



Center for Shock Wave-processing of Advanced Reactive Materials (C-SWARM)

Computer Science
Andrew Lumsdaine



Overview

- **Challenges and Goals**
- Exascale Overview
- Overall Computer Science Strategy
- ParalleX and HPX Runtime System
- Active System Libraries

C-SWARM is a Good “CS Problem”

- Problem sizes that can challenge any capability machine
- More than just an “adaptive mesh”
 - Multiple “zones” of computation, with moving boundaries
 - Significantly different computational paradigms in each
 - Data-driven algorithmic feedback to ensure accuracy
- Not just dynamic load-balancing but also dynamic data migration and dynamic control at run-time
- No single current architecture/software model is a good fit
 - Different kernels need different computational mixes
 - Fine-grained variation in load and computation scheduling
 - Significant memory/communication demands

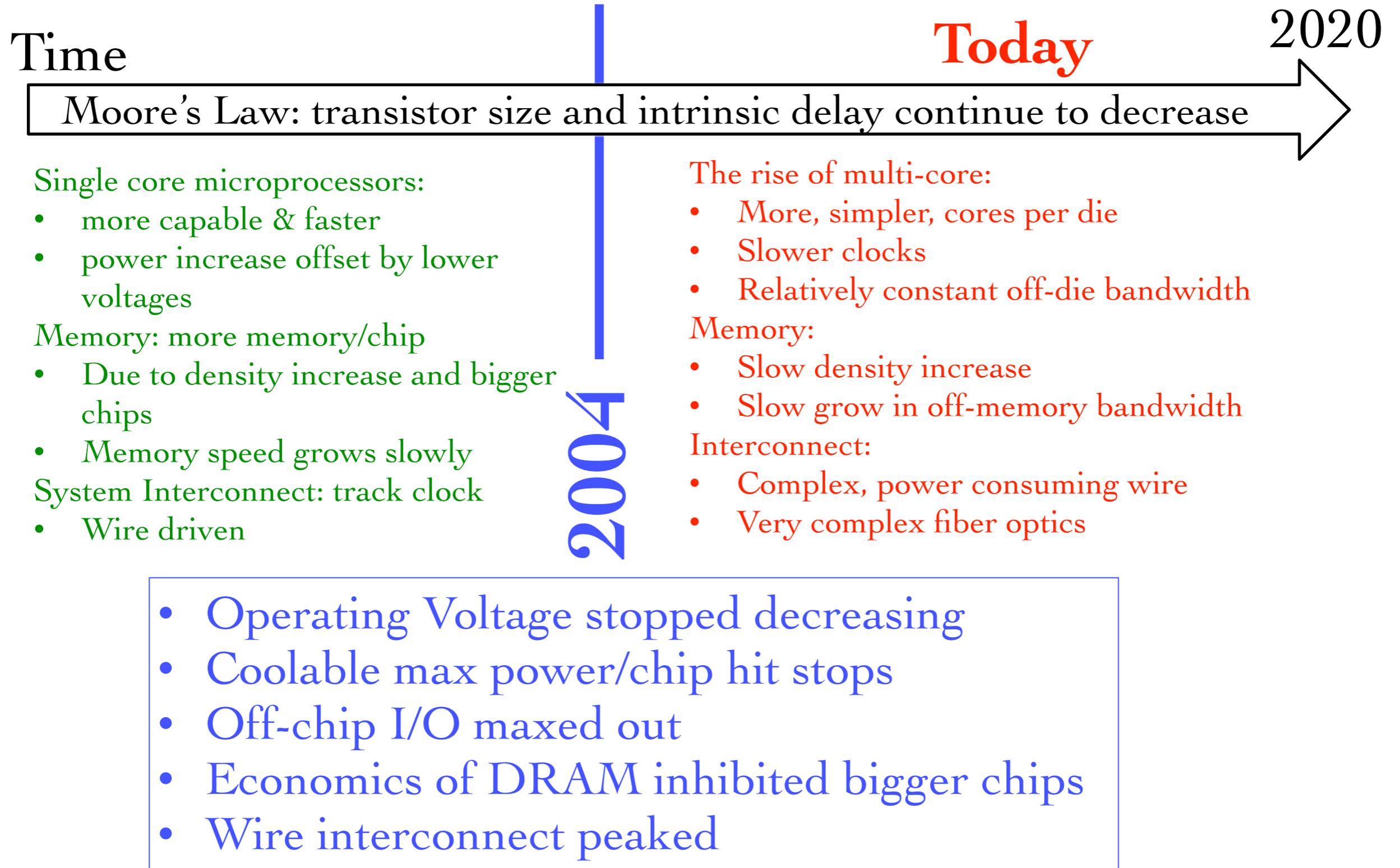
Three Key CS Questions

- How can we guide C-SWARM development to mesh most effectively with future “Exascale” systems?
- How can we manage the highly dynamic C-SWARM computations to run efficiently on such systems?
- How can we simplify the production of the C-SWARM code in ways that provide continuing and long-term improvements in *performance* and *productivity*?

