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 E. Bernholdt (ORNL)

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is:** Anshu Dubey, David E. Bernholdt, Greg Becker, and Jared O'Neal, Software Productivity and Sustainability track, in Argonne Training Program on Extreme-Scale Computing, St. Charles, Illinois, 2023. DOI: 10.6084/m9.figshare.23823822.
- Individual modules may be cited as *Speaker, Module Title*, in *Tutorial Title*, …

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.

Introduction

- Investing some thought in design of software makes it possible to maintain, reuse and extend it
- Even if some research software begins its life as a one-off use case, it often gets reused
  - Without proper design it is likely to accrete features haphazardly and become a monstrosity
    - Acquires a lot of technical debt in the process
  - Many projects have had this happen
  - Most end up with a hard reset and start over again
- In this module we will cover general design principles and those that are tailored for scientific software
- We will also work through two use cases

General Design Principles for Maintainable Software

Some definitions from the web

- ❑ Encapsulate what varies
- ❑ Favor composition over inheritance
- ❑ Program to interfaces not implementations
- ❑ Loose coupling – interacting components should have minimal knowledge about each other
- ❑ SOLID

https://bootcamp.uxdesign.cc/software-design-principles-every-developers-should-know-23d24735518e

General Design Principles for Maintainable Software

Found on the web

- ❑ Encapsulate what varies
- ❑ Favor composition over inheritance
- ❑ Program to interfaces not implementations
- ❑ Loose coupling – interacting components should have minimal knowledge about each other
- ❑ SOLID

https://bootcamp.uxdesign.cc/software-design-principles-every-developers-should-know-23d24735518e

SOLID

- ❑ Single responsibility
  - ❑ Class/method/function should do only one thing
- ❑ Open/closed
  - ❑ Open for extension, close for modification
- ❑ Liskov substitution
  - ❑ Implementations of an interface should give same result
- ❑ Interface segregation
  - ❑ Client should not have to use methods it does not need
- ❑ Dependency inversion
  - ❑ High level modules should not depend on low level modules, only on abstractions

IDEAS productivity   ECP EXASCALE COMPUTING PROJECT

Designing Software – High Level Phases

Requirements gathering

- ❑ Features and capabilities
- ❑ Constraints
- ❑ Limitations
- ❑ Target users
- ❑ Other …..

Decomposition

- ❑ Understand design space
- ❑ Decompose into high level components
- ❑ Bin components into types

Connectivity

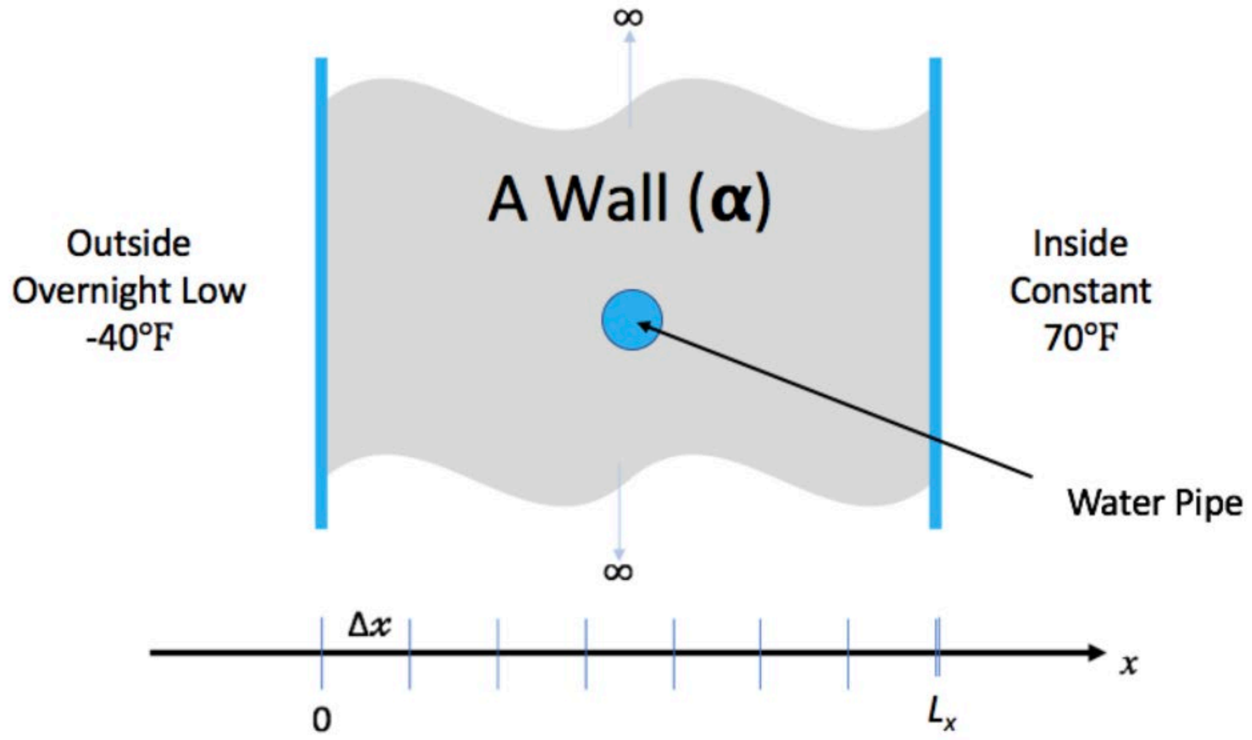
- ❑ Understand component hierarchy
- ❑ Figure out connectivity among components
- ❑ Articulate dependencies

Example 1 – Problem Description

We have a house with exterior walls made of single material of thickness L_x
The wall has some water pipes shown in the picture.

The inside temperature is kept at 70 degrees. But outside temperature is expected to be -40 degrees for 15.5 hours.

