AN APPLICATION PERSPECTIVE ON PROGRAMMING MODELS FOR THE FUTURE

ANSHU DUBEY

PAW-ATM
SC18
November 16, 2018

SETTING THE STAGE

What should my ideal computational tool do?

Everything really.

- Scan my brain
- Figure out what I want
- Scan the literature
- Figure out the equations
- Auto-generate the code
- Run it
- Analyze the data

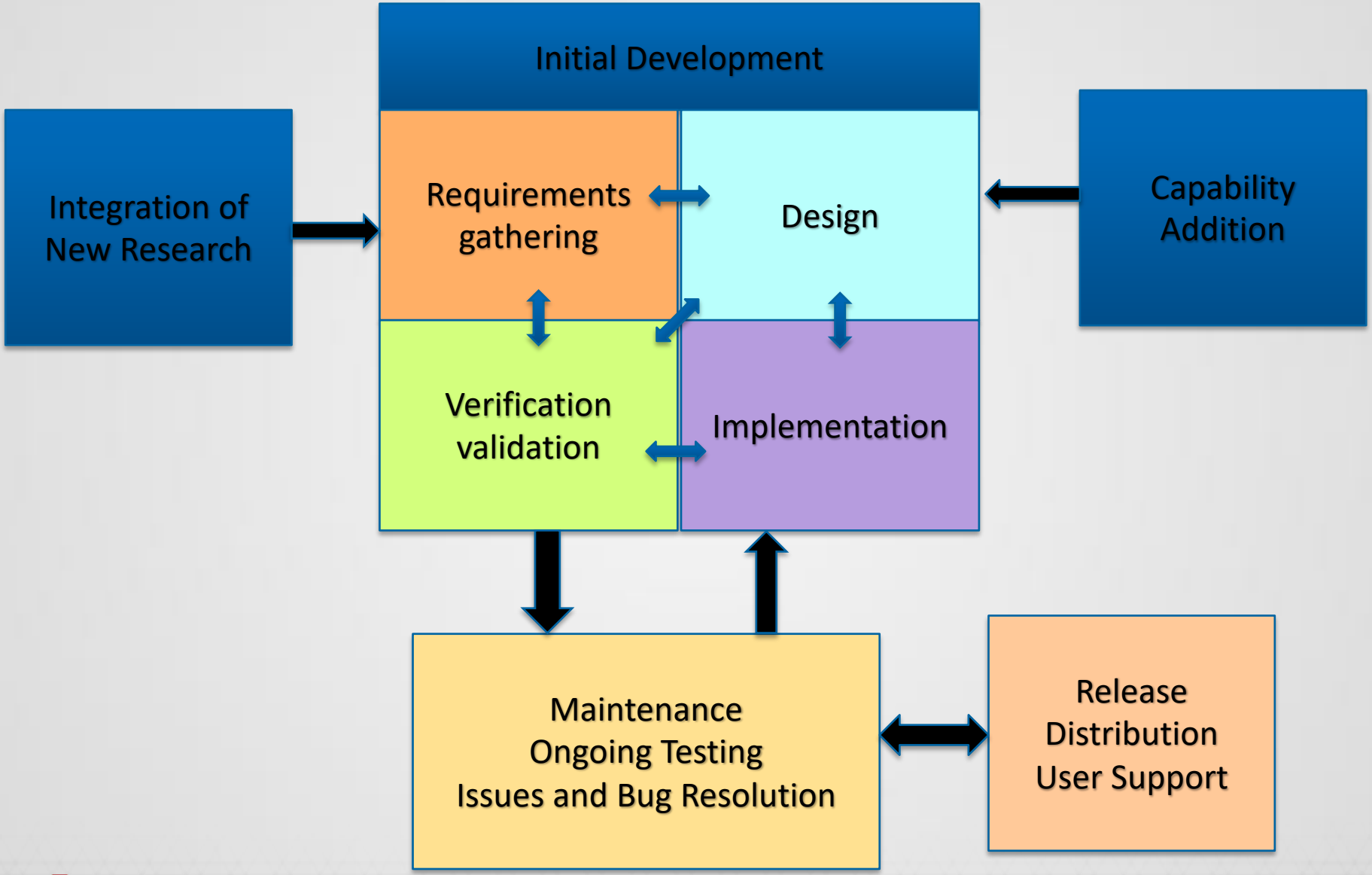
I am happy to present the results.

SETTING THE STAGE

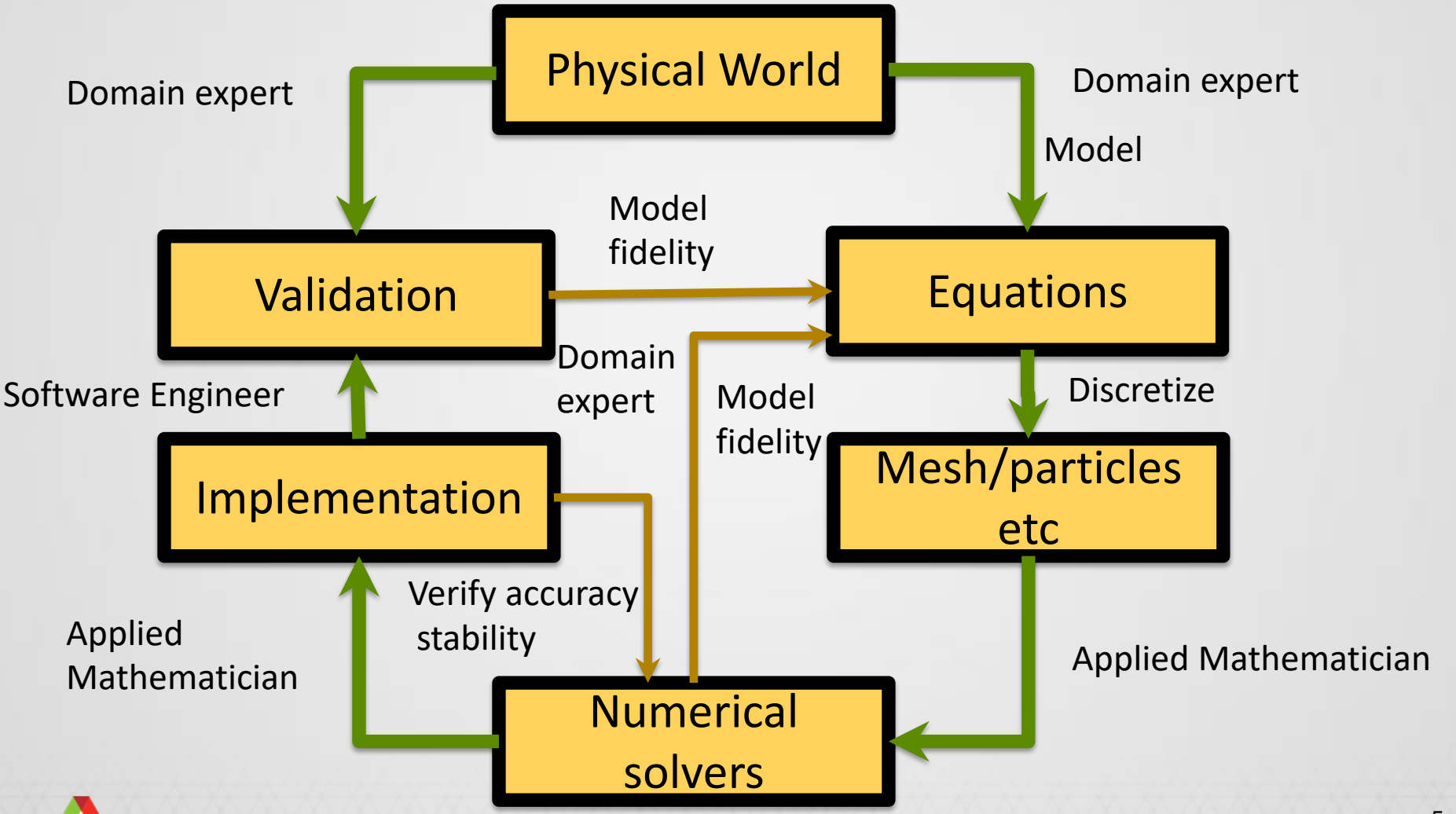
Since programming models and other tools are not so obliging, let me reduce the complexity by several orders of magnitude.

