HPC COLLABORATIVE SOURCE & STANDARDS AT NVIDIA

Bryce Adelstein Lelbach @blelbach

Principal Architect

NVIDIA

Why Is NVIDIA Here?

Why Do HPC Users Want Collaborative Source & Standards From Vendors?

- Allow modification to suit their needs.
- Facilitate collaboration and innovation.
- Ensure portability.
- Guarantee long-term support.
- Enable result reproduction.

Open Source & Standards (Non-Collaborative)

- ☑ Open license.
- ☒ No contributions.
- ☒ Private planning.
- ☒ Private development.
- ☒ Private testing.
- ☒ Private issue tracking.

Collaborative Source & Standards

- ☑ Open license.
- ☑ Open contributions.
- ☑ Open planning.
- ☑ Open development.
- ☑ Open testing.
- ☑ Open issue tracking.

Open Source & Standards (Non-Collaborative)	Collaborative Source & Standards
☑ Open license.	☑ Open license.
☒ No contributions.	☑ Open contributions.
☒ Private planning.	☑ Open planning.
☒ Private development.	☑ Open development.
☒ Private testing.	☑ Open testing.
☒ Private issue tracking.	☑ Open issue tracking.

I have a helpful PR!

Sorry, it failed our CI.

Can I see the failure logs so I can fix it?

Nope, our CI is internal & proprietary.

Then how do I fix it?

How Do Vendors Benefit From Collaborative Source & Standards?

- Greater adoption of our products.
- Insight into how our product is used.
- Insight into who uses our product.
- Enable result reproduction.

Collaborative Source & Standards

☑Open license.

☑Open contributions.

☑Open planning.

☑Open development.

☑Open testing.

☑Open issue tracking.

Projects needs greater engineering investment to support collaboration.

Contributing

- Existing projects/communities.
- Build trust.
- Learn process.
- Provide feedback.
- Show respect.

Leading

- New projects/communities.
- Build community.
- Establish process.
- Listen to feedback.
- Share control.

The C++ parallel algorithms library

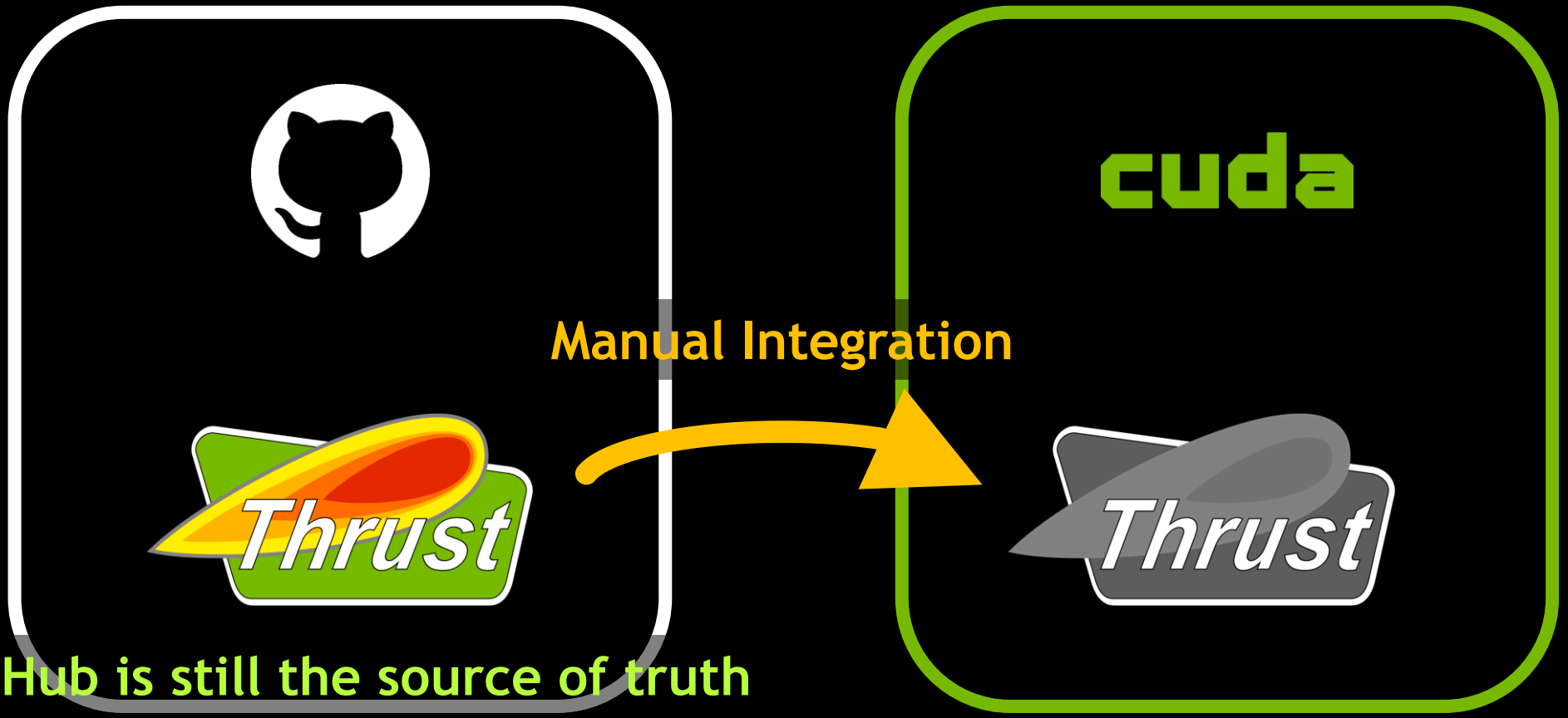
https://github.com/NVIDIA/thrust

```
void
normalize(thrust::universal_vector<float>& v)
{
 auto m = thrust::max_element(
  v.begin(), v.end());

 auto mit = thrust::make_constant_iterator(*m);

 thrust::transform(v.begin(), v.end(),
                   mit, v.begin(),
                   thrust::divides{});
}
```

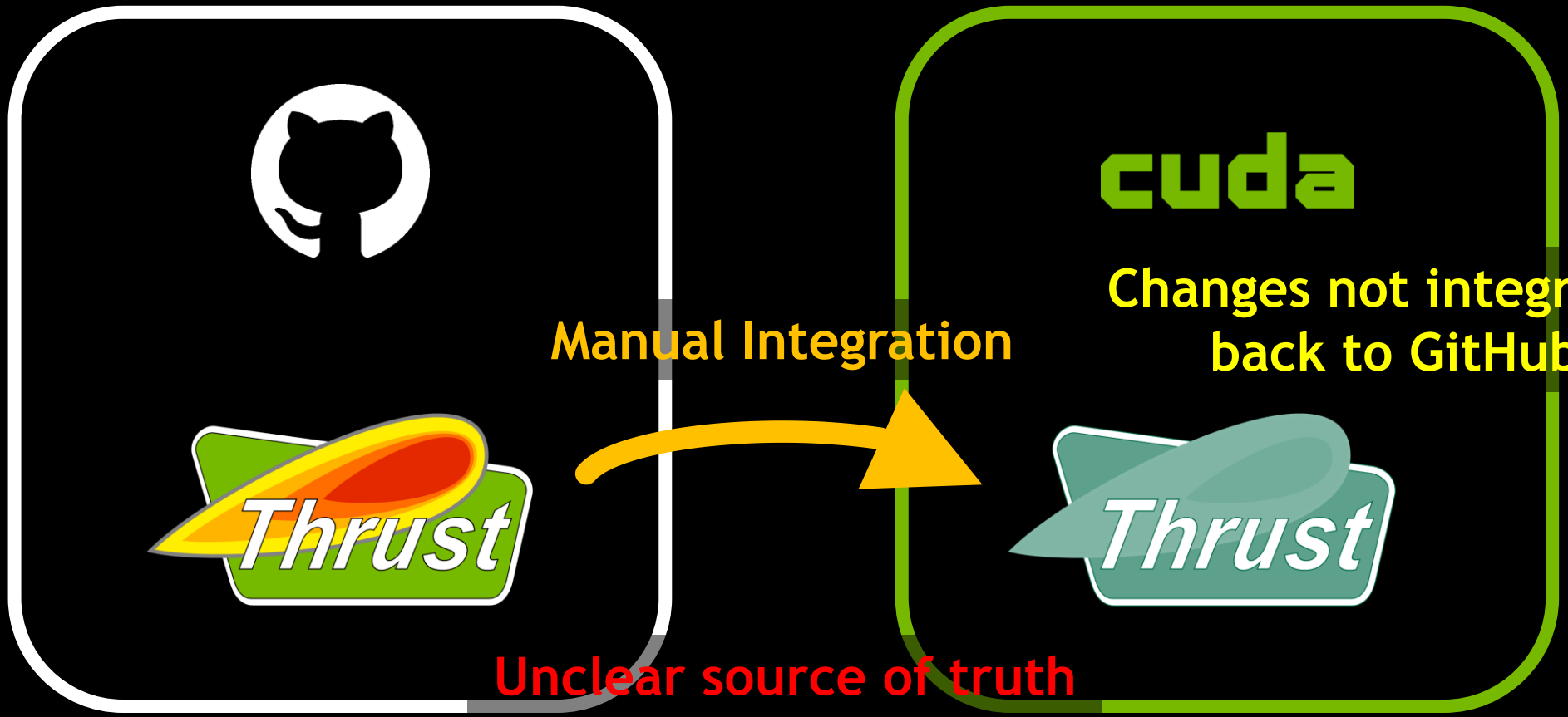
- Inspired the Standard C++ parallel algorithms.
- One of NVIDIA's oldest collaborative source projects.
- LLVM/BSD/Boost licensed.
- Supports both CPU and GPU backends.

2009 to 2011
GitHub Source of Truth

2011
Added to the CUDA Toolkit

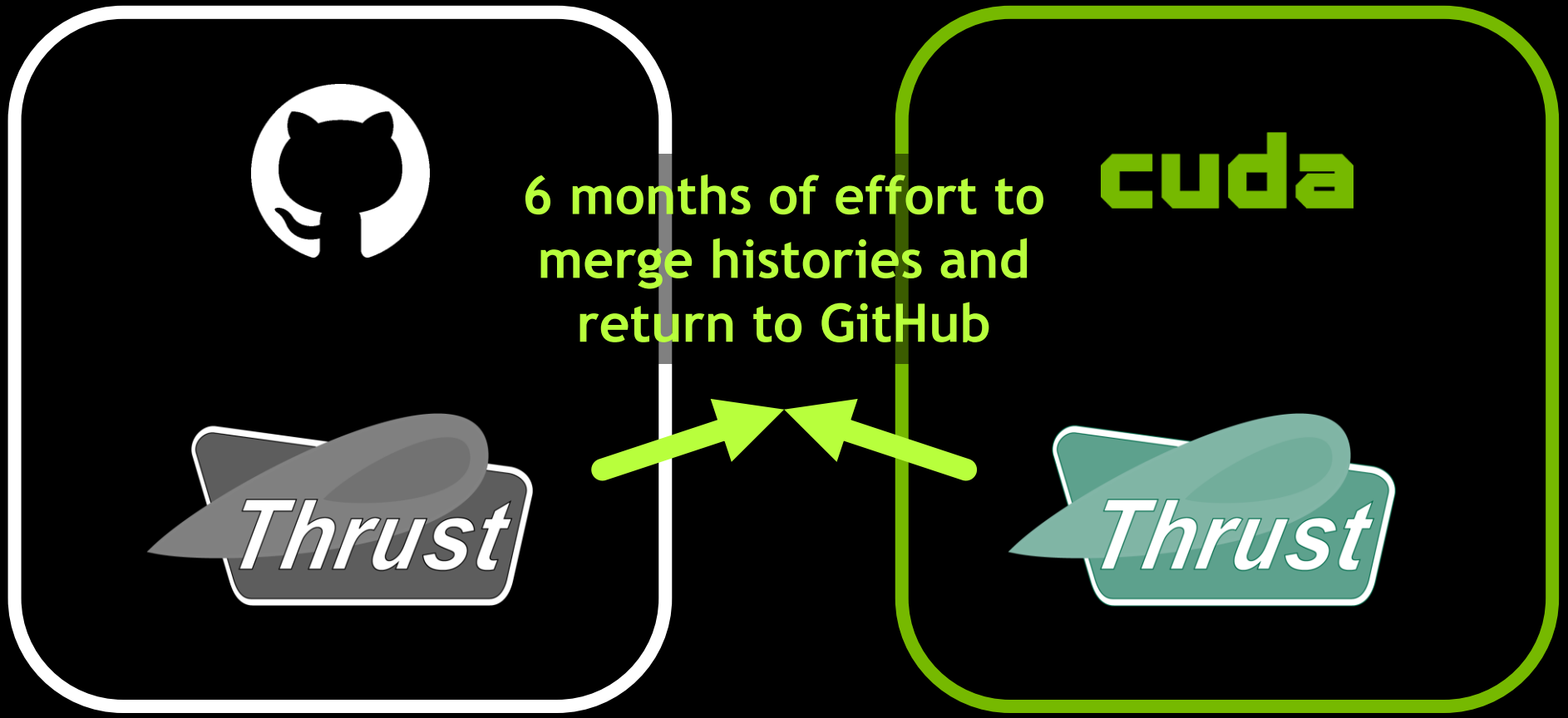


2011 to 2015
GitHub/CUDA Toolkit Divergence

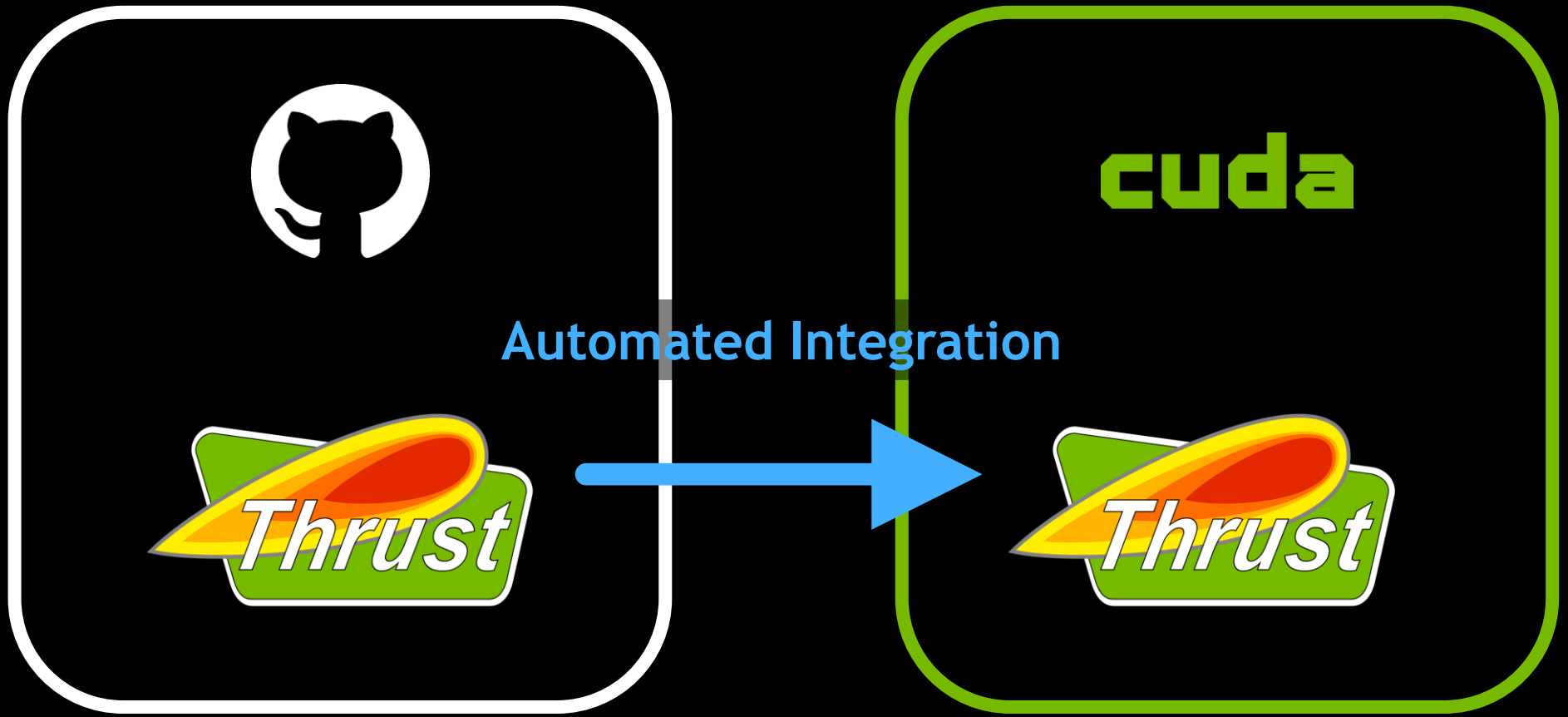


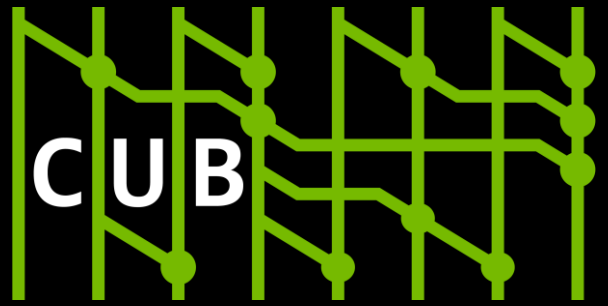
2015 to 2017
CUDA Toolkit Source of Truth

2017
Reunification



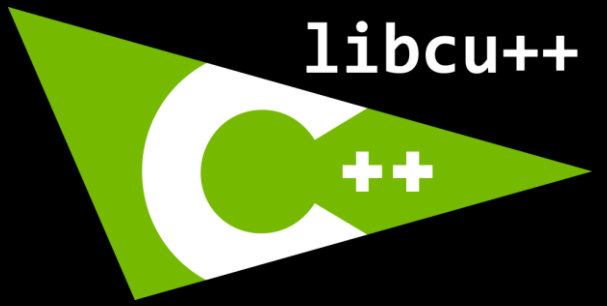
2017 to Present
GitHub Source of Truth, CUDA Toolkit Downstream





Cooperative primitives for CUDA C++

https://github.com/NVIDIA/cub



The NVIDIA C++ Standard Library

https://github.com/NVIDIA/libcudacxx

```
template <typename R>
__global__ void f(R, …) {
  __shared__ typename R::TempStorage tmp;
  int local_s(…);
  int s = R(tmp).Sum(local_s);
}

f<<<32, 1>>>(cub::WarpReduce<int>{}, …);
f<<<128, 1>>>(cub::BlockReduce<int, 128>{}, …);
```

Host's Standard Library

```
#include <…>
std::
```

```
#include <cuda/std/…>
cuda::std::

#include <cuda/…>
cuda::
```

libcu++

The C++ parallel algorithms library

https://github.com/NVIDIA/thrust

```
thrust::host_vector<float> h;
thrust::device_vector<float> d;

thrust::event e = thrust::async::copy(
  par, h.begin(), h.end(), d.begin());
thrust::future<float> f = thrust::async::reduce(
  par.after(e), d.begin(), d.end());

…

float r = f.get();
```

```
template <typename R>
__global__ void f(R, …) {
  __shared__ typename R::TempStorage tmp;
  int local_s(…);
  int s = R(tmp).Sum(local_s);
}

f<<<32, 1>>>(cub::WarpReduce<int>{}, …);
f<<<128, 1>>>(cub::BlockReduce<int, 128>{}, …);
```

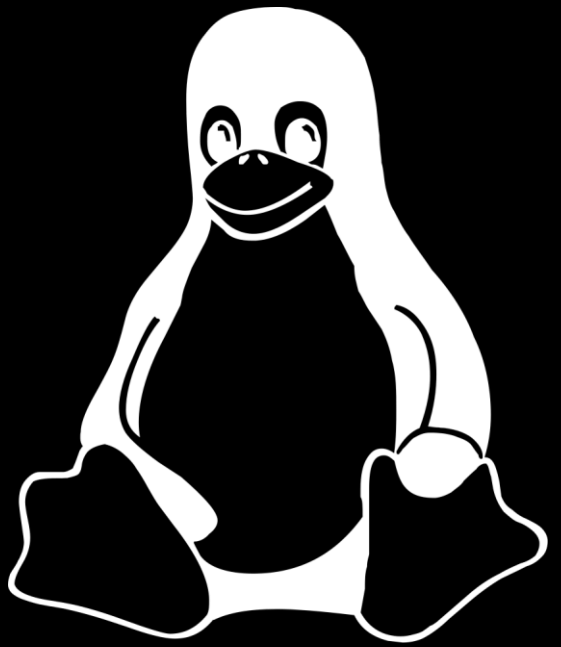
Flang

+

- Modern Fortran compiler for LLVM.
  - Official subproject of LLVM in the LLVM GitHub repo.
  - MLIR-based.
- Features:
  - Fortran 2023 support.
  - CPU & GPU Fortran Standard Parallelism (do concurrent loops, array intrinsics).
  - CPU & GPU OpenACC and OpenMP support.
  - CUDA Fortran Support.

Linux Kernel Drivers

- Released GPL drivers in 2022.
- Improves debugging, integration, & distribution.
- Sorry for the long wait!

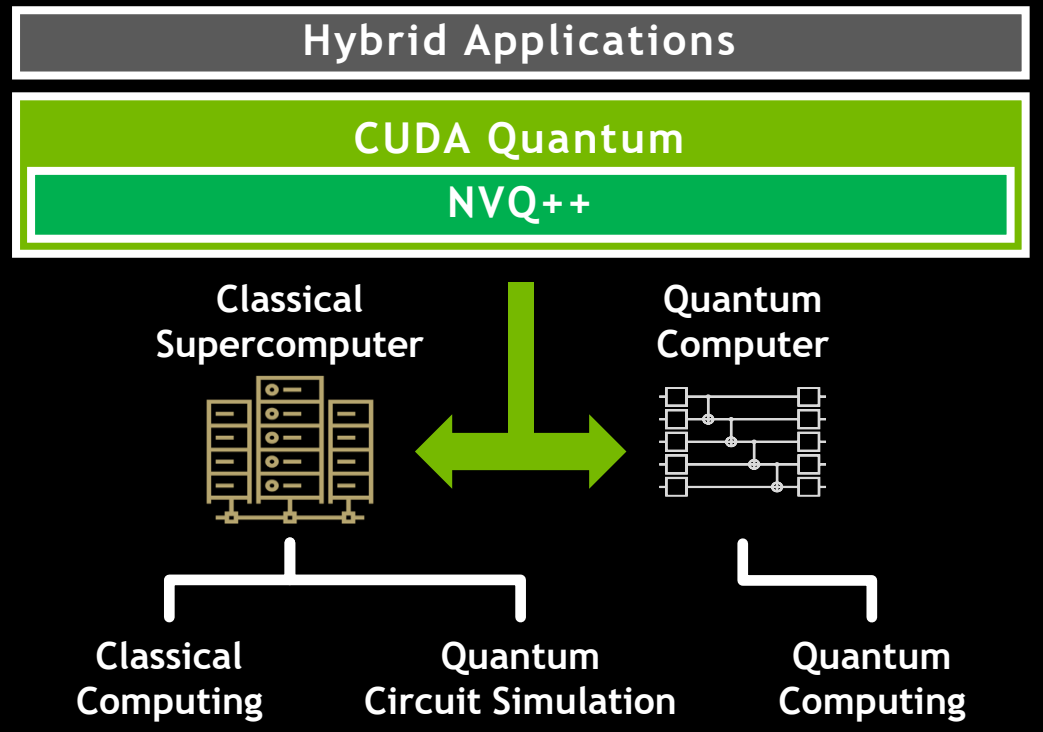



```
View<double**, LayoutRight> A = …;
View<double**, LayoutRight> B = …;

auto const policy =
  MDRangePolicy<Rank<2>>({0, 0}, {N, N});

parallel_for(policy,
  KOKKOS_LAMBDA(int i, int j) {
    B(i, j) += A(j, i);
  });

fence();
```

- HPC performance portability framework for C++.
- NVIDIA contributes to upstream.
- We collaborate together on C++ standardization.
- We've utilized components in our products (such as mdspan).

CUDA Quantum

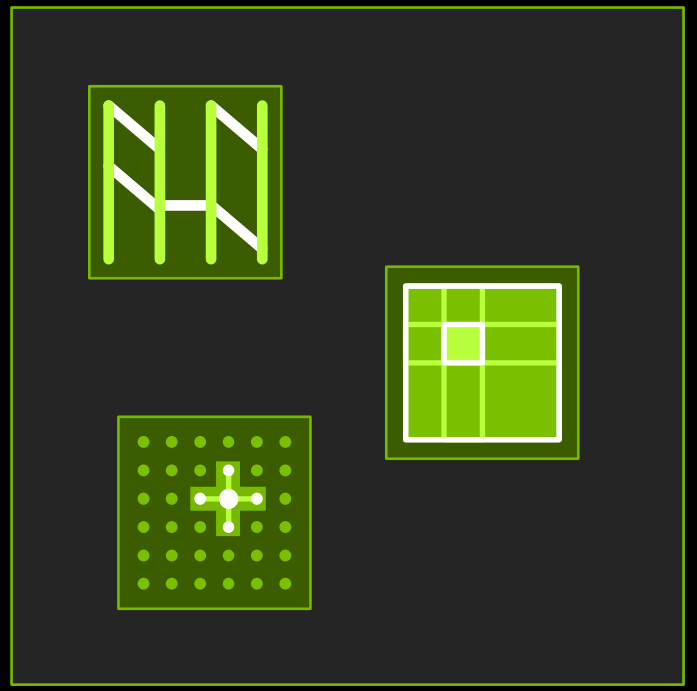
- Hybrid quantum-classical programming framework for C++ and Python.
- Collaborative source and specification.
  - Developed on GitHub.
  - Compiler based on LLVM and MLIR.
  - Uses and contributes to QIR.



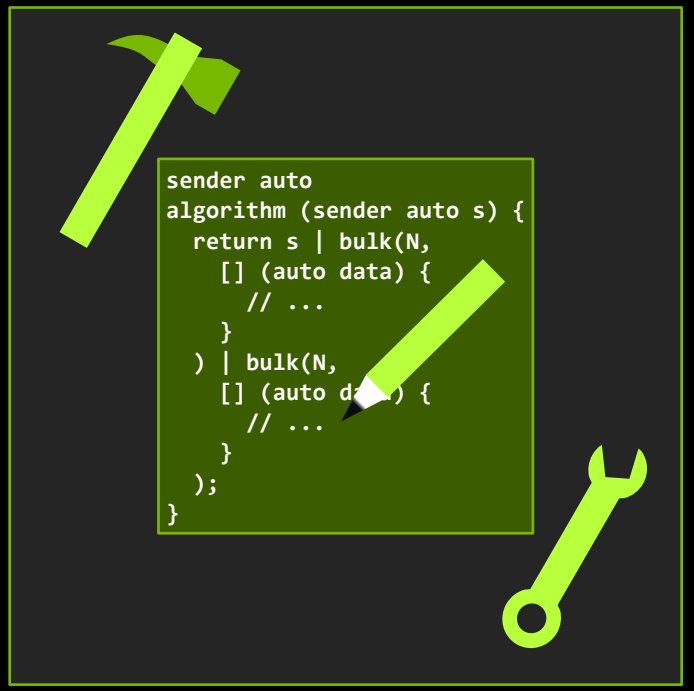

```
auto ghz = [] (int N) __qpu__ {
  qoda::qreg q(N);
  h(q[0]);
  for (auto i : stdv::iota(1, N-1))
    cnot(q[i], q[i + 1]);
  mz(q);
};
auto cnts = qoda::sample(ghz, 30);
```

ISO Standard Parallelism

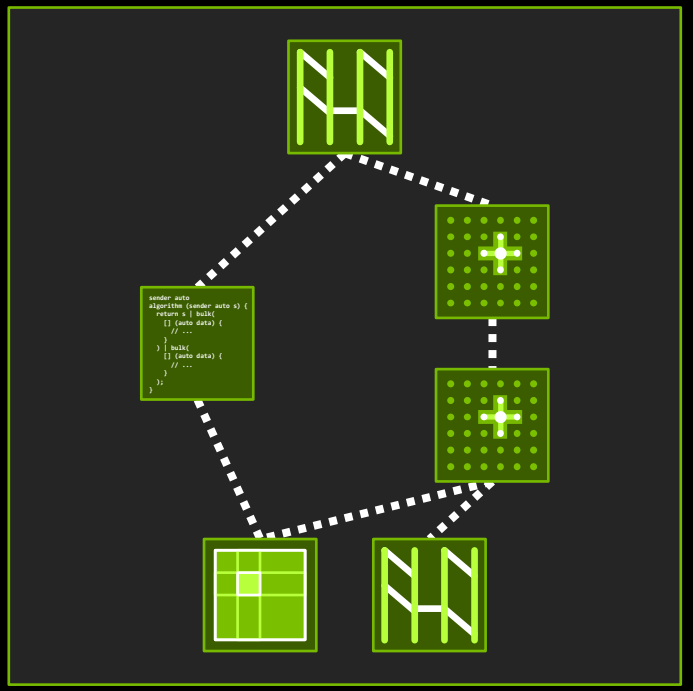
Common Algorithms that Dispatch to Vendor-Optimized Parallel Libraries



Tools to Write Your Own Parallel Algorithms that Run Anywhere

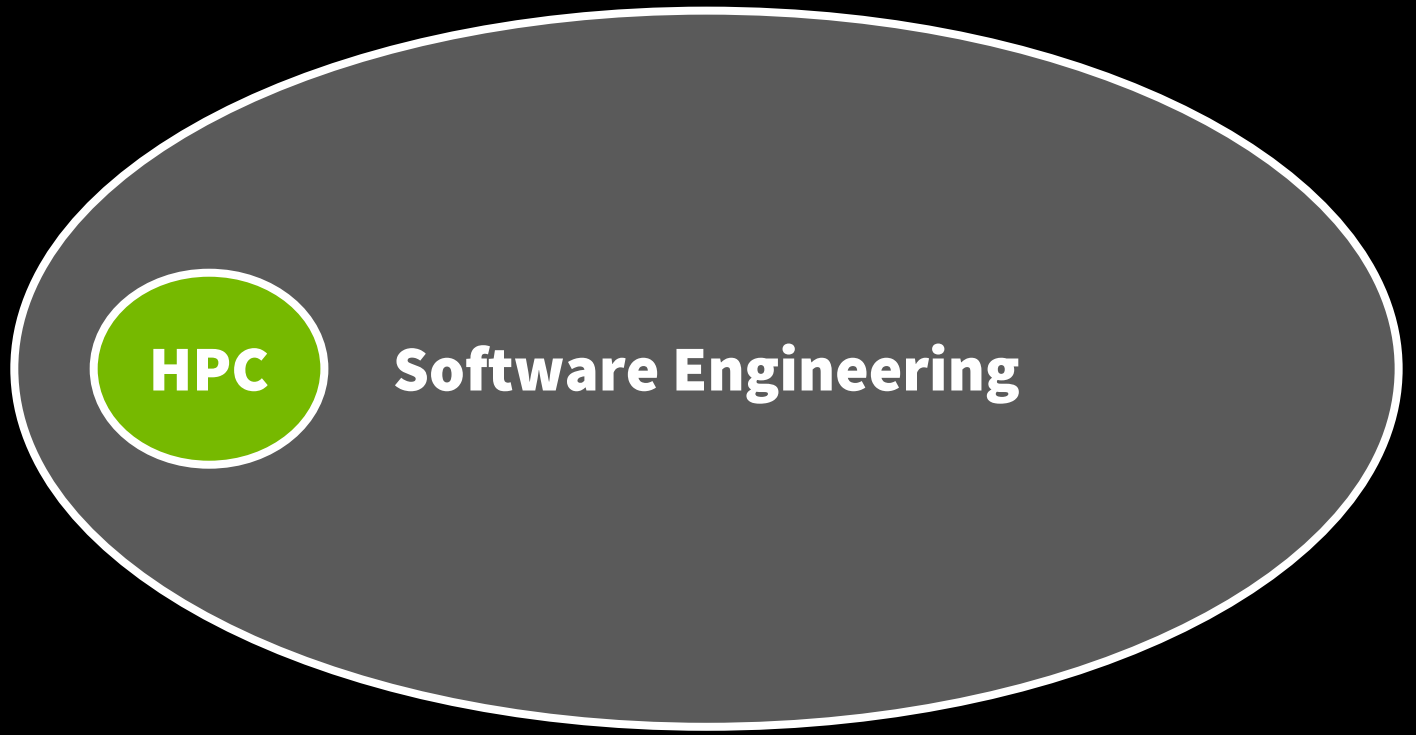


Mechanisms for Composing Parallel Invocations into Task Graphs

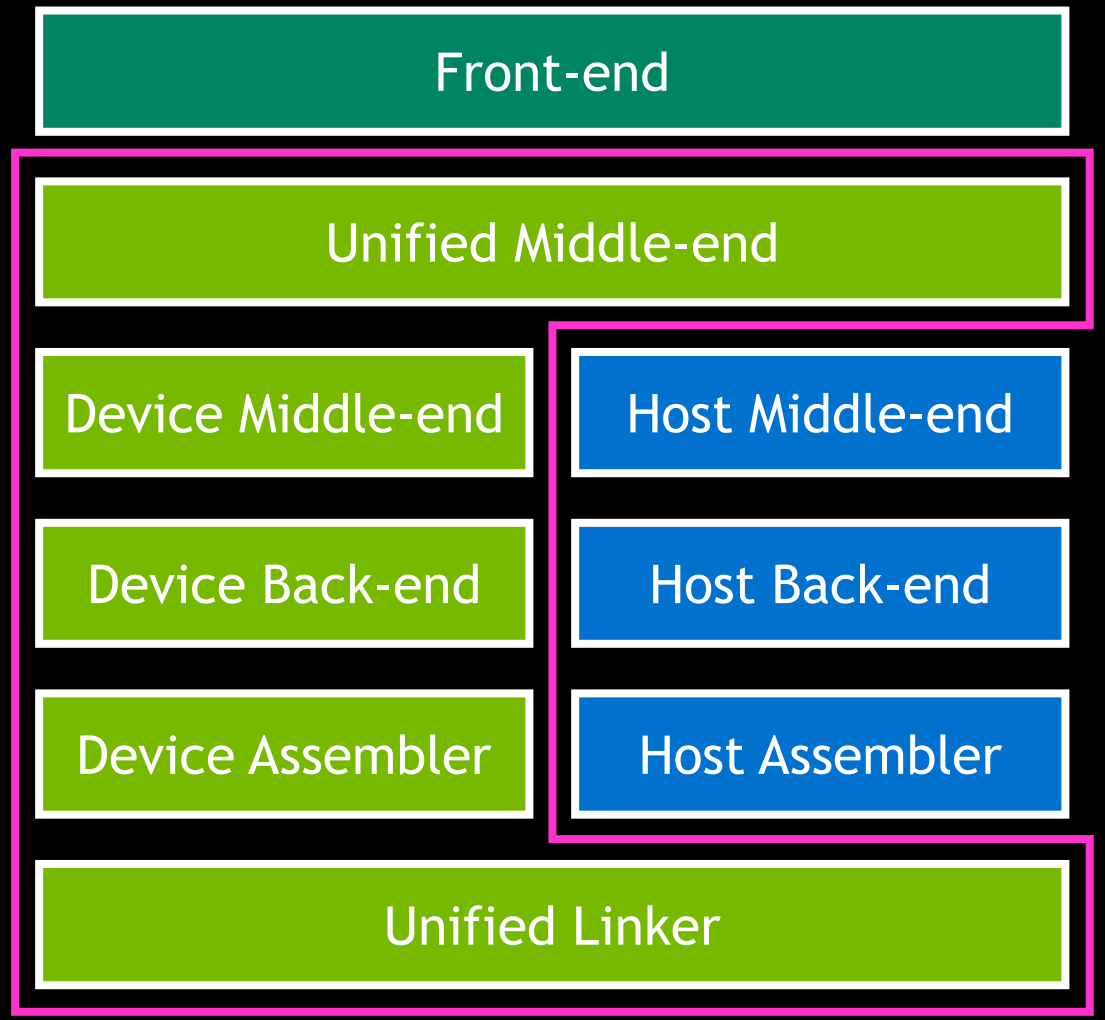


NVIDIA.

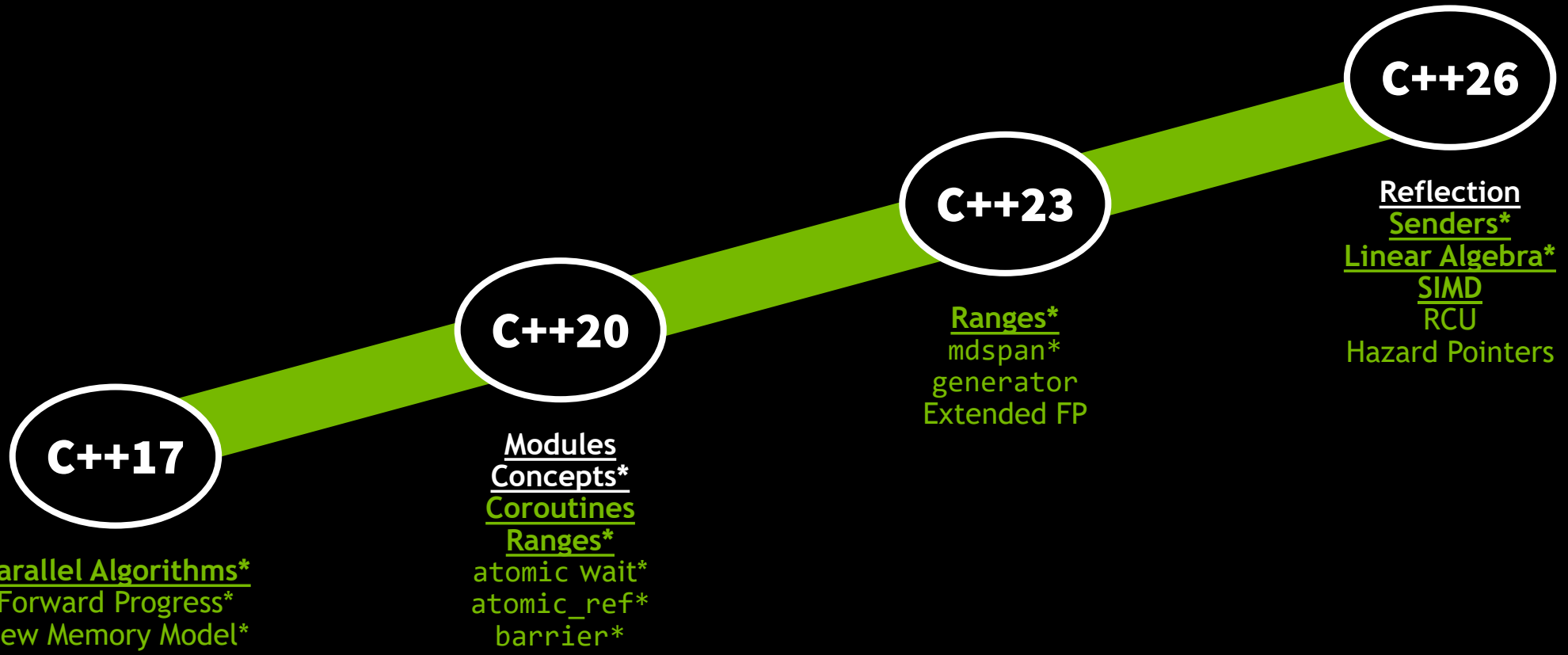
Why ISO ?



Because <u>everyone</u> implements and sustains them.


Front-end
Unified Middle-end
Device Middle-end
Host Middle-end
Device Back-end
Host Back-end
Device Assembler
Host Assembler
Unified Linker


GCC



* = Available now in the NVIDIA HPC SDK

NVIDIA.

Standard Parallel Algorithms

```
std::vector<double> x(…), y(…);
double dot_product =
  std::transform_reduce(std::execution::par,
                        x.begin(), x.end(), y.begin());
```

`nvc++ -stdpar=multicore`

`nvc++ -stdpar=gpu`

Since C++17 and the initial release of NVC++!

Standard Parallel Algorithms & Ranges

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};

auto v = stdv::cartesian_product(
  stdv::iota(0, A.extent(0)),
  stdv::iota(0, A.extent(1)));

std::for_each(ex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j] = idx;
    B[j, i] = A[i, j];
  });
```

In C++23 and NVC++ 22.5!

Standard Senders

```
std::mdspan A{input, N, N}, B{output, N, N};

auto v = stdv::cartesian_product(
  stdv::iota(0, A.extent(0)), stdv::iota(0, A.extent(1)));
```

Synchronous

```
std::for_each(stdex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j] = idx;
    B[j, i] = A[i, j];
  });

std::matrix_product(
  stdex::par_unseq, A, B, B);
```

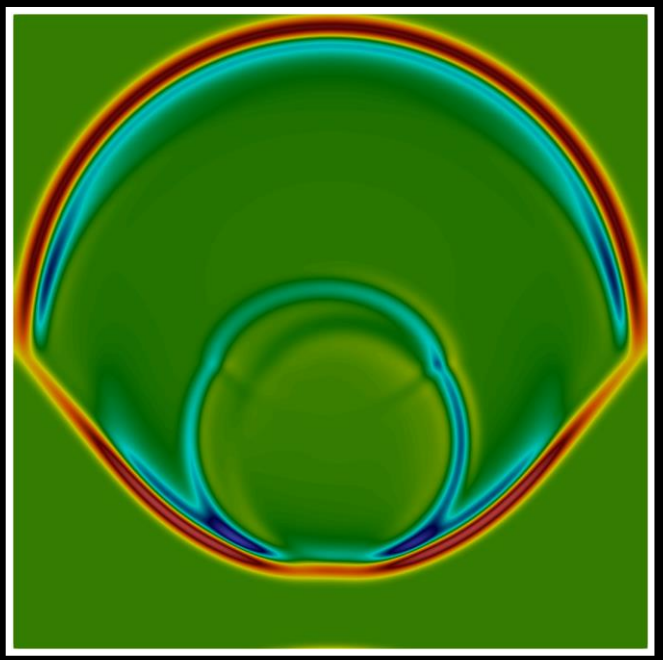
Asynchronous

```
auto s = stdex::just_on(sch, v)
       | stdex::bulk(N,
           [=] (auto idx) {
             auto [i, j] = idx;
             B[j, i] = A[i, j];
           })
       |
stdex::matrix_product_async(B, B);
```

Planned for C++26 and available at github.com/nvidia/stdexec!

Standard Senders

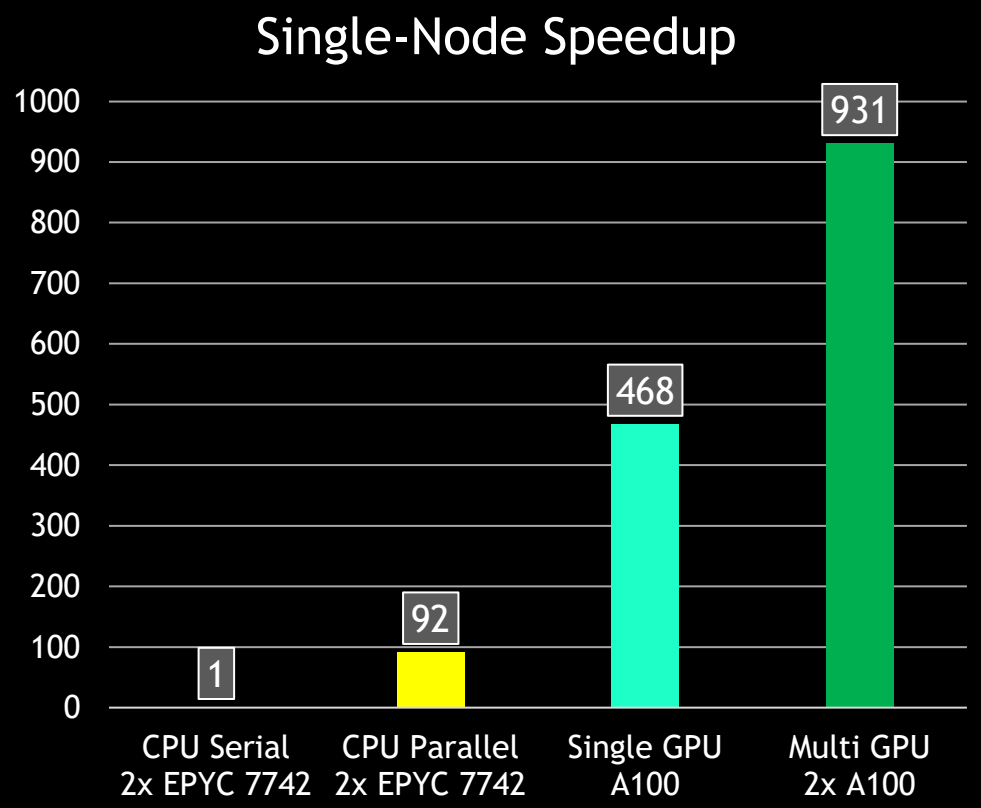
```
sender auto maxwell_eqs(scheduler auto &compute,
                        grid_accessor A, …) {
  return repeat_n(n_outer_iterations,
           repeat_n(n_inner_iterations,
               schedule(compute)
             | bulk(G.cells, update_h(G))
             | halo_exchange(G, hx, hy)
             | bulk(G.cells, update_e(time, dt, G))
             | halo_exchange(G, hx, hy))
           | transfer(cpu_serial_scheduler)
           | then(output_results))
         );
}
```

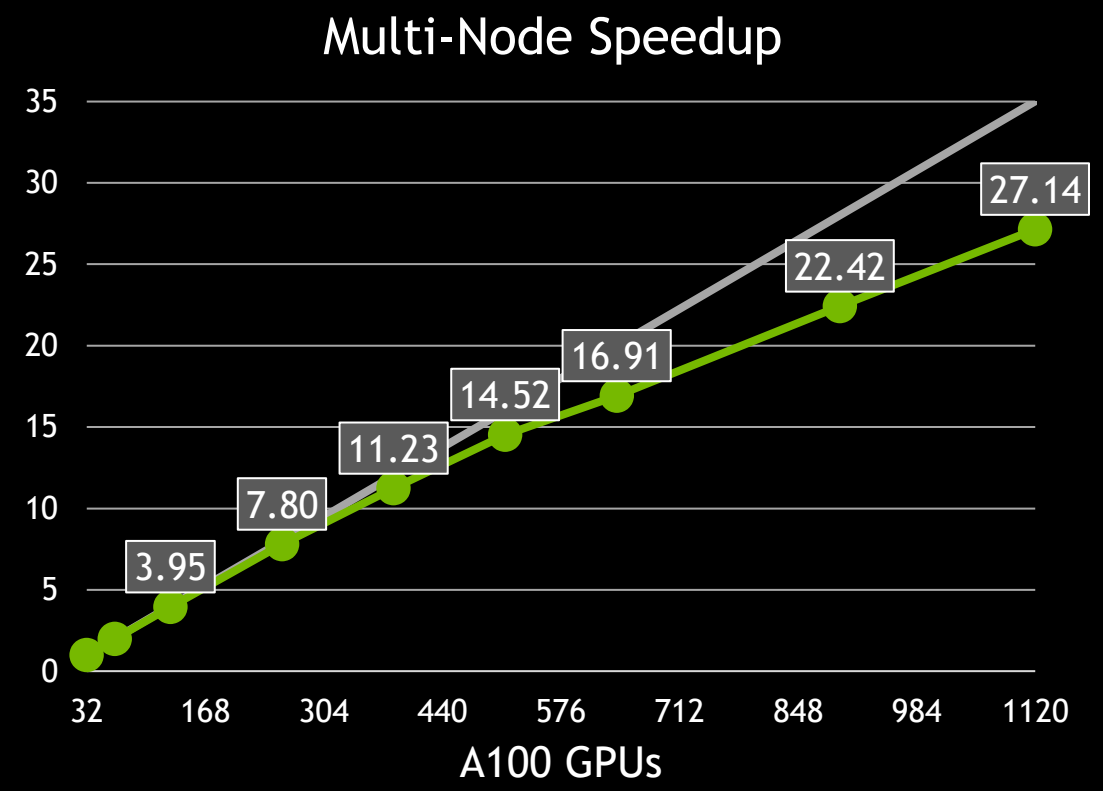


Planned for C++26 and available at github.com/nvidia/stdexec!

Standard Senders

Change one line of code and scale from a single CPU thread up to a cluster of GPUs!





Planned for C++26 and available at github.com/nvidia/stdexec!

NVIDIA

Standard Multidimensional Spans

```
template <class T, class Extents, class LayoutPolicy = …, class AccessorPolicy = …>
class std::mdspan;
```

```
mdspan A{data, N, M};
mdspan A{data, layout_right::mapping{N, M}};

A[i, j]     == data[i * M + j]
A.stride(0) == M
A.stride(1) == 1
```

Location	Element
0	a_{11}
1	a_{12}
2	a_{21}
3	a_{22}

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

```
mdspan B{data, layout_left::mapping{N, M}};

B[i, j]     == data[i + j * N]
B.stride(0) == 1
B.stride(1) == N
```

Location	Element
0	a_{11}
1	a_{21}
2	a_{12}
3	a_{22}

In C++23 and NVC++ 22.7!

Standard Linear Algebra

```
std::mdspan A{…, N, M};
std::mdspan x{…, M};
std::mdspan y{…, N};

// y = 3.0 A x + 2.0 y
std::matrix_vector_product(
  ex::par_unseq,
  std::scaled(3.0, A), x,
  std::scaled(2.0, y), y);
```

On the C++26 roadmap and in NVC++ 22.7!

New C++ Standard Floating Point Types

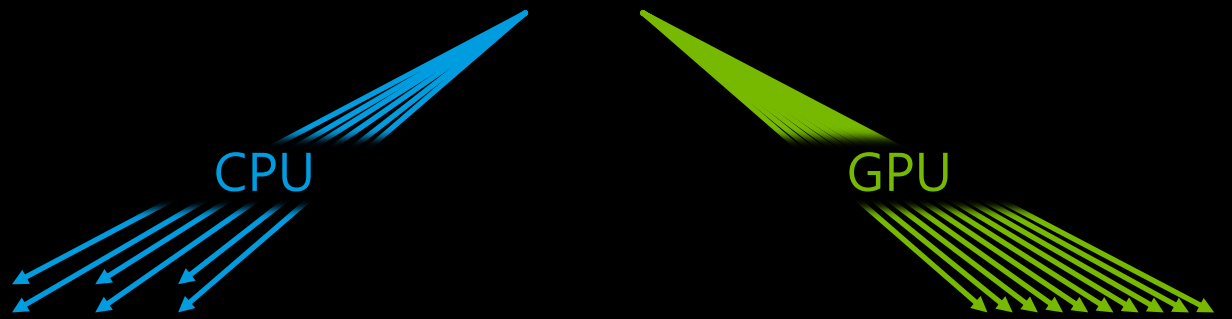
```
std::float16_t  // IEEE-754-2008 binary16.

std::float32_t  // IEEE-754-2008 binary32.

std::float64_t  // IEEE-754-2008 binary64.

std::float128_t // IEEE-754-2008 binary128.

std::bfloat16_t // binary32 with 16 bits truncated.
```

On the C++23 and NVHPC SDK roadmap!

Feature	C++ Release	Product Release
Parallel Algorithms	C++17	Since release
Ranges	C++20, C++23	HPC SDK 22.5
Ranges ⇔ Parallel Algorithms	C++23	HPC SDK 22.5
Senders	C++26	GitHub
`mdspan`	C++23	HPC SDK 22.7
Linear Algebra	C++26	HPC SDK 22.7
Extended Floating Point	C++23	Soon
`atomic`	C++11	libcu++ 1.0.0
`atomic_ref`	C++20	libcu++ 1.7.0
Scalable Synchronization Primitives	C++20	libcu++ 1.1.0

Standard do concurrent

```
do concurrent (j = 1 : N, i = 1 : N)
  B(j, i) = A(i, j)
end do
```




```
nvfortran -stdpar=multicore
```

```
nvfortran -stdpar=gpu
```

Since Fortran 2008 and NVFORTRAN 20.11!

Standard do concurrent reduce

```
s = 0
do concurrent (i = 1 : size(a)) reduce(+ : s)
  s = s + (2 * a(i) - 1)
end do

low = a(1)
do concurrent (i = 2 : size(a)) reduce(min : low)
  low = min(low, a(i))
end do
```

Supported Operators: `+ * .and. .or. .eqv. .neqv.`
Supported Procedures: `iand ior ieor min max`

Since Fortran 2023 and NVFORTRAN 20.11!

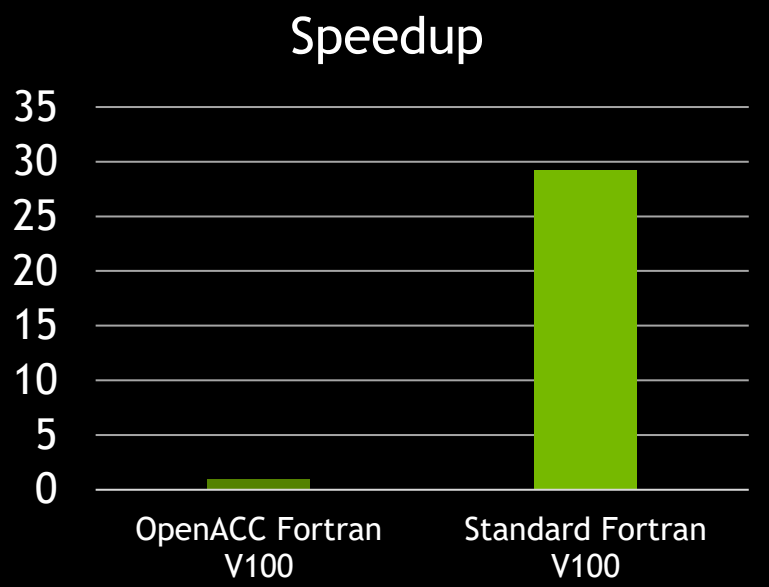
Standard do concurrent reduce

```
real(8), dimension(ni, nk) :: a
real(8), dimension(nk, nj) :: b
real(8), dimension(ni, nj) :: c

...

!$acc enter data copyin(a, b, c) create(d)

do nt = 1, ntimes
 !$acc kernels
 do j = 1, nj
  do i = 1, ni
   d(i, j) = c(i, j)
   do k = 1, nk
    d(i, j) = d(i, j) + a(i, k) * b(k, j)
   end do
  end do
 end do
 !$acc end kernels
end do

!$acc exit data copyout(d)
```

OpenACC Fortran

```
real(8), dimension(ni, nk) :: a
real(8), dimension(nk, nj) :: b
real(8), dimension(ni, nj) :: c

...

do nt = 1, ntimes
 d = c + matmul(a, b)
end do
```

Standard Fortran



Since Fortran 202x and NVFORTRAN XX.YY!

NVIDIA

Questions?

HPC COLLABORATIVE SOURCE & STANDARDS AT NVIDIA

Bryce Adelstein Lelbach @blelbach

Principal Architect

NVIDIA