Programming Models with the ROCm™ Compiler

Speaker: JP Lehr

AMD
together we advance_

AMD Compilers

AMD INSTINCT

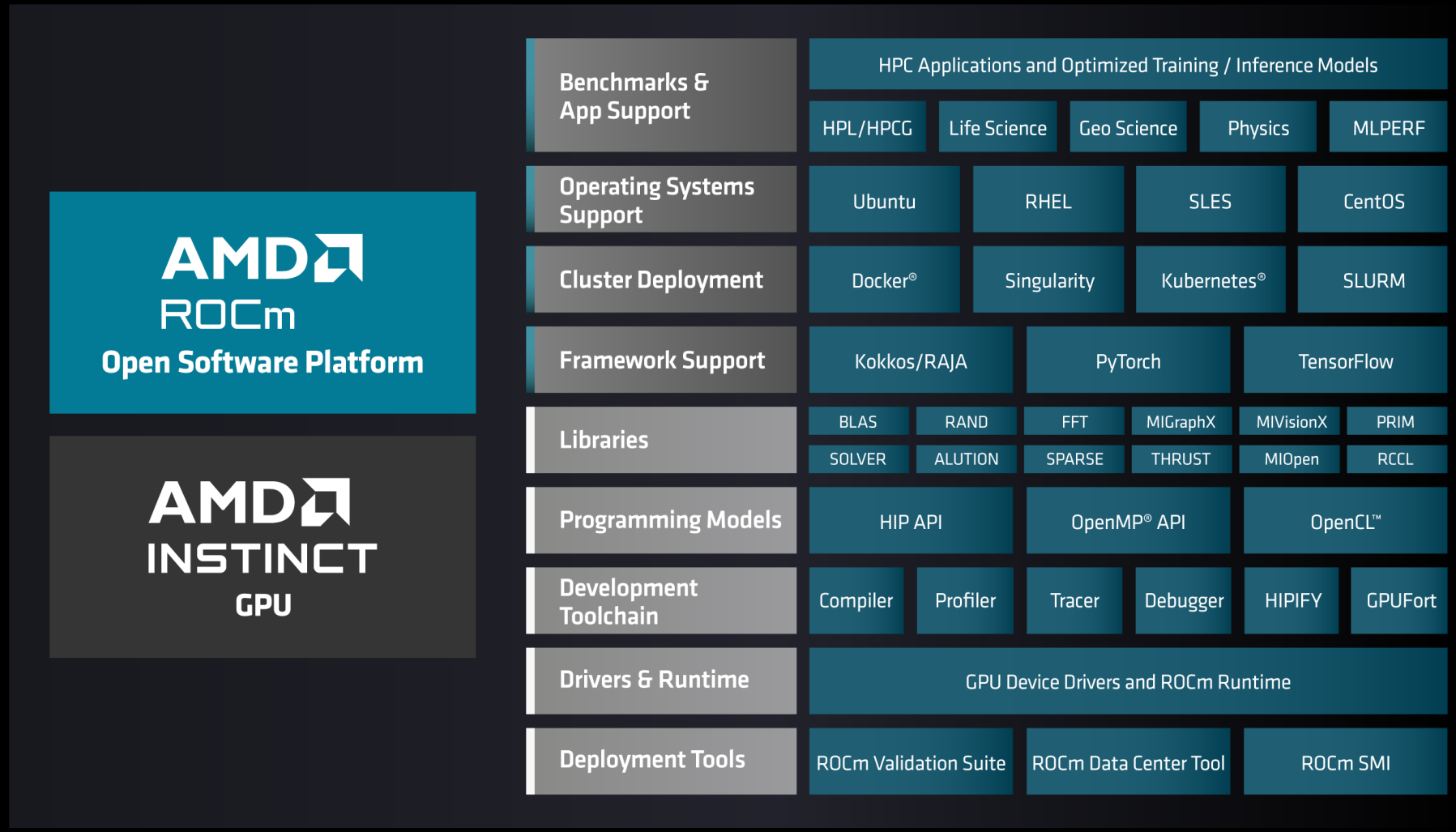
- **ROCm™ Compiler Collection**
- **Supports offloading to AMD GPUs**

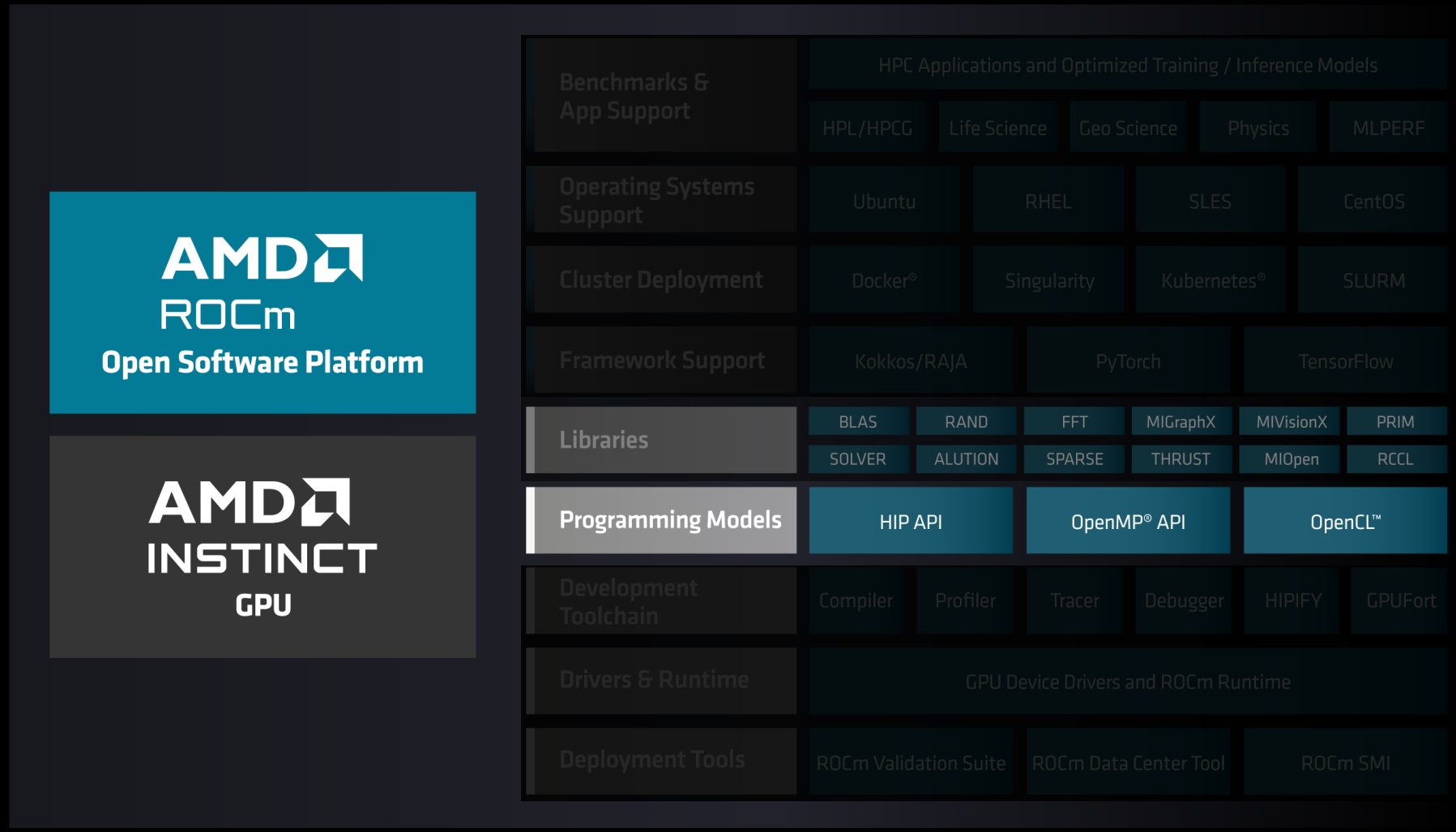
- C, C++ and Fortran compilers based on LLVM with additional open-source features and optimizations

