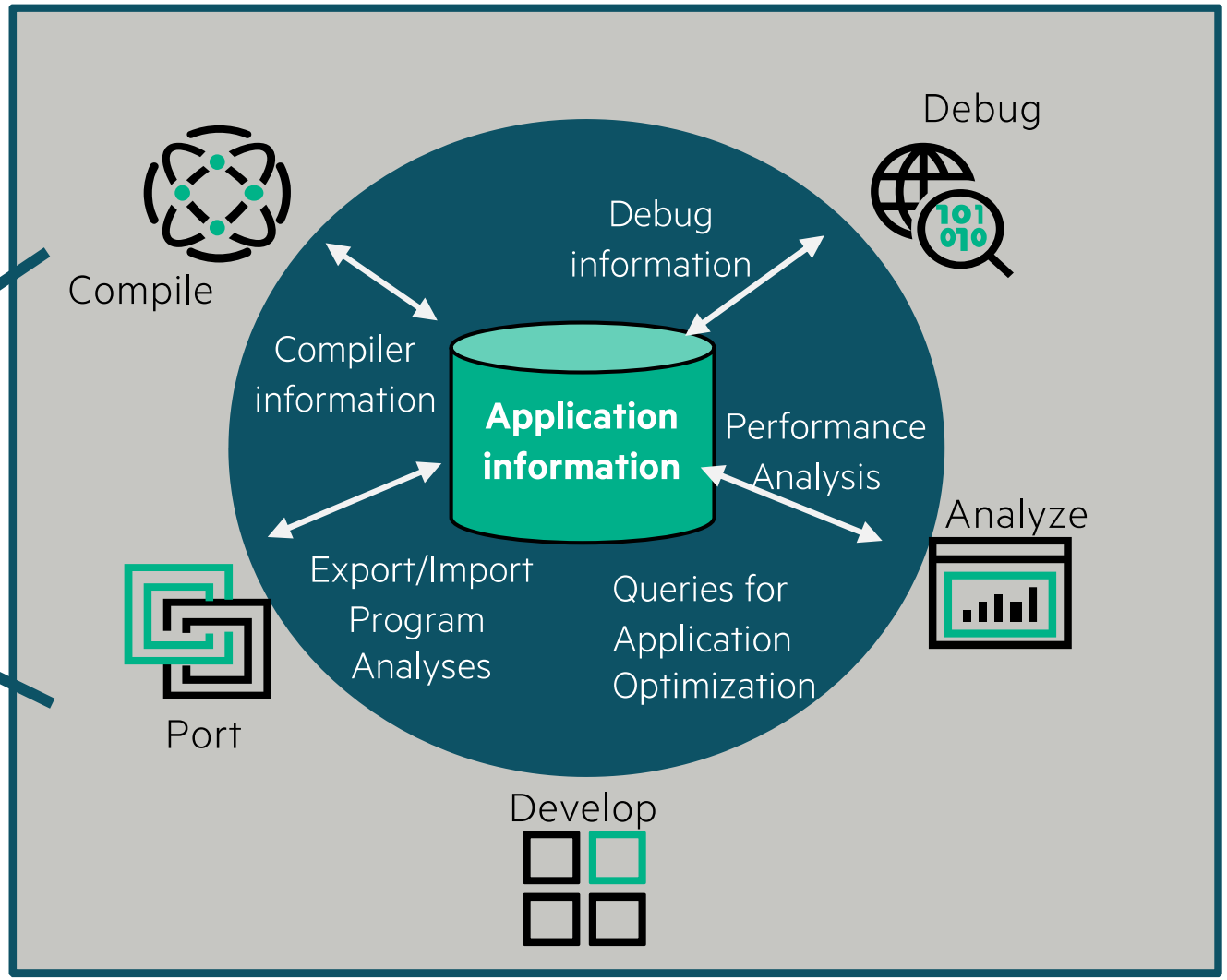
Hewlett Packard
Enterprise

Portability Across GPU Architectures and Languages

Tim Dykes & Harvey Richardson, HPE HPC&AI EMEA Research Lab

PDC Summer School, Aug 21-22, 2024

The Application Porting Life Cycle



The topic of this talk

Approaches to Accelerate Applications

Accelerated Libraries

- The easiest solution, just link the library to your application without in-depth knowledge of GPU programming
- Many libraries are optimized by GPU vendors, eg. algebra libraries