If we were starting a new multiphysics exascale software project today, that expects to have long term use for scientific discovery, how should we design the software?

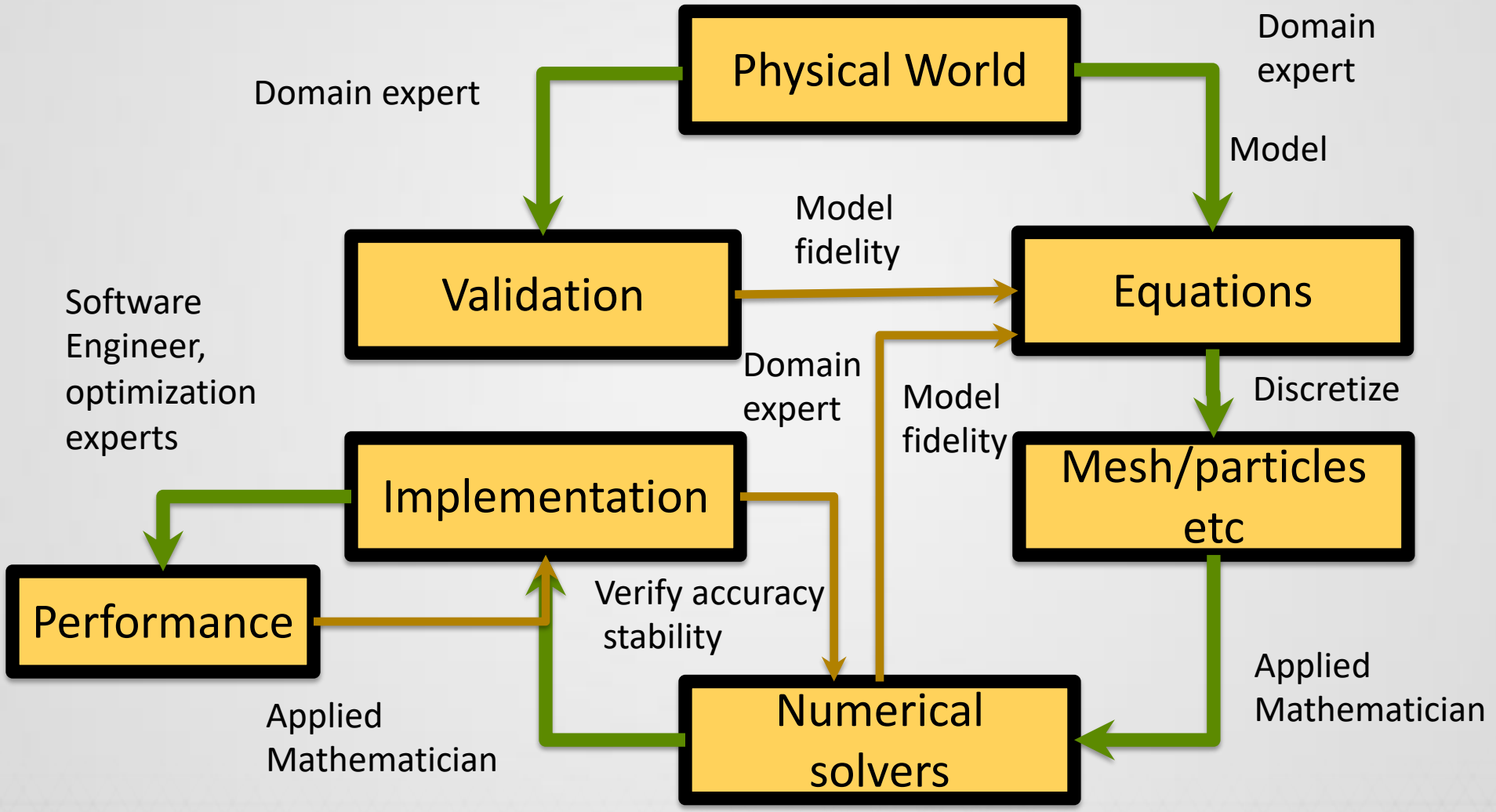
LIFECYCLE


Initial Development
Integration of New Research
Requirements gathering
Design
Capability Addition
Verification validation
Implementation
Maintenance
Ongoing Testing
Issues and Bug Resolution
Release
Distribution
User Support

