Parallel Programming models in the era of multi-core processors:

Laxmikant Kale

http://charm.cs.uiuc.edu

Parallel Programming Laboratory

Department of Computer Science

University of Illinois at Urbana Champaign

Requirements

- Composibility
- Respect for locality
- Dealing with heterogeneity
- Dealing with the memory wall
- Dealing with dynamic resource variation
  - Machine running 2 parallel apps on 64 cores, needs to run a third one
  - Shrink and expand the sets of cores assigned to a job
- Dealing with Static resource variation : *Fwd Scaling*
  - I.e. Parallel App should run unchanged on the next generation manycore with twice as many cores
- Above all: Simplicity

Guidelines

- A guideline that appeals to me:
  - Bottom-up, application-driven development of abstractions
- Aim at a good division of labor between the programmer and System
  - Automate what the system can do well
  - Allow programmer to do what they can do best

Foundation: Adaptive Runtime System

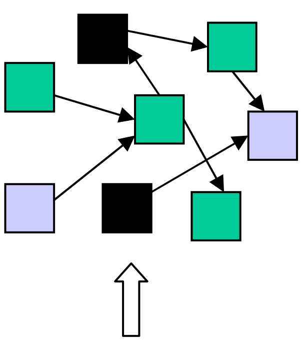
For me, Based on Migratable Objects

Programmer: **[Over] decomposition into virtual processors**

Runtime: **Assigns VPs to processors**

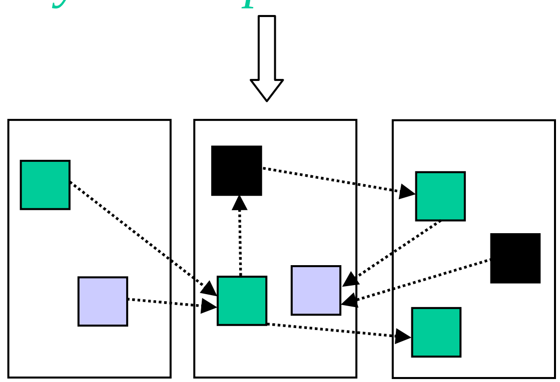
Enables *adaptive runtime strategies*

Implementations: Charm++, AMPI



User View

System implementation



Benefits

- Software engineering
  - Num. of VPs to match application logic (not physical cores)
  - Separate VPs for different modules
- Message driven execution
  - Predictability :
  - Asynchronous reductions
- Dynamic mapping
  - Heterogeneity
    - Vacate, adjust to speed, share
  - Change set of processors used
  - Dynamic load balancing

What is the cost of
Processor Virtualization?

"Overhead" of Virtualization

$$T_1 = T(1 + \frac{v}{g})$$

$$T_p = \max(g, \frac{T_1}{P})$$

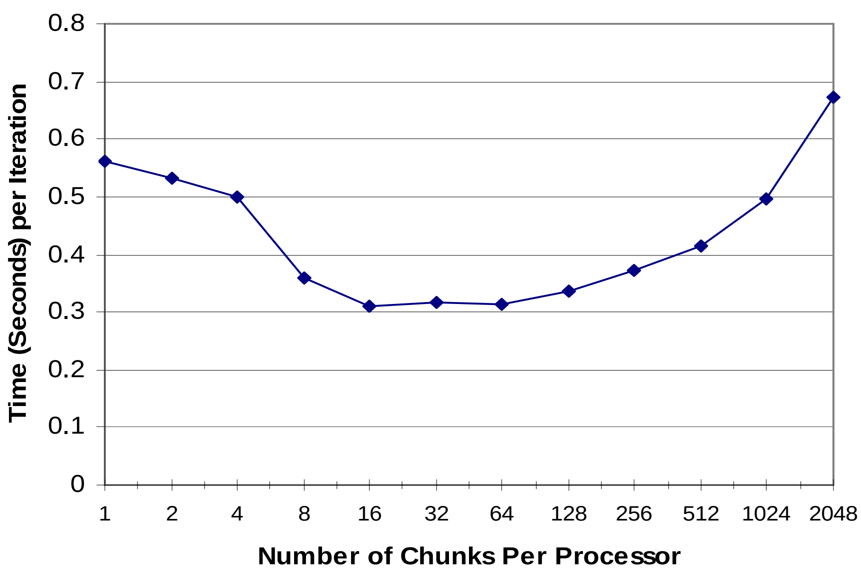
$$T_p = \max(g, \frac{T * (1 + \frac{v}{g})}{P})$$

V: overhead per message

Tp: p processor completion time

G: grainsize (computation per message)

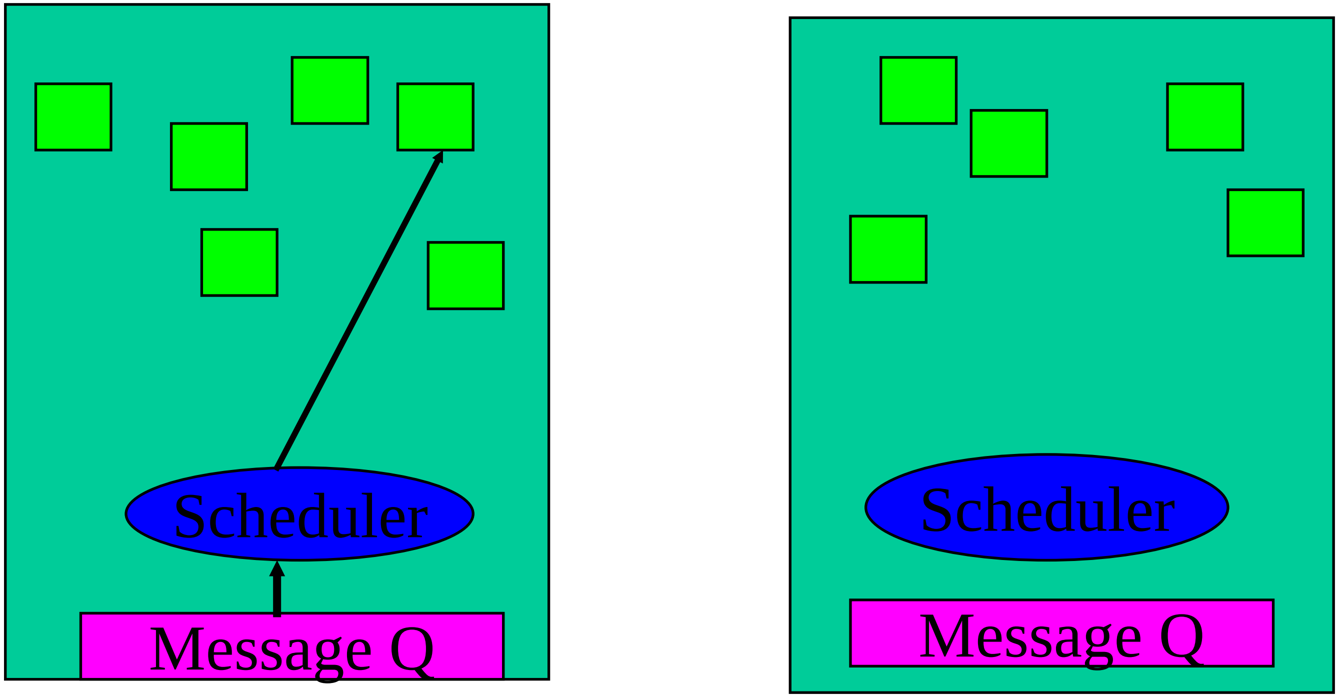
- Fragmentation cost?
  - Cache performance improves
  - Adaptive overlap improves
  - Difficult to see cost..
- Fixable Problems:
  - Memory overhead: (larger ghost areas)
  - Fine-grained messaging:



