Half-Precision Scalar Support in Kokkos and Kokkos Kernels: An Engineering Study and Experience Report

Evan Harvey, Reed Milewicz, Christian Trott,
Luc Berger-Vergiat, Siva Rajamanickam

Research Software Engineers in eScience Workshop (RSE-eScience-2022)

10 October 2022

Icons courtesy of the Noun Project; Minnie Pigeon, Amythest Studio, Juicy Fish, and Adrien Coquet Illustrations from Undraw.co (copyright 2022 Katerina Limpitsouni) and Draw.io.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2022-13941 C

Overview

In our paper, we present a two-part study on the development of a performance portability library feature to support science and engineering applications.

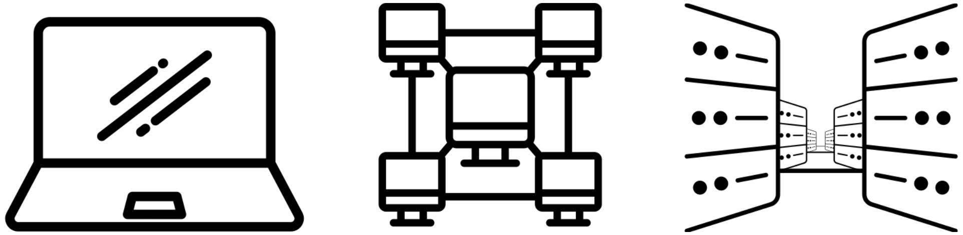
❑ An **engineering study** on the technical implementation of the feature.

❑ An **experience report**, from an RSE perspective, on the challenges and lessons learned.

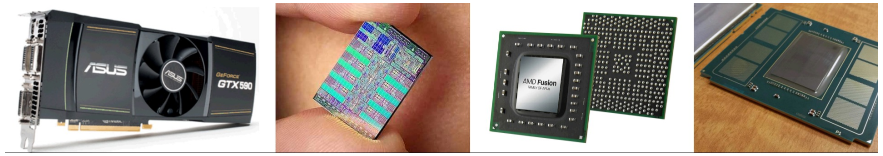
Engineering Study

What is Kokkos?

- A C++ programming model and software library ecosystem for performance portability
  - Implemented as a template library on top of CUDA, OpenMP, HPX, …
  - Aims to be descriptive not prescriptive
  - Aligns with developments in the C++ standard
  - Replaces usage of CUDA, OpenMP, HIP, etc.
- Expanding solution for common needs of modern science/engineering codes
  - Math libraries based on Kokkos
  - Tools which enable insight into Kokkos
- Open source and widely used across a range of institutions and disciplines
  - Maintained and developed at https://github.com/kokkos

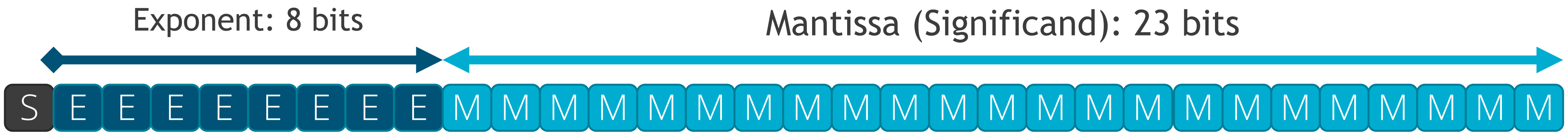


Performance portability
from **laptops** to **clusters** to **supercomputers**

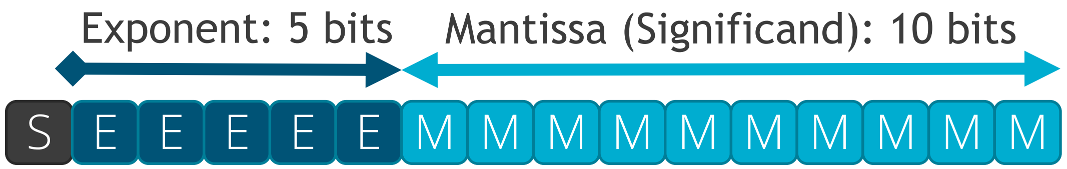


And many types of **hardware**

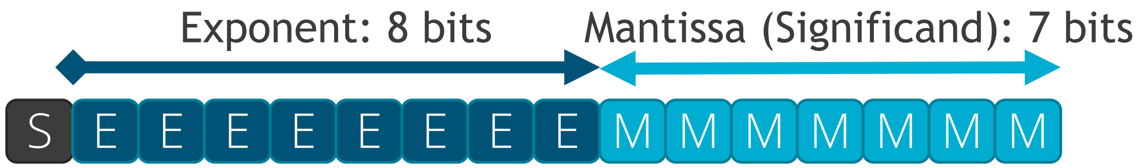
Adding Half-Precision Floating Point Support to Kokkos


