# Evolutionary Support for Revolutionary Programming Models and Runtime Systems

Pavan Balaji

Argonne National Laboratory

## Presentation Layout

- **State of Programming Models and Runtime Systems**
- Application Requirements for the Exascale Era
- (Example) Key Challenges for Interoperable Runtimes
- Concluding Remarks

## Current State of Programming Models and Runtime Systems

- (Too) Many programming models and runtime systems have been proposed recently
  - Why? No single model seems to provide everything applications need
    - Diverse application needs dictate many programmability constraints
  - Each model provides unique capabilities, but comes with its set of challenges as well
- Several categories of these exist:
  - High-level Compilers/Languages
    - UPC, Chapel, CAF, X10, …
  - High-level Libraries
    - Global data space models (Global Arrays, Global Trees) and Global computation space models (ADLB, Scioto, Charm++)
  - Low-level Runtime Systems
    - MPI, ARMCI, GASNET, OSPRI, accelerator models (OpenCL, CUDA), …

## Usage of Programming Models and Runtime Systems

- Many of these models have aimed at “revolutionizing” the programming interfaces for applications (not interoperable with legacy code)
  - Provide a rich set of capabilities
  - Allow applications to easily express their requirements
- So what’s stopping applications from using these models?
  - Biggest drawback: there is no transition path for existing applications!
    - Very few applications will be written from scratch (without reusing anything from the past: e.g., math libraries, load balancing tools)
    - Incremental transition is critical for real applications; validation and verification done on application codes is too expensive to throw away
  - “Revolutionary” programming models are, well, “revolutionary”
  - Many of these models require almost a complete rewrite of applications

## Multi-model Programming Might be the Future

- For multimodule applications primarily based on MPI, how can a new module be written using alternate models such as UPC or CAF in such a way that it can interoperate with the rest of the application?
- How can an application written in Cray Chapel or IBM X10 utilize math libraries written in MPI, such as PETSc, that have had close to a hundred man-years of development invested in them?
- Can an MPI application directly move data from a local accelerator device to another physical node without explicitly using accelerator programming models to stage data locally before using MPI to move it outside the node?
- Can OpenMP and Intel TBB co-exist within the same application?
- If you have an ADLB or Charm++ application using work stealing and task migration, can it interact with Global Arrays to provide a globally accessible data region?

## Application Usage

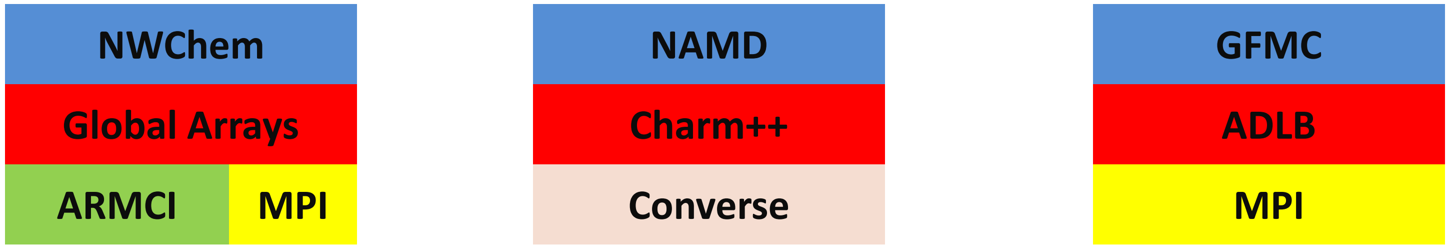
- Applications have so far relied on more-or-less a single model
  - Most applications use MPI (either directly or through high-level domain-specific libraries); many moving to hybrid MPI+OpenMP model
  - Some applications use alternate models such as Global Arrays (NWChem) or UPC (NSA applications)
- As we move forward to exascale, applications will need more!
  - While the programming models that exist today lack capabilities to handle exascale challenges, we are not yet at a point where we need a completely new model
    - Each model has its flaws, but each model has its strengths too
    - Each model is very good at the set of things it is built for
  - Instead of redesigning a completely new programming model, we should leverage the strengths of the different models

## Runtime Challenges for Interoperability

- Unfortunately, using multiple programming models is not possible today
  - Programming models are not interoperable today because their runtime systems do not cooperate
  - UPC and CAF use the GASNet runtime system; Global Arrays uses ARMCI; MPI uses its own internal runtime system; OpenMP and TBB uses their own separate thread management layers
  - Impossible to inter-mix these different runtime systems without they knowing of each other
    - Resource conflicts
    - Progress deadlocks
    - Data corruption because of data access contention

