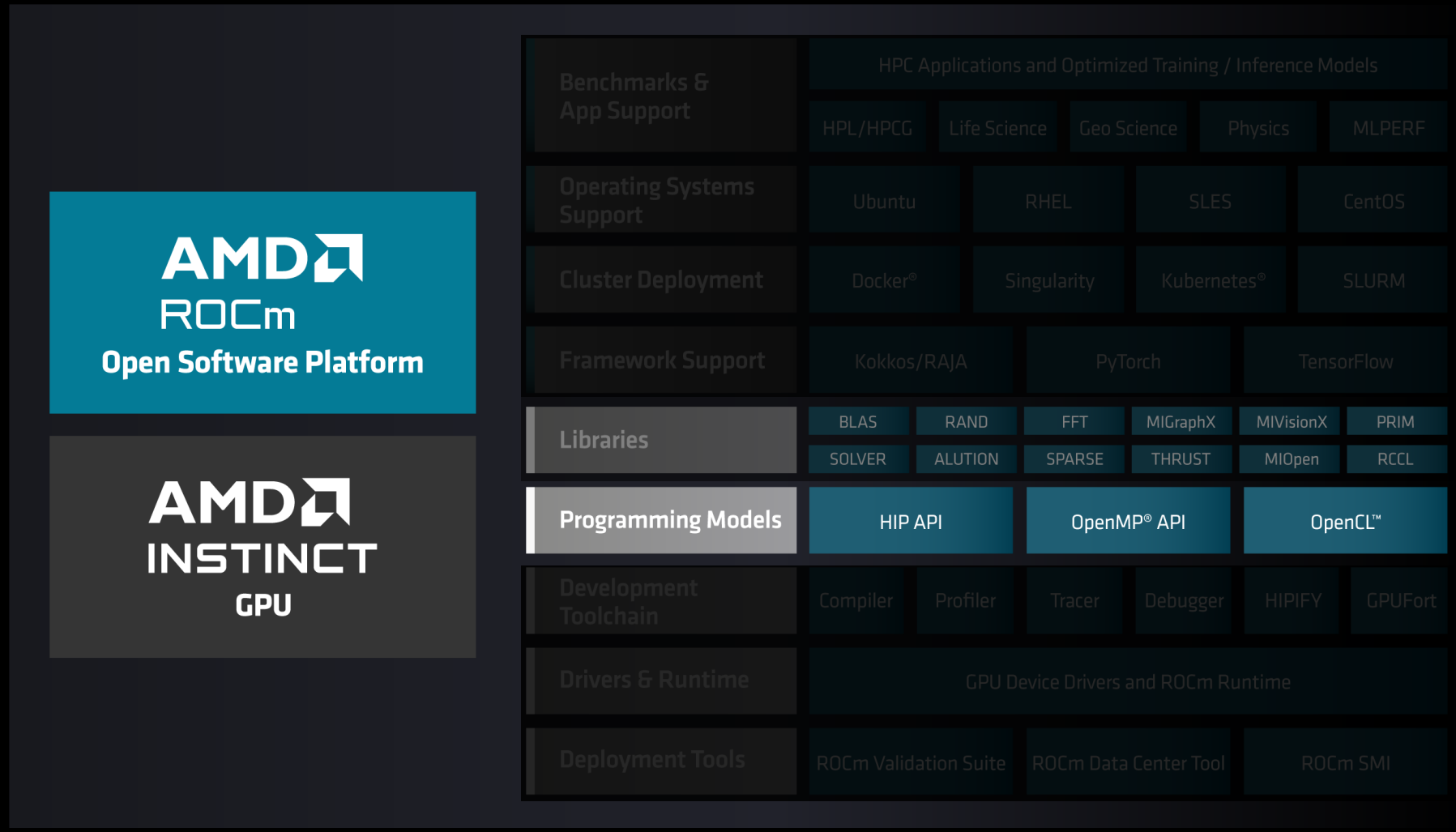


https://github.com/llvm/llvm-project/



Infinity Hub Tile





ROCm™ Programming Models

HIP	Grid Language	Maximum Control
OpenMP®	Fork-Join Model	Standardized
OpenCL™	Grid Language	Standardized