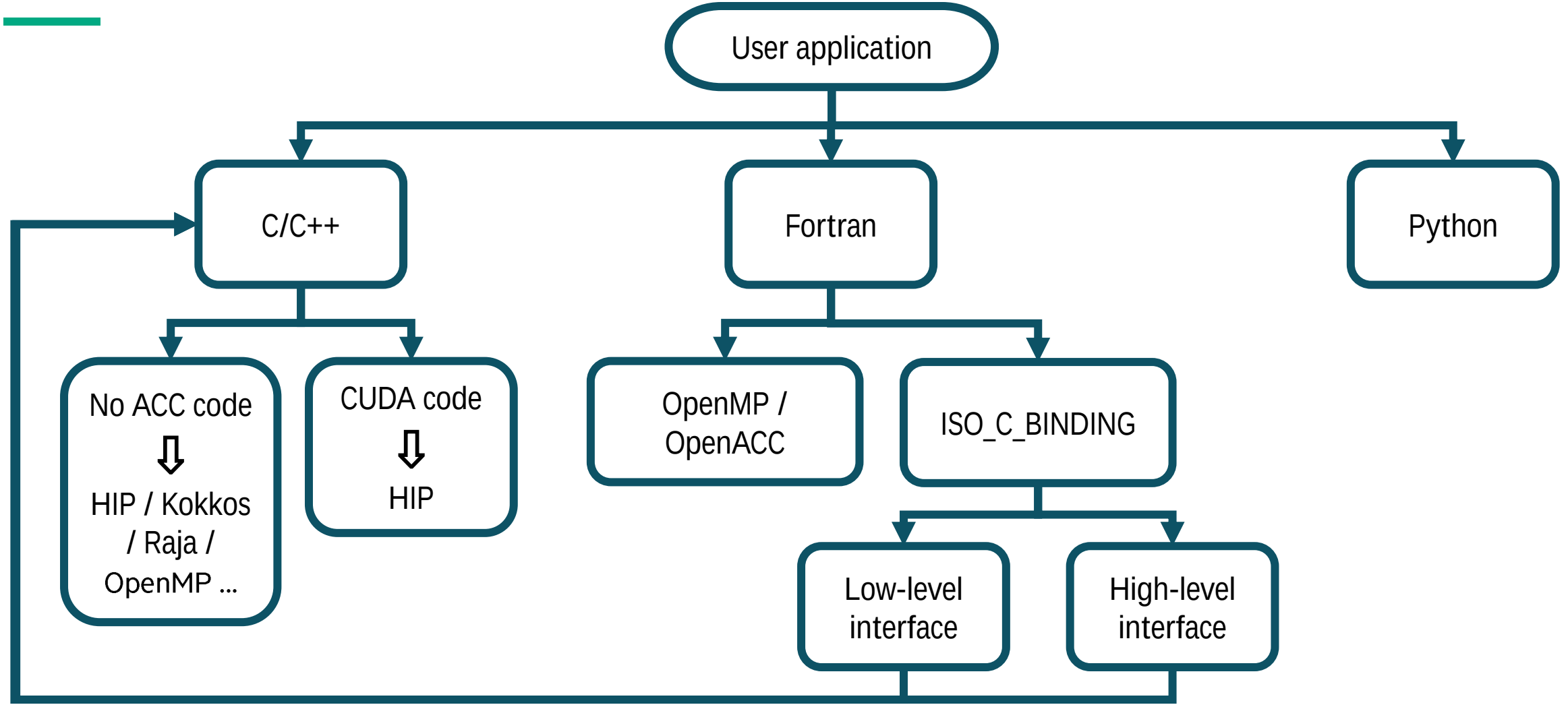
Directive based methods

- Add acceleration to your existing code (C, C++, Fortran)
- Can reach good performance with somehow minimal code changes
- OpenACC, OpenMP

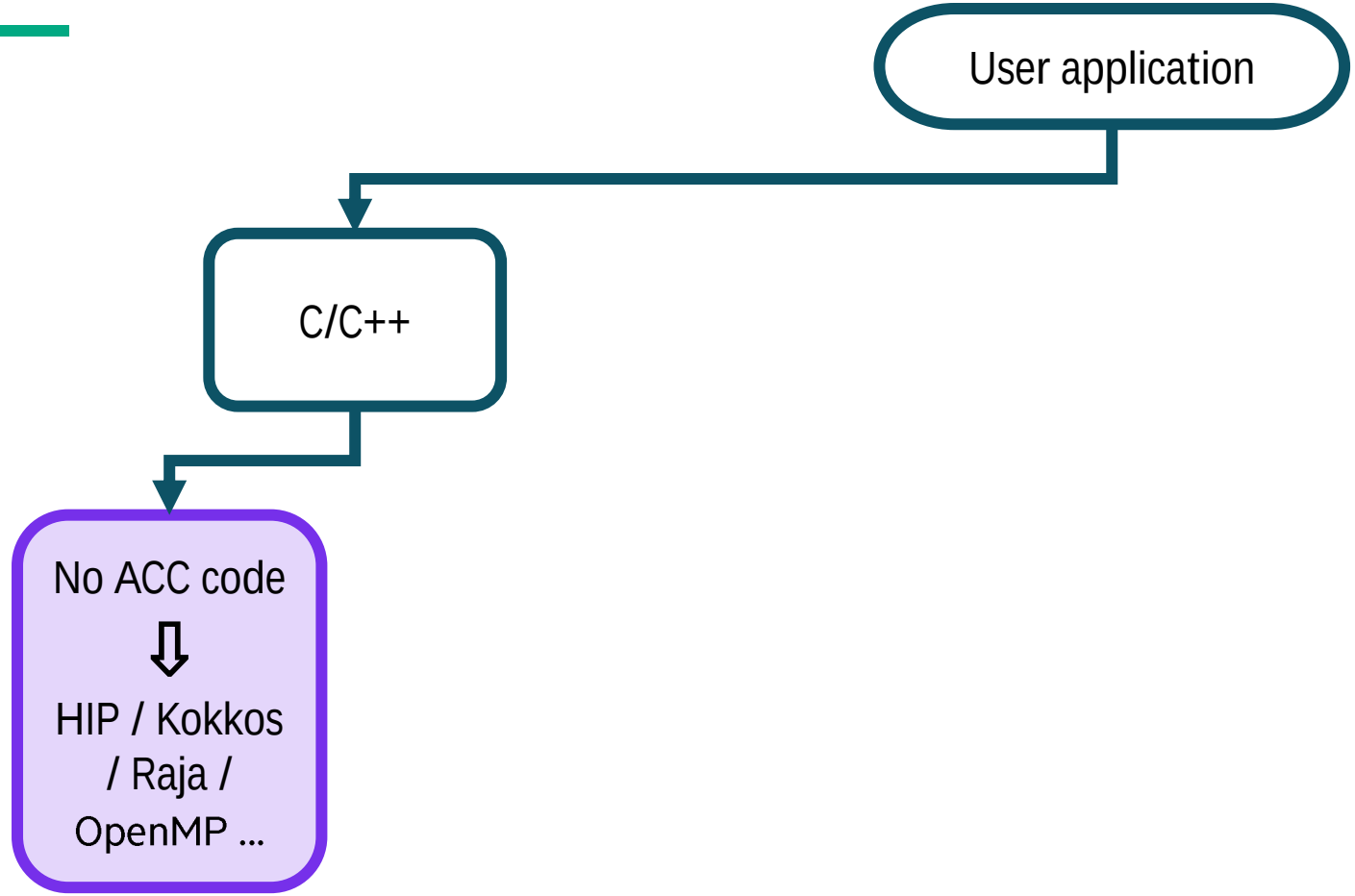
Programming Languages

- Maximum flexibility, require in-depth knowledge of GPU programming and code rewriting (especially for Fortran)
- Kokkos, RAJA, CUDA, HIP, OpenCL, SYCL

Use-cases



Use-cases



C/C++ - No ACC code – HIP programming

- If AMD and NVIDIA GPUs are your only concern, then HIP is a reasonable choice

- Start by profiling the application to identify which parts to offload
  - Remember the discussion from Lecture 01, GPUs want lot of work!
  - We are looking for: high arithmetic intensity, fine grained parallelism, and minimal serial paths

- Start writing HIP code
  - Data movement CPU-GPU
  - Kernels for GPU computation

- Important – check results are correct!