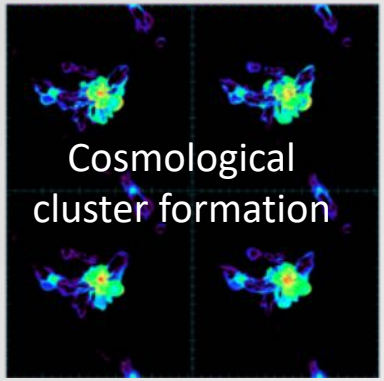
EXPERTISE MAP

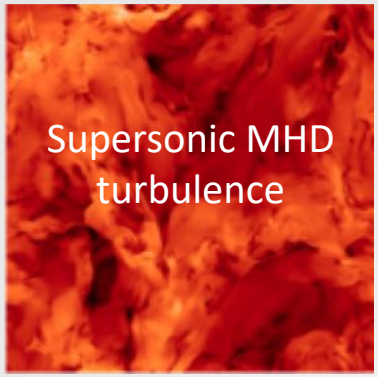


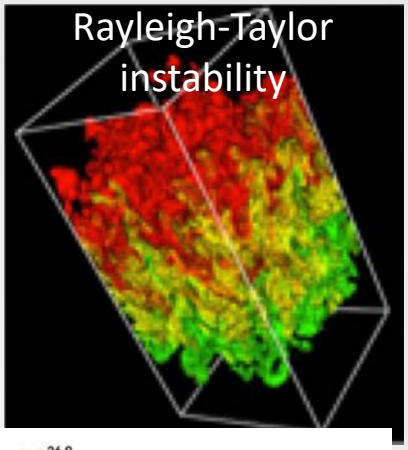
THERE IS MORE

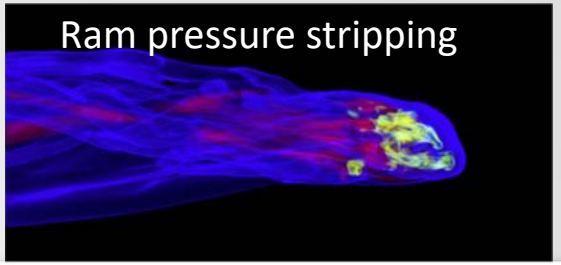


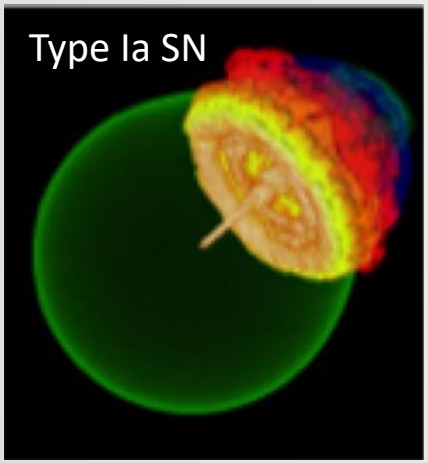
Example Software: FLASH

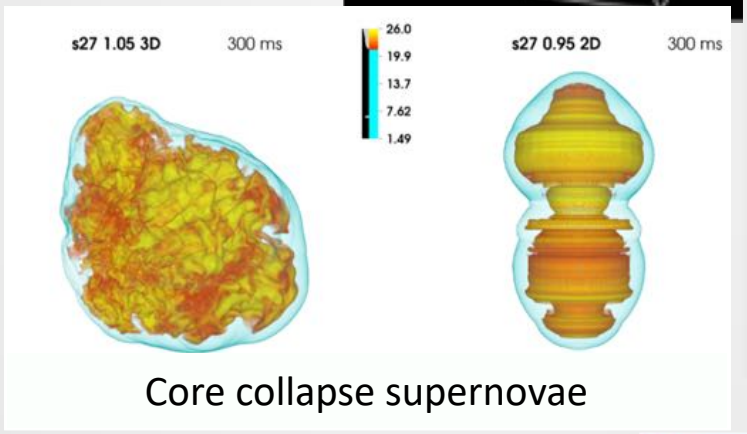


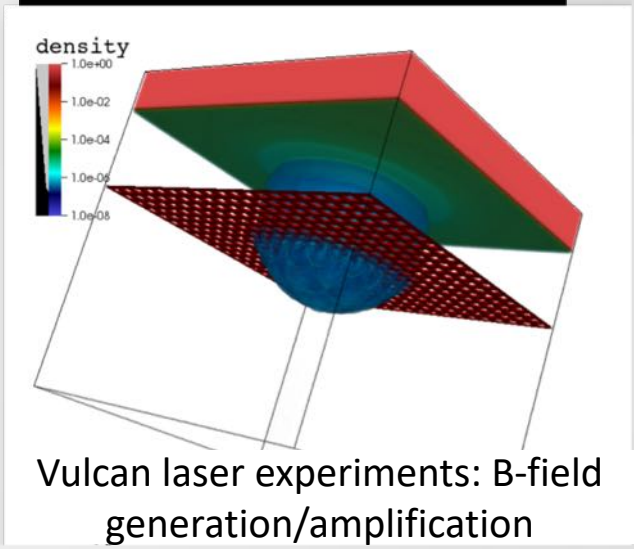


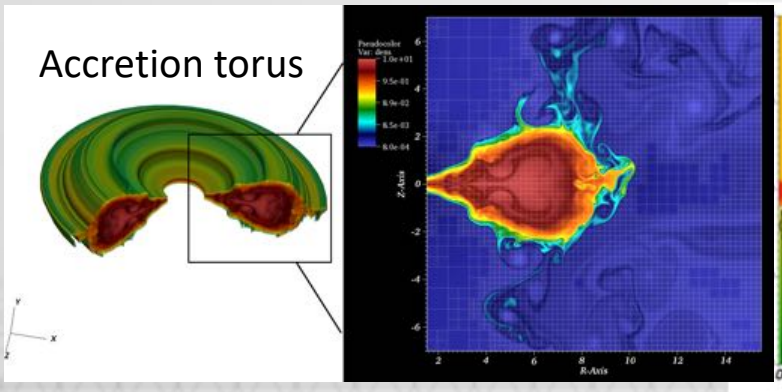


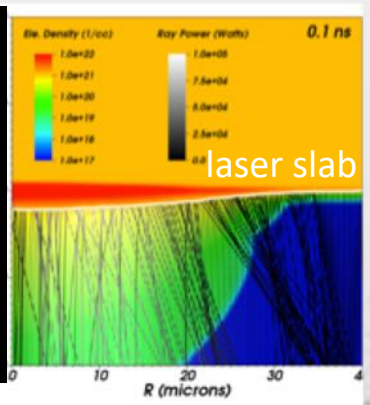


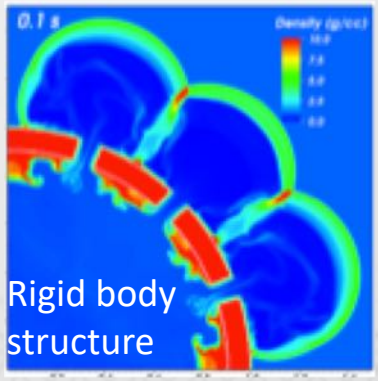


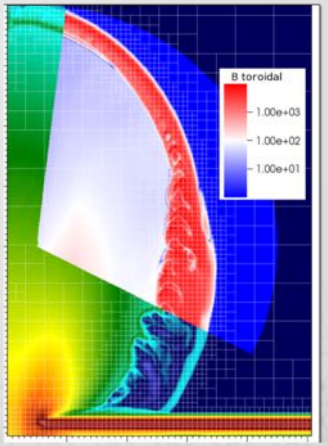










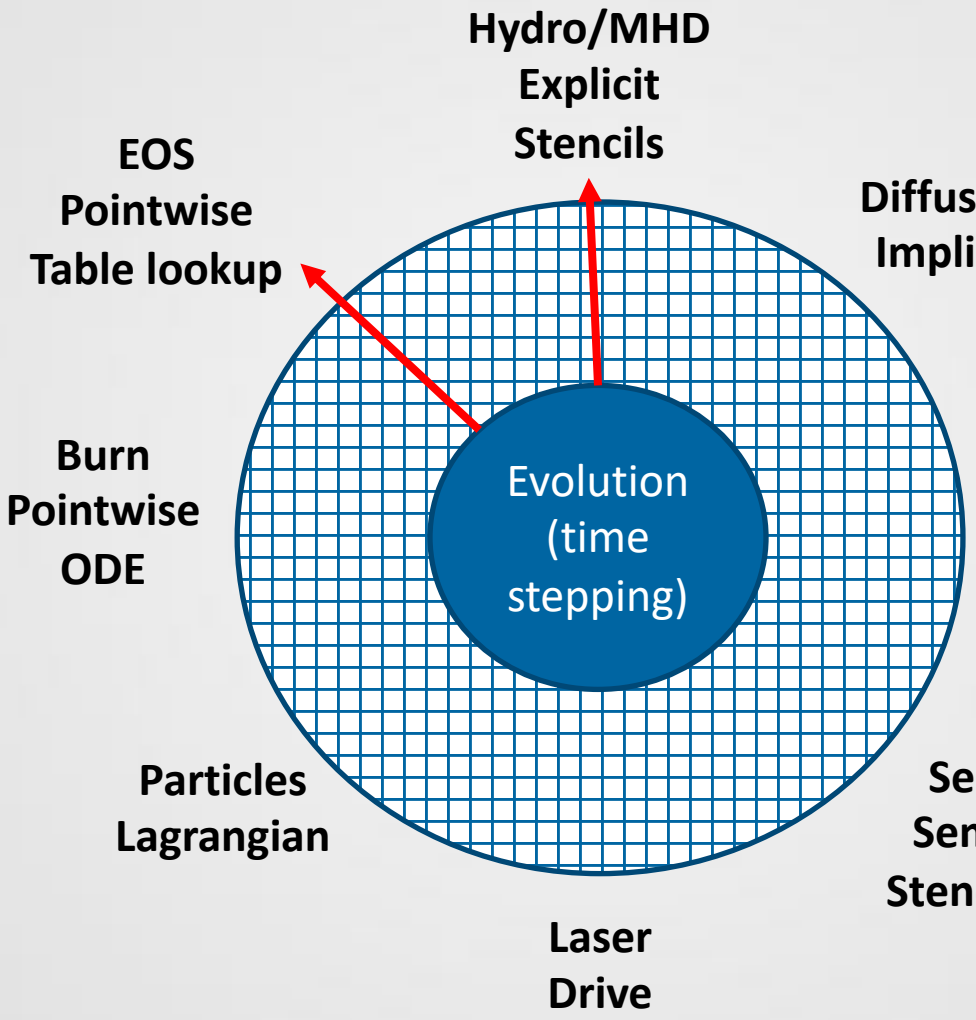


FLASH CODE BASICS

- An application code, composed of encapsulated functional *units*.
  - Units are combined and composed to form applications
  - Not one monolithic binary, each problem has its own distinct binary
- Setup tool (python) parses meta information, picks specific implementations of units and composes full application
  - Units can have alternative implementations
    - Three implementations of mesh are supported
  - Composability implies any of the implementations can be picked
- Mostly Fortran, some C, about 1.5 million lines of code
- Portable, and until recently performance portable

