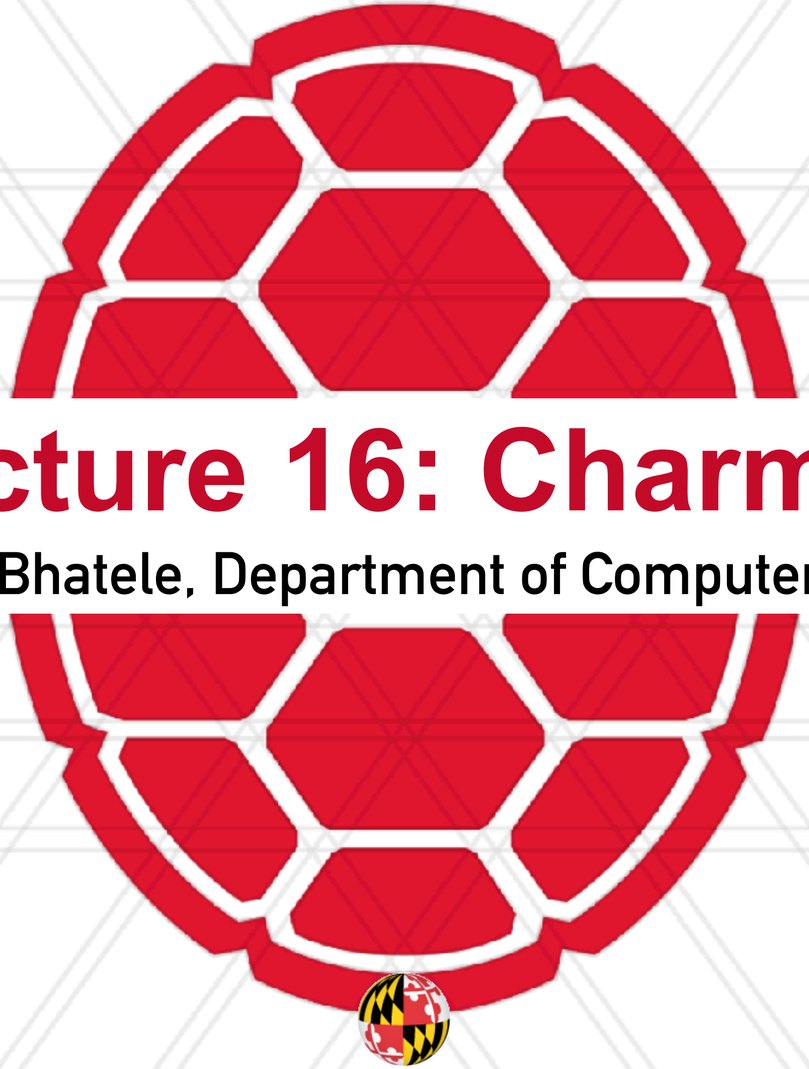
Introduction to Parallel Computing (CMSC498X / CMSC818X)



Lecture 16: Charm++

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Assignment 3 will be posted on Oct 26 at midnight AoE
- Midterm will be posted on Oct 26 midnight AoE and due on Oct 27 midnight AoE

Hello world: .ci file

```
mainmodule hello {

  readonly CProxy_MyMain myMainProxy;
  readonly int numChares;

  mainchare MyMain {
    entry MyMain(CkArgMsg *msg);
    entry void done(void);
  };

  array [1D] Hello {
    entry Hello(void);
    entry void sayHi(int);
  };

};
```

Hello world: MyMain class

```
/*readonly*/ CProxy_MyMain myMainProxy;
/*readonly*/ int numChares;

class MyMain: public CBase_MyMain {
  public:
    MyMain(CkArgMsg* msg) {
      numChares = atoi(msg->argv[1]); // number of elements

      myMainProxy = thisProxy;
      CProxy_Hello helArrProxy = CProxy_Hello::ckNew(numChares);

      helArrProxy[0].sayHi(20);
    }

    void done(void) {
      ckout << "All done" << endl;
      CkExit();
    }
};
```