Overview

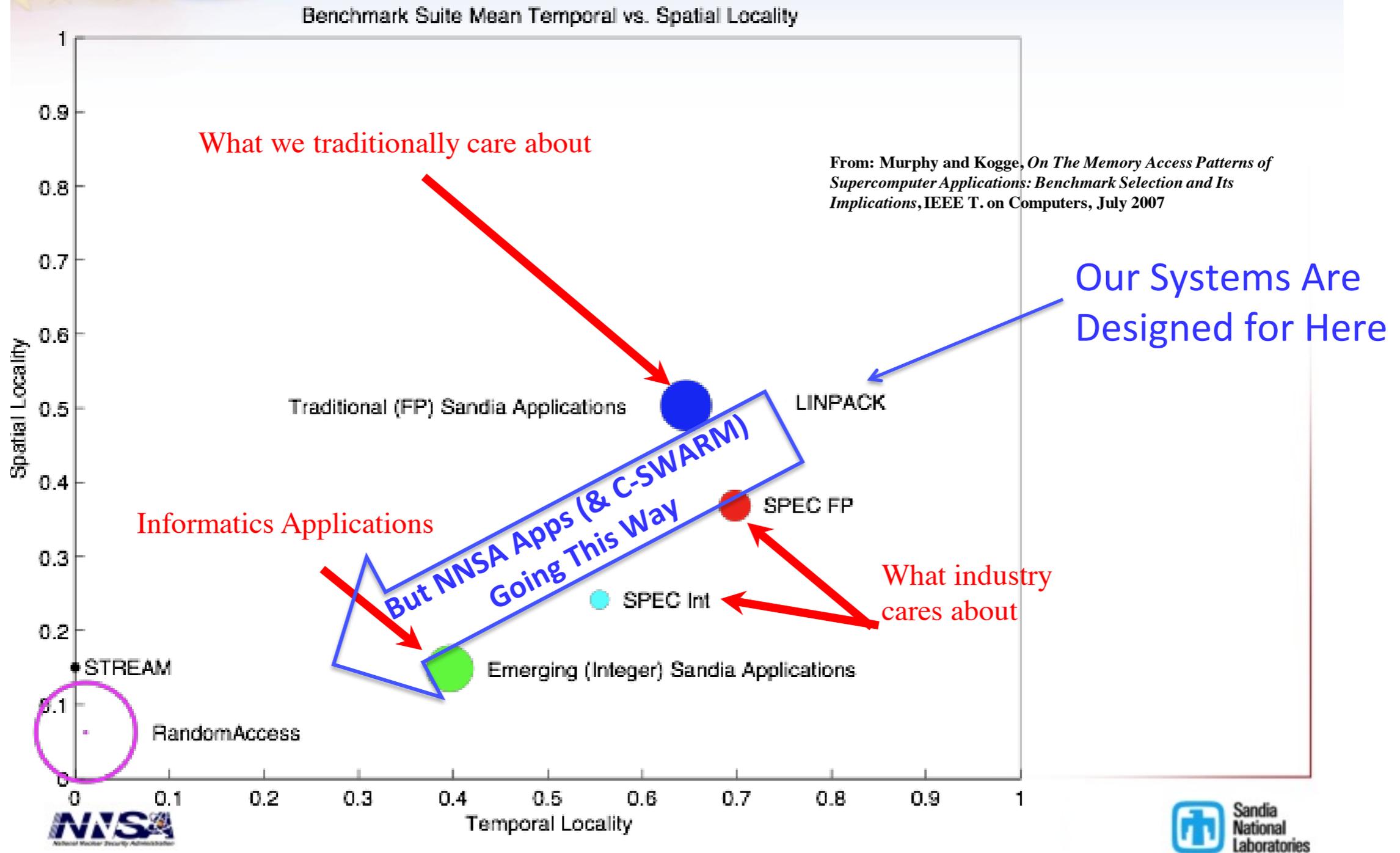
- Challenges and Goals
- **Exascale Overview**
- Overall Computer Science Strategy
- ParalleX and HPX Runtime System
- Active System Libraries

Technology Events Have Radically Changed the Architecture Base of Future Exascale Candidates



Real NNSA Codes are also *NOT* LINPACK

How are applications changing?



Original Chart from R. Murphy, SNL, June 2010

Getting C-SWARM to Exascale

- Target architectures will become
 - *Very* massively parallel
 - *Heavily* heterogeneous
 - With *many* FPUs per core
 - And at best *small* memory and limited per core bandwidth
 - And power dissipation a 1st class design constraint
- At limits of capability machines – 1 billion+ ops must be managed *every cycle*
- *We must* understand key performance metrics and how they scale across emerging architectures
- *We must* understand optimal policies for scheduling and load balancing
- *We must* focus on designing C-SWARM around **memory** and **communications** activity
 - With efficient flops a **secondary concern**

Answering the “Metrics Vs Architecture” Question

- Cross cutting theme in C-SWARM
- Work *in tandem* with algorithm and application designers
- Understand scaling of basic kernels
- Select/instrument/measure metrics that reflect “tall poles” in kernel execution, especially memory & node-node
- Develop architecture-based models for predicting future C-SWARM performance
- Explore alternative policies for scheduling kernels
- Validate against execution on C-SWARM clusters
- Use continuing ties to other DOE/NNSA Exascale initiatives to expand applicability

Overview

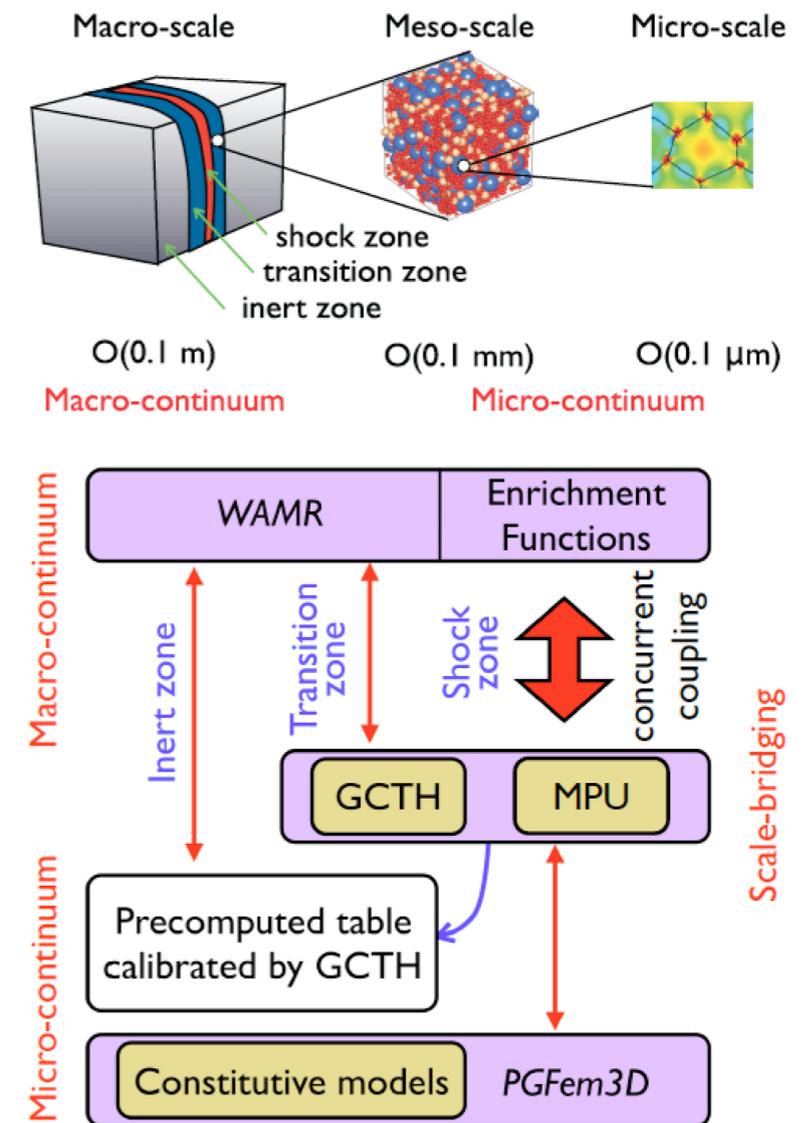
- Challenges and Goals
- Exascale Overview
- **Overall Computer Science Strategy**
- ParalleX and HPX Runtime System
- Active System Libraries

