



# Scientific Software Design



David E. Bernholdt (he/him)  
Oak Ridge National Laboratory

Better Scientific Software tutorial @ Improving Scientific Software 2022

Contributors: David E. Bernholdt (ORNL), Anshu Dubey (ANL), Mark C. Miller (LLNL)




See slide 2 for  
license details



# License, Citation and Acknowledgements

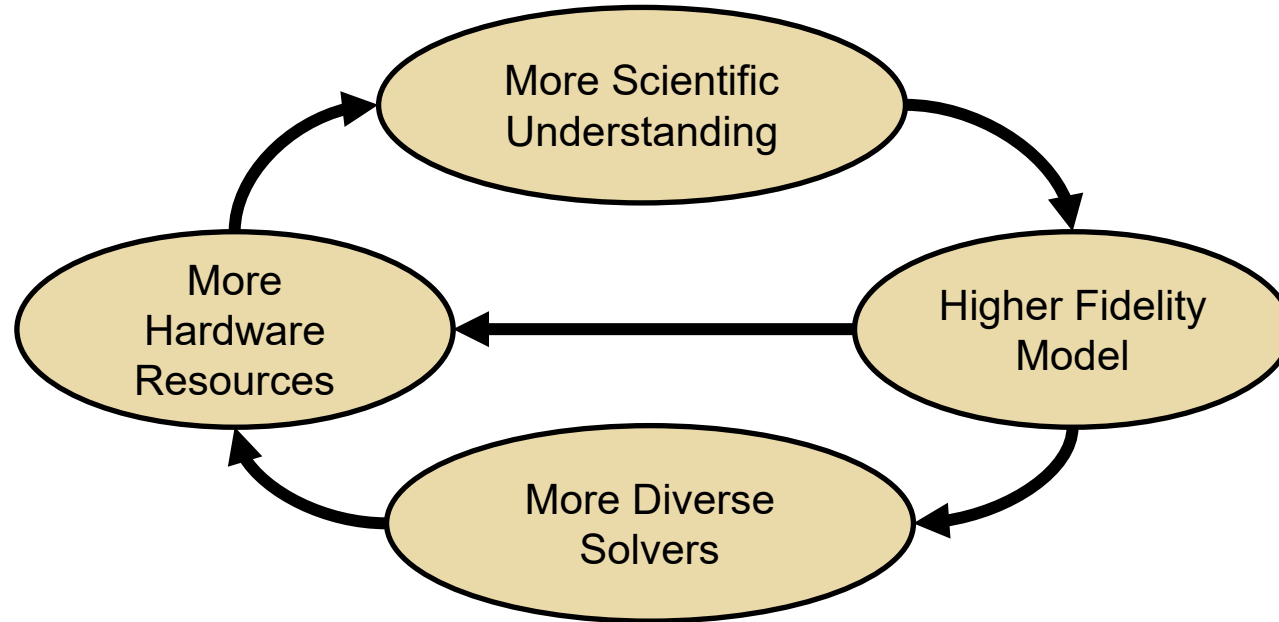
## License and Citation

- This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) (CC BY 4.0). 
- **The requested citation the overall tutorial is: David E. Bernholdt, Patricia A. Grubel, Rinku K. Gupta, and David M. Rogers, Better Scientific Software tutorial, in Improving Scientific Software conference, online, 2022. DOI: [10.6084/m9.figshare.19416767](https://doi.org/10.6084/m9.figshare.19416767)**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial...

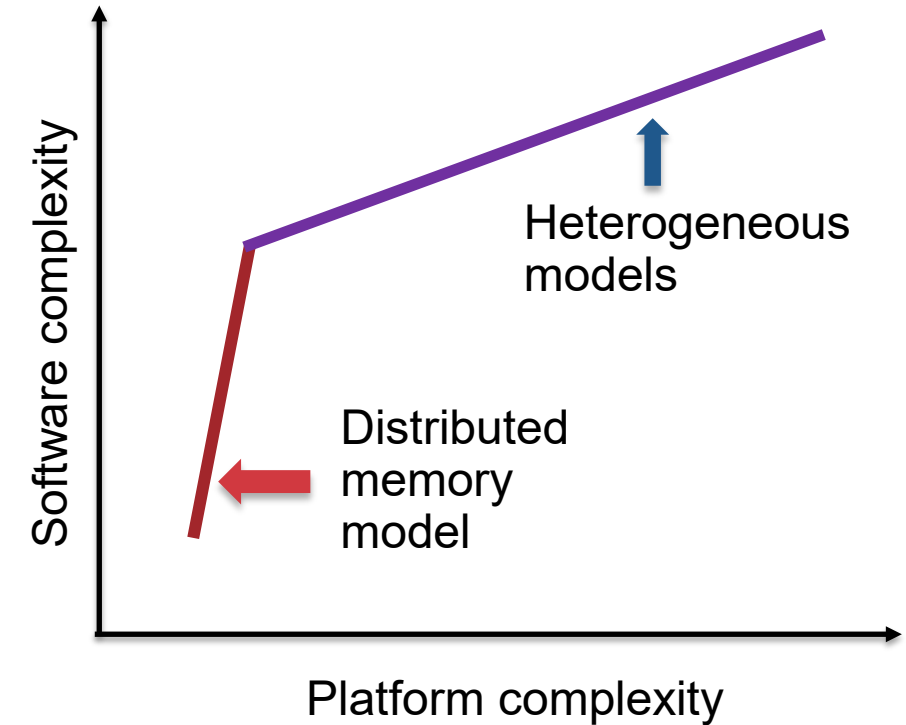
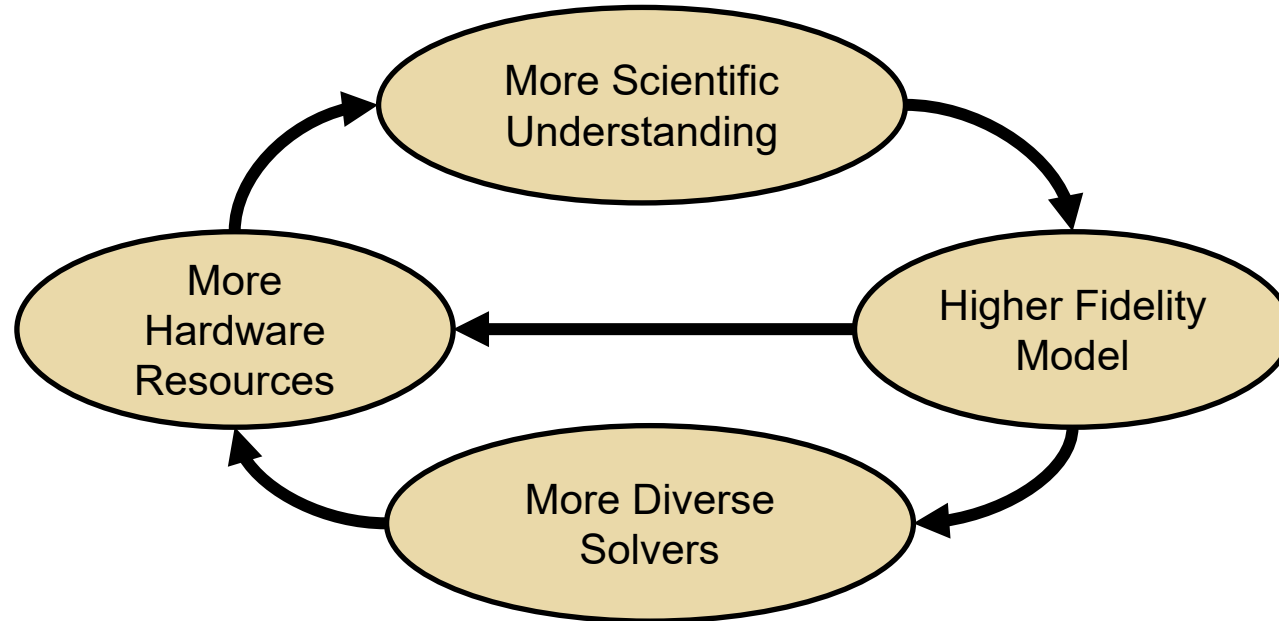
## Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No. 89233218CNA000001
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and 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.

