Scientific Software Design

Anshu Dubey (she/her)
Argonne National Laboratory

Software Productivity and Sustainability track @ Argonne Training Program on Extreme-Scale Computing summer school

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

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Todd Gamblin, Jared O'Neal, and Boyana R. Norris, Software Productivity and Sustainability track, in Argonne Training Program on Extreme-Scale Computing, St. Charles, Illinois, 2022. DOI: 10.6084/m9.figshare.20416215.**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial, ISC, 2022 …

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 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 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 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.
- This work was performed in part at University of Oregon through a subcontract with Argonne National Laboratory.

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

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

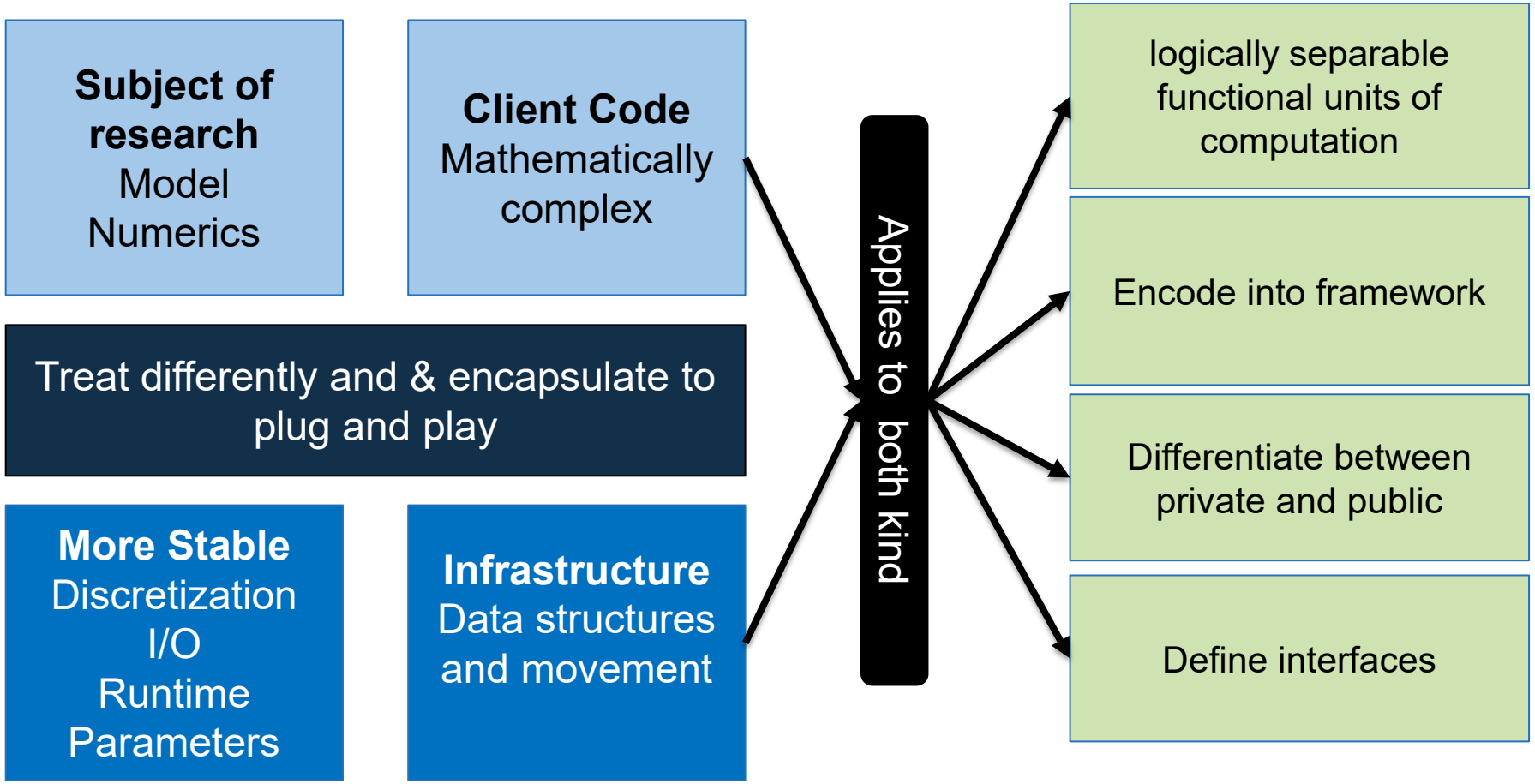
Design Implications

- 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

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

General Design Principles for HPC Scientific Software



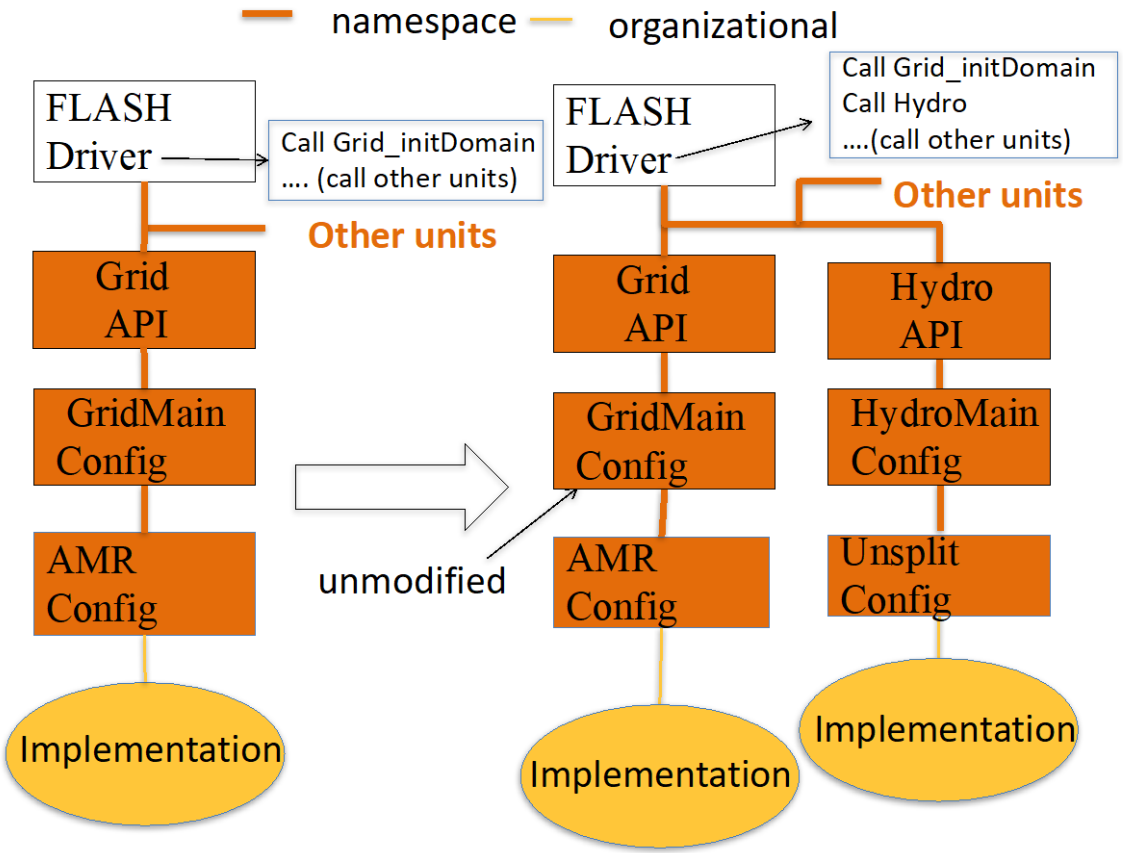
Design first, then apply programming model to the design instead of taking a programming model and fitting your design to it.

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

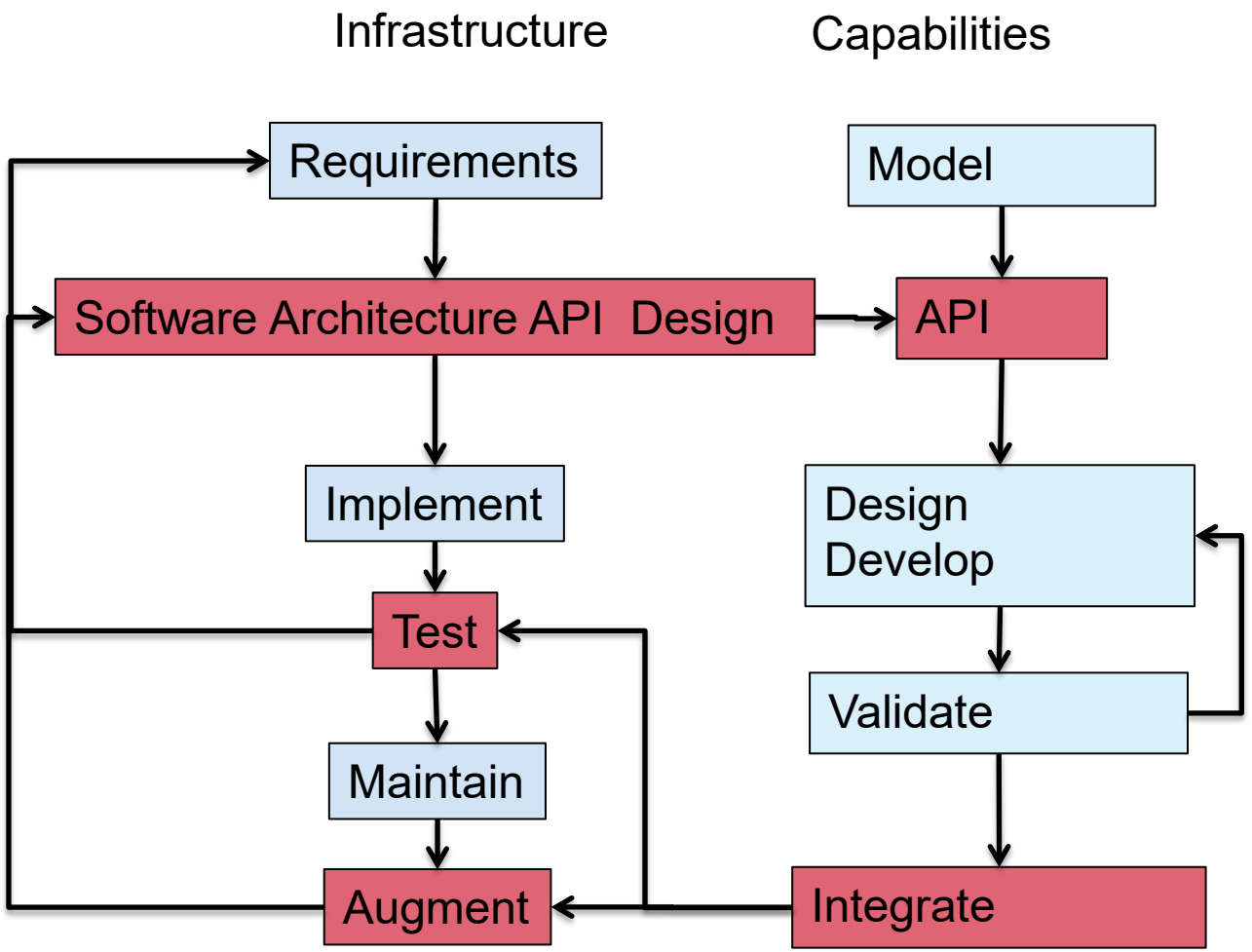
Assumed that capabilities will be added for better models