- Repeat the procedure to optimize and offload more
  - Eg. Utilising vendor libraries where possible; overlapped computation / communication; increasing GPU occupancy; and more

  (this procedure is valid for any porting technique)

Portability AMD – NVIDIA

- HIP can run on NVIDIA, with some exceptions:
  - Wavefront size is 64 for AMD, 32 for NVIDIA
  - Dynamic parallelism not supported on AMD
- Because HIP is (almost) 1:1 to CUDA, you can use macros to support both in the same source baseline
  - Used in the DBCSR library, https://github.com/cp2k/dbcsr, `src/acc` directory
  - E.g. (source in example codes: hipify/hip_cuda.cpp)

```
ACC_API_CALL(SetDevice, (device_id)); // Only specify function “root” name
```

where

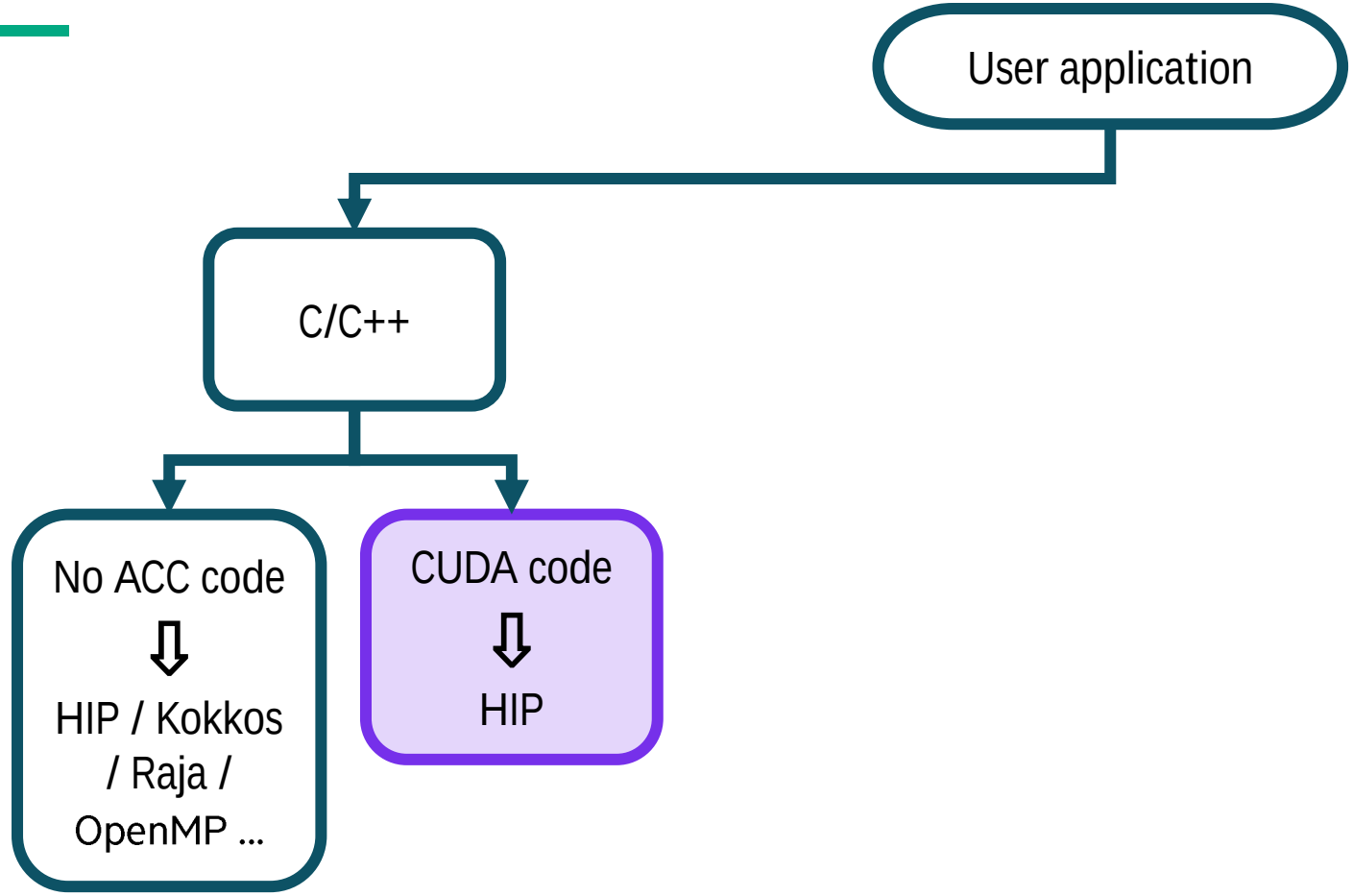
```
#if defined(__CUDA) // compile time flag, i.e. -D__CUDA
#define ACC(x) cuda##x
#elif defined(__HIP) // // compile time flag, i.e. -D__HIP
#define ACC(x) hip##x
#endif
#define ACC_API_CALL(func, args)                                                    \
  {                                                                                 \
    ACC(Error_t) result = ACC(func) args;                                           \
    if (result != ACC(Success)) {                                                   \
      printf("\nACC API error %s (%s::%d)\n", ACC(GetErrorName)(result), __FILE__, __LINE__);  \
      exit(1);                                                                      \
    }                                                                               \
  }
```

C/C++ - No ACC code – Portable approaches

- Kokkos, Raja, Alpaka
  - High-level C++ libraries to provide performance portability across accelerators
  - Built on top of specific hardware backends (including CPU), .e.g CUDA, HIP, HPX, OpenMP, C++ threads
  - Well-supported
- SYCL
  - Standard developed by Khronos Group (https://www.khronos.org/sycl/)
  - Requires hipSYCL to compile for the AMD GPU backend
    - Eg. https://lumi-supercomputer.github.io/LUMI-EasyBuild-docs/h/hipSYCL/
- Directive based approach with OpenMP offload
- Offload C++ Standard Parallelism (stdpar)
  - Evolving support as part of the C++ standard
  - https://github.com/ROCm/roc-stdpar
  - https://rocm.blogs.amd.com/software-tools-optimization/hipstdpar/README.html
  - https://arxiv.org/pdf/2401.02680.pdf
- OpenCL (not very popular nowadays)

