Kokkos in a nutshell

Thomas Padioleau
Mathieu Lobet
Julien Bigot

CExA Team

Journées des PTCs – November, 22 2023

Who are the CExA members?

Research-engineers from all the CEA departments with accumulated experience in :

- Application development
- Application support
- GPU porting
- Library development
- Researchers in computer-science
- Numerical physics

∇ NabLab

a Discrete Domain Computation library

PDI

GYSELA-X

ARCANE FRAMEWORK

Smilei)

Disclaimers

- This presentation is partly based on the introduction slides of the Kokkos tutorials
- Some slides may reflect the speaker’s opinion and not the Kokkos team’s opinion
- Examples assume we are implicitly using the namespace Kokkos meaning that “Kokkos::xxxx” becomes “xxxx” for simplicity

Useful links and Materials

- Kokkos Github – https://github.com/kokkos
- Kokkos tutorials – https://github.com/kokkos/kokkos-tutorials
- Kokkos documentation - https://kokkos.github.io/kokkos-core-wiki/index.html
- Kokkos Slack channel - https://kokkosteam.slack.com
  - Ask your questions to the team
  - We now have a specific general-fr channel for French speakers
- CExA website - https://cexa-project.org/
- CExA GitHub - https://github.com/CExA-project

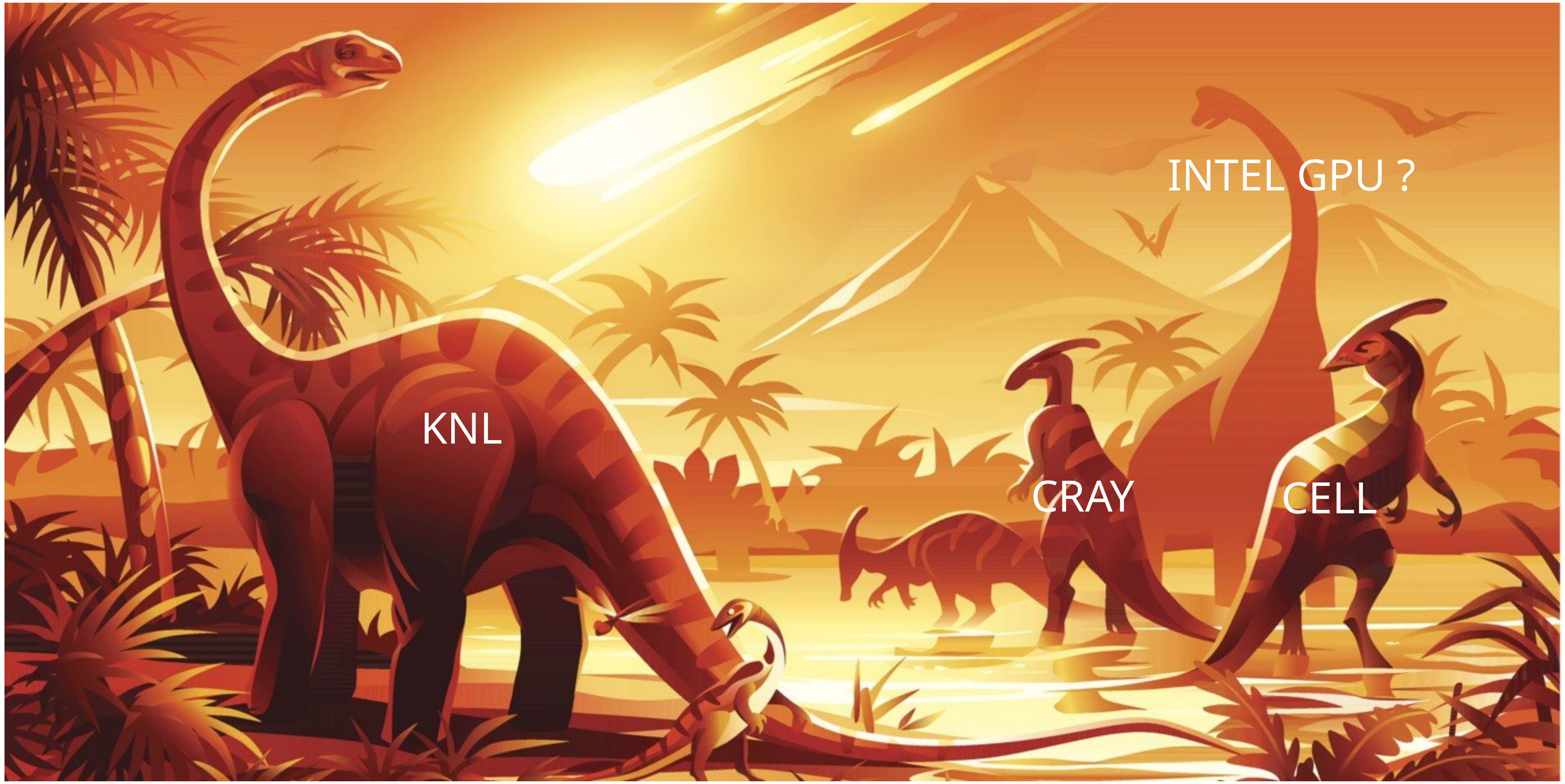
Our presentation into 4 points

1. Context
2. What is Kokkos ?
3. How it works (basically)
4. Real-life example

1. Context

Life in our HPC world

Evolution in HPC : exotic and disruptive hardware are permanently appearing (and disappearing) in super-computers history



A lot of work for HPC developers

New technologies are exciting for researchers but may be stressful for application developers and users :

- May require vendor specific programming models
- May require new programming paradigms
- Algorithms may have to be rewritten