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

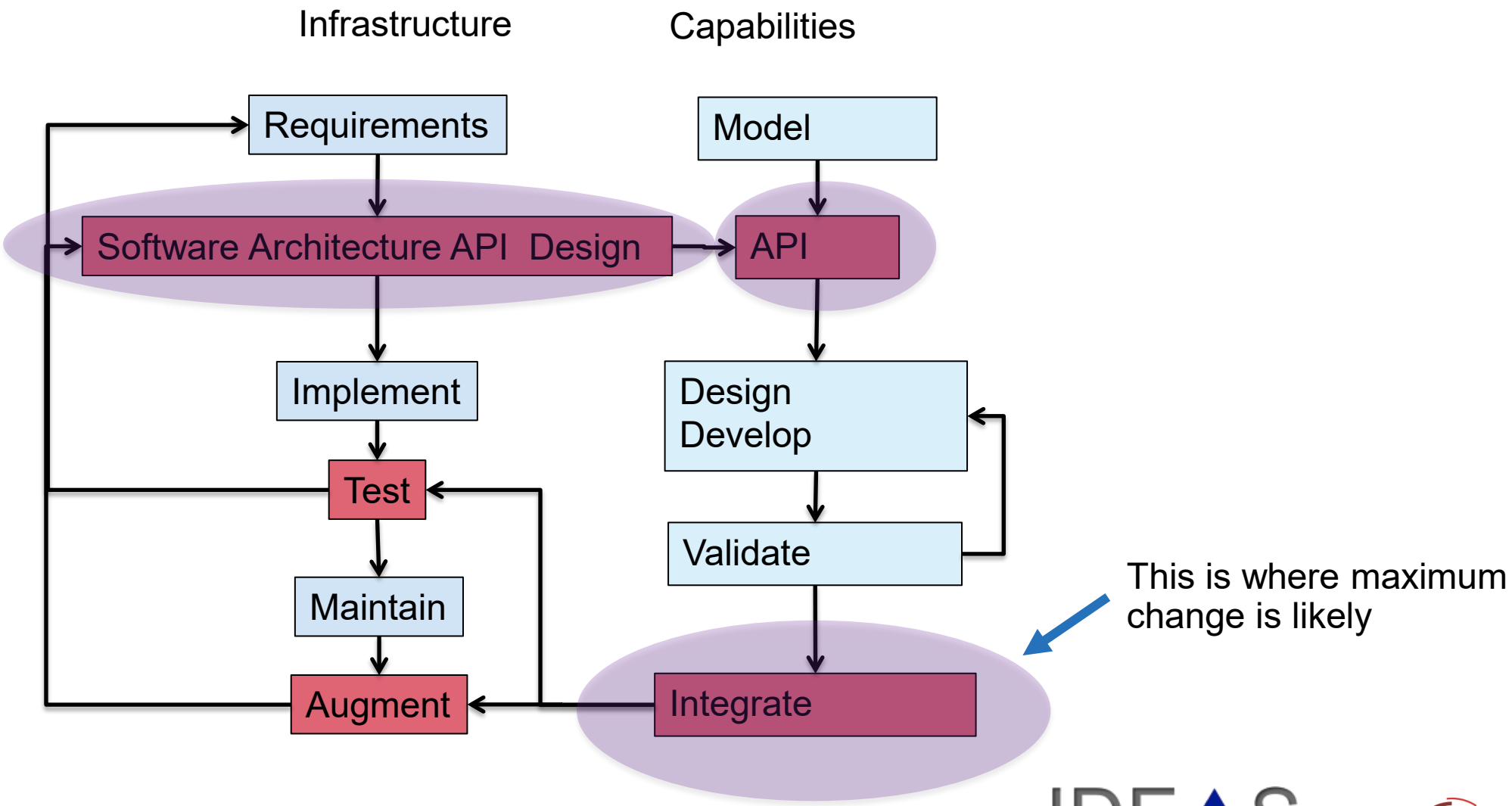
Key idea is distributed intelligence



A Design Model for Separation of Concerns



Handling Heterogeneity – Hardware and Software



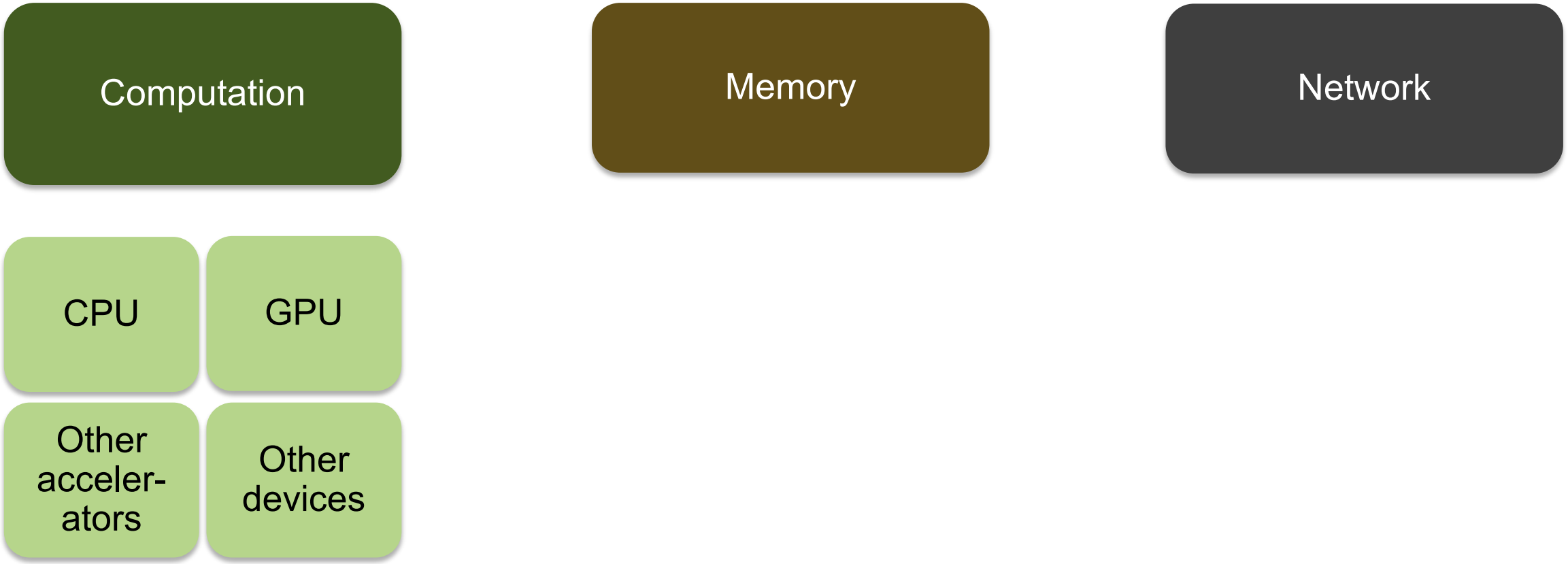
