Migratable Objects and Task-Based Parallel Programming with Charm++

Challenges in Parallel Programming

- Applications are getting more sophisticated
  - Adaptive refinement
  - Multi-scale, multi-module, multi-physics
  - E.g. load imbalance emerges as a huge problem for some apps
- Exacerbated by strong scaling needs from apps
  - Strong scaling: run an application with same input data on more processors, and get better speedups
  - Weak scaling: larger datasets on more processors in the same time
- Hardware variability
  - Static/dynamic
  - Heterogeneity: processor types, process variation, etc.
  - Power/Temperature/Energy
  - Component failure

Our View

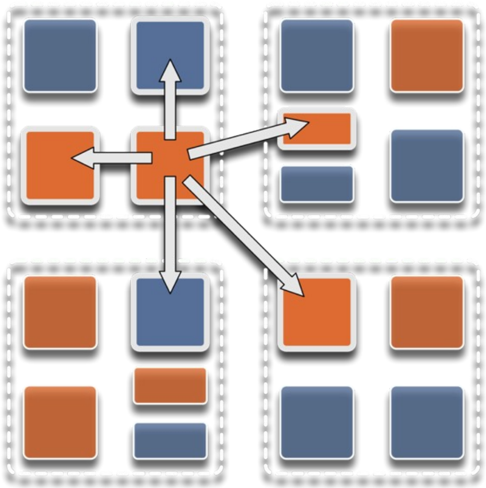
- To deal with these challenges, we must seek:
  - Not full automation
  - Not full burden on app-developers
  - But: a good division of labor between the system and app developers
    - Programmer: what to do in parallel, System: where,when
- Develop language driven by needs of real applications
  - Avoid "platonic" pursuit of "beautiful" ideas
  - Co-developed with NAMD, ChaNGa, OpenAtom,..
- Pragmatic focus
  - Ground-up development, portability,
  - accessibility for a broad user base

What is Charm++?

- Charm++ is a generalized approach to writing parallel programs
  - An alternative to the likes of MPI, UPC, GA etc.
  - But not to sequential languages such as C, C++, and Fortran
- Represents:
  - The style of writing parallel programs
  - The runtime system
  - And the entire ecosystem that surrounds it
- Three design principles:
  - Overdecomposition, Migratability, Asynchrony

Overdecomposition

- Decompose the work units & data units into many more pieces than execution units
  - Cores/Nodes/…
- Not so hard: we do decomposition anyway



Migratability

- Allow these work and data units to be migratable at runtime
  - i.e. the programmer or runtime can move them
- Consequences for the application developer
  - Communication must now be addressed to logical units with global names, not to physical processors
  - But this is a good thing
- Consequences for RTS
  - Must keep track of where each unit is
  - Naming and location management

Asynchrony: Message-Driven Execution

- With over-decomposition and migratability:
  - You have multiple units on each processor
  - They address each other via logical names
- Need for scheduling:
  - What sequence should the work units execute in?
  - One answer: let the programmer sequence them
    - Seen in current codes, e.g. some AMR frameworks
  - Message-driven execution:
    - Let the work-unit that happens to have data ("message") available for it execute next
    - Let the RTS select among ready work units
    - Programmer should not specify what executes next, but can influence it via priorities