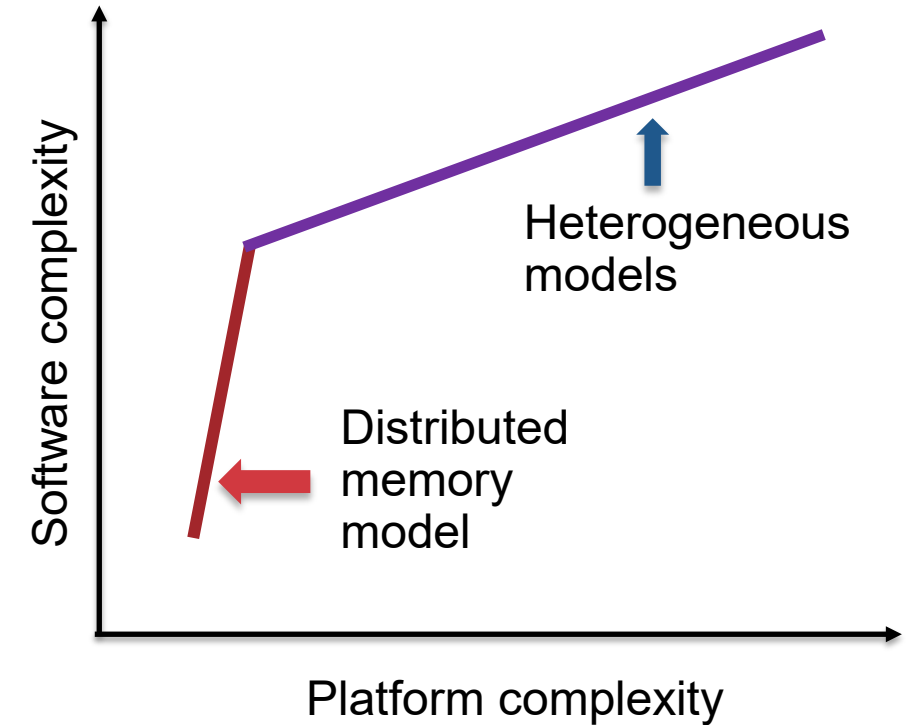
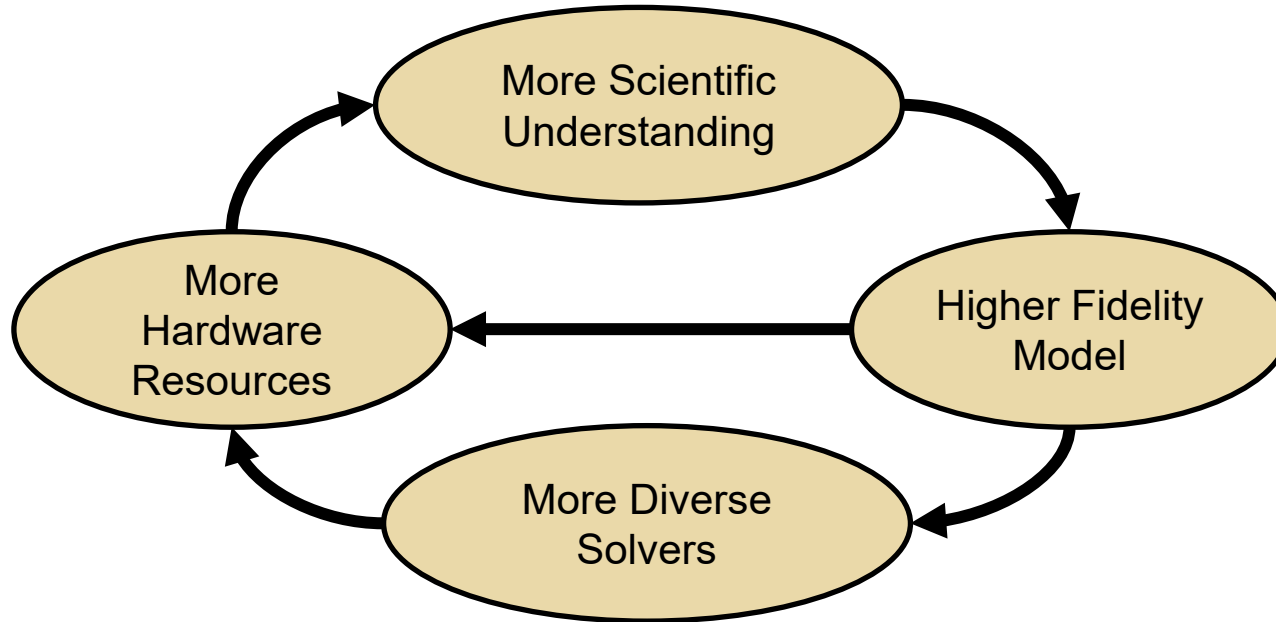
# HPC Computational Science Use-case



# HPC Computational Science Use-case



# HPC Computational Science Use-case



- ❑ Many components may be under research
- ❑ Software continuously evolves
- ❑ All use cases are different and unique