AMD 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





ROCm™ Programming Models

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

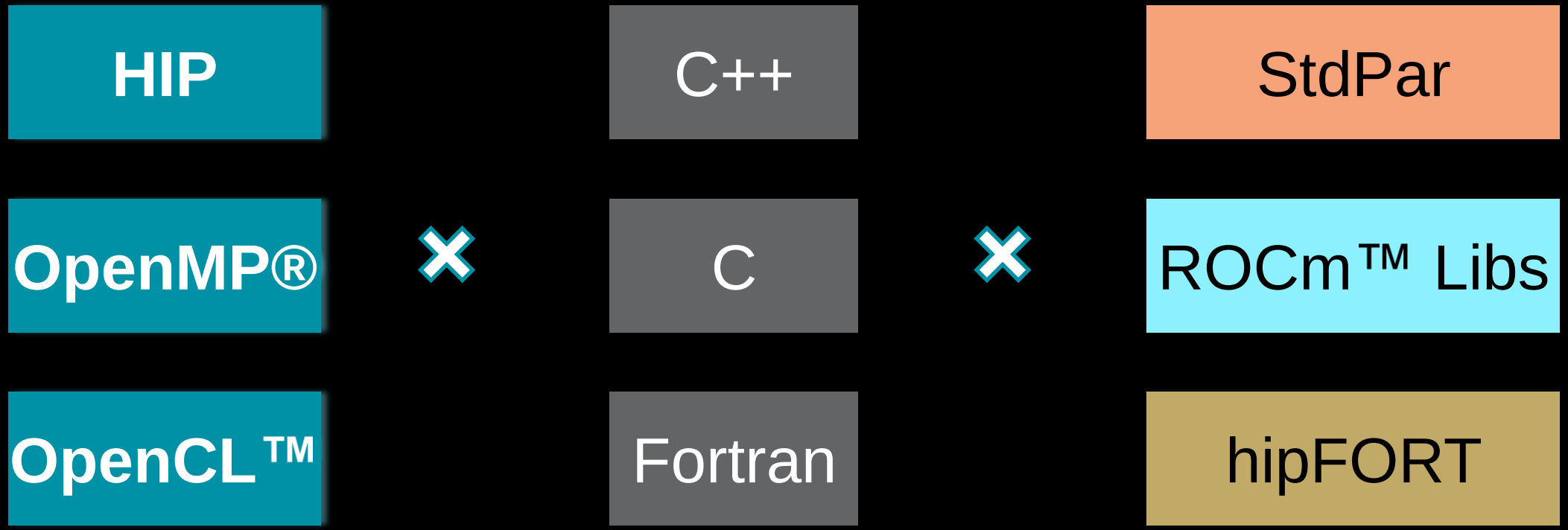
ROCm™ Programming Models

HIP

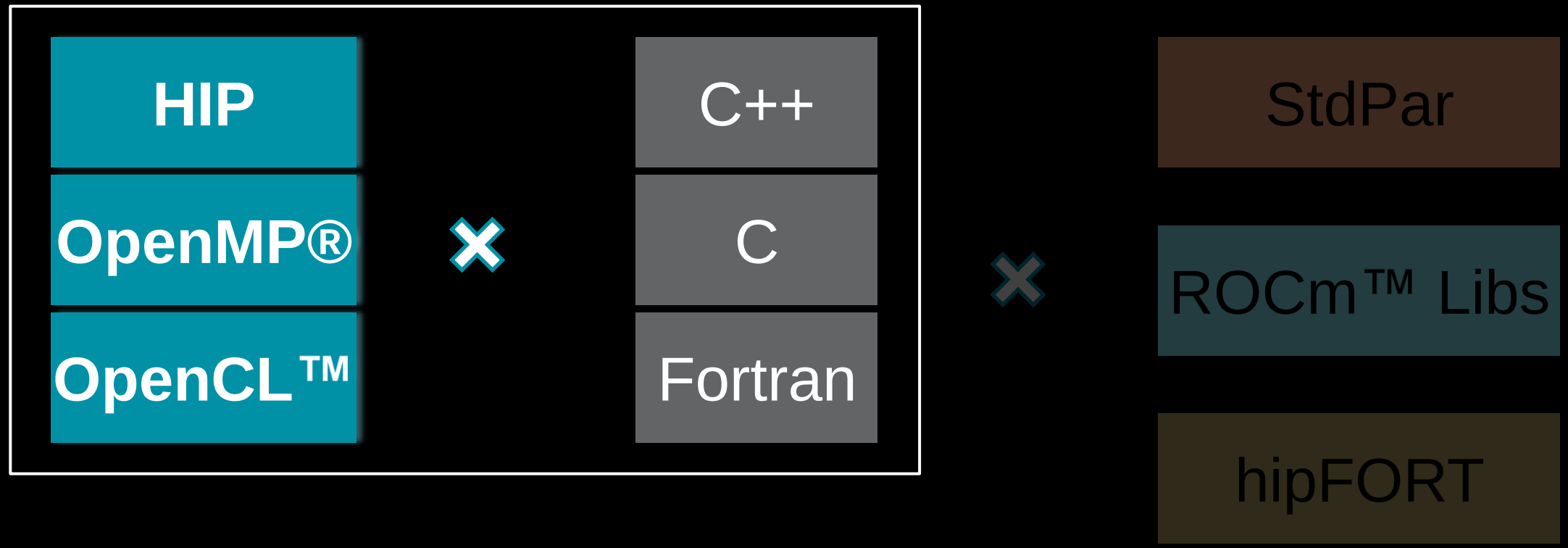
