Exascale Co-Design Center for Materials in Extreme Environments

"Algorithm research has been driven by hard to use machines."
–Rob Schreiber (HP Labs)

"People who are serious about software should make their own hardware."
–Alan Kay (Xerox PARC)

Timothy Germann, Director

James Belak, Deputy Director

Allen McPherson, Computer Science Lead

David Richards, Proxy App Lead

JASON Study on Exascale

27-29 June 02012

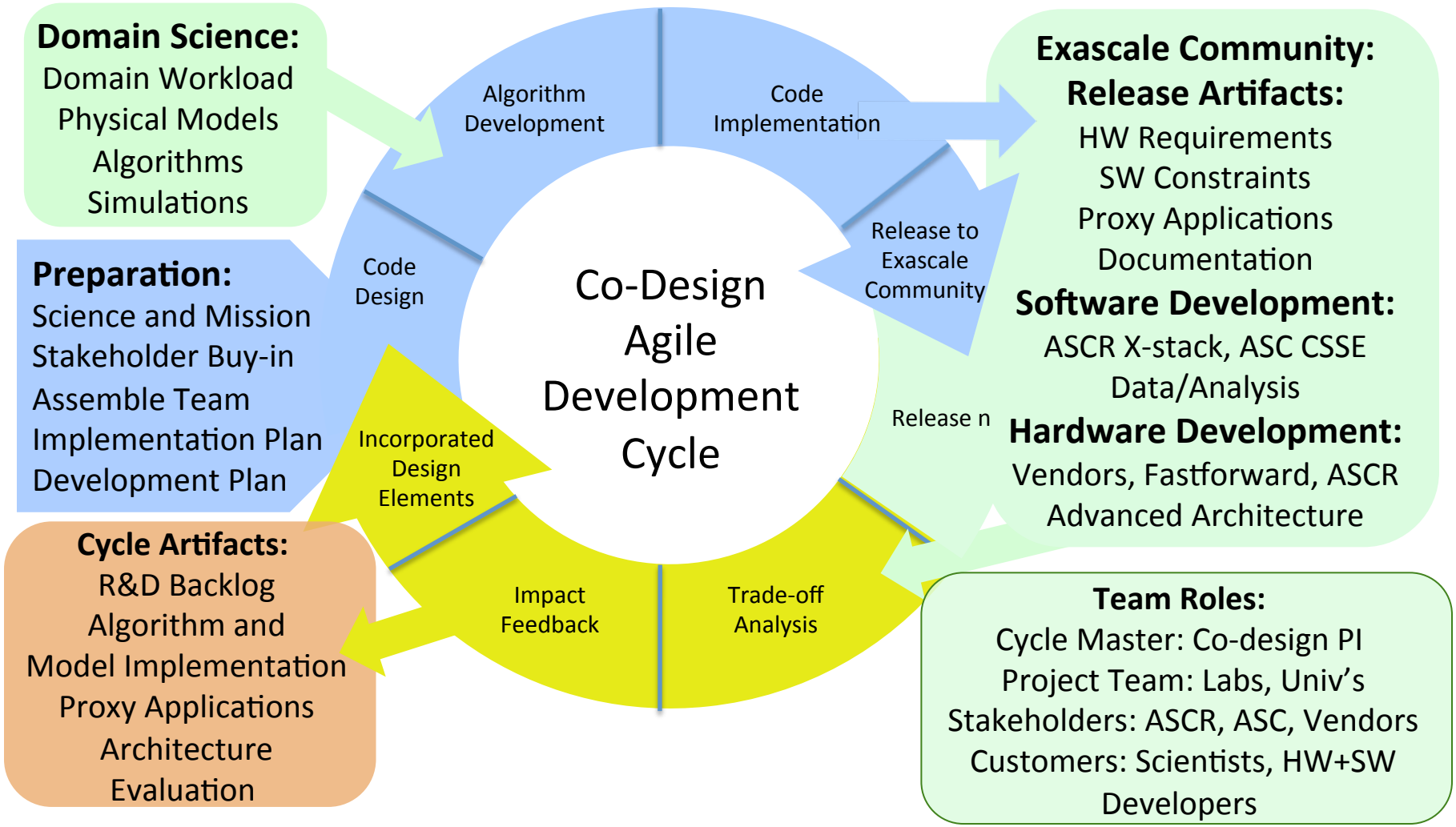
La Jolla, CA

"(Application driven) co-design is the process where scientific problem requirements influence computer architecture design, and technology constraints inform formulation and design of algorithms and software." –Bill Harrod (DOE)

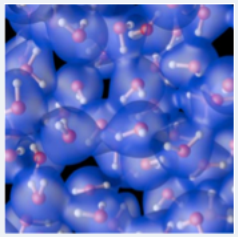
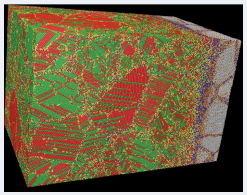
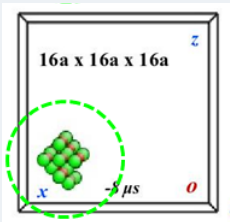
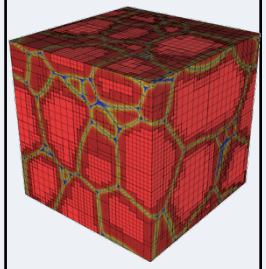
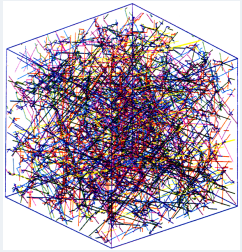
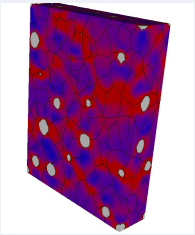
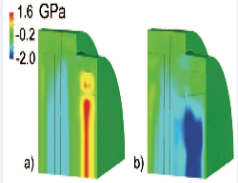
LLNL-PRES-562153

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

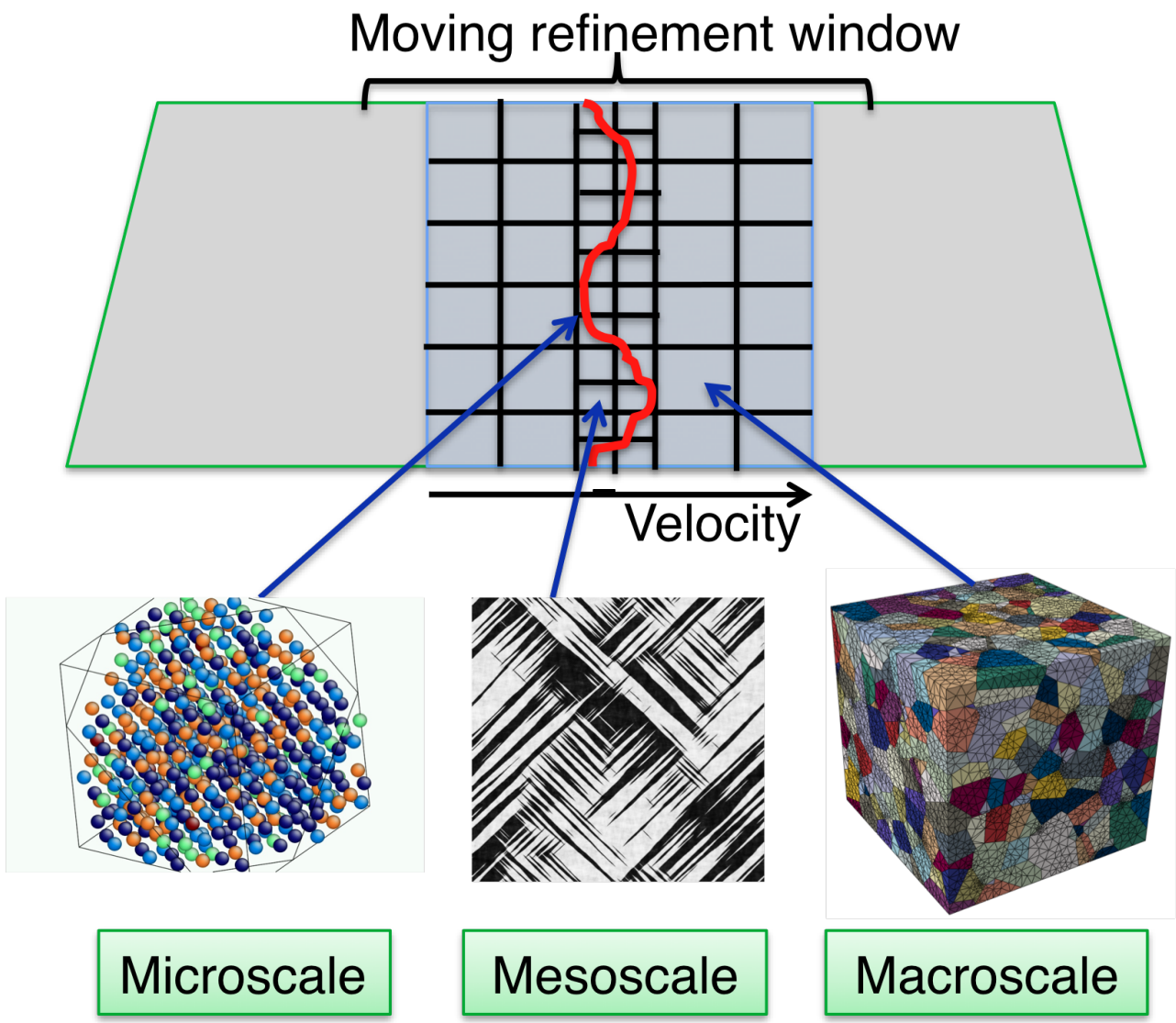
Creation of a functional exascale simulation environment requires our co-design process to be *adaptive, iterative,* and *lightweight* – i.e. agile



Exascale is about better Physics Fidelity: Engineering assessment of material behavior is limited by physics fidelity

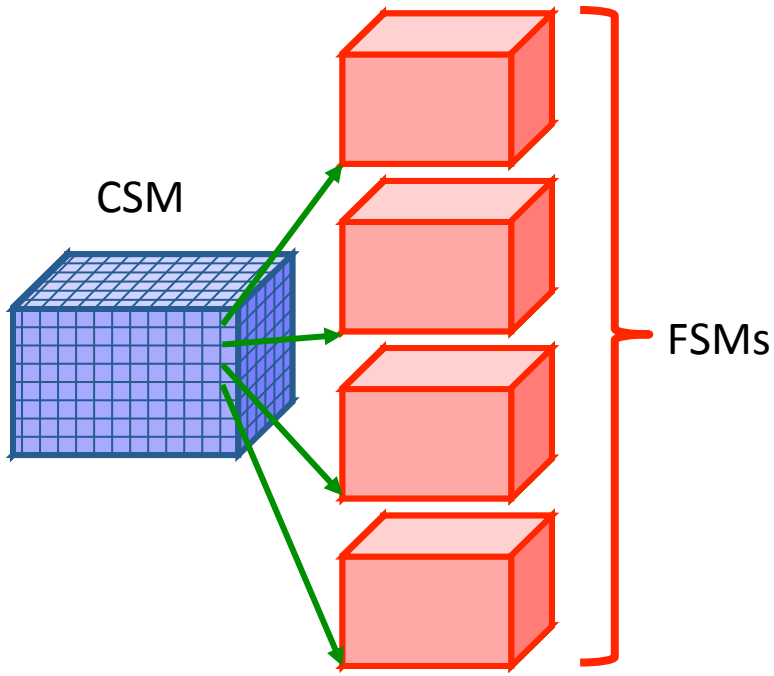
Ab-initio	MD	Long-time	Phase Field	Dislocation	Crystal	Continuum
Inter-atomic forces, EOS, excited states	**Defects and interfaces, nucleation**	**Defects and defect structures**	**Meso-scale multi-phase evolution**	**Meso-scale strength**	**Meso-scale material response**	**Macro-scale material response**
						
Code: Qbox/ LATTE Motif: Particles and wavefunctions, plane wave DFT, ScaLAPACK, BLACS, and custom parallel 3D FFTs Prog. Model: MPI + CUBLAS/CUDA	Code: SPaSM/ ddcMD/CoMD Motif: Particles, explicit time integration, neighbor and linked lists, dynamic load balancing, parity error recovery, and *in situ* visualization Prog. Model: MPI + Threads	Code: SEAKMC Motif: Particles and defects, explicit time integration, neighbor and linked lists, and *in situ* visualization Prog. Model: MPI + Threads	Code: AMPE/GL Motif: Regular and adaptive grids, implicit time integration, real-space and spectral methods, complex order parameter Prog. Model: MPI	Code: ParaDiS Motif: "segments" Regular mesh, implicit time integration, fast multipole method Prog. Model: MPI	Code: VP-FFT Motif: Regular grids, tensor arithmetic, meshless image processing, implicit time integration, 3D FFTs. Prog. Model: MPI + Threads	Code: ALE3D/ LULESH Motif: Regular and irregular grids, explicit and implicit time integration. Prog. Model: MPI + Threads

High fidelity adaptive materials simulation is a direct multi-scale embedding of fine-scale simulation into coarse scale simulation



Direct multi-scale embedding requires full utilization of exascale concurrency and locality

- *Brute force multi-scale coupling:* Full fine scale model (FSM, e.g. a crystal plasticity model) run for every zone & time step of coarse scale mode (CSM, e.g. an ALE code)
- *Adaptive Sampling:*
  - Save FSM results in database
  - Before running another FSM, check database for FSM results similar enough to those needed that interpolation or extrapolation suffices
  - Only run full FSM when results in database not close enough

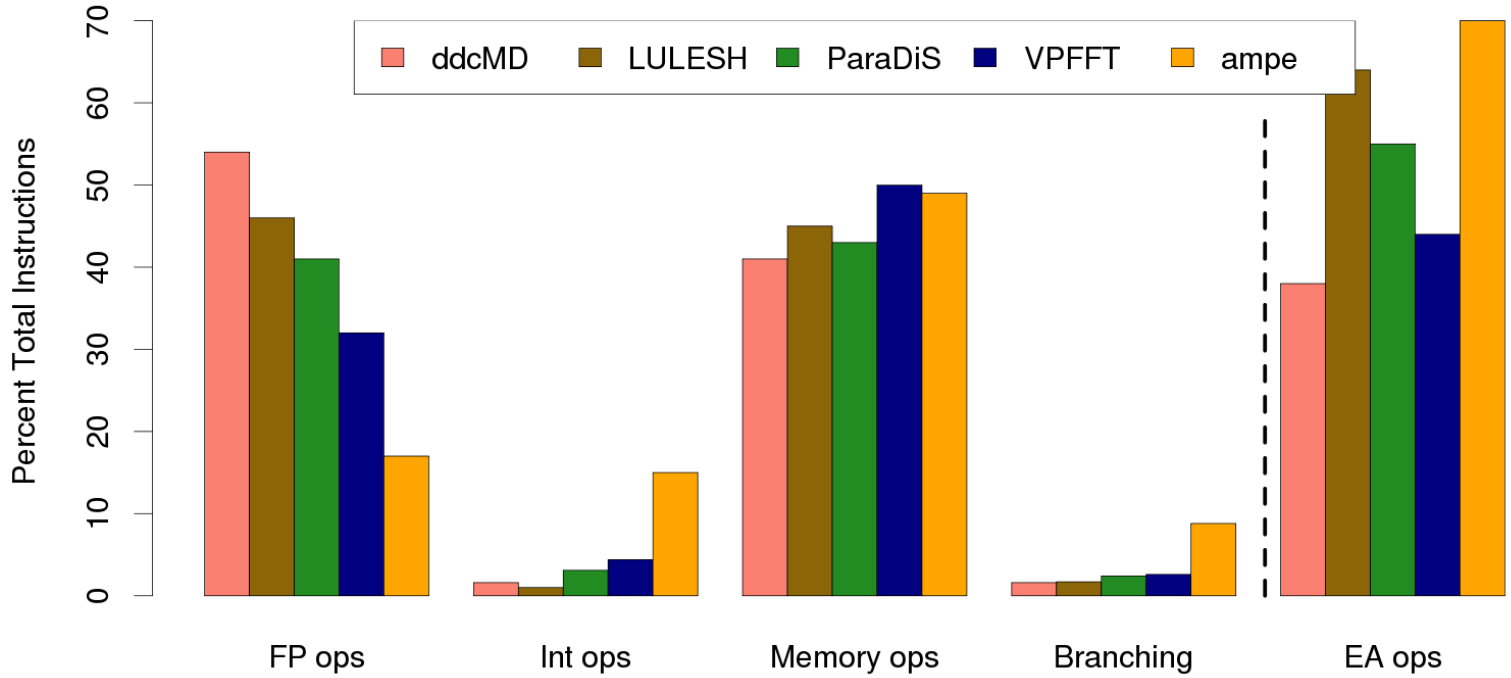


- Heterogeneous, hierarchical MPMD algorithms map naturally to anticipated heterogeneous, hierarchical architectures
- Escape the traditional bulk synchronous SPMD paradigm, improve scalability and reduce scheduling
- Task-based MPMD approach leverages concurrency and heterogeneity at exascale while enabling novel data models, power management, and fault tolerance strategies