# General Design Principles for HPC Scientific Software

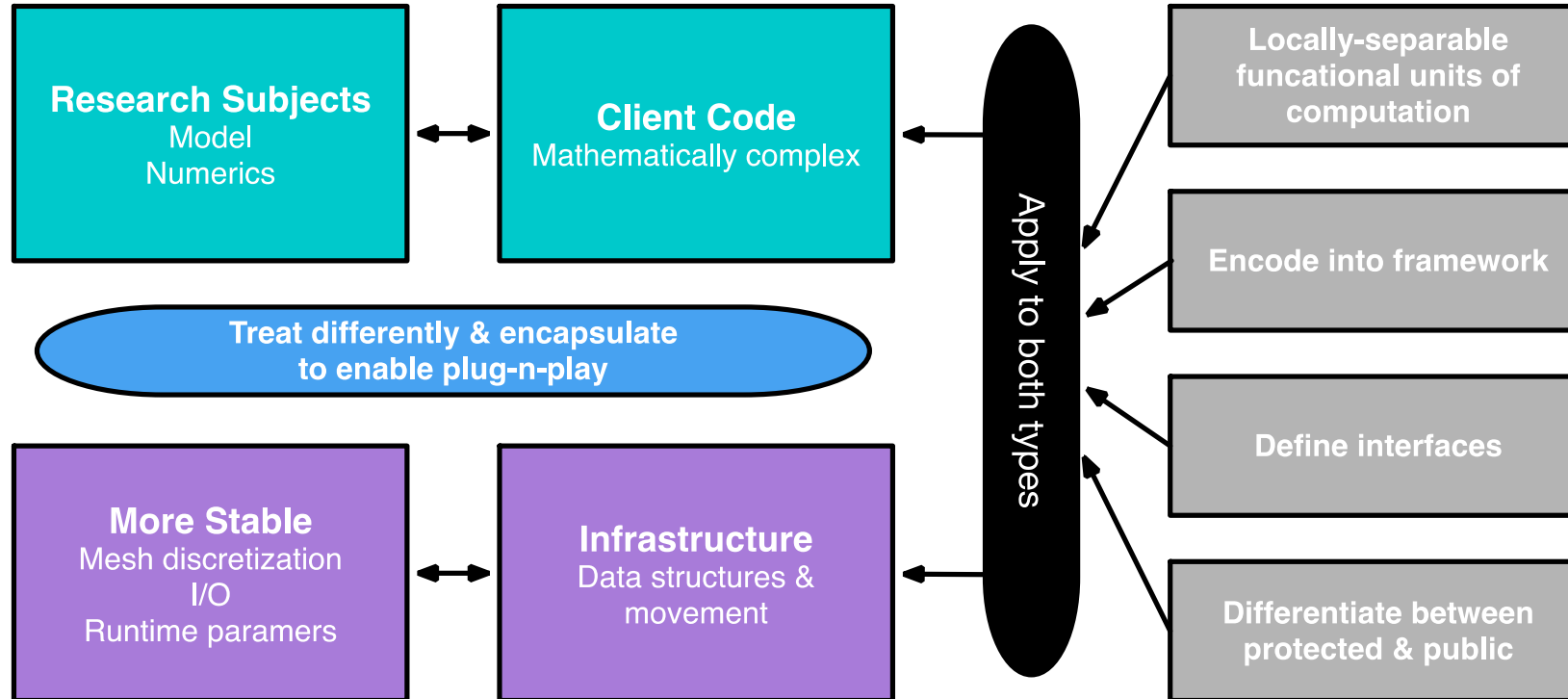
## Considerations

- ❑ Multidisciplinary teams
  - ❑ Many facets of knowledge
  - ❑ To know everything is not feasible
- ❑ Two types of code components
  - ❑ Infrastructure (mesh/IO/runtime ...)
  - ❑ Science models (numerical methods)
- ❑ Codes grow
  - ❑ New ideas => new features
  - ❑ Code reuse by others

## Design Implications

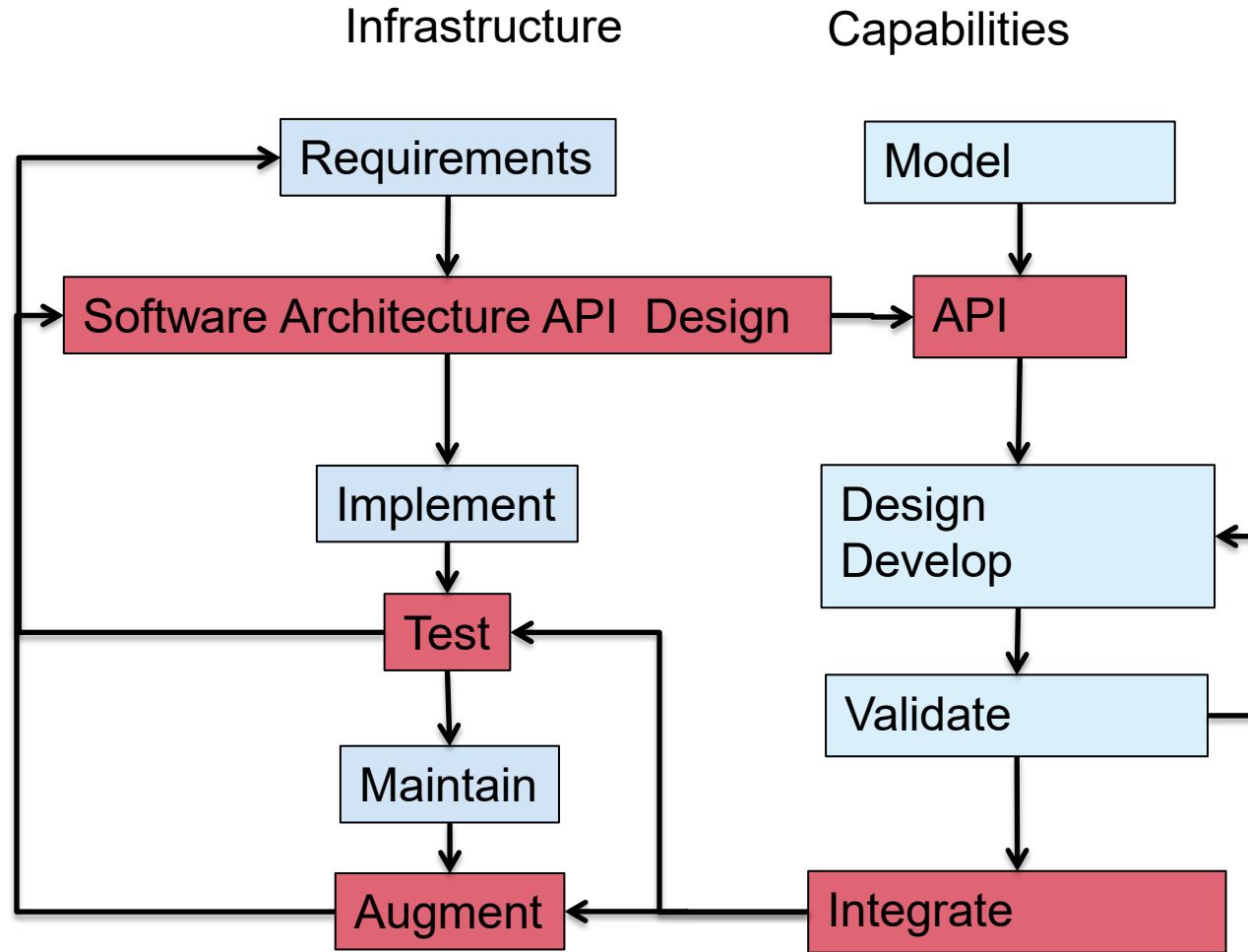
- ❑ Separation of Concerns
  - ❑ Shield developers from unnecessary complexities
- ❑ Work with different lifecycles
  - ❑ Long-lasting vs quick changing
  - ❑ Logically vs mathematically complex
- ❑ Extensibility built in
  - ❑ Ease of adding new capabilities
  - ❑ Customizing existing capabilities

# General Design Principles for HPC Scientific Software



**Design first, then apply programming model to the design instead of taking a programming model and fitting your design to it.**