Modularity
and
Concurrent Composibility

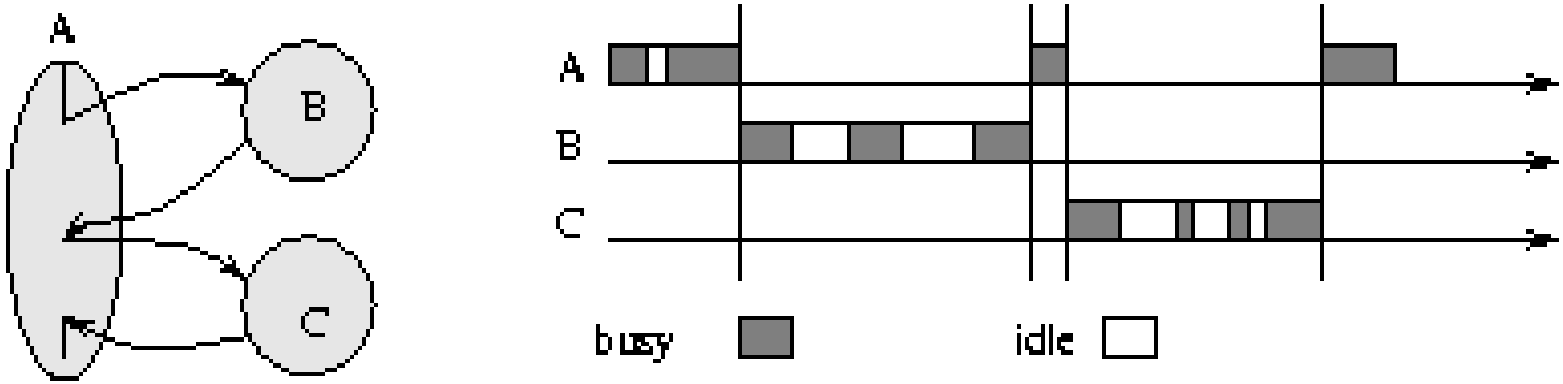
Message Driven Execution

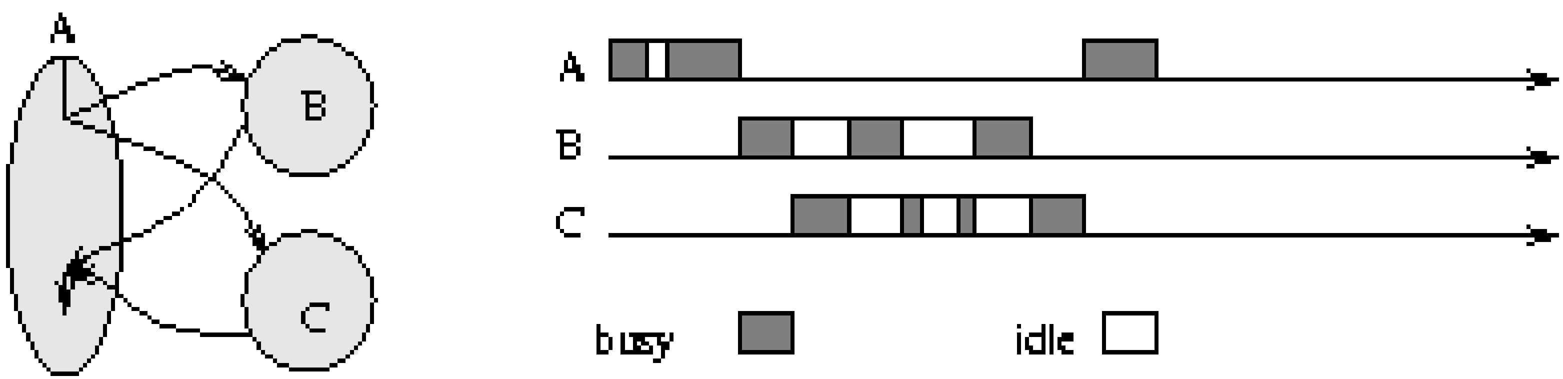
Virtualization leads to *Message Driven Execution*



Which leads to Automatic Adaptive overlap of computation and communication

Adaptive overlap and modules

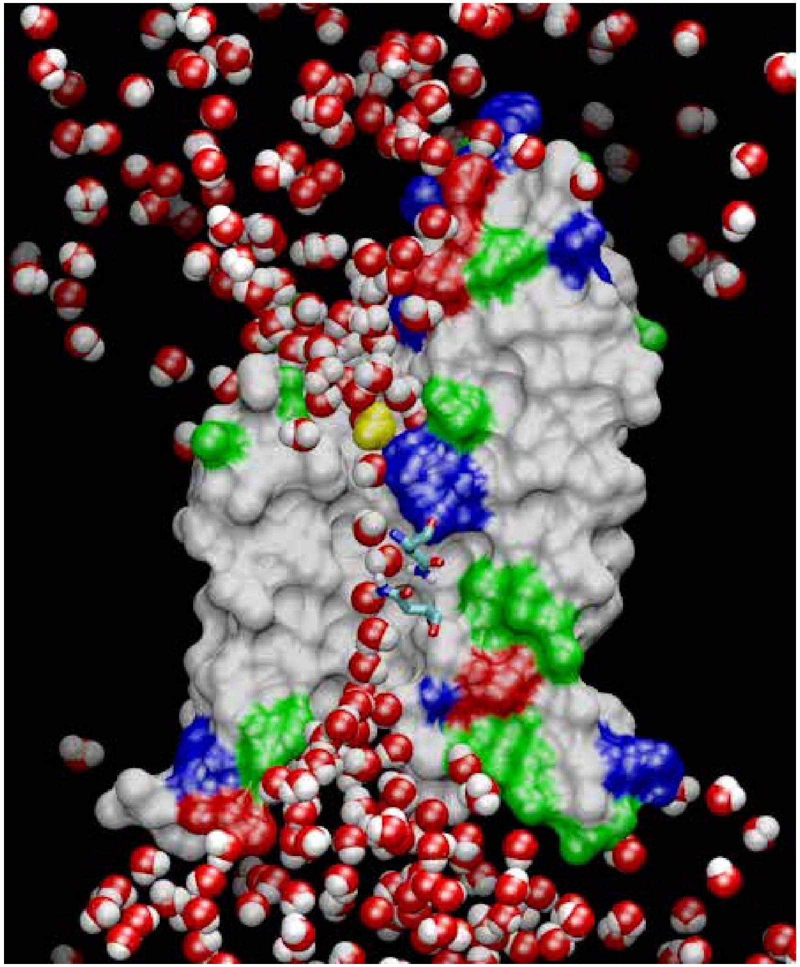




SPMD and Message-Driven Modules
(From A. Gursoy, *Simplified expression of message-driven programs and quantification of their impact on performance*, Ph.D Thesis, Apr 1994.)

Modularity, Reuse, and Efficiency with Message-Driven Libraries: Proc. of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, San Fransisco, 1995