Platform Heterogeneity

Computation

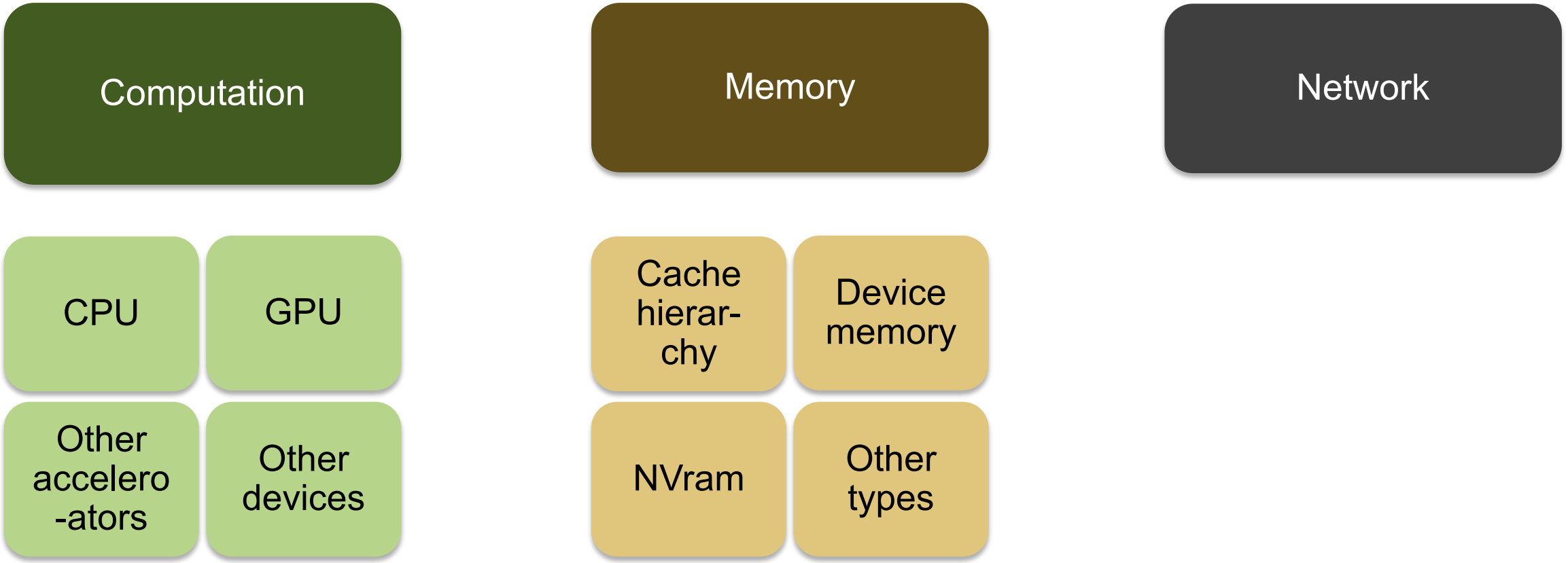
Memory

Network

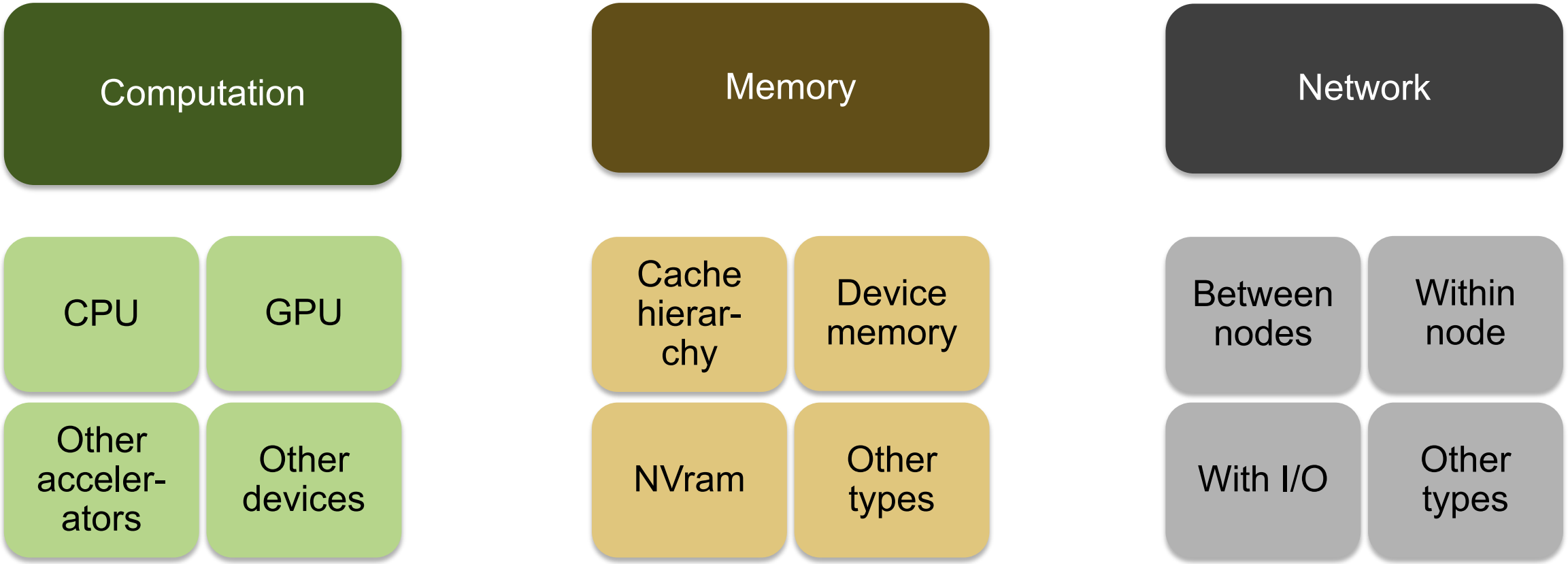
Platform Heterogeneity



Platform Heterogeneity



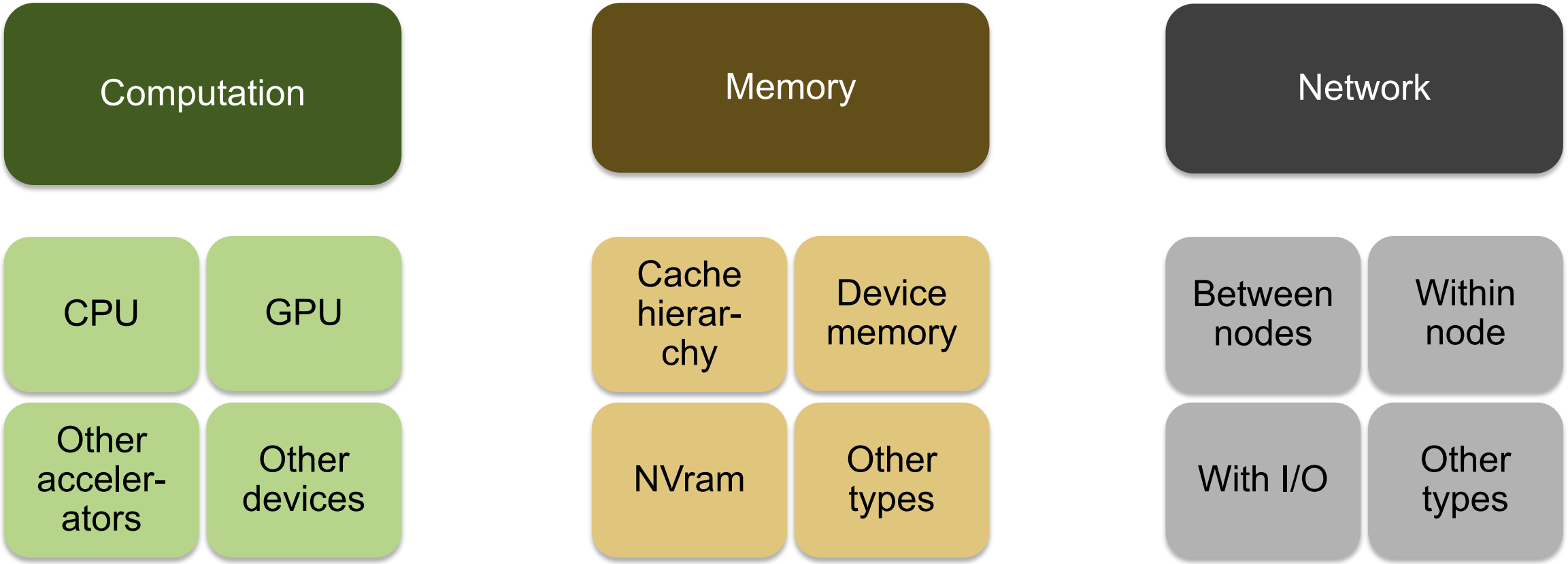
Platform Heterogeneity



IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Platform Heterogeneity



And memory access models: unified memory / gpu-direct / explicit transfer

Mechanisms Needed by the Code

Mechanisms to unify expression of computation

- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Mechanisms Needed by the Code