Will the pipes freeze before the storm is over



IDEAS productivity ECP EXASCALE COMPUTING PROJECT

Mathematical formulation

- Heat conduction is governed by a partial differential equation

$$\frac{\partial u}{\partial t} - \nabla \cdot \alpha \nabla u = 0$$	(1)

- We make some simplifying assumptions
  - The thermal diffusivity is constant for all space and time.
  - The only heat source is from the initial and/or boundary conditions.
  - We will deal only with the one dimensional problem in Cartesian coordinates.
  - That reduces the heat equation to

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$	(2)

The repository has solutions using three numerical methods

- Foward Time Centered Space (FTCS), an explicit method
- Crank-Nicholson, an implicit method
- Upwind-15, another explicit method with higher spatial order than FTCS.

We will use FTCS for this exercise

Requirements gathering

- To solve heat equation we need:
  - a discretization scheme
  - a driver for running and book-keeping
  - an integration method to evolve solution
  - Initial conditions
  - Boundary conditions
- To make sure that we are doing it correctly we need:
  - Ways to inspect the results
  - Ways of verification

Decomposition

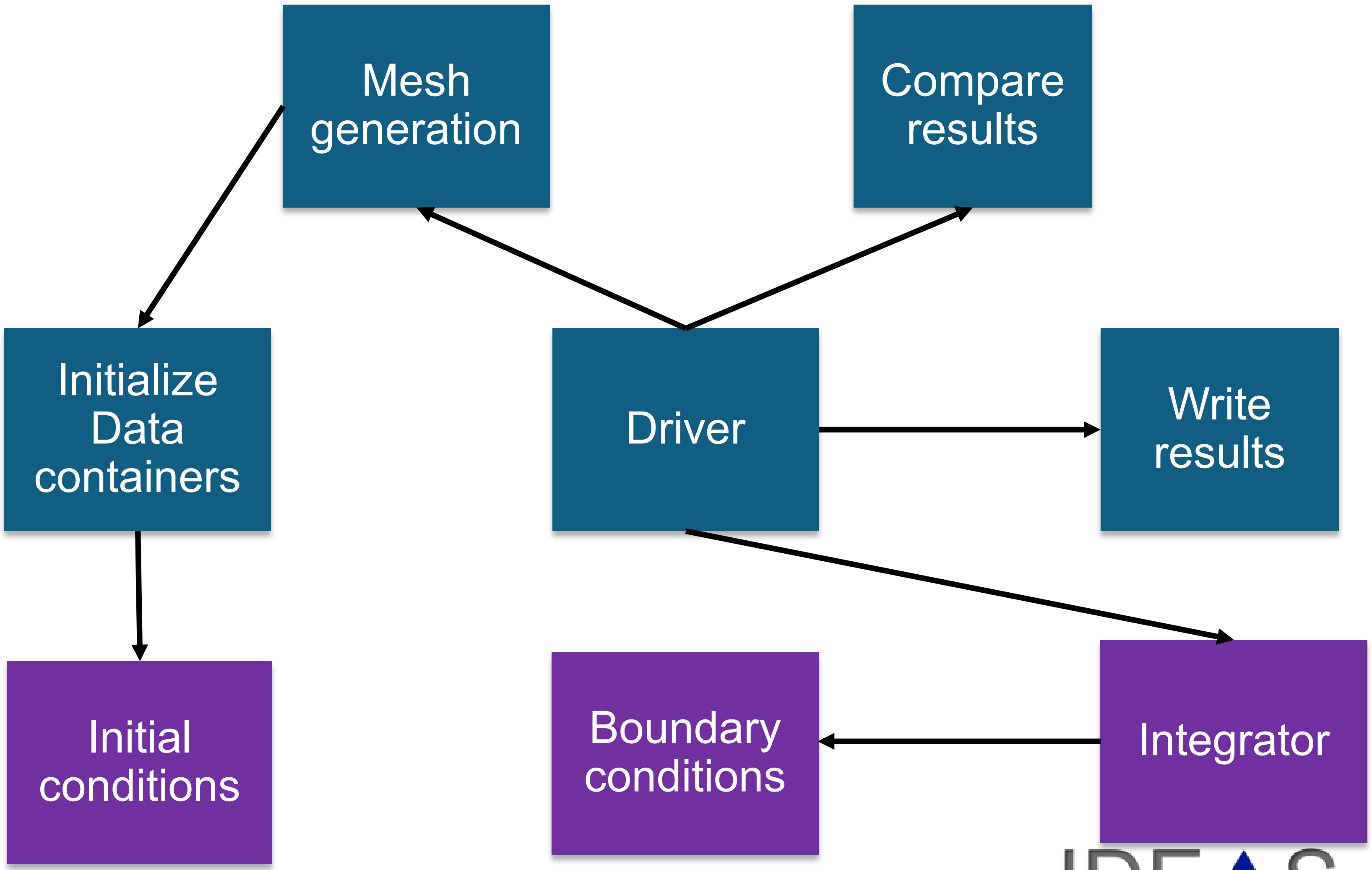
This is a small design space

- Several requirements can directly map to components – in this instance functions
  - Driver
  - Initialization – data containers
  - Mesh initialization – applying initial conditions
  - Integrator
  - I/O
  - Boundary conditions
  - Comparison utility

