Software Design Insights for Longevity of Scientific software

EXASCALE COMPUTING PROJECT

IDEAS productivity

better scientific software

Anshu Dubey

Argonne National Laboratory

Presentation to Middleware and Grid Interagency Coordination Team

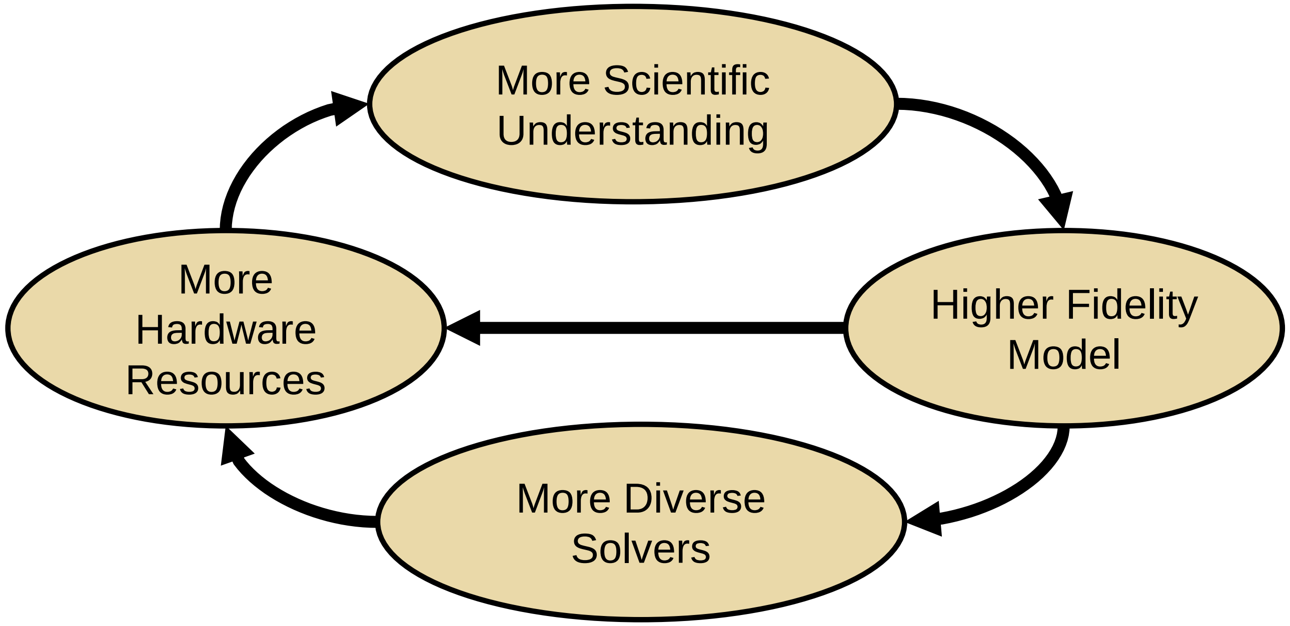
September 1, 2021

Acknowledgements

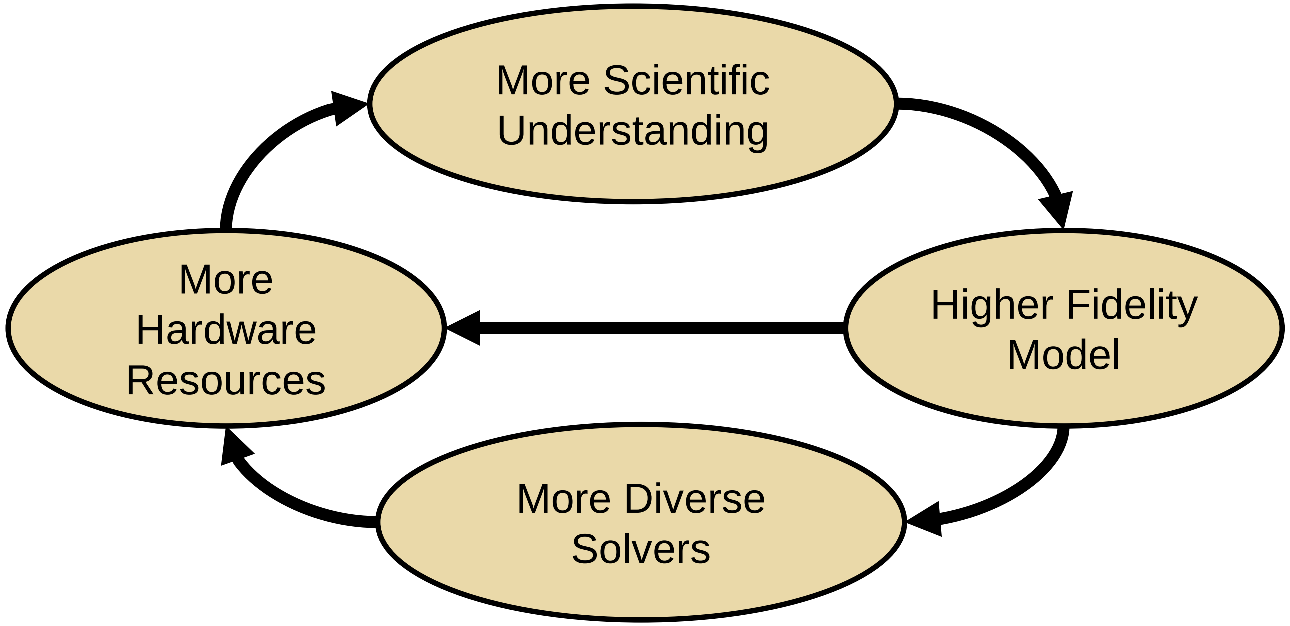
- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

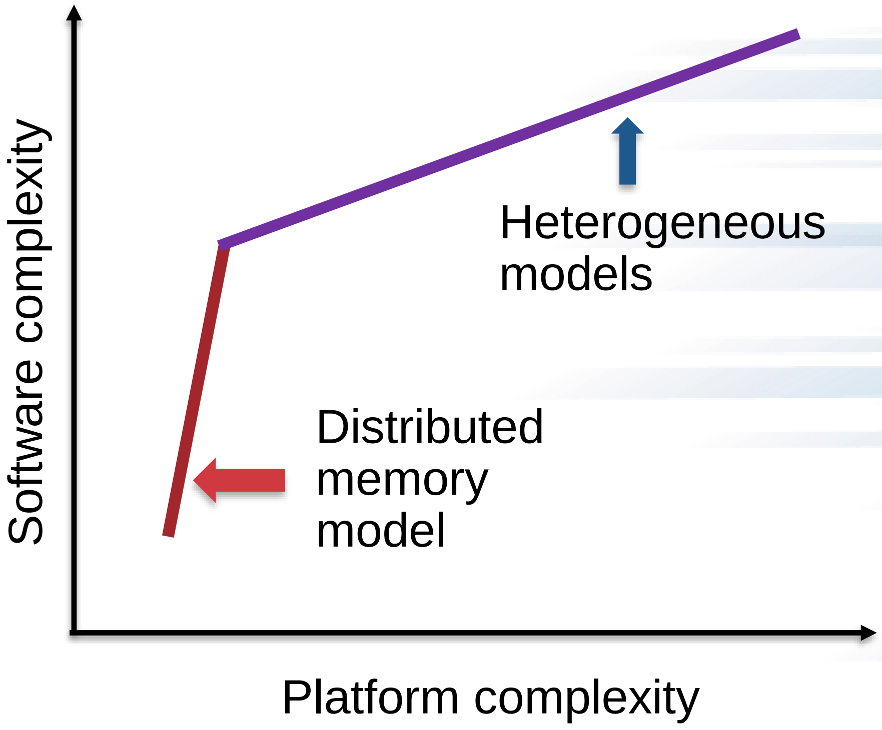
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.

