GeantV Geometry: SIMD abstraction and interfacing with CUDA

Johannes de Fine Licht (johannes.definelicht@cern.ch)
Sandro Wenzel (sandro.wenzel@cern.ch)

*Thanks to **Michal Husejko** and **Romain Wartel** at **TechLab** for providing hardware.*

Abstracted algorithms

- Write **common code** for multiple types of SIMD **“backends”** (Vc/Cilk/CUDA).
- Eases code maintenance; requires only one kernel to be written for a given functionality
- **Must retain performance of a low level implementation**

Illustrating scalar/SIMD abstraction and kernels

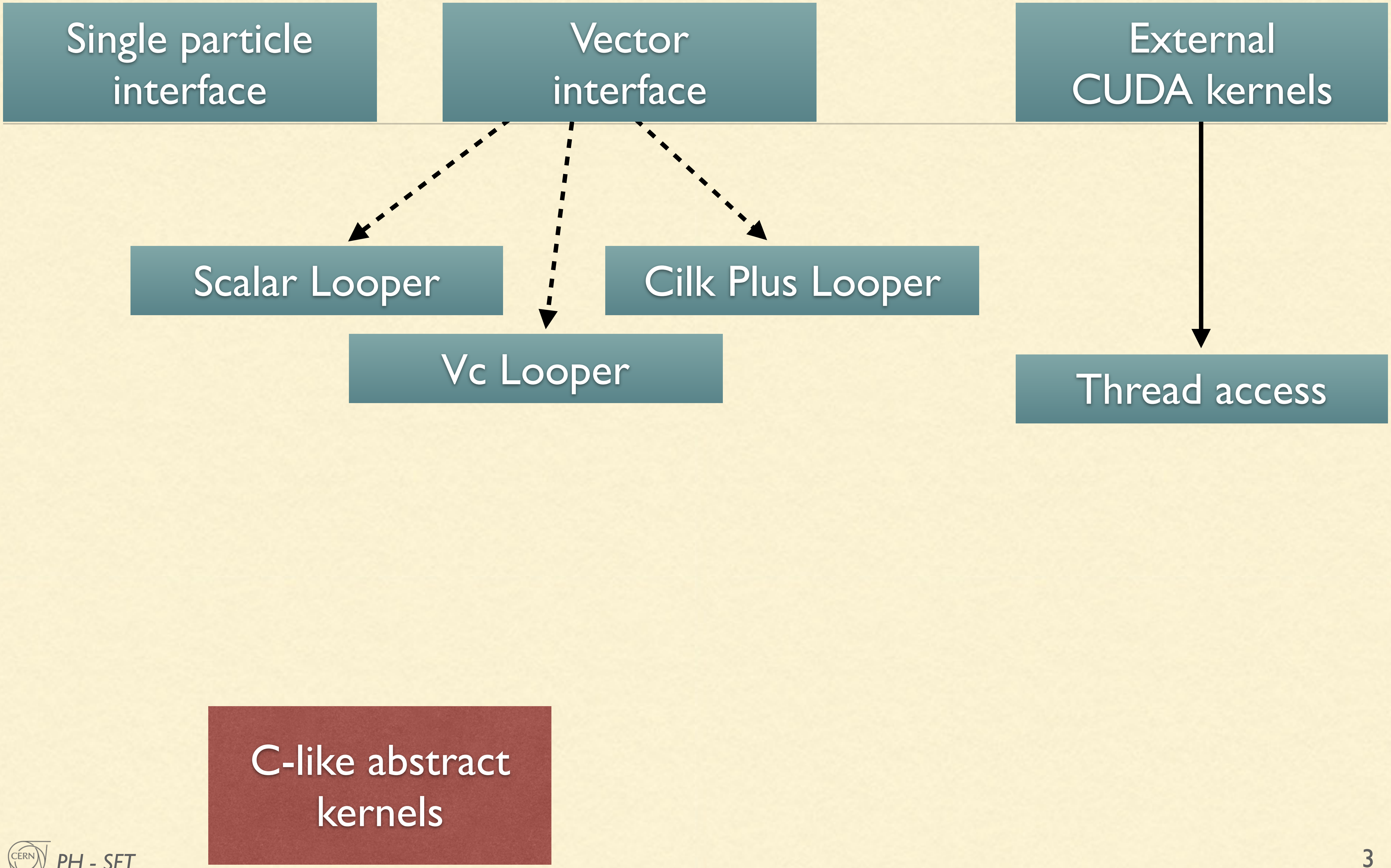
Single particle interface

Vector interface

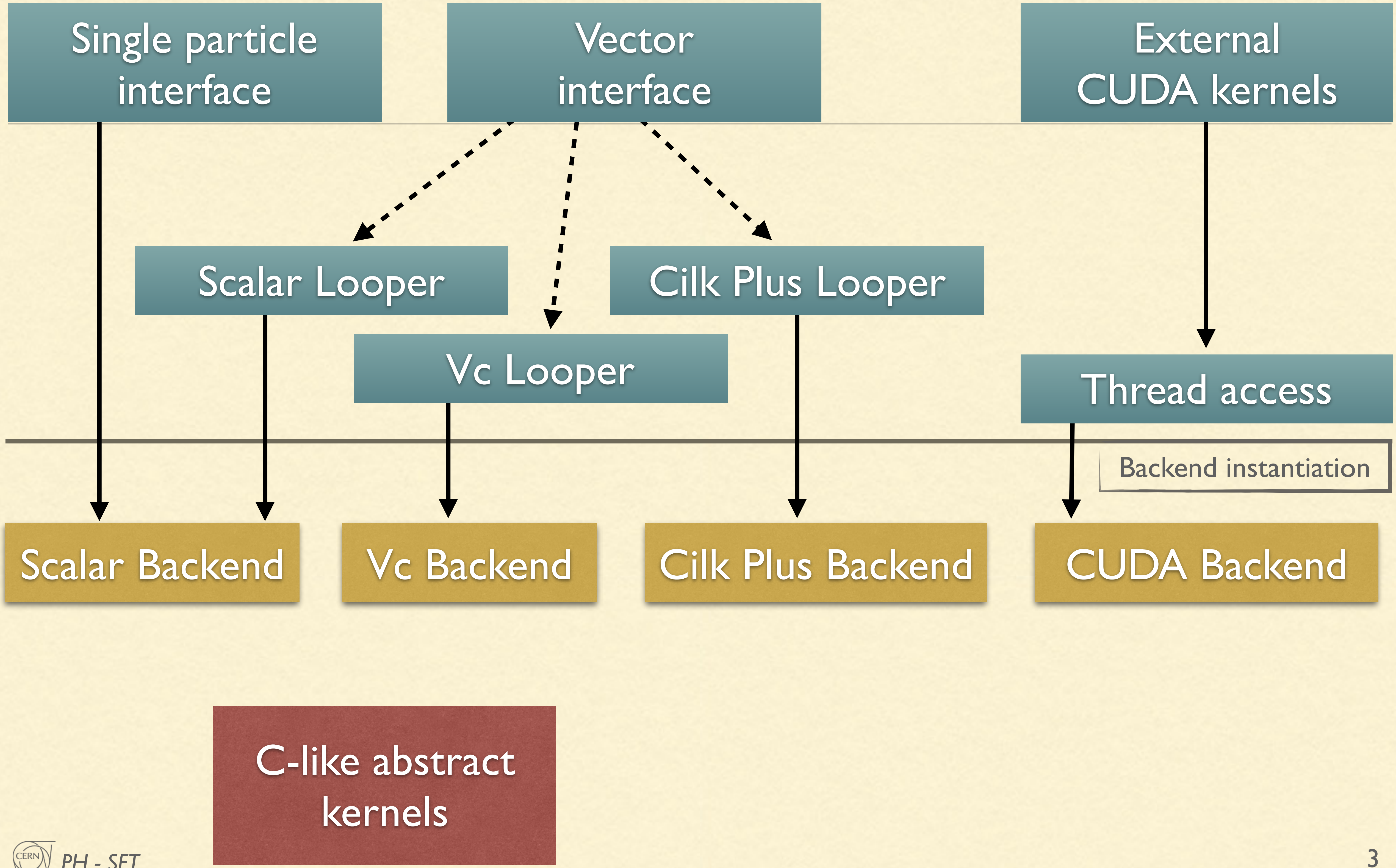
External CUDA kernels

C-like abstract kernels

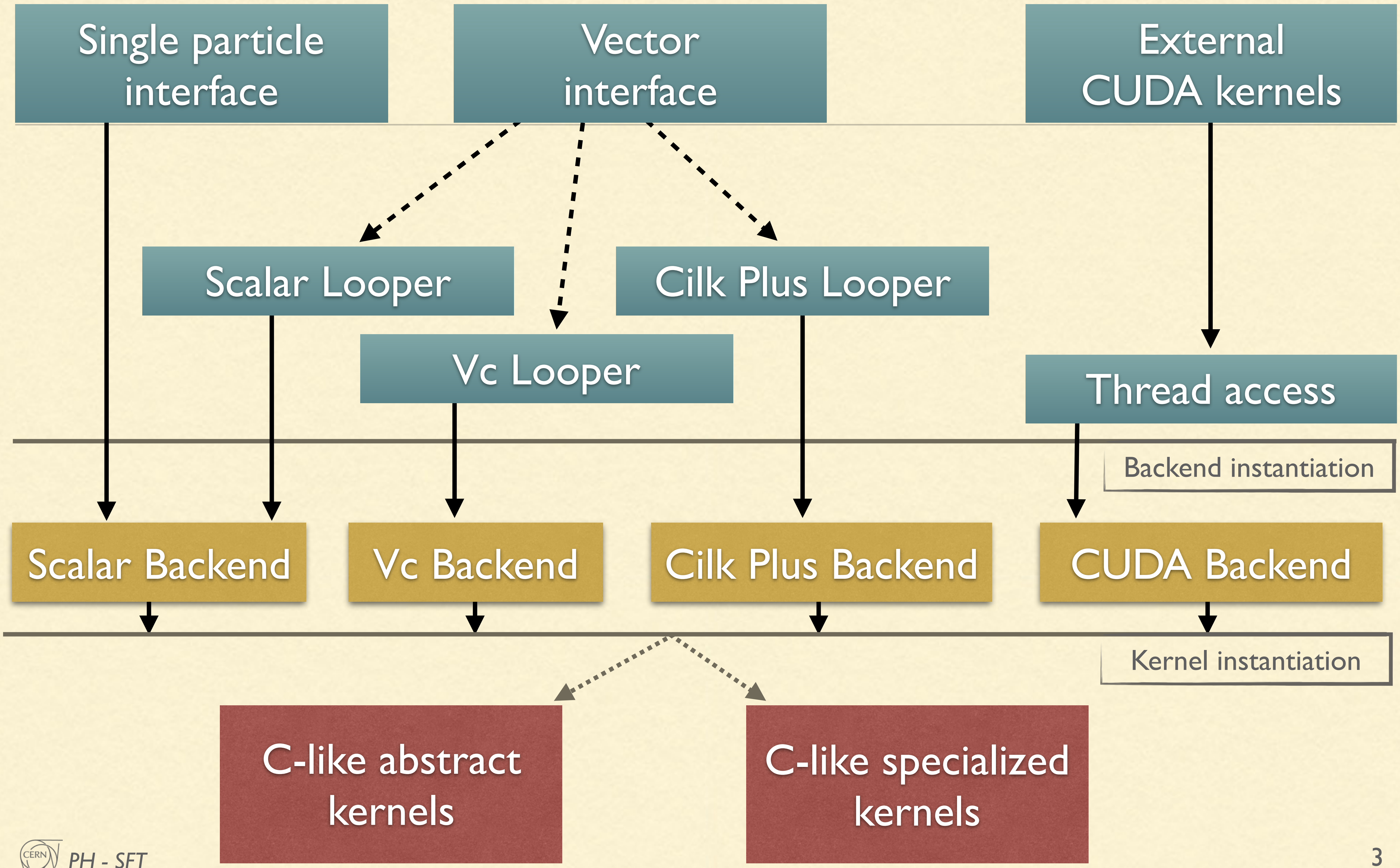
Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels



Generic kernels on abstract types

```
template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
void BoxDistanceToIn(
```

- Generic programming; write algorithms in a **generic** way on **abstract** types

```
    typedef typename Backend::precision_v Float;
    typedef typename Backend::bool_v Bool;
```

```
    Float dist_x = (dimensions[0] - pos[0]) * dir_x_inv;
```

- Implementation depends on the backend

- **Backend** determines the types on which the high level operations are performed

```
struct kVc {
  typedef Vc::int_v                      int_v;
  typedef Vc::Vector<Precision>          precision_v;
  typedef Vc::Vector<Precision>::Mask bool_v;
  constexpr static bool early_returns = false;
  const static precision_v kOne;
  const static precision_v kZero;
  const static bool_v kTrue;
  const static bool_v kFalse;
};
```

Generic kernels on abstract types

```
template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
void BoxDistanceToIn(
```

- Generic programming; write algorithms in a **generic** way on **abstract** types

```
  typedef typename Backend::precision_v Float;
  typedef typename Backend::bool_v Bool;
```

```
  Float dist_x = (dimensions[0] - pos[0]) * dir_x_inv;
```

- Implementation depends on the backend

- **Backend** determines the types on which the high level operations are performed

```
struct kVc {
  typedef Vc::int_v                      int_v;
  typedef Vc::Vector<Precision>          precision_v;
  typedef Vc::Vector<Precision>::Mask bool_v;
  constexpr static bool early_returns = false;
  const static precision_v kOne;
  const static precision_v kZero;
  const static bool_v kTrue;
  const static bool_v kFalse;
};
```

Implementations wrapped in types

- Operations are abstracted

```
safety[0] = Abs(pos_local[0]) - dimensions[0];
```

- Backends implement all operations

```
VcPrecision Abs(VcPrecision const &val) {
  return Vc::abs(val);
}
```

```
CilkPrecision Abs(CilkPrecision const &val) {
  CilkPrecision result;
  result.Map(std::fabs);
  return result;
}
```

```
Type Abs(const Type val) {
  return fabs(val);
}
```

- Vc already supports arithmetics
- CUDA uses primitive types
- Wrapper structs are implemented if needed

```
template <typename Type, int vec_size>
struct CilkVector {

  Type __attribute__((align(64))) vec[vec_size];
```

Interfacing with CUDA

- Involving the GPU should be easy

```
CreateRootGeometry();

RootManager::Instance().LoadRootGeometry();
CudaManager::Instance().LoadGeometry();
CudaManager::Instance().Synchronize();
```

- Principle of <u>common code</u>

- *Issue:* GPU code needs nvcc, but we want a different compiler for CPU

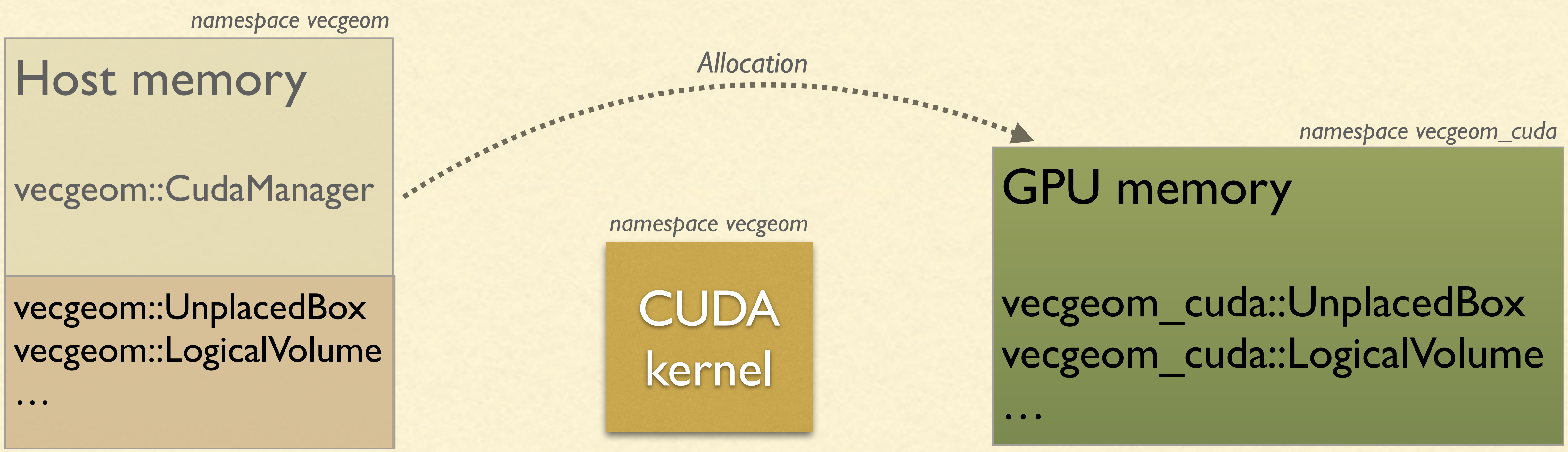
GNU/Clang: C++11 Vc Cilk	vs.	nvcc: CUDA

Dual namespaces

- Utilize distinct namespaces but **same code** for CPU and GPU
- Classes live in **one** or **both** environments
- Provide abstraction to interface between environments