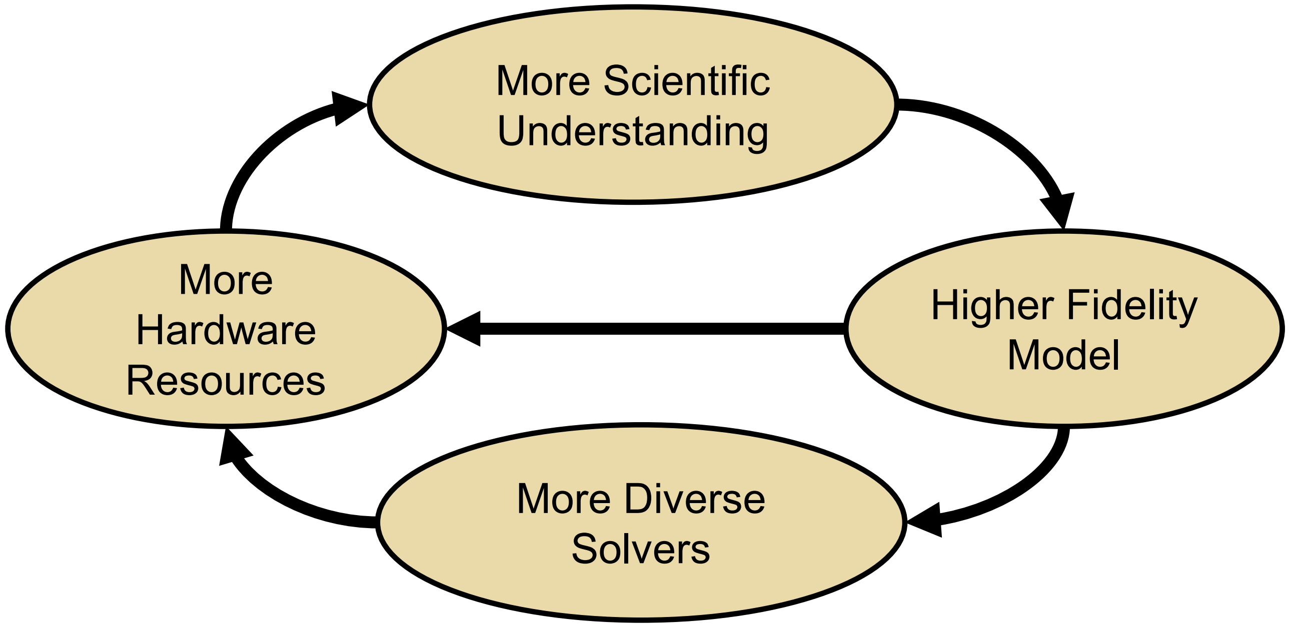
Binning components

- ❑ Components that will work for any application of heat equation
  - ❑ Driver
  - ❑ Initialization – data containers
  - ❑ I/O
  - ❑ Comparison utility
- ❑ Components that are
  - ❑ Mesh initialization – applying initial conditions
  - ❑ Integrator
  - ❑ Boundary conditions

Connectivity



Research Software Challenges



- Many parts of the model and software system can be under research
- Requirements change throughout the lifecycle as knowledge grows
- Verification complicated by floating point representation
- Real world is messy

SOLID Principles Pose Some Difficulties

- Single responsibility
  - Class/method/function should do only one thing
- Open/closed
  - Open for extension, close for modification
- Liskov substitution
  - Implementations of an interface should give same result
- Interface segregation
  - Client should not have to use methods it does not need
- Dependency inversion
  - High level modules should not depend on low level modules, only on abstractions

- Function calls have overheads
  - Performance matters – quick turnaround of results desirable
- New insights may cause modification
  - May lead to unmaintainable code duplication
- It is not always possible to eliminate lateral interactions
- Not always possible

IDEAS productivity
ECP EXASCALE COMPUTING PROJECT

Additional Considerations for Research Software

Considerations

- ❑ Multidisciplinary
  - ❑ 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

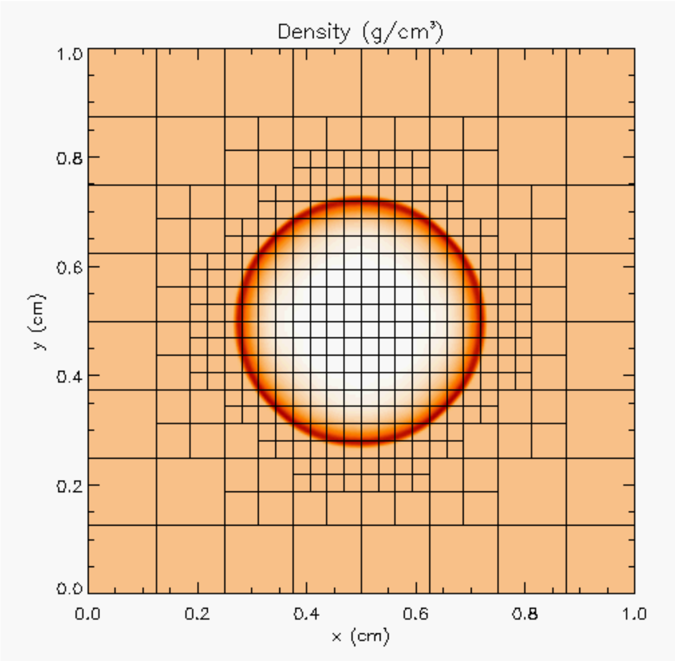
IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

More Complex Application Design – Sedov Blast Wave

Description

High pressure at the center cause a shock to moves out in a circle. High resolution is needed only at and near the shock



Requirements

- Adaptive mesh refinement
  - Easiest with finite volume methods
- Driver
- I/O
- Initial condition
- Boundary condition
- Shock Hydrodynamics
- Ideal gas equation of state
- Method of verification

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Deeper Dive into Requirements

- Adaptive mesh refinement => divide domain into blocks
  - Blocks need halos to be filled with values from neighbors or boundary conditions
    - At fine-coarse boundaries there is interpolation and restriction
  - Blocks are dynamic, go in and out of existence
  - Conservation needs reconciliation at fine-coarse boundaries
- Shock hydrodynamics
  - Solver for Euler's equations at discontinuities
  - EOS provides closure
  - Riemann solver
  - Halo cells are fine-coarse boundaries need EOS after interpolation
- Method of verification
  - An indirect way of checking – shock distance traveled can be computed analytically