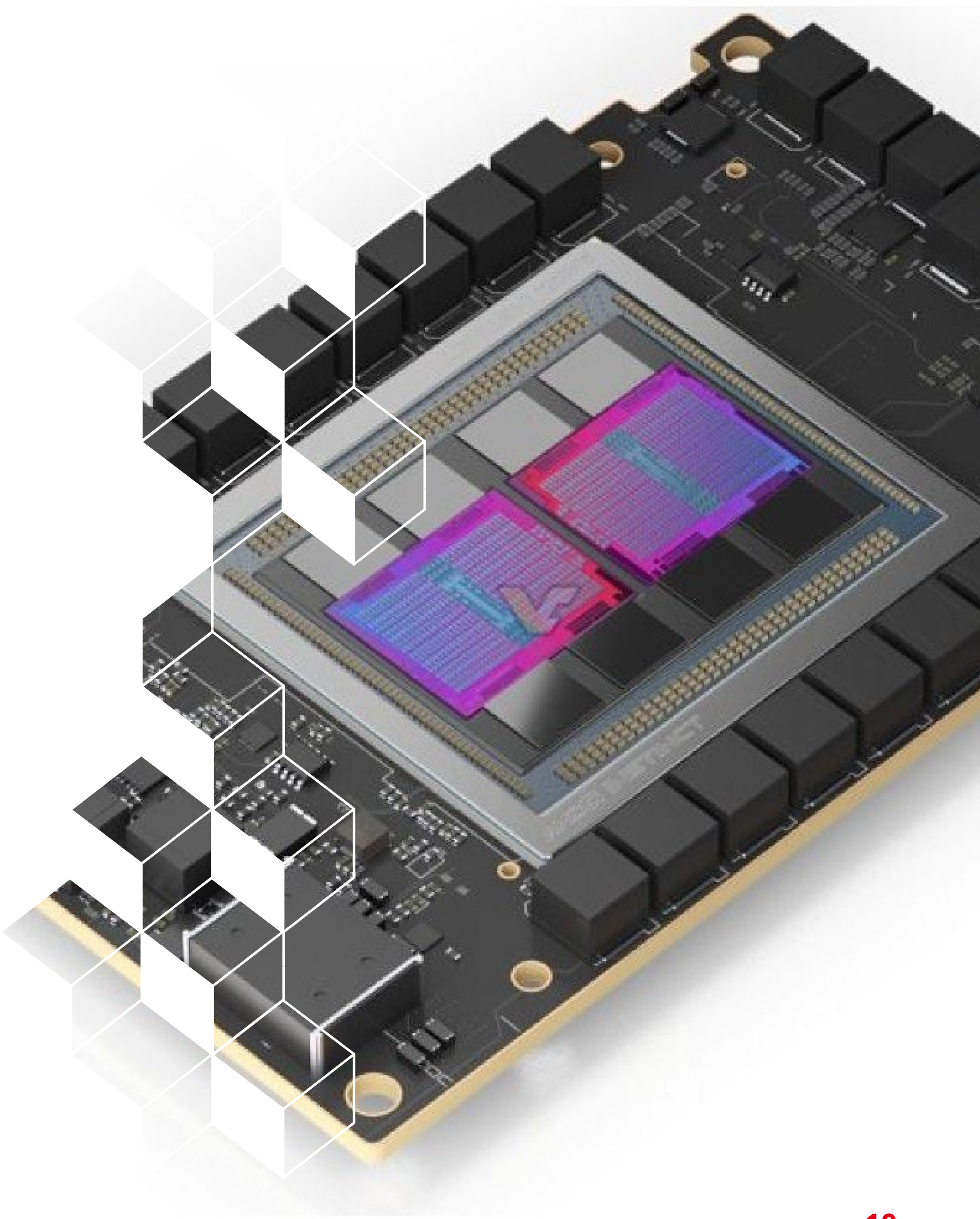
Developers must :

- Update their knowledge to handle new hardware specificity and programming models
- Rewrite or duplicate part of their applications
- Bet on a technology without long-term vision

- A conservative estimate from the Kokkos team : An application of 200 000 lines need a full-time engineer during a year to switch programming models
- We have been working for more than 4 years to port the SMILEI application and we are still on it
- Easier for large application teams with dedicated HPC engineers
- Impossible for applications maintained by a single physicist, PhDs or Postdocs