OpenMP®

OpenCL™

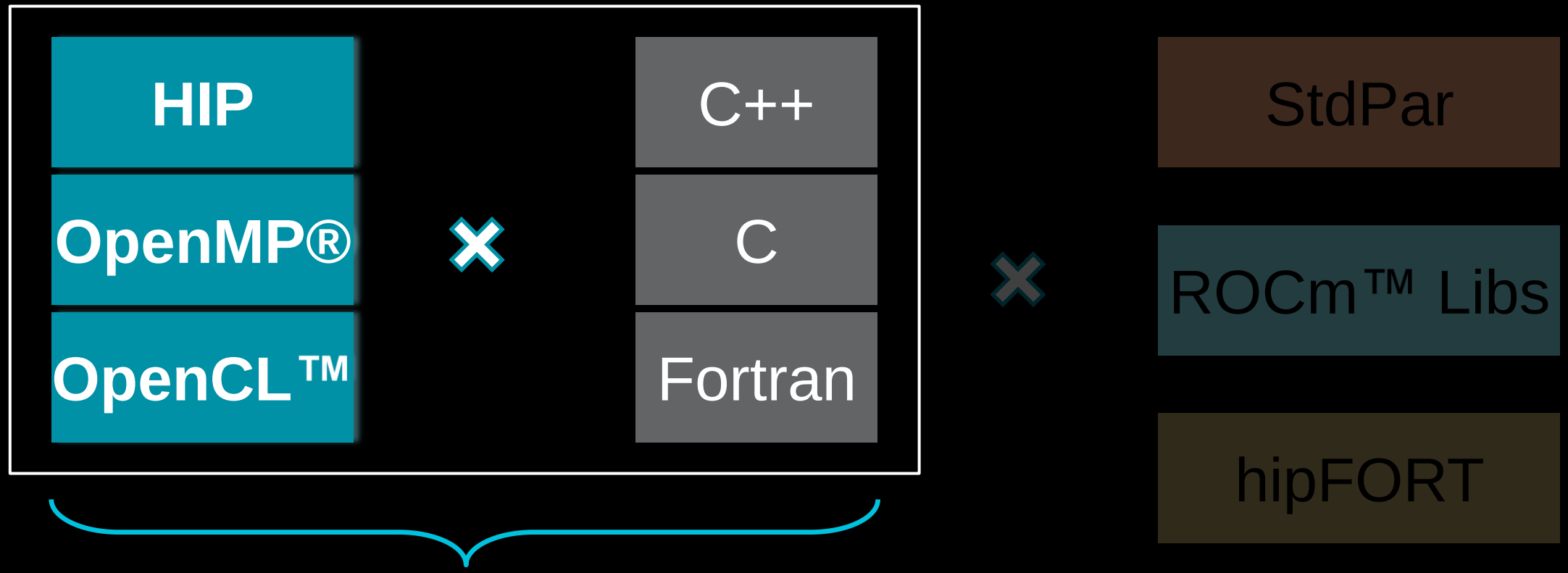
ROCm™ Programming Models



ROCm™ Programming Models



ROCm™ Programming Models



Common LLVM Compiler Backend

ROCm™ Programming Models

HIP

OpenMP®

OpenCL™

C++

C