Components

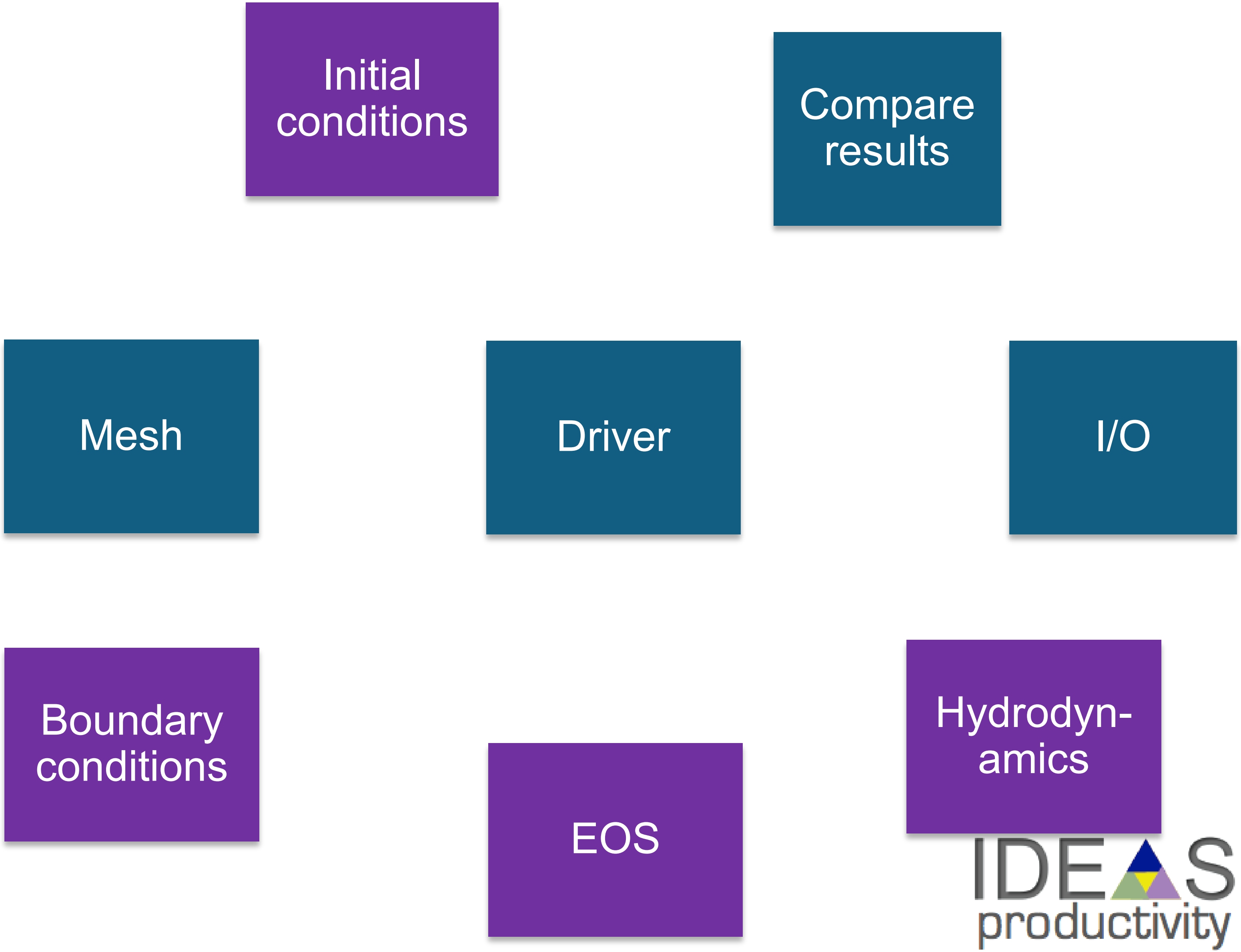
Binned Components

- Unchanging or slow changing infrastructure
  - Mesh
  - I/O
  - Driver
  - Comparison utility
- Components evolving with research – physics solvers
  - Initial and boundary conditions
  - Hydrodynamics
  - EOS