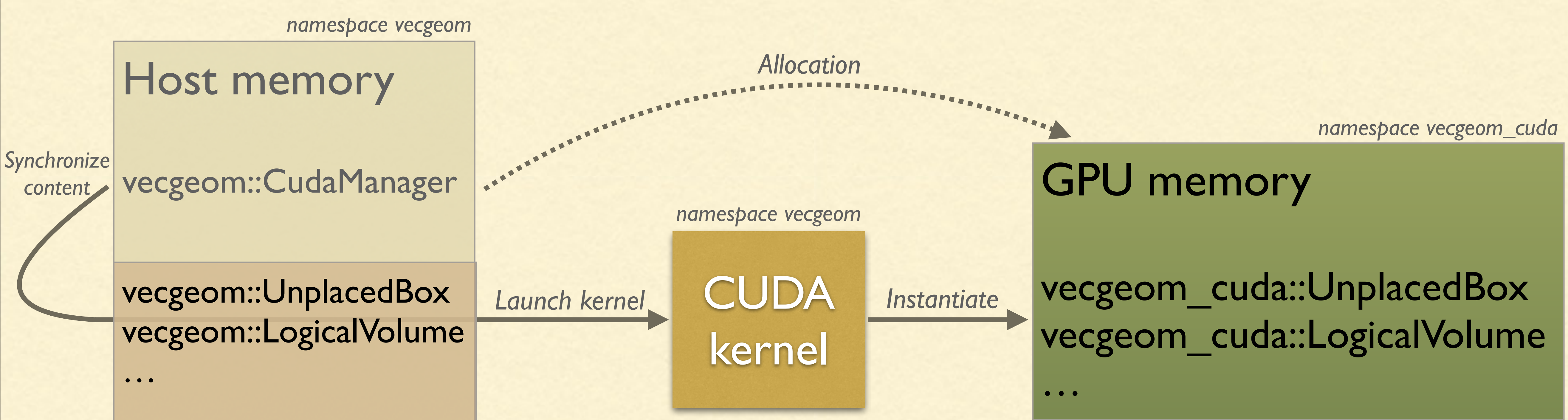
Dual namespaces

- Utilize distinct namespaces but **same code** for CPU and GPU
- Classes live in **one** or **both** environments
- Provide abstraction to interface between environments