Interoperability of the techniques is possible, but requires special care

Use-cases



C/C++ - CUDA code

- ROCm provides a tool to “hipify” CUDA code
  - hipify-clang
    - Compiler (clang) based translator
    - Handles very complex constructs
    - Prints an error if not able to translate
    - Supports clang options
    - Requires CUDA
  - hipify-perl
    - Perl script
    - Relies on regular expressions
    - May struggle with complex constructs
    - Does not require CUDA

➢https://github.com/ROCm-Developer-Tools/HIPIFY
➢https://rocmdocs.amd.com/projects/HIPIFY/en/latest/

Hipify-perl example (1)

- `hipify-perl –examin <file>.cu`
  - For initial assessment
  - No replacements done
  - Prints basic statistics and the number of replacements

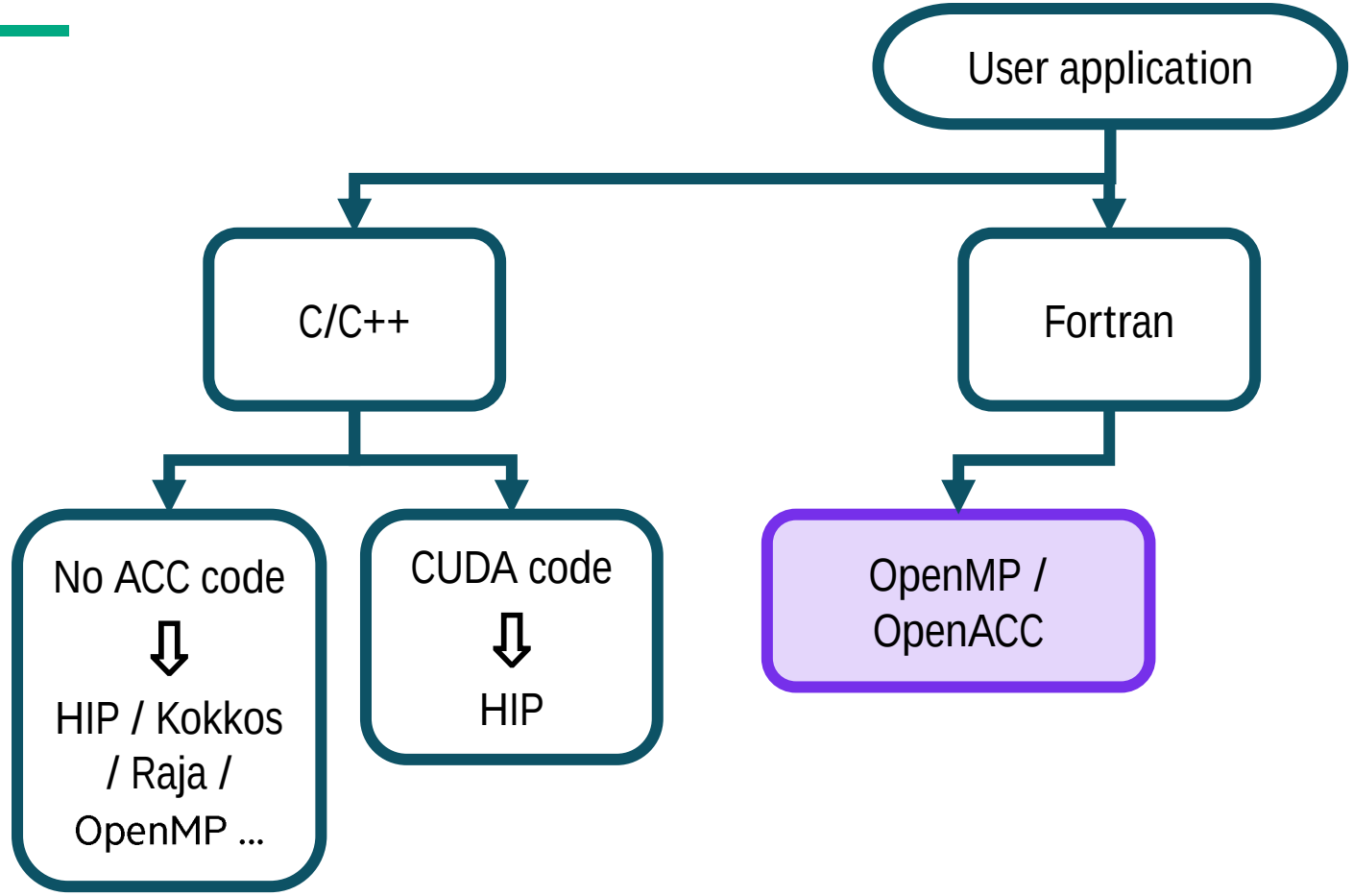
```
> hipify-perl -examin example_slide.cu

[HIPIFY] info: file 'example_slide.cu' statisitics:
  CONVERTED refs count: 18
  TOTAL lines of code: 56
  WARNINGS: 0
[HIPIFY] info: CONVERTED refs by names:
  cuda.h => hip/hip_runtime.h: 1
  cudaError_t => hipError_t: 1
  cudaFree => hipFree: 3
  cudaGetErrorName => hipGetErrorName: 1
  cudaMalloc => hipMalloc: 3
  cudaMemcpy => hipMemcpy: 3
  cudaMemcpyDeviceToHost => hipMemcpyDeviceToHost: 1
  cudaMemcpyHostToDevice => hipMemcpyHostToDevice: 2
  cudaSuccess => hipSuccess: 1
```

Hipify-perl example (2)

- `hipify-perl <file>.cu`
  - Translating a file to standard output

- Other options
  - `-inplace` Backup the input file in .prehip file, modify the input file inplace
  - Recursively do folders

Use-cases



Fortran – OpenMP / OpenACC

- Incremental parallelism, minimal changes in the code (in principle)

	CCE Compiler	AMD Compiler	GNU Compiler (v13+)	NVIDIA Compiler
OpenACC	X		X	X
OpenMP	X		X	X

  - AMD compiler only supports C/C++ OpenMP offload
  - Offload GNU compiler and NVIDIA compiler not available on LUMI
  - GCC reference at https://gcc.gnu.org/wiki/Offloading
  - Other info at https://www.lumi-supercomputer.eu/offloading-code-with-compiler-directives/

- OpenACC well established in some communities (e.g. Climate)
- OpenMP is reasonable for new Fortran-based porting projects
  - Tool to migrate OpenACC to OpenMP: https://github.com/intel/intel-application-migration-tool-for-openacc-to-openmp