Ref: Barton *et.al*, 'A call to arms for task parallelism in multi-scale materials modeling,' *Int. J. Numer. Meth. Engng* 2011; 86:744–764

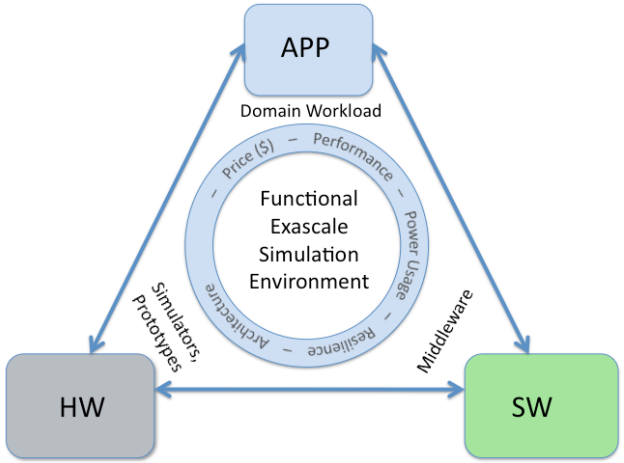
Metrics for computational work measure the behavior of the code within the computational ecosystem (e.g. HW/Stack/Compiler/etc.)

- Pin is a tool that measures utilization of specific functional units in the processor (e.g. floating point operations)
- Both ddcMD and LULESH are highly optimized codes. Pin analysis on entire code suite (see VG 3) in progress
- Analysis for Intel Sandy Bridge processor with Intel compiler (cab)
- LULESH percent vector utilization: Intel compiler = 8.7%, GCC = 0.15% (of FP)



Productive Exascale Simulation requires the coordinated efforts of Domain Scientists, Computer Scientists and Hardware Developers

- Many, many-task coordination issues
  - Greater than one hundred million, more is different
  - Synchronization (essential for time evolution)
  - Stalls (keeping everyone working)
- Better exposure into hardware details for the exascale application developer
  - Compiler Interface
  - Simulators+Emulators+Tools measure code/ecosystem metrics
    - Are we defining the right metrics?

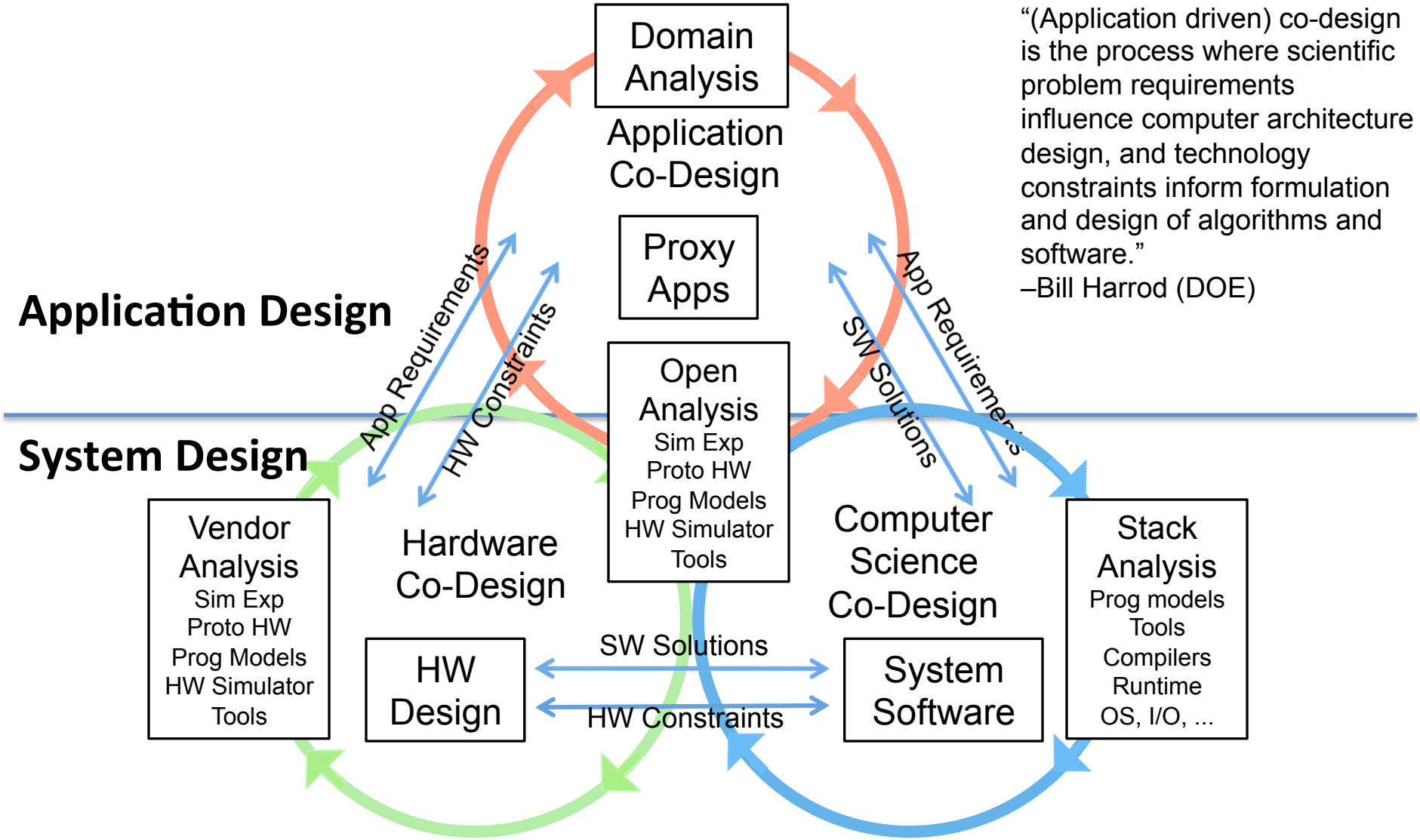


- Application developers need a better way to express (code) the computational work of the application into the exascale computational ecosystem
  - Better programming models (e.g. domain specific languages)
  - Runtime support for heterogeneous multi-program, multi-data (MPMD) applications
- The petascale science apps are NOT general apps. They have been painfully optimized for the petascale architecture by the app developer. How do we get exascale lessons learned into quotidian science applications (VASP, LAMMPs, …)?
- The petascale codes already account for data movement, it is only going to get worse
  - Bandwidth to memory is scaling slower than compute
  - Memory access is dominating power
- The exascale codes will need to learn to adaptively respond to the system
  - Fault tolerance, process difference, power management, …

What did we learn from creating petascale science apps and what does that mean for exascale?

- Problem: Fault tolerance is a problem at 10^5 and will be a much bigger problem at 10^8:
  - Solution: Application assisted error recovery
    - Parity error triggers exception handler (like FPE)
    - Application knows what memory is "important" can catch exception and repair data
  - Exascale runtime will need to support task migration across nodes
- Problem: Scaling (absolutely crucial for exascale) requires very very good load balancing:
  - Solution: Decomposition based on Computational Work
    - Particle-based domain decomposition - processors own particles, not regions - allows decomposition to persist through atom movement
    - Maintain minimum communication list for given decomposition - allows extended range of "interaction"
    - Arbitrary domain shape - allows minimal surface to volume ratio for communication
  - Exascale: decomposition has to become dynamic and adaptive
- Problem: HW specific algorithms are crucial for performance but limit portability
  - E.g. Linked cells map better to current petascale systems than neighbor lists
  - Ordering neighbors within a cell exposes SIMD parallelism
- Problem: I/O does not work with too many files or one large file
  - Solution: Divide and concur, what is the optimal number of files?
  - Exascale: Dedicated checkpoint filesystem (flash?)

Model for the *Workflow of Co-design* between Application Co-design Centers, Vendors, and the broader Research Community



"(Application driven) co-design is the process where scientific problem requirements influence computer architecture design, and technology constraints inform formulation and design of algorithms and software."
–Bill Harrod (DOE)