Computer Science Contributions to C-SWARM

- Empower scientists to do science (not computer science)
 - Develop software infrastructure to make C-SWARM applications possible on current and next-generation (Exascale) HPC platforms
 - Efficiency and scalability
 - Productivity and reliability
 - Separate domain science from details of runtime system, computer architecture via advanced software libraries (ASLib) and run-time interfaces (XPI)
- At the same time**

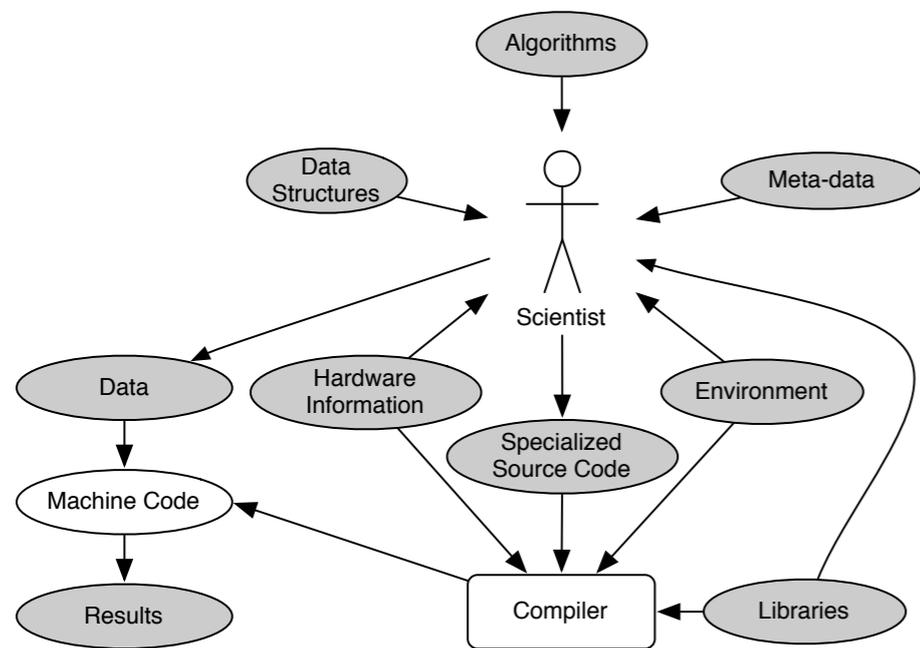
What is the Need?

- Quotes
 - “I spend 80% of my time trying to trick MPI into doing what it doesn’t want to do.”
 - “20% of existing code is physics”
- Our response
 - No tricks needed with HPX runtime – control needs of problem are met by mechanisms of **runtime system**
 - Active System Libraries facilitates **separation of concerns** between science description and machine-oriented optimization

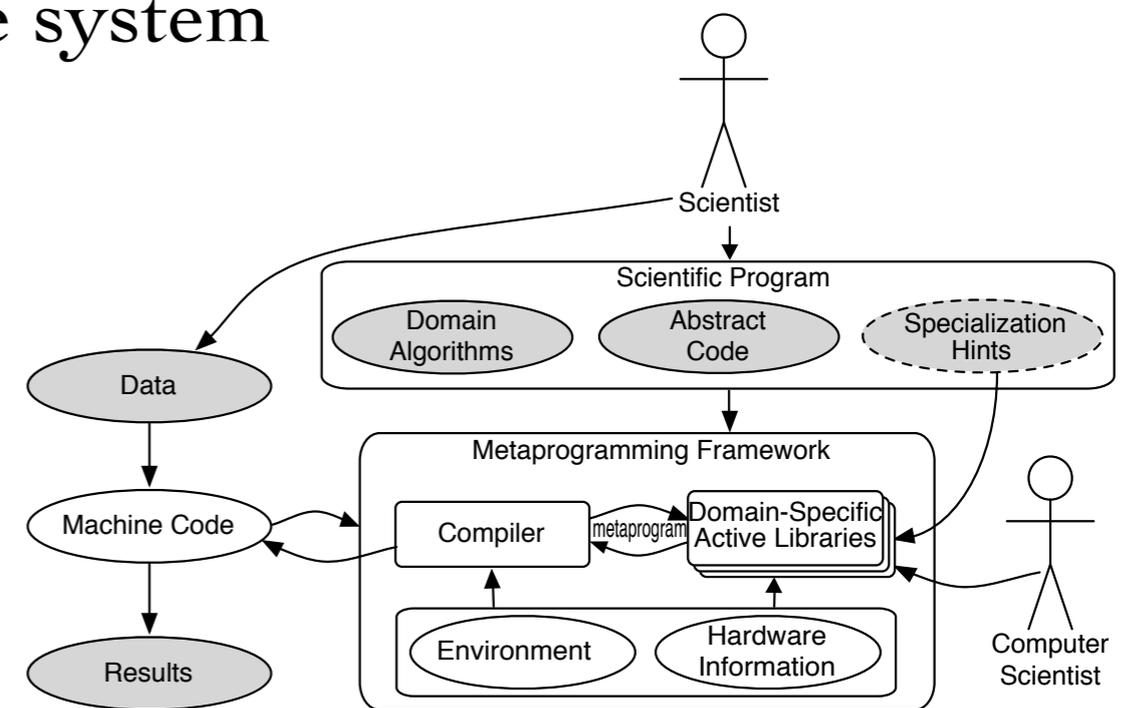
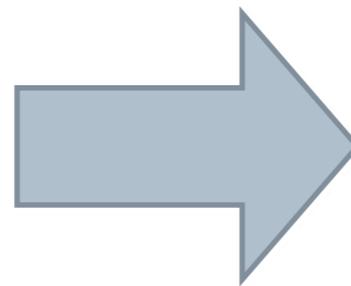