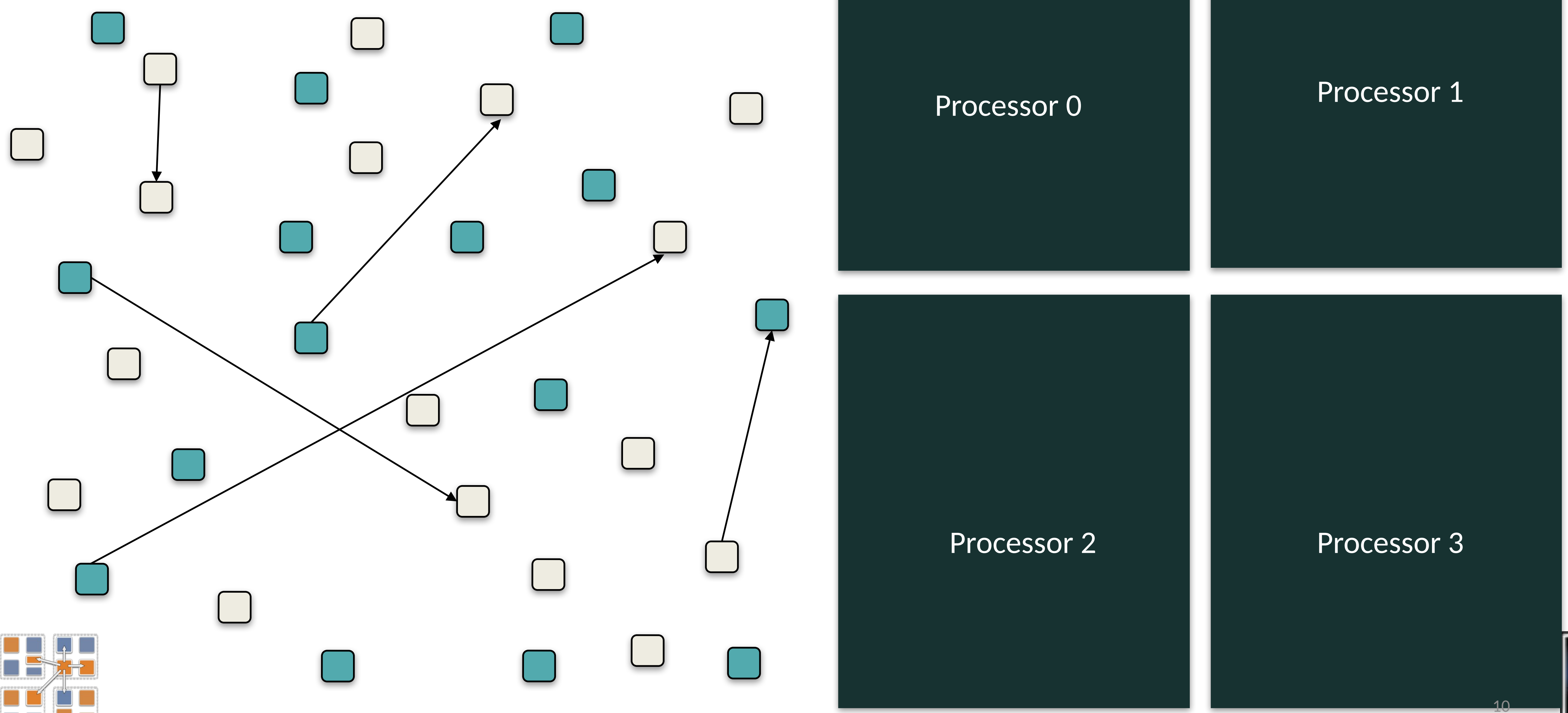
Key Ideas in our parallel programming model

- Let the programmer decide what to do in parallel
  - Express decomposition, interactions
- Let the system decide where and when
- How: virtualize the notion of a processor
  - So as to automate resource management and associated functionalities
- The migratable objects programming model
  - Charm++ is one of the (first/foundational) programming system within this model

Realization of This Model in Charm++

- Overdecomposed entities: chares
  - Chares are C++ objects
  - With methods designated as “entry” methods
    - Which can be invoked asynchronously by remote chares
  - Chares are organized into indexed collections
    - Each collection may have its own indexing scheme
      - 1D, ..., 6D
      - Sparse
      - Bitvector or string as an index
  - Chares communicate via asynchronous method invocations
    - `A[i].foo(…);`
      - A is the name of a collection, `i` is the index of the particular chare.