* Internal Release

CONFIGURATION

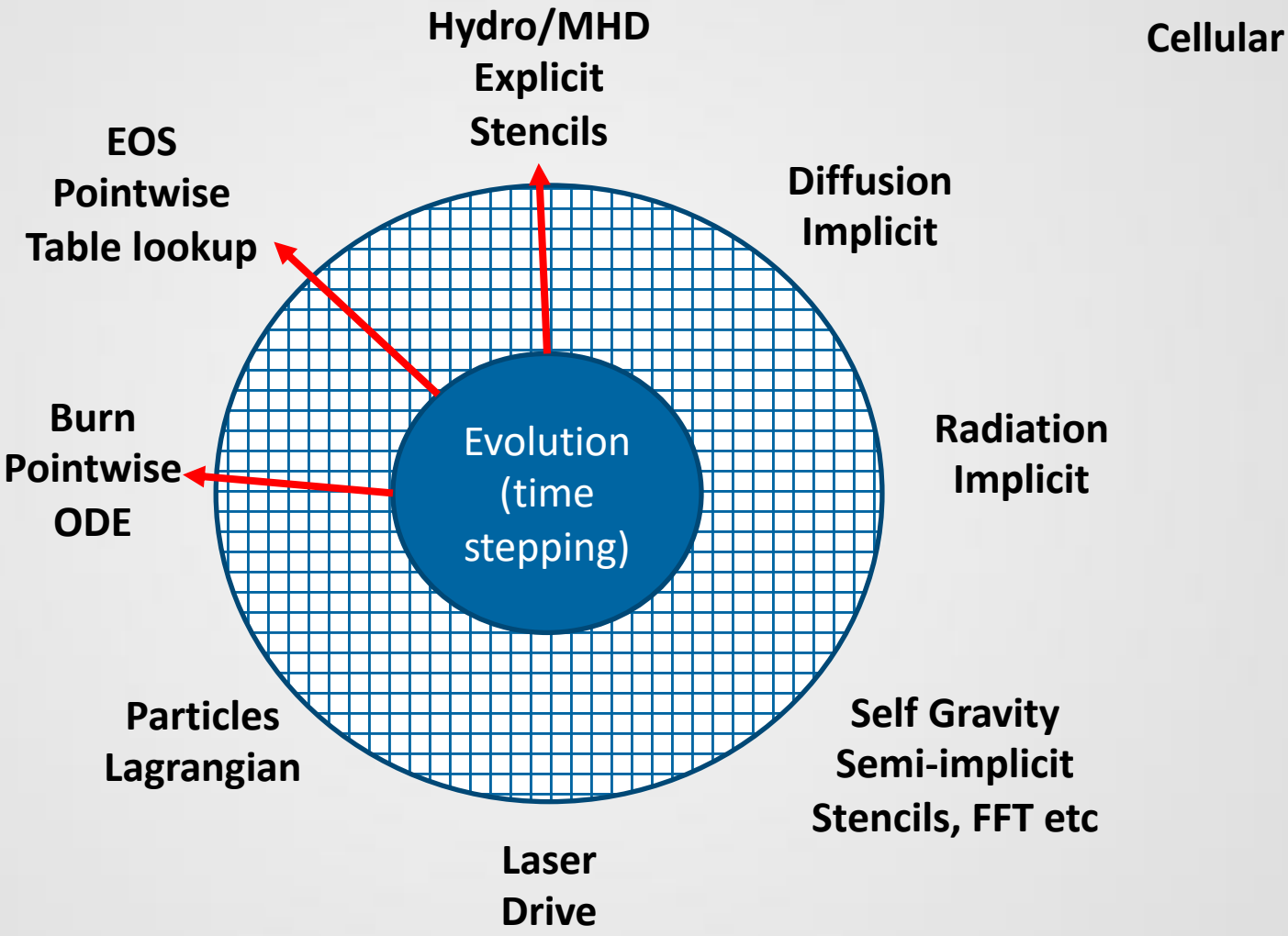


Shock Tube

Library

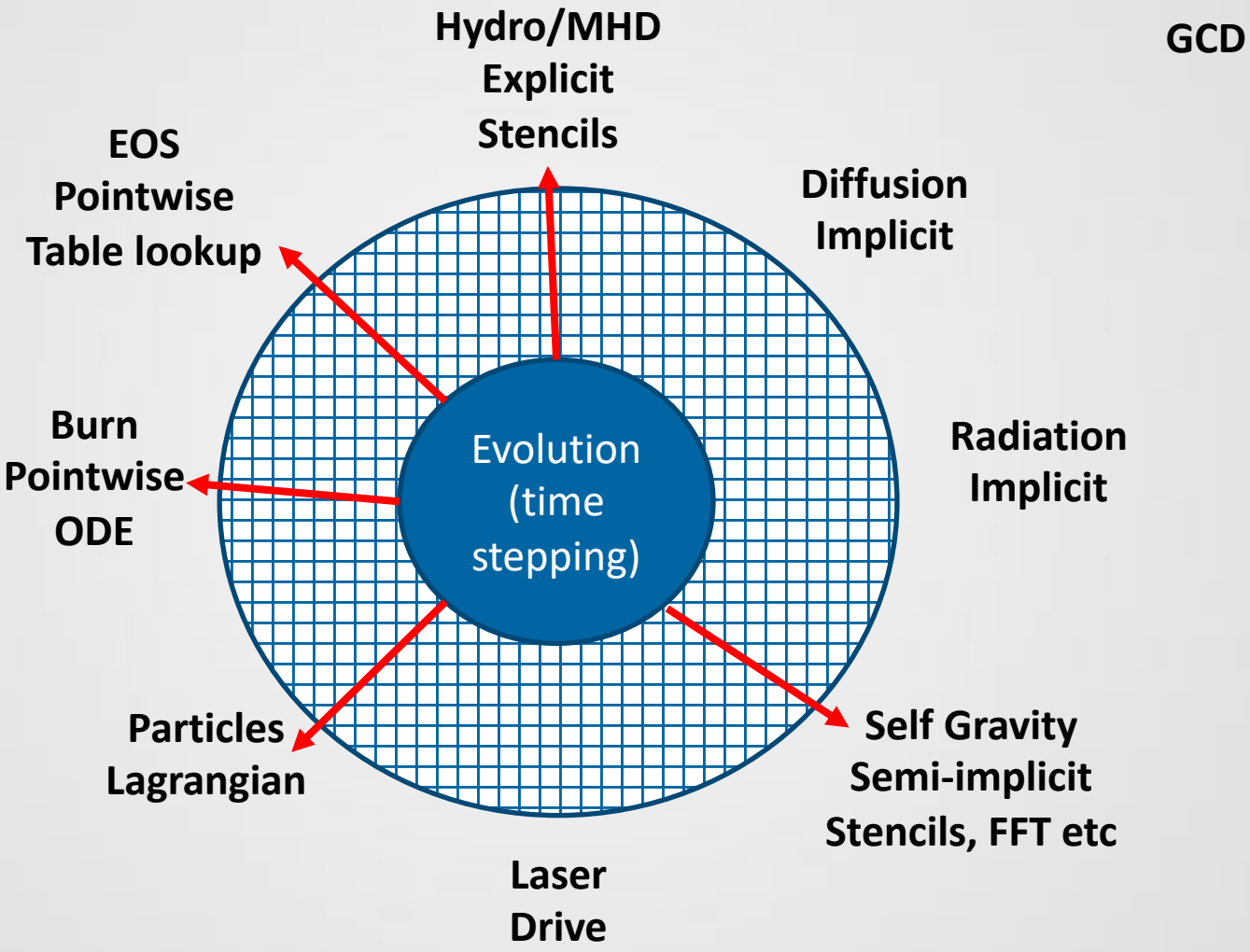
Dubey et al, Parallel Computing 2009

CONFIGURATION



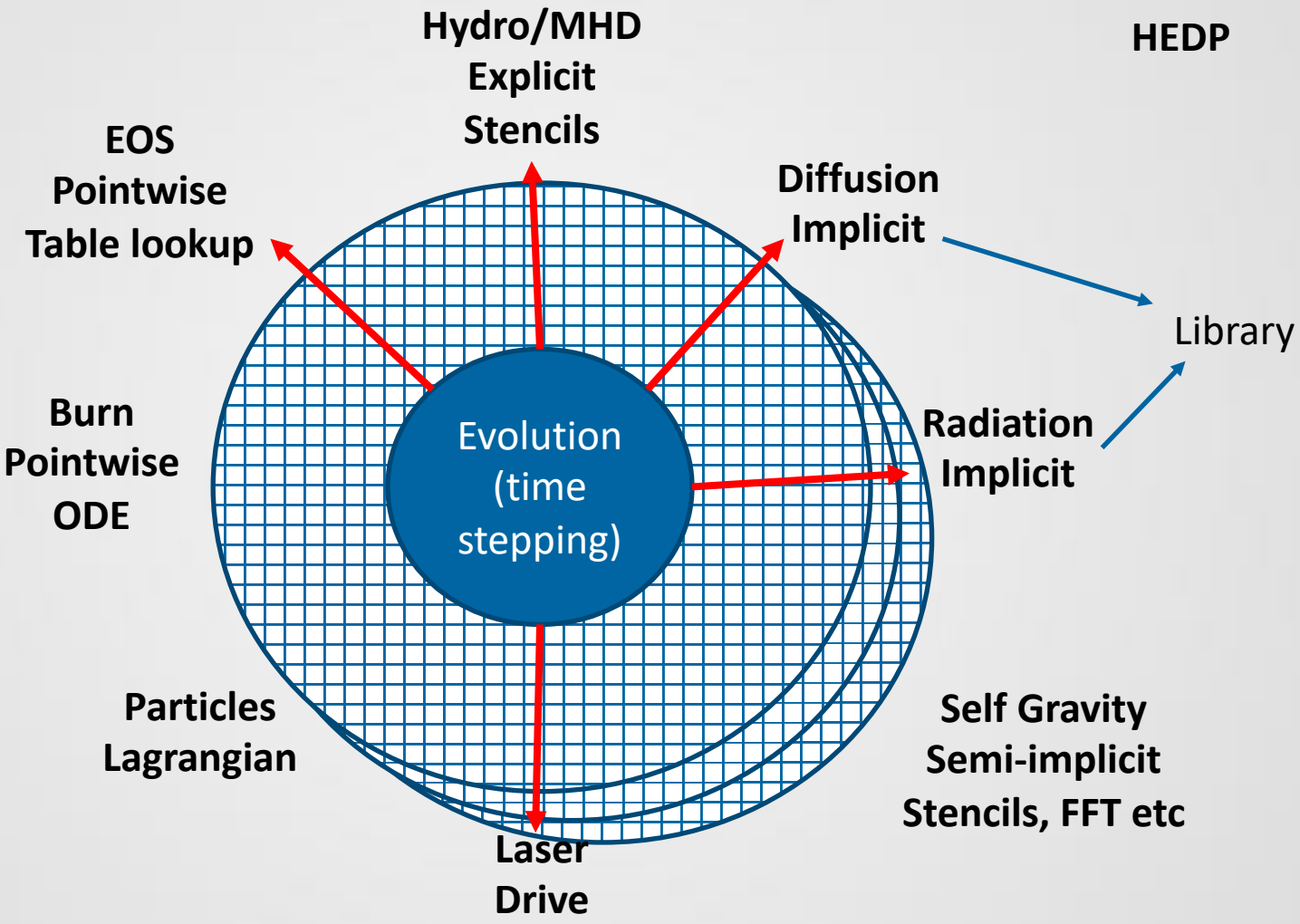
Dubey et al, Parallel Computing 2009