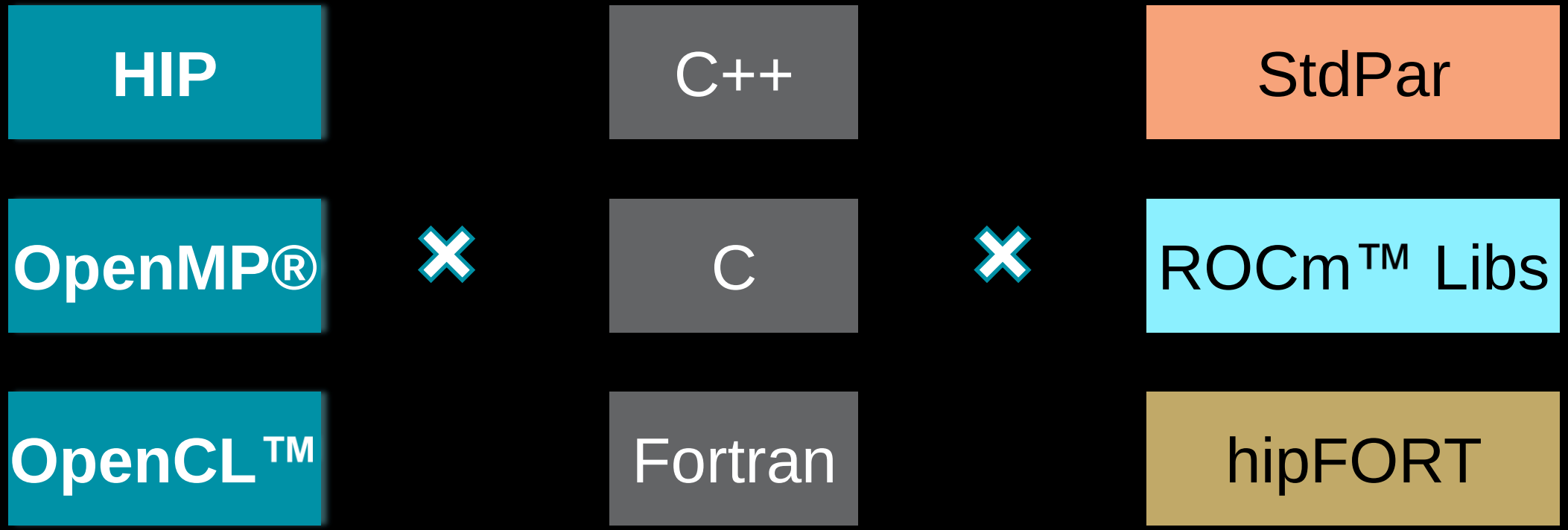
ROCm™ Programming Models

HIP

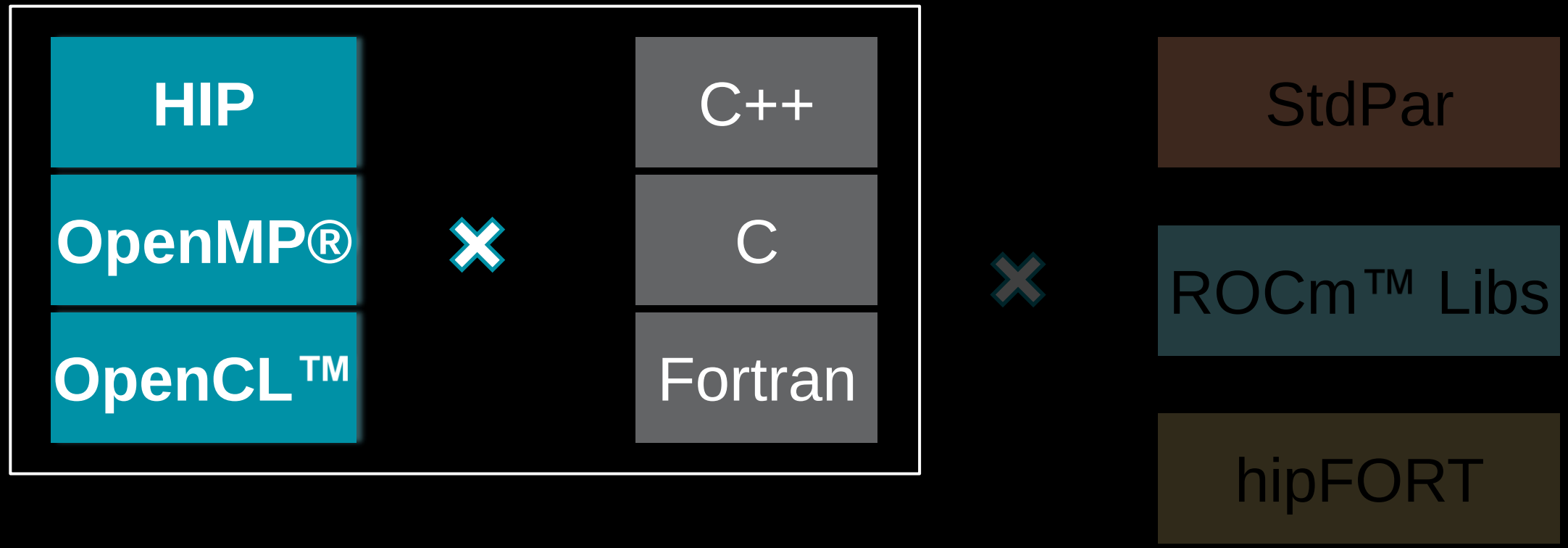
OpenMP®

OpenCL™

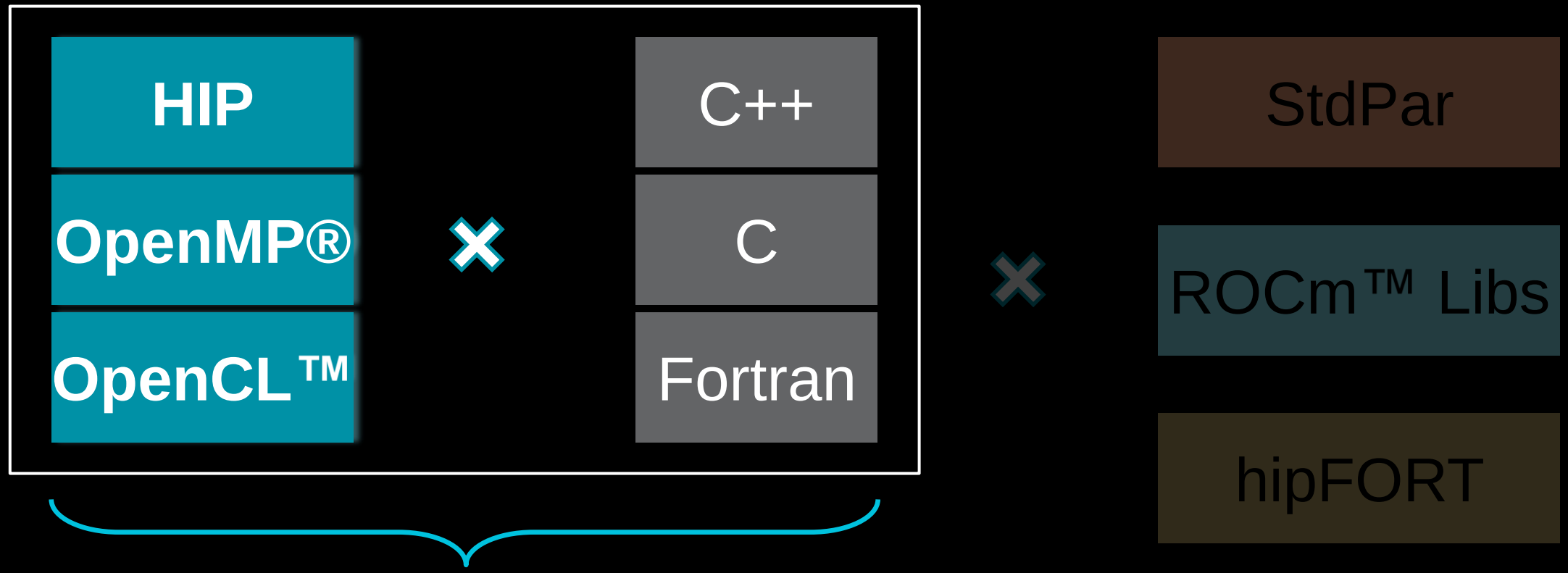
ROCm™ Programming Models



ROCm™ Programming Models



ROCm™ Programming Models



Common LLVM Compiler Backend

ROCm™ Programming Models

HIP

C++

StdPar

OpenMP®

C

ROCm™ Libs

OpenCL™

Fortran

hipFORT

ROCm™ Programming Models

HIP

C++

StdPar

OpenMP®

C

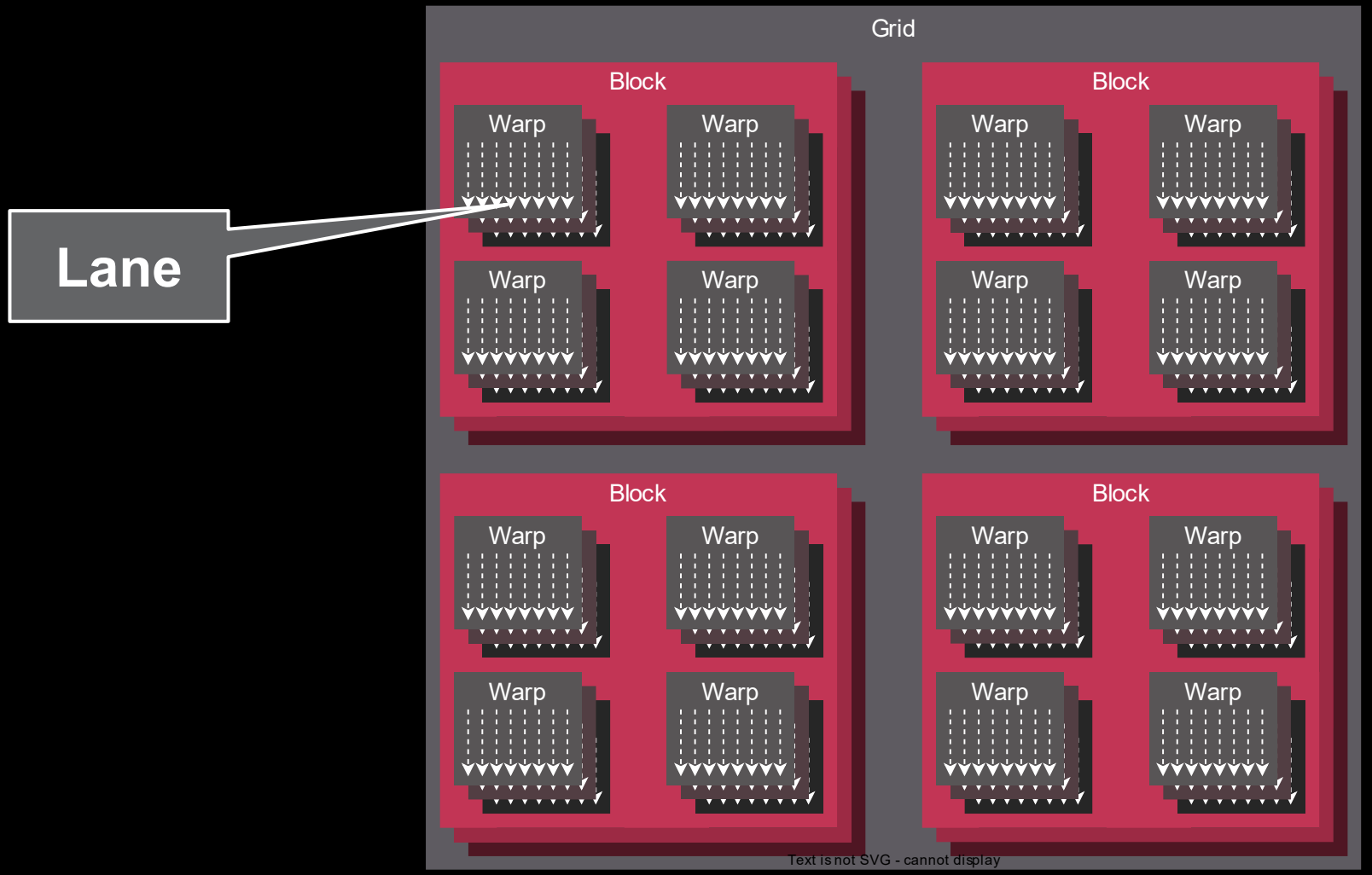
ROCm™ Libs

OpenCL™