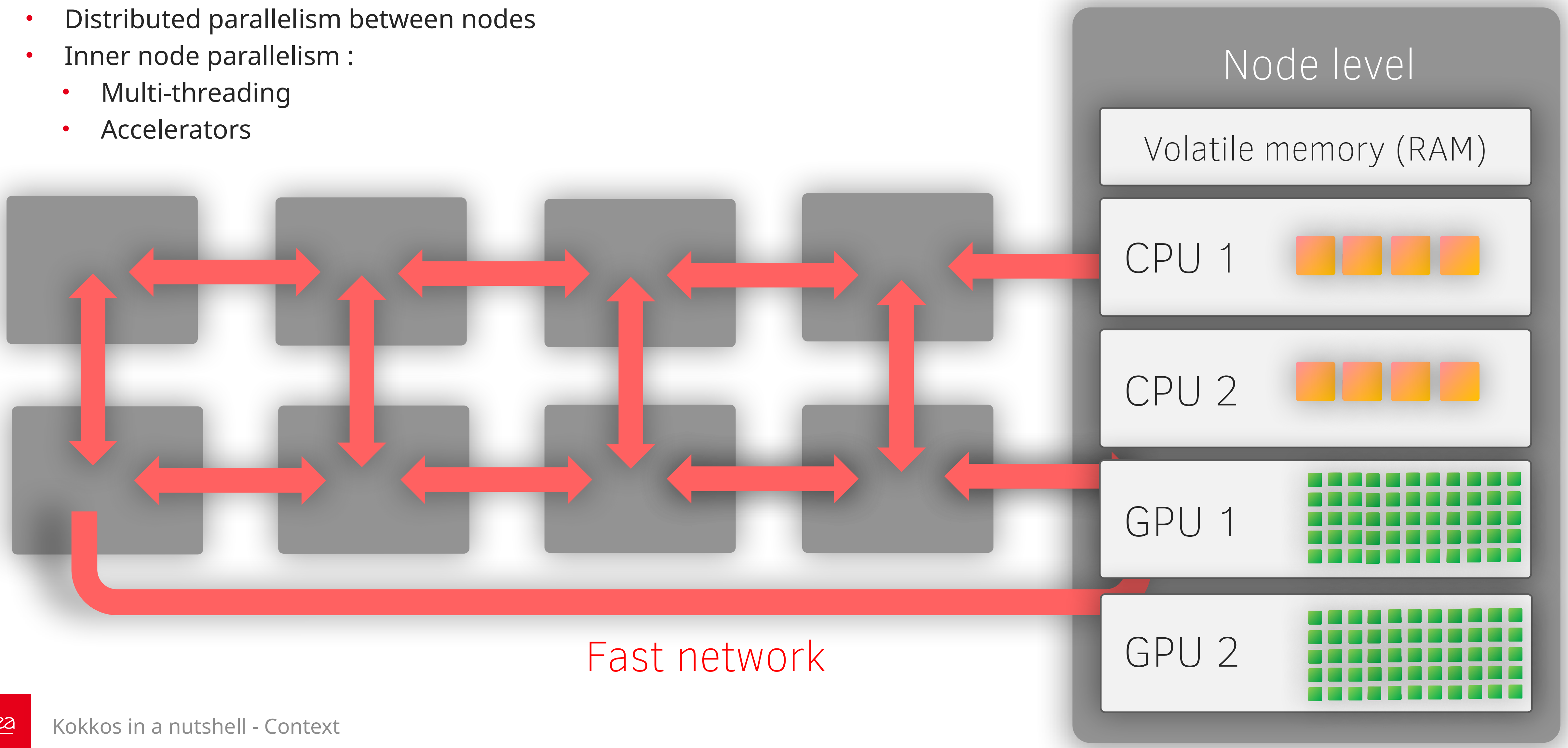
Current state of HPC hardware

- Super-computers tend to be more and more heterogeneous : The CPU is coupled with one or multiple accelerator cards, most often GPUs
- The pure computational power is not balanced, mostly dominated by the GPU side
- NVIDIA used to dominate the HPC GPU market (as Intel used to be for CPUs)
  - Good for technology stability (programming models, optimization, etc)
  - Bad for market price and innovation
- Today's landscape is composed of many vendors with emergent actors :
  - AMD CPUs and GPUs
  - ARM based processors and accelerators
  - Intel GPUs



How to program heterogeneous systems

- Super-computers still have multiple parallelism layers
  - Distributed parallelism between nodes
  - Inner node parallelism :
    - Multi-threading
    - Accelerators



Need for performance portable programming models

- Developers still need a model MPI + X so far, may change in the future

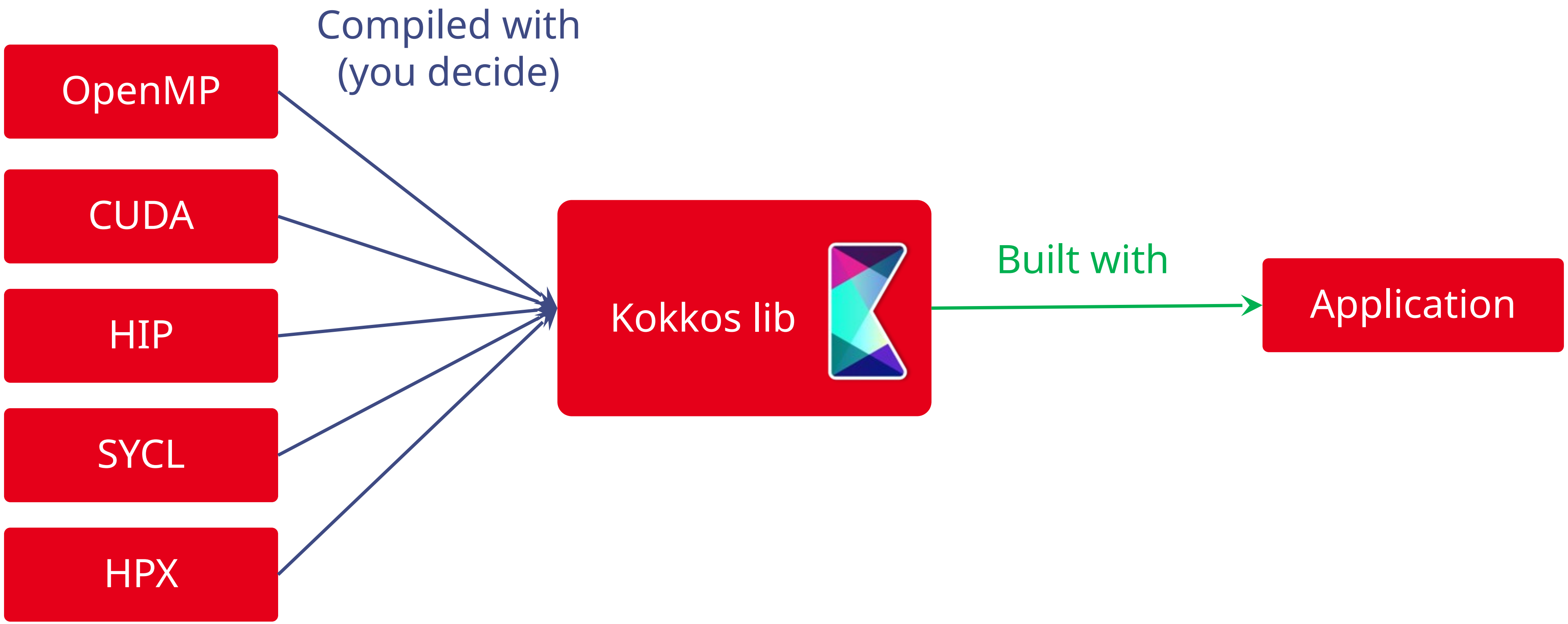
- A performance portable programming model offers decent performance across a wide range of architectures using a single source code

- The choice depends on the value of the cursor between many parameters :
  - Performance
  - Portability
  - Maturity (bugs, features)
  - Long-term support
  - Code maintainability
  - Programing complexity (required programming skilled)
  - Ecosystem and interoperability

2. What is Kokkos

Kokkos, what is it ?

