BLUE WATERS-GEO

MAPPING AND MODELING THE WORLD

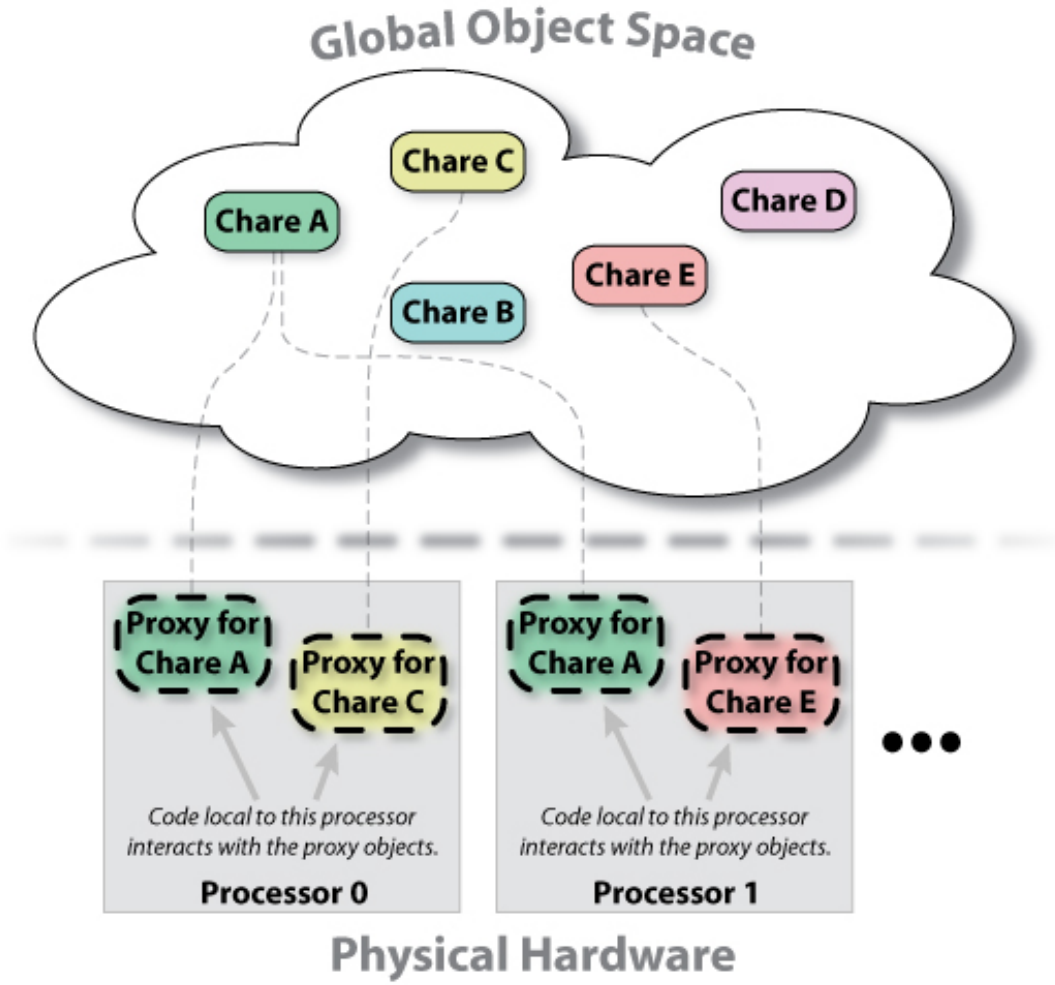
The Charm++ Parallel Programming Framework