- A Charm++ computation consists of multiple collections of globally visible objects
- Each collection is individually indexed

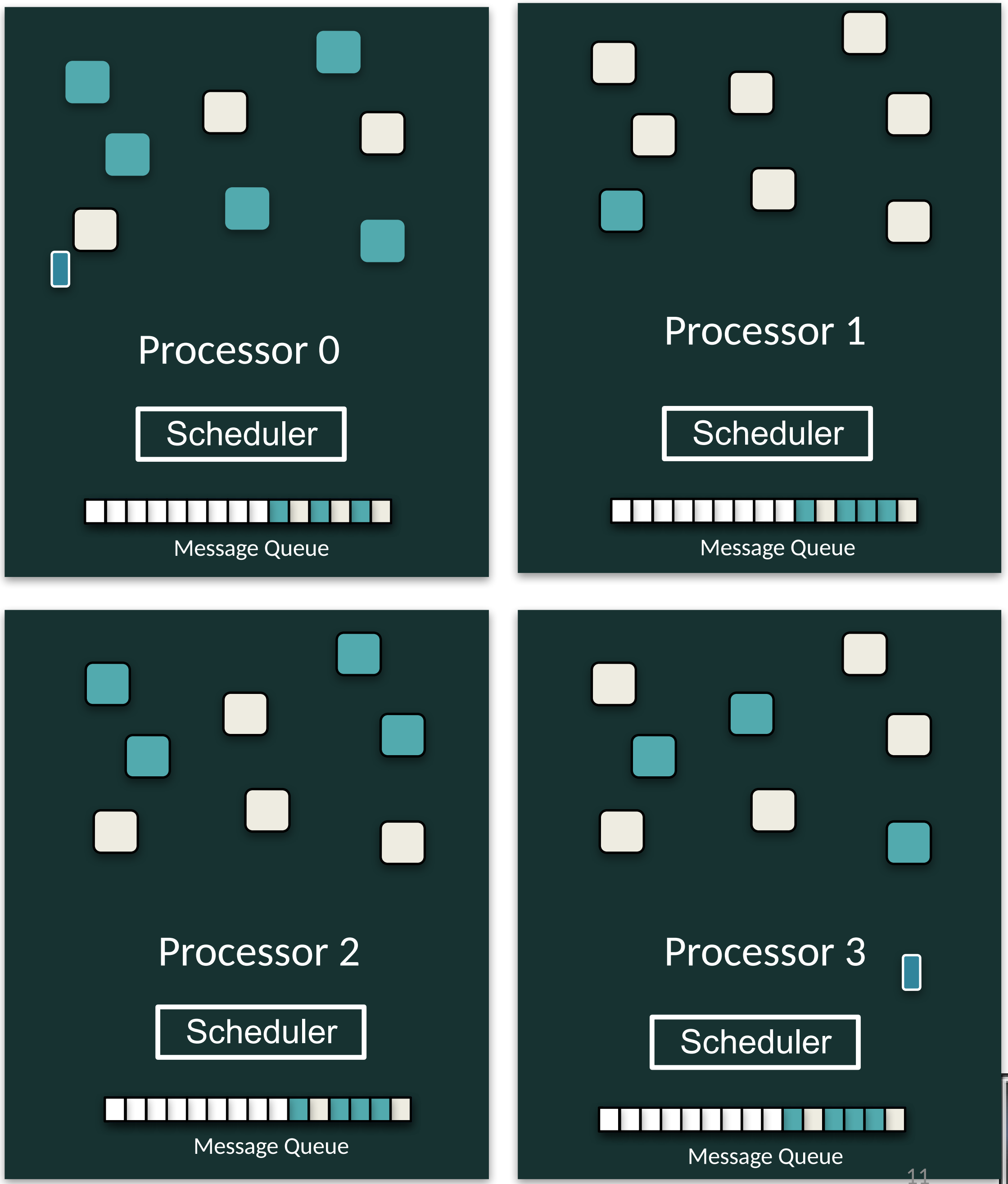


- A Charm++ computation consists of multiple collections of globally visible objects
- Each collection is individually indexed