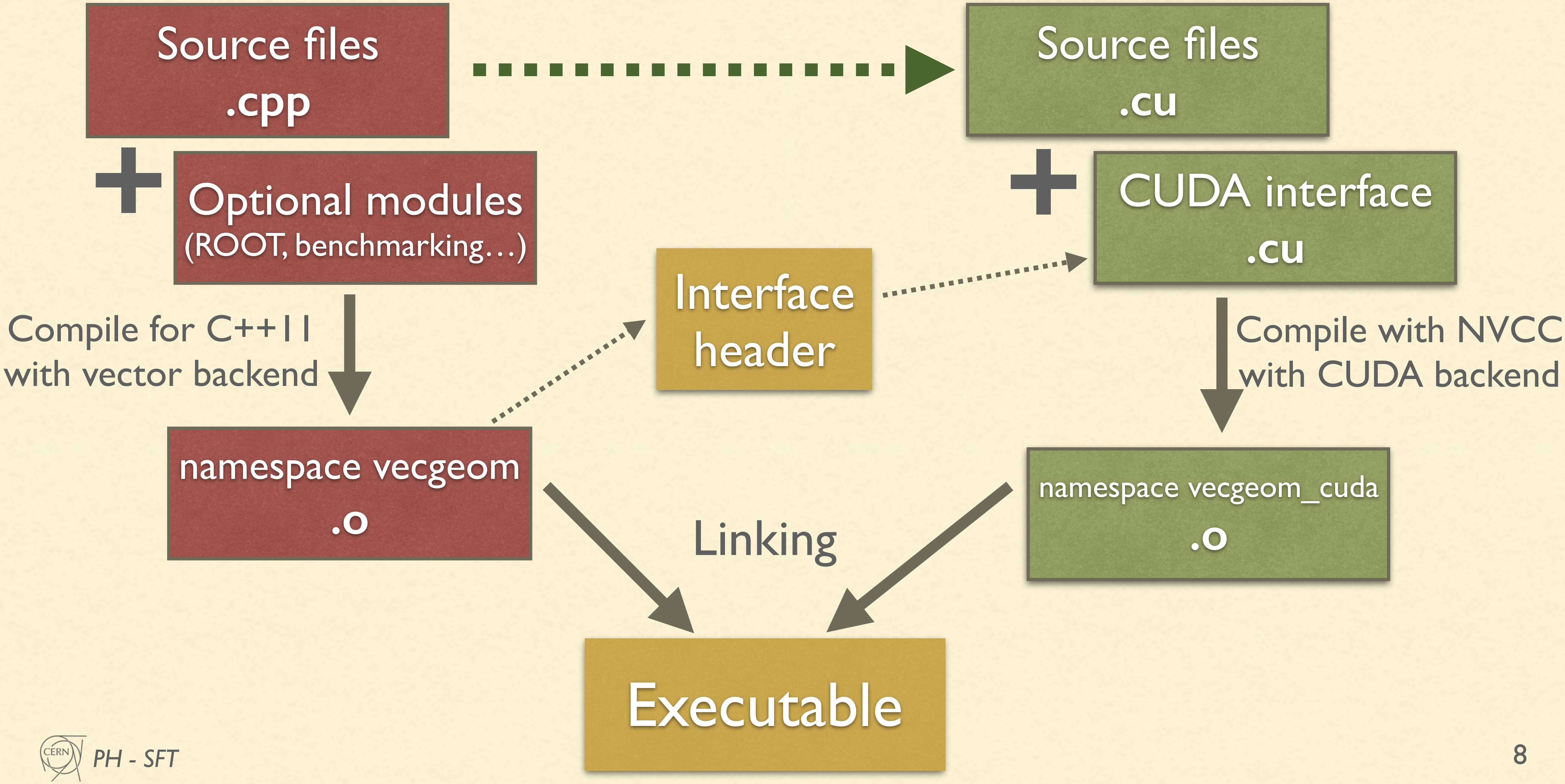


Dual namespaces

- Utilize distinct namespaces but **same code** for CPU and GPU
- Classes live in **one** or **both** environments
- Provide abstraction to interface between environments

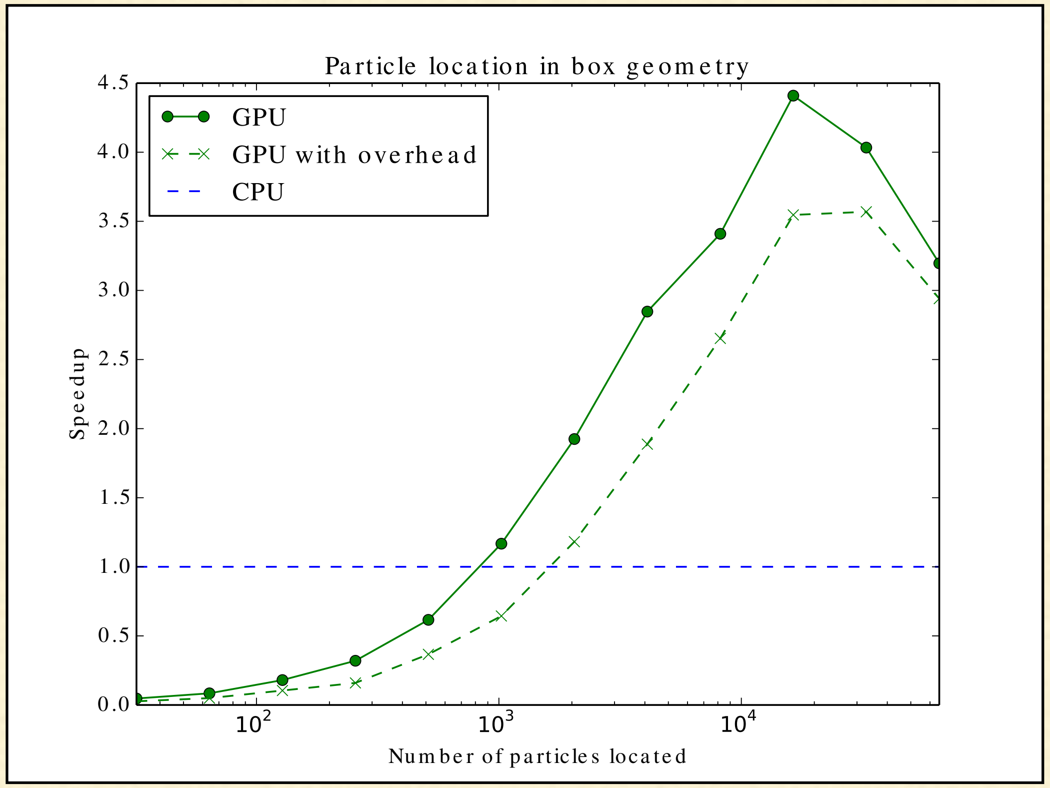


Compilation scheme



Status

- Common code for box methods runs for scalar, Vc, Cilk and CUDA backends
- Implemented GPU memory synchronization
- Dual namespaces allows dispatching work to CPU or GPU
- Location can run in either environment
- **Results are in agreement**



Preliminary benchmarks run in hybrid CPU/GPU environment for four-level box geometry. Location of particles is a tree algorithm, so speedups are only seen at very high particle multiplicity.

Johannes de Fine Licht (johannes.definelicht@cern.ch)

Appendix: CPU/GPU same interface

Main executable compiled with C++ compiler

```
for (int i = 0; i < n; ++i) {
  NavigationState path(depth);
  results[i] =
      navigator.LocatePoint(GeoManager::Instance().world(), points[i], path,
                            true)->id();
}
```

Same code used

```
template <typename TrackContainer>
__global__
void CudaManagerLocatePointsKernel(
    vecgeom_cuda::VPlacedVolume const *const world,
    vecgeom_cuda::SimpleNavigator const *const navigator,
    vecgeom_cuda::NavigationState *const paths,
    TrackContainer const *const points, const int n,
    int *const output) {
  const int i = vecgeom_cuda::ThreadIndex();
  if (i >= n) return; // Out of range
  output[i] =
      navigator->LocatePoint(world, (*points)[i], paths[i], true)->id();
}
```

CUDA kernel compiled with NVCC

Johannes de Fine Licht (johannes.definelicht@cern.ch)

Appendix: Common code

```cpp
template<typename Backend>
VECGEOM_INLINE
VECGEOM_CUDA_HEADER_BOTH
void BoxUnplacedInside( Vector3D<Precision> const & dimensions,
                        Vector3D<typename Backend::precision_v> const &localpoint,
                        typename Backend::bool_v *const inside )
{
  Vector3D<typename Backend::bool_v> inside_dim(Backend::kFalse);
    for (int i = 0; i < 3; ++i) {
      inside_dim[i] = Abs(localpoint[i]) < dimensions[i];
      if (Backend::early_returns) {
        if (!inside_dim[i]) {
          *inside = Backend::kFalse;
          return;
        }
      }
    }

    if (Backend::early_returns) {
      *inside = Backend::kTrue;
    } else {
      *inside = inside_dim[0] && inside_dim[1] && inside_dim[2];
    }
}
```