Implementation Model

- Scientist writes software in domain-specific fashion
- Computer scientist specifies mapping from high-level (domain) to low-level (exec target)
- Compiler/Library framework applies mappings
- Runtime system carries out load-balancing, fault tolerance, adaptive tuning/optimization
- H/W accessed via interface to runtime system



Conventional



C-SWARM

Computer Science Strategy

- Apply and extend advanced **execution model** with essential properties for computational challenges
 - For dramatic improvements in efficiency and scalability
- Deliver **runtime system** for dynamic resource management and task scheduling
 - Exploit runtime information for continuous and adaptive control
- Develop domain-specific hierarchical **programming interface**
 - For rapid algorithm development and testing
 - To enable separate optimization (parameterization and composition)
 - Support UQ and V&V
- Phased development for immediate impact on project and long term improvements in performance and productivity
- Leverage on-going research for mutual benefit of DOE programs

Concrete Computer Science Deliverables

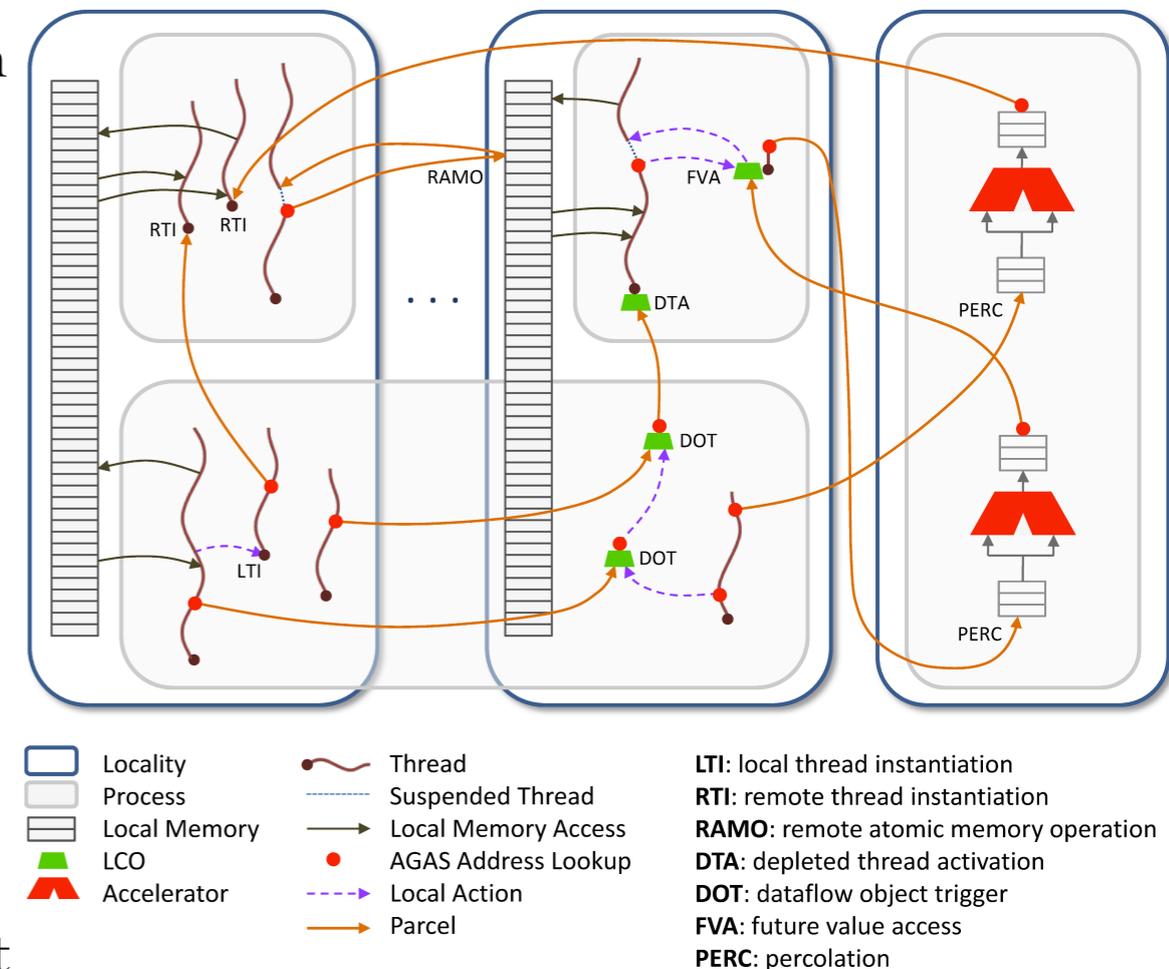
- ParalleX execution model
 - Cross-cutting guiding principles for co-design of application and system software
- HPX runtime system
 - Dynamic adaptive resource management and task scheduling
 - Advanced control policies and parallelism discovery
- XPI low-level programming interface
 - Stable interface to HPX
 - Target for high-level programming methods
- ASLib(s) – Separation of concerns
 - Generic libraries (compositional)
 - Domain-specific language (DSL)
 - Semantics necessary for physics, hiding system-specific issues
 - Optimization framework