## Current State: A Separate Runtime System for each Application

- Each application packaged with its own high-level programming library (GA, Charm++, ADLB, MADNESS runtime) on top of a different low-level runtime (MPI, ARMCI, GASNET)
- This model is fundamentally not sustainable at Exascale
  - *Interoperability between application models is difficult* – underlying runtime infrastructure has to be either interoperable or integrated
  - *Research optimizations are either redundant or not interoperable*
    - GA, GT, Data Spaces, etc., mostly do the same optimizations
    - For what's not repeated (e.g., if GA does something DS doesn't), they are not interoperable
  - *Commercial support impractical* – vendors will not support five runtime libraries – hard enough to get support for MPI + <anything else>



## Presentation Layout

- State of Programming Models and Runtime Systems
- **Application Requirements for the Exascale Era**
- (Example) Key Challenges for Interoperable Runtimes
- Concluding Remarks

## Application Requirements for the Exascale Era

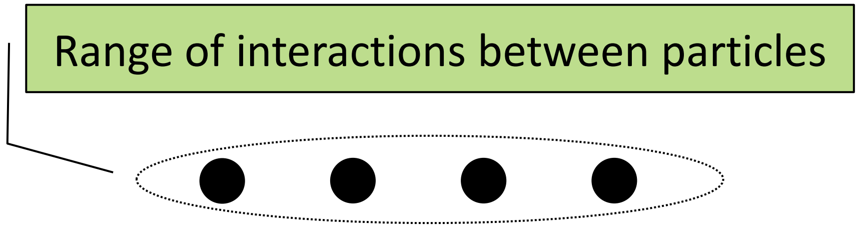
- Applications need to deal with two dimensions of problems:
  - The science they are trying to solve is becoming more complex (hence the need for exascale computing)
    - More data requirements, more computation
  - Hardware architectures are becoming more complex (hierarchical architectures, heterogeneous systems)
    - Current machines cannot just scale up because of cost and power constraints
- Current computation and communication methodologies used by applications cannot just migrate to exascale architectures
  - Too many variables here; everything will not magically scale

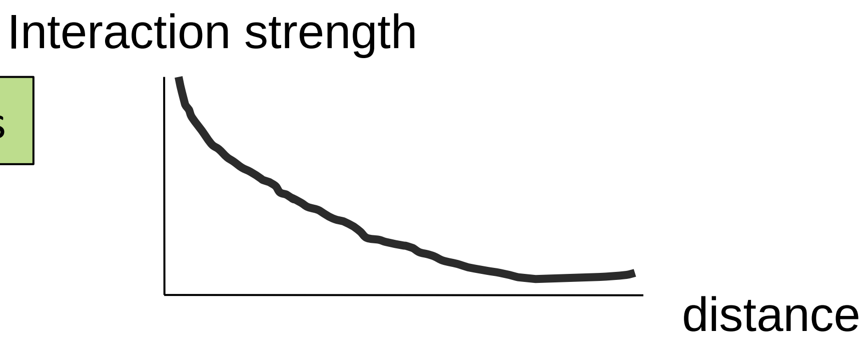
## N-Body Coulomb Interactions

- Current applications have been looking at small-to-medium molecules consisting of 20-100 atoms
  - Amount of computation per data element is reasonably large, so scientists have been reasonably successful decoupling computation and data movement
- For exascale systems, scientists want to study molecules of the order of a 1000 atoms or larger
  - Coulomb interactions between the atoms is much stronger in the problems today than what we expect for exascale-level problems
  - Larger problems will need to support short-range and longer-range components of the coulomb interactions (possibly using different solvers)
    - Diversity in the amount of computation per data element is going to increase substantially
    - Regularity of data and/or computation would be substantially different

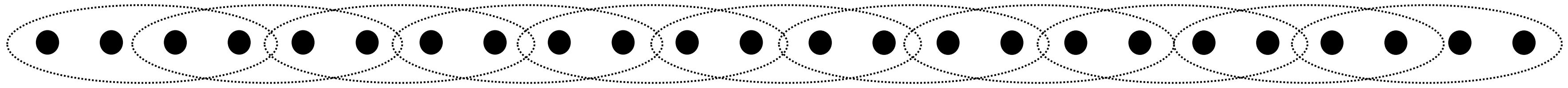
# Quantum mechanical interactions are near-sighted (Walter Kohn)