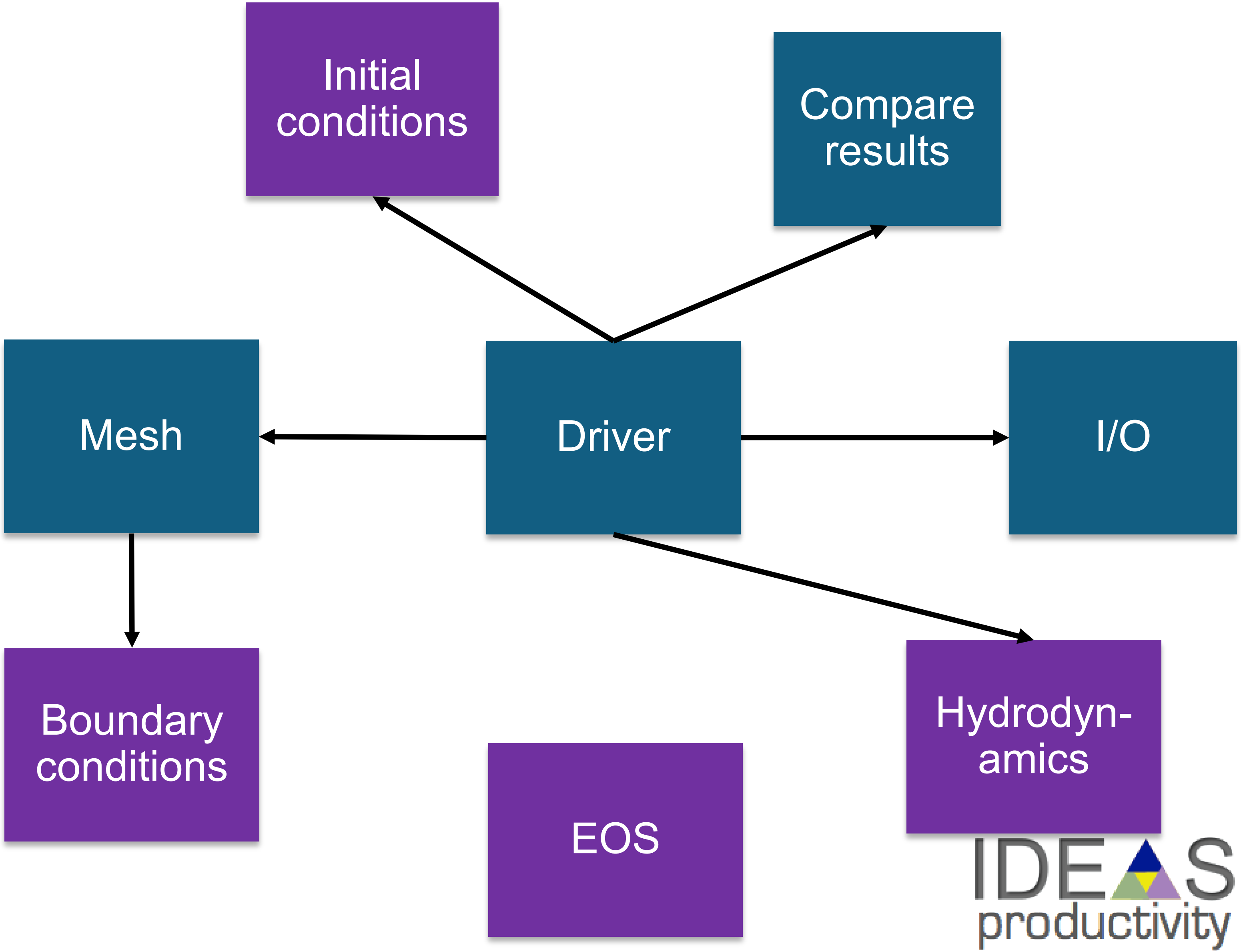
Deeper Dive into some Components

- Driver
  - Iterate over blocks
  - Implement connectivity
- Mesh
  - Data containers
  - Halo cell fill, including application of boundary conditions
  - Reconciliation of quantities at fine-coarse block boundaries
  - Remesh when refinement patterns change
- I/O
  - Getting runtime parameters and possibly initial conditions
  - Writing checkpoint and analysis data

