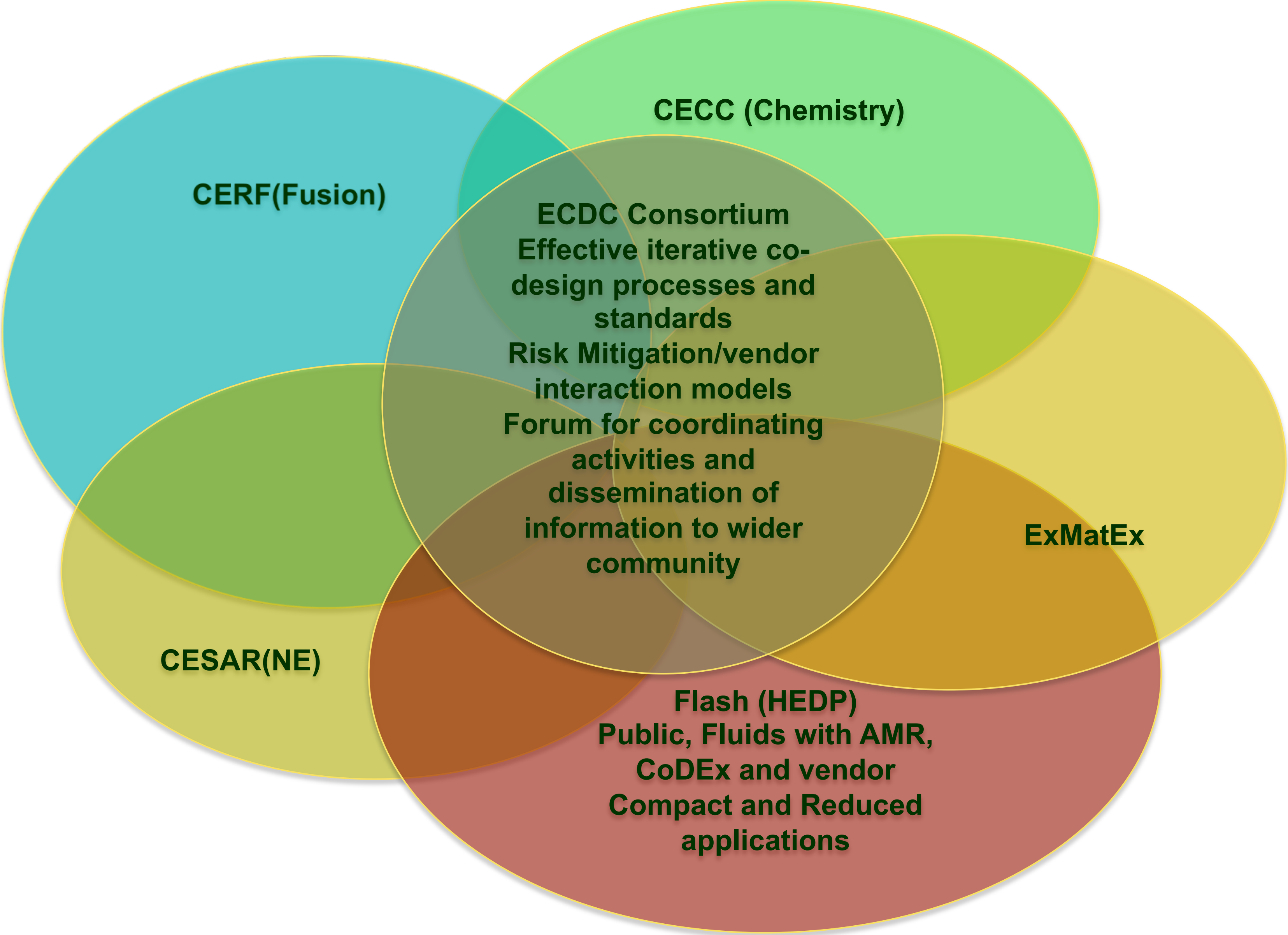
Flash Co-Design Methodology

Anshu Dubey
DOE NNSA OASCR Flash Center
University of Chicago

NVIDIA
February 2011

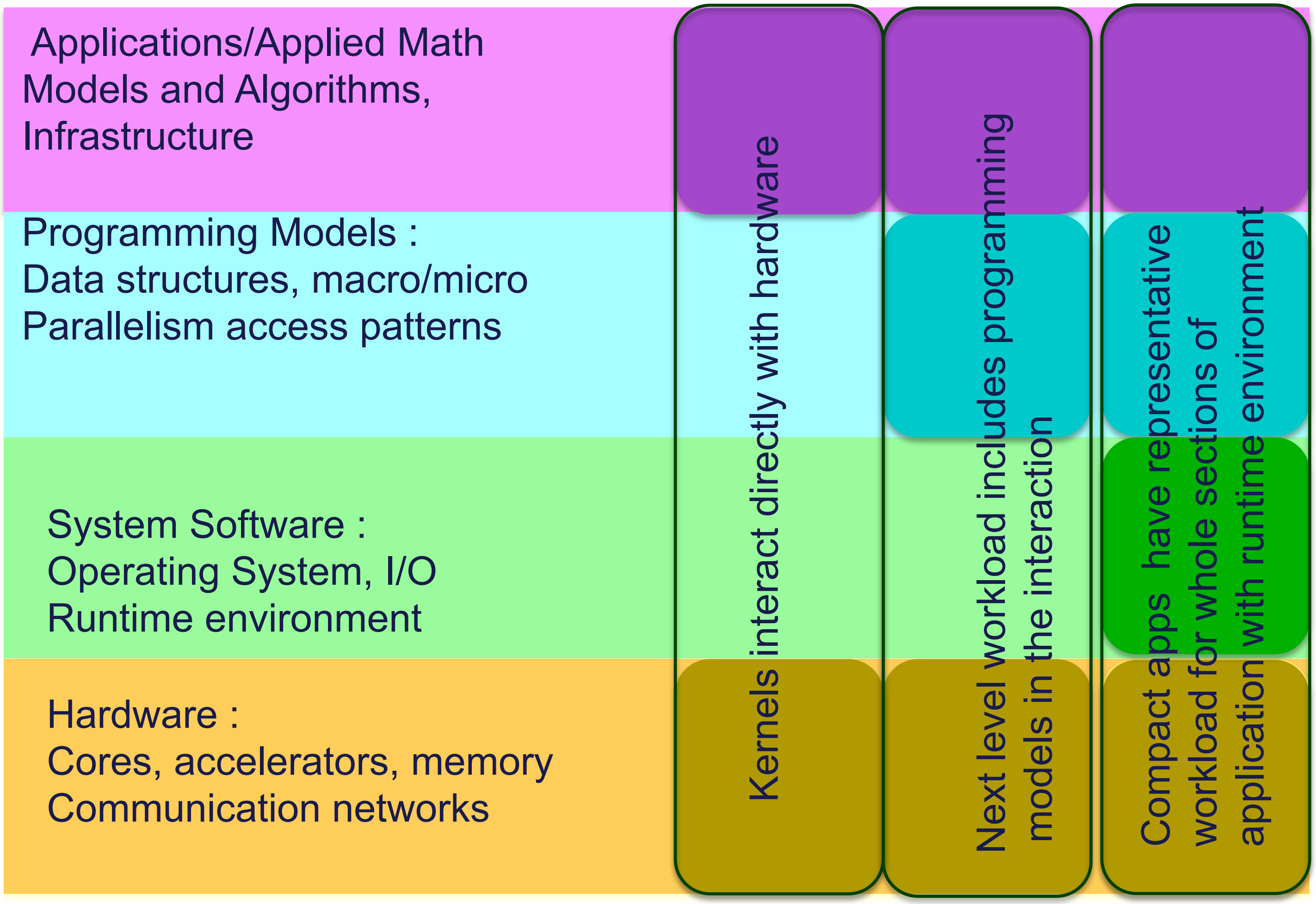
Co-design Centers and Consortium



ECDC Consortium
Effective iterative co-design processes and standards
Risk Mitigation/vendor interaction models
Forum for coordinating activities and dissemination of information to wider community

Flash (HEDP)
Public, Fluids with AMR, CoDEx and vendor
Compact and Reduced applications