Fortran

hipFORT

HIP Grid Fundamentals



HIP Kernel Example

```
__global__
void saxpy(float a, const float* d_x, float* d_y, unsigned int size) {

  const unsigned int global_idx = blockIdx.x * blockDim.x + threadIdx.x;

  if (global_idx < size)
    d_y[global_idx] = a * d_x[global_idx] + d_y[global_idx];
}
```

HIP Kernel Example

```
__global__
void saxpy(float a, const float* d_x, float* d_y, unsigned int size) {

  const unsigned int global_idx = blockIdx.x * blockDim.x + threadIdx.x;

  if (global_idx < size)
    d_y[global_idx] = a * d_x[global_idx] + d_y[global_idx];
}
```

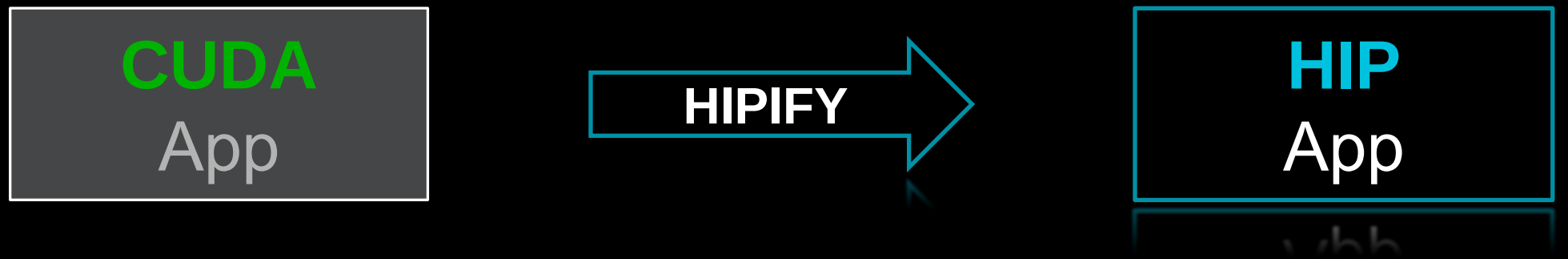
HIP Kernel Launch Example

```
float* d_x{}; float* d_y{};

hipMalloc(&d_x, size_bytes);
hipMalloc(&d_y, size_bytes);
hipMemcpy(d_x, x.data(), size_bytes, hipMemcpyHostToDevice);
hipMemcpy(d_y, y.data(), size_bytes, hipMemcpyHostToDevice);

saxpy <<<dim3(grid_size),
         dim3(block_size),
         0,
         hipStreamDefault>>> (a, d_x, d_y, size);
```