Hello world: Hello class

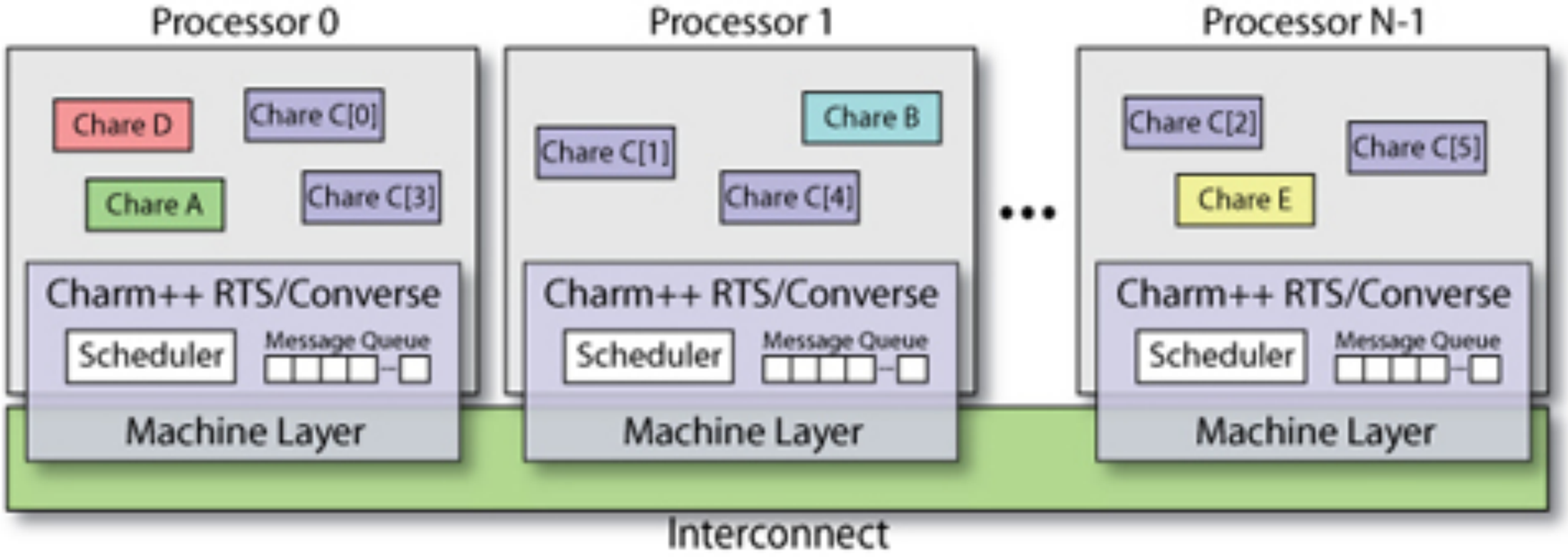
```
#include “hello.decl.h”
extern /*readonly*/ CProxy_MyMain myMainProxy;

class Hello: public CBase_Hello {
  public:
    Hello(void) { }

    void sayHi(int num) {
      ckout << “Chare ” << thisIndex << “says Hi!” << num << endl;

      if(thisIndex < numChares-1)
        thisProxy[thisIndex+1].sayHi(num+1);
      else
        myMainProxy.done();
    }
};

#include “hello.def.h”
```

Proxy class

- Runtime needs to pack/unpack data and also figure out where the chare is
- Proxy class generated for each chare class
  - Proxy objects know where the real object is
  - Methods invoked on these proxy objects lead to messages being sent to the destination processor

Charm scheduler and message queue

- An object is scheduled by the runtime scheduler only when a message for it is received
- Facilitates adaptive overlap of computation and communication



Broadcast, barrier, and reduction

- Entry method called on a chare proxy without subscript is essentially a broadcast:

```
chareProxy.entryMethod()
```

- Barrier: reduction without arguments:

```
contribute();
```

- Reduction with arguments:

```
void contribute(int bytes, const void *data, CkReduction::reducerType type);
```

Callback for reduction

- Where does the output of the reduction go?
- Use a callback object known as a reduction client

```
CkCallback* cb = new CkCallback(CkIndex_myType::myReductionFunction(NULL), thisProxy);
contribute(bytes, data, reducerType, cb);
```

- Use the reduction data in the callback:

```
void myType::myReductionFunction(CkReductionMsg *msg) {
  int size = msg->getSize() / sizeof(type);
  type *output = (type *) msg->getData();

  ...
}
```

UNIVERSITY OF
MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu