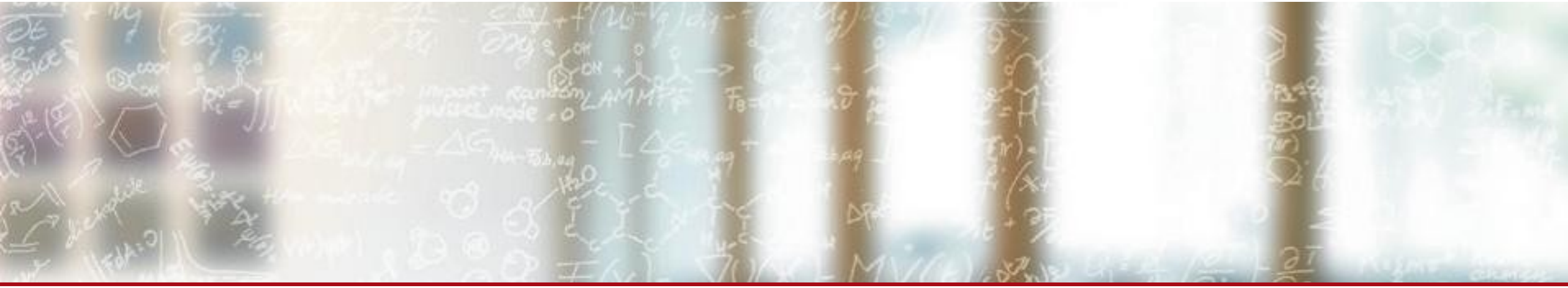
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



std::execution and pika at CSCS

CERN
Mikael Simberg, CSCS
October 5th, 2022

Motivation

- Piz Daint at CSCS
  - One of the biggest GPU supercomputers when installed (first in 2013, upgraded in 2017)
  - **Massive parallelism** and **heterogeneity** here to stay
- How to program?
  - CUDA? ×
  - OpenACC? ×
  - OpenMP? ×
  - SYCL? ×
  - Kokkos? ×
  - Standard C++? maybe?

P2300 std::execution

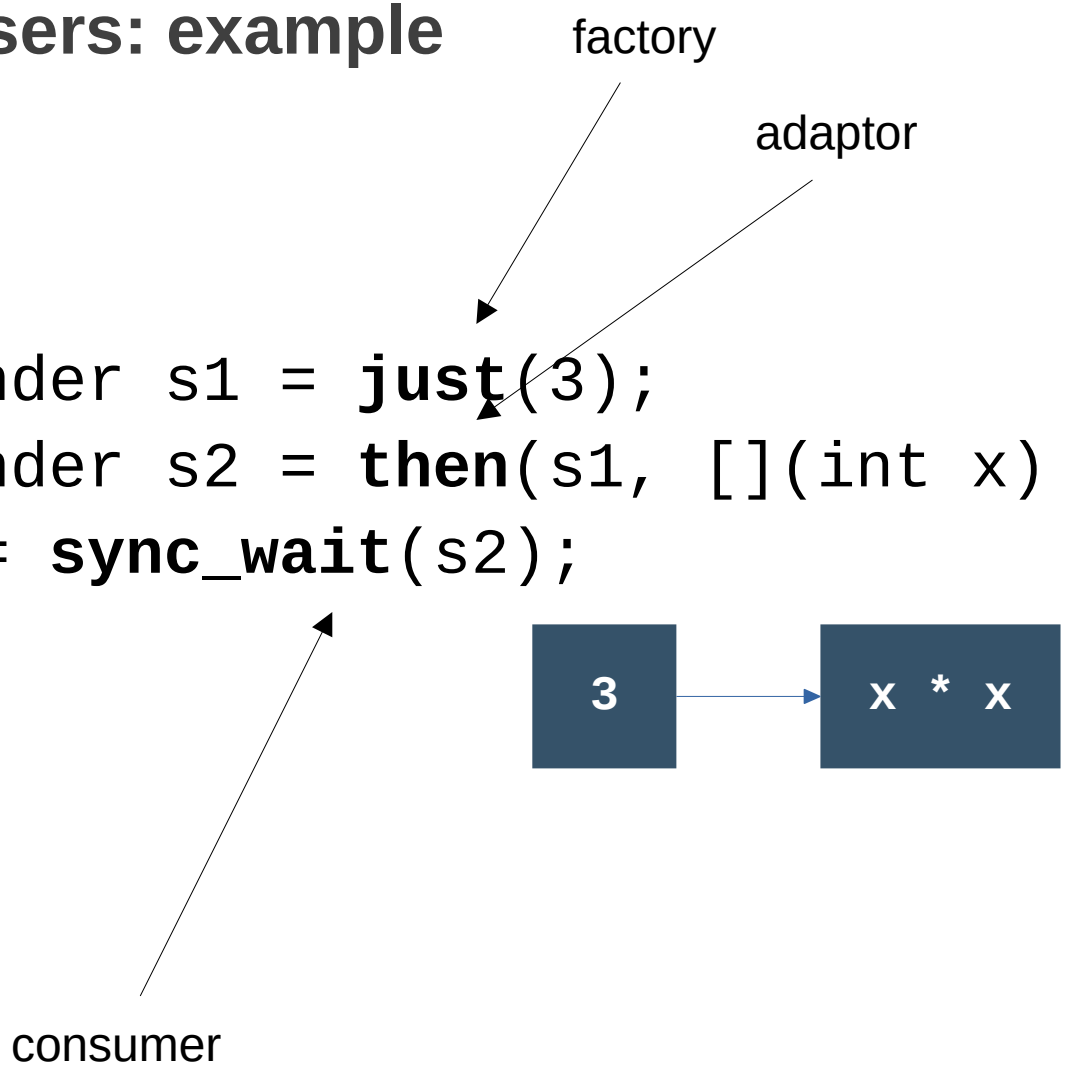
P2300: std::execution

- Proposes a generic library for dealing with
  - Execution on various **execution contexts**: CPU, GPU, etc.
  - **Transitions** between execution contexts
  - **Algorithms** for performing work on different execution contexts (cf. C++ ranges)
  - **Concepts**, not implementations (maybe a system thread pool as a separate proposal)
- Stand-alone proposal
  - Builds upon years of previous work (e.g. P0443)
- https://wg21.link/p2300

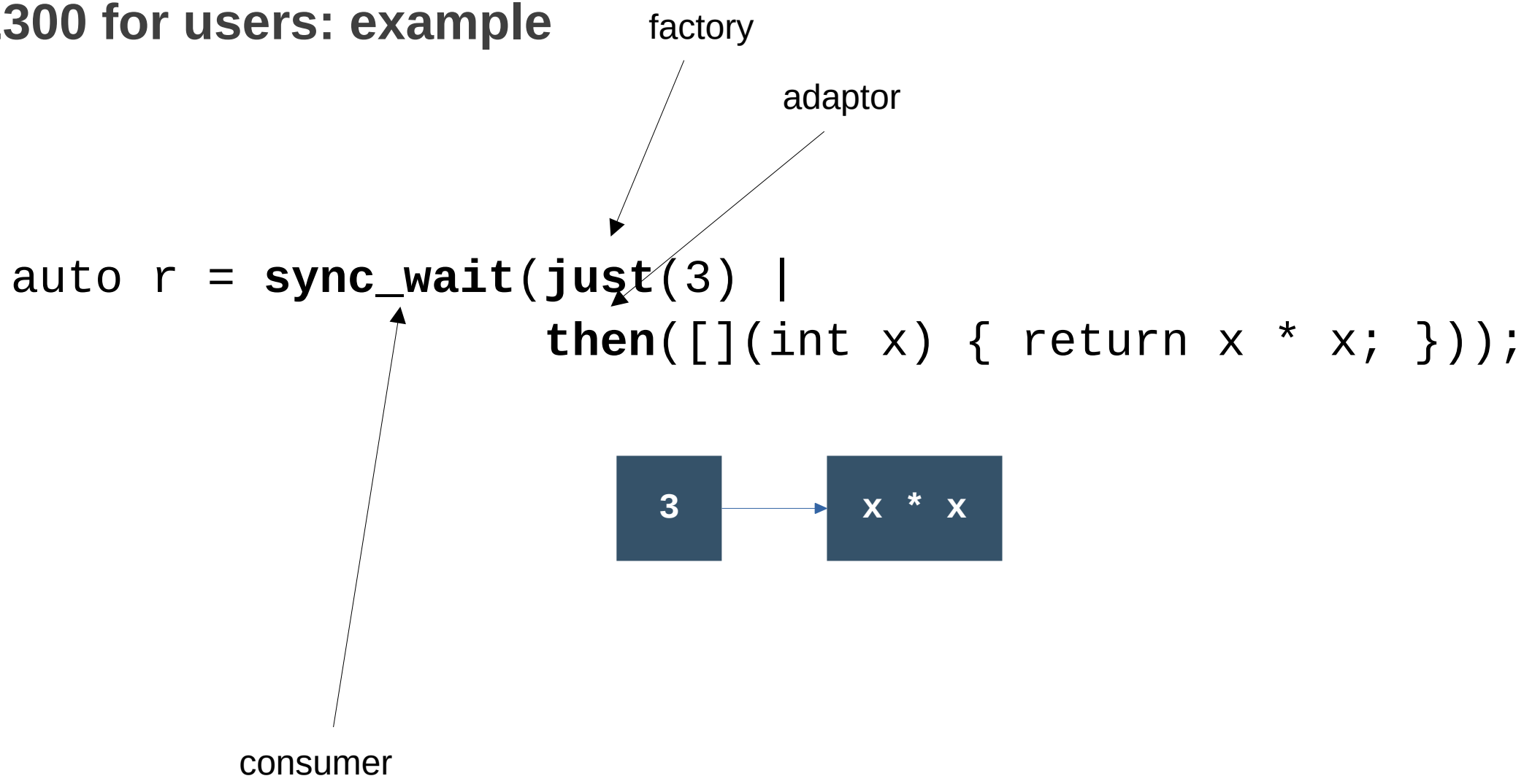
P2300 for users: most important parts

- **Senders**
  - Represent lazily submitted work
  - Generalization of a future into a concept
- **Sender algorithms**
  - Sender factories, adaptors, and consumers
  - Composed to create graphs of work
  - Most have default implementations, but can be customized
- **Schedulers**
  - a.k.a. executors
  - Handle to an execution context

P2300 for users: example

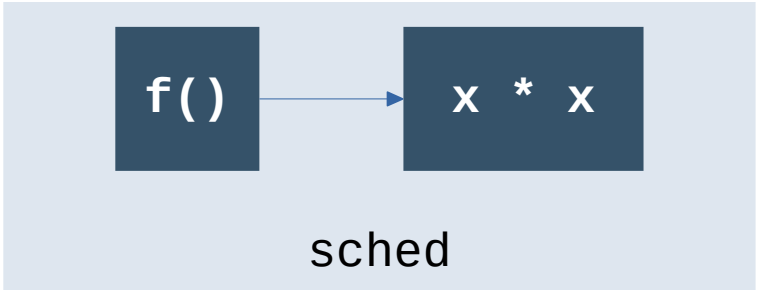


P2300 for users: example



P2300 for users: example

```
auto r = sync_wait(schedule(sched) |
                   then(f) |
                   then([](int x) { return x * x; }));
```

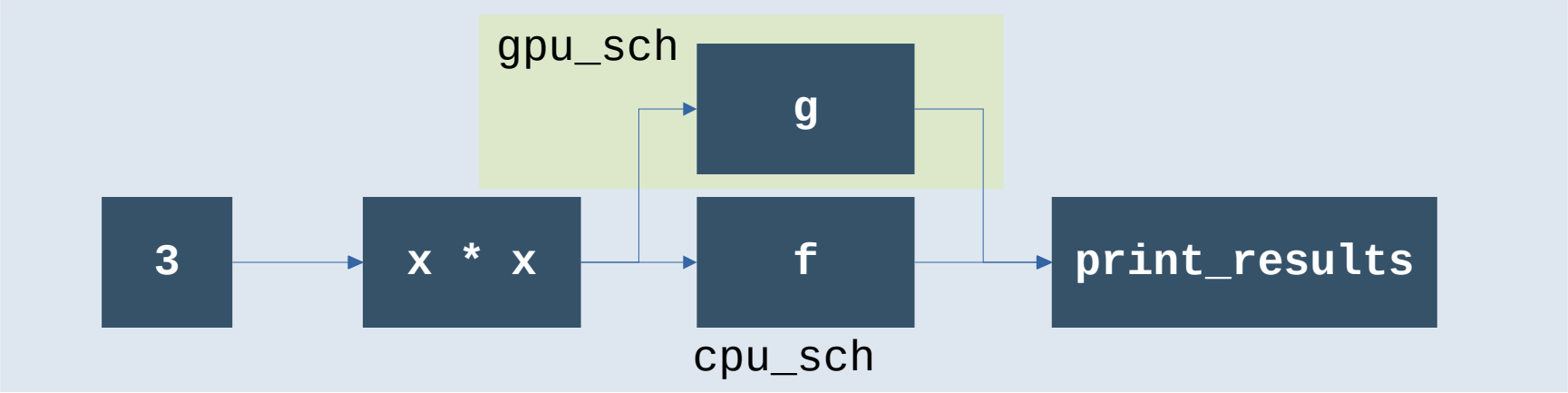


P2300 for users: more adaptors

- Long list of sender adaptors… a selection of useful ones below
  - `when_all`
  - `split`
  - `let_value`
  - `on_error`
  - `bulk`
  - `ensure_started`
  - `...`

P2300 for users: example

```
auto s1 = transfer_just(cpu_sch, 3) |
          then([](int x) { return x * x; }));
auto s2 = split(s1);
auto s3 = then(s2, f);
auto s4 = transfer(gpu_sch) | then(g) | transfer(cpu_sch);
auto s5 = when_all(s3, s4) | then(print_results);
sync_wait(s5);
```



P2300 for users: value, error, and cancellation channels

- Senders inform successors about the types that they “send” through completion signatures
  - `set_value_t()`
  - `set_value_t(int)`
  - `set_value_t(float, string)`
- Separate channels for errors and cancellation
  - `set_error_t(exception_ptr)`
  - `set_stopped_t()`

P2300 for implementers

- **Receivers**
  - Handle the completion signals from senders, senders and receivers are connected, used internally in sender algorithms
  - cf. `std::promise`
- **Operation states**
  - The result of connecting a sender and a receiver
  - Holds data required for an entire graph of operations
  - Allows to combine and elide heap allocations
  - Immovable
- Algorithms are customized using `tag_invoke` (https://wg21.link/p1895)

Why not `std::future`?

- Always type-erased
- Mandatory heap allocations
- Synchronization between execution contexts not customizable
  - Always requires synchronization on the CPU
- `std::future` has the wrong defaults to be a good base for asynchrony
- **But type-erasure still useful**
  - API boundaries
  - Limiting nested templates
  - `any_sender` currently not proposed, but exists in libunifex, pika, etc.

Can I already use P2300?

- **Reference implementation: https://github.com/NVIDIA/stdexec**
- libunifex: https://github.com/facebookexperimental/libunifex
- HPX: https://github.com/STEllAR-GROUP/hpx
- pika: https://github.com/pika-org/pika
- concore: https://github.com/lucteo/concore
- Simplified demo implementation: https://twitter.com/ericniebler/status/1524901033134022656

P2300 recap

- Generic library for asynchrony and heterogeneity
- Targeted for C++26
- Stand-alone proposal

pika

pika

- P2300 provides concepts, not implementations
- pika is a C++ library that provides implementations (and concepts, for now...):
  - A C++17 implementation of a subset of P2300
  - A CPU thread pool with user-level threads
  - CUDA, HIP, and MPI integration
  - Parallel algorithms implementation
- pika is a fork of HPX with most distributed functionality removed
- 0.9.0 release due today, monthly releases
- Follow semver, do not guarantee backwards compatibility in the API until 1.0.0
  - Will move to C++20 as soon as nvcc/nvhpc/CMake support it
  - Already have optional support for using the P2300 reference implementation

pika

- Thread pool with user-level stackful threads
  - Allows fine-grained tasking on the CPU
  - As good or better than OpenMP tasking
- MPI integration
  - `MPI_I*`, asynchronous MPI functions polled in the scheduler for completion
- CUDA/HIP integration
  - `cuda*Async,` events polled in the scheduler for completion

pika: wrapping asynchronous interfaces

- General requirement for asynchronous interfaces: a way to install a callback to signal completion
- CUDA integration in pika

```
cuda_scheduler sched;
sync_wait(transfer_just(sched, dst, src, count, kind) |
          then_with_stream(cudaMemcpyAsync) |
          ...);
```

- CUDA
  - Polling on CUDA events
  - `cudaLaunchHostFunc`
- MPI
  - Polling on requests

pika: why fork?

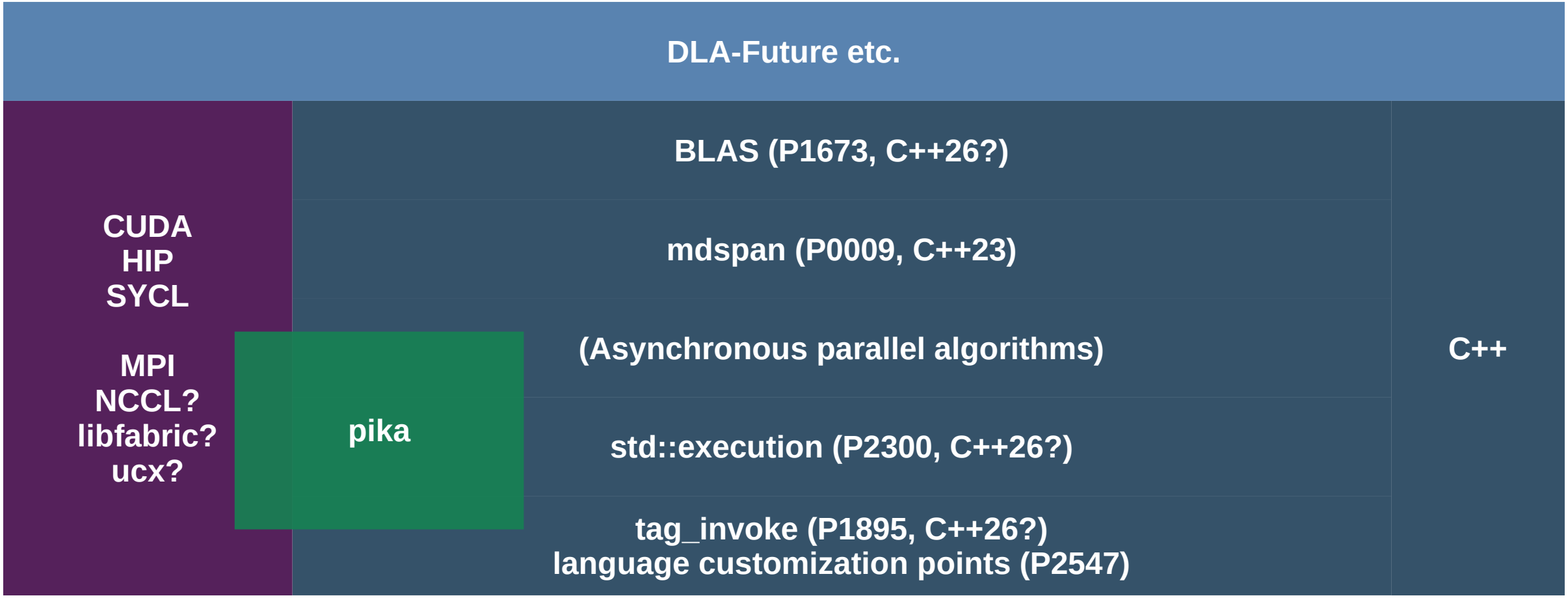
- HPX has much bigger ambitions than pika
- pika is a smaller development target
  - Faster development
  - Faster deprecation cycle
- Goal is to **gradually replace pika with standard and/or vendor alternatives**

P2300 + pika: the good, the bad, and the ugly

- The good
  - Customization, optimization, flexibility
  - Futures ↔ sender interop was very straightforward to implement, so far haven't had anything we felt we couldn't implement
  - Usage typically straightforward for users to understand
- The bad
  - Grokking the internals of the P2300 model takes time
  - Compile-times (as with anything heavily templated in C++)
  - Error messages (as with anything heavily templated in C++)
- The ugly
  - C++26, maybe?

std::execution + pika + ? in the future

C++ scientific computing ecosystem



Conclusion

Conclusion

- Working assumption: C++ is at the moment the most appropriate language for performance portable HPC applications → focus on solutions that are or will be available in the C++ standard
- P2300 latest proposal for a unified asynchronous model for C++
- pika is our vehicle for transitioning to a standard solution
- **Questions?**