HIP Porting From CUDA



HIP supports AMDGPU and CUDA

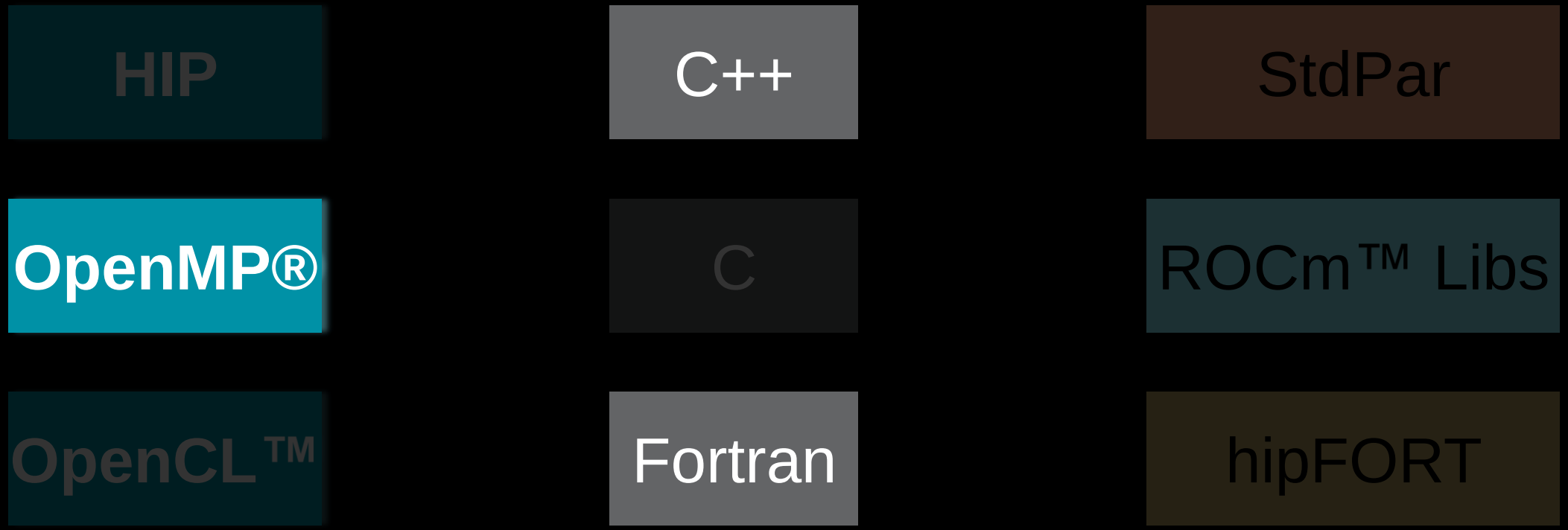
Allows incremental porting

HIP Porting From CUDA

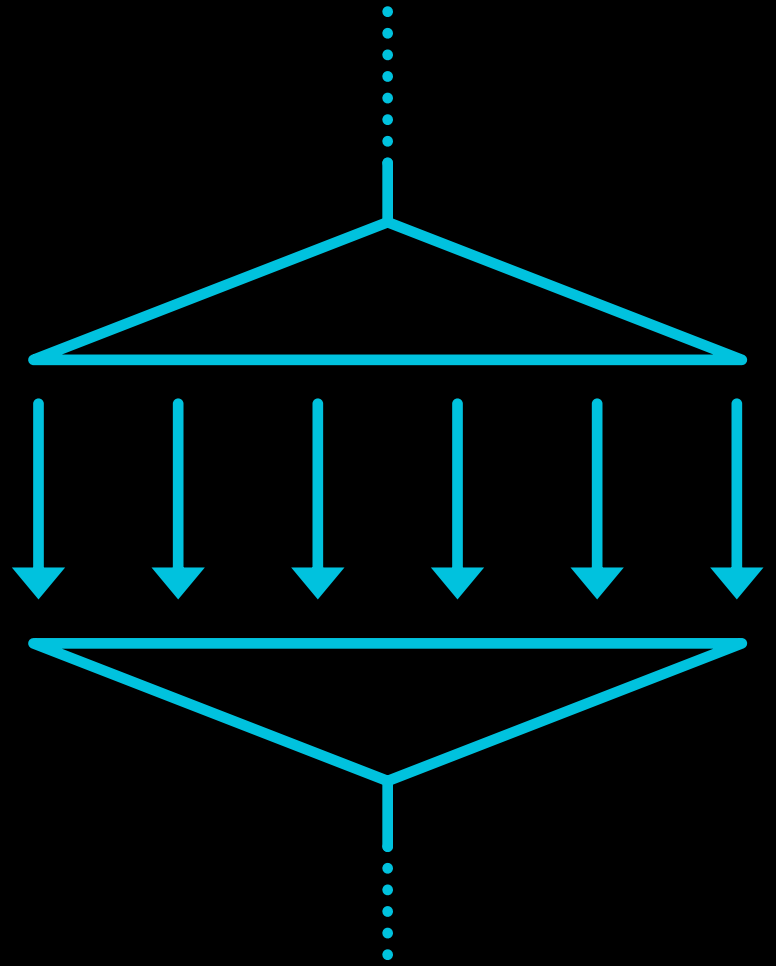
Text-based **HIP** translation

Compiler-based **HIP** translation

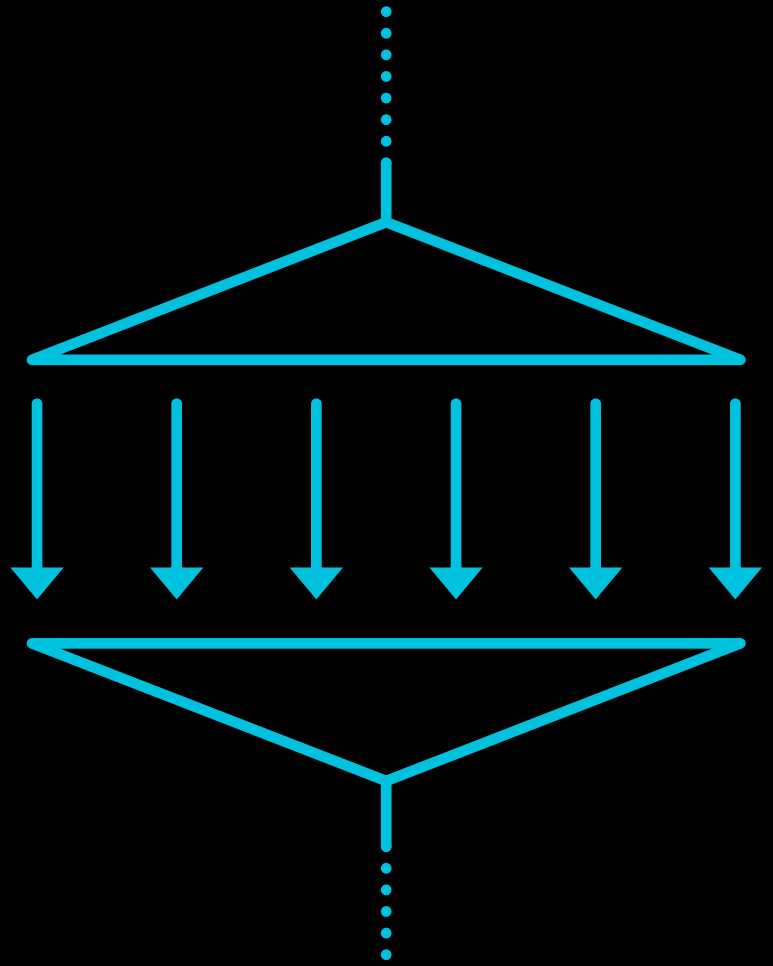
ROCm™ Programming Models



OpenMP® API Fundamentals



OpenMP® Fundamentals



Compiler

OpenMP® API and C++

```
void saxpy(float a, const float* x, float* y, unsigned int size) {

  #pragma omp target teams distribute parallel for \
              map(to: x[0:size]) map(tofrom: y[0:size])
  for (int i = 0; i < size; ++i)
    y[i] = a * x[i] + y[i];

}
```