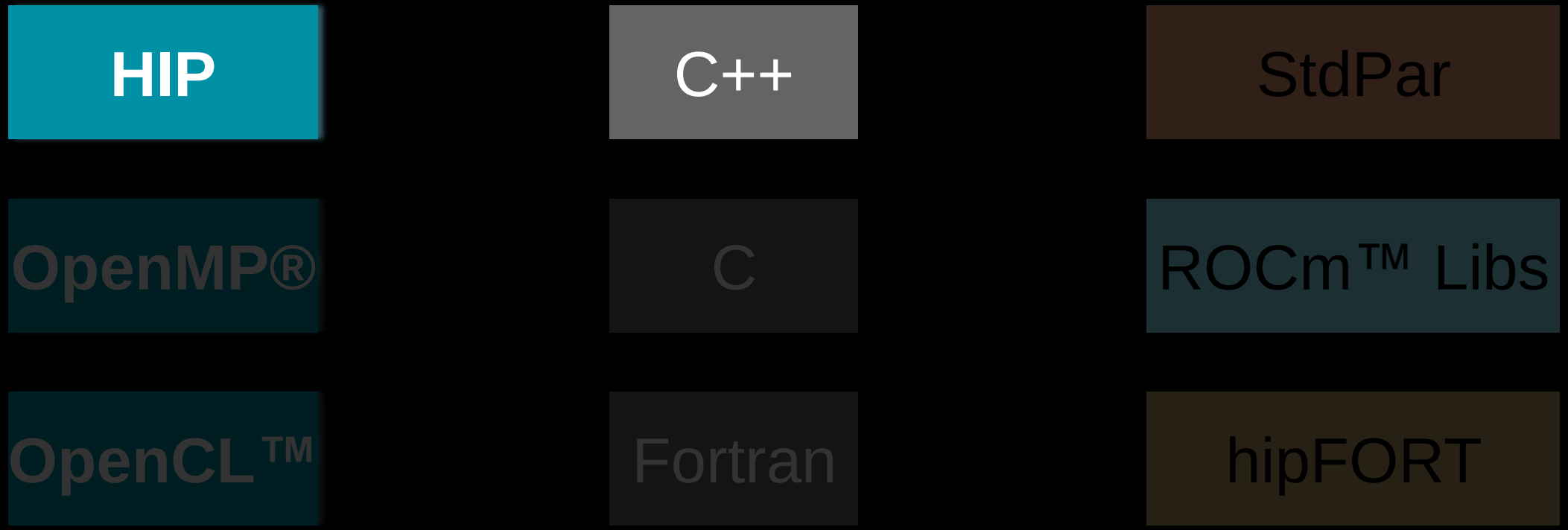
Fortran

StdPar

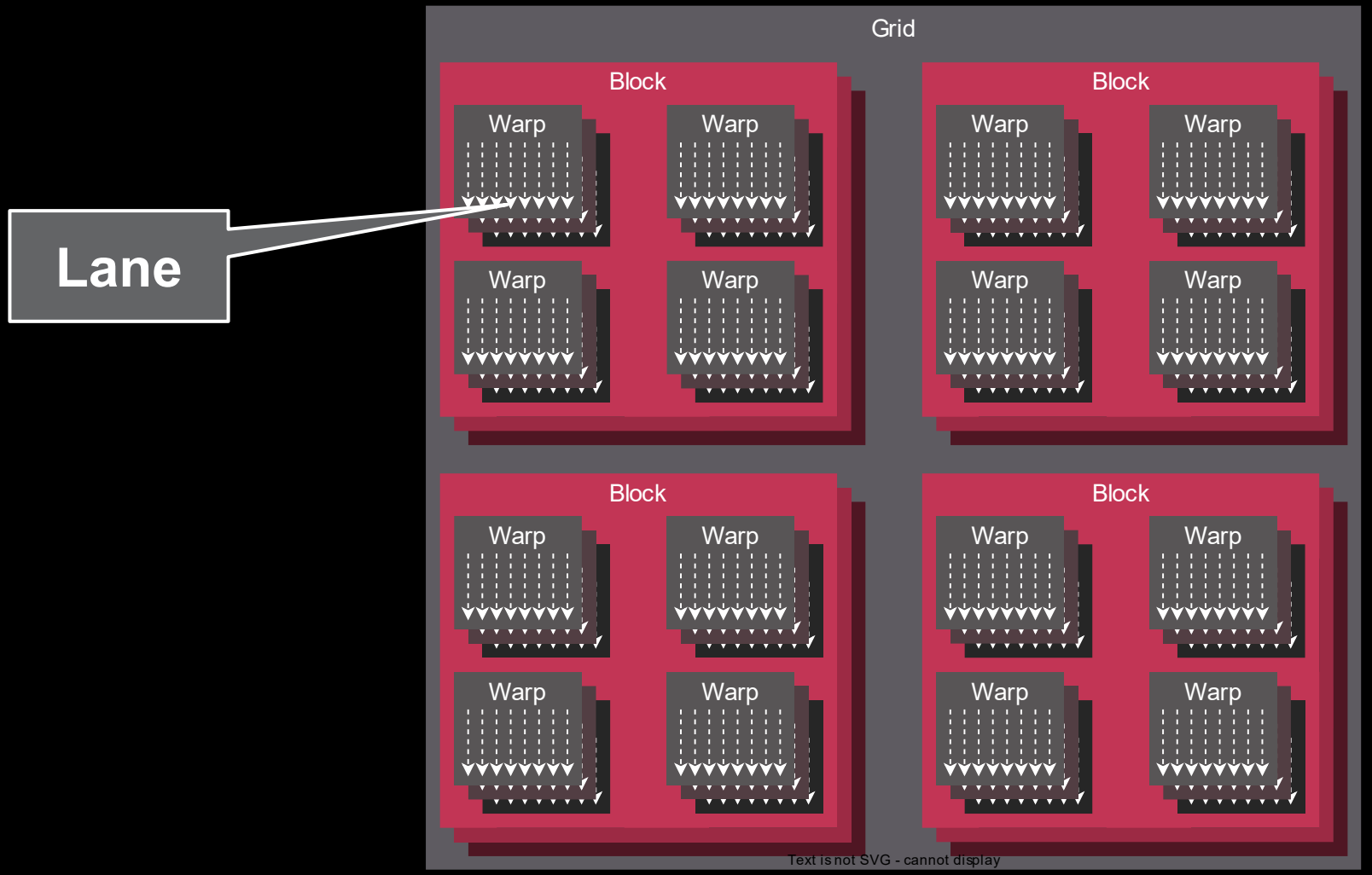
ROCm™ Libs

hipFORT

ROCm™ Programming Models



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];
}
```

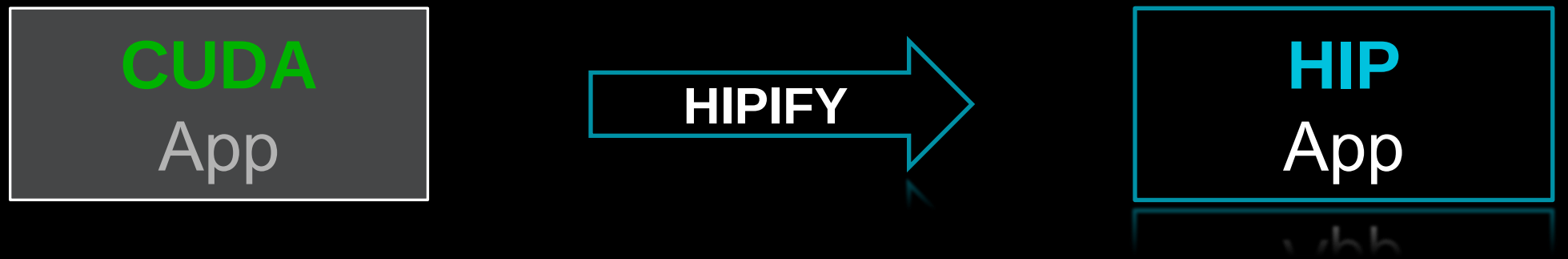
HIP Kernel Example