Appendix: Template hierarchy

```
template <TranslationCode trans_code, RotationCode rot_code>
void SpecializedBox<trans_code, rot_code>::DistanceToIn(
    SOA3D<Precision> const &positions,
    SOA3D<Precision> const &directions,
    Precision const *const step_max,
    Precision *const output) const {
  DistanceToIn_Looper<trans_code, rot_code>(*this, positions, directions,
                                            step_max, output);
}
```

```
template <TranslationCode trans_code, RotationCode rot_code,
          typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
                                        ContainerType const &positions,
                                        ContainerType const &directions,
                                        Precision const *const step_max,
                                        Precision *const output) {
  for (int i = 0; i < positions.fillsize(); i += kVectorSize) {
    const VcPrecision result =
        volume.template DistanceToInDispatch<trans_code, rot_code, kVc>(
          Vector3D<VcPrecision>(VcPrecision(&positions.ContainerType::x(i)),
                                VcPrecision(&positions.ContainerType::y(i)),
                                VcPrecision(&positions.ContainerType::z(i))),
          Vector3D<VcPrecision>(VcPrecision(&directions.ContainerType::x(i)),
                                VcPrecision(&directions.ContainerType::y(i)),
                                VcPrecision(&directions.ContainerType::z(i))),
          VcPrecision(&step_max[i])
        );
    result.store(&output[i]);
  }
}
```

```
struct kVc {
  typedef Vc::int_v                    int_v;
  typedef Vc::Vector<Precision>        precision_v;
  typedef Vc::Vector<Precision>::Mask  bool_v;
  constexpr static bool early_returns = false;
  const static precision_v kOne;
  const static precision_v kZero;
  const static bool_v kTrue;
  const static bool_v kFalse;
};
```

```
template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
void BoxDistanceToIn(
    Vector3D<Precision> const &dimensions,
    TransformationMatrix const &matrix,
    Vector3D<typename Backend::precision_v> const &pos,
    Vector3D<typename Backend::precision_v> const &dir,
    typename Backend::precision_v const &step_max,
    typename Backend::precision_v *const distance) {

  typedef typename Backend::precision_v Float;
  typedef typename Backend::bool_v Bool;

  Vector3D<Float> safety;
  Vector3D<Float> pos_local;
  Vector3D<Float> dir_local;
  Bool hit(false);
  Bool done(false);
  *distance = kInfinity;

  matrix.Transform<trans_code, rot_code>(pos, pos_local);
  matrix.TransformRotation<rot_code>(dir, dir_local);
```

Johannes de Fine Licht (johannes.definelicht@cern.ch)

Vector interface

```cpp
template <TranslationCode trans_code, RotationCode rot_code>
void SpecializedBox<trans_code, rot_code>::DistanceToIn(
    SOA3D<Precision> const &positions,
    SOA3D<Precision> const &directions,
    Precision const *const step_max,
    Precision *const output) const {
  DistanceToIn_Looper<trans_code, rot_code>(*this, positions, directions,
                                            step_max, output);
}
```

```cpp
template <TranslationCode trans_code, RotationCode rot_code,
          typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
                                        ContainerType const &positions,
                                        ContainerType const &directions,
                                        Precision const *const step_max,
                                        Precision *const output) {
  for (int i = 0; i < positions.fillsize(); ++i) {
    output[i] =
        volume.template DistanceToInDispatch<trans_code, rot_code, kScalar>(
          positions[i], directions[i], step_max[i]
        );
  }
}
```

```cpp
template <TranslationCode trans_code, RotationCode rot_code,
          typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
                                        ContainerType const &positions,
                                        ContainerType const &directions,
                                        Precision const *const step_max,
                                        Precision *const output) {
  for (int i = 0; i < positions.fillsize(); i += kVectorSize) {
    const VcPrecision result =
        volume.template DistanceToInDispatch<trans_code, rot_code, kVc>(
          Vector3D<VcPrecision>(VcPrecision(&positions.ContainerType::x(i)),
                                VcPrecision(&positions.ContainerType::y(i)),
                                VcPrecision(&positions.ContainerType::z(i))),
          Vector3D<VcPrecision>(VcPrecision(&directions.ContainerType::x(i)),
                                VcPrecision(&directions.ContainerType::y(i)),
                                VcPrecision(&directions.ContainerType::z(i))),
          VcPrecision(&step_max[i])
        );
    result.store(&output[i]);
  }
}
```

```cpp
template <TranslationCode trans_code, RotationCode rot_code,
          typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
                                        ContainerType const &positions,
                                        ContainerType const &directions,
                                        Precision const *const step_max,
                                        Precision *const output) {
  for (int i = 0; i < positions.fillsize(); i += CilkPrecision::size()) {
    const CilkPrecision result =
        volume.template DistanceToInDispatch<trans_code, rot_code,
                                             kCilk>(
          Vector3D<CilkPrecision>(CilkPrecision(&positions.x(i)),
                                  CilkPrecision(&positions.y(i)),
                                  CilkPrecision(&positions.z(i))),
          Vector3D<CilkPrecision>(CilkPrecision(&directions.x(i)),
                                  CilkPrecision(&directions.y(i)),
                                  CilkPrecision(&directions.z(i))),
          CilkPrecision(&step_max[i])
        );
    result.store(&output[i]);
  }
}
```

```cpp
struct kScalar {
  typedef int       int_v;
  typedef Precision precision_v;
  typedef bool      bool_v;
  const static bool early_returns = true;
  const static precision_v kOne;
  const static precision_v kZero;
  const static bool_v kTrue = true;
  const static bool_v kFalse = false;
};
```

```cpp
struct kVc {
  typedef Vc::int_v                     int_v;
  typedef Vc::Vector<Precision>         precision_v;
  typedef Vc::Vector<Precision>::Mask bool_v;
  constexpr static bool early_returns = false;
  const static precision_v kOne;
  const static precision_v kZero;
  const static bool_v kTrue;
  const static bool_v kFalse;
};
```

```cpp
struct kCilk {
  typedef CilkVector<int>       int_v;
  typedef CilkVector<Precision> precision_v;
  typedef CilkVector<bool>      bool_v;
  constexpr static bool early_returns = false;
  const static precision_v kOne;
  const static precision_v kZero;
  const static bool_v kTrue;
  const static bool_v kFalse;
};
```

```cpp
template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
VECGEOM_CUDA_HEADER_BOTH
VECGEOM_INLINE
typename Backend::precision_v PlacedBox::DistanceToInDispatch(
    Vector3D<typename Backend::precision_v> const &position,
    Vector3D<typename Backend::precision_v> const &direction,
    const typename Backend::precision_v step_max) const {

  typename Backend::precision_v output;
  BoxDistanceToIn<trans_code, rot_code, Backend>(
    unplaced_box()->dimensions(),
    *this->matrix(),
    position,
    direction,
    step_max,
    &output
  );
  return output;
}
```

```cpp
struct kCuda {
  typedef int       int_v;
  typedef Precision precision_v;
  typedef bool      bool_v;
  const static bool early_returns = false;
  const static precision_v kOne = 1.0;
  const static precision_v kZero = 0.0;
  const static bool_v kTrue = true;
  const static bool_v kFalse = false;
};
```