CONFIGURATION



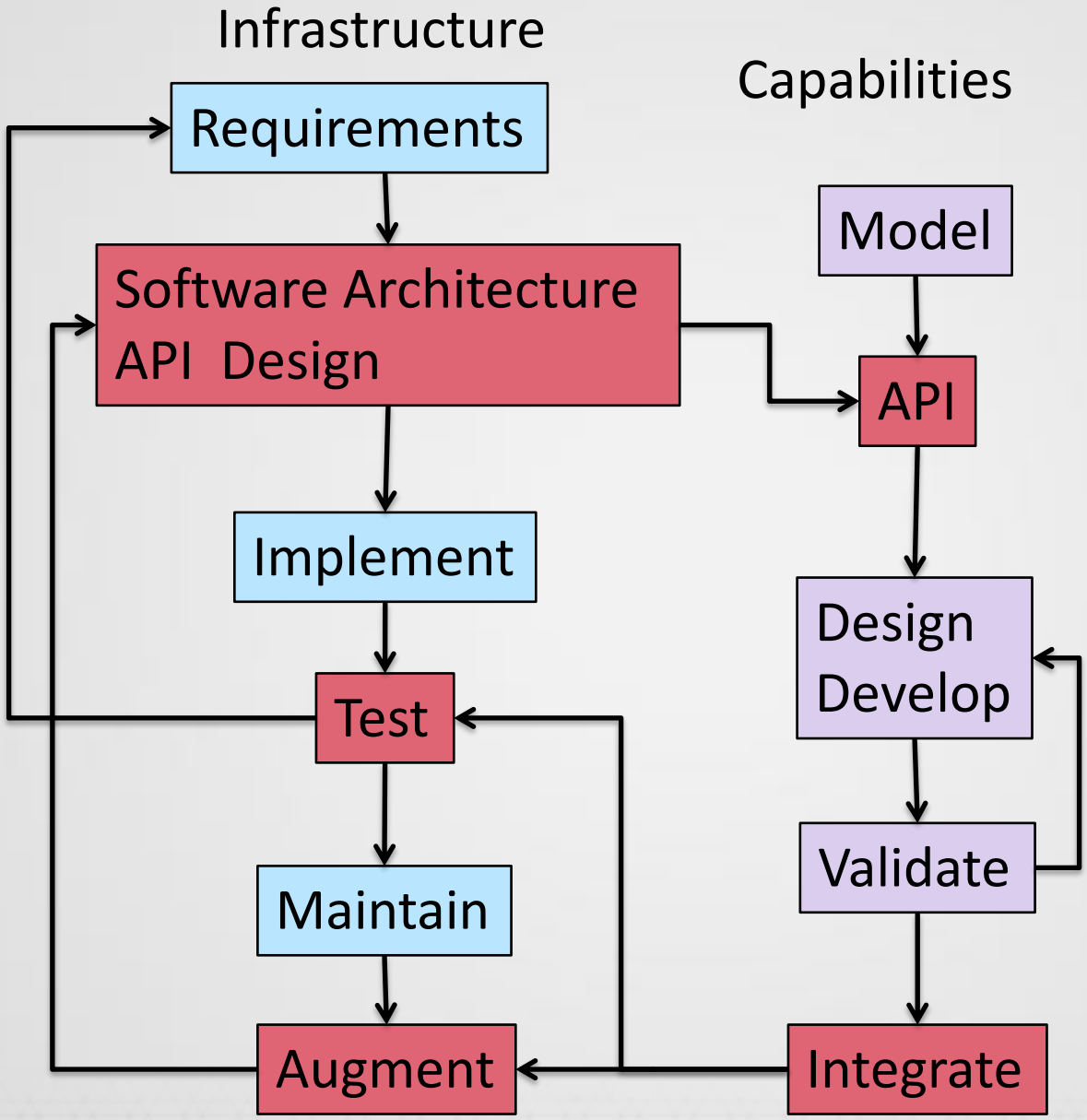
Dubey et al, Parallel Computing 2009

CONFIGURATION



Dubey et al, Parallel Computing 2009

HOW DO WE MAKE IT WORK?

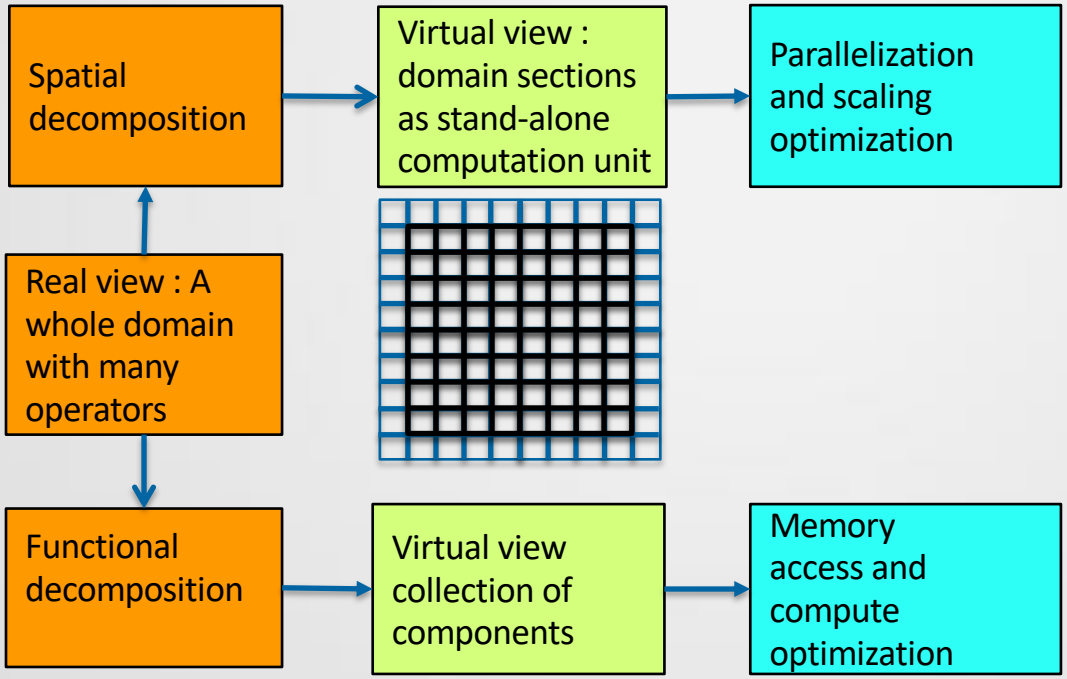


DESIRABLE PROGRAMMING MODELS