Traditional quantum chemistry studies lie within the nearsighted range where interactions are dense:





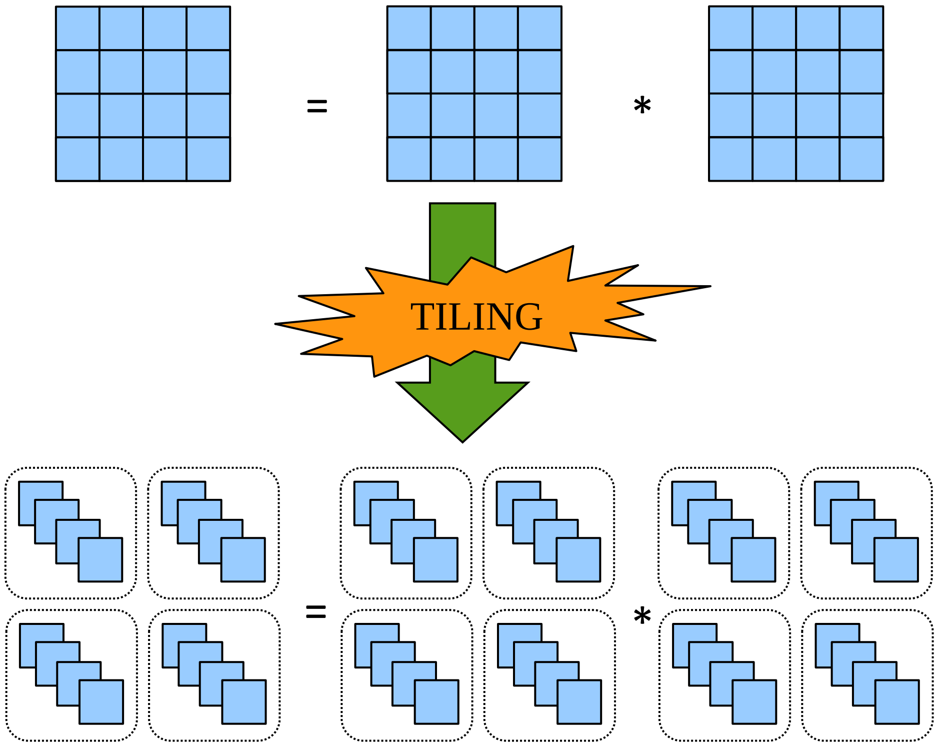
Future quantum chemistry studies expose both short- and long-range interactions:



Note that the figures are phenomenological.  Quantum chemistry methods treat correlation using a variety of approaches and have different short/long-range cutoffs.

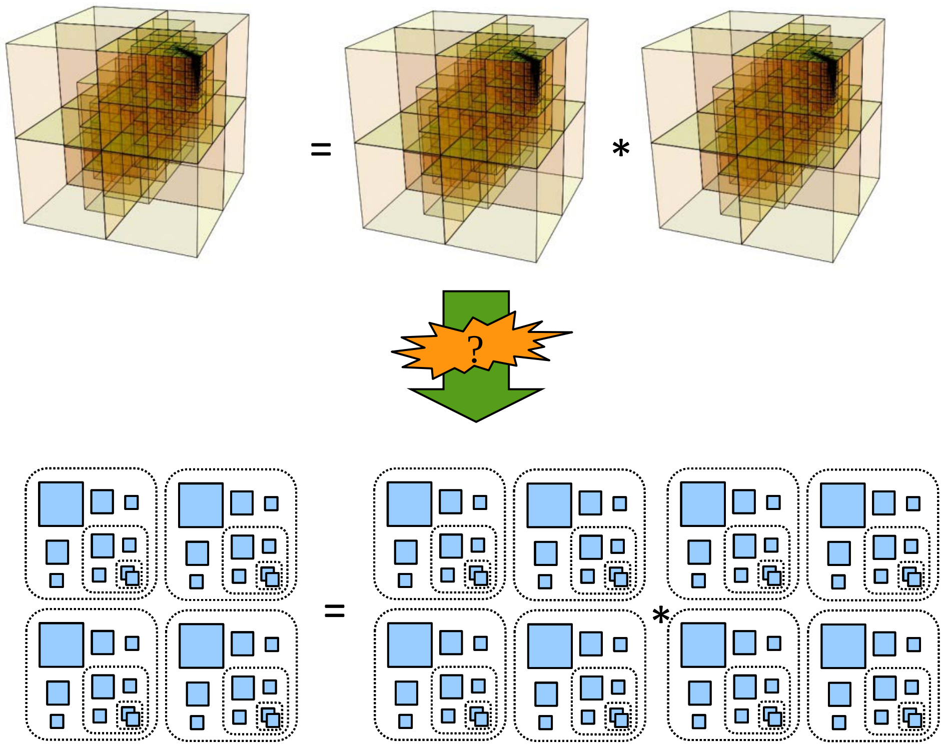
*Courtesy Jeff Hammond, Argonne National Laboratory*