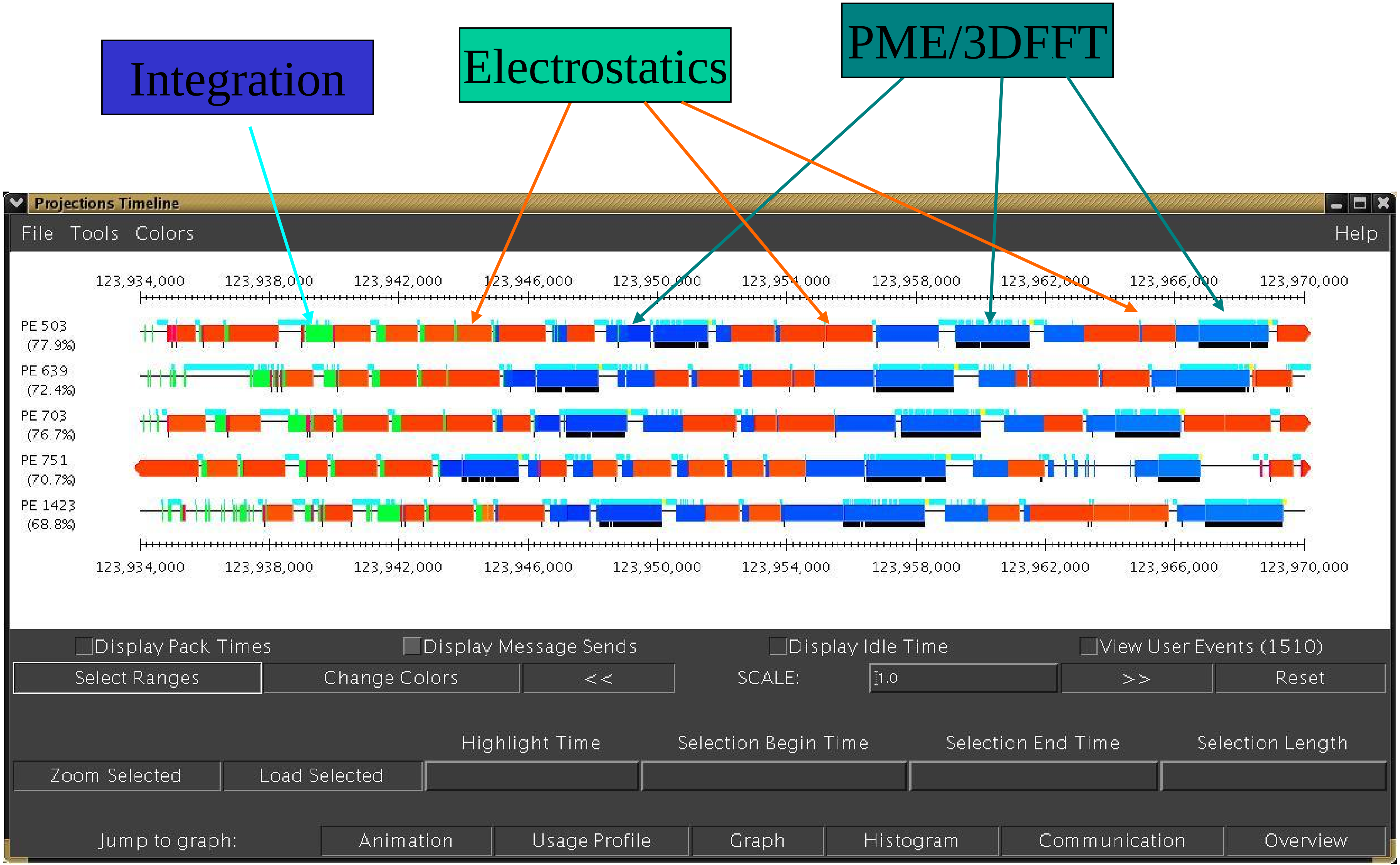
NAMD: A Production MD program



NAMD

- Fully featured program
- NIH-funded development
- Installed at NSF centers
- Large published simulations
- *We were able to demonstrate the utility of adaptive overlap, and share the Gordon Bell award in 2002*

Collaboration with K. Schulten, R. Skeel, and coworkers


Integration
Electrostatics
PME/3DFFT
Projections Timeline
File Tools Colors
Help
123,934,000 123,938,000 123,942,000 123,946,000 123,950,000 123,954,000 123,958,000 123,962,000 123,966,000 123,970,000
PE 503 (77.9%)
PE 639 (72.4%)
PE 703 (76.7%)
PE 751 (70.7%)
PE 1423 (68.8%)
Display Pack Times
Display Message Sends
Display Idle Time
View User Events (1510)
Select Ranges
Change Colors
<<
SCALE:
1.0
>>
Reset
Highlight Time
Selection Begin Time
Selection End Time
Selection Length
Zoom Selected
Load Selected
Jump to graph:
Animation
Usage Profile
Graph
Histogram
Communication
Overview

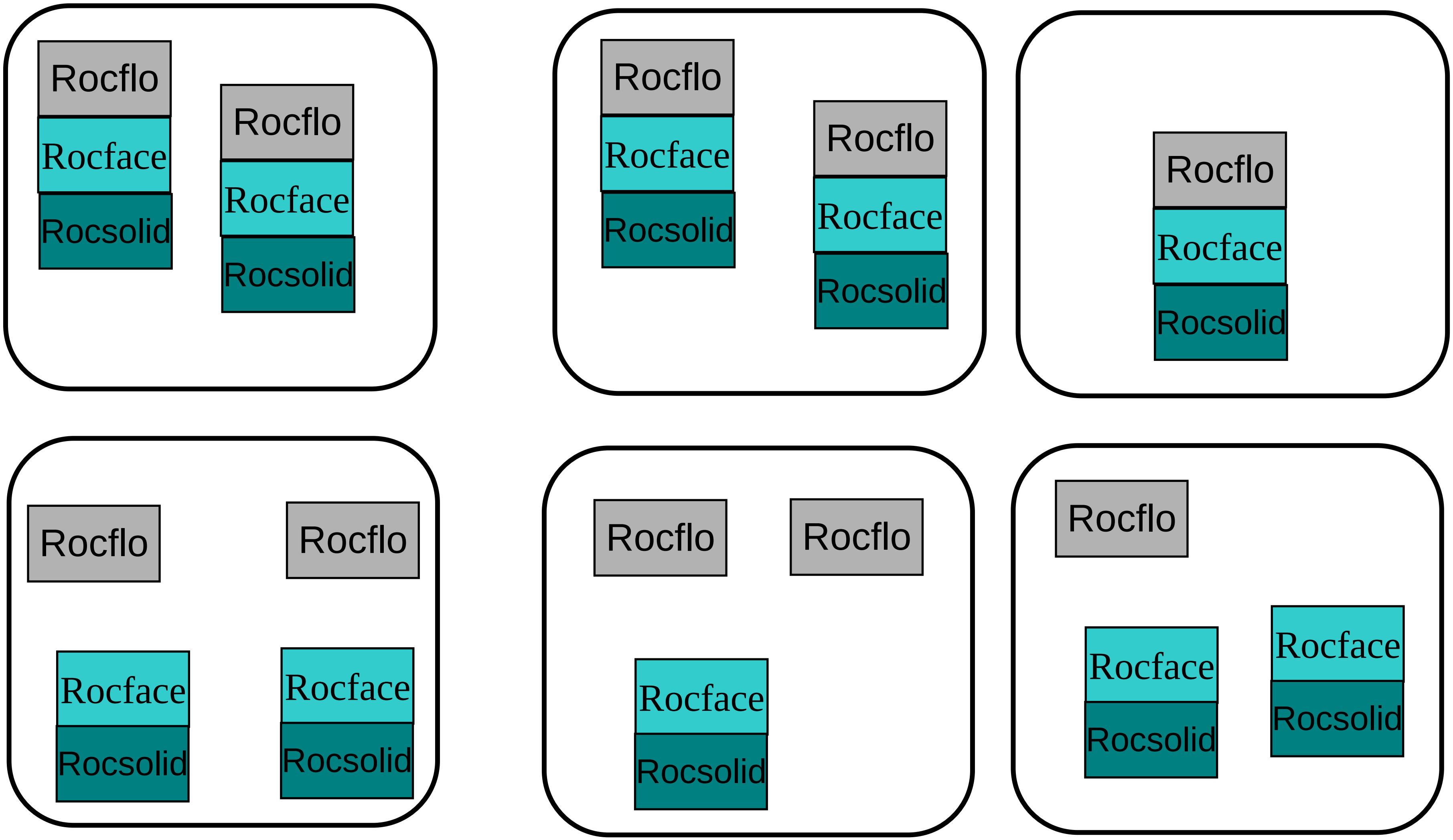
Modularization

- Logical Units decoupled from “Number of processors”
  - E.G. Oct tree nodes for particle data
  - No artificial restriction on the number of processors
    - Cube of power of 2
- Modularity:
  - Software engineering: cohesion and coupling
  - MPI’s “are on the same processor” is a bad coupling principle
  - Objects liberate you from that:
    - E.G. Solid and fluid modules in a rocket simulation