Overview

- Challenges and Goals
- Exascale Overview
- Overall Computer Science Strategy
- **ParalleX and HPX Runtime System**
- Active System Libraries

A Quick ParalleX Overview

- Localities – Synchronous Domains
- AGAS – Active Global Address Space
- ParalleX Processes – with capabilities protection
- Computational Complexes – threads & fine grain dataflow
- Local Control Objects – synchronization and global distributed control state
- Distributed control operation – global mutable data structures
- Parcels – message-driven execution and continuation migration
- Percolation – heterogeneous control
- Micro-checkpointing – compute-validate-commit
- Self-aware – introspection and declarative management



HPX Runtime System

- ParalleX execution model provides conceptual framework for C-SWARM software co-design and integration
 - Attacks performance degradation for efficiency and scalability
 - Starvation, latency, overhead, contention, uncertainty of asynchrony
- Addresses critical C-SWARM needs for dynamic adaptive system software
 - Provides support for multi-scale multi-physics time-varying application
 - Dynamic adaptive resource management and task scheduling for unprecedented efficiency and scalability
 - Optimized scheduling and allocation policies
 - Reliability through micro-checkpointing and compute-validate-commit cycle
- XPI low-level programming interface
 - Low-risk means of building C-SWARM on top of HPX
- Leverages and complements prior and ongoing work of other programs

HPX Addresses C-SWARM Computational Needs

- Starvation
 - Discover parallelism within adaptive runtime data structures
 - Lightweight dynamic threads deliver new parallelism relative to static MPI processes
 - Load balance to match algorithm parallelism to physical resource
 - Pipeline to overlap success phases of computation
- Latency
 - Overlap computation and communication (multithreading)
 - Message-driven computation (move work to data)
- Overheads
 - Active global address space
 - Powerful but lightweight synchronization constructs
 - Lightweight thread context switching

Overview

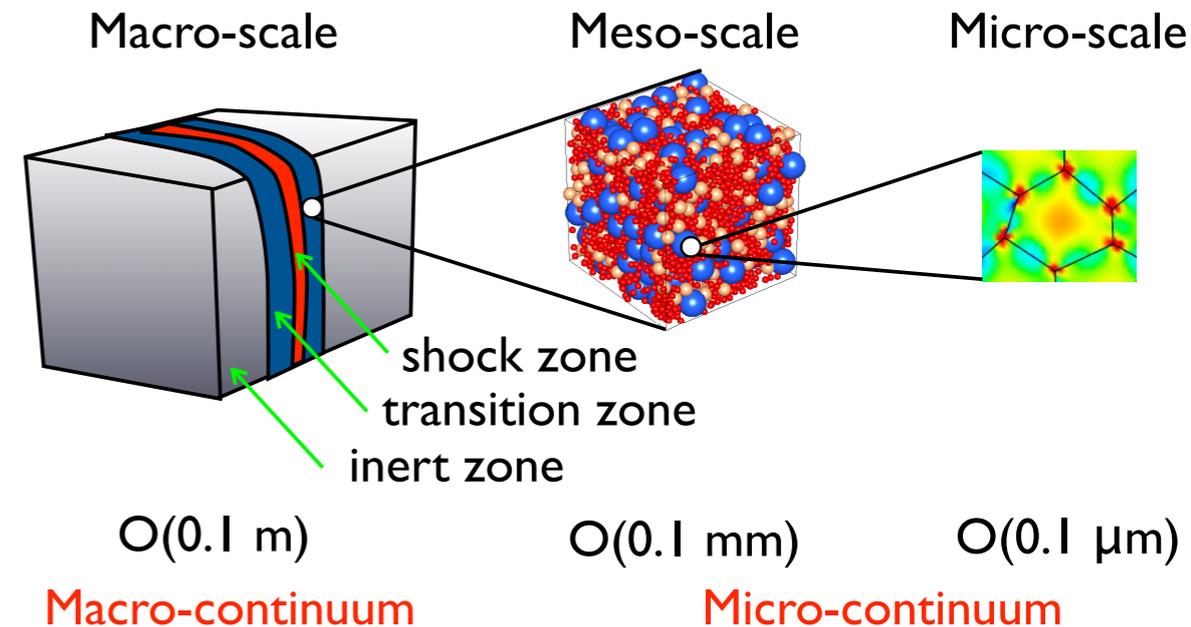
- Challenges and Goals
- Exascale Overview
- Overall Computer Science Strategy
- ParalleX and HPX Runtime System
- **Active System Libraries**

Enabling Software Technology

- Maxwell's compatibility criterion
 - Retains Lagrangian representation in *WAMR* solver
- Enhance GFEM for *PGFem3D*
 - Coupled with Lagrangian face-offsetting tracking method
 - Surfaces represented explicitly
- Immersed Boundary Method for *WAMR*
 - Unstructured triangulated surface grid
 - Riemann problem in a local coordinate system
- Enhance Multi-time / Multi-domain Geometric Integrators
 - Mixed mode of integration (explicit & implicit)
 - Asynchronous time steps
- Enhance GCTH and Develop MPU
 - Nested coupling algorithm between M&m

Computational and Computer Science Synergies

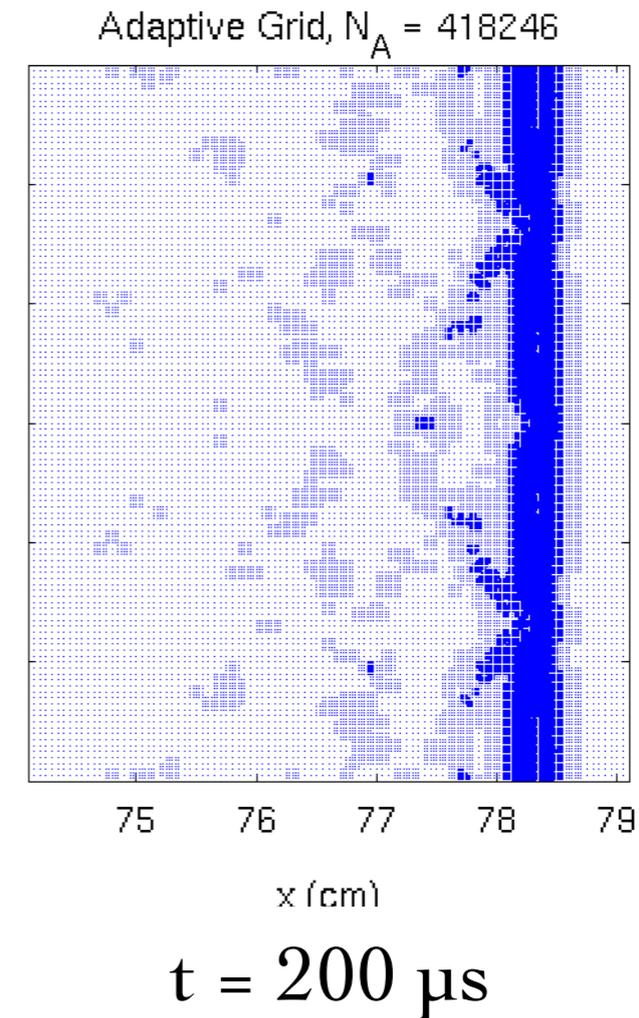
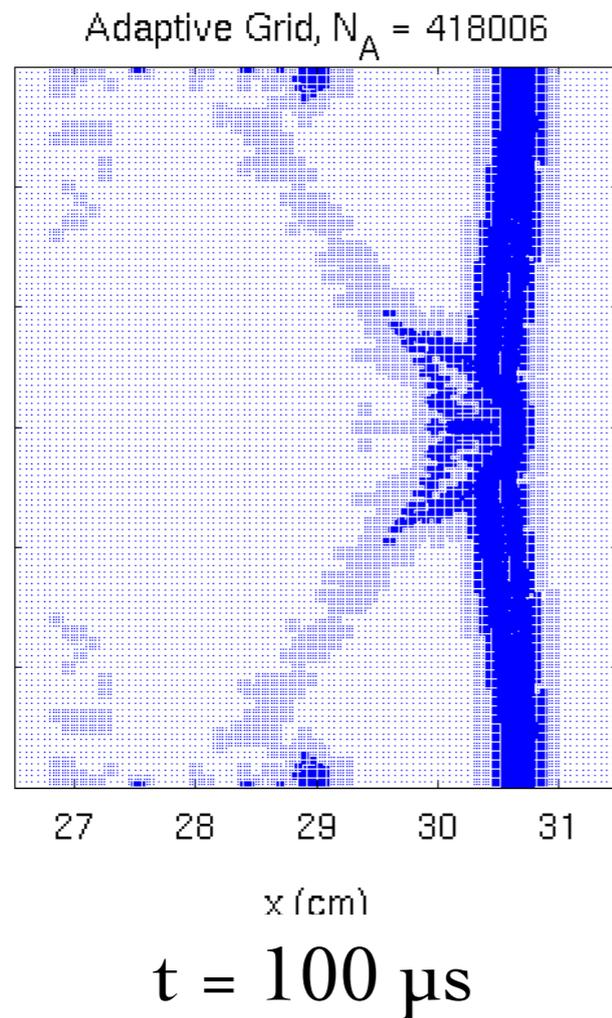
► BSP-based programming with MPI is static and inefficient. Time is spent on data-structures, domain decomposition, load balancing, communications, etc.



- Dynamic adaptive resource management and task scheduling - ParalleX execution model
- Reliable runtime system - HPX embodiment of ParalleX
- Productive code development - XPI and ASLibs
- Starvation, latency, overhead, contention, uncertainty of asynchrony, etc.
- C-SWARM software suite reliability

Computational and Computer Science Synergies

■ Starvation



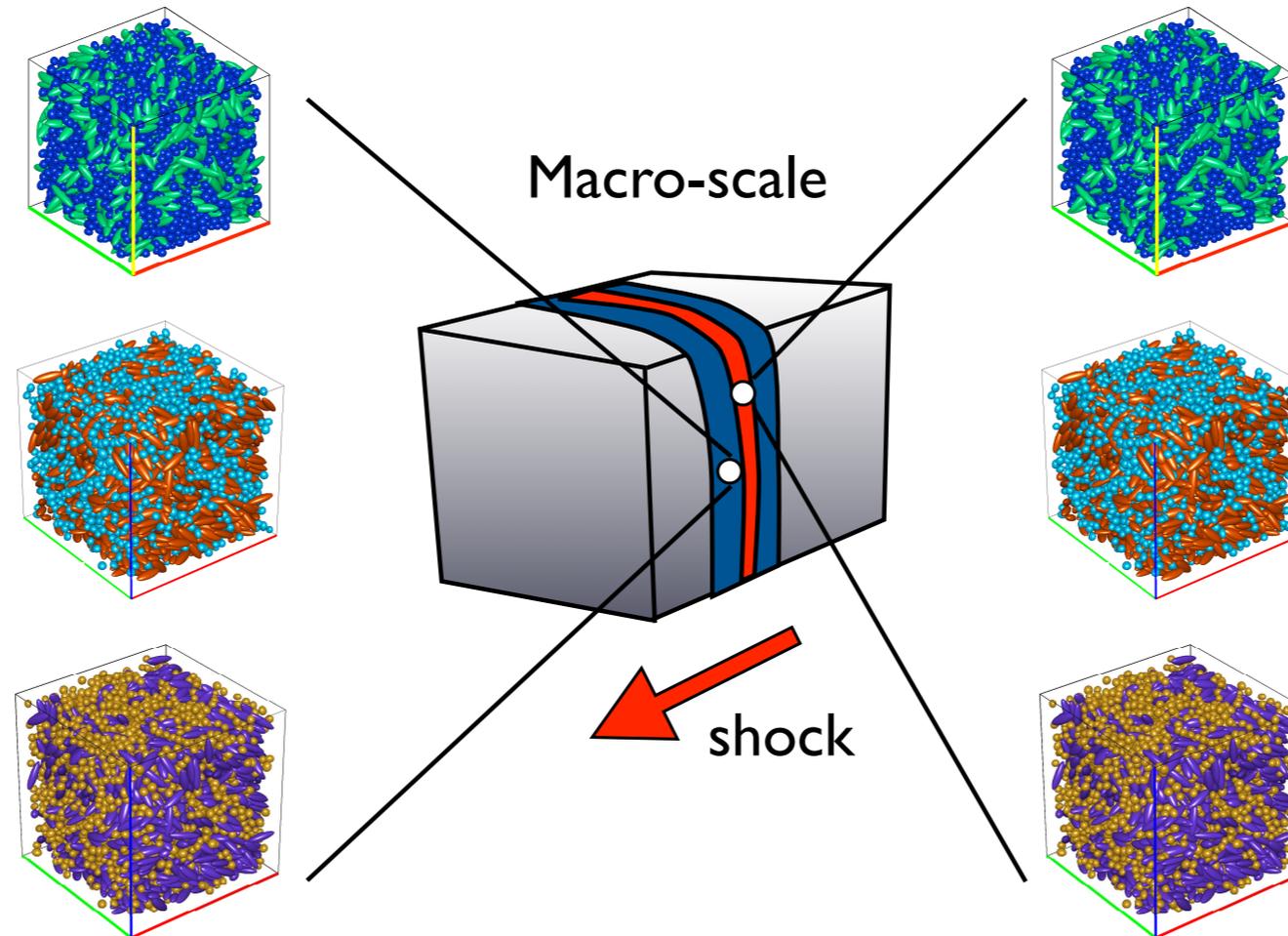
- Number and position of collocation points is not known *a priori*
- Extensive use of adaptive space/time/model refinements
- ▶ Lightweight user threads
- ▶ Exploiting new forms of parallelism

Computational and Computer Science Synergies

■ Latency

Meso-scale ensemble 1

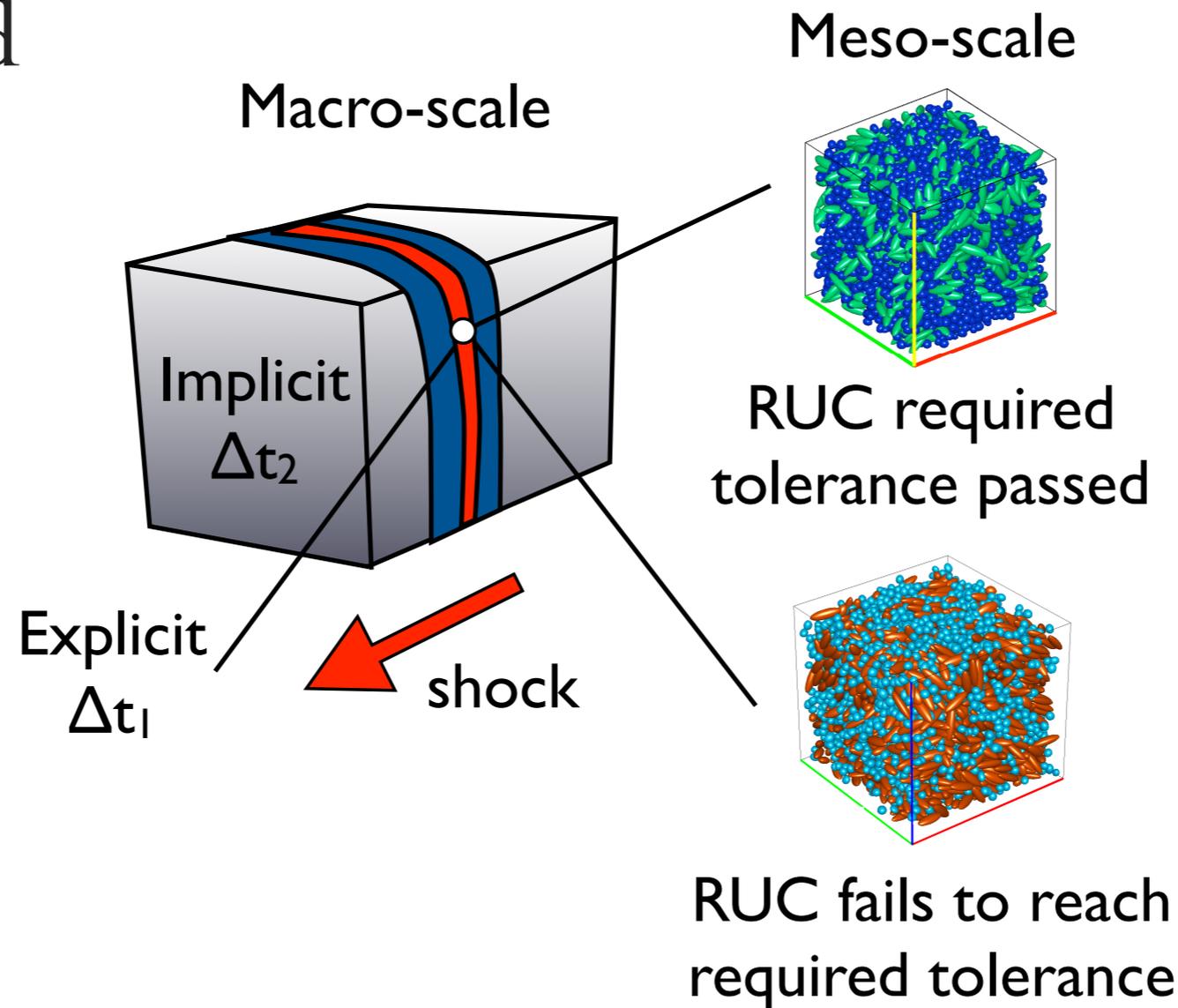
Meso-scale ensemble 2



- Order of solution of cells is not known *a priori*
- Concurrent coupling based on nested iterations
- ▶ Multi-threaded local execution control
- ▶ Message-driven computation that moves work to data