Kokkos is performance portability parallel programming model build upon the C++-17 standard designed to abstract already-existing parallel programming models

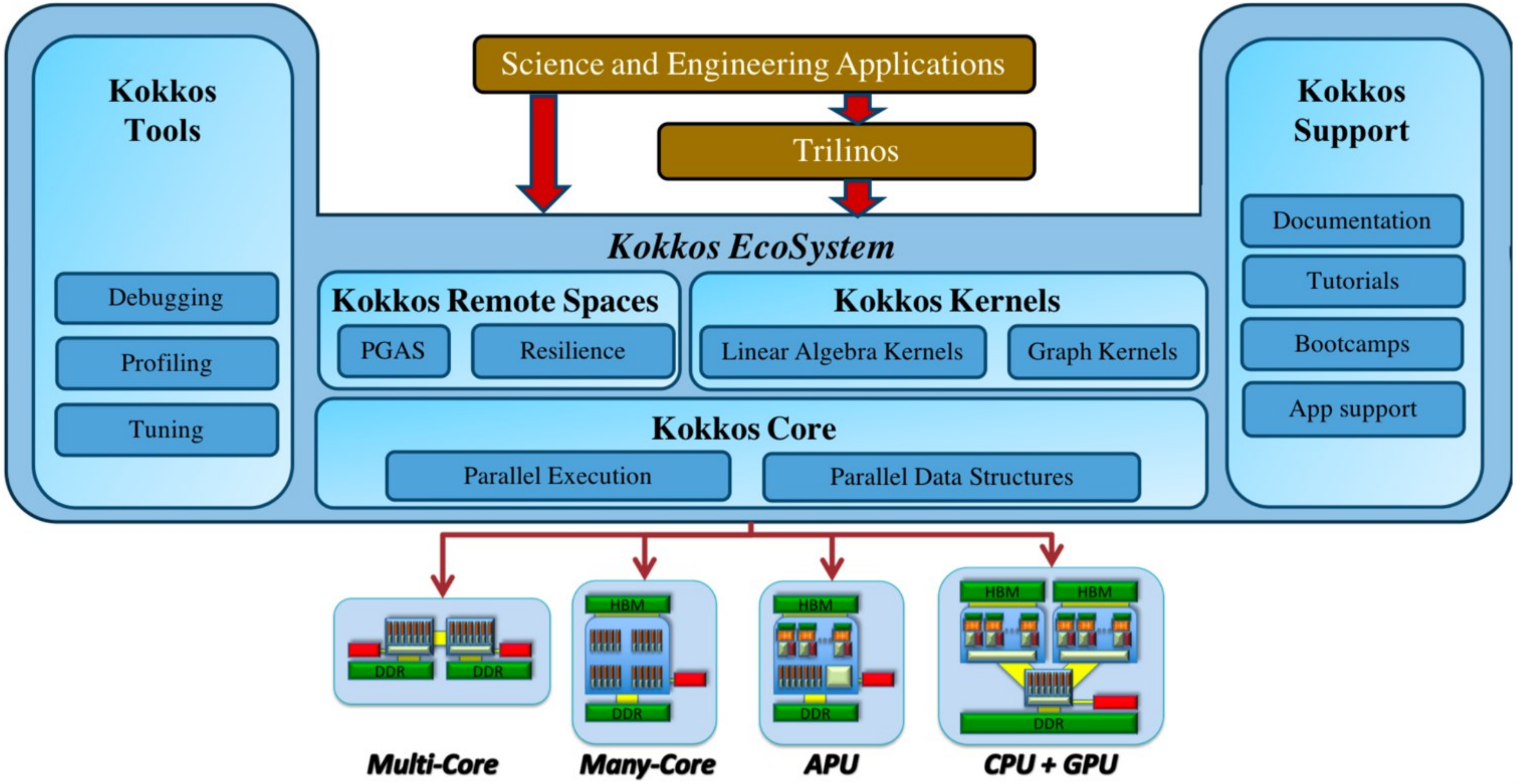


Kokkos, what is it ?

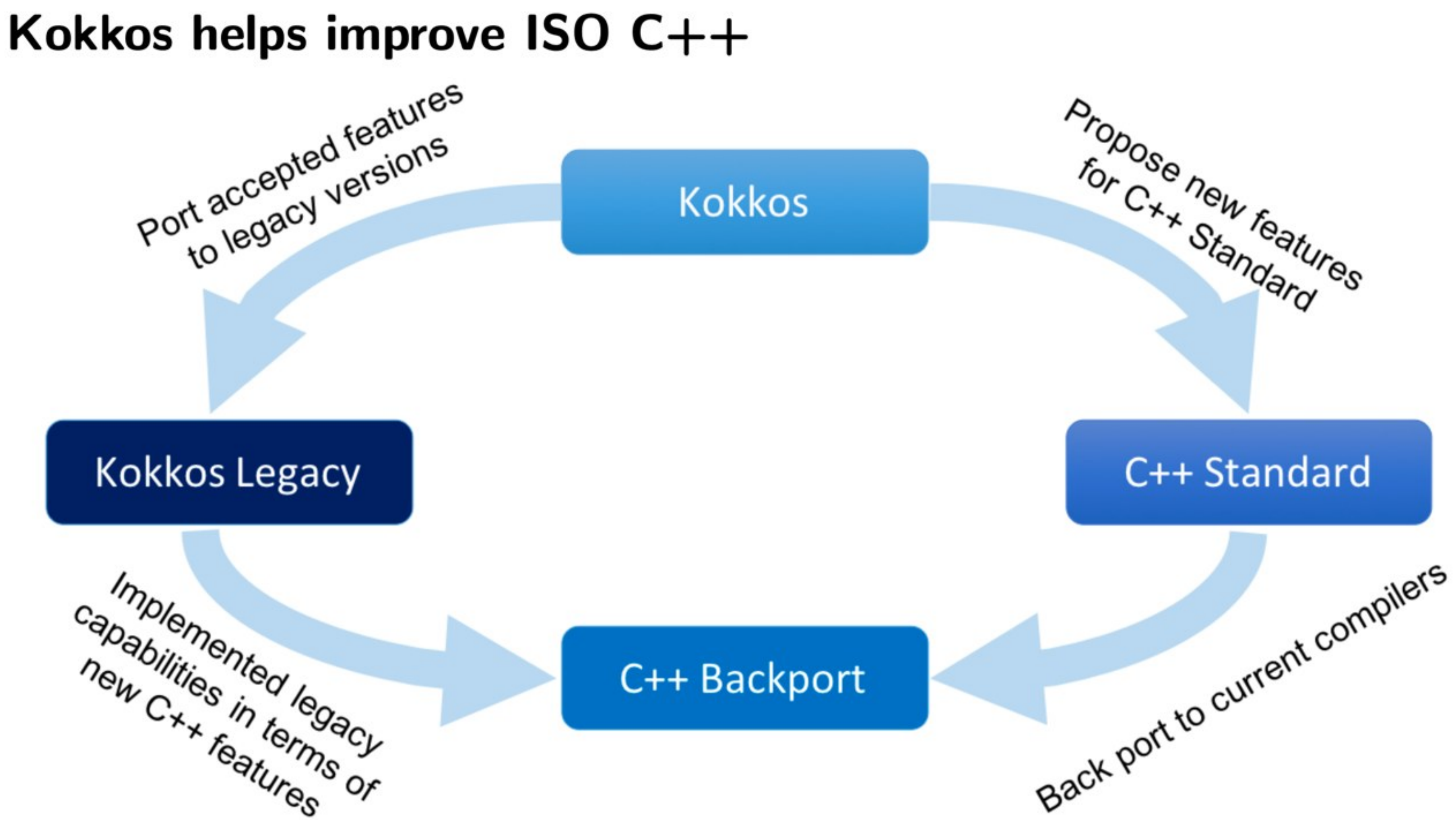
Kokkos is a good trade-off:

- Portable: compile for all CPU and GPU of the market and have access to experimental hardware thanks to close contact with vendors
- Performance: build on top of vendor-specific back-ends (i.e. CUDA, HIP, SYCL)
- Maturity: well established project since 2012 adopted by more than 100 projects
- Long-term support: used and developed by many DOE labs, part of the ECP project
- Maintainability: descriptive single source code
- Complexity: build upon advanced C++ without the need to master it, common objects and functions used in numerical science, extensive documentation, tutorial materials, chat room for questions
- Ecosystem and interoperability: expanding solution for common needs of modern science and engineering codes (math libs, debuggers, performance analysis)

The whole ecosystem picture



Kokkos helps improve ISO C++



Ten current or former Kokkos members are members of the ISO C++ standard committee

Kokkos main capability

Basic features:

- Parallel loop: one-dimension, multi-dimensions, reduction patterns and more like OpenMP
- Multidimensional arrays like Fortran or Python
- Memory and execution policy to decide where data is located and where kernels are run
- Implicit data layout and data access management for performance

More advanced features:

- Thread safety, thread scalability and atomic operation
- Hierarchical parallelism (threading, vectorization, SIMT, etc.)
- Optimization capability

Tools:

- Compatibility with classical debuggers and profilers
- Build-in algorithm (sorting) like Thrust and mathematic features (linear algebra)
- Interoperability with Python, Fortran and other programming models

However, be careful, Kokkos is not magic

Portability and performance portability are not the same

- Hardware optimized algorithms may not scale
- Best performance with a single source implementation is not always possible, especially targeting both CPU and GPU, due to hardware differences
  - May need specific algorithm or parallelism hierarchies to leverage the maximum performance of a specific architecture
  - Kokkos will not do that for you
  - Trade-off between best performance and other goals (portability, etc)

3. How to use Kokkos

Kokkos Uses the concept of data parallelism as OpenMP does

```
for (int j = 0 ; j < column_size ; ++j) {
        for (int i = 0 ; i < line_size ; ++i) {
                y[j] += x[i] * A[j][i];
        }
}
```

Pattern: structure of the computation (for, reduction, scan, graph)

Execution policy: how computations are executed (static, dynamic, task)

Body: code which performs each unit of work

OpenMP versus Kokkos for a simple loop

```
#pragma omp parallel for
for (int j = 0 ; j < column_size ; ++j) {
    for (int i = 0 ; i < line_size ; ++i) {
        y[j] += x[i] * A[j][i];
    }
}
```

```
parallel_for( column_size,
        KOKKOS_LAMBDA(const int j) {
              for (int i = 0 ; i < line_size ; +
+i) {
                            y(j) += x(i) * A(j,i);
                   }
        }
)
```

- Kokkos syntax is different but still understandable as OpenMP
- Kokkos uses the notion of lambda function in C++: small structure that describes a function

Execution pattern

What basic pattern to execute:

- Kokkos::parallel_for -
- Kokkos::parallel_reduce
- Kokkos::parallel_scan
- etc

```
#pragma omp parallel for reduction(+:sum)
for (int i = 0 ; i < N ; ++i) {
    sum += x[i];
}
```

```
parallel_reduce( N,
        KOKKOS_LAMBDA(const int i, double& sum) {
                sum += x[i];
        },
sum)
```

Execution space

Where computation is executed:
- Kokkos::serial – Serial execution on CPU
- Kokkos::DefaultHostExecutionSpace – Host execution
- Kokkos::DefaultExecutionSpace - Device if compiled for GPU, else Host

Execution space can also make the back-end explicit:
- Kokkos::Cuda
- Kokkos::OpenMP
- Kokkos::HIP

The selection depends on:
- Compile-time options
- Default choices
- Run-time choices

Warning: Kokkos only sees a single device per process. Therefore, multi-device requires MPI.

Execution policy

How computation is executed:

- Kokkos::RangePolicy – 1D loop
- Kokkos::MDRangePolicy – multi-D loop
- Kokkos::TeamPolicy – for hierarchical parallelism

- Adapt the parallel execution to the hardware and to the characteristics of the algorithm

```
parallel_for( N,
   KOKKOS_LAMBDA (int n) { /* ... */ }
);
```

```
parallel_for(
        RangePolicy<DefaultExecutionSpace>(0, N),
   KOKKOS_LAMBDA(int n) { /* ... */ }
);
```

OpenMP vs Kokkos

Case 1:

- I only want to use the CPU (ignore the GPU if exists)
- I want to use OpenMP backend
- I compile Kokkos only with the OpenMP backend

```
// OpenMP
#pragma omp parallel for
for (int i = 0 ; i < N ; ++i) {
    /* ... */
}
```

```
// Kokkos
Kokkos::parallel_for( N,
    KOKKOS_LAMBDA (int i) { /* ... */ }
);
```

```
// Kokkos explicit execution policy
Kokkos::parallel_for(
Kokkos::RangePolicy<Kokkos::DefaultExecutionSpace
>(        0, N ),
        KOKKOS_LAMBDA(int n) {
                /* ... */
        }
);
```

```
// Kokkos explicited execution policy
Kokkos::parallel_for(
Kokkos::RangePolicy<Kokkos::OpenMP>( 0, N ),
        KOKKOS_LAMBDA(int n) {
                /* ... */
        }
);
```

OpenMP vs Kokkos

Case 2:

- I use a hybrid node with a CPU and a GPU
- I compile Kokkos with OpenMP for the CPU and with CUDA for the GPU

```
// OpenMP CPU code
#pragma omp parallel for
for (int i = 0 ; i < N ; ++i) {
    /* ... */
}
```

```
// Kokkos CPU code
parallel_for(
RangePolicy<DefaultHostExecutionSpace>(0, N ),
        KOKKOS_LAMBDA(int n) {
                /* ... */
        }
);
```

```
// OpenMP GPU code
#pragma omp target teams distribute parallel for
for (int i = 0 ; i < N ; ++i) {
    /* ... */
}
```

```
// Kokkos GPU code
parallel_for(
        RangePolicy<DefaultExecutionSpace>( 0, N
),
        KOKKOS_LAMBDA(int n) {
                /* ... */
        }
);
```

Loops and debugging

- Each loop can be assigned a name to facilitate the debugging part

```
// Kokkos
Kokkos::parallel_for( N,
   KOKKOS_LAMBDA (int i) { /* ... */ }
);
```

```
// Kokkos
Kokkos::parallel_for("my loop", N,
   KOKKOS_LAMBDA (int i) { /* ... */ }
);
```

Multi-D Arrays equivalent = Kokkos::view

•Kokkos::view is a class designed to represent a multi-dimensional array with additional capabilities:

- Simple accessor (i, j ,k …)
- Abstracted or explicit memory layout (row major, column major, etc.)
- Static or dynamic dimensions
- Resize capacity like std::vector
- Memory space – where the array is stored
- Memory Traits – additional properties (atomic operation, shared memory, etc.)
- Can be assigned a name for debugging
- Shallow copy by default like Python (reference counting)

Kokkos::view simple examples

```
// simple 1d array
View <int*> A ("A",N);

// 3d dynamic array
View <double***> A ("A",Nx, Ny, Nz);

// simple 1d array with static dimension
View<char[4]> A ("A");

// partly static/dynamic 2d array
View<float*[4]> A ("A",N);
```

Memory Space

- On heterogeneous systems, devices have a separate memory from the RAM
- Memory space enables to decide where the view data is located:
    - Kokkos::HostSpace
    - Kokkos::CudaSpace
    - Kokkos::HIPSpace
    - Kokkos::SharedSpace – for unified memory between host and device

- By default, the memory space is the one of the default Execution Space (should be the device if using it)

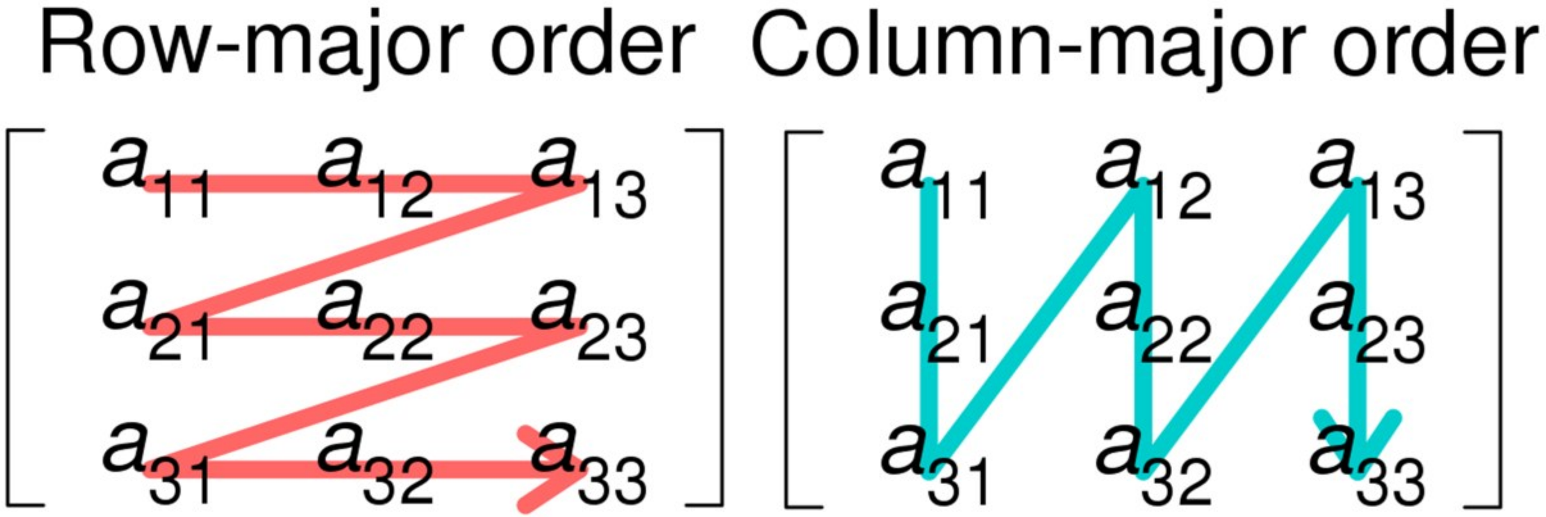
- Contratry to OpenMP or OpenACC, no need to map a host and device view

Kokkos::view examples with Memory Space

```
// simple 1d dynamic array
View <int*, HostSpace> A ("A",N);

// simple 1d dynamic array
View <int*, DefaultExecutionSpace::memory_space> A ("A",N);


// 3d dynamic array allocated in the device memory using CUDA
View <double***, CudaSpace> A ("A",Nx, Ny, Nz);
```

Memory layout

- The layout is how the memory is stored and organized
- The data layout can impact the performance and depends on the computing architecture
- Kokkos determines the suitable layout by default for the memory space



- Kokkos::LayoutRight – Row-major order
- Kokkos::LayoutLeft – Column-major order
- Kokkos::LayoutStride – custom ordering

Kokkos::view examples with Layout

```
// simple 2d array (default layout)
View <int**, HostSpace> A ("A",N, M);

// simple 2d array
View <int**, HostSpace, LayoutRight> A ("A",N, M);
View <int**, HostSpace, LayoutLeft> A ("A",N, M);
```

Memory traits

- Memory traits are additional properties given to the view on how the data is accessed
  - Kokkos::Unmanaged – allocation is not managed by Kokkos, can be useful to map a view with an already existing array (via std::vector or cudaMalloc for instance)
  - Kokkos::Atomics – atomics operations on the array elements
  - Kokkos::RandomAccess – optimize for random access. If the view is also const this will trigger special load operations on GPUs (i.e. texture fetches).
  - Kokkos::restrict - There is no aliasing of the view by other data structures in the current scope

Building applications with Kokkos

- Kokkos primary build system is CMAKE
- Makefile can be designed for simple projects