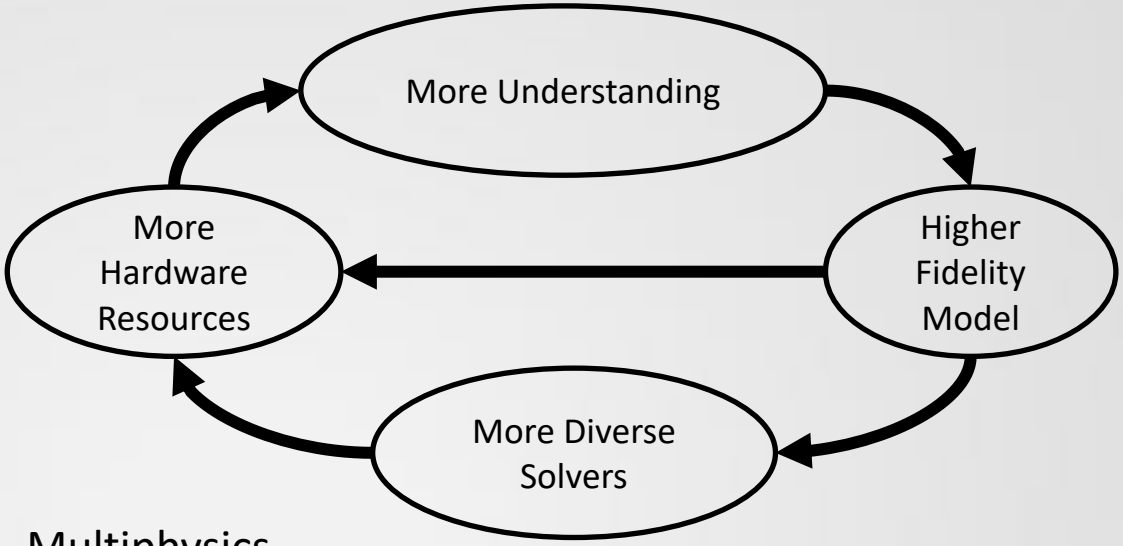
- The most important
  - Separation of concerns
  - Expertise encapsulation
- The next
  - Convergence
    - Abstractions
    - Not the implementations, just conceptual details
    - Abstract machine models help