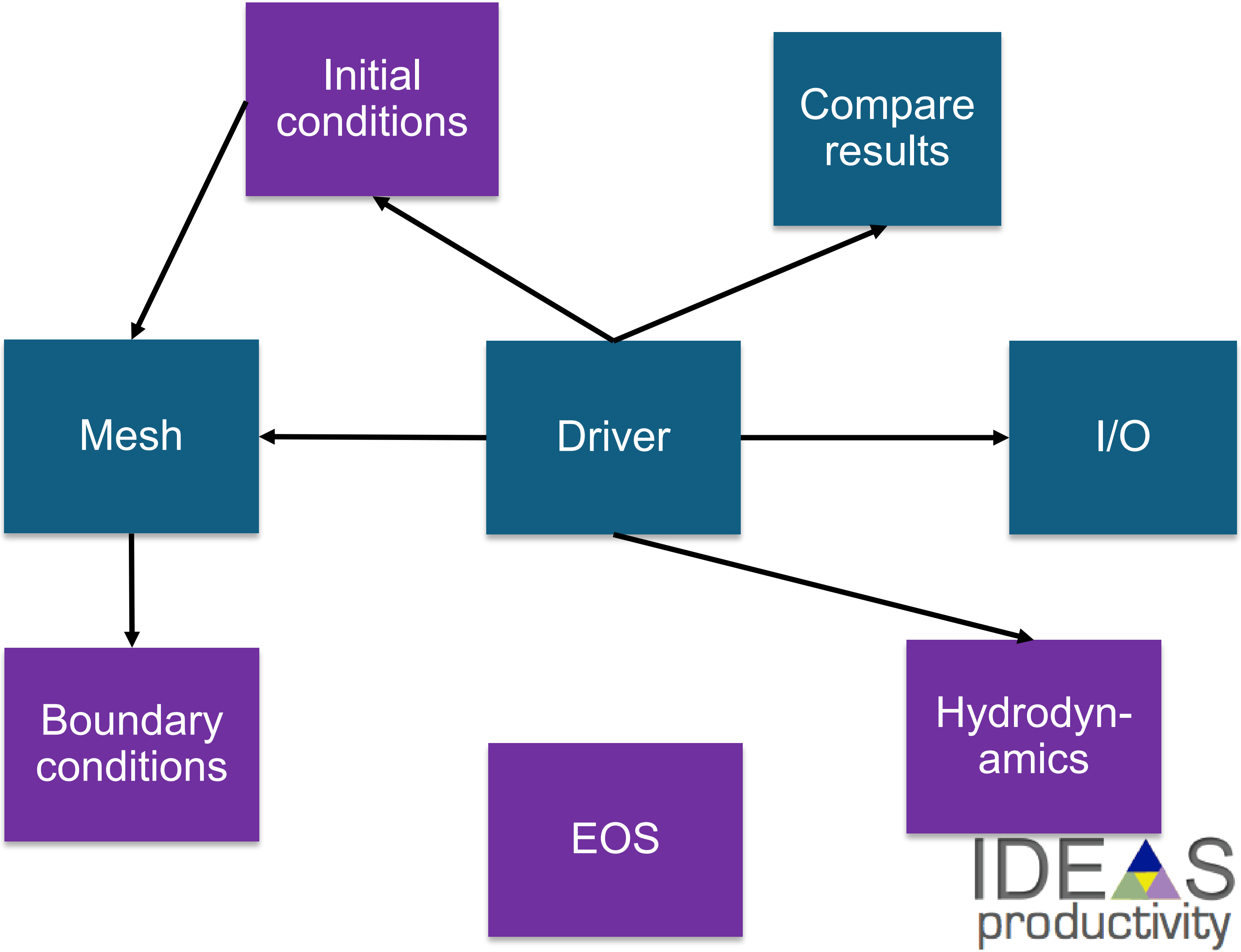
Connectivity



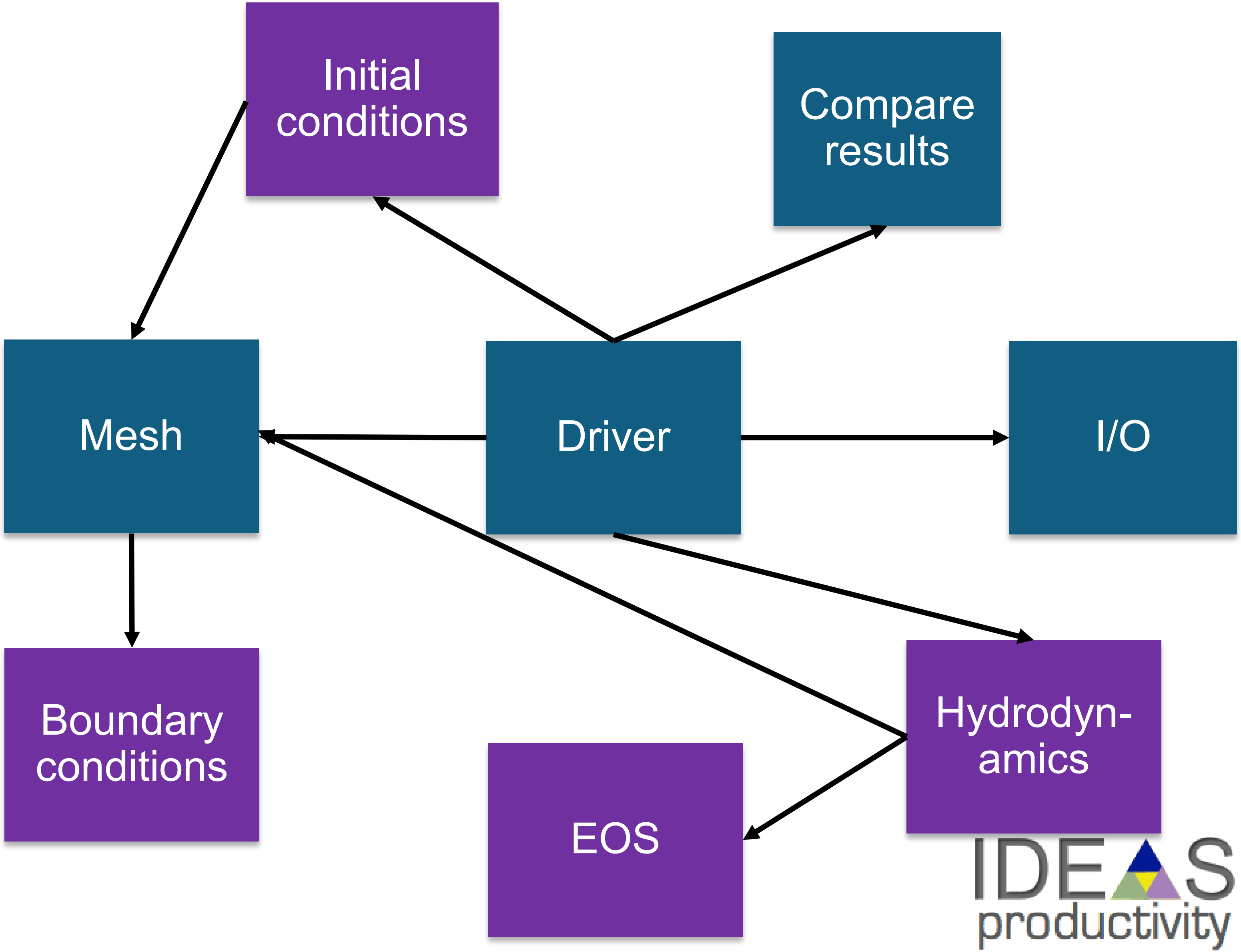
Connectivity



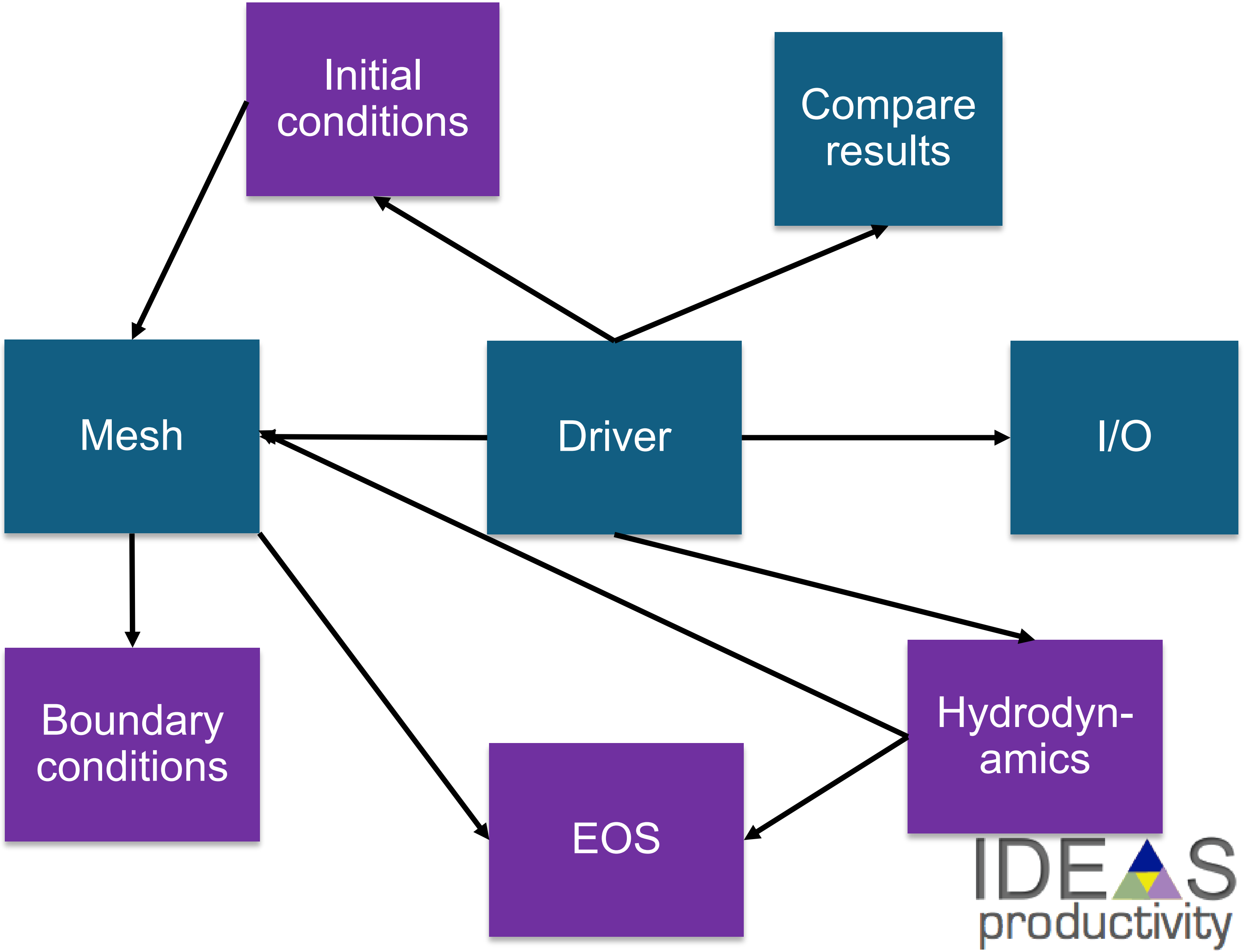
Connectivity



Connectivity

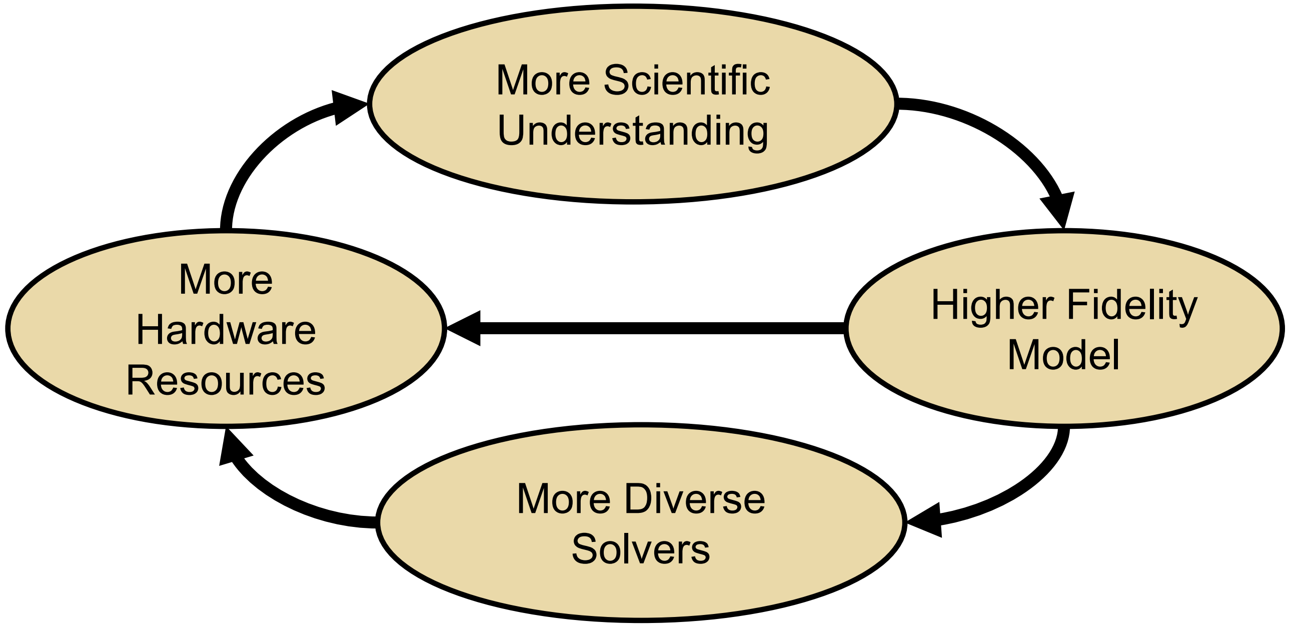


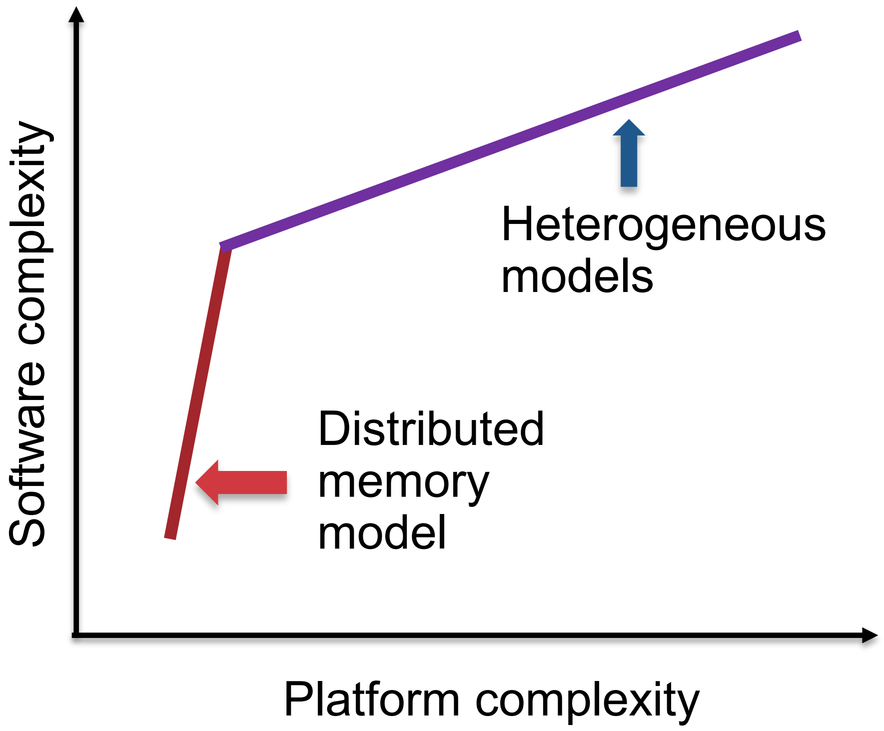
Connectivity



IDEAS productivity
ECP EXASCALE COMPUTING PROJECT

New Paradigm Because of Platform Heterogeneity





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 off node

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 off node