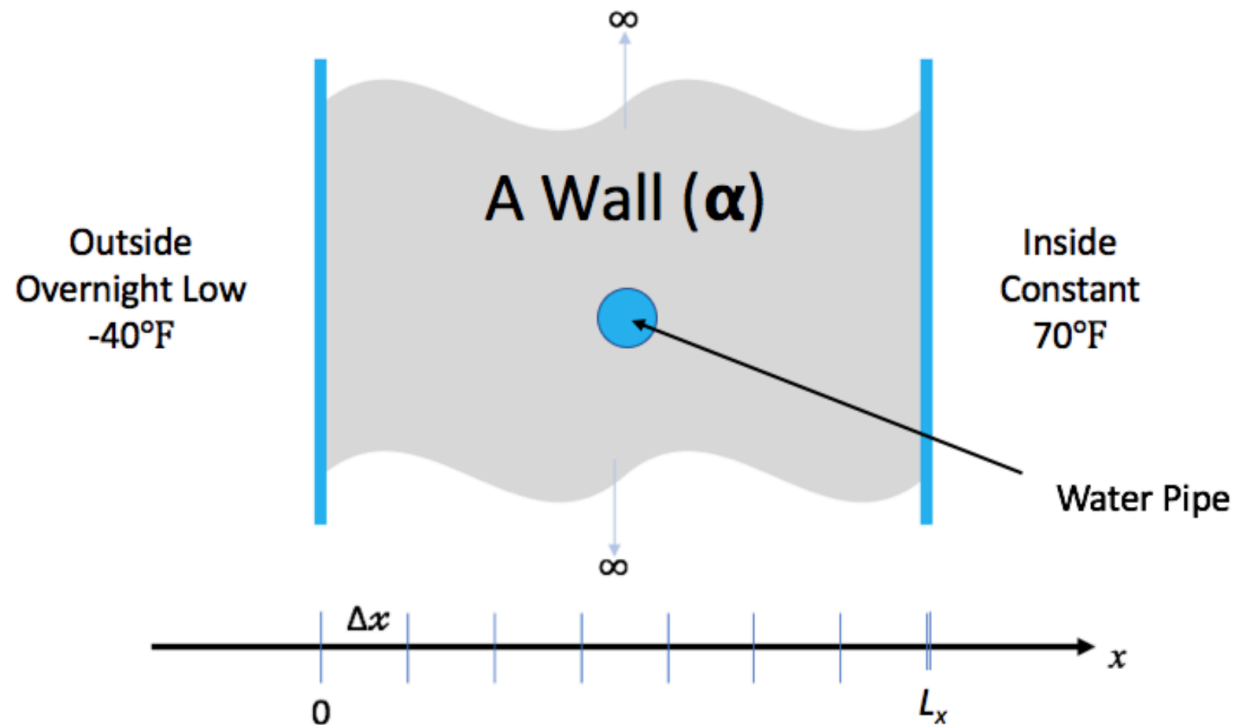
# A Design Model for Separation of Concerns





# The Running Example

Lets say you live in a house with exterior walls made of a single material of thickness,  $L_x$ . Inside the walls are some water pipes as pictured below.



You keep the inside temperature of the house always at 70 degrees F. But, there is an overnight storm coming. The outside temperature is expected to drop to  $-40$  degrees F for 15.5 hours. Will your pipes freeze before the storm is over?

# Problem Specification - Design Considerations

- Specification
  - Solve heat equation with some initial and boundary conditions
  - Apply different integration methods

- What is infrastructure here?
  - Discretization/ State
  - Verification
  - I/O
  - Application of initial conditions
  - Runtime parameters
  - Comparison

- What is model here?
  - Initial conditions
  - Boundary conditions
  - Integration

# Infrastructure API

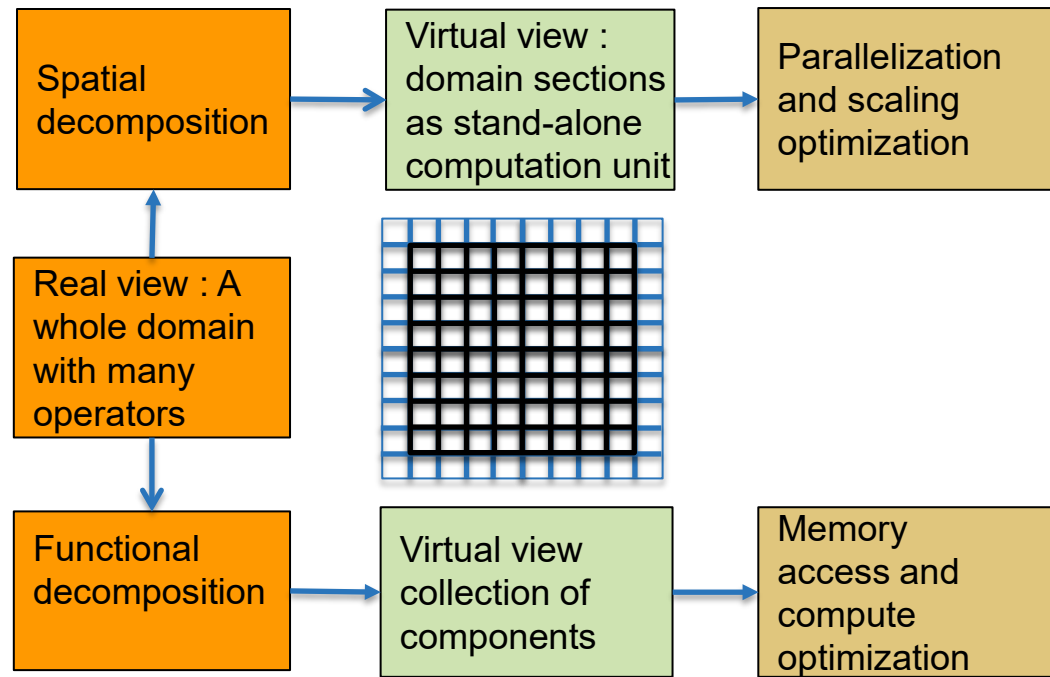
- **process\_args**(int argc, char \*\*argv)
- static void **initialize**(void)
- void **copy**(int n, double \*dst, double const \*src)
- void **write\_array**(int t, int n, double dx, double const \*a)
- void **set\_initial\_condition**(int n, double \*a, double dx, char const \*ic)

# Numerics API

- double **l2\_norm**(int n, double const \*a, double const \*b)
- bool **update\_solution\_crankn**(int n, double \*curr, double const \*last, double const \*cn\_Amat, double bc\_0, double bc\_1)
- bool **update\_solution\_upwind15**(int n, double \*curr, double const \*last, double alpha, double dx, double dt, double bc\_0, double bc\_1)
- bool **update\_solution\_ftcs**( int n, double \*uk1, double const \*uk0, double alpha, double dx, double dt, double bc0, double bc1)
- void **compute\_exact\_solution**(int n, double \*a, double dx, char const \*ic, double alpha, double t, double bc0, double bc1)