External CUDA kernels

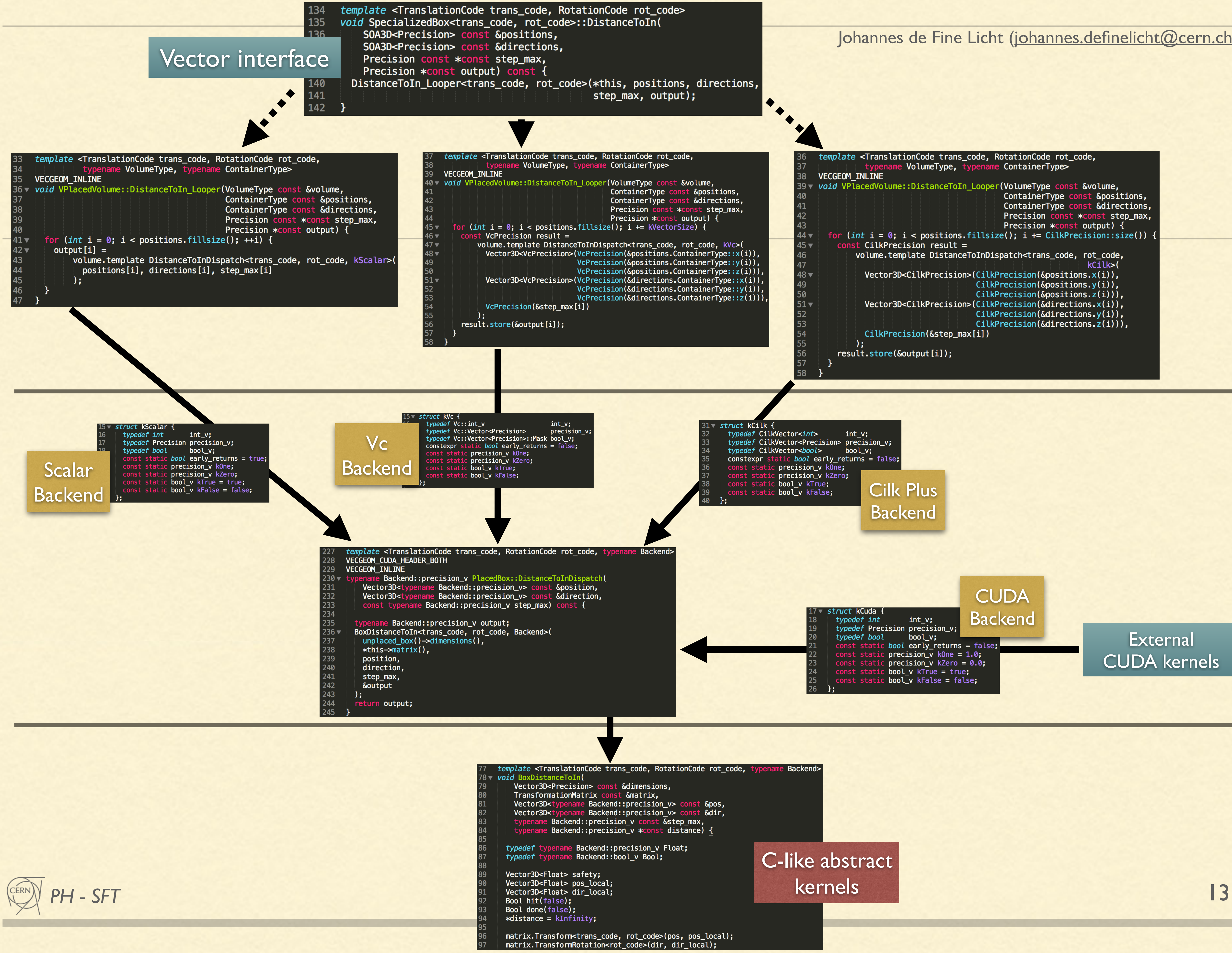
```cpp
template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
void BoxDistanceToIn(
    Vector3D<Precision> const &dimensions,
    TransformationMatrix const &matrix,
    Vector3D<typename Backend::precision_v> const &pos,
    Vector3D<typename Backend::precision_v> const &dir,
    typename Backend::precision_v const &step_max,
    typename Backend::precision_v *const distance) {

  typedef typename Backend::precision_v Float;
  typedef typename Backend::bool_v Bool;

  Vector3D<Float> safety;
  Vector3D<Float> pos_local;
  Vector3D<Float> dir_local;
  Bool hit(false);
  Bool done(false);
  *distance = kInfinity;

  matrix.Transform<trans_code, rot_code>(pos, pos_local);
  matrix.TransformRotation<rot_code>(dir, dir_local);
```