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

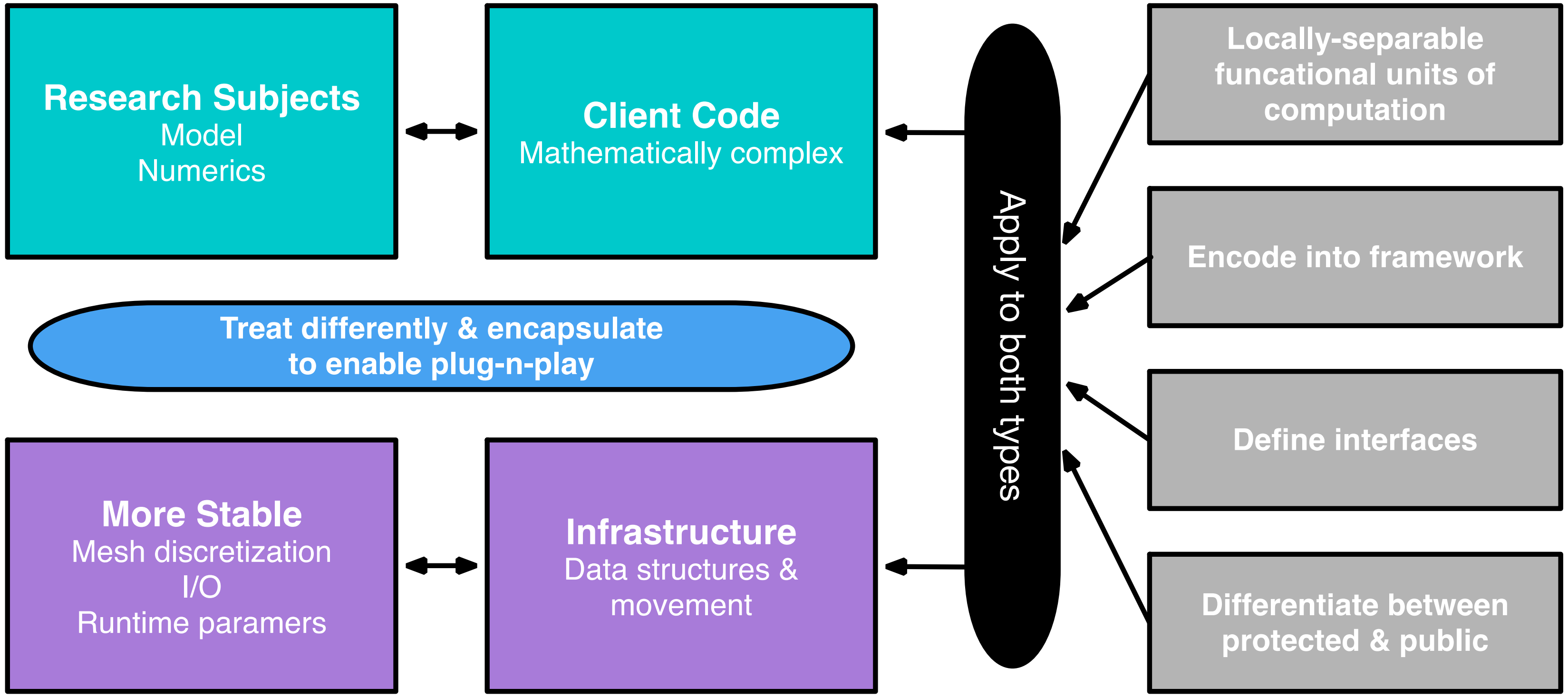
Considerations

- ❑ 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

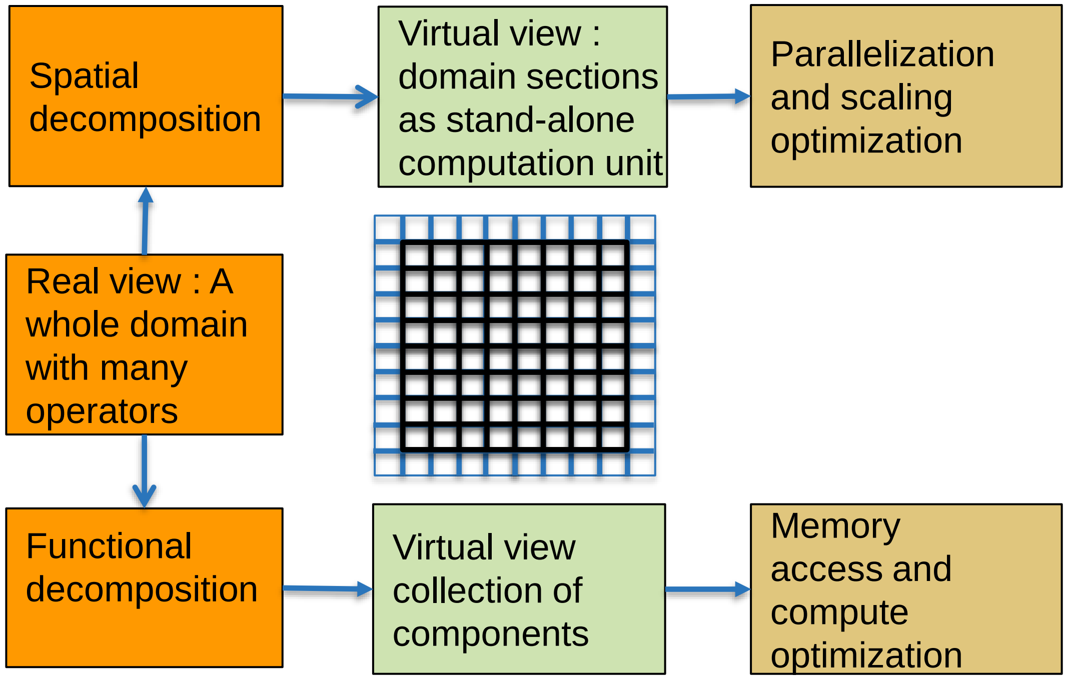