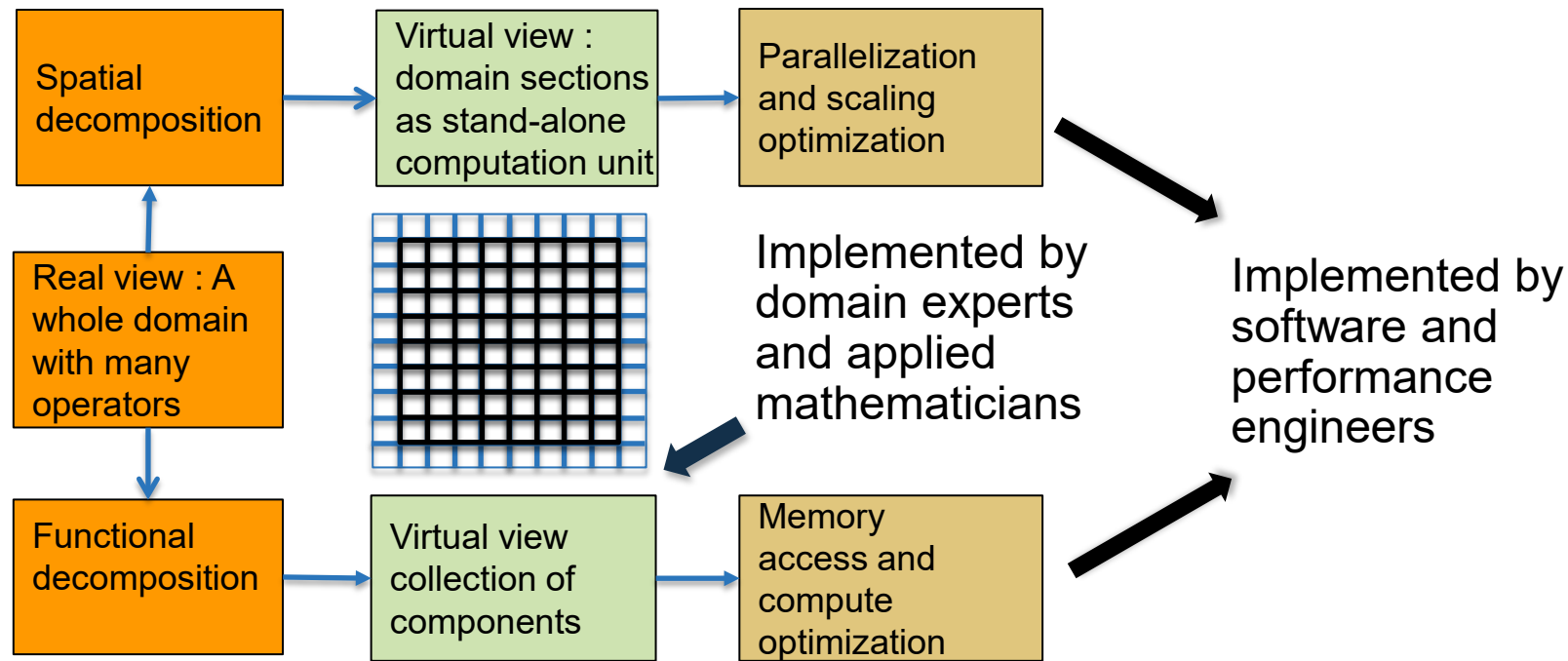
# Example: Architecting Multiphysics PDEs

- Virtual view of functionalities
- Decomposition into units and definition of interfaces



# Example: Multiphysics PDEs for Distributed Memory Parallelism

- Virtual view of functionalities
- Decomposition into units and definition of interfaces