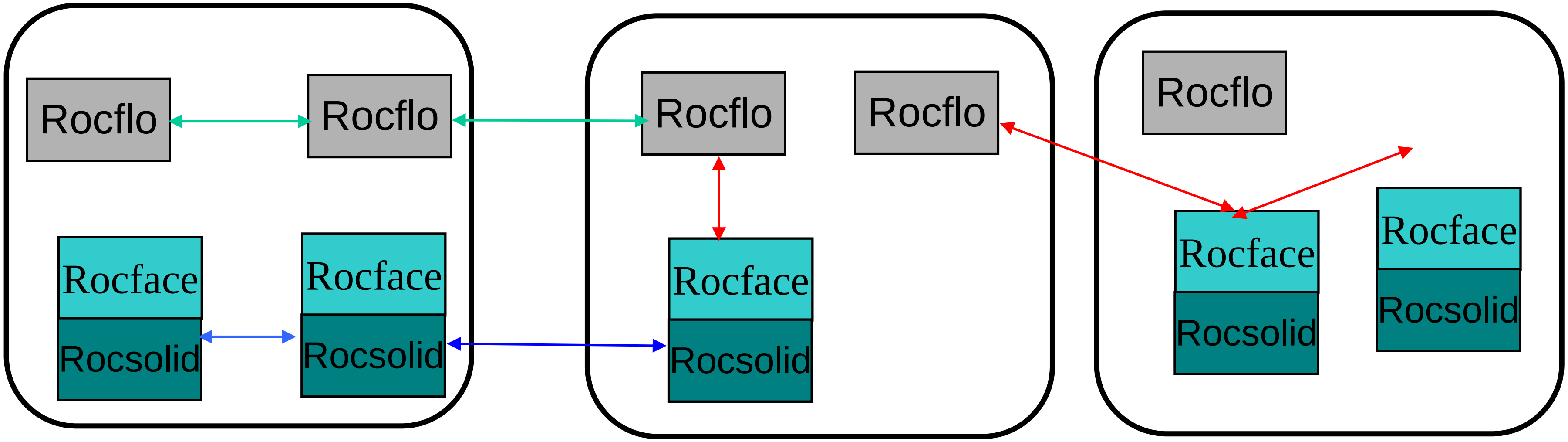
Rocket Simulation

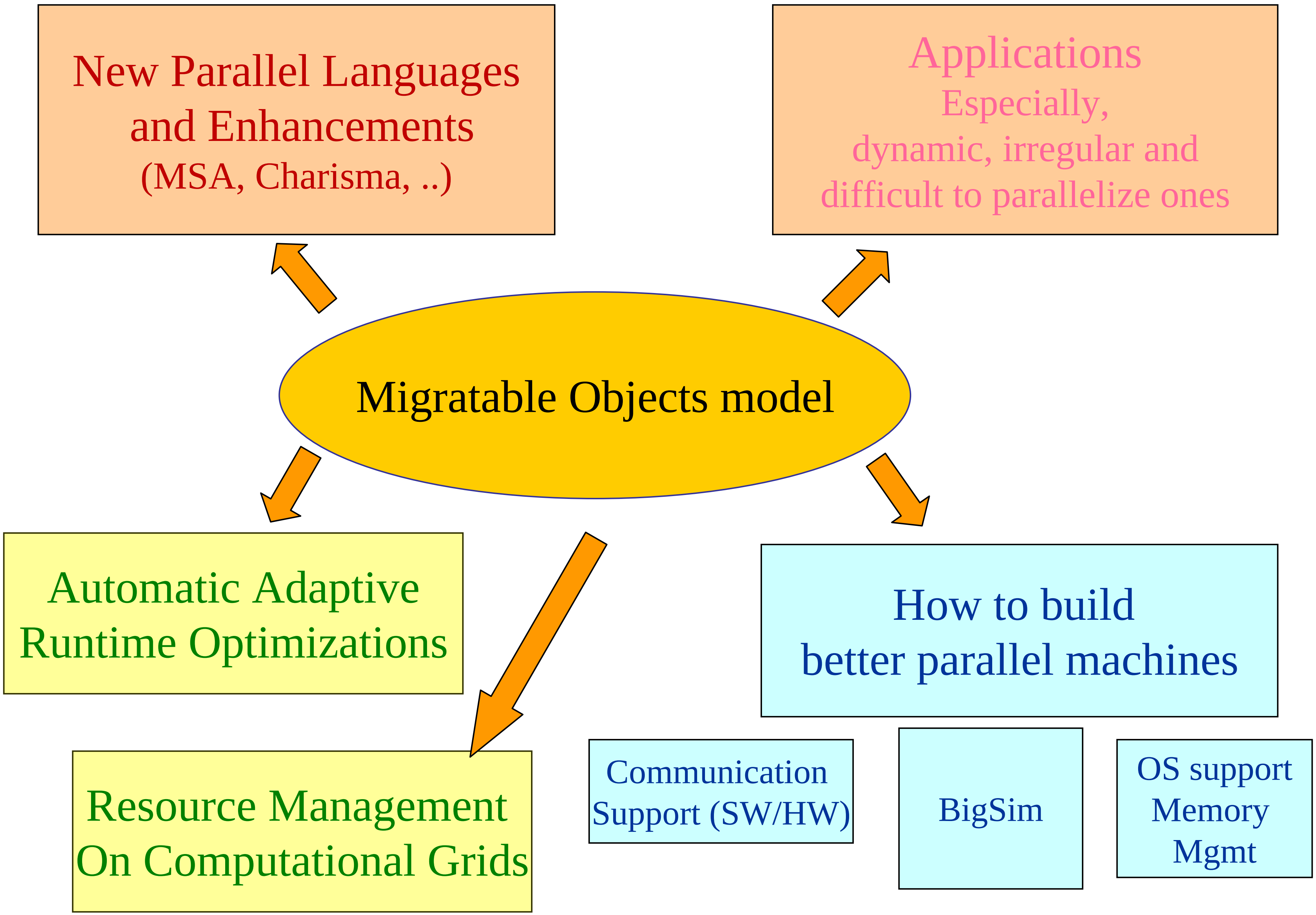
- Large Collaboration headed Mike Heath
  - DOE supported ASCI center
- Challenge:
  - Multi-component code, with modules from independent researchers
  - MPI was common base
- AMPI: new wine in old bottle
  - Easier to convert
  - Can still run original codes on MPI, unchanged

Rocket Simulation Components in AMPI



AMPI and Roc* communications



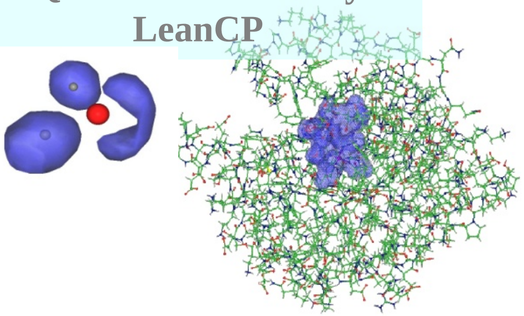

New Parallel Languages and Enhancements
(MSA, Charisma, ..)
Applications
Especially,
dynamic, irregular and
difficult to parallelize ones
Migratable Objects model
Automatic Adaptive Runtime Optimizations
Resource Management On Computational Grids
How to build better parallel machines
Communication Support (SW/HW)
BigSim
OS support Memory Mgmt

Charm++/AMPI are mature systems

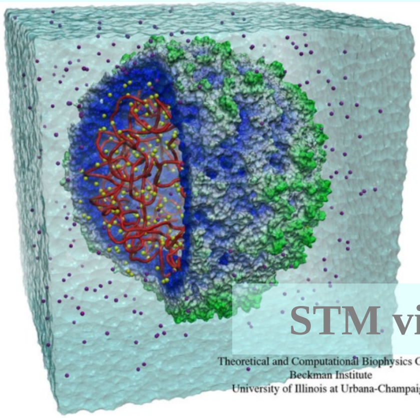
- Available on all parallel machines we know of
  - Clusters, Vendor supported: IBM, SGI, HP (Q), BlueGene/L, …
- Tools:
  - Performance analysis/visualization
  - Debuggers
  - Live visualization
  - Libraries and frameworks
- Used by many applications
  - 17,000+ installations
  - NAMD , Rocket simulation, Quantum Chemistry, Space-time meshes, animation graphics, Astronomy, ..
- It is C++, with message (event) driven execution
  - So, a familiar model for desktop programmers