General Design Principles for HPC Scientific Software



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

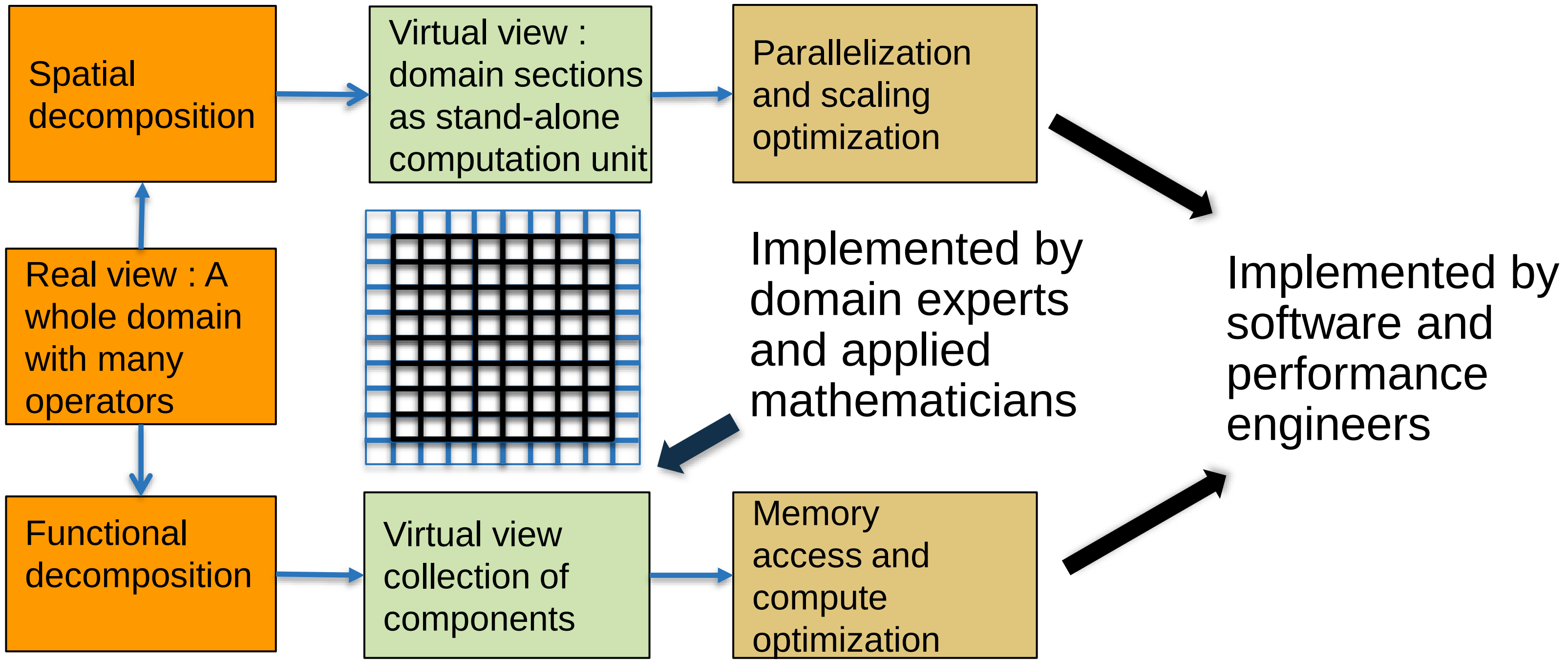
Example: Multiphysics PDEs for Distributed Memory Parallelism