OpenMP® API and C++ - Attribute Syntax

```
void saxpy(float a, const float* x, float* y, unsigned int size) {

  [[omp::directive(target teams loop \
                   map(to: x[0:size]) map(tofrom: y[0:size])]]
  for (int i = 0; i < size; ++i)
    y[i] = a * x[i] + y[i];

}
```

OpenMP® API and C++

```
int main() {
  float a = .8f;
  float *x = new float[size];
  float *y = new float[size];

  #pragma omp target teams distribute parallel for \
              map(to: x[0:size]) map(tofrom: y[0:size])
  for (int i = 0; i < size; ++i)
    y[i] = a * x[i] + y[i];

  return 0;
}
```

Initialization omitted for brevity.

Some more code omitted for brevity.

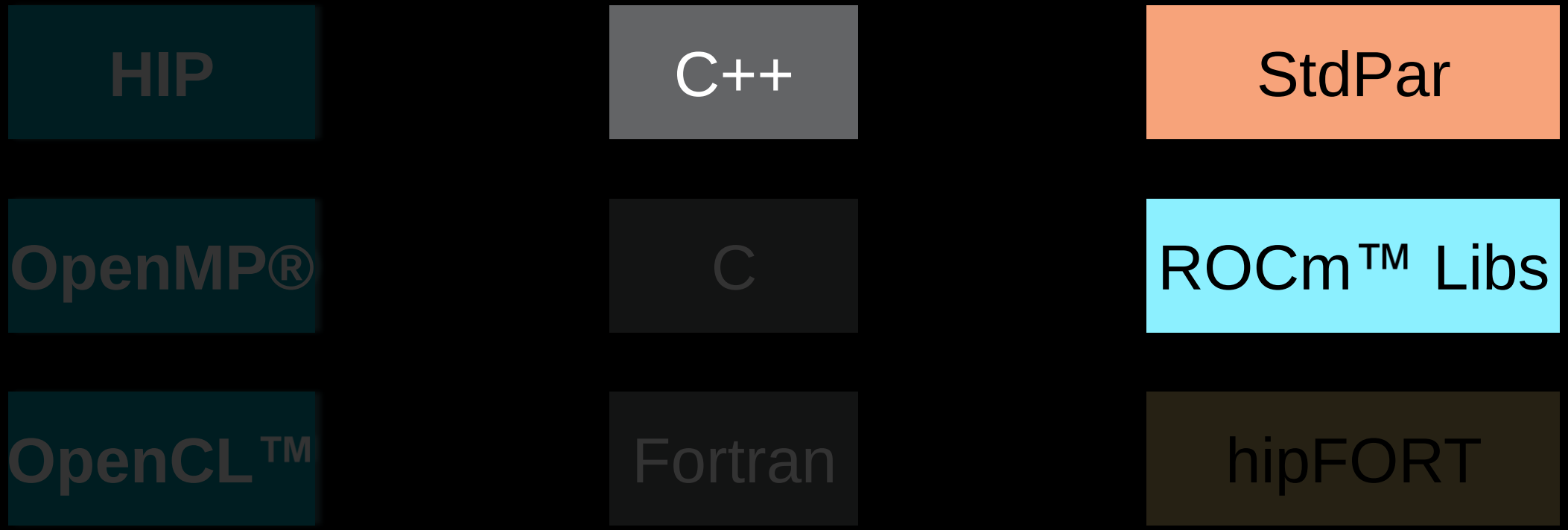
OpenMP® API and Fortran

```
subroutine saxpy(a, x, y, size)
  real :: a, x(size), y(size)
  integer :: size, i
  !$omp target teams distribute parallel do map(to:x) map(tofrom:y)
  do i = 1, size
    y(i) = a*x(i)+y(i)
  enddo
end subroutine saxpy
```

OpenMP® API and Fortran

```
subroutine saxpy(a, x, y, size)
  real :: a, x(size), y(size)
  integer :: size
  !$omp target teams workdistribute map(to:x) map(tofrom:y)
    y = a * x + y
  !$omp end target teams workdistribute
end subroutine saxpy
```

ROCm™ Programming Models



C++ and StdPar

```
void saxpy(float a, std::vector<float> &x, std::vector<float> &y) {

  std::transform(x.begin(), x.end(), y.begin(), y.begin(),
                 [a](float xi, float yi) { return a * xi + yi; });

}
```

C++ and StdPar

```
void saxpy(float a, std::vector<float> &x, std::vector<float> &y) {

  std::transform(std::execution::par_unseq,
                 x.begin(), x.end(), y.begin(), y.begin(),
                 [a](float xi, float yi) { return a * xi + yi; });

}
```

Fortran and “StdPar”

```
subroutine saxpy(a, x, y, size)
  real :: a, x(size), y(size)
  integer :: size, i

  do concurrent (i=1:size)
    y(i) = a*x(i)+y(i)
  enddo
end subroutine saxpy
```

Can be parallelized for OpenMP host threads (now) and AMD GPU (wip).