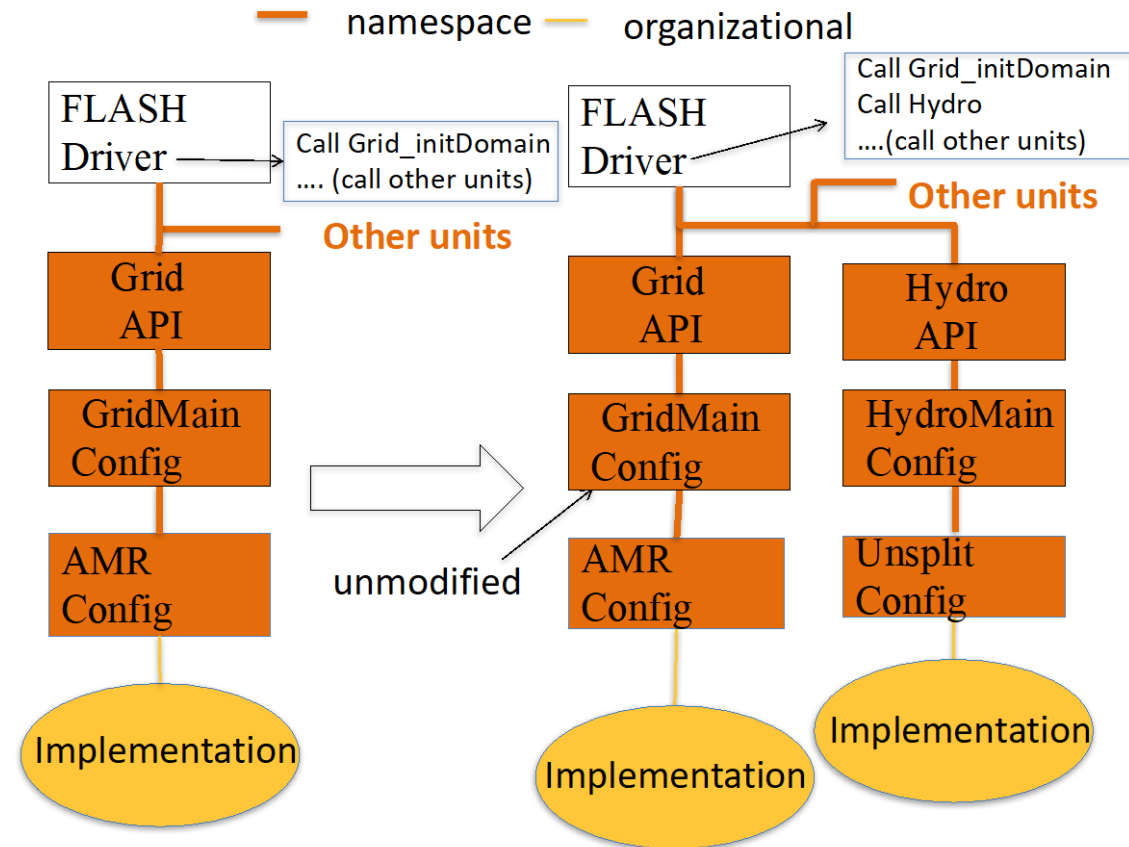


# Example: Design for Extensibility from FLASH, Now Flash-X

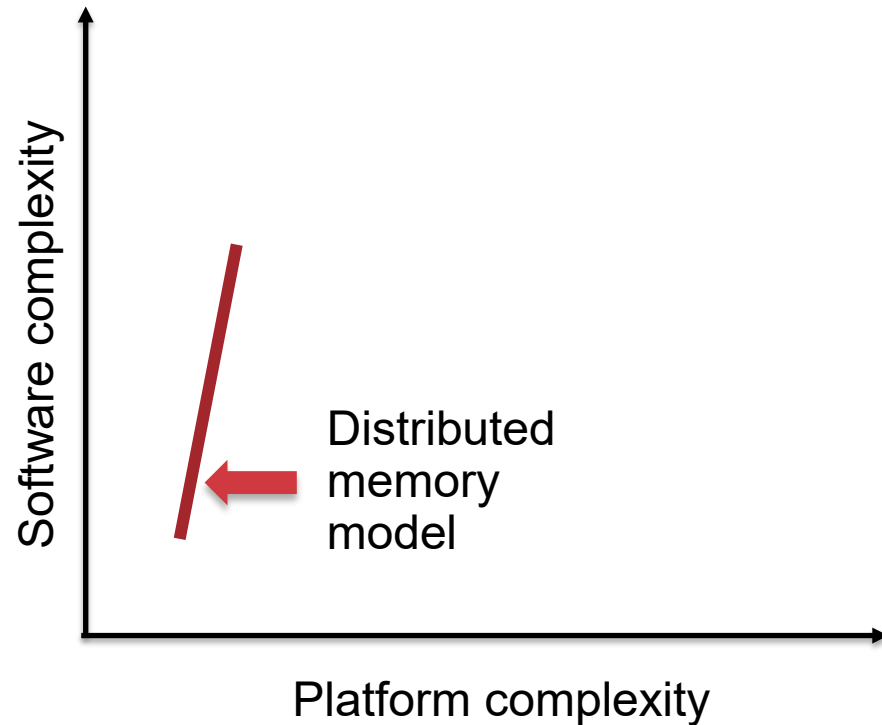
Assumed that capabilities will be added for better models

- Assembly from components
- Decentralized maintenance of metadata
- Python tool to parse and configure
- OOP implemented through Unix directory structure and configuration tool

**Key idea is distributed intelligence**



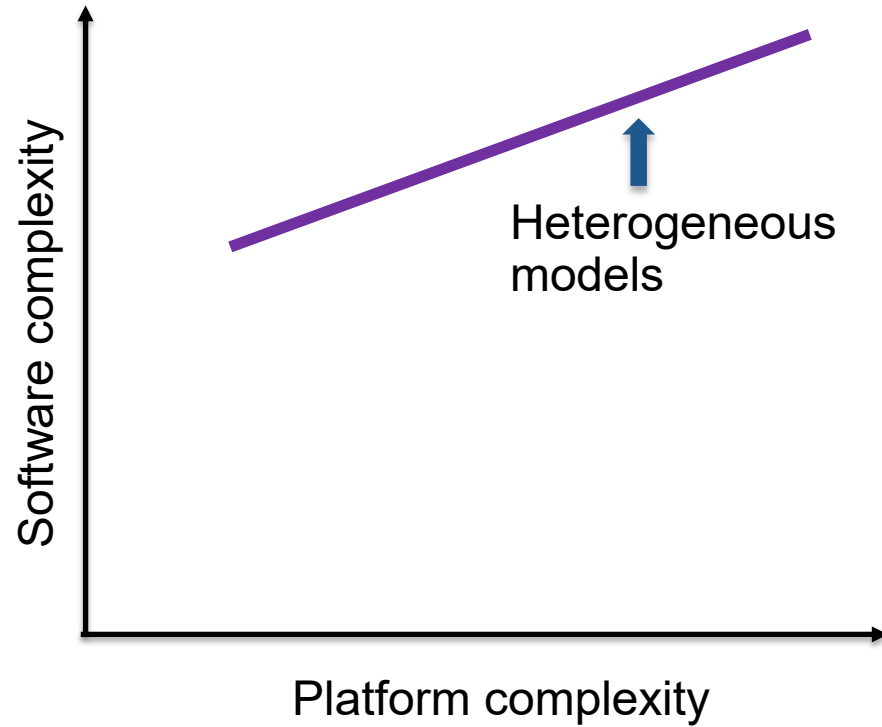
# Takeaways Until Now



- Differentiate between slow changing and fast changing components of your code
- Understand the requirements of your infrastructure
- Implement separation of concerns
- Design with portability, extensibility, reproducibility and maintainability in mind
- Do not design with a specific programming model in mind