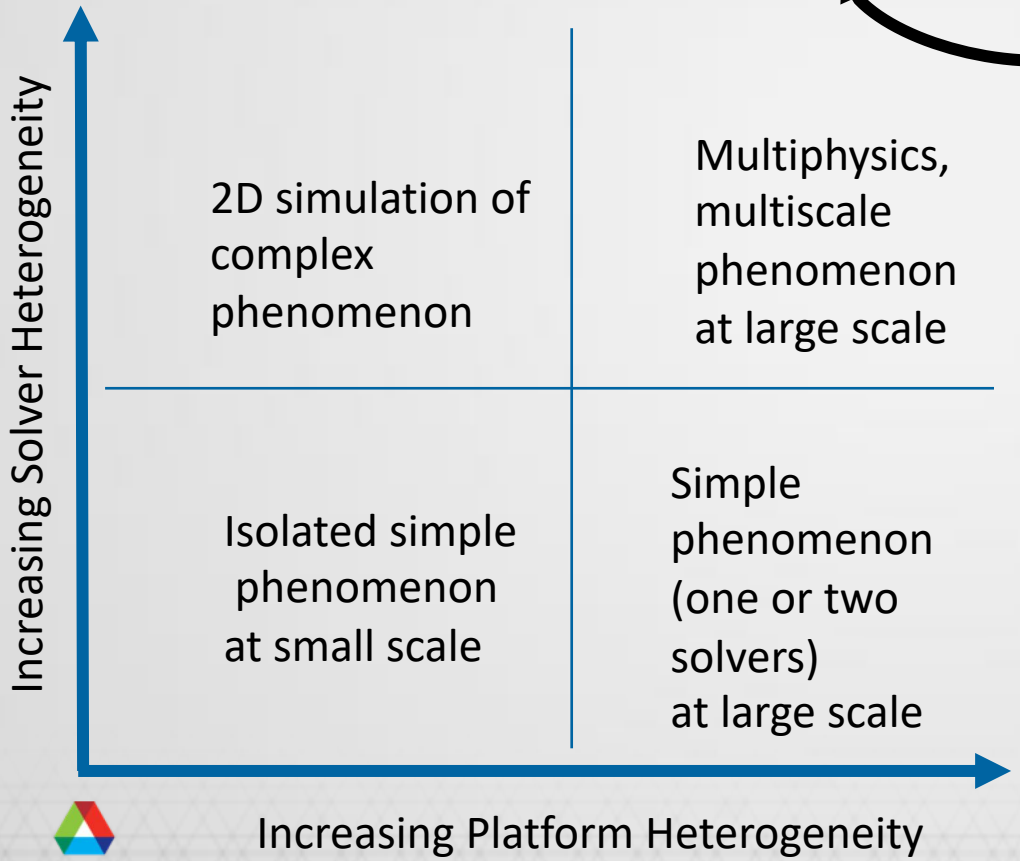
WHAT WE DID BEFORE

- Distributed memory parallelism
- Abstract machine model is processor with memory hierarchy and communication channels

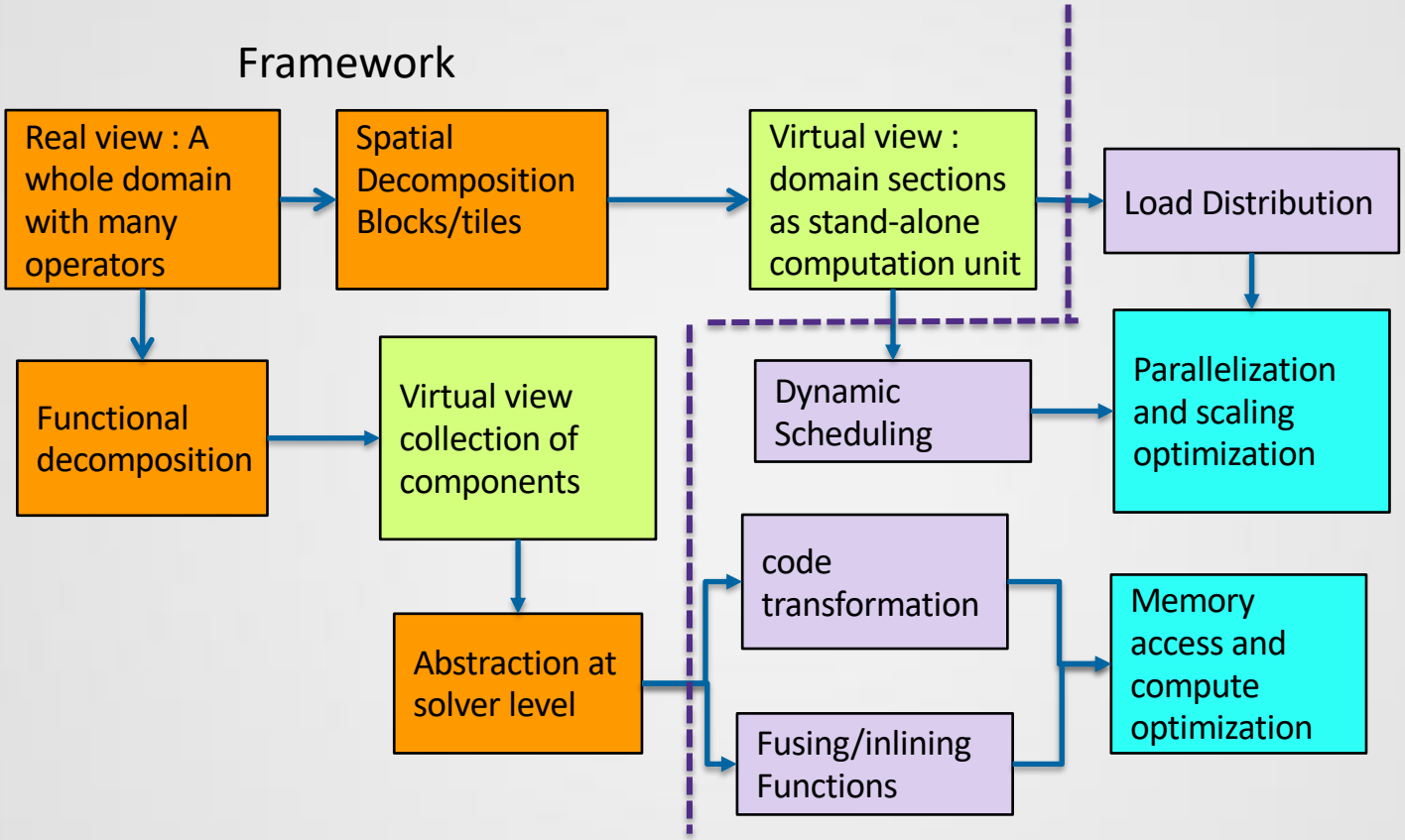


WHERE WE ARE NOW?