Exponent: 8 bits
Mantissa (Significand): 23 bits
S E E E E E E E E M M M M M M M M M M M M M M M M M M M M M M M

`fp32`


Exponent: 5 bits
Mantissa (Significand): 10 bits
S E E E E E M M M M M M M M M M

`binary16`


Exponent: 8 bits
Mantissa (Significand): 7 bits
S E E E E E E E E M M M M M M M

`bfloat16`

Computational science and machine learning researchers are increasingly interested in utilizing half-precision to optimize and scale their algorithms, could benefit from the addition of half-precision support.

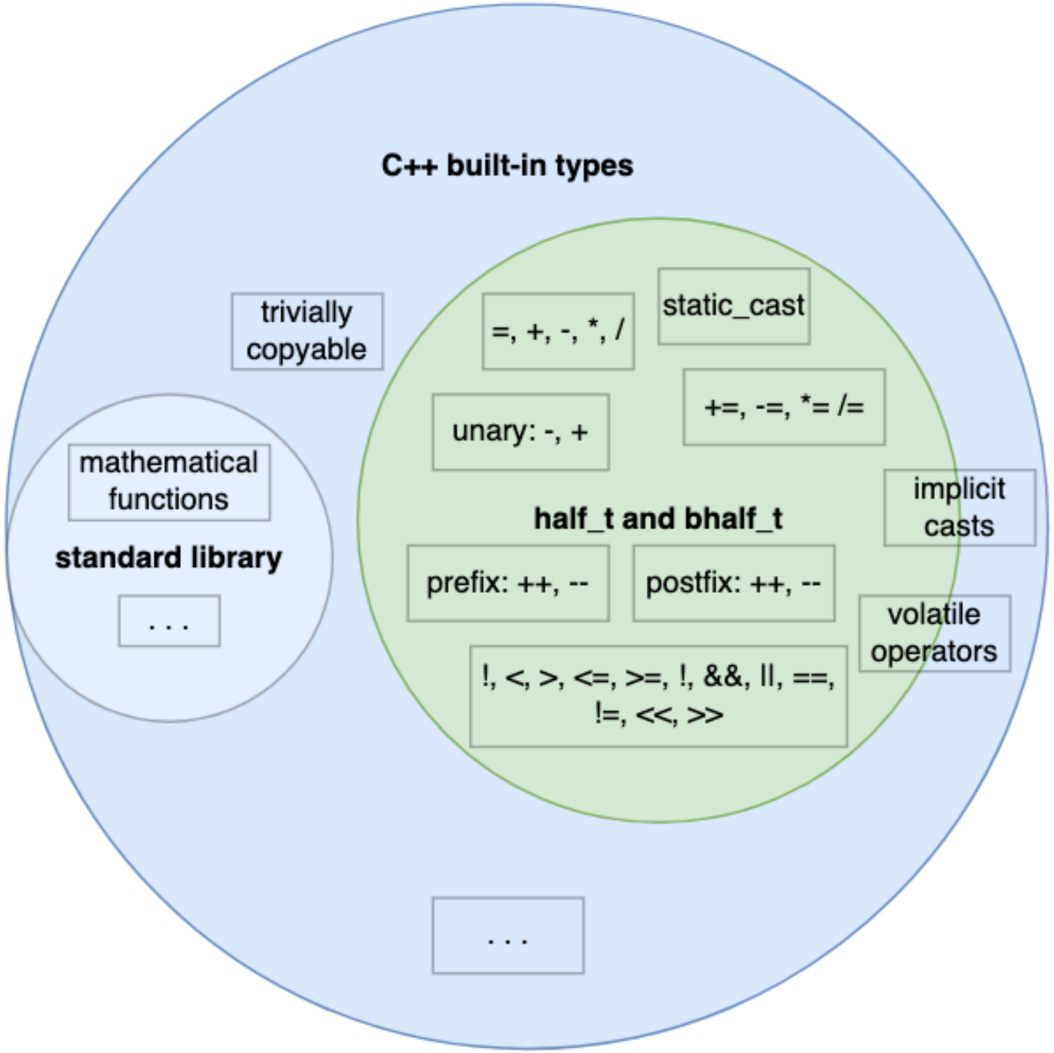
❑ A 16-bit floating point encoding

❑ `float` encapsulates fp32

❑ `Kokkos::Experimental::half_t` encapsulates binary16

❑ `Kokkos::Experimental::bhalf_t` encapsulates bfloat16

What We Implemented in Kokkos: `half_t`



❑ `half_t` is either an alias to *float* or a C++ class

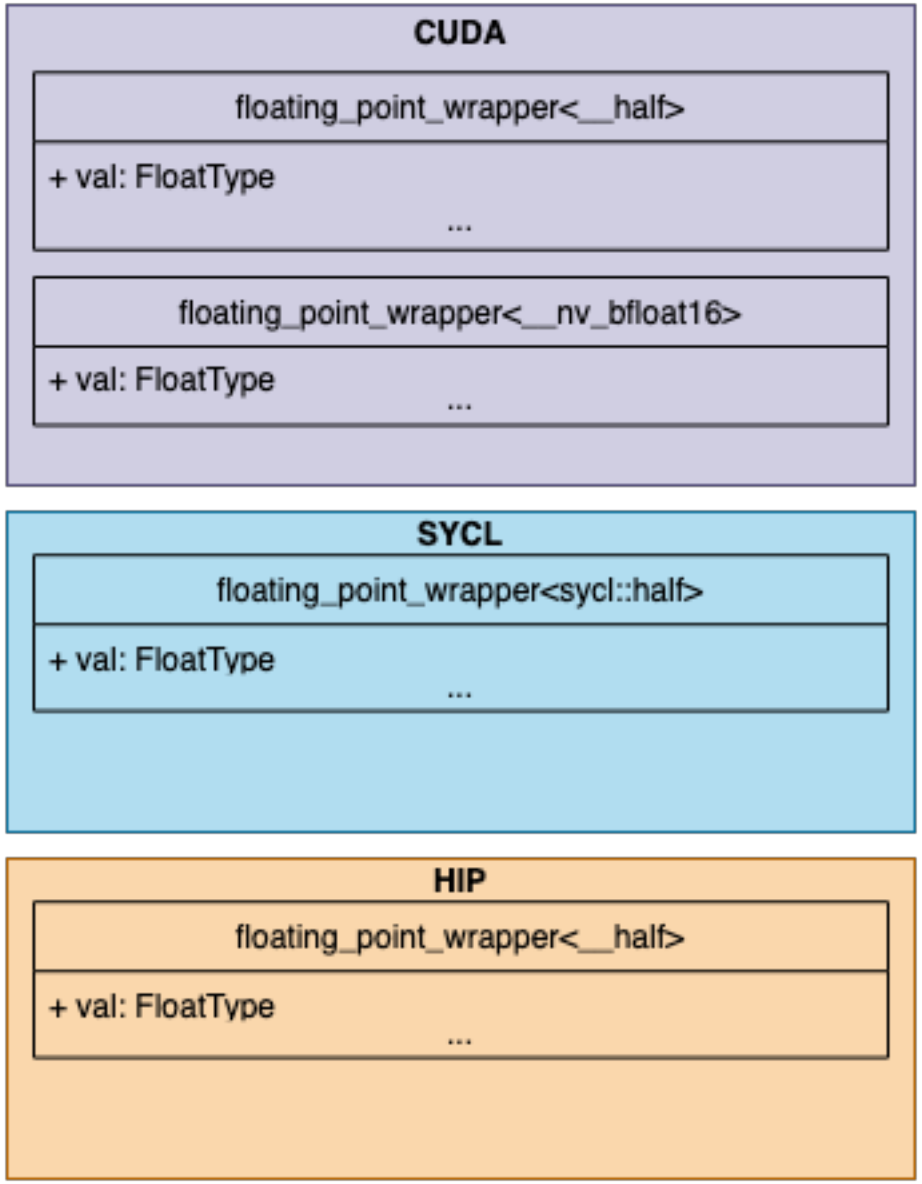
❑ `half_t` acts like float via:
- ○ casting wrappers with forward declarations
- ○ operator overloading with compile-time branches