## Current: Regular Dense Computation



- Traditional models such as MPI or GA alone have been sufficient for this model of computation
  - Fetch data locally and compute

## Exascale: Irregular Dense/Sparse Computation



- Traditional models "individually" are no longer sufficient
  - MPI or GA like model is good for dense parts of the data (fetch data locally and compute)
  - Charm++, ADLB or Scioto like model is good for the sparse parts

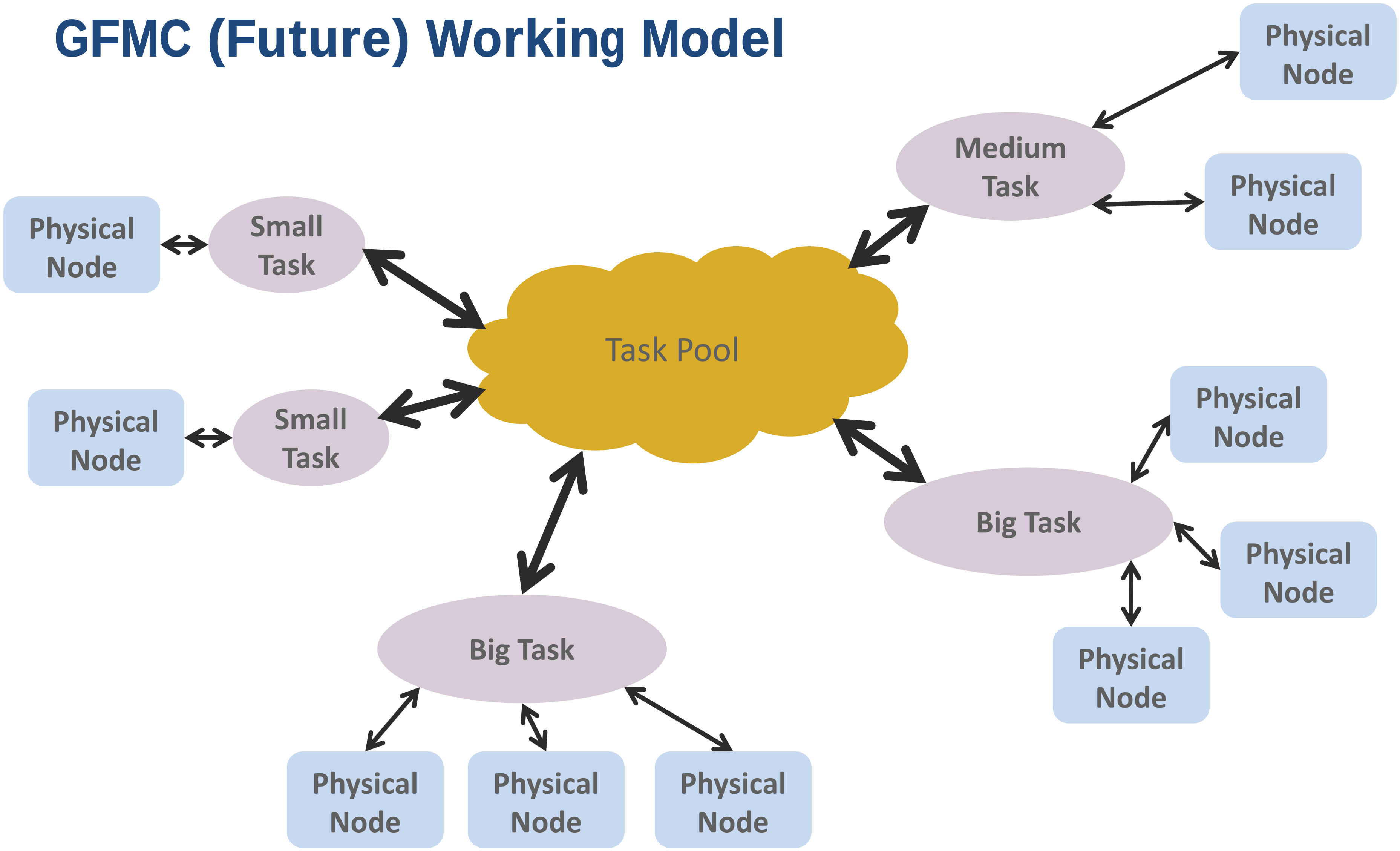
## Another Motivating Example: GFMC

- Green's Function Monte Carlo -- the "gold standard" for *ab initio* calculations in nuclear physics at Argonne
  - A non-trivial master/slave algorithm, with assorted work types and priorities; multiple processes create work; large work units
  - Uses ADLB for task management, the Asynchronous Dynamic Load Balancing Library (written in MPI)
- Scaled to 2000 processors on BG/L a little over two years ago, then hit scalability wall
- Need to get to 10's of thousands of processors at least, in order to carry out calculations on $^{12}C$, an explicit goal of the UNEDF SciDAC project
- The algorithm has had to become even more complex, with more types and dependencies among work units, together with smaller work units

## Memory Scalability of GFMC

- GFMC's view of ADLB is that of a "generalized master-worker"
  - Each worker provides tasks to the "master" (physically distributed set of servers), and other workers can steal this work
  - Issues related to task dependencies/load-balancing are handled within ADLB (GFMC gives hints, but doesn't explicitly handle it)
- As GFMC moved to larger elements, the memory available to each task was no longer sufficient (factorial of atomic weight)
- First solution was MPI + OpenMP: allowed GFMC to scale to C-12
- Next steps forward are C-14 and O-16, and a simple task-based model such as ADLB is no longer sufficient
  - We need to investigate using ADLB in conjunction with GA or UPC, …
  - MPI to move data within an address space, but GA or UPC to expand the address space available to each process (global space)