- Kokkos can be built and installed easily from sources as a library (recommended for large applications)
- It can be built as well inline
- It can be pulled via Spack

- The back-end to use is provided at compile time, for instance
  - -Dkokkos_ENABLE_CUDA=ON
  - -Dkokkos_ENABLE_OPENMP=ON

- One CPU, one GPU and one serial backend at a time
- Device architecture can as well be provided if not detected by default

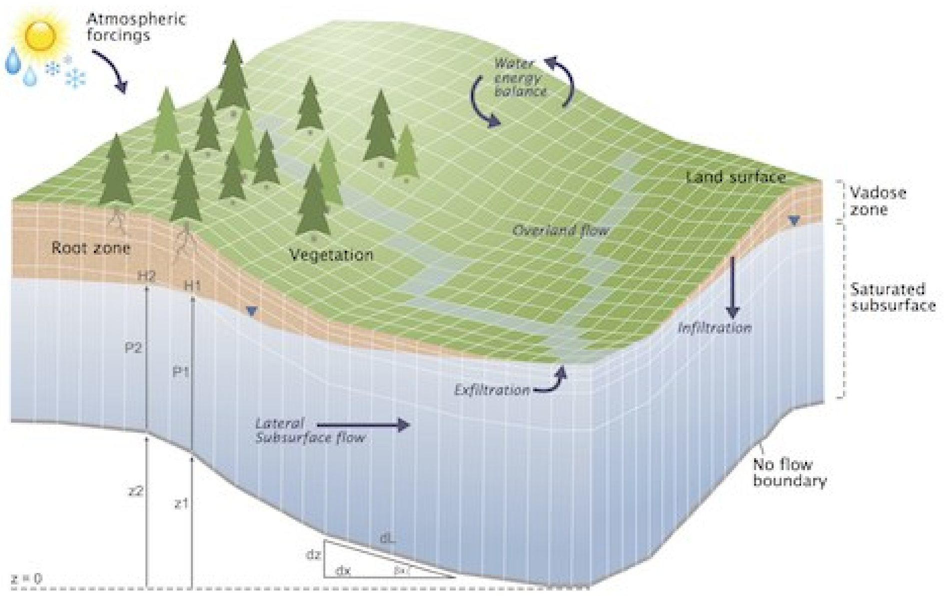
Kokkos::view examples with Layout

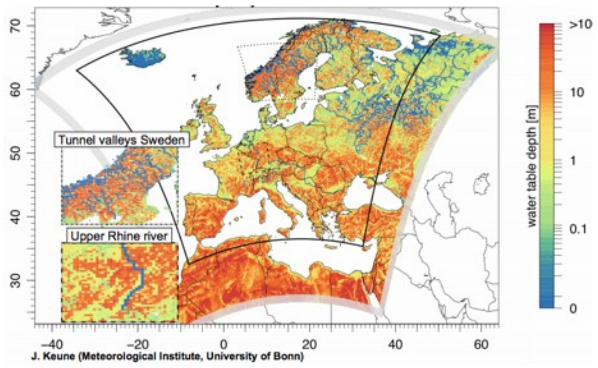
```
int main(int argc, char* argv[])
{


  constexpr int nvar = 2; // compile time size
  int nx = 100;                    // run time
size
  double* array = new double[nx*nvar];

  for (int ix=0; ix<nx; ++ix)
  {
    array[ix*2 + 0] = 1.0*ix;
    array[ix*2 + 1] = 2.0*ix;
  }

  return 0;
}
```

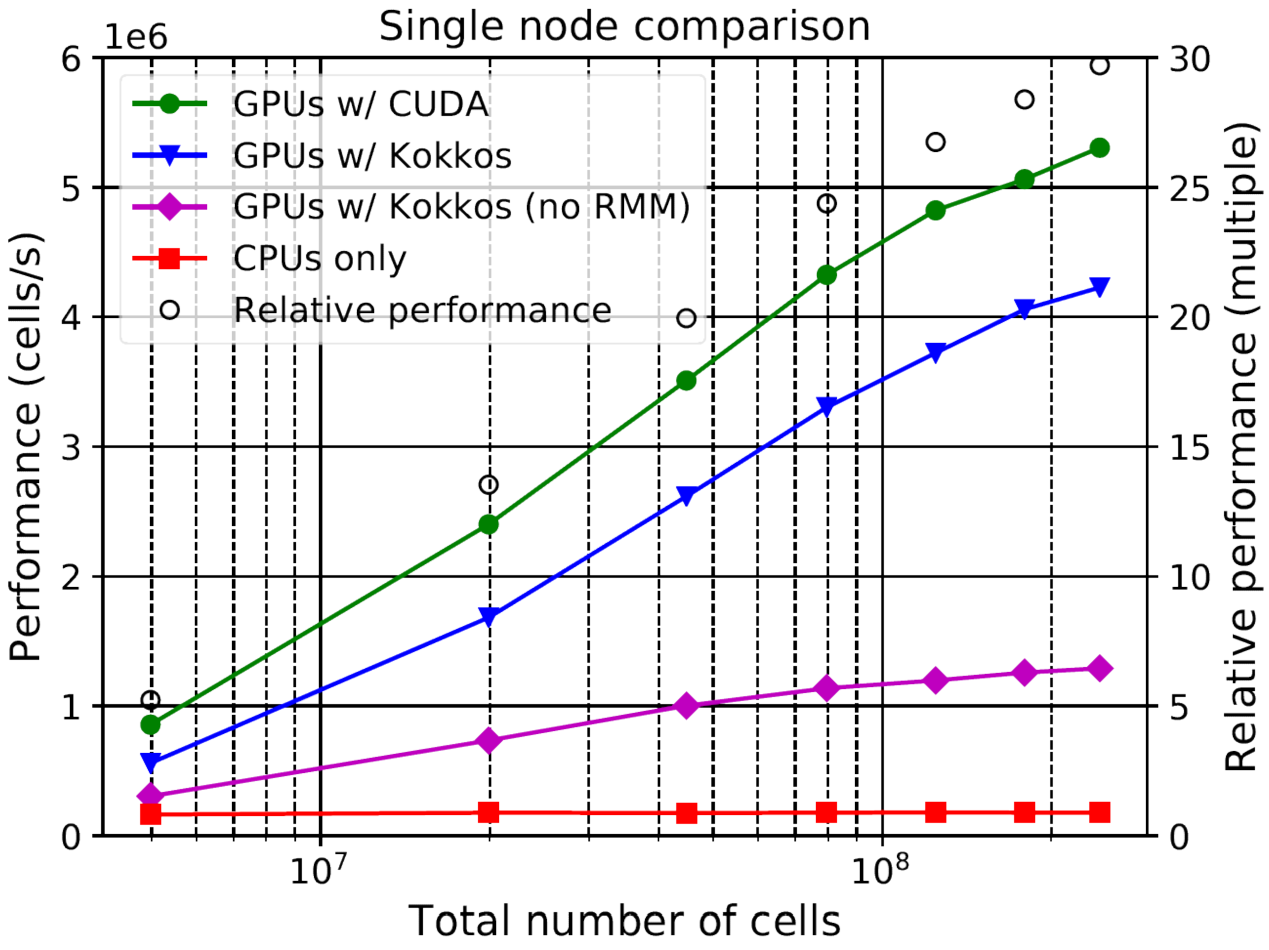
```
int main(int argc, char* argv[])
{
    Kokkos::ScopeGuard  scope(argc,  argv);  //
initialize & finalize

  constexpr int nvar = 2; // compile time size
   int  nx  =  100;                      //  run  time
size
    Kokkos::View<double*[nvar]>  array("Array",
nx);

   Kokkos::parallel_for(nx,  KOKKOS_LAMBDA  (int
ix)
  {
    array(ix, 0) = 1.0*ix;
    array(ix, 1) = 2.0*ix;
  });

  return 0;
}
```