Vertical Integration



Applications/Applied Math
Models and Algorithms,
Infrastructure

Programming Models :
Data structures, macro/micro
Parallelism access patterns

System Software :
Operating System, I/O
Runtime environment

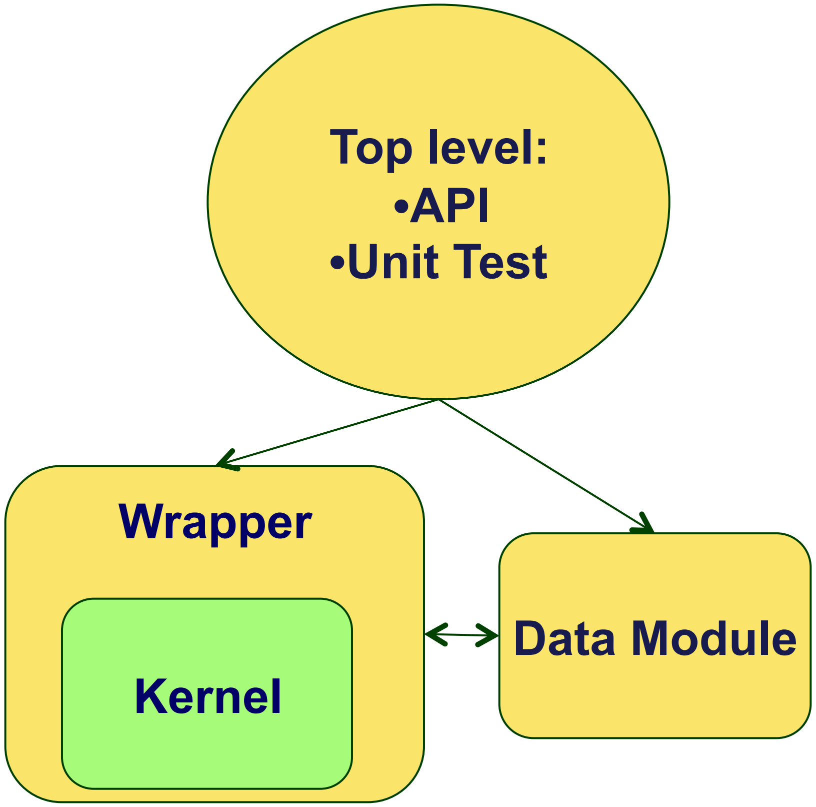
Hardware :
Cores, accelerators, memory
Communication networks