```
__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

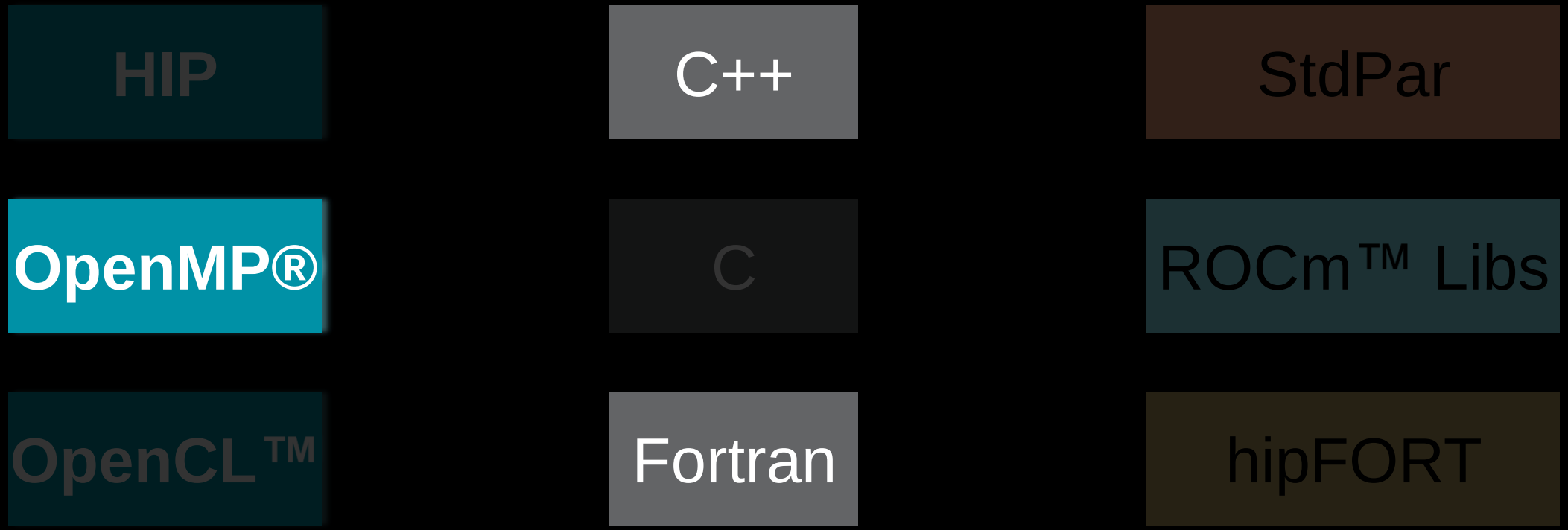
Allows incremental porting

HIP Porting From CUDA

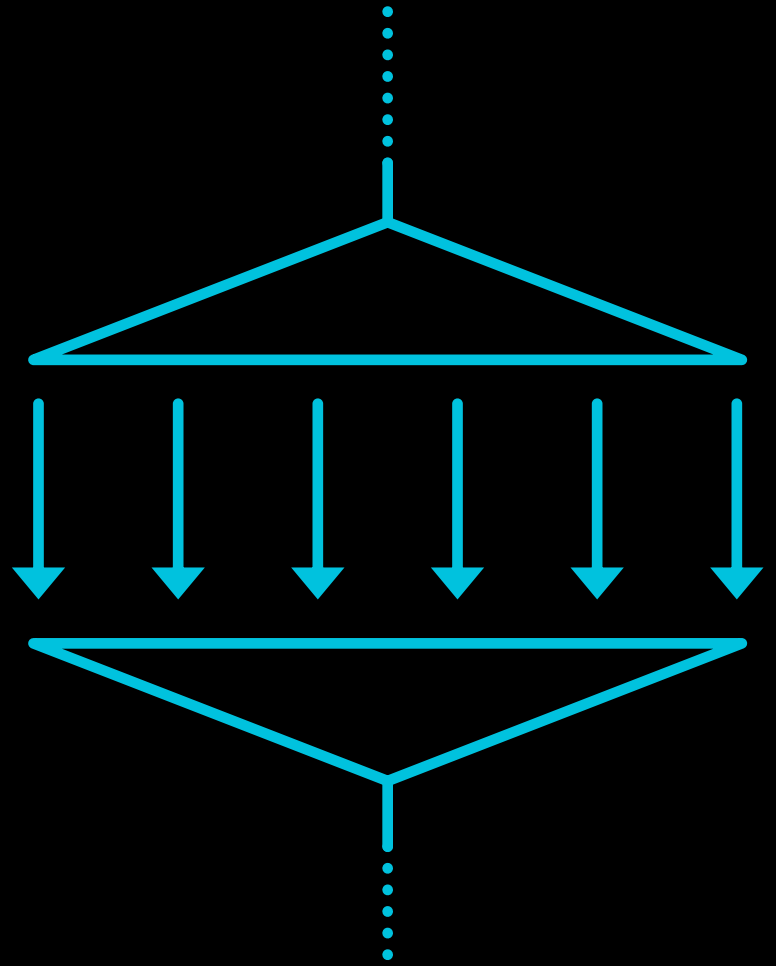
Text-based **HIP** translation

Compiler-based **HIP** translation

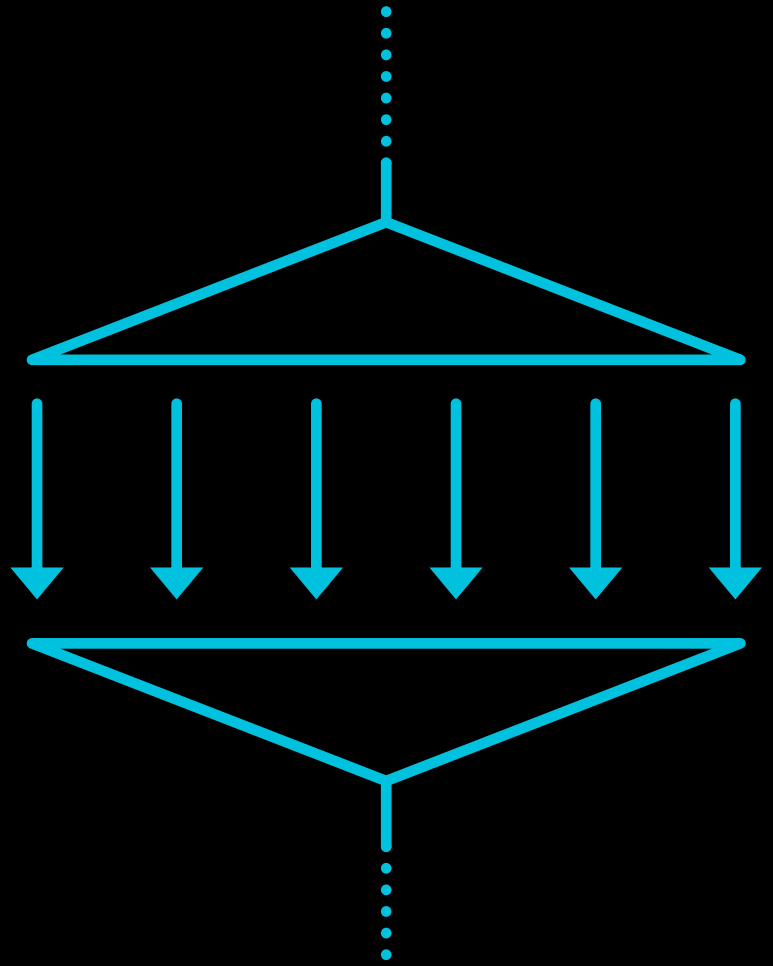
ROCm™ Programming Models



OpenMP® Fundamentals



OpenMP® Fundamentals



Compiler

OpenMP® 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® 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

Printing y and deletion omitted for brevity

OpenMP® and Fortran

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

ROCm™ Programming Models

HIP

C++

StdPar

OpenMP®

C

ROCm™ Libs

OpenCL™

Fortran

hipFORT

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; });