James C. Phillips, Ph.D.
Senior Research Programmer, Blue Waters Project Office

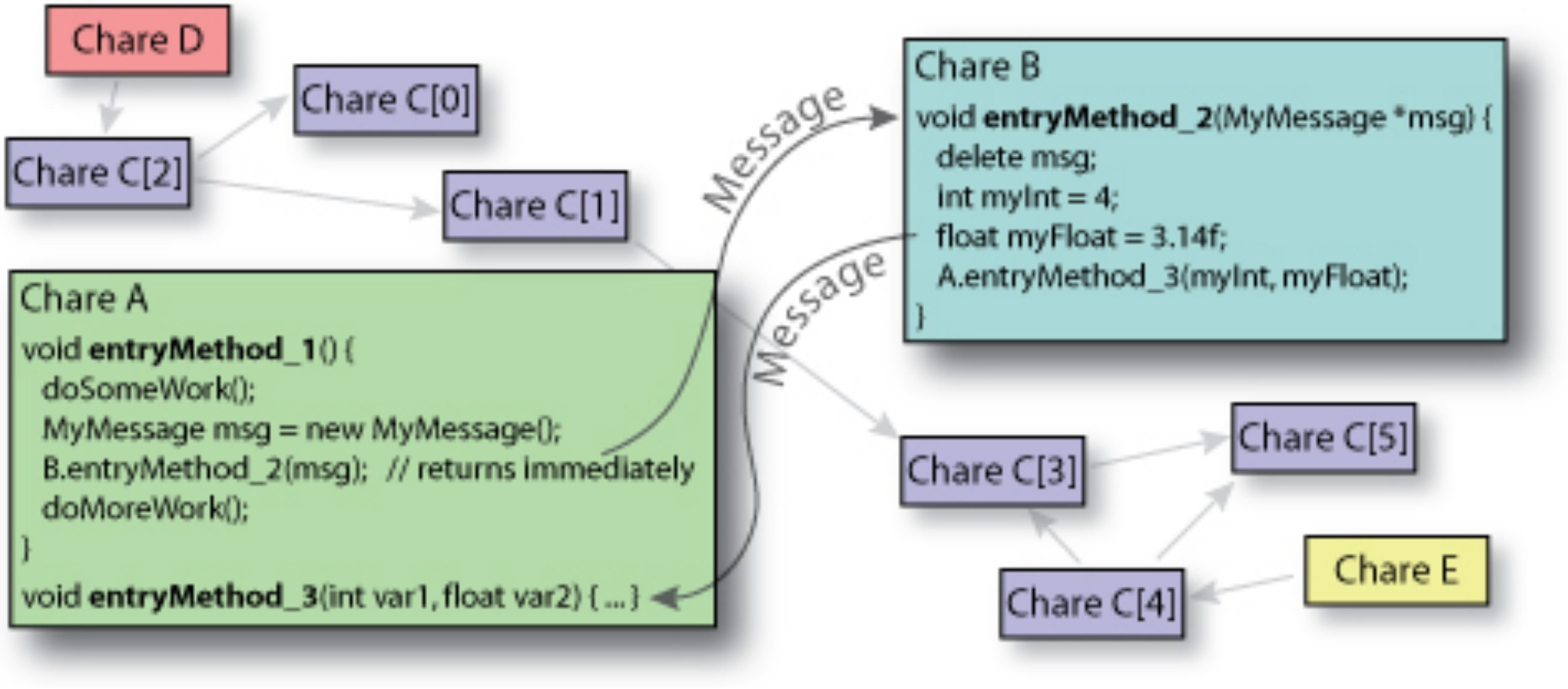
Charm++ is an alternative to MPI

- Both enable distributed memory parallelism (unlike OpenMP)
- MPI programs are decomposed into “ranks” (i.e., processes)
  - Number of ranks is determined when program is launched
  - Ranks call MPI send and receive functions to communicate
- Charm++ programs are *over*-decomposed into (arrays of) “chares”
  - Number of chares (in each array) is controlled by the programmer
  - Chares communicate via asynchronous “method” invocations