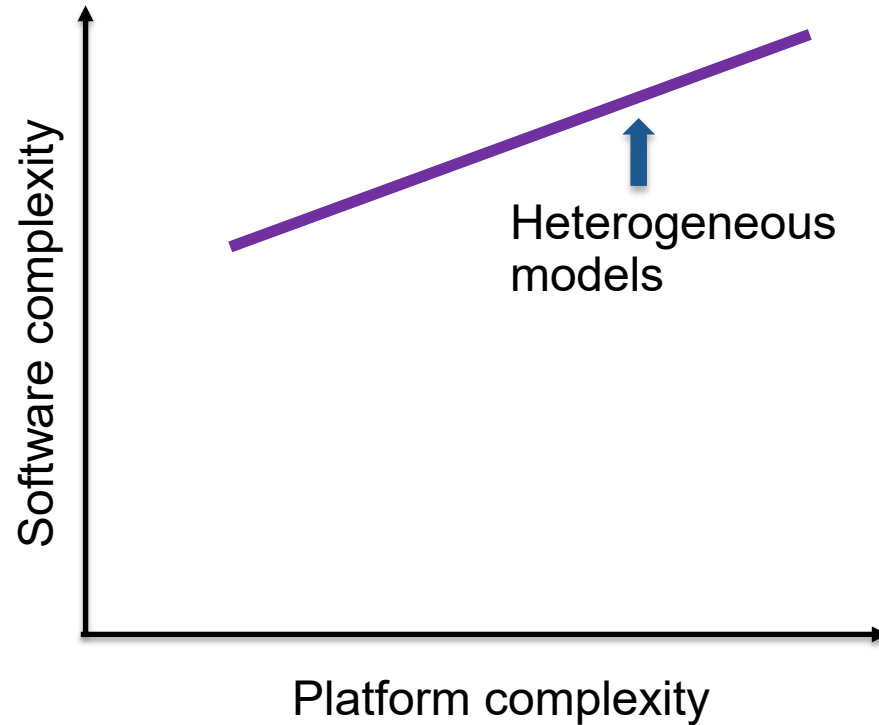


# A New Paradigm Because of Platform Heterogeneity



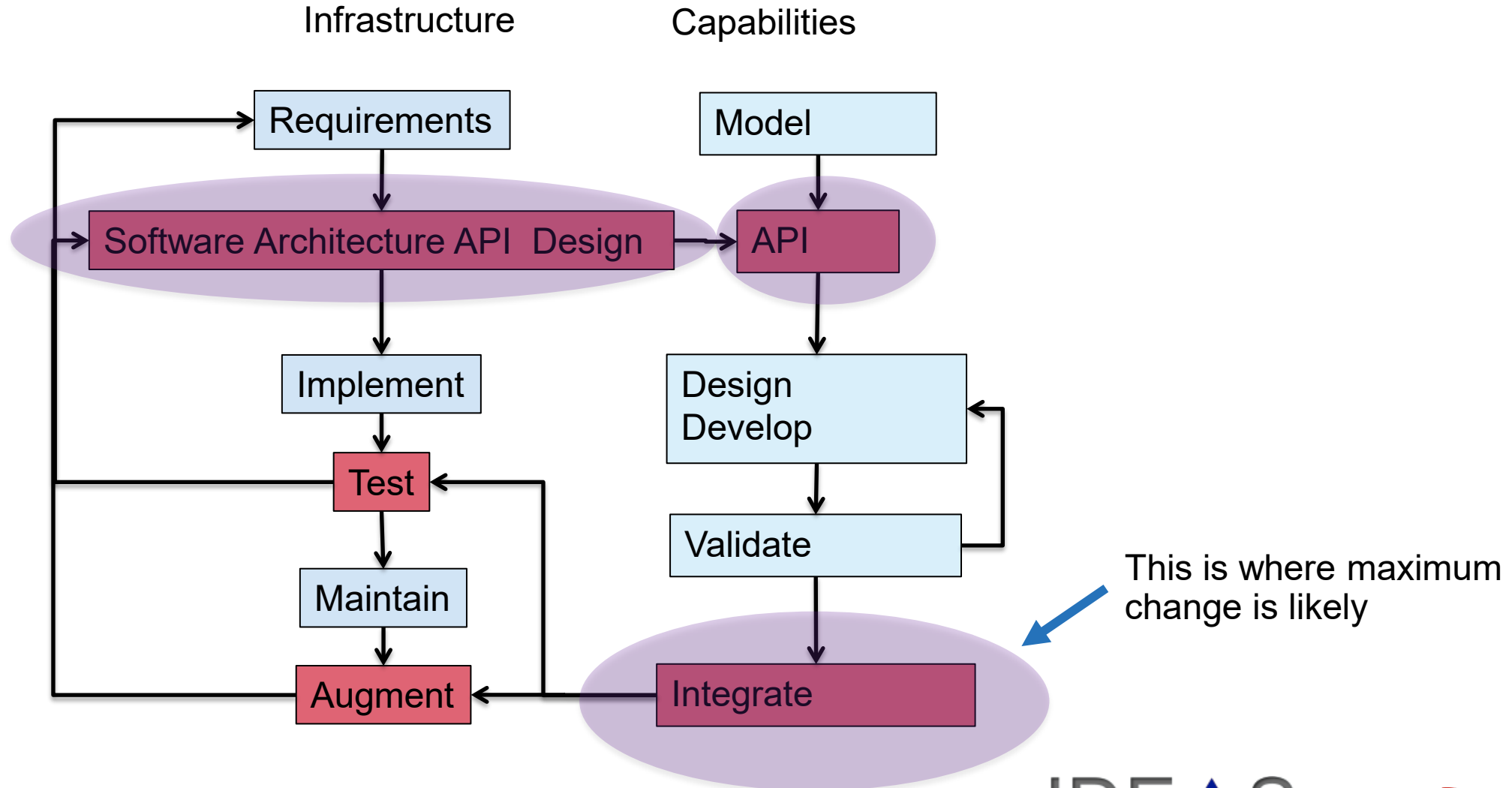
- Question - do the design principles change?

# A New Paradigm Because of Platform Heterogeneity



- Question - do the design principles change?
- The answer is – not really
- The details get more involved