Mechanisms to unify expression of computation

- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Mechanisms to move work and data to computational targets

- Moving between devices
  - Launching work at the destination
  - Hiding latency of movement
- Moving data offnode

Mechanisms Needed by the Code

Mechanisms to unify expression of computation

- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Mechanisms to move work and data to computational targets

- Moving between devices
  - Launching work at the destination
  - Hiding latency of movement
- Moving data offnode

Mechanisms to map work to computational targets

- Figuring out the map
  - Expression of dependencies
  - Cost models
- Expressing the map

Mechanisms Needed by the Code

Mechanisms to unify expression of computation
- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Mechanisms to move work and data to computational targets
- Moving between devices
  - Launching work at the destination
  - Hiding latency of movement
- Moving data offnode

Mechanisms to map work to computational targets
- Figuring out the map
  - Expression of dependencies
  - Cost models
- Expressing the map

So what do we need?

- Abstractions layers
- Code transformation tools
- Data movement orchestrators

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Underlying Ideas: Unification of Computational Expressions

Make the same code work on different devices

Same algorithm different data layouts or operation sequence:

- A way to let compiler know that ”this” expression can be specialized in many ways
- Definition of specializations
- Often done with template meta-programming

More challenging if algorithms need to be fundamentally different

- Support for alternatives

Underlying Ideas: Moving Work and Data to the Target

Parallelization Models

Hierarchy in domain decomposition

- Distributed memory model at node level – still very prevalent, likely to remain so for a while
- Also done with PGAS models – shared with locality being important

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

- Task based work distribution

Underlying Ideas: Mapping Work to Targets

This is how many 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

It can also be the end user who figures out the mapping
In either case performance depends upon how well the mapping is done

Mechanisms Needed by the Code : Example Flash-X

Mechanisms to unify expression of computation

Macros with inheritance

Mechanisms Needed by the Code : Example Flash-X

Mechanisms to unify expression of computation

Macros with inheritance

Mechanisms to move work and data to computational targets

Domain specific runtime

Mechanisms Needed by the Code : Example Flash-X

Mechanisms to unify expression of computation

Macros with inheritance

Mechanisms to move work and data to computational targets

Domain specific runtime

Mechanisms to map work to computational targets

DSL for recipes with code generator

Mechanisms Needed by the Code : Example Flash-X

Mechanisms to unify expression of computation

Macros with inheritance

Mechanisms to move work and data to computational targets

Domain specific runtime

Mechanisms to map work to computational targets

DSL for recipes with code generator

Composability in the source
A toolset of each mechanism
Independent tool sets

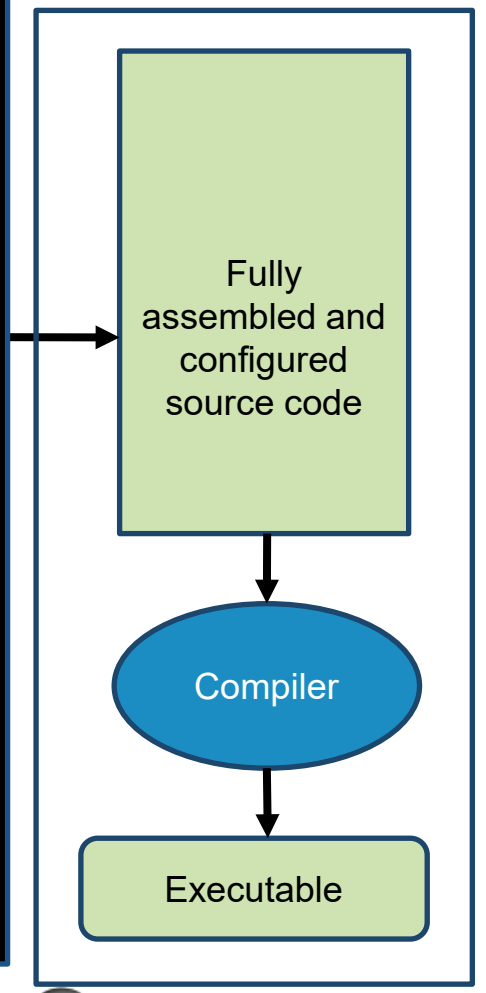
IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Overview of Flash-X Design Approach with Separation of Concerns in tools