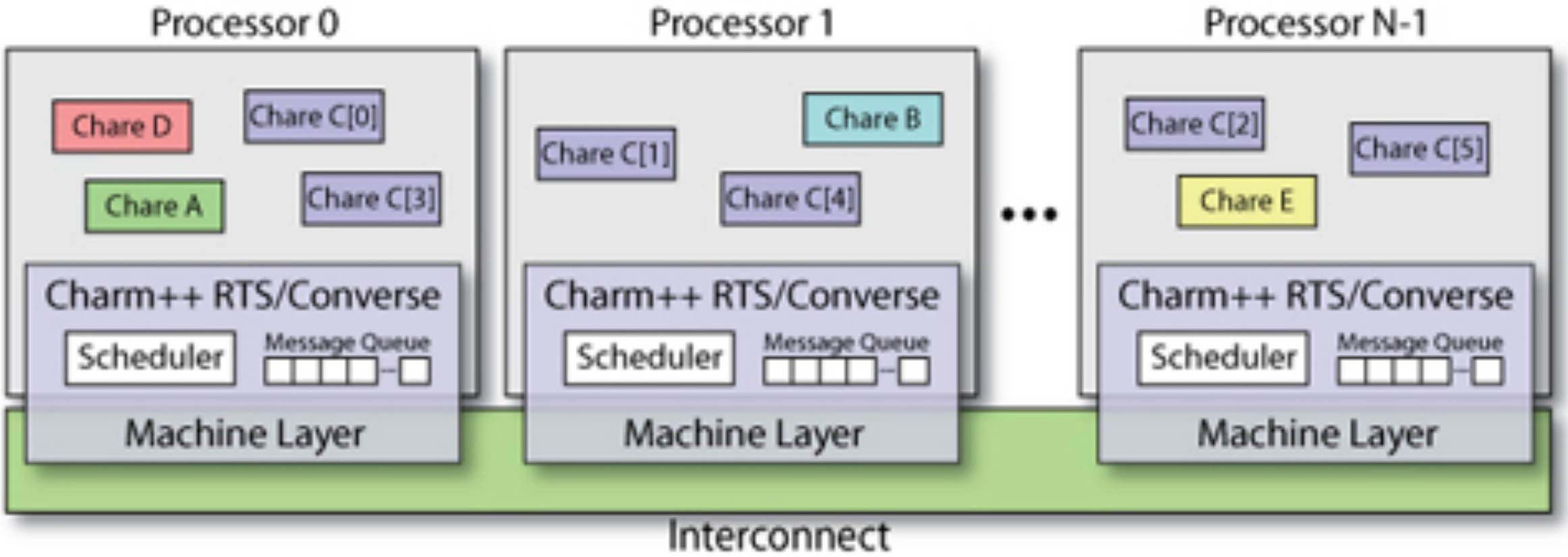
Local "proxy" objects enable remote method invocation



Messages both carry data and drive program execution



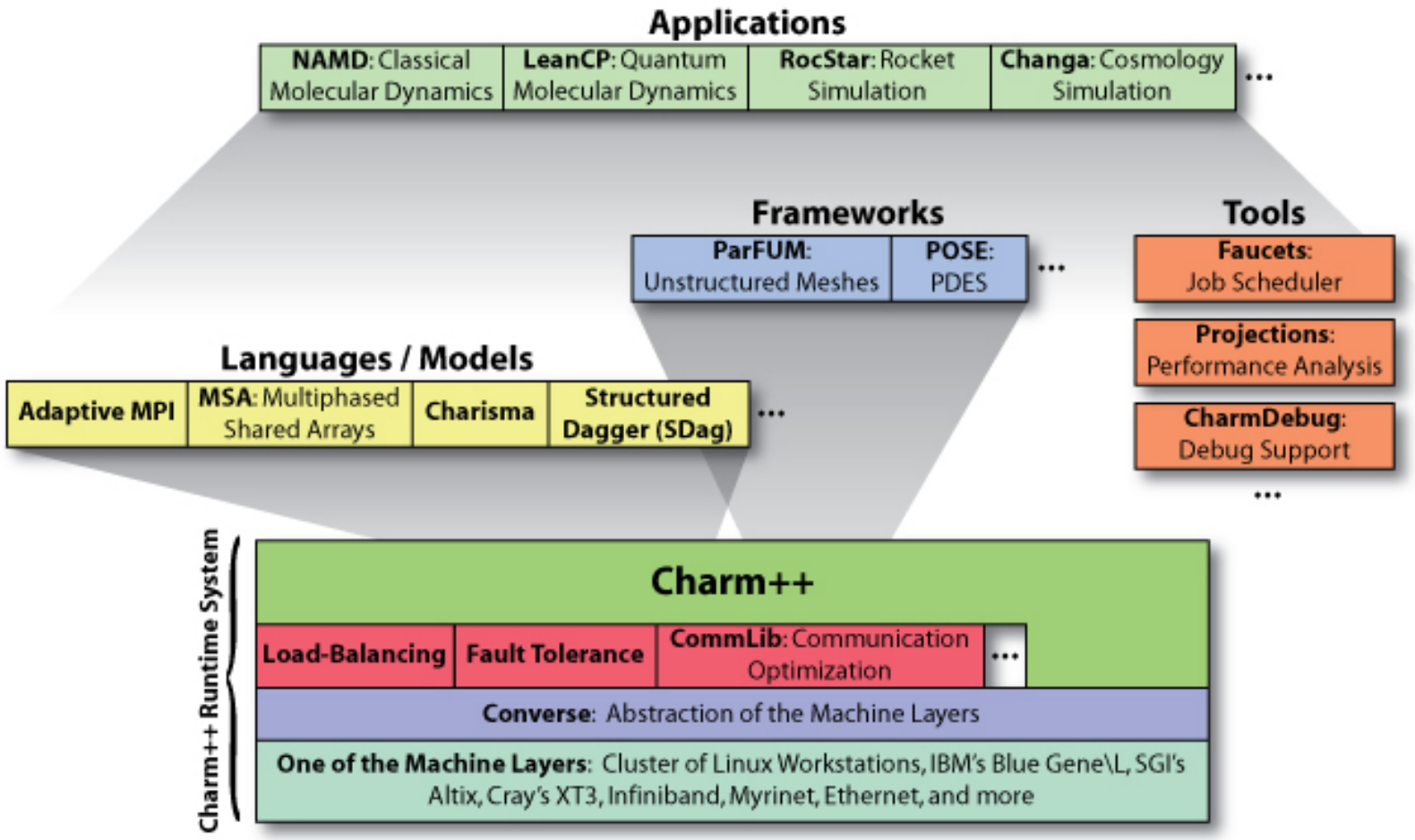
The Charm++ runtime transfers and queues messages