❑ Volatile operations

❑ Mixed precision:
- ○ `T op half_t`
- ○ `half_t op T`

What We Implemented in Kokkos: `bhalf_t`

- □ Uses the same code as `half_t` except for:
  - ○ Underlying data-type encodes bfloat16 via template argument
  - ○ Casting wrappers are overloaded to call bfloat16 intrinsics



CUDA

floating_point_wrapper<__half>

+ val: FloatType

...

floating_point_wrapper<__nv_bfloat16>

+ val: FloatType

...

SYCL

floating_point_wrapper<sycl::half>

+ val: FloatType

...

HIP

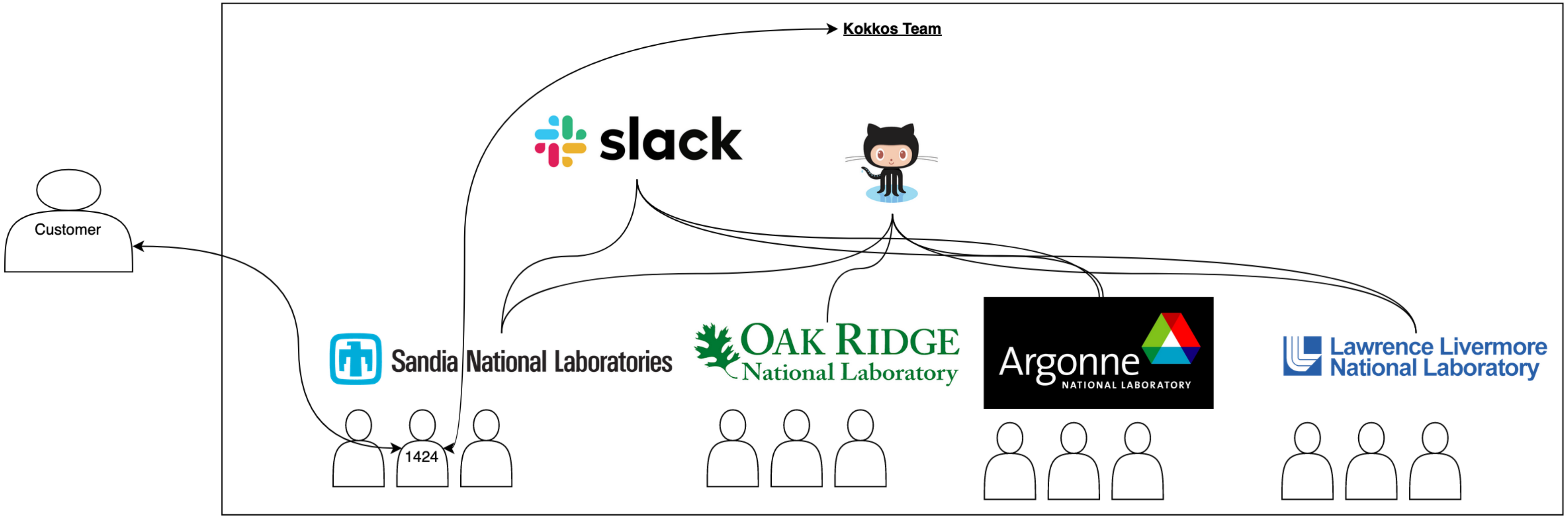
floating_point_wrapper<__half>

+ val: FloatType

...

Experience Report

Lesson Learned: Stay Engaged with Real Users and Their Needs


Kokkos Team
slack
Customer
Sandia National Laboratories
1424
Oak Ridge National Laboratory
Argonne NATIONAL LABORATORY
Lawrence Livermore National Laboratory