# GFMC (Future) Working Model

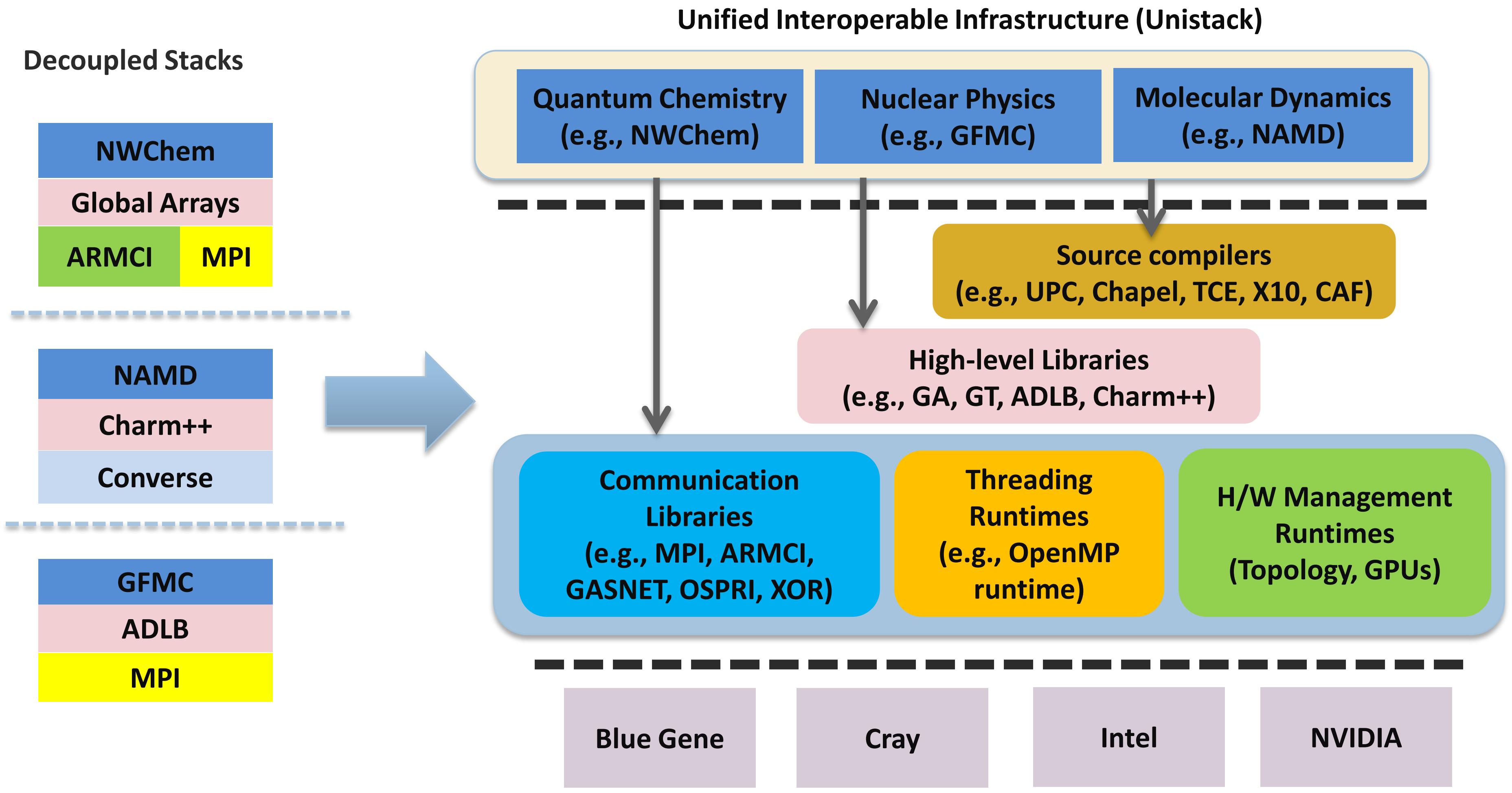


*Some mixture of high-level task-parallel model (ADLB, Charm++) in conjunction with some form of global data space model (GA, UPC, CAF) would be required to scale GFMC to the next problem of interest: Oxygen-16*

## Presentation Layout

- State of Programming Models and Runtime Systems
- Application Requirements for the Exascale Era
- **(Example) Key Challenges for Interoperable Runtimes**
- Concluding Remarks

# Unistack: One Possible Model for a Unified and Interoperable Programming Infrastructure


Decoupled Stacks
NWChem
Global Arrays
ARMCI
MPI
NAMD
Charm++
Converse
GFMC
ADLB
MPI
Unified Interoperable Infrastructure (Unistack)
Quantum Chemistry (e.g., NWChem)
Nuclear Physics (e.g., GFMC)
Molecular Dynamics (e.g., NAMD)
Source compilers (e.g., UPC, Chapel, TCE, X10, CAF)
High-level Libraries (e.g., GA, GT, ADLB, Charm++)
Communication Libraries (e.g., MPI, ARMCI, GASNET, OSPRI, XOR)
Threading Runtimes (e.g., OpenMP runtime)
H/W Management Runtimes (Topology, GPUs)
Blue Gene
Cray
Intel
NVIDIA

**The key is to provide a unified and interoperable architecture with multiple levels of capabilities and ALLOW APPLICATIONS TO BREAK THE LAYERING → transition path for applications!**

## Several Challenges in Allowing Multiple Programming Models to Co-exist

- Co-existing is hard! Semantics, not just programming them
- Defining "interoperability"
  - Using data from one model in a different model is messy
    - But often cannot be avoided: Data is King!
  - Completion semantics, safe use of data buffers
  - Using data objects on other models: MPI using GASNet allocated buffers
- Resource contentions