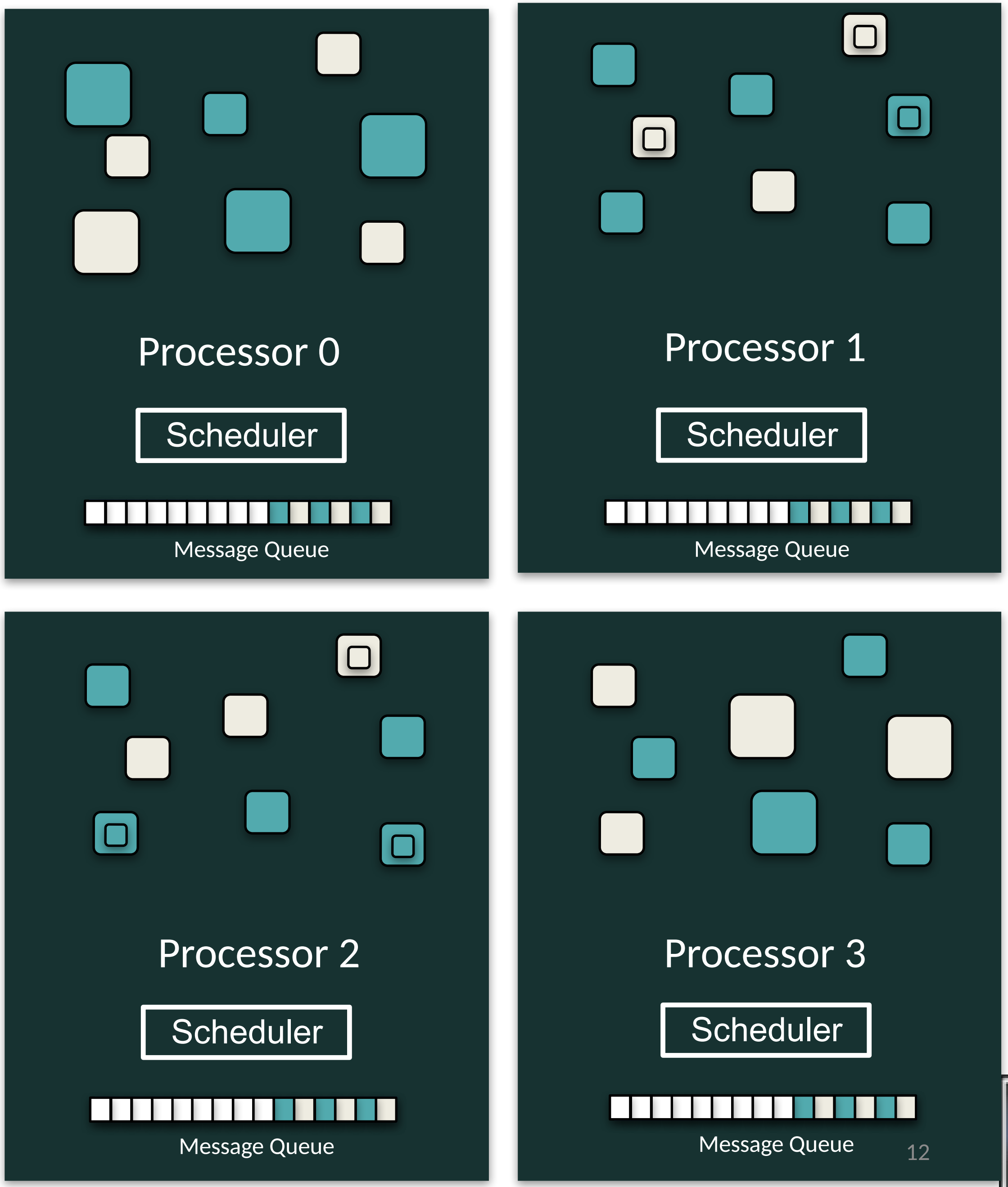
- Objects are assigned to processors by the runtime system
    - Programmer does not need to know where an object is located

- Scheduling on each processors is under the control of a user-space message-driven scheduler