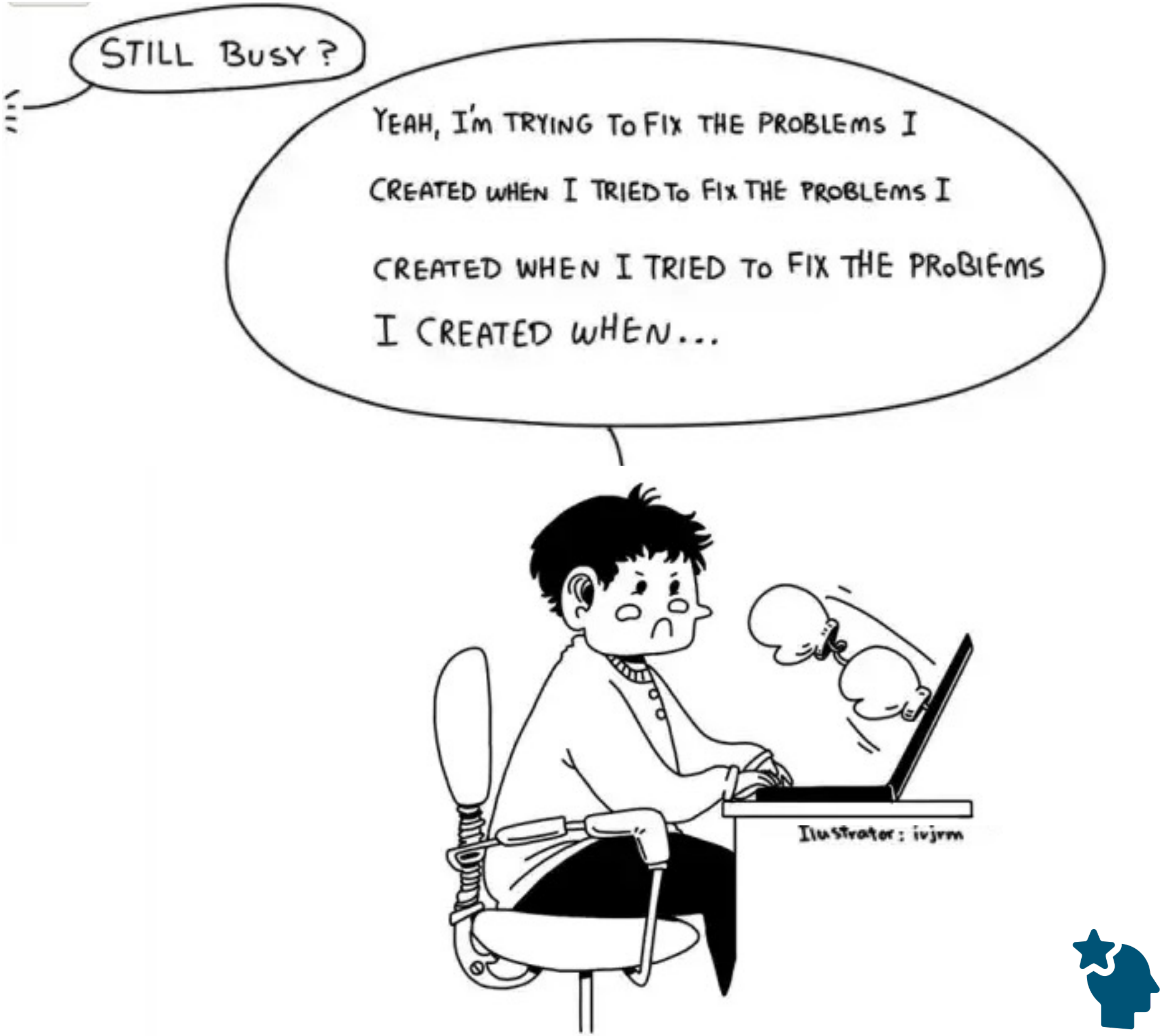
Feature development for scientific software libraries should be grounded in the **needs of real users**. Proactively identify prospective stakeholders and engage with them frequently to **gather requirements**.

Lesson Learned: Choose Development Methodologies Carefully

Be **intentional in the choice of development methodology**, and consider both your individual needs as a developer and those of your customers – different tasks may require different approaches.