Develop abstractions in context of full-scale applications

Quantum Chemistry LeanCP

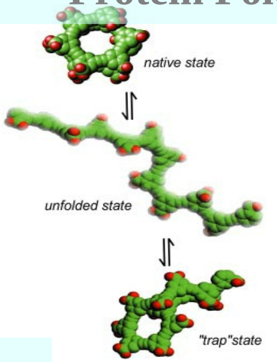


NAMD: Molecular Dynamics

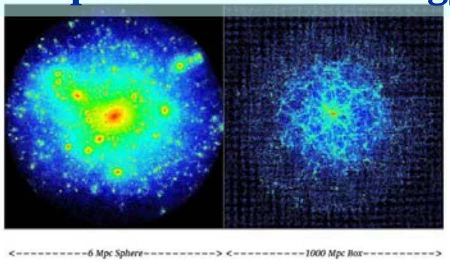


STM virus simulation

Protein Folding



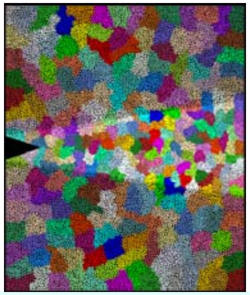
Computational Cosmology



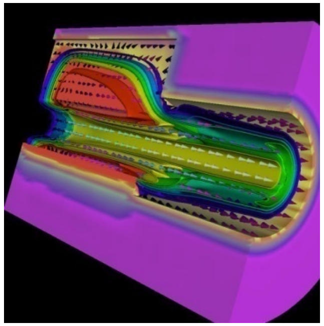
Parallel Objects,

Adaptive Runtime System

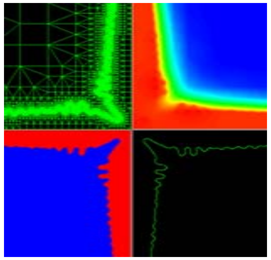
Libraries and Tools



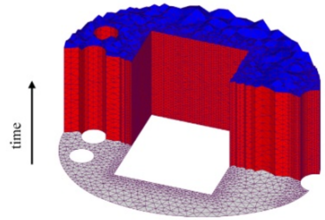
Crack Propagation



Rocket Simulation



Dendritic Growth



Space-time meshes

The enabling CS technology of parallel objects and intelligent Runtime systems has led to several collaborative applications in CSE

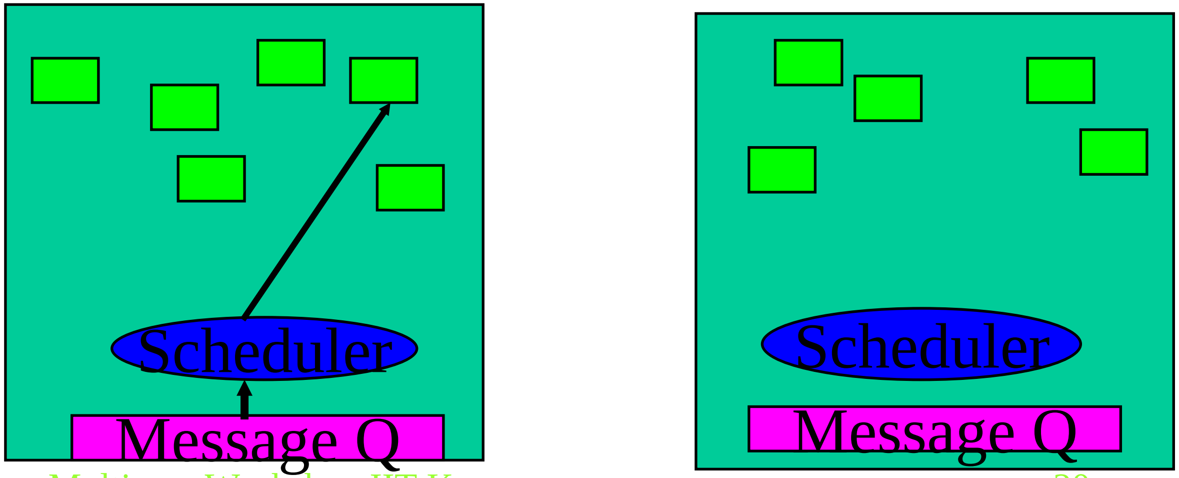
CSE to ManyCore

- The Charm++ model has succeeded in CSE/HPC
- Because:
  - Resource management, …
- In spite of:
  - Based on C++, not Fortran, message-driven model,..
- But is an even better fit for desktop programmers
  - C++, event driven execution
  - Predictability of data/code accesses

15% of cycles at NCSA, 20% at PSC, were used on Charm++ apps, in a one year period

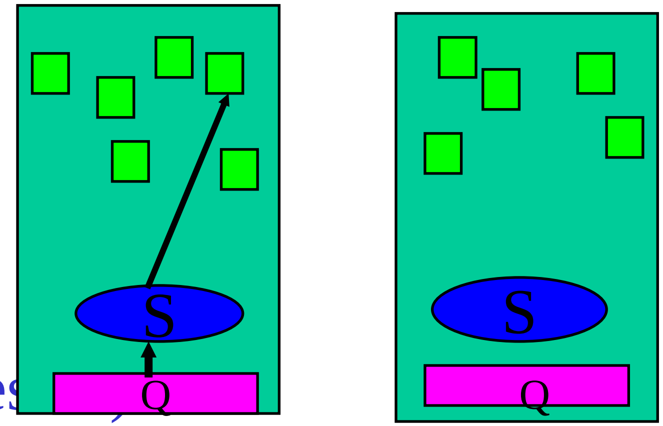
Why is it suitable for Multi-cores

- Objects connote and promote locality
- Message-driven execution
  - A strong principle of prediction for data and code use
  - Much stronger than Principle of locality
    - Can use to scale memory wall:
    - Prefetching of needed data:
      - into scratch pad memories, for example



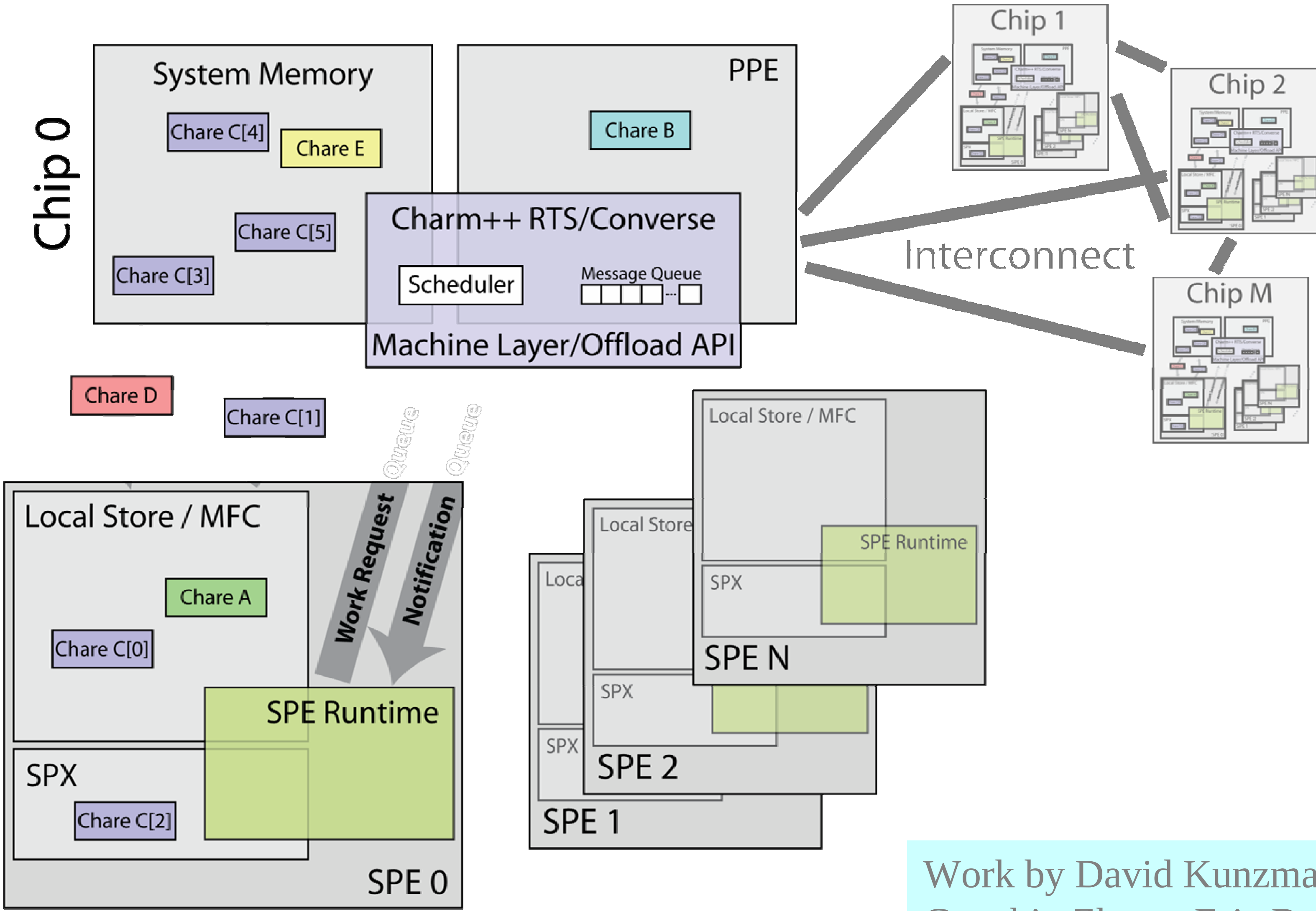
Why Charm++ & Cell?