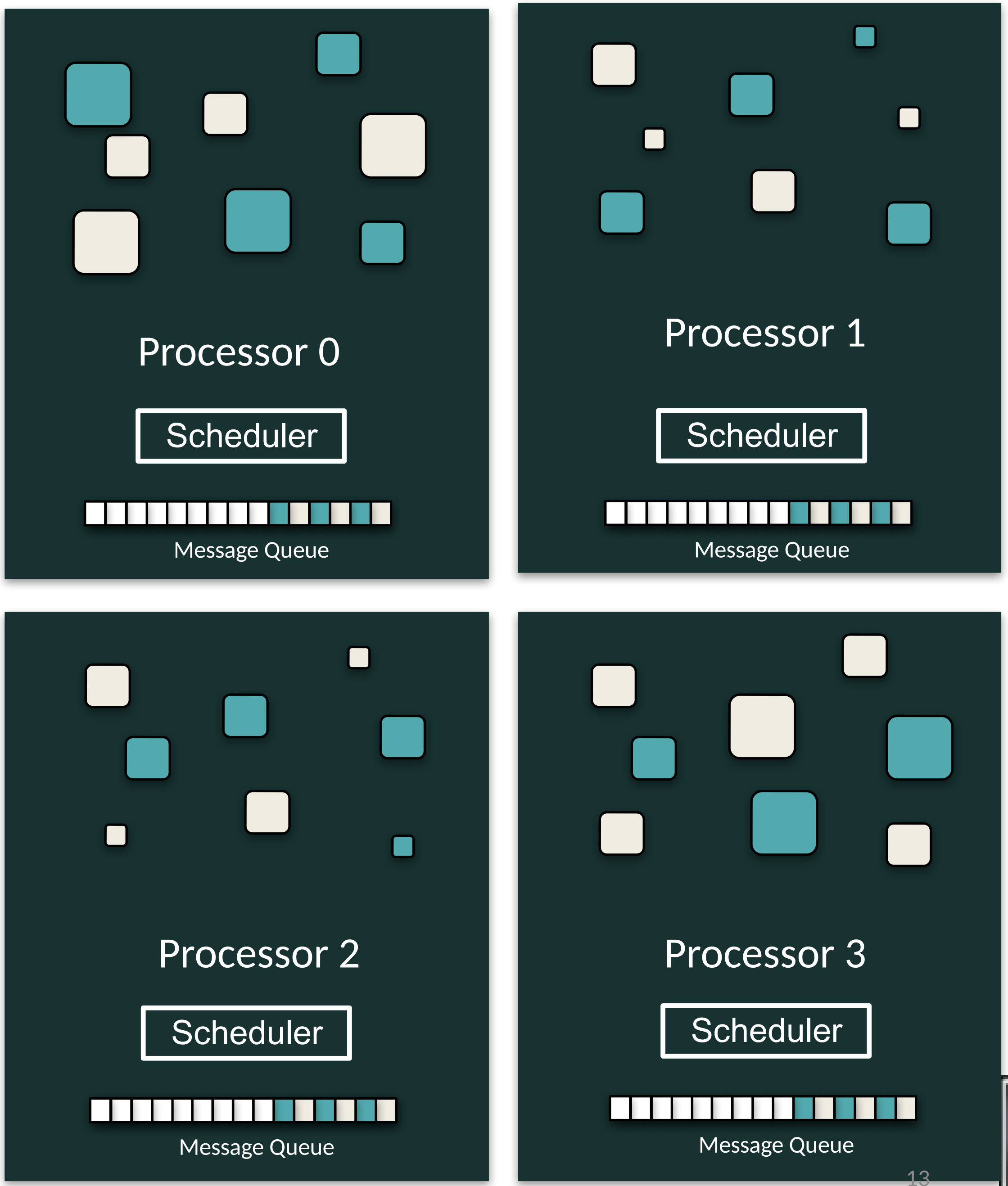
- Example: an object on 0 wants to invoke a method on object A[23]
    - The Runtime System packages the method invocation into a message
    - Locates where the target object is
    - Sends the message to the queue on destination processor
    - Scheduler invokes the method on the target object



The runtime system knows which processors are overloaded, which objects are computationally heavy, which objects talk to which



Using this information, it migrates objects to rebalance load and optimize communication