Design Philosophy

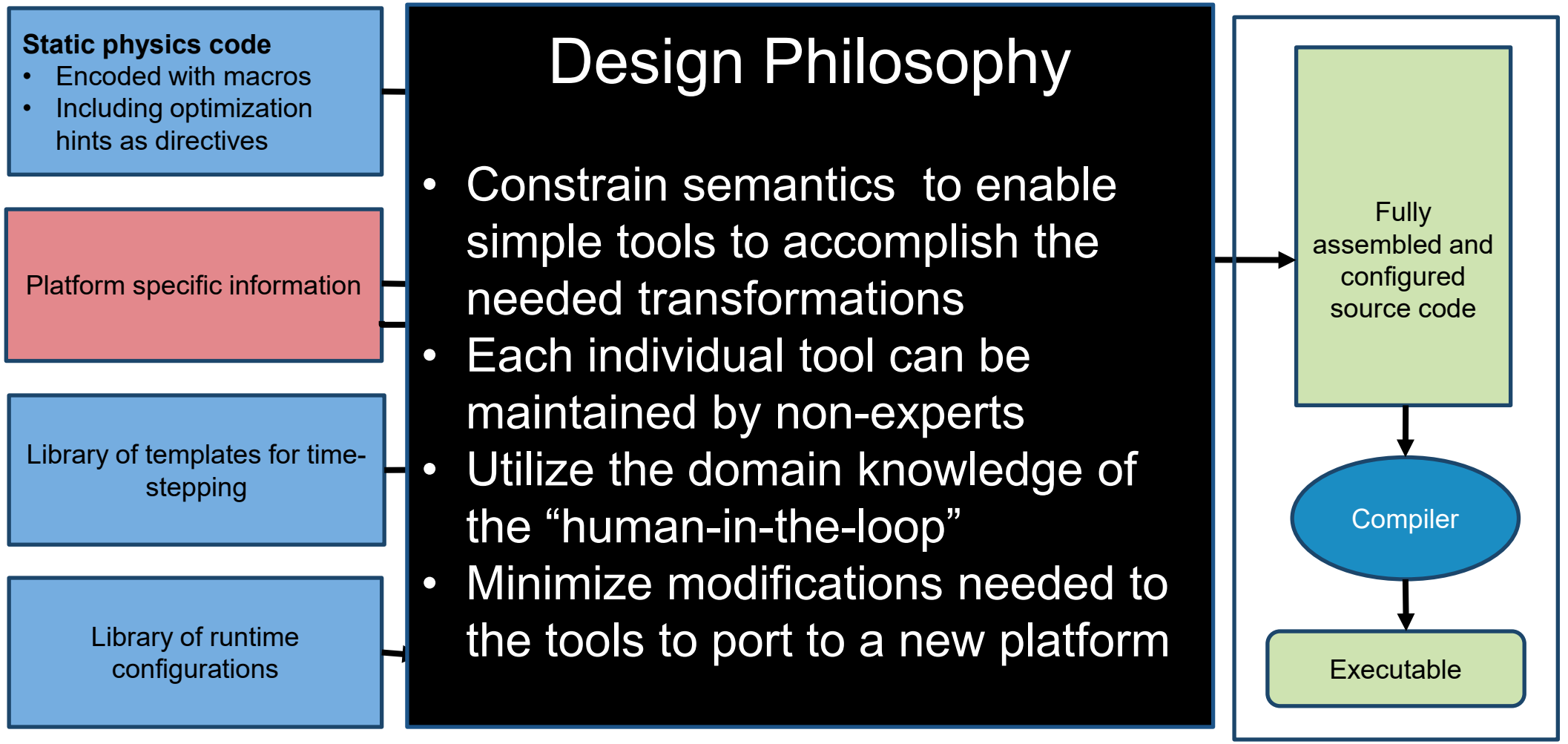
- Constrain semantics to enable simple tools to accomplish the needed transformations
- Each individual tool can be maintained by non-experts
- Utilize the domain knowledge of the “human-in-the-loop”
- Minimize modifications needed to the tools to port to a new platform



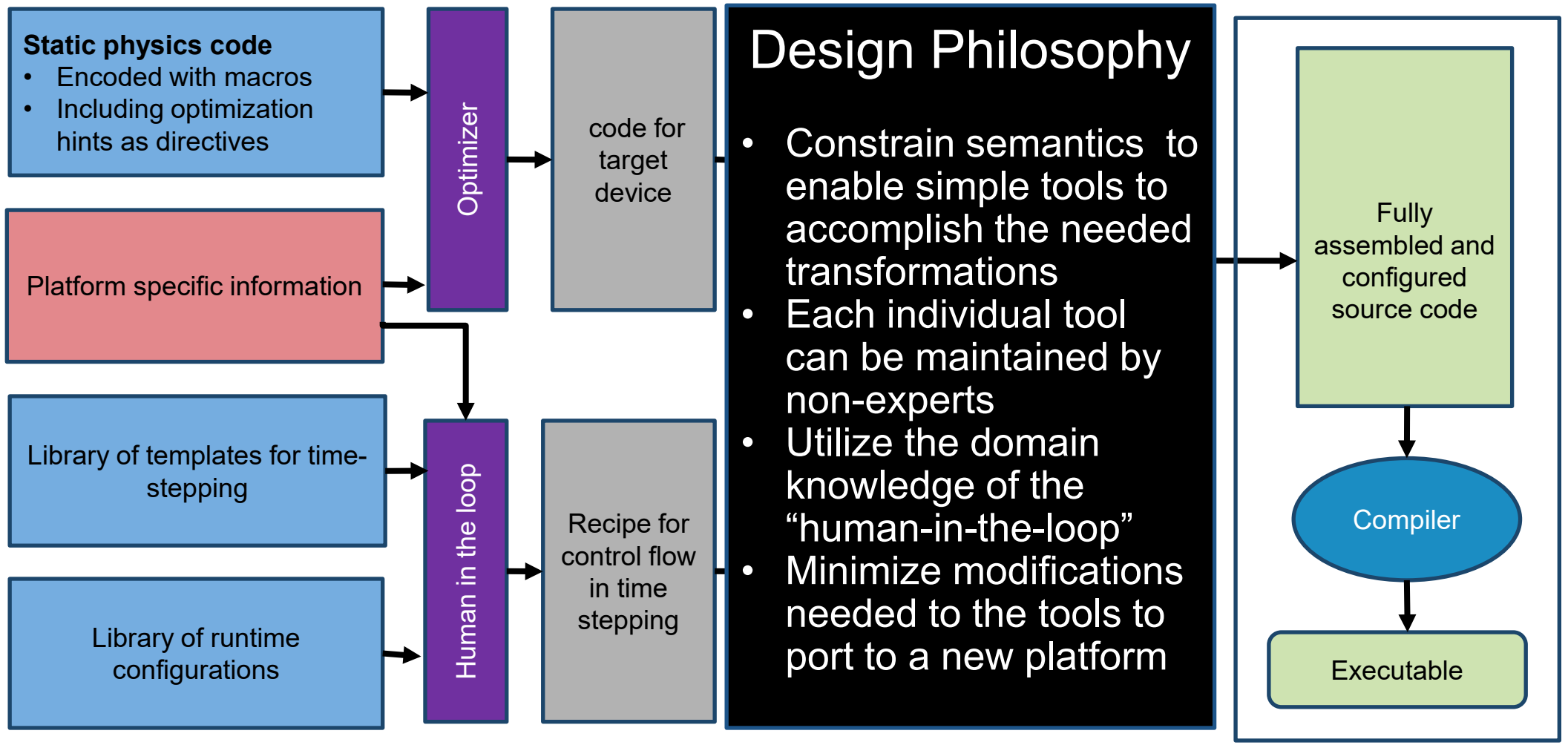
IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