  - Buffer management, progress semantics (asynchronous agents) to avoid deadlocks/livelocks, compute resources (e.g., using OpenMP and TBB in the same applications)
- Tools
  - Debugging with one model is hard enough
- And many others!

# This is not a completely new concept!

- Different forms of ad-hoc interactions do exist
- MPI + OpenMP (or other threading models)
  - Well defined in the specifications
  - Many optimizations done by researchers all around the world
- MPI + UPC
  - Hard problem because MPI is a runtime library, and UPC has compiler capabilities; some work done by various groups in this area
- ADLB + MPI
  - (Almost) trivial interoperability because of a layered model (ADLB is layered on top of MPI)
- *But we need a more truly interoperable (drag-and-drop) model*
  - *Migration path for applications to start using other models (any other model!) in conjunction with what they are already doing*

## Presentation Layout

- State of Programming Models and Runtime Systems
- Application Requirements for the Exascale Era
- (Example) Key Challenges for Interoperable Runtimes
- **Concluding Remarks**

## Concluding Remarks

- Several programming models out there, but many of them are too "revolutionary" for applications to move to them
  - Too much initial migration effort required
- We need to make the path to these "revolutionary" models more "evolutionary"
  - The jump cannot be so drastic
- Interoperability with what exists is the key!
  - While there has been some work that performs ad-hoc interactions between select model, we need a focused effort in combining the capabilities of many (or all) of these models
  - Applications should be able to pick and choose what they want to use based on application characteristics and requirements