Kernels interact directly with hardware

Next level workload includes programming models in the interaction

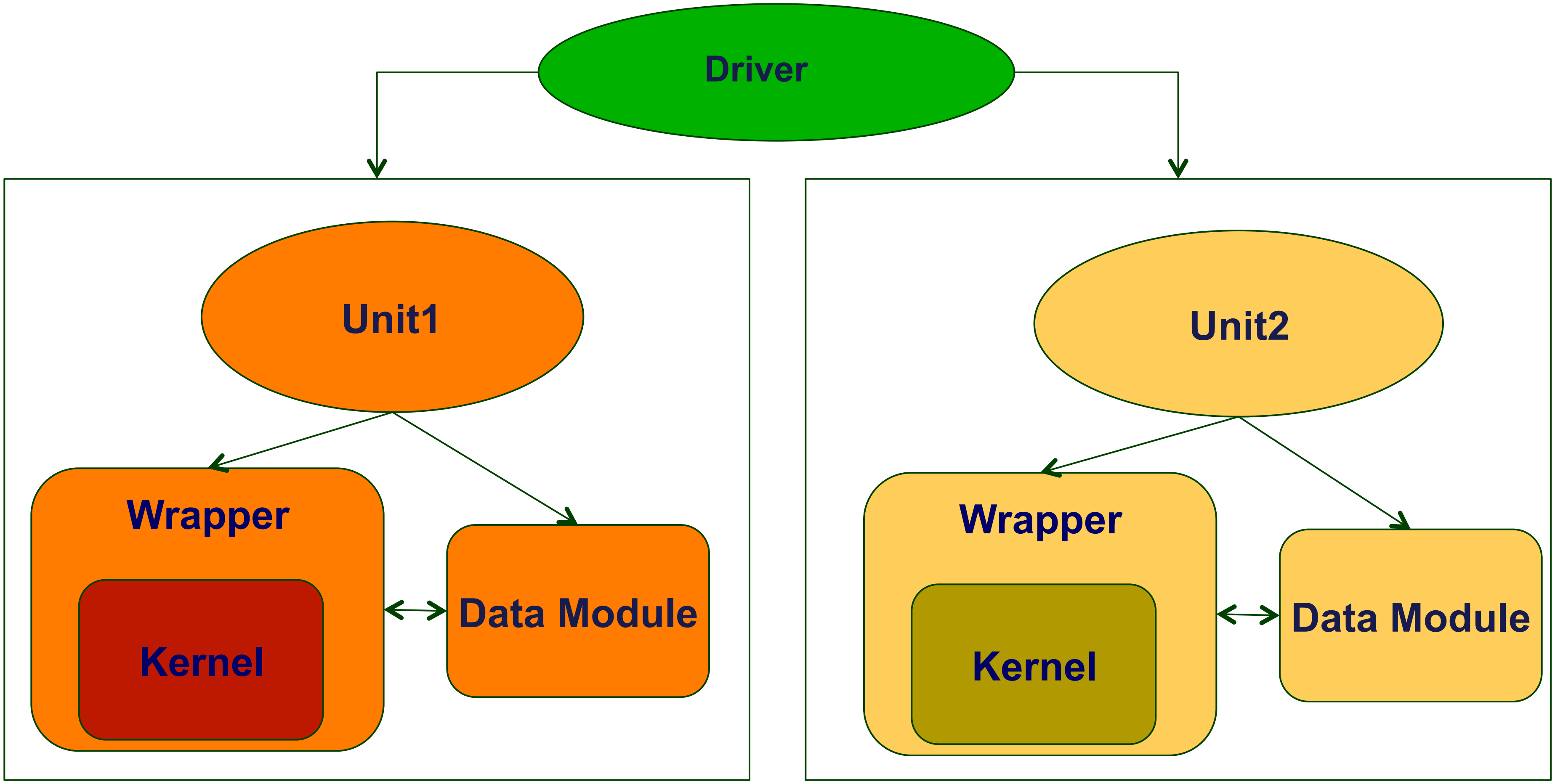
Compact apps have representative workload for whole section of application with runtime environment

Kernels & Reduced Application



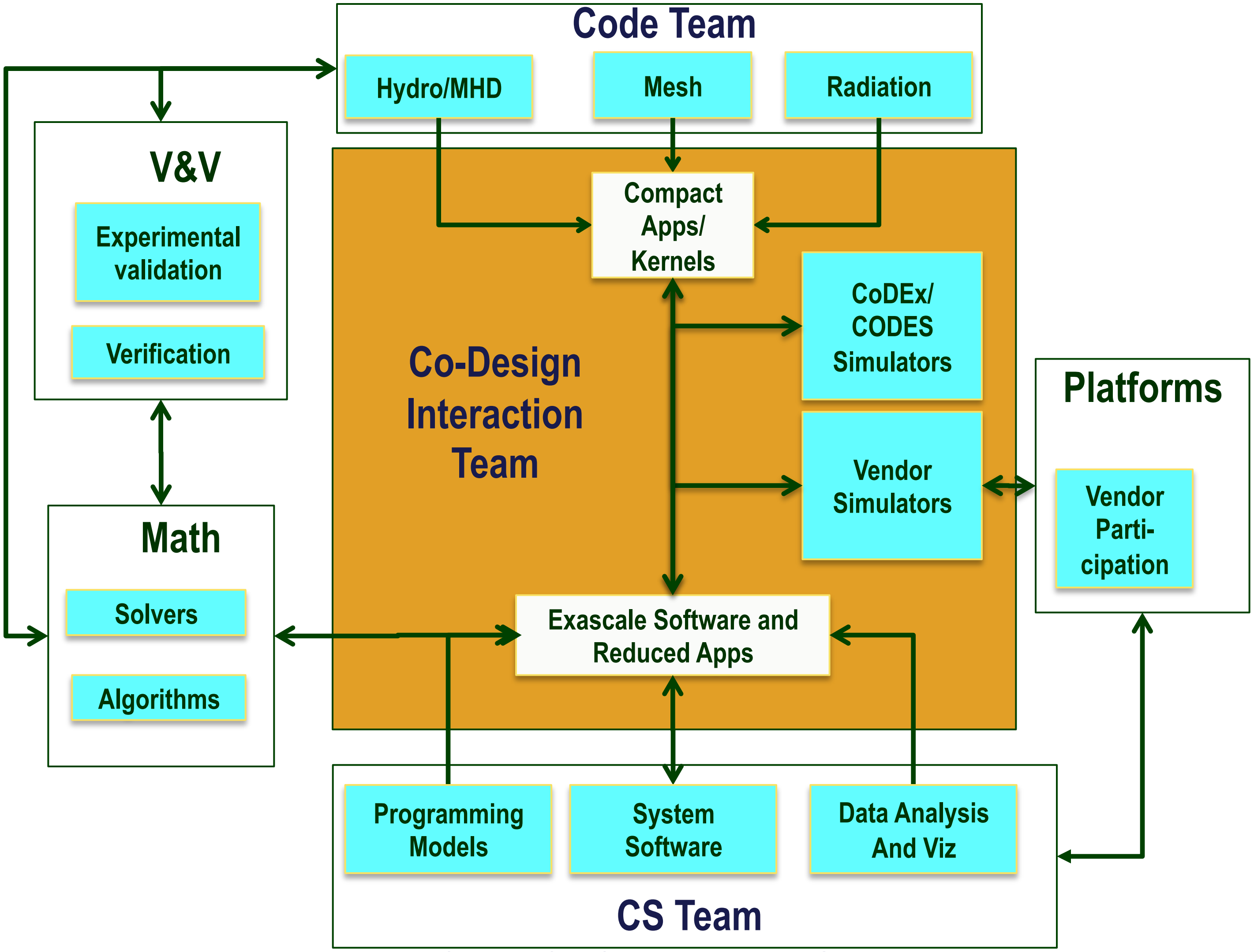
Unit test is effectively a reduced application