4. Real-life examples

EoCoE-II: porting ParFlow to GPU

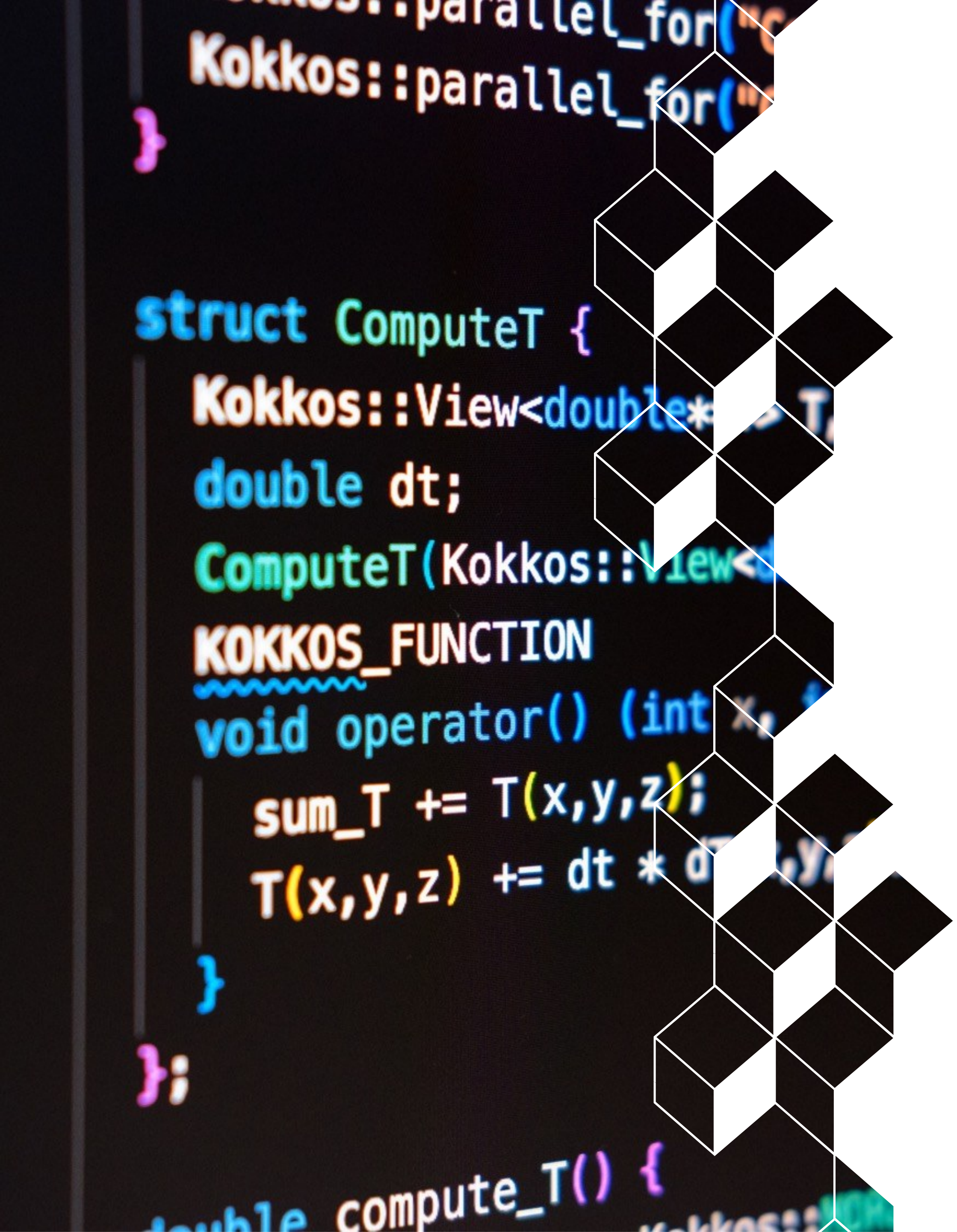






Kokkos, an excellent tool for students

- A perfect choice for students starting a new code:
  - Students learn how to use a cutting-edge technology
  - They can sell it in research and industry (better than Fortran)
  - Facilitate the creation of new applications
  - Facilitate the GPU porting of these applications (learning slope much faster than any programming model)
  - Applications are ready to run on most-advanced super-computers whatever the architecture without any strong expertise (just need to add the distributed parallelism)



Conclusion

- Kokkos is a C++ meta-programming performance portable library
- Kokkos enables a single-source implementation to run on multiple architectures (CPU, GPU) efficiently
- Target both new users and advanced C++ users, no need to be a C++ expert to understand the basic concepts
- Simple things stay simple
- Not a research library, designed for productivity
- Combine advantages of many existing programming models and languages
- Suitable for scientific applications (multi-D array, linear algebra, etc)

Merci pour votre attention