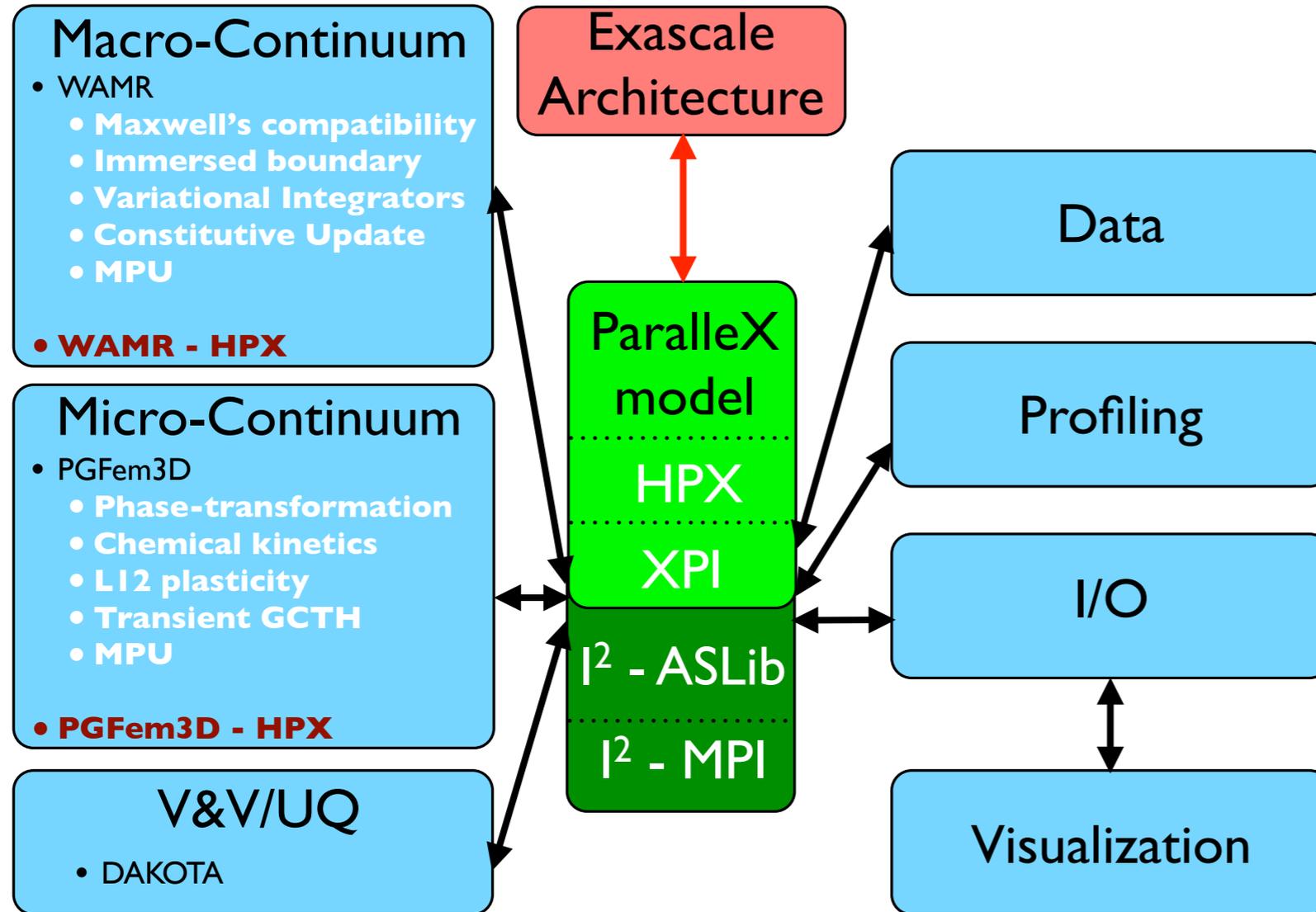
Computational and Computer Science Synergies

■ Overhead



- Asynchronous time integration (explicit and implicit)
- Error control in space time and constitutive update
- ▶ Light-weight synchronization like dataflow
- ▶ Percolation, a form of controlled workflow management

Software Integration Framework



I²-MPI/I²-ASLib/I²-XPI

high-level abstractions
of interface data

- physics-aware
- input data-aware
- numeric-aware
- V&V/UQ-aware
- visualization-aware

- Typically solvers exchange data horizontally
- We propose vertical coupling across scales
- Modularity preserved with exceptional scaling properties

► Similar to the Roccom module - ASC CSAR center

Active System Libraries

- Metaprogramming-based interfaces to low-level libraries
 - Rather than just calling functions, library and calling code influence each other
 - Code generation and customized compilation enable usability and performance
- Benefits:
 - Check uses of library and better diagnose errors
 - Configure library to specific computer system, use case
 - Domain-specific notation
 - Optimize code using library
- Techniques:
 - C++ template metaprogramming (cf. Parallel Boost Graph, AM++, STAPL, ...)
 - Compiler infrastructures (e.g., ROSE, LLVM)
 - Run-time code generation (e.g., SEJITS, CorePy)

Progressive Implementation Strategy

- Basic domain specific libraries layered on XPI
- Generic programming approach to separate concerns
 - Allow simultaneous development of M&m and HPX
 - Provide declarative semantics of M&m codes
 - Bridge abstraction-performance divide
 - Provide “insulation layer” between the C-SWARM framework, the HPX runtime system, and the hardware
- Evolve to “DSL”
- Reliability through micro-checkpointing and compute-validate-commit cycle



Center for Shock Wave-processing of Advanced Reactive Materials (C-SWARM)

Exascale Plan
Andrew Lumsdaine