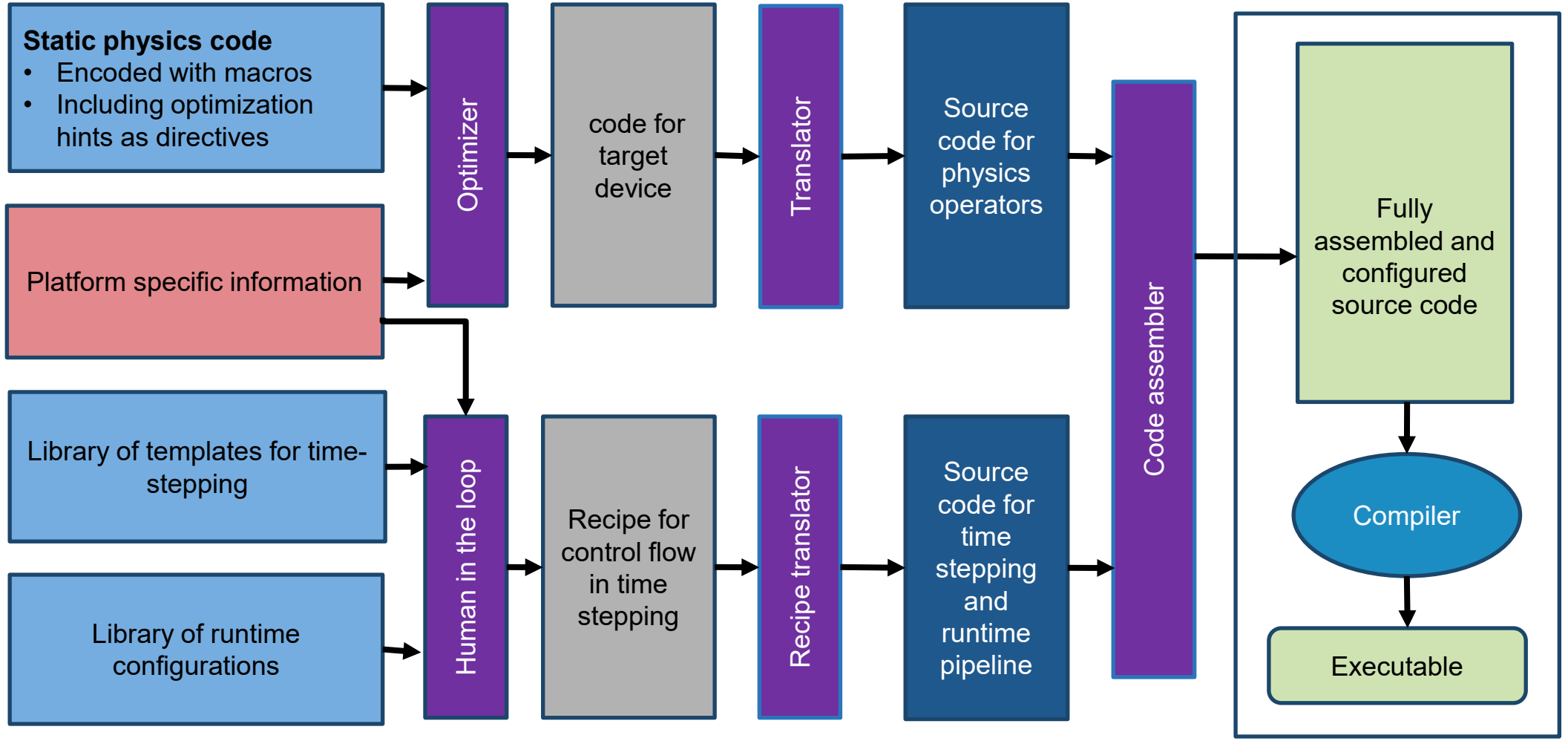
Overview of Flash-X Design Approach with Separation of Concerns in tools



Overview of Flash-X Design Approach with Separation of Concerns in tools



Overview of Flash-X Design Approach with Separation of Concerns in tools



Other Rules of Thumb

- 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

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://bssw.io/blog_posts/performance-portability-and-the-exascale-computing-project
  - https://www.exascaleproject.org/event/kokkos-class-series
  - A Design Proposal for a Next Generation Scientific Software Framework
  - Software Design for Longevity with Performance Portability