C-like abstract kernels


Vector interface
template <TranslationCode trans_code, RotationCode rot_code>
void SpecializedBox<trans_code, rot_code>::DistanceToIn(
SOA3D<Precision> const &positions,
SOA3D<Precision> const &directions,
Precision const *const step_max,
Precision *const output) const {
DistanceToIn_Looper<trans_code, rot_code>(*this, positions, directions,
step_max, output);
}
template <TranslationCode trans_code, RotationCode rot_code,
typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
ContainerType const &positions,
ContainerType const &directions,
Precision const *const step_max,
Precision *const output) {
for (int i = 0; i < positions.fillsize(); ++i) {
output[i] =
volume.template DistanceToInDispatch<trans_code, rot_code, kScalar>(
positions[i], directions[i], step_max[i]
);
}
}
template <TranslationCode trans_code, RotationCode rot_code,
typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
ContainerType const &positions,
ContainerType const &directions,
Precision const *const step_max,
Precision *const output) {
for (int i = 0; i < positions.fillsize(); i += kVectorSize) {
const VcPrecision result =
volume.template DistanceToInDispatch<trans_code, rot_code, kVc>(
Vector3D<VcPrecision>(VcPrecision(&positions.ContainerType::x(i)),
VcPrecision(&positions.ContainerType::y(i)),
VcPrecision(&positions.ContainerType::z(i))),
Vector3D<VcPrecision>(VcPrecision(&directions.ContainerType::x(i)),
VcPrecision(&directions.ContainerType::y(i)),
VcPrecision(&directions.ContainerType::z(i))),
VcPrecision(&step_max[i])
);
result.store(&output[i]);
}
}
template <TranslationCode trans_code, RotationCode rot_code,
typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
ContainerType const &positions,
ContainerType const &directions,
Precision const *const step_max,
Precision *const output) {
for (int i = 0; i < positions.fillsize(); i += CilkPrecision::size()) {
const CilkPrecision result =
volume.template DistanceToInDispatch<trans_code, rot_code,
kCilk>(
Vector3D<CilkPrecision>(CilkPrecision(&positions.x(i)),
CilkPrecision(&positions.y(i)),
CilkPrecision(&positions.z(i))),
Vector3D<CilkPrecision>(CilkPrecision(&directions.x(i)),
CilkPrecision(&directions.y(i)),
CilkPrecision(&directions.z(i))),
CilkPrecision(&step_max[i])
);
result.store(&output[i]);
}
}
Scalar Backend
struct kScalar {
typedef int int_v;
typedef Precision precision_v;
typedef bool bool_v;
const static bool early_returns = true;
const static precision_v kOne;
const static precision_v kZero;
const static bool_v kTrue = true;
const static bool_v kFalse = false;
};
Vc Backend
struct kVc {
typedef Vc::int_v int_v;
typedef Vc::Vector<Precision> precision_v;
typedef Vc::Vector<Precision>::Mask bool_v;
constexpr static bool early_returns = false;
const static precision_v kOne;
const static precision_v kZero;
const static bool_v kTrue;
const static bool_v kFalse;
};
Cilk Plus Backend
struct kCilk {
typedef CilkVector<int> int_v;
typedef CilkVector<Precision> precision_v;
typedef CilkVector<bool> bool_v;
constexpr static bool early_returns = false;
const static precision_v kOne;
const static precision_v kZero;
const static bool_v kTrue;
const static bool_v kFalse;
};
template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
VECGEOM_CUDA_HEADER_BOTH
VECGEOM_INLINE
typename Backend::precision_v PlacedBox::DistanceToInDispatch(
Vector3D<typename Backend::precision_v> const &position,
Vector3D<typename Backend::precision_v> const &direction,
const typename Backend::precision_v step_max) const {
typename Backend::precision_v output;
BoxDistanceToIn<trans_code, rot_code, Backend>(
unplaced_box()->dimensions(),
*this->matrix(),
position,
direction,
step_max,
&output
);
return output;
}
CUDA Backend
struct kCuda {
typedef int int_v;
typedef Precision precision_v;
typedef bool bool_v;
const static bool early_returns = false;
const static precision_v kOne = 1.0;
const static precision_v kZero = 0.0;
const static bool_v kTrue = true;
const static bool_v kFalse = false;
};
External CUDA kernels
template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
void BoxDistanceToIn(
Vector3D<Precision> const &dimensions,
TransformationMatrix const &matrix,
Vector3D<typename Backend::precision_v> const &pos,
Vector3D<typename Backend::precision_v> const &dir,
typename Backend::precision_v const &step_max,
typename Backend::precision_v *const distance) {
typedef typename Backend::precision_v Float;
typedef typename Backend::bool_v Bool;
Vector3D<Float> safety;
Vector3D<Float> pos_local;
Vector3D<Float> dir_local;
Bool hit(false);
Bool done(false);
*distance = kInfinity;
matrix.Transform<trans_code, rot_code>(pos, pos_local);
matrix.TransformRotation<rot_code>(dir, dir_local);
C-like abstract kernels