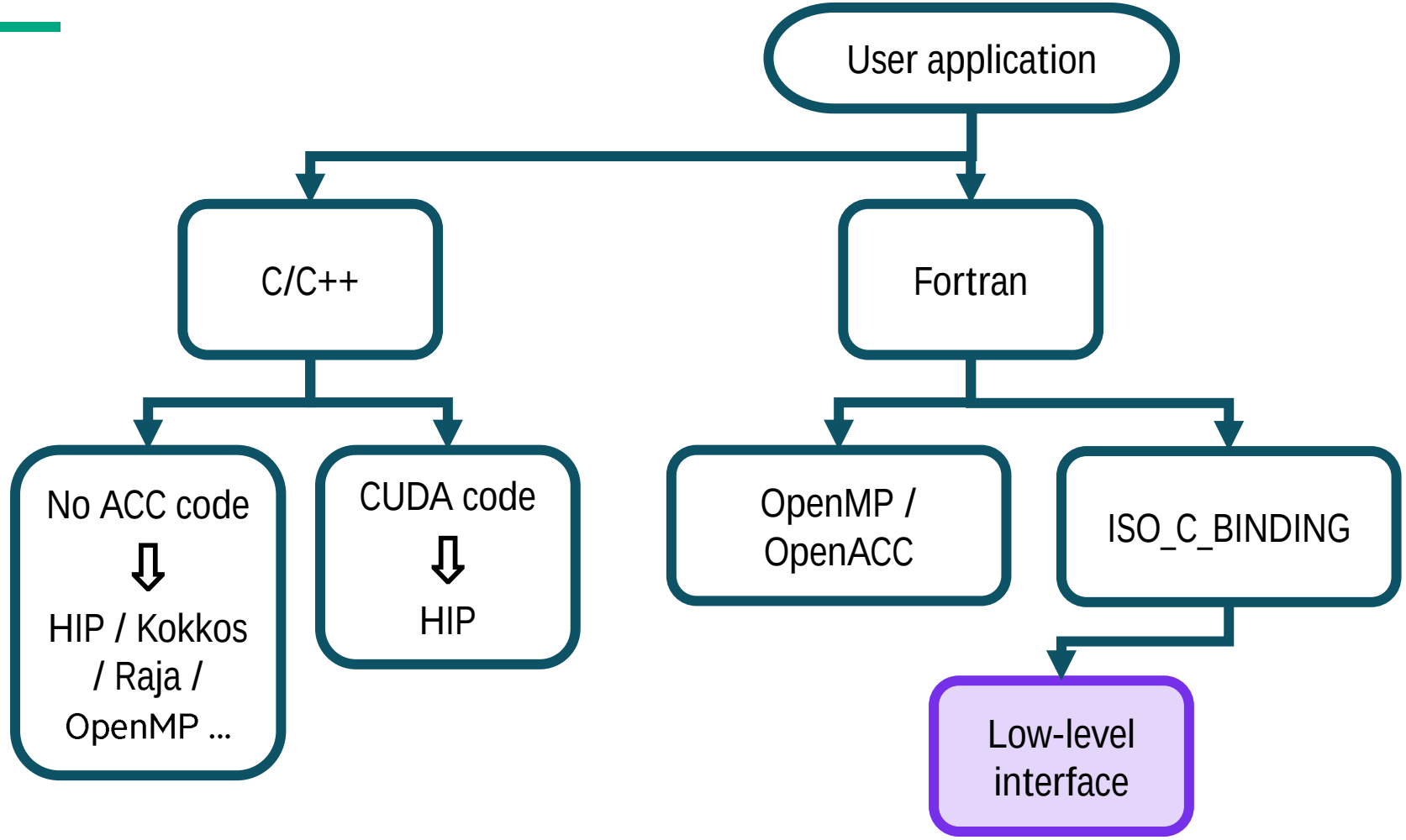
Fortran – OpenMP / OpenACC – Performance

- You should not expect any difference in performance
  - Can use both in the same code baseline for comparison (and debugging) purpose
  - Device-side directives can be selected at compile time via macro
  - E.g. Yambo (https://github.com/yambo-code/yambo)

```
!DEV_OMP target enter data map(to: B, C) map(alloc: A)
!DEV_ACC enter data copyin(B, C) create(A)
!DEV_OMP target teams distribute parallel do simd
!DEV_ACC parallel loop
  do i = 1, n
    A(i) = B(i) + scalar * C(i)
  end do
!DEV_OMP end target teams distribute parallel do simd
!DEV_ACC end parallel loop
!DEV_OMP target exit data map(from: A)
!DEV_ACC exit data copyout(A)
```

```
#if defined _OPENACC
#  define DEV_ACC $acc
#else
#  define DEV_ACC !!!!
#endif

#if defined _OPENMP
#  define DEV_OMP $omp
#else
#  define DEV_OMP !!!!
#endif
```

Use-cases



Fortran – Low-level interface

- Low-level interface approach limits HIP (C++) to relevant functions calls and kernels
  - Fortran code to implement the application logic
- Bind HIP functions and types via the ISO_C_BINDING module
  - Approach used by NEKO code, https://github.com/ExtremeFLOW/neko, file `src/device/hip_intf.F90`
- Example of interface

```
interface
   integer (c_int) function hipMalloc(ptr_d, s) &
        bind(c, name='hipMalloc')
     use, intrinsic :: iso_c_binding
     implicit none
     type(c_ptr) :: ptr_d
     integer(c_size_t), value :: s
   end function hipMalloc
end interface
```

- Kernels declared in C++
  - Fortran interfaces via ISO_C_BINDING module
- No equivalent of CUDA Fortran, which is not standard at all!