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 off node

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

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

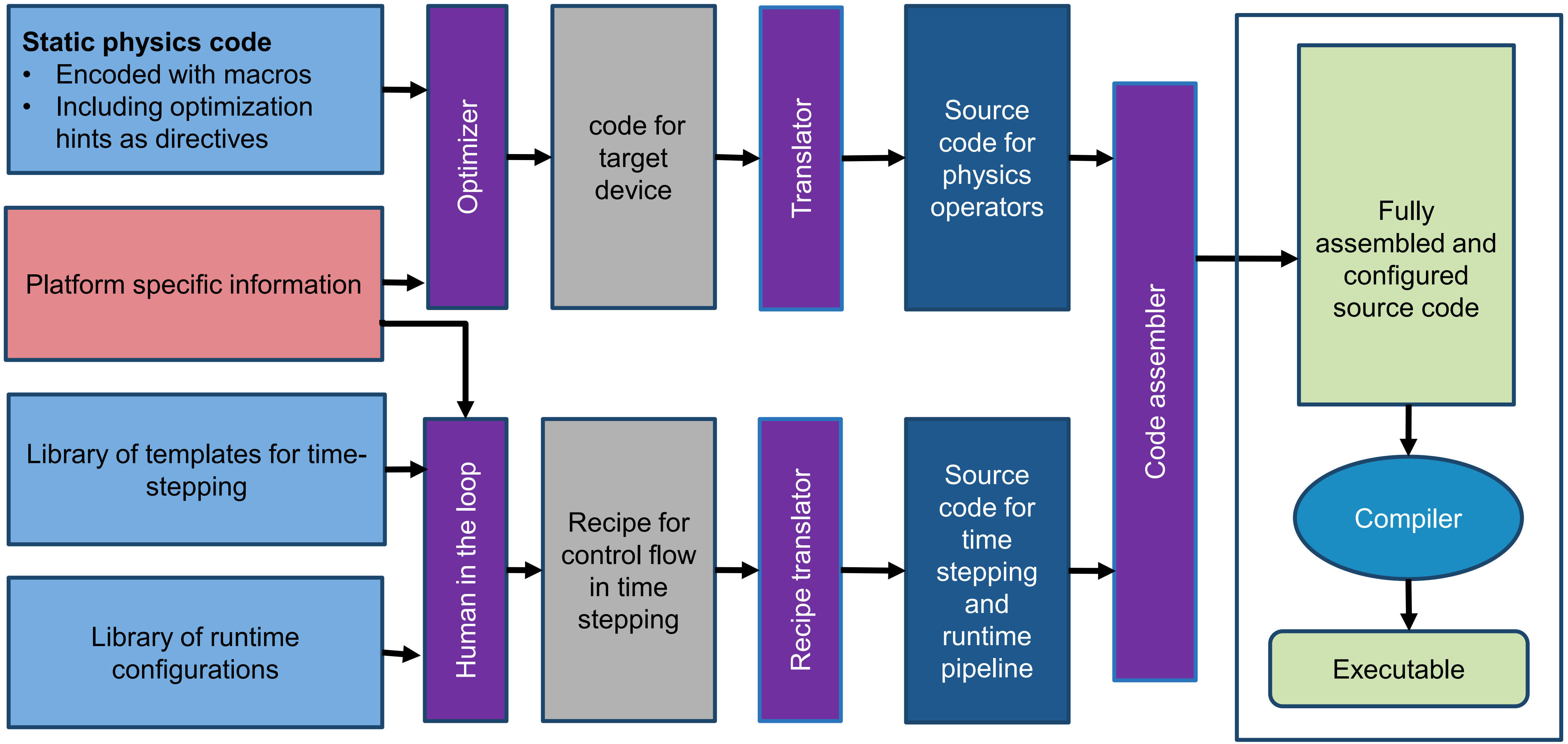
Domain specific runtime

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

Construction of Application with Components and Tools



Takeaways

- Requirements gathering and intentional design are indispensable for sustainable software development
- Many books and online resources available for good design principles
- Research software poses additional constraints on design because of its exploratory nature
  - Scientific research software has further challenges
  - High performance computing research software has even more challenges
  - That are further exacerbated by the ubiquity of accelerators in platforms
- Separation of concerns at various granularities, and abstractions enable sustainable software design