- Data Encapsulation / Locality
  - Each message associated with…
    - Code : Entry Method
    - Data : Message & Chare Data
  - Entry methods tend to access data local to chare and message



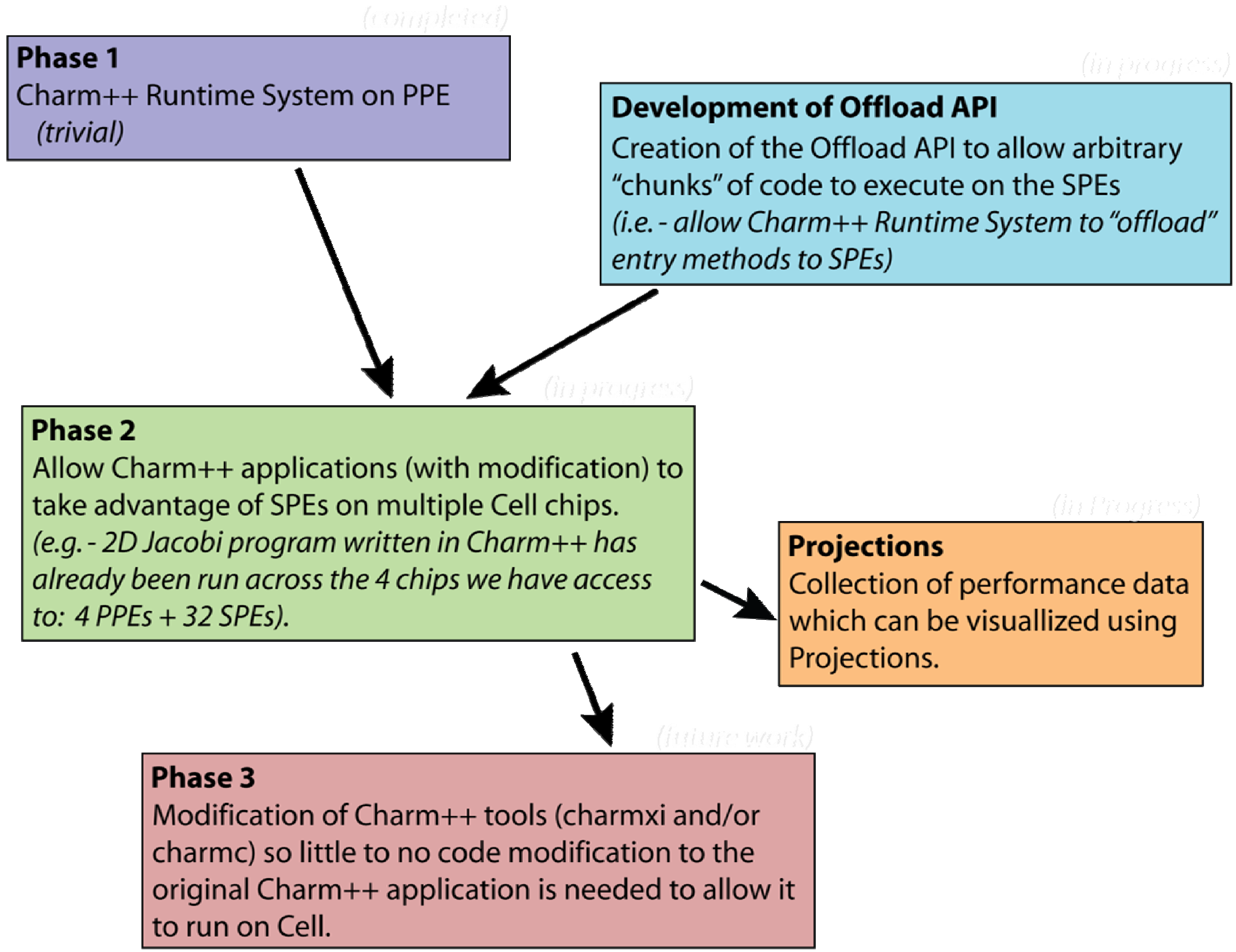
- Virtualization (many chares per processor)
  - Provides opportunity to overlap SPE computation with DMA transactions
  - Helps ensure there is always useful work to do
- Message Queue Peek-Ahead / Predictability
  - Peek-ahead in message queue to determine future work
  - Fetch code and data before execution of entry method

System View on Cell



Work by David Kunzman (with Gengbin Zheng, Eric Bohm,)

Charm++ on Cell Roadmap



So, I expect Charm++ to be a strong contender for manycore models

BUT:
What about the quest for Simplicity?

Charm++ is powerful, but not much simpler than, say, MPI

How to Get to Simple Parallel Programming Models?

- Parallel Programming is much too complex
  - In part because of resource management issues :
    - Handled by Adaptive Runtime Systems
  - In a larger part, because of unintended non-determinacy
    - Race conditions
- Clearly, we need ***simple*** models
  - But what are willing to give up? (No free lunch)
  - Give up “Completeness”!?!
  - May be one can design a language that is simple to use, but not expressive enough to capture all needs

Simplicity?

- A collection of “incomplete” languages, backed by a (few) complete ones, will do the trick
  - As long as they are interoperable
- Where does simplicity come from?
  - Outlaw non-determinacy!
  - Deterministic, Simple, parallel programming models
    - With Marc Snir, Vikram Adve, ..
  - Are there examples of such paradigms?
    - Multiphase shared Arrays : [LCPC ‘04]
    - Charisma++ : [LCR ’04]

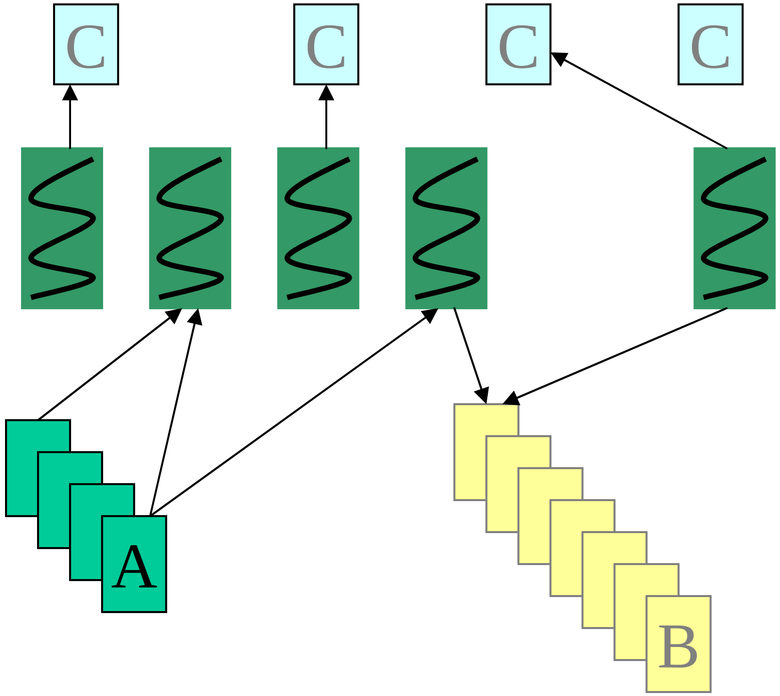
Shared memory or not

- Smart people on both sides:
  - Thesis, antithesis
- Clearly, needs a "synthesis"
- "Shared memory is easy to program" has
  - Only a grain of truth
  - But there exists that grain of truth
- We as a community, need to have this debate
  - Put some armor on, drink friendship potion, but debate the issue threadbare..
  - What do we mean by SAS model and what we like and dislike about it

Multiphase Shared Arrays

- Observations:
  - General shared address space abstraction is complex
  - Certain special cases are simple, and cover most uses
- Each array is in one mode at a time
  - But its mode may change from phase to phase
- Modes
  - Write-once
  - Read-only
  - Accumulate
  - Owner-computes
- All workers `sync`, at end of each phase

MSA:



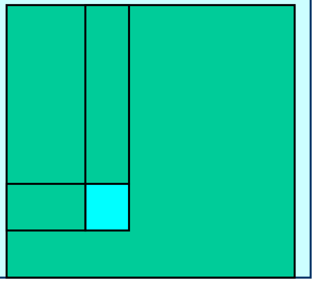
- In the simple model:
- A program consists of
  - A collection of Charm threads, and
  - Multiple collections of data-arrays
    - Partitioned into pages (user-specified)
- Execution begins in a "main"
  - Then all threads are fired in parallel
- More complex model
  - Multiple collections of threads
  - …

MSA: Plimpton MD

```
for  timestep = 0 to Tmax {
  // Phase I : Force Computation: for a section of the interaction matrix
  for i = i_start to i_end
    for j = j_start to j_end
      if (nbrlist[i][j]) { // nbrlist enters ReadOnly mode
        force = calculateForce(coords[i], atominfo[i], coords[j], atominfo[j]);
        forces[i] += force; // Accumulate mode
        forces[j] += -force;
      }
  nbrlist.sync(); forces.sync();  coords.sync(); atominfo.sync();

  for k = myAtomsbegin to myAtomsEnd // Phase II : Integration
     coords[k] = integrate(atominfo[k], forces[k]); // WriteOnly mode
  coords.sync(); atominfo.sync(); forces.sync();

  if  (timestep %8 == 0) { // Phase III: update neighbor list every 8 steps
     for i = i_start to i_end
        for j = j_start to j_end
          nbrList[i][j] = distance( coords[i],coords[j]) < CUTOFF;
    nbrList.sync(); coords.sync();
  }
}
```



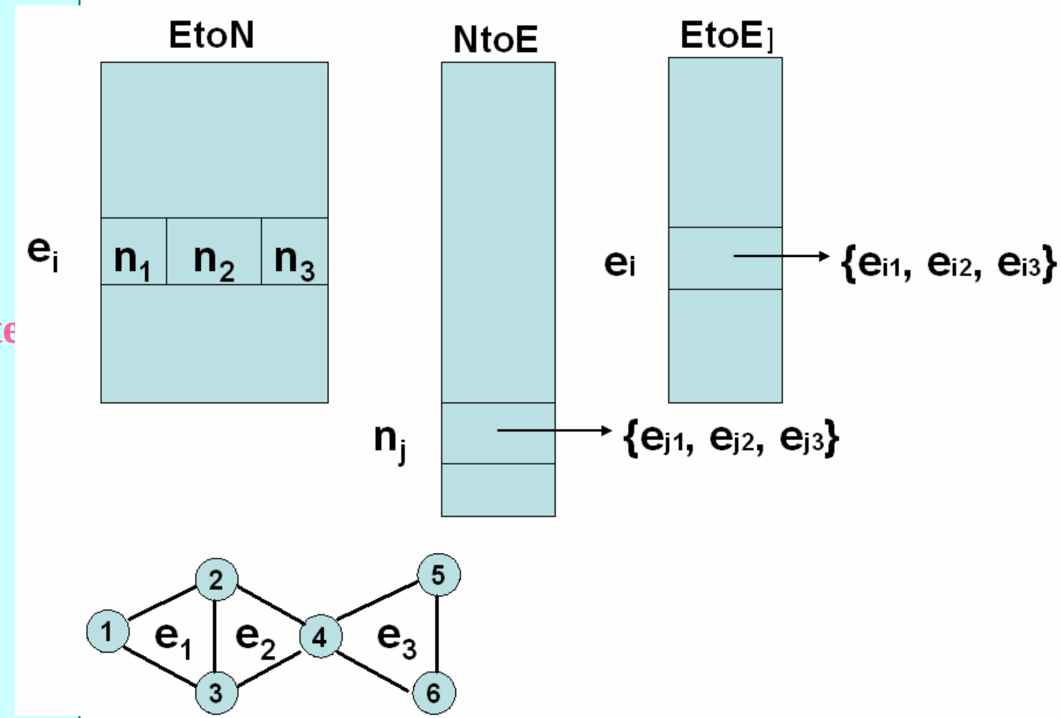
Extensions

- Need check for each access: is the page here?
  - Pre-fetching, and known-local accesses
- A Twist on ACCUMULATE
  - Each array element can be a set
  - Set Union operation is a valid accumulate operation.
  - Example:
    - Appending a list of (x,y) points

MSA: Graph Partition

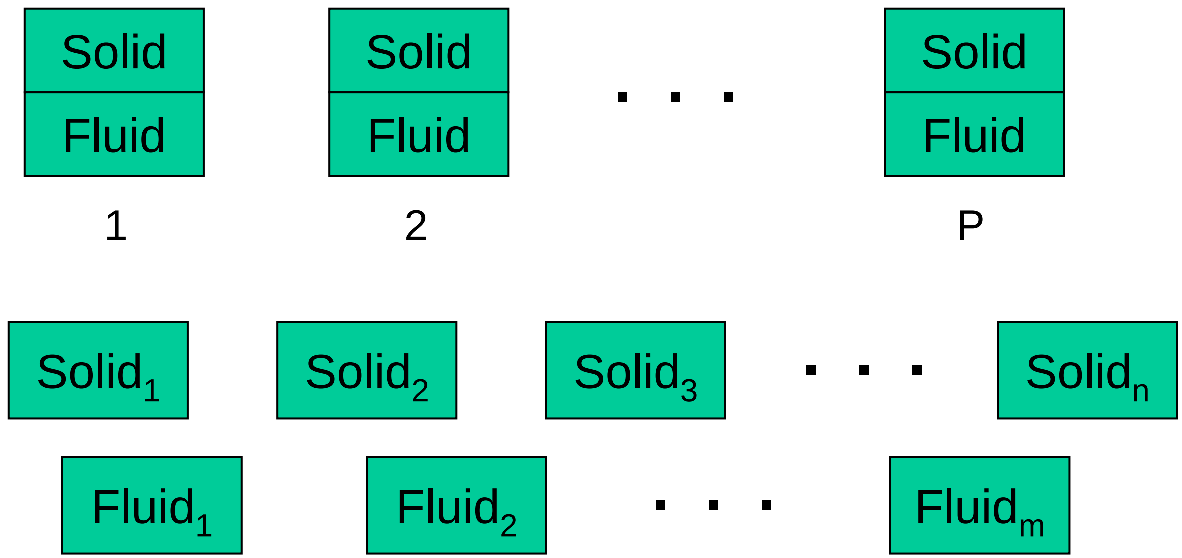
```
// Phase I: EtoN: RO, NtoE: Accumulate
for i=1 to EtoN.length()
  for j=1 to EtoN[i].length() {
    n = EtoN[i][j];
    NtoE[n] += i; // Accumulate
  }
EtoN.sync(); NtoE.sync();

// Phase II: NtoE: RO, EtoE: Accumulate
for j = my section of j
  //foreach pair e1, e2 elementof NtoE[j]
  for i1 = 1 to NtoE[j].length()
      for i2 = i1 + 1 to NtoE[j].length() {
         e1 = NtoE[j][i1];
         e2 = NtoE[j][i2];
         EtoE[e1] += e2; // Accumulate
         EtoE[e2] += e1;
    }
EtoN.sync(); NtoE.sync();
```