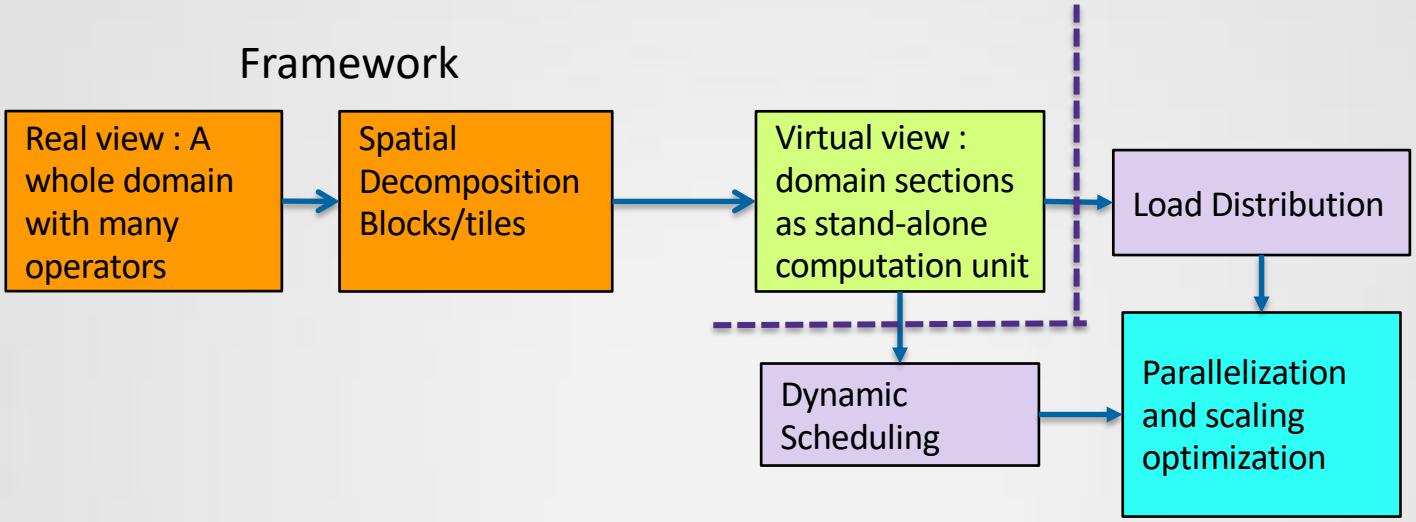


WHAT WE WANT TO DO NOW



Management of hierarchical parallelism through tiling and custom asynchronous execution under the hood by infrastructure and hidden from physics units through smart iterators

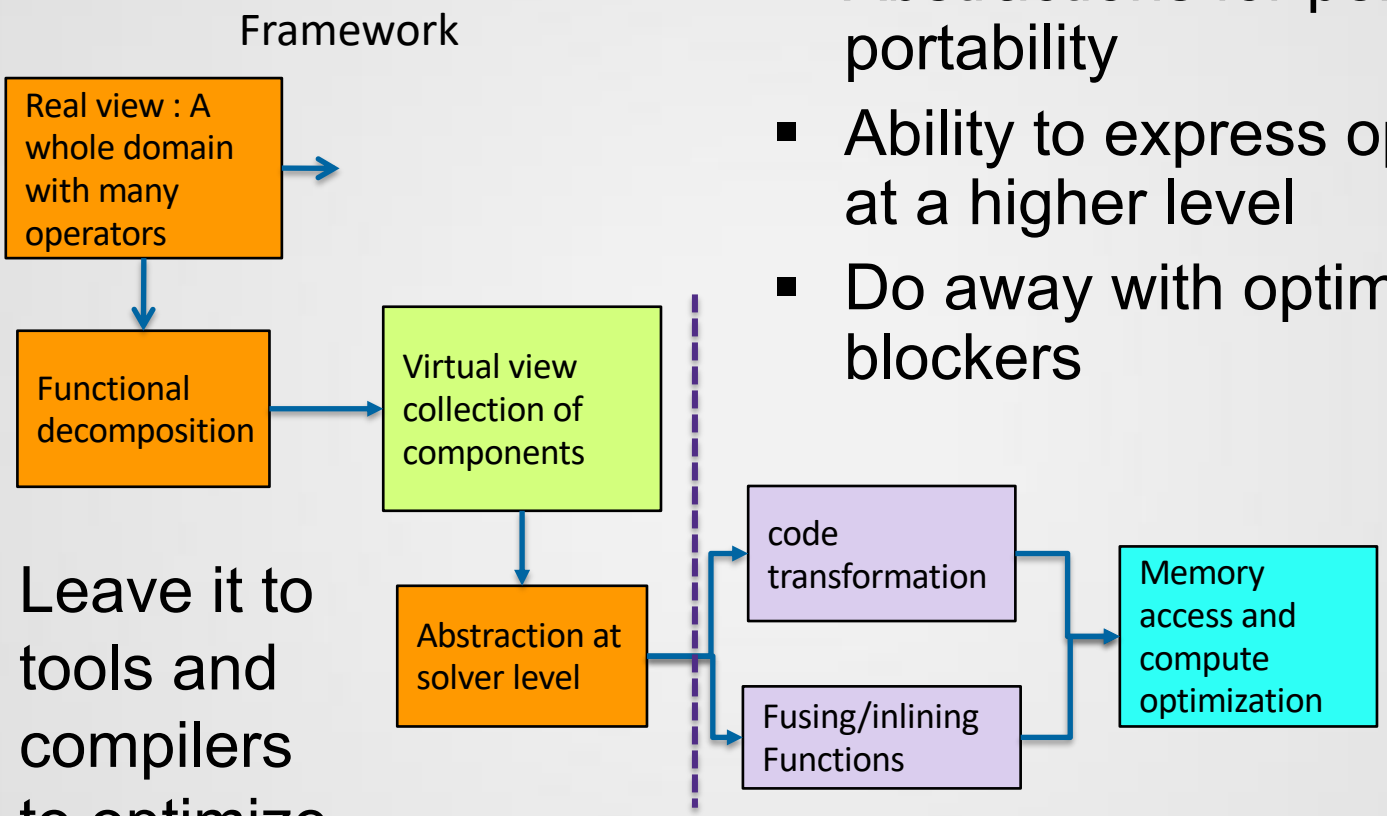
WHAT WE WANT TO DO NOW



- AMR infrastructure: refinement, load balancing, work redistribution
- Meta-information about domain sections
- Asynchronization at block and operator level
- No kernel optimization in this part

WHAT WE WANT TO DO NOW

- Abstractions for performance portability
- Ability to express operations at a higher level
- Do away with optimization blockers



- Leave it to tools and compilers to optimize

THE BOTTOM LINE: PARALLELISM

- MPI is not difficult, decomposition is
- In parallelization neither all nor none is good
  - All – leave everything to the compiler
    - Domain specific knowledge lost – wasted opportunity
    - Compilers get impossible job, cannot optimize
  - None – orchestrate everything explicitly
    - Not feasible for even moderately complex application
    - Impossible from productivity perspective
- Whichever model is used, understanding the parallelizable structure of application is critical
- Constructs to encode the understanding needed

THE BOTTOM LINE: KERNELS

- C++ => Pushing a needlessly complex language that lacks basic structures
  - If there is a mesh there are 3D arrays
    - meta-data built and carried around
    - Explicit order of access and order of operations
  - No graceful way to encode lack of dependence

- Not that Fortran is much better
  - Array intrinsics useless for expressing stencils

THE BOTTOM BOTTOM LINE

- MPI and/or Fortran can be abandoned
- For a collection of abstractions or a language
- The language must have
  - Constructs for expressing parallelism at different granularities
  - Express independence of operators
  - Data structures in tune with scientific software needs
  - Interoperability with current languages
- And then all stars should line up
  - Community support, longevity, funding, compilers

Chapel's design fits the bill, will stars line up ?
(I haven't looked at Julia, that might be another candidate)

Questions ?