- Virtual view of domain and functionalities
- Decomposition into components and definition of interfaces

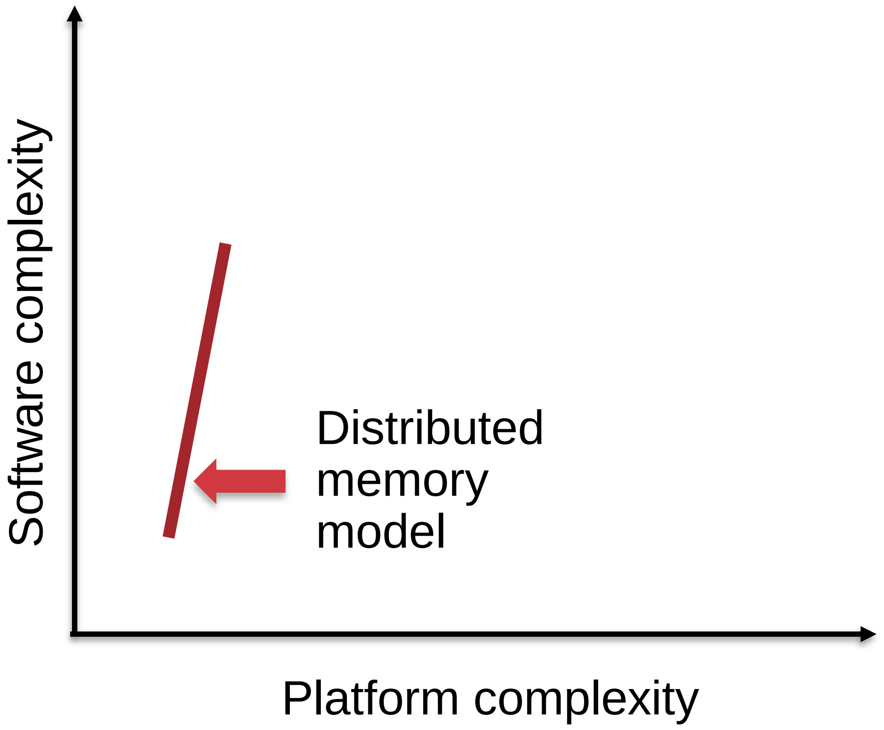


Example: Multiphysics PDEs for Distributed Memory Parallelism

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

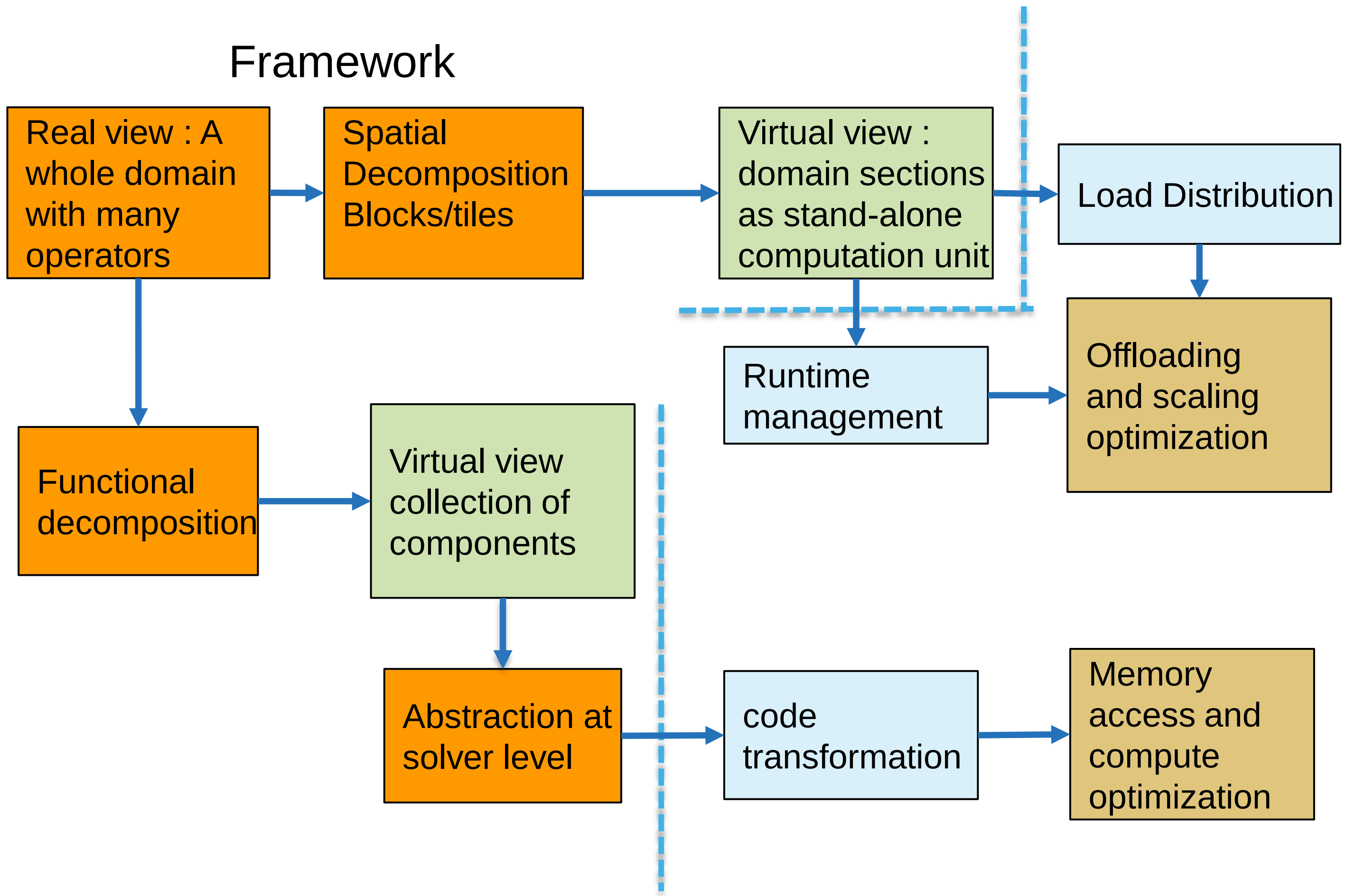


Takeaways Until Now



- 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

Features and Abstractions that must Come in



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

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

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.