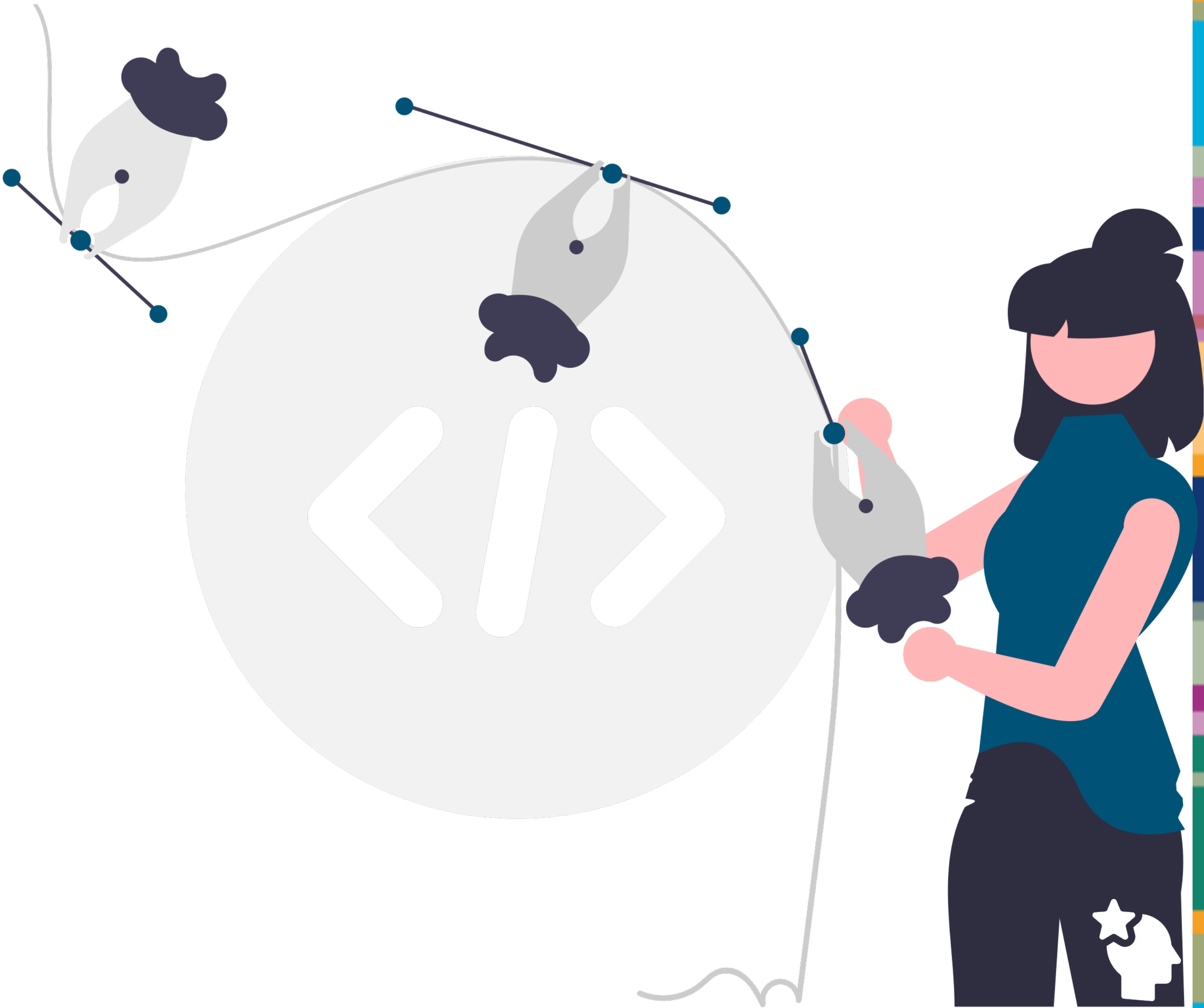
Lesson Learned: Pay Down Technical Debt Early and Often

It is important to **pay down technical debt by refactoring early and often**. Of particular note when developing scientific software libraries, latent technical debt can emerge in public interfaces and, once in place, is persistent and hard to remove.


STILL BUSY?
YEAH, I'M TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...
Illustrator: ivjrm

Lesson Learned: Know Your Tools

As an RSE, **know your tools**. Case in point, modern programming languages have powerful and flexible features, but they can also be a source of complexity that must be managed. Knowing what language features to use and when is a key part of good software craftsmanship.



Acknowledgments

- For questions and comments, feel free to reach out to me at eharvey@sandia.gov
- I would like to thank my mentors
  - Reed Milewicz
  - Christian Trott
  - Siva Rajamanickam
- Our sponsor
  - Exascale Computing Project
- Lastly, I would like to thank my manager

ECP