ROCm™ Libraries

```
rocblas_create_handle(&handle);

// -- Allocate and initialize/copy device d_x and d_y; h_alpha on host

rocblas_set_pointer_mode(handle, rocblas_pointer_mode_host);

rocblas_saxpy(handle, n, &h_alpha, d_x, incx, d_y, incy);

// -- Copy result back to host

rocblas_destroy_handle(handle);
```

ROCm™ Libraries

```
rocblas_create_handle(&handle);

// -- Allocate and initialize/copy device d_x and d_y; h_alpha on host

rocblas_set_pointer_mode(handle, rocblas_pointer_mode_host);

rocblas_saxpy(handle, n, &h_alpha, d_x, incx, d_y, incy);

// -- Copy result back to host

rocblas_destroy_handle(handle);
```

ROCm™ Programming Models

HIP

C++

StdPar

OpenMP®

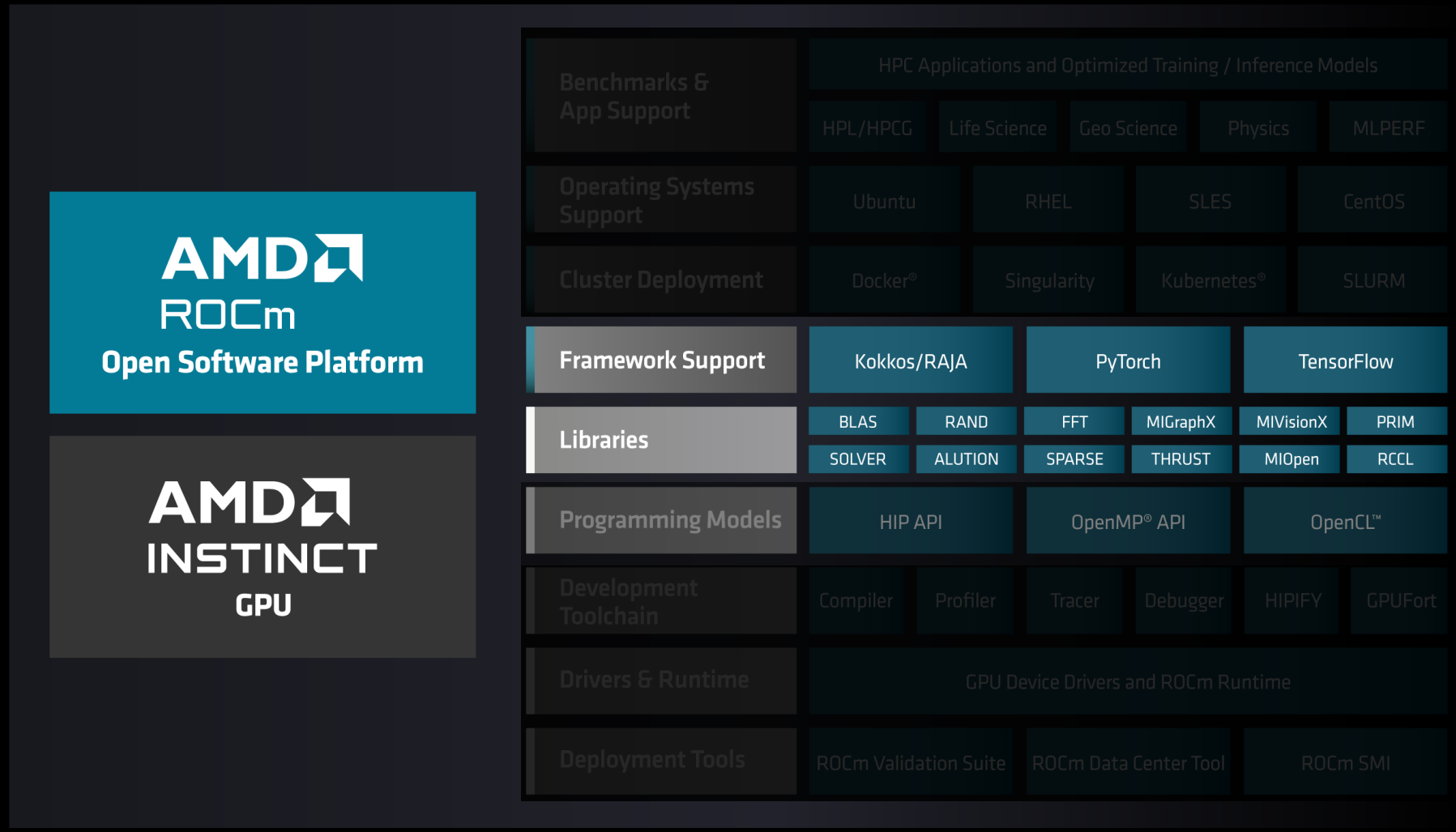
C

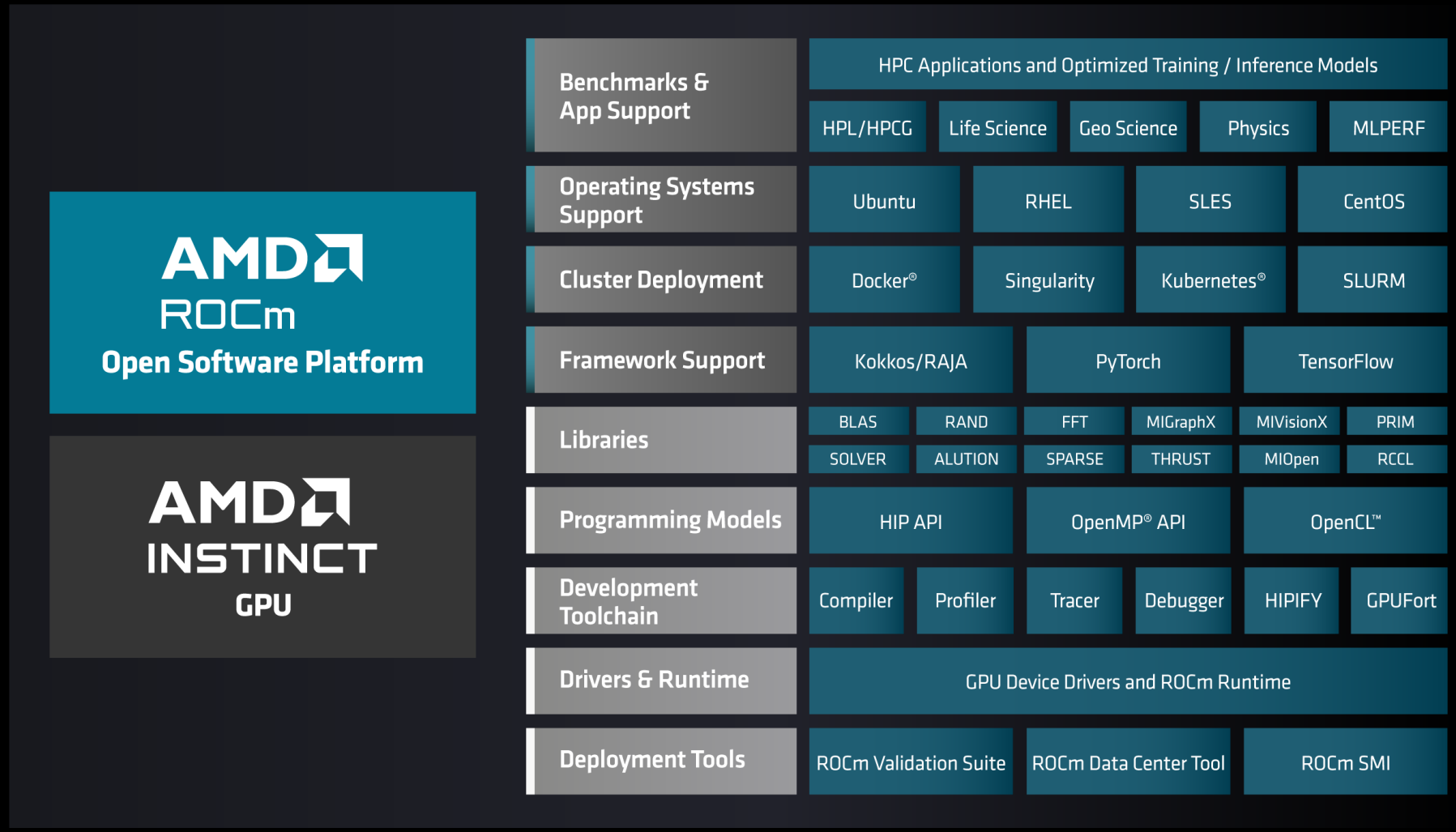
ROCm™ Libs

OpenCL™

Fortran