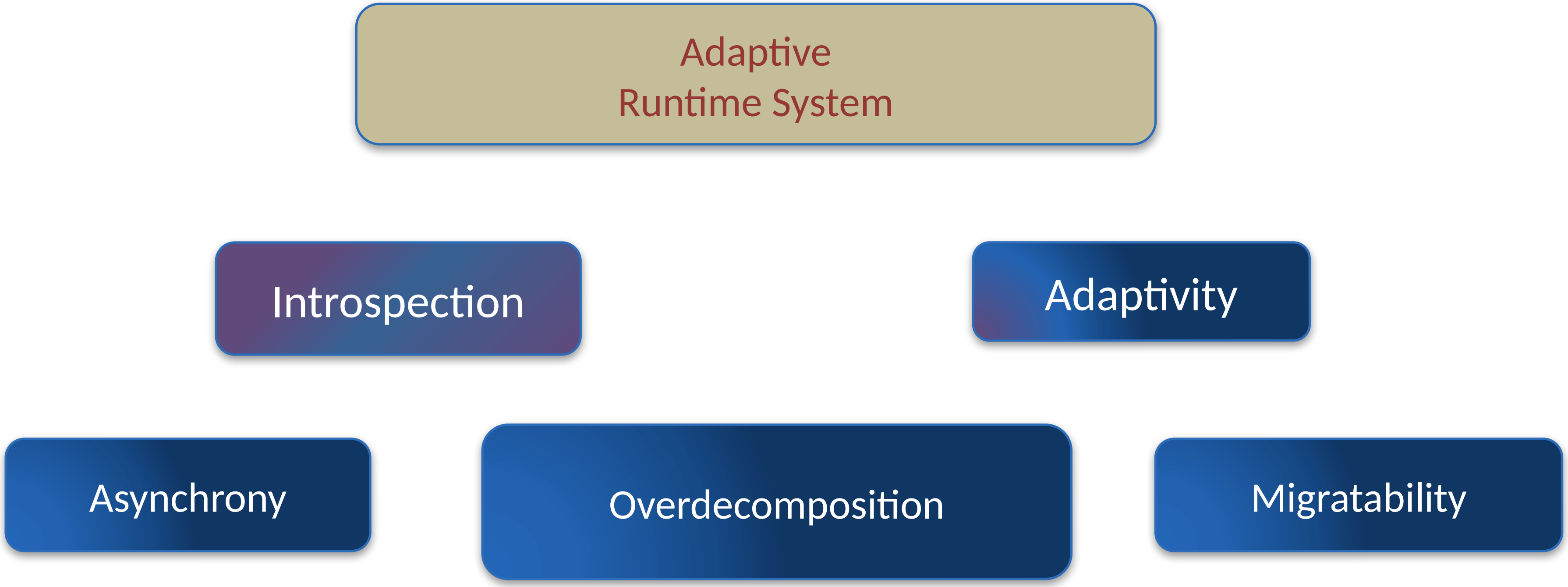
Capabilities

- Capabilities
  - Dynamic load balancing
  - Fault Tolerance
  - Elasticity:
    - change the set of nodes allocated to a job
  - Adaptive overlap of communication and computation*
  - Communication optimizations
  - Out-of-core execution
  - Energy optimizations*
  - Asynchronous GPGPU interface

Demo on raspberry pi cluster:
https://www.hpccharm.com/demo

- Programming Systems
  - Charm++
  - Adaptive MPI
  - Charm4Py
  - Charades
  - Experimental DSLs:
    - MSA, Charisma, ParFUM, …
- Ongoing work on DSLs
  - Ergoline, EIR: DSL framework with compiler support
    - Justin Szaday
  - Python-based DSLs: libraries
  - Enthusiastic students, unfunded projects