}
```

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

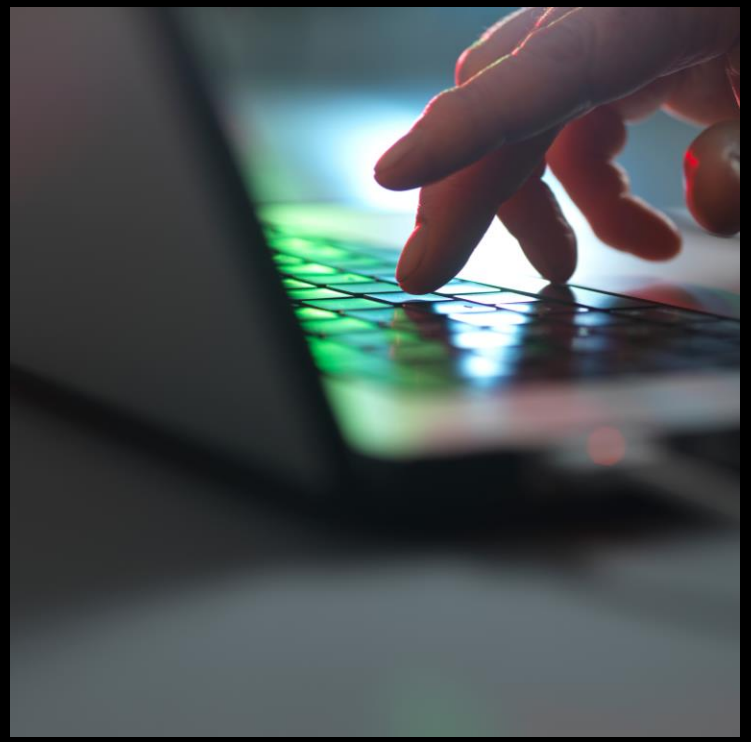
ROCm™ Libs

OpenCL™

Fortran

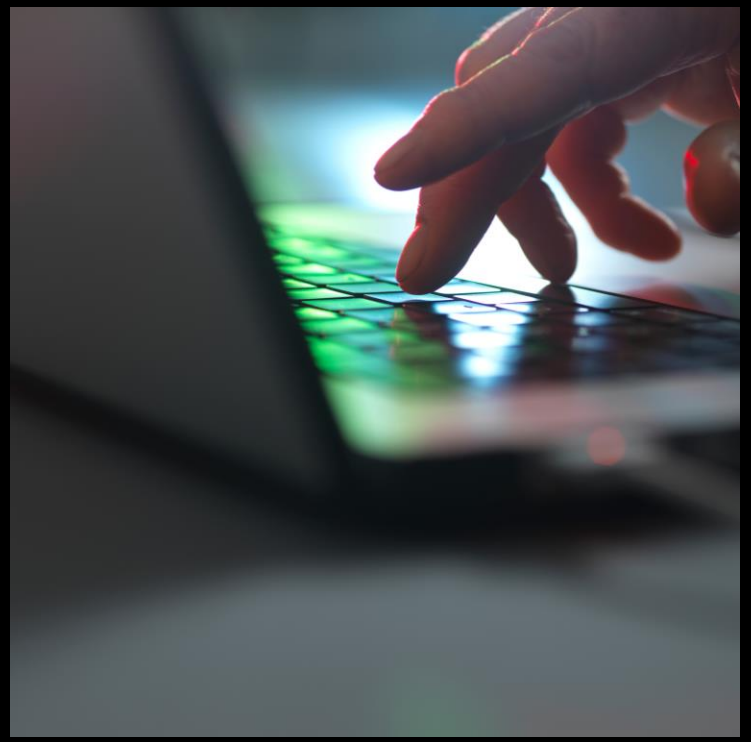
hipFORT

Next Generation Fortran Compiler Journey

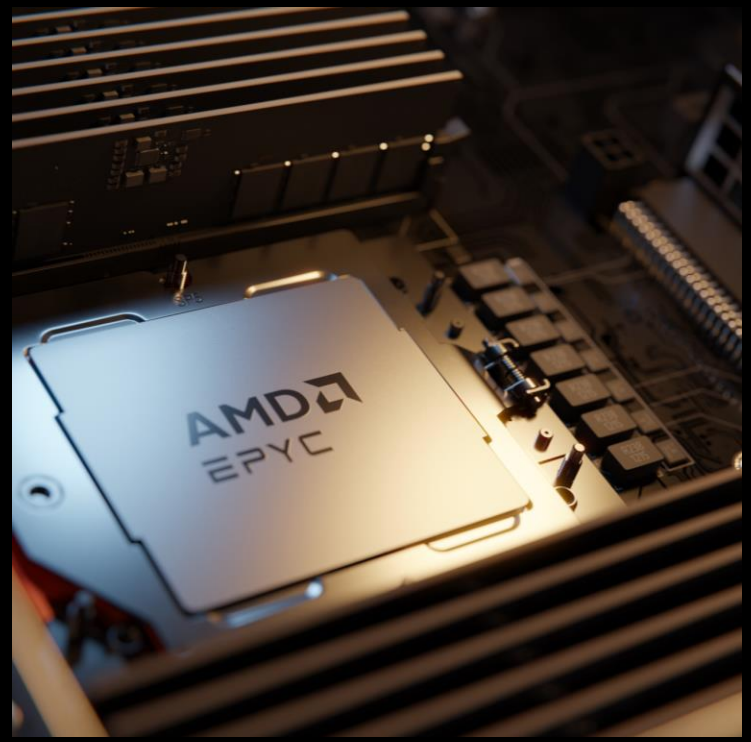


Fortran Language

Next Generation Fortran Compiler Journey

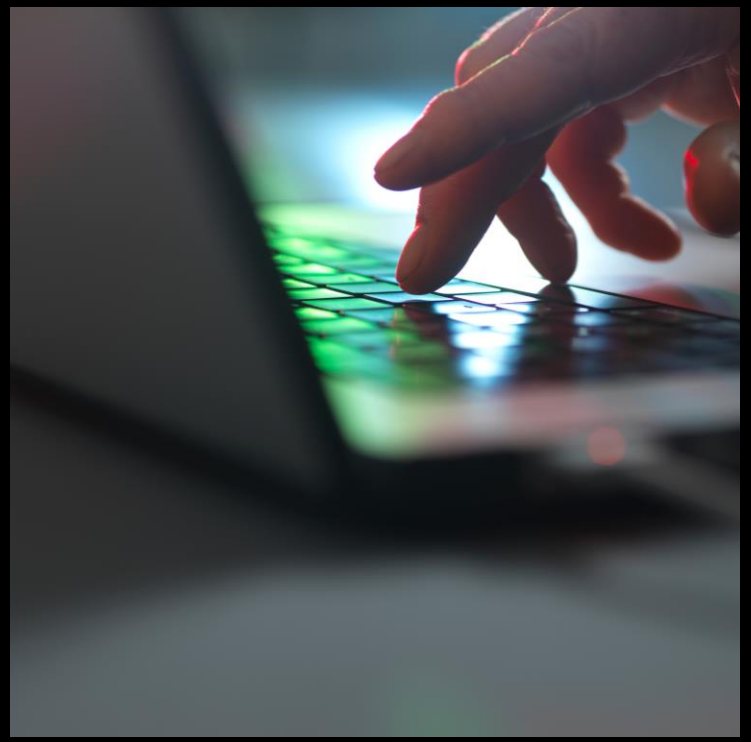


Fortran Language

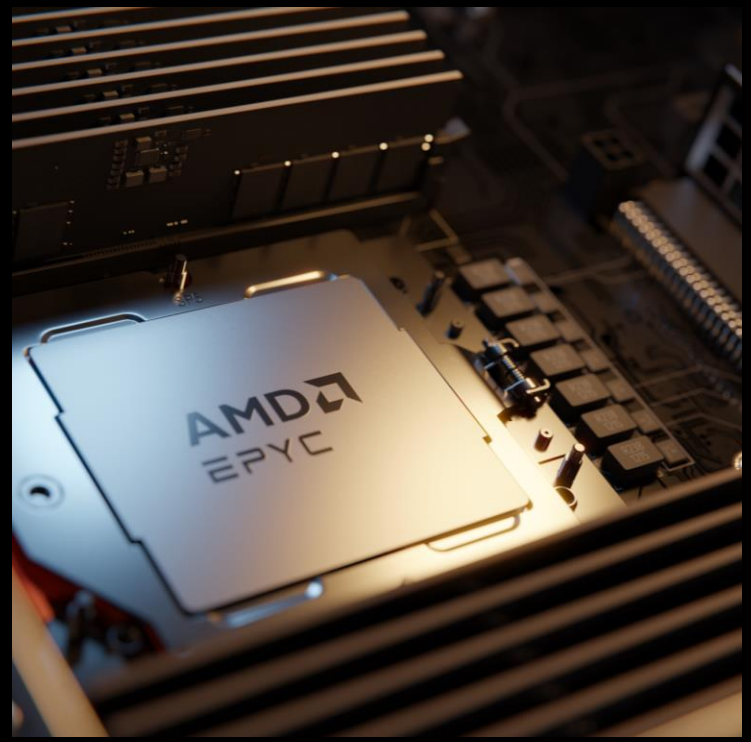


OpenMP® Host Support

Next Generation Fortran Compiler Journey



Fortran Language



OpenMP® Host Support

OpenMP® GPU Support

Fortran Journey Blog Post

Jacobi Solver