Fortran – Hipfort (1)

- ROCm provides `hipfort`
  - Interfaces to main HIP and ROCm libraries
  - F2003 and F2008 interfaces
- Available under `${ROCM_PATH}/hipfort`
- Example adapted from `test/f2003/vecadd`

```
 1#include <hip/hip_runtime.h>
 2
 3__global__ void vector_add(double *out, double const *a,
 4                           double const *b, int N)
 5{
 6  size_t index = blockIdx.x * blockDim.x + threadIdx.x;
 7
 8  if (index<N)
 9    out[index] = a[index] + b[index];
10}
11
12
13extern "C"
14{
15  void launch(double **dout, double **da, double **db, int N)
16  {
17    int num_threads = 256;
18    int num_blocks = (N+num_threads-1)/num_threads;
19    vector_add<<<num_blocks, num_threads>>>(*dout, *da, *db, N);
20  }
21}
```

```
 1 use iso_c_binding
 2 use hipfort
 3 use hipfort_check
 4
 5 implicit none
 6
 7 interface
 8    subroutine launch(out,a,b,N) bind(c)
 9      use iso_c_binding
10      implicit none
11      type(c_ptr) :: a, b, out
12      integer, value :: N
13    end subroutine
14 end interface
15
16 type(c_ptr) :: da = c_null_ptr
17 type(c_ptr) :: db = c_null_ptr
18 type(c_ptr) :: dout = c_null_ptr
19
20 integer, parameter :: N = 1000000
21 integer, parameter :: bytes_per_element = 8 !double precision
22 integer(c_size_t), parameter :: Nbytes = N*bytes_per_element
23
24 ! Plain real should be equivalent to float
25 double precision,allocatable,target,dimension(:) :: a, b, out
26
27 integer :: i
28
29 ! Allocate host memory
30 allocate(a(N)) ; allocate(b(N)) ; allocate(out(N))
31
32 ! Initialize host arrays
33 a(:) = 1.0 ; b(:) = 2.0
34
35 ! Allocate array space on the device
36 call hipCheck(hipMalloc(da,Nbytes)) ; call hipCheck(hipMalloc(db,Nbytes)) ; call hipCheck(hipMalloc(dout,Nbytes))
37
38 ! Transfer data from host to device memory
39 call hipCheck(hipMemcpy(da, c_loc(a(1)), Nbytes, hipMemcpyHostToDevice))
40 call hipCheck(hipMemcpy(db, c_loc(b(1)), Nbytes, hipMemcpyHostToDevice))
41
42 call launch(dout,da,db,N)
43
44 ! Transfer data back to host memory
45 call hipCheck(hipMemcpy(c_loc(out(1)), dout, Nbytes, hipMemcpyDeviceToHost))
46
47 call hipCheck(hipFree(da)) ; call hipCheck(hipFree(db)) ; call hipCheck(hipFree(dout))
48
49 ! Deallocate host memory
50 deallocate(a) ; deallocate(b) ; deallocate(out)
51
52end program vecadd
```

Fortran – Hipfort (2)

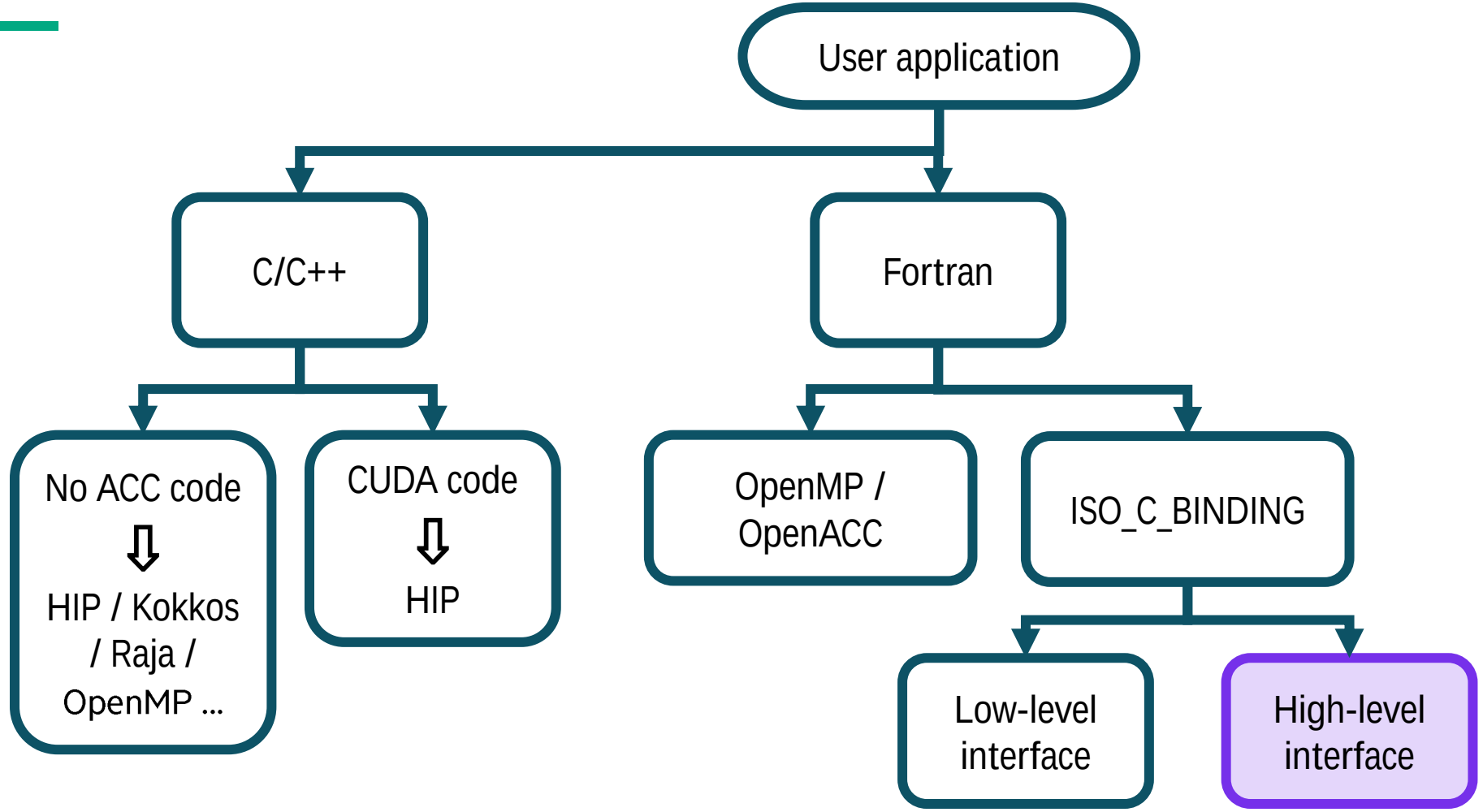
- Compile and link with `PrgEnv-amd`, adding `hipfort` includes and libraries, e.g.

```
> CC -xhip -c vecadd_kernel.cpp
> ftn -I${ROCM_PATH}/hipfort/include/hipfort/amdgcn vecadd.f03 vecadd_kernel.o
-L${ROCM_PATH}/hipfort/lib -lhipfort-amdgcn -o vecadd.x
```

- Can use PrgEnv-cray but need to recompile hipfort with it to get compatible Fortran modules.