Empowering the RTS



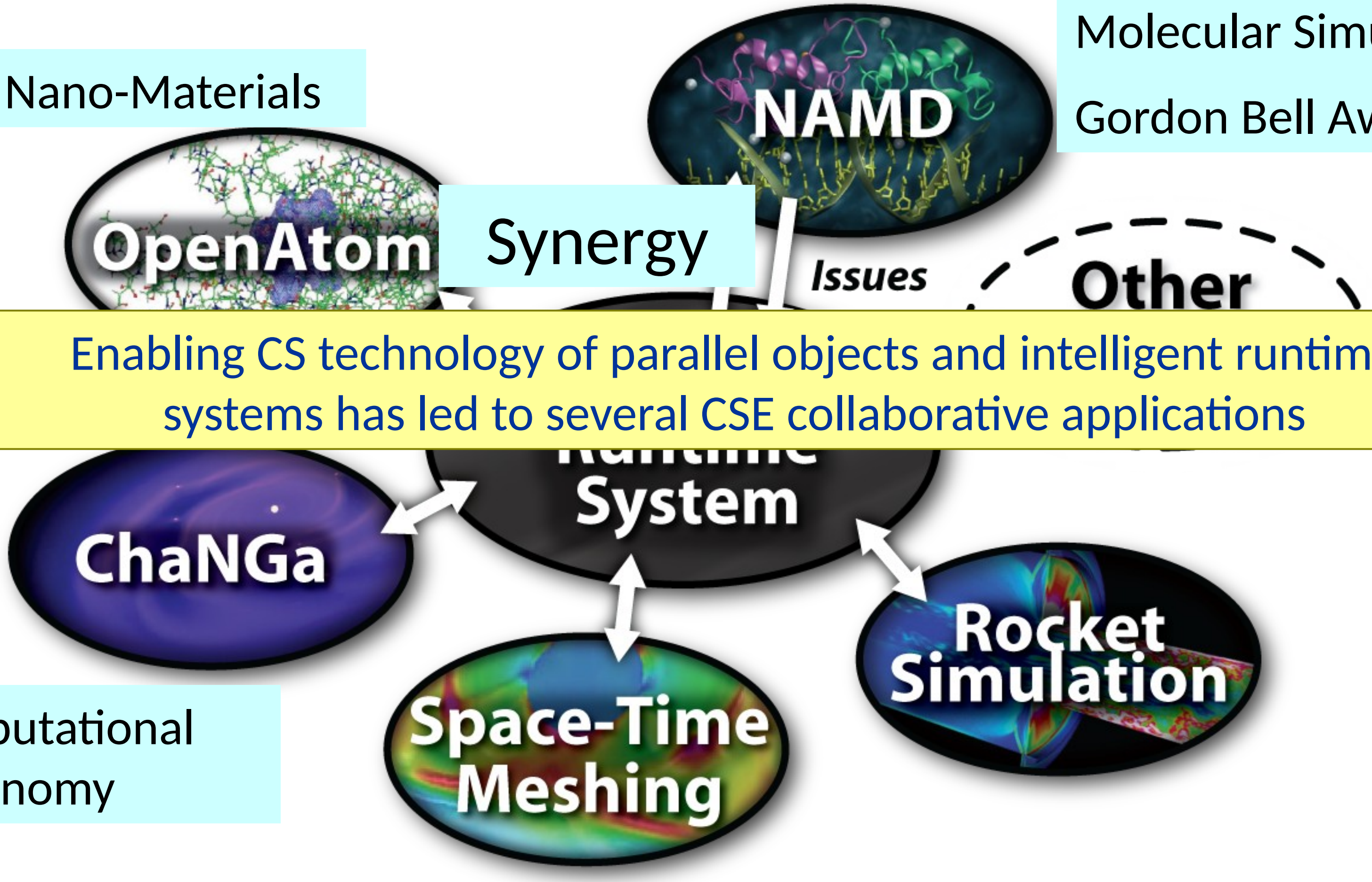
- The Adaptive RTS can:
  - Dynamically balance loads
  - Optimize communication:
    - Spread over time, async collectives
  - Automatic latency tolerance
  - Prefetch data with almost perfect predictability

Charm++ and CSE Applications

Well-known Biophysics
Molecular Simulation App

Gordon Bell Award, 2002

Nano-Materials



Synergy

Enabling CS technology of parallel objects and intelligent runtime systems has led to several CSE collaborative applications

Computational
Astronomy

Summary: What is Charm++?

- Charm++ is a way of parallel programming
- It is based on:
  - Objects
  - Overdecomposition
  - Asynchrony
    - Asynchronous method invocations
  - Migratability
  - Adaptive runtime system
- It has been co-developed synergistically with multiple CSE applications