Charm++ integrates load-balancing capabilities

- Parallel performance is determined by the highest-load processor
- MPI programs require explicit load balancing
  - For uniform, persistent workload this may be trivial and static
  - Otherwise, programmer must alter decomposition to balance load
- The Charm++ runtime is *adaptive*
  - Runtime measures load and communication pattern of chares
  - Runtime migrates chares between processors to balance load

The Charm++ ecosystem includes tools and frameworks



Charm++ has been in production use in NAMD since 1999

Charm++ programs may be written in C++, F90, or Python

```
from charm4py import charm

def main(args):
    # create distributed 3D 10x10x10 array of cells
    grid = charm4py.Array(Cell, (10,10,10))
    done = charm.createFuture()
    # start simulation
    grid.work(done)
    done.get()  # wait until future receives value
    exit()

charm.start(main)
```

```
class Cell(charm4py.Chare):
    def __init__(self):
        self.data = ...  # initialize my region
        self.adjacent_cells = [self.thisProxy[idx] for idx in ...]

    @threaded
    def work(self, done_future):
        for self.iter in range(NUM_ITER):
            # compute can be compiled (Numba, Cython, etc.)
            compute(self.data)
            for nb in adjacent_cells:
                nb.recvGhost(self.iter, ghost_data)
            self.wait('self.msg_cnt == len(self.adjacent_cells)')
            self.msg_cnt = 0
        self.contribute(None, None, done_future)

    @when('self.iter == iter')
    def recvGhost(self, iter, nb_data):
        # process nb_data ...
        self.msg_cnt += 1
```

Charm++ integrates with other programming models

- Adaptive MPI – write in over-decomposed MPI, run in Charm++
- MPI interoperability – call Charm++ modules from MPI program

- SMP mode – single-threaded chares, multiple threads per process
- OpenMP – single-threaded Charm++ runtime, chares call OpenMP

- CUDA etc. – chares offload work to GPU and wait for result
- GPU Manager – runtime triggers chares when results are available
  - Overlap increases GPU utilization and reduces CPU bottlenecks

Further information on Charm++ is readily available

- charmplusplus.org – General overview, tutorials, mini-apps
- charm.readthedocs.io – Charm++ and frameworks documentation
- charm4py.readthedocs.io – Python documentation and tutorial
- charm.cs.Illinois.edu – Parallel Programming Laboratory
  - People, publications, annual Charm++ workshop recordings, etc.
- github.com/UIUC-PPL – Charm++ and Charm4py repositories
- hpccharm.com – Commercial licensing, support, consulting, etc.
  - This is not an endorsement or recommendation of Charmworks, Inc.