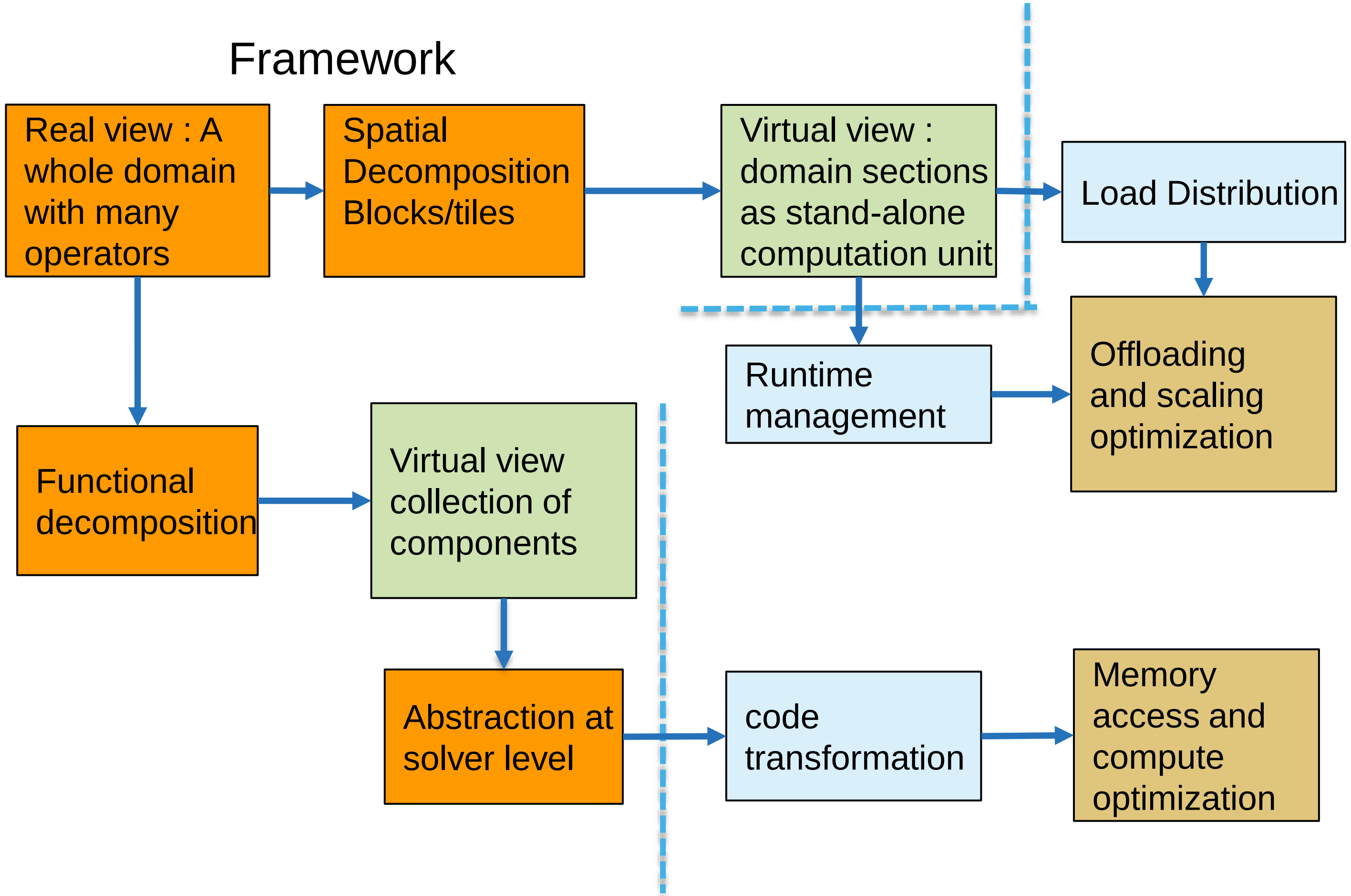
Even when using third party abstraction tools understanding the code’s structure and needs is critical for performance portability
… that translates to investing in design

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

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.

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://bssw.io/blog_posts/performance-portability-and-the-exascale-computing-project

https://www.exascaleproject.org/event/kokkos-class-series

"Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program."

The Networking and Information Technology Research and Development (NITRD) Program

Mailing Address: NCO/NITRD, 2415 Eisenhower Avenue, Alexandria, VA 22314

Physical Address: 490 L'Enfant Plaza SW, Suite 8001, Washington, DC 20024, USA Tel: 202-459-9674, Fax: 202-459-9673, Email: nco@nitrd.gov, Website: https://www.nitrd.gov