hipFORT



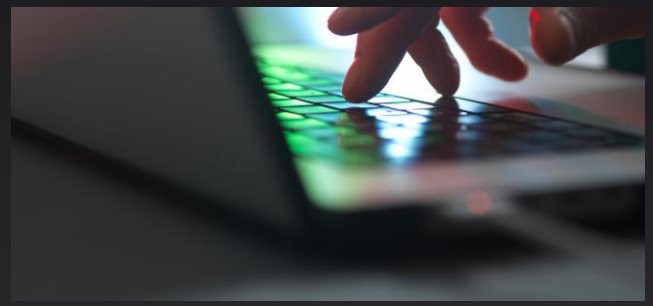


AMD ROCm™ Platform

High Performance

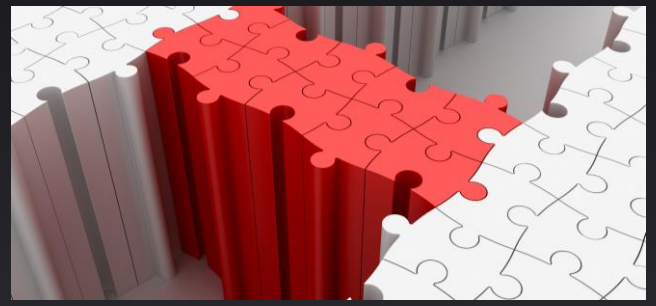
- Powers the top500 list leader
- Solutions for HPC and AI
- Compilers, Libraries, Frameworks

Open Source



- Committed to open ecosystem
- Active community engagement
- Driving development

Portable



- Portable / Standardized languages
- Solutions for evolving accelerators
- Support via third-party libraries

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD Arrow logo, EPYC, Instinct, ROCm and combinations thereof are trademarks of Advanced Micro Devices, Inc. PCIe is a registered trademark of PCI-SIG Corporation. OpenCL is a trademark of Apple Inc. used by permission by Khronos Group, Inc. The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.


AMD