```
module jacobi_mod
  use kind_mod, only: RK
  implicit none

  type :: jacobi_t
    real(kind=RK), allocatable :: u(:,:), rhs(:,:), au(:,:), res(:,:)
  end type jacobi_t

  contains
    < Next Slides >

end module jaboci_mod
```

Jacobi Solver

```
subroutine init_jacobi(this, mesh)
  type(jacobi_t), intent(inout) :: this
  type(mesh_t), intent(inout) :: mesh

  allocate( < All components > )
  ! ... Initialization code removed for brevity ...

  !$omp target enter data map(to:this%u,this%rhs,this%au,this%res)

  ! ... some code removed for brevity ...
end subroutine init_jacobi
```

Jacobi Solver

```
subroutine run_jacobi(this, mesh)
  do while (this%iters < max_iters .and. resid > tolerance)

    call laplacian(mesh,this%u,this%au)
    call boundary_conditions(mesh,this%u,this%au)
    call update(mesh,this%rhs,this%au,this%u,this%res)

    resid = norm(mesh,this%res)

    this%iters = this%iters + 1
  end do
end subroutine run_jacobi
```

Jacobi Solver

```
subroutine run_jacobi(this, mesh)
  do while (this%iters < max_iters .and. resid > tolerance)

    call laplacian(mesh,this%u,this%au)
    call boundary_conditions(mesh,this%u,this%au)
    call update(mesh,this%rhs,this%au,this%u,this%res)

    resid = norm(mesh,this%res)

    this%iters = this%iters + 1
  end do
end subroutine run_jacobi
```

Jacobi Solver

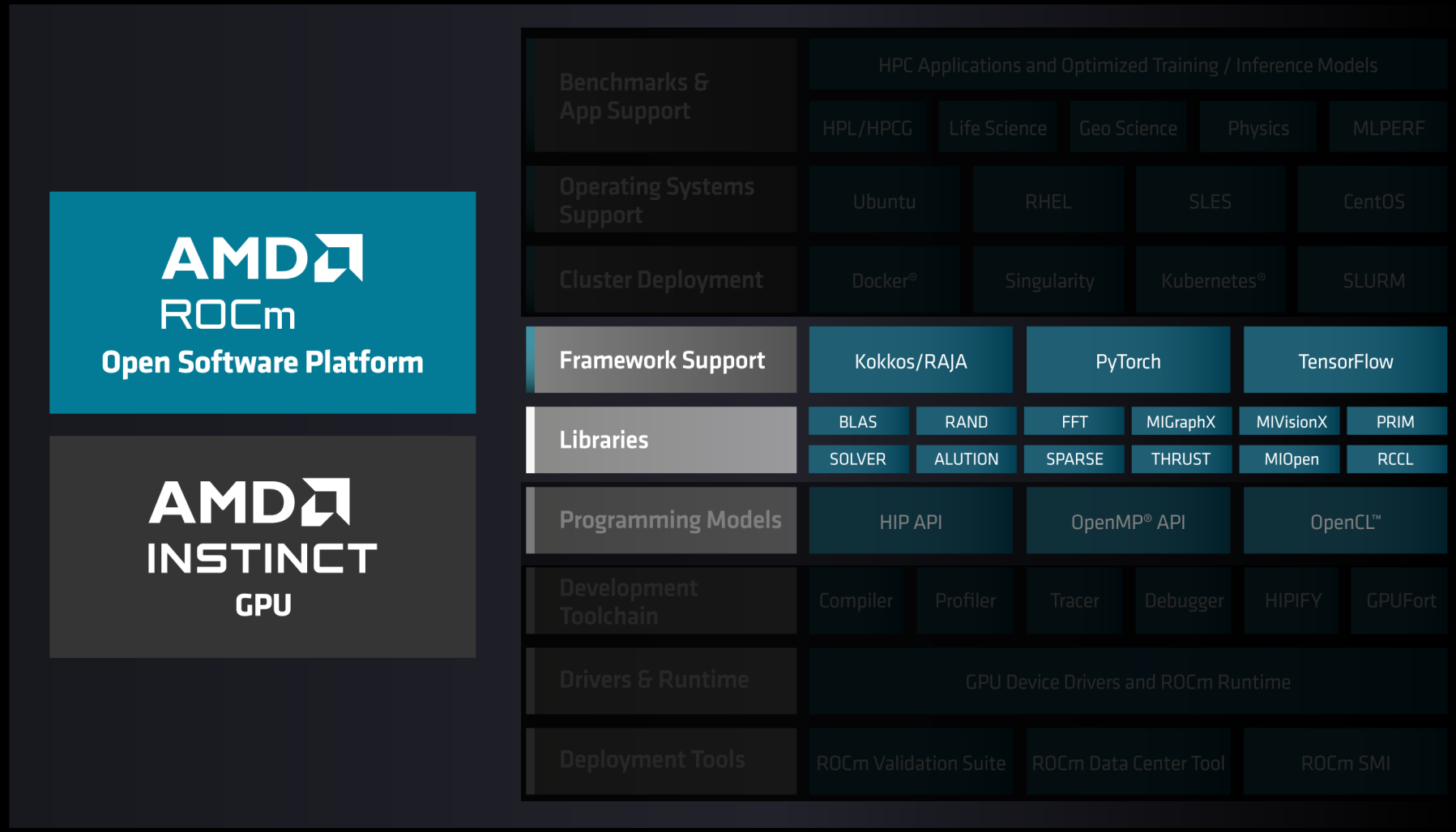
```
subroutine update(mesh,rhs,au,u,res)
  real(kind=RK) :: temp