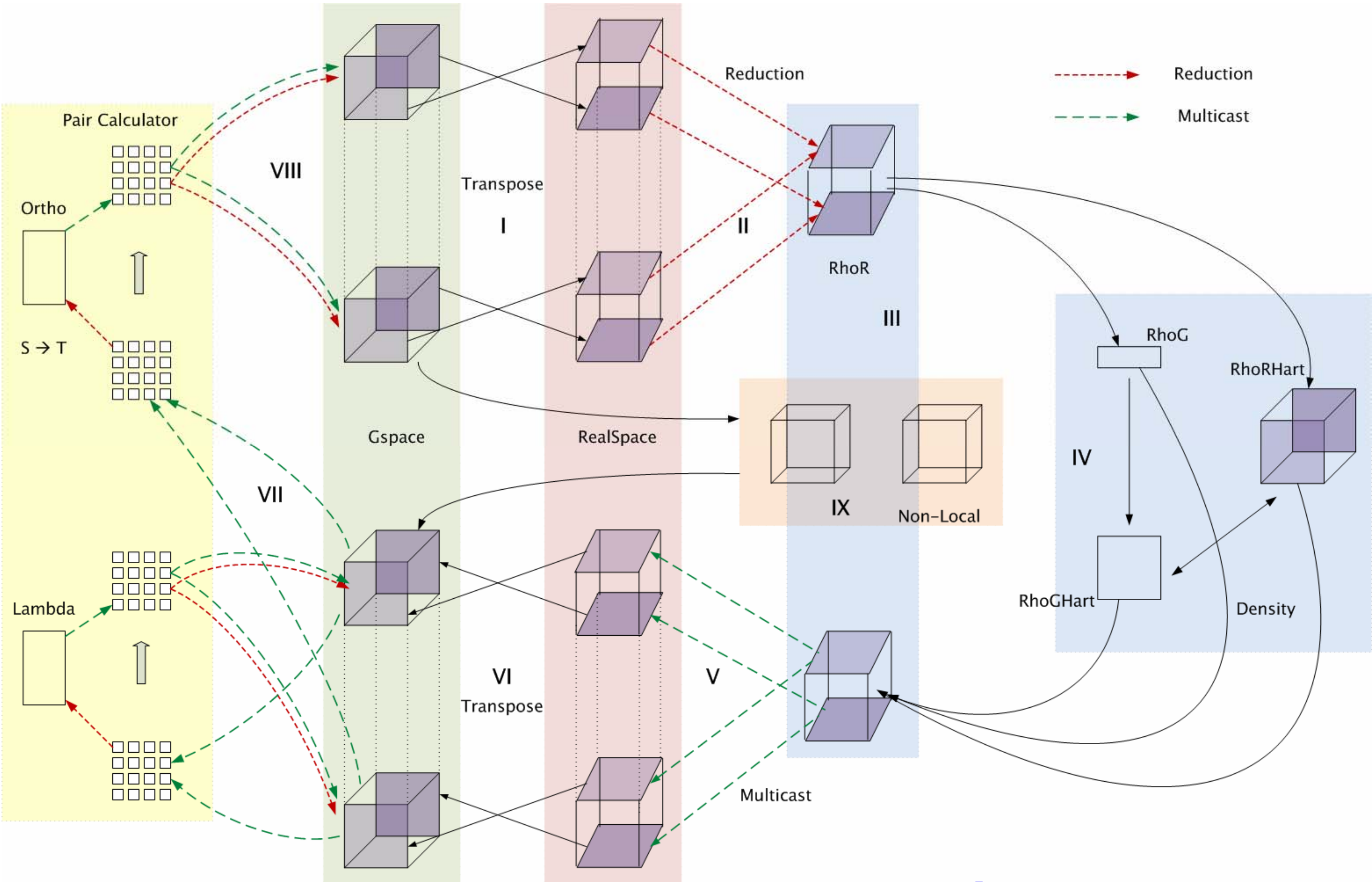
Charisma: Motivation

- Rocket simulation example under traditional MPI vs. Charm++/AMPI framework



  - Benefit: load balance, communication optimizations, modularity
  - Problem: flow of control buried in asynchronous method invocations

Motivation: Car-Parrinello Ab Initio Molecular Dynamics (CPMD)



Charisma++ example (Simple)

```
Jacobi 1D
   begin
     forall i in J
       <lb[i],rb[i]> := J[i].init();
     end-forall
     while (e > threshold)
       forall i in J
         <+e, lb[i], rb[i]> :=  J[i].compute(rb[i-1],lb[i+1]);
       end-forall
     end-while
   end
```

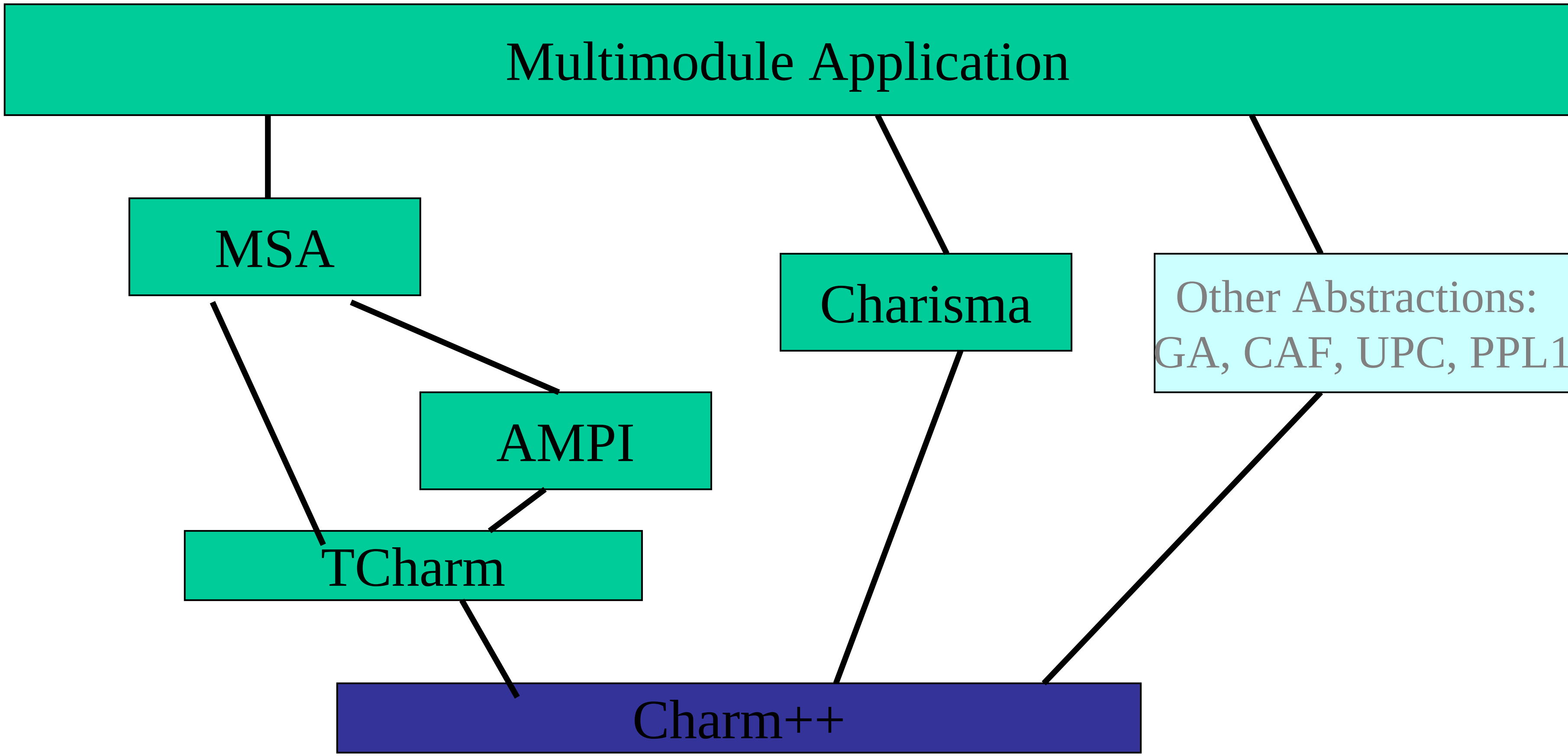
Mol. Dynamics with Spatial Decomposition

```
 foreach i,j,k in cells
              <atoms[i,j,k]>:= cells[i,j,k].produceAtoms();
 end-foreach
for iter := 0 to MAX_ITER
   foreach i1,j1,k1,i2,j2,k2 in cellpairs
       <+forces[i1,j1,k1]> :=
            cellpairs[i1,j1,k1,i2,j2,k2].computeCoulombForces(
                                         atoms[i1,j1,k1],atoms[i2,j2,k2]);
   end-foreach

   foreach … for bonded forces.. Uses atoms and add to forces

   foreach i,j,k in cells
            <atoms[i,j,k]> :=  cells[i,j,k].integrate(forces[i,j,k]);
   end-foreach
end-for
```

A set of "incomplete" but elegant/simple languages, backed by a low-level complete one



Lets play together

- Multiple programming models need to be investigated
  - "Survival of the fittest" doesn't lead to a single species, it leads to an eco-system.
- Different ones may be good for different algorithms/domains/…
- Allow them to interoperate in a multi-paradigm environment

Summary

- It is necessary to raise the level of abstraction
  - Foundation: adaptive runtime system, based on migratable objects
    - Automate resource management
    - Composibility
    - Interoperability
  - Design new Models that avoid data races, and promote locality
  - Incorporate good aspects of shared memory model

More info on my group's work:
http://charm.cs.uiuc.edu