# A Design Model for Separation of Concerns



# Design Guidance For Performance Portability

Design for Hierarchical parallelism

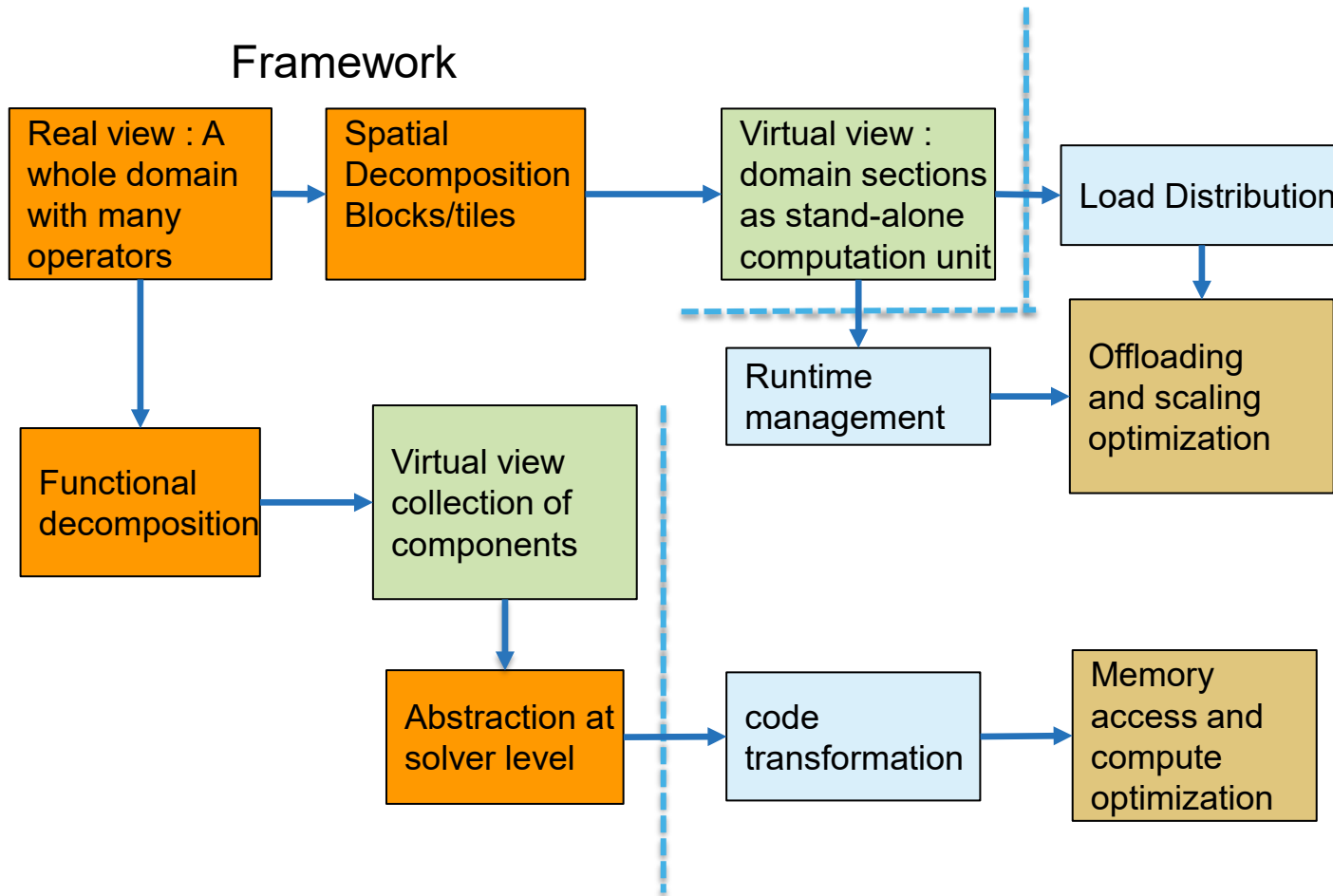
Design towards several thousand threads

Design for a hierarchical memory space

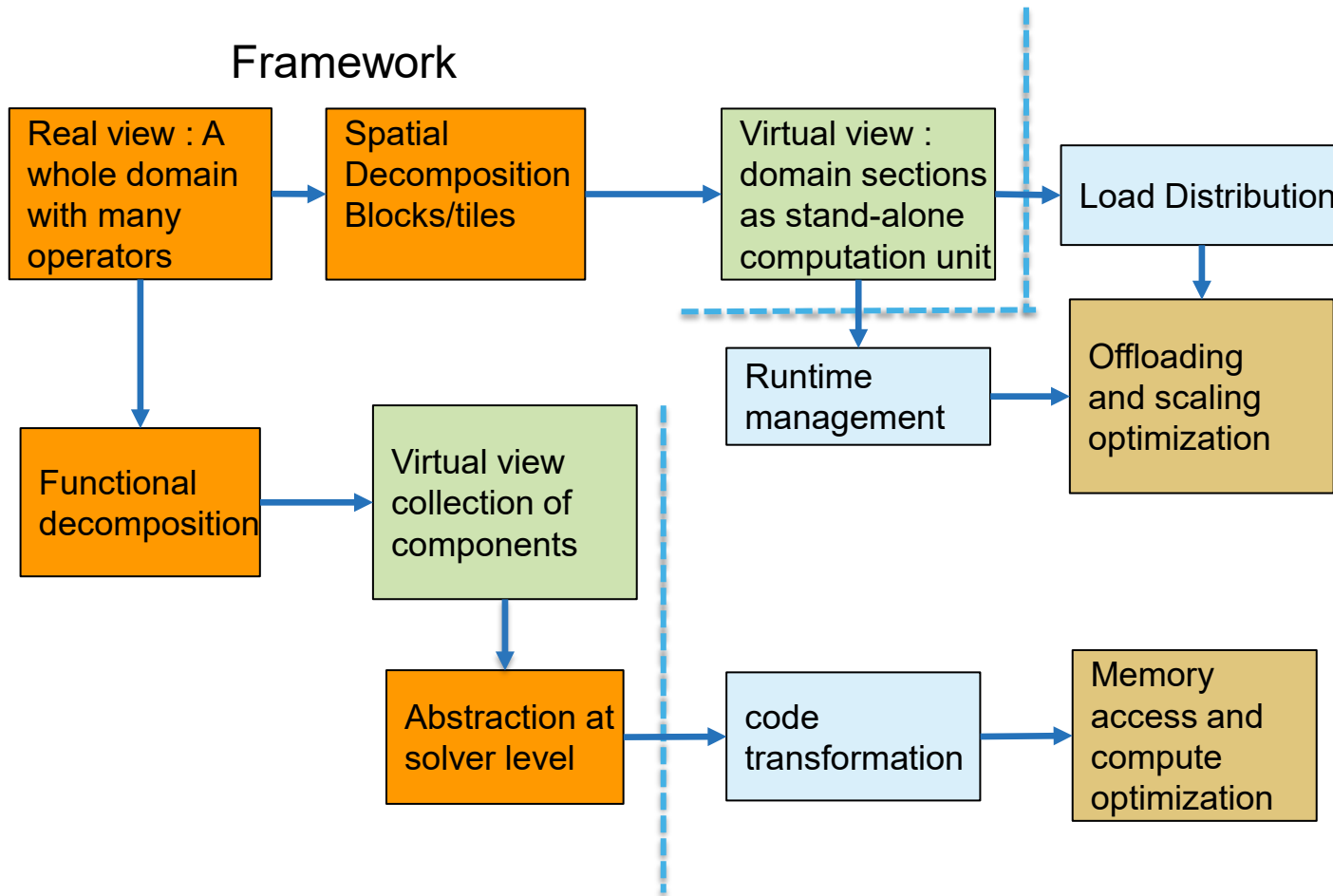
Design patterns that count, allocate, and reuse memory

Avoid exposing/using non-portable vendor-specific options

# Features and Abstractions that must Come in



# Features and Abstractions that must Come in



## How do abstraction layers work

- ☐ Infer the structure of the code
- ☐ Infer the map between algorithms and devices
- ☐ Infer the data movements
- ☐ Map computations to devices
- ☐ These are specified either through constructs or pragmas

**Performance depends upon how well the mapping is done.**

# Underlying Ideas

## **Make the same code work on different devices**

- A way to let compiler know that "this" expression can be specialized in many ways
- Definition of specializations

## **Template meta-programming in abstraction layers**

# Underlying Ideas

## Make the same code work on different devices

- A way to let compiler know that "this" expression can be specialized in many ways
- Definition of specializations

**Template meta-programming in abstraction layers**

## Assigning work within the node

- "Parallel For" or directives with unified memory
- Directives or specific programming model for explicit data movement

**More complex data orchestration system for asynchronous computation**



# Underlying Ideas

## Make the same code work on different devices

- A way to let compiler know that "this" expression can be specialized in many ways
- Definition of specializations

## Template meta-programming in abstraction layers

Look at what is needed, design for commonalities.

## Assigning work within the node

- "Parallel For" or directives with unified memory
- Directives or specific programming model for explicit data movement

More complex data orchestration system for asynchronous computation

# Underlying Ideas

## Make the same code work on different devices

- A way to let compiler know that "this" expression can be specialized in many ways
- Definition of specializations

## Template meta-programming in abstraction layers

Look at what is needed, design for commonalities.

Even when using third party abstraction tools understanding the code's structure and needs is critical for performance portability

## Assigning work within the node

- "Parallel For" or directives with unified memory
- Directives or specific programming model for explicit data movement

More complex data orchestration system for asynchronous computation

# Underlying Ideas

## Make the same code work on different devices

- A way to let compiler know that "this" expression can be specialized in many ways
- Definition of specializations

## Template meta-programming in abstraction layers

Look at what is needed, design for commonalities.

Even when using third party abstraction tools  
understanding the code's structure and needs is  
critical for performance portability  
... that translates to investing in design

## Assigning work within the node

- "Parallel For" or directives with unified memory
- Directives or specific programming model for explicit data movement

More complex data  
orchestration system for  
asynchronous  
computation

# Final takeaways

- The key to both performance portability and longevity is careful software design
- Extensibility should be built into the design
- Design should be independent of any specific programming model
- Composability and flexibility help with performance portability
- Resources:
  - <https://www.exascaleproject.org/>
  - <https://doi.org/10.6084/m9.figshare.13283714.v1>
  - [https://figshare.com/articles/presentation/SC20\\_Tutorial\\_Better\\_Scientific\\_Software/12994376?file=25219346](https://figshare.com/articles/presentation/SC20_Tutorial_Better_Scientific_Software/12994376?file=25219346)
  - [https://bssw.io/blog\\_posts/performance-portability-and-the-exascale-computing-project](https://bssw.io/blog_posts/performance-portability-and-the-exascale-computing-project)
  - <https://www.exascaleproject.org/event/kokkos-class-series>