Compact Applications



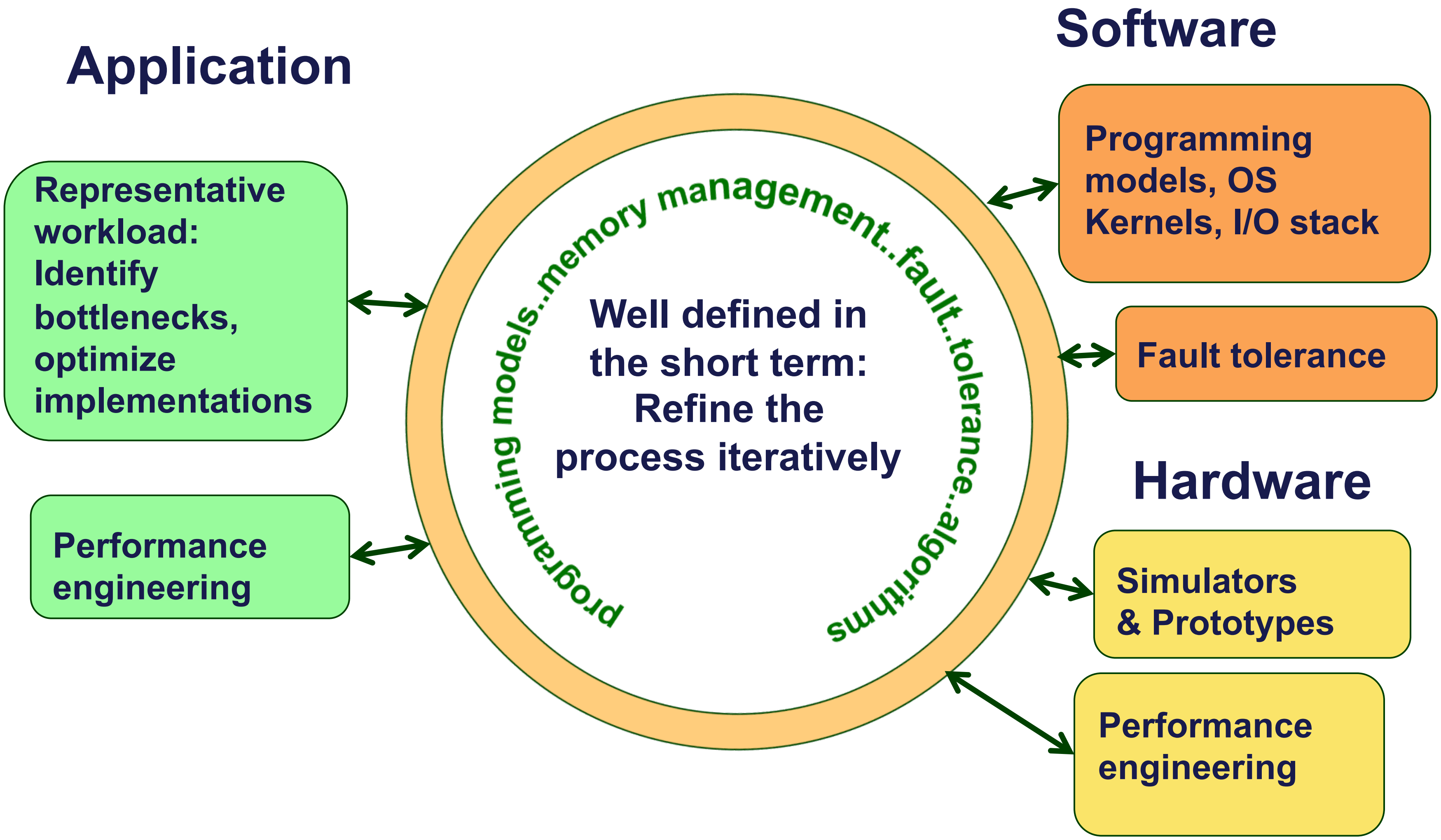
Compact applications use simplified interaction among units to create a subset of the application functionality