➢ https://github.com/ROCm/hipfort
➢ https://rocm.docs.amd.com/projects/hipfort/en/latest/

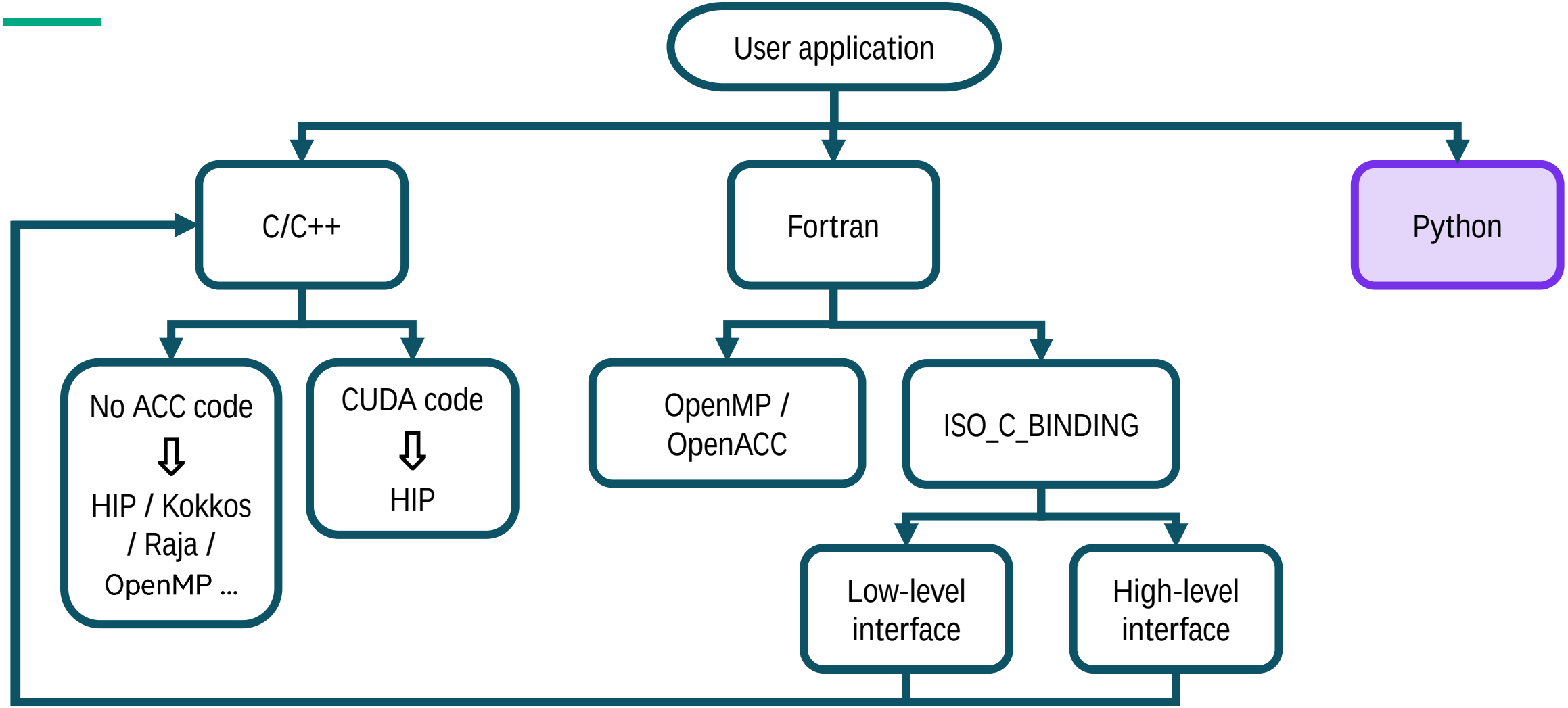
Use-cases



Fortran – High-level interface

- Implement GPU code and the related logic in C++
- Only few high-level interface provided in Fortran for GPU operations
- Good solution for complex existing Fortran codes
  - Separation of concerns between GPU (C/C++) and the existing Fortran code
- Example: DBCSR library (https://github.com/cp2k/dbcsr)

Use-cases



Python

- Low-level Python and Cython Bindings for HIP provided by HIP Python

➢https://github.com/ROCm/hip-python
➢https://rocm.docs.amd.com/projects/hip-python/en/latest/

There are many complex frameworks and libraries that implement GPU support

- CuPy (https://cupy.dev/) provides AMD support for computing with Python
- PyTorch, Tensorflow, etc..

Python Environment

```
$> module avail cray-python
cray-python/3.9.12.1    cray-python/3.9.13.1    cray-python/3.10.10 (D)
```

- Do not use system python installations such as `/usr/bin/python3` (v3.6.15)
- Load a `cray-python` module instead

```
$> pip list --format=freeze
atomicwrites==1.4.0 beniget==0.4.1 cloudpickle==2.2.1 Cython==0.29.36 dask==2022.7.0
exceptiongroup==1.1.2 fsspec==2023.6.0 gast==0.5.4 importlib-metadata==6.8.0
iniconfig==2.0.0 joblib==1.3.2 locket==1.0.0 mpi4py==3.1.4 msgpack==1.0.7 numpy==1.23.5
packaging==23.1 pandas==1.5.3 partd==1.4.0 pip==22.3.1 pluggy==1.2.0 ply==3.11 py==1.11.0
pybind11==2.11.1 pyparsing==3.1.1 pytest==7.4.0 python-dateutil==2.8.2 pythran==0.13.1
pytz==2023.3 PyYAML==6.0.1 SciPy==1.10.0 setuptools==65.5.0 setuptools-scm==7.1.0
six==1.16.0 toml==0.10.2 tomli==2.0.1 toolz==0.12.0 typing_extensions==4.7.1 wcwidth==0.2.6
zipp==3.16.2
```

- Several packages such as numpy and mpi4py are preinstalled.
- First set `PYTHONUSERBASE=/work/<your_path>/.local` to install new packages and then use `pip install --user <package>`. Update the `PYTHONPATH` and `PATH` accordingly if needed.

Launch a Python script

```
$> srun –n 1 python script.py
```

- Use `srun` to launch a python script from within an interactive session or a batch script. (In particular large frameworks like tensorflow or pytorch.)
- Also if the script does not contain apparent multiprocessing/multithreading.
- Importing many module can result in large startup times.

```
$>  srun -n 1  -c 2 --cpu_bind=core python multi.py
Process psutil.Process(pid=67518, name='python', status='running', started='18:52:29') has affinity [0, 128]
Process psutil.Process(pid=67519, name='python', status='running', started='18:52:29') has affinity [0, 128]
```

- Use `psutil.Process().cpu_affinity()` to gather information.

Launch an mpi4py Python script

```
$> srun -n 4 --cpu_bind=map_cpu:1,3,5,7  python hello.py

Process psutil.Process(pid=64648, name='python', status='running',
started='18:35:57') has affinity [1]
Hello world from node nid003404, rank 0 out of 4.

Process psutil.Process(pid=64650, name='python', status='running',
started='18:35:57') has affinity [5]
Hello world from node nid003404, rank 2 out of 4.

…
```

```
import psutil
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
procname = MPI.Get_processor_name()

print(f"Process {psutil.Process()} has affinity \
{psutil.Process().cpu_affinity()}")

print("Hello world from node {}, rank {:d} out of \
{:d}.".format(procname, rank, size))
```

- Launch the mpi4py program with `srun`.
- mpi4py will use cray-mpich which in turn is configured for SLURM.
- If you see "… attempting to use MPI before initialization … " it could be related to mpi4py being built with GCC. Try `LD_PRELOAD=/opt/cray/pe/lib64/libmpi_gnu_91.so` or another GNU cray-mpich version.
- Affinity can be checked or set with `psutil.Process().cpu_affinity()`

GPU-aware MPI

- If the variable `MPICH_GPU_SUPPORT_ENABLED` is set, then MPI assumes that you link with the GTL library to enable the GPU support in MPI
  - Error message

```
MPIDI_CRAY_init: GPU_SUPPORT_ENABLED is requested, but GTL library is not linked
```

- To link the GTL library in python

Before the import of mpi4py

```
from os import environ
if environ.get("MPICH_GPU_SUPPORT_ENABLED", False):
    from ctypes import CDLL, RTLD_GLOBAL
    CDLL(f"{environ.get('CRAY_MPICH_ROOTDIR')}/gtl/lib/libmpi_gtl_hsa.so",
mode=RTLD_GLOBAL)

from mpi4py import MPI
```

Using a virtual environment based on cray-python

- For example, assume we want to take cray-python as a basis and add matplotlib

```
> module load cray-python
> python -m venv --system-site-packages craympl-venv
> source craympl-venv/bin/activate
(craympl-venv) > which pip
/tmp/craympl-venv/bin/pip
(craympl-venv) > pip install matplotlib
Collecting matplotlib
. . .
srun –n 8 python myapp.py
```

- The `--system-site-packages` option gives the venv access to the system site packages directory
- The resulting venv can now use the cray-python modules such as mpi4py

Netcdf with Python

```
$> pip install --user netcdf4
```

- Install the `netcdf4` package with pip.
- Look for instance at min/max/avg of all variables in a netcdf file.

```
$> module load cray-python
$> pip install --user recursive-diff
$> srun ncdiff --recursive --rtol 1e-6 --atol 1e-8
-b lhs rhs
```

- Netcdf files and directories can be compared in python recursively with the `ncdiff` utility.
- Pay attention to automatic installation of required package versions. A new numpy installation is not necessarily linked against cray-libsci

```
from netCDF4 import Dataset
import numpy as np
import pandas as pd
import sys

'''
Loops over all variables present in the netcdf file
specified on the command line and
prints min/max/avg for every variable.
'''

filename = sys.argv[1]

tmp = Dataset(filename, "r", format="NETCDF4")

mins = []
maxs = []
avgs = []
for v in tmp.variables.keys():
            tmp2 = np.frombuffer(tmp[v][:])
            mins.append(tmp2.min())
            maxs.append(tmp2.max())
            avgs.append(tmp2.sum()/tmp2.size)

nc_df = pd.DataFrame(data={'min':mins, 'max':maxs,
'avg':avgs}, index=tmp.variables.keys())
print(nc_df)
```

Perftools for Python (experimental support)

```
$> module load perftools-preload
$> module load cray-python
$> srun -n 4 pat_run `which python` hello.py
```

- Load the cray-python module.
- Use `pat_run` and the absolute path to python.

```
$> pat_report -o myrep <exp-dir>
$> pat_report -O ct+src -o myrep.ct <exp-dir>
```

- Generate call tree report in addition to default one.
- Python module must be loaded when `pat_report` is invocated.
- Python methods from the source code are prepended with `python.*`
- Check the `pat_run` man page for more details.
- MPI function trace introduced with CPE >=23.05

```
Table 1:  Calltree View with Callsite Line Numbers

  Samp% |  Samp | Calltree
        |       |  PE=HIDE

 100.0% | 157.0 | Total
|-----------------------------------------------------
|  73.2% | 115.0 | python.<module>:heat-p2p.py:line.134
||----------------------------------------------------
||  46.3% |  72.8 | python.main:heat-p2p.py:line.125
|||---------------------------------------------------
3||  37.9% |  59.5 | python.iterate:heat-p2p.py:line.82
||||--------------------------------------------------
4|||  15.3% |  24.0 | python.evolve:heat-p2p.py:line.51
5|||        |       |  Py_BytesMain:main.c:line.732
4|||  11.3% |  17.8 | python.evolve:heat-p2p.py:line.53
5|||        |       |  Py_BytesMain:main.c:line.732
4|||   7.8% |  12.2 | python.evolve:heat-p2p.py:line.50
5|||        |       |  Py_BytesMain:main.c:line.732
4|||   3.5% |   5.5 | python.evolve:heat-p2p.py:line.55
5|||        |       |  Py_BytesMain:main.c:line.732
||||=================================================
```

Not covered in this session

- Deep dives into mpi4py and numpy.
- Improving communication of frameworks like tensorflow or pytorch.

- Further information:
  ARCHER2 Python guide https://docs.archer2.ac.uk/user-guide/python
  LUMI PyTorch guide https://docs.lumi-supercomputer.eu/software/packages/pytorch/

Hands-on

- Hipify
- HipFort
- Multiple streams