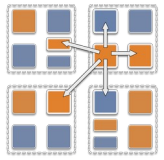
Charm Interface: Modules

- Charm++ programs are organized as a collection of modules
- Each module has one or more chares
- The module that contains the mainchare is declared as the mainmodule
- Each module, when compiled, generates two files: MyModule.decl.h and MyModule.def.h

.ci file

```
[main]module MyModule {
   //... chare definitions ...
};
```



PPL UIUC

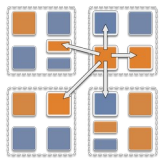
Charm Interface: Chares

- Chares are parallel objects that are managed by the RTS
- Each chare has a set of entry methods, which are asynchronous methods that may be invoked remotely
- The following code, when compiled, generates a C++ class `CBase_MyChare` that encapsulates the RTS object
- This generated class is extended and implemented in the .cpp file
- .ci file

```
[main]chare MyChare {
   //... entry method definitions ...
};
```

- .cpp file

```
class MyChare : public CBase_MyChare {
   //... entry method implementations ...
};
```



PPL UIUC

Charm Interface: Entry Methods

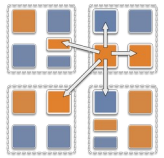
- Entry methods are C++ methods that can be remotely and asynchronously invoked by another chare
- .ci file

```
entry MyChare(); /* constructor entry method */
entry void foo();
entry void bar(int param);
```

- .cpp file

```
MyChare::MyChare() { /*... constructor code ...*/ }
MyChare::foo() { /*... code to execute ...*/ }
MyChare::bar(int param) { /*... code to execute ...*/ }
```

This necessitates changing both (or all: .ci, .h, .cpp) files if you add/remove/change a parameter to an entry method, which is a common pitfall



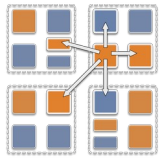
PPL UIUC

Charm Interface: mainchare

- Execution begins with the mainchare's constructor
- The mainchare's constructor takes a pointer to system-defined class `CkArgMsg`
- `CkArgMsg` contains `argv` and `argc`
- The mainchare will typically creates some additional chares

There can be more than one chare.. They all will start, in unspecified order, on one of the PEs.

But it is customary to have only one main chare in a program.



PPL UIUC

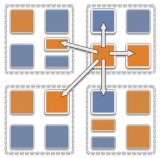
Creating a Chare

- A `chare` declared as `chare MyChare {...};` can be instantiated by the following call:

```
CProxy_MyChare::ckNew(…constructor arguments…);
```

- To communicate with this class in the future, a *proxy* to it must be retained

```
CProxy_MyChare proxy = CProxy_MyChare::ckNew(arg1);
```



PPL UIUC

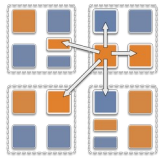
Chare Proxies

- A chare's own proxy can be obtained through a special variable `thisProxy`
- Chare proxies can also be passed so chares can learn about others
- In this snippet, `MyChare` learns about a chare instance `main`, and then invokes a method on it:
- .ci file

```
entry void foobar(CProxy_Main main);
```

- .cpp file

```
MyChare::foobar(CProxy_Main main) {
   main.foo();
}
```



PPL UIUC

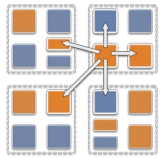
Hello World with Chares

hello.ci

```
mainmodule hello {
  mainchare Main {
    entry Main(CkArgMsg *m);
  };
  chare Singleton {
    entry Singleton();
  };
};
```

hello.cpp

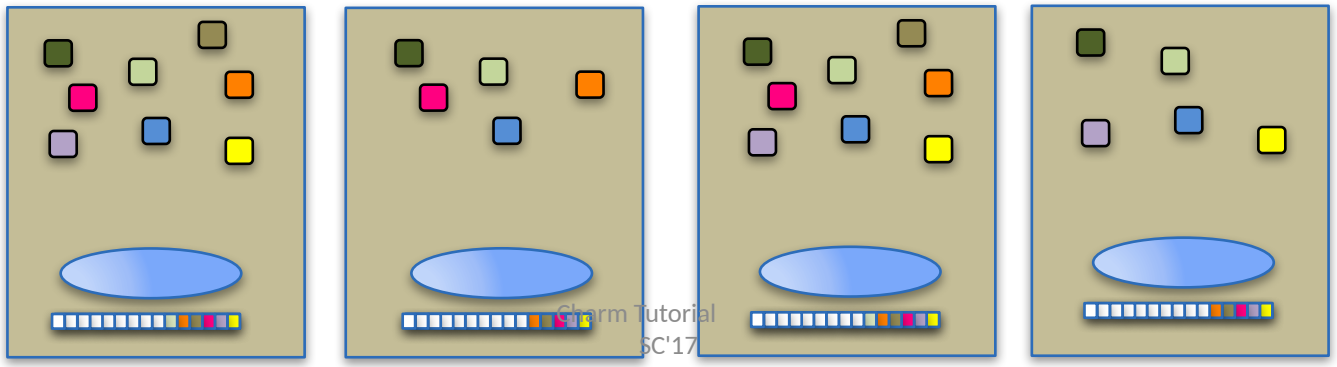
```
#include <stdio.h>
#include "hello.decl.h"

class Main : public CBase_Main {
  public: Main(CkArgMsg* m) {
    CProxy_Singleton::ckNew();
  };
};
class Singleton : public CBase_Singleton
{
  public: Singleton() {
    ckout << "Hello World!" << endl;
    CkExit();
  };
};
#include "hello.def.h"
```

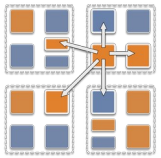


PPL UIUC

Charm Termination

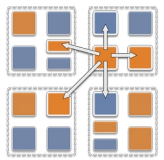
- There is a special system call `CkExit()` that terminates the parallel execution on all processors (but it is called on one processor) and performs the requisite cleanup
- The traditional `exit()` is insufficient because it only terminates one process, not the entire parallel job (and will cause a hang)
- `CkExit()` should be called when you can safely terminate the application (you may want to synchronize before calling this)





Chare Creation Example: .ci file

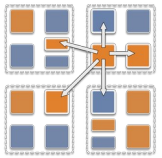
```
mainmodule MyModule {
    mainchare Main {
        entry Main(CkArgMsg *m);
    };

    chare Simple {
        entry Simple(int x, double y);
    };
};
```



PPL UIUC

Chare Creation Example: .cpp file

```
#include “MyModule.decl.h”

class Main : public CBase_Main {
public: Main(CkArgMsg* m) {
    ckout << “Hello World!” << endl;
    CProxy_Simple::ckNew(12, 3.1415);
  } };

class Simple : public CBase_Simple {
public: Simple(int x, double y) {
    ckout << “Radius:” << x << “, Area:” << y*x*x << endl;
    CkExit();
  } };

#include “MyModule.def.h”
```



PPL UIUC