Flash Co-Design Interaction



Flash Center for Computational Science
University of Chicago

Co-Design Process



Application

Representative workload: Identify bottlenecks, optimize implementations

Performance engineering

programming models..memory management..fault..tolerance..algorithms

Well defined in the short term: Refine the process iteratively

Software

Programming models, OS Kernels, I/O stack

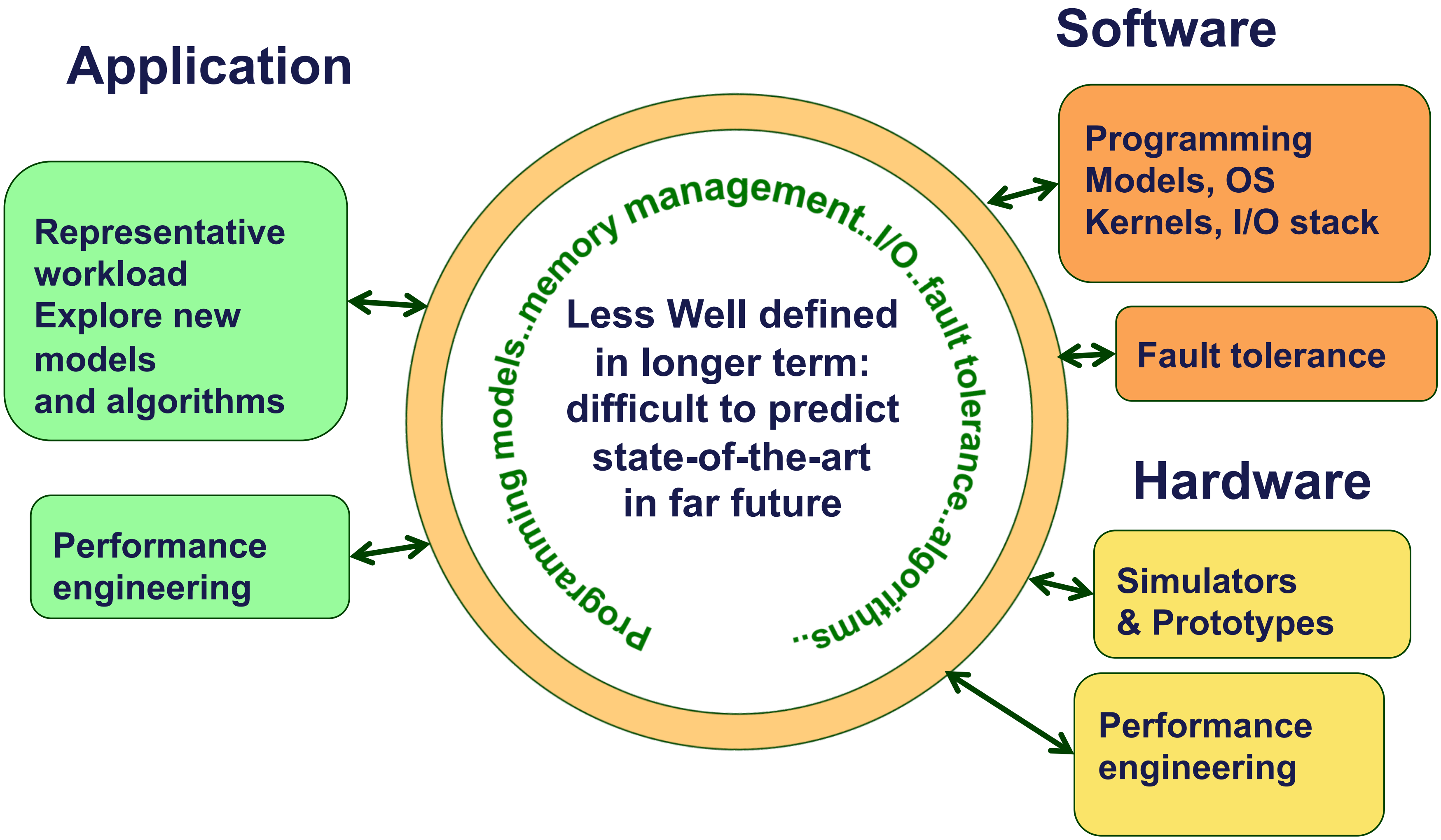
Fault tolerance

Hardware

Simulators & Prototypes

Performance engineering

Co-Design Process



Application

Representative workload
Explore new models
and algorithms

Performance engineering

Programming models..memory management..I/O..fault tolerance..algorithms..

Less Well defined in longer term: difficult to predict state-of-the-art in far future

Software

Programming Models, OS
Kernels, I/O stack

Fault tolerance

Hardware

Simulators & Prototypes

Performance engineering

Expected Challenges

Across the Nodes

- Parallel IO
- Higher degree of macro parallelism
- Higher fidelity physics dictates greater coupling
    - Implicit/semi-implicit treatment
- New parallel algorithms
- Trade-off between duplication and communication
- Possibly more hierarchy

At Node Level

- Memory intensive computations
- Increasing limits on available memory per process
- Aggressive reuse of memory
- Distinguish between cores
- New algorithms

Faults

- Frequent failures
- Silent errors
- Non-deterministic algorithms
- Redundancy

What Applications Can Do

- Greater encapsulation
  - Minimize common data
  - Maximize code sections that are re-entrant
  - Increase isolation between layers
  - Separate code functionalities such that different optimizations are applicable to different layers
- Minimize kernel dependency on programming models

- Expose optimization and fault tolerance possibilities
  - Be clearer about dependencies
  - Identify critical sections Vs the non critical sections
  - Define more compact working sets
- Explore more inherently robust alternative algorithms
  - Stochastic Vs deterministic

What the applications shouldn’t have to do is to rewrite half a million lines of code. It isn’t even always possible because physics is what it is

What Applications need from Co-Design

- Abstraction
  - Retain code portability
  - Enable layering in architecture
  - Standardized interfaces for common functionalities

- Framework for testing ideas
  - Fault notification and recovery models
  - Ability to evaluate direct influence of architecture decisions
  - Tuning parameters