  !$omp target teams distribute parallel do collapse(2) private(temp)
  do j = 1,mesh%n_y
    do i = 1,mesh%n_x
      temp = rhs(i,j) - au(i,j)
      res(i,j) = temp
      u(i,j) = u(i,j) + temp*factor
    end do
  end do
end subroutine
```

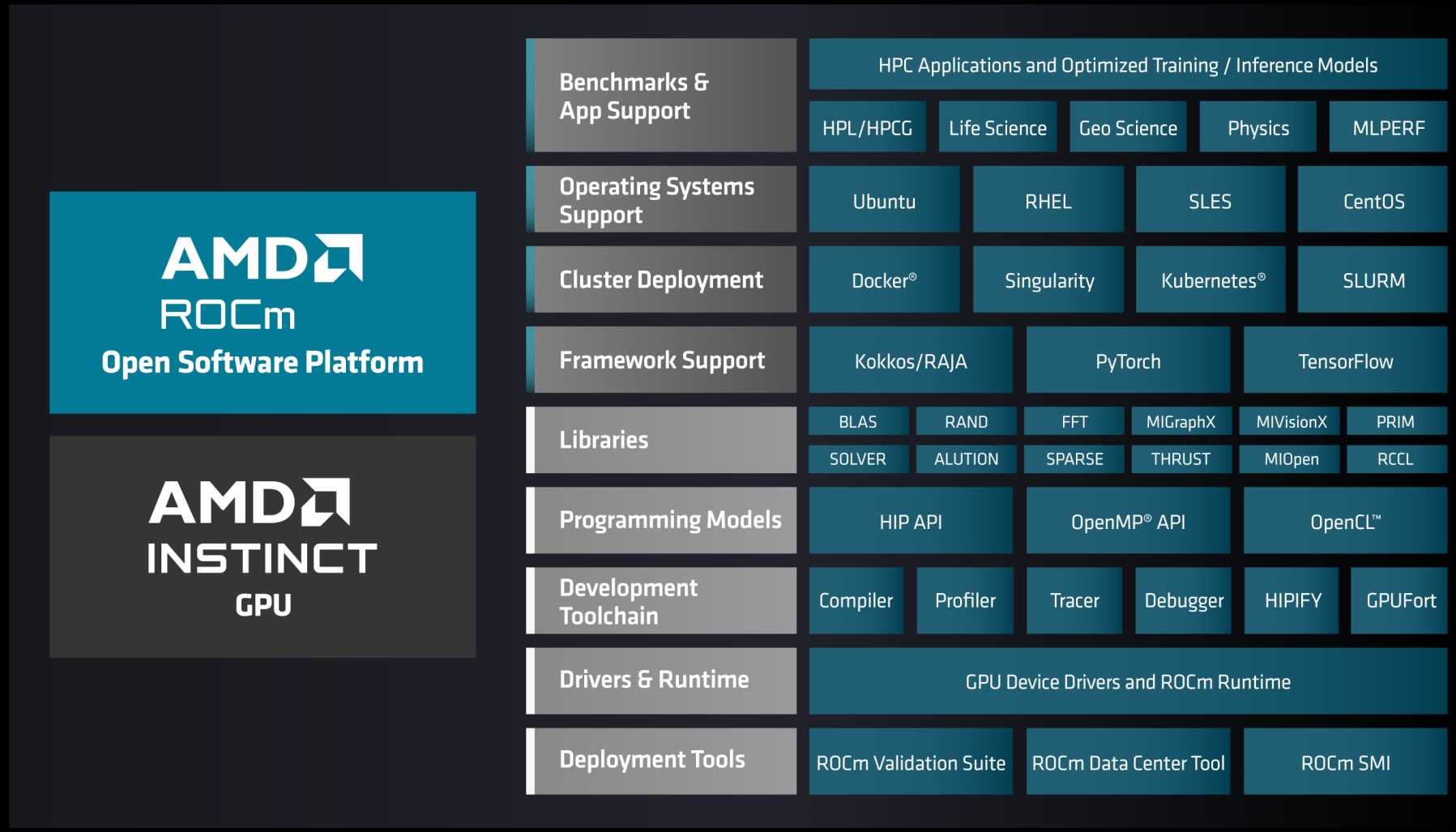
Jacobi Solver

```
function norm(mesh, res) result(norm_val)

  !$omp target teams distribute parallel do collapse(2)
  !$omp& reduction(+:norm_val)
  do j = 1,mesh%n_y
    do i = 1,mesh%n_x
      norm_val = norm_val + res(i,j)**2*dx*dy
    end do
  end do

  norm_val = sqrt(norm_val)/(mesh%n_x*mesh%n_y)
end function norm
```

The ROCm™ Next Generation Fortran Compiler

Latest Preview Available on Infinity Hub



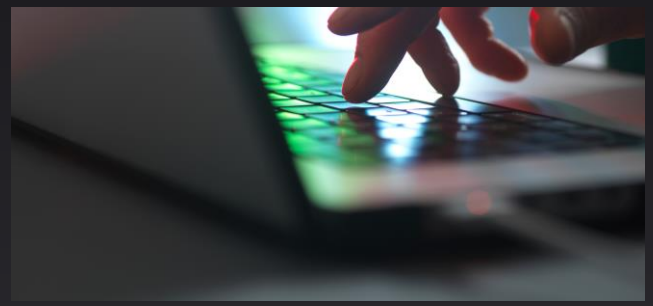


AMD ROCm™

High Performance

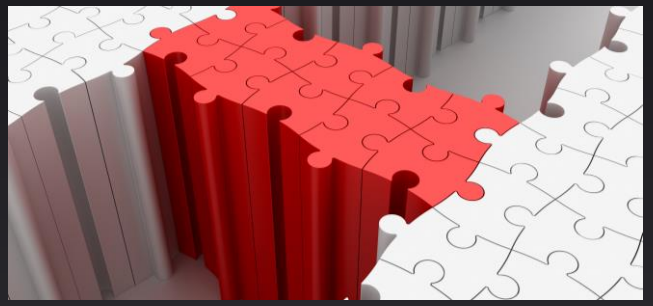
- Powers the top500 list leader
- Solutions for HPC and AI
- Compilers, Libraries, Frameworks

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